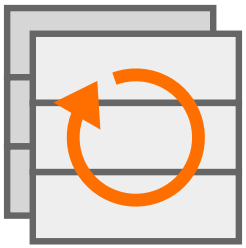


Operating System Group

TUHH

Technische Universität Hamburg

BSB – Übung 7: Anwendung für StuBS

Yannick Loeck

2022-07-13

Inhalt

TUHH
Technische Universität Hamburg

BSB - HÜ7

Eine richtige Anwendung schreiben

Worum geht es in der Übung heute:

- Ihr habt ein OS geschrieben
- Jetzt wollt ihr auch was damit machen
- Optionale Aufgabe

Was jetzt?

TUHH
Technische Universität Hamburg

Ihr bekommt einiges mitgeliefert:

- Zufallsgenerator
- Speicherallokator (Heap)
- Grafikkarte
- PNG-Decoder
- Festplattentreiber/Dateien

Was jetzt?

TUHH
Technische Universität Hamburg

Ihr bekommt einiges mitgeliefert:

- Zufallsgenerator
- **Speicherallokator (Heap)**
- **Grafikkarte**
- PNG-Decoder
- **Festplattentreiber/Dateien**

Allokator

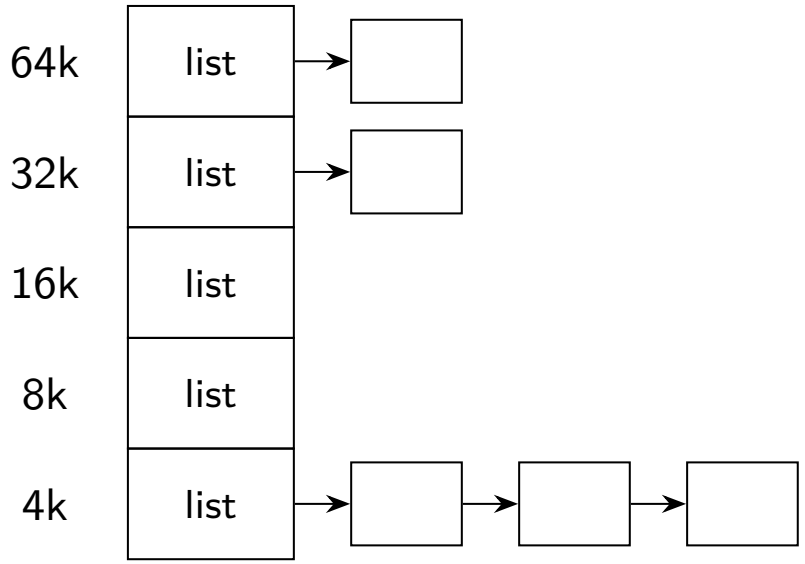
Dynamischer Allokator

TUHH
Technische Universität Hamburg

- Speicher in globaler Variable: z.B. `char mem[16MiB]`
- Allokator arbeitet darauf: `malloc, free`
- Buddy Allocator: Speicher in Zweierpotenzen
- Gut für kleine Objekte: nächste Zweierpotenz nah dran
- Buddy: Speicherblöcke haben Partner
- Partner können verschmolzen werden, oder Blöcke aufgeteilt

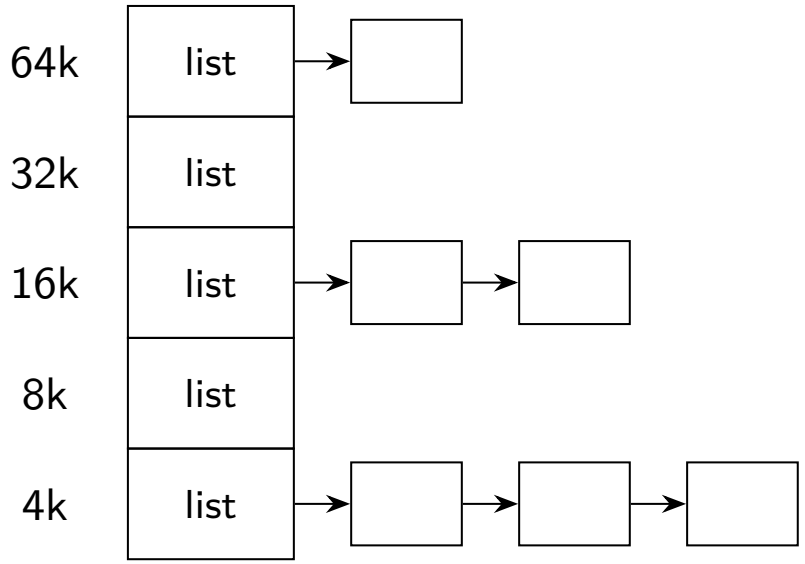
2 Datenstrukturen (in utils/alloc_buddy.h)

- Array von Listen/Buckets
  - Buckets enthalten freie Blöcke bestimmter Größe
  - Einer für jede mögliche Blockgröße (2^n)
  - `alloc(7000)` $\rightarrow$ `alloc(8192)`



