Grundlagen der Betriebssysteme | F.1

Franz J. Hauck | Institut für Verteilte Systeme, Univ. Ulm

F | Dateiverwaltung

Grundlagen der Betriebssysteme

Franz J. Hauck | Institut für Verteilte Systeme, Univ. Ulm

Überblick

Überblick der Themenabschnitte

- A – Organisatorisches
- B – Zahlendarstellung und Rechnerarithmetik GdBS
- C – Aufbau eines Rechnersystems
- D – Einführung in Betriebssysteme
- E – Prozessverwaltung und Nebenläufigkeit
- F – Dateiverwaltung
- G – Speicherverwaltung
- H – Ein-, Ausgabe und Geräteverwaltung
- I – Virtualisierung BS
- J – Verklemmungen BS
- K – Rechteverwaltung

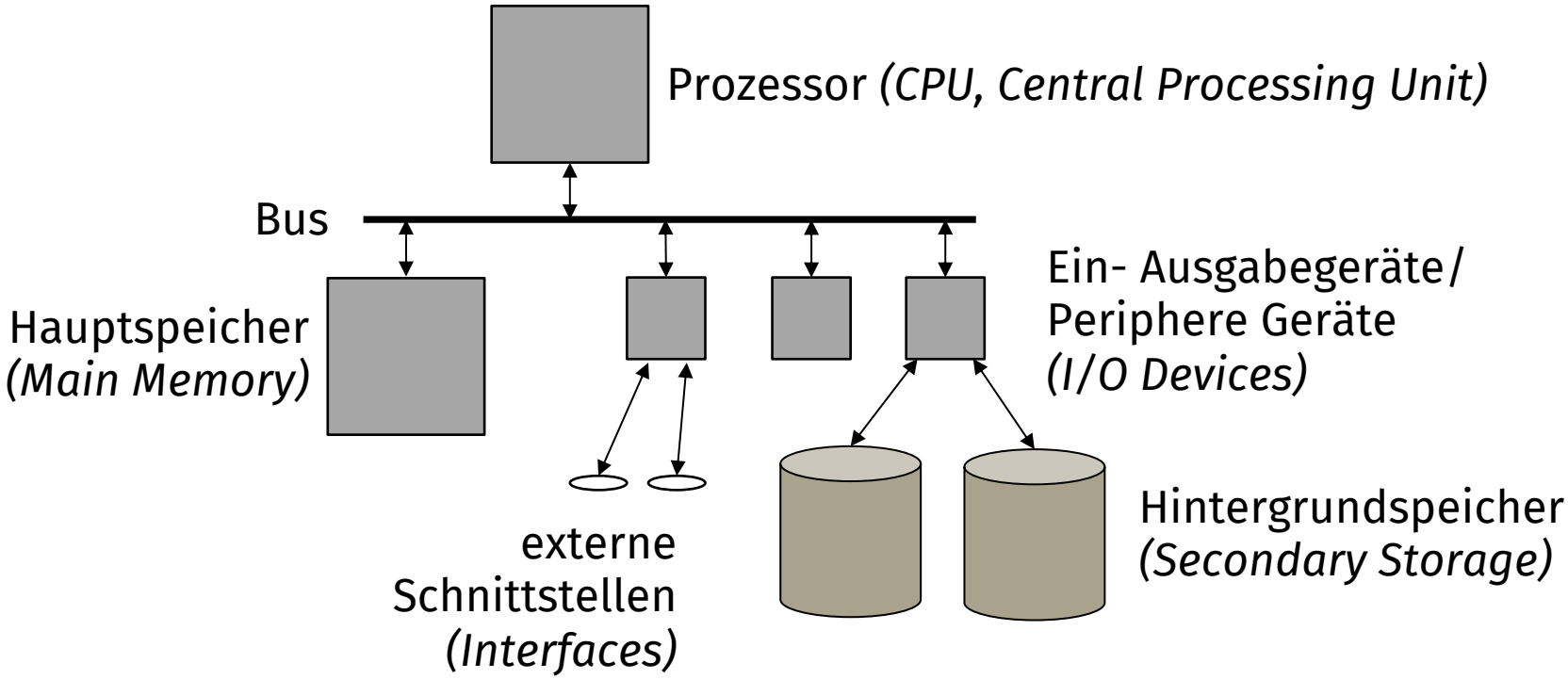
Inhaltsüberblick

Dateiverwaltung

- Einheiten und Speicherhierarchie
- Aufbau von Platten
- Anwendersicht unter Linux
  - Operationen und Attribute
- Anwendersicht unter Windows
- Unix/Linux Dateisysteme
  - Mounten, Inodes, UFS, BSD 4.2, EXT2
- Windows Dateisysteme
  - FAT32, NTFS
- Zuverlässige Dateisysteme

Einordnung

Betroffene physikalische Ressourcen



Einheiten für Speicherkapazität

Messung der Kapazität von Speichermedien (Sicht der Hersteller)

- *Kilobyte (kByte, kB)*
  1 kB = 1.000 Bytes
- *Megabyte (MB)*
  1 MB = 1.000 kB = 1.000.000 Bytes
- *Gigabyte (GB)*
  1 GB = 1.000 MB = 1.000.000.000 Bytes
- *Terabyte (TB)*
  1 TB = 1.000 GB = 1.000.000.000.000 Bytes
- *Petabyte (PB)*
  1 PB = 1.000 TB = 1.000.000.000.000.000 Bytes

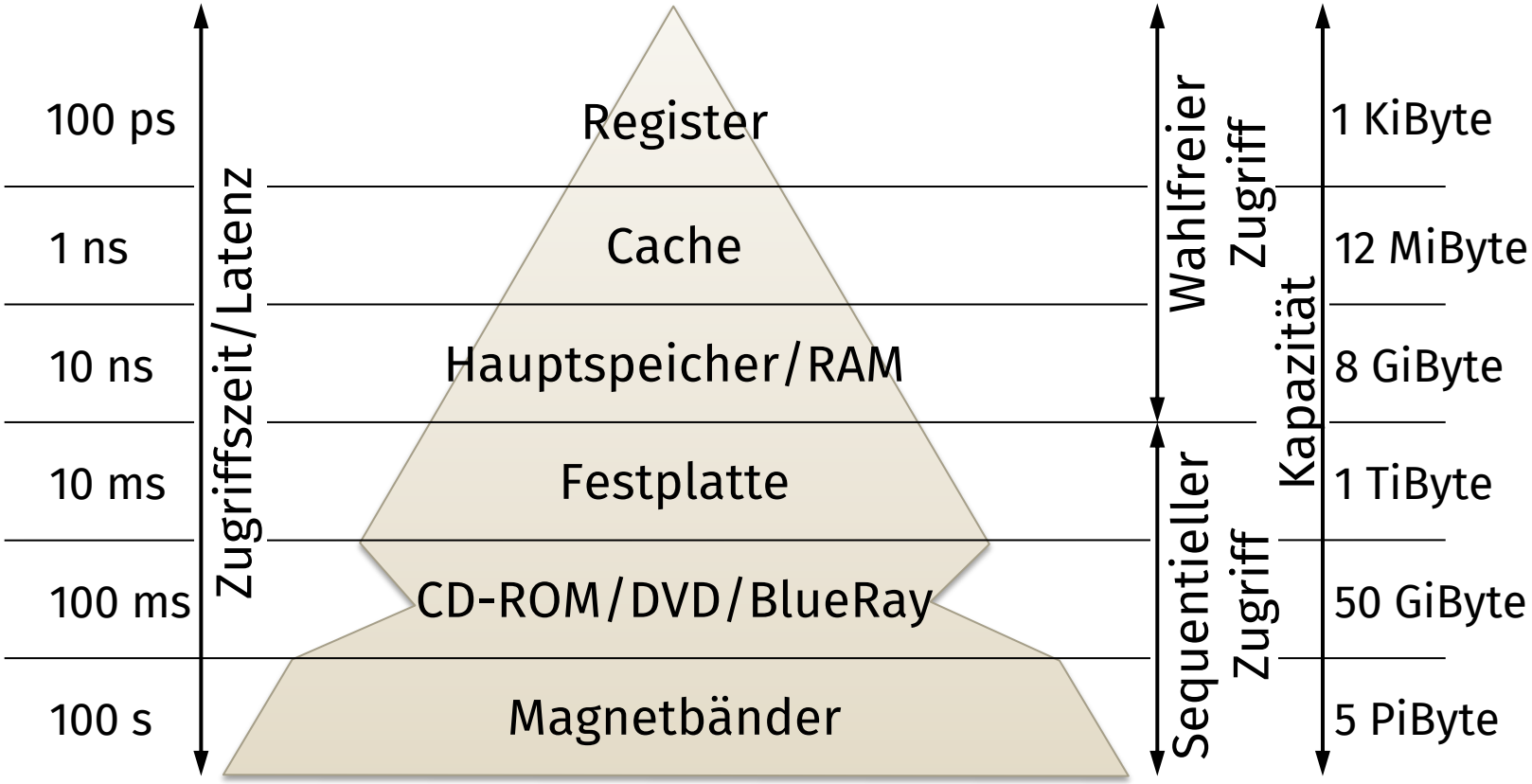
Einheiten für Speicherkapazität (2)

Messung der Kapazität von Speicherbausteinen (2er-Potenzen)

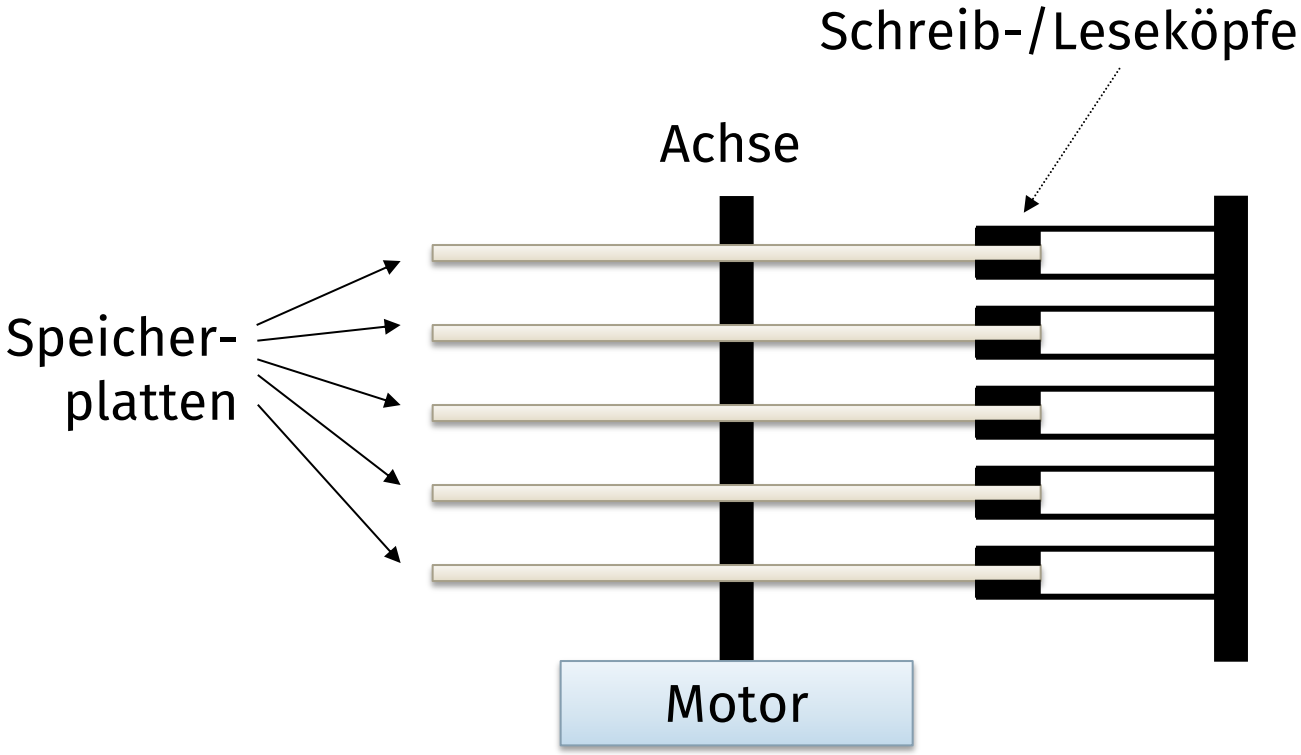
- *Kibibyte (KiB)*
  1 KiB = 1.024 Bytes = 2^{10} Bytes
- *Mebibyte (MiB)*
  1 MiB = 1.024 KiB = 2^{20} Bytes = 1.048.576 Bytes
- *Gibibyte (GiB)*
  1 GiB = 1.024 MiB = 2^{30} Bytes = 1.073.741.824 Bytes
- *Tebibyte (TiB)*
  1 TiB = 1.024 GiB = 2^{40} Bytes = 1.099.511.627.776 Bytes
- *Pebibyte (PiB)*
  1 PiB = 1.024 TiB = 2^{50} Bytes = 1.125.899.906.842.624 Bytes

Speicherhierarchie

Bis zu sechs Ebenen in modernen Systemen unterscheidbar

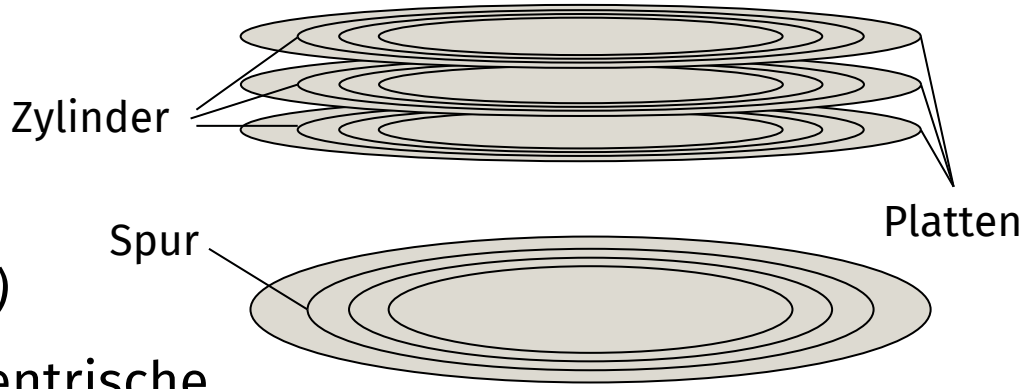


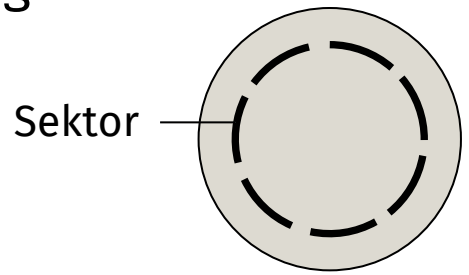
Schematischer Aufbau eines Plattenlaufwerks



