Kapitel 5
Externe Geräte & Dateisysteme

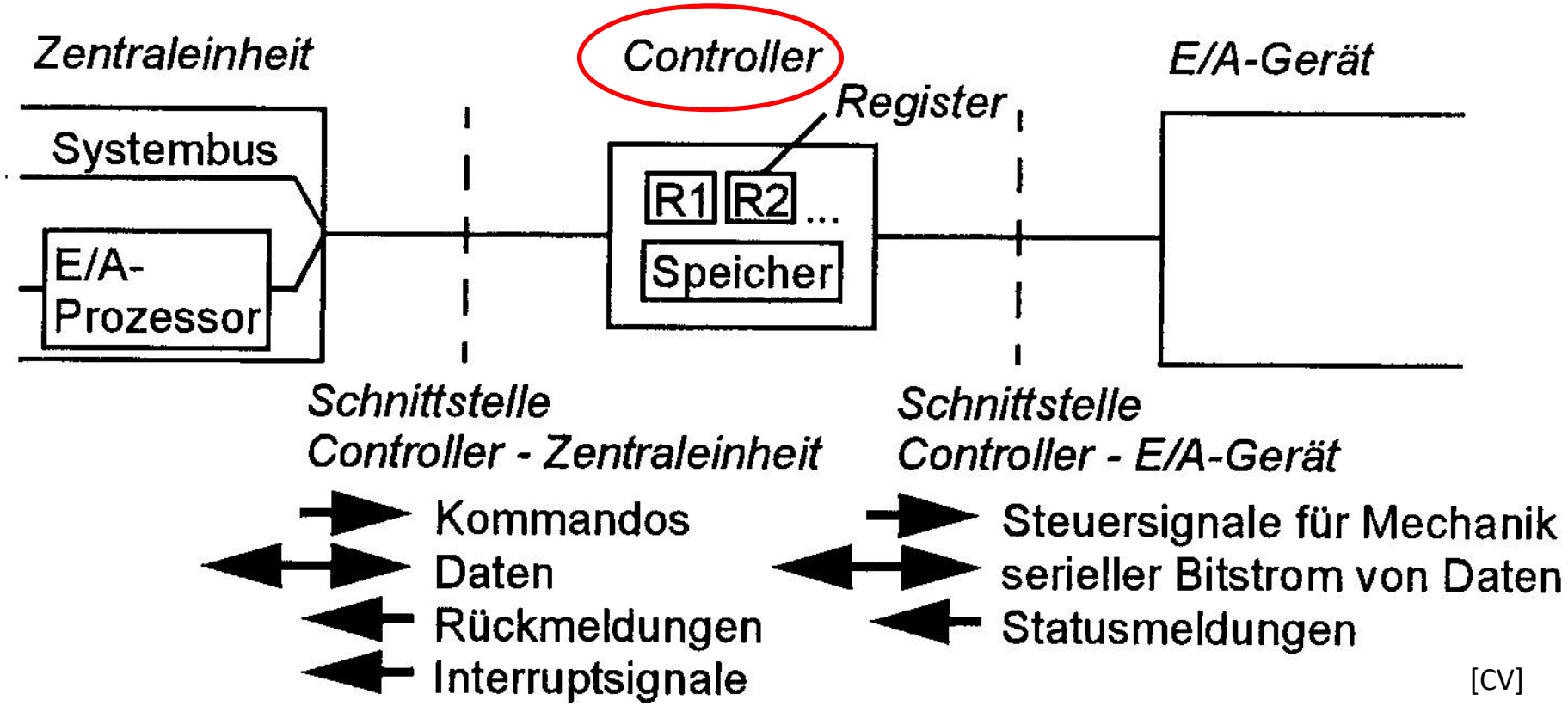
Externe Geräte

- Klassen von Geräten
  - Speicher
    - Magnet-Festplatte, SSD, Magnet-Band, DVD-Brenner, …
  - Eingabegeräte
    - Tastatur, Maus, Joystick, Sensor, …
  - Ausgabegeräte
    - Bildschirm, Drucker, Roboterarm, …
- Alle diese Geräte enthalten
  - Mechanische Komponenten
  - Elektronische Komponenten („Controller“)

Controller

- … befinden sich
  - entweder auf dem Mainboard
  - oder auf einer eigenen Steckkarte in einem Slot
  - oder im Gerät selbst
- … besitzen (mindestens) zwei Schnittstellen
  - eine Schnittstelle zur Kommunikation mit dem Prozessor / Hauptspeicher
  - eine Schnittstelle zur Steuerung des Gerätes
- … verfügen über
  - einen internen Speicher (Puffer)
  - spezielle Register

Position und Aufgaben eines Controllers



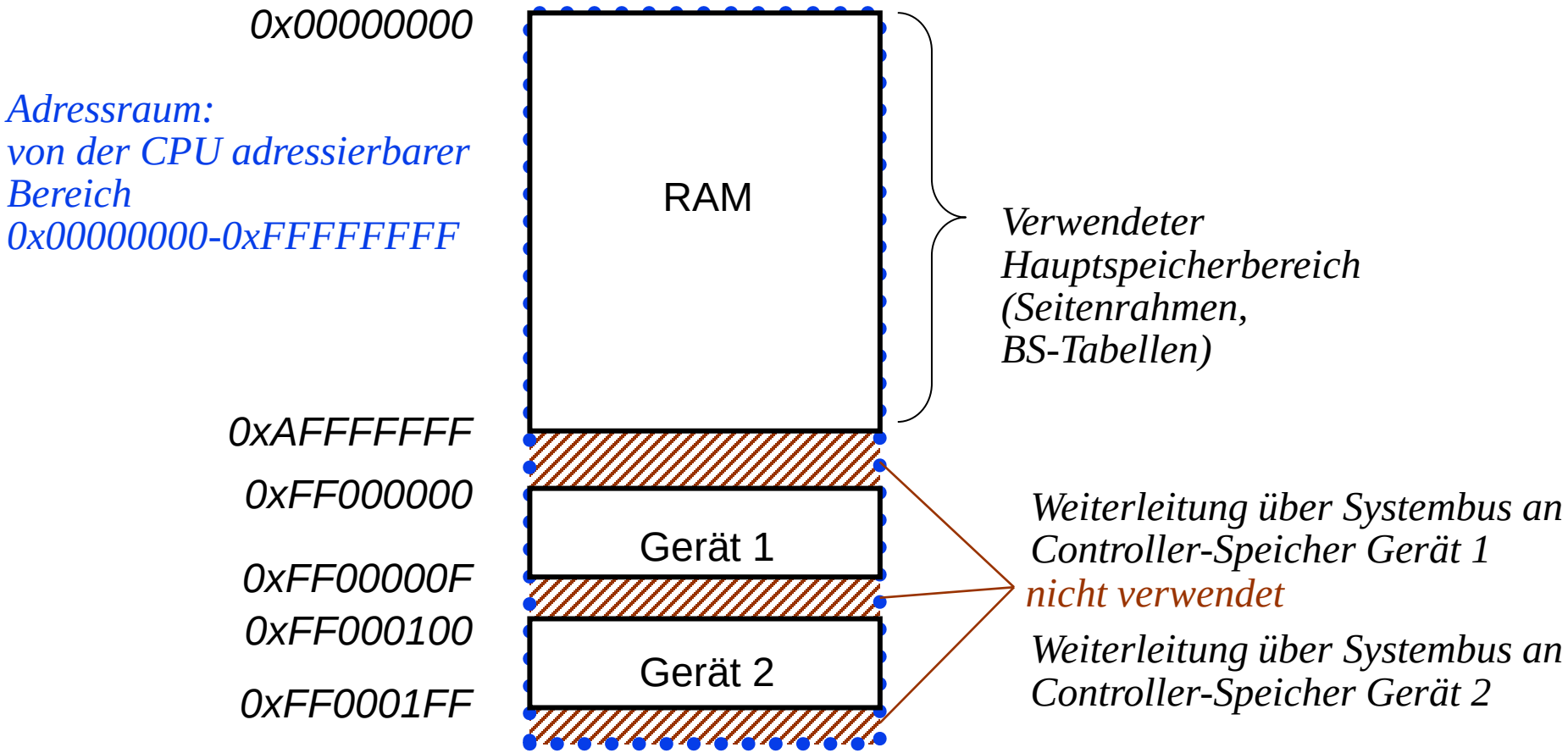
[CV]

