
Betriebssysteme (BS)

11. Dateisysteme

<https://sys.cs.tu-dortmund.de/de/lehre/ss24/bs>

01.07.2024

Peter Ulbrich

peter.ulbrich@tu-dortmund.de

bs-problems@ls12.cs.tu-dortmund.de

<https://sys.cs.tu-dortmund.de/de/lehre/kummerkasten>

In Teilen basierend auf *Betriebssysteme* von Olaf Spinczyk, Universität Osnabrück

Wiederholung: Betriebsmittel

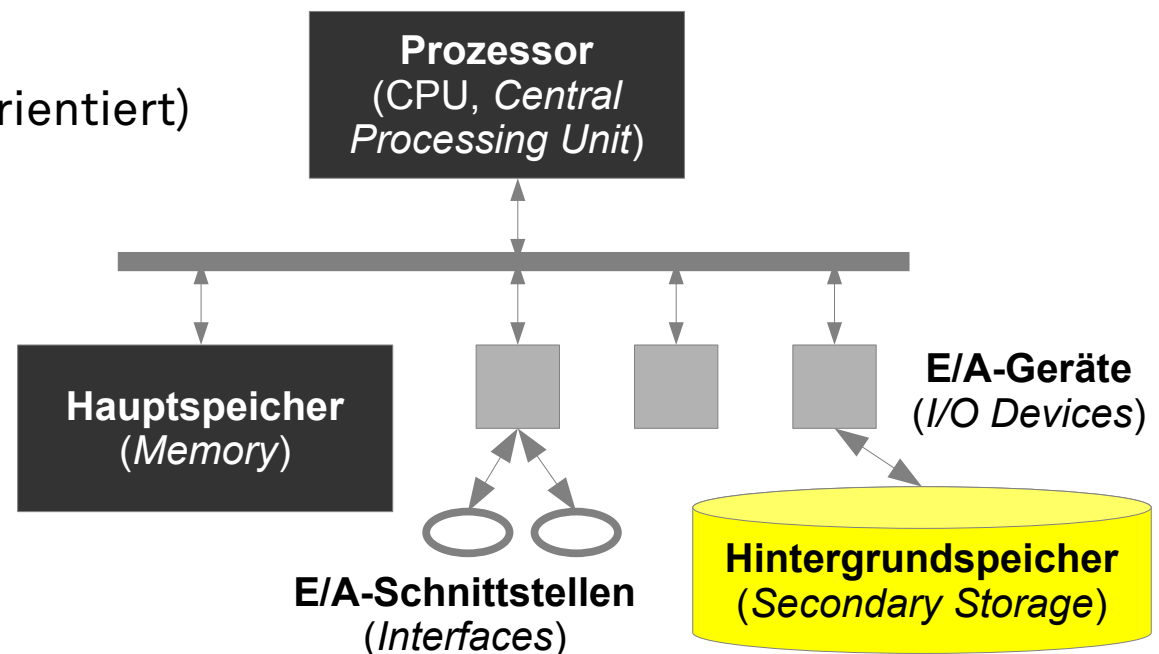
- Das Betriebssystem hat folgende Aufgaben:
 - Verwaltung der Betriebsmittel des Rechners
 - Schaffung von Abstraktionen, die Anwendungen einen einfachen und effizienten Umgang mit Betriebsmitteln erlauben

■ Bisher:

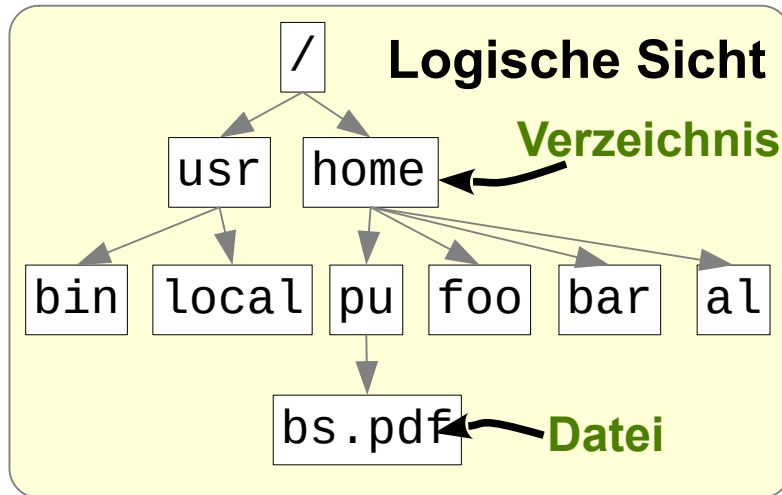
- Prozesse
- Arbeitsspeicher
- E/A-Geräte (insb. blockorientiert)

■ Heute: Dateisysteme

- Organisation des Hintergrundspeichers

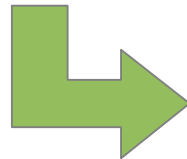


Hintergrundspeicher

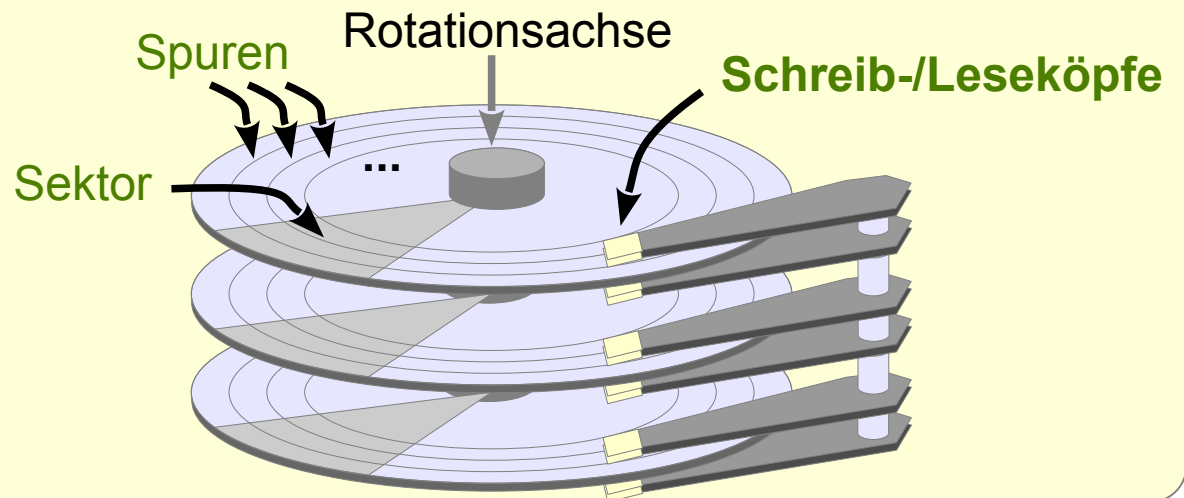


Dateisysteme erlauben die dauerhafte Speicherung großer Datenmengen.

Abbildung



Physikalische Sicht



Festplatte mit 6 Oberflächen

Das Betriebssystem stellt den Anwendungen die logische Sicht zur Verfügung und muss diese effizient realisieren.

