ulm university universität ulm

Grundlagen der Betriebssysteme

[CS2100]

Sommersemester 2014

Heiko Falk

Institut für Eingebettete Systeme/Echtzeitsysteme
Ingenieurwissenschaften und Informatik
Universität Ulm

ulm university universität ulm

Kapitel 5

Filesysteme

Inhalte der Vorlesung

Inhalte des Kapitels (1)

5. Filesysteme

- Einleitung
  - Aufbau von Festplatten
  - Bestandteile von Filesystemen
  - Operationen und Attribute
- Beispiel: UNIX/Linux
  - Baumstruktur, Pfade
  - *Links*
  - *Mounten* von Filesystemen
  - *Inodes*
  - UNIX-Filesysteme: UFS, BSD 4.2, EXT2
- ...

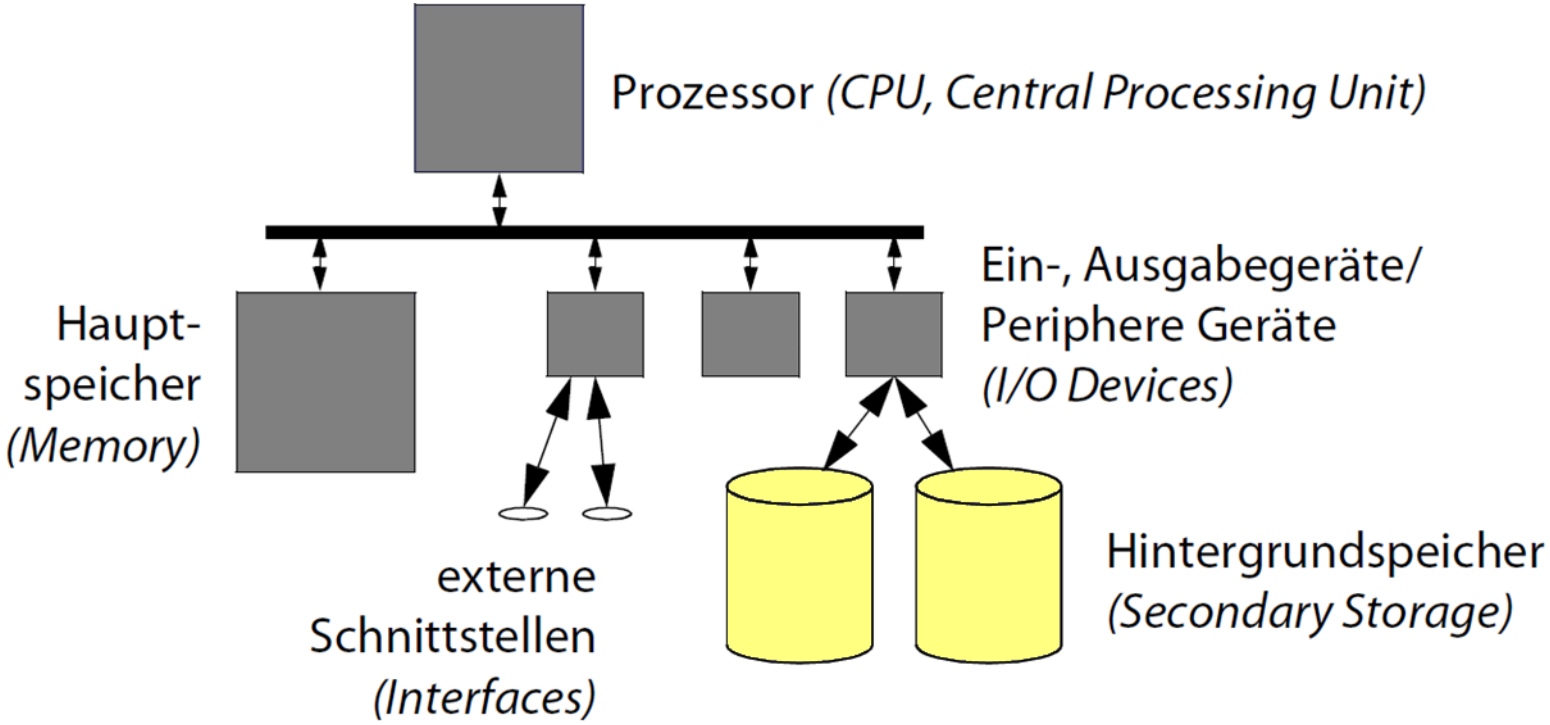
Inhalte des Kapitels (2)

5. Filesysteme

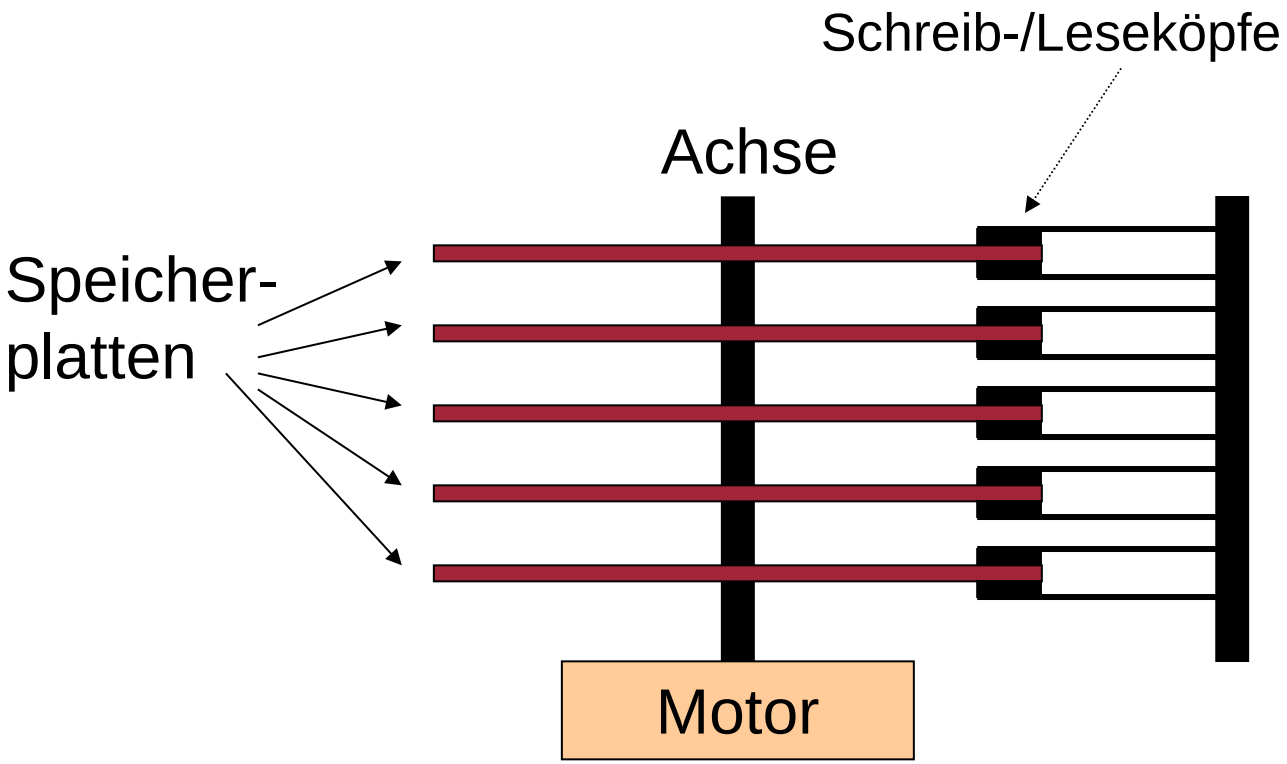
- ...
- Beispiel: FAT32
  - Struktur
  - *File Allocation Table*
  - Verzeichnisspeicherung
- Beispiel: NTFS
  - Struktur *(Streams)*
  - Dateiverwaltung *(Master File Table)*
  - Metadaten
- Zuverlässige Filesysteme
  - Journal *(Log)*; *Undo & Redo*
  - *Log-Structured File Systems*
- Limitierung der Plattennutzung, fehlerhafte Blöcke

Einordnung

Betroffene physikalische Ressourcen

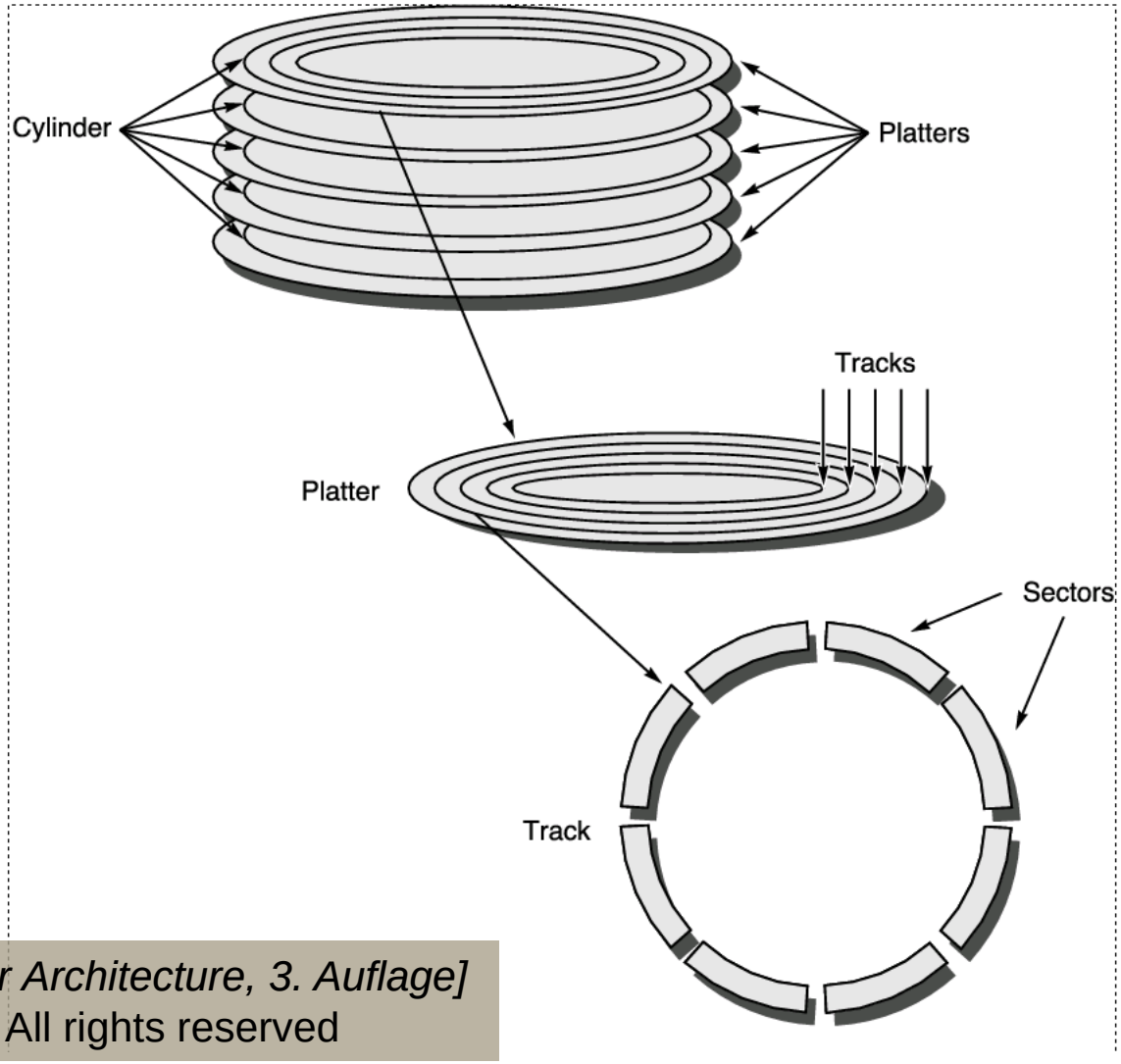


Schematischer Aufbau eines Plattenlaufwerks



Einteilung der Platten in Sektoren, Spuren und Zylinder (1)

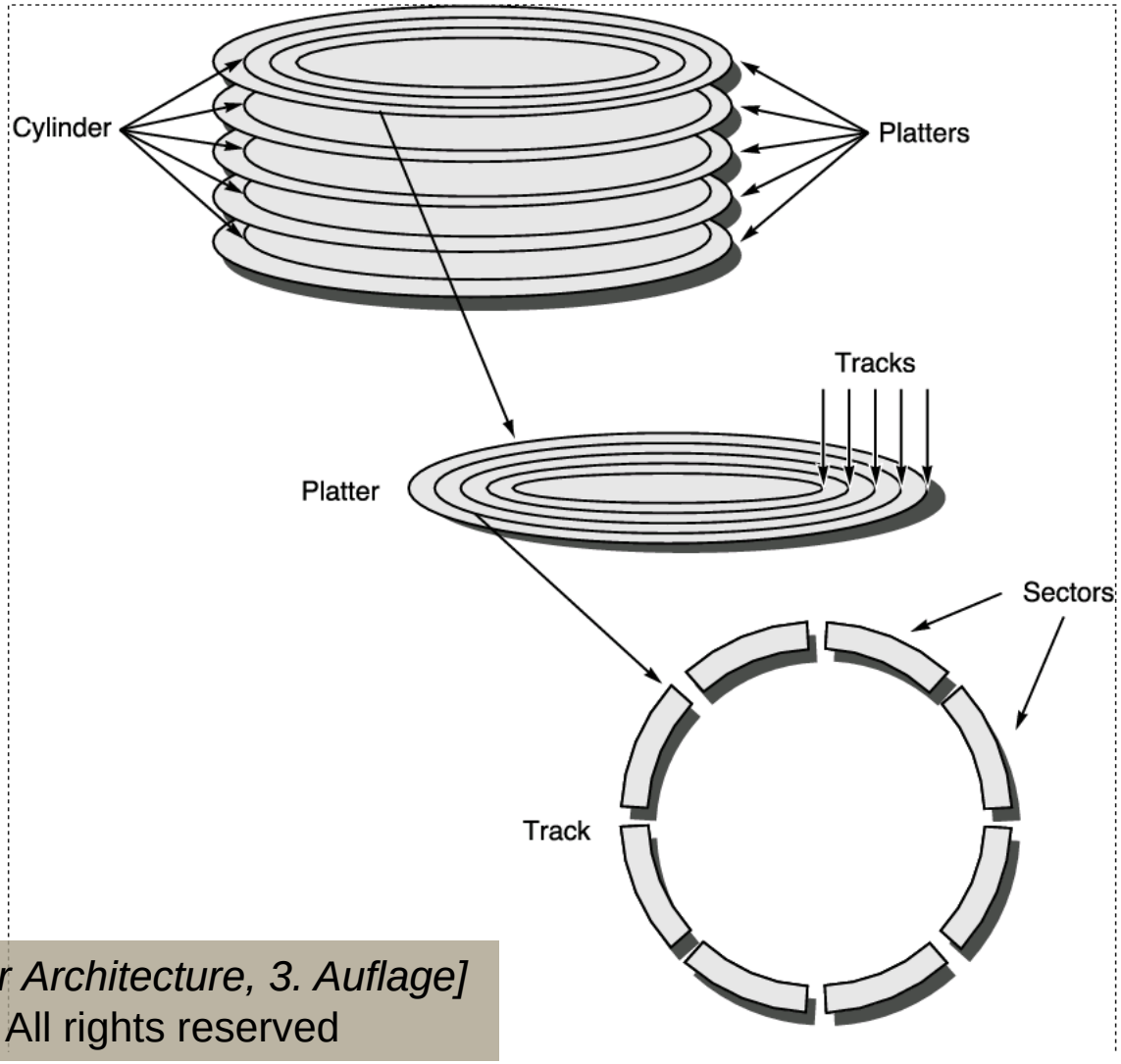
- Jede Speicherplatte *(Platter)* hat auf Ober- und Unterseite einen eigenen Schreib-/ Lesekopf *(Head)*
- Jedes *Platter* ist in konzentrische Spuren *(Tracks)* eingeteilt
- Ein und dieselbe Spur über alle *Platter* hinweg ergibt einen Zylinder



