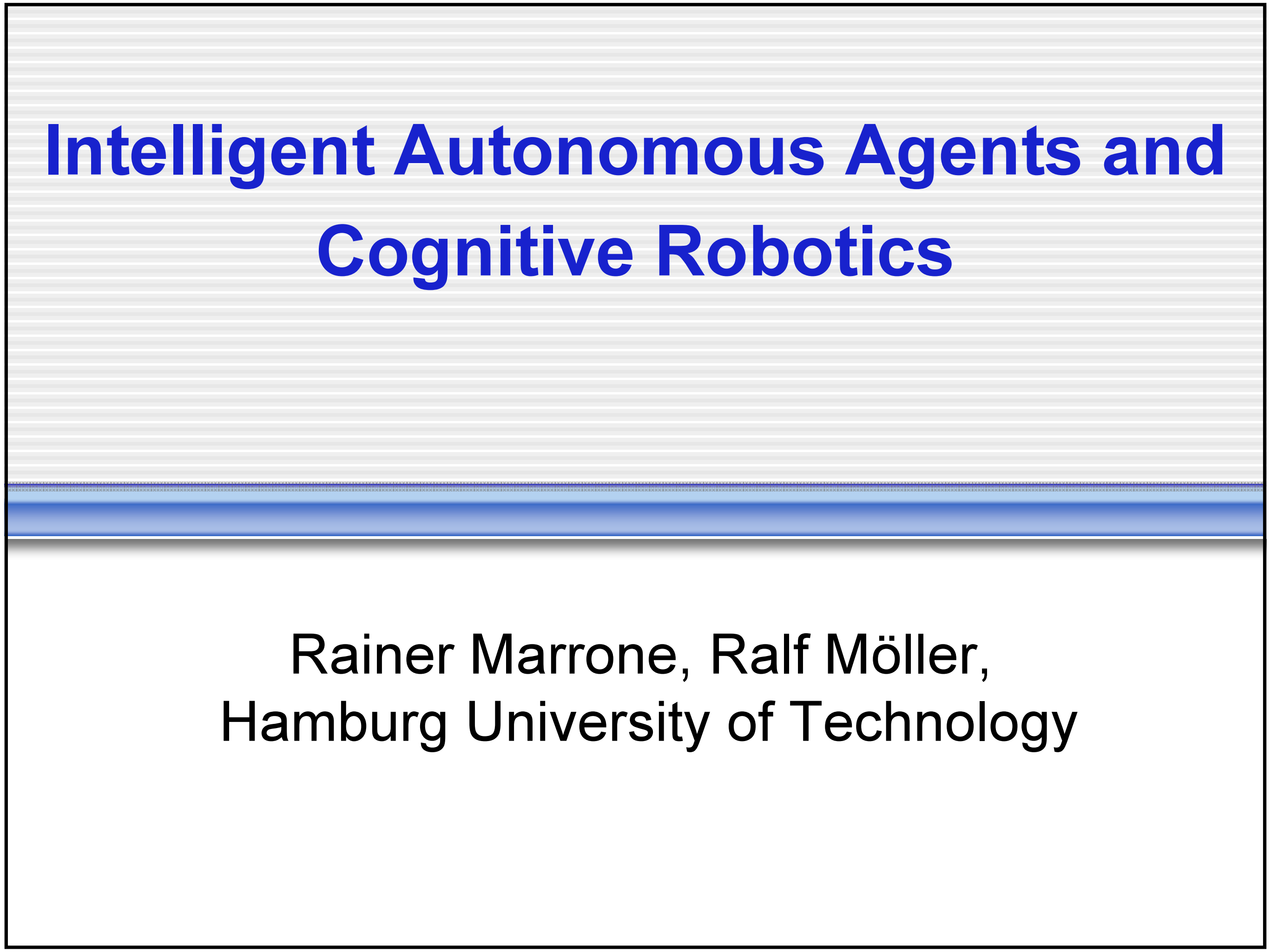
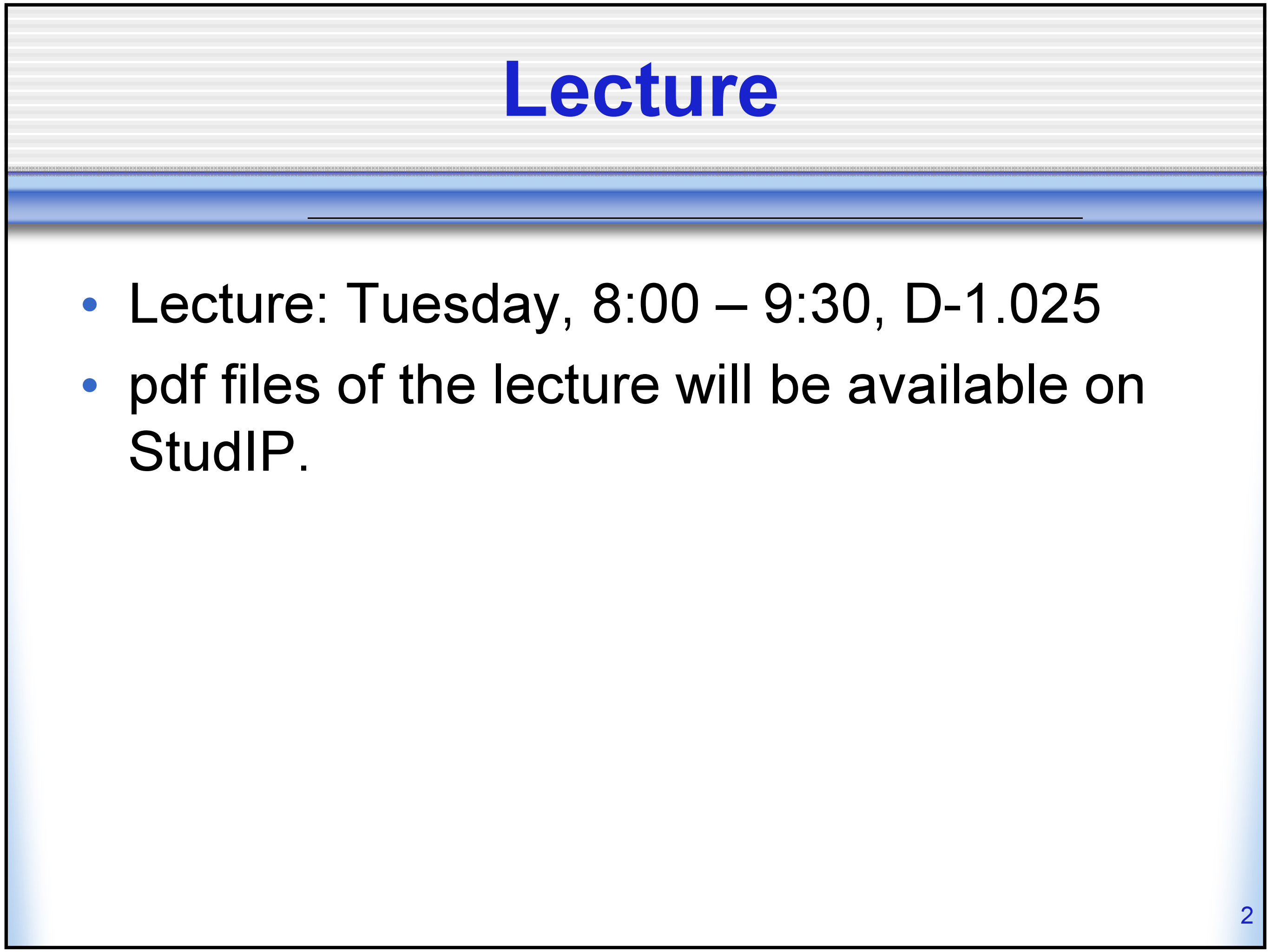


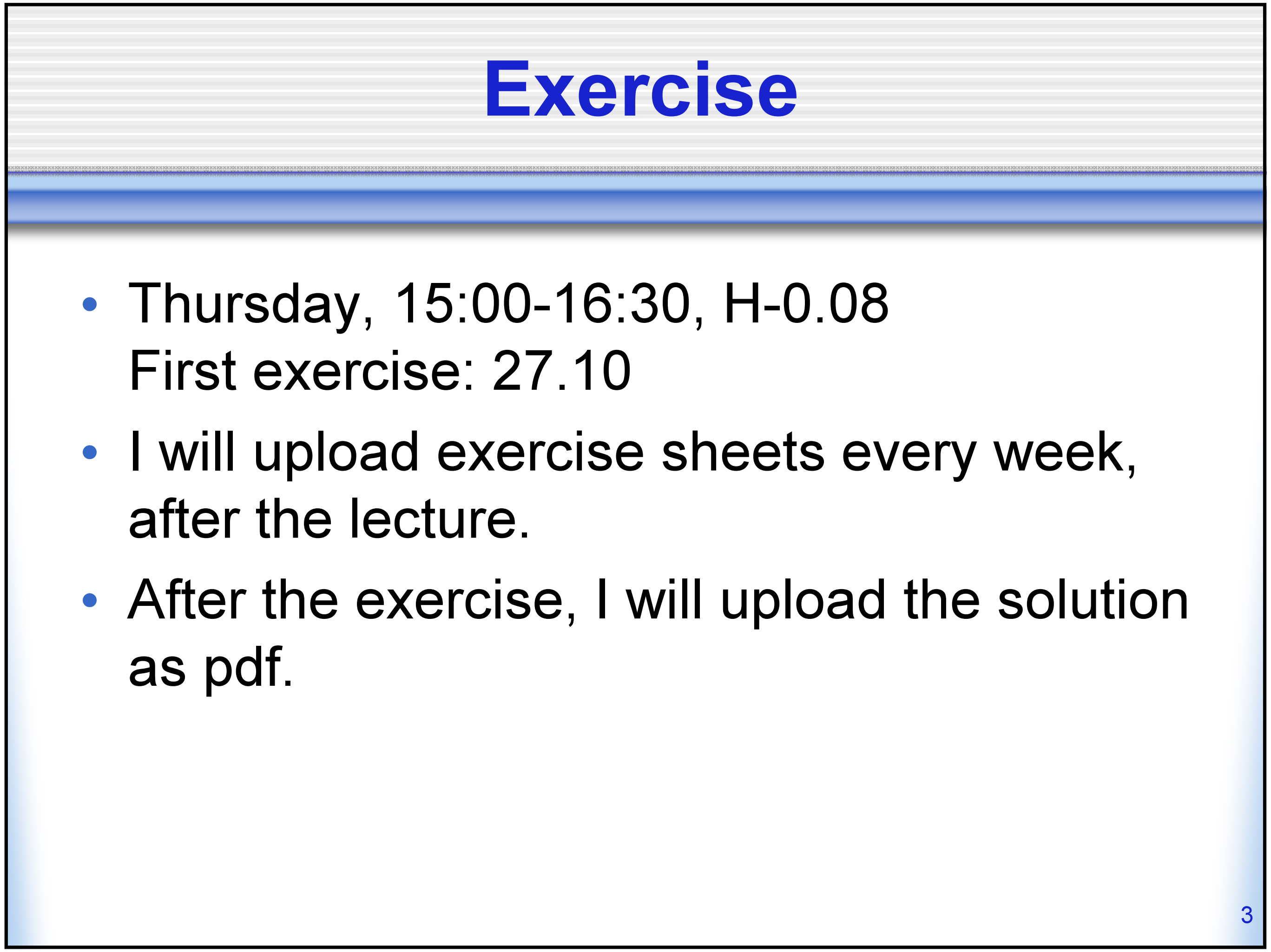
# Intelligent Autonomous Agents and Cognitive Robotics

Rainer Marrone, Ralf Möller,
Hamburg University of Technology



# Lecture

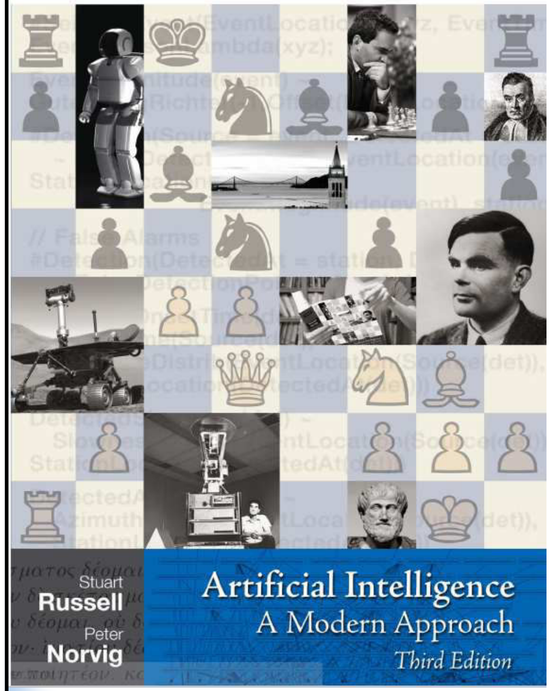
- Lecture: Tuesday, 8:00 – 9:30, D-1.025
- pdf files of the lecture will be available on StudIP.



# Exercise

- Thursday, 15:00-16:30, H-0.08
  First exercise: 27.10
- I will upload exercise sheets every week, after the lecture.
- After the exercise, I will upload the solution as pdf.

# Literature

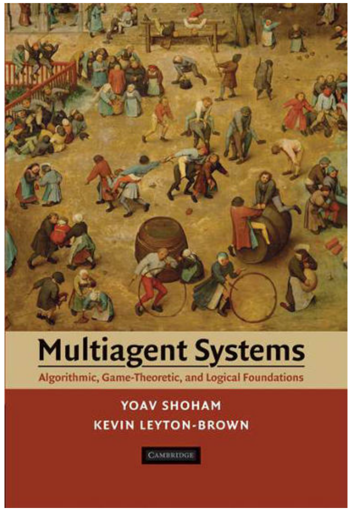

Stuart Russell
Peter Norvig
Artificial Intelligence
A Modern Approach
Third Edition

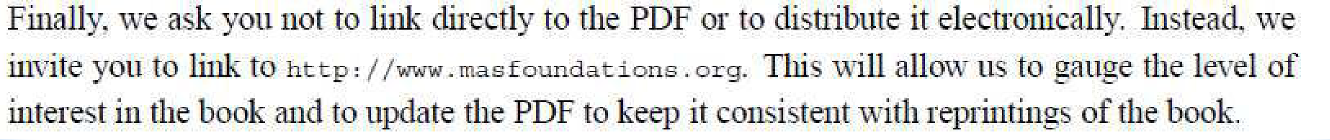
Chapters 2-5, 13 - 17

http://aima.cs.berkeley.edu

- with code repository
- further readings

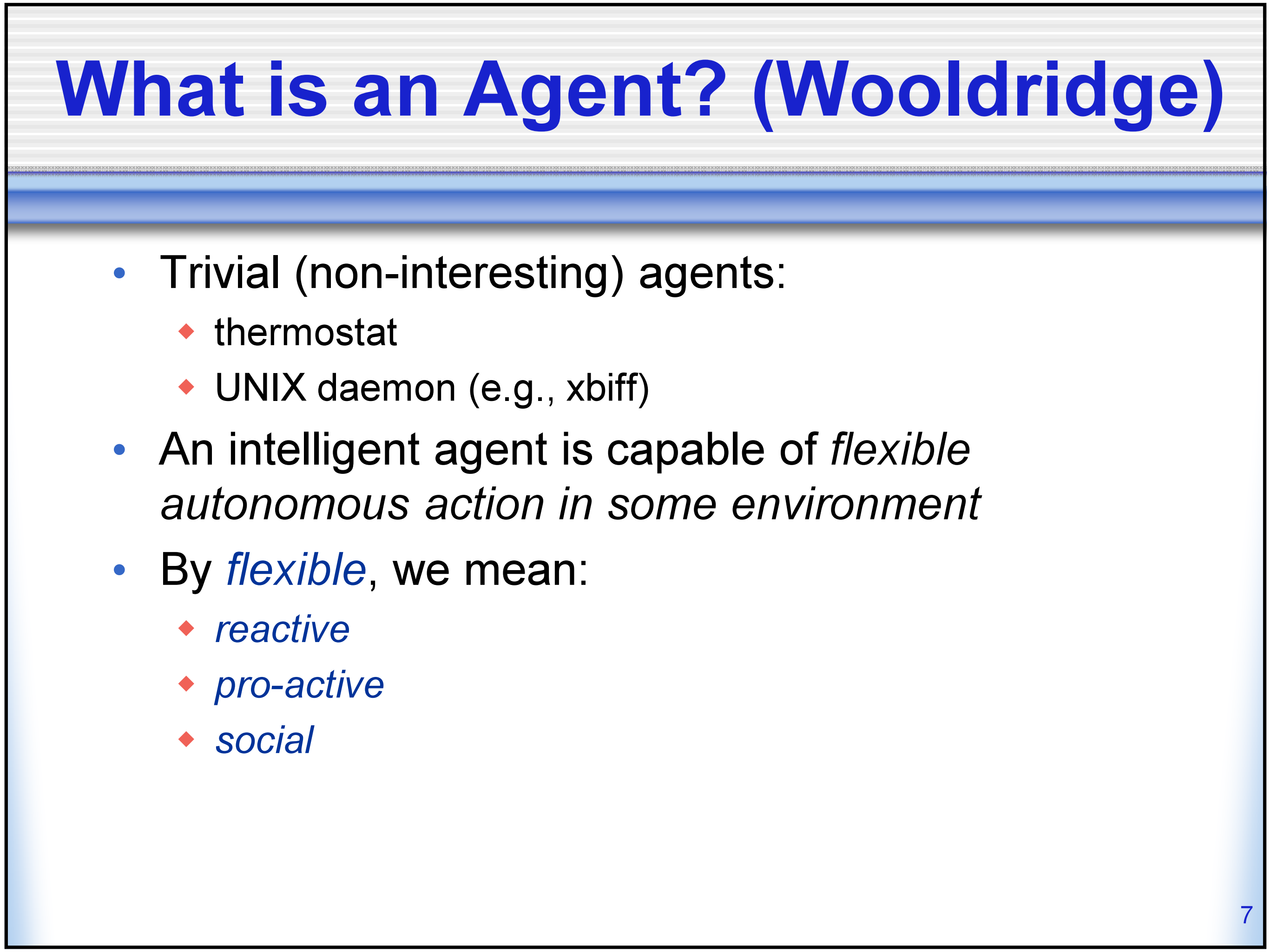
# Literature




Finally, we ask you not to link directly to the PDF or to distribute it electronically. Instead, we invite you to link to `http://www.masfoundations.org`. This will allow us to gauge the level of interest in the book and to update the PDF to keep it consistent with reprintings of the book.

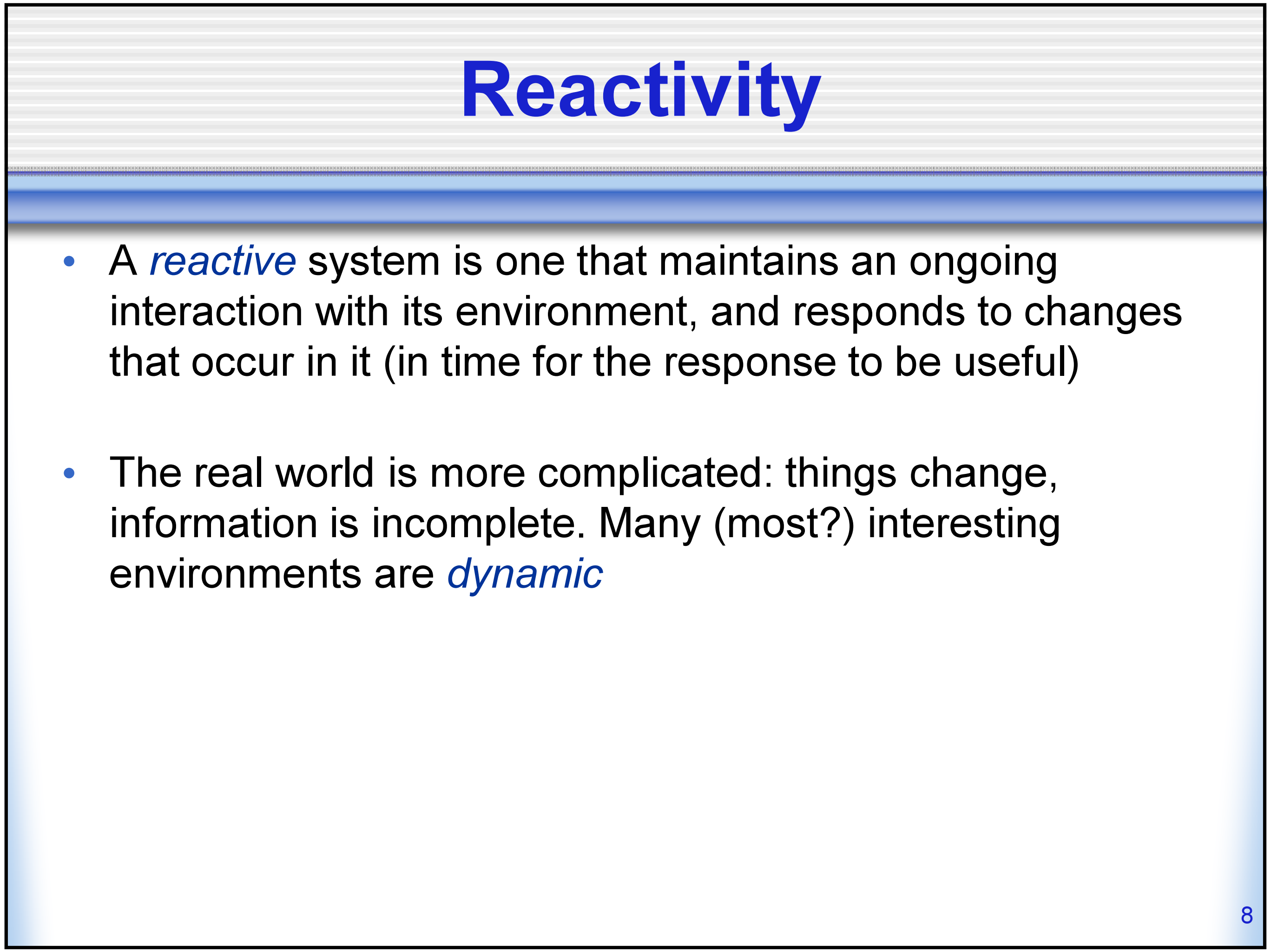
# Main Topics

- Solving Problems by Searching
- Adversarial Agents
- Constraint Satisfaction Problems
- Bayesian Networks
- Probabilistic Reasoning Over Time
- Decision Making
- Game Theory
- Mechanism Design



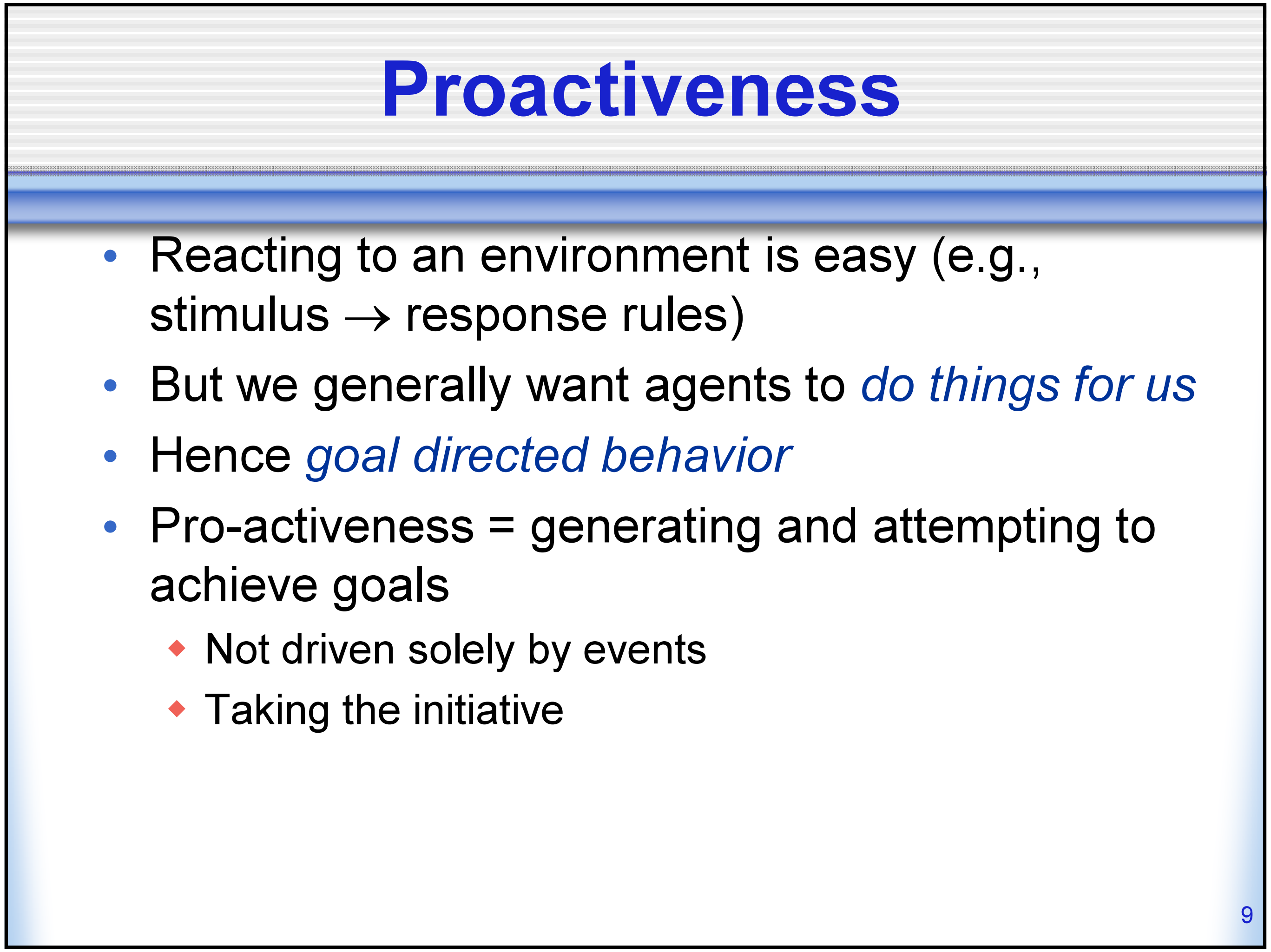
# What is an Agent? (Wooldridge)

- Trivial (non-interesting) agents:
  - thermostat
  - UNIX daemon (e.g., xbiff)
- An intelligent agent is capable of *flexible autonomous action in some environment*
- By *flexible*, we mean:
  - *reactive*
  - *pro-active*
  - *social*



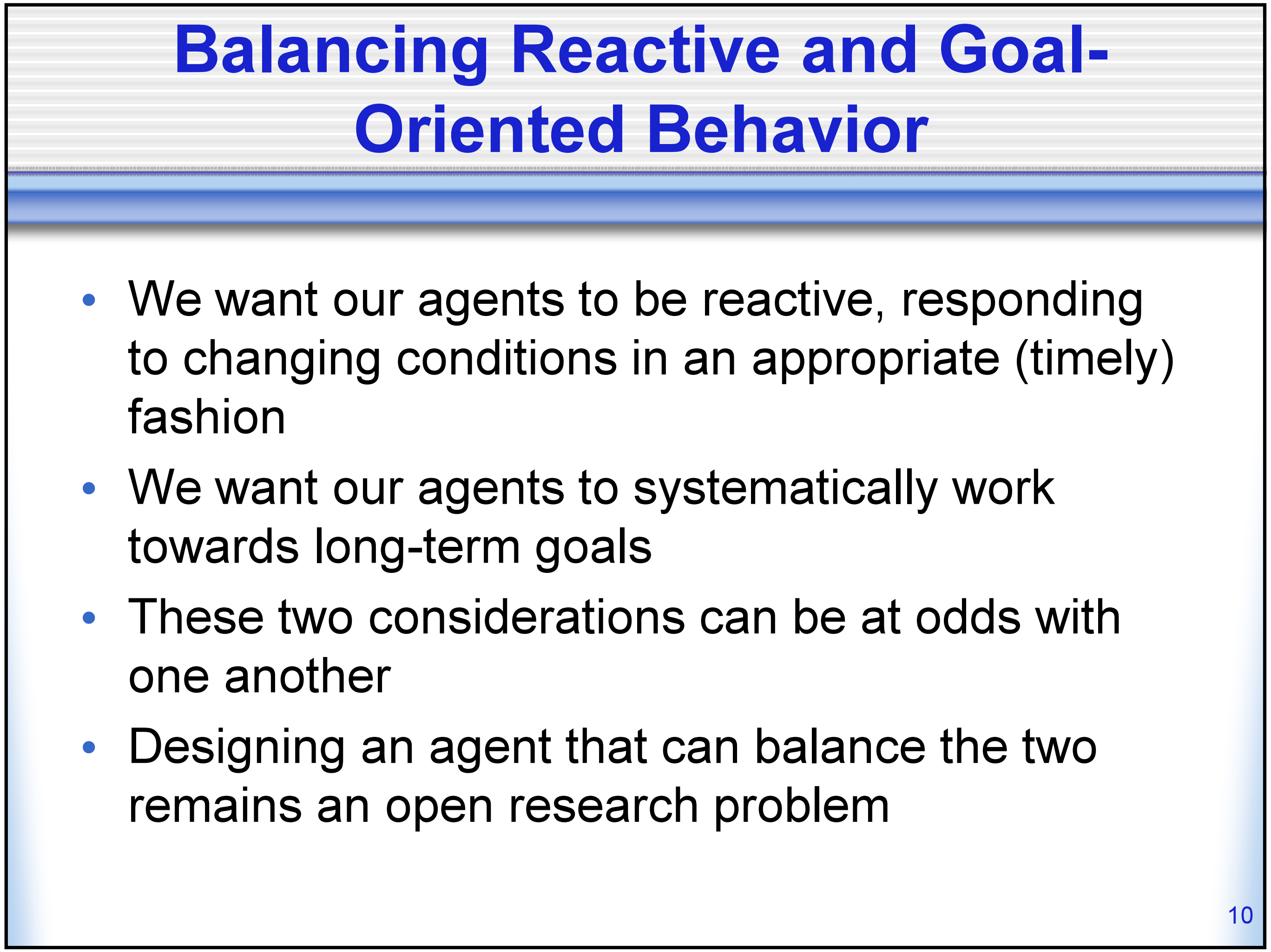
# Reactivity

- A *reactive* system is one that maintains an ongoing interaction with its environment, and responds to changes that occur in it (in time for the response to be useful)
- The real world is more complicated: things change, information is incomplete. Many (most?) interesting environments are *dynamic*



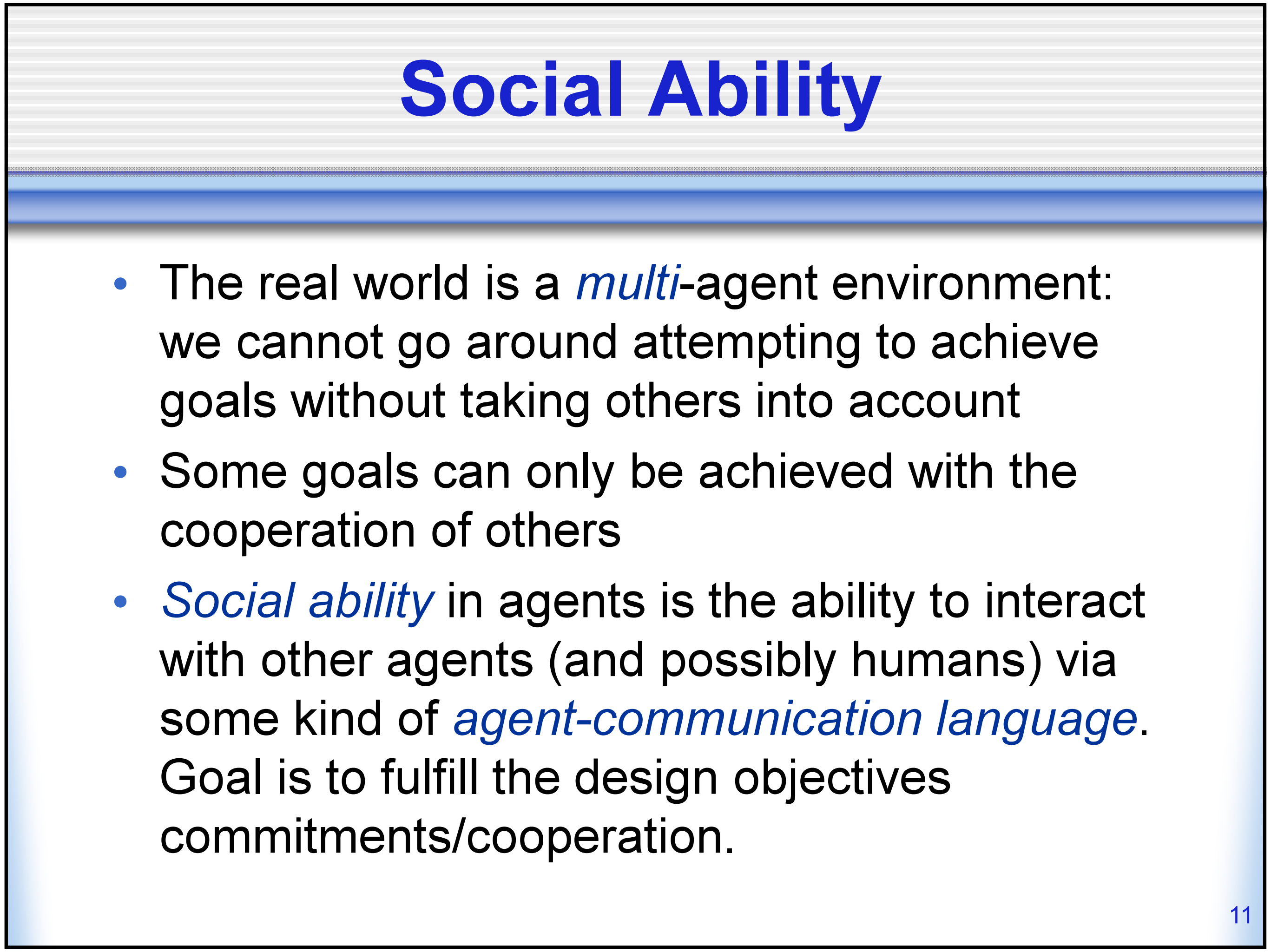
# Proactiveness

- Reacting to an environment is easy (e.g., stimulus → response rules)
- But we generally want agents to *do things for us*
- Hence *goal directed behavior*
- Pro-activeness = generating and attempting to achieve goals
  - Not driven solely by events
  - Taking the initiative



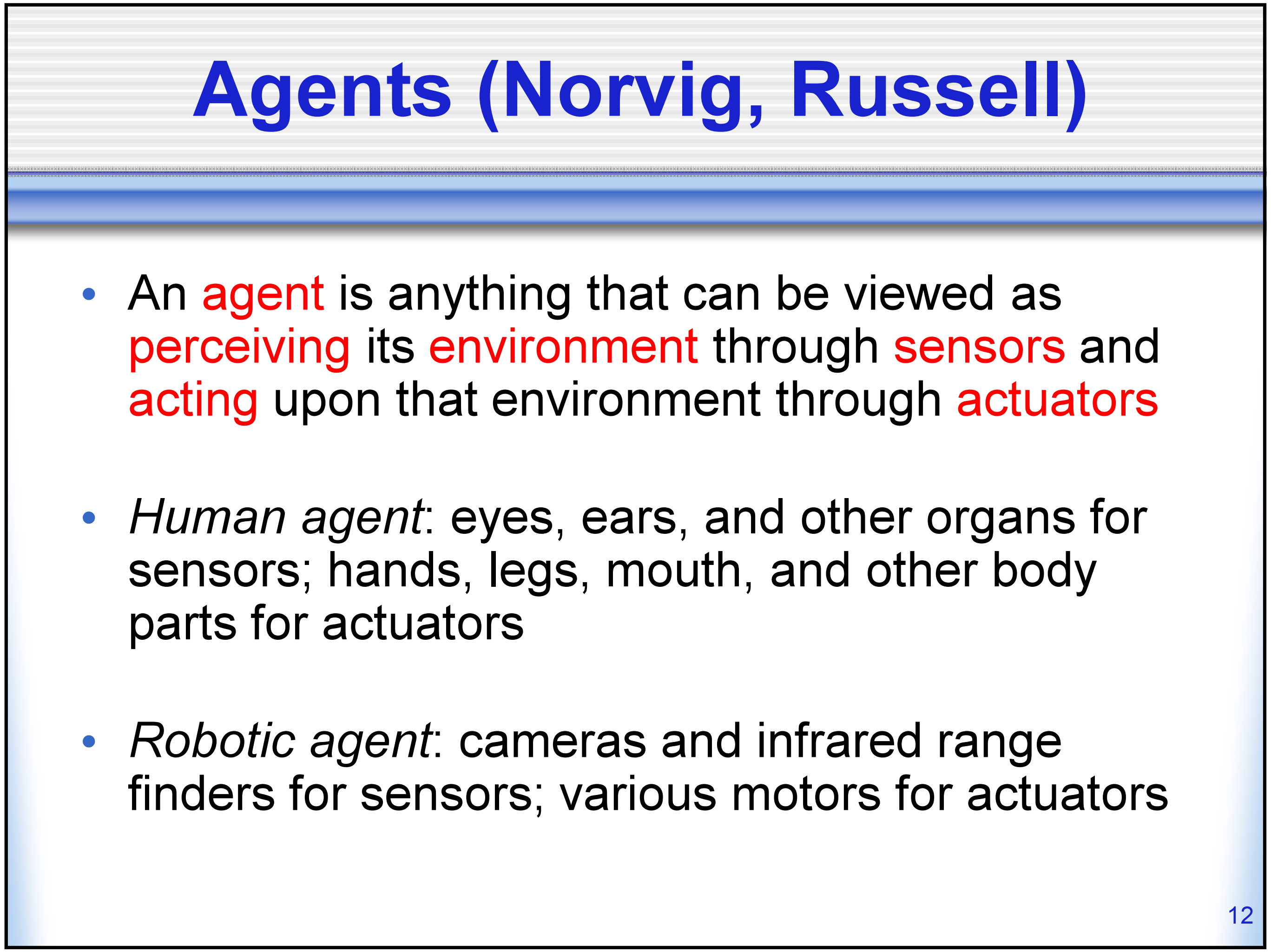
# Balancing Reactive and Goal-Oriented Behavior

- We want our agents to be reactive, responding to changing conditions in an appropriate (timely) fashion
- We want our agents to systematically work towards long-term goals
- These two considerations can be at odds with one another
- Designing an agent that can balance the two remains an open research problem



# Social Ability

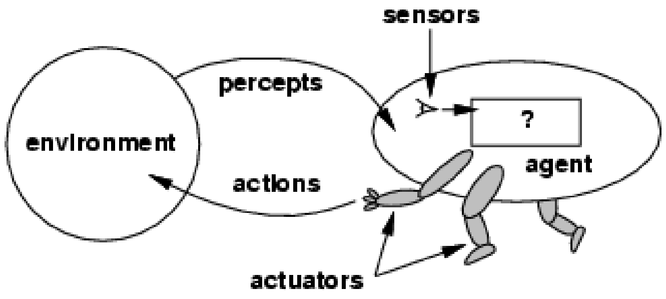
- The real world is a *multi*-agent environment: we cannot go around attempting to achieve goals without taking others into account
- Some goals can only be achieved with the cooperation of others
- *Social ability* in agents is the ability to interact with other agents (and possibly humans) via some kind of *agent-communication language*. Goal is to fulfill the design objectives commitments/cooperation.



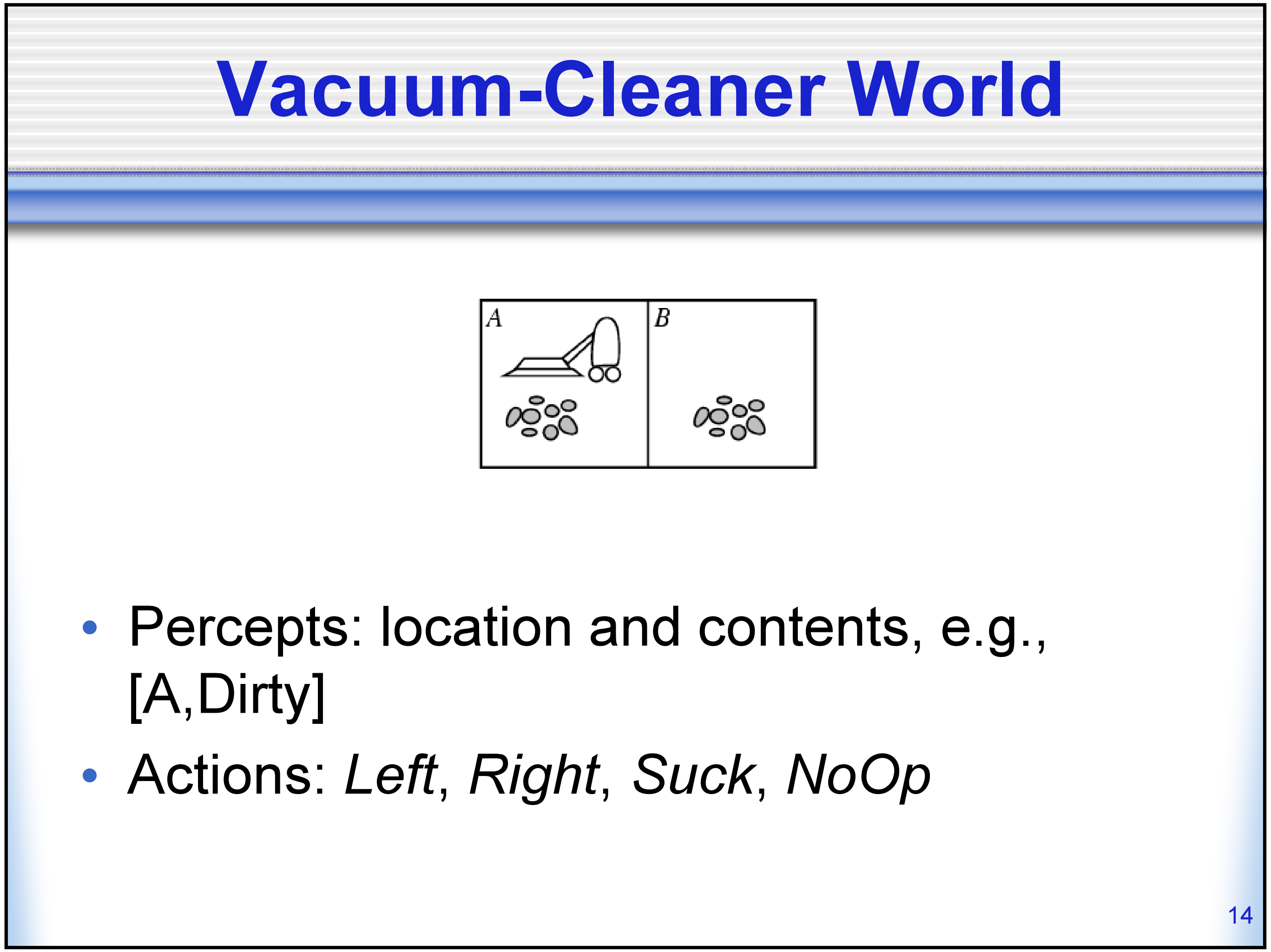
# Agents (Norvig, Russell)

- An agent is anything that can be viewed as perceiving its environment through sensors and acting upon that environment through actuators
- *Human agent*: eyes, ears, and other organs for sensors; hands, legs, mouth, and other body parts for actuators
- *Robotic agent*: cameras and infrared range finders for sensors; various motors for actuators

# Agents and environments

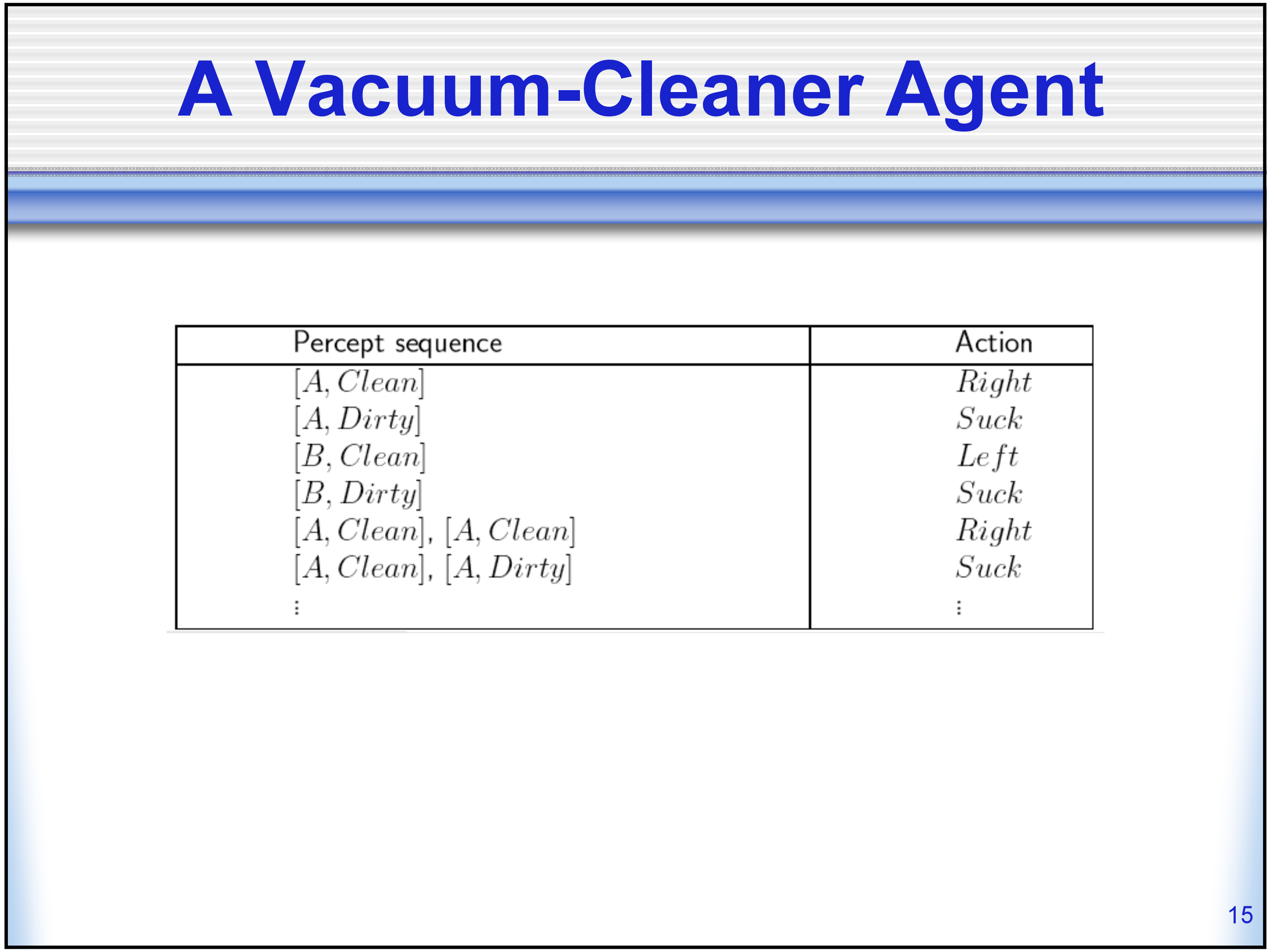


- The agent function maps from percept histories to actions .
  f : P*→ A
- The agent program runs on the physical architecture to produce *f*
- agent = architecture + program



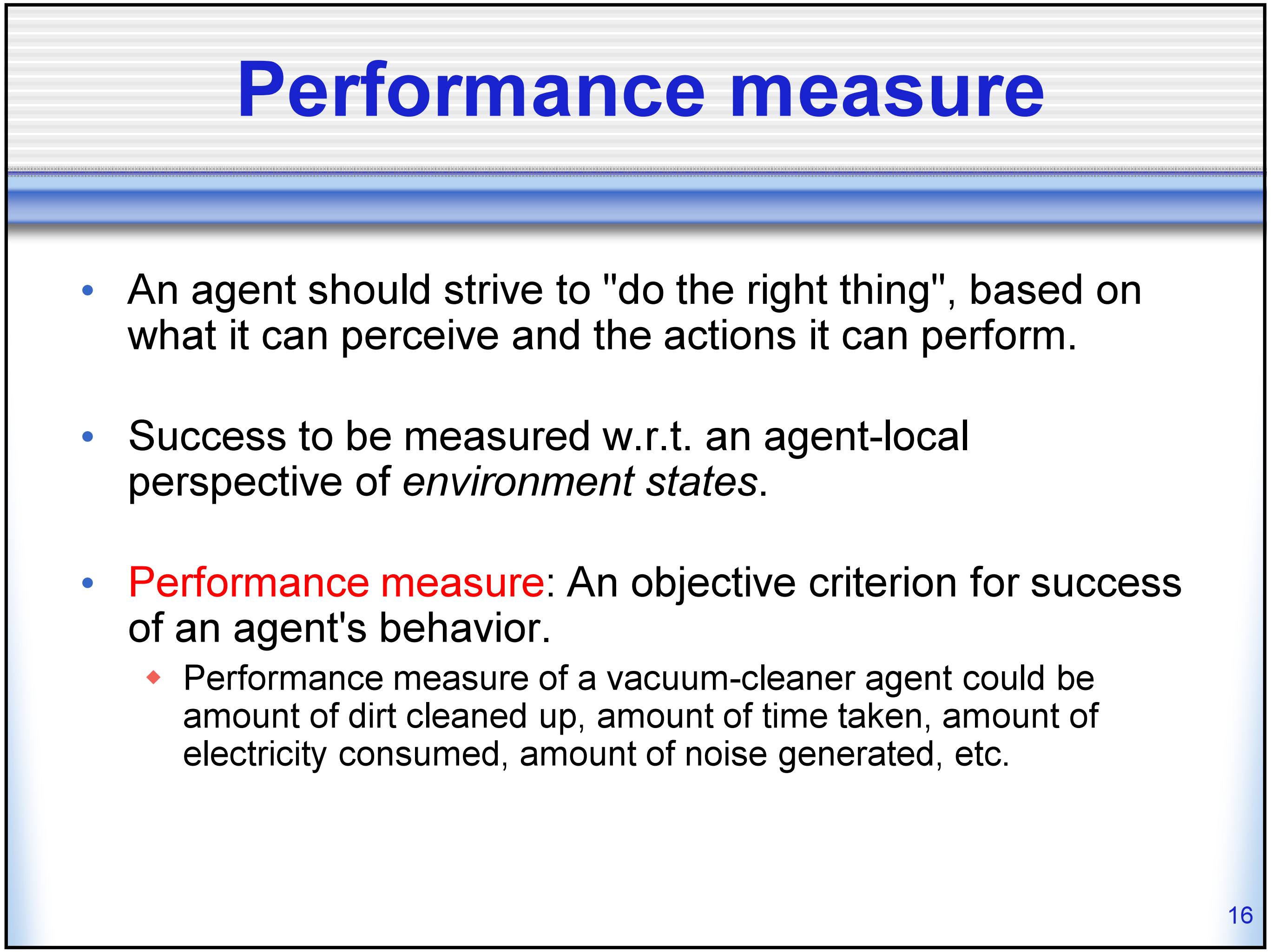
## Vacuum-Cleaner World

- Percepts: location and contents, e.g., [A,Dirty]
- Actions: *Left*, *Right*, *Suck*, *NoOp*



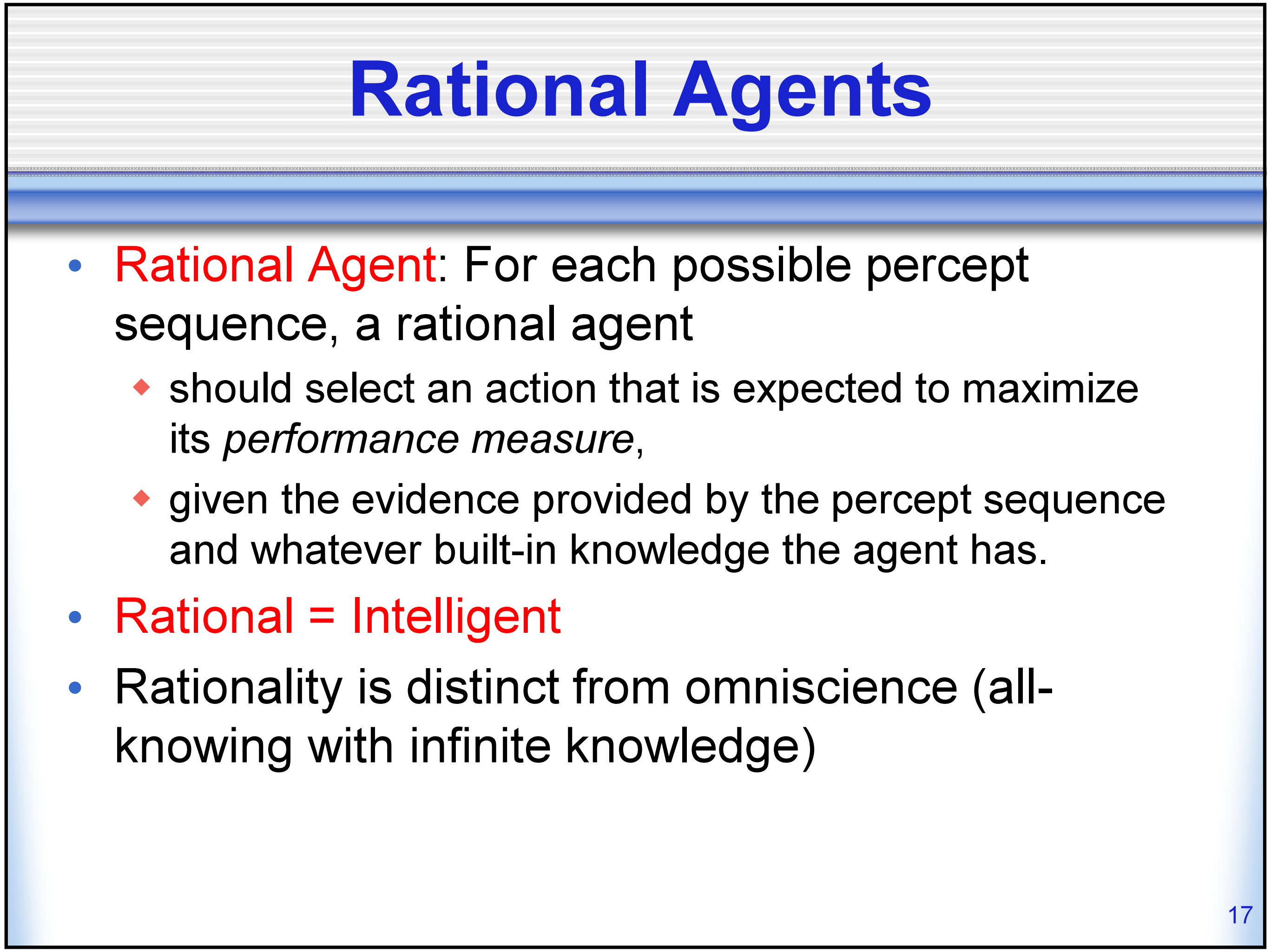
# A Vacuum-Cleaner Agent

| Percept sequence | Action |
|---|---|
| $[A, Clean]$ | $Right$ |
| $[A, Dirty]$ | $Suck$ |
| $[B, Clean]$ | $Left$ |
| $[B, Dirty]$ | $Suck$ |
| $[A, Clean], [A, Clean]$ | $Right$ |
| $[A, Clean], [A, Dirty]$ | $Suck$ |
| ⋮ | ⋮ |



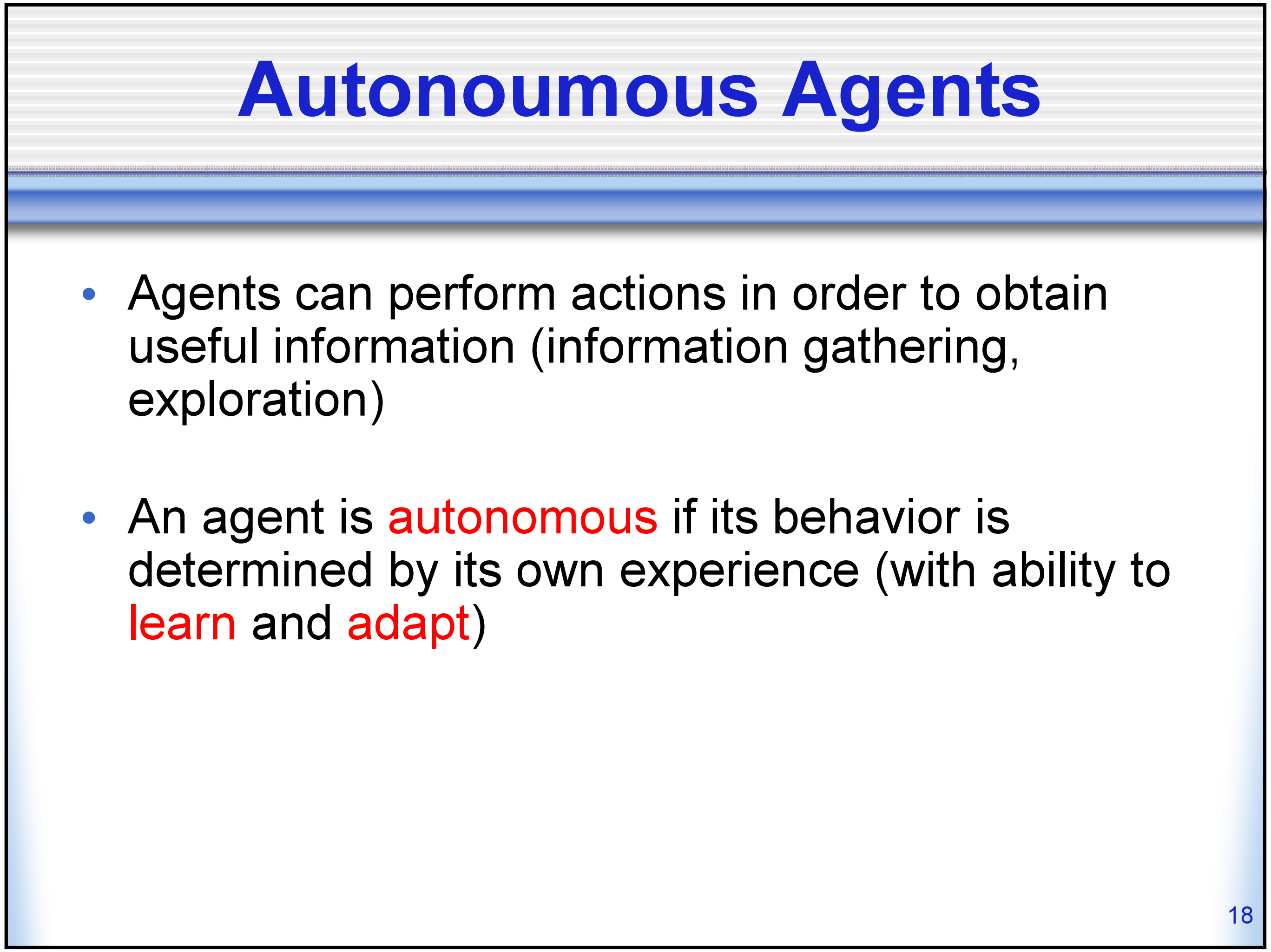
# Performance measure

- An agent should strive to "do the right thing", based on what it can perceive and the actions it can perform.
- Success to be measured w.r.t. an agent-local perspective of *environment states*.
- Performance measure: An objective criterion for success of an agent's behavior.
    - Performance measure of a vacuum-cleaner agent could be amount of dirt cleaned up, amount of time taken, amount of electricity consumed, amount of noise generated, etc.



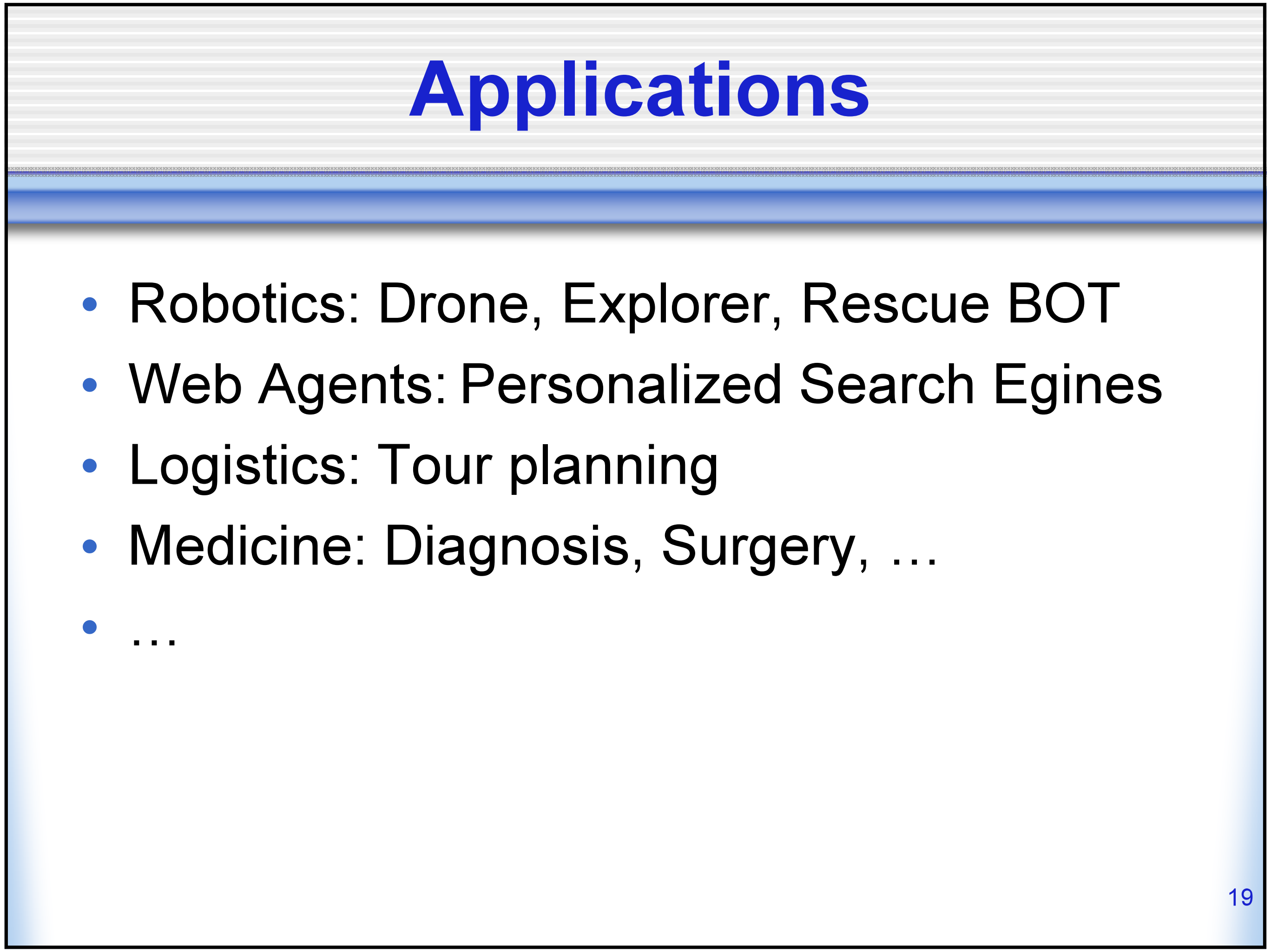
# Rational Agents

- Rational Agent: For each possible percept sequence, a rational agent
  - should select an action that is expected to maximize its *performance measure*,
  - given the evidence provided by the percept sequence and whatever built-in knowledge the agent has.
- Rational = Intelligent
- Rationality is distinct from omniscience (all-knowing with infinite knowledge)



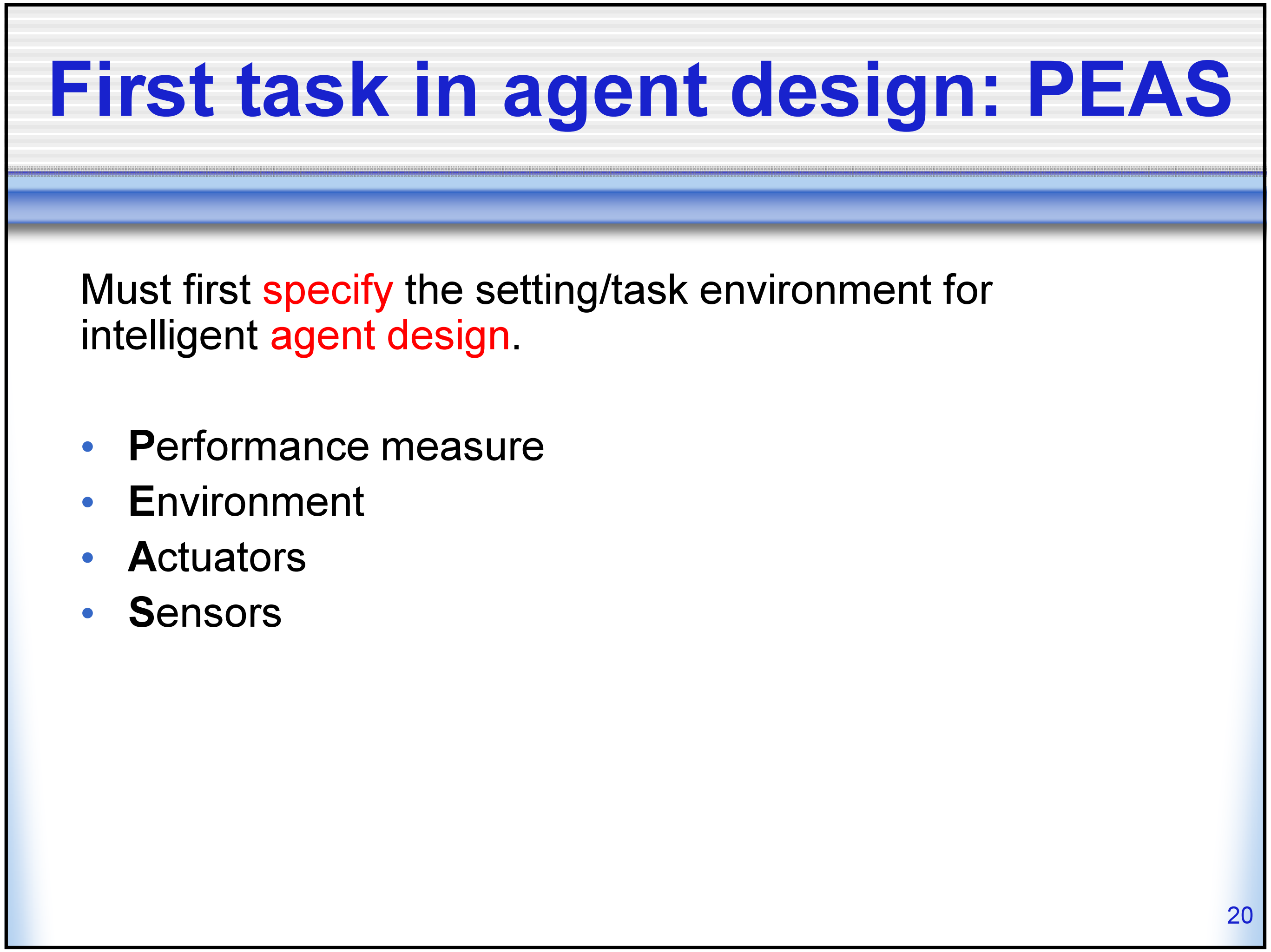
# Autonoumous Agents

- Agents can perform actions in order to obtain useful information (information gathering, exploration)
- An agent is autonomous if its behavior is determined by its own experience (with ability to learn and adapt)



# Applications

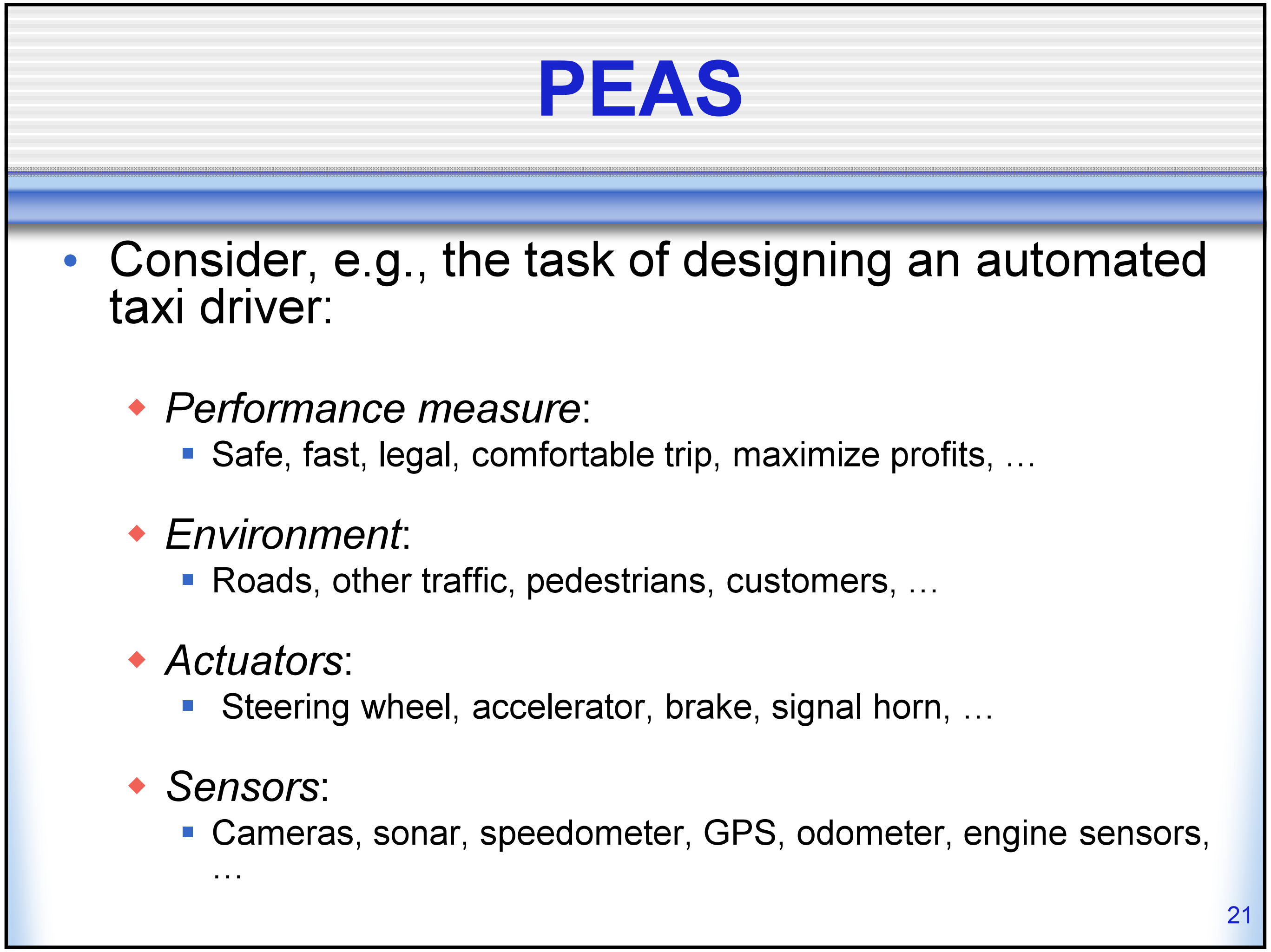
- Robotics: Drone, Explorer, Rescue BOT
- Web Agents: Personalized Search Egines
- Logistics: Tour planning
- Medicine: Diagnosis, Surgery, …
- …



# First task in agent design: PEAS

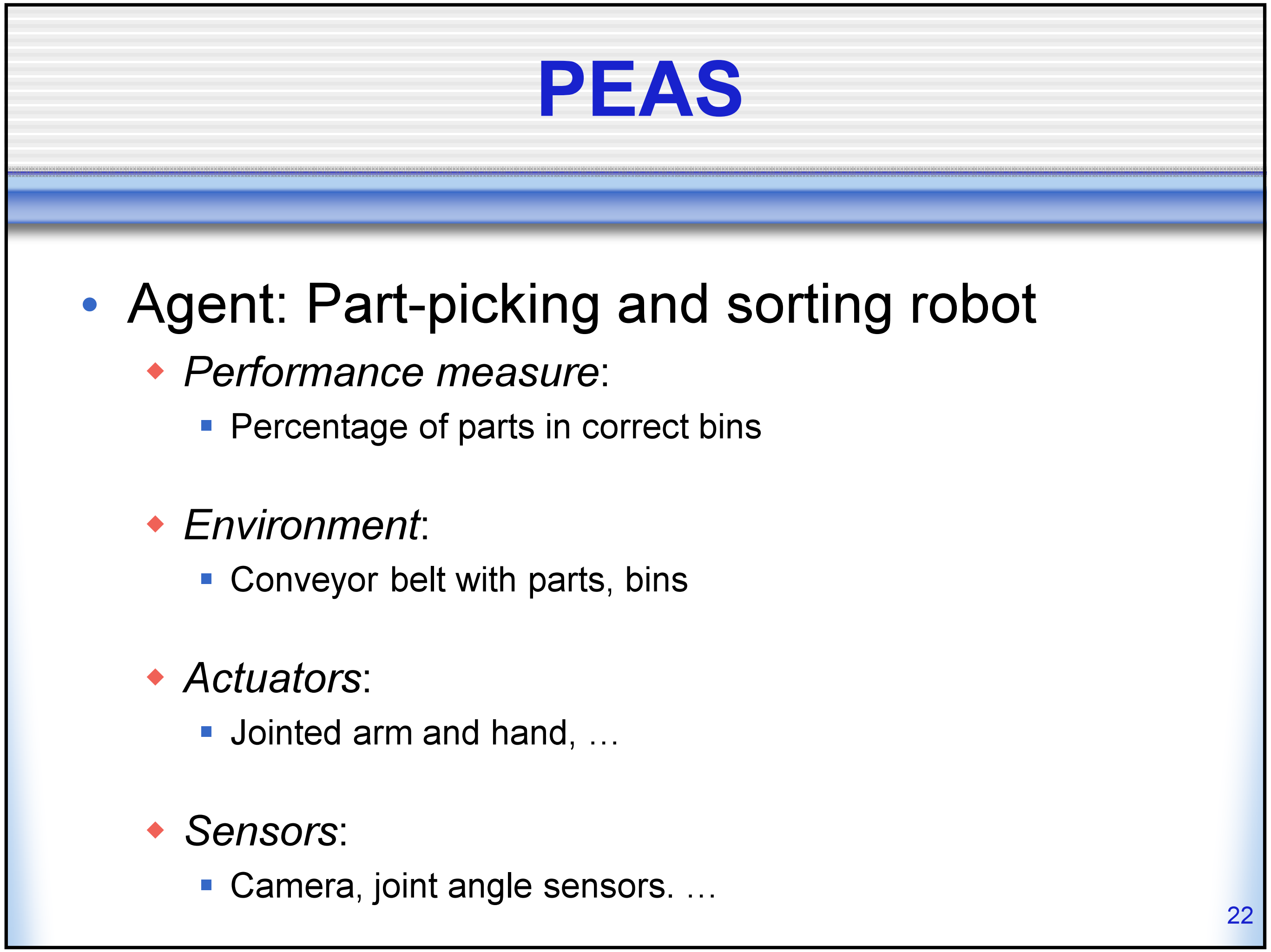
Must first specify the setting/task environment for intelligent agent design.

- **P**erformance measure
- **E**nvironment
- **A**ctuators
- **S**ensors



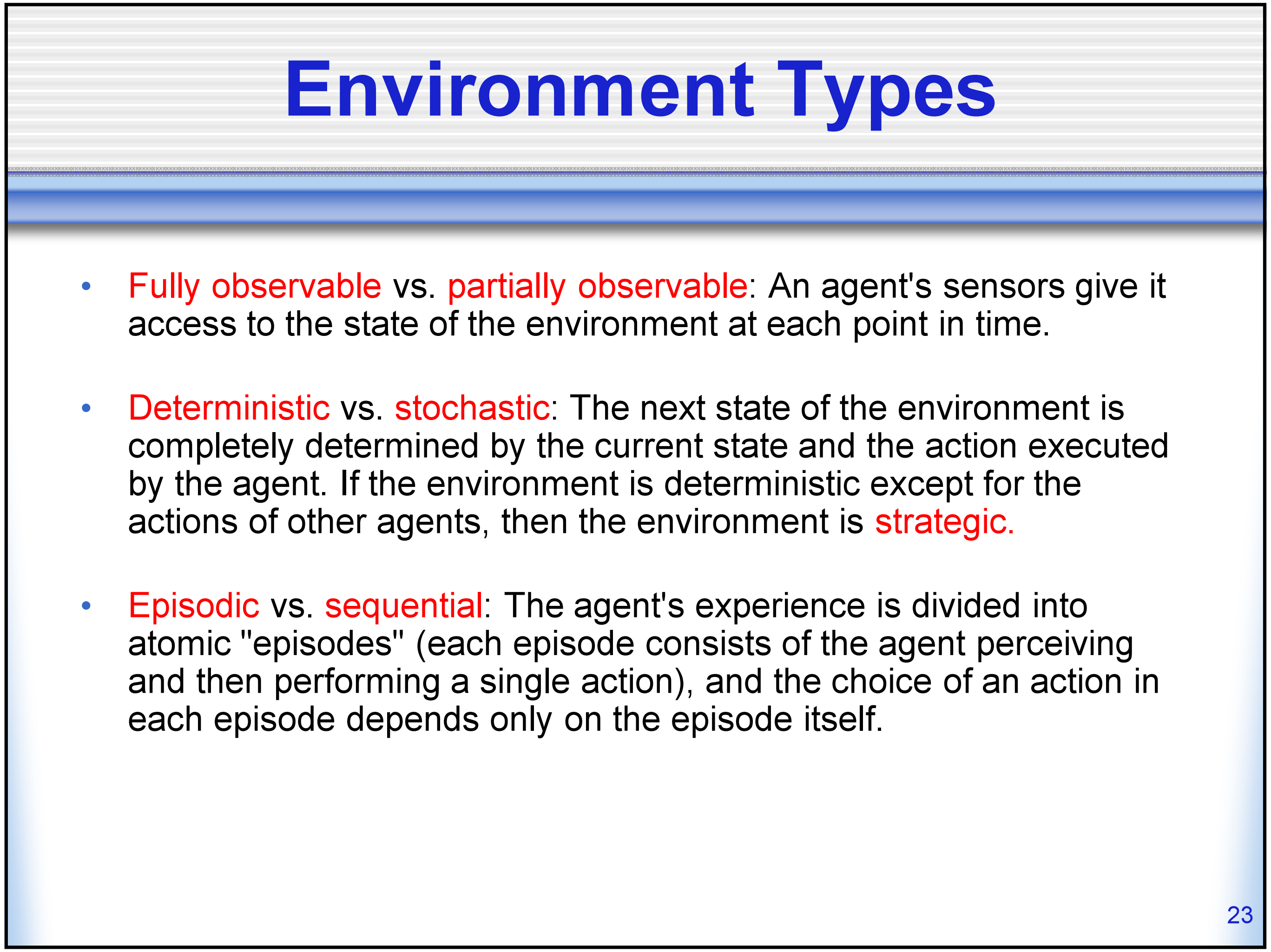
# PEAS

- Consider, e.g., the task of designing an automated taxi driver:
  - *Performance measure*:
    - Safe, fast, legal, comfortable trip, maximize profits, …
  - *Environment*:
    - Roads, other traffic, pedestrians, customers, …
  - *Actuators*:
    - Steering wheel, accelerator, brake, signal horn, …
  - *Sensors*:
    - Cameras, sonar, speedometer, GPS, odometer, engine sensors, …



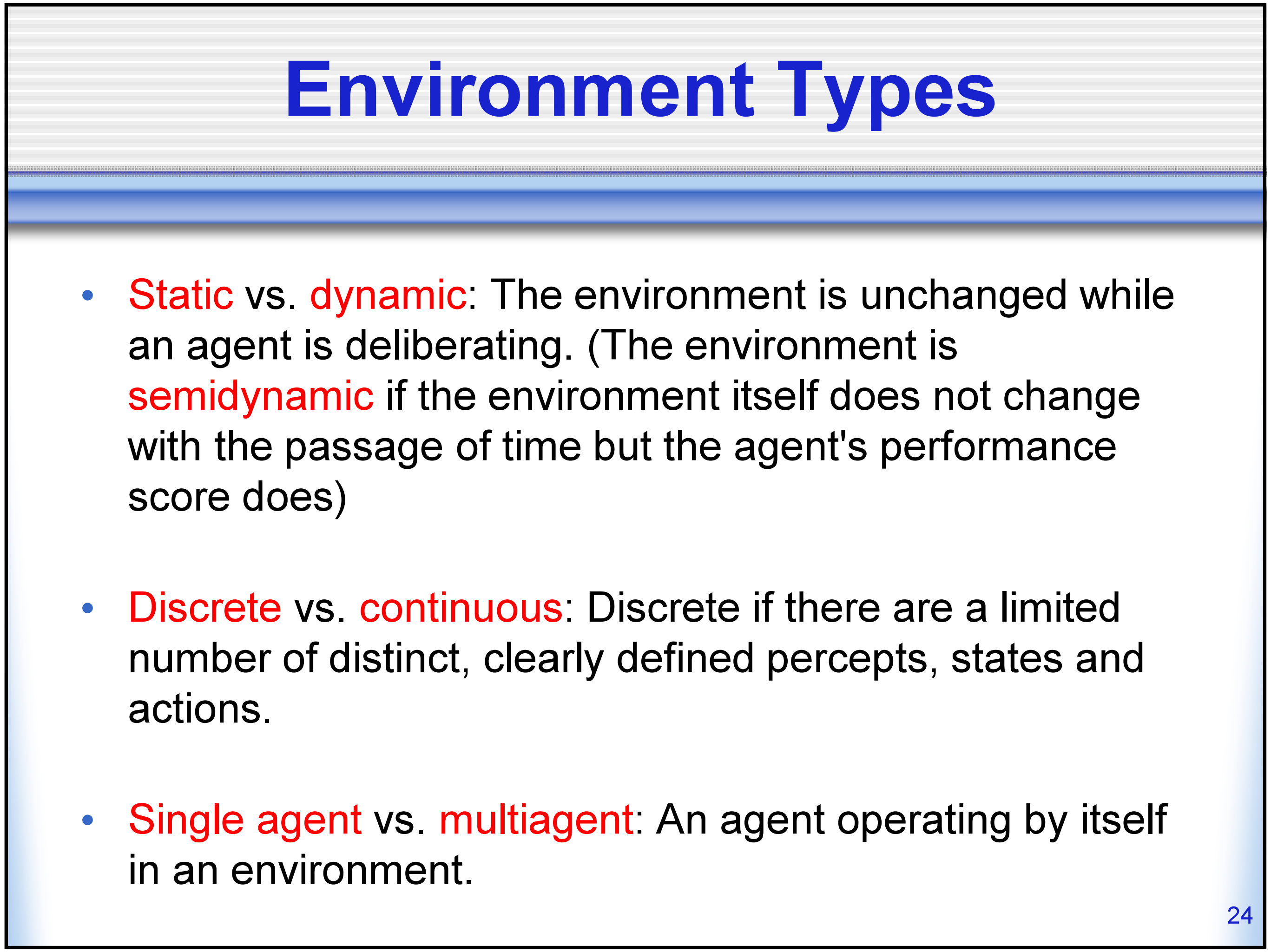
# PEAS

- Agent: Part-picking and sorting robot
  - *Performance measure*:
    - Percentage of parts in correct bins
  - *Environment*:
    - Conveyor belt with parts, bins
  - *Actuators*:
    - Jointed arm and hand, …
  - *Sensors*:
    - Camera, joint angle sensors. …



# Environment Types

- Fully observable vs. partially observable: An agent's sensors give it access to the state of the environment at each point in time.
- Deterministic vs. stochastic: The next state of the environment is completely determined by the current state and the action executed by the agent. If the environment is deterministic except for the actions of other agents, then the environment is strategic.
- Episodic vs. sequential: The agent's experience is divided into atomic "episodes" (each episode consists of the agent perceiving and then performing a single action), and the choice of an action in each episode depends only on the episode itself.



# Environment Types

- Static vs. dynamic: The environment is unchanged while an agent is deliberating. (The environment is semidynamic if the environment itself does not change with the passage of time but the agent's performance score does)

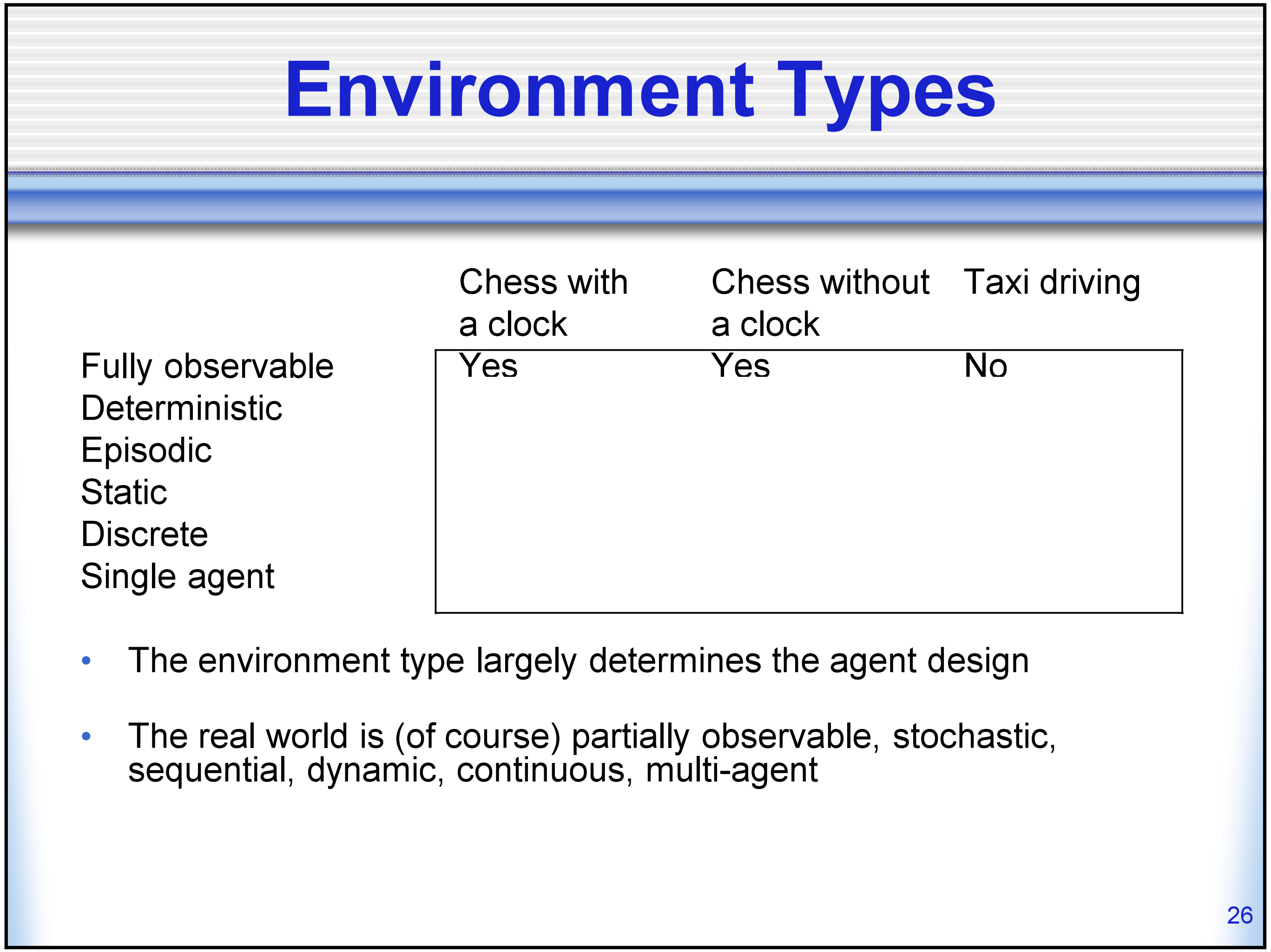
- Discrete vs. continuous: Discrete if there are a limited number of distinct, clearly defined percepts, states and actions.

- Single agent vs. multiagent: An agent operating by itself in an environment.

# Environment Types

| | Chess with a clock | Chess without a clock | Taxi driving |
|---|---|---|---|
| Fully observable | | | |
| Deterministic | | | |
| Episodic | | | |
| Static | | | |
| Discrete | | | |
| Single agent | | | |

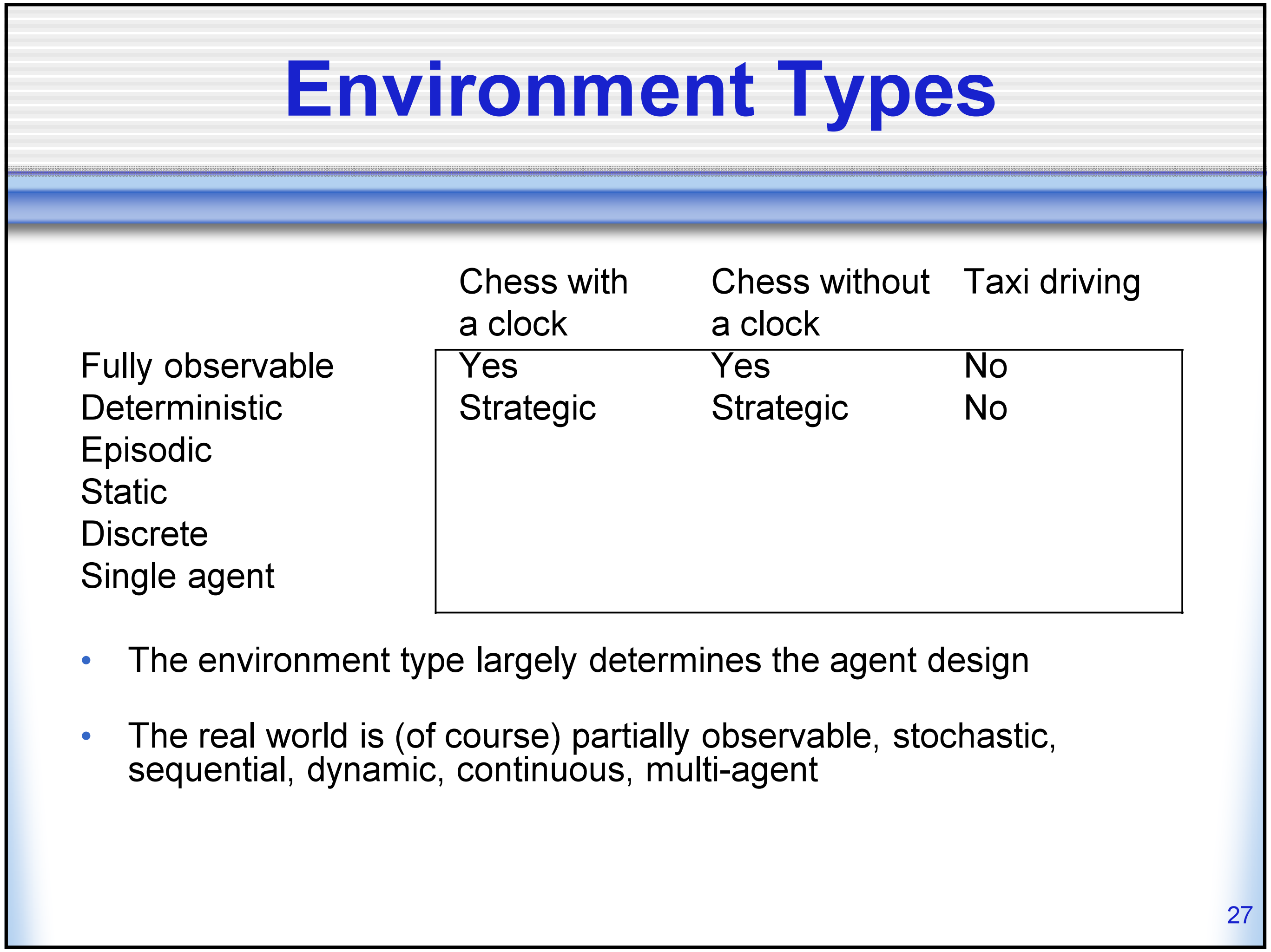
- The environment type largely determines the agent design
- The real world is (of course) partially observable, stochastic, sequential, dynamic, continuous, multi-agent



# Environment Types

| | Chess with a clock | Chess without a clock | Taxi driving |
|---|---|---|---|
| Fully observable | Yes | Yes | No |
| Deterministic | | | |
| Episodic | | | |
| Static | | | |
| Discrete | | | |
| Single agent | | | |

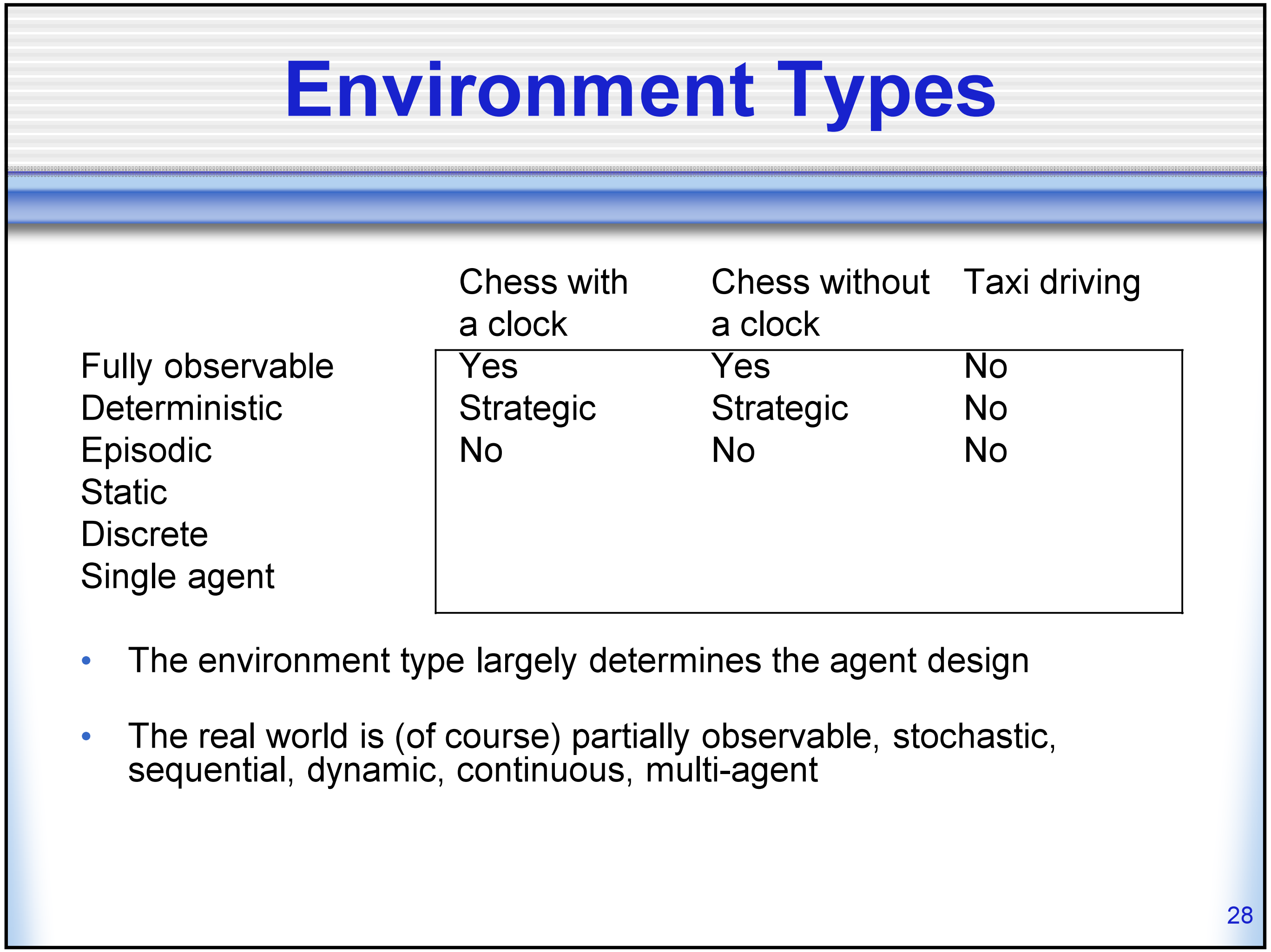
- The environment type largely determines the agent design
- The real world is (of course) partially observable, stochastic, sequential, dynamic, continuous, multi-agent



# Environment Types

| | Chess with a clock | Chess without a clock | Taxi driving |
|---|---|---|---|
| Fully observable | Yes | Yes | No |
| Deterministic | Strategic | Strategic | No |
| Episodic | | | |
| Static | | | |
| Discrete | | | |
| Single agent | | | |

- The environment type largely determines the agent design
- The real world is (of course) partially observable, stochastic, sequential, dynamic, continuous, multi-agent



# Environment Types

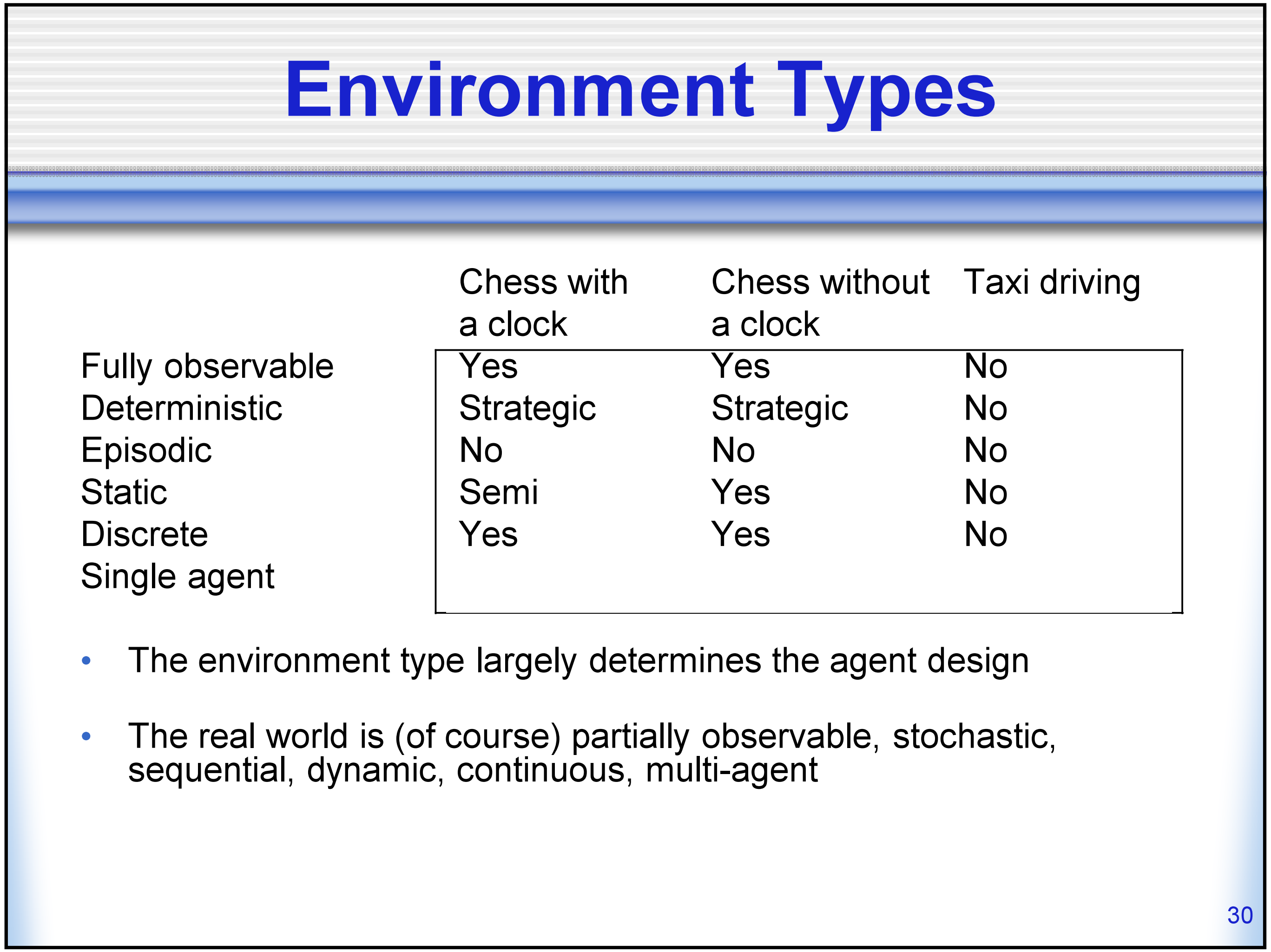
| | Chess with a clock | Chess without a clock | Taxi driving |
|---|---|---|---|
| Fully observable | Yes | Yes | No |
| Deterministic | Strategic | Strategic | No |
| Episodic | No | No | No |
| Static | | | |
| Discrete | | | |
| Single agent | | | |

- The environment type largely determines the agent design
- The real world is (of course) partially observable, stochastic, sequential, dynamic, continuous, multi-agent

# Environment Types

| | Chess with a clock | Chess without a clock | Taxi driving |
|---|---|---|---|
| Fully observable | Yes | Yes | No |
| Deterministic | Strategic | Strategic | No |
| Episodic | No | No | No |
| Static | Semi | Yes | No |
| Discrete | | | |
| Single agent | | | |

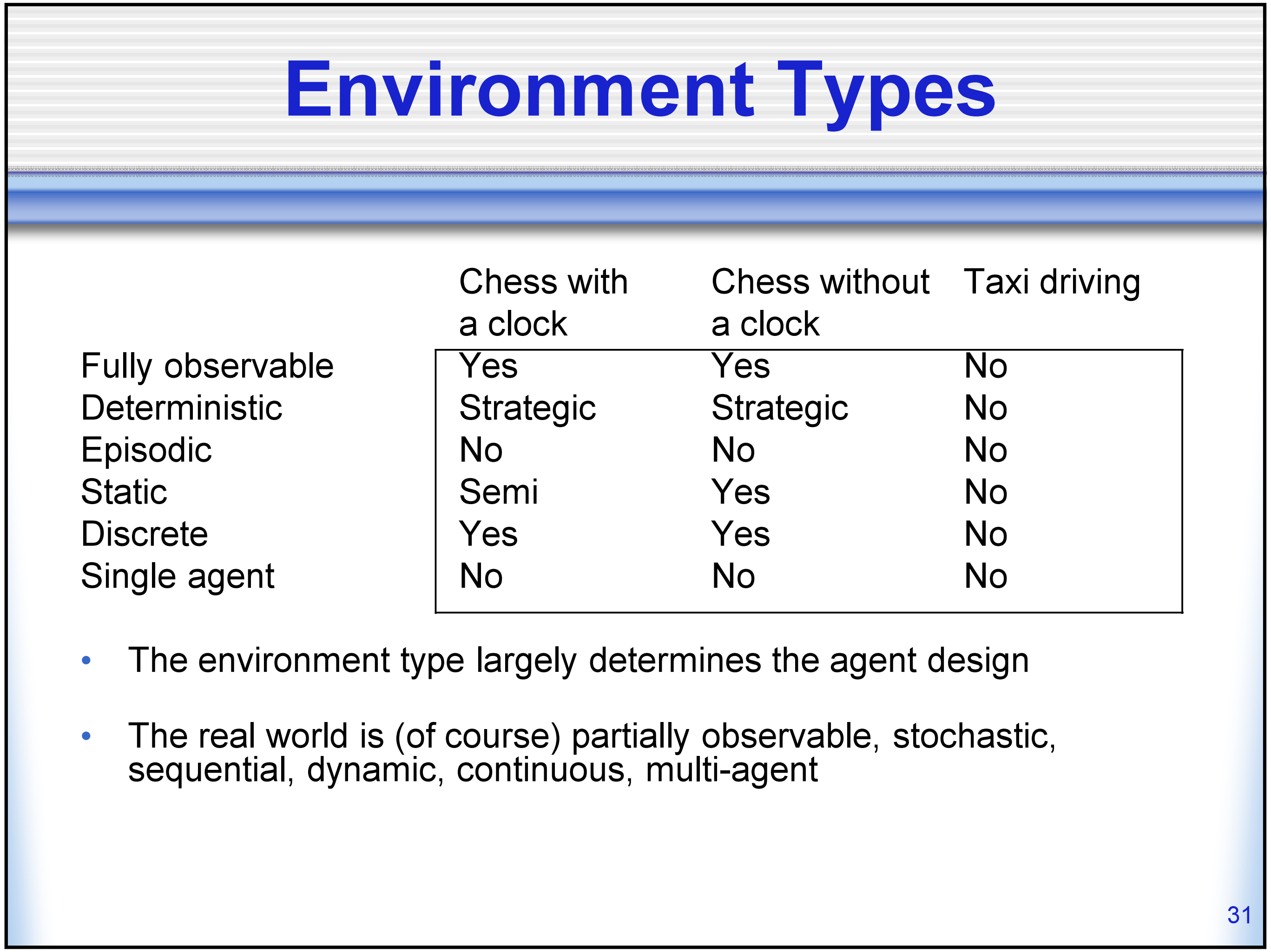
- The environment type largely determines the agent design
- The real world is (of course) partially observable, stochastic, sequential, dynamic, continuous, multi-agent



# Environment Types

| | Chess with a clock | Chess without a clock | Taxi driving |
|---|---|---|---|
| Fully observable | Yes | Yes | No |
| Deterministic | Strategic | Strategic | No |
| Episodic | No | No | No |
| Static | Semi | Yes | No |
| Discrete | Yes | Yes | No |
| Single agent | | | |

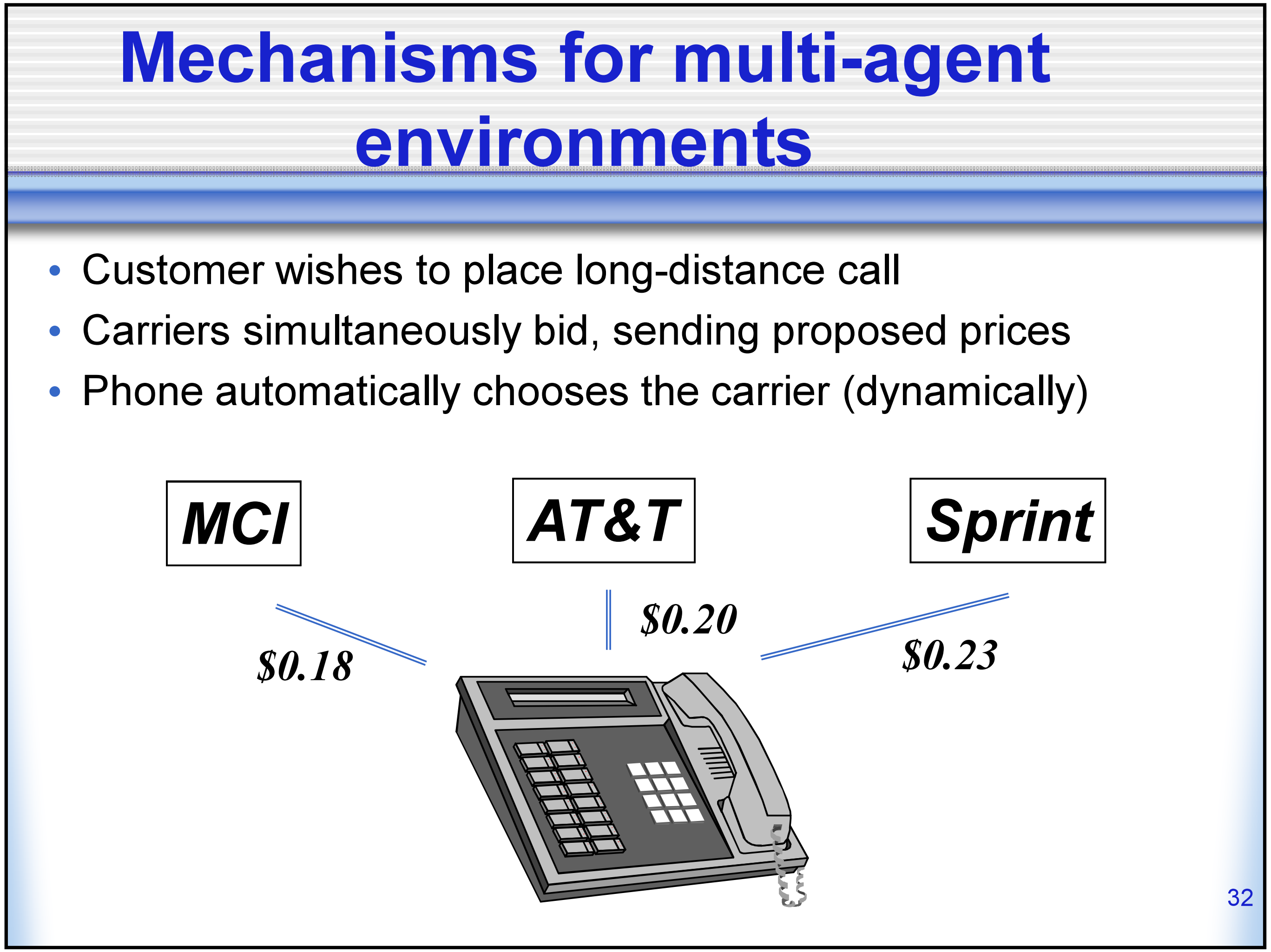
- The environment type largely determines the agent design
- The real world is (of course) partially observable, stochastic, sequential, dynamic, continuous, multi-agent



# Environment Types

| | Chess with a clock | Chess without a clock | Taxi driving |
|---|---|---|---|
| Fully observable | Yes | Yes | No |
| Deterministic | Strategic | Strategic | No |
| Episodic | No | No | No |
| Static | Semi | Yes | No |
| Discrete | Yes | Yes | No |
| Single agent | No | No | No |

- The environment type largely determines the agent design
- The real world is (of course) partially observable, stochastic, sequential, dynamic, continuous, multi-agent



# Mechanisms for multi-agent environments

- Customer wishes to place long-distance call
- Carriers simultaneously bid, sending proposed prices
- Phone automatically chooses the carrier (dynamically)

MCI

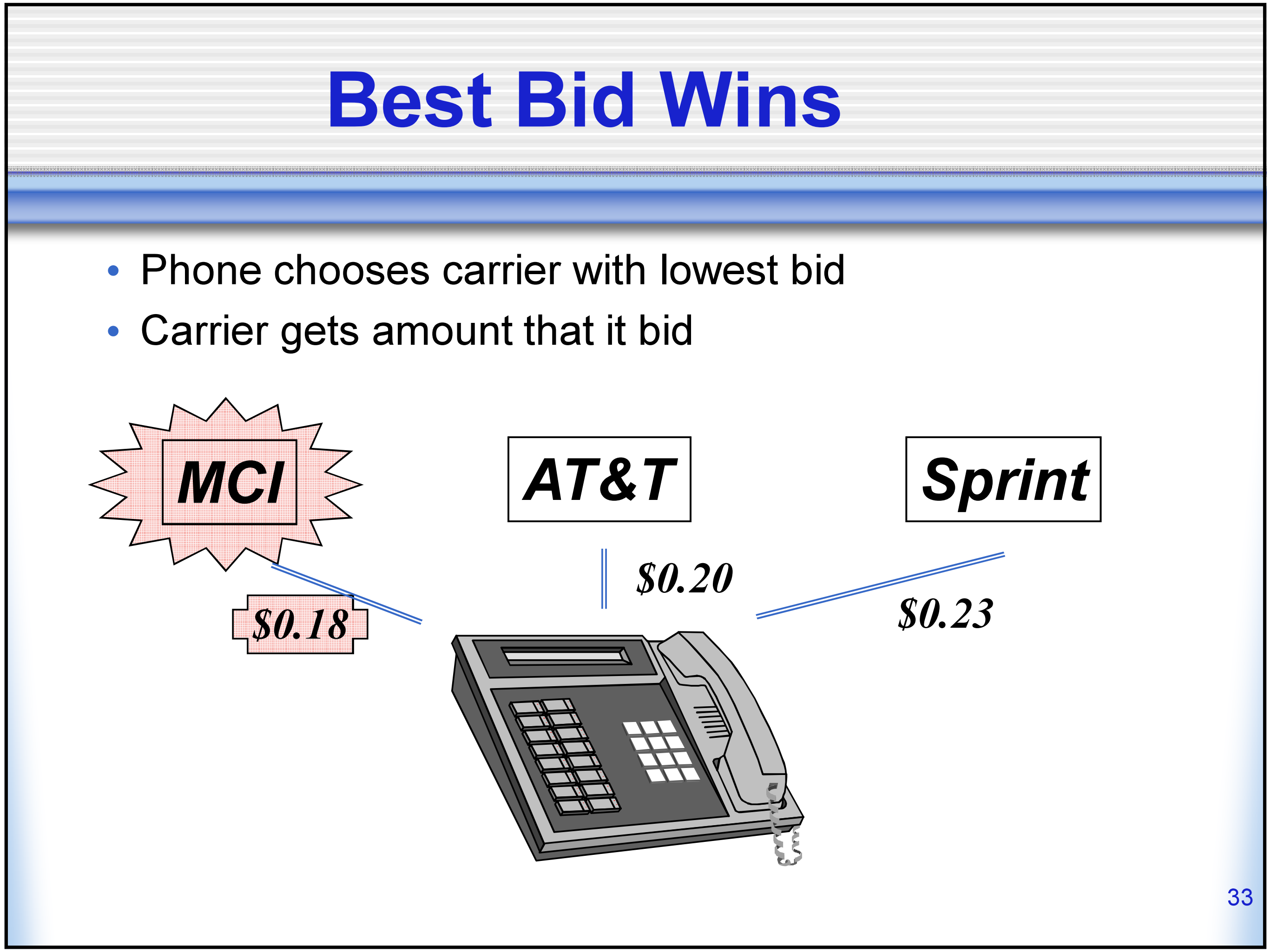
AT&T

Sprint

$0.18

$0.20

$0.23



# Best Bid Wins

- Phone chooses carrier with lowest bid
- Carrier gets amount that it bid

MCI

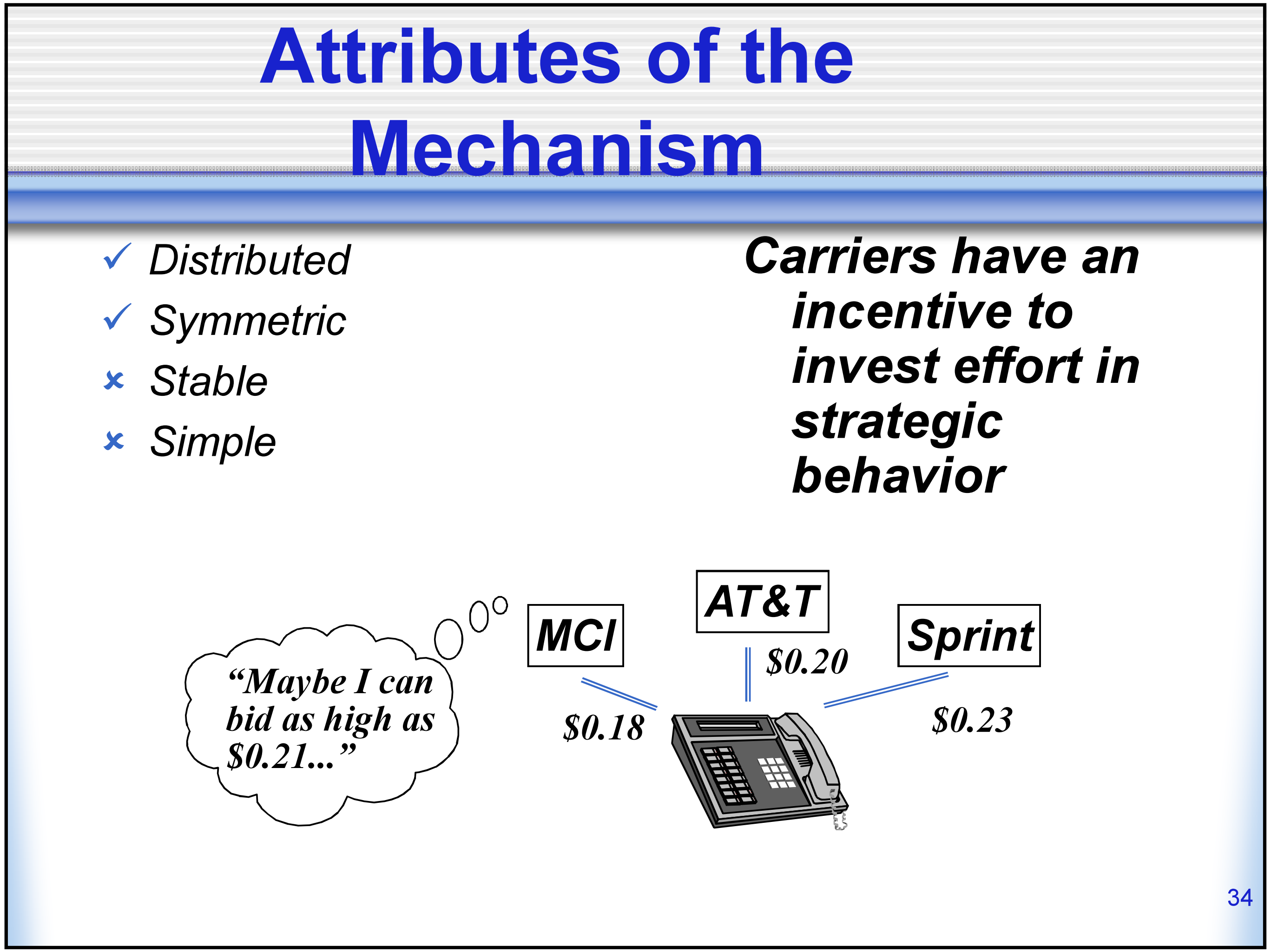
AT&T

Sprint

$0.18

$0.20

$0.23



# Attributes of the Mechanism

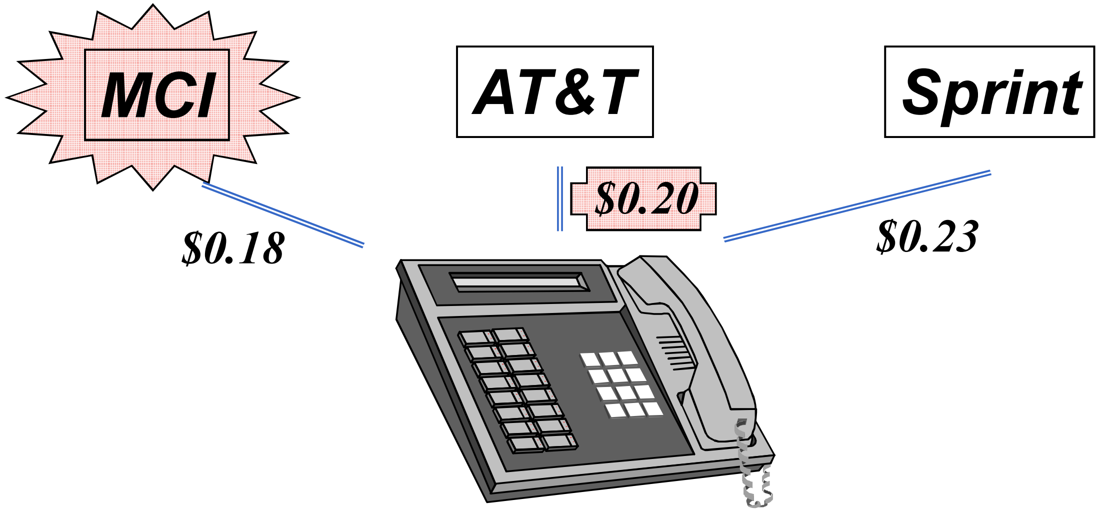
- ✓ *Distributed*
- ✓ *Symmetric*
- ✗ *Stable*
- ✗ *Simple*

***Carriers have an incentive to invest effort in strategic behavior***

"Maybe I can bid as high as $0.21…"
MCI
$0.18
AT&T
$0.20
Sprint
$0.23

# Best Bid Wins, Gets Second Price (Vickrey Auction)

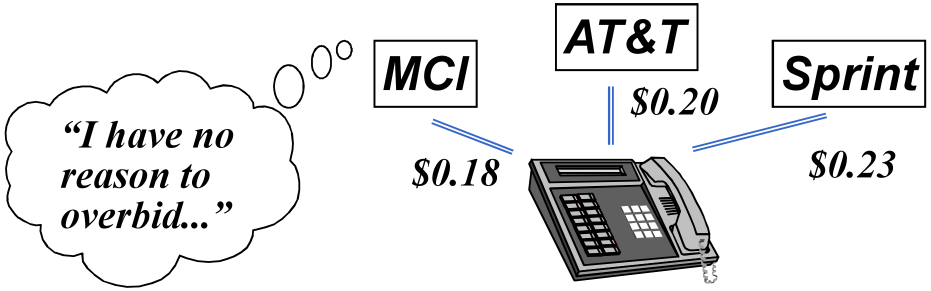
- Phone chooses carrier with lowest bid
- Carrier gets amount of second-best price



# Attributes of the Vickrey Mechanism

- ✓ *Distributed*
- ✓ *Symmetric*
- ✓ *Stable*
- ✓ *Simple*

***Carriers have no incentive to invest effort in strategic behavior***



*"I have no reason to overbid..."*

**MCI** *$0.18*

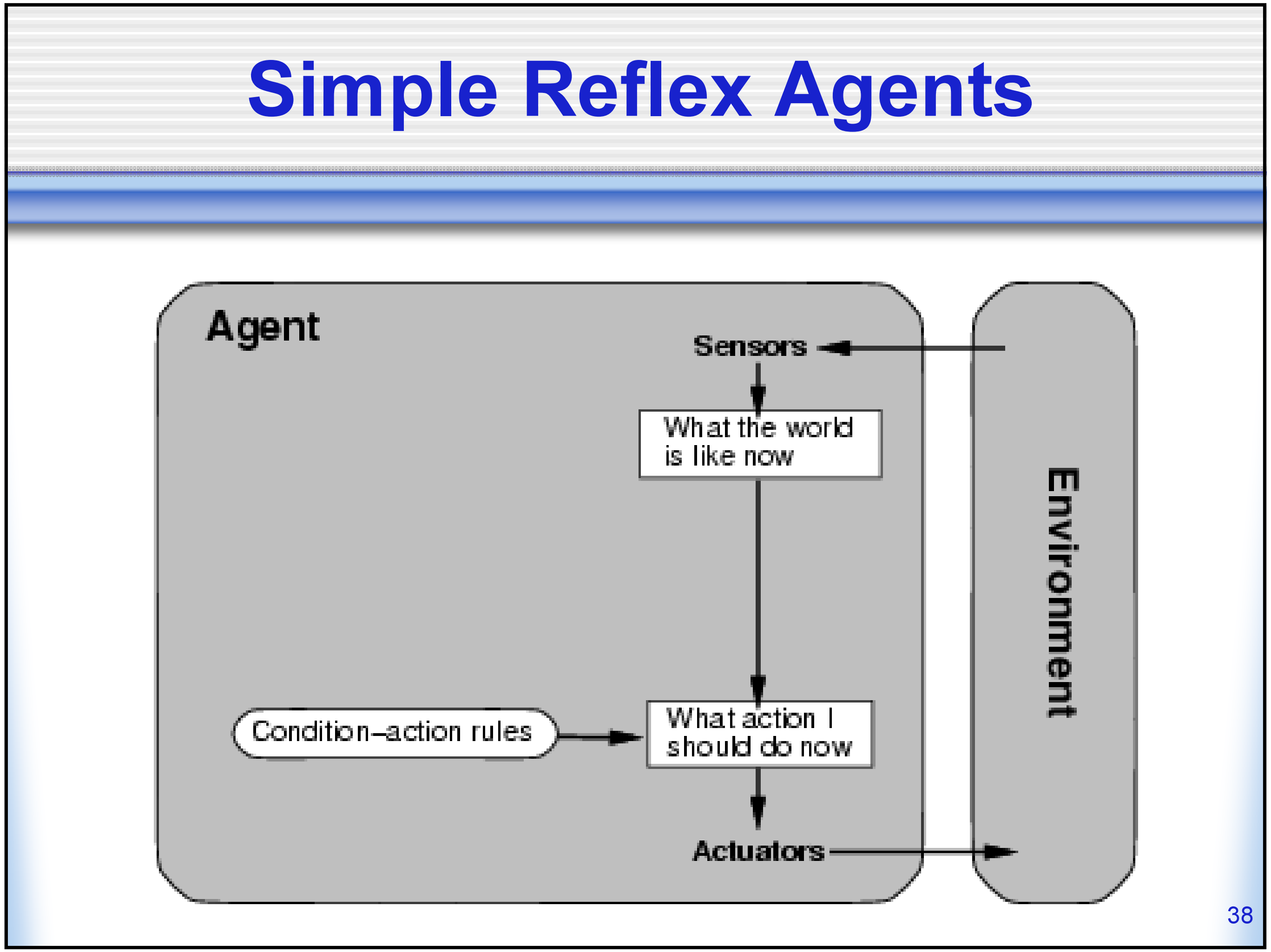
**AT&T** *$0.20*

**Sprint** *$0.23*

# Agent Types

- Five basic types in order of increasing generality:
  - Simple reflex agents
  - Model-based reflex agents
  - Goal-based agents
  - Utility-based agents
  - Learning agents
    see lecture *Machine Learning*

add features



# Simple Reflex Agents

Agent

Sensors

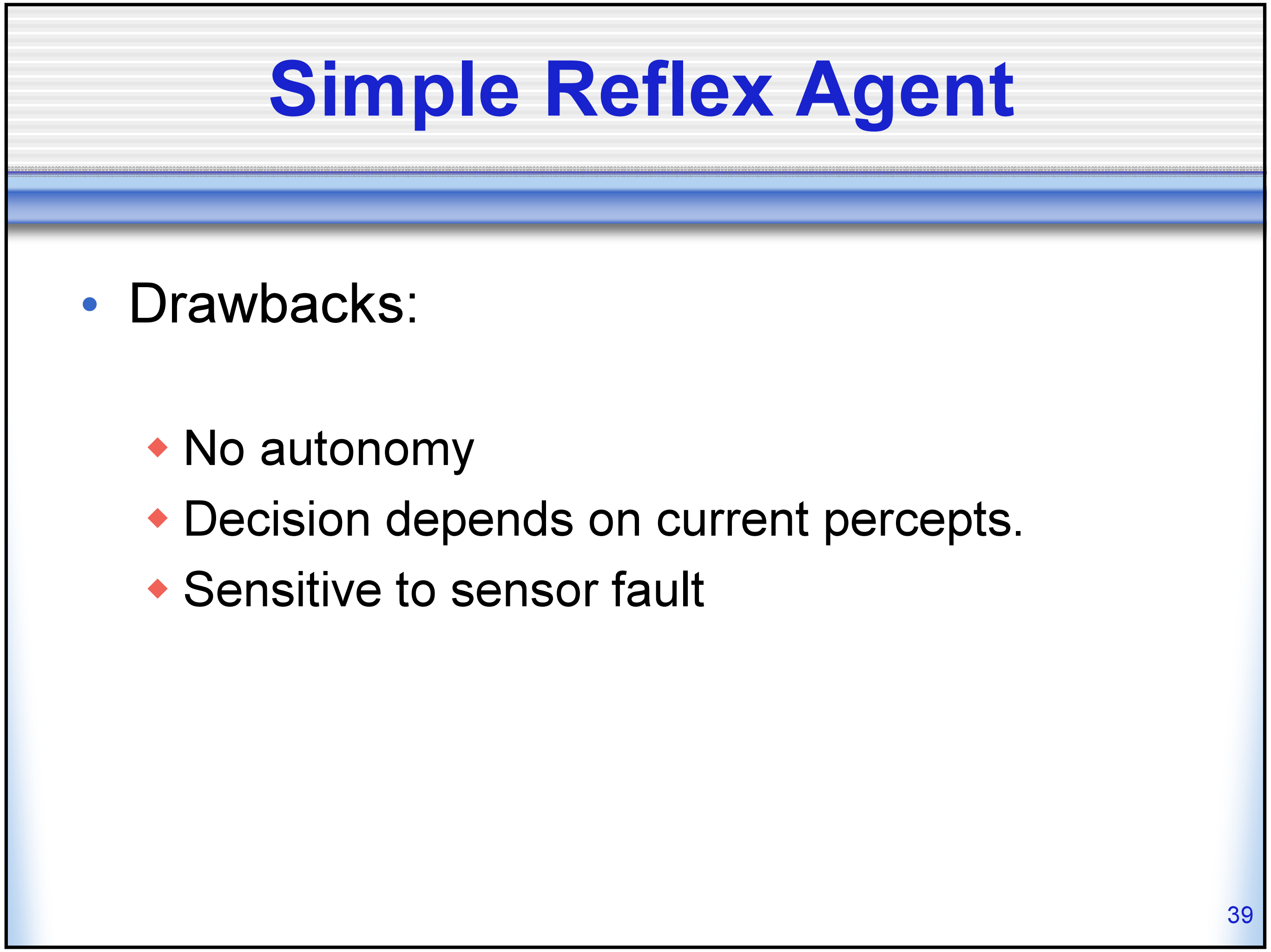
What the world is like now

Condition–action rules

What action I should do now

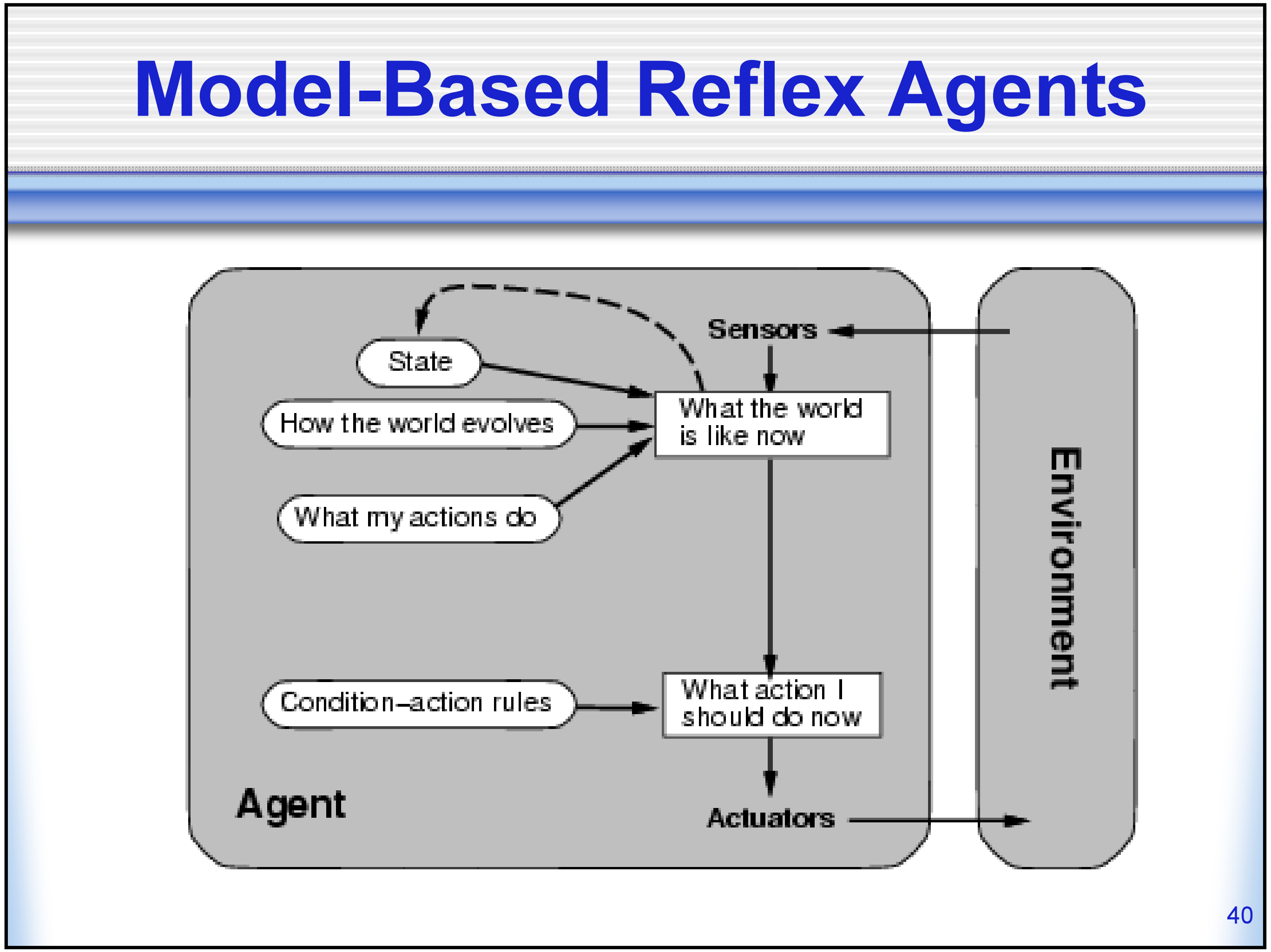
Actuators

Environment



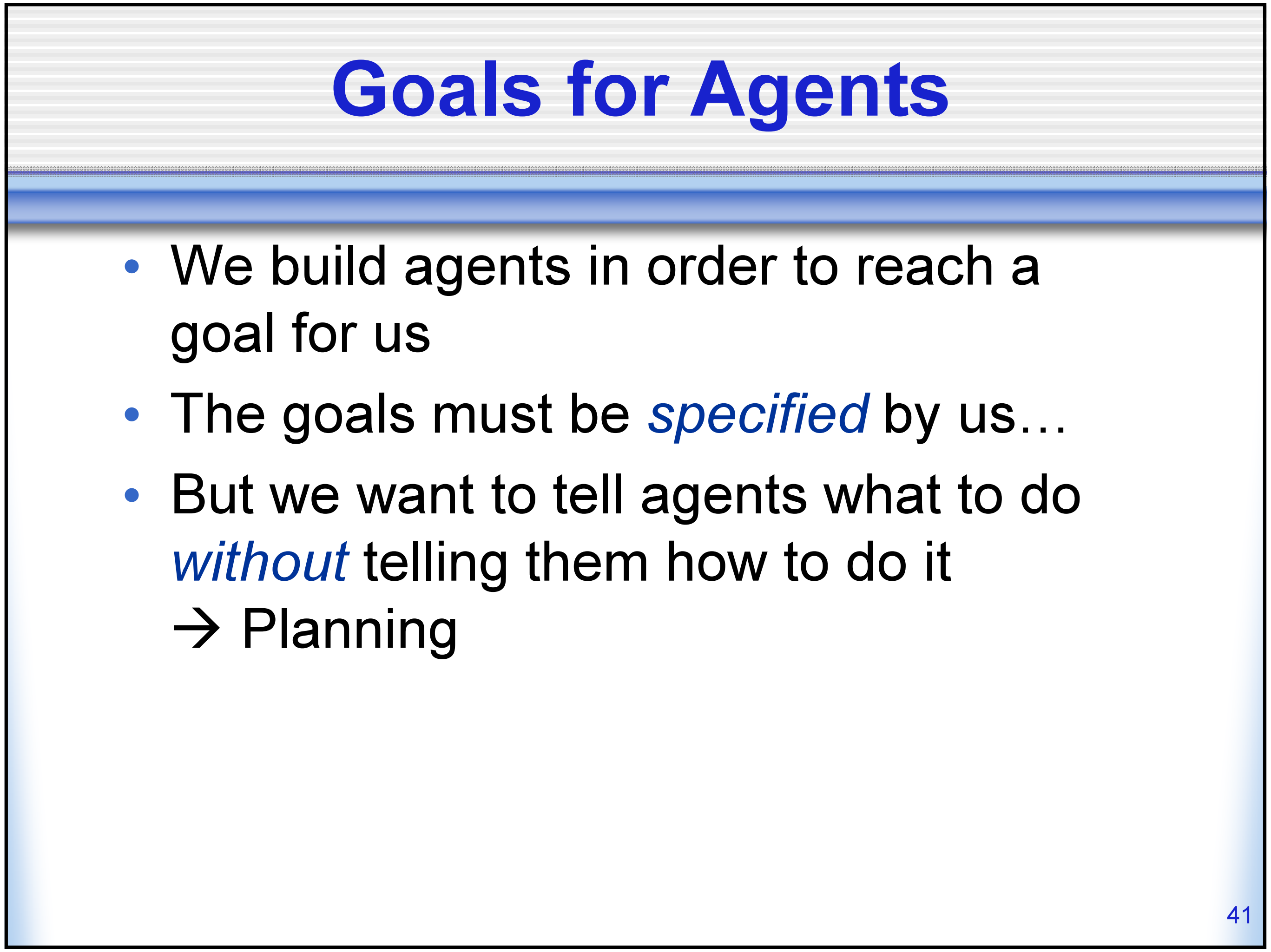
# Simple Reflex Agent

- Drawbacks:
  - No autonomy
  - Decision depends on current percepts.
  - Sensitive to sensor fault



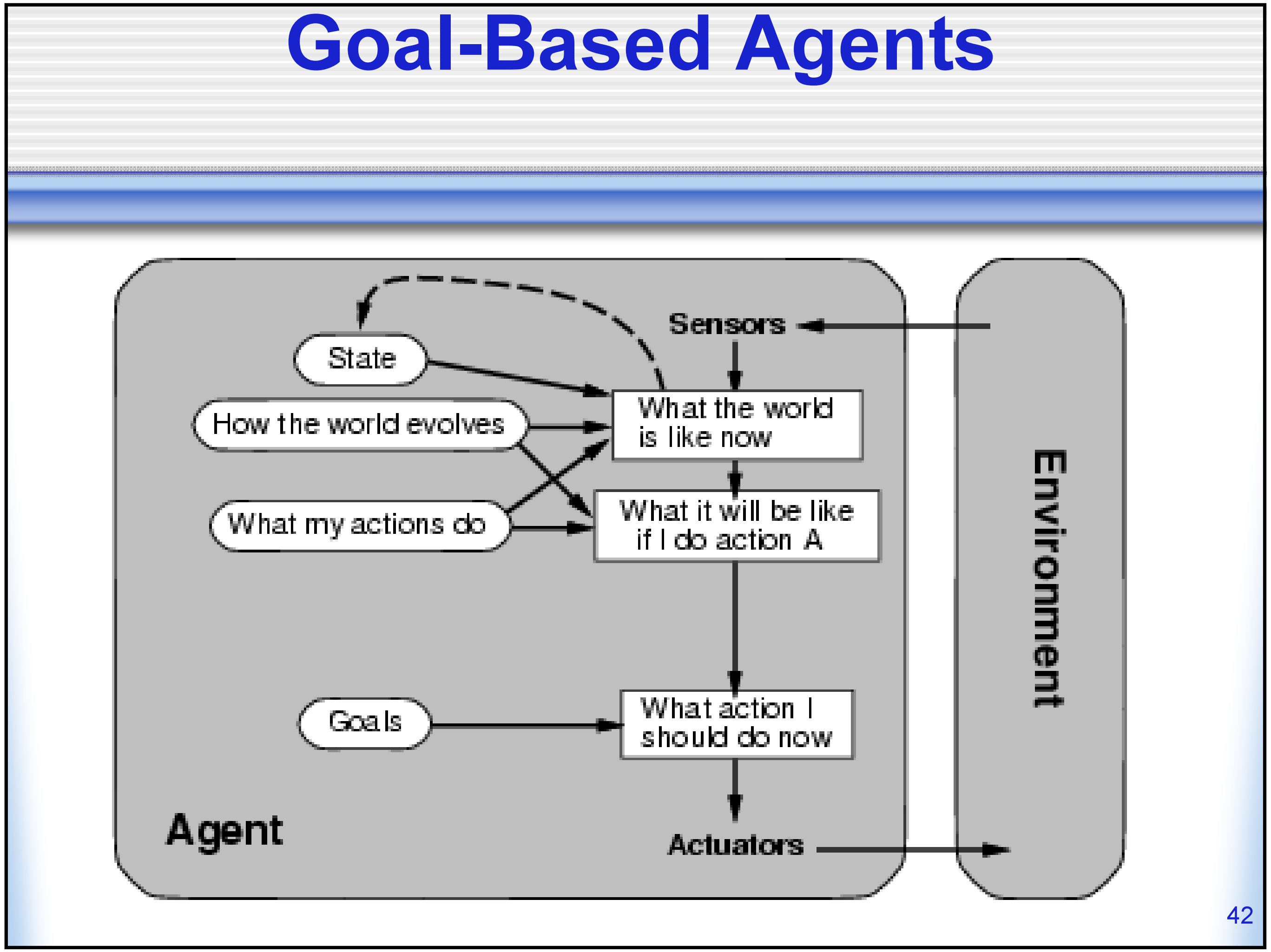
# Model-Based Reflex Agents

40



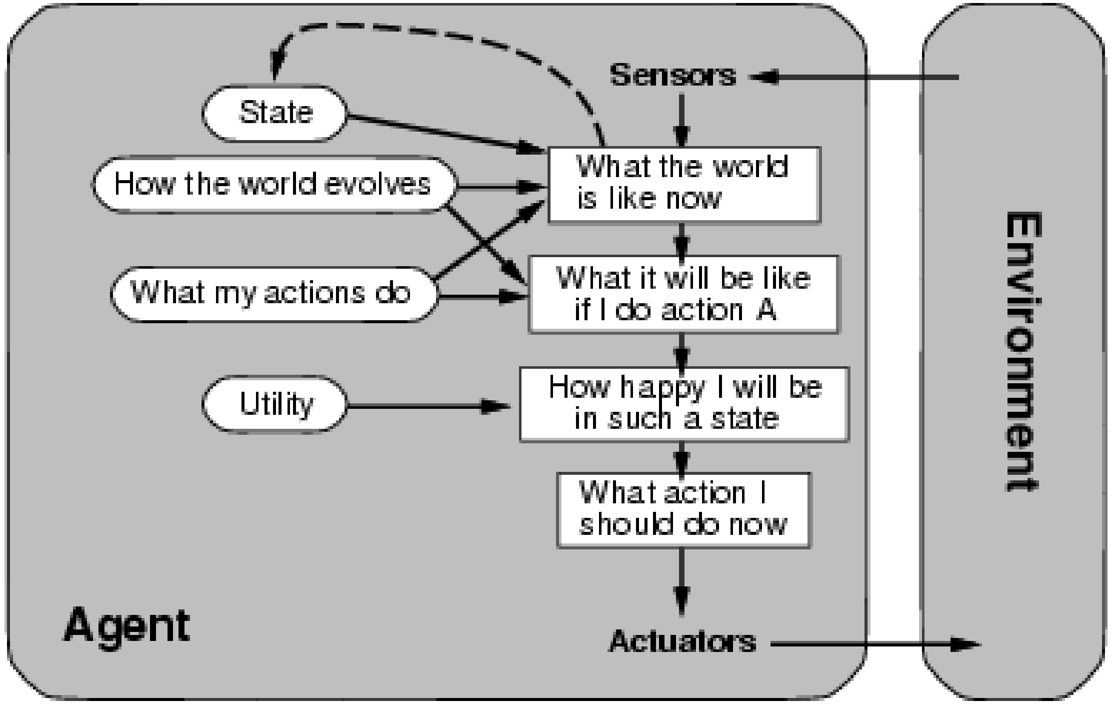
# Goals for Agents

- We build agents in order to reach a goal for us
- The goals must be *specified* by us…
- But we want to tell agents what to do *without* telling them how to do it
  → Planning

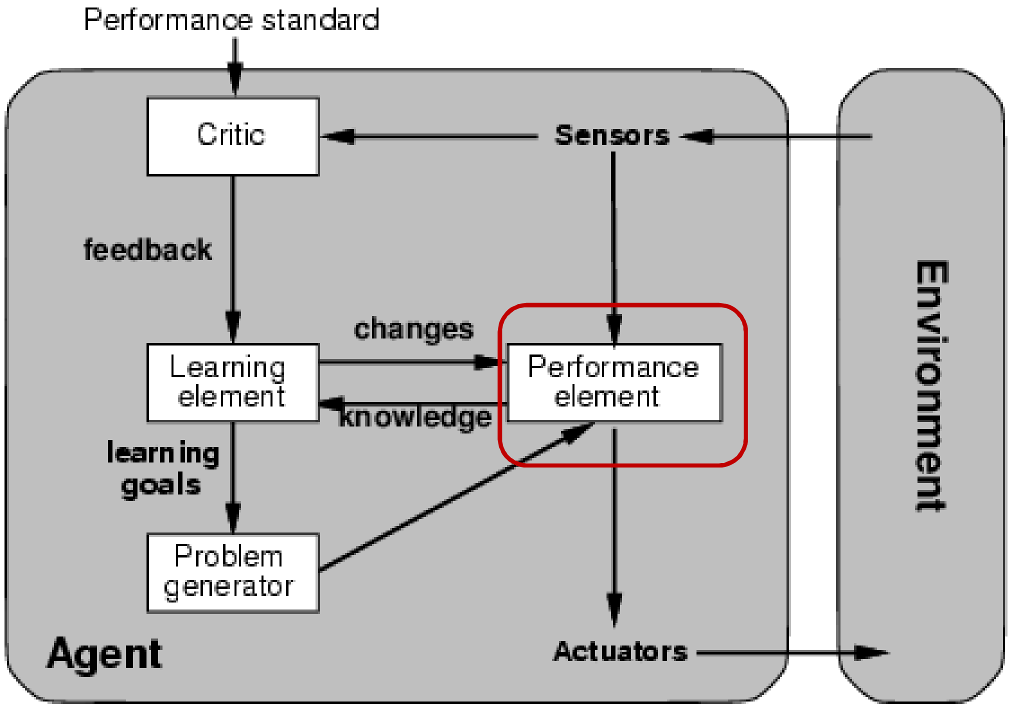


# Goal-Based Agents

# Utility-Based Agents

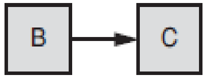

State
How the world evolves
What my actions do
Utility
Sensors
What the world is like now
What it will be like if I do action A
How happy I will be in such a state
What action I should do now
Actuators
Agent
Environment

# Learning Agents



# Representation of agent states

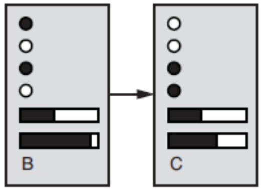
- How to represent the states of agents?
- Atomic



- State is a black box.
- Example:
  Want to travel from city B to C. Cities are represented as names.

# Representation of agent states

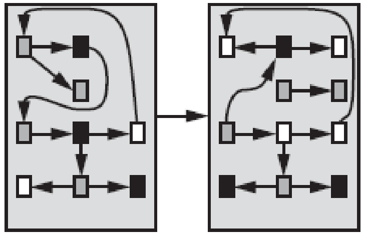
- Factored



- A state consists of a vector of attribute values
- Example: A car with GPS location, Fuel, radio station, …

# Representation of agent states

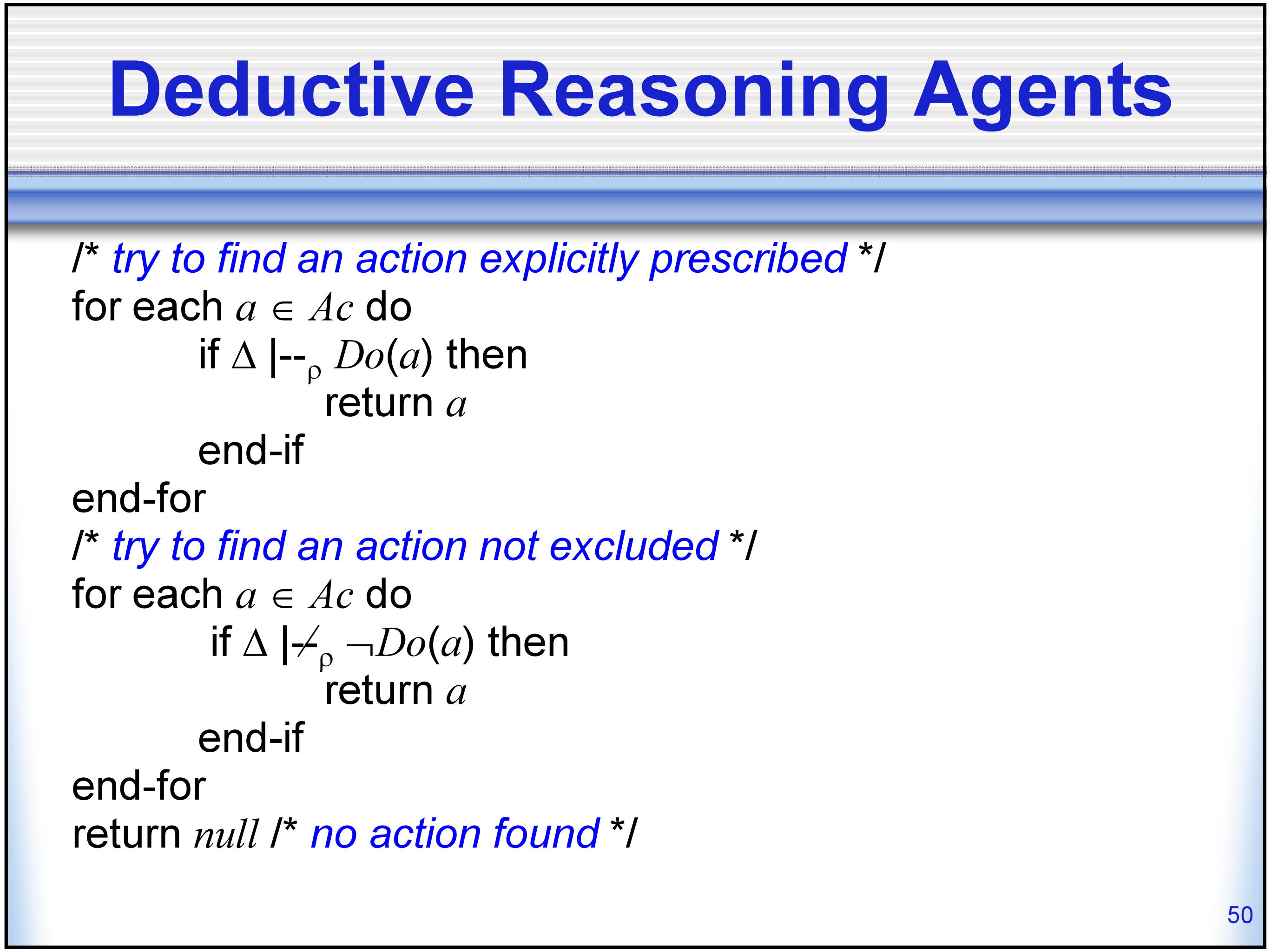
- Structured



- A state includes objects, each of which may have attributes of its own as well as relationships to other objects
- Example: natural language understanding

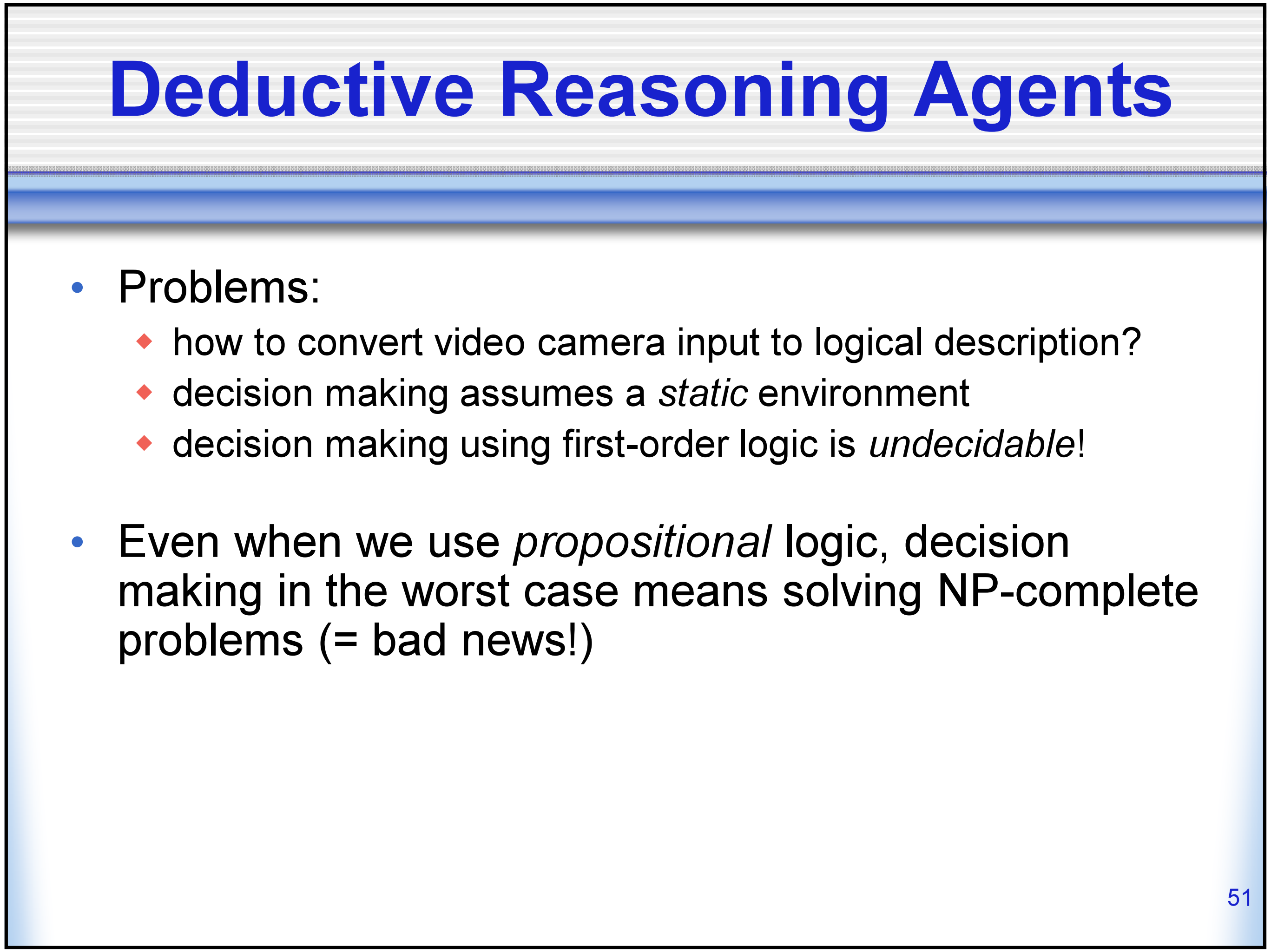
# AI first approach: Deductive Reasoning Agents

- How can an agent decide what to do using theorem proving?
- Basic idea is to use logic to encode a theory stating the *best* action to perform in any given situation
- Let:
  - $\rho$ be this theory (e.g. a set of rules)
  - $\Delta$ be a logical database that describes the current state of the world
  - $Ac$ be the set of actions the agent can perform
  - $\Delta \vdash_{\rho} \phi$ mean that $\phi$ can be proven from $\Delta$ using $\rho$



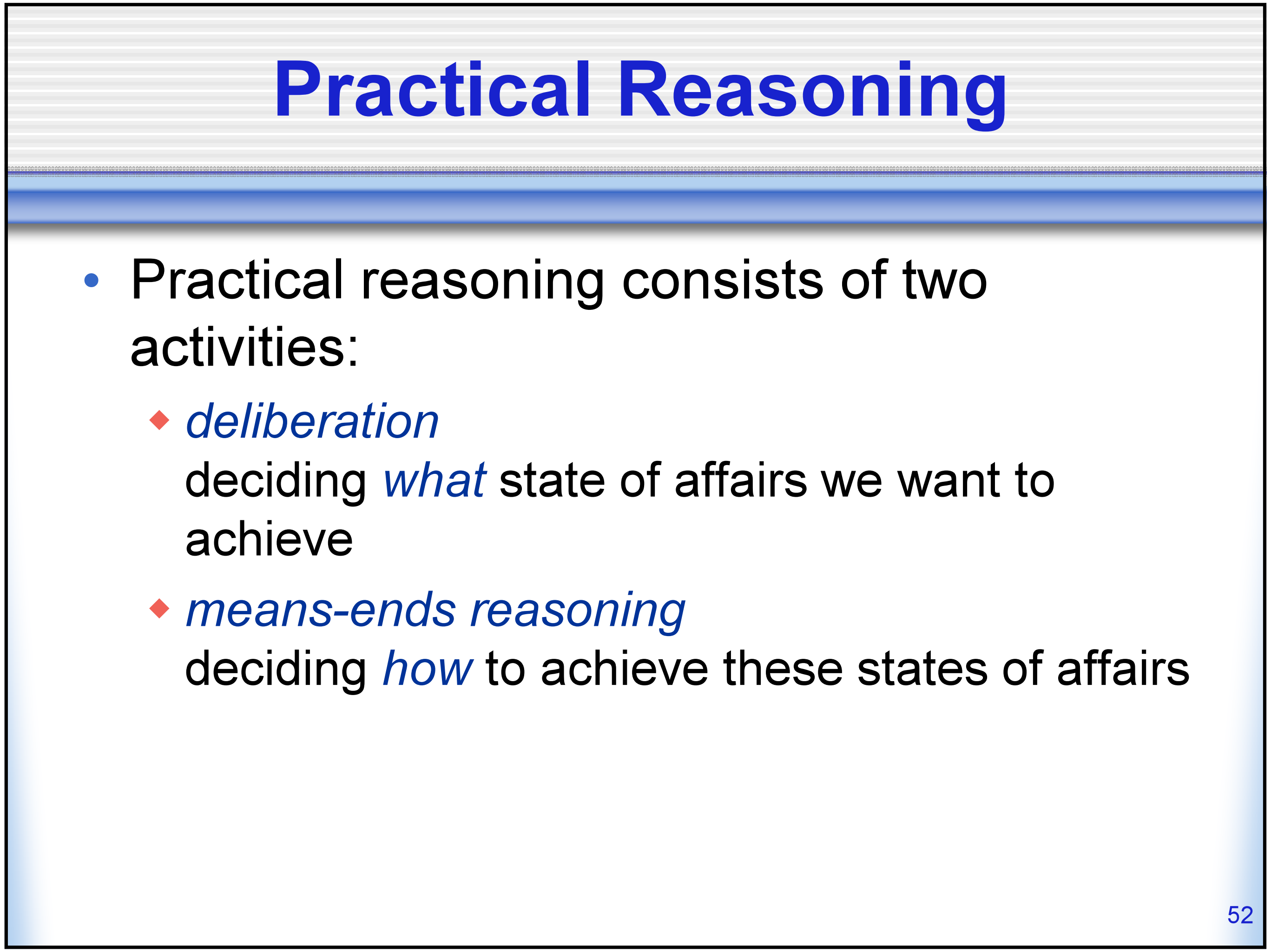
# Deductive Reasoning Agents

```
/* try to find an action explicitly prescribed */
for each $a \in Ac$ do
        if $\Delta \vdash_\rho Do(a)$ then
                return $a$
        end-if
end-for
/* try to find an action not excluded */
for each $a \in Ac$ do
        if $\Delta \nvdash_\rho \neg Do(a)$ then
                return $a$
        end-if
end-for
return $null$ /* no action found */
```



# Deductive Reasoning Agents

- Problems:
  - how to convert video camera input to logical description?
  - decision making assumes a *static* environment
  - decision making using first-order logic is *undecidable*!
- Even when we use *propositional* logic, decision making in the worst case means solving NP-complete problems (= bad news!)



# Practical Reasoning

- Practical reasoning consists of two activities:
  - *deliberation*
    deciding *what* state of affairs we want to achieve
  - *means-ends reasoning*
    deciding *how* to achieve these states of affairs

# What is Means-End Reasoning?

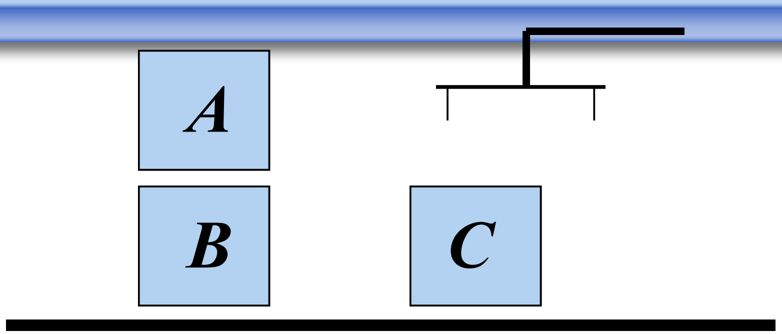
- Basic idea is to give an agent:
  - representation of goal to achieve
  - representation of actions it can perform
  - representation of the environment

and have it generate a *plan* to achieve the goal

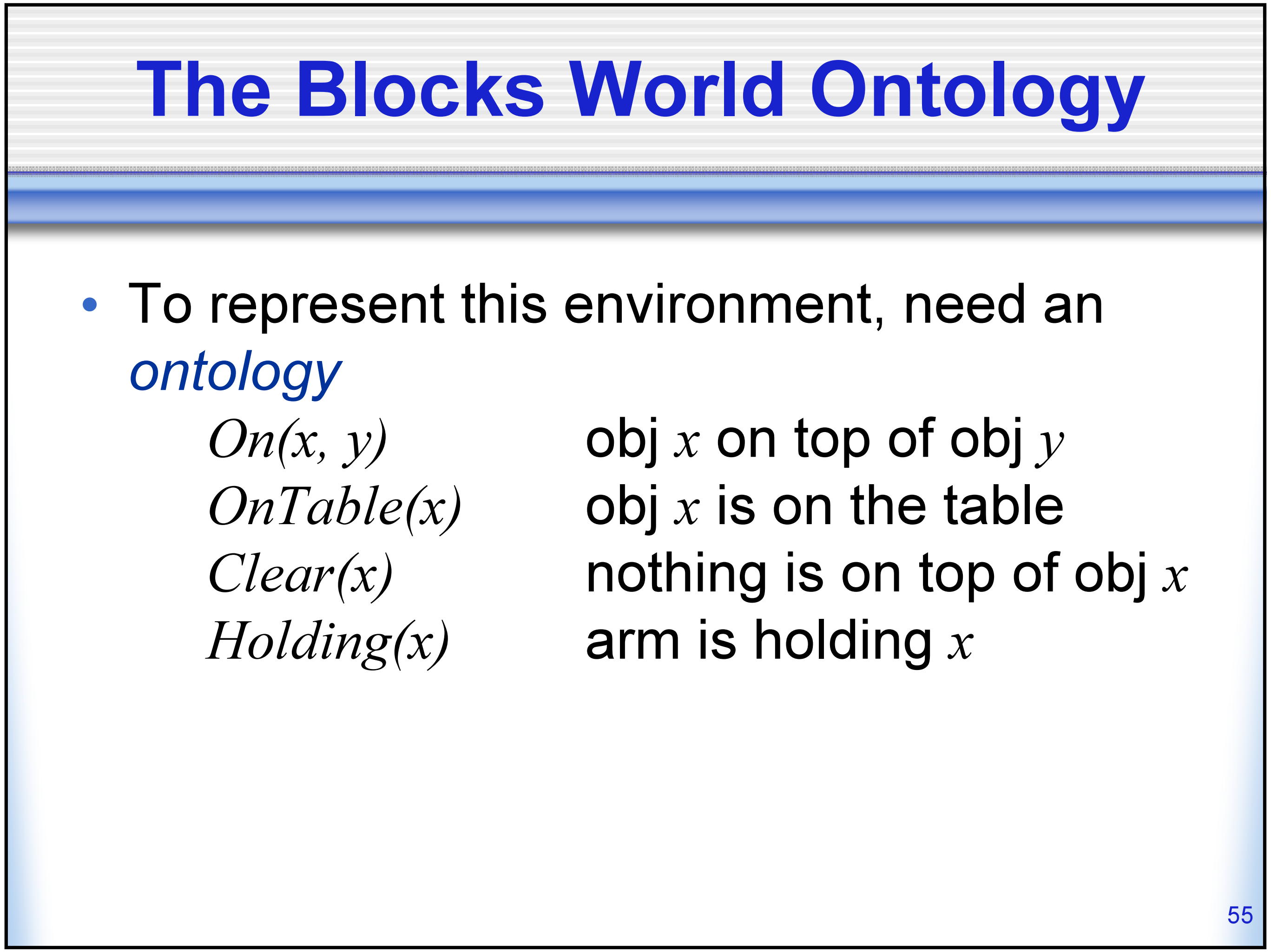
- Essentially, this is

*automatic programming*

# The Blocks World



- We’ll illustrate the techniques with the *blocks world*
- Contains a robot arm, 3 blocks (A, B, and C) of equal size, and a table



# The Blocks World Ontology

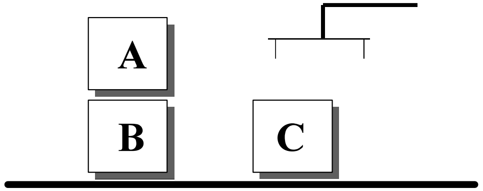
- To represent this environment, need an *ontology*

| | |
|---|---|
| *On(x, y)* | obj $x$ on top of obj $y$ |
| *OnTable(x)* | obj $x$ is on the table |
| *Clear(x)* | nothing is on top of obj $x$ |
| *Holding(x)* | arm is holding $x$ |

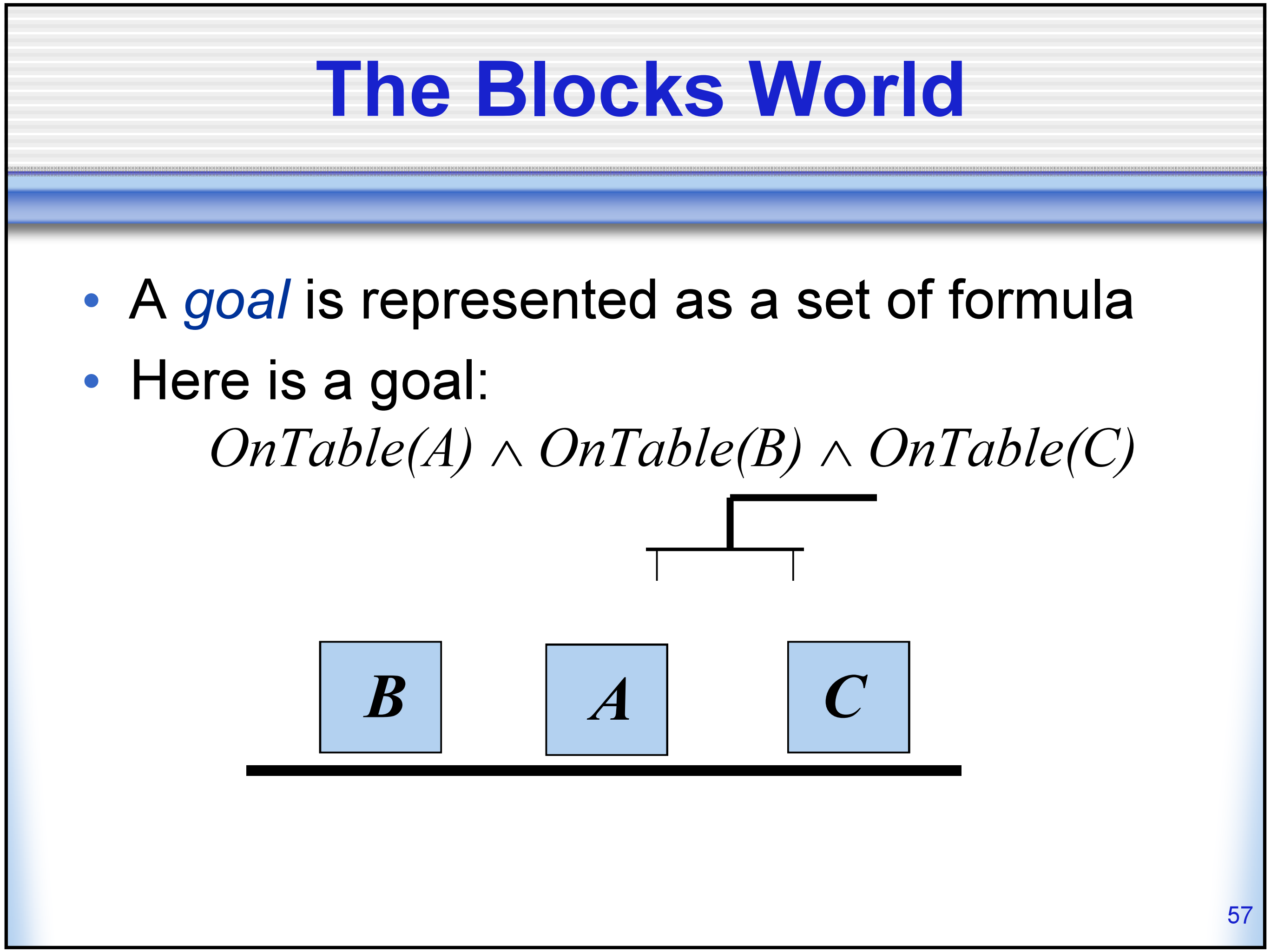
# The Blocks World

- Here is a representation of the blocks world described above:

  *Clear(A)*
  *On(A, B)*
  *OnTable(B)*
  *OnTable(C)*



- Use the *closed world assumption*: anything not stated is assumed to be *false*



# The Blocks World

- A *goal* is represented as a set of formula
- Here is a goal:

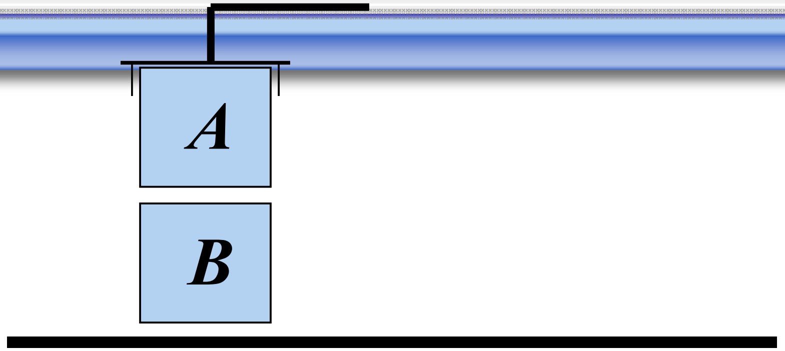
$$OnTable(A) \wedge OnTable(B) \wedge OnTable(C)$$

# The Blocks World

- *Actions* are represented using a technique that was developed in the STRIPS planner
- Each action has:
  - a *name*
    which may have arguments
  - a *pre-condition list*
    list of facts which must be true for action to be executed
  - a *delete list*
    list of facts that are no longer true after action is performed
  - an *add list*
    list of facts made true by executing the action

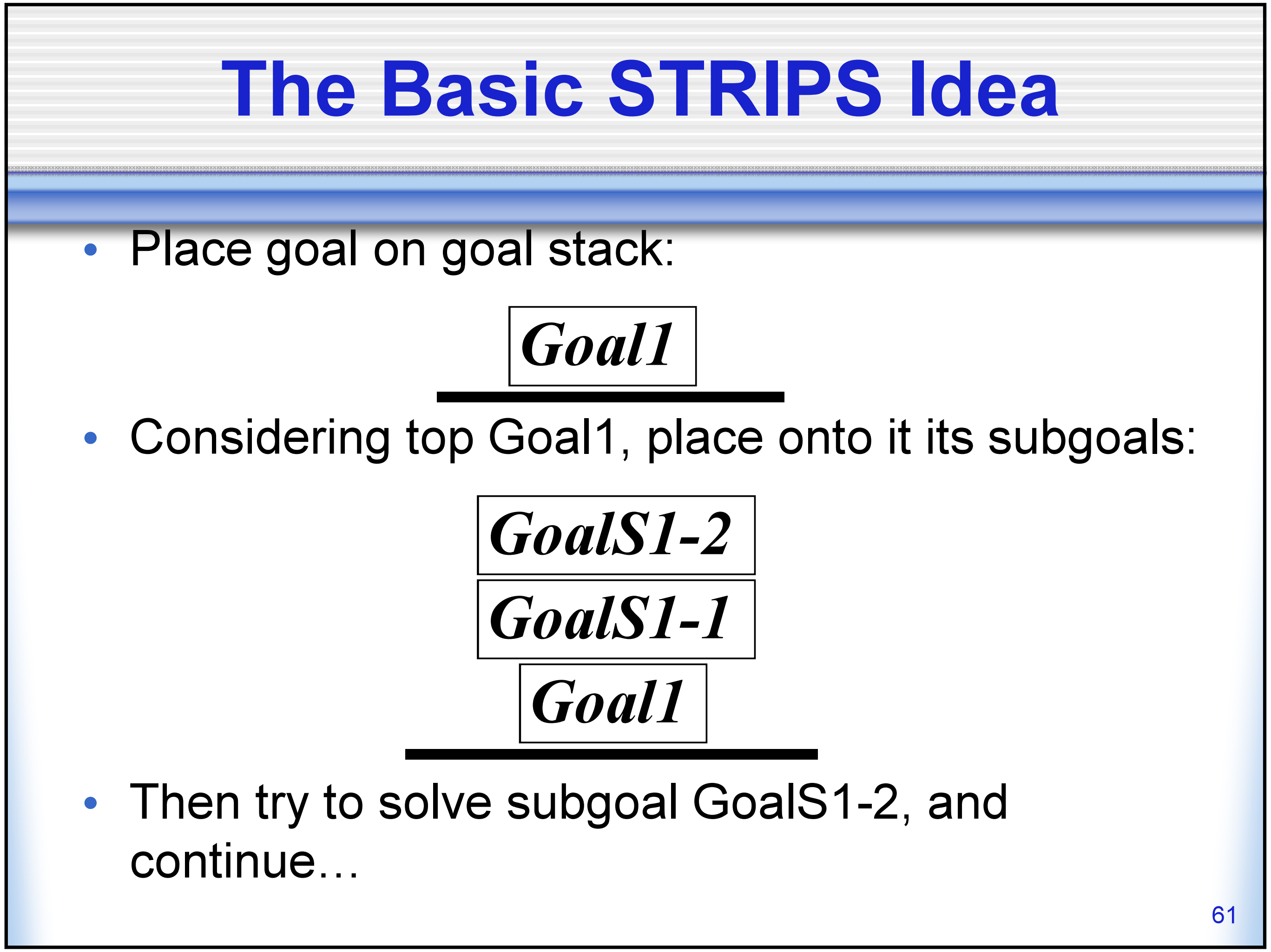
Each of these may contain *variables*

# The Blocks World Operators



- Example 1:
  The *stack* action occurs when the robot arm places the object $x$ it is holding is placed on top of object $y$.

| | $Stack(x, y)$ |
|---|---|
| pre | $Clear(y) \wedge Holding(x)$ |
| del | $Clear(y) \wedge Holding(x)$ |
| add | $ArmEmpty \wedge On(x, y)$ |



# The Basic STRIPS Idea

- Place goal on goal stack:

*Goal1*

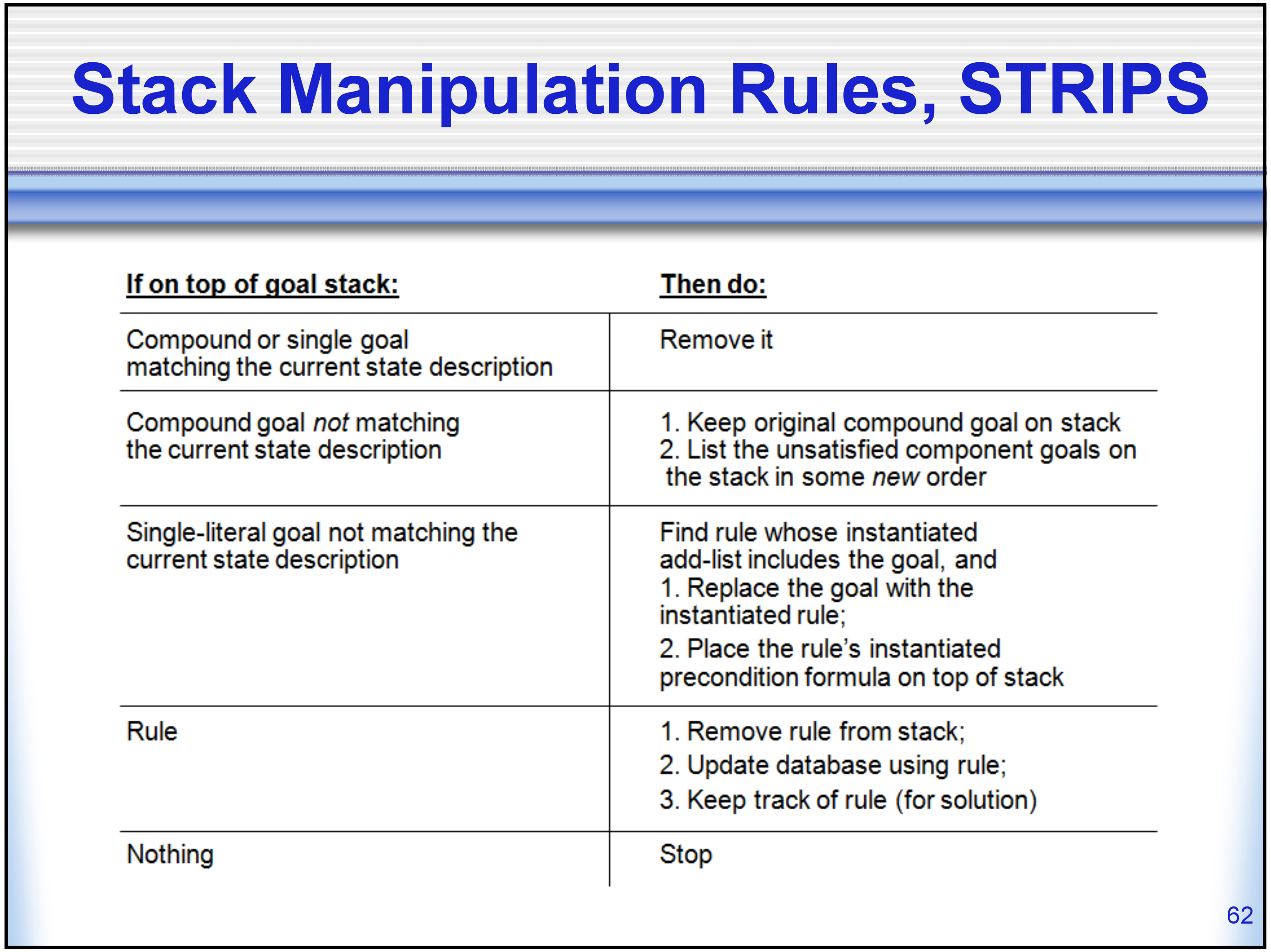
- Considering top Goal1, place onto it its subgoals:

*GoalS1-2*

*GoalS1-1*

*Goal1*

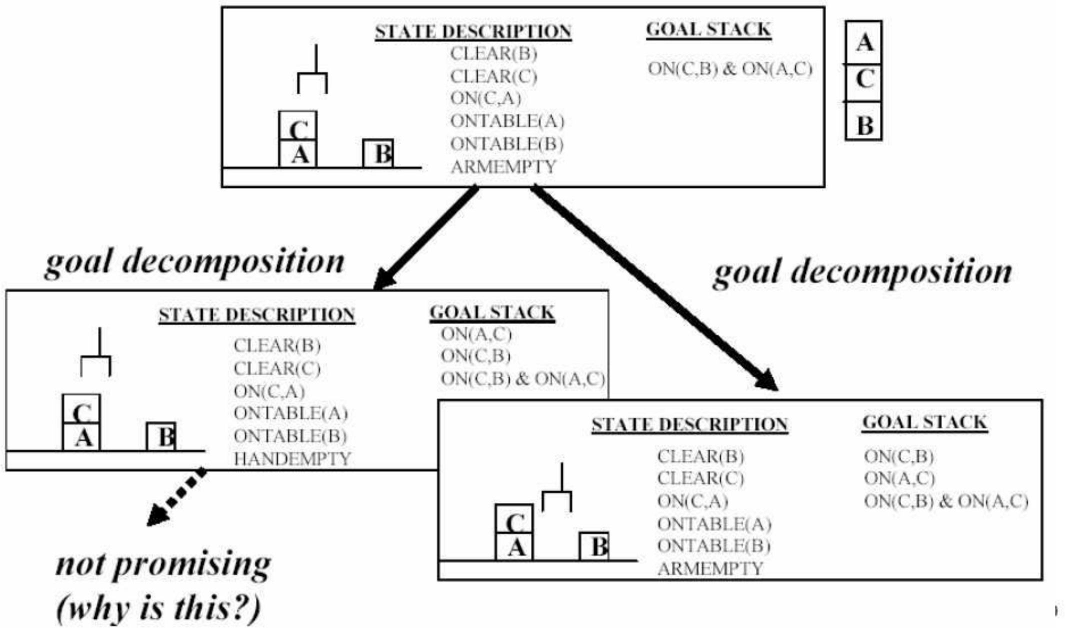
- Then try to solve subgoal GoalS1-2, and continue…



# Stack Manipulation Rules, STRIPS

| **If on top of goal stack:** | **Then do:** |
|---|---|
| Compound or single goal matching the current state description | Remove it |
| Compound goal *not* matching the current state description | 1. Keep original compound goal on stack<br>2. List the unsatisfied component goals on the stack in some *new* order |
| Single-literal goal not matching the current state description | Find rule whose instantiated add-list includes the goal, and<br>1. Replace the goal with the instantiated rule;<br>2. Place the rule's instantiated precondition formula on top of stack |
| Rule | 1. Remove rule from stack;<br>2. Update database using rule;<br>3. Keep track of rule (for solution) |
| Nothing | Stop |

# STRIPS in Action



STATE DESCRIPTION
CLEAR(B)
CLEAR(C)
ON(C,A)
ONTABLE(A)
ONTABLE(B)
ARMEMPTY

GOAL STACK
ON(C,B) & ON(A,C)

*goal decomposition*

STATE DESCRIPTION
CLEAR(B)
CLEAR(C)
ON(C,A)
ONTABLE(A)
ONTABLE(B)
HANDEMPTY

GOAL STACK
ON(A,C)
ON(C,B)
ON(C,B) & ON(A,C)

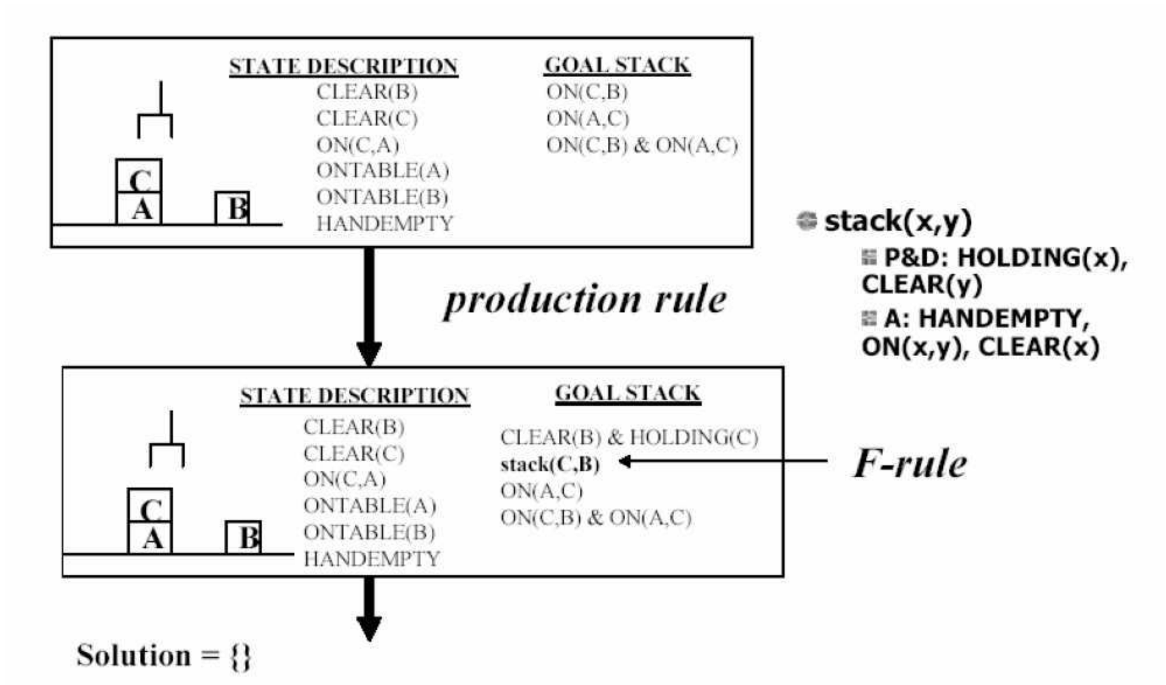
*not promising*
*(why is this?)*

*goal decomposition*

STATE DESCRIPTION
CLEAR(B)
CLEAR(C)
ON(C,A)
ONTABLE(A)
ONTABLE(B)
ARMEMPTY

GOAL STACK
ON(C,B)
ON(A,C)
ON(C,B) & ON(A,C)

# STRIPS in Action



**STATE DESCRIPTION**
CLEAR(B)
CLEAR(C)
ON(C,A)
ONTABLE(A)
ONTABLE(B)
HANDEMPTY

**GOAL STACK**
ON(C,B)
ON(A,C)
ON(C,B) & ON(A,C)

*production rule*

- stack(x,y)
  - P&D: HOLDING(x), CLEAR(y)
  - A: HANDEMPTY, ON(x,y), CLEAR(x)

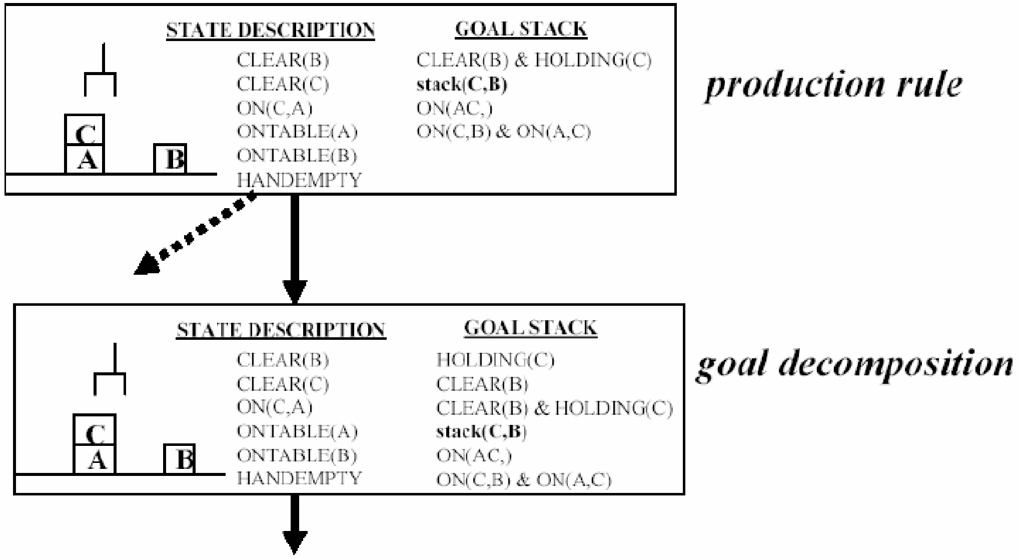
**STATE DESCRIPTION**
CLEAR(B)
CLEAR(C)
ON(C,A)
ONTABLE(A)
ONTABLE(B)
HANDEMPTY

**GOAL STACK**
CLEAR(B) & HOLDING(C)
**stack(C,B)**
ON(A,C)
ON(C,B) & ON(A,C)

*F-rule*

Solution = {}

# STRIPS in Action



STATE DESCRIPTION

CLEAR(B)
CLEAR(C)
ON(C,A)
ONTABLE(A)
ONTABLE(B)
HANDEMPTY

GOAL STACK

CLEAR(B) & HOLDING(C)
**stack(C,B)**
ON(AC,)
ON(C,B) & ON(A,C)

*production rule*

STATE DESCRIPTION

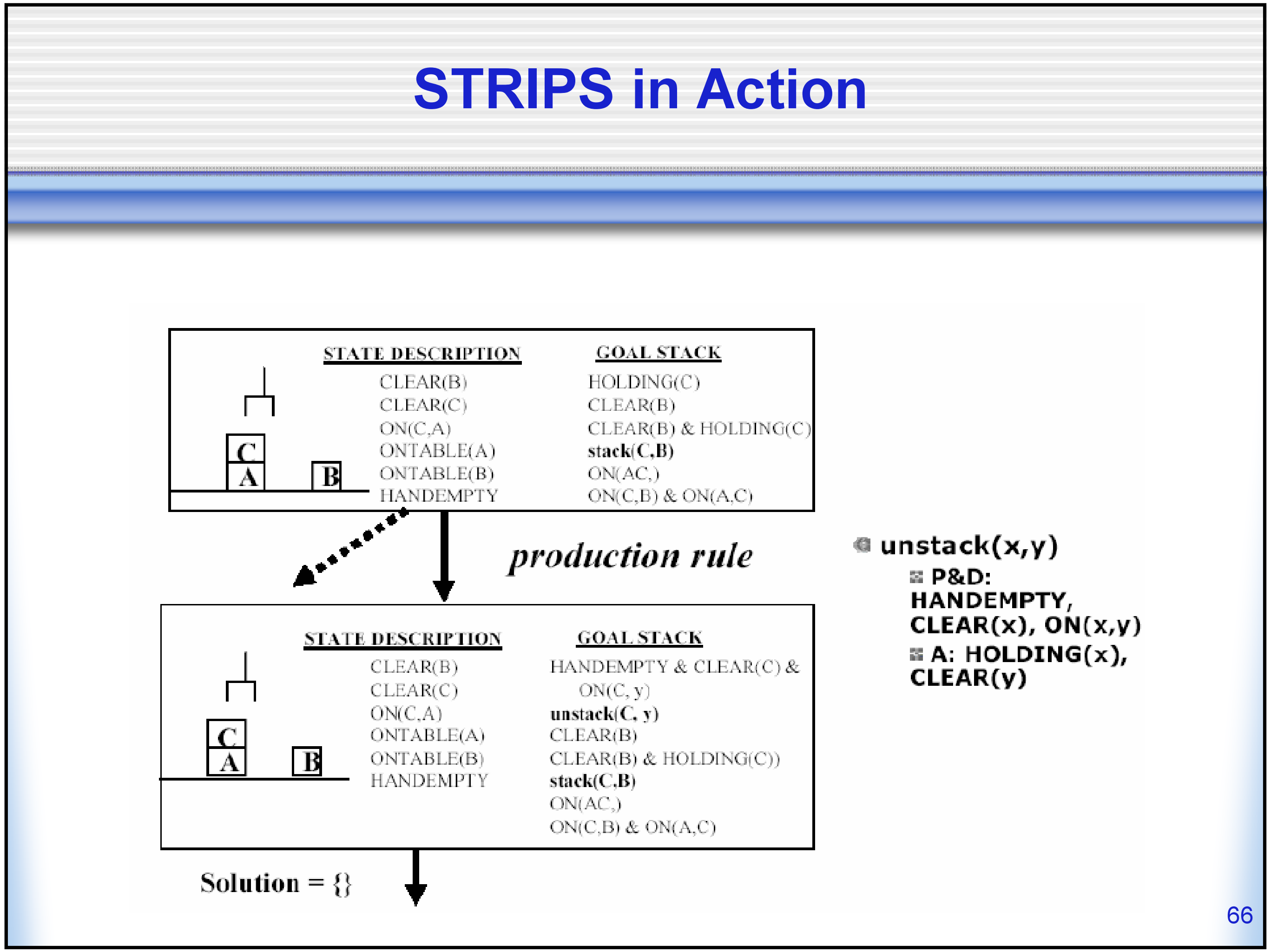
CLEAR(B)
CLEAR(C)
ON(C,A)
ONTABLE(A)
ONTABLE(B)
HANDEMPTY

GOAL STACK

HOLDING(C)
CLEAR(B)
CLEAR(B) & HOLDING(C)
**stack(C,B)**
ON(AC,)
ON(C,B) & ON(A,C)

*goal decomposition*

**Solution = {}**



# STRIPS in Action

STATE DESCRIPTION

CLEAR(B)
CLEAR(C)
ON(C,A)
ONTABLE(A)
ONTABLE(B)
HANDEMPTY

GOAL STACK

HOLDING(C)
CLEAR(B)
CLEAR(B) & HOLDING(C)
**stack(C,B)**
ON(AC,)
ON(C,B) & ON(A,C)

*production rule*

STATE DESCRIPTION

CLEAR(B)
CLEAR(C)
ON(C,A)
ONTABLE(A)
ONTABLE(B)
HANDEMPTY

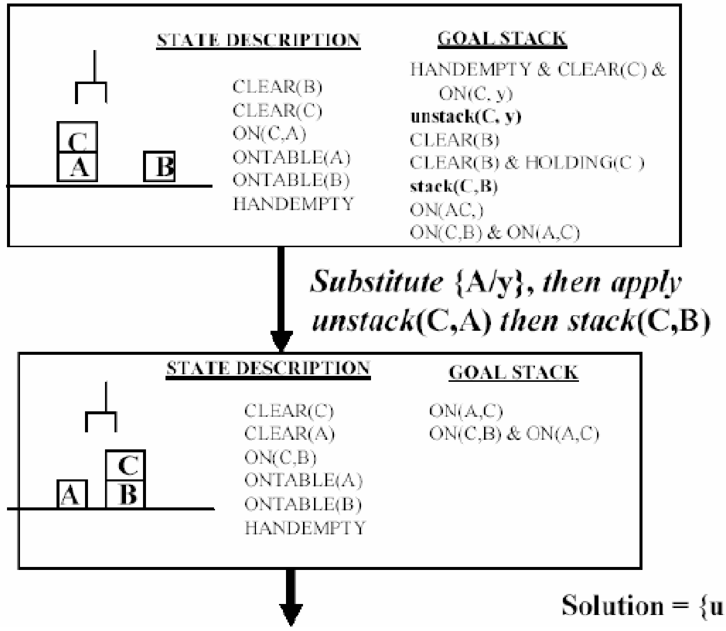
GOAL STACK

HANDEMPTY & CLEAR(C) &
ON(C, y)
**unstack(C, y)**
CLEAR(B)
CLEAR(B) & HOLDING(C))
**stack(C,B)**
ON(AC,)
ON(C,B) & ON(A,C)

Solution = {}

- unstack(x,y)
  - P&D: HANDEMPTY, CLEAR(x), ON(x,y)
  - A: HOLDING(x), CLEAR(y)

# STRIPS in Action



STATE DESCRIPTION

CLEAR(B)
CLEAR(C)
ON(C,A)
ONTABLE(A)
ONTABLE(B)
HANDEMPTY

GOAL STACK

HANDEMPTY & CLEAR(C) &
ON(C, y)
**unstack(C, y)**
CLEAR(B)
CLEAR(B) & HOLDING(C )
**stack(C,B)**
ON(AC,)
ON(C,B) & ON(A,C)

*Substitute* {A/y}, *then apply*
*unstack*(C,A) *then stack*(C,B)

STATE DESCRIPTION

CLEAR(C)
CLEAR(A)
ON(C,B)
ONTABLE(A)
ONTABLE(B)
HANDEMPTY

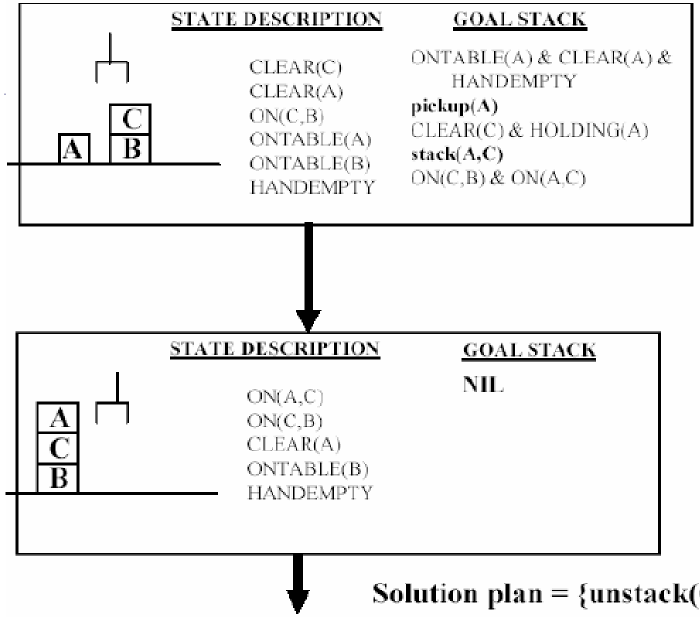
GOAL STACK

ON(A,C)
ON(C,B) & ON(A,C)

- **unstack(x,y)**
  - **P&D: HANDEMPTY, CLEAR(x), ON(x,y)**
  - **A: HOLDING(x), CLEAR(y)**
- **stack(x,y)**
  - **P&D: HOLDING(x), CLEAR(y)**
  - **A: HANDEMPTY, ON(x,y), CLEAR(x)**

Solution = {unstack(C,A), stack(C,B)}

# STRIPS in Action



STATE DESCRIPTION

CLEAR(C)
CLEAR(A)
ON(C,B)
ONTABLE(A)
ONTABLE(B)
HANDEMPTY

GOAL STACK

ONTABLE(A) & CLEAR(A) & HANDEMPTY
**pickup(A)**
CLEAR(C) & HOLDING(A)
**stack(A,C)**
ON(C,B) & ON(A,C)

STATE DESCRIPTION

ON(A,C)
ON(C,B)
CLEAR(A)
ONTABLE(B)
HANDEMPTY

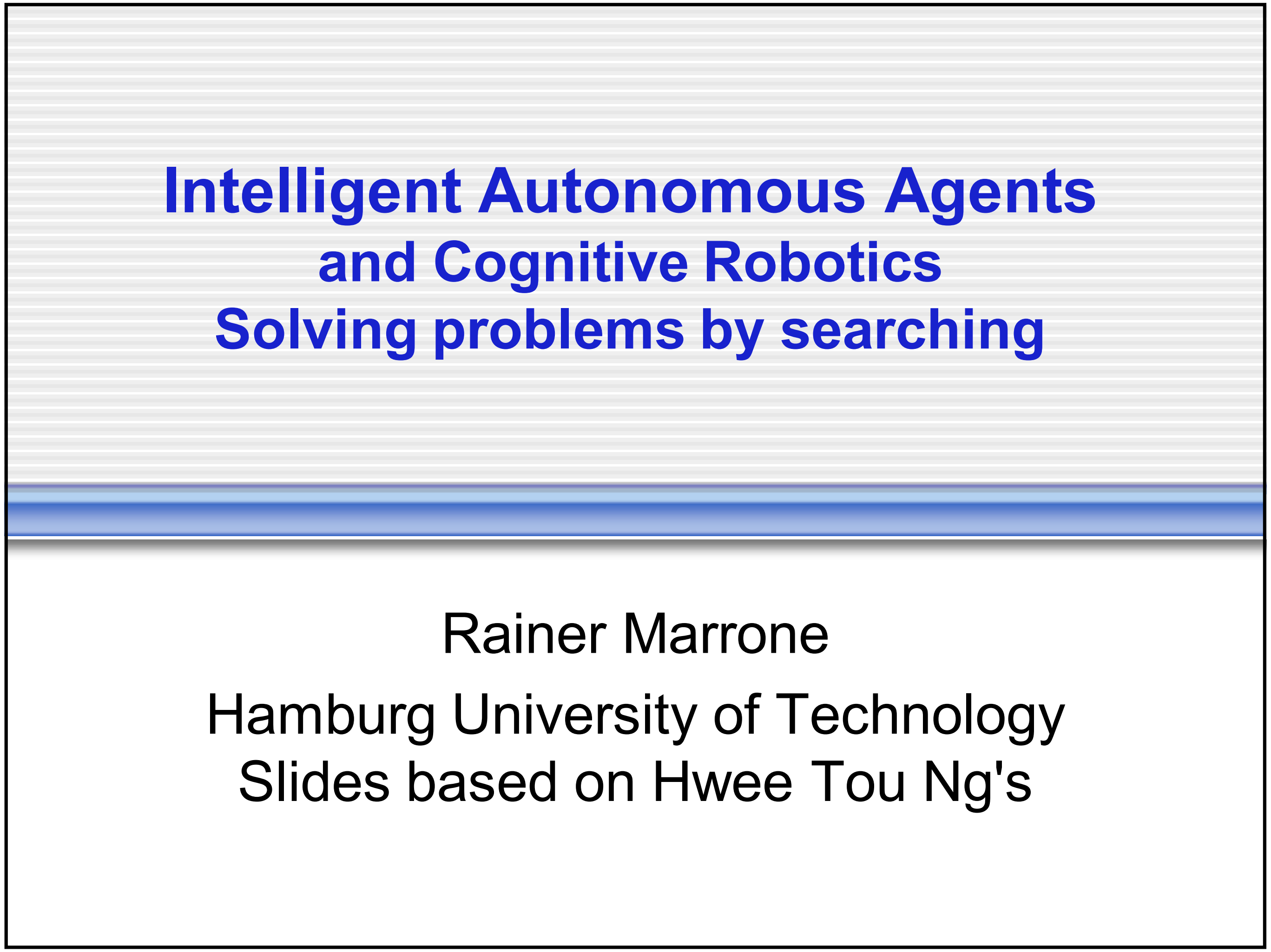
GOAL STACK

NIL

**Solution plan = {unstack(C,A), stack(C,B), pickup(A), stack(A,C)}**

*Apply pickup(A)*
*And then stack(A,C)*

- **pickup(x)**
  - **P&D: ONTABLE(x), CLEAR(x), HANDEMPTY**
  - **A: HOLDING(x)**
- **stack(x,y)**
  - **P&D: HOLDING(x), CLEAR(y)**
  - **A: HANDEMPTY, ON(x,y), CLEAR(x)**



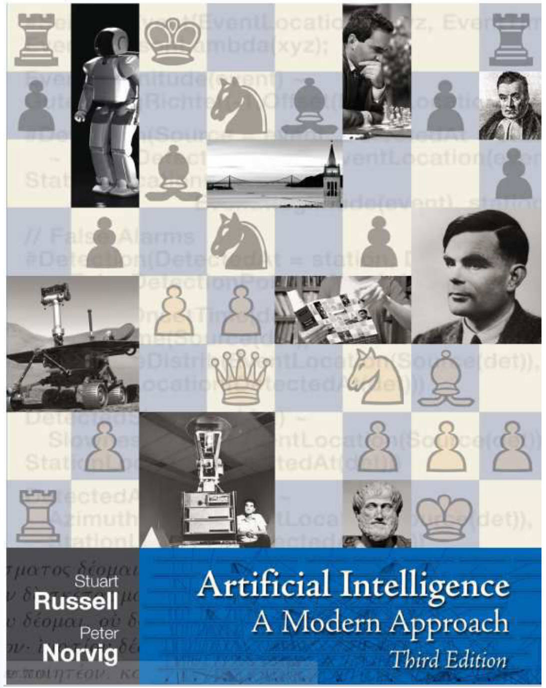
# Intelligent Autonomous Agents
## and Cognitive Robotics
## Solving problems by searching

Rainer Marrone

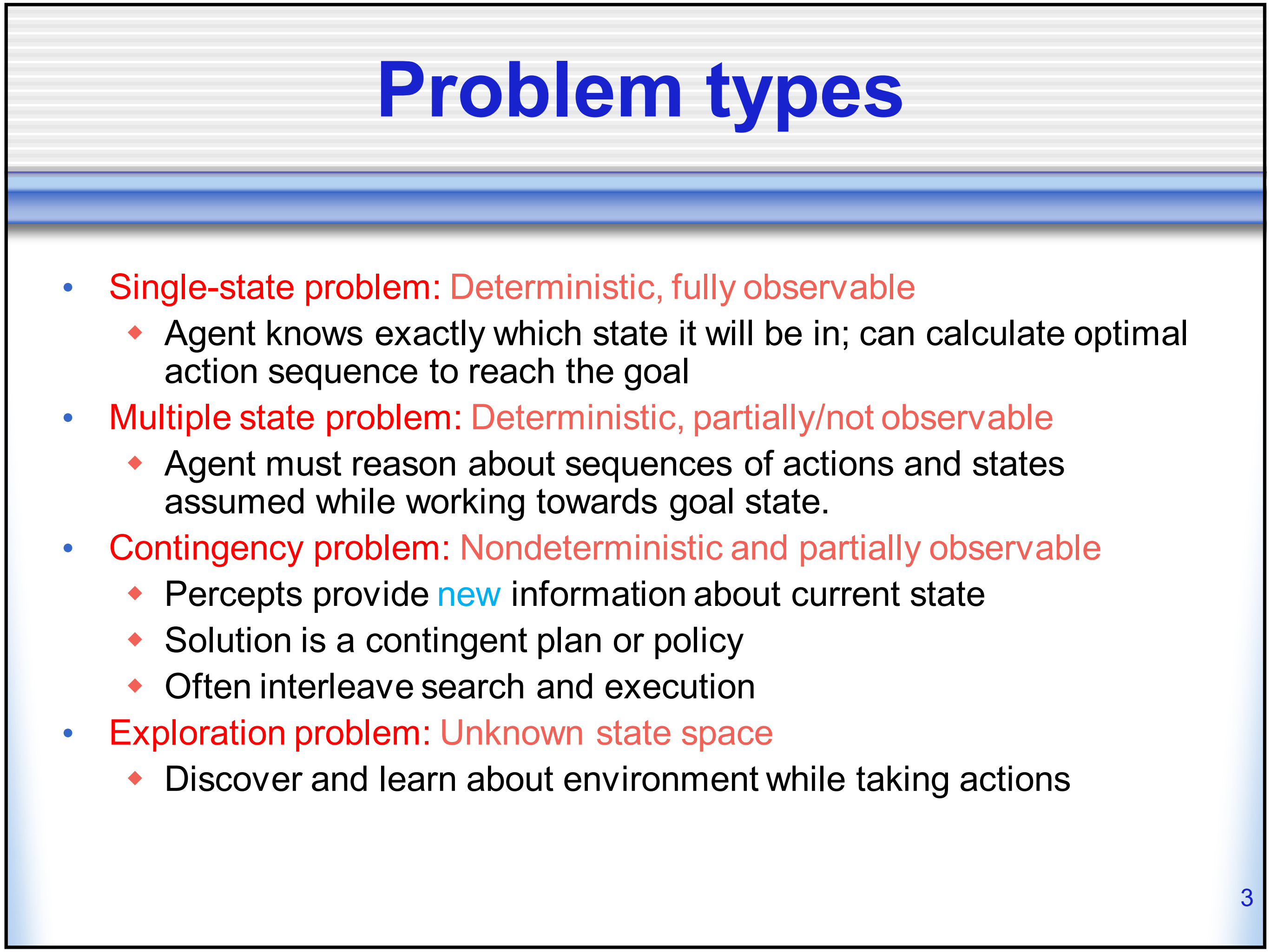
Hamburg University of Technology

Slides based on Hwee Tou Ng's

# Literature



- Chapter 3

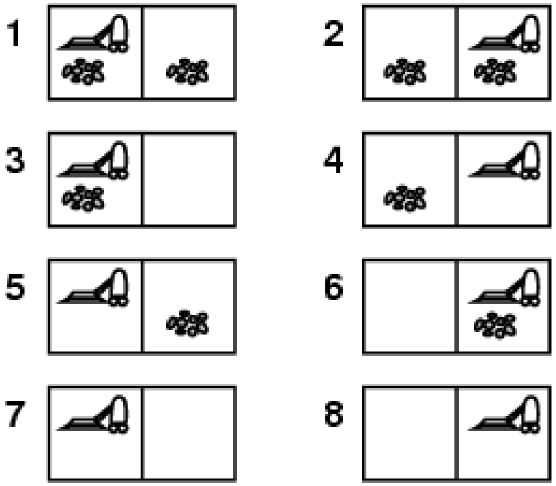


# Problem types

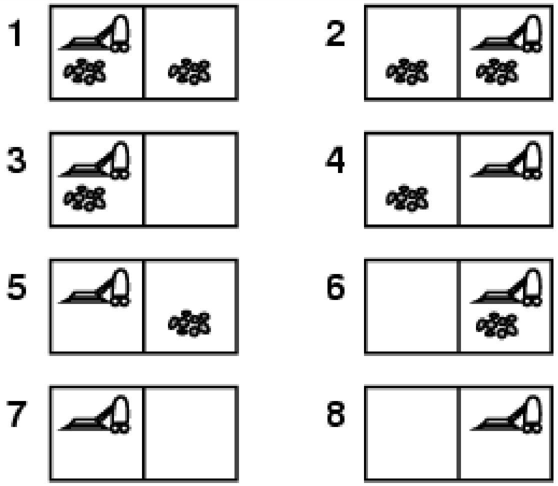
- Single-state problem: Deterministic, fully observable
  - Agent knows exactly which state it will be in; can calculate optimal action sequence to reach the goal
- Multiple state problem: Deterministic, partially/not observable
  - Agent must reason about sequences of actions and states assumed while working towards goal state.
- Contingency problem: Nondeterministic and partially observable
  - Percepts provide new information about current state
  - Solution is a contingent plan or policy
  - Often interleave search and execution
- Exploration problem: Unknown state space
  - Discover and learn about environment while taking actions

# Example: vacuum world

- Single-state, start in #5.
  Solution?
  *[Right, Suck]*
- Multiple-state, start in {*1,2,3,4,5,6,7,8*} e.g., *Right* goes to {*2,4,6,8*}
  Solution?
  *[Right,Suck,Left,Suck]*

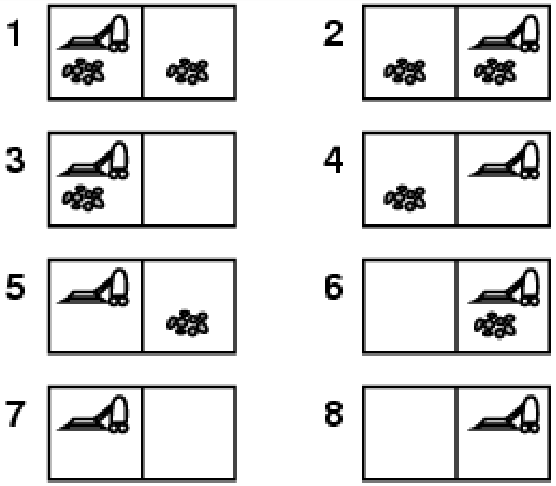


# Example: vacuum world

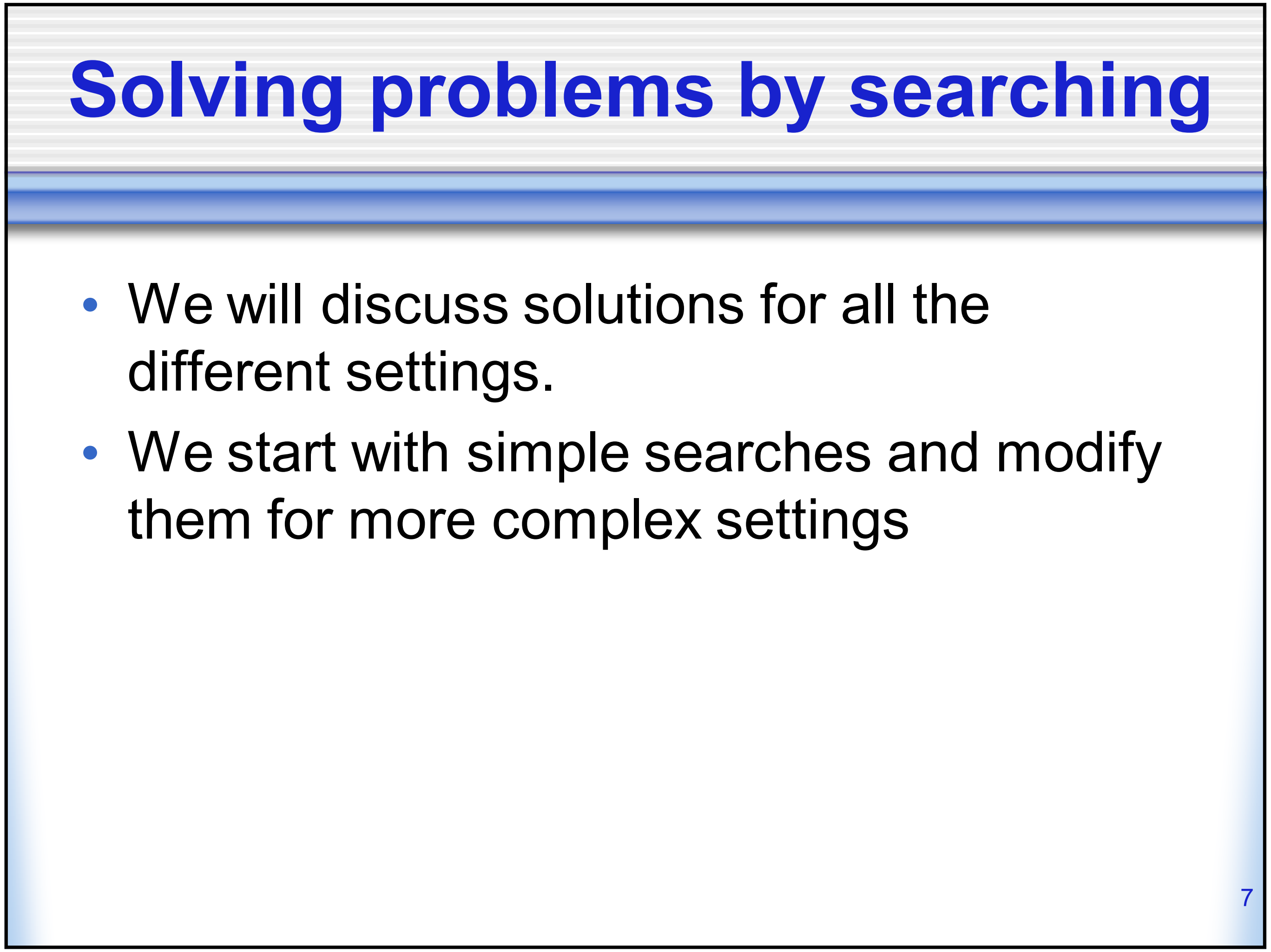


- Contingency
  - Nondeterministic: *Suck* may dirty a clean carpet
  - Partially observable: location, dirt at current location.
  - Percept: *[L, Clean],* i.e., start in #5 or #7
    Solution

# Example: vacuum world

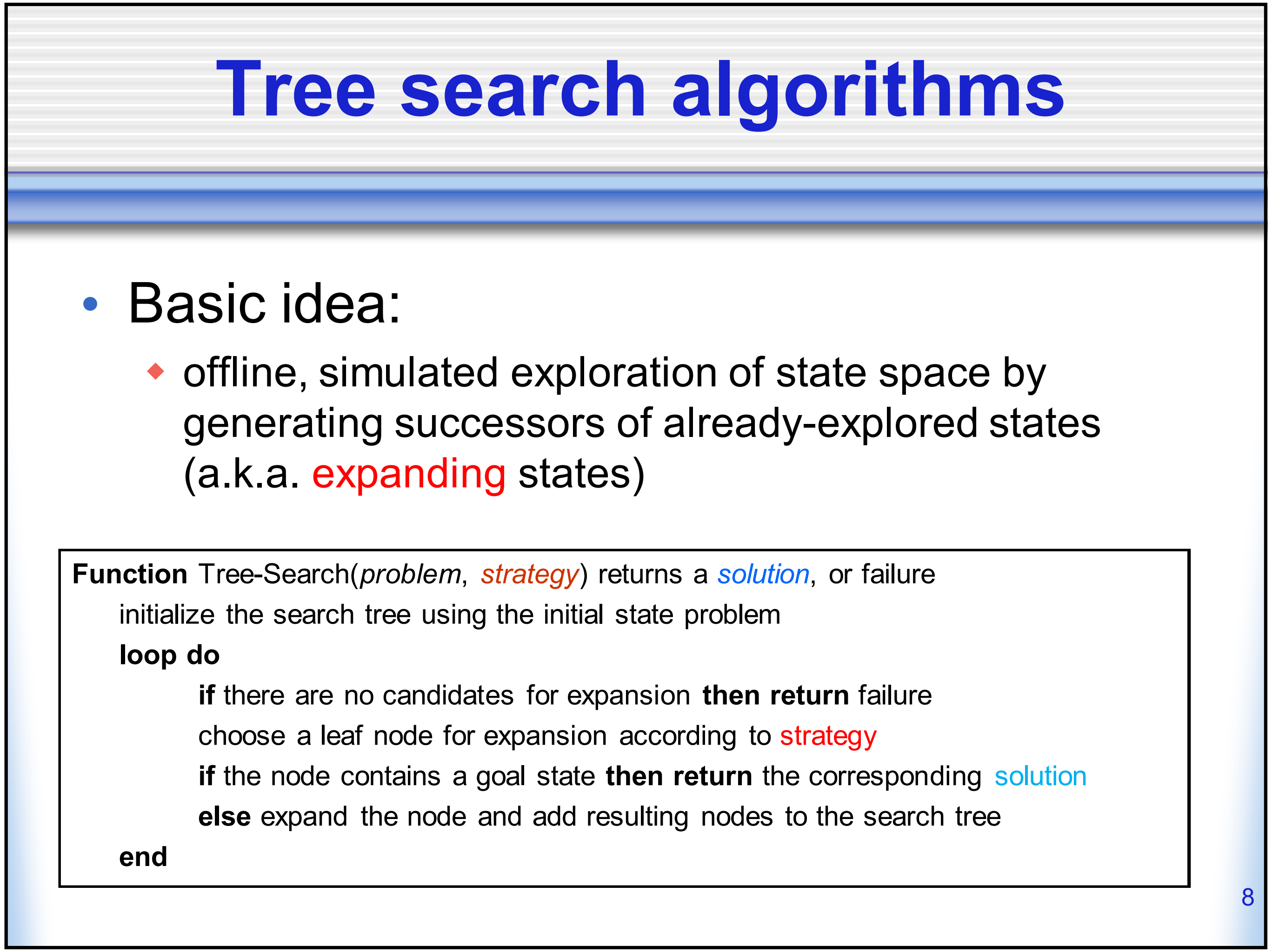


- Contingency
  - Nondeterministic: *Suck* may dirty a clean carpet
  - Partially observable: location, dirt at current location.
  - Percept: *[L, Clean],* i.e., start in #5 or #7
    Solution? *[Right, **if** dirt **then** Suck]*



# Solving problems by searching

- We will discuss solutions for all the different settings.
- We start with simple searches and modify them for more complex settings



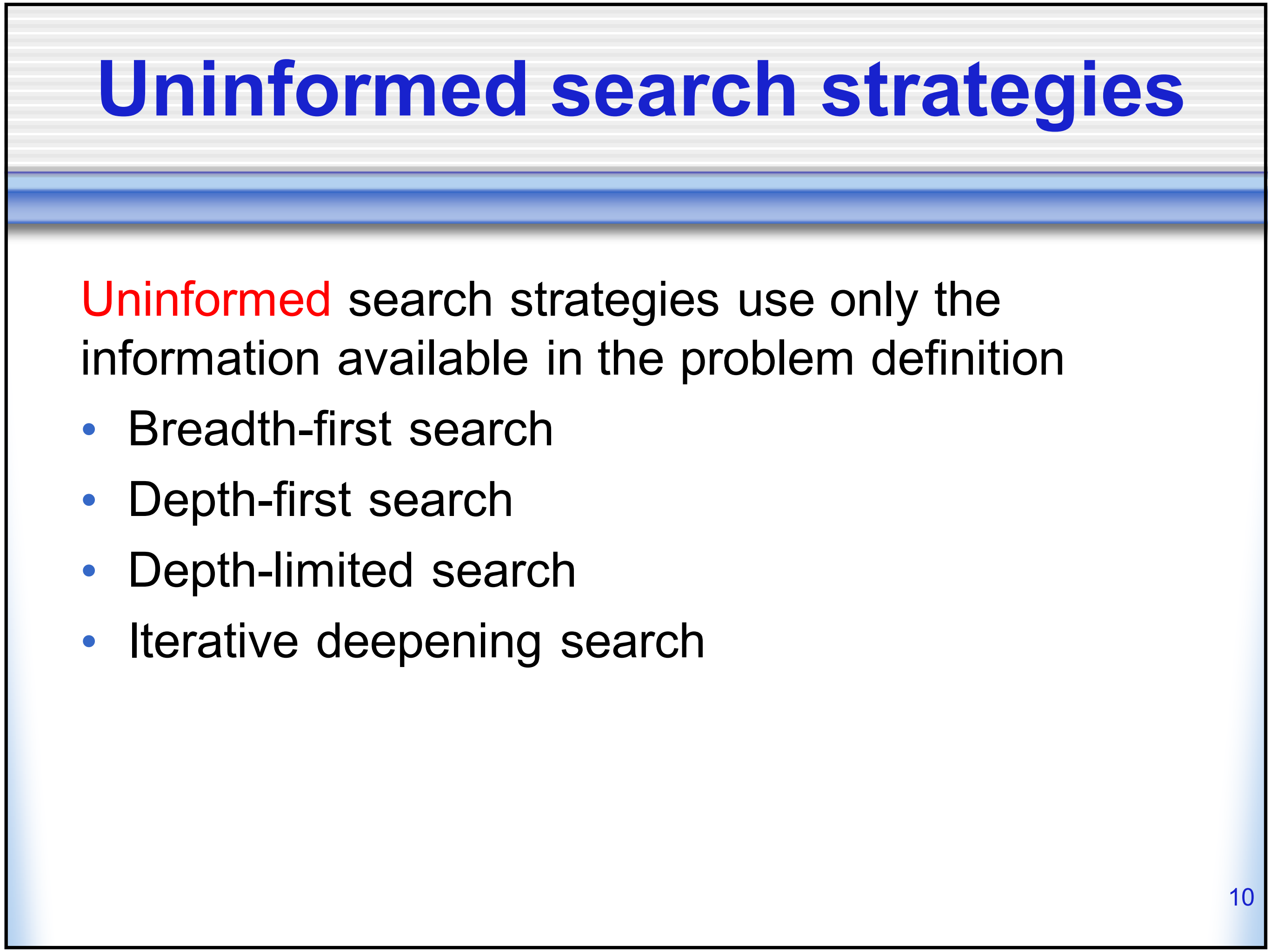
# Tree search algorithms

- Basic idea:
  - offline, simulated exploration of state space by generating successors of already-explored states (a.k.a. expanding states)

```
Function Tree-Search(problem, strategy) returns a solution, or failure
    initialize the search tree using the initial state problem
    loop do
        if there are no candidates for expansion then return failure
        choose a leaf node for expansion according to strategy
        if the node contains a goal state then return the corresponding solution
        else expand the node and add resulting nodes to the search tree
    end
```

# Measuring search performance

- A search strategy is defined by picking the order of node expansion
- Strategies are evaluated along the following dimensions:
  - completeness: does it always find a solution if one exists?
  - time complexity: number of nodes generated
  - space complexity: maximum number of nodes in memory
  - optimality: does it always find a least-cost solution?
- Time and space complexity are measured in terms of
  - *b:* maximum branching factor of the search tree
  - *d:* depth of the least-cost solution
  - *m*: maximum depth of the state space (may be ∞)



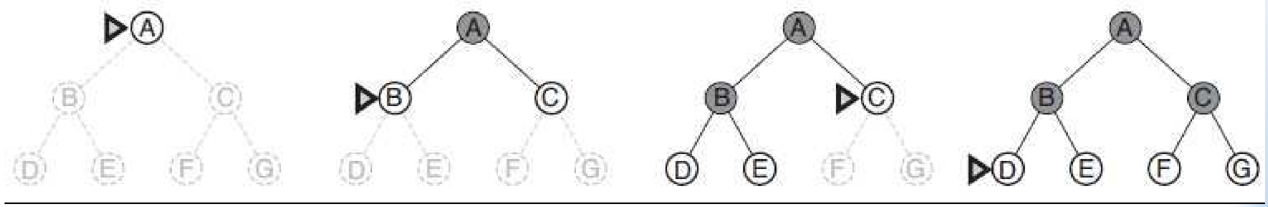
# Uninformed search strategies

Uninformed search strategies use only the information available in the problem definition

- Breadth-first search
- Depth-first search
- Depth-limited search
- Iterative deepening search

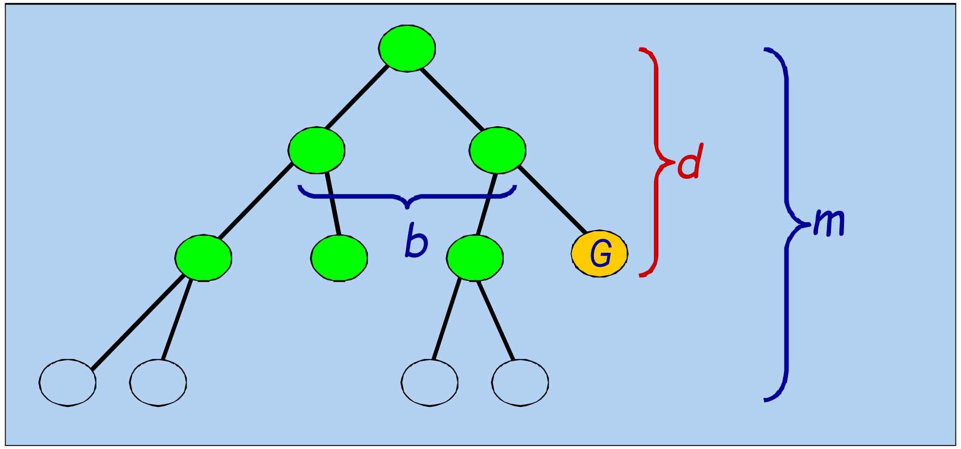
# Breadth-first search

- Expand shallowest unexpanded node
- Implementation:
  - *fringe* is a FIFO queue, i.e., new successors go at end

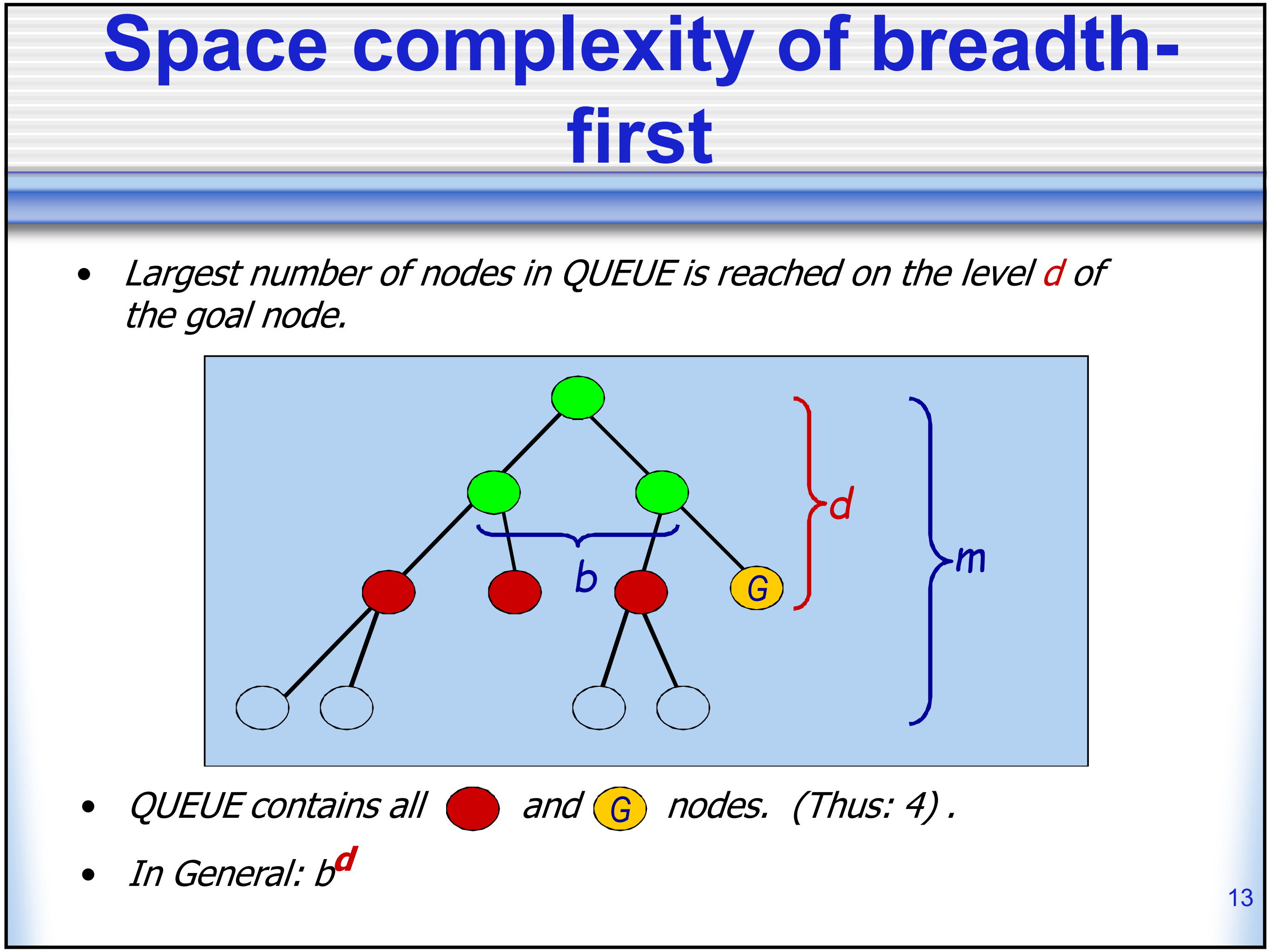


# Time complexity of breadth-first search

- *If a goal node is found on depth d of the tree, all nodes up till that depth are created.*



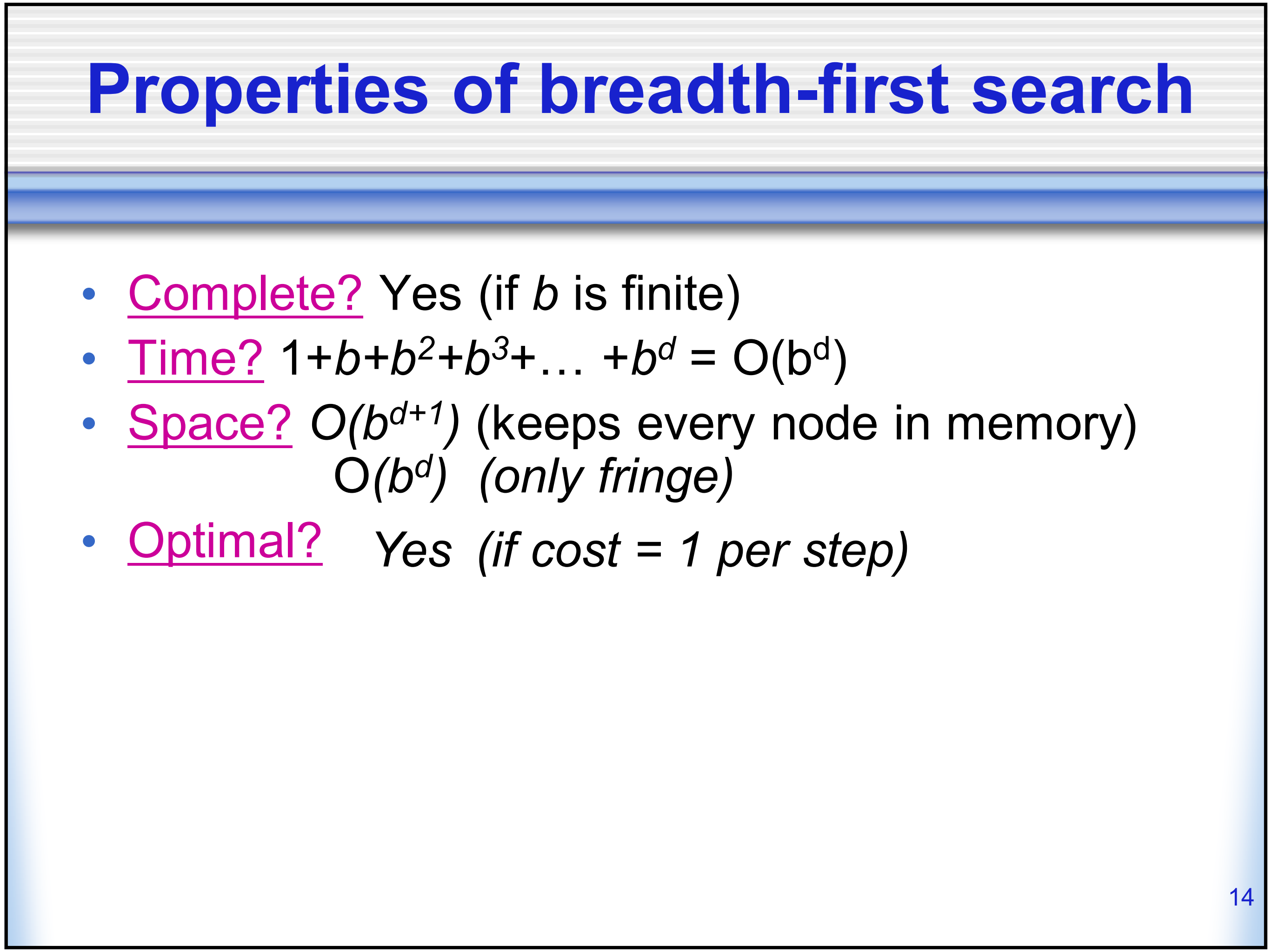
- *Thus:* $O(b^d)$



# Space complexity of breadth-first

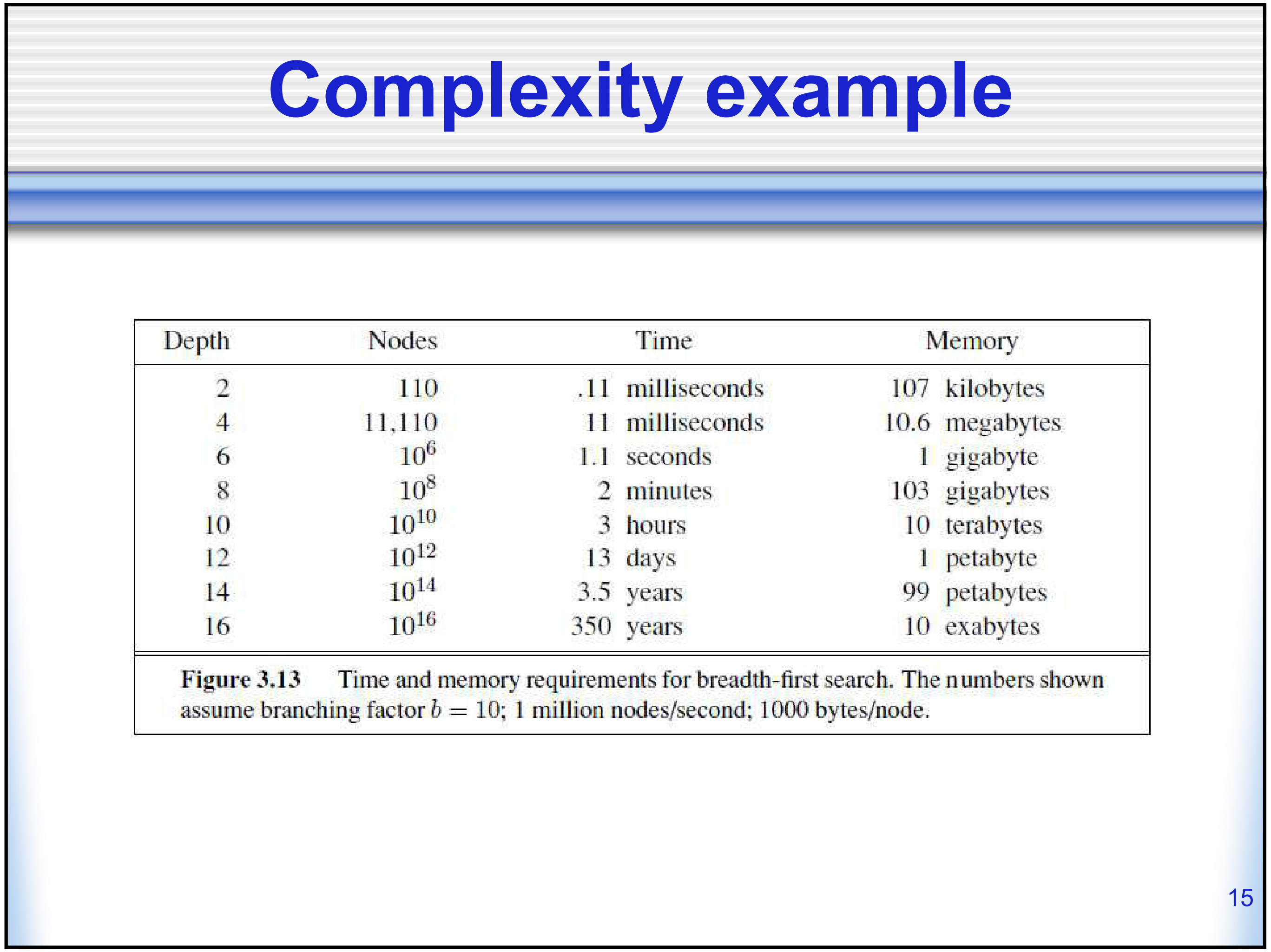
- *Largest number of nodes in QUEUE is reached on the level d of the goal node.*

- *QUEUE contains all ● and G nodes. (Thus: 4) .*
- *In General: $b^d$*



# Properties of breadth-first search

- Complete? Yes (if *b* is finite)
- Time? 1+$b$+$b^2$+$b^3$+… +$b^d$ = O(b$^d$)
- Space? $O(b^{d+1})$ (keeps every node in memory)<br>
  O($b^d$)  *(only fringe)*
- Optimal?  *Yes  (if cost = 1 per step)*



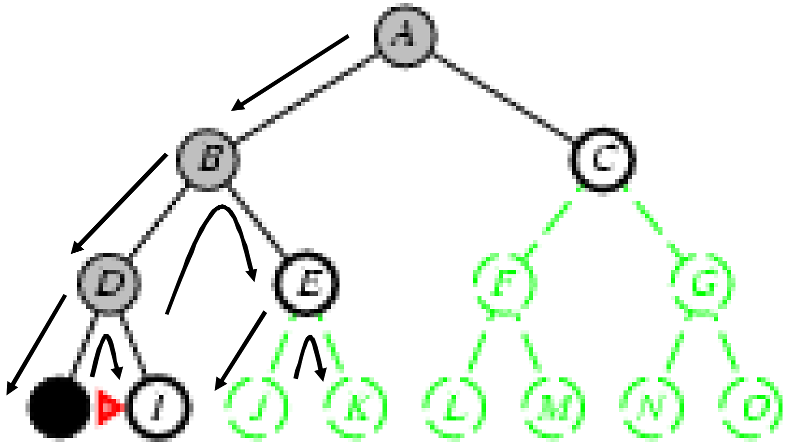
# Complexity example

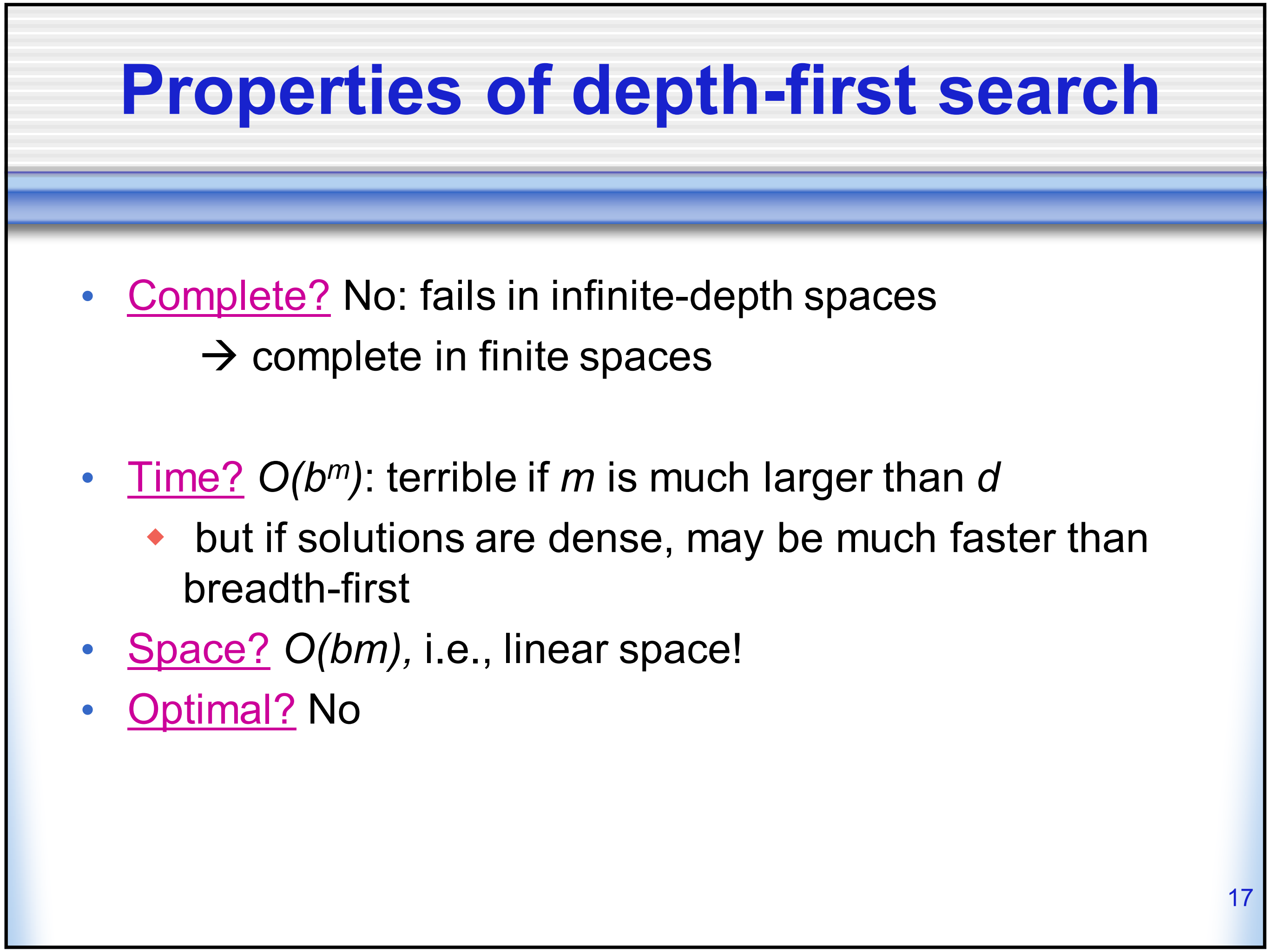
| Depth | Nodes | Time | Memory |
|---|---|---|---|
| 2 | 110 | .11 milliseconds | 107 kilobytes |
| 4 | 11,110 | 11 milliseconds | 10.6 megabytes |
| 6 | $10^6$ | 1.1 seconds | 1 gigabyte |
| 8 | $10^8$ | 2 minutes | 103 gigabytes |
| 10 | $10^{10}$ | 3 hours | 10 terabytes |
| 12 | $10^{12}$ | 13 days | 1 petabyte |
| 14 | $10^{14}$ | 3.5 years | 99 petabytes |
| 16 | $10^{16}$ | 350 years | 10 exabytes |

**Figure 3.13** Time and memory requirements for breadth-first search. The numbers shown assume branching factor $b = 10$; 1 million nodes/second; 1000 bytes/node.

