DHBW
Duale Hochschule
Baden-Württemberg
Mannheim

Betriebssysteme

Kapitel 4 Dateiverwaltung

Ziele der Vorlesung

- Einführung
  - -Historischer Überblick
  - -Betriebssystemkonzepte
- Prozesse und Threads
  - -Einführung in das Konzept der Prozesse
  - -Prozesskommunikation
  - -Scheduling von Prozessen
  - -Threads
- Speicherverwaltung
  - -Einfache Speicherverwaltung
  - -Virtueller Speicher
  - -Segmentierter Speicher
- Dateien und Dateisysteme
  - -Dateien
  - -Verzeichnisse
  - -Implementierung von Dateisystemen
- Grundlegende Eigenschaften der I/O-Hardware
  - -Festplatten
  - -Terminals
  - -Die I/O-Software
- Deadlocks/Verklemmungen
- Virtualisierung und die Cloud
- Multiprozessor-Systeme
- IT-Sicherheit
- Fallstudien

Dateiverwaltung

- Bisher:
  - Prozesse nutzen Adressraum zur Speicherverwaltung
- Probleme und daraus resultierende Anforderungen:
  - Während der Ausführungszeit kann Prozess eine begrenzte Menge von Informationen im eigenen Adressraum abspeichern,die Speicherkapazität ist durch die Größe des virt. Adressraums beschränkt, z.B. Flugbuchung, Bankensoftware,...
    Anforderung: Es muss möglich sein, eine grosse Menge von Informationen zu speichern
  - Wenn der Prozess beendet wird, sind die Daten verloren
    Anforderung: Information muss auch nach der Beendigung des jeweils auf sie zugreifenden Prozesses noch erhalten bleiben
  - Mehrere Prozesse greifen gleichzeitig auf die gleichen Informationen (oder Teile davon) zu
    Anforderung: Information muss außerhalb des Adressraumes gehalten werden

→ Für das Speichern werden heute meist Flashsysteme benutzt

Fragen resultierend aus der Speicherverwaltung

- Wie findet man die gespeicherte Information wieder?
- Wie verhindert man, daß ein Benutzer die Daten des anderen liest?
- Woher weiß man, welche Blöcke unbenutzt sind?

Vergleiche Prozessor und physischer Speicher

- Betriebssystem abstrahiert den Prozessor durch Prozess
- Betriebssystem abstrahiert den phys. Speicher durch einen virtuellen Adressraum

→ Abstraktion der Information auf Speichermedium ist Datei

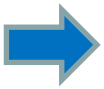
Dateiverwaltung

- Neue Abstraktion: Dateien
  - logische Informationseinheit, die von Prozessen erzeugt werden (eine Platte enthält gewöhnlich Tausende oder sogar Millionen, jede ist unabhängig von der anderen)
  - persistente Speicherung von Informationen, Datei verschwindet erst endgültig, wenn ihr Eigentümer sie ausdrücklich löscht
  - Dateien werden vom Betriebssystem verwaltet;
    - Wie sind Dateien strukturiert?
    - Wie werden sie benannt?
    - Wie wird auf sie zugegriffen?
    - Wie werden sie benutzt, geschützt, implementiert…
      → Der Teil des Betriebssystems, der für Dateien zuständig ist, ist das Dateisystem
  - Dateien modellieren die Platte (vergl. Adressraum modelliert den Arbeitsspeicher)

Dateiverwaltung

- Benutzersicht
  - Wie präsentiert sich die Datei bzw. das Dateisystem
    - Woraus besteht eine Datei?
    - Wie ist der Name?
    - Wie ist sie geschützt?
    - Welche Operationen sind auf ihr erlaubt?

- Entwicklersicht
  - Wie darf der Nutzer auf Dateien zugreifen?
  - Wie wird das System realisiert, z.B. Wieviele Sektoren sind in einem logischen Block untergebracht,
  - Werden für die Verwaltung des freien Speicherplatzes verkettete Listen oder Bitmaps benutzt?



Ziel ist Abstraktion zum Speichern und Lesen (eine einheitliche Schnittstelle zu den Datenträgern)

Aufgabe/Frage

Was ist das wesentlichste Merkmal von Dateien?
Wie sieht dieses Merkmal in Windows und Unix aus?

Bedingung:

→ ad hoc

Dateiverwaltung

- Dateien
  - Windows 95 und Windows 98: MS-DOS Dateisystem FAT-16
  - Windows 98: FAT-16 Erweiterung auf Dateisystem FAT-32
  - Windows NT, 2000, XP, Vista, Windows 7 und 8: FAT-16, FAT-32, NTFS (NTFS = Namensgebung in Unicode*)
  - Erweiterung von FAT-32 ist das neue FAT-basierte Dateisystem exFAT (für Flash, große Dateisysteme, OS X kompatibel)

***Was ist Unicode?**

- Unicode=Universelle Zeichencodierung (*„Universal Character Encoding")*
- **Standard zum Kodieren von Schriftzeichen in Binärdarstellung (fürs Speichern und Verarbeiten von Texten in digitalen Systemen)**
- nicht an die Formate und Kodierungen eines einzelnen Alphabets bestimmter menschlicher Sprache gebunden
- Ziel: einheitlicher Standard für die Abbildung **sämtlicher von Menschen entwickelter Schriftsysteme und Zeichen**
- Unicode wird **intern von Browsern und Betriebssystemen als einheitliches Format verwendet**.
- Das Unicode Consortium hat im Jahr 2022 die Version 15.0 veröffentlicht (insgesamt 149,186 Zeichen).
- Hier findet sich auch die Full Emoji List, v15.0: https://unicode.org/emoji/charts/full-emoji-list.html

Dateiverwaltung

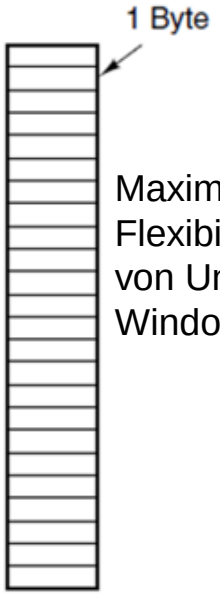
- Dateien
  - Zweigeteilte Dateinamen
  - Beide Teile durch Punkt getrennt
  - Teil hinter dem Punkt heißt Dateiendung oder Dateinamenserweiterung (engl. file extension)
    - Enthält bestimmte Informationen über die Datei
  - MS-DOS: 1-8 Buchstaben + optionale Erweiterung von 1-3 Buchstaben
  - Unix: Länge der Erweiterung benutzerüberlassen, auch mehrere Erweiterungen möglich; Punkt ist kein Sonder- bzw. Trennzeichen

Erweiterung	Bedeutung
.bak	Sicherungsdatei
.c	C-Quelltextdatei
.gif	Bilddatei im Compuserve Graphical Interchange Format
.hlp	Hilfedatei
.html	Dokument in Hypertext Markup Language für das WWW
.eps	Bilddatei nach dem JPEG-Standard codiert
.mp3	Musikdatei im MPEG-Layer-3-Format
.mpg	Film nach dem MPEG-Standard codiert
.o	Objektdatei (übersetzt, noch nicht gebunden)
.pdf	Datei im Portable Document Format
.ps	PostScript-Datei
.tex	Eingabedatei für das TeX-Satzsystem
.txt	Allgemeine Textdatei
.zip	Komprimiertes Archiv

Abbildung 4.1: Einige typische Dateiendungen.

- Dateistruktur

Unstrukturierte Byte-Folge

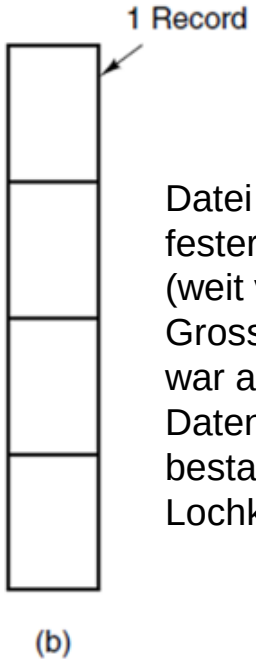


(a)

Maximum an Flexibilität (benutzt von Unix, Linux, Windows)

Betriebssystem weiß nichts über den Inhalt; die Interpretation muss von den Programmen auf der Benutzerebene kommen

Datensätze mit interner Struktur



(b)

Datei = Sequenz von Datensätzen fester Größe, mit interner Struktur (weit verbreitet in Zeiten von Grossrechnern, z.B. Dateisystem war aus Dateien aufgebaut, die aus Datensätzen mit 80 Zeichen bestanden, also Abbildungen der Lochkarten)

Konzept besteht darin, daß Leseoperationen jeweils einen Datensatz zurückgeben und Schreiboperationen jeweils einen Datensatz überschreiben bzw. anhängen.

Dateiverwaltung

- Dateitypen
  - Reguläre Dateien
    - Enthalten Benutzerinformationen
    - ASCII- oder Binärdateien
      - ASCII-Dateien
        - lassen sich anschauen und ausdrucken und mit Texteditor bearbeiten
        - ASCII-Dateien bestehen aus Textzeilen (Zeilenende=CR)
      - Binärdateien
        - Ausgabe von Binärdateien würde unverständliche Zeichenfolge von seltsamen Zeichen ergeben.
        - Die innere Struktur von Binärdateien ist dem Programm bekannt
  - Verzeichnisse
    - Systemdateien zur Verwaltung der Struktur eines Dateisystems
  - Zeichendateien
    - Beziehen sich auf Ein-/Ausgabe und modellieren serielle Ein-/Ausgabegeräte (Drucker, Terminal (/dev/tty), …)
  - Blockdateien
    - Zur Modellierung von Plattenspeicher (/dev/hda)

Aufgabe/Frage

- Was bedeutet ASCII?
- Was bedeutet EBCDIC?

Bedingung:
→ ad hoc

- Dateitypen
  - Technisch gesehen nur eine Folge von Bytes
  - Betriebssystem führt Datei nur dann aus, wenn Datei richtiges Format hat

Magische Zahl kennzeichnet die Datei als ausführbar und beschreibt den genauen Dateityp

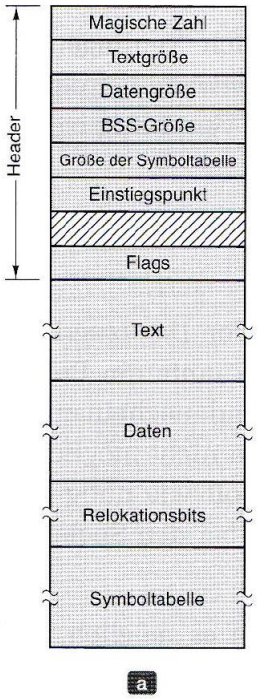


Abbildung 4.3: (a) Ausführbare Datei.

