Systeme I: Betriebssysteme

Kapitel 3 Dateisysteme

Wolfram Burgard

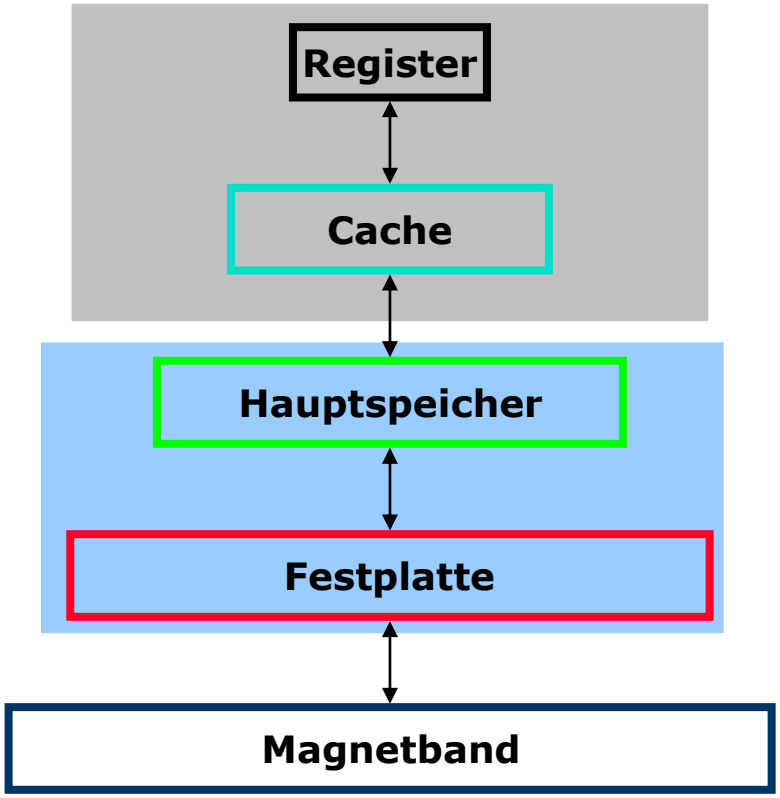
Version 25.10.2017

Weiterer Inhalt der Vorlesung

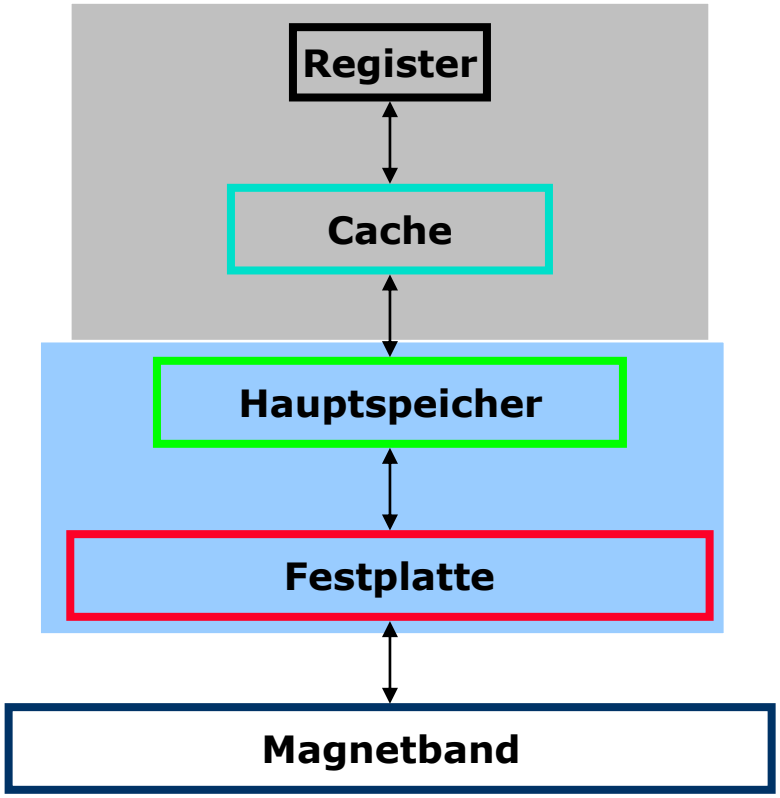
Verschiedene Komponenten und Konzepte von Betriebssystemen

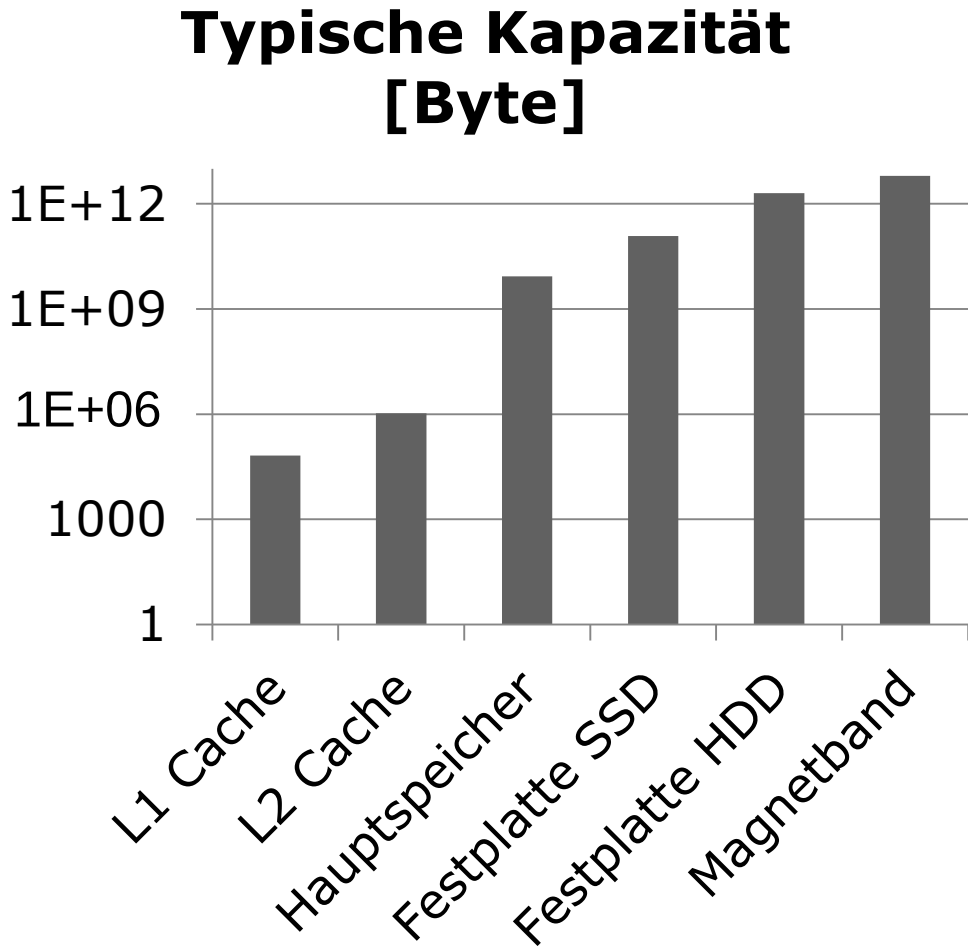
- Dateisysteme
- Prozesse
- Nebenläufigkeit und wechselseitiger Ausschluss
- Deadlocks
- Scheduling
- Speicherverwaltung
- Sicherheit

