Bedeutung

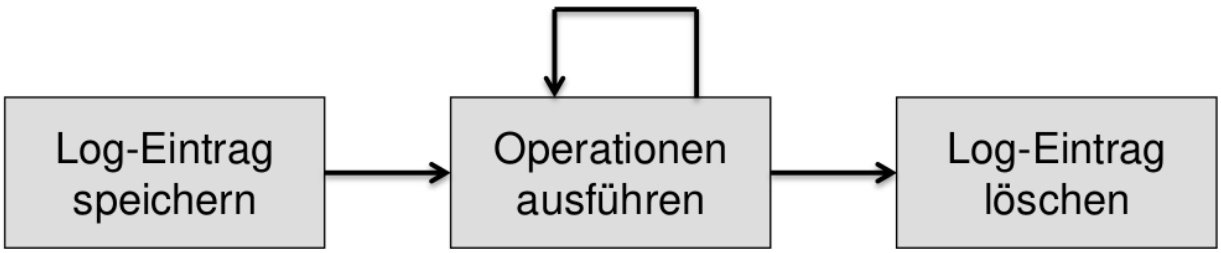
- Bereiche auf Medium werden mit Dateisystem versehen
- stellt Verwaltungsstruktur für Objekte (Dateien, Verzeichnisse, Hardlinks, Softlinks etc.) bereit
- der Verzeichnisbaum kann aus mehreren Dateisystemen zusammengebaut sein
  - die Verzeichnise `/boot/EFI` und `/home` sind oft auf einer anderen Partition und wurden da bei der Installation gemountet
- Dateisystem voll, wenn
  1. keine Datenblöcke mehr frei (überprüfbar mit `df -h`)
  2. keine I-Nodes mehr frei (überprüfbar mit `df -i`)

Realsierung von Dateien

1. **Sequentieller Zugriff:** Um auf einen bestimmten Datensatz zugreifen zu können, müssen alle Datensätze zwischen Start- und Zielposition besucht werden
2. **Wahlfreier Zugriff:** Beliebiges Festlegen der aktuellen Position durch Seek-Operation möglich

Jouranling Dateisysteme (ext3, ext4, NTFS)

- Problem: Inkonsistentes Dateisystem nach Systemabsturz. Inkonsistenzen entstehen, falls nicht alle Operationen ausgeführt wurden (Bsp. Löschen einer Datei unter Unix: Löschen der Datei aus derm Verzeichnis, Freigabe des I-Nodes, Freigabe der Plattenblöcke)
- Grundidee Journaling:
  - Log-Eintrag mit Operationen auf Platte speichern
  - Operationen ausführen
  - Log-Eintrag löschen



- Nach Systemabsturz können noch ausstehende Operationen durchgeführt werden

Beispielparition für ext3, ext4 (Linux):

Bootblock	Superblock	Freispeicherverw.	I-Nodes	Datenblöcke

Bootblock

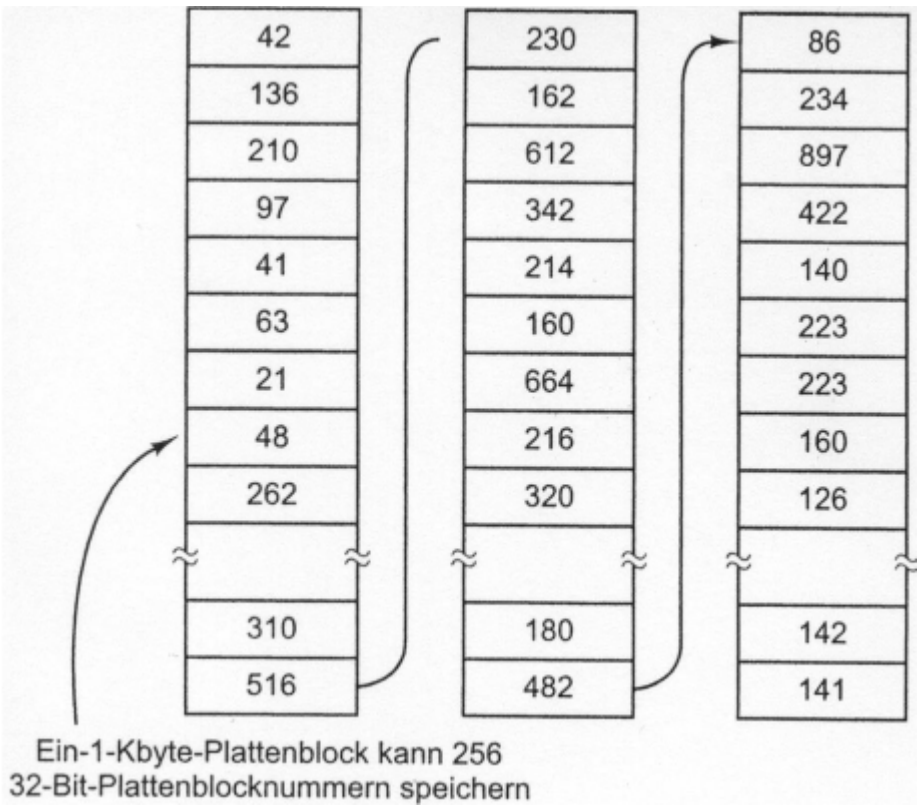
- initialisiert Dateisystem der Partition.