Einteilung der Platten in Sektoren, Spuren und Zylinder

- Speicherplatte *(Platter)*
- hat auf Ober- und Unterseite einen eigenen Schreib-/Lesekopf *(Head)*
- Platter eingeteilt in konzentrische Spuren *(Tracks)*
- ein und dieselbe Spur über alle Platters hinweg ergibt einen Zylinder (Cylinder)
- jede Spur eingeteilt in Sektoren *(Sector)*





Benennung der Sektoren / Plattenblöcke

- eindeutige Identifikation eines Blocks
  - identifizierbar über die CHS-Information *(Cylinder, Head, Sector)*
  - heute meist durchlaufende Nummerierung (0 bis N-1)

❖ Achtung:

- Blockgröße des Betriebssystems kann Vielfaches der Sektorgröße auf der Platte sein
- typische Sektorgröße 512 B oder 4 KiB

Aufbau einer SSD

Solid State Disk – SSD

- Blöcke von Flash-Speicherbereichen
  - erhalten Daten auch ohne Stromversorgung
  - nur zwischen 1.000 und 100.000 mal beschreibbar
    - intelligente Auswahl des physischen Blocks und Zuordnung zu einem logischen Block
- Seiten von mehreren Blöcken
  - werden gemeinsam gelöscht
    - vor erneutem Schreiben muss erst gelöscht werden
  - bis zum Löschen werden veraltete Blöcke als ungültig markiert

Plattencontroller

Mikrocontroller in der Platte

- steuert mechanische Ansteuerung
  - Festplatte: Motor, Schreib-Lese-Köpfe
- organisiert Blockzuteilung
- interpretiert und führt Kommandos über Schnittstelle aus
  - z.B. SATA = Serial AT Attachment

Beispieldaten einiger Festplattenlaufwerke (2021)

	Seagate Exos X12	Western Digital WD Blue	Seagate Ultrathin HDD	Seagate Barracuda SSD
Durchmesser	3,5“	3,5“	1,8“	2,5“
Kapazität	12 TB	2 TB	500 GB	500 GB
Umdrehungen	7200 U/min	5400 U/min	5400 U/min	SSD
mittl. Zugriffszeit	4,2 ms	18,3 ms	25,0 ms	<0,1 ms
max. Übertr.Rate kont.	261 MB/s	180 MB/s	100 MB/s	560 MB/s
max. Übertr.Rate kurzfr.	6 Gb/s	6 Gb/s	600 MB/s	?
Leistung passiv/aktiv	5,4 W / 9,3 W	0,6 W / 4,1 W	0,5 W / 1,4 W	0,1W / 2,5W
Puffer	256 MB	256 MB	16 MB	?
Schocktoleranz Betrieb/außer Betrieb	70 G / 250 G	35 G / 250 G	400 G / 1000 G	1500 G
Interface	SATA 6 Gb/s	SATA 6 Gb/s	SATA 6 Gb/s	SATA 6 Gb/s

Grundlagen der Betriebssysteme | F.2

Franz J. Hauck | Institut für Verteilte Systeme, Univ. Ulm

Inhaltsüberblick

Dateiverwaltung

- Einheiten und Speicherhierarchie
- Aufbau von Platten
- Anwendersicht unter Linux
  - Operationen und Attribute
- Anwendersicht unter Windows
- Unix/Linux Dateisysteme
  - Mounten, Inodes, UFS, BSD 4.2, EXT2
- Windows Dateisysteme
  - FAT32, NTFS
- Zuverlässige Dateisysteme

Anwendersicht

Dateien

- speichern eigentliche Daten
  - unstrukturiert: Folge von Bytes
  - strukturiert: bestehend aus mehreren Datensätzen
- Zugriff
  - sequenziell: Auslesen von Beginn bis Ende
  - wahlfrei: Position in der Datei wählbar

Anwendersicht (2)

Verzeichnisse

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

Anwendersicht unter Linux

Dateien

- einfache, unstrukturierte Folge von Bytes
- beliebiger Inhalt
  - für das Betriebssystem ist der Inhalt transparent
- dynamisch erweiterbar
- wahlfreier Zugriff

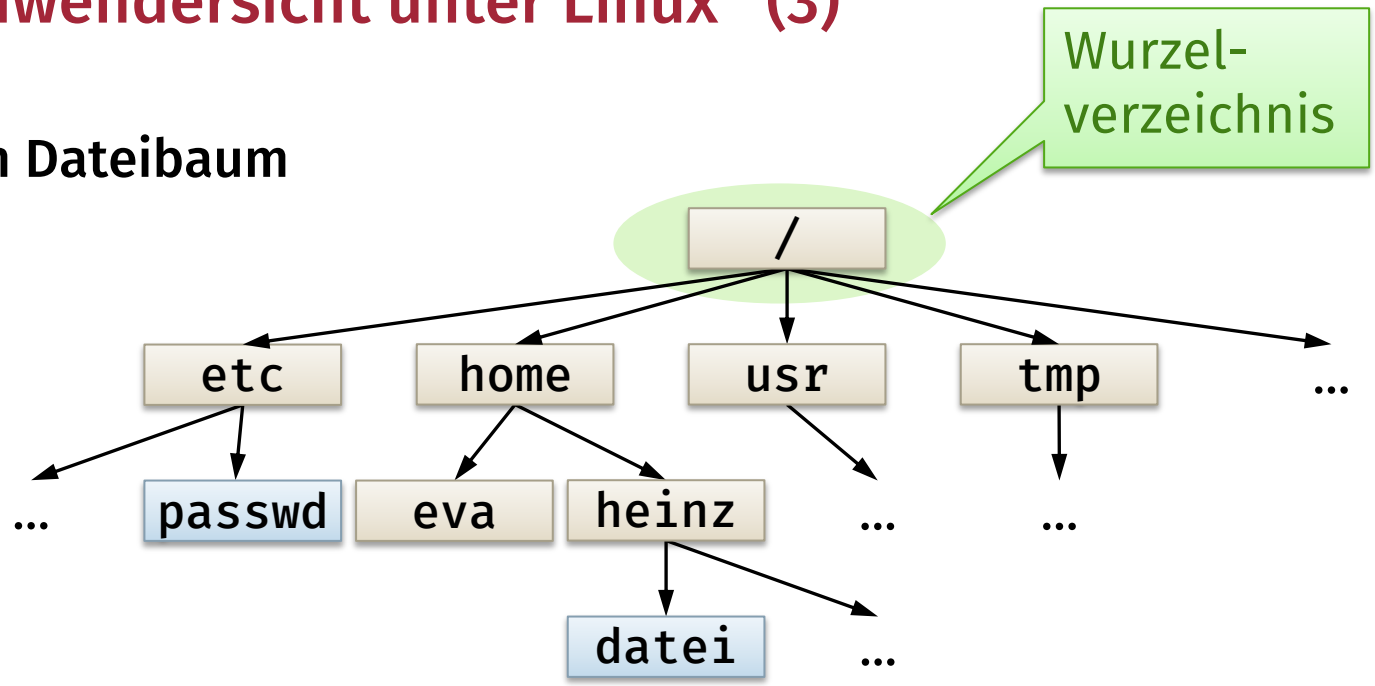
Anwendersicht unter Linux (2)

Verzeichnisse

- baumförmig strukturiert
  - Knoten des Baums sind Verzeichnisse
  - Blätter des Baums sind Verweise auf Dateien (Links)
- aktuelles Verzeichnis *(Current Working Directory)*
  - jeder Prozess hat eigenes aktuelles Verzeichnis
  - kann geändert werden

Anwendersicht unter Linux (3)

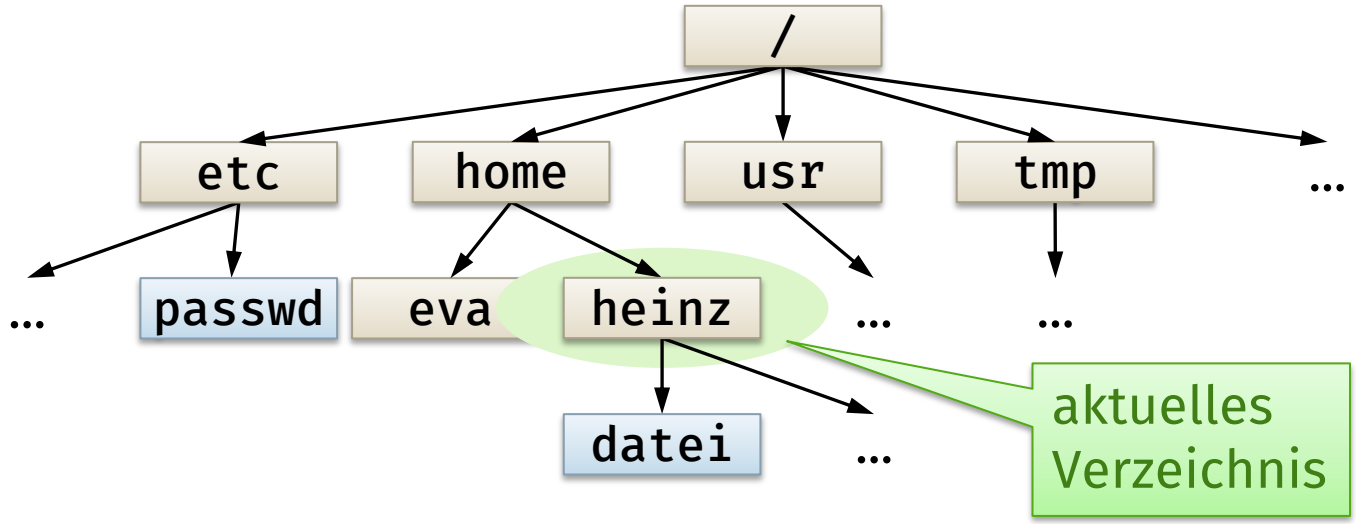
Ein Dateibaum



- Verzeichnis *(Directory)* `dir`
  - gruppiert mehrere Dateien oder Unterverzeichnisse
- Datei *(File)* `file`

Anwendersicht unter Linux (4)

Pfadnamen

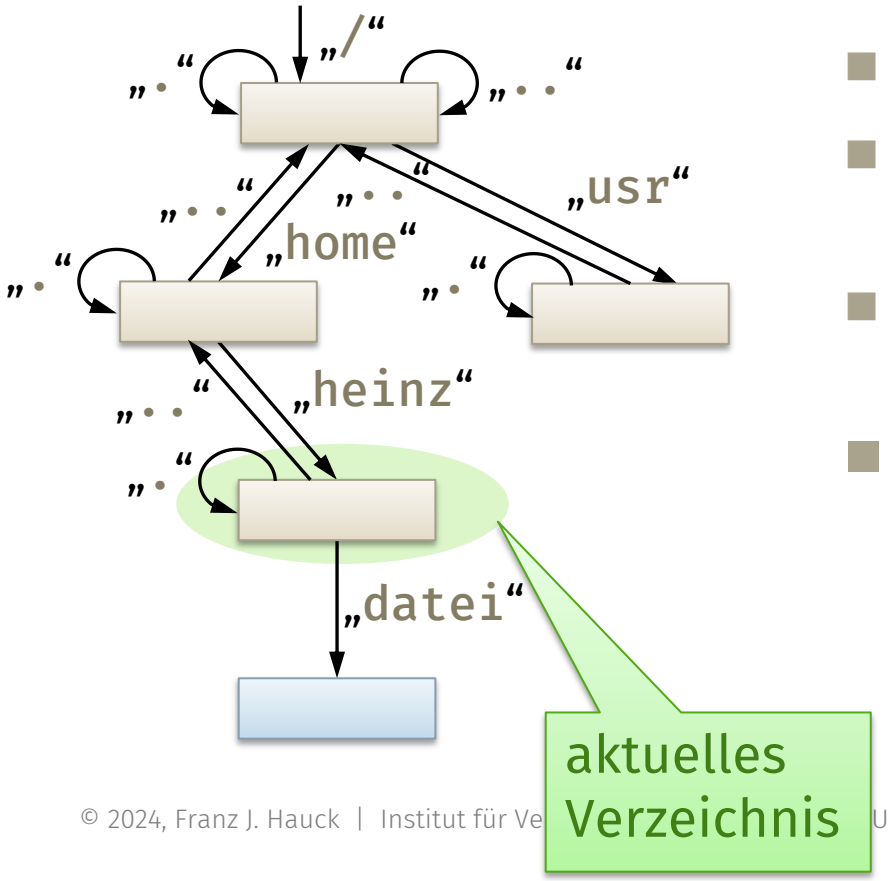


- Benennung von Dateien und Verzeichnissen
  - absolute Pfade vom Wurzelverzeichnis mit Trennsymbol „/“, z.B. `/etc`, `/home/eva`, `/home/heinz/datei`
  - relative Pfade ausgehend vom aktuellen Verzeichnis, z.B. `datei`

Anwendersicht unter Linux (5)

Pfadnamen (fortges.)

