TECHNISCHE UNIVERSITÄT DRESDEN

Fakultät Informatik Institut für Systemarchitektur, Professur für Betriebssysteme

BETRIEBSSYSTEME UND SICHERHEIT

mit Material von Olaf Spinczyk,
Universität Osnabrück

Moderne Dateisysteme

https://tud.de/inf/os/studium/vorlesungen/bs

HORST SCHIRMEIER

Inhalt

- Wiederholung
- Herausforderungen
  - Zuverlässigkeit, Leistungsoptimierung, flexiblere Datenträgerverwaltung
- Intelligente Blockgeräte
  - *Logical Volume Management*, *Block Buffer Cache*, RAID
- Konzepte moderner Dateisysteme
  - Journale, *Log*-basierte Dateisysteme, *Copy-on-Write*-Prinzip
- Beispiel: Btrfs
- Zusammenfassung

Inhalt

- **Wiederholung**
- Herausforderungen
  - Zuverlässigkeit, Leistungsoptimierung, flexiblere Datenträgerverwaltung
- Intelligente Blockgeräte
  - *Logical Volume Management*, *Block Buffer Cache*, RAID
- Konzepte moderner Dateisysteme
  - Journale, *Log*-basierte Dateisysteme, *Copy-on-Write*-Prinzip
- Beispiel: Btrfs
- Zusammenfassung

Wiederholung

- Festplatten und *Solid State Disks* (SSDs) haben eine Blockstruktur und wahlfreien Zugriff
  - **Lokalität der Zugriffe** ist entscheidend für die Leistung (insbesondere **Spurwechsel** kosten viel Zeit – mehrere ms)
  - SSDs: keine mechanischen Verzögerungen, Lokalität dennoch wichtig
- Dateisysteme bieten **Abstraktionen** zum Umgang von Programmen mit **persistenten Daten**
  - **Dateien** und Hierarchien von **Verzeichnissen**
  - **Metadaten**, zum Beispiel Name, Größe, Erstellungsdatum, …
- Es gibt unterschiedlichste Möglichkeiten beides aufeinander abzubilden.

Inhalt

- Wiederholung
- **Herausforderungen**
  - Zuverlässigkeit, Leistungsoptimierung, flexiblere Datenträgerverwaltung
- Intelligente Blockgeräte
  - *Logical Volume Management*, *Block Buffer Cache*, RAID
- Konzepte moderner Dateisysteme
  - Journale, *Log*-basierte Dateisysteme, *Copy-on-Write*-Prinzip
- Beispiel: Btrfs
- Zusammenfassung

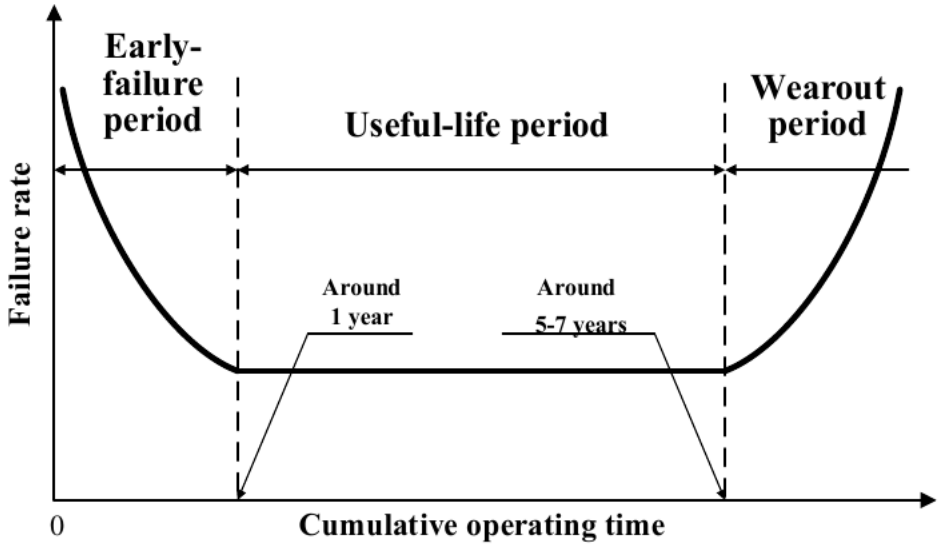
Herausforderung: Zuverlässigkeit (1)

Probleme:

- Defekte Platten oder Blöcke;
- Rechnerabsturz oder -ausfall

Auswirkungen:

- kompletter Datenverlust,
- kaputte Datenblöcke, z.B.
  - Anwendung kann Datei nicht mehr interpretieren
- inkonsistente Metadaten, z.B.
  - Katalogeintrag fehlt zur Datei oder umgekehrt
  - Block ist benutzt, aber nicht als belegt markiert



Die „Badewannenkurve“ zeigt, wie sich die Fehlerrate von Festplatten (und den meisten anderen technischen Produkten) über ihre Lebenszeit hinweg typischerweise entwickelt.