Format besteht aus 5 Abschnitten
- **Header**
- **Text**
- **Daten**
- **Relokationsbits und**
- **Symboltabelle**

Magische Zahl im Header kennzeichnet die Datei
- als ausführbar
- beschreibt den genauen Dateityp

Danach folgen im Header
- Größenangaben der verschiedenen Abschnitte der Datei
- Anfangsadresse, von der die Ausführung beginnt und
- einige Markierungsbits

Es folgt der Programmtext und Programmdaten

Anordnung von Programmtext und –daten anhand Relokationsbits

Symboltabelle für Fehlerbeseitigung

Dateiverwaltung

- Dateizugriff
  - Sequentieller Zugriff (sequential access)
    - In sehr alten Systemen einzige Art des Zugriffs
    - Einlesen der Bytes (oder Datensätze) einer Datei von Anfang an
      - Überspringen war nicht möglich
      - Zugriff ausserhalb der Reihenfolge nicht möglich
      - Seq. Dateien konnten zurückgespult werden (z.B. Magnetbänder)

  - Wahlfreier Zugriff (random access)
    - Einlesen der Bytes (oder Datensätze) einer Datei in beliebiger Reihenfolge
      - Unentbehrlich für Anwendungen (Datenbanken)
      - „read“ gibt Startposition in der Datei an
      - „seek“ legt die Position des sog. Lese/Schreibzeigers (auch Dateizeiger) innerhalb der Datei fest
        - Danach wird Datei sequenziell von dieser aktuellen Position aus gelesen
      - Festplatte

Aufgabe/Frage

Sie kennen nun die Informationseinheit Datei.

Fragen:

- Welche Informationen über Dateien wollen **SIE** als Nutzer kennen?
  (Hilfe: Welche Attribute kennen Sie?)
- Erstellen Sie eine Liste!

Bedingung:

→ 3 min

Dateiverwaltung

- Dateiattribute
  - Datei hat einen Name und die darin abgespeicherten Daten
  - Zusätzliche Informationen über Dateien werden Attribute (oder Metadaten) genannt
    - Von System zu System variiert der Inhalt der Attribute
    - Mögliche Attribute:
      Urheber, Eigentümer, Datensatzlänge, Aktuelle Größe, Maximale Größe, Read-Only-Flag, Zeitpunkt der Erstellung, Zeitpunkt des letzten Zugriffs, Zeitpunkt der letzten Änderung, Hidden-Flag, System-Flag, Random-Access-Flag, Schlüsselposition, Schlüssellänge, Datensatzlänge, …

Dateiverwaltung

- Dateioperationen
  - Dateien sollen Informationen zum späteren Abruf speichern
  - Systemaufrufe
    - Create
    - Delete
    - Open
    - Close
    - Read
    - Write
    - Append
    - Seek
    - SetAttribute
    - GetAttribute
    - Rename

Dateiverwaltung

- Dateiorganisation
  - Dateisysteme haben Verzeichnisse (directories) oder Ordner (folder)
  - Flache Organisation (alle Dateien in einer Ebene)
    - Einfachste Form eines Verzeichnissystems ist **EIN** Verzeichnis, das alle Dateien beinhaltet
    - Dieses **EINE** Verzeichnis wird Wurzelverzeichnis genannt (Anfänge PC-Systeme und eingebettete Systeme)
    - Dateien werden schnell gefunden, weil nur an einem Ort gesucht werden muss

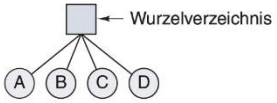


Abbildung 4.6: Ein Verzeichnissystem mit einer Ebene und vier Dateien

  - Hierarchische Organisation
    - Über Baum-Struktur (Verzeichnisbaum)
    - Beliebige Anzahl von Unterverzeichnissen
    - Mächtiges Werkzeug zur Strukturierung der Arbeit

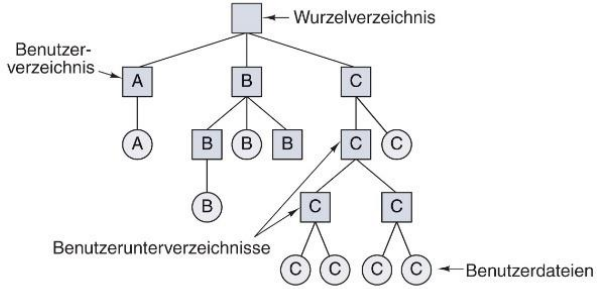


Abbildung 4.7: Ein hierarchisches Verzeichnissystem

Verzeichnisse A, B, C im Wurzelverzeichnis gehören zu verschiedenen Benutzern, zwei haben Unterverzeichnisse

Dateiverwaltung

- Navigation im Verzeichnisbaum
  - Weg zum Dateinamen wird benötigt
  - Pfadnamen zur Spezifizierung der Dateien im Verzeichnisbaum
    - **Absolute Pfadnamen**
      - Gesamter Pfad von Wurzel bis zur Datei
      - Separatoren trennen die einzelnen Bestandteile (Windows=\; Unix=/)
      - Wenn Erster Buchstabe = Separator, dann Pfadname absolut

        Beispiele:
        Windows: C:\dokumente\lohnert\betriebssysteme.odp
        Unix: /usr/lohnert/betriebssysteme.odp

    - **Relative Pfadnamen**
      - Arbeitsverzeichnis
        - Benutzer gibt aktuelles Verzeichnis als Arbeitsverzeichnis an
        - Startpunkt für relative Pfadnamen
      - Spezialeinträge
        - „.“ bestimmt das aktuelle Verzeichnis,
        - „..“ bestimmt das übergeordnete Verzeichnis

Aufgabe/Frage

Aufgabe 1:

Gegeben sind 2 Unix-Kommandos

cp /usr/ast/mailbox /usr/ast/mailbox.bak

UND

cp mailbox mailbox.bak

Was ist die Voraussetzung, daß beide Kommandos das Gleiche bewirken?

Aufgabe 2:

Ein Prozess hat /usr/ast als Arbeitsverzeichnis.

Mit welchen Befehlen lässt sich /usr/lib/dictionary in sein eigenes Verzeichnis kopieren?

Bedingung:

→ 5 min

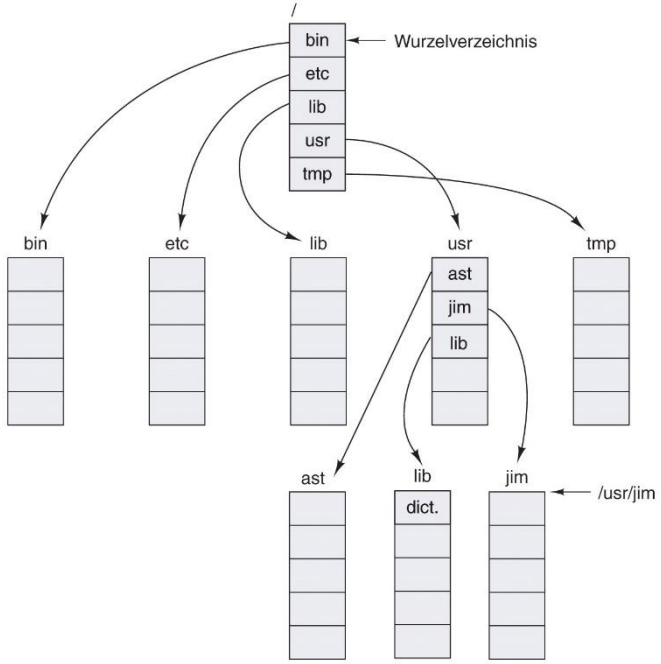


Abbildung 4.8: UNIX-Verzeichnisbaum

5 min

Dateiverwaltung

- Verwaltung des Verzeichnisbaums
  - Operationen auf Verzeichnisse variieren von System zu System
  - Übliche Systemaufrufe aus der Unix-Welt
    - Create, Delete
    - Opendir, Closedir, Readdir
    - Rename
    - Link, Unlink

Aufgabe

- Bisher haben wir das Konzept Dateien aus der Sicht des Benutzers besprochen
  - Dateien speichern Information
  - Auf Dateien wird zugriffen
  - Dateien werden über Verzeichnisse organisiert

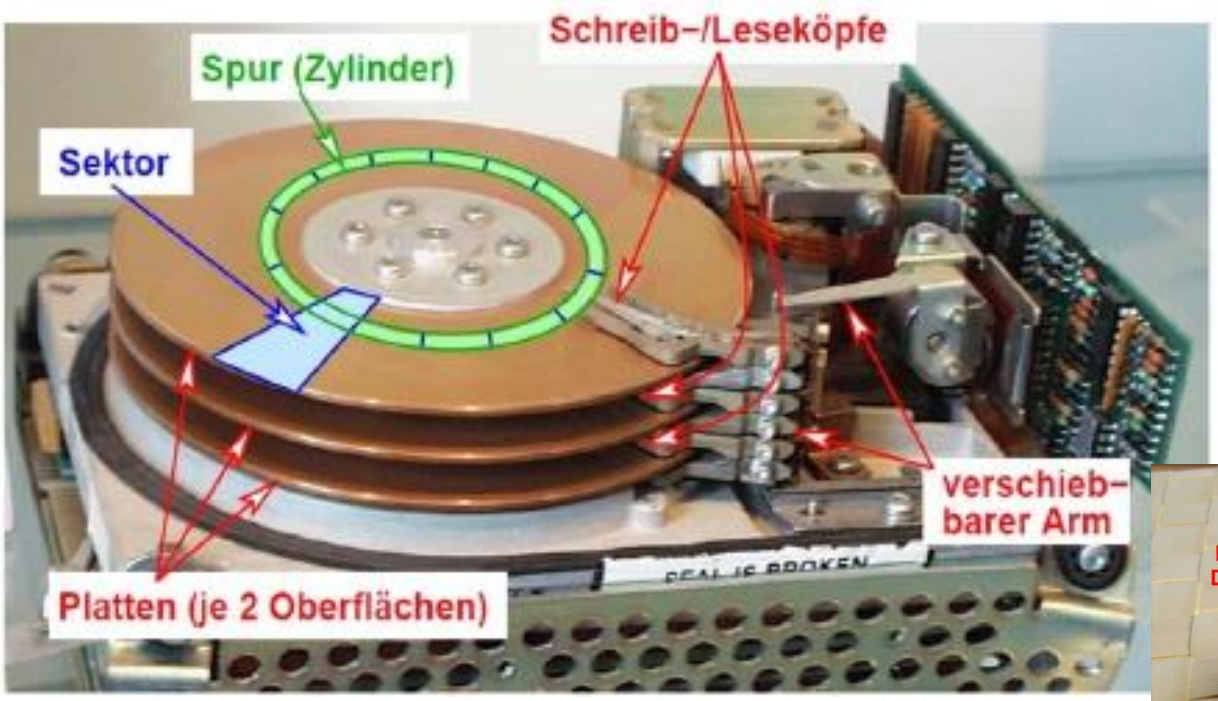
- Jetzt wollen wir in die Rolle des Betriebssystem-Entwicklers wechseln!
  - Was muss der Entwickler über das System wissen?
  - Welche Fragen muss der Entwickler beantworten, um die Abstraktion „Datei“ zu implementieren?