*[**Hennessy/Patterson,** Computer Architecture, 3. Auflage]*

Einteilung der Platten in Sektoren, Spuren und Zylinder (2)

- Jede Spur ist in Sektoren eingeteilt
- Ein Datum kann eindeutig auf einer Festplatte über die CHS-Information *(Cylinder, Head, Sector)* lokalisiert werden



*[**Hennessy/Patterson,** Computer Architecture, 3. Auflage]*

Daten einiger Plattenlaufwerke (2011)

	Seagate Barracuda XT.	Seagate Momentus Thin	Toshiba MK1214GAH
Durchmesser ['"]	3,5	2,5	1,8
Kapazität [GB]	2000	320	320
Umdrehungen [RPM]	7200	7200	5400
Mittl. Latenz [ms]	4,17	4,17	5,56
Average seek [ms]	8,5 read/9,5 write	11	15
Übertragungsrate	Max. 600 MB/s, 138 MB/s dauerhaft	300 MB/s	300 MB/s
Leistung [W] *passiv/aktiv*	6,39/9,23	0,66/1,6	0,40/1,3
Puffer [MB]	64	16	16
Gb/Quadratzoll	347	425	?
Schock-Toleranz [Betrieb/außer Betrieb]	86 G/300 G	350 G/1000 G	500 G/1500 G
Interface	SATA 6 Gb/s	SATA, 3 Gb/s	SATA, 3 Gb/s

Motivation (1)

Filesysteme speichern Daten und Programme persistent in Dateien

- Betriebssystemabstraktion zur Nutzung von Hintergrundspeichern (z.B. Festplatten, Flash-Speicher, CD-ROMs, DVDs, Bandlaufwerke, ...)
  - Benutzer muss sich nicht um die Ansteuerung verschiedener Speichermedien kümmern
  - Einheitliche Sicht auf den Sekundärspeicher

Filesysteme bestehen aus

- Dateien *(files)*
- Verzeichnissen *(directories)*
- Partitionen *(partitions)*

Motivation (2)

Datei

– Speichert Daten oder Programme

Verzeichnis

– Fasst Dateien (und Verzeichnisse) zusammen

Partitionen

– Eine Menge von Verzeichnissen und deren Dateien
– Dient zum physischen oder logischen Trennen von Dateimengen
  – *physisch:* Festplatte, Diskette
  – *logisch:* Teilbereich auf Festplatte oder CD, Zusammenfassung mehrerer Teilbereiche

Dateien (1)

Kleinste Einheit, in der etwas auf den Hintergrundspeicher geschrieben werden kann.

Typische Attribute

- *Name* – Symbolischer Name, vom Benutzer les- und interpretierbar
  - z.B. **`AUTOEXEC.BAT`**
- *Typ* – Für Dateisysteme, die verschiedene Dateitypen unterscheiden
  - z.B. sequentielle Datei, zeichenorientierte Datei
- *Ortsinformation* – Wo werden die Daten physisch gespeichert?
  - Gerätenummer, Nummern der Plattenblöcke
- *Größe* – Länge der Datei in Größeneinheiten (z.B. Bytes, Blöcke)
  - Steht in engem Zusammenhang mit der Ortsinformation
  - Wird zum Prüfen der Dateigrenzen z.B. beim Lesen benötigt

Dateien (2)

Typische Attribute (fortges.)

- Zeitstempel – z.B. Zeit und Datum der Erstellung, letzten Änderung
  - Unterstützt Backup, Entwicklungswerkzeuge, Benutzerüberwachung
- Rechte – Zugriffsrechte, z.B. Lese-, Schreibberechtigung
  - z.B. nur für den Eigentümer schreibbar, für alle anderen nur lesbar
- Eigentümer – Identifikation des Eigentümers
  - Eventuell eng mit den Rechten verknüpft
  - Zuordnung beim *Accounting* (Abrechnung des Plattenplatzes)

Typische Operationen

- Öffnen *(open)*, Schließen *(close)*, Erzeugen *(create)*, Löschen *(delete)*
- Lesen *(read)*, Schreiben *(write)*
- Positionieren des Schreib-, Lesezeigers *(seek)*

Verzeichnisse (1)

Ein Verzeichnis gruppiert Dateien und evtl. andere Verzeichnisse

Gruppierungsalternativen:

- Verknüpfung mit der Benennung
  - Verzeichnis enthält Namen und Verweise auf Dateien und andere Verzeichnisse, z.B. UNIX, MS-DOS
- Gruppierung über Bedingung
  - Verzeichnis enthält Namen und Verweise auf Dateien, die einer bestimmten Bedingung gehorchen (z.B. gleiche Gruppennummer in CP/M, eigenschaftsorientierte und dynamische Gruppierung in BeOS-BFS)

Verzeichnis ermöglicht das Auffinden von Dateien

- Vermittlung zwischen externer/interner Bezeichnung (Filename – Blöcke)

Verzeichnisse (2)

Operationen auf Verzeichnissen

- Öffnen, Schließen, Erzeugen, Löschen
- Lesen
  - Meist eintragsweises Lesen des Verzeichnisinhalts
  - Schreiben erfolgt implizit über Erzeugungs- bzw. Löschoperationen

Attribute von Verzeichnissen

- Ähnlich wie bei Dateien

Roter Faden

5. Filesysteme

- Einleitung
  - Aufbau von Festplatten
  - Bestandteile von Filesystemen
  - Operationen und Attribute
- Beispiel: UNIX/Linux
- Beispiel: FAT32
- Beispiel: NTFS
- Zuverlässige Filesysteme
- Limitierung der Plattennutzung, fehlerhafte Blöcke

Beispiel: UNIX/Linux

Datei

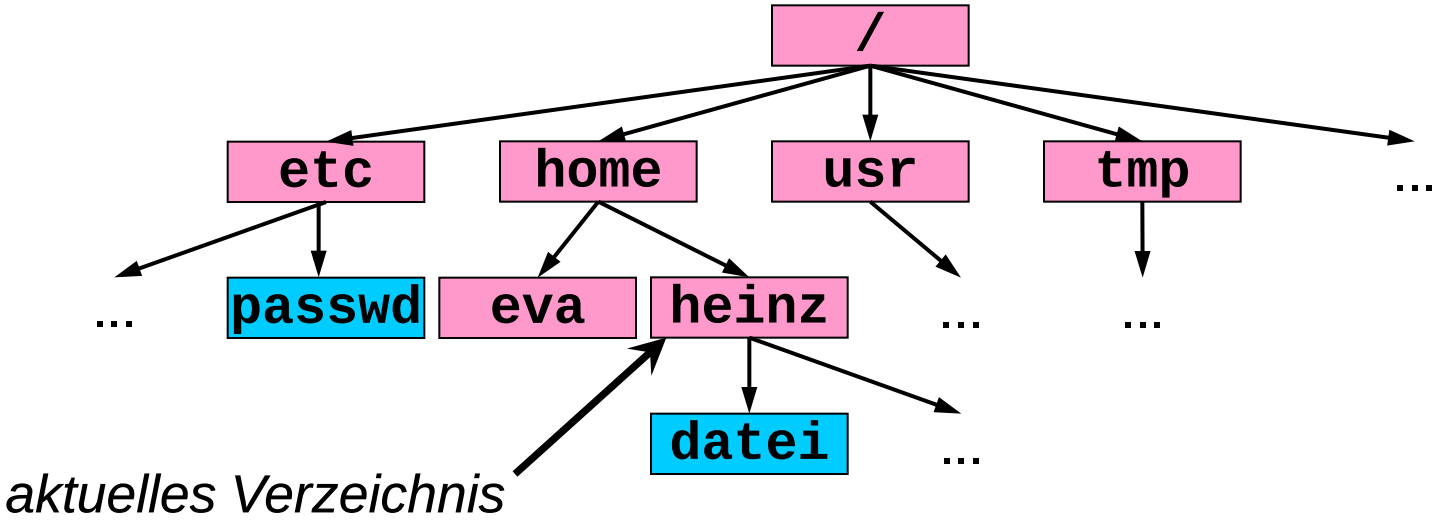
- Einfache, unstrukturierte Folge von Bytes
- Beliebiger Inhalt; für das Betriebssystem ist der Inhalt transparent
- Dynamisch erweiterbar
- Zugriffsrechte: lesbar, schreibbar, ausführbar

Verzeichnis

- Baumförmig strukturiert
  - Knoten des Baums sind Verzeichnisse
  - Blätter des Baums sind Verweise auf Dateien *(Links)*
- Jedem UNIX-Prozess ist zu jeder Zeit ein aktuelles Verzeichnis *(Current Working Directory)* zugeordnet
- Zugriffsrechte: lesbar, schreibbar, durchsuchbar, „nur“ erweiterbar

Pfadnamen (1)

Baumstruktur

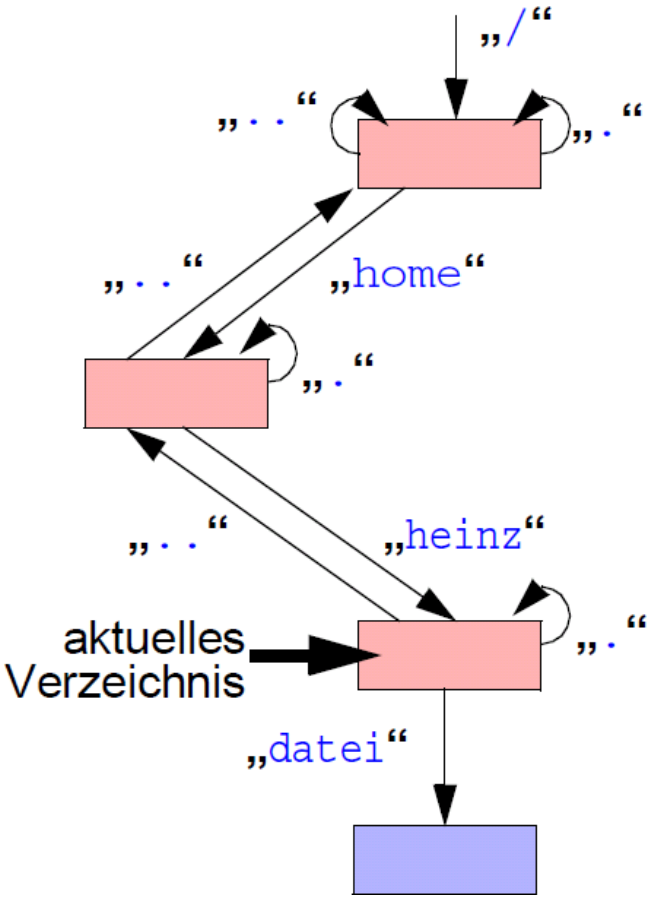


Pfade

- z.B. **`/home/heinz/datei`**, **`/tmp`**, **`datei`**
- „**`/`**“ ist Trennsymbol *(Slash)*; führender „**`/`**“ bezeichnet Wurzelverzeichnis; sonst Beginn implizit mit dem aktuellen Verzeichnis

Pfadnamen (2)

Eigentliche Baumstruktur

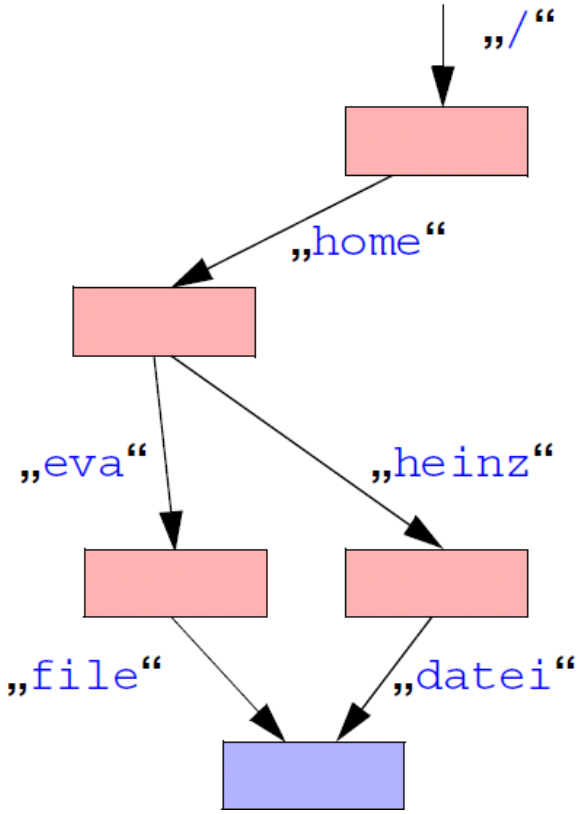


- Benannt sind nicht Dateien und Verzeichnisse, sondern die Verbindungen zwischen ihnen
  - Verzeichnisse und Dateien können auf verschiedenen Pfaden erreichbar sein, z.B. **`../heinz/datei`** und **`/home/heinz/datei`**
  - Jedes Verzeichnis enthält einen Verweis auf sich selbst („`.`“) und einen Verweis auf das darüber liegende Verzeichnis im Baum („`..`“)

Pfadnamen (3)

Links (Hard Links)

– Dateien können mehrere auf sich zeigende Verweise besitzen, sogenannte *Hard Links* (nicht jedoch Verzeichnisse)

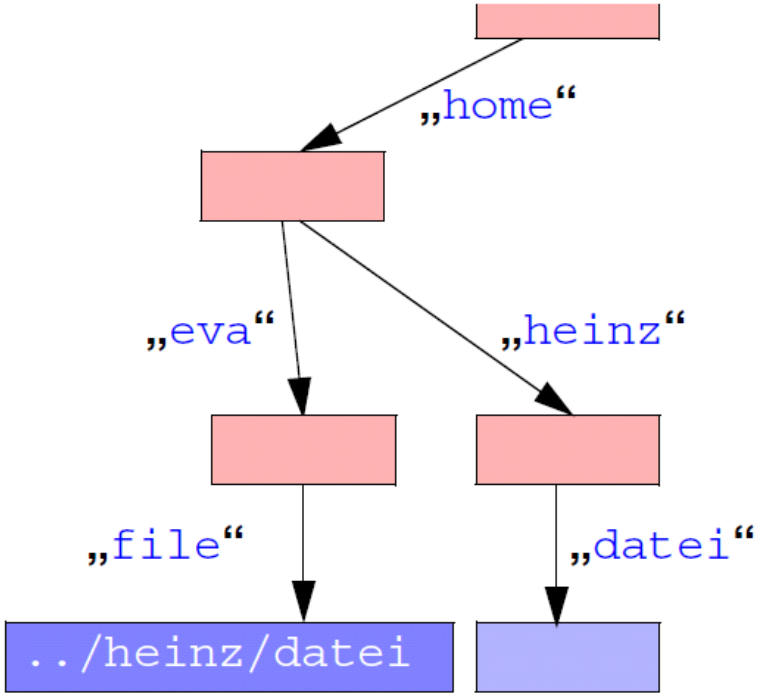


– Die Datei hat zwei Einträge in verschiedenen Verzeichnissen, die völlig gleichwertig sind:
`/home/eva/file`
`/home/heinz/datei`
– Datei wird erst gelöscht, wenn letzter *Link* gekappt wird.