Herausforderung: Zuverlässigkeit (2)

- **Lösungsansatz: *Backup***
  - Regelmäßige inkrementelle und vollständige Sicherung der Daten
  - Probleme: Zeit- und Speicherplatzbedarf
- **Lösungsansatz: Prüfsummen**
  - Daten könnten mit einer Prüfsumme (Detektion) oder einem fehlerkorrigierenden Code (Reparatur) versehen werden
  - Probleme: Speicherplatzbedarf; Verantwortlichkeiten (Ebene)
- **Lösungsansatz: Reparaturprogramme**
  - Programme wie **chkdsk**, **scandisk** oder **fsck** können inkonsistente Metadaten reparieren
  - Probleme: Datenverluste bei Reparatur möglich; lange Laufzeiten der Reparaturprogramme bei großen Platten

Herausforderung: Leistungsoptimierung

Problem:

- Festplatten haben geringe Schreib-/Leseraten und eine **hohe Positionierungslatenz**
- CPU-/Hauptspeicherleistung und Plattenleistung divergieren

Beispiel: Toshiba X300
- Kapazität 4-16 TB
- Ø Latenz **4,17 ms**
- MTTF 600.000 Std.
- Benchmarks (6 TB):
  - Seq. 130 MB/s
  - Rand. **2,25 MB/s**

Auswirkungen:

- Bei E/A-intensiven Anwendungen, z.B. Datenbanken, und Vorgängen, z.B. *Booten* oder Programmstart, wird die Platte zum **Flaschenhals**.

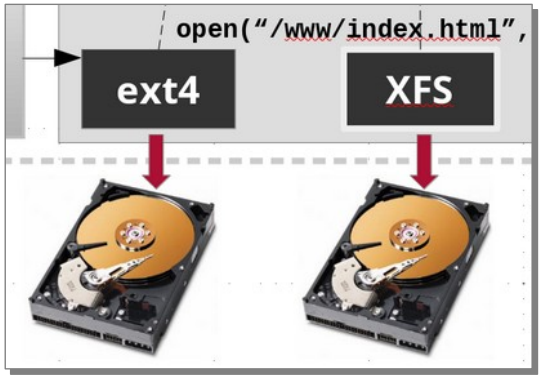
Lösungsansatz: *Cache*

- Wichtige (Meta-)Daten im Hauptspeicher vorhalten
  - Problem: **Konsistenz** zwischen *Cache* und Platte

Herausforderung: Datenträgerverwaltung

Problem:

- Physikalische Laufwerksdimensionen beschränken die Größe der Dateisysteme.
  - ***Platte voll, was nun?***



1:1-Beziehung zwischen Dateisystem und Platte

Auswirkung:

- Plattenkapazität wird überdimensioniert, um aufwändiges Umkopieren zu vermeiden

Lösungsansatz: Virtuelles Dateisystem

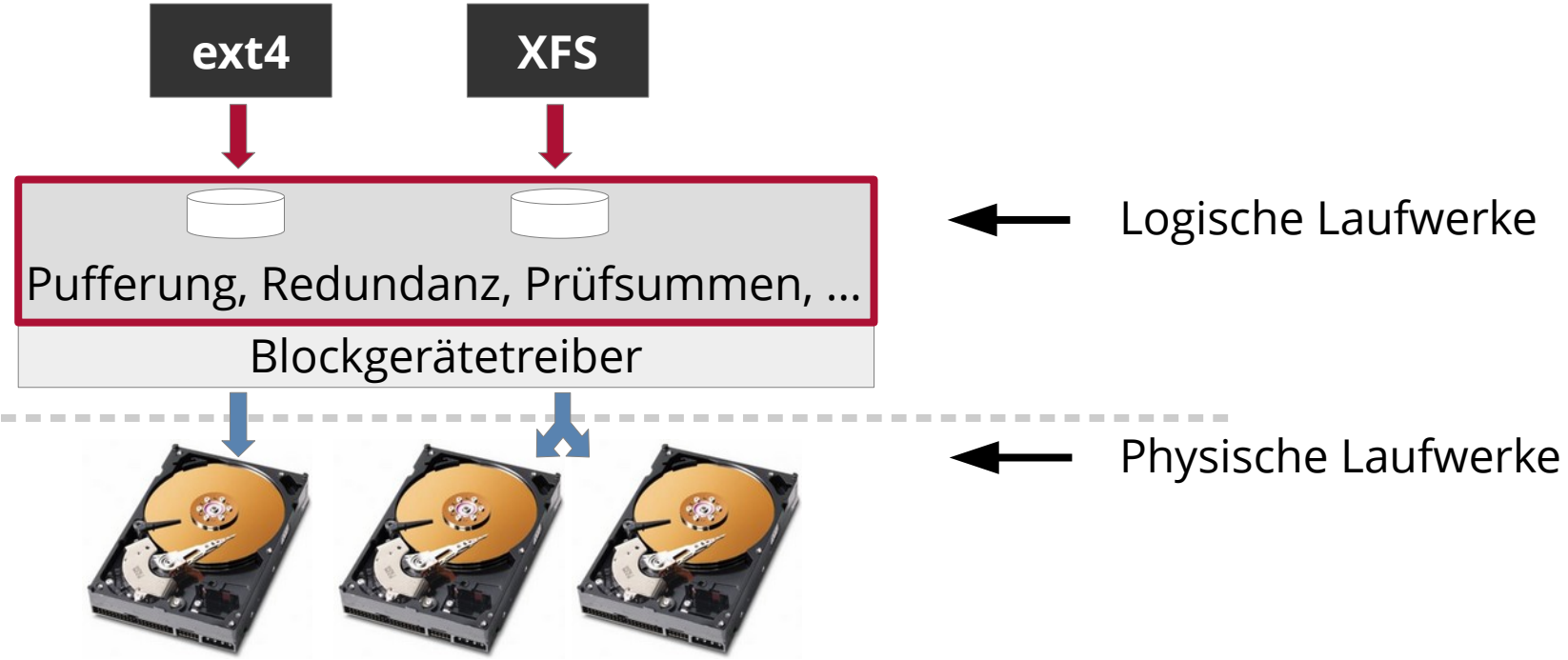
- Neue Platten als Verzeichnisse (ggf. mit ***„Soft Links“***) einhängen
  - Probleme: Nicht transparent für Benutzer/Anwendungen; Größenbeschränkung für bestehende Verzeichnisse bleibt erhalten