Kommunikation zwischen Betriebssystem und Controller

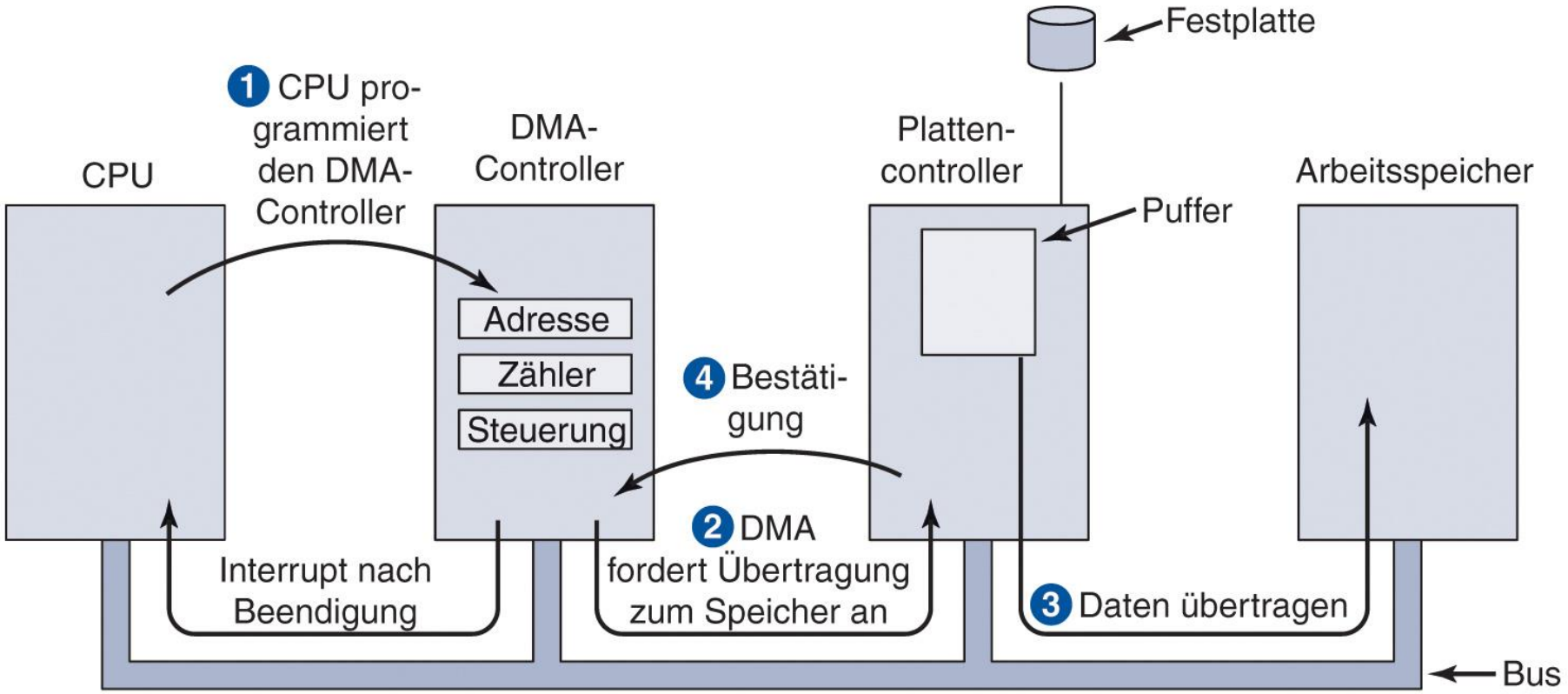
- Signalisierungsmethoden:
  - Registerzugriff und aktives Warten („Programmed I/O“)
  - Interrupts
- Datenaustausch:
  a) Das Betriebssystem greift auf Controller-Register über spezielle Befehle zu („I/O-Ports“)
  → *mehrere Adressräume mit unterschiedlicher Syntax*
  b) Ein Controller erhält für seinen Speicher vom Betriebssystem reservierte Hauptspeicheradressen („Memory-Mapped-I/O“)
  → *ein einheitlicher Adressraum für alle Speicherzugriffe*
  c) Direct Memory Access („DMA“) [Transfer großer Datenmengen]:
    - Das Betriebssystem übergibt Anfangsadresse und Größe des Hauptspeicherbereichs an den DMA-Controller
    - Der DMA-Controller steuert den Datentransfer zwischen Gerätecontroller und Hauptspeicher
    - Der DMA-Controller informiert das Betriebssystem mittels Interrupt über das Ende der Operation

Memory Mapped I/O

Der Zugriff auf Controller-Register eines Memory-Mapped-Gerätes unterscheidet sich vom Zugriff auf eine Speicherstelle des Hauptspeichers lediglich durch den verwendeten Adressbereich



Direct Memory Access („DMA“)