# Depth-first search

- Expand deepest unexpanded node





# Properties of depth-first search

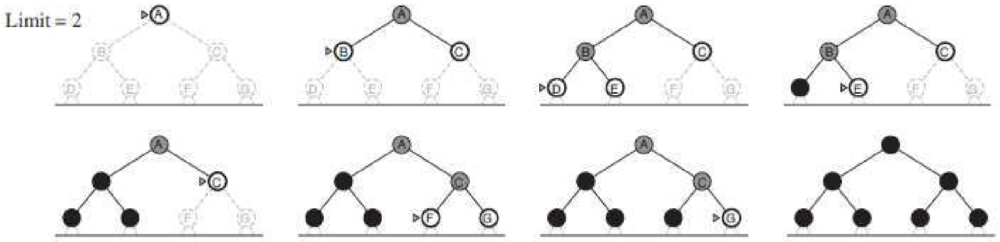
- Complete? No: fails in infinite-depth spaces

  → complete in finite spaces

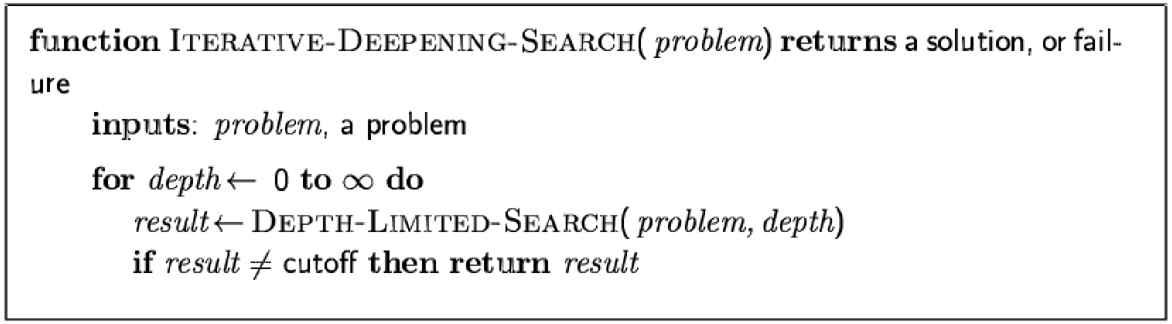
- Time? $O(b^m)$: terrible if $m$ is much larger than $d$
  - but if solutions are dense, may be much faster than breadth-first
- Space? $O(bm)$, i.e., linear space!
- Optimal? No

# Depth-limited search

- depth-first search with depth limit *l*, i.e., nodes at depth *l* have no successors
- Solves infinite path problem
- Incomplete if *l*<d (shallowest goal node)
- Nonoptimal if *l*>d

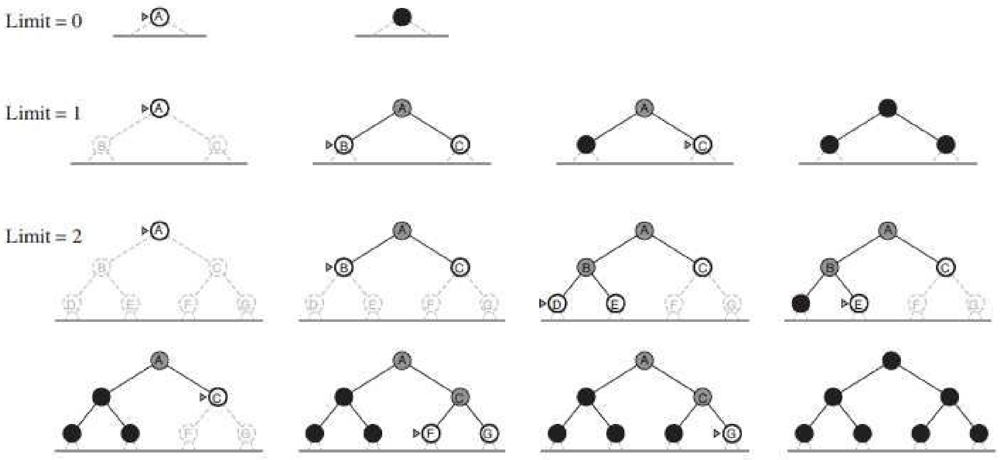


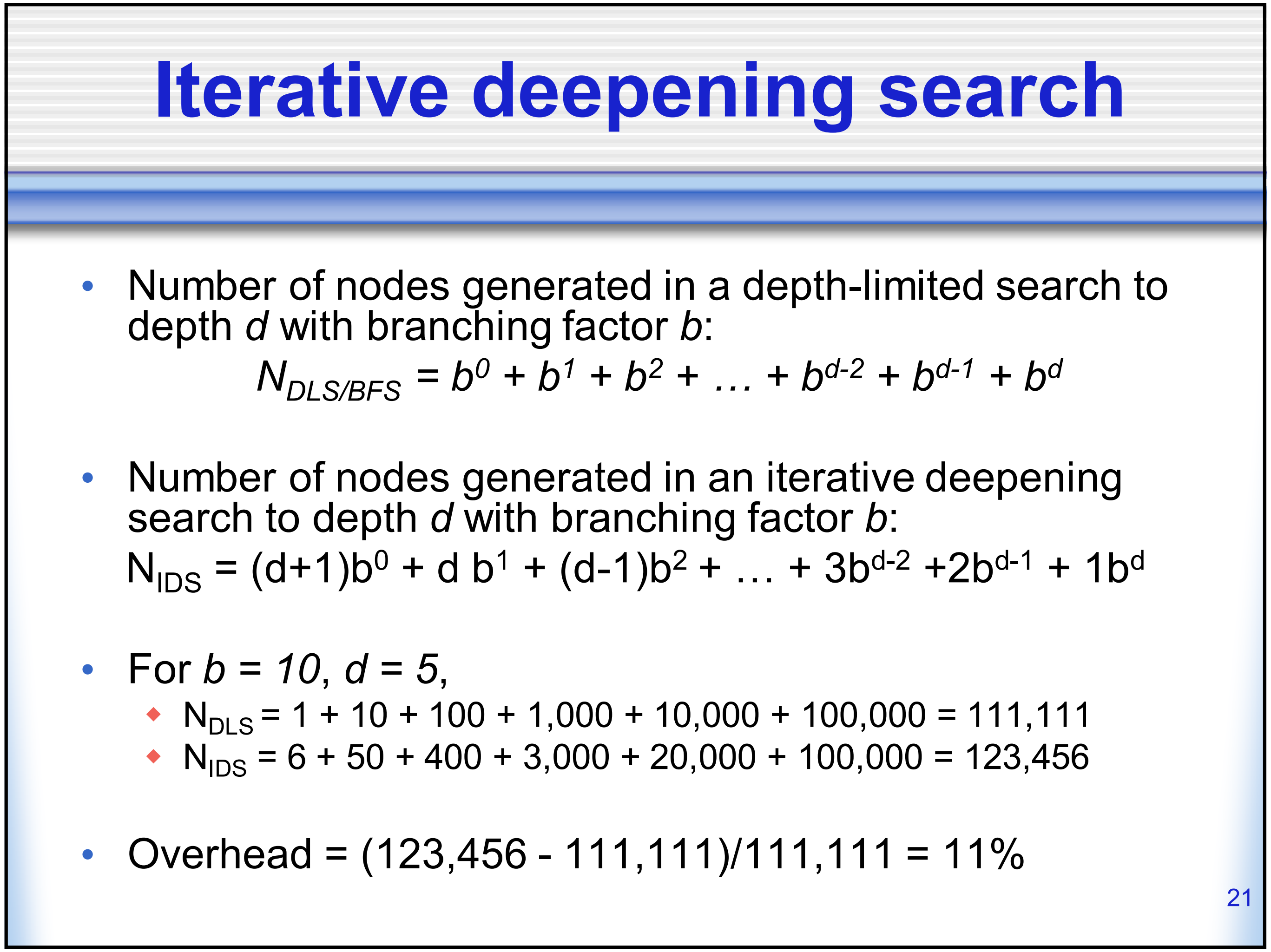
# Iterative deepening search



```
function ITERATIVE-DEEPENING-SEARCH(problem) returns a solution, or failure
    inputs: problem, a problem
    for depth ← 0 to ∞ do
        result ← DEPTH-LIMITED-SEARCH(problem, depth)
        if result ≠ cutoff then return result
```

# Iterative deepening search





# Iterative deepening search

- Number of nodes generated in a depth-limited search to depth *d* with branching factor *b*:

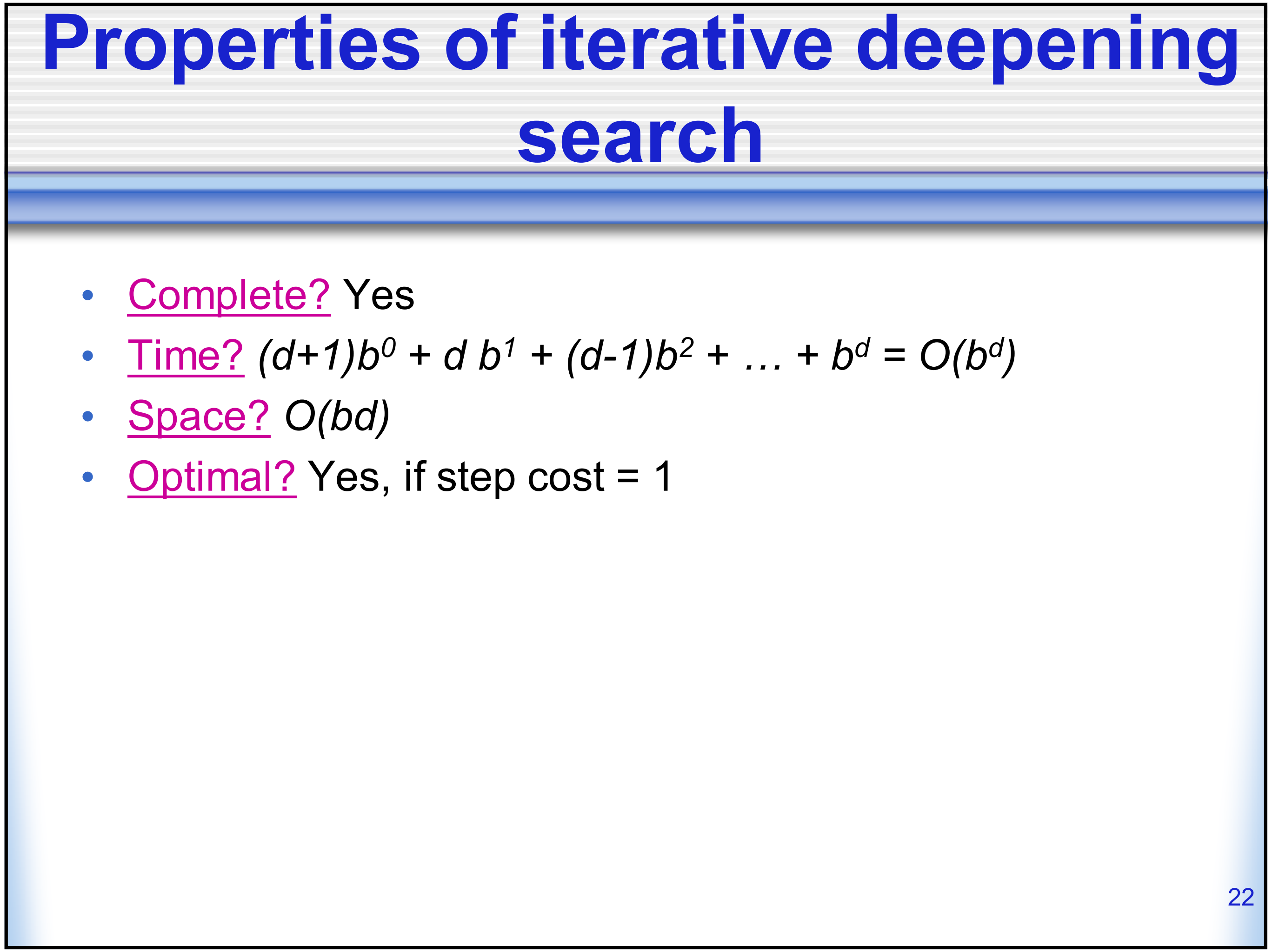
$$N_{DLS/BFS} = b^0 + b^1 + b^2 + \ldots + b^{d-2} + b^{d-1} + b^d$$

- Number of nodes generated in an iterative deepening search to depth *d* with branching factor *b*:

  $N_{\mathrm{IDS}} = (\mathrm{d}+1)\mathrm{b}^0 + \mathrm{d}\ \mathrm{b}^1 + (\mathrm{d}-1)\mathrm{b}^2 + \ldots + 3\mathrm{b}^{\mathrm{d}-2} + 2\mathrm{b}^{\mathrm{d}-1} + 1\mathrm{b}^\mathrm{d}$

- For *b = 10*, *d = 5*,
  - $N_{\mathrm{DLS}}$ = 1 + 10 + 100 + 1,000 + 10,000 + 100,000 = 111,111
  - $N_{\mathrm{IDS}}$ = 6 + 50 + 400 + 3,000 + 20,000 + 100,000 = 123,456

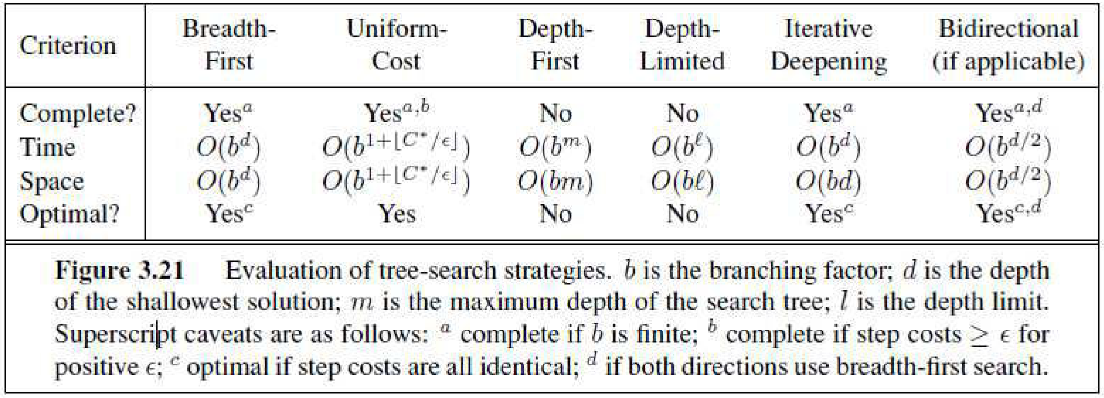
- Overhead = (123,456 - 111,111)/111,111 = 11%



# Properties of iterative deepening search

- Complete? Yes
- Time? $(d+1)b^0 + d\,b^1 + (d-1)b^2 + \ldots + b^d = O(b^d)$
- Space? $O(bd)$
- Optimal? Yes, if step cost = 1

# Summary of algorithms

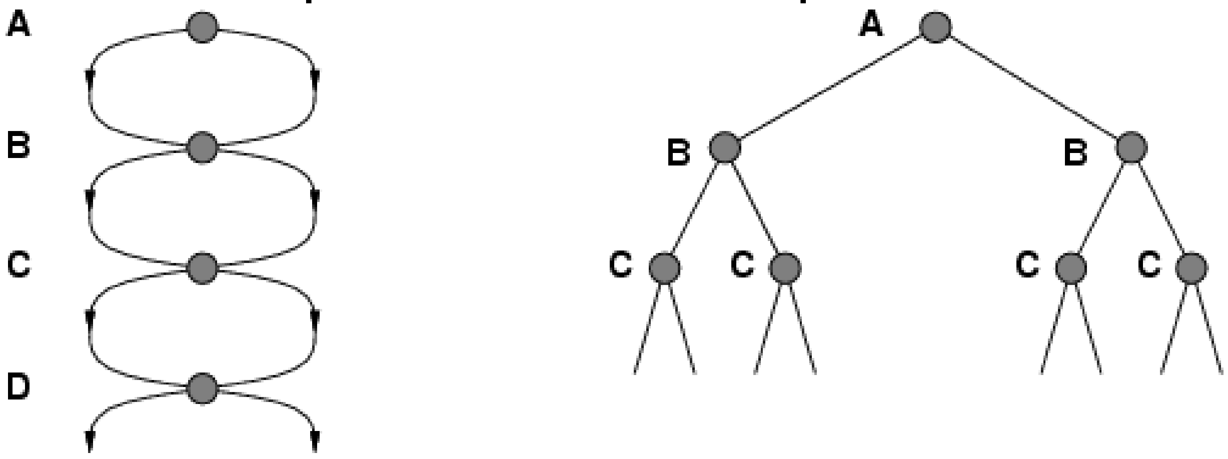


| Criterion | Breadth-First | Uniform-Cost | Depth-First | Depth-Limited | Iterative Deepening | Bidirectional (if applicable) |
|---|---|---|---|---|---|---|
| Complete? | Yes$^{a}$ | Yes$^{a,b}$ | No | No | Yes$^{a}$ | Yes$^{a,d}$ |
| Time | $O(b^d)$ | $O(b^{1+\lfloor C^*/\epsilon \rfloor})$ | $O(b^m)$ | $O(b^\ell)$ | $O(b^d)$ | $O(b^{d/2})$ |
| Space | $O(b^d)$ | $O(b^{1+\lfloor C^*/\epsilon \rfloor})$ | $O(bm)$ | $O(b\ell)$ | $O(bd)$ | $O(b^{d/2})$ |
| Optimal? | Yes$^{c}$ | Yes | No | No | Yes$^{c}$ | Yes$^{c,d}$ |

**Figure 3.21** Evaluation of tree-search strategies. $b$ is the branching factor; $d$ is the depth of the shallowest solution; $m$ is the maximum depth of the search tree; $l$ is the depth limit. Superscript caveats are as follows: $^{a}$ complete if $b$ is finite; $^{b}$ complete if step costs $\geq \epsilon$ for positive $\epsilon$; $^{c}$ optimal if step costs are all identical; $^{d}$ if both directions use breadth-first search.

# Repeated states

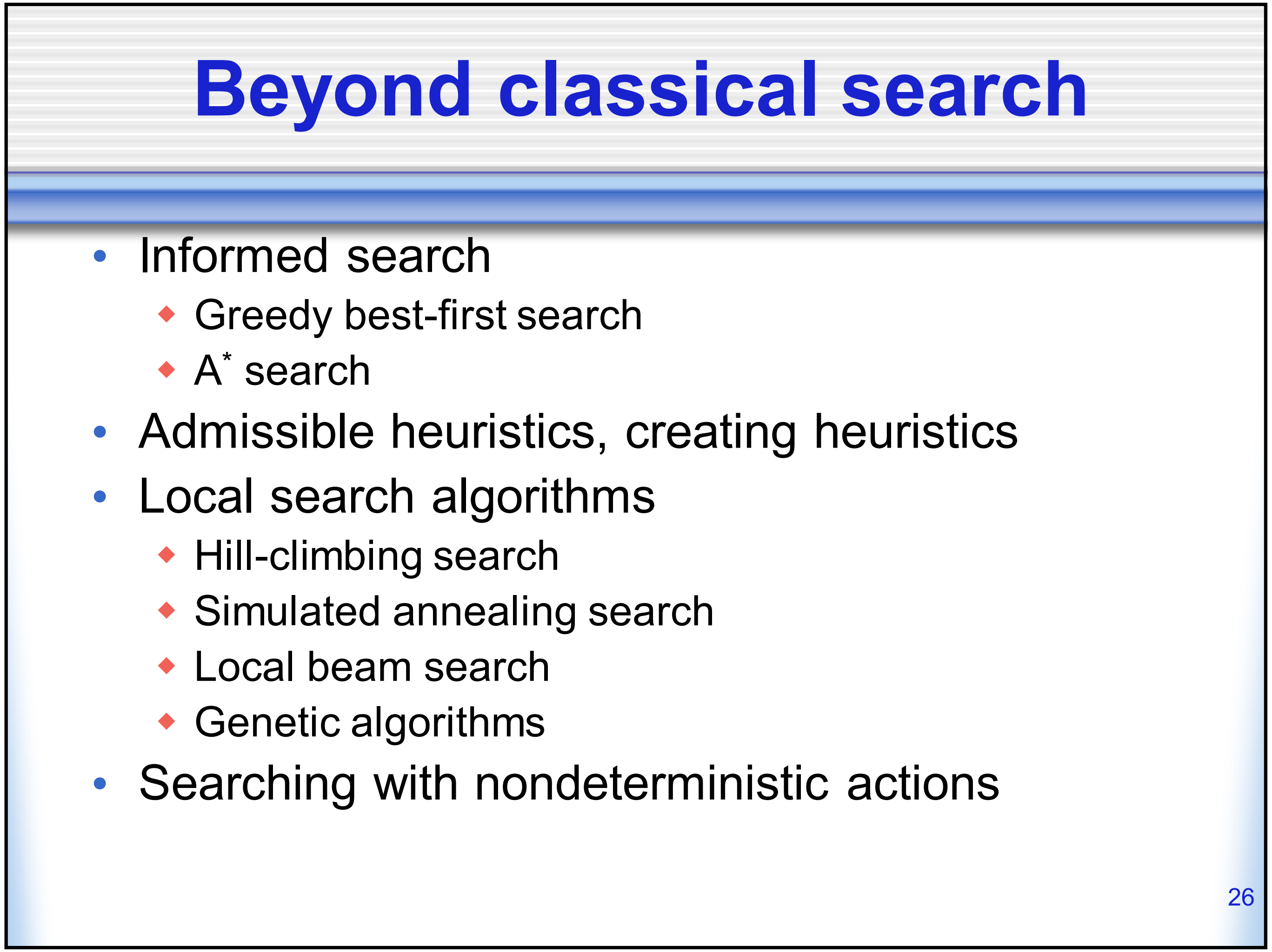
- Failure to detect repeated states can turn a linear problem into an exponential one!



# Graph search

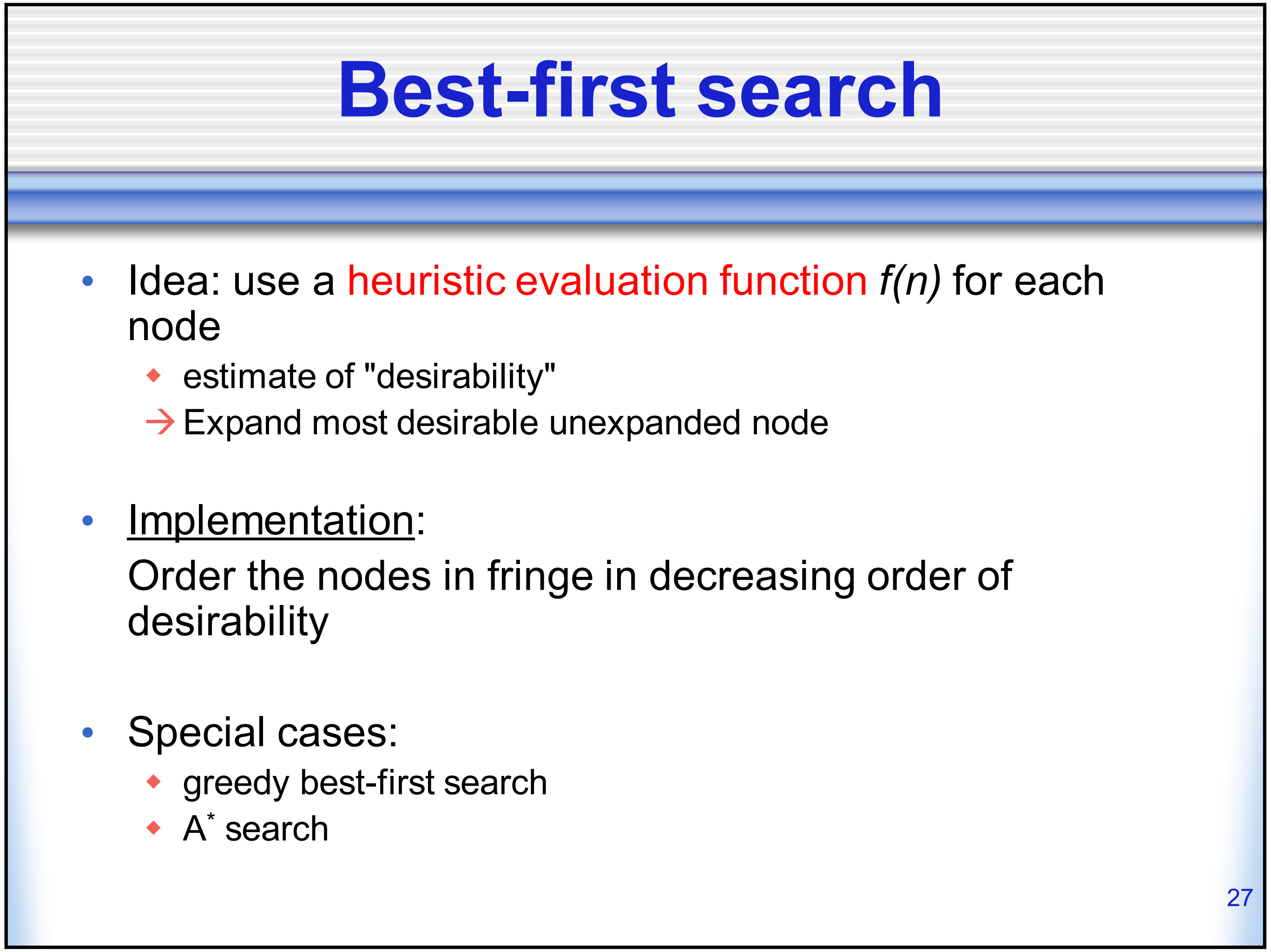
```
function GRAPH-SEARCH( problem, fringe) returns a solution, or failure
    closed ← an empty set
    fringe ← INSERT(MAKE-NODE(INITIAL-STATE[problem]), fringe)
    loop do
        if fringe is empty then return failure
        node ← REMOVE-FRONT(fringe)
        if GOAL-TEST[problem](STATE[node]) then return SOLUTION(node)
        if STATE[node] is not in closed then
            add STATE[node] to closed
            fringe ← INSERTALL(EXPAND(node, problem), fringe)
```

*Remember nodes visited*



# Beyond classical search

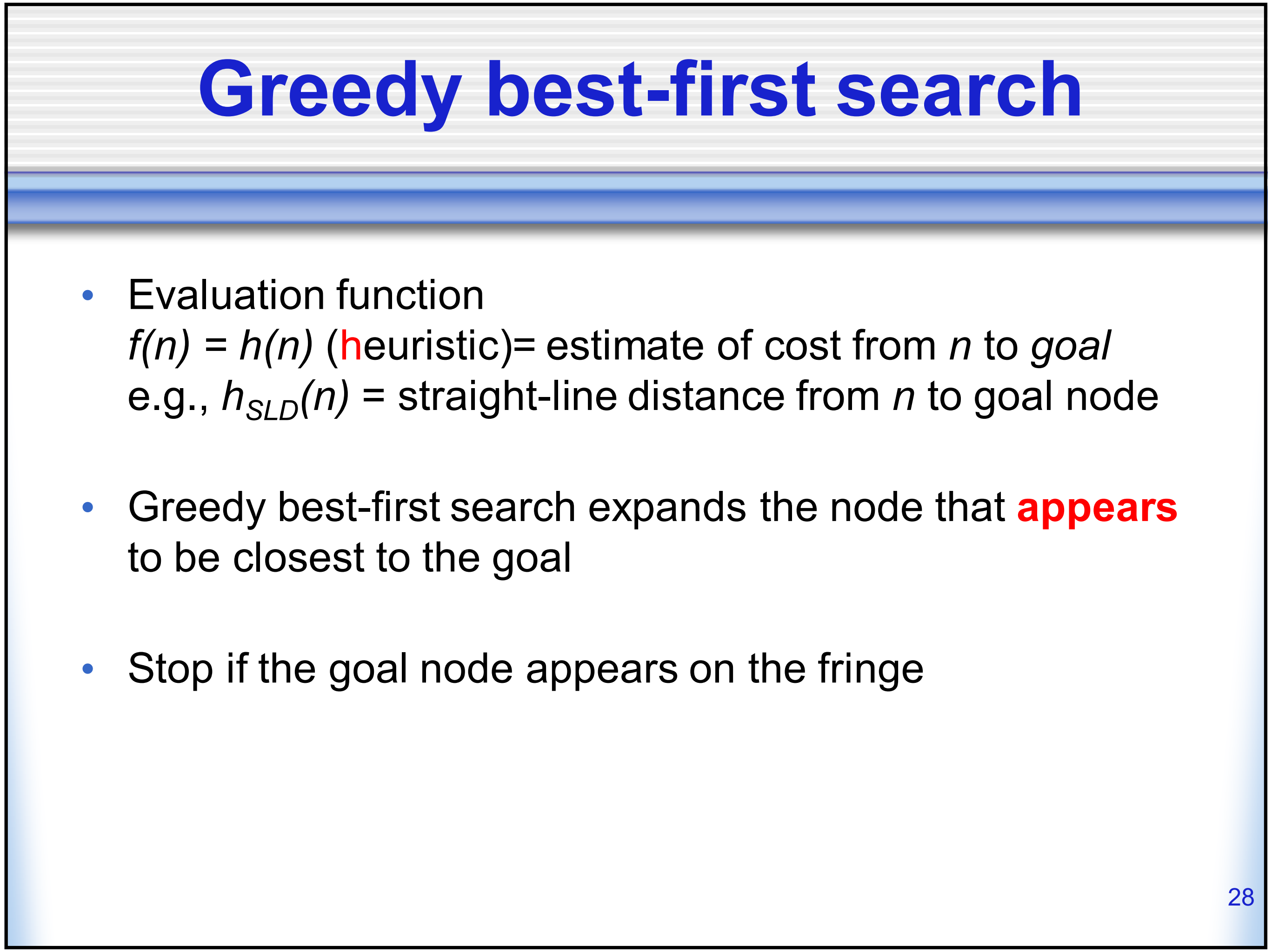
- Informed search
  - Greedy best-first search
  - A$^*$ search
- Admissible heuristics, creating heuristics
- Local search algorithms
  - Hill-climbing search
  - Simulated annealing search
  - Local beam search
  - Genetic algorithms
- Searching with nondeterministic actions



# Best-first search

- Idea: use a heuristic evaluation function *f(n)* for each node
  - estimate of "desirability"
  - → Expand most desirable unexpanded node
- Implementation:

  Order the nodes in fringe in decreasing order of desirability
- Special cases:
  - greedy best-first search
  - $A^*$ search



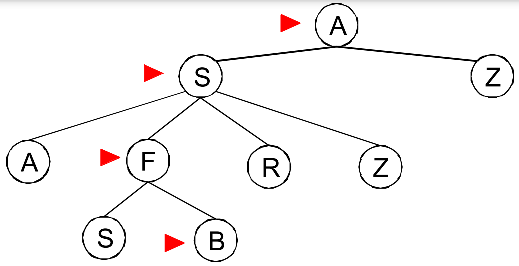
# Greedy best-first search

- Evaluation function
  *f(n)* = *h(n)* (heuristic)= estimate of cost from *n* to *goal*
  e.g., $h_{SLD}(n)$ = straight-line distance from *n* to goal node

- Greedy best-first search expands the node that **appears** to be closest to the goal

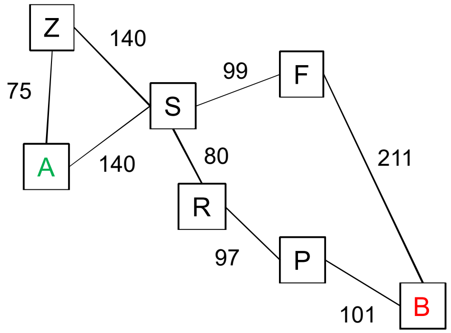
- Stop if the goal node appears on the fringe

# Greedy best-first search example: Go from A to B

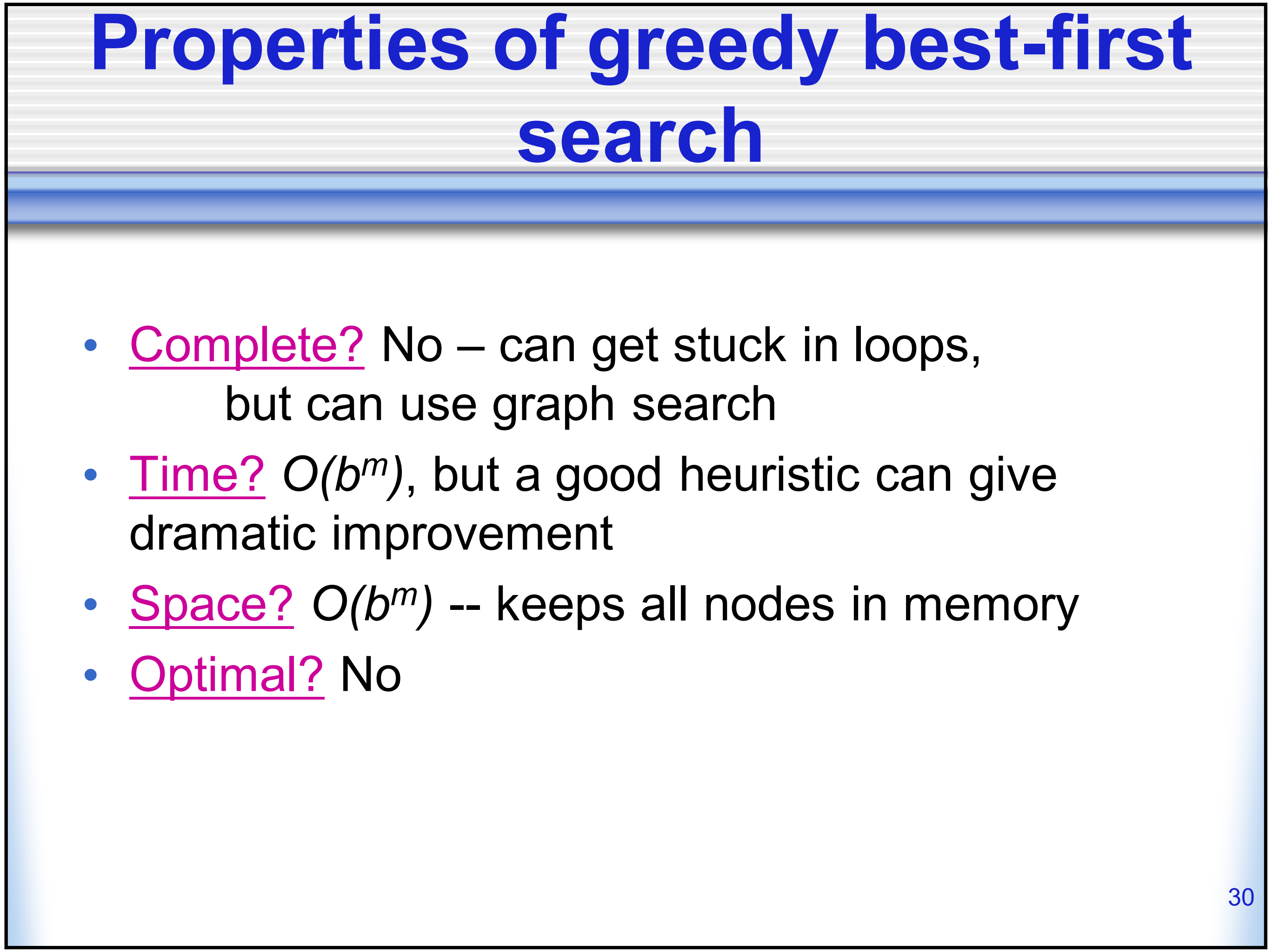


140
99
211

**450**

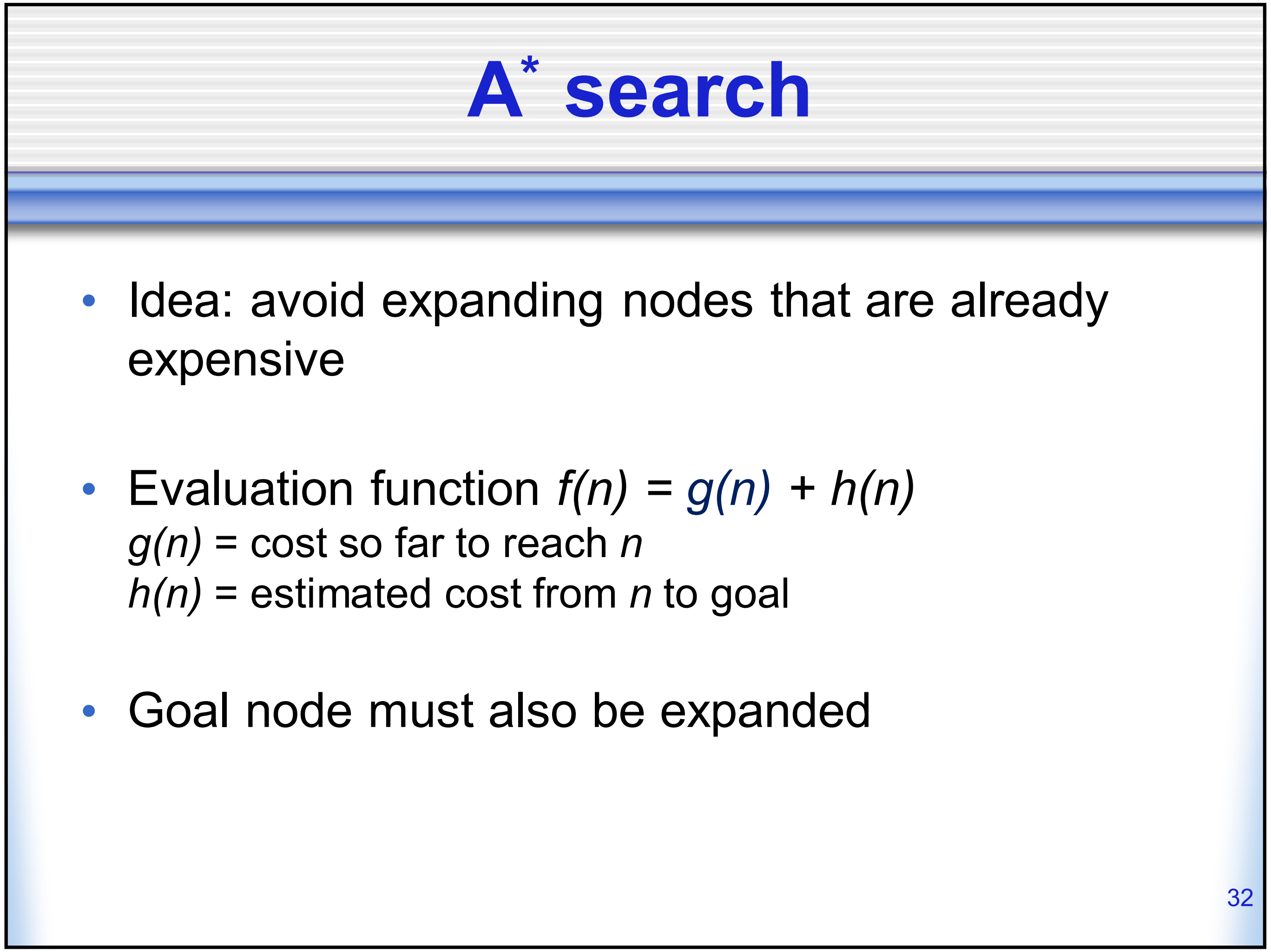


|   | B |
|---|---|
| A | 366 |
| Z | 374 |
| S | 253 |
| R | 193 |
| P | 100 |
| F | 176 |
| B | 0 |



# Properties of greedy best-first search

- Complete? No – can get stuck in loops, but can use graph search
- Time? $O(b^m)$, but a good heuristic can give dramatic improvement
- Space? $O(b^m)$ -- keeps all nodes in memory
- Optimal? No

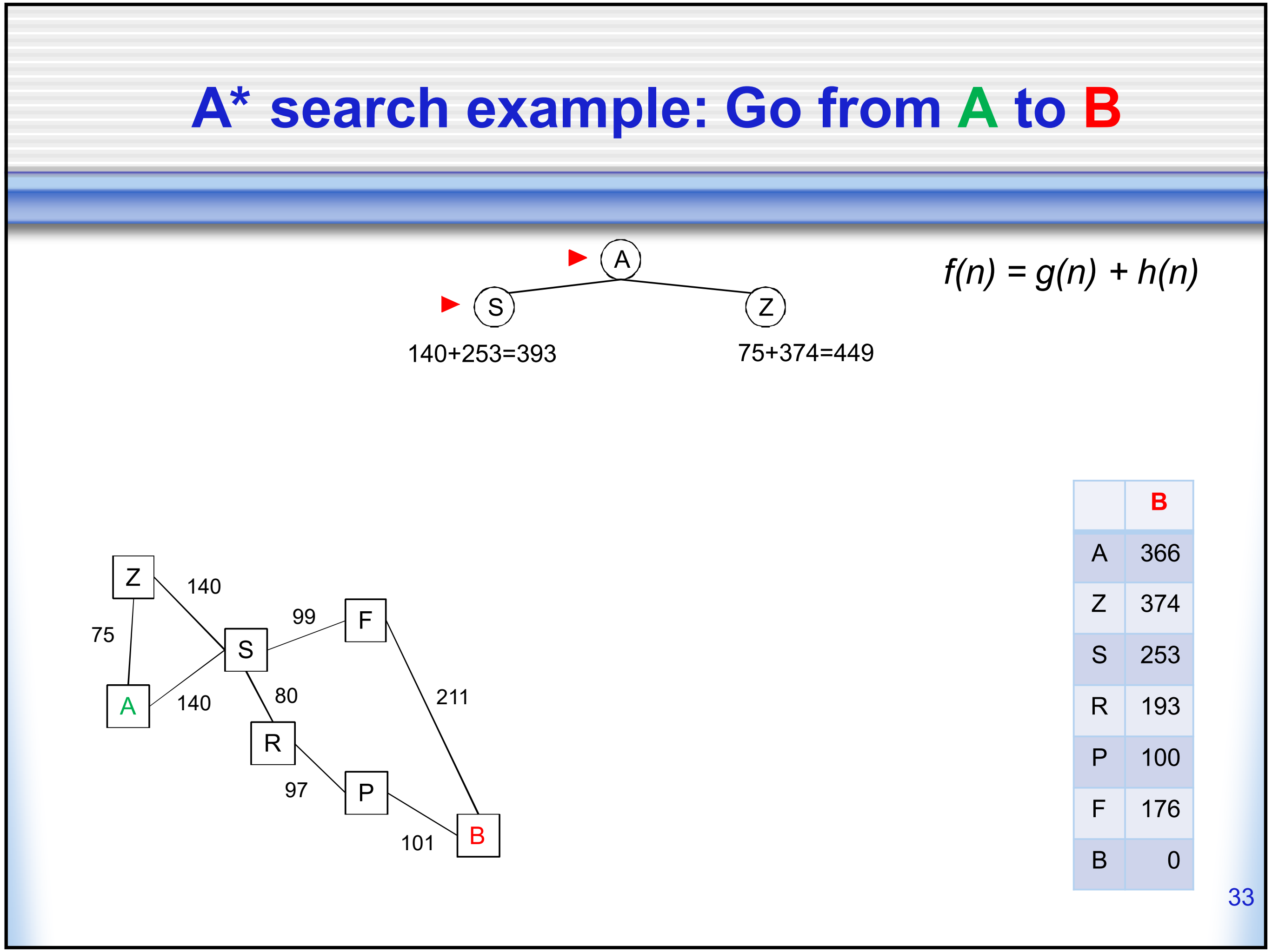


# A$^*$ search

- Idea: avoid expanding nodes that are already expensive

- Evaluation function *f(n)* = *g(n)* + *h(n)*
  *g(n)* = cost so far to reach *n*
  *h(n)* = estimated cost from *n* to goal

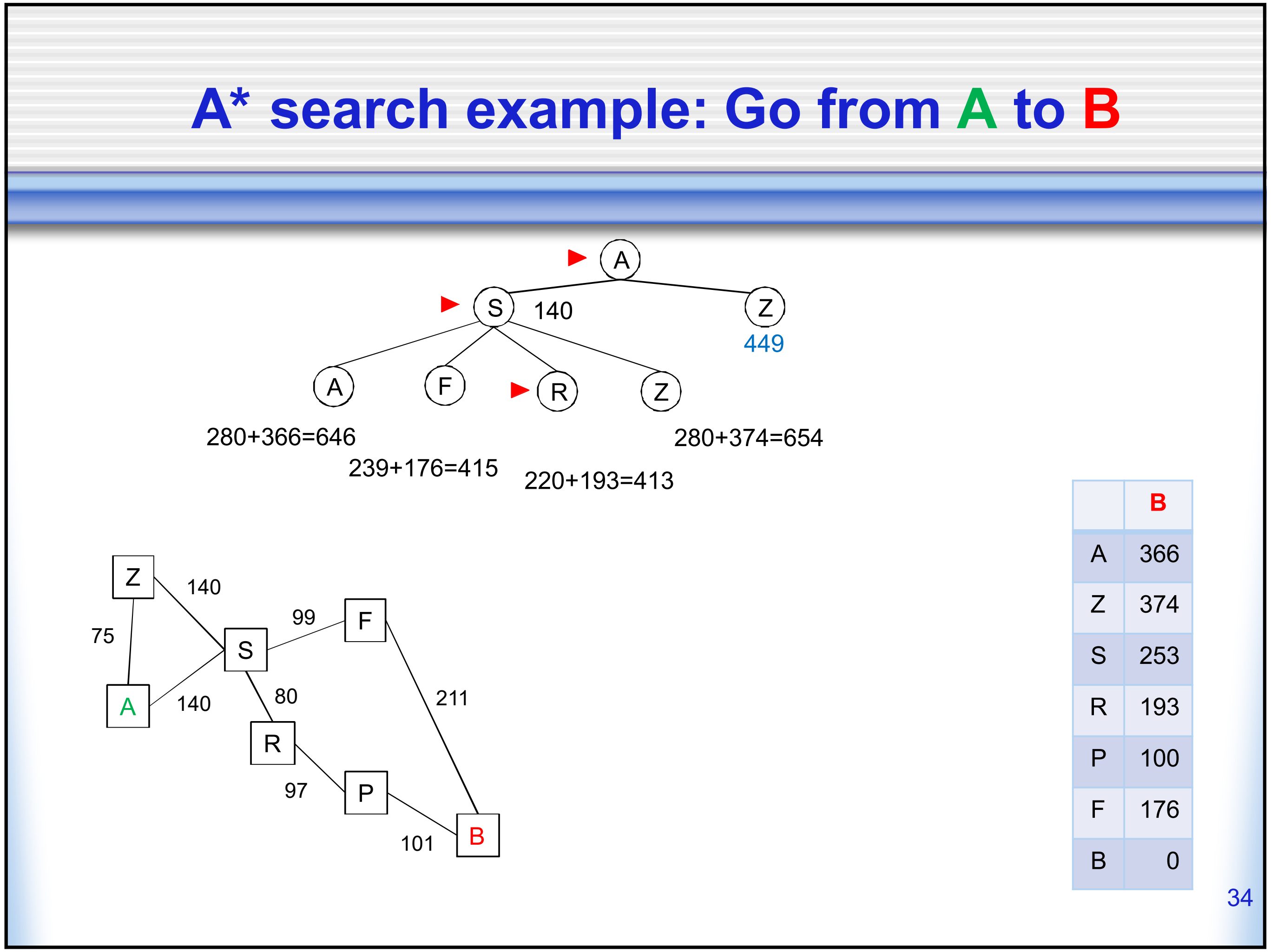
- Goal node must also be expanded



# A* search example: Go from A to B

$f(n) = g(n) + h(n)$

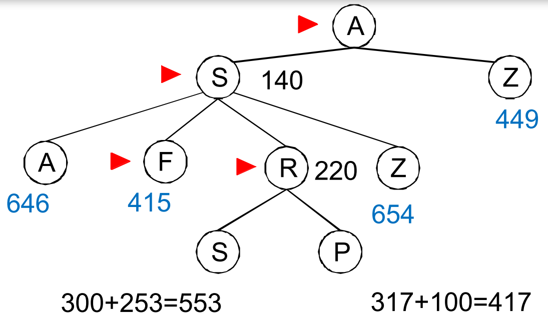
| | B |
|---|---|
| A | 366 |
| Z | 374 |
| S | 253 |
| R | 193 |
| P | 100 |
| F | 176 |
| B | 0 |

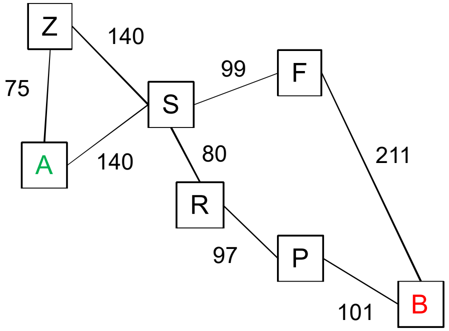


# A* search example: Go from A to B

| | B |
|---|---|
| A | 366 |
| Z | 374 |
| S | 253 |
| R | 193 |
| P | 100 |
| F | 176 |
| B | 0 |

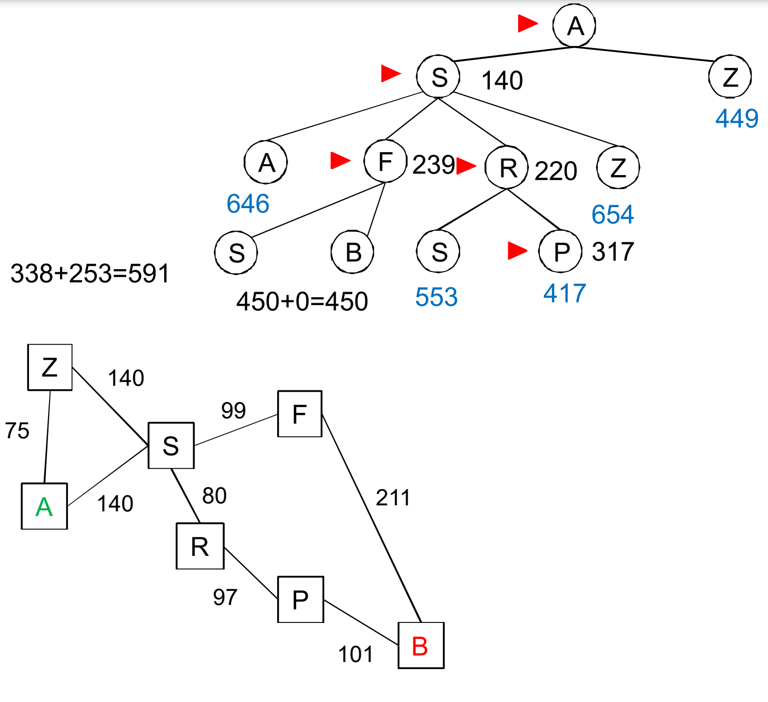
# A* search example: Go from A to B





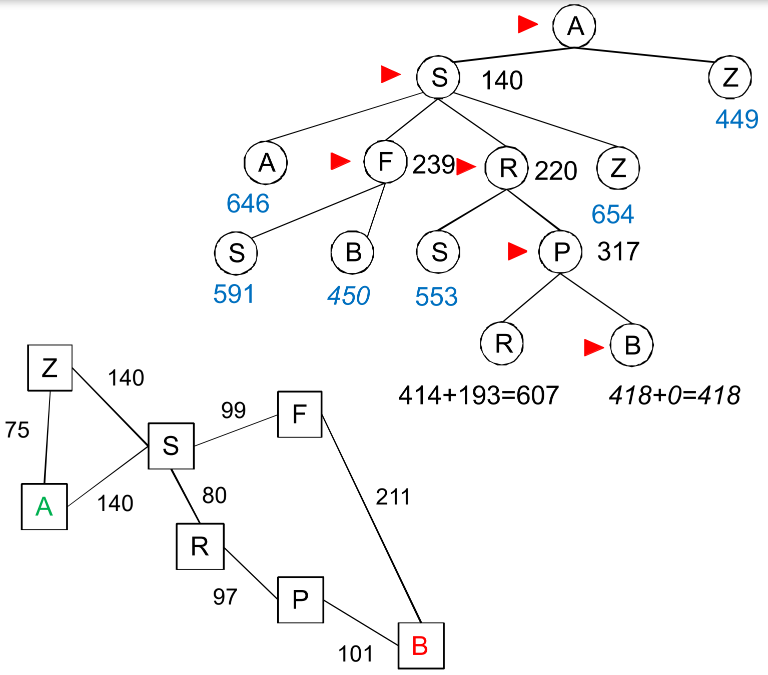
| | B |
|---|---|
| A | 366 |
| Z | 374 |
| S | 253 |
| R | 193 |
| P | 100 |
| F | 176 |
| B | 0 |

# A* search example: Go from A to B



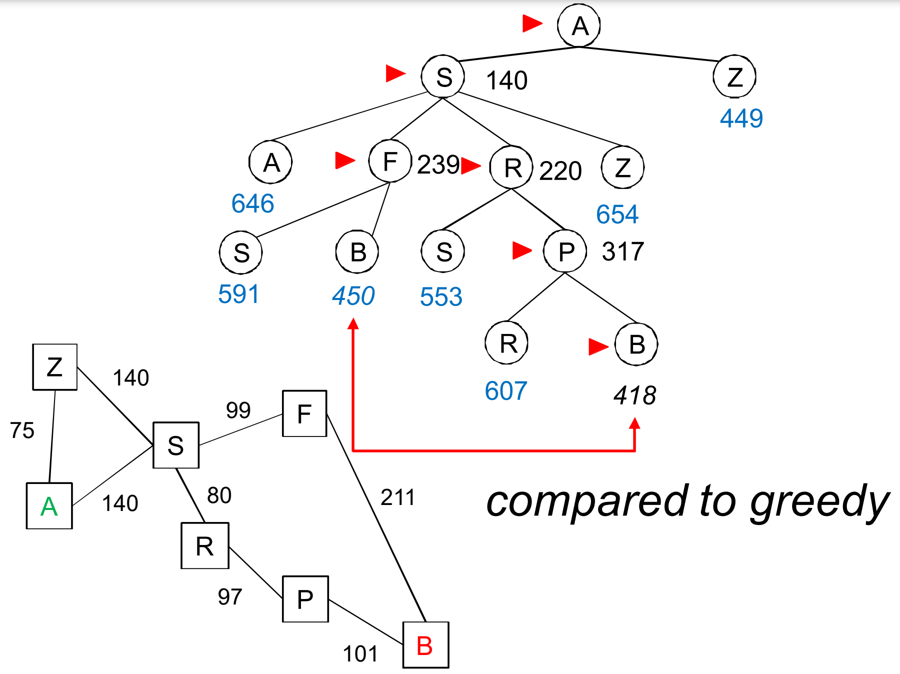
| | B |
|---|---|
| A | 366 |
| Z | 374 |
| S | 253 |
| R | 193 |
| P | 100 |
| F | 176 |
| B | 0 |

# A* search example: Go from A to B



|   | B |
|---|---|
| A | 366 |
| Z | 374 |
| S | 253 |
| R | 193 |
| P | 100 |
| F | 176 |
| B | 0 |

# A* search example: Go from A to B



compared to greedy

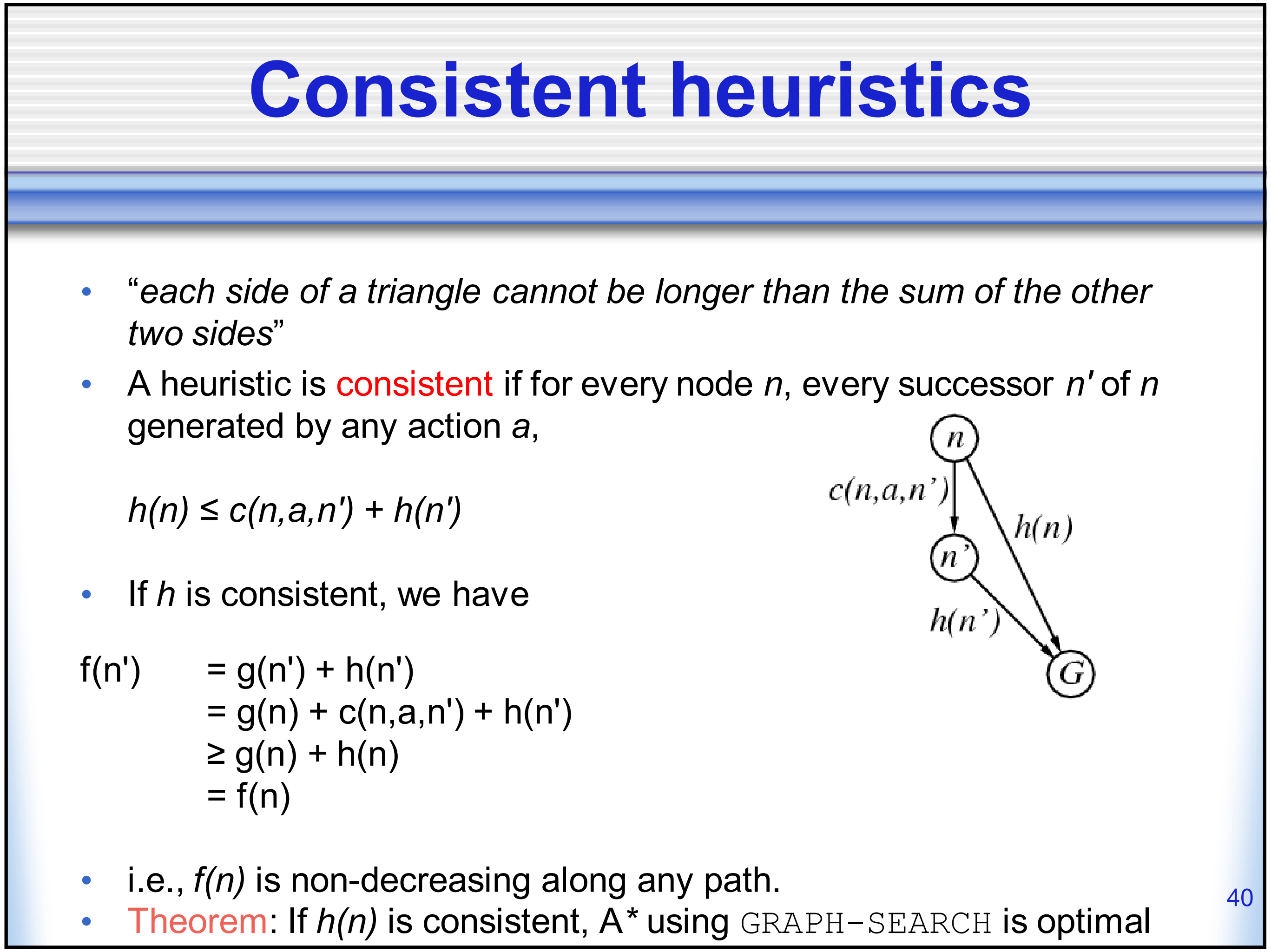
| | B |
|---|---|
| A | 366 |
| Z | 374 |
| S | 253 |
| R | 193 |
| P | 100 |
| F | 176 |
| B | 0 |

# Admissible heuristics

- A heuristic *h(n)* is admissible if for every node *n*,
  *h(n)* ≤ *h*$^*$*(n)*, where *h*$^*$*(n)* is the true cost to reach the goal state from *n*.
  *An admissible heuristic never overestimates the cost to reach the goal, i.e., it is optimistic*

- Example: $h_{SLD}(n)$ (never overestimates the actual road distance)

- Theorem: If *h(n)* is admissible,
  A$^*$ using `TREE-SEARCH` is optimal
- For graph searches we nee a stronger criteria



# Consistent heuristics

- *“each side of a triangle cannot be longer than the sum of the other two sides”*
- A heuristic is consistent if for every node *n*, every successor *n'* of *n* generated by any action *a*,

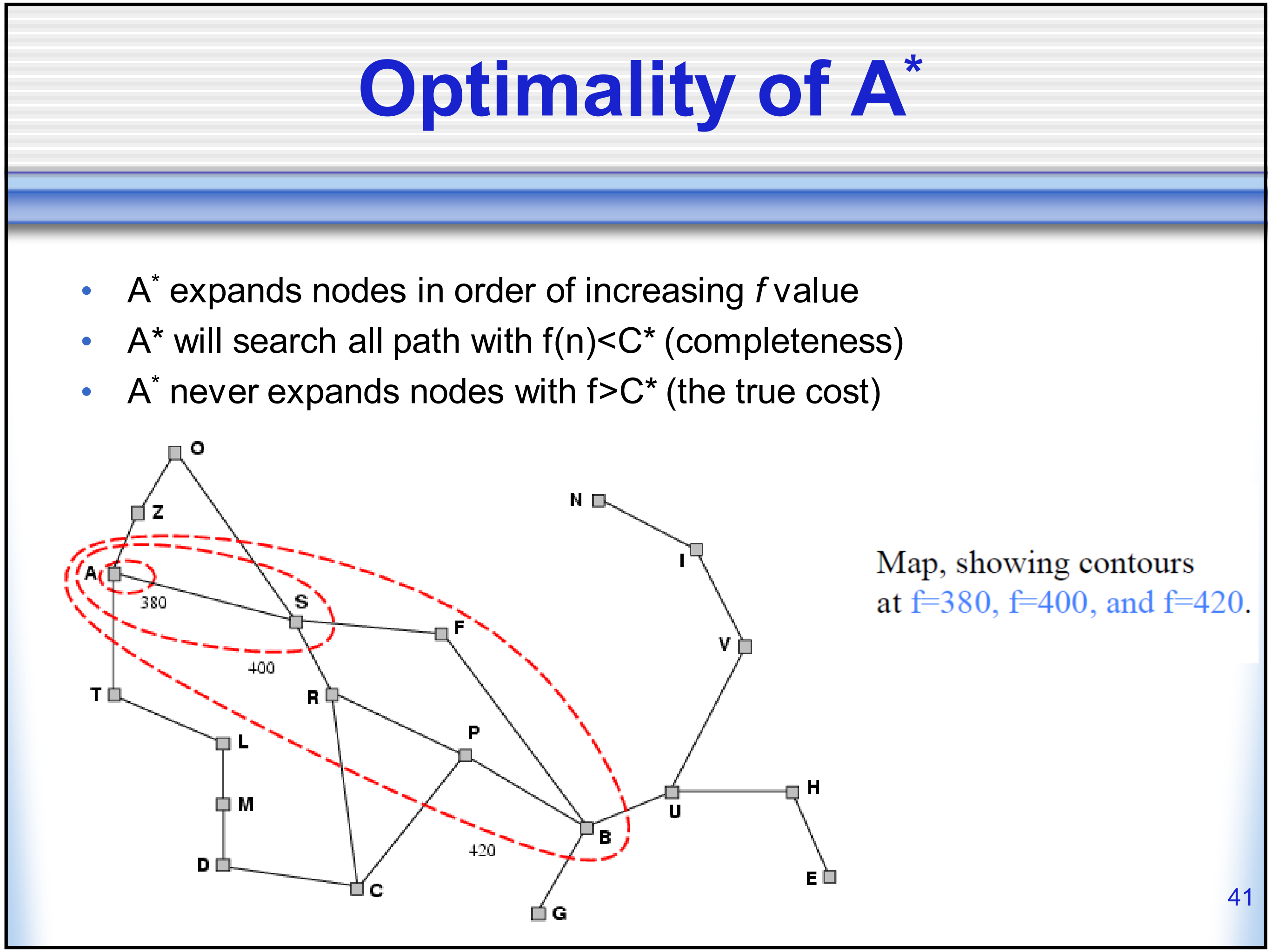
  $h(n) \leq c(n,a,n') + h(n')$

- If *h* is consistent, we have

$$
\begin{aligned}
f(n') &= g(n') + h(n') \\
&= g(n) + c(n,a,n') + h(n') \\
&\geq g(n) + h(n) \\
&= f(n)
\end{aligned}
$$

n
c(n,a,n')
h(n)
n'
h(n')
G

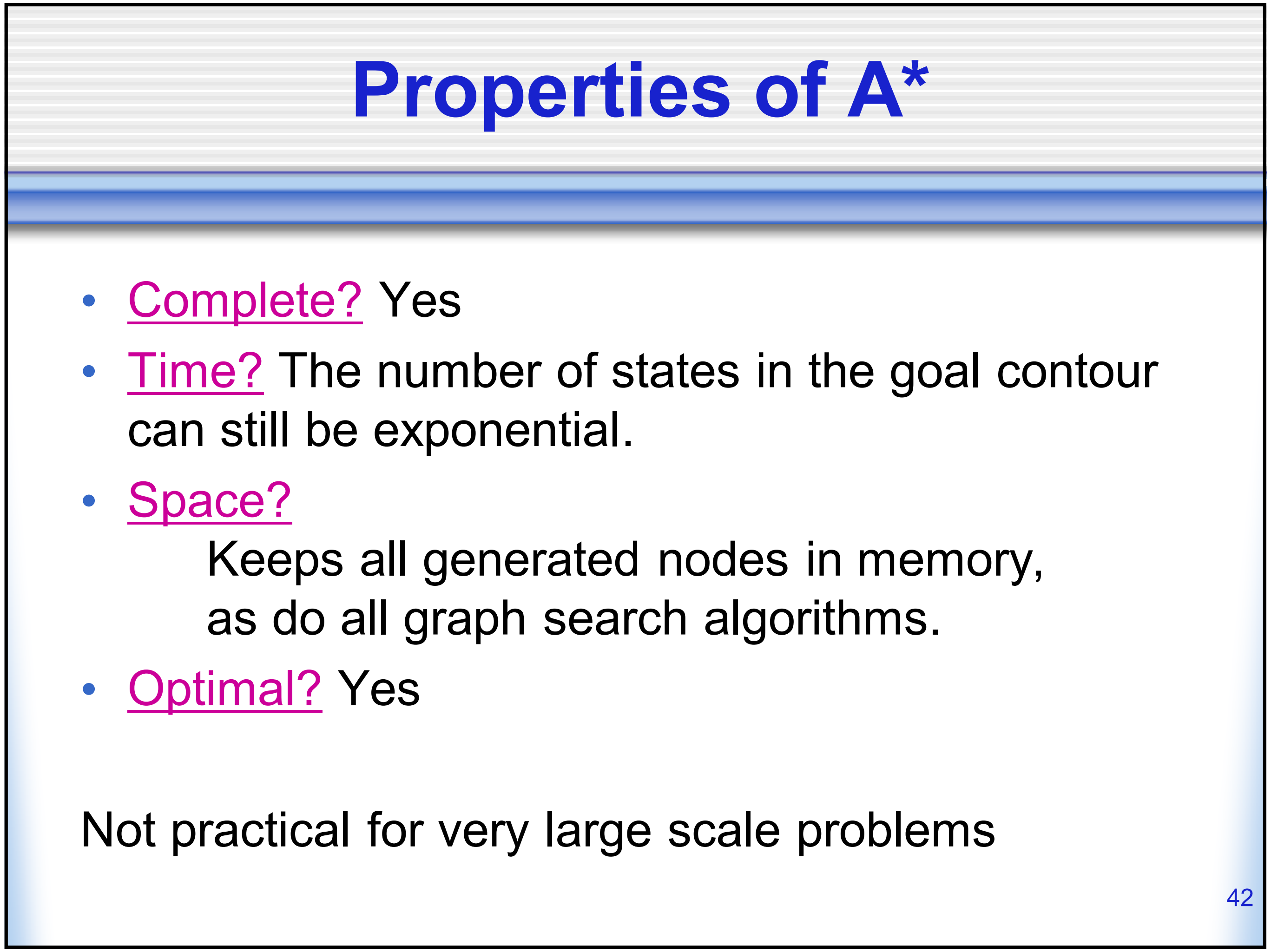
- i.e., *f(n)* is non-decreasing along any path.
- Theorem: If *h(n)* is consistent, A* using `GRAPH-SEARCH` is optimal



# Optimality of A$^*$

- A$^*$ expands nodes in order of increasing *f* value
- A* will search all path with f(n)<C* (completeness)
- A$^*$ never expands nodes with f>C* (the true cost)

Map, showing contours at f=380, f=400, and f=420.



# Properties of A*

- Complete? Yes
- Time? The number of states in the goal contour can still be exponential.
- Space?

  Keeps all generated nodes in memory,
  as do all graph search algorithms.
- Optimal? Yes

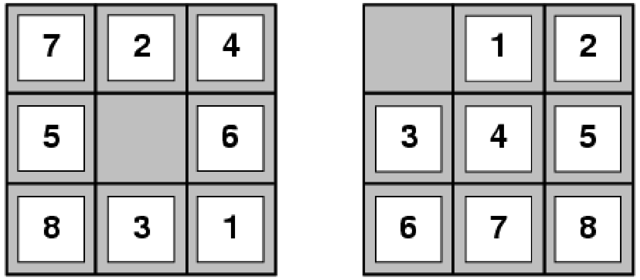
Not practical for very large scale problems

# Admissible heuristics

E.g., for the 8-puzzle:

- $h_1(n)$ = number of misplaced tiles
- $h_2(n)$ = total Manhattan distance

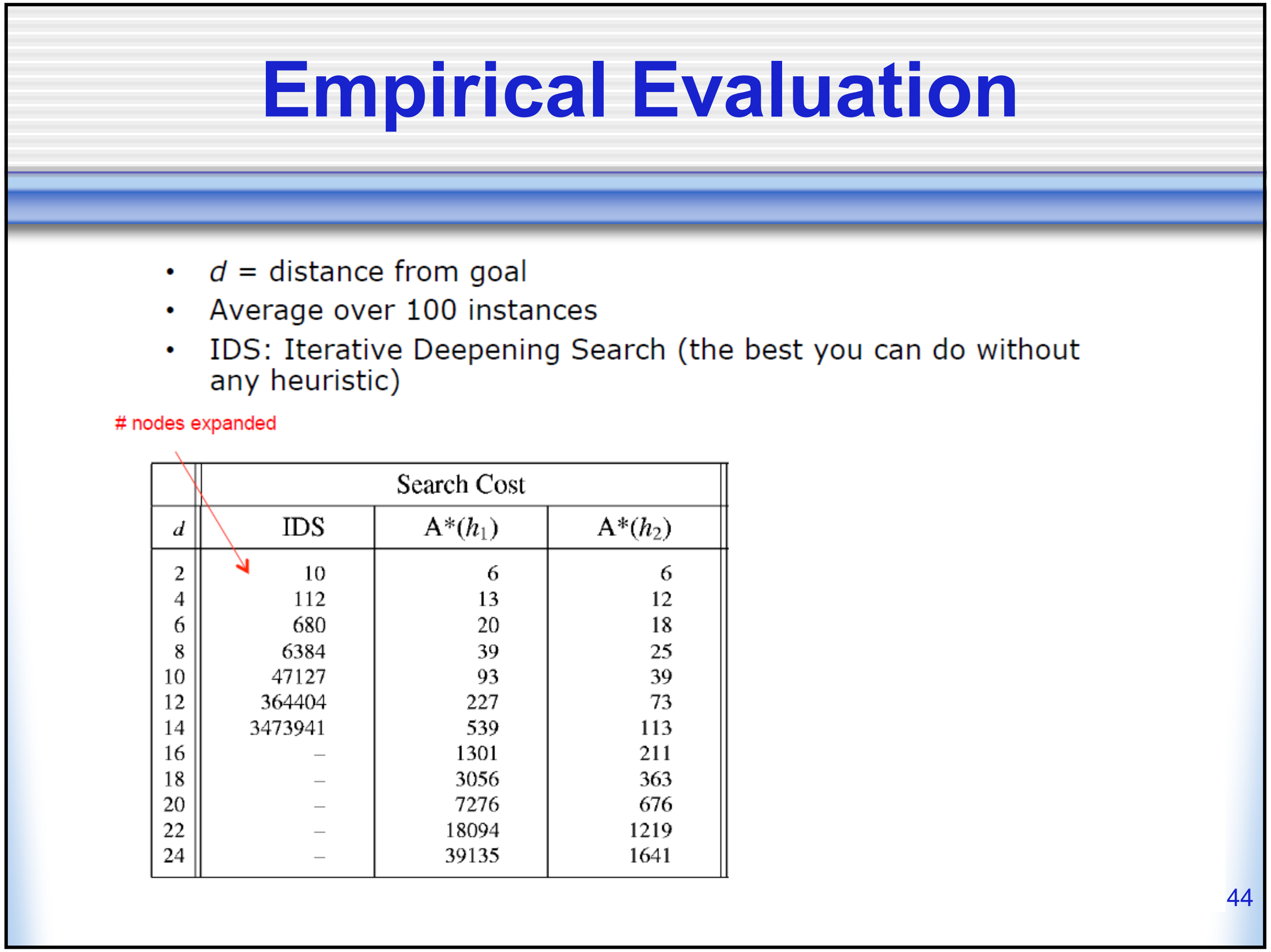
(i.e., no. of squares from desired location of each tile)



Start State

Goal State

- $h_1(S)$ = ? 8
- $h_2(S)$ = ? 3+1+2+2+2+3+3+2 = 18

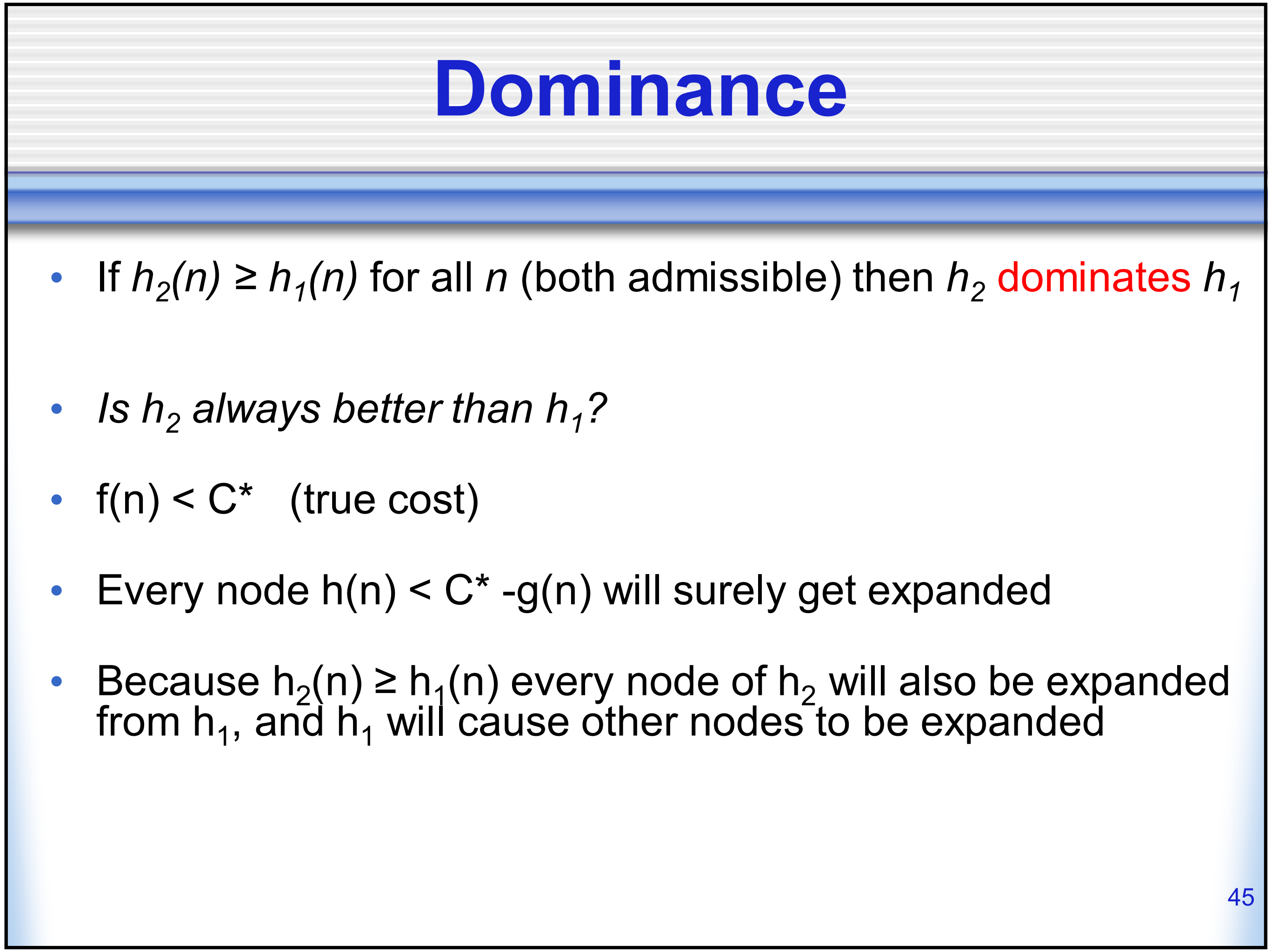


# Empirical Evaluation

- *d* = distance from goal
- Average over 100 instances
- IDS: Iterative Deepening Search (the best you can do without any heuristic)

# nodes expanded

| | Search Cost | | |
|---|---|---|---|
| $d$ | IDS | $A^*(h_1)$ | $A^*(h_2)$ |
| 2 | 10 | 6 | 6 |
| 4 | 112 | 13 | 12 |
| 6 | 680 | 20 | 18 |
| 8 | 6384 | 39 | 25 |
| 10 | 47127 | 93 | 39 |
| 12 | 364404 | 227 | 73 |
| 14 | 3473941 | 539 | 113 |
| 16 | – | 1301 | 211 |
| 18 | – | 3056 | 363 |
| 20 | – | 7276 | 676 |
| 22 | – | 18094 | 1219 |
| 24 | – | 39135 | 1641 |



# Dominance

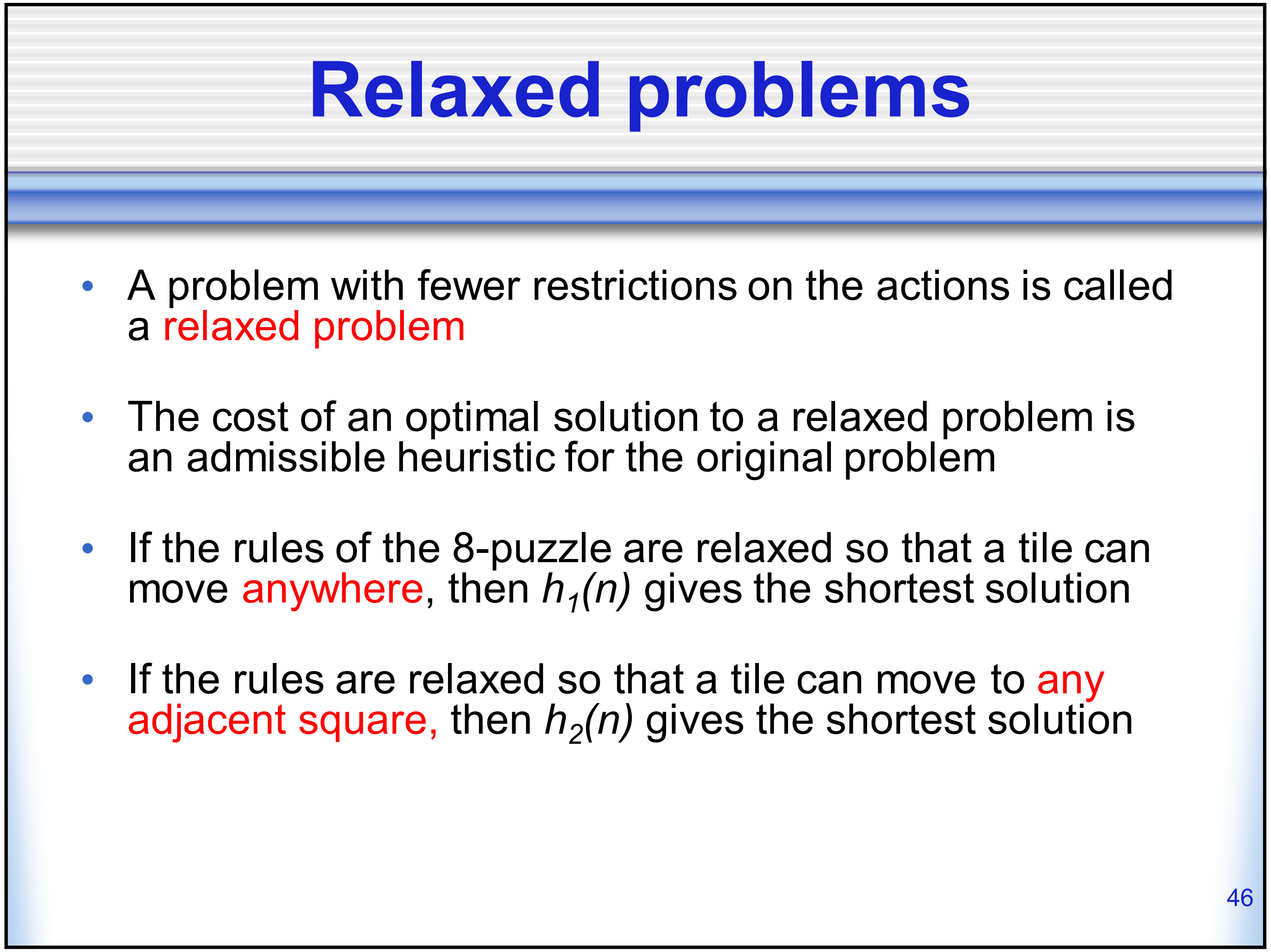
- If $h_2(n) \geq h_1(n)$ for all $n$ (both admissible) then $h_2$ dominates $h_1$

- *Is $h_2$ always better than $h_1$?*

- $f(n) < C^*$ (true cost)

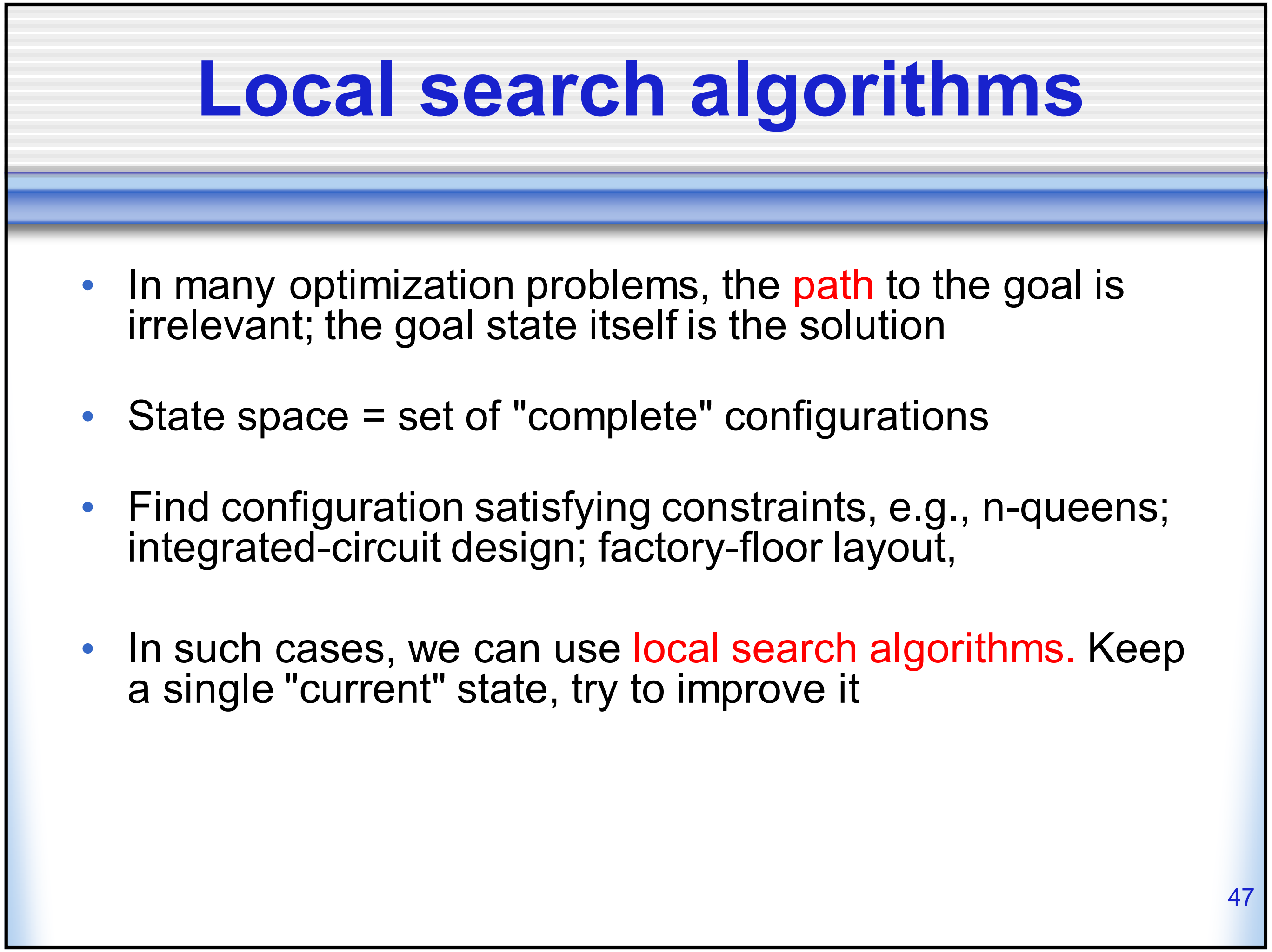
- Every node $h(n) < C^* - g(n)$ will surely get expanded

- Because $h_2(n) \geq h_1(n)$ every node of $h_2$ will also be expanded from $h_1$, and $h_1$ will cause other nodes to be expanded



# Relaxed problems

- A problem with fewer restrictions on the actions is called a relaxed problem
- The cost of an optimal solution to a relaxed problem is an admissible heuristic for the original problem
- If the rules of the 8-puzzle are relaxed so that a tile can move anywhere, then $h_1(n)$ gives the shortest solution
- If the rules are relaxed so that a tile can move to any adjacent square, then $h_2(n)$ gives the shortest solution



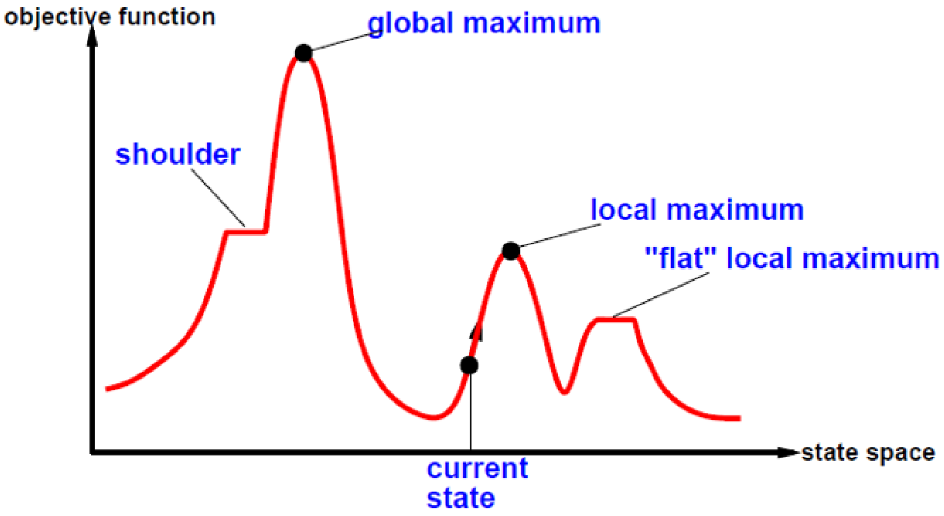
# Local search algorithms

- In many optimization problems, the path to the goal is irrelevant; the goal state itself is the solution
- State space = set of "complete" configurations
- Find configuration satisfying constraints, e.g., n-queens; integrated-circuit design; factory-floor layout,
- In such cases, we can use local search algorithms. Keep a single "current" state, try to improve it

47

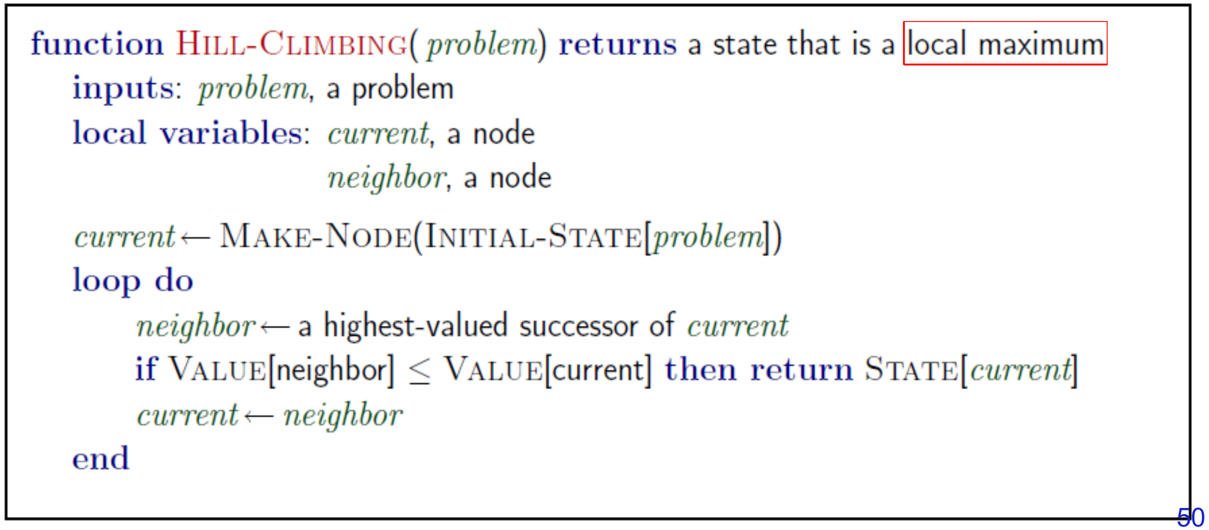
# State space and objective Funtion

Useful to consider state space landscape



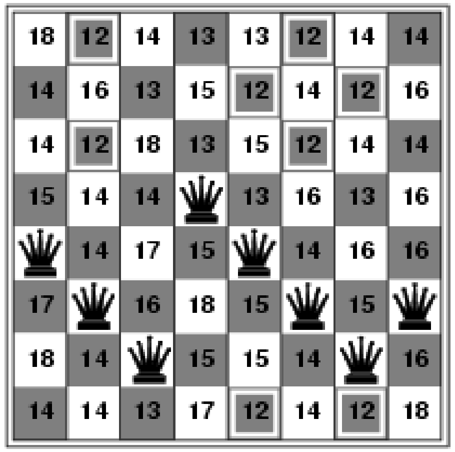
# Hill-climbing search

- "Like climbing Everest in thick fog with amnesia“ (Russell, Norvig)



```
function HILL-CLIMBING( problem) returns a state that is a local maximum
   inputs: problem, a problem
   local variables: current, a node
                    neighbor, a node

   current $\leftarrow$ MAKE-NODE(INITIAL-STATE[problem])
   loop do
       neighbor $\leftarrow$ a highest-valued successor of current
       if VALUE[neighbor] $\leq$ VALUE[current] then return STATE[current]
       current $\leftarrow$ neighbor
   end
```

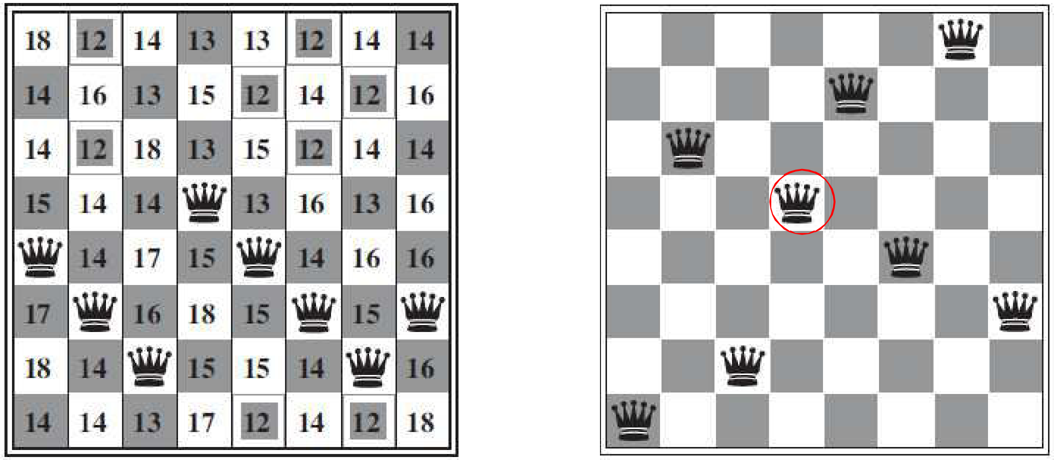
# Hill-climbing search: 8-queens problem



| | | | | | | | |
|---|---|---|---|---|---|---|---|
| 18 | 12 | 14 | 13 | 13 | 12 | 14 | 14 |
| 14 | 16 | 13 | 15 | 12 | 14 | 12 | 16 |
| 14 | 12 | 18 | 13 | 15 | 12 | 14 | 14 |
| 15 | 14 | 14 | ♛ | 13 | 16 | 13 | 16 |
| ♛ | 14 | 17 | 15 | ♛ | 14 | 16 | 16 |
| 17 | ♛ | 16 | 18 | 15 | ♛ | 15 | ♛ |
| 18 | 14 | ♛ | 15 | 15 | 14 | ♛ | 16 |
| 14 | 14 | 13 | 17 | 12 | 14 | 12 | 18 |

- The successors of a state are all possible states generated by moving a single queen to another square in the same column (so each state has 8 × 7=56 successors)
- Cost function: *h* = number of pairs of queens that are attacking each other, either directly or indirectly
- *h* = *17* for the above state, best moves are marked.

# Hill-climbing search: 8-queens problem

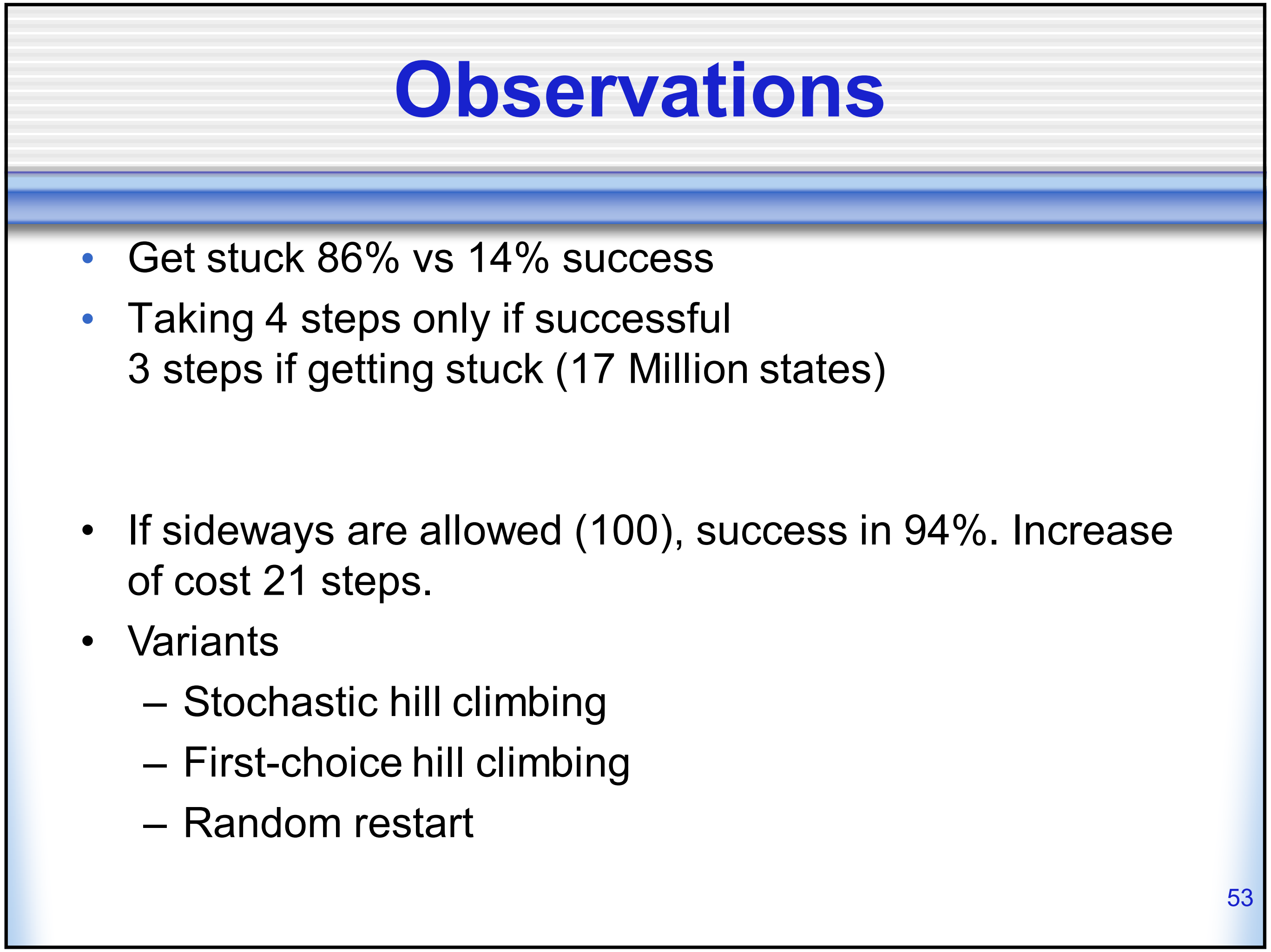


|  |  |  |  |  |  |  |  |
|---|---|---|---|---|---|---|---|
| 18 | 12 | 14 | 13 | 13 | 12 | 14 | 14 |
| 14 | 16 | 13 | 15 | 12 | 14 | 12 | 16 |
| 14 | 12 | 18 | 13 | 15 | 12 | 14 | 14 |
| 15 | 14 | 14 | ♛ | 13 | 16 | 13 | 16 |
| ♛ | 14 | 17 | 15 | ♛ | 14 | 16 | 16 |
| 17 | ♛ | 16 | 18 | 15 | ♛ | 15 | ♛ |
| 18 | 14 | ♛ | 15 | 15 | 14 | ♛ | 16 |
| 14 | 14 | 13 | 17 | 12 | 14 | 12 | 18 |

(a)

(b)

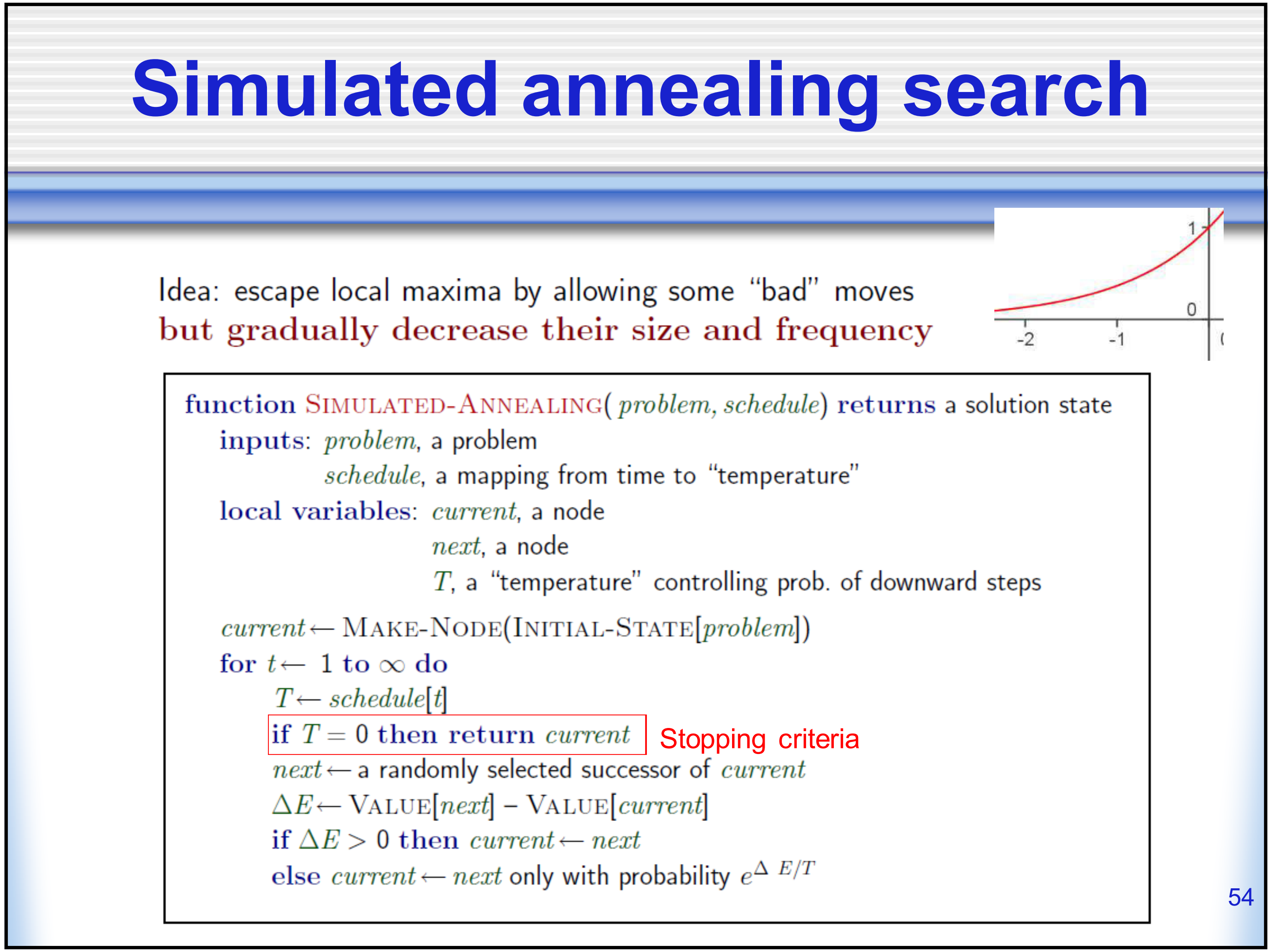
**Figure 4.3** (a) An 8-queens state with heuristic cost estimate $h = 17$, showing the value of $h$ for each possible successor obtained by moving a queen within its column. The best moves are marked. (b) A local minimum in the 8-queens state space; the state has $h = 1$ but every successor has a higher cost.



# Observations

- Get stuck 86% vs 14% success
- Taking 4 steps only if successful
  3 steps if getting stuck (17 Million states)

- If sideways are allowed (100), success in 94%. Increase of cost 21 steps.
- Variants
  - Stochastic hill climbing
  - First-choice hill climbing
  - Random restart



# Simulated annealing search

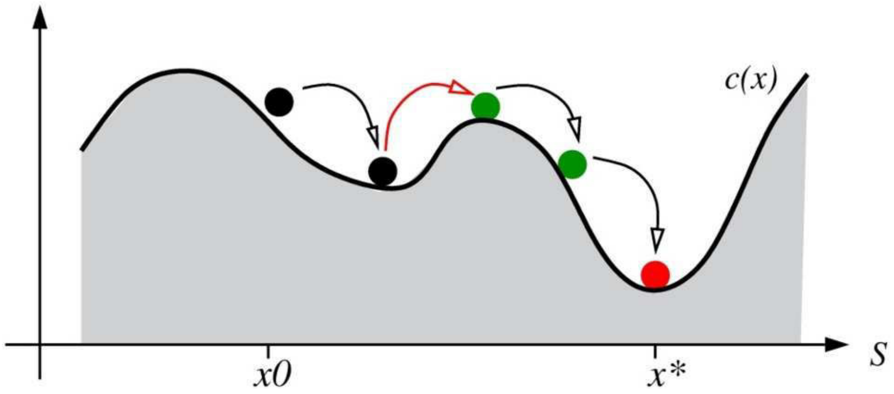
Idea: escape local maxima by allowing some "bad" moves
but gradually decrease their size and frequency

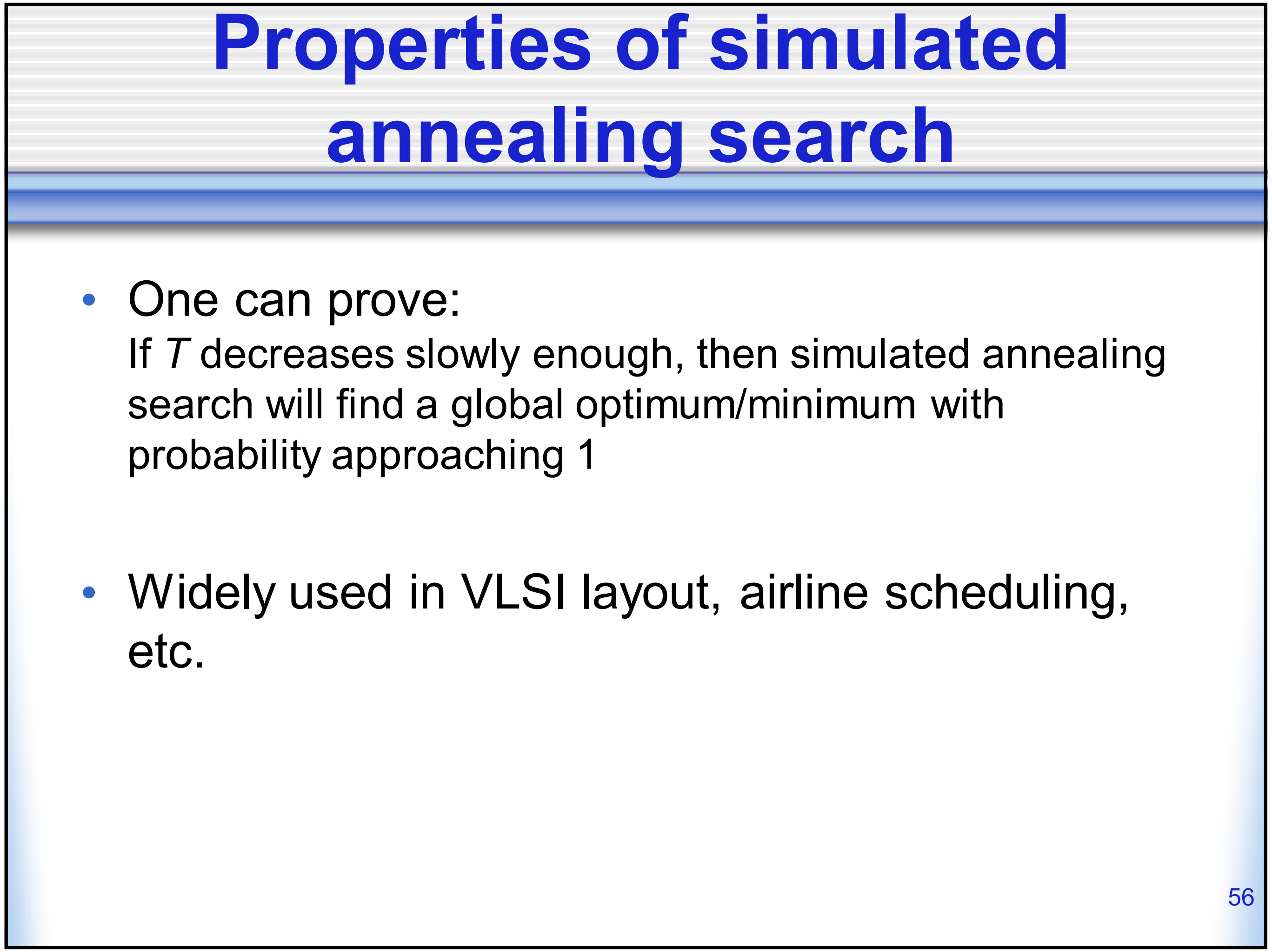
```
function SIMULATED-ANNEALING( problem, schedule) returns a solution state
   inputs: problem, a problem
           schedule, a mapping from time to "temperature"
   local variables: current, a node
                    next, a node
                    T, a "temperature" controlling prob. of downward steps

   current ← MAKE-NODE(INITIAL-STATE[problem])
   for t ← 1 to ∞ do
      T ← schedule[t]
      if $T = 0$ then return current
      next ← a randomly selected successor of current
      $\Delta E$ ← VALUE[next] – VALUE[current]
      if $\Delta E > 0$ then current ← next
      else current ← next only with probability $e^{\Delta E/T}$
```

Stopping criteria

54

# Simulated Annealing

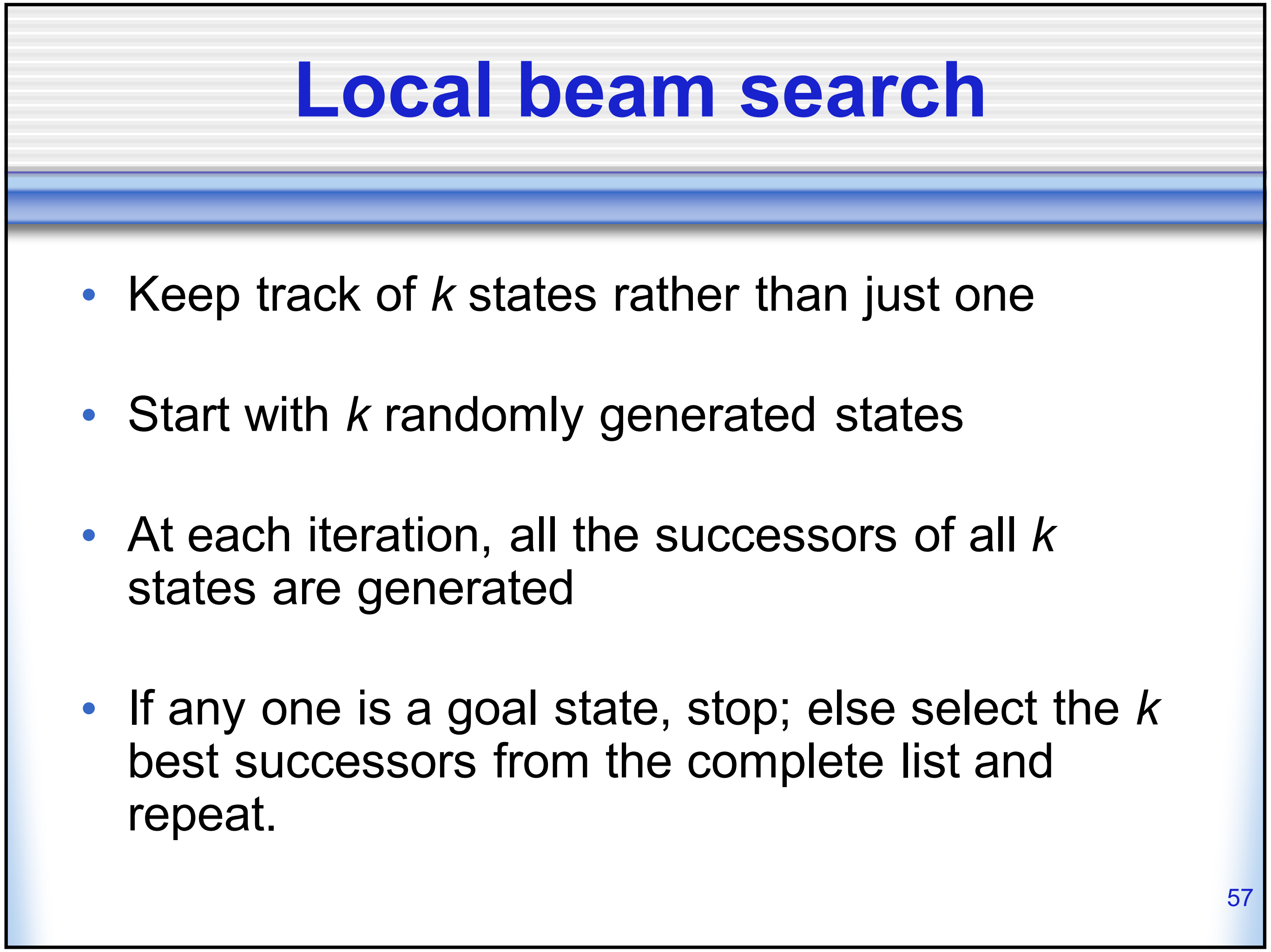




# Properties of simulated annealing search

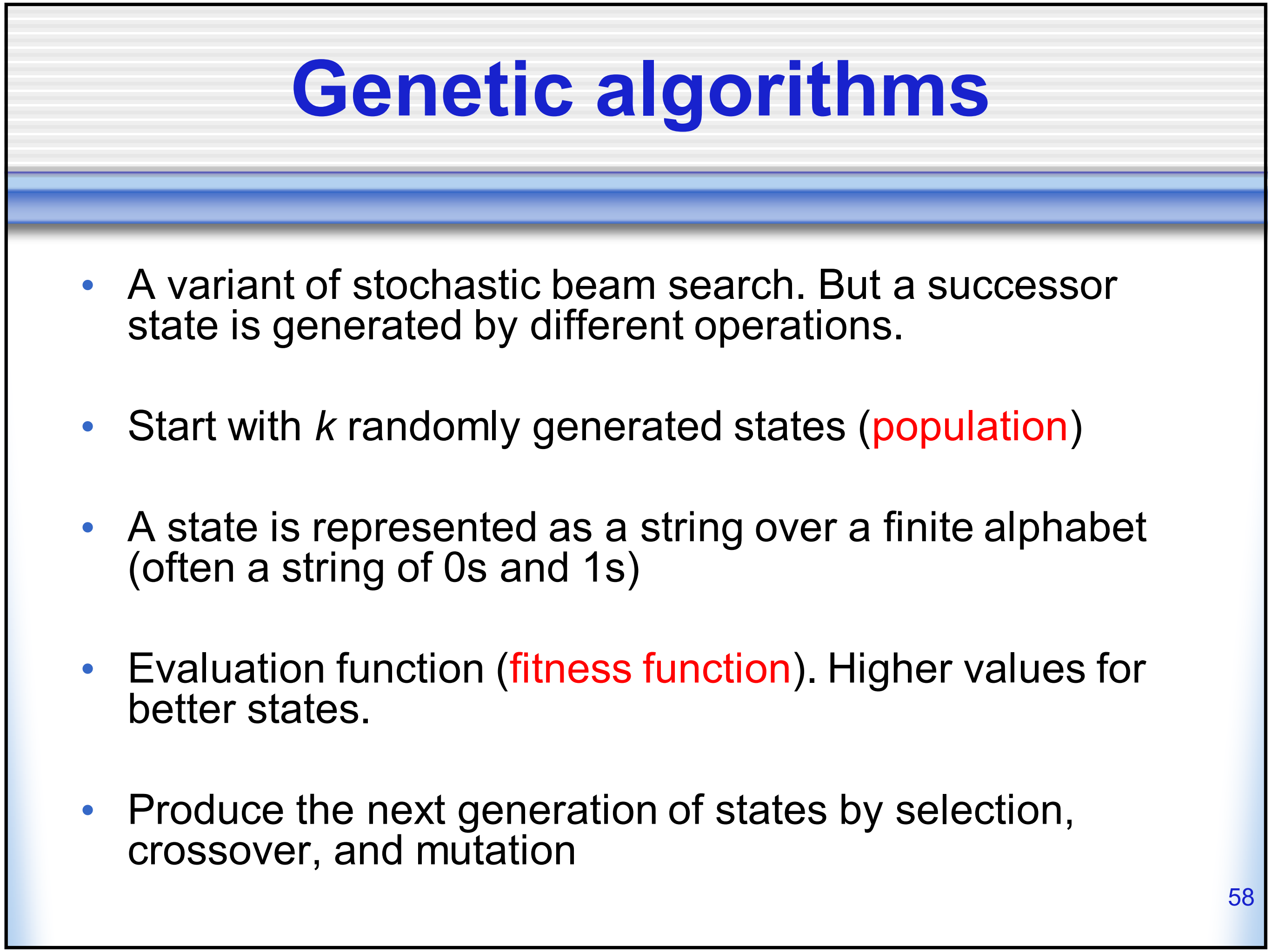
- One can prove:
  If *T* decreases slowly enough, then simulated annealing search will find a global optimum/minimum with probability approaching 1

- Widely used in VLSI layout, airline scheduling, etc.



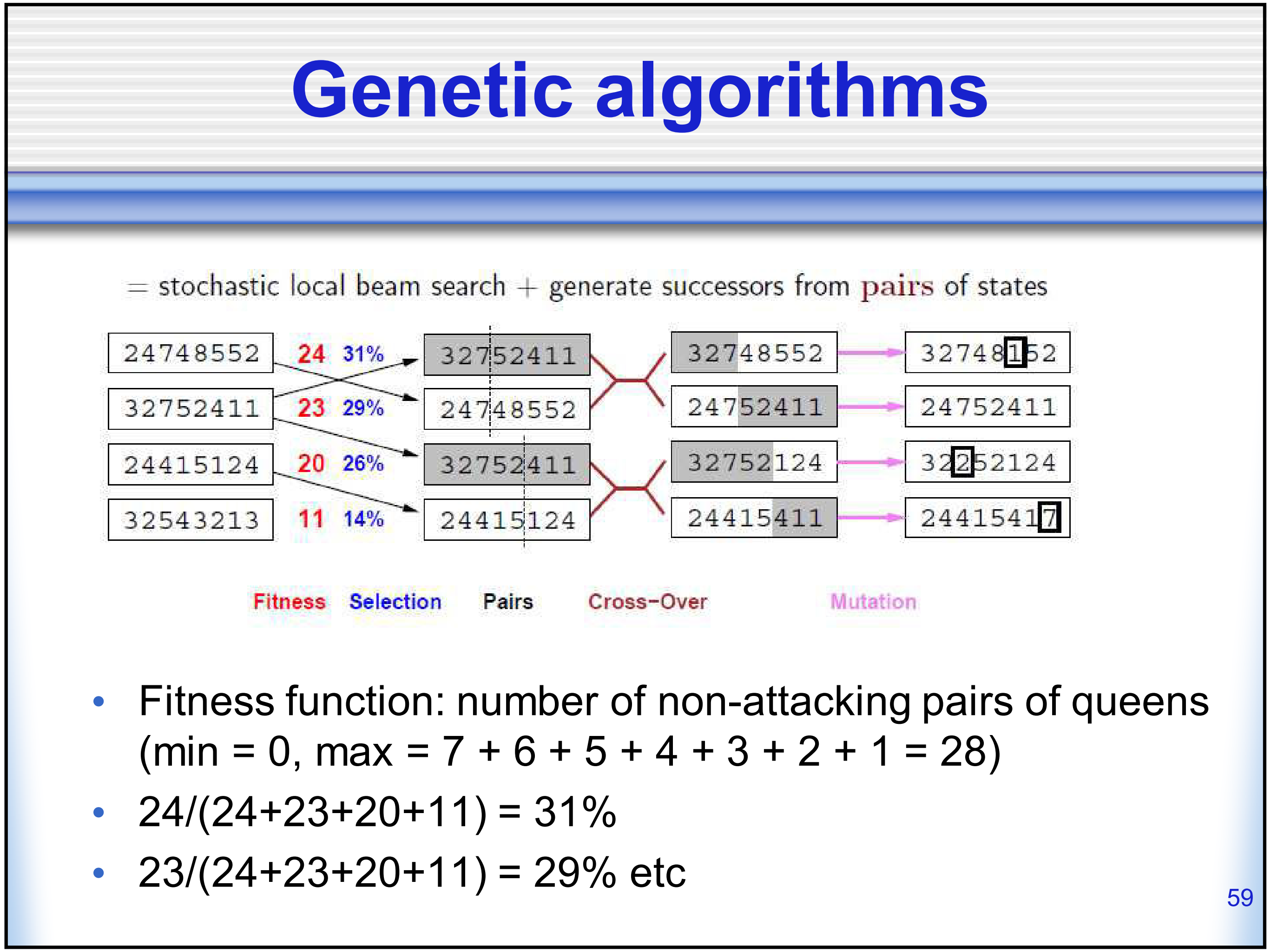
# Local beam search

- Keep track of *k* states rather than just one
- Start with *k* randomly generated states
- At each iteration, all the successors of all *k* states are generated
- If any one is a goal state, stop; else select the *k* best successors from the complete list and repeat.



# Genetic algorithms

- A variant of stochastic beam search. But a successor state is generated by different operations.
- Start with *k* randomly generated states (population)
- A state is represented as a string over a finite alphabet (often a string of 0s and 1s)
- Evaluation function (fitness function). Higher values for better states.
- Produce the next generation of states by selection, crossover, and mutation



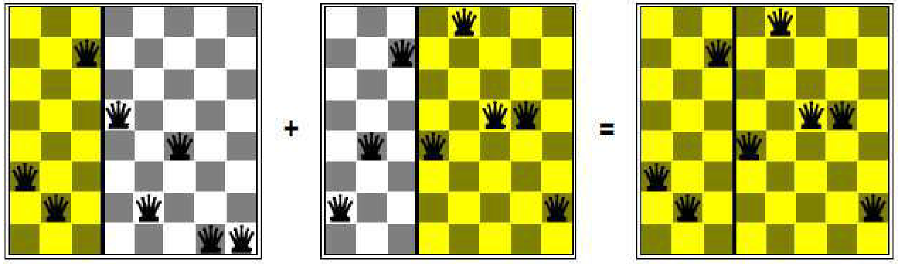
# Genetic algorithms

= stochastic local beam search + generate successors from pairs of states

- Fitness function: number of non-attacking pairs of queens (min = 0, max = 7 + 6 + 5 + 4 + 3 + 2 + 1 = 28)
- 24/(24+23+20+11) = 31%
- 23/(24+23+20+11) = 29% etc

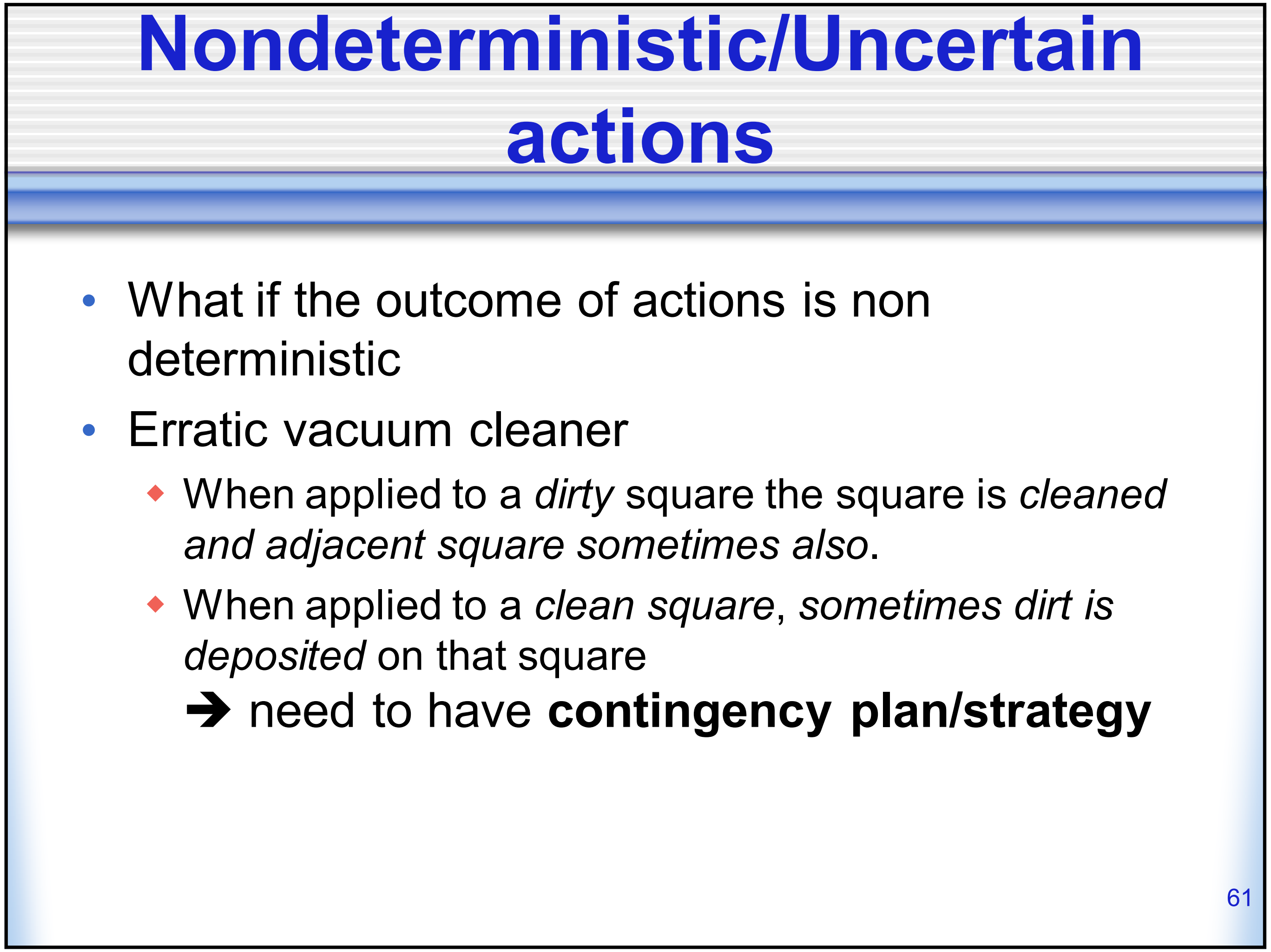
# Genetic algorithms

Crossover helps iff substrings are meaningful components



GAs ≠ evolution: e.g., real genes encode replication machinery!

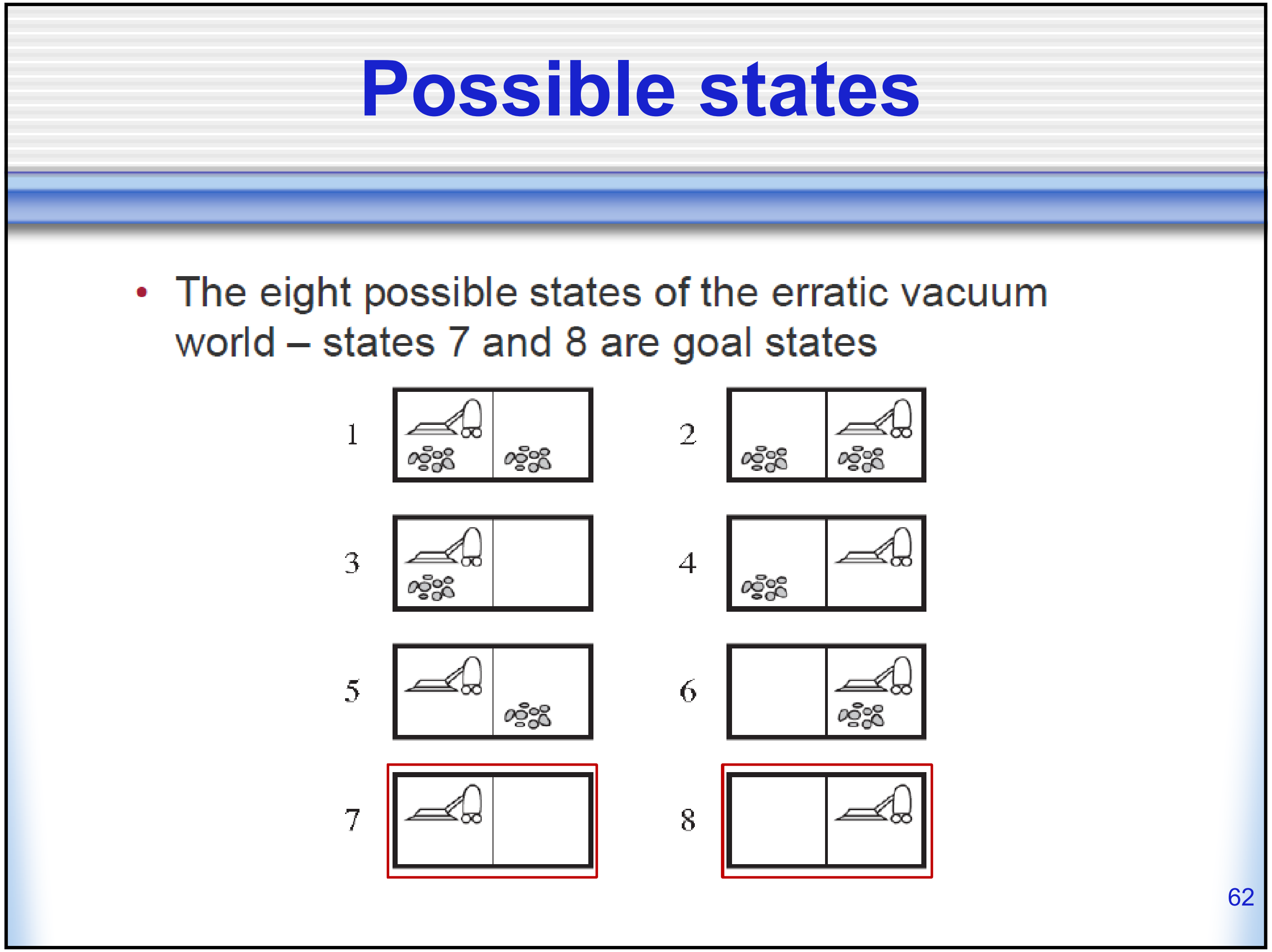
- *How many crossover, mutations*
- *How to encode the problem, fitness function*
- *One (more popular) vs. two child's*



# Nondeterministic/Uncertain actions

- What if the outcome of actions is non deterministic
- Erratic vacuum cleaner
  - When applied to a *dirty* square the square is *cleaned and adjacent square sometimes also*.
  - When applied to a *clean square*, *sometimes dirt is deposited* on that square

    ➔ need to have **contingency plan/strategy**

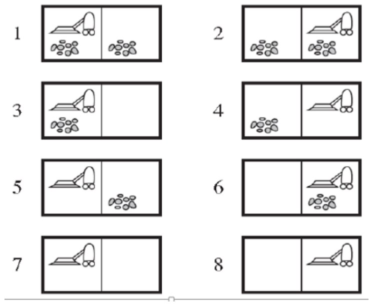


# Possible states

- The eight possible states of the erratic vacuum world – states 7 and 8 are goal states

# Multiple States

- The result of an action is a set of states
- *Suck* in state 1 returns the set {5,7}
- We also need to generalize the concept of solution, since for example, if we **start** in state **1** there is no single sequence of actions to solve the problem instead we need a contingency plan like:
  [*Suck,* **if** *State*=5 **then** [*Right, Suck*] **else** []]

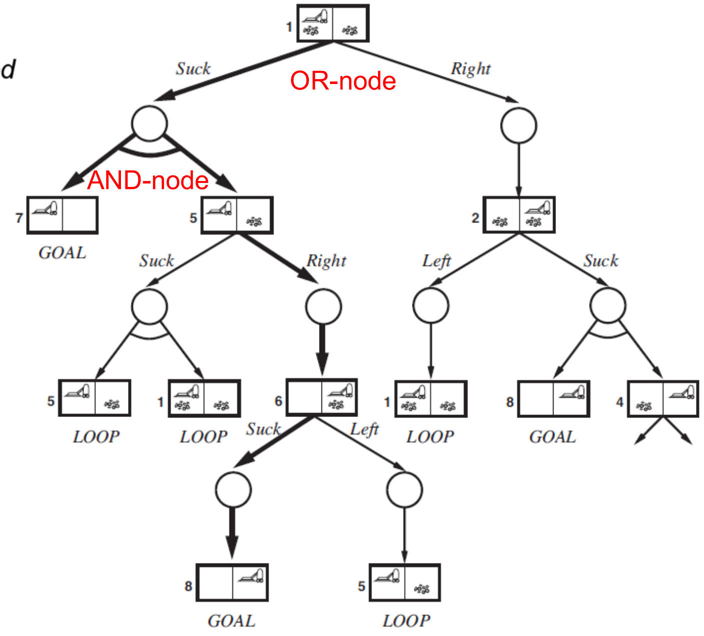


# AND-OR Search trees

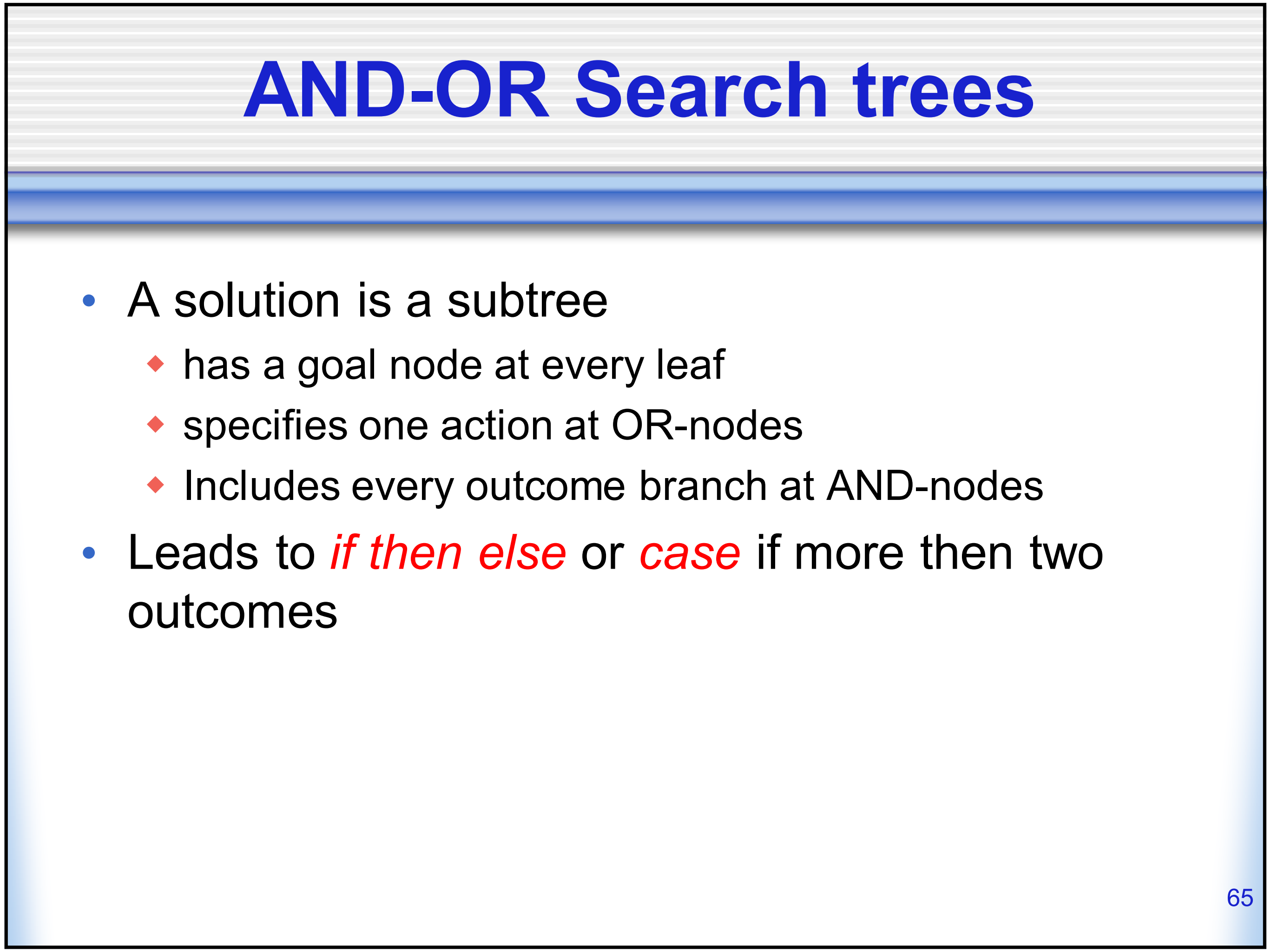
- Branching is also introduced by the environment choice of the outcome of actions.

When applied to a *dirty* square the square is *cleaned and adjacent square sometimes also*.

When applied to a *clean square*, *sometimes dirt is deposited* on that square

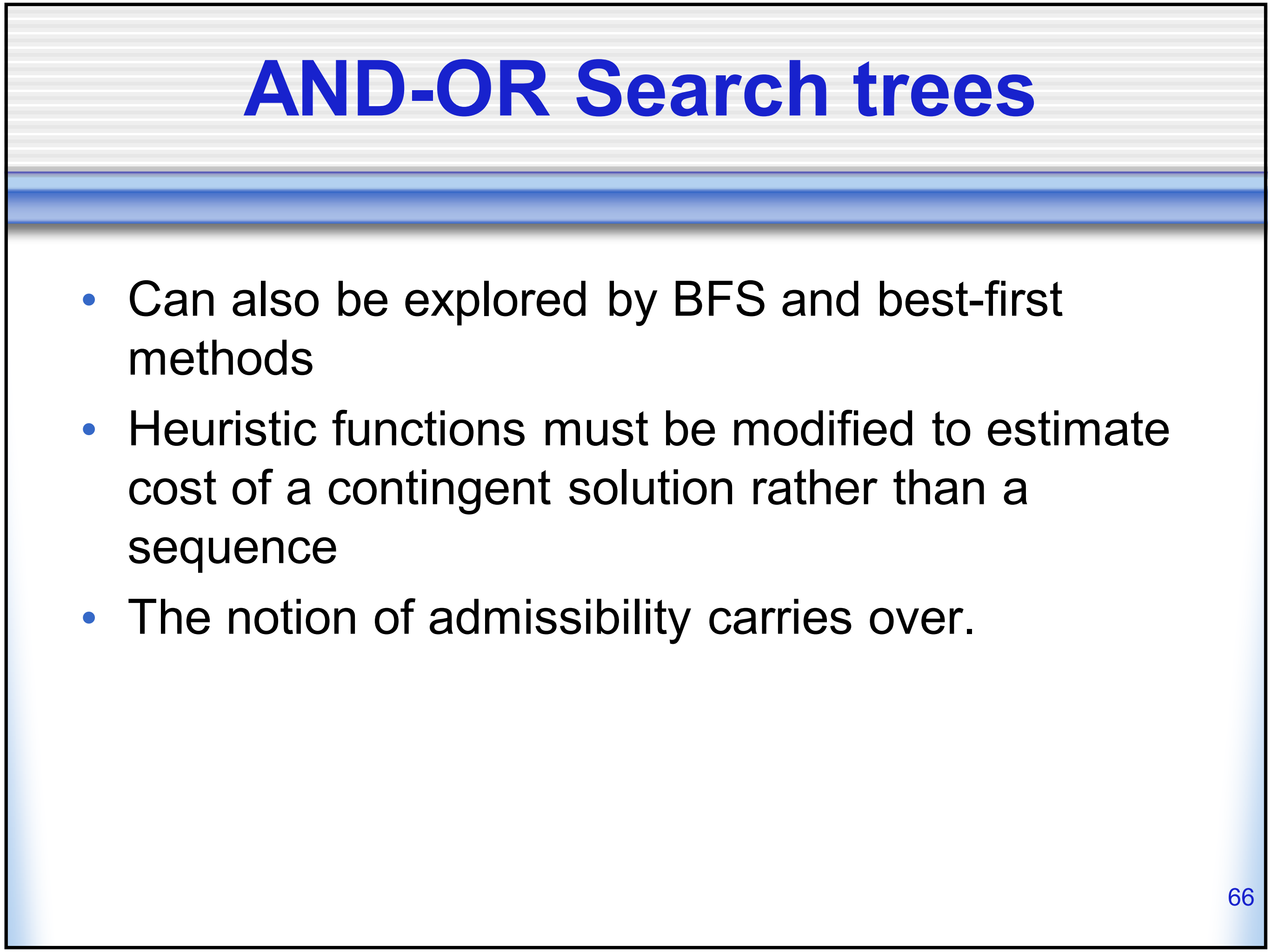


- This leads to AND-OR trees
- The bold path is the current plan



# AND-OR Search trees

- A solution is a subtree
  - has a goal node at every leaf
  - specifies one action at OR-nodes
  - Includes every outcome branch at AND-nodes
- Leads to *if then else* or *case* if more then two outcomes

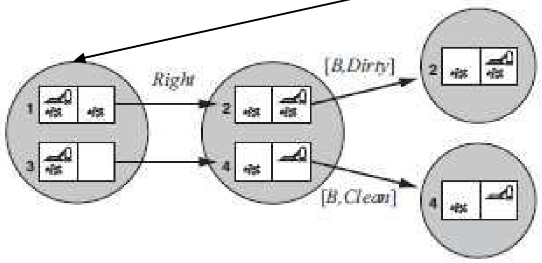


# AND-OR Search trees

- Can also be explored by BFS and best-first methods
- Heuristic functions must be modified to estimate cost of a contingent solution rather than a sequence
- The notion of admissibility carries over.

# Partial Observable Env.

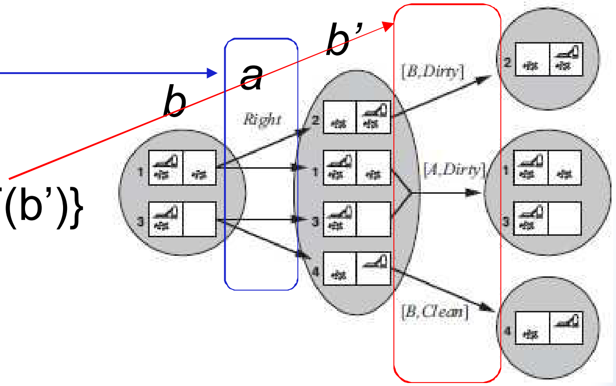
- The vacuum cleaner has only partial information, e.g., if he is in the left square he does not see the state of the right square.
  If the initial state is left and dirt, we have a **belief state** rather than a physical state



- But we also have uncertain actions: Move action may fail

# Uncertain actions & partial observable

- Prediction:
  b’=Predict(b, a)
- Possible observations in b’
  Percepts(b’)={o: o=PERCEPT(b’)}

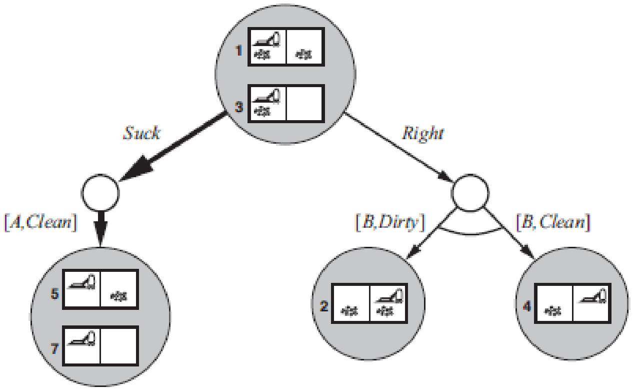


- Update of belief state:
  $b_o$= UPDATE(b’,o)= {s:o = PERCEPT(s) and s∈b’}
- Putting all together:

$$\text{RESULTS}(b,a) = \{b_o : b_o = \text{UPDATE}(\text{PREDICT}(b,a),o) \text{ and } o \in \text{POSSIBLE-PERCEPTS}(\text{PREDICT}(b,a))\}$$

# Structure

- Can use different search structures
- E.g. And-Or-Graphs



30.10.2022

# Intelligent Autonomous Agents

## and Cognitive Robotics:

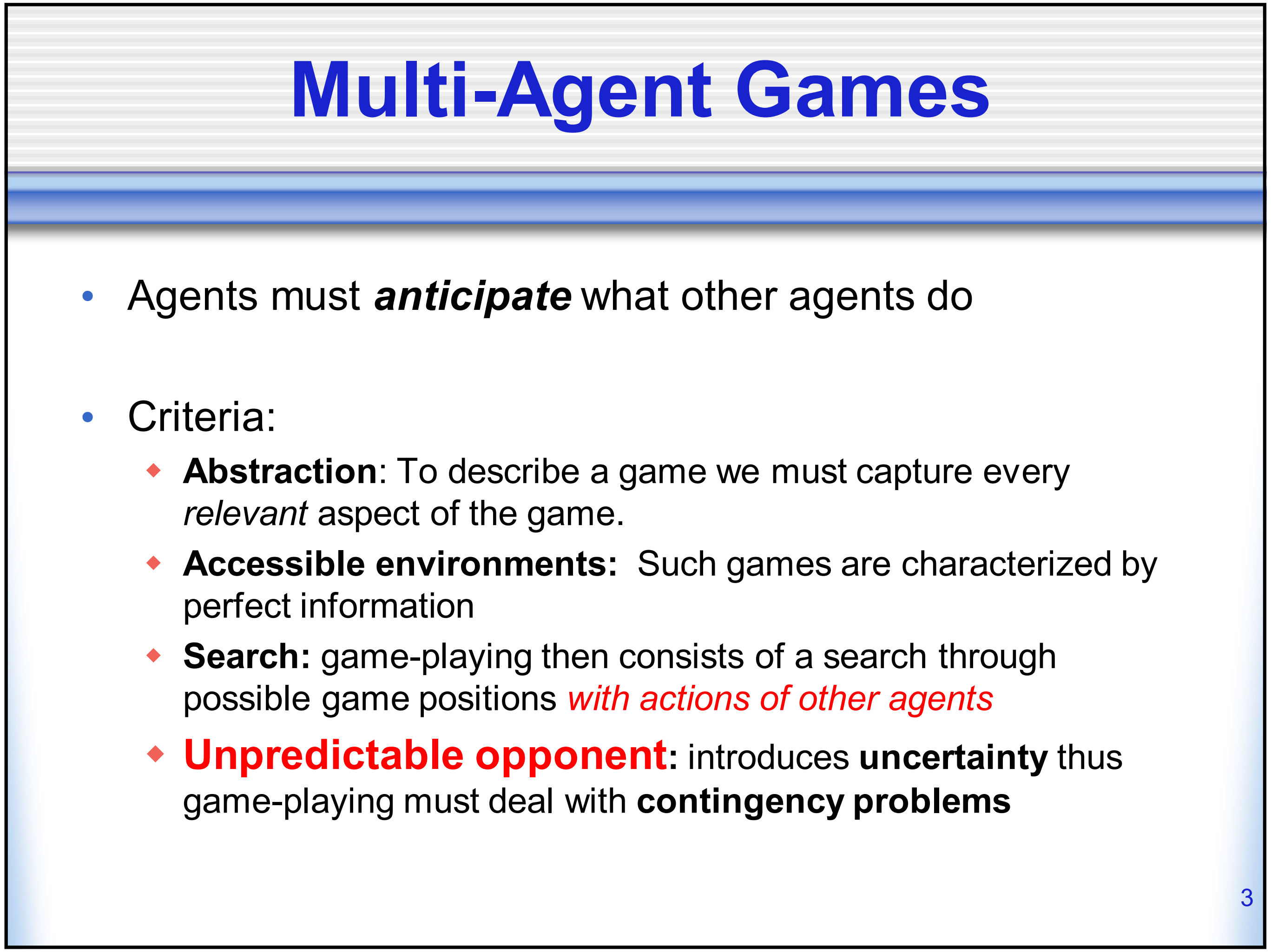
### Adversarial Agents

Ralf Möller, Rainer Marrone
Hamburg University of Technology

# Adversarial Agents

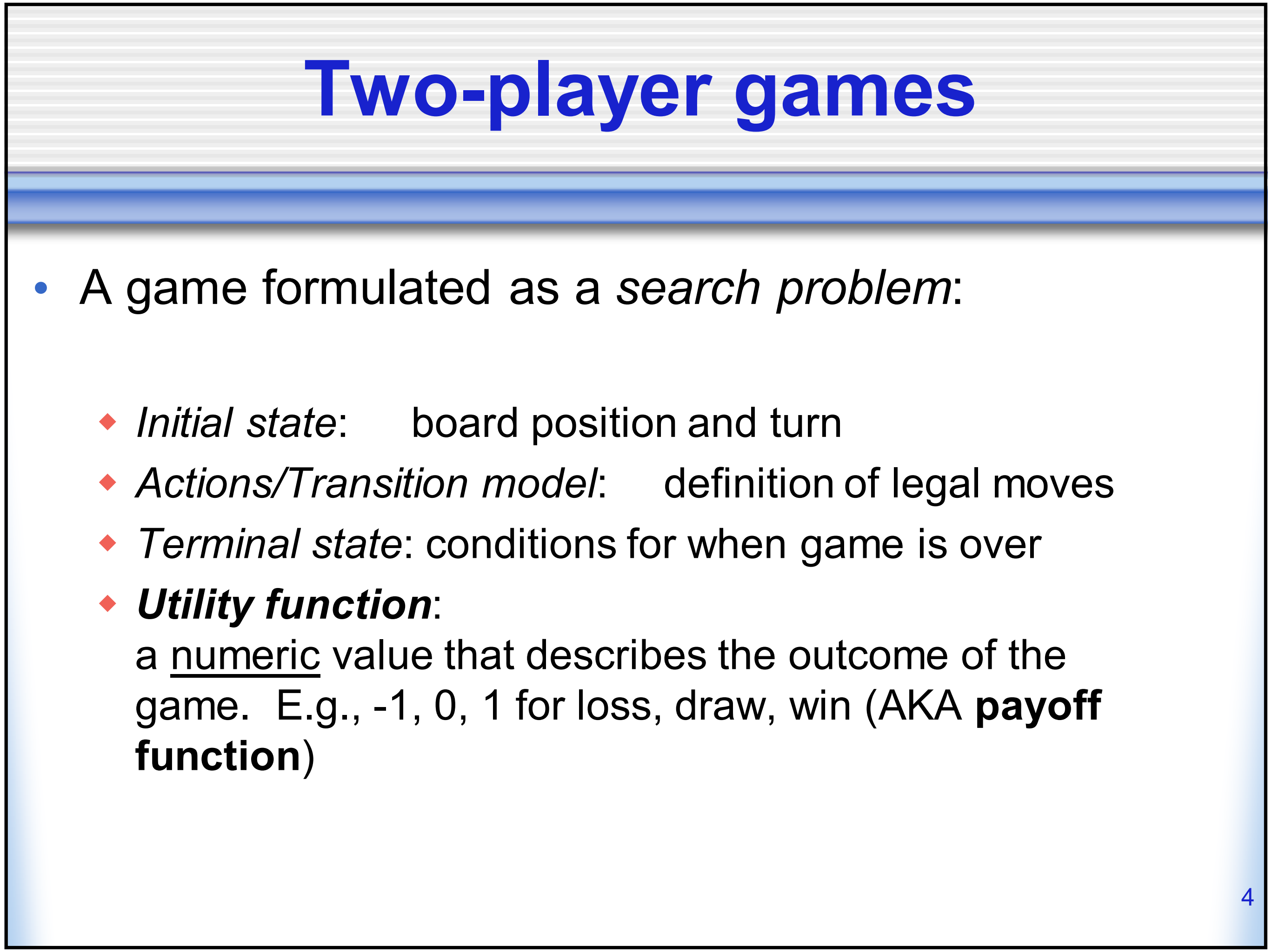
- In this chapter we cover **competitive environments**, in which the agents goals are in conflict, giving rise to adversarial search problems often known as games.

- Mathematical game theory, a branch of economics, views any multi-agent environment as a game, regardless of whether the agents are cooperative or competitive.



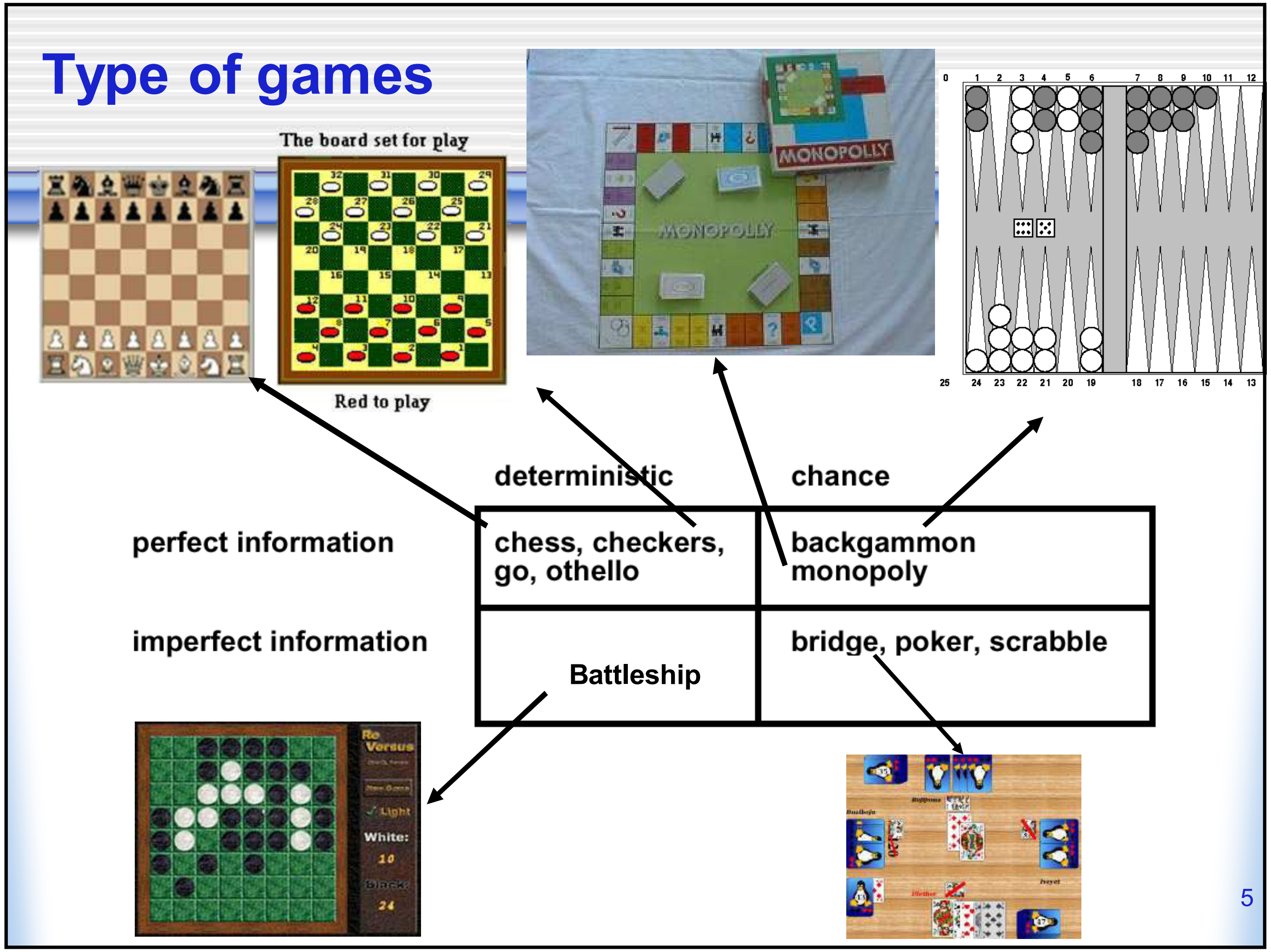
# Multi-Agent Games

- Agents must ***anticipate*** what other agents do
- Criteria:
  - **Abstraction**: To describe a game we must capture every *relevant* aspect of the game.
  - **Accessible environments:** Such games are characterized by perfect information
  - **Search:** game-playing then consists of a search through possible game positions *with actions of other agents*
  - **Unpredictable opponent:** introduces **uncertainty** thus game-playing must deal with **contingency problems**



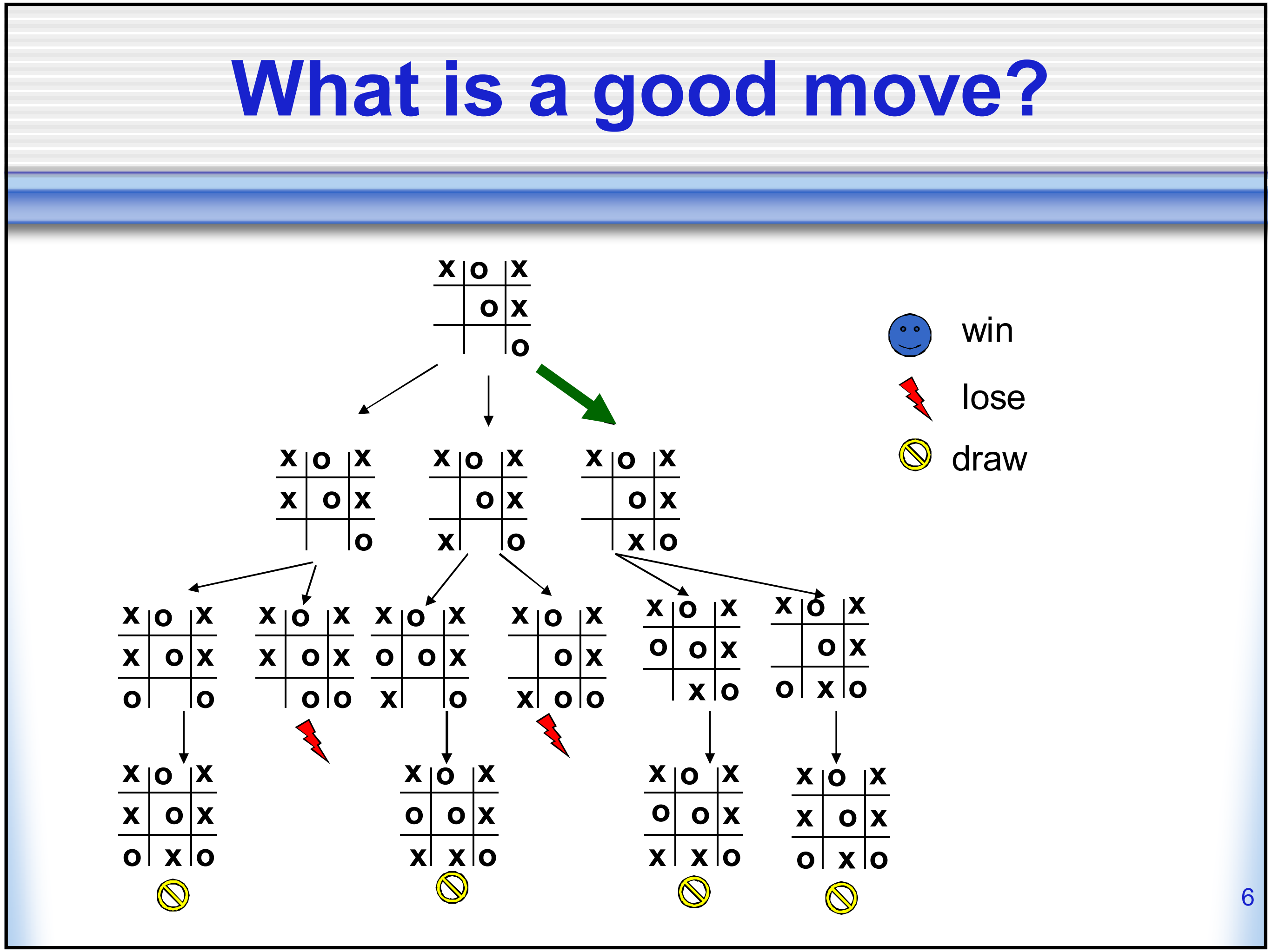
# Two-player games

- A game formulated as a *search problem*:
  - *Initial state*: board position and turn
  - *Actions/Transition model*: definition of legal moves
  - *Terminal state*: conditions for when game is over
  - ***Utility function***: a numeric value that describes the outcome of the game. E.g., -1, 0, 1 for loss, draw, win (AKA **payoff function**)



# Type of games

| | deterministic | chance |
|---|---|---|
| perfect information | chess, checkers, go, othello | backgammon monopoly |
| imperfect information | Battleship | bridge, poker, scrabble |



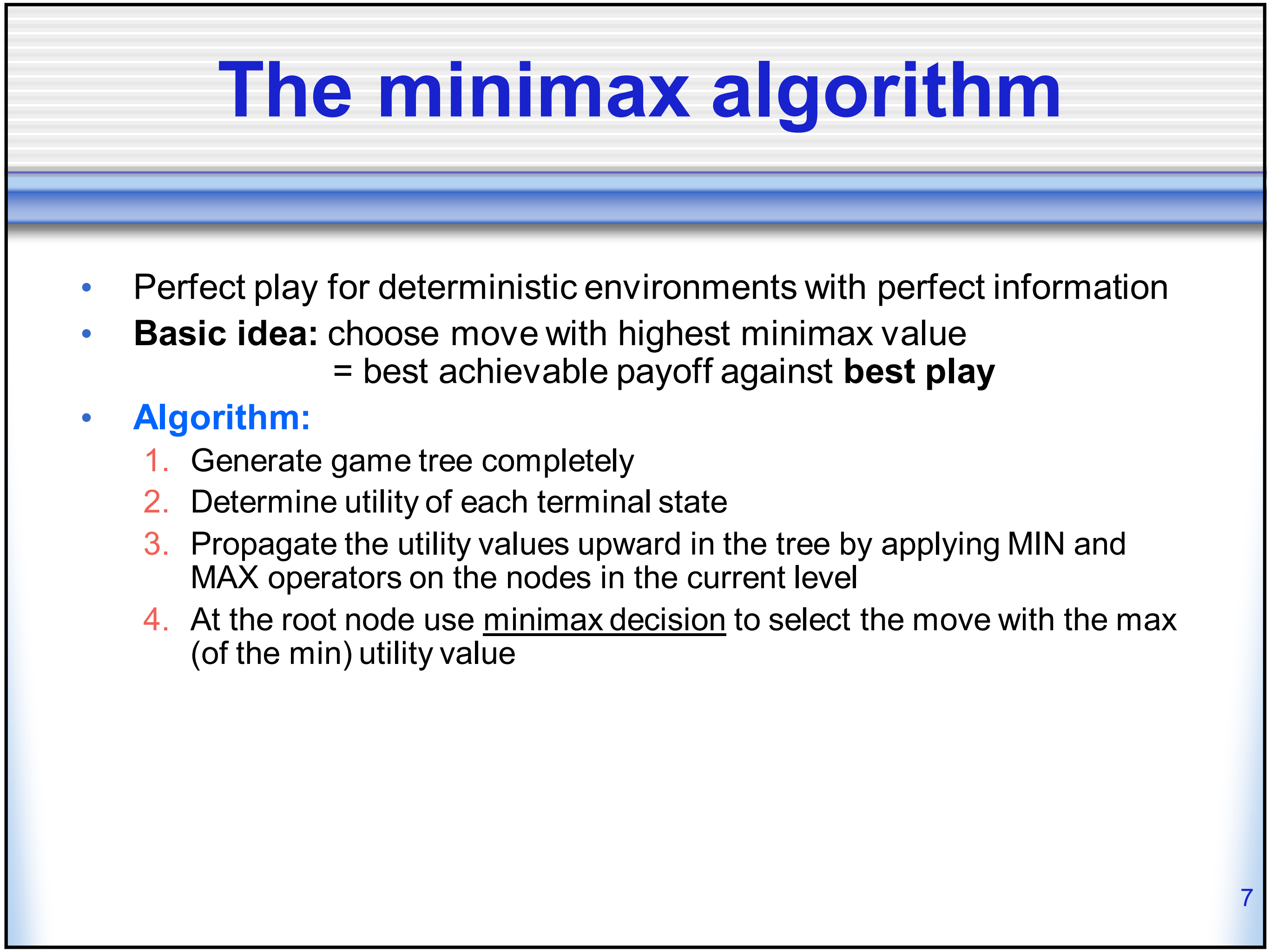
# What is a good move?

☺ win

⚡ lose

🚫 draw

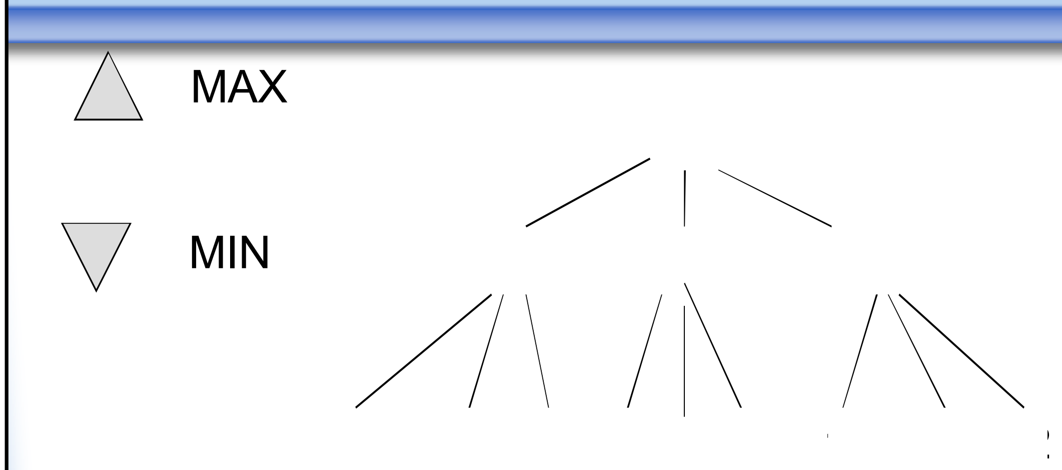
6



# The minimax algorithm

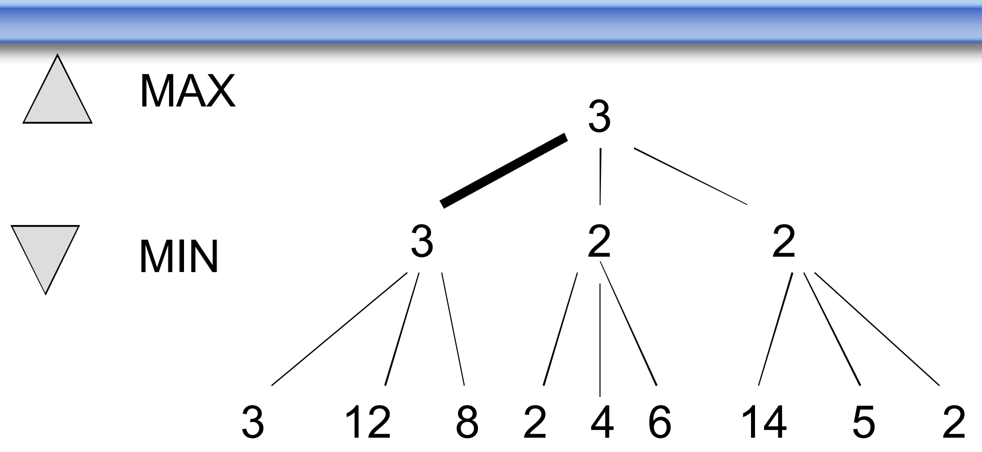
- Perfect play for deterministic environments with perfect information
- **Basic idea:** choose move with highest minimax value
  = best achievable payoff against **best play**
- **Algorithm:**
  1. Generate game tree completely
  2. Determine utility of each terminal state
  3. Propagate the utility values upward in the tree by applying MIN and MAX operators on the nodes in the current level
  4. At the root node use minimax decision to select the move with the max (of the min) utility value

# Minimax algorithm



- Minimize opponent’s chance
- Maximize your chance

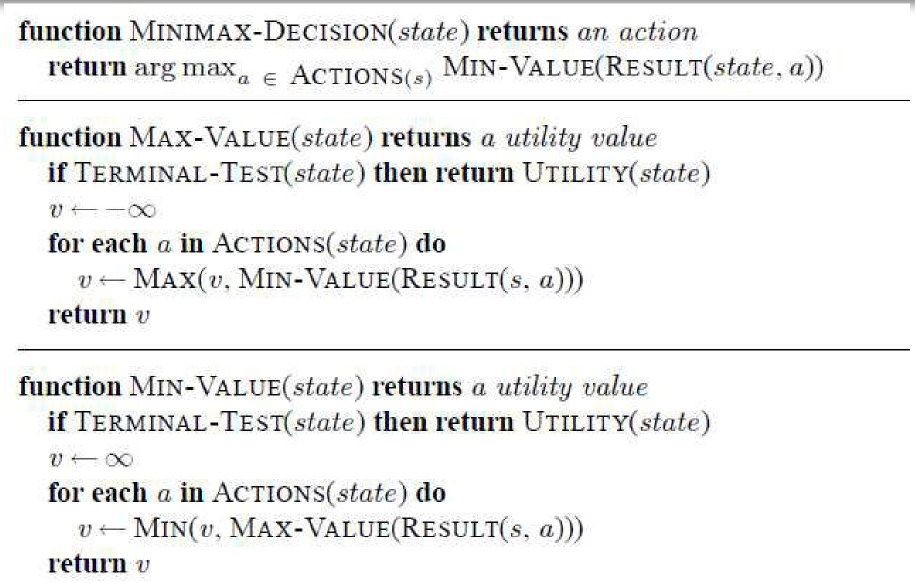
# Minimax



- Minimize opponent’s chance
- Maximize your chance

$$\text{MINIMAX-VALUE}(n) = \begin{cases} \text{UTILITY}(n) & \text{if } n \text{ is a terminal state} \\ \max_{s \in Successors(n)} \text{MINIMAX-VALUE}(s) & \text{if } n \text{ is a MAX node} \\ \min_{s \in Successors(n)} \text{MINIMAX-VALUE}(s) & \text{if } n \text{ is a MIN node.} \end{cases}$$

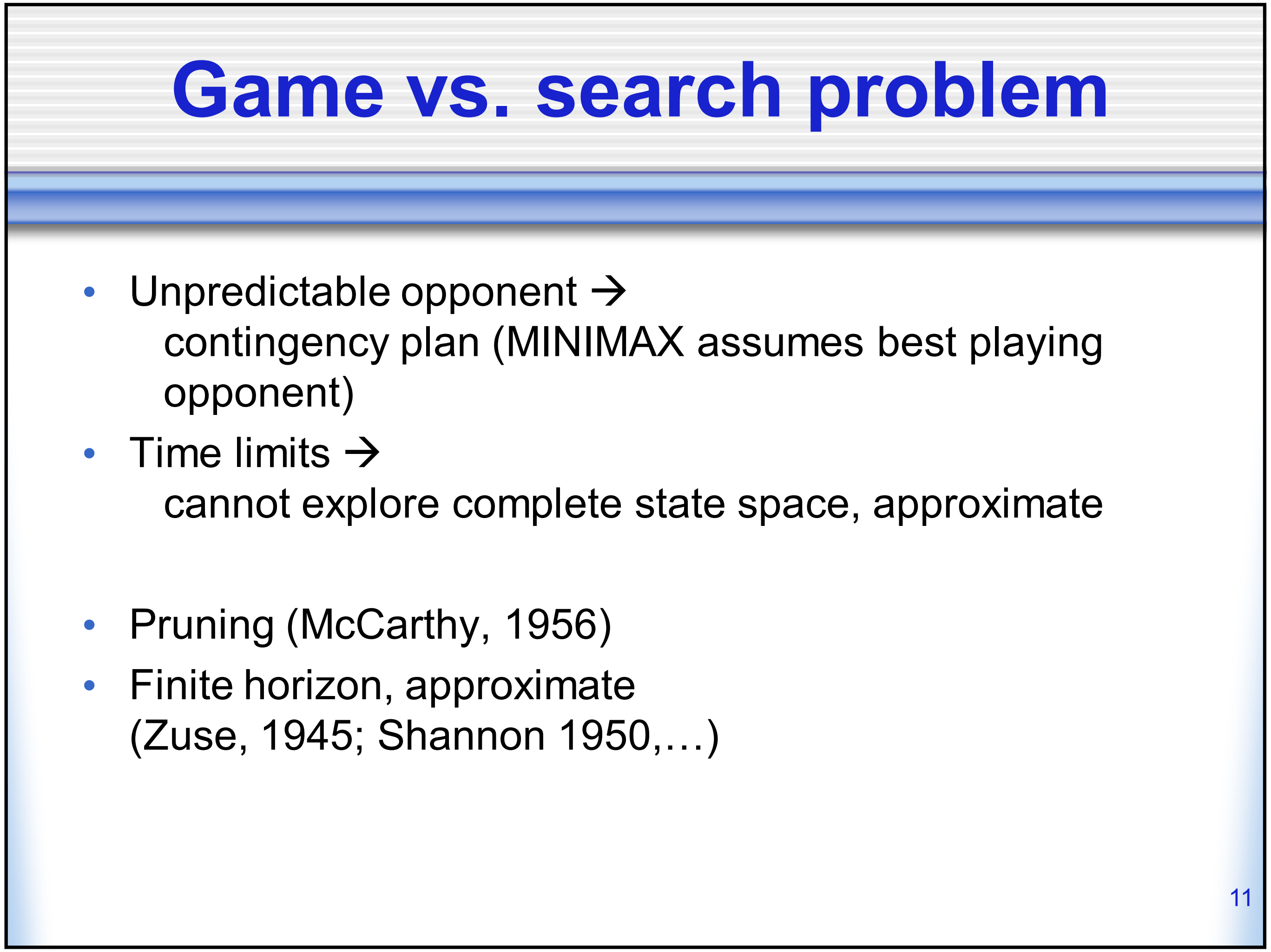
# Minimax: Recursive implementation



```
function MINIMAX-DECISION(state) returns an action
   return $\arg\max_{a \in \text{ACTIONS}(s)}$ MIN-VALUE(RESULT(state, a))

function MAX-VALUE(state) returns a utility value
   if TERMINAL-TEST(state) then return UTILITY(state)
   $v \leftarrow -\infty$
   for each a in ACTIONS(state) do
      $v \leftarrow$ MAX(v, MIN-VALUE(RESULT(s, a)))
   return v

function MIN-VALUE(state) returns a utility value
   if TERMINAL-TEST(state) then return UTILITY(state)
   $v \leftarrow \infty$
   for each a in ACTIONS(state) do
      $v \leftarrow$ MIN(v, MAX-VALUE(RESULT(s, a)))
   return v
```

**Complete:** Yes, for finite state-space **Time complexity:** O($b^m$)

**Optimal:** Yes, if winning is the goal **Space complexity:** O(bm) or O(m)



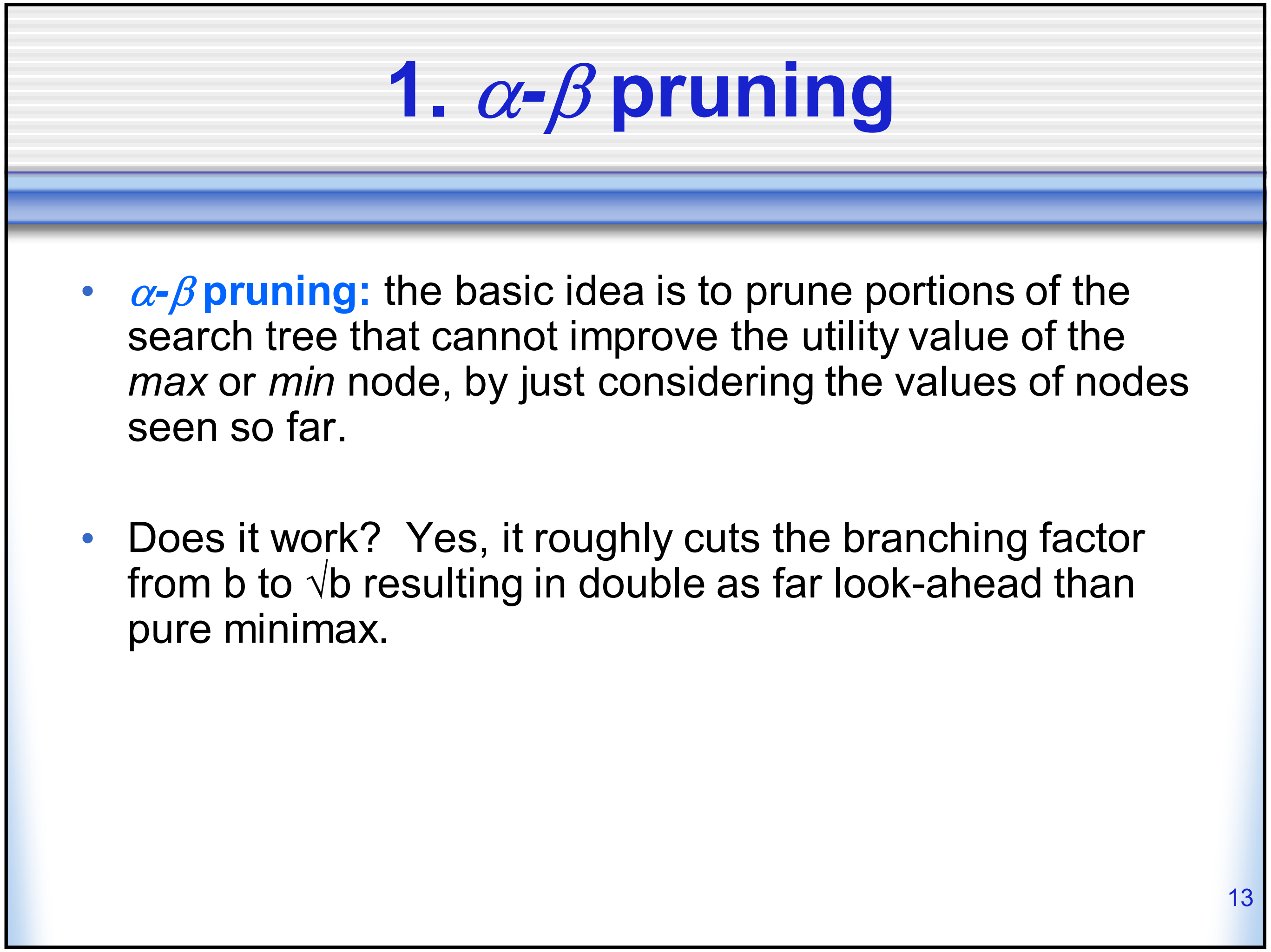
# Game vs. search problem

- Unpredictable opponent →
  contingency plan (MINIMAX assumes best playing opponent)
- Time limits →
  cannot explore complete state space, approximate
- Pruning (McCarthy, 1956)
- Finite horizon, approximate
  (Zuse, 1945; Shannon 1950,…)

# Searching for the next move

- **Complexity:** many games have a huge search space
  - **Chess:** $b = 35, m=100 \Rightarrow nodes = 35^{100}$ means more than $10^{154}$ in a search tree and more than $10^{40}$ nodes in a search graph. Take several millennia to compute moves. $35^{100} = 10^{\log(35^{100})} = 10^{100*\log(35)} = 10^{100*1,54}= 10^{154}$
- **Resource (e.g., time, memory) limit:** optimal solution not feasible/possible, thus must approximate

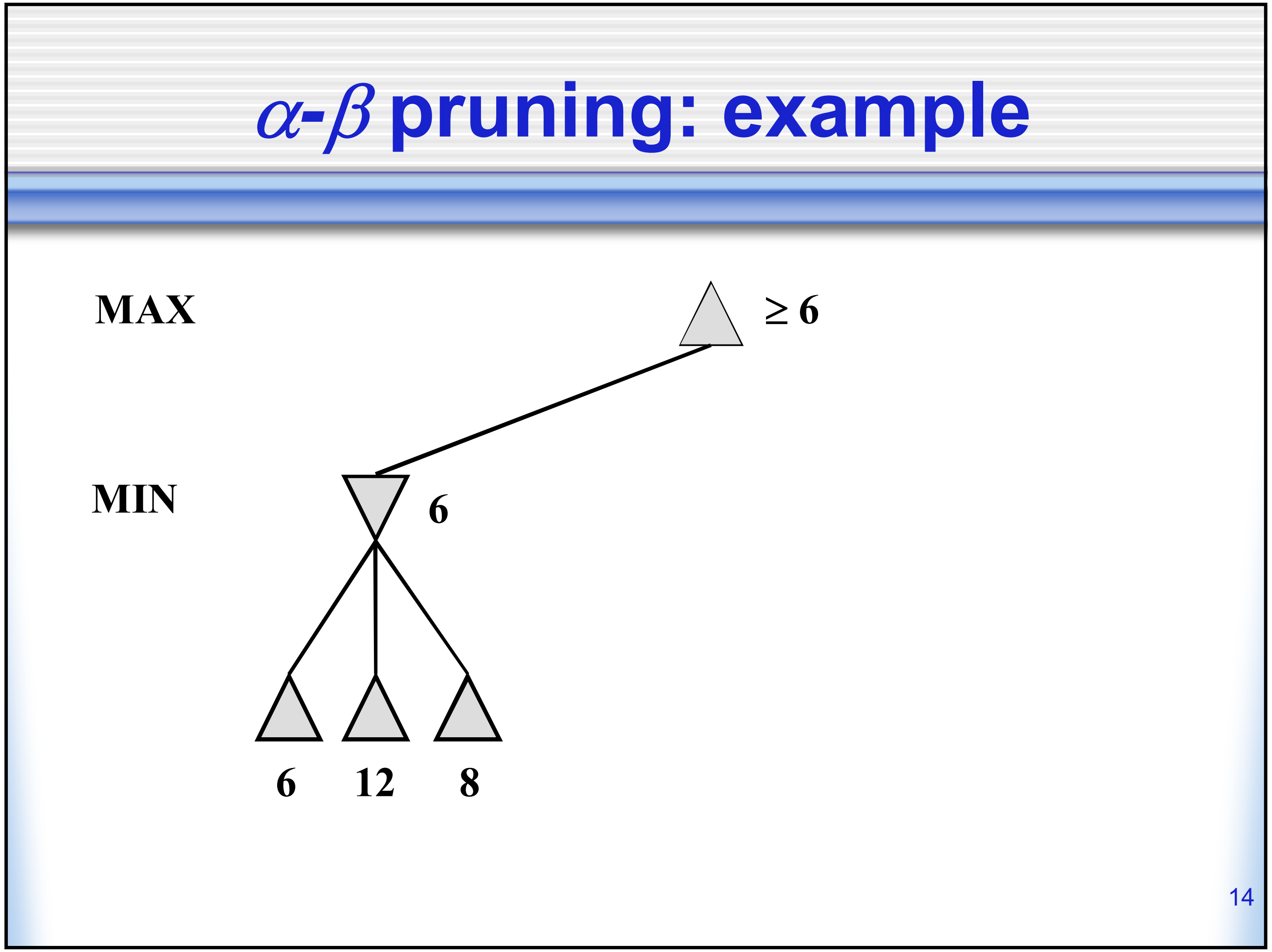
1. **Pruning:** makes the search more efficient by discarding portions of the search tree that cannot improve quality.
2. **Evaluation functions:** heuristics to evaluate utility of a state without exhaustive search.



# 1. $\alpha$-$\beta$ pruning

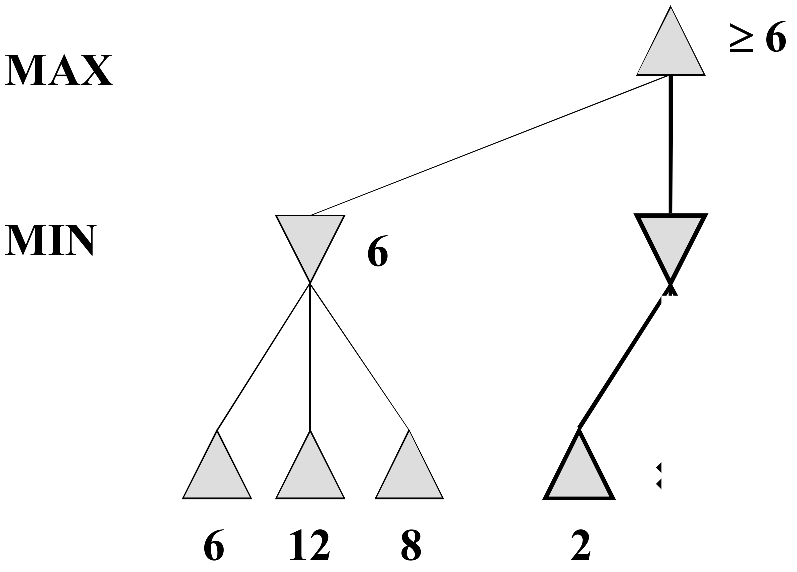
- **$\alpha$-$\beta$ pruning:** the basic idea is to prune portions of the search tree that cannot improve the utility value of the *max* or *min* node, by just considering the values of nodes seen so far.

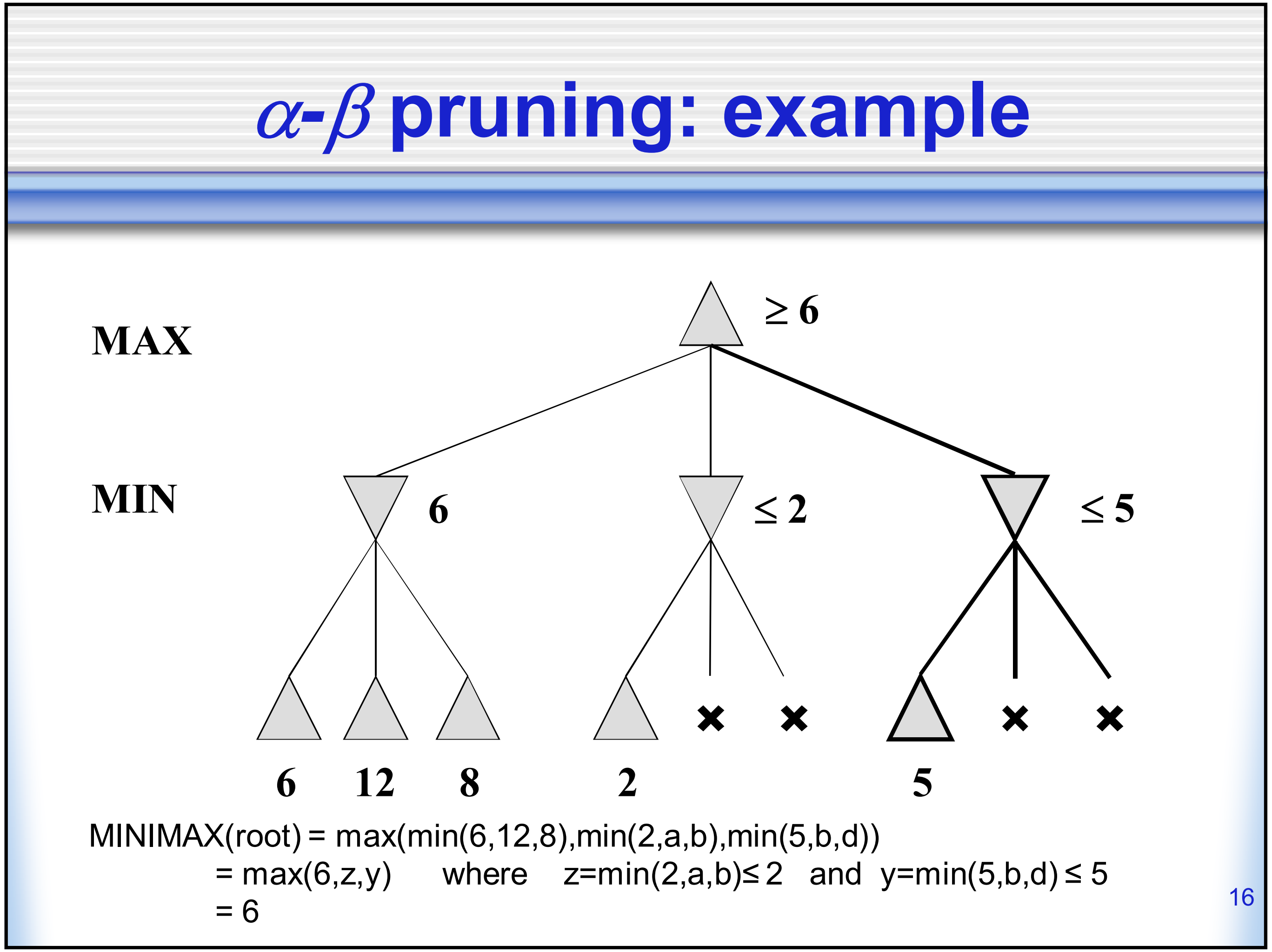
- Does it work? Yes, it roughly cuts the branching factor from b to √b resulting in double as far look-ahead than pure minimax.



# $\alpha$-$\beta$ pruning: example

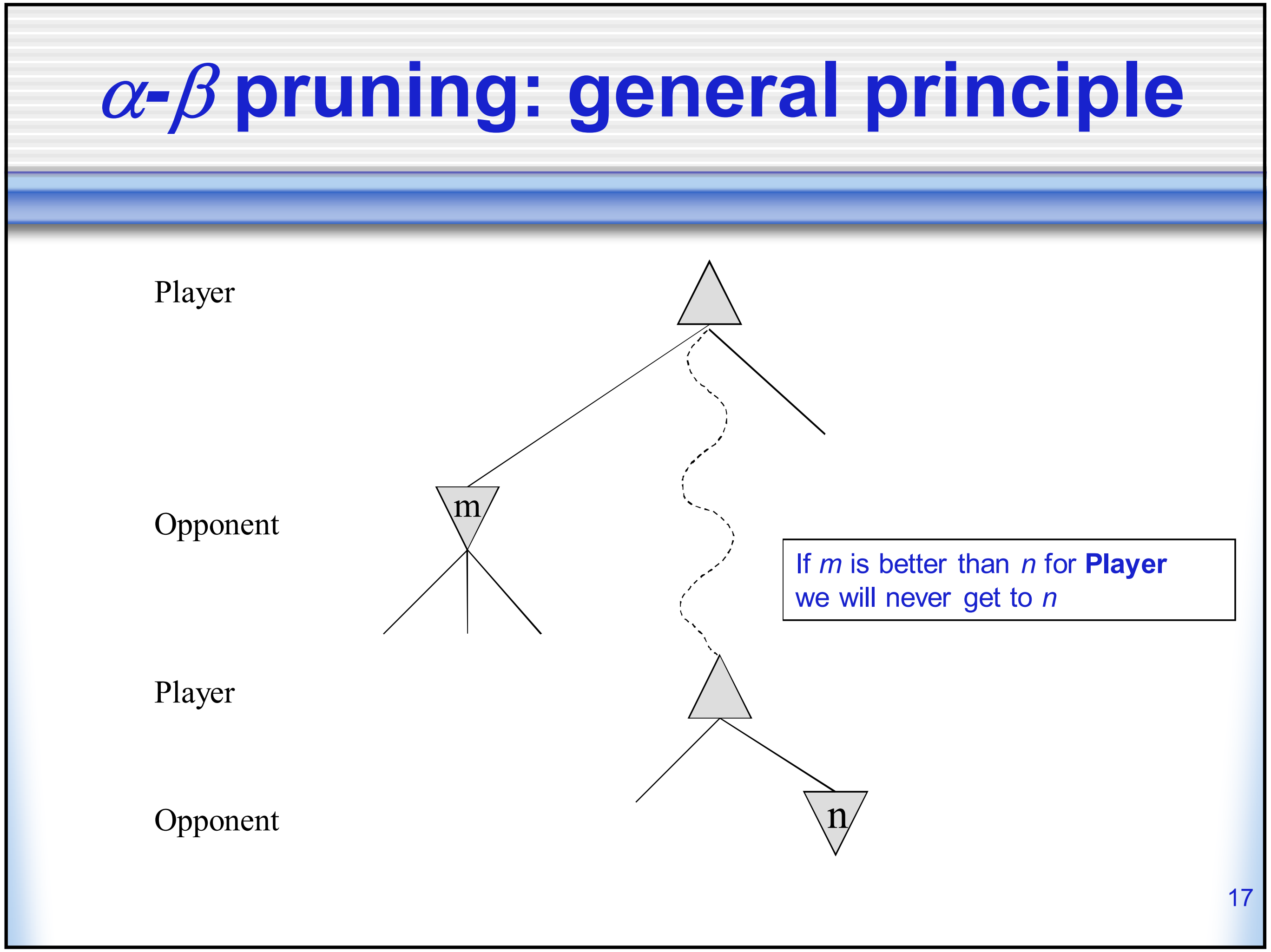
# $\alpha$-$\beta$ pruning: example





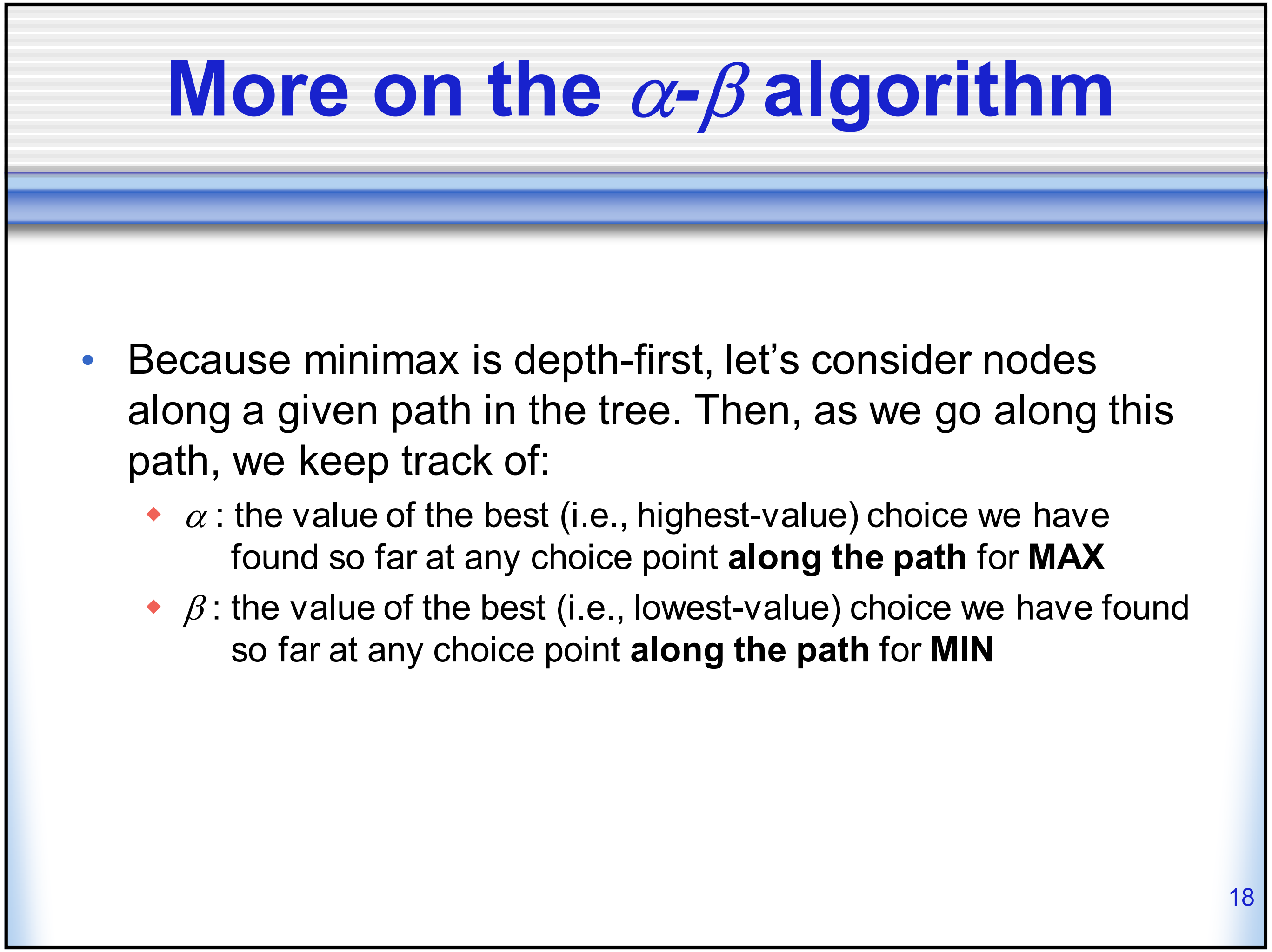
# $\alpha$-$\beta$ pruning: example

MINIMAX(root) = max(min(6,12,8),min(2,a,b),min(5,b,d))
= max(6,z,y) where z=min(2,a,b)≤ 2 and y=min(5,b,d) ≤ 5
= 6



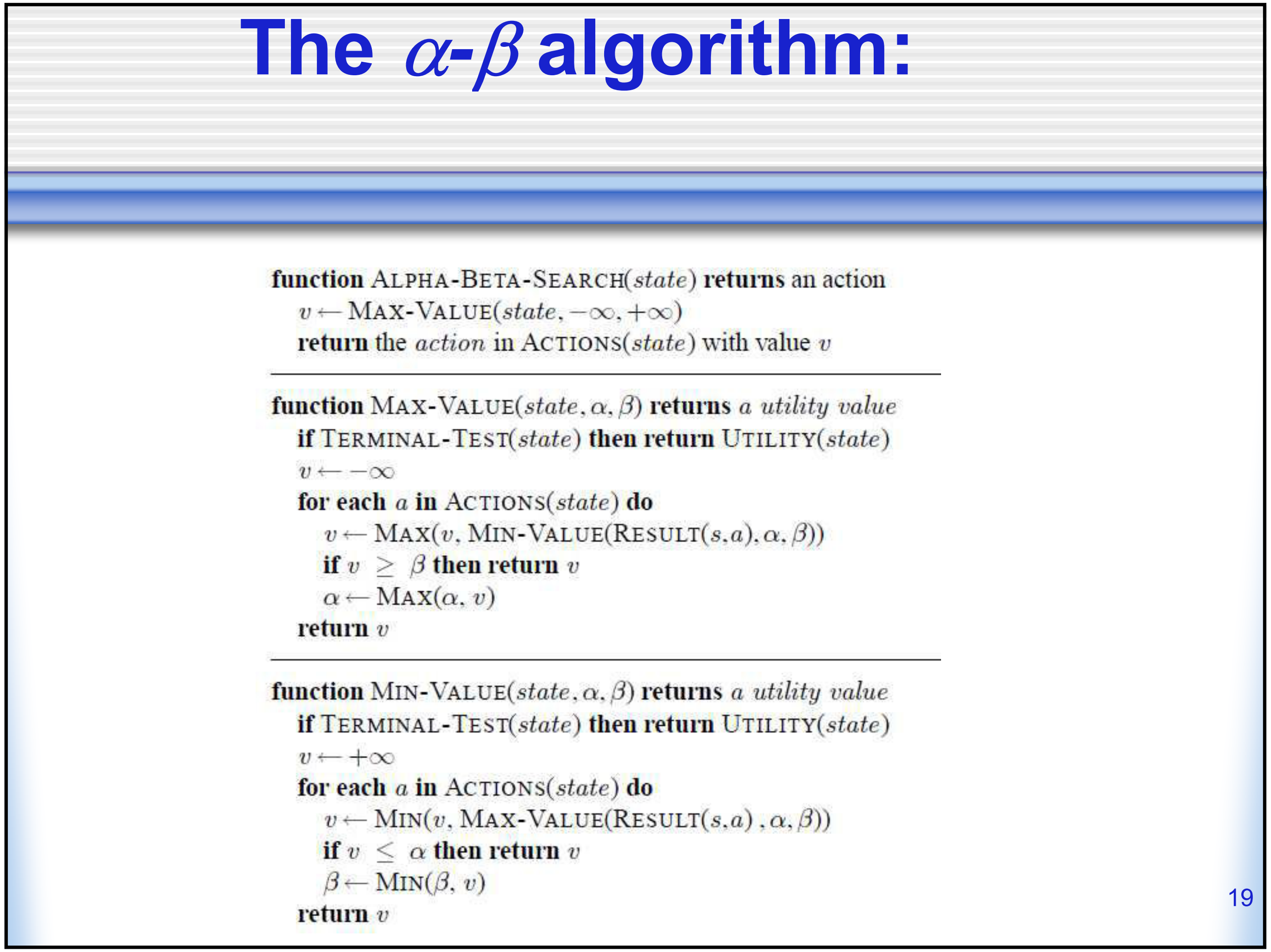
# $\alpha$-$\beta$ pruning: general principle

If *m* is better than *n* for **Player** we will never get to *n*



# More on the $\alpha$-$\beta$ algorithm

- Because minimax is depth-first, let's consider nodes along a given path in the tree. Then, as we go along this path, we keep track of:
  - $\alpha$ : the value of the best (i.e., highest-value) choice we have found so far at any choice point **along the path** for **MAX**
  - $\beta$ : the value of the best (i.e., lowest-value) choice we have found so far at any choice point **along the path** for **MIN**

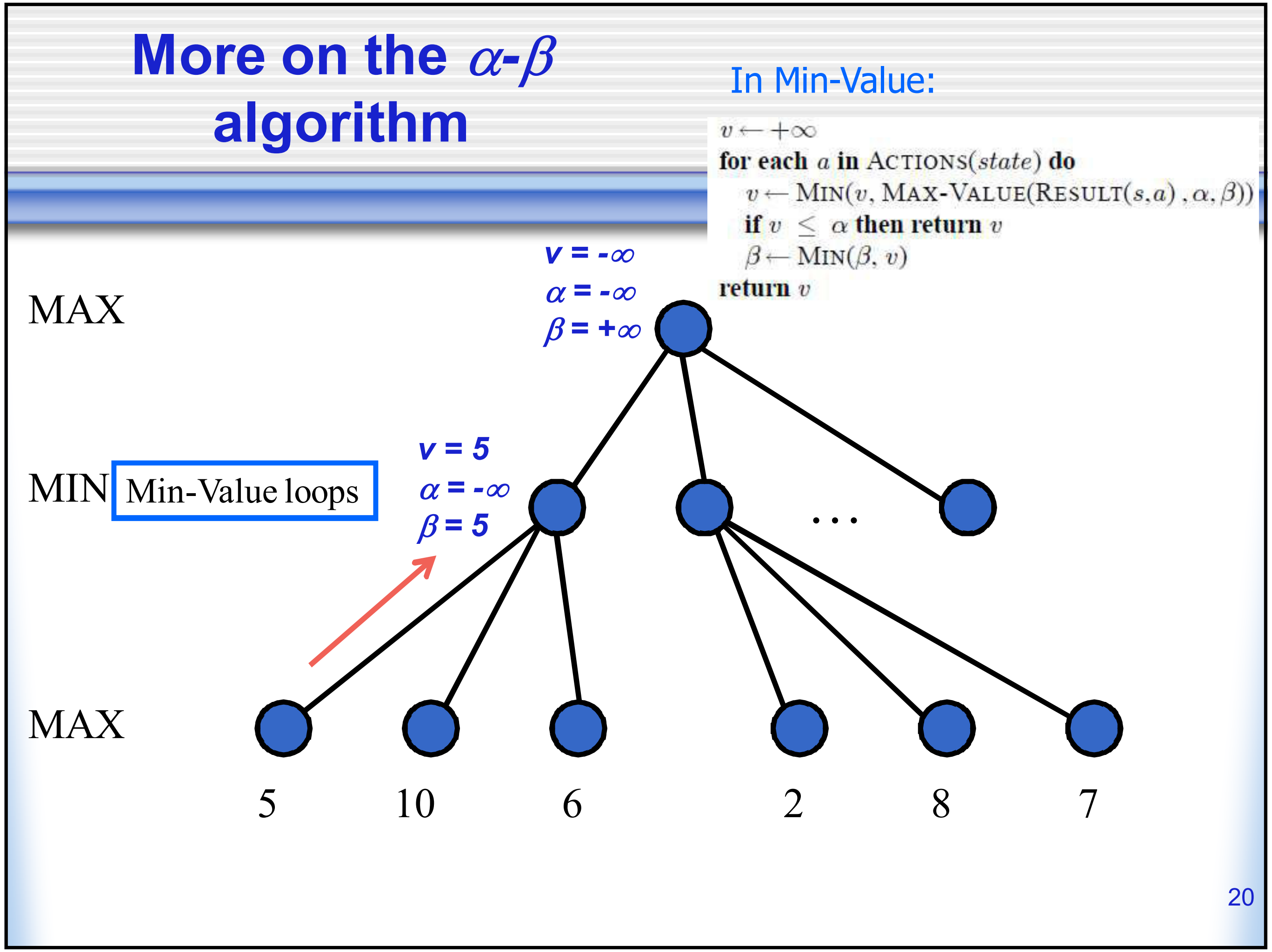


# The $\alpha$-$\beta$ algorithm:

```
function ALPHA-BETA-SEARCH(state) returns an action
   $v \leftarrow$ MAX-VALUE($state, -\infty, +\infty$)
   return the action in ACTIONS(state) with value $v$
```

```
function MAX-VALUE($state, \alpha, \beta$) returns a utility value
   if TERMINAL-TEST(state) then return UTILITY(state)
   $v \leftarrow -\infty$
   for each $a$ in ACTIONS(state) do
      $v \leftarrow$ MAX($v$, MIN-VALUE(RESULT($s,a$), $\alpha, \beta$))
      if $v \geq \beta$ then return $v$
      $\alpha \leftarrow$ MAX($\alpha, v$)
   return $v$
```

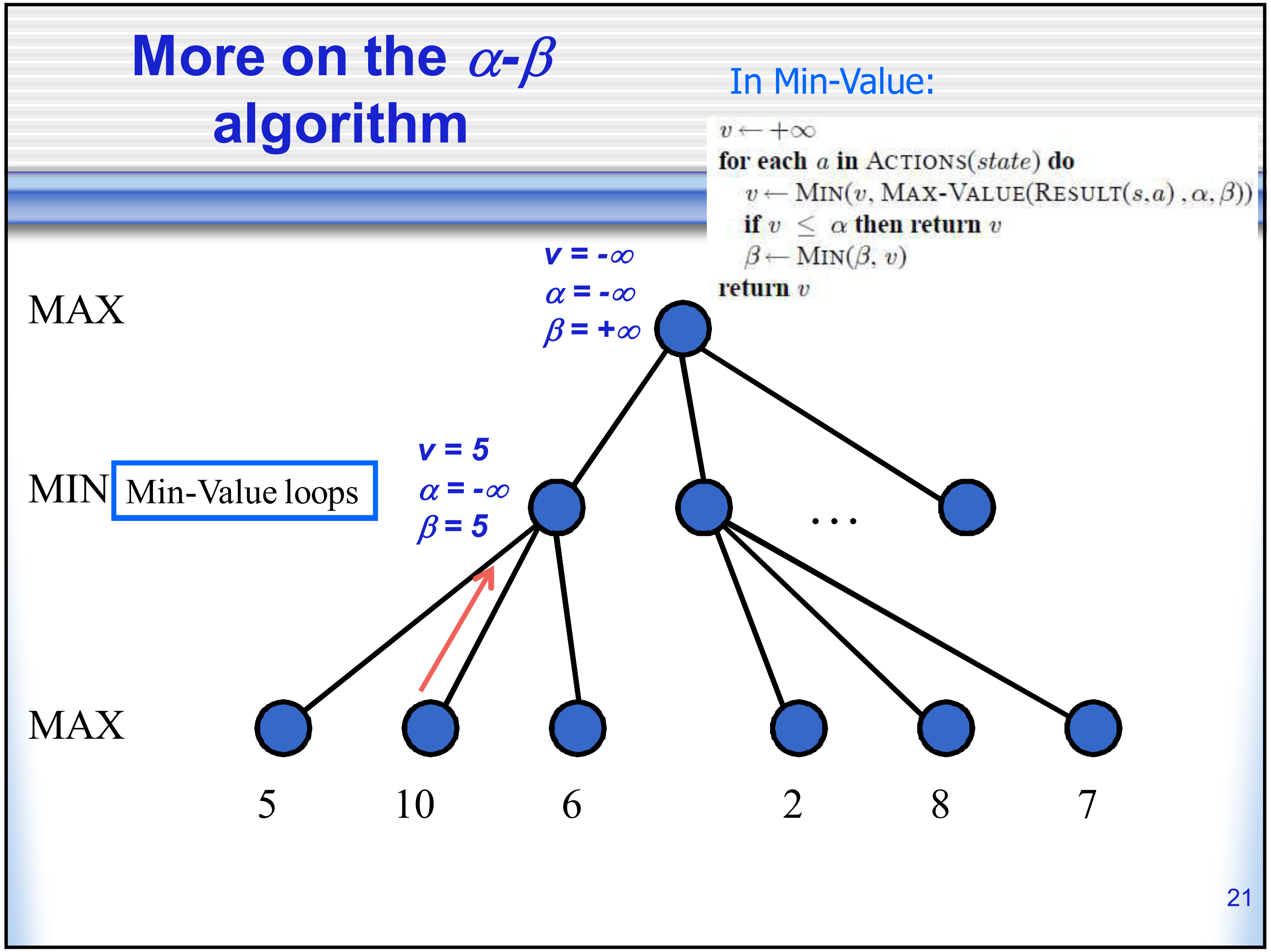
```
function MIN-VALUE($state, \alpha, \beta$) returns a utility value
   if TERMINAL-TEST(state) then return UTILITY(state)
   $v \leftarrow +\infty$
   for each $a$ in ACTIONS(state) do
      $v \leftarrow$ MIN($v$, MAX-VALUE(RESULT($s,a$) , $\alpha, \beta$))
      if $v \leq \alpha$ then return $v$
      $\beta \leftarrow$ MIN($\beta, v$)
   return $v$
```



## More on the $\alpha$-$\beta$ algorithm

In Min-Value:

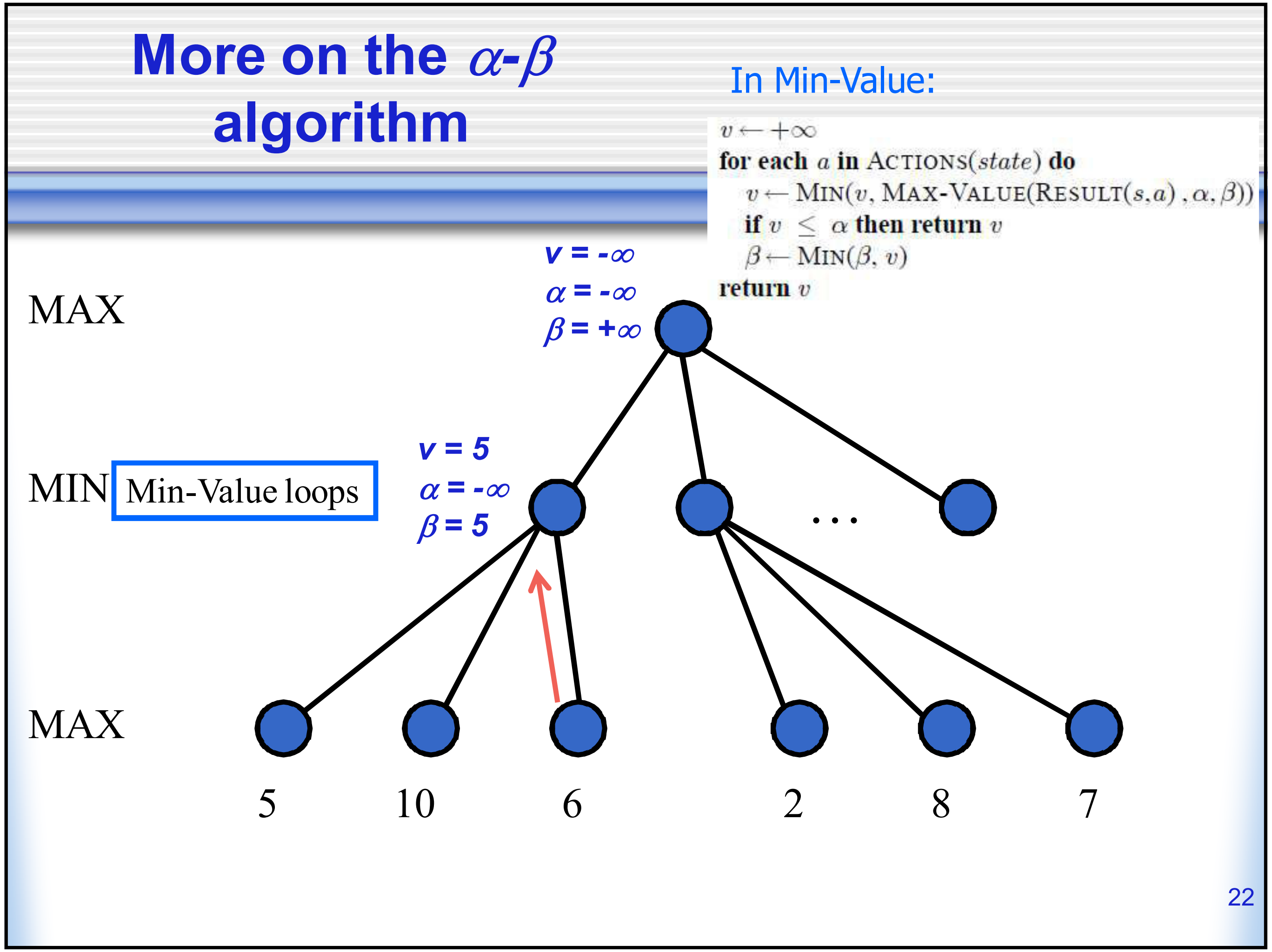
```
$v \leftarrow +\infty$
for each $a$ in ACTIONS($state$) do
    $v \leftarrow$ MIN($v$, MAX-VALUE(RESULT($s,a$) , $\alpha$, $\beta$))
    if $v \leq \alpha$ then return $v$
    $\beta \leftarrow$ MIN($\beta$, $v$)
return $v$
```



## More on the $\alpha$-$\beta$ algorithm

In Min-Value:

```
$v \leftarrow +\infty$
for each $a$ in ACTIONS($state$) do
    $v \leftarrow$ MIN($v$, MAX-VALUE(RESULT($s,a$) , $\alpha$, $\beta$))
    if $v \leq \alpha$ then return $v$
    $\beta \leftarrow$ MIN($\beta$, $v$)
return $v$
```

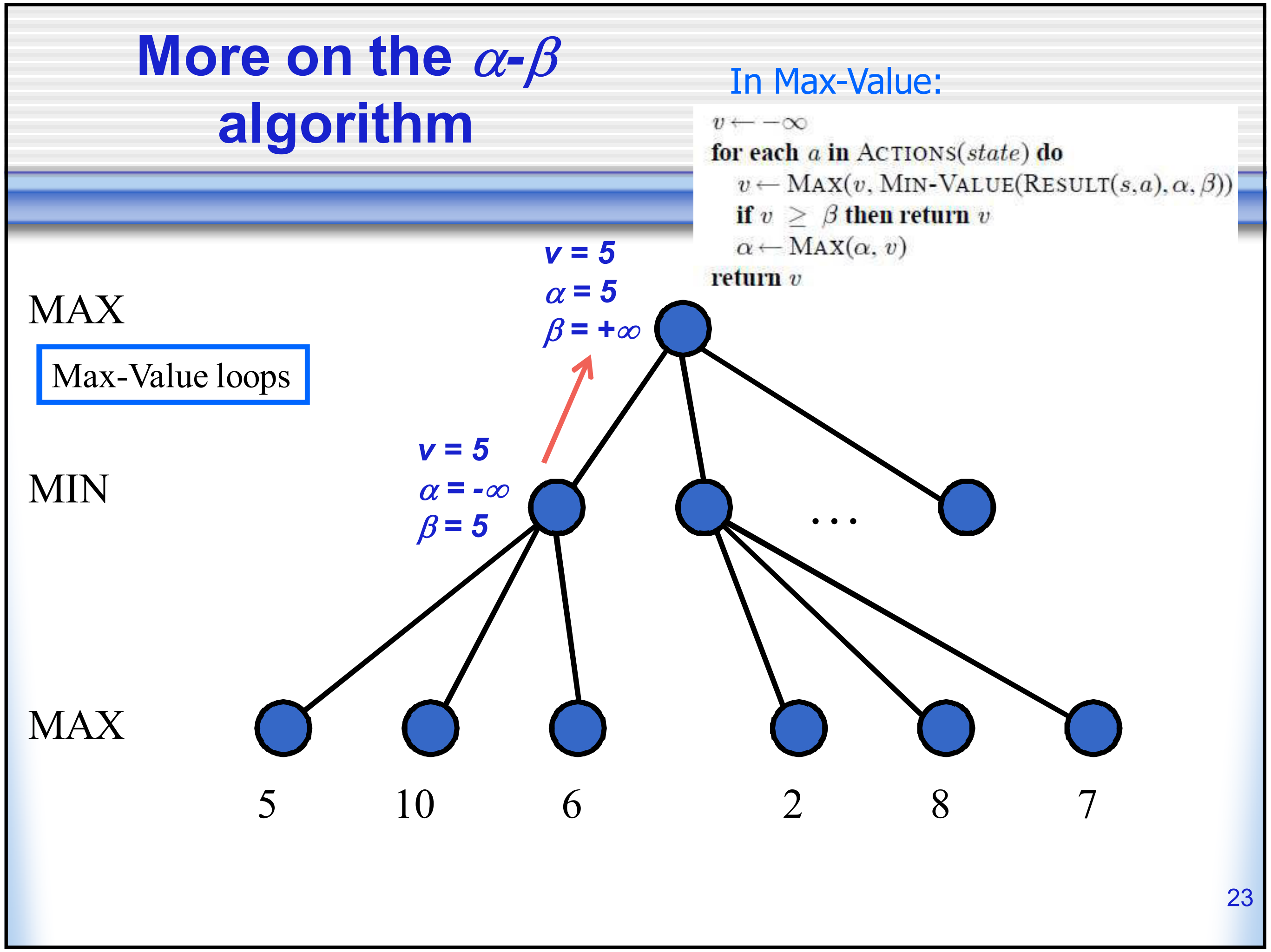


# More on the $\alpha$-$\beta$ algorithm

In Min-Value:

```
$v \leftarrow +\infty$
for each $a$ in ACTIONS($state$) do
    $v \leftarrow$ MIN($v$, MAX-VALUE(RESULT($s,a$) , $\alpha, \beta$))
    if $v \leq \alpha$ then return $v$
    $\beta \leftarrow$ MIN($\beta$, $v$)
return $v$
```

22

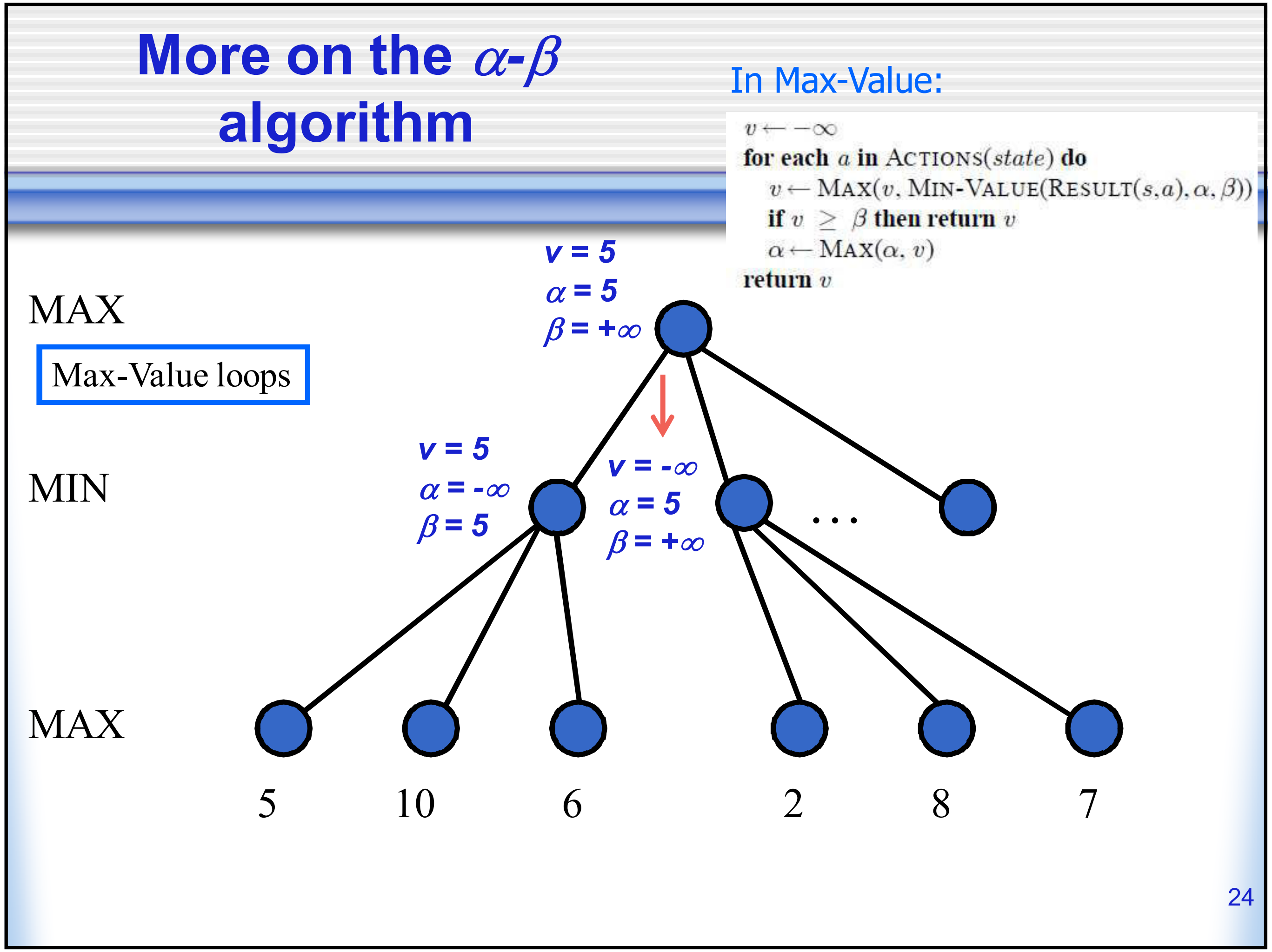


# More on the $\alpha$-$\beta$ algorithm

In Max-Value:

```
$v \leftarrow -\infty$
for each $a$ in ACTIONS($state$) do
   $v \leftarrow$ MAX($v$, MIN-VALUE(RESULT($s$,$a$), $\alpha$, $\beta$))
   if $v \geq \beta$ then return $v$
   $\alpha \leftarrow$ MAX($\alpha$, $v$)
return $v$
```

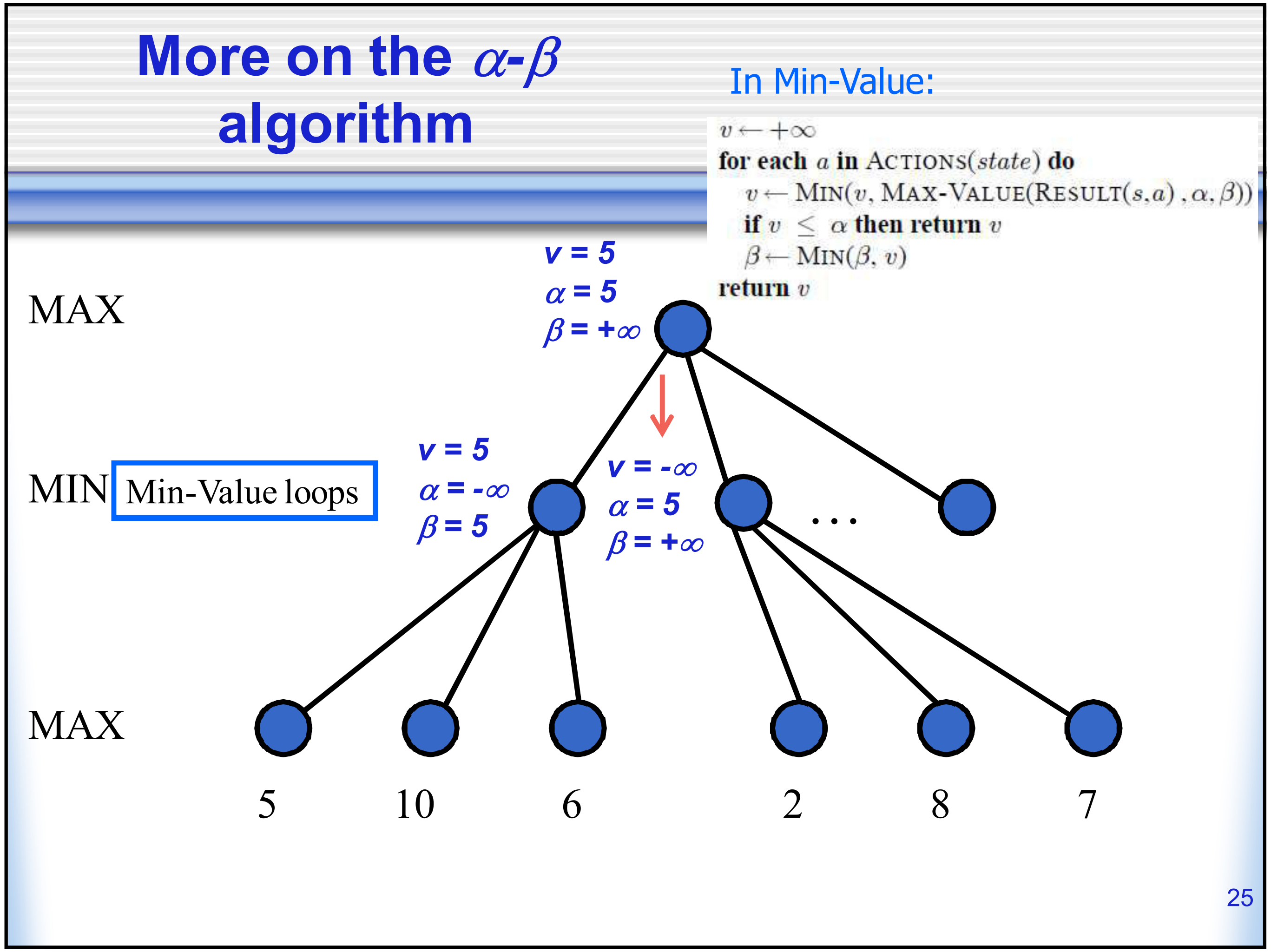
Max-Value loops



# More on the $\alpha$-$\beta$ algorithm

In Max-Value:

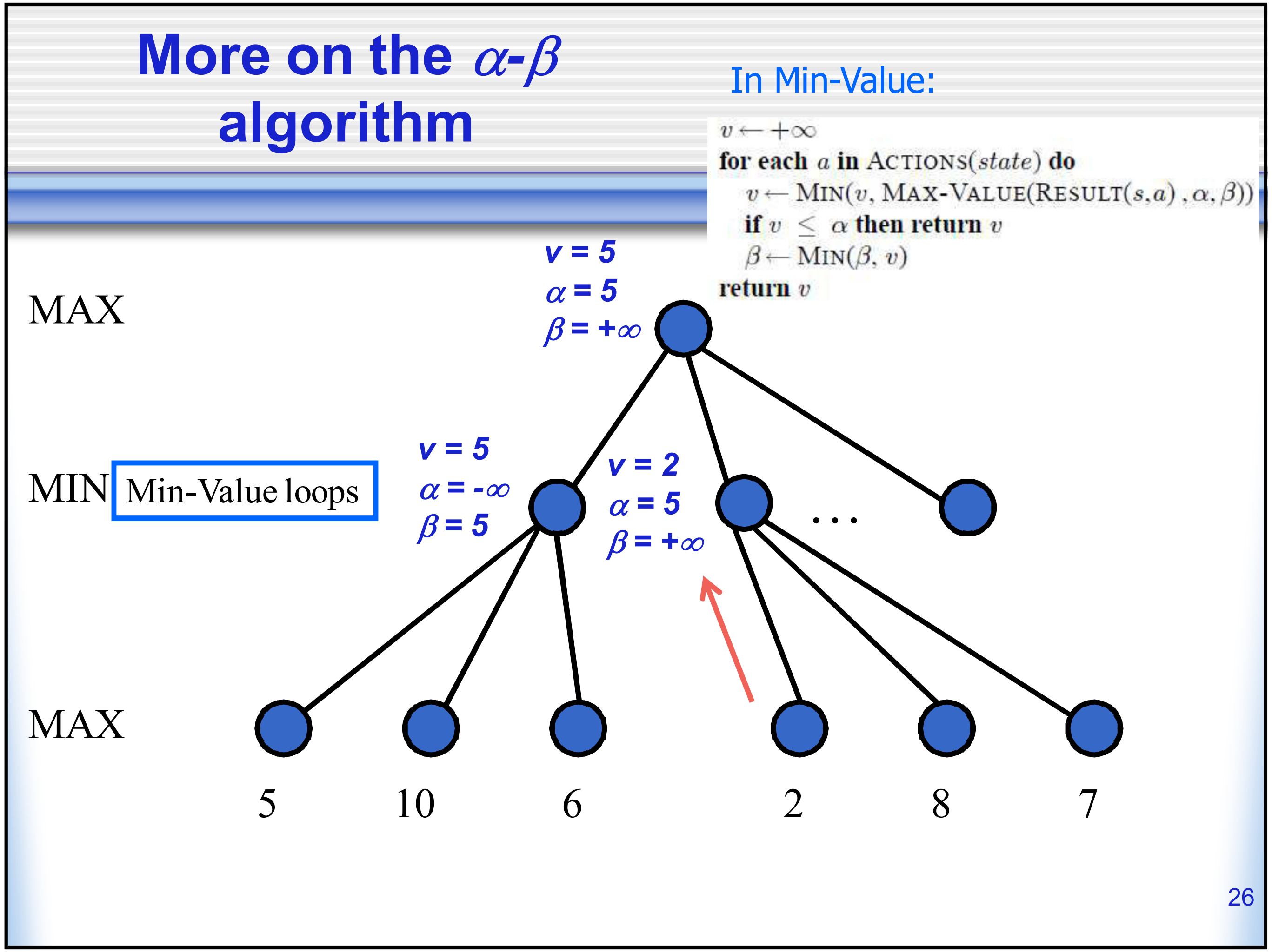
```
$v \leftarrow -\infty$
for each $a$ in ACTIONS($state$) do
    $v \leftarrow$ MAX($v$, MIN-VALUE(RESULT($s,a$), $\alpha$, $\beta$))
    if $v \geq \beta$ then return $v$
    $\alpha \leftarrow$ MAX($\alpha$, $v$)
return $v$
```



## More on the $\alpha$-$\beta$ algorithm

In Min-Value:

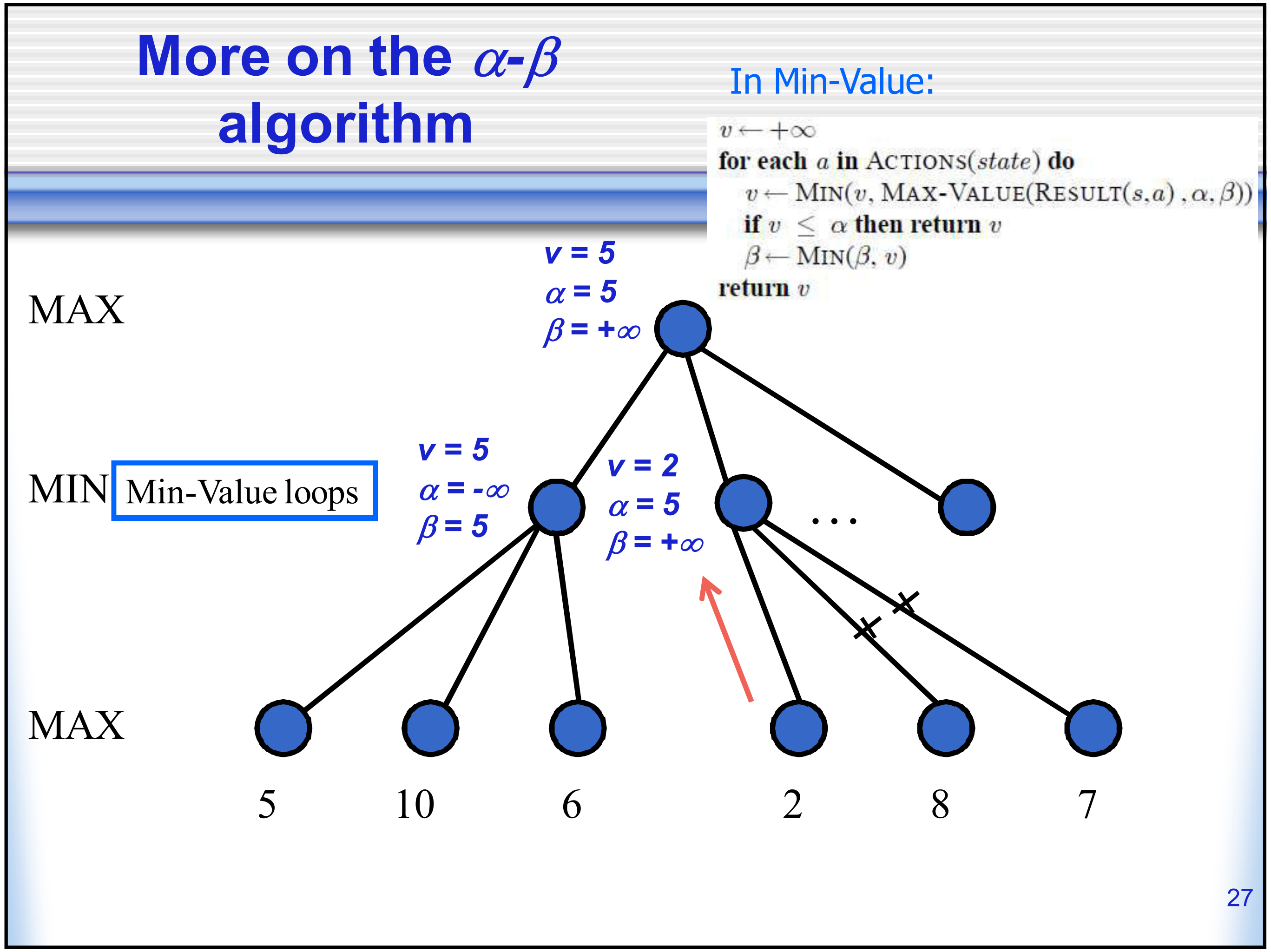
```
$v \leftarrow +\infty$
for each $a$ in ACTIONS($state$) do
    $v \leftarrow$ MIN($v$, MAX-VALUE(RESULT($s$,$a$) , $\alpha$, $\beta$))
    if $v \leq \alpha$ then return $v$
    $\beta \leftarrow$ MIN($\beta$, $v$)
return $v$
```



# More on the $\alpha$-$\beta$ algorithm

In Min-Value:

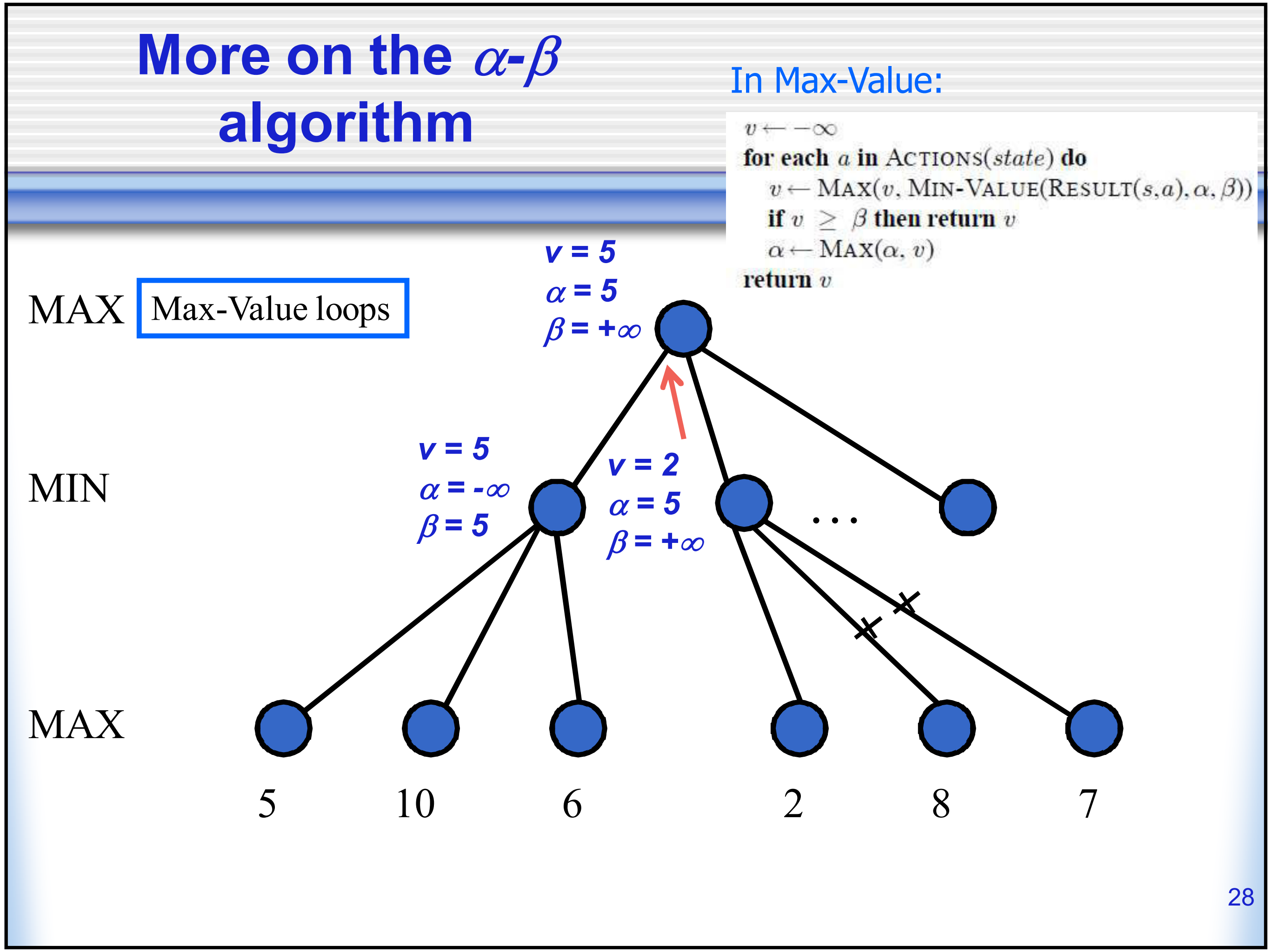
```
$v \leftarrow +\infty$
for each $a$ in ACTIONS($state$) do
    $v \leftarrow$ MIN($v$, MAX-VALUE(RESULT($s,a$), $\alpha$, $\beta$))
    if $v \leq \alpha$ then return $v$
    $\beta \leftarrow$ MIN($\beta$, $v$)
return $v$
```



# More on the $\alpha$-$\beta$ algorithm

In Min-Value:

```
$v \leftarrow +\infty$
for each $a$ in ACTIONS($state$) do
    $v \leftarrow$ MIN($v$, MAX-VALUE(RESULT($s,a$) , $\alpha$, $\beta$))
    if $v \leq \alpha$ then return $v$
    $\beta \leftarrow$ MIN($\beta$, $v$)
return $v$
```

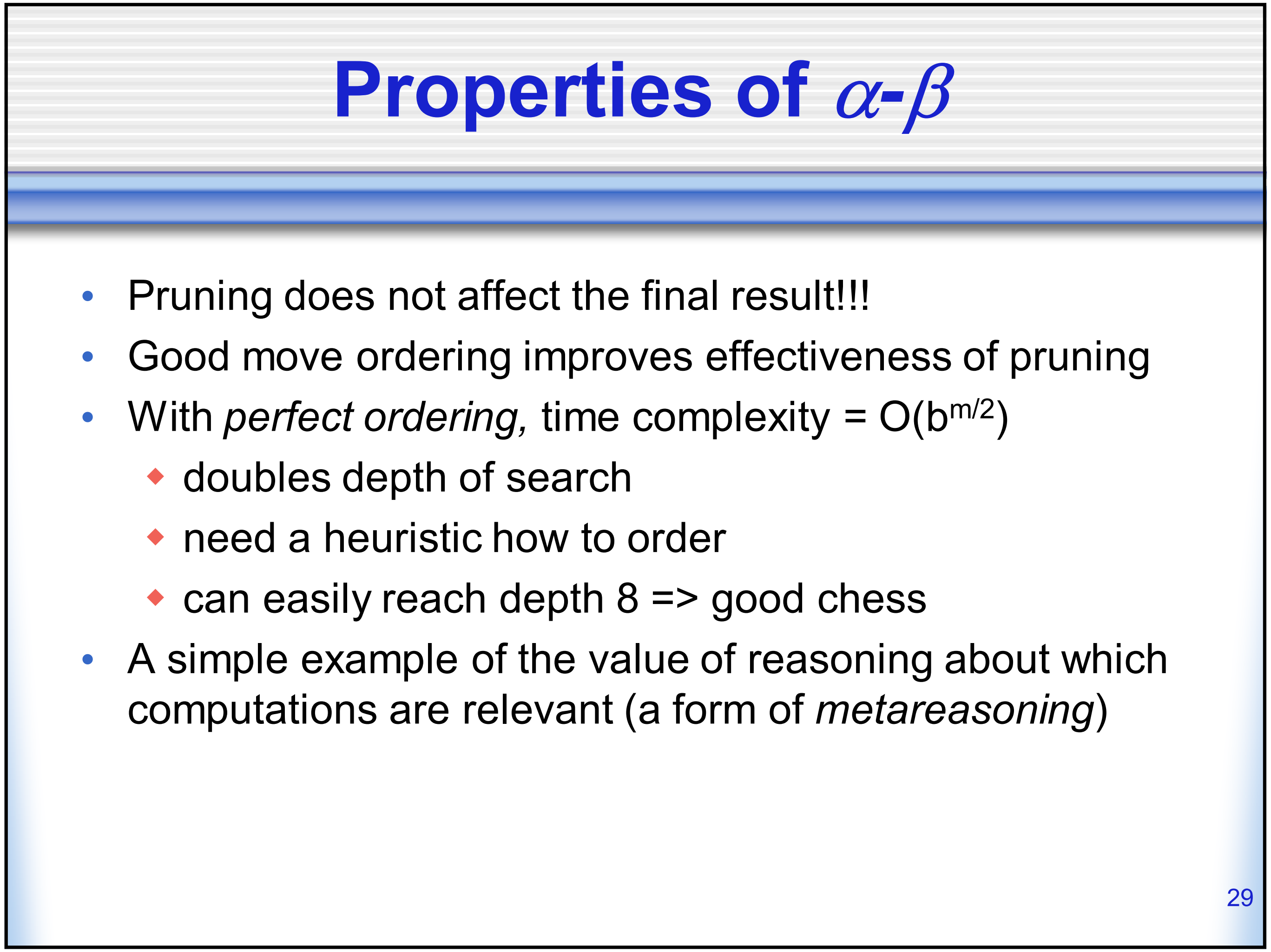


# More on the $\alpha$-$\beta$ algorithm

In Max-Value:

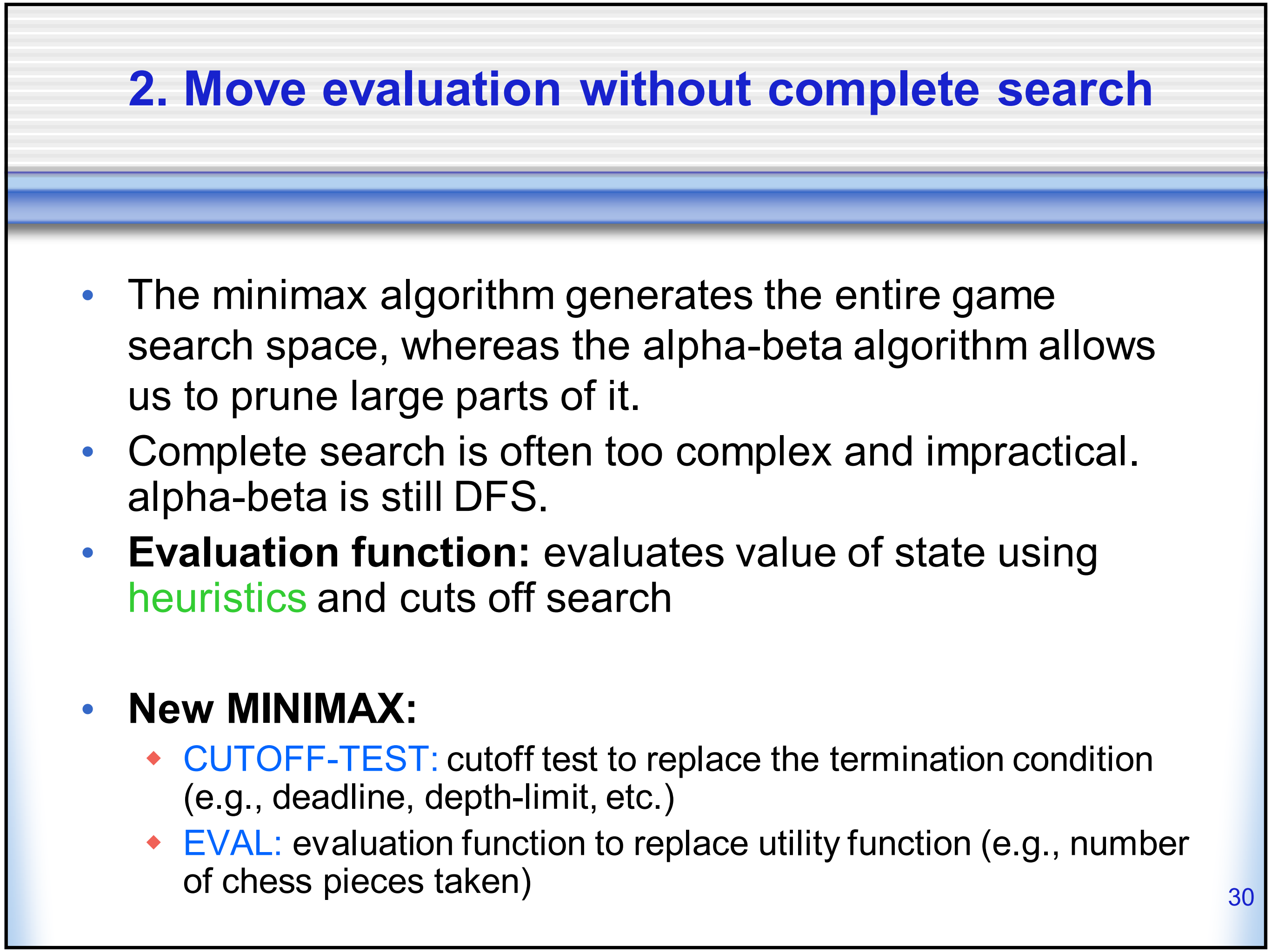
```
$v \leftarrow -\infty$
for each $a$ in ACTIONS($state$) do
    $v \leftarrow$ MAX($v$, MIN-VALUE(RESULT($s$,$a$), $\alpha$, $\beta$))
    if $v \geq \beta$ then return $v$
    $\alpha \leftarrow$ MAX($\alpha$, $v$)
return $v$
```

Max-Value loops



# Properties of $\alpha$-$\beta$

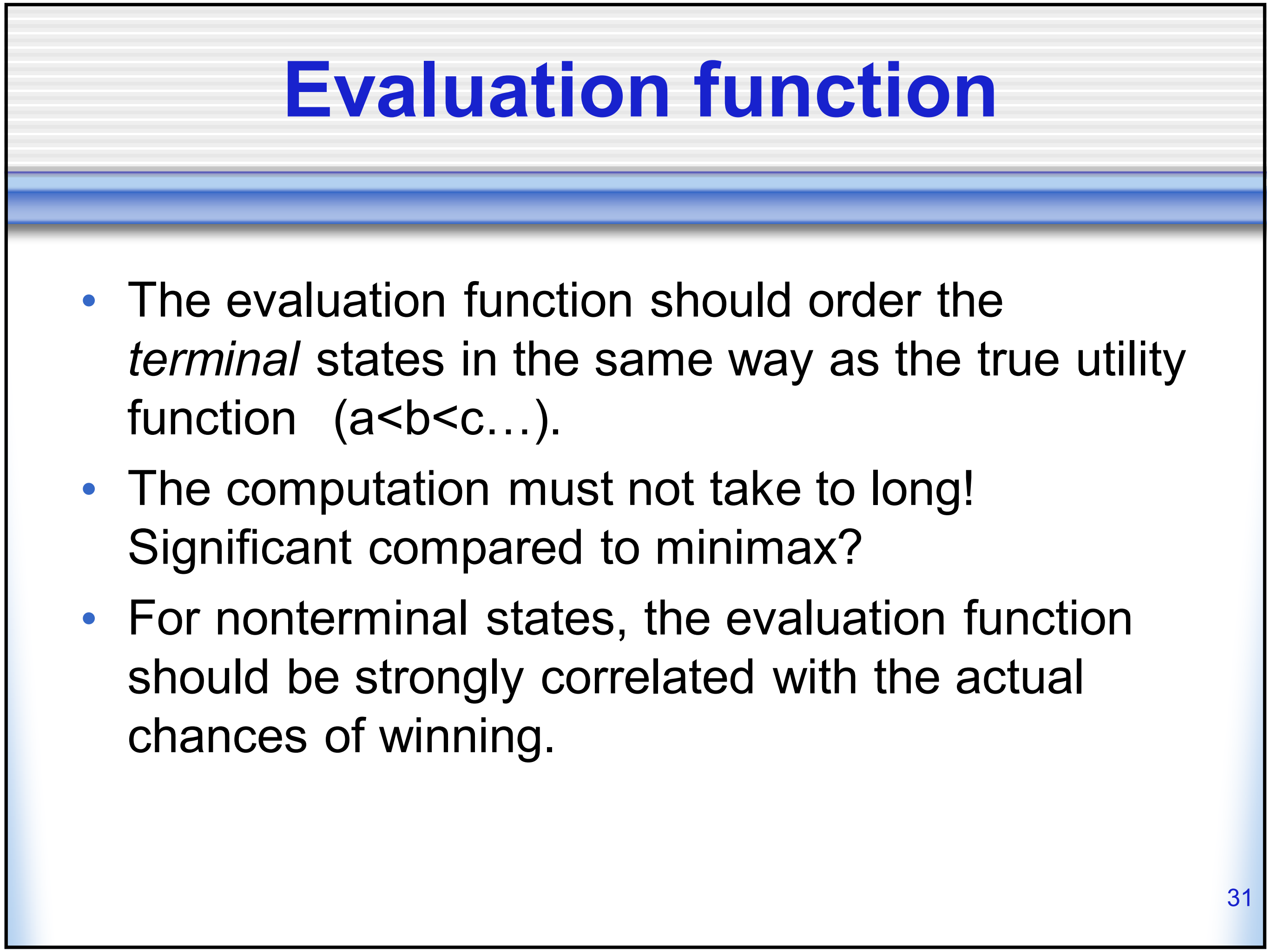
- Pruning does not affect the final result!!!
- Good move ordering improves effectiveness of pruning
- With *perfect ordering,* time complexity = O($b^{m/2}$)
  - doubles depth of search
  - need a heuristic how to order
  - can easily reach depth 8 => good chess
- A simple example of the value of reasoning about which computations are relevant (a form of *metareasoning*)



## 2. Move evaluation without complete search

- The minimax algorithm generates the entire game search space, whereas the alpha-beta algorithm allows us to prune large parts of it.
- Complete search is often too complex and impractical. alpha-beta is still DFS.
- **Evaluation function:** evaluates value of state using heuristics and cuts off search

- **New MINIMAX:**
    - CUTOFF-TEST: cutoff test to replace the termination condition (e.g., deadline, depth-limit, etc.)
    - EVAL: evaluation function to replace utility function (e.g., number of chess pieces taken)



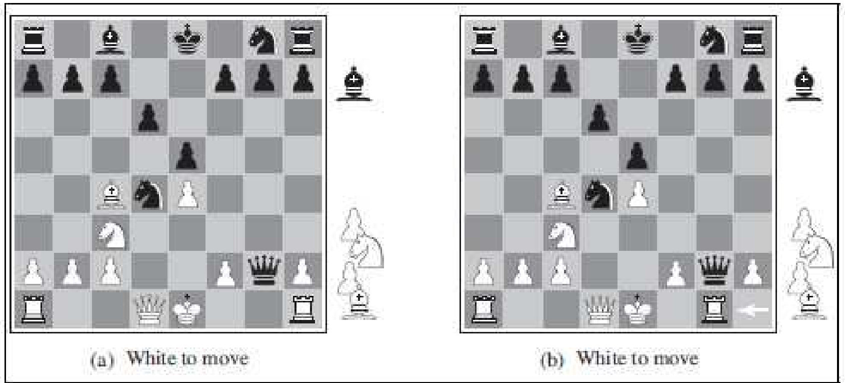
# Evaluation function

- The evaluation function should order the *terminal* states in the same way as the true utility function (a<b<c…).
- The computation must not take to long! Significant compared to minimax?
- For nonterminal states, the evaluation function should be strongly correlated with the actual chances of winning.