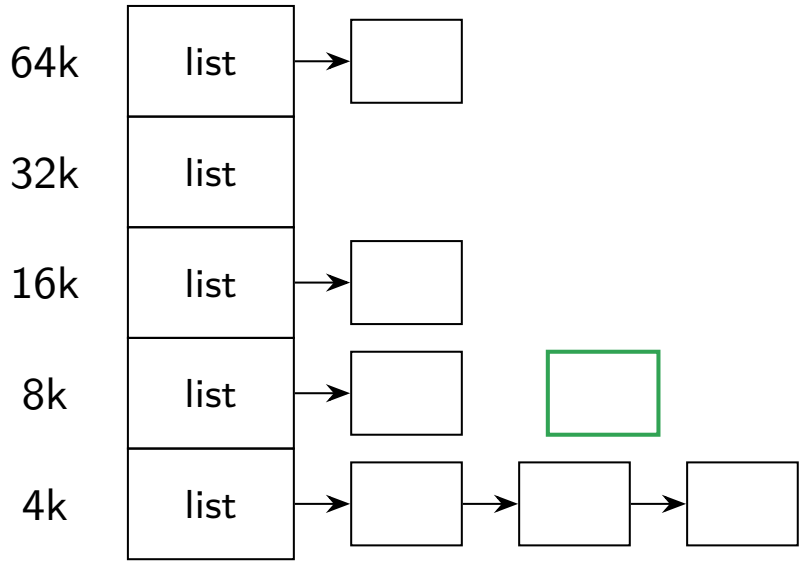
2 Datenstrukturen (in utils/alloc_buddy.h)

- Array von Listen/Buckets
  - Buckets enthalten freie Blöcke bestimmter Größe
  - Einer für jede mögliche Blockgröße (2^n)
  - `alloc(7000)` $\rightarrow$ `alloc(8192)`



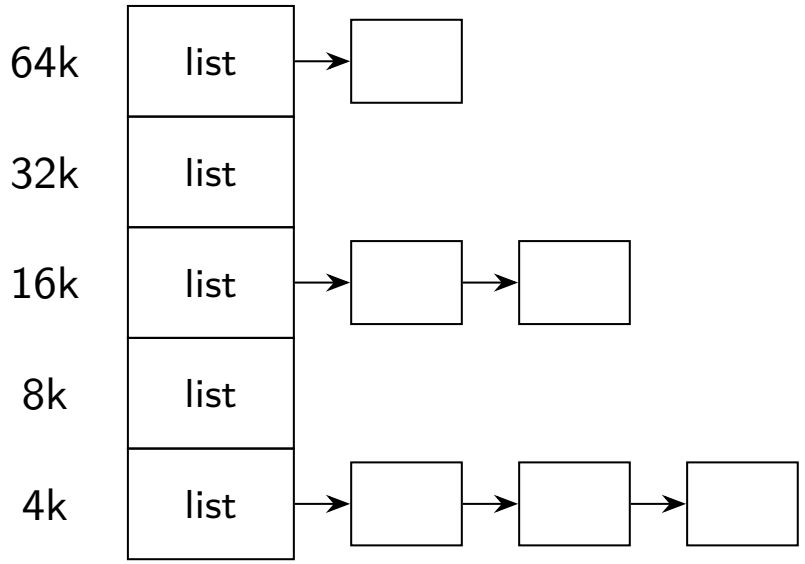
2 Datenstrukturen (in utils/alloc_buddy.h)

- Array von Listen/Buckets
  - Buckets enthalten freie Blöcke bestimmter Größe
  - Einer für jede mögliche Blockgröße (2^n)
  - `free(8192)`



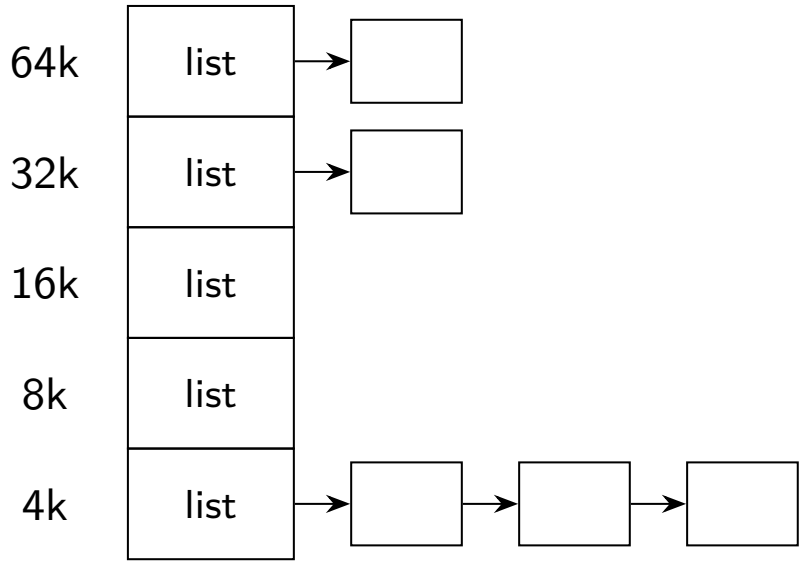
2 Datenstrukturen (in utils/alloc_buddy.h)