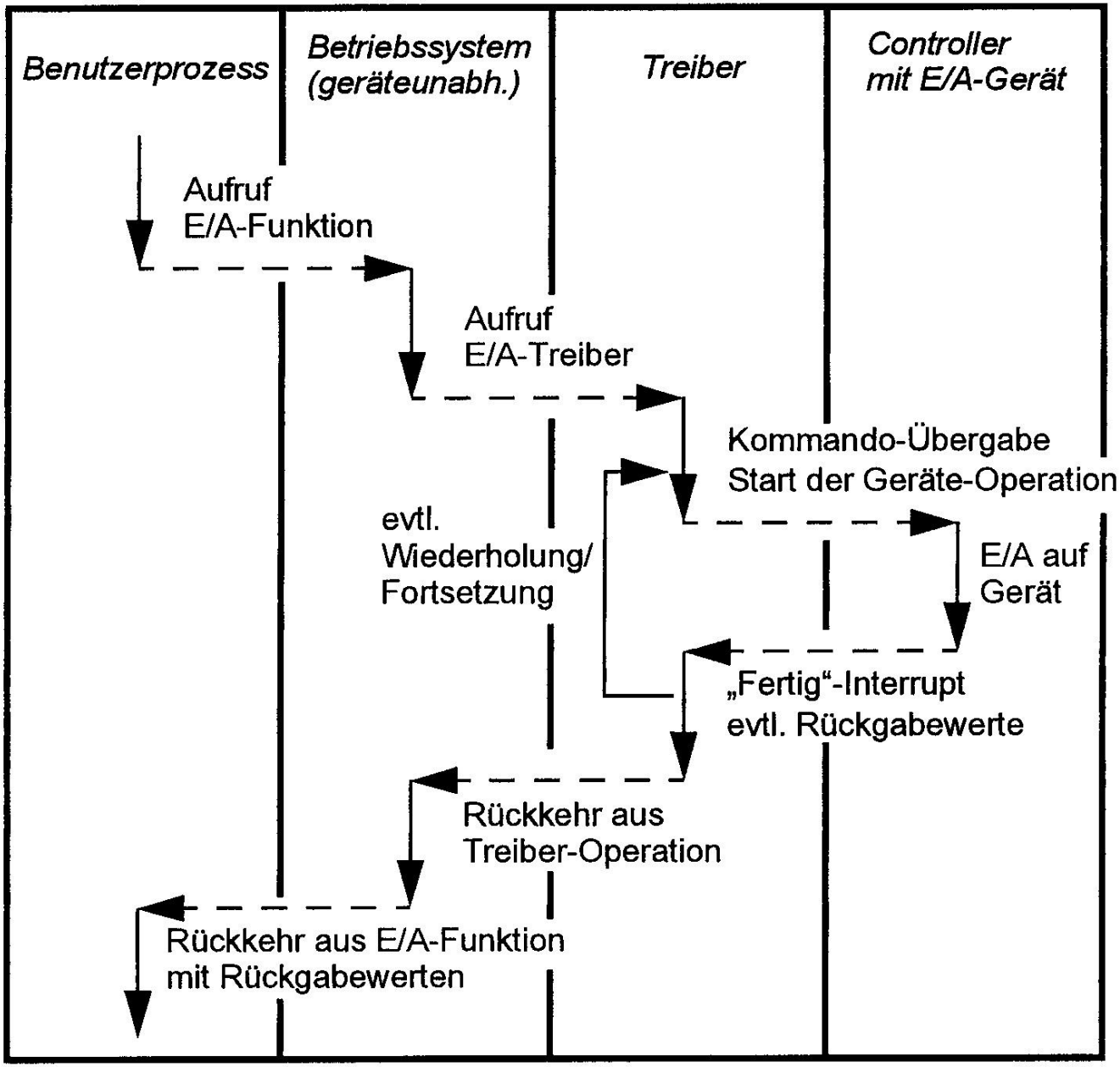


[AT]

Treiber („Device Driver“)

- Eigenschaften
  - herstellerspezifische Betriebssystemerweiterung (!)
  - steuert den Controller
- Aufgaben
  - Initialisierung des E/A-Geräts beim Systemstart
  - Vorbereitung einer E/A-Operation und Aktivierung des Geräts für den Datentransfer
  - Reaktion auf Geräte-Interrupts
  - Fehlerbehandlung
- Implementierung als
  - eigener Prozess und/oder
  - betriebssystem-interne Funktion (Aufruf aus Bibliothek)
- Einbindung in das Betriebssystem durch
  - Integration in den Kernel (Unix)
  - Laden aus Datei bei Systemstart (Windows, Unix)
  - Laden zur Laufzeit (z.B. ➔ USB-Unterstützung)

Ablauf einer E/A-Operation

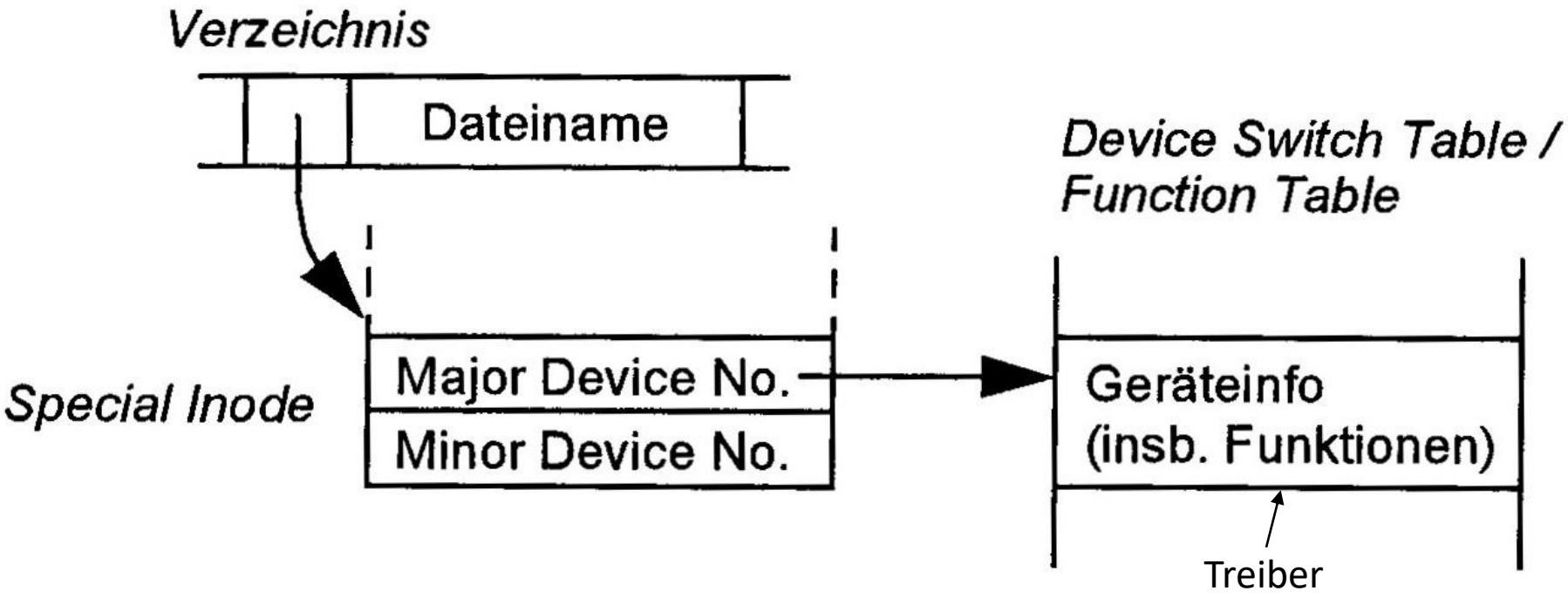


[CV]

