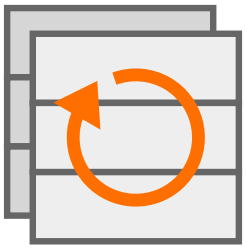


Operating
System
Group

TUHH
Technische Universität Hamburg

Betriebssystembau

Einleitung

Christian Dietrich

Sommersemester 2022

Der Dozent

- Wer ist der Dozent da? (26. Semester)
  - Informatikstudium in Erlangen (ab 2009)
  - Promotion in Hannover (2019)
  - Juniorprofessor an der TUHH (ab 2021)
  - Programmiersprachen, Emacs User, Kann an keinem Problem vorbeigehen
- Ich mag Betriebssysteme!
  - Vermittler zwischen Anwendung und Hardware
  - Synchronisierunsprobleme sind spannende Puzzle
  - Abstrakte Gedankengebäude treffen auf Hardwarefrickelei

Christian Dietrich

Der Dozent

- Wer ist der Dozent da? (26. Semester)
  - Informatikstudium in Erlangen (ab 2009)
  - Promotion in Hannover (2019)
  - Juniorprofessor an der TUHH (ab 2021)
  - Programmiersprachen, Emacs User, Kann an keinem Problem vorbeigehen

Christian Dietrich

- Ich mag Betriebssysteme!
  - Vermittler zwischen Anwendung und Hardware
  - Synchronisierunsprobleme sind spannende Puzzle
  - Abstrakte Gedankengebäude treffen auf Hardwarefrickelei

- Dies ist die zweite Runde von BSB.
  - Wir müssen das jetzt gemeinsam durchstehen.
  - Ich werde mein Bestes geben ihnen nur Sinnvolles zu vermitteln.
  - Melden Sie sich mit Fehlern, Kritik und Anregungen.

Interaktion während der Veranstaltung

TUHH
Technische Universität Hamburg

- BSB ist vom "Stil" her eine **interaktive Präsenzveranstaltung**
  - Wir wollen versuchen, dieses soweit wie möglich "online" zu retten

⇝ **Hybrides** Format – Fragen und Beteiligung ist erwünscht!

- Interaktion **während** der Veranstaltung
  - Vor Ort: Melden und drankommen.
  - Im BBB: Im Chat schreiben,
- Interaktion **außerhalb** der Veranstaltung
  - Stud.IP-Forum
  - Mattermost Kanal

Aufzeichnung der Vorlesung

You are on mic!

- **Veranstaltung wird aufgezeichnet**
  - Wird im Anschluss über Stud.IP verfügbar gemacht
  - ⇝ Geschlossene Nutzergruppe
- Aufgezeichnet wird
  - Folien + Kamera-Bild des Dozenten
  - Ihre Stimme bei Fragen und Anmerkungen

Vorwort

TUHH
Technische Universität Hamburg

Die Lehrveranstaltung ist grundsätzlich für alle Studiengänge offen. Sie verlangt allerdings gewisse Vorkenntnisse. Diese müssen nicht durch Teilnahme an den Lehrveranstaltungen erworben worden sein.

Lernziele

TUHH
Technische Universität Hamburg

- **Vertiefen** des Wissens über die interne Funktionsweise von Betriebssystemen
  - Ausgangspunkt: Betriebssysteme und/oder Rechnerarchitektur
  - Schwerpunkt: Nebenläufigkeit und Synchronisation
- **Entwickeln** eines Betriebssystems *von der Pike auf*
  - OOStuBS / MPStuBS Lehrbetriebssysteme
  - Praktische Erfahrungen im Betriebsystembau machen
- **Verstehen** der technologischen Hardware-Grundlagen
  - PC-Technologie verstehen und einschätzen können
  - Schwerpunkt: Intel x86 / IA-32

Empfohlene Vorkenntnisse

TUHH
Technische Universität Hamburg

- Rechnerarchitektur (Embedded Systems)
- **Betriebssysteme (Telematik)**
- C / **C++**, Assembler (x86)
- Ein gewisses Maß an **Durchhaltevermögen**
- Freude an systemnaher und **hardwarenaher Programmierung**

Wir arbeiten auf der "nackten Maschine" (*bare metal*)!

Empfohlene Vorkenntnisse

TUHH
Technische Universität Hamburg

- Rechnerarchitektur (Embedded Systems)
- **Betriebssysteme (Telematik)**
- C / **C++**, Assembler (x86)
- Ein gewisses Maß an **Durchhaltevermögen**
- Freude an systemnaher und **hardwarenaher Programmierung**

Wir arbeiten auf der "nackten Maschine" (*bare metal*)!

Die meisten sind überrascht, wie viel Spaß das macht :-)

Veranstaltungsbestandteile

(6 LP, 5 SWS)

Vorlesung **2V**

Vorstellung und detailierte Behandlung des Lehrstoffs

Veranstaltungsbestandteile

(6 LP, 5 SWS)

Vorlesung **2V**

Vorstellung und detailierte Behandlung des Lehrstoffs

+

Hörsaalübung **1HÜ**

- Vorstellung der Übungsaufgaben
- 6 Kerntermine ($\ddot{U}_1$–$\ddot{U}_6$), 2 Optionale Termine ($\ddot{U}_0$, $\ddot{U}_7$)
- Ungefähr alle 14 Tage

Veranstaltungsbestandteile

(6 LP, 5 SWS)

Vorlesung **2V**

Vorstellung und detailierte Behandlung des Lehrstoffs

+

Hörsaalübung **1HÜ**

- Vorstellung der Übungsaufgaben
- 6 Kerntermine ($\ddot{U}_1$–$\ddot{U}_6$), 2 Optionale Termine ($\ddot{U}_0$, $\ddot{U}_7$)
- Ungefähr alle 14 Tage

+

RÜ: *Betriebssystembau* **3 PBL**

- Übung **OOStuBS**
- Pflicht: 6 Aufgaben
- Lösungsdiskussion

oder

RÜ: *... für Mehrkernsysteme* **3 PBL**

- Übung **MPStuBS**
- erweiterte Aufgaben
- 20 % Notenbonus

Verwendbarkeit, Scheinerwerb und Modulnote

- Vertiefung: I. Computer- und Software-Engineering
  - Master Computer Science (CS)
- Modul: Technischer Ergänzungskurs (6LP)
  - Master Informatik-Ingenieurwesen (IIW)
- Prüfung und Modulnote
  - Mündliche Prüfung (25 Minuten)
  - Übung ist als Studienleistung **verpflichtend**!
  - Note ergibt sich aus der mündlichen Prüfung.

