

Betriebssysteme und nebenläufige Programmierung

SoSe 2026

Roland Wismüller
Betriebssysteme / verteilte Systeme
roland.wismueller@uni-siegen.de
Tel.: 0271/740-4050, Büro: H-B 8404

Stand: 20. März 2026


Betriebssysteme und nebenläufige Programmierung

SoSe 2026

9 Ein-/Ausgabe und Dateisysteme

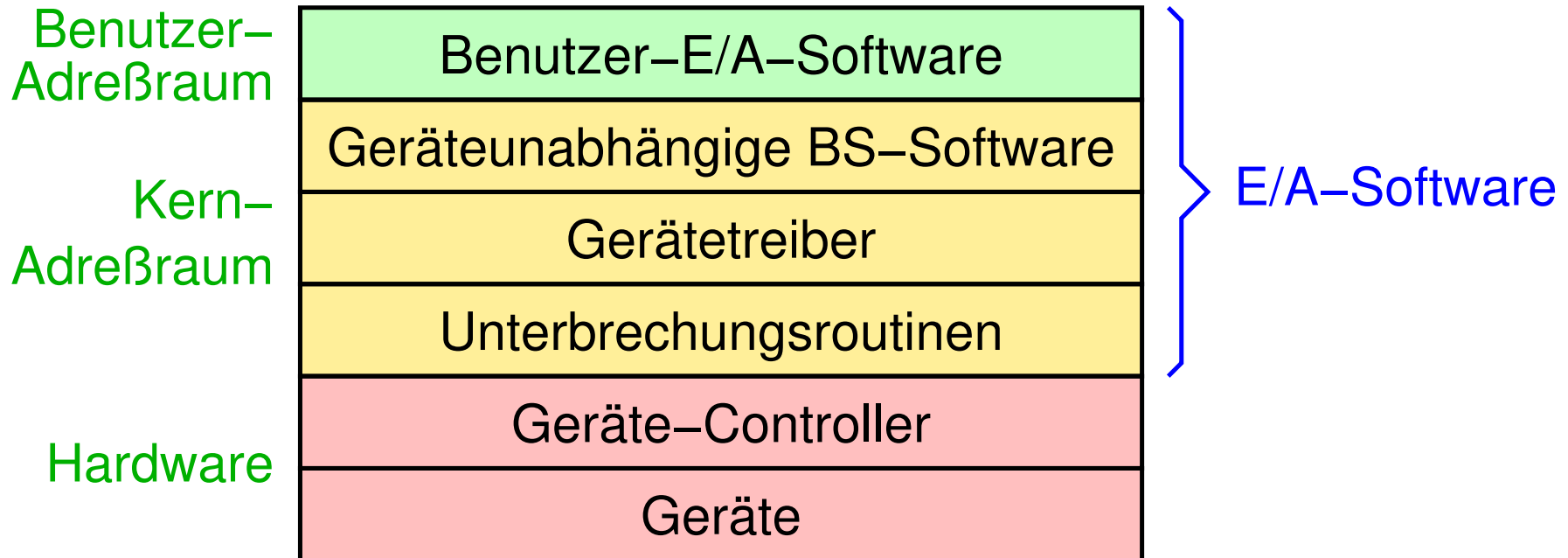


Inhalt:

- ➔ Schichten der E/A-Software
- ➔ Ansätze zur Durchführung der E/A
- ➔ Festplatten
- ➔ (Dateiverwaltung,  **1.5.3**)
- ➔ Realisierung von Dateisystemen

- ➔ Tanenbaum 5.2 - 5.4, 6.1-6.3, 6.4.5
- ➔ Stallings 11.2 - 11.6, 12.1, 12.3, 12.5-12.7
- ➔ Nehmer/Sturm 10.1, 9

➔ Schichten bei der Ein-/Ausgabe:



➔ In der Regel für jeden Gerätetyp eigene Controller und Gerätetreiber

Gerätetreiber

- ➔ Geräteabhängiger Teil der E/A-Software
 - ➔ kommuniziert direkt mit dem jeweiligen Geräte-Controller
- ➔ Heute i.d.R. in Kern-Adreßraum eingebunden
 - ➔ beim Hochfahren des BSs oder zur Laufzeit
 - ➔ Problem: fehlerhafte Treiber können zu BS-Abstürzen führen
 - ➔ Lösungsmöglichkeiten:
 - ➔ zertifizierte Treiber
 - ➔ Universalgeräte mit universellen Treibern
 - ➔ Treiber als Systemprozesse im Benutzeradreßraum
- ➔ Einheitliche Schnittstelle zwischen Treibern und BS
 - ➔ nur Unterscheidung block-/zeichenorientierte Geräte



Geräteunabhängige E/A-Software

- ➔ Aufgaben:
 - ➔ Einheitliche Schnittstelle für Gerätetreiber
 - ➔ Namensgebung für Geräte, Zuordnung zu Treibern
 - ➔ Zugriffsschutz
 - ➔ Pufferung von Daten
 - ➔ Fehlerbehandlung
 - ➔ Anforderung und Freigabe von Geräten
 - ➔ für exklusive Nutzung, z.B. CD-Brenner
 - ➔ Verdeckung von Unterschieden der Geräte
 - ➔ z.B. unterschiedliche Blockgrößen

Geräteunabhängige E/A-Software ...