- tatsächlich werden Verbindungen benannt *(Links)*

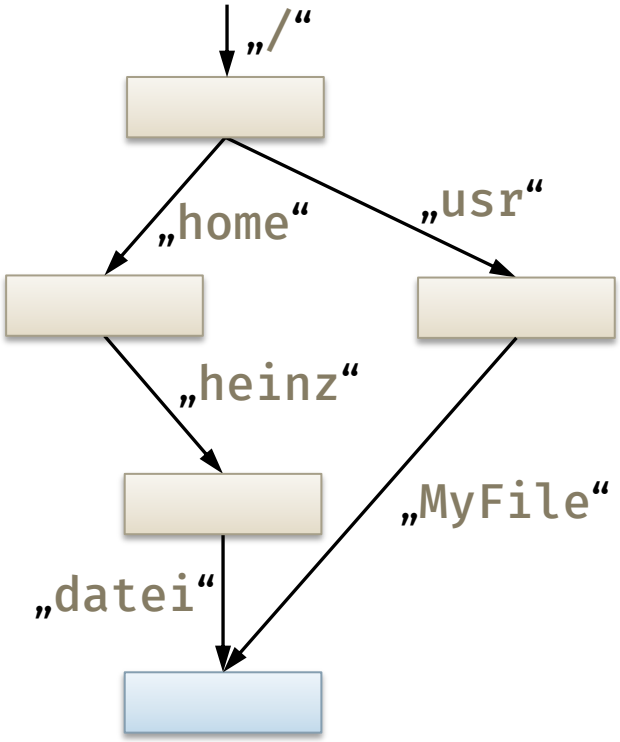


- `/` verweist auf Wurzelverzeichnis
- `.` verweist auf das eigene Verzeichnis
- `..` verweist auf das Elternverzeichnis
- mögliche Pfade zur Datei
  `/home/heinz/datei`, `datei`, `./datei`, `./../datei`, `../heinz/datei`, `/home/../home/heinz/./datei`

Anwendersicht unter Linux (6)

Mehrfachverweise *(Hard Links)*

- mehrere Verweise pro Datei

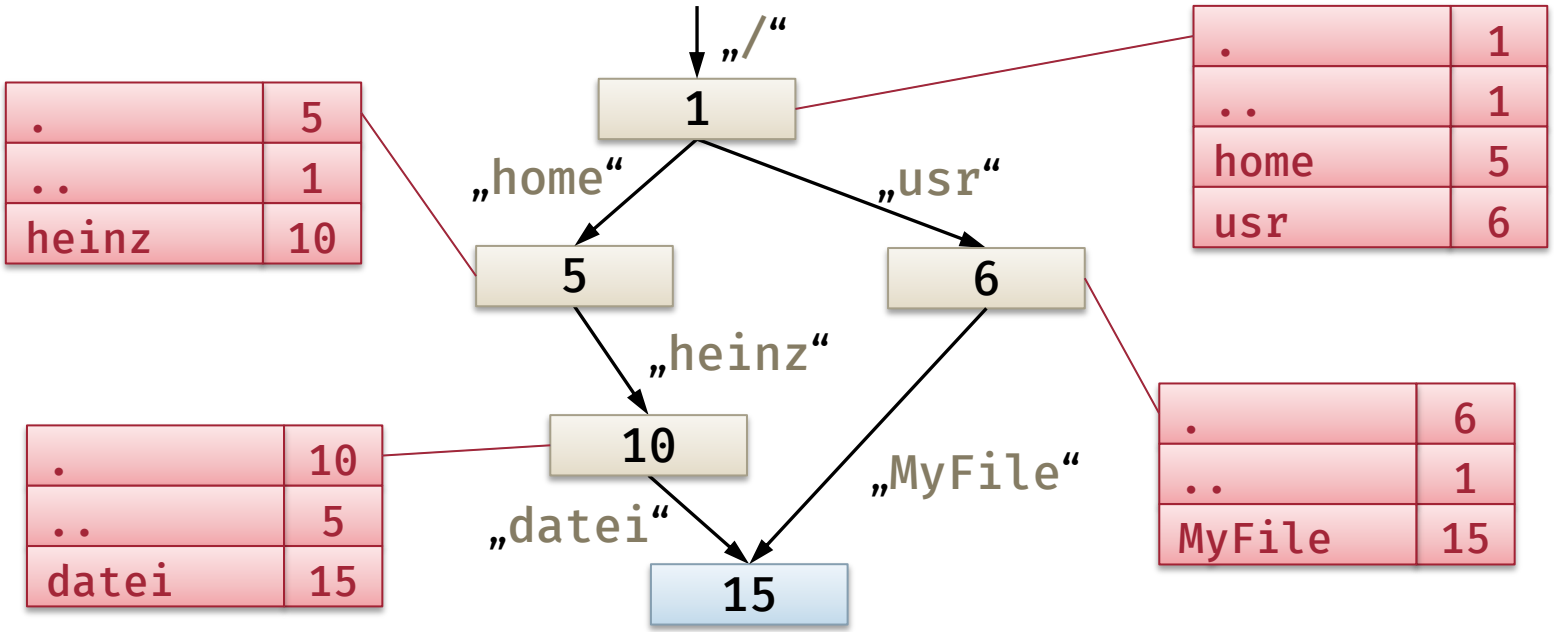


- Datei hat zwei Einträge in Verzeichnissen
  - beide gleichwertig `/home/heinz/datei` und `/usr/MyFile`
- Datei erst gelöscht, wenn letzter Link gekappt
- nur für Dateien anlegbar

Anwendersicht unter Linux (7)

Speicherung der Verzeichnisse

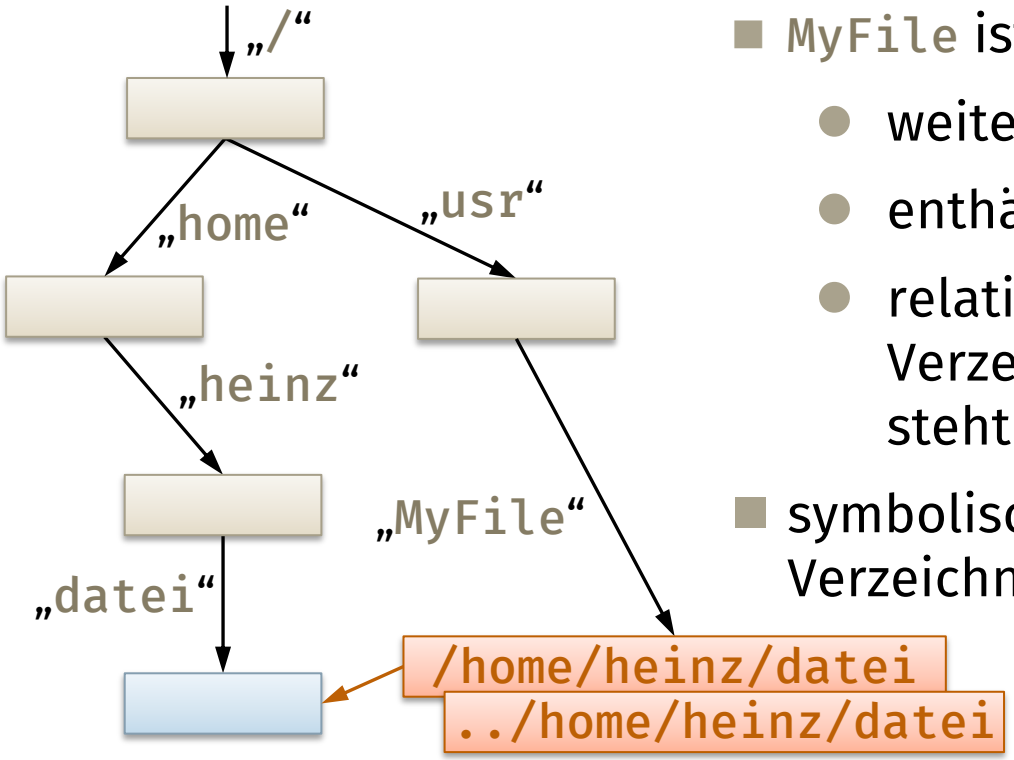
- Paare von Namen und so genannten Inode-Nummern
  - jedes Verzeichnis, jede Datei etc. besitzt einen eindeutigen Inode
  - Inode speichert Metadaten



Anwendersicht unter Linux (8)

Symbolische Verweise *(Symbolic Links)*

- Verweis über Pfad



- `MyFile` ist symbolischer Verweis
  - weiterer Typ
  - enthält Pfad zur Datei
  - relativer Pfad interpretiert zum Verzeichnis, in dem Verweis steht
- symbolische Verweise auch für Verzeichnisse verwendbar

Anwendersicht unter Linux (9)

Systemaufrufe für Dateien (Auswahl)

- Öffnen und Schließen von Dateien
  - `int fd = open( const char *path, int oflags, … );`
  - `int err = close( int fd );`
  - Filedeskriptor `fd` dient zum Zugriff auf offene Datei
  - Pfad wird nur beim Öffnen angegeben
  - Verknüpfung mit einem Schreib-/Lesezeiger
    - zeigt auf erstes zu lesendes Byte
    - wird beim Öffnen auf Anfang der Datei gesetzt

Anwendersicht unter Linux (10)

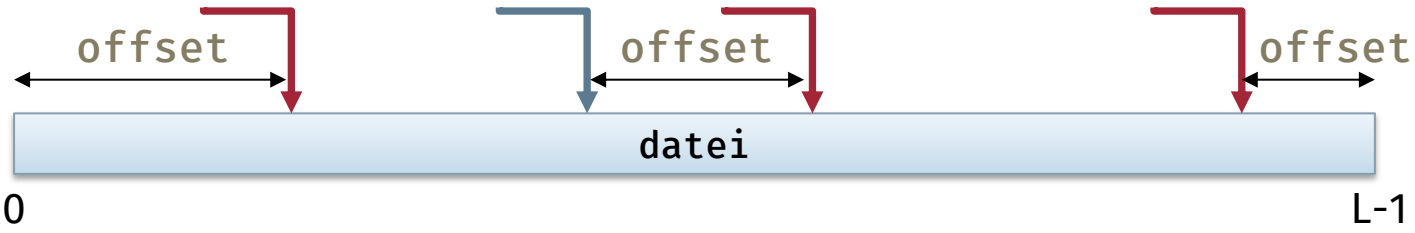
Systemaufrufe für Dateien (fortges.)

- Lesen und Schreiben von Dateien
  - `ssize_t len = read( int fd, void *buf, size_t nbyte);`
  - `ssize_t len = write( int fd, void *buf, size_t nbyte);`
  - Angabe einer Speicheradresse `buf`, sowie Anzahl der Bytes `nbyte`
  - Fortschreiben des Schreib-/Lesezeiger
  - Rückgabe der tatsächlich gelesenen oder geschriebenen Bytes

Anwendersicht unter Linux (11)

Systemaufrufe für Dateien (fortges.)

- Positionieren des Schreib-/Lesezeigers
  - `off_t off = lseek( int fd, off_t offset, int whence);`
  - absolute Positionierung: `whence = SEEK_SET`
  - relativ zur aktuellen Position: `whence = SEEK_CUR`
  - relativ zum Dateiende: `whence = SEEK_END`



Anwendersicht unter Linux (12)

Systemaufrufe für Verzeichnisse

- Lesen von Verzeichnissen
  - `DIR *opendir( const char *dirpath );`
  - `struct dirent *readdir( DIR *dirp );`
  - `int closedir( DIR *dirp );`
- Schreiben von Verzeichnissen
  - `int open( const char *path, int oflags, … );`
  - `int link( const char *path, const char *newpath );`
  - `int symlink( const char *path, const char *newpath );`
  - `int unlink( const char *path );`
  - `int mkdir( const char *path, mode_t mode);`
  - `int rmdir( const char *path );`

Anwendersicht unter Linux (13)

Metadaten pro Datei

- gespeichert im Inode
- Typ
  - Verzeichnis, Datei, Symbolischer Verweis etc.
- Länge der Daten
  - sichert, dass nicht mehr gelesen wird, als vorhanden
- Ortsinformationen
  - Wo stehen die Daten auf dem Speichermedien?
- Eigentümer und Berechtigungen
- Zeitstempel
  - letzte Änderung, letzter Zugriff, letzte Änderung am Inode

Anwendersicht unter Linux (14)

Metadatenzugriffe

- Auslesen des Inode
  - `int fstat( int fd, struct stat *buf );`
  - `int stat( const char *path, struct stat *buf );`
- Schreiben des Inodes
  - vieles nur implizit möglich (z.B. Länge der Datei)
  - `int utime( const char *path, const struct utimbuf *times);`
    - zum Setzen der Zeitstempel
  - Operationen zu Berechtigungen später
- Ändern des aktuellen Verzeichnisses
  - `int chdir(const char *path);`

Grundlagen der Betriebssysteme | F.3

Franz J. Hauck | Institut für Verteilte Systeme, Univ. Ulm

Inhaltsüberblick

Dateiverwaltung

- Einheiten und Speicherhierarchie
- Aufbau von Platten
- Anwendersicht unter Linux
  - Operationen und Attribute
- Anwendersicht unter Windows
- Unix/Linux Dateisysteme
  - Mounten, Inodes, UFS, BSD 4.2, EXT2
- Windows Dateisysteme
  - FAT32, NTFS
- Zuverlässige Dateisysteme

Anwendersicht unter Windows

Dateien

- einfache, unstrukturierte Folge von Bytes
- beliebiger Inhalt
  - für das Betriebssystem ist der Inhalt transparent
    - auch wenn das BS Annahmen über den Inhalt anhand des Namens macht
- dynamisch erweiterbar
- wahlfreier Zugriff

Anwendersicht unter Windows (2)

Verzeichnisse

- baumförmig strukturiert
  - Knoten des Baums sind Verzeichnisse
  - Blätter des Baums sind Dateien
    - Konzept der Verweise (Links) erst spät eingeführt

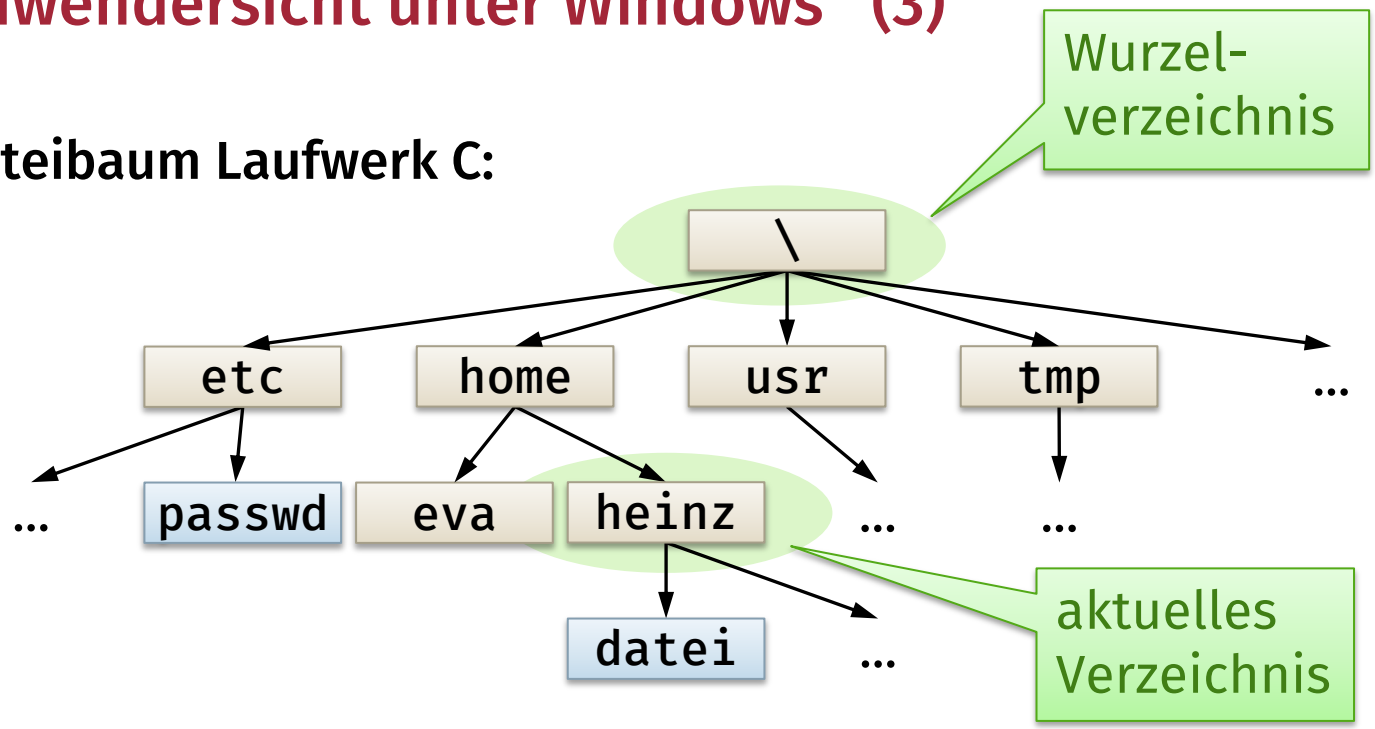
Laufwerke

- identifiziert durch Laufwerksbuchstaben, z.B. C:
- mit jeweils eigenem Dateibaum

