



**Hochschule
Augsburg** University of
Applied Sciences

Vorlesung: Betriebssysteme

Dateisysteme

Prof. Dr. Lothar Braun, Dr.-Ing. Volodymyr Brovko
Sommersemester 2024

Aufgaben eines Dateisystems

Logische Organisation

Physikalische Organisation

Dateisysteme unter Linux

Aufgaben eines Dateisystems

Logische Organisation

Physikalische Organisation

Dateisysteme unter Linux

- **Logische Organisation**

- Dateien erzeugen, löschen, lesen, schreiben
- Verzeichnisse / Namensraum
- Spezialdateien
 - Links, Gerätedateien, Pipes, ...
- Zugriffskontrolle
 - Benutzer darf festlegen **wer** mit seinen Dateien **was** tun darf
 - Administrator darf festlegen, wer Dateien anlegen / lesen / ändern darf

- **Physikalische Organisation**

- Verteilen von Dateien und Meta-Daten auf dem Datenträger
- Standardisierter Zugriff auf E/A-Geräte

- **Effizienz**

- Anforderungen je nach Anwendungen, z.B.:
 - sequentieller \leftrightarrow randomisierter Zugriff
 - lesender \leftrightarrow schreibender Zugriff
- Physikalische Aspekte
 - Festplatte: Aufwendige Spurwechsel vermeiden
 - SSD: Häufiges Schreiben in gleichen Speicherblock vermeiden

- **Robustheit**

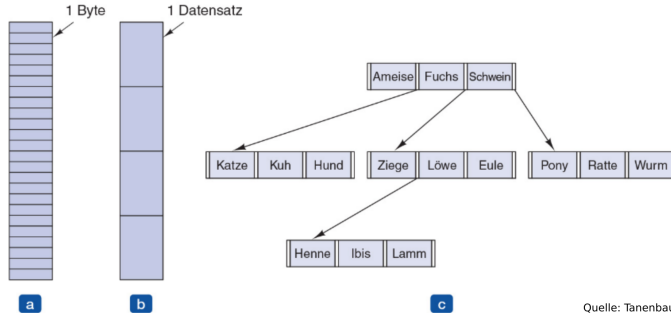
- Stromausfall
- defekte Sektoren
- ...

Aufgaben eines Dateisystems

Logische Organisation

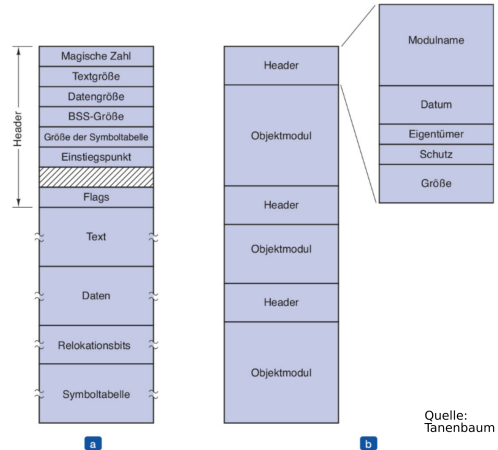
Physikalische Organisation

Dateisysteme unter Linux



a) Byte-Sequenz b) Folge von Datensätzen c) Baum

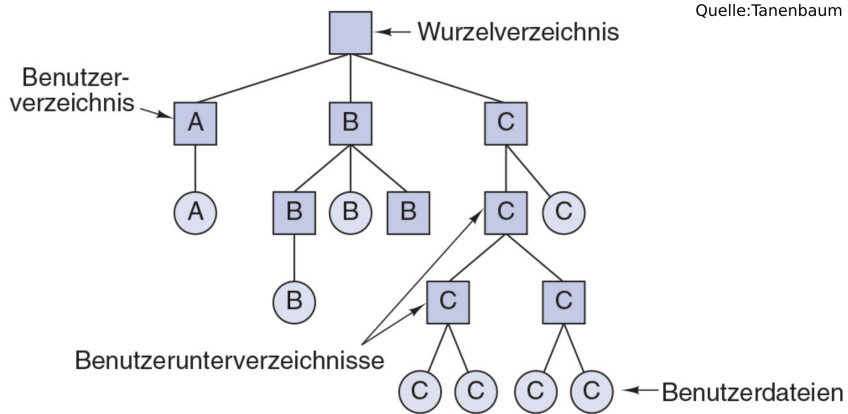
- **Reguläre Dateien**
 - Betriebssystem (Loader) muss nur Format ausführbarer Dateien kennen
- **Verzeichnis**
- **Geräte-Datei**
- **Pipe**
- **Link (Verknüpfung)**