Inhalt

- Wiederholung
- Herausforderungen
  - Zuverlässigkeit, Leistungsoptimierung, flexiblere Datenträgerverwaltung
- **Intelligente Blockgeräte**
  - *Logical Volume Management*, *Block Buffer Cache*, RAID
- Konzepte moderner Dateisysteme
  - Journale, *Log*-basierte Dateisysteme, *Copy-on-Write*-Prinzip
- Beispiel: Btrfs
- Zusammenfassung

Intelligente Blockgeräte(-treiber)

- **Ansatz:** Herausforderungen unterhalb der Ebene der Dateisysteme angehen
- **Vorteil:** Alle Dateisystemimplementierungen profitieren!



UNIX Block Buffer Cache

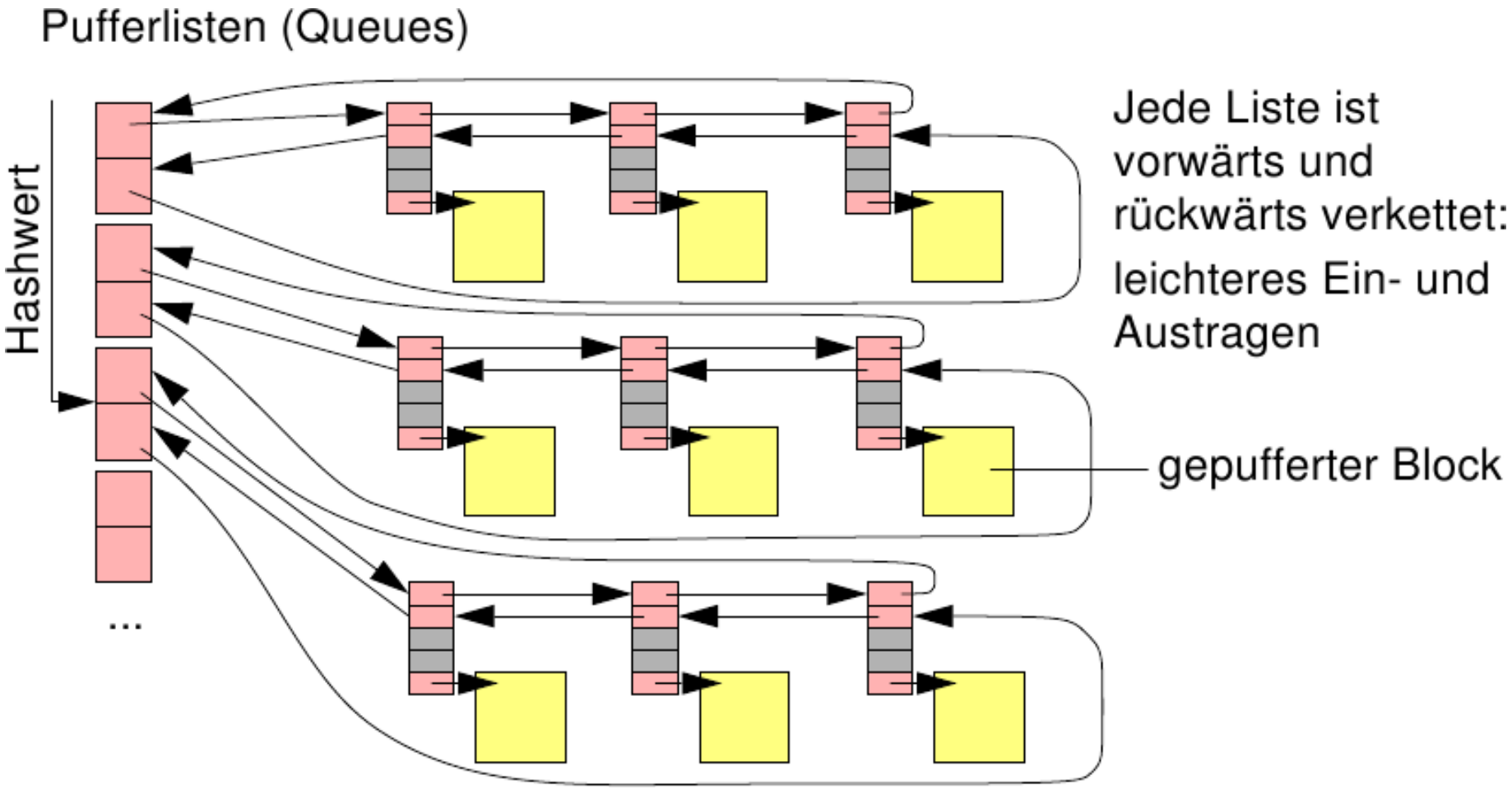
- Pufferspeicher für Plattenblöcke im Hauptspeicher
  - Verwaltung mit Algorithmen ähnlich wie bei Kachelverwaltung
  - ***Read ahead:*** beim sequentiellen Lesen wird auch der Transfer von **Folgeblöcken** angestoßen
  - ***Lazy write:*** Block wird **nicht sofort** auf Platte geschrieben (erlaubt Optimierung 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)