Speicherhierarchie

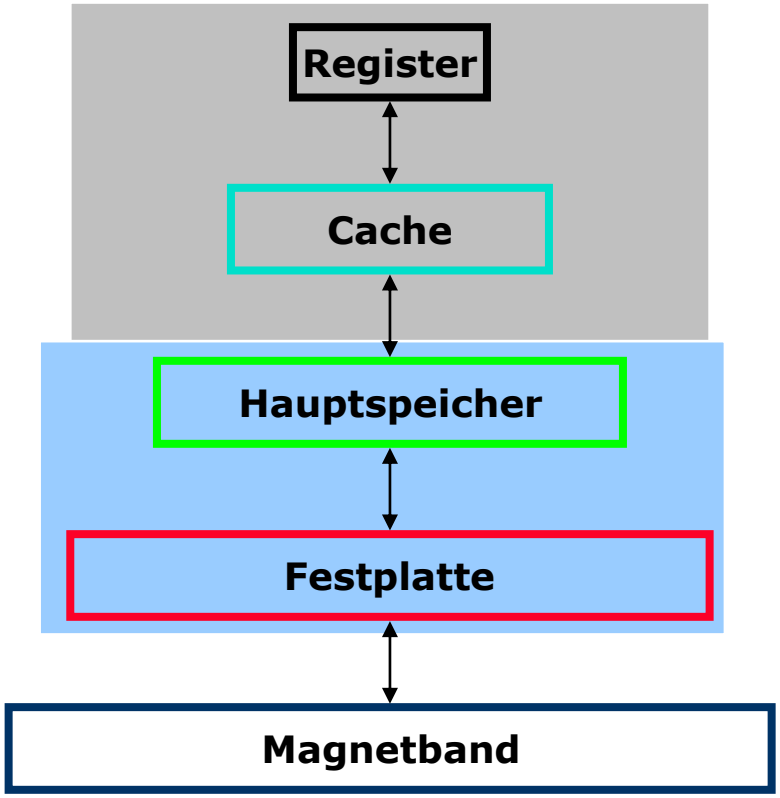


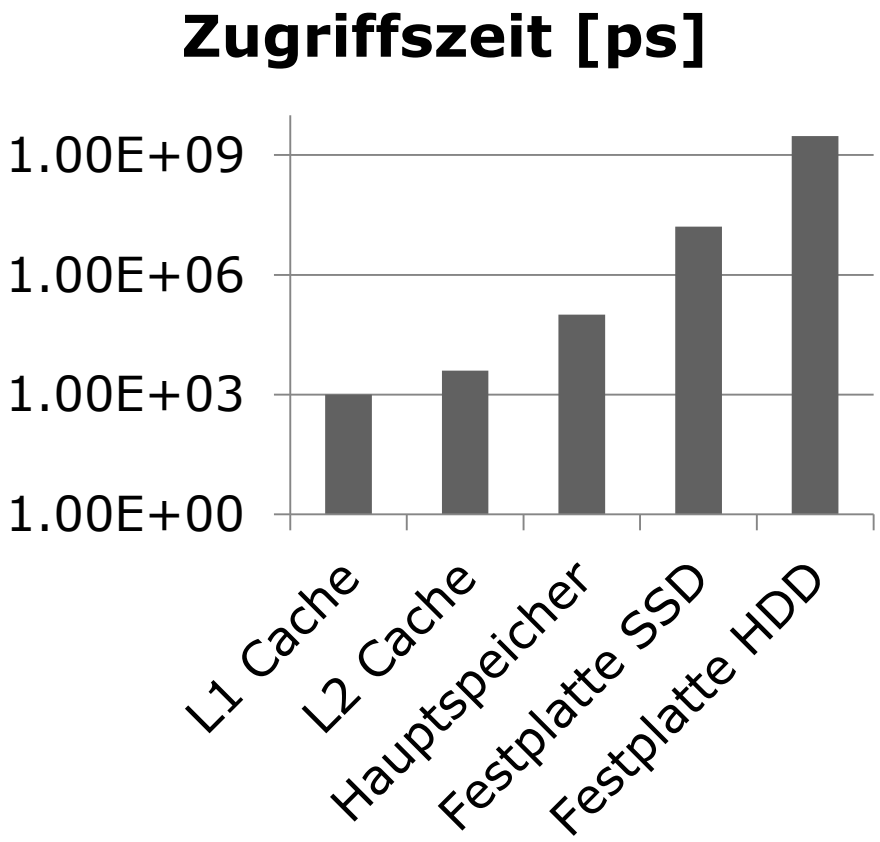
Speicherhierarchie



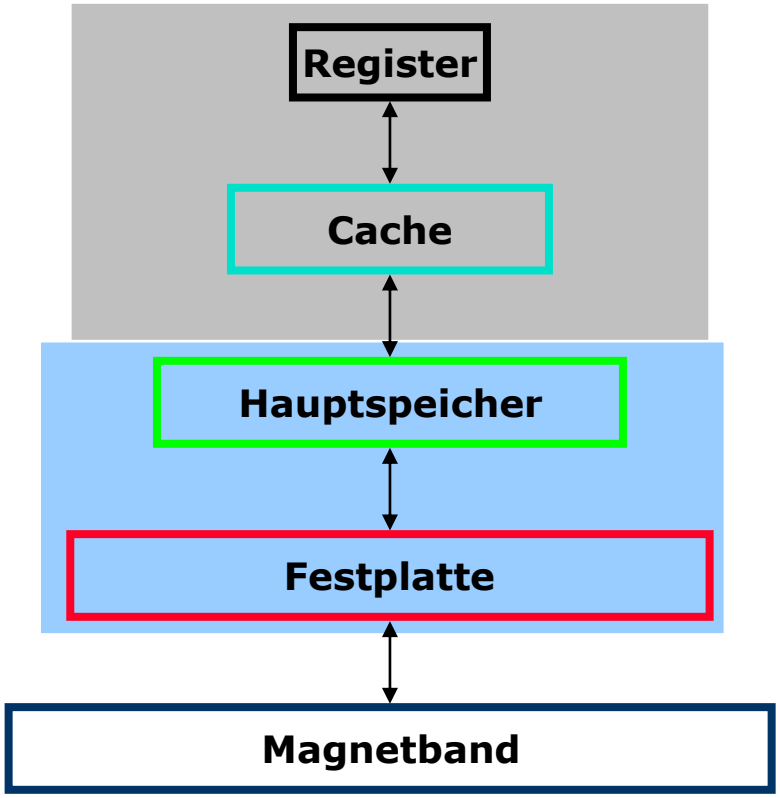


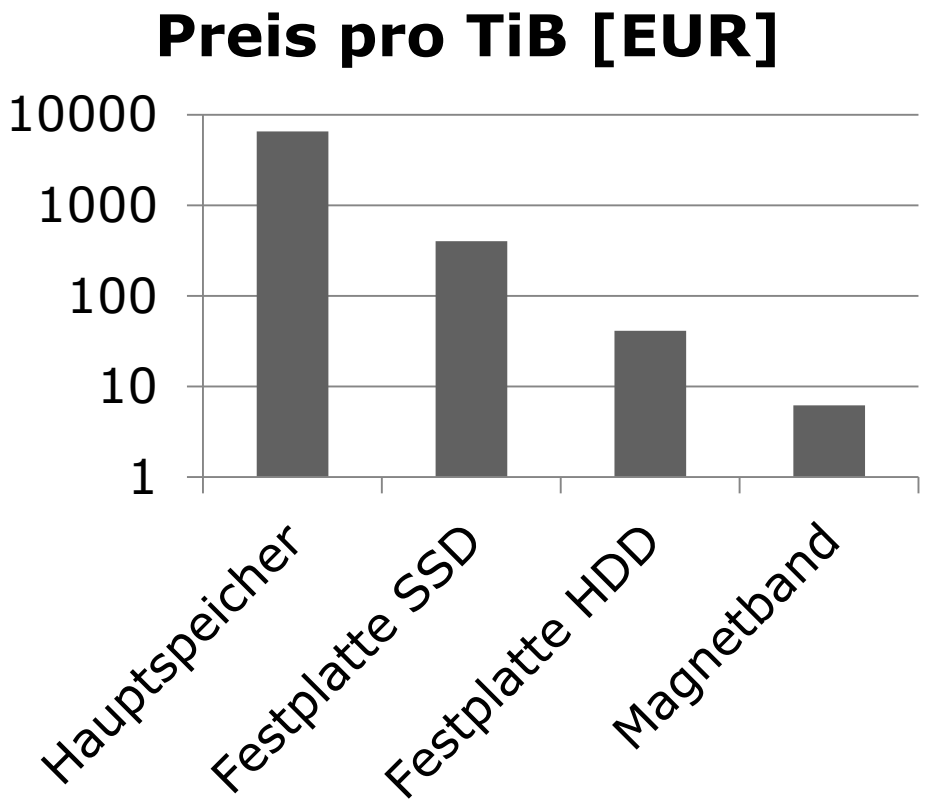
Speicherhierarchie



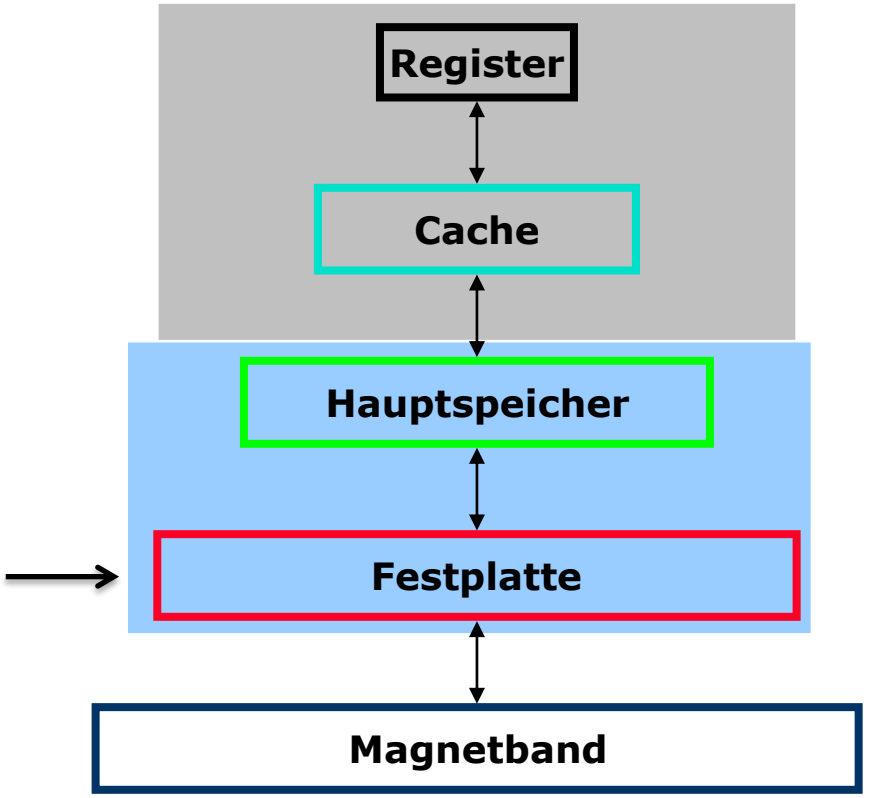


Speicherhierarchie





Speicherhierarchie



Kleine Kapazität, kurze Zugriffszeit,
hohe Kosten pro Bit

Große Kapazität, lange Zugriffszeit,
niedrige Kosten pro Bit

Dateisysteme

- Wichtige Aufgabe eines Betriebssystems: Dauerhaftes Speichern von Informationen
- Speichern von Daten auf Festplatten, USB-Sticks, ...
- Organisationseinheit: Datei
- Dateisystem :=
  - Hier: Betriebssystemteil, der sich mit Dateien befasst
  - Später in der Vorlesung auch: (Typ der) Verwaltungsstruktur einer Partition

Zwei Sichtweisen

- Externe Sicht: Vom Dateisystem angebotene Funktionalität
- Interne Sicht: Implementierung von Dateisystemen

Heutige Vorlesung

- Was sind die Aufgaben eines Dateisystems?
- Dateisystem aus Benutzersicht
  - Wie greift man auf Dateien zu?
  - Wie funktionieren Zugriffsrechte?
- Wie können Dateisysteme implementiert werden?
  - Wie werden Dateien (physisch) realisiert?

Aufgabe von Dateisystemen (1)

- Dauerhafte Speicherung von Daten in Form benannter Objekte
- Bereiche auf dem Speichermedium werden mit einem so genannten Dateisystem versehen
- Dateisystem stellt die Verwaltungsstruktur für Objekte bereit

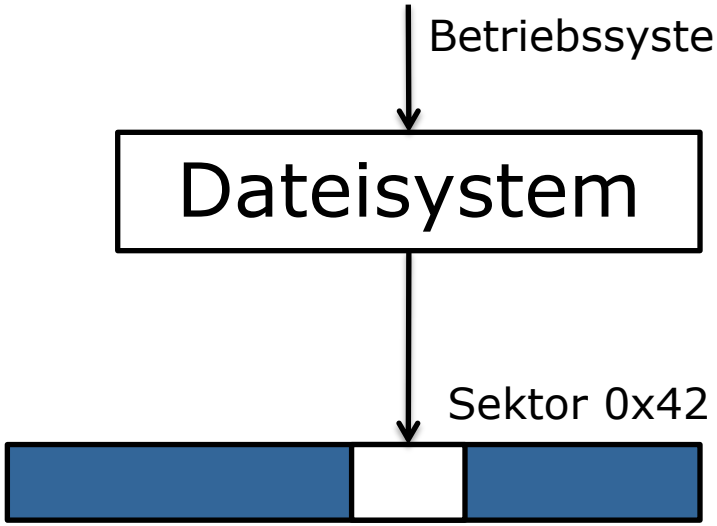
Aufgabe von Dateisystemen (2)

Anfrage der Benutzerin nach Inhalt einer Datei muss vom Betriebssystem beantwortet werden

Öffne Datei `/home/someuser/Documents/systeme1.txt`

Betriebssystem:

Hardware (Festplatte):



Eigenschaften der Verwaltungsstruktur

- Hierarchische Strukturierung: Verzeichnisse mit Objekten
- Beispiele für Objekte:
  - Reguläre Dateien: Text-, Binärdateien
  - Verzeichnisse
  - Gerätedateien: Schnittstelle zwischen Hardware und Anwendungsprogrammen
  - Links: Verweise auf Dateien

Benennung von Dateien (1)

- Benennung durch Dateinamen
- Zugriff über Namen
- Regeln für die Benennung variieren von System zu System

Benennung von Dateien (2)

MS-DOS: Dateinamen bestanden aus

- Eigentlichem Namen (bis zu acht gültige Zeichen, beginnend mit Buchstaben)
- Dateierweiterung nach Punkt (bis zu drei Zeichen, gibt den Typ an)
- Keine Unterscheidung zwischen Groß- und Kleinschreibung
- Beispiele: PROG.C, HALLO.TXT

Benennung von Dateien (3)

Windows: Dateien bestehen aus

- Name bis zu 256 Zeichen (ab 95 bis 7) bzw. 32000 Zeichen (Windows 8)
- Mehrere Punkte möglich, letzter legt Dateityp fest: z.B. mein.prog.c, hilfe.jetzt.pdf
- Keine Unterscheidung zwischen Groß- und Kleinschreibung
- Nicht erlaubt: Einzelne Sonderzeichen, reservierte Wörter: z.B. q"uote"s.txt, frage?.txt, aux.txt

Benennung von Dateien (4)

Unix/Linux: Dateien bestehen aus

- Name bis zu 255 Zeichen, mehrere Punkte erlaubt
- Unterscheidung zwischen Groß- und Kleinschreibung
- Dateierweiterung: nur Konvention zur Übersichtlichkeit
- Beispiele: prog.c, Prog.c, prog.c.Z
- Dateien oder Verzeichnisse, deren Name mit einem Punkt beginnt, sind „versteckte“ Dateien und werden nur angezeigt, wenn der Benutzer dies explizit angibt (ls -a)

Dateitypen (1): Reguläre Dateien

Textdateien:

- Folge von Zeilen unterschiedlicher Länge
- Inhalt interpretierbar gemäß Zeichensatz (z.B. Unicode, ISO, ASCII)
- Mit Texteditor editierbar

Binärdateien:

- Interpretation anwendungsabhängig
- Verwendungszweck typischerweise als Dateiendung angegeben

Dateitypen (2)

- Verzeichnisse: Systemdateien zur Strukturierung des Dateisystems
- Modellierung von E/A-Geräten durch Gerätedateien:
  - Spezielle Zeichendateien zur Modellierung serieller Geräte, z.B. Drucker, Netzwerke
  - Spezielle Blockdateien zur Modellierung von Plattenspeicher, z.B. Festplatten oder Images

Dateiattribute

- Zusatzinformationen über Datei, die das Betriebssystem speichert (Metadaten)
- Beispiele:
  - Entstehungszeitpunkt (Datum, Uhrzeit)
  - Dateigröße
  - Zugriffsrechte: Wer darf in welcher Weise auf die Datei zugreifen?

Zugriffsrechte (1)

- Zugriffsarten:
  - Lesender Zugriff
  - Schreibender Zugriff
  - Ausführung
- Zugreifende:
  - Dateieigentümerin
  - Benutzergruppe der Dateieigentümerin
  - Alle anderen Benutzerinnen

Zugriffsrechte (2)

- Sicht des Benutzers (Bsp. Unix / Linux):

```
$ ls -l
drwxr-xr-x 2 hk1032 uni     52 25. Okt 20:36 folien
drwx------ 3 hk1032 uni     27 25. Sep 20:40 klausur
-rw-r--r-- 1 hk1032 uni 285696 10. Okt 11:42 zeitplan.ods
```

- Bedeutung der Felder:
  - Typ des Eintrags: Datei, Verzeichnis
  - Rechte: Besitzer, Gruppenbesitzer, alle anderen Benutzer
  - Anzahl Hardlinks (Datei), Anzahl Unterverzeichnisse (Verzeichnis)
  - Besitzer
  - Gruppenbesitzer
  - Speicherplatzverbrauch
  - Datum und Zeit der letzten Modifikation
  - Name

Mögliche Dateiattribute

Attribute	Bedeutung
Schutz	Wer kann auf die Datei zugreifen
Passwort	Passwort für den Zugriff auf die Datei
Urheber	ID der Person, die die Datei erzeugt hat
Eigentümerin	Aktuelle Eigentümerin
Read-only-Flag	0: Lesen/Schreiben; 1: nur Lesen
Hidden-Flag	0: normal; 1: in Listen 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 Änderung der Datei
Aktuelle Größe	Anzahl der Bytes in der Datei
Maximale Größe	Anzahl der Bytes für maximale Größe der Datei

Rechteverwaltung Unix (1)

Standardrechte

- Reguläre Dateien
  - r: Datei darf gelesen werden (read)
  - w: Datei darf geändert werden (write)
  - x: Datei darf als Programm ausgeführt werden (execute)
- Verzeichnisse
  - r: Der Inhalt des Verzeichnisses darf gelesen werden
  - w: Der Inhalt des Verzeichnisses darf geändert werden: Dateien anlegen, umbenennen, löschen
  - x: Man darf in das Verzeichnis wechseln und die Objekte darin benutzen

Rechteverwaltung Unix (2)

- Mit dem Befehl `chmod` verändert man die Zugriffsrechte von Dateien
- Zuerst steht der Benutzertyp, dessen Rechte verändert werden sollen: u=user, g=group, o=others oder a=all
- Dann folgt entweder
  - + um Rechte zu setzen oder
  - – um Rechte zu entziehen oder
  - = um nur die explizit angegebenen Rechte zu setzen (und die restlichen zu entziehen)
- Am Schluss folgen die Rechte
- Beispiel: `chmod go-rw dateiname.txt`

Rechteverwaltung Unix (3)

Sonderrechte

SUID (set user ID):

- Erweitertes Zugriffsrecht für Dateien
- Unprivilegierte Benutzer erlangen kontrollierten Zugriff auf privilegierte Dateien
- Setzen des Bits: `chmod u+s datei`
- Ausführung mit den Rechten der Besitzerin der Datei (anstatt mit den Rechten der ausführenden Benutzerin)
- Optische Notation bei ls -l:

  -rwsr-x---

Rechteverwaltung Unix (4)

- root-User: Hat Zugriff auf alles (Superuser)
- Beispiel SUID:

```
-r-sr-xr-x … root root … /usr/bin/passwd
```

  - Beim Aufruf von 'passwd' läuft das Programm mit der uid des Besitzers root und darf in /etc/shadow schreiben
  - SUID-root-Programme sind sicherheitskritisch!

Rechteverwaltung Unix (5)

Sonderrechte

Analog: SGID (set group ID)

- Ausführung mit den Rechten der Gruppe, der die Datei/das Verzeichnis gehört (anstatt mit den Rechten der Gruppe, die ausführt)
- Verzeichnisse: Neu angelegte Dateien gehören der Gruppe, der auch das Verzeichnis gehört (anstatt der Gruppe, die eine Datei erzeugt)
- Setzen des Bits: `chmod g+s verzeichnis`
- Optische Notation bei ls -l:

```
drwxrws---
```

Rechteverwaltung Unix (6)

Sonderrechte

Beispiel:

- Verzeichnis work gehört der Gruppe aisstu
  `drwxrwxr-x meier  aisstu  work`
- Benutzerin mueller gehört zur Gruppe ais (standard) aber auch zu Gruppe aisstu
- Anlegen einer Datei durch mueller in work setzt die Gruppe der Datei auf ais:
  `-rw-rw-r--  mueller  ais  test.txt`
- Gruppe aisstu kann in diese Datei nicht schreiben!