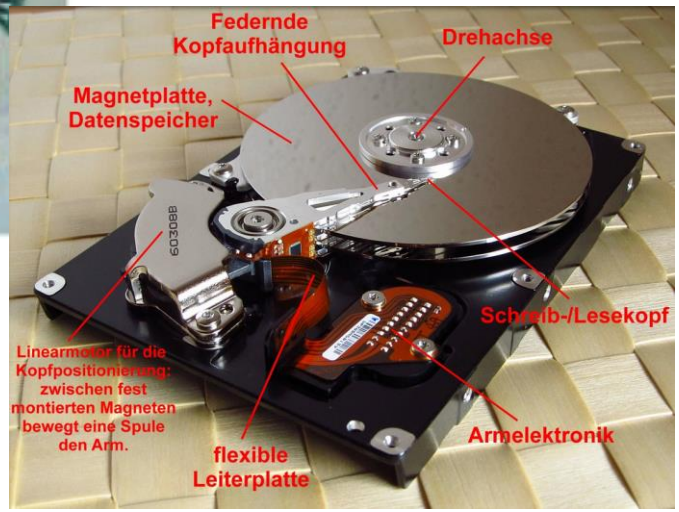
Bedingung:

→ 5 min

Dateiverwaltung

- Festplatte





https://www.lokalkompass.de/xanten/c-ratgeber/aufgeschraubt-eine-computer-festplatte-von-innen_a363614#gallery=default&pid=4766391

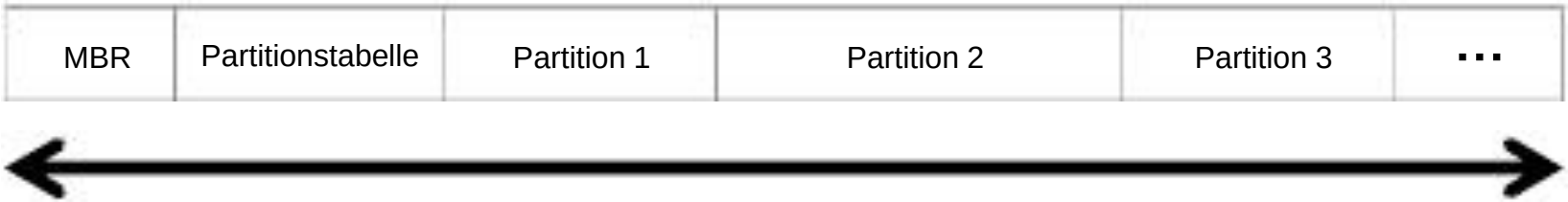
- SSD

  Eine SSD ist ein **elektronisches Speichermedium**, das aus Flash-Speicherbausteinen besteht (Synchronous Dynamic Random Access Memory). SSDs sind **deutlich schneller** (max. 600 MB/s) als Festplatten (max. 120 MB/s). SDDs sind **geräuschlos , kühler, robuster** (Sturz-unanfälliger) und wartungsärmer, weil sie keine beweglichen Teile verbaut haben. SSDs haben eine begrenzte Lebensdauer, je nach Speichertechnik. Besonders lang leben **Single-Level-Cell-SSDs** (SLC), die allerdings nur 1 Bit pro Speicherzelle speichern können. Sie vertragen bis zu 100.000 Schreibzyklen pro Zelle und gelten als besonders schnell, langlebig und ausfallsicher. Multi-Level Cell-SSDs (MLC) haben eine höhere Speicherdichte und können 2 Bit pro Flash-Zelle speichern. Sie sind im Vergleich zum SLC-Typ kostengünstiger, vertragen aber nur bis zu 10.000 Schreibzyklen pro Zelle. Triple-Level-Cell-SSDs (TLC) können 3 Informationsbits pro Speicherzelle aufnehmen. Allerdings sinkt gleichzeitig auch die Lebenserwartung auf bis zu 3000 Speicherzyklen pro Zelle. Bei Quad-Level-Cell-SSDs (QLC) werden vier Informationsbits pro Zelle untergebracht. Hersteller garantieren meist lediglich 1000 Schreib- bzw. Löschzyklen pro Zelle.

- NVMe (Non-Volatile Memory Express)

  Non-Volatile Memory Express (NVMe) ist eine 2013 eingeführte Speicherschnittstelle. „Nichtflüchtig“ bedeutet, dass der Speicher beim Neustart Ihres Computers nicht gelöscht wird, während „Express“ die Tatsache bezeichnet, dass die Daten über PCI Express übertragen werden (PCIe) -Schnittstelle auf dem Motherboard Ihres Computers. Dadurch erhält das Laufwerk eine direktere Verbindung zu Ihrem Motherboard, da die Daten nicht über einen SATA-Controller (Serial Advance Technology Attachment) übertragen werden müssen.

- Layout Datenträger
  - Layout
    - Datenträger (meist Platte) sind in **Partitionen** eingeteilt

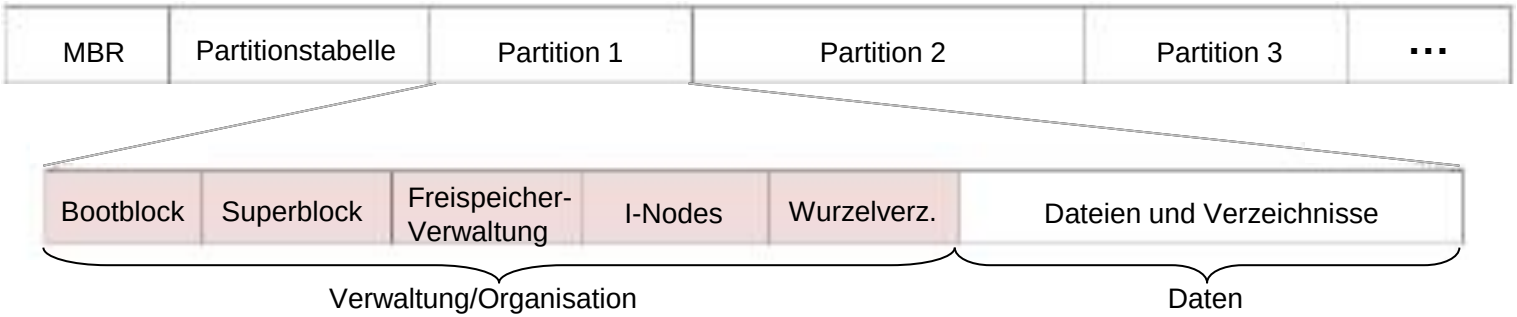


    - **Master Boot Record** (MBR):
      - wichtig beim Systemstart (Hochfahren, Sektor 0)
      - Am Ende des MBR steht die Partitionstabelle
        - enthält Anfangs- und Endadresse jeder Partition
      - Enthält einen Verweis auf die aktive Partition

- Beispiel: Hochfahren des Computer
  - Beim Start liest BIOS den MBR ein und führt ihn aus
  - MBR-Programm lokalisiert die aktive Partition, liest deren ersten Block (Boot-Block) und führt ihn aus
    - Jede Partition beginnt mit einem Boot-Block (auch wenn nicht bootfähig)
  - Programm im Boot-Block: Laden Betriebssystem, das in dieser Partition gespeichert ist.

Dateiverwaltung

- Layout Datenträger
  - Layout
    - Datenträger (meist Platte) ist in Partitionen eingeteilt
    - Layout Partitionen



- Boot-Block: Programm zum Laden des Betriebssystems
- SuperBlock: enthält alle Schlüsselparameter des Dateisystems (Typ, MagischeZahl, Anzahl der Blöcke, …)
- Freispeicherverwaltung: Information über die freien Blöcke im Dateisystem
- I-Nodes: Liste von Datenstrukturen, die alle Informationen über je eine Datei enthält
- Wurzelverzeichnis: Einstiegspunkt in die Baumstruktur des Dateisystems
- Uuuunnnnddddd die Dateien und Verzeichniss selbst!

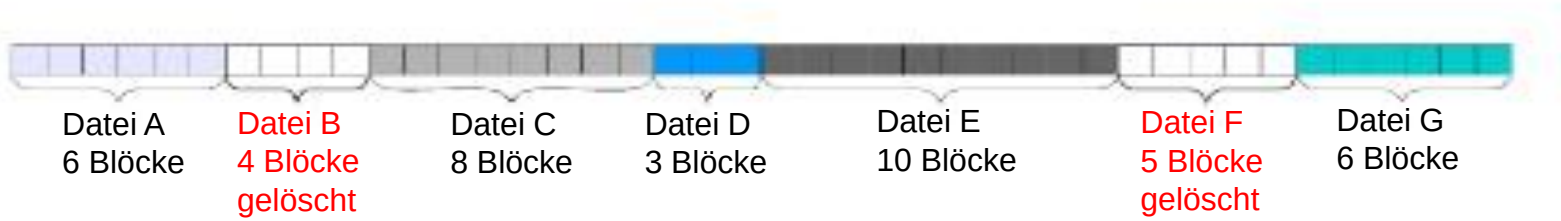
- Implementierung von Dateien
  - Wichtigster Aspekt: Welche Blöcke auf der Festplatte sind mit welcher Datei assoziiert?
  - **Ansatz 1: zusammenhängende Belegung**
    - Abspeichern als zusammenhängende Menge von Blöcken
      Beispiel: 1 KB Blockgröße, 50 KB Datei→50 aufeinander-folgende Blöcke

Datei A 6 Blöcke | Datei B 4 Blöcke | Datei C 8 Blöcke | Datei D 3 Blöcke | Datei E 10 Blöcke | Datei F 5 Blöcke | Datei G 6 Blöcke

Erklärung: Sieben Dateien; jede beginnt sofort hinter dem letzten Block der vorhergehenden Datei

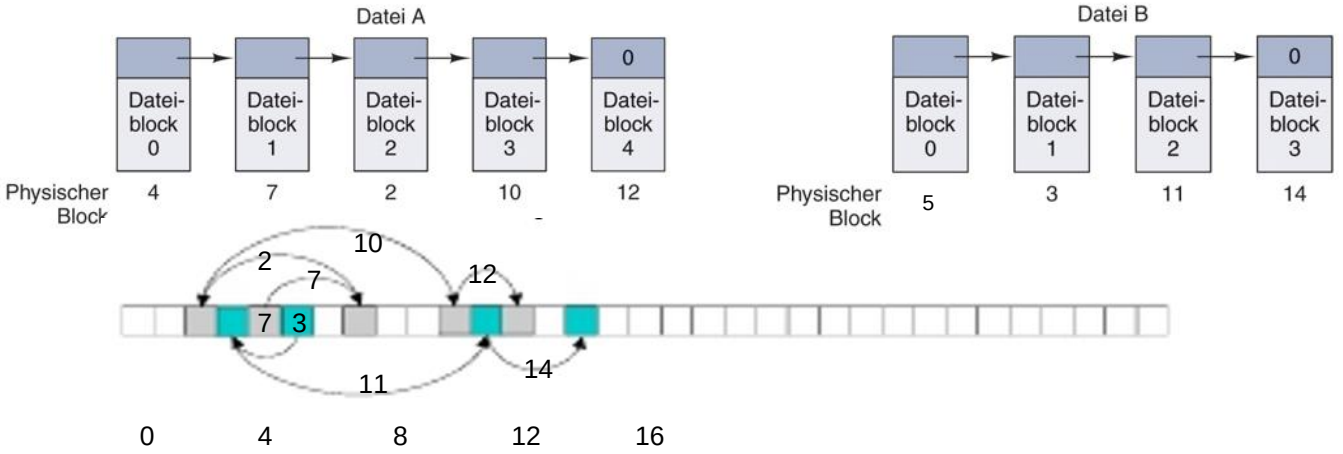
    - Vorteile:
      - Einfache Implementierung
        → Parameter (Plattenadresse des ersten Blocks und Anzahl der Blöcke)
      - Hervorragende Leseleistung
        → nur ein Positionierungschritt und ein Ausleseschritt