UNIX – E/A-Konzept

- Alle Geräte sind in das Dateisystem integriert als „special files“ (Verzeichnis /dev/..)
- Die meisten Dateifunktionen können auch für Geräte verwendet werden (open, close, read, write)
- Zusätzliche Funktion: `ioctl()` zur spezifischen Ansteuerung eines Geräts
- Zugriff über spezielle I-Nodes (Dateireferenzen) (→ Aufruf der Treiber)
  - Major Device No.: Information über eine Geräte-Klasse
  - Minor Device No.: Spezielle Geräte-Instanz

Geräte-I-Nodes in UNIX



[CV]

Windows - E/A-Konzept

- Erzeugen von Treiber-Objekten durch den E/A-Manager (unterstützt vom Plug-and-Play-Manager) zur Laufzeit
- Kommunikation zwischen Treiber und Controller läuft über die HAL (Hardware Abstraktion Layer)
- Aufgaben der HAL:
  - Adressierung der Geräte-Hardware (unabhängig vom Bus): Abbildung logische Geräteadresse → physikalische Geräteadresse
  - Zugriff auf Geräteregister
  - BIOS-Schnittstelle
  - …
- Direktzugriffe von Treibern bedingt möglich durch DirectX-Prozeduren

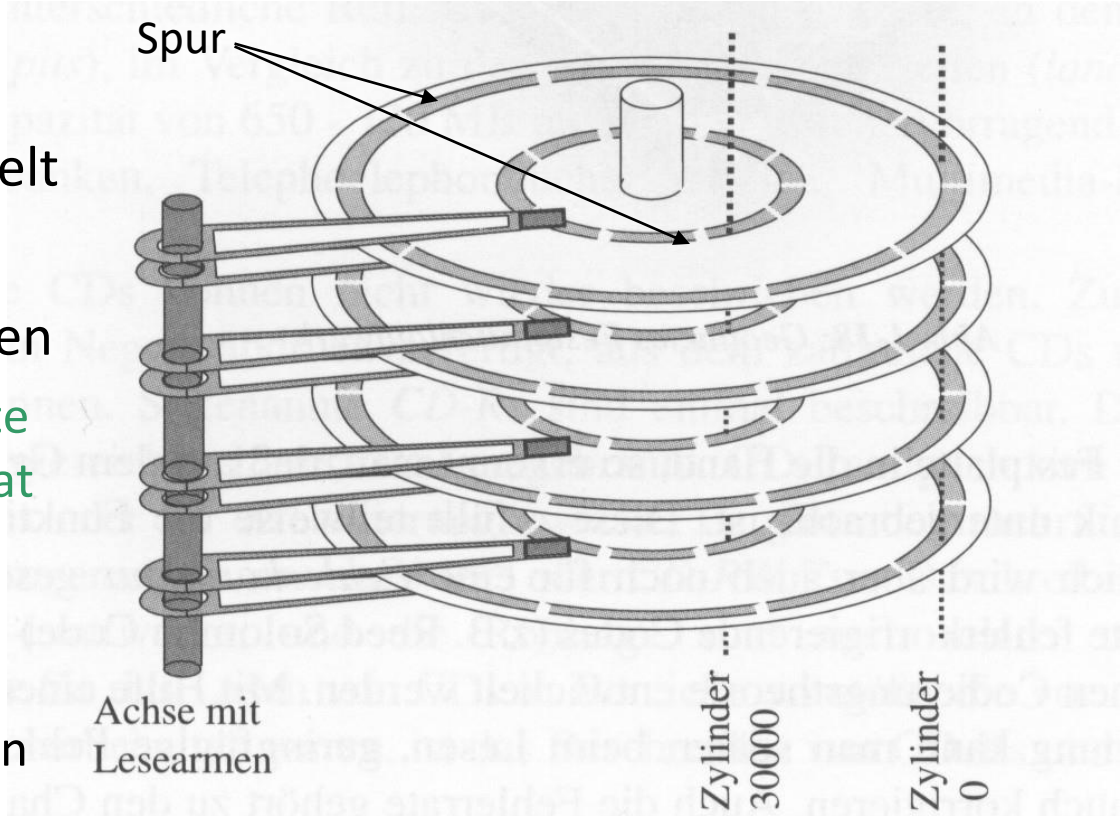
Windows – Treiber

- Windows-Driver-Model muss eingehalten werden:
  1. Behandlung von E/A-Anfragen im Standardformat
  2. Verwendung des Objektmodells von Windows
  3. Plug-and-Play-Unterstützung
  4. Power-Management-Unterstützung (wenn möglich)
  5. Konfigurierbarkeit bzgl. Ressourcenverbrauch
  6. Wiedereintrittsfähigkeit (Multiprozessorunterstützung)
  7. Lauffähigkeit unter verschiedenen Windows-Versionen
- Implementierung von Standard-Methoden im Treiber (z.B. **`DriverEntry`**, **`AddDevice`**, Interrupthandler, …)
- Aufteilung des Treibers in mehrere Komponenten wird unterstützt (inkl. Verschachtelung)

