

Systemprogrammierung

Grundlage von Betriebssystemen

Teil C – XIII. Dateisysteme

Jürgen Kleinöder

18. Januar 2022

25. Januar 2022



Agenda

Medien

Speicherung von Dateien

Freispeicherverwaltung

Beispiele: Dateisysteme unter UNIX und Windows

Dateisysteme mit Fehlererholung

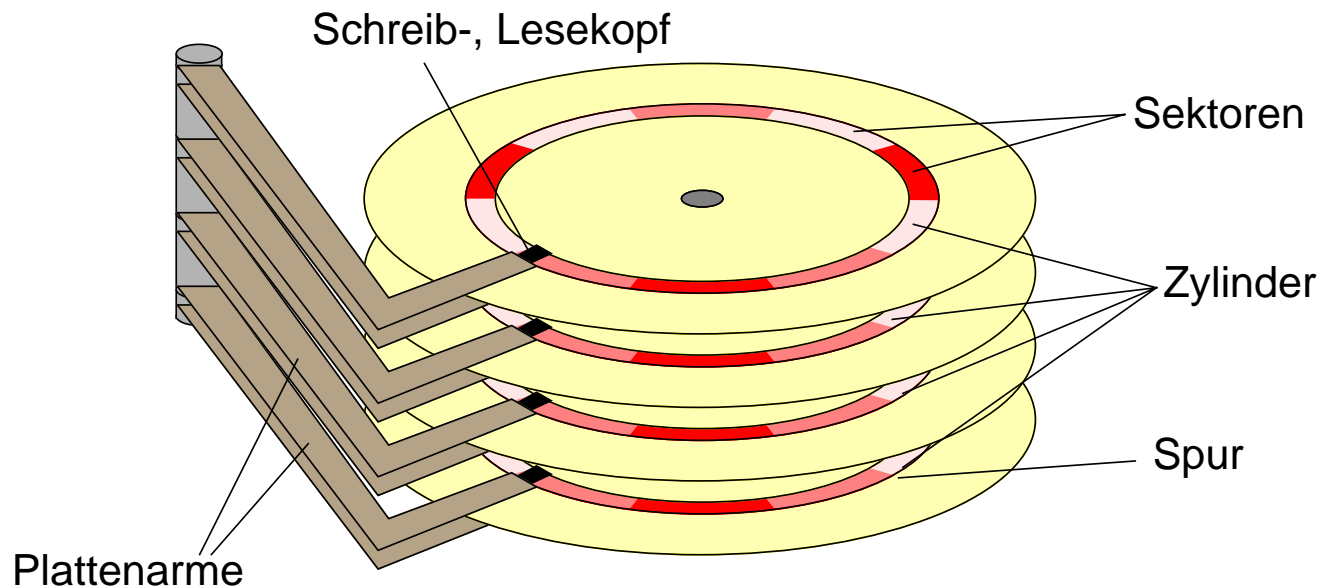
Datensicherung



Medien

Festplatten (hard disk drive - HDD)

- Lange Zeit häufigstes Medium zum Speichern von Dateien
- Aufbau einer Festplatte

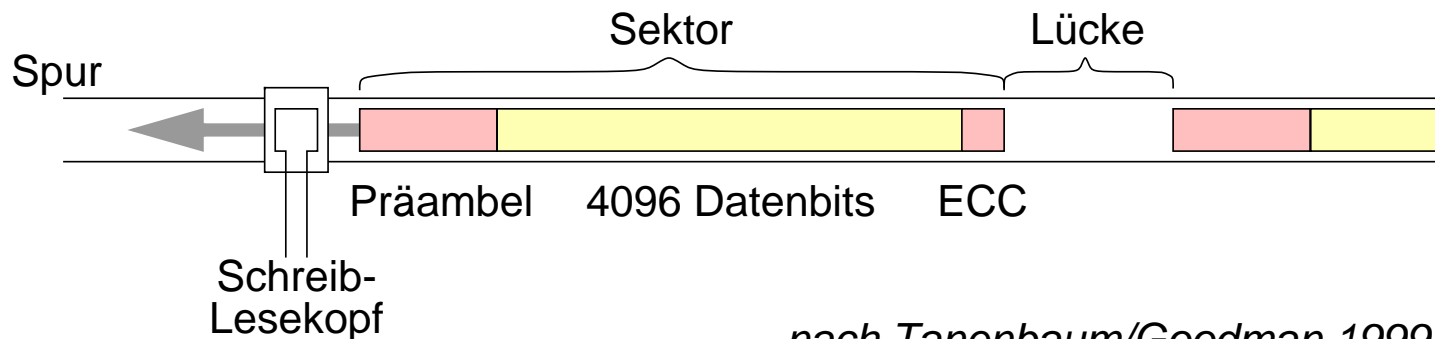


- Kopf schwebt auf Luftpolster



Festplatten (2)

■ Sektoraufbau



- Breite der Spur: $0,2 \mu\text{m}$,
bei "Shingled Magnetic Recording" überlappende Spuren
- Spuren pro Zentimeter: ca. 10.000
- Bitdichte: 34 Bit/ μm

■ Zonen

- Mehrere Zylinder (10–30) bilden eine Zone mit gleicher Sektorenanzahl (bessere Plattenausnutzung)



HDD / SSD

- SSD (solid-state disk)
 - Nicht-flüchtiger Halbleiter-Speicher (Flash-Mem.), meist NAND-Chips
 - Schnittstelle zum Rechner kompatibel mit HDD
 - inzwischen erheblich höhere Kapazitäten als bei HDD möglich
 - erheblich schneller und robuster (keine Mechanik)
 - noch erheblich teurer (Faktor 2 - 4)

- Zugriffsmerkmale
 - blockorientierter und wahlfreier Zugriff
 - Blockgröße zwischen 32 und 4096 Bytes (typisch 512 Bytes)
 - HDD: Zugriff erfordert Positionierung des Schwenkarms auf den richtigen Zylinder und Warten auf den entsprechenden Sektor
 - SSD: Lesen direkt und schnell, Schreiben nur blockweise und langsamer (Faktor 10), Blöcke müssen vorher explizit gelöscht worden sein
 - Interner Controller verbirgt die Hardware-Details, enthält Cache-Speicher, optimiert Zugriffe, Blöcke sind durchnummeriert



HDD / SSD (2)

■ Datenblätter von drei (alten) Beispielplatten

| Plattentyp | | Fujitsu M2344 (1987) | Seagate Cheetah | Seagate Barracuda |
|------------------------|--------------|-------------------------|-----------------|-------------------------|
| Kapazität | | 690 MB | 300 GB | 400 GB |
| Platten/Köpfe | | 8 / 28 | 4 / 8 | 781.422.768 Sektoren |
| Zylinderzahl | | 624 | 90.774 | |
| Cache | | - | 4 MB | 8 MB |
| Positionier- zeiten | Spur zu Spur | 4 ms | 0,5 ms | - |
| | mittlere | 16 ms | 5,3 ms | 8 ms |
| | maximale | 33 ms | 10,3 ms | - |
| Transferrate | | 2,4 MB/s | 320 MB/s | -150 MB/s |
| Rotationsgeschw. | | 3.600 U/min | 10.000 U/min | 7.200 U/min |
| eine Plattenumdrehung | | 16 ms | 6 ms | 8 ms |
| Stromaufnahme | | ? | 16-18 W | 12,8 W |

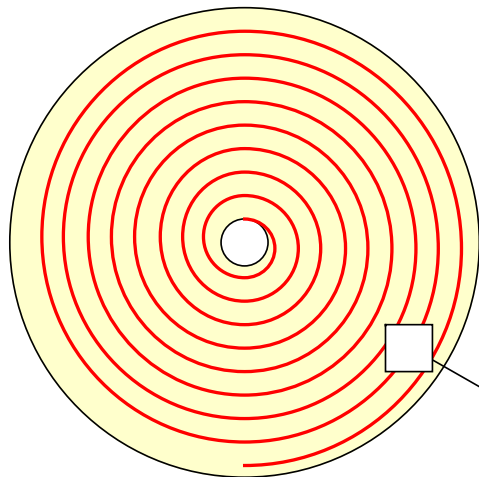
12.2020: Kapazität bis 20TB bei 7.200 U/min oder 0,6 - 2 TB bei 15.000 U/min,
Zugriffszeit ab 2 ms, Transferrate bis 315 MB/s