- Implementierung von Dateien
  - **Ansatz 1: zusammenhängende Belegung**
    - Nachteil: Beim Löschen beginnt die Platte zu fragmentieren



    - Mögliche Lösungen
      - Verdichten/Defragmentieren
        - Verschiebe alle Dateien um Lücken zu schließen
          → Sehr teuer
      - Verwaltung einer Liste mit freien Blöcken zur Wiederverwendung
        - Schon bei der Erzeugung der Datei muss die Größe bekannt sein
          → Kein realistisches Szenario
          → Verwendung auf CD-ROMS (Dateigrößen im voraus bekannt und ändern sich auch nicht während des Gebrauchs)

- Implementierung von Dateien
  - **Ansatz 2: Belegung durch verkettete Liste**
    - Datei ist verkettete Liste
      - Erster Eintrag im phys. Block ist Zeiger auf den nächsten Block

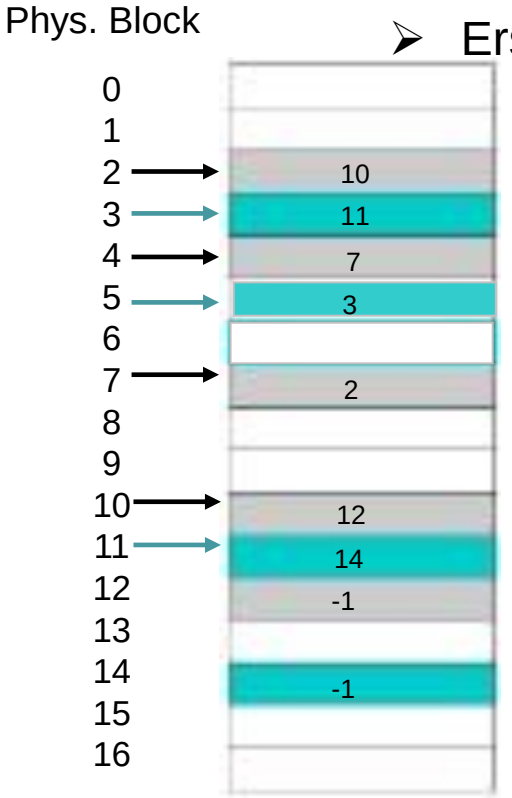


    - Vorteile:
      - Keine Fragmentierung
      - Als Parameter reicht die Nummer des ersten Blocks aus (Datei A=4, Datei B=5), Rest kann von dort gefunden werden

- Implementierung von Dateien
  - **Ansatz 2: Belegung durch verkettete Liste**
    - Nachteile:
      - Speicherplatz pro Block wird um den Zeiger verringert (um auf Block n zugreifen zu können, muss Betriebssystem am Anfang beginnen und die n-1 Blöcke davor einzeln lesen)
      - Viele Systemaufrufe zur Positionierung
        - Sequentielles Lesen ist akzeptabel
        - Wahlfreier Zugriff ist aber sehr teuer, da immer wieder vom Anfang begonnen werden muss

➡ Lösung: Lege die Zeiger jedes Plattenblocks in Tabelle in Arbeitsspeicher ab (FAT)

- Implementierung von Dateien
  - **Ansatz 3: Belegung durch verkettete Listen im Arbeitsspeicher**
    - Dateiallokationstabelle (File Allocation Table=FAT)
      - ➢ Erster Eintrag im phys. Block ist Zeiger auf den nächsten Block



Beispiel: Datei A (Beginn=4): 4-7-2-10-12;
Datei B (Beginn=5): 5-3-11-14

- Vorteile:
  - \+ Der gesamte Block steht zur Verfügung
  - \+ Wahlfreier Zugriff geht Schneller (im Arbeitsspeicher wird gesucht OHNE Plattenzugriffe davor
  - \+ immer noch **nur ein** Parameter für die Datei (Position des ersten Blocks in der Tabelle)

- Nachteile:
  - \- Tabelle benötigt Arbeitsspeicher
    - wächst linear mit der Größe der Platte,
    - Nicht bei großen Platten geeignet

Aufgabe/Frage

In einem MS-DOS Dateisystem (FAT) existiert folgender Verzeichniseintrag:

Fiffi	txt	Attribute	Zeit	Erste Block Nr. = 4	Größe

Die File Allocation Table (FAT) hat folgende Einträge:

Block #1	-1
Block #2	#6
Block #3	#17
Block #4	#2
Block #5	#1476
Block #6	#1

Welche Blöcke werden in welcher Reihenfolge gelesen?

3 min

Bedingung:

→ 3 min

- Implementierung von Dateien
  - **Ansatz 4: I-Nodes**
    - Jeder Datei wird eine Datenstruktur (I-Node=Index-Node) zugewiesen
    - Diese Datenstruktur listet die Attribute und Plattenadressen der Blöcke der Datei auf
    - Weiß man den Index Node: Alle Blöcke der Datei lassen sich aufspüren. Eindeutige Identifizierung I-Node innerhalb einer Partition durch seine I-Node-Nummer.
    - Jeder Namenseintrag in einem Verzeichnis verweist auf genau einen I-Node (enthält die Metadaten der Datei, verweist auf die Daten der Datei bzw. auf die Dateiliste des Verzeichnisses
    - Alle “ext” Dateisysteme verwenden I-Nodes. Die Anzahl verfügbarer Inodes wird bei der Erstellung des Dateisystems definiert und lässt sich anschliessend nicht mehr ändern (z.B. ein ext4-Inode hat eine Grösse von 256 Bytes und belegt daher zu viel Speicherplatz)

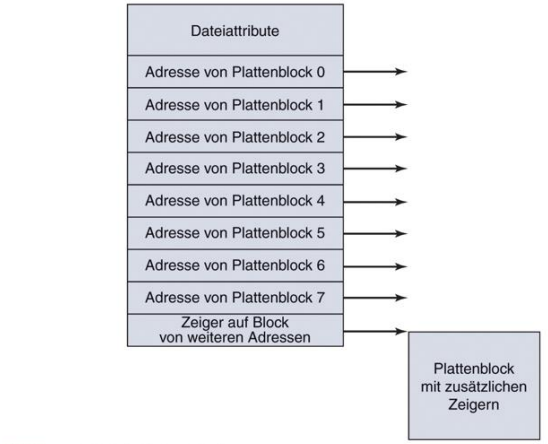
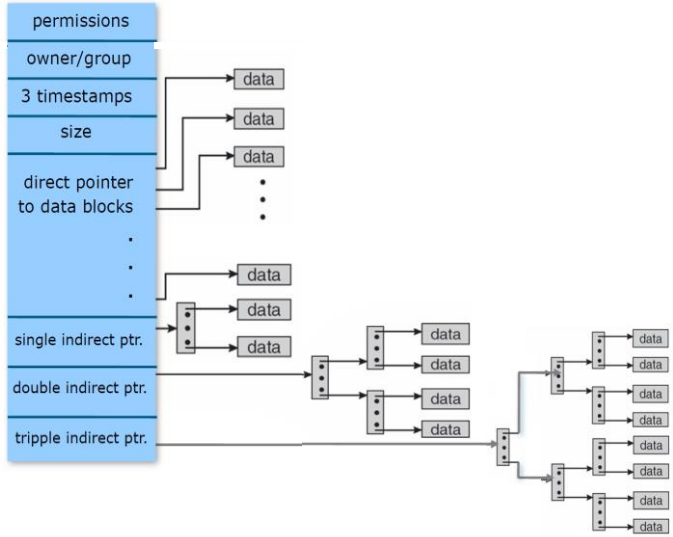


Abbildung 4.13: Beispiel eines I-Node

Grundlegende Datenstruktur zur Verwaltung von Dateisystemen mit Unix-artigen Betriebssystemen.

- Implementierung von Dateien
  - **Ansatz 4: I-Nodes**
    - Vorteile:
      - Mit I-Node sind alle Blöcke bekannt
      - I-Nodes müssen nur im Speicher sein, wenn Datei geöffnet ist
        - → Reservierter Speicher für I-Nodes hängt von der **Anzahl geöffneter Dateien** ab
        - → Im Gegensatz zu FAT (**Speicher von der Plattengröße abhängig)**
      - Umgang mit großen Dateien, die viele Blöcke haben
        - → Zeiger auf einen Block mit weiteren Adressen
        - → Hierarchie von I-Nodes entsteht
    - Beim Lesen und Schreiben von Daten: Anwendungssoftware unterscheidet nicht mehr zwischen Gerätetreibern und regulären Dateien.
    - Durch I-Node-Konzept: Bei Unixvarianten gilt alles als Datei mit Dateiattributen und Adressen aller Blöcke (Plattenadresse)

- Implementierung von Dateien
  - **Ansatz 4: I-Nodes**
    - Beispiel I-Node Zeigerstruktur
      - moderne Dateisysteme nutzen 15 Zeiger. Diese Zeiger beinhalten (von 15 ausgehend):
        - Zwölf Zeiger, die direkt auf Datenblöcke der Datei verweisen (direkte Zeiger)
        - Einen indirekten Zeiger (ein Zeiger, der auf Block von Zeigern verweist, dessen Zeiger auf die Daten verweisen)
        - Ein doppelt indirekter Zeiger (ein Zeiger, der auf Block von Zeigern verweist, dessen Zeiger auf weiteren Block aus Zeigern verweist, dessen Zeiger auf die Daten der Datei verweisen)
        - Ein dreifach indirekter Zeiger (ein Zeiger, der auf Block von Zeigern verweist, der auf Block von Zeigern verweist, der auf Block von Zeigern verweist, dessen Zeiger auf die Daten der Datei verweisen)



Aufgabe/Frage

Bisher haben wir uns über das Verbinden von Datenbereichen auf einer Festplatte zu einer Datei Gedanken gemacht!

Frage:

- Wie funktioniert die Zuordnung von Namen zu Dateien?