Pfadnamen (4)

Symbolische Namen *(Symbolic Links)*

- Verweise auf einen anderen Pfadnamen (sowohl auf Dateien als auch Verzeichnisse)
- Symbolischer Name bleibt auch bestehen, wenn Datei oder Verzeichnis nicht mehr existiert



- Symbolischer Name enthält einen neuen Pfadnamen, der vom Filesystem interpretiert wird.

Eigentümer und Rechte

Eigentümer

- Benutzer werden durch eindeutige Nummer *(User-ID, UID)* repräsentiert
- Ein Benutzer kann einer oder mehreren Benutzergruppen angehören, die durch eine eindeutige Nummer *(Group-ID, GID)* repräsentiert werden
- Dateien/Verzeichnisse sind genau einem Benutzer/einer Gruppe zugeordnet

Rechte auf Dateien

- Lesen, Schreiben, Ausführen (nur vom Eigentümer veränderbar)
- Einzeln für den Eigentümer, für Angehörige der Gruppe und für alle anderen einstellbar

Rechte auf Verzeichnissen

- Lesen, Schreiben (Löschen/Anlegen von Dateien etc.), Durchgangsrecht

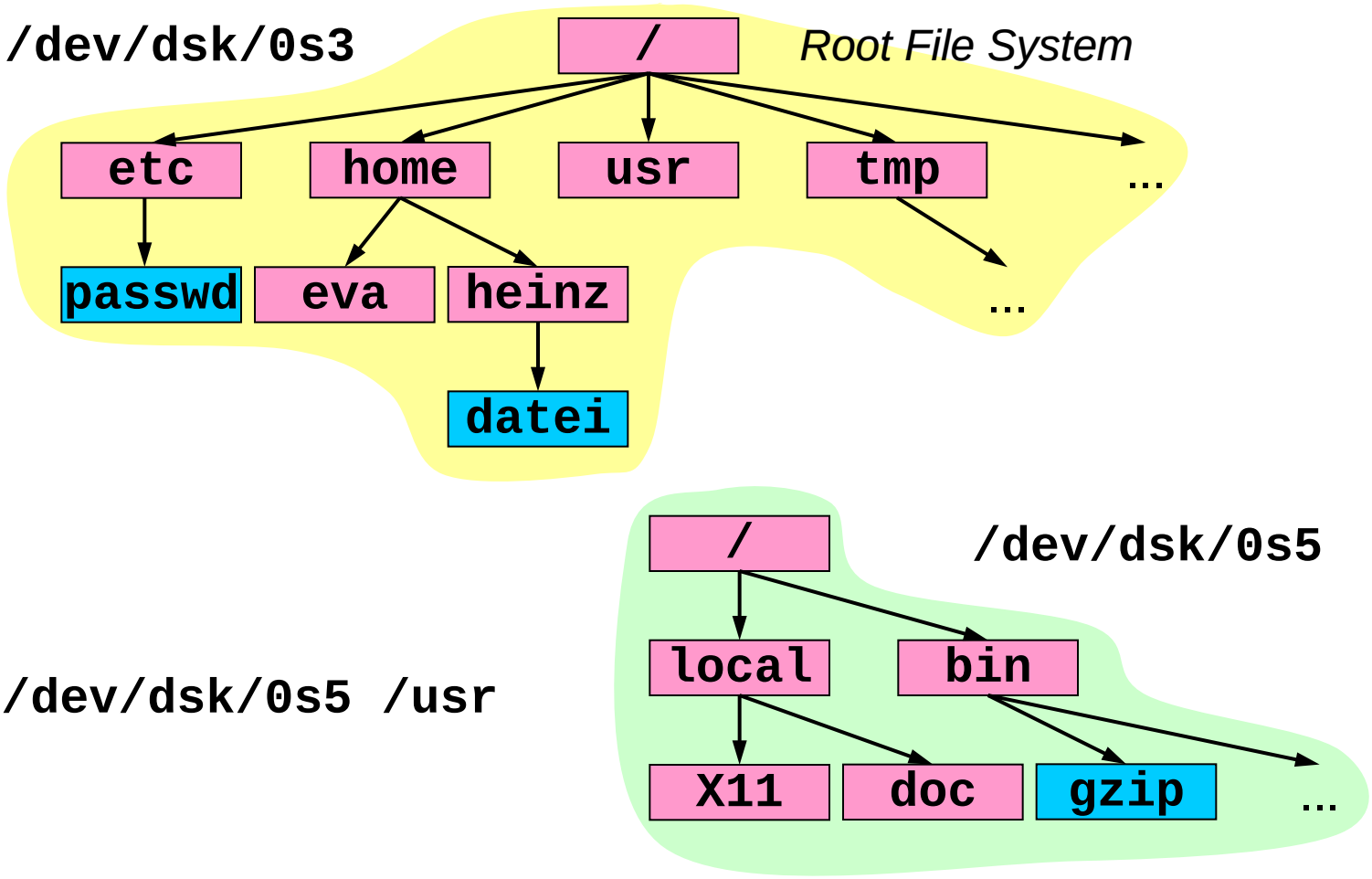
Montieren/Mounten des Dateibaums (1)

Der UNIX-Dateibaum kann aus mehreren Partitionen zusammenmontiert werden

- Partition wird Dateisystem genannt *(File System)*
- Wird durch blockorientierte Spezialdatei repräsentiert (z.B. **`/dev/dsk/0s3`**)
- Das Montieren wird *Mounten* genannt
- Ausgezeichnetes Dateisystem ist das *Root File System*, dessen Wurzelverzeichnis gleichzeitig Wurzelverzeichnis des Gesamtsystems ist
- Andere Filesysteme können mit dem Befehl **`mount`** in das bestehende System hineinmontiert werden

Montieren/Mounten des Dateibaums (2)

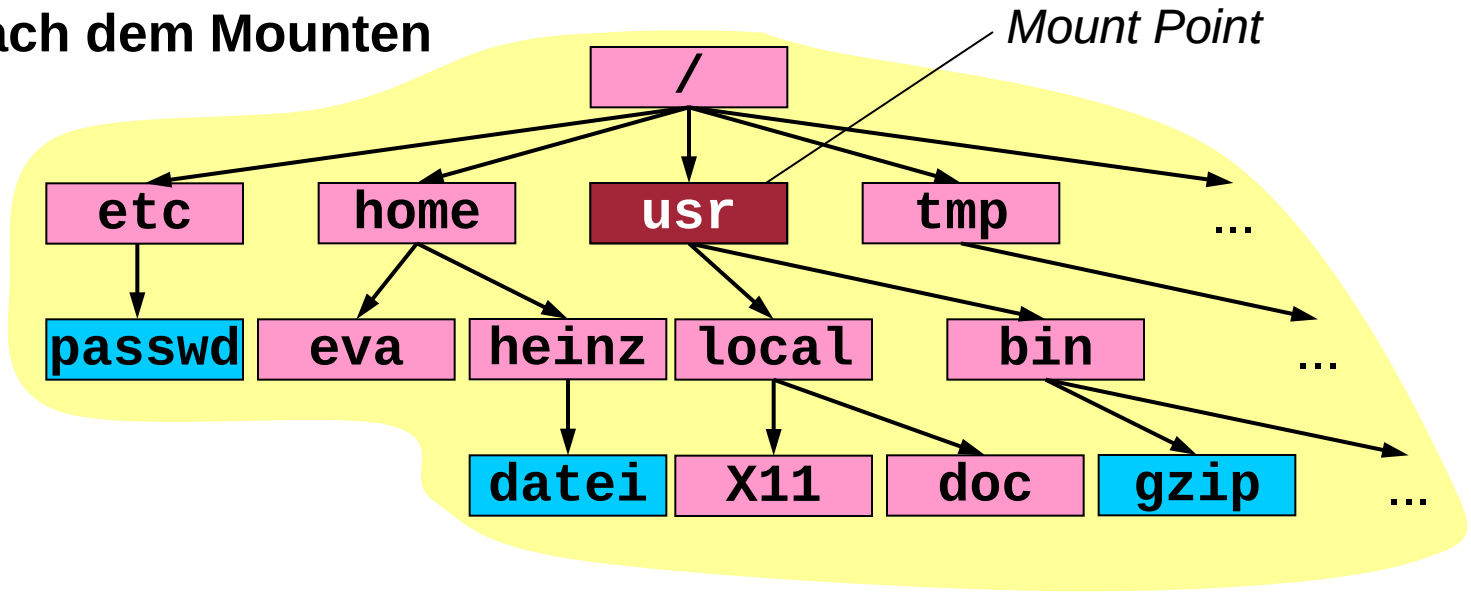
Beispiel




```
mount /dev/dsk/0s5 /usr
```

Montieren/Mounten des Dateibaums (3)

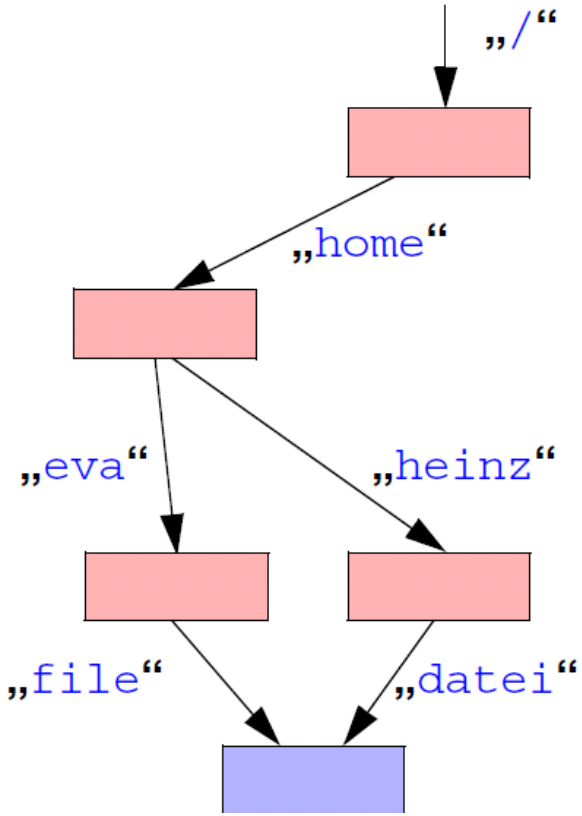
Beispiel nach dem Mounten



Inodes (1)

Attribute einer Datei und Ortsinformationen über ihren Inhalt werden in einem sogenannten *Inode* gehalten (ein Block auf Platte)

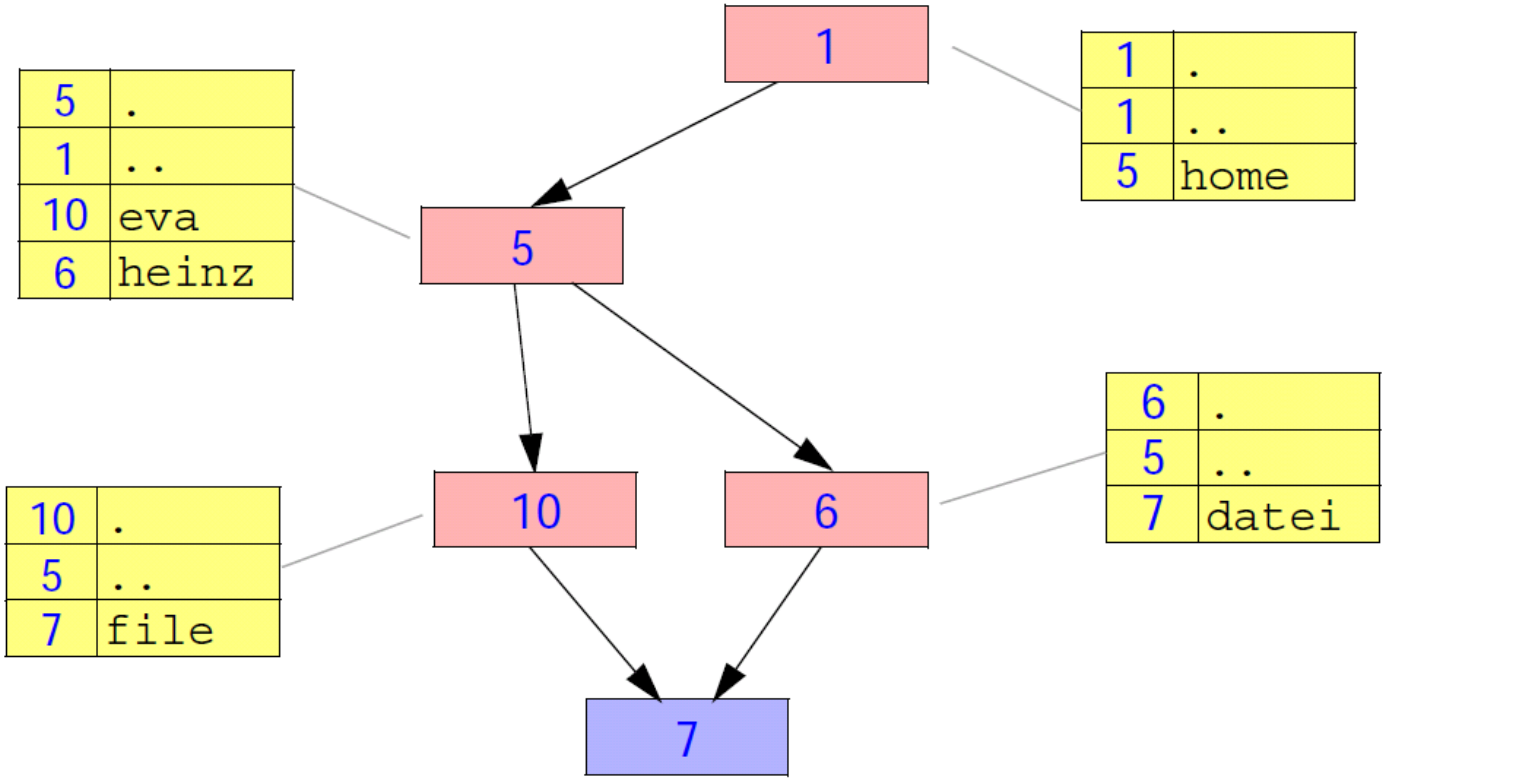
– *Inodes* pro Partition nummeriert *(Inode Number)*



logischer Dateibaum

Inodes (2)