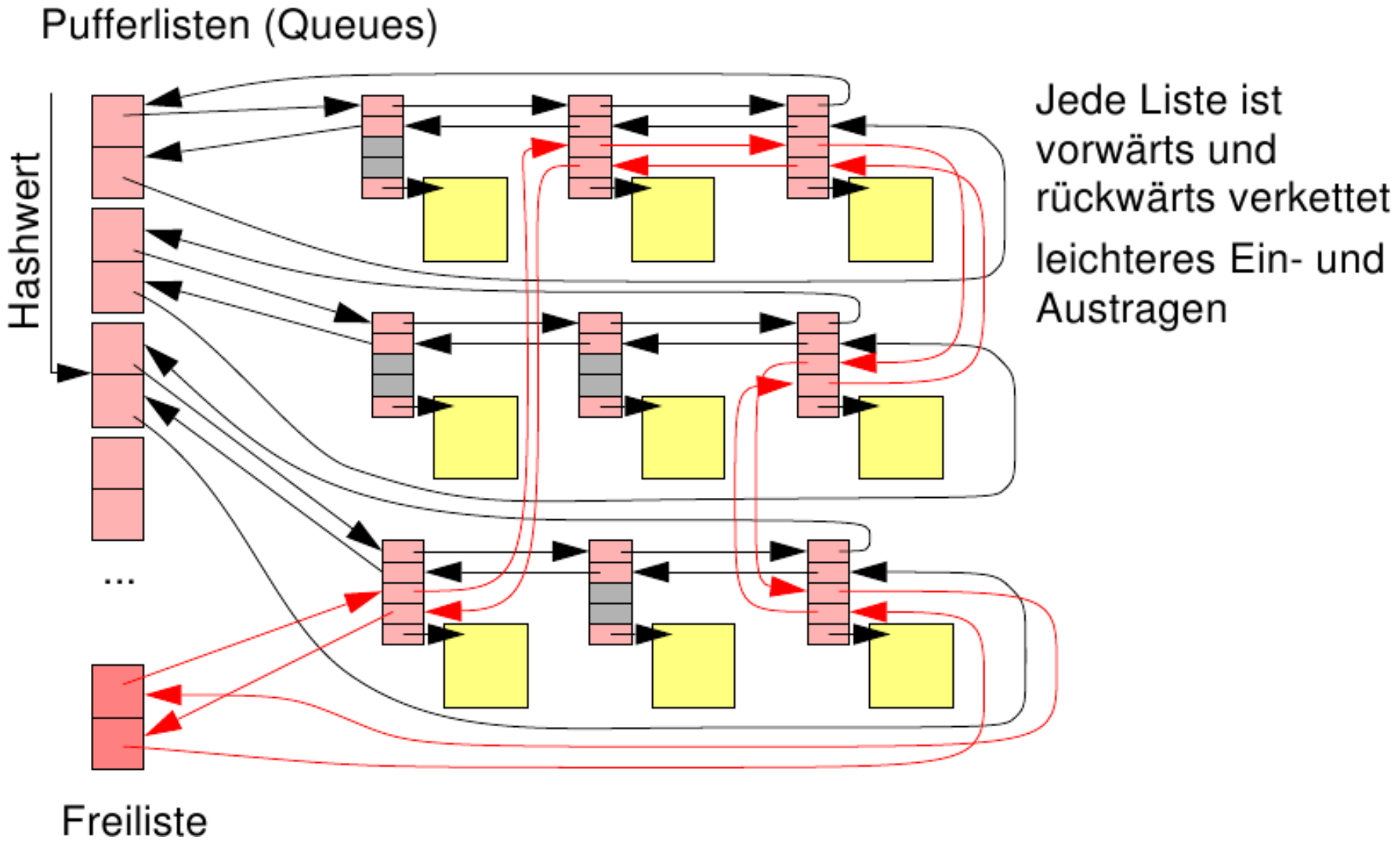
- Schreiben erfolgt, wenn
  - keine freien Puffer mehr vorhanden sind,
  - regelmäßig vom System (**fsflush**-Prozess, **update**-Prozess),
  - beim Systemaufruf **sync**(),
  - und nach jedem Schreibaufruf im Modus O_SYNC.