Rechteverwaltung Unix (7)

Sonderrechte

- Dies kann verhindert werden durch SGID:
  `chmod g+s work:`
  `drwxrwsr-x meier aisstu work`
- Anlegen einer Datei durch mueller in work setzt nun die Gruppe der Datei auf aisstu:
  `-rw-rw-r-- mueller aisstu test.txt`
- Gruppe aisstu kann jetzt in die Datei schreiben
- Alternativ könnte Benutzerin mueller die Schreibrechte für alle Benutzerinnen setzen, dies ist u.U. aber nicht gewünscht!
- Beispiel: Homepage auf Webserver

Rechteverwaltung Unix (8)

Sonderrechte

- SVTX (save text bit, sticky bit):
  - Zum Löschen der Datei in einem Verzeichnis muss die Benutzerin auch die Eigentümerin der Datei oder des Verzeichnissen sein
  - Optische Notation: `drwxrwxrwt`
- Beispiel:

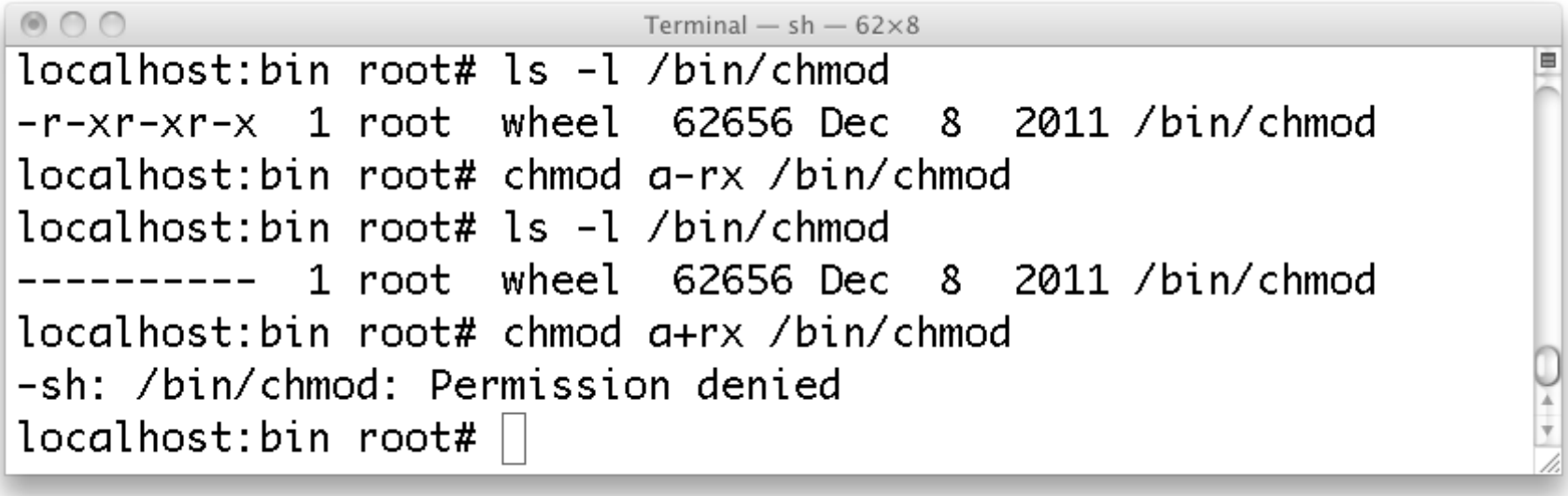
```
drwxrwxrwt … root root … /tmp
```

SVTX verhindert, dass jeder alles löschen darf

Rechteverwaltung Unix (9)

- Ergänzende, komplexere Zugriffsrechte: Access Control Lists (ACLs)
- Einzelnen Nutzerinnen (oder auch Gruppen) können gezielt Rechte an einzelnen Dateien gegeben bzw. entzogen werden
- Damit ist z.B. Folgendes möglich
  - Zwei Benutzerinnen haben das Schreibrecht
  - Sonst niemand
- Optische Notation: `-rw-r--r--+`

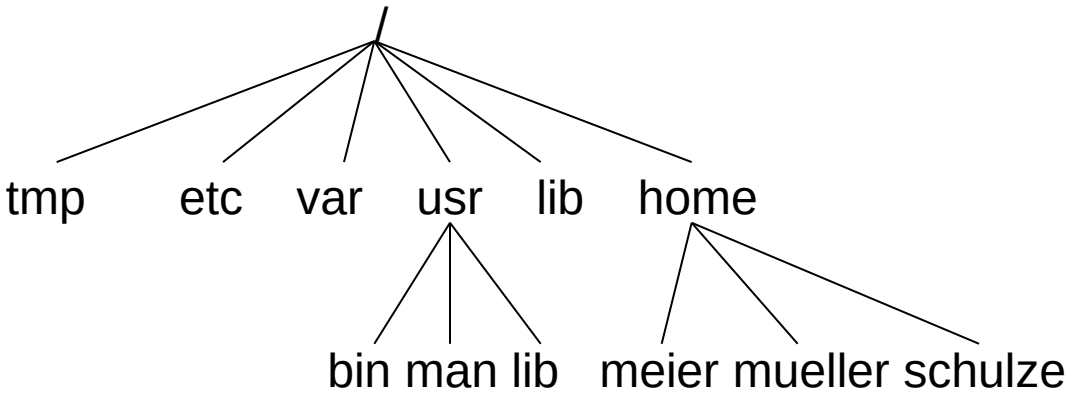
Vorsicht!


Terminal — sh — 62×8
localhost:bin root# ls -l /bin/chmod
-r-xr-xr-x 1 root wheel 62656 Dec 8 2011 /bin/chmod
localhost:bin root# chmod a-rx /bin/chmod
localhost:bin root# ls -l /bin/chmod
---------- 1 root wheel 62656 Dec 8 2011 /bin/chmod
localhost:bin root# chmod a+rx /bin/chmod
-sh: /bin/chmod: Permission denied
localhost:bin root#

Verzeichnisbaum

Baumartige hierarchische Strukturierung

- Wurzelverzeichnis (root directory)
- Restliche Verzeichnisse baumartig angehängt

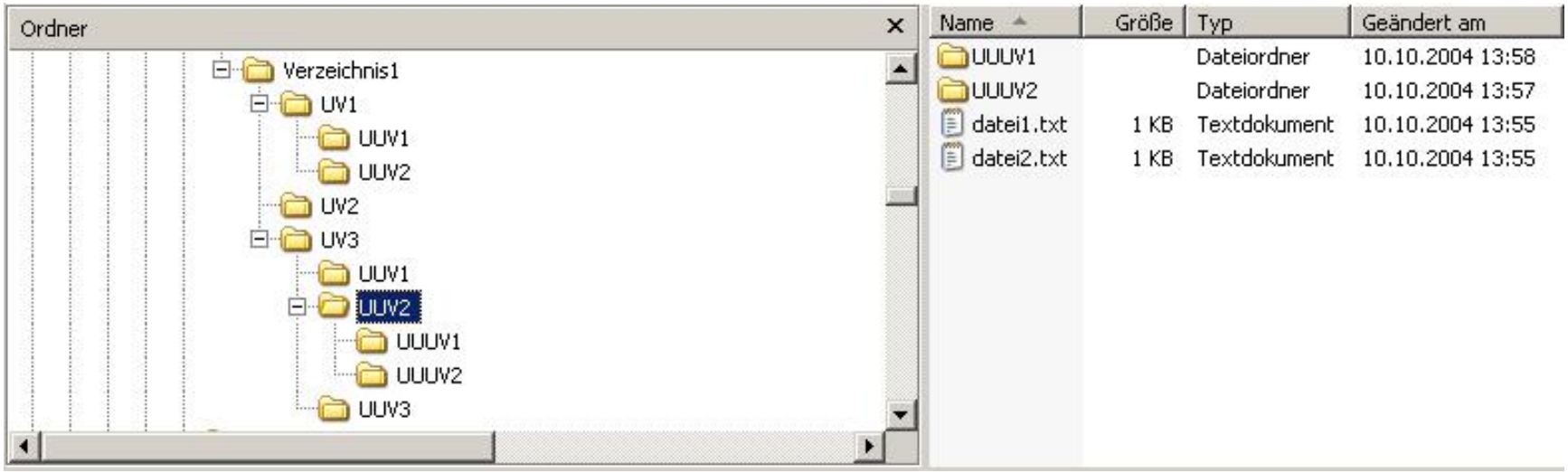


Verzeichnisse

- Absolute Pfade beginnen mit /,
  z.B. /home/meier
- Relative Pfade beziehen sich auf das aktuelle Arbeitsverzeichnis
- Relative Pfade beginnen nicht mit /,
  z.B. meinverzeichnis
- Wechsel des aktuellen Arbeitsverzeichnisses durch cd („change directory“)

Verzeichnisse/Verzeichnisbaum

Graphische Repräsentation (hier: Windows)



Zusammenbau von Verzeichnisbäumen (1)

- Temporäres, manuelles Anhängen (Mounten) von Dateisystemen
- `mount /dev/sda1 /mnt/extern`
- Hängt Dateisystem in /dev/sda1 an das Verzeichnis /mnt/extern im bestehenden Verzeichnisbaum an
- /mnt/extern heißt Mountpoint

Zusammenbau von Verzeichnisbäumen (2)

- Evtl. vorhandene Einträge in /mnt/extern werden temporär verdeckt und sind nach Abhängen wieder zugreifbar
- Abhängen mit `umount /mnt/extern`
- Geht nur, wenn keine Datei im angehängten Dateisystem geöffnet ist

Zusammenbau von Verzeichnisbäumen (3)

Graphische Oberfläche:

- Das Mounten von Wechseldatenträgern wie USB-Sticks, CDs, externen Festplatten, etc. geschieht meist automatisch
- Typischerweise im Verzeichnis /media oder /Volumes

Symbolische Links (1)

- Zweck von Links: Ansprechen desselben Objekts mit mehreren Namen
- Beispiel: Benutzerin meier befindet sich in Arbeitsverzeichnis /home/meier
- meier will bequem auf ausführbare Datei /home/mueller/mytools/Text/script zugreifen
- Anlegen eines Links im Verzeichnis /home/meier:

```
ln –s /home/mueller/mytools/Text/script shortcut
```

Symbolische Links (2)

- Ziel des Links mit ls anzeigbar

```
$ ls -l shortcut
lrwxrwxrwx 1 meier users … shortcut->
/home/mueller/mytools/Text/script
```

- Aufruf durch shortcut in Verzeichnis /home/meier nun möglich

Symbolische Links (3)

- Anlegen: `ln -s ziel linkname`
- Enthalten Pfade zu Referenzobjekten (Ziel)
- Löschen/Verschieben/Umbenennen von Referenzobjekt: Link zeigt ins Leere
- Löschen von Link: Referenzobjekt ist unverändert
- Rechte am Link: Entsprechend Referenzobjekt
- Zulässiges Referenzobjekt: beliebiges Objekt im Dateibaum
- Technische Realisierung: später

Harte Links

- Anlegen: `ln ziel linkname`
- Erstellt Verzeichniseintrag in Dateisystem mit weiterem Namen für Dateiobjekt
- Zulässiges Referenzobjekt: Dateiobjekt in demselben Dateisystem (Festplattenpartition)
- Nach Umbenennen/Verschieben funktioniert Link noch
- Technische Realisierung: später

Harte Links: 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 den Wert 0 hat

Zugriff auf Dateiinhalte

- Sequentieller Zugriff (Folgezugriff)

- Wahlfreier Zugriff (Direktzugriff, Random Access)

Sequentieller Dateizugriff

- Alle Bytes können nur nacheinander vom Datenspeicher gelesen werden
- Kein Überspringen möglich
- Um auf einen bestimmten Datensatz zugreifen zu können, müssen alle Datensätze zwischen Start- und Zielposition besucht werden
- Die Zugriffszeit ist von der Entfernung der Datensätze vom ersten Datensatz abhängig

Wahlfreier Dateizugriff

- Zugriff auf ein beliebiges Element in konstanter Zeit
- Bytes/Datensätze können in beliebiger Reihenfolge ausgelesen werden
- Befehl seek: setzt den Dateizeiger auf eine beliebige Stelle (Byteposition innerhalb der Datei)
- Danach kann die Datei sequentiell von dieser Position gelesen werden
- Für viele Anwendungen relevant (z.B. Datenbanksysteme)

Mögliche Operationen auf Dateien (Systemaufrufe)

- Create: Erzeugen
- Delete: Löschen
- Open: Öffnen
- Close: Schließen
- Read: Lesen von aktueller Position
- Write: Schreiben an aktuelle Position

Mögliche Operationen auf Dateien (Systemaufrufe)

- Append: Anhängen an Dateiende
- Seek: Setzt Dateizeiger an beliebige Stelle
- Get Attributes: Lesen der Dateiattribute
- Set Attributes: Verändern der Dateiattribute
- Rename: Umbenennen

Mögliche Operationen auf Verzeichnissen (Systemaufrufe)

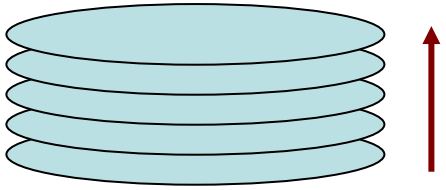
- Create: Erzeugen
- Delete: Löschen
- Opendir: Öffnen
- Closedir: Schließen
- Readdir: Nächsten Verzeichniseintrag lesen
- Get attributes: Lesen der Verzeichnisattribute
- Set attributes: Verändern der Verzeichnisattribute
- Rename: Umbenennen des Verzeichnis

Heutige Vorlesung

- Was sind die Aufgaben eines Dateisystems?
- Dateisystem aus Benutzersicht
  - Wie greift man auf Dateien zu?
  - Wie funktionieren Zugriffsrechte?
- **Wie können Dateisysteme implementiert werden?**
  - Wie werden Dateien (physisch) realisiert?