# Evaluation functions

- Most calculate features – e.g., number of pawns
- From that we can form categories, equivalence classes.
- Any category represent states that win, lose or result in draws.
- If we know 72% lead to win (+1), 20% to loss (0), 8% drawn (1/2).
  Expected value:
- (0,72* +1) + (0,20* 0) + (0,08 * 1/2)= 0,76

# Evaluation functions



(a) White to move

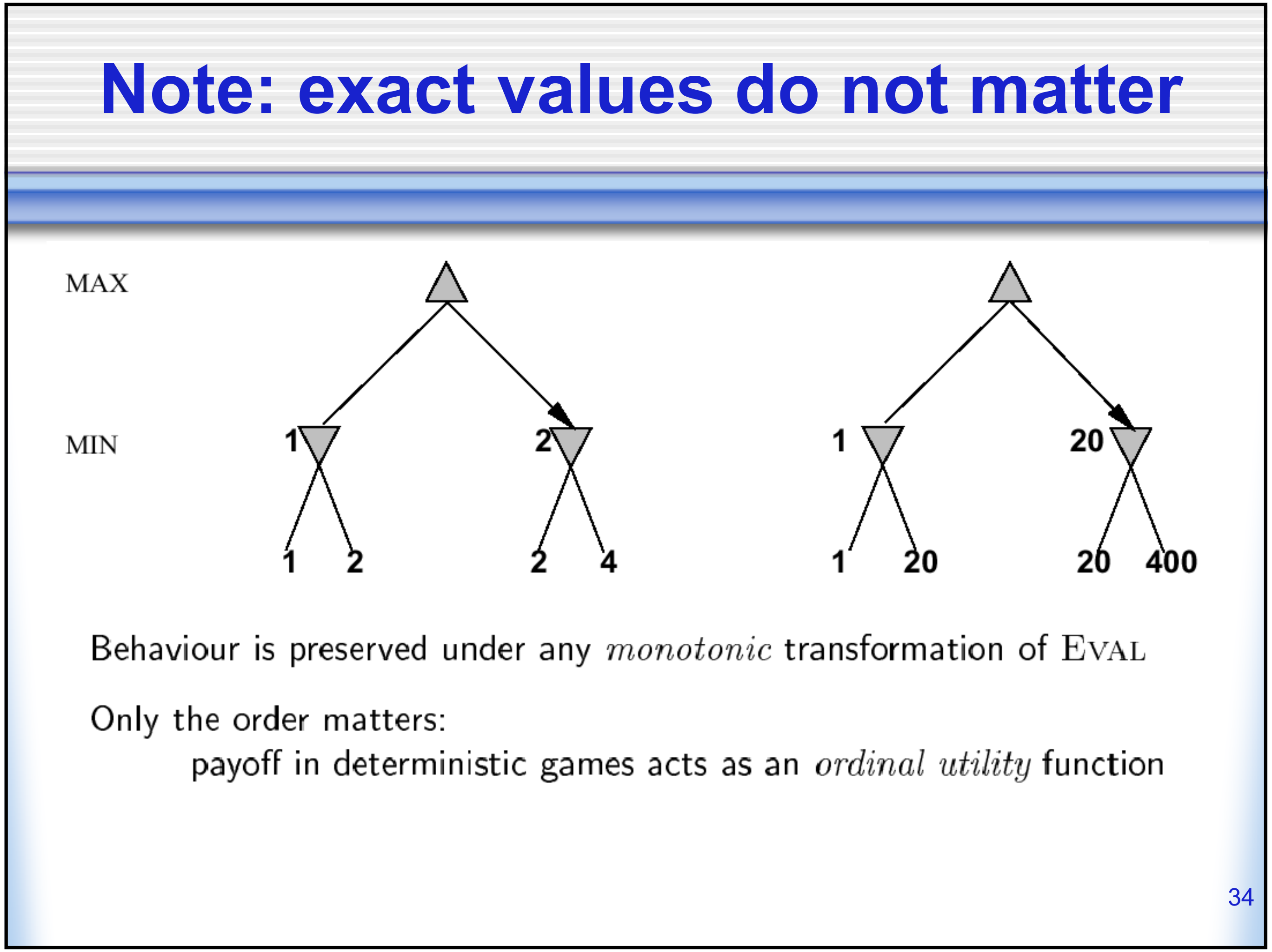
(b) White to move

Figure 5.8 FILES: figures/chess-evaluation3.eps (Tue Nov 3 16:22:33 2009). Two chess positions that differ only in the position of the rook at lower right. In (a), Black has an advantage of a knight and two pawns, which should be enough to win the game. In (b), White will capture the queen, giving it an advantage that should be strong enough to win.

- **Weighted linear evaluation function:** to combine *n* heuristics

$$f = w_1 f_1 + w_2 f_2 + \ldots + w_n f_n$$

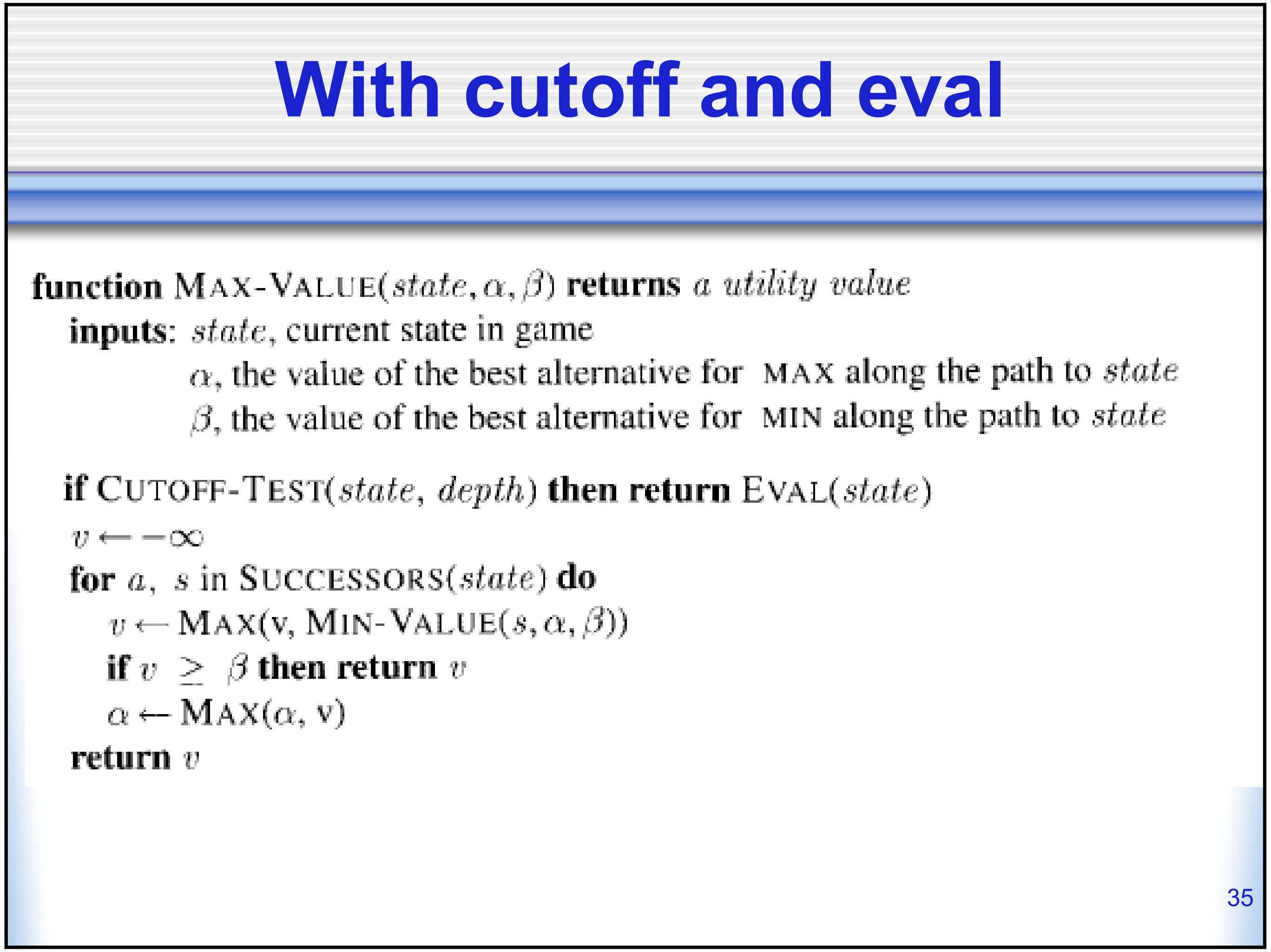
E.g, *w's* could be the values of pieces (1 for pawn, 3 for bishop etc.)
*f's* could be the number of type of pieces on the board



# Note: exact values do not matter

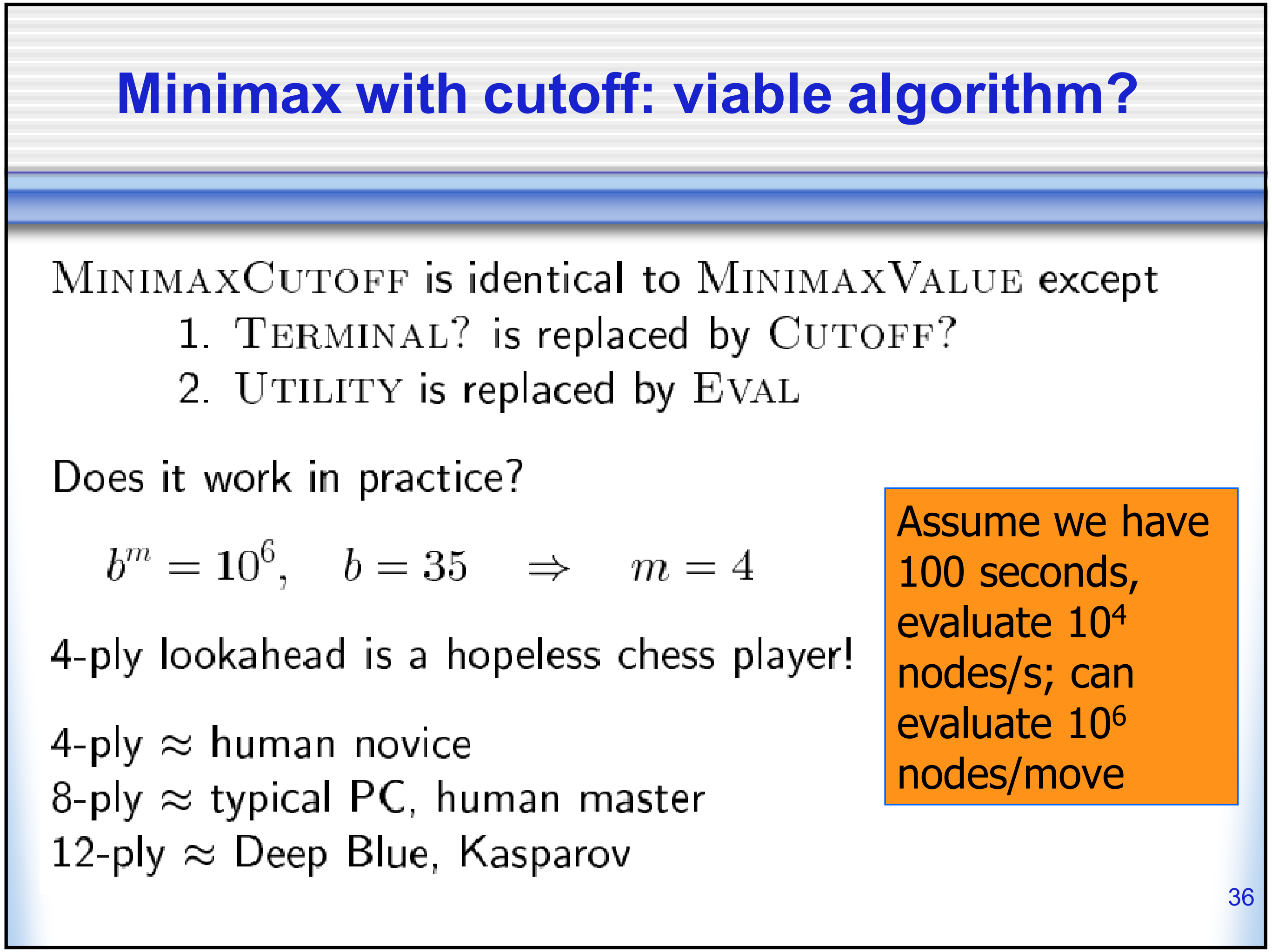
Behaviour is preserved under any *monotonic* transformation of EVAL

Only the order matters:
payoff in deterministic games acts as an *ordinal utility* function



# With cutoff and eval

```
function MAX-VALUE($state, \alpha, \beta$) returns $a$ utility value
  inputs: $state$, current state in game
          $\alpha$, the value of the best alternative for MAX along the path to $state$
          $\beta$, the value of the best alternative for MIN along the path to $state$

  if CUTOFF-TEST($state$, $depth$) then return EVAL($state$)
  $v \leftarrow -\infty$
  for $a$, $s$ in SUCCESSORS($state$) do
    $v \leftarrow$ MAX(v, MIN-VALUE($s, \alpha, \beta$))
    if $v \geq \beta$ then return $v$
    $\alpha \leftarrow$ MAX($\alpha$, v)
  return $v$
```



## Minimax with cutoff: viable algorithm?

MINIMAXCUTOFF is identical to MINIMAXVALUE except

1. TERMINAL? is replaced by CUTOFF?
2. UTILITY is replaced by EVAL

Does it work in practice?

$$b^m = 10^6, \quad b = 35 \quad \Rightarrow \quad m = 4$$

4-ply lookahead is a hopeless chess player!

4-ply ≈ human novice
8-ply ≈ typical PC, human master
12-ply ≈ Deep Blue, Kasparov

Assume we have 100 seconds, evaluate $10^4$ nodes/s; can evaluate $10^6$ nodes/move

# Other Cutoff methods

- Quiescent search
  apply eval only to positions that are quiescent, have no big change of value in the near future.
- Forward pruning
  considers not all moves in a concrete position.
  Beam search is one approach to forward pruning.
- ProbCut
  probabilistic alpha-beta with statistical prior knowledge

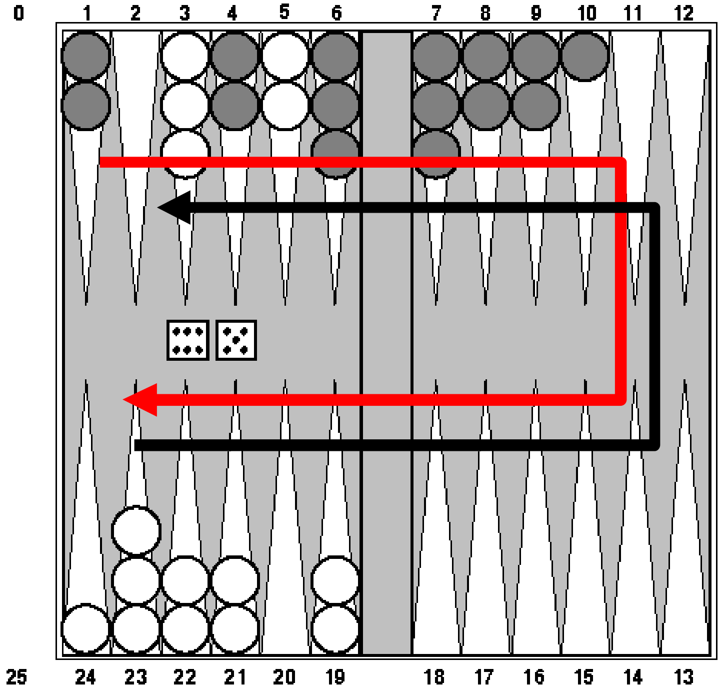
# Games of chance

- Backgammon is a two-player game with **uncertainty**.

- Players roll dices to determine what moves to make.

- White/red arrow has just rolled 5 and 6 and has four legal moves:
  - 5-10, 5-11
  - 5-11, 19-24
  - 5-10, 10-16
  - 5-11, 11-16

- Such games are good for exploring decision making in adversarial problems involving skill and luck.



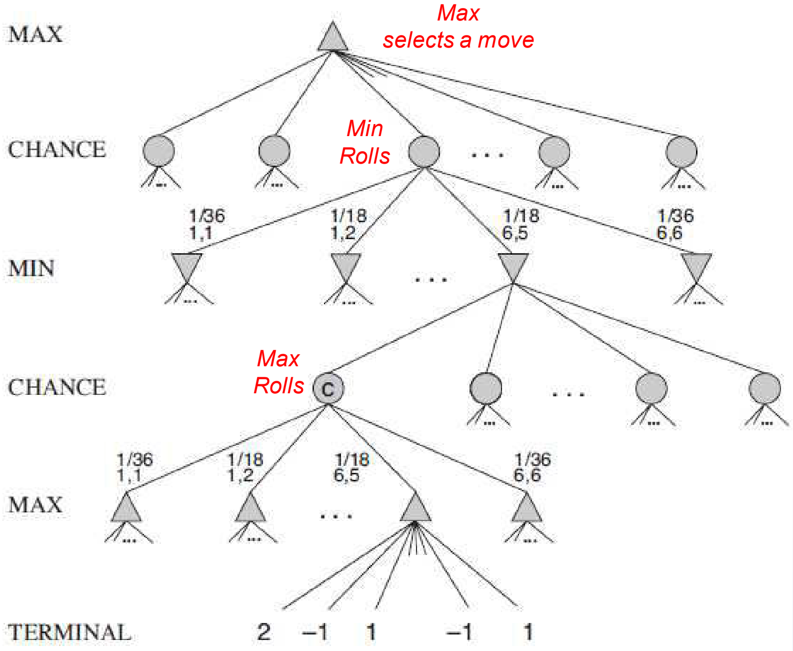
# Game Trees with Chance Nodes

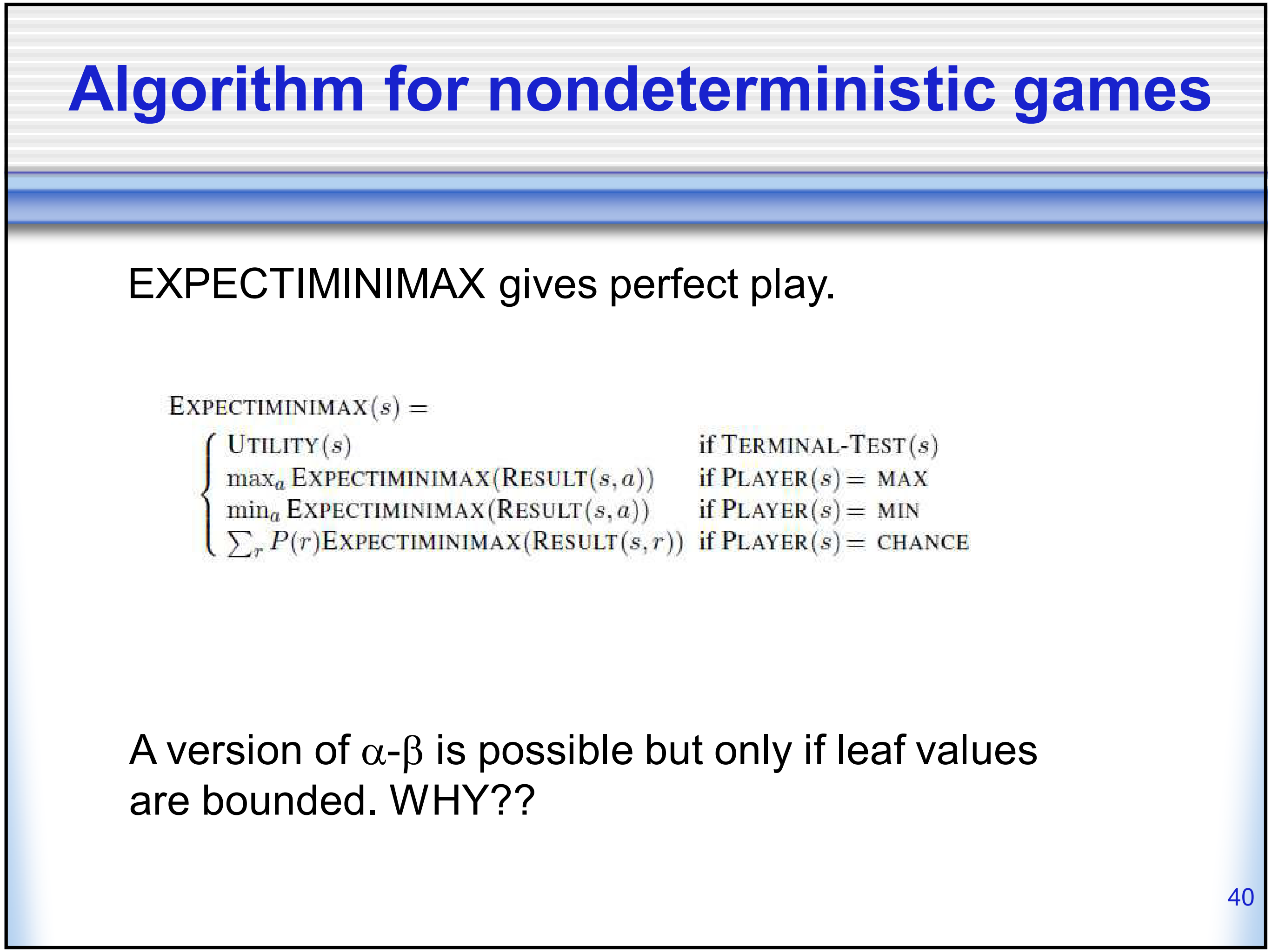
- Use minimax to compute values for MAX and MIN nodes
- Use **expected values** for chance nodes
- For chance nodes over a max node, as in C:

expectimax(C) = $\mathrm{Sum}_i(P(d_i)$ * maxvalue(i))

- For chance nodes over a min node:

expectimin(N) = $\mathrm{Sum}_i(P(d_i)$ * minvalue(i))





# Algorithm for nondeterministic games

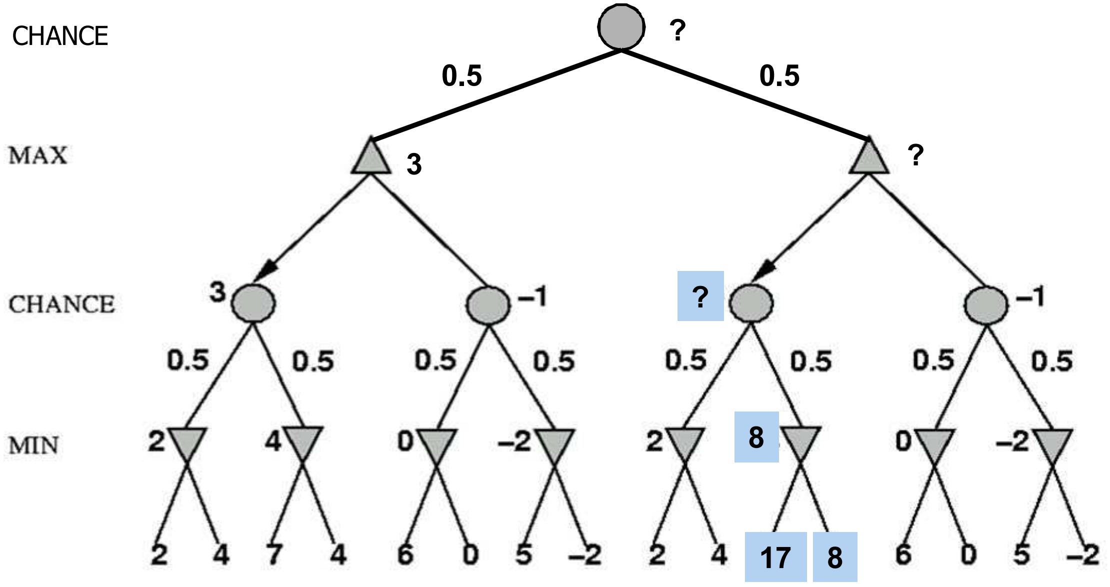
EXPECTIMINIMAX gives perfect play.

$$\textsc{Expectiminimax}(s) =$$
$$\begin{cases} \textsc{Utility}(s) & \text{if } \textsc{Terminal-Test}(s) \\ \max_a \textsc{Expectiminimax}(\textsc{Result}(s,a)) & \text{if } \textsc{Player}(s) = \textsc{max} \\ \min_a \textsc{Expectiminimax}(\textsc{Result}(s,a)) & \text{if } \textsc{Player}(s) = \textsc{min} \\ \sum_r P(r)\textsc{Expectiminimax}(\textsc{Result}(s,r)) & \text{if } \textsc{Player}(s) = \textsc{chance} \end{cases}$$

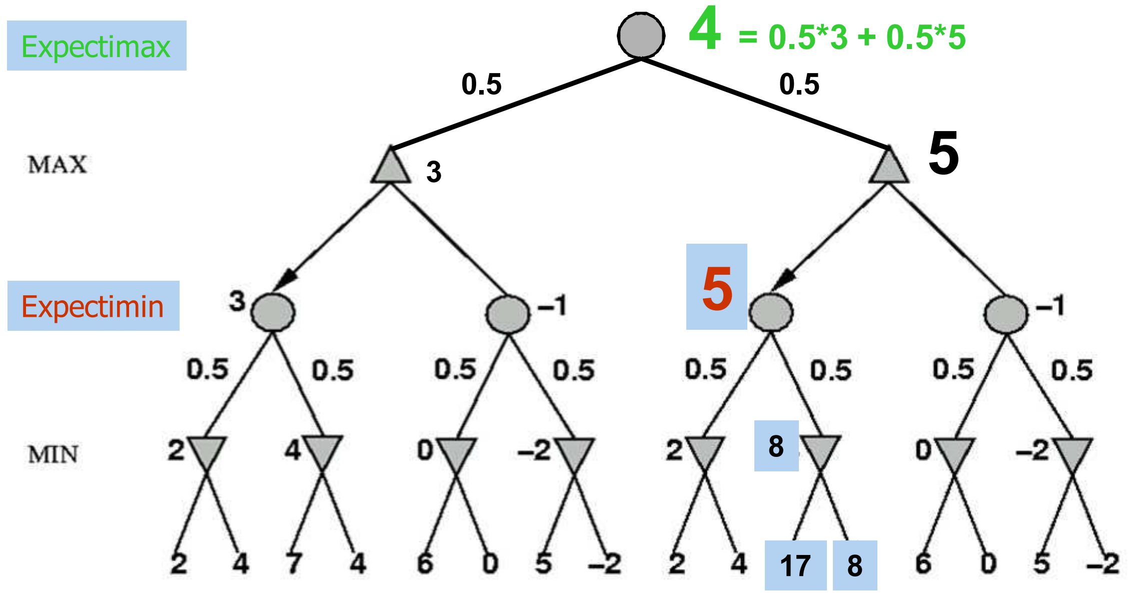
A version of $\alpha$-$\beta$ is possible but only if leaf values are bounded. WHY??

# Nondeterministic games: the element of chance

**expectimax** and **expectimin**, expected values over all possible outcomes

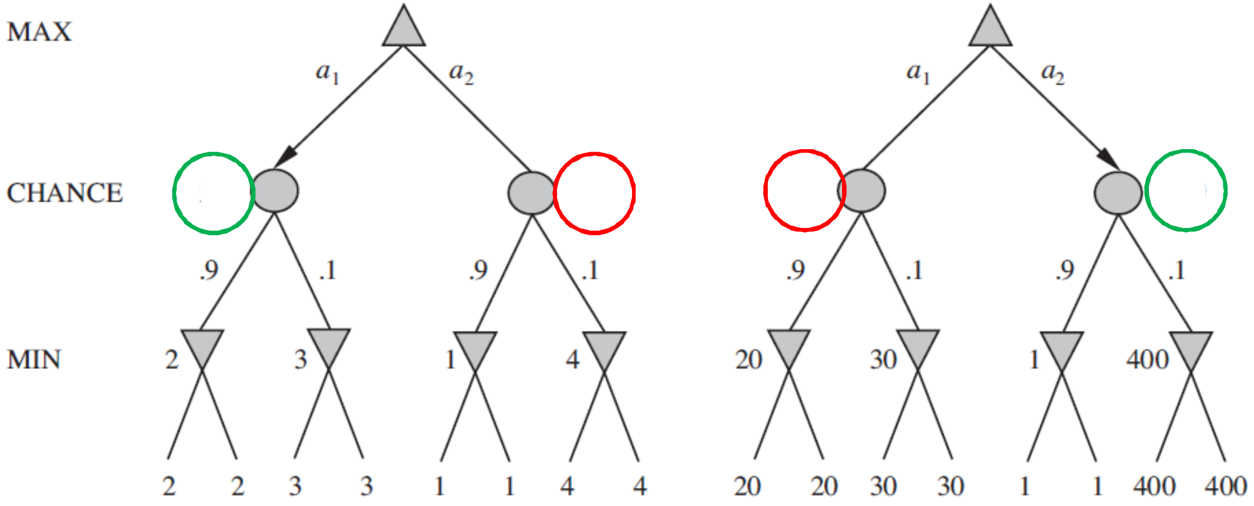


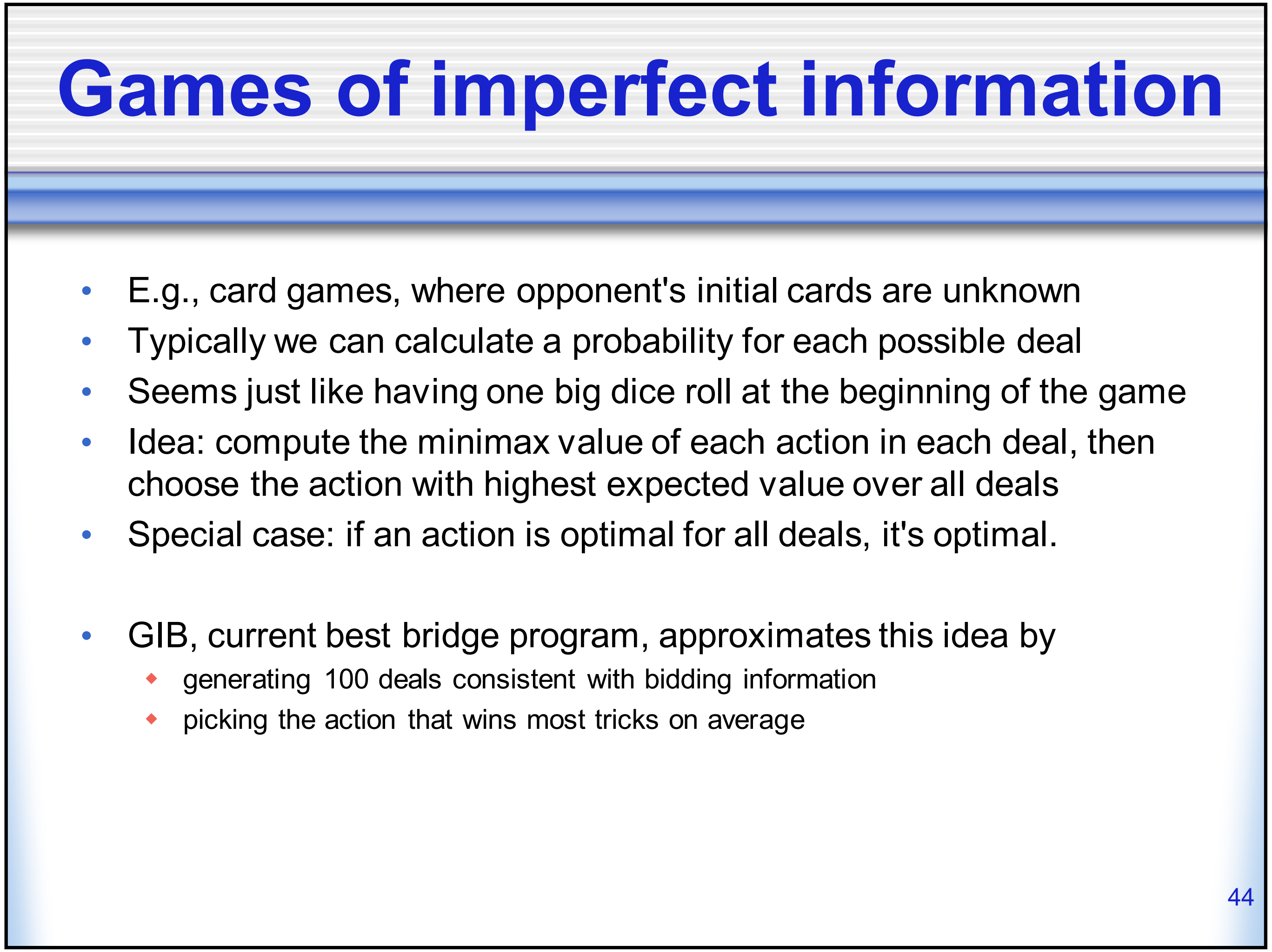
# Nondeterministic games: the element of chance



# Evaluation functions

**Order-preserving transformation do not necessarily behave the same!**

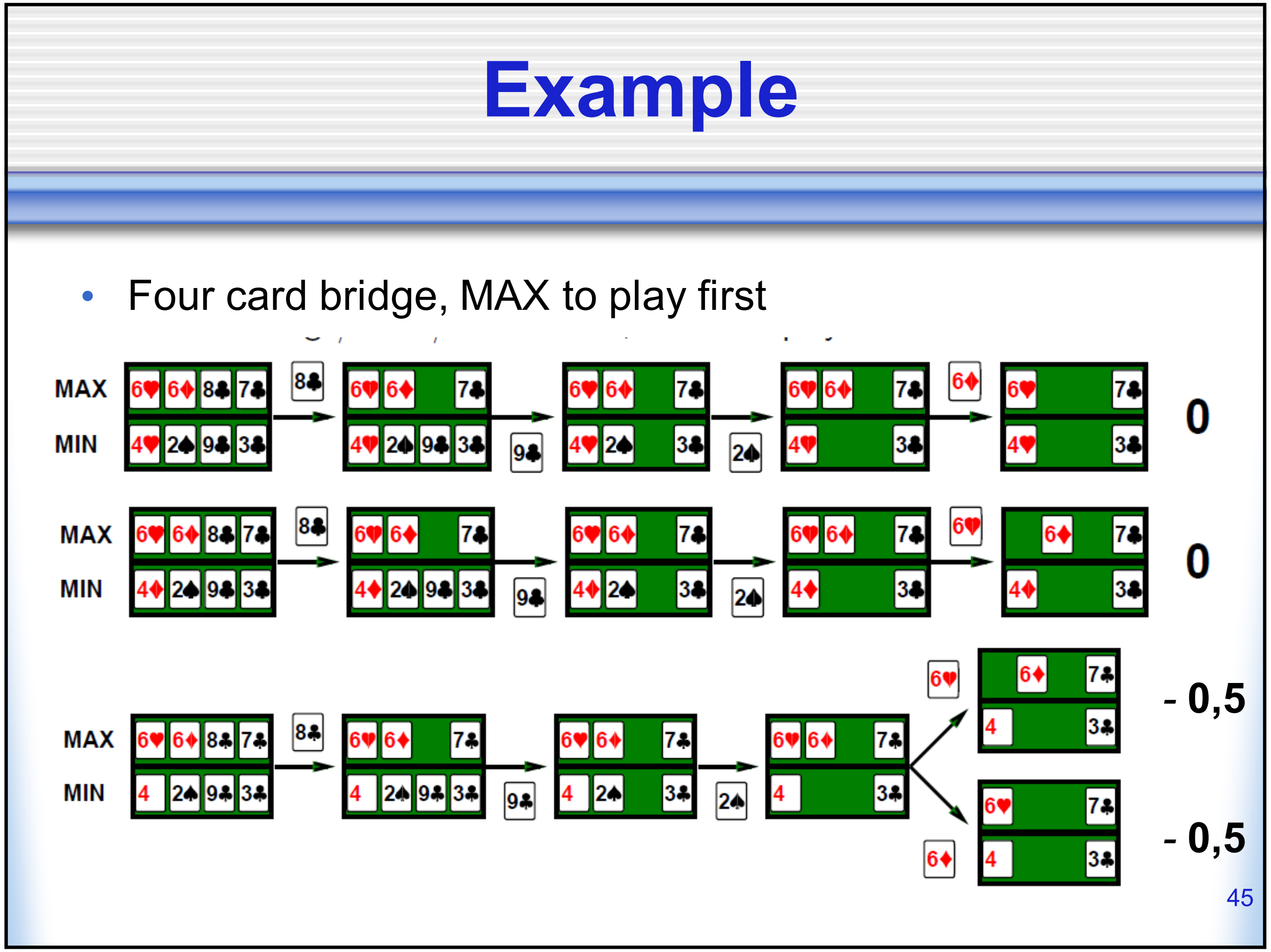




# Games of imperfect information

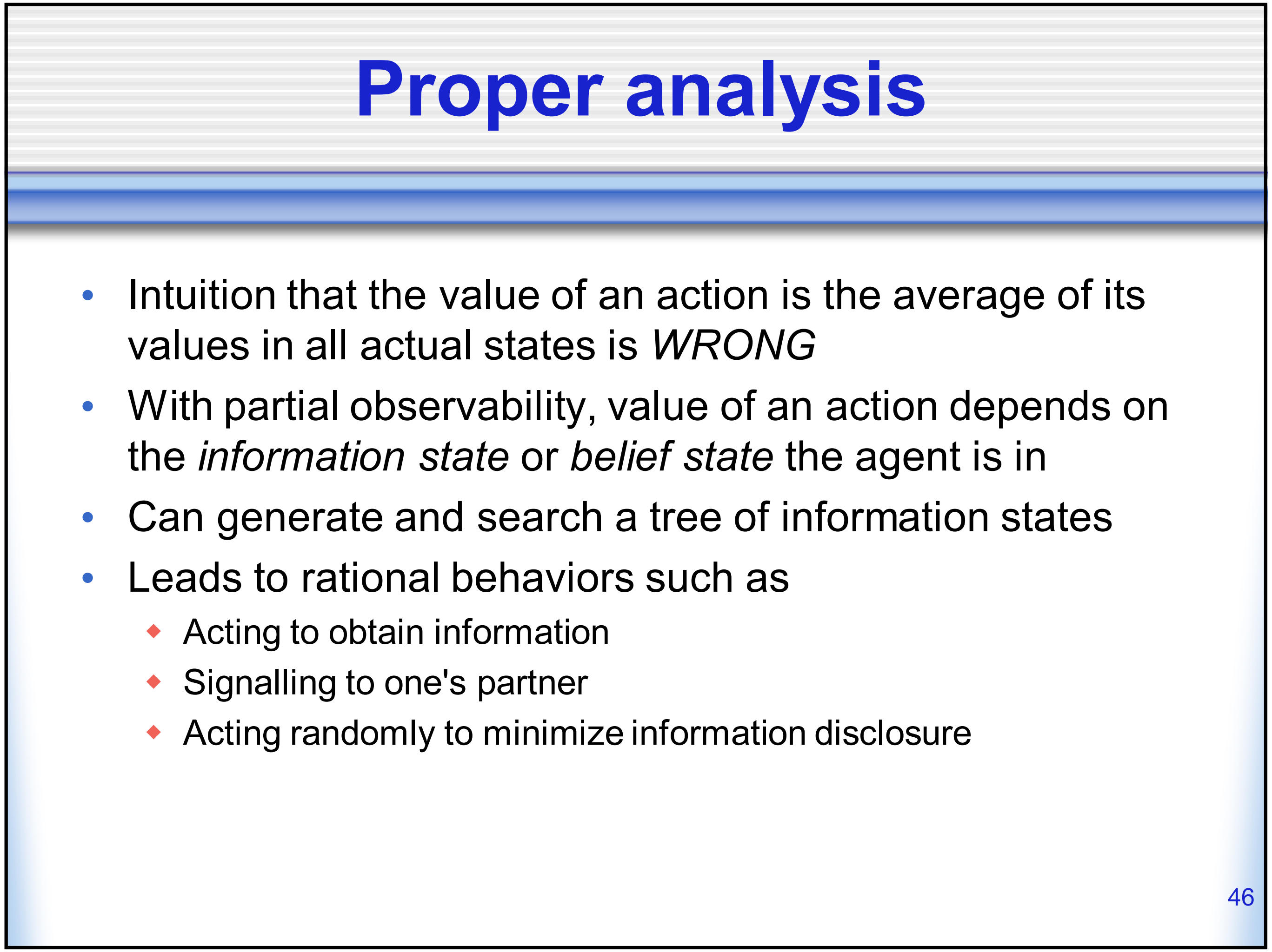
- E.g., card games, where opponent's initial cards are unknown
- Typically we can calculate a probability for each possible deal
- Seems just like having one big dice roll at the beginning of the game
- Idea: compute the minimax value of each action in each deal, then choose the action with highest expected value over all deals
- Special case: if an action is optimal for all deals, it's optimal.

- GIB, current best bridge program, approximates this idea by
  - generating 100 deals consistent with bidding information
  - picking the action that wins most tricks on average



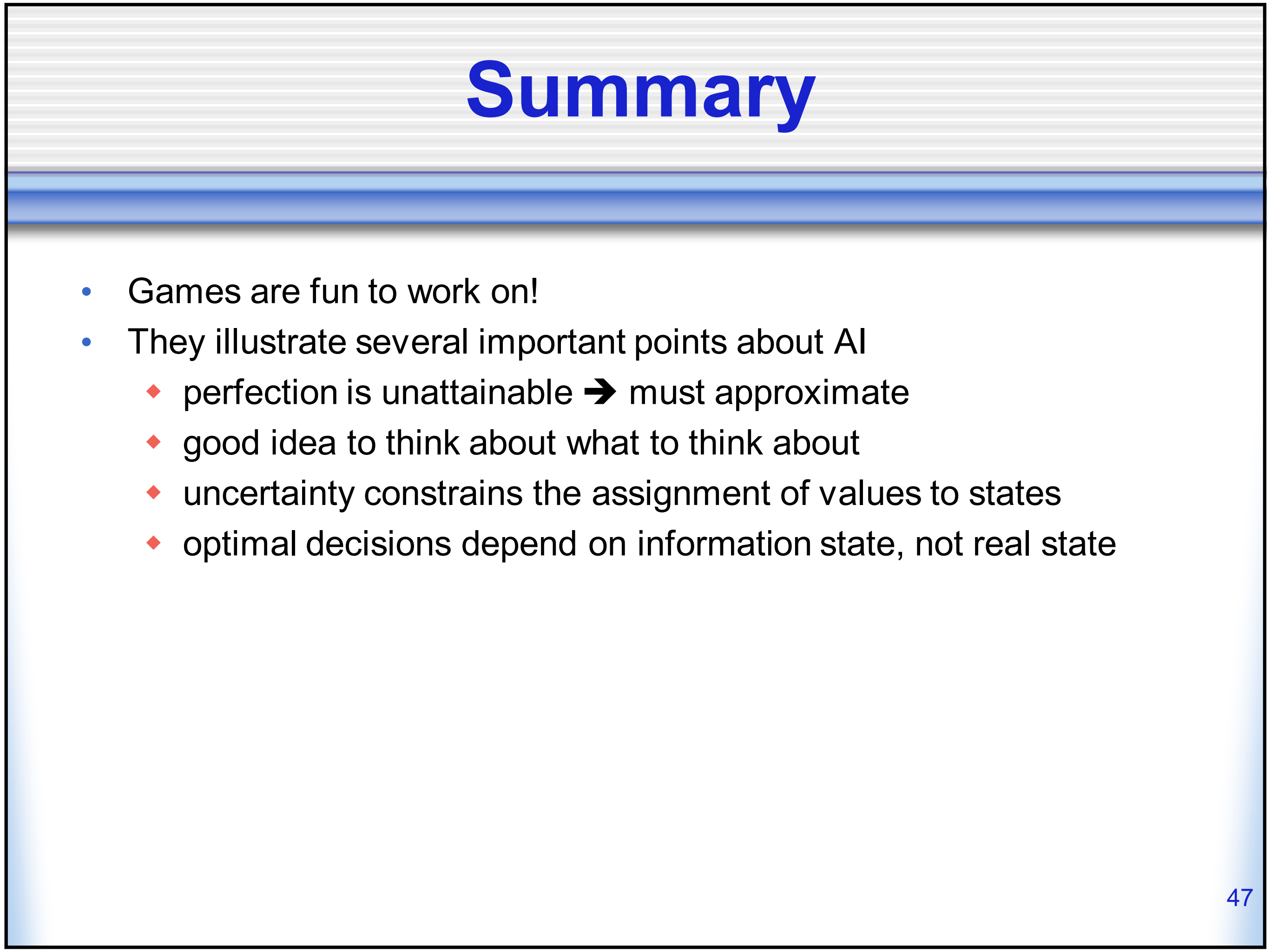
# Example

- Four card bridge, MAX to play first



# Proper analysis

- Intuition that the value of an action is the average of its values in all actual states is *WRONG*
- With partial observability, value of an action depends on the *information state* or *belief state* the agent is in
- Can generate and search a tree of information states
- Leads to rational behaviors such as
  - Acting to obtain information
  - Signalling to one's partner
  - Acting randomly to minimize information disclosure

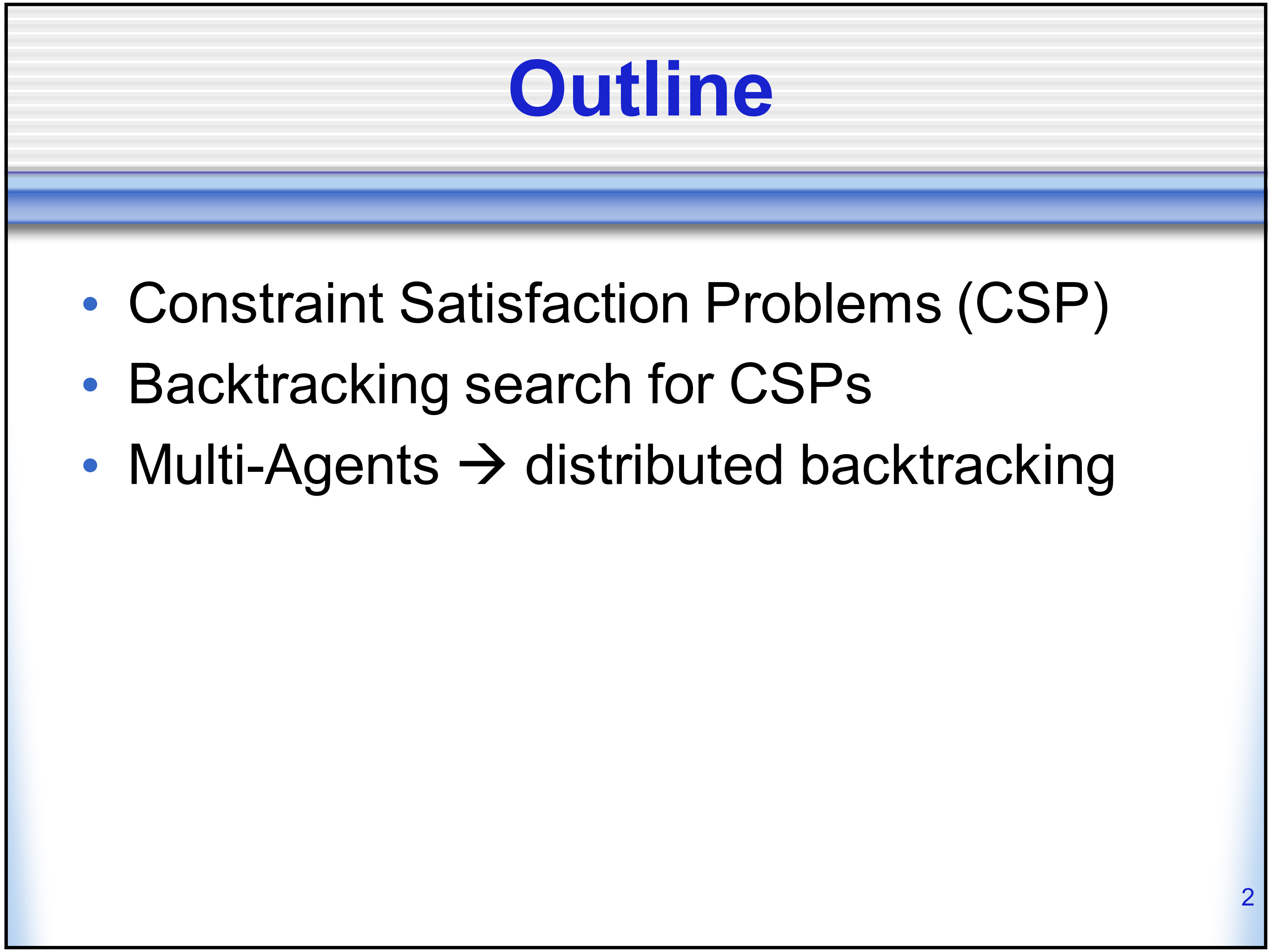


# Summary

- Games are fun to work on!
- They illustrate several important points about AI
  - perfection is unattainable ➔ must approximate
  - good idea to think about what to think about
  - uncertainty constrains the assignment of values to states
  - optimal decisions depend on information state, not real state

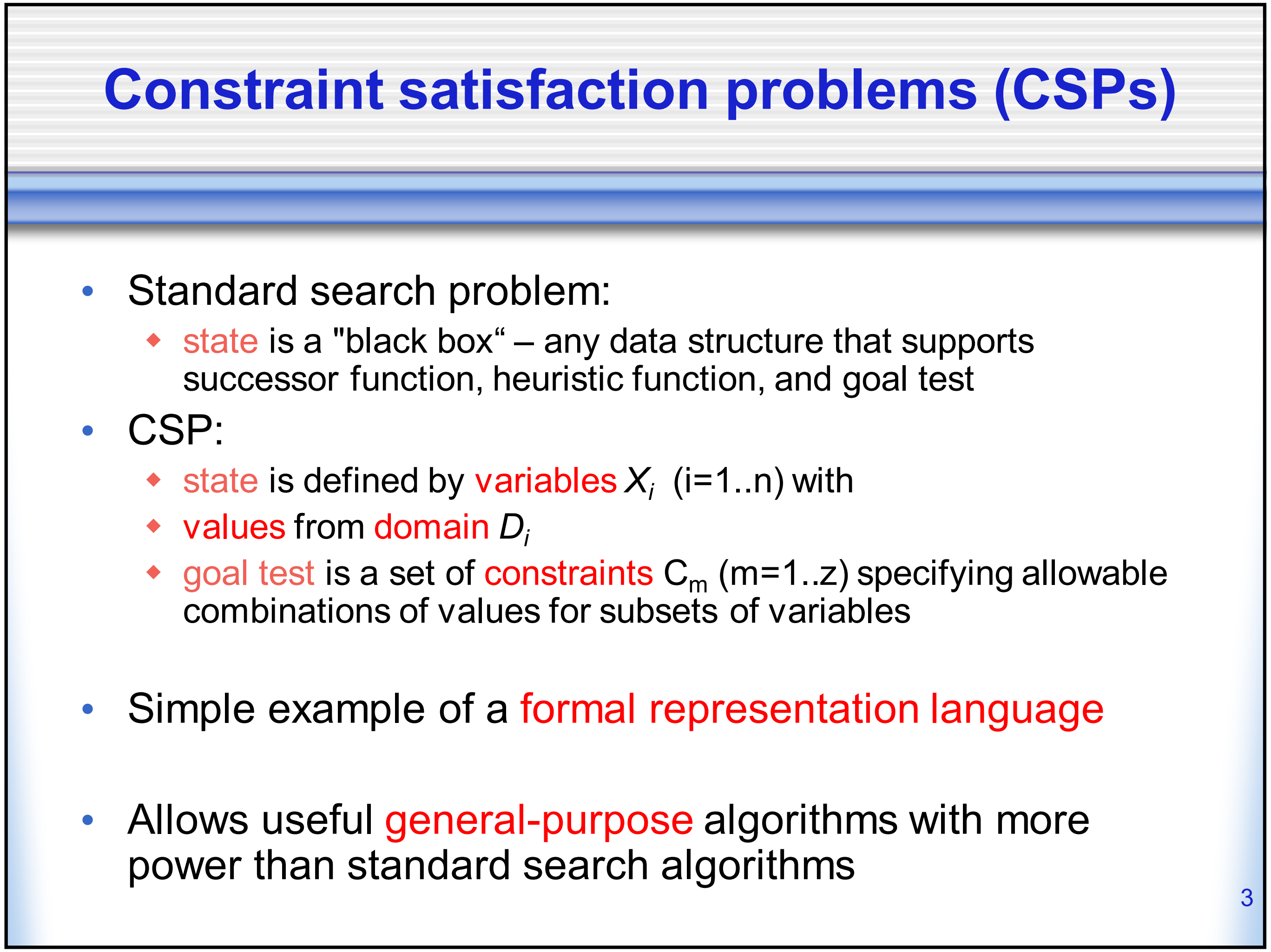
# Intelligent Autonomous Agents and Cognitive Robotics Topic 3: Constraint Satisfaction Problems

Slides partly from Hwee Tou Ng's
Chapter 5 of AIMA



# Outline

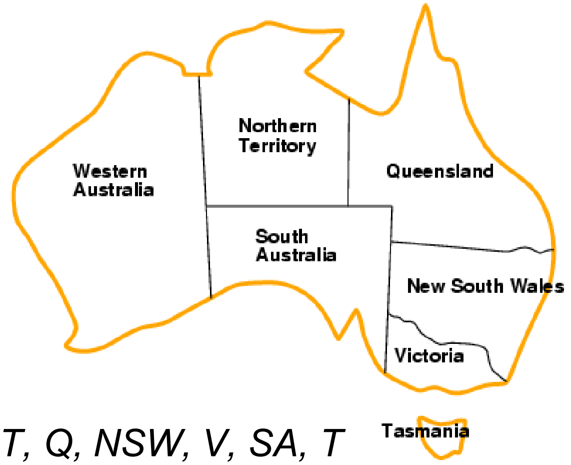
- Constraint Satisfaction Problems (CSP)
- Backtracking search for CSPs
- Multi-Agents → distributed backtracking



## Constraint satisfaction problems (CSPs)

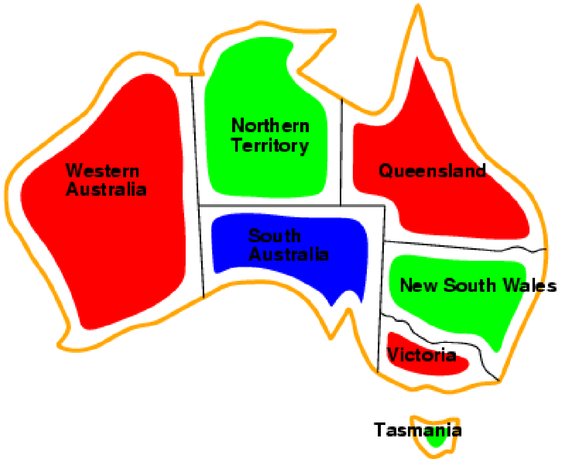
- Standard search problem:
  - state is a "black box“ – any data structure that supports successor function, heuristic function, and goal test
- CSP:
  - state is defined by variables $X_i$ (i=1..n) with
  - values from domain $D_i$
  - goal test is a set of constraints $C_m$ (m=1..z) specifying allowable combinations of values for subsets of variables
- Simple example of a formal representation language
- Allows useful general-purpose algorithms with more power than standard search algorithms

# Visual example: Map-Coloring



- Variables: *WA, NT, Q, NSW, V, SA, T*
- Domains: $\forall i, D_i$ = {red, green, blue}
- Constraints: adjacent regions must have different colors
  - e.g., WA ≠ NT
  - or (WA,NT) ∈ {(red,green),(red,blue),(green,red), (green,blue), (blue,red), (blue,green)}

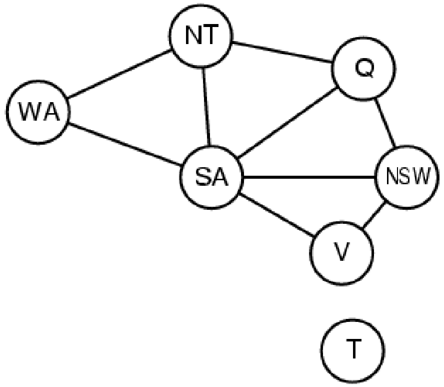
# Example: Map-Coloring



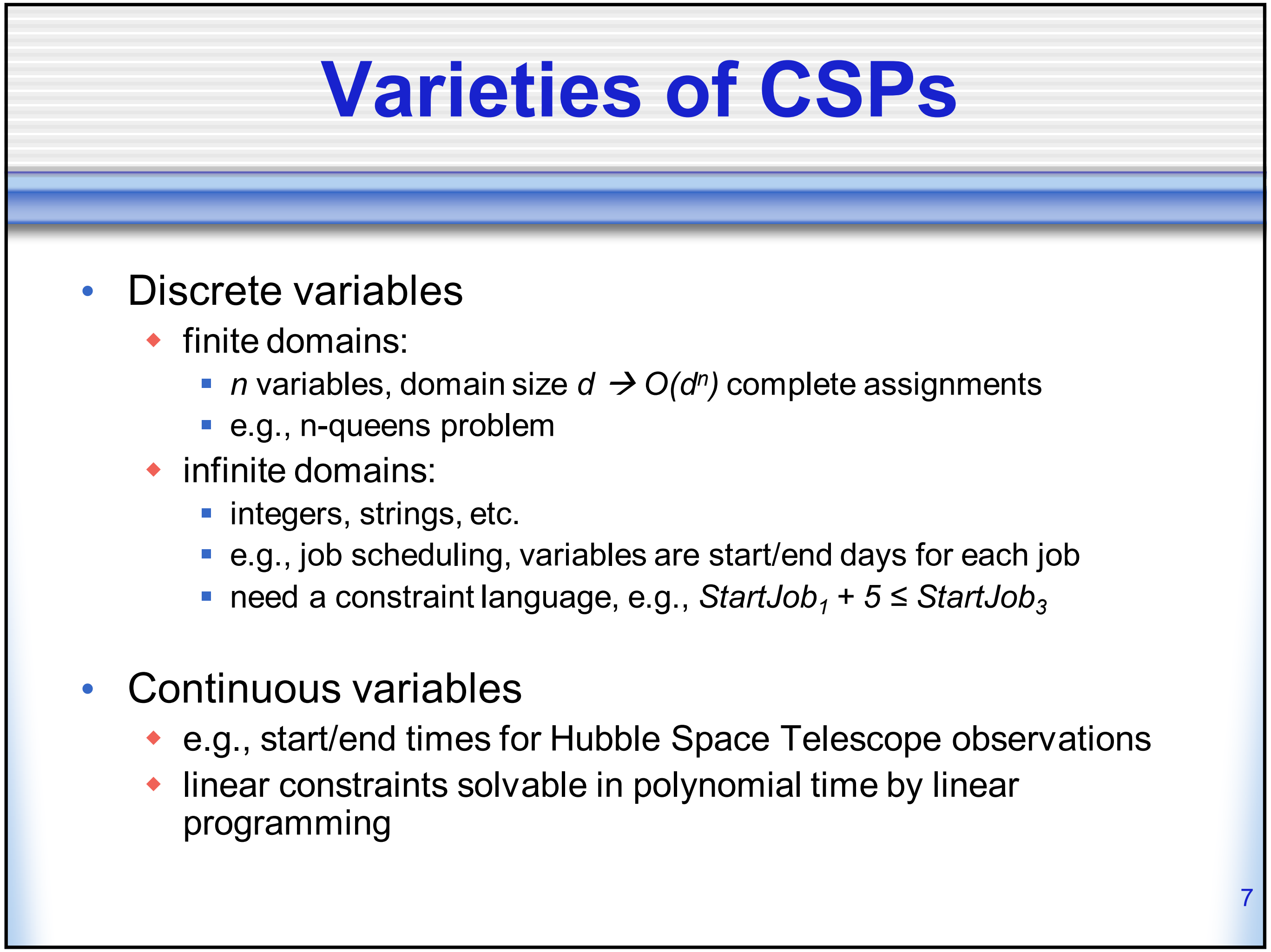
- Solutions are complete and consistent assignments, e.g.,
  - WA = red, NT = green, Q = red, NSW = green, V = red, SA = blue, T = green

# Constraint graph

- Binary CSP: each constraint relates two variables
- Constraint graph: nodes are variables, arcs are constraints



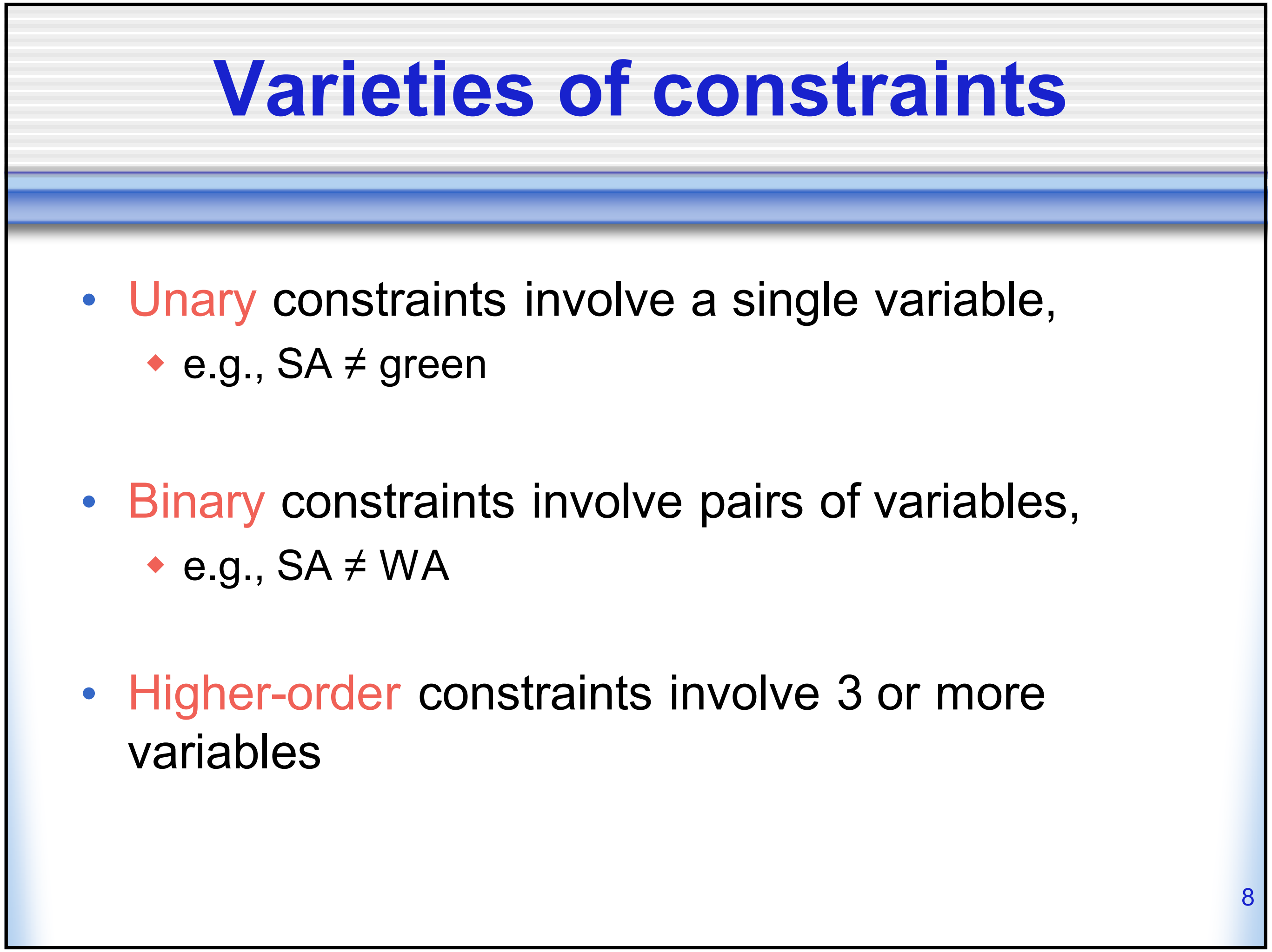
General-purpose CSP algorithms use the graph structure to speed up search. E.g., Tasmania is an independent sub problem!



# Varieties of CSPs

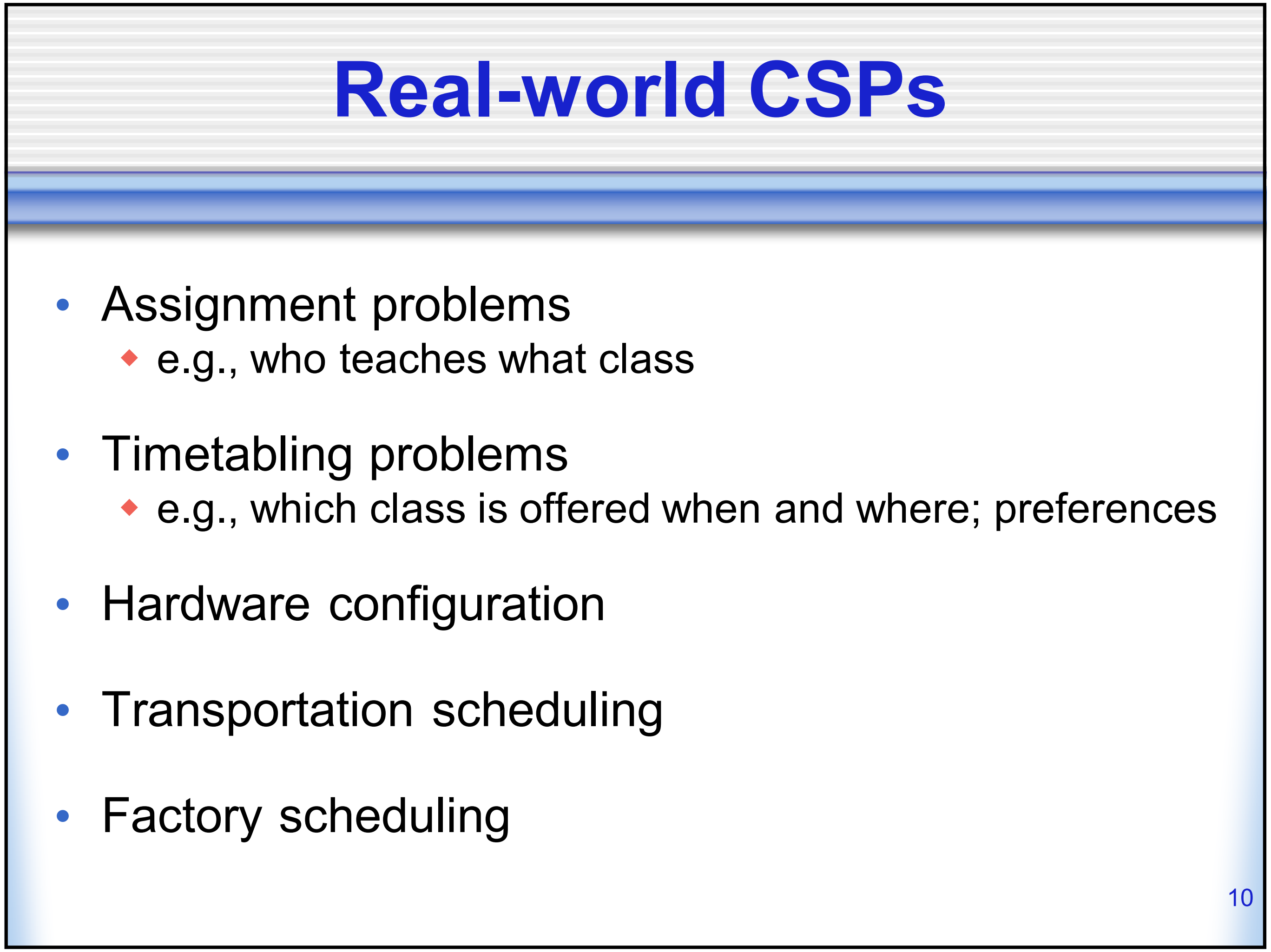
- Discrete variables
  - finite domains:
    - *n* variables, domain size *d* → $O(d^n)$ complete assignments
    - e.g., n-queens problem
  - infinite domains:
    - integers, strings, etc.
    - e.g., job scheduling, variables are start/end days for each job
    - need a constraint language, e.g., $StartJob_1 + 5 \leq StartJob_3$

- Continuous variables
  - e.g., start/end times for Hubble Space Telescope observations
  - linear constraints solvable in polynomial time by linear programming



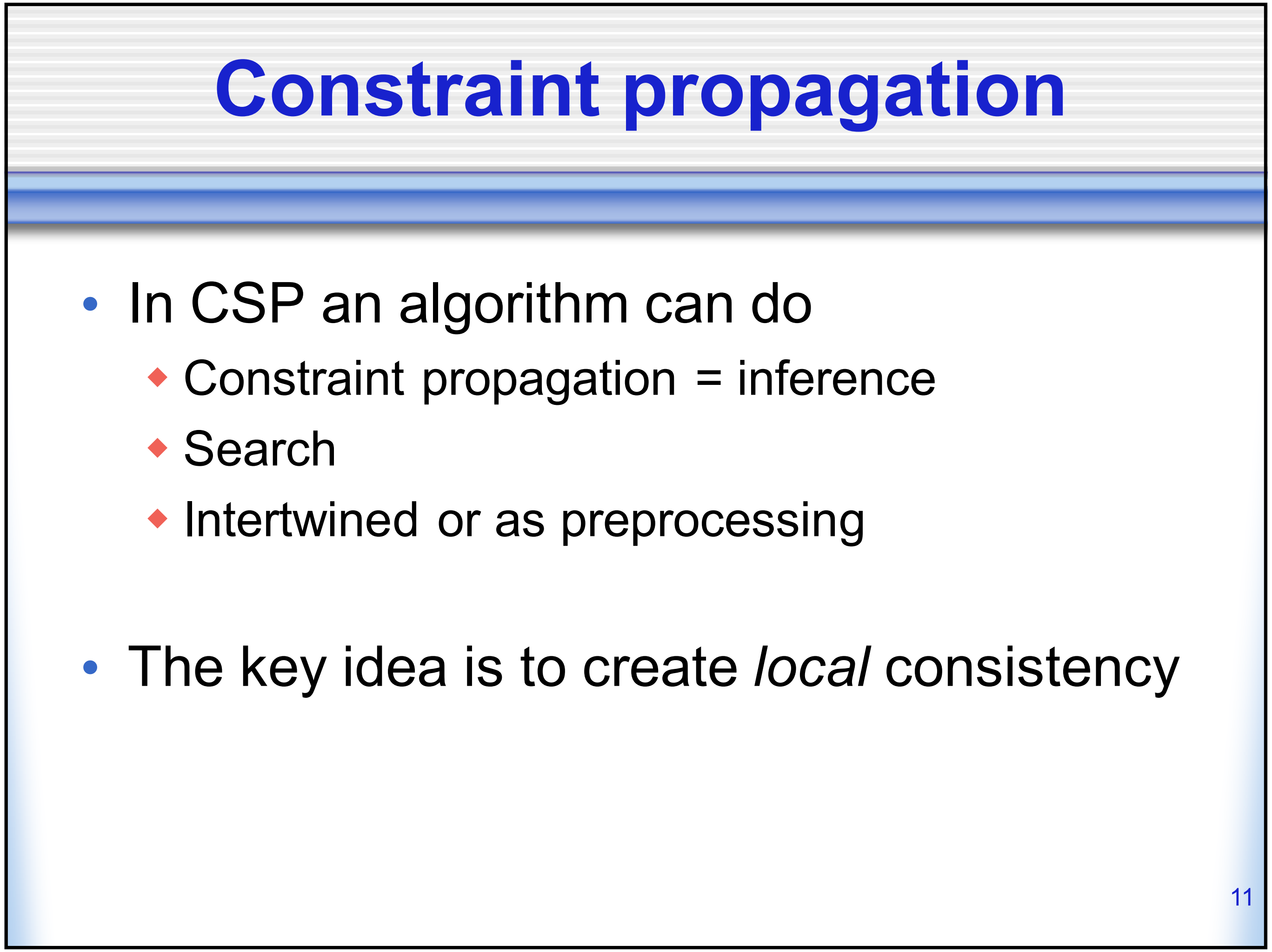
# Varieties of constraints

- Unary constraints involve a single variable,
  - e.g., SA ≠ green
- Binary constraints involve pairs of variables,
  - e.g., SA ≠ WA
- Higher-order constraints involve 3 or more variables



# Real-world CSPs

- Assignment problems
  - e.g., who teaches what class
- Timetabling problems
  - e.g., which class is offered when and where; preferences
- Hardware configuration
- Transportation scheduling
- Factory scheduling



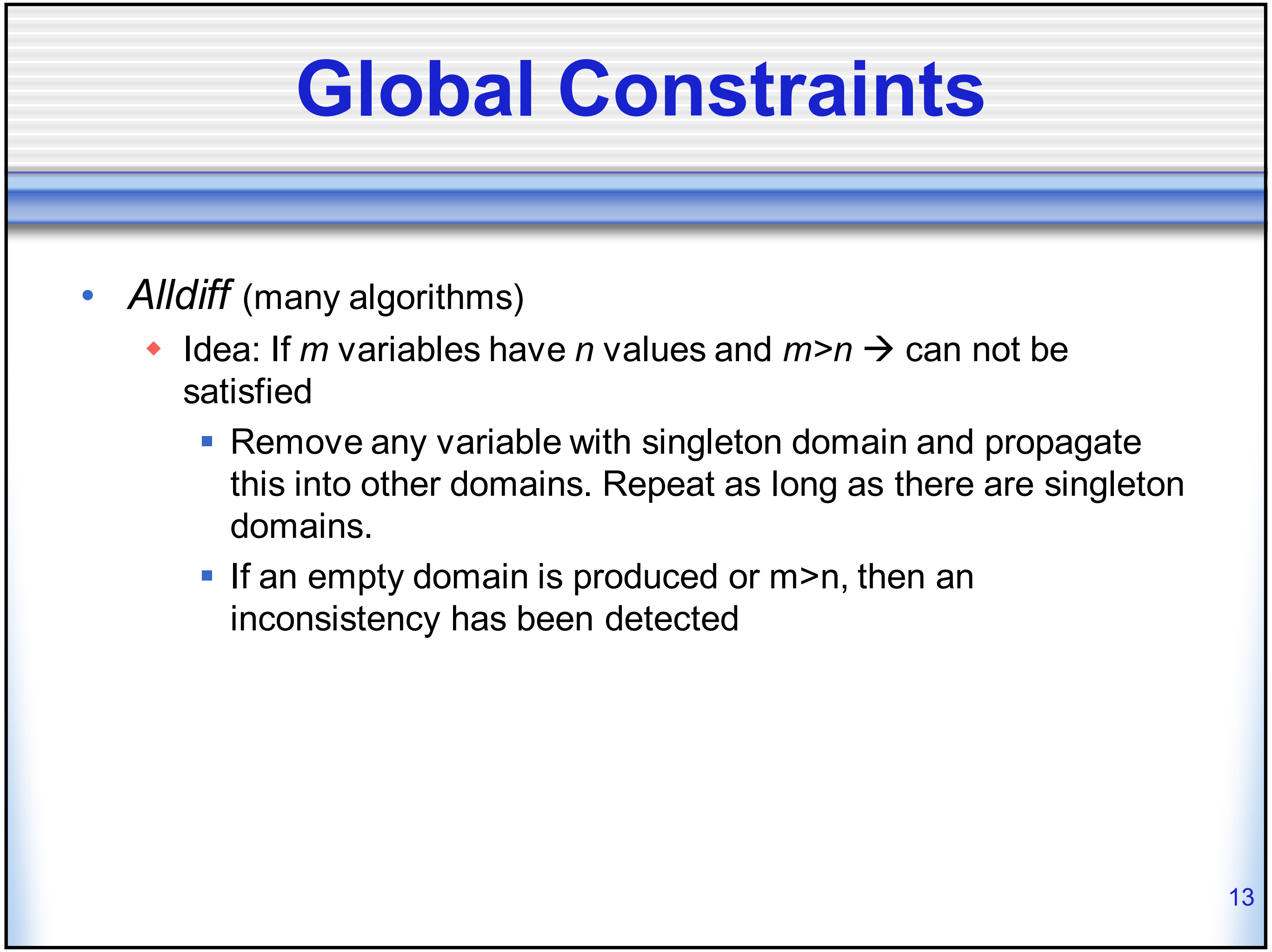
# Constraint propagation

- In CSP an algorithm can do
  - Constraint propagation = inference
  - Search
  - Intertwined or as preprocessing

- The key idea is to create *local* consistency

# Node consistency

- A variable is node-consistent if all the values satisfy the unary constraints
- Infer the values that are legal for a variable,
  - e.g. if South Australia does not like green, eliminate it {red, blue}
  - e.g. don’t want to teach at 8 pm

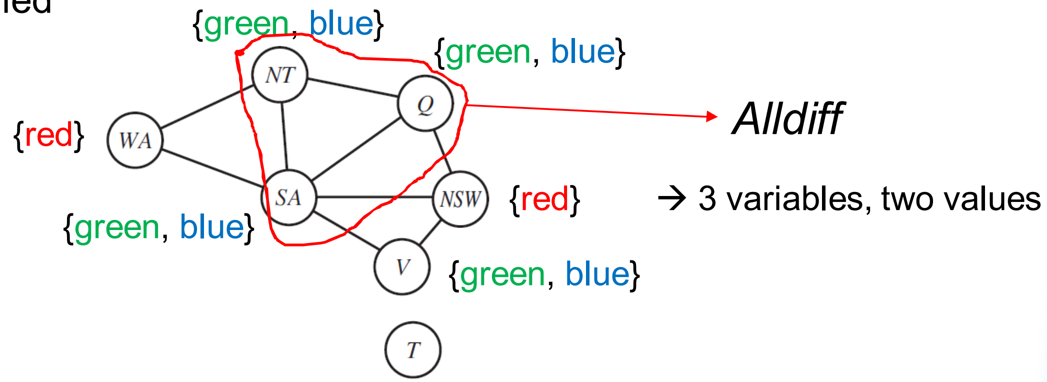


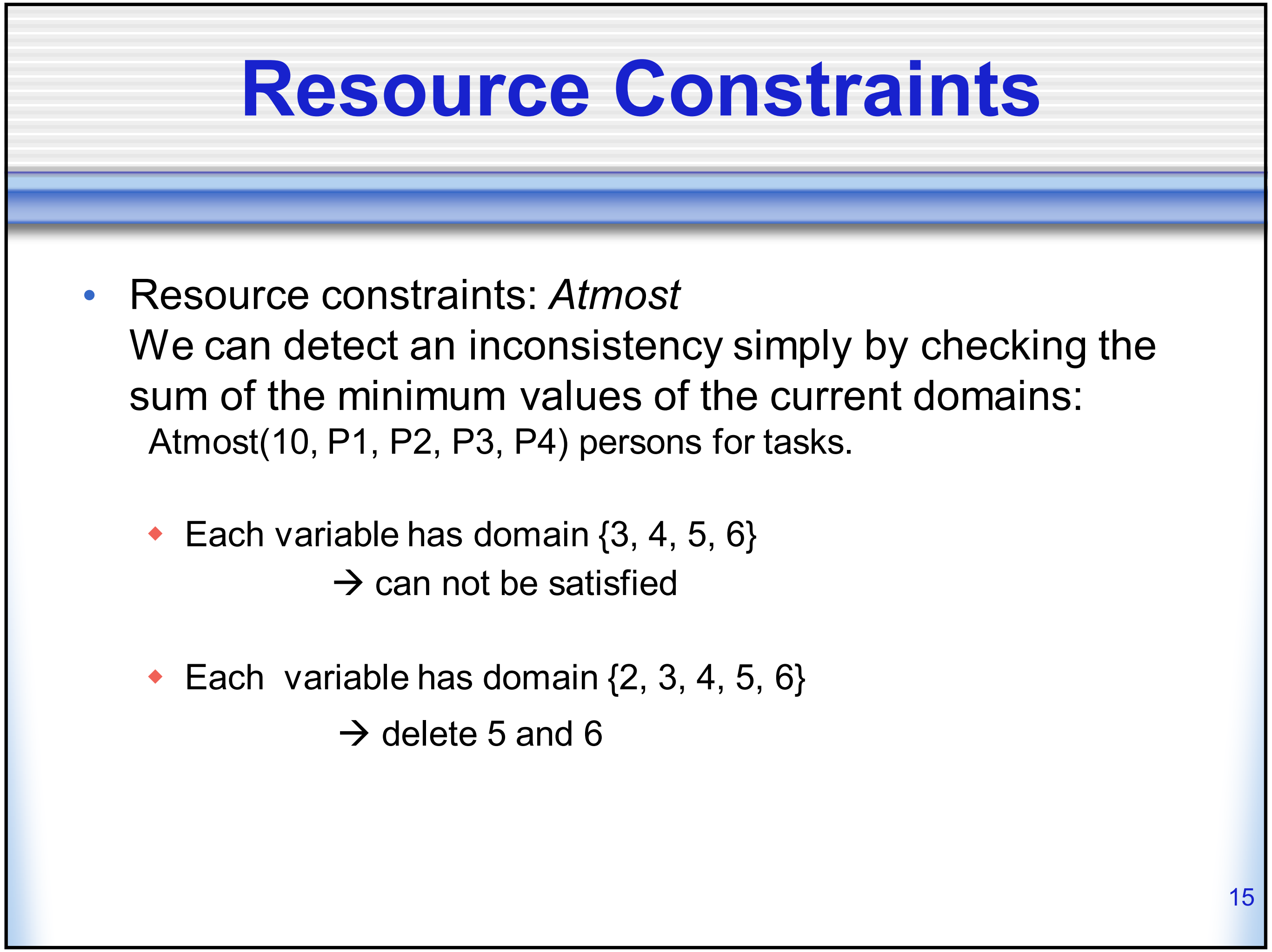
# Global Constraints

- *Alldiff* (many algorithms)
  - Idea: If *m* variables have *n* values and *m>n* → can not be satisfied
    - Remove any variable with singleton domain and propagate this into other domains. Repeat as long as there are singleton domains.
    - If an empty domain is produced or m>n, then an inconsistency has been detected

# Global Constraints

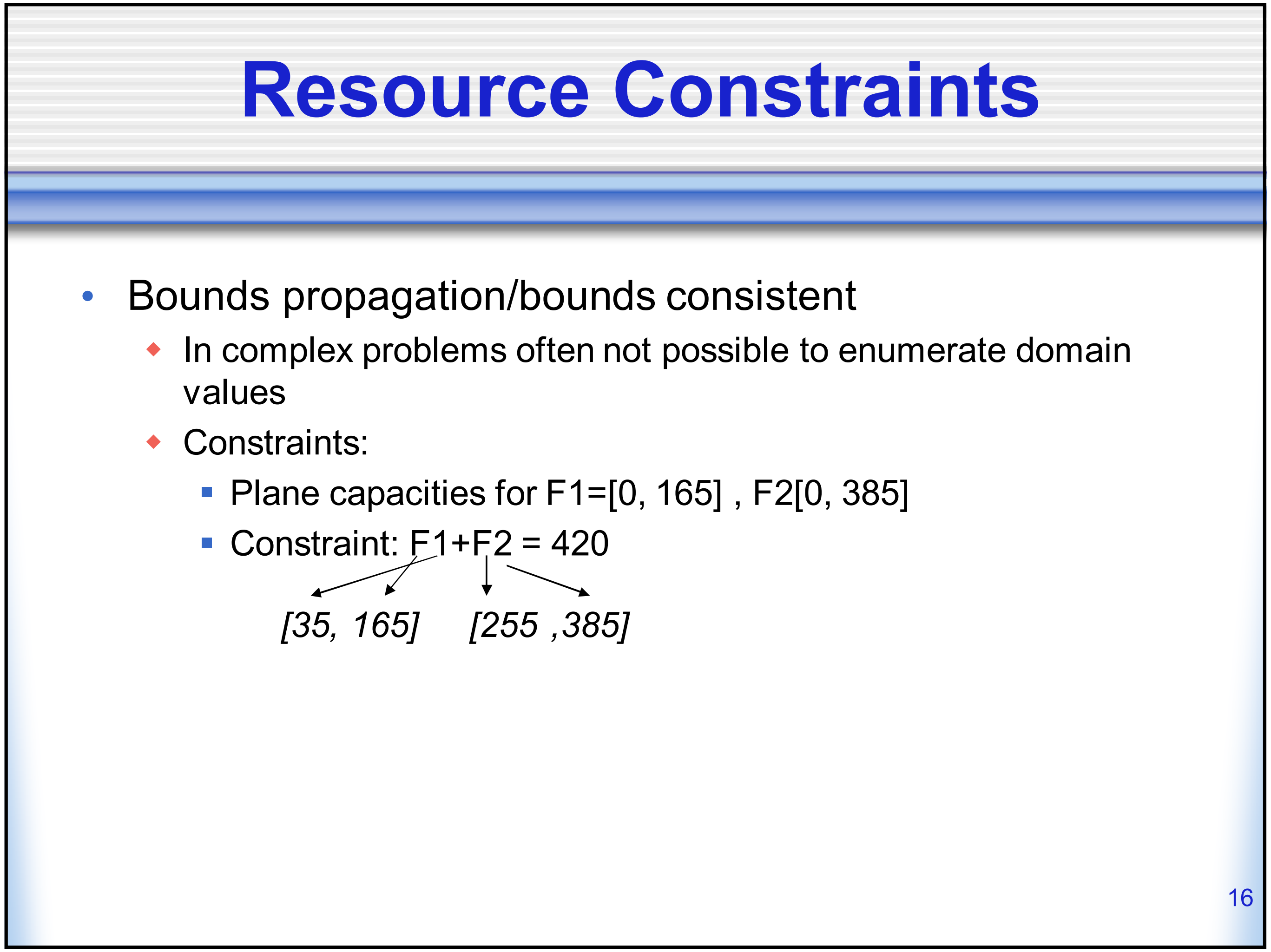
- *Alldiff* (many algorithms)
  - Idea: If *m* variables have *n* values and *m>n* → can not be satisfied





# Resource Constraints

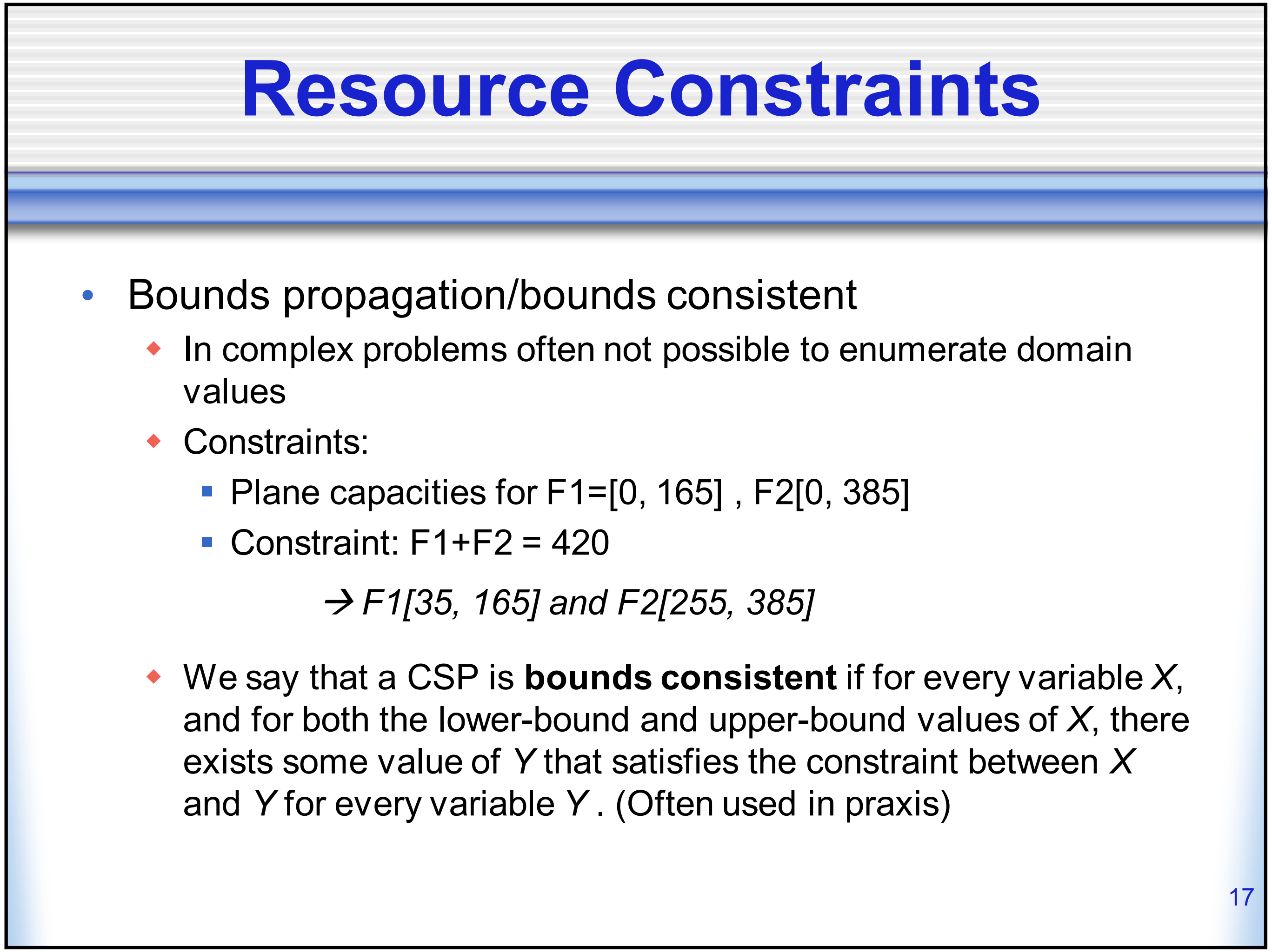
- Resource constraints: *Atmost*
  We can detect an inconsistency simply by checking the sum of the minimum values of the current domains:
  Atmost(10, P1, P2, P3, P4) persons for tasks.
  - Each variable has domain {3, 4, 5, 6}
    → can not be satisfied
  - Each  variable has domain {2, 3, 4, 5, 6}
    → delete 5 and 6



# Resource Constraints

- Bounds propagation/bounds consistent
  - In complex problems often not possible to enumerate domain values
  - Constraints:
    - Plane capacities for F1=[0, 165] , F2[0, 385]
    - Constraint: F1+F2 = 420

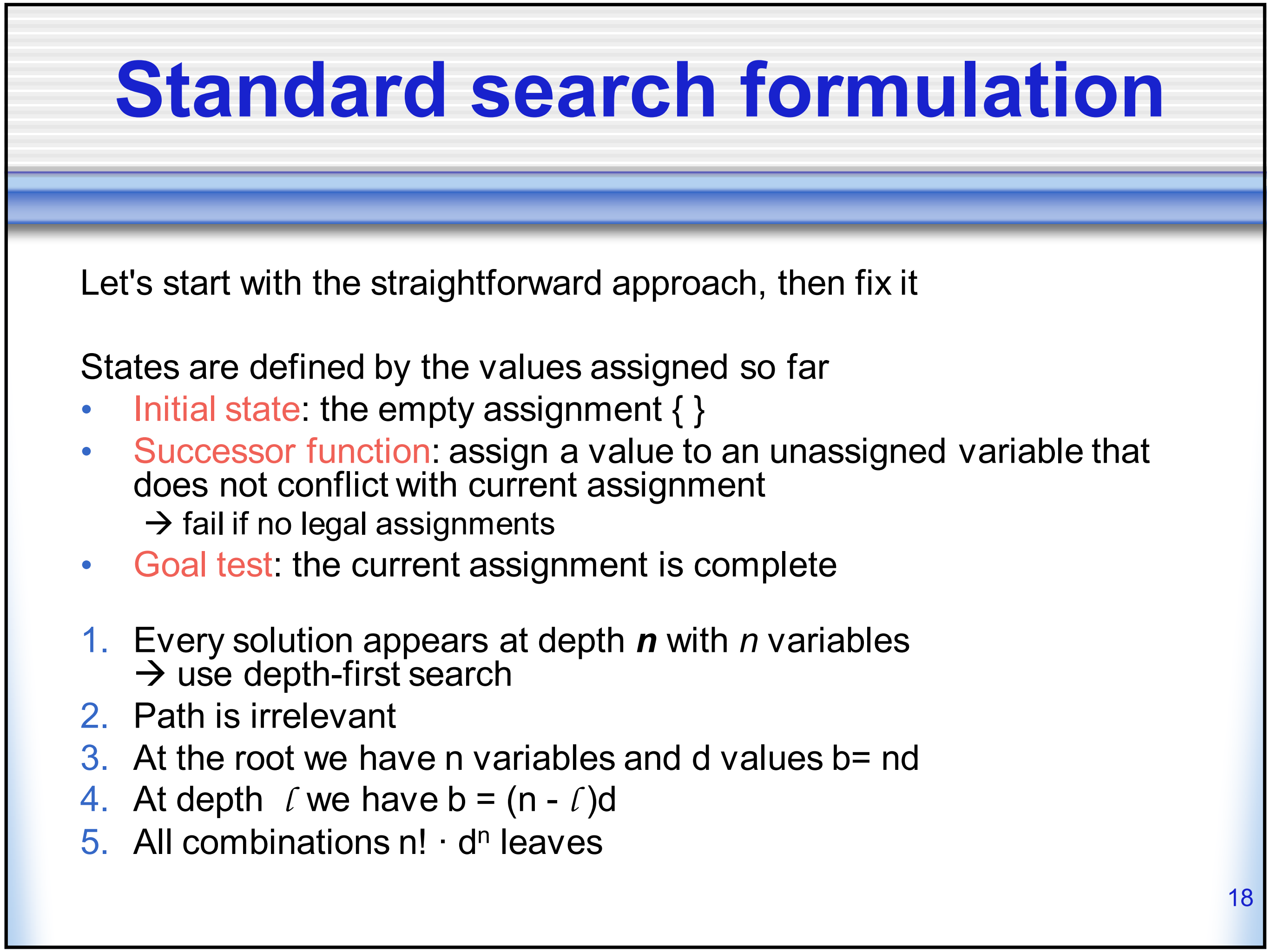
*[35, 165]* *[255 ,385]*



# Resource Constraints

- Bounds propagation/bounds consistent
  - In complex problems often not possible to enumerate domain values
  - Constraints:
    - Plane capacities for F1=[0, 165] , F2[0, 385]
    - Constraint: F1+F2 = 420

      → *F1[35, 165] and F2[255, 385]*
  - We say that a CSP is **bounds consistent** if for every variable *X*, and for both the lower-bound and upper-bound values of *X*, there exists some value of *Y* that satisfies the constraint between *X* and *Y* for every variable *Y* . (Often used in praxis)



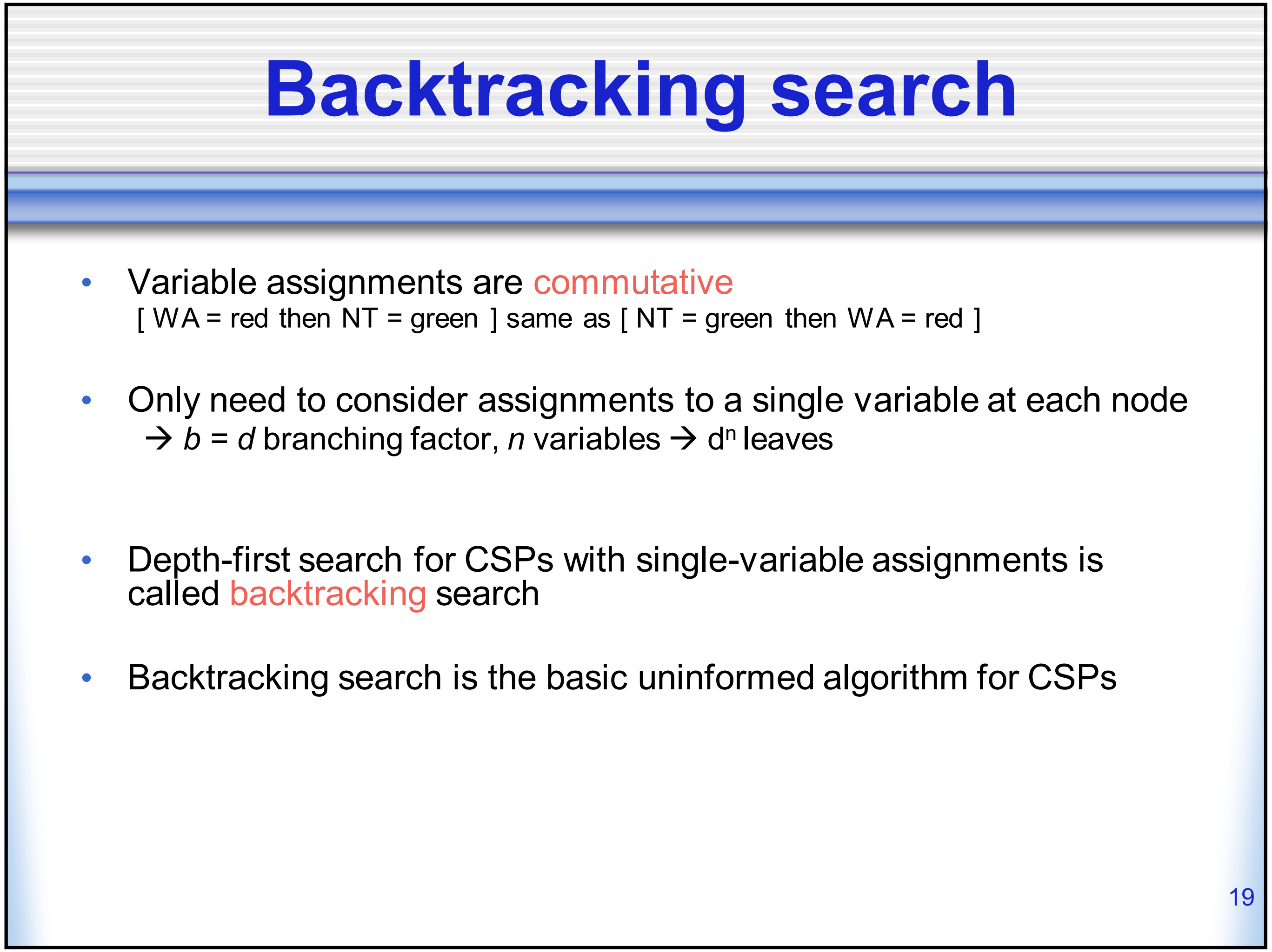
# Standard search formulation

Let's start with the straightforward approach, then fix it

States are defined by the values assigned so far

- Initial state: the empty assignment { }
- Successor function: assign a value to an unassigned variable that does not conflict with current assignment
  → fail if no legal assignments
- Goal test: the current assignment is complete

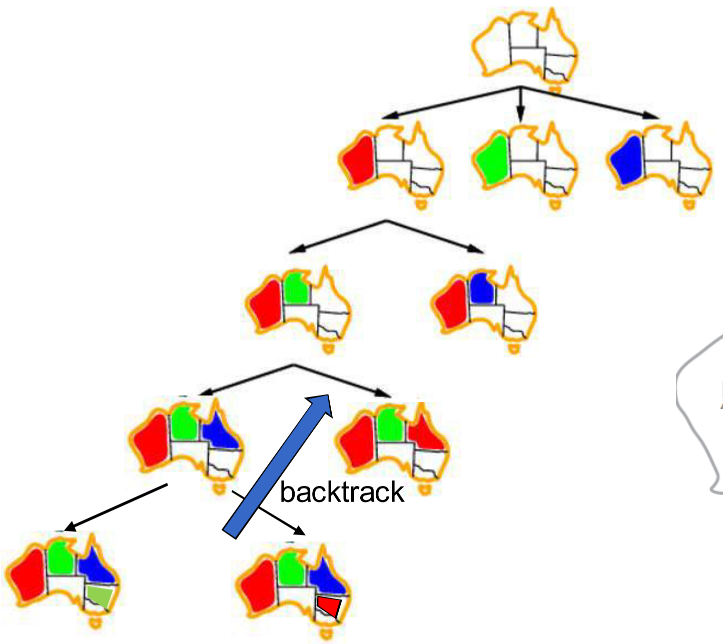
1. Every solution appears at depth ***n*** with *n* variables
   → use depth-first search
2. Path is irrelevant
3. At the root we have n variables and d values b= nd
4. At depth $\ell$ we have b = (n - $\ell$)d
5. All combinations $n! \cdot d^n$ leaves

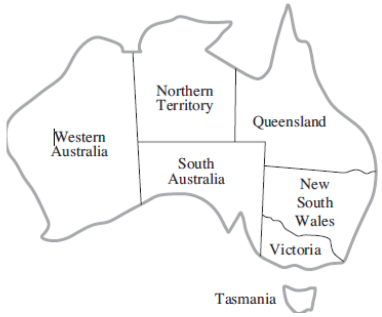


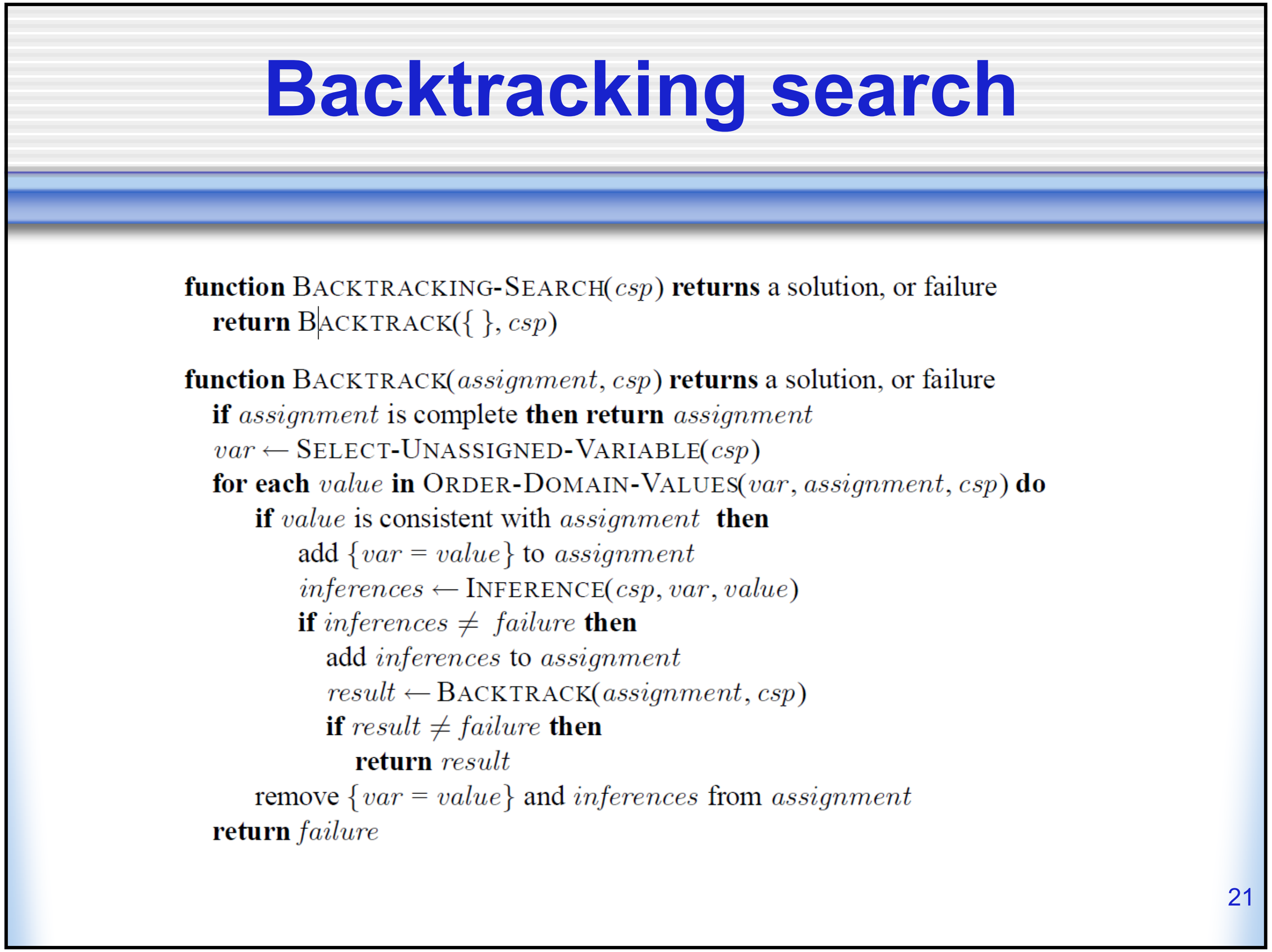
# Backtracking search

- Variable assignments are commutative
  [ WA = red then NT = green ] same as [ NT = green then WA = red ]
- Only need to consider assignments to a single variable at each node
  → *b* = *d* branching factor, *n* variables → $d^n$ leaves
- Depth-first search for CSPs with single-variable assignments is called backtracking search
- Backtracking search is the basic uninformed algorithm for CSPs

# Backtracking example







# Backtracking search

```
function BACKTRACKING-SEARCH(csp) returns a solution, or failure
  return BACKTRACK({ }, csp)

function BACKTRACK(assignment, csp) returns a solution, or failure
  if assignment is complete then return assignment
  var $\leftarrow$ SELECT-UNASSIGNED-VARIABLE(csp)
  for each value in ORDER-DOMAIN-VALUES(var, assignment, csp) do
     if value is consistent with assignment then
        add {var = value} to assignment
        inferences $\leftarrow$ INFERENCE(csp, var, value)
        if inferences $\neq$ failure then
           add inferences to assignment
           result $\leftarrow$ BACKTRACK(assignment, csp)
           if result $\neq$ failure then
              return result
     remove {var = value} and inferences from assignment
  return failure
```

21

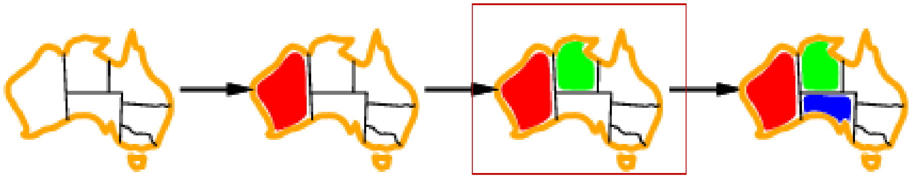
# Improving backtracking efficiency

- General-purpose methods can give huge gains in speed:
  - Which variable should be assigned next SELECT-UNASSIGNED-VARIABLE?
  - In what order should its values be tried ORDER-DOMAIN-VALUES?
  - What inferences should be performed at each step in the search INFERENCE?
  - Can we detect inevitable failure early?

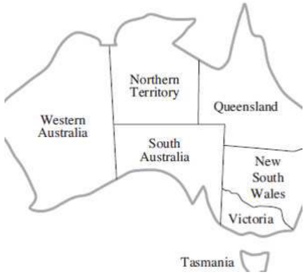
# Most constrained variable

- Most constrained variable:

  choose the variable with the fewest legal values



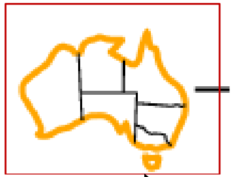
- a.k.a. minimum remaining values (MRV) heuristic



# Most constrained variable

- Most constrained variable:

  choose the variable with the fewest legal values



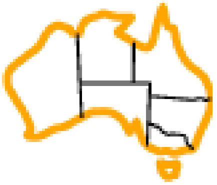
- a.k.a. minimum remaining values (MRV) heuristic

What about the first state

MRV does not help in the first state

# Degree heuristic

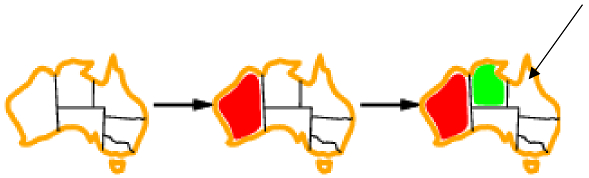
- Tie-breaker among most constrained variables: Degree heuristic
- Most constraining variable:
  - choose the variable with the most constraints on remaining variables
  - used together with MRV



# Least constraining value

- Given a variable, choose the least constraining value:
  - the one that rules out the fewest values in the remaining variables

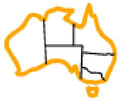
*Queensland is selected*



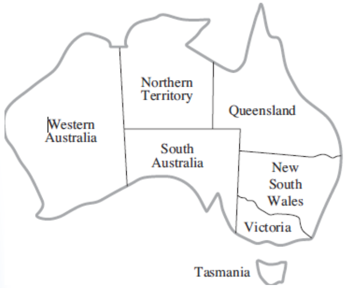
- Combining these heuristics makes 1000 queens feasible

# Inference: Forward checking

- Idea:
  - Keep track of remaining legal values for unassigned neighbors
  - Terminate search when any variable has no legal values



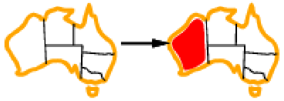
| WA | NT | Q | NSW | V | SA | T |
|---|---|---|---|---|---|---|
| 🟥🟩🟦 | 🟥🟩🟦 | 🟥🟩🟦 | 🟥🟩🟦 | 🟥🟩🟦 | 🟥🟩🟦 | 🟥🟩🟦 |



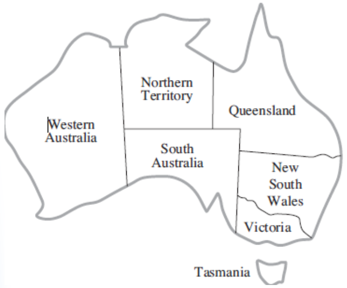
WA = red

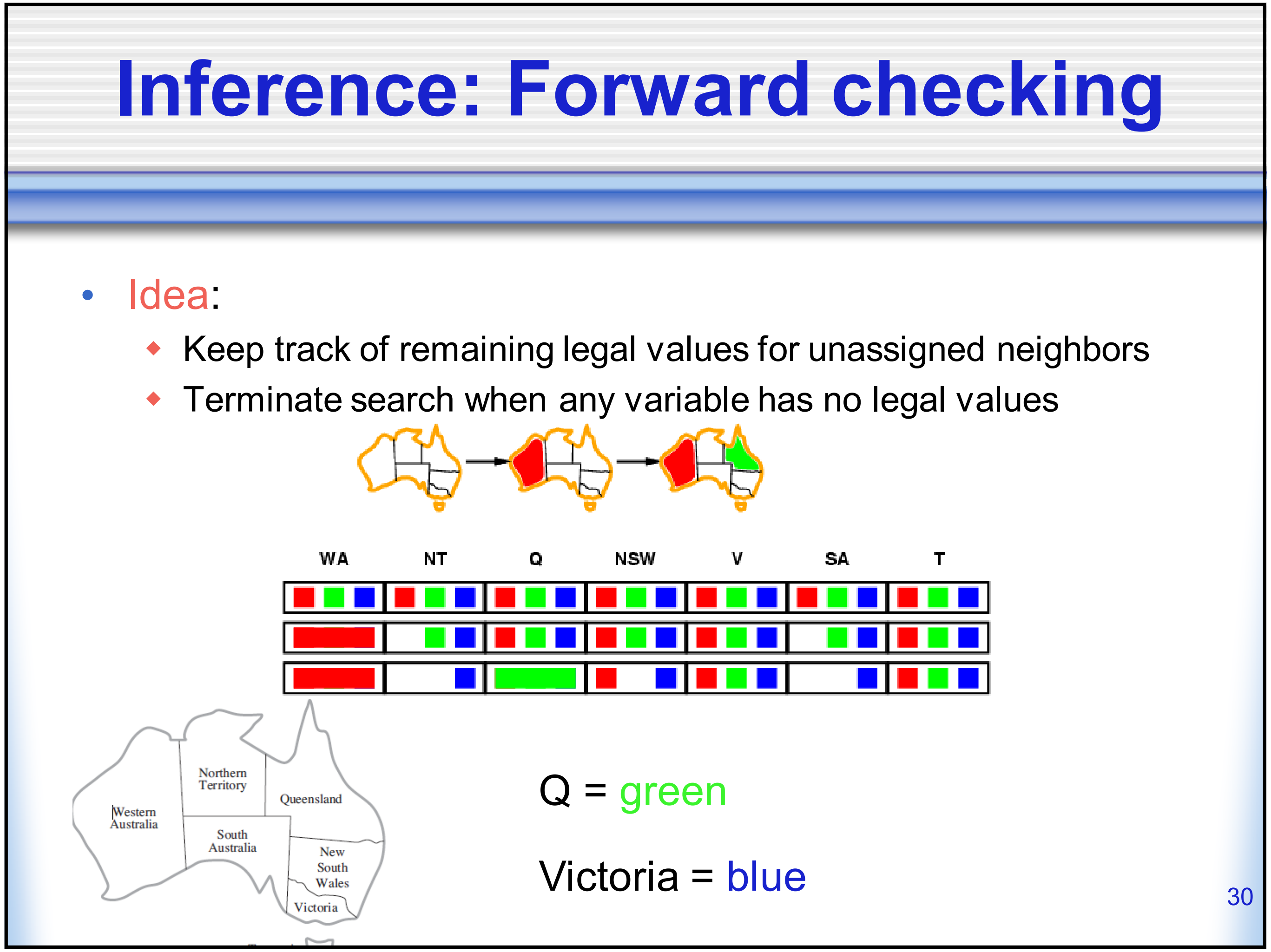
# Inference: Forward checking

- Idea:
  - Keep track of remaining legal values for unassigned neighbors
  - Terminate search when any variable has no legal values



| WA | NT | Q | NSW | V | SA | T |
|---|---|---|---|---|---|---|
| 🟥🟩🟦 | 🟥🟩🟦 | 🟥🟩🟦 | 🟥🟩🟦 | 🟥🟩🟦 | 🟥🟩🟦 | 🟥🟩🟦 |
| 🟥 | 🟩🟦 | 🟥🟩🟦 | 🟥🟩🟦 | 🟥🟩🟦 | 🟩🟦 | 🟥🟩🟦 |





# Inference: Forward checking

- Idea:
  - Keep track of remaining legal values for unassigned neighbors
  - Terminate search when any variable has no legal values

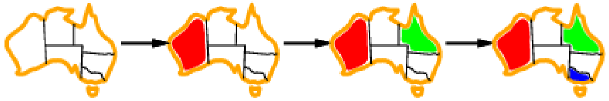
| WA | NT | Q | NSW | V | SA | T |
|---|---|---|---|---|---|---|
| 🟥🟩🟦 | 🟥🟩🟦 | 🟥🟩🟦 | 🟥🟩🟦 | 🟥🟩🟦 | 🟥🟩🟦 | 🟥🟩🟦 |
| 🟥 | 🟩🟦 | 🟥🟩🟦 | 🟥🟩🟦 | 🟥🟩🟦 | 🟩🟦 | 🟥🟩🟦 |
| 🟥 | 🟦 | 🟩 | 🟥🟦 | 🟥🟩🟦 | 🟦 | 🟥🟩🟦 |

Q = green

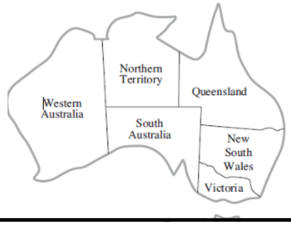
Victoria = blue

# Inference: Forward checking

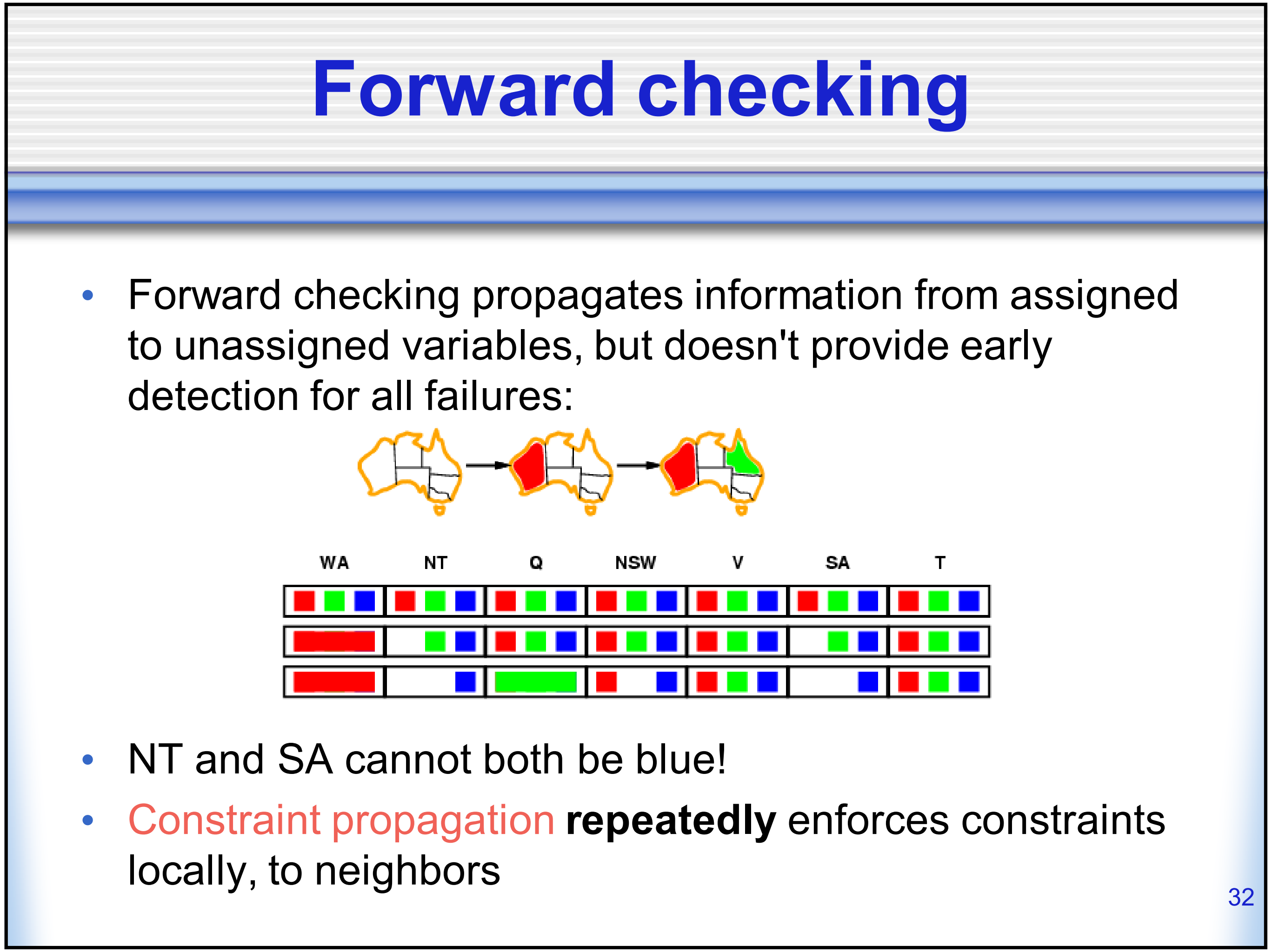
- Idea:
  - Keep track of remaining legal values for unassigned neighbors
  - Terminate search when any variable has no legal values



| WA | NT | Q | NSW | V | SA | T |
|---|---|---|---|---|---|---|
| 🟥 🟩 🟦 | 🟥 🟩 🟦 | 🟥 🟩 🟦 | 🟥 🟩 🟦 | 🟥 🟩 🟦 | 🟥 🟩 🟦 | 🟥 🟩 🟦 |
| 🟥 | 🟩 🟦 | 🟥 🟩 🟦 | 🟥 🟩 🟦 | 🟥 🟩 🟦 | 🟩 🟦 | 🟥 🟩 🟦 |
| 🟥 | 🟦 | 🟩 | 🟥 🟦 | 🟥 🟩 🟦 | 🟦 | 🟥 🟩 🟦 |
| 🟥 | 🟦 | 🟩 | 🟥 | 🟦 |  | 🟥 🟩 🟦 |



Victoria = blue

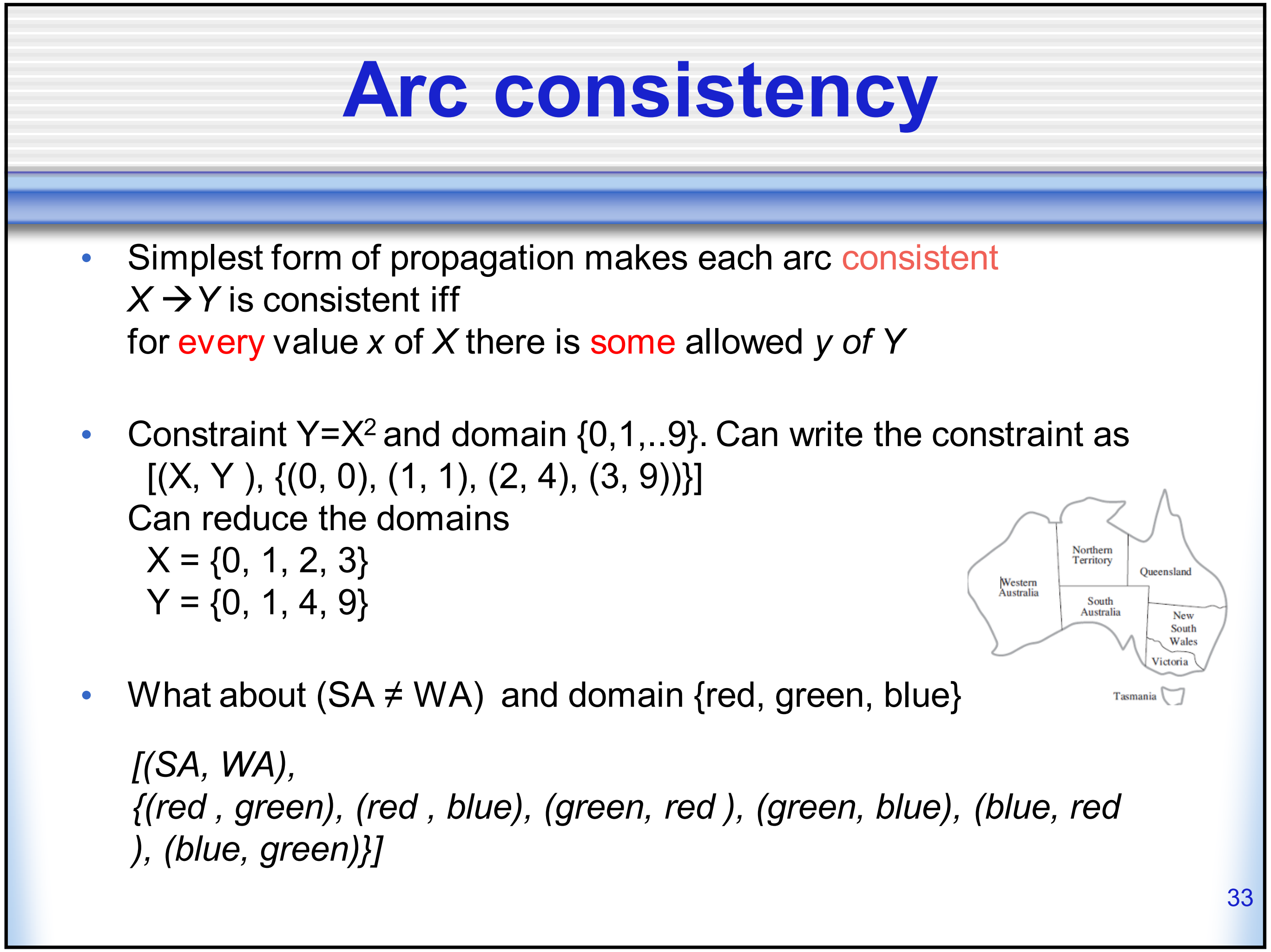


# Forward checking

- Forward checking propagates information from assigned to unassigned variables, but doesn't provide early detection for all failures:

| WA | NT | Q | NSW | V | SA | T |
|---|---|---|---|---|---|---|
| 🟥 🟩 🟦 | 🟥 🟩 🟦 | 🟥 🟩 🟦 | 🟥 🟩 🟦 | 🟥 🟩 🟦 | 🟥 🟩 🟦 | 🟥 🟩 🟦 |
| 🟥 | 🟩 🟦 | 🟥 🟩 🟦 | 🟥 🟩 🟦 | 🟥 🟩 🟦 | 🟩 🟦 | 🟥 🟩 🟦 |
| 🟥 | 🟦 | 🟩 | 🟥 🟦 | 🟥 🟩 🟦 | 🟦 | 🟥 🟩 🟦 |

- NT and SA cannot both be blue!
- Constraint propagation **repeatedly** enforces constraints locally, to neighbors



# Arc consistency

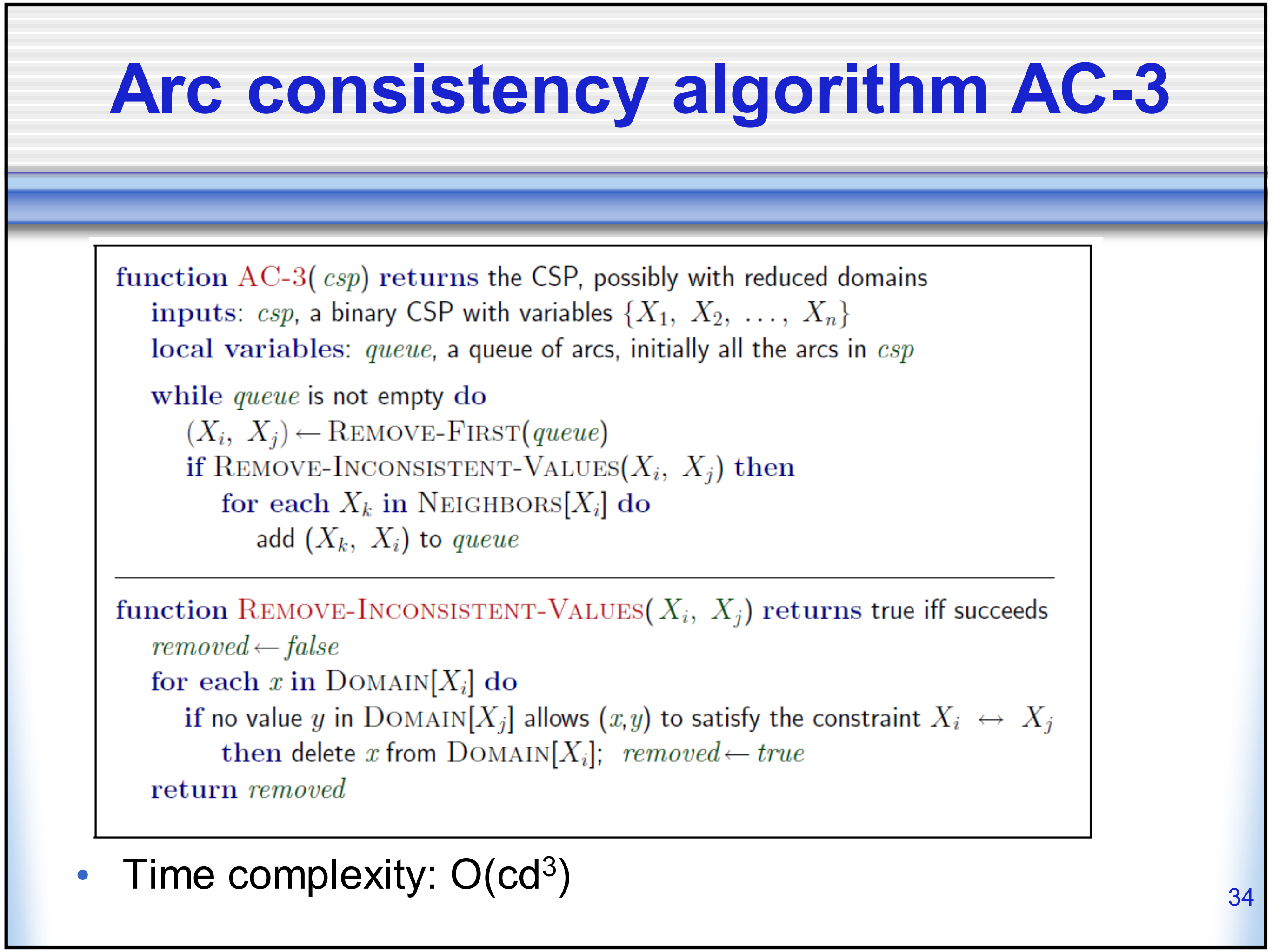
- Simplest form of propagation makes each arc consistent
  *X* → *Y* is consistent iff
  for every value *x* of *X* there is some allowed *y of Y*

- Constraint Y=X$^2$ and domain {0,1,..9}. Can write the constraint as
  [(X, Y ), {(0, 0), (1, 1), (2, 4), (3, 9))}]
  Can reduce the domains
  X = {0, 1, 2, 3}
  Y = {0, 1, 4, 9}

- What about (SA ≠ WA) and domain {red, green, blue}

  *[(SA, WA),*
  *{(red , green), (red , blue), (green, red ), (green, blue), (blue, red ), (blue, green)}]*

Northern Territory
Queensland
Western Australia
South Australia
New South Wales
Victoria
Tasmania



# Arc consistency algorithm AC-3

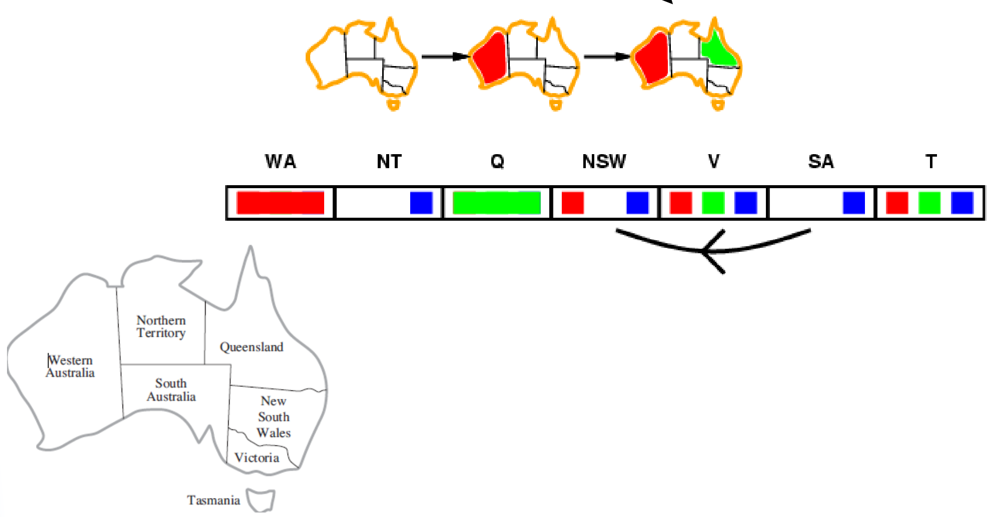
```
function AC-3( csp) returns the CSP, possibly with reduced domains
    inputs: csp, a binary CSP with variables $\{X_1, X_2, \ldots, X_n\}$
    local variables: queue, a queue of arcs, initially all the arcs in csp

    while queue is not empty do
        $(X_i, X_j) \leftarrow$ REMOVE-FIRST(queue)
        if REMOVE-INCONSISTENT-VALUES($X_i, X_j$) then
            for each $X_k$ in NEIGHBORS[$X_i$] do
                add ($X_k, X_i$) to queue
```

```
function REMOVE-INCONSISTENT-VALUES( $X_i, X_j$) returns true iff succeeds
    removed $\leftarrow$ false
    for each $x$ in DOMAIN[$X_i$] do
        if no value $y$ in DOMAIN[$X_j$] allows ($x,y$) to satisfy the constraint $X_i \leftrightarrow X_j$
            then delete $x$ from DOMAIN[$X_i$];  removed $\leftarrow$ true
    return removed
```

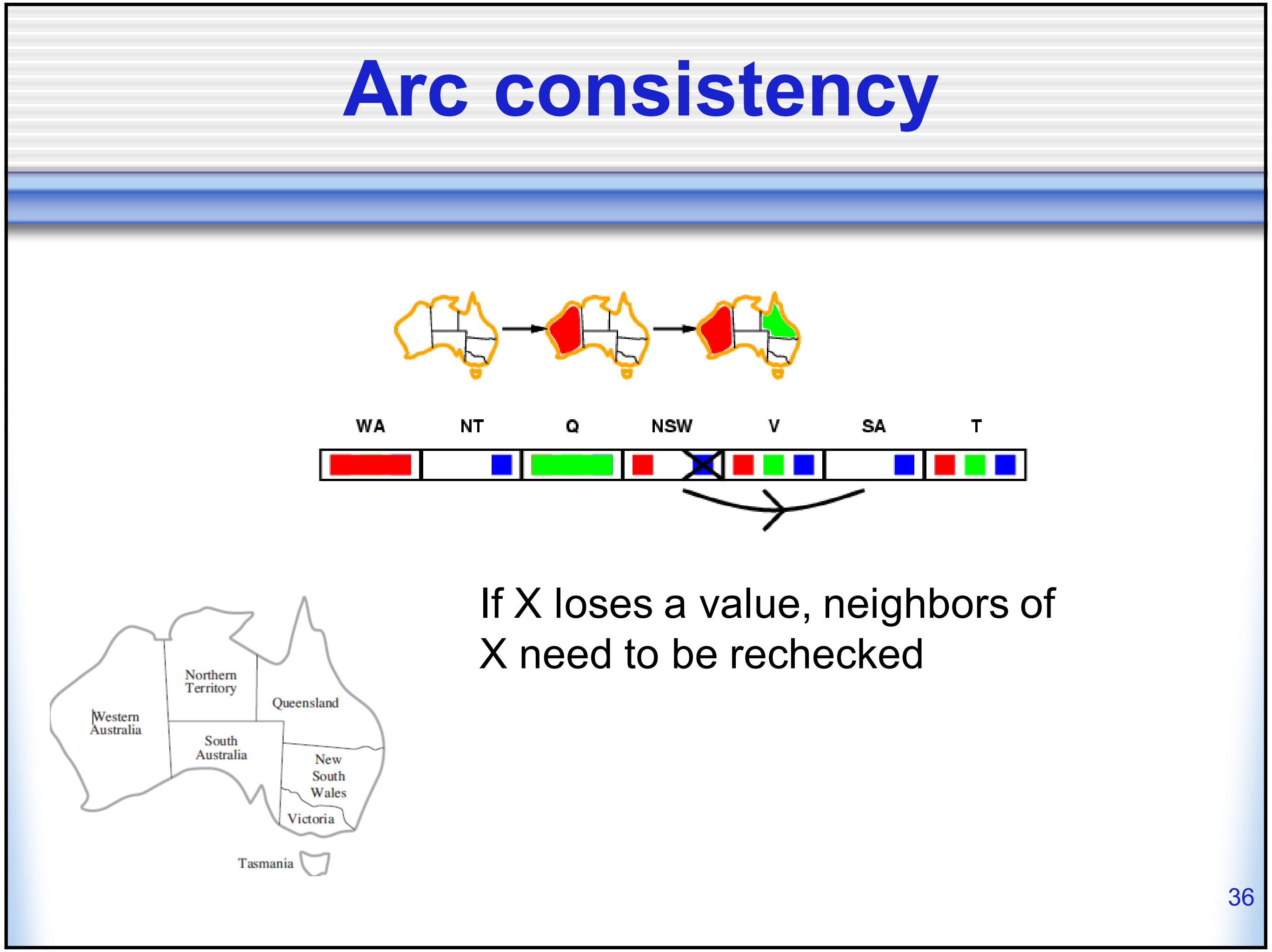
- Time complexity: O(cd$^3$)

# Arc consistency

- Assume we begin in state



| WA | NT | Q | NSW | V | SA | T |
|---|---|---|---|---|---|---|
| 🟥 | 🟦 | 🟩 | 🟥 🟦 | 🟥 🟩 🟦 | 🟦 | 🟥 🟩 🟦 |



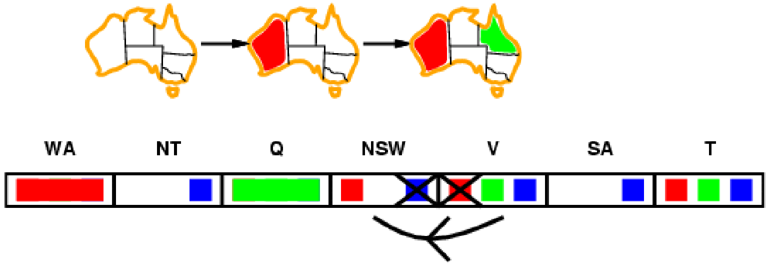
# Arc consistency

| WA | NT | Q | NSW | V | SA | T |
|---|---|---|---|---|---|---|
| 🟥 | 🟦 | 🟩 | 🟥 ~~🟦~~ | 🟥 🟩 🟦 | 🟦 | 🟥 🟩 🟦 |

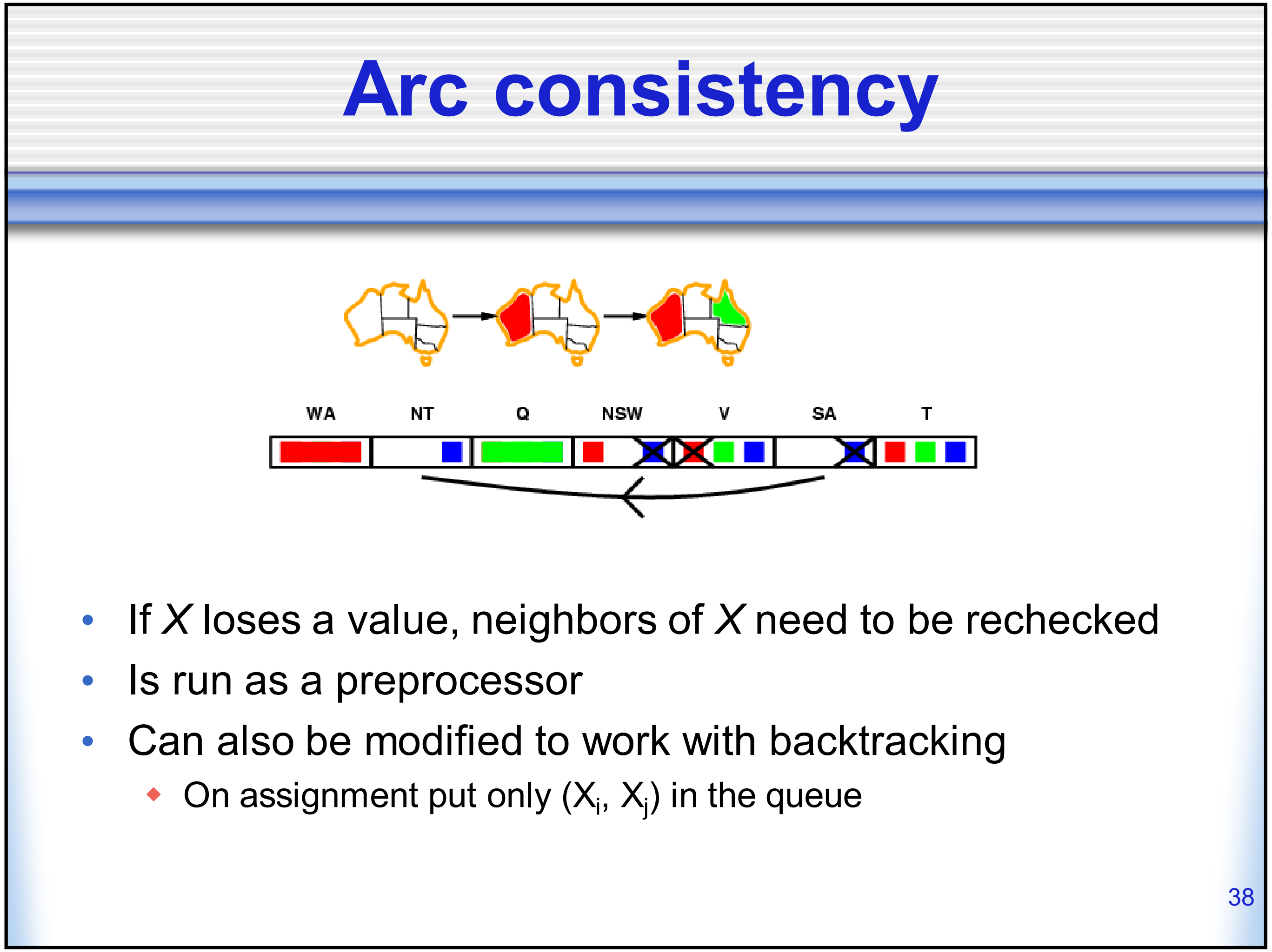
If X loses a value, neighbors of X need to be rechecked

# Arc consistency

Western Australia
Northern Territory
Queensland
South Australia
New South Wales
Victoria
Tasmania

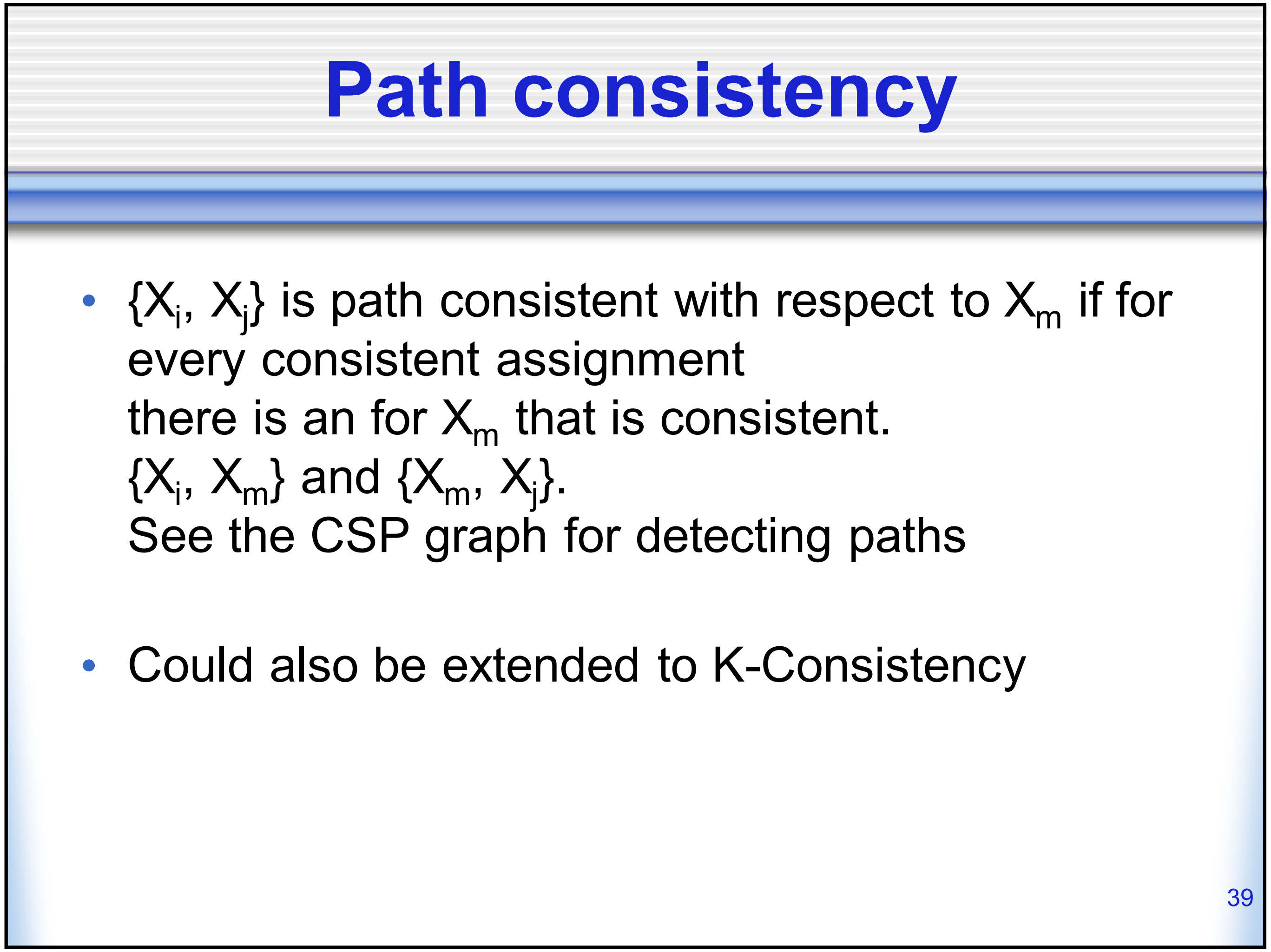


- If *X* loses a value, neighbors of *X* need to be rechecked


WA
NT
Q
NSW
V
SA
T

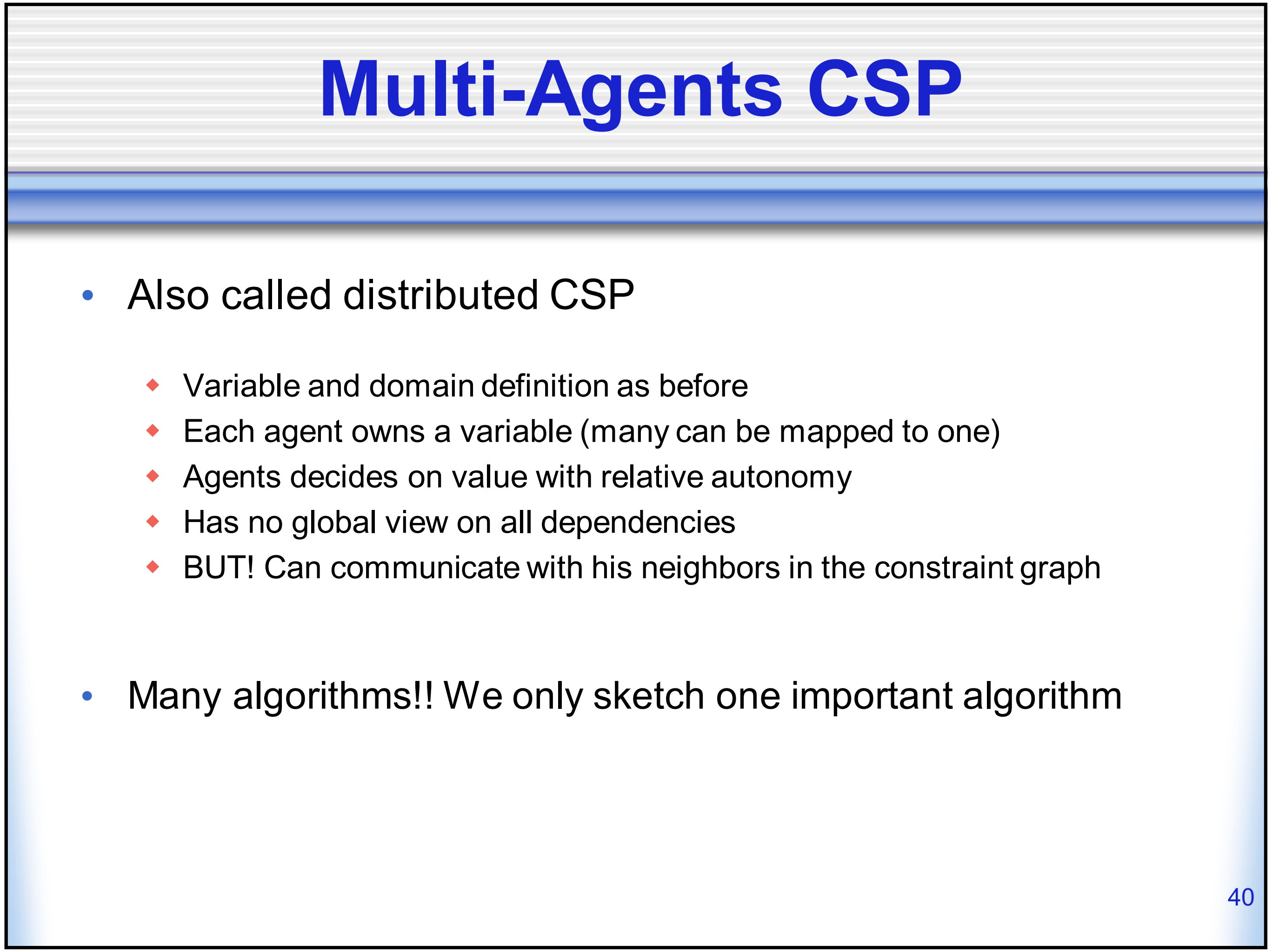
# Arc consistency

- If *X* loses a value, neighbors of *X* need to be rechecked
- Is run as a preprocessor
- Can also be modified to work with backtracking
  - On assignment put only ($X_i$, $X_j$) in the queue



# Path consistency

- $\{X_i, X_j\}$ is path consistent with respect to $X_m$ if for every consistent assignment  
there is an for $X_m$ that is consistent.  
$\{X_i, X_m\}$ and $\{X_m, X_j\}$.  
See the CSP graph for detecting paths
- Could also be extended to K-Consistency

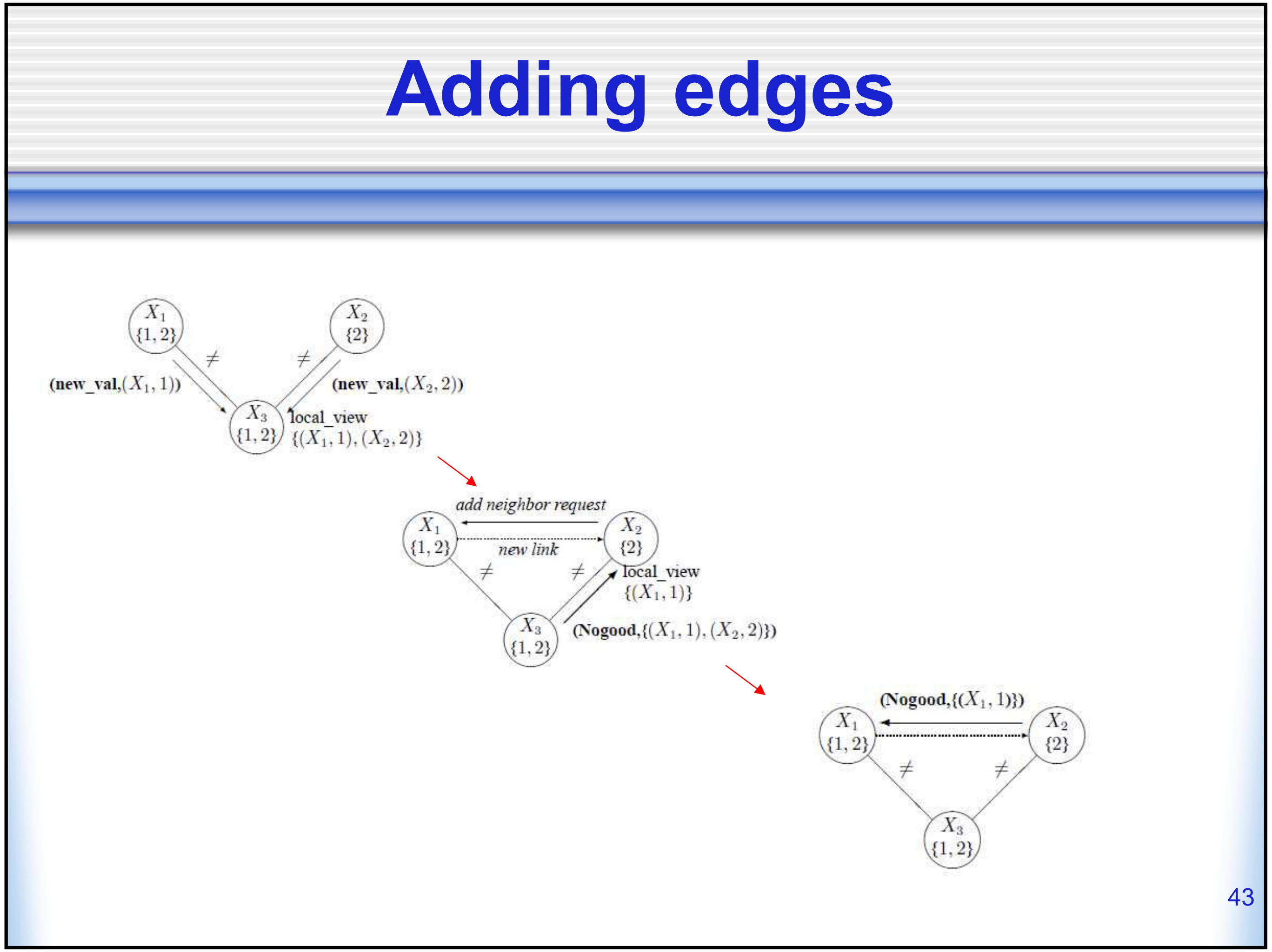


# Multi-Agents CSP

- Also called distributed CSP
  - Variable and domain definition as before
  - Each agent owns a variable (many can be mapped to one)
  - Agents decides on value with relative autonomy
  - Has no global view on all dependencies
  - BUT! Can communicate with his neighbors in the constraint graph
- Many algorithms!! We only sketch one important algorithm

# Multi-Agents CSP: Asynchronous Backtracking

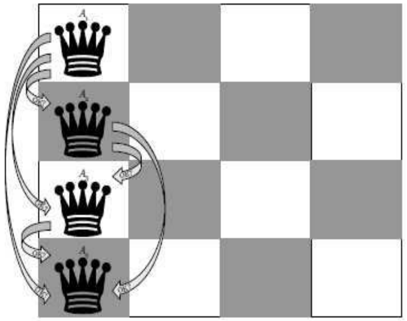
- The algorithm makes an ordering on agents and assigns them priority numbers. All agents set their initial value concurrently
- a higher-priority agent *j* informs all lower-priority agents $k_i$ of its assignment if connected in constraint graph
- lower-priority agent *k* evaluates the shared $C_{jk}$ constraint with its own assignment
  - if constraints are satisfied with the current assignment → no action
  - otherwise, agent k looks for a different value consistent with choice of agent j
  - if such a consistent value exists → agent j adopts this value and informs other low-priority agents
  - if such a consistent value does not exist, agent j *updates NoGood list* and sends the message to agent j and seek for a value that is consistent with all connected higher priority agents
  - j receives a NoGood mentioning i it is not connected with j. j asks i to set up a link



# Adding edges

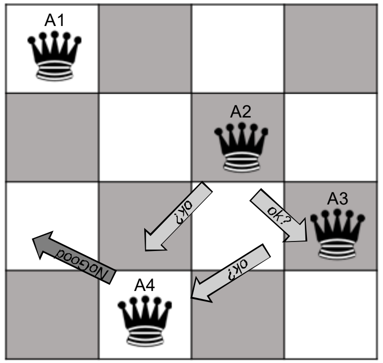
43

# Example: 4-Queens

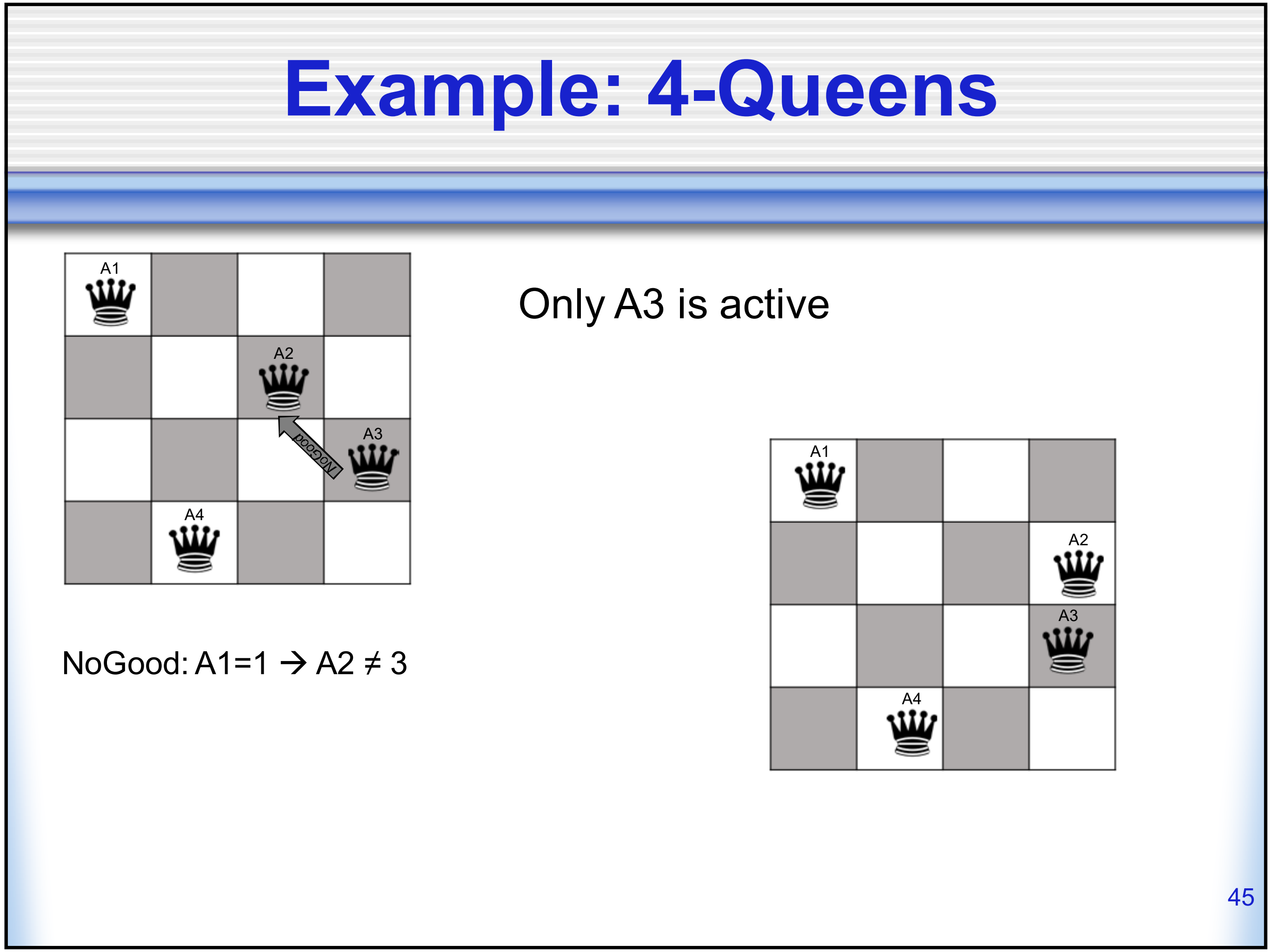


A1 knows no position
A2 knows A1
A3 knows A2 and A1
A4 knows all positions

Based on local information each queen checks where to move or to resolve conflicts with upper queen. Afterwards do nothing, send “OK?” or “NoGood” messages.



NoGood: A1=1 and A2=1 → A3 ≠ 1

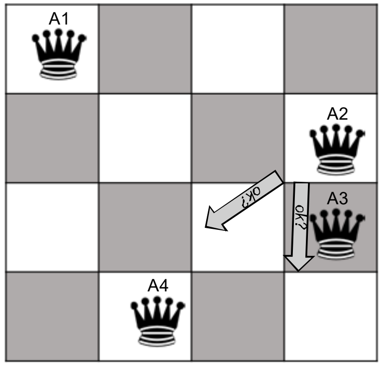


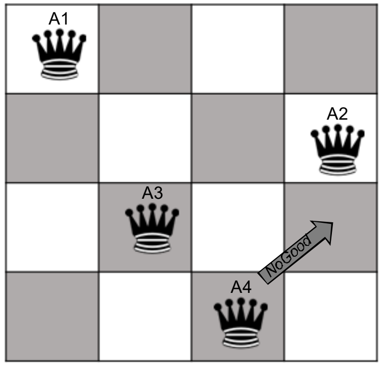
# Example: 4-Queens

Only A3 is active

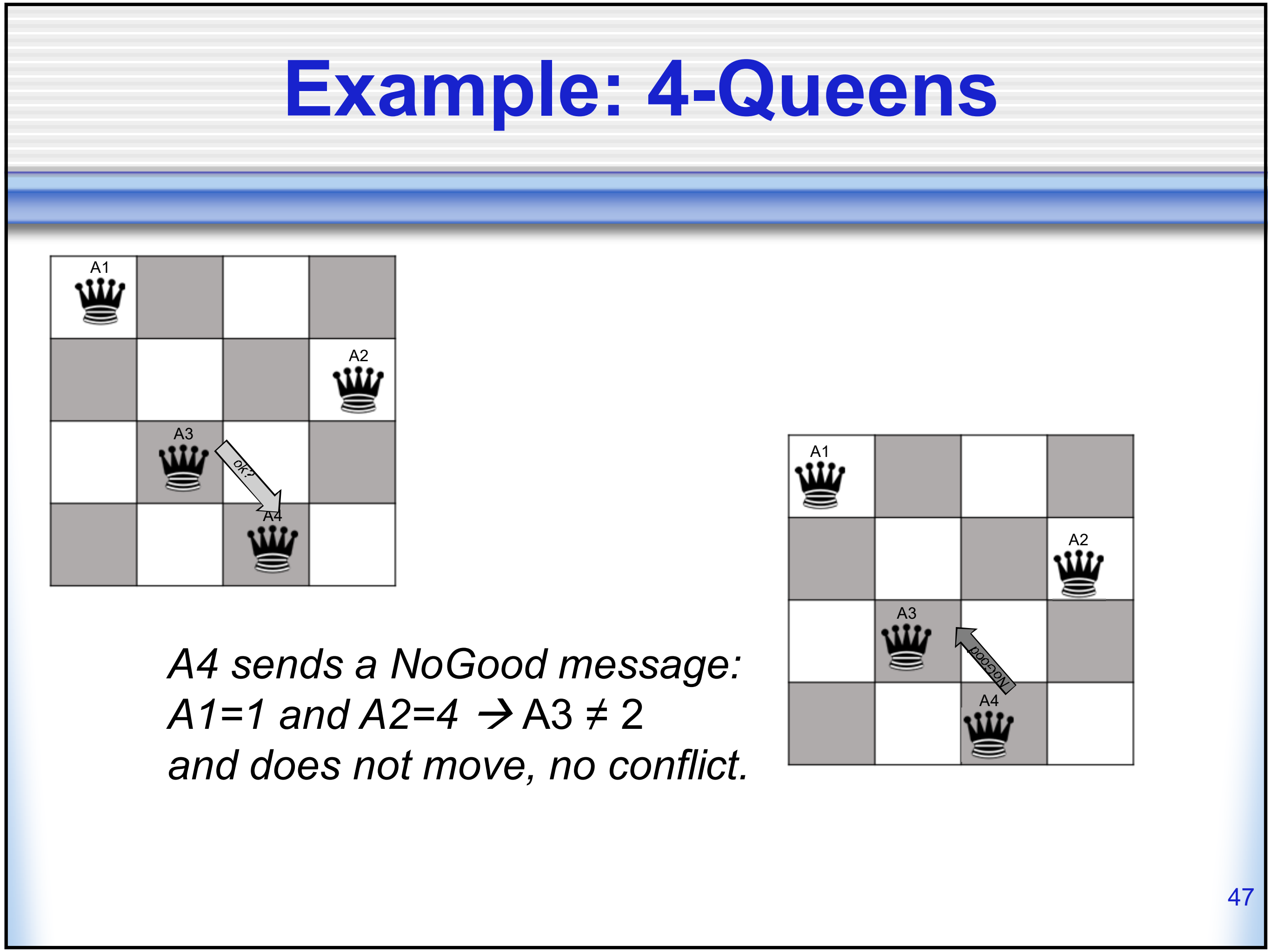
NoGood: A1=1 → A2 ≠ 3

# Example: 4-Queens





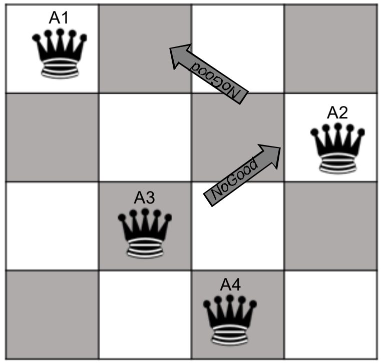
*A4 sends a NoGood message:*
*A1=1 and A2=4* → A3 ≠ 4 (no longer valid)
*and moves.*

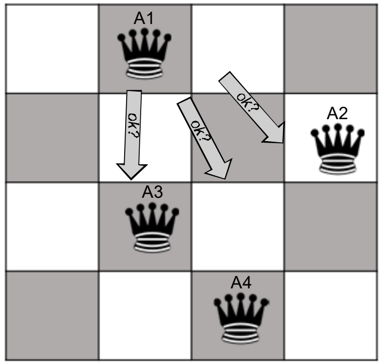


# Example: 4-Queens

*A4 sends a NoGood message:*
*A1=1 and A2=4* → A3 ≠ 2
*and does not move, no conflict.*

# Example: 4-Queens

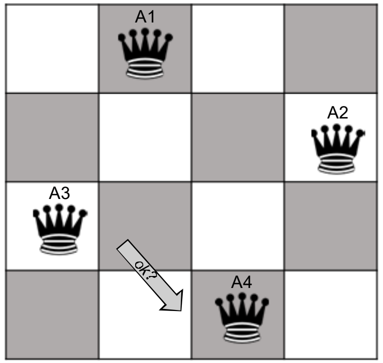




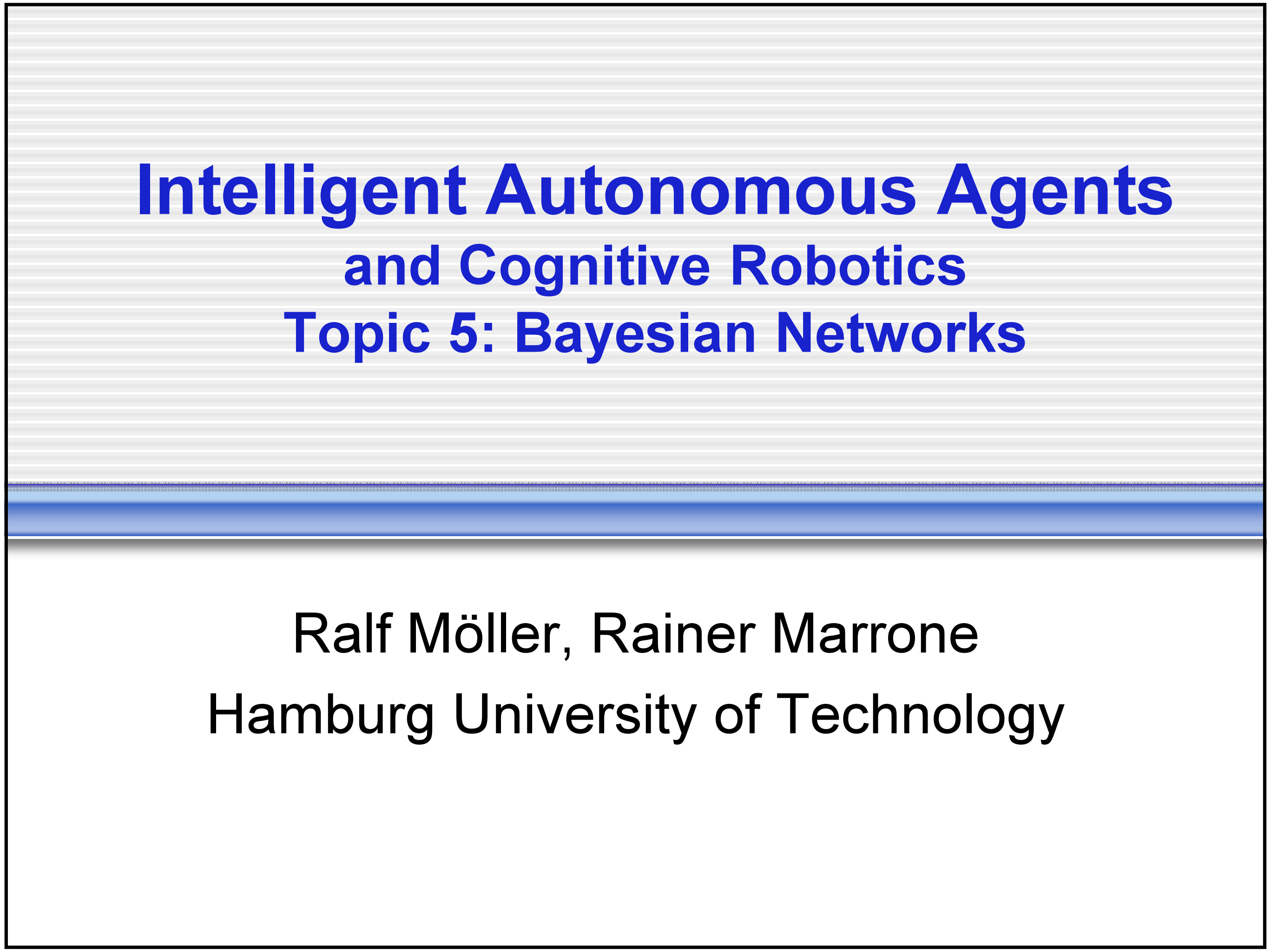
A3 has no option →NoGood: A1=1 → A2 ≠ 4,

A2 had a former NoGood message from A3 not to stay in 3 →send NoGood: A1 ≠ 1

# Example: 4-Queens

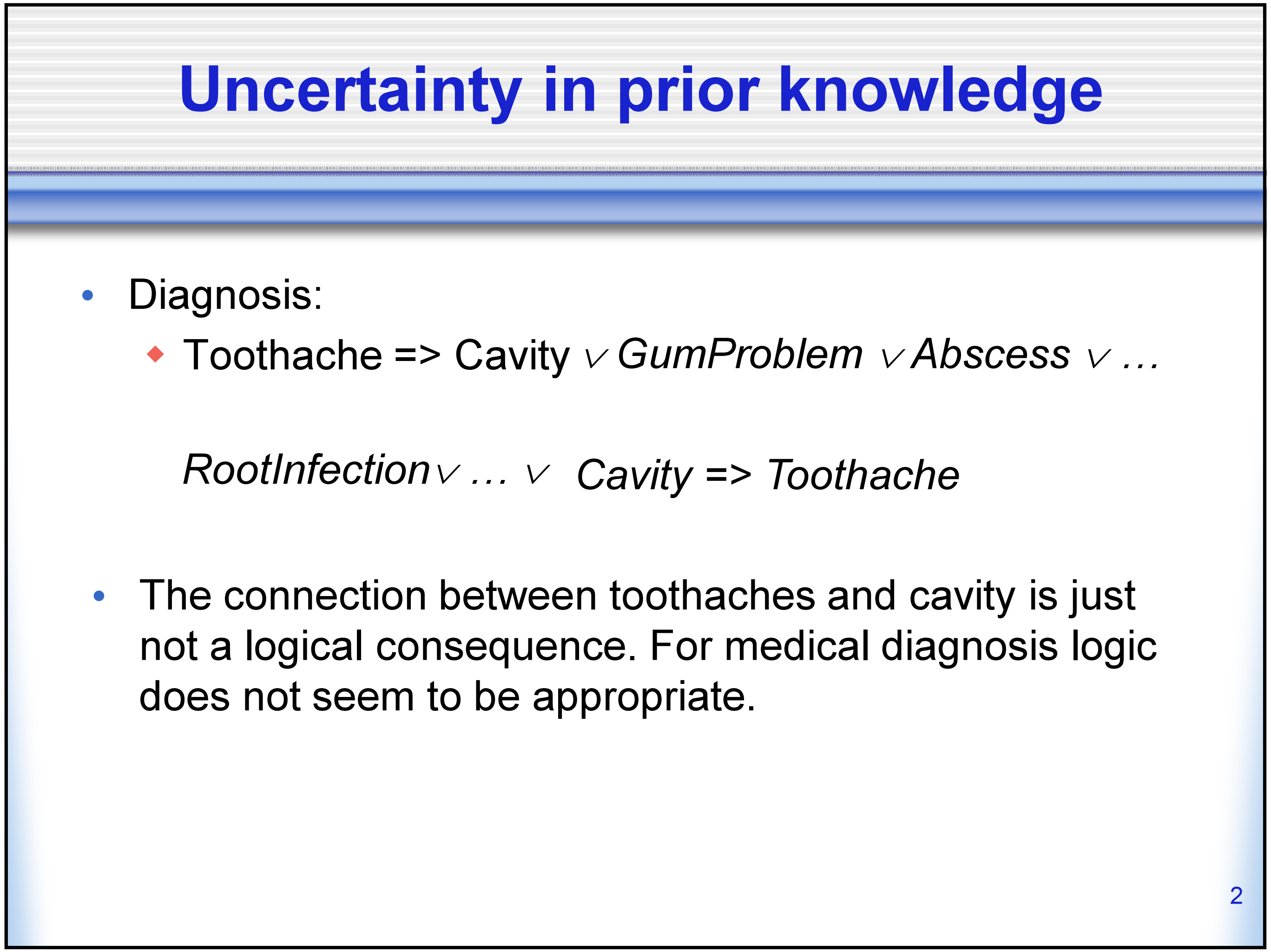


*No conflict for any queen → solved*



# Intelligent Autonomous Agents
## and Cognitive Robotics
## Topic 5: Bayesian Networks

Ralf Möller, Rainer Marrone
Hamburg University of Technology

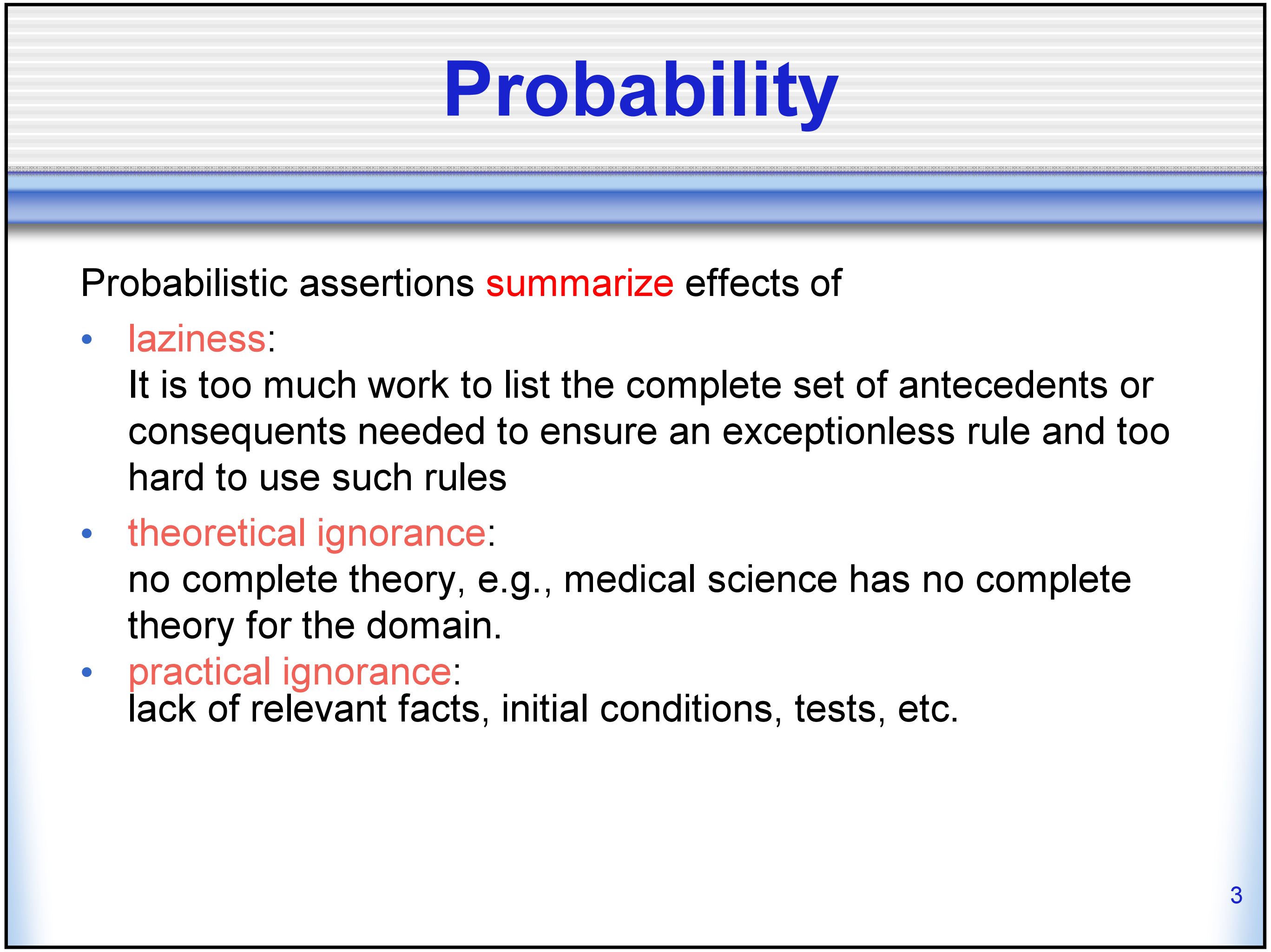


# Uncertainty in prior knowledge

- Diagnosis:
  - Toothache => Cavity ∨ *GumProblem* ∨ *Abscess* ∨ …

    *RootInfection* ∨ … ∨ *Cavity => Toothache*

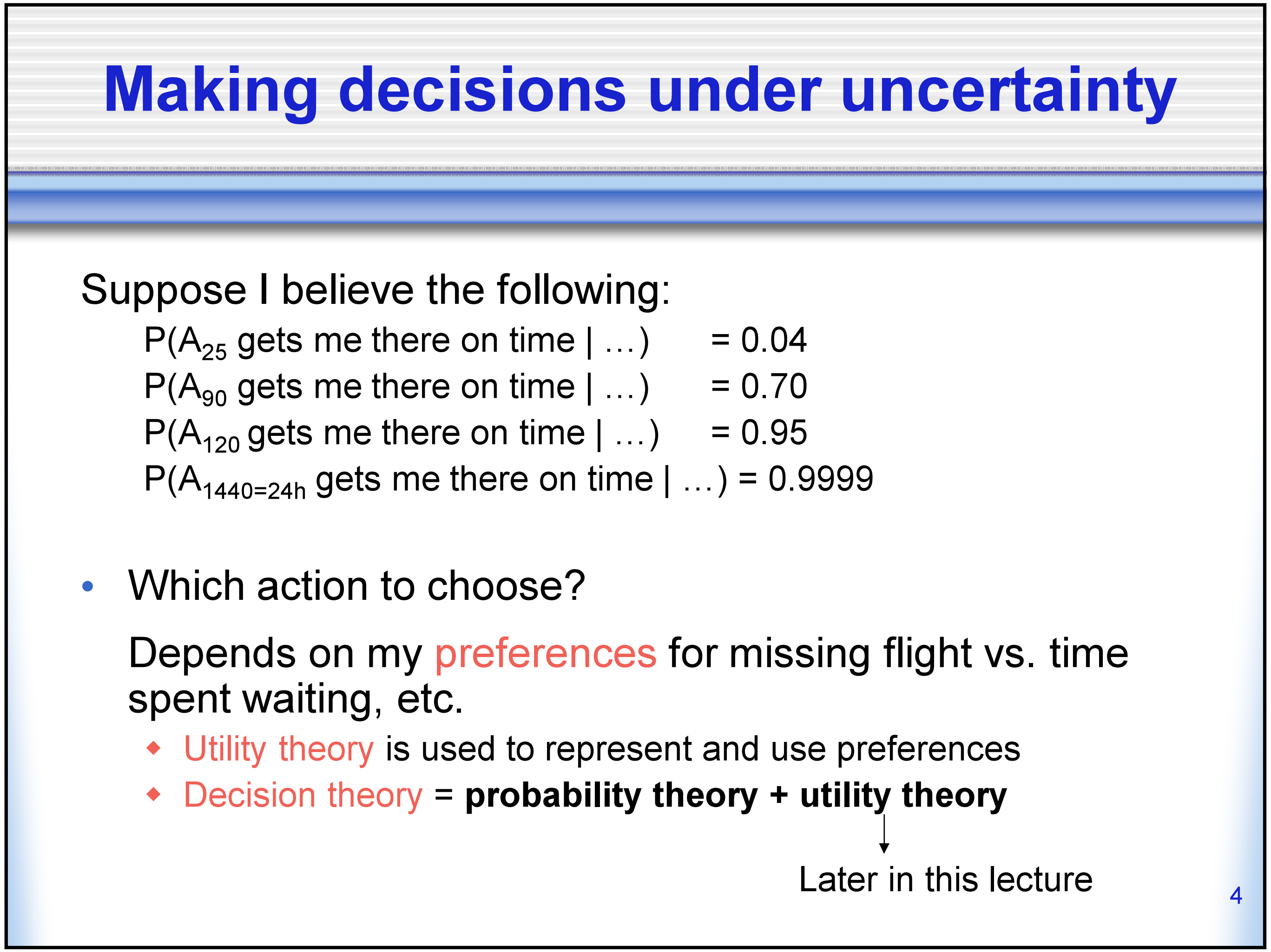
- The connection between toothaches and cavity is just not a logical consequence. For medical diagnosis logic does not seem to be appropriate.



# Probability

Probabilistic assertions summarize effects of

- laziness:
  It is too much work to list the complete set of antecedents or consequents needed to ensure an exceptionless rule and too hard to use such rules
- theoretical ignorance:
  no complete theory, e.g., medical science has no complete theory for the domain.
- practical ignorance:
  lack of relevant facts, initial conditions, tests, etc.



# Making decisions under uncertainty

Suppose I believe the following:

$P(A_{25}$ gets me there on time | …) = 0.04

$P(A_{90}$ gets me there on time | …) = 0.70

$P(A_{120}$ gets me there on time | …) = 0.95

$P(A_{1440=24h}$ gets me there on time | …) = 0.9999

- Which action to choose?

  Depends on my preferences for missing flight vs. time spent waiting, etc.

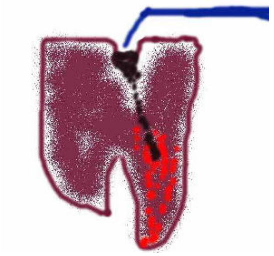
  - Utility theory is used to represent and use preferences
  - Decision theory = **probability theory + utility theory**

↓ Later in this lecture

# Example world

Example: *Dentist problem* with four variables:

- *Toothache* (I have a toothache)
- *Cavity* (I have a cavity)
- *Catch* (steel probe catches in my tooth)
- *Weather* (*sunny,rainy,cloudy,snow* )



# Prior probability

- Prior or unconditional probabilities of propositions

  e.g., P(*Cavity* = true) = 0.1 and P(*Weather* = sunny) = 0.72

  correspond to belief prior to arrival of any (new) evidence

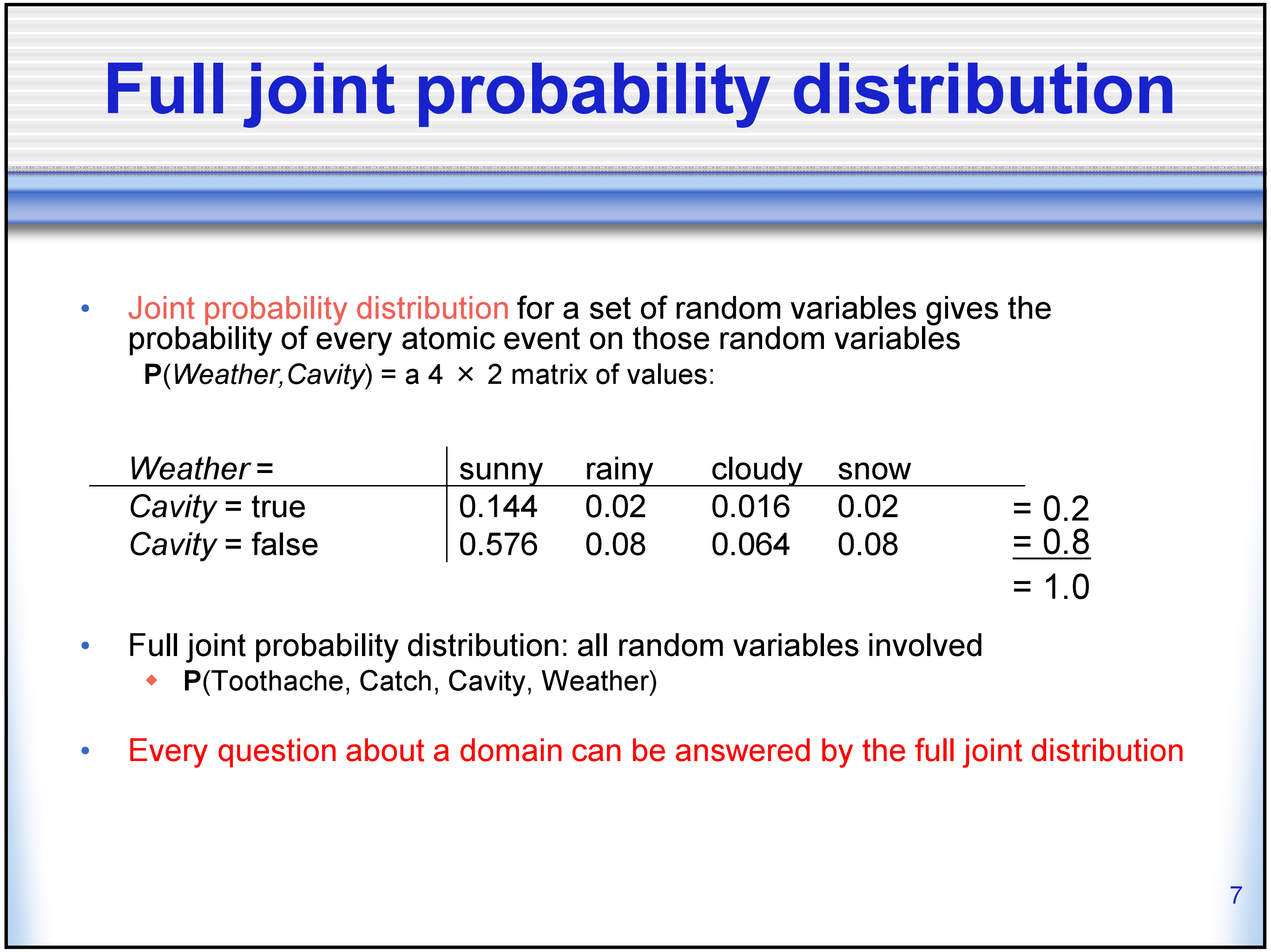
- Probability distribution

  gives values for all possible assignments

  (*sunny,rainy,cloudy,snow* ):

  **P**(*Weather*) = <0.72,0.1,0.08,0.1>

  (normalized, i.e., sums to 1 because one must be the case)



# Full joint probability distribution

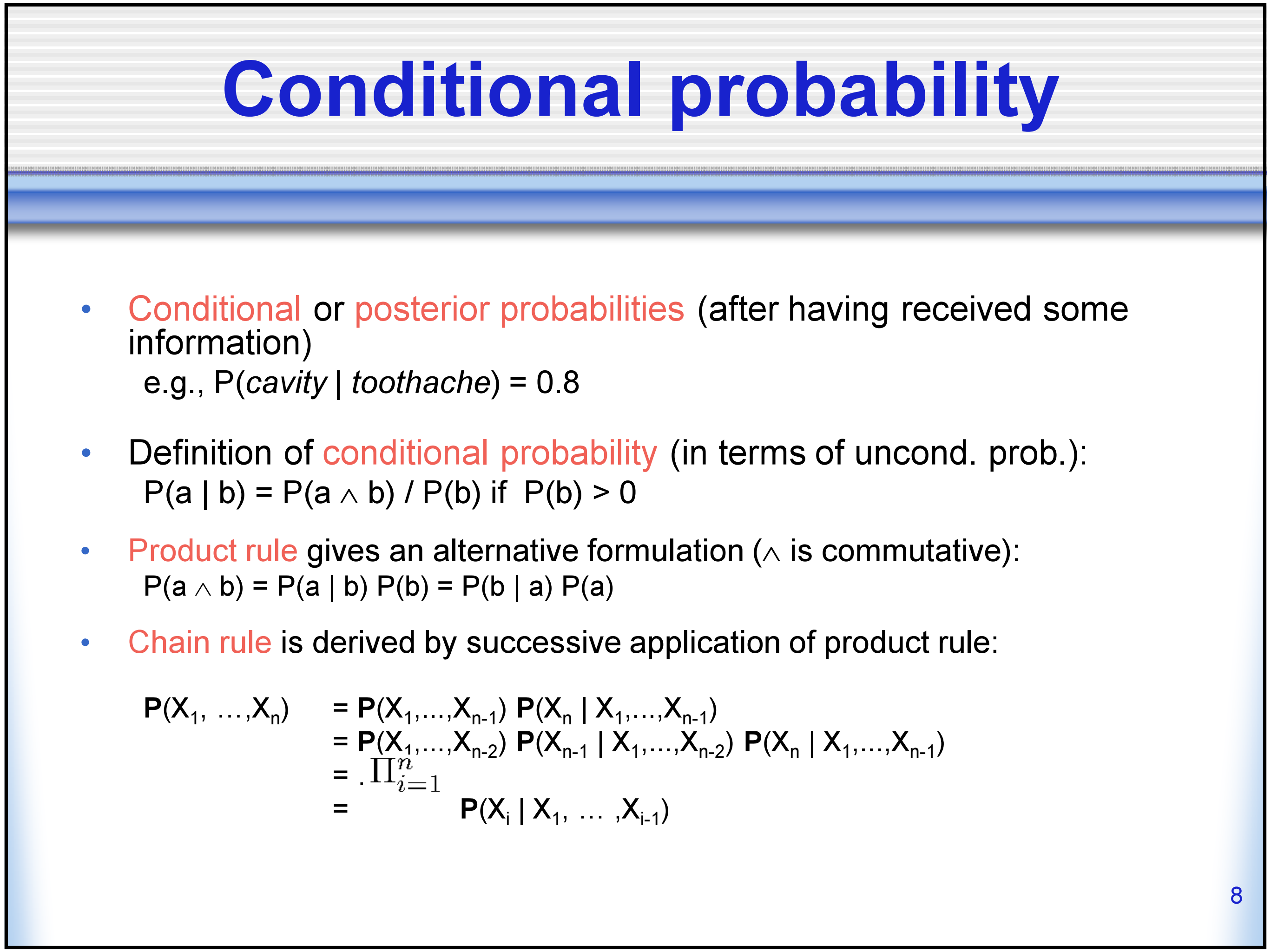
- Joint probability distribution for a set of random variables gives the probability of every atomic event on those random variables

  **P**(*Weather*, *Cavity*) = a 4 × 2 matrix of values:

| *Weather* = | sunny | rainy | cloudy | snow | |
|---|---|---|---|---|---|
| *Cavity* = true | 0.144 | 0.02 | 0.016 | 0.02 | = 0.2 |
| *Cavity* = false | 0.576 | 0.08 | 0.064 | 0.08 | = 0.8 |
| | | | | | = 1.0 |

- Full joint probability distribution: all random variables involved
  - **P**(Toothache, Catch, Cavity, Weather)

- Every question about a domain can be answered by the full joint distribution



# Conditional probability

- Conditional or posterior probabilities (after having received some information)
  e.g., P(*cavity* | *toothache*) = 0.8

- Definition of conditional probability (in terms of uncond. prob.):
  $P(a \mid b) = P(a \wedge b) / P(b)$ if $P(b) > 0$

- Product rule gives an alternative formulation ($\wedge$ is commutative):
  $P(a \wedge b) = P(a \mid b)\, P(b) = P(b \mid a)\, P(a)$

- Chain rule is derived by successive application of product rule:

$$
\begin{aligned}
\mathbf{P}(X_1, \ldots, X_n) &= \mathbf{P}(X_1,\ldots,X_{n-1})\ \mathbf{P}(X_n \mid X_1,\ldots,X_{n-1}) \\
&= \mathbf{P}(X_1,\ldots,X_{n-2})\ \mathbf{P}(X_{n-1} \mid X_1,\ldots,X_{n-2})\ \mathbf{P}(X_n \mid X_1,\ldots,X_{n-1}) \\
&= .\prod_{i=1}^{n} \\
&= \qquad \mathbf{P}(X_i \mid X_1, \ldots, X_{i-1})
\end{aligned}
$$

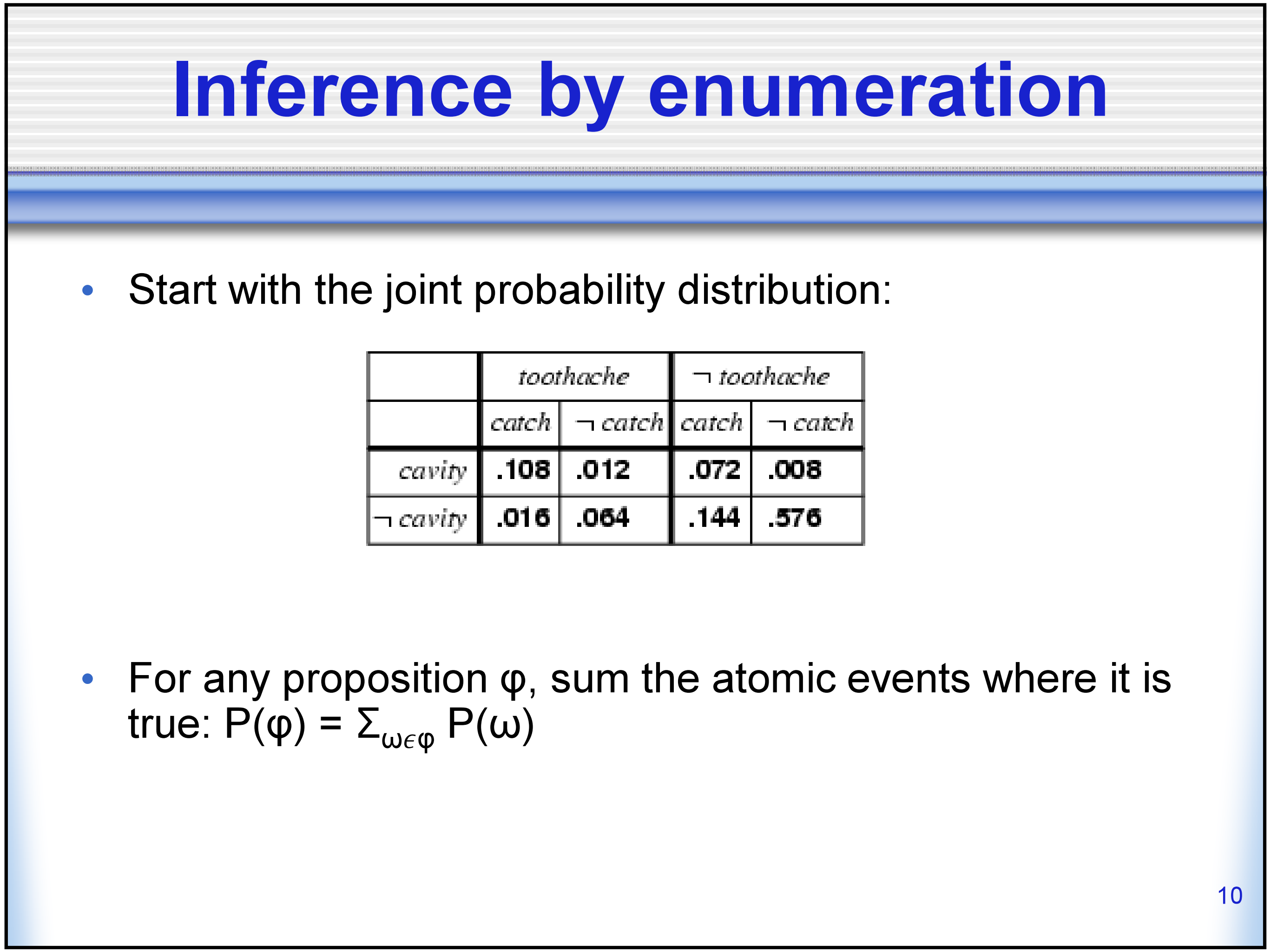
# Bayes rule

***Product rule***: $P(x,y) = P(x|y)P(y) = P(y|x)P(x)$

$$\Longrightarrow$$

$$P(cause|effect) = \frac{P(effect|cause) * P(cause)}{P(effect)} = \frac{likelihood * prior}{evidence}$$

$$P(X|Y) = \alpha\, P(Y|X)P(X)$$



# Inference by enumeration

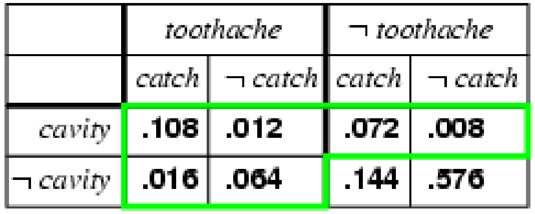
- Start with the joint probability distribution:

| | *toothache* | | *¬ toothache* | |
|---|---|---|---|---|
| | *catch* | *¬ catch* | *catch* | *¬ catch* |
| *cavity* | .108 | .012 | .072 | .008 |
| *¬ cavity* | .016 | .064 | .144 | .576 |

- For any proposition φ, sum the atomic events where it is true: $P(\varphi) = \Sigma_{\omega\epsilon\varphi} P(\omega)$

# Inference by enumeration

- Start with the joint probability distribution:



| | *toothache* | | ¬ *toothache* | |
|---|---|---|---|---|
| | *catch* | ¬ *catch* | *catch* | ¬ *catch* |
| *cavity* | .108 | .012 | .072 | .008 |
| ¬ *cavity* | .016 | .064 | .144 | .576 |

P(cavity ∨ *toothache*) =
0.108 + 0.012 + 0.072 + 0.008+ 0.016 + 0.064 = 0.28

# Inference by enumeration

- Start with the joint probability distribution:

| | *toothache* | | ¬ *toothache* | |
|---|---|---|---|---|
| | *catch* | ¬ *catch* | *catch* | ¬ *catch* |
| *cavity* | .108 | .012 | .072 | .008 |
| ¬ *cavity* | .016 | .064 | .144 | .576 |

- Can also compute conditional probabilities:

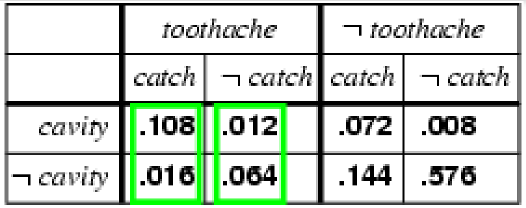
$$P(\neg \mathit{cavity} \mid \mathit{toothache}) = \frac{P(\neg \mathit{cavity} \wedge \mathit{toothache})}{P(\mathit{toothache})}$$

Product rule

$$= \frac{0.016+0.064}{0.108 + 0.012 + 0.016 + 0.064}$$

$$= 0.4$$

# Normalization



| | *toothache* | | *¬ toothache* | |
|---|---|---|---|---|
| | *catch* | *¬ catch* | *catch* | *¬ catch* |
| *cavity* | .108 | .012 | .072 | .008 |
| *¬ cavity* | .016 | .064 | .144 | .576 |

- Denominator **P**(**z**) (or P(toothache) in the example before) can be viewed as a normalization constant α

**P**(*Cavity* | *toothache*) = **P**(Cavity,toothache)/P(toothache)
= α **P**(*Cavity,toothache*)
= α [**P**(*Cavity,toothache,catch*) + **P**(*Cavity,toothache,¬ catch*)]
= α [<0.108,0.016> + <0.012,0.064>]
= α <0.12,0.08> = <0.6,0.4>

*α*(0,12+0,08)=1*
*α=1/0,2=5*
*5*0,12=0,6*
*5*0,08=0,4*

General idea: compute distribution on query variable by fixing evidence variables (toothache) and summing over hidden variables (Catch)

# General inference procedure

Typically, we are interested in

the posterior joint distribution of the query variables **Y**
given specific values **e** for the evidence variables **E**
**X** are all variables of the modeled world

Let the hidden variables be **H** = **X** - **Y** – **E** then the required summation of joint entries is done by summing out the hidden variables:

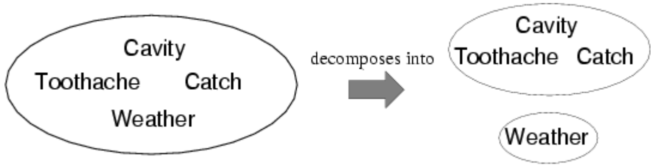
$$\mathbf{P}(\mathbf{Y} \mid \mathbf{E} = \mathbf{e}) = \alpha\mathbf{P}(\mathbf{Y},\mathbf{E} = \mathbf{e}) = \alpha\Sigma_h\mathbf{P}(\mathbf{Y},\mathbf{E}= \mathbf{e}, \mathbf{H} = \mathbf{h})$$

- The terms in the summation are joint entries because **Y**, **E** and **H** together exhaust the set of random variables (**X**)

- Obvious problems:
  1. Space complexity $O(d^n)$ to store the joint distribution where $d$ is the largest arity and n denotes the number of random variables
  2. Worst-case time complexity $O(d^n)$
  3. How to find the numbers for $O(d^n)$ entries?

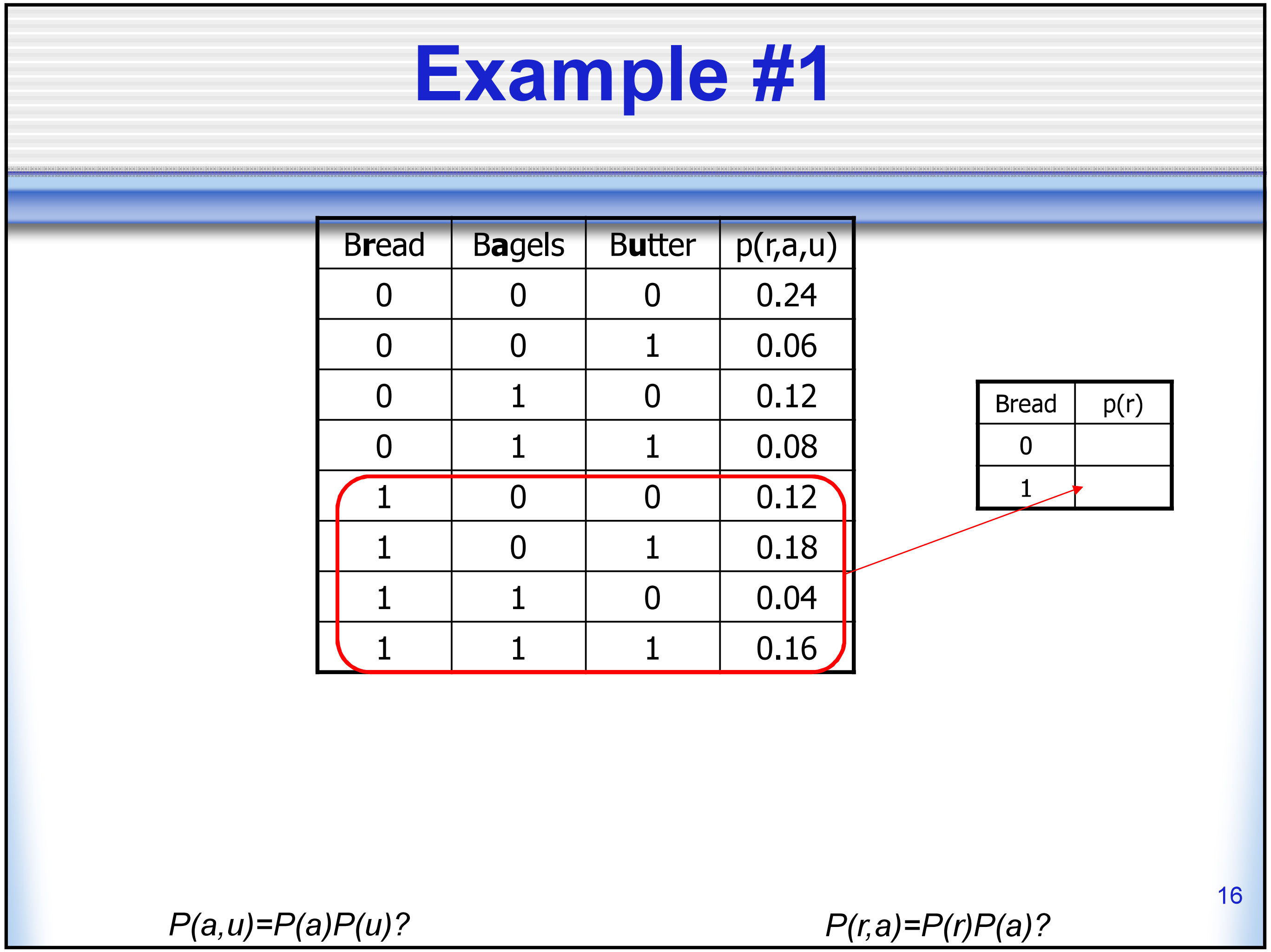
# Independence

- *A* and *B* are independent iff
  **P**(*A*|*B*) = **P**(*A*) or **P**(*B*|*A*) = **P**(*B*) or **P**(A, B) = **P**(*A*) **P**(*B*)



**P**(*Toothache, Catch, Cavity, Weather*)
= **P**(*Toothache, Catch, Cavity*) **P**(*Weather*)

- 32 entries table can be constructed from 8 and 4 entries;
- Absolute independence powerful but rare
- How can we check whether we have independent variables in the full joint?



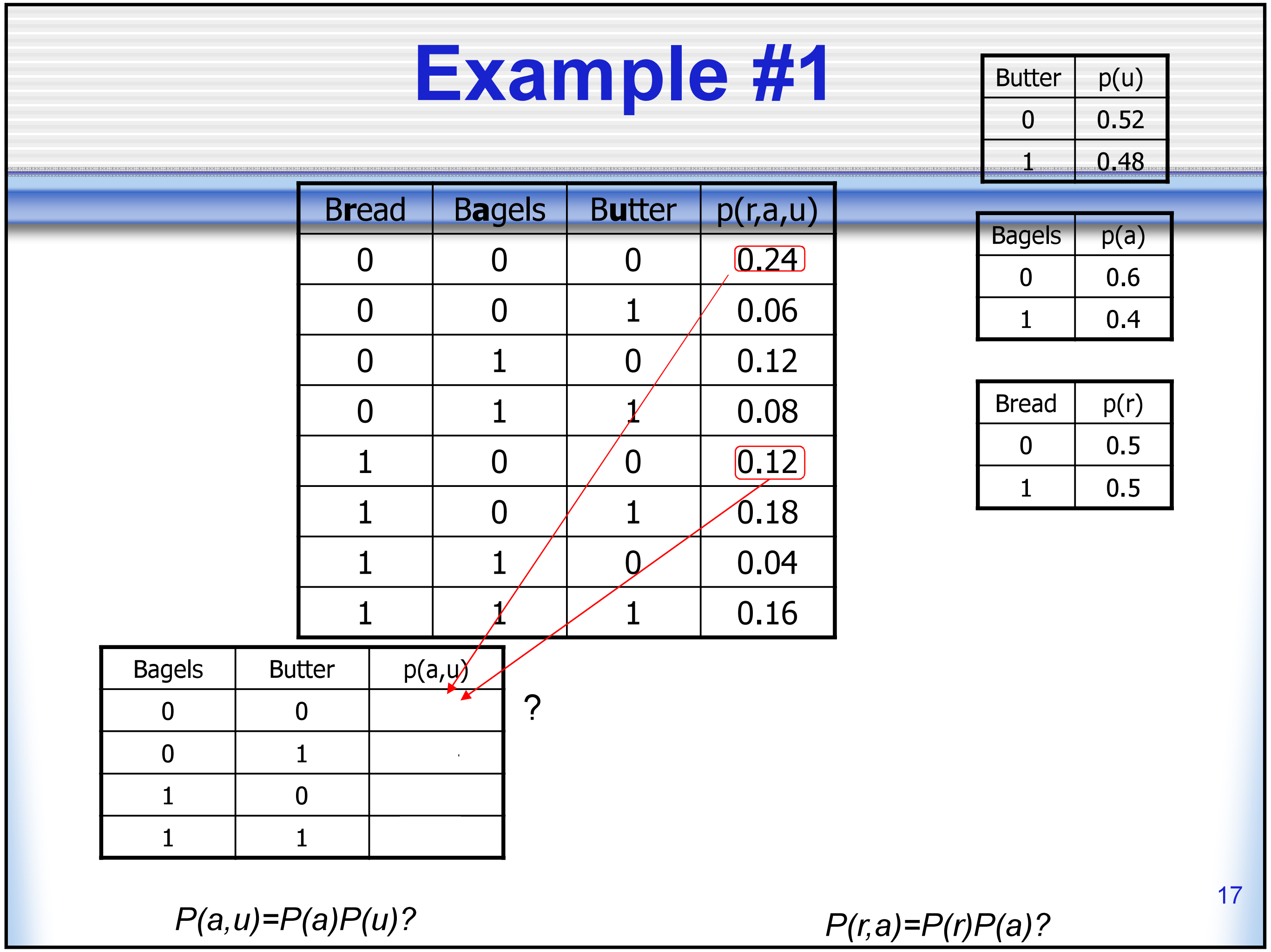
# Example #1

| B**r**ead | B**a**gels | B**u**tter | p(r,a,u) |
|---|---|---|---|
| 0 | 0 | 0 | 0.24 |
| 0 | 0 | 1 | 0.06 |
| 0 | 1 | 0 | 0.12 |
| 0 | 1 | 1 | 0.08 |
| 1 | 0 | 0 | 0.12 |
| 1 | 0 | 1 | 0.18 |
| 1 | 1 | 0 | 0.04 |
| 1 | 1 | 1 | 0.16 |

| Bread | p(r) |
|---|---|
| 0 | |
| 1 | |

*P(a,u)=P(a)P(u)?*

*P(r,a)=P(r)P(a)?*



# Example #1

| Butter | p(u) |
|---|---|
| 0 | 0.52 |
| 1 | 0.48 |

| Bread | Bagels | Butter | p(r,a,u) |
|---|---|---|---|
| 0 | 0 | 0 | 0.24 |
| 0 | 0 | 1 | 0.06 |
| 0 | 1 | 0 | 0.12 |
| 0 | 1 | 1 | 0.08 |
| 1 | 0 | 0 | 0.12 |
| 1 | 0 | 1 | 0.18 |
| 1 | 1 | 0 | 0.04 |
| 1 | 1 | 1 | 0.16 |

| Bagels | p(a) |
|---|---|
| 0 | 0.6 |
| 1 | 0.4 |

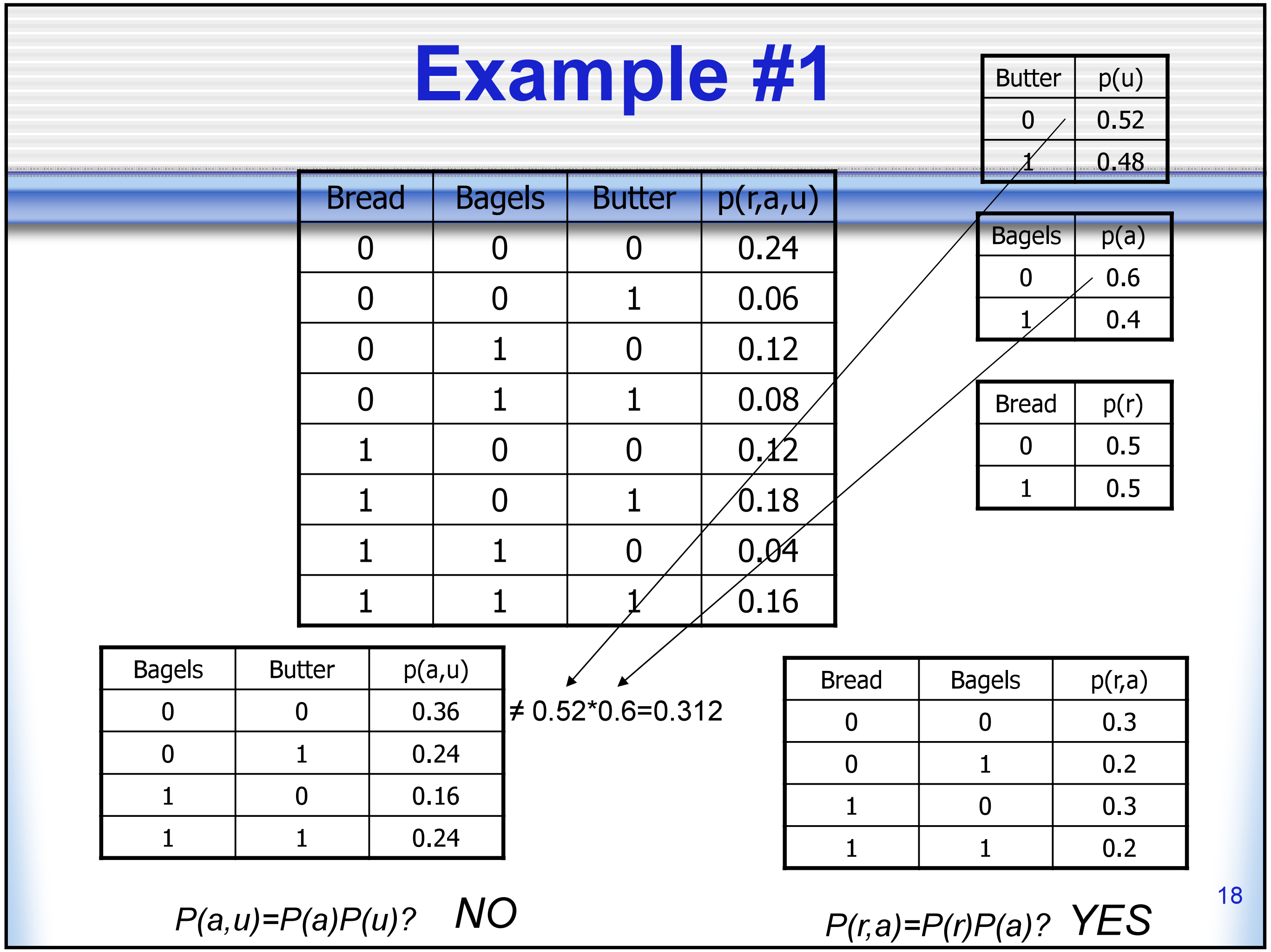
| Bread | p(r) |
|---|---|
| 0 | 0.5 |
| 1 | 0.5 |

| Bagels | Butter | p(a,u) |
|---|---|---|
| 0 | 0 | |
| 0 | 1 | |
| 1 | 0 | |
| 1 | 1 | |

?