- 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

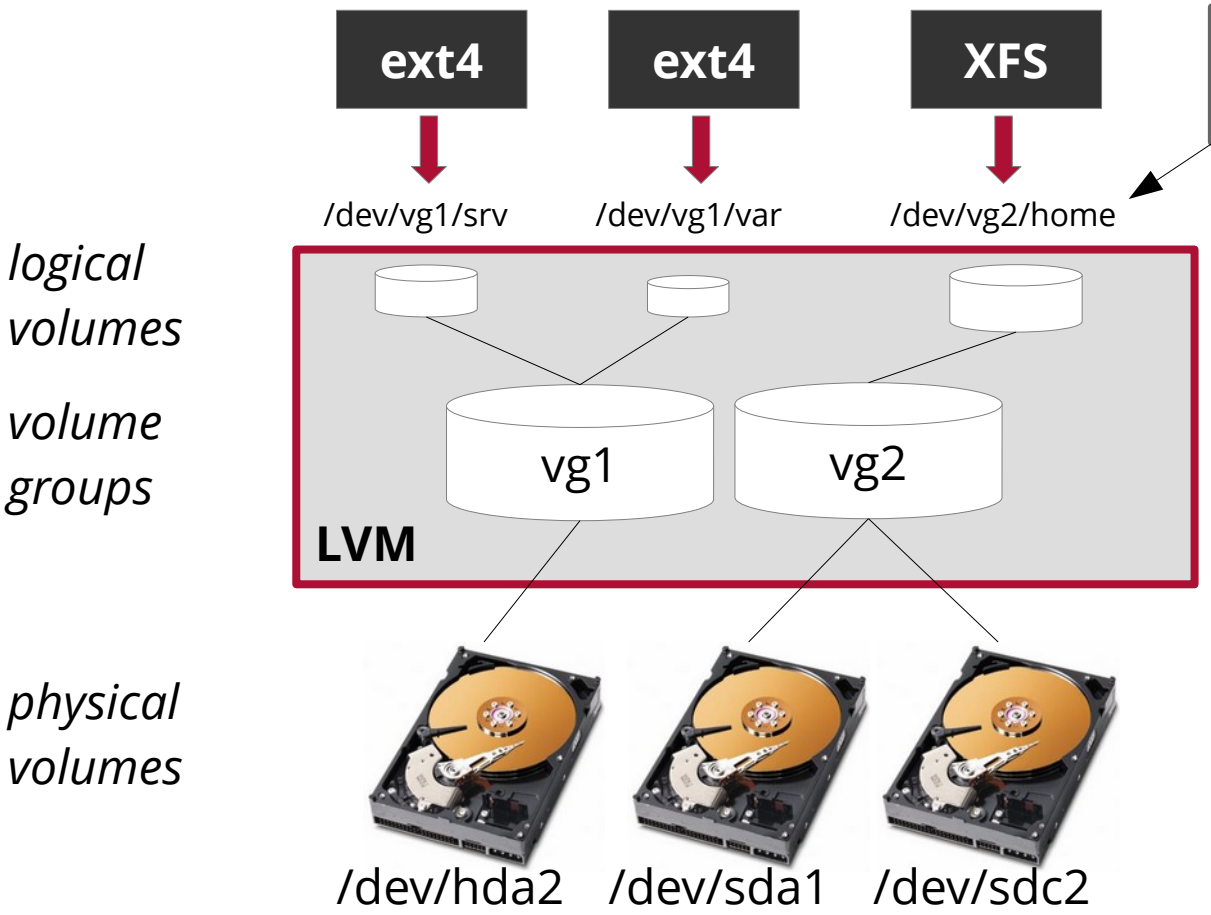


UNIX Block Buffer Cache: Aufbau (2)



[Linux] *Logical Volume Management*

- 1:1-Beziehung Dateisystem/Platte wird aufgebrochen




```
> pvcreate /dev/sda1 /dev/sdc2
> vgcreate vg2 /dev/sda1 /dev/sdc2
> lvcreate -L10G -name home vg2
```

Logical Volumes können beliebig vergrößert, verkleinert und verschoben werden (sofern das Dateisystem mitspielt).

Redundant Arrays of Inexpensive Disks

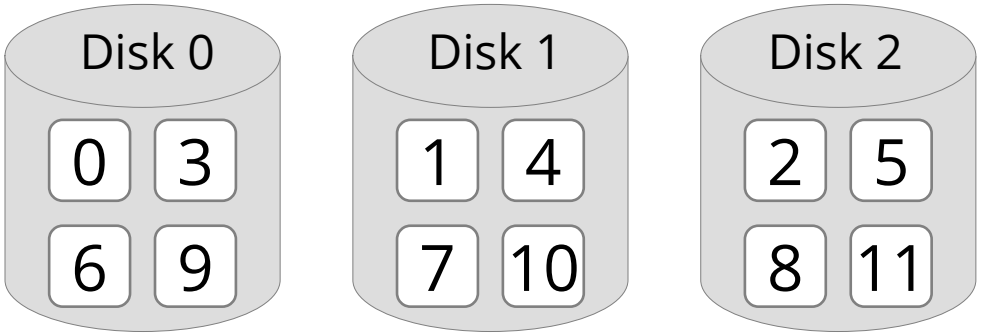
(kurz: RAID)

- Ursprüngliche Idee: **Kosten sparen**, indem man große logische Laufwerke aus kosteneffizienten kleineren Laufwerken realisiert (Preis pro GiB)
- Zusätzliche Features:
  - Bessere Ausnutzung des verfügbaren **Datendurchsatzes** durch parallele Transfers
  - Fehlertoleranz mit Hilfe von **Redundanz**
- Zwei Varianten:
  - **Hardware-RAID:** Plattencontroller mit spezieller Management-Software
  - **Software-RAID:** Schicht zwischen Plattentreiber und Dateisystemcode

RAID 0: Gestreifte Platten

(engl. ***striping***)

- **Ansatz:** Daten eines großen logischen Laufwerks werden reihum über *N* physische Platten verteilt gespeichert:



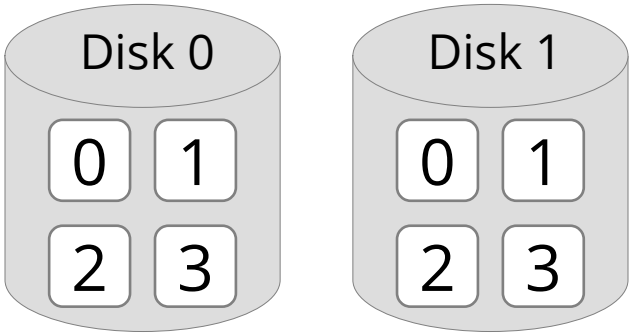

```
pdsk = lblk % N;
pblk = lblk / N;
```

- **Effekt:** Höherer Durchsatz, da mehrere Platten parallel angesprochen werden
- **Nachteil:** Ausfallwahrscheinlichkeit erhöht sich

RAID 1: Gespiegelte Platten

(engl. ***mirroring***)