Superblock

- enthält Schlüsselparameter des Dateisystems (z.B. Schreibschutzmarkierung, Name des Dateisystemtyps, Anzahl Blöcker etc.)

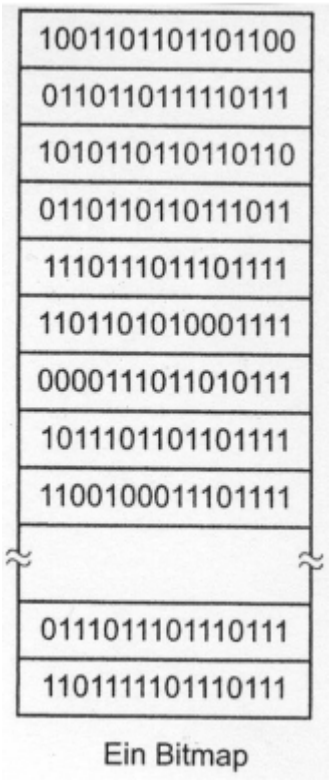
Freispeicherverwaltung

- Informationen über freie Blöcke im Dateisystem
- Datenblöcke werden nicht notwendigerweise an beliebigen Stellen im Dateisystem genutzt
- **Möglichkeiten:**
  - Freibereichsliste als verkettete Liste:



    - speichere Nummern von freien Plattenblöcken
    - Benutze zum Speichern der Nummern freie Plattenblöcke, die miteinander verkettet werden

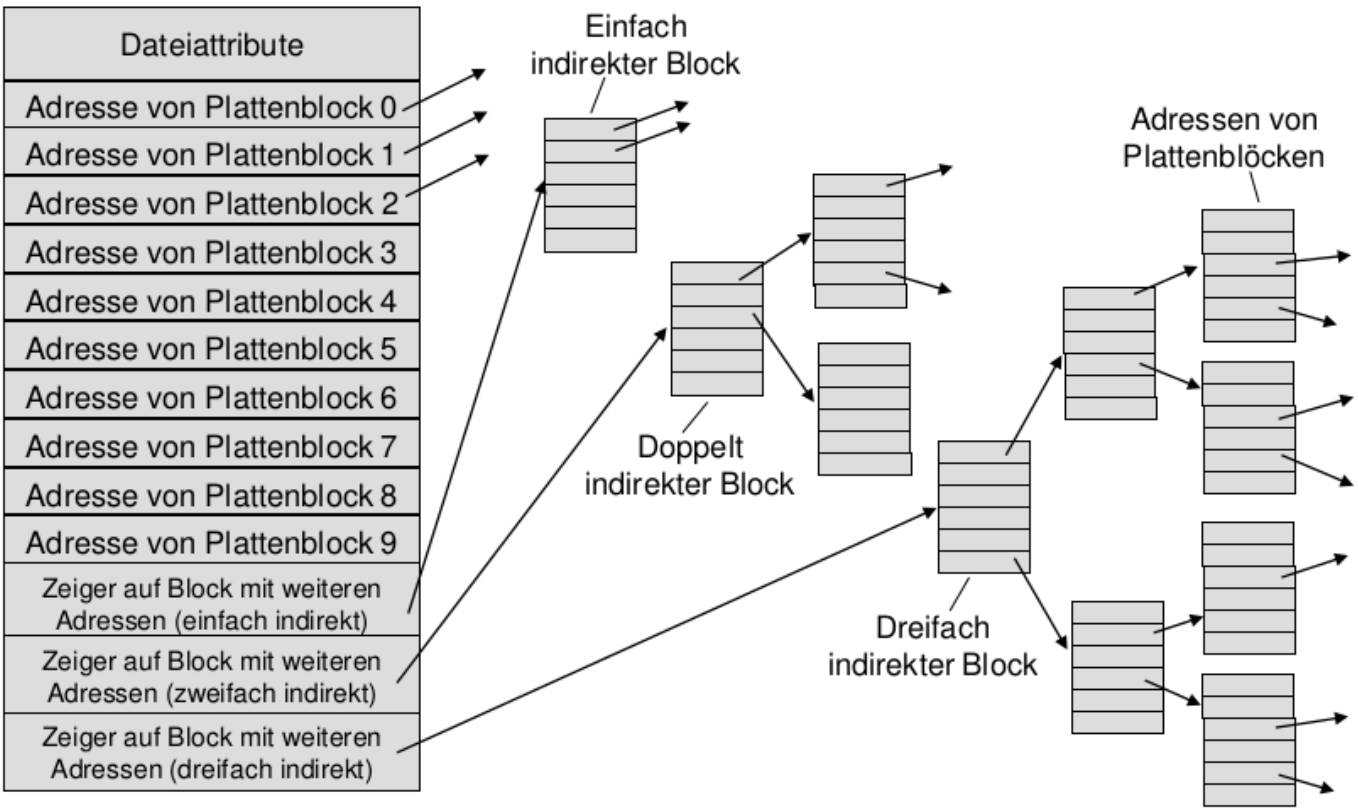
- Bitmap:



  - Bitmap mit 1 Bit für jeden Plattenblock
  - Plattenblock frei ⇔ Bit = 1

I-Nodes (Index-Node)

Beispiel für UNIX System V:



- Größe der I-Node-Tabelle bei Anlegen des Dateisystems festgelegt
- zu jeder geöffneten Datei steht I-Node im Hauptspeicher und nicht für alle Plattenblöcke
- **Inhalt:**
  - enthalten Metadaten (Attribute) der Dateien (Eigentümer, Zugriffsrechte, Dateityp, Größe, Linkzähler, etc.)
  - Adressen von Plattenblöcken (Verweis auf Dateiinhalt), ermöglich Zugriff auf alle Blöcke der Datei
    - 10 direkte Zeiger auf Datenblöcke
    - 1 Zeiger auf einfach indirekten Block, 1 Zeiger auf doppelt indirekten Block und 1 Zeiger auf dreifach indirekten Block (Zeiger auf Blöcke mit weiteren Adressen)
- k offene Dateien und I-Node benötigt n Bytes, dann insgesamt: $k \cdot n\mathrm{Byte}$ im Hauptspeicher
- **Maximale Dateigröße:**
  - $b =$ Blockgröße (z.B. $1KiB$), $z =$ Zeigergröße (z.B. $4Byte$)
  - Zeiger in einem Block: $\lfloor \frac{b}{z} \rfloor$

- Anzahl Datenblöcke: $N = 10 + \lfloor \frac{b}{z} \rfloor + \lfloor \frac{b}{z} \rfloor^2 + \lfloor \frac{b}{z} \rfloor^3$

  $\Rightarrow S = (10 + \lfloor \frac{b}{z} \rfloor + \lfloor \frac{b}{z} \rfloor^2 + \lfloor \frac{b}{z} \rfloor^3) \cdot b$

Datenblöcke

- Eigentliche Inhalte der Dateien / Verzeichnisse

FAT-12, FAT-16, FAT-32, VFAT

Inhalt	Bootsektor	FS Informations Bereich (nur FAT32)	Ggf. weitere reservierte Bereiche	File Allocation Table (FAT) Nummer 1	File Allocation Table Nummer 2 (& ggf. weitere)	Root-Verzeichnis (nur FAT12/FAT16)	Datenbereich (für Dateien und Ordnerstrukturen) ... (bis zum Ende der Partition)