Kapitel 5
Externe Geräte & Dateisysteme

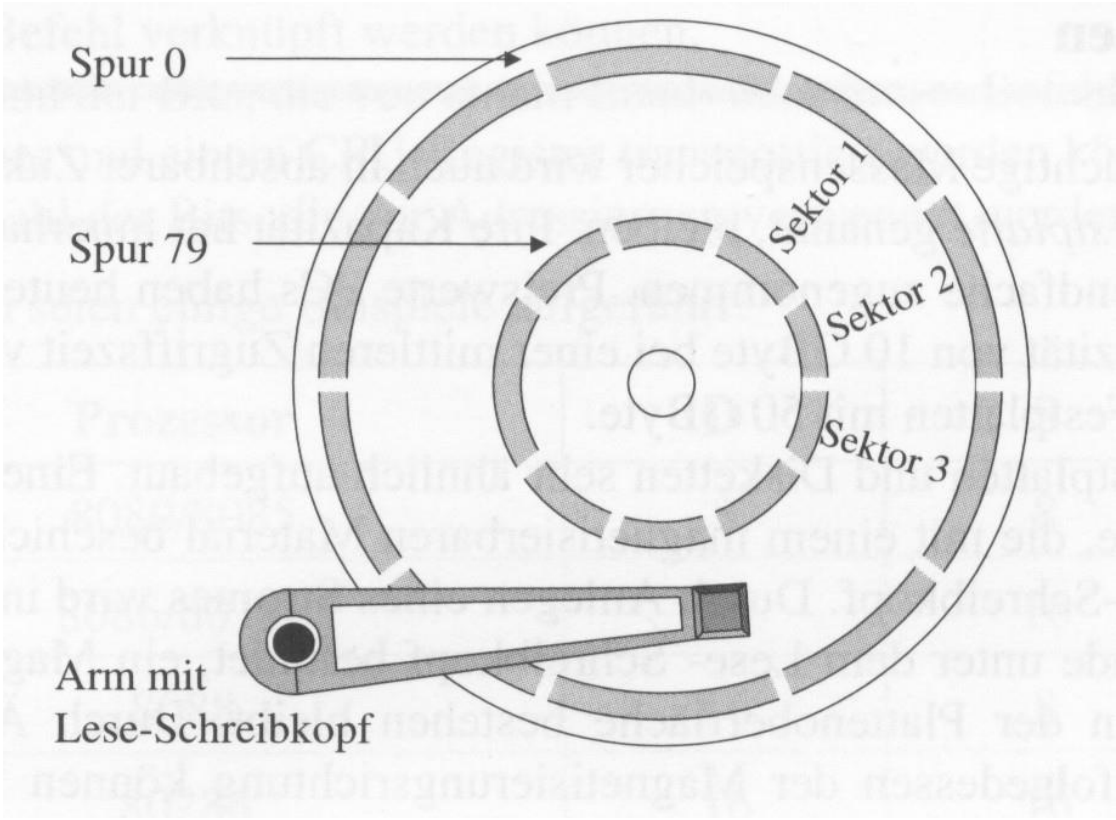
Eigenschaften einer Magnet-Festplatte (1)

- Rotierende Scheiben, magnetisch beschichtet
- Mehrere Scheiben gestapelt
- In Spuren aufgeteilt
- Jede Spur enhält n Sektoren
  - Sektor: klassisch 512 Byte Heute „Advanced Format Drives“ mit Sektorgröße 4096 Byte
  - Die entsprechenden Sektoren im Stapel bilden einen Zylinder



Eigenschaften einer Magnet-Festplatte (2)

- Schreib- / Lesekopf wird durch einen Arm auf eine Spur / einen Sektor positioniert
  - Pro Oberfläche ein Arm mit Schreib-/ Lesekopf
- Der Zugriff ist wahlfrei
- Interne Adressierung:
  - Nr. der Oberfläche
  - Nr. der Spur
  - Nr. des Sektors innerhalb der Spur
- Platten-Controller liefert an das BS linearen Adressraum von Sektornummern (0 – nn)



Solid State Disk (SSD)

- Nachteile von **Magnet-Festplatten (HDD):**
  - Hohe Wartezeit, bis gesuchter Bereich unter Lese-Schreibköpfen ist
  - Hoher Energieverbrauch
  - Geringe Stoßfestigkeit
  - Laufgeräusche
- Alternative: **Halbleiterspeicher (SSD)**
  - Daten (in der Regel) elektronisch **dauerhaft** gespeichert in sogenannten **Flash-Speichern** *(„Floating Gate“-Transistoren)*
  - Anzahl Schreibzyklen pro Speicherzelle ist beschränkt *(Lebensdauer)*
  - Bieten u.a. dieselben Schnittstellen wie HDD *(z.B. SATA)*
    → *können Magnet-Festplatten ersetzen*
  - Größenordnungsmäßig schneller als HDD *(Faktor 100 - 1000)*!

Festplatten - Cache

- An zwei Stellen wird i.d.R. ein Puffer bzw. Cache benutzt:
  - Die Platten-Controller besitzen einen Puffer für die gelesenen / zu schreibenden Blöcke (Sektoren)

  ➔ Achtung bei Stromausfall!

  - Im Hauptspeicher wird vom Betriebssystem ein Puffer (Cache) für Plattenblöcke reserviert
    - Es werden Kopien von den Plattenblöcken gespeichert, auf die zuletzt zugegriffen wurde

Kapitel 5
Externe Geräte & Dateisysteme

1. Externe Geräte
   a) Controller und Driver
   b) Festplattenorganisation
2. Dateisysteme
   a) Dateien und Verzeichnisse
   b) Implementierung von Dateisystemen
   c) Dateisystem-Beispiele
3. Zuverlässigkeit von Dateisystemen
   a) Konsistenz von Dateisystemen
   b) Zuverlässiger Betrieb von Dateisystemen

Dateisysteme

- Dateien („Files“) sind persistente Daten-“Behälter“, die
  - als Datenquelle einer Applikation genutzt werden
  - als langfristiger Datenspeicher einer Applikation genutzt werden
- Zur Verwaltung werden Dateien zu Dateisystemen zusammengefasst.
- Dateisysteme werden auf externen Datenträgern gespeichert (Festplatte, SSD, USB-Stick, CD, DVD,…)

Dateisystemfunktionen

- Bereitstellung von Dateioperationen (Read, Write, ..)
- Verwaltung von Verzeichnissen („Directories“), um Ort und Eigenschaften aller Dateien zu speichern
- Überprüfung von Zugriffsrechten
- Abbildung von logischen Dateizugriffen (Dateioperationen) auf physikalische Einheiten (Blöcke)
- Verwaltung von Plattenblöcken
  → externe Datenträger

Nutzersicht: Operationen auf Dateien

create	Die Datei wird ohne Daten erstellt; einige Attribute sind gesetzt
remove	Löschen der Datei und Freigeben des Speicherplatzes
open	Vor der Nutzung einer Datei muss ein Prozess den Zugriff initialisieren
close	Schließen der Datei um Zugriff freizugeben
read	Daten werden aus der Datei gelesen
write	Daten werden in die Datei geschrieben
append	Beschränkte Form des Schreibens – nur an das Ende der Datei
seek	Der Positionszeiger wird auf eine bestimmte neue Position versetzt
get_Attributes	Lesen der Attribute einer Datei
set_Attributes	Setzen der Attribute, welche vom Nutzer geändert werden dürfen
rename	Umbenennen einer Datei
link	Symbolischen oder direkten Verweis („Link“) auf die Datei erzeugen