- Array von Listen/Buckets
  - Buckets enthalten freie Blöcke bestimmter Größe
  - Einer für jede mögliche Blockgröße (2^n)
  - `free(8192)`



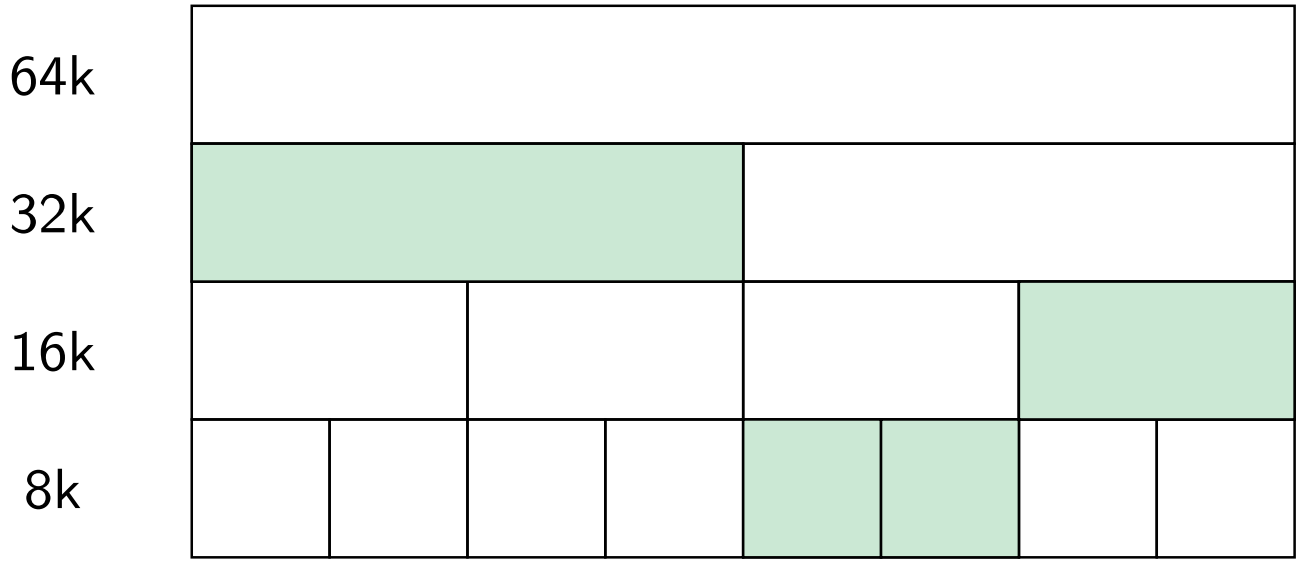
2 Datenstrukturen (in utils/alloc_buddy.h)

- Array von Listen/Buckets
  - Buckets enthalten freie Blöcke bestimmter Größe
  - Einer für jede mögliche Blockgröße (2^n)
  - `free(8192)`



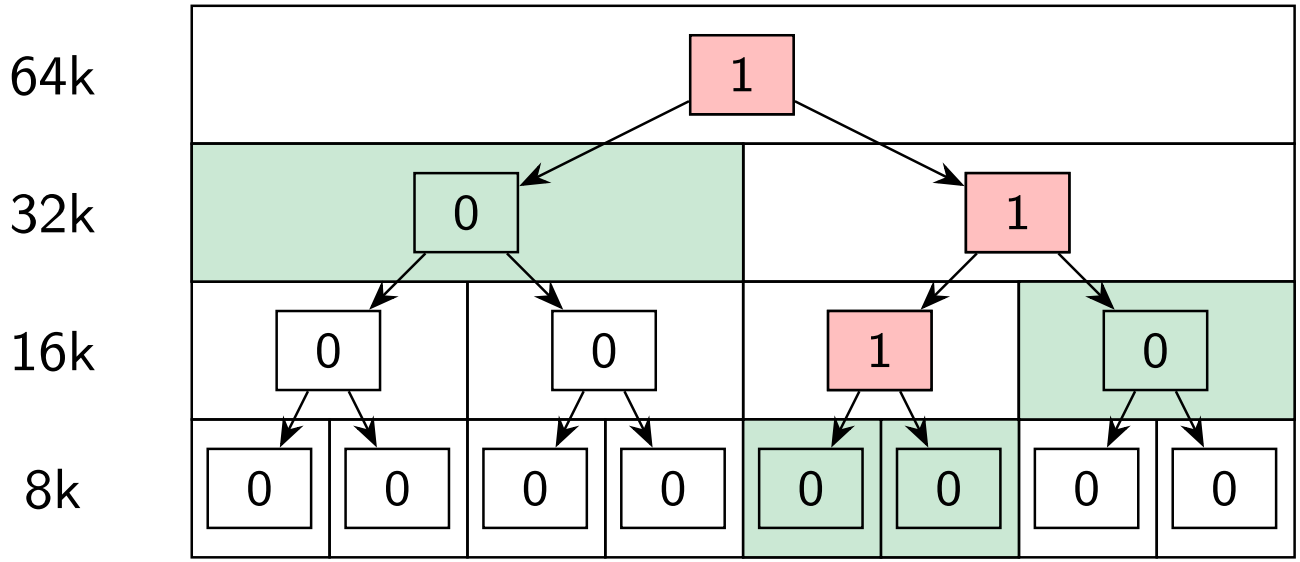
2 Datenstrukturen (in utils/alloc_buddy.h)