Übungsbetrieb

TUHH
Technische Universität Hamburg

Hörsaalübung (H 0.02)

- Mi, 12:15 – 13:45
- Übungsaufgaben sollen in 2er-Gruppen bearbeitet werden
- Abwechselnd mit langer Rechnerübung

Übungsbetrieb

TUHH
Technische Universität Hamburg

Hörsaalübung (H 0.02)

- Mi, 12:15 – 13:45
- Übungsaufgaben sollen in 2er-Gruppen bearbeitet werden
- Abwechselnd mit langer Rechnerübung

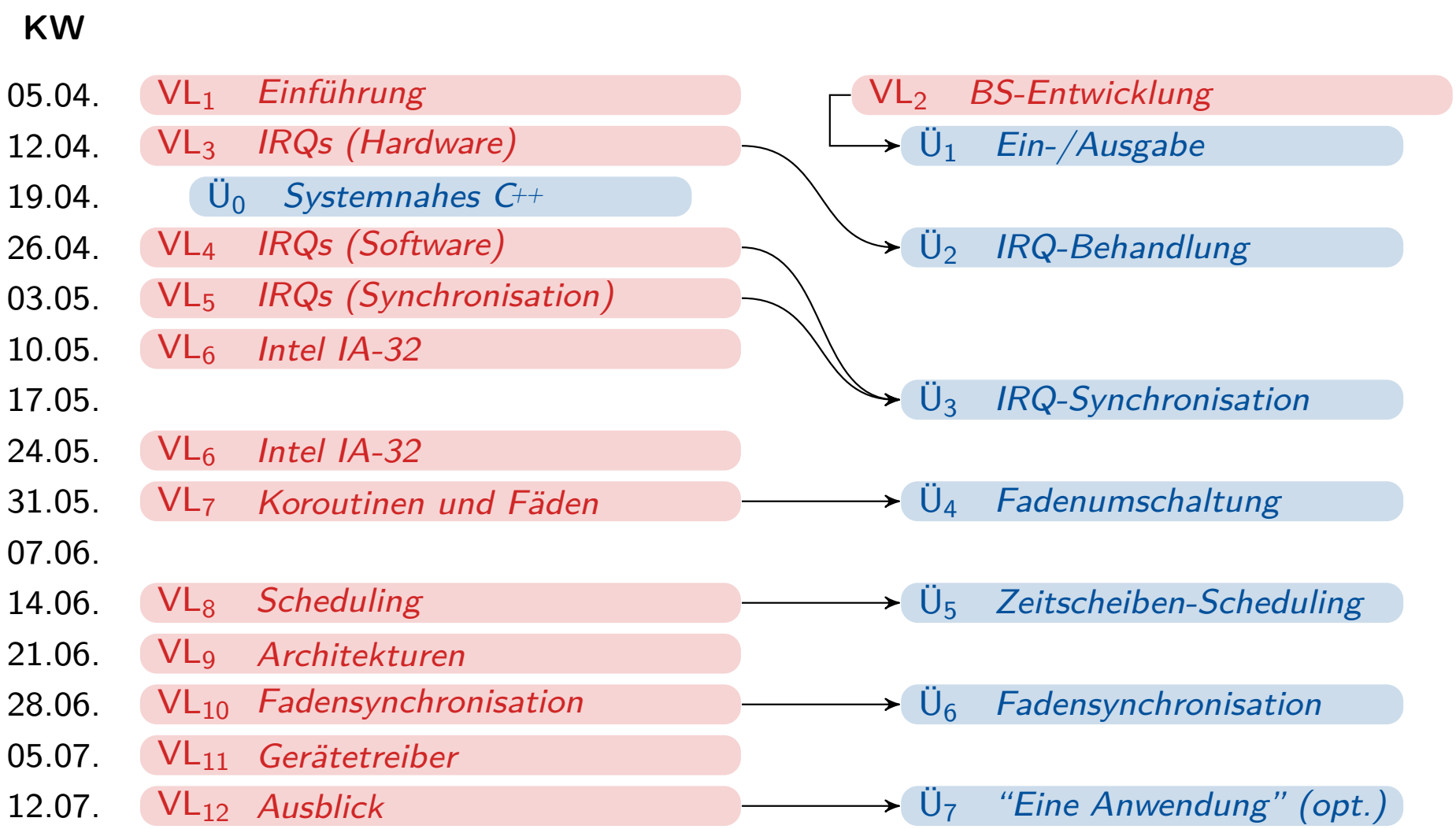
Betreute Rechnerzeit (CIP/E 2.024P3c)

- Woche ohne Hörsaalübung: Mi, 12:15 – 16:00
- Woche mit Hörsaalübung: Mi, 13:45 – 16:00
- Fragen auch im StudIP Forum

Kritische Begleitung durch ZLL

- Veranstaltung soll durch das ZLL begleitet werden
  - Frage: Welche Vorteile bietet das BSB Format?
  - Anderer Blickwinkel als Lehrveranstaltungsevaluation
- 2-3 Fragebögen im Laufe des Semesters
  - ZLL agiert als unbeteiligter Dritte
  - Ich bekomme nur anonymisierte Daten
- **Bitte mitmachen!**
  - Nötig für meine Evaluation als Juniorprofessor
  - Argument für mehr praktische Veranstaltung an der TUHH.

Verzahnung von Vorlesung und Übungsaufgaben



Ebenenmodell
und
Virtuelle Maschinen

[Problem]

Hochsprache
⇑
Semantische Lücke
⇓
CPU-Steuersignale

[Ausführung]

- Informatisches Urproblem:
  Schließen der Semantischen Lücke
- Komplexitätsreduktion durch Abstraktion
  - Hierarchisch angeordnete virtuelle Maschinen
  - Definierte Schnittstellen nach oben
  - Schrittweise Vereinfachungen nach unten
- BSB: Erweiterung der Hardware
  - Top-Down: Schnittstellen für die Anwendung
  - Bottom-Up: Geräte virtualisieren

Hierarchie virtueller Maschinen [1, S. 3]

- Schrittweises Schließen der semantischen Lücke

Ebene	Abstraktion	
n	virtuelle Maschine M_n	Sprache L_n
⋮	⋮	⋮
2	virtuelle Maschine M_2	Sprache L_2
1	virtuelle Maschine M_1	Sprache L_1
0	reale Maschine M_0	Sprache L_0

- L_x-Programm wird auf Maschine M_{x-1} abgebildet
  - **Dynamisch**: Interpreter als L_{x-1}-Programm
  - **Statisch**: Compiler erzeugt L_{x-1}-Programm
    Spätere Abbildung von L_{x-1} auf M_{x-2}

Dedoushka
(CC BY-SA 3.0)

"Virtuelle Maschinen"

TUHH
Technische Universität Hamburg

Wichtig: Wir meinen nicht VMWare, VirtualBox oder KVM.