Implementierung von Dateisystem (Festplatte)

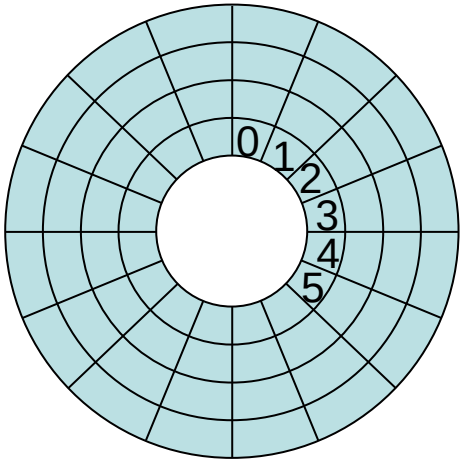
- Eine Festplatte besteht aus mehreren Scheiben und einem Lesekopf



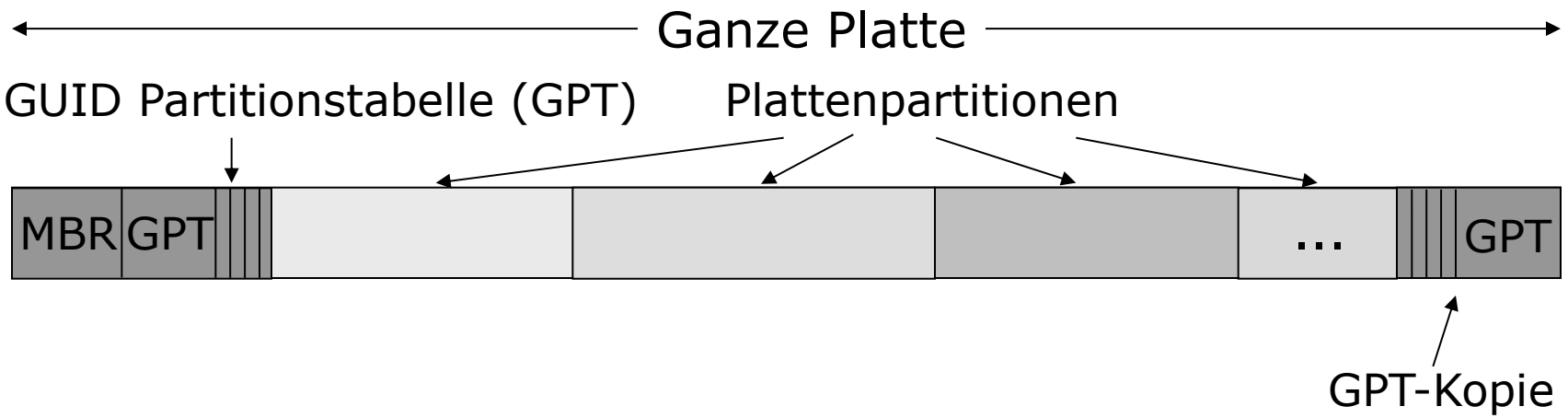
- Festplatten können in eine oder mehrere Partitionen unterteilt werden
- Einzelne Partitionen können unabhängige Dateisysteme besitzen (z.B. Windows und Linux Dateisystem)

Implementierung von Dateisystem (Festplatte)

- Scheiben sind eingeteilt in Blöcke (Sektoren), alle Blöcke der Festplatte sind durchnummeriert

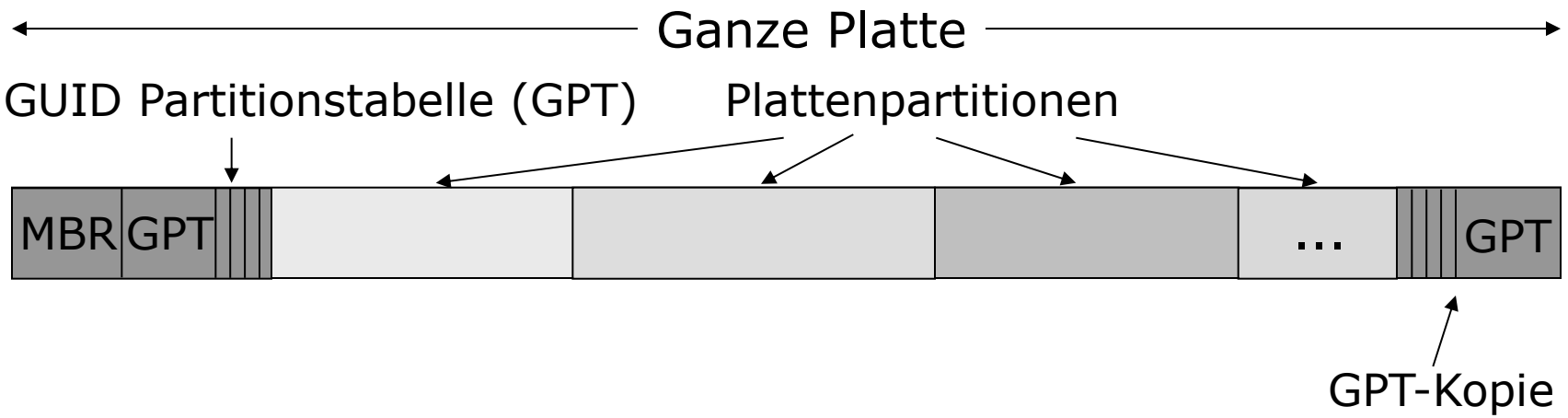


Layout einer Festplatte (1)



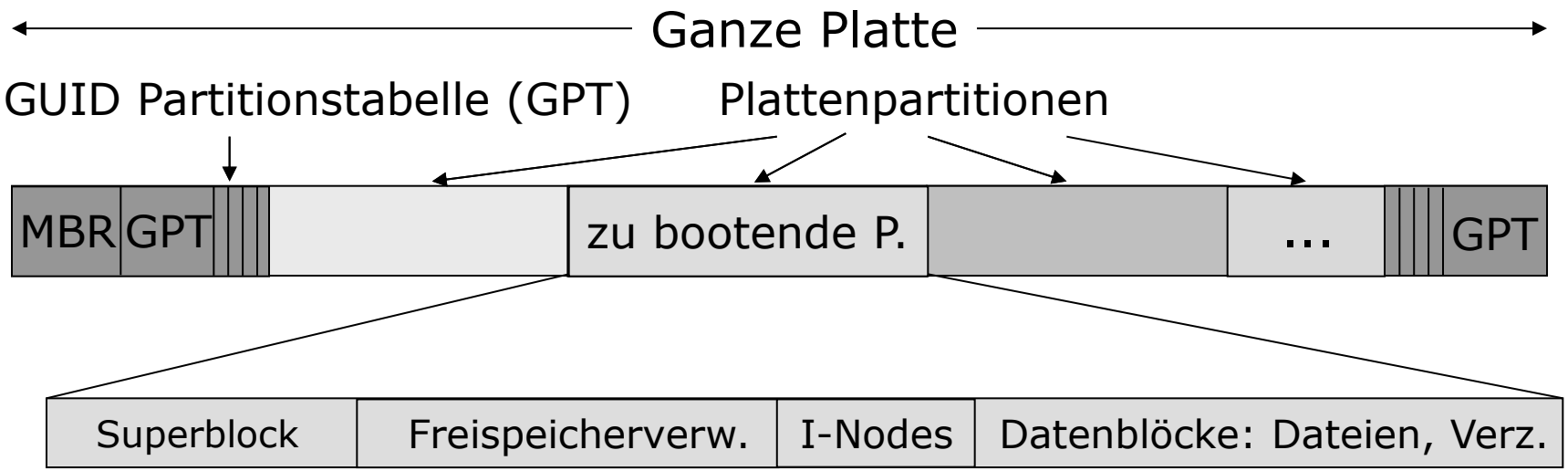
- GPT-formatierte Festplatte
- GPT = GUID (Globally Unique Identifier) Partitionstabelle
- GPT enthält Anfangs- und Endadresse jeder Partition
- Erster Sektor der Platte enthält MBR (Master Boot Record, nur für alte Betriebssysteme)

Layout einer Festplatte (2)



- UEFI (Unified Extensible Firmware Interface): Schnittstelle zwischen Hardware und Betriebssystem beim Bootvorgang (Nachfolger von BIOS)
- Wahl der zu bootenden Partition durch Boot-Manager (Teil von UEFI)
- Betriebssystem von der gewählten Partition wird eingelesen und gestartet

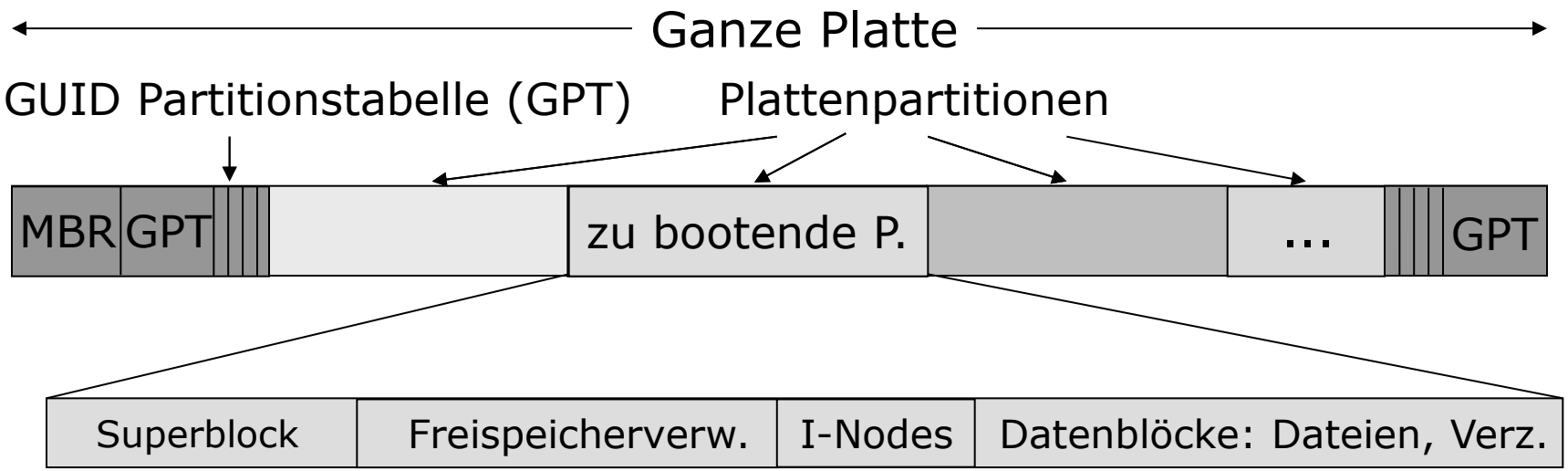
Layout einer Festplatte (3)



Bsp. Linux:

- Superblock enthält Schlüsselparameter des Dateisystems (z.B. Name des Dateisystemtyps, Anzahl Blöcke, ...)
- Freispeicherverwaltung: Informationen über freie Blöcke im Dateisystem

Layout einer Festplatte (4)



- I-Nodes (Index Node):
  - Enthalten Metadaten der Dateien (Eigentümer, Zugriffsrechte, Dateityp, Größe, Linkzähler, etc.)
  - Zu jeder Datei gehört ein einziger I-Node
- Datenblöcke: Eigentliche Inhalte der Dateien

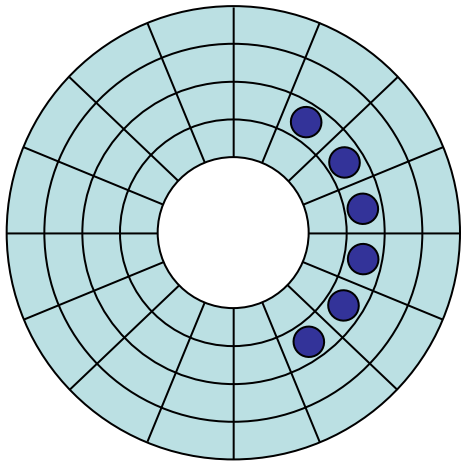
Realisierung von Dateien

Drei verschiedene Alternativen zur Realisierung von Dateien:

- Zusammenhängende Belegung
- Belegung durch verkettete Listen
- I-Nodes (bzw. File Records)

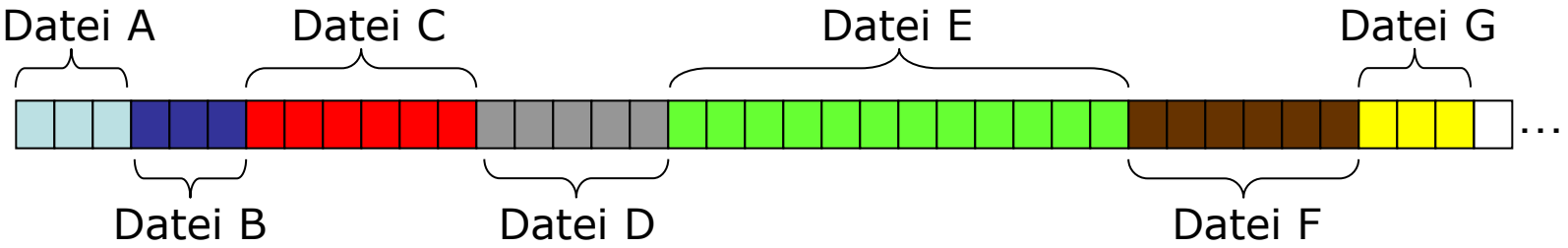
Realisierung von Dateien: Zusammenhängende Belegung (1)

- Abspeicherung von Dateien durch zusammenhängende Menge von Plattenblöcken



Realisierung von Dateien: Zusammenhängende Belegung (2)

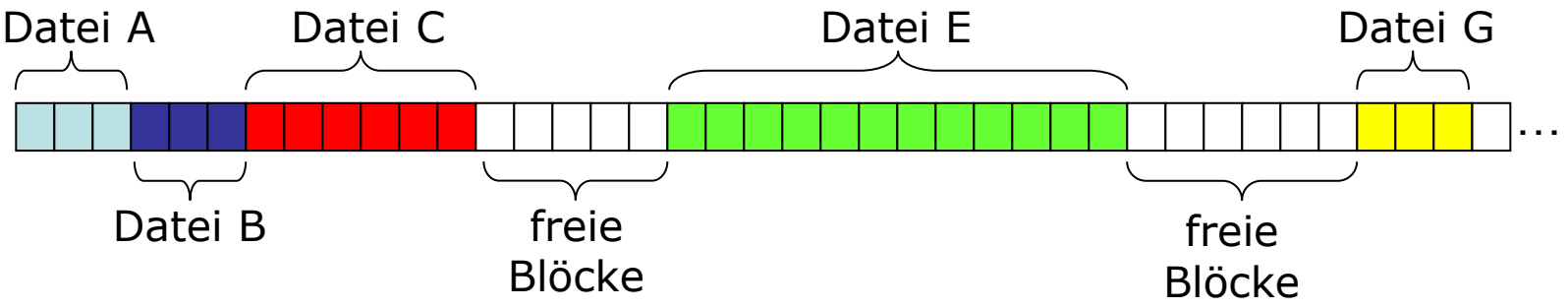
- Abspeicherung von Dateien durch zusammenhängende Menge von Plattenblöcken



- Vorteil: Lesegeschwindigkeit (wenige Leseoperationen für gesamte Datei)

Realisierung von Dateien: Zusammenhängende Belegung (3)

- Situation nach Löschen von D und F:



- Nachteil: Externe Fragmentierung der Platte
- Verschiebung der Blöcke (Defragmentierung) ist ineffizient

Realisierung von Dateien: Zusammenhängende Belegung (4)