Verzeichnisse

Ein **Verzeichnis** ("Directory")

- … leistet die Abbildung zwischen den Dateinamen und den entsprechenden Dateien des Verzeichnisses
- … ist selbst eine **Datei** (mit einer speziellen Funktion)
- … enthält eine Liste mit allen Dateinamen + zugehörigen Verwaltungsinformationen aller Dateien des Verzeichnisses, z.B.
  - Dateiname, Speicherort, Attribute *(Typ, Größe, Speicherdatum, Status),* Besitz- / Zugriffsrechte, …
- … ist bei den meisten Dateisystemen Teil einer hierarchischen (Baum-) Struktur:
  - Das Hauptverzeichnis bildet den Hauptstamm (auch Wurzelverzeichnis oder „root“ genannt)
  - Die Astverzweigungen sind die Verzeichnisse
  - Die Blätter sind die Dateien

Nutzersicht:
Spezielle Operationen auf Verzeichnissen

mkdir	Das Verzeichnis wird angelegt; es ist leer bis auf die Einträge „.“ und „..“
rmdir	Das Verzeichnis wird gelöscht (muss leer sein)
opendir	Das Verzeichnis wird geöffnet um u. U. anschließend gelesen zu werden
closedir	Schließen des Verzeichnisses um Zugriff freizugeben
readdir	Übergibt den nächsten Eintrag in einem geöffneten Verzeichnis

Ansonsten können die Datei-Operationen verwendet werden
(Verzeichnis = spezielle Datei)!

Kapitel 5
Externe Geräte & Dateisysteme

1. Externe Geräte
   a) Controller und Driver
   b) Festplattenorganisation
2. Dateisysteme
   a) Dateien und Verzeichnisse
   b) Implementierung von Dateisystemen
   c) Dateisystem-Beispiele
3. Zuverlässigkeit von Dateisystemen
   a) Konsistenz von Dateisystemen
   b) Zuverlässiger Betrieb von Dateisystemen

Fragestellung

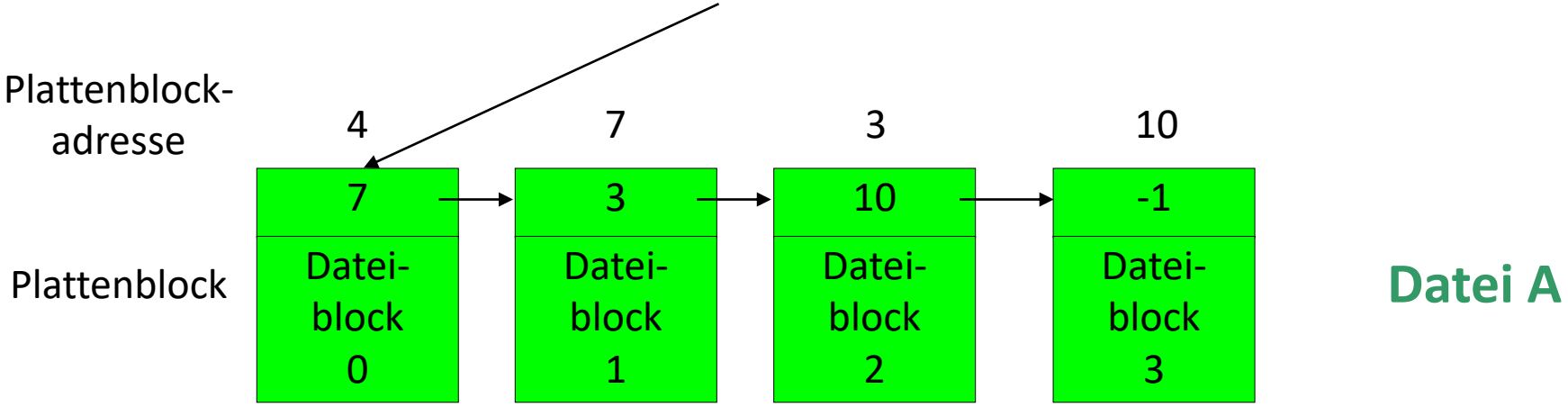
- Benutzersicht: Dateien und Verzeichnisse
- Hardwaresicht: externe Datenträger (Festplatte) mit einzeln adressierbaren Blöcken (Sektoren)
- Betriebssystem: Welche Blöcke gehören in welcher Reihenfolge zu welcher Datei (welchem Verzeichnis)?
  ➔ „Belegungsmethoden“

1. Belegungsmethode: Zusammenhängende Speicherung

- Alle Blöcke einer Datei werden in physikalisch zusammenhängenden Speicherbereichen abgelegt
- Der Dateieintrag im Verzeichnis enthält die Startadresse des ersten Blocks und die Anzahl der kontinuierlich belegten Blöcke.
- **Probleme?** (→ Dynamische Hauptspeicherpartitionierung)
  - Lückenverwaltung („Fragmentierung“ des Speichers)
  - Eine genügend große Lücke muss vorhanden sein
  - Die Dateigröße muss vorher bekannt sein!

2. Belegung durch verkettete Listen

- Die Blöcke einer Datei können auf der Platte verstreut sein (nicht zusammenhängend gespeichert)
  → Verkettung der Blöcke ist nötig
- Das erste Wort eines jeden Plattenblocks wird als Zeiger auf den nächsten Block der Datei benutzt (enthält dessen Plattenblockadresse)
- Der Dateieintrag im **Verzeichnis** enthält einen Zeiger auf den ersten Block der Datei: ("Datei A", 4), der letzte Block hat den Zeiger -1