- (Vollständiger) Binärer Baum
  - Ist Block gesplittet/Ist der Nachbar frei?
  - Ebenenweise linearisiert im Speicher, 0-indexed
  - Elternknoten einfach ausrechnen: `(int) (idx-1)/2`



2 Datenstrukturen (in utils/alloc_buddy.h)

- (Vollständiger) Binärer Baum
  - Ist Block gesplittet/Ist der Nachbar frei?
  - Ebenenweise linearisiert im Speicher, 0-indexed
  - Elternknoten einfach ausrechnen: `(int) (idx-1)/2`



Grafik

Grafik

TUHH
Technische Universität Hamburg

- Nicht mehr nur Textmodus, sondern Pixel einzeln ansprechen
- Grafikkarte von grub initialisieren lassen
- Dafür OS ELF in CDROM image einfügen lassen mit:
  `make {qemu, kvm}-iso`
- Benutzen: **syscall/guarded_graphics.h**
  `graphics.init(true)` in main
- Multiboot-config in **boot/multiboot/config.inc** anpassen:
  `MULTIBOOT_HEADER_FLAGS equ MULTIBOOT_VIDEO_MODE`

Debugging

Über `SerialStream`, auf stdout umleiten mit `make *-serial`

Wie funktioniert die Grafik?

TUHH
Technische Universität Hamburg

- VESA-Grafikkarte mit Video Memory (VMEM)
- Man könnte direkt in VMEM schreiben
- **Aber**: Wann wird das gelesen?

- Lösung: Double Buffering, Kopien des VMEM in front und back buffer
- Front buffer regelmäßig in VMEM kopieren (z.B. in `Watch` epilogue)
- Warum 2 Buffer? Nur schreiben wenn Operationen fertig sind

- Grafikoperationen in back buffer
- Pointer von back und front buffer tauschen (`switchBuffer`)

Grafik - Text und Bilder

TUHH
Technische Universität Hamburg

- *Text*: braucht jetzt auch Fonts
- Einige in **graphics/fonts/** mitgeliefert
- *Bilder*: als PNGs gespeichert, Grafikkarte benötigt bitmaps
- Library zur Konversion mitgeliefert
- Achtung: Speicherintensiv, Stack größer machen (mindestens 8K)

Dateisystem

Dateisystem

TUHH
Technische Universität Hamburg

- Minix: Lehrbetriebssystem von A. Tanenbaum
- Minix3-Dateisystem, Festplattentreiber
- Zusätzlich: `Ramdisk : public BlockDevice`
  Abbild für Dateisystem im RAM, von grub geladen
- Operationen: `open`, `read`, `write`, `close`, `mount`, `unmount`

- Daten müssen nicht direkt in OS-binary mit rein
- Erstellen von Dateisystem mit `mkfs.minix -3`
- Einfüllen von Dateien mit **fs/tool/fstool**
- Dokumentiert in **fs/README.md** und doxygen

Aufteilung der Festplatte

TUHH
Technische Universität Hamburg

- Treiber greift über Block Layer auf Festplatte zu
- Blockorientiertes Gerät $\rightarrow$ Nicht Byte-Adressierbar

- Festplatte in Blöcke/Zonen aufgeteilt
- Einstellbare Blocklänge, Minimum durch Festplatte gegeben
- Default Block Size: 1024B
- Zonen als Sammlung von 2^n Blöcken

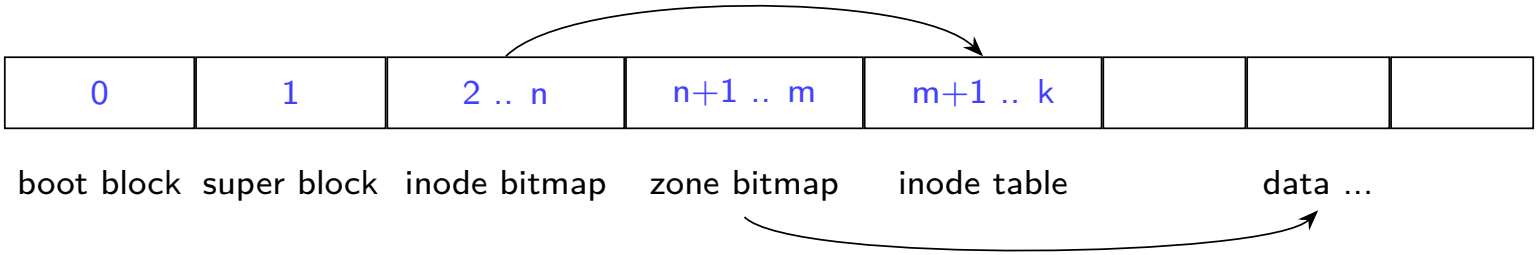
Festplattentreiber

TUHH
Technische Universität Hamburg

Drei Operationen:

- `fix()`: Block-Objekt in Speicher anlegen
  - Pointer auf Stück Speicher mit Blockgröße
  - Blocknummer und dirty bit
  - Treiber macht read/write Operationen in diesen Speicher
- `sync()`: Block aus Speicher auf Festplatte schreiben
  - Dirty bit: Block im RAM divergiert
  - Synchronisieren
- `unfix()`: `sync()` und Freigabe

Dateisystem - Aufbau



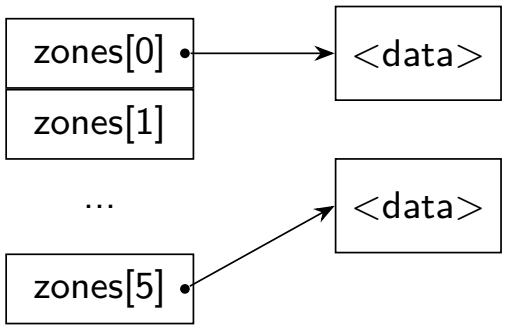
- 1. Block (bnr 0): Boot Block, 1024B, bleibt frei, für MBR
- 2. Block (bnr 1): Superblock, Dateisystemweite Informationen

Ressourcenbelegung

- Inode Bitmap: Für Inode-Tabelle
- Zone Bitmap: Für Daten
- Inode-Tabelle
  - Array von `struct inode`
  - Inodes kleiner als Block, dürfen nicht über Block-Grenzen hinweg gehen
- Rest: Daten, mit Zonen- statt Blocknummern

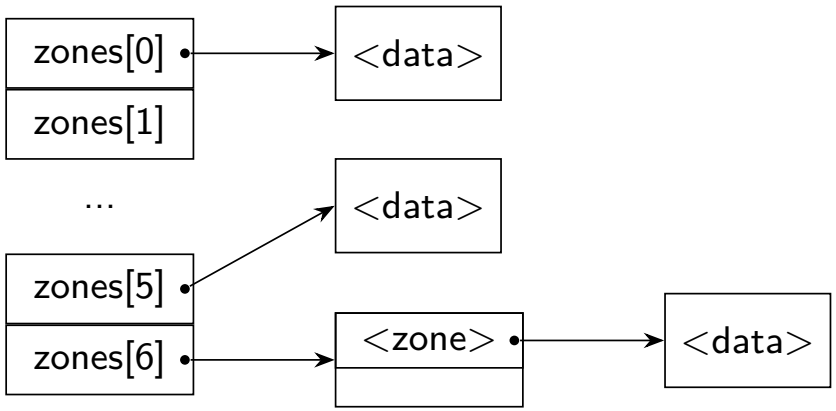
Dateisystem - Inodes

- `mode`: Zugriffsrechte, Directory (`drwxr-xr-x`)
- `uid, gid`: Besitzer der Datei
- `zones[]`: Inhalt der Datei, Indirektionsstufen
- Zonen 0-5: *Direkte* Zoneneinträge
- Zone 6: *Indirekte* Zone