- jedem Windows-Prozess zu jeder Zeit zugeordnet:
  - ein aktuelles Laufwerk
  - für jedes Laufwerk ein aktuelles Verzeichnis

Anwendersicht unter Windows (3)

Dateibaum Laufwerk C:



- Silbentrenner nun „\“
- mögliche Pfade bei aktuellem Laufwerk C:
  - `C:\home\heinz\datei`, `\tmp`, `C:datei` , `datei`

Anwendersicht unter Windows (4)

Namenskonvention

- historisch: 8 Zeichen Name, 3 Zeichen Erweiterung
  - z.B. AUTOEXEC.BAT
  - heute noch als Kompatibilitätsmodus vorhanden
- heute: 255 Zeichen Name
  - Punkt . gehört zum Namen

Verzeichnisse

- jedes Verzeichnis mit Verweis auf sich selbst „.“ und Verweis auf Elternverzeichnis „..“
  - Ausnahme: Wurzelverzeichnis

Anwendersicht unter Windows (5)

Operationen

- Windows-spezifisch (hier nicht betrachtet)
- POSIX-kompatible
  - POSIX = Portable Operating System Interface
  - identisch zu Linux-Systemaufrufen

Entwicklung der Dateiverwaltung

- Annäherung im Laufe der Zeit zwischen Linux und Windows
  - Linux kann jetzt auch erweiterte Attribute (Metadaten)
  - Windows kann jetzt auch Hard und Symbolic Links
  - in beiden Fällen Funktionsumfang jedoch abhängig von Dateisystemimplementierung

Dateiverwaltung

Implementierung der Anwendersicht

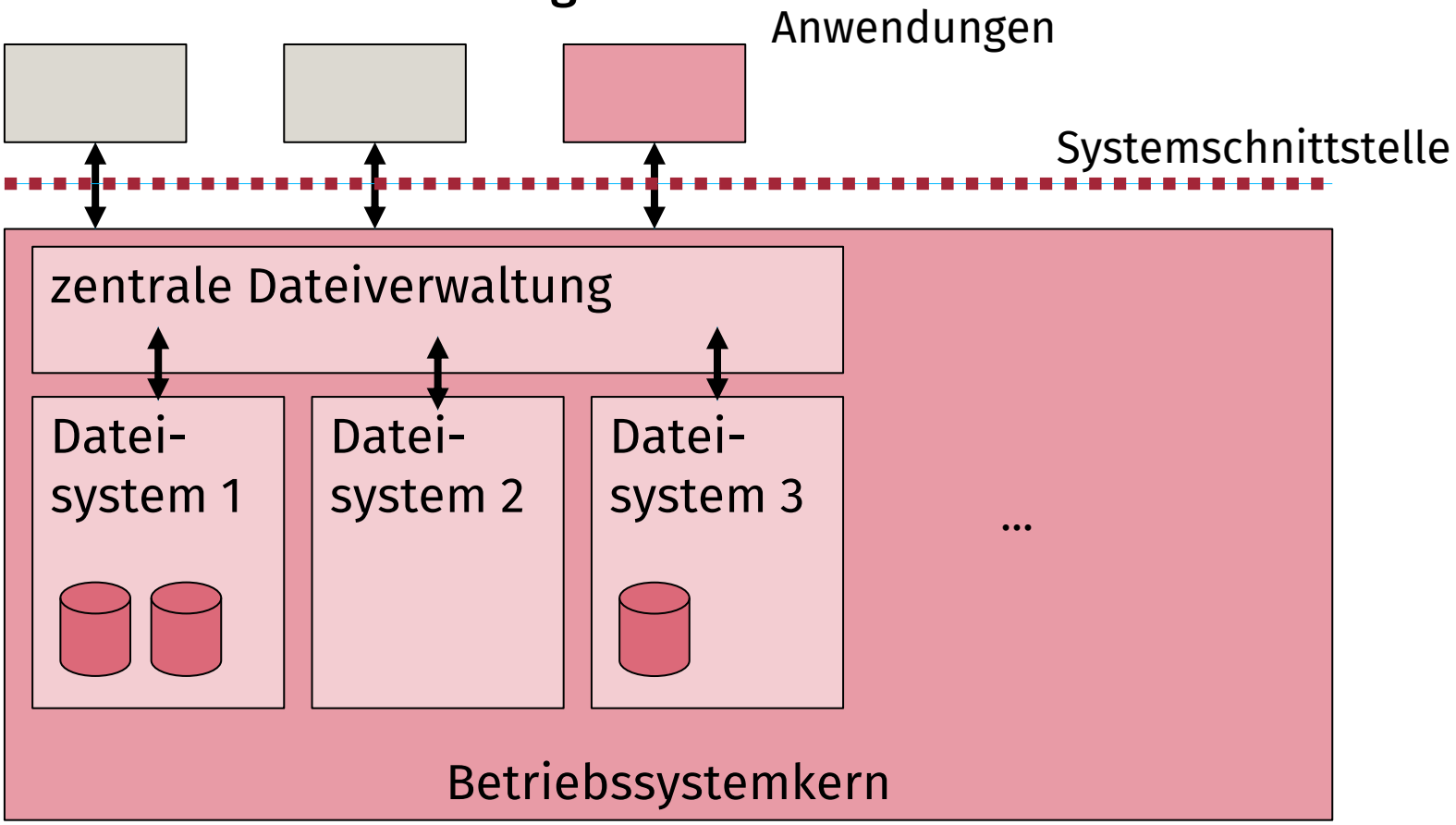
- Schnittstelle
- Semantik der Operationen

Verschiedene Darstellung auf Speichermedien

- so genannte Dateisysteme
- Begriff wird sowohl für Typ als auch für Instanzen genutzt
  - Typ: z.B. FAT32-Dateisystem
    - d.h. Implementierung der Darstellung auf Datenträger
  - Instanz: z.B. das FAT32-Dateisystem auf einem USB-Stick
    - d.h. konkrete Darstellung auf Datenträger

Dateiverwaltung (2)

Schematische Darstellung



Dateiverwaltung (3)

Abbildung von Dateisystemen auf Datenträger

- typisch: pro Datenträger ein Dateisystem
  - d.h. eine Instanz eines bestimmten Typs
- aber: andere Zuordnungen ebenfalls gebräuchlich
  - mehrere Dateisysteme pro Datenträger
    - z.B. Partitionen auf Festplatten,
      d.h. Aufteilung der Platte in mehrere Bereiche
  - ein Dateisystem über mehrere Datenträger
    - z.B. ein großes Dateisystem über ein Array von Festplatten,
      d.h. jede Platte stellt einen Teil des Gesamtspeicherbereichs bereit

Dateiverwaltung (4)

Vorteil einer Dateiverwaltung

- Dateisysteme speichern Daten und Programme persistent in Dateien
- Betriebssystemabstraktion zur Nutzung von Hintergrundspeichern
  - z.B. Festplatten, Flash-Speicher, CD-ROMs, DVDs, Bandlaufwerke, ...
- Nutzer muss sich nicht um die Ansteuerung verschiedener Speichermedien kümmern
- einheitliche Sicht auf den Sekundärspeicher

Grundlagen der Betriebssysteme | F.4

Franz J. Hauck | Institut für Verteilte Systeme, Univ. Ulm

Inhaltsüberblick

Dateiverwaltung

- Aufbau von Platten
- Anwendersicht unter Linux
  - Operationen und Attribute
- Anwendersicht unter Windows
- Unix/Linux Dateisysteme
  - Mounten, Inodes
  - UFS, BSD 4.2, EXT2
- Windows Dateisysteme
  - FAT32, NTFS
- Zuverlässige Dateisysteme

Dateisysteme unter Linux

Dateisystem auf Datenträger

- besitzt Wurzelverzeichnis
  - baumartig strukturiert wie in Anwendersicht erläutert
- verschiedene Formate auf Datenträger
  - Dateien und Verzeichnisse unterschiedlich realisiert

Anwendersicht

- unabhängig von Dateisystemen
- nur ein Dateibaum in der Anwendersicht

Montage des Dateibaums unter Linux

Repräsentation der Dateisysteme

- durch blockorientierte Spezialdatei
  - weiterer Typ im Dateisystem
  - z.B. `/dev/dsk/0s3`

Wurzeldateisystem *(Root File System)*

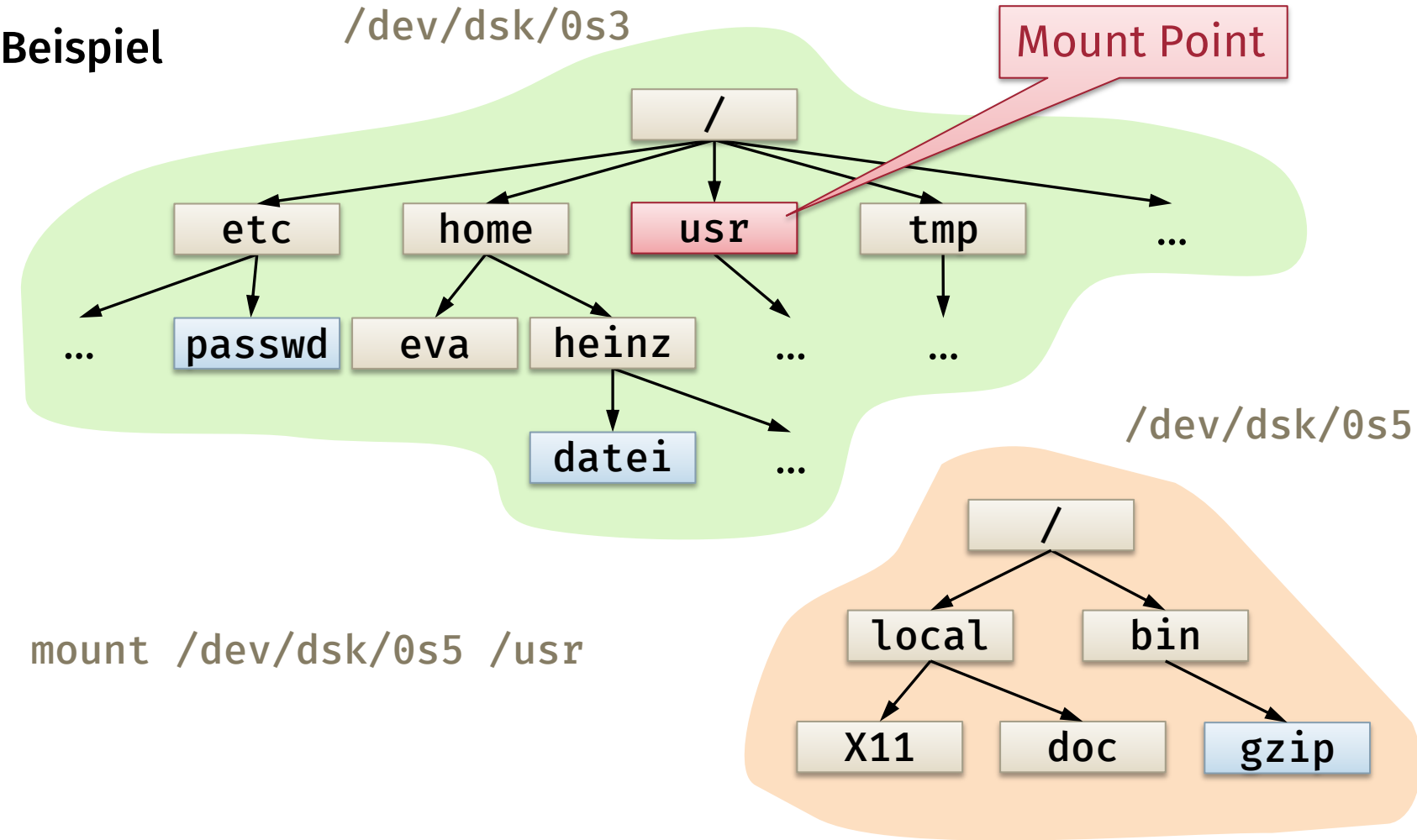
- ist vorkonfiguriert
- wird zum Hochfahren *(Booten)* benutzt

Weitere Dateisysteme

- werden in den bestehenden Dateibaum montiert
  - mit Systemaufruf `mount` bzw. gleichnamigem Programm

Montage des Dateibaums unter Linux (2)

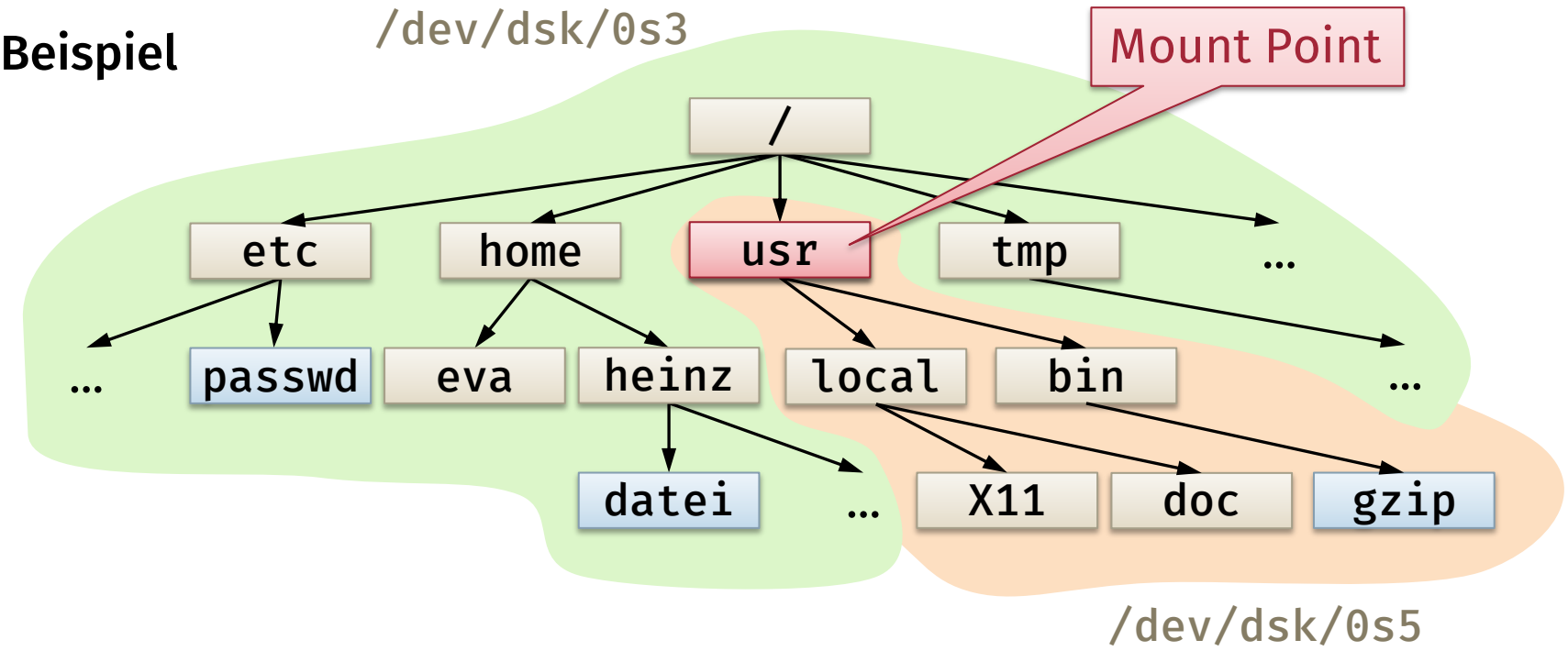
Beispiel




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

Montage des Dateibaums unter Linux (3)