Inhalt

- Dateien
- Freispeicherverwaltung
- Verzeichnisse
- Dateisysteme
- Pufferspeicher
- Dateisysteme mit Fehlererholung
- Zusammenfassung

Tanenbaum

6: Dateisysteme

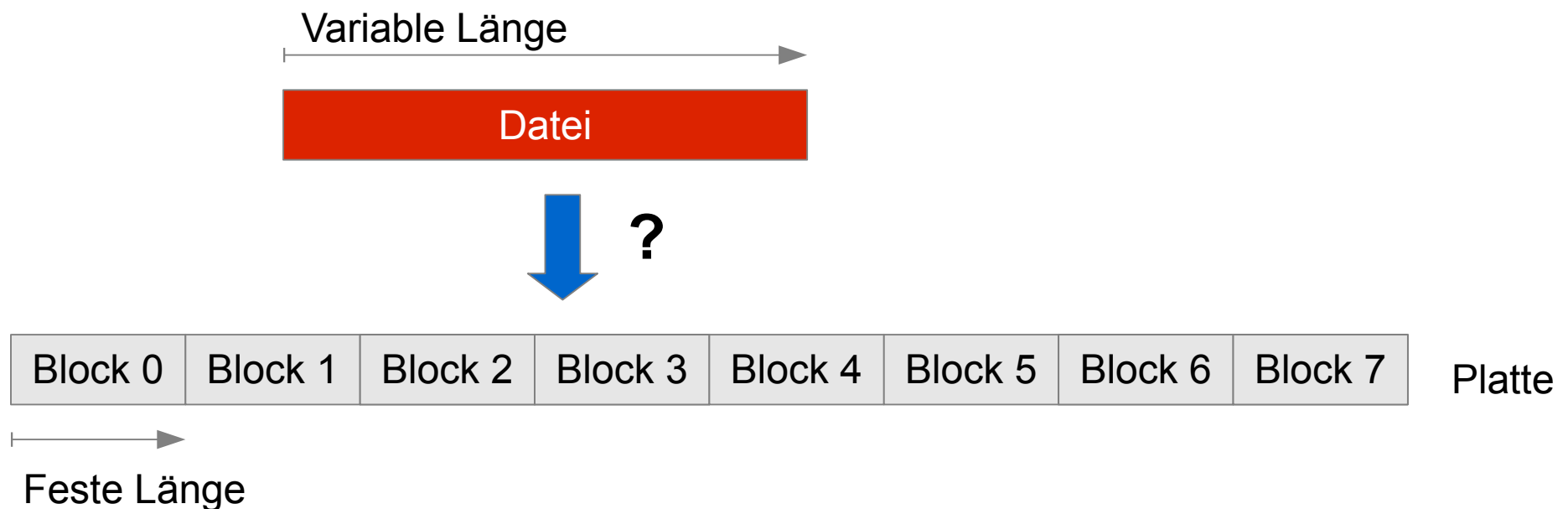
Silberschatz

10: File System

11: Implementing File Systems

Speicherung von Dateien

- Dateien benötigen oft **mehr als einen Block** auf der Festplatte
 - Welche Blöcke werden für die Speicherung einer Datei verwendet?



Kontinuierliche Speicherung

- Datei wird in Blöcken mit aufsteigenden Blocknummern gespeichert
 - Nummer des ersten Blocks und Anzahl der Folgeblöcke muss gespeichert werden, z.B. **Start: Block 4; Länge: 3.**



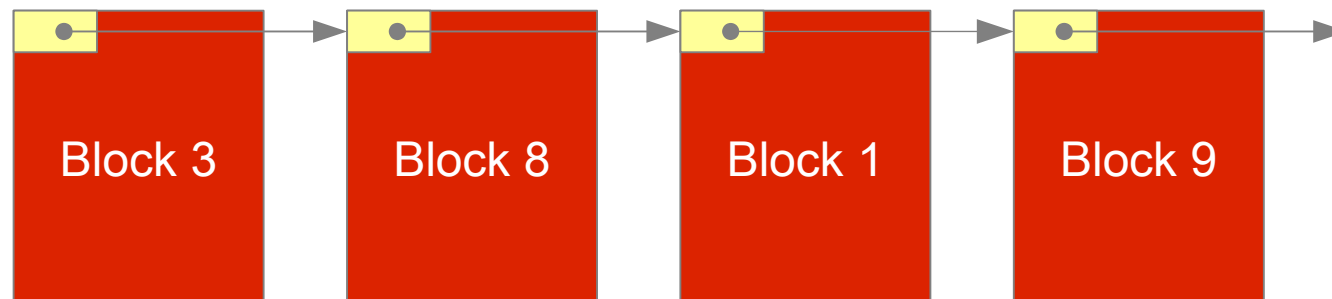
- **Vorteile:**
 - Zugriff auf alle Blöcke mit **minimaler Positionierzeit** des Schwenkarms
 - schneller direkter Zugriff auf **bestimmte Dateiposition**
 - Einsatz z.B. bei nicht modifizierbaren Dateisystemen wie auf CDs/DVDs

Kontinuierliche Speicherung: Probleme

- **Finden des freien Platzes auf der Festplatte**
 - Menge aufeinanderfolgender und freier Plattenblöcke
- **Fragmentierungsproblem**
 - Verschnitt: nicht nutzbare Plattenblöcke; analog zur Speicherverwaltung
- **Größe bei neuen Dateien oft nicht im Voraus bekannt**
- **Erweitern ist problematisch**
- **Umkopieren**, falls kein freier angrenzender Block mehr verfügbar

Verkettete Speicherung

- Blöcke einer Datei sind **verkettet**



- z.B. Commodore-Systeme (CBM 64 etc.)
 - **Blockgröße 256 Bytes**
 - Die ersten zwei Bytes bezeichnen Spur- und Sektornummer des nächsten Blocks;
 - wenn Spurnummer gleich Null → letzter Block.
 - **254 Bytes Nutzdaten**

→ Datei kann **vergrößert und verkleinert** werden

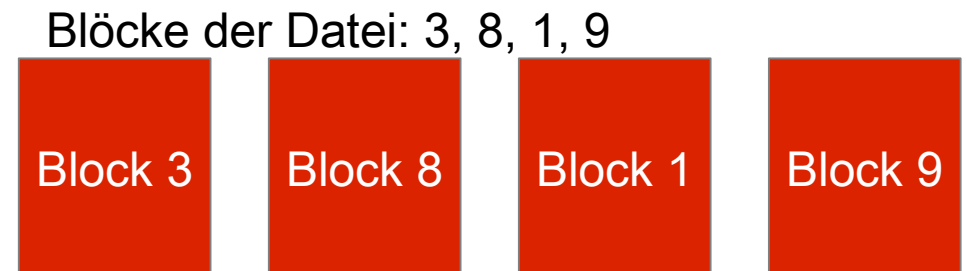
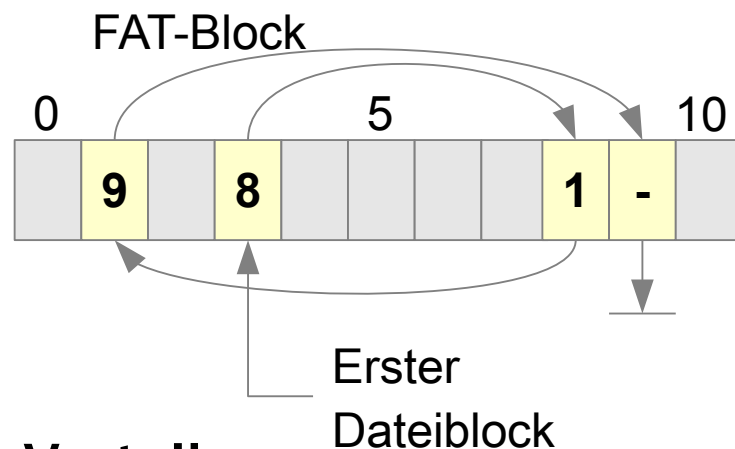
Verkettete Speicherung: Probleme