- ➔ Bereitgestellte Grundfunktionen:
 - ➔ Öffnen eines Geräts
 - ➔ Argumente: Gerätename, Betriebsparameter
 - ➔ Ergebnis: *Handle* zum Zugriff auf das Gerät
 - ➔ Schließen des Geräts
 - ➔ Lesen / Schreiben von Daten(blöcken)
- ➔ In UNIX:
 - ➔ Geräte sind im Namensraum des Dateisystems abgebildet
 - ➔ z.B. /dev/mouse, /dev/hda
 - ➔ Zugriff auf Geräte über normale Dateioperationen

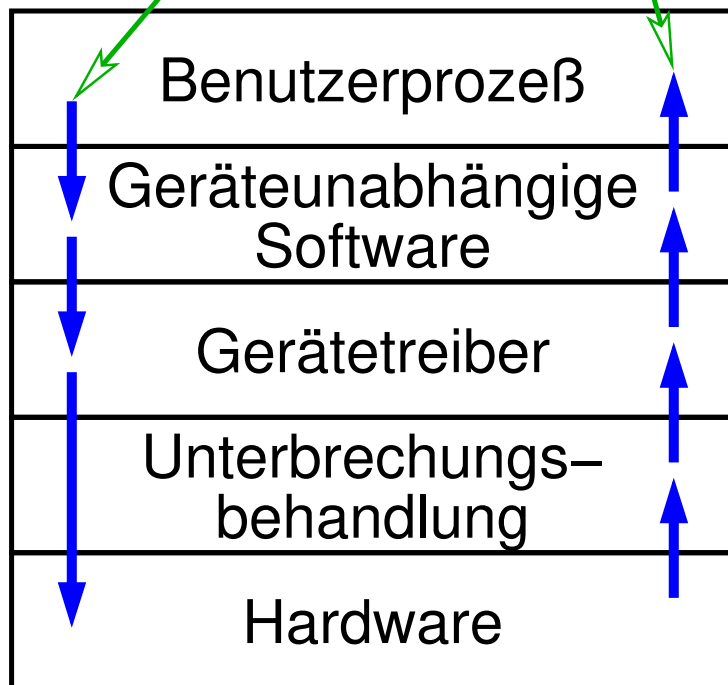


Benutzer-E/A-Software

- ➔ Bibliotheksfunktionen
 - ➔ z.B. zur formatierten Ein-/Ausgabe von Zeichenketten
- ➔ Spooling-System
 - ➔ Hintergrundprozesse nehmen Auftrag entgegen und führen eigentliche E/A durch
 - ➔ E/A-Geräte müssen nicht mehr exklusiv zugeteilt werden
 - ➔ Beispiele: Drucker, Mail-System

Zusammenfassung: Kontrollfluß im E/A-System

E/A-Anforderung E/A-Antwort



E/A-Funktionen:

E/A-Aufruf, Formatierung, Spooling

Benennung, Schutz, Puffern, Belegen

Gerätregister schreiben/lesen, Status prüfen

Treiber aktivieren, wenn E/A beendet

E/A-Operation durchführen

Programmierte E/A

- ➔ Gerätetreiber wartet nach einem Auftrag an den Controller aktiv (in einer Warteschleife) auf das Ergebnis
 - ➔ **aktives Warten** (*busy waiting*)
- ➔ Beispiel: BS-Code zur Ausgabe auf einen Drucker

Puffer aus Benutzer-Adreßraum in Kern-Adreßraum kopieren;

// buf = Puffer im Kern, count = Länge

Treiber

```
for (i=0; i<count; i++) {  
    while (printer.status != READY); // aktives Warten!  
    printer.data = buf[i];           // ein Zeichen ausgeben  
}
```

scheduler (); // Rückkehr in Benutzermodus

- ➔ Nachteil: ineffizient
 - ➔ CPU könnte der Zwischenzeit andere Aufgaben erfüllen

Interrupt-gesteuerte E/A

- ➔ Treiber beauftragt den Controller und kehrt sofort zurück
 - ➔ auftraggebender Thread wird ggf. blockiert
- ➔ Controller sendet Interrupt an CPU, wenn der Auftrag erledigt ist
 - ➔ d.h. Gerät ist wieder bereit
 - ➔ i.d.R.: Interrupt-Nummer identifiziert das Gerät
- ➔ Treiber behandelt die Unterbrechung
 - ➔ auftraggebender Thread wird ggf. wieder rechenbereit gesetzt
- ➔ Sinnvoll bei langsamen E/A-Geräten



Interrupt-gesteuerte E/A ...