SSD: Kapazität bis 100 TB, Zugriffszeit ab. 0,03/0,3 ms, Transferrate bis 4 GB/s

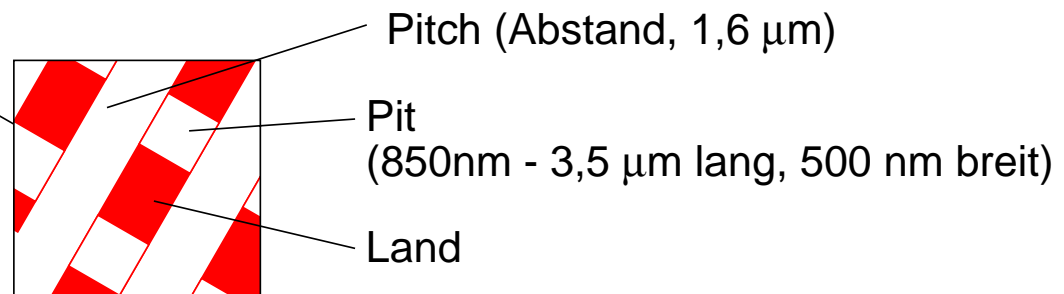


CD-ROM / DVD

■ Aufbau einer CD



- Spirale, beginnend im Inneren
- 22188 Umdrehungen (600 pro mm)
- Gesamtlänge 5,6 km



- **Pit:** Vertiefung, wird von Laser (780 nm Wellenlänge) abgetastet

■ DVD

- gleiches Grundkonzept, Wellenlänge des Lasers 650 nm
- Pits und Spurabstand weniger als halb so groß



CD-ROM / DVD (2)

■ Kodierung einer CD

- **Symbol:** ein Byte wird mit 14 Bits kodiert
(kann bereits bis zu zwei Bitfehler korrigieren)
- **Frame:** 42 Symbole (192 Datenbits, 396 Fehlerkorrekturbits)
- **Sektor** : 98 Frames werden zusammengefasst
(16 Bytes Präambel, 2048 Datenbytes, 288 Bytes Fehlerkorrektur)
- *Effizienz* : 7203 Bytes transportieren 2048 Nutzbytes (28,4 %)

■ Kodierung einer DVD

- Codierung mit Reed-Solomon-Product-Code, 8/16-Bit-Modulation,
43,2 % Nutzdaten

■ Transferrate

- CD-Single-Speed-Laufwerk: 75 Sektoren/Sek. (153.600 Bytes/Sek.)
- CD-72-fach-Laufwerk: 11,06 MB/Sek.
- DVD 1-fach: 1.3 MB/sec, 24-fach: 33.2 MB/sec



CD-ROM / DVD (3)

■ Kapazität

- CD: ca. 650 MB
- DVD single layer: 4.7 GB
- DVD dual layer: 8.5 GB, beidseitig: 17 GB

■ Varianten

- **DVD/CD-R** (Recordable): einmal beschreibbar
- **DVD/CD-RW** (Rewritable): mehrfach beschreibbar



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
- ★ Vorteile
 - Zugriff auf alle Blöcke mit minimaler Positionierzeit des Schwenkarms
 - Schneller direkter Zugriff auf bestimmter Dateiposition
 - Einsatz z. B. bei Systemen mit Echtzeitanforderungen



Kontinuierliche Speicherung (2)

▲ Probleme

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



Kontinuierliche Speicherung (3)

■ Variation

- Unterteilen einer Datei in Folgen von Blöcken (*Chunks, Extents*)
- Blockfolgen werden kontinuierlich gespeichert
- Pro Datei muss erster Block und Länge jedes einzelnen Chunks gespeichert werden

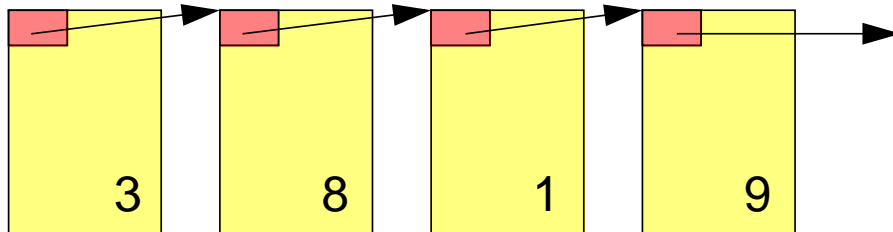
▲ Problem

- Verschnitt innerhalb einer Folge (siehe auch Speicherverwaltung: interner Verschnitt bei Seitenadressierung)



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 wachsen und verlängert werden



Verkettete Speicherung (2)

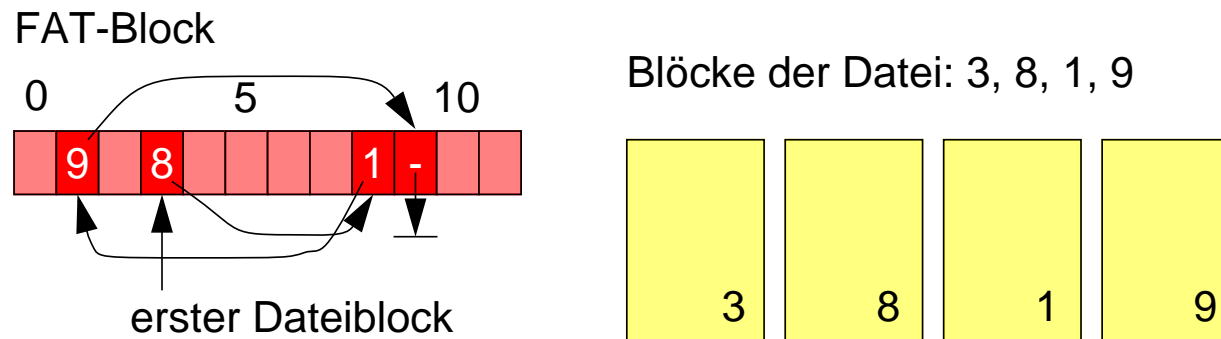
▲ Probleme

- Speicher für Verzeigerung geht von den Nutzdaten im Block ab (ungünstig im Zusammenhang mit Paging: Seite würde immer aus Teilen von zwei Plattenblöcken bestehen)
- Fehleranfälligkeit: Datei ist nicht restaurierbar wenn ein Fehler in der Verzeigerung entsteht
- schlechter direkter Zugriff auf bestimmte Dateiposition
- häufiges Positionieren des Schreib-Lesekopfs bei verstreuten Datenblöcken



Verkettete Speicherung (3)

- Verkettung wird in speziellem Plattenblocks gespeichert
 - FAT-Ansatz (*FAT: File Allocation Table*), z. B. MS-DOS, Windows 95



- Dateiverzeichnis enthält zu jeder Datei die Nummer des ersten Blocks
- ★ Vorteile
 - kompletter Inhalt des Datenblocks ist nutzbar (günstig bei Paging)
 - mehrfache Speicherung der FAT möglich: Einschränkung der Fehleranfälligkeit



Verkettete Speicherung (4)

▲ Probleme

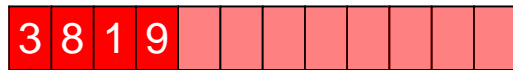
- mindestens ein zusätzlicher Block muss geladen werden (Caching der FAT zur Effizienzsteigerung nötig)
- FAT enthält Verkettungen für alle Dateien: das Laden der FAT-Blöcke lädt auch nicht benötigte Informationen
- aufwändige Suche nach dem zugehörigen Datenblock bei bekannter Position in der Datei
- häufiges Positionieren des Schreib-, Lesekopfs bei verstreuten Datenblöcken