- Maschinenmodell der virtuellen Maschine
  - Speicher/Objekte: Wie kann man Daten ablegen und wieder abrufen?
  - Befehle/Operationen: Wie kann man Daten miteinander kombinieren?
  - Ein "Prozessor" kann dieses Modell implementieren.
- Programme der virtuellen Maschine
  - Eine Menge von Befehlen, die gegen das Modell geschrieben wurden.
  - Valide Programme müssen syntaktischen und semantischen Regeln folgen.
- Prozessoren (Software *oder* Hardware) verarbeiten diese Programme
  - Übersetzter: Programme zwischen virtuellen Maschinen transformieren
  - Interpreter: Ausführung des Programms zur Berechnung des Ergebnisses

Beispiel: (virtuelle) RISC-V Maschine

TUHH
Technische Universität Hamburg

RISC-V ist eine Rechnerarchitektur, die an industrieller Relevanz gewinnt.

- Maschinenmodell der Befehlssatzebene E_2: RV32I
  - Speicher: 32 CPU Register + frei adressierbarer Speicher
  - Befehle: 47 Instruktionen mit arithmetischen, binären und Sprungbefehlen
  - Der SweRV von Western Digital ist ein Interpreter für dieses Modell.
- Beispiel eines Befehlssatzprogrammes: 02 b5 05 3b 80 82

Beispiel: (virtuelle) RISC-V Maschine

TUHH
Technische Universität Hamburg

RISC-V ist eine Rechnerarchitektur, die an industrieller Relevanz gewinnt.

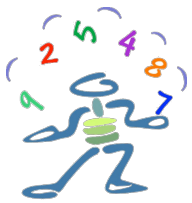
- Maschinenmodell der Befehlssatzebene E_2: RV32I
  - Speicher: 32 CPU Register + frei adressierbarer Speicher
  - Befehle: 47 Instruktionen mit arithmetischen, binären und Sprungbefehlen
  - Der SweRV von Western Digital ist ein Interpreter für dieses Modell.
- Beispiel eines Befehlssatzprogrammes: 02 b5 05 3b 80 82
- Zugehöriges Assemblerprogramm (E_4)

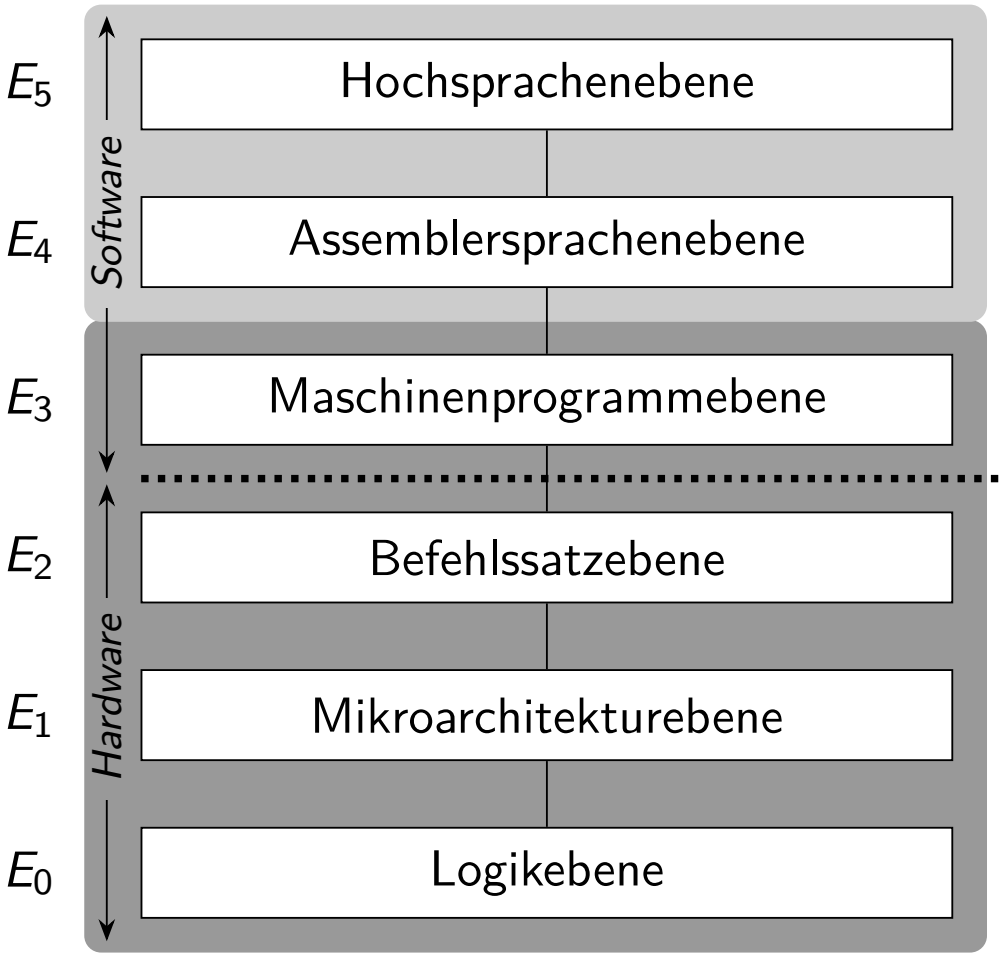
```
multiply: # int multiply(int a, int b) { return a*b; }
    mul a0, a0, a1
    ret
```

RV32I

  - Assembler war Übersetzer des Programms von $E_4 \rightarrow E_2$
  - SweRV oder QEMU können als Interpreter des Progamms dienen.