- **Speicher für Verzeigerung geht von Nutzdaten im Block ab**
 - Ungünstig im Zusammenhang mit **Paging**:
Seite würde immer aus Teilen von **zwei Plattenblöcken** bestehen
- **Fehleranfälligkeit**
 - Datei ist nicht restaurierbar, falls einmal Verzeigerung fehlerhaft
- **Schlechter direkter Zugriff auf bestimmte Dateiposition**
- **Häufiges Positionieren** des Schreib-, Lesekopfs bei verstreuten Datenblöcken

Verkettete Speicherung: FAT

- Verkettung wird in **separaten Plattenblöcken** gespeichert

- FAT-Ansatz (FAT: **File Allocation Table**)
 - z.B. MS-DOS, Windows 95



- **Vorteile:**

- kompletter Inhalt des Datenblocks ist nutzbar
- mehrfache Speicherung der FAT möglich: Einschränkung der Fehleranfälligkeit

Verkettete Speicherung: Probleme (2)

- **Zusätzliches Laden** mindestens eines Blocks
 - Caching der FAT zur Effizienzsteigerung nötig
- **Laden unbenötigter Informationen**
 - FAT enthält Verkettungen für alle Dateien
- **Aufwändige Suche** nach dem zugehörigen Datenblock bei bekannter Position in der Datei
- **Häufiges Positionieren** des Schreib-, Lesekopfs bei verstreuten Datenblöcken

Diskussion: *Chunks/Extents/Clusters*

■ Variation:

- Unterteilen einer Datei in kontinuierlich **gespeicherte Folgen von Blöcken** (*Chunk, Extent* oder *Cluster* genannt)
- Reduziert die Zahl der Positionierungsvorgänge
- Blocksuche wird linear in Abhängigkeit von der Chunk-Größe beschleunigt

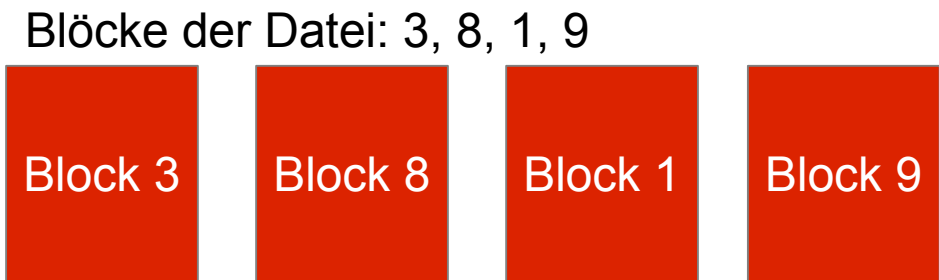
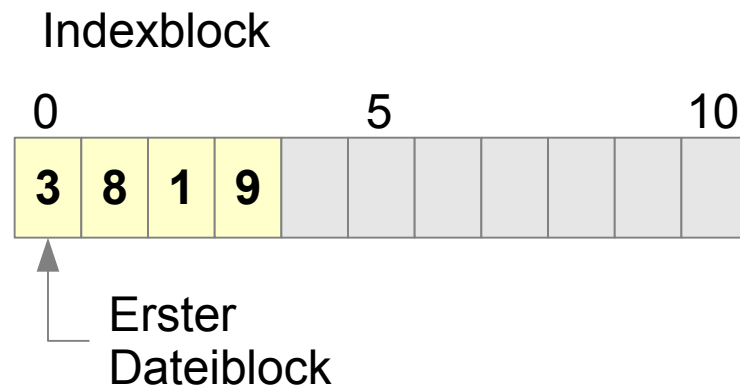
■ Probleme:

- zusätzliche Verwaltungsinformationen
- Verschnitt
 - feste Größe: **innerhalb** einer Folge (interner Verschnitt)
 - variable Größe: **außerhalb** der Folgen (externer Verschnitt)

- Wird eingesetzt, bringt aber keinen fundamentalen Fortschritt.

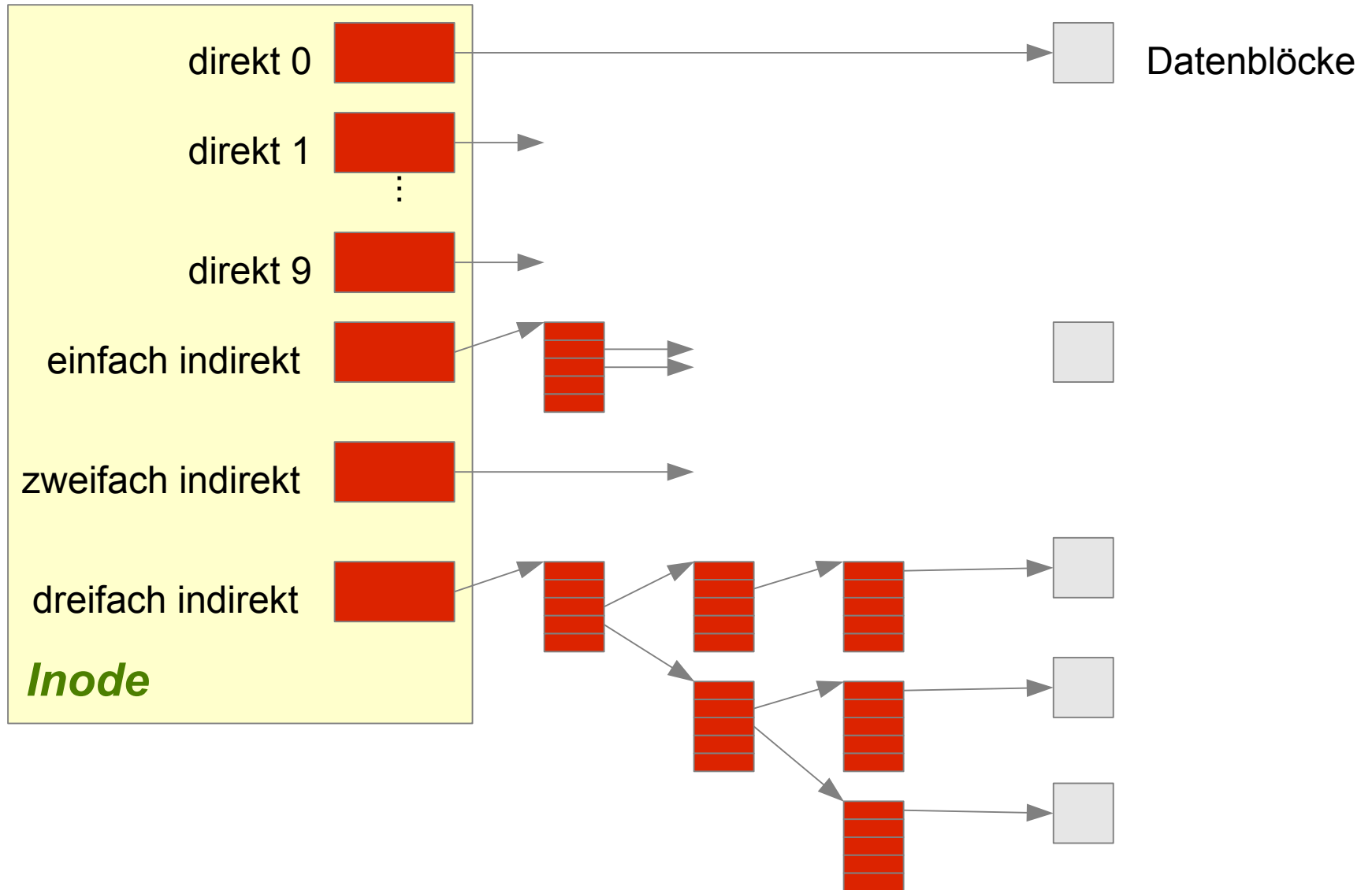
Indiziertes Speichern

- **Spezieller Plattenblock (Indexblock)** enthält Blocknummern der Datenblöcke einer Datei:



- **Problem: Feste Anzahl von Blöcken im Indexblock**
 - Verschnitt bei kleinen Dateien
 - Erweiterung nötig für große Dateien

Indiziertes Speichern: UNIX-Inode

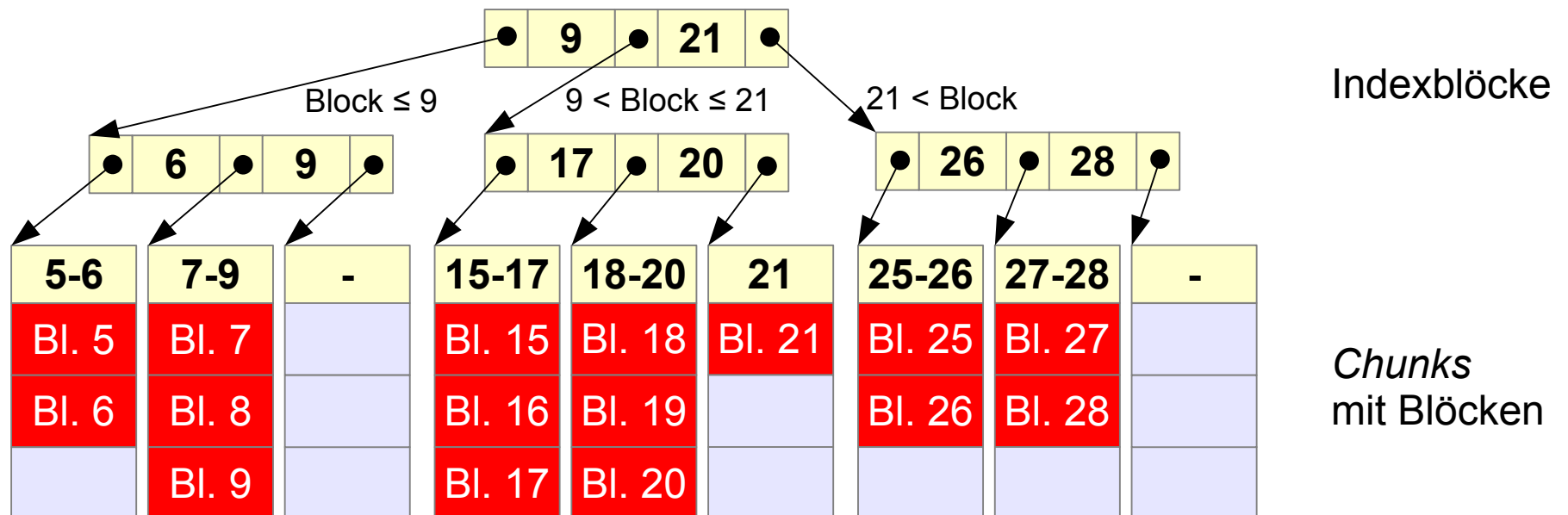


Indiziertes Speichern: Diskussion

- Einsatz von **mehreren Stufen** der Indizierung
 - Inode benötigt sowieso einen Block auf der Platte (Verschnitt unproblematisch bei kleinen Dateien)
 - durch mehrere Stufen der Indizierung auch große Dateien adressierbar
- **Nachteil:**
 - mehrere Blöcke müssen geladen werden (nur bei langen Dateien)

Baumsequentielle Speicherung

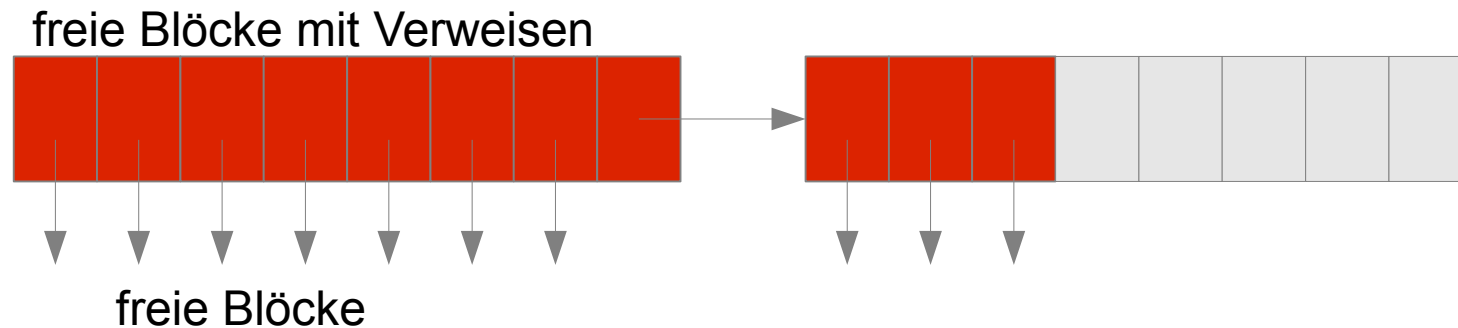
- Wird bei Datenbanken zum effizienten Auffinden eines Datensatzes mit Hilfe eines Schlüssels eingesetzt.
 - Schlüsselraum darf dünn besetzt sein.
- Kann auch verwendet werden, um Datei-*Chunks* mit bestimmtem Datei-*Offset* aufzufinden
 - z.B. NTFS, ReiserFS, Btrfs, IBMs JFS2-Dateisystem (B⁺-Baum)



Inhalt

- Dateien
- Freispeicherverwaltung
- Verzeichnisse
- Dateisysteme
- Pufferspeicher
- Dateisysteme mit Fehlererholung
- Zusammenfassung

Freispeicherverwaltung



Ähnlich wie Verwaltung von freiem Hauptspeicher

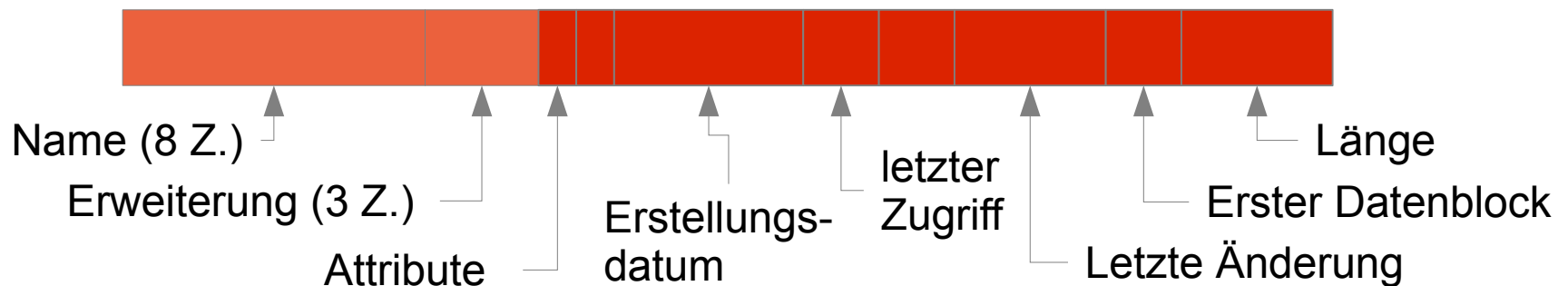
- **Bitvektoren** zeigen für jeden Block Belegung an
- oder **verkettete Listen** repräsentieren freie Blöcke
 - Verkettung kann in den freien Blöcken vorgenommen werden.
 - **Optimierung:** Aufeinanderfolgende Blöcke werden nicht einzeln aufgenommen, sondern am Stück verwaltet.
 - **Optimierung:** Ein freier Block enthält viele Blocknummern weiterer freier Blöcke, und evtl. die Blocknummer eines weiteren Blocks mit den Nummern freier Blöcke.
- **Baumsequentielle Speicherung** freier Blockfolgen
 - Erlaubt schnelle Suche nach freier Blockfolge bestimmter Größe
 - Anwendung z.B. im SGI XFS