a) Ausführbare Datei
b) Archiv

Quelle:
Tanenbaum

Attribute	Bedeutung
Schutz	Wer kann wie auf die Datei zugreifen?
Passwort	Passwort für den Zugriff auf die Datei
Urheber	ID der Person, die die Datei erzeugt hat
Eigentümer	Aktueller Eigentümer
Read-only-Flag	0: Lesen/Schreiben; 1: nur Lesen
Hidden-Flag	0: normal; 1: in Listen nicht sichtbar
System-Flag	0: normale Datei; 1: Systemdatei
Archiv-Flag	0: wurde gesichert; 1: muss noch gesichert werden
ASCII/Binär-Flag	0: ASCII-Datei; 1: Binärdatei
Random-Access-Flag	0: nur sequenzieller Zugriff; 1: wahlfreier Zugriff
Temporary-Flag	0: normal; 1: Datei bei Prozessende löschen
Sperr-Flags	0: nicht gesperrt; nicht null: gesperrt
Datensatzlänge	Anzahl der Bytes in einem Datensatz
Schlüsselposition	Offset des Schlüssels innerhalb des Datensatzes
Schlüssellänge	Anzahl der Bytes im Schlüsselfeld
Erstellungszeit	Datum und Zeitpunkt der Dateierstellung
Zeitpunkt des letzten Zugriffs	Datum und Zeitpunkt des letzten Zugriffs
Zeitpunkt der letzten Änderung	Datum und Zeitpunkt der letzten Dateiänderung
Aktuelle Größe	Anzahl der Bytes in der Datei
Maximale Größe	Anzahl der Bytes für maximale Größe der Datei

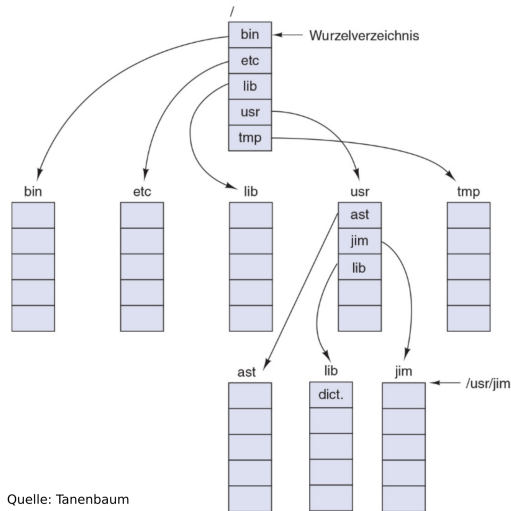


- **Dateiname setzt sich aus Gerätenamen, Verzeichnis und Datei zusammen**
 - C:\windows\system32\cmd.exe
 - D:\movie.mp4
 - \\server\freigabe\datei.txt
- **Einhängen (“mounten”) aller verwendeter Dateisysteme in einen logischen Baum**

```
$ mount
sysfs on /sys type sysfs (rw,nosuid,nodev,noexec,relatime)
proc on /proc type proc (rw,nosuid,nodev,noexec,relatime)
udev on /dev type devtmpfs (rw,nosuid,noexec,relatime,size=7995092k,nr_inodes=1998773,mode=755)
devpts on /dev/pts type devpts (rw,nosuid,noexec,relatime,gid=5,mode=620,ptmxmode=000)
tmpfs on /run type tmpfs (rw,nosuid,nodev,noexec,relatime,size=1608316k,mode=755)
/dev/mapper/vgubuntu-root on / type ext4 (rw,relatime,errors=remount-ro)
...
```

- **Vorteil: Trennung von *logischer* und *physikalischer* Organisation**
 - Verschieben von Dateien auf andere Datenträger ohne Änderung des logischen Dateinamen

- **Jeder Prozess besitzt ein *aktuelles Arbeitsverzeichnis***
 - Aktuelles Arbeitsverzeichnis mit Befehl: *pwd* erfahrbar
 - Wenn kein absoluter Pfad als Dateiname angegeben → Dateiname relative zum aktuellen Verzeichnis
- **Jedes Verzeichnis besitzt zwei zusätzliche Einträge**
 - “.” zeigt auf das Verzeichnis
 - “..” zeigt auf das übergeordnete Verzeichnis



Quelle: Tannenbaum

- **Relevante Frage für Schutz und Rechte: *Wer darf was?***

- **Was darf getan werden?**

- lesen
- schreiben
- ausführen
- sehen
- erzeugen
- löschen
- Attribute ändern

- **Wer darf etwas tun?**

- Eigentümer
- bestimmte benannte Benutzer
- Gruppe von Benutzern
- alle

- **Konzept: Jede Datei enthält Liste mit Rechte-Paaren**
 - Eintrag in Liste: (Benutzer-ID, Rechte-Flags)
- **Vorteil:**
 - Rechte können sehr detailliert festgelegt werden
- **Nachteil:**
 - Speicheraufwand für ACLs
 - Komplexe Listen können Nutzer überfordern
 - Beispiel NTFS: viele Zugriffsmodi, Vererbung von Rechten auf Unterverzeichnisse, Negativ-Rechte, ...