Bootsektor

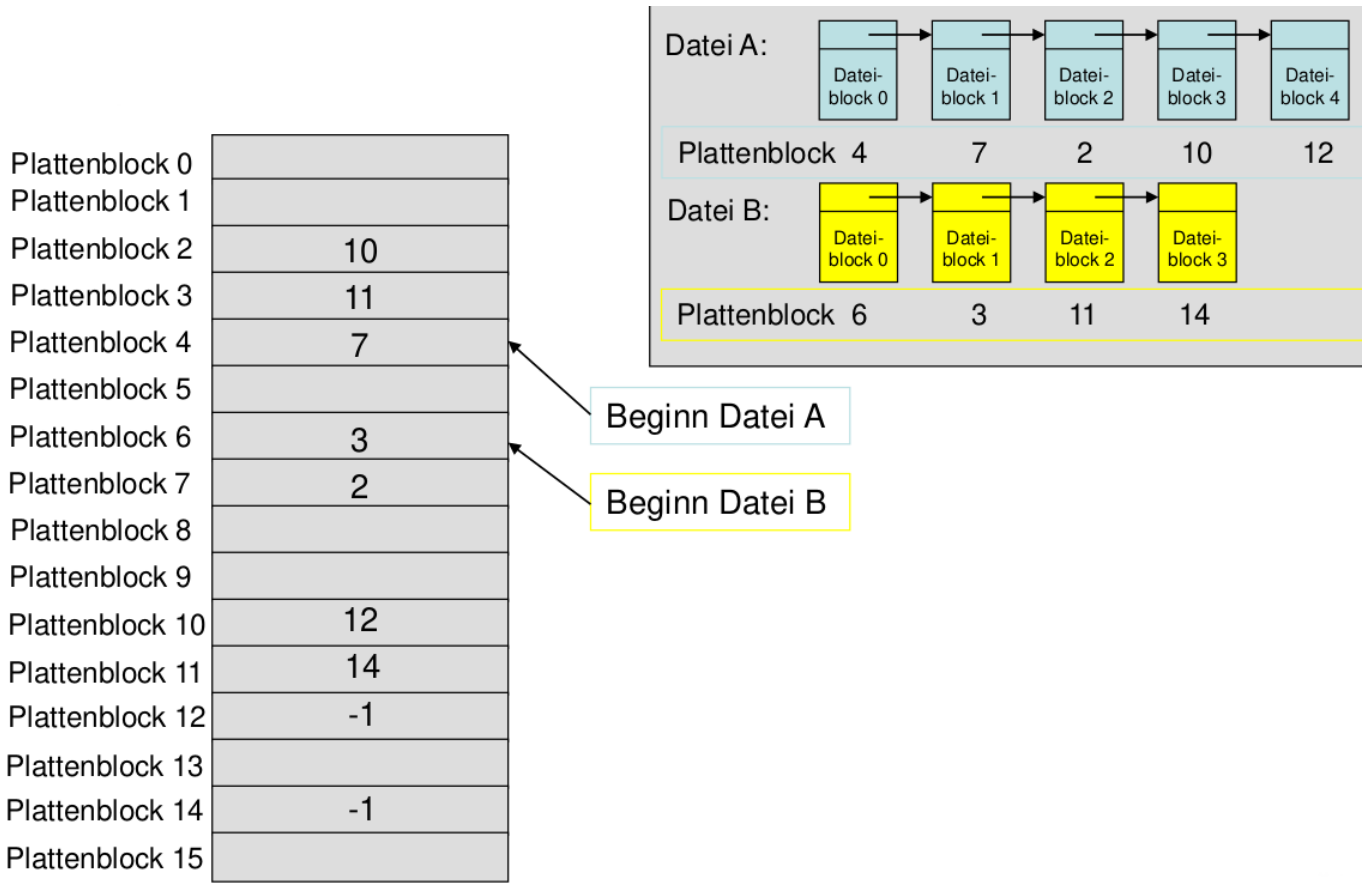
- der Bootsektor kann x86-Maschinencode (Bootloader) enthalten, der das Betriebssystem laden soll. An anderen Stellen enthält er Informationen über das FAT-Dateisystem

Reservierte Sektoren

- zwischen Bootsektor und der ersten FAT können Sektoren reserviert werden, die vom Dateisystem nicht benutzt werden. Dieser Bereich kann von einem Bootmanager oder für betriebssystemspezifische Erweiterungen genutzt werden.
- **Unterschiede:**
  - auf den meisten FAT12- oder FAT16-Dateisystemen existieren (außer dem Bootsektor) keine weiteren reservierten Sektoren. Die FAT folgt somit direkt im Anschluss an den Bootsektor
  - FAT32-Dateisysteme enthalten in der Regel noch einige Erweiterungen zum Bootsektor sowie eine komplette Sicherungskopie des Bootsektors und der Erweiterungen

FAT (File Allocation Table)