Verzeichnisse enthalten lediglich Paare von Namen und *Inode*-Nummern



tatsächlich gespeicherter Baum

Inodes (3)

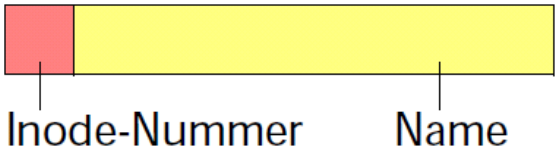
Inhalt eines *Inodes*

- *Inode*-Nummer
- Dateityp: Verzeichnis, normale Datei, Spezialdatei (z.B. Gerät)
- Eigentümer und Gruppe
- Zugriffsrechte
- Zugriffszeiten: letzte Änderung *(mtime)*, letzter Zugriff *(atime)*, letzte Änderung des Inodes *(ctime)*
- Anzahl der *Hard Links* auf den *Inode*
- Dateigröße (in Bytes)
- Adressen der Datenblöcke des Datei- oder Verzeichnisinhalts (zwölf direkte Adressen und drei indirekte)

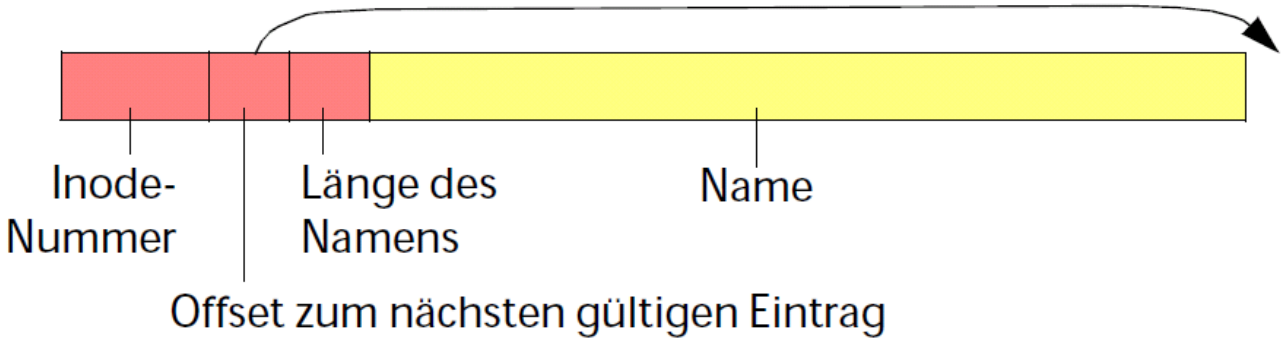
Inodes (4)

Speicherung der Verzeichniseinträge in einer speziellen Datei

- Verzeichniseinträge gleicher Länge
  - z.B. UNIX System V.3

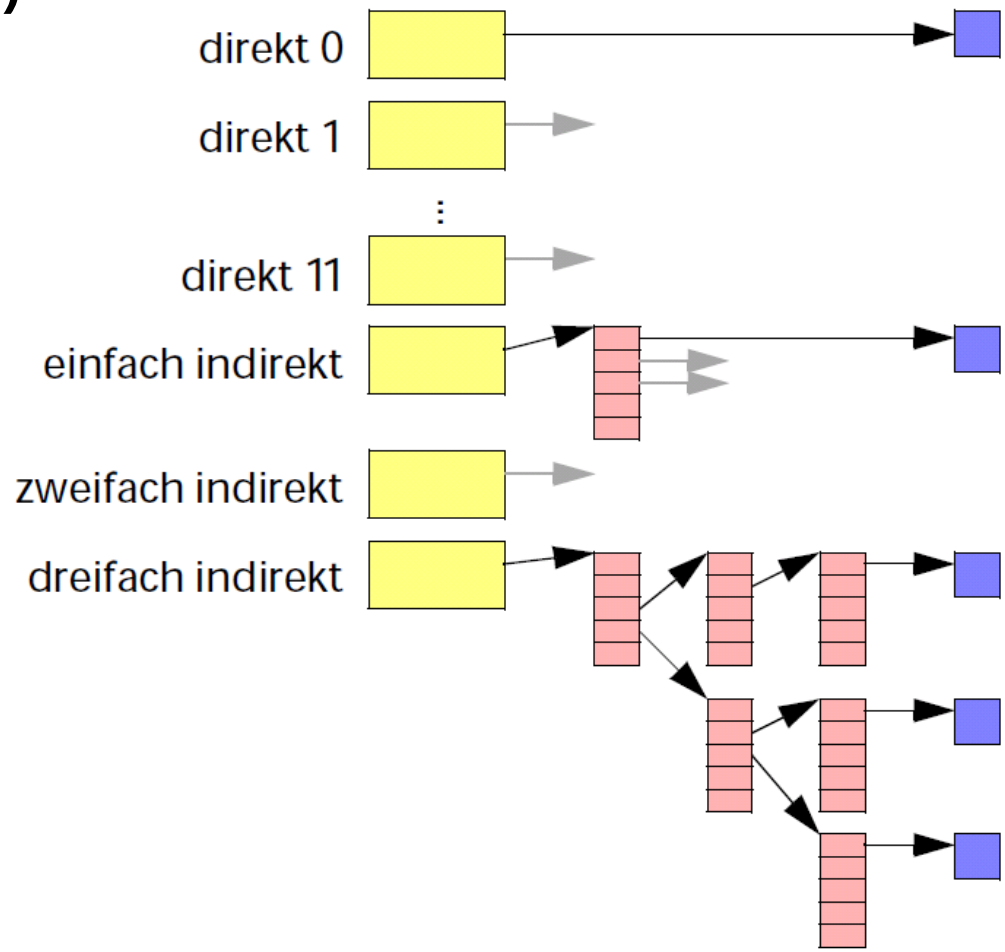


- Verzeichniseinträge variabler Länge
  - z.B. BSD 4.2, System V.4, u.a.



Inodes (5)

Adressierung der Datenblöcke innerhalb des *Inodes* (indizierte Speicherung)



Inodes (6)

Einsatz mehrerer Stufen der Indizierung

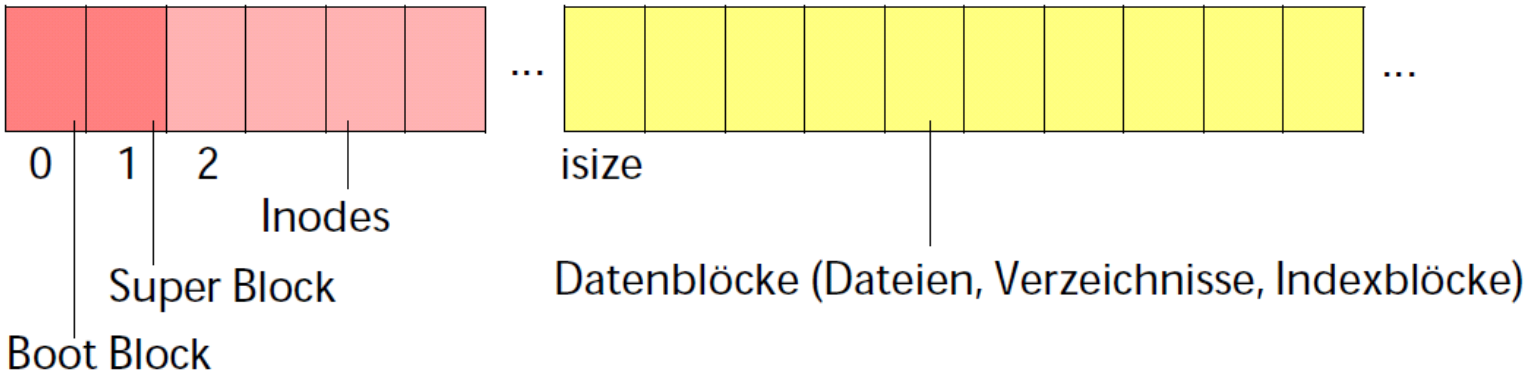
- *Inode* benötigt sowieso einen Block auf der Platte (Verschnitt unproblematisch bei kleinen Dateien)
- Durch mehrere Stufen der Indizierung auch große Dateien adressierbar
- Schnelle Positionierung des Schreib-/Lesezeigers

Nachteil

- Mehrere Blöcke müssen geladen werden (nur bei langen Dateien)

System V File System / UFS (1)

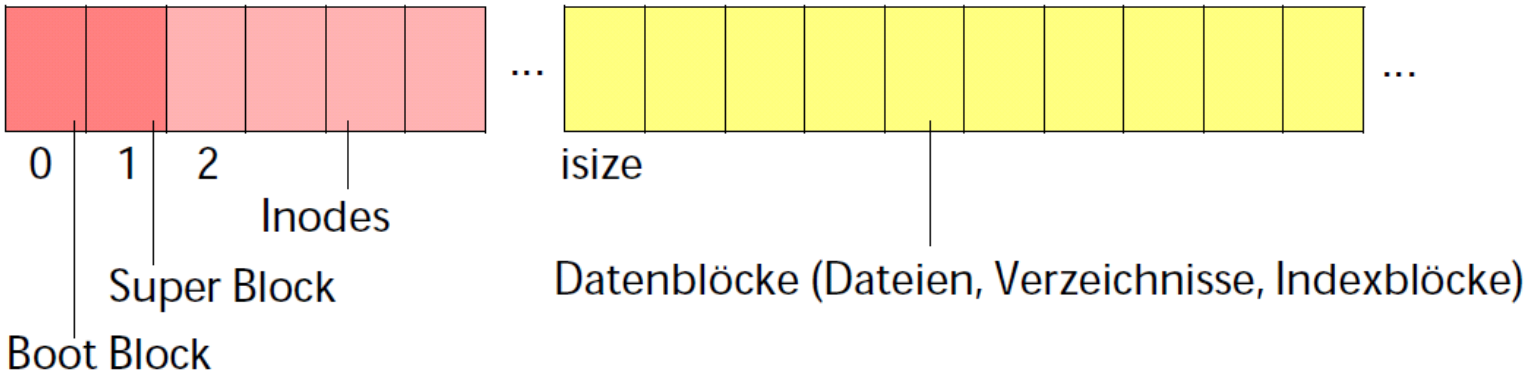
Blockorganisation



– *Boot Block* enthält Informationen zum Laden eines initialen Programms

System V File System / UFS (2)

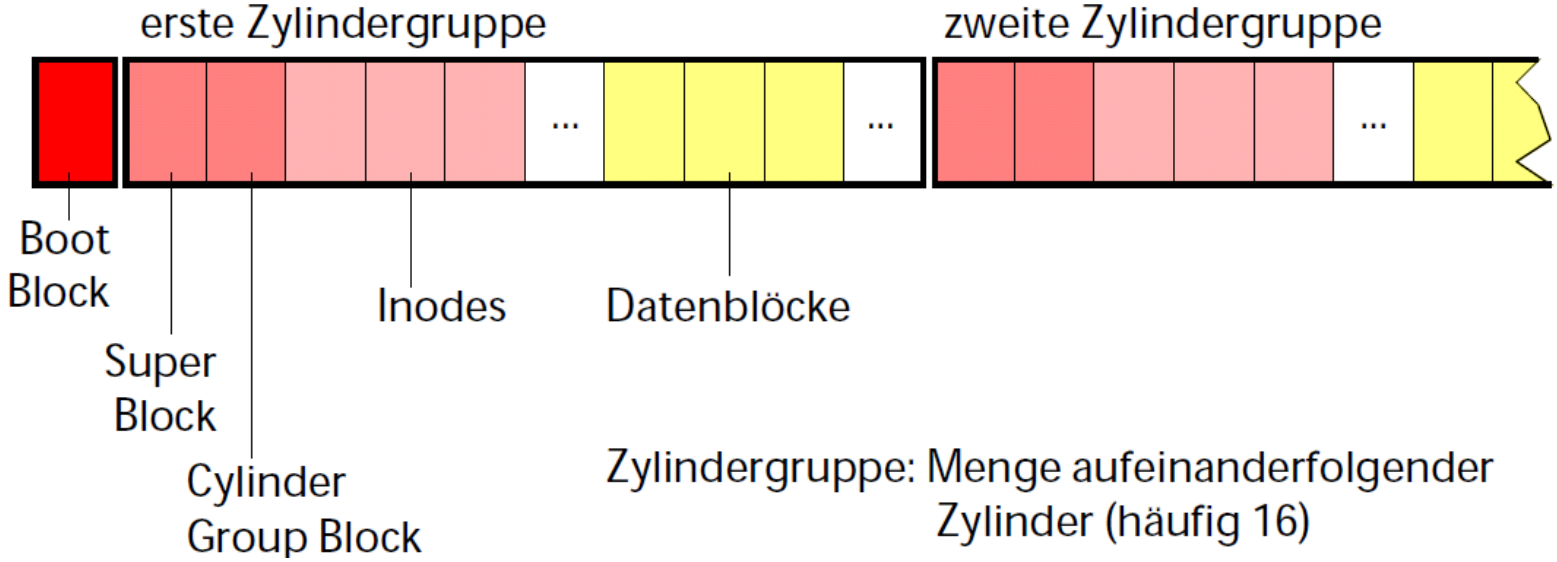
Blockorganisation



- *Super Block* enthält Verwaltungsinformation für ein Dateisystem
  - Anzahl der Blöcke, Anzahl der *Inodes*
  - Anzahl und Liste freier Blöcke und freier *Inodes*
  - Filesystem-Attribute (z.B. *clean*, *active*)
  - Filesystem-*Label* und Pfadname des letzten *Mount Points*