- **Wer darf etwas mit einer Datei tun?**
 - Datei hatte Eigentümer (UID) und Gruppe (GID)
 - Benutzer können Mitglied mehrerer Gruppen sein
- **Was darf mit Dateien getan werden?**
 - Rechte werden definiert für *Eigentümer*, *Gruppe*, und *Andere*
 - Existierende Rechte:
 - *r* - Erlaubt lesen
 - *w* - Erlaubt schreiben
 - *x* - Erlaubt ausführen (bei Verzeichnis: erlaubt Wechsel in Verzeichnis)
 - Zusätzliche Flags:
 - *set-UID* - bewirkt dass Datei mit Rechten des Eigentümers ausgeführt wird
 - *set-GID* - bewirkt dass Datei mit Rechten der Gruppe ausgeführt wird
- **Ändern von Rechten: *chmod(3)* oder *chown(3)***

Binär	Symbol	Erlaubte Dateizugriffe
111000000	rwX-----	Eigentümer darf lesen, schreiben und ausführen
111111000	rwXrwX---	Eigentümer und Gruppe dürfen lesen, schreiben und ausführen
110100000	rw-r-----	Eigentümer darf lesen und schreiben; Gruppe darf lesen
110100100	rw-r-r--	Eigentümer darf lesen und schreiben; alle anderen dürfen lesen
111101101	rwXr-Xr-X	Eigentümer darf alles; andere dürfen lesen und ausführen
000000000	-----	Keiner hat irgendwelche Rechte
000000111	-----rwx	Nur Außenstehende dürfen zugreifen (merkwürdig, aber korrekt)

Aufgaben eines Dateisystems

Logische Organisation

Physikalische Organisation

Dateisysteme unter Linux

- **Verwaltung von**

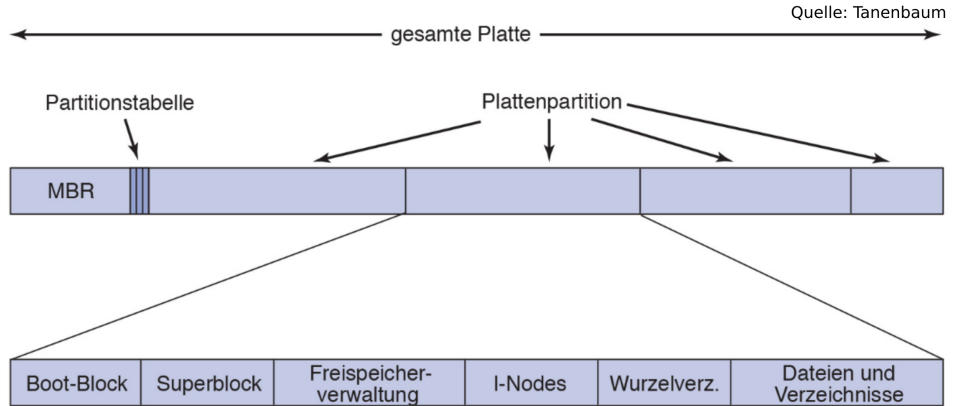
- Dateien
- Verzeichnissen
- freiem Platz
- fehlerhaften Sektoren

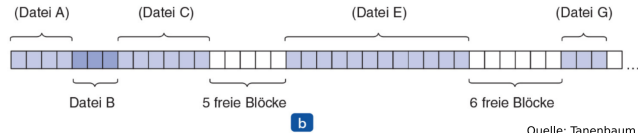
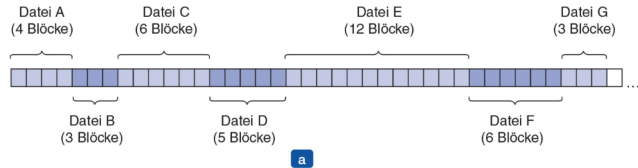
- **Effizienz**

- Fragmentierung vermeiden, Dateien möglichst zusammenhängend ablegen
- schneller Zugriff auf Verzeichnisse, Dateien und Daten

- **Robustheit**

- Toleranz gegenüber Fehlern
- Beispiele: Stromausfall beim Schreiben, Ausfall von wichtigen Sektoren

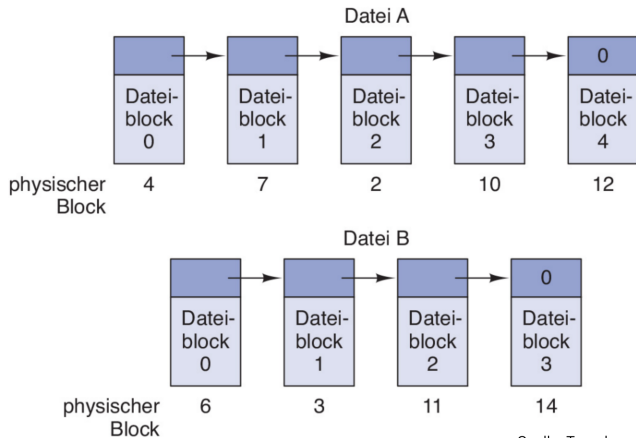




Quelle: Tanenbaum

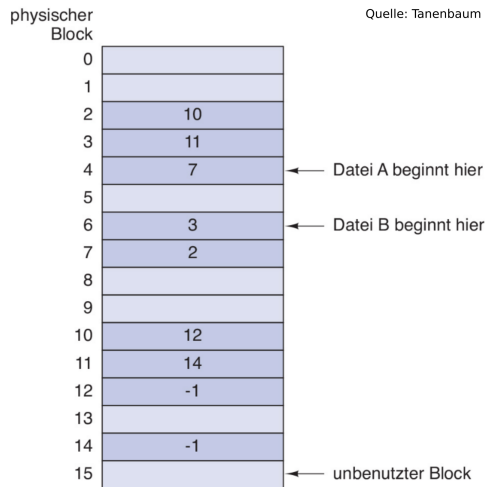
- a) Zusammenhängende Belegung des Plattenplatzes für sieben Dateien
- b) Nach Löschen von D und F

- **Probleme:**
 - Fragmentierung bei Löschen und Anlegen von Dateien
 - Was wenn Dateien wachsen?