- hat Information über Verkettung der Blöcke auf der Platte (Dateien werden als verkettete Listen von Plattenblöcken gespeichert)
  - auch Datei-Allokationstabelle
  - FAT ist im Hauptspeicher abgelegt, daher muss bei **wahlfreiem Zugriff** auf Block n nur eine Kette von Verweisen im Hauptspeicher verfolgt werden



- **FAT-16:**
  - Anzahl Zeiger: 2^{16} Zeiger
  - Größe eines Zeigers: $16Bit = 2^1 B$
  - Größe der Zeiger im Hauptspeicher:
    - $2^{16} \cdot 2^1 B = 2^7 \cdot 2^{10} B = 128\text{KiB}$
  - Größe der verwaltbaren Partition:
    - Maximale Blockgröße: FAT-16 muss z.B. für eine $2GiB$-Partition eine Blockgröße von $32KiB$ verwenden. Andernfalls kann mit den 2^{16} verschiedenen Zeigern nicht die ganze Partition adressiert werden: $\frac{2GiB}{2^{16}} = \frac{2^1 \cdot 2^{30} B}{2^{16}} = \frac{2^{21} KiB}{2^{16}} = 2^5 KiB = 32KiB$
    - $2^{16} \cdot 32 \cdot 2^{10} B = 2^{16} \cdot 2^5 \cdot 2^{10} B = 2^1 \cdot 2^{30} B = 2GiB$
- **FAT-32:**
  - bei FAT-32: 28 Bit für Zeiger, 4 Bit für andere Zwecke, z.B. Markierung freier Blöcke
  - Anzahl Zeiger: 2^{28} Zeiger
  - Größe eines Zeigers: $32Bit = 4B = 2^2 B$
  - Größe der Zeiger im Hauptspeicher:
    - $2^{28} \cdot 4B = 2^{28} \cdot 2^2 B = 2^{30} B = 1GiB$

    - Größe der verwaltbaren Partition:
        - Maximale Blockgröße: 32KiB
        - $2^{28} \cdot 32 \cdot 2^{10} B = 2^{28} \cdot 2^5 \cdot 2^{10} B = 2^3 \cdot 2^{40} B = 8TiB$
- **VFAT (Virtual File Allocation Table):**
    - `VFAT` is a module (confirm withconfirm with `modinfo vfat`), used to mount the FAT file systems in Linux. The choosing of the driver determines how some of the features are applied to the file system
    - ist eine Erweiterung des FAT-Formats zur Verwendung langer Dateinamen, die auf FAT12, FAT16 und FAT32 angewendet werden kann
    - die Datei wird wie bisher als 8.3-Dateiname gespeichert, bei längeren Namen wird jedoch ein Alias in der Form `xxxxxx~1.xxx` verwendet, wobei die Nummer hochgezählt wird. Der lange Name wird dann über mehrere Verzeichniseinträge verteilt, die eine Kombination von Datei-Attributen aufweisen
    - das endgültige Format erlaubt bis zu 255 Zeichen lange Dateinamen (wobei der Name inklusive Speicherpfad bis zu 260 Zeichen enthalten kann)
- **Vorteil:** Keine Fragmentierung, ersetzt bei wahlfreiem Zugriff Plattenzugriffe durch schnellere Hauptspeicherzugriffe
- **Nachteil:** Größe der FAT im Speicher. Anzahl der Einträge = Gesamtzahl der Plattenblöcke. Auch wenn Platte fast komplett unbelegt. Maximale Dateigröße: `4GiB` (Grund: $4B = 4 \cdot 8Bit = 32Bit$ großes Feld für die Dateigröße in der Directory-Tabelle $\Rightarrow$ die größe darstellbare Zahl ist $2^{32} = 2^2 \cdot 2^{30} = 4GiB$)
- Gegenmaßnahmen für zu große FAT: Nur wirklich benötigten Teil verwalten (Fenster) und Fenster über der FAT, welches im Speicher bleibt bei Bedarf auswechseln
- Größe der Platte $\sim$ #Zeiger
- Kleinere Blöcke führen zu weniger verschwendetem Platz pro Datei (interne Fragmentierung). Je kleiner die Blockgröße, desto mehr Zeiger bei gleicher Dateigröße, desto größer die FAT im Hauptspeicher
- Maximale Größe des ganzen Dateisystems wird durch Zeigergröße und Blockgröße begrenzt