BSD 4.2 (Berkeley Fast File System)

Blockorganisation

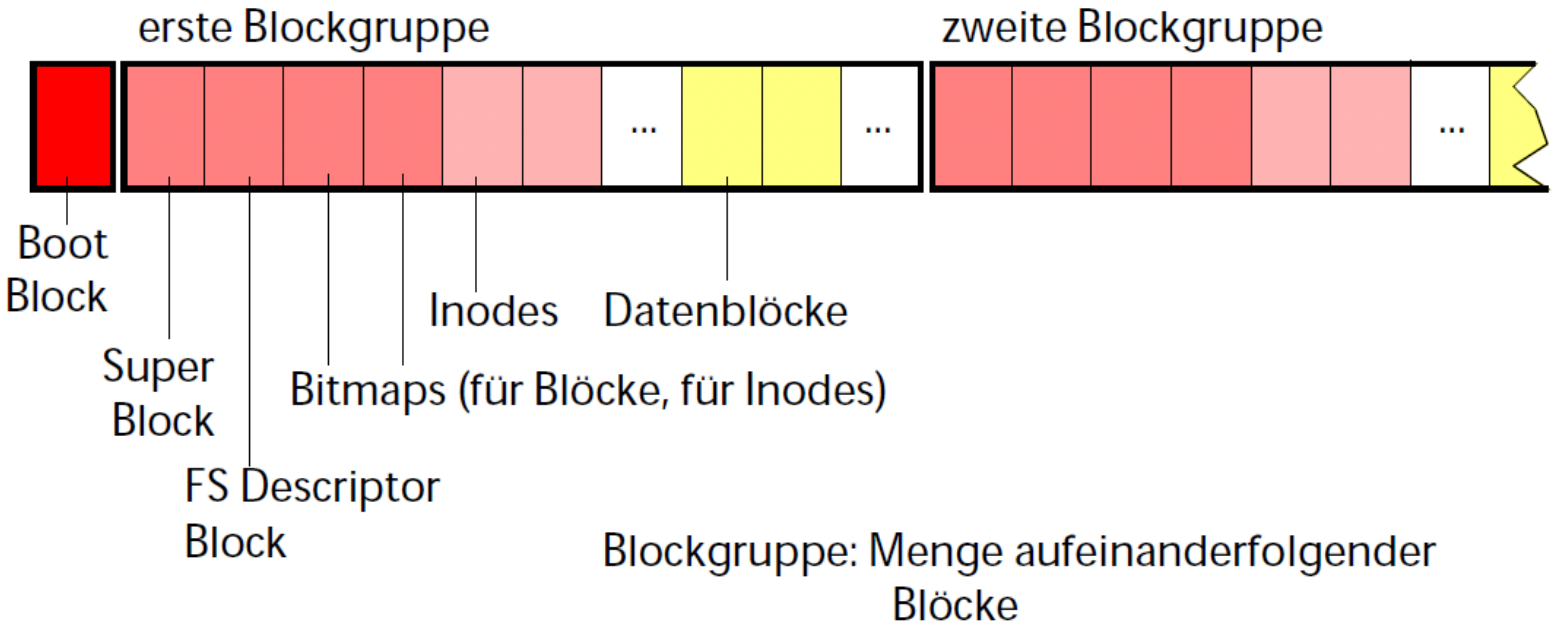


- Kopie des *Super Blocks* in jeder Zylindergruppe
- Freie *Inodes* und Datenblöcke werden im *Cylinder Group Block* gehalten
- Eine Datei wird möglichst innerhalb einer Zylindergruppe gespeichert

Vorteil: Kürzere Positionierungszeiten

Linux EXT2 File System (1)

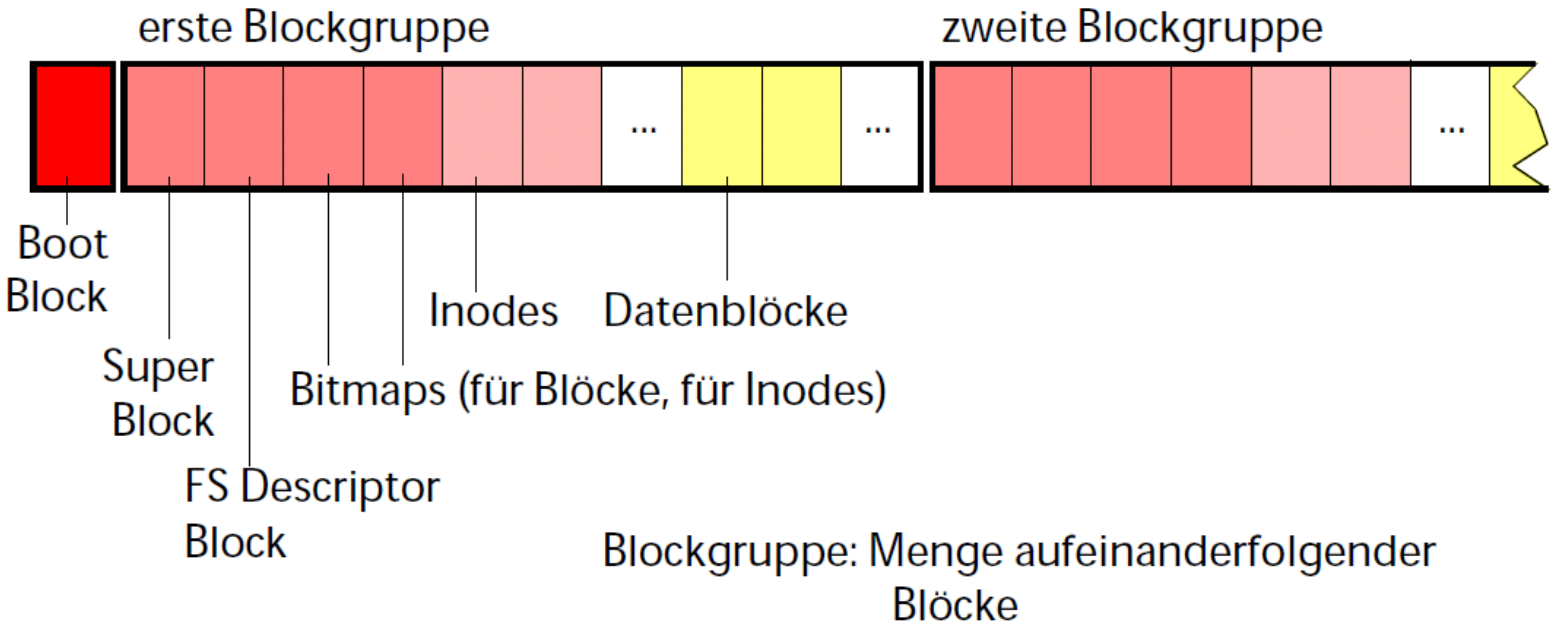
Blockorganisation



- Ähnliches Layout wie BSD FFS
- Blockgruppen unabhängig von Zylindern

Linux EXT2 File System (2)

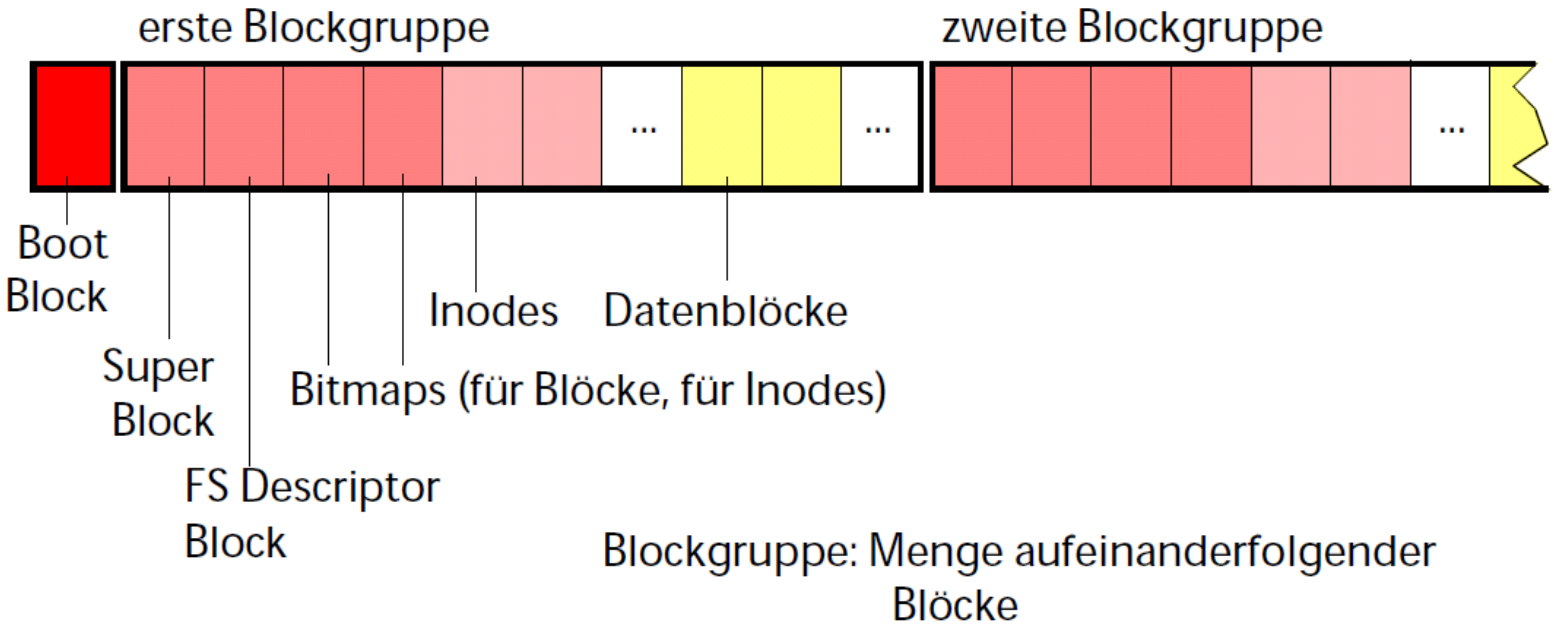
Blockorganisation



- *FS Descriptor / Block Group Descriptor*
  - Speicherort von *inode bitmap* und *block bitmap*
  - Anzahl freier Blöcke und *Inodes*

Linux EXT2 File System (3)

Blockorganisation



- *Block bitmap (Inode bitmap* analog*)*
  - Jedes Bit codiert Zustand eines Blocks in der Blockgruppe
  - 1 = „*used*“, 0 = „*free/available*“

Roter Faden

5. Filesysteme

- Einleitung
- Beispiel: UNIX/Linux
  - Baumstruktur, Pfade
  - *Links*
  - *Mounten* von Filesystemen
  - *Inodes*
  - UNIX-Filesysteme: UFS, BSD 4.2, EXT2
- Beispiel: FAT32
- Beispiel: NTFS
- Zuverlässige Filesysteme
- Limitierung der Plattennutzung, fehlerhafte Blöcke

Beispiel: FAT32 (1)

Datei

- Einfache, unstrukturierte Folge von Bytes
- Beliebiger Inhalt; für das Betriebssystem ist der Inhalt transparent
- Dynamisch erweiterbar
- Zugriffsrechte: nur lesbar, schreib- und lesbar

Partitionen heißen Laufwerke

- Sie werden durch einen Buchstaben dargestellt (z.B. `C:`)

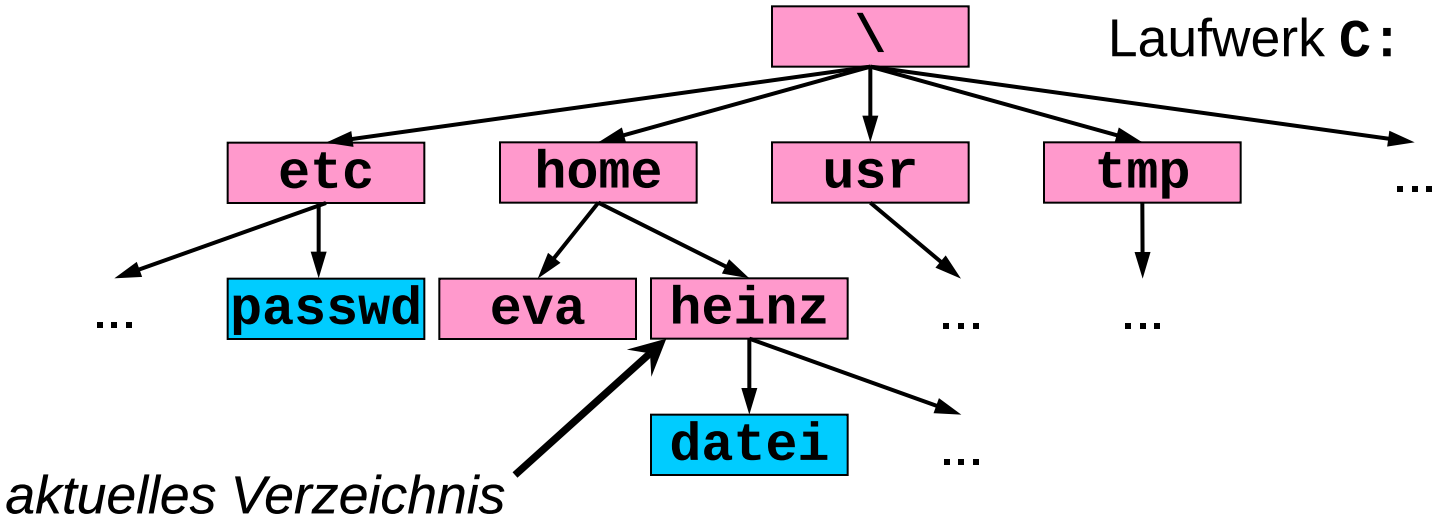
Beispiel: FAT32 (2)

Verzeichnis

- Baumförmig strukturiert
  - Knoten des Baums sind Verzeichnisse
  - Blätter des Baums sind Dateien
- Jedem Windows-Programm ist zu jeder Zeit ein aktuelles Laufwerk und ein aktuelles Verzeichnis pro Laufwerk zugeordnet
- Zugriffsrechte: nur lesbar, schreib- und lesbar