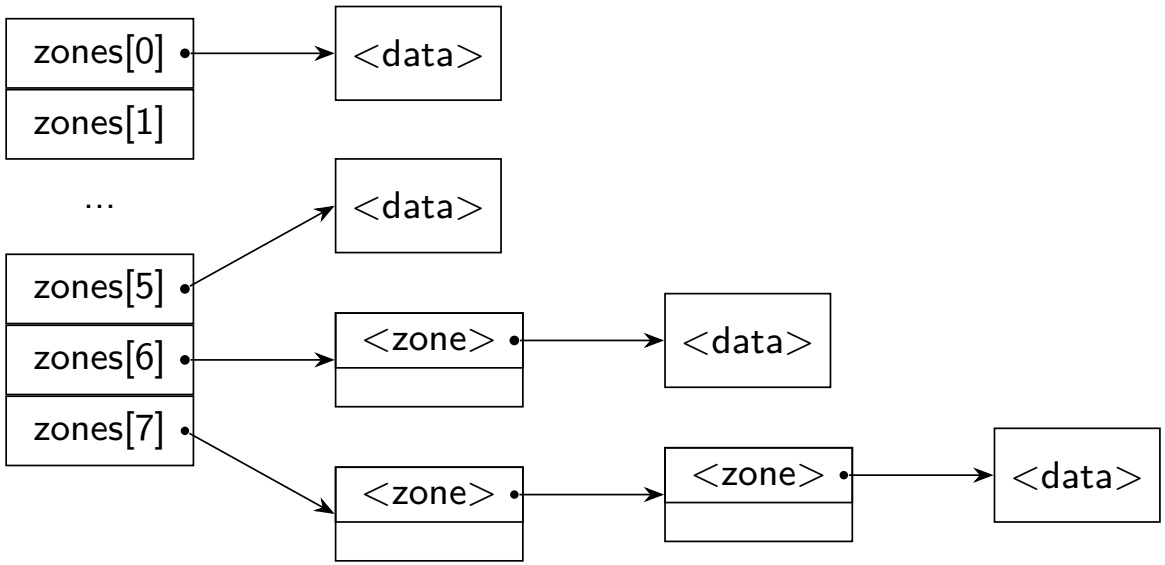
Dateisystem - Inodes

- `mode`: Zugriffsrechte, Directory (`drwxr-xr-x`)
- `uid, gid`: Besitzer der Datei
- `zones[]`: Inhalt der Datei, Indirektionsstufen
- Zonen 0-5: *Direkte* Zoneneinträge
- Zone 6: *Indirekte* Zone
- Zone 7: *Doppelt indirekte* Zone

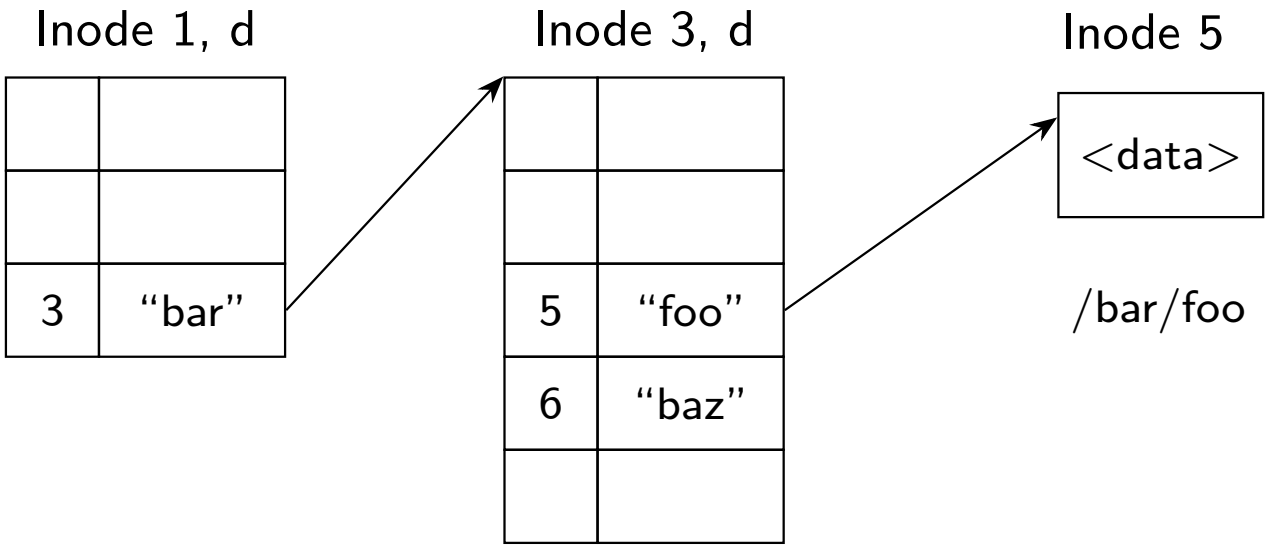


Dateisystem - Inodes

- `mode`: Zugriffsrechte, Directory (`drwxr-xr-x`)
- `uid, gid`: Besitzer der Datei
- `zones[]`: Inhalt der Datei, Indirektionsstufen
- Zonen 0-5: *Direkte* Zoneneinträge
- Zone 6: *Indirekte* Zone
- Zone 7: *Doppelt indirekte* Zone