➔ Beispiel: Ausgabe auf Drucker

BS-Code

```
Benutzer-Puffer kopieren;  
// ==> buf, count           Treiber  
while (printer.status  
        != READY);  
printer.data = buf[0];  
i = 1;  
aktuellen Thread T blockieren;  
scheduler();
```

Interrupt-Handler (im Treiber)

```
if (i == count) {  
    Thread T bereit setzen;  
} else {  
    printer.data = buf[i];  
    i = i+1;  
}  
Interrupt bestätigen;  
Rückkehr aus Interrupt-Routine;
```

Direkter Speicherzugriff (*Direct Memory Access, DMA*)

- ➔ Transport der Daten zwischen Controller und Speicher erfolgt nicht durch die CPU, sondern durch separate Hardware (**DMA-Controller**)
 - ➔ oft auch in Geräte-Controller integriert
- ➔ Treiber sendet Geräte-Adresse, Startadresse und Länge der Daten, sowie Transferrichtung an DMA-Controller
- ➔ DMA-Controller erzeugt Interrupt als Fertigmeldung
- ➔ Vorteile: Entlastung der CPU, Einsparung von Interrupts
- ➔ Sinnvoll (nur) bei Übertragung größerer Datenblöcke
 - ➔ DMA-Controller arbeitet nebenläufig mit CPU
 - ➔ DMA-Controller aber oft langsamer als CPU



Direkter Speicherzugriff (*Direct Memory Access, DMA*) ...

➔ Beispiel: Ausgabe auf Drucker

Treiber-Code

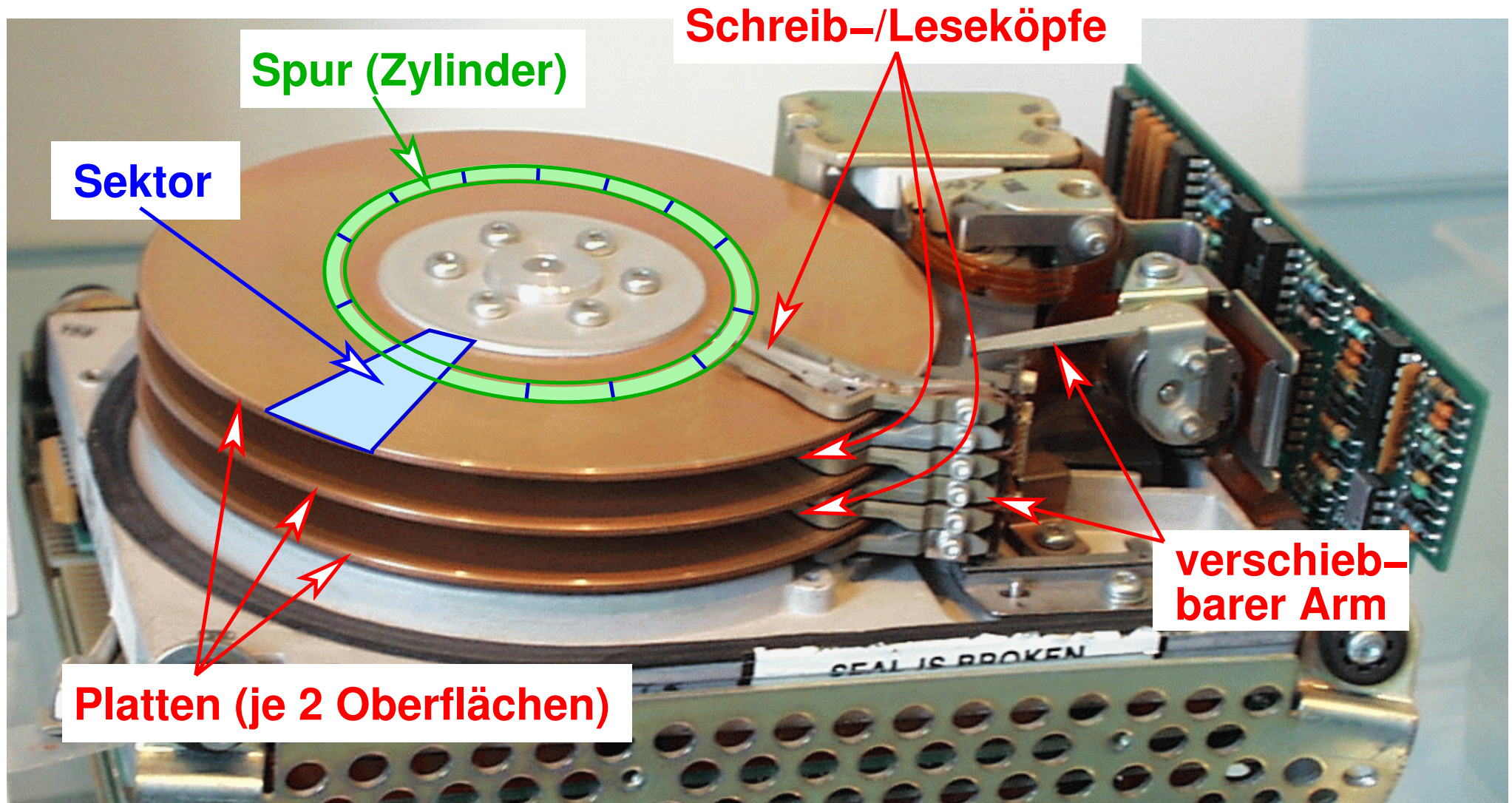
```
evtl.: Benutzer-Puffer kopieren;  
// ==> buf, count  
DMA-Controller aufsetzen  
// Parameter: buf, count, printer  
aktuellen Thread T blockieren;  
scheduler ( ) ;
```