Beispiel



- Dateiverwaltung leitet alle Anfragen unter `/usr` an zweites Dateisystem
  - verwaltet Mount Points

Hard Links

Hinweis zu Verweisen *(Hard Links)*

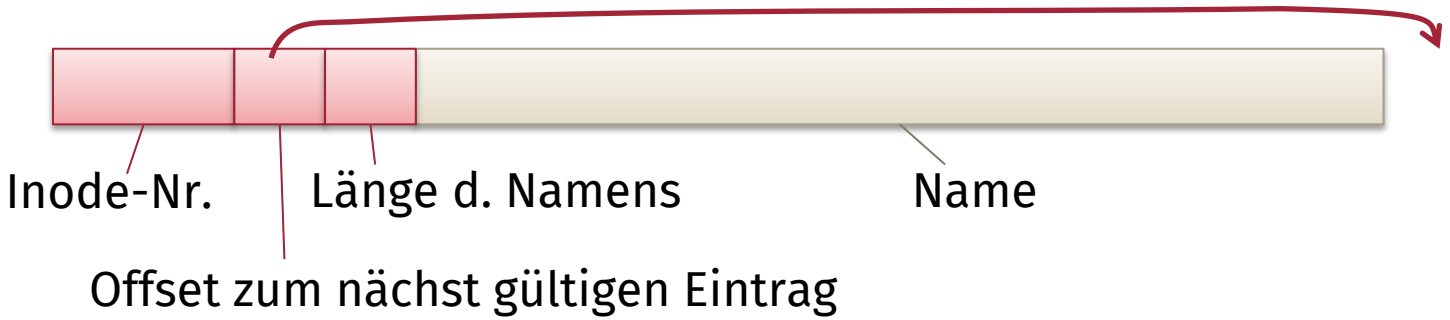
- nur möglich innerhalb einer Dateisysteminstanz
  - nicht transparent über den Dateibaum
  - Abhilfe: Nutzung symbolischer Verweise
- Verweis „. .“ in Mount Points
  - wird automatisch „verbogen“ auf Elternverzeichnis des ursprünglichen Verzeichnisses

Speicherung von Verzeichnissen

Daten ähnlich wie in einer normalen Datei

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

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



Inodes

Typische Inhalte eines Inodes

- Inode-Nummer
- Typ
  - Verzeichnis, normale Datei, symbolischer Link, Spezialdatei (z.B. Gerät) etc.
- Rechteinformationen
  - 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

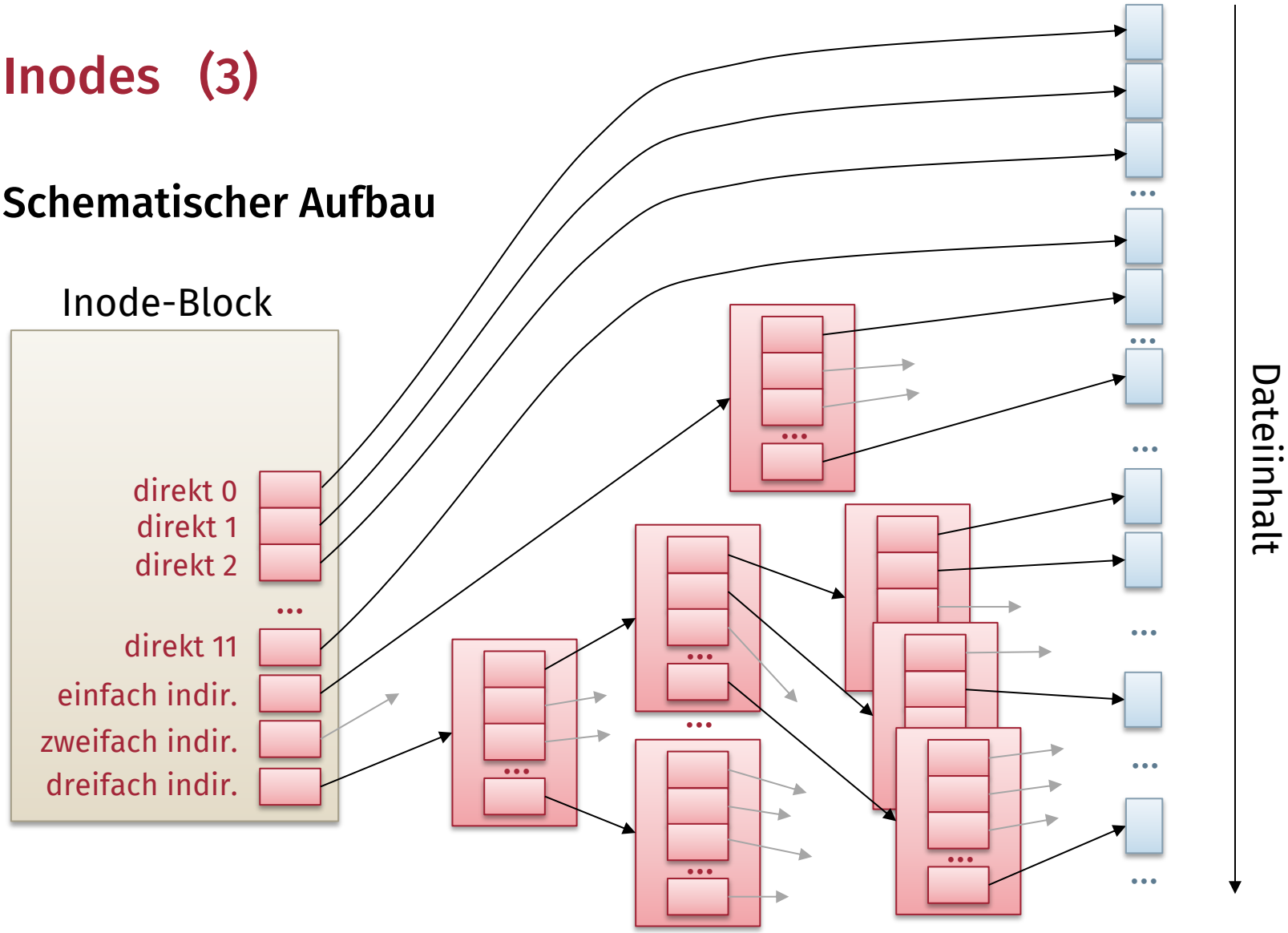
Inodes (2)

Typische Inhalte (fortges.)

- Ortsinformation für Datenblöcke
  - z.B. zwölf direkte Verweise, ein einfach, ein zweifach und ein dreifach indirekter Verweis
  - direkter Verweis: Nummer eine Plattenblocks mit Dateiinhalt
  - indirekter Verweis: Nummer eines Plattenblocks
    - enthält lauter Nummern weiterer Plattenblocks

Inodes (3)

Schematischer Aufbau



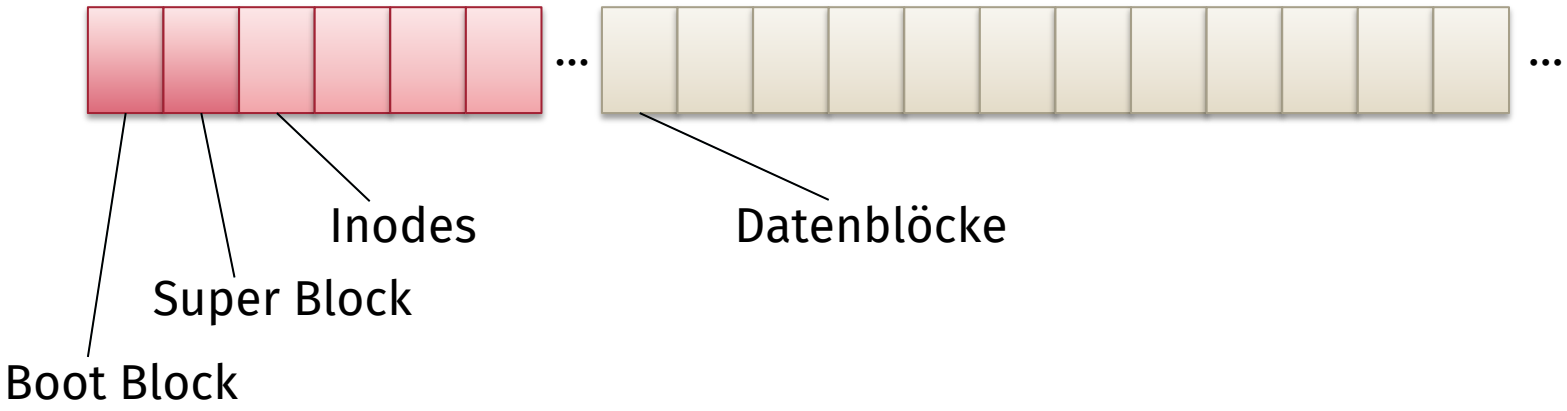
Inodes (4)

Einsatz mehrerer indirekter Stufen

- Vorteil
  - Inode benötigt sowieso Block auf Platte
    - kann ausgenutzt werden für direkte Verweise auf Datenblöcke
  - durch mehrere Indirektionsstufen auch sehr große Dateien adressierbar
  - schnellere Positionierung des Schreib-/Lesezeigers
    - im Vergleich zu gefädelter Ortsinformation (z.B. bei FAT)
- Nachteil
  - Indirektionsblöcke müssen zusätzlich geladen werden
    - nur bei langen Dateien

System V File System / UFS

Blockorganisation



- Boot Block
  - enthält Informationen zum Laden eines initialen Programms
- Super Block
  - enthält Verwaltungsinformation für das Dateisystem

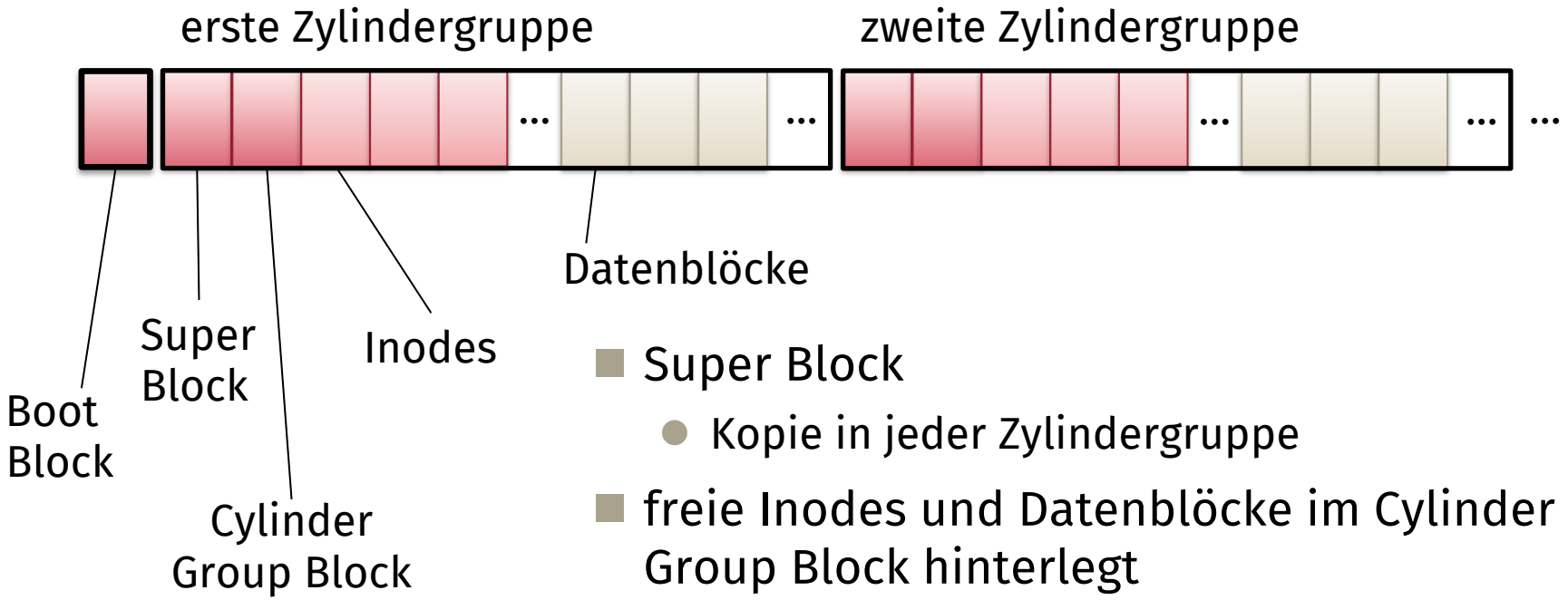
System V File System / UFS (2)

Inhalt des Super Block

- Anzahl Blöcke und Inodes
- Liste der freien Blöcke und Inodes
- Dateisystem-Attribute (z.B. clean, active)
- Bezeichnung des Dateisystems *(Label)*
- letzter Mount Point