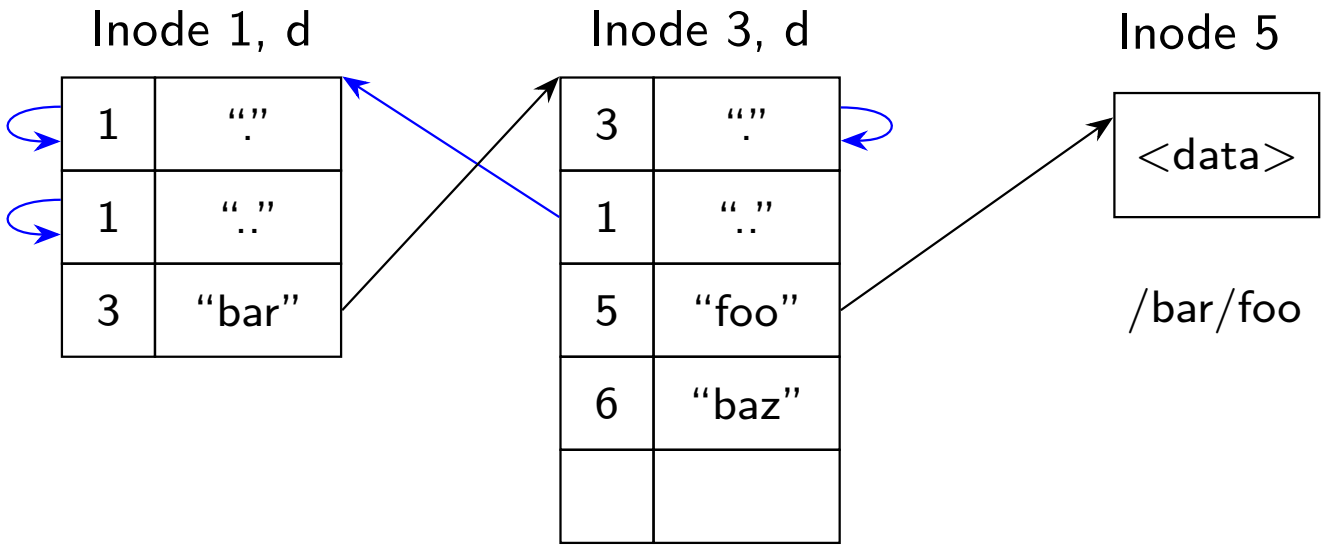


Dateien und Verzeichnisse



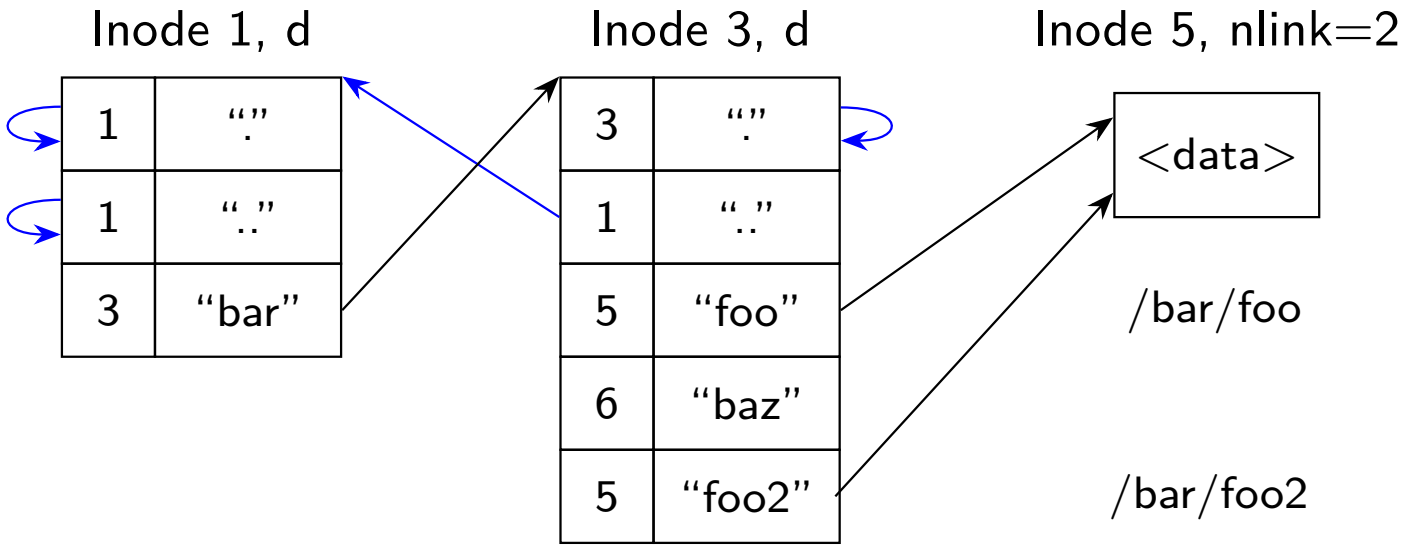
- Dateiname kommt aus Verzeichnis
- Verzeichnis ist Datei mit Modus “d” (directory flag)
- In Verzeichnis-Inode keine Daten sondern dir-entries
  `{uint32 inodenum, char name[60]}`
- Verzeichnisname kommt aus darüberliegendem Verzeichnis
- Das geht bis zum root-Verzeichnis (inode 1)