Speicherung von Dateien als Liste von Plattenblöcken

- **FAT: “File Allocation Table”**
- **Kopie der Tabelle wird im Speicher gehalten**
 - Schneller Zugriff auf Blöcke in Dateien
- **Vorteile:**
 - Schnelles Springen (seek) möglich
 - Löschen sehr effizient
- **Nachteil:**
 - Hauptspeicherbedarf: Tabelle wächst mit Größe der Platte

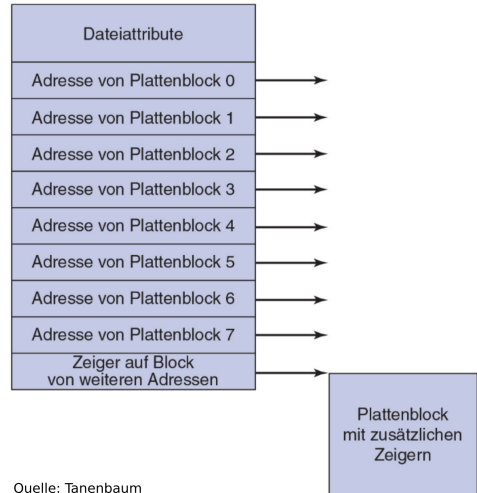


- **Eine Datenstruktur pro Datei**

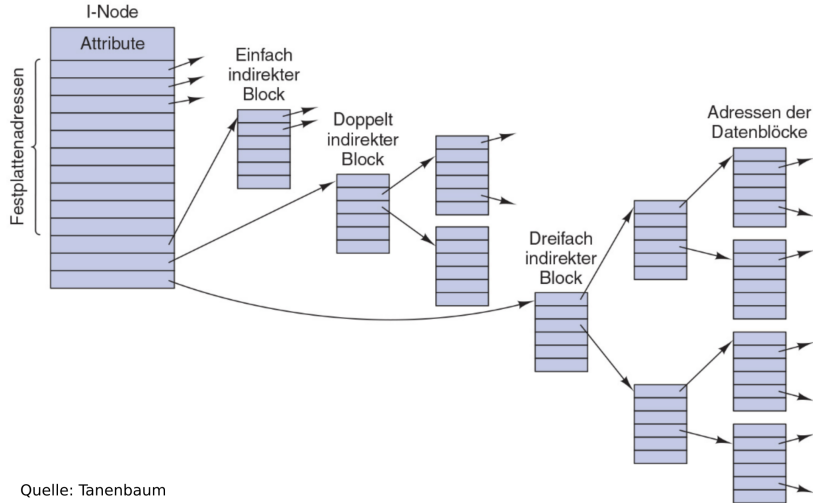
- Beinhaltet Dateiattribute
- Liste von Blöcken der Datei

- **Vorteile:**

- I-Node muss nur im Speicher gehalten werden, wenn Datei offen
- Hauptspeicherbedarf unabhängig von Plattengröße



Quelle: Tanenbaum

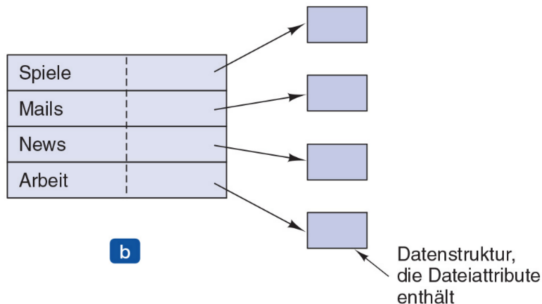


- **Auch hier: Unterschiedliche Realisierungen existieren**
 - Name und Dateiattribute in einem Verzeichniseintrag (fester Länge)
 - Name und Verweis auf I-Node

Spiele	Attribute
Mails	Attribute
News	Attribute
Arbeit	Attribute

a

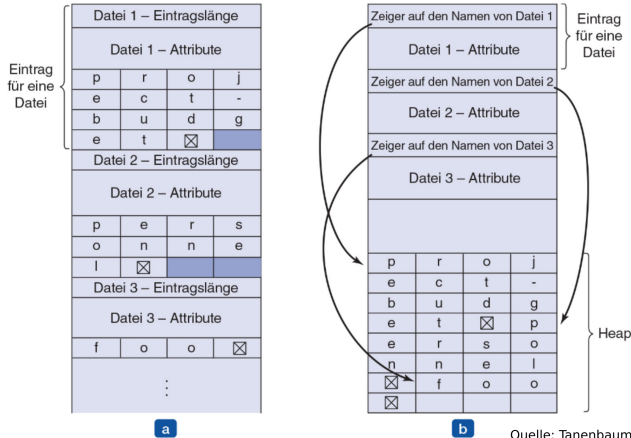
Quelle: Tanenbaum



b

a) Beispiel für Einträge bei FAT

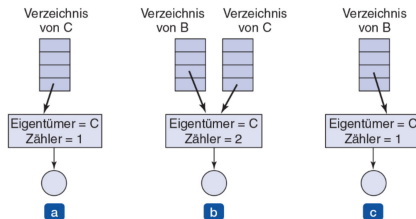
b) Verzeichnis enthält Referenz auf I-Nodes, mit allen Informationen. Beispiele: Ext2/3/4 und NTFS