Pfadnamen (1)

Baumstruktur



Pfade

- z.B. **`C:\home\heinz\datei`**, **`\tmp`**, **`C:datei`**
- „\“ ist Trennsymbol *(Backslash)*; führender „\“ bezeichnet Wurzelverzeichnis; sonst Beginn implizit mit dem aktuellen Verzeichnis
- Beginnt der Pfad ohne Laufwerksbuchstabe, wird das aktuelle Laufwerk verwendet

Pfadnamen (2)

Namenskonvention

– Kompatibilitätsmodus: 8 Zeichen Name, 3 Zeichen Erweiterung
  (z.B. **`AUTOEXEC.BAT`**)
– Sonst: 255 Zeichen inklusive Sonderzeichen
  (z.B. **`Eigene Dateien`**)

Verzeichnisse

– Jedes Verzeichnis enthält einen Verweis auf sich selbst („`.`“) und einen Verweis auf das darüber liegende Verzeichnis im Baum („`..`“)
  (Ausnahme: Wurzelverzeichnis)
– Keine *Hard Links* oder symbolische Namen

Rechte

Rechte pro Datei und Verzeichnis

– Schreib- und lesbar – nur lesbar *(read only)*

Keine Benutzeridentifikation

– Rechte garantieren keinen Schutz, da von jedermann veränderbar

Dateien

Attribute

- Name, Dateilänge
- Attribute: versteckt *(hidden)*, archiviert *(archive)*, Systemdatei *(system)*
- Rechte
- Ortsinformation: Nummer des ersten Plattenblocks
- Zeitstempel: Erzeugung, letzter Schreib- und Lesezugriff

Struktur von FAT32

Bootsektor

– Informationen zum FAT32 Filesystem (z.B. Bytes pro Sektor, Anzahl Sektoren, Sektoren pro Spur)
– Ausführbarer x86-Code zum Bootstrapping

Reservierte Sektoren

– Optional, z.B. für OS-spezifische Erweiterungen
– FAT32: i.d.R. komplette Sicherungskopie des Bootsektors

Wurzelverzeichnis

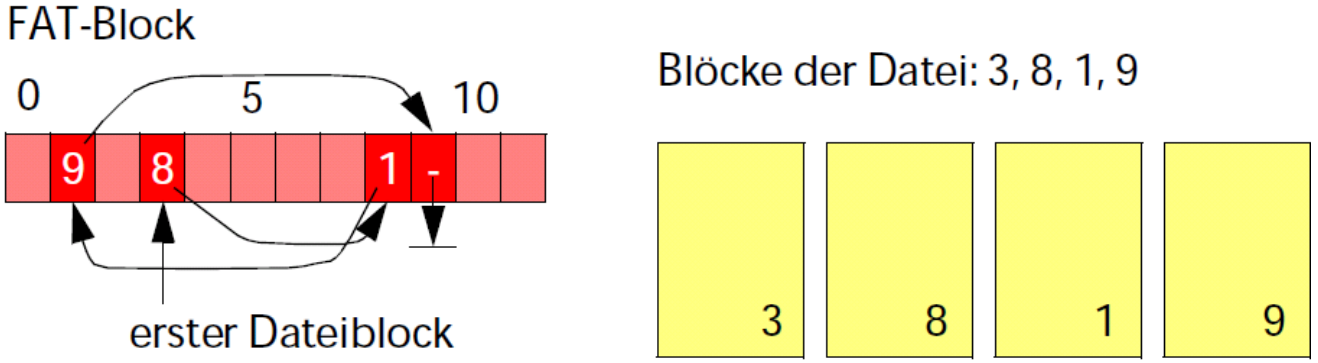
– Tabelle von Einträgen für \

Datenspeicherung (1)

Verkettung der Datenblöcke einer Datei wird in speziellen Plattenblocks gespeichert

– FAT-Ansatz *(FAT: File Allocation Table)*

– FAT enthält für jeden Datenblock einen Eintrag
– Markierung jedes Eintrags als frei, beschädigt, belegt
– FAT realisiert für jede einzelne Datei eine einfach verkettete Liste



– Mehrfache Speicherung der FAT möglich: Einschränkung der Fehleranfälligkeit

Datenspeicherung (2)

Probleme

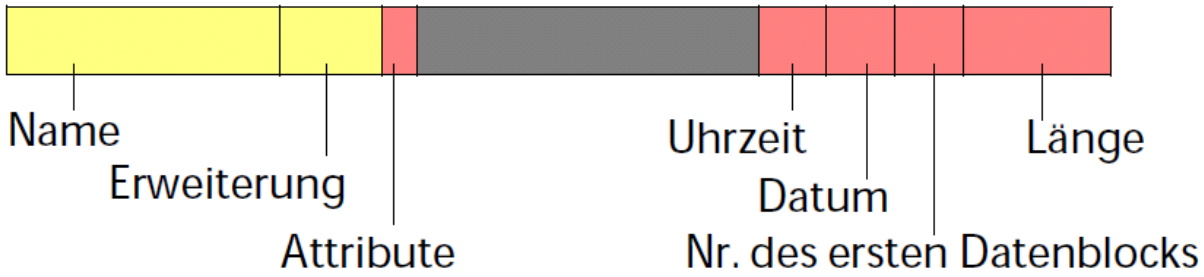
- Mindestens ein zusätzlicher Block muss geladen werden (Caching der FAT zur Effizienzsteigerung nötig)
- FAT enthält Verkettungen für alle Dateien: das Laden der FAT-Blöcke lädt auch nicht benötigte Informationen
- Aufwändige Suche nach dem zugehörigen Datenblock bei bekannter Position in der Datei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

Verzeichnisspeicherung

Verzeichniseinträge in spezieller Datei

– Für Wurzelverzeichnis Position festgelegt

Einträge gleicher Länge werden hintereinander in einer Liste gespeichert



– Für FAT32 werden mehrere Einträge zusammen verwendet, um den langen Namen aufzunehmen
  – Kurze Einträge Relikt aus MS-DOS-FAT-Filesystem

Roter Faden

5. Filesysteme

- Einleitung
- Beispiel: UNIX/Linux
- Beispiel: FAT32
  - Struktur
  - *File Allocation Table*
  - Verzeichnisspeicherung
- Beispiel: NTFS
- Zuverlässige Filesysteme
- Limitierung der Plattennutzung, fehlerhafte Blöcke

Beispiel: *New Technology File System (NTFS)*

Filesystem für Windows NT

Datei

- Einfache, unstrukturierte Folge von Bytes
- Beliebiger Inhalt; für das Betriebssystem ist der Inhalt transparent
- Dynamisch erweiterbare Dateien
- Rechte verknüpft mit NT-Benutzern und -Gruppen
- Datei kann automatisch komprimiert abgespeichert werden
- Große Dateien bis zu 16 Exabytes lang lt. NTFS-Standard, 16 Terabytes in aktuellen Implementierungen
- *Hard Links:* mehrere Einträge derselben Datei in verschiedenen Verzeichnissen möglich

Beispiel: *New Technology File System (NTFS)*

Verzeichnis

- Baumförmig strukturiert
  - Knoten des Baums sind Verzeichnisse
  - Blätter des Baums sind Dateien
- Rechte wie bei Dateien
- Alle Dateien eines Verzeichnisses automatisch komprimierbar

Partitionen heißen *Volumes*

- *Volume* wird (in der Regel) durch einen Laufwerksbuchstaben dargestellt
  z.B. **`C:`**

Rechte

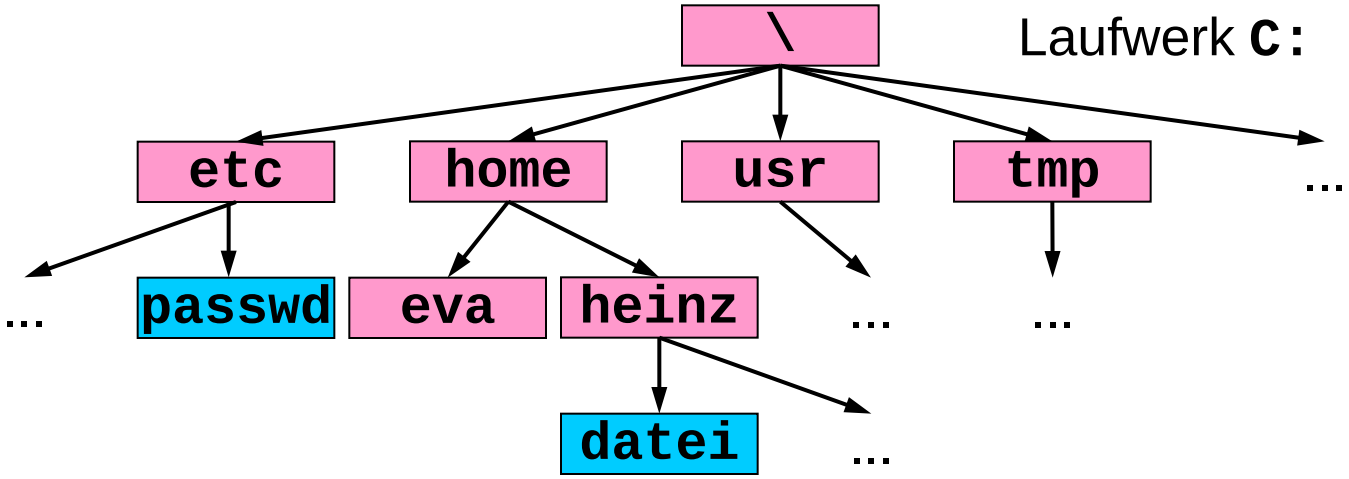
Eines der folgenden Rechte pro Benutzer oder Benutzergruppe

- *No access:* Kein Zugriff
- *List:* Anzeige von Dateien in Verzeichnissen
- *Read:* Inhalt von Dateien lesen, und *list*
- *Add:* Hinzufügen von Dateien zu einem Verzeichnis, und *list*
- *Read & Add:* Wie *read* und *add*
- *Change:* Ändern von Dateiinhalten, Löschen von Dateien und *read & add*
- *Full:* Ändern von Eigentümer und Zugriffsrechten und *change*

☞ *Access Control Lists (ACLs)*

Pfadnamen (1)

Baumstruktur



Pfade

- wie unter FAT-Filesystem
- z.B. **`C:\home\heinz\datei`**, **`\tmp`**, **`C:datei`**

Pfadnamen (2)

Baumstruktur

– 255 Zeichen inklusive Sonderzeichen
  (z.B. **`Eigene Dateien`**)
– Automatischer Kompatibilitätsmodus: 8 Zeichen Name, 3 Zeichen Erweiterung, falls „langer Name“ unter MS-DOS ungültig
  (z.B. **`AUTOEXEC.BAT`**)

Verzeichnis

– Jedes Verzeichnis enthält einen Verweis auf sich selbst („`.`“) und einen Verweis auf das darüber liegende Verzeichnis im Baum („`..`“)
– *Hard Links*
  – Für Dateien in demselben *Volume*, jedoch nicht für Verzeichnisse
  – Keine symbolischen Namen direkt im NTFS

Dateiverwaltung (1)

Basiseinheit *„Cluster“*

– 512 Bytes bis 4 Kilobytes (beim Formatieren festgelegt)
– Wird auf eine Menge aufeinanderfolgender Blöcke abgebildet
– Logische *Cluster*-Nummer als Adresse (LCN)

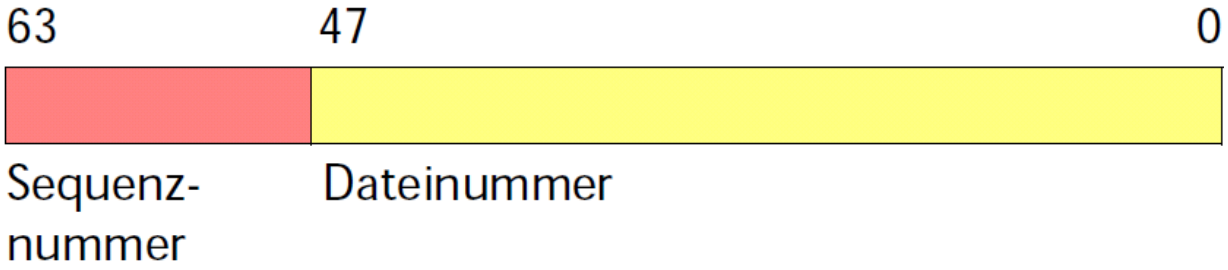
Basiseinheit „Strom“

– Jede Datei kann mehrere (Daten-) Ströme speichern (z.B. „`text.txt:extrastream`“)
– Hauptstrom wird für die eigentlichen Daten verwendet, „Nebenströme“ werden i.d.R. durch Windows nicht mehr unterstützt (z.B. Anzeigen im Explorer, Kopieren auf USB-Sticks, Anhängen an Mails, …)
– 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 ein Verzeichnis

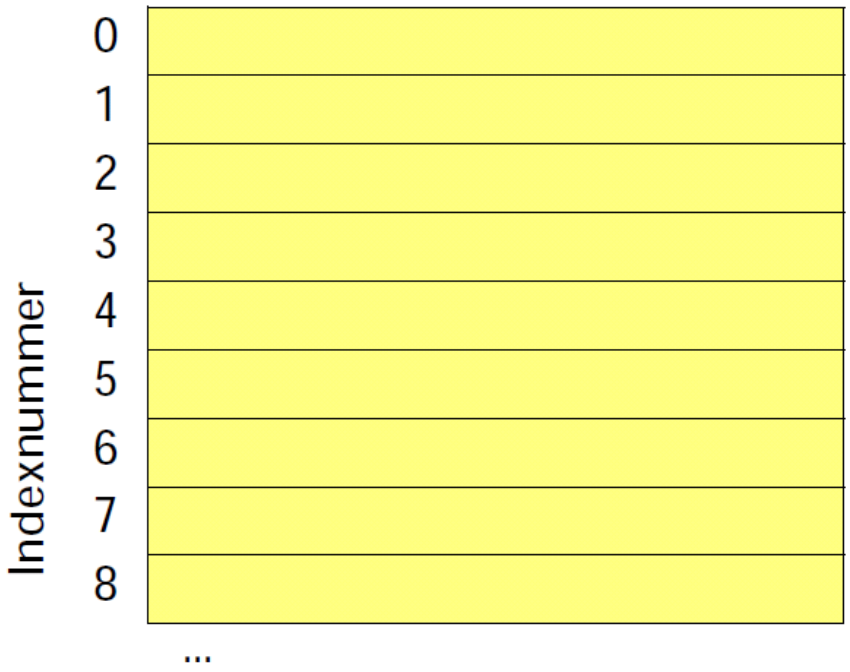


  – Dateinummer ist Index in eine globale Tabelle *(MFT: Master File Table)*
  – Dateinummer von gelöschter Datei kann wiederverwendet werden
  – Sequenznummer wird hochgezählt für jede neue Datei mit gleicher Dateinummer