BSD 4.2 – Berkeley Fast File System

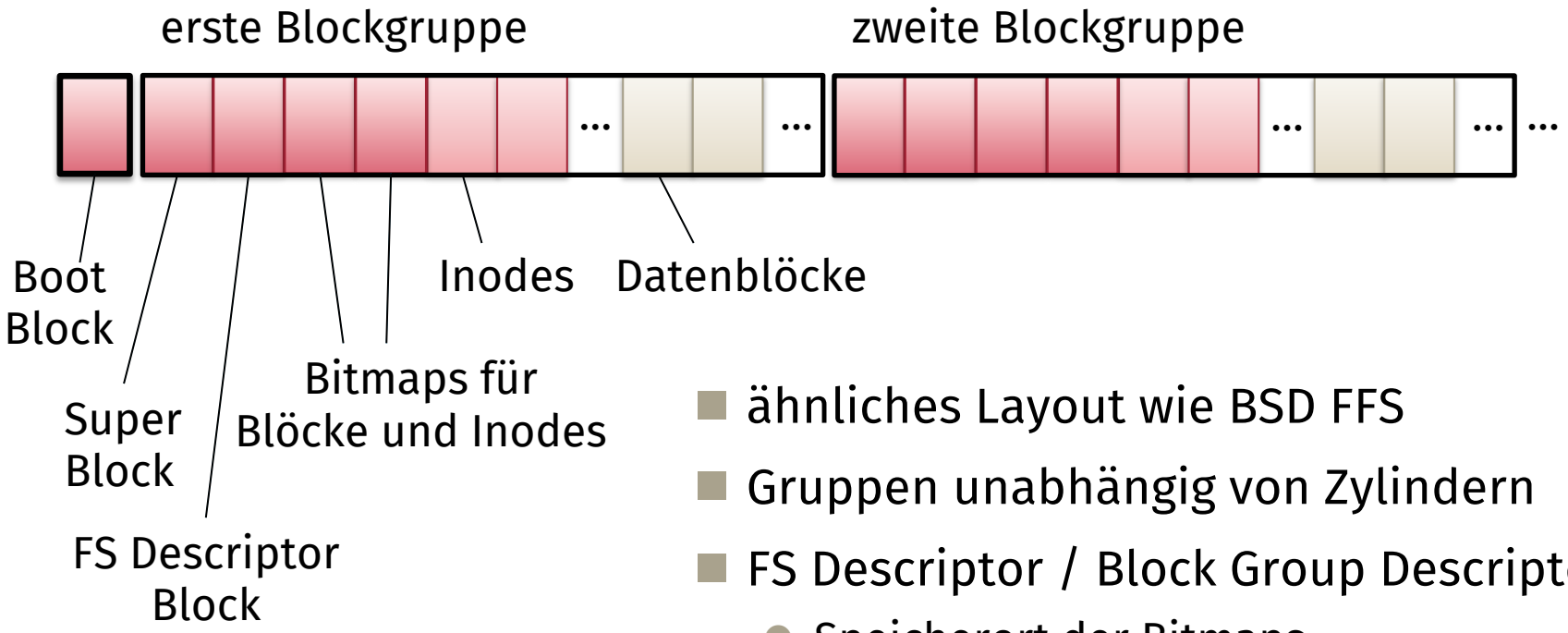
Blockorganisation



- Super Block
  - Kopie in jeder Zylindergruppe
- freie Inodes und Datenblöcke im Cylinder Group Block hinterlegt
- eine Datei bleibt möglichst in einer Gruppe
  - kürzere Positionierzeiten

Linux EXT2 File System

Blockorganisation



- ähnliches Layout wie BSD FFS
- Gruppen unabhängig von Zylindern
- FS Descriptor / Block Group Descriptor
  - Speicherort der Bitmaps
  - Anzahl freier Blöcke und Inodes

Linux EXT2 File System (2)

Blockorganisation (fortges.)

- Bitmaps
  - für Blöcke und für Inodes
  - jedes Bit codiert Zustand eines Blocks in der Blockgruppe
    - `1` = belegt, `0` = frei

Grundlagen der Betriebssysteme | F.5

Franz J. Hauck | Institut für Verteilte Systeme, Univ. Ulm

Inhaltsüberblick

Dateiverwaltung

- Einheiten und Speicherhierarchie
- Aufbau von Platten
- Anwendersicht unter Linux
  - Operationen und Attribute
- Anwendersicht unter Windows
- Unix/Linux Dateisysteme
  - Mounten, Inodes, UFS, BSD 4.2, EXT2
- Windows Dateisysteme
  - FAT32, NTFS
- Zuverlässige Dateisysteme

Dateisysteme unter Windows

Dateisystem auf Datenträger

- besitzt Wurzelverzeichnis
  - baumartig strukturiert wie in Anwendersicht erläutert
- verschiedene Formate auf Datenträger
  - Dateien und Verzeichnisse unterschiedlich realisiert

Anwendersicht

- unabhängig von Dateisystemen
- jeder Datenträger eigenes Laufwerk
  - auch Montieren möglich, aber niemand nutzt das

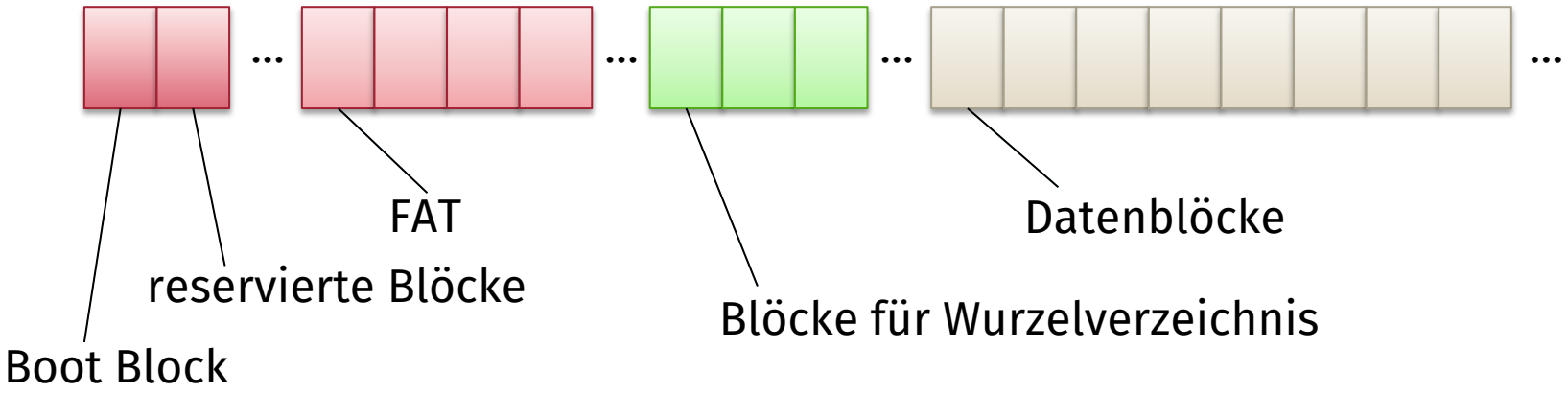
FAT-Dateisystem

Historie

- „uralte“ Dateisystemimplementierung
  - angeblich 1977
  - Einsatz unter MS-DOS
- FAT steht für File Allocation Table
  - gibt es als 12 Bit-, 16 Bit- und 32 Bit-Version
  - abhängig von Anzahl der verwalteten Blöcke auf dem Datenträger
- Einsatz heute:
  - selten für Festplatten
  - Floppy-Disks
  - USB-Sticks
  - optische Speichermedien (CD, DVD, BlueRay)

FAT-Dateisystem (2)

Blockorganisation



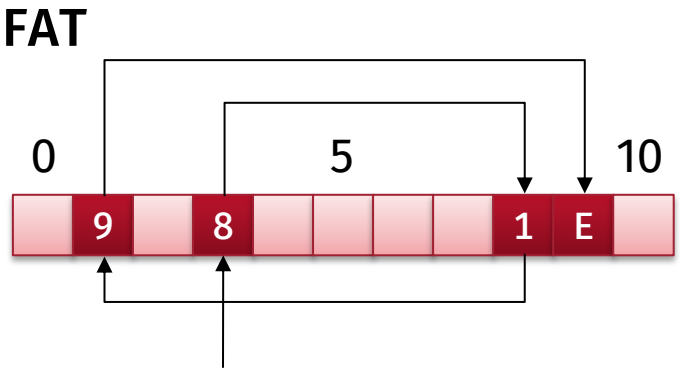
- reservierte Blöcke
  - weitere Blöcke zum Booten
  - Sicherungskopie für Boot Block
- Datenblöcke für Dateien und Unterverzeichnisse

Datenspeicherung und FAT

Kette der Datenblöcke einer Datei

- gespeichert in der FAT *(File Allocation Table)*
- FAT enthält für jeden Datenblock einen Eintrag
  - FAT32: 32Bit Wert, aber nur 28Bit benutzt
  - Blöcke werden auch Cluster genannt
- Markierung eines Eintrags als
  - frei = `0x00000000`
  - belegt = Nummer des nächsten Blocks oder `0x0ffffff8-0x0fffffff` für letzten Block
  - beschädigt = `0x0ffffff7`
- FAT realisiert für jede Datei eine einfach verkettete Liste

Datenspeicherung und FAT (2)



- mehrfache Speicherung der FAT
  - verhindert Totalausfall, falls ein FAT-Block nicht mehr lesbar

Datenspeicherung und FAT (3)

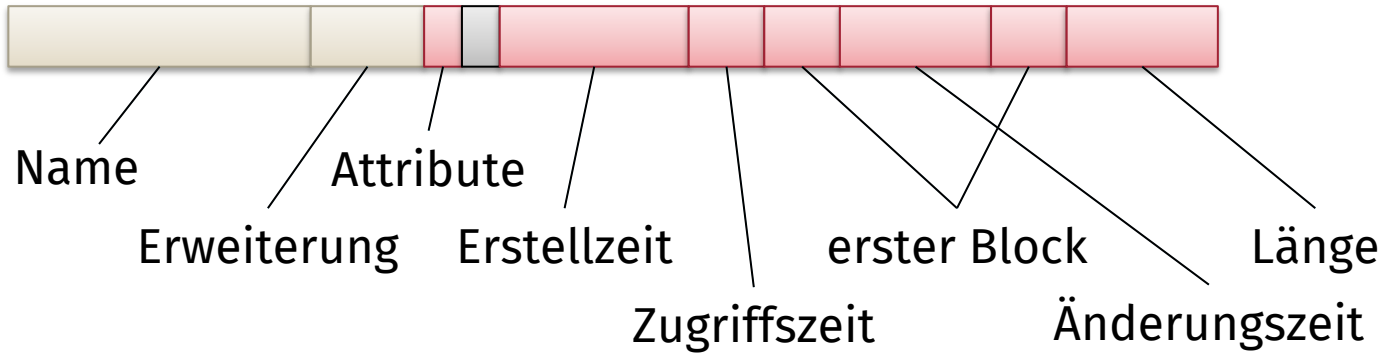
Probleme

- mindestens ein zusätzlicher Block muss geladen werden
- häufiges Positionieren des Schreib-/Lesekopfes bei verstreuten Datenblöcken
- Laden der FAT-Blöcke enthält auch nicht aktuell benötigte Info
- ❖ kann durch Caching im Hauptspeicher ausgeglichen werden
  - FAT wird häufig benötigt
- aufwändige Suche nach dem Datenblock bei bekannter Position in der Datei

FAT Verzeichniseintrag

Einträge gleicher Länge

- hintereinander gespeichert



- Kurzform der Namen
  - erhält Kompatibilität
- Langform durch Zusammenfassen mehrere Verzeichniseinträge
  - bis 255 Zeichen lange Namen

FAT Verzeichniseintrag (2)

Inhaltseinträge

- Attribute
  - schreibgeschützt *(read only)*
  - versteckt *(hidden)*
  - Systemdatei *(system)*
  - Volume-Label (Bezeichnung für den Datenträger)
  - Unterverzeichnis *(sub-directory)*
  - Archivbit *(archive)*
- verschiedene Zeitstempel

Grundlagen der Betriebssysteme | F.6

Franz J. Hauck | Institut für Verteilte Systeme, Univ. Ulm

Inhaltsüberblick

Dateiverwaltung

- Einheiten und Speicherhierarchie
- Aufbau von Platten
- Anwendersicht unter Linux
  - Operationen und Attribute
- Anwendersicht unter Windows
- Unix/Linux Dateisysteme
  - Mounten, Inodes, UFS, BSD 4.2, EXT2
- Windows Dateisysteme
  - FAT32, NTFS
- Zuverlässige Dateisysteme

NTFS – New Technology File System

Ursprünglich Dateisystem für Windows NT

- feingranulare Berechtigungen
- automatische Dateikompression
- große Dateien bis 16 Exabytes
  - 16 TB in aktuellen Implementierungen
- Hard Links
- sichere Speicherung

NTFS-Eigenschaften

Verweise

- enthalten alle Verweis auf sich selbst und Elternverzeichnis
  - „`.`“ und „`..`“
- Hard Links innerhalb einer Dateisysteminstanz *(Volume)*
- keine symbolischen Verweise in NTFS integriert
  - aber Windows Dateiverwaltung kennt `*.lnk` Dateien (Verknüpfungen)

Namen

- bis 255 Zeichen lang
  - aber kompatibler Name für MS-DOS integriert

NTFS-Eigenschaften (2)

Cluster

- Basiseinheit für das Dateisystem
  - 512B bis 4KiB
  - beim Formatieren festgelegt
- logische Cluster-Nummer als Adresse *(LCN)*

Extent

- mehrere aufeinanderfolgende Cluster

NTFS-Eigenschaften (3)

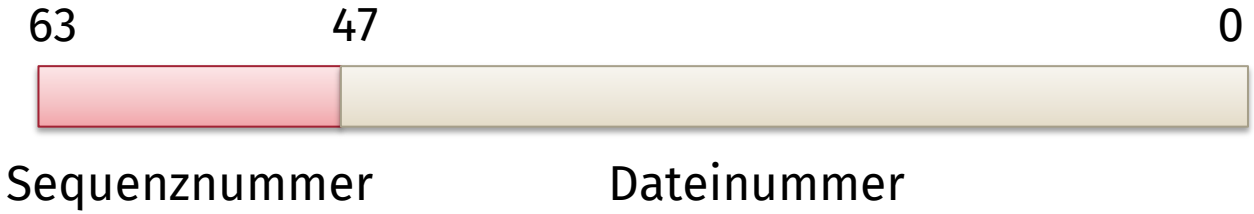
Strom

- Datei speichert mehrere Datenströme
  - einzeln benennbar, z.B. `text.txt:extrastream`
- Hauptstrom für eigentliche Daten
- „Nebenströme“ i.d.R. durch Windows nicht mehr unterstützt
  - z.B. Anzeigen im Explorer, Kopieren auf USB-Sticks, Anhängen an Mails, ...
- intern eigene Ströme für
  - Dateiname, MS-DOS Dateiname
  - Zugriffsrechte
  - Zeitstempel
  - u.a. Attribute

NTFS-Dateiorganisation

File Reference

- eindeutiger Bezeichner für Datei oder Verzeichnis



- Dateinummer ist Index in eine globale Tabelle
  - Master File Table (MFT)
- Dateinummer gelöschter Einträge kann wiederverwendet werden
  - Sequenznummer wird hochgezählt
  - neue Datei hat andere File Reference bei gleicher Dateinummer

NTFS-Dateiorganisation (2)