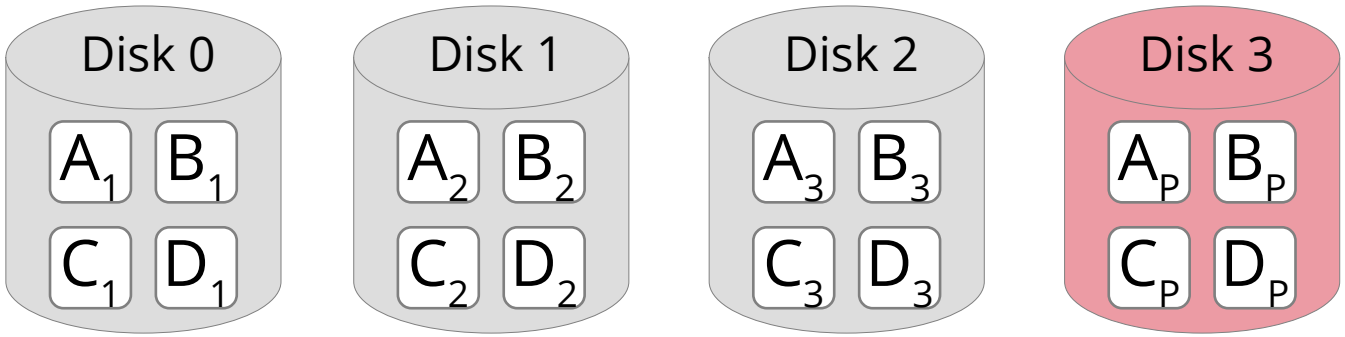
- **Ansatz:** Daten werden auf zwei (oder mehr) Platten gleichzeitig gespeichert gespeichert (redundant):



- **Effekt:** Höherer Durchsatz beim Lesen, etwas langsameres Schreiben und erhöhte Ausfallsicherheit durch Kopie(n)
- **Nachteil:** N-facher Speicherplatzbedarf

RAID 4: Zusätzliche Paritätsplatte

- **Ansatz:** Daten werden über mehrere Platten verteilt (*striping*), eine Platte enthält zugehörige Parität

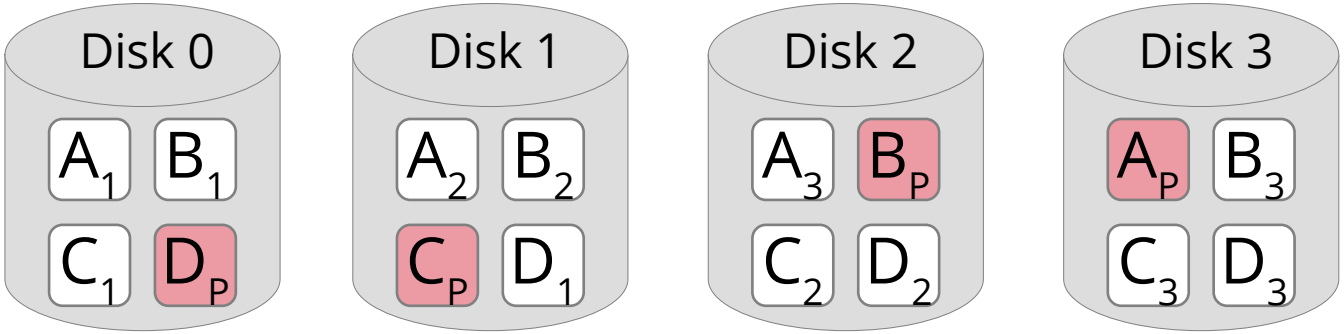


Paritätsblock enthält bitweise XOR-Verknüpfung der zugehörigen Blöcke aus den anderen Streifen.

- **Effekt:** Fehler (einer Platte) können erkannt und behoben werden ohne großen Mehrbedarf an Speicherplatz. Schnelles Lesen.
- **Nachteil:** Paritätsplatte wird beim Schreiben zum Flaschenhals.