Treiber

Interrupt-Handler (im Treiber)

```
Thread T bereit setzen;  
Interrupt bestätigen;  
Rückkehr aus Interrupt-Routine;
```

Aufbau einer Festplatte





Aufbau einer Festplatte ...

- ➔ Datenspeicherung durch Magnetisierung der Plattenbeschichtung
- ➔ Spurwechsel durch Verschieben des Arms (d.h. aller Köpfe)
 - ➔ Menge der jeweils übereinanderliegenden Spuren: Zylinder
- ➔ Einteilung der Festplatte:
 - ➔ Oberfläche (Kopf), Zylinder, Sektor
 - ➔ Sektor einer Spur nimmt einen Datenblock auf (meist 512 Byte, teilweise 4096 Byte)
 - ➔ äußere Spuren besitzen mehr Sektoren als innere
 - ➔ Größenordnungen:
 - ➔ 4 - 12 Oberflächen, 10^5 - 10^6 Zylinder, 500 - 4000 Sektoren
- ➔ Adressierung der Blöcke durch fortlaufende Nummer (LBA)
 - ➔ Controller bildet Blocknummer auf (Kopf, Zylinder, Sektor) ab

Zugriffszeit einer Festplatte

- ➔ Drei bestimmende Faktoren:
 - ➔ Suchzeit (Anfahren der gewünschten Spur)
 - ➔ im Durchschnitt ca. 5 - 10 ms
 - ➔ Rotationsverzögerung (bis Sektor unter dem Kopf ist)
 - ➔ im Durchschnitt ca. 2 - 6 ms (5400 - 15000 U/min)
 - ➔ Dauer der Datenübertragung
 - ➔ ca. 2 - 100 μ s pro Block (bis ca. 200MB/s)
- ➔ Zugriffszeit dominiert durch Suchzeit, daher:
 - ➔ Dateien möglichst in aufeinanderfolgenden Sektoren
 - ➔ meist auch: *Prefetching* und *Caching*
 - ➔ geeignetes Scheduling der Plattenzugriffe

Solid State Disks (SSDs)

- ➔ Datenspeicherung in isoliertem Gate eines MOS-FETs
 - ➔ bis zu 4 Bit pro Transistor
 - ➔ Auslesen durch Bestimmung der Schwellenspannung
- ➔ Organisation in Blöcken (4 - 16 KiB)
 - ➔ Überschreiben erst nach vorherigem Löschen möglich
 - ➔ Löschen nur in größeren Blöcken (256 - 512 KiB) möglich
 - ➔ ineffizient beim Ändern weniger Bytes
- ➔ Zugriffszeit („Suchzeit“): ca. 20 - 500 μ s
- ➔ Dauer der Datenübertragung: wenige μ s pro Block (bis ca. 3GB/s)
 - ➔ durch paralleles Auslesen vieler Speicherzellen



SSDs: Diskussion

➔ Vorteile:

- ➔ keine beweglichen Teile (leise, stoßfest)
- ➔ leicht
- ➔ energieeffizient (im Ruhebetrieb)
- ➔ hohe Datenübertragungsrate