Inhalt

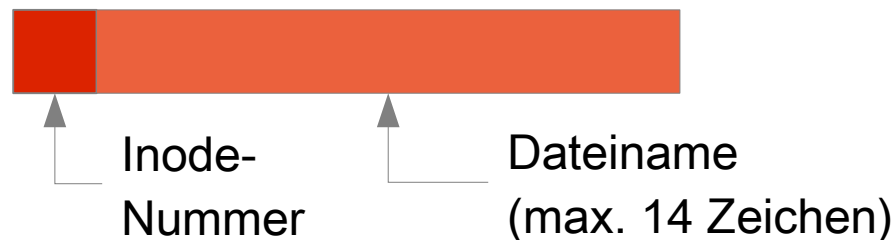
- Dateien
- Freispeicherverwaltung
- Verzeichnisse
- Dateisysteme
- Pufferspeicher
- Dateisysteme mit Fehlererholung
- Zusammenfassung

Verzeichnis als Liste

- Einträge gleicher Länge hintereinander in einer Liste, z.B.
 - *FAT File systems* (VFAT nutzt mehrere Einträge für lange Dateinamen)



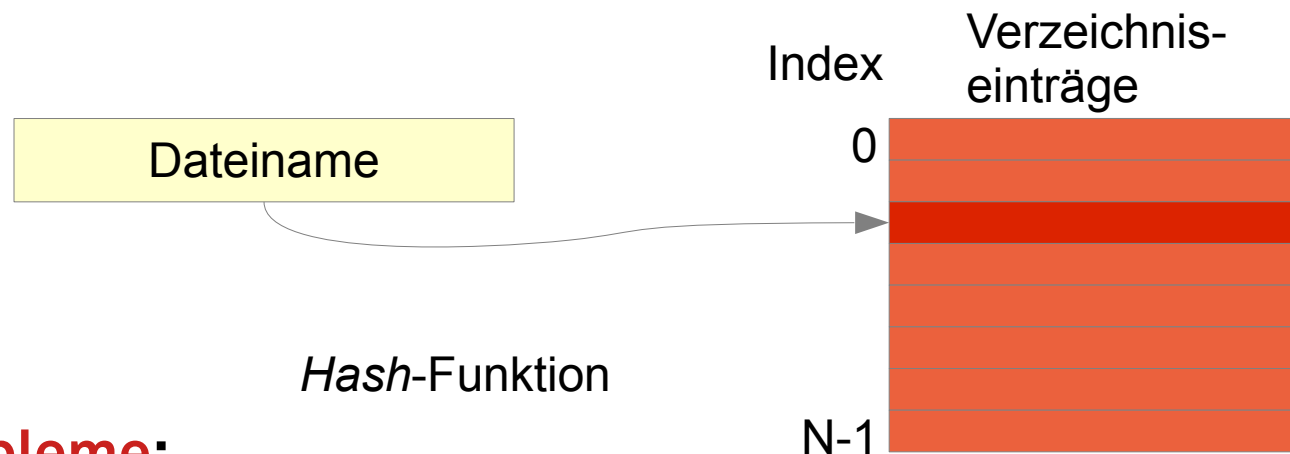
- *UNIX System V.3*



- **Probleme:**
 - Suche nach bestimmtem Eintrag muss linear erfolgen
 - Bei Sortierung der Liste: Schnelles Suchen, Aufwand beim Einfügen

Einsatz von *Hash*-Funktionen

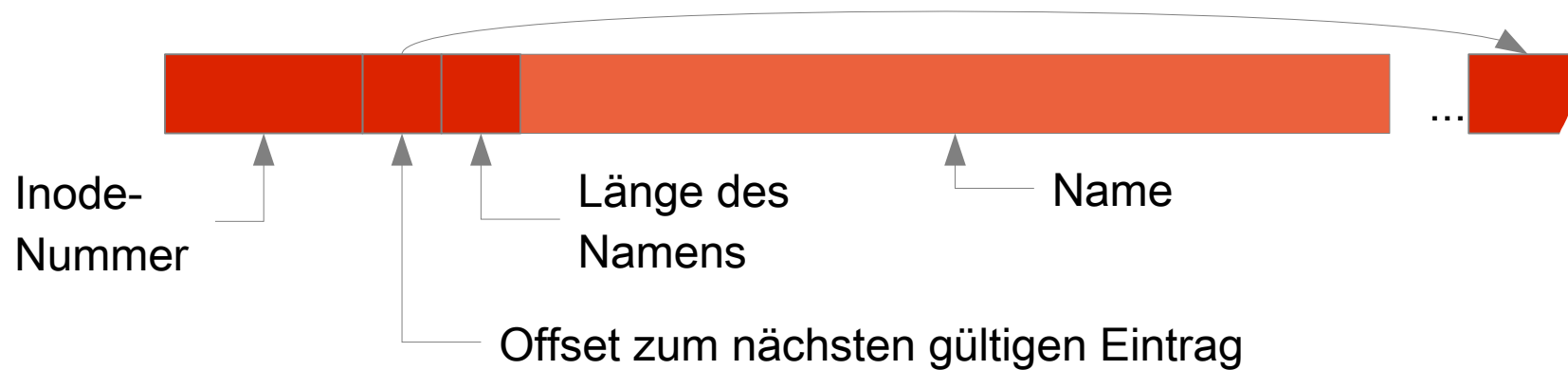
- Funktion bildet Dateinamen auf einen Index in die Katalogliste ab
 - schnellerer Zugriff auf den Eintrag möglich (kein lineares Suchen)
- Einfaches (aber schlechtes) Beispiel: $(\sum \text{Zeichen}) \bmod N$



- **Probleme:**
 - Kollisionen (mehrere Dateinamen werden auf denselben Eintrag abgebildet)
 - Anpassung der Listengröße, wenn Liste voll

Variabel lange Listenelemente

- Beispiel: 4.2 BSD, System V Rel. 4, u.a.