Beispiel: Arbeitsplatzrechner





Übersetzung (Compiler)

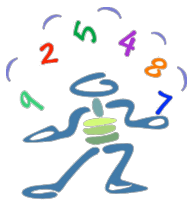
Übersetzung (Assembler)
Bindung (Linker)

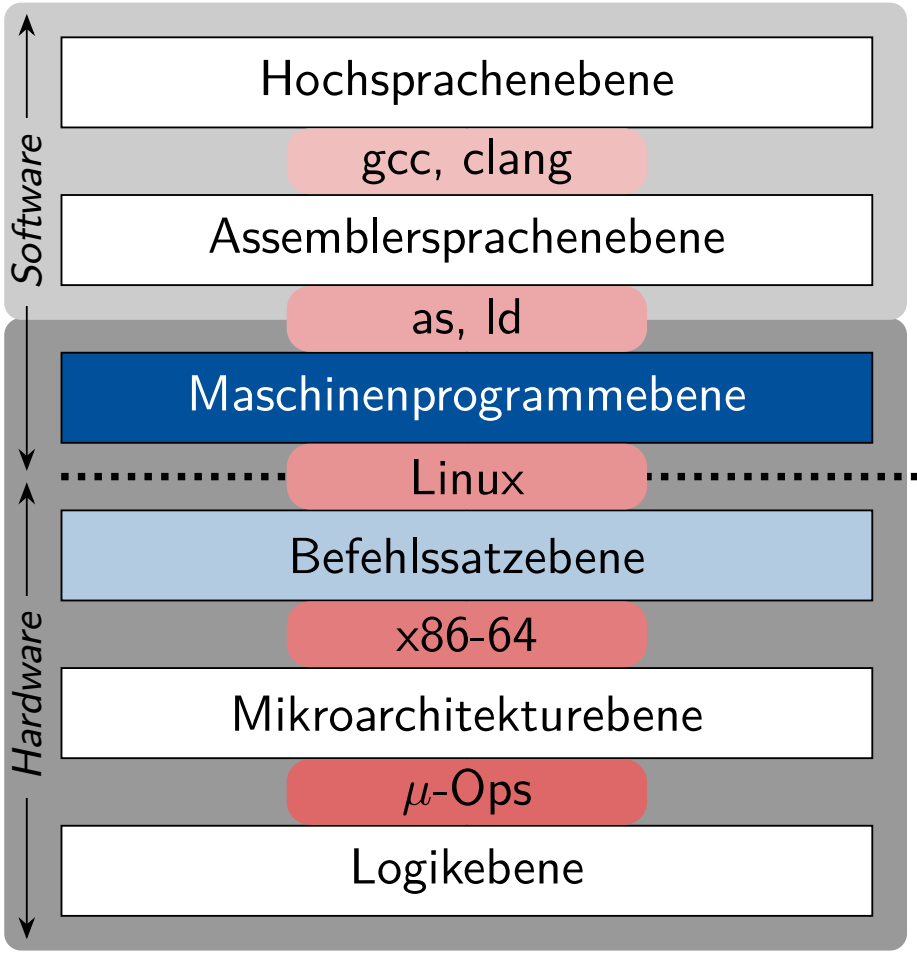
Teilinterpretation (Betriebssystem)

Interpretation (Mikroprogramm)

Ausführung

Beispiel: Arbeitsplatzrechner





Betriebssystem: Erweiterung der realen Maschine

- **Problem:** Die E_2-CPU ist sehr rudimentär in ihrem Befehlsumfang
  - Addieren, Laden, Speicher, Steuersignale nach Außen setzen
  - Keinen Befehl für "Öffne eine Datei", "Schreibe auf Bildschirm"

↪ Die EntwicklerInnen schreien nach besseren Abstraktionen!

Betriebssystem: Erweiterung der realen Maschine

- **Problem:** Die E_2-CPU ist sehr rudimentär in ihrem Befehlsumfang
  - Addieren, Laden, Speicher, Steuersignale nach Außen setzen
  - Keinen Befehl für "Öffne eine Datei", "Schreibe auf Bildschirm"

  ↪ Die EntwicklerInnen schreien nach besseren Abstraktionen!

  - Die E_3-Maschine: Erweiterung der E_2-Maschine um solche Befehle

E_3	mul	writeint	ret
	02 b5 05 3b	0d 70 08 93 00 00 00 73	80 82

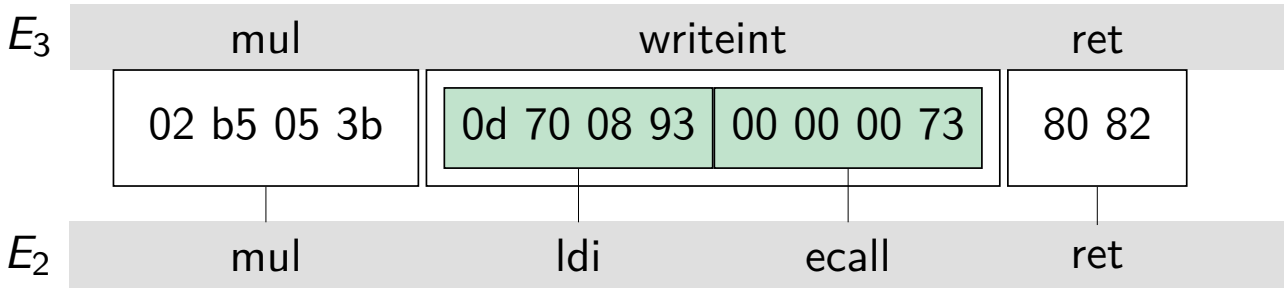
  - Umsetzung: Übersetzen, Hardware erweitern, **Teilinterpretation**

Betriebssystem: Erweiterung der realen Maschine

- **Problem:** Die E_2-CPU ist sehr rudimentär in ihrem Befehlsumfang
  - Addieren, Laden, Speicher, Steuersignale nach Außen setzen
  - Keinen Befehl für "Öffne eine Datei", "Schreibe auf Bildschirm"

  ↪ Die EntwicklerInnen schreien nach besseren Abstraktionen!