Bedingung:

→ ad hoc

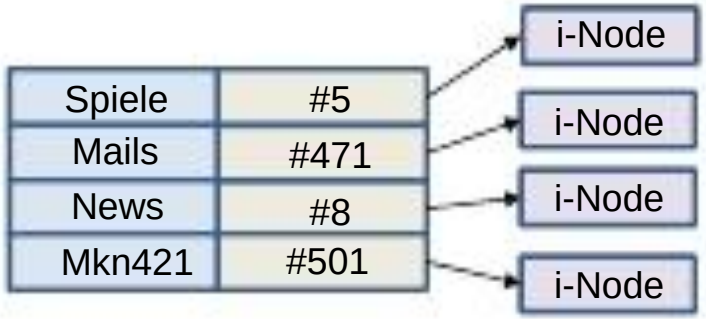
Dateiverwaltung

- Implementierung von Verzeichnissen
  - Verzeichnis = eine Datei mit Tabelle mit Verzeichniseinträgen
  - Verzeichniseintrag = Information zum Auffinden von Plattenblöcken und evtl. Information über Attribute der Datei
    - Abbilden von ASCII-Namen auf Datei-Information
      - Bei zusammenhängender Belegung: Plattenadresse und Größe
      - Bei verketteten Listen: die Nummer des ersten Blocks
      - Bei I-Nodes: die Nummer des I-Nodes

Spiele	Attribute	#5
Mails	Attribute	#471
News	Attribute	#8
Mkn421	Attribute	#501

Attribute sind Teil des Verzeichnis-Eintrags

Für jede Datei gibt es einen Eintrag mit einem Dateinamen (fester Größe), einer Struktur für die Dateiattribute und einer Plattenadresse



Attribute sind Teil des I-Nodes

Attribute in I-Nodes bedeutet, daß Verzeichniseinträge kürzer sind

Dateiverwaltung

- Implementierung von Verzeichnissen
  - Was passiert bei der Erzeugung einer Datei unter Linux?
    - I-Node wird für die Datei erzeugt
    - I-Node wird für das Verzeichnis aktualisiert (Attribute)
    - Neuer Eintrag wird in den Verzeichnisblock eingetragen
    - „Datei selbst wird geschrieben“

    Probleme:
    - Bei vielen Dateien steigt der Overhead an!
    - Wichtig: Die obigen Aktionen sollen zeitnah ausgeführt werden
    - Gefahr: Inkonsistenz-Probleme, wenn nicht alle Aktionen für „eine“ Datei aufgrund eines Systemabsturzes ausgeführt werden

Aufgabe/Frage

Jetzt haben wir ein allgemeines Konzept für Dateien.

- Dateien bestehen aus Blöcken und es wird über „Metadaten“ auf diese Blöcke zugegriffen
- Verzeichnisse sind Dateien mit Listen von Verzeichniseinträgen, die die Auflösung von Namen ermöglichen
- Änderungen werden sofort auf die Dateien angewendet durch Systemaufrufe

Fragen:

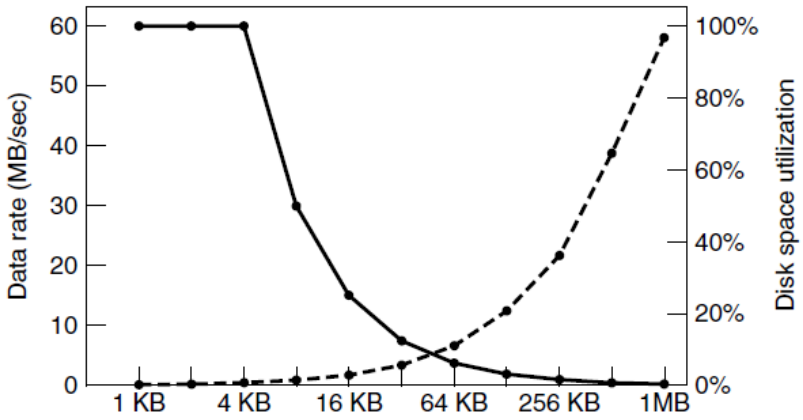
- Was muss die Implementierung eines Dateisystems noch leisten?

- Was muss bei der Anwendung des Konzepts auf eine reale Festplatte berücksichtigt werden?

Bedingung:

→ ad hoc

- Implementierung von Verzeichnissen
  - **Blockgröße** (Wie gross soll die Blockgröße sein?)
    - Orientierung an Plattenorganisation: Sektoren, Spuren, Zylinder
    - Orientierung an Seitengröße
    - Große Blöcke verschwenden Platz beim Abspeichern kleiner Dateien
    - Kleine Blöcke verschwenden Zeit durch viele Zugriffe und rotationsbedingte Wartezeiten, sind aber gut für Platzausnutzung
    - Optimum hängt von der **„üblichen“** Größe der Dateien ab



Größe aller Dateien
(übliche Größe)=4KB

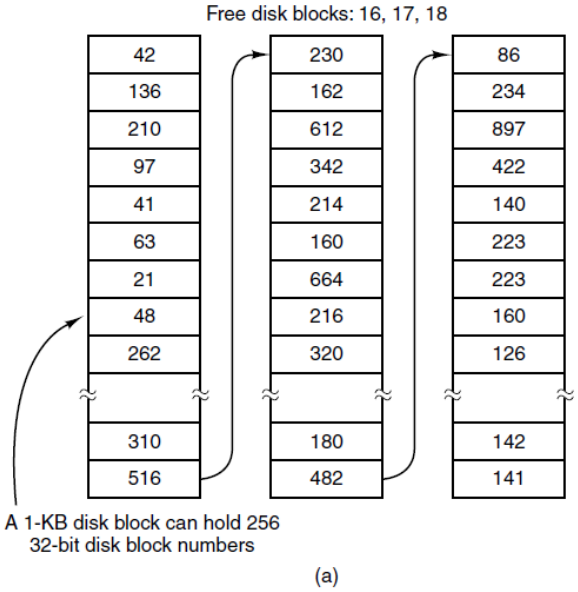
Gestrichelte Linie zeigt die Datenrate einer Platte als Funktion der Blockgröße
Durchgezogene Linie zeigt die Platzeffizienz der Platte.
Annahme: alle Dateien 4KB groß.

→ Zugriffszeit auf einen Block wird ausschließlich durch die Plattenzugriffszeit und die Rotationsverzögerung bestimmt; Datenrate steigt fast linear mit Blockgröße

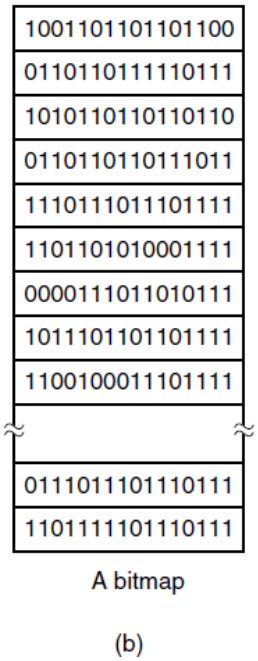
→ Platzeffizienz: Bei 4 KB-Dateien und Blockgrößen von 1KB, 2 KB und 4 KB keine Verschwendung; bei 8 KB-Block Platzeffizienz fällt auf 50%; bei einem 16 KB-Block weiter auf 25%.

→ Konkurrenz zw. Performanz und Speicherplatzausnutzung

- Implementierung von Verzeichnissen
  - **Verwaltung freier Blöcke** (Wie behält das Betriebssystem Überblick über die freien Blöcke?)
    - **Ansatz 1: Verkettete Listen von Plattenblöcken**
      - Jeder Block enthält so viele Blocknummern wie möglich
      - Bei einer Blockgröße von 1KB und 32-Bit Blocknummern kann jeder Block der Freibereichsliste 255 freie Blöcke aufnehmen (1 Eintrag wird benötigt für Zeiger)
      - Bei leerer Platte sehr groß
      - Freie Blöcke werden benutzt, um die Freibereichsliste aufzunehmen. Die Speicherung der freien Blöcke kostet keinen wertvollen Speicherplatz.
        - Beispiel: 1TB Platte hat ca. 1 Mrd. Plattenblöcke. Es werden ca. 4 Mill. Blöcke benötigt, um all diese Adressen (255 pro Block) zu speichern.



- Implementierung von Verzeichnissen
  - **Verwaltung freier Blöcke** (Wie behält das Betriebssystem Überblick über die freien Blöcke?)
    - **Ansatz 2: Bitmap**
      - Eine Platte mit n Blöcken benötigt eine Bitmap mit n Bit.
      - Darstellung der freien Blöcke durch eine 1, belegt durch 0 ODER umgekehrt
      - Nur bei voller Platte (wenige freie Blöcke) belegen die Bitmaps mehr Blöcke als die verketteten Listen
        - Beispiel: 1TB Platte benötigt 1 Mrd. Bit für die Bitmap; es werden 130.000 1KB-Blöcke für deren Speicherung benötigt.



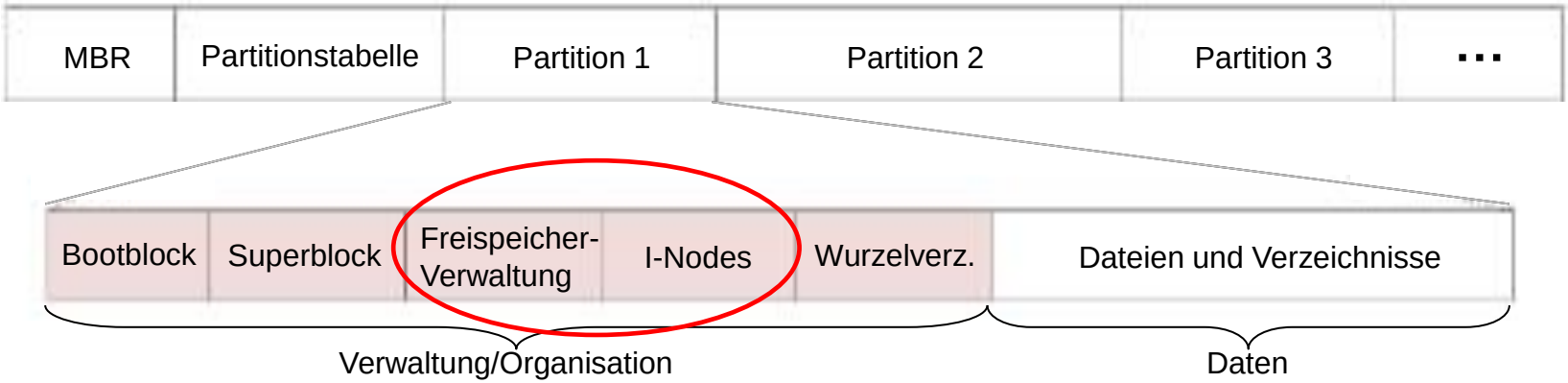
A bitmap

(b)

Bitmap benutzt nur 1 bit pro Block statt 32 Bit bei verketteten Listen.

Dateiverwaltung

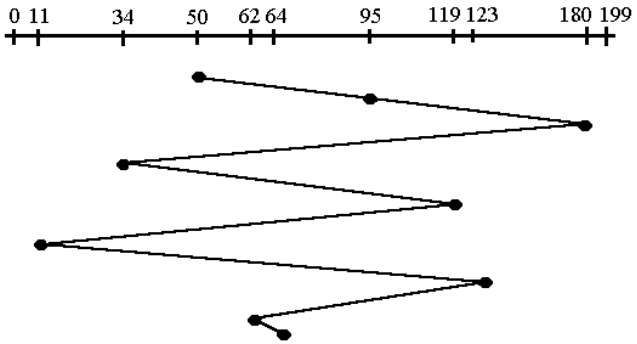
- Zur Erinnerung: Layout Datenträger
  - Layout
    - Datenträger (meist Platte) wird in Partitionen eingeteilt



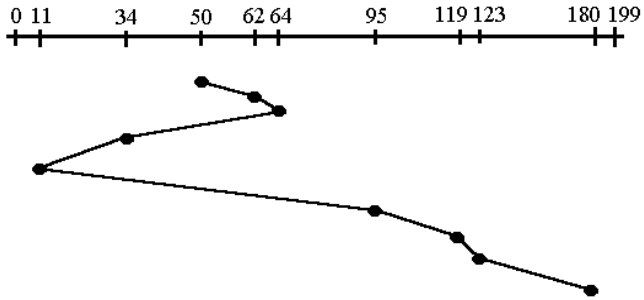
- Boot-Block: Programm zum Laden des Betriebssystems
- SuperBlock: Schlüsselparameter des Dateisystems (Typ, MagischeZahl,Anzahl der Blöcke, …)
- Freispeicherverwaltung: Information über die freien Blöcke im Dateisystem
- I-Nodes: Liste von Datenstrukturen, die alle Informationen über je eine Datei enthält (später!)
- Wurzelverzeichnis: Einstiegspunkt in die Baumstruktur des Dateisystems
- Und schließlich die Daten selbst!