Indiziertes Speichern

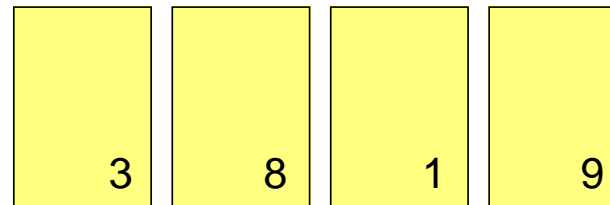
- Spezieller Plattenblock enthält Blocknummern der Datenblocks einer Datei
 - entspricht eigener FAT pro Datei

Indexblock



↑
erster Dateiblock

Blöcke der Datei: 3, 8, 1, 9



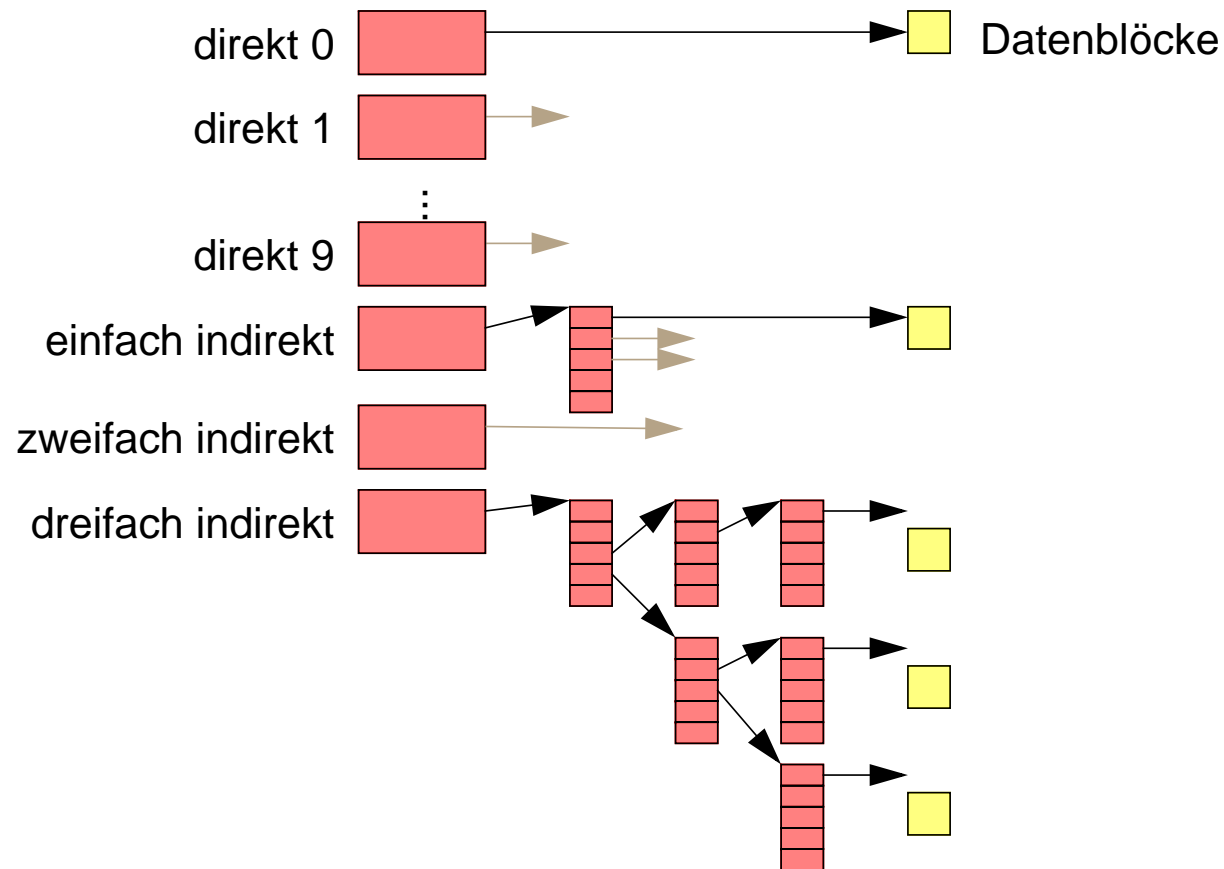
▲ Problem

- feste Anzahl von Blöcken im Indexblock
 - Verschnitt bei kleinen Dateien
 - Erweiterung nötig für große Dateien



Indiziertes Speichern (2)

■ Beispiel UNIX Inode



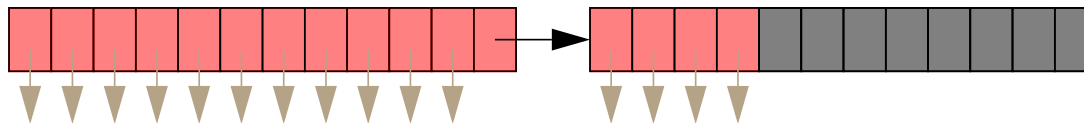
Indiziertes Speichern (3)

- ★ 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)



Freispeicherverwaltung

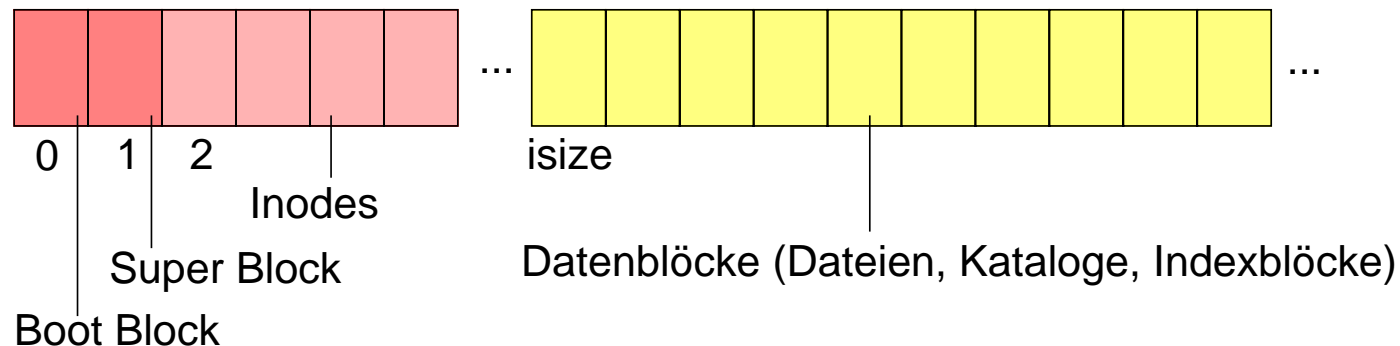
- Prinzipiell ähnlich wie Verwaltung von freiem Hauptspeicher
 - Bitvektoren zeigen für jeden Block Belegung an
 - verkettete Listen repräsentieren freie Blöcke
 - Verkettung kann in den freien Blöcken vorgenommen werden
 - Optimierung: aufeinanderfolgende Blöcke werden nicht einzeln aufgenommen, sondern als Stück verwaltet
 - Optimierung: ein freier Block enthält viele Blocknummern weiterer freier Blöcke und evtl. die Blocknummer eines weiteren Blocks mit den Nummern freier Blöcke