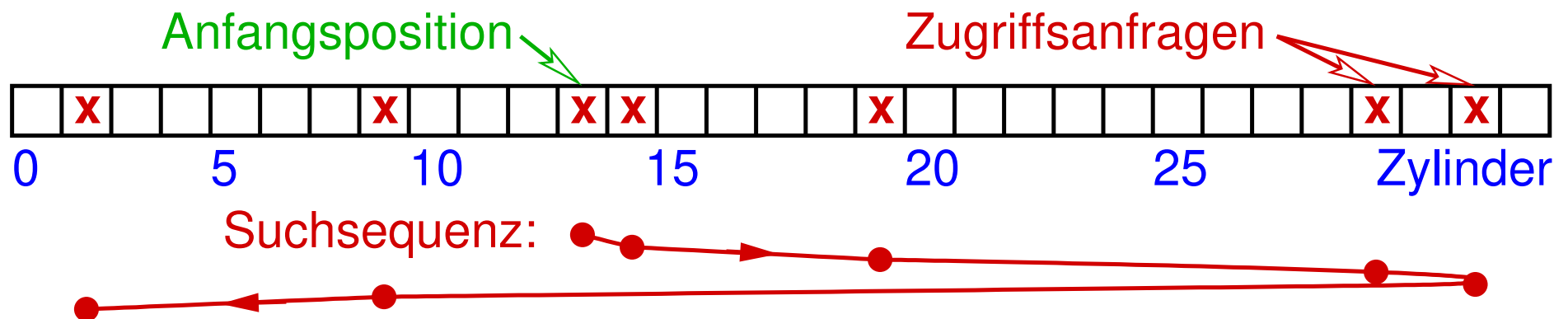
➔ Nachteile:

- ➔ begrenzte Überschreibbarkeit (1000 - 100 000 mal)
- ➔ begrenzter Datenerhalt (ca. 10 Jahre)
- ➔ schwierigere Ausfallvorhersage

➔ Im Serverbereich daher i.d.R. konventionelle Festplatten

Scheduling von Plattenzugriffen

- ➔ FCFS: viele unnötige Bewegungen des Plattenarms
- ➔ SSF (*Shortest Seek First*)
 - ➔ Zugriffe in der Nähe der aktuellen Position bevorzugen
 - ➔ Problem: Unfairness, Verhungerung möglich
- ➔ Aufzug-Algorithmus
 - ➔ erst in eine Richtung, bis es in dieser Richtung keine Anfragen mehr gibt; dann Richtung wechseln

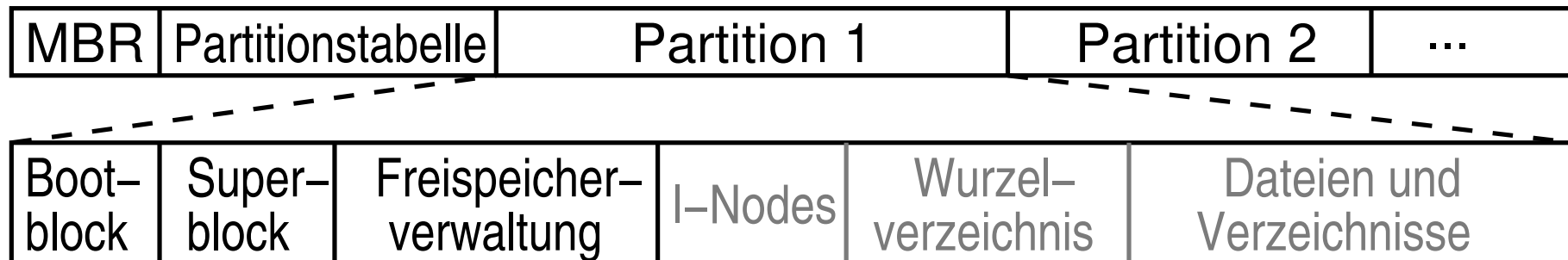


Schichtenmodell

- ➔ Datenträgerorganisation
 - ➔ einheitliche Schnittstelle zu allen Datenträgern
 - ➔ Datenträger als Folge von Blöcken betrachtet
- ➔ Blockorientiertes Dateisystem
 - ➔ Realisierung von Dateien
- ➔ Dateiverwaltung
 - ➔ Dateinamen und Verzeichnisse

Datenträgerorganisation

- ➔ Evtl. Einteilung des Datenträgers in Partitionen
- ➔ Partition wird als Folge von (logischen) Blöcken betrachtet
 - ➔ Blöcke fortlaufend nummeriert
- ➔ Typisches Layout einer Festplatte (UNIX):



- ➔ MBR: *Master Boot Record*
- ➔ Superblock: Verwaltungsinformation der Partition
 - ➔ Größe, Blockgröße, ...