  - Die E_3-Maschine: Erweiterung der E_2-Maschine um solche Befehle



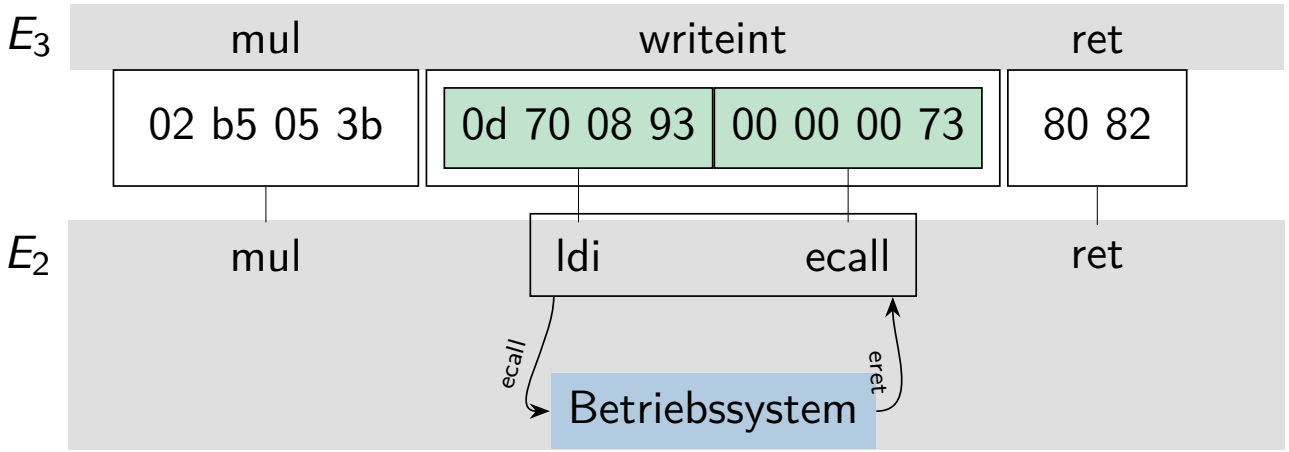
- Umsetzung: Übersetzen, Hardware erweitern, **Teilinterpretation**
- E_3-Befehle als E_2-Befehle codiert, die das Betriebssystem aktivieren.

Betriebssystem: Erweiterung der realen Maschine

- **Problem:** Die E_2-CPU ist sehr rudimentär in ihrem Befehlsumfang
  - Addieren, Laden, Speicher, Steuersignale nach Außen setzen
  - Keinen Befehl für "Öffne eine Datei", "Schreibe auf Bildschirm"

  ↪ Die EntwicklerInnen schreien nach besseren Abstraktionen!

  - Die E_3-Maschine: Erweiterung der E_2-Maschine um solche Befehle

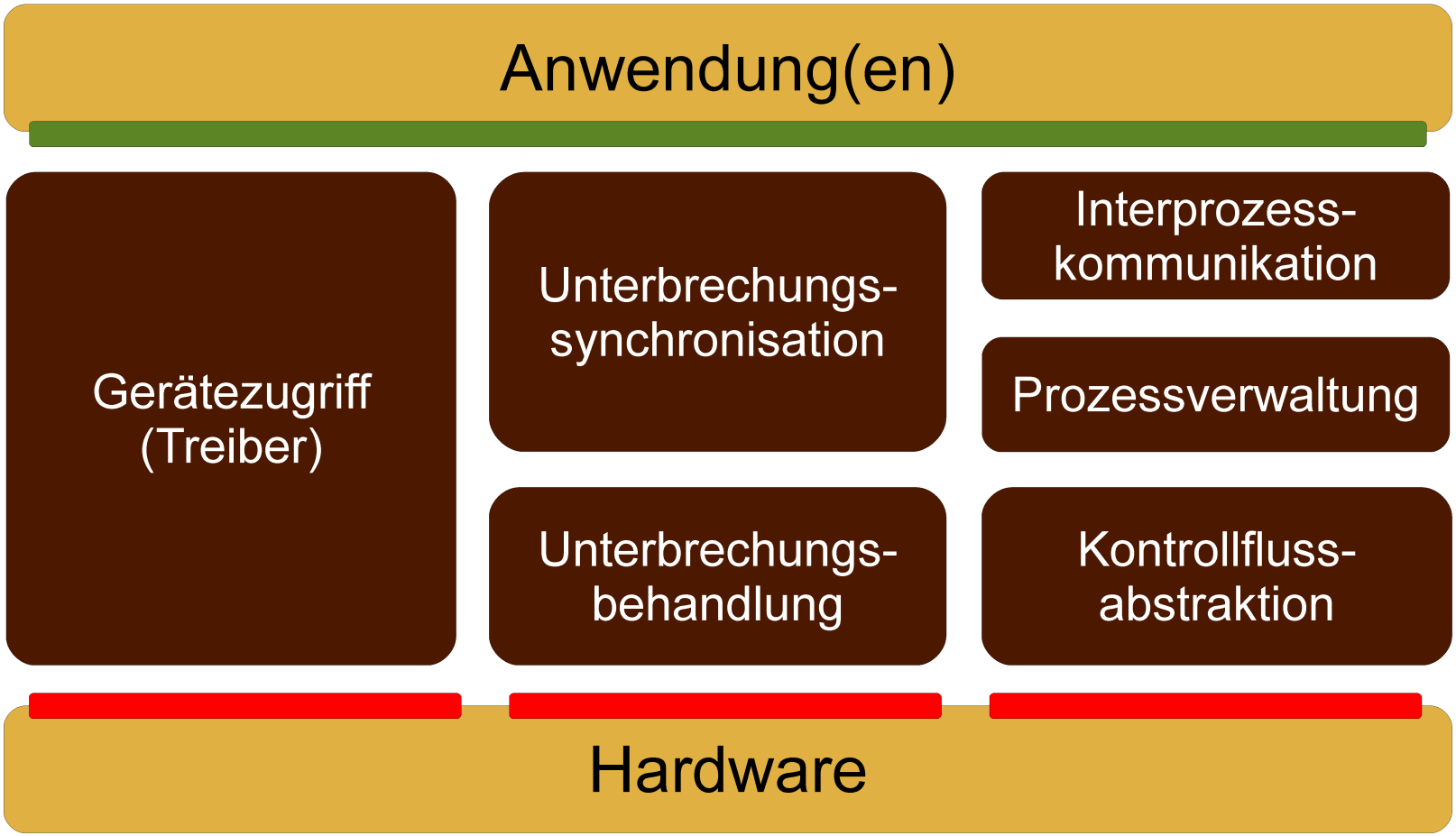


  - Umsetzung: Übersetzen, Hardware erweitern, **Teilinterpretation**
  - E_3-Befehle als E_2-Befehle codiert, die das Betriebssystem aktivieren.
  - Betriebssystem ist ein E_2-Programm und führt Teilinterpreation durch.