Master File Table (1)

Rückgrat des gesamten Systems

– Große Tabelle mit gleich langen Elementen (1KB, 2KB oder 4KB groß, je nach *Cluster*-Größe)

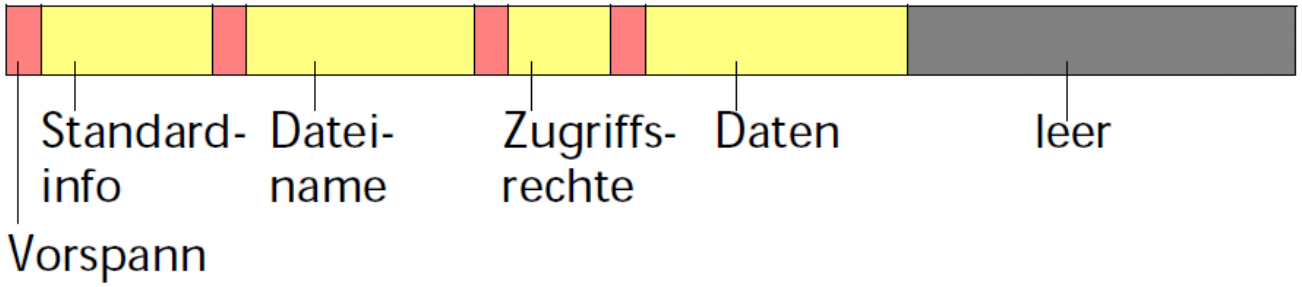


Entsprechender Eintrag für eine *File Reference* enthält Informationen über bzw. die Ströme der Datei

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

Master File Table (2)

Einträge für eine kurze Datei (mehrere Ströme)

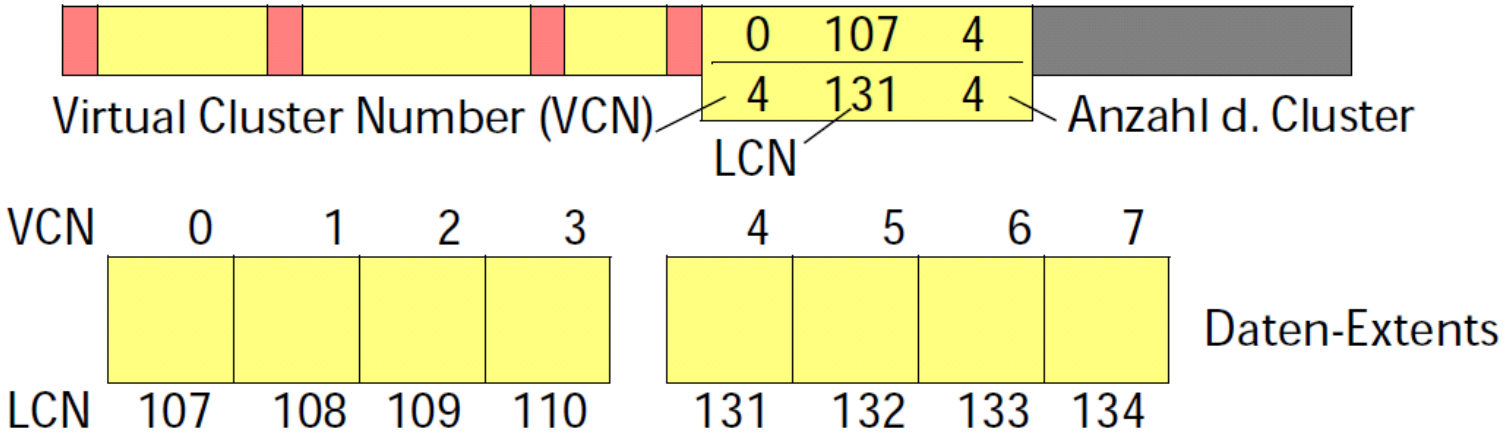


Ströme

- Standard-Information (immer in der MFT)
  - Enthält Länge, MS-DOS Attribute, Zeitstempel, Anzahl der *Hard Links*, Sequenznummer der gültigen *File Reference*
- Dateiname (immer in der MFT)
  - Kann mehrfach vorkommen (*Hard Links*, MS-DOS Name)
- Zugriffsrechte *(Security Descriptor)*
- Eigentliche Daten

Master File Table (3)

Einträge für eine längere Datei



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

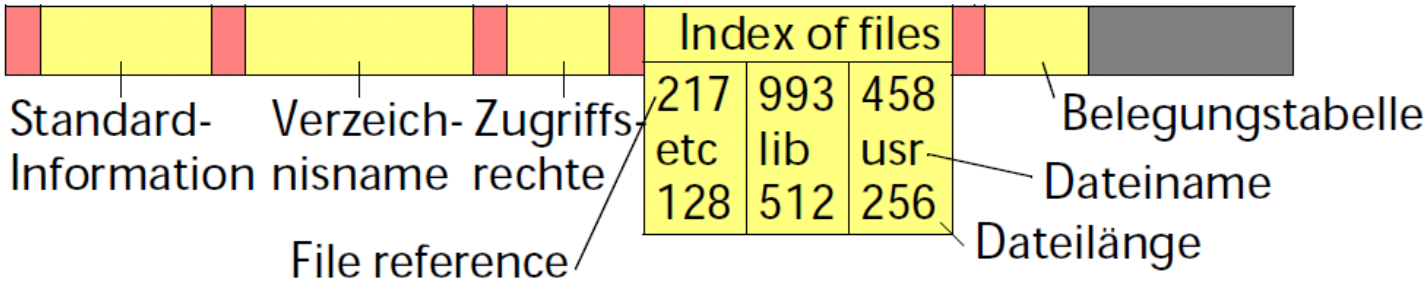
Master File Table (4)

Mögliche weitere Ströme *(Attributes)*

- Index
  - Index über einen Attributschlüssel (z.B. Dateinamen) implementiert Verzeichnis
- Indexbelegungstabelle
  - Belegung der Struktur eines Index
- Attributliste (immer in der MFT)
  - Wird benötigt, falls nicht alle Ströme in einen MFT-Eintrag passen
  - Referenzieren weitere MFT-Einträge und deren Inhalt

Master File Table (5)

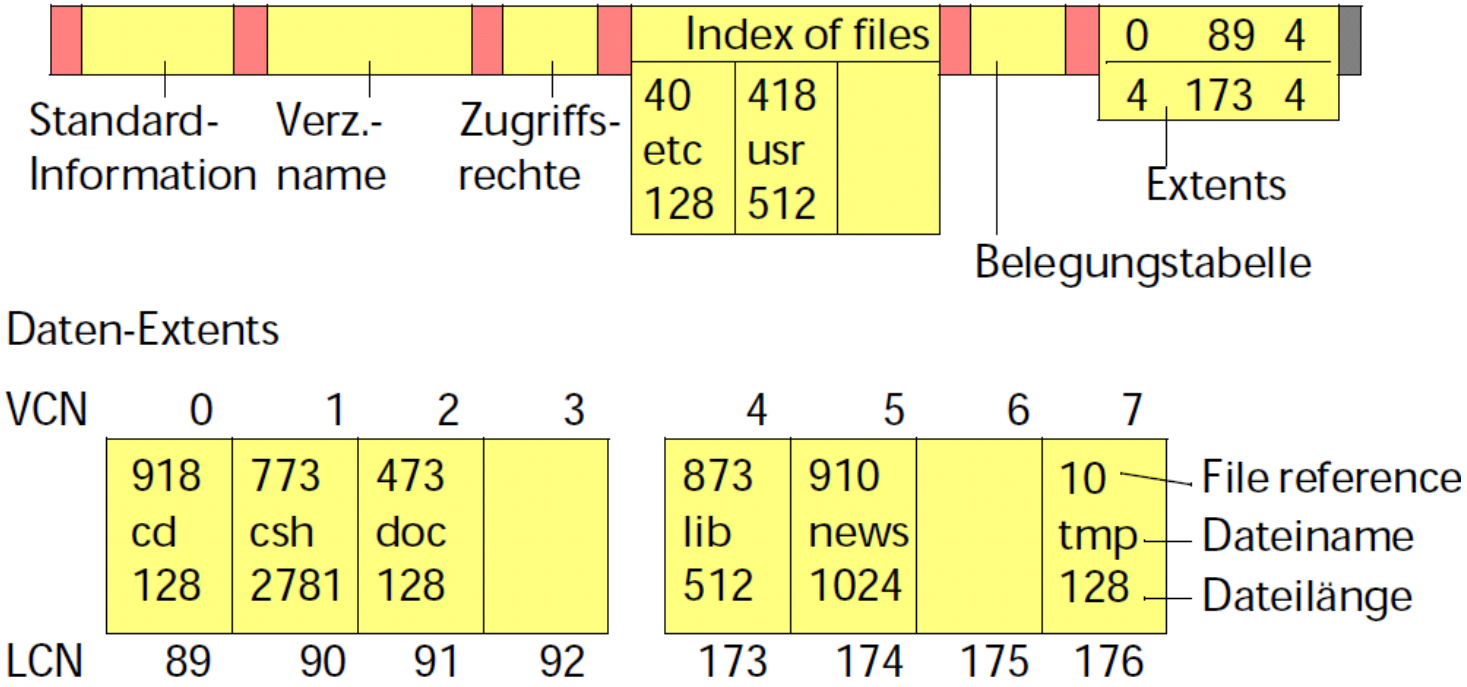
Eintrag für ein kurzes Verzeichnis



- Dateien des Verzeichnisses werden mit *File References* benannt
- Name und Länge der im Verzeichnis enthaltenen Dateien und Verzeichnisse werden auch im Index gespeichert
  (doppelter Aufwand beim Update; schnellerer Zugriff beim Listen des Verzeichnisses)

Master File Table (6)

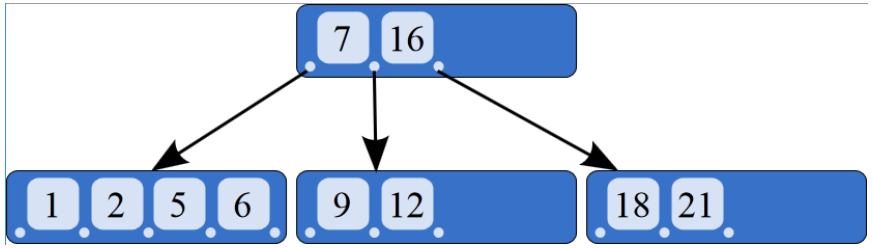
Eintrag für ein längeres Verzeichnis



– Speicherung als B^+-Baum (lexikographisch aufsteigend nach Dateinamen sortiert, schneller Zugriff)

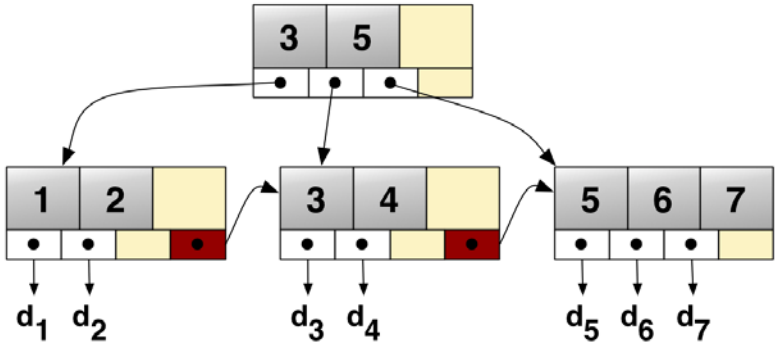
B+-Bäume

B-Bäume



- Suchbäume mit mehreren Einträgen für Daten in jedem Knoten
- Knoten können *Fan-Out* > 2 haben

B+-Bäume

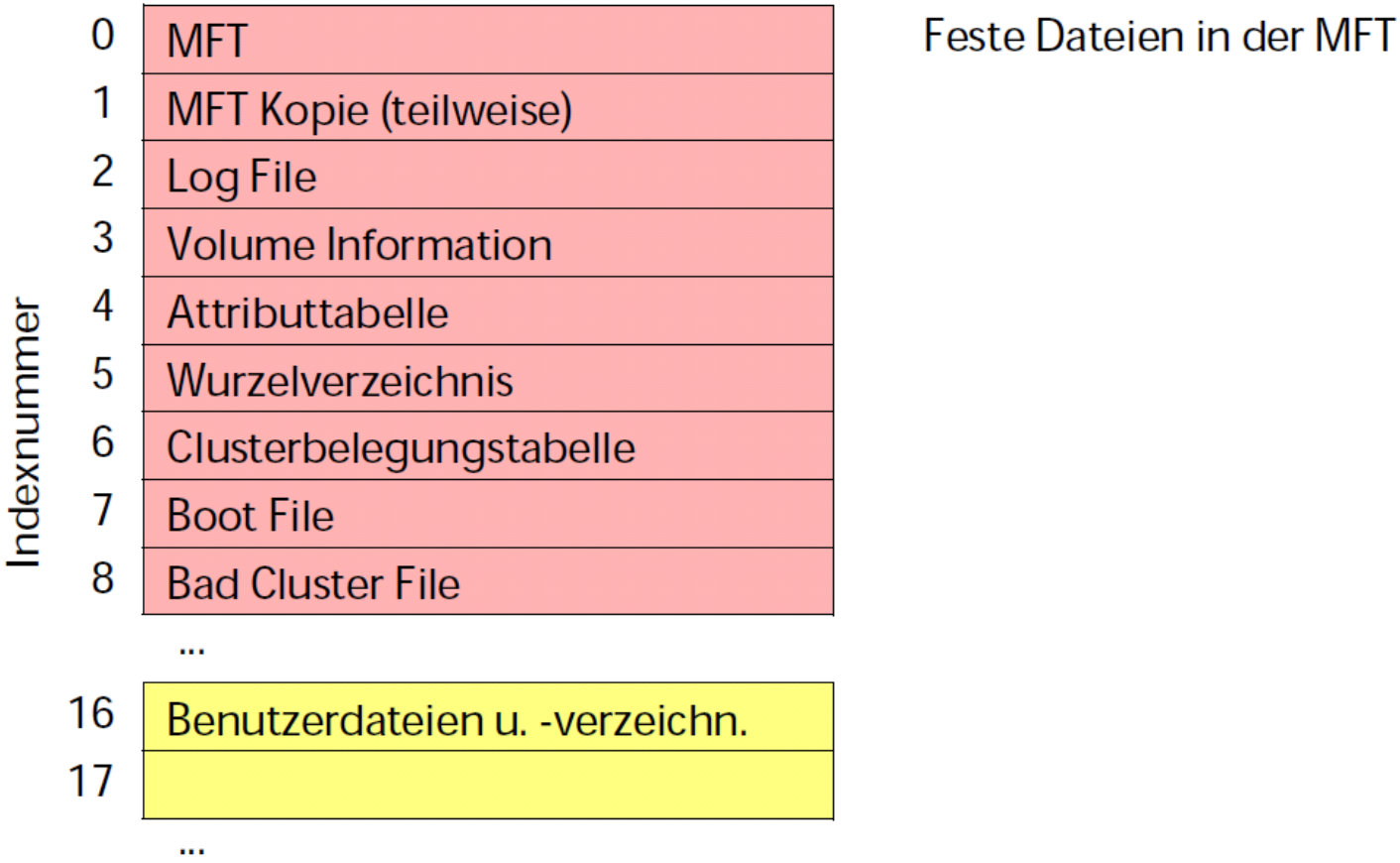


- Wie B-Bäume, nur keine effektiven Daten in inneren Knoten
- Lineare Verkettung auf Ebene von Blättern möglich