Beispiel: UNIX Dateisysteme

System V File System

■ Blockorganisation

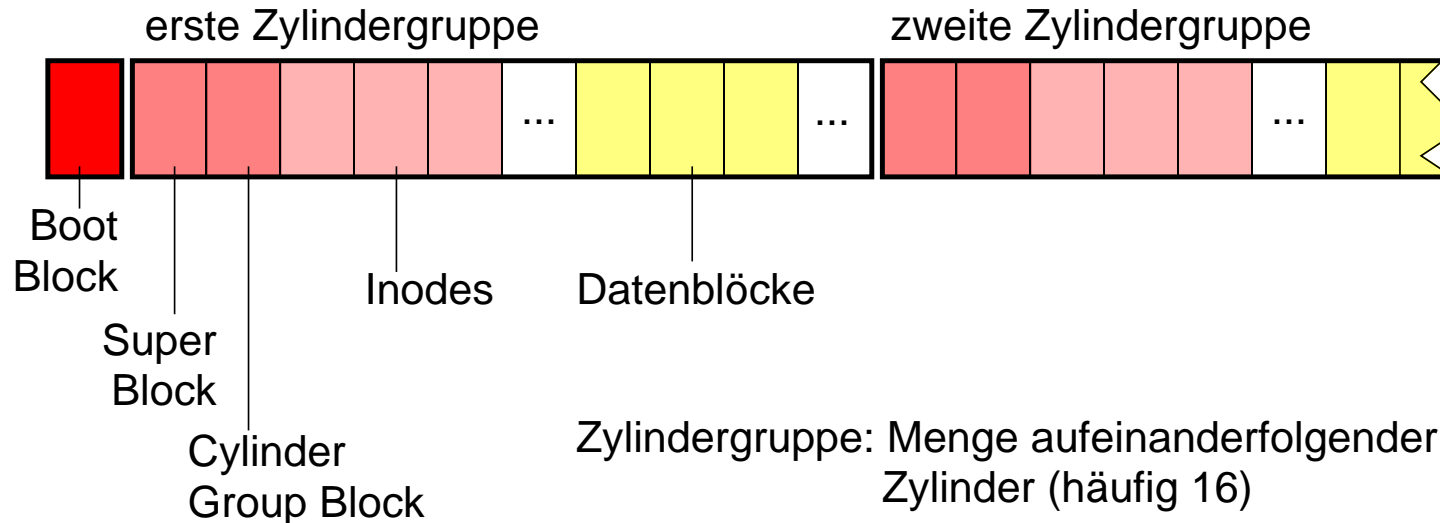


- Boot Block enthält Informationen zum Laden eines initialen Programms
- Super Block enthält Verwaltungsinformation für ein Dateisystem
 - Anzahl der Blöcke, Anzahl der Inodes
 - Anzahl und Liste freier Blöcke und freier Inodes
 - Attribute (z.B. *Modified flag*)
- seit den 1970er Jahren in den ersten UNIX-Systemen eingesetzt



BSD 4.2 (Berkeley Fast File System)

■ Blockorganisation



- Kopie des Super Blocks in jeder Zylindergruppe
- freie Inodes u. freie Datenblöcke werden im *Cylinder Group Block* gehalten
- eine Datei wird möglichst innerhalb einer Zylindergruppe gespeichert

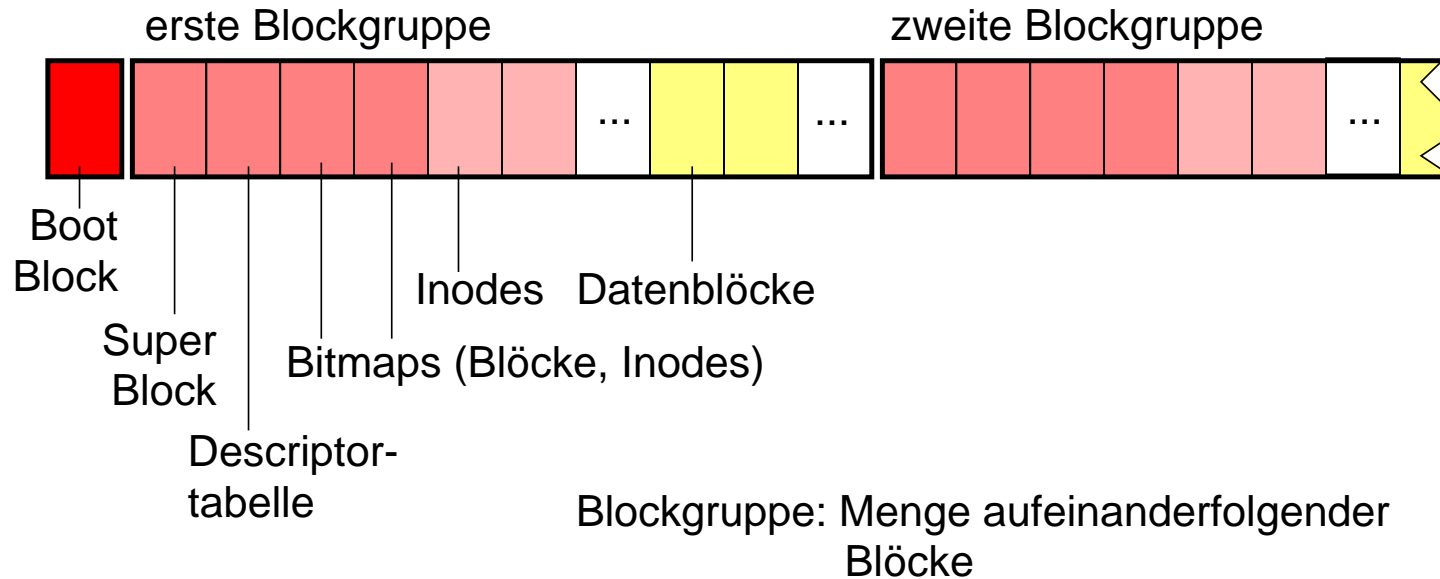
★ Vorteil: kürzere Positionierungszeiten

- Optimierung der Zugriffszeiten. in den 1980er Jahren entwickelt



Linux EXT2/3/4

■ EXT2: Blockorganisation



■ Ähnliches Layout wie BSD FFS

■ Blockgruppen unabhängig von Zylindern

■ EXT3: Erweiterung als Journaling-File-System (siehe Abschnitt 7.2)

■ EXT4: Einführung von Extents (siehe NTFS) und viele neue Details



Beispiel: Windows NTFS

- Dateisystem für Windows-Systeme (seit Windows NT 3.1, 1993)
- Datei
 - beliebiger Inhalt; für das Betriebssystem ist der Inhalt transparent
 - Rechte verknüpft mit NT-Benutzern und -Gruppen
 - Datei kann automatisch komprimiert oder verschlüsselt gespeichert werden
 - große Dateien bis zu 2^{64} Bytes lang
 - Hard links: mehrere Einträge derselben Datei in verschiedenen Katalogen möglich
- Dateiinhalt: Sammlung von *Streams*
 - *Stream*: einfache, unstrukturierte Folge von Bytes
 - "normaler Inhalt" = unbenannter Stream (default stream)
 - dynamisch erweiterbar
 - Syntax: dateiname:streamname



Dateiverwaltung

- Basiseinheit „Cluster“
 - 512 Bytes bis 4 Kilobytes (beim Formatieren festgelegt)
 - wird auf eine Menge von hintereinanderfolgenden Blöcken abgebildet
 - logische Cluster-Nummer als Adresse (LCN)

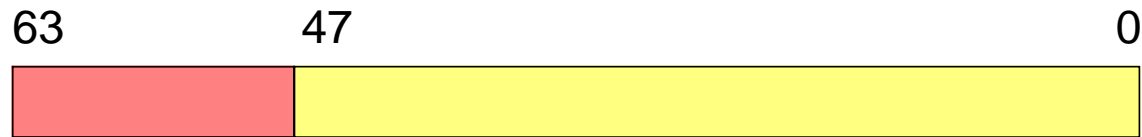
- Basiseinheit „Strom“
 - jede Datei kann mehrere (Daten-)Ströme speichern
 - einer der Ströme wird für die eigentlichen Daten verwendet
 - Dateiname, MS-DOS Dateiname, Zugriffsrechte, Attribute und Zeitstempel werden jeweils in eigenen Datenströmen gespeichert (leichte Erweiterbarkeit des Systems)



Dateiverwaltung (2)