- Implementierung Performanz
  - Performanz ist abhängig von der Größe der zu lesenden/schreibenden Information
  - Schreiben geht langsamer als Lesen, Mechanik der Festplatte muss berücksichtigt werden
  - Lesen 32-Bit-Speicherwort im RAM=ca. 10ns, Festplatte = ca. 100 MB/s
  - **Optimierungsmaßnahmen**
    - **Cache**
      - Zur Reduzierung des Festplattenzugriffs
      - Ansammlung von Blöcken, die logisch zur Platte gehören, jedoch aus Performanzgründen im Speicher gehalten werden
      - Blöcke werden gleichzeitig im Speicher gehalten, um schnell darauf zuzugreifen
    - Algorithmus: Vorausschauendes Lesen von Blöcken
      - Lese i, und falls i+1 nicht im Cache ist, lese auch i+1
      - Nützlich beim sequentiellen Lesen, Schädlich bei wahlfreiem Zugriff

- Implementierung Performanz
  - **Optimierungsmaßnahmen**
    - Reduzierung der Bewegungen des Plattenarms
    - Freier Block soll so gewählt werden, daß er so nah wie möglich zum vorhergehenden Block liegt
    - Integration der Optimierung im Festplatten-Controller
    - Beispiel: 200 Zylinder, Lese Block auf **Zylinder 50**, weitere Anfragen für die Zylinder 95, 180, 34, 119, 11, 123 (FCFS)
    - Shortest Seek First (SSF): 62, 64, 95, 180, 34,119, 11, 123



First Come First Serve (FCFS)



Shortest Seek First (SSF)

- Implementierung Performanz
  - Problem
    - Vergangenheit: Festplattenzugriffszeit blieb nahezu konstant, während alle Komponenten im System schneller wurden
    - Dateisysteme wurden Engpass im System
      - Alternative Ansätze waren notwendig:

        → **Log-Structured File-System (LFS)**

        → **Journaling-File-System (JFS)**

    - Heute: Moderne SSDs mit Flash-Speicher haben keine aufwendige Neupositionierung des Festplattenkopfes

- Implementierung Performanz
  - **LFS (Log-Structured File Systems, Platte als Log)**
    - Mit schnelleren CPUs und immer größerem RAM wächst der Platten-Cache sehr stark
    - Erheblicher Teil aller **Lesezugriffe** geht direkt auf den Platten-Cache, ohne Festplattenzugriffe
    - In Zukunft: Hauptanteil Plattenzugriffe sind Schreibzugriffe
    - Vorausschauendes Lesen bringt keinen Performanzgewinn
    - Bei den meisten Dateiystemen erfolgen Schreibzugriffe in sehr kleinen Stückchen → Kleine Schreibzugriffe sind höchst ineffizient
      (Erzeugung einer neuen Datei unter Unix: Um diese Datei in Unix zu schreiben, müssen der I-Node für das Verzeichnis, der Verzeichnisblock, der I-Node für die Datei und die Datei selbst geschrieben werden)
    - Diese Schreibvorgänge können zwar verzögert werden, aber das Dateisystem kann ernsthafte Konsistenzprobleme bekommen, fall das System abstürzt, bevor alle Schreibzugriffe ausgeführt werden
      → I-Nodes werden sofort geschrieben.

- Implementierung Performanz
  - **LFS (Log-Structured File Systems, Platte als Log)**
    - Grundlegende Idee von LFS ist die Strukturierung der gesamten Platte als ein großes Log.
    - LFS sammelt in regelmäßigen Abständen und bei speziellem Anlass im Speicher gepufferte Schreibaufträge als zusammenhängendes Segment und schreibt sie als zusammenhängendes Segment an das Ende eines Logs auf die Platte.
    - Ein einzelnes Segment kann in beliebiger Reihenfolge I-Nodes, Verzeichnisblöcke und Datenblöcke enthalten. Am Anfang jedes Segments steht eine Segmentzusammenfassung, die Auskunft über den Inhalt des Segments gibt. Auffinden eines I-Nodes viel schwieriger, da nicht mehr aus der I-Nodenummer berechnetw erden kann.
    - Einführung einer I-Node-Map, die über die I-Node-Nummer indiziert wird
      - Diese wird komplizierter und muss ständig aktualisiert werden
    - Beispiel: Öffnen der Datei
      - Suchen in der I-Node-Map nach der I-Node der Datei
      - Diese Blöcke werden in einem Segment irgendwo im Log liegen
      - Adressen der Blöcke können bestimmt werden

- Implementierung Performanz
  - **LFS (Log-Structured File Systems, Platte als Log)**
    - Segmente enthalten auch alte Blöcke
      - Betriebssystem „überschreibt“
    - Gefahr, daß die Log-Datei zu groß wird
      - Regelmäßig startende Cleaner-Thread gibt I-Nodes und Blöcke wieder frei
    - LFS ist schneller bei Schreibzugriffen mit kleinen Blöcken (10x)
    - Nicht weit verbreitet, da inkompatibel zu existierende Dateisystemen

- Implementierung Performanz
  - **JFS (Journaling File System, NTFS, ext3, ReiserFS)**
    - Verbindet Aspekte von Log-basierten Dateisystemen mit existierenden Implementierungen

    - Ablauf:
      - Geplante Operationen werden in einer Datei geloggt
        → Gefahr, daß die Log-Datei zu groß wird
      - Dann werden die Operationen durchgeführt
      - Wenn erfolgreich, dann werden die Operationen aus dem Log gelöscht

    - Bei Systemabsturz: zuerst Abarbeitung der noch im Log stehenden Aufgaben beim Neustart
      - Struktur des Dateisystems überlebt Systemabstürze

- Implementierung Gemeinsame genutzte Dateien
  - **Links**
    - An einer Datei arbeiten mehrere Benutzer gleichzeitig
      - Datei kommt gleichzeitig in Verzeichnissen verschiedener Benutzer vor
        → Verwendung von Links
        → eher ein DAG (Directed Acyclic Graph) als ein Baum

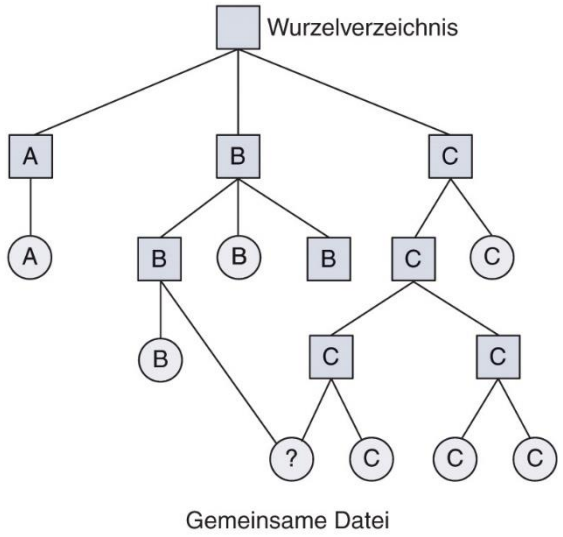


Abbildung 4.16: Ein Dateisystem mit einer gemeinsam benutzten Datei

- Implementierung Gemeinsame genutzte Dateien
  - **Links**
    - Problem:
      - Sind die Adressen im Verzeichnis gespeichert, müssen Änderungen in allen Verzeichnissen durchgeführt werden, nicht nur in einem Zweig
      - Lösung 1: Plattenblöcke nicht im Verzeichnis sondern Attribute in einer Metadaten-Struktur (wie I-Nodes)
      - Lösung 2: Symbolischer **Link** (auf ursprünglichen Pfad der Datei) z.B. B auf Eigentümer C

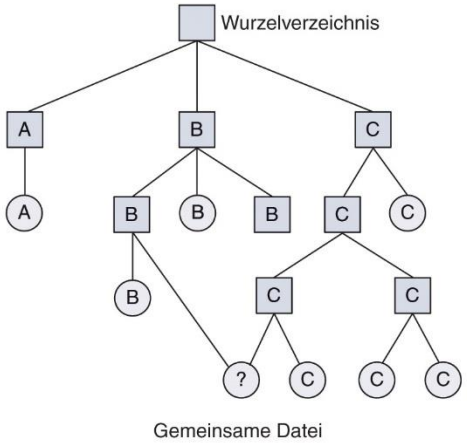


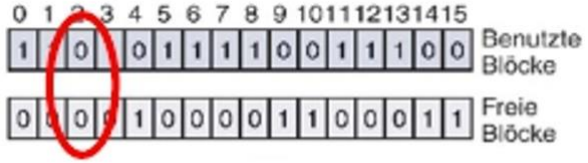
Abbildung 4.16: Ein Dateisystem mit einer gemeinsam benutzten Datei

Vorsicht bei der Verwendung von gemeinsamen genutzten Dateien

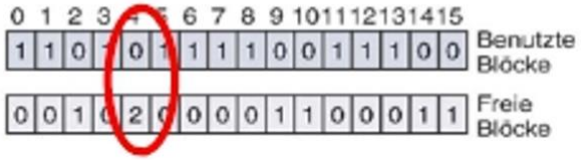
Dateiverwaltung

- Implementierung Konsistenz
  - Dateisysteme lesen Blöcke, modifizieren und schreiben diese
  - Bevor alle veränderten Blöcke zurückgeschrieben wurden → Systemabsturz → Inkonsistenz
  - Auf inkonsistentem Dateisystem werden Konsistenzprüfungen durchgeführt (Hilfsprogramm UNIX = file system check=fsck, Windows = system file ckecker=sfc oder Scandisk)
  - Konsistenz-Prüfungen
    - Findet auf Ebene der **Blöcke** statt und erstellt
      - **Tabelle 1** für das Vorkommen eines Blocks in einer Datei
      - **Tabelle 2** für das Vorkommen eines Blocks in der Freibereichsleiste
      - Schleife über alle I-Nodes und den Freibereich
      - Alles OK: Jeder Block ist **nur einmal** entweder in Tabelle 1 oder 2

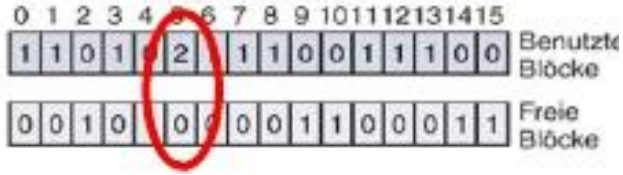
- Implementierung Konsistenz
  - Problemfälle nach Absturz:
    1. **Fehlender Block (Block 2)**
       - Kein Schaden
       - Reduziert die Festplattenkapazität
       - Korrektur: Block zur Freibereichs-liste hinzufügen