[en.wikipedia.org]

Metadaten (1)

Alle Metadaten werden in Dateien gehalten



Metadaten (2)

Bedeutung der Metadaten

- *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
- *Wurzelverzeichnis*
- *Clusterbelegungstabelle:* Bitmap für jeden *Cluster* des *Volumes*
- *Boot File:* enthält initiales Programm zum Laden, sowie die Nummern der ersten *Cluster* der MFT
- *Bad Cluster File:* enthält alle nicht lesbaren *Cluster* der Platte
  NTFS markiert automatisch alle schlechten *Cluster* und versucht, die Dateien in einen anderen *Cluster* zu retten

Weitere Features von NTFS

Kompression

– Kompression von Dateien (LZ77) inhärent in NTFS integriert
– Sollte lt. Microsoft nur auf Dateien angewendet werden, die i.d.R. regulär strukturiert sind, selten geschrieben werden und auf die sequentiell zugegriffen wird

Verschlüsselung

– *Encrypting File System (EFS)*
– Transparent für Anwender auf Ebene von Dateien und Verzeichnissen möglich
– DESX, Triple DES, AES

Roter Faden

5. Filesysteme

- Einleitung
- Beispiel: UNIX/Linux
- Beispiel: FAT32
- Beispiel: NTFS
  - Struktur *(Streams)*
  - Dateiverwaltung *(Master File Table)*
  - Metadaten
- Zuverlässige Filesysteme
- Limitierung der Plattennutzung, fehlerhafte Blöcke

Journaling

NTFS ist ein *Journaling*-Filesystem

– 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

Journaling-Filesysteme (1)

Mögliche Fehler

– Stromausfall (dummer Benutzer schaltet einfach Rechner aus)
– Systemabsturz

Auswirkungen auf das Filesystem

– Inkonsistente Metadaten
  z.B. Verzeichniseintrag 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 implizieren lange Laufzeiten der Reparaturprogramme**

Journaling-Filesysteme (2)

Zusätzlich zum Schreiben der Daten und Metadaten (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)*
- Beim Bootvorgang wird Protokolldatei mit den aktuellen Änderungen abgeglichen, damit werden Inkonsistenzen vermieden

Journaling-Filesysteme (3)

Protokollierung

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

Journaling-Filesysteme (4)

Fehlererholung

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

Journaling-Filesysteme (5)

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 Verzeichniseintrags der Datei (evtl. Freigeben eines *Extents* aus dem Index)
  - Ende der Transaktion
- Alle Vorgänge werden unter der *File Reference* im *Log* protokolliert, danach jeweils durchgeführt.
  - Protokolleinträge enthalten Informationen zum *Redo* und zum *Undo*

Journaling-Filesysteme (6)

- *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 Operationen 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 vorher können gelöscht werden)
- Ähnlich verfährt NTFS, wenn Ende des *Log Files* erreicht wird

Journaling-Filesysteme (7)

Ergebnis

- 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

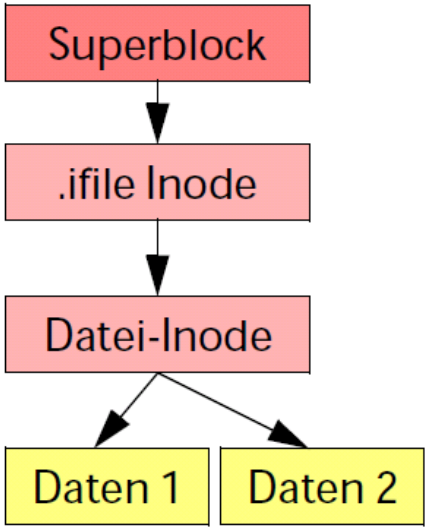
- Ineffizienter, da zusätzliches *Log File* geschrieben wird

Beispiele: NTFS, EXT3, ReiserFS

Log-Structured File Systems (1)

Alle Änderungen im Dateisystem erfolgen auf Kopien

– Der Inhalt veränderter Blöcke wird in einen neuen Block geschrieben

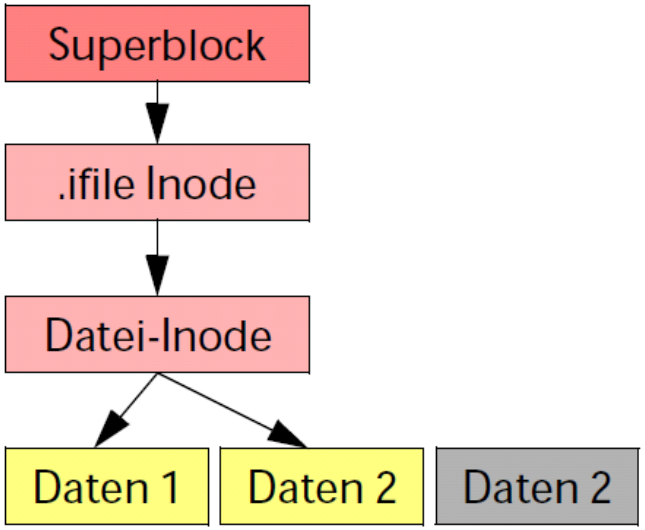


– Beispiel LinLogFS: Superblock einziger nicht ersetzter Block

Log-Structured File Systems (2)

Alle Änderungen im Dateisystem erfolgen auf Kopien

– Der Inhalt veränderter Blöcke wird in einen neuen Block geschrieben

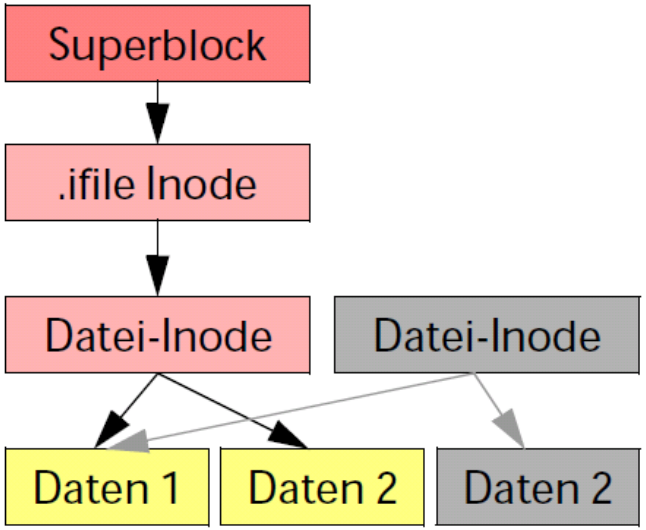


– Beispiel LinLogFS: Superblock einziger nicht ersetzter Block

Log-Structured File Systems (3)

Alle Änderungen im Dateisystem erfolgen auf Kopien

– Der Inhalt veränderter Blöcke wird in einen neuen Block geschrieben

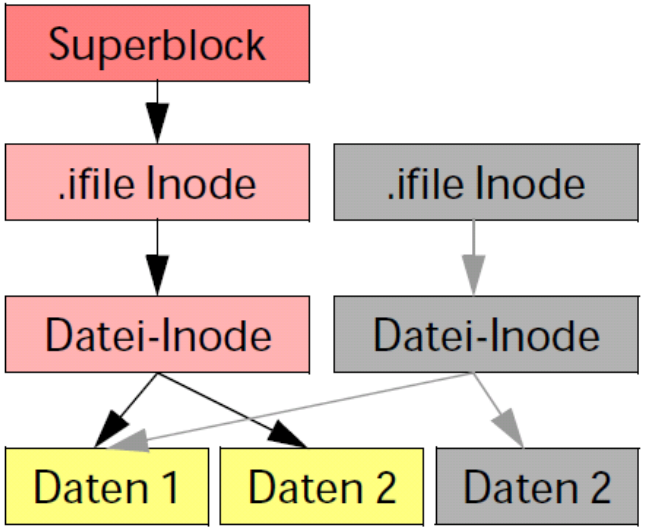


– Beispiel LinLogFS: Superblock einziger nicht ersetzter Block

Log-Structured File Systems (4)

Alle Änderungen im Dateisystem erfolgen auf Kopien

– Der Inhalt veränderter Blöcke wird in einen neuen Block geschrieben

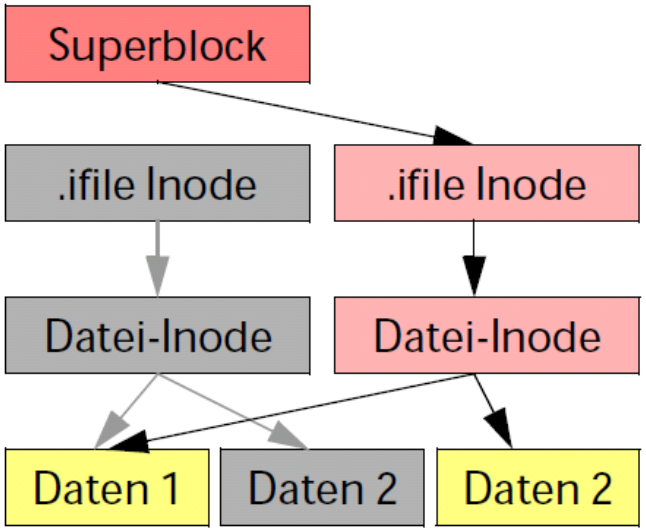


– Beispiel LinLogFS: Superblock einziger nicht ersetzter Block

Log-Structured File Systems (5)

Alle Änderungen im Dateisystem erfolgen auf Kopien

– Der Inhalt veränderter Blöcke wird in einen neuen Block geschrieben



– Beispiel LinLogFS: Superblock einziger nicht ersetzter Block

Log-Structured File Systems (6)

Vorteile

- Datenkonsistenz bei Systemausfällen
  - Eine atomare Änderung macht alle zusammengehörigen Änderungen sichtbar
- Schnappschüsse / *Checkpoints* einfach realisierbar
- Gute Schreibeffizienz
  - Alle zu schreibenden Blöcke werden kontinuierlich geschrieben

Nachteile

- Gesamtperformanz geringer

Beispiele: LinLogFS, BSD LFS, AIX XFS

Limitierung der Plattennutzung

Mehrbenutzersysteme

– Einzelnen Benutzern sollen verschieden große Kontingente zur Verfügung stehen
– Gegenseitige Beeinflussung soll vermieden werden (*Disk full* Fehlermeldung)

***Quota*-Systeme**

– Tabelle enthält maximale und augenblickliche Anzahl von Blöcken für die Dateien und Verzeichnisse eines Benutzers
– Tabelle steht auf Platte und wird vom Filesystem fortgeschrieben
– Benutzer erhält *Disk full* Meldung, wenn seine *Quota* verbraucht ist
– Üblicherweise gibt es eine weiche und eine harte Grenze (weiche Grenze kann für eine bestimmte Zeit überschritten werden)

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

– Filesystem bemerkt fehlerhafte Blöcke und markiert diese auch als belegt

Zusammenfassung (1)

Einleitung

- Festplatten: Platten, Sektoren, Spuren, Zylinder
- Bestandteile von Filesystemen: Dateien, Verzeichnisse, Partitionen
- Attribute: Name, Typ, Größe, Zeitstempel, ...
- Operationen: lesen, schreiben, ausführen, erzeugen, löschen, ...

UNIX/Linux

- Baumförmige, hierarchische Organisation
- Harte und symbolische *Links*: „Abkürzungen“ im Verzeichnisbaum
- *Inodes:* zentrale Datenstruktur zur Verwaltung von Dateien und Verzeichnissen; enthalten Dateityp, Größe, Eigentümer, Rechte, ...; 12 direkte, 3 ein- bis dreifach-indirekte Adressen auf Datenblöcke (zumindest in EXT2)
- BSD: Zylindergruppen; EXT2: Blockgruppen

Zusammenfassung (2)

FAT32

- Partitionen mit Laufwerks-Buchstaben; 8.3 Namenskonvention
- FAT: pro Datenblock ein Eintrag; Block markiert als frei, belegt, defekt; Implementierung von Files als einfach verkettete Liste von Datenblöcken
- Verzeichnisse werden in speziellen Dateien gespeichert

NTFS

- Dateien: können aus mehreren Strömen bestehen, die Nutzdaten unterschiedlicher Bedeutung enthalten
- *Master File Table:* enthält Einträge für Dateien und Verzeichnisse; *Extents:* Zusätzlicher Platz, falls MFT-Eintrag nicht für große Datei oder Verzeichnis ausreicht
- Metadaten zu NTFS-Filesystem werden wiederum in Dateien gehalten

Zusammenfassung (3)

Zuverlässige Filesysteme

– *Journaling:* alle Änderungen am Filesystem werden in Transaktionen aufgeteilt; jede Transaktion wird erst im *Log* protokolliert, bevor sie durchgeführt wird
  Beim Booten wird Ist-Zustand des Filesystems mit *Log* verglichen; entdeckte Inkonsistenzen werden per *Redo* oder *Undo* behoben
– *Log-Structured File Systems:* alle Änderungen am Dateisystem passieren auf Kopien; erst zuletzt werden Verweise von Original-Blöcken auf modifizierte Kopien umgebogen

Sonstiges

– Plattennutzung: Tabelle mit maximaler/momentaner Anzahl von Blöcken pro Benutzer; *soft & hard Quotas*
– *Bad Blocks:* Erkennung und Markierung in Hardware oder Software