■ *File-Reference*

- Bezeichnet eindeutig eine Datei oder einen Katalog



Sequenz-
nummer

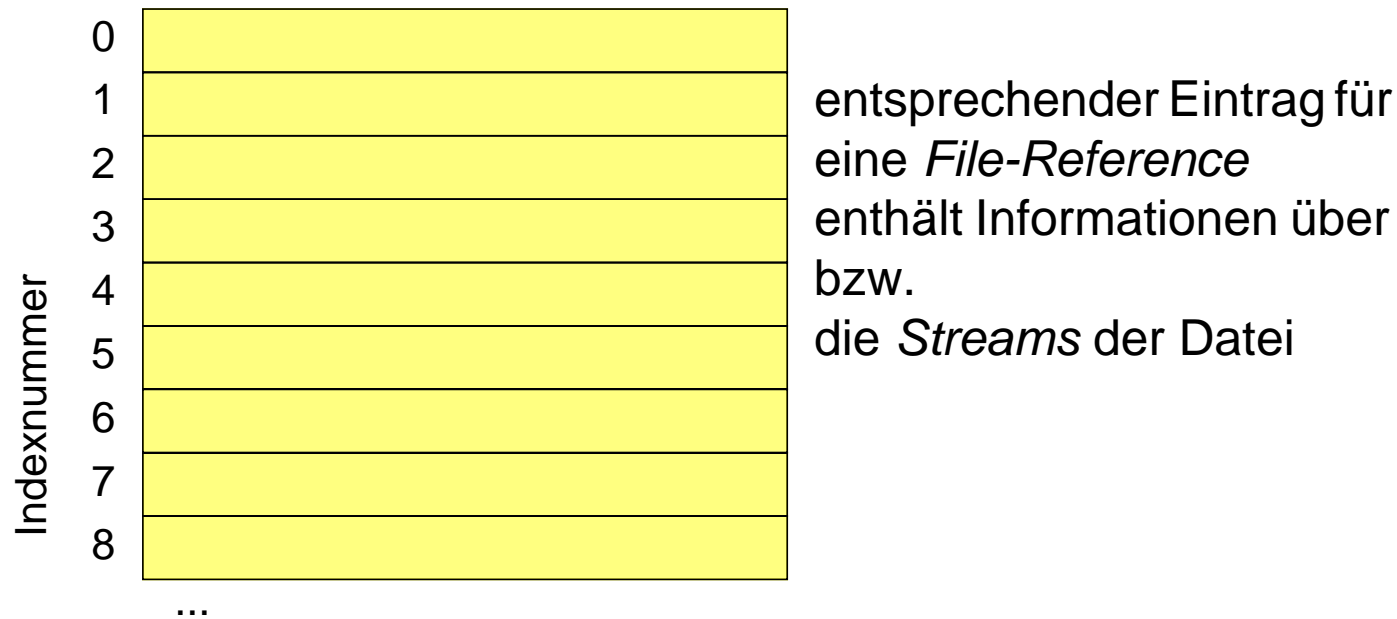
Dateinummer

- Dateinummer ist Index in eine globale Tabelle (*MFT: Master File Table*)
- Sequenznummer wird hochgezählt, für jede neue Datei mit gleicher Dateinummer



Master-File-Table

- Rückgrat des gesamten Systems
 - große Tabelle mit gleich langen Elementen (1KB, 2KB oder 4KB groß, je nach Clustergröße)
 - kann dynamisch erweitert werden

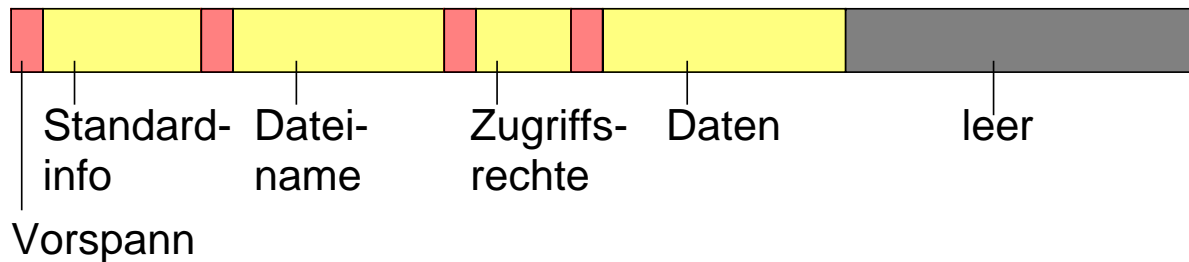


- Index in die Tabelle ist Teil der *File-Reference*



Master-File-Table (2)

■ Eintrag für eine kurze Datei



■ Streams

■ Standard-Information (immer in der MFT)

- enthält Länge, Standard-Attribute, Zeitstempel, Anzahl der Hard links, Sequenznummer der gültigen File-Reference

■ Dateiname (immer in der MFT)

- kann mehrfach vorkommen (Hard links)

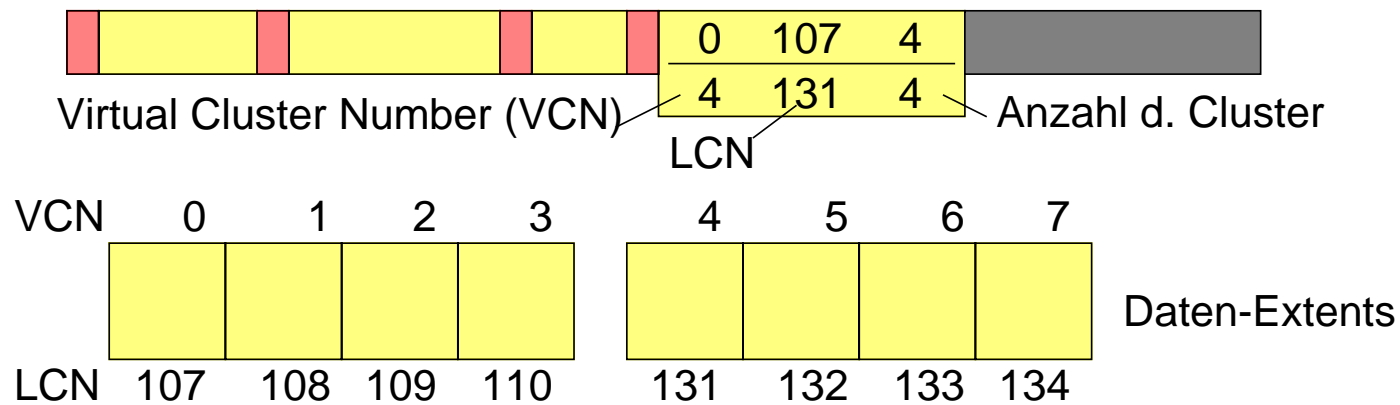
■ Zugriffsrechte (*Security Descriptor*)

■ Eigentliche Daten



Master-File-Table (3)

- Eintrag für eine längere Datei



- **Extents** werden außerhalb der MFT in aufeinanderfolgenden Clustern gespeichert
- Lokalisierungsinformationen werden in einem eigenen Stream gespeichert



Master-File-Table (4)

■ Mögliche weitere Streams (*Attributes*)

■ Index

- Index über einen Attributsschlüssel (z.B. Dateinamen) implementiert Katalog

■ Indexbelegungstabelle

- Belegung der Struktur eines Index

■ Attributliste (immer in der MFT)

- wird benötigt, falls nicht alle Streams in einen MFT Eintrag passen
- referenzieren weitere MFT Einträge und deren Inhalt