Master File Table

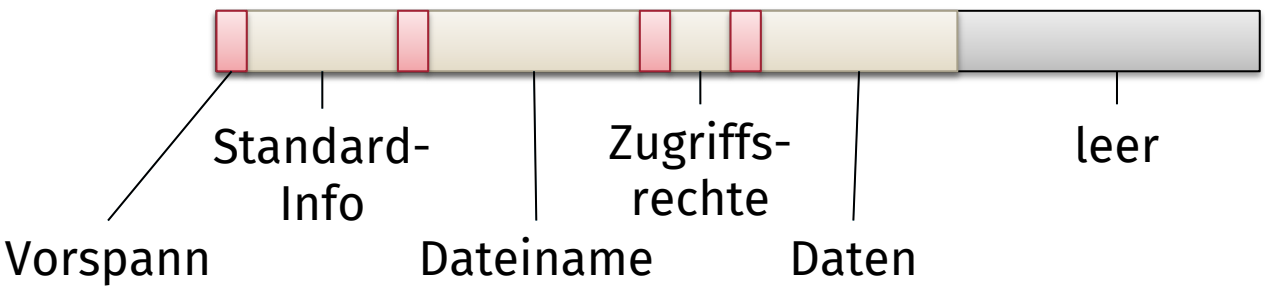
- Rückgrat der Dateisysteminstanz
- große Tabelle mit gleich langen Einträgen
  - 1KiB, 2KiB oder 4KiB je nach Cluster-Größe

Dateinummer in File Reference bestimmt Eintrag für Datei oder Verzeichnis

NTFS-Dateiorganisation (3)

Kurze Datei

- Einträge in der MFT

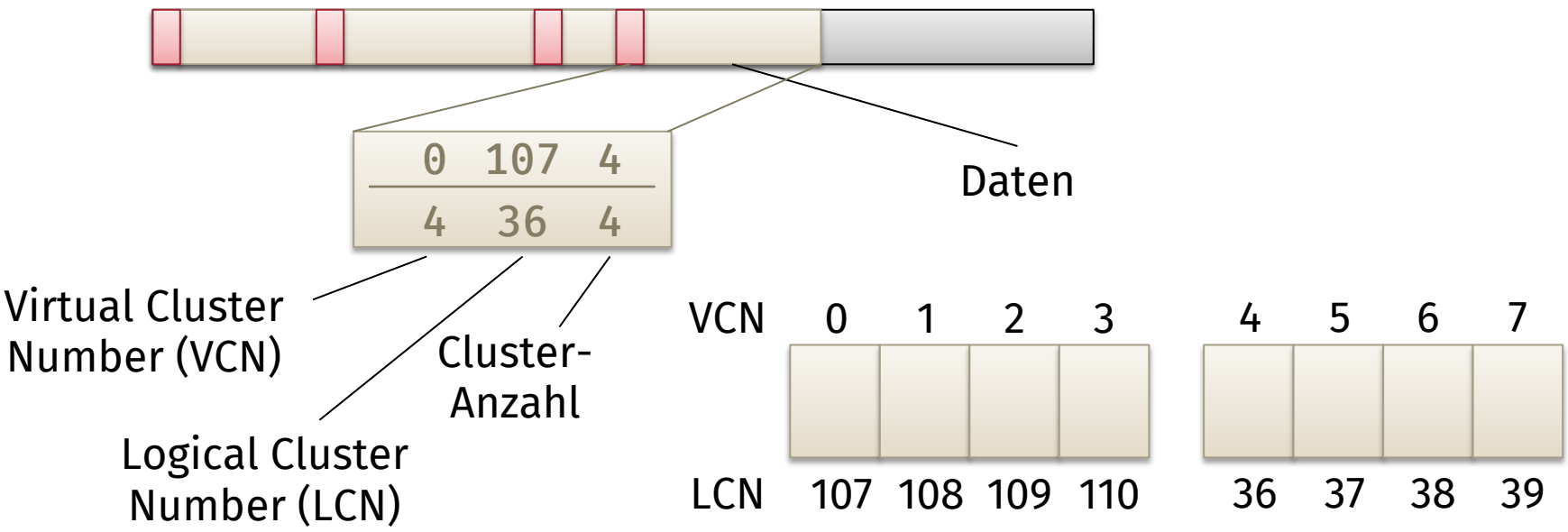


- Standard-Info (immer in der MFT)
  - Länge, MS-DOS-Attribute (siehe FAT), Zeitstempel, Anzahl Hard Links, Sequenznummer aktueller File Reference
- Dateiname (immer in der MFT)
  - kann mehrfach vorkommen (Hard Links, MS-DOS-Name)

NTFS-Dateiorganisation (3)

Längere Datei

- Daten passen nicht in MFT-Eintrag



- Erweiterung eines Datenstroms durch Ortsinformation über externe Extents

NTFS-Dateiorganisation (4)

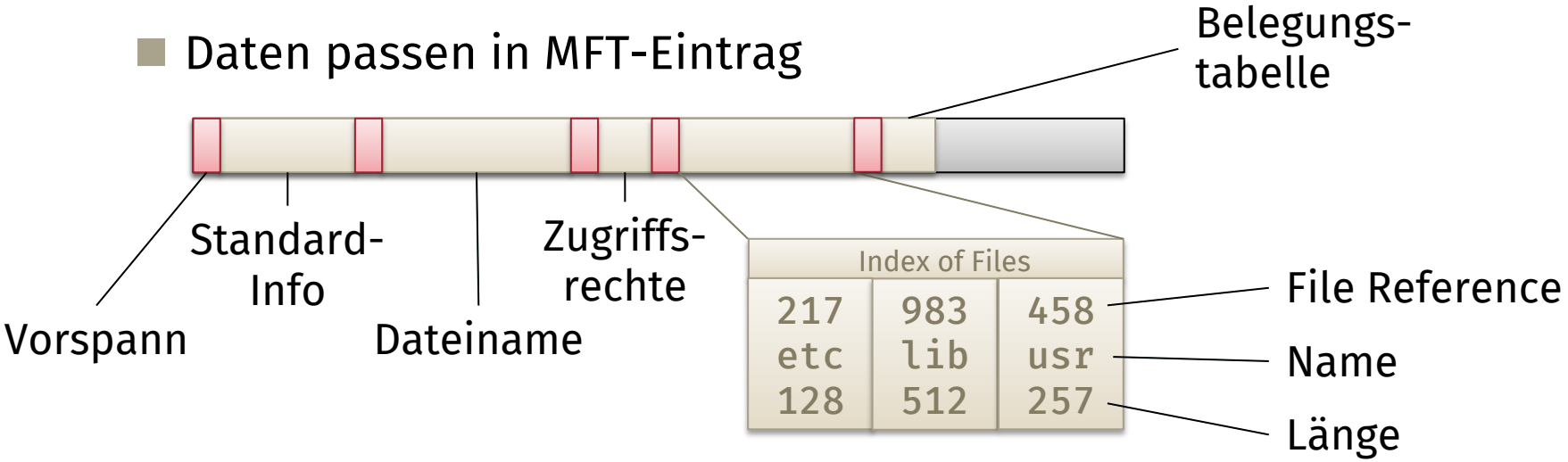
Weitere Ströme *(Attributes)*

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

NTFS-Dateiorganisation (5)

Kurzes Verzeichnis

- Daten passen in MFT-Eintrag

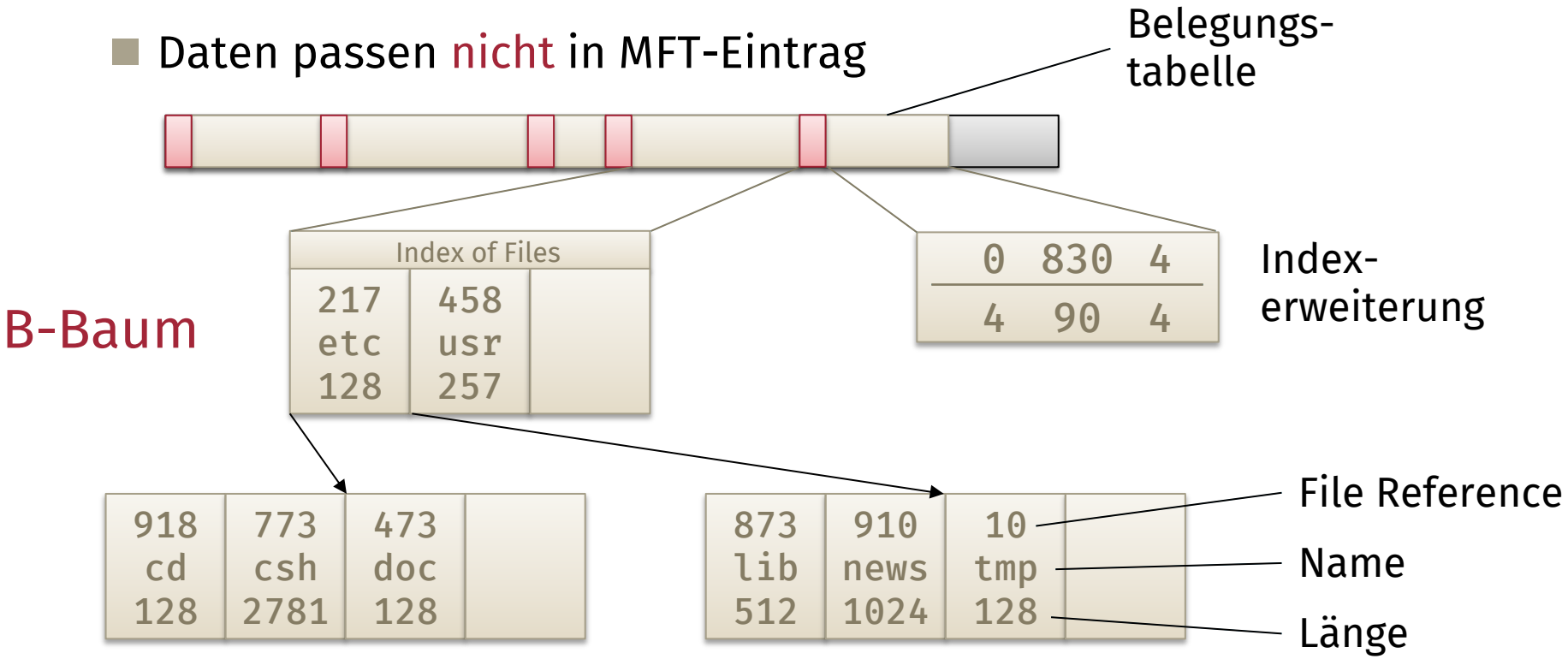


- Verweise mit Hilfe der File Reference
- Name und Länge im Verzeichnis und in der Datei gespeichert
  - doppelter Aufwand bei Aktualisierungen
  - schnellerer Zugriff beim Listen eines Verzeichnisses

NTFS-Dateiorganisation (5)

Langes Verzeichnis

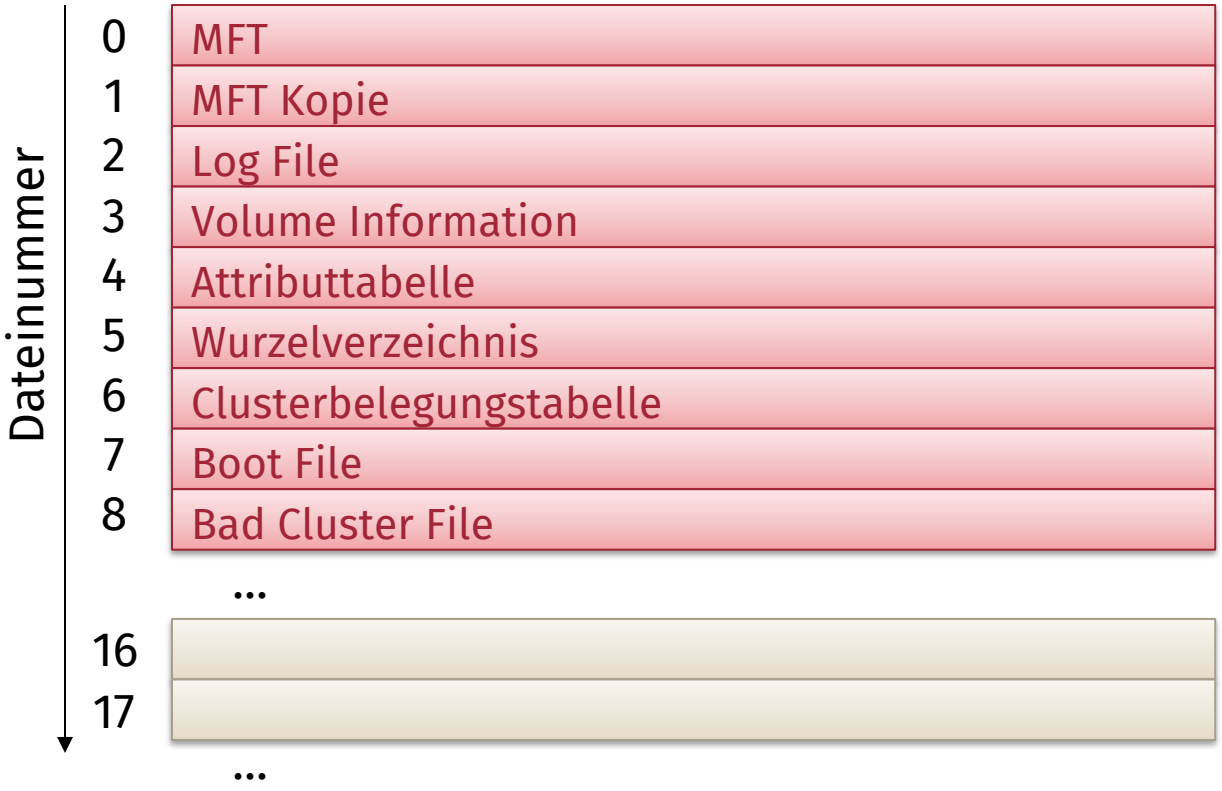
- Daten passen nicht in MFT-Eintrag



Extents mit Listen on Einträgen

NTFS-Dateiorganisation (6)

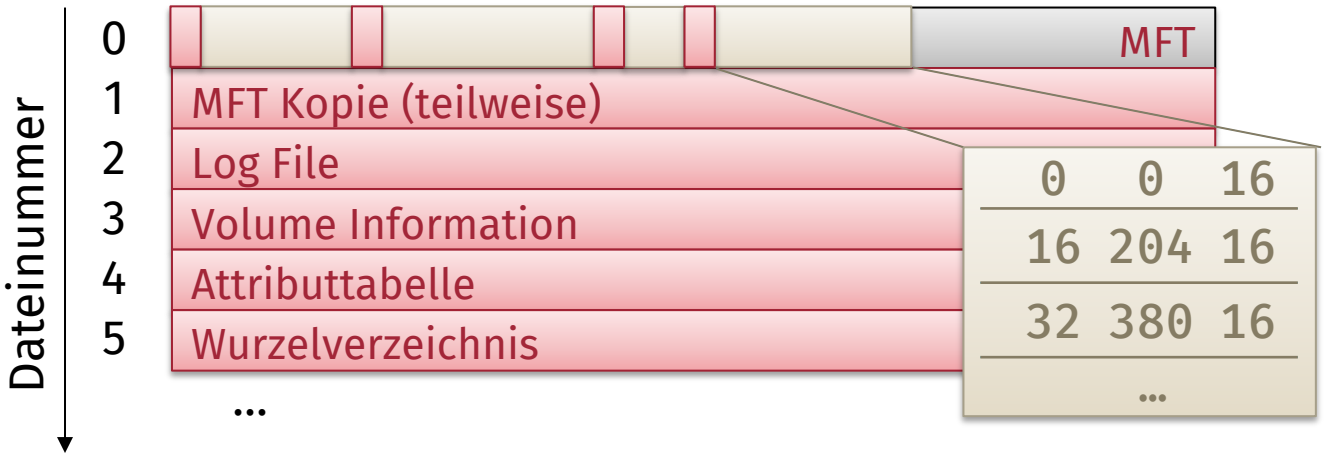
Alle Metadaten in Dateien



NTFS-Dateiorganisation (7)

Metadaten-Dateien

- MFT und MFT-Kopie
  - selbst eine Datei
  - Kopie des ersten Eintrags
    - Fehlertoleranz



NTFS-Dateiorganisation (8)

Metadaten-Dateien (fortges.)