$P(a,u)=P(a)P(u)?$

$P(r,a)=P(r)P(a)?$



# Example #1

| Bread | Bagels | Butter | p(r,a,u) |
|---|---|---|---|
| 0 | 0 | 0 | 0.24 |
| 0 | 0 | 1 | 0.06 |
| 0 | 1 | 0 | 0.12 |
| 0 | 1 | 1 | 0.08 |
| 1 | 0 | 0 | 0.12 |
| 1 | 0 | 1 | 0.18 |
| 1 | 1 | 0 | 0.04 |
| 1 | 1 | 1 | 0.16 |

| Butter | p(u) |
|---|---|
| 0 | 0.52 |
| 1 | 0.48 |

| Bagels | p(a) |
|---|---|
| 0 | 0.6 |
| 1 | 0.4 |

| Bread | p(r) |
|---|---|
| 0 | 0.5 |
| 1 | 0.5 |

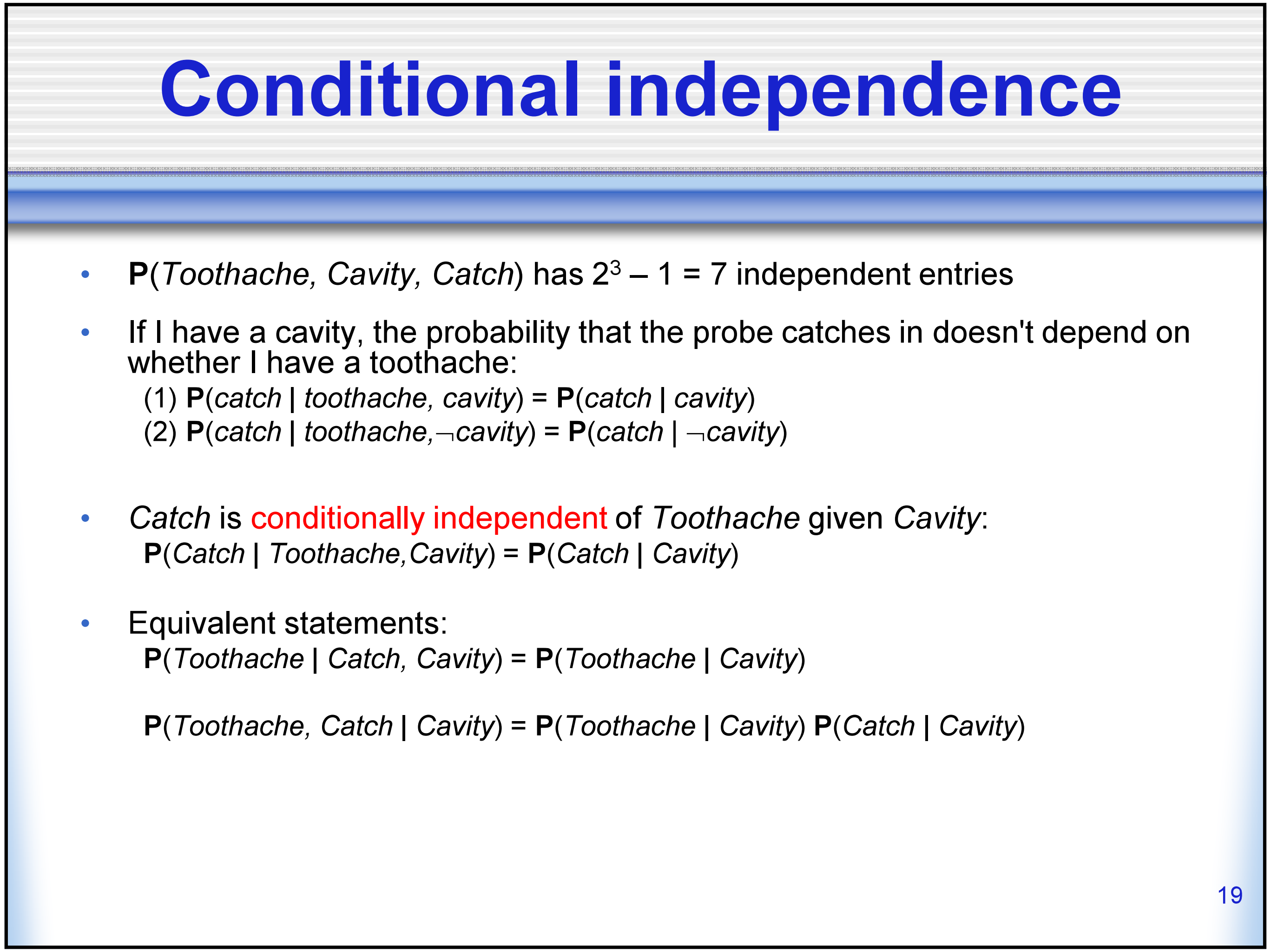
| Bagels | Butter | p(a,u) |
|---|---|---|
| 0 | 0 | 0.36 |
| 0 | 1 | 0.24 |
| 1 | 0 | 0.16 |
| 1 | 1 | 0.24 |

≠ 0.52*0.6=0.312

| Bread | Bagels | p(r,a) |
|---|---|---|
| 0 | 0 | 0.3 |
| 0 | 1 | 0.2 |
| 1 | 0 | 0.3 |
| 1 | 1 | 0.2 |

*P(a,u)=P(a)P(u)?* *NO*

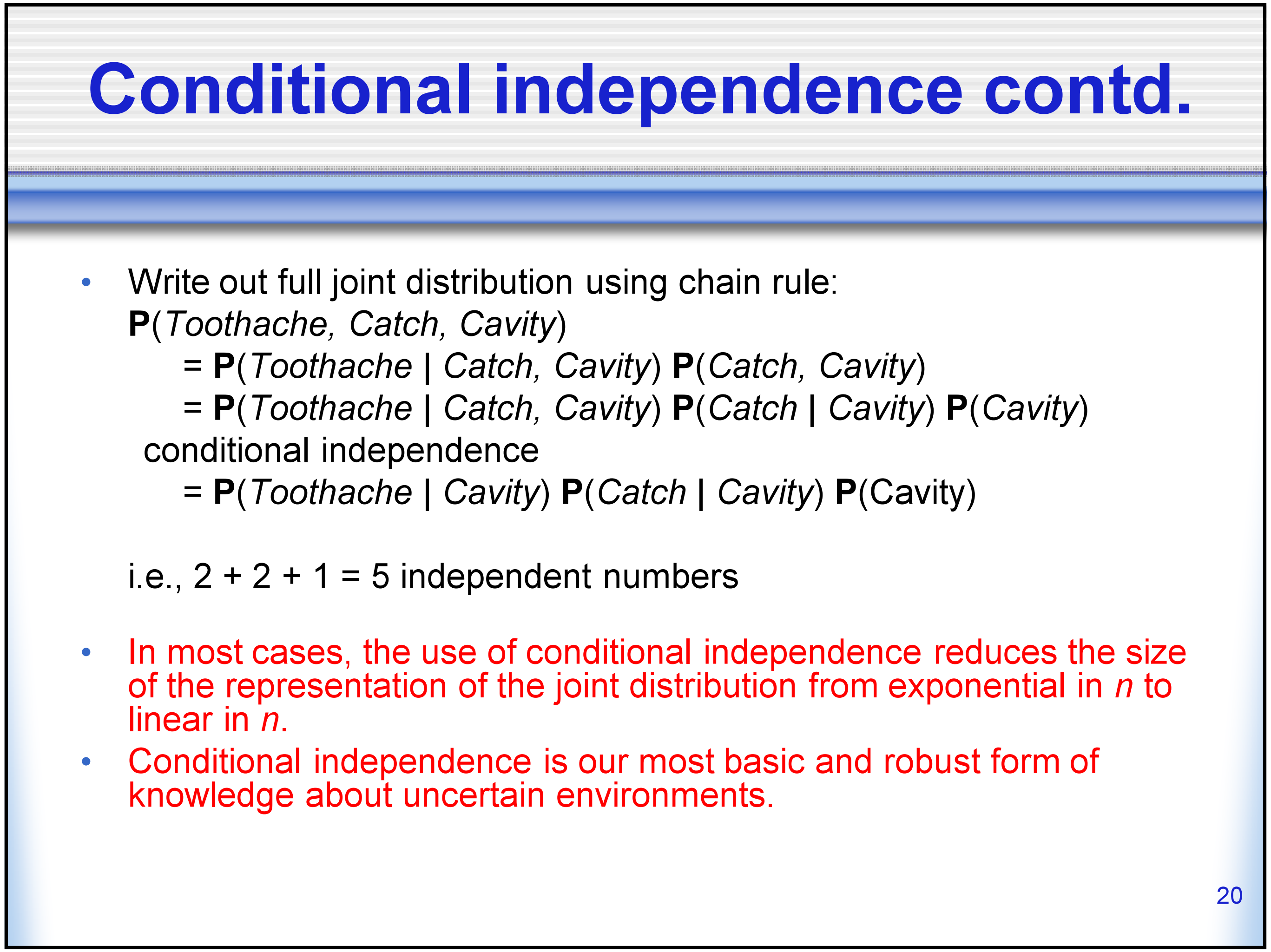
*P(r,a)=P(r)P(a)?* *YES*



# Conditional independence

- $\mathbf{P}(\mathit{Toothache, Cavity, Catch})$ has $2^3 - 1 = 7$ independent entries
- If I have a cavity, the probability that the probe catches in doesn't depend on whether I have a toothache:
  (1) $\mathbf{P}(\mathit{catch} \mid \mathit{toothache, cavity}) = \mathbf{P}(\mathit{catch} \mid \mathit{cavity})$
  (2) $\mathbf{P}(\mathit{catch} \mid \mathit{toothache}, \neg \mathit{cavity}) = \mathbf{P}(\mathit{catch} \mid \neg \mathit{cavity})$
- *Catch* is conditionally independent of *Toothache* given *Cavity*:
  $\mathbf{P}(\mathit{Catch} \mid \mathit{Toothache,Cavity}) = \mathbf{P}(\mathit{Catch} \mid \mathit{Cavity})$
- Equivalent statements:
  $\mathbf{P}(\mathit{Toothache} \mid \mathit{Catch, Cavity}) = \mathbf{P}(\mathit{Toothache} \mid \mathit{Cavity})$

  $\mathbf{P}(\mathit{Toothache, Catch} \mid \mathit{Cavity}) = \mathbf{P}(\mathit{Toothache} \mid \mathit{Cavity})\ \mathbf{P}(\mathit{Catch} \mid \mathit{Cavity})$

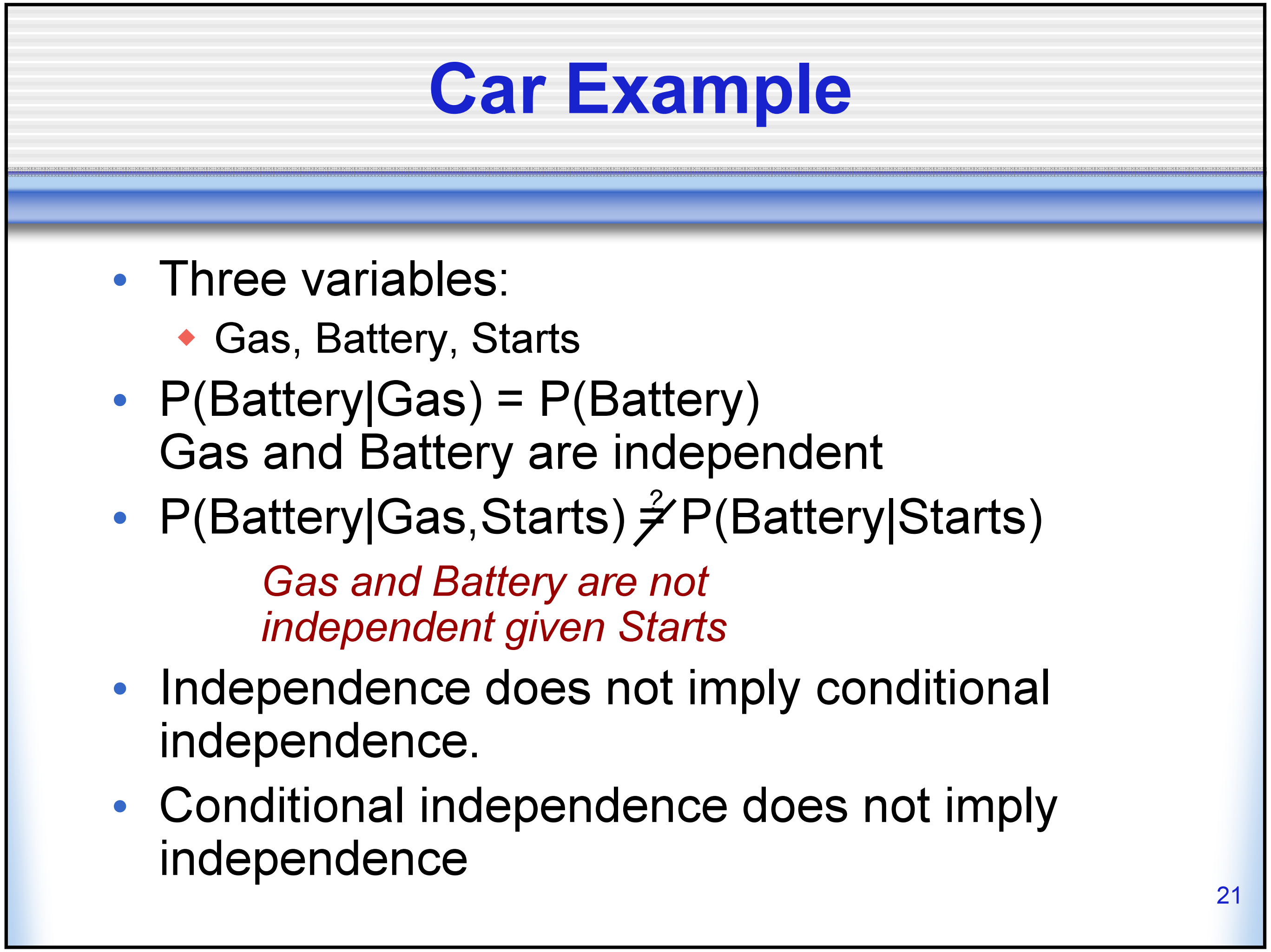


# Conditional independence contd.

- Write out full joint distribution using chain rule:
  **P**(*Toothache, Catch, Cavity*)
  = **P**(*Toothache* | *Catch, Cavity*) **P**(*Catch, Cavity*)
  = **P**(*Toothache* | *Catch, Cavity*) **P**(*Catch* | *Cavity*) **P**(*Cavity*)
  conditional independence
  = **P**(*Toothache* | *Cavity*) **P**(*Catch* | *Cavity*) **P**(Cavity)

  i.e., 2 + 2 + 1 = 5 independent numbers

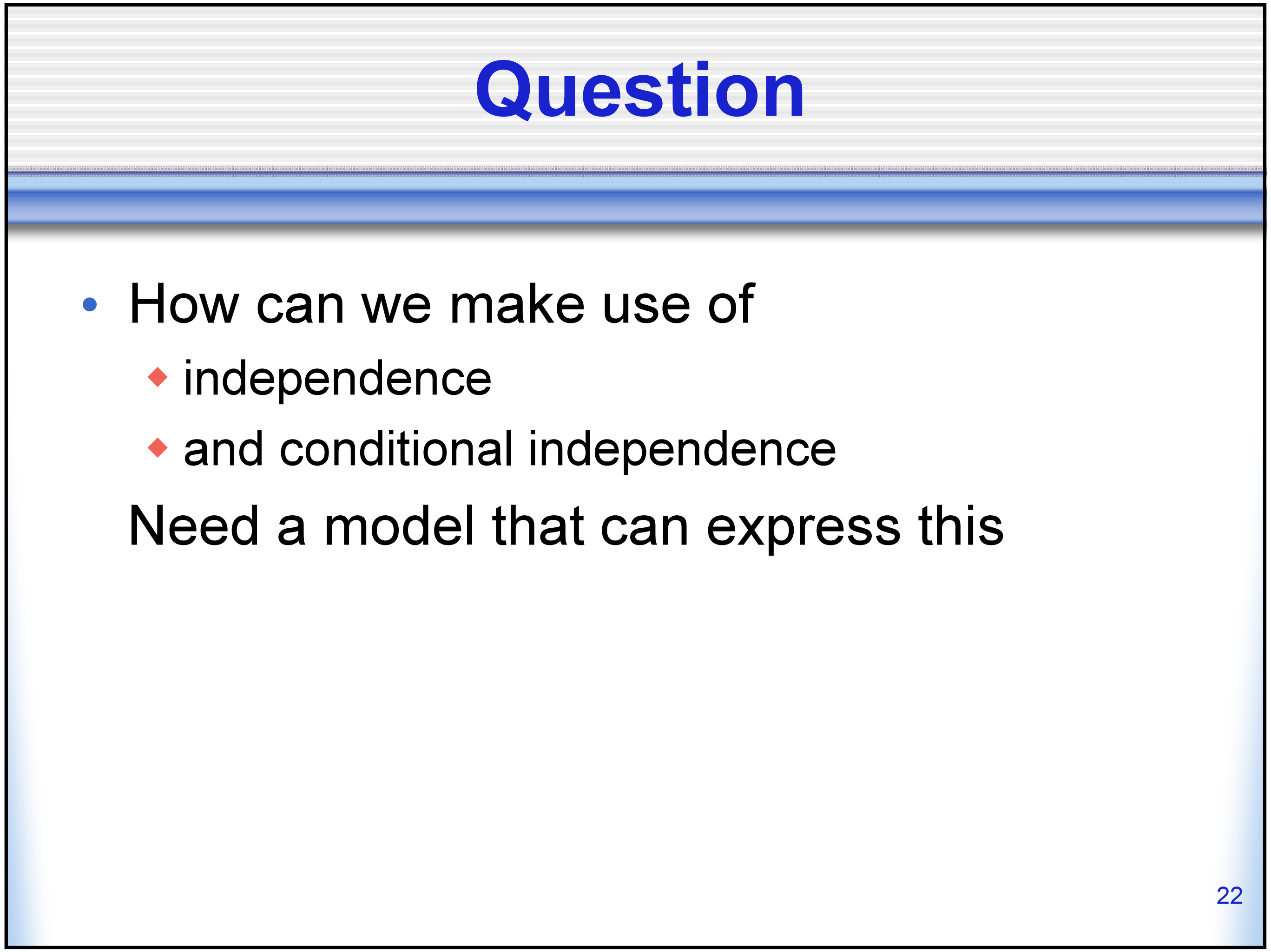
- In most cases, the use of conditional independence reduces the size of the representation of the joint distribution from exponential in *n* to linear in *n*.
- Conditional independence is our most basic and robust form of knowledge about uncertain environments.



# Car Example

- Three variables:
  - Gas, Battery, Starts
- P(Battery|Gas) = P(Battery)
  Gas and Battery are independent
- $P(\text{Battery}|\text{Gas},\text{Starts}) \overset{?}{\neq} P(\text{Battery}|\text{Starts})$

  *Gas and Battery are not independent given Starts*
- Independence does not imply conditional independence.
- Conditional independence does not imply independence



# Question

- How can we make use of
  - independence
  - and conditional independence

  Need a model that can express this

# Bayesian networks

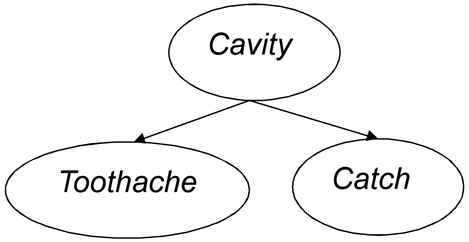
- A simple, graphical notation for **conditional independence assertions** and hence for compact specification of the full joint distributions

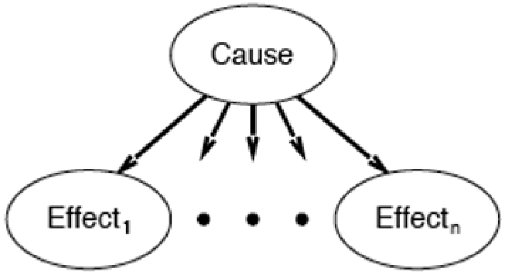
- Syntax:
  - a set of nodes, one per variable
  - a directed, acyclic graph (link ≈ "directly influences")
  - a conditional distribution for each node given its parents:

$$\mathbf{P}\ (X_i \mid \text{Parents}\ (X_i))$$

- In the simplest case, conditional distribution represented as a conditional probability table (CPT) giving the distribution over $X_i$ for each combination of parent values

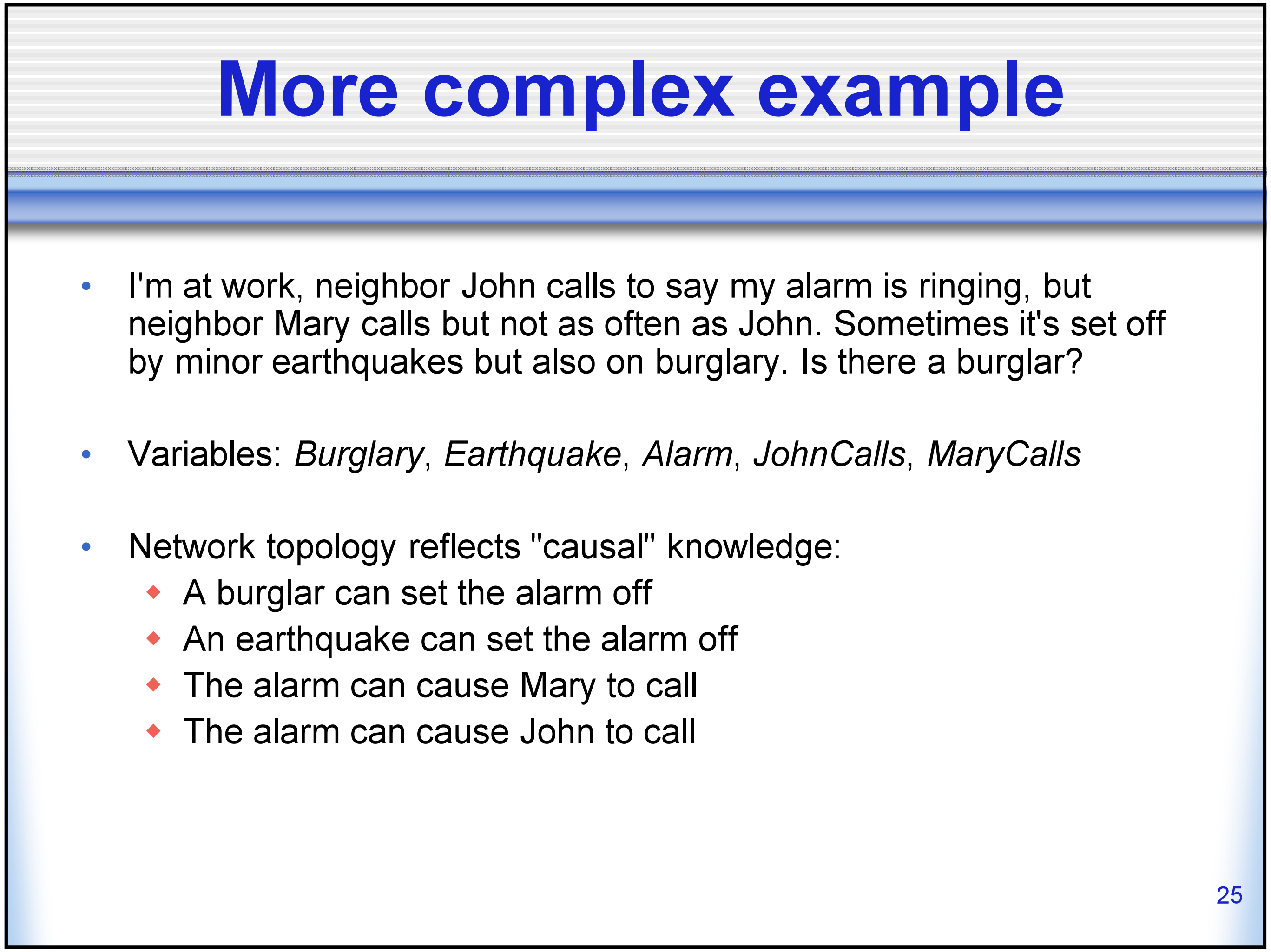
# Simplest Bayesian Network





$$P(Cause|Effect_1, Effect_2, \ldots) = \alpha P(Cause) \prod_i P(Effect_i|Cause)$$

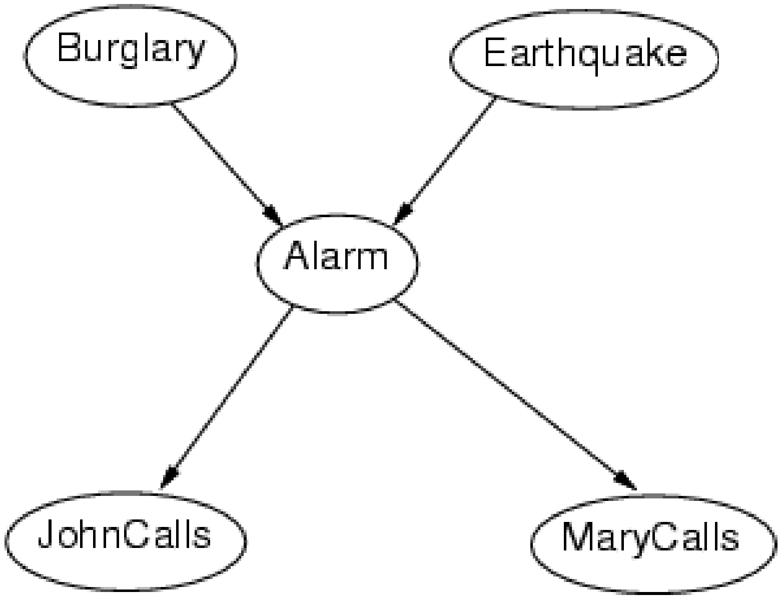
- also called Naïve Bayesian networks
- conditional independence of all effect variables



# More complex example

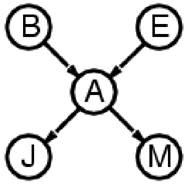
- I'm at work, neighbor John calls to say my alarm is ringing, but neighbor Mary calls but not as often as John. Sometimes it's set off by minor earthquakes but also on burglary. Is there a burglar?
- Variables: *Burglary*, *Earthquake*, *Alarm*, *JohnCalls*, *MaryCalls*
- Network topology reflects "causal" knowledge:
  - A burglar can set the alarm off
  - An earthquake can set the alarm off
  - The alarm can cause Mary to call
  - The alarm can cause John to call

# Example contd.



# Compactness

- A CPT for Boolean $X_i$ with $k$ Boolean parents has $2^k$ rows for the combinations of parent values



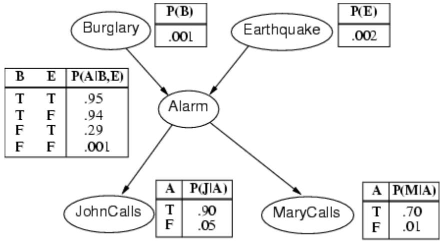
- Each row requires one number $p$ for $X_i$ = *true* (the number for $X_i$ = *false* is just *1-p*)
- If each of $n$ Boolean variables has no more than $k$ parents, the complete network requires $O(n \cdot 2^k)$ numbers
  i.e., grows linearly with $n$, vs. $O(2^n)$ for the full joint distribution
- For burglary net? 1 + 1 + 4 + 2 + 2 = 10 numbers (vs. $2^5$-1 = 31)

**k** parents with **n** values each and **m** values for the child node of the parents?

Number of indepenent values = $n^k \cdot (m-1)$

# Semantics

The full joint distribution can be rewritten using the *chain rule*:



$$P(X_1, \ldots, X_n) = \prod_{i=1}^{n} P(X_i | X_1, \ldots, X_{i-1})$$

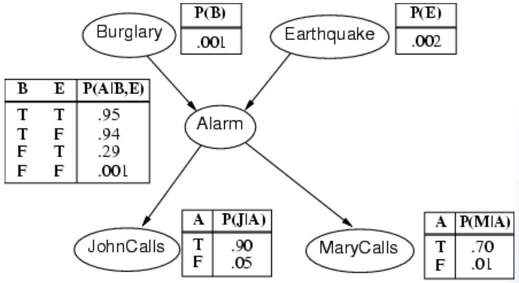
$$P(X_1, \ldots, X_n) = \prod_{i=1}^{n} P(X_i | \, parent(X_i))$$

Assumption: Independence and Conditional independence assertions are correctly modeled

# Semantics

The full joint distribution is defined as the product of the local conditional distributions:

$$P(X_1, \ldots, X_n) = \prod_{i=1}^{n} P(X_i | \, parent(X_i))$$



| P(B) |
|---|
| .001 |

| P(E) |
|---|
| .002 |

| B | E | P(A\|B,E) |
|---|---|---|
| T | T | .95 |
| T | F | .94 |
| F | T | .29 |
| F | F | .001 |

| A | P(J\|A) |
|---|---|
| T | .90 |
| F | .05 |

| A | P(M\|A) |
|---|---|
| T | .70 |
| F | .01 |

e.g., $\mathbf{P}(j \wedge m \wedge a \wedge \neg b \wedge \neg e)$

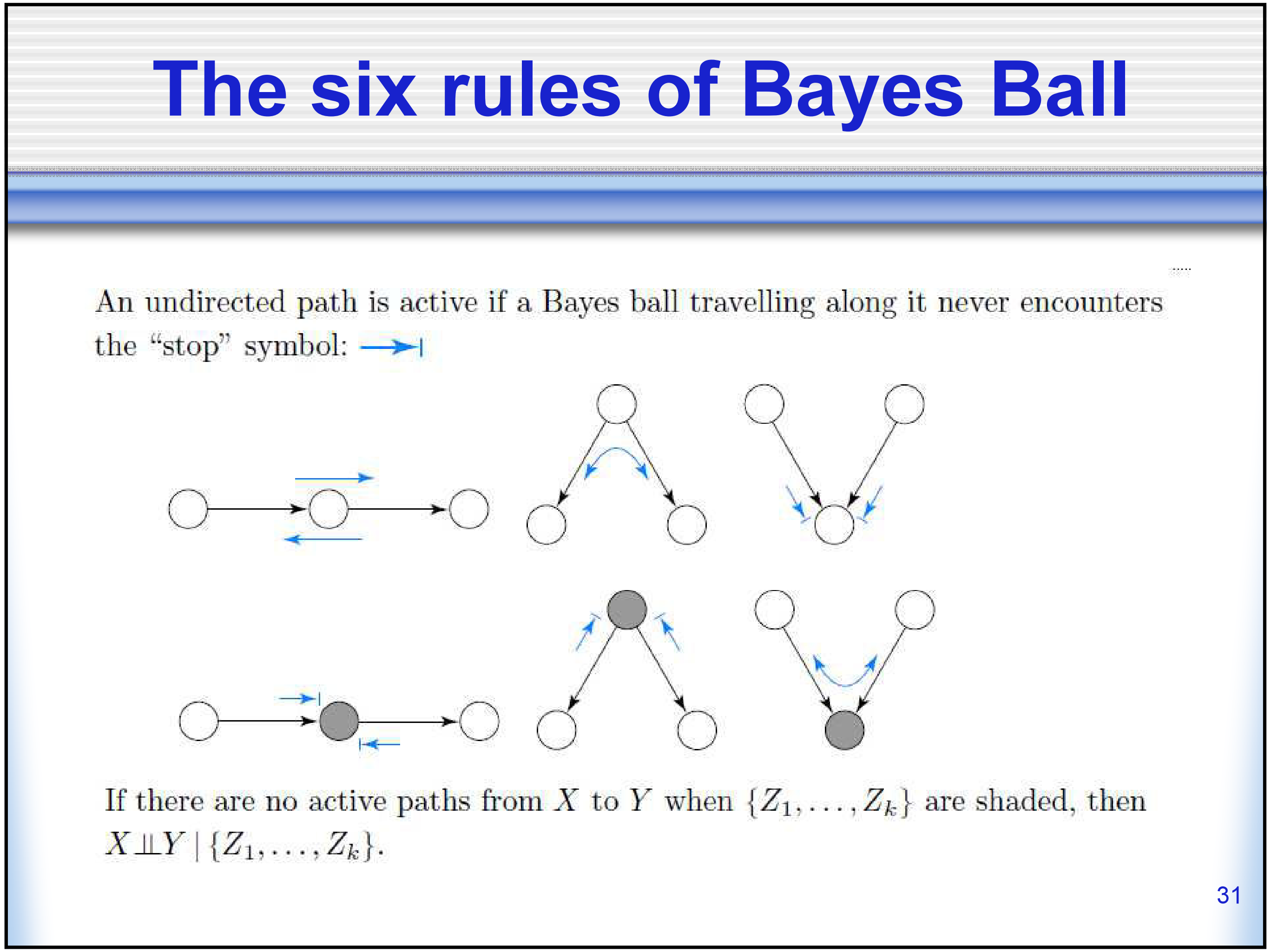
$= \mathbf{P}(j \mid a)\, \mathbf{P}(m \mid a)\, \mathbf{P}(a \mid \neg b, \neg e)\, \mathbf{P}(\neg b)\, \mathbf{P}(\neg e)$

$= 0.90 \text{x} 0.7 \text{x} 0.001 \text{x} 0.999 \text{x} 0.998$

$\approx 0.00063$

# Encoding conditional independence via d-separation

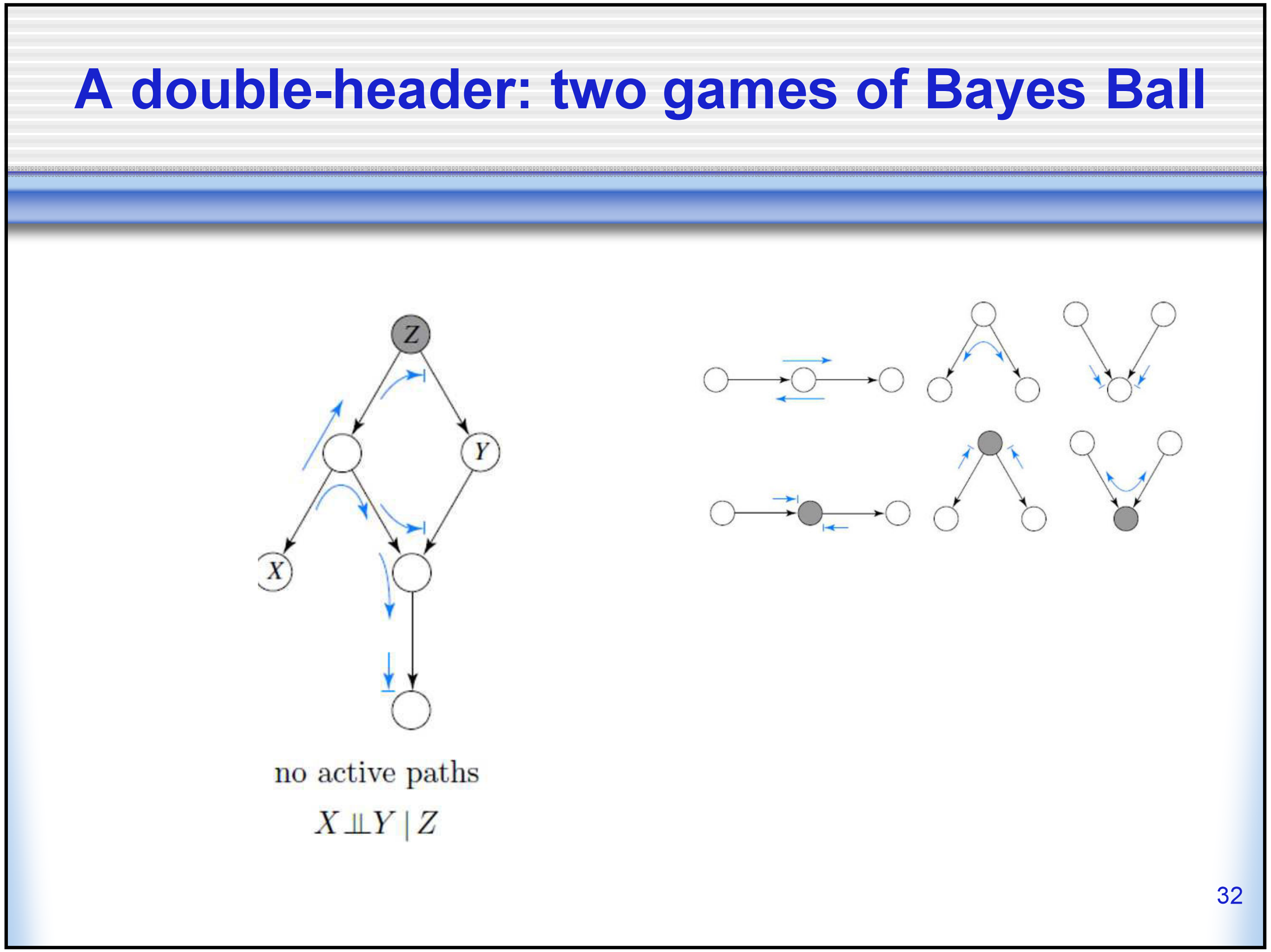
- We can determine if  conditional independence holds by a graph separation criterion called ***d-separation*** *(direction dependent separation)*
- X and Y are ***d-separated*** if there is no active path between them.
- The formals definition of ***active*** is somewhat involved. The Bayes Ball Algorithm gives a nice graphical definition.



# The six rules of Bayes Ball

An undirected path is active if a Bayes ball travelling along it never encounters the “stop” symbol: $\longrightarrow\!\!|$

If there are no active paths from $X$ to $Y$ when $\{Z_1, \ldots, Z_k\}$ are shaded, then $X \perp\!\!\!\perp Y \mid \{Z_1, \ldots, Z_k\}$.



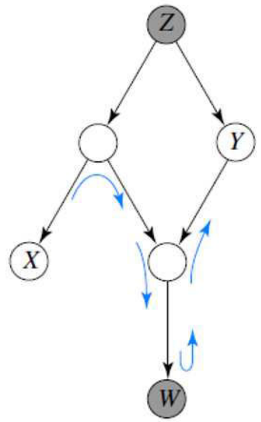
# A double-header: two games of Bayes Ball

no active paths

$X \perp\!\!\!\perp Y \mid Z$

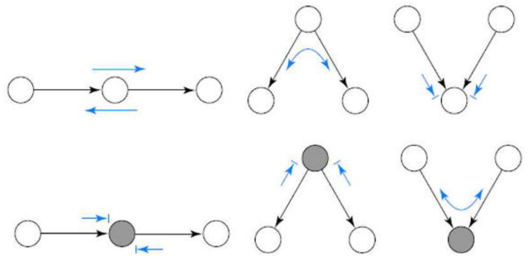
32

# A double-header: two games of Bayes Ball



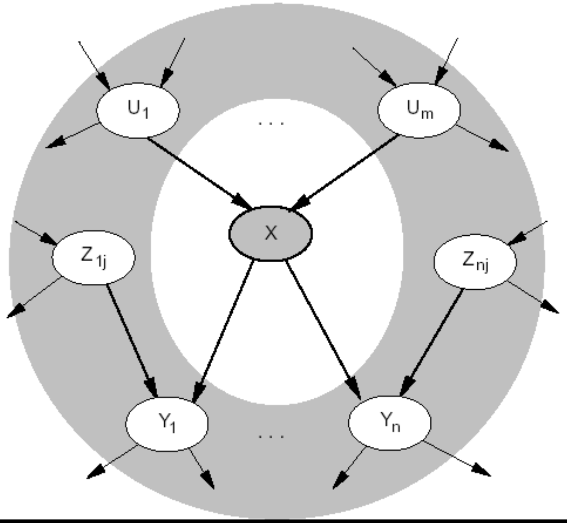
one active path

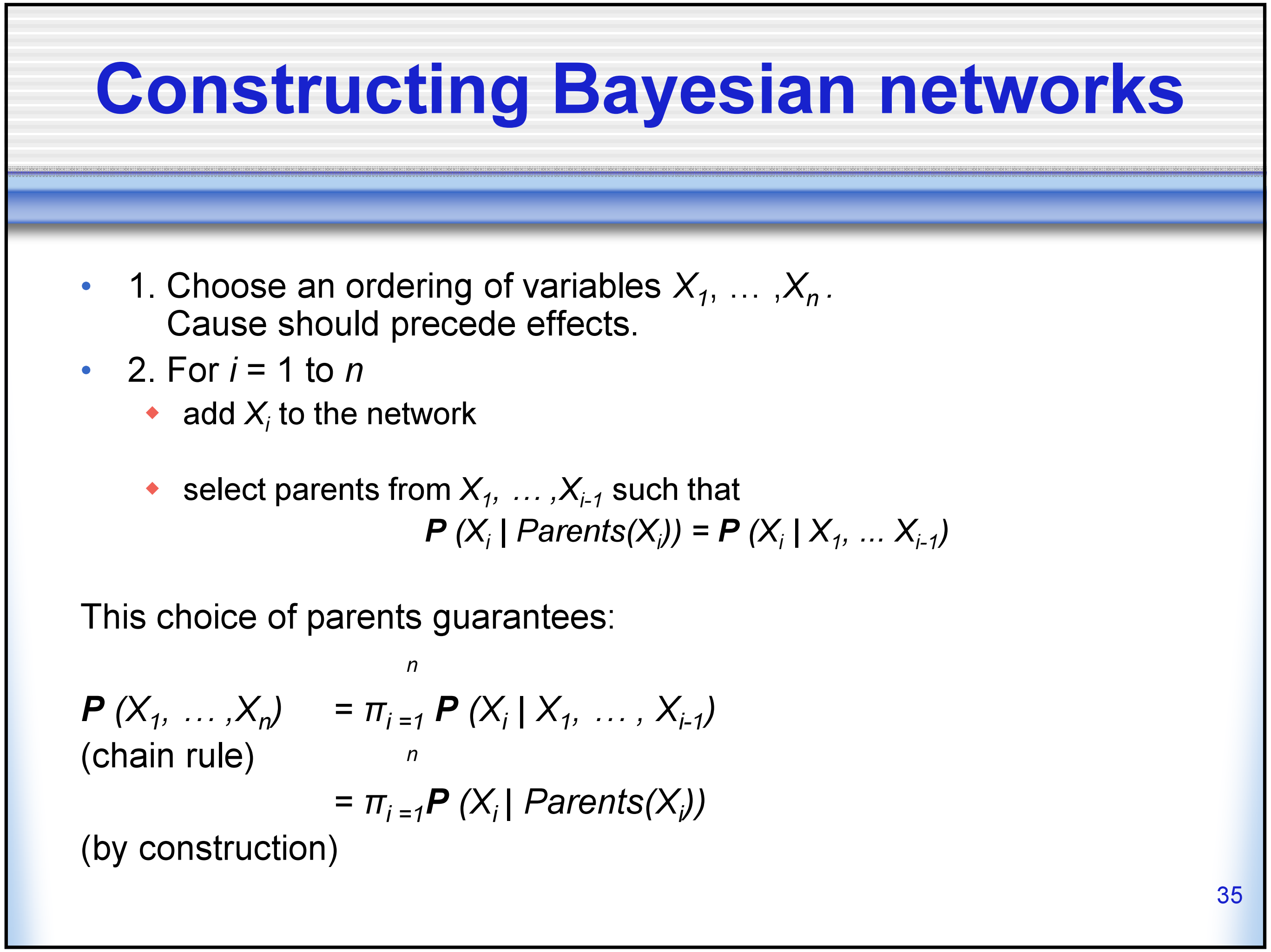
$X \not\perp\!\!\!\perp Y \mid \{W, Z\}$



# Markov Blanket

- Markov blanket: Parents + children + children's parents
- Node is conditionally independent of all other nodes in network, given its Markov Blanket -> simplifies computation -> gather information on the nodes of the Markov Blanket?





# Constructing Bayesian networks

- 1. Choose an ordering of variables $X_1, \dots, X_n$.
  Cause should precede effects.
- 2. For $i = 1$ to $n$
  - add $X_i$ to the network
  - select parents from $X_1, \dots, X_{i-1}$ such that
    $\mathbf{P}(X_i \mid Parents(X_i)) = \mathbf{P}(X_i \mid X_1, \dots X_{i-1})$

This choice of parents guarantees:

$$\mathbf{P}(X_1, \dots, X_n) = \pi_{i=1}^{n} \mathbf{P}(X_i \mid X_1, \dots, X_{i-1})$$

(chain rule)

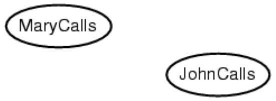
$$= \pi_{i=1}^{n} \mathbf{P}(X_i \mid Parents(X_i))$$

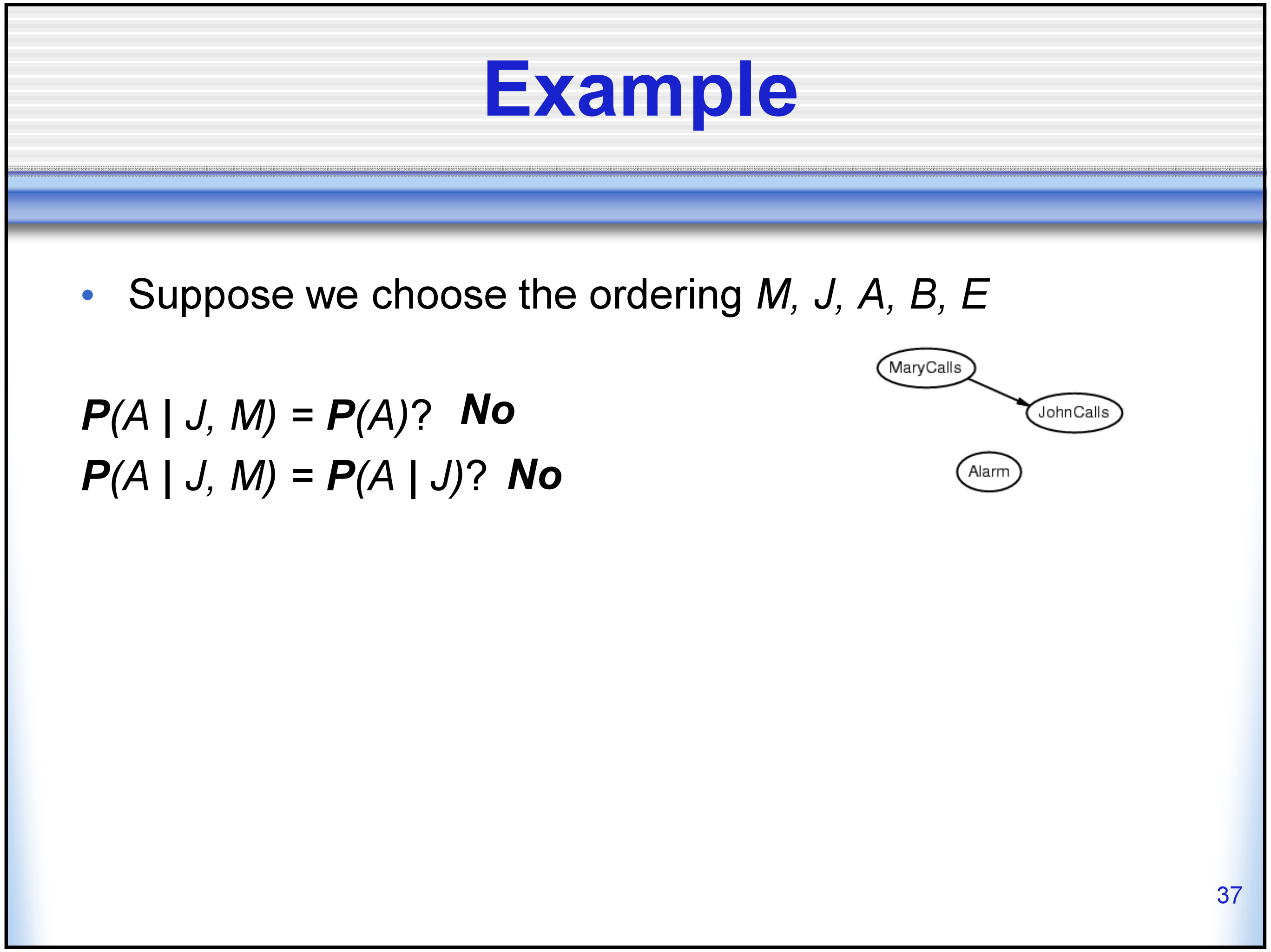
(by construction)

# Example

- Suppose we choose the ordering *M, J, A, B, E*

***P***(*J* | *M*) = ***P***(*J*)? **No**





# Example

- Suppose we choose the ordering *M, J, A, B, E*

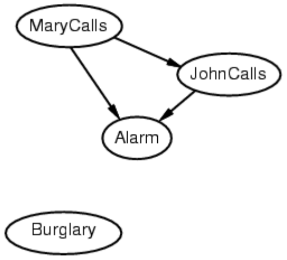
$\mathbf{P}(A \mid J, M) = \mathbf{P}(A)$? **No**

$\mathbf{P}(A \mid J, M) = \mathbf{P}(A \mid J)$? **No**

MaryCalls
JohnCalls
Alarm

# Example

- Suppose we choose the ordering *M, J, A, B, E*

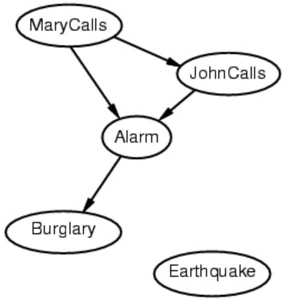


***P***(*B* | *A, J, M*) = ***P***(*B*)? ***No***

***P***(*B* | *A, J, M*) = ***P***(*B* | *A*)? ***Yes***

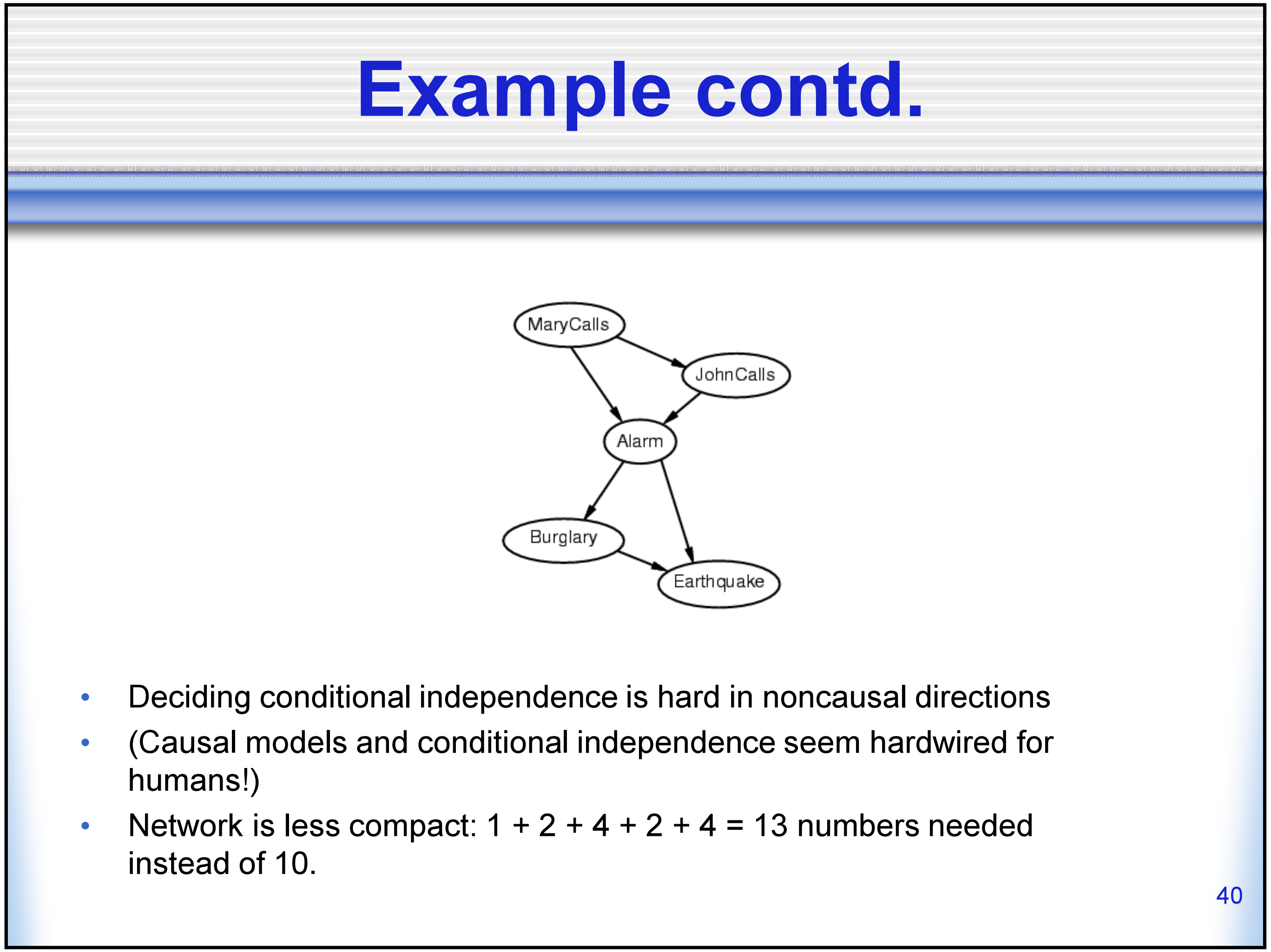
# Example

- Suppose we choose the ordering M, J, A, B, E



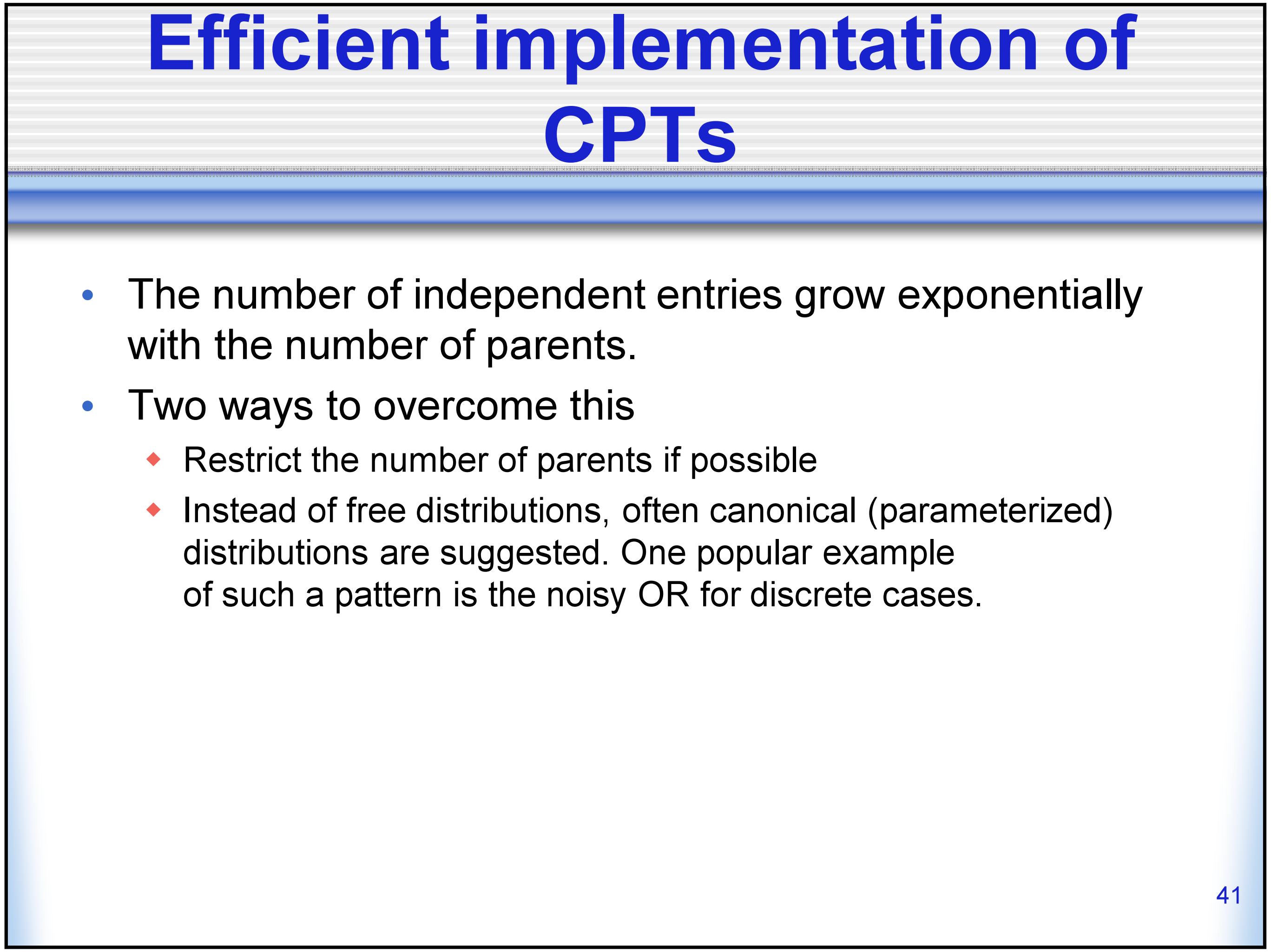
$\mathbf{P}(E \mid B, A, J, M) = \mathbf{P}(E \mid A)$? **No**

$\mathbf{P}(E \mid B, A, J, M) = \mathbf{P}(E \mid A, B)$? **Yes**



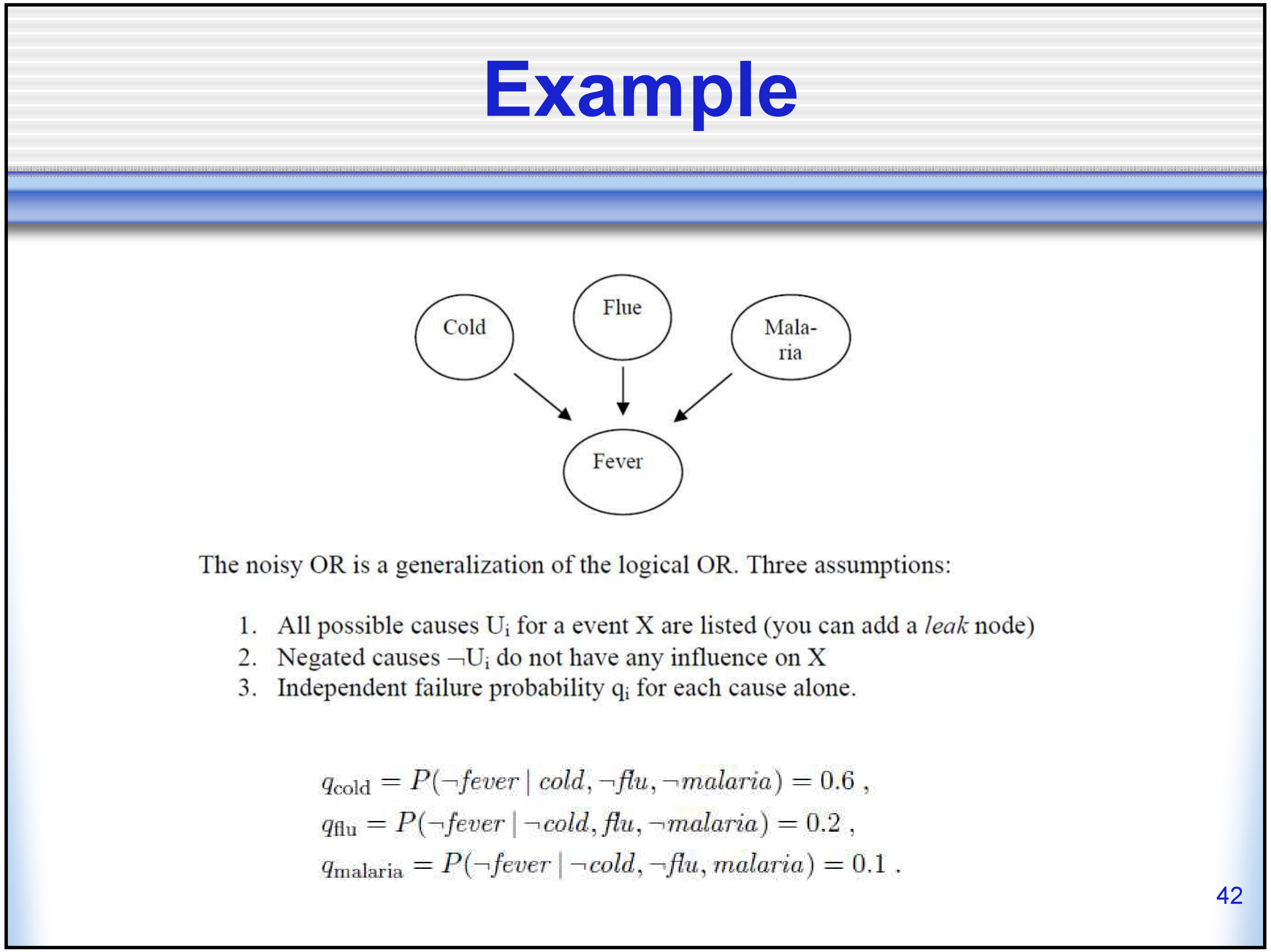
# Example contd.

- Deciding conditional independence is hard in noncausal directions
- (Causal models and conditional independence seem hardwired for humans!)
- Network is less compact: 1 + 2 + 4 + 2 + 4 = 13 numbers needed instead of 10.



# Efficient implementation of CPTs

- The number of independent entries grow exponentially with the number of parents.
- Two ways to overcome this
  - Restrict the number of parents if possible
  - Instead of free distributions, often canonical (parameterized) distributions are suggested. One popular example of such a pattern is the noisy OR for discrete cases.



# Example

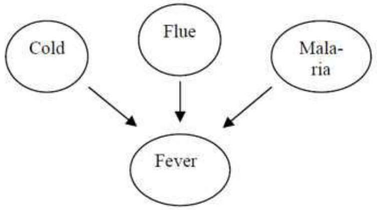
The noisy OR is a generalization of the logical OR. Three assumptions:

1. All possible causes $U_i$ for a event X are listed (you can add a *leak* node)
2. Negated causes $\neg U_i$ do not have any influence on X
3. Independent failure probability $q_i$ for each cause alone.

$$q_{\text{cold}} = P(\neg fever \mid cold, \neg flu, \neg malaria) = 0.6\ ,$$
$$q_{\text{flu}} = P(\neg fever \mid \neg cold, flu, \neg malaria) = 0.2\ ,$$
$$q_{\text{malaria}} = P(\neg fever \mid \neg cold, \neg flu, malaria) = 0.1\ .$$

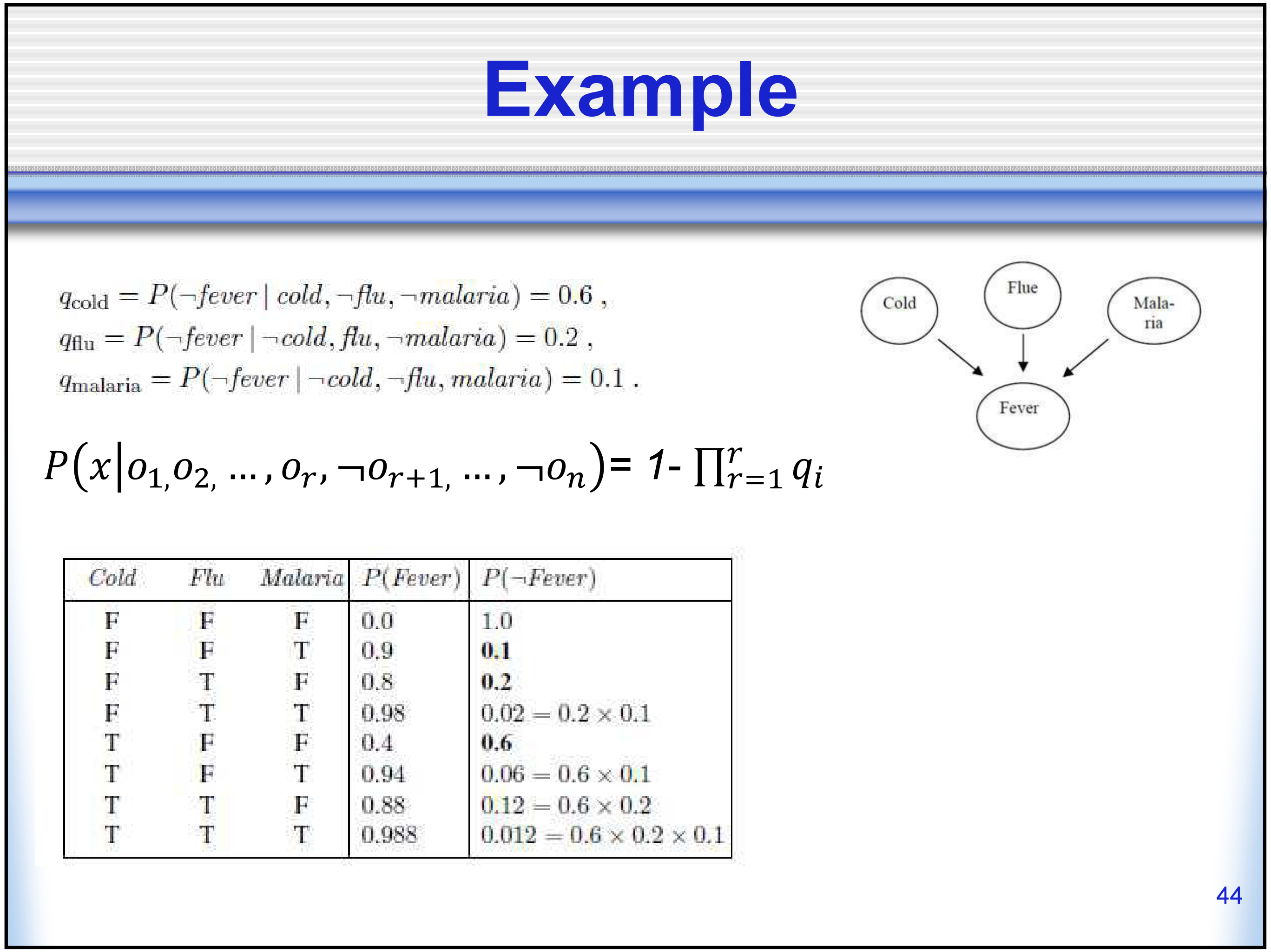
# Example

$$q_{\text{cold}} = P(\neg fever \mid cold, \neg flu, \neg malaria) = 0.6\ ,$$
$$q_{\text{flu}} = P(\neg fever \mid \neg cold, flu, \neg malaria) = 0.2\ ,$$
$$q_{\text{malaria}} = P(\neg fever \mid \neg cold, \neg flu, malaria) = 0.1\ .$$



$$P(\neg x | o_{1,} o_{2,} \ldots, o_r, \neg o_{r+1,} \ldots, \neg o_n) = \prod_{r=1}^{r} q_i$$

| *Cold* | *Flu* | *Malaria* | *P(Fever)* | *P(¬Fever)* |
|---|---|---|---|---|
| F | F | F | | |
| F | F | T | | **0.1** |
| F | T | F | | **0.2** |
| F | T | T | | |
| T | F | F | | **0.6** |
| T | F | T | | |
| T | T | F | | |
| T | T | T | | |



# Example

$$q_{\text{cold}} = P(\neg fever \mid cold, \neg flu, \neg malaria) = 0.6\ ,$$
$$q_{\text{flu}} = P(\neg fever \mid \neg cold, flu, \neg malaria) = 0.2\ ,$$
$$q_{\text{malaria}} = P(\neg fever \mid \neg cold, \neg flu, malaria) = 0.1\ .$$

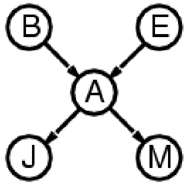
Cold
Flue
Mala-
ria
Fever

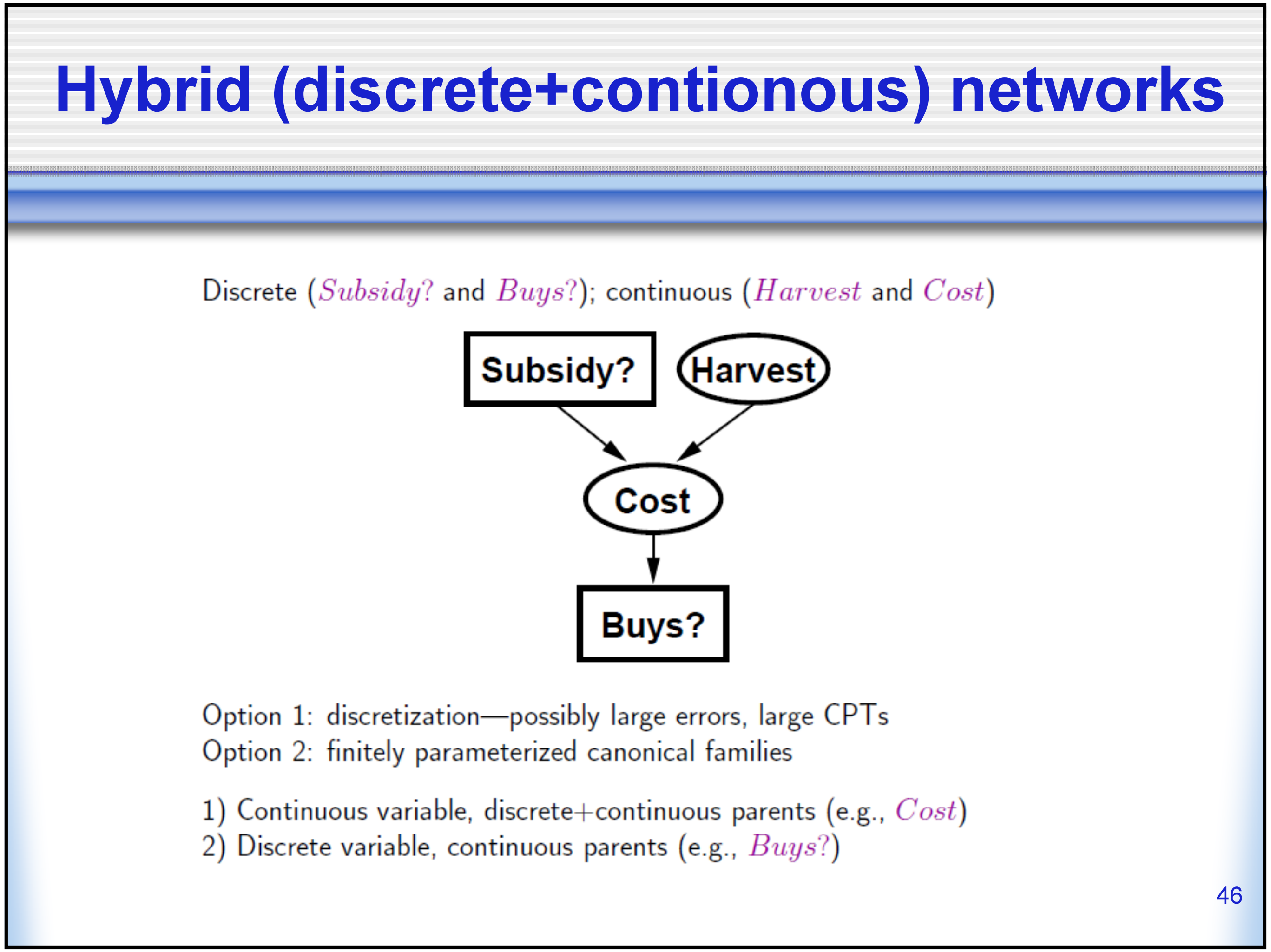
$$P(x | o_{1,} o_{2,} \dots, o_r, \neg o_{r+1,} \dots, \neg o_n) = 1 - \prod_{r=1}^{r} q_i$$

| *Cold* | *Flu* | *Malaria* | *P(Fever)* | *P(¬Fever)* |
|---|---|---|---|---|
| F | F | F | 0.0 | 1.0 |
| F | F | T | 0.9 | **0.1** |
| F | T | F | 0.8 | **0.2** |
| F | T | T | 0.98 | $0.02 = 0.2 \times 0.1$ |
| T | F | F | 0.4 | **0.6** |
| T | F | T | 0.94 | $0.06 = 0.6 \times 0.1$ |
| T | T | F | 0.88 | $0.12 = 0.6 \times 0.2$ |
| T | T | T | 0.988 | $0.012 = 0.6 \times 0.2 \times 0.1$ |

# Last Time

- Structure and semantic of BN
  - Modelling of independence and conditional independence
  - Causal and non-causal networks
  - d-separation, Markov blanket
  - Efficient CPTs, e.g., noisy OR, trees, Min, Max, …





# Hybrid (discrete+contionous) networks

Discrete (*Subsidy?* and *Buys?*); continuous (*Harvest* and *Cost*)

Subsidy?

Harvest

Cost

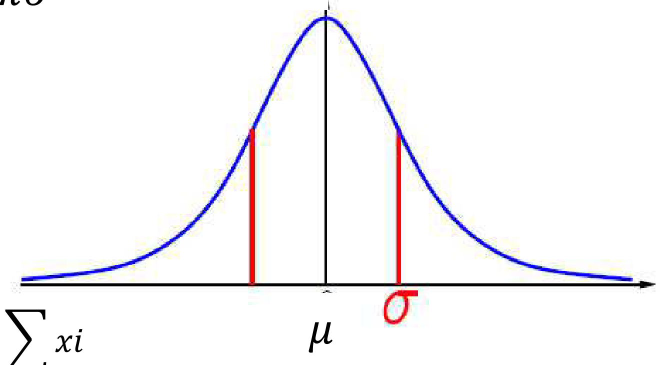
Buys?

Option 1: discretization—possibly large errors, large CPTs
Option 2: finitely parameterized canonical families

1) Continuous variable, discrete+continuous parents (e.g., *Cost*)
2) Discrete variable, continuous parents (e.g., *Buys?*)

# Continous variables: Gaussian density

$$P(x) = \frac{1}{\sqrt{2\pi\sigma}} e^{-(x-\mu)^2} / 2\sigma^2$$



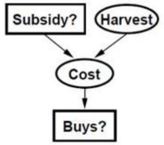
$$Mean \quad \mu = \frac{1}{n}\sum_{n} xi$$

$$Variance\ \sigma^2 = \frac{1}{n-1}\sum_{n}(xi\ -\ \mu)^2$$

$$Standard\ deviation\ \sigma = \sqrt{\sigma^2}$$

# Continuous child variables

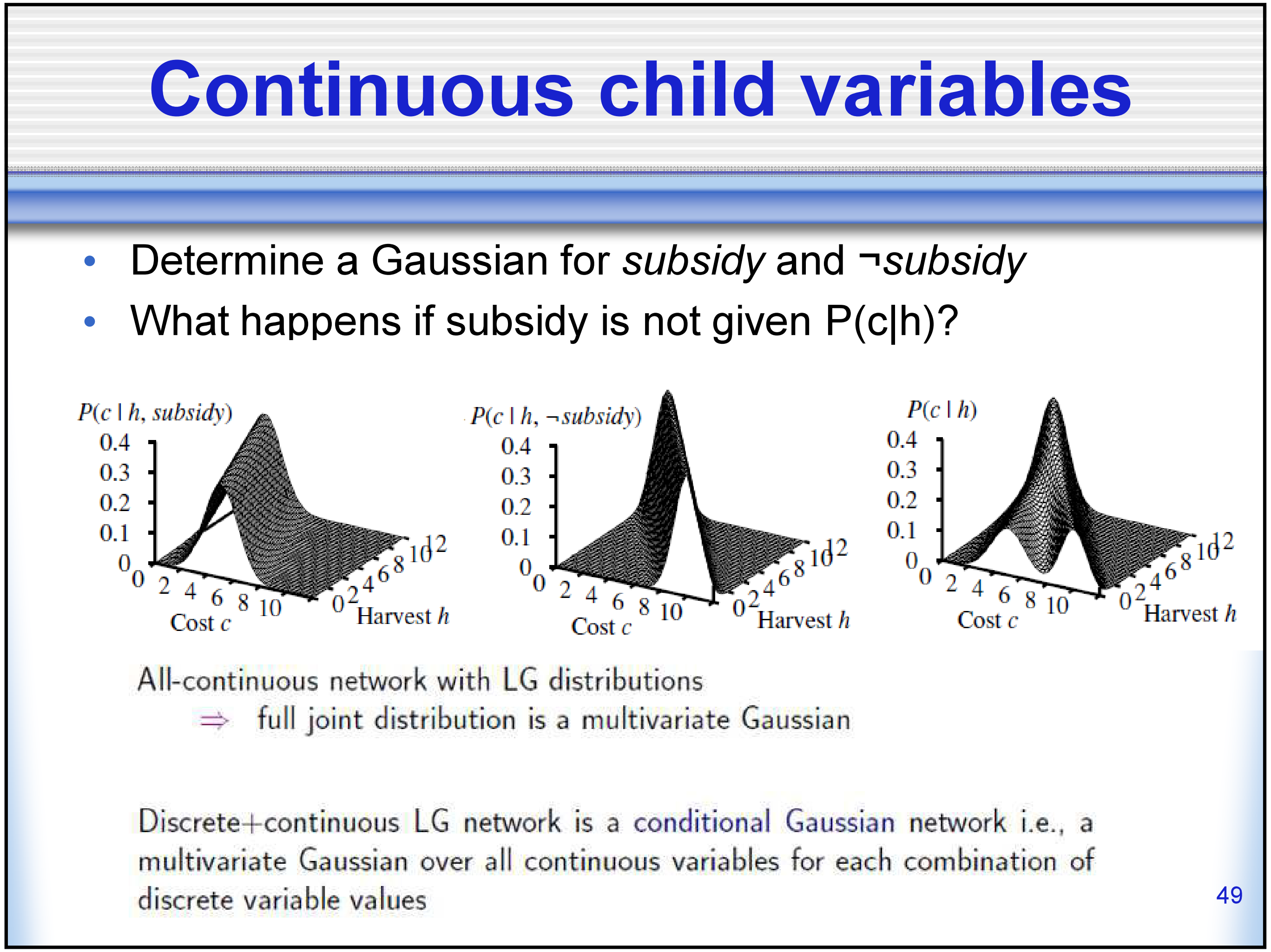
- Need one *conditional density* function for child variable given
  - continuous parents
  - for each discrete value of parents



$$P(c\,|\,h, subsidy) \;=\; N(a_t h + b_t, \sigma_t^2)(c) = \frac{1}{\sigma_t\sqrt{2\pi}}\, e^{-\frac{1}{2}\left(\frac{c-(a_t h+b_t)}{\sigma_t}\right)^2}$$

$$P(c\,|\,h, \neg subsidy) \;=\; N(a_f h + b_f, \sigma_f^2)(c) = \frac{1}{\sigma_f\sqrt{2\pi}}\, e^{-\frac{1}{2}\left(\frac{c-(a_f h+b_f)}{\sigma_f}\right)^2}$$

- Mean **Cost** varies *linearly* with **Harvest**, variance fixed
- Linear variation is unreasonable over the full range but works if the likely range of **Harvest** is narrow



# Continuous child variables

- Determine a Gaussian for *subsidy* and ¬*subsidy*
- What happens if subsidy is not given P(c|h)?

$P(c \mid h, subsidy)$
$P(c \mid h, \neg subsidy)$
$P(c \mid h)$
0.4
0.3
0.2
0.1
0
0 2 4 6 8 10
Cost $c$
0 2 4 6 8 10 12
Harvest $h$

All-continuous network with LG distributions

⇒ full joint distribution is a multivariate Gaussian

Discrete+continuous LG network is a conditional Gaussian network i.e., a multivariate Gaussian over all continuous variables for each combination of discrete variable values

# Discrete variabel cont. parents

- Probability of Buys given Cost should be a soft threshold

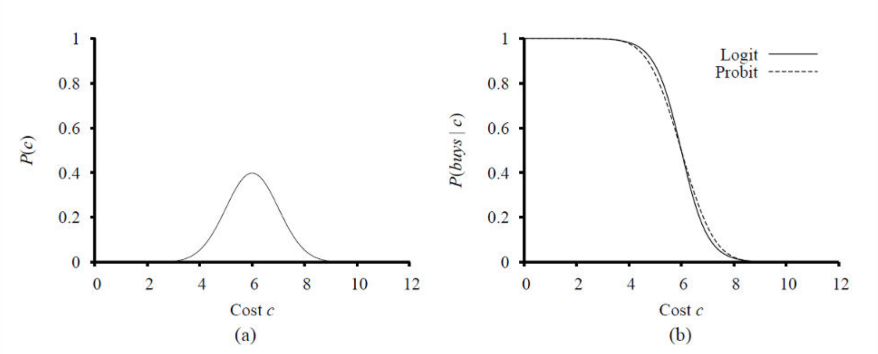
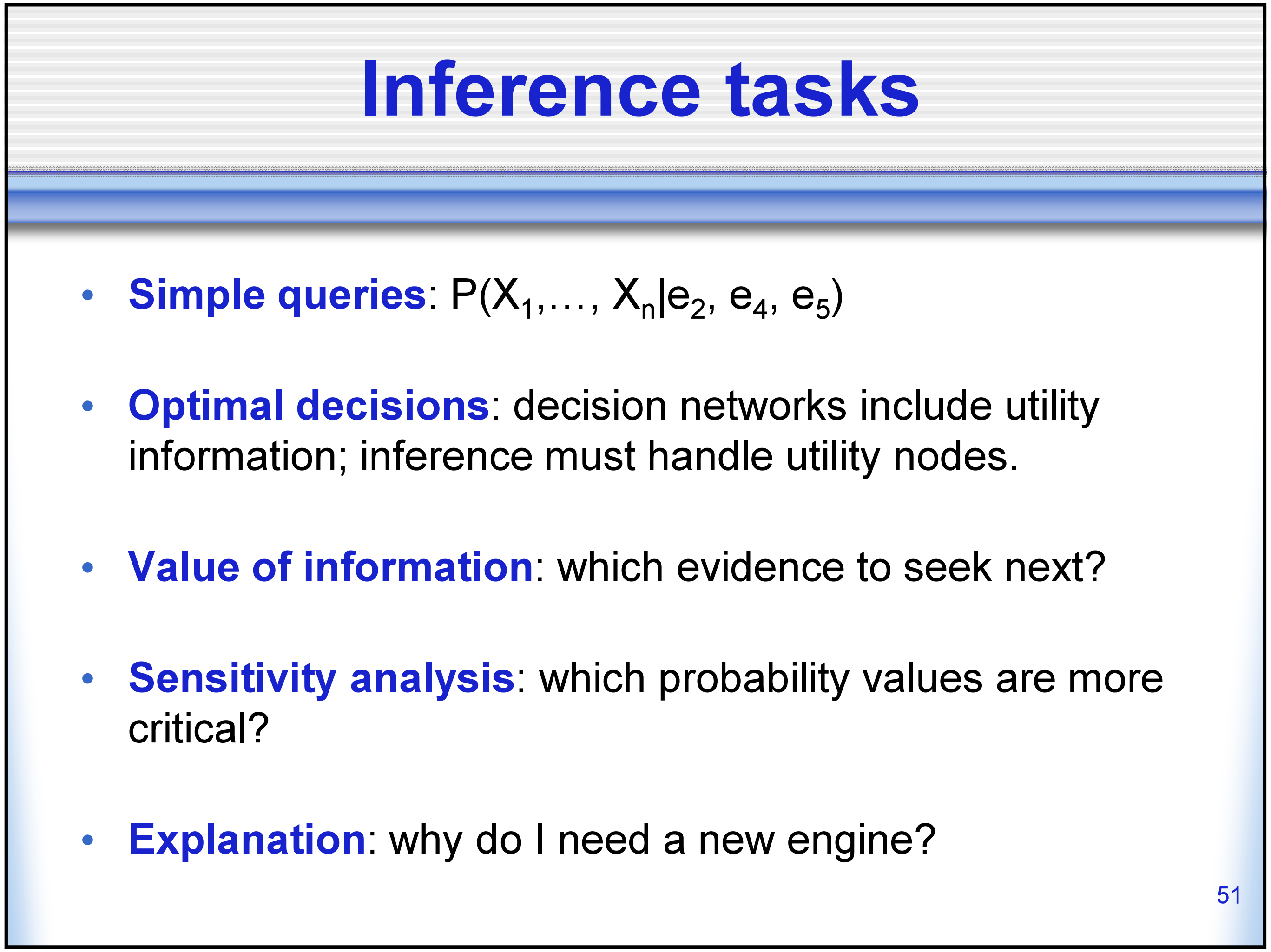


**Figure 14.7** **FILES: .** (a) A normal (Gaussian) distribution for the cost threshold, centered on $\mu = 6.0$ with standard deviation $\sigma = 1.0$. (b) Logit and probit distributions for the probability of *buys* given *cost*, for the parameters $\mu = 6.0$ and $\sigma = 1.0$.

Use integral $\Phi(x) = \int_{-\infty}^{x} N(0,1)(x)dx$

Leads to $P(buys \mid Cost = c) = \Phi((-c + \mu)/\sigma)$ Probit

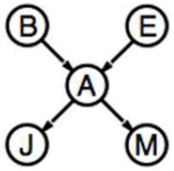
Alternativ $P(buys \mid Cost = c) = \frac{1}{1 + exp(-2\frac{-c+\mu}{\sigma})}$. Logit



# Inference tasks

- **Simple queries**: $P(X_1, \ldots, X_n | e_2, e_4, e_5)$
- **Optimal decisions**: decision networks include utility information; inference must handle utility nodes.
- **Value of information**: which evidence to seek next?
- **Sensitivity analysis**: which probability values are more critical?
- **Explanation**: why do I need a new engine?

# Inference by enumeration

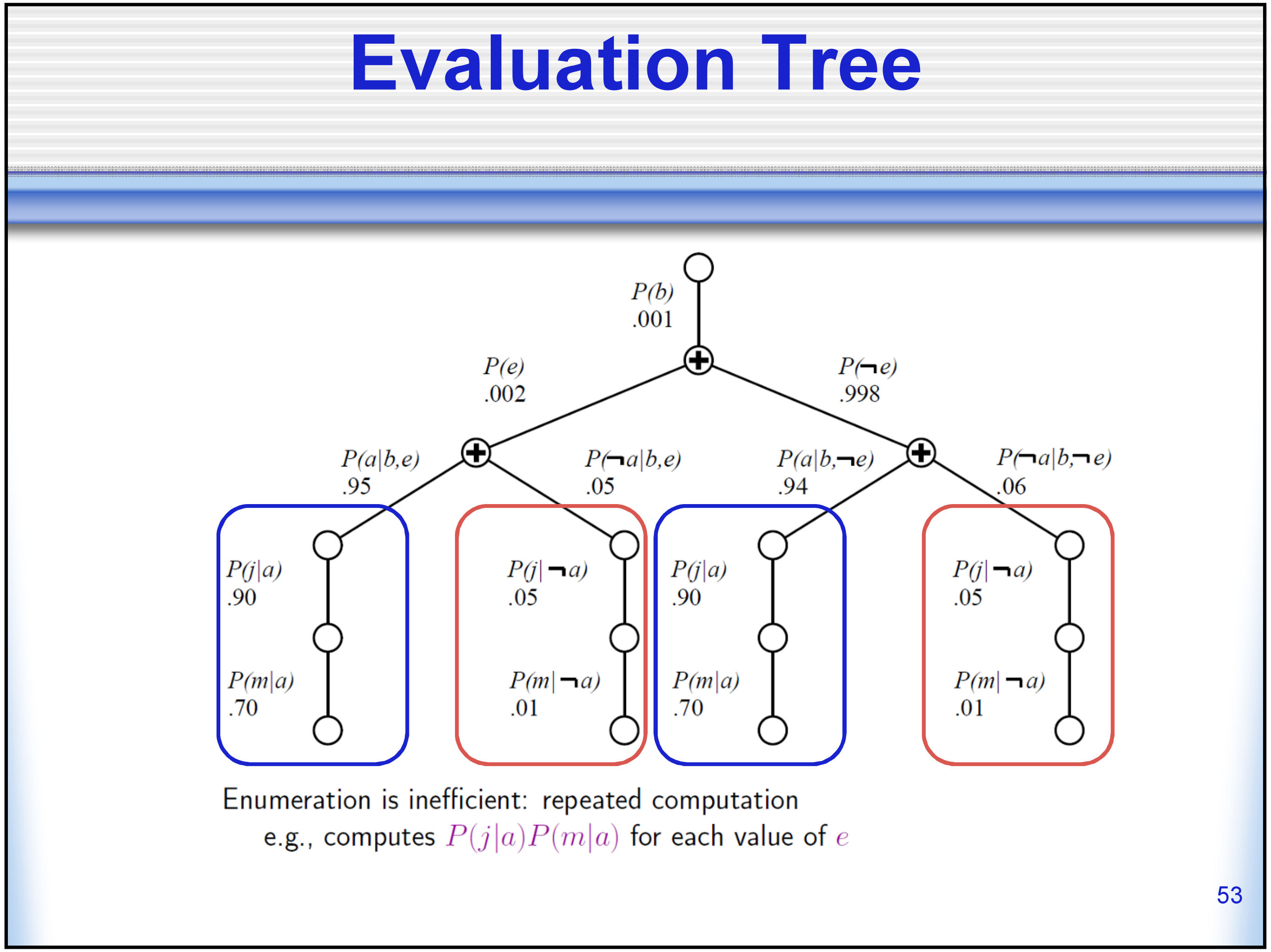


$P(b|j,m) = \alpha\ P(b,j,m)$

$= \alpha\ \Sigma_a\Sigma_e P(b \wedge j \wedge m \wedge a \wedge e)$ [marginalization]

$= \alpha\ \Sigma_a\Sigma_e P(b)P(e)P(a|b,e)P(j|a)P(m|a)$ [BN]

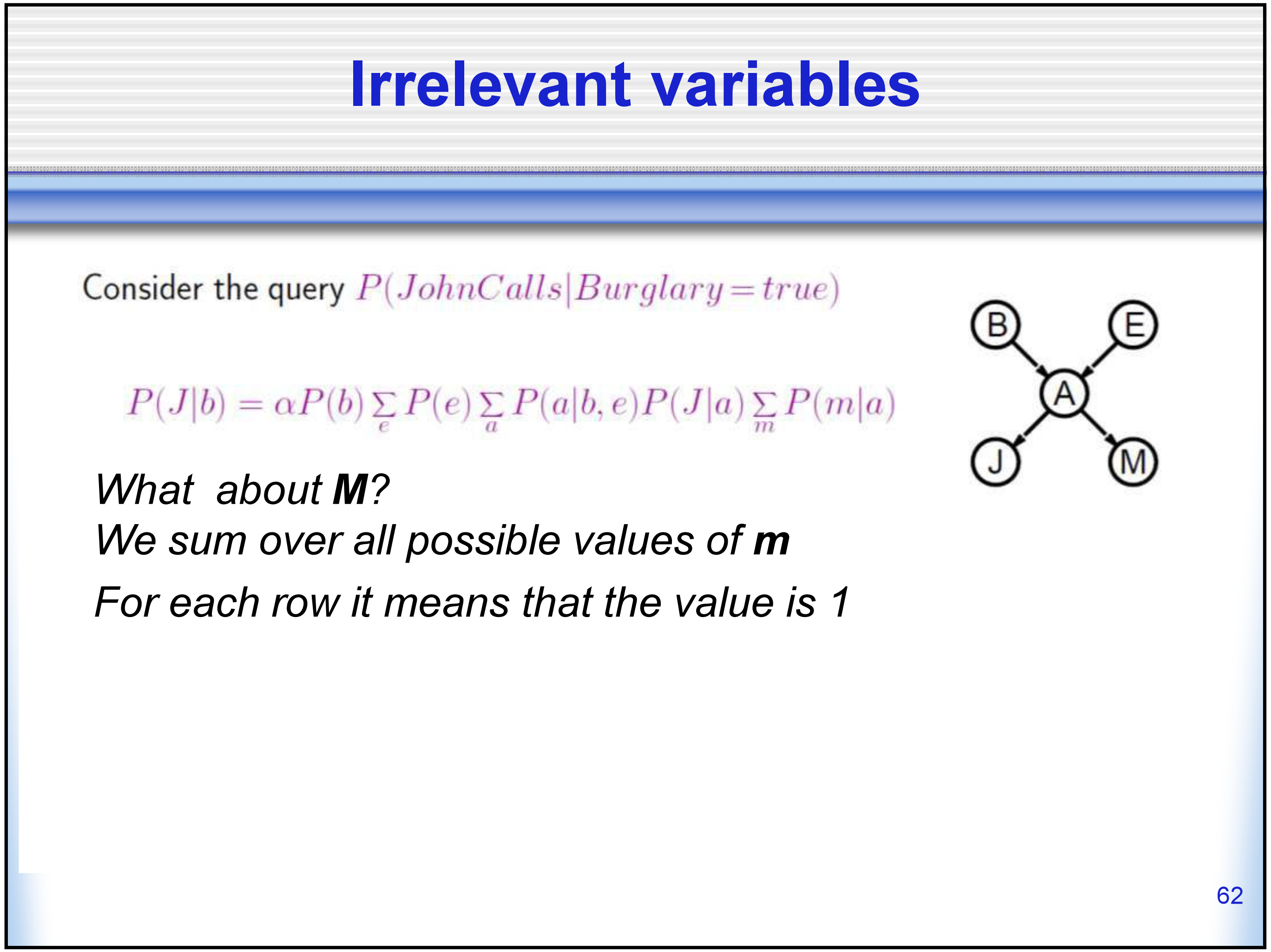
$= \alpha\ P(b)\Sigma_e P(e)\Sigma_a P(a|b,e)P(j|a)P(m|a)$ [re-ordering]



# Evaluation Tree

Enumeration is inefficient: repeated computation

e.g., computes $P(j|a)P(m|a)$ for each value of $e$



## Irrelevant variables

Consider the query $P(JohnCalls|Burglary=true)$

$$P(J|b) = \alpha P(b) \sum_{e} P(e) \sum_{a} P(a|b,e) P(J|a) \sum_{m} P(m|a)$$

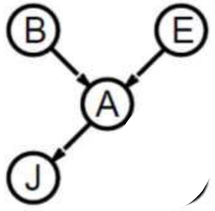
*What about **M**?*
*We sum over all possible values of **m***

*For each row it means that the value is 1*

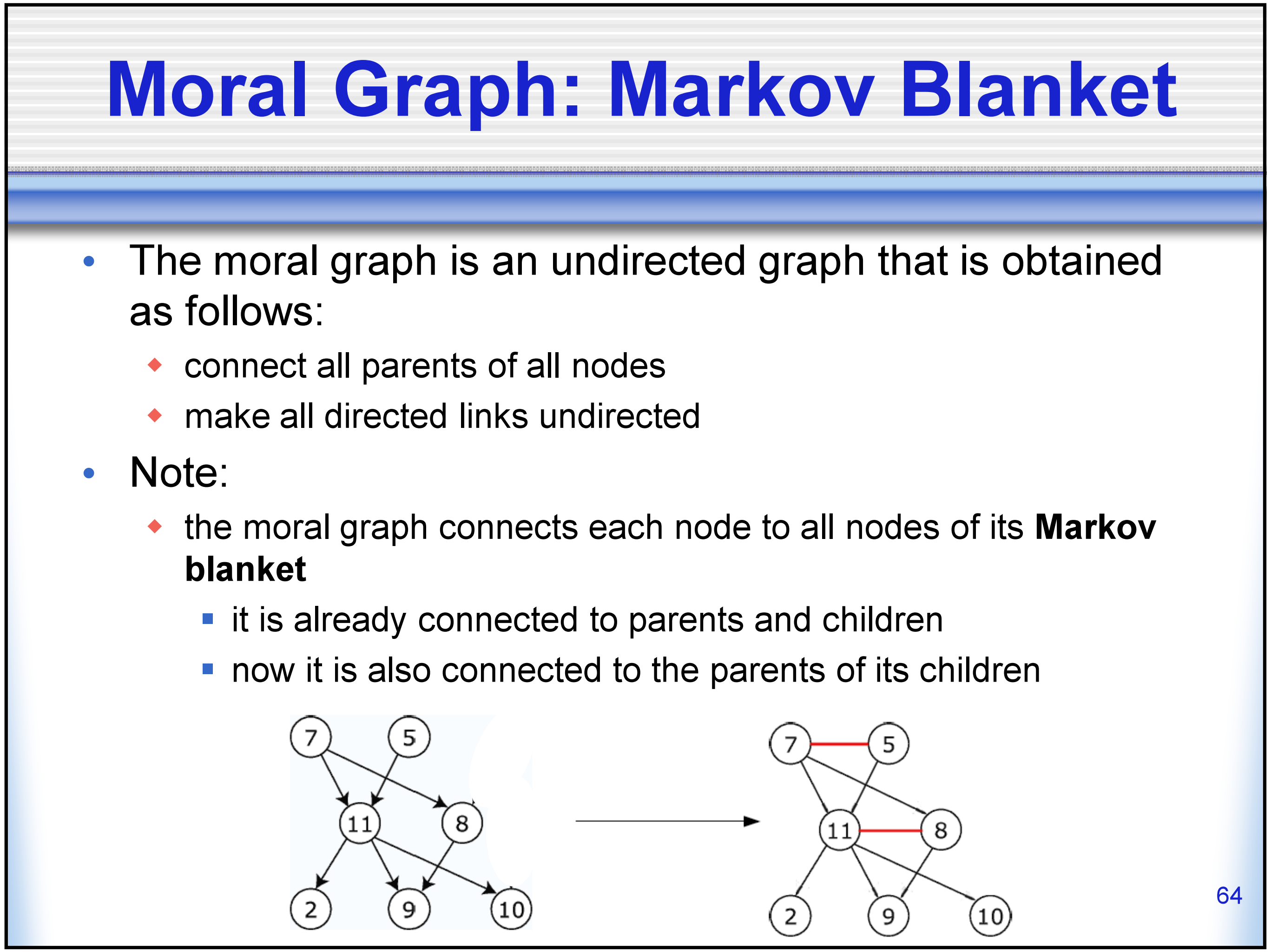
# Irrelevant variables

Consider the query $P(JohnCalls|Burglary=true)$

$$P(J|b) = \alpha P(b) \sum_e P(e) \sum_a P(a|b,e) P(J|a) \sum_m P(m|a)$$

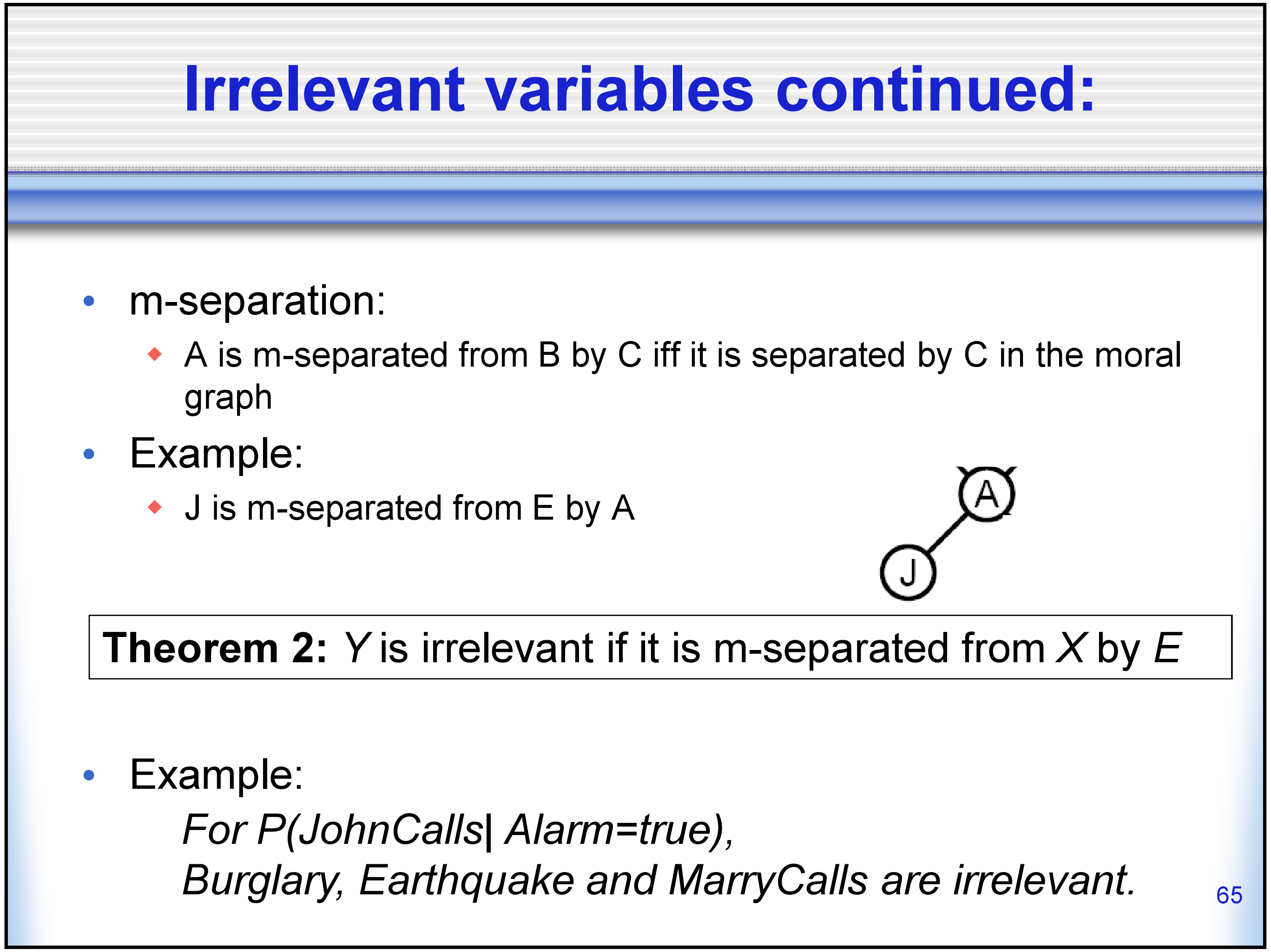


*For each row it means that the value is 1*



# Moral Graph: Markov Blanket

- The moral graph is an undirected graph that is obtained as follows:
  - connect all parents of all nodes
  - make all directed links undirected
- Note:
  - the moral graph connects each node to all nodes of its **Markov blanket**
    - it is already connected to parents and children
    - now it is also connected to the parents of its children



# Irrelevant variables continued:

- m-separation:
  - A is m-separated from B by C iff it is separated by C in the moral graph
- Example:
  - J is m-separated from E by A

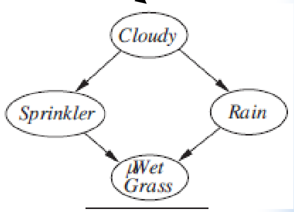
**Theorem 2:** *Y* is irrelevant if it is m-separated from *X* by *E*

- Example:

  *For P(JohnCalls| Alarm=true),*
  *Burglary, Earthquake and MarryCalls are irrelevant.*

# Approximate Inference In Bayesian Network

- Singly connected networks (or polytrees):
  - any two nodes are connected by at most one (undirected) path
  - time and space cost of variable elimination linear in the size of the network (number of CPT entries; number of parents O($d^k n$)).
- Multiply connected networks: NP-hard
- We need **approximate inference techniques**!!!!!!!
- Monte Carlo algorithm
  - Widely used to estimate quantities that are difficult to calculate exactly
  - Randomized sampling algorithm
  - Accuracy depends on the number of samples
  - Two families
    - Direct sampling
    - Markov chain sampling



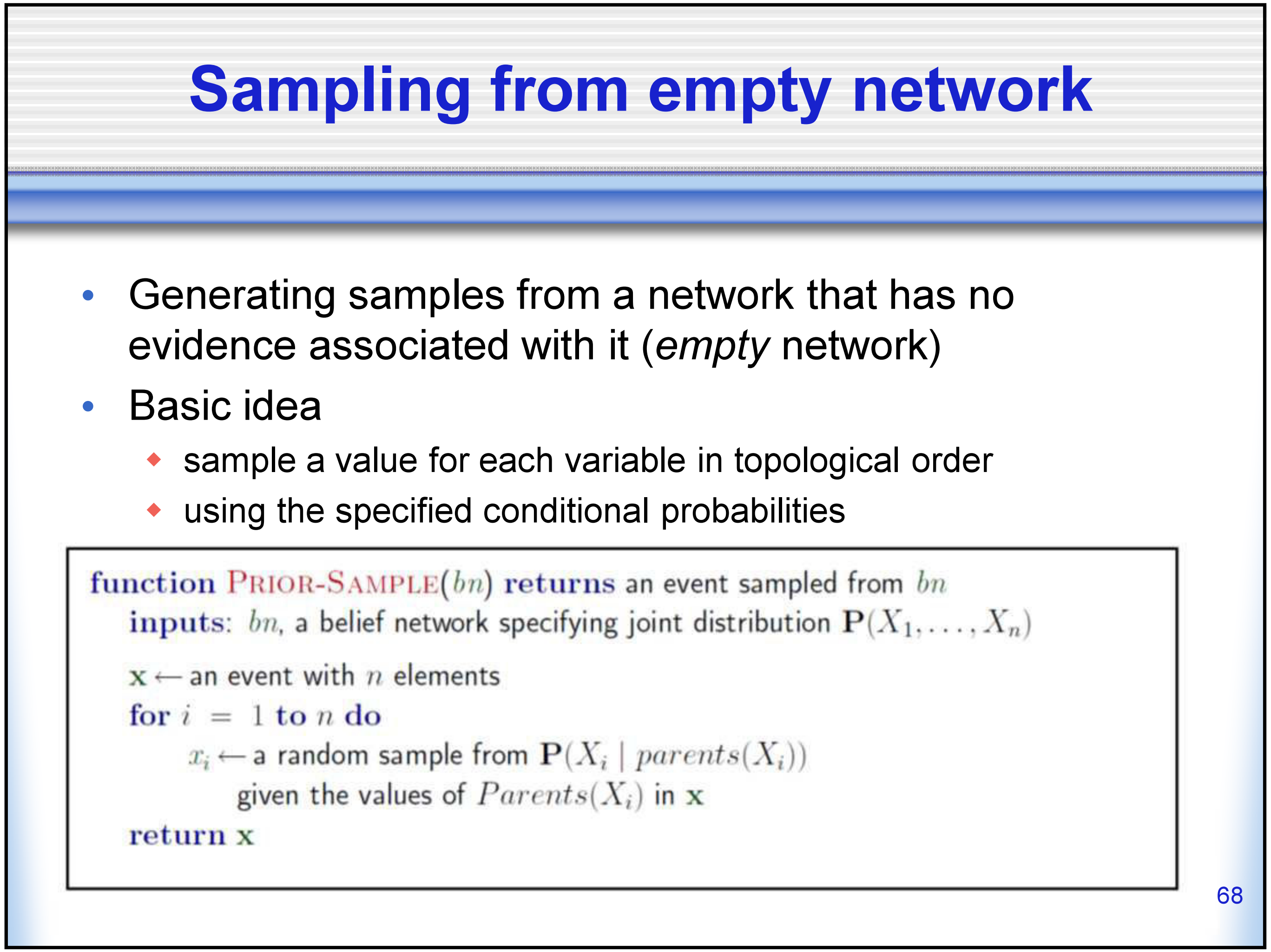
# Inference by stochastic simulation

Basic idea:

1) Draw $N$ samples from a sampling distribution $S$
2) Compute an approximate posterior probability $\hat{P}$
3) Show this converges to the true probability $P$

Outline:

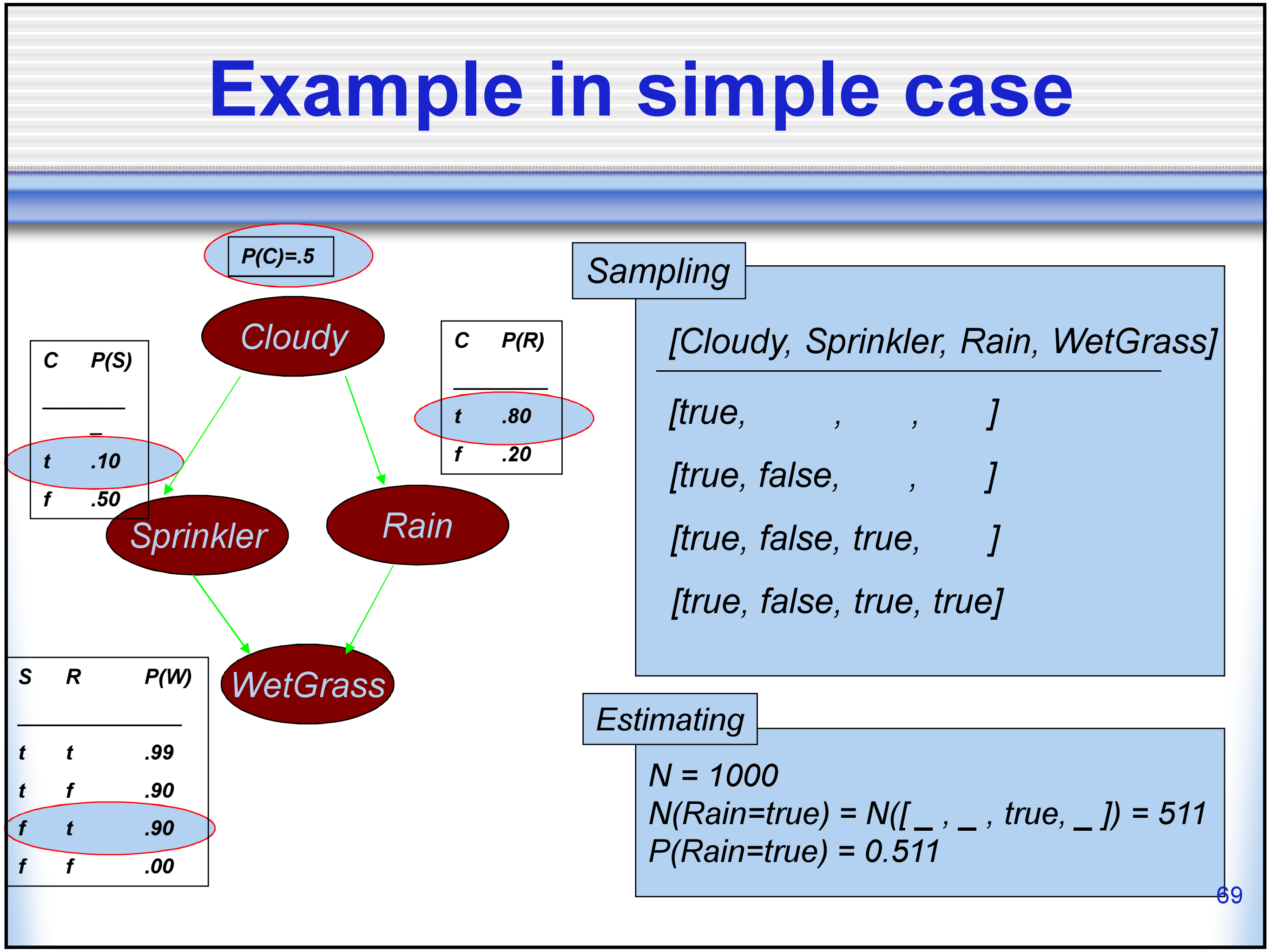
- Sampling from an empty network
- Rejection sampling: reject samples disagreeing with evidence
- Likelihood weighting: use evidence to weight samples
- Markov chain Monte Carlo (MCMC): sample from a stochastic process whose stationary distribution is the true posterior



## Sampling from empty network

- Generating samples from a network that has no evidence associated with it (*empty* network)
- Basic idea
  - sample a value for each variable in topological order
  - using the specified conditional probabilities

```
function PRIOR-SAMPLE($bn$) returns an event sampled from $bn$
   inputs: $bn$, a belief network specifying joint distribution $\mathbf{P}(X_1, \ldots, X_n)$

   $\mathbf{x} \leftarrow$ an event with $n$ elements
   for $i = 1$ to $n$ do
       $x_i \leftarrow$ a random sample from $\mathbf{P}(X_i \mid parents(X_i))$
            given the values of $Parents(X_i)$ in $\mathbf{x}$
   return $\mathbf{x}$
```



# Example in simple case

P(C)=.5

| C | P(S) |
|---|---|
| t | .10 |
| f | .50 |

| C | P(R) |
|---|---|
| t | .80 |
| f | .20 |

| S | R | P(W) |
|---|---|---|
| t | t | .99 |
| t | f | .90 |
| f | t | .90 |
| f | f | .00 |

## Sampling

[Cloudy, Sprinkler, Rain, WetGrass]

[true, , , ]

[true, false, , ]

[true, false, true, ]

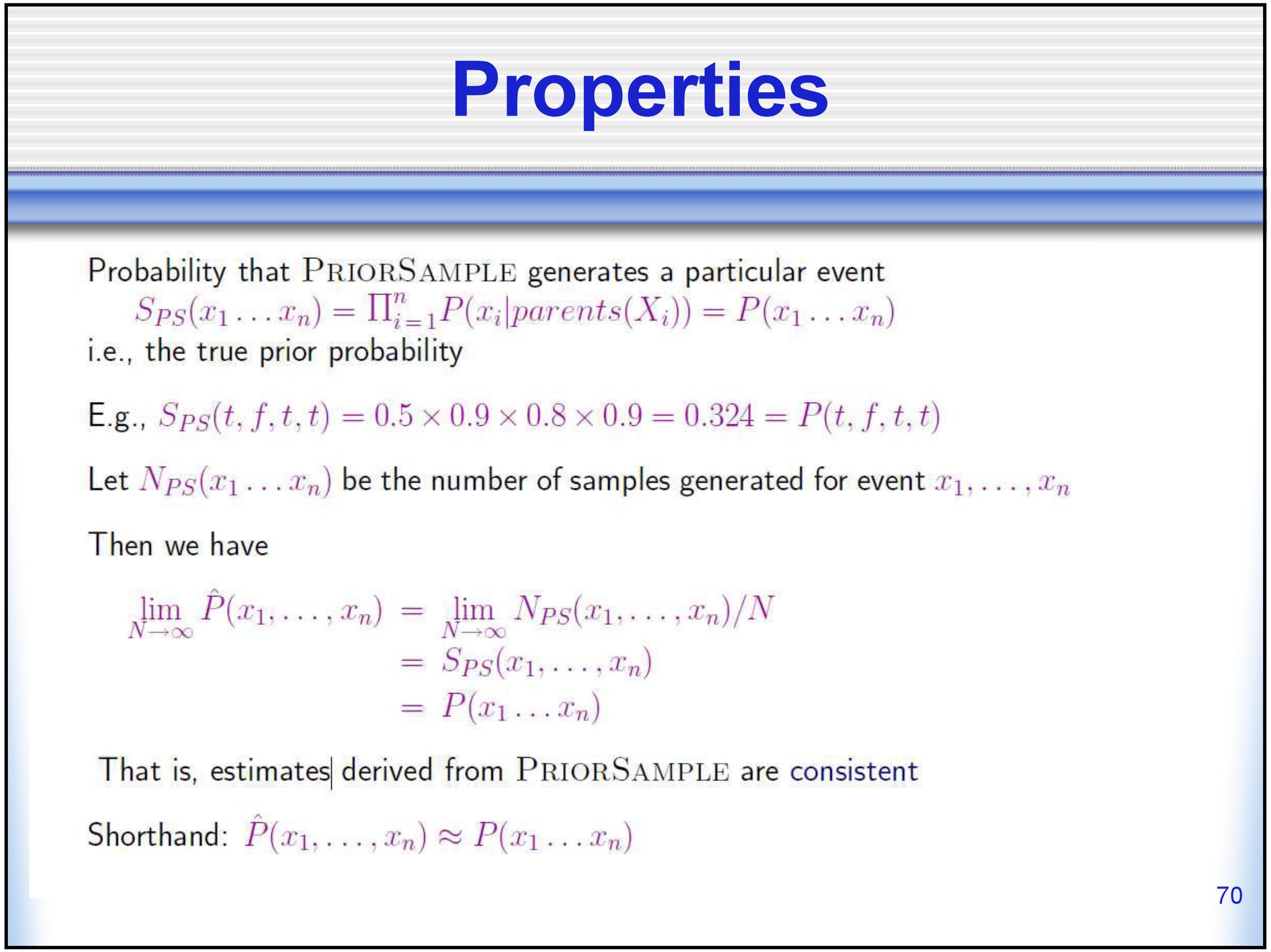
[true, false, true, true]

## Estimating

N = 1000

N(Rain=true) = N([ _ , _ , true, _ ]) = 511

P(Rain=true) = 0.511



# Properties

Probability that PRIORSAMPLE generates a particular event

$S_{PS}(x_1 \ldots x_n) = \Pi_{i=1}^{n} P(x_i | parents(X_i)) = P(x_1 \ldots x_n)$

i.e., the true prior probability

E.g., $S_{PS}(t, f, t, t) = 0.5 \times 0.9 \times 0.8 \times 0.9 = 0.324 = P(t, f, t, t)$

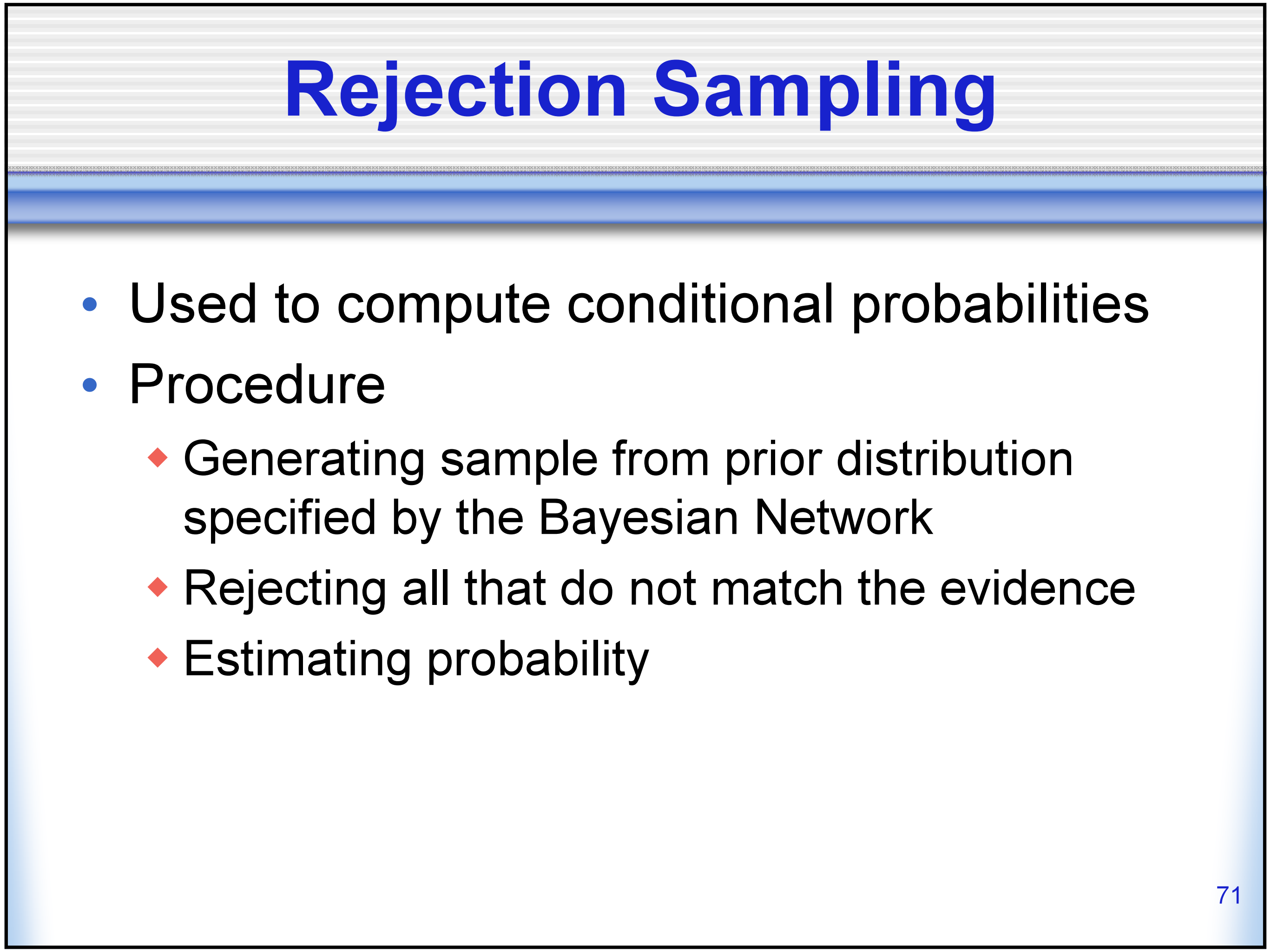
Let $N_{PS}(x_1 \ldots x_n)$ be the number of samples generated for event $x_1, \ldots, x_n$

Then we have

$$\begin{aligned}
\lim_{N \to \infty} \hat{P}(x_1, \ldots, x_n) &= \lim_{N \to \infty} N_{PS}(x_1, \ldots, x_n)/N \\
&= S_{PS}(x_1, \ldots, x_n) \\
&= P(x_1 \ldots x_n)
\end{aligned}$$

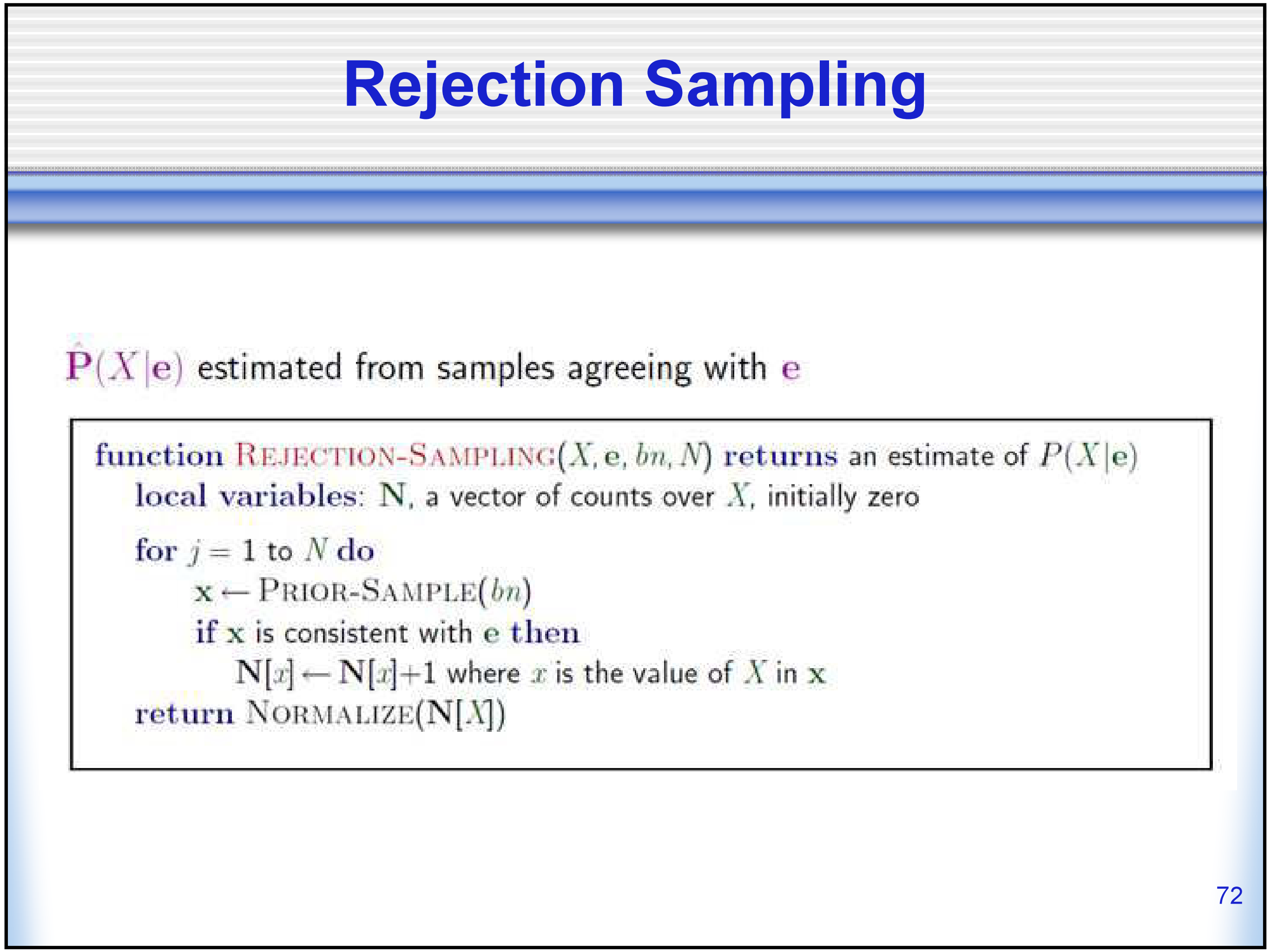
That is, estimates derived from PRIORSAMPLE are consistent

Shorthand: $\hat{P}(x_1, \ldots, x_n) \approx P(x_1 \ldots x_n)$



# Rejection Sampling

- Used to compute conditional probabilities
- Procedure
  - Generating sample from prior distribution specified by the Bayesian Network
  - Rejecting all that do not match the evidence
  - Estimating probability

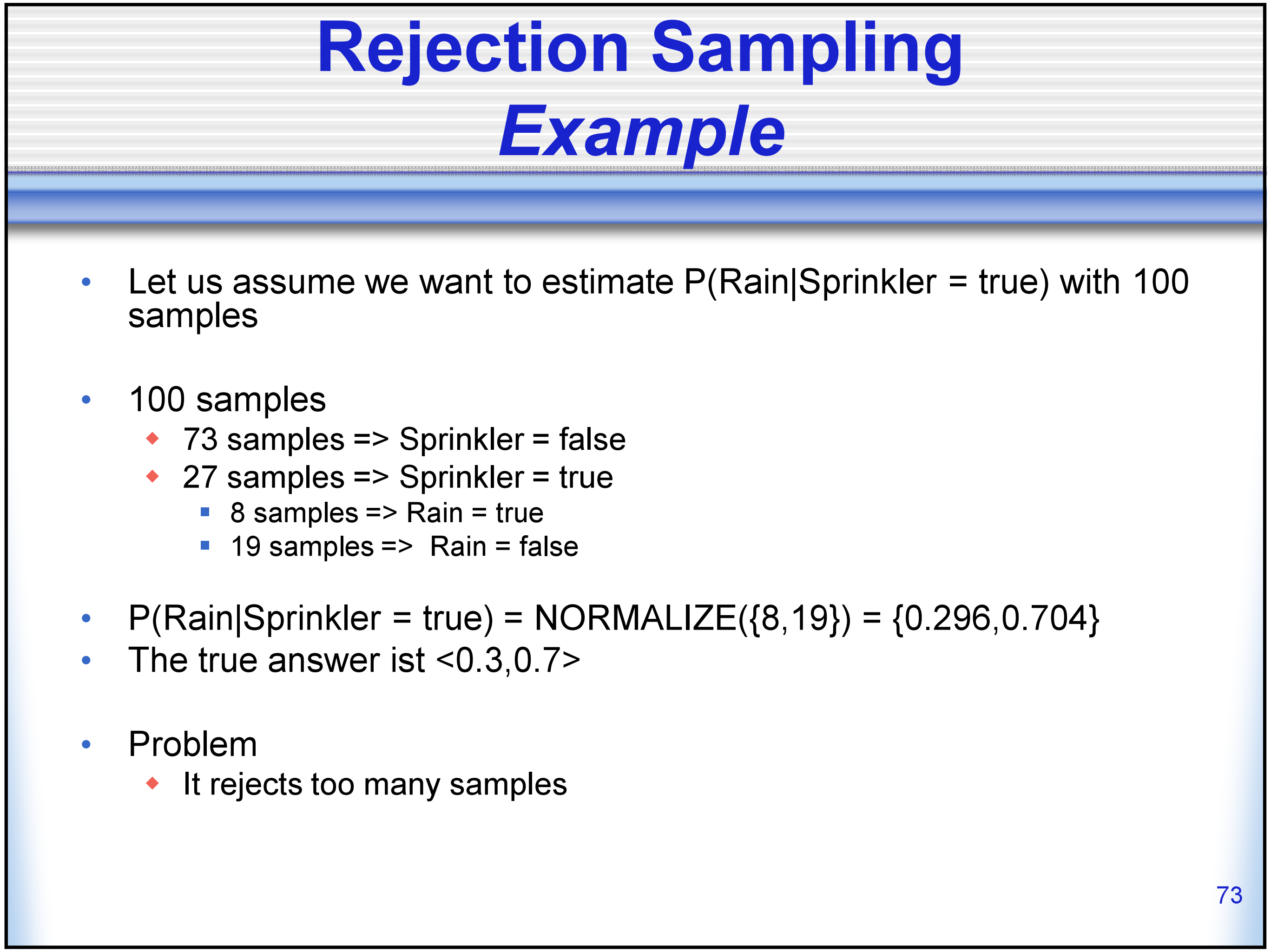


# Rejection Sampling

$\hat{\mathbf{P}}(X|\mathbf{e})$ estimated from samples agreeing with $\mathbf{e}$

```
function REJECTION-SAMPLING($X$, e, $bn$, $N$) returns an estimate of $P(X|\mathbf{e})$
  local variables: N, a vector of counts over $X$, initially zero

  for $j = 1$ to $N$ do
      x ← PRIOR-SAMPLE($bn$)
      if x is consistent with e then
          N[$x$] ← N[$x$]+1 where $x$ is the value of $X$ in x
  return NORMALIZE(N[$X$])
```

72



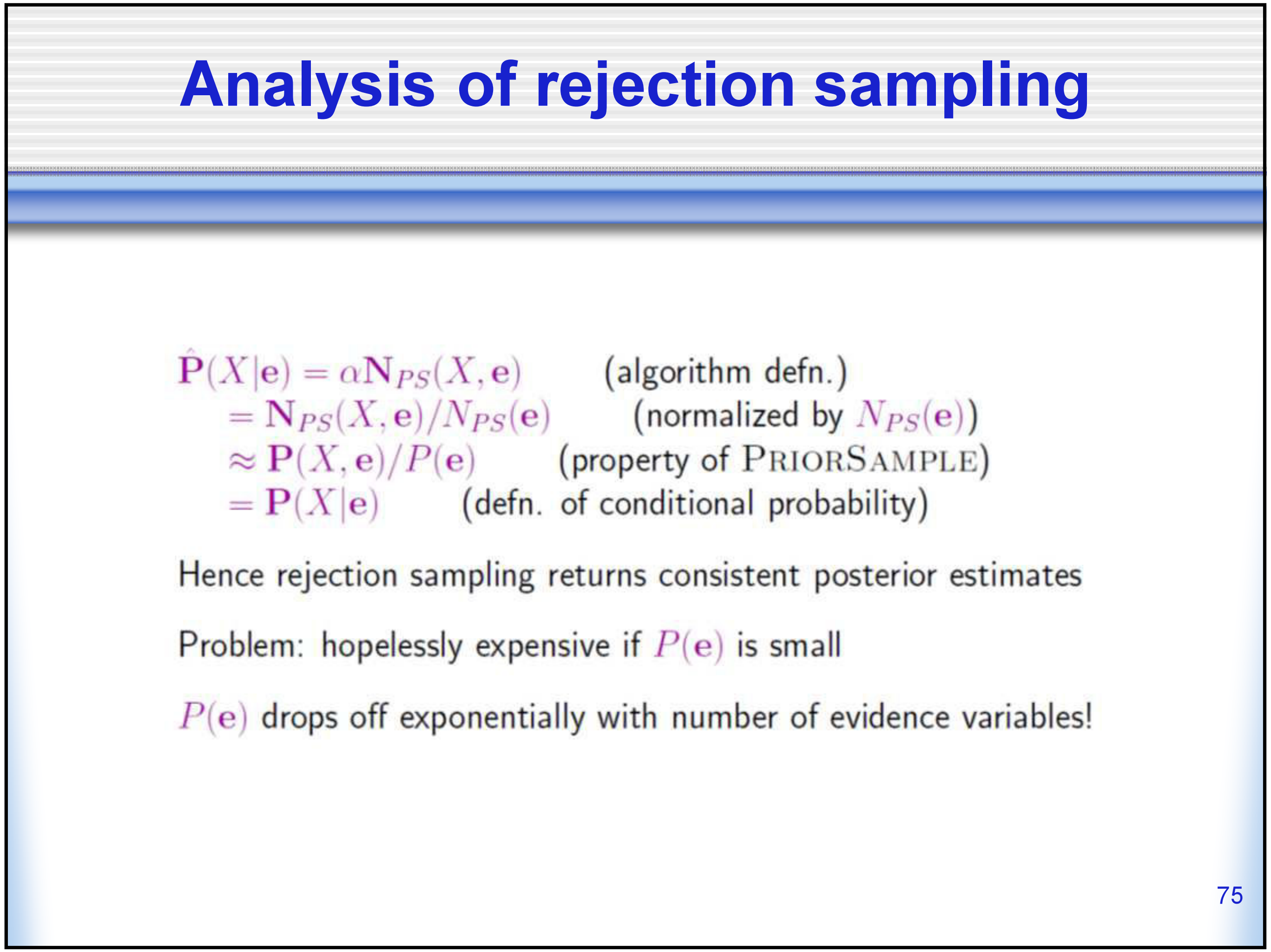
# Rejection Sampling *Example*

- Let us assume we want to estimate P(Rain|Sprinkler = true) with 100 samples

- 100 samples
  - 73 samples => Sprinkler = false
  - 27 samples => Sprinkler = true
    - 8 samples => Rain = true
    - 19 samples => Rain = false

- P(Rain|Sprinkler = true) = NORMALIZE({8,19}) = {0.296,0.704}
- The true answer ist <0.3,0.7>

- Problem
  - It rejects too many samples



# Analysis of rejection sampling

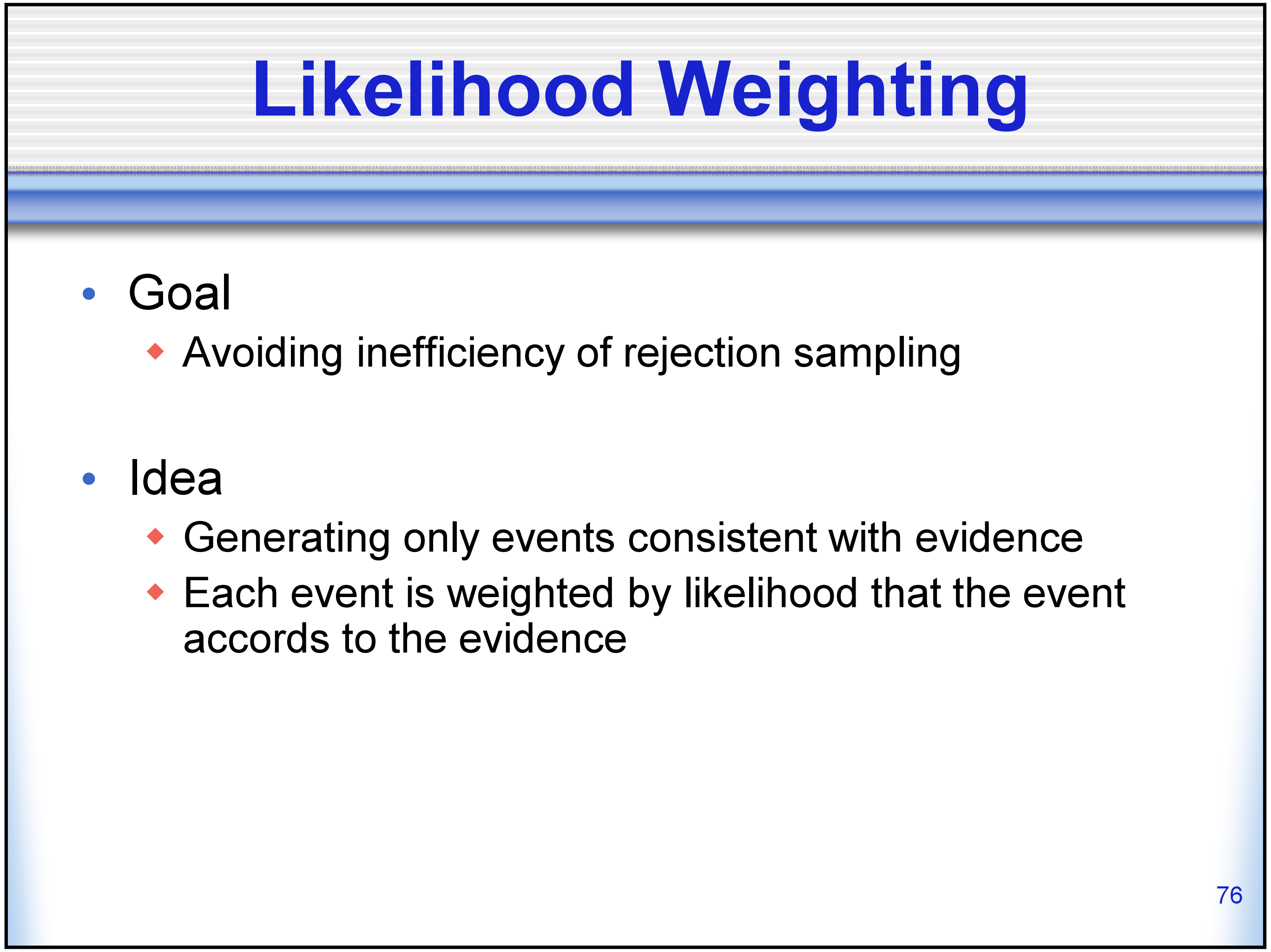
$$\begin{aligned}\hat{\mathbf{P}}(X|\mathbf{e}) &= \alpha \mathbf{N}_{PS}(X,\mathbf{e}) && \text{(algorithm defn.)}\\ &= \mathbf{N}_{PS}(X,\mathbf{e})/N_{PS}(\mathbf{e}) && \text{(normalized by } N_{PS}(\mathbf{e})\text{)}\\ &\approx \mathbf{P}(X,\mathbf{e})/P(\mathbf{e}) && \text{(property of PriorSample)}\\ &= \mathbf{P}(X|\mathbf{e}) && \text{(defn. of conditional probability)}\end{aligned}$$

Hence rejection sampling returns consistent posterior estimates

Problem: hopelessly expensive if $P(\mathbf{e})$ is small

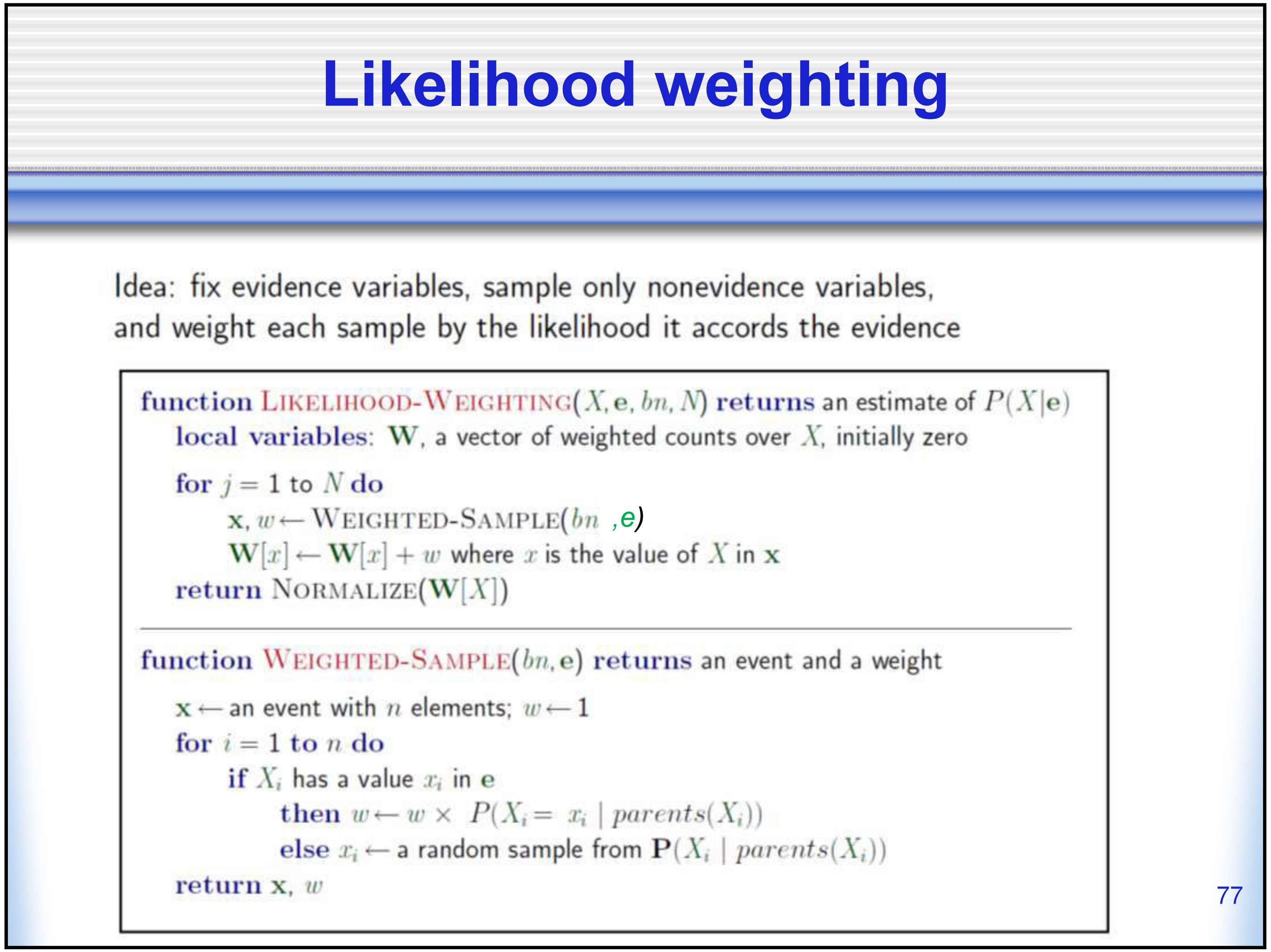
$P(\mathbf{e})$ drops off exponentially with number of evidence variables!

75



# Likelihood Weighting

- Goal
  - Avoiding inefficiency of rejection sampling
- Idea
  - Generating only events consistent with evidence
  - Each event is weighted by likelihood that the event accords to the evidence



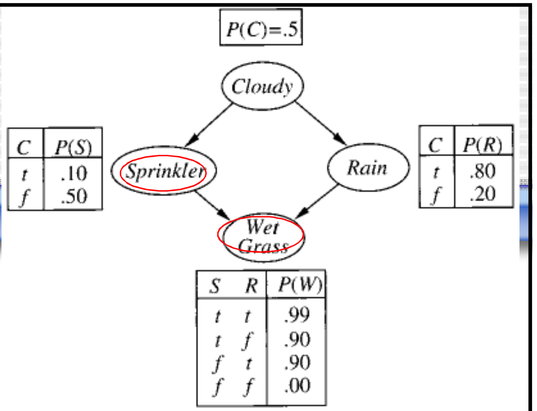
# Likelihood weighting

Idea: fix evidence variables, sample only nonevidence variables, and weight each sample by the likelihood it accords the evidence

```
function LIKELIHOOD-WEIGHTING($X$, $\mathbf{e}$, $bn$, $N$) returns an estimate of $P(X|\mathbf{e})$
    local variables: $\mathbf{W}$, a vector of weighted counts over $X$, initially zero

    for $j$ = 1 to $N$ do
        $\mathbf{x}$, $w \leftarrow$ WEIGHTED-SAMPLE($bn$ ,e)
        $\mathbf{W}[x] \leftarrow \mathbf{W}[x] + w$ where $x$ is the value of $X$ in $\mathbf{x}$
    return NORMALIZE($\mathbf{W}[X]$)
```

```
function WEIGHTED-SAMPLE($bn$, $\mathbf{e}$) returns an event and a weight

    $\mathbf{x} \leftarrow$ an event with $n$ elements; $w \leftarrow 1$
    for $i$ = 1 to $n$ do
        if $X_i$ has a value $x_i$ in $\mathbf{e}$
            then $w \leftarrow w \times P(X_i = x_i \mid parents(X_i))$
            else $x_i \leftarrow$ a random sample from $\mathbf{P}(X_i \mid parents(X_i))$
    return $\mathbf{x}$, $w$
```

# Likelihood Weighting *Example*

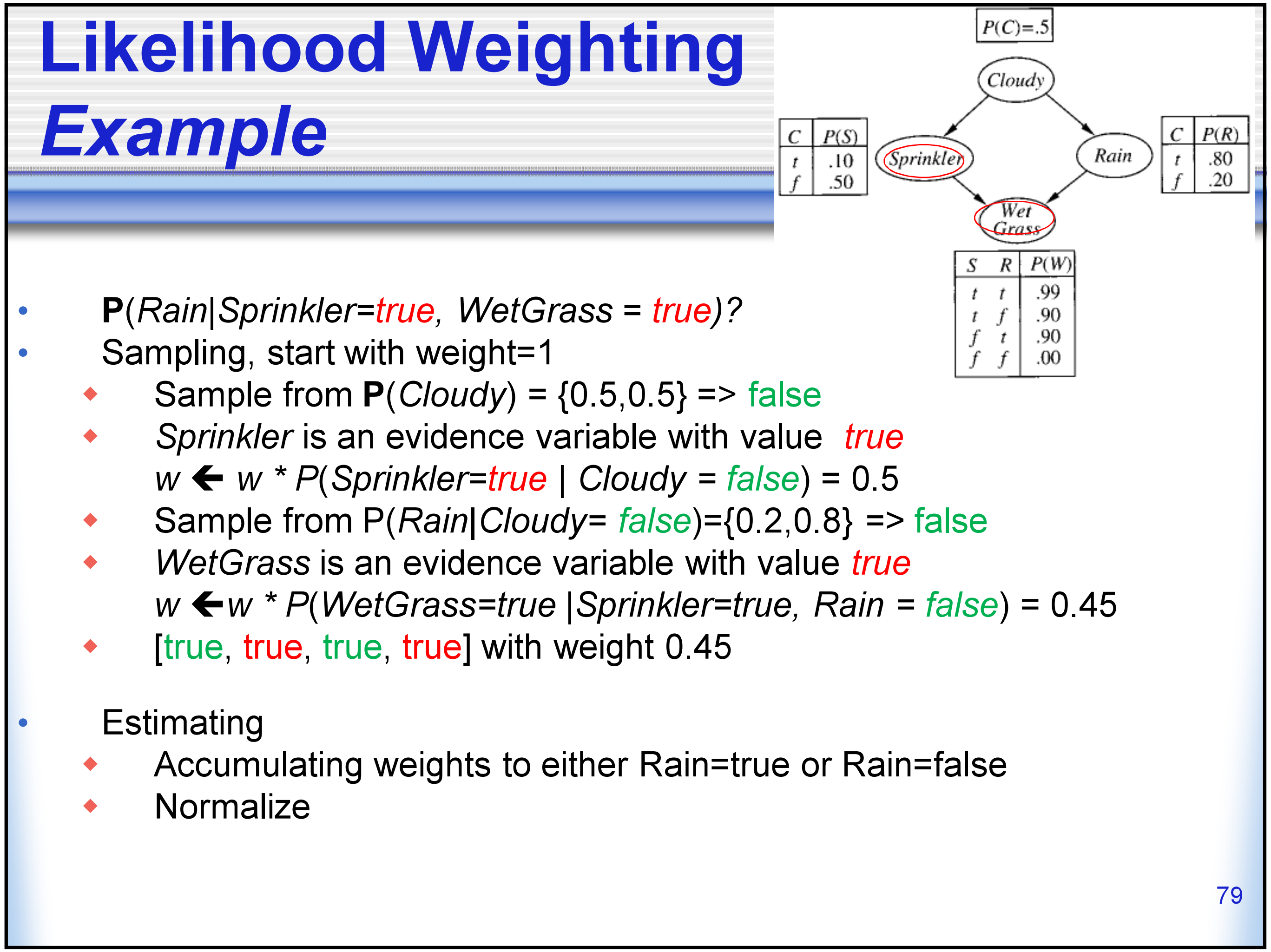


| C | P(S) |
|---|---|
| t | .10 |
| f | .50 |

| C | P(R) |
|---|---|
| t | .80 |
| f | .20 |

| S | R | P(W) |
|---|---|---|
| t | t | .99 |
| t | f | .90 |
| f | t | .90 |
| f | f | .00 |

- **P**(*Rain*|*Sprinkler*=*true*, *WetGrass* = *true*)?
- Sampling, start with weight=1
  - Sample from **P**(*Cloudy*) = {0.5,0.5} => true
  - *Sprinkler* is an evidence variable with value *true*
    *w* ← *w* * *P*(*Sprinkler*=*true* | *Cloudy* = *true*) = 0.1
  - Sample from P(*Rain*|*Cloudy*=*true*)={0.8,0.2} => *true*
  - *WetGrass* is an evidence variable with value *true*
    *w* ←*w* * *P*(*WetGrass*=*true* |*Sprinkler*=*true*, *Rain* = *true*) = 0.099
  - [true, true, true, true] with weight 0.099



# Likelihood Weighting *Example*

| P(C)=.5 |
|---|

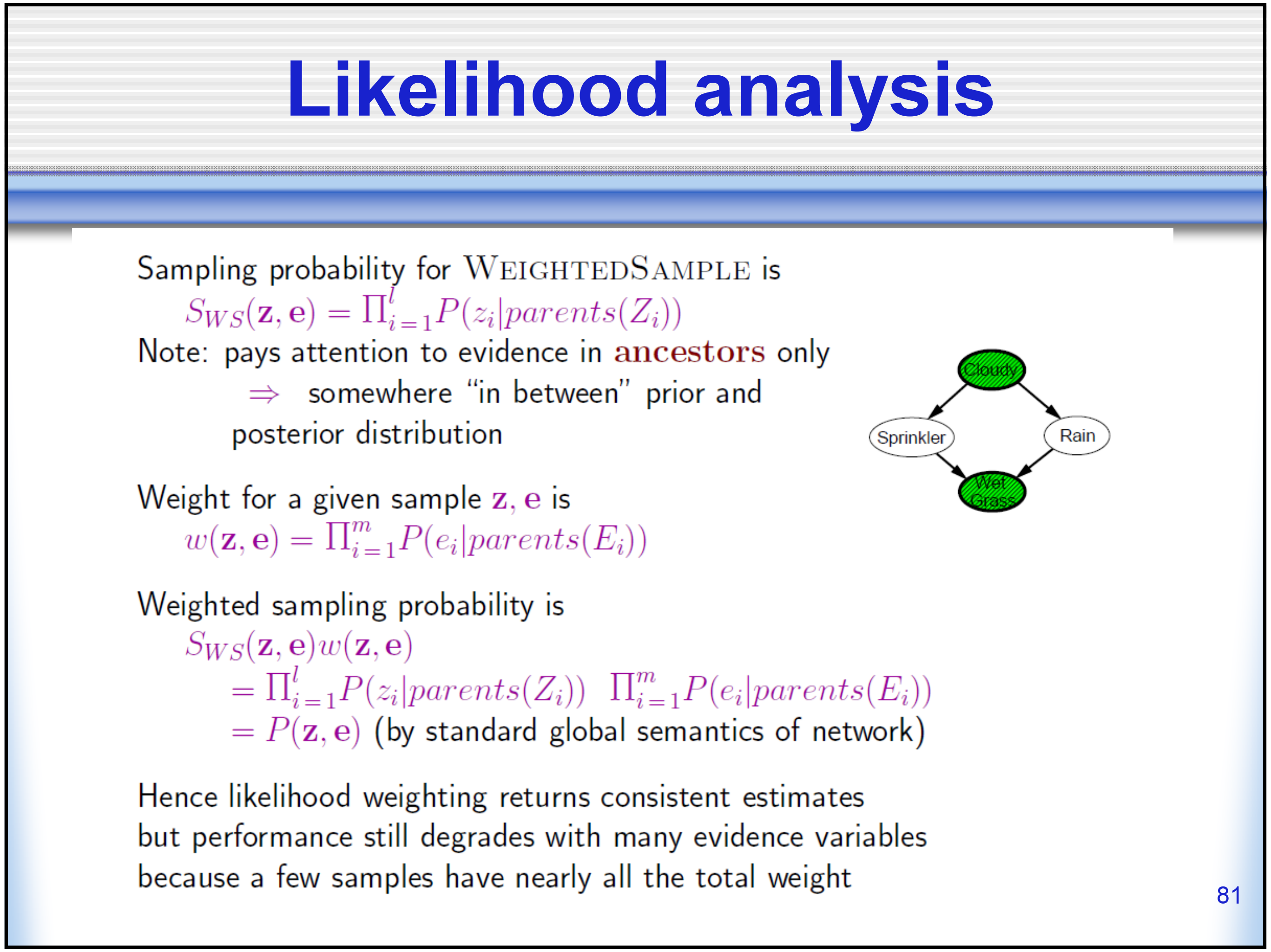
| C | P(S) |
|---|---|
| t | .10 |
| f | .50 |

| C | P(R) |
|---|---|
| t | .80 |
| f | .20 |

| S | R | P(W) |
|---|---|---|
| t | t | .99 |
| t | f | .90 |
| f | t | .90 |
| f | f | .00 |

- **P**(*Rain*|*Sprinkler*=*true*, *WetGrass* = *true*)?
- Sampling, start with weight=1
  - Sample from **P**(*Cloudy*) = {0.5,0.5} => false
  - *Sprinkler* is an evidence variable with value *true*
    *w* 🡐 *w* * *P*(*Sprinkler*=*true* | *Cloudy* = *false*) = 0.5
  - Sample from P(*Rain*|*Cloudy*= *false*)={0.2,0.8} => false
  - *WetGrass* is an evidence variable with value *true*
    *w* 🡐*w* * *P*(*WetGrass*=*true* |*Sprinkler*=*true*, *Rain* = *false*) = 0.45
  - [true, true, true, true] with weight 0.45

- Estimating
  - Accumulating weights to either Rain=true or Rain=false
  - Normalize



# Likelihood analysis

Sampling probability for WEIGHTEDSAMPLE is

$S_{WS}(\mathbf{z},\mathbf{e}) = \Pi_{i=1}^{l} P(z_i|parents(Z_i))$

Note: pays attention to evidence in **ancestors** only

$\Rightarrow$ somewhere "in between" prior and posterior distribution

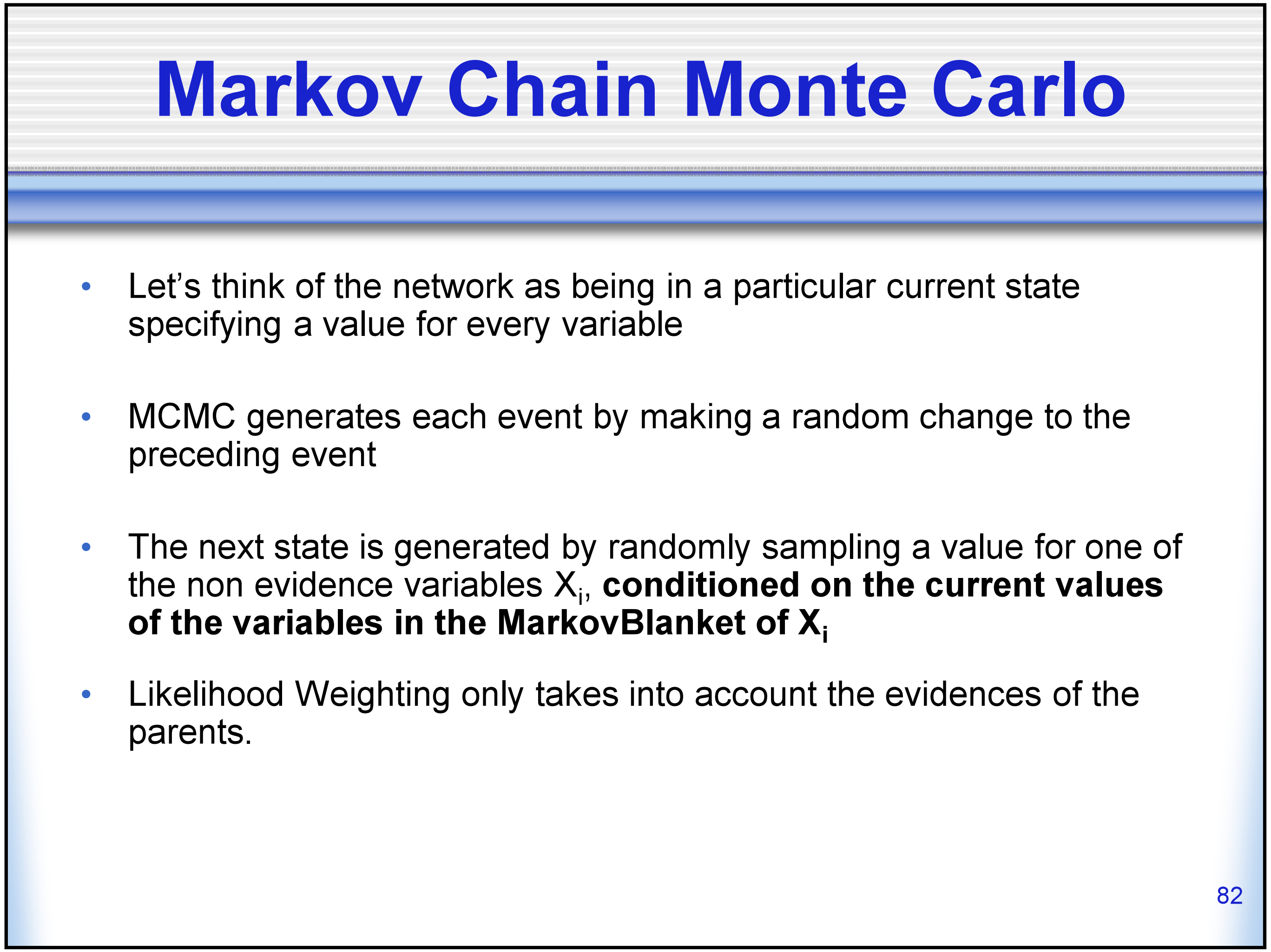
Weight for a given sample $\mathbf{z},\mathbf{e}$ is

$w(\mathbf{z},\mathbf{e}) = \Pi_{i=1}^{m} P(e_i|parents(E_i))$

Weighted sampling probability is

$$S_{WS}(\mathbf{z},\mathbf{e}) w(\mathbf{z},\mathbf{e})$$
$$= \Pi_{i=1}^{l} P(z_i|parents(Z_i)) \;\; \Pi_{i=1}^{m} P(e_i|parents(E_i))$$
$$= P(\mathbf{z},\mathbf{e}) \text{ (by standard global semantics of network)}$$

Hence likelihood weighting returns consistent estimates
but performance still degrades with many evidence variables
because a few samples have nearly all the total weight

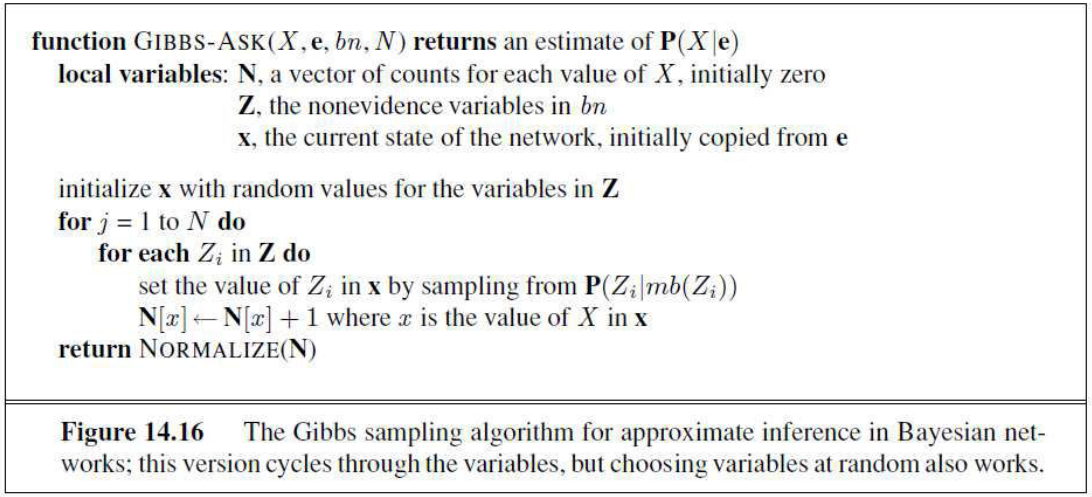


# Markov Chain Monte Carlo

- Let’s think of the network as being in a particular current state specifying a value for every variable
- MCMC generates each event by making a random change to the preceding event
- The next state is generated by randomly sampling a value for one of the non evidence variables $X_i$, **conditioned on the current values of the variables in the MarkovBlanket of $X_i$**
- Likelihood Weighting only takes into account the evidences of the parents.

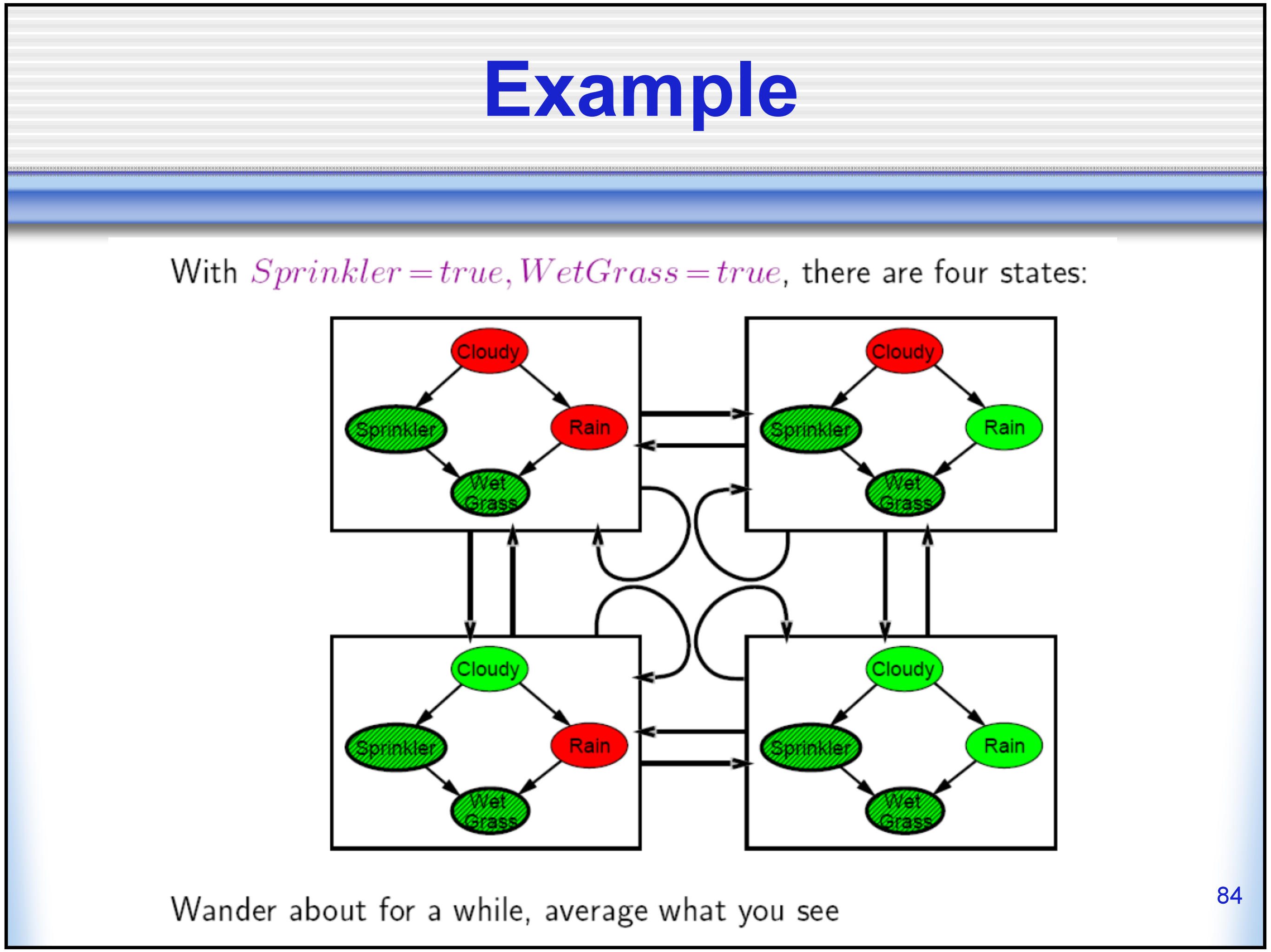
# Gibbs sampling

- Gibbs sampling is a MCMC method
  - State of the network => current assignment
  - Generate next state by sampling one non-evidence variable given Markov blanket
  - Sample each variable in turn ( can choose it random)



```
function GIBBS-ASK($X$, $\mathbf{e}$, $bn$, $N$) returns an estimate of $\mathbf{P}(X|\mathbf{e})$
  local variables: $\mathbf{N}$, a vector of counts for each value of $X$, initially zero
                   $\mathbf{Z}$, the nonevidence variables in $bn$
                   $\mathbf{x}$, the current state of the network, initially copied from $\mathbf{e}$

  initialize $\mathbf{x}$ with random values for the variables in $\mathbf{Z}$
  for $j$ = 1 to $N$ do
      for each $Z_i$ in $\mathbf{Z}$ do
          set the value of $Z_i$ in $\mathbf{x}$ by sampling from $\mathbf{P}(Z_i|mb(Z_i))$
          $\mathbf{N}[x] \leftarrow \mathbf{N}[x] + 1$ where $x$ is the value of $X$ in $\mathbf{x}$
  return NORMALIZE($\mathbf{N}$)
```

**Figure 14.16** The Gibbs sampling algorithm for approximate inference in Bayesian networks; this version cycles through the variables, but choosing variables at random also works.

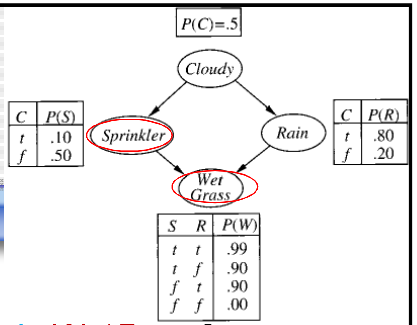


# Example

With $Sprinkler = true, WetGrass = true$, there are four states:

Wander about for a while, average what you see

# Gibbs *Example*

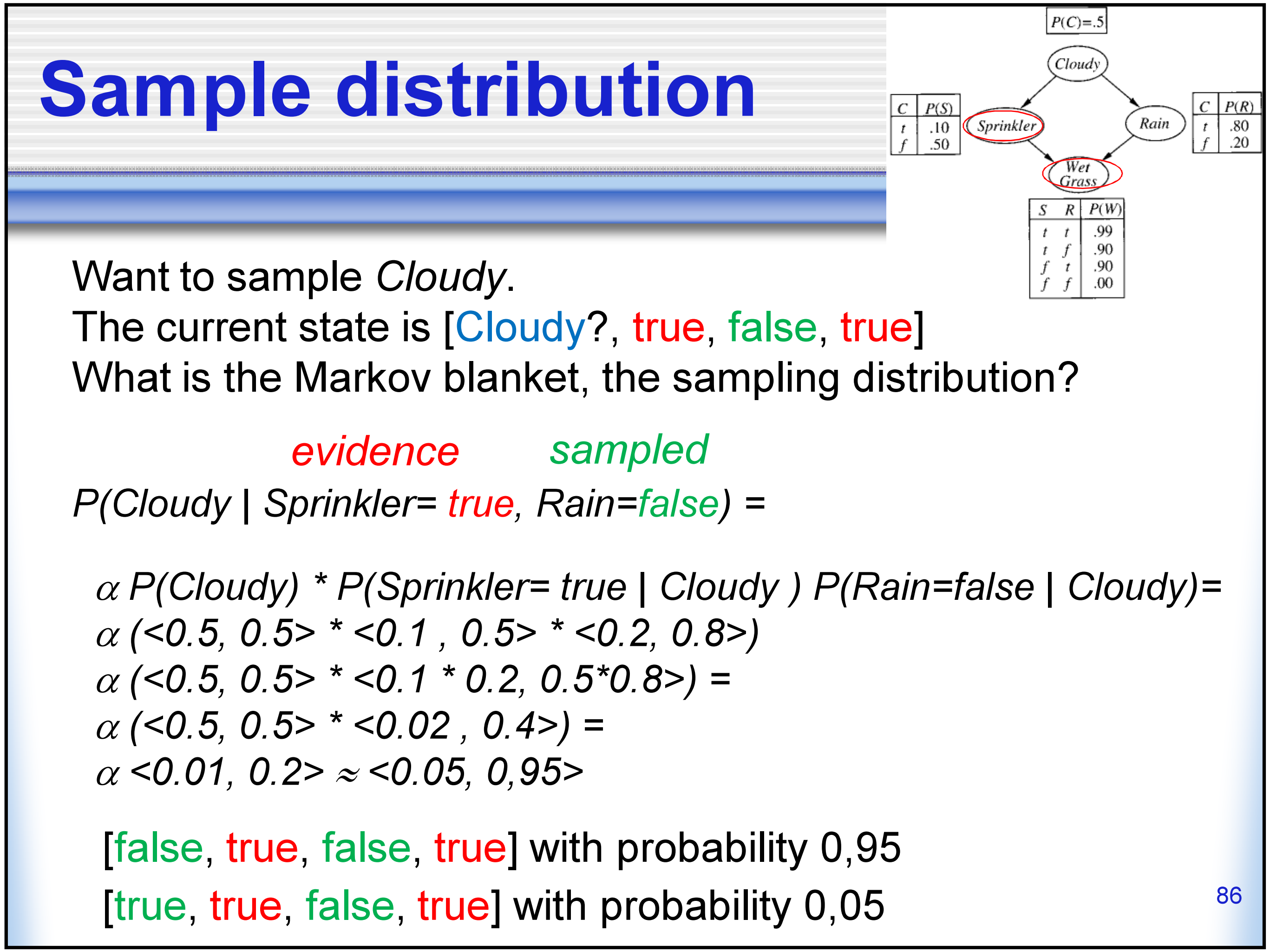


- Query P(Rain| Sprinkler = true, WetGrass = true)
- Initial state is [true, true, false, true] [Cloudy,Sprinkler,Rain,WetGrass]

- The following steps are executed repeatedly:
  - *Cloudy* is sampled, given the current values of its Markov Blanket variables
    So, we sample from *P(Cloudy|Sprinkler= true, Rain=false)*
    The result is Cloudy = false (???????)
  - Now current state is [false, true, false, true] and counts are updated

  - *Rain* is sampled, given the current values of its Markov Blanket variables
    Sample from *P(Rain|Cloudy=false,Sprinkler=true, WetGrass=true)*
    First create the distribution we want to sample from.
    →Rain = true.
  - Current state is [false, true, true, true]

- After all the iterations, let's say the process visited 20 states where rain is true and 60 states where rain is false then the answer of the query is NORMALIZE({20,60})={0.25,0.75}



# Sample distribution

| C | P(S) |
|---|---|
| t | .10 |
| f | .50 |

| C | P(R) |
|---|---|
| t | .80 |
| f | .20 |

| S | R | P(W) |
|---|---|---|
| t | t | .99 |
| t | f | .90 |
| f | t | .90 |
| f | f | .00 |

Want to sample *Cloudy*.
The current state is [Cloudy?, true, false, true]
What is the Markov blanket, the sampling distribution?

*evidence* *sampled*

*P(Cloudy | Sprinkler= true, Rain=false) =*

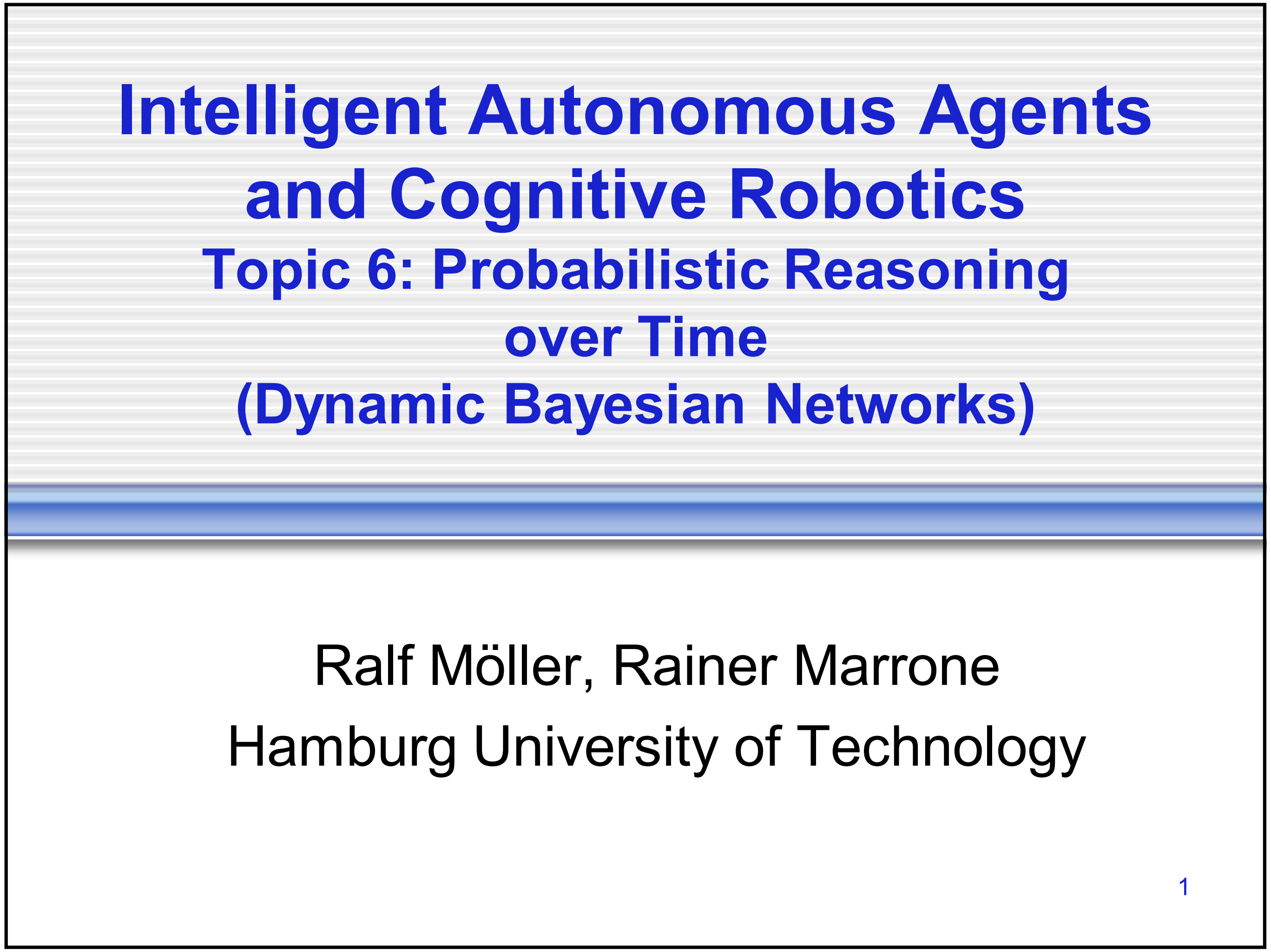
$\alpha$ *P(Cloudy) * P(Sprinkler= true | Cloudy ) P(Rain=false | Cloudy)=*
$\alpha$ *(<0.5, 0.5> * <0.1 , 0.5> * <0.2, 0.8>)*
$\alpha$ *(<0.5, 0.5> * <0.1 * 0.2, 0.5*0.8>) =*
$\alpha$ *(<0.5, 0.5> * <0.02 , 0.4>) =*
$\alpha$ *<0.01, 0.2>* $\approx$ *<0.05, 0,95>*

[false, true, false, true] with probability 0,95

[true, true, false, true] with probability 0,05

# Summary

- Bayesian networks provide a natural representation for (causally induced) conditional independence
- Topology + CPTs = compact representation of joint distribution
- Generally easy for domain experts to construct (if not to big)
- Exact inference by variable elimination
  - polytime on polytrees, NP-hard on general graphs
  - space can be exponential as well
- Approximate inference based on sampling and counting help to overcome complexity of exact inference



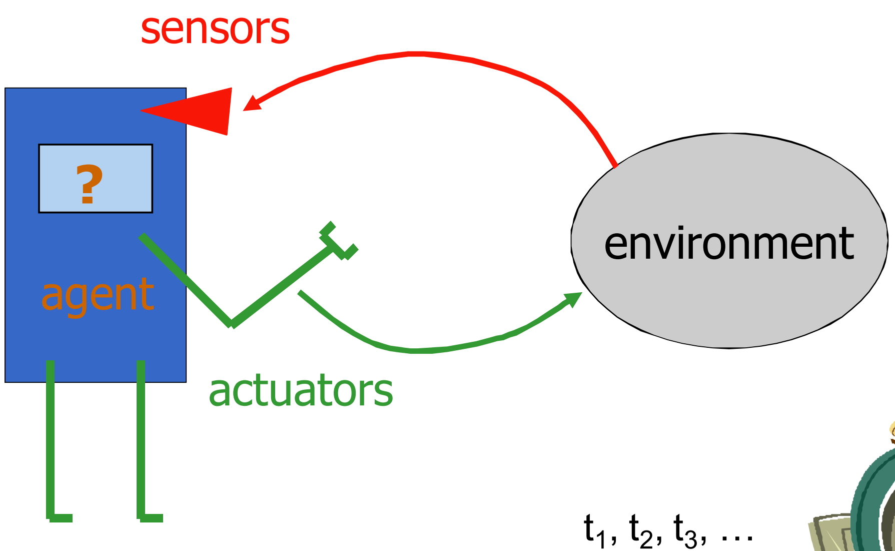
# Intelligent Autonomous Agents and Cognitive Robotics

## Topic 6: Probabilistic Reasoning over Time (Dynamic Bayesian Networks)

Ralf Möller, Rainer Marrone

Hamburg University of Technology

# Temporal Probabilistic Agent

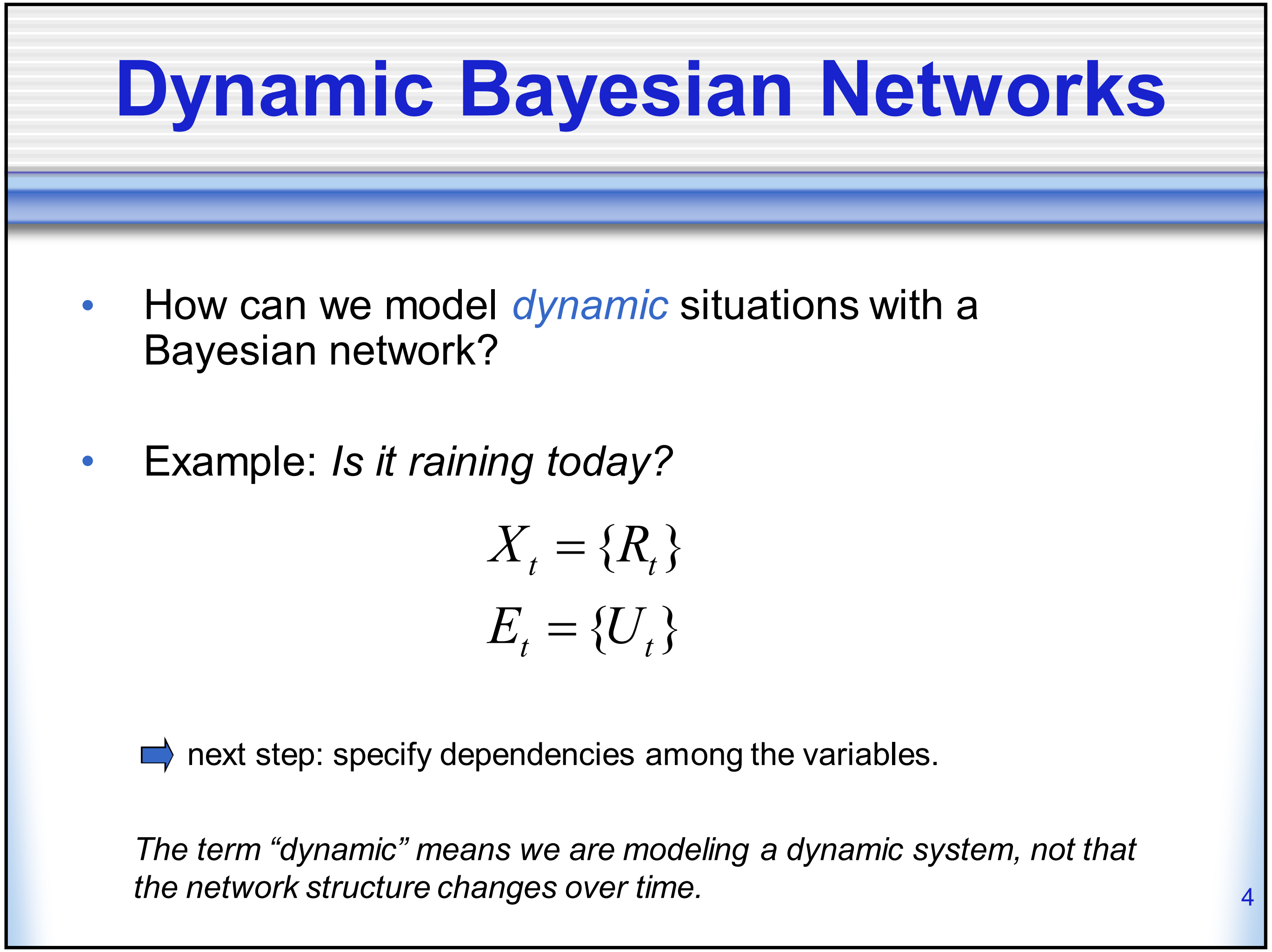


$t_1$, $t_2$, $t_3$, …

So far we only have taken care about one moment in time !!!!!!

# Time and Uncertainty

- The world changes over time, we need to track and predict it
- Examples:
  diabetes management, localization, speech recognition, …
- Basic idea: copy state and evidence variables for each time step
- **X**$_t$ – set of unobservable state variables at time t
  - e.g., BloodSugar$_t$, StomachContents$_t$, …
- **E**$_t$ – set of evidence variables at time t
  - e.g., MeasuredBloodSugar$_t$, PulseRate$_t$, FoodEaten$_t$ ,…
- Assumes discrete time steps



# Dynamic Bayesian Networks

- How can we model *dynamic* situations with a Bayesian network?

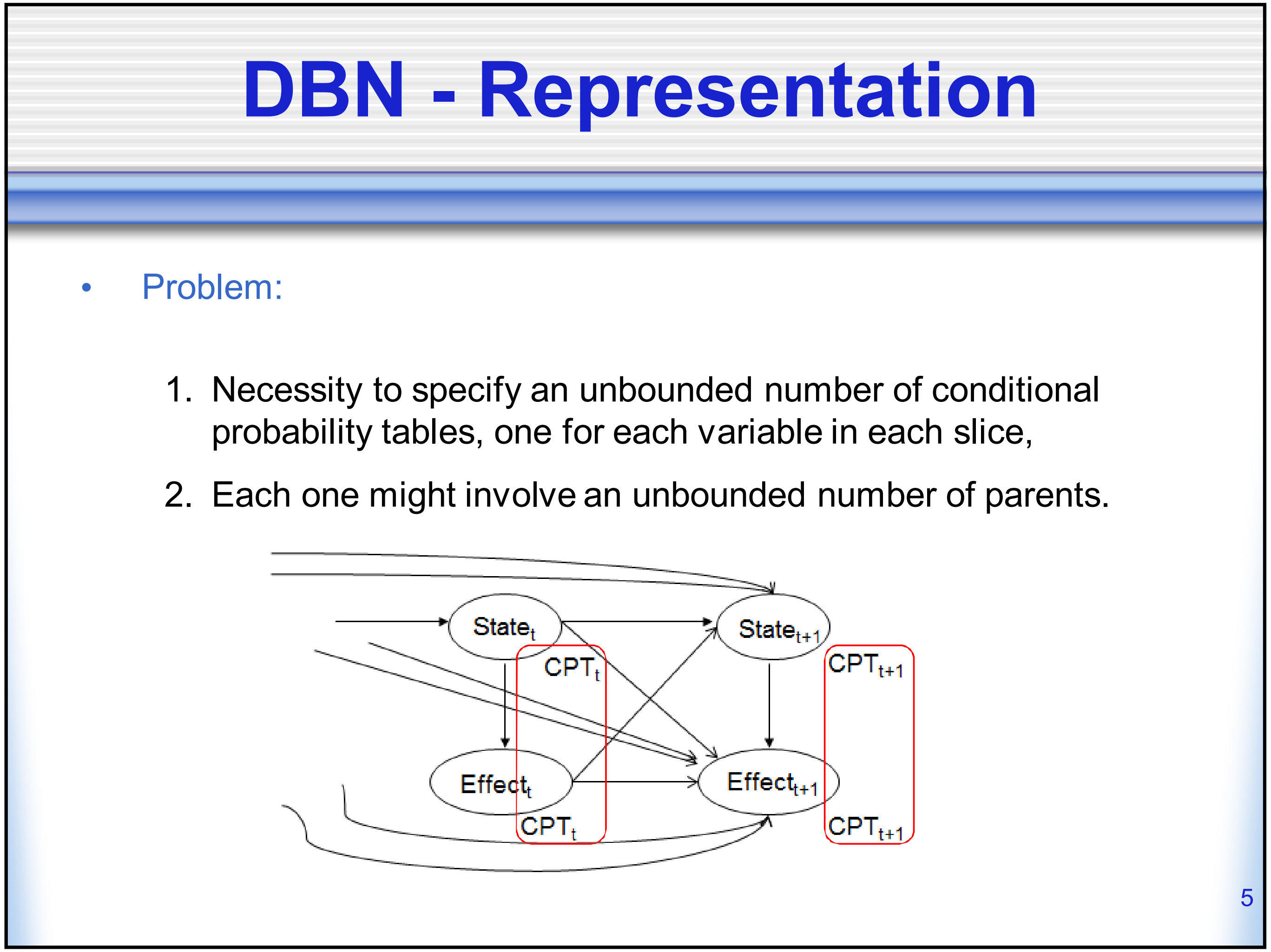
- Example: *Is it raining today?*

$$X_t = \{R_t\}$$
$$E_t = \{U_t\}$$

⇨ next step: specify dependencies among the variables.

*The term "dynamic" means we are modeling a dynamic system, not that the network structure changes over time.*

4

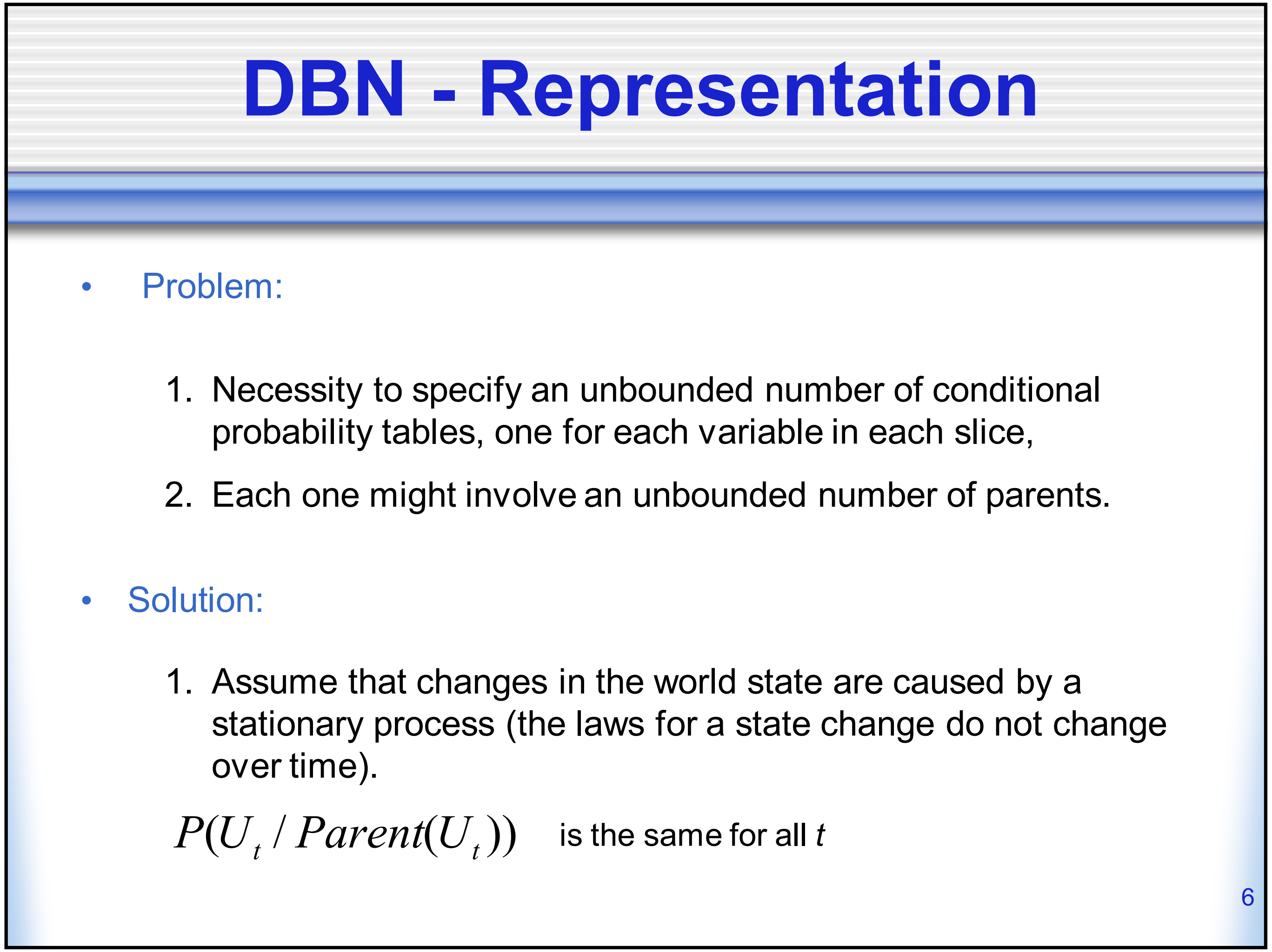


# DBN - Representation

- Problem:

1. Necessity to specify an unbounded number of conditional probability tables, one for each variable in each slice,
2. Each one might involve an unbounded number of parents.

$State_t$
$State_{t+1}$
$CPT_t$
$CPT_{t+1}$
$Effect_t$
$Effect_{t+1}$
$CPT_t$
$CPT_{t+1}$



# DBN - Representation

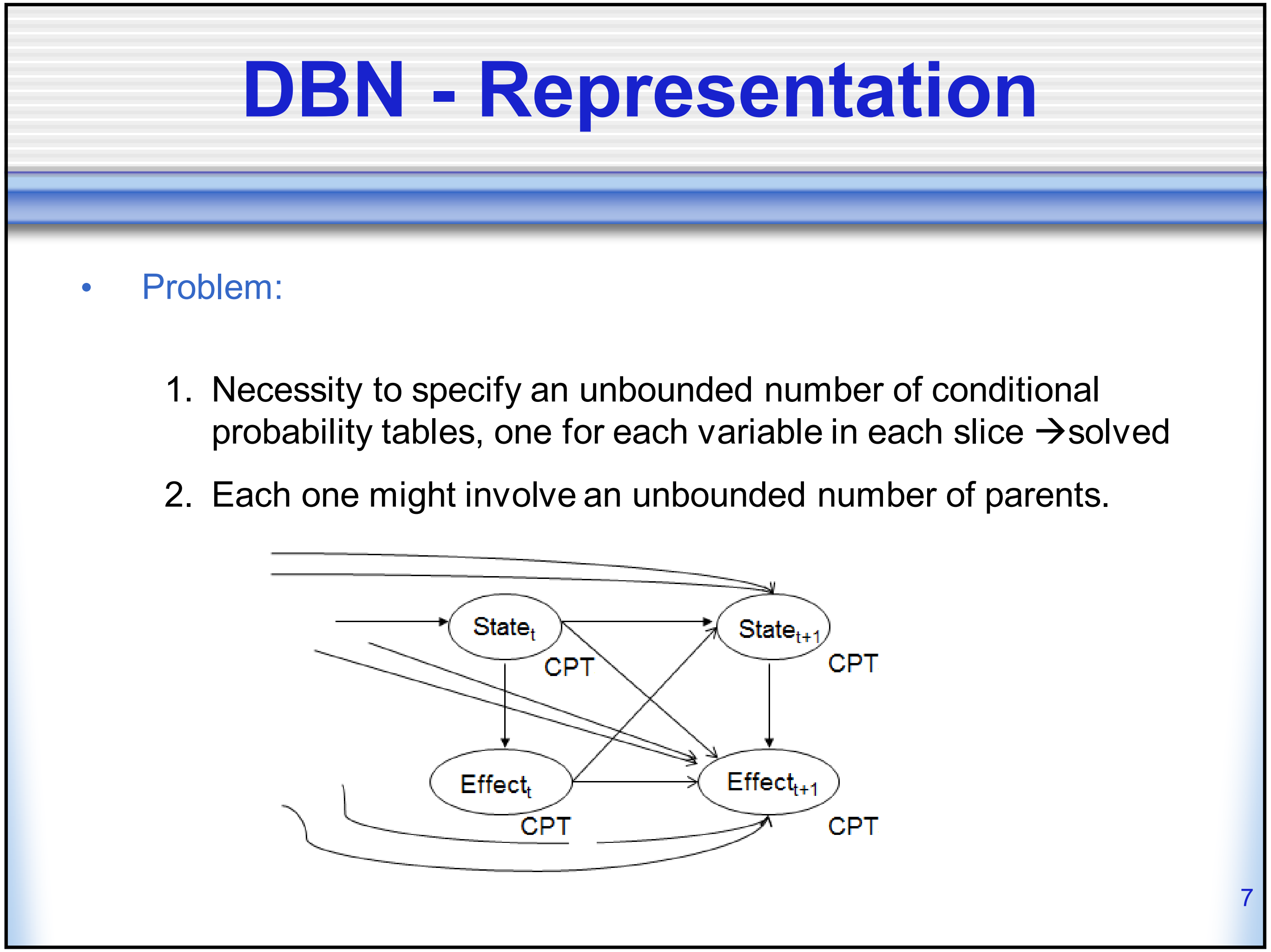
- Problem:

  1. Necessity to specify an unbounded number of conditional probability tables, one for each variable in each slice,
  2. Each one might involve an unbounded number of parents.

- Solution:

  1. Assume that changes in the world state are caused by a stationary process (the laws for a state change do not change over time).

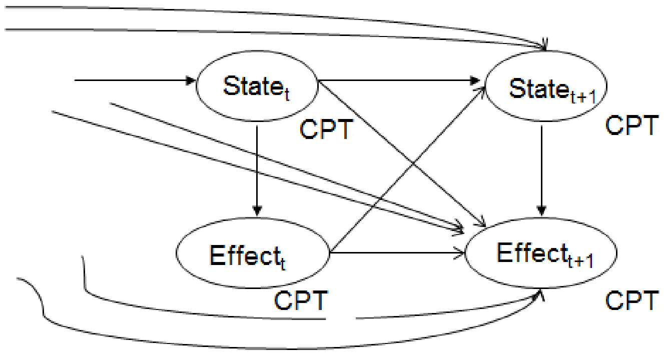
  $P(U_t / Parent(U_t))$ is the same for all *t*



# DBN - Representation

- Problem:

1. Necessity to specify an unbounded number of conditional probability tables, one for each variable in each slice →solved
2. Each one might involve an unbounded number of parents.



# DBN - Representation

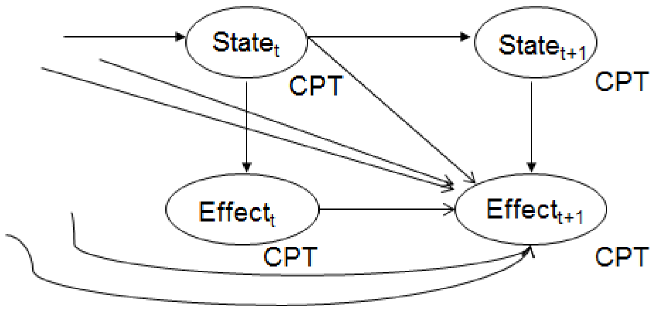
- Solution cont.:

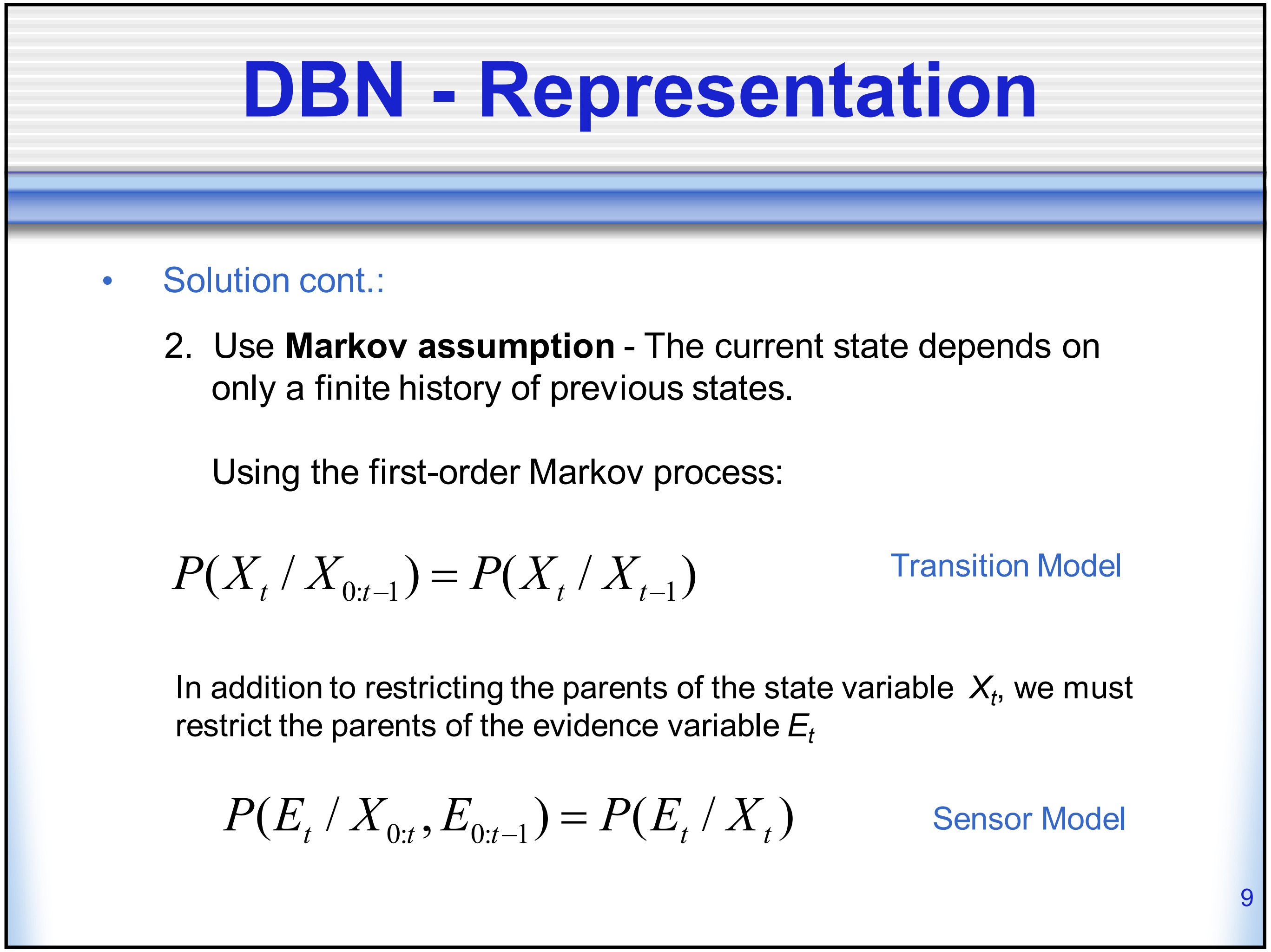
2. Use **Markov assumption** - The current state depends on only a finite history of previous states.

   Using the first-order Markov process:

$$P(X_t / X_{0:t-1}) = P(X_t / X_{t-1})$$

Transition Model





# DBN - Representation

- Solution cont.:

  2. Use **Markov assumption** - The current state depends on only a finite history of previous states.

     Using the first-order Markov process:

$$P(X_t / X_{0:t-1}) = P(X_t / X_{t-1})$$

Transition Model

In addition to restricting the parents of the state variable *X*$_t$, we must restrict the parents of the evidence variable *E*$_t$

$$P(E_t / X_{0:t}, E_{0:t-1}) = P(E_t / X_t)$$

Sensor Model

# DBN - Representation

- Solution cont.:

  2. Use **Markov assumption** - The current state depends on only in a finite history of previous states.

     Using the first-order Markov process:

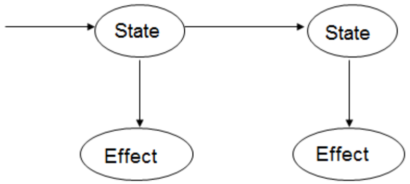
$$P(X_t / X_{0:t-1}) = P(X_t / X_{t-1})$$

Transition Model

In addition to restricting the parents of the state variable *X*$_t$, we must restrict the parents of the evidence variable *E*$_t$

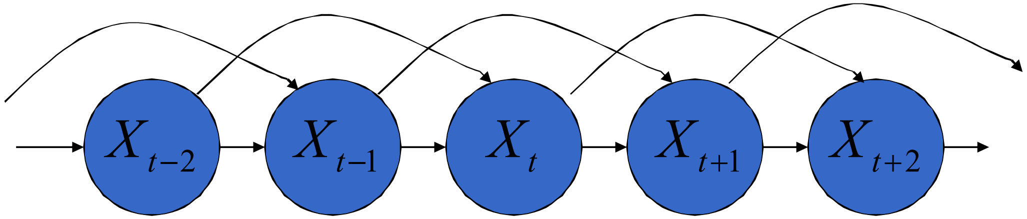
Sensor Model

$$P(E_t / X_{0:t}, E_{0:t-1}) = P(E_t / X_t)$$



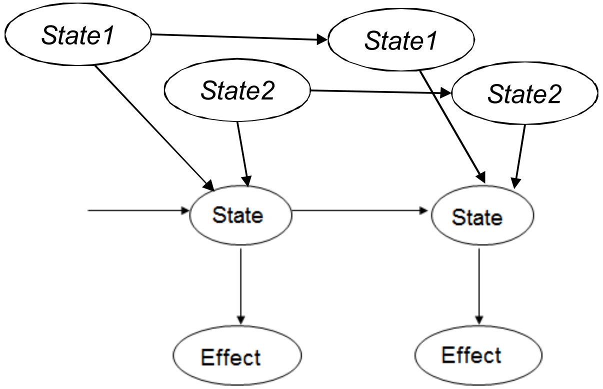
# Dynamic Bayesian Networks

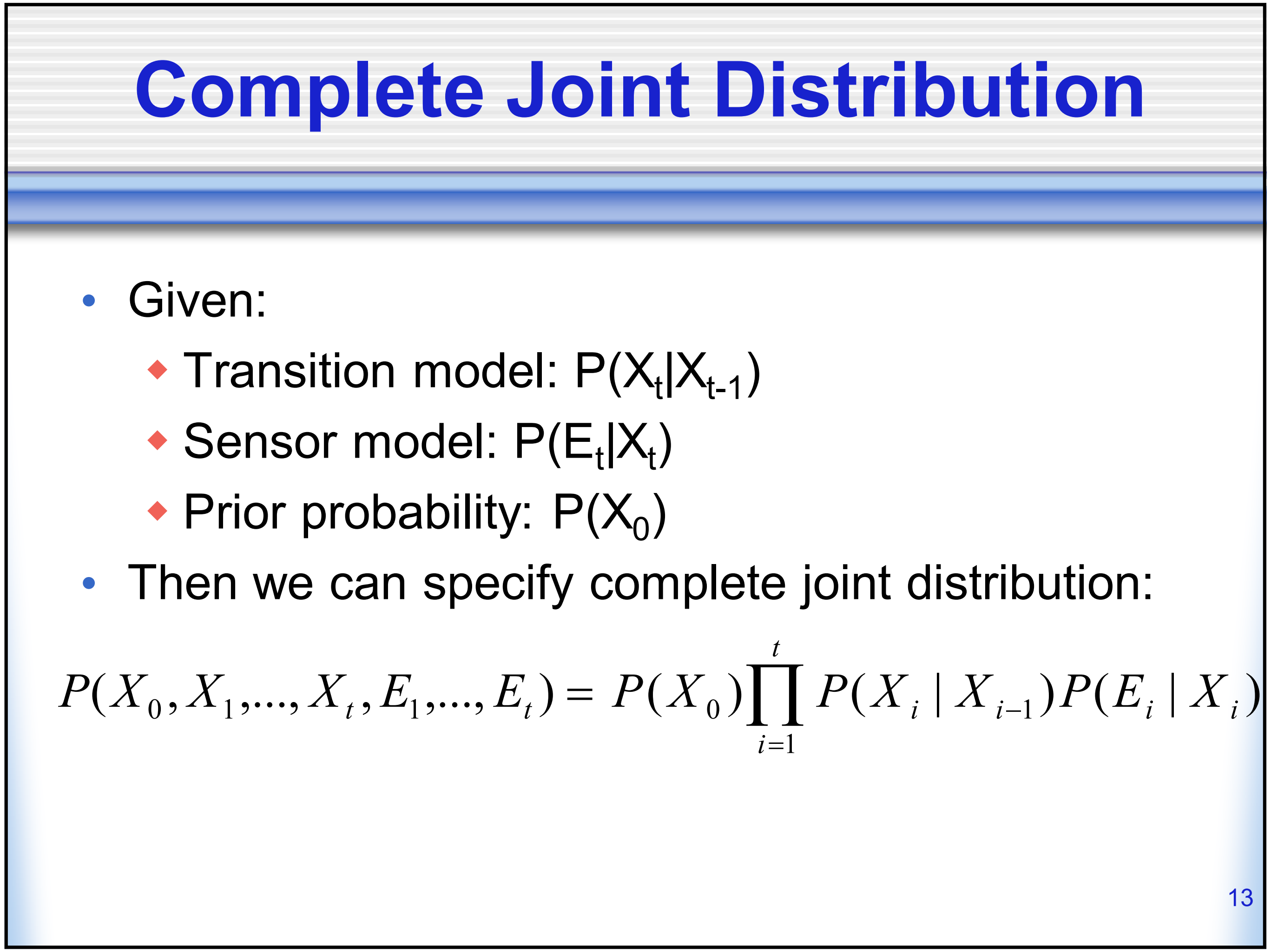
- There are two possible fixes if the approximation is too inaccurate:
  - Increasing the order of the Markov process model. For example, adding ***Rain***$_{t\text{-}2}$ as a parent of ***Rain***$_t$ , which might give slightly more accurate predictions.



# Dynamic Bayesian Networks

- There are two possible fixes if the approximation is too inaccurate:
  - Increasing the set of state variables. For example, adding ***Season$_t$*** to allow to incorporate historical records of rainy seasons, or adding ***Temprature$_t$*** , ***Humidity$_t$*** and ***Presssure$_t$*** to allow to use a physical model of rainy conditions.

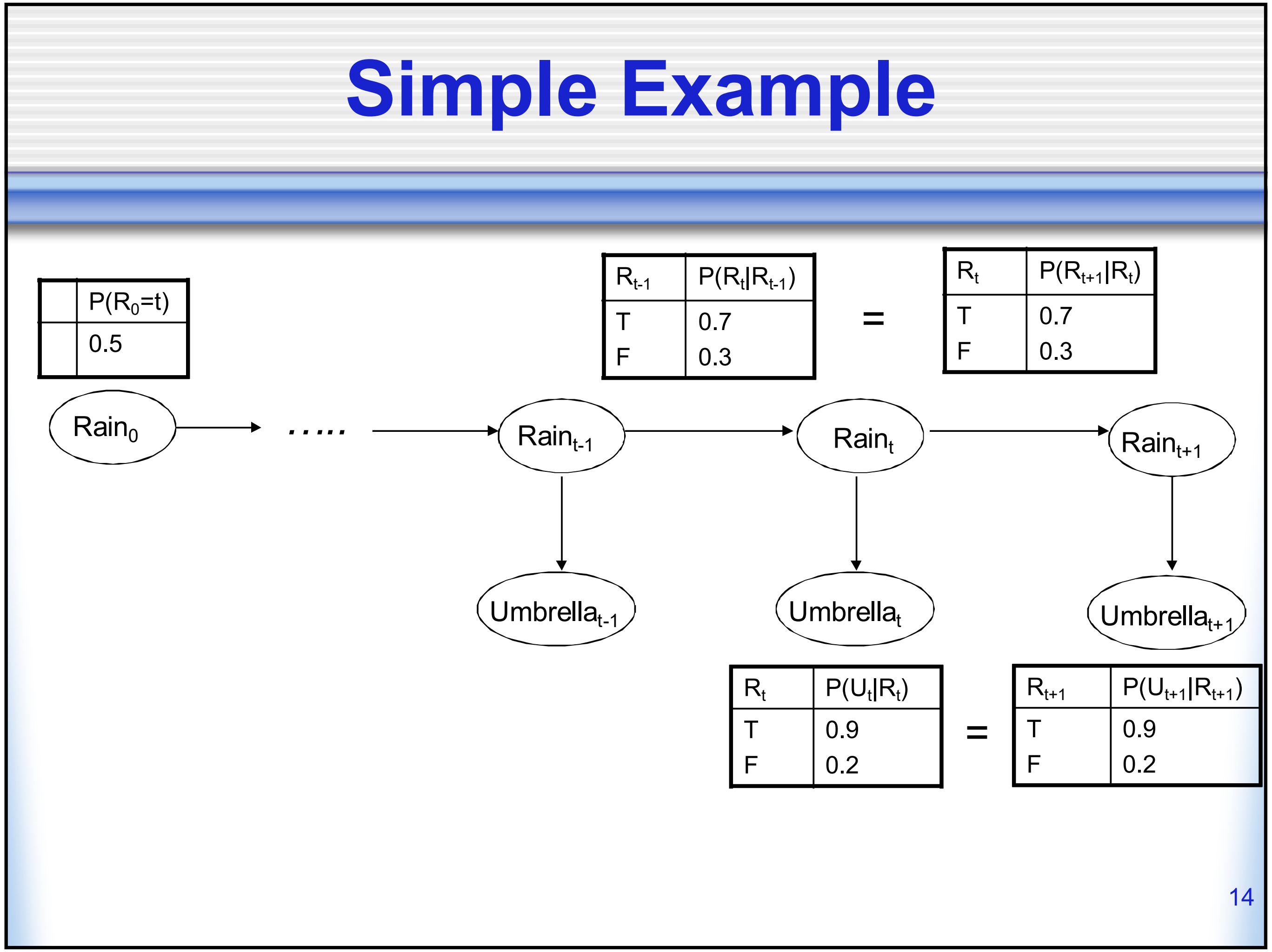




# Complete Joint Distribution

- Given:
  - Transition model: $P(X_t|X_{t-1})$
  - Sensor model: $P(E_t|X_t)$
  - Prior probability: $P(X_0)$
- Then we can specify complete joint distribution:

$$P(X_0, X_1, \ldots, X_t, E_1, \ldots, E_t) = P(X_0)\prod_{i=1}^{t} P(X_i \mid X_{i-1})P(E_i \mid X_i)$$



# Simple Example

|  | $P(R_0=t)$ |
|---|---|
|  | 0.5 |

| $R_{t-1}$ | $P(R_t \mid R_{t-1})$ |
|---|---|
| T | 0.7 |
| F | 0.3 |

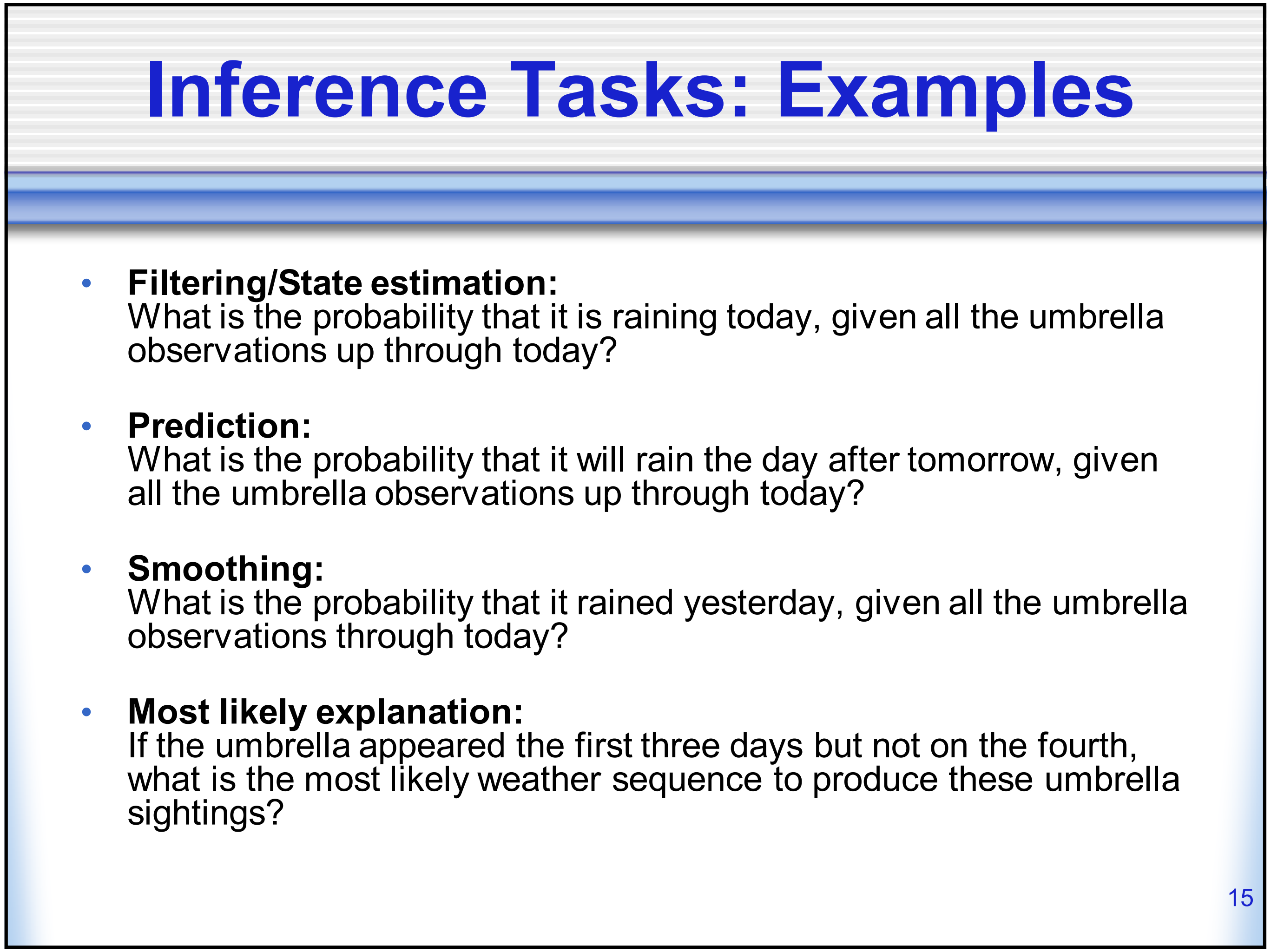
=

| $R_t$ | $P(R_{t+1} \mid R_t)$ |
|---|---|
| T | 0.7 |
| F | 0.3 |

| $R_t$ | $P(U_t \mid R_t)$ |
|---|---|
| T | 0.9 |
| F | 0.2 |

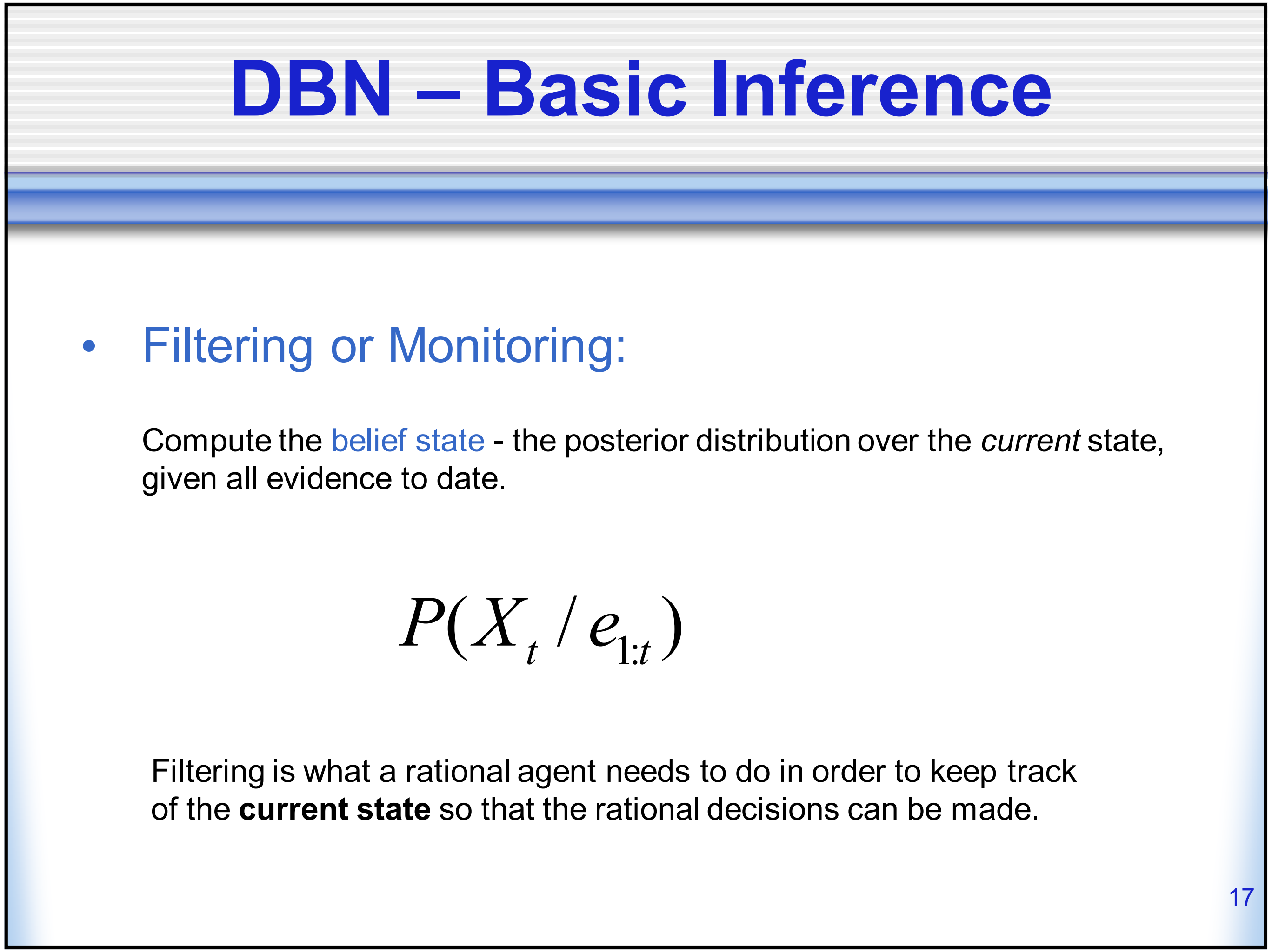
=

| $R_{t+1}$ | $P(U_{t+1} \mid R_{t+1})$ |
|---|---|
| T | 0.9 |
| F | 0.2 |



# Inference Tasks: Examples

- **Filtering/State estimation:**
  What is the probability that it is raining today, given all the umbrella observations up through today?
- **Prediction:**
  What is the probability that it will rain the day after tomorrow, given all the umbrella observations up through today?
- **Smoothing:**
  What is the probability that it rained yesterday, given all the umbrella observations through today?
- **Most likely explanation:**
  If the umbrella appeared the first three days but not on the fourth, what is the most likely weather sequence to produce these umbrella sightings?



# DBN – Basic Inference

- Filtering or Monitoring:

Compute the belief state - the posterior distribution over the *current* state, given all evidence to date.

$$P(X_t / e_{1:t})$$

Filtering is what a rational agent needs to do in order to keep track of the **current state** so that the rational decisions can be made.

# DBN – Basic Inference

- Filtering cont.

$P(B|A,C) = \alpha\, P(A|B,C)\, P(B|C)$

Given the results of filtering up to time *t*, one can easily compute the result for *t+1* from the new evidence $e_{t+1}$

$$\boxed{P(X_{t+1} / e_{1:t+1}) = f(e_{t+1,} P(X_t / e_{1:t}))}$$

(seeking for some recursive function *f* ?)

$$= P(X_{t+1} / e_{1:t,} e_{t+1})$$

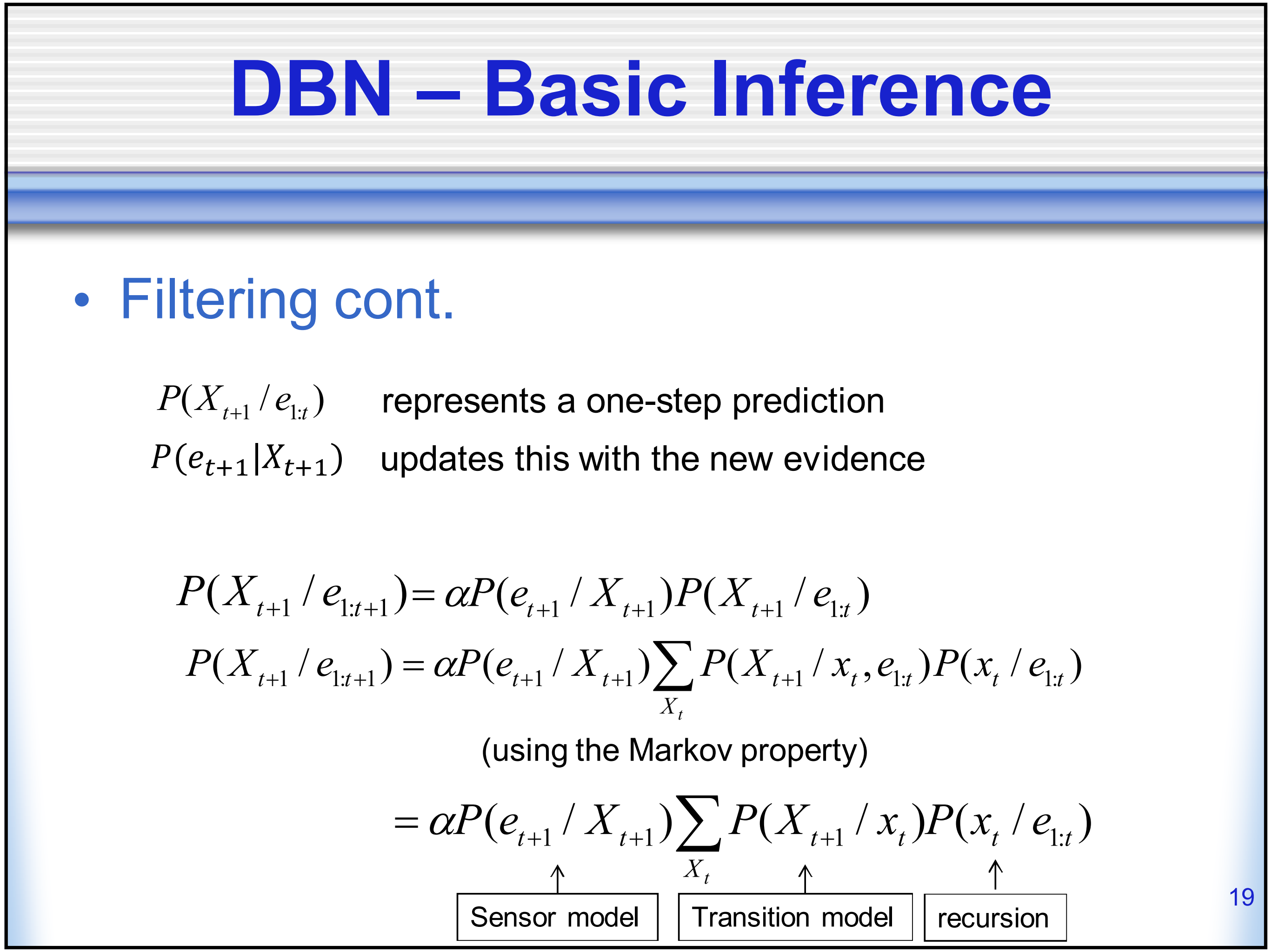
(dividing up the evidence)

$$= \alpha P(e_{t+1} / X_{t+1,} e_{1:t}) P(X_{t+1} / e_{1:t})$$

(using Bayes' Theorem)

$$= \alpha P(e_{t+1} / X_{t+1}) P(X_{t+1} / e_{1:t})$$

(by the Markov property of evidence)



# DBN – Basic Inference

- Filtering cont.