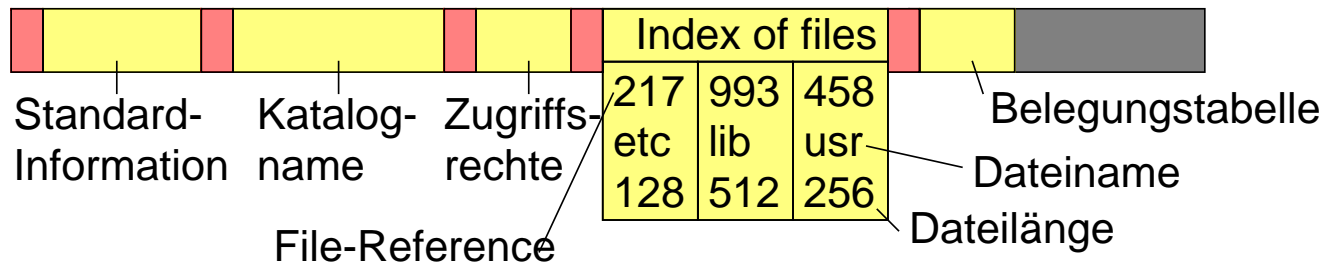
■ Streams mit beliebigen Daten

- wird gerne zum Verstecken von Viren genutzt, da viele Standard-Werkzeuge von Windows nicht auf die Bearbeitung mehrerer Streams eingestellt sind (arbeiten nur mit dem unbenannten Stream)



Master File Table (5)

■ Eintrag für einen kurzen Katalog

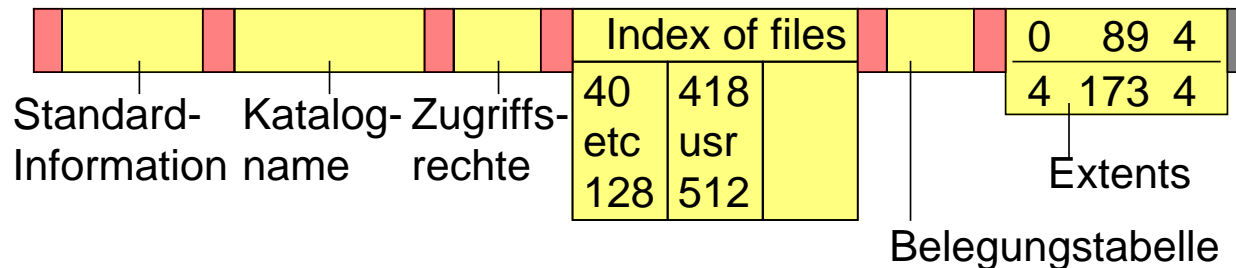


- Dateien des Katalogs werden mit File-References benannt
- Name und Standard-Attribute (z.B. Länge) der im Katalog enthaltenen Dateien und Kataloge werden auch im Index gespeichert (doppelter Aufwand beim Update; schnellerer Zugriff beim Kataloglisten)



Master File Table (6)

Eintrag für einen längeren Katalog



Daten-Extents

| VCN | 0 | 1 | 2 | 3 | 4 | 5 | 6 | 7 |
|-----|-----|------|-----|----|-----|------|-----|-----|
| | 918 | 773 | 473 | | 873 | 910 | | 10 |
| | cd | csh | doc | | lib | news | | tmp |
| | 128 | 2781 | 128 | | 512 | 1024 | | 128 |
| LCN | 89 | 90 | 91 | 92 | 173 | 174 | 175 | 176 |

File reference
Dateiname
Dateilänge

- Speicherung als B⁺-Baum (sortiert, schneller Zugriff)
- in einen Cluster passen zwischen 3 und 15 Dateien (im Bild nur eine)



Metadaten

- Alle Metadaten werden in Dateien gehalten

| | | | |
|-------------|-----|------------------------------|--------------------------|
| Indexnummer | 0 | MFT | Feste Dateien in der MFT |
| | 1 | MFT Kopie (teilweise) | |
| | 2 | Log File | |
| | 3 | Volume Information | |
| | 4 | Attributtabelle | |
| | 5 | Wurzelkatalog | |
| | 6 | Clusterbelegungstabelle | |
| | 7 | Boot File | |
| | 8 | Bad Cluster File | |
| | ... | | |
| | 16 | Benutzerdateien u. -kataloge | |
| | 17 | | |
| | ... | | |



Metadaten (2)

- Bedeutung der Metadateien
 - MFT und MFT Kopie: MFT wird selbst als Datei gehalten (d.h. Cluster der MFT stehen im Eintrag 0)
MFT Kopie enthält die ersten 16 Einträge der MFT (Fehlertoleranz)
 - Log File: enthält protokollierte Änderungen am Dateisystem
 - Volume Information: Name, Größe und ähnliche Attribute des Volumes
 - Attributtabelle: definiert mögliche Ströme in den Einträgen
 - Wurzelkatalog
 - Clusterbelegungstabelle: Bitmap für jeden Cluster des Volumes
 - Boot File: enthält initiales Programm zum Laden, sowie ersten Cluster der MFT
 - Bad Cluster File: enthält alle nicht lesbaren Cluster der Platte
NTFS markiert automatisch alle schlechten Cluster und versucht die Daten in einen anderen Cluster zu retten



Fehlererholung

- NTFS ist ein Journaling-File-System
 - Änderungen an der MFT und an Dateien werden protokolliert.
 - Konsistenz der Daten und Metadaten kann nach einem Systemausfall durch Abgleich des Protokolls mit den Daten wieder hergestellt werden.

- ▲ Nachteile
 - etwas ineffizienter
 - nur für Volumes >400 MB geeignet



Dateisysteme mit Fehlererholung

- Metadaten und aktuell genutzte Datenblöcke geöffneter Dateien werden im Hauptspeicher gehalten (Dateisystem-Cache)
 - effizienter Zugriff
 - Konsistenz zwischen Cache und Platte muss regelmäßig hergestellt werden
 - synchrone Änderungen: Operation kehrt erst zurück, wenn Änderungen auf der Platte gespeichert wurden
 - asynchrone Änderungen: Änderungen erfolgen nur im Cache, Operation kehrt danach sofort zurück, Synchronisation mit der Platte erfolgt später

- Mögliche Fehlerursachen
 - Stromausfall (oder dummer Benutzer schaltet einfach Rechner aus)
 - Systemabsturz



Konsistenzprobleme

- Fehlerursachen & Auswirkungen auf das Dateisystem
 - Cache-Inhalte und aktuelle E/A-Operationen gehen verloren
 - 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
- ▲ Datenverluste bei Reparatur möglich
- ▲ Große Platten bedeuten lange Laufzeiten der Reparaturprogramme



Journaling-File-Systems

- Zusätzlich zum Schreiben der Daten und Meta-Daten (z. B. Inodes) wird ein Protokoll der Änderungen geführt
 - Grundidee: Log-based Recovery bei Datenbanken
 - 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*)
 - beim Bootvorgang wird Protokolldatei mit den aktuellen Änderungen abgeglichen und damit werden Inkonsistenzen vermieden.



Journaling-File-Systems (2)

■ Protokollierung

- für jeden Einzelvorgang einer Transaktion wird zunächst ein Logeintrag erzeugt und
- danach die Änderung am Dateisystem vorgenommen
- dabei gilt:
 - der Logeintrag wird immer **vor** der eigentlichen Änderung auf Platte geschrieben
 - wurde etwas auf Platte geändert, steht damit sicher auch der Protokolleintrag dazu auf der Platte

■ Fehlererholung

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



Journaling-File-Systems (3)

- Beispiel: Löschen einer Datei im NTFS
 - Vorgänge der Transaktion
 - Beginn der Transaktion
 - Freigeben der Extents durch Löschen der entsprechenden Bits in der Belegungstabelle (gesetzte Bits kennzeichnen belegten Cluster)
 - Freigeben des MFT-Eintrags der Datei
 - Löschen des Katalogeintrags der Datei (evtl. Freigeben eines Extents aus dem Index)
 - Ende der Transaktion
 - Alle Vorgänge werden unter der File-Reference im Log-File protokolliert, danach jeweils durchgeführt.
 - Protokolleinträge enthalten Informationen zum *Redo* und zum *Undo*



Journaling-File-Systems (4)

- Log vollständig (Ende der Transaktion wurde protokolliert und steht auf Platte):
 - *Redo* der Transaktion:
alle Operationen werden wiederholt, falls nötig
- Log unvollständig (Ende der Transaktion steht nicht auf Platte):
 - *Undo* der Transaktion:
in umgekehrter Reihenfolge werden alle Operation rückgängig gemacht
- Checkpoints
 - Log-File kann nicht beliebig groß werden
 - gelegentlich wird für einen konsistenten Zustand auf Platte gesorgt (*Checkpoint*) und dieser Zustand protokolliert (alle Protokolleinträge von vorher können gelöscht werden)