Dateien und Verzeichnisse



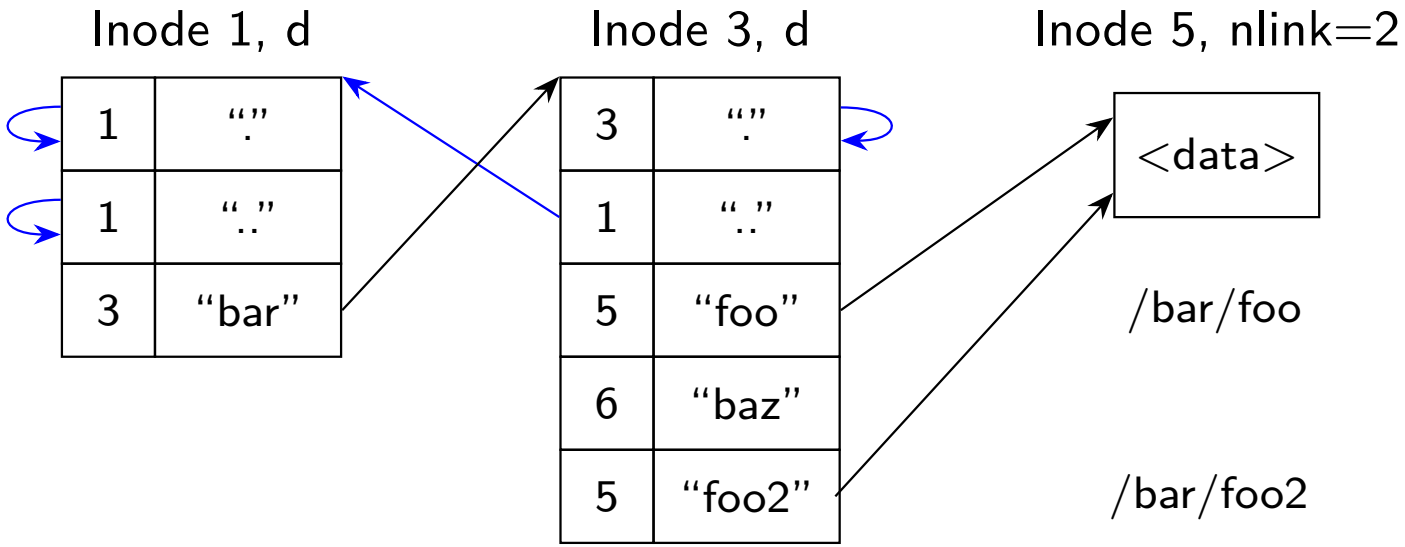
- Besondere Einträge “.” und “..”
- Mehrere Einträge für selbe Datei: Hardlinks
- Auch in unterschiedlichen Verzeichnissen
- Dafür haben Inodes reference counter `nlink`
- `nlink` von Verzeichnissen: min. 2

Dateien und Verzeichnisse



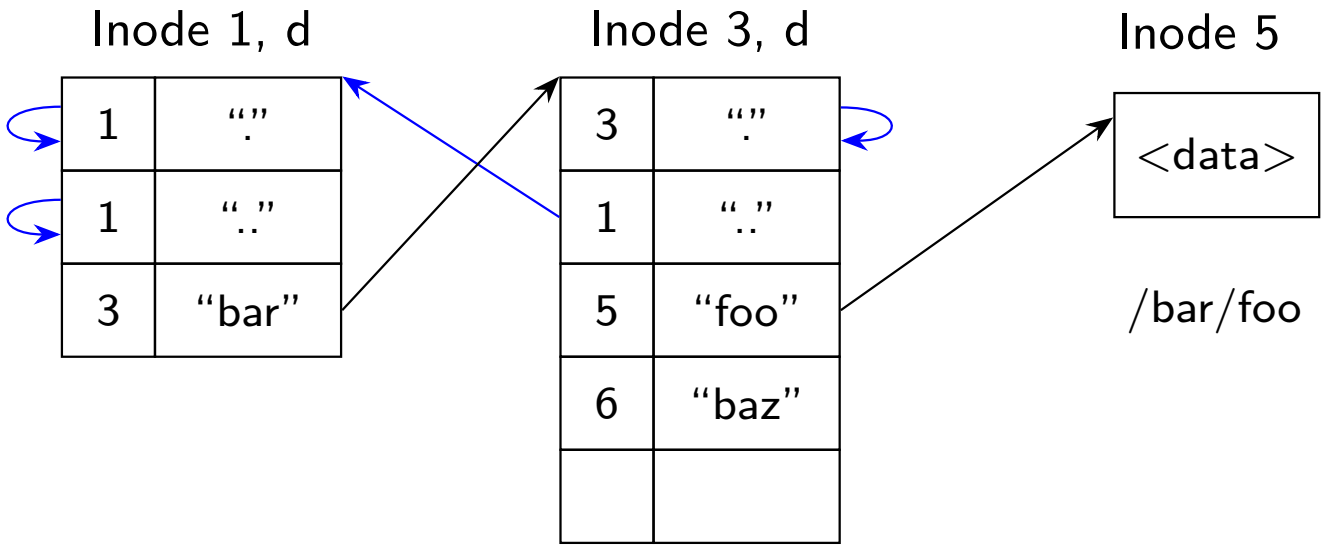
- Besondere Einträge “.” und “..”
- Mehrere Einträge für selbe Datei: Hardlinks
- Auch in unterschiedlichen Verzeichnissen
- Dafür haben Inodes reference counter `nlink`
- `nlink` von Verzeichnissen: min. 2

Dateien und Verzeichnisse



- Löschen im Verzeichnis:
  1. Entfernen des Inode-Eintrags
  2. Dekrementieren des Referenzzählers
- `nlink == 0`: Inode nicht mehr benötigt
  1. Daten freigeben
  2. Einträge in Zone und Inode bitmap als unbenutzt markieren

Dateien und Verzeichnisse



- Löschen im Verzeichnis:
  1. Entfernen des Inode-Eintrags
  2. Dekrementieren des Referenzzählers
- `nlink == 0`: Inode nicht mehr benötigt
  1. Daten freigeben
  2. Einträge in Zone und Inode bitmap als unbenutzt markieren

Danke!