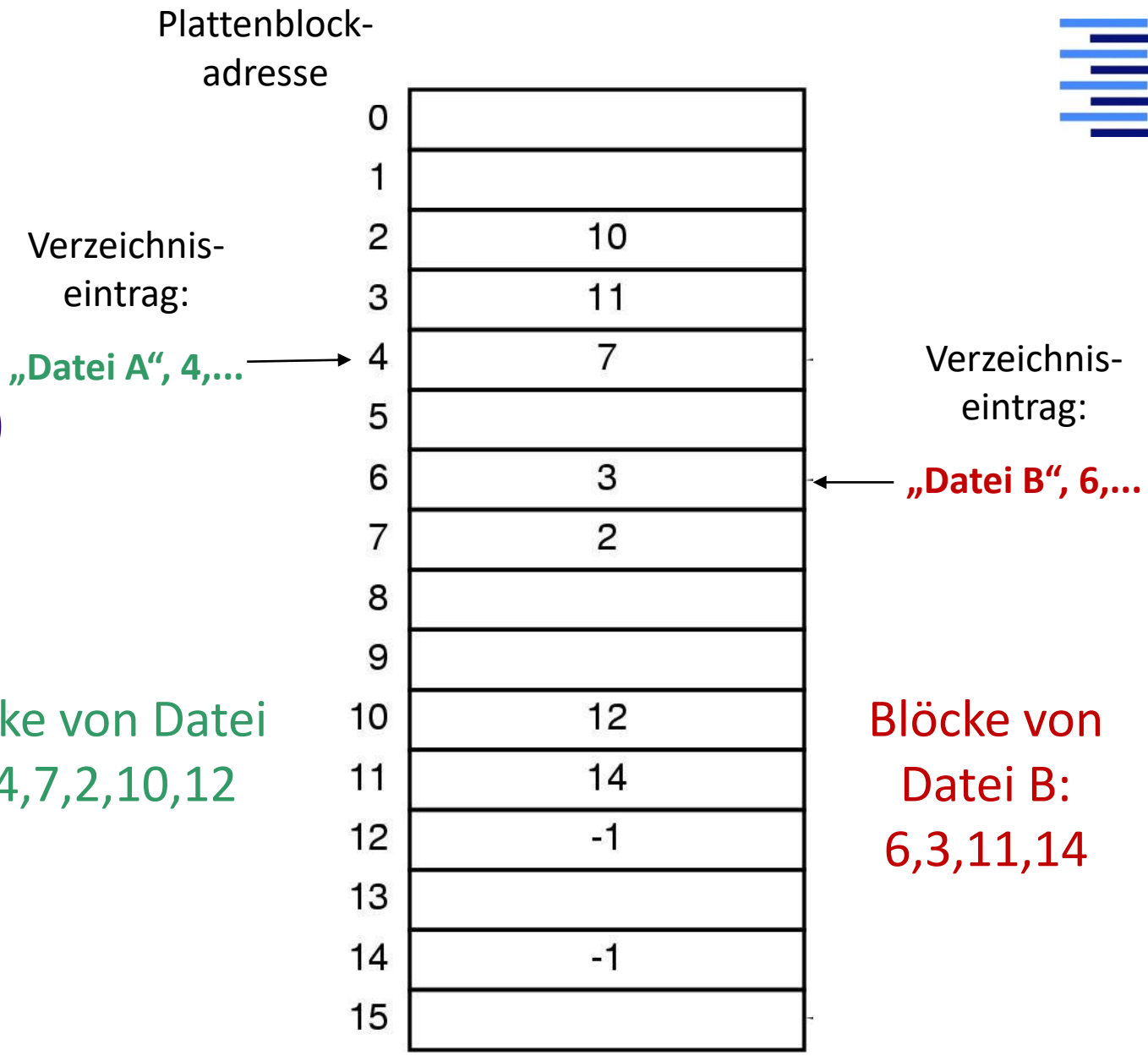


Probleme?

3. Belegung durch verkettete Listen mit globaler Zeigertabelle: File Allocation Table (FAT)

- Idee: „Herausziehen“ der Zeiger aus den Plattenblöcken
- Alle Plattenblöcke werden durch eine separate Zeigertabelle im Hauptspeicher beschrieben („File Allocation Table“ = FAT) mit Tabellenindex = Plattenblockadresse
  - Der Verzeichniseintrag für eine Datei enthält den Namen und die erste Plattenblockadresse = Index des ersten Eintrags in der FAT
  - Jeder Eintrag in der FAT ist die Adresse des nächsten Plattenblocks
  - Der FAT-Eintrag für den letzten Block enthält -1 oder EOF (= „End Of File“)
- Weitere Informationen über eine Datei (Änderungsdatum, genaue Größe, Zugriffsrechte, ..) müssen ebenfalls im Verzeichniseintrag gespeichert werden *(zusätzlich zum Namen und der ersten Plattenblockadresse)*!

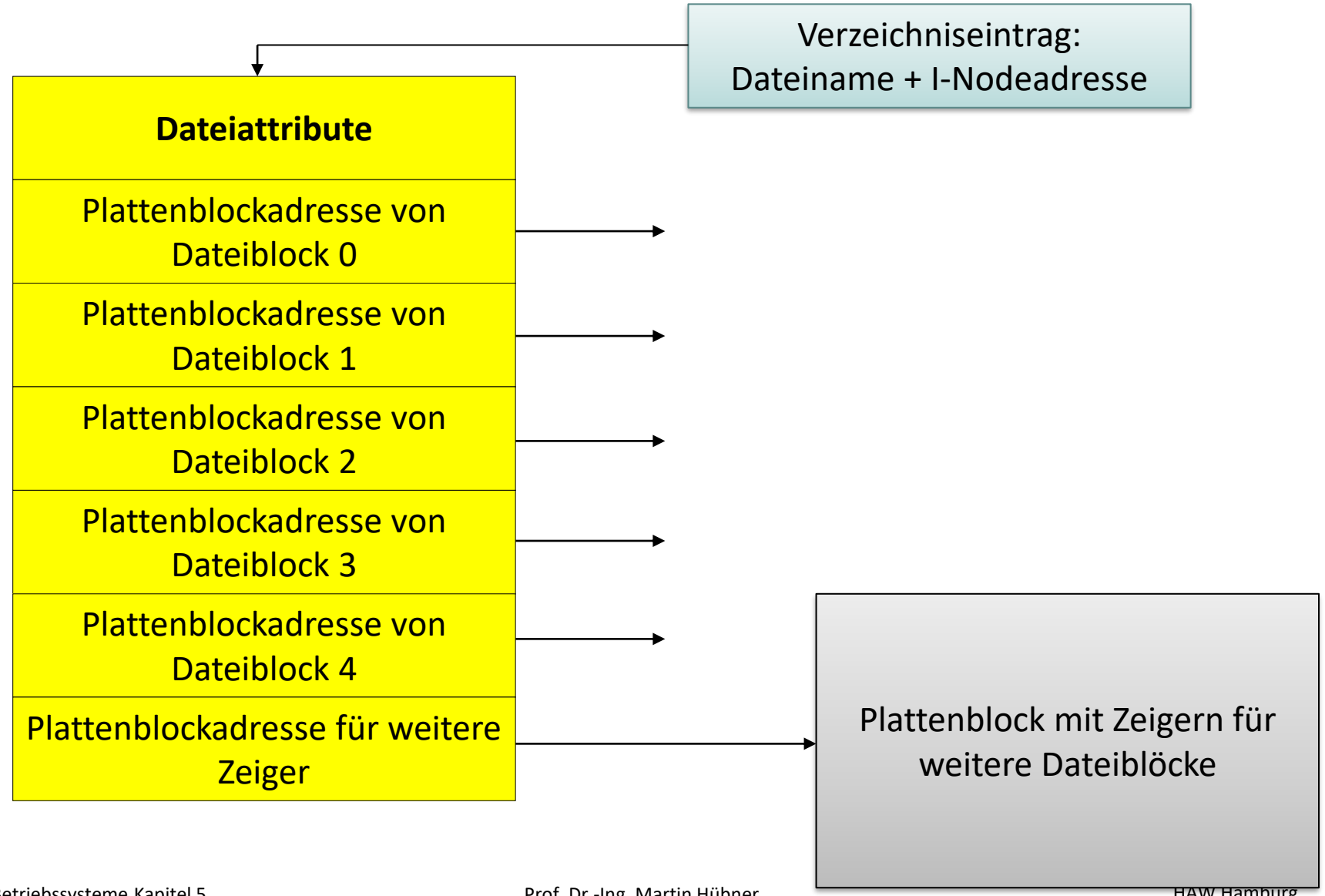
Beispiel: File Allocation Table („FAT“)