Journaling-File-Systems (5)

★ Ergebnis

- eine Transaktion ist entweder vollständig durchgeführt oder gar nicht
- Nutzer können 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

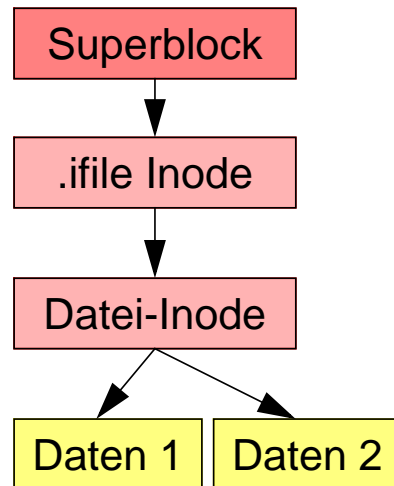
▲ Nachteile

- etwas langsamer, da zusätzlich Log-File-Einträge geschrieben werden müssen
- Beispiele: NTFS, EXT3, EXT4, ReiserFS



Copy-on-Write- / Log-Structured-File-Systems

- Alternatives Konzept zur Realisierung von atomaren Änderungen
- Alle Änderungen im Dateisystem erfolgen auf Kopien
 - Der Inhalt veränderter Blöcke wird in einen neuen Block geschrieben

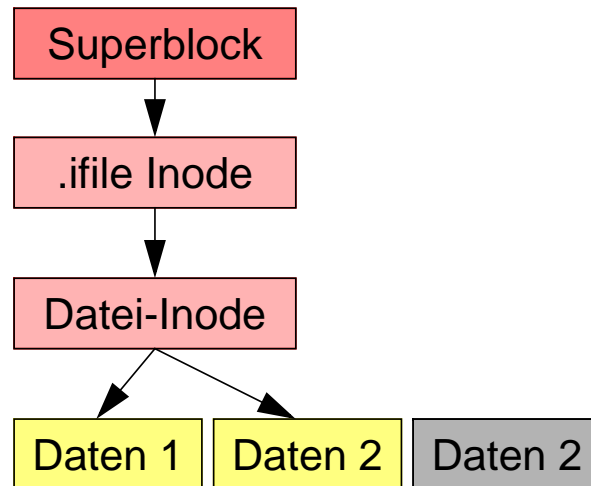


- Beispiel LinLogFS: Superblock einziger nicht ersetzter Block



Copy-on-Write- / Log-Structured-File-Systems

- Alternatives Konzept zur Realisierung von atomaren Änderungen
- Alle Änderungen im Dateisystem erfolgen auf Kopien
 - Der Inhalt veränderter Blöcke wird in einen neuen Block geschrieben

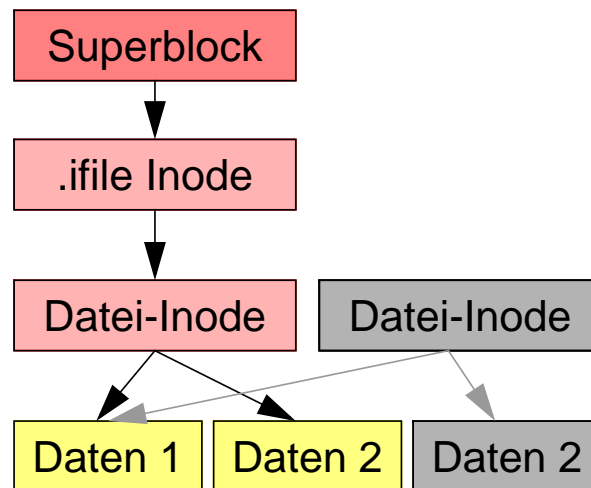


- Beispiel LinLogFS: Superblock einziger nicht ersetzter Block



Copy-on-Write- / Log-Structured-File-Systems

- Alternatives Konzept zur Realisierung von atomaren Änderungen
- Alle Änderungen im Dateisystem erfolgen auf Kopien
 - Der Inhalt veränderter Blöcke wird in einen neuen Block geschrieben

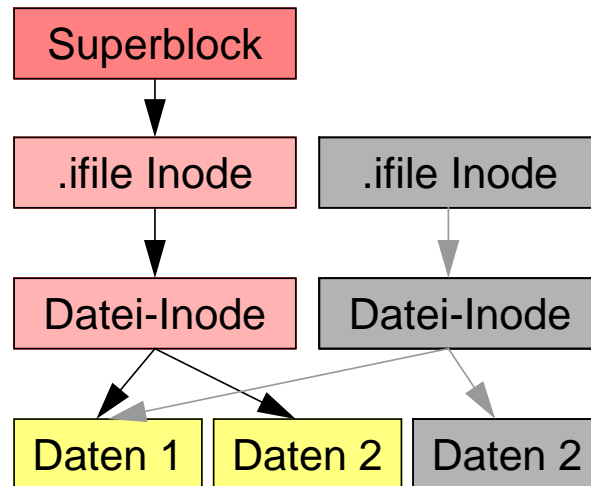


- Beispiel LinLogFS: Superblock einziger nicht ersetzter Block



Copy-on-Write- / Log-Structured-File-Systems

- Alternatives Konzept zur Realisierung von atomaren Änderungen
- Alle Änderungen im Dateisystem erfolgen auf Kopien
 - Der Inhalt veränderter Blöcke wird in einen neuen Block geschrieben

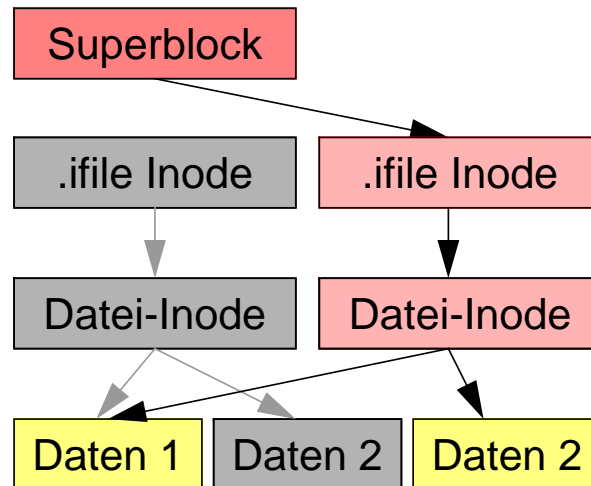


- Beispiel LinLogFS: Superblock einziger nicht ersetzter Block



Copy-on-Write- / Log-Structured-File-Systems

- Alternatives Konzept zur Realisierung von atomaren Änderungen
- Alle Änderungen im Dateisystem erfolgen auf Kopien
 - Der Inhalt veränderter Blöcke wird in einen neuen Block geschrieben