a) Dateinamen im Eintrag
b) Namen auf einem Heap

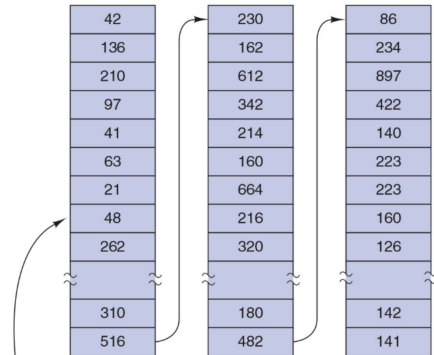
- **Mehrere Möglichkeiten Speicherung von Dateinamen variabler Länge**
 - Im Eintrag (benötigt typischerweise Padding)
 - Auf einem separaten Heap
- **Verwendung von Heap: Weniger Fragmentierung**
- **Lineare Suche in Dateieinträgen → langsam für große Verzeichnisse**
 - Optimierung durch Hashtabellen oder Caches

- **Symbolischer Link (*Soft Link*)**
 - Reguläre Datei mit Attribut *Soft Link*
 - Inhalt der Datei: Pfadname der Zieldatei
- **Hard Link**
 - Mehrere Verzeichniseinträge zeigen auf die gleiche physikalische Datei



- a) Vor Verlinkung
b) Nach Erzeugung des Links
c) Nach Löschen durch Eigentümer

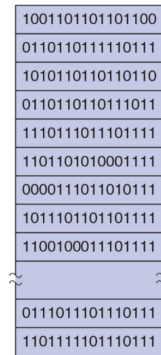
Freie Plattenblöcke: 16, 17, 18



Ein 1-KB-Plattenblock kann 256
32-Bit-Plattenblocknummern speichern

a

Quelle: Tanenbaum



Eine Bitmap

b

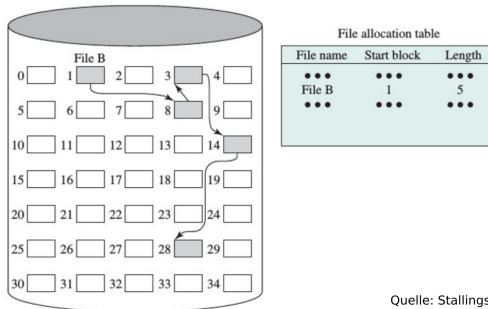
1) Verkettete Liste

- Länge proportional zur Anzahl freier Blöcke
- Benötigt mehr Speicher wenn Platte leer

2) Bitmap

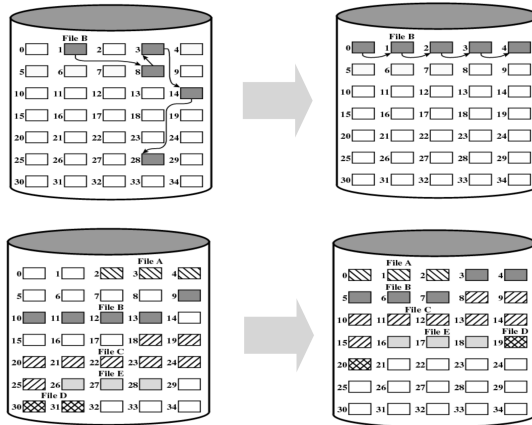
- Speicherbedarf: 1 Bit / Block
- Benötigt meistens weniger Speicher
- Zusammenhänge Blöcke einfach zu finden

- **Über Zeit hinweg: freier Platz über viele kleine freie Bereiche verteilt**
 - Alte Dateien werden gelöscht, neue kleinere angelegt
- **Problem**
 - Festplatten müssen viele aufwändige Kopfbewegungen machen um Daten zu lesen
 - SSDs: Problem ist kleiner da keine physikalische Kopfbewegung



Quelle: Stallings

- **Umsortieren der Blöcke der Dateien zu zusammenhängenden Bereichen**
 - Relevant für Festplatten und Dateisystemen die zu hoher Fragmentierung tendieren
 - Kontraproduktiv für SSDs: häufige Verwendung sorgt für hohen Verschleiß



- **Pre-Allokation**

- Allokation von mehreren zusammenhängenden Blöcken im voraus
- Beim Schließen der Datei: Freigabe überzähliger Blöcke

- **Block- bzw. Zylinder-Gruppen**

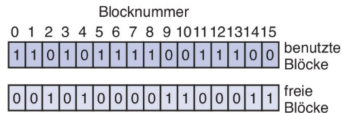
- Dateien und deren Attribute werden bevorzugt innerhalb einer Gruppe, die auch das Mutterverzeichnis enthält, abgelegt.
- Top-Level-Verzeichnisse werden auf verschiedene Gruppen verteilt