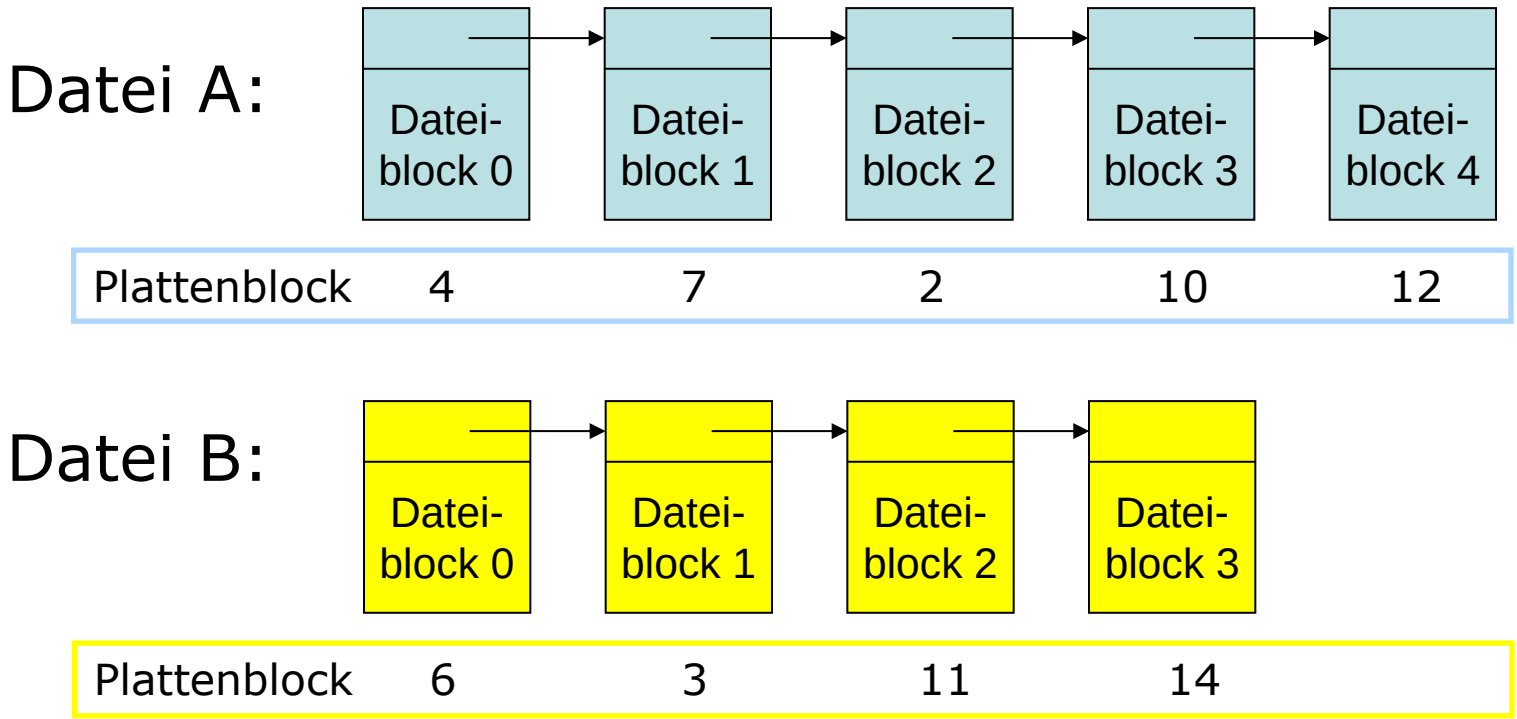
- Verwaltung der entstehenden Lücken in Listen theoretisch möglich
- Allerdings: Endgröße von Dateien muss bekannt sein, um passende Lücke zu finden (Dateien können nicht wachsen!)
- Oft ist diese nicht bekannt und würde daher deutlich überschätzt
- Nachteil: Evtl. viel Speicherplatz ungenutzt

Realisierung von Dateien: Zusammenhängende Belegung (5)

- Zusammenhängende Belegung im allgemeinen keine gute Idee!
- Benutzt bei Dateisystemen für einmal beschreibbare Medien, z.B. CD-ROMs: Dateigröße im Voraus bekannt.

Realisierung von Dateien: Belegung durch verkettete Listen (1)

- Dateien gespeichert als verkettete Listen von Plattenblöcken: Zeiger auf nächsten Block
- Variante 1:



Realisierung von Dateien: Belegung durch verkettete Listen (2)

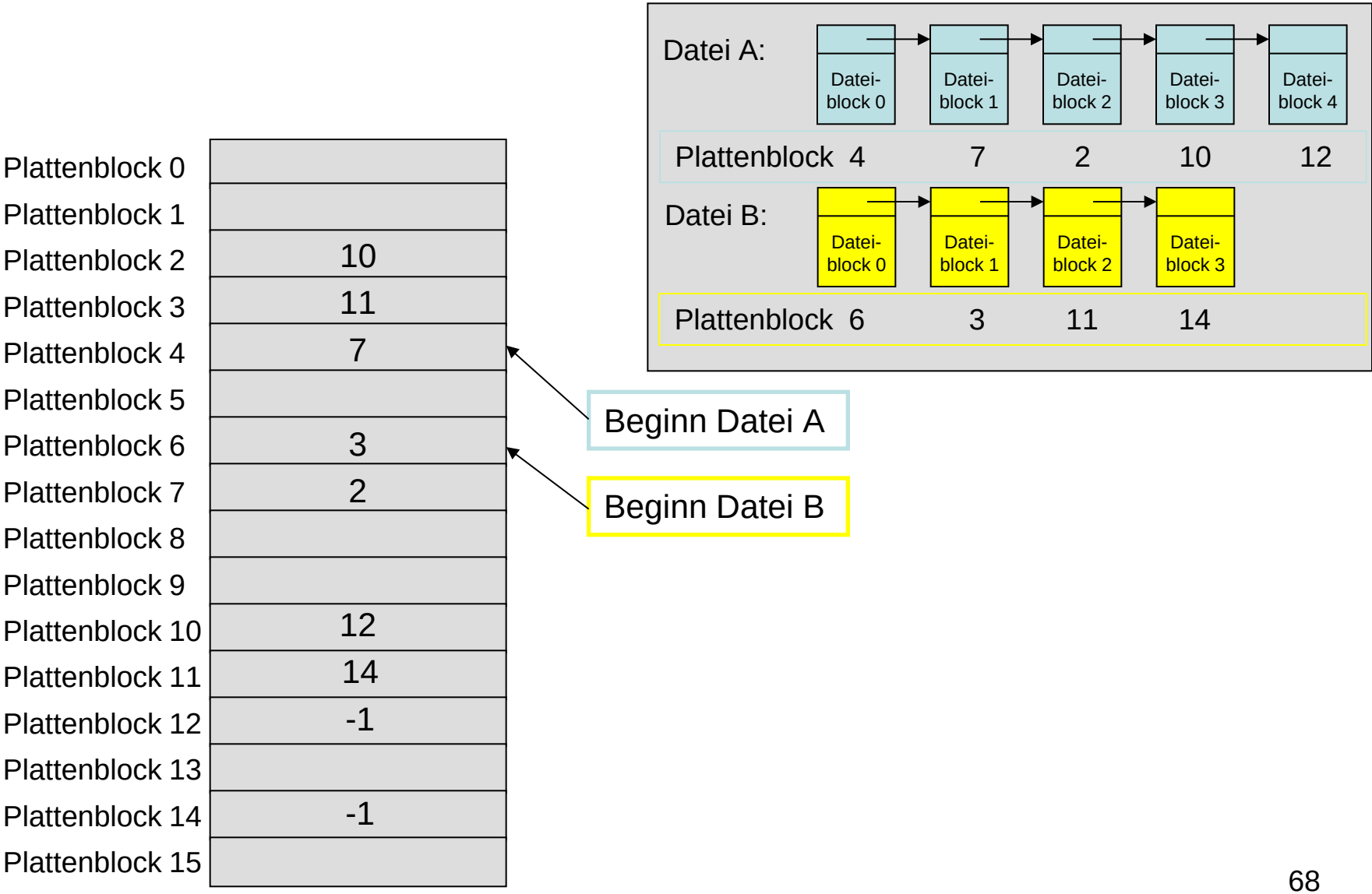
- Vorteile:
  - Fragmentierung der Festplatte führt nicht zu Verlust von Speicherplatz
  - Dateien beliebiger Größe können angelegt werden (solange genug Plattenplatz vorhanden)
- Nachteile:
  - Langsamer wahlfreier Zugriff
  - Zugriff auf Dateiblock *n*: *n-1* Lesezugriffe auf die Platte, um Block *n* zu lokalisieren

Realisierung von Dateien: Belegung durch verkettete Listen (3)

Variante 2:

- Information über Verkettung der Blöcke im Hauptspeicher (Arbeitsspeicher, RAM)
- Ersetze bei wahlfreiem Zugriff Plattenzugriffe durch schnellere Hauptspeicherzugriffe
- Datei-Allokationstabelle (FAT=File Allocation Table) im Hauptspeicher
- Von Microsoft entwickeltes Dateisystem (FAT-12, FAT-16, FAT-32)

FAT Beispiel



FAT Dateisystem

- Vorteil:
  - FAT ist im Hauptspeicher
  - Bei wahlfreiem Zugriff auf Block *n* muss nur eine Kette von Verweisen im Hauptspeicher verfolgt werden (nicht auf der Platte)
- Nachteil:
  - Größe der FAT im Speicher: Jeder Block hat einen Zeiger in der FAT
  - Anzahl der Einträge = Gesamtzahl der Plattenblöcke (auch wenn die Festplatte fast komplett unbelegt ist)

FAT-16: Größe Tabelle / Speicher

- Wie groß ist die FAT im Speicher?
  - FAT-16: Zeigergröße ist 16 Bit = 2 Byte
  - 2^{16} verschiedene Zeiger für 2^{16} Plattenblöcke
  - Tabelle hat 2^{16} Einträge, jeweils 2 Byte groß:
    $2^{16} * 2$ Byte = 2^{17} Byte =
    2^7 Kibibyte = 128 Kibibyte
    (2^{10} Byte = 1 Kibibyte = 1 KiB = 1024 Byte)
- Wieviel Speicher kann verwaltet werden?
  - Maximale Blockgröße bei FAT-16: 32 KiB
  - $2^{16} * 32$ KiB = $2^{16} * 2^5$ KiB = 2^{21} KiB
    = 2^{31} Byte = 2 Gibibyte (= max. Partitionsgröße)

FAT-32 für größere Platten

- Mehr Zeiger für größere Platten
- Für Zeiger: 28 Bit; 4 Bit für andere Zwecke, z.B. Markierung freier Blöcken
- 28 Bit zur Adressierung: 2^{28} verschiedene Zeiger, je 32 Bit = 4 Byte groß
- Größe der FAT: $2^{28} * 4$ Byte = 2^{30} = 1 GiB
- Gegenmaßnahmen für zu große FAT:
  - Nur wirklich benötigten Teil verwalten
  - Betrachte Fenster über der FAT, welches im Speicher bleibt, bei Bedarf auswechseln

FAT-32 System

- Vorteile FAT-32:
  - Maximale Partitionsgröße: 2 TiB
  - Die meisten Betriebssysteme können darauf zugreifen
  - Viele externe Geräte verwenden es heute (Digitalkamera, MP3-Player, ...)
- Nachteile FAT-32:
  - Größe der FAT
  - Maximale Dateigröße: 4 GiB
    (Grund: 4 Byte großes Feld für die Dateigröße in der Directory-Tabelle)

Abschließende Bemerkungen

- Generell: Kleinere Blöcke führen zu weniger verschwendetem Platz pro Datei (interne Fragmentierung)
- Je kleiner die Blockgröße, desto mehr Zeiger, desto größer die FAT im Hauptspeicher
- Maximale Größe des ganzen Dateisystems wird durch Blockgröße begrenzt
- FAT-16 muss z.B. für eine 2 GiB-Partition eine Blockgröße von 32 KiB verwenden
- Andernfalls kann mit den 2^{16} verschiedenen Zeigern nicht die ganze Partition adressiert werden

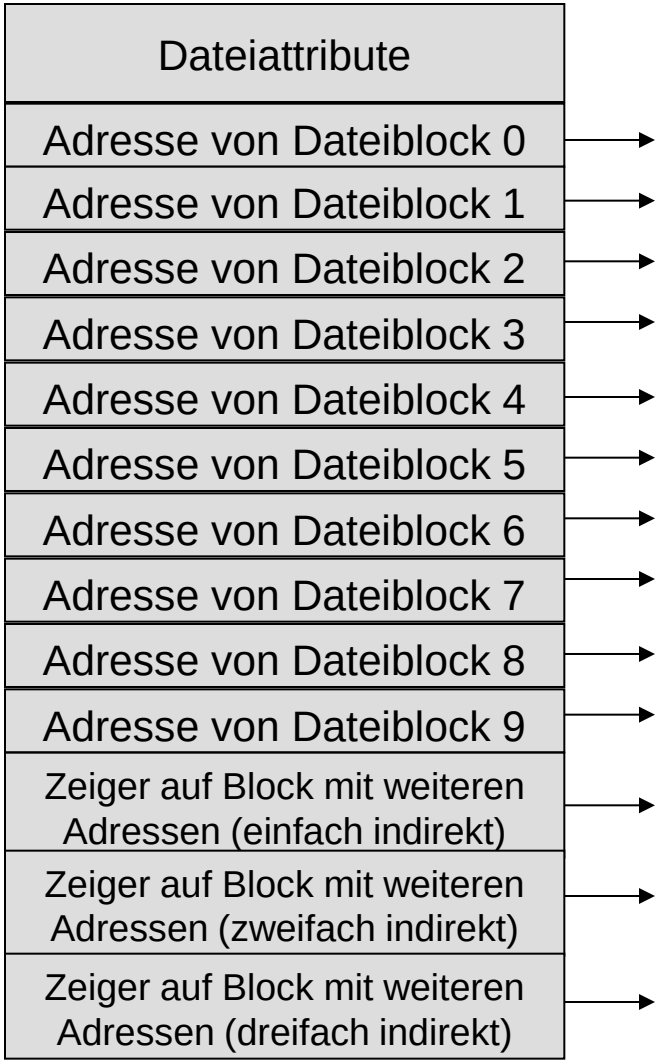
Beobachtung

- Eigentlich braucht man nicht die Informationen der Verkettung für alle Plattenblöcke
- Sondern nur für aktuelle geöffnete Dateien
- Konzept der I-Nodes (Linux, Unix) bzw. File Records (Windows)

Realisierung von Dateien: I-Nodes (1)