BS $\mapsto$ Multiplexer und Virtualisierer

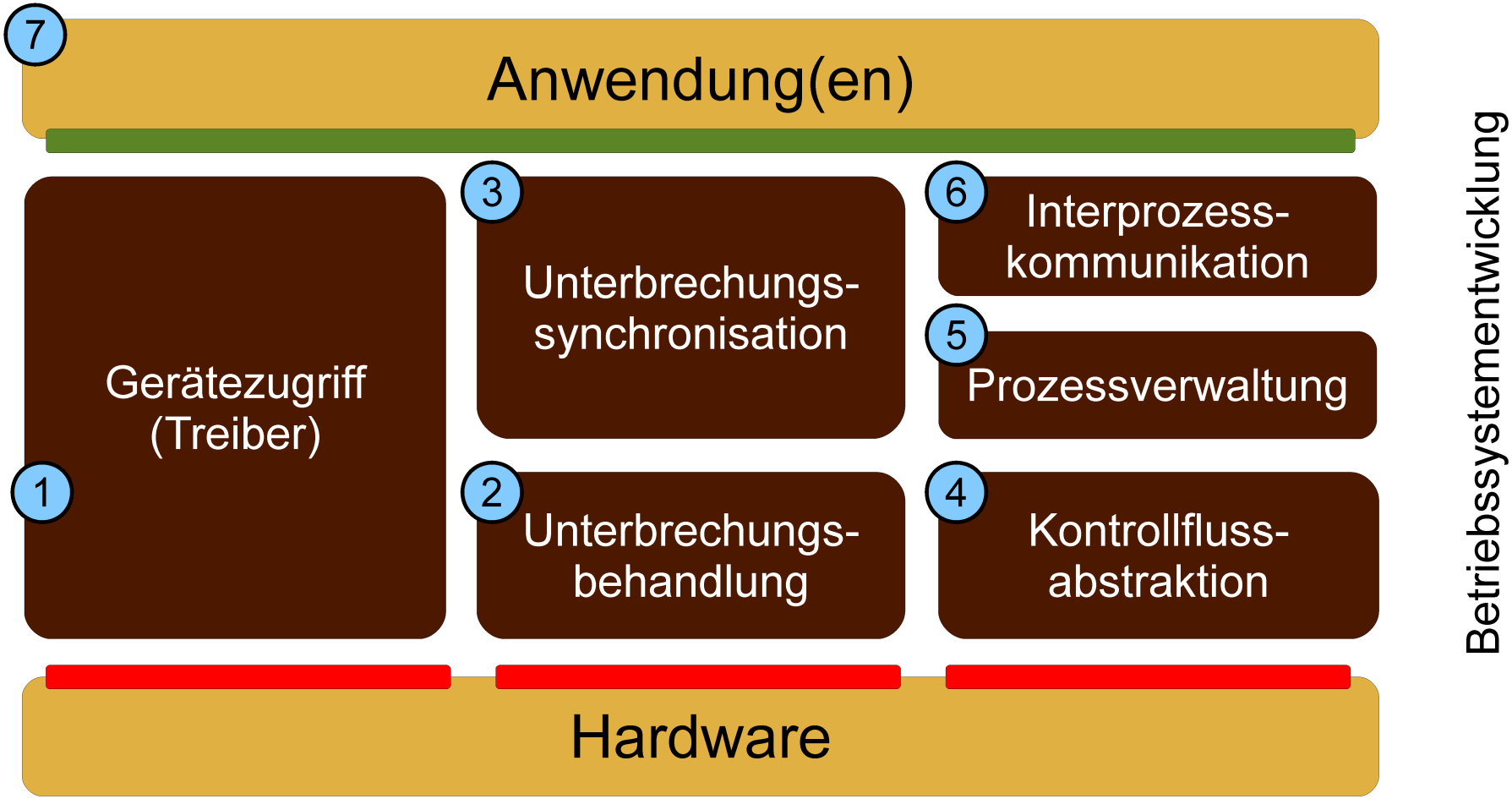
- Das BS bietet dem Anwender (Programm) eine erweiterte Maschinenschnittstelle (teilinterpretierende virtuelle Maschine, E_3).
- Der Befehlssatz der HW (E_2) wird (weitestgehend) "durchgereicht" und um zusätzliche Systemaufrufe (system calls) erweitert.
- Die Ressourcen der HW (Prozessor, Speicher, IO-Geräte, ...) werden durch Multiplexing virtualisiert (Mehrprogrammbetrieb).
- Virtuelle Hardwareressourcen werden durch Schutzmechanismen (räumlich und zeitlich) voneinander isoliert (Mehrbenutzerbetrieb).
- Virtuelle Hardwareressourcen werden repräsentiert durch die grundlegenden Konzepte (Abstraktionen) eines Betriebssystems.

Was ist ein Betriebssystem?