- **Inkonsistenz bedeutet dass Verwaltungsdaten nicht mit Daten übereinstimmen**
- **Potentielle Beispiele für inkonsistente Zustände**
 - Die Freiliste enthält einen Block der von Datei belegt wird
 - Freier Block ist nicht in der Freiliste enthalten
 - Datenblöcke werden von mehreren Dateien verwendet
 - Referenzzähler von *Hard Links* stimmt nicht mit tatsächlicher Anzahl überein
 - zu groß → Blöcke werden am Ende nicht freigegeben
 - zu klein → Block wird gelöscht obwohl noch benötigt

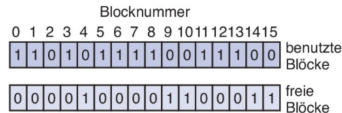
- **Systemabsturz oder Stromausfall**
 - I-Node geschrieben, Freispeicher-Bitmap noch nicht
 - I-Node geschrieben, Verzeichnis-Eintrag noch nicht
 - I-Node geschrieben, nachgeordnete Belegungsblöcke noch nicht
 - ...
- **Gerät oder Medium im Betrieb entfernt**
 - Häufig: USB-Geräte vor “sicher entfernen” ausgesteckt
- **Hardware-Fehler**
 - Sektor defekt
 - Bitfehler (entweder auf Platte oder im RAM)
 - ...
- **Bug im Dateisystem-Treiber**

- **Überprüfung der Block-Belegung**

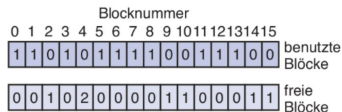
- Durchlaufen aller Dateien: Zählen der Referenzen pro Block
- Durchlaufen der Freiliste



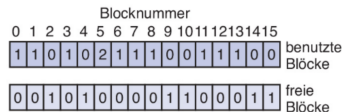
a



b



c



d

Quelle: Tanenbaum

a) Konsistent b) fehlender Block c) Block doppelt als frei markiert d) doppelt belegter Block

- **Ziel**
 - Dateisystem nach Stromausfall schnell in einen konsistenten Zustand bringen
- **Vorgehen bei einer Datei-Operation (Transaktion)**
 - 1) Geänderte Blöcke werden erst in einen separaten Bereich - das Journal - geschrieben
 - 2) *Commit*-Markierung in Journal setzen.
 - 3) Nach dem Ende (*commit*) der Transaktion: Blöcke in das Dateisystem schreiben
 - 4) Blöcke im Journal löschen
- **Nach einem Stromausfall**
 - Fall a) Journal ist vollständig (*committed*)
 - Übertrage Änderungen aus Journal in das Dateisystem
 - Fall b) Journal ist unvollständig
 - Verwerfe Journal
 - **Ergebnis:**
 - Datei-Operation wurde entweder komplett oder gar nicht ausgeführt
 - Dateisystem ist in jedem Fall in einem konsistenten Zustand

- **Zugriff auf Sekundärspeicher langsam → Optimierungen existieren**
- **Optimierungen haben unterschiedliche Ziele:**
 - Caching
 - Vorrorausschauendes Lesen
 - Reduzierung von Plattenarmbewegungen

- **Bei Leseanfragen Prüfung ob Block im Cache**
 - Falls ja: Ausliefern aus Cache
 - Falls nein: Laden von Block in Cache, Antwort dann aus Cache
- **Schnelle Suche im Cache: Hashtabellen**
 - Schlüssel besteht aus Geräte-ID und Adresse des Blocks
- **Ersetzen der Daten im Cache**
 - Gleiche Algorithmen wie bei virtueller Speicherverwaltung zur Auslagerung verwendbar
 - Aber: Reiner LRU nicht unbedingt wünschenswert
 - I-Nodes sollen häufig geschrieben werden um Konsistenz zu bewahren
- **Erzwingen von zurückschreiben von Caches:**
 - Linux / Unix: *sync*
 - Windows: *FlushFileBuffer*

- **Dateien werden häufig sequentiell gelesen**
 - Wahrscheinlichkeit dass Block $n + 1$ nach Block n gelesen wird ist hoch
 - Cache-Trefferquote kann erhöht werden wenn gleich mehr Blöcke aus Datei liest
 - Problem: Nicht-sequentielles Lesen kann schaden
- **Vermeidung von Plattenarmbewegungen**
 - Häufig verwendete Blöcke sollen nahe beieinander liegen
 - Irrelevant für SSDs

Aufgaben eines Dateisystems

Logische Organisation

Physikalische Organisation

Dateisysteme unter Linux

- **MINIX-Dateisystem**
 - ursprüngliches Dateisystem in Linux
 - Beschränkungen: Dateinamen ≤ 14 Zeichen und Dateigrößen $\leq 64\text{MB}$
- **Ext**
 - Dateinamen bis 255 Zeichen, Dateien bis 2 GB
 - langsamer als das MINIX-Dateisystem
- **Ext2**
 - bessere Performance
 - lange Zeit das Standard-Dateisystem unter Linux
- **Ext3**
 - Journaling
- **Ext4**
 - Hauptsächlich Performance-Verbesserungen
- **Viele weitere Dateisysteme**

- **Dateien sind Folge von 0 oder mehr Bytes**

- Es wird (bei regulären) Dateien nicht zwischen Binär- oder sonstigen Dateien unterscheiden
- Dateinamen auf 255 Zeichen begrenzt, alle Zeichen aus (NULL und /) erlaubt
- Konventionen: Dateiname und Dateiendung (z.B. prog.c oder prog.py)
- Aber: Hoheit über Interpretation liegt nur bei der verarbeitenden Anwendung