- Zu jeder Datei gehört eine Datenstruktur, der sogenannte I-Node (Index-Knoten)
- I-Node enthält: Metadaten und Adressen von Plattenblöcken
- I-Node ermöglicht Zugriff auf alle Blöcke der Datei
- I-Node einer Datei muss nur dann im Hauptspeicher sein, wenn die Datei offen ist
- Wenn k Dateien offen sind und ein I-Node n Bytes benötigt:
  $k*n$ Byte an Speicher werden benötigt

Realisierung von Dateien: I-Nodes (2)



Realisierung von Dateien: I-Nodes (3)

- Viel weniger Bedarf an Hauptspeicher als bei FAT System
- FAT wächst proportional mit Plattengröße
- I-Node-Modell: Größe des benötigten Speichers ist proportional zur maximalen Anzahl gleichzeitig geöffneter Dateien
- Unabhängig davon, wie groß die Platte ist

Realisierung von Dateien: I-Nodes (4)

I-Node einer Datei enthält

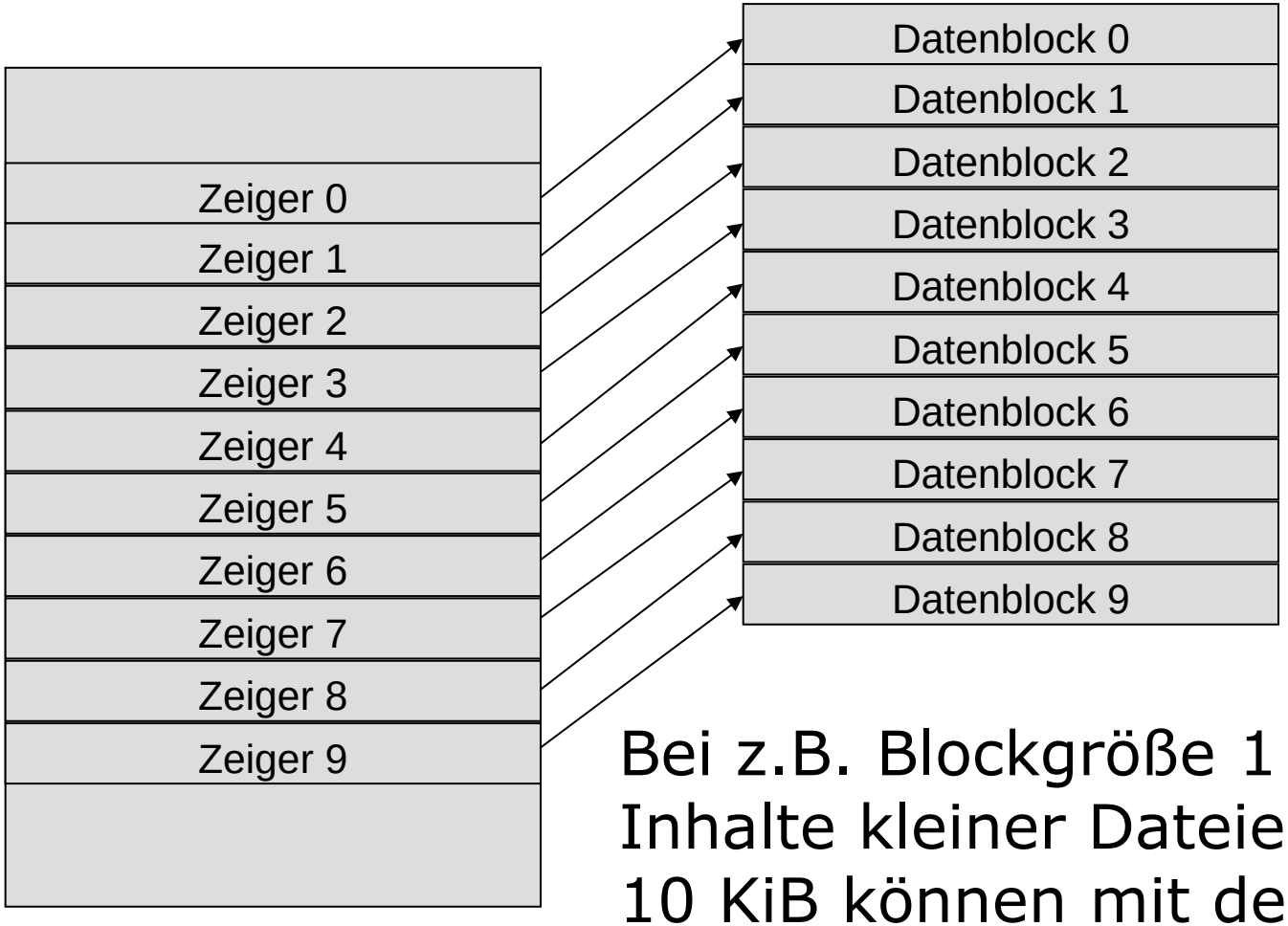
- Alle Attribute der Datei
- *m* Adressen von Plattenblöcken (UNIX System V: *m*=10, ext2/3: m=12)
- Verweise auf Plattenblöcke, die weitere Verweise auf Blöcke enthalten
- Auf die ersten *m* Plattenblöcke kann schnell zugegriffen werden
- Für die folgenden Plattenblöcke sind zusätzliche Zugriffe nötig

Auszug aus Metadaten in I-Nodes (ext2/3/4)

Feld	Beschreibung
`mode`	Dateityp, Schutzbits, setuid, setgid Bits
`links_count`	Anzahl der Hard Links zu diesem I-Node
`uid`	UID des Besitzers
`gid`	GID des Besitzers
`size`	Dateigröße in Bytes
`block`	ext2/ext3: Zeiger zu 12 direkten Blöcken und je einem einfach, doppelt und dreifach indirektem Block
`generation`	Version des I-Nodes, für Verwendung im Netzwerk
`atime`	Zeitpunkt des letzten Zugriffs
`mtime`	Zeitpunkt der letzten Änderung des Dateiinhaltes
`ctime`	Zeitpunkt der letzten Änderung des I-Nodes

Quellcode mit allen Feldern: https://git.kernel.org/cgit/linux/kernel/git/stable/linux-stable.git/tree/fs/ext4/ext4.h?id=refs/tags/v4.8.4#n699

I-Nodes: Direkte Zeiger

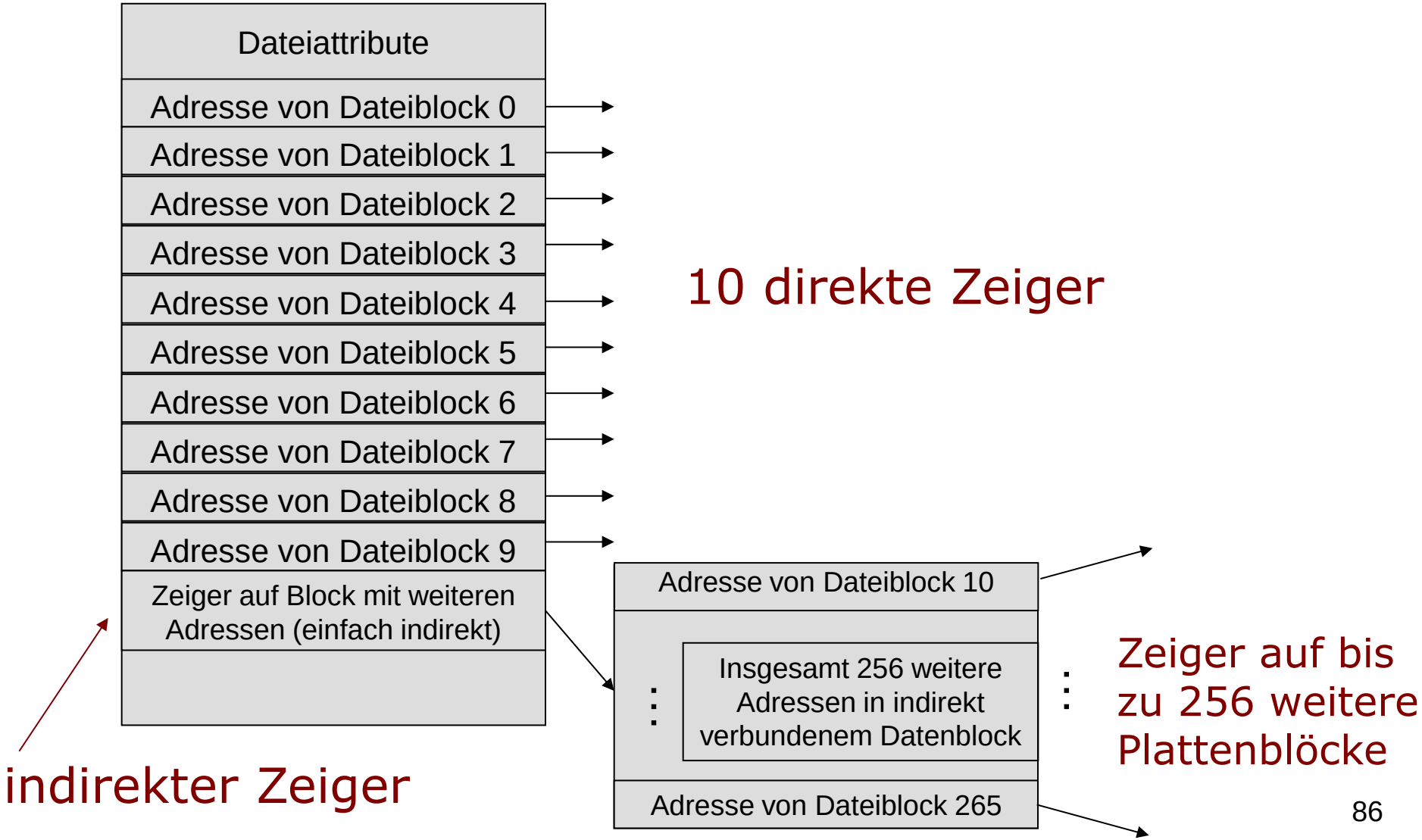


Bei z.B. Blockgröße 1 KiB:
Inhalte kleiner Dateien bis
10 KiB können mit den direkten
Zeigern angesprochen werden

I-Nodes: Indirekte Zeiger (1)

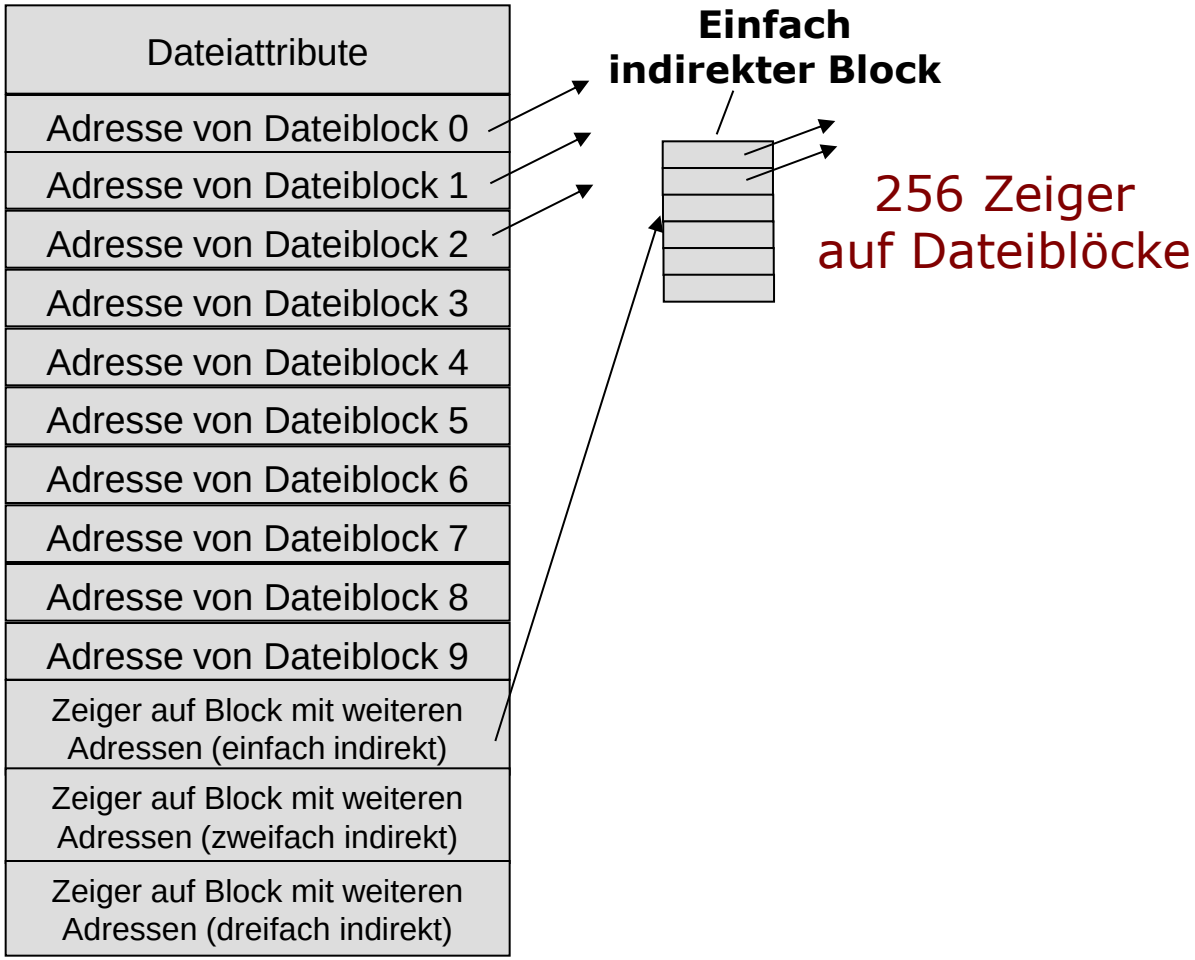
- Für größere Dateien werden Datenblöcke zur Speicherung von weiteren Plattenadressen genutzt
- Beispiel: Blockgröße 1 KiB
- Größe eines Zeigers auf Plattenblock: 4 Byte (also 2^{32} Adressierungsmöglichkeiten)
- 256 Zeiger (je 4 Byte) passen in einen Plattenblock
- Nach 10 (bzw. 12) direkten Plattenadressen gibt es im I-Node einen Zeiger auf einen Plattenblock mit 256 weiteren Plattenadressen von Dateiblöcken
- Nun Dateien möglich bis zu einer Größe von (10 + 256) x 1 KiB = 266 KiB

I-Nodes: Indirekte Zeiger (2)