Datenträgerorganisation: Aufgaben

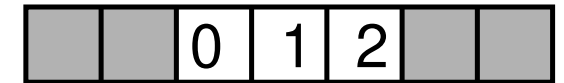
- ➔ Formatieren des Datenträgers
- ➔ Lesen / Schreiben von Blöcken
- ➔ Verwaltung freier Blöcke
 - ➔ vgl. dynamische Speicherverwaltung!
 - ➔ meist: Bitvektoren statt Freispeicherliste
 - ➔ Bit i gesetzt \Leftrightarrow Block i belegt
 - ➔ Bitvektor wird auf Datenträger gespeichert
- ➔ Verwaltung defekter Blöcke
 - ➔ ebenfalls über Bitvektoren

Blockorientiertes Dateisystem

➔ Datei als Folge von Blöcken realisiert

➔ Zuteilung von Blöcken an Dateien:

➔ zusammenhängende Belegung



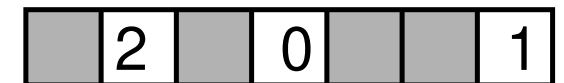
➔ Datei durch Anfangsblock und Blockanzahl beschrieben

➔ sehr gute Performance beim Lesen

➔ Problem: Speicherverwaltung (vgl. 6.2)

➔ Anfügen an Dateien, Fragmentierung, ...

➔ verteilte Belegung

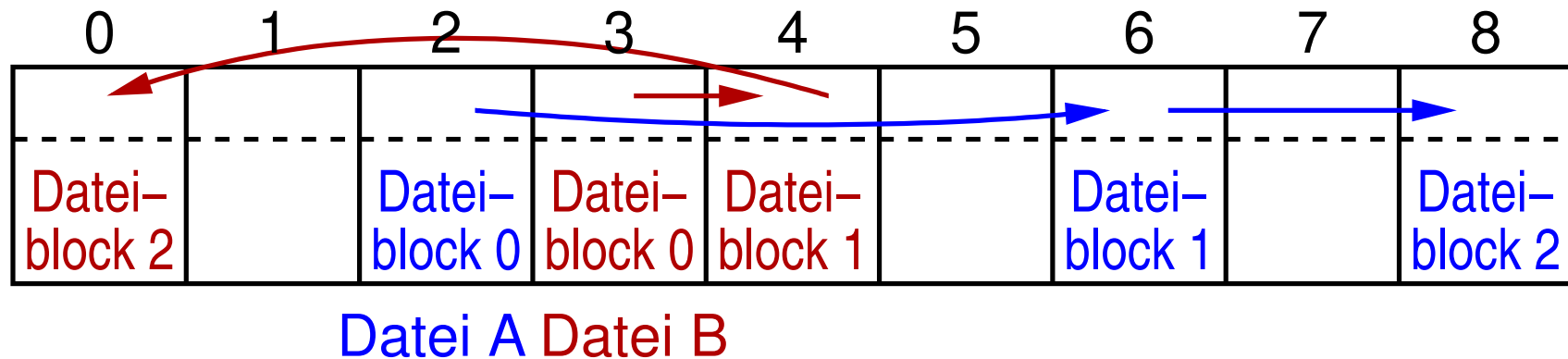


➔ einfache Speicherverwaltung, schlechtere Performance

➔ Praxis: Belegung möglichst zusammenhängend

Blockorientiertes Dateisystem: verteilte Belegung

- ➔ Realisierung durch verkettete Listen
- ➔ Verkettung innerhalb der Blöcke



- ➔ Nachteile: wahlfreier Zugriff ineffizient, Dateiblock-Länge keine Zweierpotenz mehr
- ➔ Realisierung durch externe Tabelle (*File Allocation Table*, FAT)
- ➔ Verkettung ausserhalb der Blöcke
- ➔ Tabelle kann (teilweise) im Hauptspeicher gehalten werden

Blockorientiertes Dateisystem: verteilte Belegung ...