Verzeichnis	Inhalte
bin	Binärdateien (ausführbar)
dev	Spezialdateien für Ein-/Ausgabegeräte
etc	Verschiedene Systemdateien
lib	Bibliotheken
usr	Benutzerverzeichnisse

- **Ziel: Einheitliches Verzeichnis-Layout in UNIX-Systemen**
 - Einheitliches Verzeichnis-Layout in UNIX-Systemen
- **/bin/, /lib**
 - Essentielle System-Programme, -Bibliotheken
- **/usr: (/usr/bin/, /usr/lib)**
 - Installierte Benutzer-Programme mit Bibliotheken
- **/usr/local, /opt**
 - wie /usr, für manuell installierte Programme oder Pakete
- **/var (/var/spool, /var/tmp,/ /var/mail, ...)**
 - Variable Daten (z.B. Drucker-Spooling)
- **/etc**
 - Konfigurationsdateien (Host-Ebene)
- **/home/<benutzer>/**
 - Benutzer-Daten (z.B. Dokumente)

Systemaufruf	Beschreibung
<code>fd = creat(name, mode)</code>	Eine Möglichkeit, eine Datei zu erzeugen
<code>fd = open(file, how, ...)</code>	Datei zum Lesen, Schreiben oder beidem öffnen
<code>s = close(fd)</code>	Offene Datei schließen
<code>n = read(fd, buffer, nbytes)</code>	Daten einer Datei in einen Puffer lesen
<code>n = write(fd, buffer, nbytes)</code>	Daten von einem Puffer in einer Datei speichern
<code>position = lseek(fd, offset, whence)</code>	Dateizeiger bewegen
<code>s = stat(name, &buf)</code>	Statusinformationen einer Datei holen
<code>s = fstat(fd, &buf)</code>	Statusinformationen einer Datei holen
<code>s = pipe(&fd[0])</code>	Pipe erzeugen
<code>s = fcntl(fd, cmd, ...)</code>	Dateisperren und andere Operationen

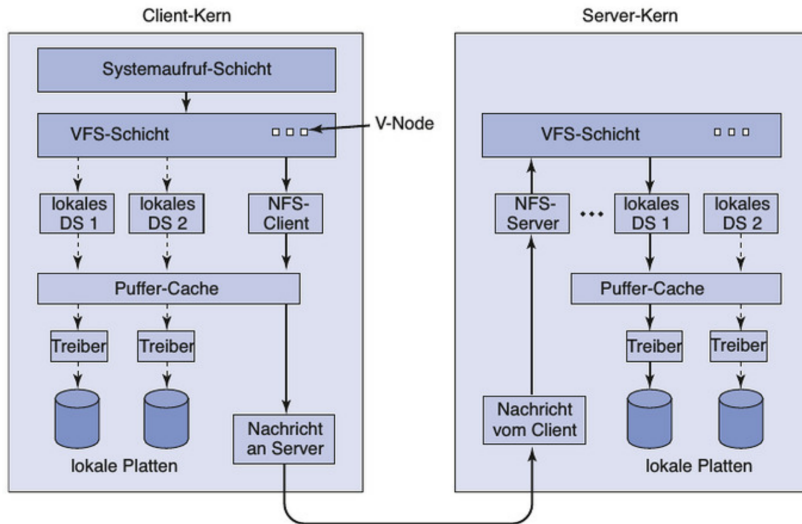
Auswahl an Aufrufen, Rückgabewert im Fehlerfall -1

Systemaufruf	Beschreibung
<code>s = mkdir(path, mode)</code>	Neues Verzeichnis erzeugen
<code>s = rmdir(path)</code>	Verzeichnis entfernen
<code>s = link(oldpath, newpath)</code>	Link auf bestehende Datei erzeugen
<code>s = unlink(path)</code>	Link löschen
<code>s = chdir(path)</code>	Aktuelles Verzeichnis wechseln
<code>dir = opendir(path)</code>	Verzeichnis zum Lesen öffnen
<code>s = closedir(dir)</code>	Verzeichnis schließen
<code>dirent = readdir(dir)</code>	Lesen eines Verzeichniseintrags
<code>rewinddir(dir)</code>	Verzeichnis noch einmal von Beginn an neu lesen

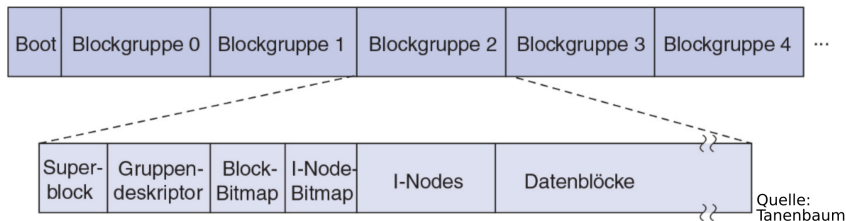
Auswahl an Aufrufen, Rückgabewert im Fehlerfall -1