$P(X_{t+1}/e_{1:t})$ represents a one-step prediction

$P(e_{t+1}|X_{t+1})$ updates this with the new evidence

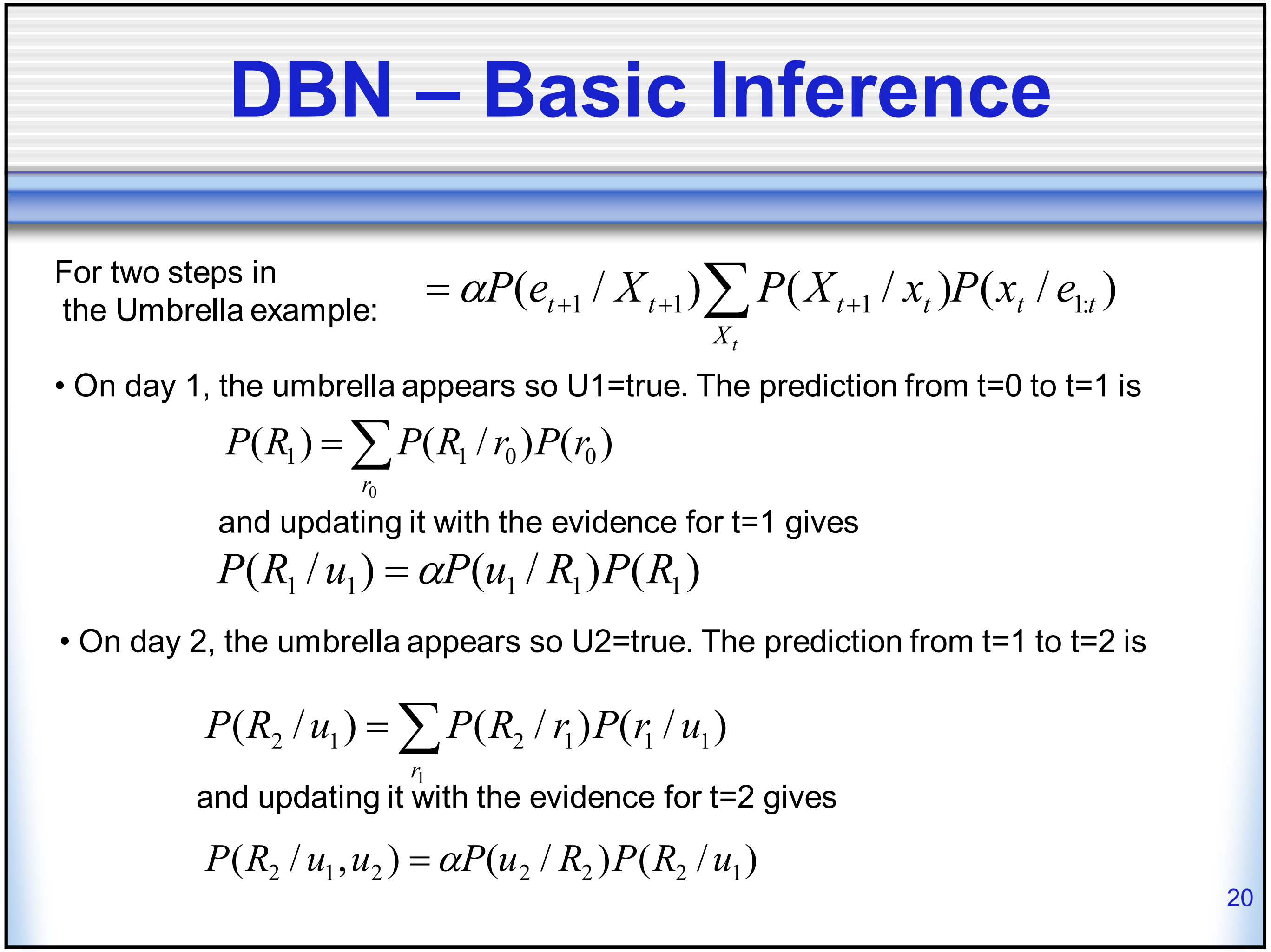
$$P(X_{t+1}/e_{1:t+1}) = \alpha P(e_{t+1}/X_{t+1}) P(X_{t+1}/e_{1:t})$$

$$P(X_{t+1}/e_{1:t+1}) = \alpha P(e_{t+1}/X_{t+1}) \sum_{X_t} P(X_{t+1}/x_t, e_{1:t}) P(x_t/e_{1:t})$$

(using the Markov property)

$$= \alpha \underset{\substack{\uparrow\\ \boxed{\text{Sensor model}}}}{P(e_{t+1}/X_{t+1})} \sum_{X_t} \underset{\substack{\uparrow\\ \boxed{\text{Transition model}}}}{P(X_{t+1}/x_t)} \underset{\substack{\uparrow\\ \boxed{\text{recursion}}}}{P(x_t/e_{1:t})}$$

19



# DBN – Basic Inference

For two steps in
the Umbrella example:

$$= \alpha P(e_{t+1} / X_{t+1}) \sum_{X_t} P(X_{t+1} / x_t) P(x_t / e_{1:t})$$

- On day 1, the umbrella appears so U1=true. The prediction from t=0 to t=1 is

$$P(R_1) = \sum_{r_0} P(R_1 / r_0) P(r_0)$$

and updating it with the evidence for t=1 gives

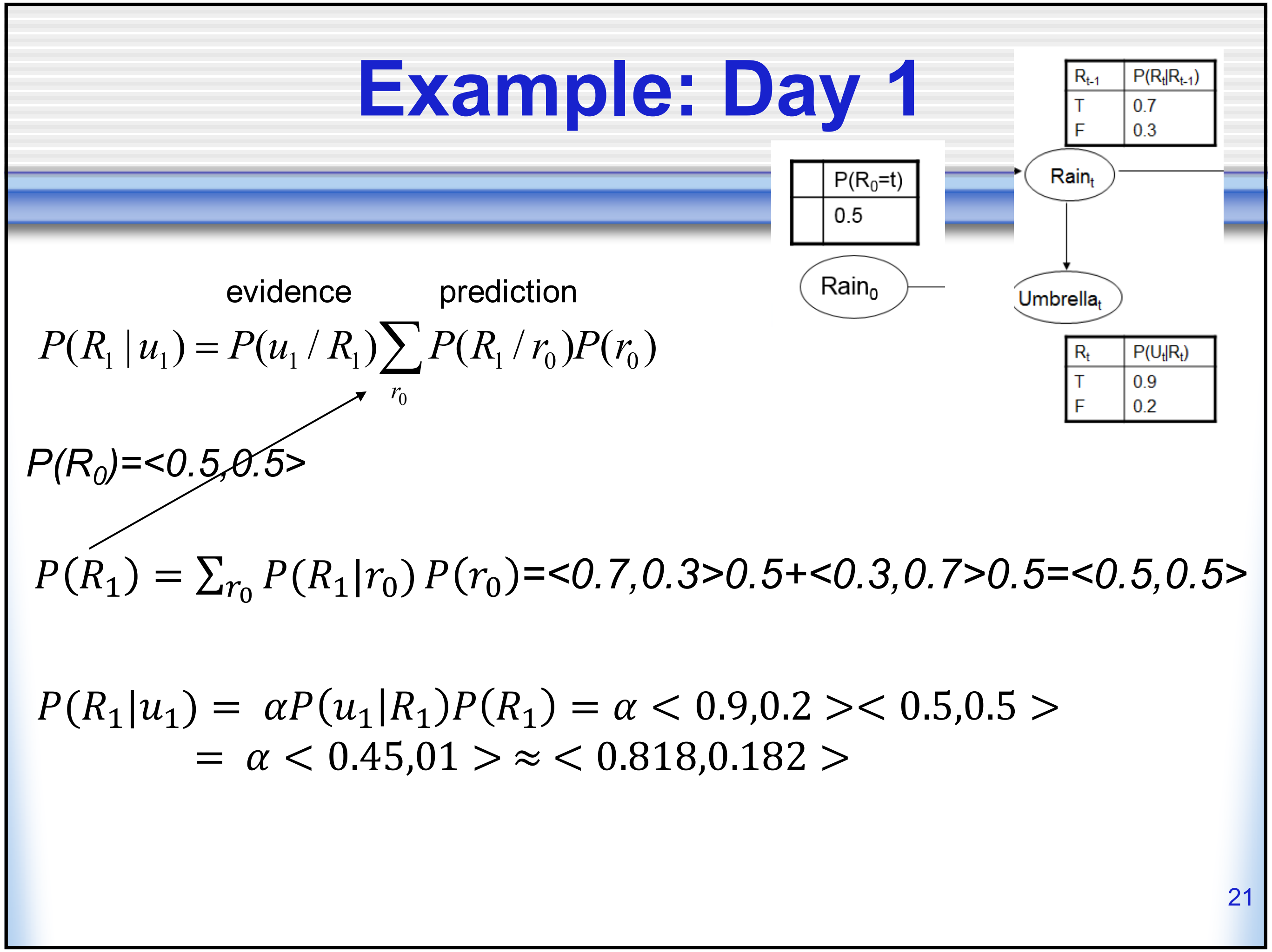
$$P(R_1 / u_1) = \alpha P(u_1 / R_1) P(R_1)$$

- On day 2, the umbrella appears so U2=true. The prediction from t=1 to t=2 is

$$P(R_2 / u_1) = \sum_{r_1} P(R_2 / r_1) P(r_1 / u_1)$$

and updating it with the evidence for t=2 gives

$$P(R_2 / u_1, u_2) = \alpha P(u_2 / R_2) P(R_2 / u_1)$$



# Example: Day 1

| | $P(R_0=t)$ |
|---|---|
| | 0.5 |

| $R_{t-1}$ | $P(R_t \mid R_{t-1})$ |
|---|---|
| T | 0.7 |
| F | 0.3 |

| $R_t$ | $P(U_t \mid R_t)$ |
|---|---|
| T | 0.9 |
| F | 0.2 |

evidence prediction

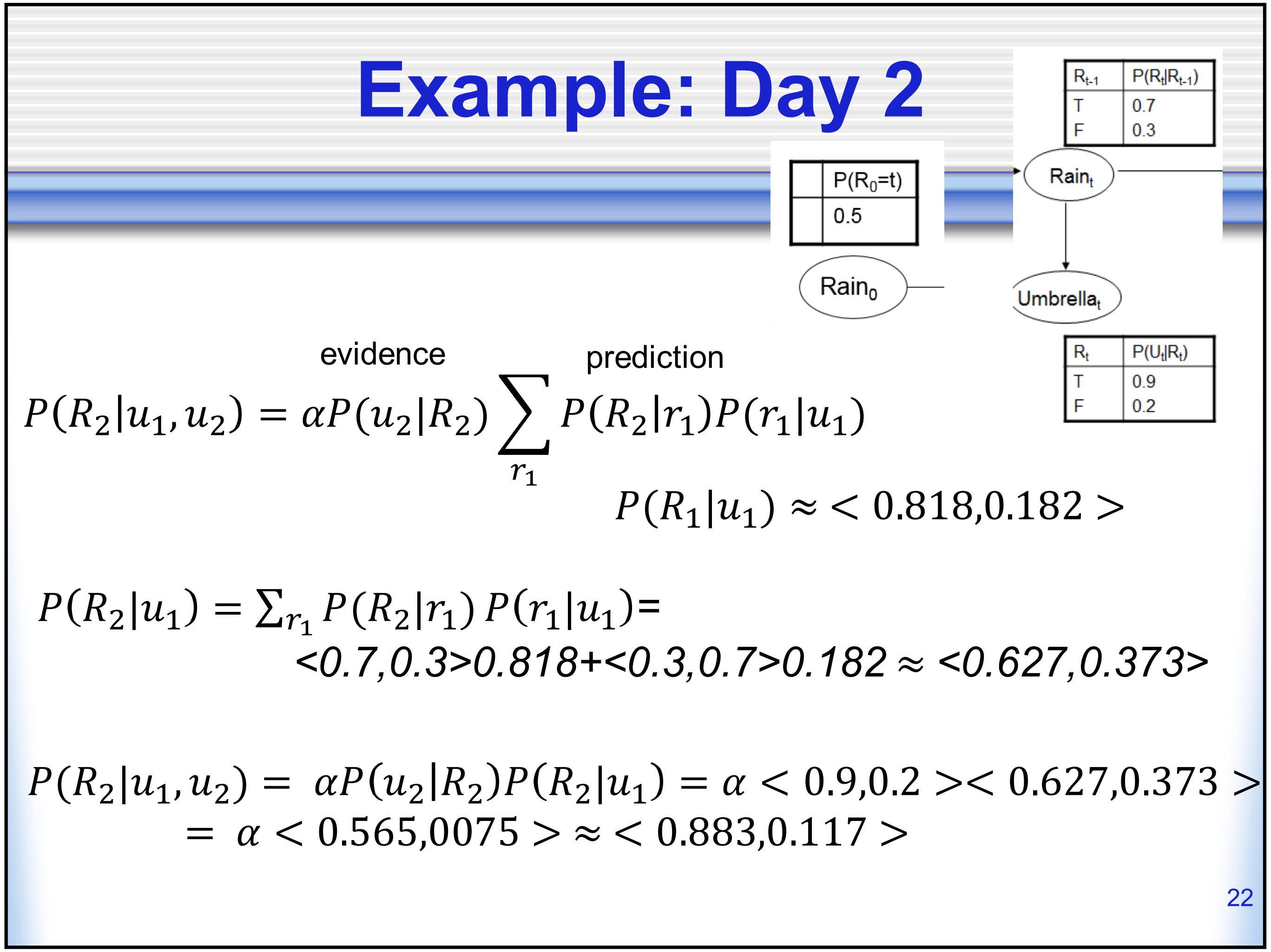
$$P(R_1 | u_1) = P(u_1 / R_1) \sum_{r_0} P(R_1 / r_0) P(r_0)$$

$P(R_0)$=<0.5,0.5>

$P(R_1) = \Sigma_{r_0} P(R_1|r_0)\, P(r_0)$=<0.7,0.3>0.5+<0.3,0.7>0.5=<0.5,0.5>

$$P(R_1|u_1) = \alpha P(u_1|R_1)P(R_1) = \alpha < 0.9,0.2 >< 0.5,0.5 >$$
$$= \alpha < 0.45,01 > \approx < 0.818,0.182 >$$

27.11.2022



# Example: Day 2

|  | $P(R_0=t)$ |
|---|---|
|  | 0.5 |

| $R_{t-1}$ | $P(R_t\|R_{t-1})$ |
|---|---|
| T | 0.7 |
| F | 0.3 |

$\text{Rain}_0$
$\text{Rain}_t$
$\text{Umbrella}_t$

| $R_t$ | $P(U_t\|R_t)$ |
|---|---|
| T | 0.9 |
| F | 0.2 |

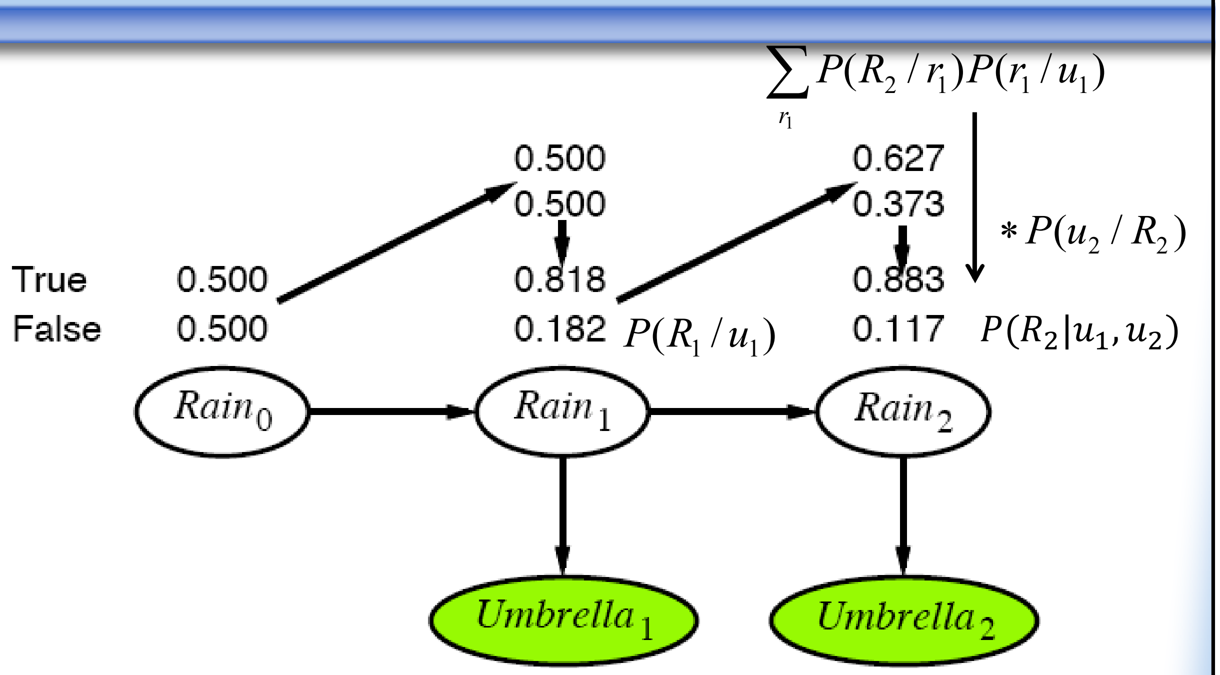
$$P(R_2|u_1,u_2) = \alpha \overset{\text{evidence}}{P(u_2|R_2)} \sum_{r_1} \overset{\text{prediction}}{P(R_2|r_1)P(r_1|u_1)}$$

$$P(R_1|u_1) \approx < 0.818{,}0.182 >$$

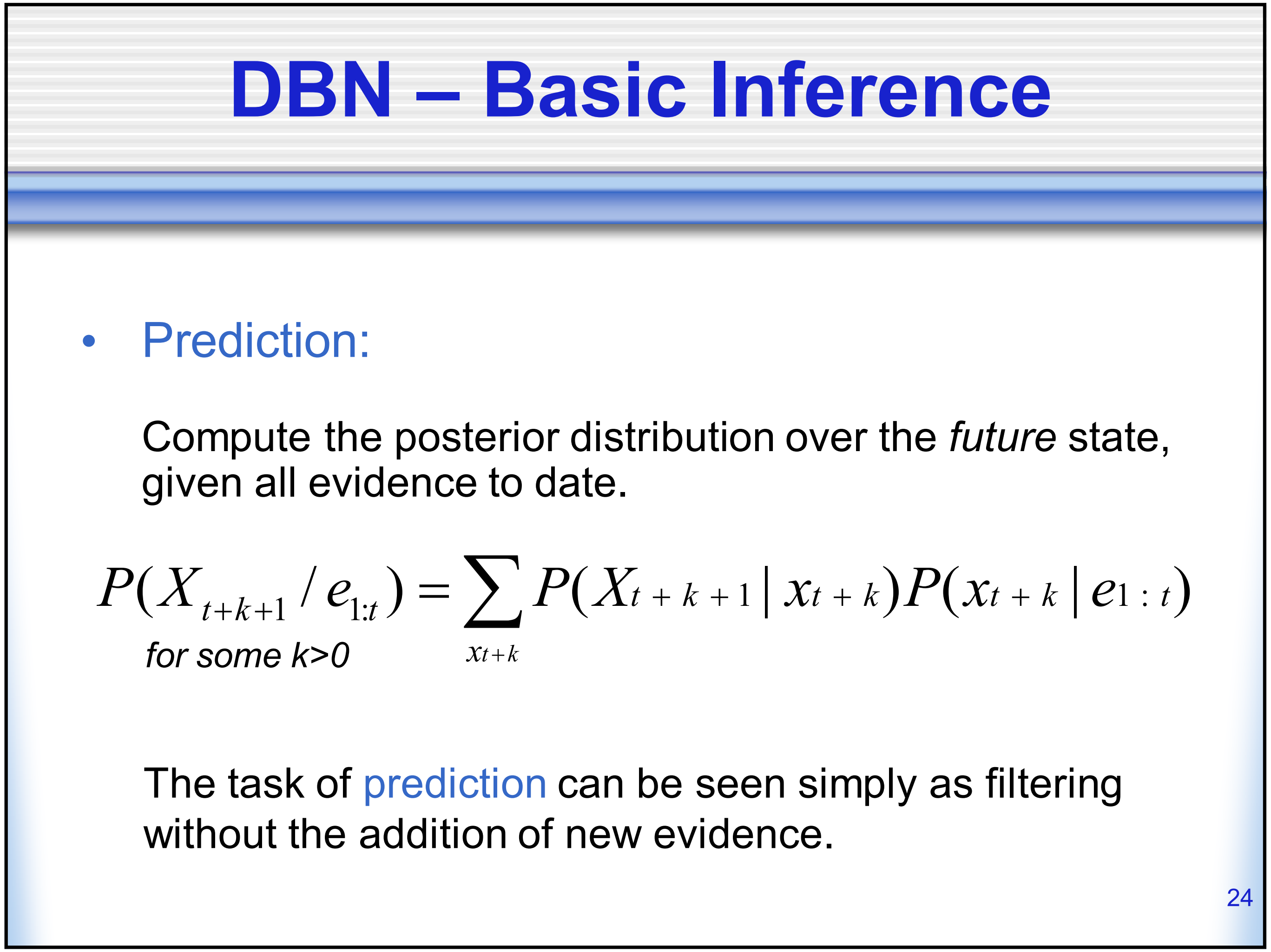
$$P(R_2|u_1) = \textstyle\sum_{r_1} P(R_2|r_1)\, P(r_1|u_1) =$$
$$<0.7,0.3>0.818+<0.3,0.7>0.182 \approx <0.627,0.373>$$

$$P(R_2|u_1,u_2) = \alpha P(u_2|R_2)P(R_2|u_1) = \alpha < 0.9{,}0.2 >< 0.627{,}0.373 >$$
$$= \alpha < 0.565{,}0075 > \approx < 0.883{,}0.117 >$$

# Example



$\sum_{r_1} P(R_2 / r_1)P(r_1 / u_1)$
$* P(u_2 / R_2)$
True
False
0.500
0.500
0.500
0.500
0.818
0.182
$P(R_1 / u_1)$
0.627
0.373
0.883
0.117
$P(R_2|u_1,u_2)$
$Rain_0$
$Rain_1$
$Rain_2$
$Umbrella_1$
$Umbrella_2$



# DBN – Basic Inference

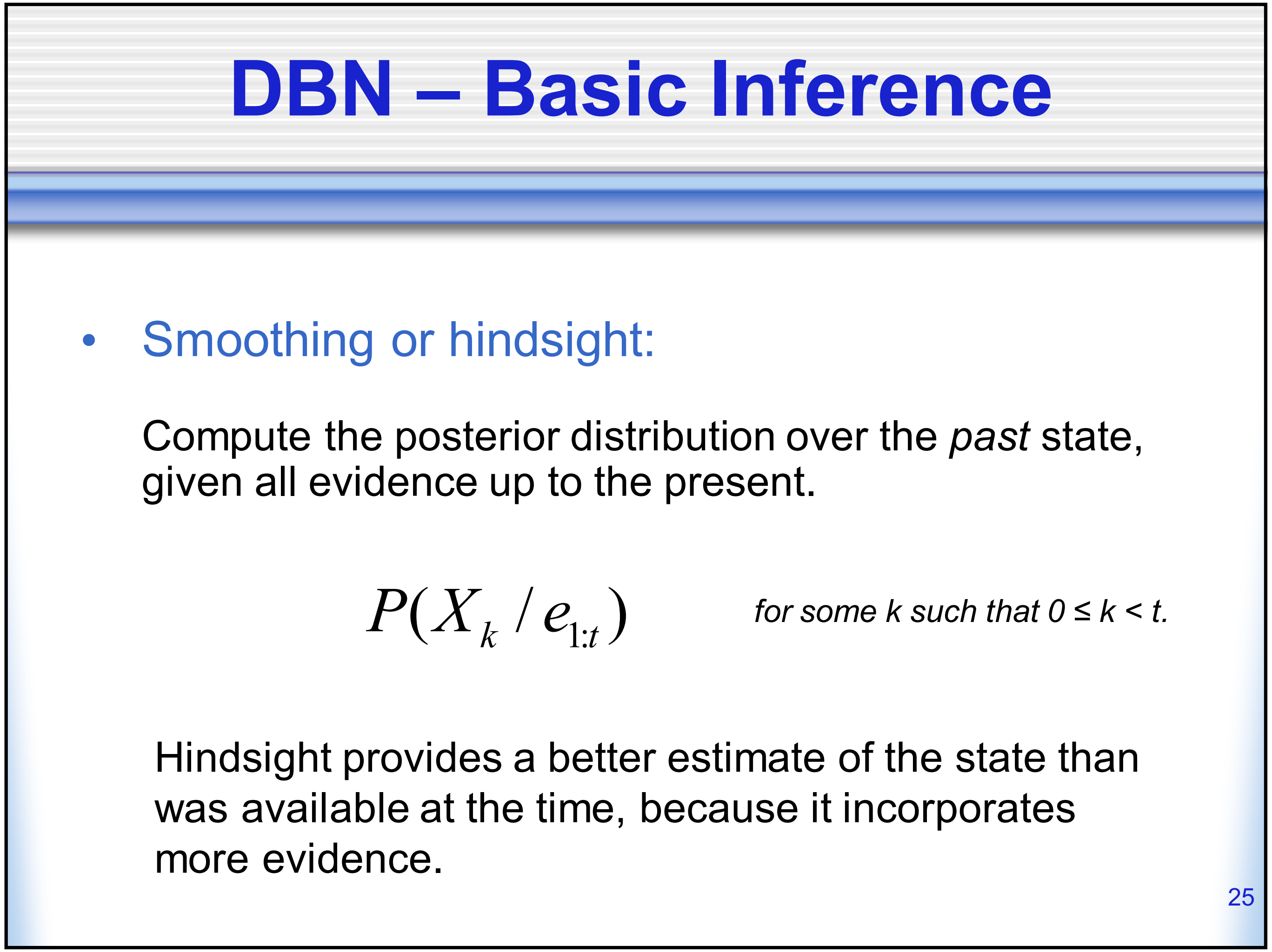
- Prediction:

Compute the posterior distribution over the *future* state, given all evidence to date.

$$P(X_{t+k+1} / e_{1:t}) = \sum_{x_{t+k}} P(X_{t+k+1} \mid x_{t+k}) P(x_{t+k} \mid e_{1:t})$$

*for some k>0*

The task of prediction can be seen simply as filtering without the addition of new evidence.



# DBN – Basic Inference

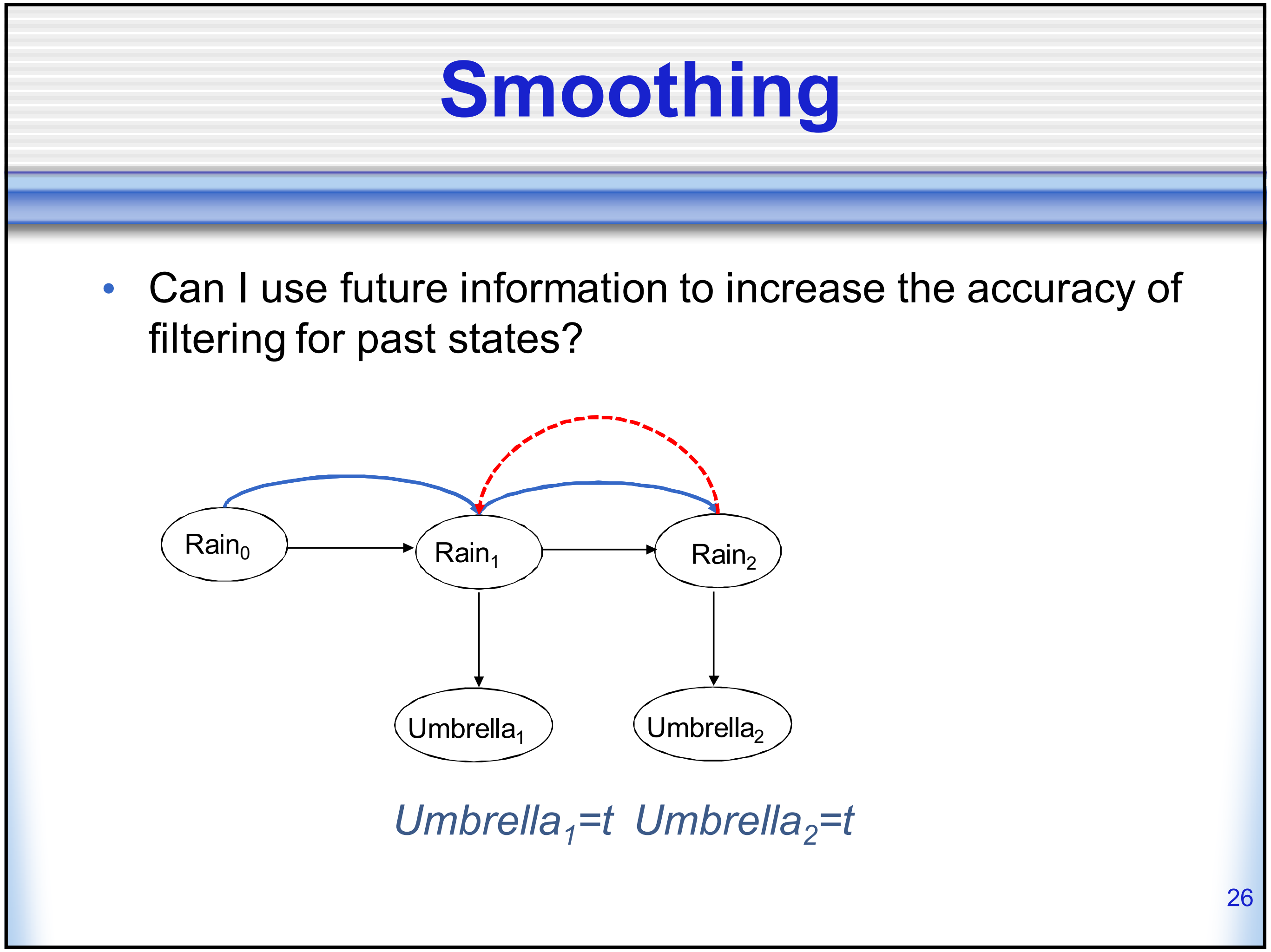
- Smoothing or hindsight:

Compute the posterior distribution over the *past* state, given all evidence up to the present.

$$P(X_k / e_{1:t})$$

*for some k such that 0 ≤ k < t.*

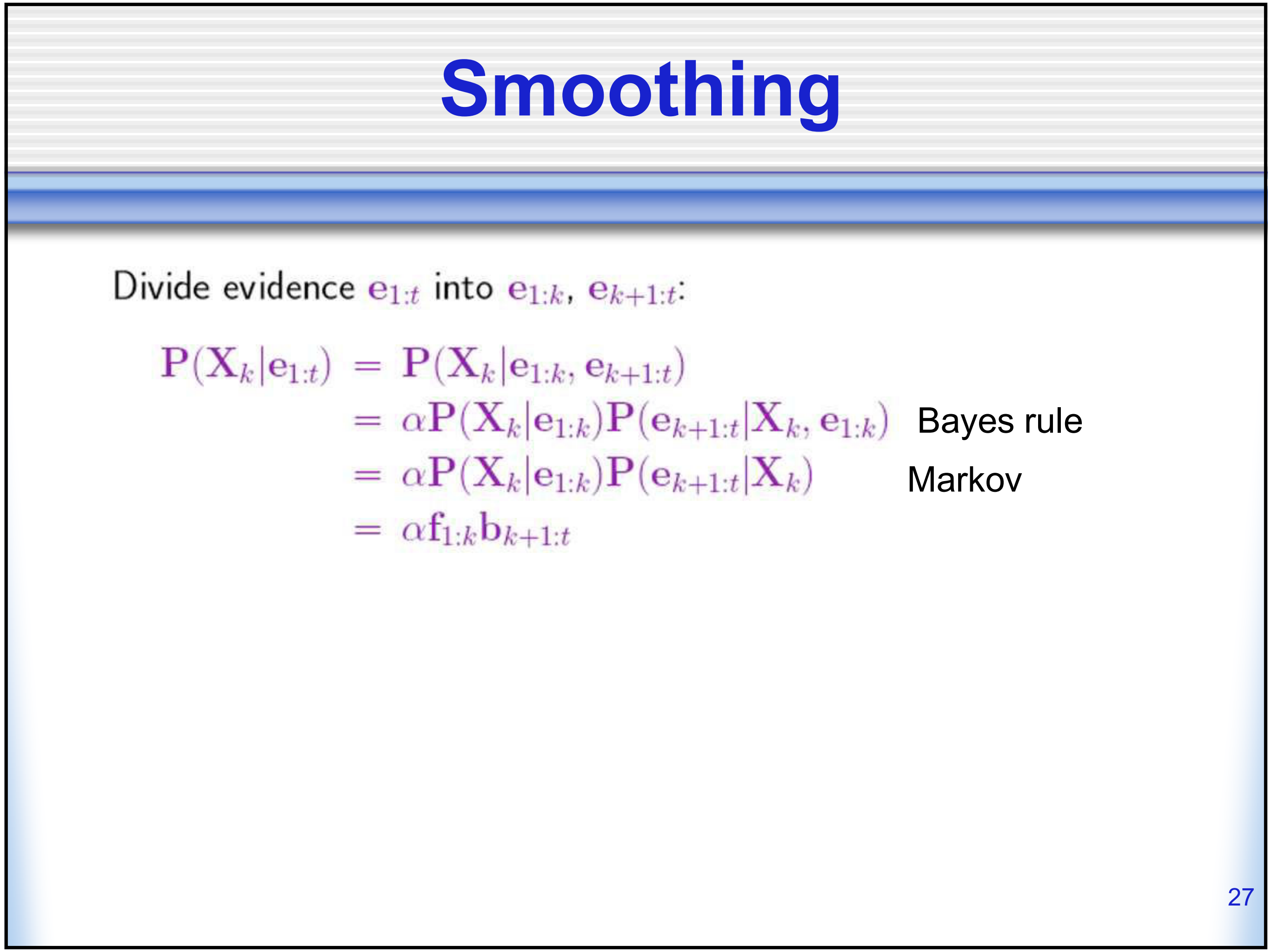
Hindsight provides a better estimate of the state than was available at the time, because it incorporates more evidence.



# Smoothing

- Can I use future information to increase the accuracy of filtering for past states?

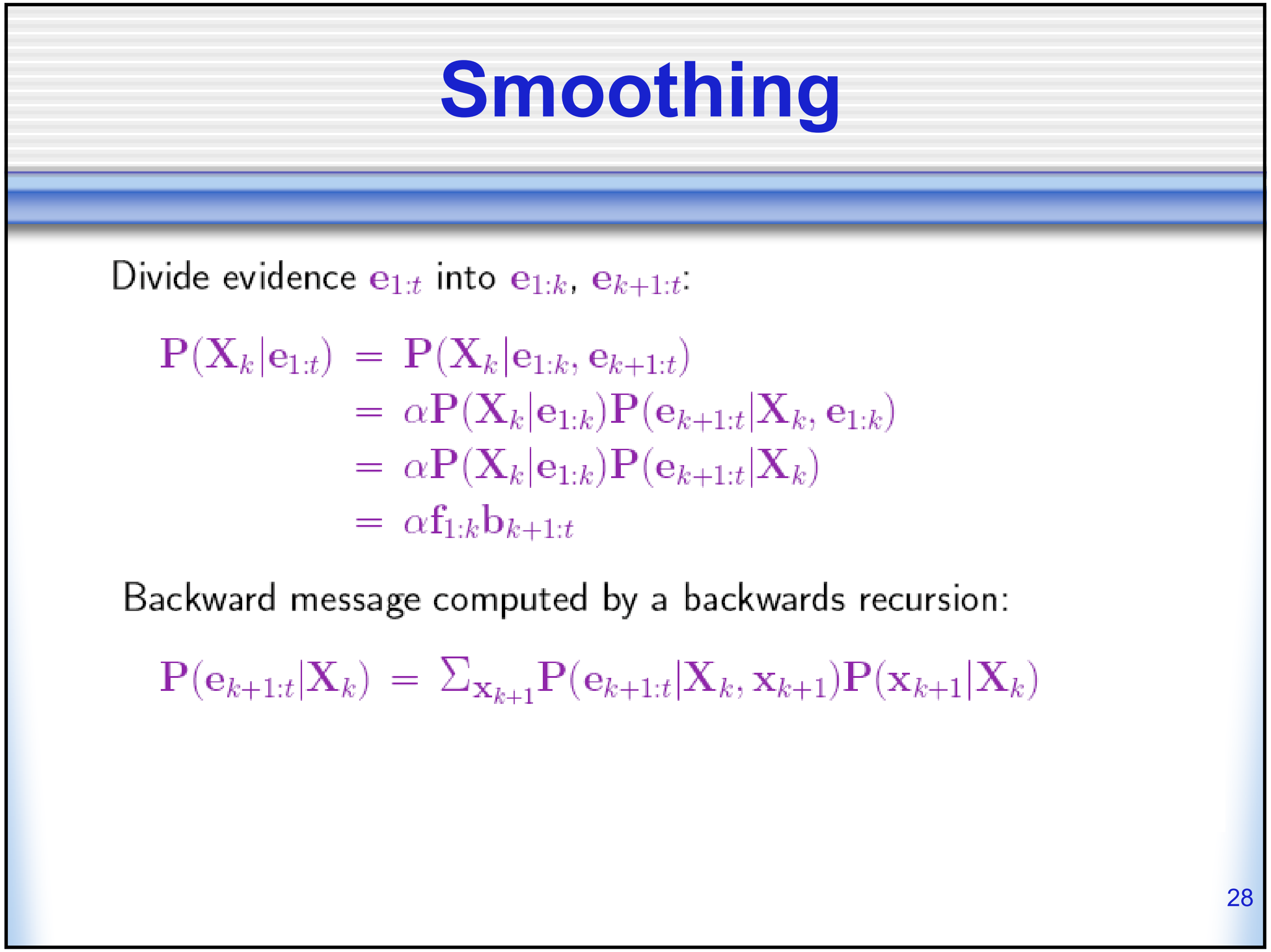
*$Umbrella_1$=t $Umbrella_2$=t*



# Smoothing

Divide evidence $\mathbf{e}_{1:t}$ into $\mathbf{e}_{1:k}$, $\mathbf{e}_{k+1:t}$:

$$\begin{aligned}
\mathbf{P}(\mathbf{X}_k|\mathbf{e}_{1:t}) &= \mathbf{P}(\mathbf{X}_k|\mathbf{e}_{1:k}, \mathbf{e}_{k+1:t}) \\
&= \alpha\mathbf{P}(\mathbf{X}_k|\mathbf{e}_{1:k})\mathbf{P}(\mathbf{e}_{k+1:t}|\mathbf{X}_k, \mathbf{e}_{1:k}) \quad \text{Bayes rule} \\
&= \alpha\mathbf{P}(\mathbf{X}_k|\mathbf{e}_{1:k})\mathbf{P}(\mathbf{e}_{k+1:t}|\mathbf{X}_k) \quad \text{Markov} \\
&= \alpha\mathbf{f}_{1:k}\mathbf{b}_{k+1:t}
\end{aligned}$$



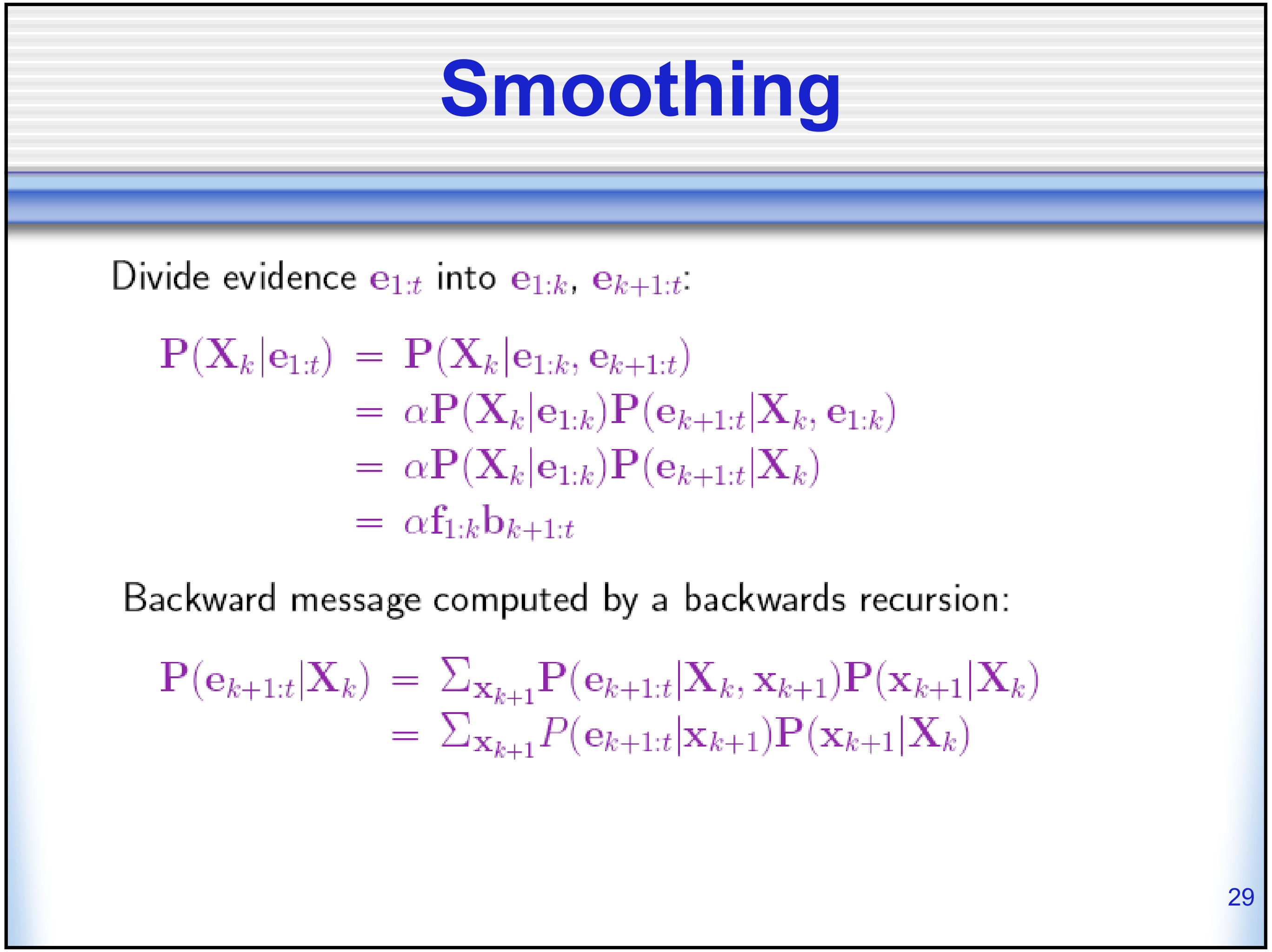
# Smoothing

Divide evidence $\mathbf{e}_{1:t}$ into $\mathbf{e}_{1:k}$, $\mathbf{e}_{k+1:t}$:

$$\begin{aligned}\mathbf{P}(\mathbf{X}_k|\mathbf{e}_{1:t}) &= \mathbf{P}(\mathbf{X}_k|\mathbf{e}_{1:k},\mathbf{e}_{k+1:t})\\ &= \alpha\mathbf{P}(\mathbf{X}_k|\mathbf{e}_{1:k})\mathbf{P}(\mathbf{e}_{k+1:t}|\mathbf{X}_k,\mathbf{e}_{1:k})\\ &= \alpha\mathbf{P}(\mathbf{X}_k|\mathbf{e}_{1:k})\mathbf{P}(\mathbf{e}_{k+1:t}|\mathbf{X}_k)\\ &= \alpha\mathbf{f}_{1:k}\mathbf{b}_{k+1:t}\end{aligned}$$

Backward message computed by a backwards recursion:

$$\mathbf{P}(\mathbf{e}_{k+1:t}|\mathbf{X}_k) = \Sigma_{\mathbf{x}_{k+1}}\mathbf{P}(\mathbf{e}_{k+1:t}|\mathbf{X}_k,\mathbf{x}_{k+1})\mathbf{P}(\mathbf{x}_{k+1}|\mathbf{X}_k)$$



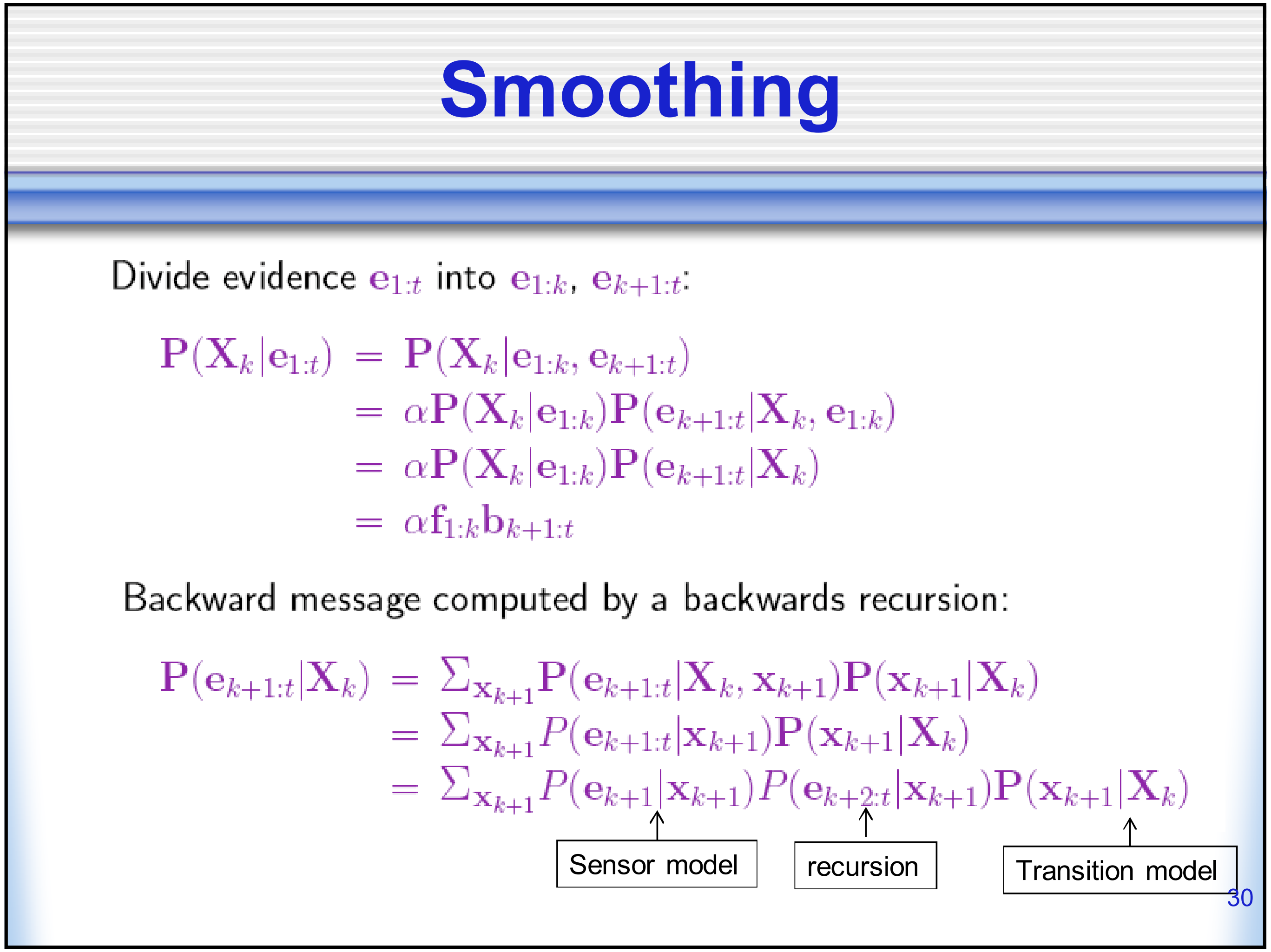
# Smoothing

Divide evidence $\mathbf{e}_{1:t}$ into $\mathbf{e}_{1:k}$, $\mathbf{e}_{k+1:t}$:

$$\begin{aligned}
\mathbf{P}(\mathbf{X}_k|\mathbf{e}_{1:t}) &= \mathbf{P}(\mathbf{X}_k|\mathbf{e}_{1:k}, \mathbf{e}_{k+1:t}) \\
&= \alpha\mathbf{P}(\mathbf{X}_k|\mathbf{e}_{1:k})\mathbf{P}(\mathbf{e}_{k+1:t}|\mathbf{X}_k, \mathbf{e}_{1:k}) \\
&= \alpha\mathbf{P}(\mathbf{X}_k|\mathbf{e}_{1:k})\mathbf{P}(\mathbf{e}_{k+1:t}|\mathbf{X}_k) \\
&= \alpha\mathbf{f}_{1:k}\mathbf{b}_{k+1:t}
\end{aligned}$$

Backward message computed by a backwards recursion:

$$\begin{aligned}
\mathbf{P}(\mathbf{e}_{k+1:t}|\mathbf{X}_k) &= \Sigma_{\mathbf{x}_{k+1}}\mathbf{P}(\mathbf{e}_{k+1:t}|\mathbf{X}_k, \mathbf{x}_{k+1})\mathbf{P}(\mathbf{x}_{k+1}|\mathbf{X}_k) \\
&= \Sigma_{\mathbf{x}_{k+1}}P(\mathbf{e}_{k+1:t}|\mathbf{x}_{k+1})\mathbf{P}(\mathbf{x}_{k+1}|\mathbf{X}_k)
\end{aligned}$$



# Smoothing

Divide evidence $\mathbf{e}_{1:t}$ into $\mathbf{e}_{1:k}$, $\mathbf{e}_{k+1:t}$:

$$\begin{aligned}
\mathbf{P}(\mathbf{X}_k|\mathbf{e}_{1:t}) &= \mathbf{P}(\mathbf{X}_k|\mathbf{e}_{1:k}, \mathbf{e}_{k+1:t}) \\
&= \alpha \mathbf{P}(\mathbf{X}_k|\mathbf{e}_{1:k})\mathbf{P}(\mathbf{e}_{k+1:t}|\mathbf{X}_k, \mathbf{e}_{1:k}) \\
&= \alpha \mathbf{P}(\mathbf{X}_k|\mathbf{e}_{1:k})\mathbf{P}(\mathbf{e}_{k+1:t}|\mathbf{X}_k) \\
&= \alpha \mathbf{f}_{1:k}\mathbf{b}_{k+1:t}
\end{aligned}$$

Backward message computed by a backwards recursion:

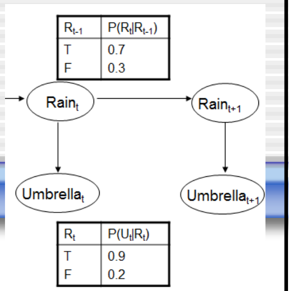
$$\begin{aligned}
\mathbf{P}(\mathbf{e}_{k+1:t}|\mathbf{X}_k) &= \Sigma_{\mathbf{x}_{k+1}} \mathbf{P}(\mathbf{e}_{k+1:t}|\mathbf{X}_k, \mathbf{x}_{k+1})\mathbf{P}(\mathbf{x}_{k+1}|\mathbf{X}_k) \\
&= \Sigma_{\mathbf{x}_{k+1}} P(\mathbf{e}_{k+1:t}|\mathbf{x}_{k+1})\mathbf{P}(\mathbf{x}_{k+1}|\mathbf{X}_k) \\
&= \Sigma_{\mathbf{x}_{k+1}} P(\mathbf{e}_{k+1}|\mathbf{x}_{k+1})P(\mathbf{e}_{k+2:t}|\mathbf{x}_{k+1})\mathbf{P}(\mathbf{x}_{k+1}|\mathbf{X}_k)
\end{aligned}$$

Sensor model

recursion

Transition model

# Example

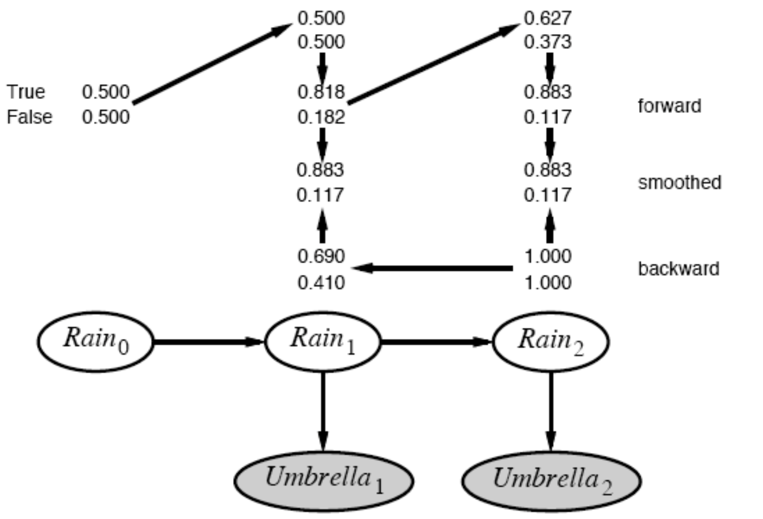


| $R_{t-1}$ | $P(R_t \mid R_{t-1})$ |
|---|---|
| T | 0.7 |
| F | 0.3 |

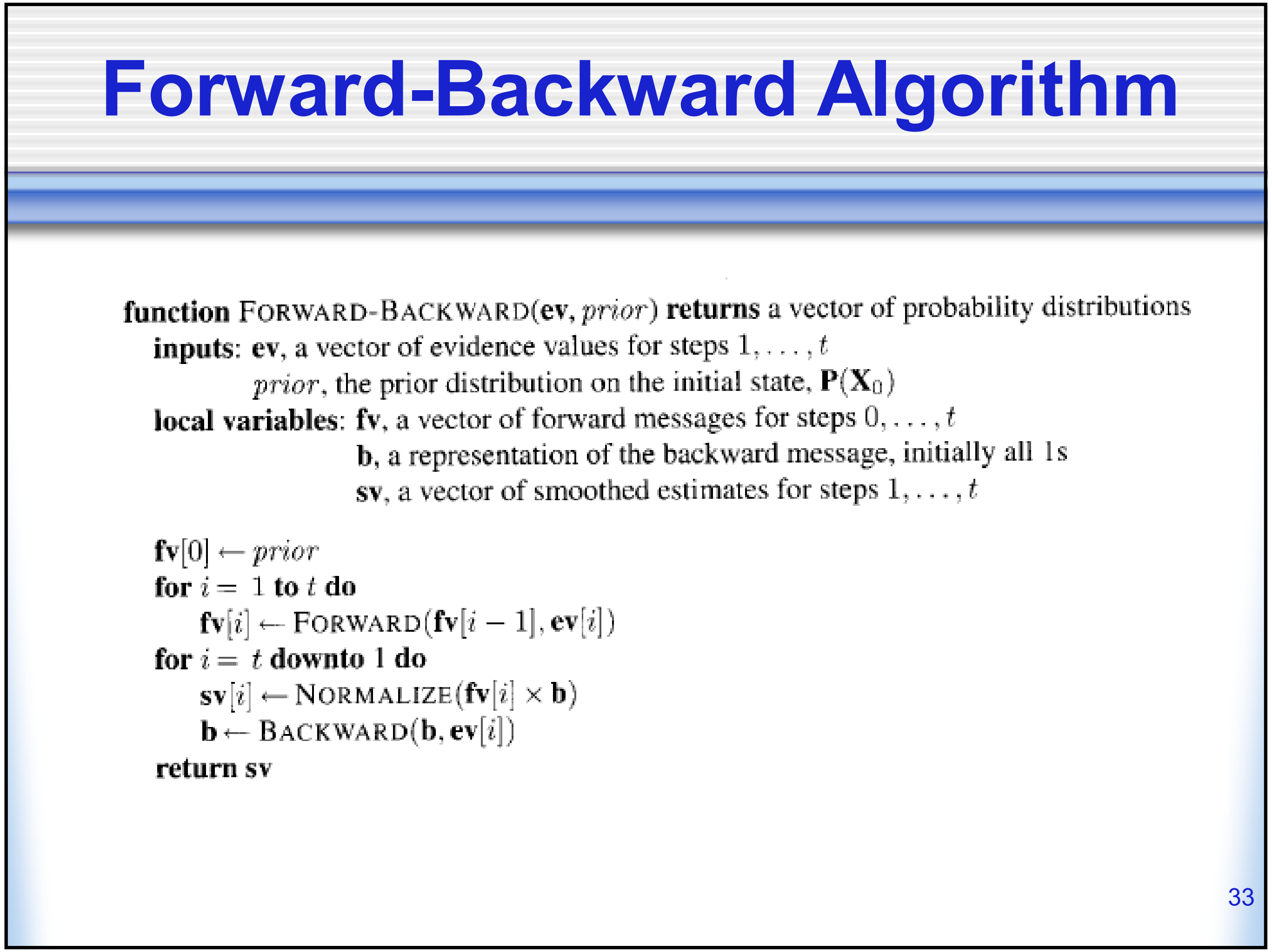
| $R_t$ | $P(U_t \mid R_t)$ |
|---|---|
| T | 0.9 |
| F | 0.2 |

- Smoothed estimate for rain at k=1, given $u_1$, $u_2$.
  $\mathbf{P}(R_1|u_1,u_2) = \alpha\, \mathbf{P}(R_1|u_1)\mathbf{P}(u_2|R_1)$
- The first term is taken from the forward example
  <0.818, 0.182>
- $\mathbf{P}(u_2|R_1) = \Sigma_{r2}\, P(u_2|r_2)P(|r_2)\ \mathbf{P}(r_2|R_1)$
  $= (0.9 \text{ x } 1 \text{ x } <0.7,0.3>)+(0.2 \text{ x } 1 \text{ x } <0.3,0.7>)$
  $= <0.69, 0.41>$
- $\mathbf{P}(R_1|u_1,u_2) = \alpha <0.818, 0.182> \text{ x } <0.69. 0.41>$
  $\approx <0.883, 0.117>$
- If we do it for each time slice $O(t^2)$!!!

# Example contd.

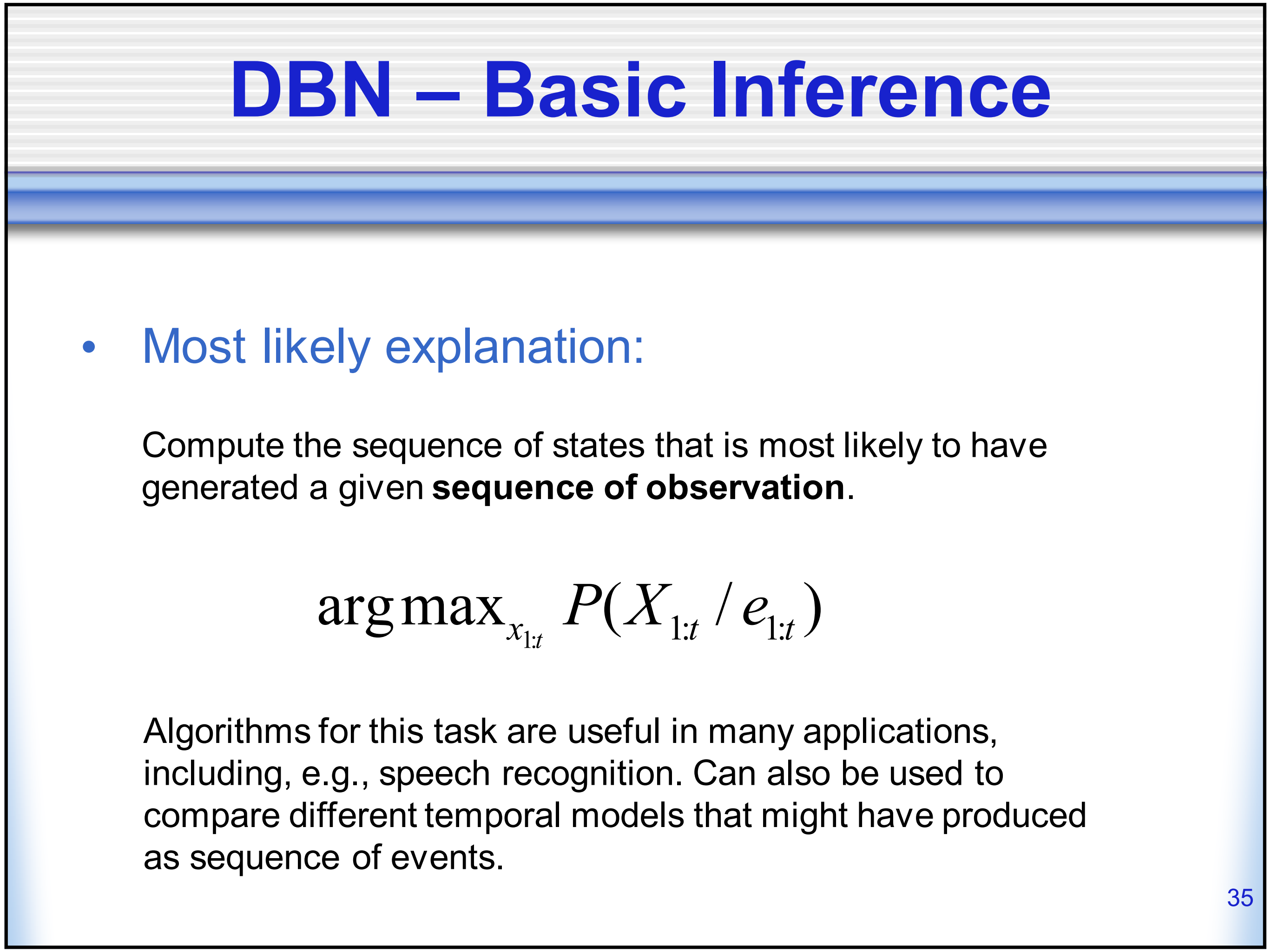


Forward–backward algorithm: cache forward messages along the way
Time linear in $t$ (polytree inference), space $O(t|\mathbf{f}|)$



# Forward-Backward Algorithm

```
function FORWARD-BACKWARD(ev, prior) returns a vector of probability distributions
   inputs: ev, a vector of evidence values for steps $1, \ldots, t$
           prior, the prior distribution on the initial state, $\mathbf{P}(\mathbf{X}_0)$
   local variables: fv, a vector of forward messages for steps $0, \ldots, t$
                    b, a representation of the backward message, initially all 1s
                    sv, a vector of smoothed estimates for steps $1, \ldots, t$

   fv[0] ← prior
   for $i = 1$ to $t$ do
      fv[$i$] ← FORWARD(fv[$i-1$], ev[$i$])
   for $i = t$ downto 1 do
      sv[$i$] ← NORMALIZE(fv[$i$] × b)
      b ← BACKWARD(b, ev[$i$])
   return sv
```



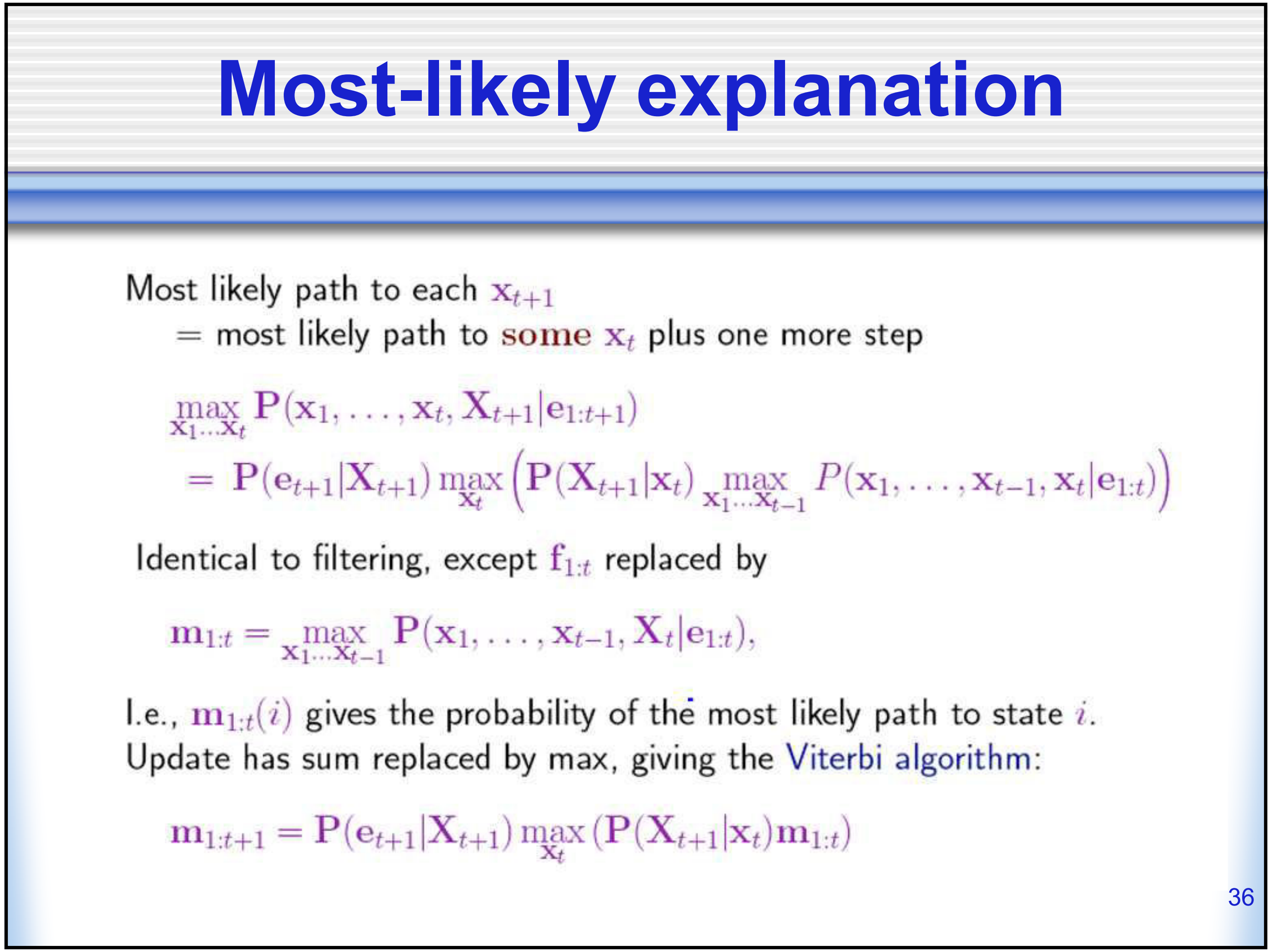
# DBN – Basic Inference

- Most likely explanation:

Compute the sequence of states that is most likely to have generated a given **sequence of observation**.

$$\arg\max_{x_{1:t}} P(X_{1:t} / e_{1:t})$$

Algorithms for this task are useful in many applications, including, e.g., speech recognition. Can also be used to compare different temporal models that might have produced as sequence of events.



# Most-likely explanation

Most likely path to each $\mathbf{x}_{t+1}$
= most likely path to **some** $\mathbf{x}_t$ plus one more step

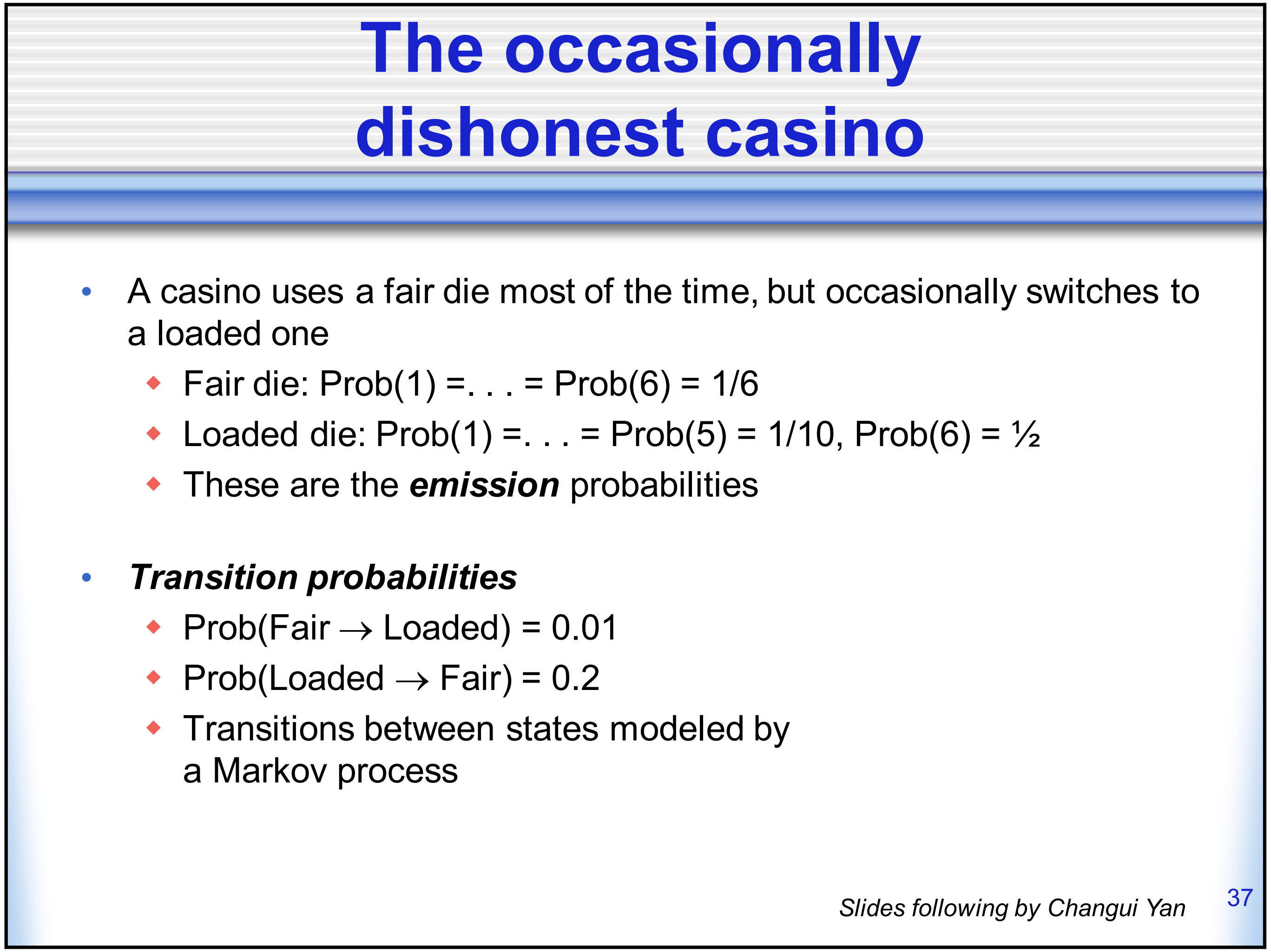
$$\max_{\mathbf{x}_1 \ldots \mathbf{x}_t} \mathbf{P}(\mathbf{x}_1, \ldots, \mathbf{x}_t, \mathbf{X}_{t+1} | \mathbf{e}_{1:t+1})$$
$$= \mathbf{P}(\mathbf{e}_{t+1} | \mathbf{X}_{t+1}) \max_{\mathbf{x}_t} \left( \mathbf{P}(\mathbf{X}_{t+1} | \mathbf{x}_t) \max_{\mathbf{x}_1 \ldots \mathbf{x}_{t-1}} P(\mathbf{x}_1, \ldots, \mathbf{x}_{t-1}, \mathbf{x}_t | \mathbf{e}_{1:t}) \right)$$

Identical to filtering, except $\mathbf{f}_{1:t}$ replaced by

$$\mathbf{m}_{1:t} = \max_{\mathbf{x}_1 \ldots \mathbf{x}_{t-1}} \mathbf{P}(\mathbf{x}_1, \ldots, \mathbf{x}_{t-1}, \mathbf{X}_t | \mathbf{e}_{1:t}),$$

I.e., $\mathbf{m}_{1:t}(i)$ gives the probability of the most likely path to state $i$.
Update has sum replaced by max, giving the Viterbi algorithm:

$$\mathbf{m}_{1:t+1} = \mathbf{P}(\mathbf{e}_{t+1} | \mathbf{X}_{t+1}) \max_{\mathbf{x}_t} (\mathbf{P}(\mathbf{X}_{t+1} | \mathbf{x}_t) \mathbf{m}_{1:t})$$

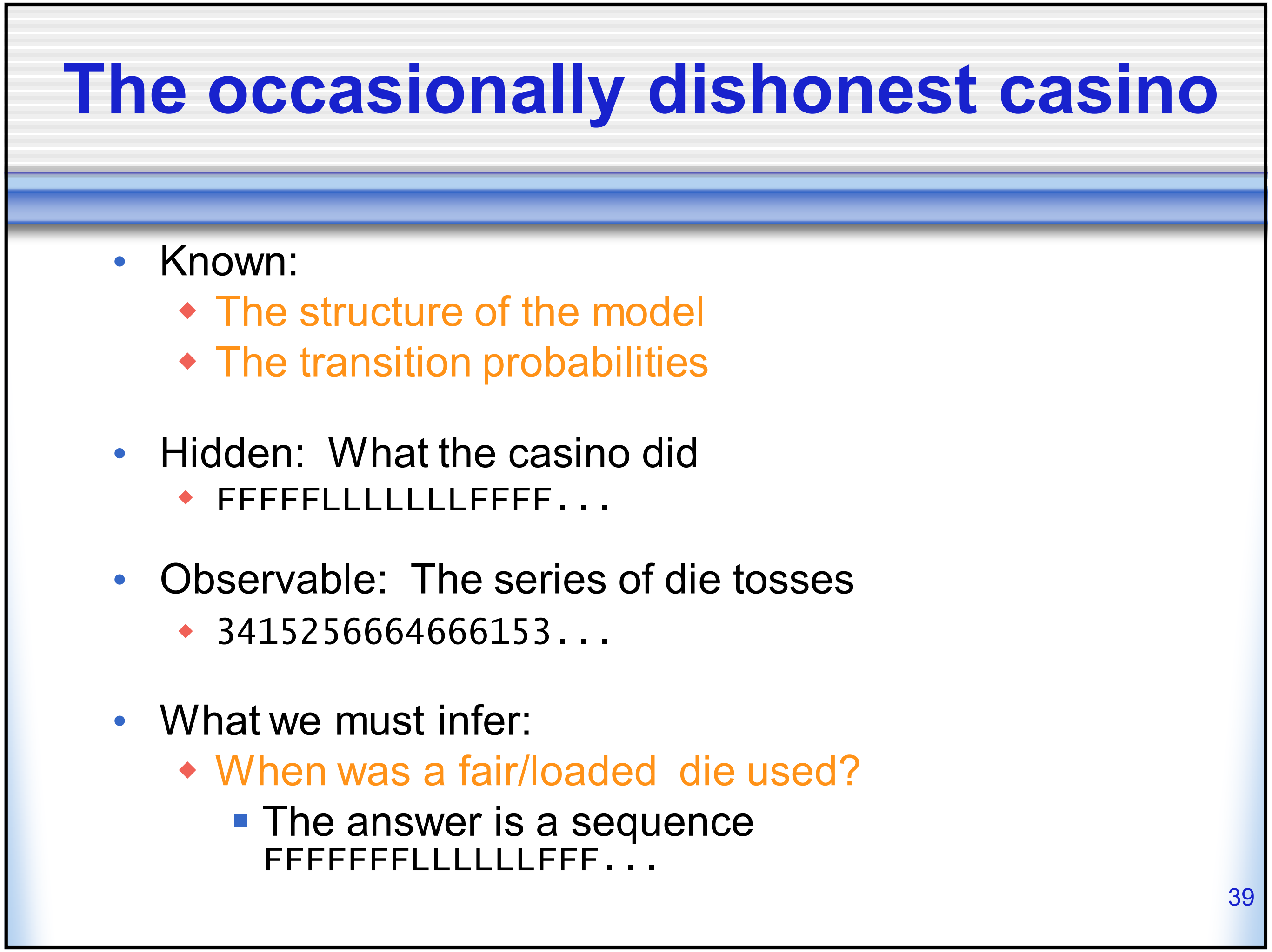


# The occasionally dishonest casino

- A casino uses a fair die most of the time, but occasionally switches to a loaded one
  - Fair die: Prob(1) =. . . = Prob(6) = 1/6
  - Loaded die: Prob(1) =. . . = Prob(5) = 1/10, Prob(6) = ½
  - These are the ***emission*** probabilities

- ***Transition probabilities***
  - Prob(Fair → Loaded) = 0.01
  - Prob(Loaded → Fair) = 0.2
  - Transitions between states modeled by a Markov process

*Slides following by Changui Yan*



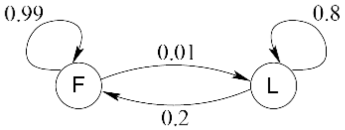
# The occasionally dishonest casino

- Known:
  - The structure of the model
  - The transition probabilities
- Hidden: What the casino did
  - `FFFFFLLLLLLLFFFF...`
- Observable: The series of die tosses
  - `3415256664666153...`
- What we must infer:
  - When was a fair/loaded die used?
    - The answer is a sequence `FFFFFFFLLLLLLFFF...`

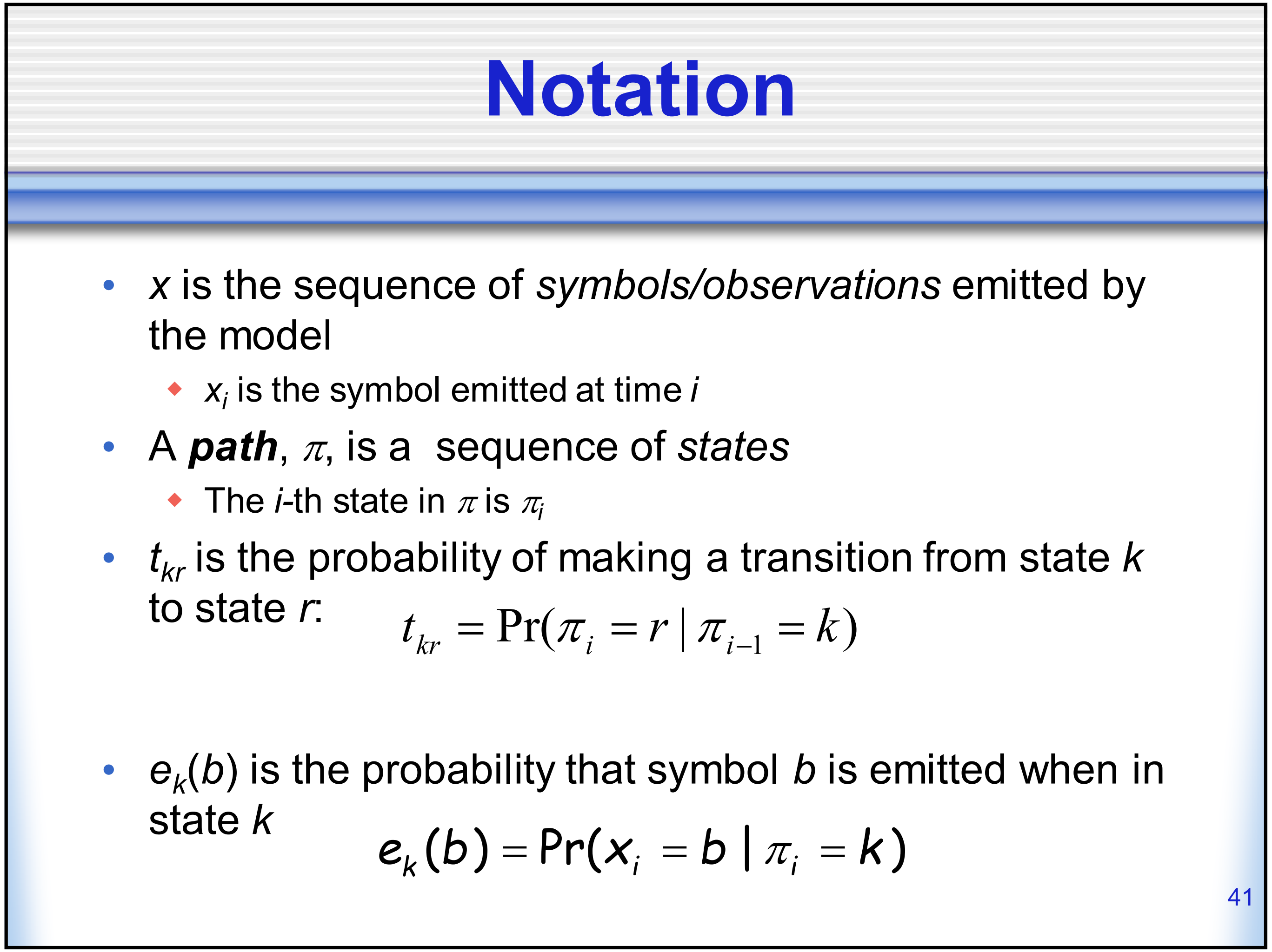
# Making the inference

- Model assigns a probability to each explanation of the observation:

  P(326|FFL)<br>
  = P(3|F)·P(F→F)·P(2|F)·P(F→L)·P(6|L)<br>
  = 1/6 · 0.99 · 1/6 · 0.01 · ½



- **Maximum Likelihood:** Determine which explanation is most likely
  - Find the path *most likely* to have produced the observed sequence
- **Total probability:** Determine the probability that the observed sequence was produced by the model
  - Consider *all* paths that could have produced the observed sequence



# Notation

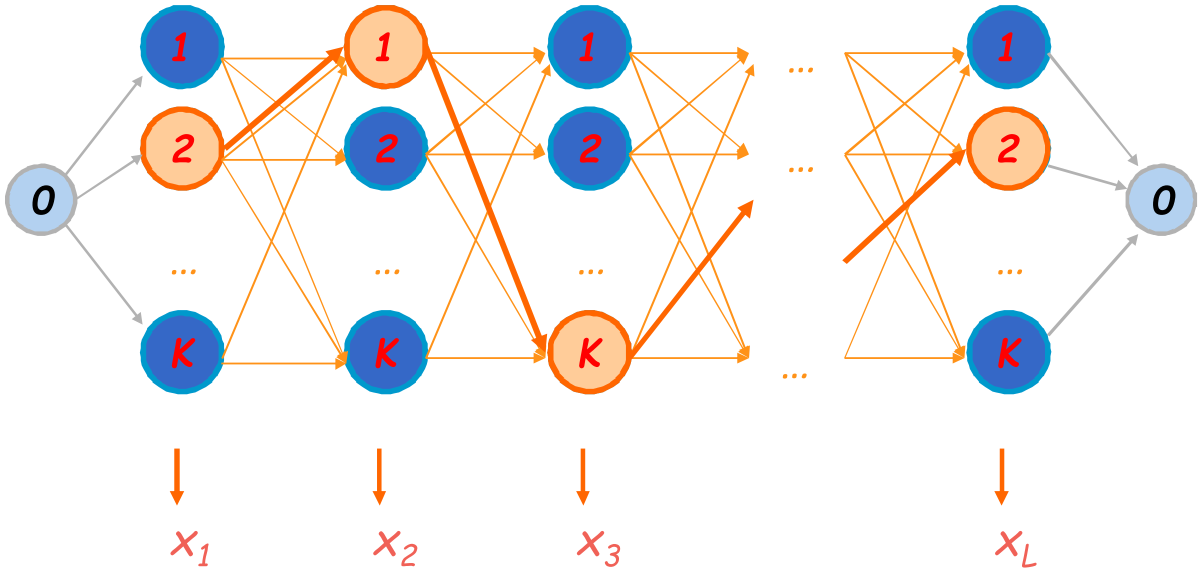
- $x$ is the sequence of *symbols/observations* emitted by the model
  - $x_i$ is the symbol emitted at time $i$
- A ***path***, $\pi$, is a sequence of *states*
  - The $i$-th state in $\pi$ is $\pi_i$
- $t_{kr}$ is the probability of making a transition from state $k$ to state $r$:

$$t_{kr} = \Pr(\pi_i = r \mid \pi_{i-1} = k)$$

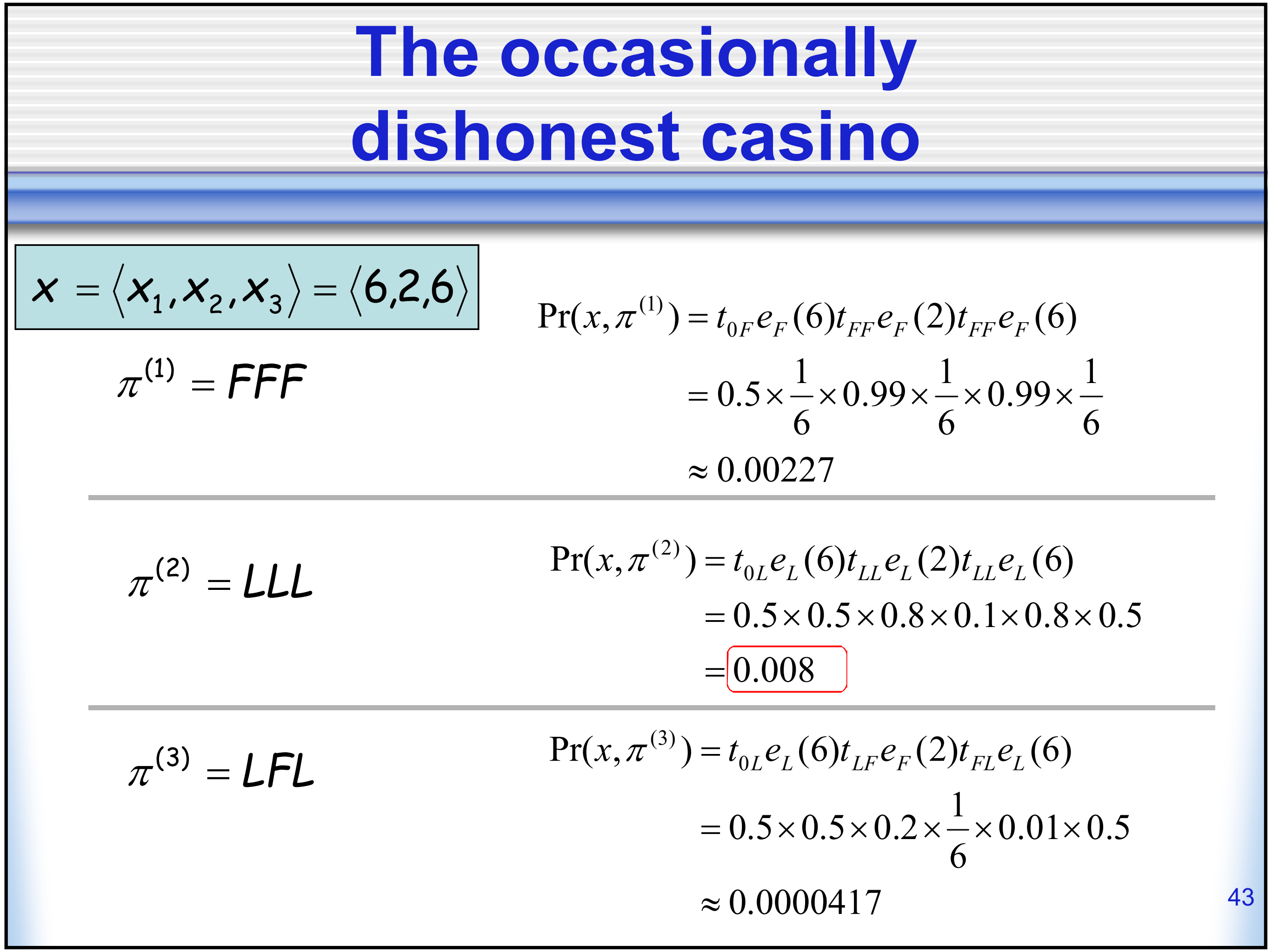
- $e_k(b)$ is the probability that symbol $b$ is emitted when in state $k$

$$e_k(b) = \Pr(x_i = b \mid \pi_i = k)$$

# A "parse" of a sequence



$$\Pr(x,\pi) = t_{0\pi_1} \prod_{i=1}^{L} e_{\pi_i}(x_i) \cdot t_{\pi_i \pi_{i+1}}$$



# The occasionally dishonest casino

$$x = \langle x_1, x_2, x_3 \rangle = \langle 6,2,6 \rangle$$

$$\pi^{(1)} = FFF$$

$$\begin{aligned}\Pr(x,\pi^{(1)}) &= t_{0F}e_F(6)t_{FF}e_F(2)t_{FF}e_F(6) \\ &= 0.5\times\frac{1}{6}\times0.99\times\frac{1}{6}\times0.99\times\frac{1}{6} \\ &\approx 0.00227\end{aligned}$$

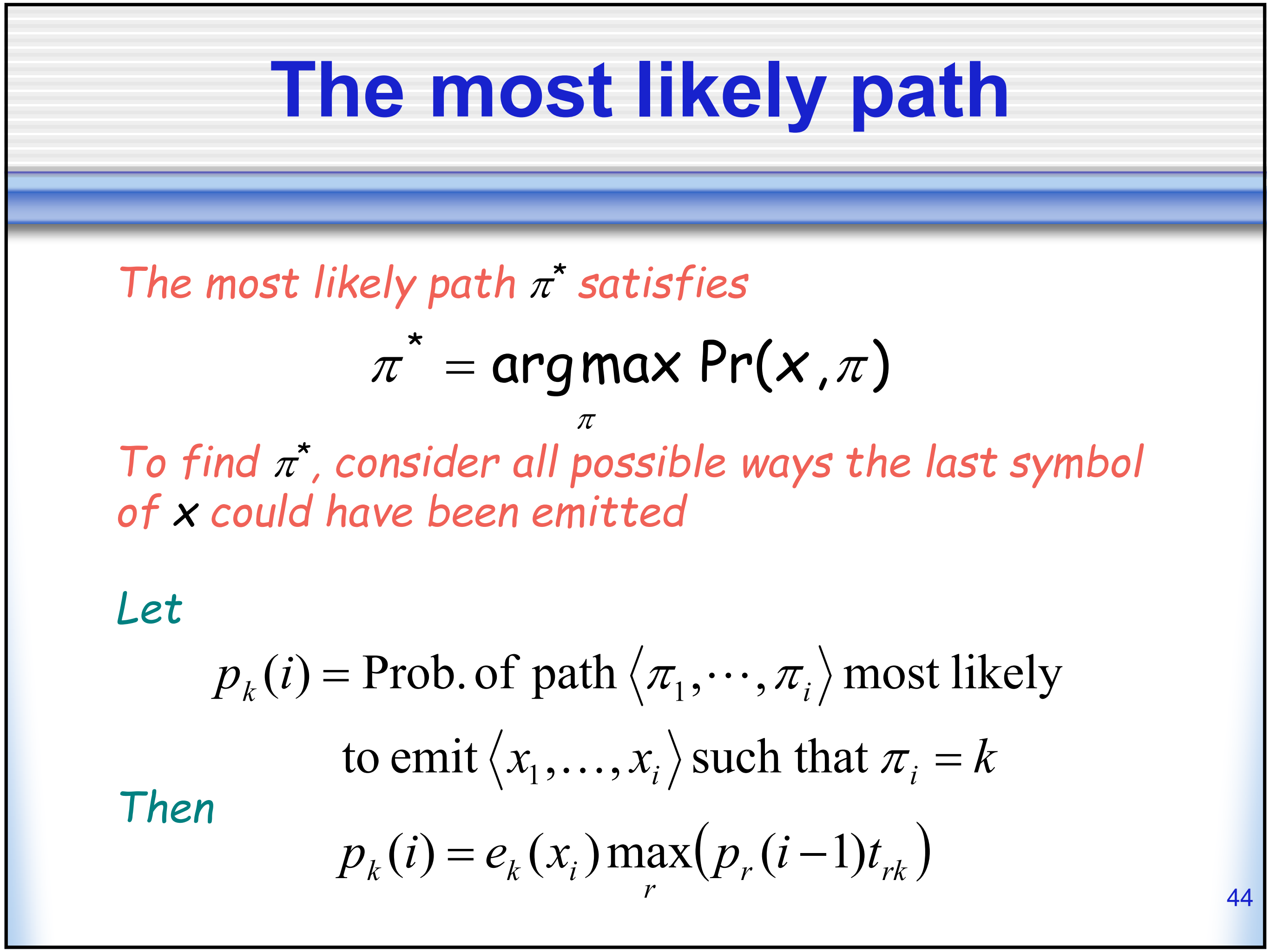
$$\pi^{(2)} = LLL$$

$$\begin{aligned}\Pr(x,\pi^{(2)}) &= t_{0L}e_L(6)t_{LL}e_L(2)t_{LL}e_L(6) \\ &= 0.5\times0.5\times0.8\times0.1\times0.8\times0.5 \\ &= \boxed{0.008}\end{aligned}$$

$$\pi^{(3)} = LFL$$

$$\begin{aligned}\Pr(x,\pi^{(3)}) &= t_{0L}e_L(6)t_{LF}e_F(2)t_{FL}e_L(6) \\ &= 0.5\times0.5\times0.2\times\frac{1}{6}\times0.01\times0.5 \\ &\approx 0.0000417\end{aligned}$$

43



# The most likely path

The most likely path $\pi^*$ satisfies

$$\pi^* = \underset{\pi}{\arg\max}\ \Pr(x,\pi)$$

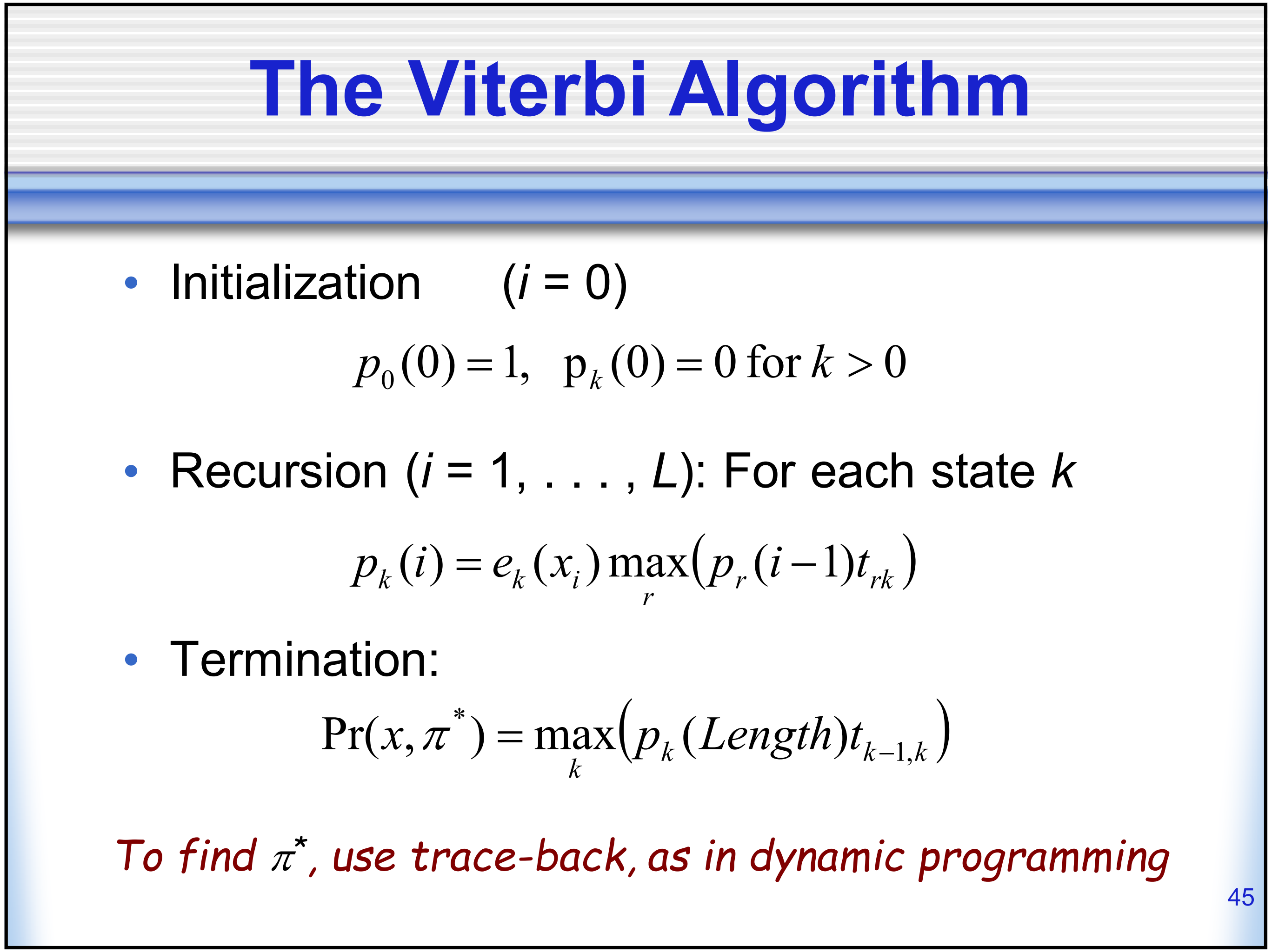
To find $\pi^*$, consider all possible ways the last symbol of $x$ could have been emitted

Let

$$p_k(i) = \text{Prob. of path } \langle \pi_1, \cdots, \pi_i \rangle \text{ most likely}$$
$$\text{to emit } \langle x_1, \ldots, x_i \rangle \text{ such that } \pi_i = k$$

Then

$$p_k(i) = e_k(x_i) \max_r \big( p_r(i-1) t_{rk} \big)$$



# The Viterbi Algorithm

- Initialization (*i* = 0)

$$p_0(0) = 1, \quad \mathrm{p}_k(0) = 0 \text{ for } k > 0$$

- Recursion (*i* = 1, . . . , *L*): For each state *k*

$$p_k(i) = e_k(x_i) \max_r \left( p_r(i-1) t_{rk} \right)$$

- Termination:

$$\Pr(x, \pi^*) = \max_k \left( p_k(Length) t_{k-1,k} \right)$$

*To find $\pi^*$, use trace-back, as in dynamic programming*

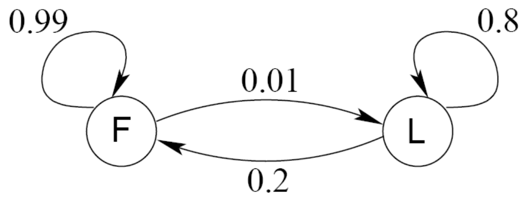
45

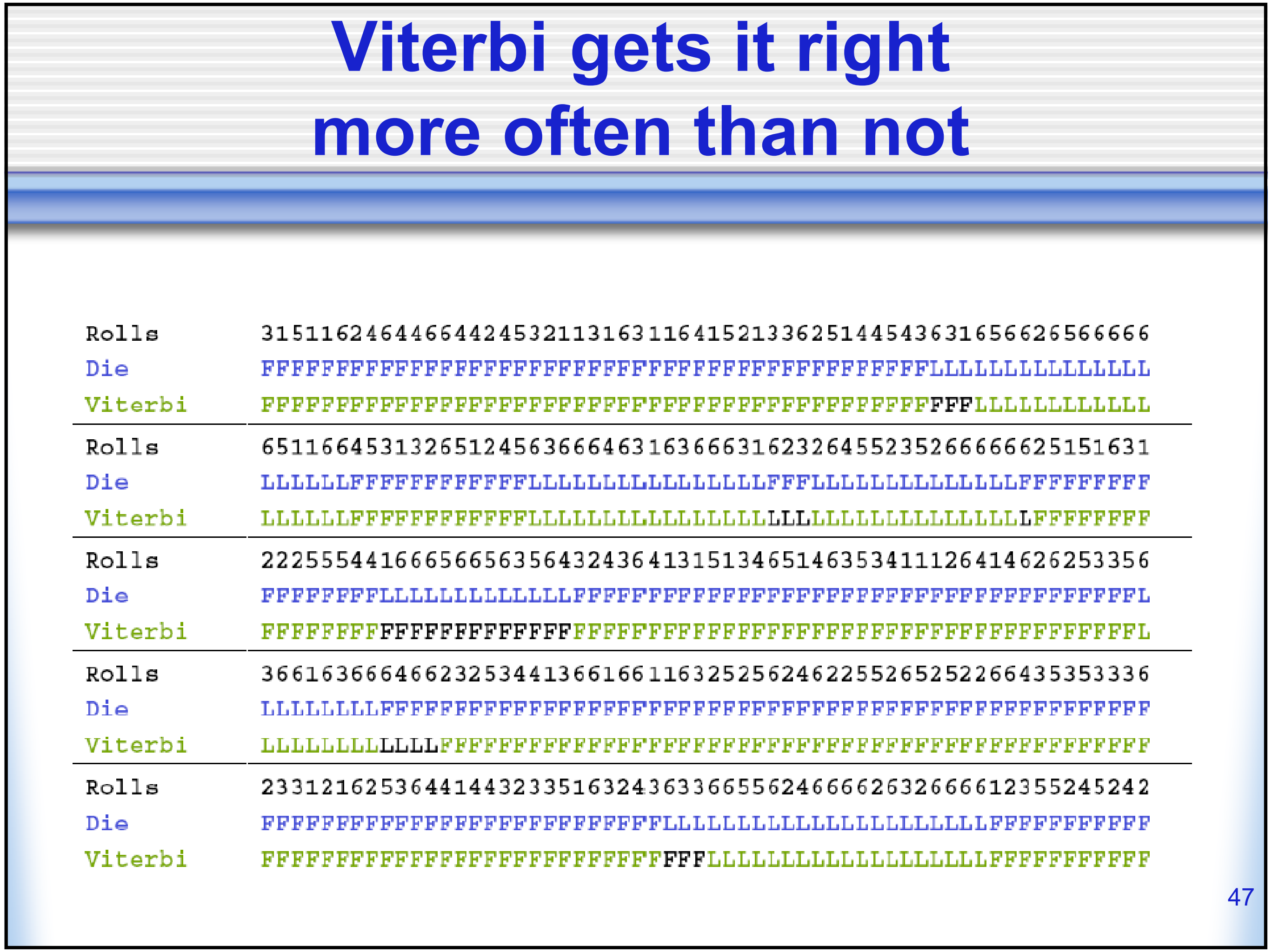
# Viterbi: Example

x

|  |  |  | 6 | 2 | 6 |
|---|---|---|---|---|---|
| $\pi$ | F | 0 | (1/6)×(1/2)<br>= 1/12 | (1/6)×max{(1/12)×0.99,<br>(1/4)×0.2}<br>= 0.01375 | (1/6)×max{0.01375×0.99,<br>0.02×0.2}<br>= 0.00226875 |
|  | L | 0 | (1/2)×(1/2)<br>= 1/4 | (1/10)×max{(1/12)×0.01,<br>(1/4)×0.8}<br>= 0.02 | (1/2)×max{0.01375×0.01,<br>0.02×0.8}<br>= 0.08 |

$$p_k(i) = e_k(x_i)\max_r\left(p_r(i-1)t_{rk}\right)$$

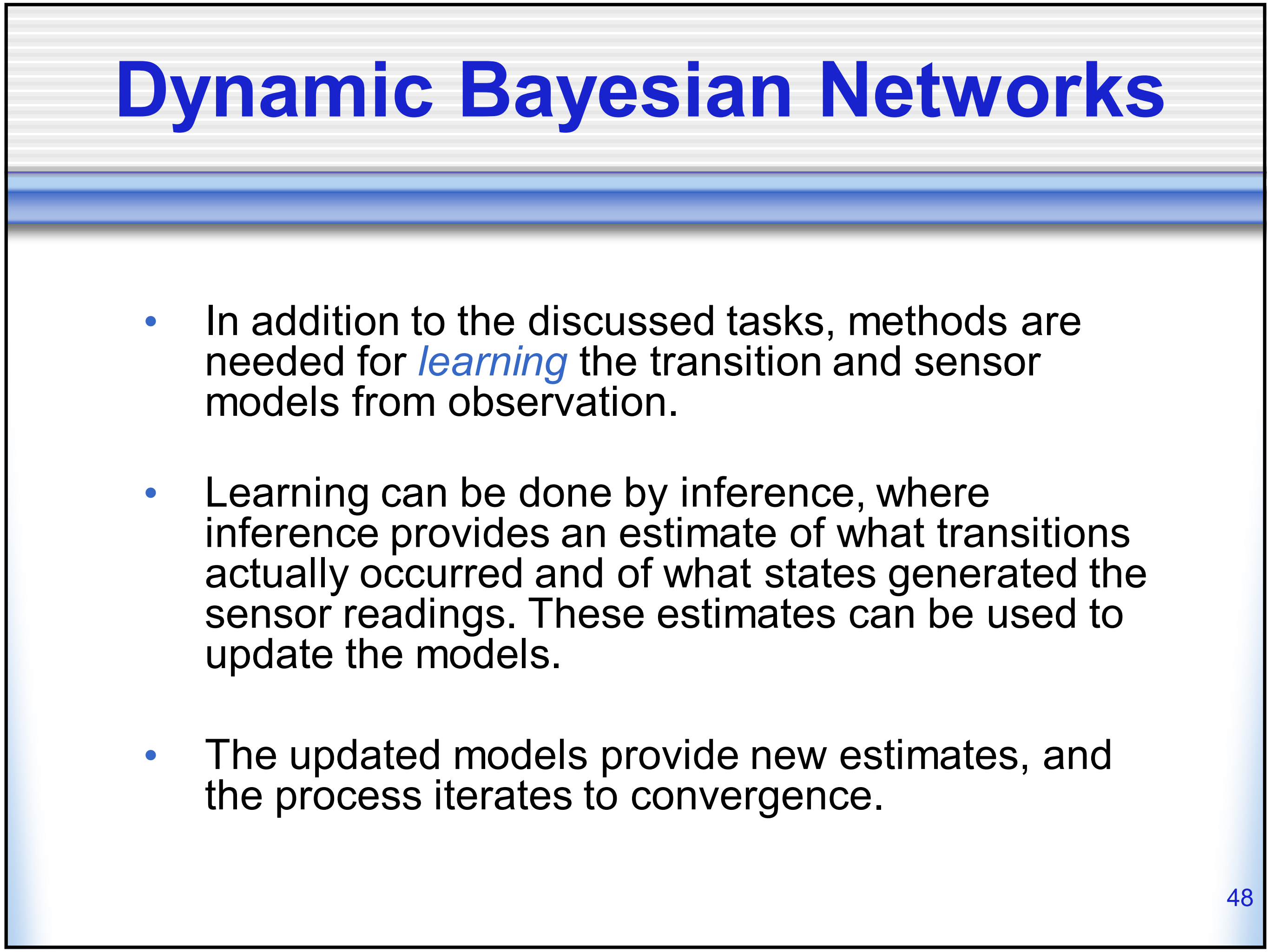




# Viterbi gets it right more often than not

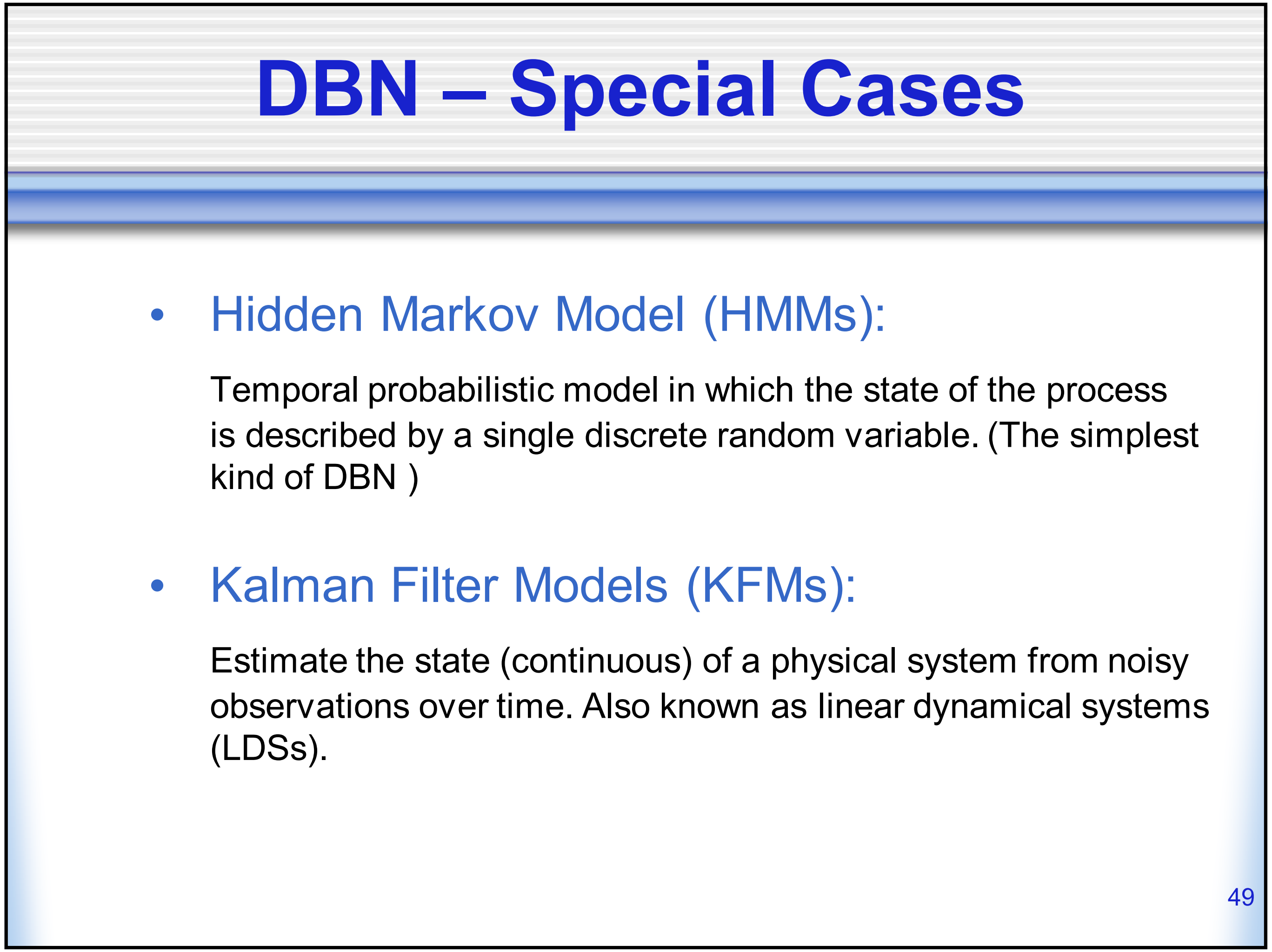
```
Rolls    315116246446644245321131631164152133625144543631656626566666
Die      FFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFLLLLLLLLLLLLLLL
Viterbi  FFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFLLLLLLLLLLLL
Rolls    651166453132651245636664631636663162326455235266666625151631
Die      LLLLLLFFFFFFFFFFFFLLLLLLLLLLLLLLLLFFFLLLLLLLLLLLLLLLFFFFFFFFF
Viterbi  LLLLLLFFFFFFFFFFFFLLLLLLLLLLLLLLLLLLLLLLLLLLLLLLLLLLFFFFFFFF
Rolls    222555441666566563564324364131513465146353411126414626253356
Die      FFFFFFFFLLLLLLLLLLLLLFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFL
Viterbi  FFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFL
Rolls    366163666466232534413661661163252562462255265252266435353336
Die      LLLLLLLLFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFF
Viterbi  LLLLLLLLLLLLFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFF
Rolls    233121625364414432335163243633665562466662632666612355245242
Die      FFFFFFFFFFFFFFFFFFFFFFFFFFFLLLLLLLLLLLLLLLLLLLLLLFFFFFFFFFFF
Viterbi  FFFFFFFFFFFFFFFFFFFFFFFFFFFFFFLLLLLLLLLLLLLLLLLLLFFFFFFFFFFF
```

47



# Dynamic Bayesian Networks

- In addition to the discussed tasks, methods are needed for *learning* the transition and sensor models from observation.
- Learning can be done by inference, where inference provides an estimate of what transitions actually occurred and of what states generated the sensor readings. These estimates can be used to update the models.
- The updated models provide new estimates, and the process iterates to convergence.



# DBN – Special Cases

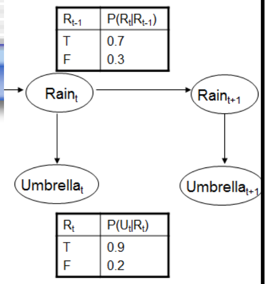
- Hidden Markov Model (HMMs):

  Temporal probabilistic model in which the state of the process is described by a single discrete random variable. (The simplest kind of DBN )

- Kalman Filter Models (KFMs):

  Estimate the state (continuous) of a physical system from noisy observations over time. Also known as linear dynamical systems (LDSs).

# Hidden Markov Models



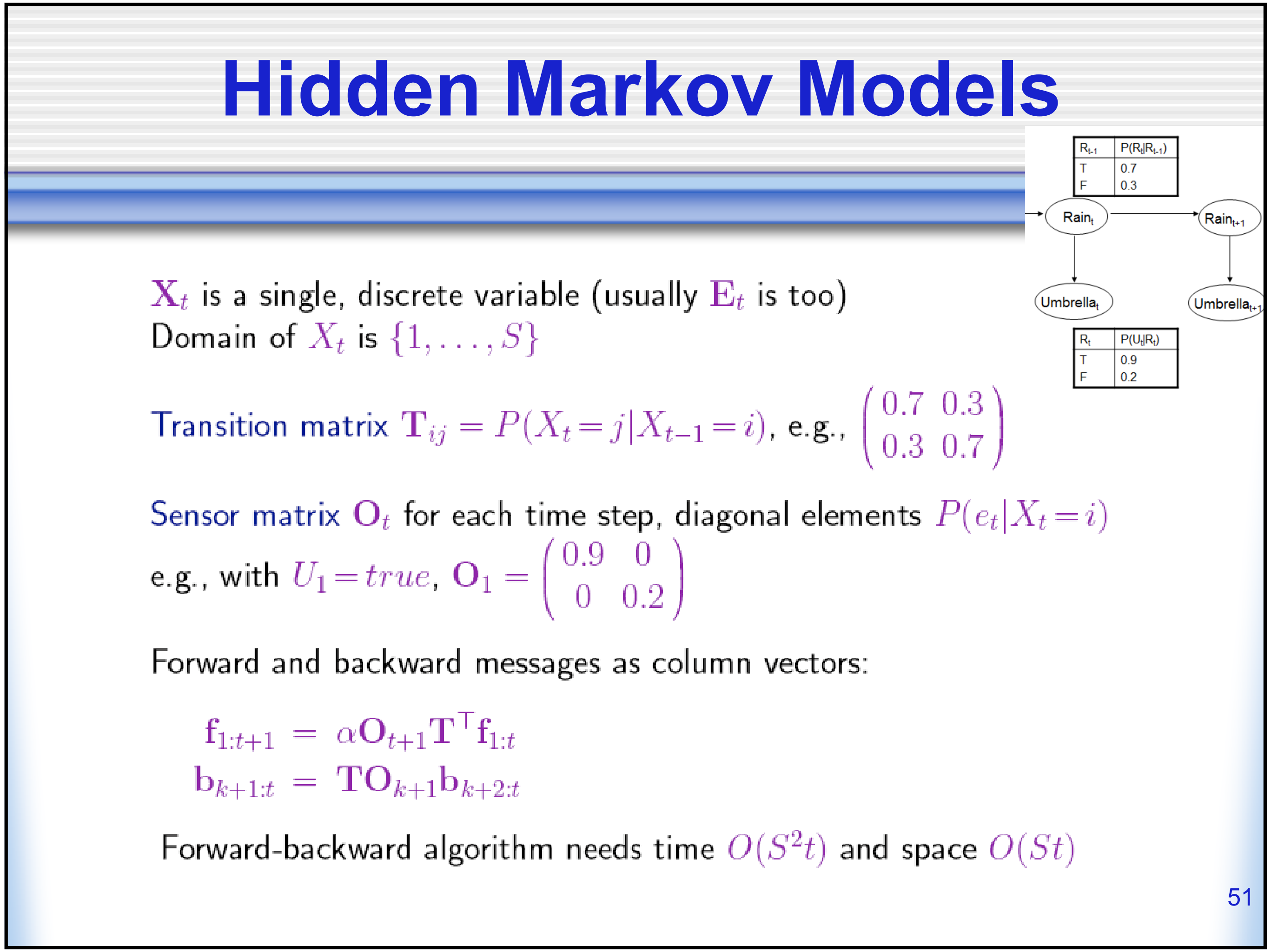
| $R_{t-1}$ | $P(R_t \vert R_{t-1})$ |
|---|---|
| T | 0.7 |
| F | 0.3 |

| $R_t$ | $P(U_t \vert R_t)$ |
|---|---|
| T | 0.9 |
| F | 0.2 |

$\mathbf{X}_t$ is a single, discrete variable (usually $\mathbf{E}_t$ is too)
Domain of $X_t$ is $\{1, \ldots, S\}$

Transition matrix $\mathbf{T}_{ij} = P(X_t = j | X_{t-1} = i)$, e.g., $\begin{pmatrix} 0.7 & 0.3 \\ 0.3 & 0.7 \end{pmatrix}$

Sensor matrix $\mathbf{O}_t$ for each time step, diagonal elements $P(e_t | X_t = i)$
e.g., with $U_1 = true$, $\mathbf{O}_1 = \begin{pmatrix} 0.9 & 0 \\ 0 & 0.2 \end{pmatrix}$



# Hidden Markov Models

| $R_{t-1}$ | $P(R_t\|R_{t-1})$ |
|---|---|
| T | 0.7 |
| F | 0.3 |

| $R_t$ | $P(U_t\|R_t)$ |
|---|---|
| T | 0.9 |
| F | 0.2 |

$\mathbf{X}_t$ is a single, discrete variable (usually $\mathbf{E}_t$ is too)
Domain of $X_t$ is $\{1, \ldots, S\}$

Transition matrix $\mathbf{T}_{ij} = P(X_t = j | X_{t-1} = i)$, e.g., $\begin{pmatrix} 0.7 & 0.3 \\ 0.3 & 0.7 \end{pmatrix}$

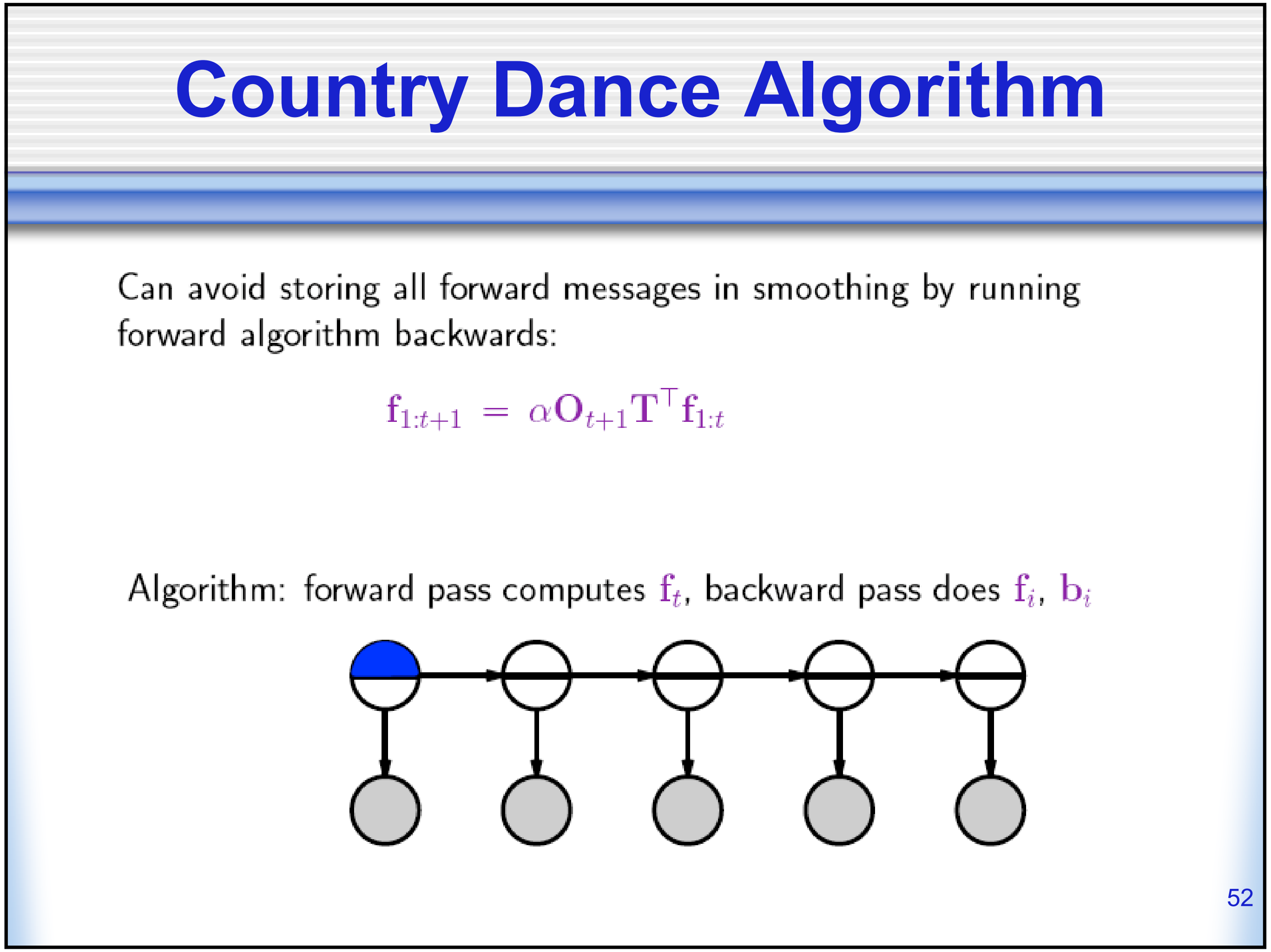
Sensor matrix $\mathbf{O}_t$ for each time step, diagonal elements $P(e_t | X_t = i)$
e.g., with $U_1 = true$, $\mathbf{O}_1 = \begin{pmatrix} 0.9 & 0 \\ 0 & 0.2 \end{pmatrix}$

Forward and backward messages as column vectors:

$$\begin{aligned} \mathbf{f}_{1:t+1} &= \alpha \mathbf{O}_{t+1} \mathbf{T}^{\top} \mathbf{f}_{1:t} \\ \mathbf{b}_{k+1:t} &= \mathbf{T} \mathbf{O}_{k+1} \mathbf{b}_{k+2:t} \end{aligned}$$

Forward-backward algorithm needs time $O(S^2 t)$ and space $O(St)$

51

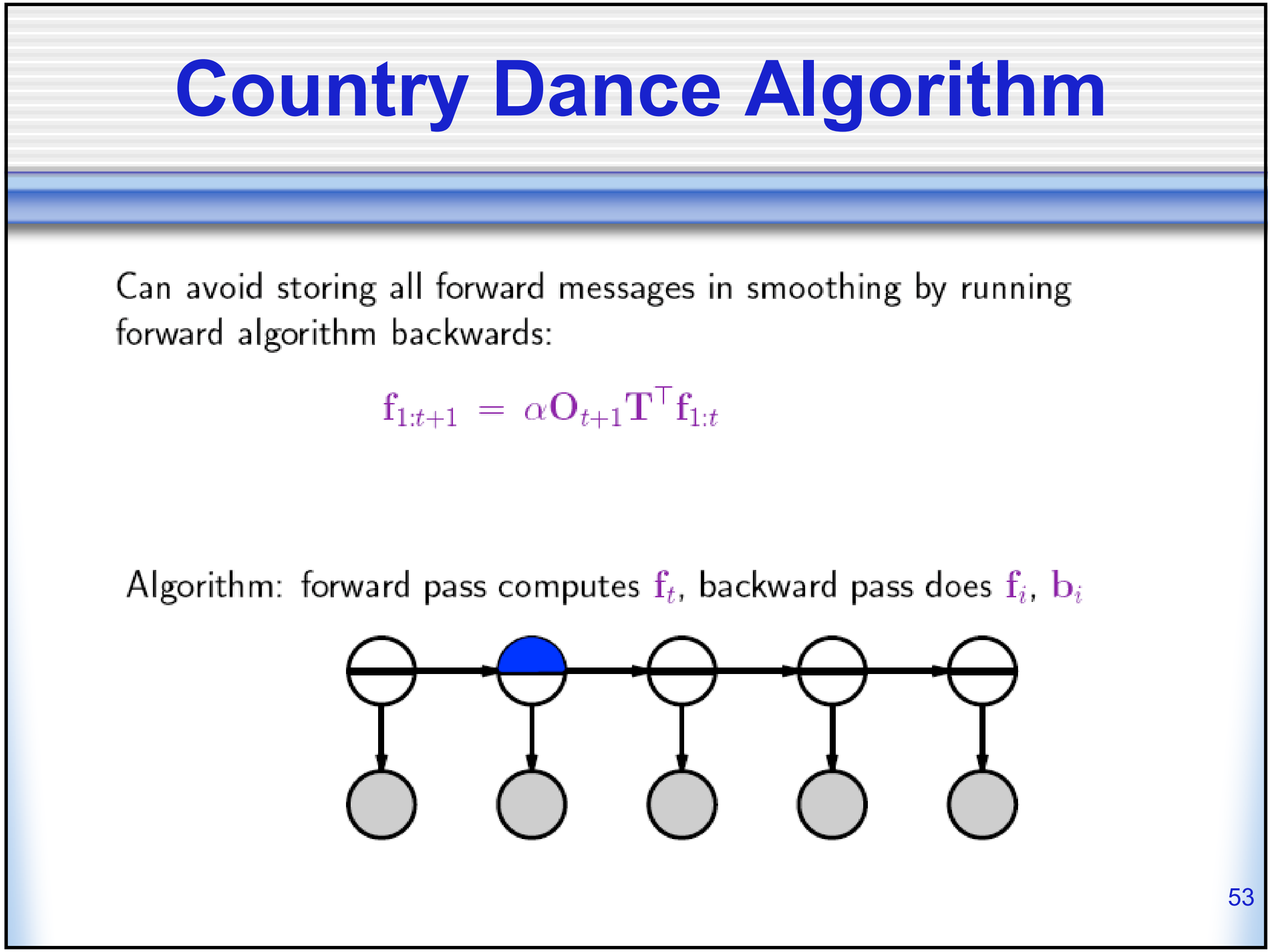


# Country Dance Algorithm

Can avoid storing all forward messages in smoothing by running forward algorithm backwards:

$$\mathbf{f}_{1:t+1} = \alpha \mathbf{O}_{t+1} \mathbf{T}^{\top} \mathbf{f}_{1:t}$$

Algorithm: forward pass computes $\mathbf{f}_t$, backward pass does $\mathbf{f}_i$, $\mathbf{b}_i$



# Country Dance Algorithm

Can avoid storing all forward messages in smoothing by running forward algorithm backwards:

$$\mathbf{f}_{1:t+1} = \alpha \mathbf{O}_{t+1} \mathbf{T}^{\top} \mathbf{f}_{1:t}$$

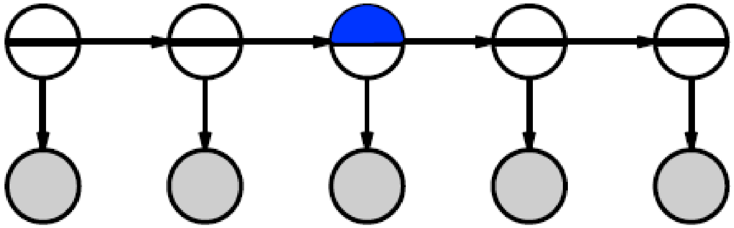
Algorithm: forward pass computes $\mathbf{f}_t$, backward pass does $\mathbf{f}_i$, $\mathbf{b}_i$

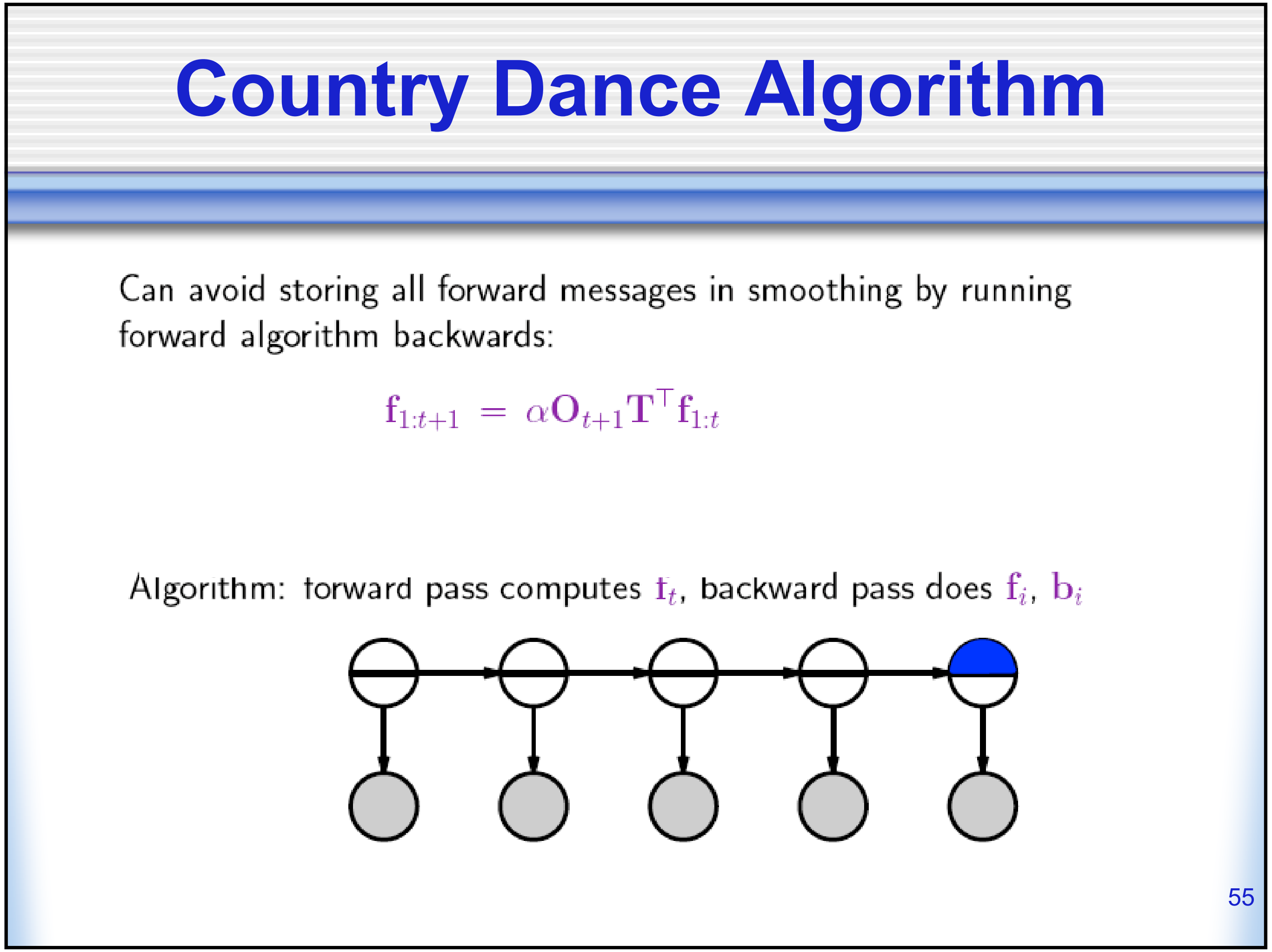
# Country Dance Algorithm

Can avoid storing all forward messages in smoothing by running forward algorithm backwards:

$$\mathbf{f}_{1:t+1} = \alpha \mathbf{O}_{t+1} \mathbf{T}^{\top} \mathbf{f}_{1:t}$$

Algorithm: forward pass computes $\mathbf{f}_t$, backward pass does $\mathbf{f}_i$, $\mathbf{b}_i$



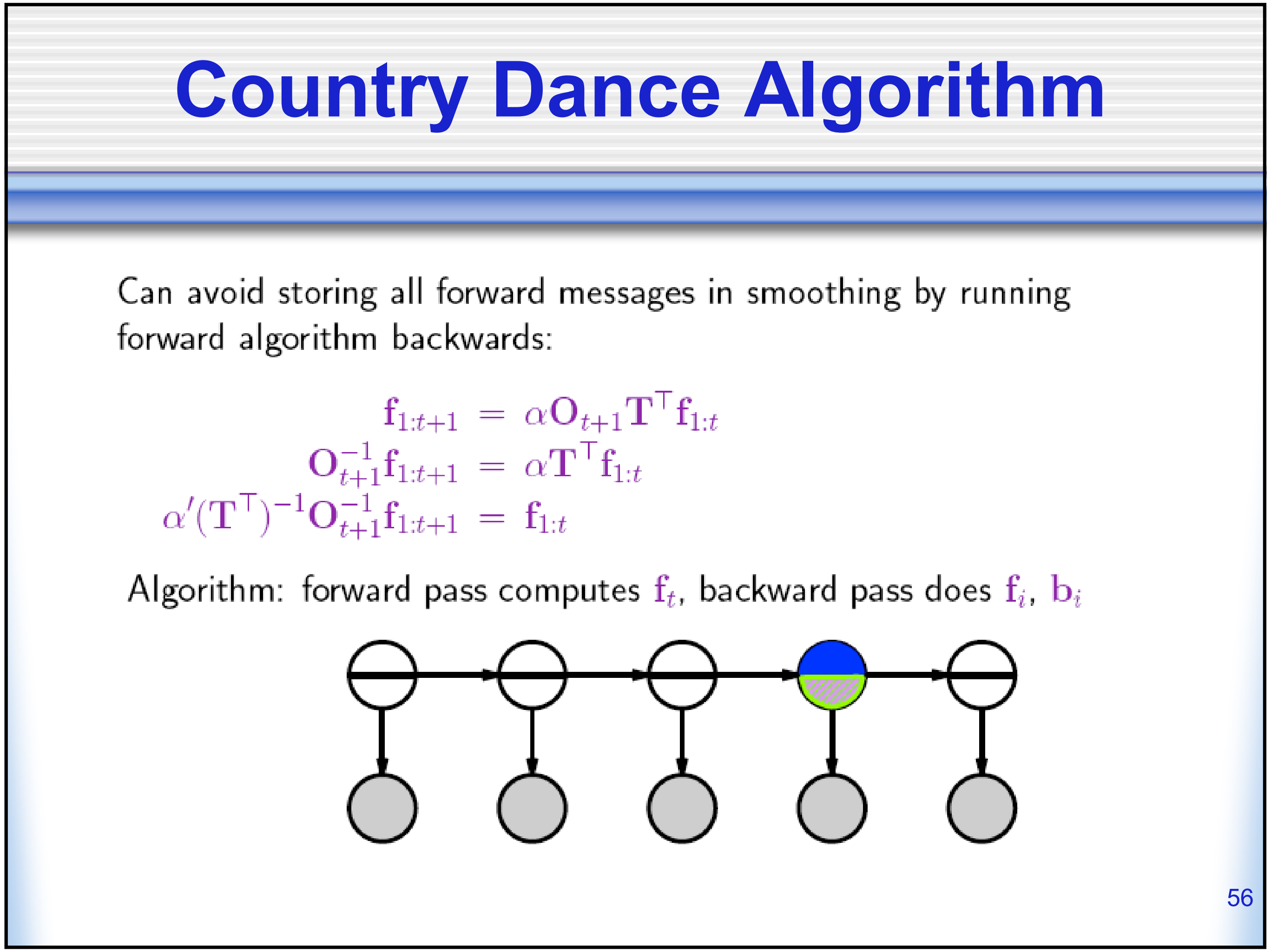


# Country Dance Algorithm

Can avoid storing all forward messages in smoothing by running forward algorithm backwards:

$$\mathbf{f}_{1:t+1} = \alpha \mathbf{O}_{t+1} \mathbf{T}^{\top} \mathbf{f}_{1:t}$$

Algorithm: forward pass computes $\mathbf{f}_t$, backward pass does $\mathbf{f}_i$, $\mathbf{b}_i$

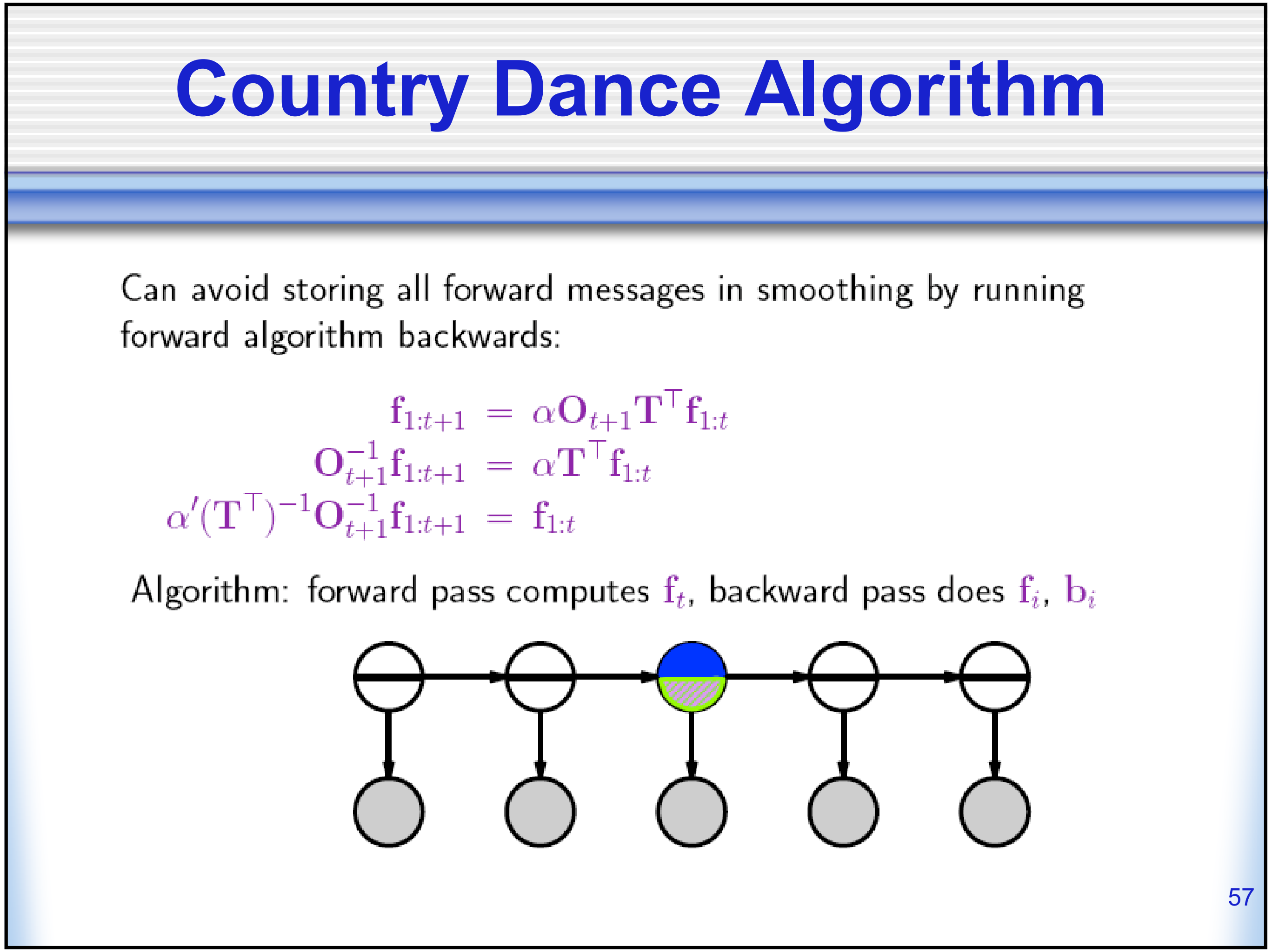


# Country Dance Algorithm

Can avoid storing all forward messages in smoothing by running forward algorithm backwards:

$$\begin{aligned} \mathbf{f}_{1:t+1} &= \alpha \mathbf{O}_{t+1} \mathbf{T}^\top \mathbf{f}_{1:t} \\ \mathbf{O}_{t+1}^{-1} \mathbf{f}_{1:t+1} &= \alpha \mathbf{T}^\top \mathbf{f}_{1:t} \\ \alpha' (\mathbf{T}^\top)^{-1} \mathbf{O}_{t+1}^{-1} \mathbf{f}_{1:t+1} &= \mathbf{f}_{1:t} \end{aligned}$$

Algorithm: forward pass computes $\mathbf{f}_t$, backward pass does $\mathbf{f}_i$, $\mathbf{b}_i$

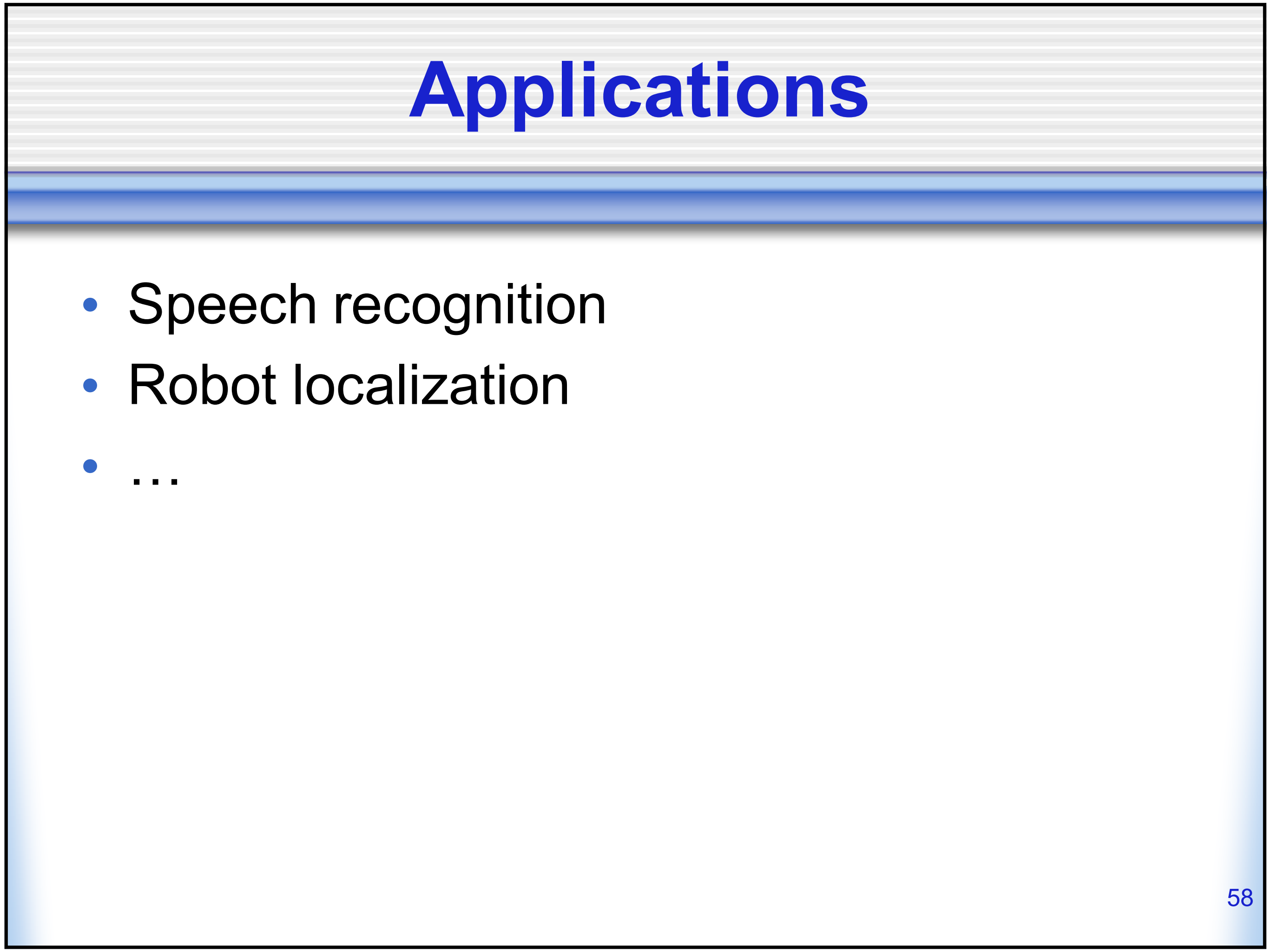


# Country Dance Algorithm

Can avoid storing all forward messages in smoothing by running forward algorithm backwards:

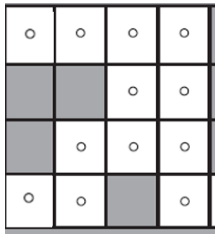
$$
\begin{aligned}
\mathbf{f}_{1:t+1} &= \alpha \mathbf{O}_{t+1} \mathbf{T}^{\top} \mathbf{f}_{1:t} \\
\mathbf{O}_{t+1}^{-1} \mathbf{f}_{1:t+1} &= \alpha \mathbf{T}^{\top} \mathbf{f}_{1:t} \\
\alpha' (\mathbf{T}^{\top})^{-1} \mathbf{O}_{t+1}^{-1} \mathbf{f}_{1:t+1} &= \mathbf{f}_{1:t}
\end{aligned}
$$

Algorithm: forward pass computes $\mathbf{f}_t$, backward pass does $\mathbf{f}_i$, $\mathbf{b}_i$



# Applications

- Speech recognition
- Robot localization
- …



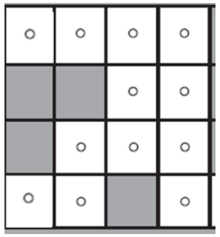
One non-deterministic operation MOVE.

$$P(X_{t+1}=j \mid X_t=i) = \mathbf{T}_{ij} = (1/N(i) \text{ if } j \in \textsc{Neighbors}(i) \text{ else } 0)$$

$E_t$ has 16 possible values, each a four-bit sequence giving the presence or absence of an obstacle: NSWE.
$\varepsilon$ is the error rate. All four bits right $(1-\varepsilon)^4$. All wrong $\varepsilon^{4.}$

$d_{it}$ is the number of bits that are different between the true values for square *i* and the actual reading $e_t$, then the probability that a robot in square *i* would receive a sensor reading $e_t$ is:

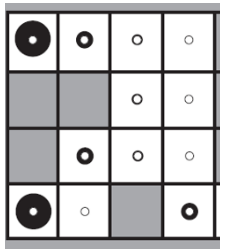
$$P(E_t=e_t \mid X_t=i) = \mathbf{O}_{t_{ii}} = (1-\epsilon)^{4-d_{it}}\epsilon^{d_{it}}$$



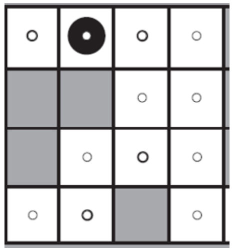
Cell numbers: start in top row, left to right

Matrix for **NSW**

| | 1 | 2 | 3 | … | 12 |
|---|---|---|---|---|---|
| 1 | $(1-\varepsilon)^4$ | | | | |
| 2 | | $(1-\varepsilon)^3\ \varepsilon$ | | | |
| 3 | | | $(1-\varepsilon)^2\ \varepsilon^2$ | | |
| … | | | | …. | |
| 12 | | | | | $(1-\varepsilon)^2\ \varepsilon^2$ |

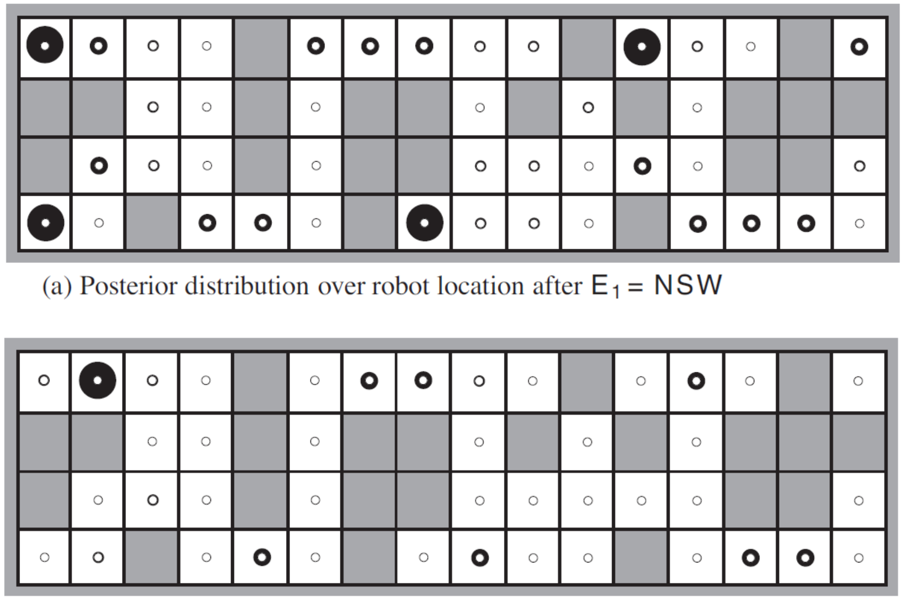


$E_1$=NSW



$E_2$=NS

# Example AIMA



(a) Posterior distribution over robot location after $E_1 = NSW$

(b) Posterior distribution over robot location after $E_1 = NSW, E_2 = NS$

# Performance

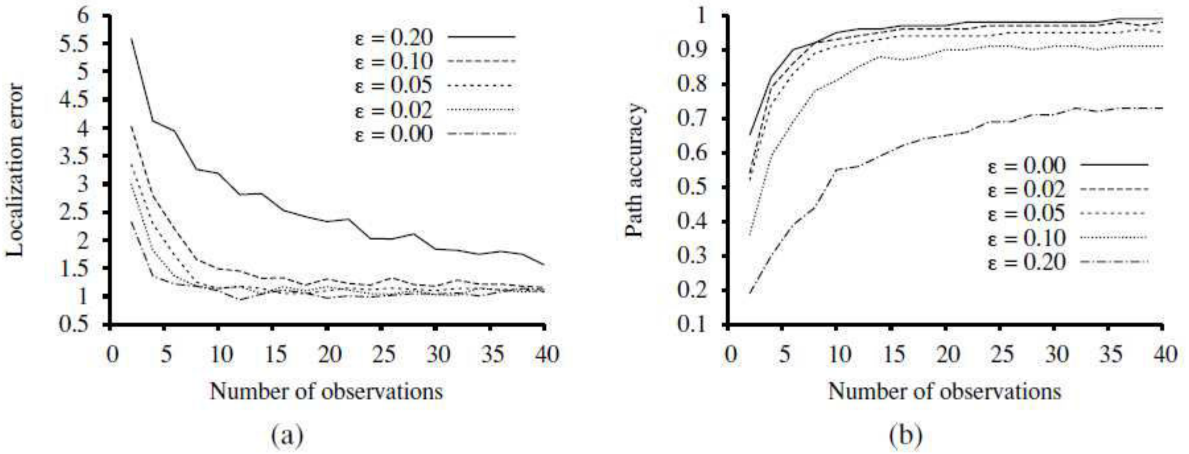
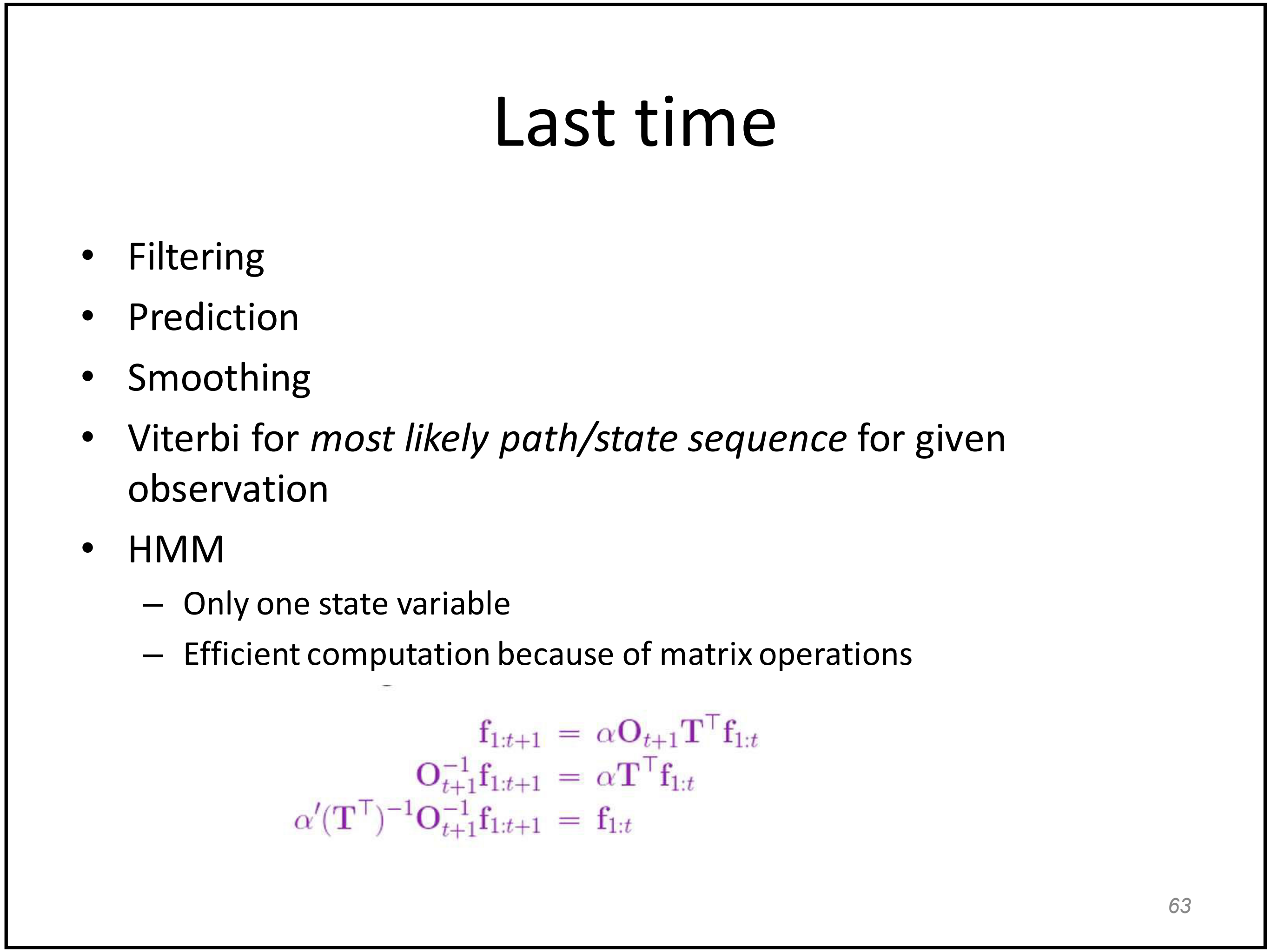


**Figure 15.8** Performance of HMM localization as a function of the length of the observation sequence for various different values of the sensor error probability $\epsilon$; data averaged over 400 runs. (a) The localization error, defined as the Manhattan distance from the true location. (b) The Viterbi path accuracy, defined as the fraction of correct states on the Viterbi path.

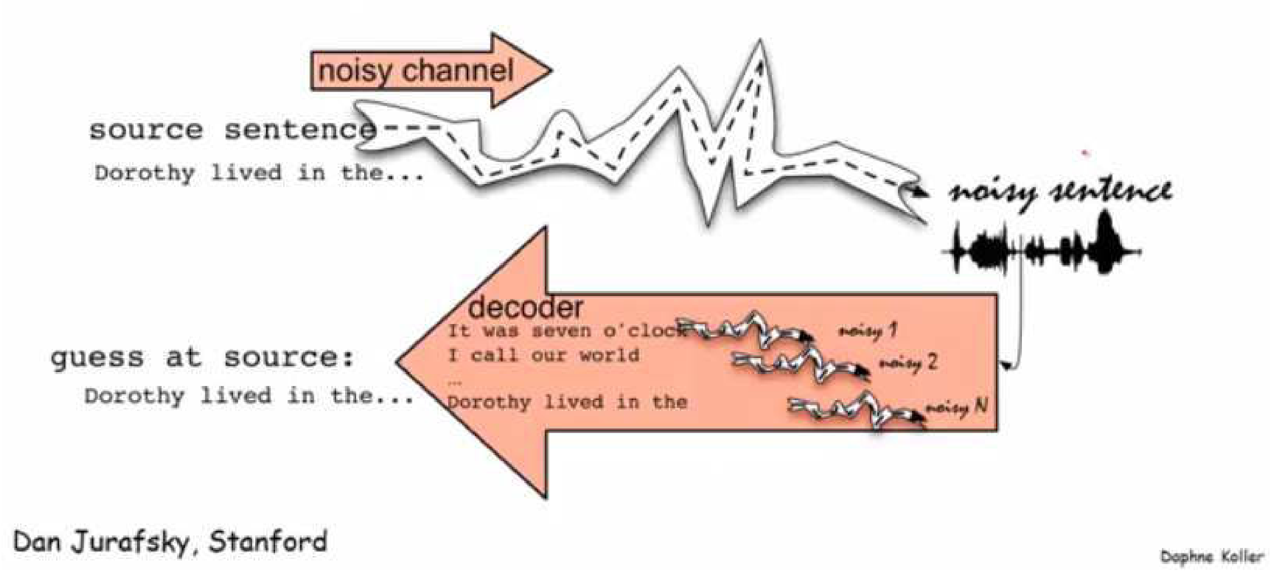


# Last time

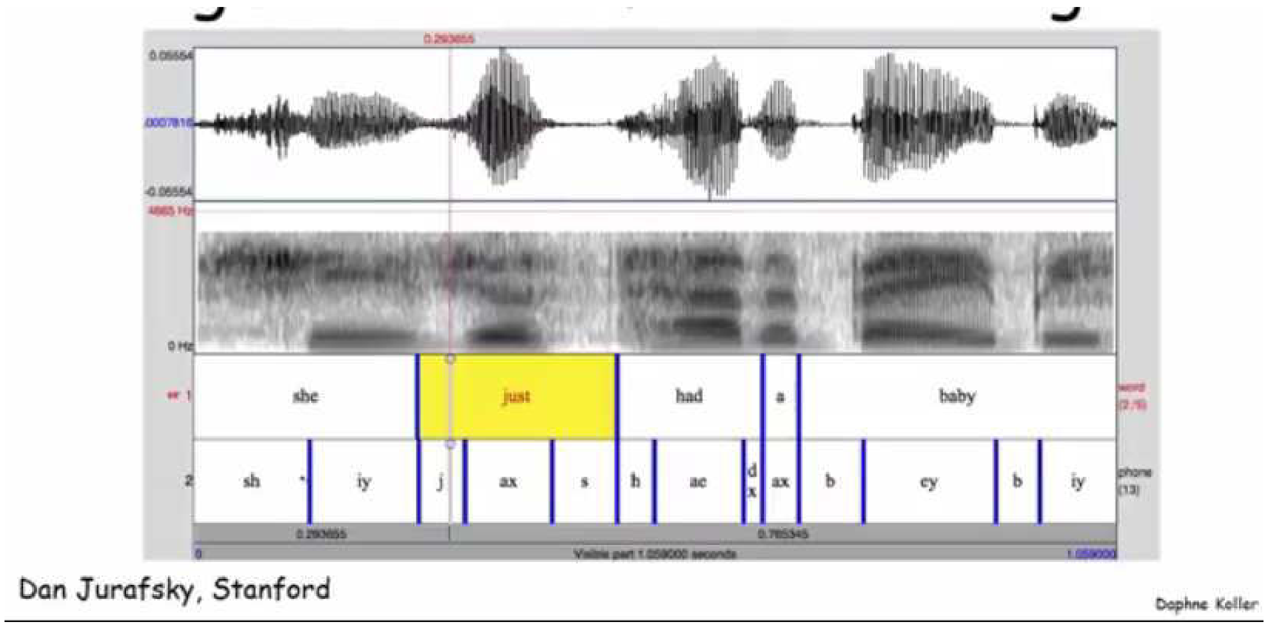
- Filtering
- Prediction
- Smoothing
- Viterbi for *most likely path/state sequence* for given observation
- HMM
  - Only one state variable
  - Efficient computation because of matrix operations

$$\begin{aligned} \mathbf{f}_{1:t+1} &= \alpha \mathbf{O}_{t+1}\mathbf{T}^{\top}\mathbf{f}_{1:t} \\ \mathbf{O}_{t+1}^{-1}\mathbf{f}_{1:t+1} &= \alpha \mathbf{T}^{\top}\mathbf{f}_{1:t} \\ \alpha'(\mathbf{T}^{\top})^{-1}\mathbf{O}_{t+1}^{-1}\mathbf{f}_{1:t+1} &= \mathbf{f}_{1:t} \end{aligned}$$

# Speech recognition



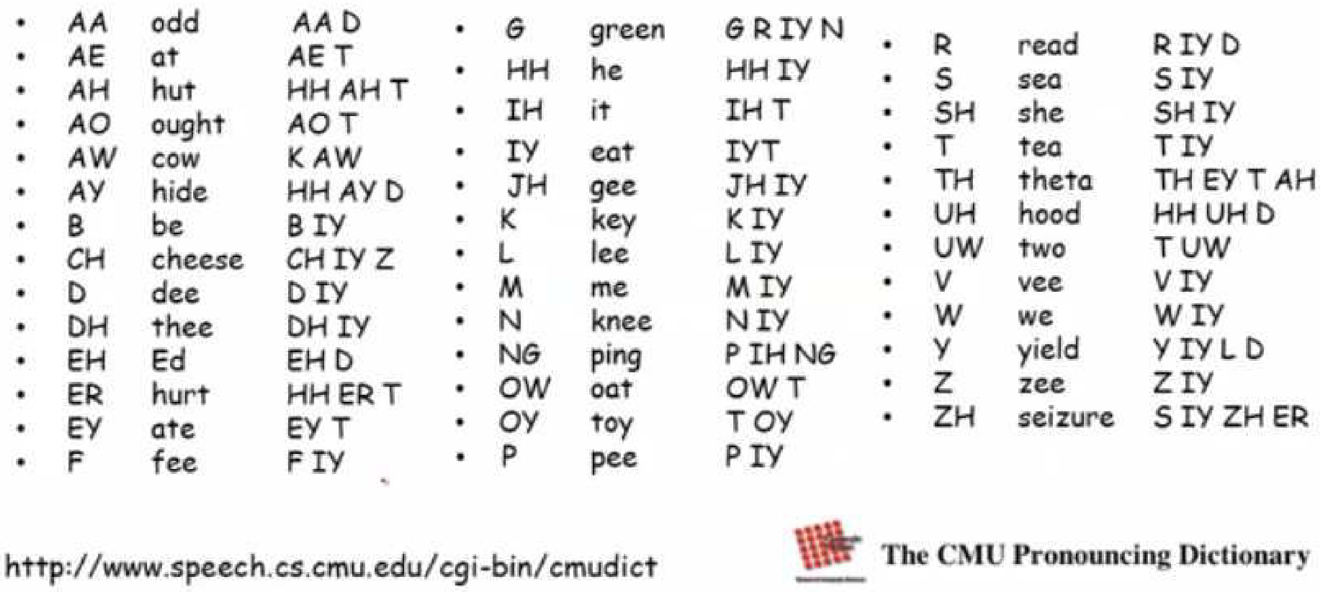
# Segmentation of Acoustic signals



Dan Jurafsky, Stanford

Daphne Koller

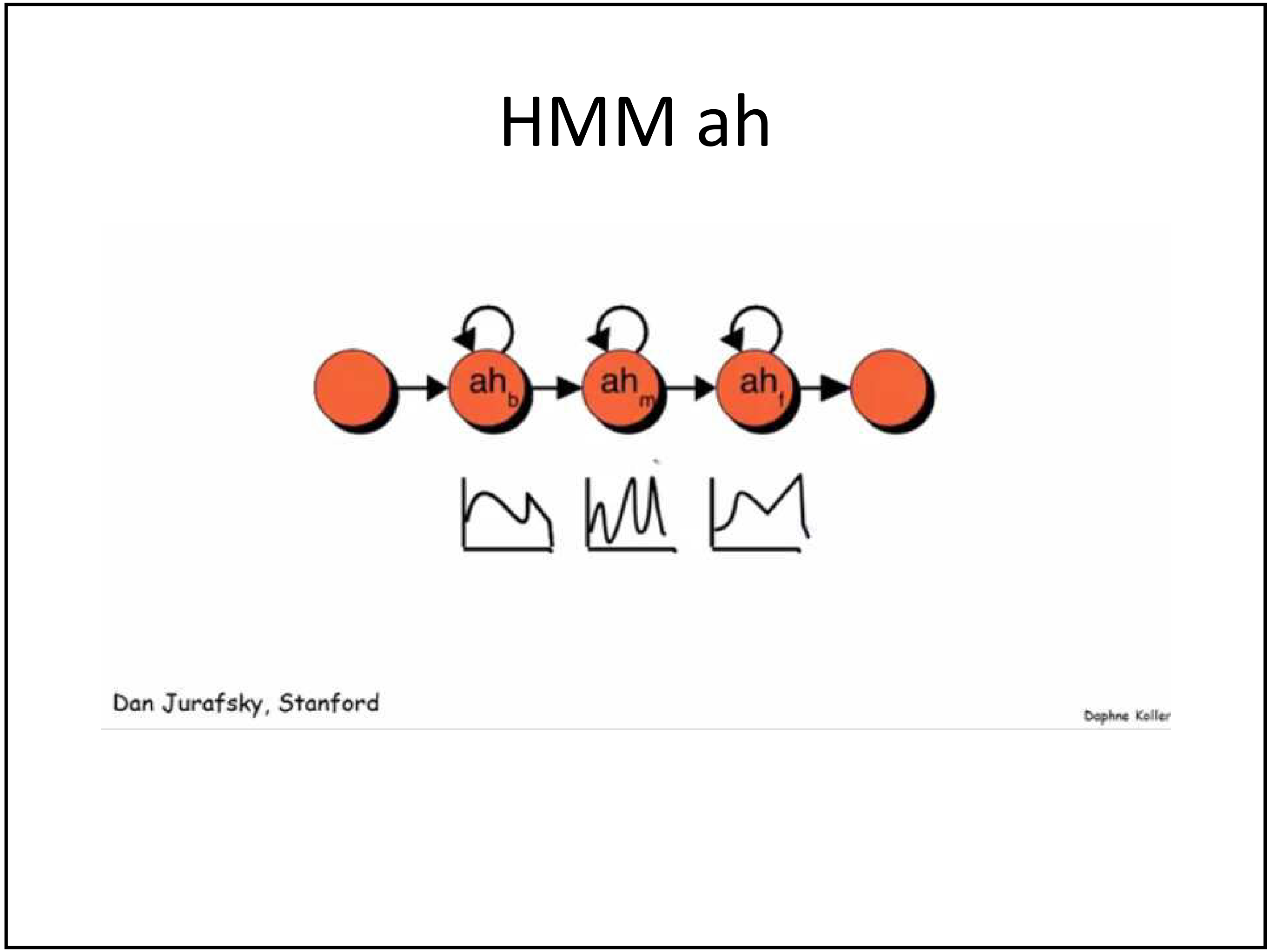
# Phonetic alphabet



| | | | | | | | | |
|---|---|---|---|---|---|---|---|---|
| AA | odd | AA D | G | green | G R IY N | R | read | R IY D |
| AE | at | AE T | HH | he | HH IY | S | sea | S IY |
| AH | hut | HH AH T | IH | it | IH T | SH | she | SH IY |
| AO | ought | AO T | IY | eat | IY T | T | tea | T IY |
| AW | cow | K AW | JH | gee | JH IY | TH | theta | TH EY T AH |
| AY | hide | HH AY D | K | key | K IY | UH | hood | HH UH D |
| B | be | B IY | L | lee | L IY | UW | two | T UW |
| CH | cheese | CH IY Z | M | me | M IY | V | vee | V IY |
| D | dee | D IY | N | knee | N IY | W | we | W IY |
| DH | thee | DH IY | NG | ping | P IH NG | Y | yield | Y IY L D |
| EH | Ed | EH D | OW | oat | OW T | Z | zee | Z IY |
| ER | hurt | HH ER T | OY | toy | T OY | ZH | seizure | S IY ZH ER |
| EY | ate | EY T | P | pee | P IY | | | |
| F | fee | F IY | | | | | | |

http://www.speech.cs.cmu.edu/cgi-bin/cmudict

The CMU Pronouncing Dictionary

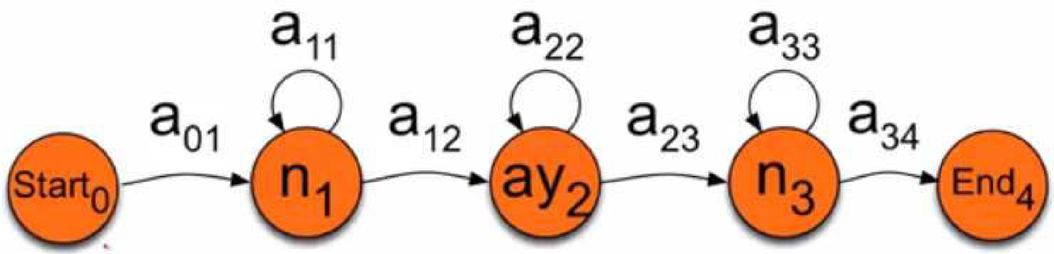


# HMM ah

Dan Jurafsky, Stanford

Daphne Koller

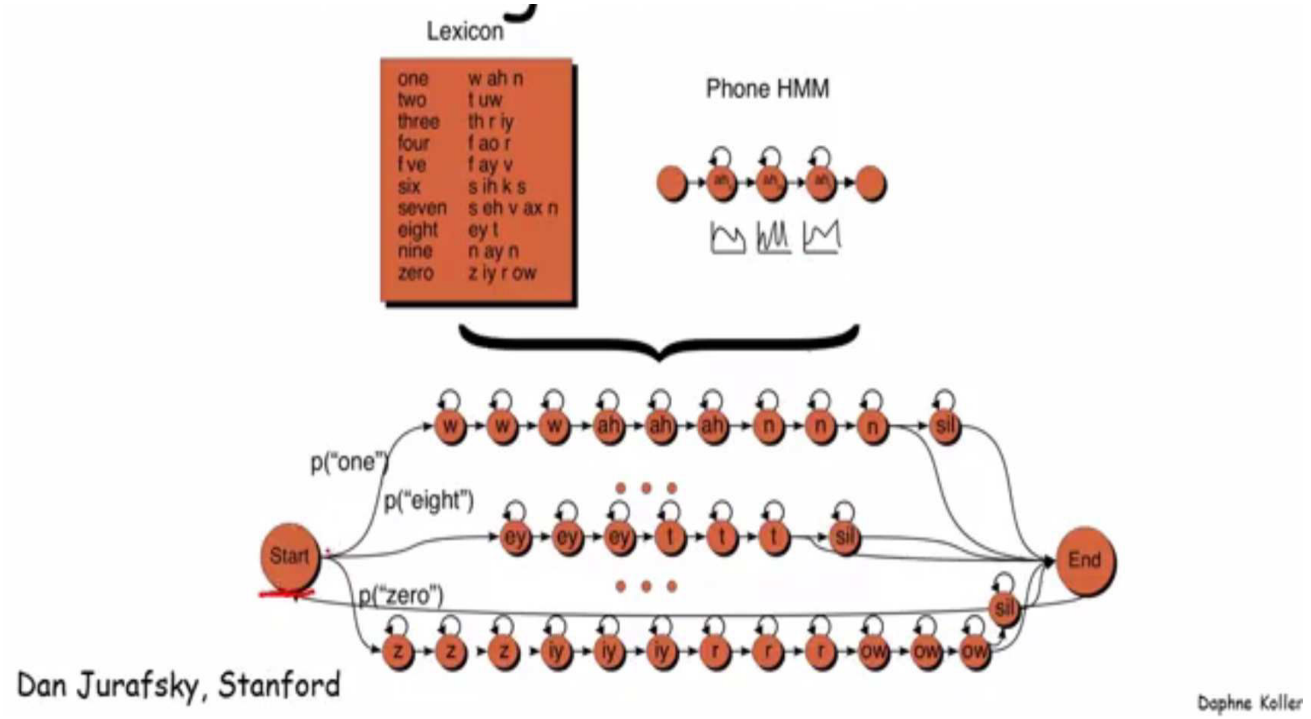
# Word HMM: nine



Dan Jurafsky, Stanford

Daphne Koller

# Recognition HMM

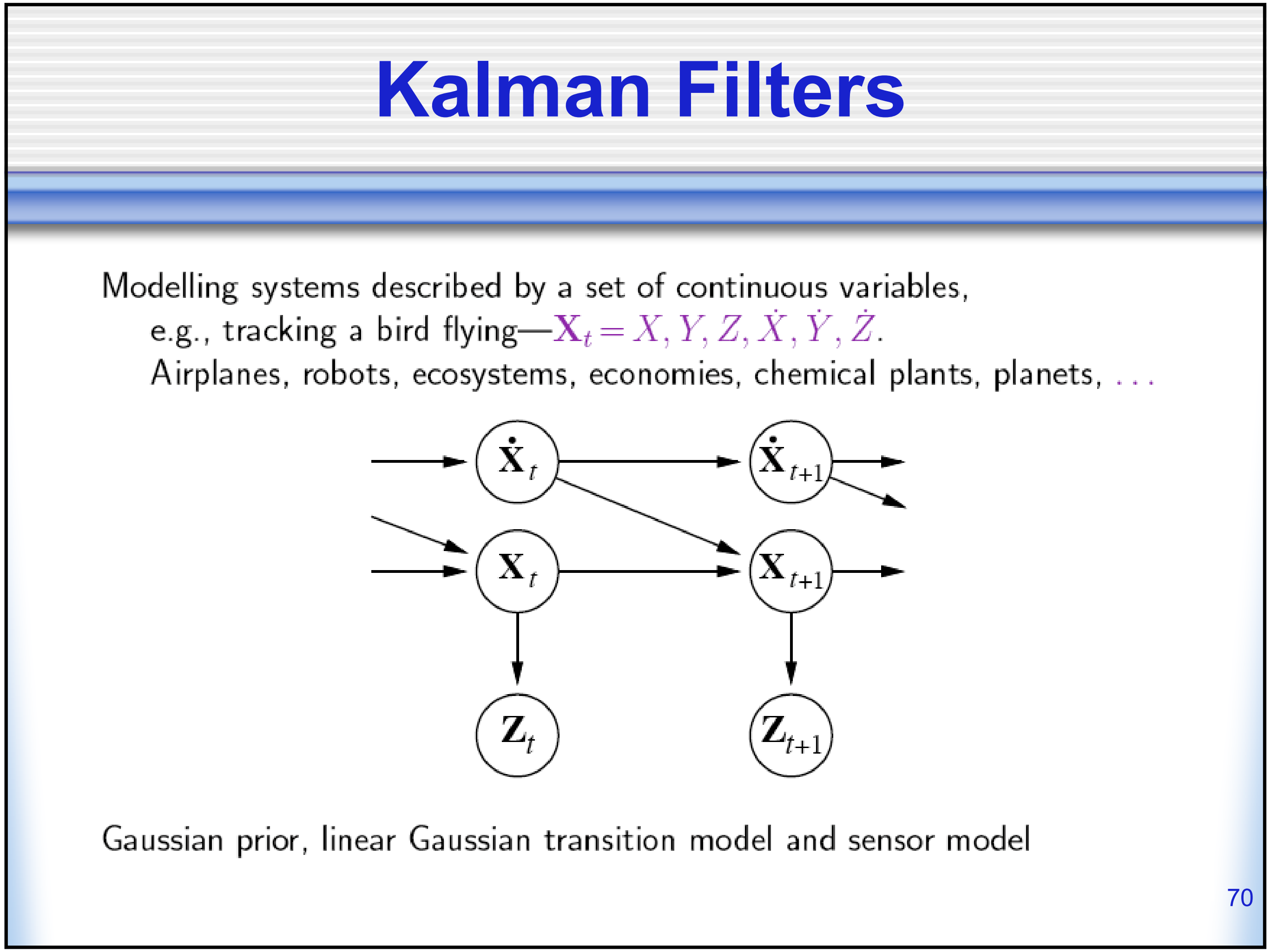


Lexicon

| | |
|---|---|
| one | w ah n |
| two | t uw |
| three | th r iy |
| four | f ao r |
| five | f ay v |
| six | s ih k s |
| seven | s eh v ax n |
| eight | ey t |
| nine | n ay n |
| zero | z iy r ow |

Dan Jurafsky, Stanford

Daphne Koller



# Kalman Filters

Modelling systems described by a set of continuous variables,

e.g., tracking a bird flying—$\mathbf{X}_t = X, Y, Z, \dot{X}, \dot{Y}, \dot{Z}$.

Airplanes, robots, ecosystems, economies, chemical plants, planets, . . .

Gaussian prior, linear Gaussian transition model and sensor model

# Updating Gaussian Distributions

Prediction step: if $\mathbf{P}(\mathbf{X}_t|\mathbf{e}_{1:t})$ is Gaussian, then prediction

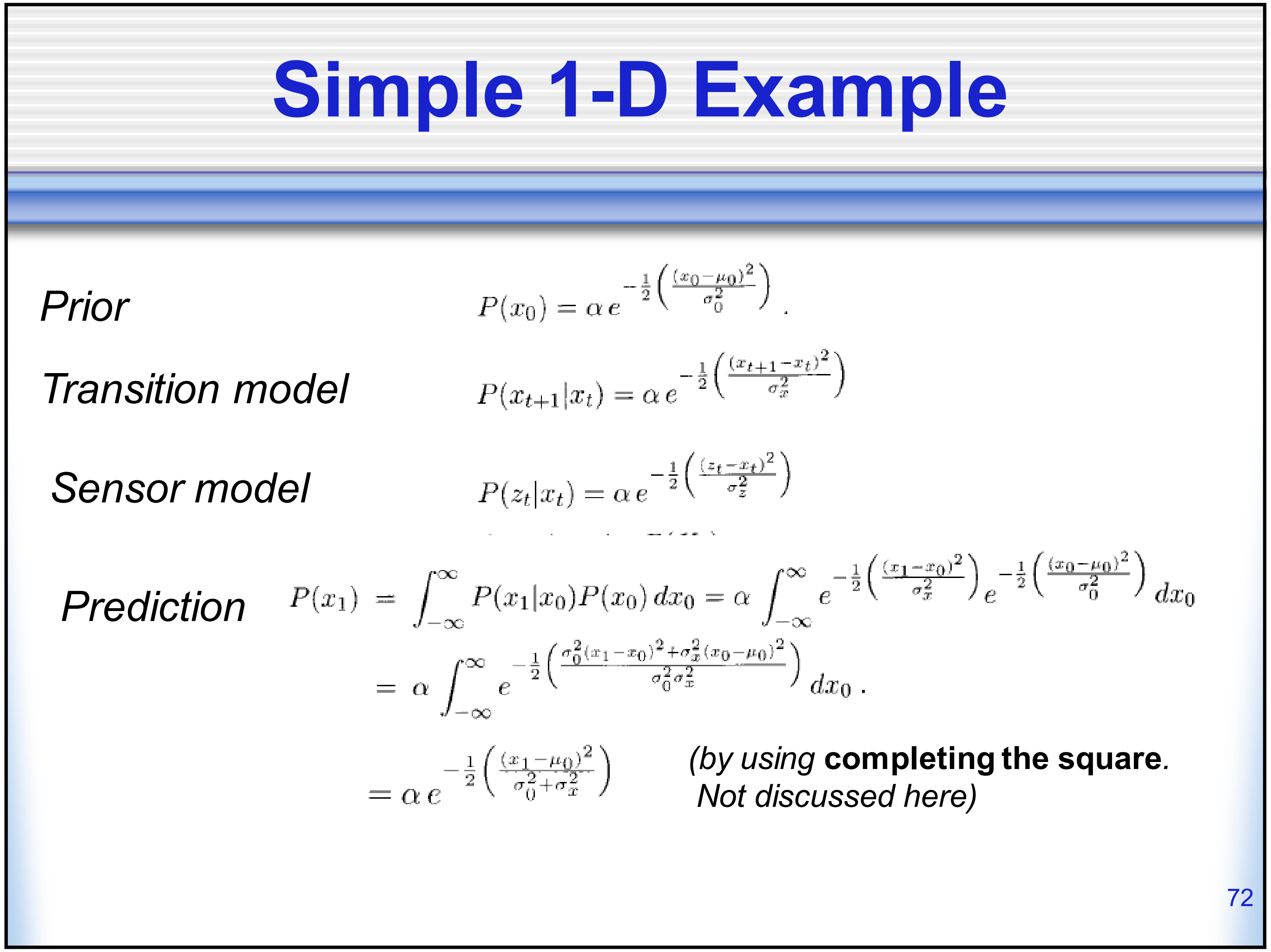
$$\mathbf{P}(\mathbf{X}_{t+1}|\mathbf{e}_{1:t}) = \int_{\mathbf{x}_t} \mathbf{P}(\mathbf{X}_{t+1}|\mathbf{x}_t)P(\mathbf{x}_t|\mathbf{e}_{1:t})\, d\mathbf{x}_t$$

is Gaussian. If $\mathbf{P}(\mathbf{X}_{t+1}|\mathbf{e}_{1:t})$ is Gaussian, then the updated distribution

$$\mathbf{P}(\mathbf{X}_{t+1}|\mathbf{e}_{1:t+1}) = \alpha\mathbf{P}(\mathbf{e}_{t+1}|\mathbf{X}_{t+1})\mathbf{P}(\mathbf{X}_{t+1}|\mathbf{e}_{1:t})$$

is Gaussian

Hence $\mathbf{P}(\mathbf{X}_t|\mathbf{e}_{1:t})$ is multivariate Gaussian $N(\boldsymbol{\mu}_t, \boldsymbol{\Sigma}_t)$ for all $t$



# Simple 1-D Example

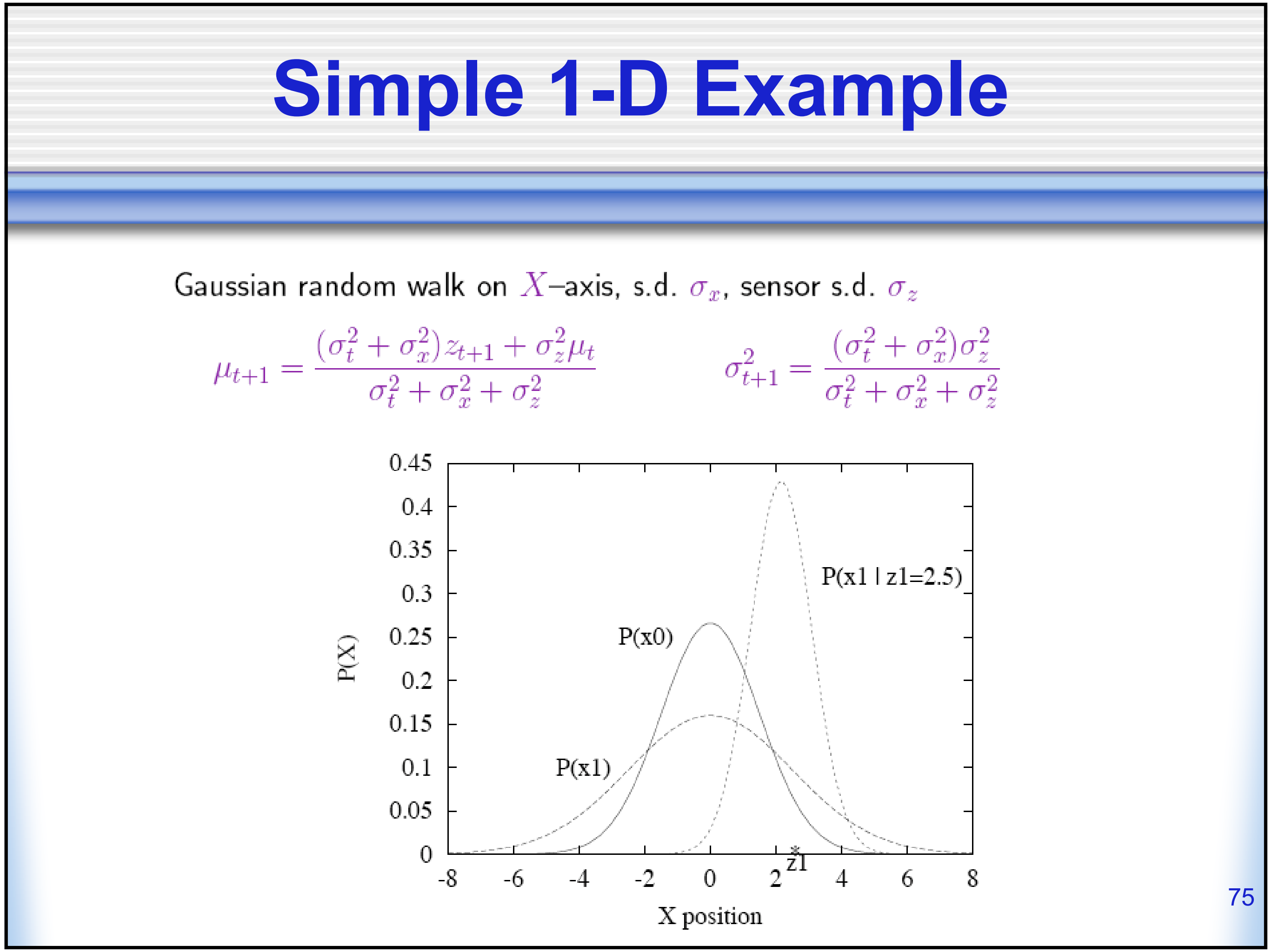
*Prior* $$P(x_0) = \alpha\, e^{-\frac{1}{2}\left(\frac{(x_0-\mu_0)^2}{\sigma_0^2}\right)} .$$

*Transition model* $$P(x_{t+1}|x_t) = \alpha\, e^{-\frac{1}{2}\left(\frac{(x_{t+1}-x_t)^2}{\sigma_x^2}\right)}$$

*Sensor model* $$P(z_t|x_t) = \alpha\, e^{-\frac{1}{2}\left(\frac{(z_t-x_t)^2}{\sigma_z^2}\right)}$$

*Prediction*
$$\begin{aligned} P(x_1) &= \int_{-\infty}^{\infty} P(x_1|x_0)P(x_0)\, dx_0 = \alpha \int_{-\infty}^{\infty} e^{-\frac{1}{2}\left(\frac{(x_1-x_0)^2}{\sigma_x^2}\right)} e^{-\frac{1}{2}\left(\frac{(x_0-\mu_0)^2}{\sigma_0^2}\right)} dx_0 \\ &= \alpha \int_{-\infty}^{\infty} e^{-\frac{1}{2}\left(\frac{\sigma_0^2(x_1-x_0)^2+\sigma_x^2(x_0-\mu_0)^2}{\sigma_0^2\sigma_x^2}\right)} dx_0 . \\ &= \alpha\, e^{-\frac{1}{2}\left(\frac{(x_1-\mu_0)^2}{\sigma_0^2+\sigma_x^2}\right)} \end{aligned}$$

*(by using* **completing the square**. *Not discussed here)*

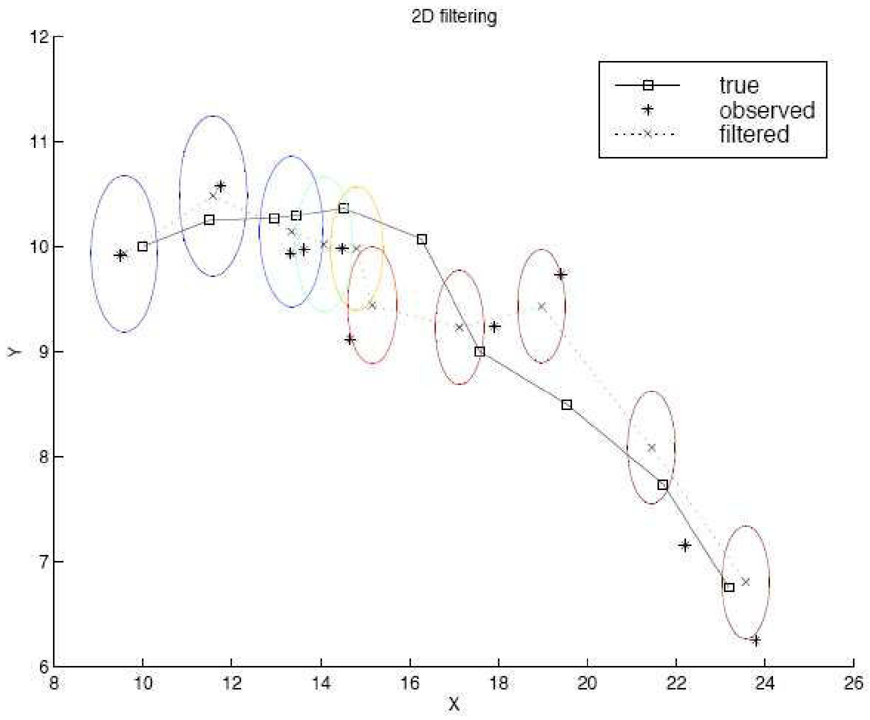


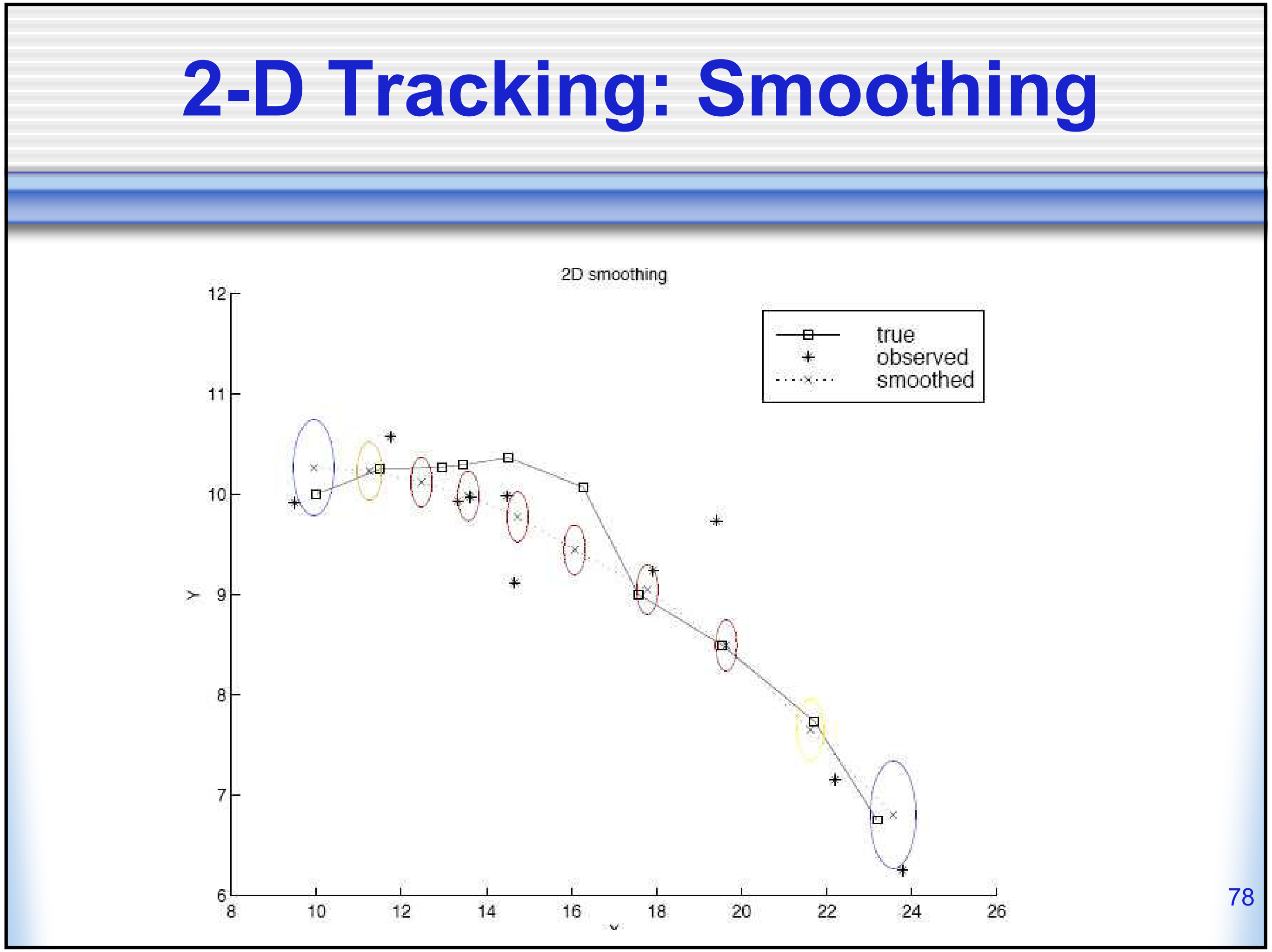
# Simple 1-D Example

Gaussian random walk on $X$–axis, s.d. $\sigma_x$, sensor s.d. $\sigma_z$

$$\mu_{t+1} = \frac{(\sigma_t^2 + \sigma_x^2) z_{t+1} + \sigma_z^2 \mu_t}{\sigma_t^2 + \sigma_x^2 + \sigma_z^2} \qquad \sigma_{t+1}^2 = \frac{(\sigma_t^2 + \sigma_x^2)\sigma_z^2}{\sigma_t^2 + \sigma_x^2 + \sigma_z^2}$$

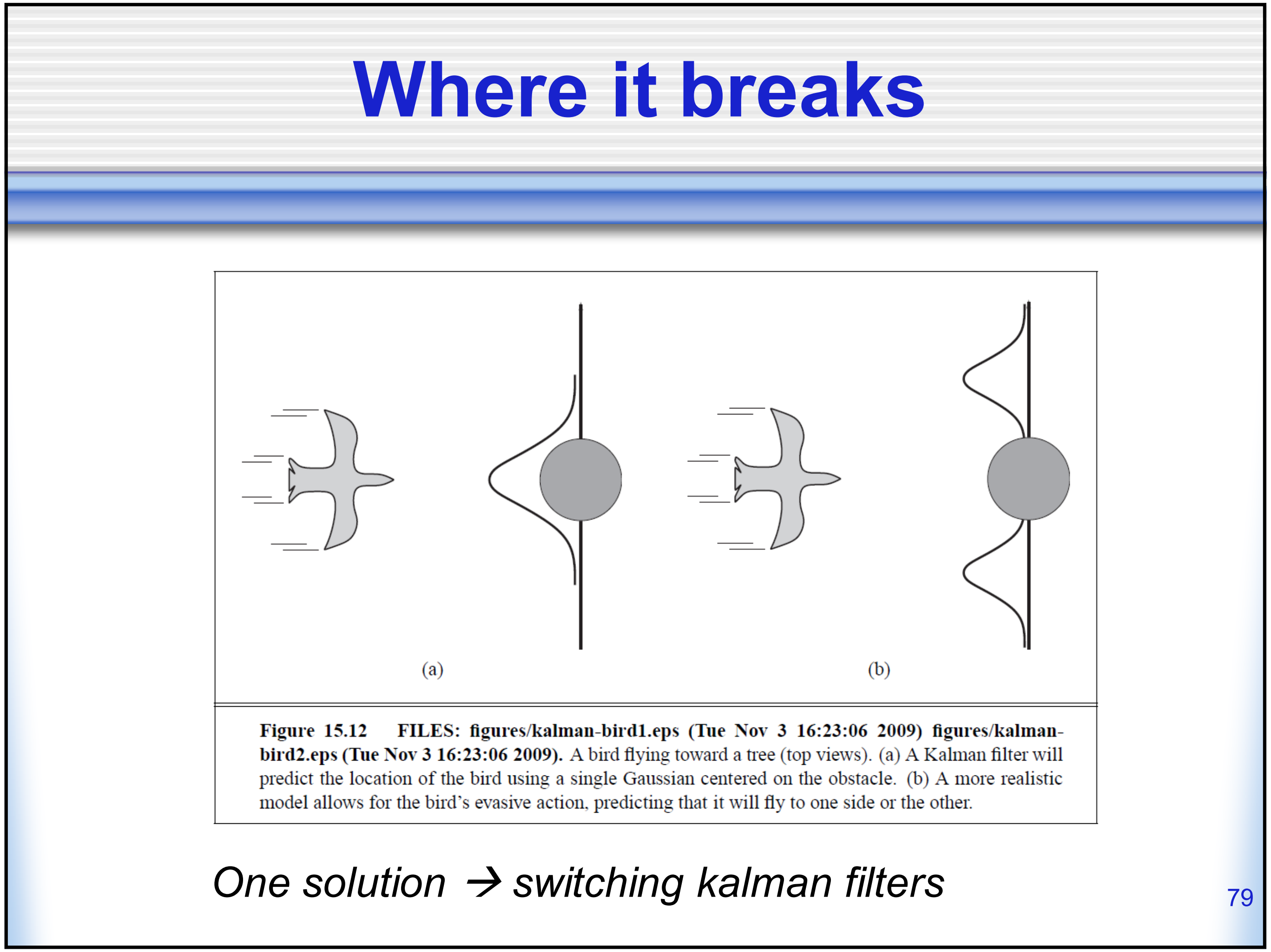
# 2-D Tracking: Filtering





# 2-D Tracking: Smoothing

78



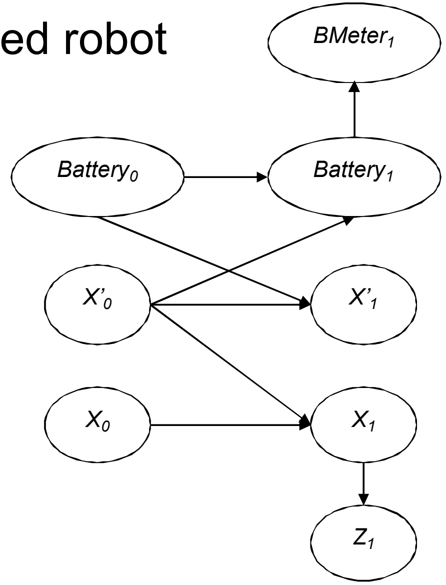
# Where it breaks

**Figure 15.12 FILES: figures/kalman-bird1.eps (Tue Nov 3 16:23:06 2009) figures/kalman-bird2.eps (Tue Nov 3 16:23:06 2009).** A bird flying toward a tree (top views). (a) A Kalman filter will predict the location of the bird using a single Gaussian centered on the obstacle. (b) A more realistic model allows for the bird's evasive action, predicting that it will fly to one side or the other.

*One solution → switching kalman filters*

# Creating DBNs with failures

- $\mathbf{X'}_t$ =( $X_t$, $Y_t$) for velocity $\mathbf{X}_t$ =($X_t$, $Y_t$) for position
- Battery powered robot

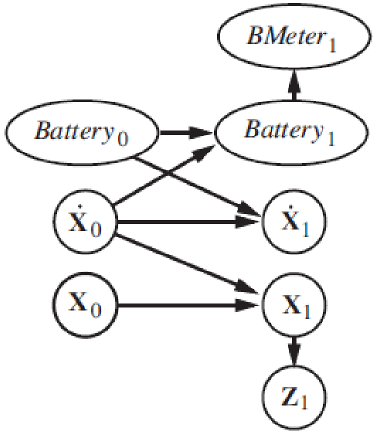


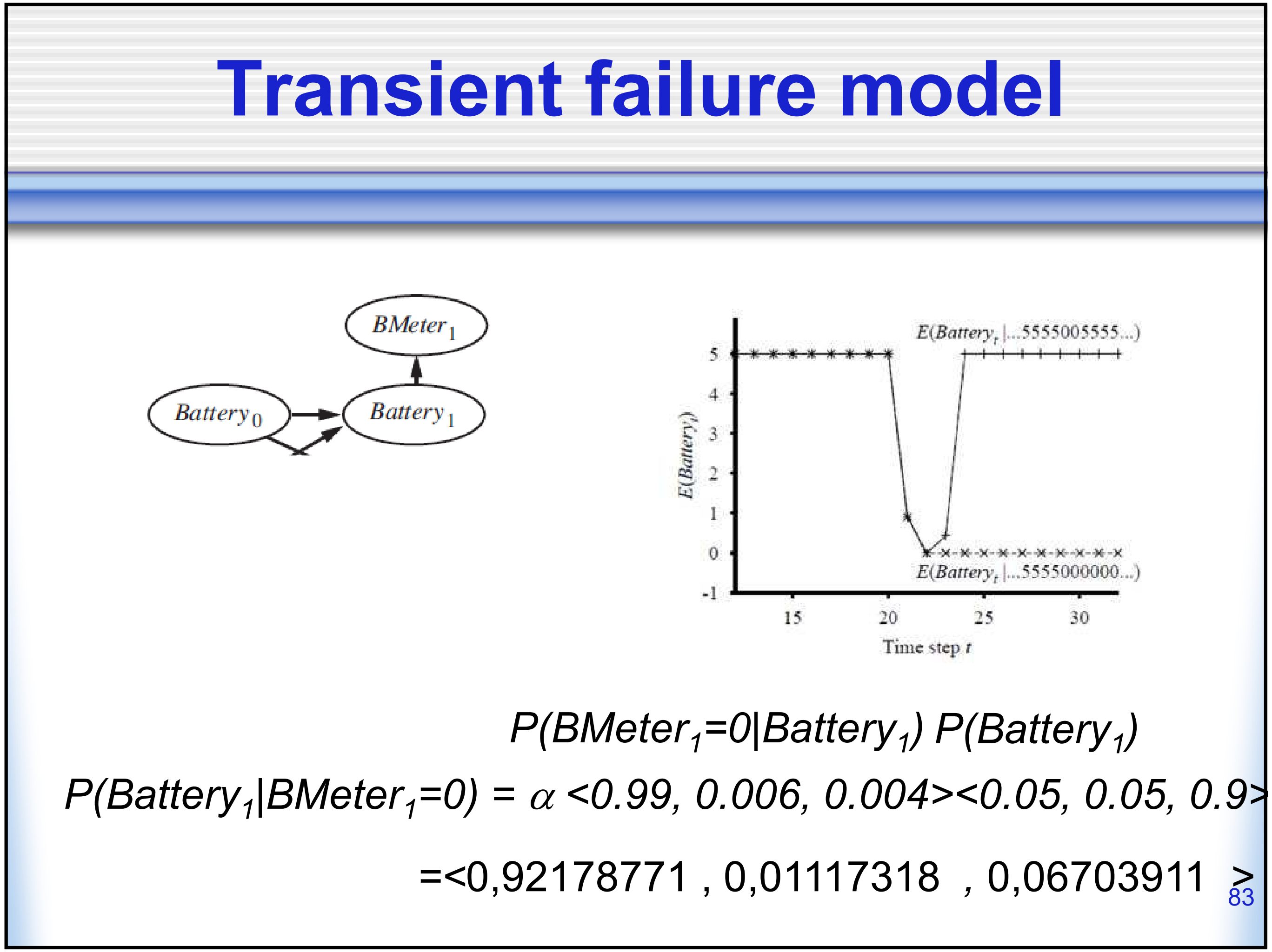
81

# Failure of sensors

- Sensor measurements are noisy
- Real sensors can fail
- May use a Gaussian error model for *discrete variables*

- Transient failure
- Persistent failure





# Transient failure model

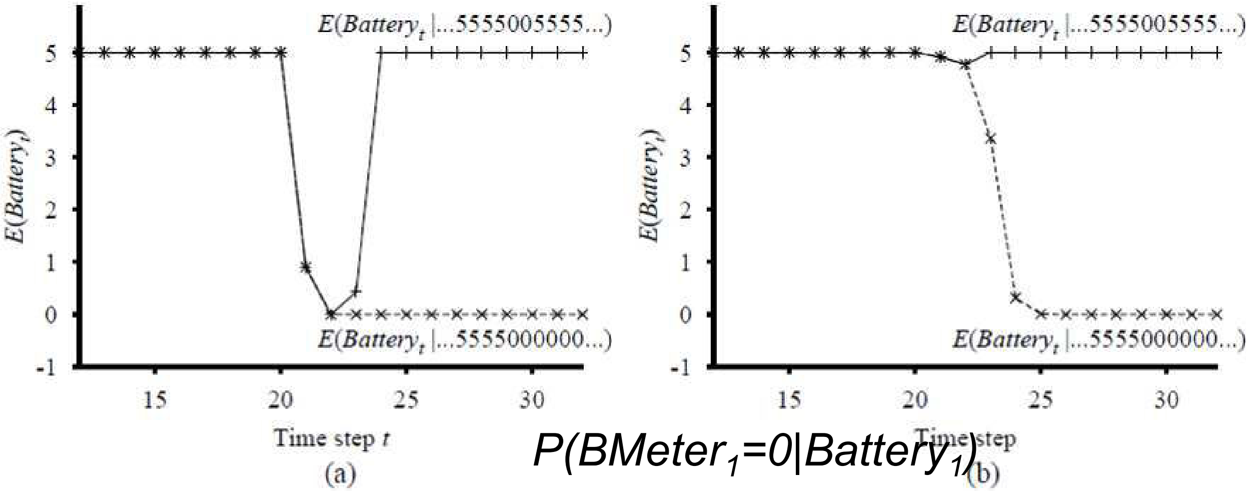
$P(BMeter_1=0|Battery_1)\ P(Battery_1)$

$P(Battery_1|BMeter_1=0) = \alpha$ <0.99, 0.006, 0.004><0.05, 0.05, 0.9>

=<0,92178771 , 0,01117318 , 0,06703911 >

83

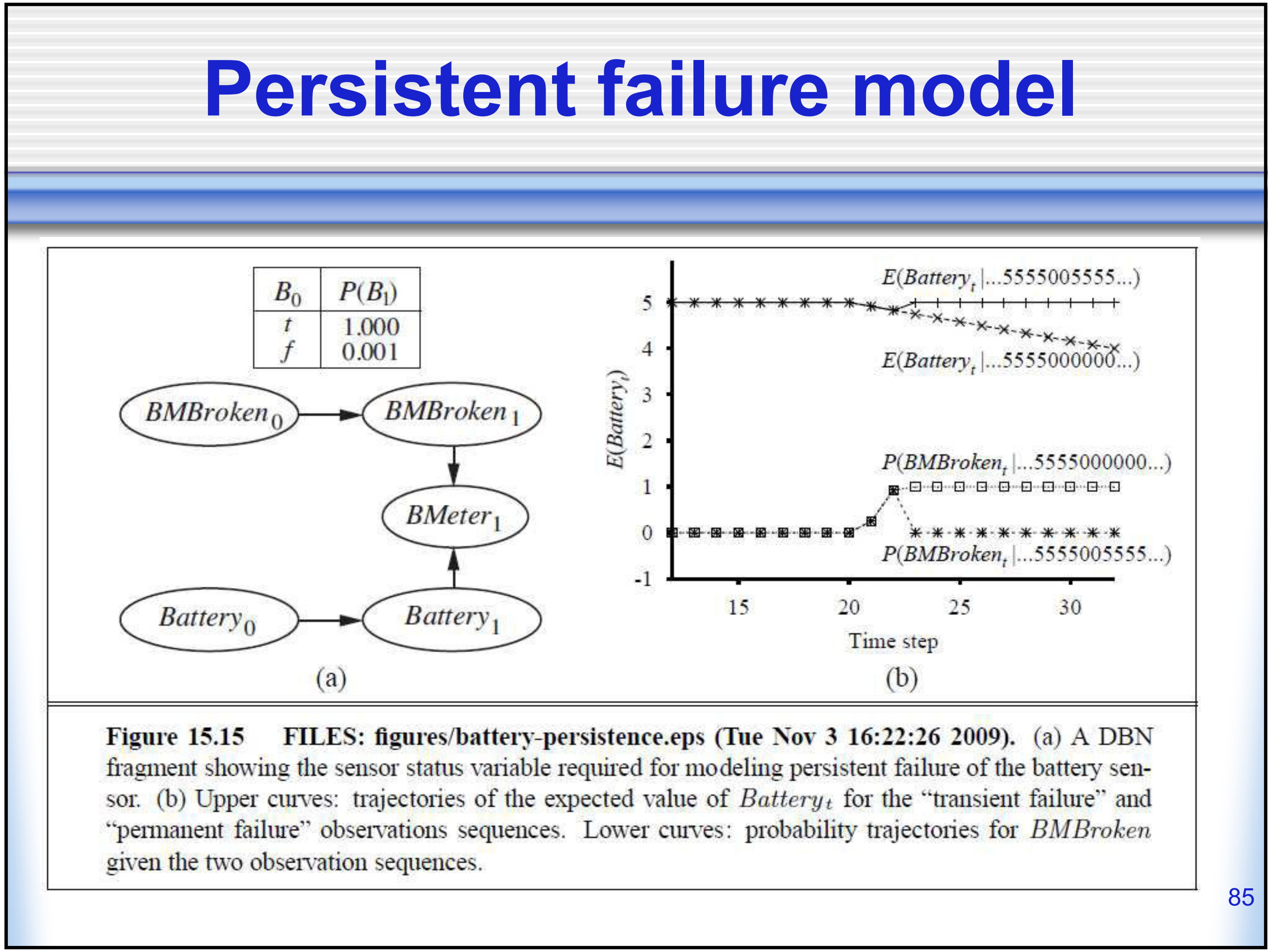
# Transient failure model



$P(BMeter_1=0|Battery_1)$

$P(Battery_1|BMeter_1=0) = \alpha <0.8, 0.1, 0.1><0.05, 0.05, 0.9>$

$\approx <0.44, \; 0.06 \; , 0.5 >$

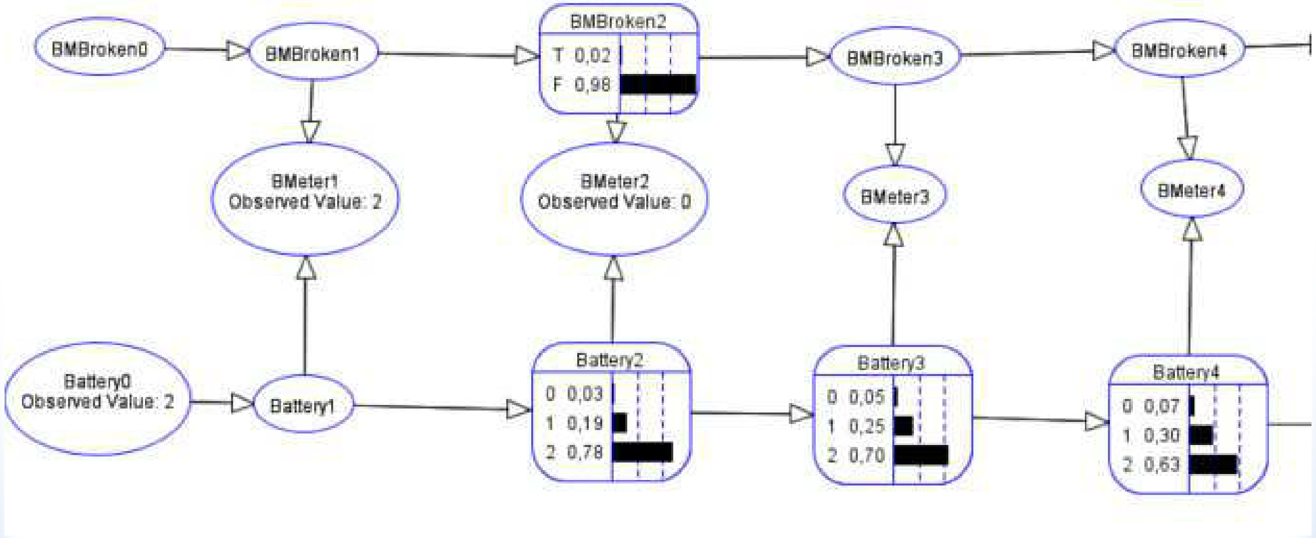


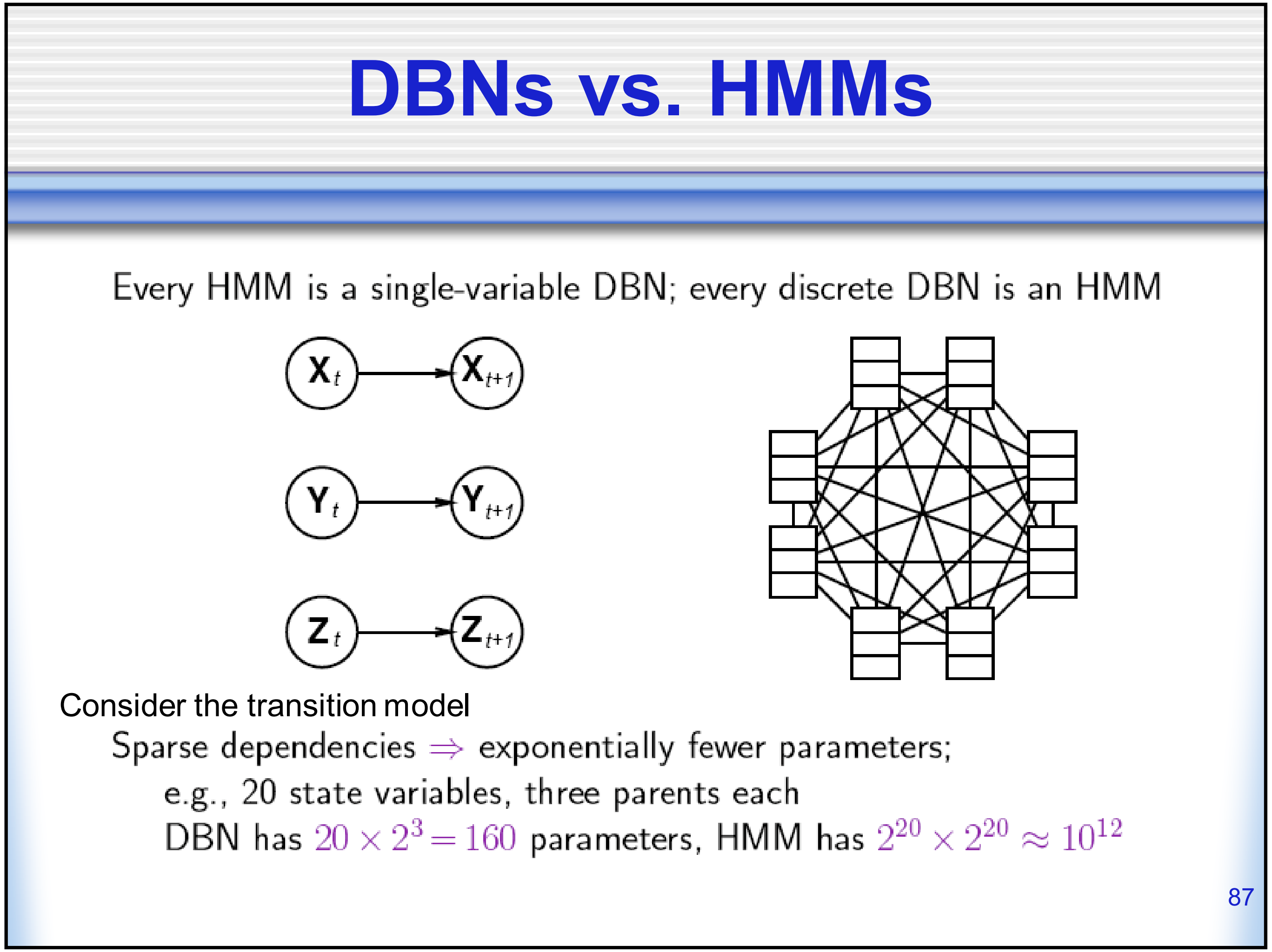
# Persistent failure model

| $B_0$ | $P(B_1)$ |
|---|---|
| *t* | 1.000 |
| *f* | 0.001 |

**Figure 15.15** **FILES: figures/battery-persistence.eps (Tue Nov 3 16:22:26 2009).** (a) A DBN fragment showing the sensor status variable required for modeling persistent failure of the battery sensor. (b) Upper curves: trajectories of the expected value of $Battery_t$ for the "transient failure" and "permanent failure" observations sequences. Lower curves: probability trajectories for $BMBroken$ given the two observation sequences.

# Example


BMBroken0
BMBroken1
BMBroken2
T 0,02
F 0,98
BMBroken3
BMBroken4
BMeter1
Observed Value: 2
BMeter2
Observed Value: 0
BMeter3
BMeter4
Battery0
Observed Value: 2
Battery1
Battery2
0 0,03
1 0,19
2 0,78
Battery3
0 0,05
1 0,25
2 0,70
Battery4
0 0,07
1 0,30
2 0,63



# DBNs vs. HMMs

Every HMM is a single-variable DBN; every discrete DBN is an HMM

$X_t$
$X_{t+1}$
$Y_t$
$Y_{t+1}$
$Z_t$
$Z_{t+1}$

Consider the transition model

Sparse dependencies ⇒ exponentially fewer parameters;

e.g., 20 state variables, three parents each

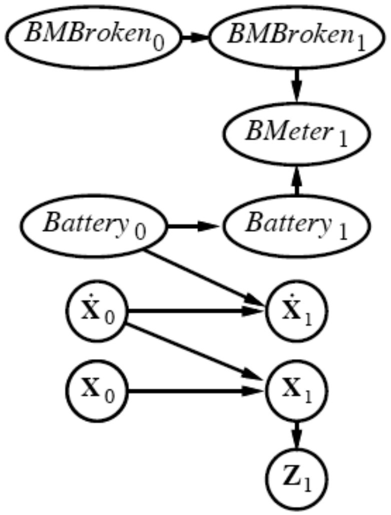
DBN has $20 \times 2^3 = 160$ parameters, HMM has $2^{20} \times 2^{20} \approx 10^{12}$

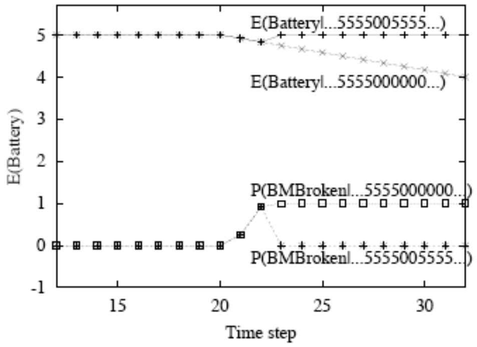
87

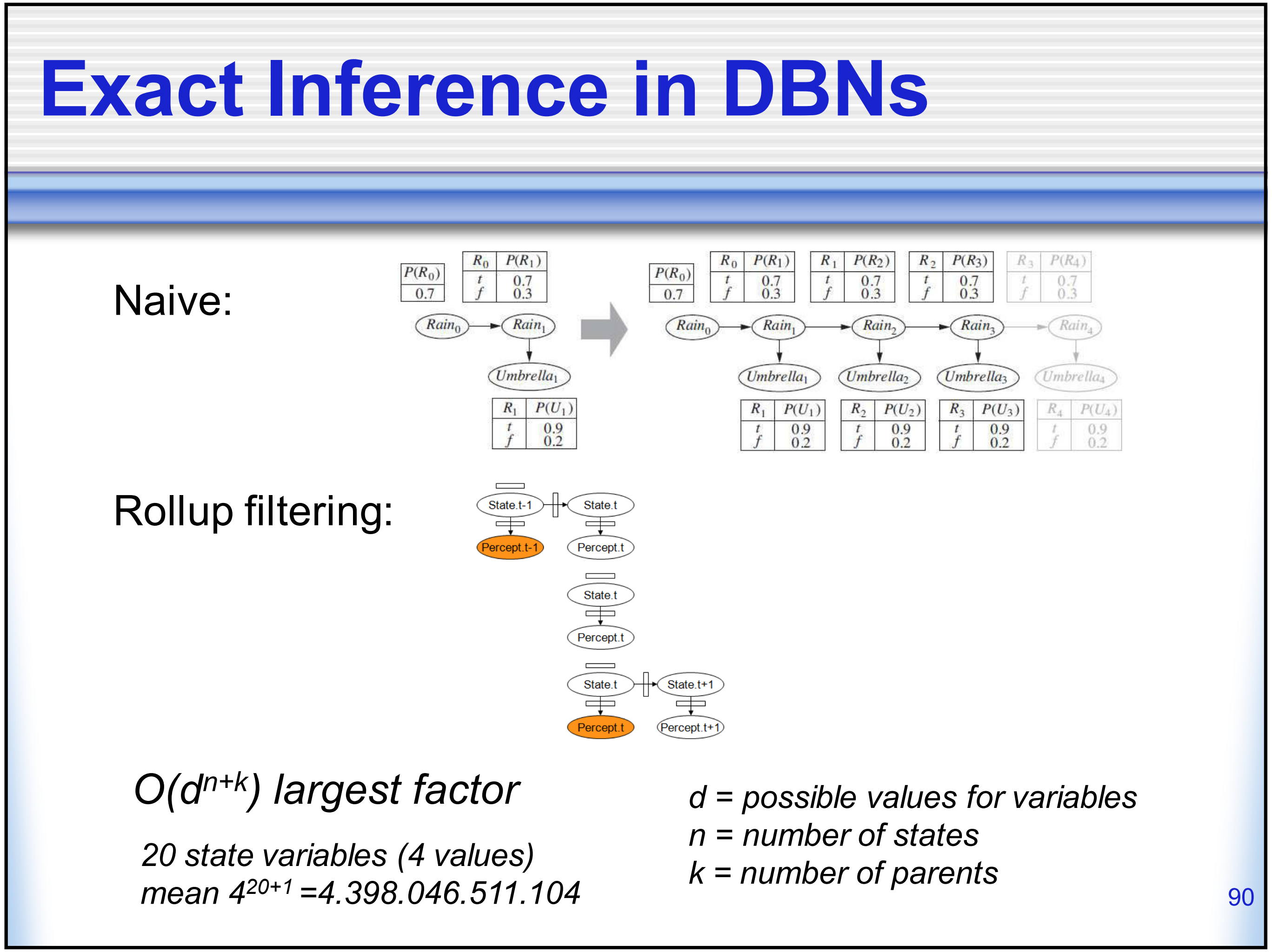
# DBNs vs. Kalman Filters

Every Kalman filter model is a DBN, but few DBNs are KFs;
real world requires non-Gaussian posteriors

E.g., where are bin Laden and my keys? What's the battery charge?







# Exact Inference in DBNs

Naive:

$P(R_0)$ 0.7
$R_0$ $P(R_1)$ t 0.7 f 0.3
$Rain_0$
$Rain_1$
$Umbrella_1$
$R_1$ $P(U_1)$ t 0.9 f 0.2
$P(R_0)$ 0.7
$R_0$ $P(R_1)$ t 0.7 f 0.3
$R_1$ $P(R_2)$ t 0.7 f 0.3
$R_2$ $P(R_3)$ t 0.7 f 0.3
$R_3$ $P(R_4)$ t 0.7 f 0.3
$Rain_0$
$Rain_1$
$Rain_2$
$Rain_3$
$Rain_4$
$Umbrella_1$
$Umbrella_2$
$Umbrella_3$
$Umbrella_4$
$R_1$ $P(U_1)$ t 0.9 f 0.2
$R_2$ $P(U_2)$ t 0.9 f 0.2
$R_3$ $P(U_3)$ t 0.9 f 0.2
$R_4$ $P(U_4)$ t 0.9 f 0.2

Rollup filtering:

State.t-1
State.t
Percept.t-1
Percept.t
State.t
Percept.t
State.t
State.t+1
Percept.t
Percept.t+1

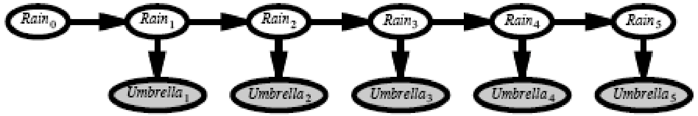
*$O(d^{n+k})$ largest factor*

*20 state variables (4 values)*
*mean $4^{20+1}$ =4.398.046.511.104*

*d = possible values for variables*
*n = number of states*
*k = number of parents*

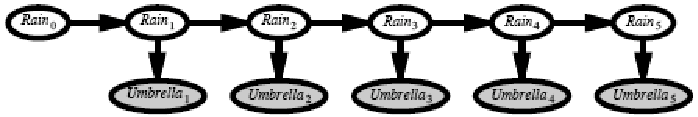
# Approximate inference: Likelihood Weighting

Set of weighted samples approximates the belief state



# Likelihood Weighting

Set of weighted samples approximates the belief state

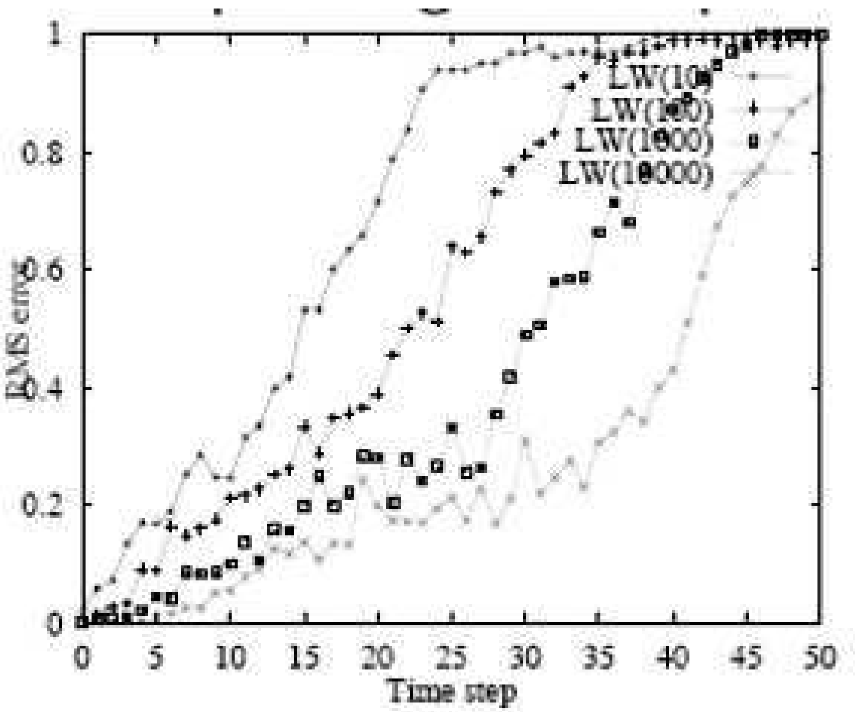


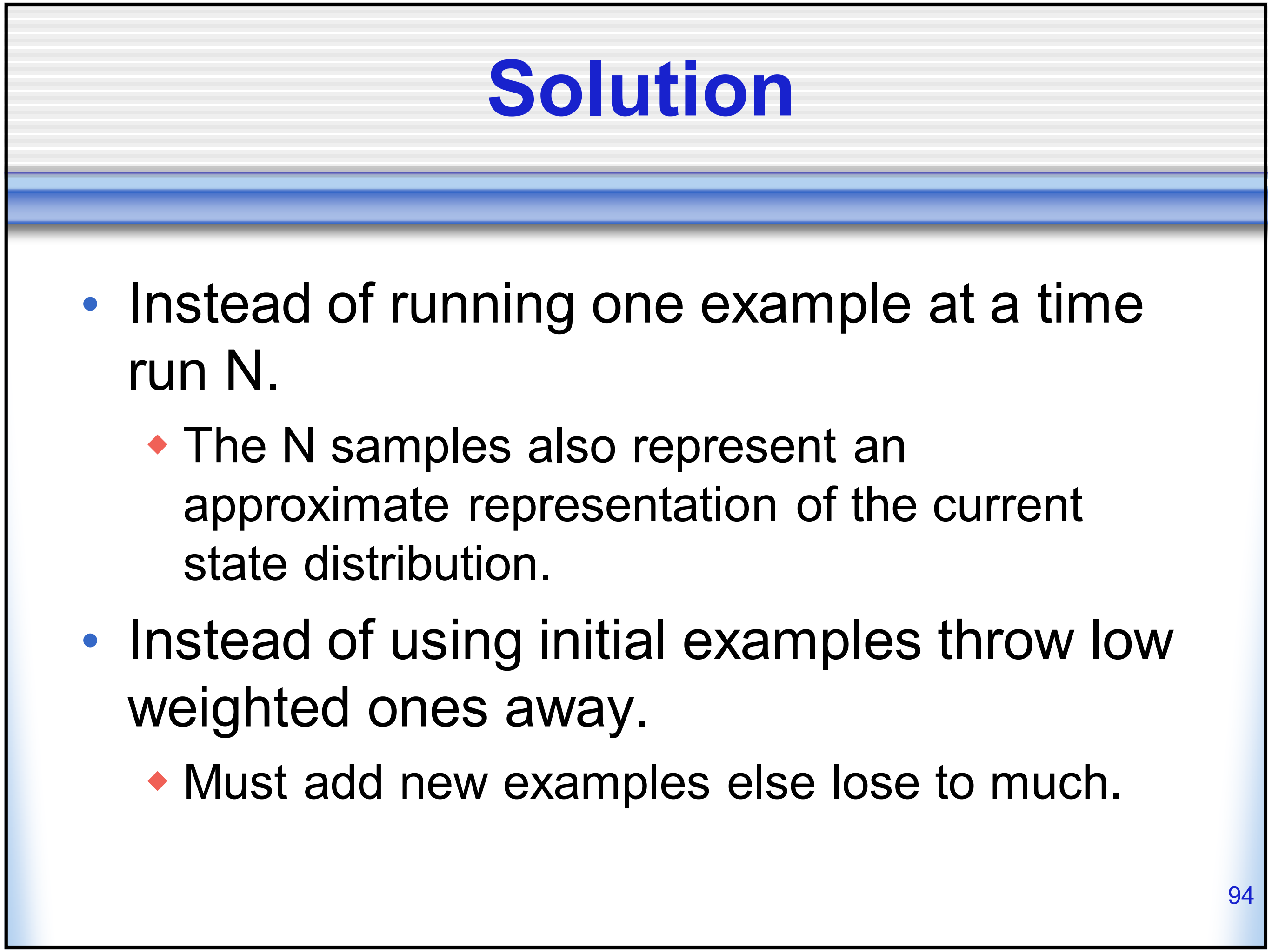
LW samples pay no attention to the evidence!

⇒ fraction “agreeing” falls exponentially with $t$

⇒ number of samples required grows exponentially with $t$

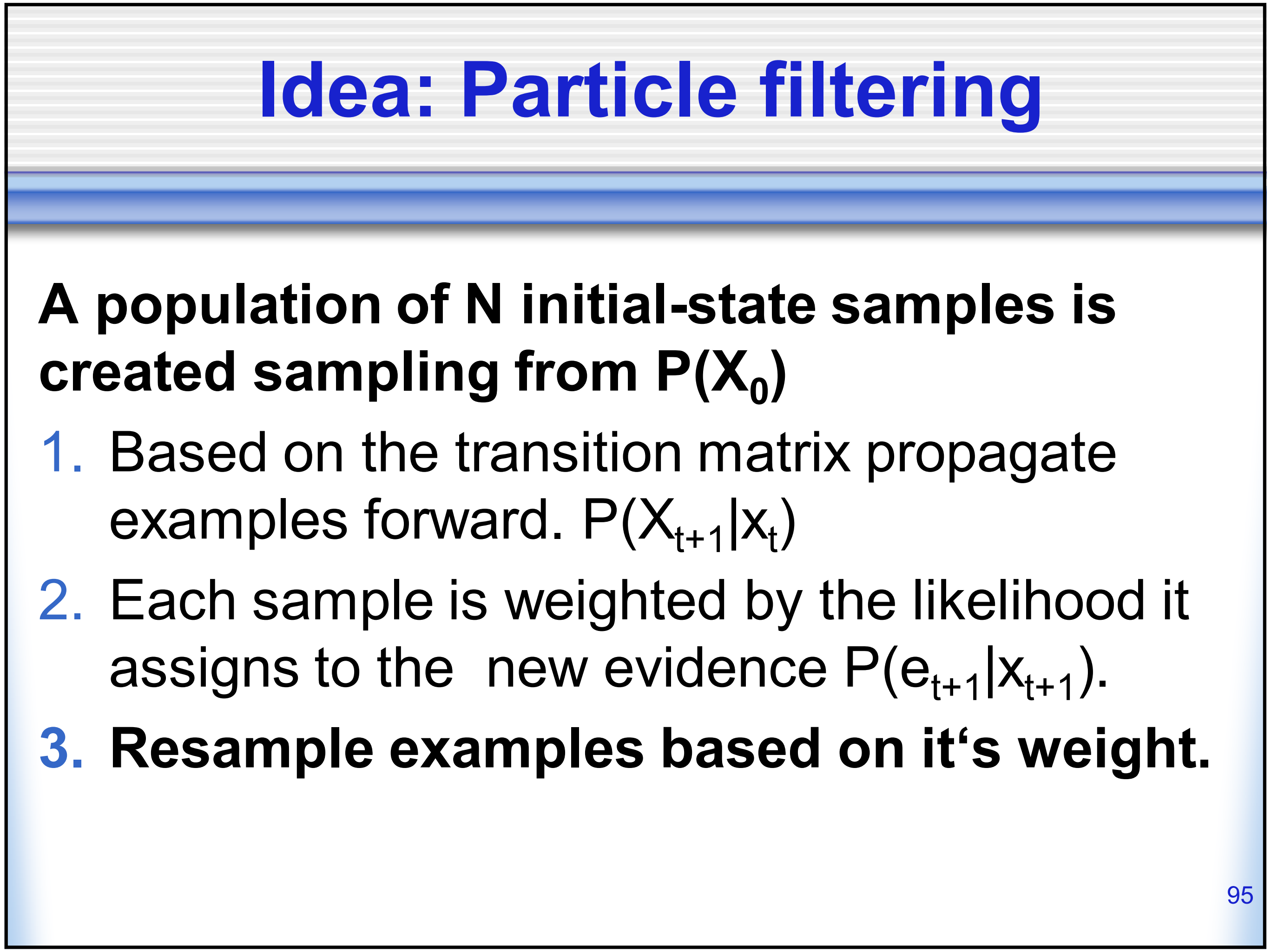
# Likelihood Weighting





# Solution

- Instead of running one example at a time run N.
  - The N samples also represent an approximate representation of the current state distribution.
- Instead of using initial examples throw low weighted ones away.
  - Must add new examples else lose to much.