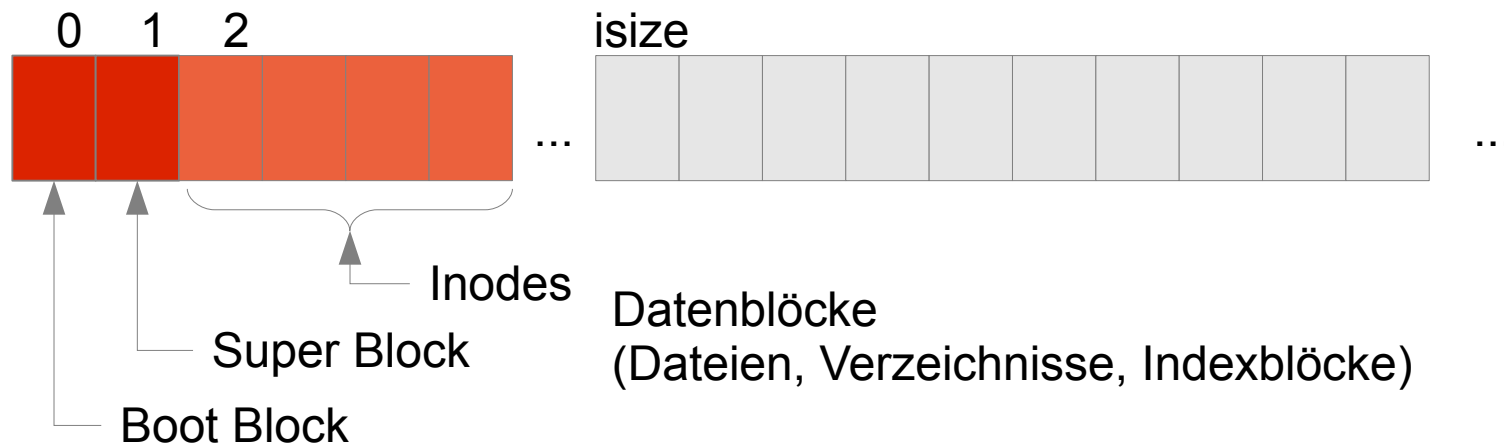
- **Probleme:**
 - Verwaltung von freien Einträgen in der Liste
 - Speicherverschnitt (Kompaktifizieren, etc.)

Inhalt

- Dateien
- Freispeicherverwaltung
- Verzeichnisse
- **Dateisysteme**
- Pufferspeicher
- Dateisysteme mit Fehlererholung
- Zusammenfassung

UNIX System V File System

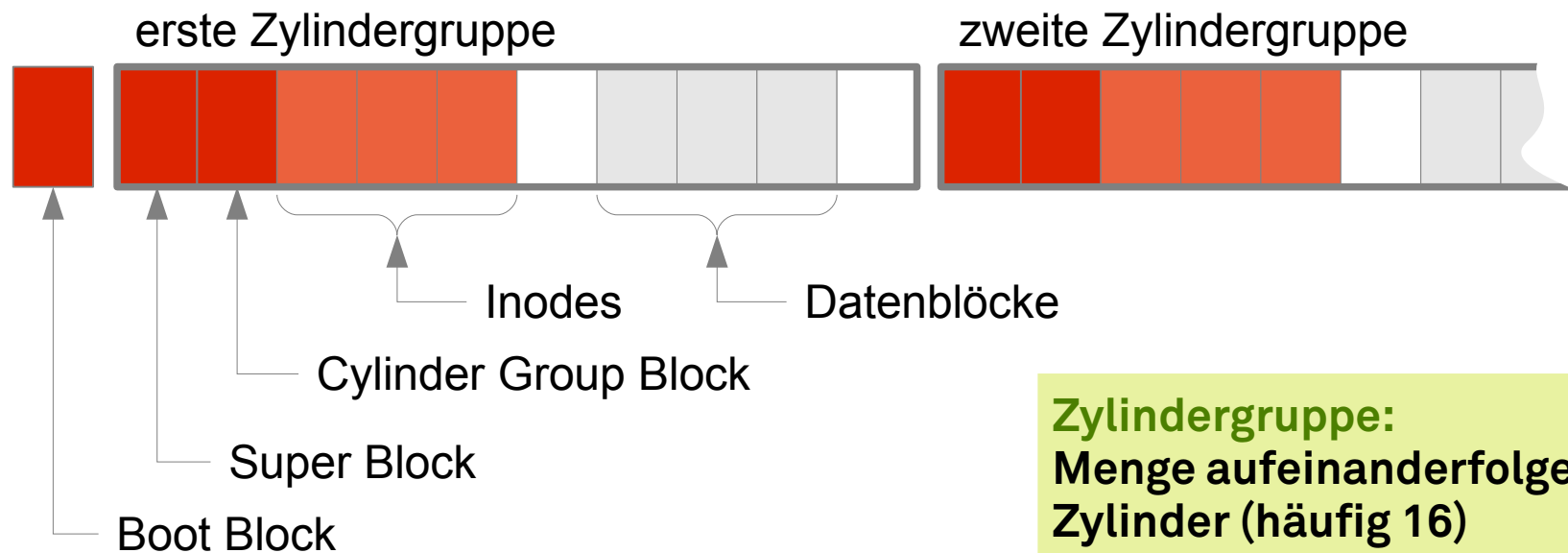
■ Blockorganisation



- **Boot Block** enthält Informationen zum Laden des Betriebssystems
- **Super Block** enthält Verwaltungsinformation für ein Dateisystem
 - Anzahl der Blöcke, Anzahl der Inodes
 - Anzahl und Liste freier Blöcke und freier Inodes
 - Attribute (z.B. Modified flag)

BSD 4.2 (*Berkeley Fast File System*)

■ Blockorganisation

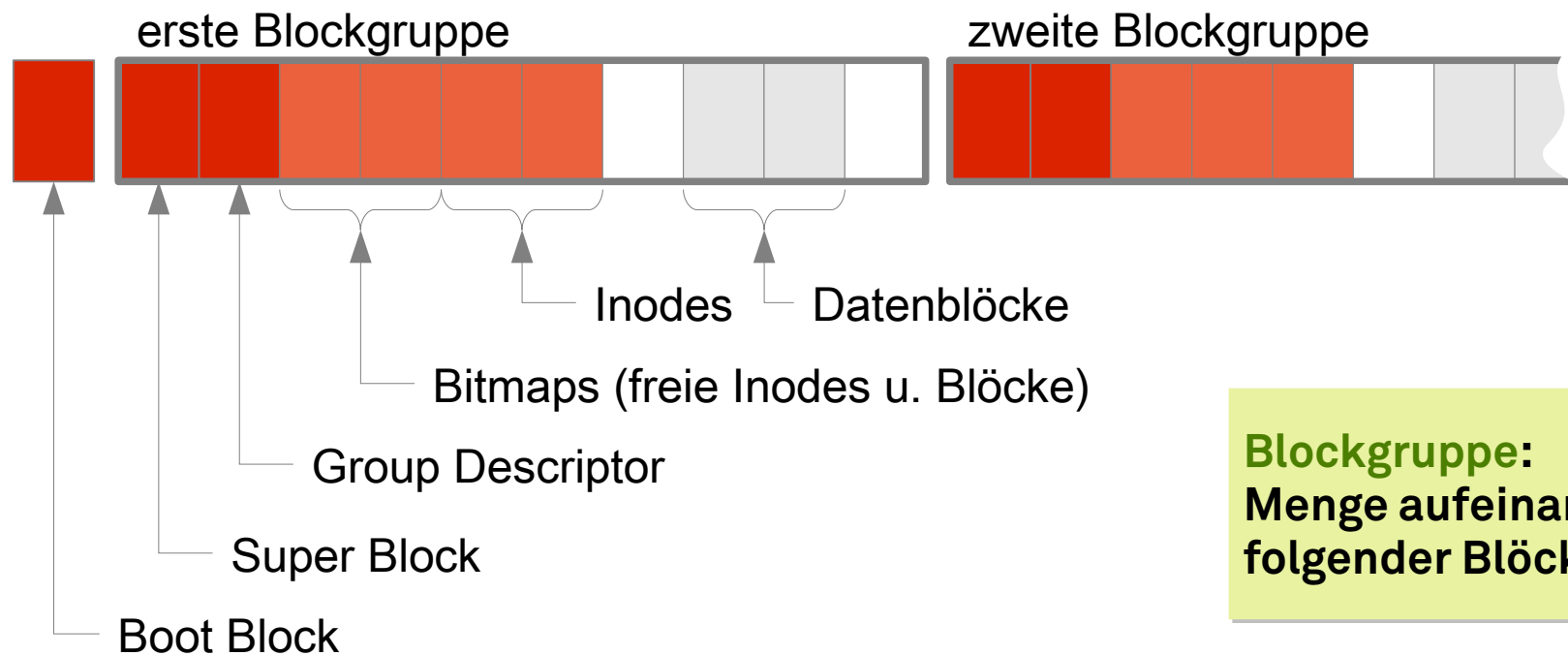


- **Kopie des Super Blocks** in jeder Zylindergruppe
- Eine Datei wird möglichst innerhalb einer Zylindergruppe gespeichert.
- Verzeichnisse werden verteilt, Dateien eines V. bleiben zusammen