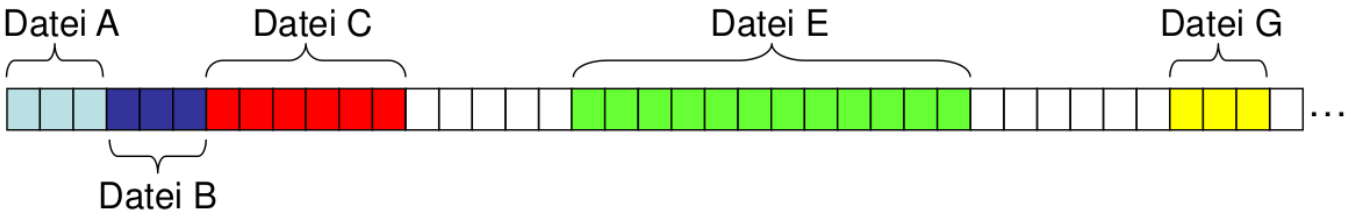
Stammverzeichnis und Unterverzeichnisse

- das Stammverzeichnis (**root directory**), auch Wurzelverzeichnis oder Hauptverzeichnis genannt, ist eine Tabelle von Verzeichniseinträgen
- jede Datei oder Unterverzeichnis wird in der Regel durch je einen Verzeichniseintrag repräsentiert
- die bei Windows 95 eingeführte Erweiterung (unter anderem VFAT) für "lange Dateinamen" benutzt jedoch ggf. mehrere Verzeichniseinträge pro Datei bzw. Verzeichnis, um die langen

Dateinamen unterzubringen

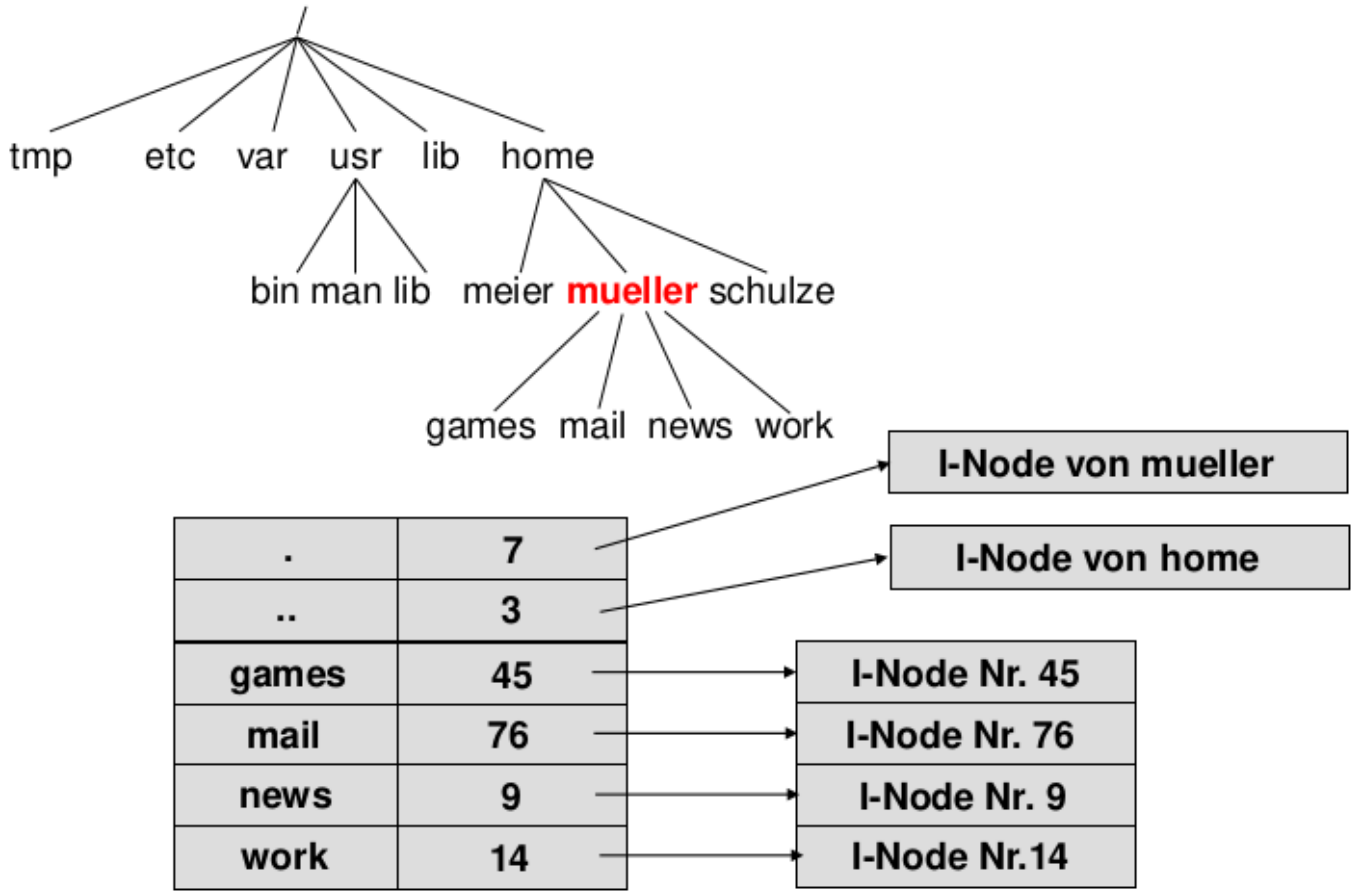
- **Unerschiede:**
  - das Stammverzeichnis folgt bei FAT12 und FAT16 direkt der FAT und hat eine feste Größe und damit eine Maximalanzahl an Verzeichniseinträgen
  - bei FAT32 hat das Stammverzeichnis eine variable Größe und kann an einer beliebigen Position des Datenbereichs beginnen