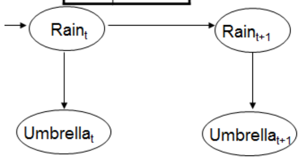
# Idea: Particle filtering

**A population of N initial-state samples is created sampling from $P(X_0)$**

1. Based on the transition matrix propagate examples forward. $P(X_{t+1}|x_t)$
2. Each sample is weighted by the likelihood it assigns to the  new evidence $P(e_{t+1}|x_{t+1})$.
3. **Resample examples based on it‘s weight.**

# Particle Filtering

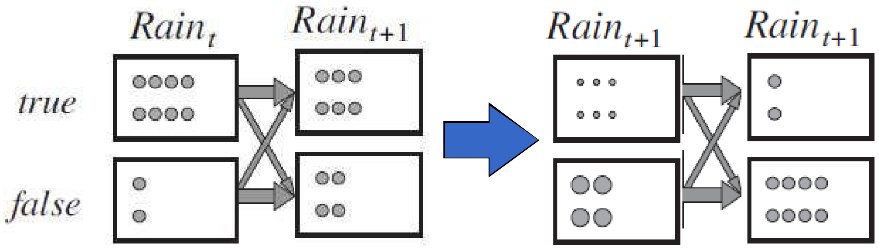
| $R_{t-1}$ | $P(R_t \mid R_{t-1})$ |
|---|---|
| T | 0.7 |
| F | 0.3 |



| $R_t$ | $P(U_t \mid R_t)$ |
|---|---|
| T | 0.9 |
| F | 0.2 |

Our current particles,10

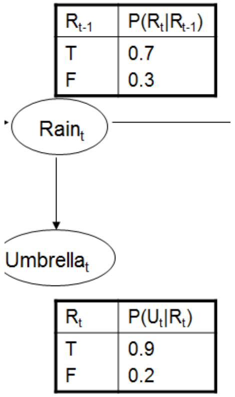
Propagate forward



No umbrella is observed at t+1

resampling

# Example



| $R_{t-1}$ | $P(R_t\|R_{t-1})$ |
|---|---|
| T | 0.7 |
| F | 0.3 |

| $R_t$ | $P(U_t\|R_t)$ |
|---|---|
| T | 0.9 |
| F | 0.2 |

$N(r_{t+1}|e) = \Sigma_{x_t} P(x_{t+1}|x_t) N(x_t|e)$

For rain = 0.7*8+0.3*2= 6.2 => 6

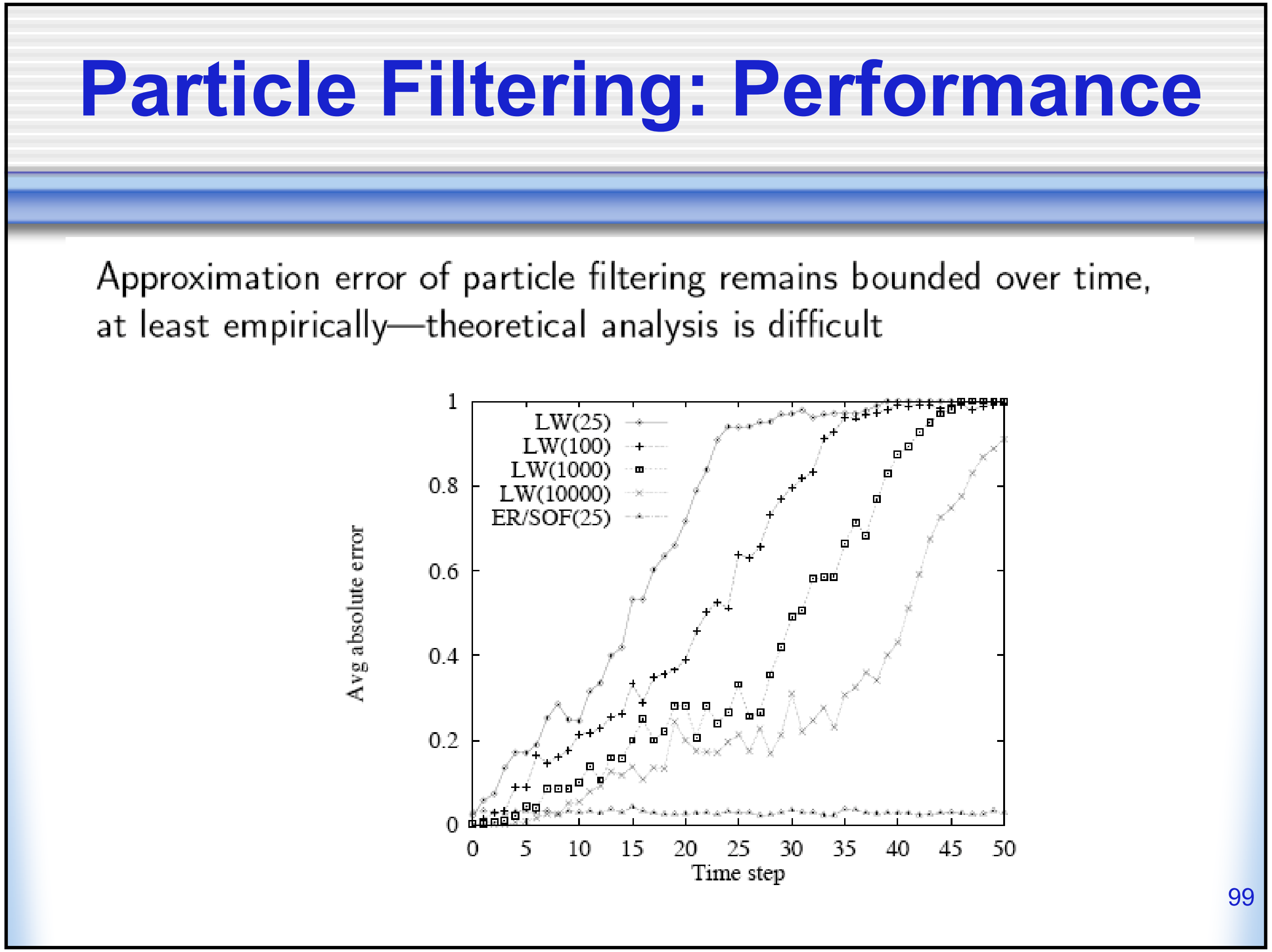
For not rain = 0.3 *8 + 0.7*2= 3.8 => 4

Suppose no umbrella for t+1

total weight(rain particles) = 0.1 * 6= 0.6

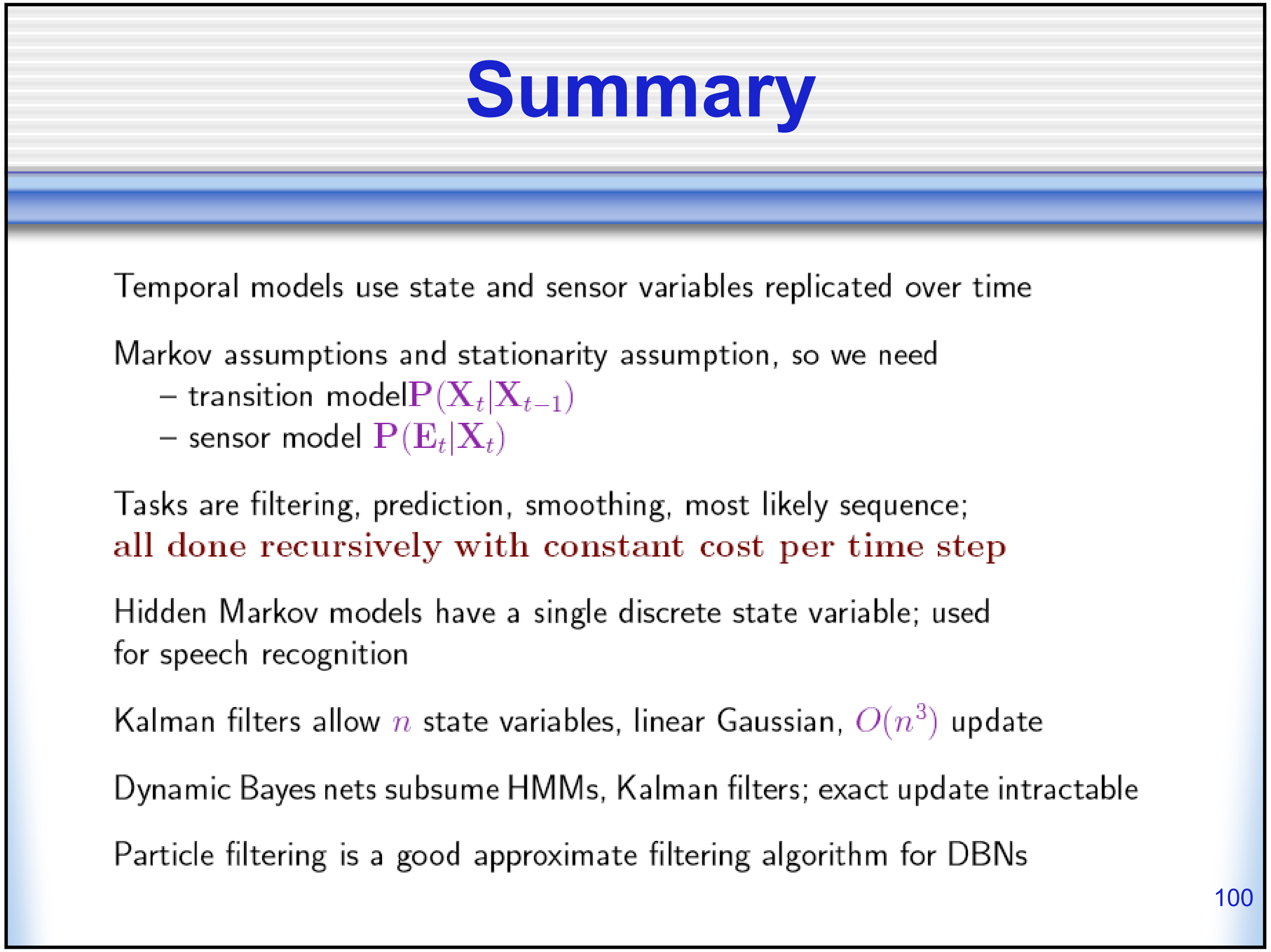
total weight(not rain) = 0.8 * 4= 3.2

Normalized =<0.17, 0.83>



# Particle Filtering: Performance

Approximation error of particle filtering remains bounded over time, at least empirically—theoretical analysis is difficult



# Summary

Temporal models use state and sensor variables replicated over time

Markov assumptions and stationarity assumption, so we need
- transition model$\mathbf{P}(\mathbf{X}_t|\mathbf{X}_{t-1})$
- sensor model $\mathbf{P}(\mathbf{E}_t|\mathbf{X}_t)$

Tasks are filtering, prediction, smoothing, most likely sequence;
**all done recursively with constant cost per time step**

Hidden Markov models have a single discrete state variable; used for speech recognition

Kalman filters allow $n$ state variables, linear Gaussian, $O(n^3)$ update

Dynamic Bayes nets subsume HMMs, Kalman filters; exact update intractable

Particle filtering is a good approximate filtering algorithm for DBNs

# Intelligent Autonomous Agents

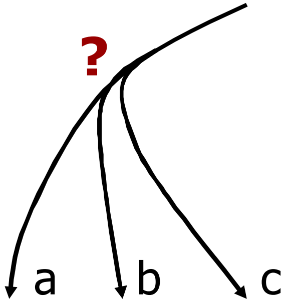
## and Cognitive Robotics

### Topic 7: Decision-Making under Uncertainty Simple Decisions

Ralf Möller, Rainer Marrone

Hamburg University of Technology

# Non-Deterministic vs. Probabilistic Uncertainty

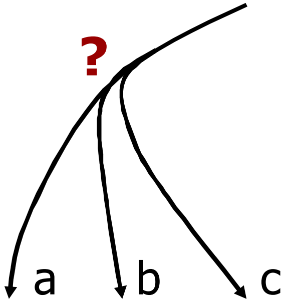


{a,b,c}

→ decision that is best for worst case

Non-deterministic model

~ Adversarial search



$\{a(p_a),b(p_b),c(p_c)\}$

→ decision that maximizes expected utility value

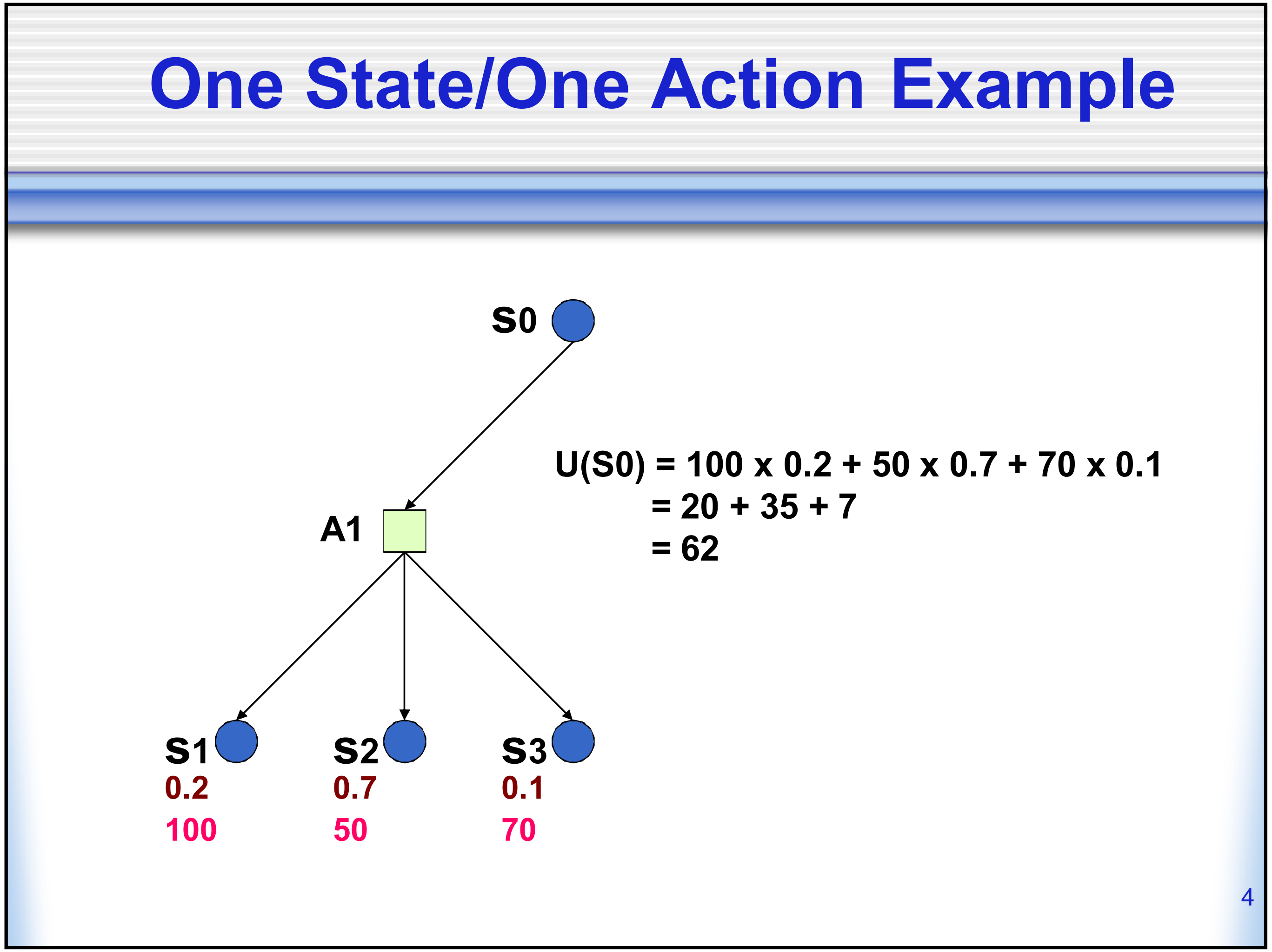
Probabilistic model

# Expected Utility

- Random variable *X* with *n* values $x_1,\ldots,x_n$ and distribution $(p_1,\ldots,p_n)$
  **X** is the state reached after doing an action **A** under uncertainty
- Function **U** of **X** : **U** is the utility of a state
- The expected utility of **A** is

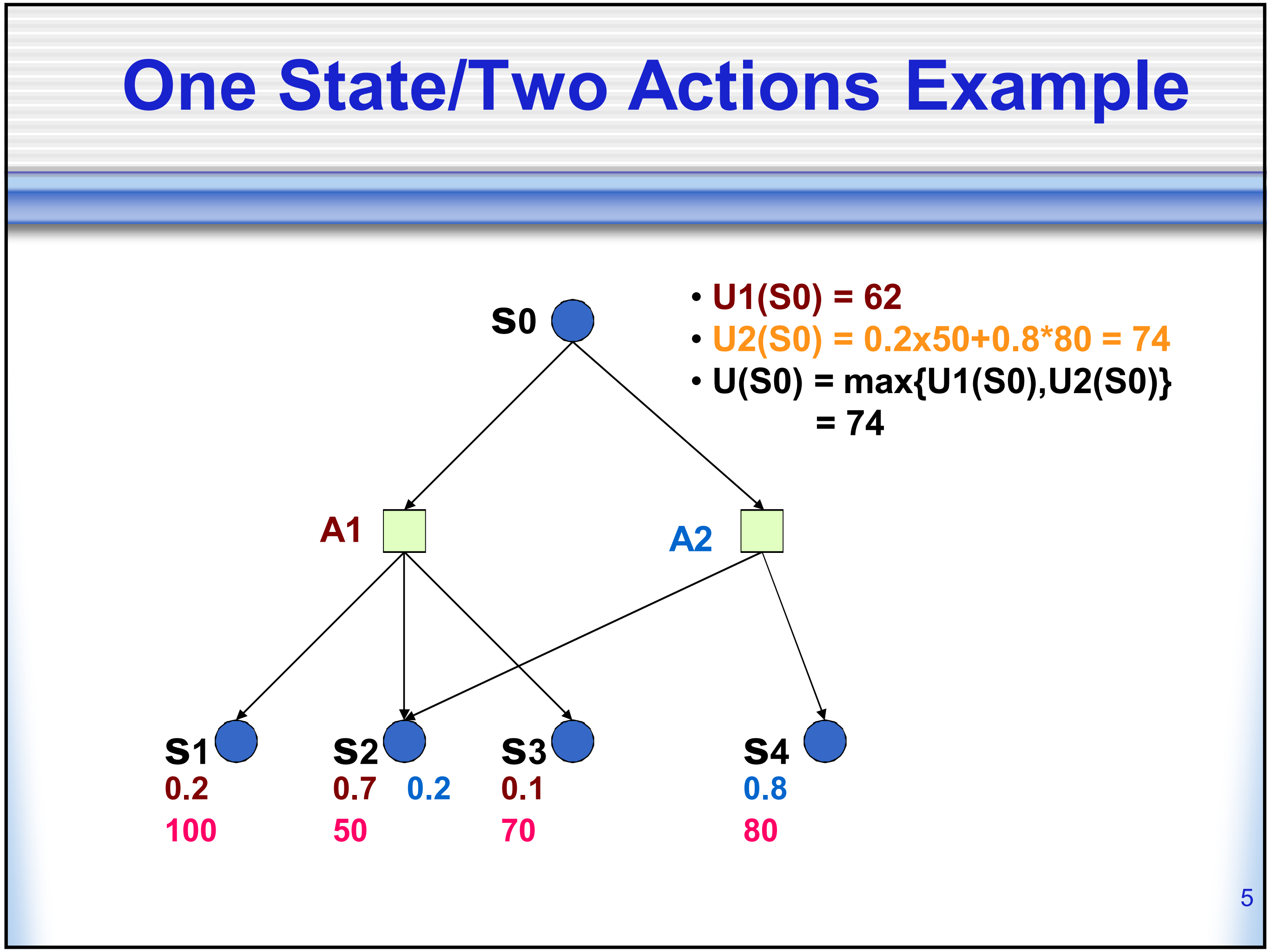
$$EU[A] = \Sigma_{i=1,\ldots,n}\, p(x_i|A)U(x_i)$$

$$MEU = \underset{A}{\mathrm{argmax}}\ EU[A]$$



# One State/One Action Example

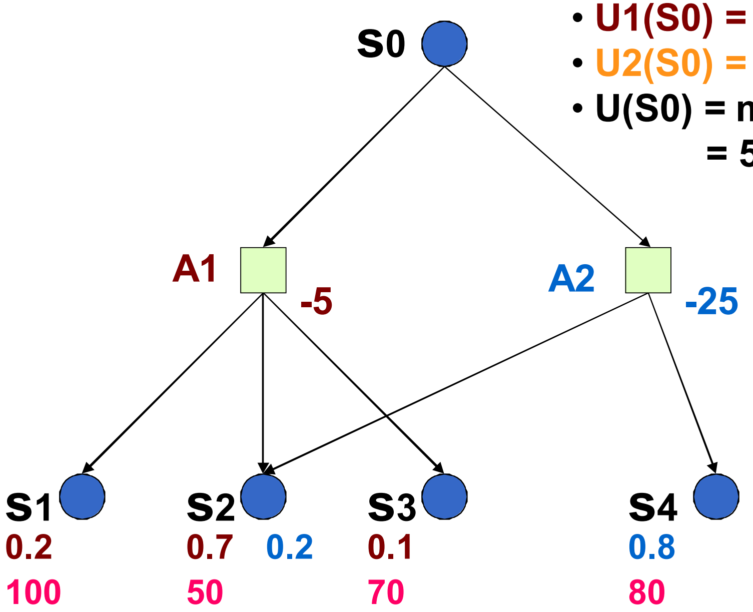
U(S0) = 100 x 0.2 + 50 x 0.7 + 70 x 0.1
= 20 + 35 + 7
= 62



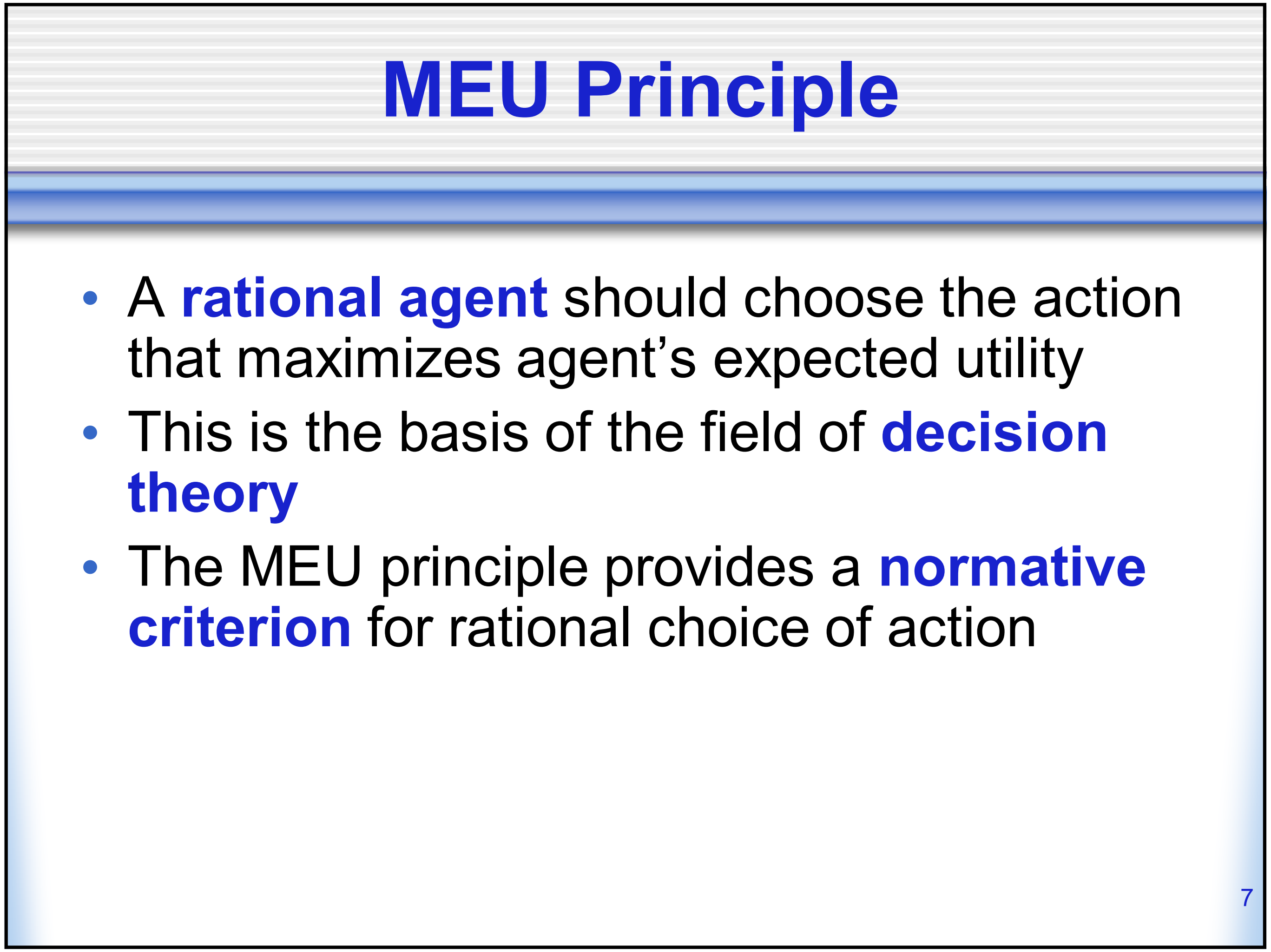
# One State/Two Actions Example

- U1(S0) = 62
- U2(S0) = 0.2x50+0.8*80 = 74
- U(S0) = max{U1(S0),U2(S0)}
  = 74

# Introducing Action Costs

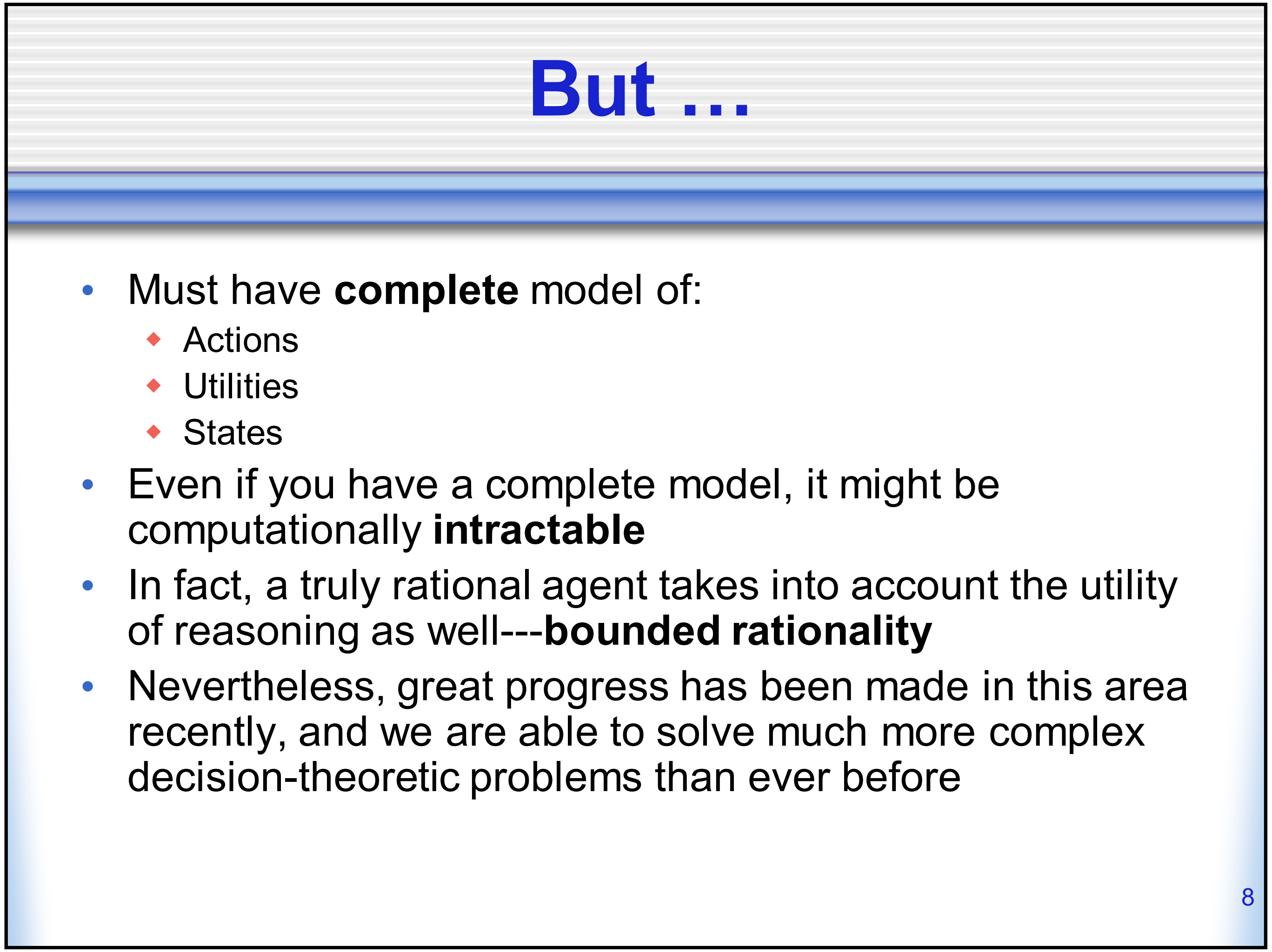


- U1(S0) = 62 – 5 = 57
- U2(S0) = 74 – 25 = 49
- U(S0) = max{U1(S0),U2(S0)}
  = 57



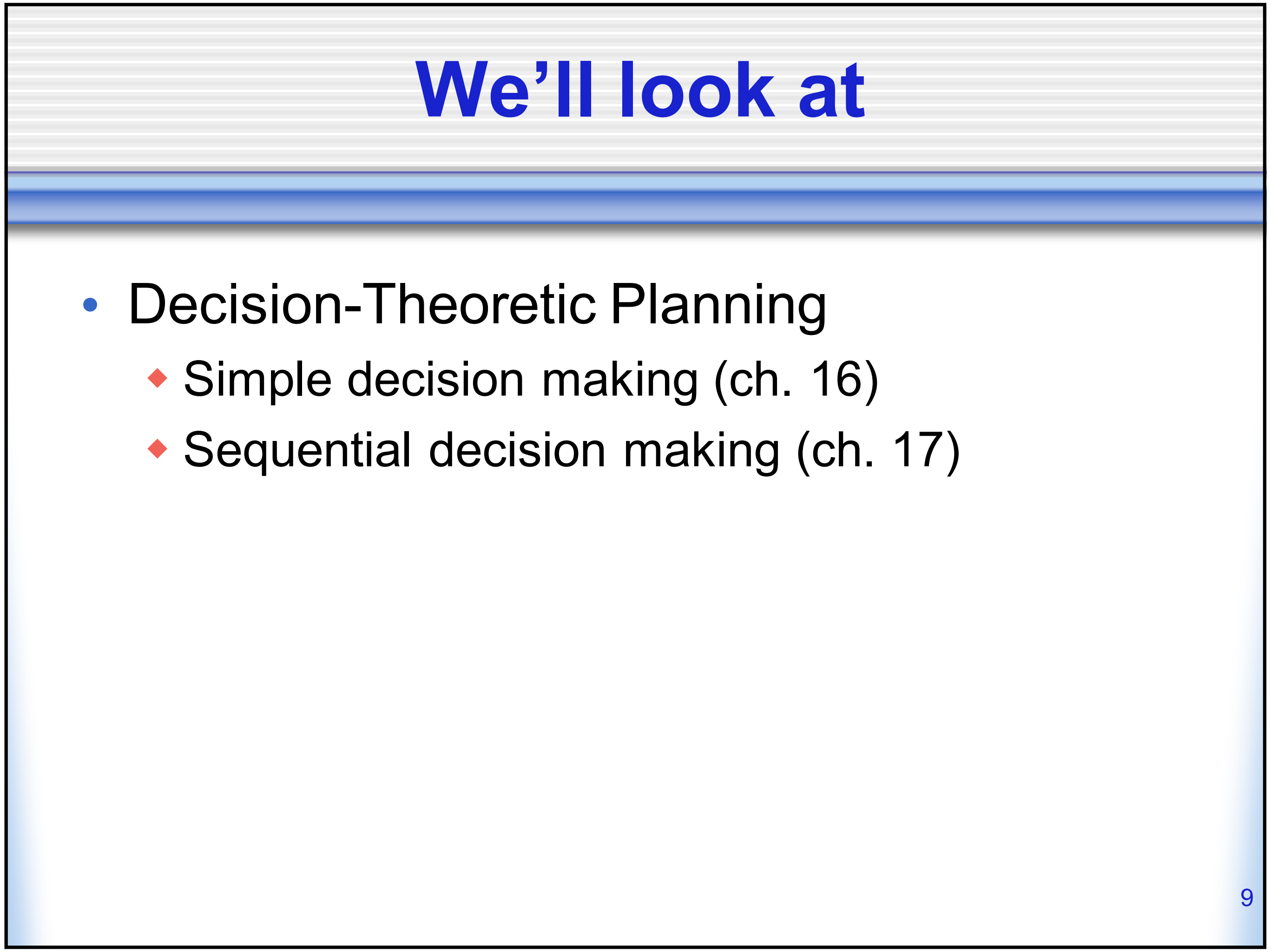
# MEU Principle

- A **rational agent** should choose the action that maximizes agent’s expected utility
- This is the basis of the field of **decision theory**
- The MEU principle provides a **normative criterion** for rational choice of action



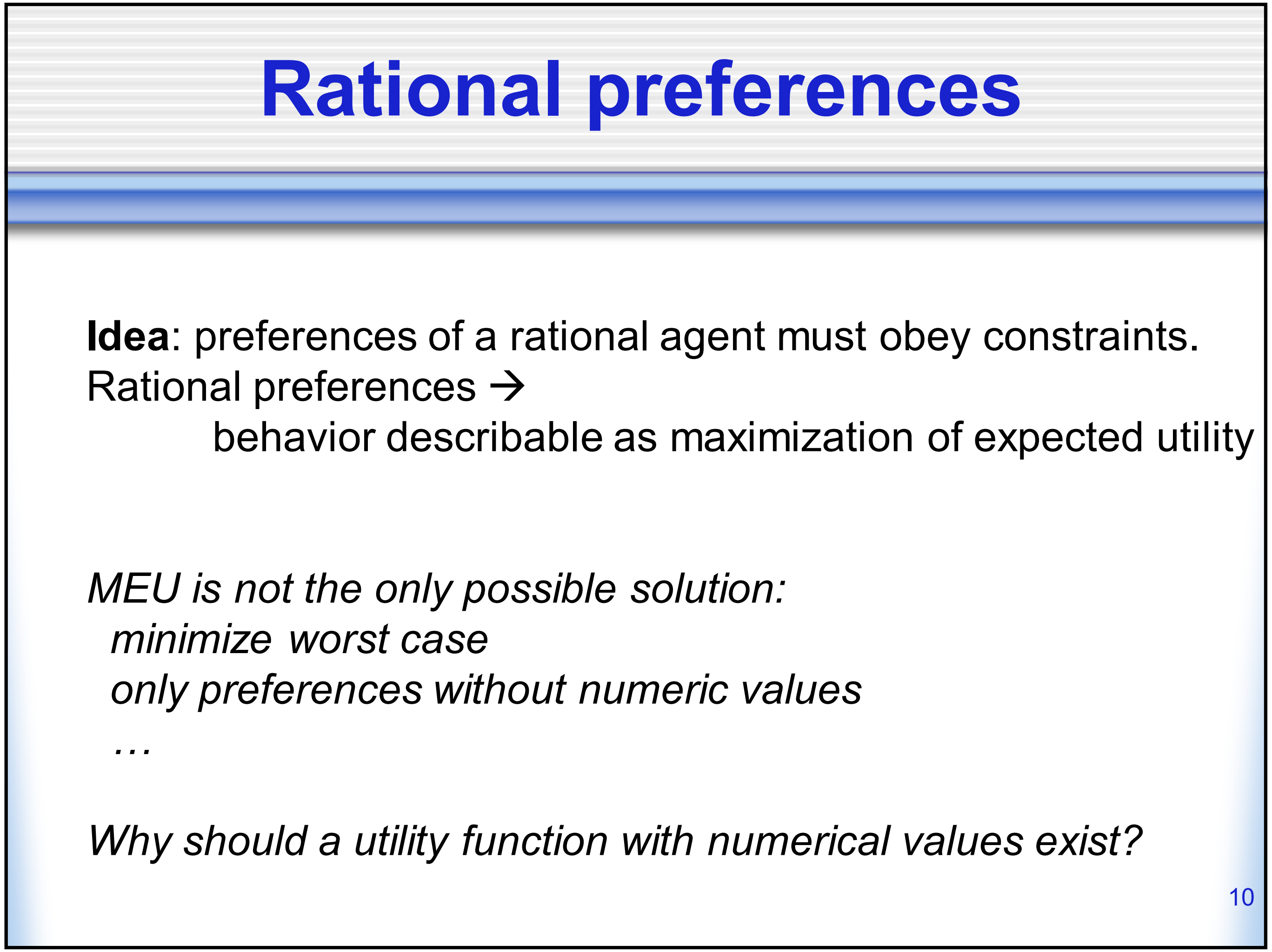
# But …

- Must have **complete** model of:
  - Actions
  - Utilities
  - States
- Even if you have a complete model, it might be computationally **intractable**
- In fact, a truly rational agent takes into account the utility of reasoning as well---**bounded rationality**
- Nevertheless, great progress has been made in this area recently, and we are able to solve much more complex decision-theoretic problems than ever before



# We’ll look at

- Decision-Theoretic Planning
  - Simple decision making (ch. 16)
  - Sequential decision making (ch. 17)



# Rational preferences

**Idea**: preferences of a rational agent must obey constraints.
Rational preferences →
behavior describable as maximization of expected utility

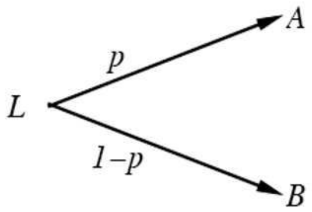
*MEU is not the only possible solution:*
*minimize worst case*
*only preferences without numeric values*
*…*

*Why should a utility function with numerical values exist?*

# Basis of utility theory: constrains on preferences

- An agent chooses among prizes (A,B,…) and lotteries, i.e., situations with uncertain prizes.

Lottery L =[p, A ; (1-p), B]



$A \succ B$ the agent prefers $A$ over $B$.

$A \sim B$ the agent is indifferent between $A$ and $B$.

A and B can be lotteries again: Prizes are special lotteries: [1, X; 0, not X]

# Axioms of Utility Theory

- **Orderability:** Given any two states, the rational agent prefers one of them, else the two as equally preferable.

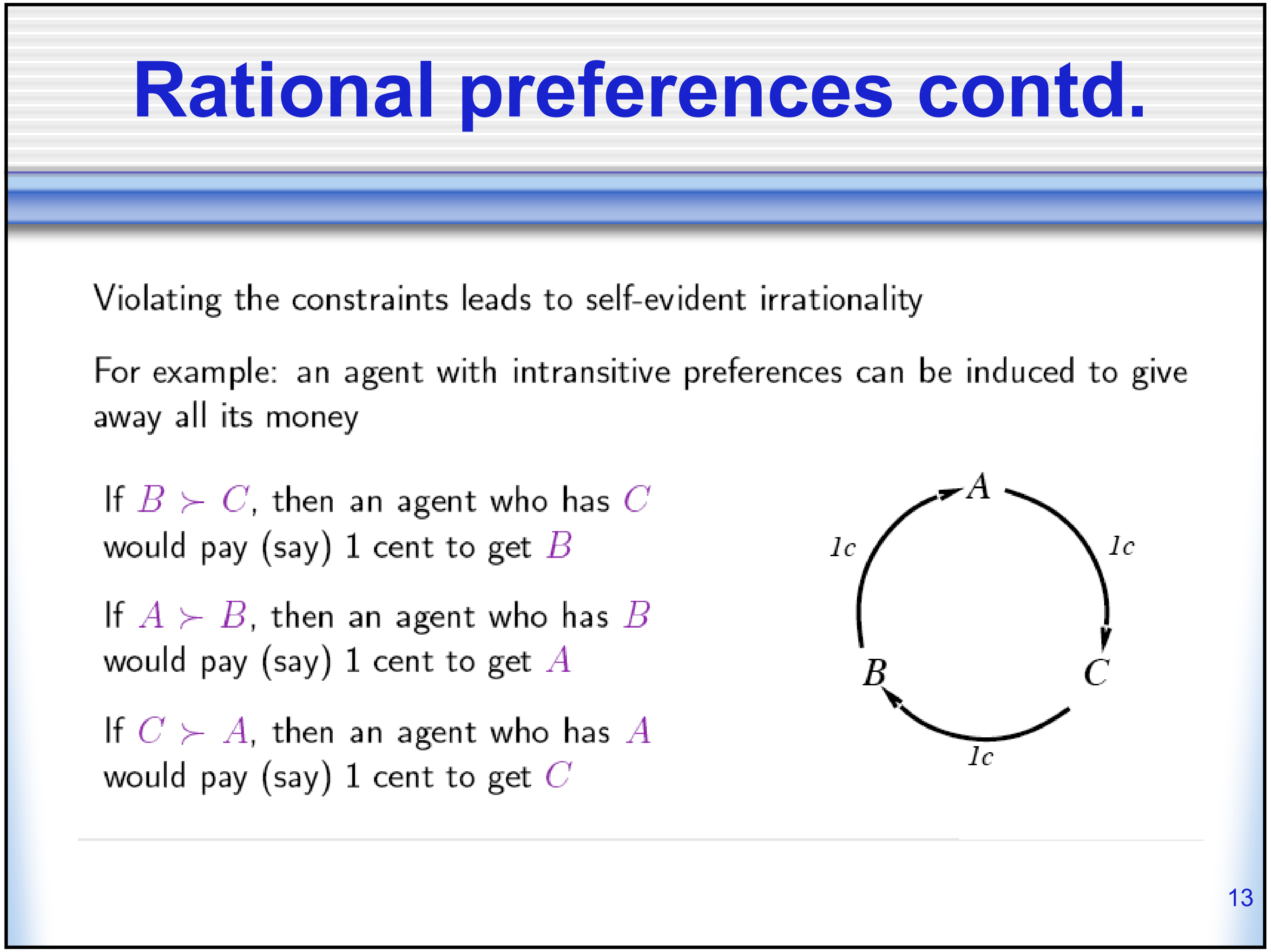
$$(A \succ B) \vee (B \succ A) \vee (A \sim B)$$

- **Transitivity:** Given any three states, if an agent prefers $A$ to $B$ and prefers $B$ to C, agent must prefer $A$ to C.

$$(A \succ B) \wedge (B \succ C) \Rightarrow (A \succ C)$$

- **Continuity:** If some state $B$ is between $A$ and C in preference, then there is a $p$ for which the rational agent will be indifferent between state B and the lottery in which A comes with probability p, C with probability (1-p).

$$(A \succ B \succ C) \Rightarrow \exists p\,[\,p : A; (1-p) : C\,] \sim B$$



# Rational preferences contd.

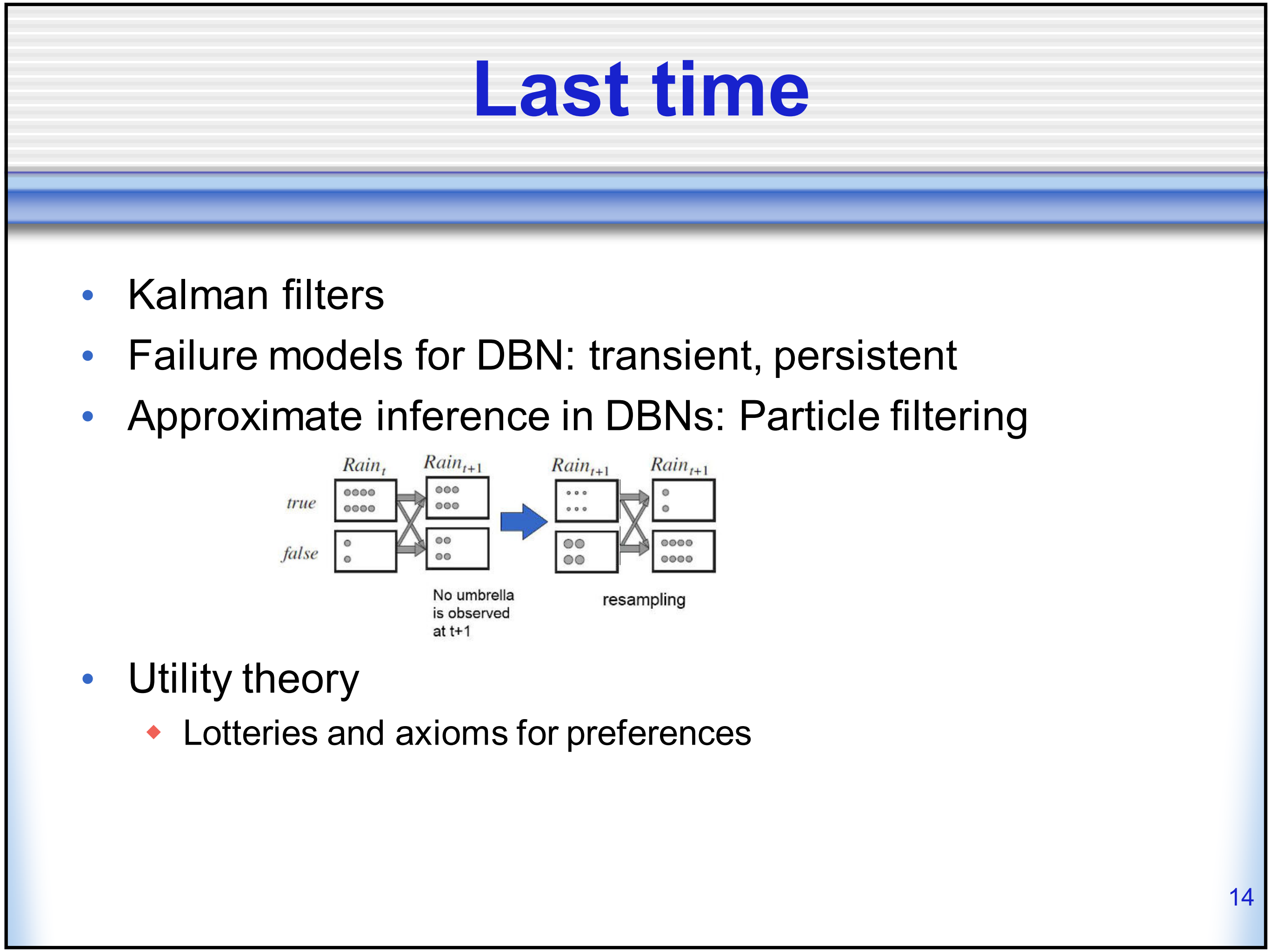
Violating the constraints leads to self-evident irrationality

For example: an agent with intransitive preferences can be induced to give away all its money

If $B \succ C$, then an agent who has $C$ would pay (say) 1 cent to get $B$

If $A \succ B$, then an agent who has $B$ would pay (say) 1 cent to get $A$

If $C \succ A$, then an agent who has $A$ would pay (say) 1 cent to get $C$

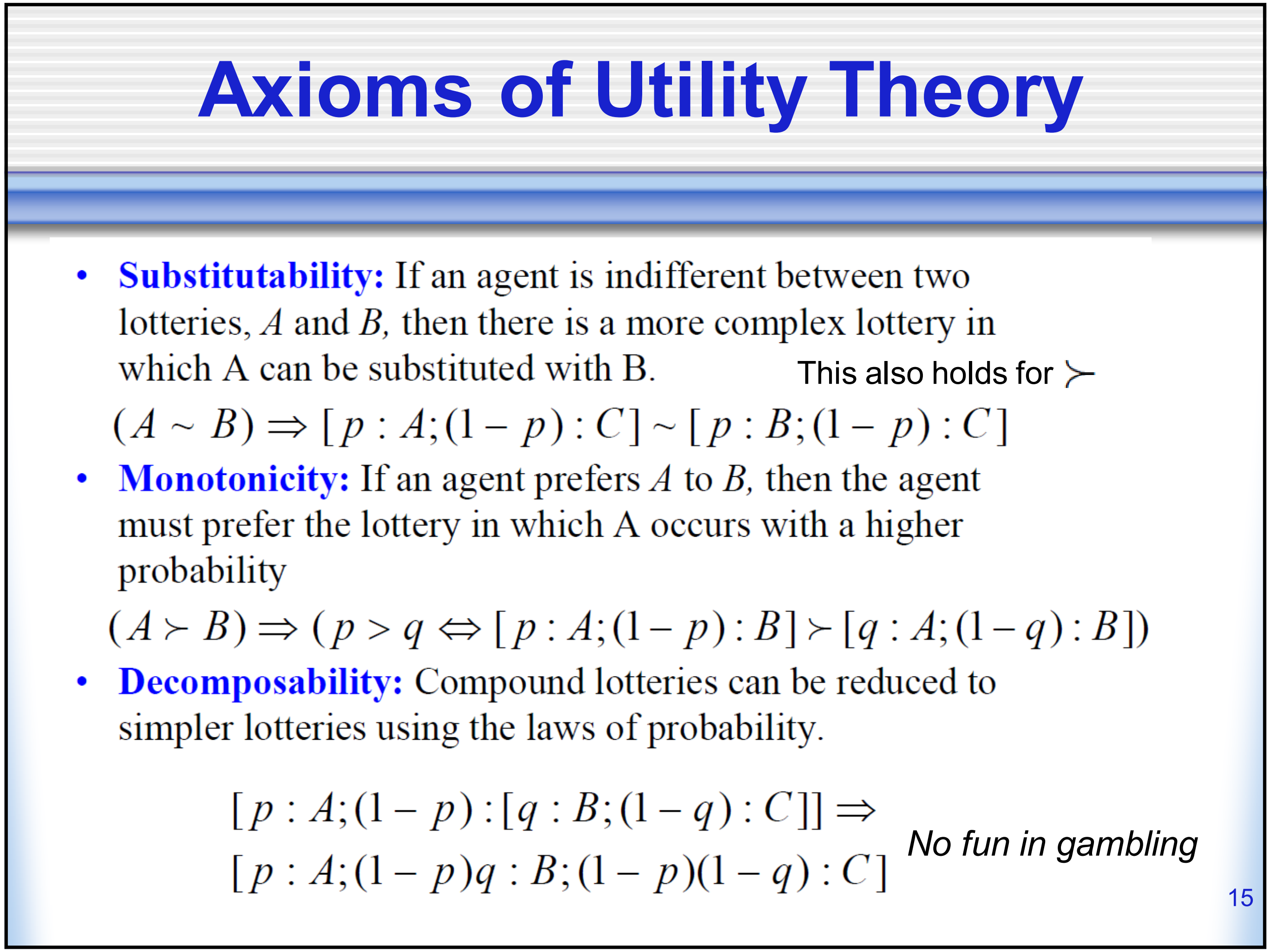


# Last time

- Kalman filters
- Failure models for DBN: transient, persistent
- Approximate inference in DBNs: Particle filtering

$Rain_t$
$Rain_{t+1}$
$Rain_{t+1}$
$Rain_{t+1}$
*true*
*false*
No umbrella is observed at t+1
resampling

- Utility theory
  - Lotteries and axioms for preferences



# Axioms of Utility Theory

- **Substitutability:** If an agent is indifferent between two lotteries, $A$ and $B$, then there is a more complex lottery in which A can be substituted with B.

This also holds for $\succ$

$$(A \sim B) \Rightarrow [p : A; (1-p) : C] \sim [p : B; (1-p) : C]$$

- **Monotonicity:** If an agent prefers $A$ to $B$, then the agent must prefer the lottery in which A occurs with a higher probability

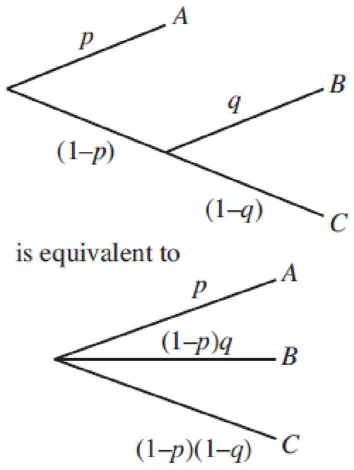
$$(A \succ B) \Rightarrow (p > q \Leftrightarrow [p : A; (1-p) : B] \succ [q : A; (1-q) : B])$$

- **Decomposability:** Compound lotteries can be reduced to simpler lotteries using the laws of probability.

$$[p : A; (1-p) : [q : B; (1-q) : C]] \Rightarrow$$
$$[p : A; (1-p)q : B; (1-p)(1-q) : C]$$

*No fun in gambling*

# Decomposabilty

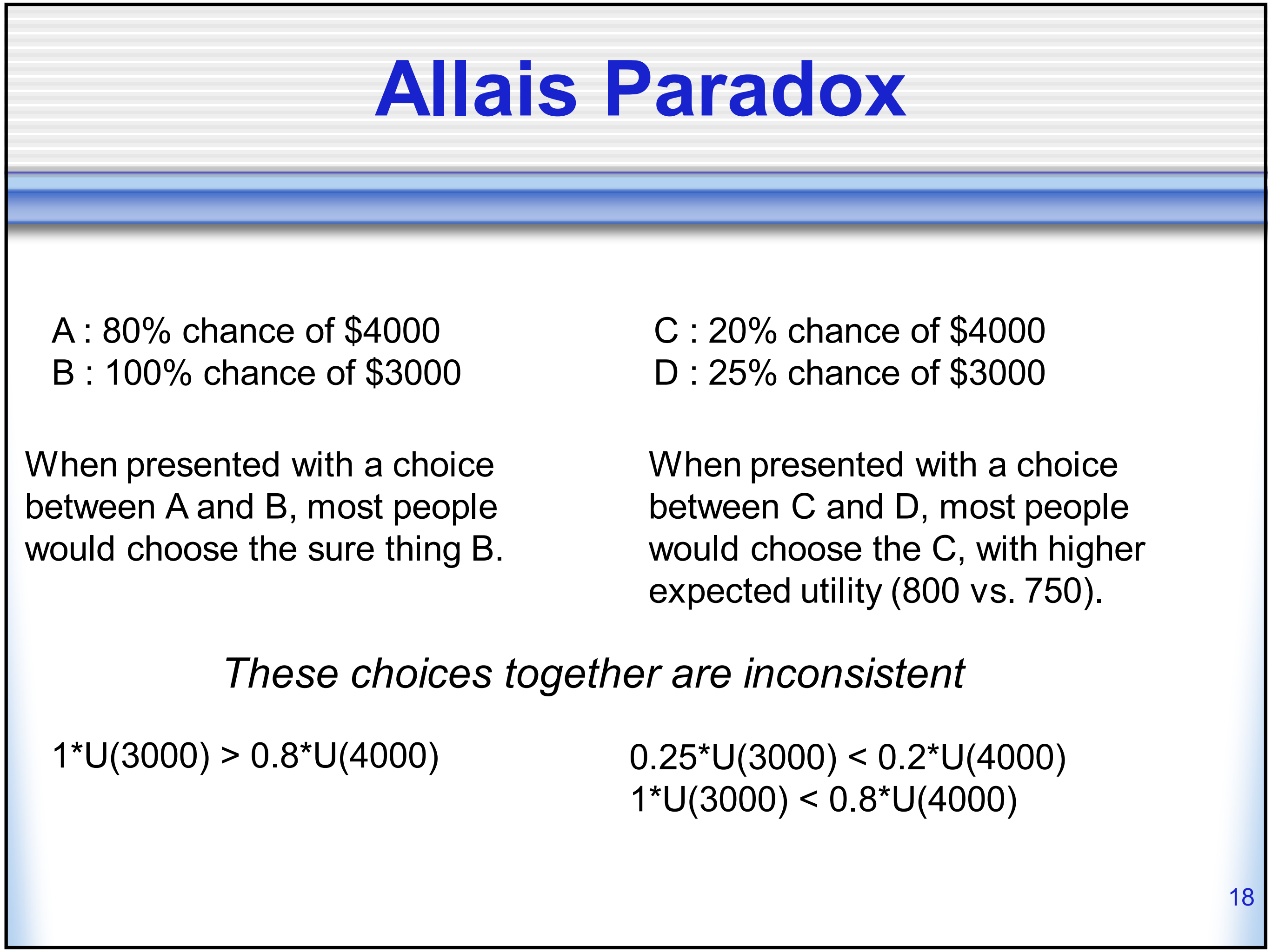


# And then there was utility

- Theorem by Neumann and Morgenstern, 1944
  Given preferences satisfying the constraints there exists a real-valued function *U* such that

$$U(A) \geq U(B) \quad \Leftrightarrow \quad A \succsim B$$
$$U([p_1, S_1; \ \ldots \ ; \ p_n, S_n]) = \Sigma_i \ p_i U(S_i)$$

**MEU** principle:
Choose the action that maximizes expected utility



# Allais Paradox

A : 80% chance of $4000
B : 100% chance of $3000

C : 20% chance of $4000
D : 25% chance of $3000

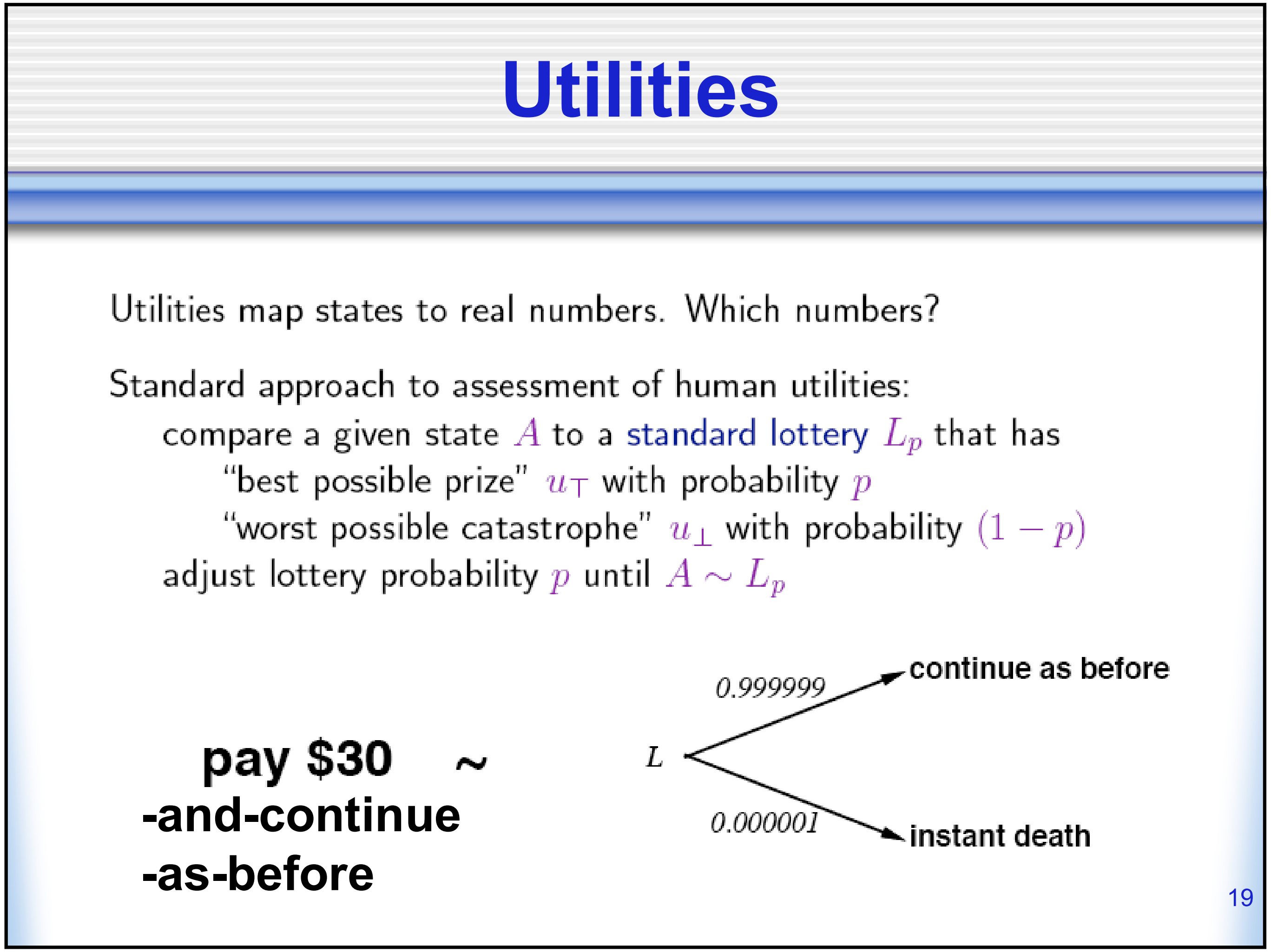
When presented with a choice between A and B, most people would choose the sure thing B.

When presented with a choice between C and D, most people would choose the C, with higher expected utility (800 vs. 750).

*These choices together are inconsistent*

1*U(3000) > 0.8*U(4000)

0.25*U(3000) < 0.2*U(4000)
1*U(3000) < 0.8*U(4000)

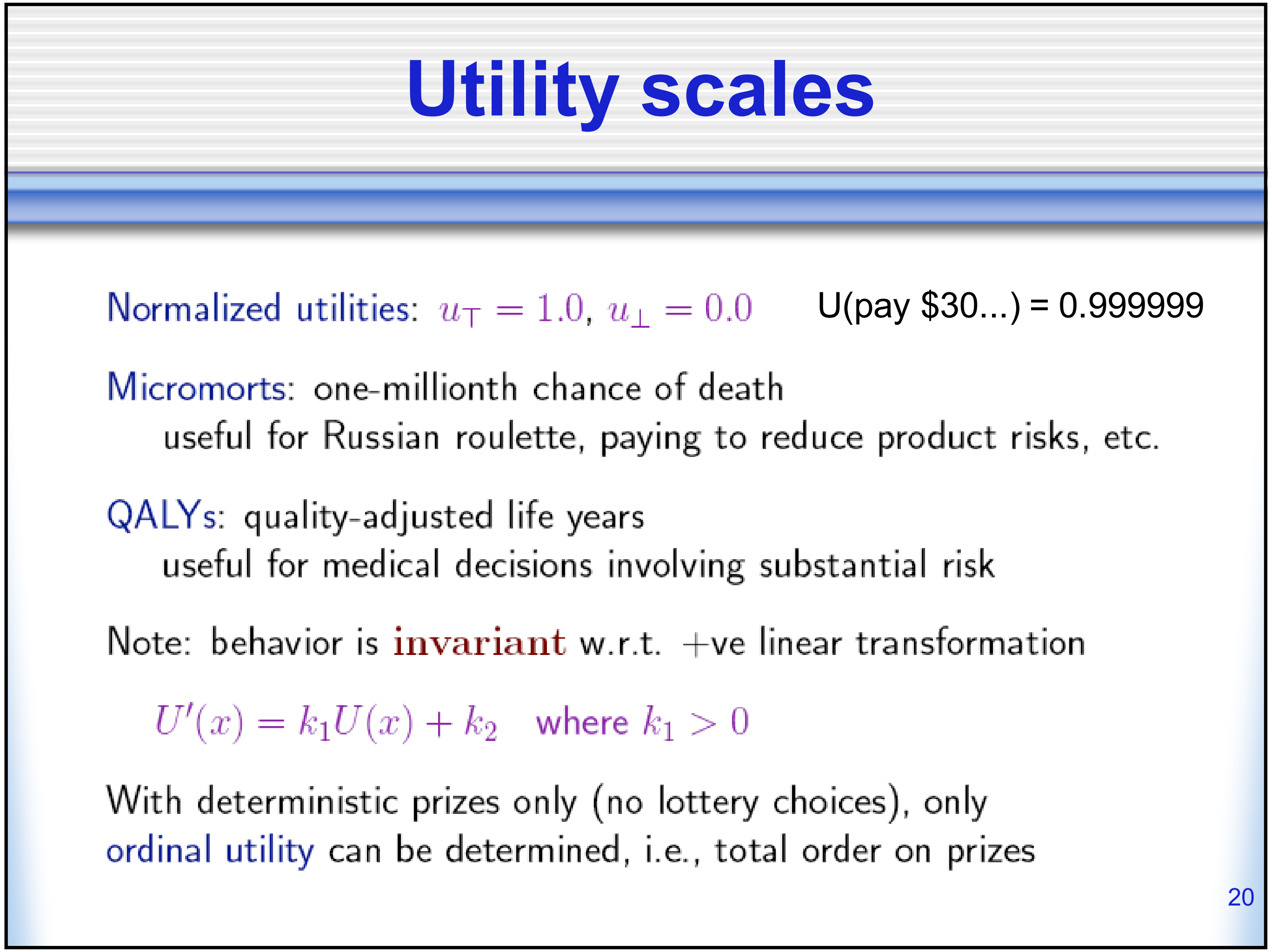


# Utilities

Utilities map states to real numbers. Which numbers?

Standard approach to assessment of human utilities:
compare a given state $A$ to a standard lottery $L_p$ that has
"best possible prize" $u_\top$ with probability $p$
"worst possible catastrophe" $u_\perp$ with probability $(1-p)$
adjust lottery probability $p$ until $A \sim L_p$

**pay $30 ~**
**-and-continue**
**-as-before**



# Utility scales

Normalized utilities: $u_\top = 1.0$, $u_\bot = 0.0$ U(pay $30...) = 0.999999

Micromorts: one-millionth chance of death
useful for Russian roulette, paying to reduce product risks, etc.

QALYs: quality-adjusted life years
useful for medical decisions involving substantial risk

Note: behavior is **invariant** w.r.t. +ve linear transformation

$$U'(x) = k_1 U(x) + k_2 \quad \text{where } k_1 > 0$$

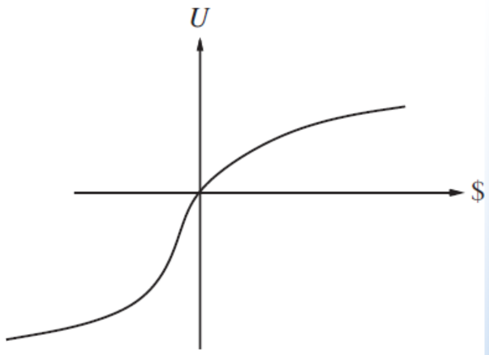
With deterministic prizes only (no lottery choices), only ordinal utility can be determined, i.e., total order on prizes

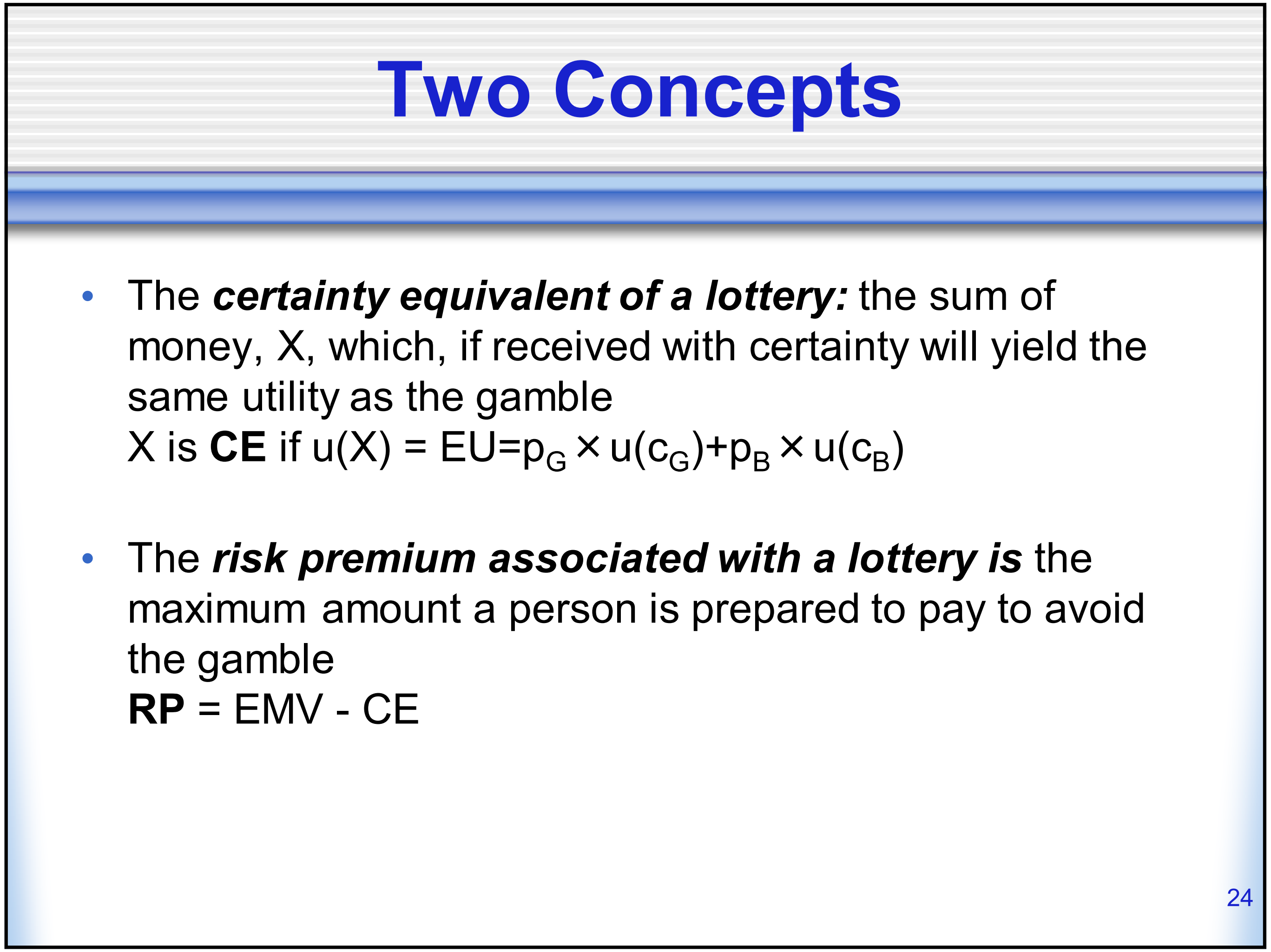
# Value Functions

- Provides a ranking of alternatives, but not a meaningful metric scale
- Also known as an “ordinal utility function”
- Remember the expectiminimax example:
  - Sometimes, only relative judgments (value functions) are necessary
  - At other times, absolute judgments (utility functions) are required

# Money Versus Utility

- Money <> Utility
  - More money is better, but not always in a linear relationship to the amount of money
- Expected Monetary Value
- Risk-averse – U(L) < U($S_{EMV(L)}$)
- Risk-seeking – U(L) > U($S_{EMV(L)}$)
- Risk-neutral – U(L) = U($S_{EMV(L)}$)

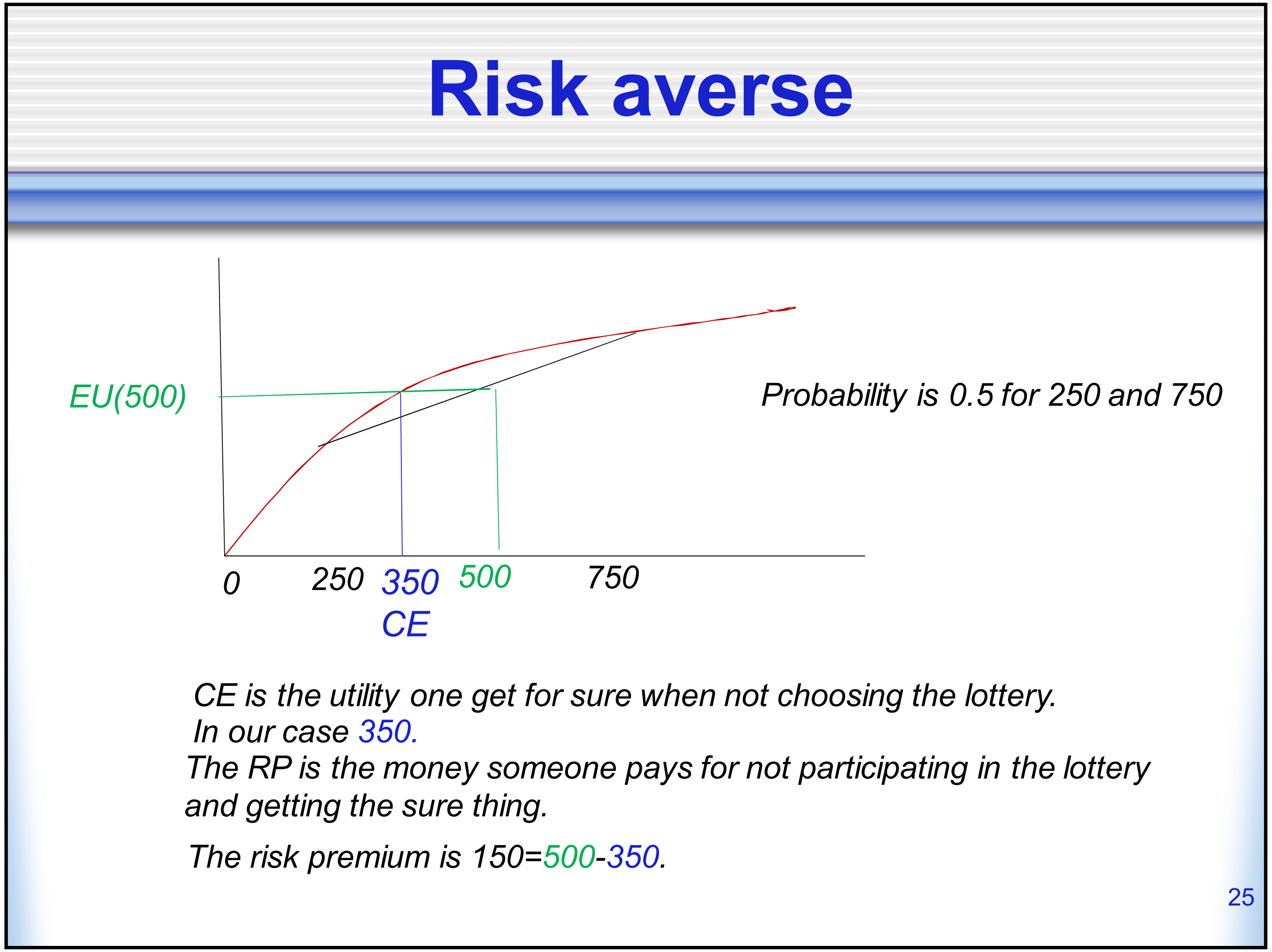




# Two Concepts

- The ***certainty equivalent of a lottery:*** the sum of money, X, which, if received with certainty will yield the same utility as the gamble
  X is **CE** if $u(X) = EU = p_G \times u(c_G) + p_B \times u(c_B)$

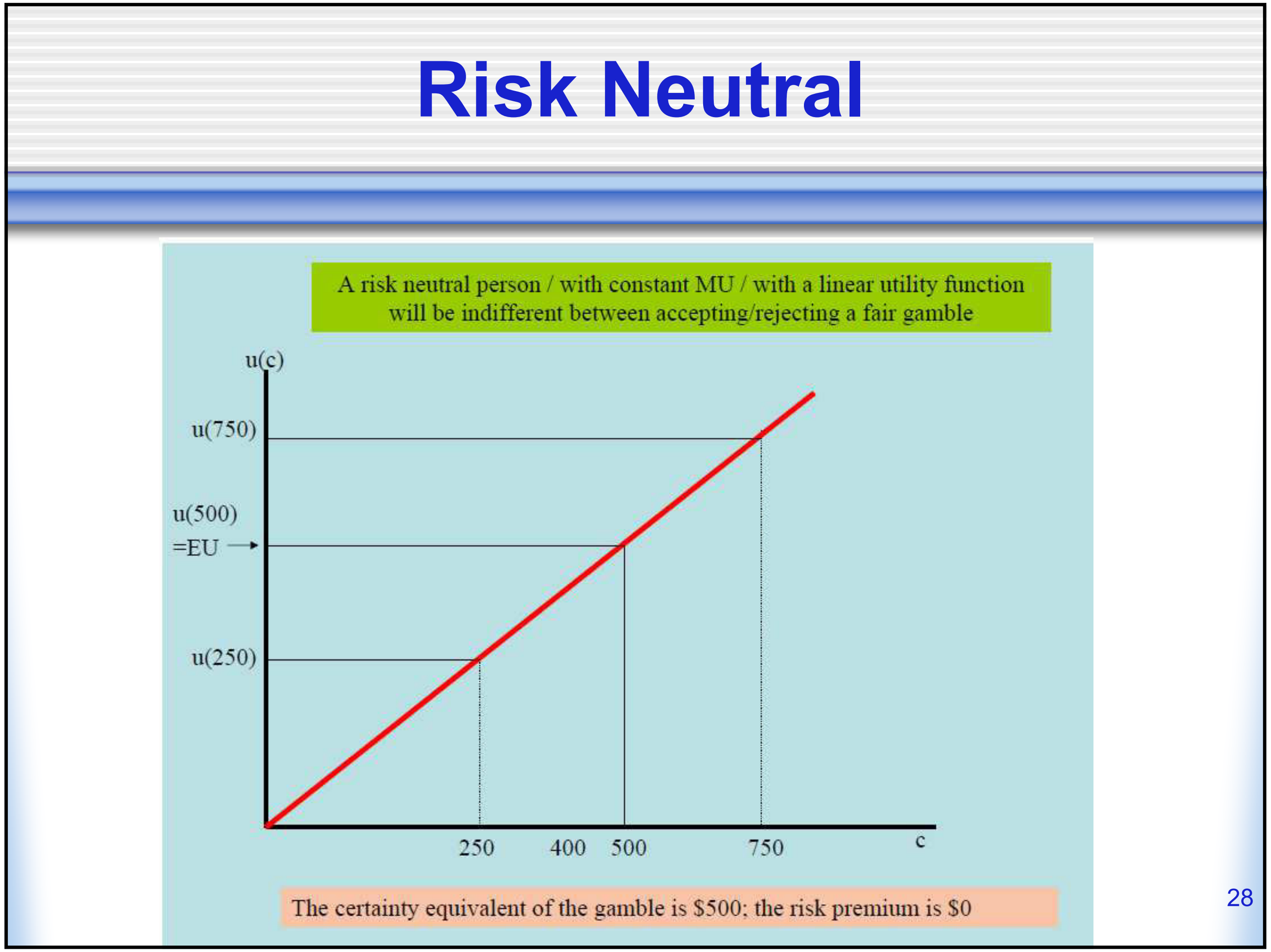
- The ***risk premium associated with a lottery is*** the maximum amount a person is prepared to pay to avoid the gamble
  **RP** = EMV - CE



# Risk averse

CE is the utility one get for sure when not choosing the lottery.
In our case 350.
The RP is the money someone pays for not participating in the lottery and getting the sure thing.

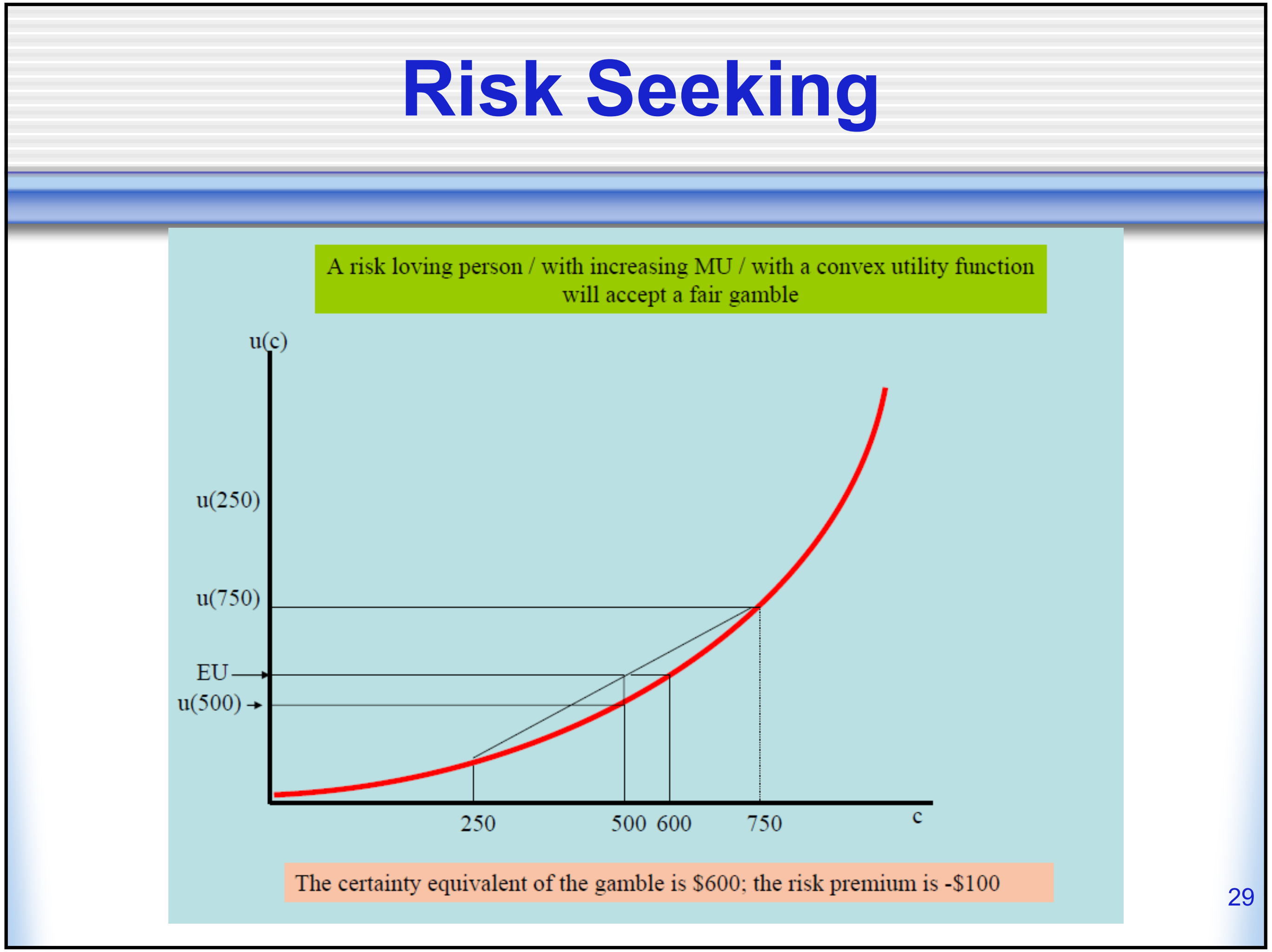
The risk premium is 150=500-350.



# Risk Neutral

A risk neutral person / with constant MU / with a linear utility function will be indifferent between accepting/rejecting a fair gamble

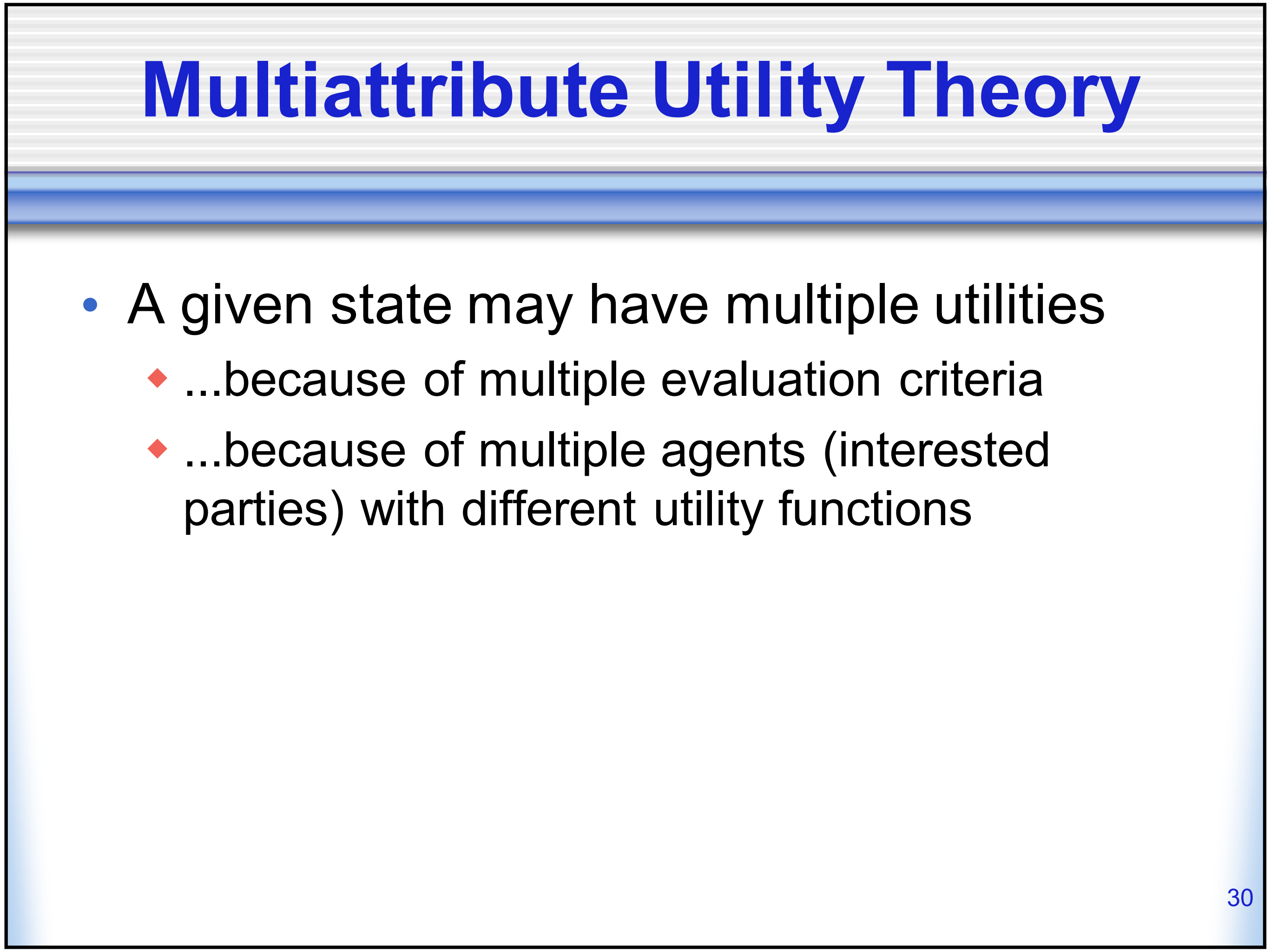
The certainty equivalent of the gamble is $500; the risk premium is $0



# Risk Seeking

A risk loving person / with increasing MU / with a convex utility function will accept a fair gamble

The certainty equivalent of the gamble is $600; the risk premium is -$100



# Multiattribute Utility Theory

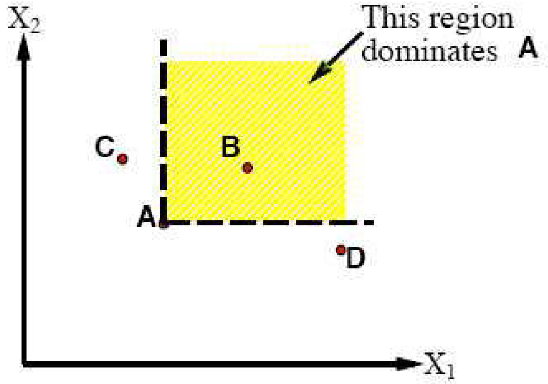
- A given state may have multiple utilities
  - ...because of multiple evaluation criteria
  - ...because of multiple agents (interested parties) with different utility functions

# Strict dominance

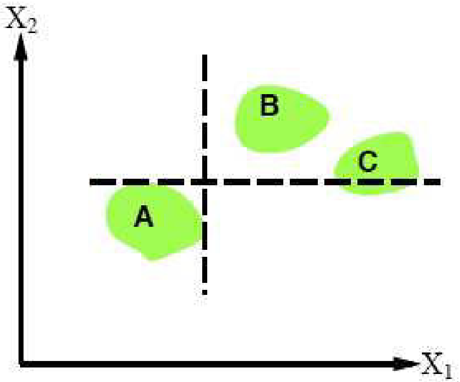
Typically define attributes such that $U$ is monotonic in each

Strict dominance: choice $B$ strictly dominates choice $A$ iff

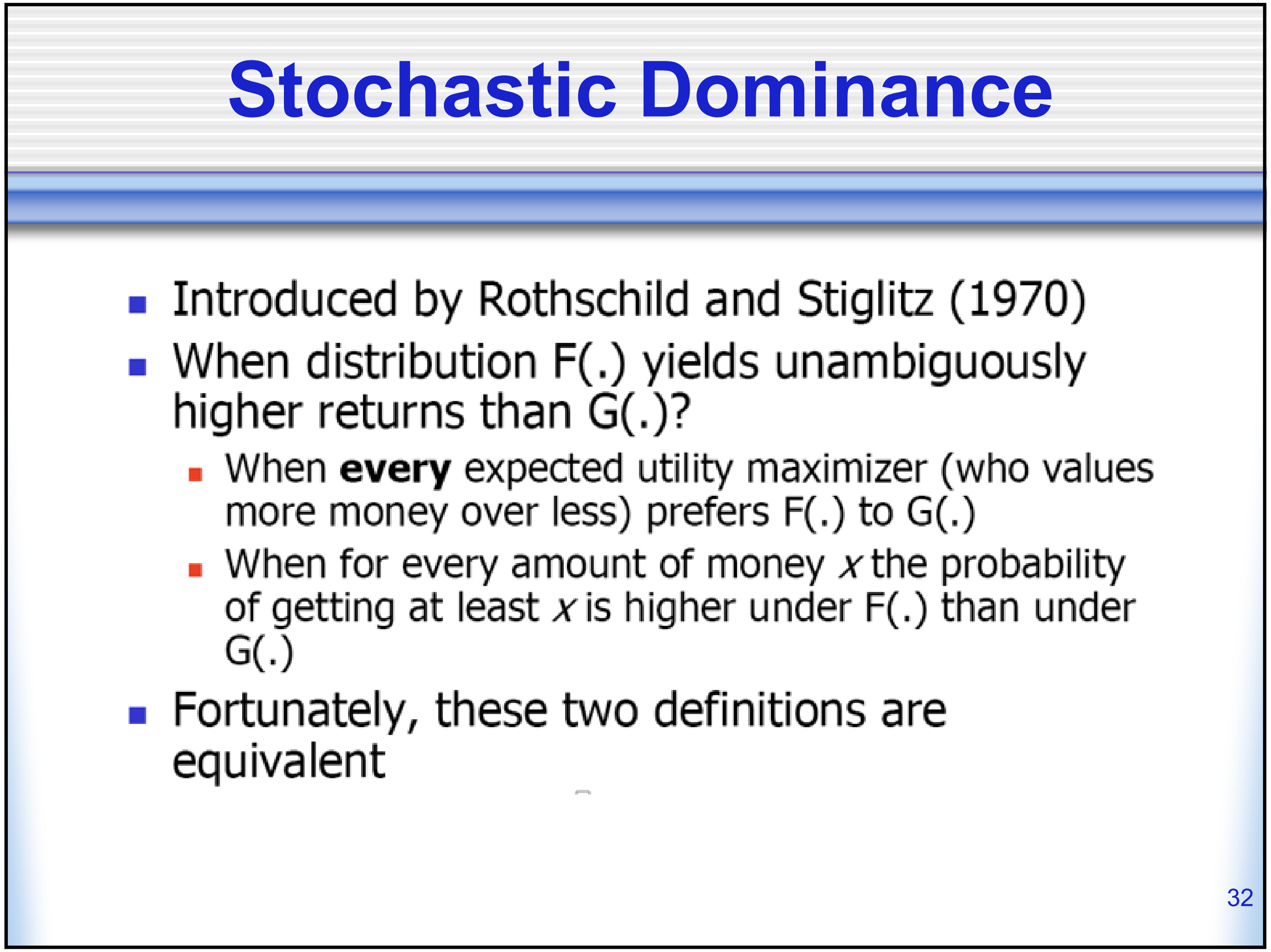
$$\forall i \;\; X_i(B) \geq X_i(A) \qquad \text{(and hence } U(B) \geq U(A)\text{)}$$



Deterministic attributes



Uncertain attributes



# Stochastic Dominance

- Introduced by Rothschild and Stiglitz (1970)
- When distribution F(.) yields unambiguously higher returns than G(.)?
  - When **every** expected utility maximizer (who values more money over less) prefers F(.) to G(.)
  - When for every amount of money *x* the probability of getting at least *x* is higher under F(.) than under G(.)
- Fortunately, these two definitions are equivalent

# Stochastic dominance

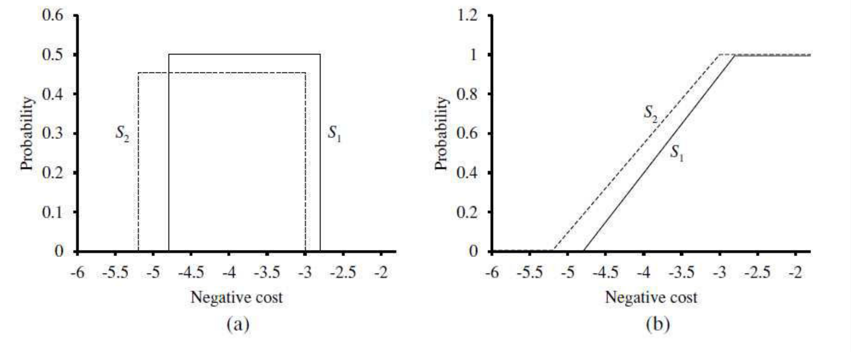
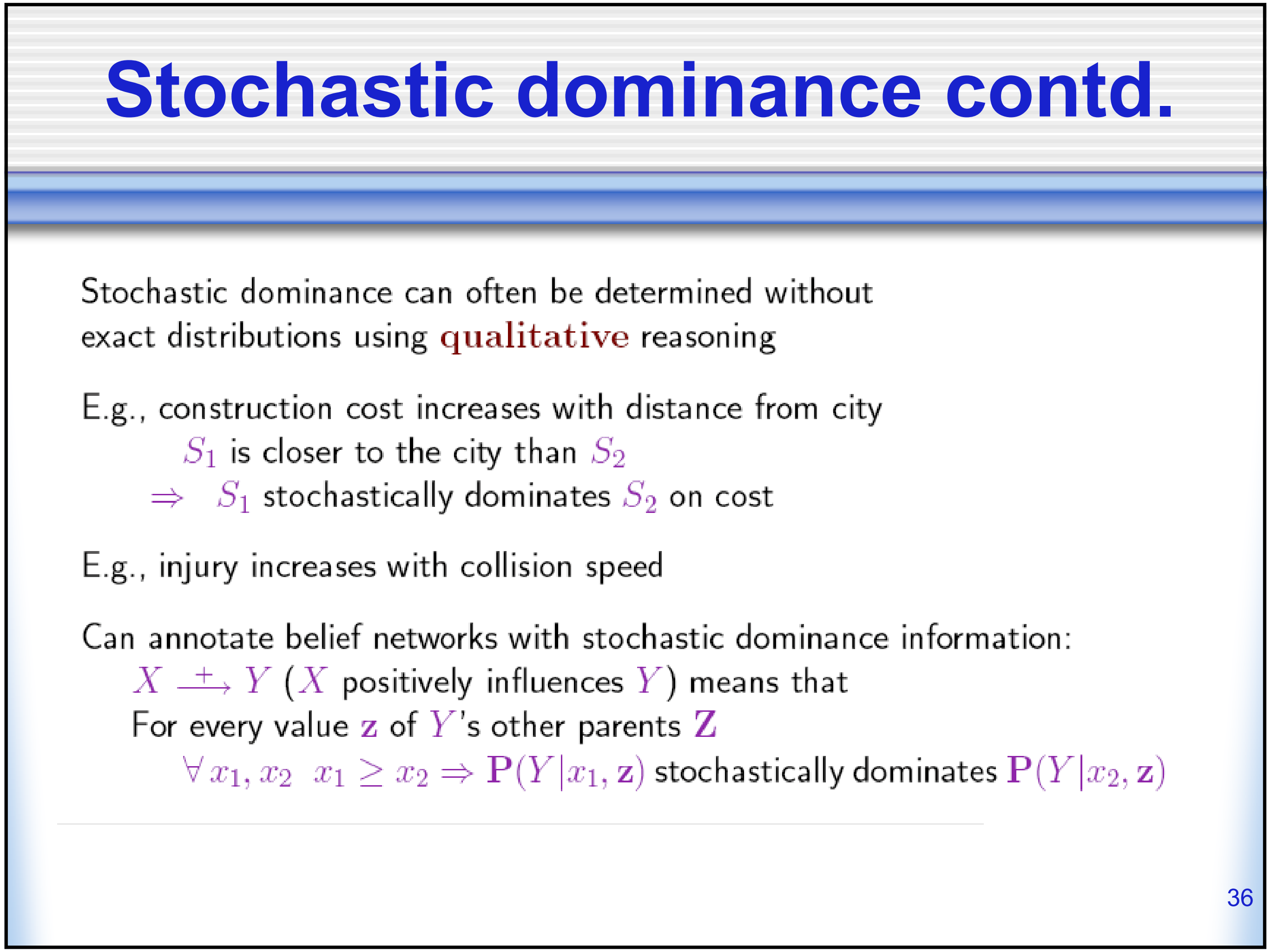


**Figure 16.5** Stochastic dominance. (a) $S_1$ stochastically dominates $S_2$ on cost. (b) Cumulative distributions for the negative cost of $S_1$ and $S_2$.

If two actions S1 and S2 lead to probability distributions $p_1(x)$ and $p_2(x)$ on attribute X, then S1 stochastically dominates S2 on X if:

$$\forall x \quad \int_{-\infty}^{x} p_1(x')\,dx' \leq \int_{-\infty}^{x} p_2(x')\,dx'$$

For any monotonically non-decreasing utility function U(x), the expected utility of S1 is at least as high as the expected utility of S2. Hence, S2 can be discarded.



# Stochastic dominance contd.

Stochastic dominance can often be determined without exact distributions using **qualitative** reasoning

E.g., construction cost increases with distance from city

$S_1$ is closer to the city than $S_2$

$\Rightarrow$ $S_1$ stochastically dominates $S_2$ on cost

E.g., injury increases with collision speed

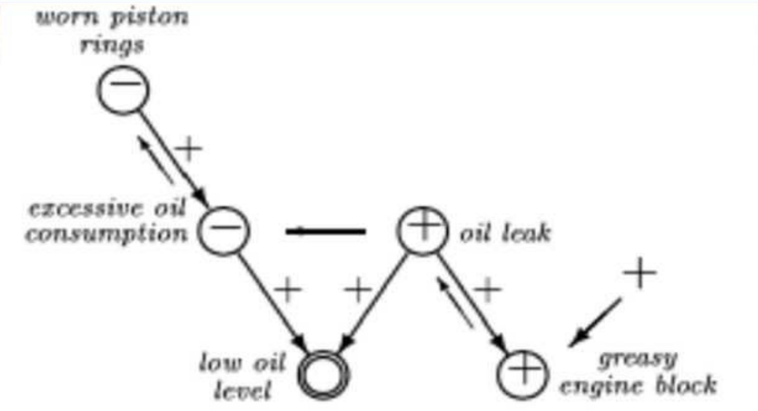
Can annotate belief networks with stochastic dominance information:

$X \xrightarrow{+} Y$ ($X$ positively influences $Y$) means that

For every value $\mathbf{z}$ of $Y$'s other parents $\mathbf{Z}$

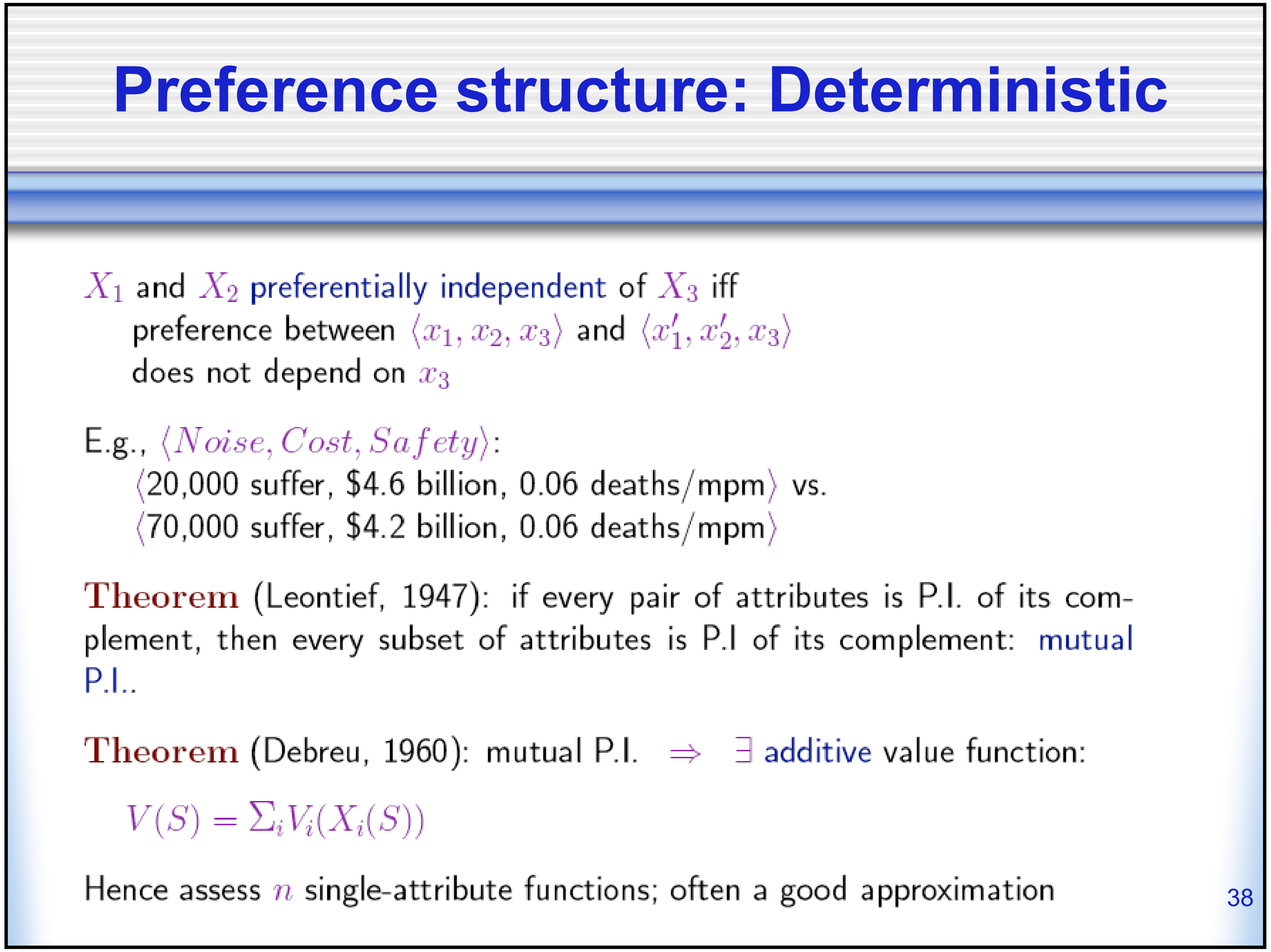
$\forall\, x_1, x_2 \;\; x_1 \geq x_2 \Rightarrow \mathbf{P}(Y|x_1, \mathbf{z})$ stochastically dominates $\mathbf{P}(Y|x_2, \mathbf{z})$

# Example



Qualitative influence of greasy engine block on worn piston rings:

*Greasy engine block is evidence of oil leak.*
*Oil leak and excessive oil consumption can each cause low oil level.*
*Oil leak explains low oil level and so is evidence against excessive oil consumption.*
*Decreased likelihood of excessive oil consumption is evidence against worn piston rings.*
*Therefore, greasy engine block is evidence against worn piston rings.*



# Preference structure: Deterministic

$X_1$ and $X_2$ preferentially independent of $X_3$ iff
preference between $\langle x_1, x_2, x_3 \rangle$ and $\langle x_1', x_2', x_3 \rangle$
does not depend on $x_3$

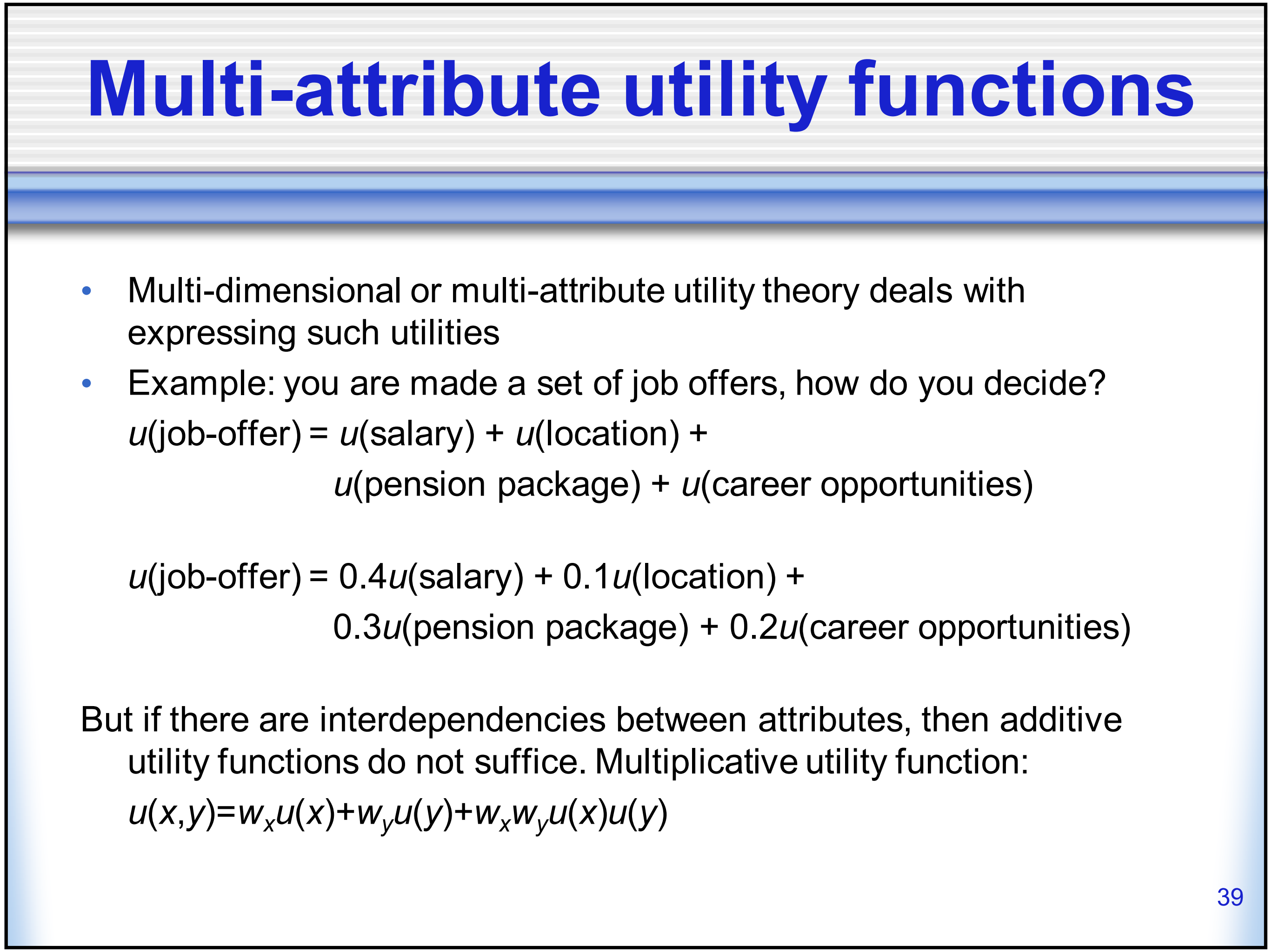
E.g., $\langle Noise, Cost, Safety \rangle$:
⟨20,000 suffer, \$4.6 billion, 0.06 deaths/mpm⟩ vs.
⟨70,000 suffer, \$4.2 billion, 0.06 deaths/mpm⟩

**Theorem** (Leontief, 1947): if every pair of attributes is P.I. of its complement, then every subset of attributes is P.I of its complement: mutual P.I..

**Theorem** (Debreu, 1960): mutual P.I. ⇒ ∃ additive value function:

$$V(S) = \Sigma_i V_i(X_i(S))$$

Hence assess $n$ single-attribute functions; often a good approximation



# Multi-attribute utility functions

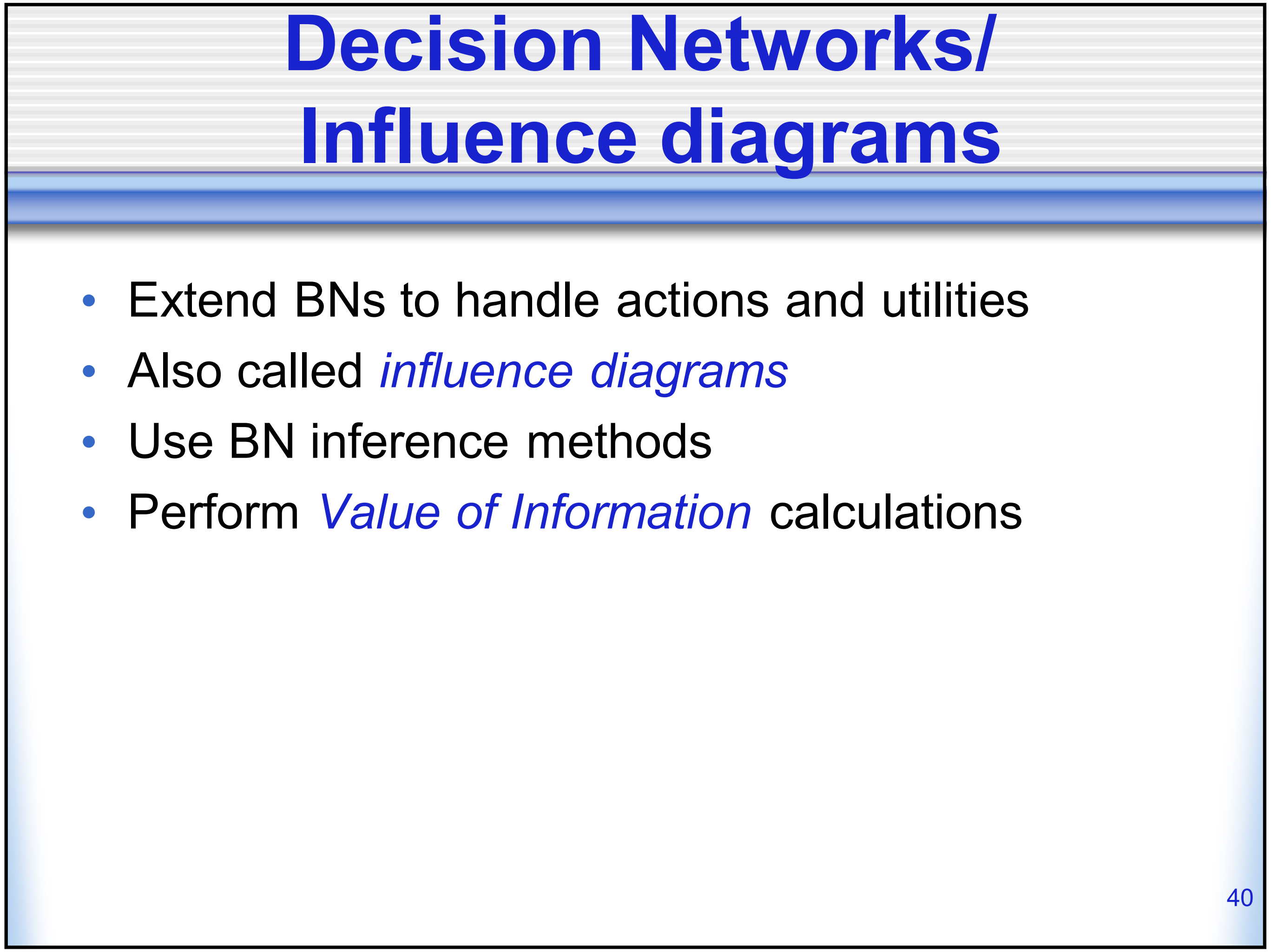
- Multi-dimensional or multi-attribute utility theory deals with expressing such utilities
- Example: you are made a set of job offers, how do you decide?

  $u$(job-offer) = $u$(salary) + $u$(location) +
  $u$(pension package) + $u$(career opportunities)

  $u$(job-offer) = 0.4$u$(salary) + 0.1$u$(location) +
  0.3$u$(pension package) + 0.2$u$(career opportunities)

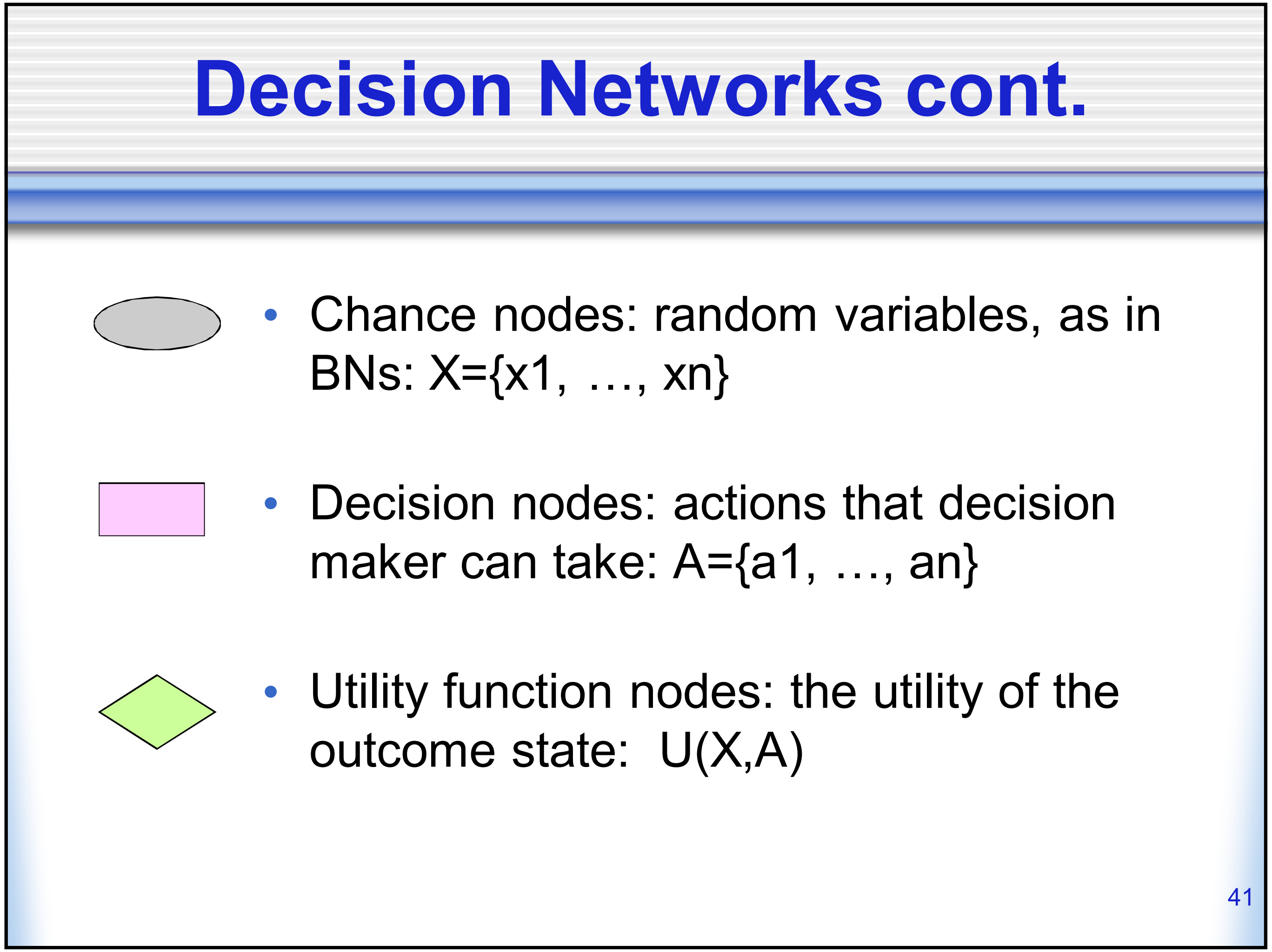
But if there are interdependencies between attributes, then additive utility functions do not suffice. Multiplicative utility function:

$u(x,y)=w_x u(x)+w_y u(y)+w_x w_y u(x)u(y)$



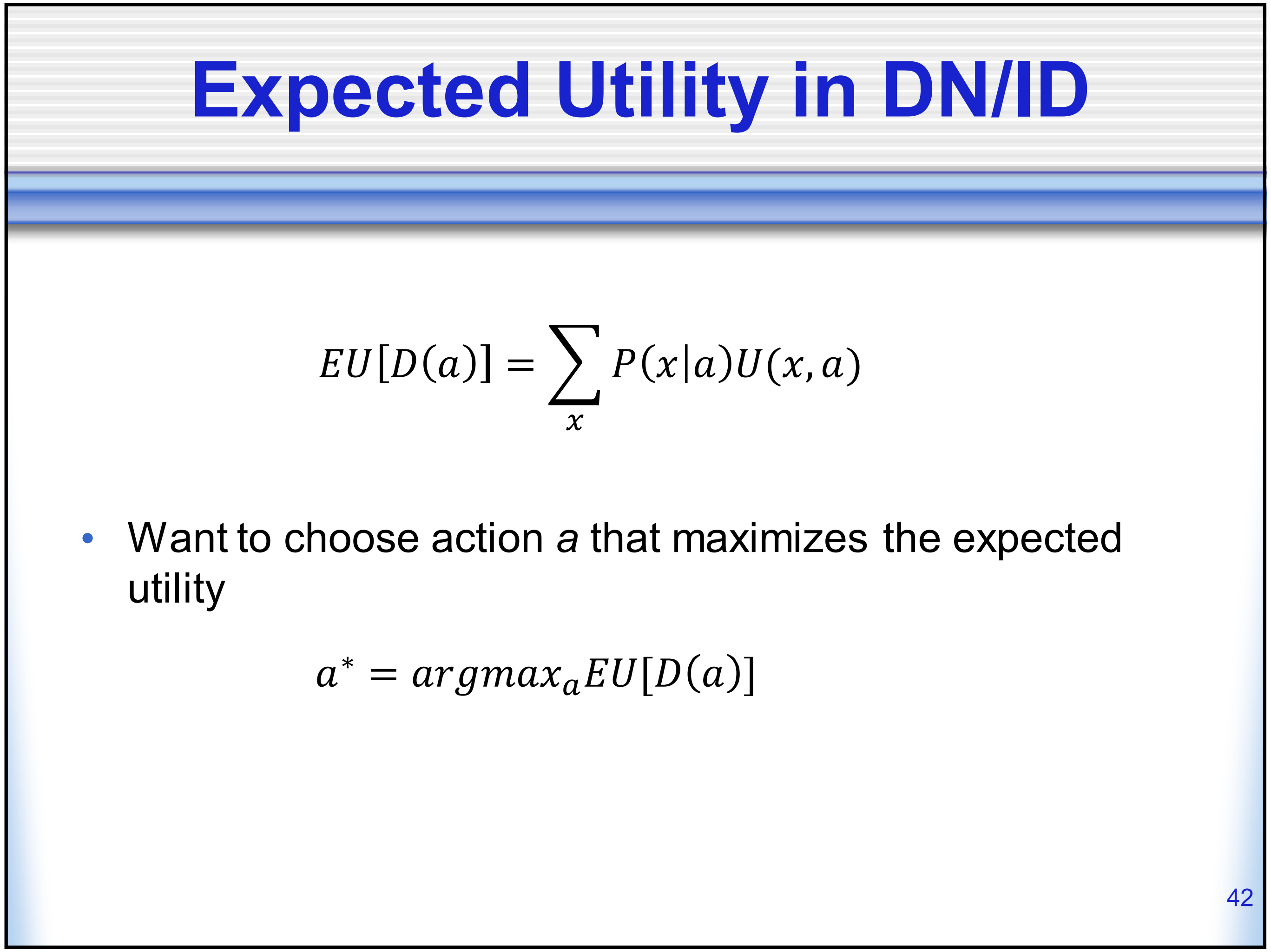
# Decision Networks/ Influence diagrams

- Extend BNs to handle actions and utilities
- Also called *influence diagrams*
- Use BN inference methods
- Perform *Value of Information* calculations



# Decision Networks cont.

- ⬭ Chance nodes: random variables, as in BNs: X={x1, …, xn}
- ▭ Decision nodes: actions that decision maker can take: A={a1, …, an}
- ◇ Utility function nodes: the utility of the outcome state: U(X,A)



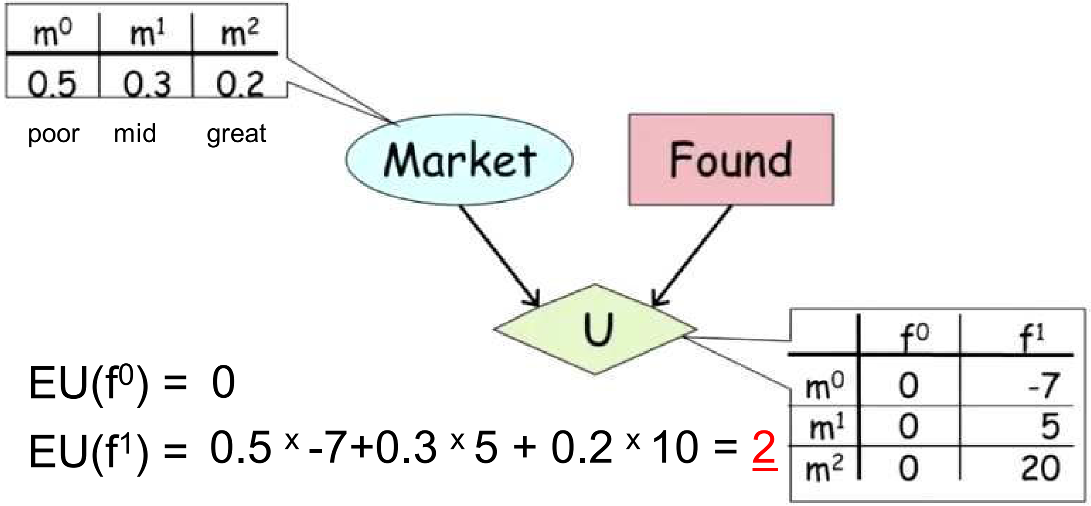
# Expected Utility in DN/ID

$$EU[D(a)] = \sum_{x} P(x|a)U(x,a)$$

- Want to choose action *a* that maximizes the expected utility

$$a^* = argmax_a EU[D(a)]$$

# Simple example



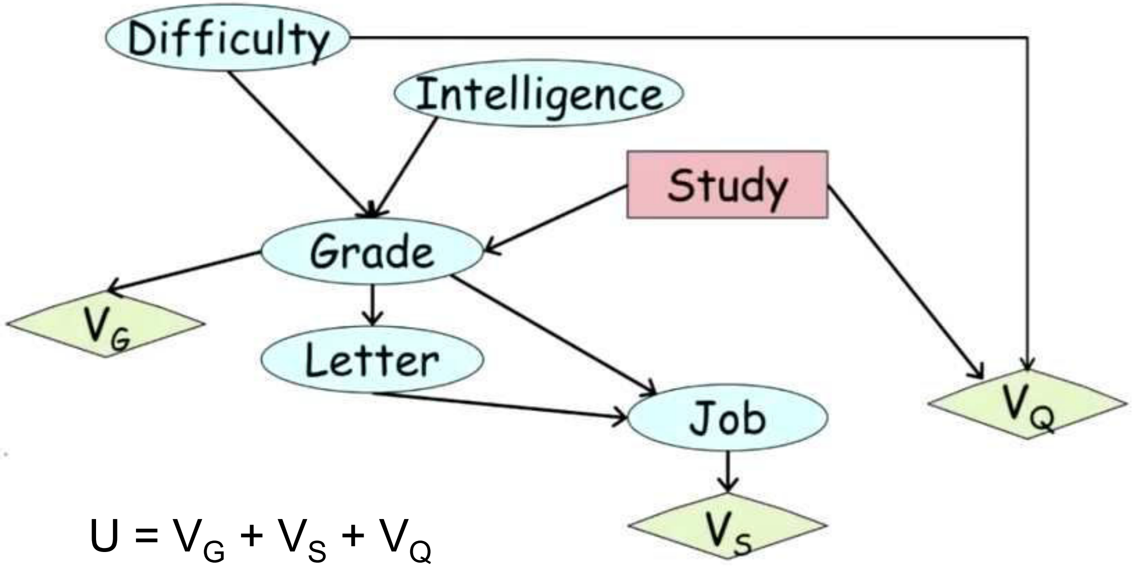
| m0 | m1 | m2 |
|---|---|---|
| 0.5 | 0.3 | 0.2 |
| poor | mid | great |

|  | f0 | f1 |
|---|---|---|
| m0 | 0 | -7 |
| m1 | 0 | 5 |
| m2 | 0 | 20 |

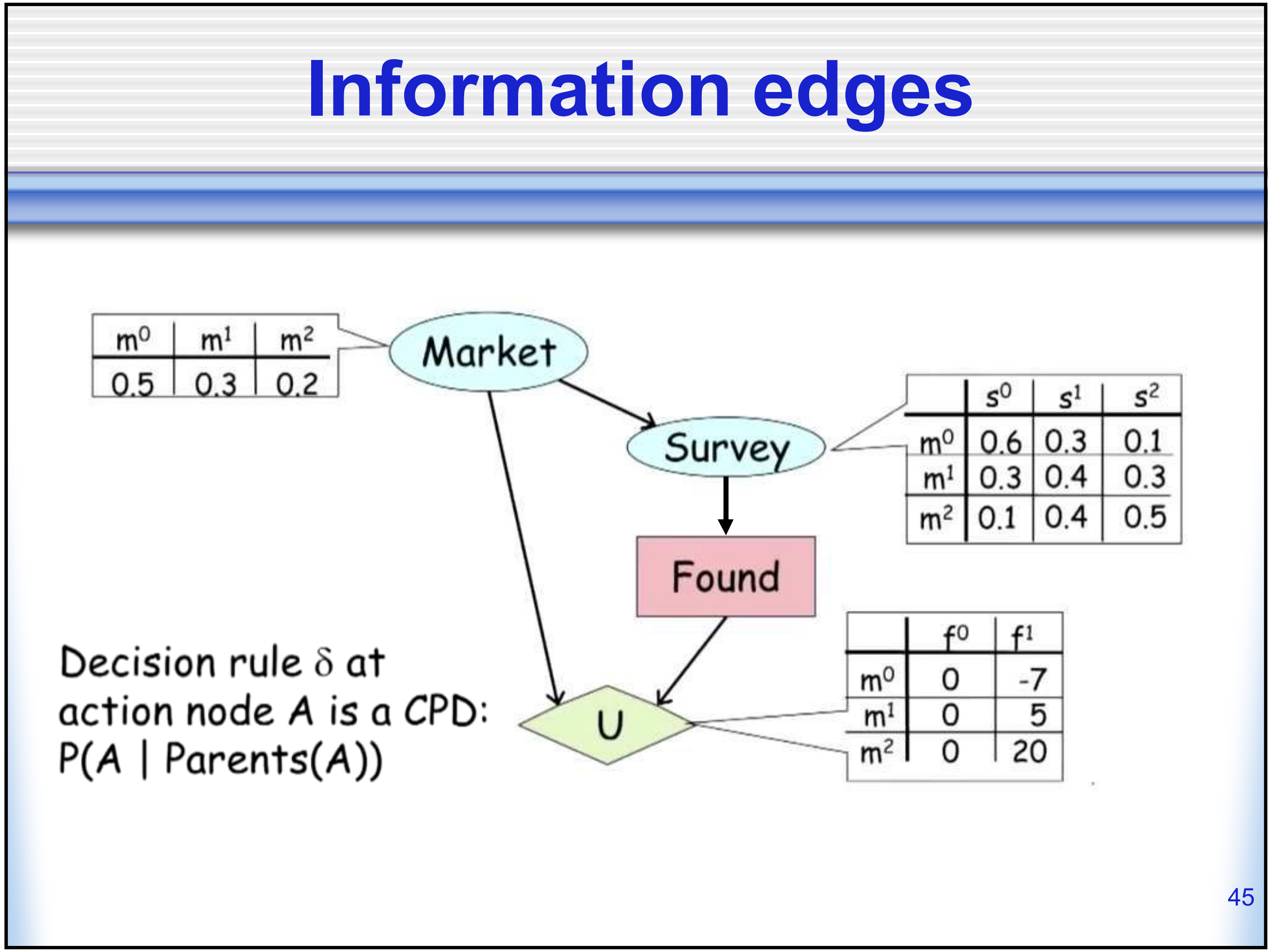
$EU(f^0) = 0$

$EU(f^1) = 0.5 \times -7 + 0.3 \times 5 + 0.2 \times 10 = \underline{2}$

# A more complex network



$U = V_G + V_S + V_Q$



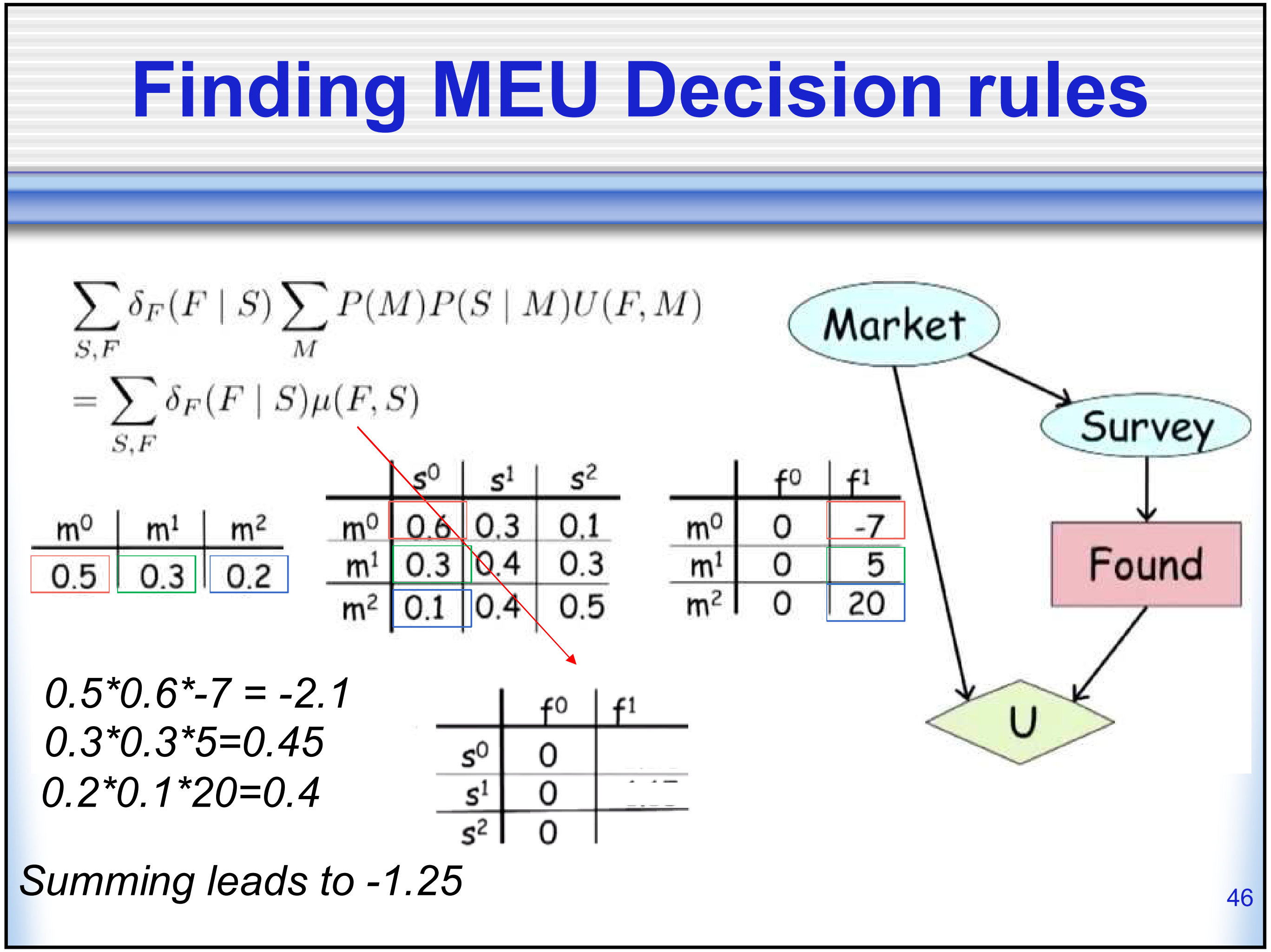
# Information edges

| $m^0$ | $m^1$ | $m^2$ |
|---|---|---|
| 0.5 | 0.3 | 0.2 |

|  | $s^0$ | $s^1$ | $s^2$ |
|---|---|---|---|
| $m^0$ | 0.6 | 0.3 | 0.1 |
| $m^1$ | 0.3 | 0.4 | 0.3 |
| $m^2$ | 0.1 | 0.4 | 0.5 |

|  | $f^0$ | $f^1$ |
|---|---|---|
| $m^0$ | 0 | -7 |
| $m^1$ | 0 | 5 |
| $m^2$ | 0 | 20 |

Decision rule $\delta$ at action node $A$ is a CPD: $P(A \mid Parents(A))$



# Finding MEU Decision rules

$$\sum_{S,F} \delta_F(F \mid S) \sum_{M} P(M)P(S \mid M)U(F,M)$$
$$= \sum_{S,F} \delta_F(F \mid S)\mu(F,S)$$

| $m^0$ | $m^1$ | $m^2$ |
|---|---|---|
| 0.5 | 0.3 | 0.2 |

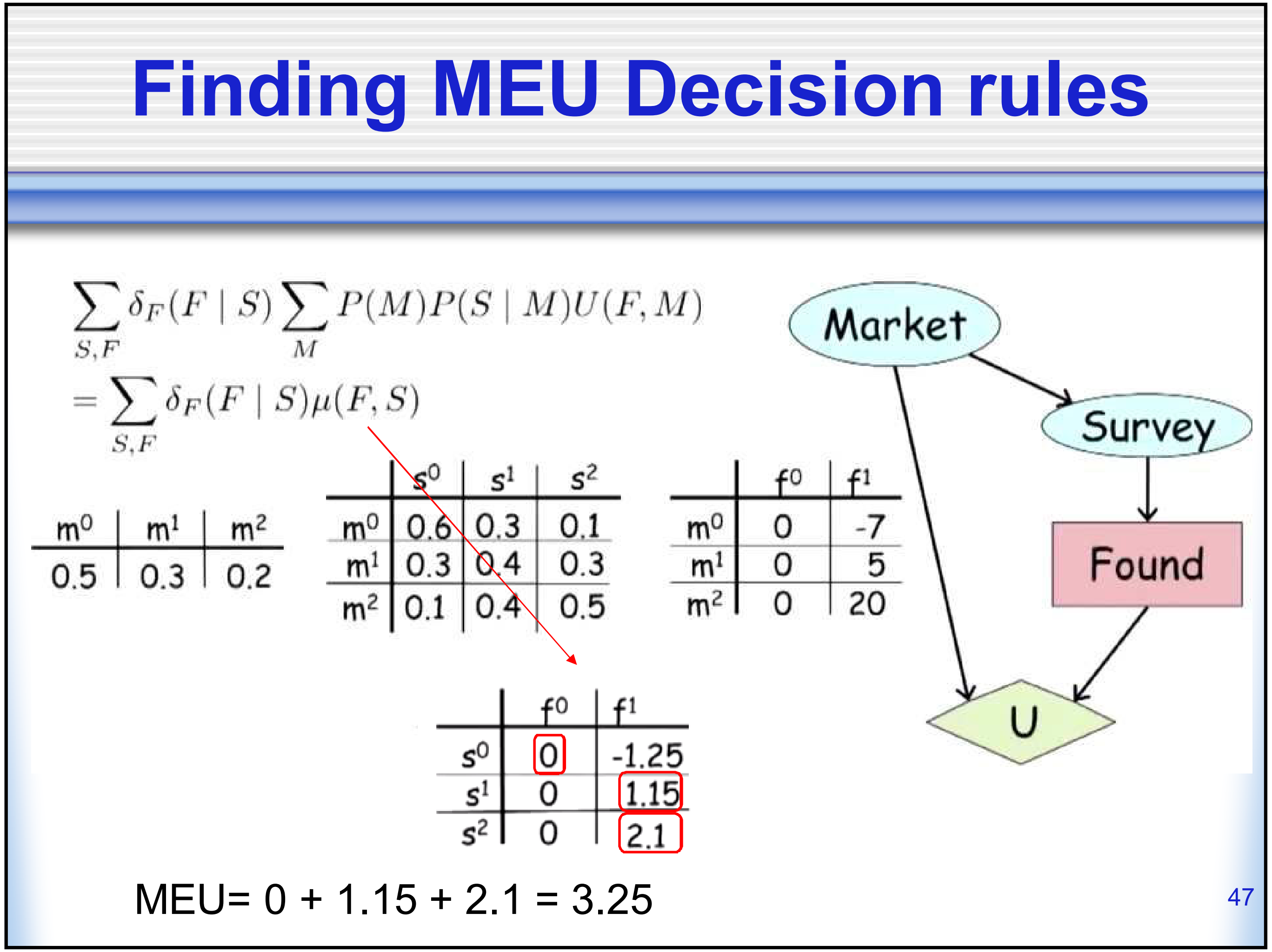
|  | $s^0$ | $s^1$ | $s^2$ |
|---|---|---|---|
| $m^0$ | 0.6 | 0.3 | 0.1 |
| $m^1$ | 0.3 | 0.4 | 0.3 |
| $m^2$ | 0.1 | 0.4 | 0.5 |

|  | $f^0$ | $f^1$ |
|---|---|---|
| $m^0$ | 0 | -7 |
| $m^1$ | 0 | 5 |
| $m^2$ | 0 | 20 |

0.5*0.6*-7 = -2.1
0.3*0.3*5=0.45
0.2*0.1*20=0.4

|  | $f^0$ | $f^1$ |
|---|---|---|
| $s^0$ | 0 |  |
| $s^1$ | 0 |  |
| $s^2$ | 0 |  |

Summing leads to -1.25



# Finding MEU Decision rules

$$\sum_{S,F} \delta_F(F \mid S) \sum_M P(M)P(S \mid M)U(F,M)$$

$$= \sum_{S,F} \delta_F(F \mid S)\mu(F,S)$$

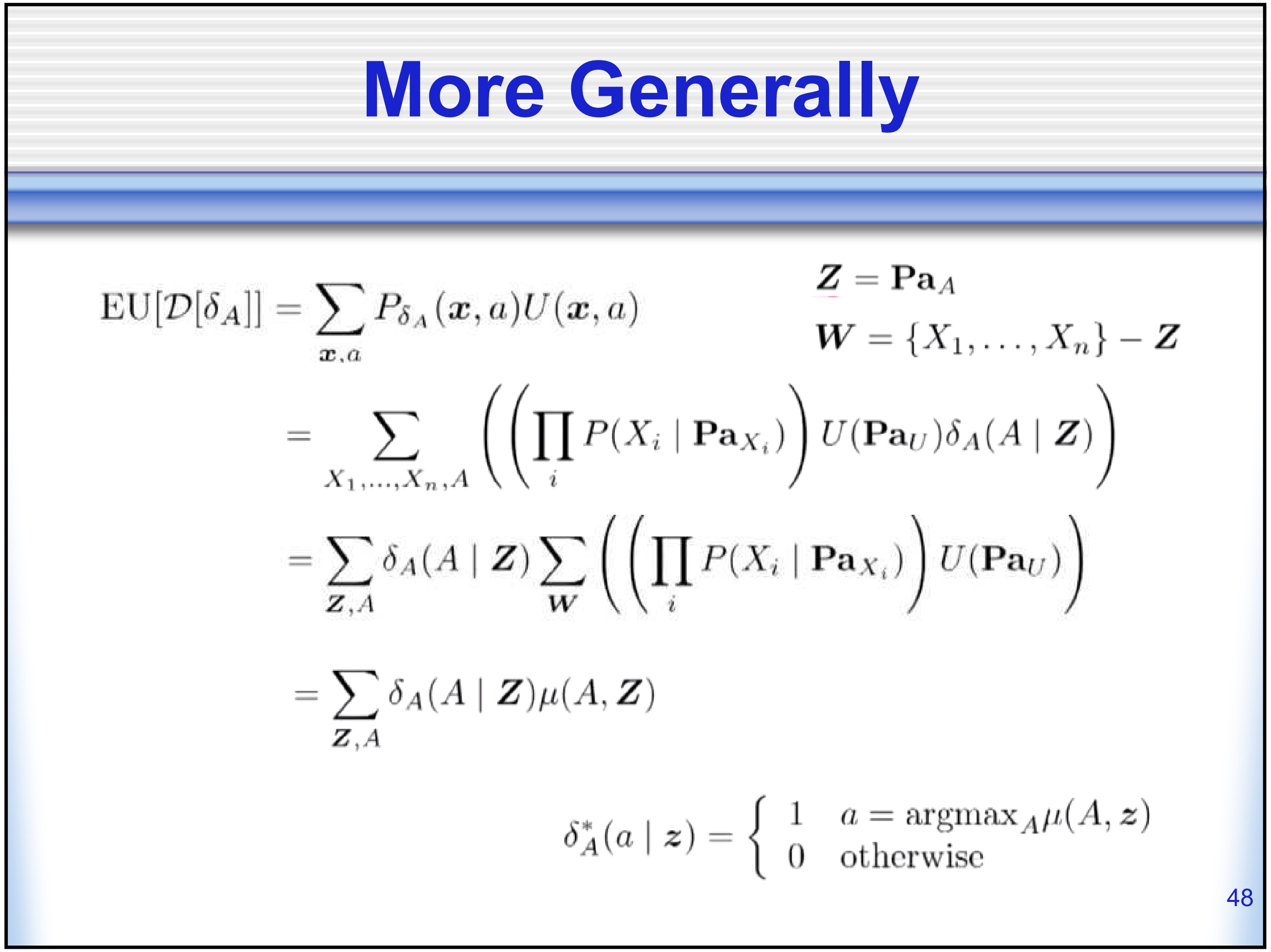
| $m^0$ | $m^1$ | $m^2$ |
|---|---|---|
| 0.5 | 0.3 | 0.2 |

|  | $s^0$ | $s^1$ | $s^2$ |
|---|---|---|---|
| $m^0$ | 0.6 | 0.3 | 0.1 |
| $m^1$ | 0.3 | 0.4 | 0.3 |
| $m^2$ | 0.1 | 0.4 | 0.5 |

|  | $f^0$ | $f^1$ |
|---|---|---|
| $m^0$ | 0 | -7 |
| $m^1$ | 0 | 5 |
| $m^2$ | 0 | 20 |

|  | $f^0$ | $f^1$ |
|---|---|---|
| $s^0$ | $\boxed{0}$ | -1.25 |
| $s^1$ | 0 | $\boxed{1.15}$ |
| $s^2$ | 0 | $\boxed{2.1}$ |

MEU= 0 + 1.15 + 2.1 = 3.25



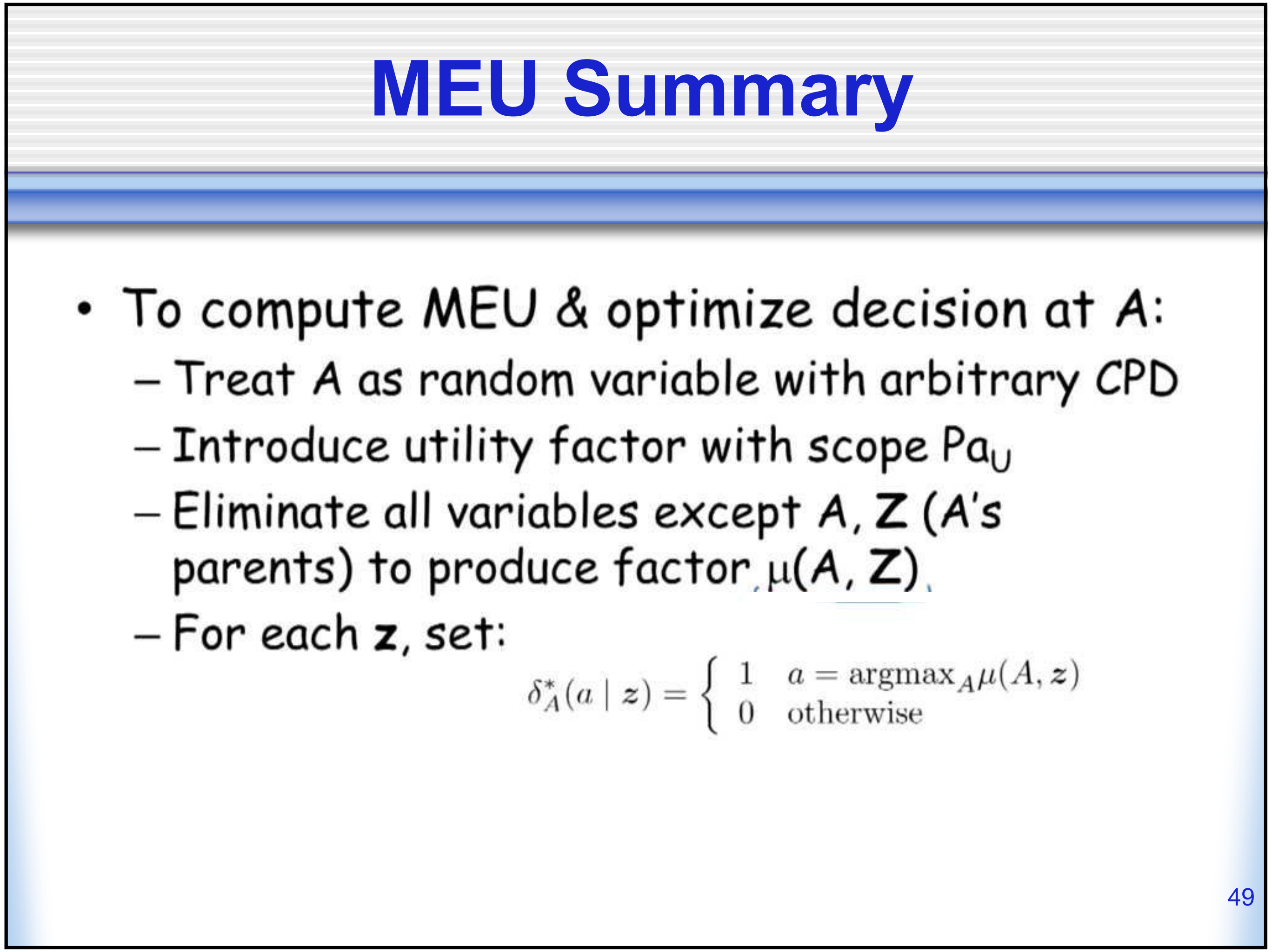
# More Generally

$$\boldsymbol{Z} = \mathbf{Pa}_A$$

$$\boldsymbol{W} = \{X_1, \ldots, X_n\} - \boldsymbol{Z}$$

$$\begin{aligned}\mathrm{EU}[\mathcal{D}[\delta_A]] &= \sum_{\boldsymbol{x},a} P_{\delta_A}(\boldsymbol{x},a)U(\boldsymbol{x},a) \\ &= \sum_{X_1,\ldots,X_n,A}\left(\left(\prod_i P(X_i \mid \mathbf{Pa}_{X_i})\right)U(\mathbf{Pa}_U)\delta_A(A \mid \boldsymbol{Z})\right) \\ &= \sum_{\boldsymbol{Z},A}\delta_A(A \mid \boldsymbol{Z})\sum_{\boldsymbol{W}}\left(\left(\prod_i P(X_i \mid \mathbf{Pa}_{X_i})\right)U(\mathbf{Pa}_U)\right) \\ &= \sum_{\boldsymbol{Z},A}\delta_A(A \mid \boldsymbol{Z})\mu(A,\boldsymbol{Z})\end{aligned}$$

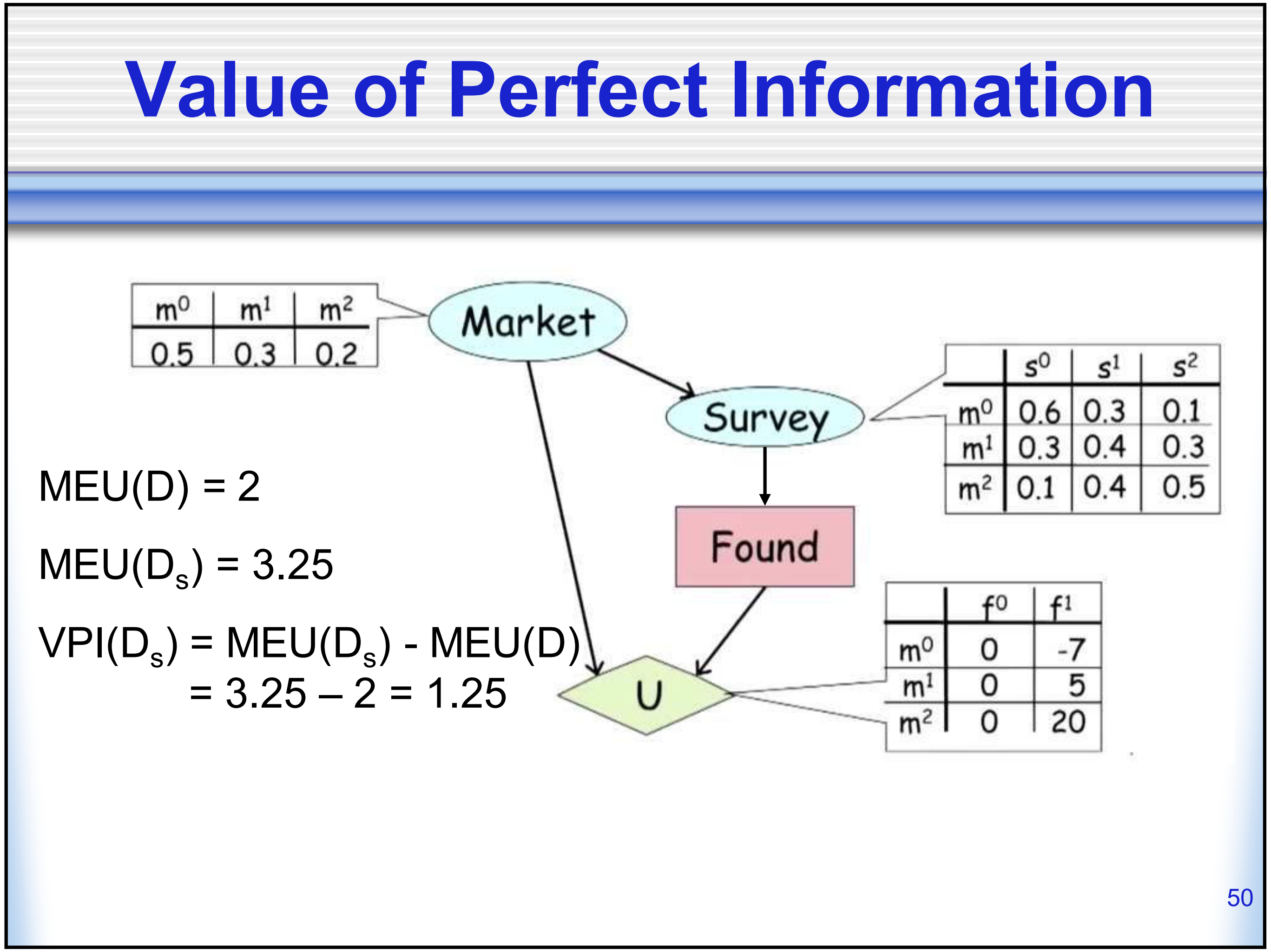
$$\delta^*_A(a \mid \boldsymbol{z}) = \begin{cases} 1 & a = \mathrm{argmax}_A \mu(A,\boldsymbol{z}) \\ 0 & \text{otherwise}\end{cases}$$



# MEU Summary

- To compute MEU & optimize decision at $A$:
  - Treat $A$ as random variable with arbitrary CPD
  - Introduce utility factor with scope $Pa_U$
  - Eliminate all variables except $A$, $\mathbf{Z}$ ($A$'s parents) to produce factor $\mu(A, \mathbf{Z})$
  - For each $\mathbf{z}$, set:

$$\delta^*_A(a \mid \boldsymbol{z}) = \begin{cases} 1 & a = \text{argmax}_A \mu(A, \boldsymbol{z}) \\ 0 & \text{otherwise} \end{cases}$$



# Value of Perfect Information

Market

| $m^0$ | $m^1$ | $m^2$ |
|---|---|---|
| 0.5 | 0.3 | 0.2 |

Survey

| | $s^0$ | $s^1$ | $s^2$ |
|---|---|---|---|
| $m^0$ | 0.6 | 0.3 | 0.1 |
| $m^1$ | 0.3 | 0.4 | 0.3 |
| $m^2$ | 0.1 | 0.4 | 0.5 |

Found

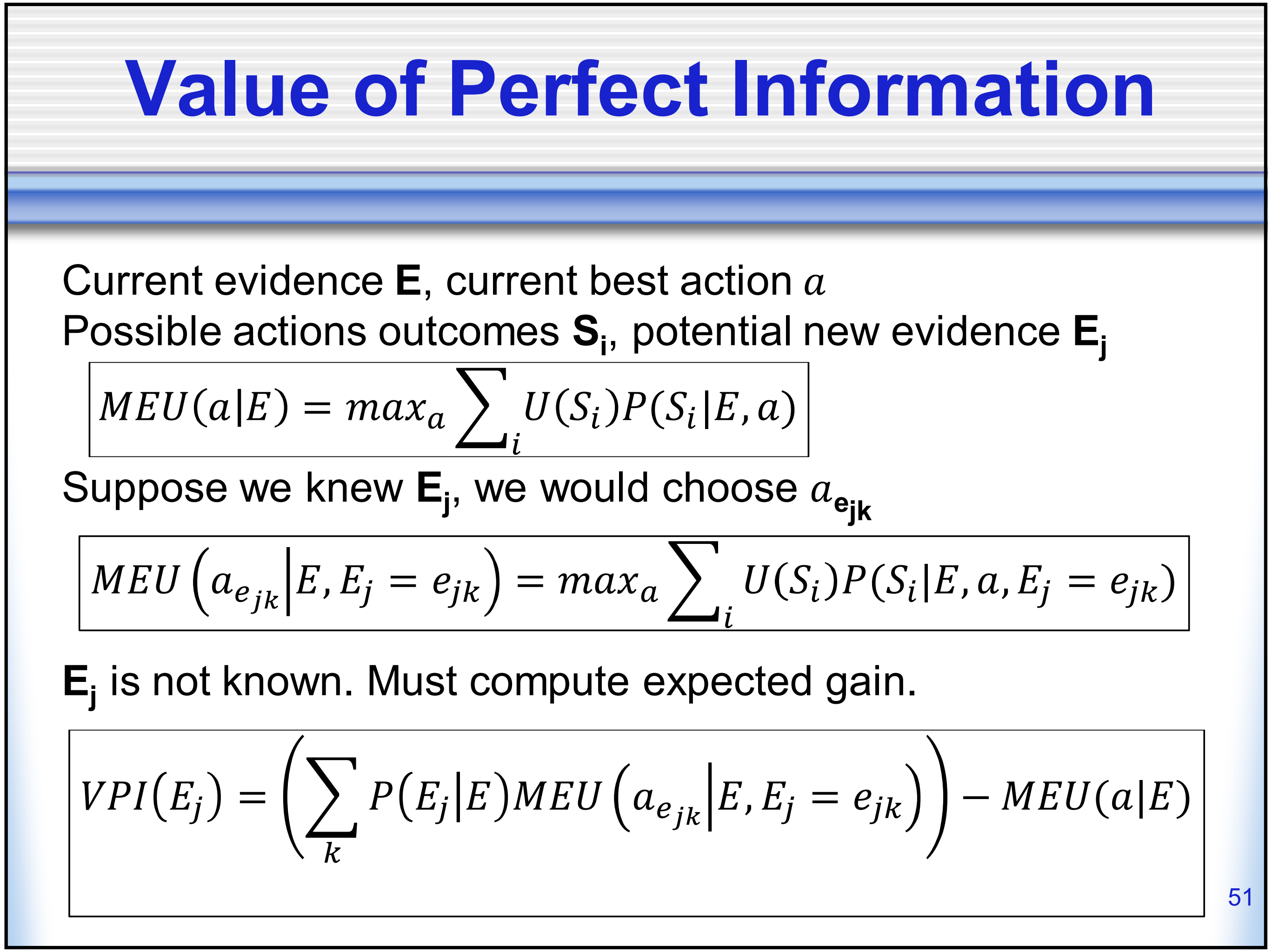
U

| | $f^0$ | $f^1$ |
|---|---|---|
| $m^0$ | 0 | -7 |
| $m^1$ | 0 | 5 |
| $m^2$ | 0 | 20 |

MEU(D) = 2

MEU($D_s$) = 3.25

VPI($D_s$) = MEU($D_s$) - MEU(D)
= 3.25 – 2 = 1.25



# Value of Perfect Information

Current evidence **E**, current best action $a$

Possible actions outcomes **S$_i$**, potential new evidence **E$_j$**

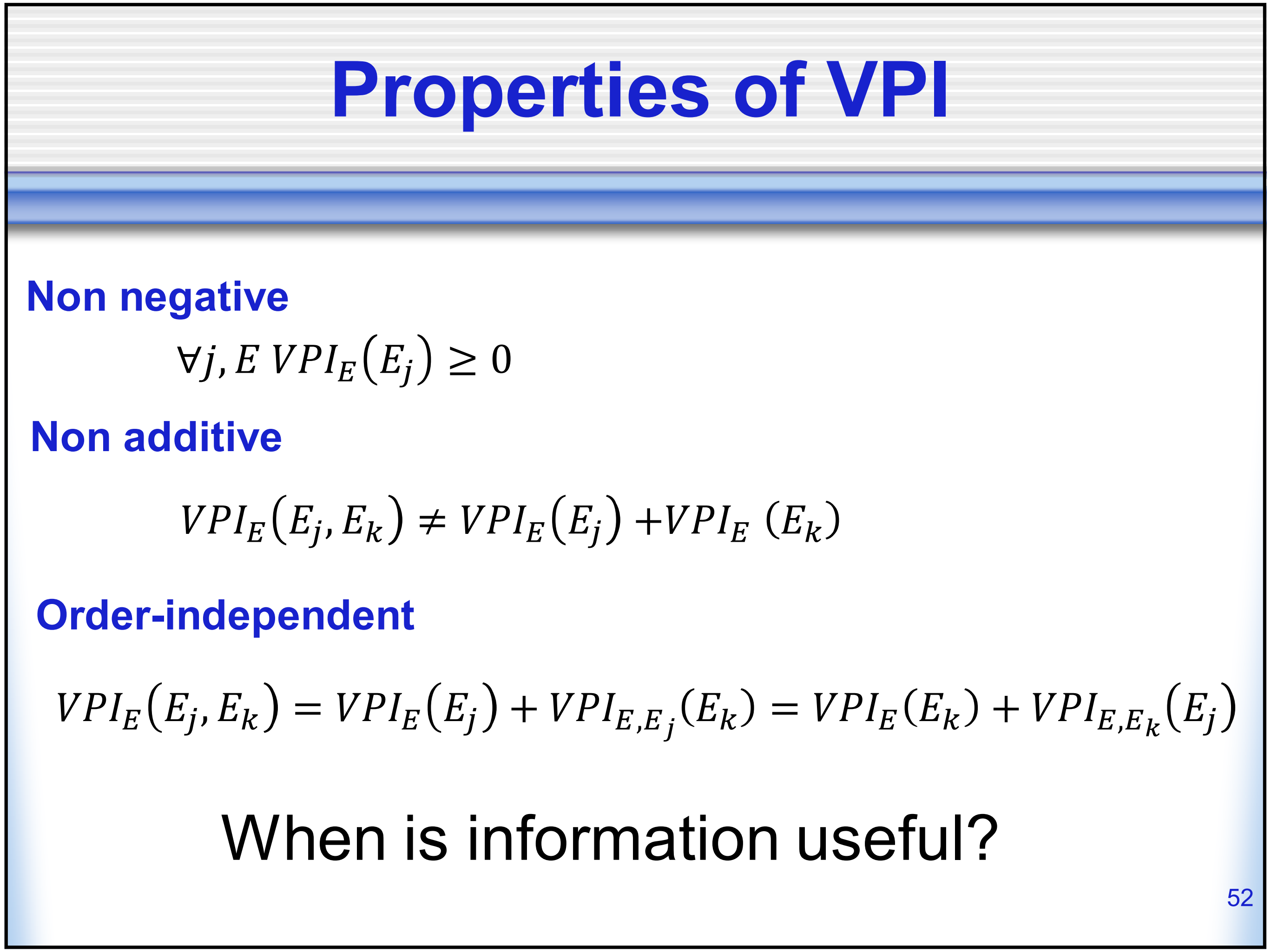
$$\boxed{MEU(a|E) = max_a \sum_i U(S_i)P(S_i|E,a)}$$

Suppose we knew **E$_j$**, we would choose $a_{\mathbf{ejk}}$

$$\boxed{MEU\left(a_{e_{jk}}\middle|E, E_j = e_{jk}\right) = max_a \sum_i U(S_i)P(S_i|E,a,E_j = e_{jk})}$$

**E$_j$** is not known. Must compute expected gain.

$$\boxed{VPI(E_j) = \left(\sum_k P(E_j|E)MEU\left(a_{e_{jk}}\middle|E, E_j = e_{jk}\right)\right) - MEU(a|E)}$$



# Properties of VPI

**Non negative**

$$\forall j, E\ VPI_E(E_j) \geq 0$$

**Non additive**

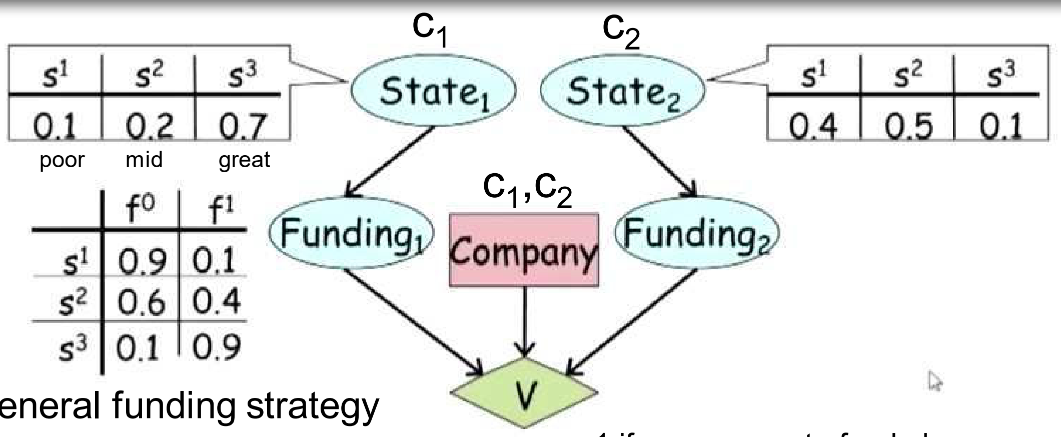
$$VPI_E(E_j, E_k) \neq VPI_E(E_j) + VPI_E\ (E_k)$$

**Order-independent**

$$VPI_E(E_j, E_k) = VPI_E(E_j) + VPI_{E,E_j}(E_k) = VPI_E(E_k) + VPI_{E,E_k}(E_j)$$

When is information useful?

# Example 1



| $s^1$ | $s^2$ | $s^3$ |
|---|---|---|
| 0.1 | 0.2 | 0.7 |
| poor | mid | great |

| $s^1$ | $s^2$ | $s^3$ |
|---|---|---|
| 0.4 | 0.5 | 0.1 |

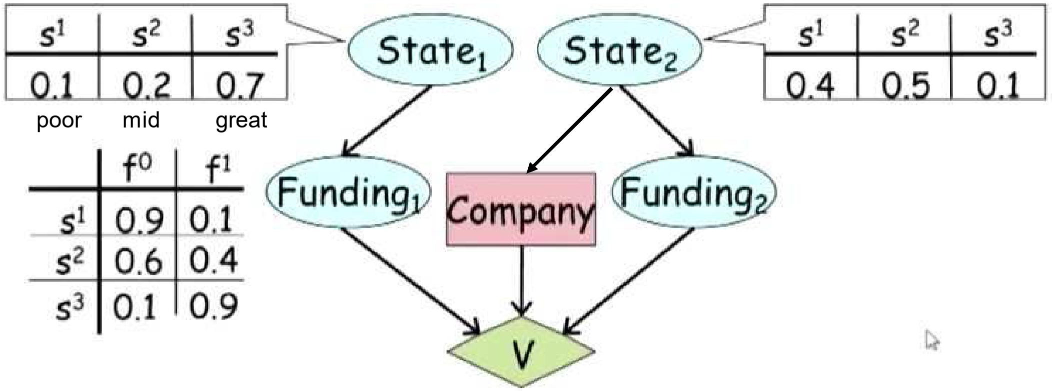
| | $f^0$ | $f^1$ |
|---|---|---|
| $s^1$ | 0.9 | 0.1 |
| $s^2$ | 0.6 | 0.4 |
| $s^3$ | 0.1 | 0.9 |

General funding strategy

1 if company gets funded
0 otherwise

$$EU(D[c_1]) = 0.1*0.1+0.2*0.4+0.7*0.9=0.72$$
$$EU(D[c_2]) = 0.4*0.1+0.5*0.4+0.1*0.9=0.33$$

# Example 1



| $s^1$ | $s^2$ | $s^3$ |
|---|---|---|
| 0.1 | 0.2 | 0.7 |
| poor | mid | great |

| $s^1$ | $s^2$ | $s^3$ |
|---|---|---|
| 0.4 | 0.5 | 0.1 |

|  | $f^0$ | $f^1$ |
|---|---|---|
| $s^1$ | 0.9 | 0.1 |
| $s^2$ | 0.6 | 0.4 |
| $s^3$ | 0.1 | 0.9 |

1 if company gets funded
0 otherwise

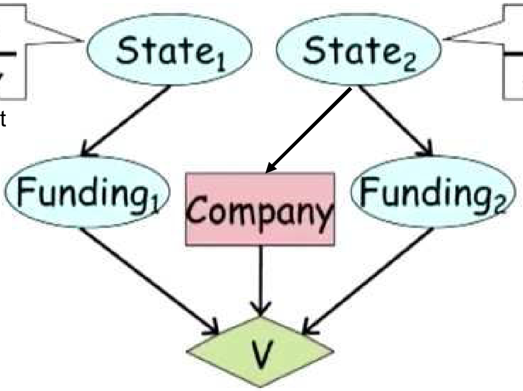
EU(D[$c_1$]) = 0.72
EU(D[$c_2$]) = 0.33

*If c2 is in state s1, the utility is 0.1*
*If c2 is in state s2, the utility is 0.4*
*If c2 is in state s3, the utility is 0.9*

# Example 1

| $s^1$ | $s^2$ | $s^3$ |
|---|---|---|
| 0.1 | 0.2 | 0.7 |
| poor | mid | great |

|  | $f^0$ | $f^1$ |
|---|---|---|
| $s^1$ | 0.9 | 0.1 |
| $s^2$ | 0.6 | 0.4 |
| $s^3$ | 0.1 | 0.9 |



| $s^1$ | $s^2$ | $s^3$ |
|---|---|---|
| 0.4 | 0.5 | 0.1 |

1 if company gets funded
0 otherwise

$$MEU(D_{State2}) = 0.4 * 0.72 = 0.288$$

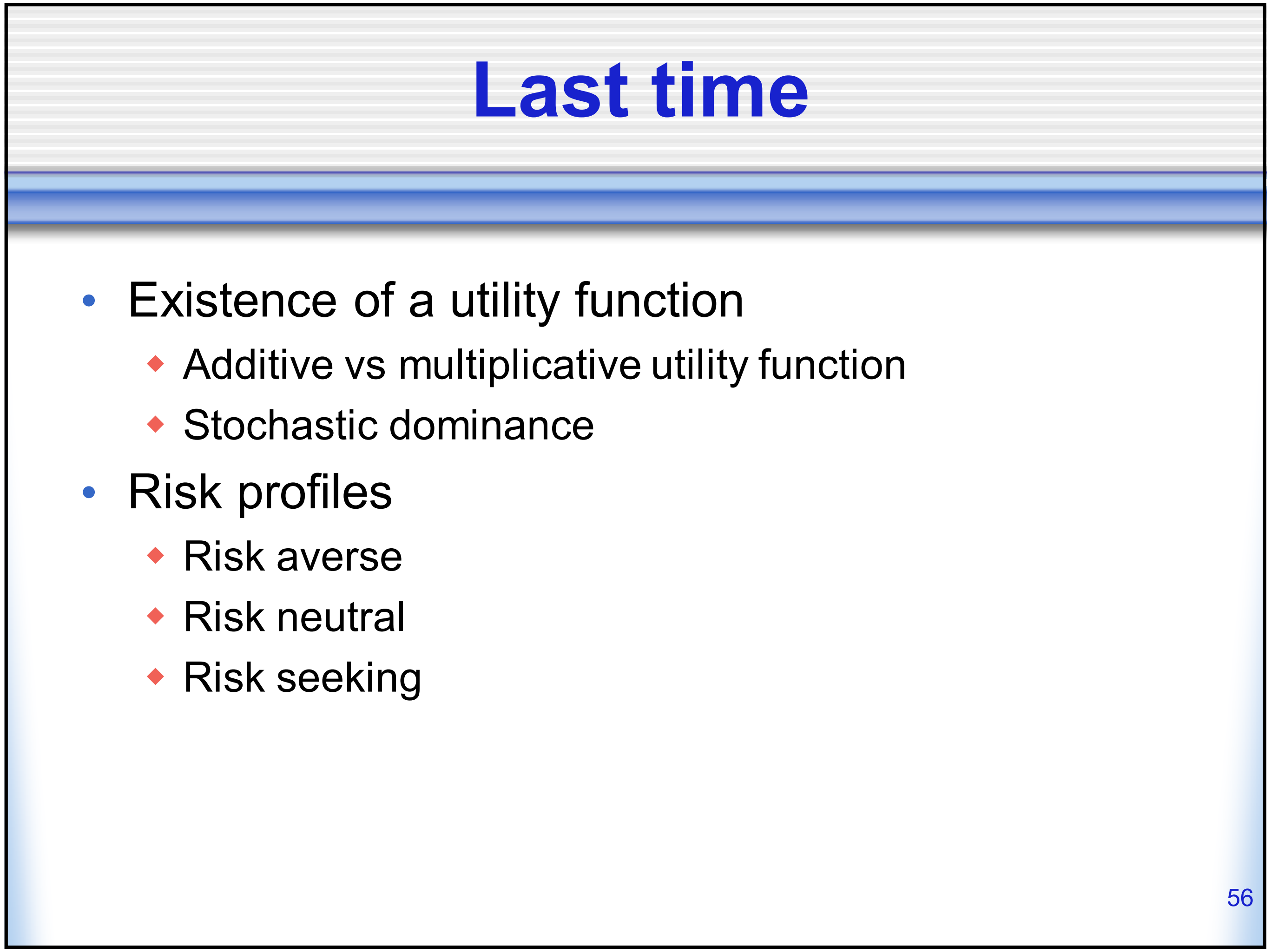
$$0.5 * 0.72 = 0.36$$

$$0.1 * 0.9 = 0.09$$

$$\sum 0.738$$

$EU(D[c_1]) = 0.72$

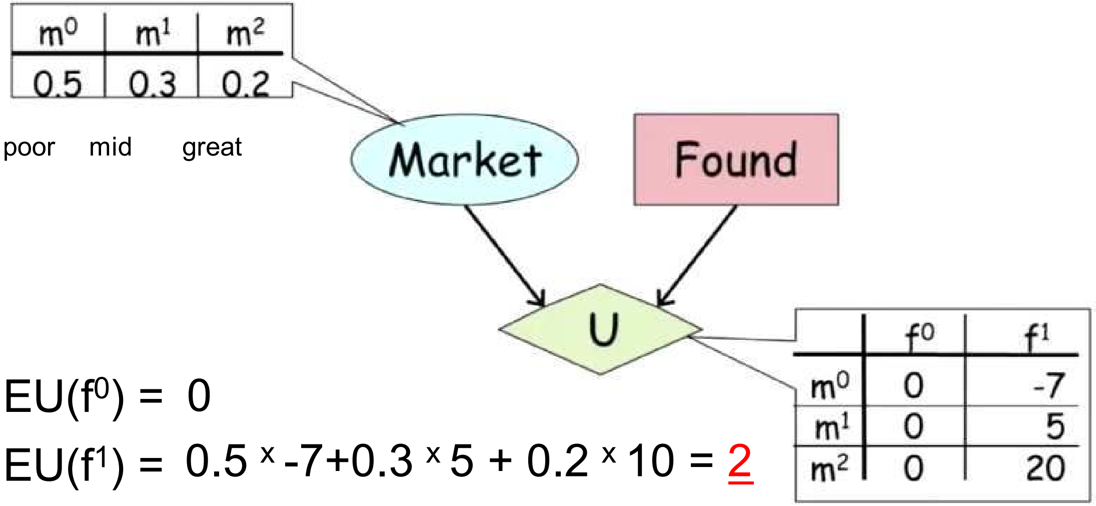
$EU(D[c_2]) = 0.33$



# Last time

- Existence of a utility function
  - Additive vs multiplicative utility function
  - Stochastic dominance
- Risk profiles
  - Risk averse
  - Risk neutral
  - Risk seeking

# Last time: Decision networks



| $m^0$ | $m^1$ | $m^2$ |
|---|---|---|
| 0.5 | 0.3 | 0.2 |

poor mid great

|  | $f^0$ | $f^1$ |
|---|---|---|
| $m^0$ | 0 | -7 |
| $m^1$ | 0 | 5 |
| $m^2$ | 0 | 20 |

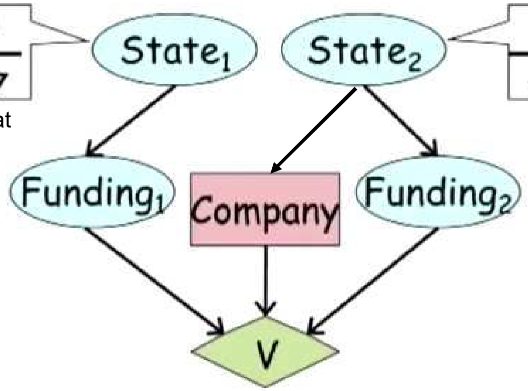
$EU(f^0) = 0$

$EU(f^1) = 0.5 \times -7 + 0.3 \times 5 + 0.2 \times 10 = \underline{2}$

# Example 1

| $s^1$ | $s^2$ | $s^3$ |
|---|---|---|
| 0.1 | 0.2 | 0.7 |
| poor | mid | great |

|  | $f^0$ | $f^1$ |
|---|---|---|
| $s^1$ | 0.9 | 0.1 |
| $s^2$ | 0.6 | 0.4 |
| $s^3$ | 0.1 | 0.9 |



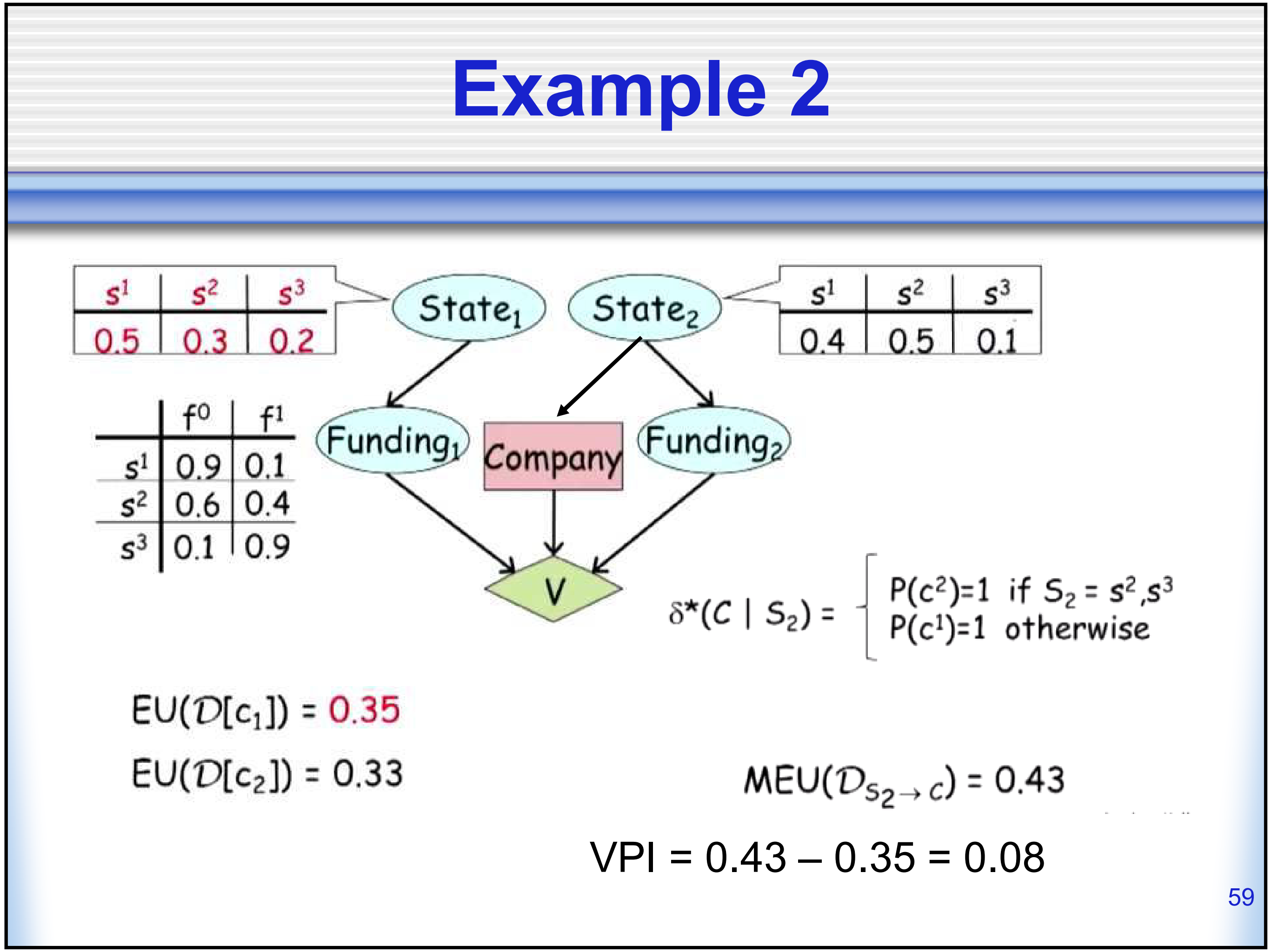
| $s^1$ | $s^2$ | $s^3$ |
|---|---|---|
| 0.4 | 0.5 | 0.1 |

1 if company gets funded
0 otherwise
Select $c_2$ if $State_2 = s_3$, $c_1$ otherwise

$EU(D[c_1]) = 0.72$
$EU(D[c_2]) = 0.33$

$MEU(D_{State2}) = 0.738$

$VPI(D_{State2}) = 0.738 - 0.72 = 0.018$



# Example 2

| $s^1$ | $s^2$ | $s^3$ |
|---|---|---|
| 0.5 | 0.3 | 0.2 |

| $s^1$ | $s^2$ | $s^3$ |
|---|---|---|
| 0.4 | 0.5 | 0.1 |

|  | $f^0$ | $f^1$ |
|---|---|---|
| $s^1$ | 0.9 | 0.1 |
| $s^2$ | 0.6 | 0.4 |
| $s^3$ | 0.1 | 0.9 |

$$\delta^*(C \mid S_2) = \begin{cases} P(c^2)=1 & \text{if } S_2 = s^2, s^3 \\ P(c^1)=1 & \text{otherwise} \end{cases}$$

$EU(\mathcal{D}[c_1]) = 0.35$

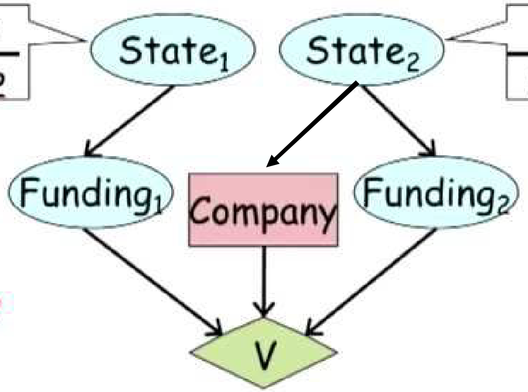
$EU(\mathcal{D}[c_2]) = 0.33$

$MEU(\mathcal{D}_{S_2 \rightarrow C}) = 0.43$

VPI = 0.43 – 0.35 = 0.08

# Example 3

| $s^1$ | $s^2$ | $s^3$ |
|---|---|---|
| 0.5 | 0.3 | 0.2 |



| $s^1$ | $s^2$ | $s^3$ |
|---|---|---|
| 0.4 | 0.5 | 0.1 |

|  | $f^0$ | $f^1$ |
|---|---|---|
| $s^1$ | 0.3 | 0.7 |
| $s^2$ | 0.2 | 0.8 |
| $s^3$ | 0.01 | 0.99 |

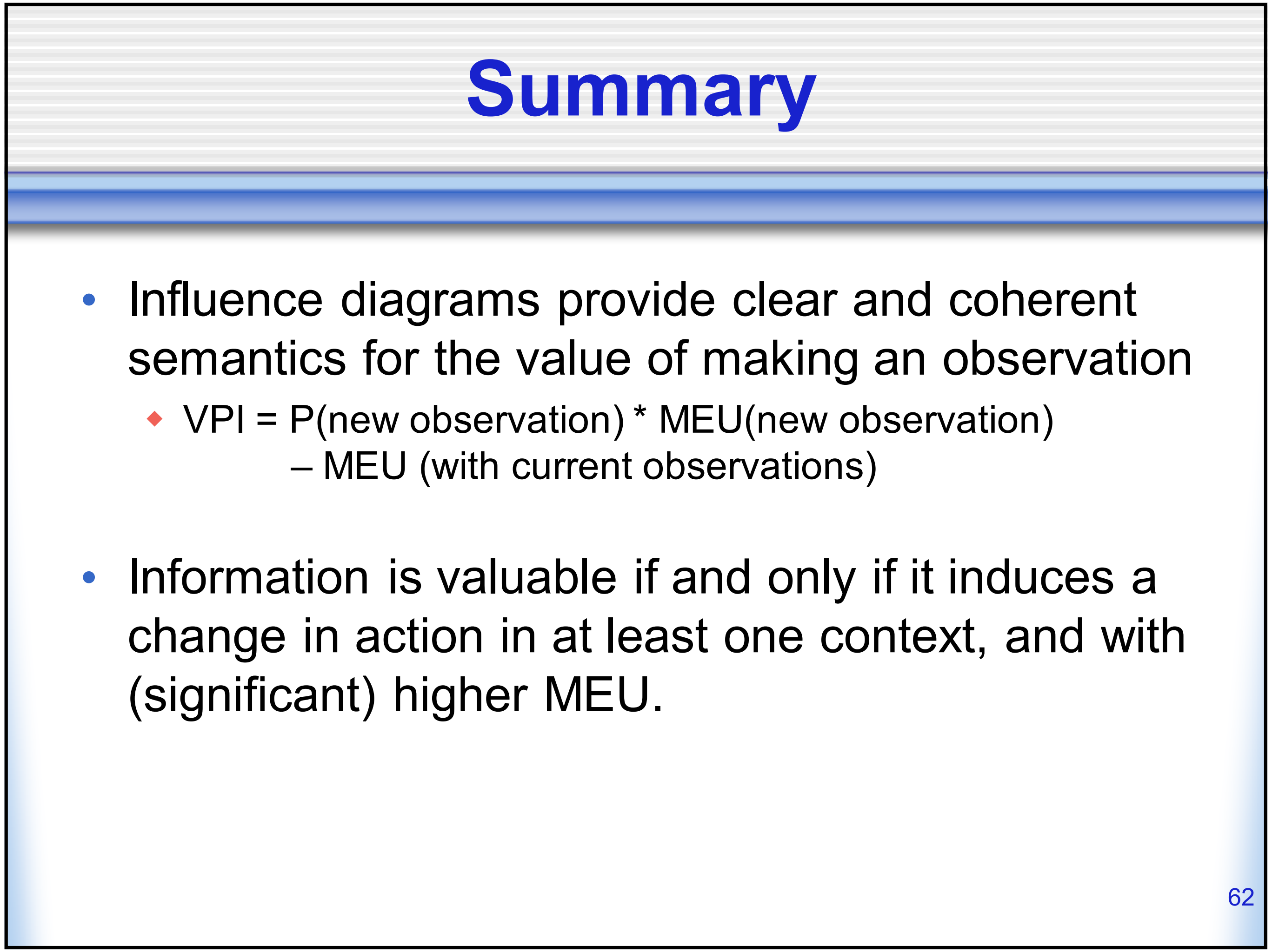
$EU(\mathcal{D}[c_1]) = 0.788$

$EU(\mathcal{D}[c_2]) = 0.779$

$$\delta^*(C \mid S_2) = \begin{cases} P(c^2)=1 & \text{if } S_2 = s^2, s^3 \\ P(c^1)=1 & \text{otherwise} \end{cases}$$

$MEU(\mathcal{D}_{S_2 \to C}) = 0.8142$

VPI = 0.8142 – 0.788 = 0.0262



# Summary

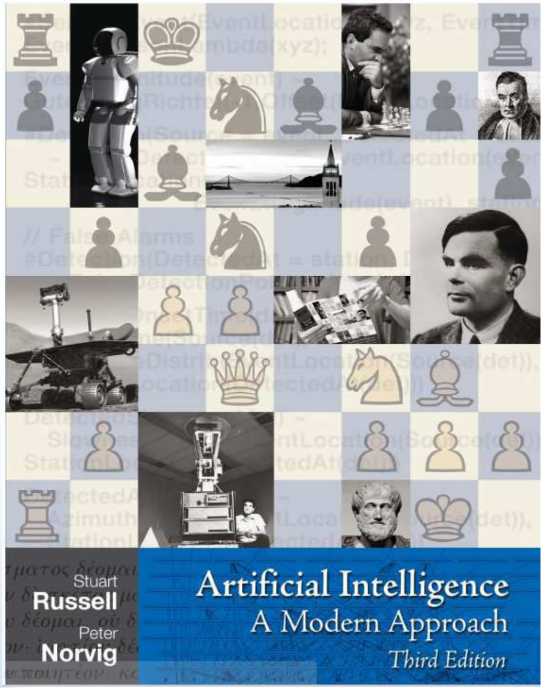
- Influence diagrams provide clear and coherent semantics for the value of making an observation
  - VPI = P(new observation) * MEU(new observation)
    – MEU (with current observations)
- Information is valuable if and only if it induces a change in action in at least one context, and with (significant) higher MEU.

# Intelligent Autonomous Agents and Cognitive Robotics

## Topic 8: Decision-Making under Uncertainty
## Complex Decisions

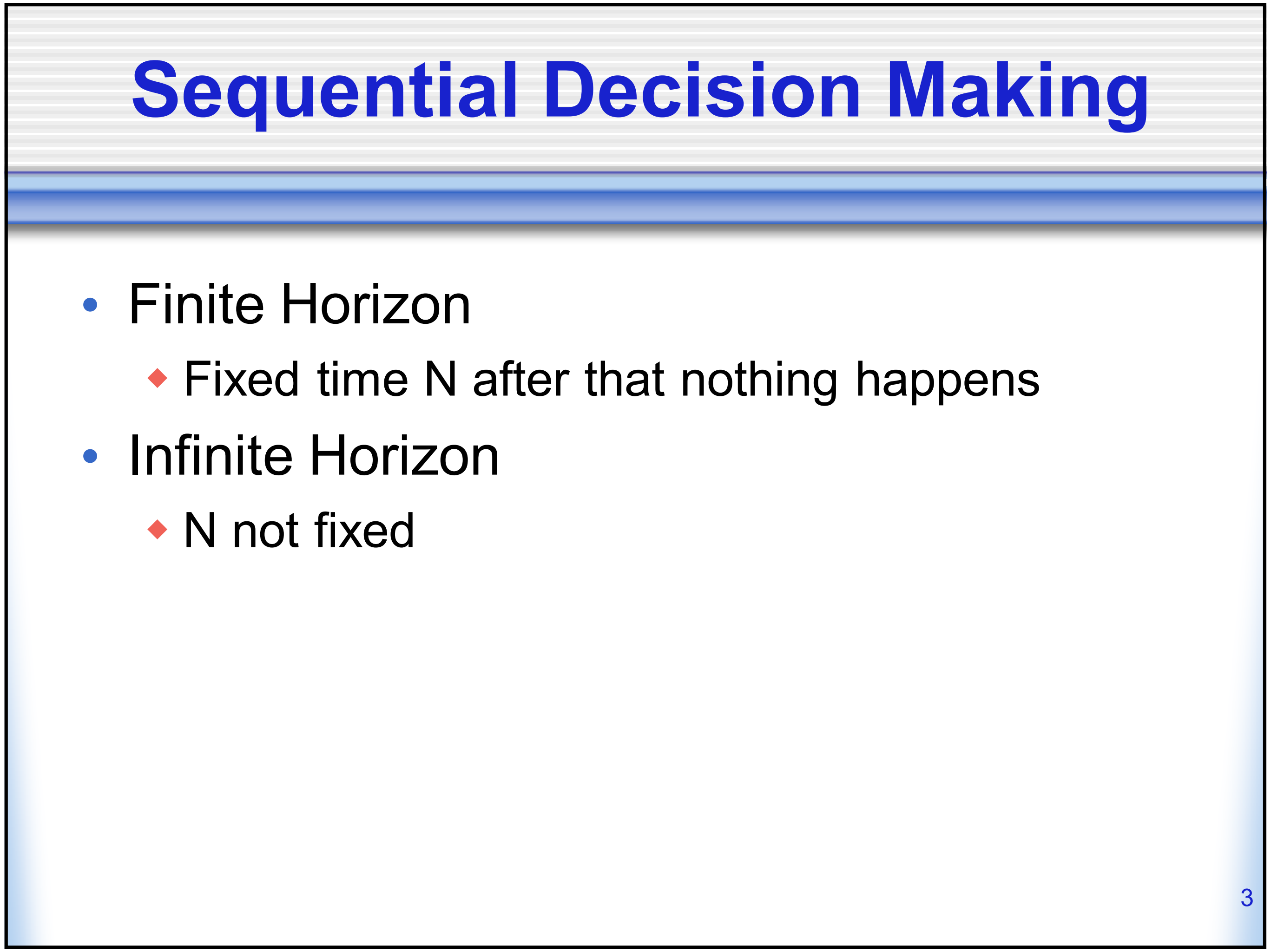
Ralf Möller, Rainer Marrone
Hamburg University of Technology

# Literature



- Chapter 17

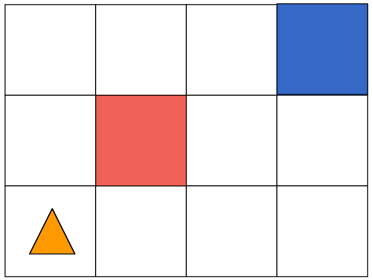
Material from Lise Getoor, Jean-Claude Latombe, Daphne Koller, and Stuart Russell



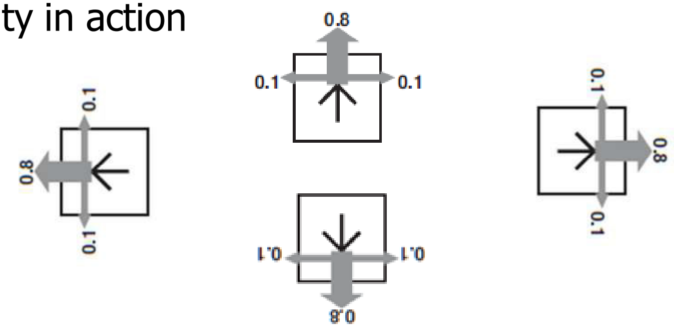
## Sequential Decision Making

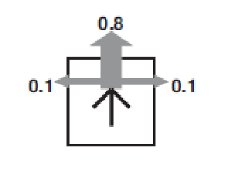
- Finite Horizon
  - ◆ Fixed time N after that nothing happens
- Infinite Horizon
  - ◆ N not fixed

# Simple Robot Navigation Problem



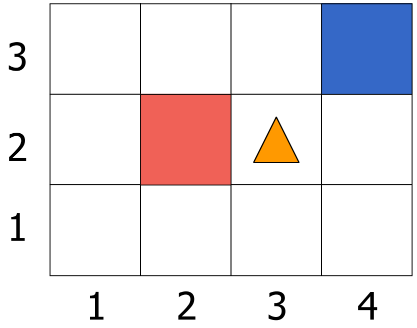
- In each state, the possible actions are U, D, R, L
  Uncertainty in action





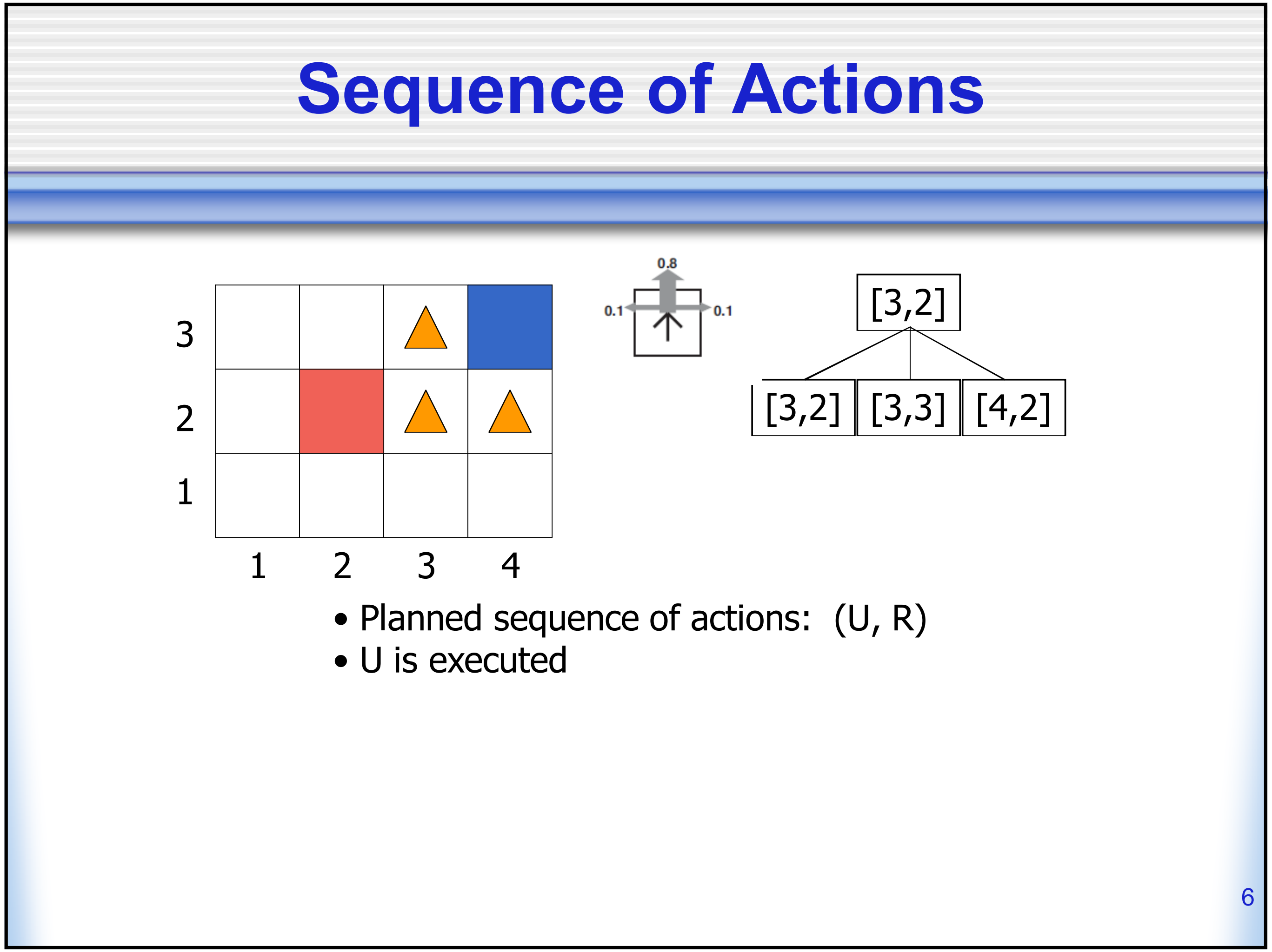
# Sequence of Actions

[3,2]

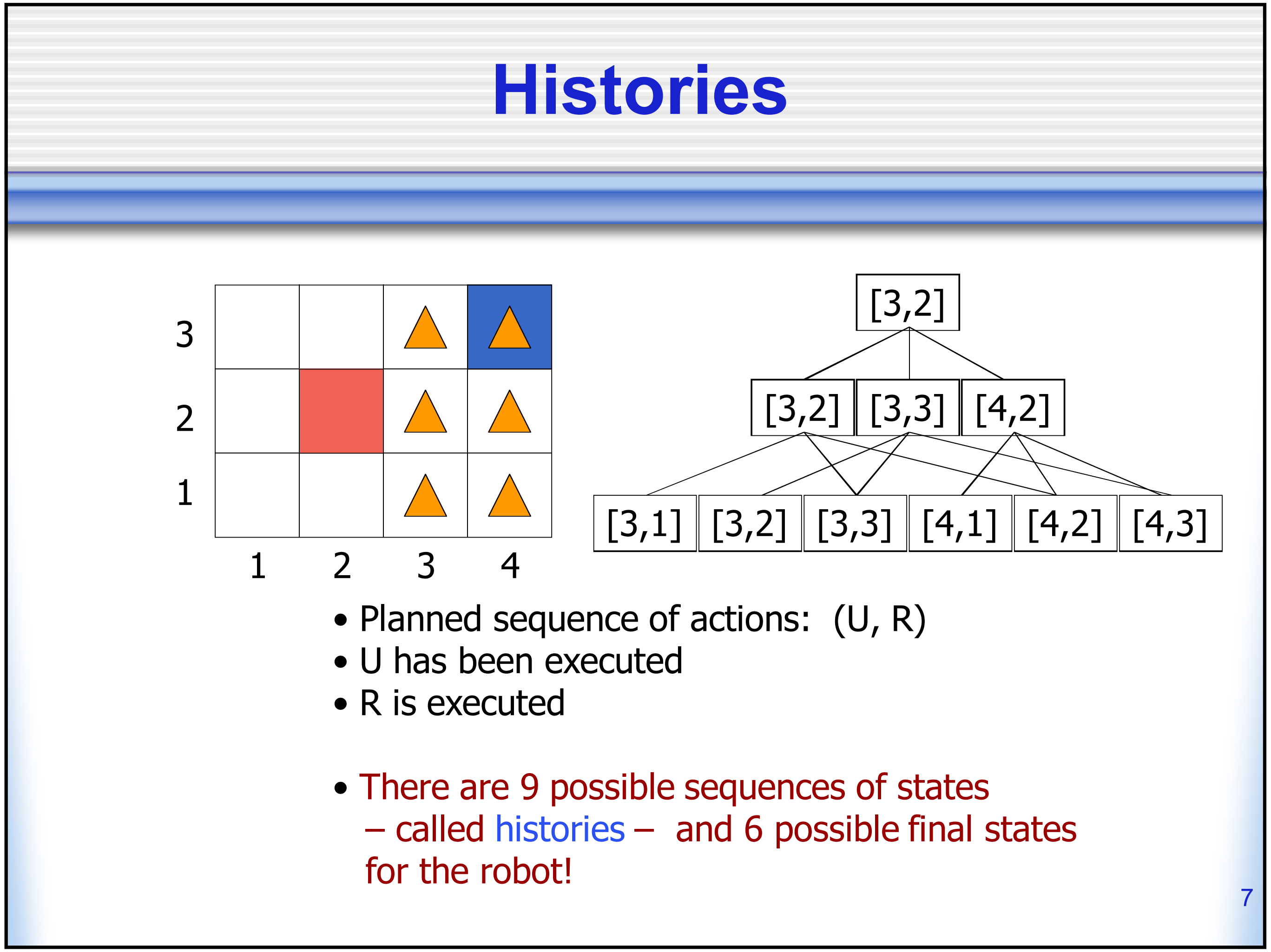


- Planned sequence of actions: (U, R)

# Sequence of Actions



- Planned sequence of actions: (U, R)
- U is executed



# Histories

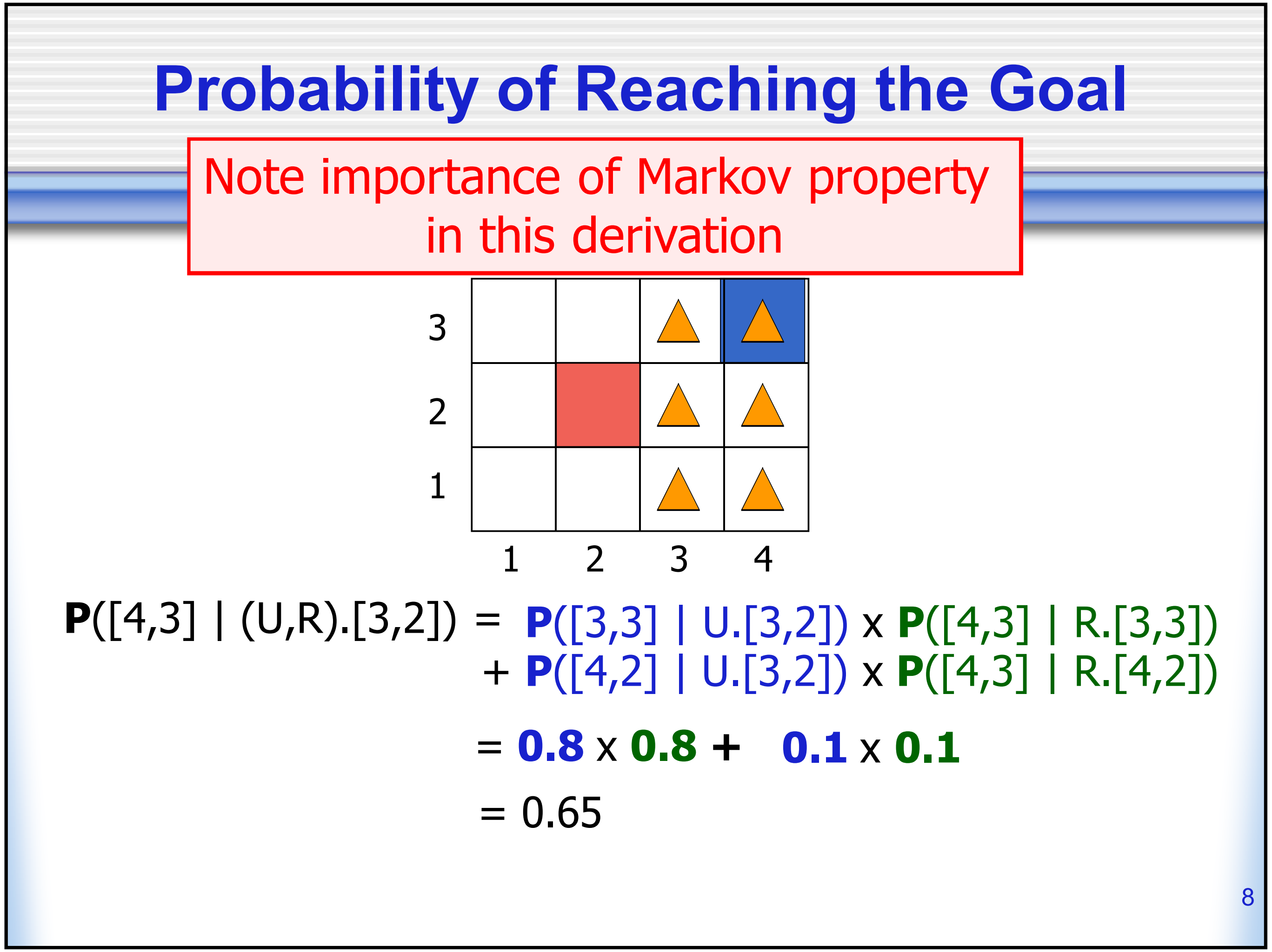
[3,2]

[3,2] [3,3] [4,2]

[3,1] [3,2] [3,3] [4,1] [4,2] [4,3]

- Planned sequence of actions: (U, R)
- U has been executed
- R is executed

- There are 9 possible sequences of states – called histories – and 6 possible final states for the robot!



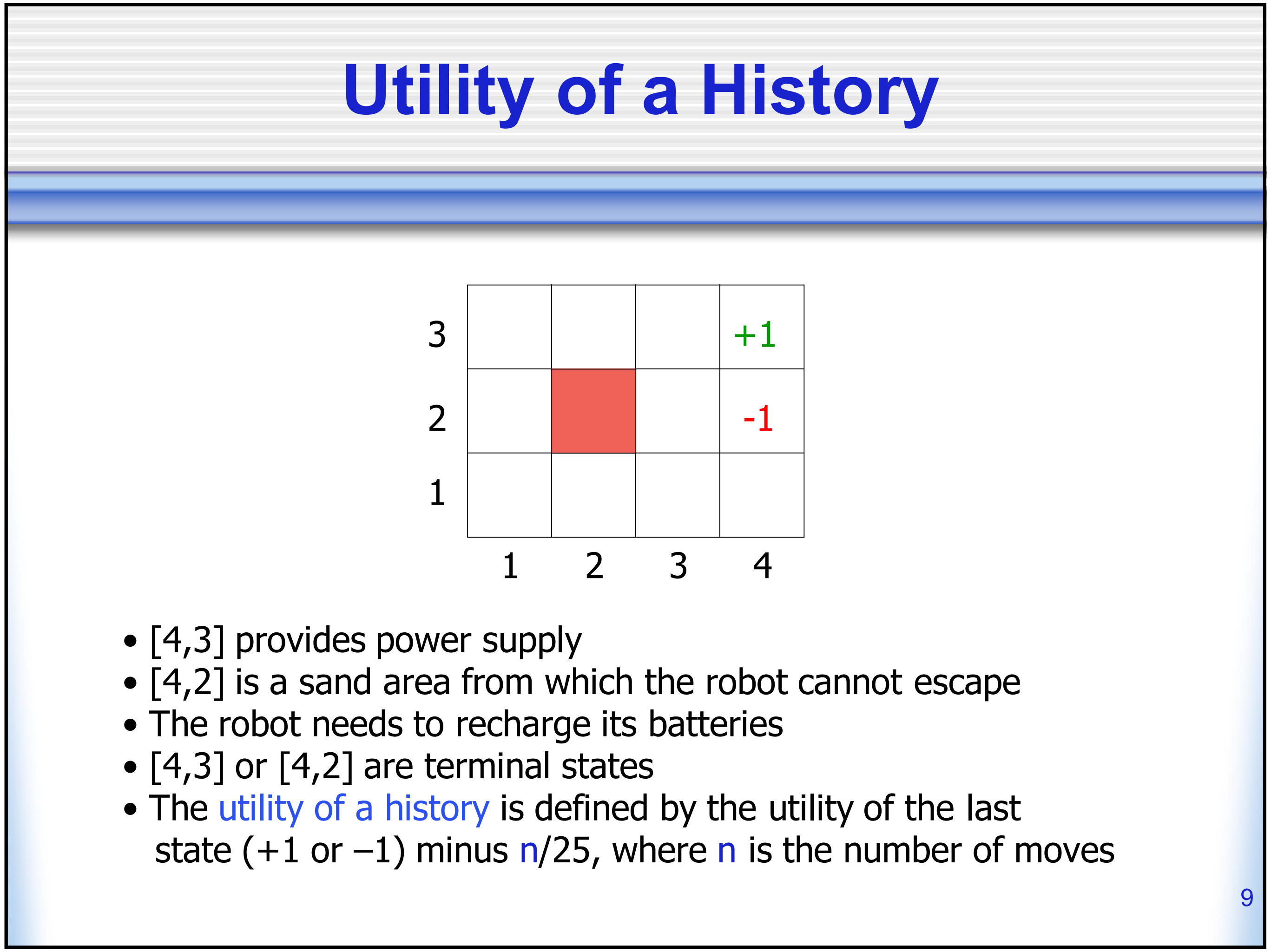
# Probability of Reaching the Goal

Note importance of Markov property in this derivation

$$\mathbf{P}([4,3] \mid (U,R).[3,2]) = \mathbf{P}([3,3] \mid U.[3,2]) \times \mathbf{P}([4,3] \mid R.[3,3]) + \mathbf{P}([4,2] \mid U.[3,2]) \times \mathbf{P}([4,3] \mid R.[4,2])$$

$$= \mathbf{0.8} \times \mathbf{0.8} + \mathbf{0.1} \times \mathbf{0.1}$$

$$= 0.65$$

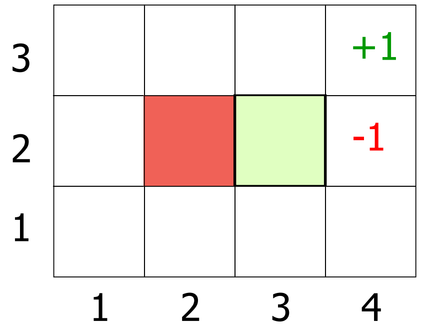


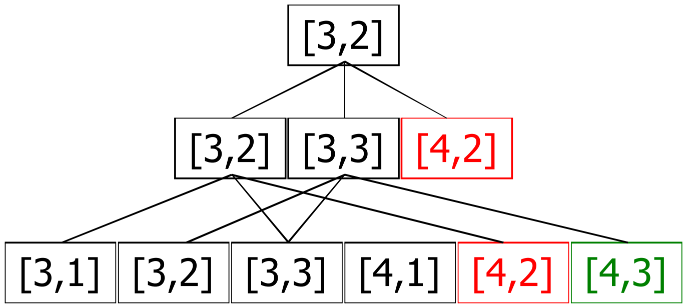
# Utility of a History

| | | | | |
|---|---|---|---|---|
| 3 | | | | +1 |
| 2 | | 🟥 | | -1 |
| 1 | | | | |
| | 1 | 2 | 3 | 4 |

- [4,3] provides power supply
- [4,2] is a sand area from which the robot cannot escape
- The robot needs to recharge its batteries
- [4,3] or [4,2] are terminal states
- The utility of a history is defined by the utility of the last state (+1 or –1) minus n/25, where n is the number of moves

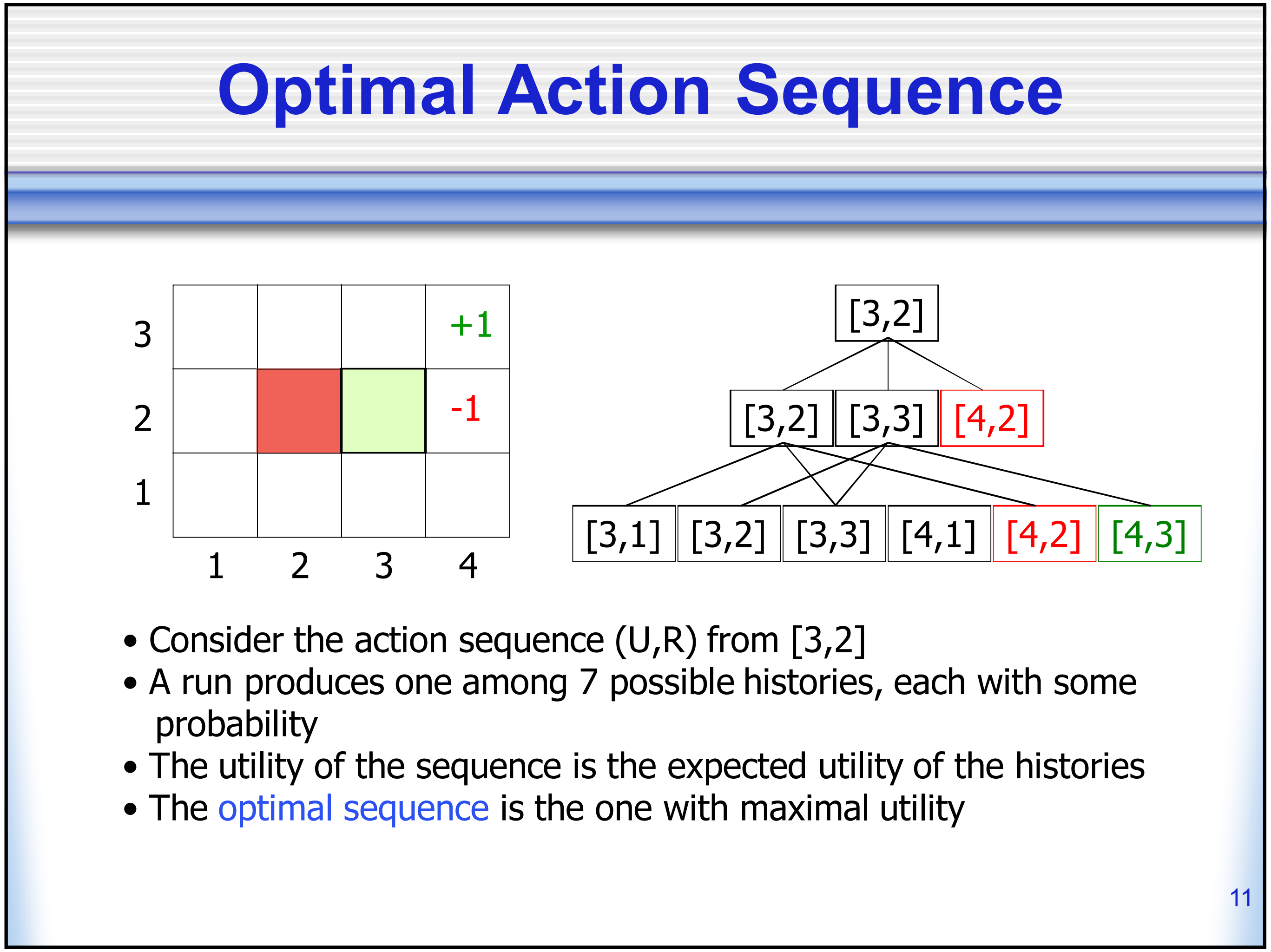
# Utility of an Action Sequence





- Consider the action sequence (U,R) from [3,2]
- A run produces one among 7 possible histories, each with some probability
- The utility of the sequence is the expected utility of the histories:

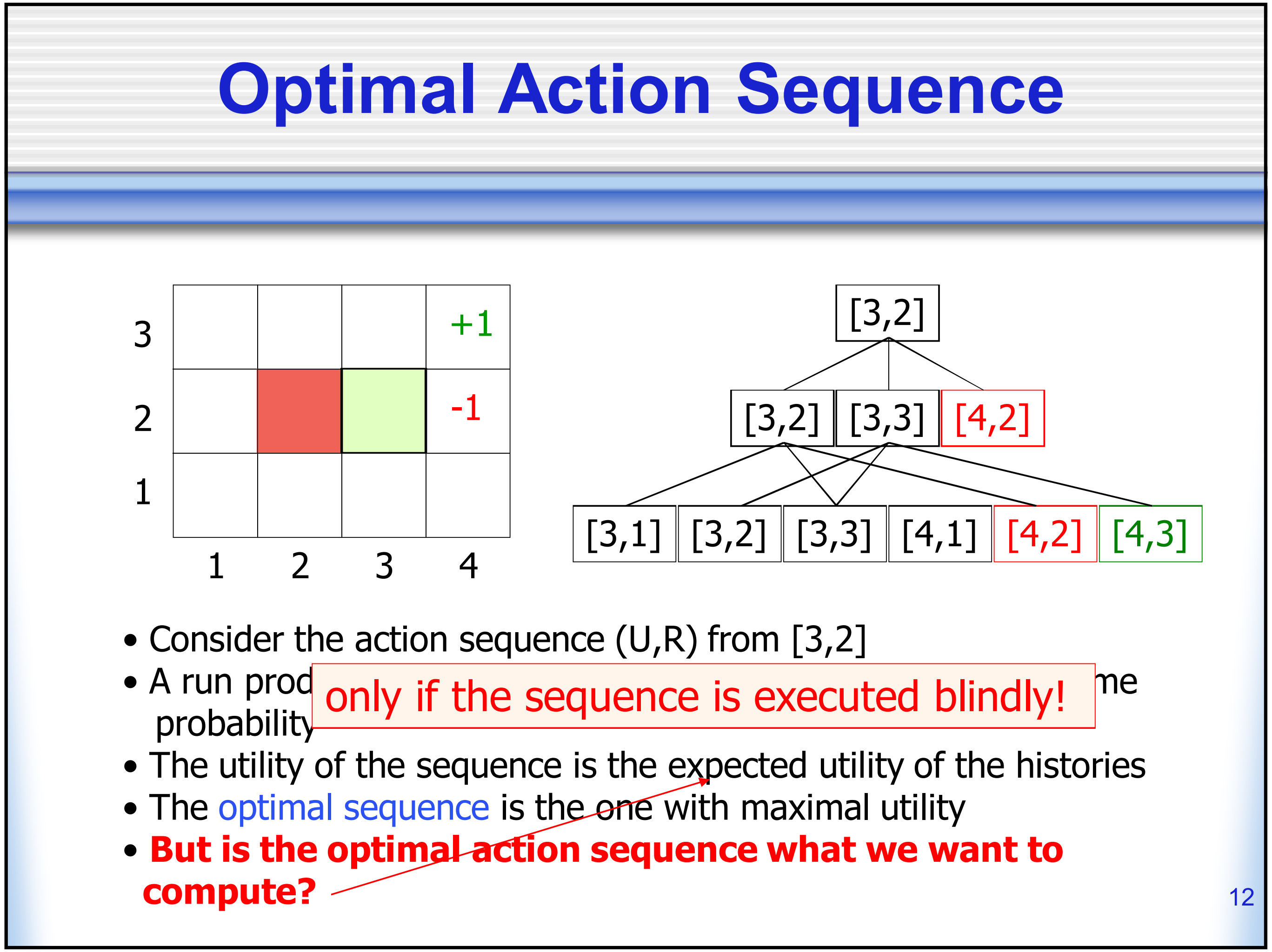
$$\mathcal{U} = \Sigma_h \mathcal{U}_h \, \mathbf{P}(h)$$



# Optimal Action Sequence

| | | | | |
|---|---|---|---|---|
| 3 | | | | +1 |
| 2 | | 🟥 | 🟩 | -1 |
| 1 | | | | |
| | 1 | 2 | 3 | 4 |

- Consider the action sequence (U,R) from [3,2]
- A run produces one among 7 possible histories, each with some probability
- The utility of the sequence is the expected utility of the histories
- The optimal sequence is the one with maximal utility



# Optimal Action Sequence

- Consider the action sequence (U,R) from [3,2]
- A run prod me probability
- The utility of the sequence is the expected utility of the histories
- The optimal sequence is the one with maximal utility
- **But is the optimal action sequence what we want to compute?**

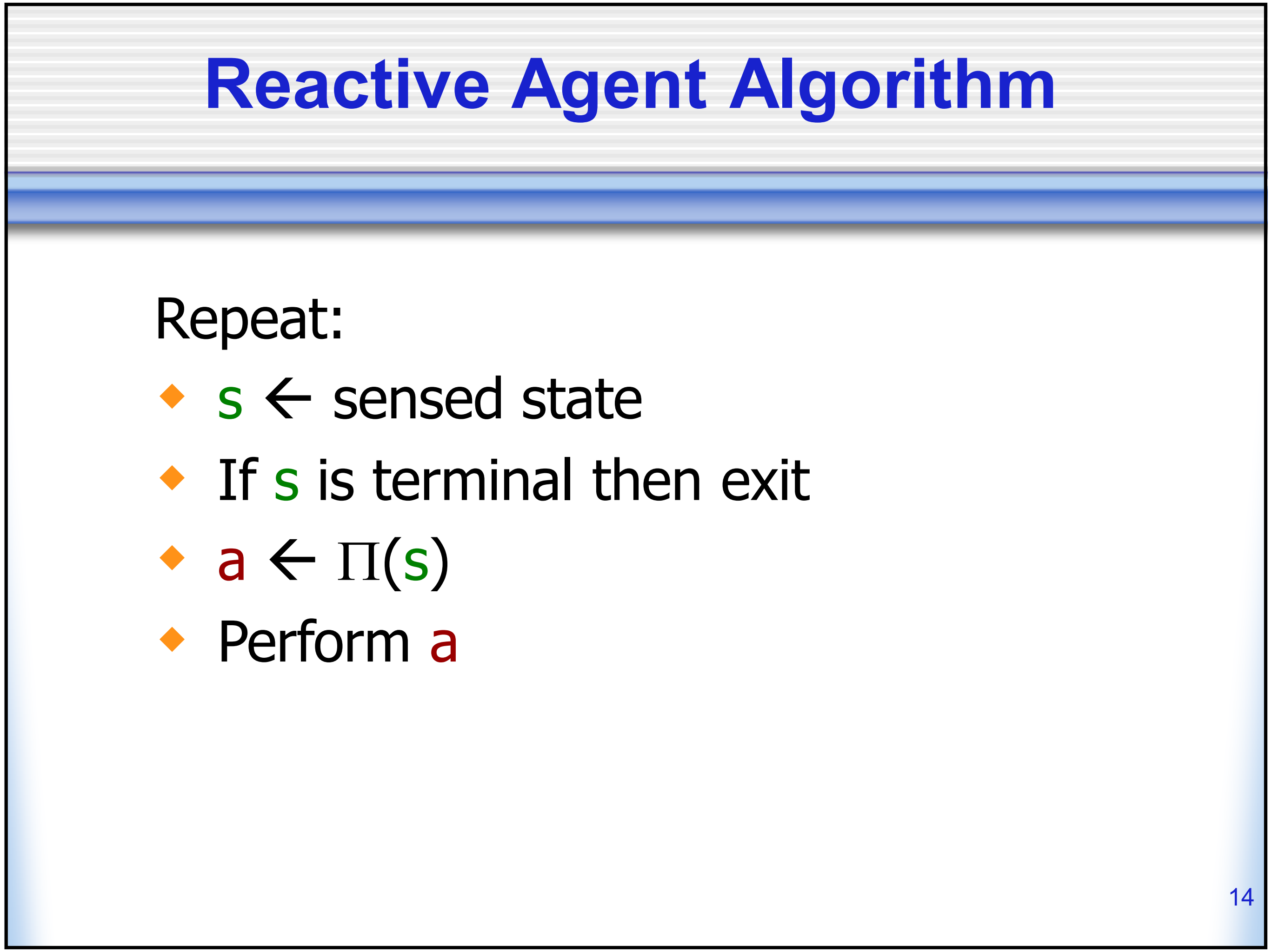
only if the sequence is executed blindly!

12

# Policy (Reactive/Closed-Loop Strategy)

| | | | | |
|---|---|---|---|---|
| 3 | → | → | → | +1 |
| 2 | ↑ | 🟥 | ↑ | -1 |
| 1 | ↑ | ← | ← | ← |
| | 1 | 2 | 3 | 4 |

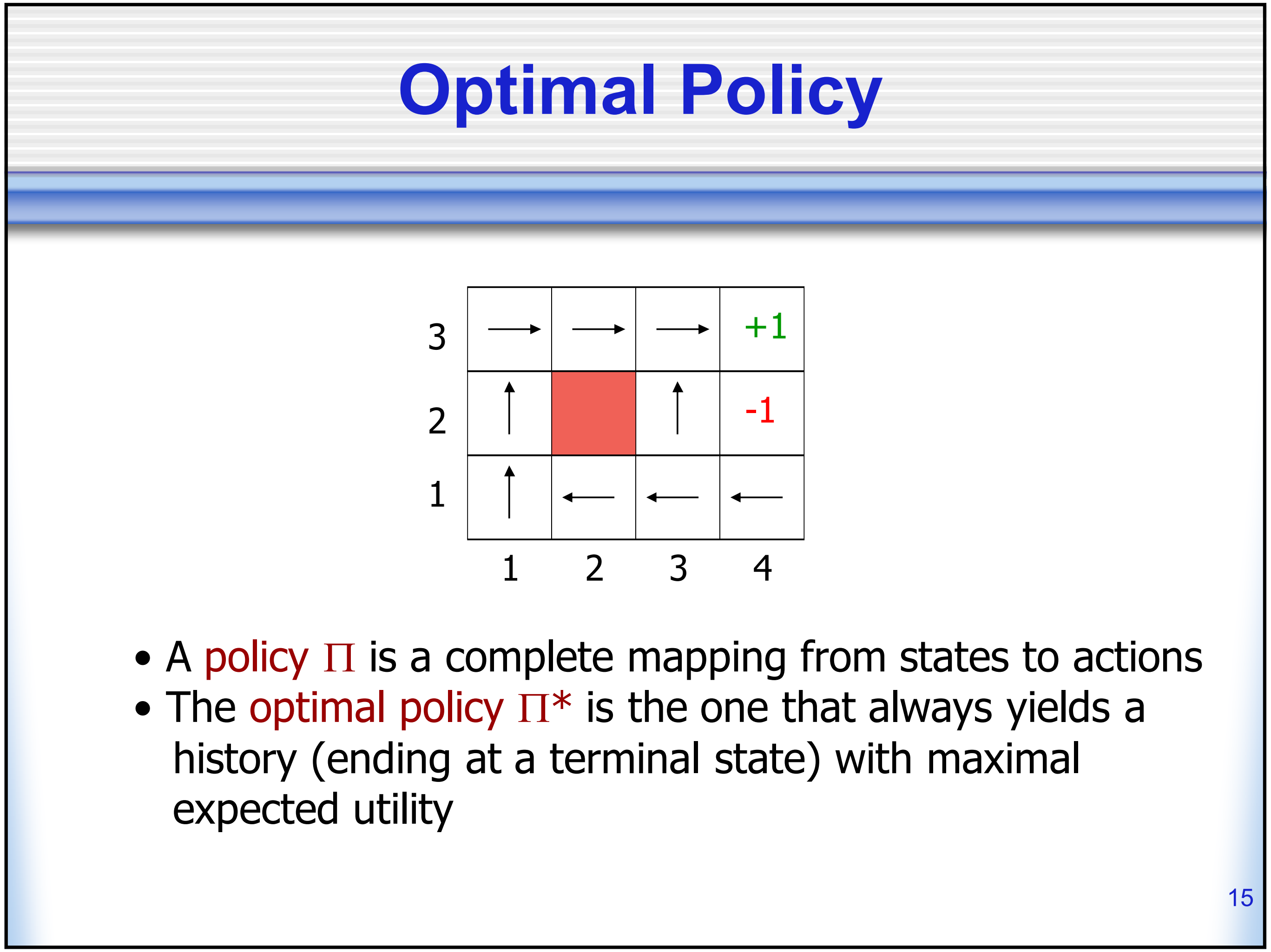
- A policy Π is a complete mapping from states to actions



# Reactive Agent Algorithm

Repeat:

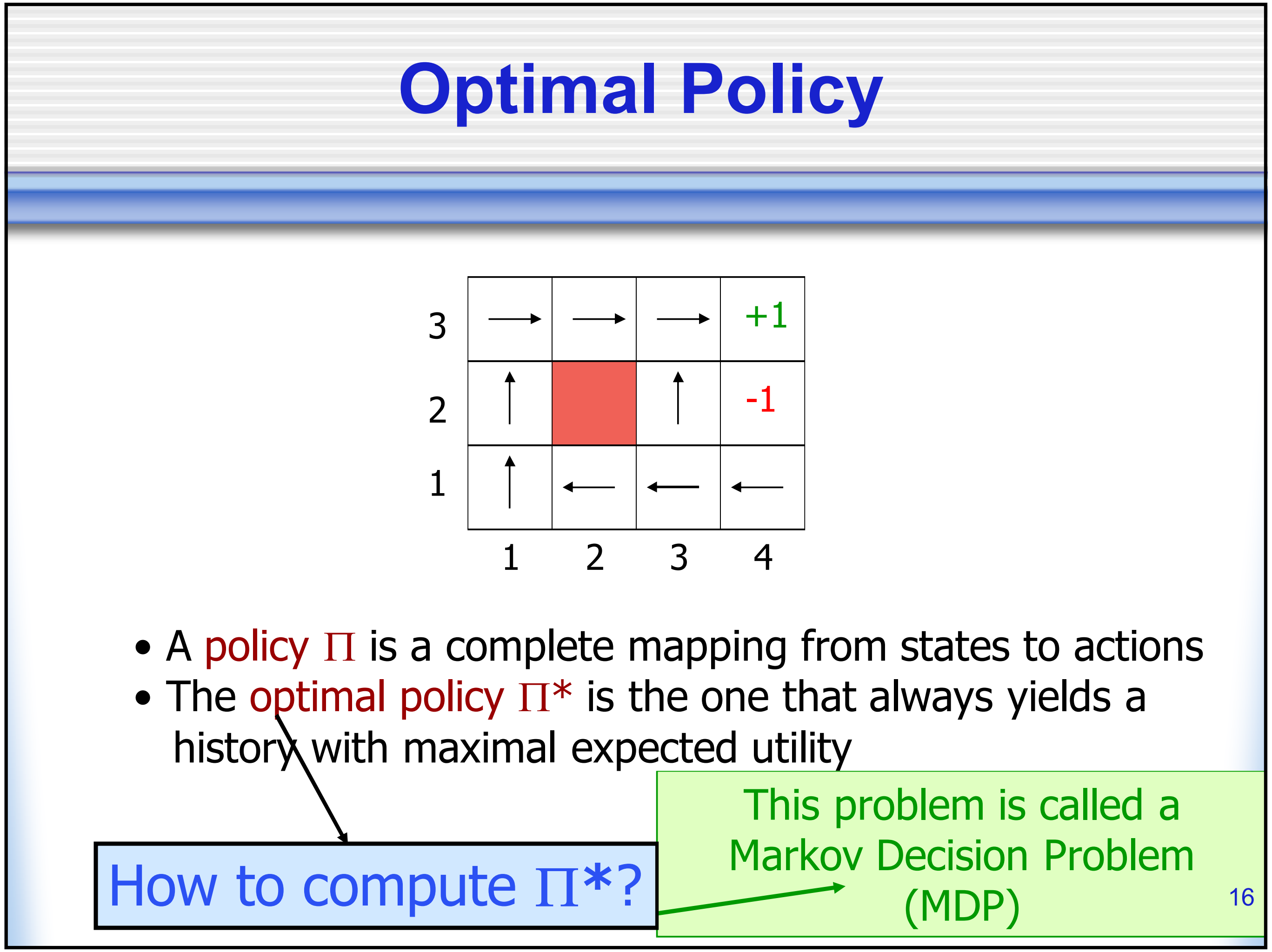
- s ← sensed state
- If s is terminal then exit
- a ← $\Pi$(s)
- Perform a



# Optimal Policy

|   |   |   |   |   |
|---|---|---|---|---|
| 3 | → | → | → | +1 |
| 2 | ↑ | 🟥 | ↑ | -1 |
| 1 | ↑ | ← | ← | ← |
|   | 1 | 2 | 3 | 4 |

- A policy Π is a complete mapping from states to actions
- The optimal policy Π* is the one that always yields a history (ending at a terminal state) with maximal expected utility



# Optimal Policy

|   |   |   |   |   |
|---|---|---|---|---|
| 3 | → | → | → | +1 |
| 2 | ↑ | 🟥 | ↑ | -1 |
| 1 | ↑ | ← | ← | ← |
|   | 1 | 2 | 3 | 4 |

- A policy Π is a complete mapping from states to actions
- The optimal policy Π* is the one that always yields a history with maximal expected utility

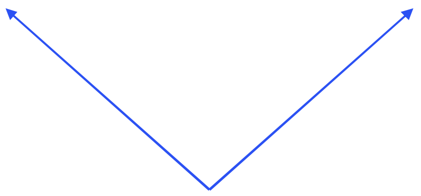
How to compute Π*?

This problem is called a Markov Decision Problem (MDP)

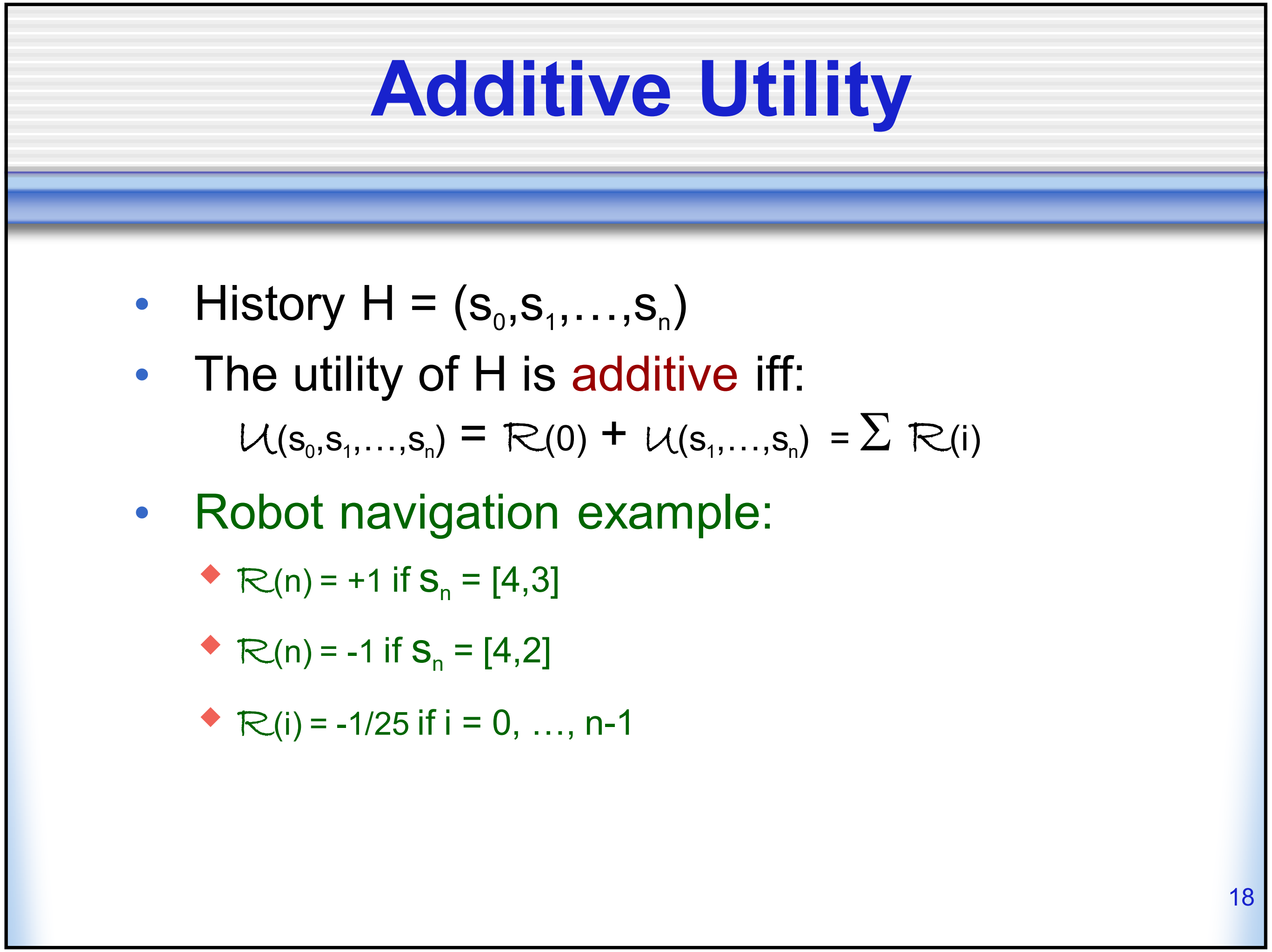
# Additive Utility: Stationarity

- History H = $(s_0,s_1,\ldots,s_n)$
- The utility of H is additive iff:

$$\mathcal{U}(s_0,s_1,\ldots,s_n) = \mathcal{R}(0) + \mathcal{U}(s_1,\ldots,s_n) = \Sigma\ \mathcal{R}(i)$$



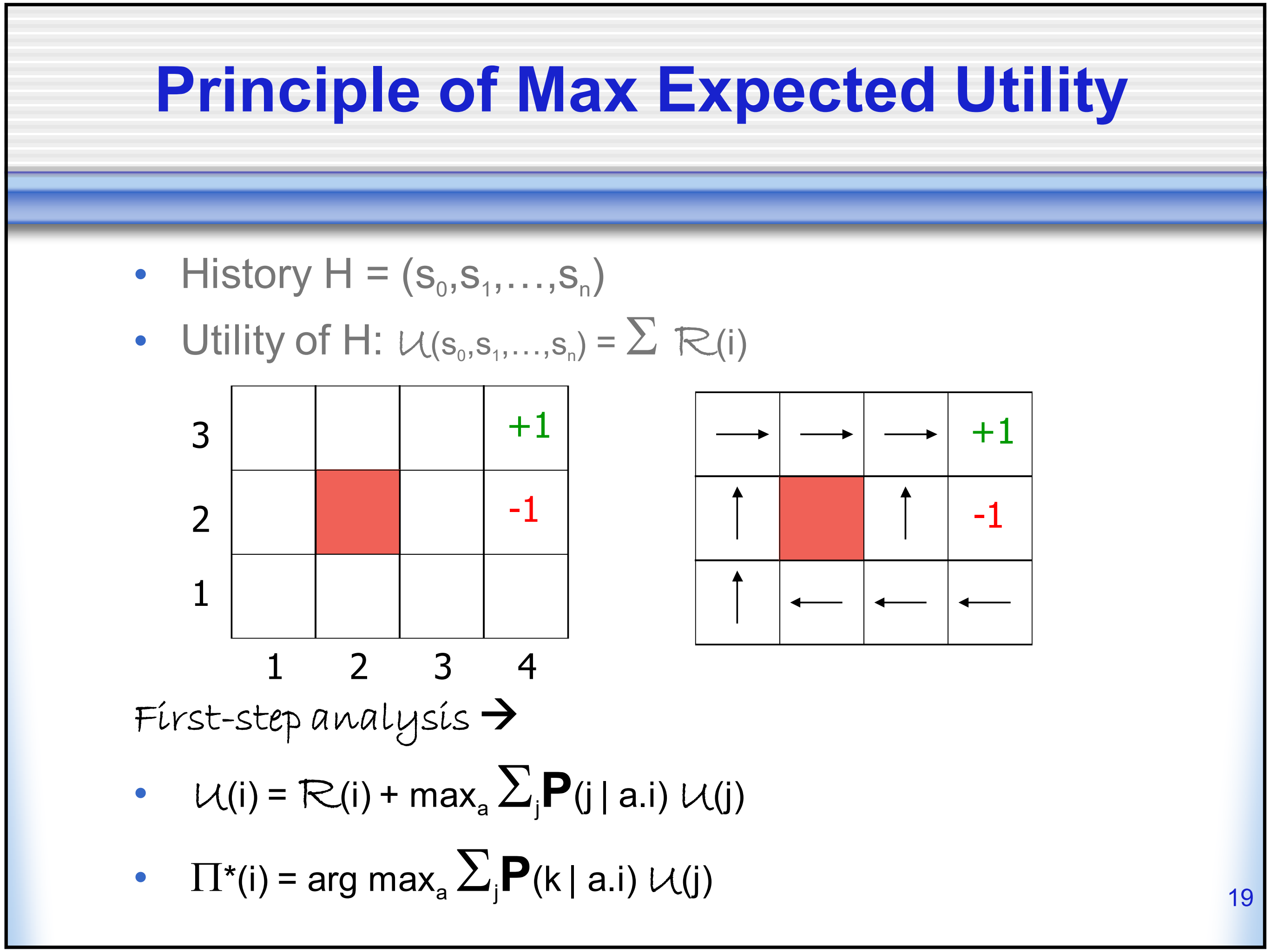
Reward



# Additive Utility

- History H = $(s_0,s_1,\ldots,s_n)$
- The utility of H is additive iff:

  $\mathcal{U}(s_0,s_1,\ldots,s_n) = \mathcal{R}(0) + \mathcal{U}(s_1,\ldots,s_n) = \Sigma\ \mathcal{R}(i)$
- Robot navigation example:
  - $\mathcal{R}(n) = +1$ if $s_n = [4,3]$
  - $\mathcal{R}(n) = -1$ if $s_n = [4,2]$
  - $\mathcal{R}(i) = -1/25$ if $i = 0, \ldots, n-1$



# Principle of Max Expected Utility

- History H = $(s_0,s_1,\ldots,s_n)$
- Utility of H: $\mathcal{U}(s_0,s_1,\ldots,s_n) = \Sigma\ \mathcal{R}(i)$

| | | | | |
|---|---|---|---|---|
| 3 | | | | +1 |
| 2 | | 🟥 | | -1 |
| 1 | | | | |
| | 1 | 2 | 3 | 4 |

| | | | |
|---|---|---|---|
| → | → | → | +1 |
| ↑ | 🟥 | ↑ | -1 |
| ↑ | ← | ← | ← |

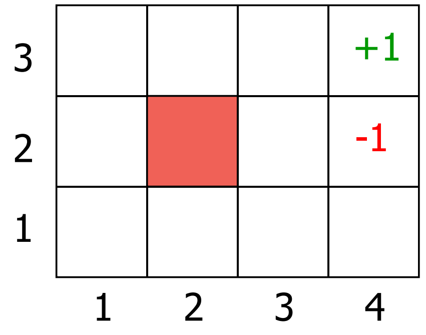
*First-step analysis →*

- $\mathcal{U}(i) = \mathcal{R}(i) + \max_a \Sigma_j \mathbf{P}(j \mid a.i)\ \mathcal{U}(j)$
- $\Pi^*(i) = \arg\max_a \Sigma_j \mathbf{P}(k \mid a.i)\ \mathcal{U}(j)$

# Value Iteration

- Initialize the utility of each non-terminal state $s_i$ to $\mathcal{U}_0(i) = 0$
- For t = 0, 1, 2, …, do:

$$\mathcal{U}_{t+1}(i) \leftarrow \mathcal{R}(i) + \max_a \sum_k \mathbf{P}(k \mid a.i)\, \mathcal{U}_t(k) \qquad \text{(Bellmann equation)}$$



**Initialization**

|   |   |   |   |   |
|---|---|---|---|---|
| 3 | 0 | 0 | 0 | +1 |
| 2 | 0 | ▨ | 0 | -1 |
| 1 | 0 | 0 | 0 | 0 |
|   | 1 | 2 | 3 | 4 |

**Iteration 1**

|   |   |   |   |   |
|---|---|---|---|---|
| 3 | 0 | 0 | 0 | +1 |
| 2 | 0 | ▨ | 0 | -1 |
| 1 | -0.04 | 0 | 0 | 0 |
|   | 1 | 2 | 3 | 4 |

$$U^{(1)}(1,1) = -0.04 + 1 * \max \begin{bmatrix} 0.8U^{(0)}(1,2) + 0.1U^{(0)}(2,1) + 0.1U^{(0)}(1,1) & UP \\ 0.9U^{(0)}(1,1) + 0.1U^{(0)}(1,2) & LEFT \\ 0.9U^{(0)}(1,1) + 0.1U^{(0)}(2,1) & DOWN \\ 0.8U^{(0)}(2,1) + 0.1U^{(0)}(1,2) + 0.1U^{(0)}(1,1) & RIGHT \end{bmatrix}$$

$$U^{(1)}(1,1) = -0.04 + \max \begin{bmatrix} 0 & UP \\ 0 & LEFT \\ 0 & DOWN \\ 0 & RIGHT \end{bmatrix}$$

**Initialization**

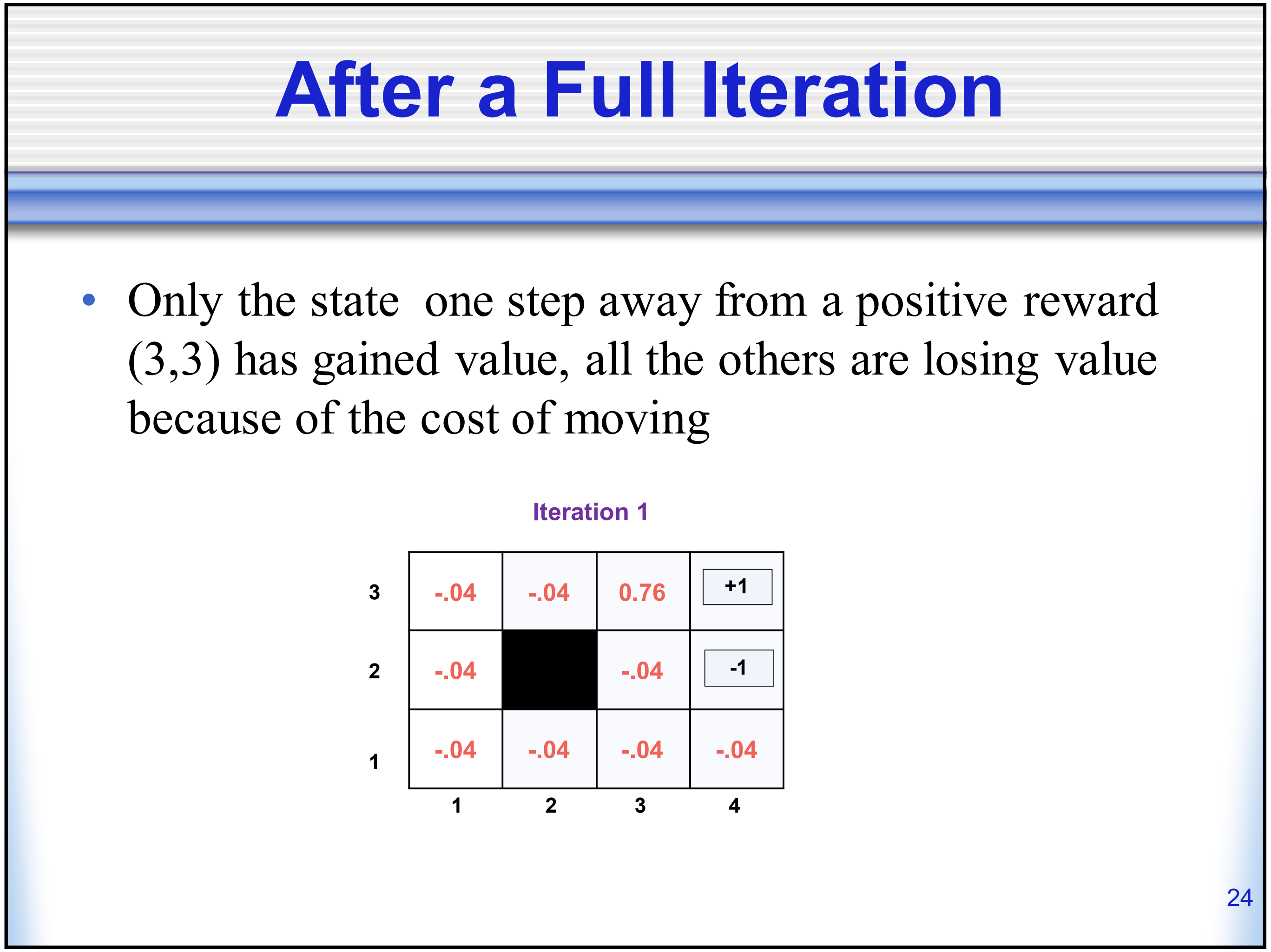
|   |   |   |   |   |
|---|---|---|---|---|
| 3 | 0 | 0 | 0 | +1 |
| 2 | 0 | ▨ | 0 | -1 |
| 1 | 0 | 0 | 0 | 0 |
|   | 1 | 2 | 3 | 4 |

**Iteration 1**

|   |   |   |   |   |
|---|---|---|---|---|
| 3 | 0 | 0 | 0.76 | +1 |
| 2 | 0 | ▨ | 0 | -1 |
| 1 | -0.04 | 0 | 0 | 0 |
|   | 1 | 2 | 3 | 4 |

$$U^{(1)}(3,3) = -0.04 + 1 * \max \begin{bmatrix} 0.8U^{(0)}(3,3) + 0.1U^{(0)}(2,3) + 0.1U^{(0)}(4,3) & UP \\ 0.8U^{(0)}(2,3) + 0.1U^{(0)}(3,3) + 0.1U^{(0)}(3,2) & LEFT \\ 0.8U^{(0)}(3,2) + 0.1U^{(0)}(2,3) + 0.1U^{(0)}(4,3) & DOWN \\ 0.8U^{(0)}(4,3) + 0.1U^{(0)}(3,3) + 0.1U^{(0)}(3,2) & RIGHT \end{bmatrix}$$

$$U^{(1)}(3,3) = -0.04 + \max \begin{bmatrix} 0.1 & UP \\ 0 & LEFT \\ 0.1 & DOWN \\ 0.8 & RIGHT \end{bmatrix}$$

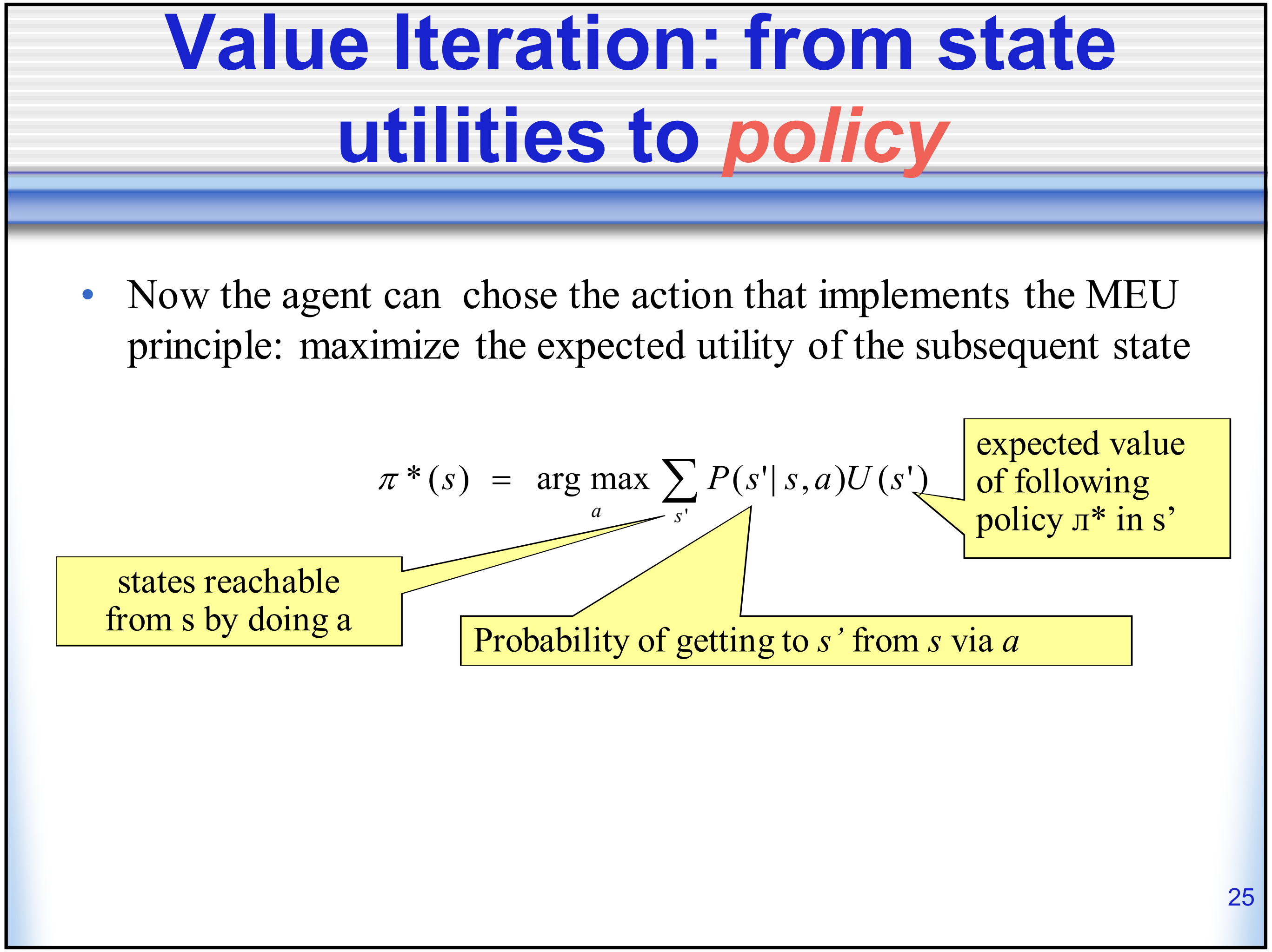


# After a Full Iteration

- Only the state one step away from a positive reward (3,3) has gained value, all the others are losing value because of the cost of moving

Iteration 1

|   |   |   |   |   |
|---|---|---|---|---|
| 3 | -.04 | -.04 | 0.76 | +1 |
| 2 | -.04 | ■ | -.04 | -1 |
| 1 | -.04 | -.04 | -.04 | -.04 |
|   | 1 | 2 | 3 | 4 |



# Value Iteration: from state utilities to *policy*

- Now the agent can chose the action that implements the MEU principle: maximize the expected utility of the subsequent state

$$\pi^*(s) = \arg\max_{a} \sum_{s'} P(s'|s,a)U(s')$$

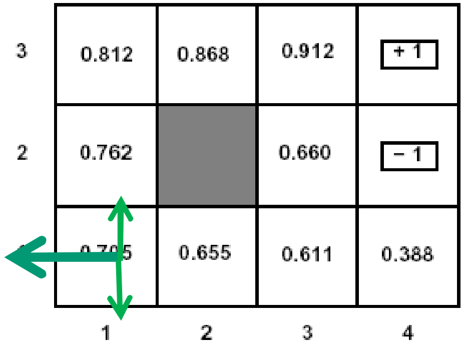
states reachable from s by doing a

Probability of getting to *s'* from *s* via *a*

expected value of following policy л* in s’

# Example

$$\pi^*(s) \;=\; \arg\max_{a} \sum_{s'} P(s'|\,s,a)U(s')$$



| | | | | |
|---|---|---|---|---|
| 3 | 0.812 | 0.868 | 0.912 | + 1 |
| 2 | 0.762 | ▨ | 0.660 | – 1 |
| 1 | 0.705 | 0.655 | 0.611 | 0.388 |
| | 1 | 2 | 3 | 4 |

- ➢ To find the best action in (1,1)

$$\pi^*(1,1) = \arg\max \left[\begin{array}{ll} 0{,}7456 & UP \\ \boxed{0{,}7107 \qquad\qquad LEFT} & \\ 0.9U(1{,}1)+0.1U(2{,}1) & DOWN \\ 0.8U(2{,}1)+0.1U(1{,}2)+0.1U(1{,}1) & RIGHT \end{array}\right]$$

- ➢ We have to do this for all fields!!!!