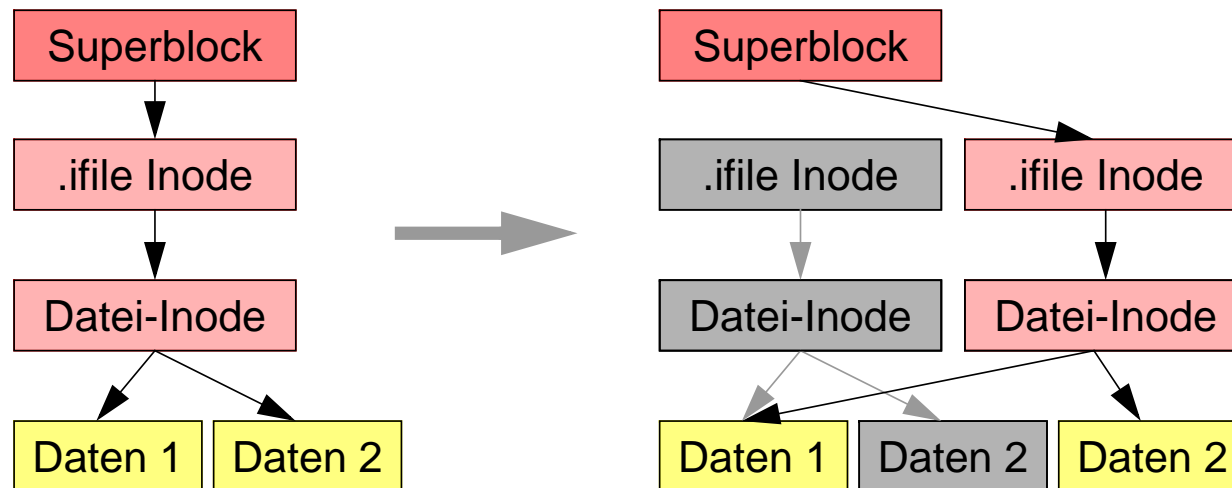


- Beispiel LinLogFS: Superblock einziger nicht ersetzter Block



Copy-on-Write- / Log-Structured-File-Systems

- Alternatives Konzept zur Realisierung von atomaren Änderungen
- Alle Änderungen im Dateisystem erfolgen auf Kopien
 - Der Inhalt veränderter Blöcke wird in einen neuen Block geschrieben



- Beispiel LinLogFS: Superblock einziger statischer Block (Anker im System)



Copy-on-Write- / Log-Structured-File-Systems (2)

★ Vorteile

- Gute Schreibeffizienz - vor allem bei Log-Structured-File-Systems
- Datenkonsistenz bei Systemausfällen
 - ein atomare Änderung macht alle zusammengehörigen Änderungen sichtbar
- Schnappschüsse / Checkpoints einfach realisierbar

▲ Nachteile

- Erzeugt starke Fragmentierung, die sich beim Lesen auswirken kann
 - ↳ Performanz nur akzeptabel, wenn Lesen primär aus Cache erfolgen kann oder Positionierzeiten keine Rolle spielen (SSD)

■ Unterschied zwischen Copy-on-Write- und Log-Structured-File-Systems

- Log-Structured-File-Systems schreiben kontinuierlich an das Ende des belegten Plattenbereichs und geben vorne die Blöcke wieder frei (kontinuierlicher Log)
- Beispiele: Log-Structured: LinLogFS, BSD LFS
 Copy-on-Write: ZFS, Btrfs (Oracle)



Fehlerhafte Plattenblöcke

- Blöcke, die beim Lesen Fehlermeldungen erzeugen
 - z.B. Prüfsummenfehler
- Hardwarelösung
 - Platte und Plattencontroller bemerken selbst fehlerhafte Blöcke und maskieren diese aus
 - Zugriff auf den Block wird vom Controller automatisch auf einen „gesunden“ Block umgeleitet
- Softwarelösung
 - File-System bemerkt fehlerhafte Blöcke und markiert diese auch als belegt



Datensicherung

- Schutz vor dem Totalausfall von Platten
 - z. B. durch Head-Crash oder andere Fehler

Sichern der Daten auf Tertiärspeicher

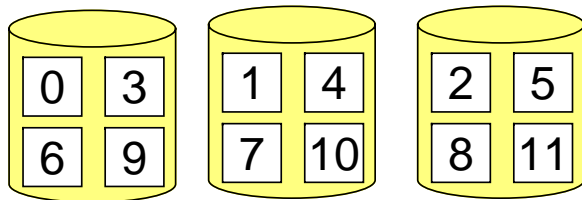
- ▶ Bänder, Bandroboter mit vorgelagertem Platten-Cache
- ▶ WORM-Speicherplatten (*Write Once Read Many*)
- Sichern großer Datenbestände
 - Total-Backups benötigen lange Zeit
 - Inkrementelle Backups sichern nur Änderungen ab einem bestimmten Zeitpunkt
 - Mischen von Total-Backups mit inkrementellen Backups



Einsatz mehrerer (redundanter) Platten

- Gestreifte Platten (*Striping*; RAID 0)

- Daten werden über mehrere Platten gespeichert



- Datentransfers sind nun schneller, da mehrere Platten gleichzeitig angesprochen werden können

- ▲ Nachteil

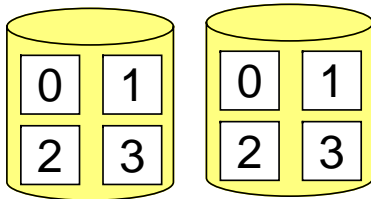
- keinerlei Datensicherung: Ausfall einer Platte lässt Gesamtsystem ausfallen



Einsatz mehrerer redundanter Platten (2)

- Gespiegelte Platten (*Mirroring*; RAID 1)

- Daten werden auf zwei Platten gleichzeitig gespeichert



- Implementierung durch Software (File-System, Plattentreiber) oder Hardware (spez. Controller)
- eine Platte kann ausfallen
- schnelleres Lesen (da zwei Platten unabhängig voneinander beauftragt werden können)

- ▲ Nachteil

- doppelter Speicherbedarf

- wenig langsames Schreiben durch Warten auf zwei Plattentransfers

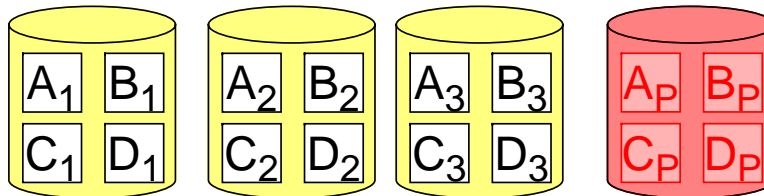
- Verknüpfung von RAID 0 und 1 möglich (RAID 0+1)



Einsatz mehrerer redundanter Platten (3)

■ Paritätsplatte (RAID 4)

- Daten werden über mehrere Platten gespeichert, eine Platte enthält Parität



- Paritätsblock enthält byteweise XOR-Verknüpfungen von den zugehörigen Blöcken aus den anderen Streifen
- eine Platte kann ausfallen
- schnelles Lesen
- prinzipiell beliebige Plattenanzahl (ab drei)



Einsatz mehrerer redundanter Platten (4)

▲ Nachteil von RAID 4

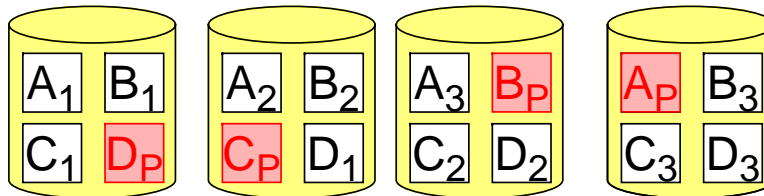
- jeder Schreibvorgang erfordert auch das Schreiben des Paritätsblocks
- Erzeugung des Paritätsblocks durch Speichern des vorherigen Blockinhalts möglich: $P_{\text{neu}} = P_{\text{alt}} \oplus B_{\text{alt}} \oplus B_{\text{neu}}$ (P=Parity, B=Block)
- Schreiben eines kompletten Streifens benötigt nur einmaliges Schreiben des Paritätsblocks
- Paritätsplatte ist hoch belastet



Einsatz mehrerer redundanter Platten (5)

■ Verstreuter Paritätsblock (RAID 5)

- Paritätsblock wird über alle Platten verstreut



- zusätzliche Belastung durch Schreiben des Paritätsblocks wird auf alle Platten verteilt
- heute gängigstes Verfahren redundanter Platten
- Vor- und Nachteile sonst wie RAID 4

■ Doppelte Paritätsblöcke (RAID 6)

- ähnlich zu RAID 5, aber zwei Paritätsblöcke (verkräftet damit den Ausfall von bis zu zwei Festplatten)
- wichtig bei sehr großen, intensiv genutzten RAID-Systemen, wenn die Wiederherstellung der Paritätsinformation nach einem Plattenausfall lange dauern kann