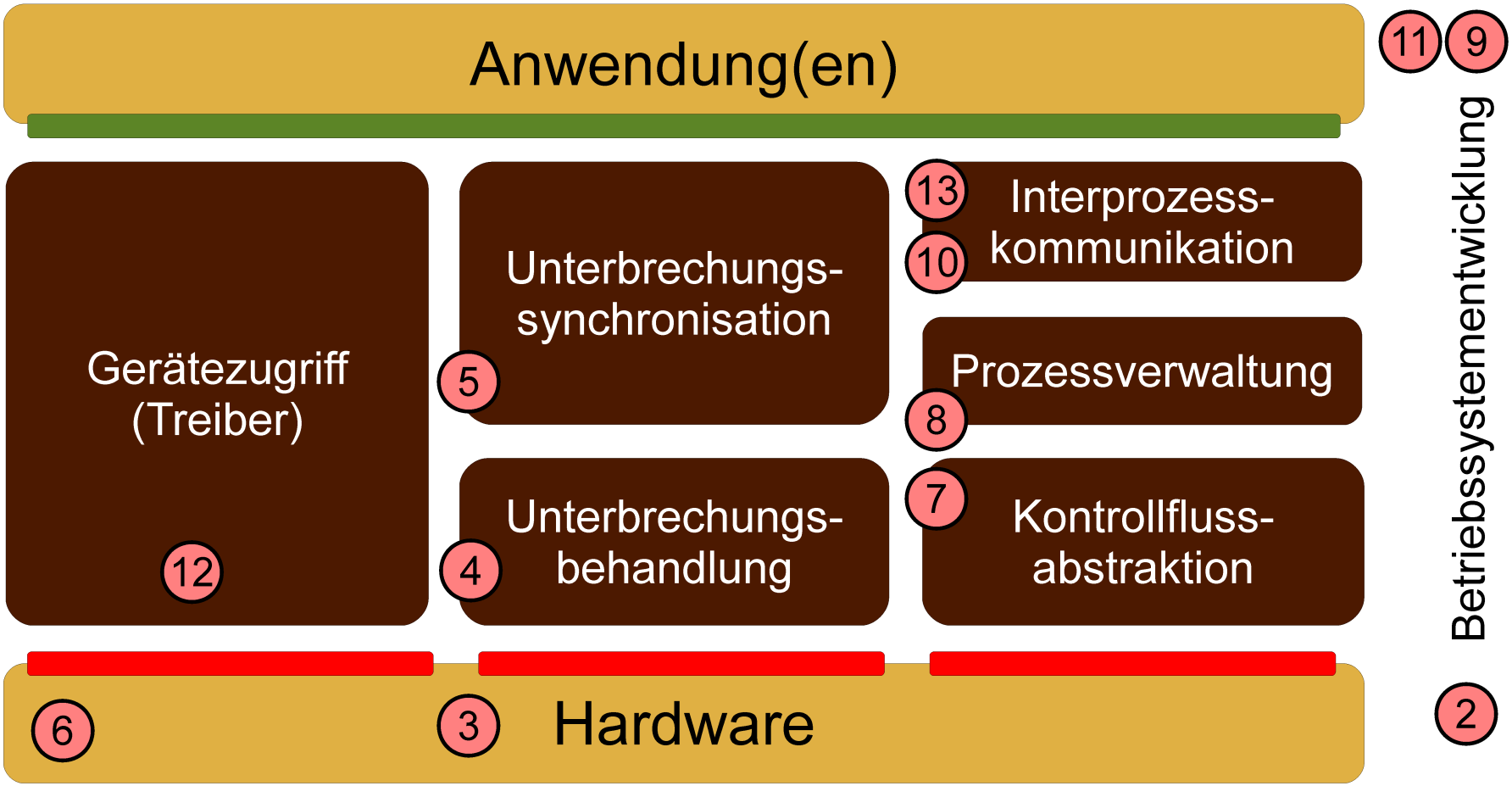
Themenübersicht Übung

Am Beispiel von: OOStuBS, MPStuBS

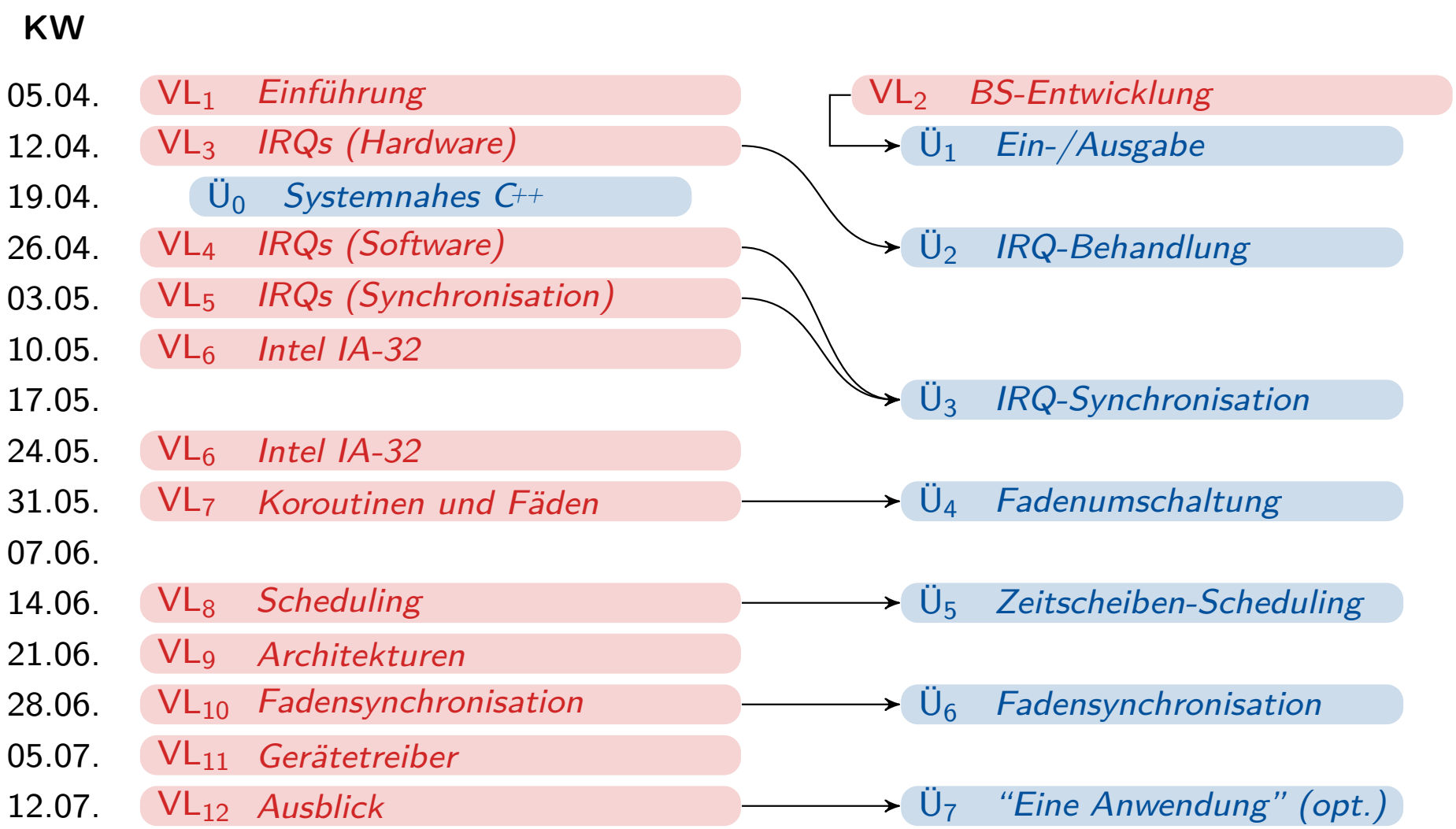


Themenübersicht Vorlesung

Am Beispiel von: x86, MC68k, TriCore; Windows, Linux



Verzahnung von Vorlesung und Übungsaufgaben



VL_2 Betriebssystementwicklung (oft ein harter Kampf!)