# Example

$$\pi^*(s) \;=\; \arg\max_{a} \sum_{s'} P(s' \mid s,a)\,U(s')$$

|   |   |   |   |   |
|---|---|---|---|---|
| 3 | 0.812 | 0.868 | 0.912 | +1 |
| 2 | 0.762 | ▨ | 0.660 | −1 |
| 1 | 0.705 | 0.655 | 0.611 | 0.388 |
|   | 1 | 2 | 3 | 4 |

- To find the best action in (1,1)

$$\pi^*(1{,}1) = \arg\max \begin{bmatrix} 0{,}7456 & \boxed{UP} \\ 0{,}7107 & LEFT \\ 0{,}7 & DOWN \\ 0.6707 & RIGHT \end{bmatrix}$$

- give *Up* as best action

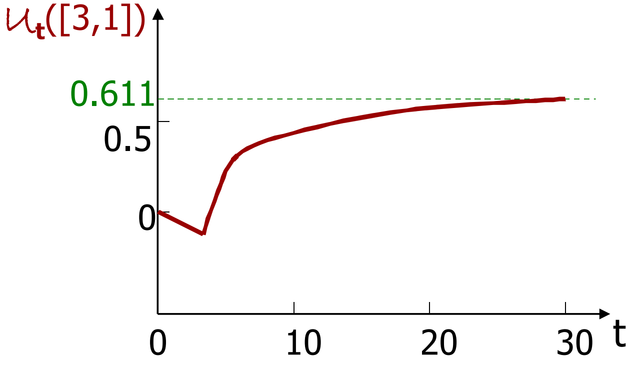
27

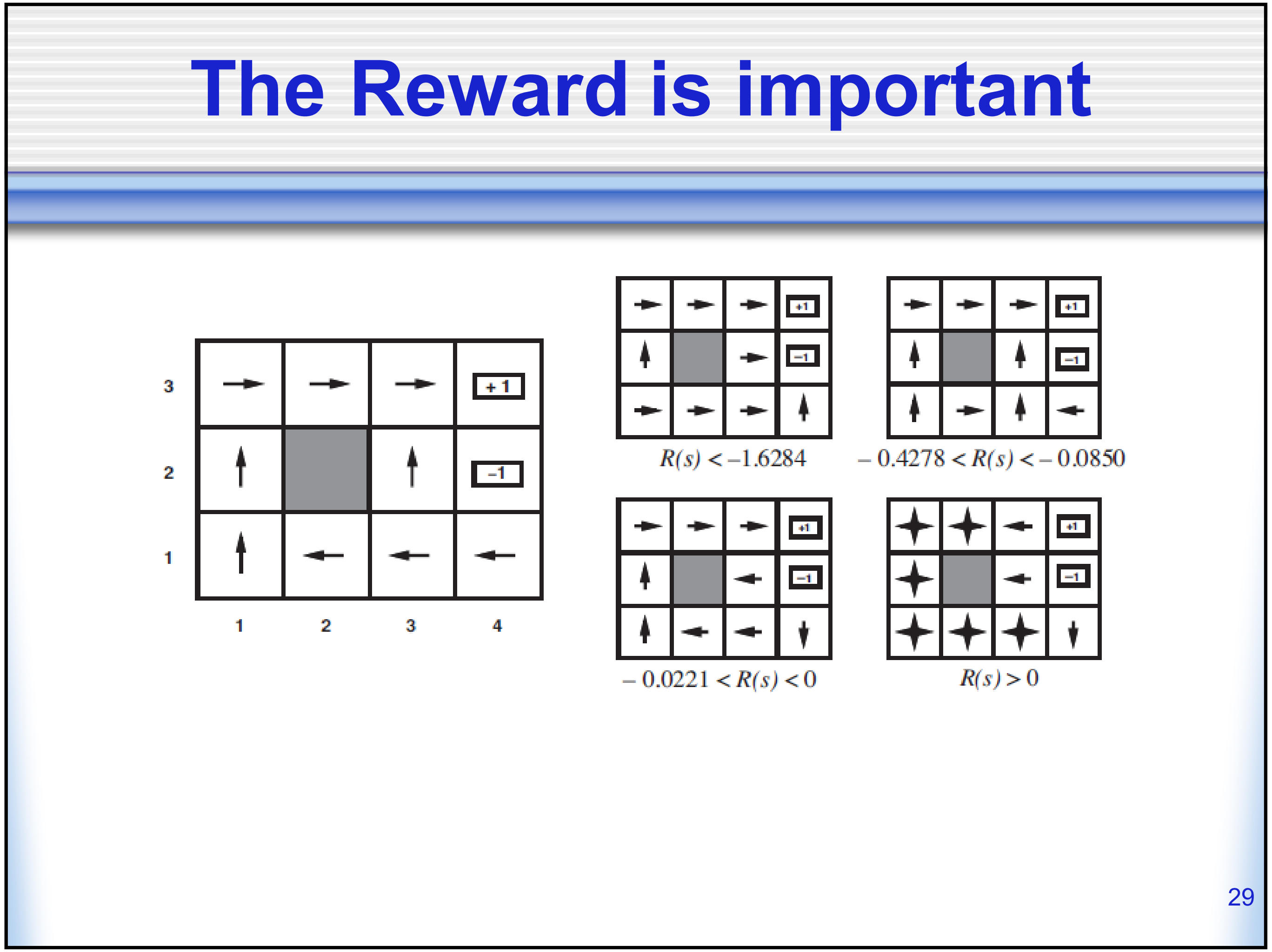
# Value Iteration: the result

- Initialize the utility of each non-terminal state $s_i$ to
  $\mathcal{U}_0(i) = 0$
- For t = 0, 1, 2, …, do:

$$\mathcal{U}_{t+1}(i) \leftarrow \mathcal{R}(i) + \max_a \sum_k \mathbf{P}(k \mid a.i)\, \mathcal{U}_t(k)$$

| | | | | |
|---|---|---|---|---|
| 3 | 0.812 → | 0.868 → | 0.918 → | +1 |
| 2 | 0.762 ↑ | 🟥 | 0.660 ↑ | -1 |
| 1 | 0.705 ↑ | 0.655 ← | 0.611 ← | 0.388 ← |
| | 1 | 2 | 3 | 4 |





# The Reward is important

$R(s) < -1.6284$

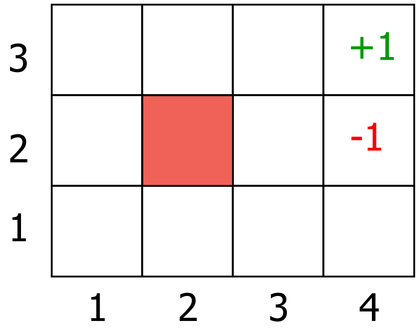
$-0.4278 < R(s) < -0.0850$

$-0.0221 < R(s) < 0$

$R(s) > 0$

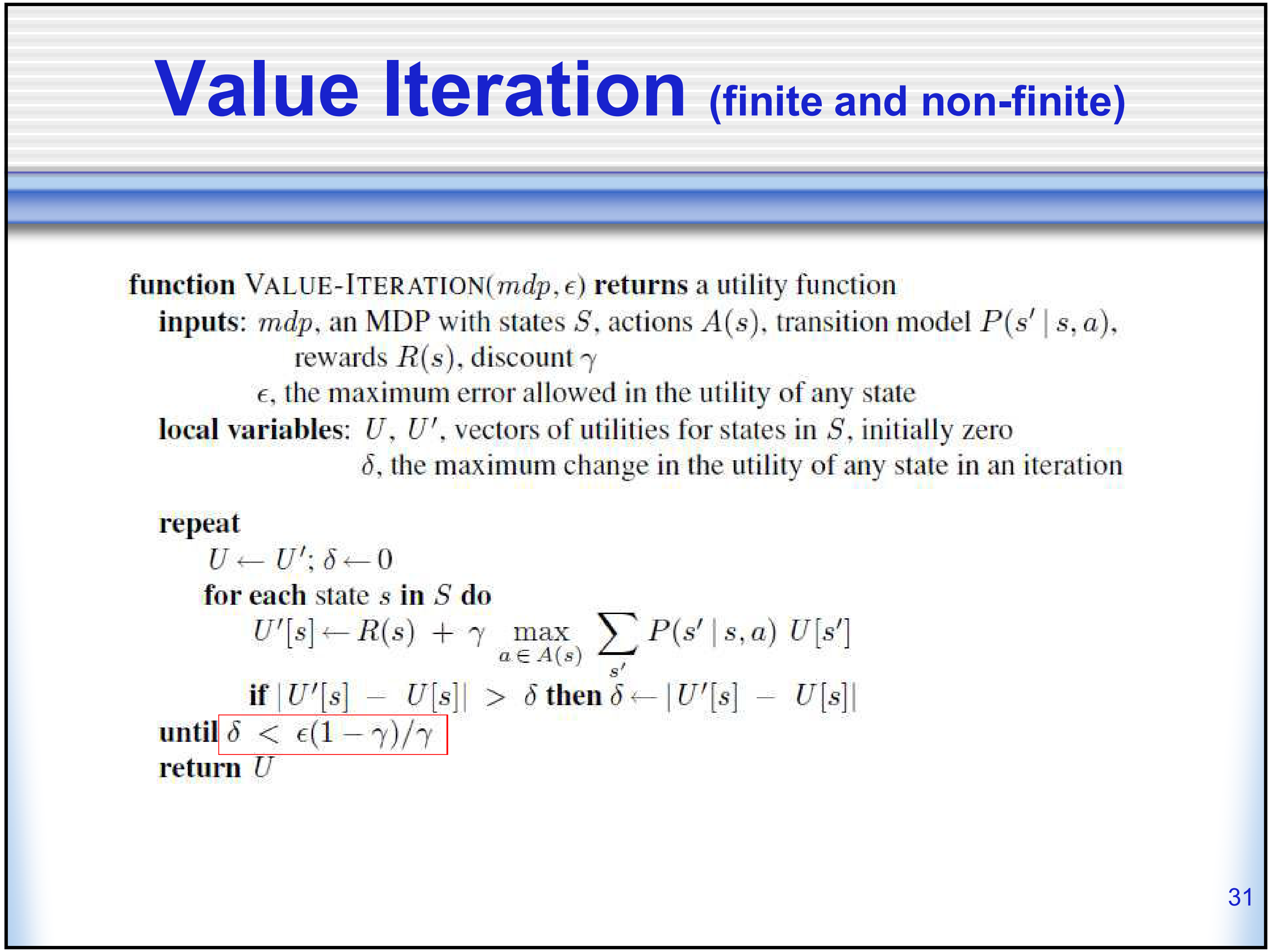
# Infinite Horizon

In many problems, e.g., the robot navigation example, histories are potentially unbounded and the same state can be reached many times



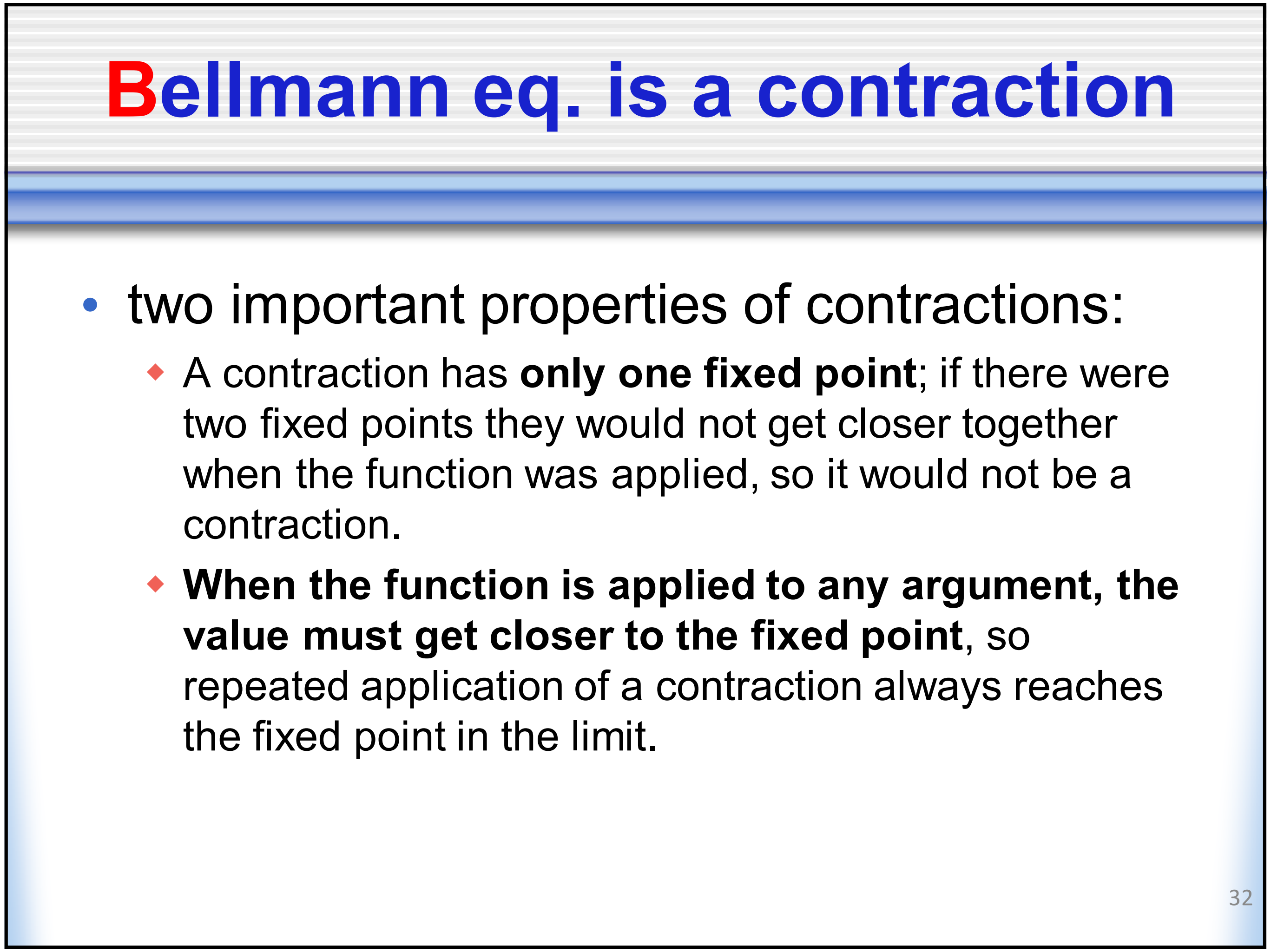
$$\mathcal{U}(i) = \mathcal{R}(i) + \gamma \max_a \sum_j \mathbf{P}(j \mid a.i)\ \mathcal{U}(j)$$

One trick:
Use discounting to make infinite Horizon problem mathematically tractable



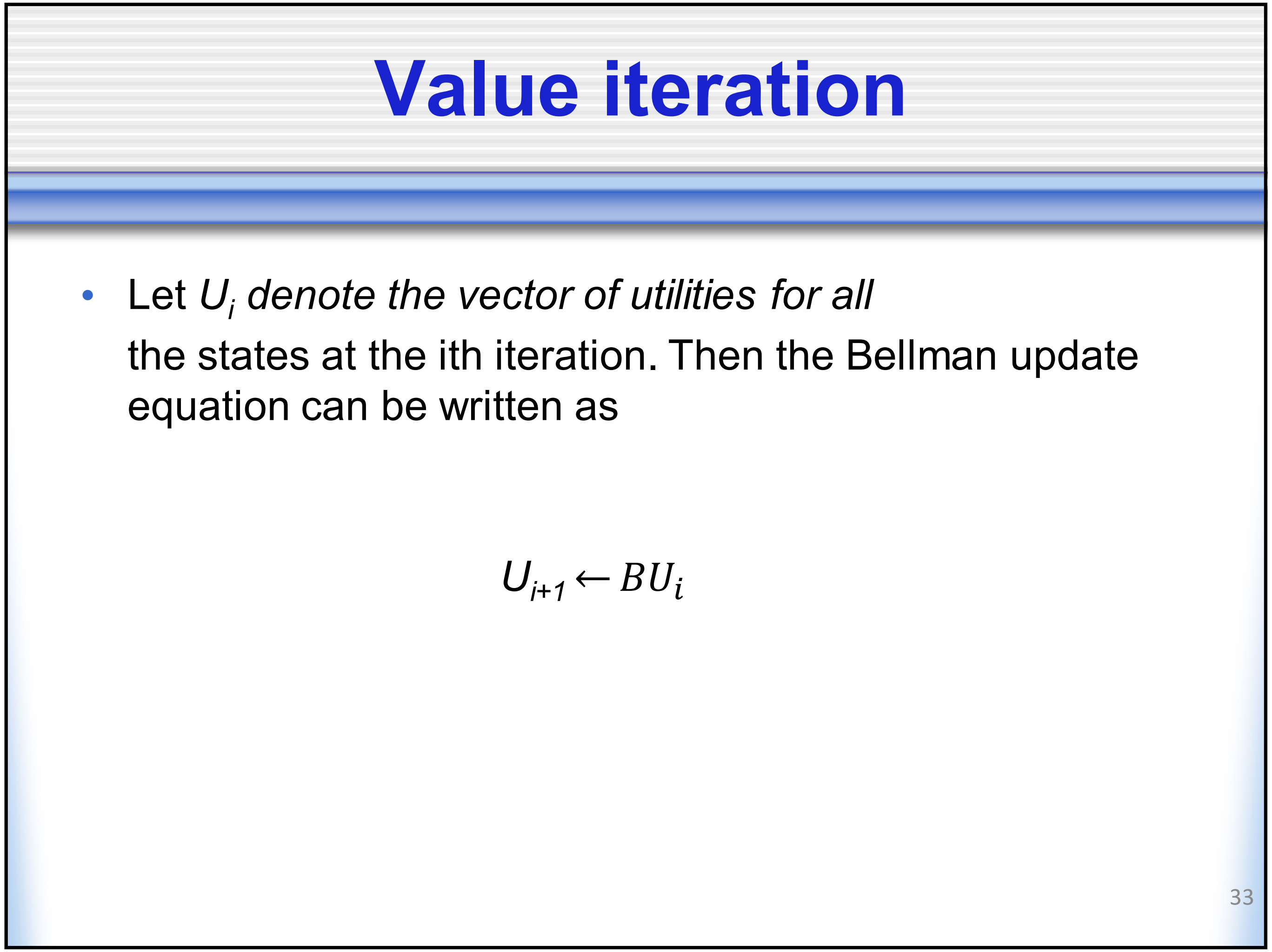
# Value Iteration (finite and non-finite)

```
function VALUE-ITERATION($mdp, \epsilon$) returns a utility function
  inputs: $mdp$, an MDP with states $S$, actions $A(s)$, transition model $P(s' \mid s, a)$,
              rewards $R(s)$, discount $\gamma$
          $\epsilon$, the maximum error allowed in the utility of any state
  local variables: $U$, $U'$, vectors of utilities for states in $S$, initially zero
                   $\delta$, the maximum change in the utility of any state in an iteration

  repeat
      $U \leftarrow U'$; $\delta \leftarrow 0$
      for each state $s$ in $S$ do
          $U'[s] \leftarrow R(s) + \gamma \max_{a \in A(s)} \sum_{s'} P(s' \mid s, a)\, U[s']$
          if $|U'[s] - U[s]| > \delta$ then $\delta \leftarrow |U'[s] - U[s]|$
  until $\boxed{\delta < \epsilon(1-\gamma)/\gamma}$
  return $U$
```



# Bellmann eq. is a contraction

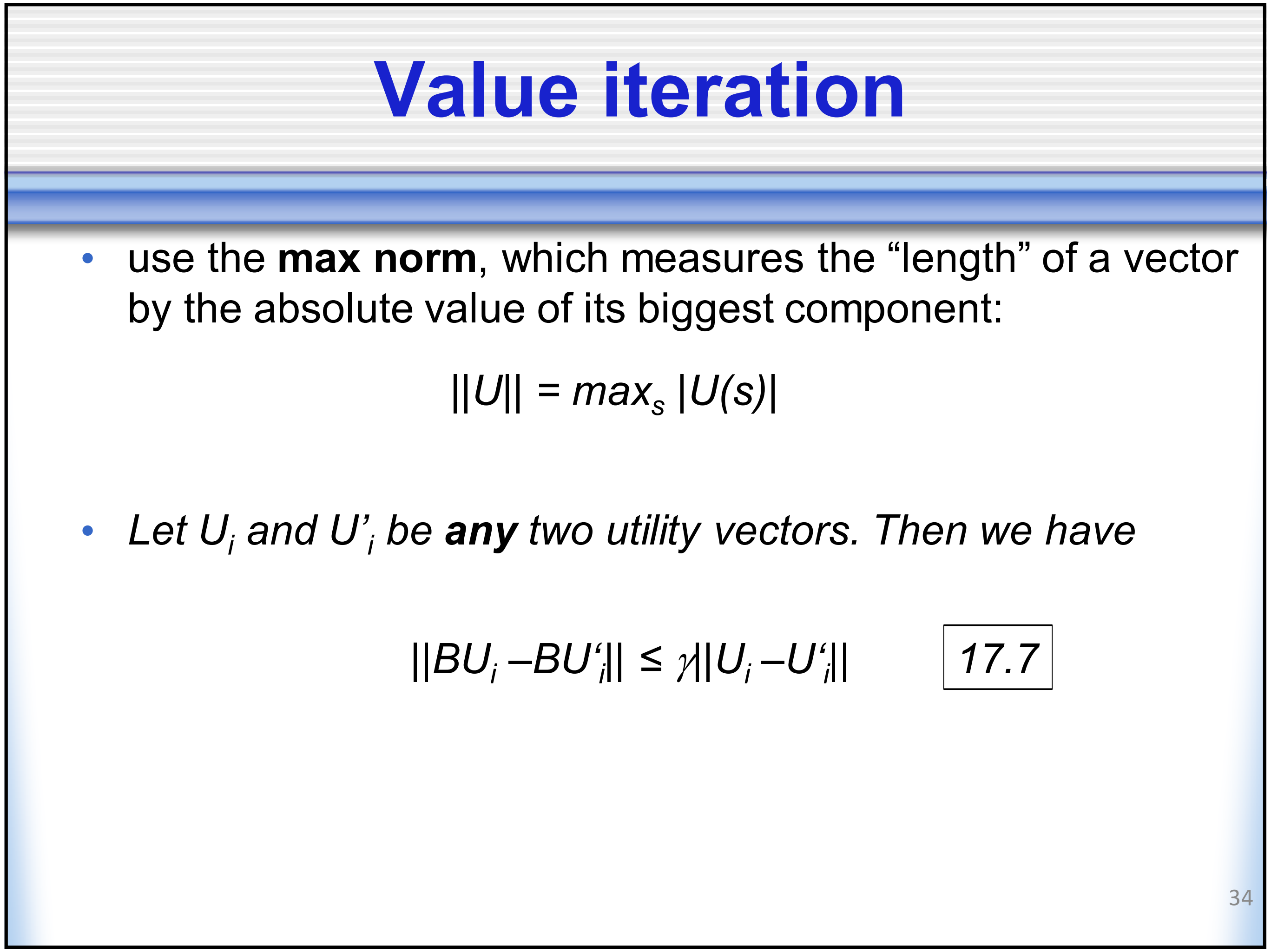
- two important properties of contractions:
  - A contraction has **only one fixed point**; if there were two fixed points they would not get closer together when the function was applied, so it would not be a contraction.
  - **When the function is applied to any argument, the value must get closer to the fixed point**, so repeated application of a contraction always reaches the fixed point in the limit.



# Value iteration

- Let $U_i$ *denote the vector of utilities for all* the states at the ith iteration. Then the Bellman update equation can be written as

$$U_{i+1} \leftarrow BU_i$$



# Value iteration

- use the **max norm**, which measures the “length” of a vector by the absolute value of its biggest component:

$$||U|| = max_s\ |U(s)|$$

- *Let $U_i$ and $U'_i$ be **any** two utility vectors. Then we have*

$$||BU_i - BU'_i|| \leq \gamma||U_i - U'_i|| \qquad \boxed{17.7}$$

# Value iteration

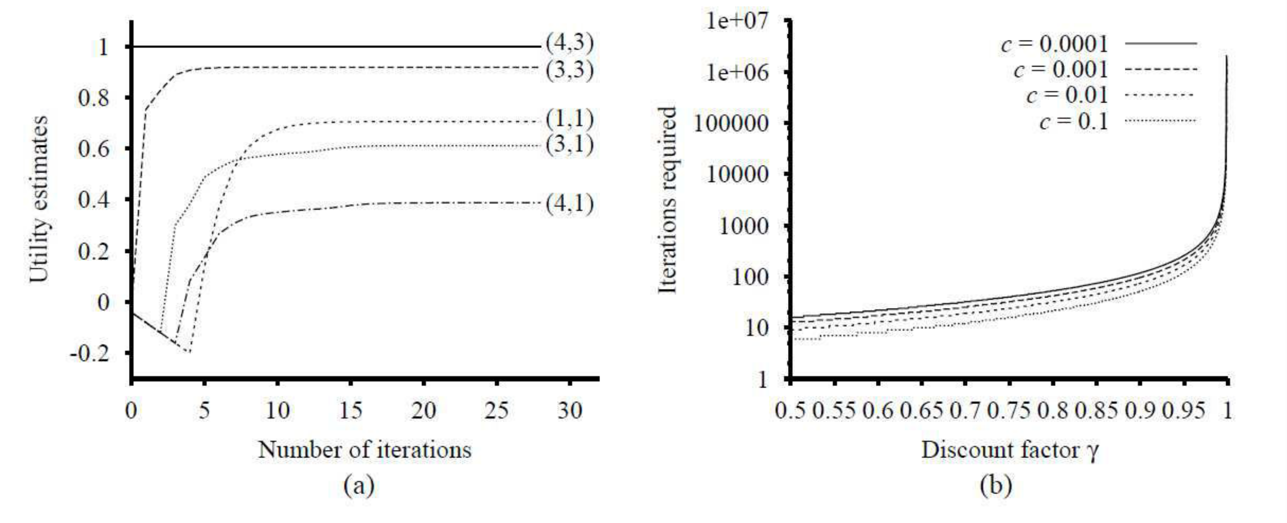
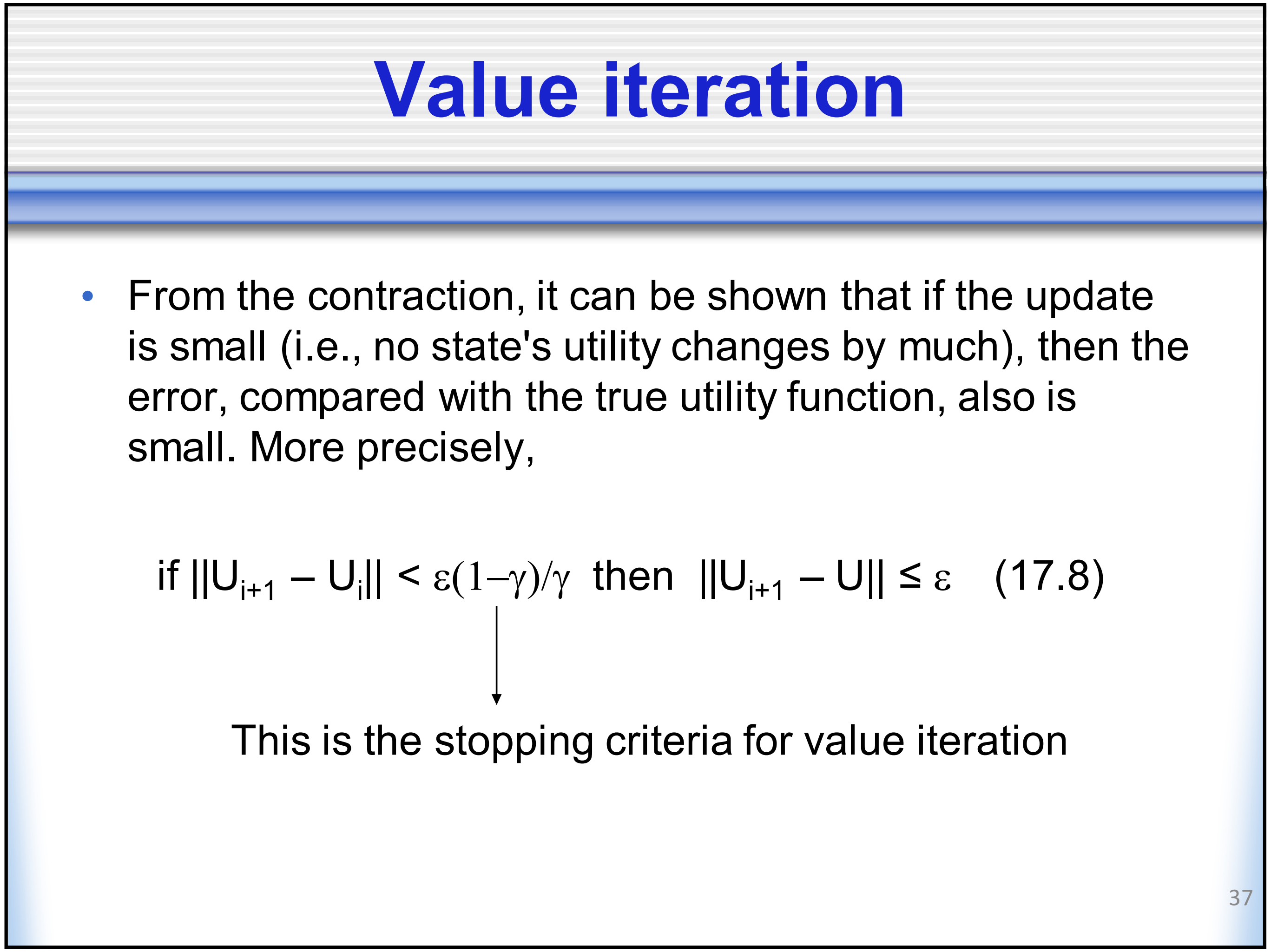


**Figure 17.5** **FILES: .** (a) Graph showing the evolution of the utilities of selected states using value iteration. (b) The number of value iterations $k$ required to guarantee an error of at most $\epsilon = c \cdot R_{\max}$, for different values of $c$, as a function of the discount factor $\gamma$.



# Value iteration

- From the contraction, it can be shown that if the update is small (i.e., no state's utility changes by much), then the error, compared with the true utility function, also is small. More precisely,

$$\text{if } \|U_{i+1} - U_i\| < \varepsilon(1-\gamma)/\gamma \quad \text{then} \quad \|U_{i+1} - U\| \le \varepsilon \qquad (17.8)$$

This is the stopping criteria for value iteration

# Value iteration

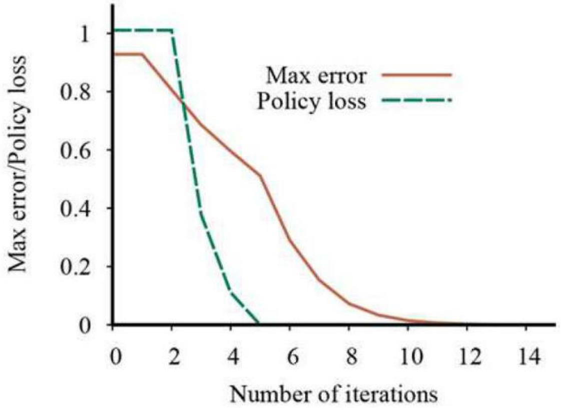
- But the crucial question is!!!! How well will I do using this utility function?
- **policy loss**

  $U^{\pi i}$ (s) is the utility obtained if πi is executed starting in *s,*
  policy loss $||U^{\pi i} – U||$ is the most the agent can lose by executing πi *instead of the optimal* policy π*

# Value iteration

- The policy loss of πi is connected to the error in $U_i$ *by the following inequality:*

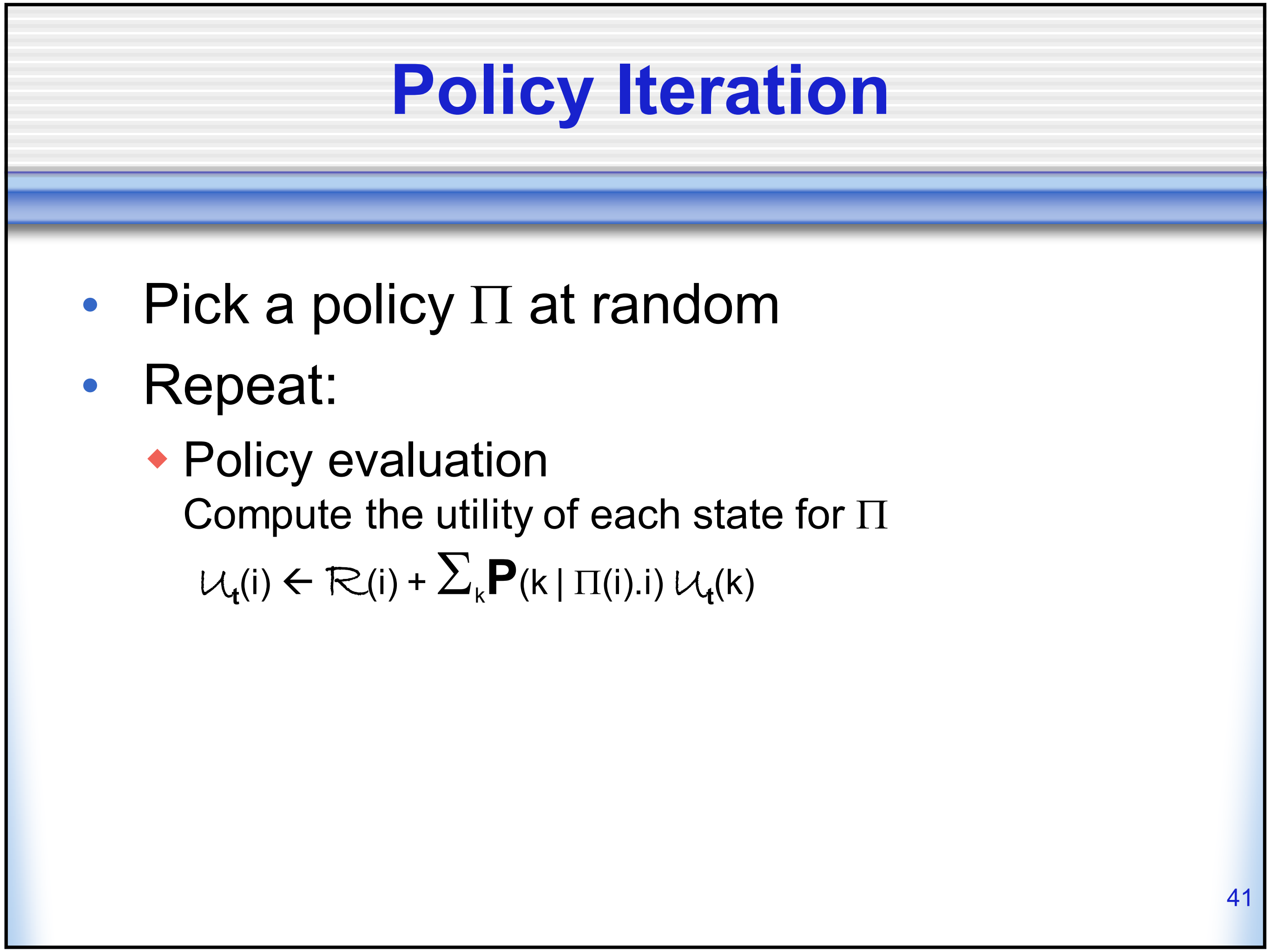
$$\text{if } \|U_i - U\| < \varepsilon \text{ then } \|U^{\pi_i} - U\| < 2\varepsilon \quad (17.9)$$



The maximum error $\|U_i - U\|$ of the utility estimates and the policy loss $\|U^{\pi_i} - U\|$, as a function of the number of iterations of value iteration on the 4 × 3 world.

# Policy Iteration

- Pick a policy $\Pi$ at random

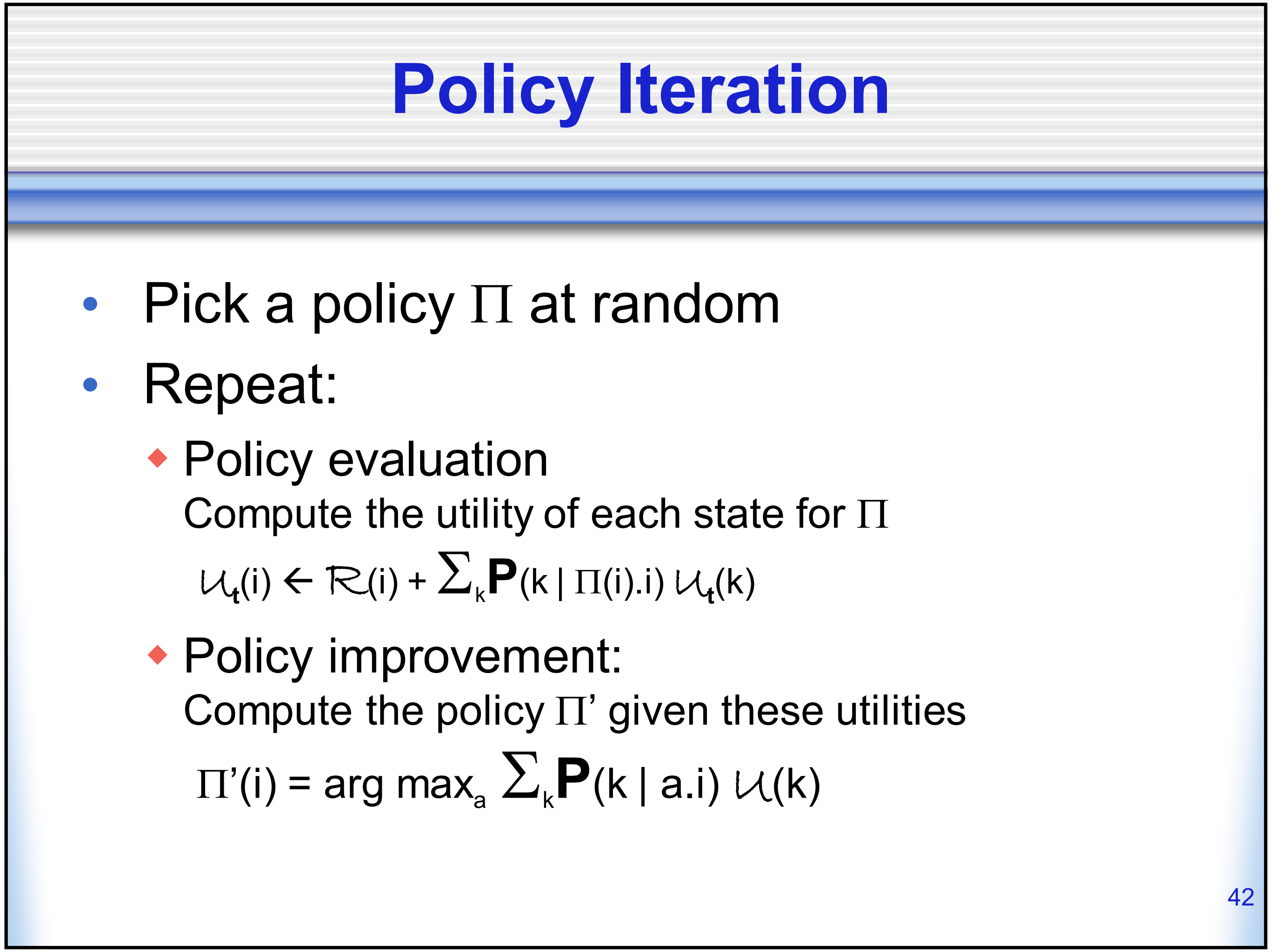


# Policy Iteration

- Pick a policy $\Pi$ at random
- Repeat:
  - Policy evaluation

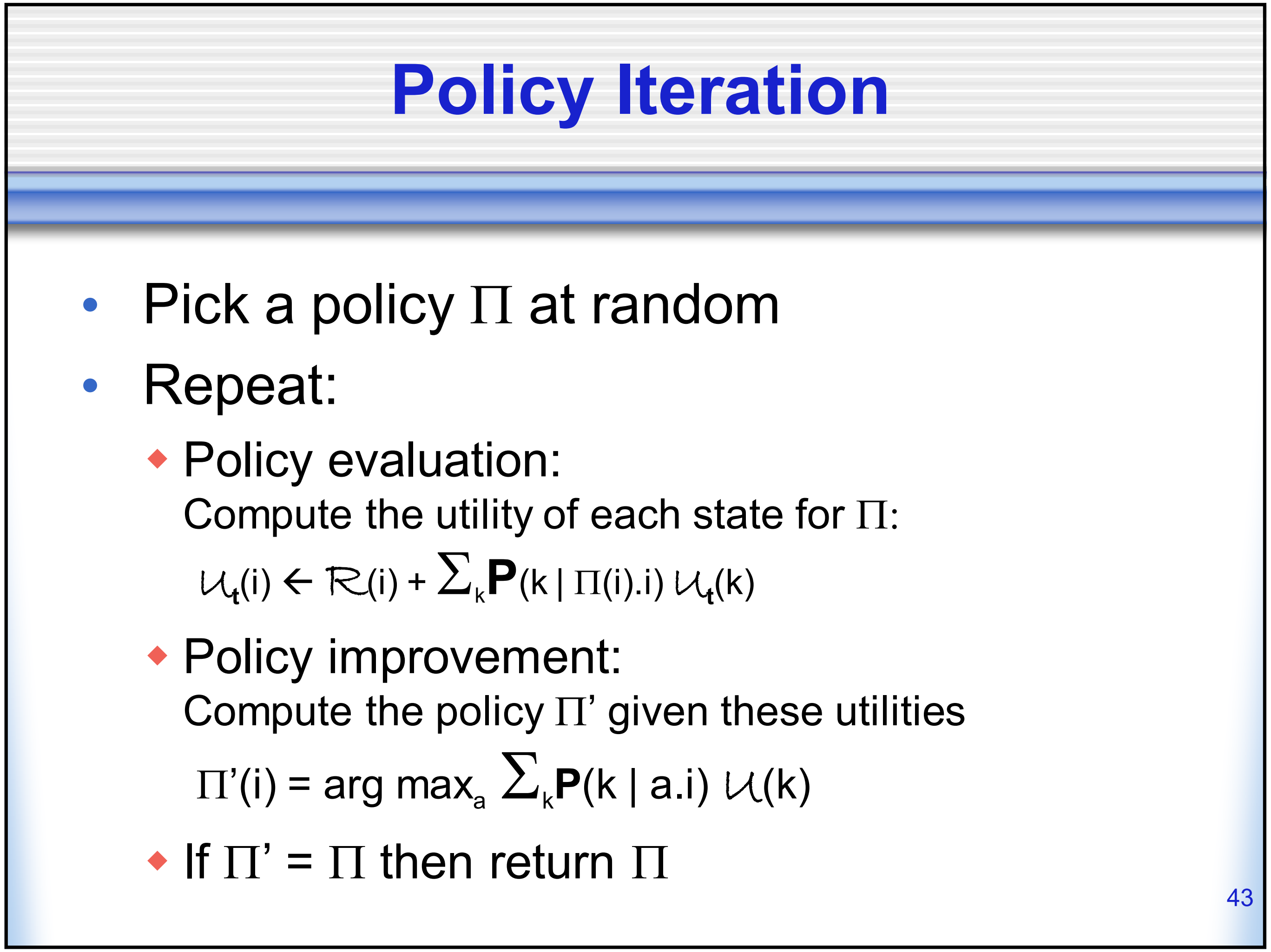
    Compute the utility of each state for $\Pi$

    $\mathcal{U}_t(i) \leftarrow \mathcal{R}(i) + \sum_k \mathbf{P}(k \mid \Pi(i).i)\, \mathcal{U}_t(k)$



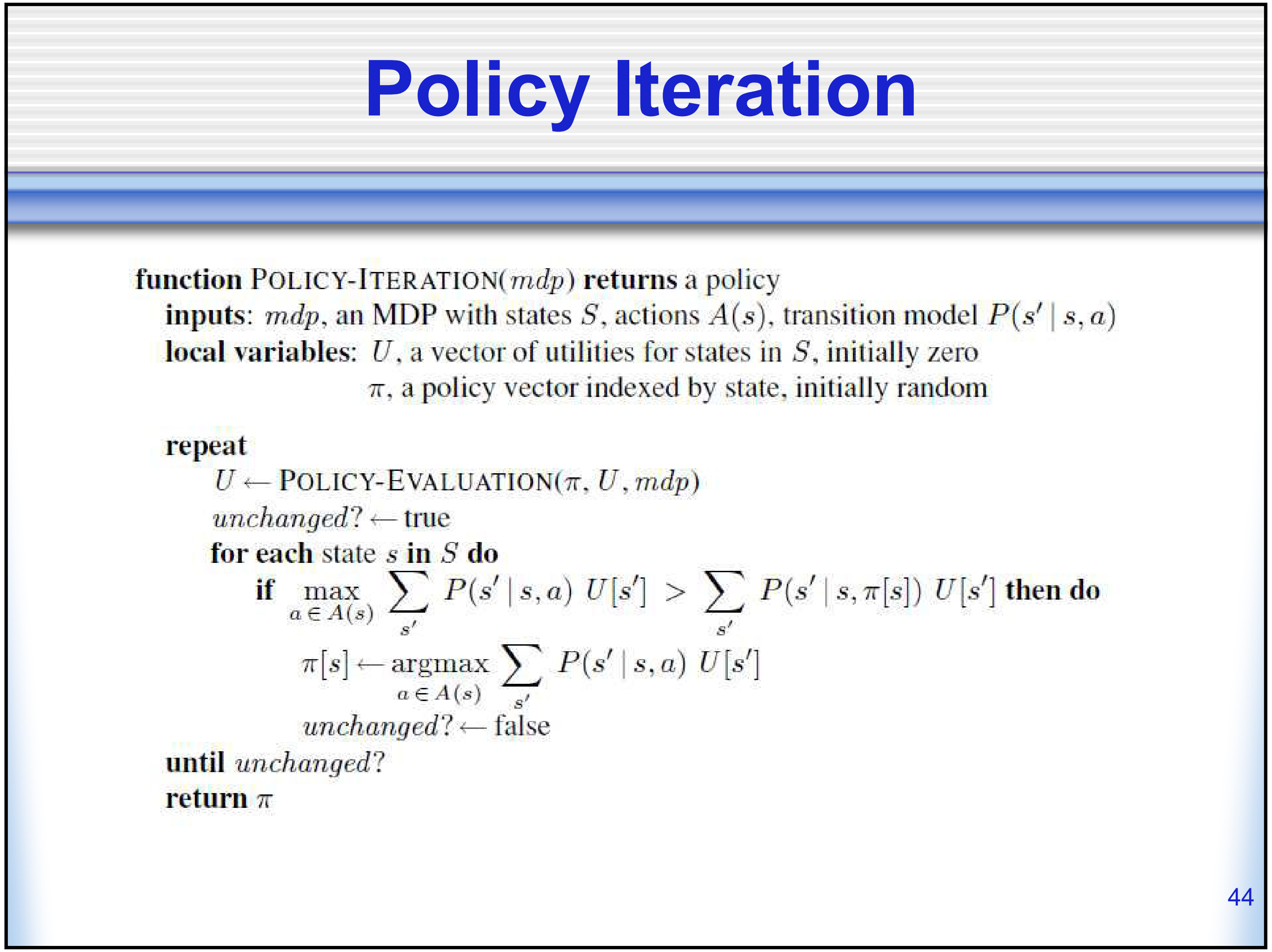
# Policy Iteration

- Pick a policy $\Pi$ at random
- Repeat:
  - Policy evaluation
    Compute the utility of each state for $\Pi$
    $\mathcal{U}_t(i) \leftarrow \mathcal{R}(i) + \sum_k \mathbf{P}(k \mid \Pi(i).i)\ \mathcal{U}_t(k)$
  - Policy improvement:
    Compute the policy $\Pi'$ given these utilities
    $\Pi'(i) = \arg\max_a \sum_k \mathbf{P}(k \mid a.i)\ \mathcal{U}(k)$



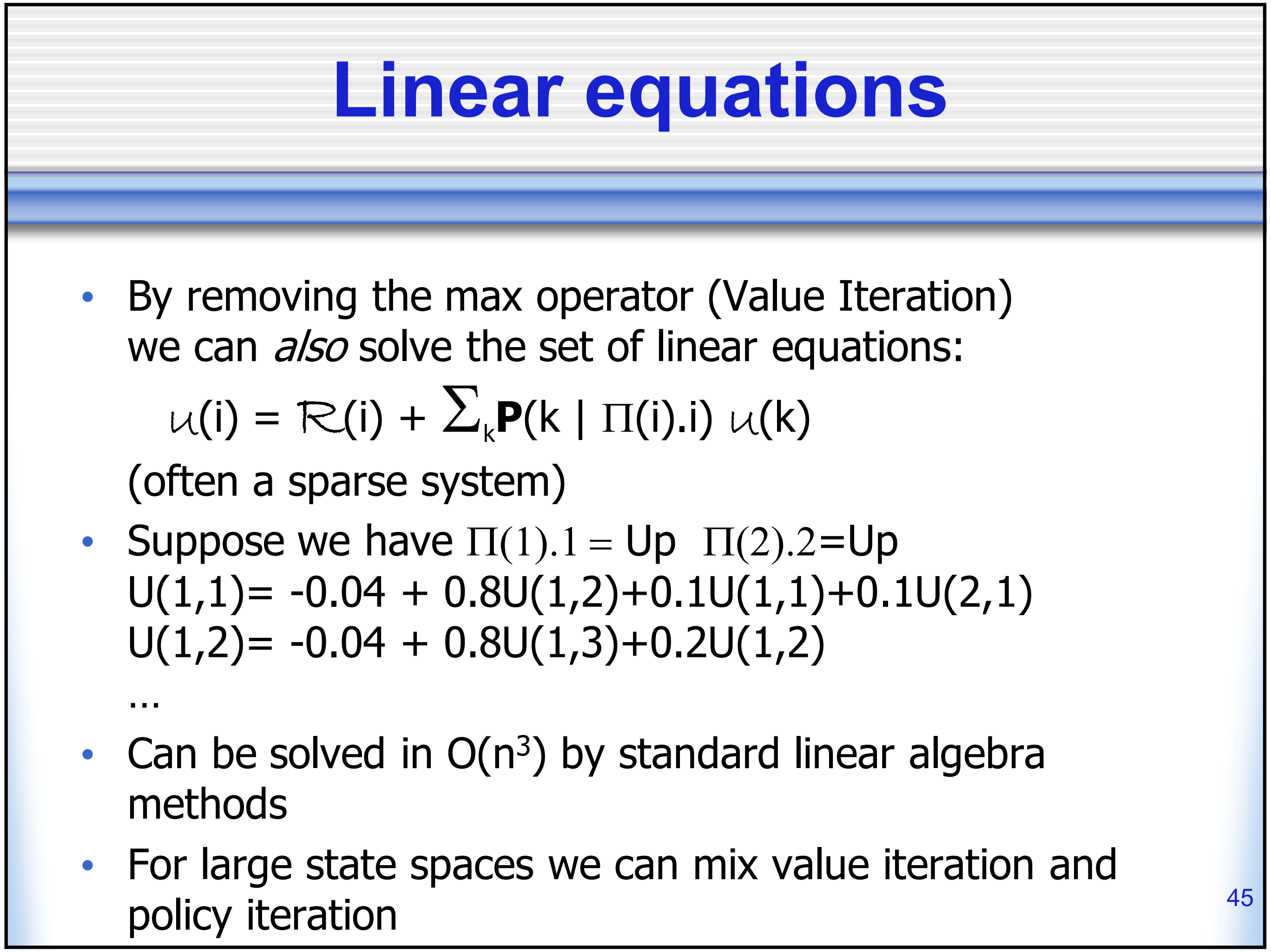
# Policy Iteration

- Pick a policy $\Pi$ at random
- Repeat:
  - Policy evaluation:
    Compute the utility of each state for $\Pi$:
    $\mathcal{U}_t(i) \leftarrow \mathcal{R}(i) + \sum_k \mathbf{P}(k \mid \Pi(i).i)\ \mathcal{U}_t(k)$
  - Policy improvement:
    Compute the policy $\Pi'$ given these utilities
    $\Pi'(i) = \arg\max_a \sum_k \mathbf{P}(k \mid a.i)\ \mathcal{U}(k)$
  - If $\Pi' = \Pi$ then return $\Pi$



# Policy Iteration

```
function POLICY-ITERATION($mdp$) returns a policy
  inputs: $mdp$, an MDP with states $S$, actions $A(s)$, transition model $P(s' \mid s, a)$
  local variables: $U$, a vector of utilities for states in $S$, initially zero
                   $\pi$, a policy vector indexed by state, initially random

  repeat
     $U \leftarrow$ POLICY-EVALUATION($\pi, U, mdp$)
     $unchanged? \leftarrow$ true
     for each state $s$ in $S$ do
        if $\max_{a \in A(s)} \sum_{s'} P(s' \mid s, a)\ U[s'] > \sum_{s'} P(s' \mid s, \pi[s])\ U[s']$ then do
           $\pi[s] \leftarrow \operatorname{argmax}_{a \in A(s)} \sum_{s'} P(s' \mid s, a)\ U[s']$
           $unchanged? \leftarrow$ false
  until $unchanged?$
  return $\pi$
```



# Linear equations

- By removing the max operator (Value Iteration) we can *also* solve the set of linear equations:

  $\mathcal{U}(i) = \mathcal{R}(i) + \sum_k \mathbf{P}(k \mid \Pi(i).i)\ \mathcal{U}(k)$

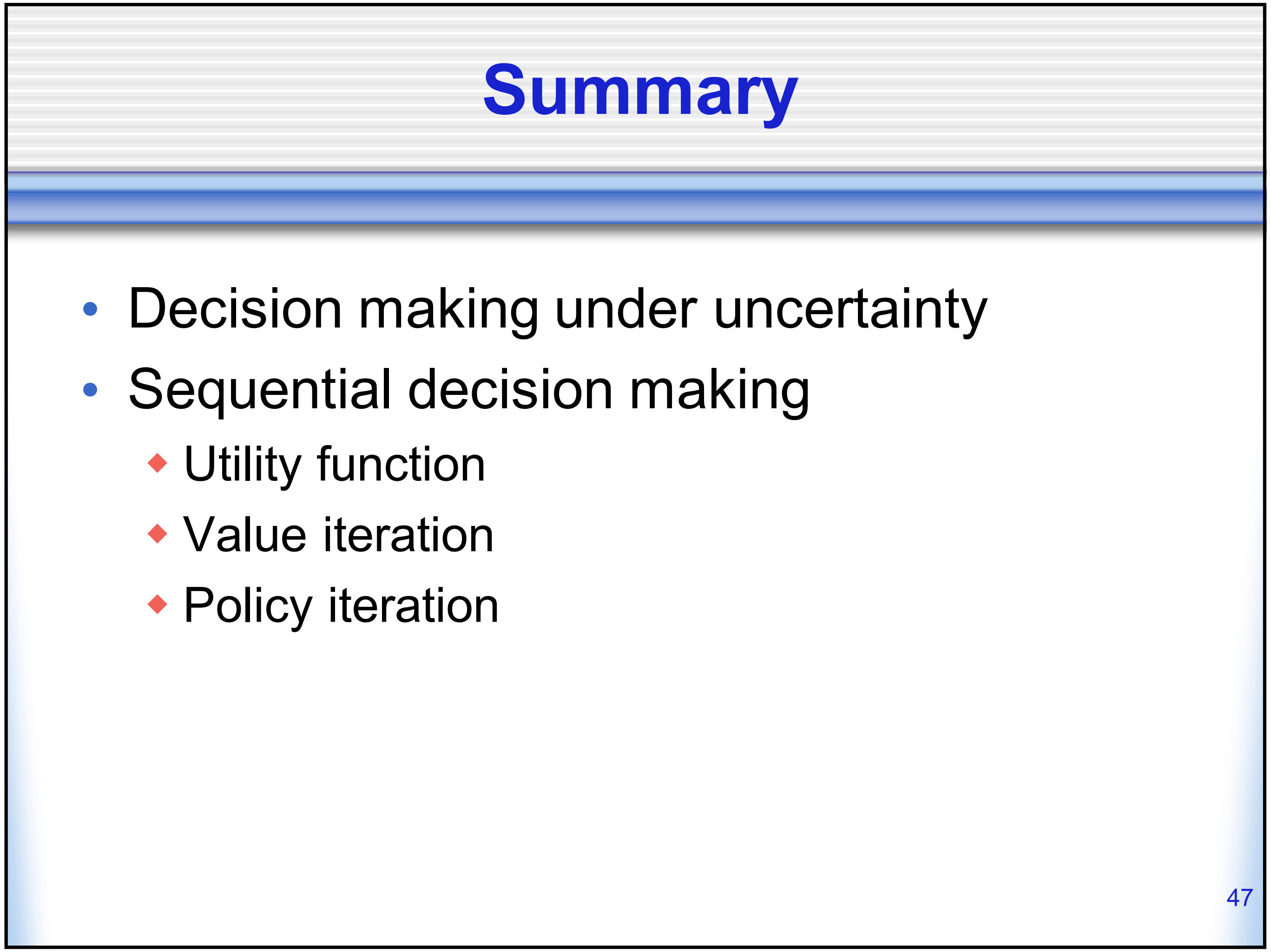
  (often a sparse system)
- Suppose we have $\Pi(1).1 = \text{Up}$ $\Pi(2).2=\text{Up}$
  U(1,1)= -0.04 + 0.8U(1,2)+0.1U(1,1)+0.1U(2,1)
  U(1,2)= -0.04 + 0.8U(1,3)+0.2U(1,2)
  …
- Can be solved in $O(n^3)$ by standard linear algebra methods
- For large state spaces we can mix value iteration and policy iteration

# Further optimization

- All algorithms require updating the utility or policy for all states at once.
- At each step we can also select a subset for updating

  **asynchronous policy iteration/mod. value iter.**

  (can show it will converge if some conditions for initial policy and utility function hold)
- Leads to heuristic algorithms that concentrate on states that are likely to be reached by a good policy.
  - “if one has no intention of throwing oneself off a cliff, one should not spend time worrying about the exact value of the resulting state”



# Summary

- Decision making under uncertainty
- Sequential decision making
  - Utility function
  - Value iteration
  - Policy iteration

# Intelligent Autonomous Agents

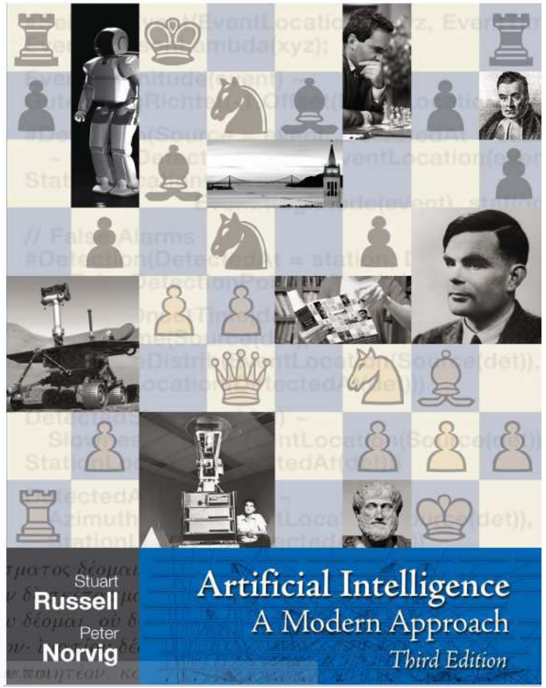
## Agents and Rational Behavior

### Topic 9: Decision-Making under Uncertainty
### Decision-Theoretic Agent Design

Ralf Möller, Rainer Marrone

Hamburg University of Technology

# Literature

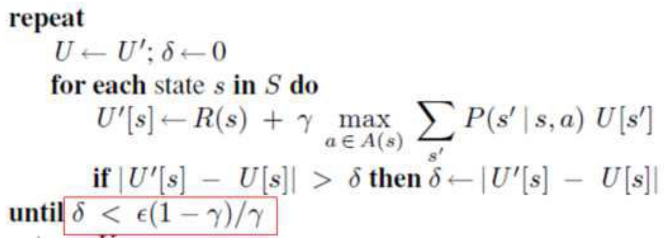


- Chapter 17

2

# Last time

- Sequential decision making (uncertain actions)
  - Need a policy -> best action for each possible state
- Finding the best policy
  - Value iteration



```
repeat
    $U \leftarrow U'; \delta \leftarrow 0$
    for each state $s$ in $S$ do
        $U'[s] \leftarrow R(s) + \gamma \max_{a \in A(s)} \sum_{s'} P(s' \mid s, a)\, U[s']$
        if $|U'[s] - U[s]| > \delta$ then $\delta \leftarrow |U'[s] - U[s]|$
until $\boxed{\delta < \epsilon(1-\gamma)/\gamma}$
```

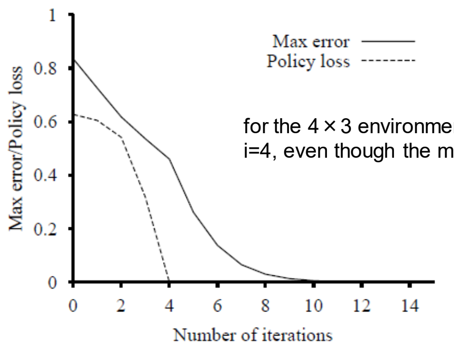
$$\pi^*(s) = \arg\max_a \sum_{s'} P(s' \mid s, a) U(s')$$

  - Bellman update is a contraction → Lead to the definition of when to stop value iteration.

# Policy Loss

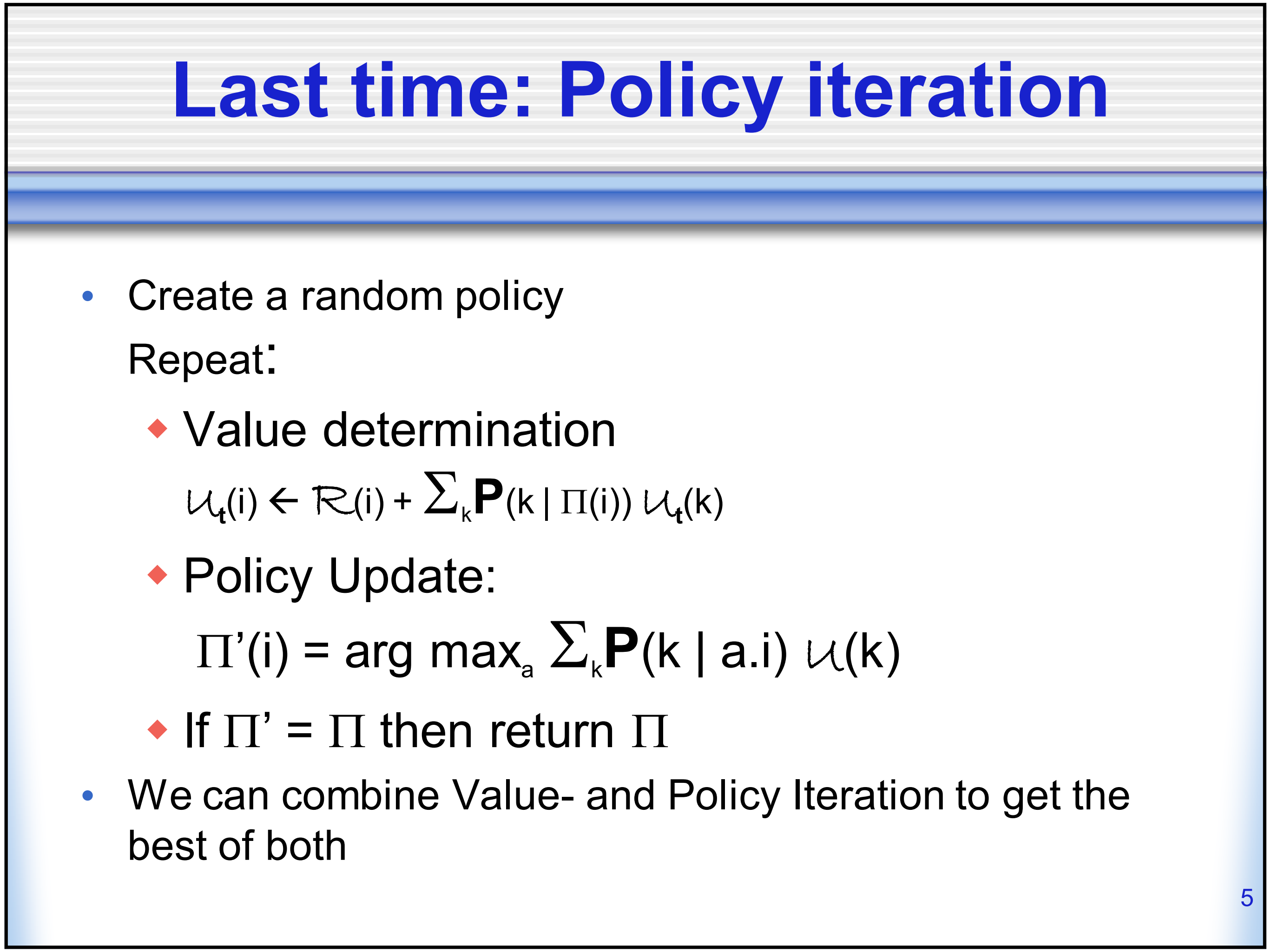
- The policy loss of $\pi_i$ is connected to the error in $U_i$ *by the following inequality:*

if $||U_i - U|| < \varepsilon$ then $||U^{\pi_i} - U|| < 2\varepsilon$ (17.9)



for the 4 × 3 environment with γ =0.9. The policy πi is optimal when i=4, even though the maximum error in Ui is still 0.46

**Figure 17.6** **FILES: .** The maximum error $||U_i - U||$ of the utility estimates and the policy loss $||U^{\pi_i} - U||$, as a function of the number of iterations of value iteration.



# Last time: Policy iteration

- Create a random policy

  Repeat:
  - Value determination

    $\mathcal{U}_t(i) \leftarrow \mathcal{R}(i) + \sum_k \mathbf{P}(k \mid \Pi(i))\ \mathcal{U}_t(k)$
  - Policy Update:

    $\Pi'(i) = \arg\max_a \sum_k \mathbf{P}(k \mid a.i)\ \mathcal{U}(k)$
  - If $\Pi' = \Pi$ then return $\Pi$
- We can combine Value- and Policy Iteration to get the best of both

# Further optimization

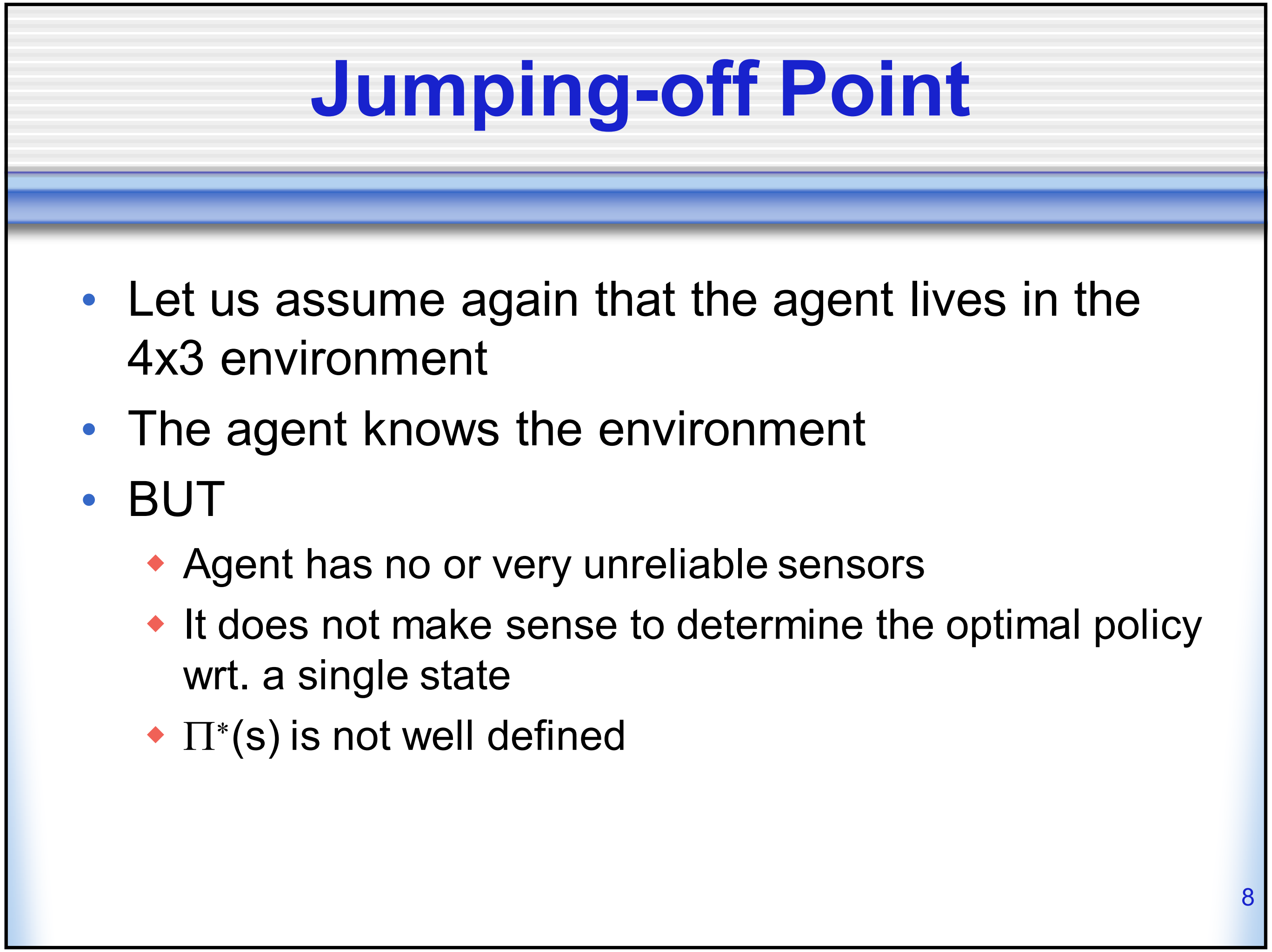
- All algorithms require updating the utility or policy for all states at once.
- At each step we can also select a subset for updating

  **asynchronous policy iteration/mod. value iter.**

  (can show it will converge if some conditions for initial policy and utility function hold)
- Leads to heuristic algorithms that concentrate on states that are likely to be reached by a good policy.
  - "if one has no intention of throwing oneself off a cliff, one should not spend time worrying about the exact value of the resulting state"

# Summary

- Decision making under uncertainty
- Sequential decision making
  - Utility of histories
  - Value iteration
  - Policy iteration



# Jumping-off Point

- Let us assume again that the agent lives in the 4x3 environment
- The agent knows the environment
- BUT
  - Agent has no or very unreliable sensors
  - It does not make sense to determine the optimal policy wrt. a single state
  - $\Pi^*$(s) is not well defined

# POMDP: Uncertainty

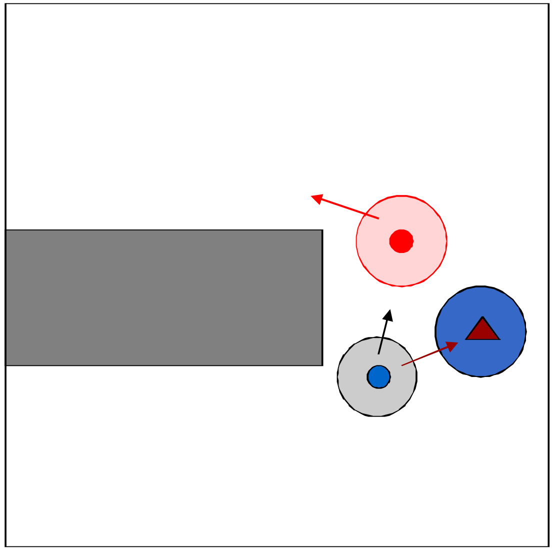
- Uncertainty about the action outcome
- Uncertainty about the world state due to imperfect (partial) information

# Example: Target Tracking

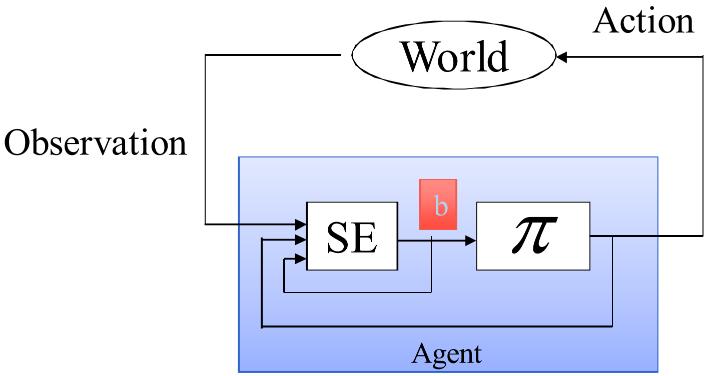
There is uncertainty in the robot’s and target’s positions; this uncertainty grows with further motion

There is a risk that the target may escape behind the corner, requiring the robot to move appropriately

But there is a positioning landmark nearby. Should the robot try to reduce its position uncertainty?



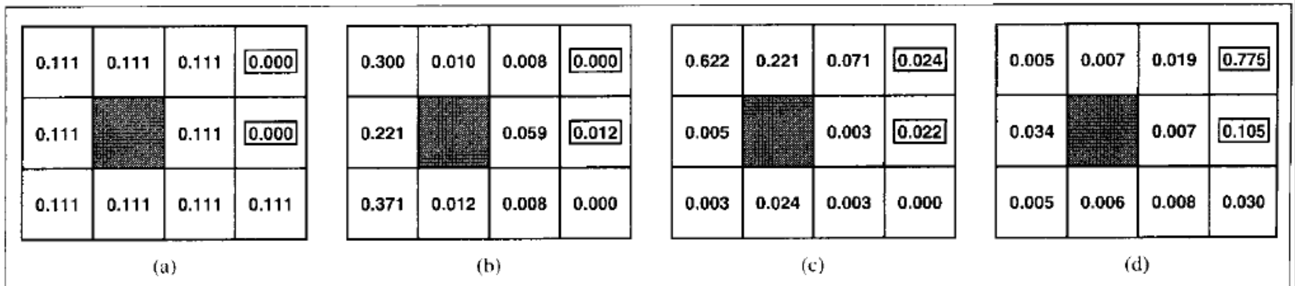
# Decision cycle of a POMDP agent



- Given the current belief state *b*, execute the action
$$a = \pi^*(b)$$
- Receive observation *o*
- Set the current belief state to *FORWARD(b,a,o)* and repeat

# Example Scenario

*The agent has no sensors!!!*



|  |  |  |  |
|---|---|---|---|
| 0.111 | 0.111 | 0.111 | 0.000 |
| 0.111 | ▨ | 0.111 | 0.000 |
| 0.111 | 0.111 | 0.111 | 0.111 |

(a)

|  |  |  |  |
|---|---|---|---|
| 0.300 | 0.010 | 0.008 | 0.000 |
| 0.221 | ▨ | 0.059 | 0.012 |
| 0.371 | 0.012 | 0.008 | 0.000 |

(b)

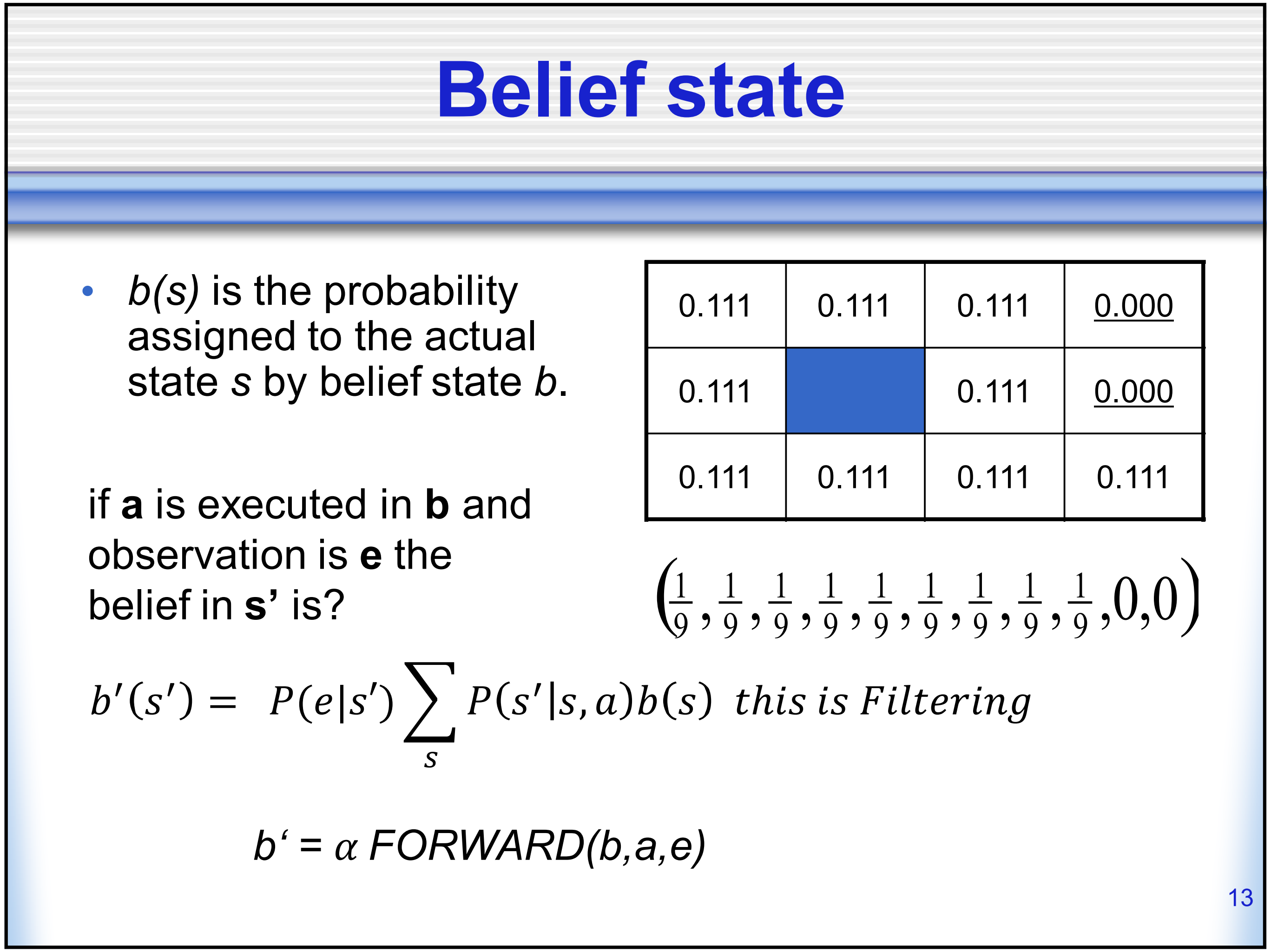
|  |  |  |  |
|---|---|---|---|
| 0.622 | 0.221 | 0.071 | 0.024 |
| 0.005 | ▨ | 0.003 | 0.022 |
| 0.003 | 0.024 | 0.003 | 0.000 |

(c)

|  |  |  |  |
|---|---|---|---|
| 0.005 | 0.007 | 0.019 | 0.775 |
| 0.034 | ▨ | 0.007 | 0.105 |
| 0.005 | 0.006 | 0.008 | 0.030 |

(d)

**Figure 17.8** (a) The initial probability distribution for the agent's location. (b) After moving *Left* five times. (c) After moving *Up* five times. (d) After moving *Right* five times.



# Belief state

- *b(s)* is the probability assigned to the actual state *s* by belief state *b*.

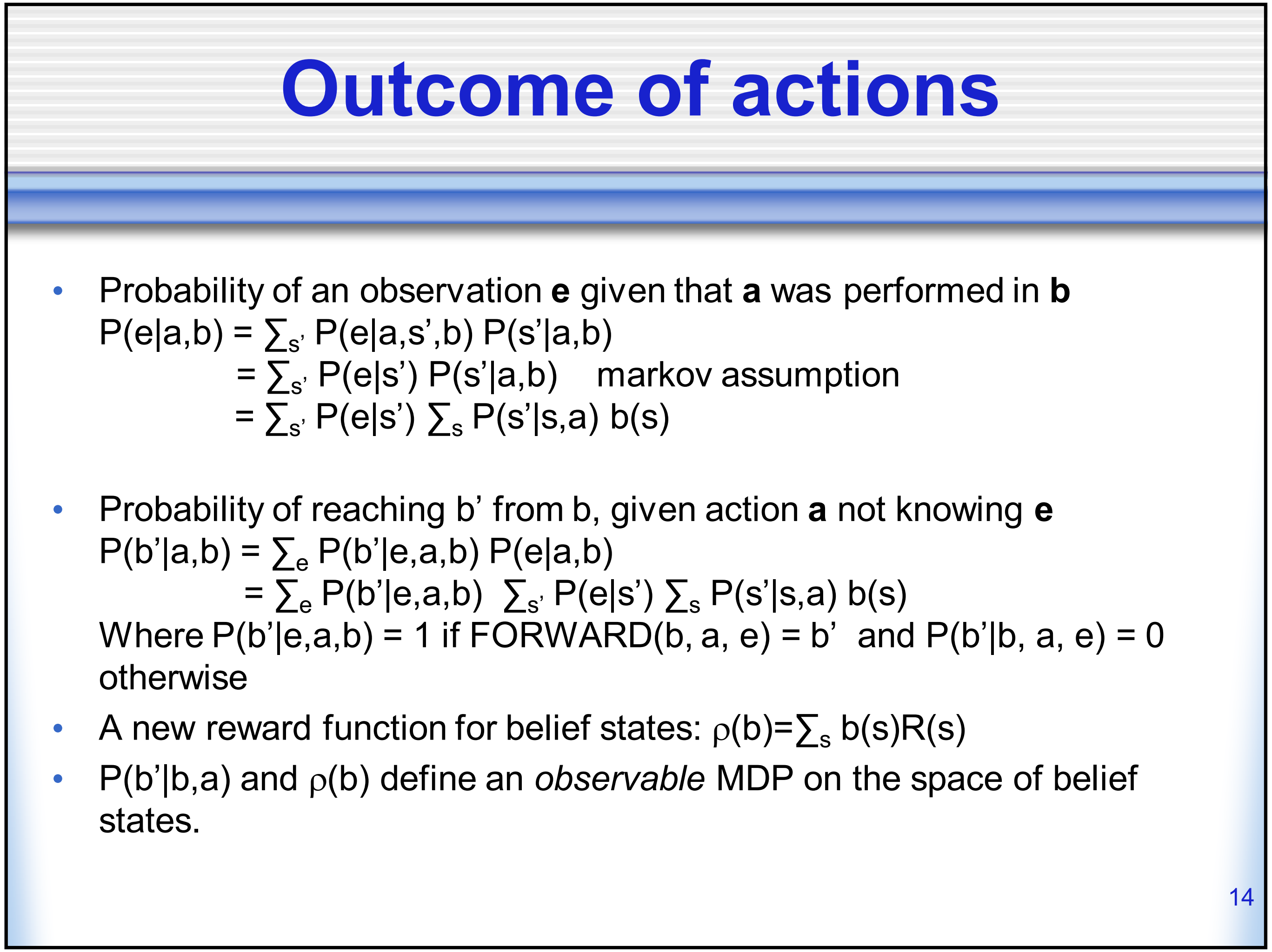
| | | | |
|---|---|---|---|
| 0.111 | 0.111 | 0.111 | 0.000 |
| 0.111 | ■ | 0.111 | 0.000 |
| 0.111 | 0.111 | 0.111 | 0.111 |

$$\left(\frac{1}{9},\frac{1}{9},\frac{1}{9},\frac{1}{9},\frac{1}{9},\frac{1}{9},\frac{1}{9},\frac{1}{9},\frac{1}{9},0,0\right)$$

if **a** is executed in **b** and observation is **e** the belief in **s'** is?

$$b'(s') = \; P(e|s')\sum_{s} P(s'|s,a)b(s) \;\; this\ is\ Filtering$$

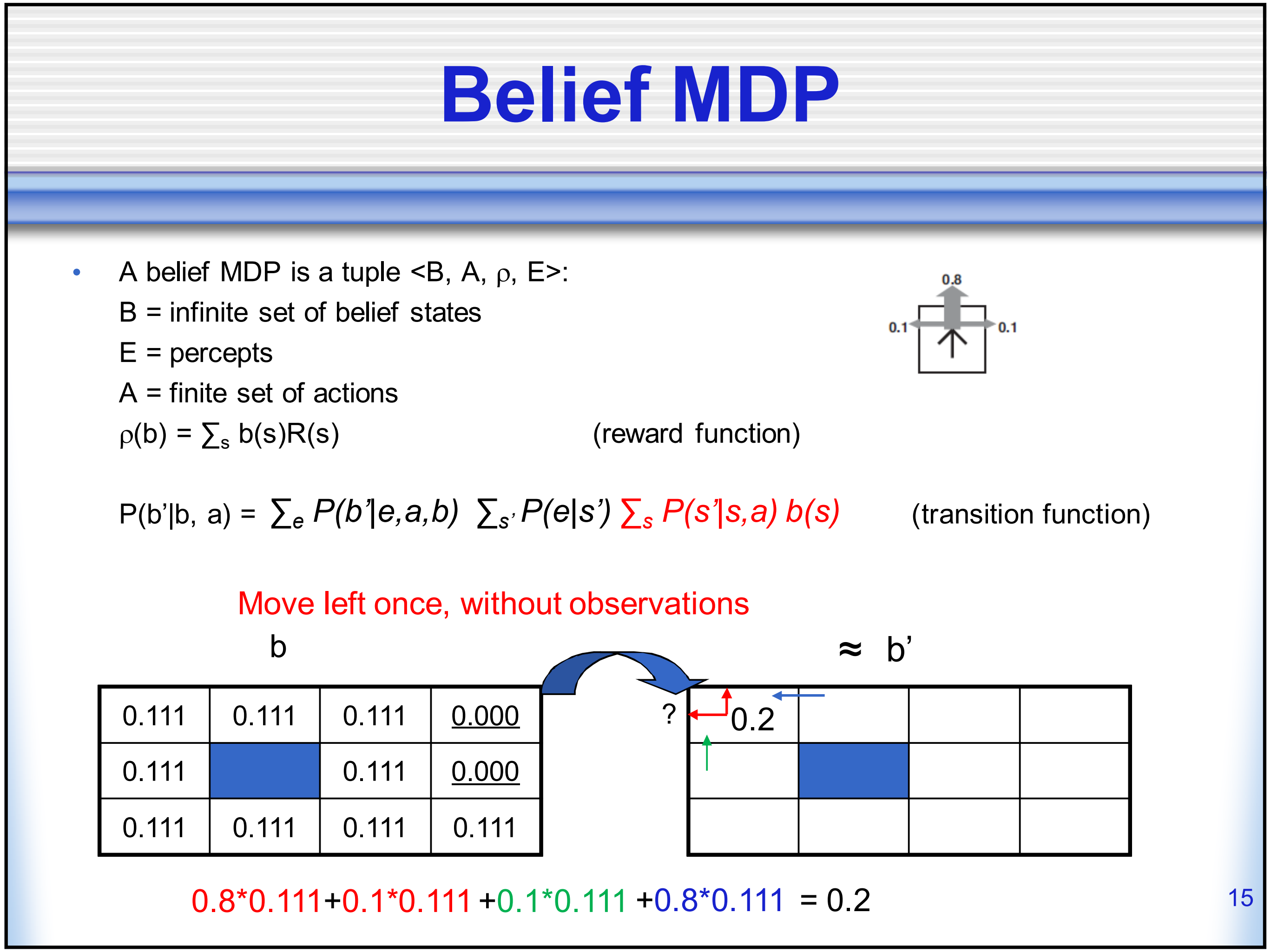
*b' =* $\alpha$ *FORWARD(b,a,e)*



# Outcome of actions

- Probability of an observation **e** given that **a** was performed in **b**
  $P(e|a,b) = \sum_{s'} P(e|a,s',b)\ P(s'|a,b)$
  $= \sum_{s'} P(e|s')\ P(s'|a,b)$ markov assumption
  $= \sum_{s'} P(e|s') \sum_{s} P(s'|s,a)\ b(s)$

- Probability of reaching b’ from b, given action **a** not knowing **e**
  $P(b'|a,b) = \sum_{e} P(b'|e,a,b)\ P(e|a,b)$
  $= \sum_{e} P(b'|e,a,b) \ \sum_{s'} P(e|s') \sum_{s} P(s'|s,a)\ b(s)$
  Where $P(b'|e,a,b) = 1$ if FORWARD(b, a, e) = b’ and $P(b'|b, a, e) = 0$ otherwise
- A new reward function for belief states: $\rho(b)=\sum_{s} b(s)R(s)$
- $P(b'|b,a)$ and $\rho(b)$ define an *observable* MDP on the space of belief states.



# Belief MDP

- A belief MDP is a tuple <B, A, ρ, E>:
  B = infinite set of belief states
  E = percepts
  A = finite set of actions
  $\rho(b) = \sum_s b(s)R(s)$ (reward function)

$P(b'|b, a) = \sum_e P(b'|e,a,b) \sum_{s'} P(e|s') \sum_s P(s'|s,a)\, b(s)$ (transition function)

Move left once, without observations

b

| | | | |
|---|---|---|---|
| 0.111 | 0.111 | 0.111 | 0.000 |
| 0.111 | ■ | 0.111 | 0.000 |
| 0.111 | 0.111 | 0.111 | 0.111 |

≈ b'

?

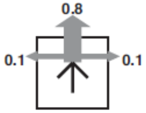
| | | | |
|---|---|---|---|
| 0.2 | | | |
| | ■ | | |
| | | | |

$0.8*0.111+0.1*0.111+0.1*0.111+0.8*0.111 = 0.2$

15

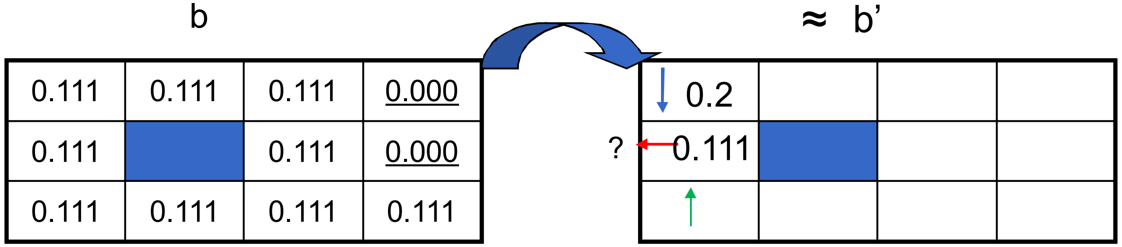
# Belief MDP

- A belief MDP is a tuple <B, A, ρ, E>:
  B = infinite set of belief states
  E = percepts
  A = finite set of actions
  $\rho(b) = \sum_s b(s)R(s)$ (reward function)



$P(b'|b, a) = \sum_e P(b'|e,a,b) \sum_{s'} P(e|s') \sum_s P(s'|s,a)\, b(s)$ (transition function)

Move left once, without observations



b

| | | | |
|---|---|---|---|
| 0.111 | 0.111 | 0.111 | 0.000 |
| 0.111 | ■ | 0.111 | 0.000 |
| 0.111 | 0.111 | 0.111 | 0.111 |

≈ b'

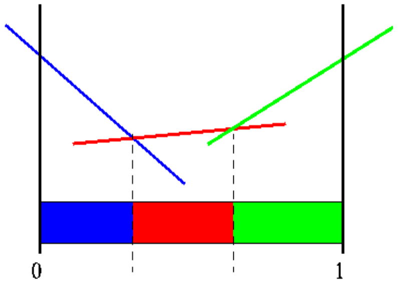
| | | | |
|---|---|---|---|
| 0.2 | | | |
| 0.111 | ■ | | |
| | | | |

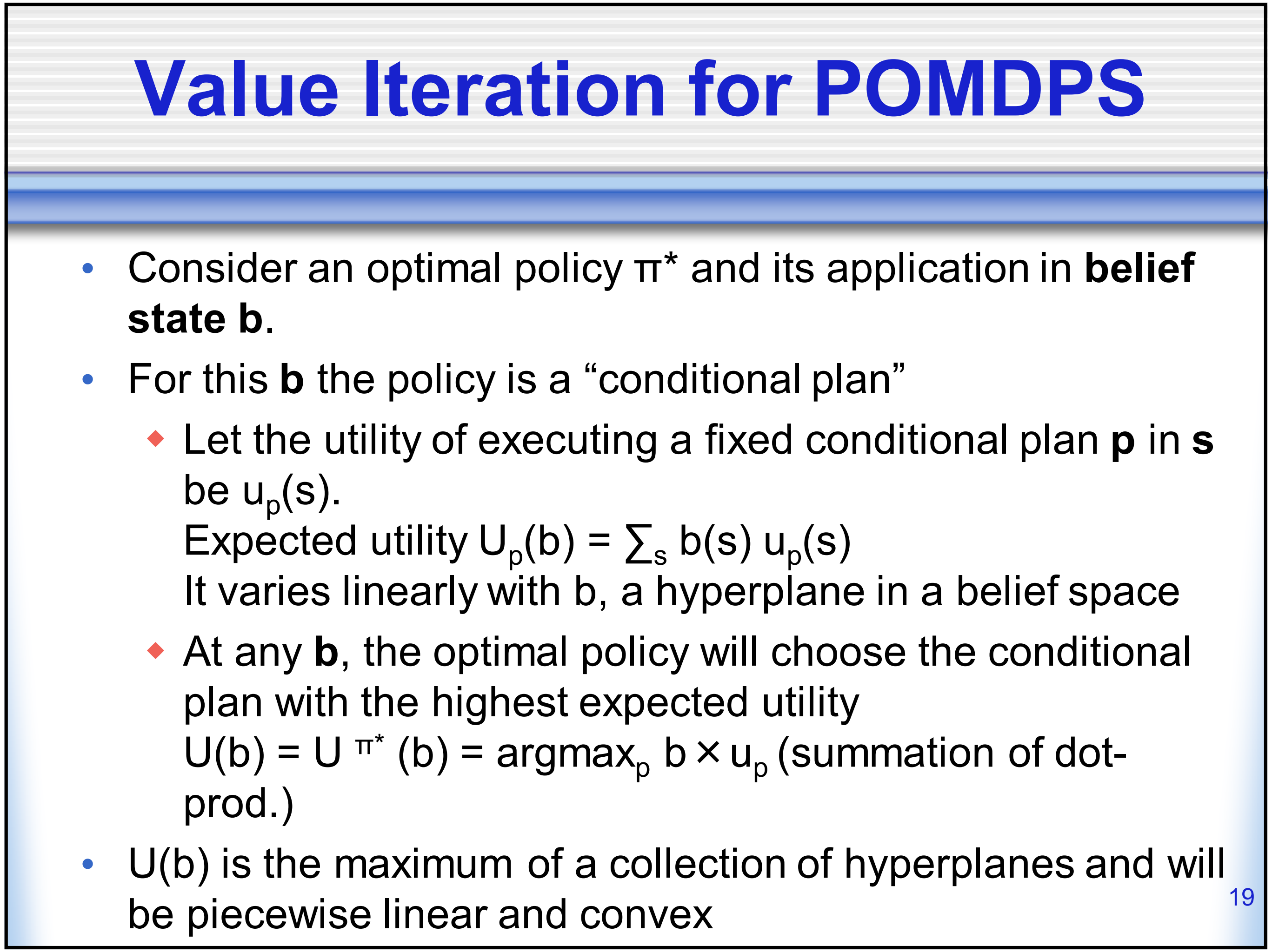
0.8*0.111 +0.1*0.111 +0.1*0.111 = 0.111

# Solutions for POMDP

- Methods based on *value* and *policy iteration*:

A policy $\pi(b)$ can be represented as a set of *regions* of belief state space, each of which is associated with a particular optimal action. The value function associates a distinct *linear* function of *b* with each region. Each value or policy iteration step refines the boundaries of the regions and may introduce new regions.



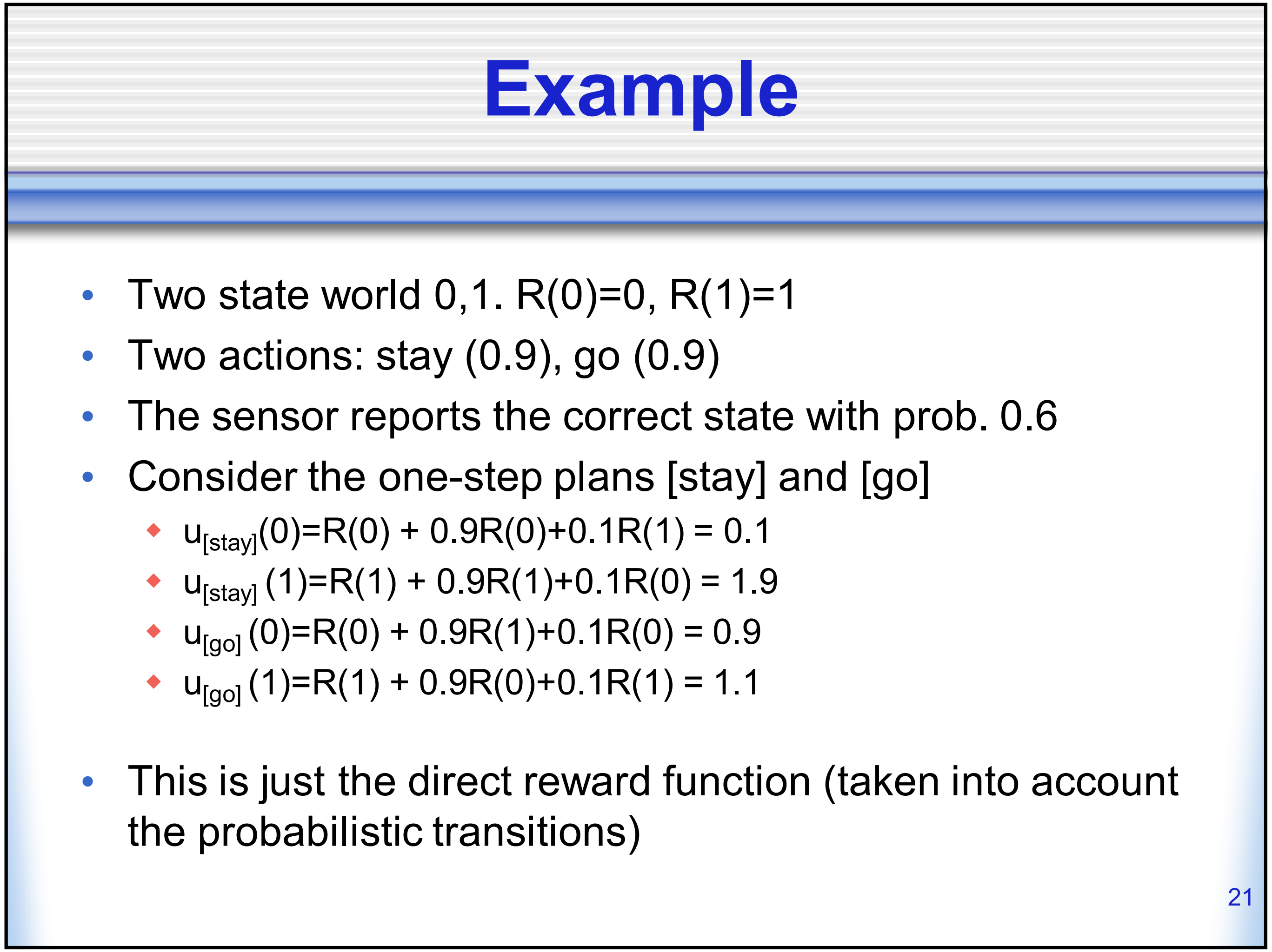


# Value Iteration for POMDPS

- Consider an optimal policy $\pi^*$ and its application in **belief state b**.
- For this **b** the policy is a “conditional plan”
  - Let the utility of executing a fixed conditional plan **p** in **s** be $u_p(s)$.
    Expected utility $U_p(b) = \sum_s b(s)\, u_p(s)$
    It varies linearly with b, a hyperplane in a belief space
  - At any **b**, the optimal policy will choose the conditional plan with the highest expected utility
    $U(b) = U^{\pi^*}(b) = \text{argmax}_p\ b \times u_p$ (summation of dot-prod.)
- U(b) is the maximum of a collection of hyperplanes and will be piecewise linear and convex

# Example: Conditional Plans

- Two state world 0,1
- Two actions: stay(s), go(s)
  - Actions achieve intended effect with some probability p
- One-step plan [go], [stay]
- Two-step plans are conditional
  - [a1, IF percept = 0 THEN a2 ELSE a3]
  - Shorthand notation: [a1, a2/a3]
- n-step plan are trees with nodes attached with actions and edges attached with percepts

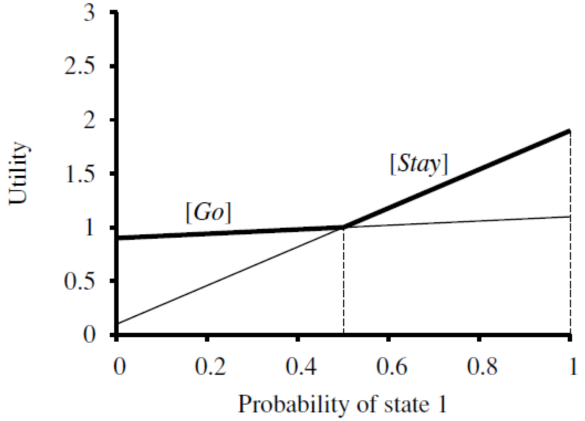


# Example

- Two state world 0,1. R(0)=0, R(1)=1
- Two actions: stay (0.9), go (0.9)
- The sensor reports the correct state with prob. 0.6
- Consider the one-step plans [stay] and [go]
  - $u_{[stay]}(0)=R(0) + 0.9R(0)+0.1R(1) = 0.1$
  - $u_{[stay]}(1)=R(1) + 0.9R(1)+0.1R(0) = 1.9$
  - $u_{[go]}(0)=R(0) + 0.9R(1)+0.1R(0) = 0.9$
  - $u_{[go]}(1)=R(1) + 0.9R(0)+0.1R(1) = 1.1$

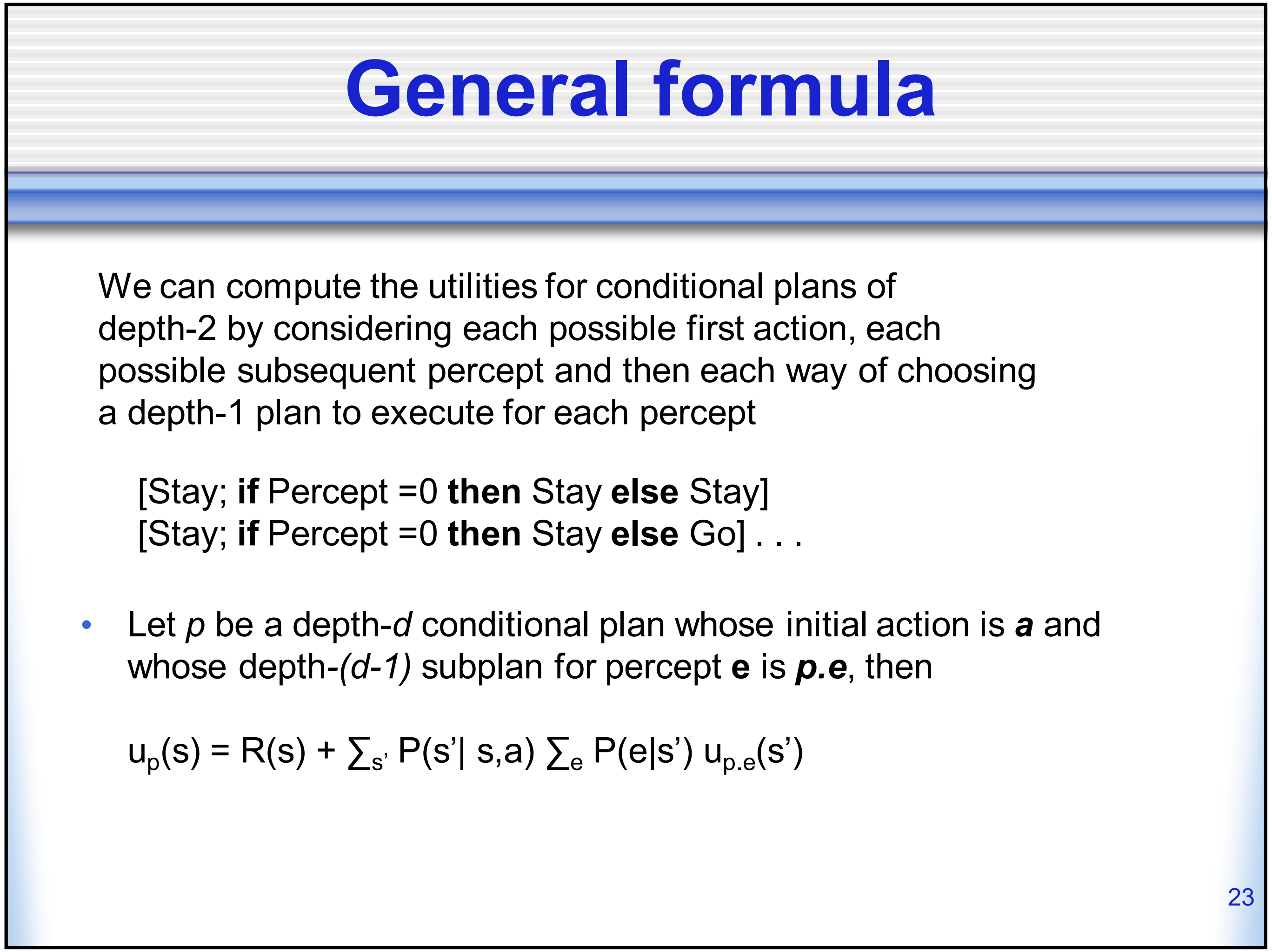
- This is just the direct reward function (taken into account the probabilistic transitions)

# Example



Utility of two one-step plans as a function of b(1)

if(b(1)>0.5) stay else go



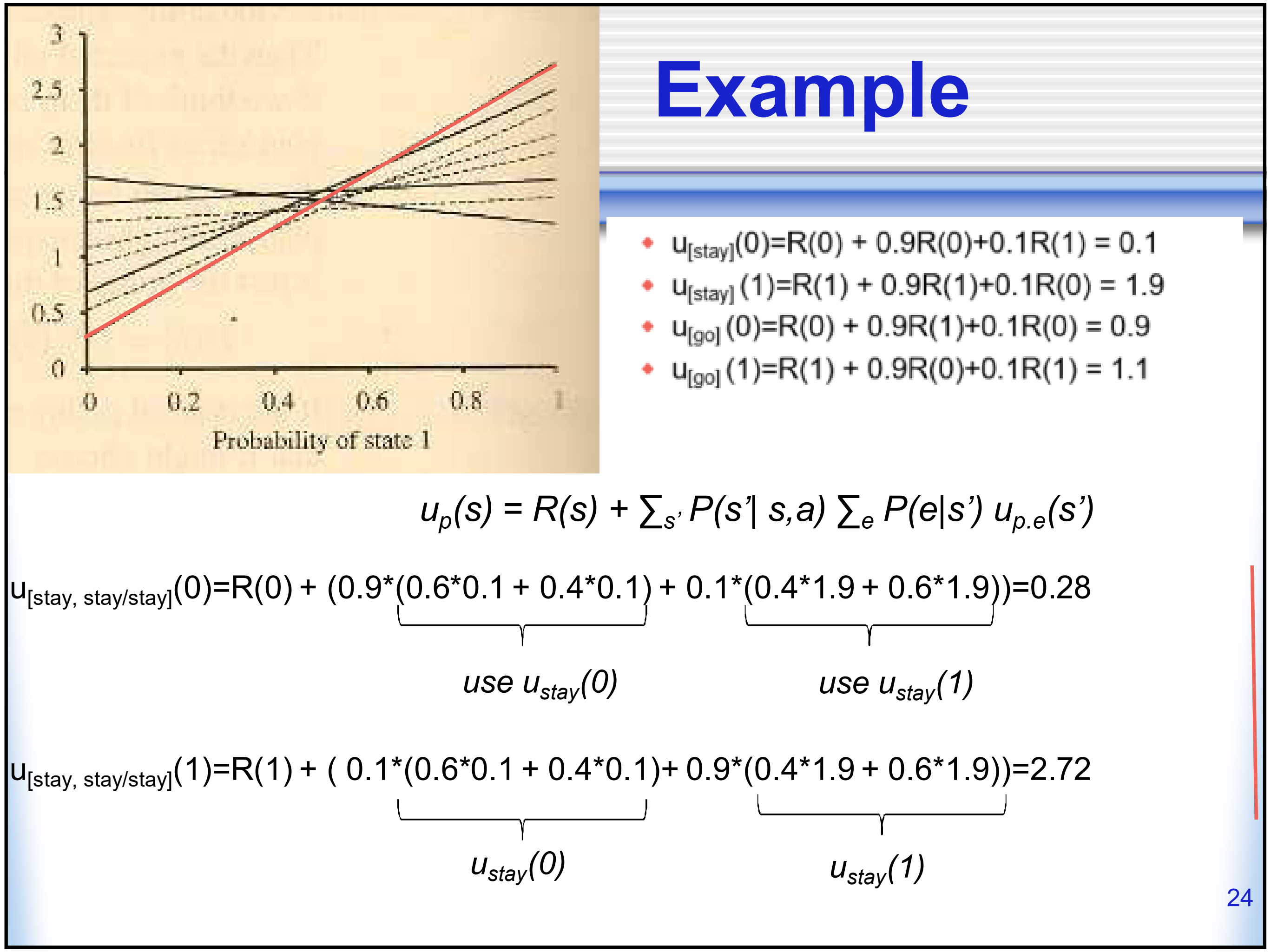
# General formula

We can compute the utilities for conditional plans of depth-2 by considering each possible first action, each possible subsequent percept and then each way of choosing a depth-1 plan to execute for each percept

[Stay; **if** Percept =0 **then** Stay **else** Stay]
[Stay; **if** Percept =0 **then** Stay **else** Go] . . .

- Let *p* be a depth-*d* conditional plan whose initial action is ***a*** and whose depth-*(d-1)* subplan for percept **e** is ***p.e***, then

  $u_p(s) = R(s) + \sum_{s'} P(s'| s,a) \sum_e P(e|s') u_{p.e}(s')$



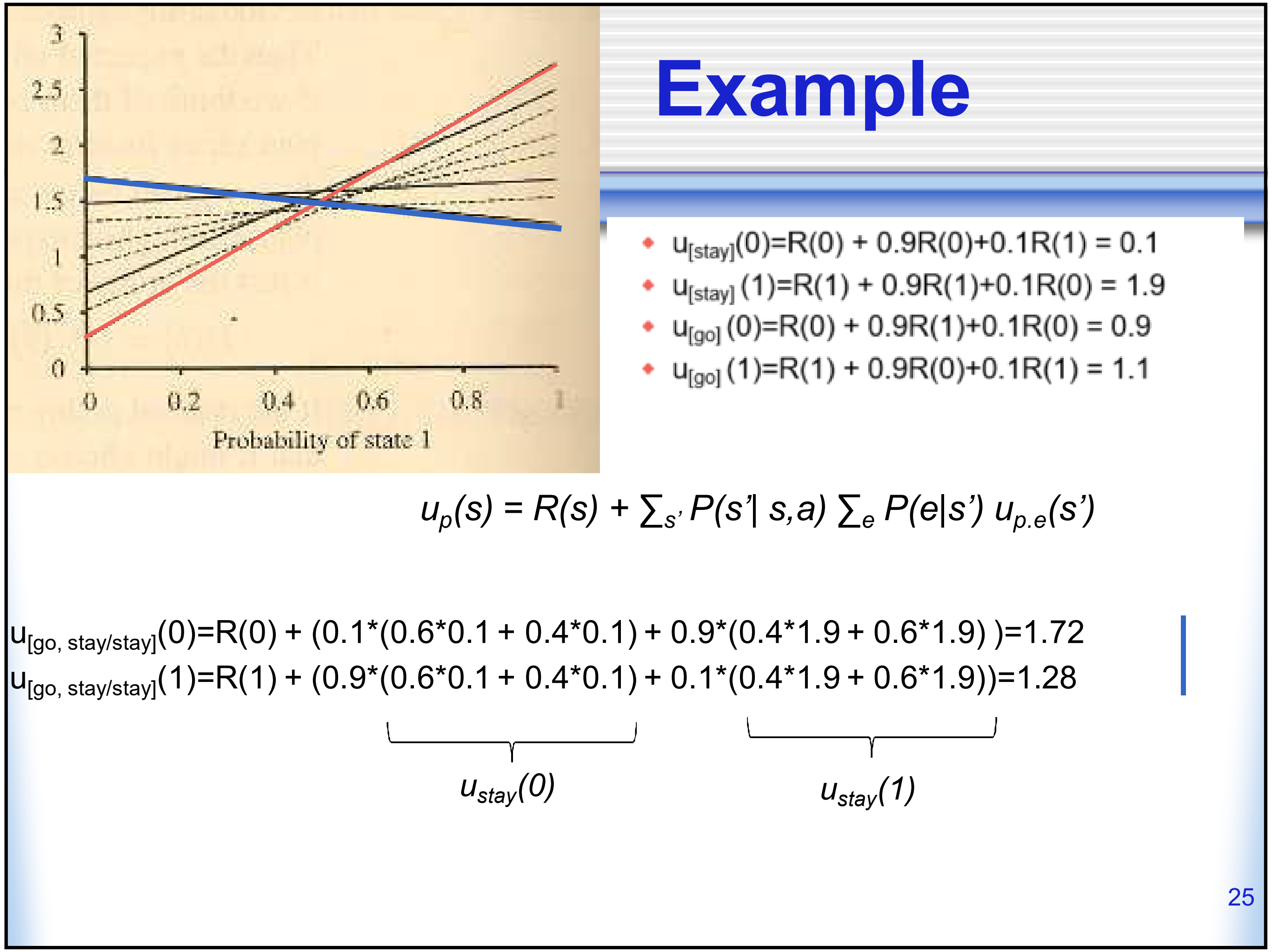
# Example

- $u_{[stay]}(0)=R(0) + 0.9R(0)+0.1R(1) = 0.1$
- $u_{[stay]}(1)=R(1) + 0.9R(1)+0.1R(0) = 1.9$
- $u_{[go]}(0)=R(0) + 0.9R(1)+0.1R(0) = 0.9$
- $u_{[go]}(1)=R(1) + 0.9R(0)+0.1R(1) = 1.1$

$$u_p(s) = R(s) + \sum_{s'} P(s'| s,a) \sum_e P(e|s')\, u_{p.e}(s')$$

$$u_{[stay,\ stay/stay]}(0)=R(0) + (0.9*\underbrace{(0.6*0.1 + 0.4*0.1)}_{\text{use } u_{stay}(0)} + 0.1*\underbrace{(0.4*1.9 + 0.6*1.9)}_{\text{use } u_{stay}(1)})=0.28$$

$$u_{[stay,\ stay/stay]}(1)=R(1) + (\ 0.1*\underbrace{(0.6*0.1 + 0.4*0.1)}_{u_{stay}(0)}+ 0.9*\underbrace{(0.4*1.9 + 0.6*1.9)}_{u_{stay}(1)})=2.72$$



# Example

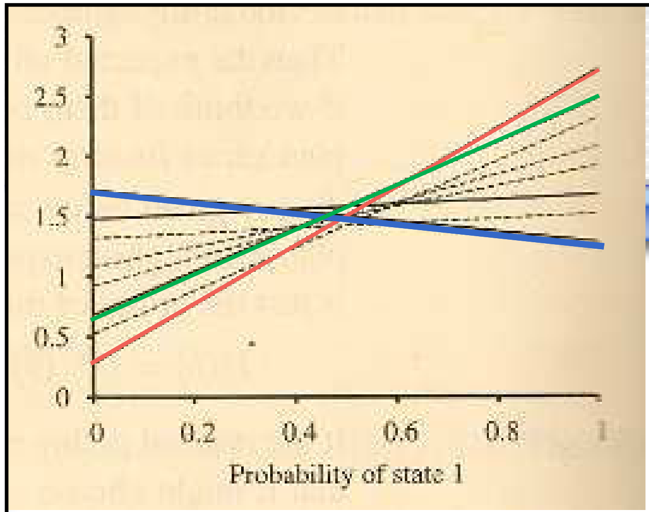
- $u_{[stay]}(0)=R(0) + 0.9R(0)+0.1R(1) = 0.1$
- $u_{[stay]}(1)=R(1) + 0.9R(1)+0.1R(0) = 1.9$
- $u_{[go]}(0)=R(0) + 0.9R(1)+0.1R(0) = 0.9$
- $u_{[go]}(1)=R(1) + 0.9R(0)+0.1R(1) = 1.1$

$$u_p(s) = R(s) + \sum_{s'} P(s'| s,a) \sum_e P(e|s') u_{p.e}(s')$$

$$u_{[go,\ stay/stay]}(0)=R(0) + (0.1*\underbrace{(0.6*0.1 + 0.4*0.1)}_{u_{stay}(0)} + 0.9*(0.4*1.9 + 0.6*1.9) )=1.72$$

$$u_{[go,\ stay/stay]}(1)=R(1) + (0.9*(0.6*0.1 + 0.4*0.1) + 0.1*\underbrace{(0.4*1.9 + 0.6*1.9)}_{u_{stay}(1)})=1.28$$

25



# Example

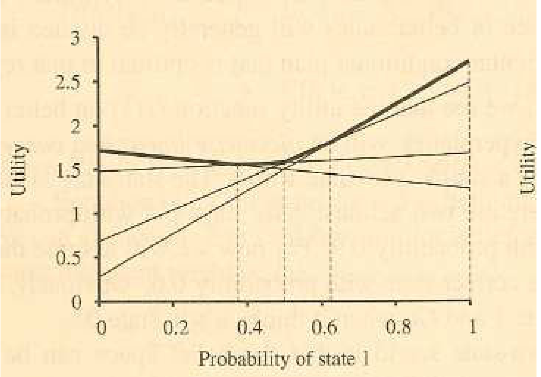
- $u_{[stay]}(0)=R(0) + 0.9R(0)+0.1R(1) = 0.1$
- $u_{[stay]}(1)=R(1) + 0.9R(1)+0.1R(0) = 1.9$
- $u_{[go]}(0)=R(0) + 0.9R(1)+0.1R(0) = 0.9$
- $u_{[go]}(1)=R(1) + 0.9R(0)+0.1R(1) = 1.1$

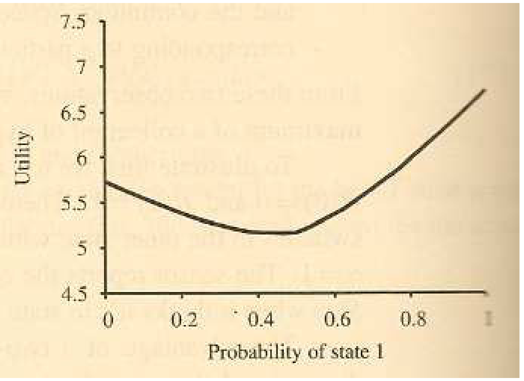
$$u_p(s) = R(s) + \sum_{s'} P(s'| s,a) \sum_e P(e|s')\, u_{p.e}(s')$$

$$u_{[stay,\ go/stay]}(0)= R(0) + (0.9*(0.6*\underset{u_{go}(0)}{0.9} + 0.4*\underset{u_{stay}(0)}{0.1}) + 0.1*(0.4*\underset{u_{go}(1)}{1.1} + 0.6*\underset{u_{stay}(1)}{1.9}))=0.68$$

$$u_{[stay,\ go/stay]}(1)= R(1) + (0.1*(0.6*0.9 + 0.4*0.1) + 0.9*(0.4*1.1+ 0.6*1.9))=2.48$$

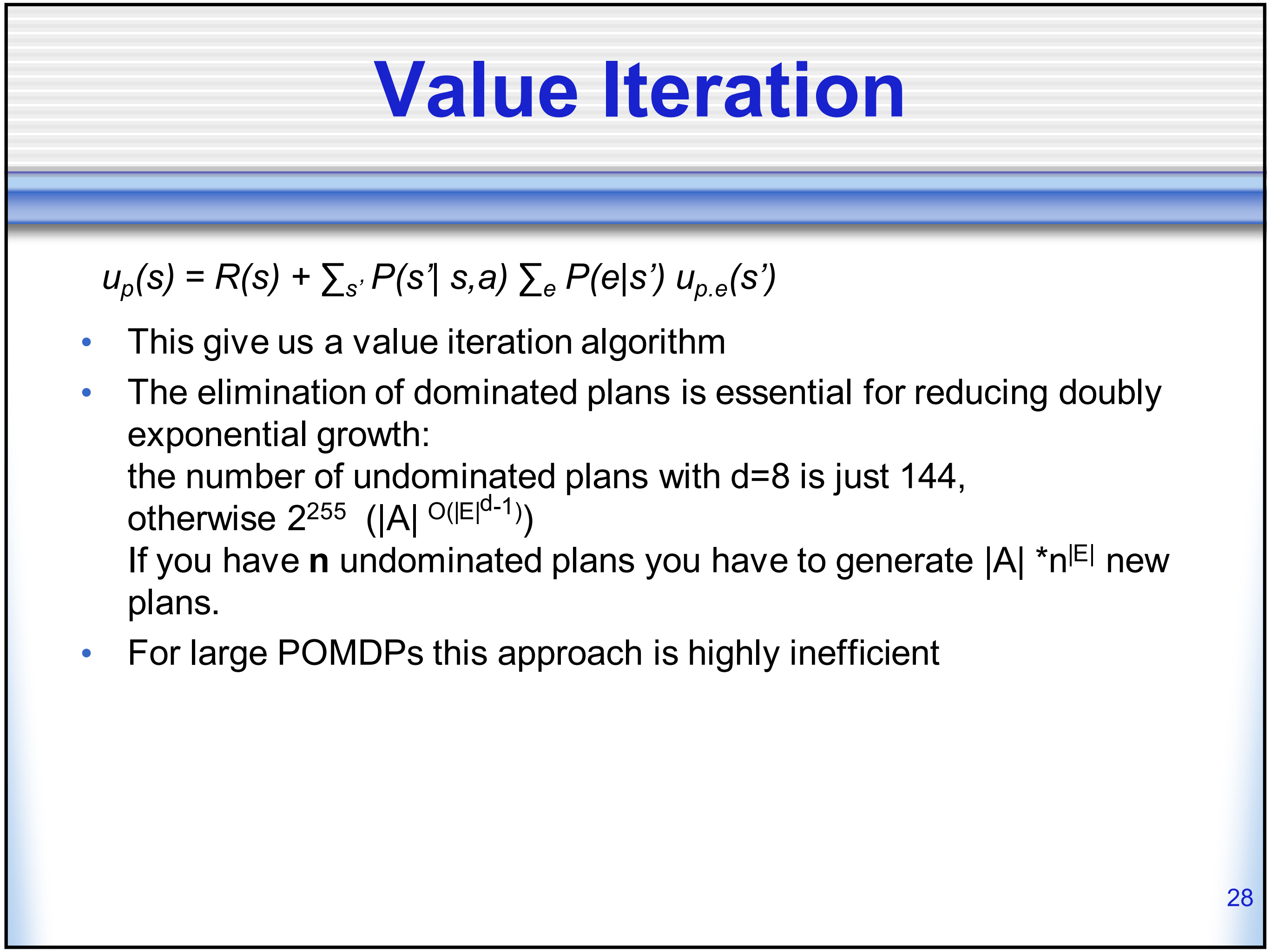
# Example





Utility of four undominated two-step plans

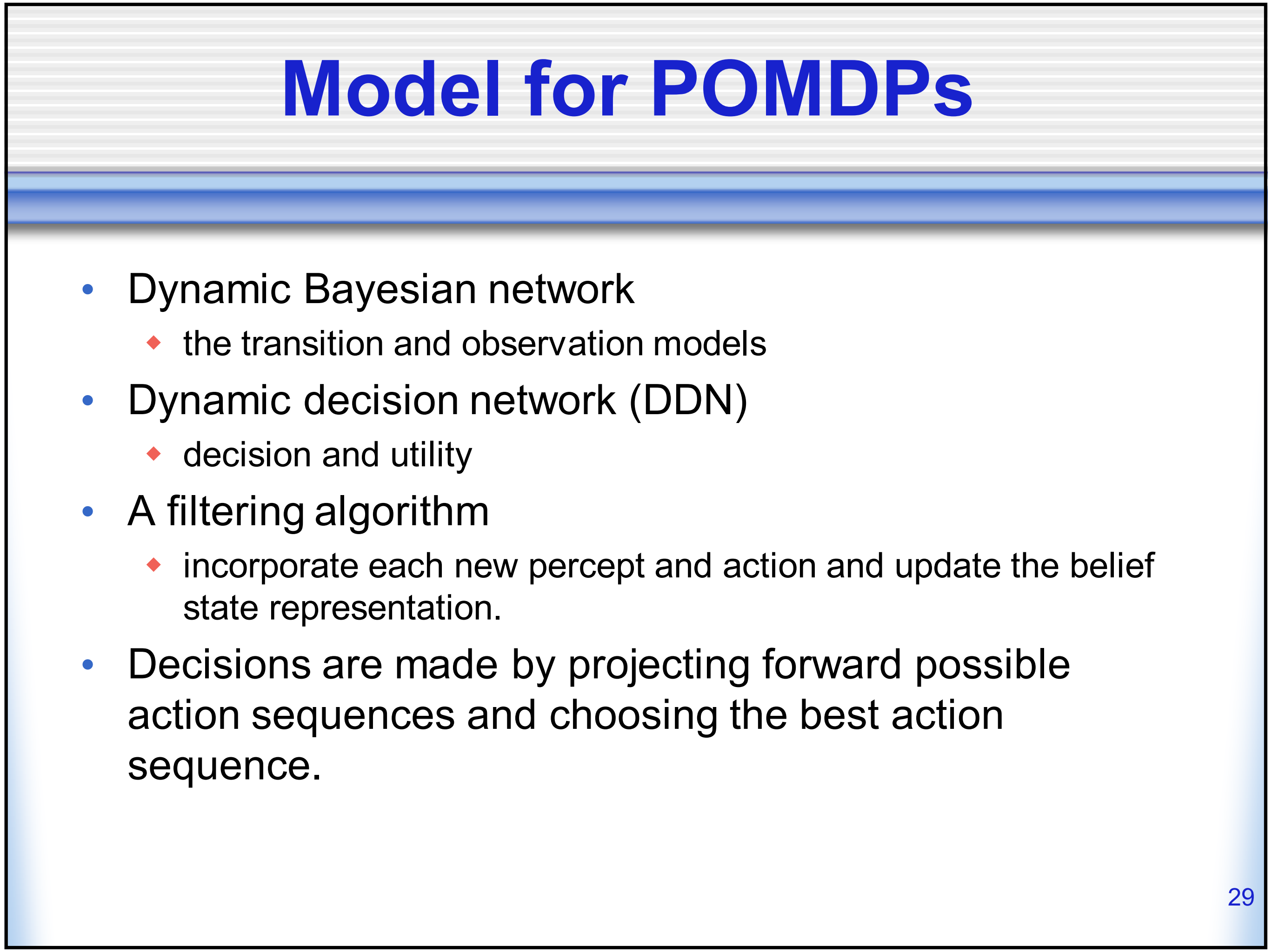
Utility function for optimal eight step plans



# Value Iteration

$u_p(s) = R(s) + \sum_{s'} P(s'| s,a) \sum_e P(e|s') u_{p.e}(s')$

- This give us a value iteration algorithm
- The elimination of dominated plans is essential for reducing doubly exponential growth:
  the number of undominated plans with d=8 is just 144,
  otherwise $2^{255}$ ($|A|^{O(|E|^{d-1})}$)
  If you have **n** undominated plans you have to generate $|A| * n^{|E|}$ new plans.
- For large POMDPs this approach is highly inefficient



# Model for POMDPs

- Dynamic Bayesian network
  - the transition and observation models
- Dynamic decision network (DDN)
  - decision and utility
- A filtering algorithm
  - incorporate each new percept and action and update the belief state representation.
- Decisions are made by projecting forward possible action sequences and choosing the best action sequence.

# The Generic Structure of a Dynamic Decision Network

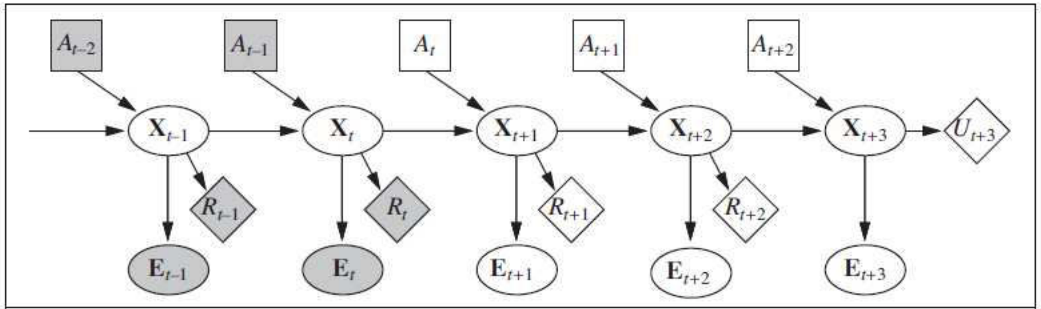
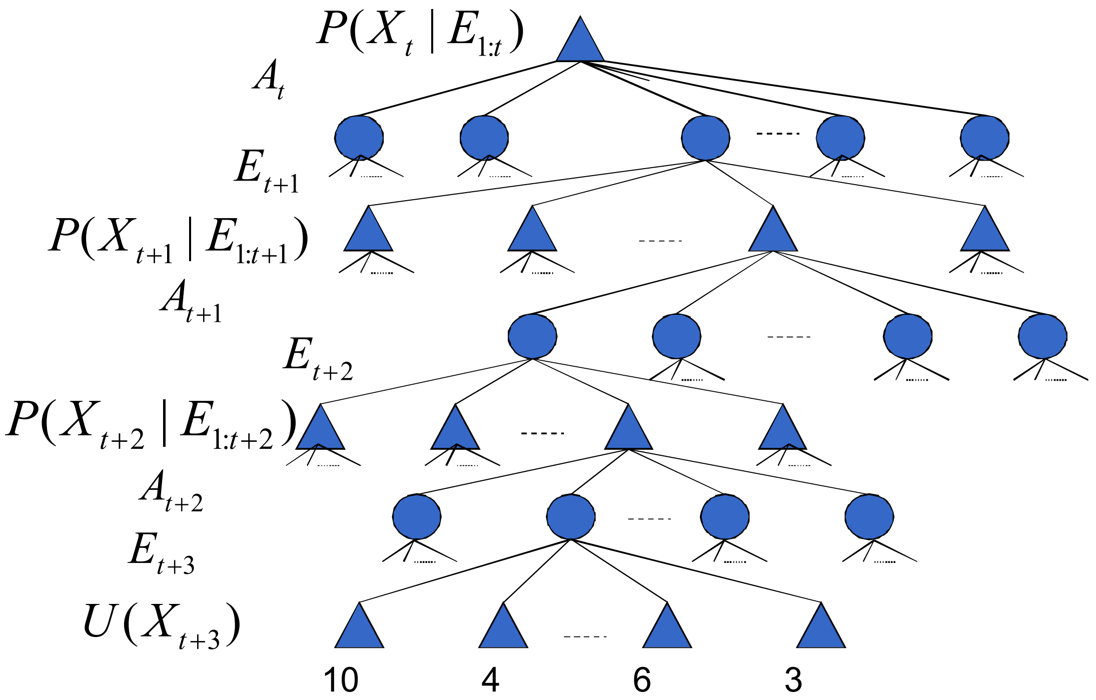


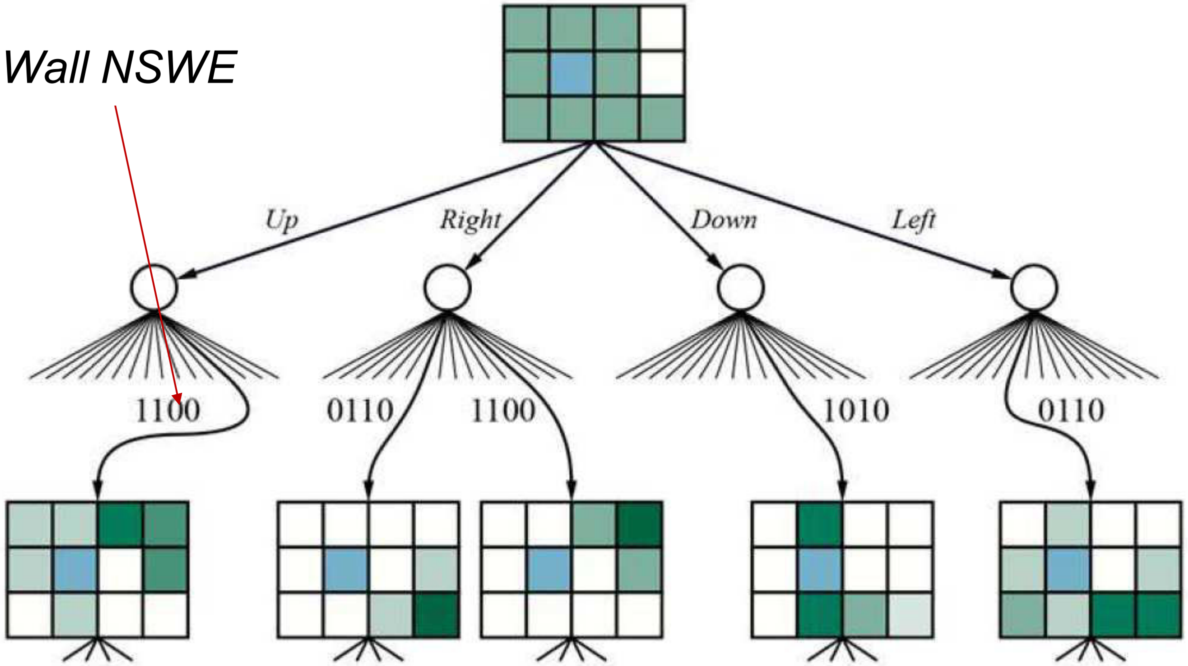
**Figure 17.10** **FILES: figures/generic-ddn.eps (Tue Nov 3 16:22:53 2009).** The generic structure of a dynamic decision network. Variables with known values are shaded. The current time is $t$ and the agent must decide what to do—that is, choose a value for $A_t$. The network has been unrolled into the future for three steps and represents future rewards, as well as the utility of the state at the look-ahead horizon.

- The decision problem involves calculating the value of $A_t$ that maximizes the agent's expected utility over the remaining state sequence.

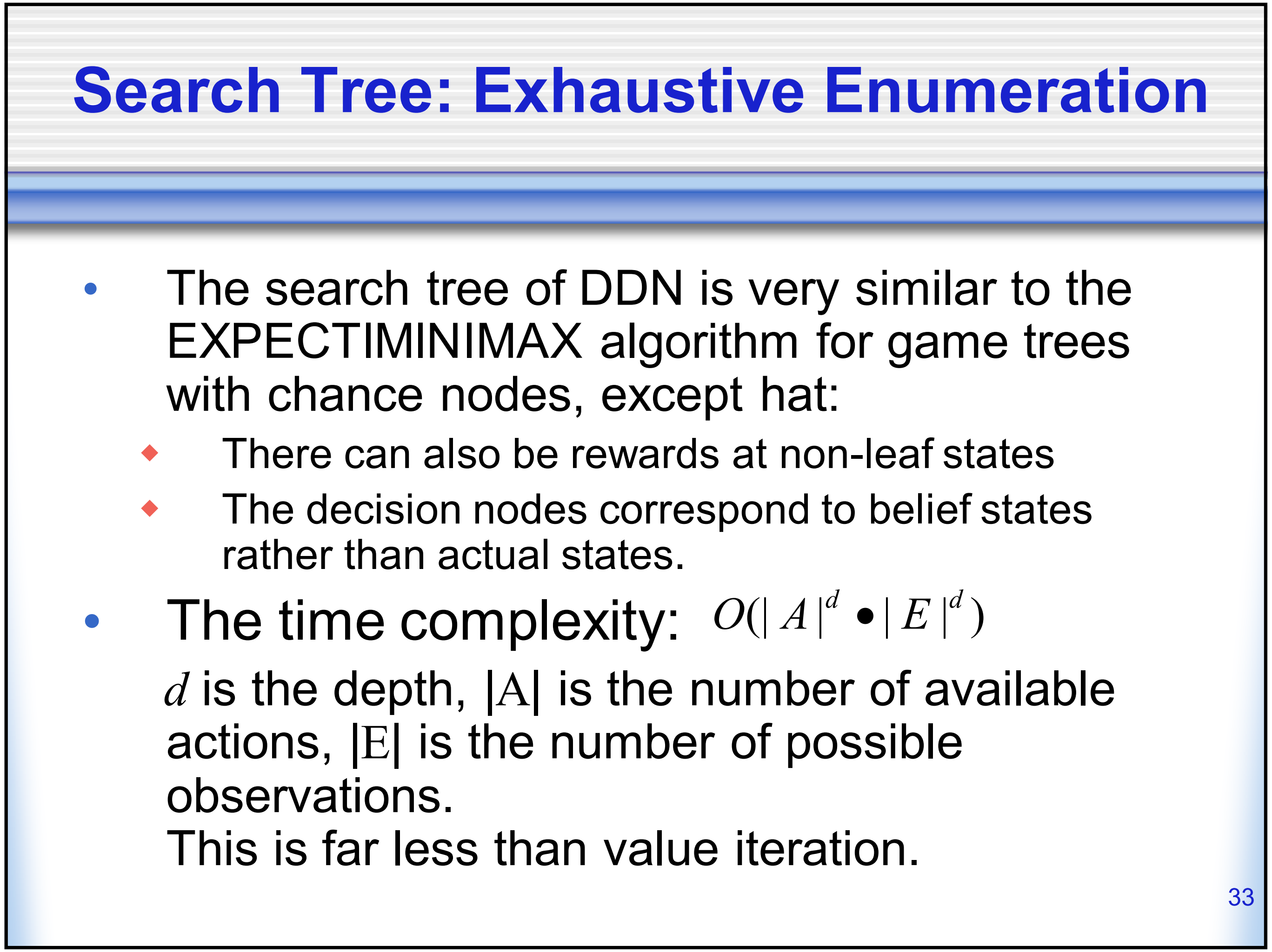
# Search Tree of the Lookahead DDN



# Search Tree of the Lookahead DDN



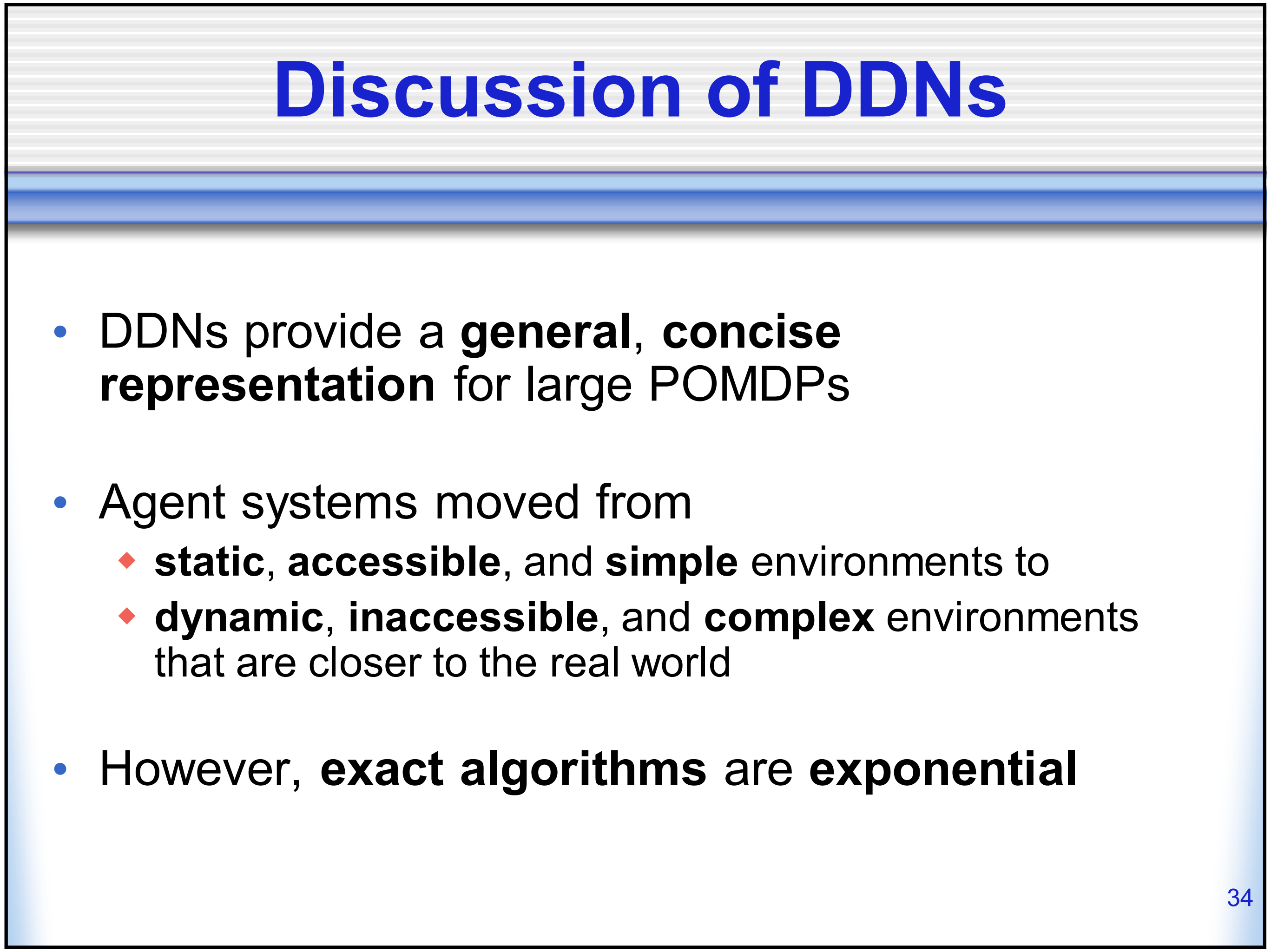
Wall NSWE



# Search Tree: Exhaustive Enumeration

- The search tree of DDN is very similar to the EXPECTIMINIMAX algorithm for game trees with chance nodes, except hat:
  - There can also be rewards at non-leaf states
  - The decision nodes correspond to belief states rather than actual states.
- The time complexity: $O(|A|^d \bullet |E|^d)$

  $d$ is the depth, $|A|$ is the number of available actions, $|E|$ is the number of possible observations.
  This is far less than value iteration.



# Discussion of DDNs

- DDNs provide a **general**, **concise representation** for large POMDPs

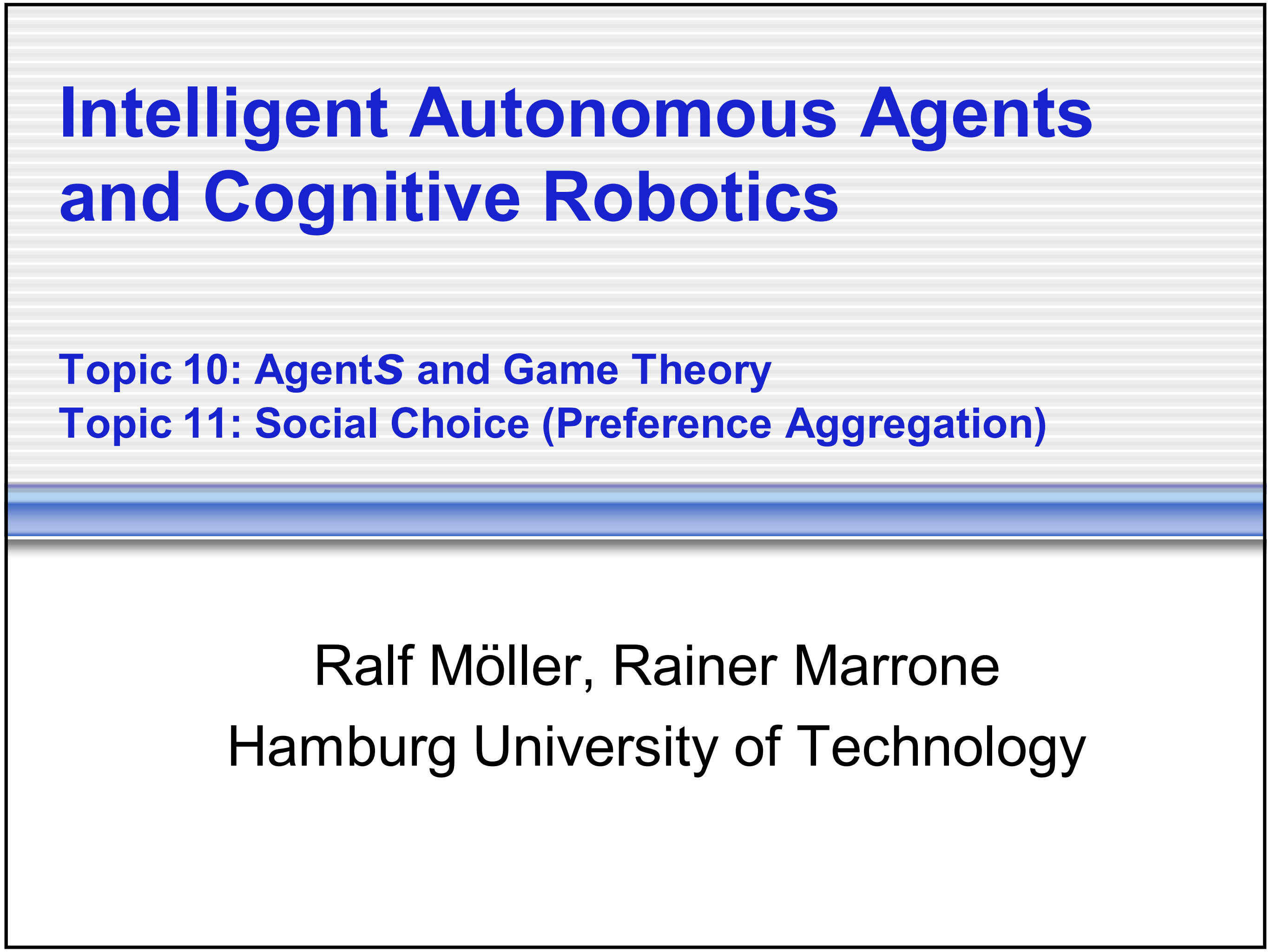
- Agent systems moved from
  - **static**, **accessible**, and **simple** environments to
  - **dynamic**, **inaccessible**, and **complex** environments that are closer to the real world

- However, **exact algorithms** are **exponential**

# Perspectives of DDNs to Reduce Complexity

- **Heuristic estimate** for
  the utility of the remaining steps
- Incremental **pruning** techniques
- Many **approximation** techniques as in our search lecture:
  - Using less detailed state variables for states in the distant future.
  - Using a greedy heuristic search through the space of decision sequences.

…



# Intelligent Autonomous Agents and Cognitive Robotics

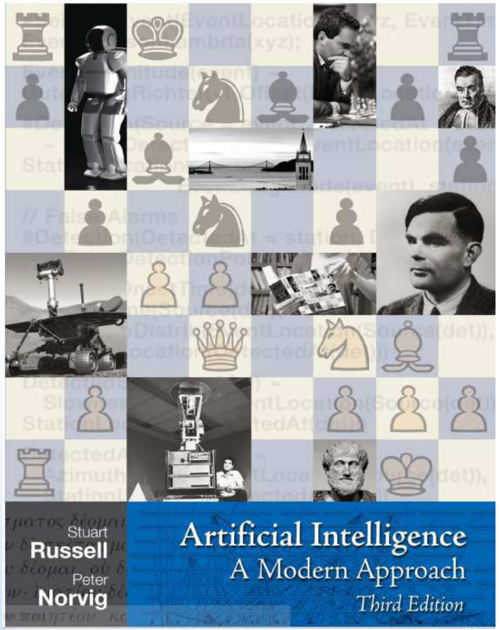
## Topic 10: Agent*S* and Game Theory
## Topic 11: Social Choice (Preference Aggregation)

Ralf Möller, Rainer Marrone

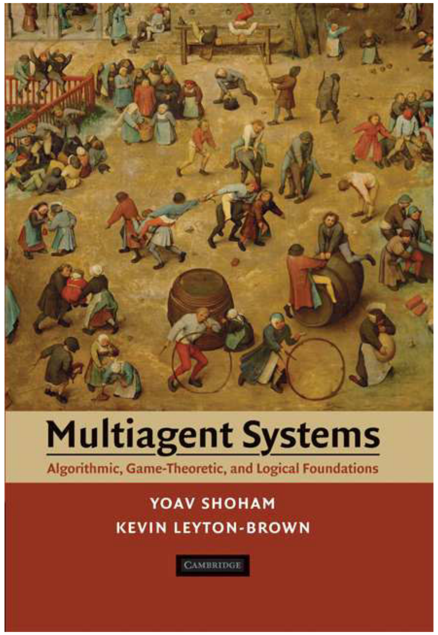
Hamburg University of Technology

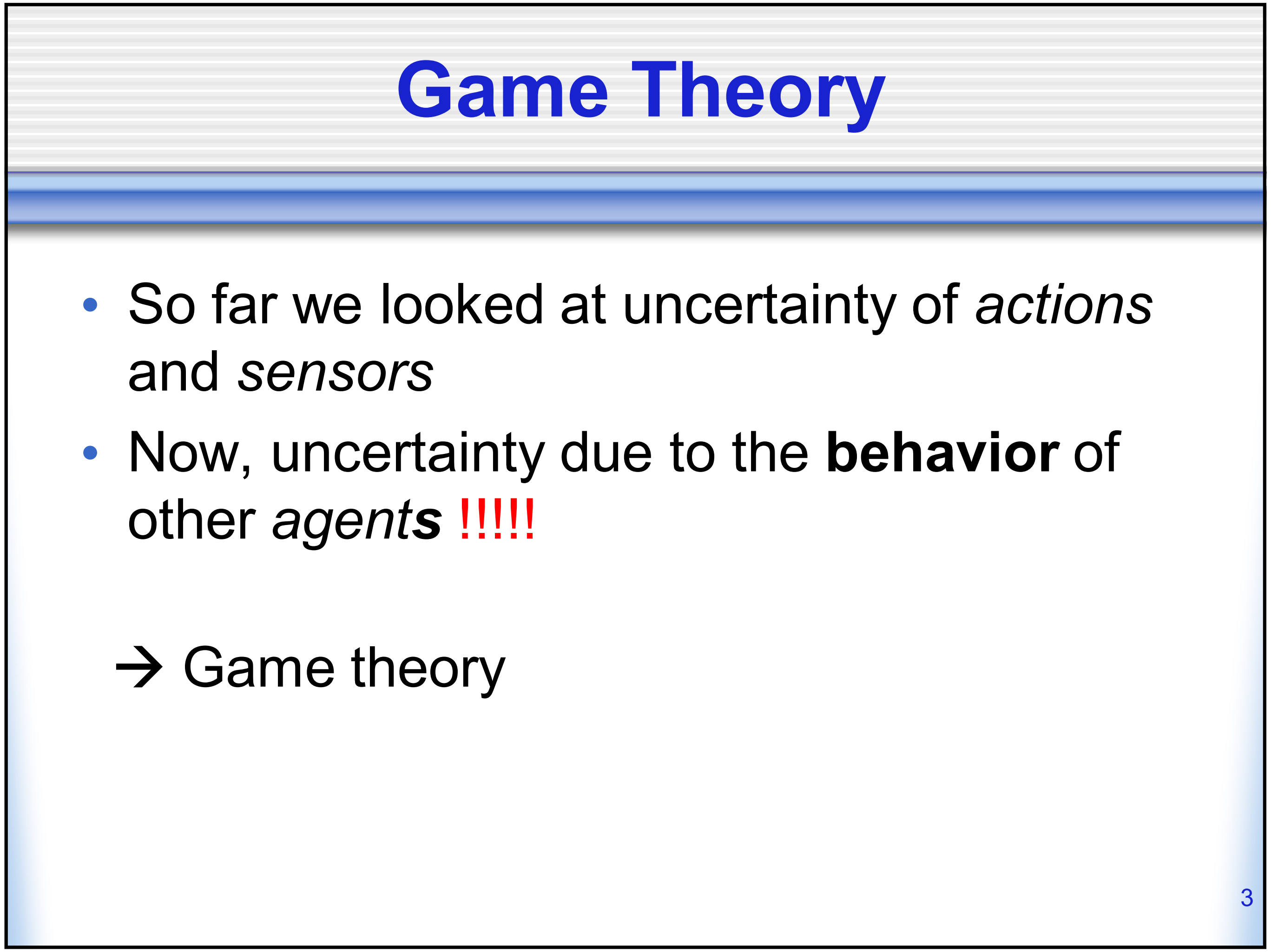
# Literature

- Chapter 17



- Chapter 3

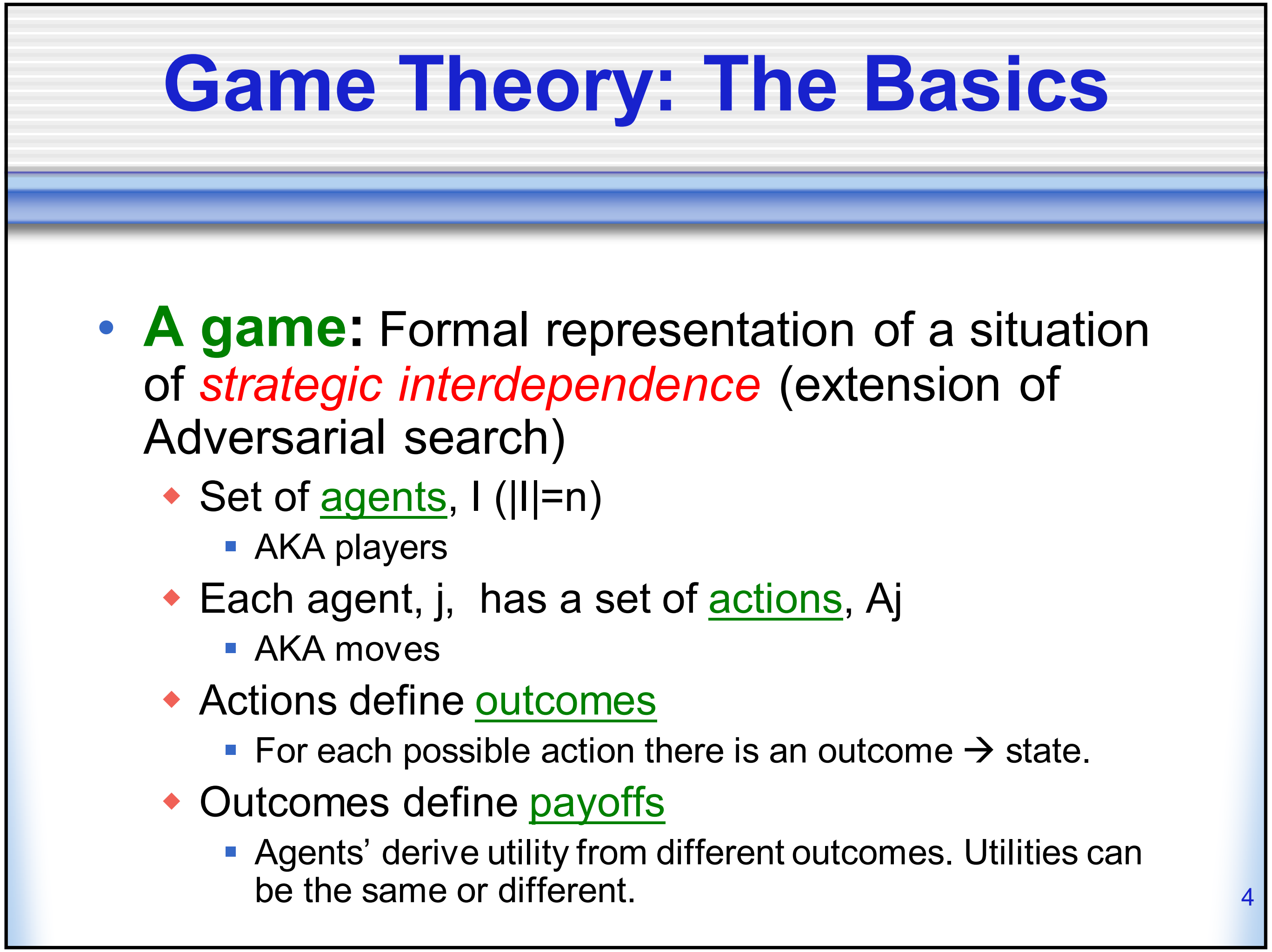




# Game Theory

- So far we looked at uncertainty of *actions* and *sensors*
- Now, uncertainty due to the **behavior** of other *agent**s*** !!!!!

→ Game theory



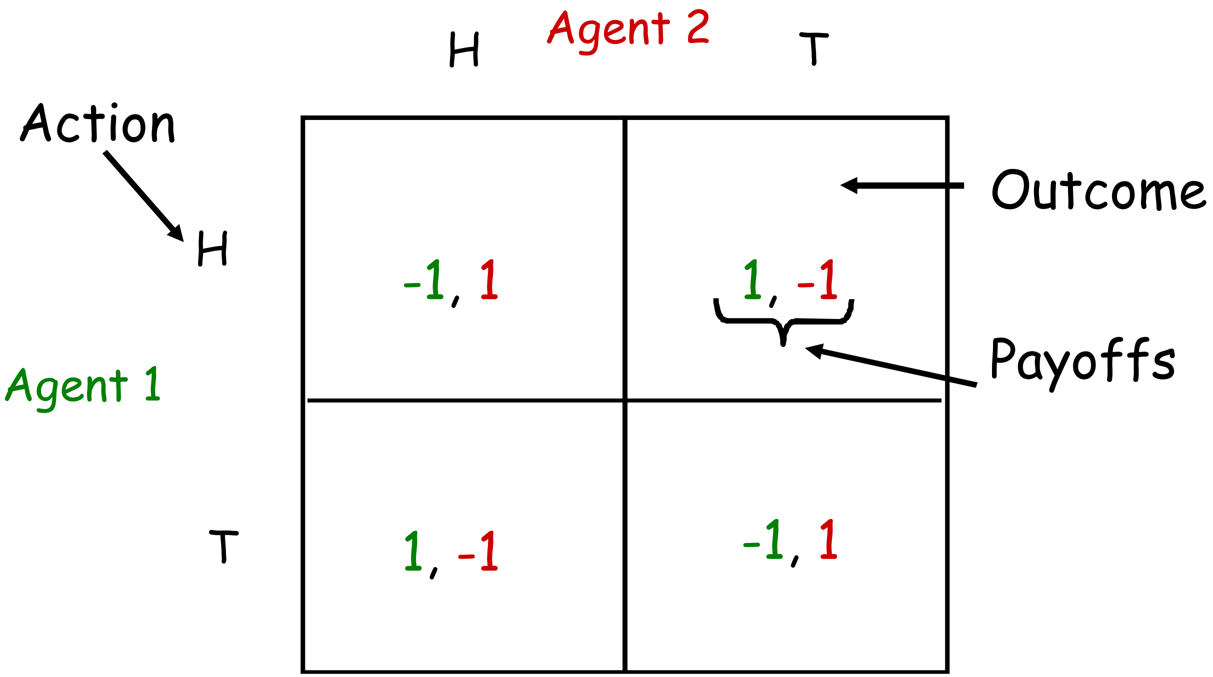
# Game Theory: The Basics

- **A game:** Formal representation of a situation of *strategic interdependence* (extension of Adversarial search)
  - Set of agents, I (|I|=n)
    - AKA players
  - Each agent, j, has a set of actions, Aj
    - AKA moves
  - Actions define outcomes
    - For each possible action there is an outcome → state.
  - Outcomes define payoffs
    - Agents’ derive utility from different outcomes. Utilities can be the same or different.

4

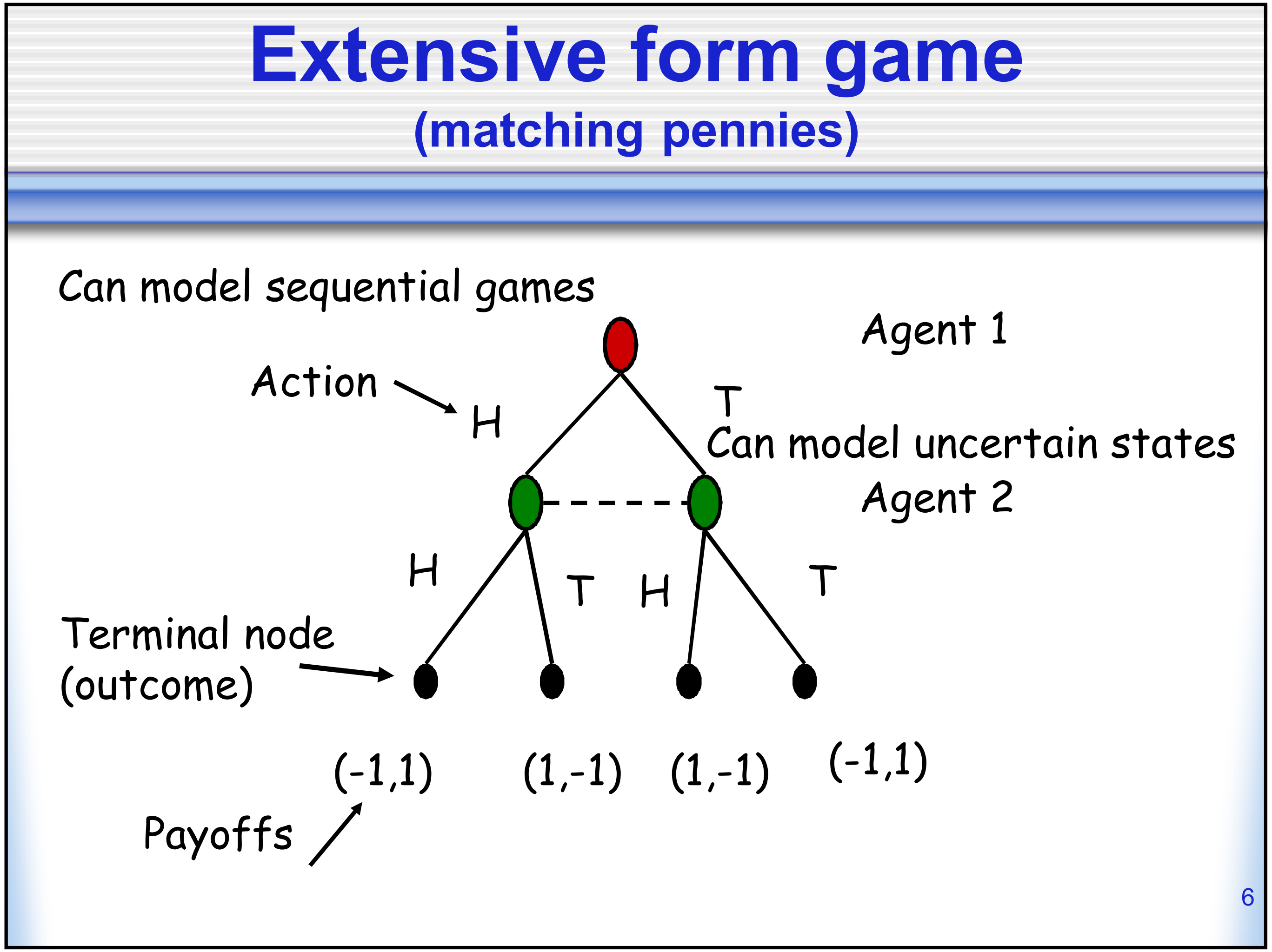
# Normal form game*

## (matching pennies)



| | | Agent 2 | |
|---|---|---|---|
| | | H | T |
| Agent 1 | H | -1, 1 | 1, -1 |
| | T | 1, -1 | -1, 1 |

*aka strategic form, matrix form

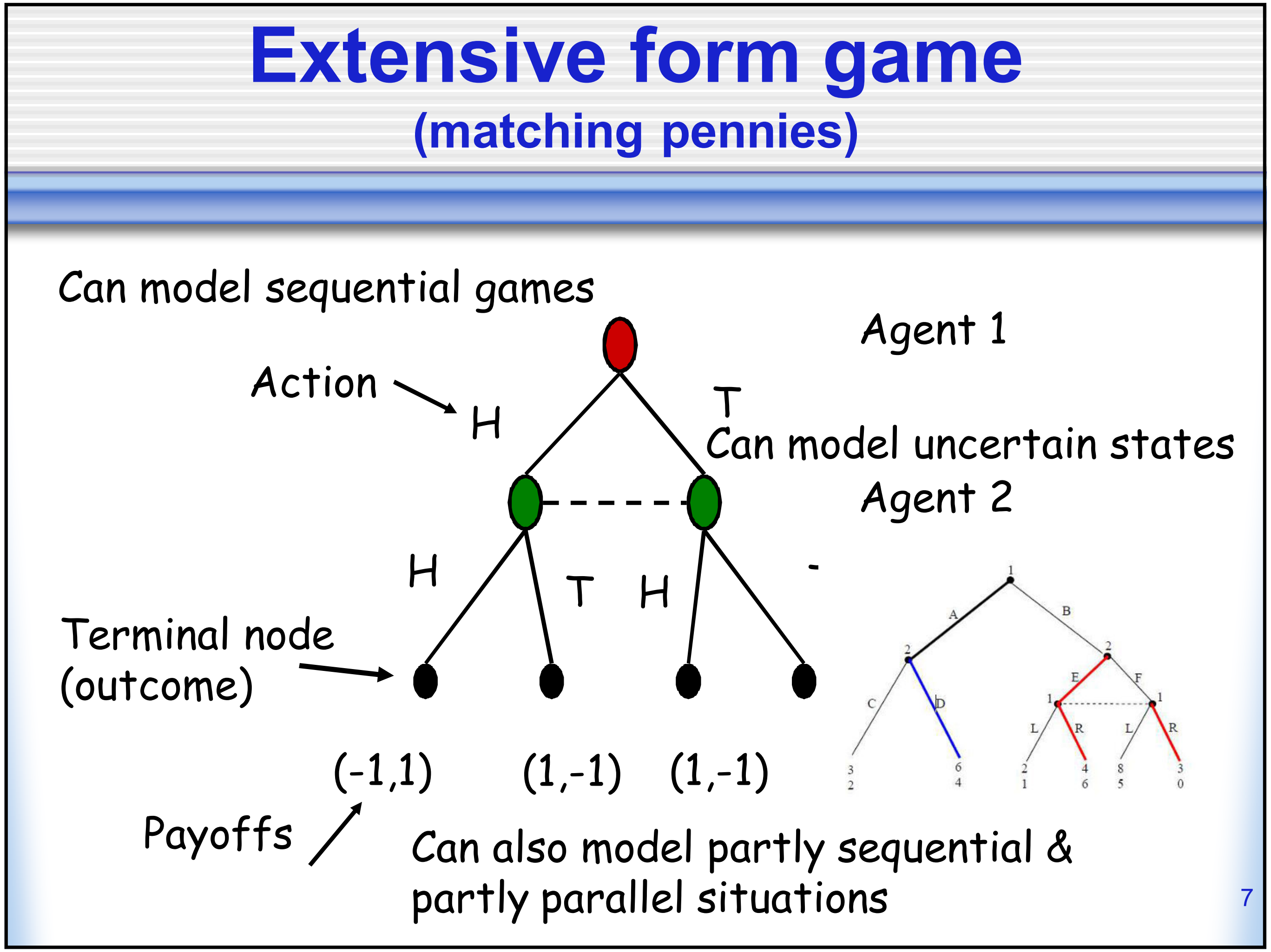


# Extensive form game

## (matching pennies)

Can model sequential games

Can model uncertain states



# Extensive form game

## (matching pennies)

Can model sequential games

Agent 1

Action

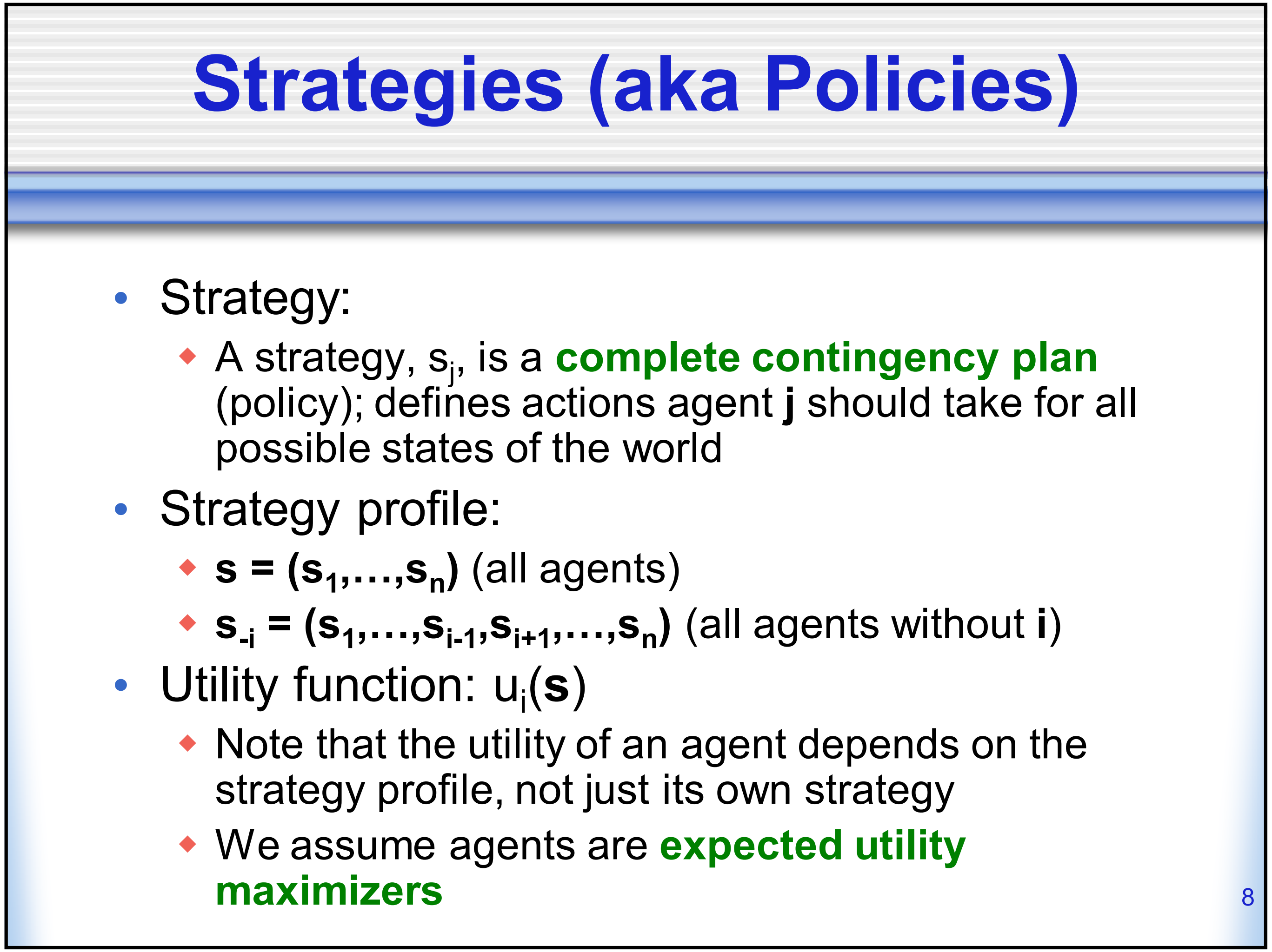
Can model uncertain states

Agent 2

Terminal node (outcome)

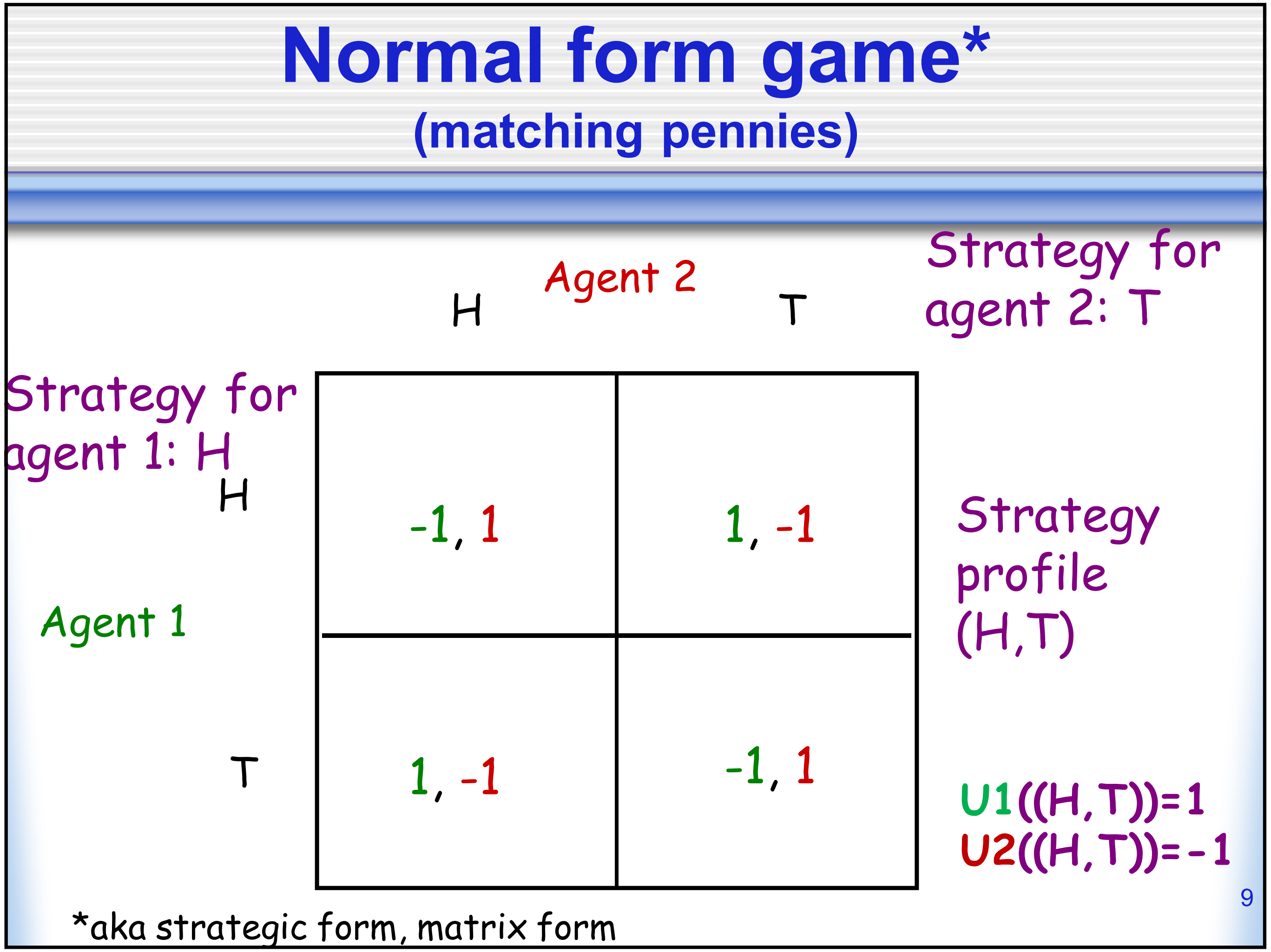
Payoffs

Can also model partly sequential & partly parallel situations



# Strategies (aka Policies)

- Strategy:
  - A strategy, $s_j$, is a **complete contingency plan** (policy); defines actions agent **j** should take for all possible states of the world
- Strategy profile:
  - $\mathbf{s} = (s_1,\ldots,s_n)$ (all agents)
  - $\mathbf{s}_{-i} = (s_1,\ldots,s_{i-1},s_{i+1},\ldots,s_n)$ (all agents without **i**)
- Utility function: $u_i(\mathbf{s})$
  - Note that the utility of an agent depends on the strategy profile, not just its own strategy
  - We assume agents are **expected utility maximizers**



# Normal form game*

## (matching pennies)

Strategy for agent 2: T

Agent 2

|  |  | H | T |
|---|---|---|---|
| Agent 1 | H | -1, 1 | 1, -1 |
|  | T | 1, -1 | -1, 1 |

Strategy for agent 1: H

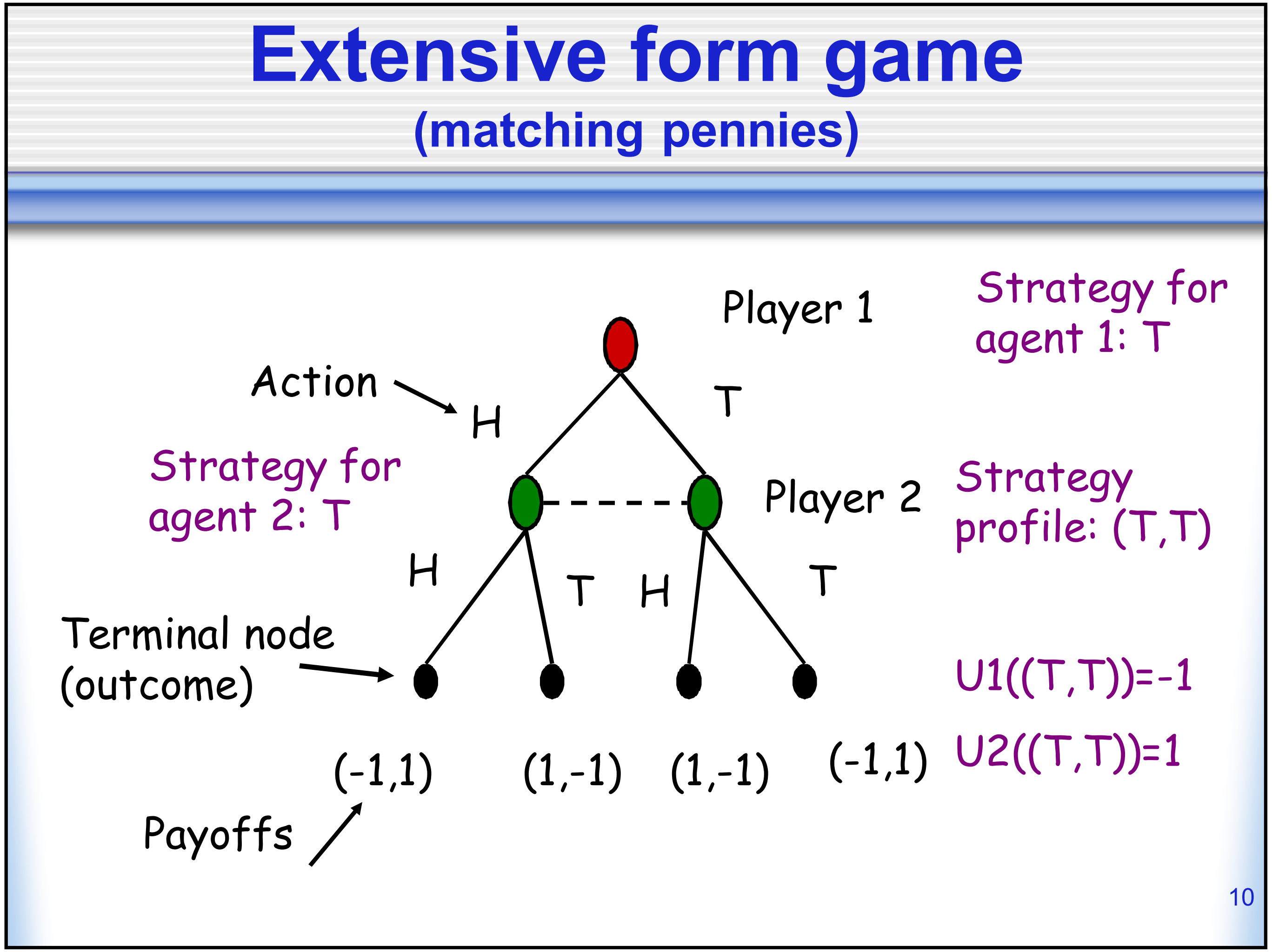
Strategy profile (H,T)

U1((H,T))=1
U2((H,T))=-1

*aka strategic form, matrix form

# Extensive form game

## (matching pennies)



Player 1

Strategy for agent 1: T

Action

Strategy for agent 2: T

Player 2

Strategy profile: (T,T)

Terminal node (outcome)

U1((T,T))=-1

(-1,1) (1,-1) (1,-1) (-1,1)

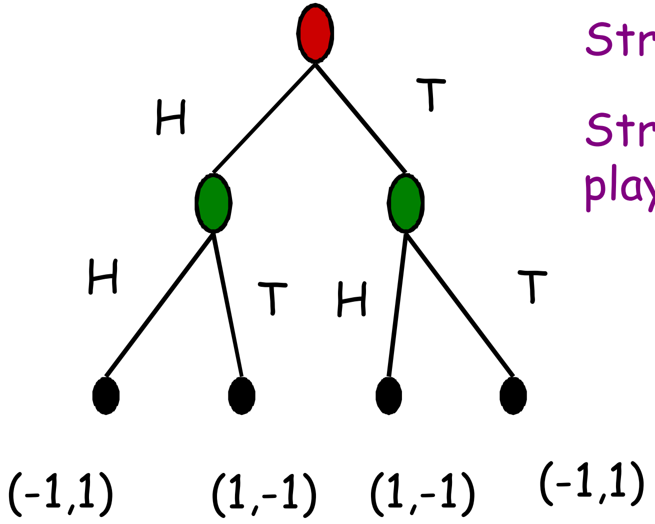
U2((T,T))=1

Payoffs

# Extensive form game

## (matching pennies, seq moves)

Recall: A strategy is a contingency plan for all states of the game



Strategy for agent 1: T

Strategy for agent 2: H if 1 plays H, T if 1 plays T (H,T)

Strategy profile: (T,(H,T))

U1((T,(H,T)))=-1

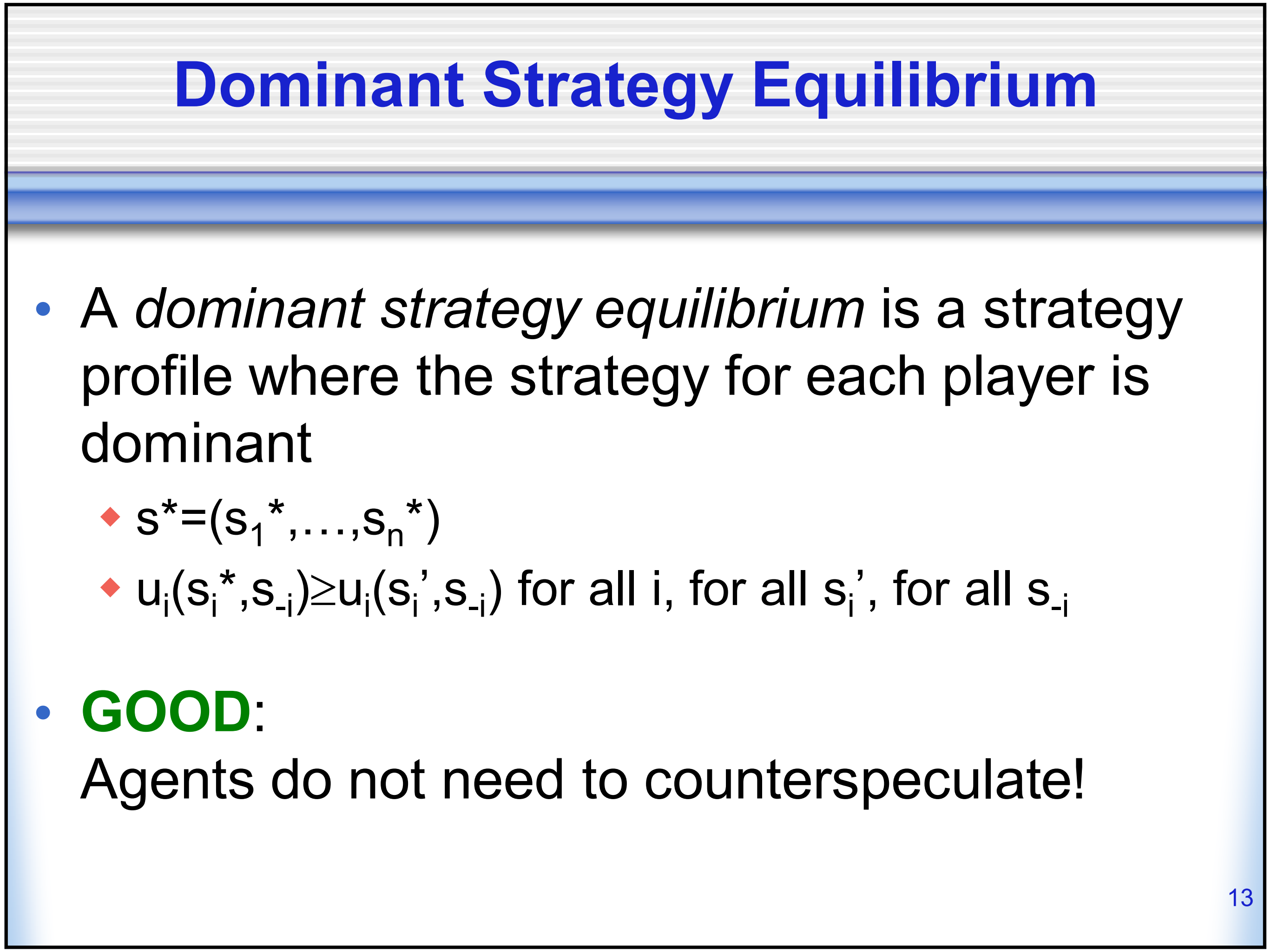
U2((T,(H,T)))=1

# Dominant Strategies

- Recall that
  - Agents' utilities depend on what strategies other agents are playing
  - Agents' are expected utility maximizers
- Agents' will play best-response strategies for $s_{-i}$

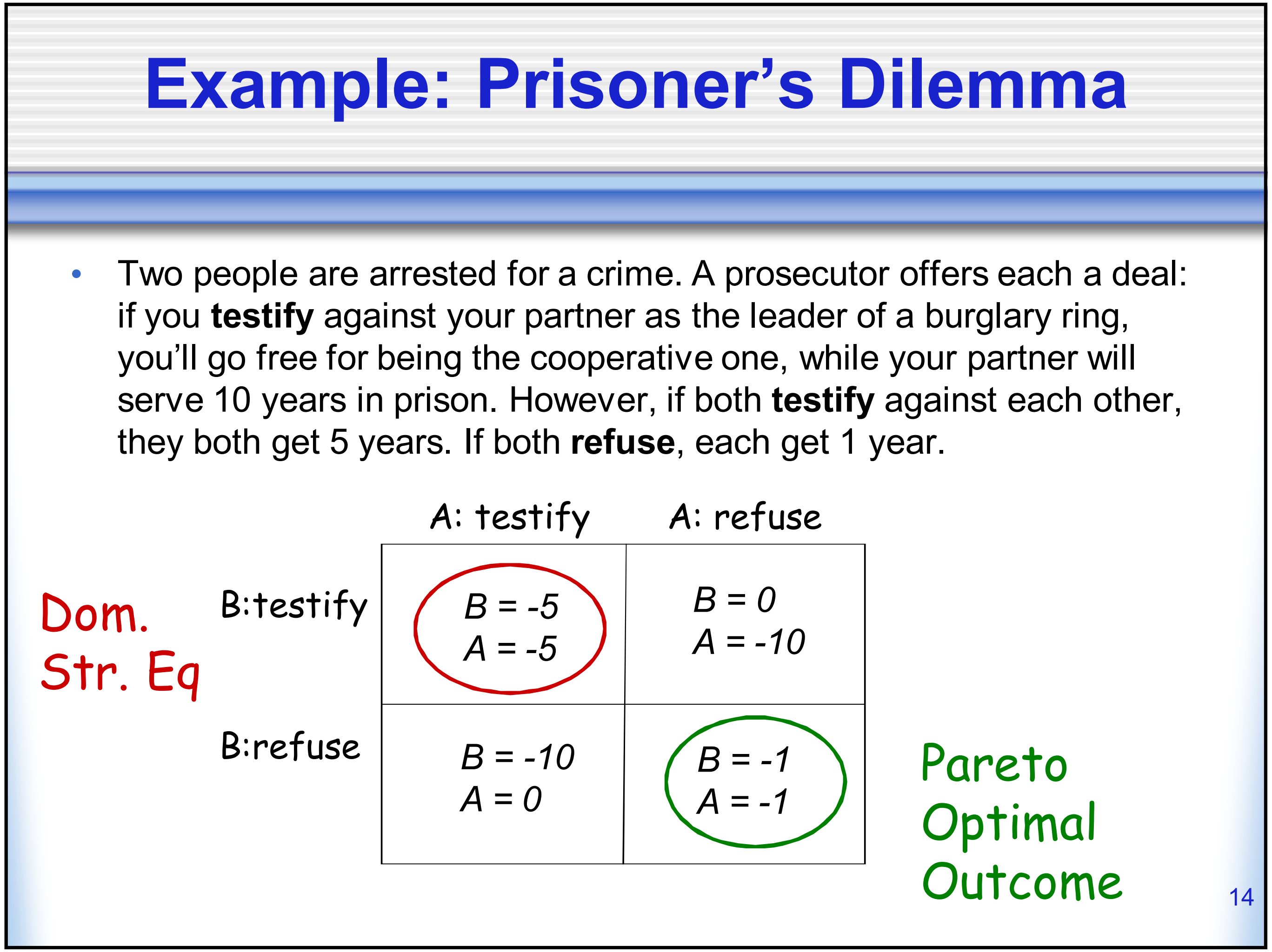
$s_i^*$ is a best response if $u_i(s_i^*, s_{-i}) \geq u_i(s_i', s_{-i})$ for all $s_i'$

- A dominant strategy is a best-response for all $s_{-i}$
  - They do not always exist
  - Inferior strategies are called dominated



## Dominant Strategy Equilibrium

- A *dominant strategy equilibrium* is a strategy profile where the strategy for each player is dominant
  - $s^*=(s_1^*,\ldots,s_n^*)$
  - $u_i(s_i^*,s_{-i})\geq u_i(s_i',s_{-i})$ for all i, for all $s_i'$, for all $s_{-i}$
- **GOOD**:
  Agents do not need to counterspeculate!



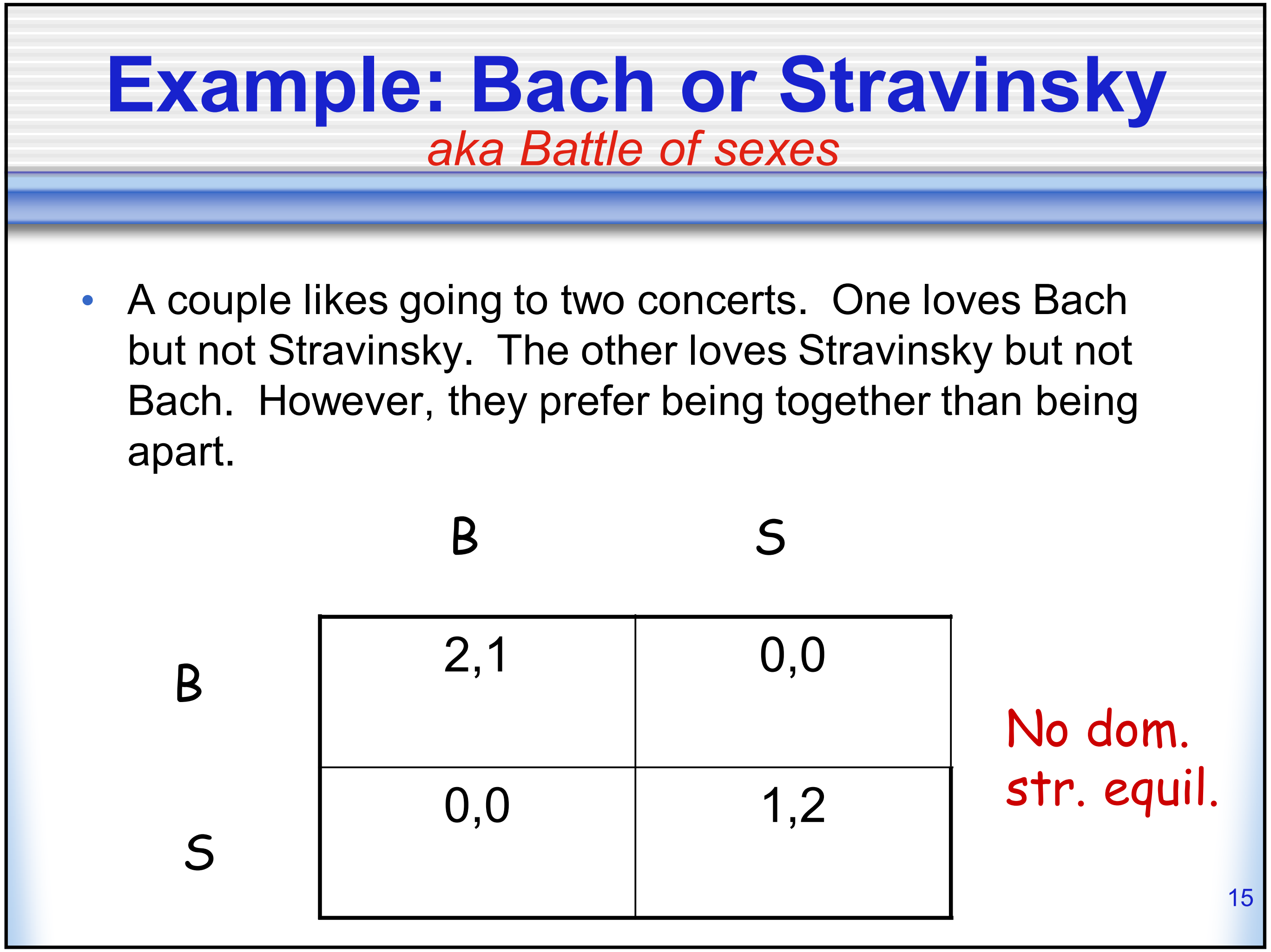
# Example: Prisoner’s Dilemma

- Two people are arrested for a crime. A prosecutor offers each a deal: if you **testify** against your partner as the leader of a burglary ring, you’ll go free for being the cooperative one, while your partner will serve 10 years in prison. However, if both **testify** against each other, they both get 5 years. If both **refuse**, each get 1 year.

| | A: testify | A: refuse |
|---|---|---|
| B:testify | B = -5<br>A = -5 | B = 0<br>A = -10 |
| B:refuse | B = -10<br>A = 0 | B = -1<br>A = -1 |

Dom. Str. Eq

Pareto Optimal Outcome



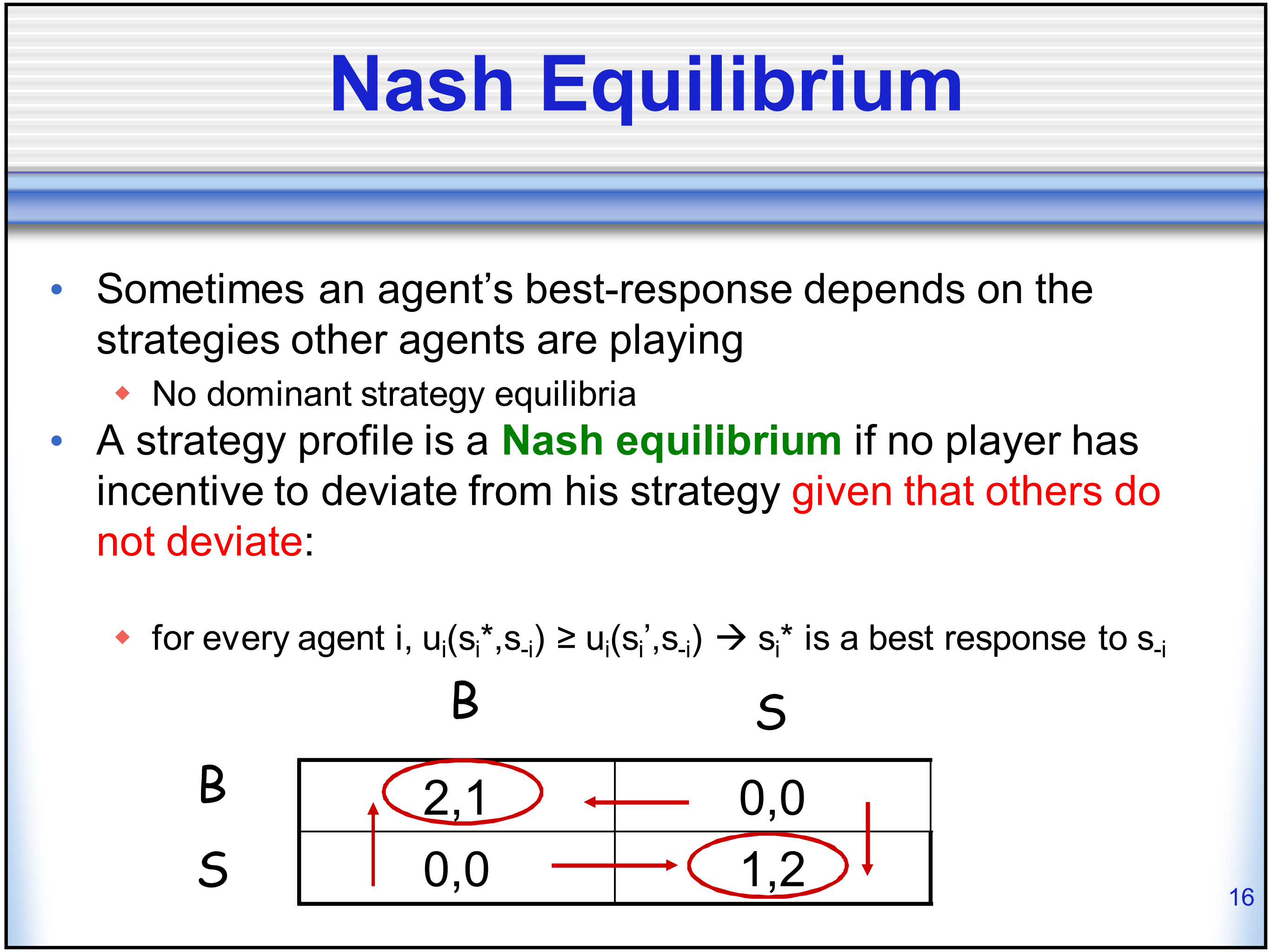
# Example: Bach or Stravinsky

*aka Battle of sexes*

- A couple likes going to two concerts. One loves Bach but not Stravinsky. The other loves Stravinsky but not Bach. However, they prefer being together than being apart.

|  | B | S |
|---|---|---|
| B | 2,1 | 0,0 |
| S | 0,0 | 1,2 |

No dom.
str. equil.

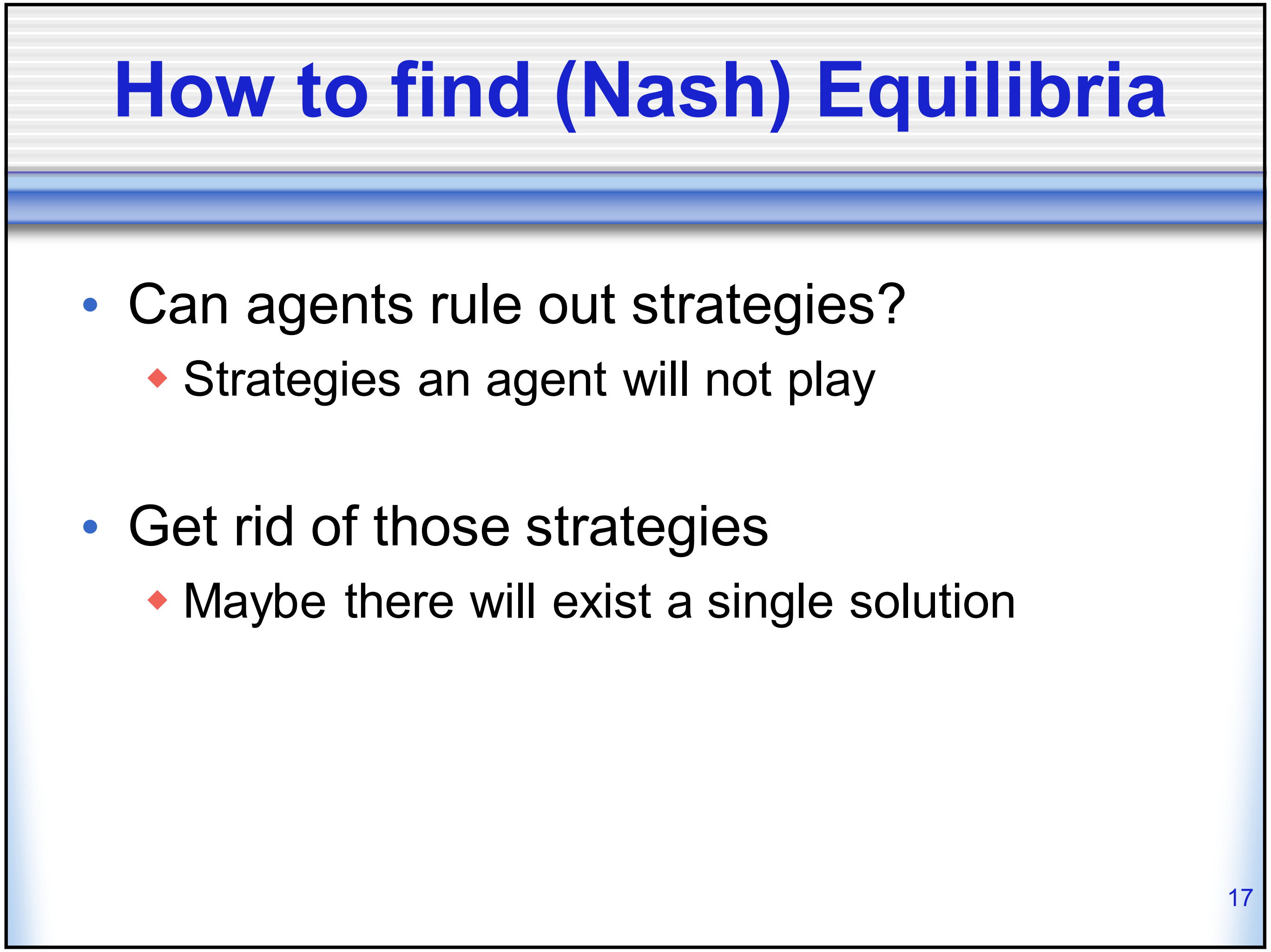


# Nash Equilibrium

- Sometimes an agent’s best-response depends on the strategies other agents are playing
  - No dominant strategy equilibria
- A strategy profile is a **Nash equilibrium** if no player has incentive to deviate from his strategy given that others do not deviate:
  - for every agent i, $u_i(s_i^*,s_{-i}) \geq u_i(s_i',s_{-i})$ → $s_i^*$ is a best response to $s_{-i}$

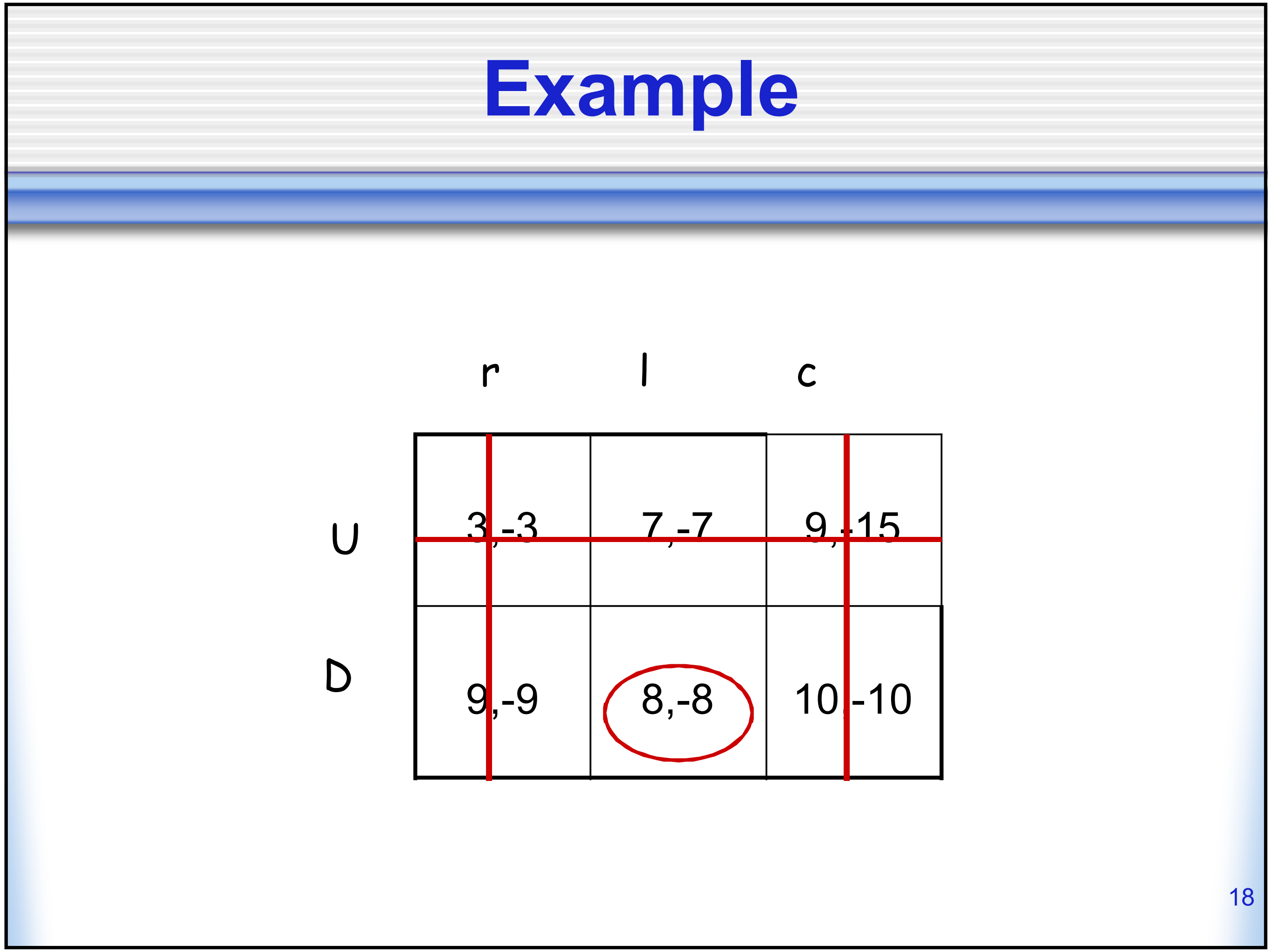
| | B | S |
|---|---|---|
| B | 2,1 | 0,0 |
| S | 0,0 | 1,2 |

16



# How to find (Nash) Equilibria

- Can agents rule out strategies?
  - Strategies an agent will not play
- Get rid of those strategies
  - Maybe there will exist a single solution



# Example

|   | r | l | c |
|---|---|---|---|
| U | ~~3,-3~~ | ~~7,-7~~ | ~~9,-15~~ |
| D | ~~9,-9~~ | 8,-8 | ~~10,-10~~ |

18

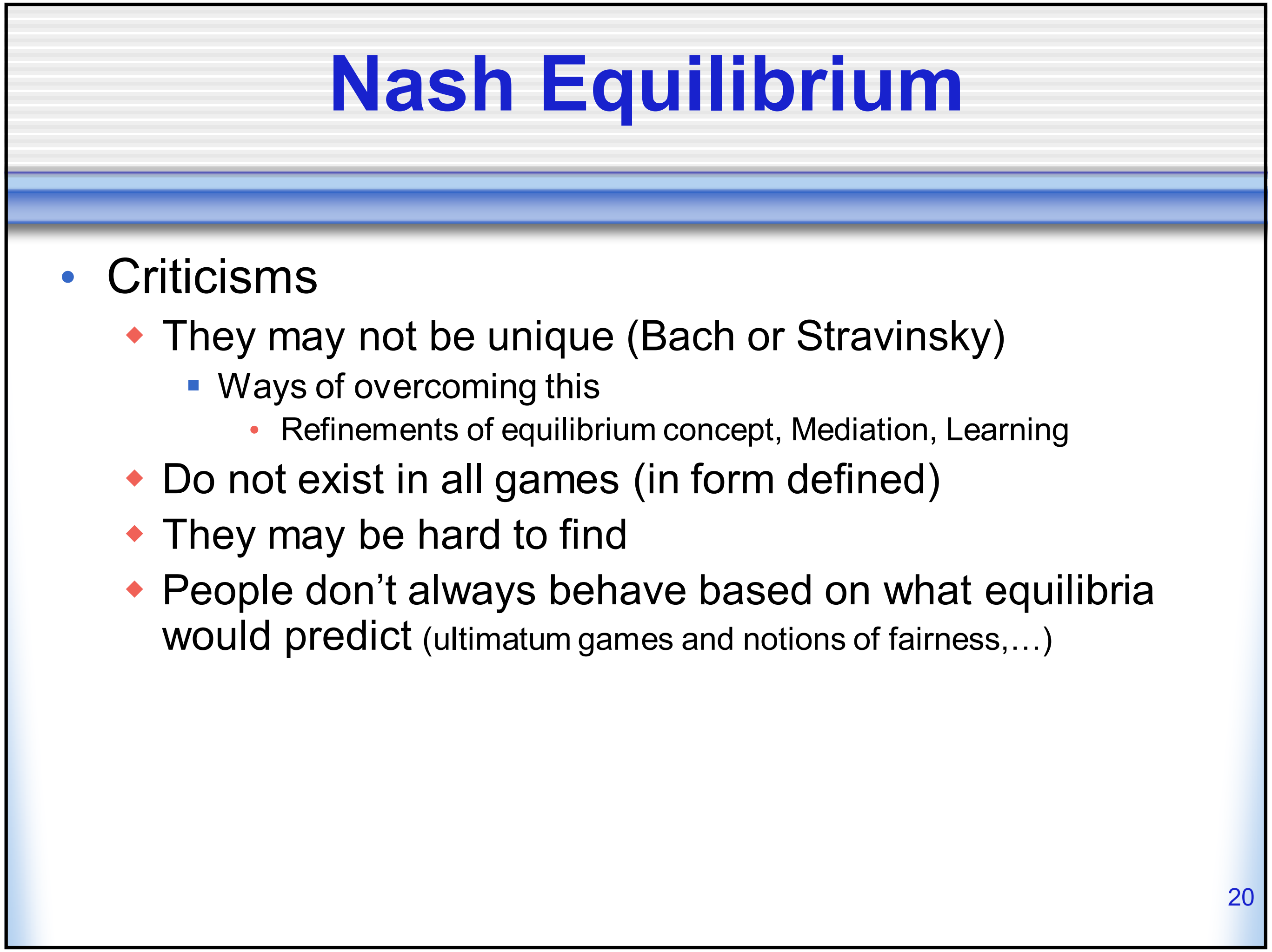
# Iterated Elimination of Dominated Strategies

- Let Ri⊆Si be the set of removed strategies for agent i
- Initially Ri=Ø
- Choose agent i, and strategy $s_i$ such that $s_i \in S_i \backslash R_i$ and there exists $s_i' \in S_i \backslash R_i$ such that

  $$u_i(s_i', s_{-i}) > u_i(s_i, s_{-i}) \text{ for all } s_{-i} \in S_{-i} \backslash R_{-i}$$

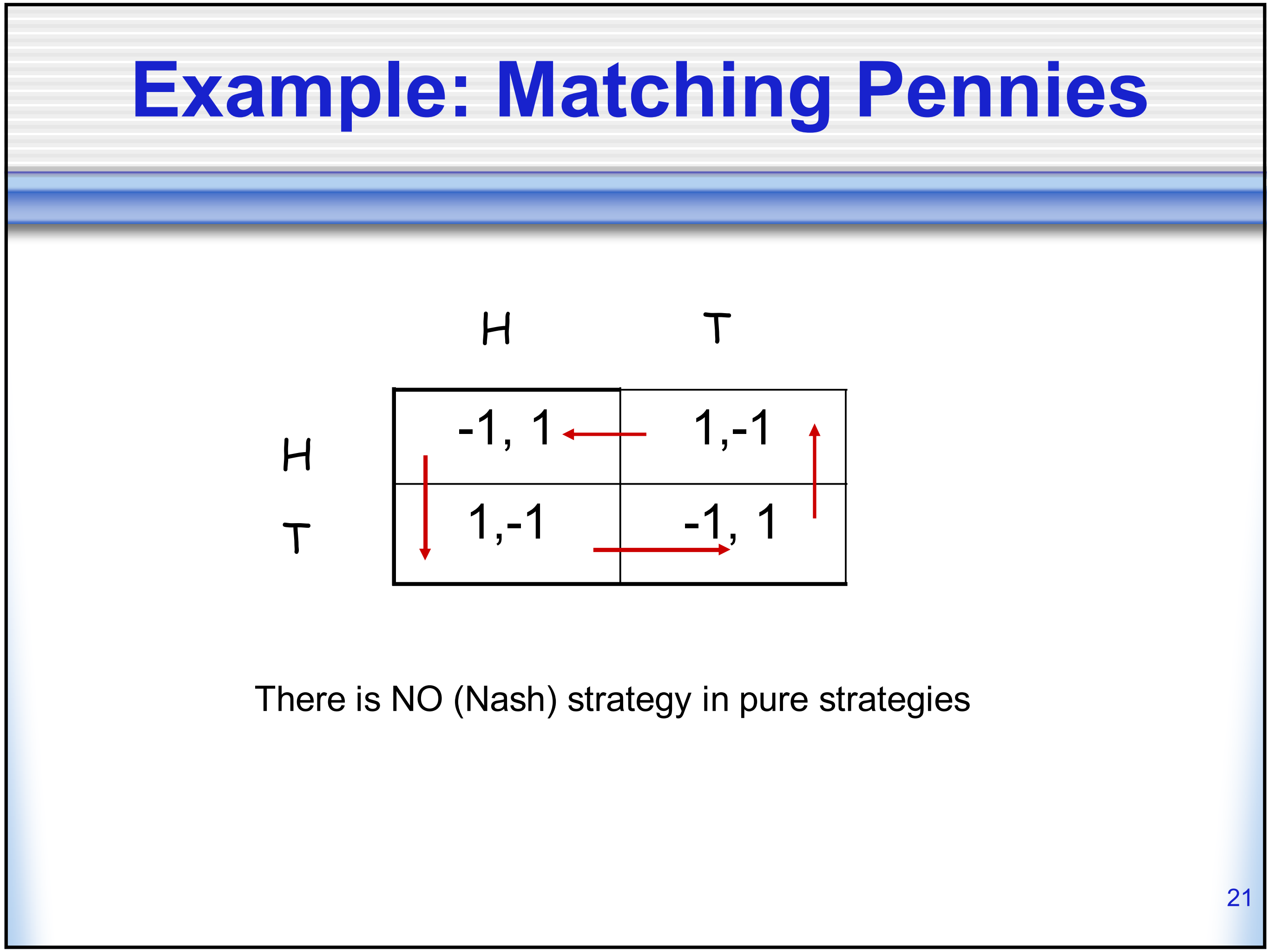
- Add $s_i$ to $R_i$, continue

- **Thm:** If a unique strategy profile, s*, survives then it is a Nash Eq.
- **Thm:** If a profile, s*, is a Nash Equilibrium then it must survive iterated elimination.



# Nash Equilibrium

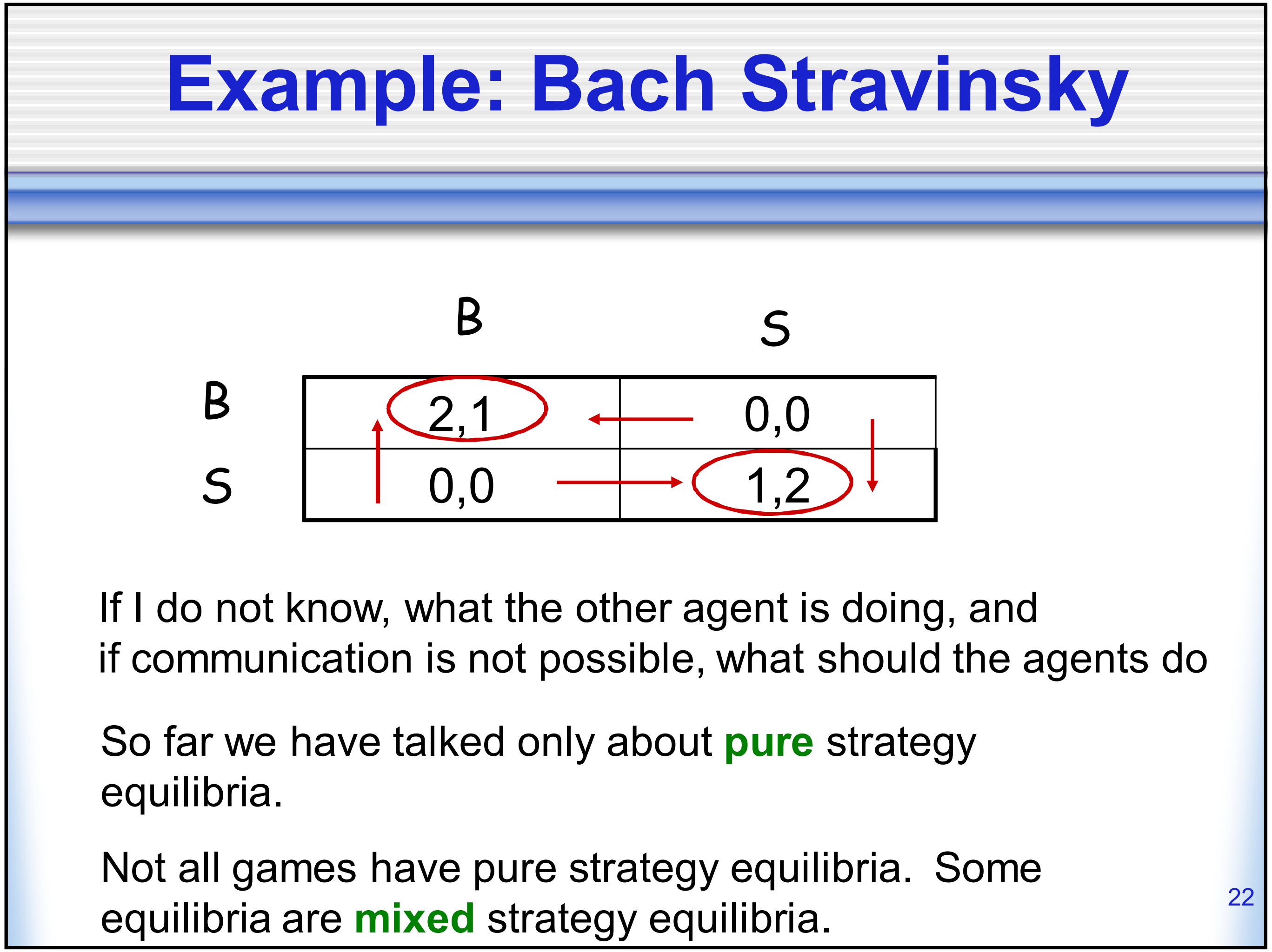
- Criticisms
  - They may not be unique (Bach or Stravinsky)
    - Ways of overcoming this
      - Refinements of equilibrium concept, Mediation, Learning
  - Do not exist in all games (in form defined)
  - They may be hard to find
  - People don’t always behave based on what equilibria would predict (ultimatum games and notions of fairness,…)



# Example: Matching Pennies

|   | H | T |
|---|---|---|
| H | -1, 1 | 1,-1 |
| T | 1,-1 | -1, 1 |

There is NO (Nash) strategy in pure strategies



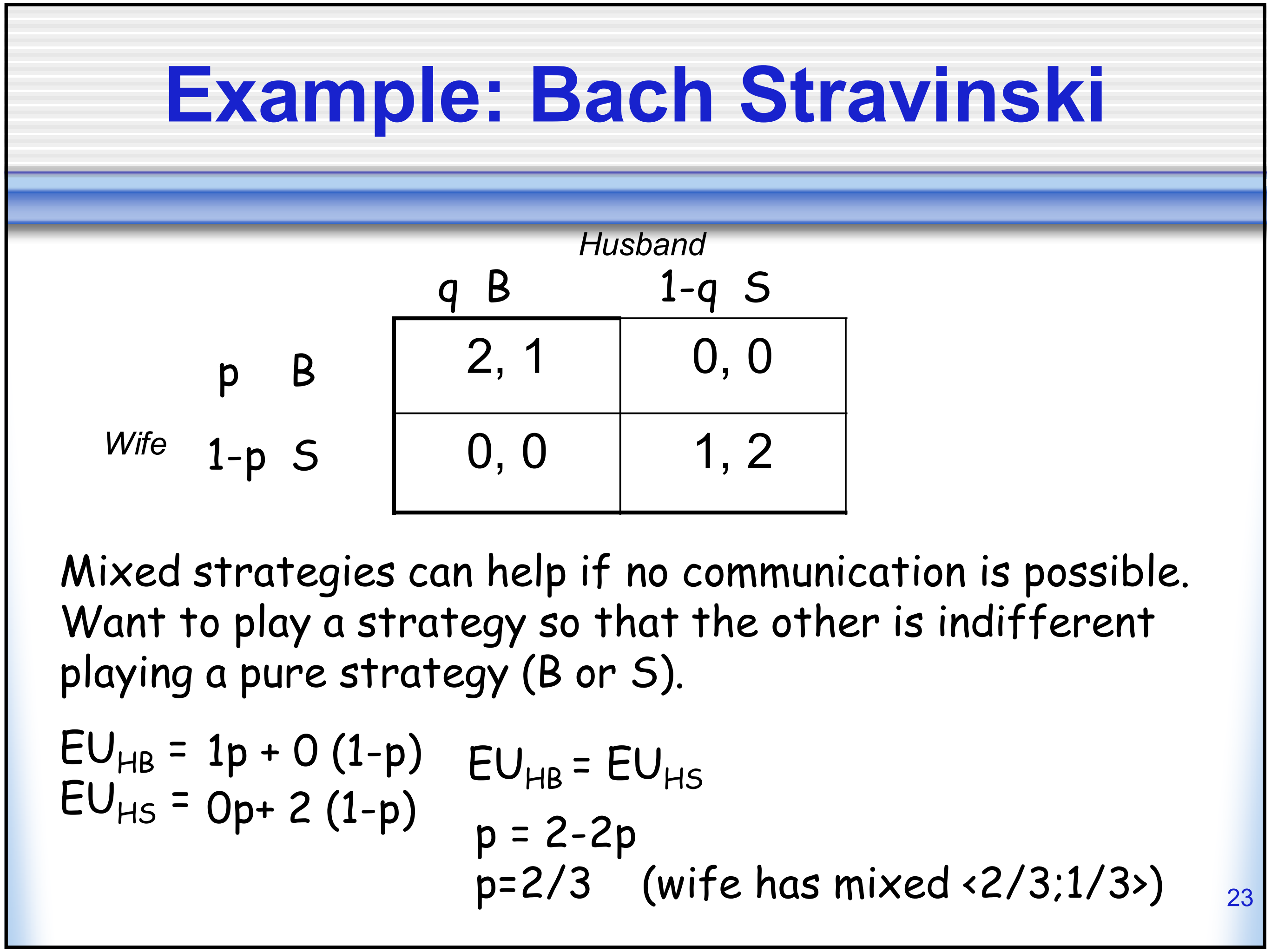
# Example: Bach Stravinsky

| | B | S |
|---|---|---|
| B | 2,1 | 0,0 |
| S | 0,0 | 1,2 |

If I do not know, what the other agent is doing, and
if communication is not possible, what should the agents do

So far we have talked only about **pure** strategy equilibria.

Not all games have pure strategy equilibria. Some equilibria are **mixed** strategy equilibria.



# Example: Bach Stravinski

*Husband*

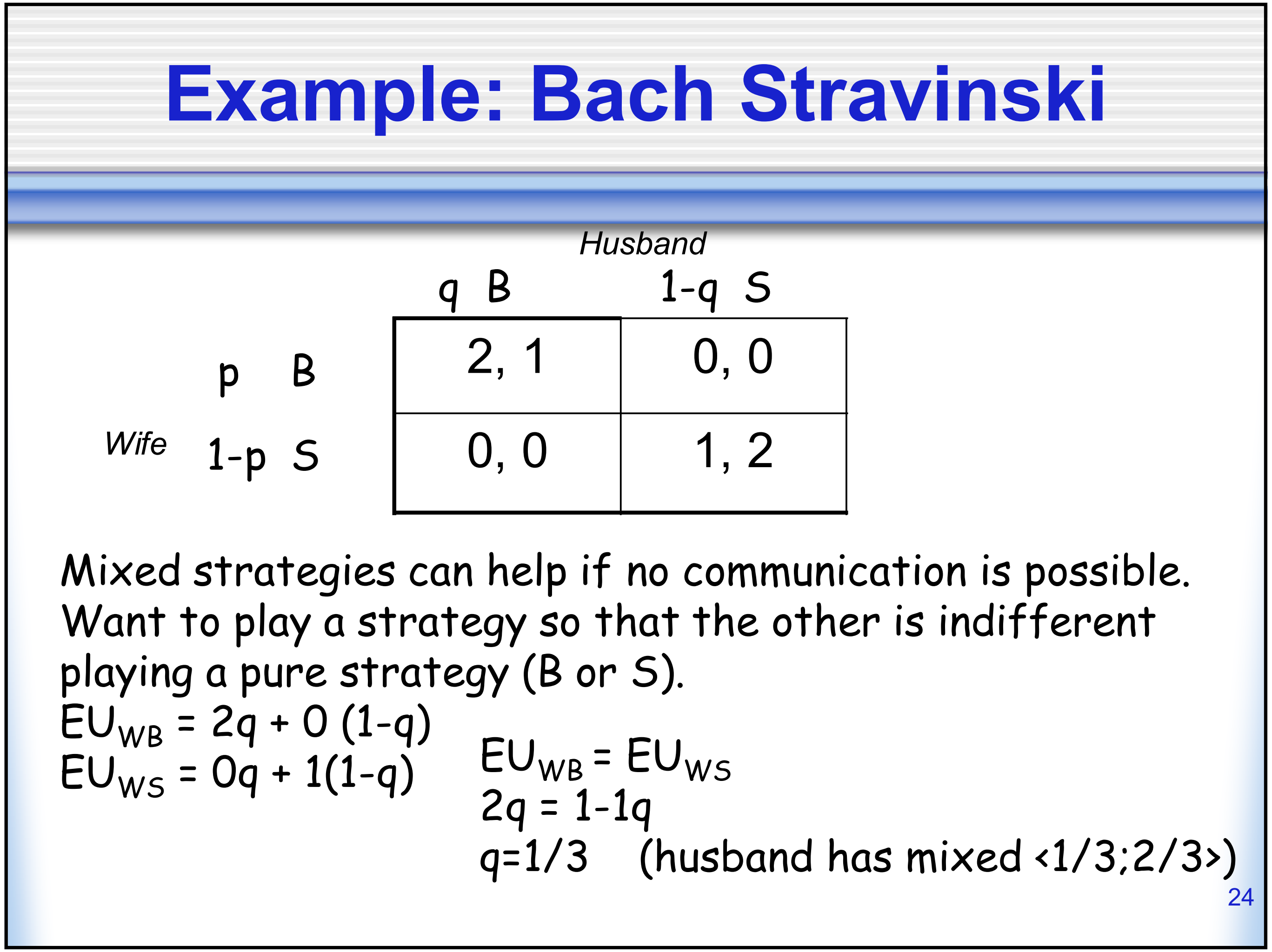
| | | q B | 1-q S |
|---|---|---|---|
| | p B | 2, 1 | 0, 0 |
| *Wife* | 1-p S | 0, 0 | 1, 2 |

Mixed strategies can help if no communication is possible. Want to play a strategy so that the other is indifferent playing a pure strategy (B or S).