- ➔ Realisierung durch Index-Knoten (*I-Nodes*)
 - ➔ jeder Datei ist eine Datenstruktur (*I-Node*) zugeordnet
 - ➔ *I-Node* enthält Tabelle mit Verweisen auf Dateiblöcke
 - ➔ wegen Speicherplatzbedarf: Tabelle ggf. mehrstufig
 - ➔ Tabelle kann im Hauptspeicher gehalten werden
 - ➔ schneller wahlfreier Zugriff auf Datei möglich
 - ➔ Beispiel: *I-Nodes* in UNIX
 - ➔ ähnliches Konzept auch in NTFS

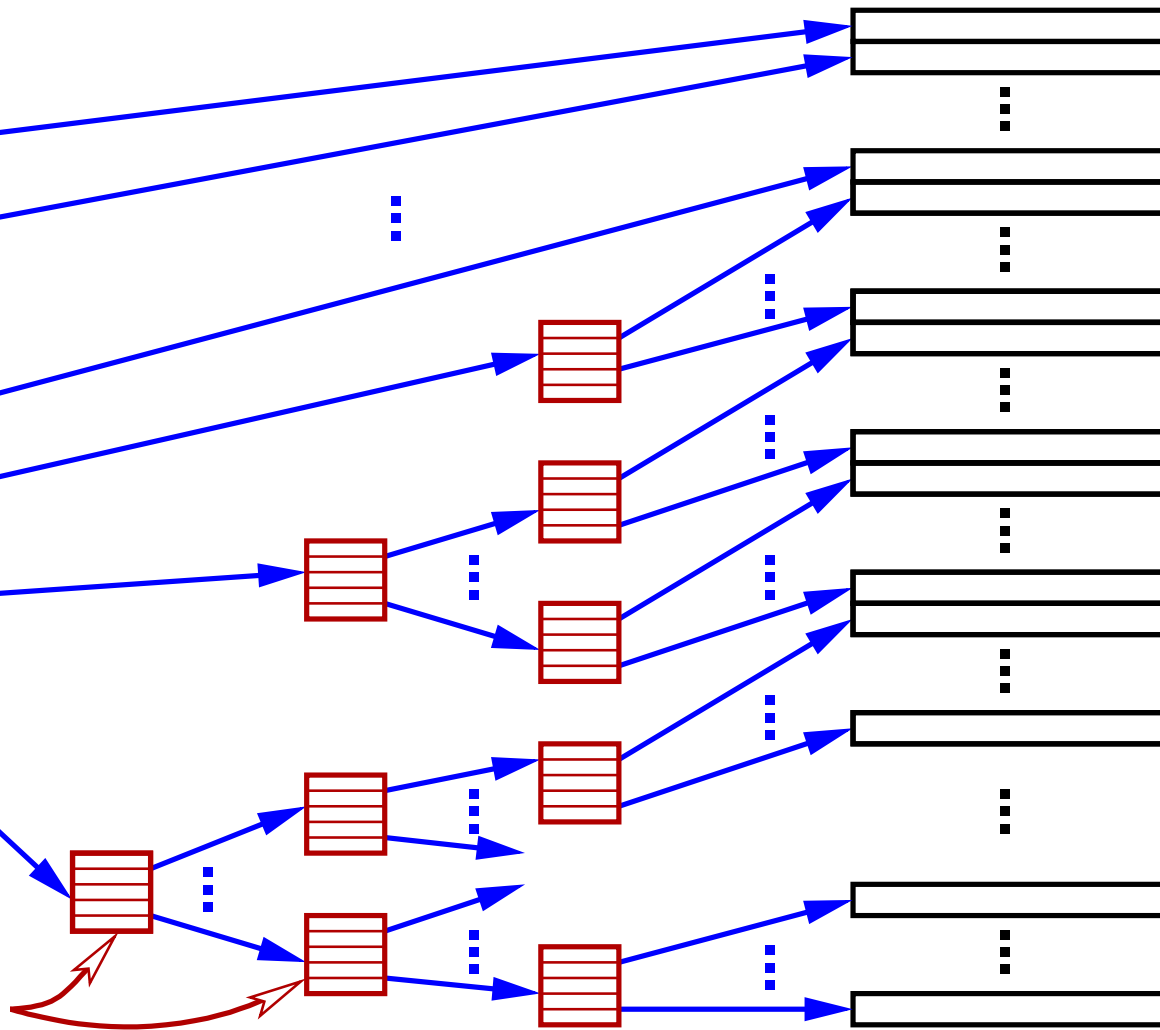


Blockorientiertes Dateisystem: *I-Nodes* in UNIX

I-Node

Adresse Block 0
Adresse Block 1
...
Adresse Block 11
Einfach indirekt
Zweifach indirekt
Dreifach indirekt

Dateiblocke (1 KB) mit
max. 256 Verweisen



Dateiblocke mit Daten der Datei

Dateiverwaltung

- ➔ Aufgabe: Realisierung von Verzeichnissen
- ➔ Verzeichnis enthält für jede Datei u.a.:
 - ➔ Dateiname, Plattenadresse (erster Dateiblock, *I-Node*)
- ➔ Dateiattribute (u.a.):
 - ➔ Eigentümer, Schutzinformation (Zugriffsrechte), ...
 - ➔ Dateityp, Sperre, archiviert, ...
 - ➔ Erstellungszeit, Zeit der letzten Änderung, ...
- ➔ Speicherung der Dateiattribute: i.a. im Index-Knoten (*I-Node*), bei FAT-Dateisystemen im Verzeichnis