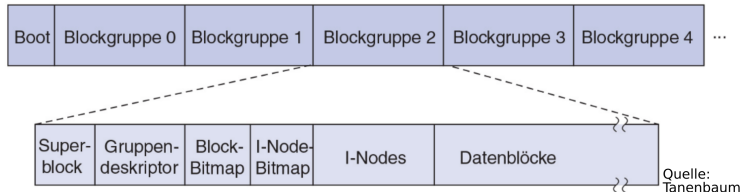
- **Grundlegende Abstraktionsschicht unter Linux (und vielen anderen Unix-Systemen)**
 - Definiert Menge von Operationen die von allen Dateisystemen erwartet werden
 - Aufrufe der letzten Folien gehen auf VFS → werden an konkretes Dateisystem weitergeleitet

Objekt	Beschreibung	Operation
Superblock	Konkretes Dateisystem	read_inode, sync_fs
Dentry	Verzeichniseintrag, einzelne Pfadkomponente	create, link
I-Node	Konkrete Datei	d_compare, d_delete
File	Öffnet eine Datei, die einem Prozess zugeordnet ist	read, write



- **Dateisysteme werden in Partitionen angelegt (oder in äquivalenten Umgebungen, z.B. LVM)**
 - Partitions-Layout definiert wo Partition (und damit Dateisystem) beginnt
 - Länge der Partition definiert wo Dateisystem endet
- **Jedes Dateisystem organisiert den Speicher innerhalb der Partition selbst**
 - Ext2 (und später) teilen Speicher einer Partition in *Blockgruppen* auf



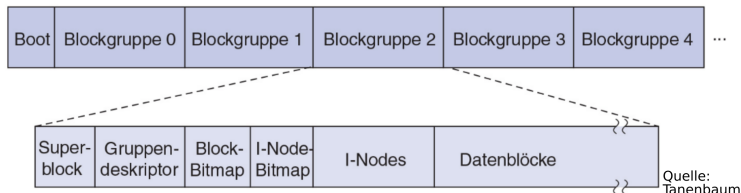


- **Super-Block**

- Informationen über Layout des Blocks
- Anzahl der I-Nodes und Plattenblöcke
- Beginn der Liste der freien Blöcke

- **Gruppen-Deskriptor**

- Startblock und Größe der Block-Bitmap
- Startblock und Größe der I-Node-Bitmap
- Anzahl der freien Blöcke, I-Nodes und Verzeichnisse der Gruppe



- **Liste der I-Nodes und Datenblöcke im Anschluss**
 - Maximale Anzahl von I-Nodes pro Blockgruppe vom System definiert
 - Ein I-Node symbolisiert eine Datei. I-Node-Größe 128 Bytes
 - Datenblöcke speichern Inhalt von Dateien und Verzeichnissen
- **Verteilung von Dateien und Verzeichnissen über Blockgruppen**
 - Verzeichnisse werden über Blockgruppen verteilt
 - Dateien werden versucht in gleiche Blockgruppe zu platzieren wie Vorgängerverzeichnis

Suche nach Verzeichnis /usr/ast/mbox

Quelle: Tanenbaum

Wurzel-
verzeichnis

1	.
1	..
4	bin
7	dev
14	lib
9	etc
6	usr
8	tmp

Suche nach
usr führt zu
I-Node 6

I-Node 6
steht für /usr

Modus Größe Zeiten
132

I-Node 6 besagt,
dass /usr in
Block 132 ist

Block 132
ist das
Verzeichnis /usr

6	.
1	..
19	dick
30	erik
51	jim
26	ast
45	bal

/usr/ast
ist in
I-Node 26

I-Node 26
steht für /usr/ast

Modus Größe Zeiten
406

I-Node 26 besagt,
dass /usr/ast in
Block 406 ist

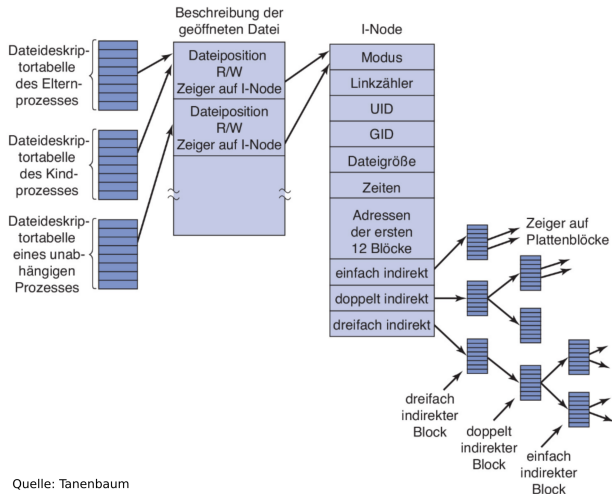
Block 406
ist das
Verzeichnis /usr/ast

26	.
6	..
64	grants
92	books
60	mbox
81	minix
17	src

/usr/ast/mbox
ist in I-Node 60

- **Besuchte Verzeichnisse werden in *Dentry* Cache für schnelleren Zugriff gespeichert**

- **Datei öffnen (open) erzeugt Eintrag in *File Descriptor Table* mit Verweis auf I-Nodes**
 - Enthält Verweis auf aktuelle Position in der Datei und I-Node



Quelle: Tanenbaum

Fragen?