RAID 5 und 6: Verstreute Paritätsdaten

- **Ansatz:** Paritätsblock wird über alle Platten verteilt

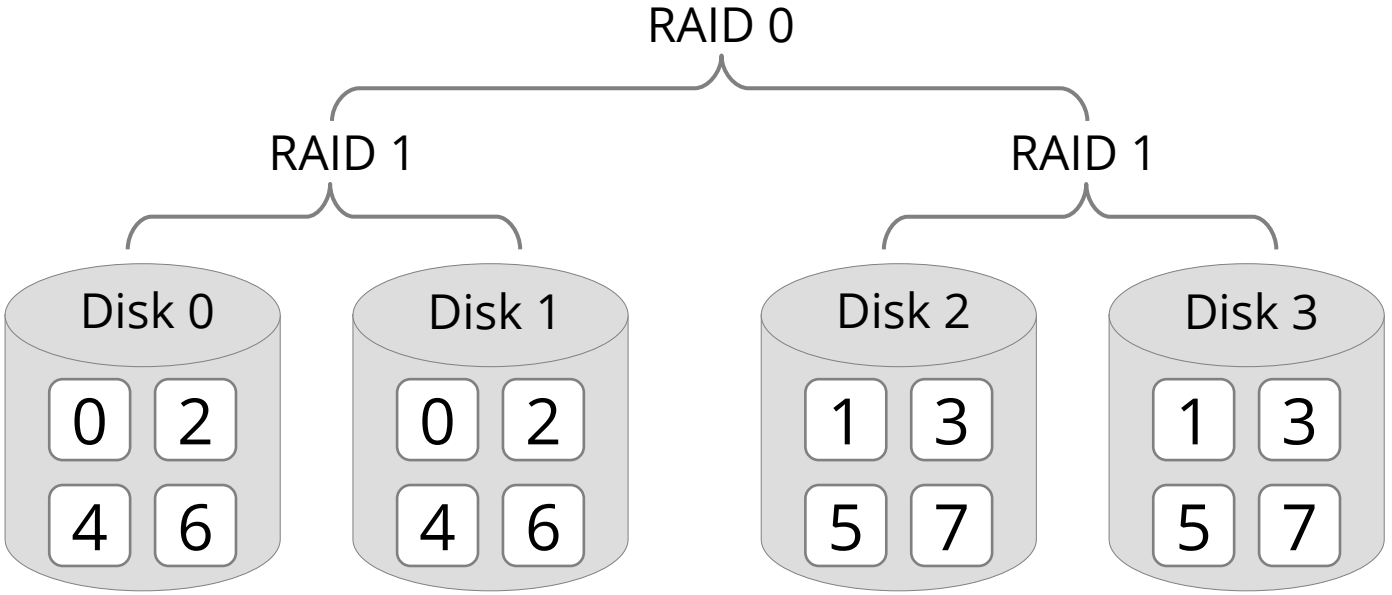


RAID 5 ist heute die üblichste RAID-Variante.

- **Effekt:** Zusätzliche Belastung beim Schreiben durch Aktualisierung des Paritätsblock wird verteilt
  - Speziell **RAID 6**: Durch einen weiteren Paritätsblock kann der Ausfall von bis zu zwei Platten verkraftet werden
- **Nachteil:** Alle Daten werden aufwändig geschützt, auch wenn ein Teil davon unkritisch ist.

RAID *x*+*y* (= RAID *xy*): Hierarchien

- **Ansatz:** Unterschiedliche RAID-Mechanismen werden geschachtelt angewendet, z.B. RAID 1+0 (= RAID 10):



- **Effekt:** Eigenschaften können kombiniert werden. Häufig: RAID 10, 50 oder 60
- **Nachteil:** Relativ große Zahl an Platten nötig

Inhalt

- Wiederholung
- Herausforderungen
  - Zuverlässigkeit, Leistungsoptimierung, flexiblere Datenträgerverwaltung
- Intelligente Blockgeräte
  - *Logical Volume Management*, *Block Buffer Cache*, RAID
- **Konzepte moderner Dateisysteme**
  - Journale, *Log*-basierte Dateisysteme, *Copy-on-Write*-Prinzip
- Beispiel: Btrfs
- Zusammenfassung

Journaled 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
    - Dateiattribute verändern
    - Datei umbenennen
  - Protokollieren aller Änderungen am Dateisystem zusätzlich in einer Protokolldatei (***Log File*** oder ***Journal***)
  - Beim Bootvorgang wird die Protokolldatei mit den aktuellen Änderungen abgeglichen und dadurch Inkonsistenzen vermieden.

Journaled 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.

Journaled 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***

Journaled 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.
    - Alternative **chkdsk** benötigt viel Zeit bei großen Platten
- **Nachteile:**
  - ineffizienter, da zusätzliches Log-File geschrieben wird
    - daher normalerweise nur ***„Metadata Journaling“***, kein ***„Full Journaling“***
- Beispiele: NTFS, EXT3/4, XFS, ...

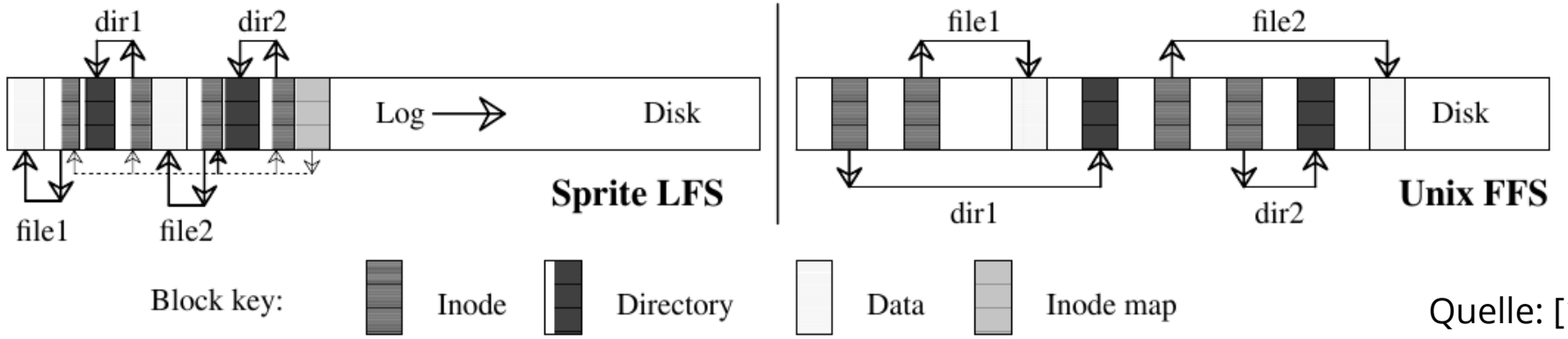
Log-strukturierte Dateisysteme

(engl. ***log-structured filesystems***)