Konzepte moderner Dateisysteme

- ➔ Vermeidung von Datenverlusten
 - ➔ bei Systemabsturz oder Stromausfall
- ➔ *Logical Volumes*
 - ➔ dynamisch veränderbare Partitionen
- ➔ *Snapshots*
 - ➔ z.B. für Backup vor einem System-Update
- ➔ *Subvolumes*
 - ➔ mehrere Sub-Dateisysteme in einer Partition
- ➔ Deduplizierung
 - ➔ Vermeidung von Redundanz



Vermeidung von Datenverlusten

- ➔ Aus Performanzgründen: Datenblöcke werden zunächst nur im Hauptspeicher verändert
 - ➔ periodisches Rückschreiben auf Festplatte ca. alle 30s
- ➔ Bei Absturz oder Stromausfall: Änderungen gehen verloren
 - ➔ dadurch evtl. Struktur des Dateisystems inkonsistent
 - ➔ ggf. Reparatur durch Dateisystem-Check
- ➔ Bessere Lösung: *Journaling*-Dateisystem
 - ➔ Journal beschreibt alle Änderungen im Dateisystem
 - ➔ sequentiell auf Platte gespeichert (keine Kopfbewegung)
 - ➔ von Zeit zu Zeit: Ausführen der Operationen und Löschung im Journal



Vermeidung von Datenverlusten ...

- ➔ Bei Absturz oder Stromausfall:
 - ➔ Operationen im Journal werden beim Neustart ausgeführt
- ➔ Anwendung z.B. in Linux `ext4` und Windows NTFS
- ➔ Details zu `ext4`
 - ➔ Journal ist zirkulärer Puffer
 - ➔ Schreib-/Leseoperationen über eigenes Gerät (JBD)
 - ➔ JBD stellt sicher, daß zusammenhängende Journal-Einträge atomar gespeichert werden
 - ➔ Verwendung des Journals i.a. nur für Metadaten
 - ➔ aber auch für alle Daten möglich



Logical Volumes

- ➔ Abstraktion der physischen Datenträger (Partitionen)
- ➔ *Logical Volume* kann sich über mehrere physische Partitionen erstrecken
 - ➔ auch dynamisch erweiterbar
 - ➔ realisiert durch Schicht oberhalb Datenträgerorganisation
 - ➔ Abbildung von logischem Block auf Partition und Block
- ➔ Beispiele: LVM (Linux), Dynamische Datenträger (Windows)
- ➔ Oft: zusätzliche (Software-)RAID-Funktion
 - ➔ RAID 1 (*Mirroring*): Daten werden zweifach gespeichert
 - ➔ RAID 5: Verteilung + Redundanz durch Paritätsbildung
 - ➔ Ausfall einer von N Platten kann toleriert werden



Snapshots, Subvolumes, Deduplizierung

- ➔ Zugrundeliegende Technik: *Copy-on-Write*
- ➔ Z.B. bei Erstellung eines *Snapshots*
 - ➔ Daten werden nicht kopiert, nur die Verweise auf die Plattenblöcke (*reflink*)
 - ➔ bei Änderung eines Blocks: Block wird kopiert, Verweis wird aktualisiert
 - ➔ damit: Kopie sehr schnell und platzsparend
- ➔ Snapshots i.d.R. erstellt bei Software-Updates
- ➔ Alter Snapshot kann als *Subvolume* direkt gemounted werden
- ➔ Deduplizierung: ersetze Block durch *reflink*, falls möglich
 - ➔ dazu: Prüfsummen aller Blöcke vergleichen

- ➔ Schichten der E/A-Software
 - ➔ Benutzer-E/A-Software, geräteunabhängige BS-Software, Gerätetreiber, Unterbrechungsrouinen
 - ➔ Treiber: geräteabhängig, zur Laufzeit in BS-Kern geladen
- ➔ Durchführung der E/A: programmierte E/A, Interrupts, DMA
- ➔ Festplatten: eingeteilt in Oberfläche, Zylinder, Sektor
- ➔ Aufbau von Dateisystemen: Schichtenmodell
 - ➔ Datenträgerorganisation
 - ➔ Freispeicherverwaltung
 - ➔ Blockorientiertes Dateisystem (Datei = Menge von Blöcken)
 - ➔ zusammenhängende / verteilte Belegung von Blöcken
 - ➔ Dateiverwaltung
 - ➔ Verzeichnisse, Dateiattribute