Dateisysteme für CD-ROMs



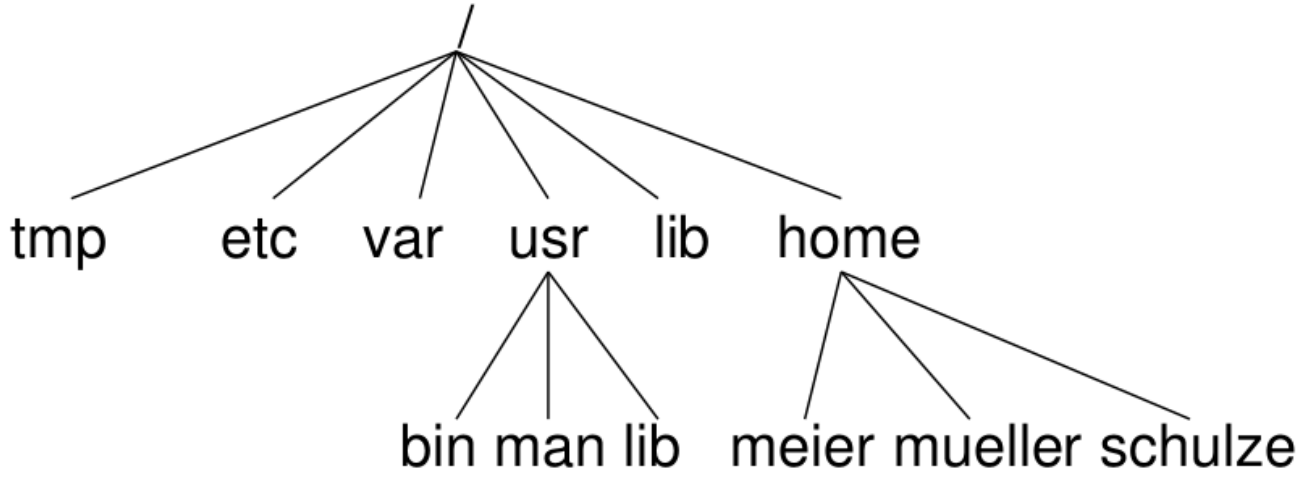
- Abspeicherung von Dateien als zusammenhängende Menge von Plattenblöcken
- **Vorteil:** Lesegeschwindigkeit
- **Nachteil:** externe Fragmentierung der Platte, durch möglicherweise entstehende Löcher (Defragmentierung möglicherweise notwendig)

Verzeichnisse



- Verzeichnisse sind Systemdateien (also nur spezielle Dateien) zur Strukturierung des Dateisystems
- liefert Abbildung von Datei- bzw. Verzeichnisnamen auf I-Node-Nummern. Jeder Verzeichniseintrag ist ein Paar aus Name und I-Node-Nummer. Über I-Nodes kommt man dann zu Dateiinhalten
- im Verzeichnis `/dev` findet man Abstraktionen von I/O-Geräten
  - z.B. Datenträger unter `dev/sda1`

Verzeichnisbaum



- root directory: `/`
- Abolsute Pfade: `/home/user/.config`
- Relative Pfade: `./.config` (beziehen sich auf das aktuelle working directory)