■ **Vorteil:** kürzere Positionierungszeiten

Linux ext2/3/4 File System

■ Blockorganisation



Blockgruppe:
Menge aufeinander folgender Blöcke

- Ähnliches Layout wie BSD Fast File System
- Blockgruppen unabhängig von Zylindern

Inhalt

- Dateien
- Freispeicherverwaltung
- Verzeichnisse
- Dateisysteme
- Pufferspeicher
- Dateisysteme mit Fehlererholung
- Zusammenfassung

UNIX Block Buffer Cache

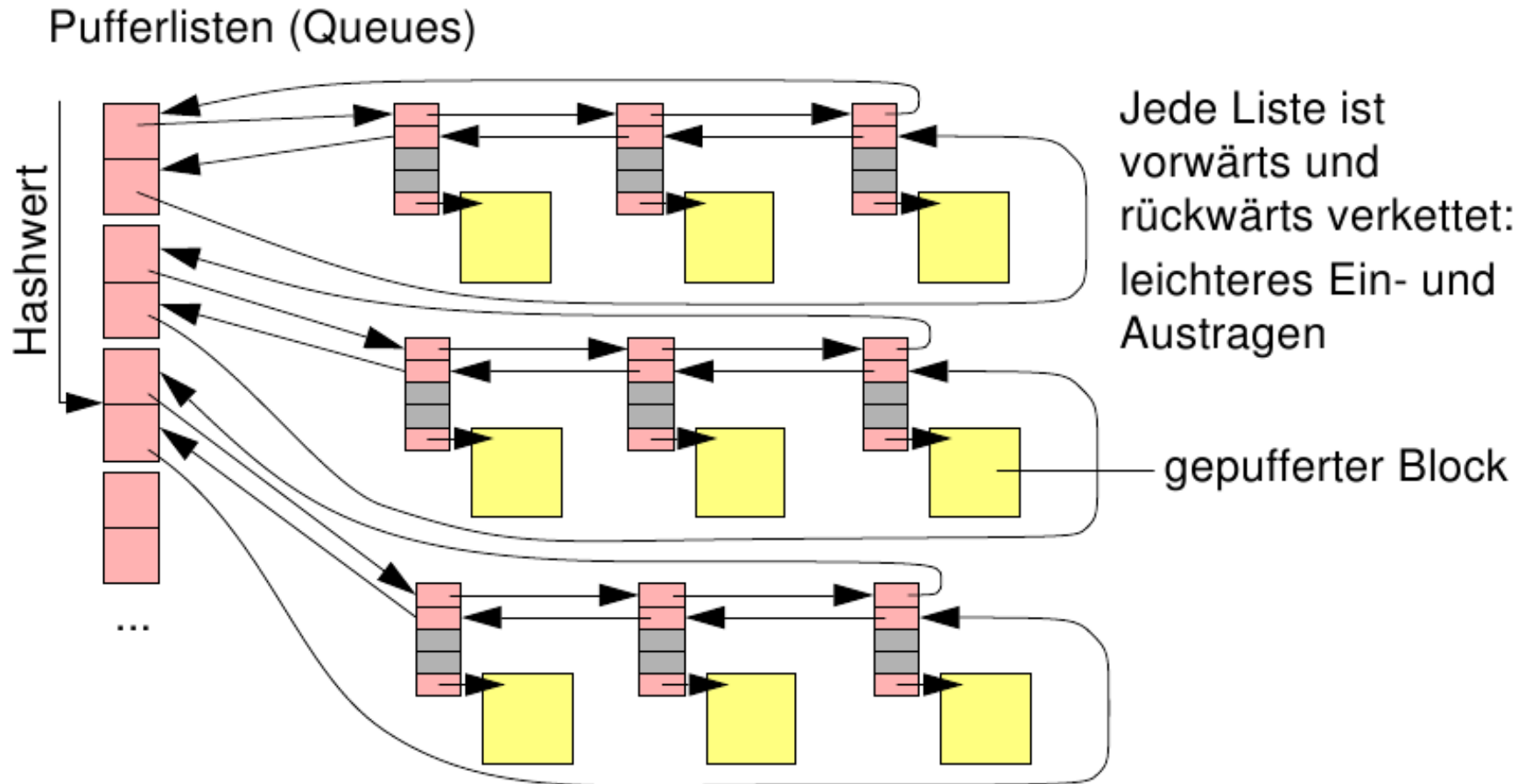
- Pufferspeicher für Plattenblöcke im Hauptspeicher
 - Verwaltung mit Algorithmen ähnlich wie bei Seitenverwaltung (Speicher)
 - **Read ahead**: beim sequentiellen Lesen wird auch der Transfer von Folgeblöcken angestoßen
 - **Lazy write**: Block wird nicht sofort auf Platte geschrieben (erlaubt Optimierung der Schreibzugriffe und blockiert den Schreiber nicht)
 - Verwaltung freier Blöcke in einer **Freiliste**:
 - Kandidaten für Freiliste werden nach LRU-Verfahren bestimmt
 - Bereits freie, aber noch nicht anderweitig benutzte Blöcke können reaktiviert werden (Reclaim)

UNIX Block Buffer Cache (2)