- Erste Schritte

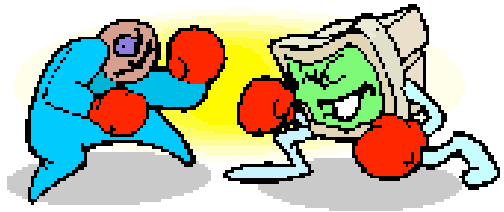
  Wie bringt man sein System auf die Zielhardware?

  - Übersetzen und Linken für "nackte Hardware"
  - Bootvorgang

- Testen und Fehlersuche

  Was tun, wenn das System nicht reagiert?

  - "printf"-*Debugging*
  - Simulatoren
  - *Debugger*
  - *Remote debugging*
  - Hardwareunterstützung



VL_{3-4} Programmunterbrechungen

TUHH
Technische Universität Hamburg

- im Prinzip
  - Unterbrechungen, *Traps* und Ausnahmen
  - Vektortabellen
  - geschachtelte Unterbrechungen
  - *spurious interrupts*
- beim PC
  - CPU und APIC
  - Unterbrechungen in Multiprozessorsystemen
- Behandlung im Betriebssystem
  - Kopplungsfunktion
  - Zustandssicherung

VL_5 Unterbrechungssynchronisation

TUHH
Technische Universität Hamburg

- Zusammenspiel zwischen Unterbrechungsbehandlung und “normalem” Kontrollfluss
  - Ursache und Problem
  - Kontrollflussebenenmodell
- Hardware-Mechanismen zur “harten Synchronisation”
  - `cli` und `sti`
  - Unterbrechungsebenen
- Software-Mechanismen zur “weichen Synchronisation”
  - Pro-/Epilogmodell und Varianten
  - Unterbrechungstransparente Algorithmen

Das Intel IA-32 Programmiermodell

- Die Entwicklung der x86 CPU-Familie
  - vom 8086 bis zum Core i7
- Relikte und Eigenarten (*quirks*)
  - *Real Mode*
  - *A20 Gate*
- Neuerungen des *Protected Mode*
  - Ringe und Schutzmodell
  - *Task*-Modell
- Hardwarevirtualisierung

VL_7 Programmfäden

TUHH
Technische Universität Hamburg

- Realisierung von Programmfäden
  - beim MC68k, Infinion TriCore, Intel x86
  - Fortsetzungen und Koroutinen als Basis
  - Implementierung des Kontextwechsels
- Fadenmodelle
  - leicht vs. schwer vs. federgewichtig vs. ...
  - Umsetzung in einer Systemfamilie

VL_8 Ablaufplanung

TUHH
Technische Universität Hamburg

- Kurze Wiederholung und Vertiefung
  - Grundprinzipien
  - Klassifikation
  - neue Strategien
- Beispiele aus der Praxis
  - Windows
  - Linux
  - Scheduling auf Multiprozessor-Systemen
- Herausforderungen beim Betriebssystembau
  - Zusammenspiel Ablaufplanung $\rightleftharpoons$ Unterbrechungssynchronisation

VL_9 Architekturen

TUHH
Technische Universität Hamburg

- Wie organisiert man ein Betriebssystem: Architekturmodelle
  - Bibliotheken
  - Monolithen
  - Mikrokerne
  - Exokerne
  - Hypervisor
- Geschichte: Revolutionen, Religionen ...und die Realität
  - Bewertungskriterien
  - Erfolgs- und Misserfolgsgeschichten
- Beispiele aus der Praxis
  - OS360, Unix, Linux, L4, Windows
  - exoKernel, xen, vmware
  - ...

VL_{10} Fadensynchronisation

TUHH
Technische Universität Hamburg

- Grundsätzliches
  - Vorraussetzungen
  - aktives und passives Warten
- Synchronisationsprimitiven
  - *Mutex*, *Semaphore* und *Condition*
  - aus der Sicht des BS-Entwicklers
- spezielle Probleme
  - Wechselwirkung Synchronisation $\rightleftharpoons$ Ablaufplanung
  - Fortschrittsgarantie und Verklemmung
- Beispiele aus der Praxis
  - Synchronisationsprimitiven in Windows

VL_{11} Geräteprogrammierung

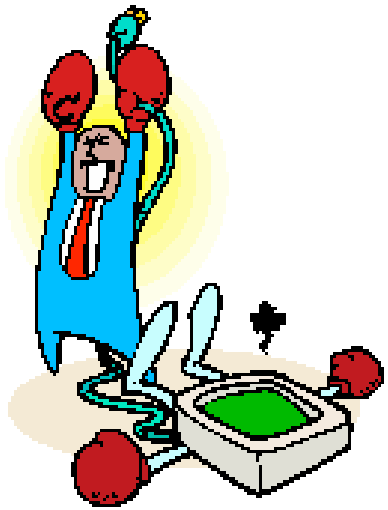
TUHH
Technische Universität Hamburg

- Treiber und ihre Bedeutung
  - Vielfalt von Geräten
  - Probleme
- Komponentenmodell für Treiber
  - Struktur eines E/A-Systems
  - Treiberklassen und -schnittstellen
- Beispiele aus der Praxis
  - Windows
  - Linux

Quo Vadis Betriebssysteme?

- Zusammenfassung
  - Zusammenfassung des Lernstoffes
  - Diskussion der Evaluationsergebnisse
  - Tipps und Hinweise für die Prüfung
- Ausblick: Neue Herausforderungen
  - Multi- und Manycore Systeme
  - Heterogene Hardware
- Ausblick: Systeme aus der Forschung
  - Corey
  - Barrelfish/Multikernel
  - Factored OS
  - TxOS
  - ...

Das Ziel der Veranstaltung



Viel Spaß!

Referenzen I

TUHH
Technische Universität Hamburg

[1] Andrew S. Tanenbaum. *Structured Computer Organization*. Fifth. Prentice Hall PTR, 2006. ISBN: 978-0131485211.