    2. **Doppelter Block im Freibereich**
       - Treten nur bei der „Liste“ auf
         → Bei Bitmap nicht möglich
       - Gefahr: doppelt vorhandener Block in der Freibereichsliste (Block 4)
       - Korrektur: Neuaufbau der Freibereichsliste



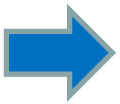
- Implementierung Konsistenz
  - Problemfälle:
    3. **Doppelter Datenblock (Block 5)**
       - Datenblock ist Teil von 2 Dateien
       - Konsequenzen:
         - Beim Löschen einer Datei erscheint der Block als benutzte und als freie Datei
         - Beim Löschen beider Dateien taucht der Block 2 mal in der Freibereichsliste auf



       - Lösung zur Rettung einer Datei:
         - Kopiere den Block auf einen freien Block
         - Eingliedern des neuen Blocks in eine der Dateien
         - Damit bleibt mindestens die Information einer Datei erhalten

Dateisysteme: ISO-9660

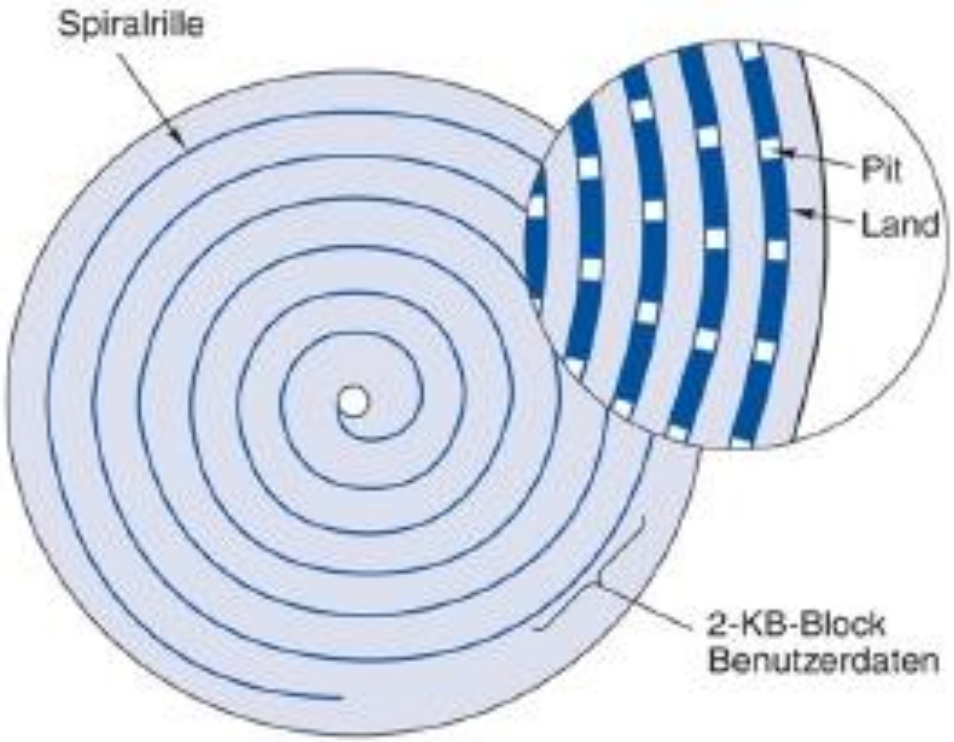
- Internationaler Standard für CD-ROM-Dateisysteme
- Eine lange Spur sequentieller Daten
  - Unterteilt in 3252 Byte lange logische Blöcke
    - Information über Präambeln, Fehlerkorrektor, …
    - 2048 Byte für Daten
  - Persistenter Datenspeicher
    - Nur Lesezugriffe möglich!
  - Verwaltung der Daten mit Hilfe des Standards, der die Struktur festlegt



Jede CD soll auf jedem Betriebssystem lesbar sein

Dateisysteme: ISO-9660

- Eine lange Spur sequentieller Daten
- Kontinuierliche Spirale, Bits stehen in einer linearen Sequenz



Aufgabe/Frage

In ISO-9660 besteht das Datumfeld aus:

- Year
- Month
- Day
- h
- m
- s
- Timezone

Die Zählung beginnt bei 1900.

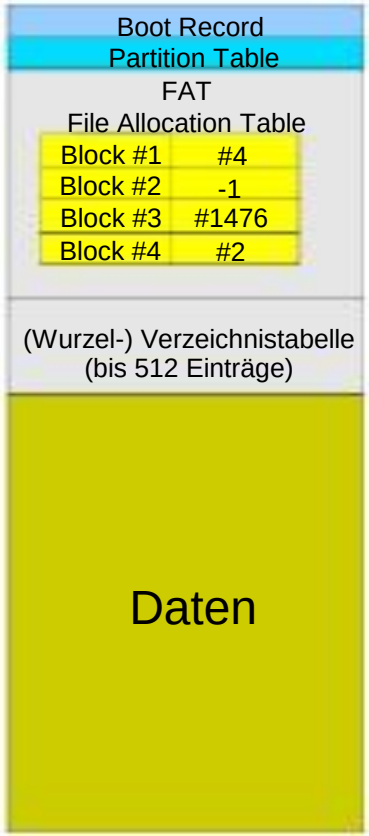
Frage:
Wann hat die CD-ROM ein Problem?

Bedingung:

→ 3 min

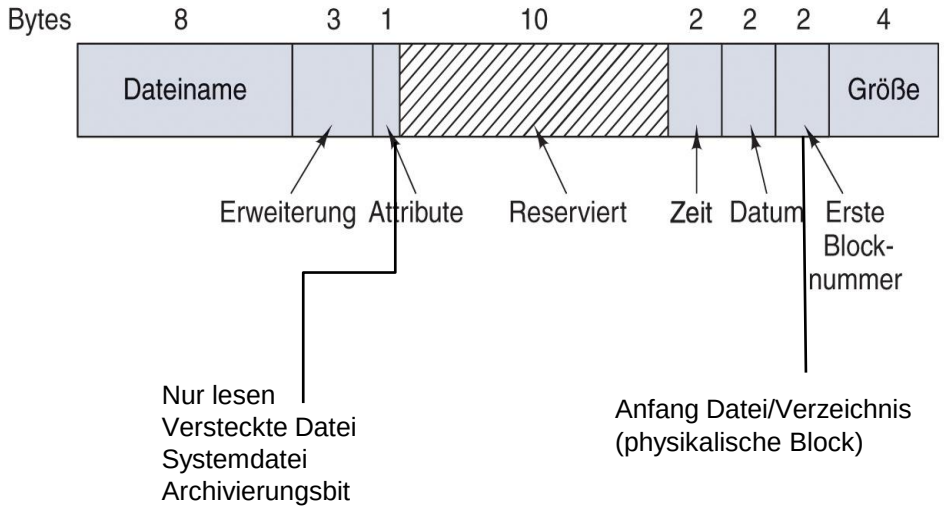
Dateisysteme: ISO-9660

- MS-DOS Dateisystems (FAT)



Informationen zum Hochfahren des Systems
Informationen zur Partition

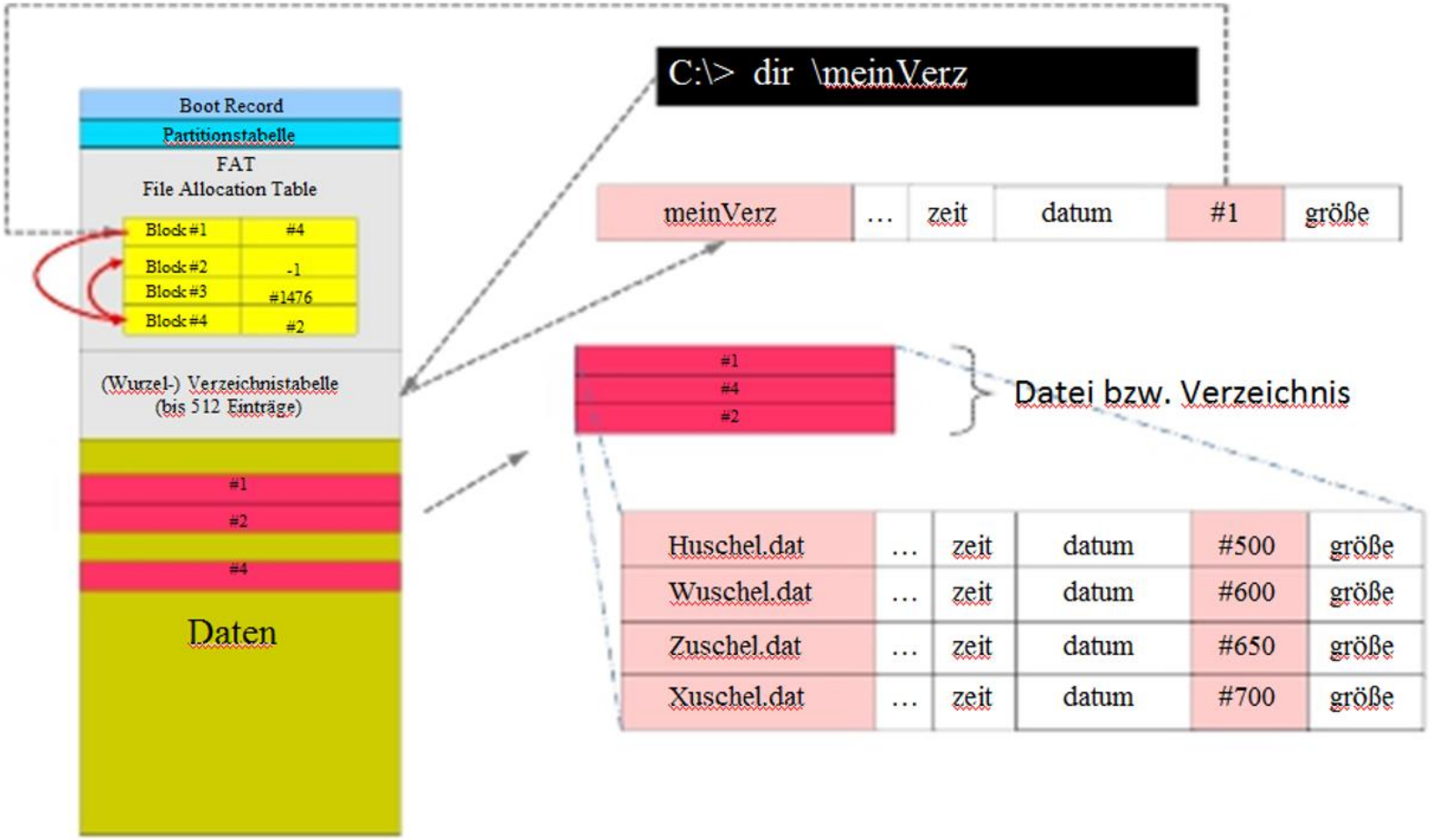
Verkettete Zeiger auf physikalischen Speicherplatz (persistente Kopie)