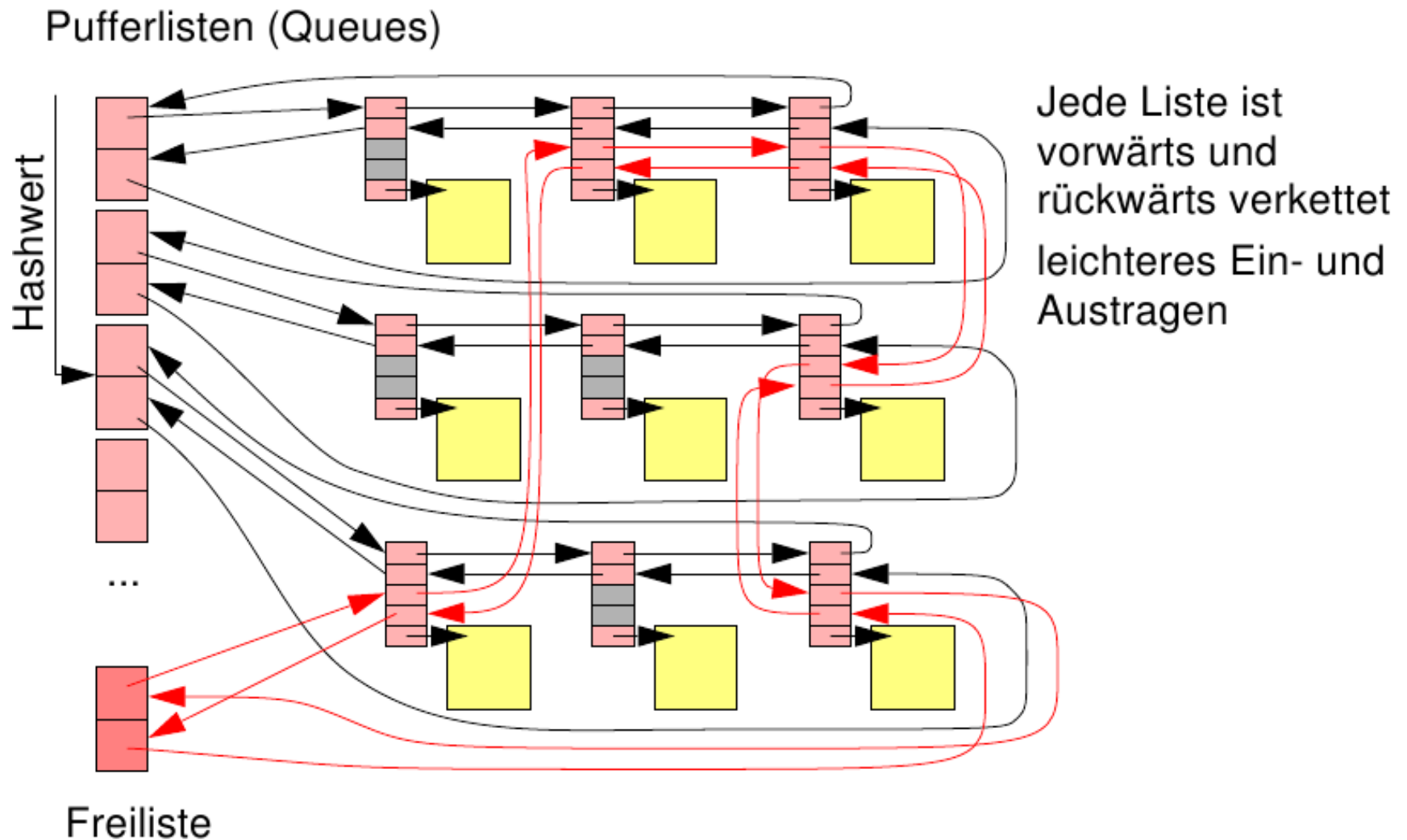
- Schreiben erfolgt, wenn ...
 - keine freien Puffer mehr vorhanden sind,
 - regelmäßig vom System (**fsflush**-Prozess, **update**-Prozess),
 - beim Systemaufruf **sync ()**,
 - und nach jedem Schreibaufwurf im Modus **O_SYNC** (siehe **open (2)**).

- Adressierung:
 - Adressierung eines Blocks erfolgt über ein Tupel:
(*Gerätenummer, Blocknummer*)
 - Über die Adresse wird ein Hash-Wert gebildet, der eine der möglichen Pufferlisten auswählt.

UNIX Block Buffer Cache: Aufbau



UNIX Block Buffer Cache: Aufbau (2)



Inhalt

- Dateien
- Freispeicherverwaltung
- Verzeichnisse
- Dateisysteme
- Pufferspeicher
- **Dateisysteme mit Fehlererholung**
- Zusammenfassung

Dateisysteme mit Fehlererholung

■ Mögliche Fehler:

- Stromausfall (ahnungsloser Benutzer schaltet einfach Rechner aus)
- Systemabsturz

■ Auswirkungen auf das Dateisystem: inkonsistente Metadaten

- z.B. Katalogeintrag fehlt zur Datei oder umgekehrt
- z.B. Block ist benutzt, aber nicht als belegt markiert

■ Reparaturprogramme

- Programme wie **chkdsk**, **scandisk** oder **fsck** können inkonsistente Metadaten reparieren

■ Probleme:

- Datenverluste bei Reparatur möglich
- lange Laufzeiten der Reparaturprogramme bei großen Platten

Journalled File Systems

- **Zusätzlich zum Schreiben der Daten und Meta-Daten (z.B. Inodes) wird ein Protokoll der Änderungen geführt**
 - Alle Änderungen treten als Teil von Transaktionen auf.
 - Beispiele für **Transaktionen**:
 - Erzeugen, Löschen, Erweitern, Verkürzen von Dateien
 - Verändern von Dateiattributen
 - Umbenennen einer Datei
 - Protokollieren aller Änderungen am Dateisystem zusätzlich in einer Protokolldatei (**Log File**)

- **Bootvorgang**
 - Abgleich der Protokolldatei mit den aktuellen Änderungen
→ Vermeidung von Inkonsistenzen

Journalled File Systems: Protokoll

- Für jeden Einzelvorgang einer Transaktion wird zunächst ein Protokolleintrag erzeugt und ...
- danach die Änderung am Dateisystem vorgenommen.
- Dabei gilt:
 - Der Protokolleintrag wird immer **vor** der eigentlichen Änderung auf Platte geschrieben.
 - Wurde etwas auf Platte geändert, steht auch der Protokolleintrag dazu auf der Platte.

Journalled File Systems: Erholung

- **Beim Bootvorgang wird überprüft**, ob die protokollierten Änderungen vorhanden sind:
 - Transaktion kann wiederholt bzw. abgeschlossen werden, falls alle Protokolleinträge vorhanden → **Redo**
 - Angefangene, aber nicht beendete Transaktionen werden rückgängig gemacht → **Undo**

Journalled File Systems: Ergebnis

■ Vorteile:

- eine Transaktion ist entweder vollständig durchgeführt oder gar nicht
- Benutzer kann ebenfalls Transaktionen über mehrere Dateizugriffe definieren, wenn diese ebenfalls im Log erfasst werden
- keine inkonsistenten Metadaten möglich
- Hochfahren eines abgestürzten Systems benötigt nur den relativ kurzen Durchgang durch das Log-File
 - Die Alternative **chkdsk** benötigt viel Zeit bei großen Platten.

■ Nachteile:

- ineffizienter, da zusätzliches Log-File geschrieben wird
 - daher meist nur *Metadata Journaling*, kein *Full Journaling*

■ Beispiele: NTFS, ext3/ext4, ReiserFS

Inhalt

- Dateien
- Freispeicherverwaltung
- Verzeichnisse
- Dateisysteme
- Pufferspeicher
- Dateisysteme mit Fehlererholung
- Zusammenfassung

Zusammenfassung: Dateisysteme

- ... sind eine **Betriebssystemabstraktion**
 - Speicherung logisch zusammenhängender Informationen als Datei
 - Meist hierarchische Verzeichnisstruktur, um Dateien zu ordnen

- ... werden durch die **Hardware beeinflusst**
 - Minimierung der Positionierungszeiten bei Platten
 - Gleichmäßige „Abnutzung“ bei FLASH-Speicher
 - Kein *Buffer-Cache* bei RAM-Disks

- ... werden durch das **Anwendungsprofil beeinflusst**
 - Blockgröße
 - zu klein → Verwaltungsstrukturen können zu *Performance*-Verlust führen
 - zu groß → Verschnitt führt zu Plattenplatzverschwendung
 - Aufbau von Verzeichnissen
 - keine *Hash*-Funktion → langwierige Suche
 - mit *Hash*-Funktion → mehr Aufwand bei der Verwaltung