- **Beobachtung:**
  - Durch große *Caches* werden Lesezugriffe seltener.
  - Schreibzugriffe sollten nicht verstreut durchgeführt werden.
- **(Radikaler) Ansatz:** Ein *Log* reicht aus für alles!
  - Blöcke werden nicht überschrieben, sondern an das *Log* gehängt
  - Anpassungen an Metadaten werden ebenfalls nur im *Log* getätigt
  - Schreiboperationen werden im Hauptspeicher gesammelt und dann als ein großes **Segment** (z.B. 1 MiB) herausgeschrieben
  - Einzig der Superblock hat eine feste Position auf der Platte
- Eine der ersten Umsetzungen in Sprite LFS [1]

Log-strukturierte Dateisysteme (2)

- **Beispiel:**

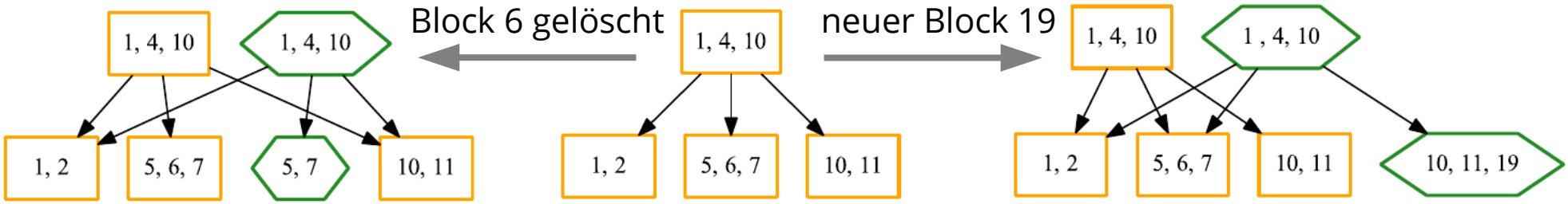


Quelle: [1]

  - *Log* arbeitet wie Ringpuffer: Vorne kommen Änderungen hinzu, hinten werden implizit Daten obsolet
  - ***Segment Cleaner***: Prozess zur Kompaktifizierung/Freigabe von Segmenten
- **Effekt:**
  - Konsistenz: Neue Segmente werden komplett oder gar nicht sichtbar
  - Auch beim Schreiben wird die Bandbreite der Platte genutzt
  - Wenn Speicher knapp wird, sinkt die Leistung drastisch

CoW: Copy-on-Write-Dateisysteme

- Viele moderne Dateisysteme verzichten auf Überschreiben
  - Idee aus Log-strukt. DS, nur flexibler bei der Belegung freier Bereiche
- **Beispiel:** Datei (B+-Baum) manipulieren … [2]



- **Beispiel:** Kompletten Verzeichnisbaum „kopieren“

Erst, wenn P oder Q geändert werden, erfolgt die Kopie – Grundlage für die effiziente Erstellung von Schnappschüssen (engl. ***snapshots***).

Inhalt

- Wiederholung
- Herausforderungen
  - Zuverlässigkeit, Leistungsoptimierung, flexiblere Datenträgerverwaltung
- Intelligente Blockgeräte
  - *Logical Volume Management*, *Block Buffer Cache*, RAID
- Konzepte moderner Dateisysteme
  - Journale, *Log*-basierte Dateisysteme, *Copy-on-Write*-Prinzip
- **Beispiel: Btrfs**
- Zusammenfassung

Btrfs: “butter” FS [2]

… laut Entwickler Chris Mason („*comes from the CoW*“)

- Weit verbreitet im Kontext von Linux, inspiriert durch ZFS
- **Features:** … ohne Ende
  - **Schnelle Schreibvorgänge:** spezielle „CoW-freundliche“ B+-Bäume
  - ressourcenschonende **Schnappschüsse**
  - **keine Datenverluste**
    - atomare Änderungen und Prüfsummen für alle Metadaten und Daten
  - Ausnutzung mehrerer **Laufwerke**
    - implementiert flexibles RAID: Unterscheidung von Daten und Metadaten
  - **Größenänderung** im laufenden Betrieb
  - **Datenkompression**

Inhalt

- Wiederholung
- Herausforderungen
  - Zuverlässigkeit, Leistungsoptimierung, flexiblere Datenträgerverwaltung
- Intelligente Blockgeräte
  - *Logical Volume Management*, *Block Buffer Cache*, RAID
- Konzepte moderner Dateisysteme
  - Journale, *Log*-basierte Dateisysteme, *Copy-on-Write*-Prinzip
- Beispiel: Btrfs
- **Zusammenfassung**

Zusammenfassung

- Moderne Dateisysteme …
  - beachten die **Eigenschaften aktueller Hardware**: Große Hauptspeicher (Cache), schnelle parallele CPU-Kerne, …
  - haben viele neue **Features**: *Snapshots*, Datenträgerverwaltung, Redundanz, …
- Grundsätzliche **Entwurfsfrage**: Gehören diese Funktionen in das Dateisystem (oder darunter)?
  - **Pro**: Mehr Flexibilität; Ausnutzung des Wissens über die Struktur des Dateisystems möglich, z.B. unterschiedlicher RAID-Level für Daten und Metadaten
  - **Contra**: Funktionalität käme auf der Treiberebene allen Dateisystemen zugute.

Literatur

[1] Mendel Rosenblum and John K. Ousterhout. 1992. *The design and implementation of a log-structured file system*. ACM Trans. Comput. Syst. 10, 1 (Feb. 1992), 26–52.
DOI: https://doi.org/10.1145/146941.146943

[2] Ohad Rodeh, Josef Bacik, and Chris Mason. 2013. Btrfs*: The Linux B-Tree Filesystem*. ACM Trans. Storage 9, 3, Article 9 (August 2013), 32 pages.
DOI: https://doi.org/10.1145/2501620.2501623