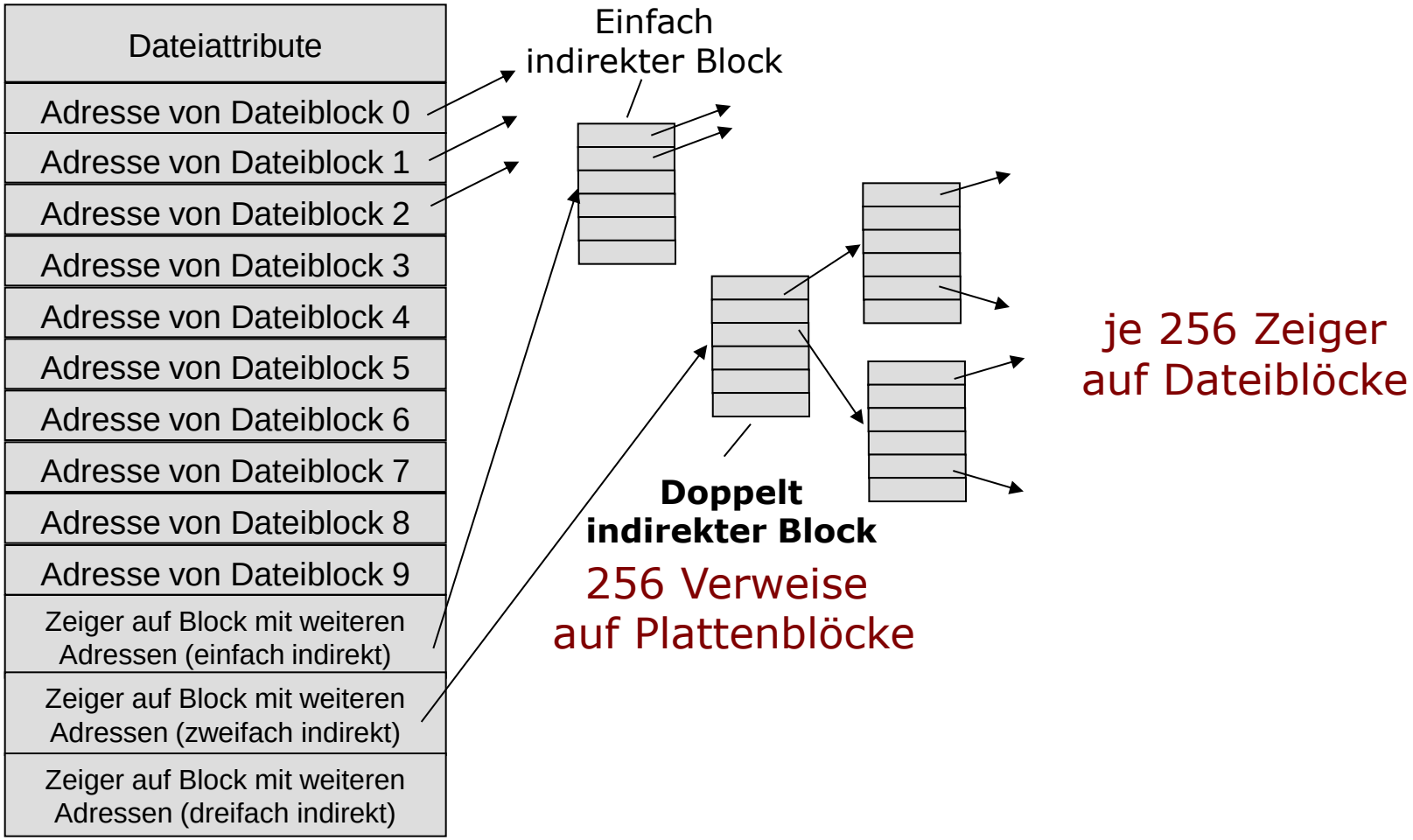
I-Nodes: Indirekte Zeiger (3)

Noch größere Dateien:



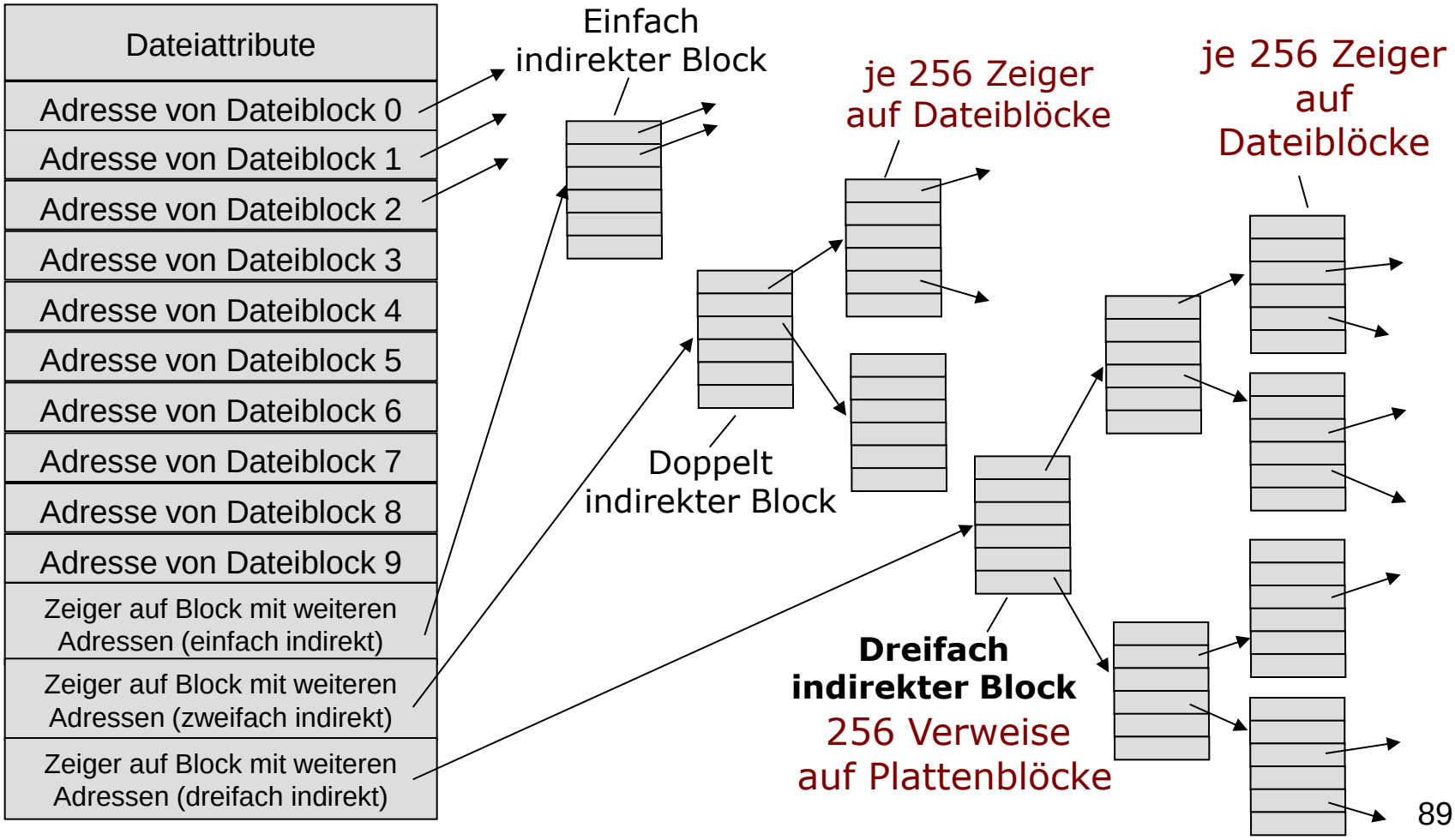
I-Nodes: Indirekte Zeiger (4)

Noch größere Dateien:



I-Nodes: Indirekte Zeiger (5)

Noch größere Dateien:



Beispiel: Maximale Dateigröße

Blockgröße: 1KiB, Zeigergröße: 4 Bytes

Beispiel: Maximale Dateigröße

- Blockgröße 1 KiB, Zeigergröße 4 Bytes
- 10 direkte Zeiger des I-Nodes: 10 KiB Daten lassen sich speichern
- Einfach indirekter Zeiger verweist auf einen Plattenblock, der maximal 1 KiB/4 Byte Zeiger verwalten kann, also 256 Zeiger
- Indirekt: 1 KiB * 256 = 256 KiB Daten
- Zweifach indirekter Zeiger:
  1 KiB * 256 * 256 = 64 MiB Daten
- Dreifach indirekter Zeiger:
  1 KiB * 256 * 256 * 256 = 16 GiB Daten

I-Node-Tabelle (1)

- I-Node-Tabelle auf Festplatte: Enthält alle I-Nodes mit Speicheradresse
- Die Größe der I-Node-Tabelle wird beim Anlegen des Dateisystems festgelegt
- Es muss eine ausreichende Anzahl von I-Nodes eingeplant werden: Jeder Datei ist ein eindeutiger I-Node zugeordnet
- Größe I-Node heute: 256 Bytes

I-Node-Tabelle (2)

- Wenn verfügbarer Plattenplatz oder die I–Nodes belegt sind: disk full
- Theoretisch möglich: Auf der Platte sind noch viele Megabytes an Platz verfügbar
- Diese können aber nicht genutzt werden, weil nicht genügend I–Nodes vorgesehen waren

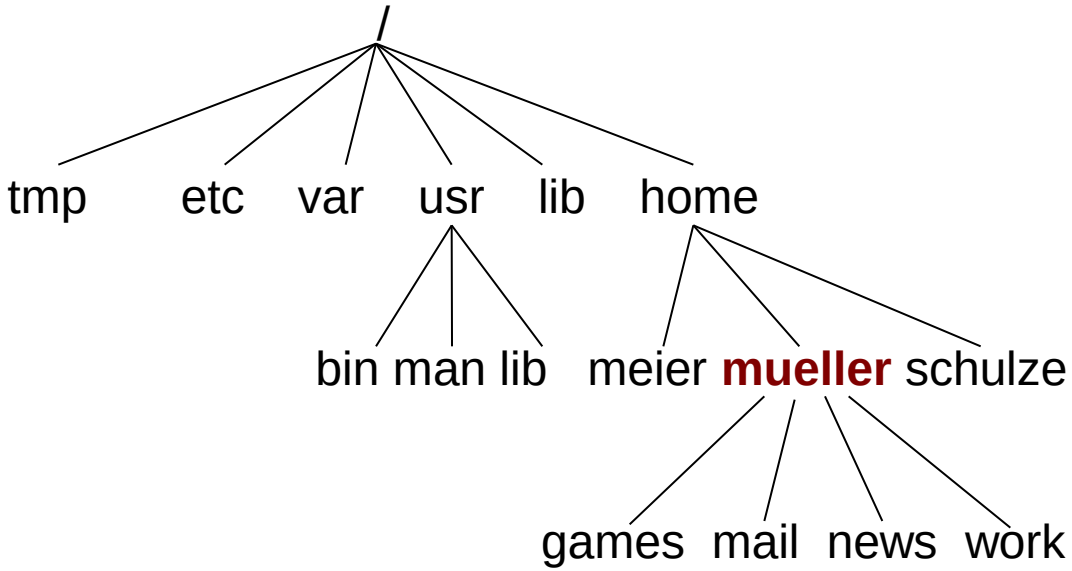
Vorteile I-Nodes

- Die Größe eines I-Nodes ist fix und relativ klein
- I-Node kann dadurch lange im Hauptspeicher bleiben
- Auf kleinere Dateien kann direkt zugegriffen werden
- Die theoretische maximale Größe einer Datei sollte ausreichend sein

Realisierung von Verzeichnissen (1)

- Verzeichnisse sind ebenfalls Dateien
- Im I-Node: Typ-Feld kennzeichnet Verzeichnis
- Verzeichnis liefert eine Abbildung von Datei- bzw. Verzeichnisnamen auf I-Node-Nummern
- Jeder Verzeichniseintrag ist ein Paar aus Name und I-Node-Nummer
- Über die I-Nodes kommt man dann zu Dateiinhalten

Realisierung von Verzeichnissen (2)



aktuelles und Vorgänger-verzeichnis

.	7
..	3
games	45
mail	76
news	9
work	14

I-Node von mueller

I-Node von home

I-Node Nr. 45

I-Node Nr. 76

I-Node Nr. 9

I-Node Nr. 14

Zeiger auf die Datenblöcke der Datei

Realisierung von Verzeichnissen (3)

```
$ ls -li
total 52
72091596 drwxr-xr-x 5 wachaja ais  4096 Jun  9 14:29 git
79692648 drwxr-xr-x 2 wachaja ais  4096 Dez  4  2014 signs
79954273 -rw-r--r-- 1 wachaja ais 42570 Mär 17  2015 test.ods
```

- Option `-i` für Ausgabe der I-Node-Nummern
- I-Node-Nummer am Zeilenanfang
- `total`: Gesamte Blockgröße in KiB aller Verzeichnisinhalte

Beispiel (1)

- Annahme: Wir sind in Verzeichnis `mueller`
- Wir wollen auf `../meier/datei1` zugreifen
- Bestimme im aktuellen Verzeichnis die I-Node-Nummer des Vorgänger-Verzeichnisses, hier: 3

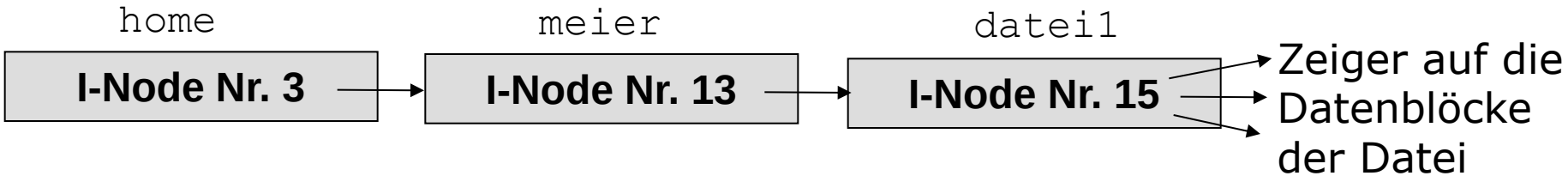
.	7
..	3
games	45
mail	76
news	9
work	14

Beispiel (1)

- Annahme: Wir sind in Verzeichnis `mueller`
- Wir wollen auf `../meier/datei1` zugreifen
- Bestimme im aktuellen Verzeichnis die I-Node-Nummer des Vorgänger-Verzeichnisses, hier: 3
- Greife über I-Node 3 auf Inhalt von Vorgänger-Verzeichnis (`home`) zu
- Bestimme dadurch den I-Node des Verzeichnisses `meier`
- In `home` gibt es z.B. den Eintrag `meier`, 13