- Log File
  - enthält protokollierte Änderungen am Dateisystem
- Volume Information
  - Name, Größe und ähnliche Attribute des Volumes
- Attributtabelle
  - definiert mögliche Ströme in den Einträgen
- Clusterbelegungstabelle
  - Bitmap für jeden Cluster des Volumes
- Bad Cluster File
  - enthält alle nicht lesbaren Cluster des Datenträgers
  - automatisch markiert

Weitere Eigenschaften von NTFS

Kompression

- optionale Kompression von Dateien (LZ77) integriert
  - kann auf Datei- oder Verzeichnisebene eingeschaltet werden
- geeignet regulär strukturierte Dateien
  - mit seltenen Schreib- und
  - häufigen sequentiellen Lesezugriffen

Verschlüsselung

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

Grundlagen der Betriebssysteme | F.7

Franz J. Hauck | Institut für Verteilte Systeme, Univ. Ulm

Inhaltsüberblick

Dateiverwaltung

- Einheiten und Speicherhierarchie
- Aufbau von Platten
- Anwendersicht unter Linux
  - Operationen und Attribute
- Anwendersicht unter Windows
- Unix/Linux Dateisysteme
  - Mounten, Inodes, UFS, BSD 4.2, EXT2
- Windows Dateisysteme
  - FAT32, NTFS
- Zuverlässige Dateisysteme

Konsistenz von Dateisystemen

Mögliche Fehler

- Stromausfall
- 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
- Ursache
  - Änderungen betreffen mehrere Teile des Dateisystems
    - z.B. Bitmap für freie Blöcke und Dateidatenstrom selbst
  - sequentielle Speicherung, die irgendwo unterbrochen wird

Konsistenz von Dateisystemen (2)

Reparaturprogramme

- Konsistenzprüfung beim Hochfahren/Montieren
  - Programme wie `chkdsk`, `scandisk` oder `fsck` können inkonsistente Metadaten reparieren
  - Datenverluste bei Reparatur möglich
- große Datenträger implizieren lange Prüf- und Reparaturzeiten

❖ Ansätze gesucht, die Prüfzeiten minimieren

Journaling File System

Protokollieren von Änderungen

- Einsatz eines Journals bzw. einer Log-Datei
  - Änderungen werden protokolliert
- bei Systemausfall kann Protokolls mit Dateisystem abgeglichen werden
  - Wiederherstellung von Änderungen oder
  - Rücknahme von Änderungen
- erheblich Verkürzung der Prüf- und Reparaturphase
- etwas ineffizienter wg. Protokollführung

Beispiele

- NTFS, EXT3 (EXT2 mit Journaling), ReiserFS

Journaling File System (2)

Idee

- jede (Teil-)Änderung gehört zu einer Transaktion
  - Transaktionen sind alle verändernden Systemaufrufe
  - z.B. Erzeugen, Löschen, Erweitern, Verkürzen von Dateien, Dateiattribute verändern, Datei umbenennen, Verzeichnis anlegen usw.
  - Beispiel für (Teil-)Änderungen der Transaktion Löschen einer Datei
    - Löschen der Verzeichniseinträge (evtl. mehrere)
    - Freigabe der Extents aller Datenströme (evtl. mehrere)
    - Freigabe des MFT-Eintrags

Journaling File System (3)

Idee (fortges.)

- jede (Teil-)Änderung wird protokolliert *(Log File)* und durchgeführt
- Vergleich von Protokoll und Dateisystem zur Erkennung von Inkonsistenzen beim Hochfahren/Montieren

Journaling File System (4)

Protokollierung

- für jede (Teil-)Änderung:
  - Schreiben eines Log-Eintrags
    - enthält Zuordnung zur Transaktion
      - Verknüpfung mit zugehörigen anderen Änderungen
    - enthält Angaben zur Ausführung der Änderung
    - enthält Angaben zur Rücknahme der Änderung
  - Schreiben der tatsächlichen Änderung im Dateisystem
- es muss gelten:
  - Log-Eintrag ist immer vor der eigentlichen Änderung auf Platte
  - Erinnerung: Plattenblöcke werden im Hauptspeicher geändert und dann erst auf Platte geschrieben

Journaling File System (5)

Protokollierung (fortges.)

- bei Ausfall drei Möglichkeiten für jede (Teil-)Änderung:
  - keine Änderung auf dem Datenträger
  - Änderung nur im Protokoll vermerkt nicht jedoch durchgeführt
  - Änderung im Protokoll und auf dem Datenträger
- niemals Änderung auf der Platte ohne Erwähnung im Protokoll

Journaling File System (6)

Fehlererholung

- beim Booten wird Log File für jede Transaktion überprüft
  - alle Log-Einträge zur Transaktion vorhanden *(Redo)*:
    - Anfang und Ende der Transaktion im Log
    - evtl. fehlende Änderungen werden auf Datenträger gespeichert
  - angefangene aber nicht beendete Transaktion *(Undo)*:
    - protokollierte Änderungen werden rückgängig gemacht
- keine Inkonsistenzen im Dateisystem mehr möglich

Journaling File System (7)

Beispiel

- Löschen einer Datei
- zugehörige Einträge im Log
  - mit gleicher Transaktionsnummer und File Reference
  - Beginn der Transaktion
  - Freigabe der Extents durch Löschen der entsprechenden Bits in der Belegungstabelle
    - gesetzte Bits kennzeichnen belegte Cluster
  - Löschen des Verzeichniseintrags
    - evtl. mehre (Teil-)Änderungen, z.B. Freigeben von Clustern des Verzeichnisses
  - Freigabe des MFT-Eintrags der Datei
  - Ende der Transaktion

Journaling File System (8)

Beispiel

- vollständige Transaktion im Log
  - Nachziehen aller Änderungen
    - soweit noch nicht auf Datenträger
  - *Redo*
- unvollständige Transaktion im Log
  - Ende der Transaktion nicht im Log
  - Rückgängigmachen aller protokollierter Änderungen
    - soweit schon auf Datenträger
  - *Undo*

Journaling File System (9)

Log File

- kann und soll nicht beliebig groß werden
  - gelegentlich konsistenter Zustand auf Datenträger
    - d.h. alle Transaktionen vollständig auf Datenträger
  - Log kann gelöscht werden
  - danach wieder neue Transaktionen mit Protokollierung im Log
- ähnlich wird verfahren, wenn
  - Log zu groß wird
  - System heruntergefahren wird
- je kleiner Log File desto schneller die Fehlererholung
  - in jedem Fall schneller als komplette Dateisystemüberprüfung

Log-structured File System

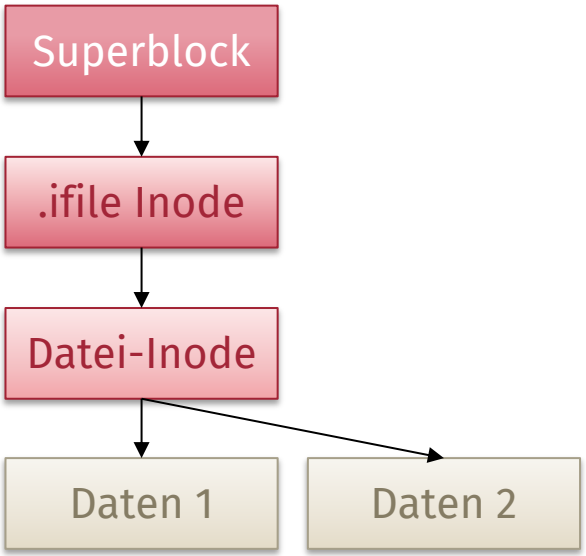
Atomare Änderungen im Dateisystem

- keine inkonsistenten Zustände möglich
- basiert auf atomarem Schreiben eines Datenblocks
  - selbst bei Stromausfall gewährleistet
- viele (Teil-)Änderungen mit einem Schreibvorgang sichtbar
- Beispiele
  - LinLogFS, BSD LFS, AIX XFS

Log-structured File System (2)

Teiländerungen erfolgen auf Kopien

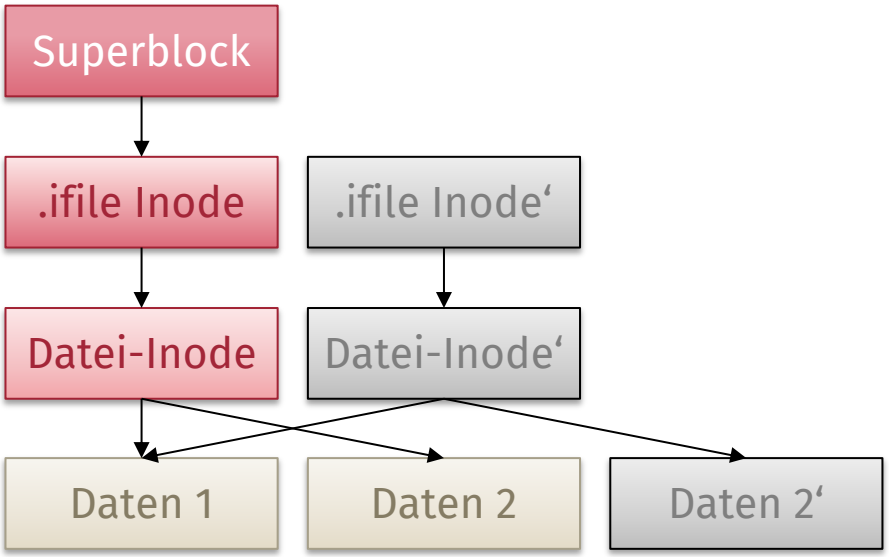
- Beispiel
  - Istzustand



Log-structured File System (3)

Teiländerungen erfolgen auf Kopien

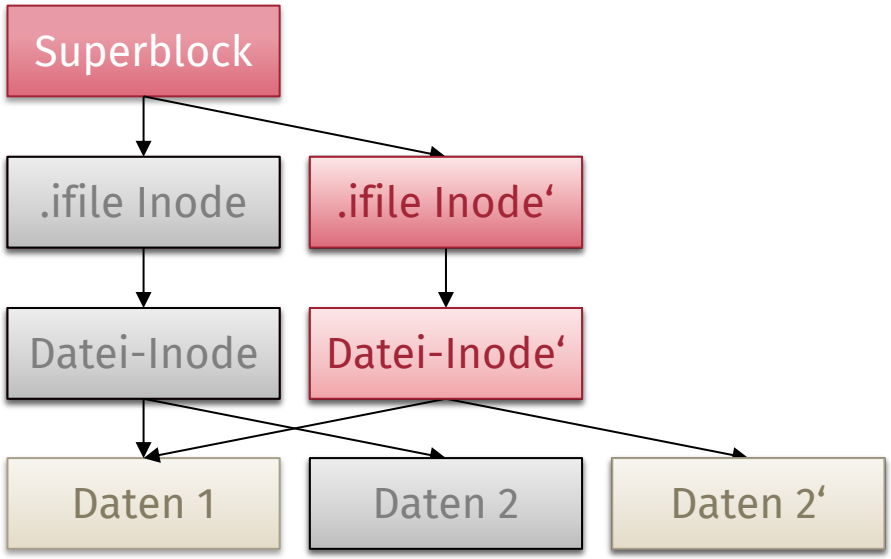
- Beispiel
  - Schritte zum Schreiben von Daten in Datenblock 2



Log-structured File System (4)

Teiländerungen erfolgen auf Kopien

- Beispiel
  - Abschlussschritt: Schreiben des geänderten Superblocks



Log-structured File System (5)

Vorteile

- Datenkonsistenz bei Systemausfällen
  - eine atomare Änderung macht alle Teiländerungen sichtbar
- Schnappschüsse (Checkpoints) einfach realisierbar
- gute Schreibeffizienz
  - alle zu schreibenden Blöcke werden kontinuierlich geschrieben

Nachteil

- Gesamtperformanz etwas geringer

Limitierung der Plattennutzung

Mehrbenutzersysteme

- einzelnen Benutzer mit verschieden große Kontingenten
  - gegenseitige Beeinflussung soll vermieden werden
  - z.B. ein Benutzer füllt den gesamten Datenträger

Quota-Systeme

- jeder Benutzer mit eigenem virtuellen Datenträger
  - Anzahl erlaubter Dateien und Verzeichnisse
  - Anzahl erlaubter Datenblöcke
- Verwaltung geeigneter Tabellen im Datensystem
  - automatische Fortschreibung bei Nutzertransaktionen

Limitierung der Plattennutzung (2)

Quota-Systeme (fortges.)

- Nutzer erhält „Disk full“ Meldung, wenn Quota verbraucht
- üblicherweise gibt es weiche und harte Grenze
  - weiche Grenze kann für bestimmte Zeit überschritten werden

Fehlerhafte Plattenblöcke

Blöcke, die beim Lesen Fehlermeldungen erzeugen

- z.B. Prüfsummenfehler

Hardwarelösung

- Plattencontroller bemerkt selbständig fehlerhafte Blöcke
  - werden maskiert
  - Zugriff automatisch umgeleitet auf einen „gesunden“ Block

Softwarelösung

- Dateisystem bemerkt selbständig fehlerhafte Blöcke
  - markiert diese als belegt und evtl. beschädigt

Inhaltsüberblick

Dateiverwaltung

- Aufbau von Platten
- Anwendersicht unter Linux
  - Operationen und Attribute
- Anwendersicht unter Windows
- Unix/Linux Dateisysteme
  - Mounten, Inodes
  - UFS, BSD 4.2, EXT2
- Windows Dateisysteme
  - FAT32, NTFS
- Zuverlässige Dateisysteme