$EU_{HB}$ = 1p + 0 (1-p)
$EU_{HS}$ = 0p+ 2 (1-p)

$EU_{HB} = EU_{HS}$

p = 2-2p
p=2/3 (wife has mixed <2/3;1/3>)



# Example: Bach Stravinski

*Husband*

| | | q B | 1-q S |
|---|---|---|---|
| | p B | 2, 1 | 0, 0 |
| *Wife* | 1-p S | 0, 0 | 1, 2 |

Mixed strategies can help if no communication is possible. Want to play a strategy so that the other is indifferent playing a pure strategy (B or S).

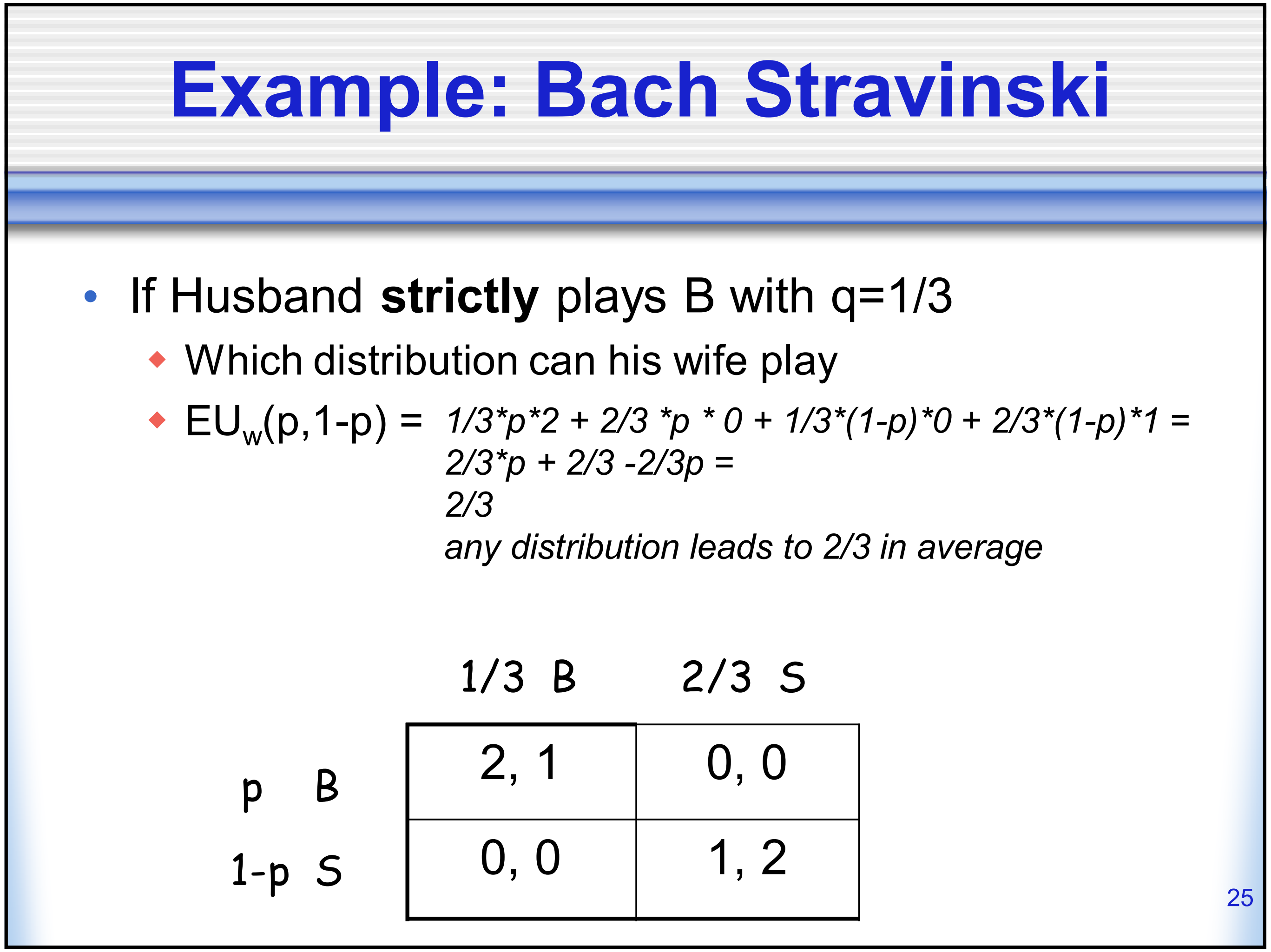
$EU_{WB} = 2q + 0\,(1\text{-}q)$

$EU_{WS} = 0q + 1(1\text{-}q)$

$EU_{WB} = EU_{WS}$

$2q = 1\text{-}1q$

$q=1/3$ (husband has mixed <1/3;2/3>)



# Example: Bach Stravinski

- If Husband **strictly** plays B with q=1/3
  - Which distribution can his wife play
  - $EU_w(p,1-p)$ = *1/3*p*2 + 2/3 *p * 0 + 1/3*(1-p)*0 + 2/3*(1-p)*1 =*  
    *2/3*p + 2/3 -2/3p =*  
    *2/3*  
    *any distribution leads to 2/3 in average*

| | | 1/3 B | 2/3 S |
|---|---|---|---|
| p | B | 2, 1 | 0, 0 |
| 1-p | S | 0, 0 | 1, 2 |

# Example: Bach Stravinski

*Husband*

| | | q B | 1-q S |
|---|---|---|---|
| | p B | 2, 1 | 0, 0 |
| *Wife* | 1-p S | 0, 0 | 1, 2 |

$EU_{WB} = 2q + 0\ (1-q)$
$EU_{WS} = 0q + 1(1-q)$

$EU_{WB} = EU_{WS}$
$2q = 1-1q$
$q=1/3$

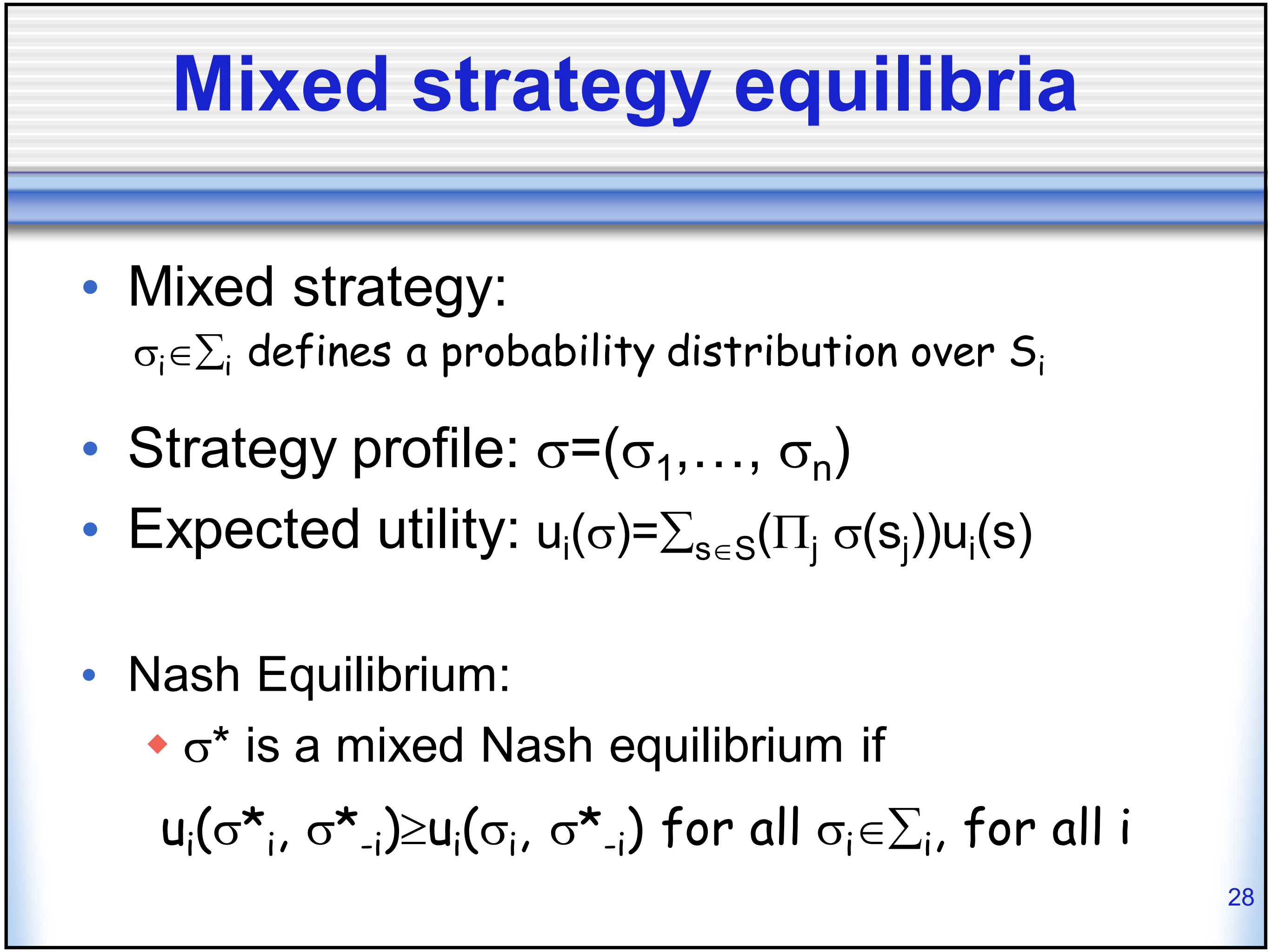
husband has mixed strategy <1/3;2/3>
wife has mixed strategy <2/3;1/3>

# Example: Bach Stravinski

- If Husband **strictly** plays B with q=1/3
  - Which distribution should wife play
  - $Eu_w(p,1-p) = 2/3$

|  |  |  | Husband |  |
|---|---|---|---|---|
|  |  |  | q B | 1-q S |
|  | p | B | 2, 1 | 0, 0 |
| Wife | 1-p | S | 0, 0 | 1, 2 |

- If Husband deviates q<1/3
  - Wife deviates plays **S**
- If Husband q>1/3
  - Wife plays **B**
- Equilibrium: {(2/3,1/3);(1/3,2/3)}



# Mixed strategy equilibria

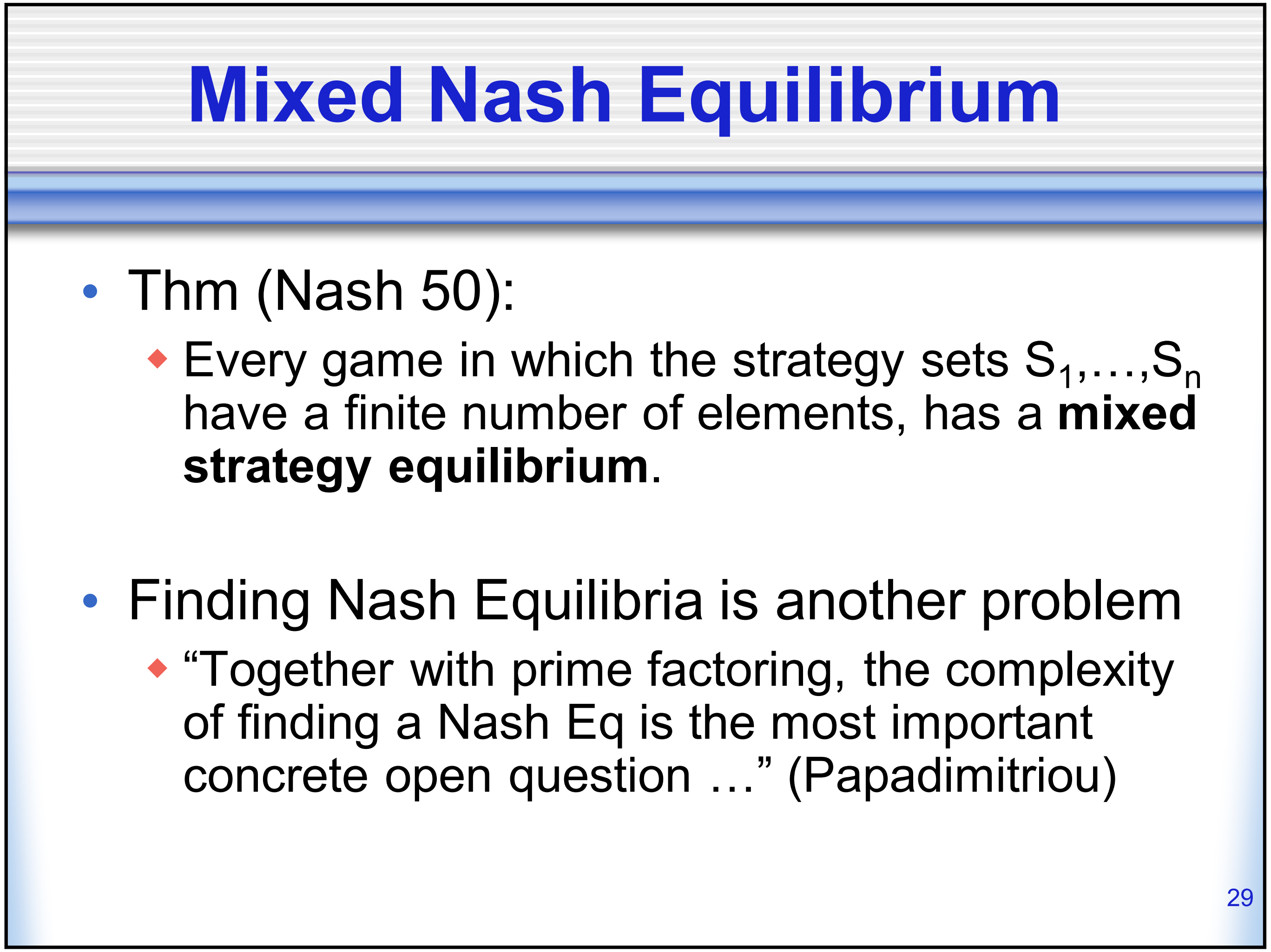
- Mixed strategy:

  $\sigma_i \in \Sigma_i$ defines a probability distribution over $S_i$

- Strategy profile: $\sigma=(\sigma_1,\ldots, \sigma_n)$
- Expected utility: $u_i(\sigma)=\Sigma_{s\in S}(\Pi_j\ \sigma(s_j))u_i(s)$

- Nash Equilibrium:
  - $\sigma^*$ is a mixed Nash equilibrium if

    $u_i(\sigma^*_i, \sigma^*_{-i}) \geq u_i(\sigma_i, \sigma^*_{-i})$ for all $\sigma_i \in \Sigma_i$, for all $i$



# Mixed Nash Equilibrium

- Thm (Nash 50):
  - Every game in which the strategy sets $S_1,\ldots,S_n$ have a finite number of elements, has a **mixed strategy equilibrium**.
- Finding Nash Equilibria is another problem
  - “Together with prime factoring, the complexity of finding a Nash Eq is the most important concrete open question …” (Papadimitriou)

# Bayesian-Nash Equil

## (Harsanyi 68)

- So far we have assumed that agents have complete information about each other (including payoffs)
  - Very strong assumption!

- Assume agent i has type $\theta_i \in \Theta_i$, defines the payoff $u_i(s, \theta_i)$

- Agents have common prior over distribution of types $p(\theta)$
  - Conditional probability $p(\theta_{-i} | \theta_i)$
    (obtained by Bayes Rule when possible)

# Battle of the sexes

- Shopping or Basketball?
- Sally knows Kevins type
  Kevin does not know Sally‘s type but possible types.

What should Sally play?

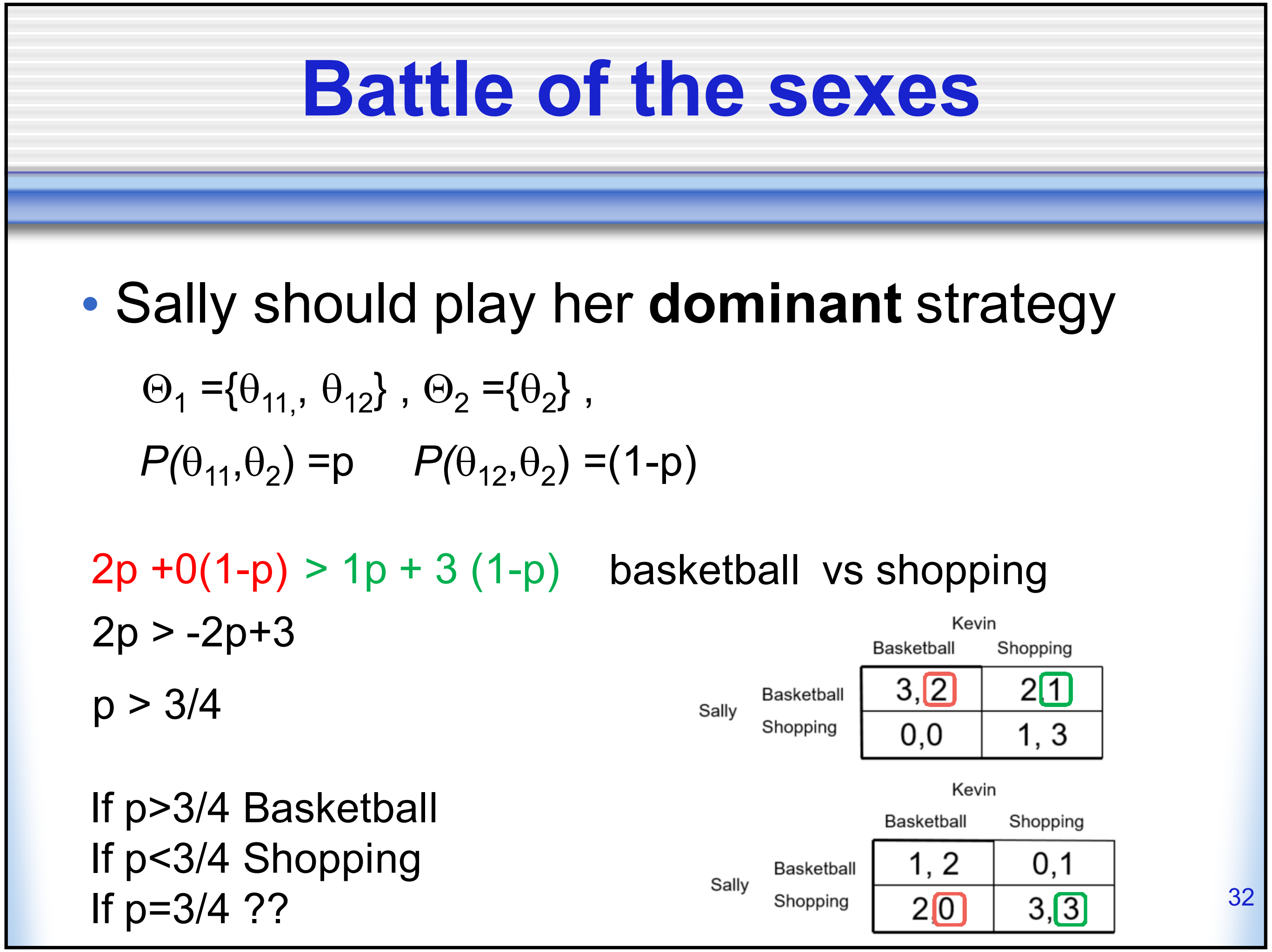
| | | Kevin | |
|---|---|---|---|
| | | Basketball | Shopping |
| Sally | Basketball | 3, 2 | 2,1 |
| | Shopping | 0,0 | 1, 3 |

Sally a basketball fan

Her **dominant** strategy!

| | | Kevin | |
|---|---|---|---|
| | | Basketball | Shopping |
| Sally | Basketball | 1, 2 | 0,1 |
| | Shopping | 2,0 | 3, 3 |

Sally a shopping fan



# Battle of the sexes

- Sally should play her **dominant** strategy

$\Theta_1 = \{\theta_{11,}, \theta_{12}\}$ , $\Theta_2 = \{\theta_2\}$ ,

$P(\theta_{11},\theta_2) = p$ $\quad P(\theta_{12},\theta_2) = (1-p)$

$2p + 0(1-p) > 1p + 3(1-p)$ basketball vs shopping

$2p > -2p+3$

$p > 3/4$

| | | Kevin | |
|---|---|---|---|
| | | Basketball | Shopping |
| Sally | Basketball | 3,$\boxed{2}$ | 2,$\boxed{1}$ |
| | Shopping | 0,0 | 1, 3 |

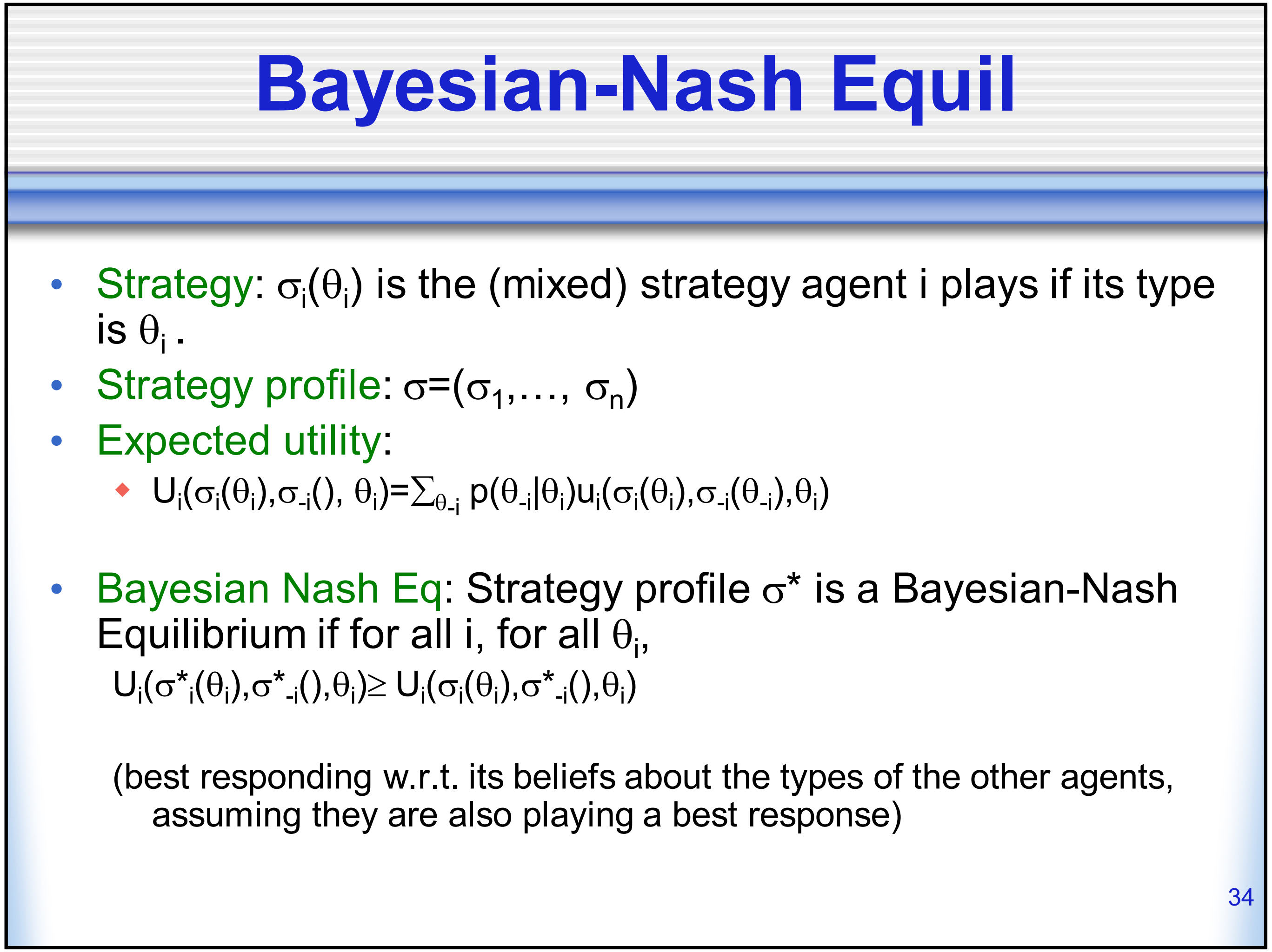
| | | Kevin | |
|---|---|---|---|
| | | Basketball | Shopping |
| Sally | Basketball | 1, 2 | 0,1 |
| | Shopping | 2,$\boxed{0}$ | 3,$\boxed{3}$ |

If p>3/4 Basketball
If p<3/4 Shopping
If p=3/4 ??

# Battle of the sexes

- Sally‘s decision depends on her known type
- Kevin’s decision depends on p

$$S_2^*(\theta_2) = \begin{cases} \textit{Basketball} & \textit{if } p>3/4 \\ (q,\ 1-q), q \in [0,1] & \textit{if } p=3/4 \\ \textit{Shopping} & \textit{if } p<3/4 \end{cases}$$



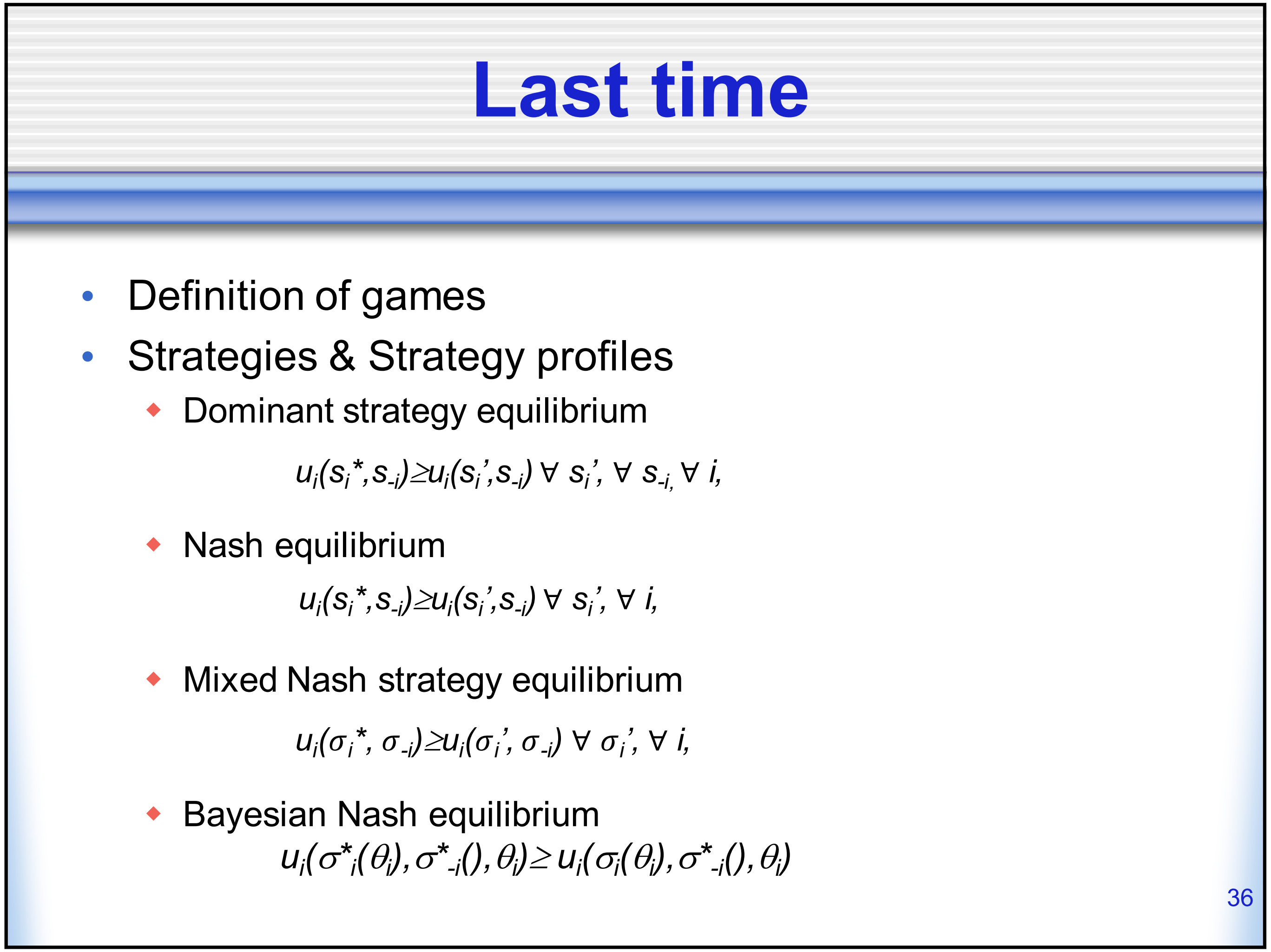
# Bayesian-Nash Equil

- Strategy: $\sigma_i(\theta_i)$ is the (mixed) strategy agent i plays if its type is $\theta_i$ .
- Strategy profile: $\sigma=(\sigma_1,\ldots,\sigma_n)$
- Expected utility:
  - $U_i(\sigma_i(\theta_i),\sigma_{-i}(), \theta_i)=\Sigma_{\theta_{-i}} p(\theta_{-i}|\theta_i)u_i(\sigma_i(\theta_i),\sigma_{-i}(\theta_{-i}),\theta_i)$

- Bayesian Nash Eq: Strategy profile $\sigma^*$ is a Bayesian-Nash Equilibrium if for all i, for all $\theta_i$,

  $U_i(\sigma^*_i(\theta_i),\sigma^*_{-i}(),\theta_i)\geq U_i(\sigma_i(\theta_i),\sigma^*_{-i}(),\theta_i)$

  (best responding w.r.t. its beliefs about the types of the other agents, assuming they are also playing a best response)



# Last time

- Definition of games
- Strategies & Strategy profiles
  - Dominant strategy equilibrium

    $$u_i(s_i^*,s_{-i})\geq u_i(s_i',s_{-i})\ \forall\ s_i',\ \forall\ s_{-i,}\ \forall\ i,$$

  - Nash equilibrium

    $$u_i(s_i^*,s_{-i})\geq u_i(s_i',s_{-i})\ \forall\ s_i',\ \forall\ i,$$

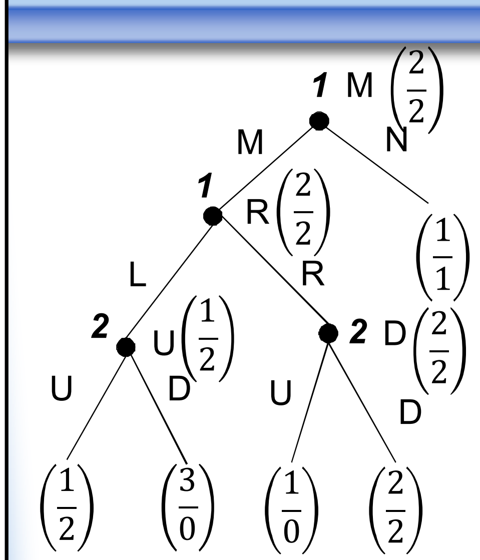
  - Mixed Nash strategy equilibrium

    $$u_i(\sigma_i^*,\ \sigma_{-i})\geq u_i(\sigma_i',\ \sigma_{-i})\ \forall\ \sigma_i',\ \forall\ i,$$

  - Bayesian Nash equilibrium

    $$u_i(\sigma^*_i(\theta_i),\sigma^*_{-i}(),\theta_i)\geq u_i(\sigma_i(\theta_i),\sigma^*_{-i}(),\theta_i)$$

# Extensive Form Games



Any finite game of perfect information has a pure strategy Nash equilibrium.

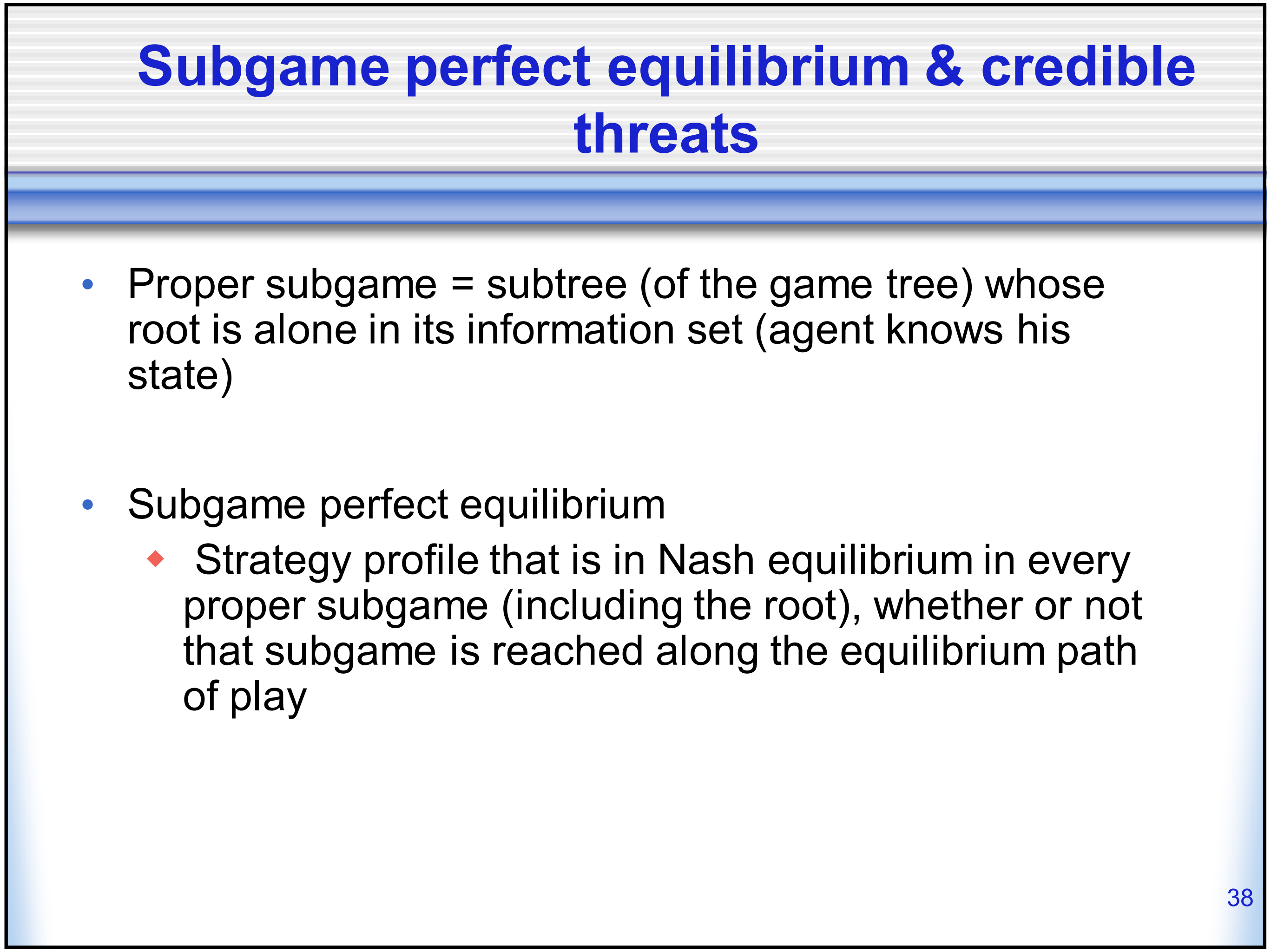
It can be found by backward induction.

How to find a Nash Equilibrium?

By backward induction!

Have to define an action for every choice point.

(MR, UD)

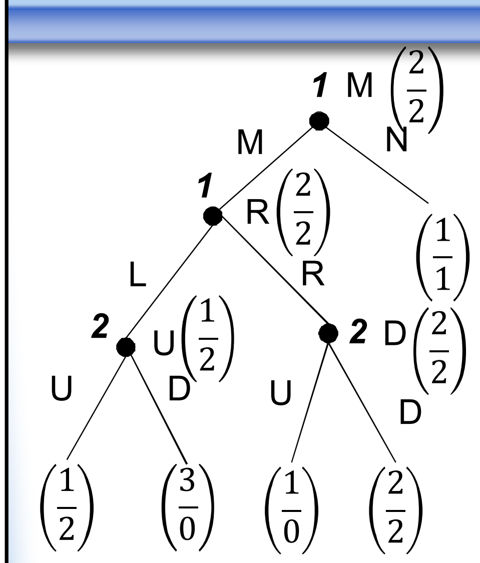


## Subgame perfect equilibrium & credible threats

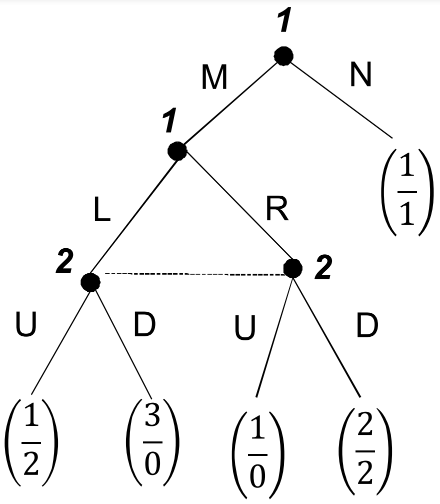
- Proper subgame = subtree (of the game tree) whose root is alone in its information set (agent knows his state)

- Subgame perfect equilibrium
  - Strategy profile that is in Nash equilibrium in every proper subgame (including the root), whether or not that subgame is reached along the equilibrium path of play

# Subgame perfect equilibrium

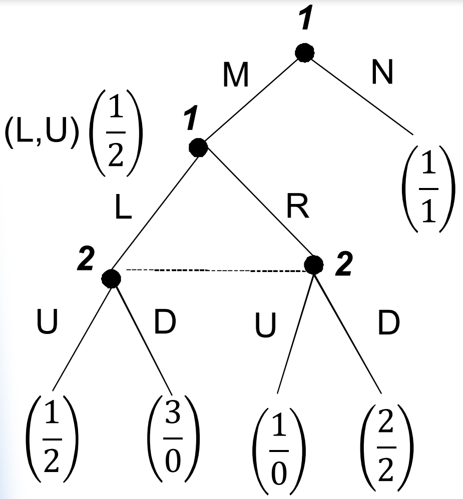


(MR, UD)



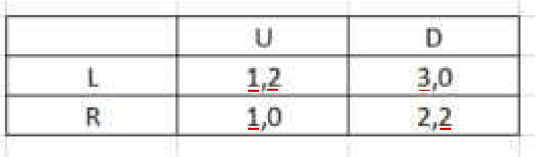
What is the strategy now?

# Subgame perfect NE equilibrium



(MR, UD)

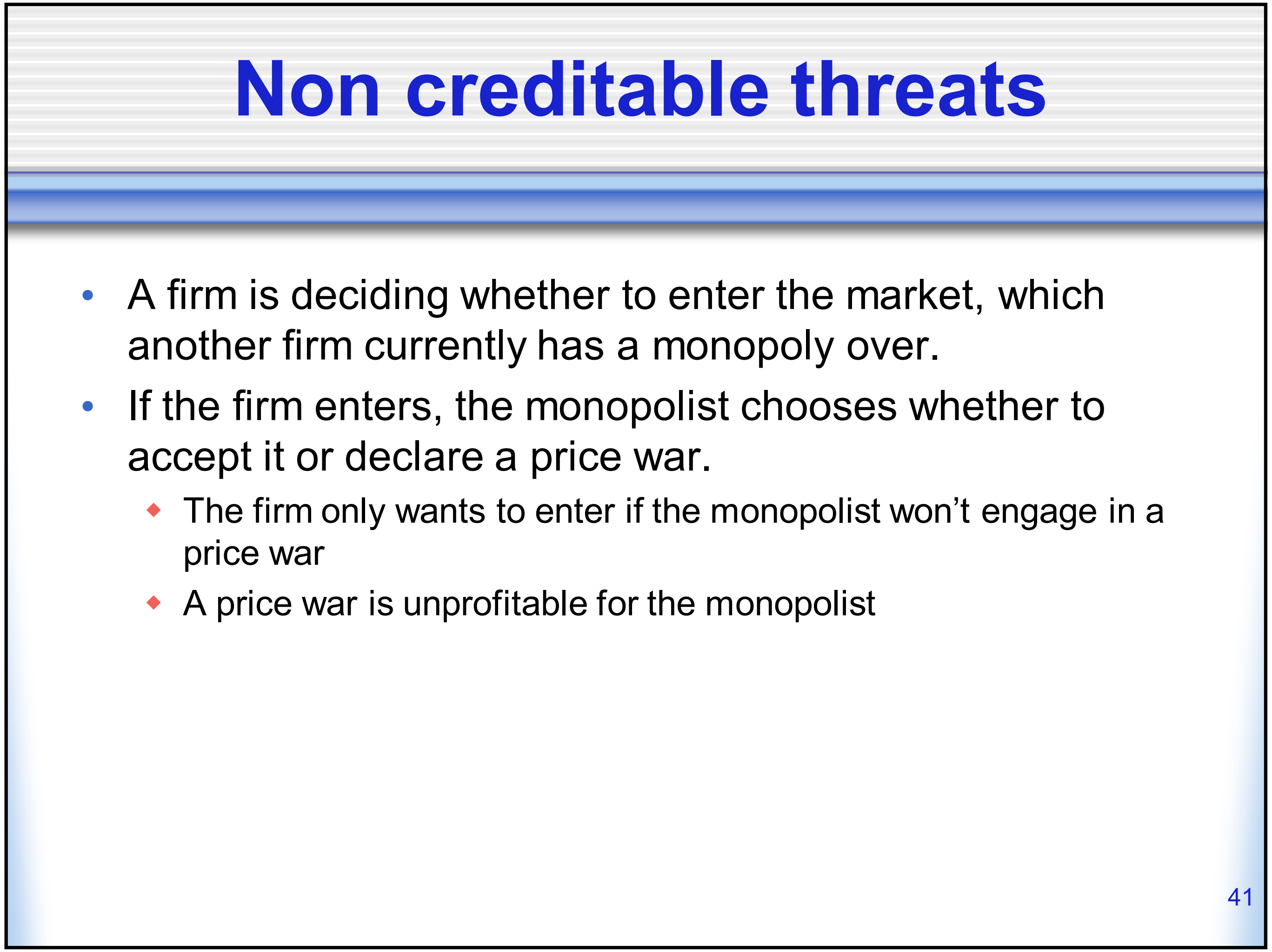
What is the strategy now?



| | U | D |
|---|---|---|
| L | 1,2 | 3,0 |
| R | 1,0 | 2,2 |

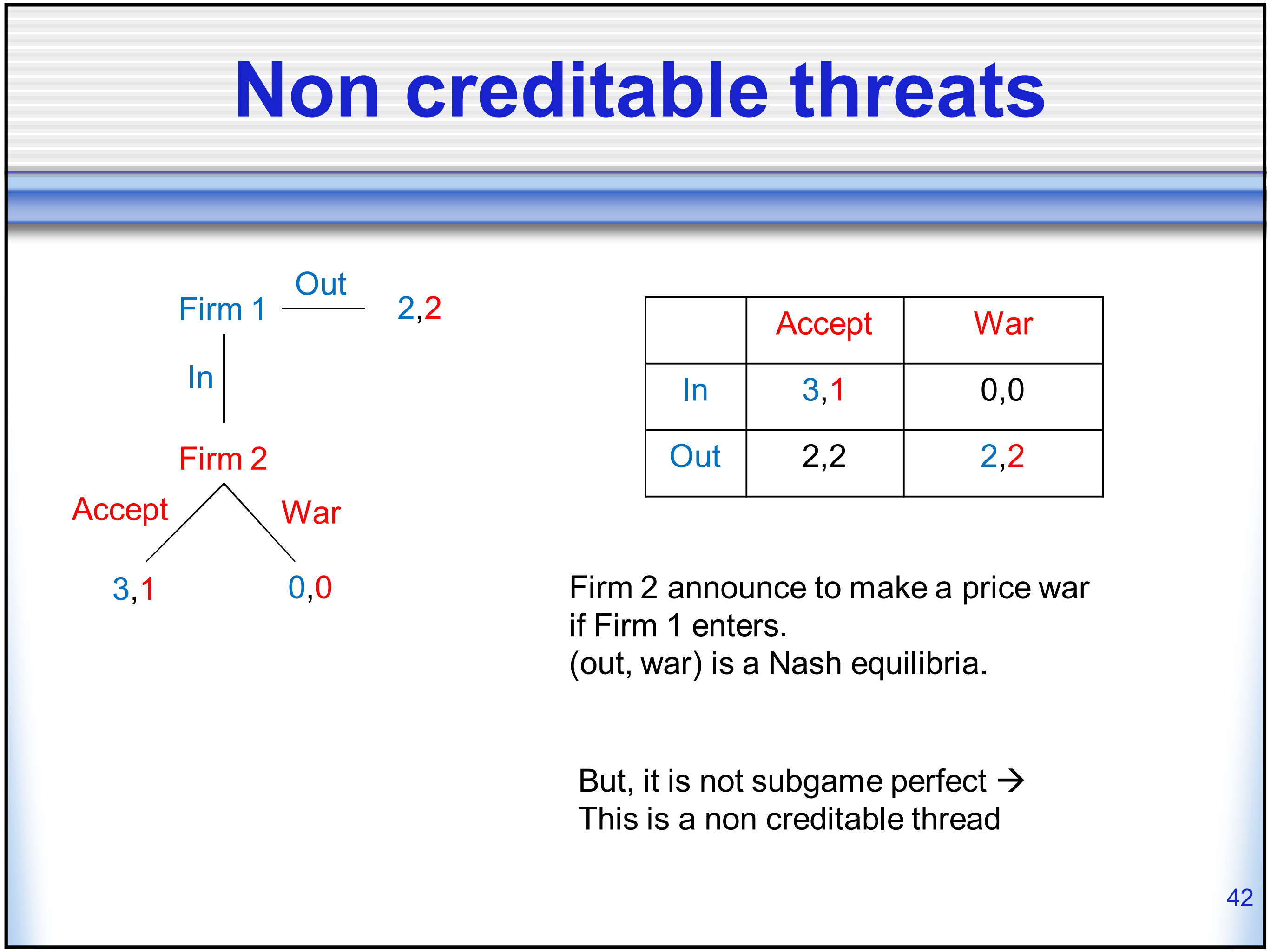
If U best is L or R
If D best is L
If L best is U
If R best is D
(L,U) is the NE

(ML, U) and (NL, U) are the SPNE of the game
$\binom{1}{2}$ $\binom{1}{1}$



# Non creditable threats

- A firm is deciding whether to enter the market, which another firm currently has a monopoly over.
- If the firm enters, the monopolist chooses whether to accept it or declare a price war.
  - The firm only wants to enter if the monopolist won’t engage in a price war
  - A price war is unprofitable for the monopolist

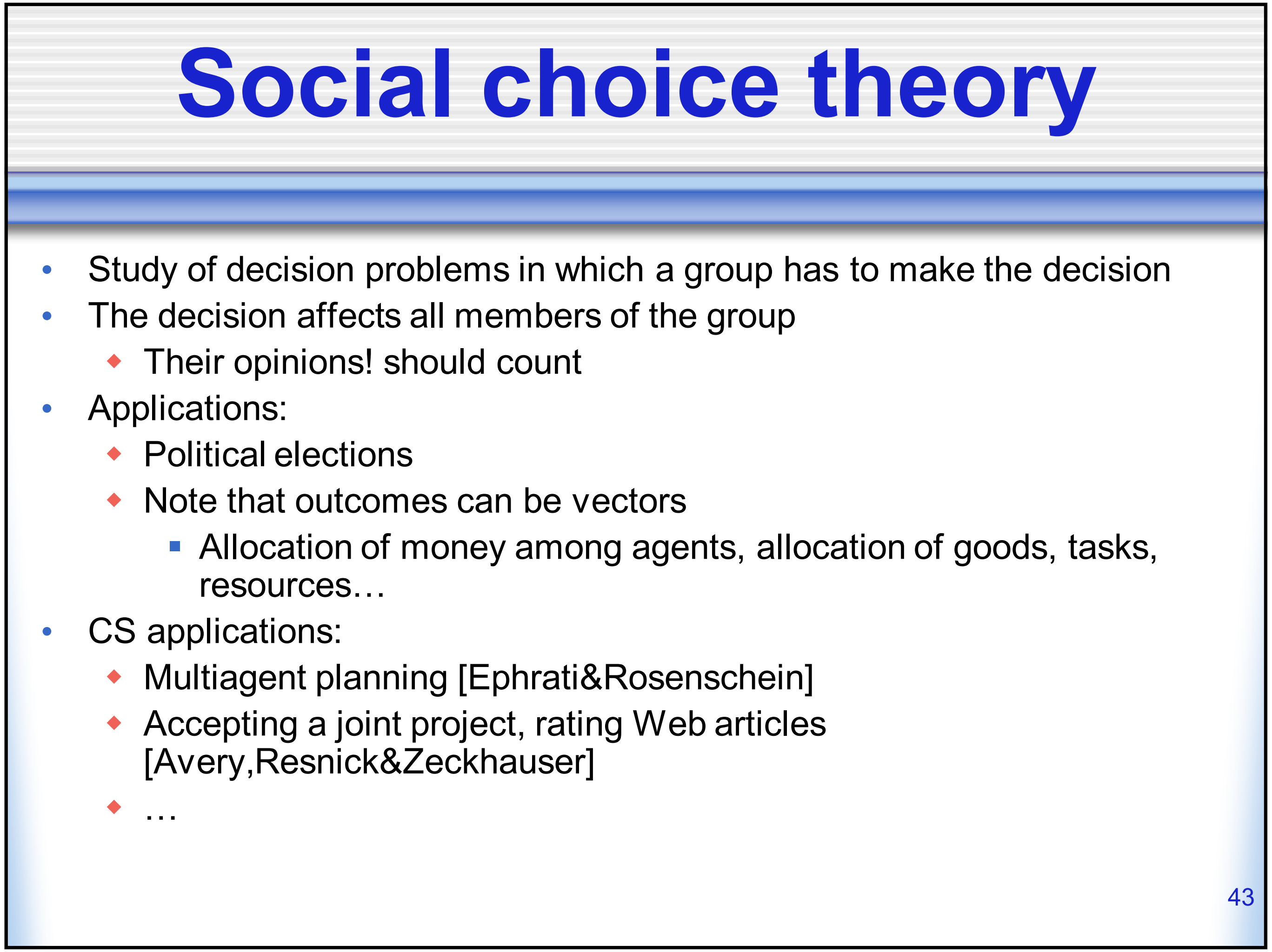


# Non creditable threats

| | Accept | War |
|---|---|---|
| In | 3,1 | 0,0 |
| Out | 2,2 | 2,2 |

Firm 2 announce to make a price war
if Firm 1 enters.
(out, war) is a Nash equilibria.

But, it is not subgame perfect →
This is a non creditable thread

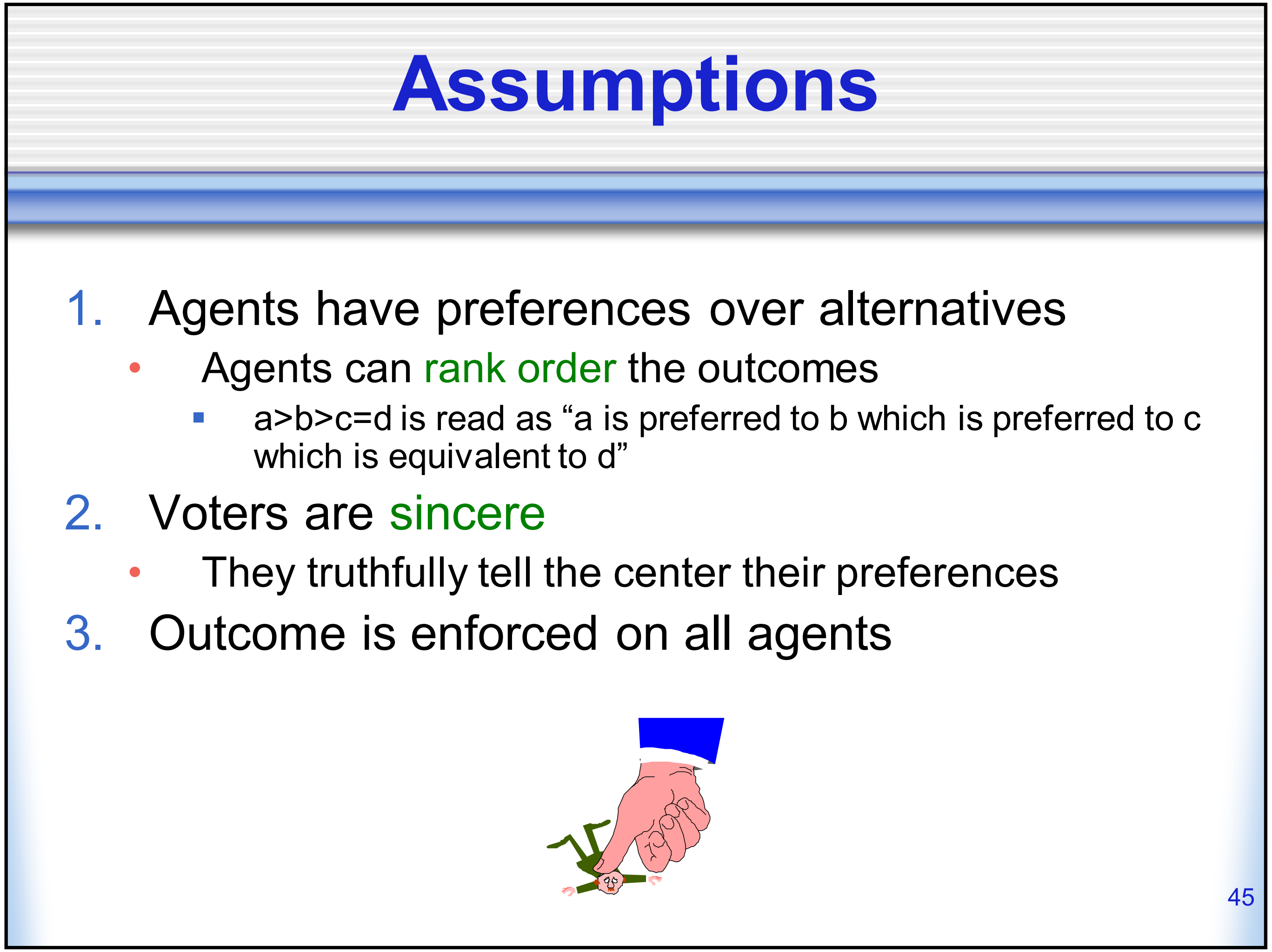


# Social choice theory

- Study of decision problems in which a group has to make the decision
- The decision affects all members of the group
  - Their opinions! should count
- Applications:
  - Political elections
  - Note that outcomes can be vectors
    - Allocation of money among agents, allocation of goods, tasks, resources…
- CS applications:
  - Multiagent planning [Ephrati&Rosenschein]
  - Accepting a joint project, rating Web articles [Avery,Resnick&Zeckhauser]
  - …

# Criteria for evaluating multiagent systems

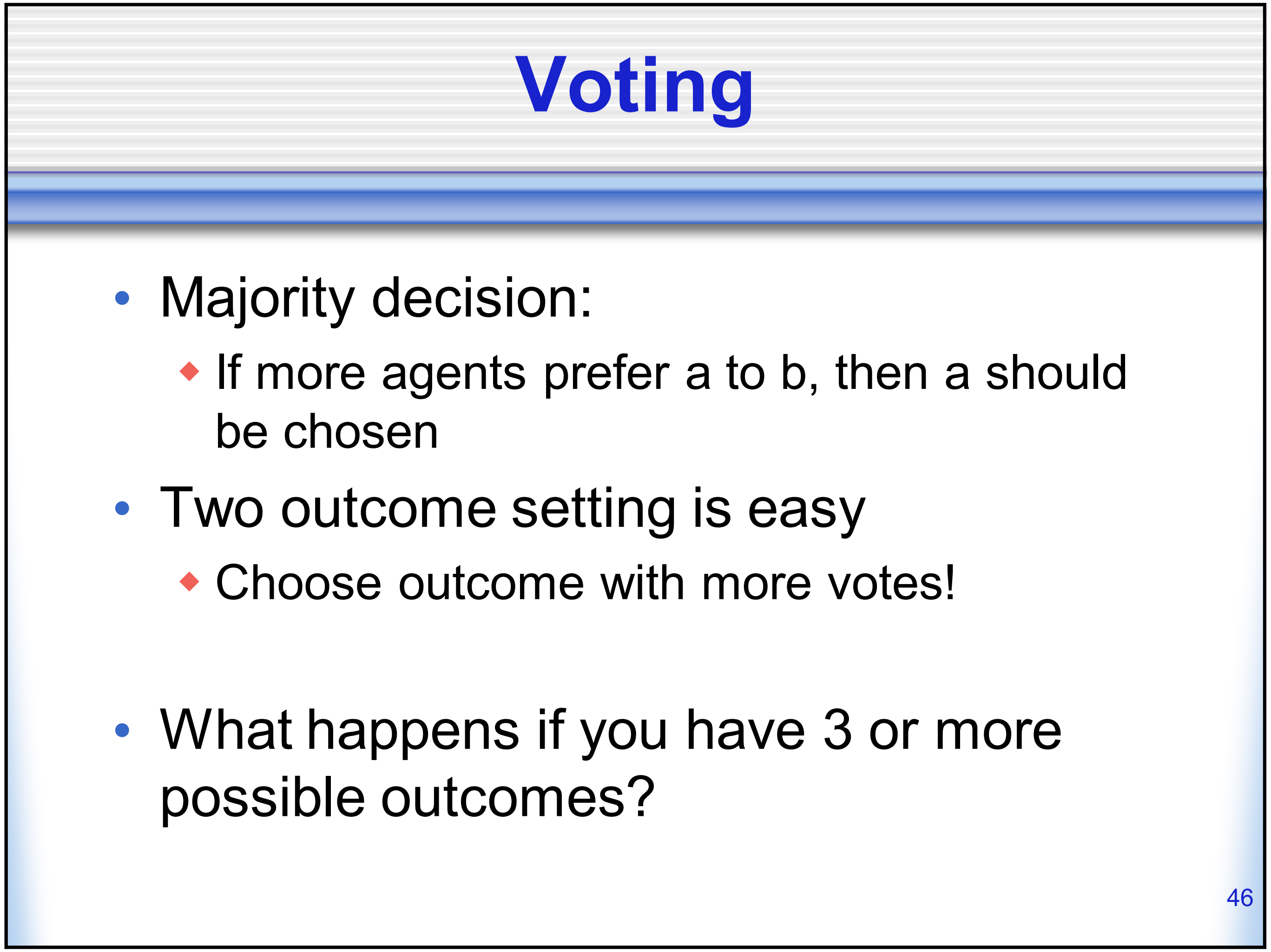
- Social welfare: $\max_{outcome} \sum_i u_i(outcome)$
- Surplus: social welfare of outcome – social welfare of status quo
  - Zero sum games have 0 surplus. Markets are not zero sum
- Pareto efficiency: An outcome o is Pareto efficient if there exists no other outcome o' s.t. some agent has higher utility in o' than in o and no agent has lower
  - Implied by social welfare maximization
- Individual rationality: Participating in the negotiation (or individual deal) is no worse than not participating
- Stability: No agents can increase their utility by changing their strategies
- Symmetry: No agent should be inherently preferred, e.g. dictator



# Assumptions

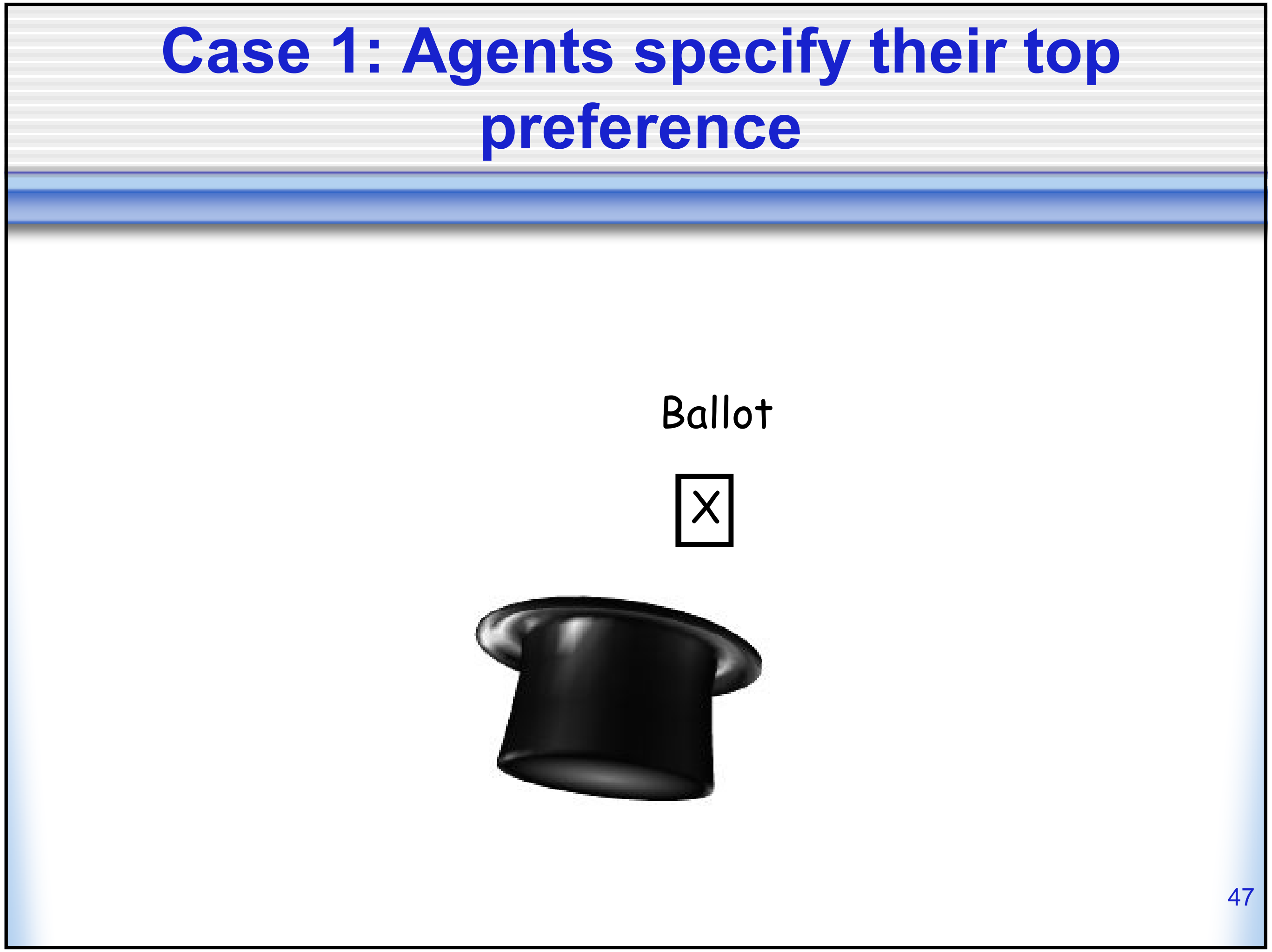
1. Agents have preferences over alternatives
   - Agents can rank order the outcomes
     - a>b>c=d is read as “a is preferred to b which is preferred to c which is equivalent to d”
2. Voters are sincere
   - They truthfully tell the center their preferences
3. Outcome is enforced on all agents

45



# Voting

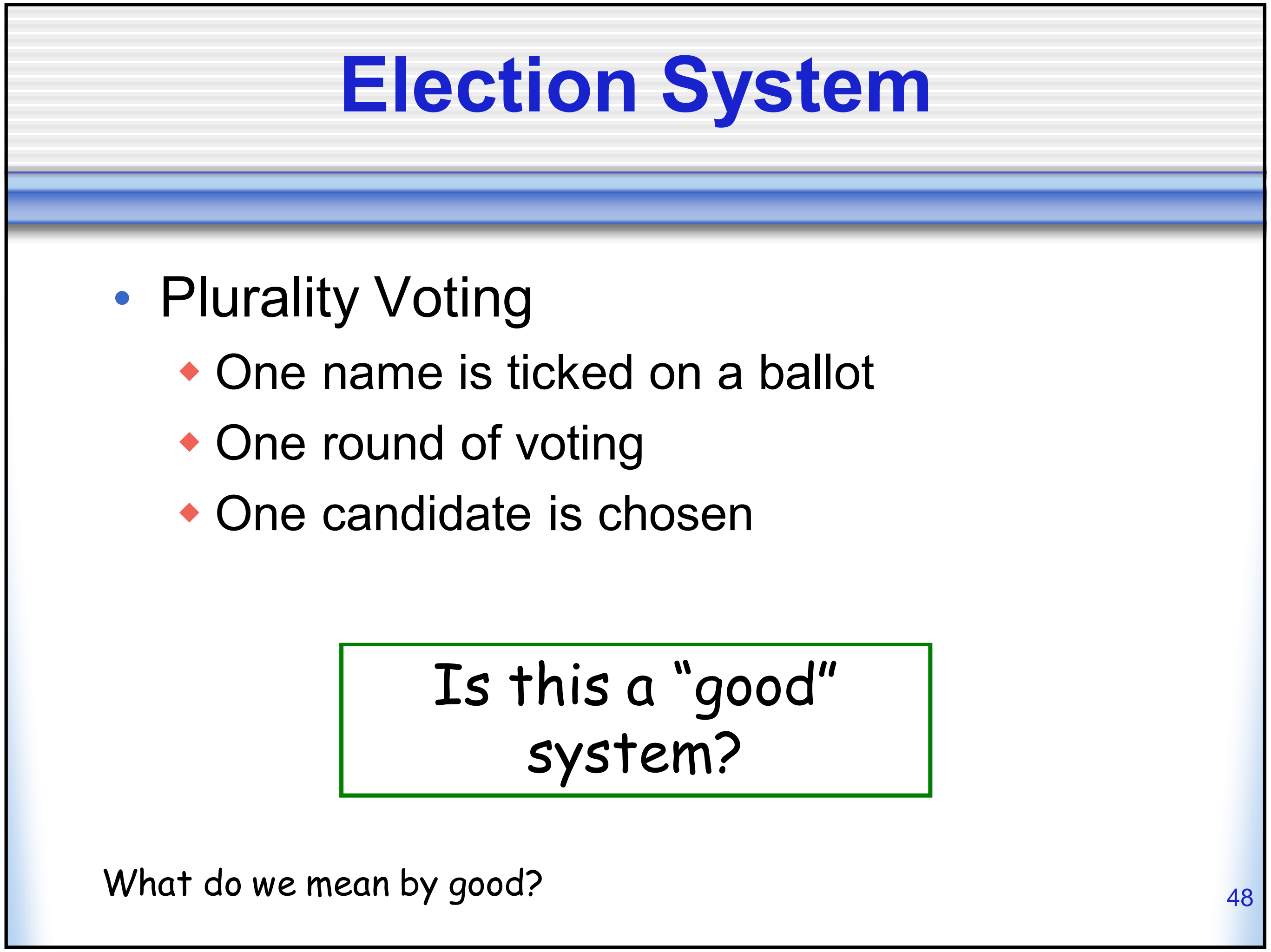
- Majority decision:
  - If more agents prefer a to b, then a should be chosen
- Two outcome setting is easy
  - Choose outcome with more votes!
- What happens if you have 3 or more possible outcomes?



# Case 1: Agents specify their top preference

Ballot

☒

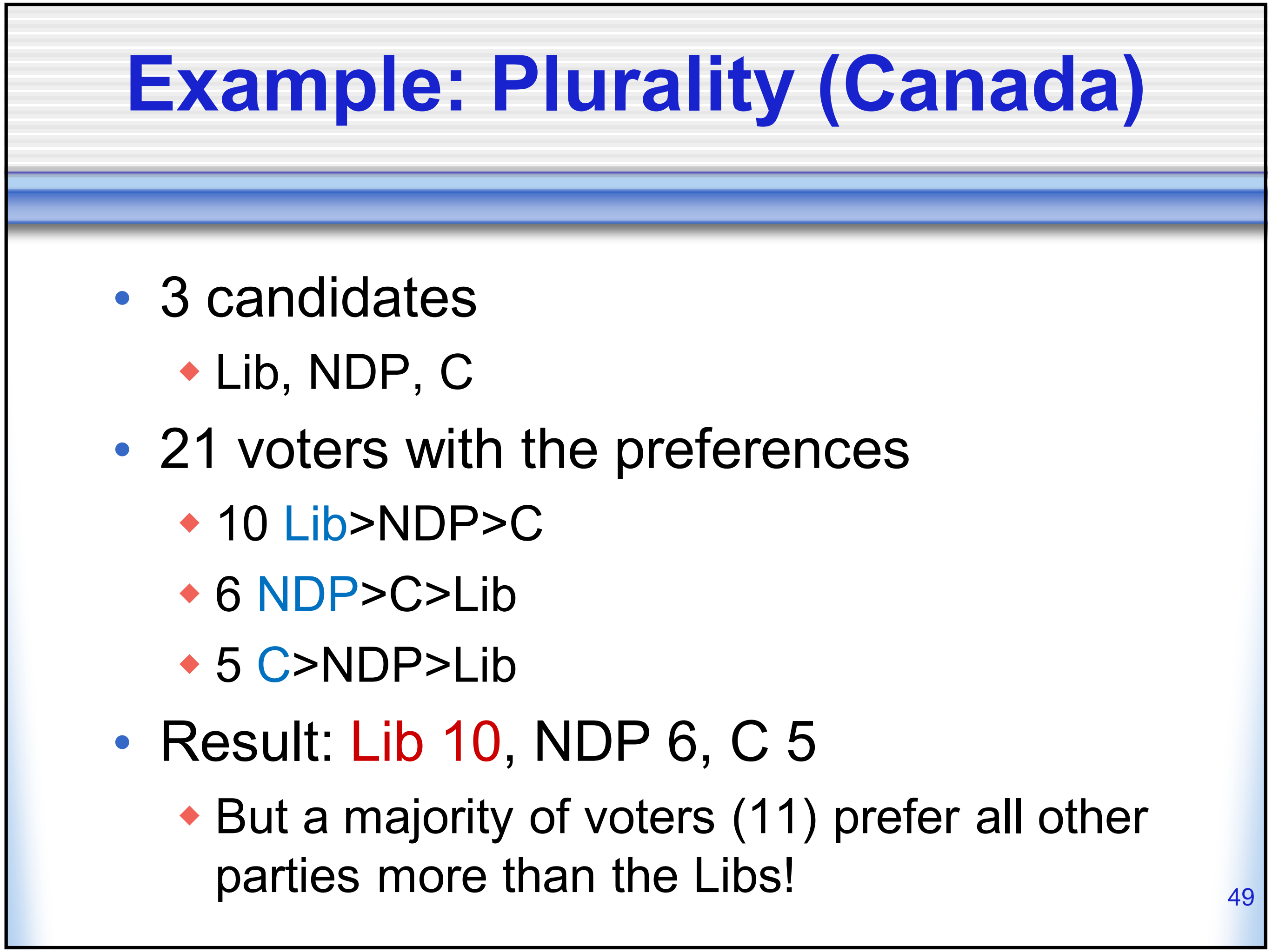


# Election System

- Plurality Voting
  - One name is ticked on a ballot
  - One round of voting
  - One candidate is chosen

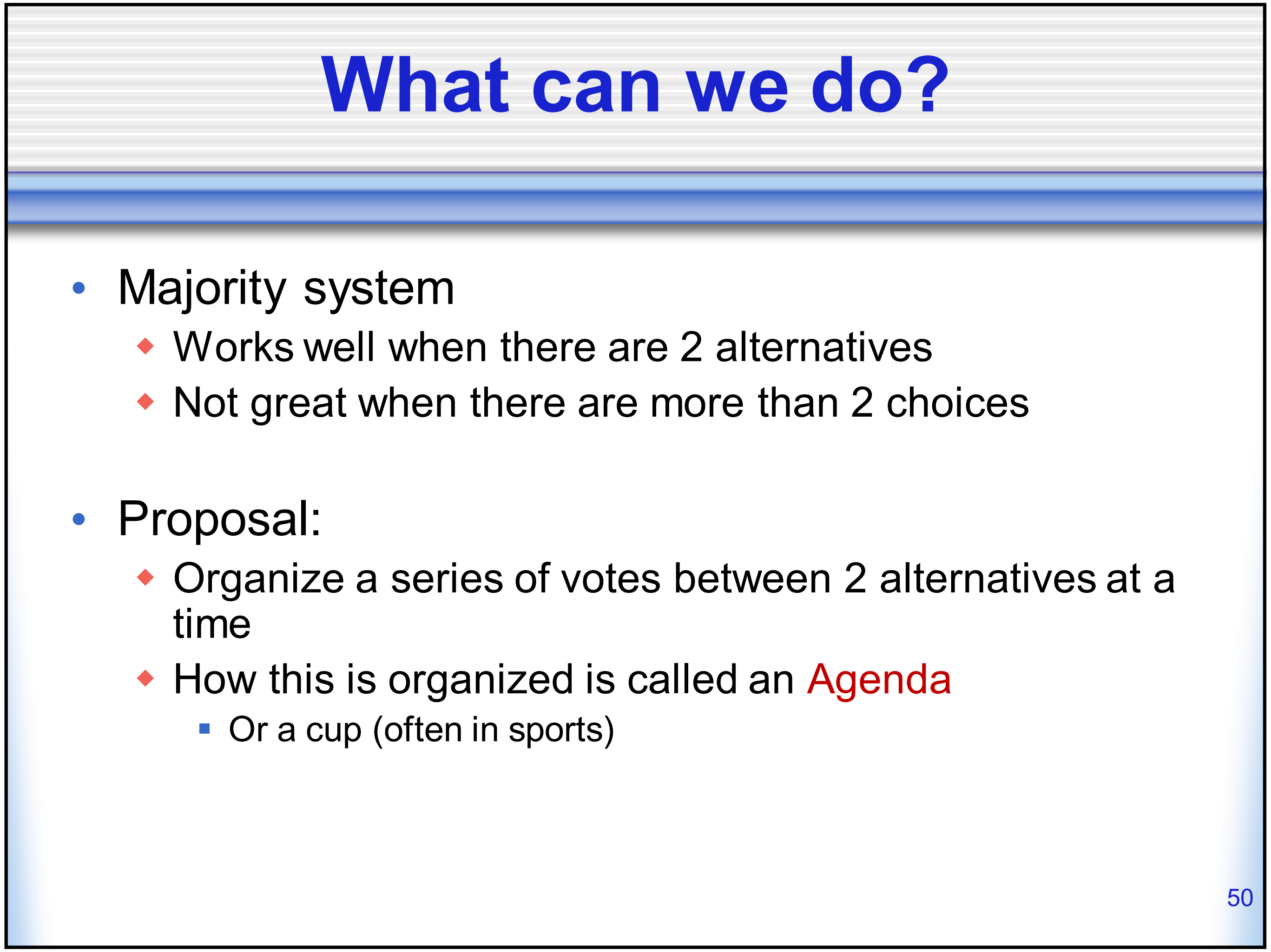
Is this a “good” system?

What do we mean by good?



# Example: Plurality (Canada)

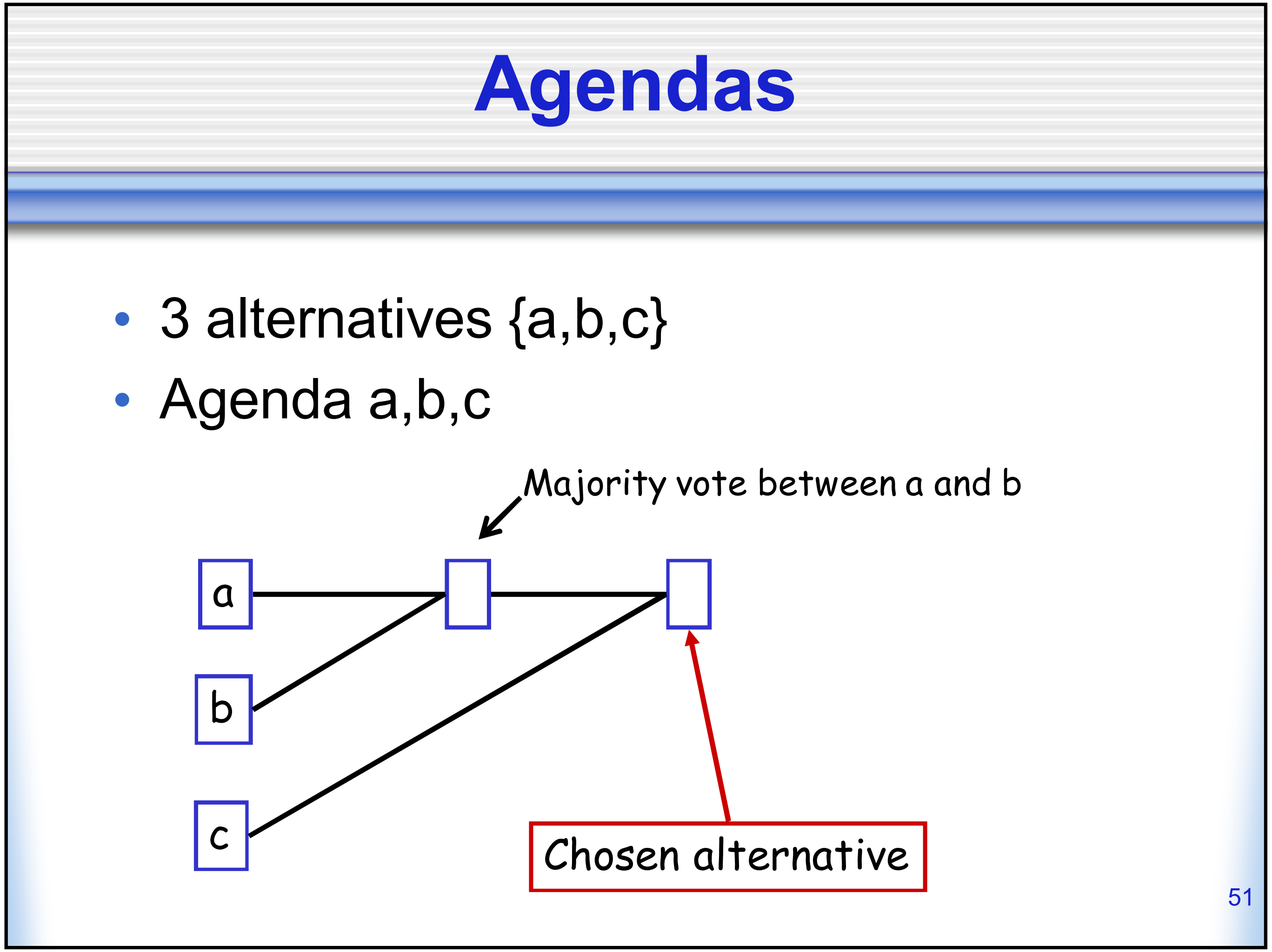
- 3 candidates
  - Lib, NDP, C
- 21 voters with the preferences
  - 10 Lib>NDP>C
  - 6 NDP>C>Lib
  - 5 C>NDP>Lib
- Result: Lib 10, NDP 6, C 5
  - But a majority of voters (11) prefer all other parties more than the Libs!



# What can we do?

- Majority system
  - Works well when there are 2 alternatives
  - Not great when there are more than 2 choices

- Proposal:
  - Organize a series of votes between 2 alternatives at a time
  - How this is organized is called an Agenda
    - Or a cup (often in sports)

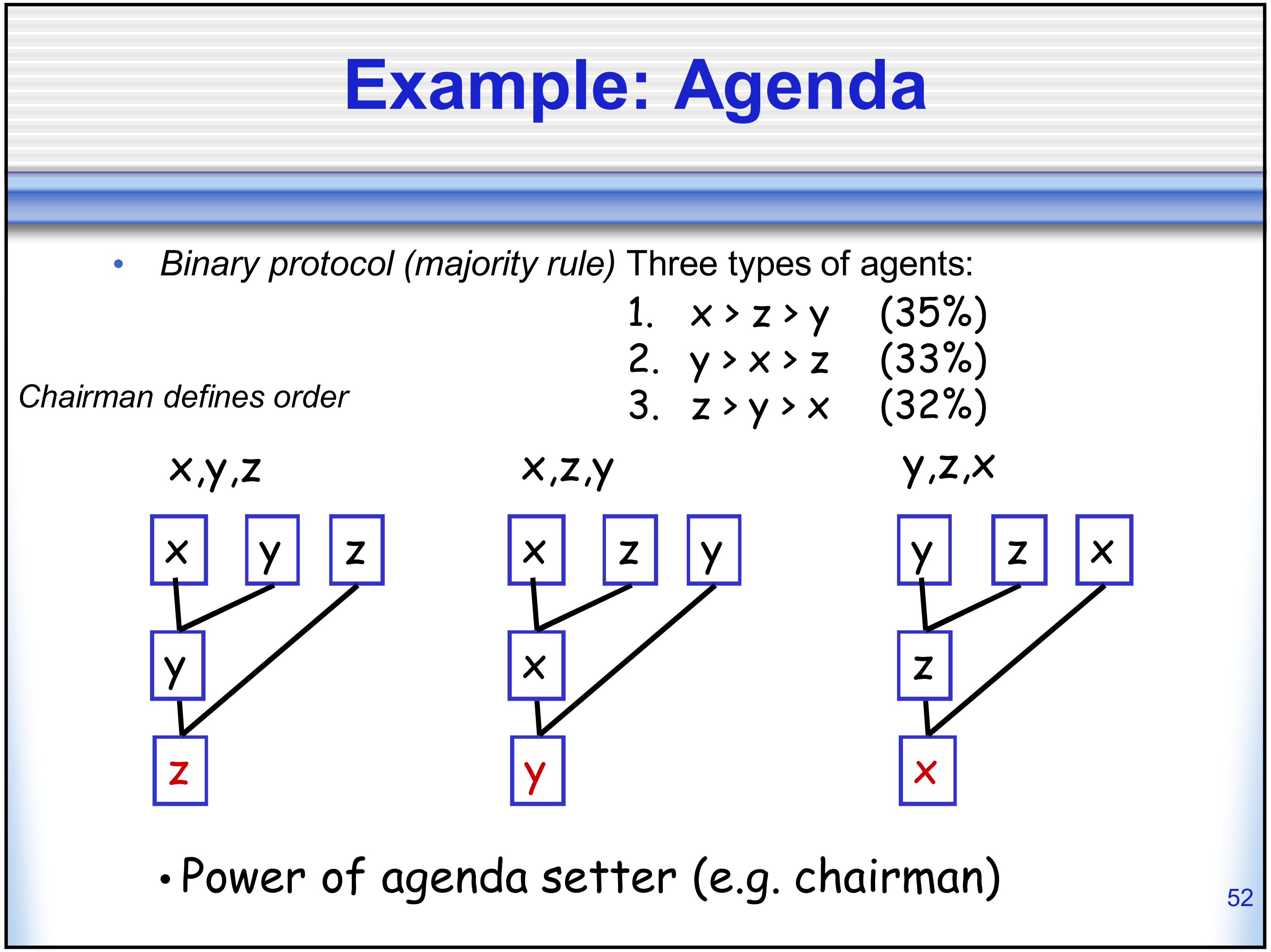


# Agendas

- 3 alternatives {a,b,c}
- Agenda a,b,c

Majority vote between a and b

Chosen alternative



# Example: Agenda

- *Binary protocol (majority rule)* Three types of agents:
  1. x > z > y (35%)
  2. y > x > z (33%)
  3. z > y > x (32%)

*Chairman defines order*

x,y,z

x y z

y

z

x,z,y

x z y

x

y

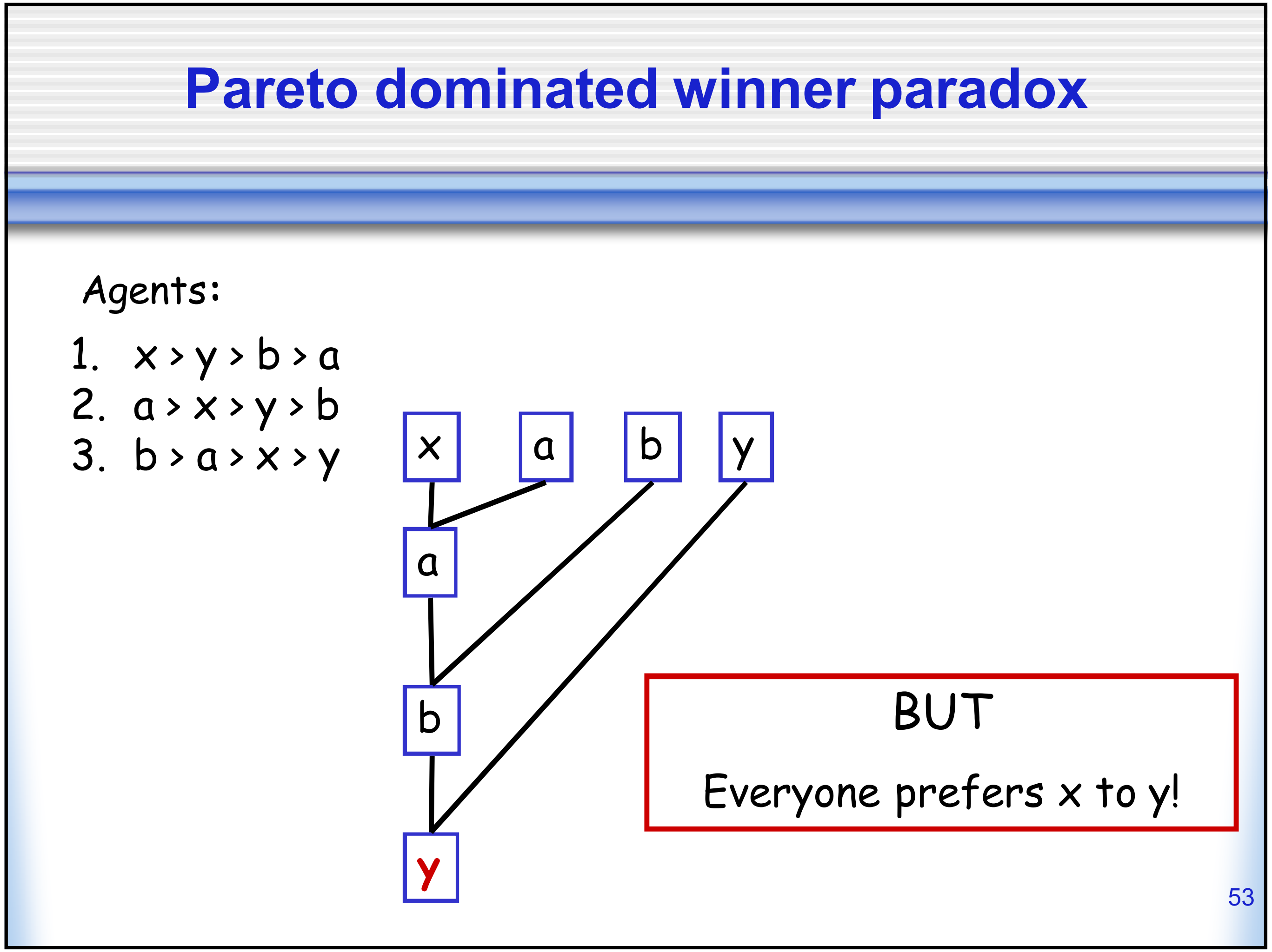
y,z,x

y z x

z

x

- Power of agenda setter (e.g. chairman)



## Pareto dominated winner paradox

Agents:

1. x > y > b > a
2. a > x > y > b
3. b > a > x > y

BUT

Everyone prefers x to y!

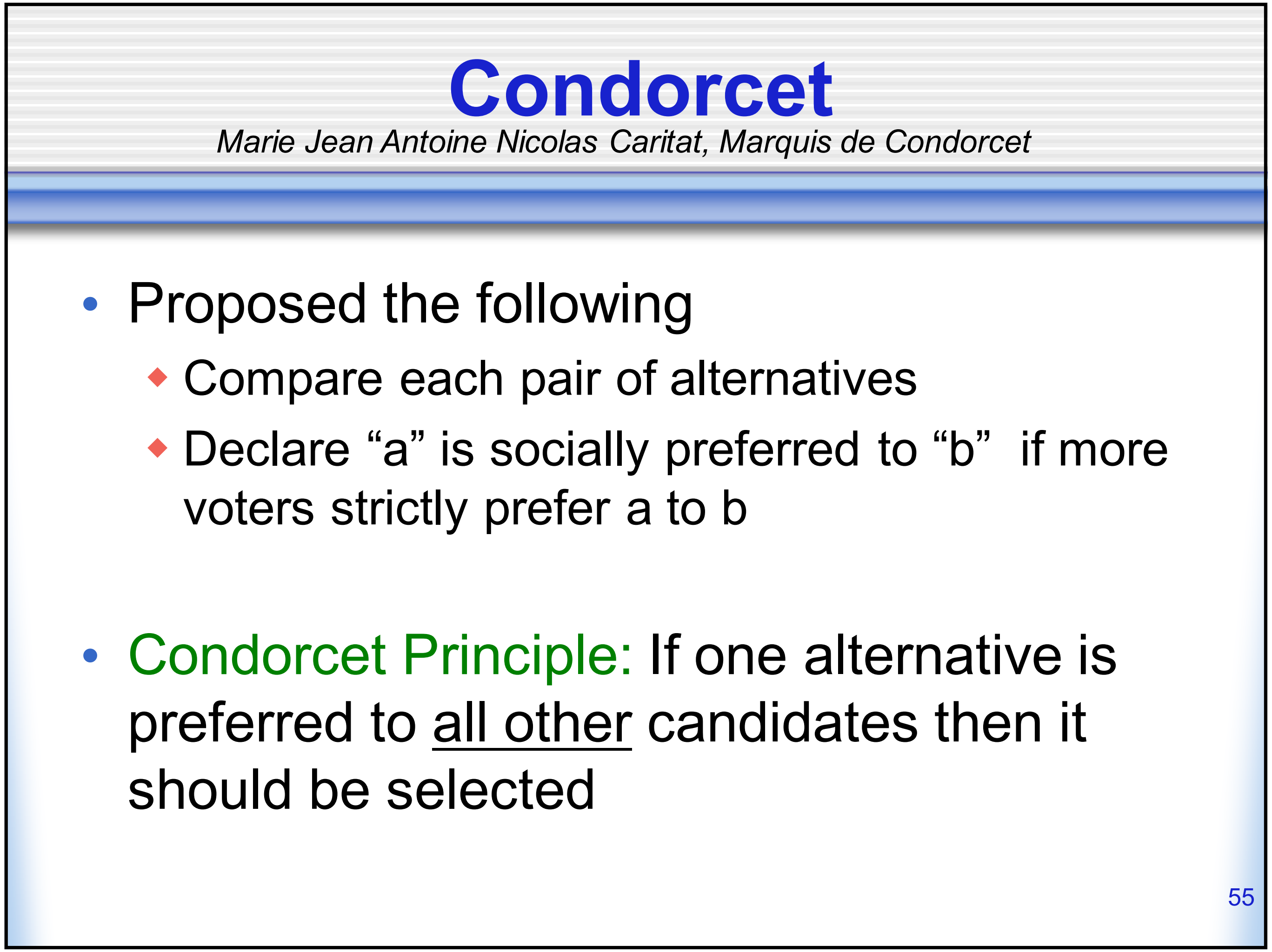
# Case 2: Agents specify their complete preferences

Maybe the problem was with the ballots!

Ballot

X>Y>Z

Now have more information



# Condorcet

*Marie Jean Antoine Nicolas Caritat, Marquis de Condorcet*

- Proposed the following
  - Compare each pair of alternatives
  - Declare “a” is socially preferred to “b” if more voters strictly prefer a to b

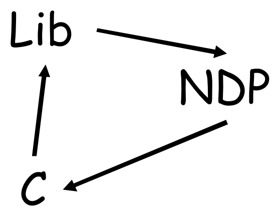
- Condorcet Principle: If one alternative is preferred to all other candidates then it should be selected

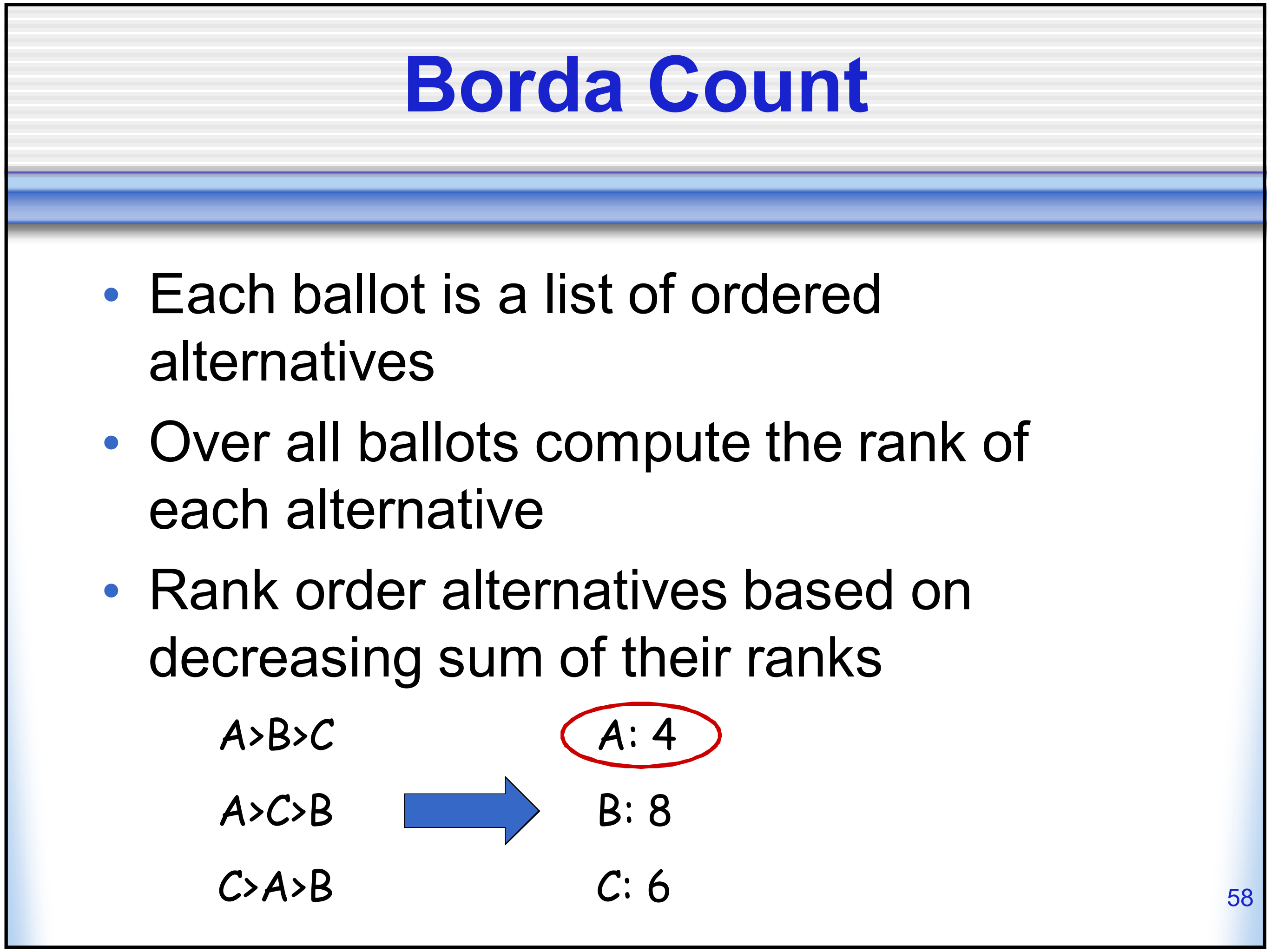
# Example: Condorcet

- 3 candidates
  - Lib, NDP, C
- 21 voters with the preferences
  - 10 Lib>NDP>C
  - 6 NDP>C>Lib
  - 5 C>NDP>Lib
- Result:
  - NDP win! (11/21 prefer them to Lib, 16/21 prefer them to C)

# A Problem

- 3 candidates
  - Lib, NDP, C
- 3 voters with the preferences
  - Lib>NDP>C
  - NDP>C>Lib
  - C>Lib>NDP
- Result:
  - No Condorcet Winner





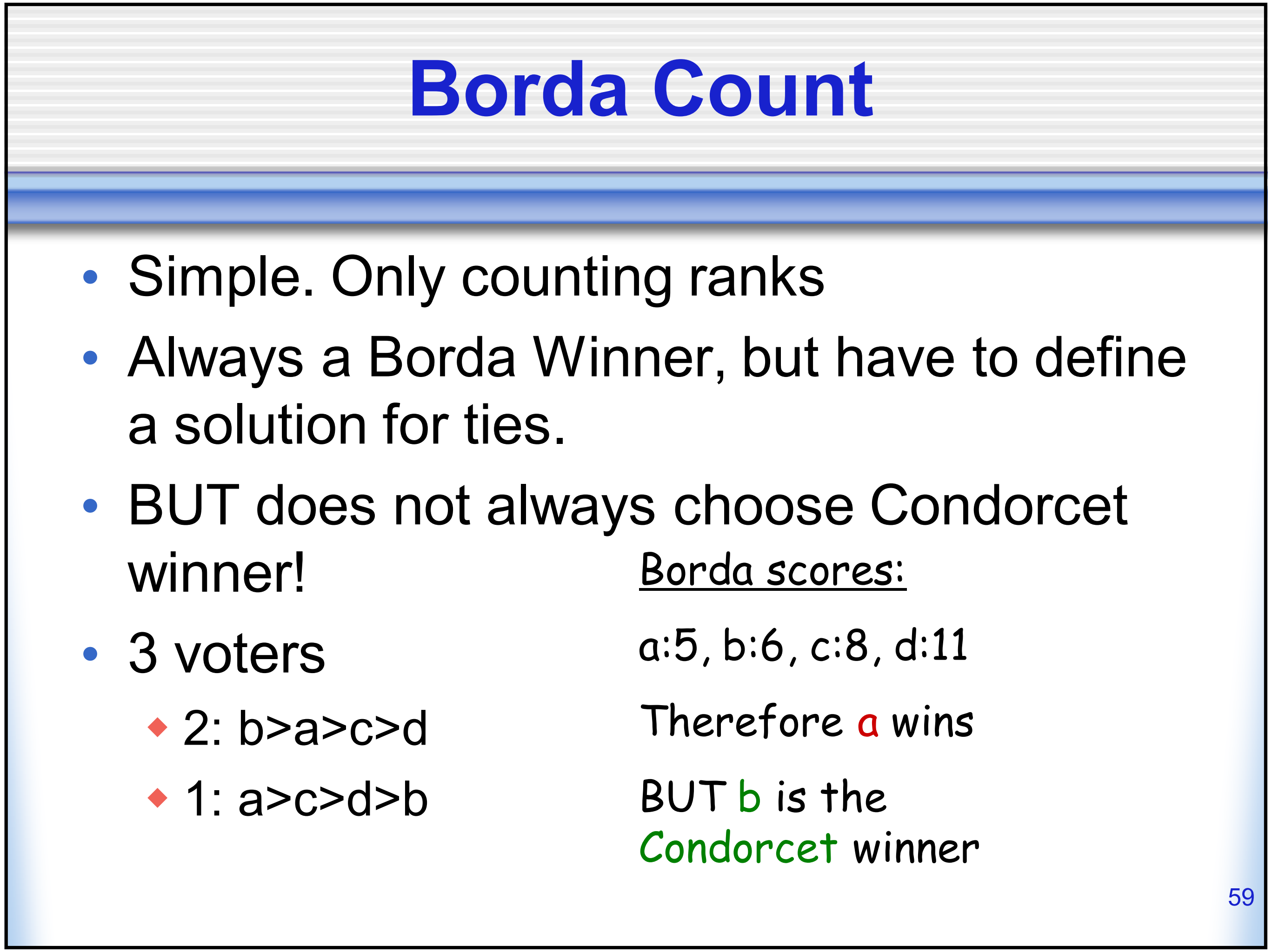
# Borda Count

- Each ballot is a list of ordered alternatives
- Over all ballots compute the rank of each alternative
- Rank order alternatives based on decreasing sum of their ranks

A>B>C A: 4

A>C>B → B: 8

C>A>B C: 6



# Borda Count

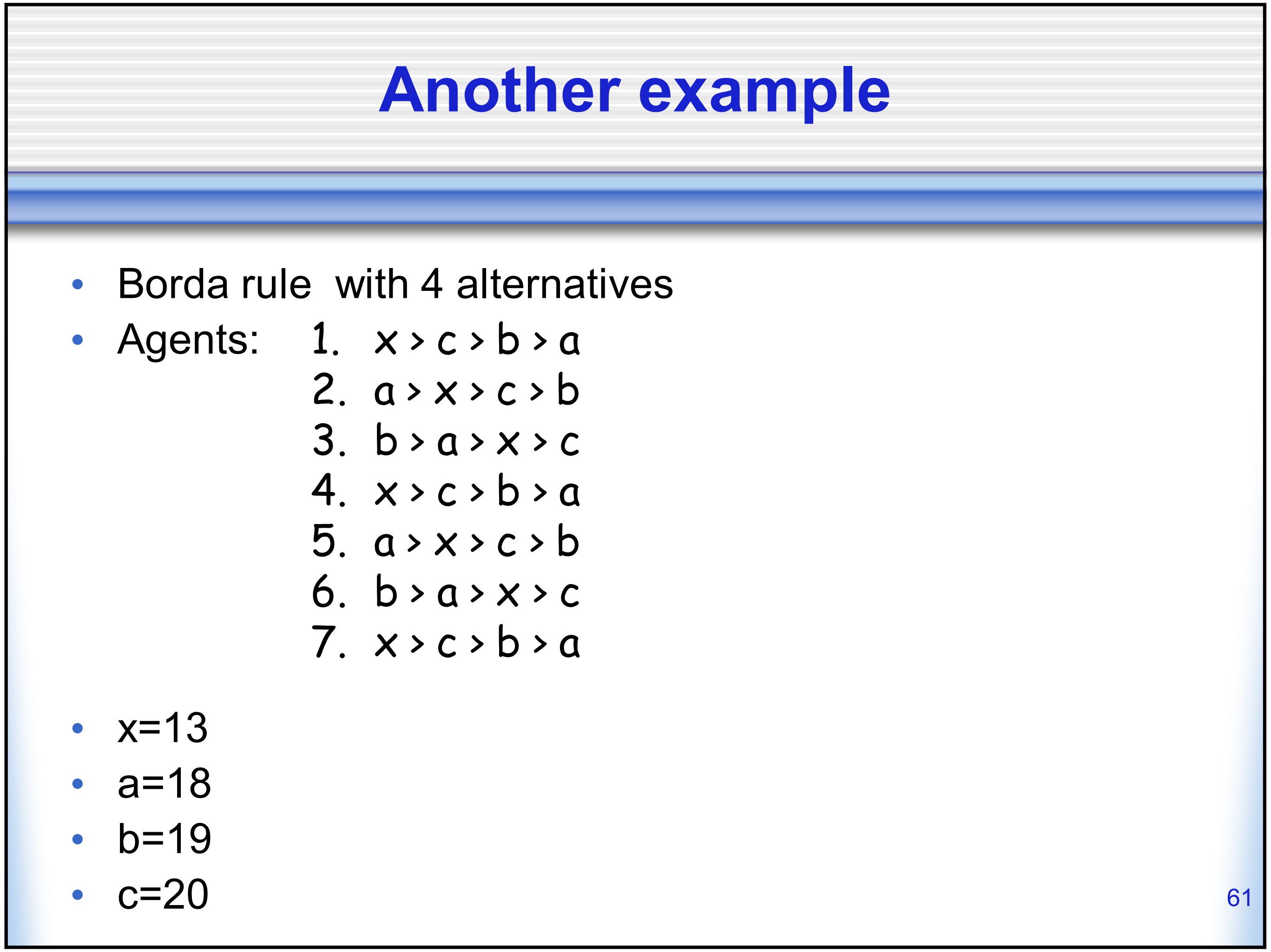
- Simple. Only counting ranks
- Always a Borda Winner, but have to define a solution for ties.
- BUT does not always choose Condorcet winner!
- 3 voters
  - 2: b>a>c>d
  - 1: a>c>d>b

<u>Borda scores:</u>

a:5, b:6, c:8, d:11

Therefore a wins

BUT b is the Condorcet winner



# Another example

- Borda rule with 4 alternatives
- Agents:
  1. $x > c > b > a$
  2. $a > x > c > b$
  3. $b > a > x > c$
  4. $x > c > b > a$
  5. $a > x > c > b$
  6. $b > a > x > c$
  7. $x > c > b > a$

- x=13
- a=18
- b=19
- c=20

# The winner is dropped

- X went out → Remove x:

1. x > c > b > a
2. a > x > c > b
3. b > a > x > c
4. x > c > b > a
5. a > x > c > b
6. b > a > x > c
7. x > c > b > a

1. c > b > a
2. a > c > b
3. b > a > c
4. c > b > a
5. a > c > b
6. b > a > c
7. c > b > a

- x=13, a=18, b=19, c=20
- c=13
- b=14
- a=15

Inverted order paradox

# Borda rule vulnerable to irrelevant alternatives

- Three types of agents:

| x | y |
|---|---|
| 35 | 70 |
| 66 | 33 |
| 64 | 32 |
| 165 | 135 |
| second | first |

1. x > y (35%)
2. y > x (33%)
3. y > x (32%)

- Borda winner is y

# Borda rule vulnerable to irrelevant alternatives

- Three types of agents:

1. x > z > y (35%)
2. y > x > z (33%)
3. z > y > x (32%)

| x | y |
|---|---|
| 35 | 70 |
| 66 | 33 |
| 64 | 32 |
| 165 | 135 |
| second | first |

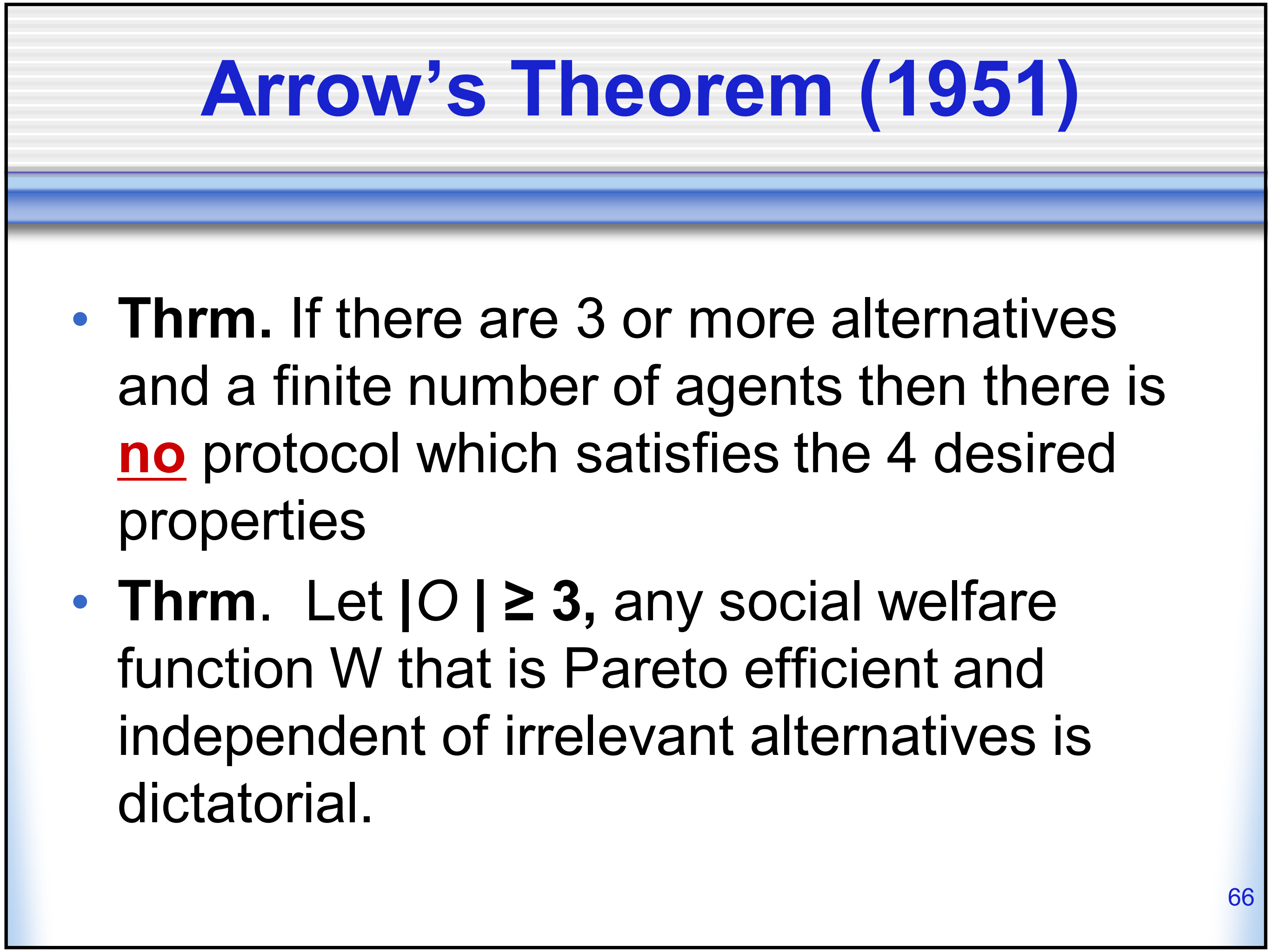
| x | y | z |
|---|---|---|
| 35 | 105 | 70 |
| 66 | 33 | 99 |
| 96 | 64 | 32 |
| 197 | 202 | 201 |
| first | third | second |

- Borda winner is y
- Add z Borda winner is x

*The social preferences between alternatives x and y depend only on the individual preferences between x and y*

# Desirable properties for a voting protocol

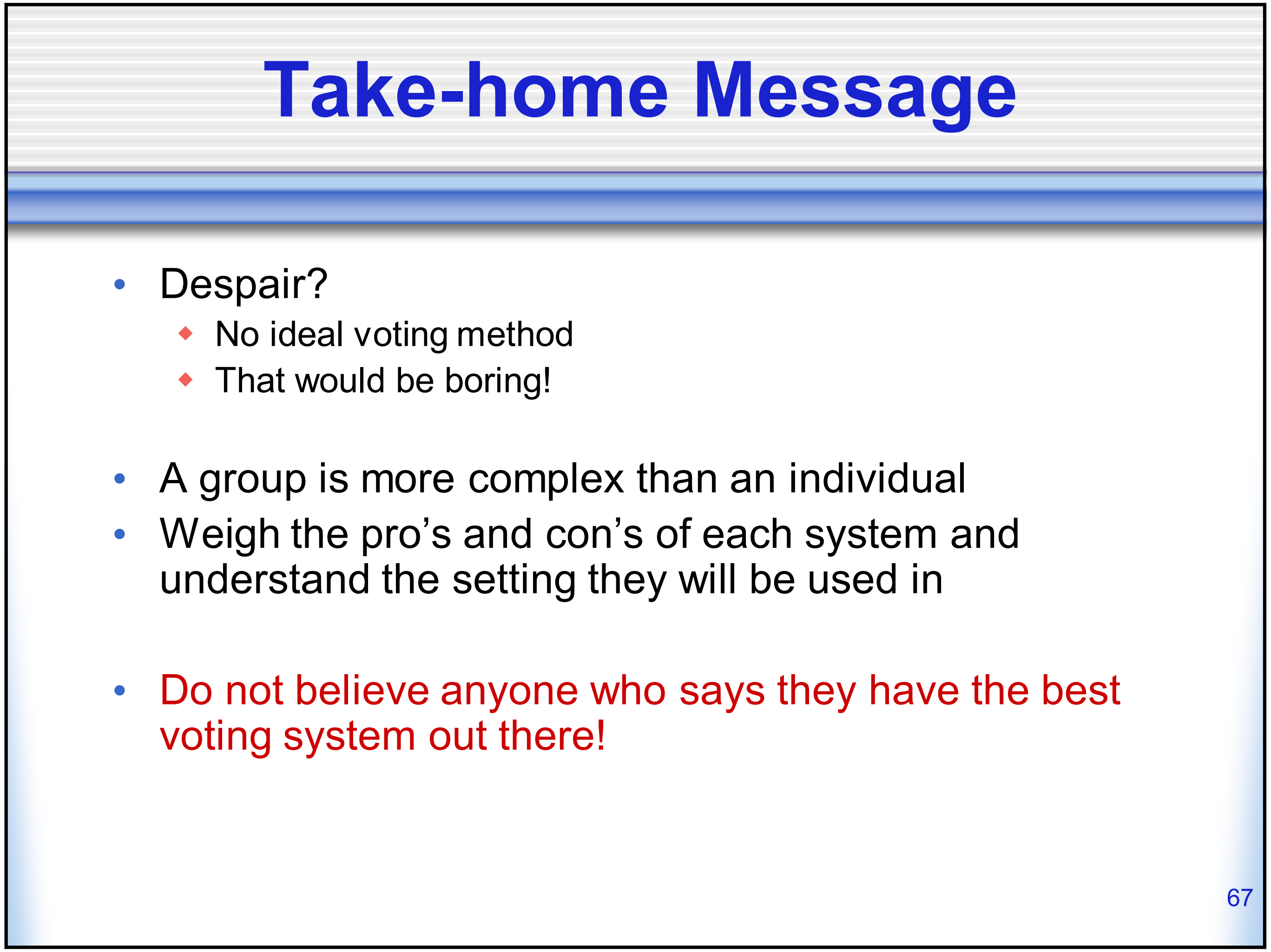
- No dictators
- Universality (unrestricted domain)
  - It should work with any set of preferences
- Independence of irrelevant alternatives
  - The comparison of two alternatives should depend only on their standings among agents’ preferences, not on the ranking of other alternatives
- Pareto efficient
  - If all agents prefer x to y then in the outcome x should be preferred to y



# Arrow’s Theorem (1951)

- **Thrm.** If there are 3 or more alternatives and a finite number of agents then there is **no** protocol which satisfies the 4 desired properties
- **Thrm**. Let **|***O* **| ≥ 3,** any social welfare function W that is Pareto efficient and independent of irrelevant alternatives is dictatorial.

66



# Take-home Message

- Despair?
  - No ideal voting method
  - That would be boring!
- A group is more complex than an individual
- Weigh the pro’s and con’s of each system and understand the setting they will be used in
- Do not believe anyone who says they have the best voting system out there!

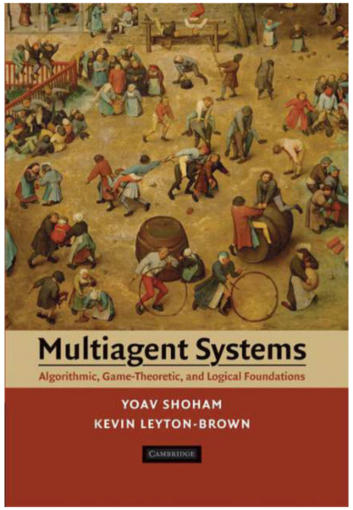
# Intelligent Autonomous Agents and Cognitive Robotics

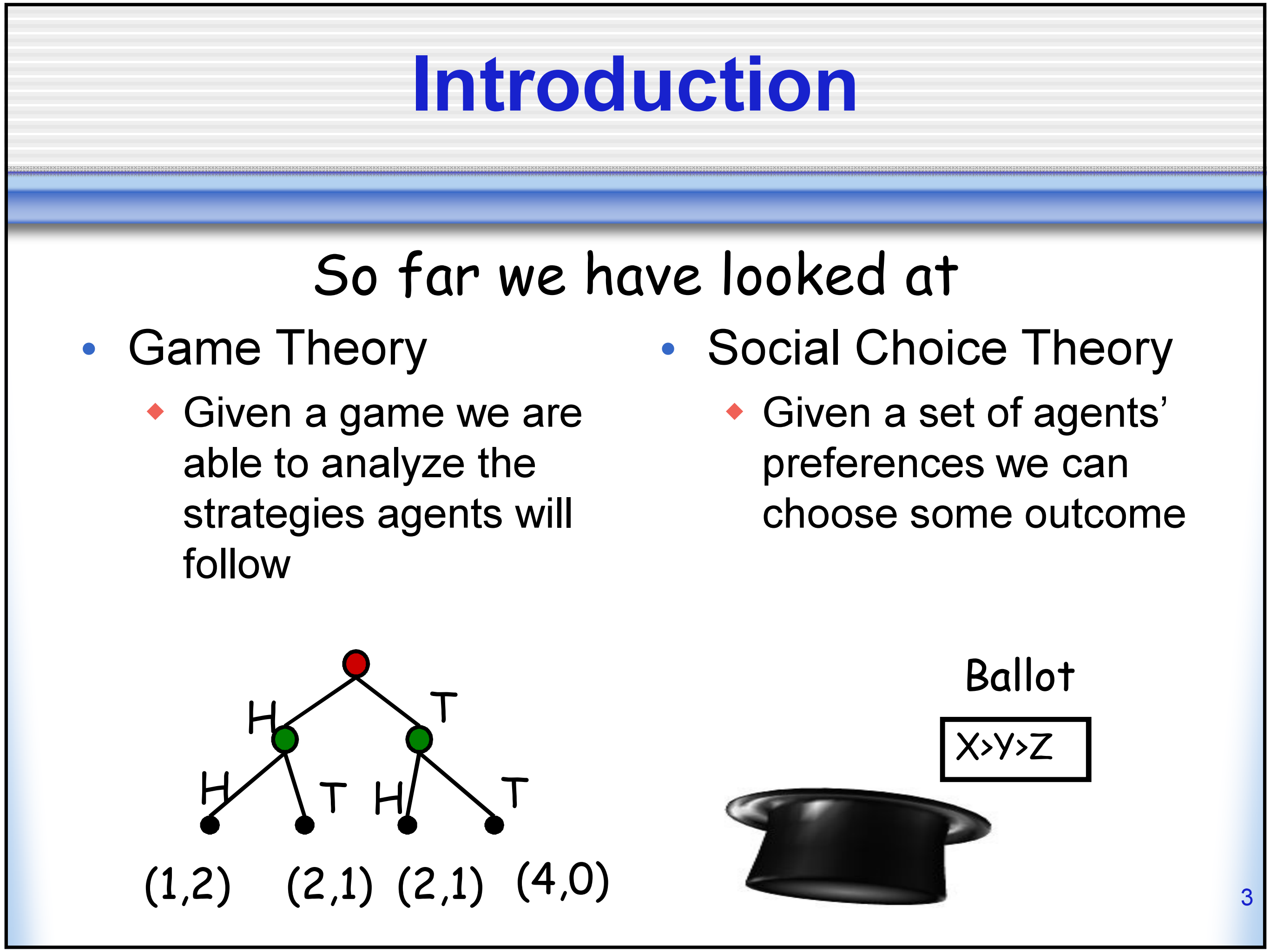
## Topic 12: Mechanism Design

Ralf Möller, Rainer Marrone
Hamburg University of Technology

# Acknowledgement

Material from CS 886
**Advanced Topics in AI**
**Electronic Market Design**
Kate Larson
Waterloo Univ.

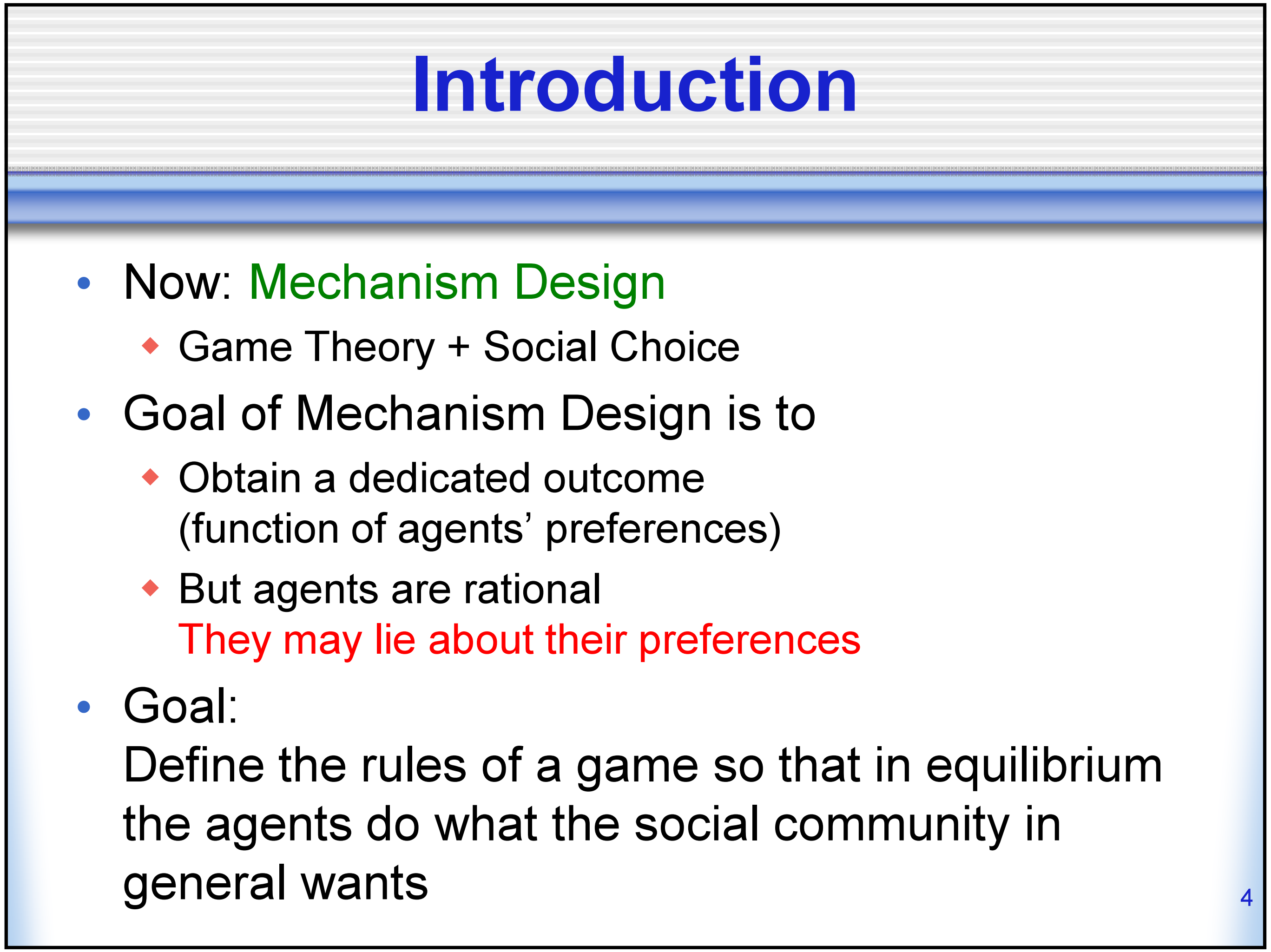




# Introduction

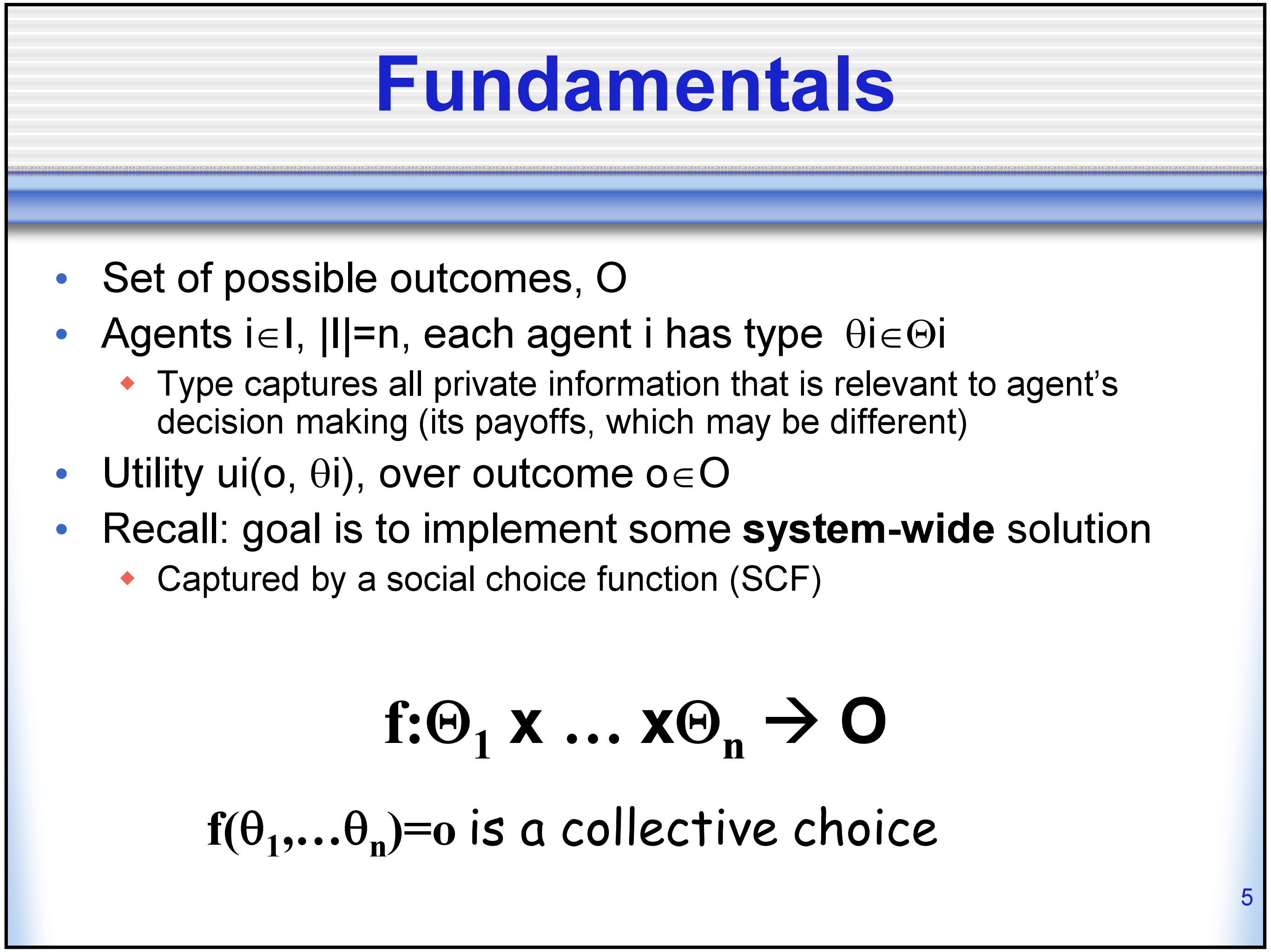
So far we have looked at

- Game Theory
  - Given a game we are able to analyze the strategies agents will follow
- Social Choice Theory
  - Given a set of agents' preferences we can choose some outcome



# Introduction

- Now: Mechanism Design
  - Game Theory + Social Choice
- Goal of Mechanism Design is to
  - Obtain a dedicated outcome
    (function of agents' preferences)
  - But agents are rational
    They may lie about their preferences
- Goal:
  Define the rules of a game so that in equilibrium the agents do what the social community in general wants

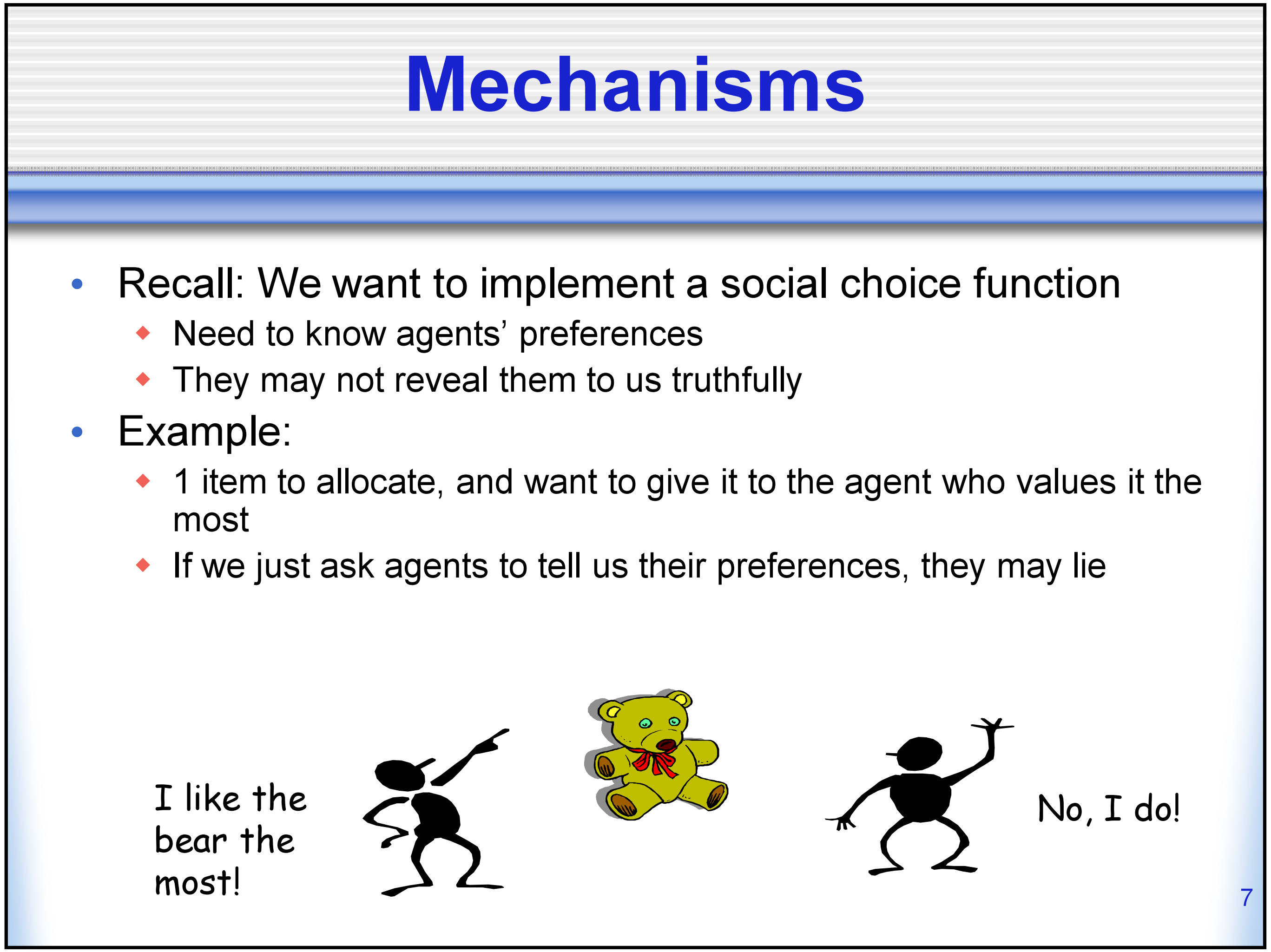


# Fundamentals

- Set of possible outcomes, O
- Agents $i \in I$, $|I|=n$, each agent i has type $\theta i \in \Theta i$
  - Type captures all private information that is relevant to agent’s decision making (its payoffs, which may be different)
- Utility $ui(o, \theta i)$, over outcome $o \in O$
- Recall: goal is to implement some **system-wide** solution
  - Captured by a social choice function (SCF)

$$\mathbf{f:\Theta_1 \text{ x } \dots \text{ x}\Theta_n \rightarrow O}$$

$f(\theta_1,\dots\theta_n)=o$ is a collective choice



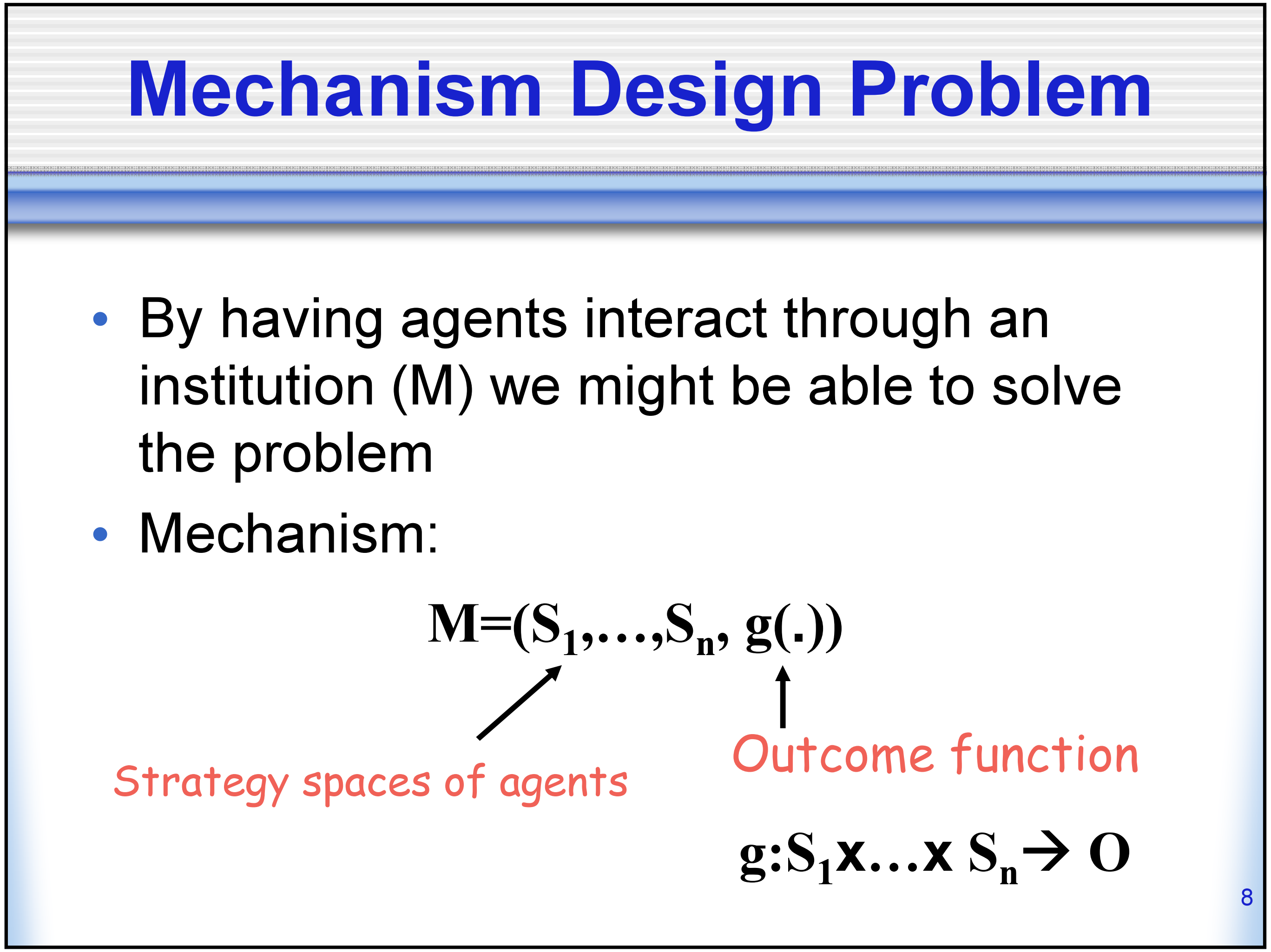
# Mechanisms

- Recall: We want to implement a social choice function
  - Need to know agents’ preferences
  - They may not reveal them to us truthfully
- Example:
  - 1 item to allocate, and want to give it to the agent who values it the most
  - If we just ask agents to tell us their preferences, they may lie

I like the bear the most!

No, I do!

7



# Mechanism Design Problem

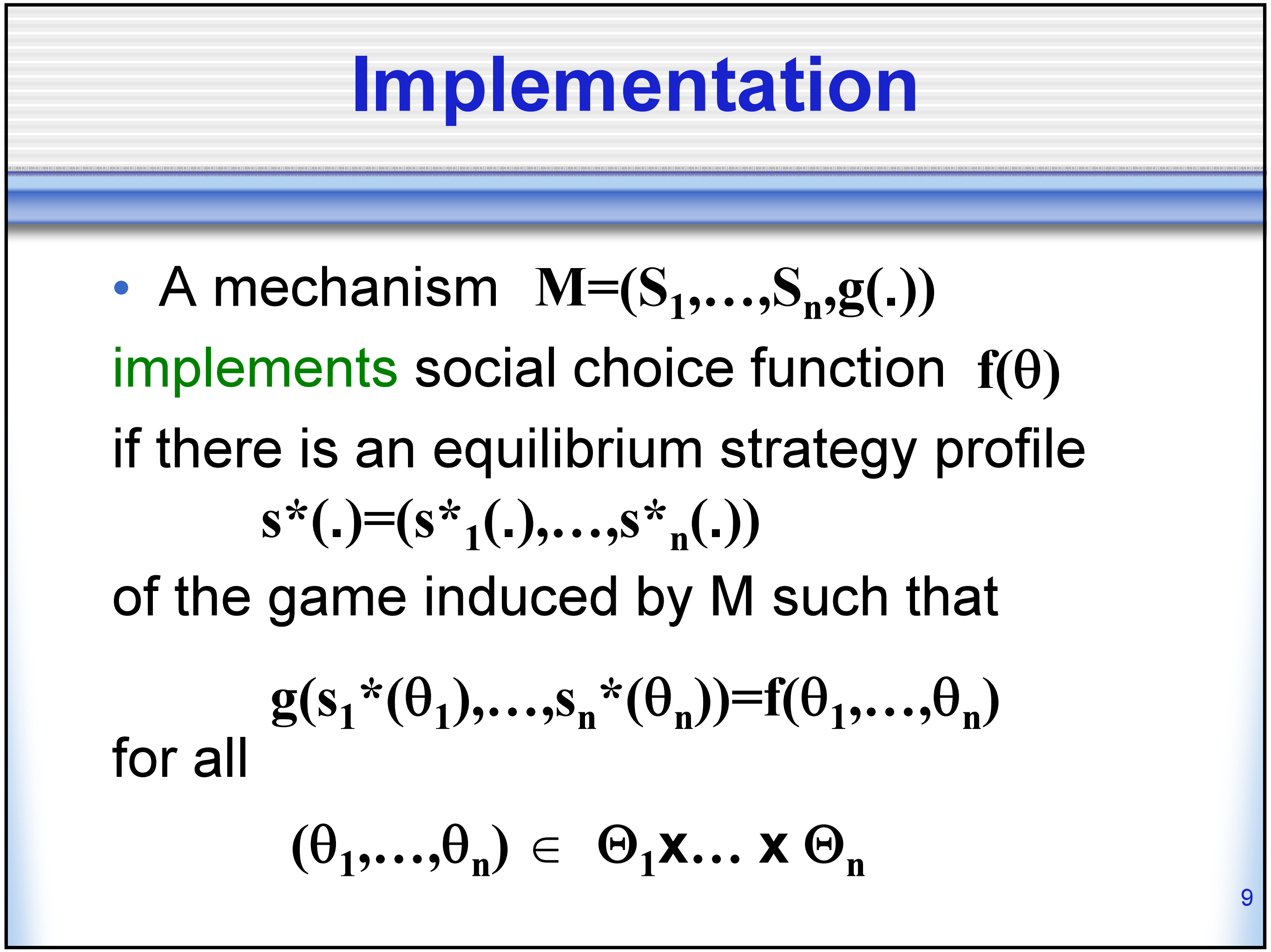
- By having agents interact through an institution (M) we might be able to solve the problem
- Mechanism:

$$M=(S_1,\ldots,S_n,\ g(.))$$

Strategy spaces of agents

Outcome function

$$g:S_1 x\ldots x\ S_n \rightarrow O$$



# Implementation

- A mechanism $\mathbf{M=(S_1,\ldots,S_n,g(.))}$

implements social choice function $\mathbf{f(\theta)}$

if there is an equilibrium strategy profile

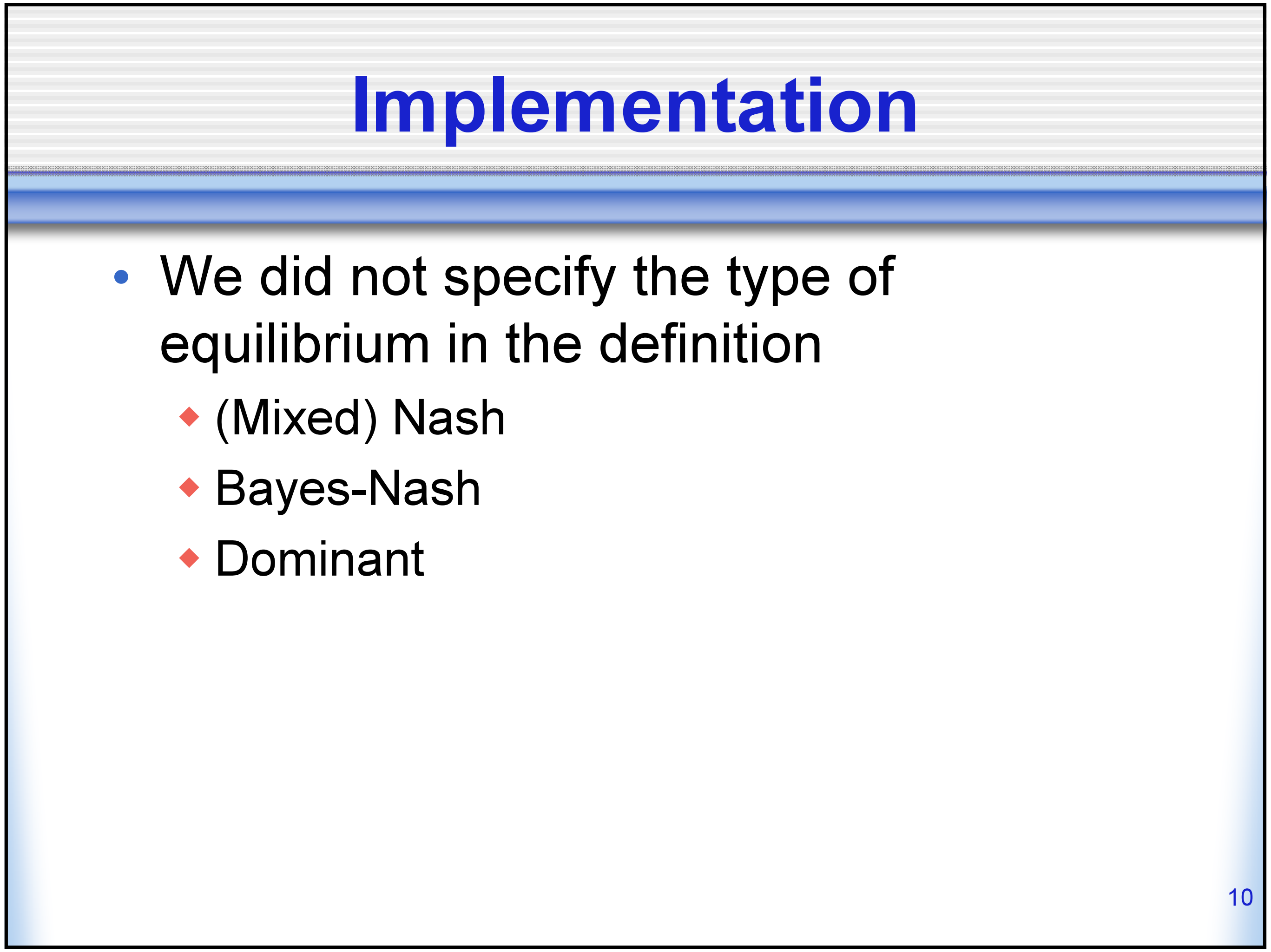
$$\mathbf{s^*(.)=(s^*_1(.),\ldots,s^*_n(.))}$$

of the game induced by M such that

$$\mathbf{g(s_1{}^*(\theta_1),\ldots,s_n{}^*(\theta_n))=f(\theta_1,\ldots,\theta_n)}$$

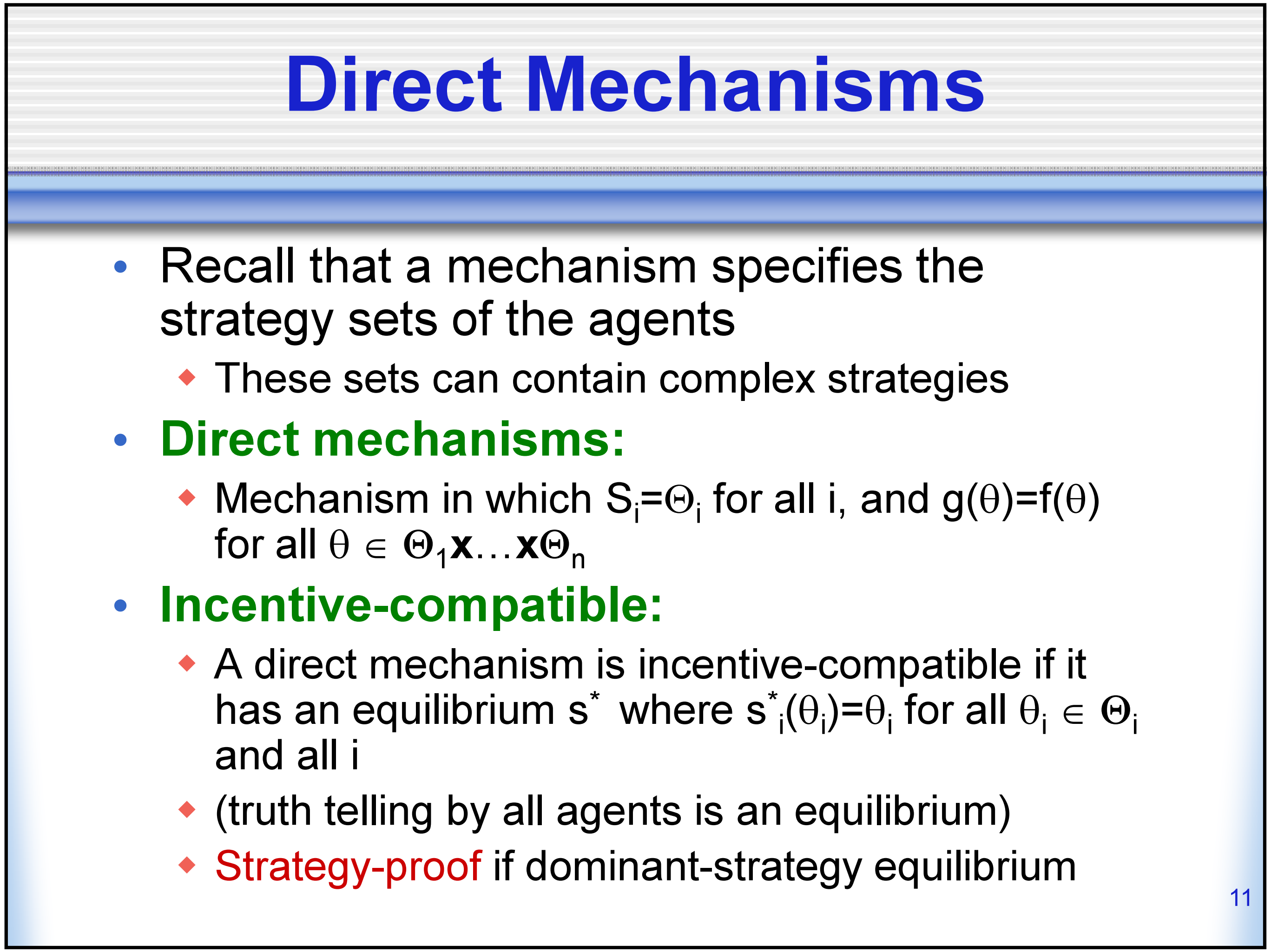
for all

$$\mathbf{(\theta_1,\ldots,\theta_n) \in \Theta_1 x\ldots x\, \Theta_n}$$



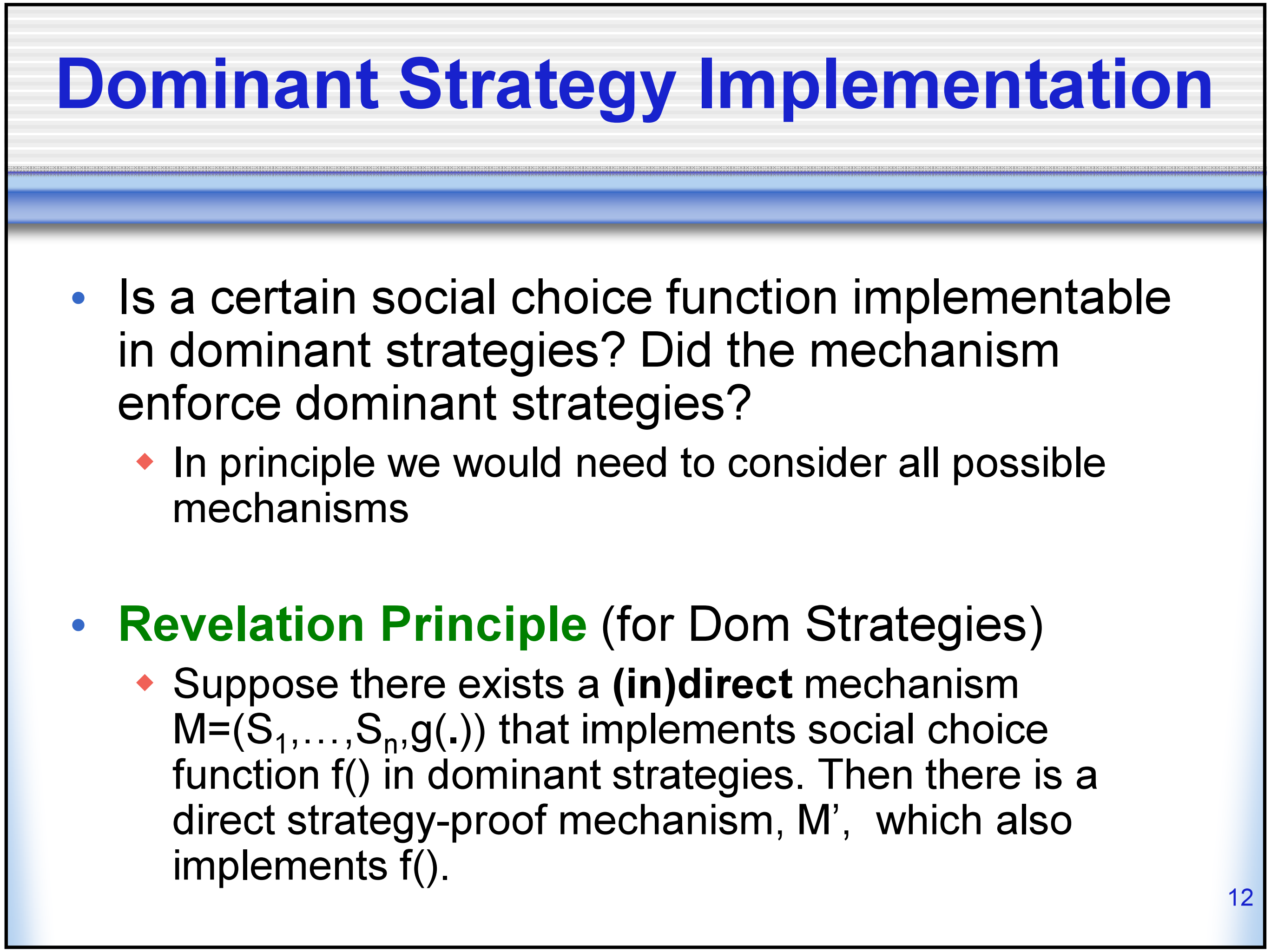
# Implementation

- We did not specify the type of equilibrium in the definition
  - (Mixed) Nash
  - Bayes-Nash
  - Dominant



# Direct Mechanisms

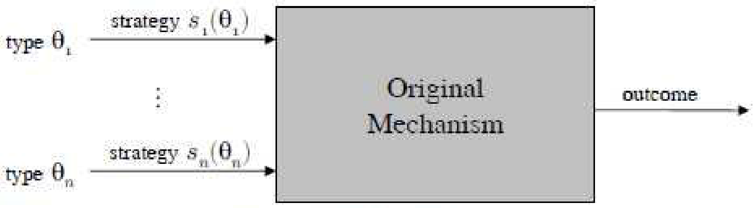
- Recall that a mechanism specifies the strategy sets of the agents
  - These sets can contain complex strategies
- **Direct mechanisms:**
  - Mechanism in which $S_i = \Theta_i$ for all i, and $g(\theta) = f(\theta)$ for all $\theta \in \Theta_1 \mathbf{x} \ldots \mathbf{x} \Theta_n$
- **Incentive-compatible:**
  - A direct mechanism is incentive-compatible if it has an equilibrium $s^*$ where $s^*_i(\theta_i) = \theta_i$ for all $\theta_i \in \Theta_i$ and all i
  - (truth telling by all agents is an equilibrium)
  - Strategy-proof if dominant-strategy equilibrium



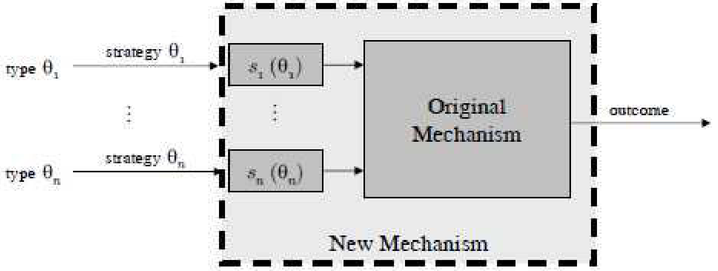
# Dominant Strategy Implementation

- Is a certain social choice function implementable in dominant strategies? Did the mechanism enforce dominant strategies?
  - In principle we would need to consider all possible mechanisms
- **Revelation Principle** (for Dom Strategies)
  - Suppose there exists a **(in)direct** mechanism M=($S_1$,…,$S_n$,g(**.**)) that implements social choice function f() in dominant strategies. Then there is a direct strategy-proof mechanism, M', which also implements f().

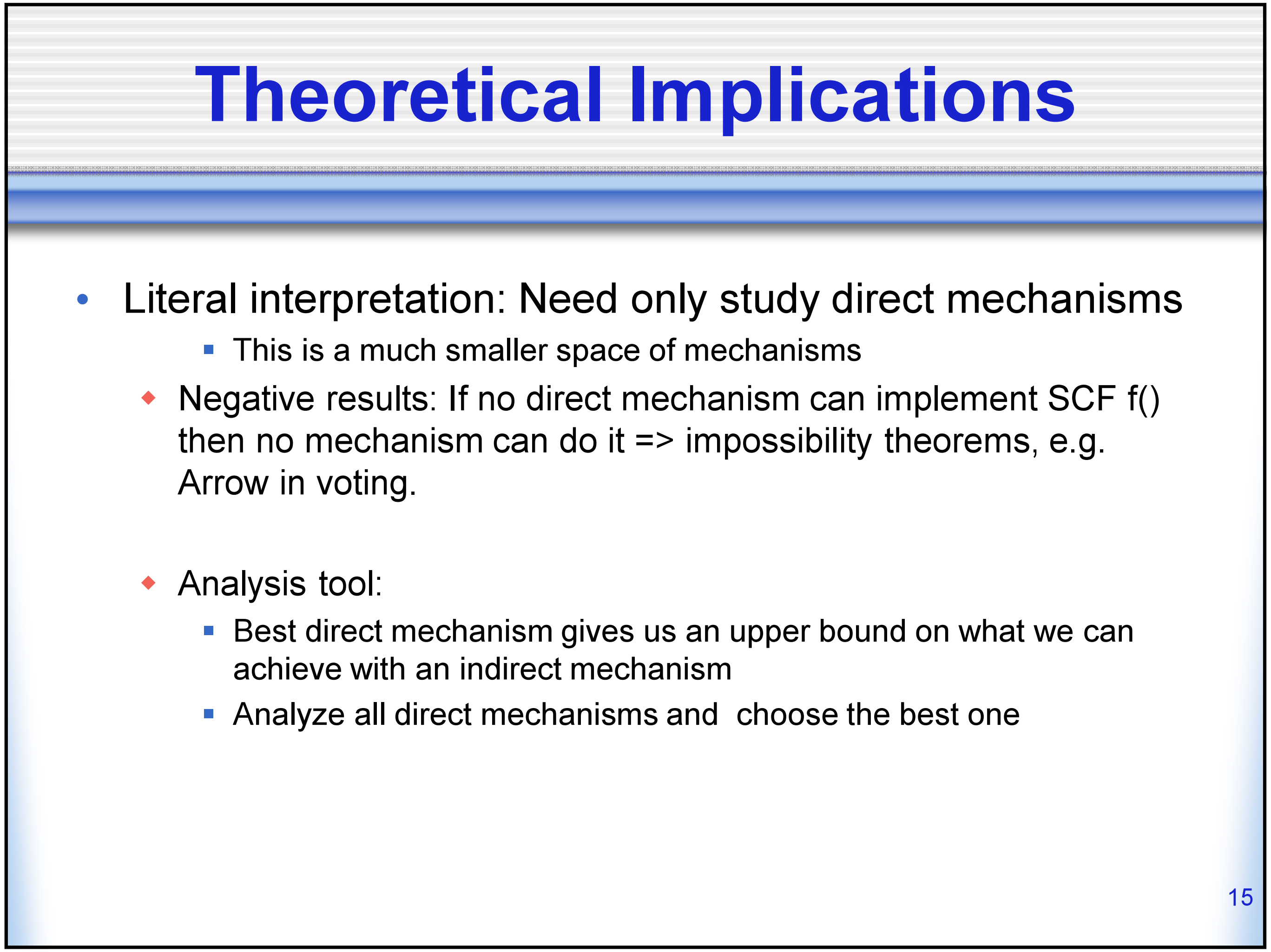
# Revelation Principle: Intuition



(a) Revelation principle: original mechanism

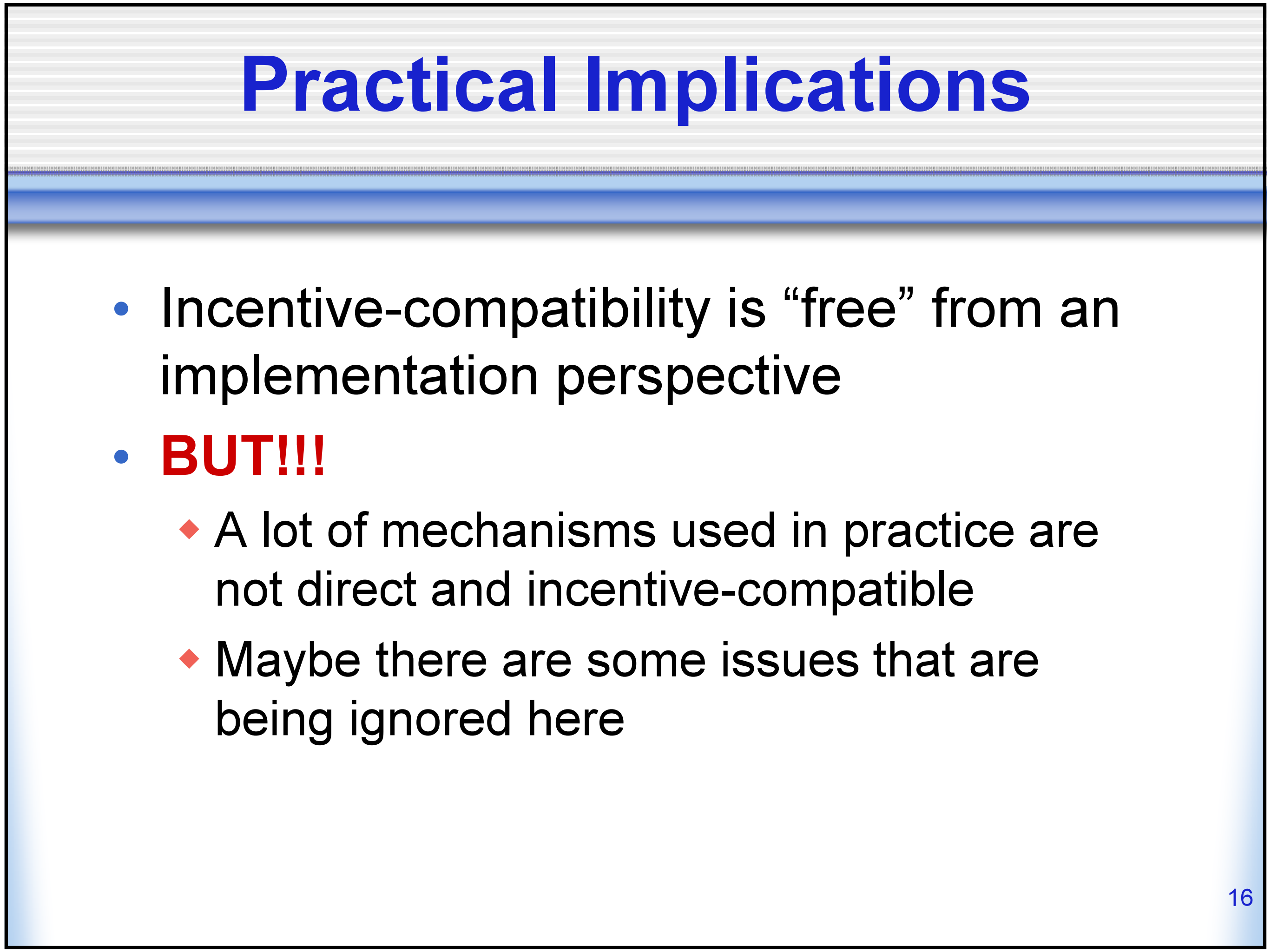


(b) Revelation principle: new mechanism



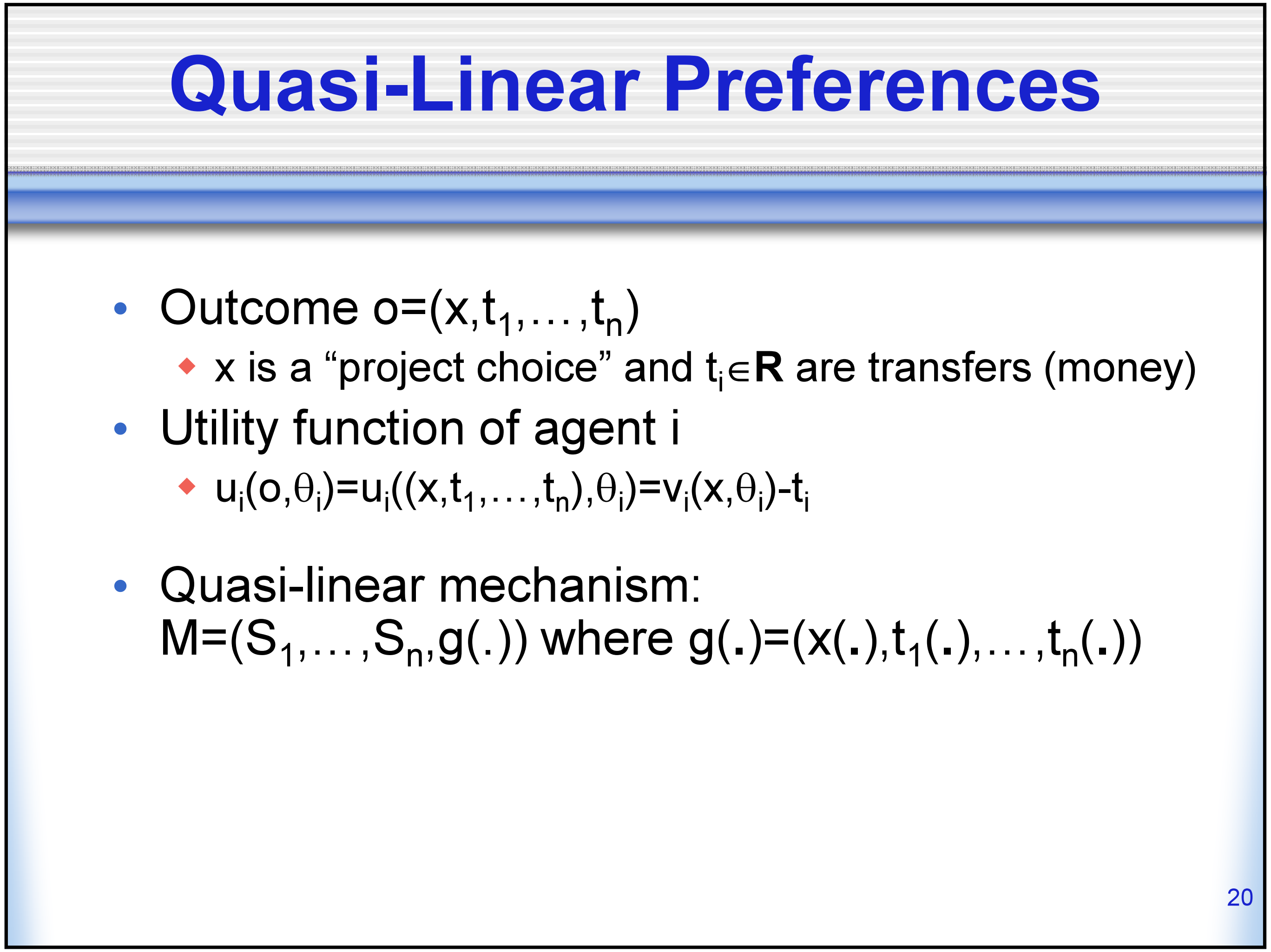
# Theoretical Implications

- Literal interpretation: Need only study direct mechanisms
    - This is a much smaller space of mechanisms
  - Negative results: If no direct mechanism can implement SCF f() then no mechanism can do it => impossibility theorems, e.g. Arrow in voting.
  - Analysis tool:
    - Best direct mechanism gives us an upper bound on what we can achieve with an indirect mechanism
    - Analyze all direct mechanisms and choose the best one



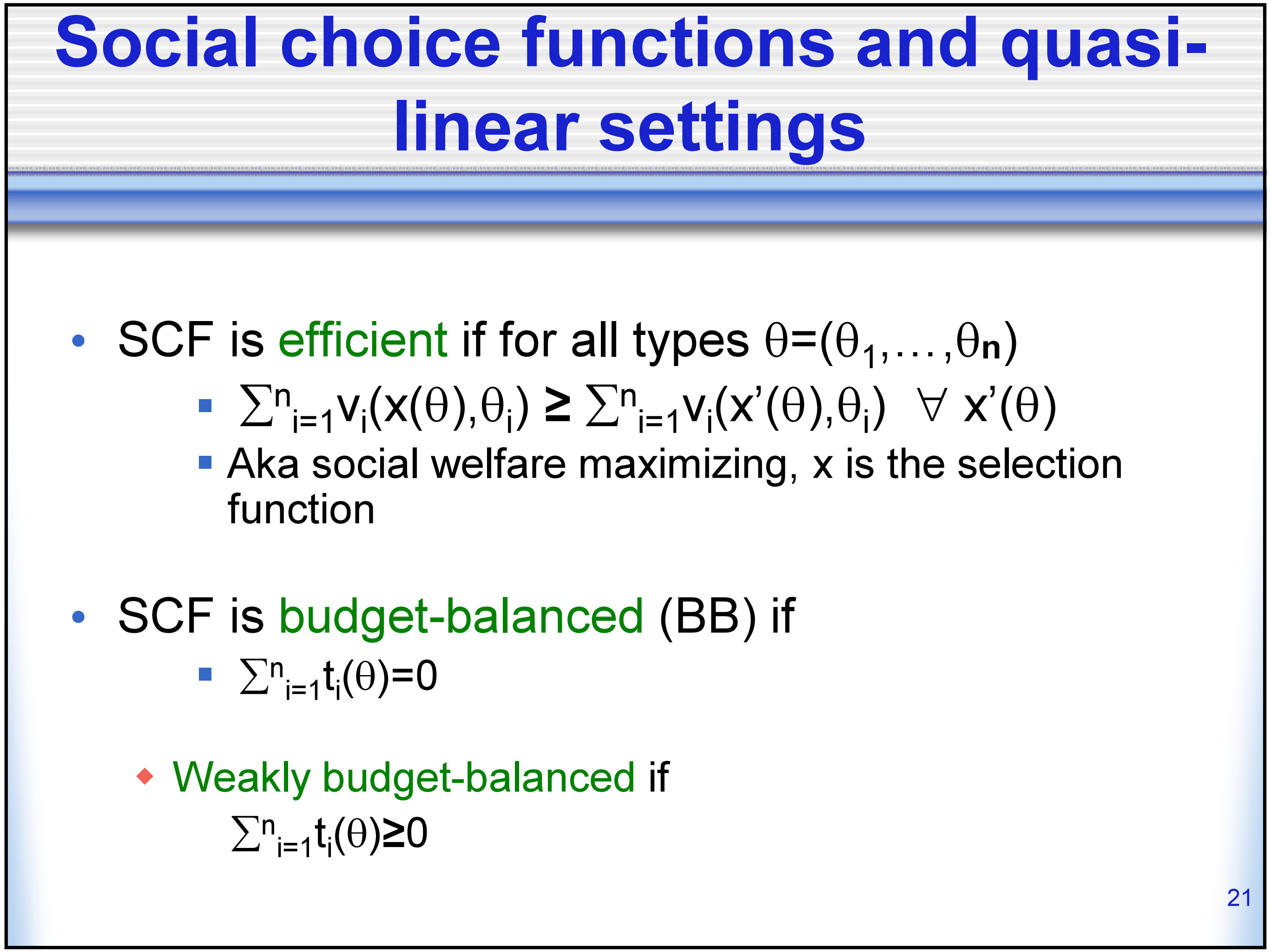
# Practical Implications

- Incentive-compatibility is “free” from an implementation perspective
- **BUT!!!**
  - A lot of mechanisms used in practice are not direct and incentive-compatible
  - Maybe there are some issues that are being ignored here



# Quasi-Linear Preferences

- Outcome $o=(x,t_1,\ldots,t_n)$
  - $x$ is a “project choice” and $t_i \in \mathbf{R}$ are transfers (money)
- Utility function of agent i
  - $u_i(o,\theta_i)=u_i((x,t_1,\ldots,t_n),\theta_i)=v_i(x,\theta_i)-t_i$
- Quasi-linear mechanism:
  $M=(S_1,\ldots,S_n,g(.))$ where $g(\mathbf{.})=(x(\mathbf{.}),t_1(\mathbf{.}),\ldots,t_n(\mathbf{.}))$



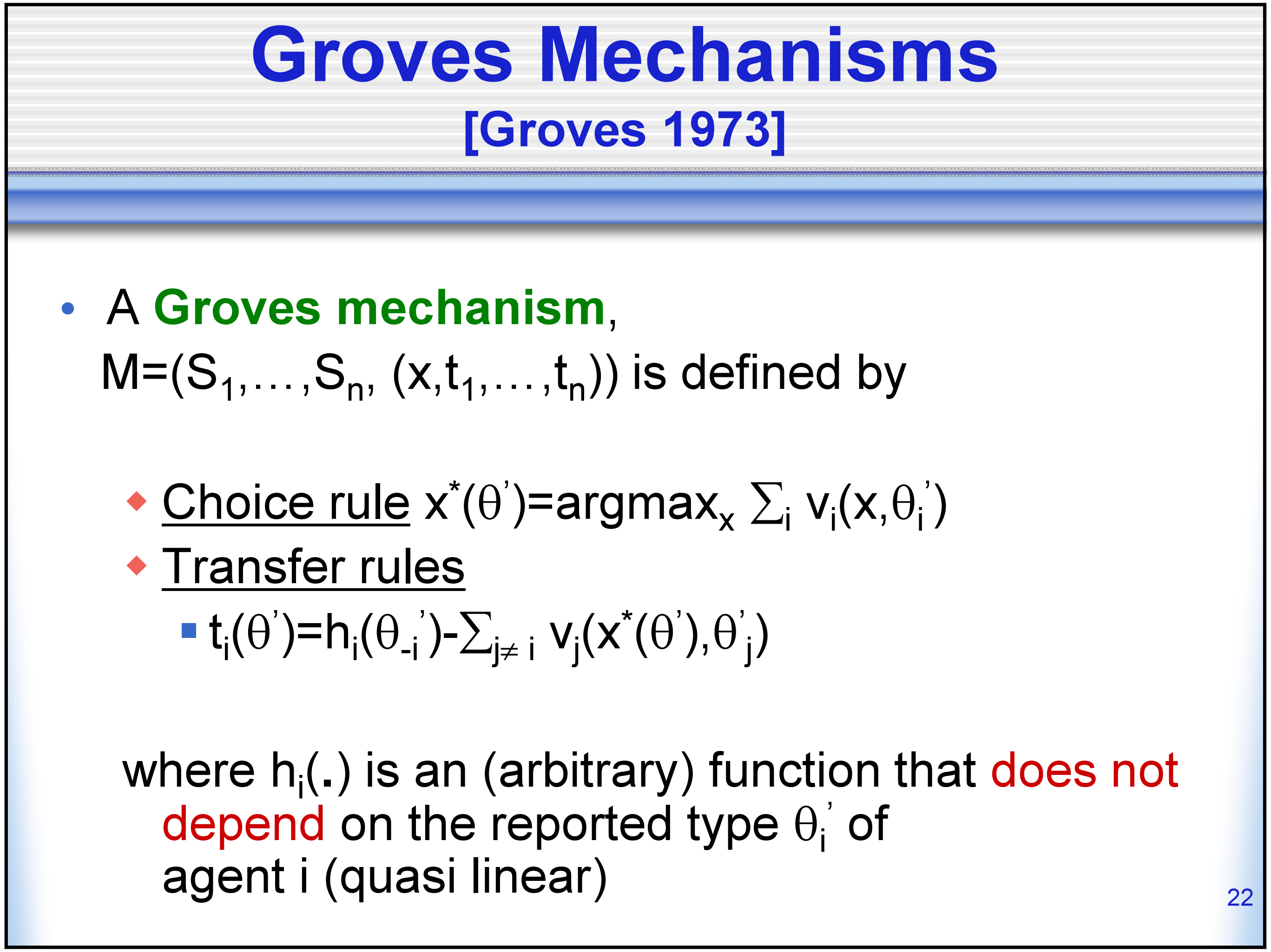
# Social choice functions and quasi-linear settings

- SCF is efficient if for all types $\theta=(\theta_1,\ldots,\theta_n)$
  - $\sum^n_{i=1} v_i(x(\theta),\theta_i) \geq \sum^n_{i=1} v_i(x'(\theta),\theta_i) \;\; \forall\, x'(\theta)$
  - Aka social welfare maximizing, x is the selection function

- SCF is budget-balanced (BB) if
  - $\sum^n_{i=1} t_i(\theta)=0$

◆ Weakly budget-balanced if

$\sum^n_{i=1} t_i(\theta)\geq 0$

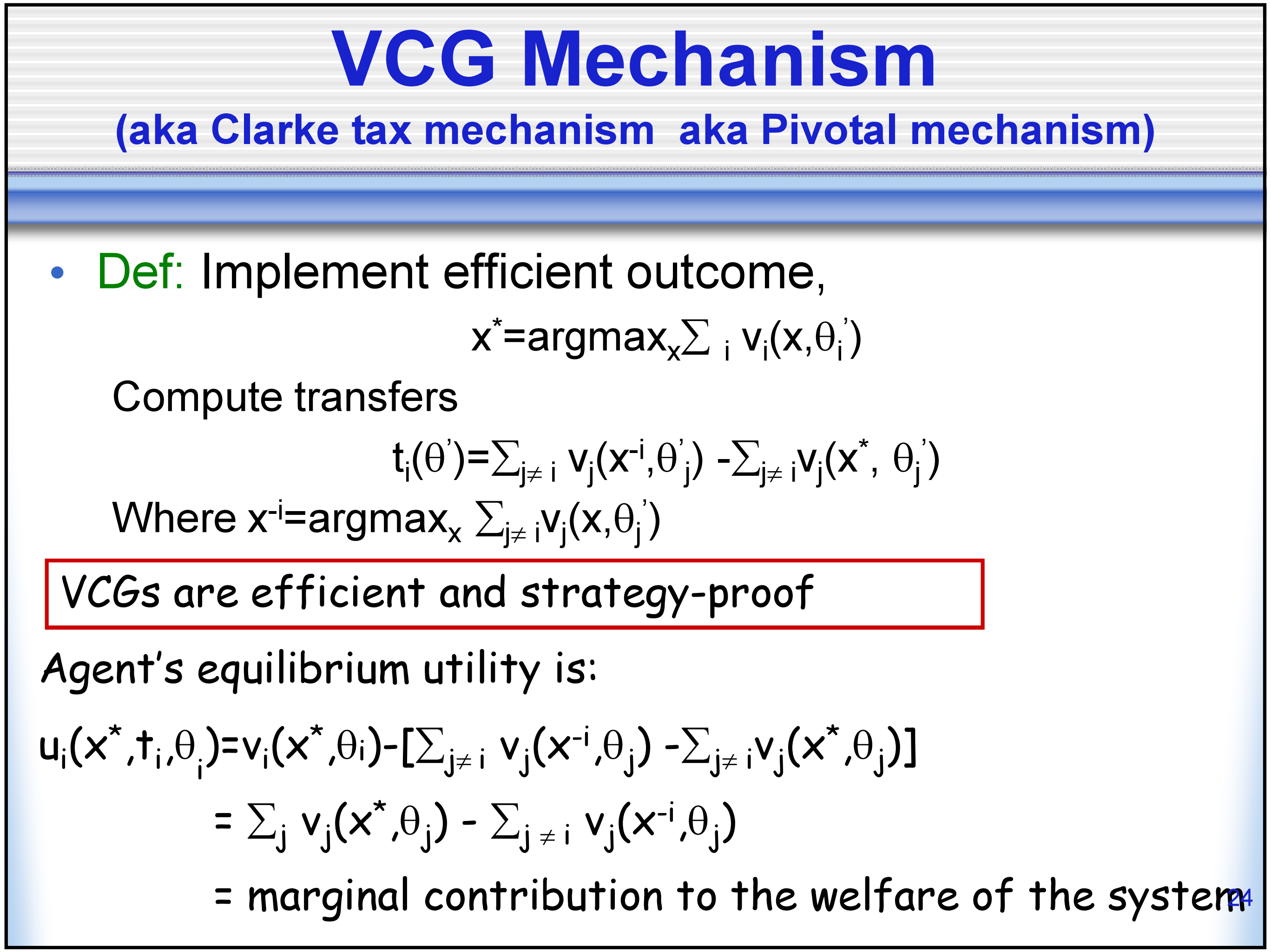


# Groves Mechanisms

## [Groves 1973]

- A **Groves mechanism**,
  $M=(S_1,\dots,S_n, (x,t_1,\dots,t_n))$ is defined by
  - Choice rule $x^*(\theta')=\text{argmax}_x \sum_i v_i(x,\theta_i')$
  - Transfer rules
    - $t_i(\theta')=h_i(\theta_{-i}')-\sum_{j\neq i} v_j(x^*(\theta'),\theta'_j)$

where $h_i(.)$ is an (arbitrary) function that does not depend on the reported type $\theta_i'$ of agent i (quasi linear)



# VCG Mechanism

## (aka Clarke tax mechanism aka Pivotal mechanism)

- Def: Implement efficient outcome,

$$x^*=\text{argmax}_x \Sigma_{\ i}\ v_i(x,\theta_i')$$

Compute transfers

$$t_i(\theta')=\Sigma_{j\neq i}\ v_j(x^{-i},\theta'_j)\ -\Sigma_{j\neq i} v_j(x^*,\ \theta_j')$$

Where $x^{-i}=\text{argmax}_x\ \Sigma_{j\neq i} v_j(x,\theta_j')$

VCGs are efficient and strategy-proof

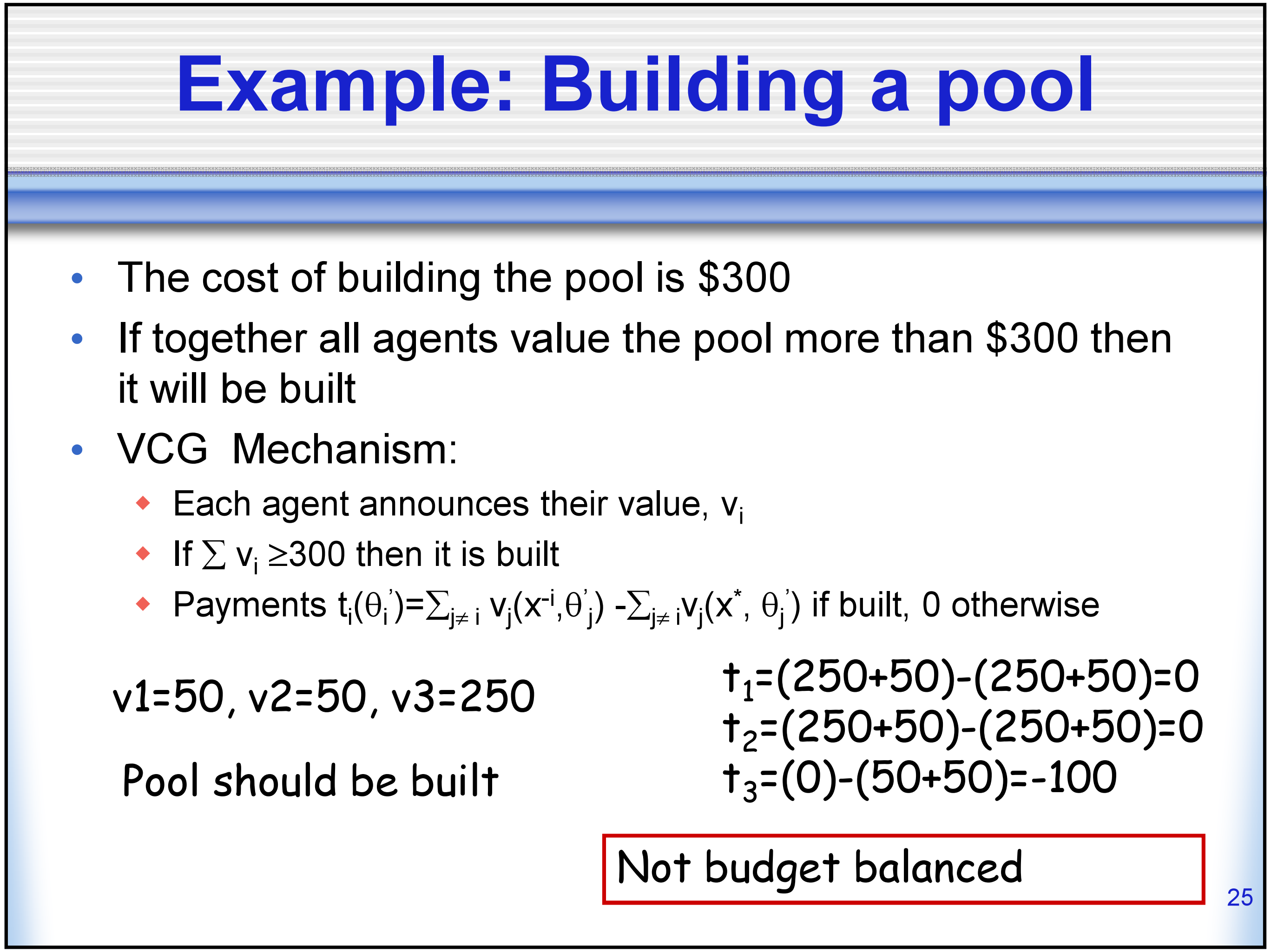
Agent's equilibrium utility is:

$$u_i(x^*,t_i,\theta_i)=v_i(x^*,\theta i)-[\Sigma_{j\neq i}\ v_j(x^{-i},\theta_j)\ -\Sigma_{j\neq i} v_j(x^*,\theta_j)]$$

$$= \Sigma_j\ v_j(x^*,\theta_j)\ -\ \Sigma_{j\ \neq i}\ v_j(x^{-i},\theta_j)$$

= marginal contribution to the welfare of the system

24



# Example: Building a pool

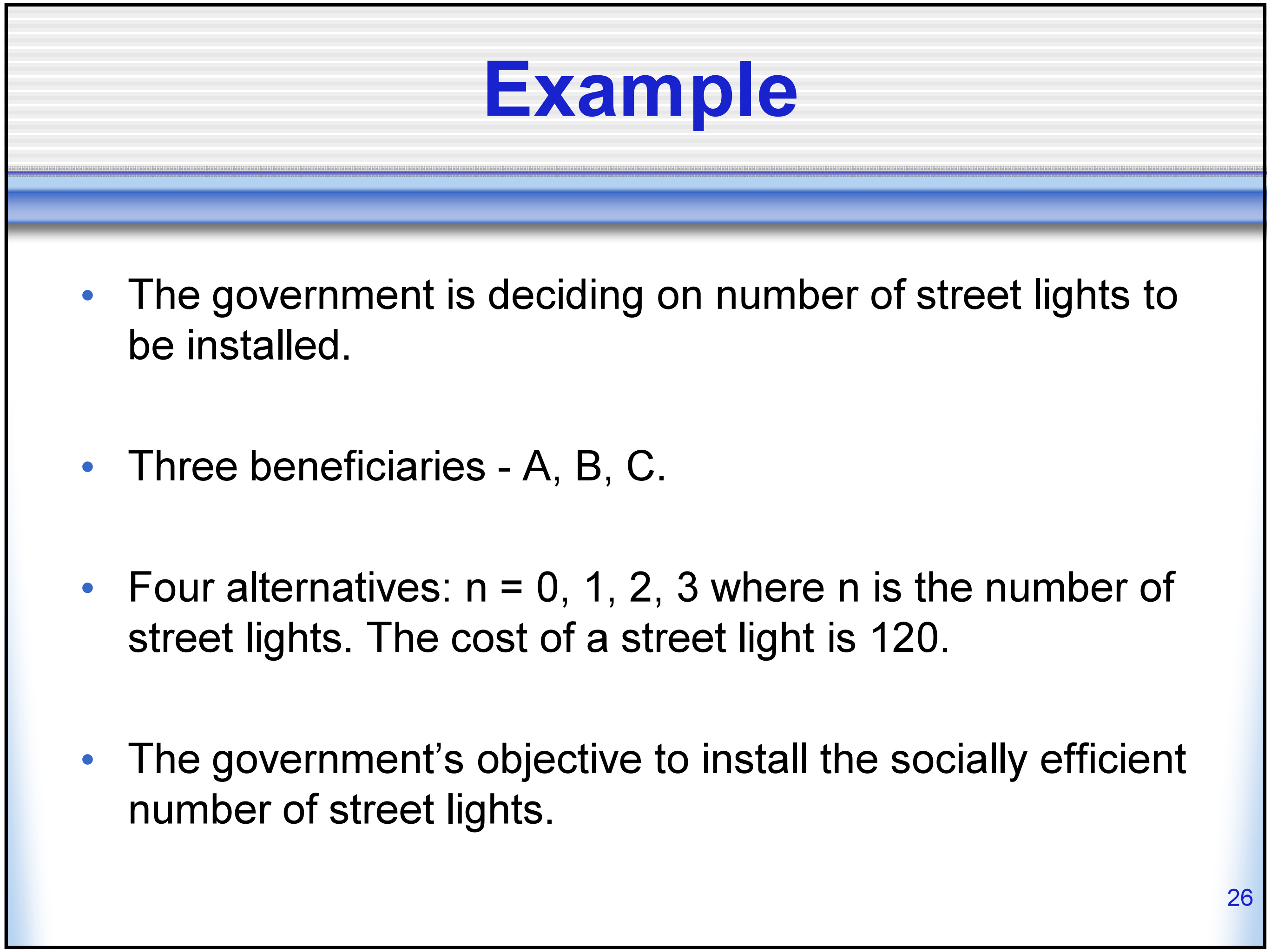
- The cost of building the pool is \$300
- If together all agents value the pool more than \$300 then it will be built
- VCG Mechanism:
  - Each agent announces their value, $v_i$
  - If $\sum v_i \geq 300$ then it is built
  - Payments $t_i(\theta_i') = \sum_{j\neq i} v_j(x^{-i},\theta_j') - \sum_{j\neq i} v_j(x^*, \theta_j')$ if built, 0 otherwise

v1=50, v2=50, v3=250

Pool should be built

$t_1=(250+50)-(250+50)=0$
$t_2=(250+50)-(250+50)=0$
$t_3=(0)-(50+50)=-100$

Not budget balanced

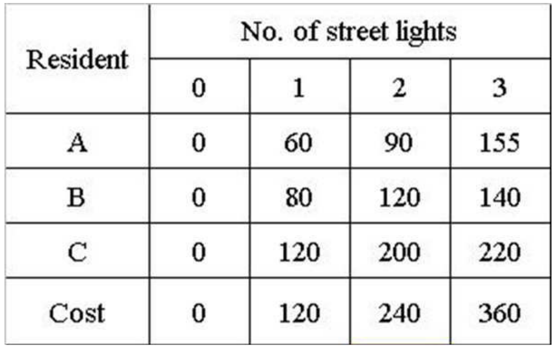


# Example

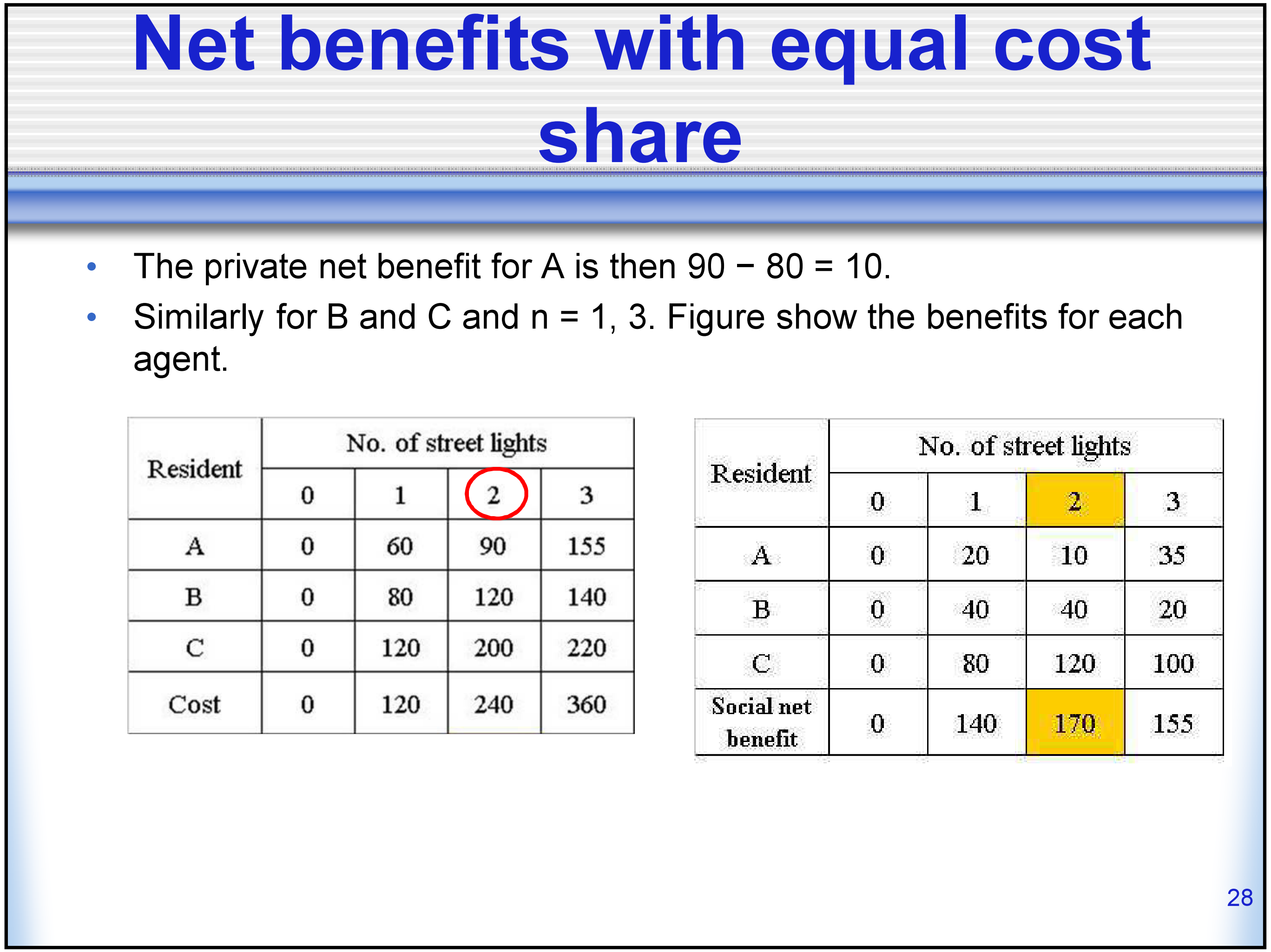
- The government is deciding on number of street lights to be installed.
- Three beneficiaries - A, B, C.
- Four alternatives: n = 0, 1, 2, 3 where n is the number of street lights. The cost of a street light is 120.
- The government’s objective to install the socially efficient number of street lights.

# Net benefits with equal cost share

- If n = 2, the total cost is 240.
  Hence, cost share for each is 80 (40 for each lamp).



| Resident | No. of street lights | | | |
|---|---|---|---|---|
| | 0 | 1 | 2 | 3 |
| A | 0 | 60 | 90 | 155 |
| B | 0 | 80 | 120 | 140 |
| C | 0 | 120 | 200 | 220 |
| Cost | 0 | 120 | 240 | 360 |



# Net benefits with equal cost share

- The private net benefit for A is then 90 − 80 = 10.
- Similarly for B and C and n = 1, 3. Figure show the benefits for each agent.

| Resident | No. of street lights | | | |
|---|---|---|---|---|
| | 0 | 1 | 2 | 3 |
| A | 0 | 60 | 90 | 155 |
| B | 0 | 80 | 120 | 140 |
| C | 0 | 120 | 200 | 220 |
| Cost | 0 | 120 | 240 | 360 |

| Resident | No. of street lights | | | |
|---|---|---|---|---|
| | 0 | 1 | 2 | 3 |
| A | 0 | 20 | 10 | 35 |
| B | 0 | 40 | 40 | 20 |
| C | 0 | 80 | 120 | 100 |
| Social net benefit | 0 | 140 | 170 | 155 |

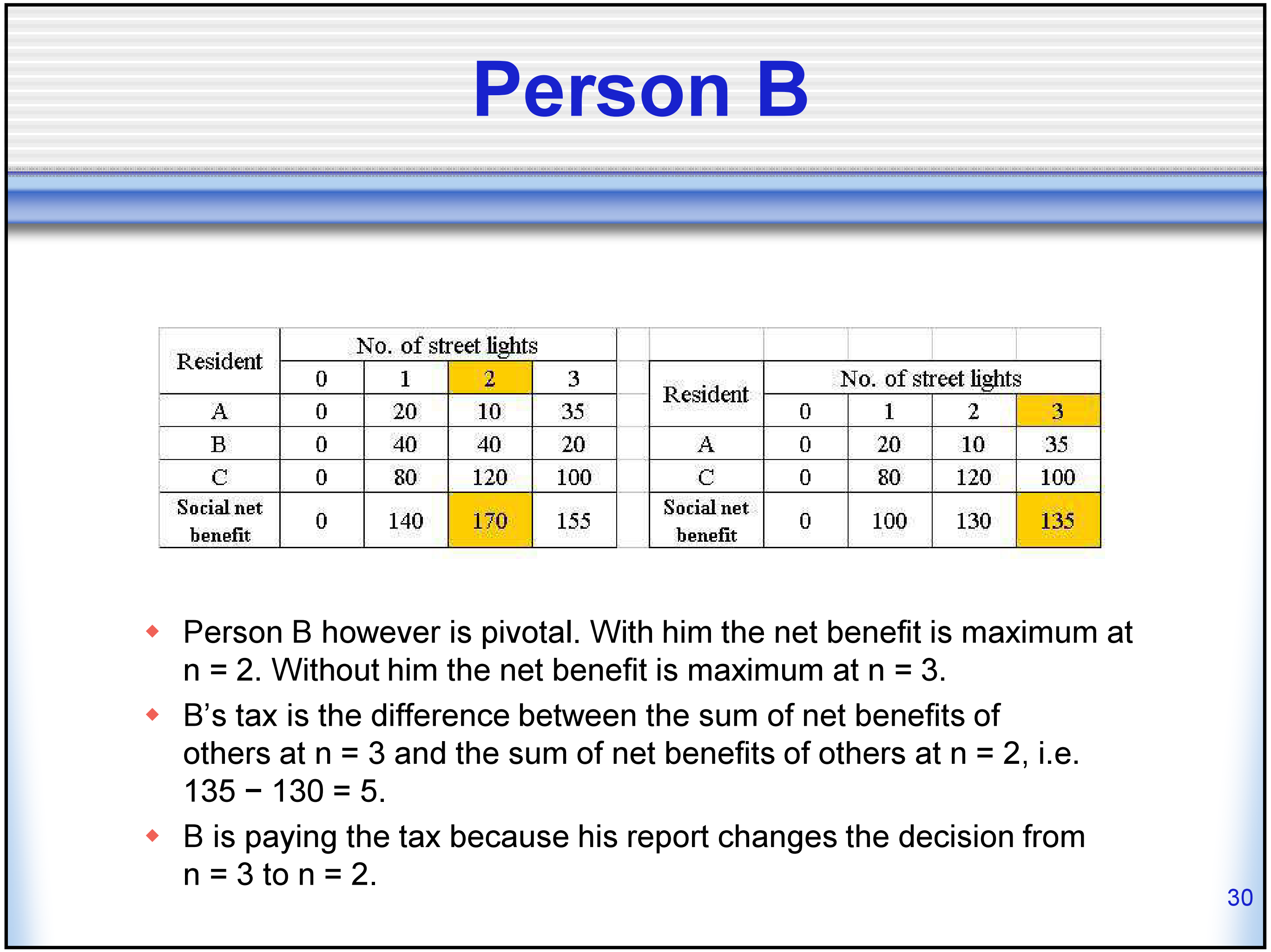
# Groves Clarke Taxes

- Is Person A pivotal? Does he has to pay a tax?

| Resident | No. of street lights | | | |
|---|---|---|---|---|
| | 0 | 1 | 2 | 3 |
| A | 0 | 20 | 10 | 35 |
| B | 0 | 40 | 40 | 20 |
| C | 0 | 80 | 120 | 100 |
| Social net benefit | 0 | 140 | 170 | 155 |

| Resident | No. of street lights | | | |
|---|---|---|---|---|
| | 0 | 1 | 2 | 3 |
| B | 0 | 40 | 40 | 20 |
| C | 0 | 80 | 120 | 100 |
| Social net benefit | 0 | 120 | 160 | 120 |

Person A is not pivotal. Without him, the net benefit is maximum at n = 2. With him the net benefit is maximum at n = 2. So his tax is zero.

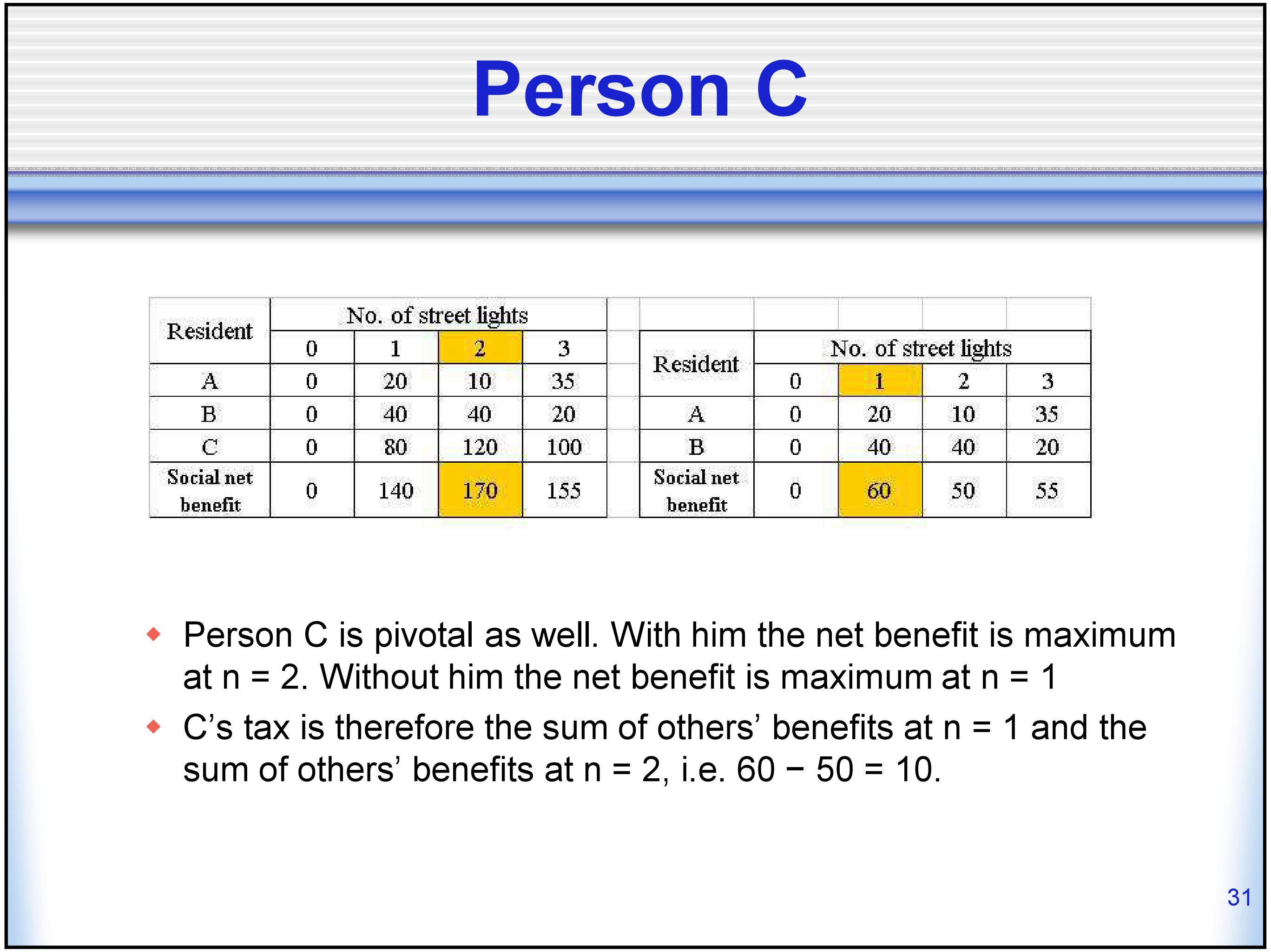


# Person B

| Resident | No. of street lights | | | |
|---|---|---|---|---|
| | 0 | 1 | 2 | 3 |
| A | 0 | 20 | 10 | 35 |
| B | 0 | 40 | 40 | 20 |
| C | 0 | 80 | 120 | 100 |
| Social net benefit | 0 | 140 | 170 | 155 |

| Resident | No. of street lights | | | |
|---|---|---|---|---|
| | 0 | 1 | 2 | 3 |
| A | 0 | 20 | 10 | 35 |
| C | 0 | 80 | 120 | 100 |
| Social net benefit | 0 | 100 | 130 | 135 |

- Person B however is pivotal. With him the net benefit is maximum at n = 2. Without him the net benefit is maximum at n = 3.
- B’s tax is the difference between the sum of net benefits of others at n = 3 and the sum of net benefits of others at n = 2, i.e. 135 − 130 = 5.
- B is paying the tax because his report changes the decision from n = 3 to n = 2.



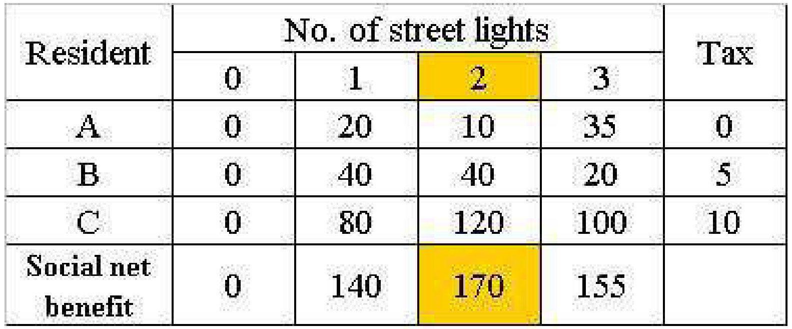
# Person C

| Resident | No. of street lights | | | |
|---|---|---|---|---|
| | 0 | 1 | 2 | 3 |
| A | 0 | 20 | 10 | 35 |
| B | 0 | 40 | 40 | 20 |
| C | 0 | 80 | 120 | 100 |
| Social net benefit | 0 | 140 | 170 | 155 |

| | | | | |
|---|---|---|---|---|
| Resident | No. of street lights | | | |
| | 0 | 1 | 2 | 3 |
| A | 0 | 20 | 10 | 35 |
| B | 0 | 40 | 40 | 20 |
| Social net benefit | 0 | 60 | 50 | 55 |

- Person C is pivotal as well. With him the net benefit is maximum at n = 2. Without him the net benefit is maximum at n = 1
- C’s tax is therefore the sum of others’ benefits at n = 1 and the sum of others’ benefits at n = 2, i.e. 60 − 50 = 10.

# Net benefits with taxes



| Resident | No. of street lights | | | | Tax |
|---|---|---|---|---|---|
| | 0 | 1 | 2 | 3 | |
| A | 0 | 20 | 10 | 35 | 0 |
| B | 0 | 40 | 40 | 20 | 5 |
| C | 0 | 80 | 120 | 100 | 10 |
| Social net benefit | 0 | 140 | 170 | 155 | |

- Post tax net benefit from this scheme:
  10 for A,
  40 − 5 = 35 for B,
  120 − 10 = 110 for C.

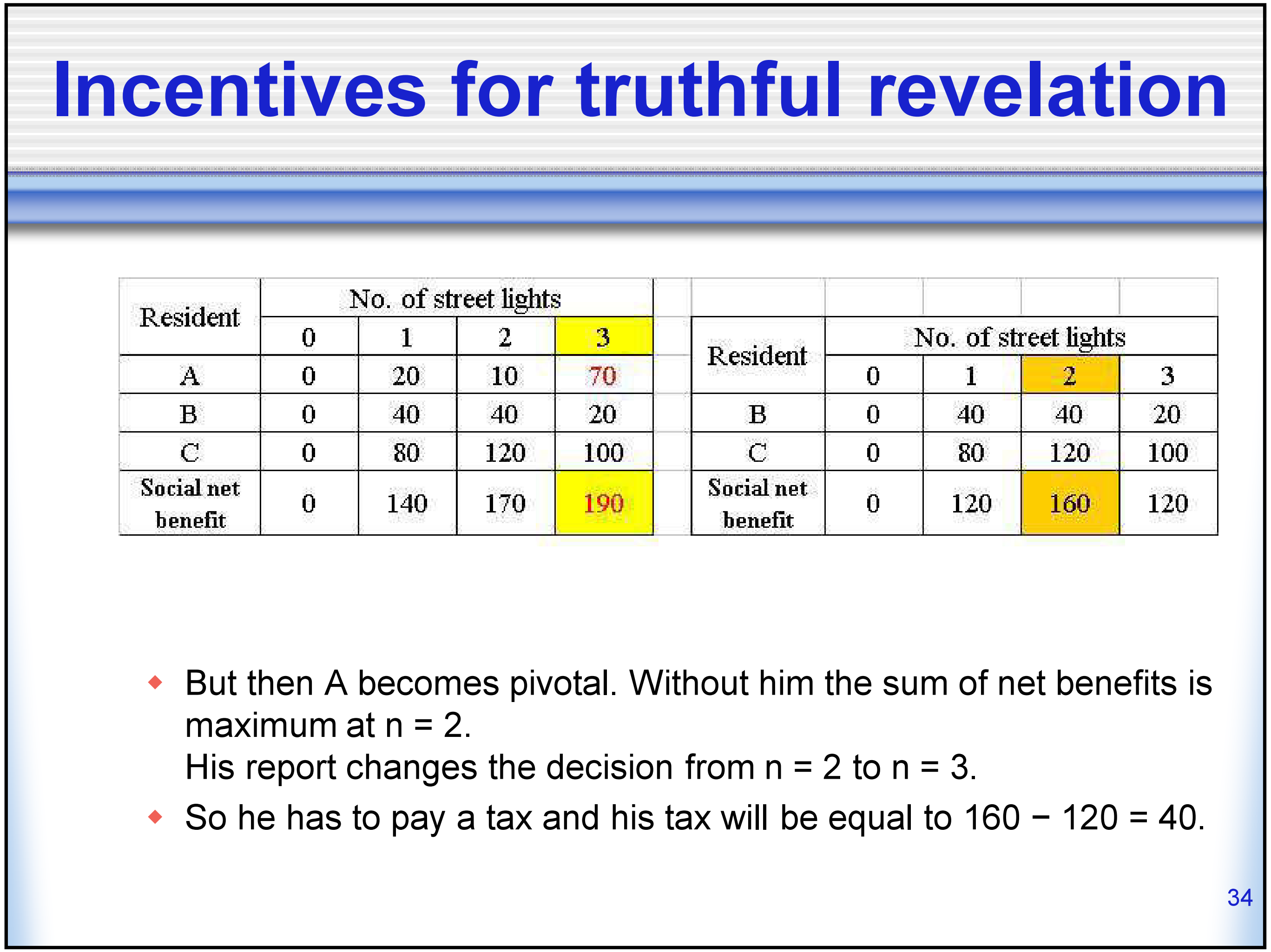
# Incentives for truthful revelation

| Resident | No. of street lights | | | |
|---|---|---|---|---|
| | 0 | 1 | 2 | 3 |
| A | 0 | 20 | 10 | 35 |
| B | 0 | 40 | 40 | 20 |
| C | 0 | 80 | 120 | 100 |
| Social net benefit | 0 | 140 | 170 | 190 |

?

70

- Notice that A’s net benefit is maximum at n = 3. Does he have an incentive to lie and change the decision to n = 3?
- Suppose A states his net benefit from n = 3 to be 70 instead of 35. Then, sum of stated net benefits is maximum at n = 3.

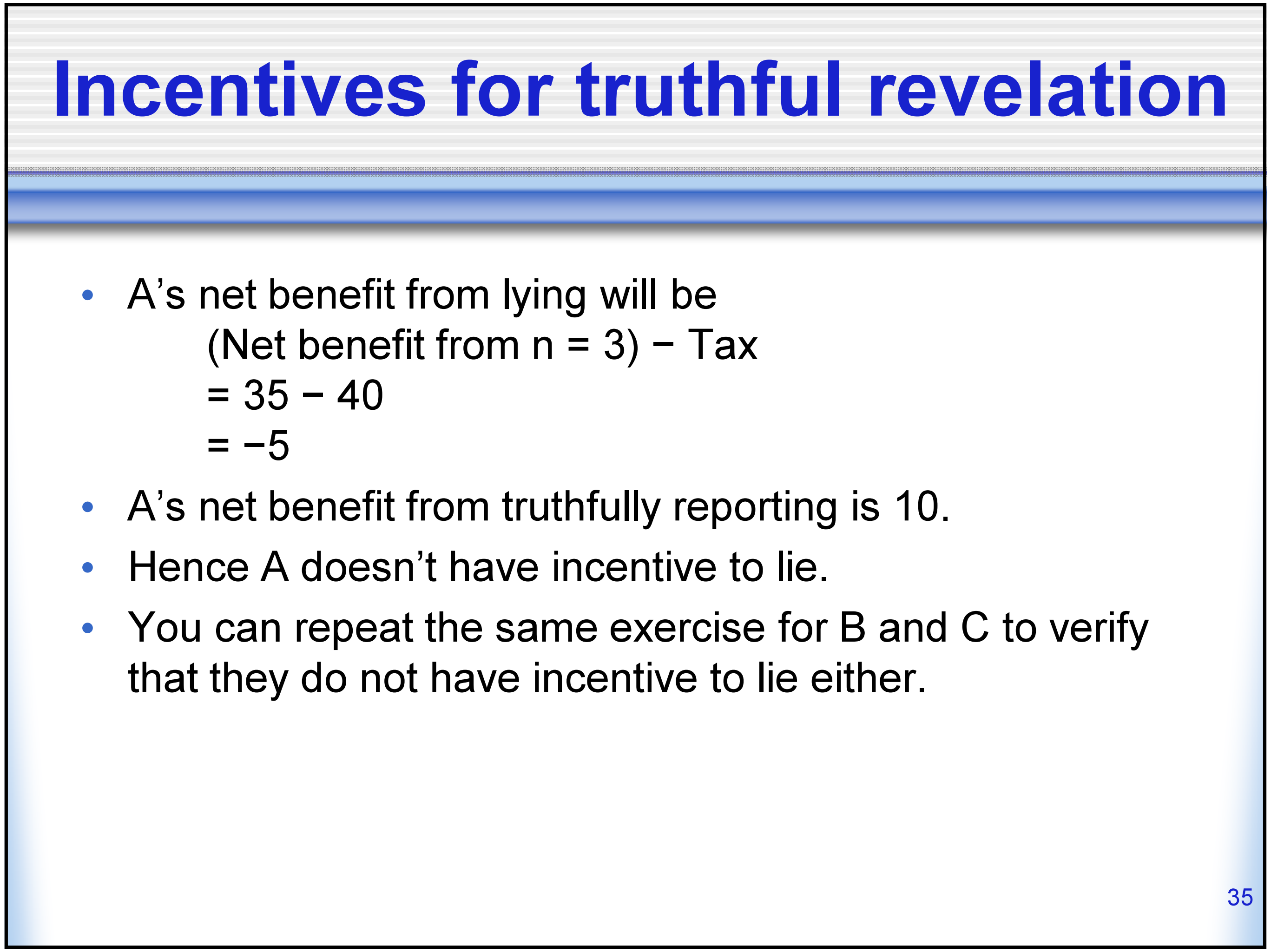


# Incentives for truthful revelation

| Resident | No. of street lights | | | |
|---|---|---|---|---|
| | 0 | 1 | 2 | 3 |
| A | 0 | 20 | 10 | 70 |
| B | 0 | 40 | 40 | 20 |
| C | 0 | 80 | 120 | 100 |
| Social net benefit | 0 | 140 | 170 | 190 |

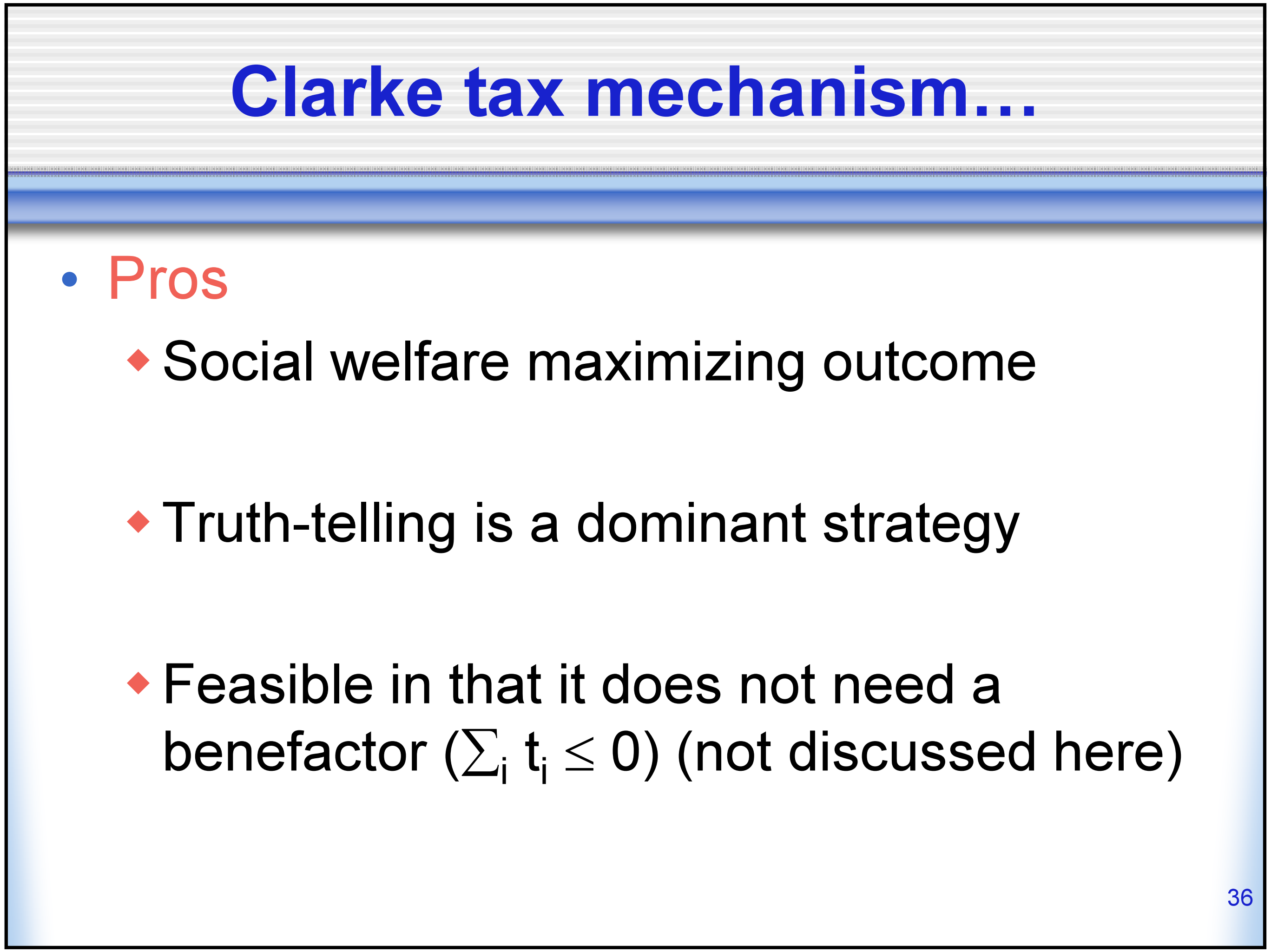
| Resident | No. of street lights | | | |
|---|---|---|---|---|
| | 0 | 1 | 2 | 3 |
| B | 0 | 40 | 40 | 20 |
| C | 0 | 80 | 120 | 100 |
| Social net benefit | 0 | 120 | 160 | 120 |

- But then A becomes pivotal. Without him the sum of net benefits is maximum at n = 2.
  His report changes the decision from n = 2 to n = 3.
- So he has to pay a tax and his tax will be equal to 160 − 120 = 40.



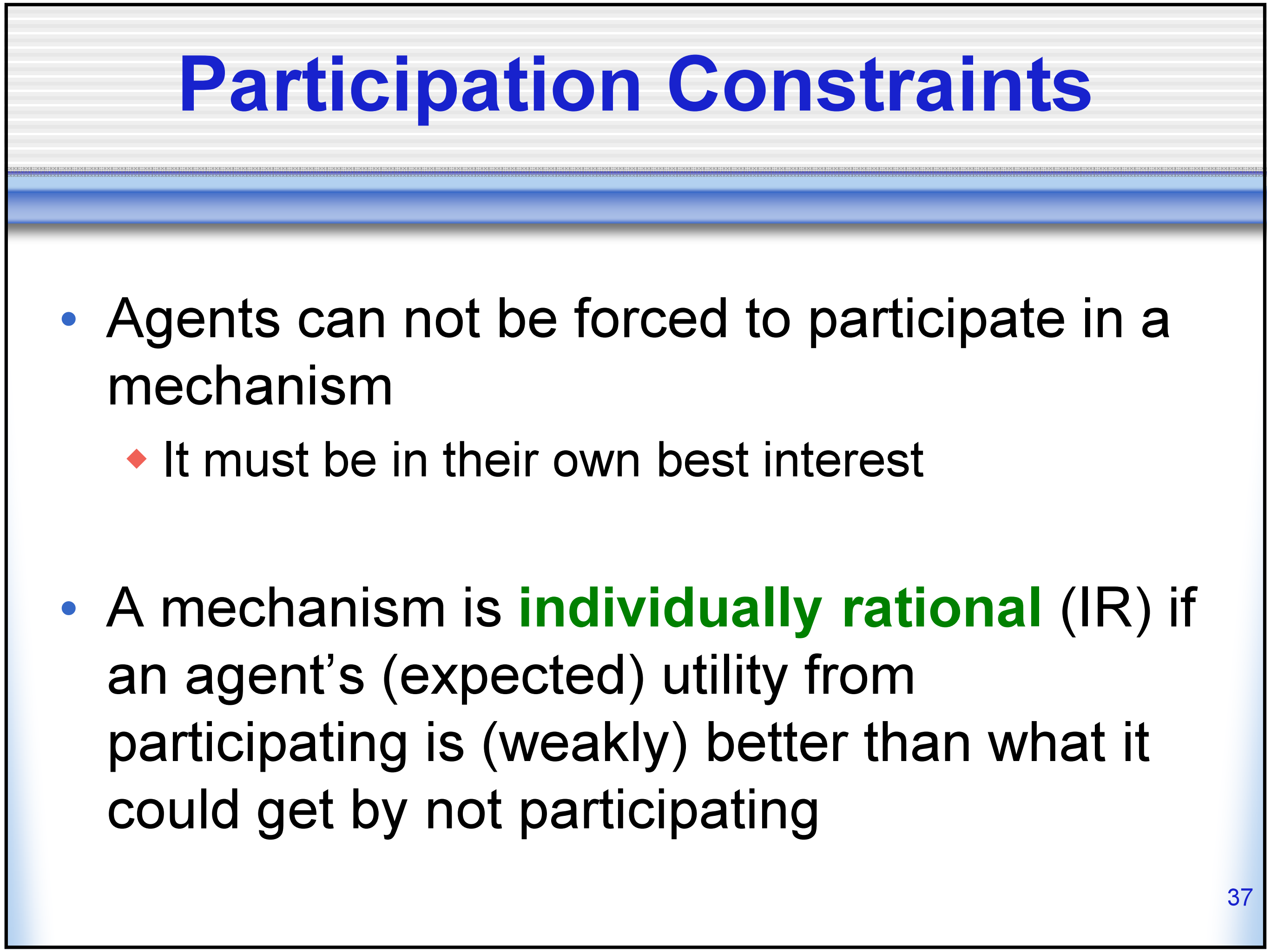
# Incentives for truthful revelation

- A’s net benefit from lying will be
  (Net benefit from n = 3) − Tax
  = 35 − 40
  = −5
- A’s net benefit from truthfully reporting is 10.
- Hence A doesn’t have incentive to lie.
- You can repeat the same exercise for B and C to verify that they do not have incentive to lie either.



# Clarke tax mechanism…

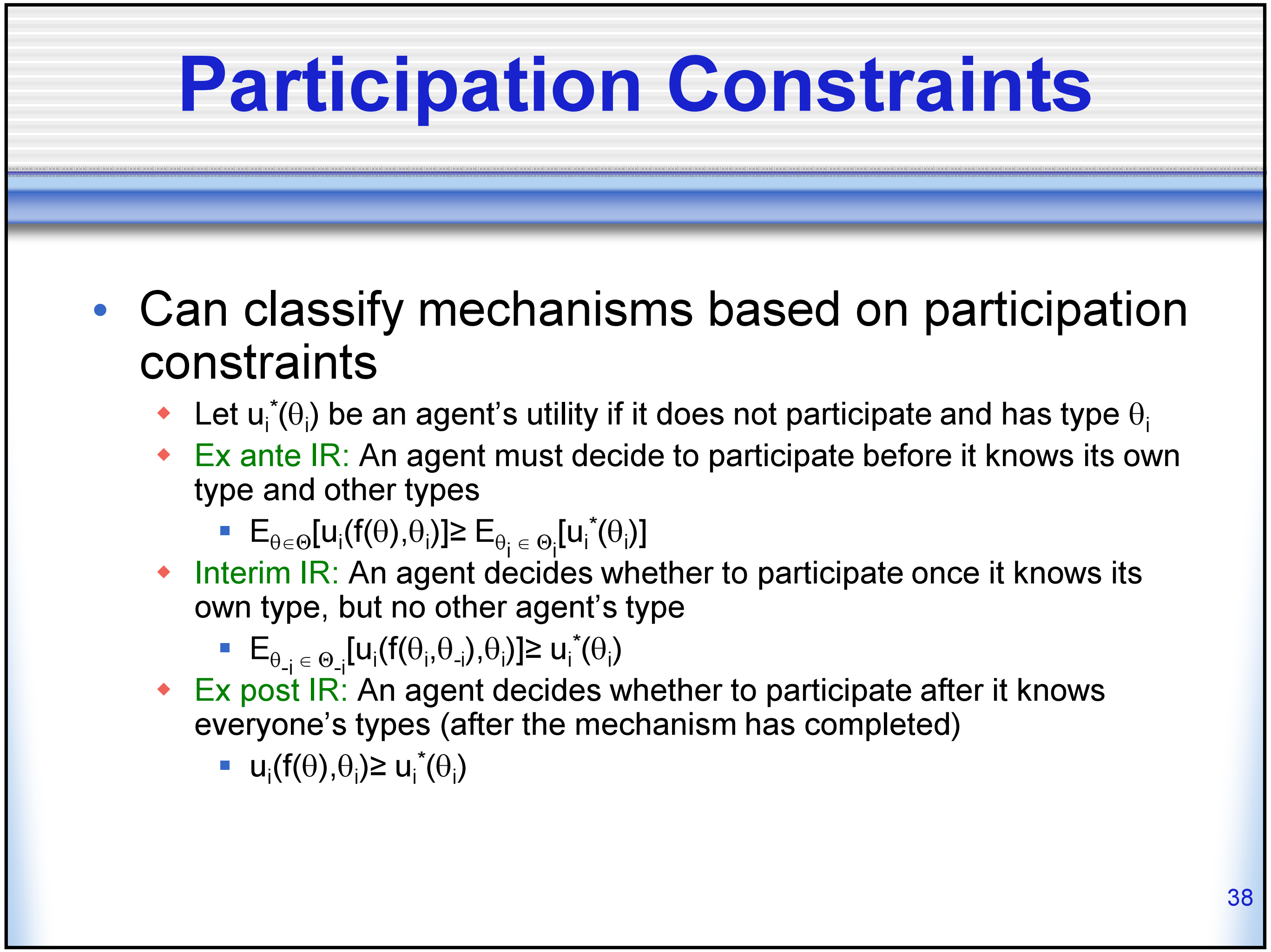
- Pros
  - Social welfare maximizing outcome
  - Truth-telling is a dominant strategy
  - Feasible in that it does not need a benefactor ($\sum_i t_i \leq 0$) (not discussed here)



# Participation Constraints

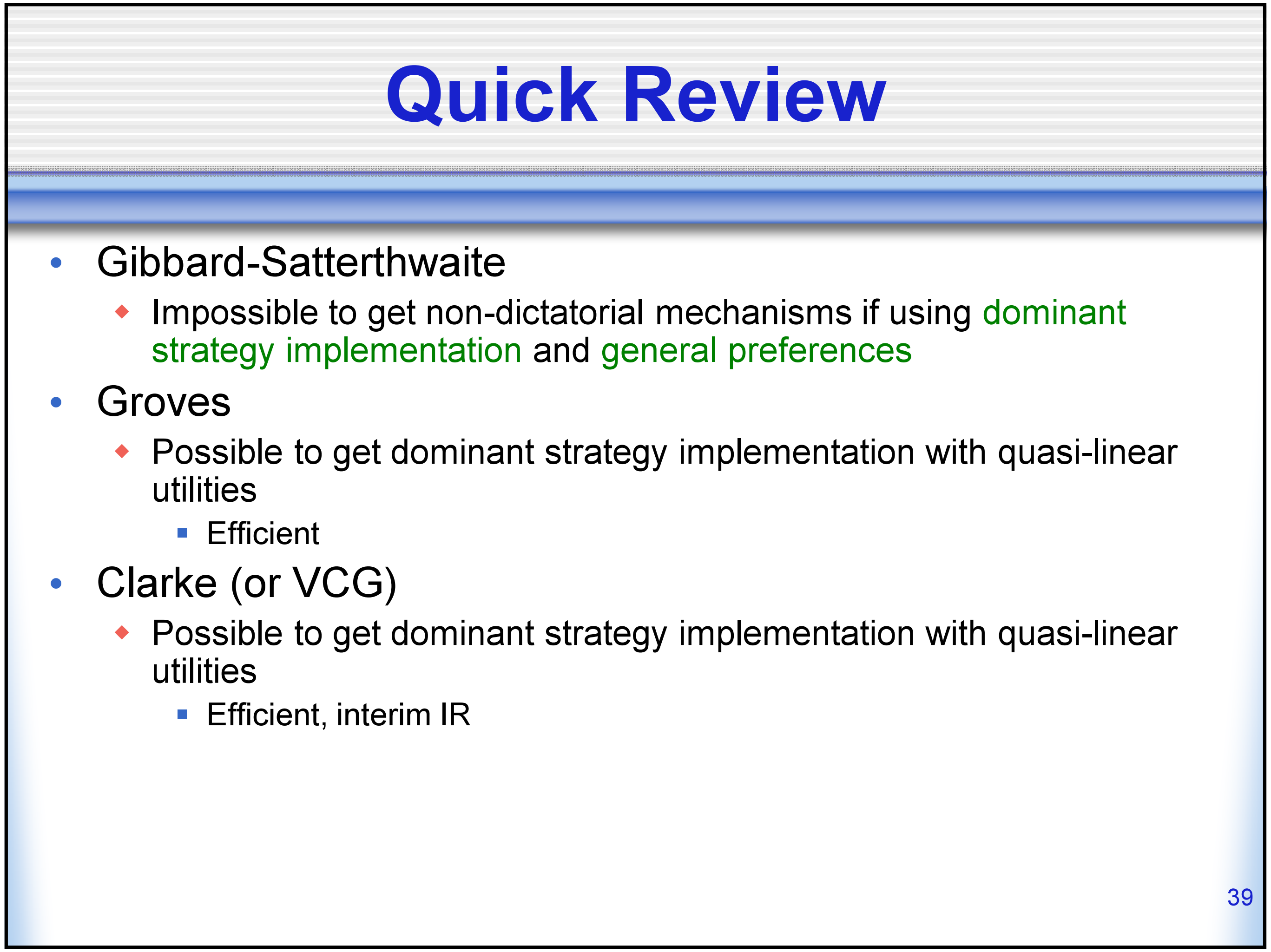
- Agents can not be forced to participate in a mechanism
  - It must be in their own best interest

- A mechanism is **individually rational** (IR) if an agent’s (expected) utility from participating is (weakly) better than what it could get by not participating



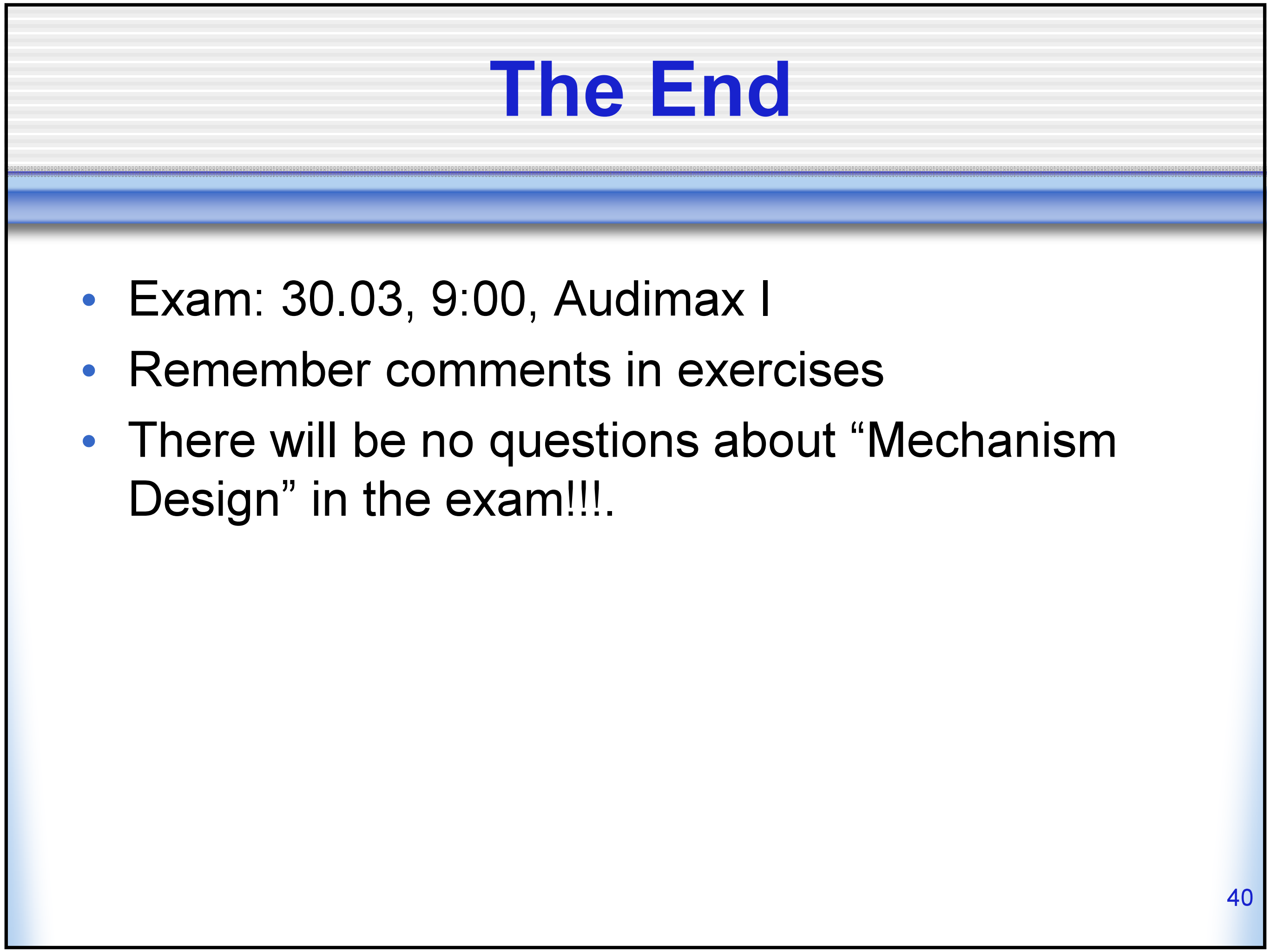
# Participation Constraints

- Can classify mechanisms based on participation constraints
  - Let $u_i^*(\theta_i)$ be an agent's utility if it does not participate and has type $\theta_i$
  - Ex ante IR: An agent must decide to participate before it knows its own type and other types
    - $E_{\theta\in\Theta}[u_i(f(\theta),\theta_i)]\geq E_{\theta_i \in \Theta_i}[u_i^*(\theta_i)]$
  - Interim IR: An agent decides whether to participate once it knows its own type, but no other agent's type
    - $E_{\theta_{-i} \in \Theta_{-i}}[u_i(f(\theta_i,\theta_{-i}),\theta_i)]\geq u_i^*(\theta_i)$
  - Ex post IR: An agent decides whether to participate after it knows everyone's types (after the mechanism has completed)
    - $u_i(f(\theta),\theta_i)\geq u_i^*(\theta_i)$



# Quick Review

- Gibbard-Satterthwaite
  - Impossible to get non-dictatorial mechanisms if using dominant strategy implementation and general preferences
- Groves
  - Possible to get dominant strategy implementation with quasi-linear utilities
    - Efficient
- Clarke (or VCG)
  - Possible to get dominant strategy implementation with quasi-linear utilities
    - Efficient, interim IR



# The End

- Exam: 30.03, 9:00, Audimax I
- Remember comments in exercises
- There will be no questions about “Mechanism Design” in the exam!!!.