Mounten und Unmounten

→ Linux_Usage

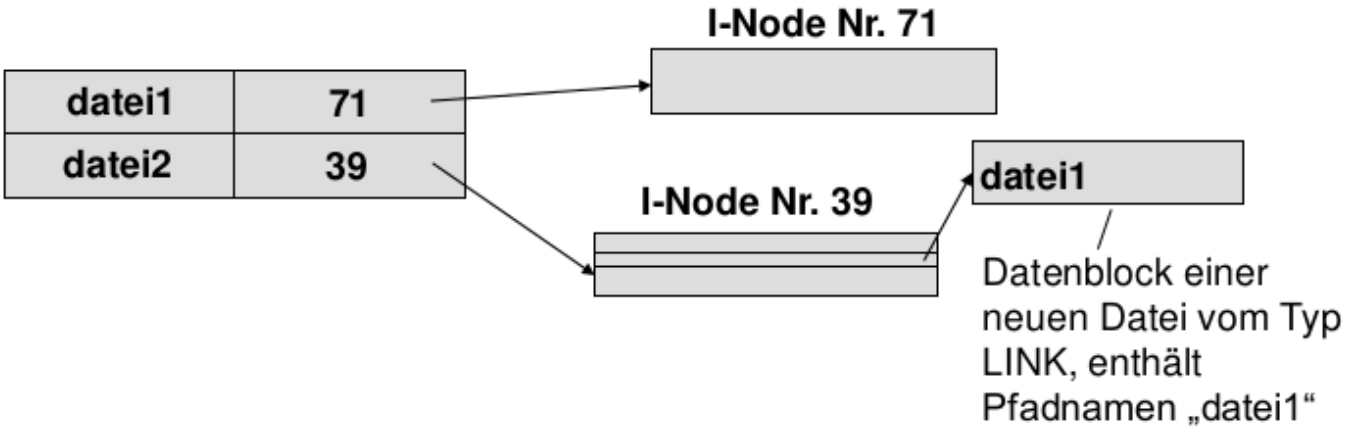
Links

- ansprechen des selben Objekts mit mehreren Namen

Symbolischer Link

- `ln -s /pfad/quelldatei /pfad/linkname`

```
$ ls -l
-rw-r----- 1 meier users … datei1
$ ln -s datei1 datei2
$ ls -l
-rw-r----- 1 meier users ... datei1
lrwxrwxrwx 1 meier users ... datei2 -> datei1
```

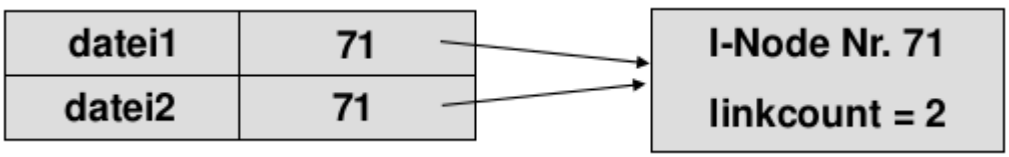


- Löschen/Verschieben/Umbennen des Referenzobjekts: Link
- zeigt ins Leere, Löschen des Links: Referenzobjekt unverändert
- Rechte am Link: bleiben immer gleich, nämlich `lrwxrwxrwx`, da symbolische Links eigene I-Nodes haben
- `chmod` command wird auf Linkziel angewandt

Hardlink

- `ln /pfad/quelldatei /pfad/linkname`

```
$ ls -l
-rw-r----- 1 meier users … datei1
$ ln datei1 datei2
$ ls -l
-rw-r----- 2 meier users … datei1
-rw-r----- 2 meier users … datei2
```



- es gibt einen Linkzähler: Dateiobjekt mit n Hardlinks hat Linkzähler $n + 1$
- Löschen eines Links: Dekrementieren des Linkzählers, Löschen des Dateiobjekts: Dekrementieren des Linkzählers, Dateiobjekt wird erst dann wirklich gelöscht, wenn Linkzähler auf 0
- Rechte am Link: wie beim Referenzobjekt (Rechte aller Hardlinks ändern sich mit, da die Rechte im I-Node gespeichert werden und die Datei und die Hardlinks auf das selbe I-Node zeigen)

Kopieren von Links

- `cp link? /tmp`: es wird die Datei hinter dem Hardlink / Softlink kopiert mit neuem I-Node und nicht der Hardlink / Softlink

Resources

- https://wiki.archlinux.org/index.php/Media_Transfer_Protocol (mount android)