Jahr ist 7-Bit-Zahl, Zeitrechnung ab 1980,
→ größtes darstellbare Jahr ist 2107

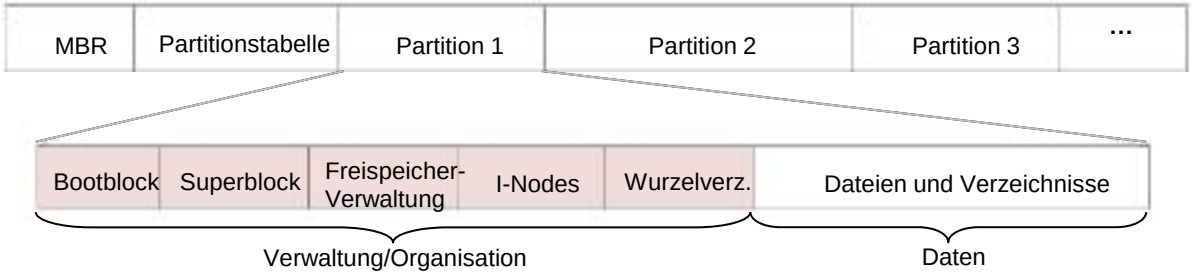
Dateisysteme: ISO-9660

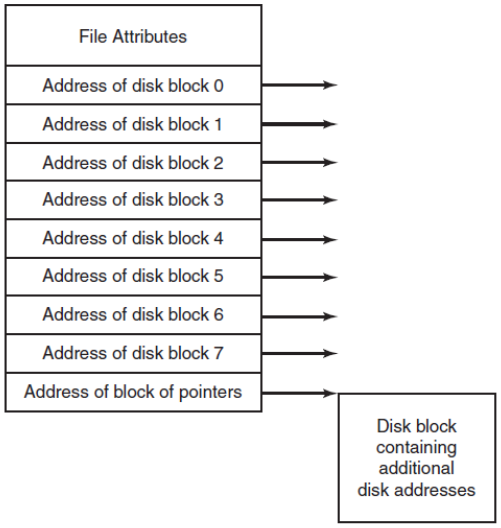
- MS-DOS Dateisystems (FAT)



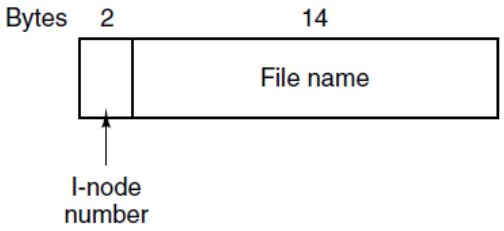
Dateisysteme: Unix-V7

- Verwendung von I-Nodes





- Datei-Eintrag in UNIX-V7
- Wurzelverzeichnis
  - Hier stehen die I-Nodes für die Verzeichnisse /root, /Bin, /home,...



Dateisysteme: Unix-V7

- Verwendung von I-Nodes
- Wie wird nach dem Pfadnamen /usr/ast/mbox gesucht?

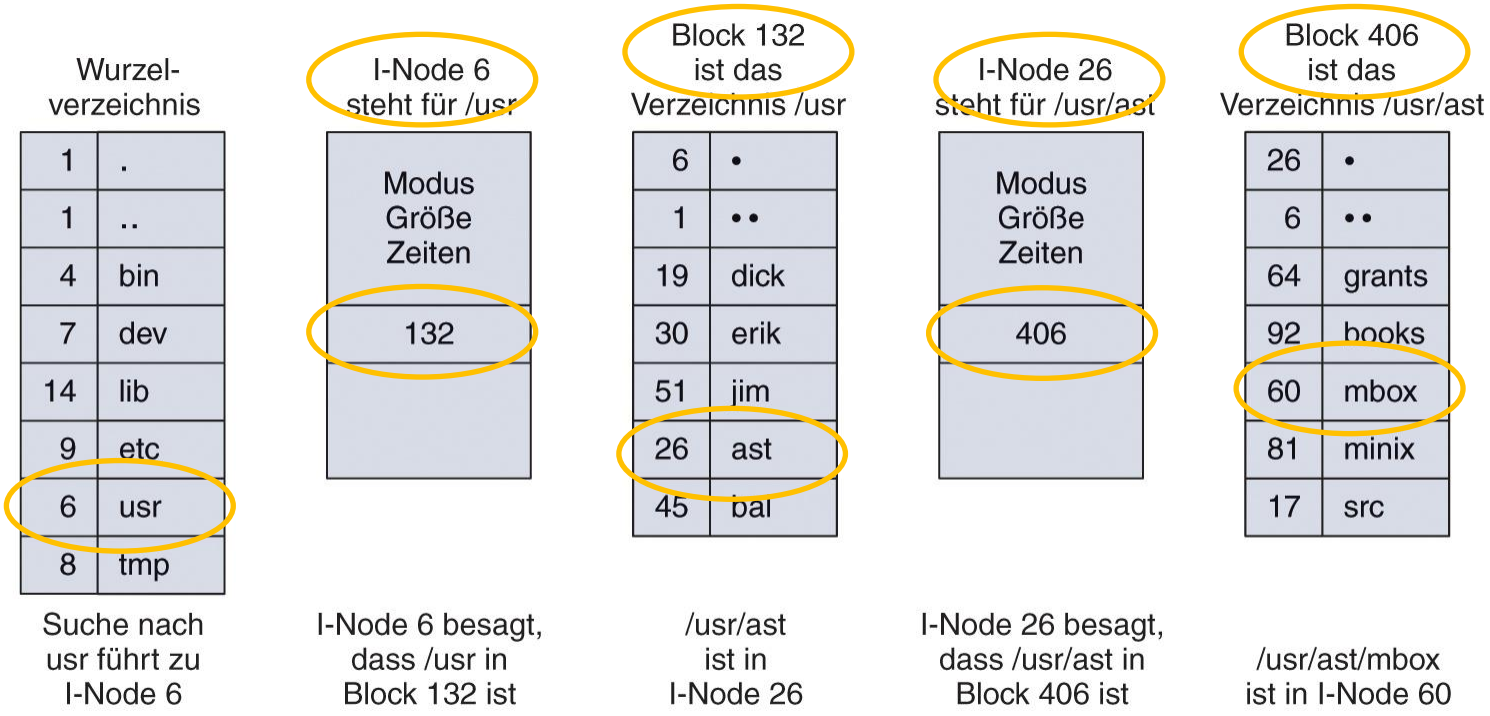


Abbildung 4.35: Schritte, um */usr/ast/mbox* zu finden

Annahme: Wurzelverzeichnis ist Block 1

Dateisysteme: Unix-V7

- Große Dateien mit I-Node

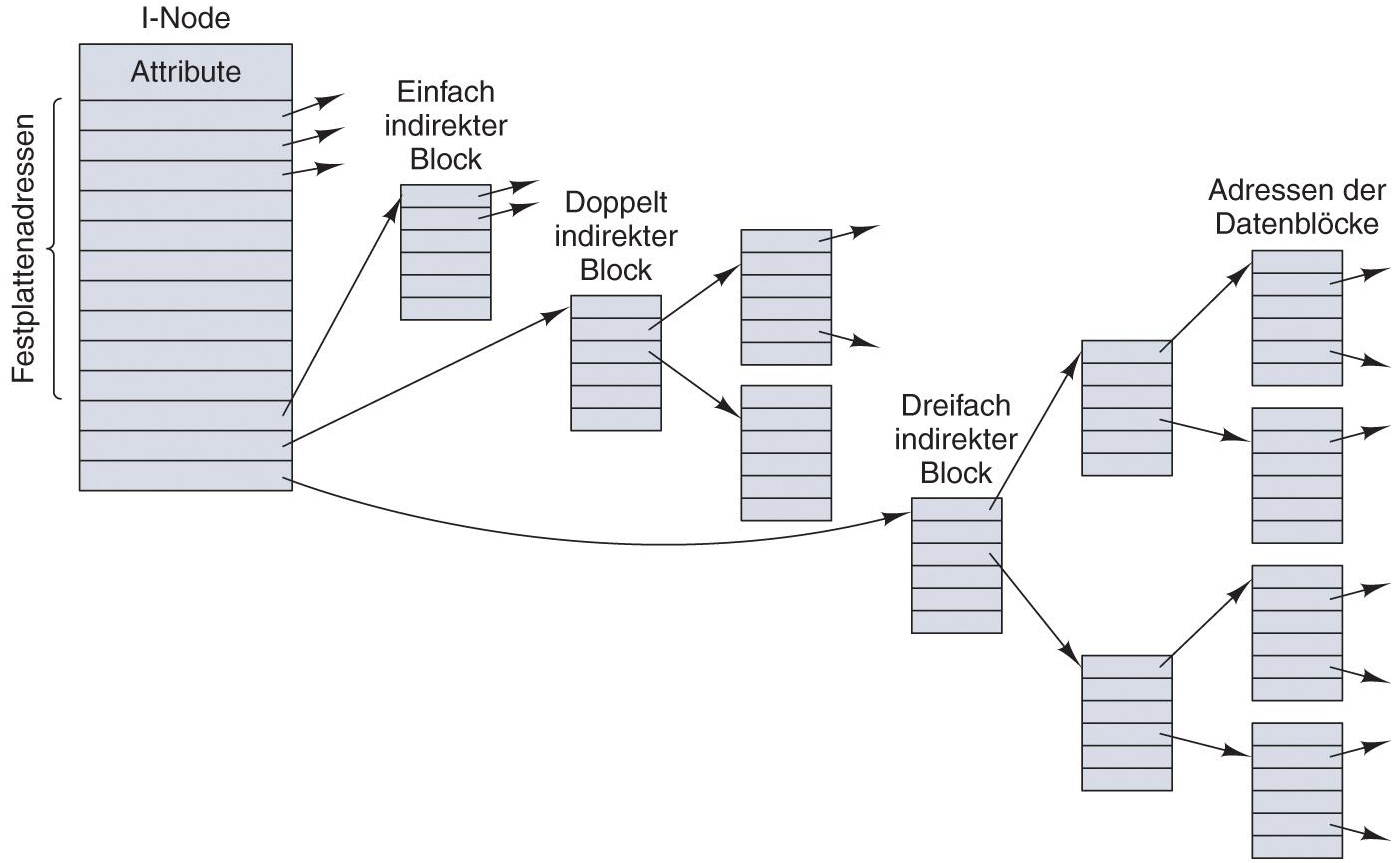


Abbildung 4.34: I-Node unter UNIX

Zusammenfassung

- Dateien als Abstraktion zur persistenten Speicherung von Informationen

- Verzeichnisse als Abstraktion zur Organisation von Dateien
  - Beide werden durch Betriebssystem verwaltet

- Layout von Datenträgern (Festplatten)
- Implementierung von Dateien und Verzeichnissen

• Zusammenhängende Belegung	ISO9660
• Belegung durch verkettete Listen	FAT
• I-Nodes	UNIX-V7
• Verzeichniseintrag	

- Festplatten-Speicherverwaltung
  - Blockgröße und Verwaltung freier Blöcke

- Konsistenz von Dateisystemen