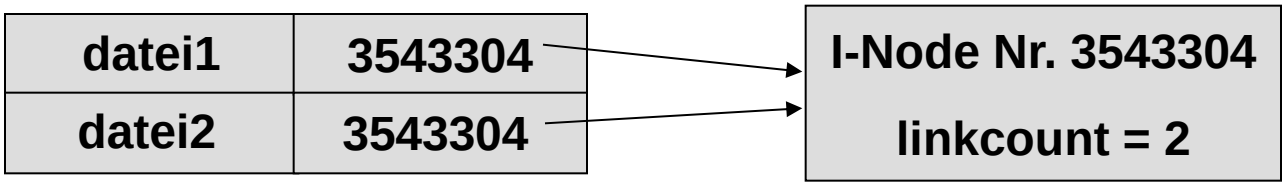
Beispiel (2)

- Also: Greife über I-Node 13 auf Inhalt von `meier` zu
- Dort steht Eintrag `datei1`, 15
- Also: Greife über I-Node 15 auf die Inhalte von `datei1` zu



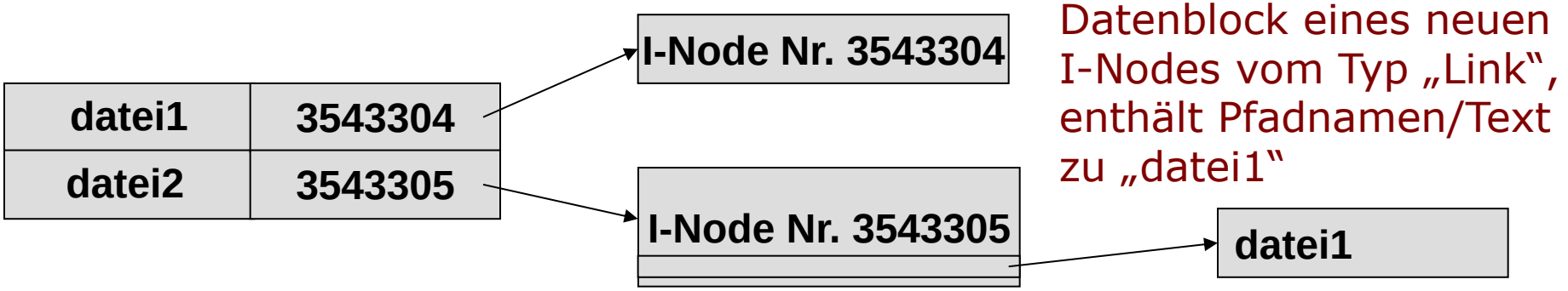
Implementierung von Hardlinks

```
$ ls -li
total 4
3543304 -rw-r--r-- 1 wachaja ais 4 Nov  9 18:16 datei1
$ ln datei1 datei2
$ ls -li
total 8
3543304 -rw-r--r-- 2 wachaja ais 4 Nov  9 18:16 datei1
3543304 -rw-r--r-- 2 wachaja ais 4 Nov  9 18:16 datei2
```



Implementierung symbolischer Links

```
$ ls -li
total 4
3543304 -rw-r--r-- 1 wachaja ais 4 Nov  9 18:16 datei1
$ ln -s datei1 datei2
$ ls -li
total 4
3543304 -rw-r--r-- 1 wachaja ais 4 Nov  9 18:16 datei1
3543305 lrwxrwxrwx 1 wachaja ais 6 Nov  9 18:22 datei2
   -> datei1
```

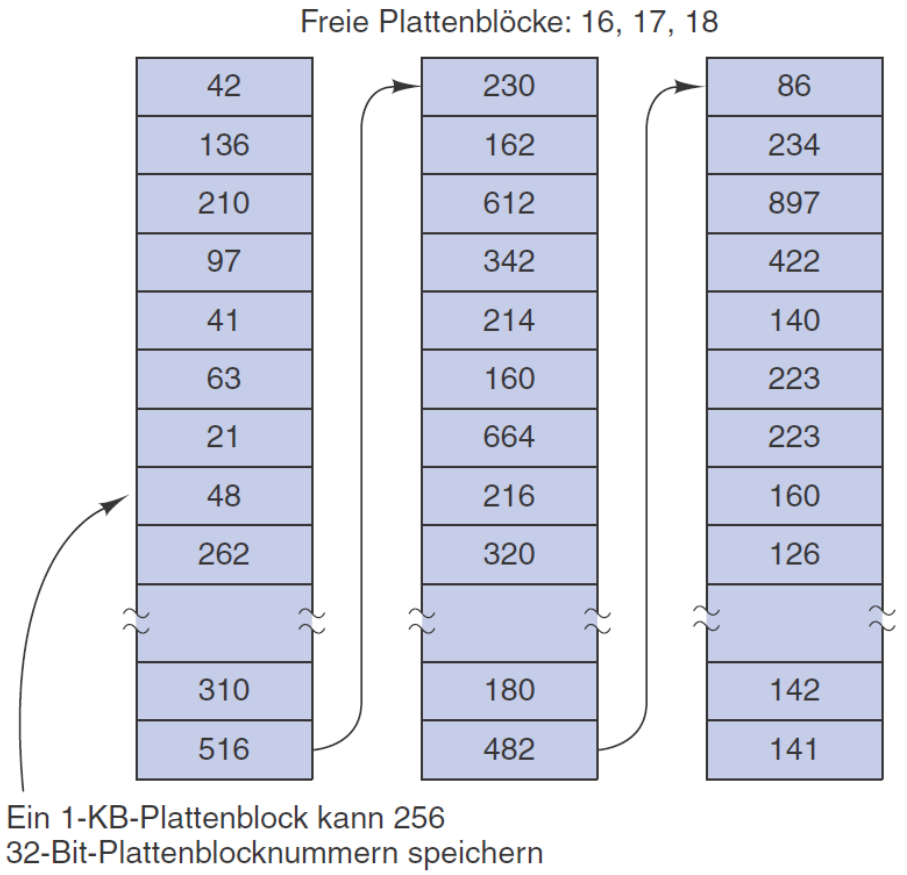


Verwaltung freier Plattenblöcke

- Abspeichern von Dateiinhalten in Datenblöcke an beliebigen Stellen im Dateisystem
- Nicht notwendigerweise aufeinanderfolgend
- Freie Blöcke müssen verwaltet werden

Verwaltung durch verkettete Liste von Plattenblöcken (1)

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

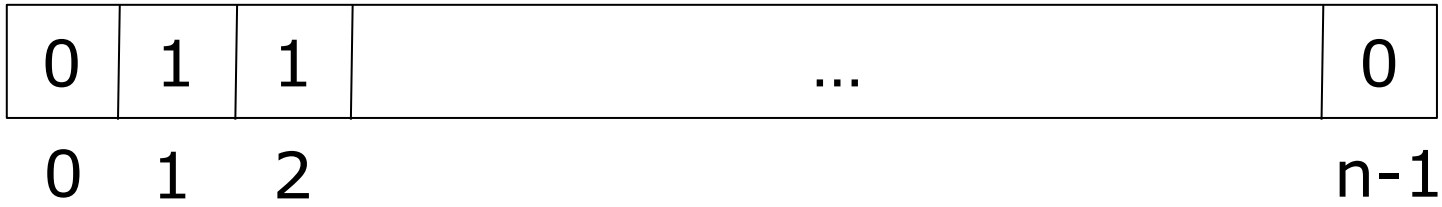


Verwaltung durch verkettete Liste von Plattenblöcken (2)

- Keine Verschwendung von Speicherplatz, da Blocknummern in den leeren Datenblöcken selbst gespeichert
- Eine große Anzahl von freien Blöcken kann schnell gefunden werden
- Schwierig, zusammenhängenden Speicherbereich zu finden
- Erweiterung: Speichere Anfangsadresse und Größe von *n* zusammenhängenden Blöcken

Verwaltung durch Bitmap

- Bitmap mit 1 Bit für jeden Plattenblock
- Plattenblock frei: entsprechendes Bit = 0



- Zusätzliche Blöcke auf der Platte werden zur Speicherung der Bitmap benötigt
- Suche nach Freibereich kann ineffizient sein
- Vereinfacht das Erzeugen von Dateien aus zusammenhängenden Blöcken

Belegung des Dateisystems

- Belegung gegeben durch Anzahl genutzter I-Nodes und Datenblöcke
- Dateisystem voll, wenn
  - Entweder keine I-Nodes mehr frei oder
  - Keine Datenblöcke mehr frei
- Überprüfung mittels `df -i`:

```
$ df -i
Filesystem      Inodes   IUsed     IFree IUse% Mounted on
/dev/sda2      5332992  801126   4531866   16% /
/dev/sdb1    122101760  569674 121532086    1% /export
```

Maximale Systemgröße ext3

- Standard Blockgröße: 4 KiByte
- Bei ext3: 32-Bit Blocknummern
- 2^{32} Adressierungsmöglichkeiten von Blöcken
- Größe von Dateisystem beschränkt auf $2^{32} * 4$ KiByte = $2^{32} * 2^{12}$ = 16 TiB

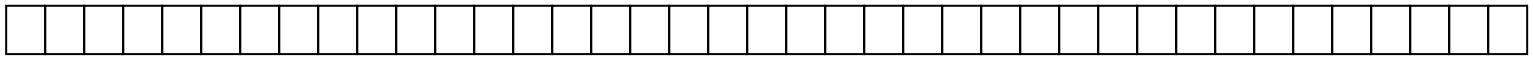
Maximale Systemgröße ext4

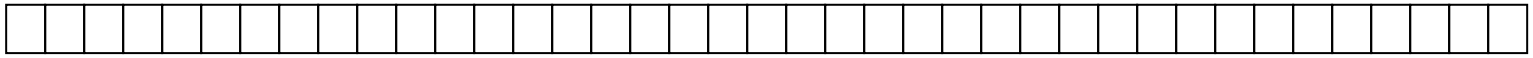
- Standard Blockgröße: 4 KiByte
- Bei ext4: 48-Bit Blocknummern
- Dadurch möglich: Dateisystem mit $2^{48} * 4$ KiByte = $2^{48} * 2^{12}$ = 1 EiB
- Indirekte Blockadressierung ersetzt durch „Extents“ = Bereiche von Datenblöcken

Wert	Kürzel	Bezeichnung
1024^0	B	Byte
1024^1	KiB	Kibibyte
1024^2	MiB	Mebibyte
1024^3	GiB	Gibibyte

Wert	Kürzel	Bezeichnung
1024^4	TiB	Tebibyte
1024^5	PiB	Pebibyte
1024^6	EiB	Exbibyte
1024^7	ZiB	Zebibyte

Beispiel: Extents



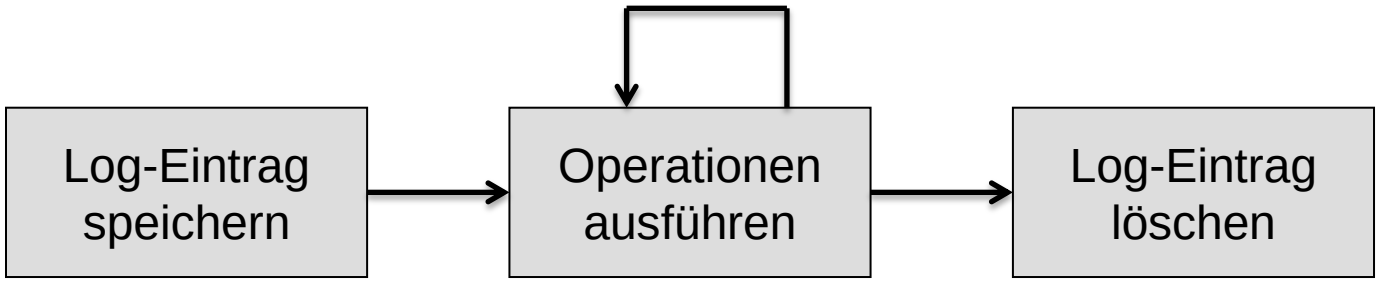


Journaling-Dateisysteme (1)

- Problem: Inkonsistentes Dateisystem nach Systemabsturz
- Beispiel: Löschen einer Datei unter UNIX
  - Löschen der Datei aus dem Verzeichnis
  - Freigabe des I-Nodes
  - Freigabe der Plattenblöcke
- Inkonsistenzen entstehen, falls nicht alle Operationen ausgeführt wurden
- Caching gefährlich bei Dateisystemen ohne Journaling

Journaling-Dateisysteme (2)

- Grundidee Journaling:
  - Log-Eintrag mit Operationen auf Platte speichern
  - Operationen ausführen
  - Log-Eintrag löschen
- Nach Systemabsturz können noch ausstehende Operationen durchgeführt werden
- Journaling-Dateisysteme: ext3, ext4, NTFS



RAID

- RAID = Redundant Array of Inexpensive / Independent Disks
- Idee: Ein Controller fasst mehrere Festplatten zu einer virtuellen zusammen
  - Höhere Performance
  - Mehr Zuverlässigkeit
  - Beides
- 6 verschiedene Level und Zwischenstufen

RAID-Level

RAID 0
Striping

Disk 0	Disk 1
Stripe 0	Stripe 1
Stripe 2	Stripe 3
Stripe 4	Stripe 5
Stripe 6	Stripe 7
Stripe 8	Stripe 9

- Hohe Performance
- Keine Redundanz
- Erhöhtes Ausfallrisiko

RAID 1
Mirroring

Disk 0	Disk 1
Stripe 0	Stripe 0
Stripe 1	Stripe 1
Stripe 2	Stripe 2
Stripe 3	Stripe 3
Stripe 4	Stripe 4

- Schneller Lesezugriff
- Hohe Redundanz

RAID 5
Block-Level Striping mit verteilter Parität

Disk 0	Disk 1	Disk 2	Disk 3
Stripe 0	Stripe 1	Stripe 2	P0-2
Stripe 3	Stripe 4	P3-5	Stripe 5
Stripe 6	P6-8	Stripe 7	Stripe 8
P9-11	Stripe 9	Stripe 10	Stripe 11
Stripe 12	Stripe 13	Stripe 14	P12-14

- Schneller Lesezugriff
- Redundanz beim Ausfall einer Platte
- Effiziente Festplattennutzung

Zusammenfassung (1)

- Verwaltung von Objekten typischerweise in Verzeichnisbäumen
  - Jede Partition braucht ein Dateisystem, um Daten zu verwalten
- Der Verzeichnisbaum kann aus mehreren Dateisystemen zusammengebaut sein
- Hauptfragen:
  - Wie wird Speicherplatz verwaltet?
  - Welche Blöcke gehören zu welcher Datei?

Zusammenfassung (2)

- Links erzeugen neue Verweise auf vorhandene Objekte
- Mögliche Implementierungen für Speicherplatzbelegung:
  - Zusammenhängende Belegung
  - Verkettete Listen (+ Dateiallokationstabellen)
  - I-Nodes
- Der Hauptunterschied liegt in der Effizienz (Speicherbelegung und Zugriffszeit)