4. Belegung durch separate Zeigerlisten („Index-Nodes“ / „I-Nodes“)

- Idee: Alle Zeiger auf die Blöcke **einer** Datei (Plattenblockadressen) werden in einer eigenen Datenstruktur (**I-Node**) zusammengefasst
- Der Dateieintrag im Verzeichnis enthält nur den Dateinamen und die Adresse des I-Nodes dieser Datei
- Der i-te Eintrag in der I-Node-Zeigerliste zeigt auf den i-ten Block der Datei.
- Falls bei fester Größe des I-Node die Anzahl der Einträge nicht ausreicht, kann auf eine weitere Liste verwiesen werden.
- Weitere Informationen über eine Datei *(Attribute wie Änderungsdatum, genaue Größe, Zugriffsrechte, ..)* werden ebenfalls zentral **im I-Node** der Datei gespeichert *(nicht im Verzeichniseintrag)*!

Struktur eines I-Nodes



Vorteile von I-Nodes gegenüber einer FAT

- Nur die benötigten I-Nodes müssen im Hauptspeicher liegen, nicht die gesamte FAT *(Problem bei großen Festplatten!)*
- Schnellerer Zugriff auf weiter hinten liegende Blöcke
- Geringere Auswirkung von Lesefehlern *(i.d.R. ist nur ein Block / eine Datei betroffen)*
- Einrichten von „Hard Links“ ist möglich *(mehrere Verzeichniseinträge für dieselbe Datei)*
- Direkte Zuordnung von Datei-Informationen ist möglich *(unabhängig von einem Verzeichniseintrag)*

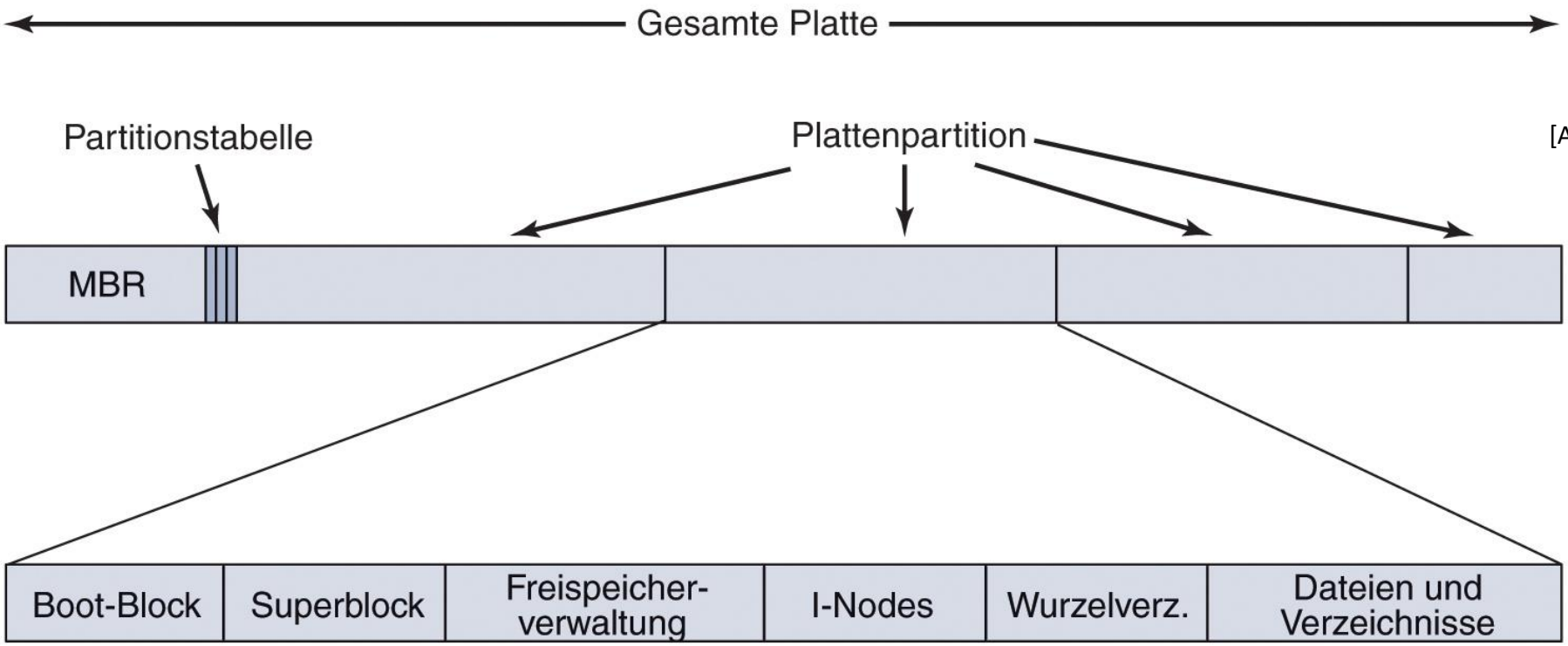
Dateisystem-Blockgröße

- **Block** *(Windows: „Cluster“)* = kleinste Speicher-/Zugriffseinheit für das **Dateisystem**
  → ***Umrechung auf Sektoren für Plattencontroller erfolgt durch das Betriebssystem!***

- Welches ist die optimale Blockgröße?
- Zu groß: Platzverschwendung bei kleinen Dateien
- Zu klein: hoher Verwaltungsaufwand (viele Zugriffe bei großen Dateien)
- Erfahrungswerte:
  - klassisches UNIX: Blockgröße 1 KiB
  - MS-DOS/Windows: 0,5 KiB – 64 KiB (abhängig von Plattengröße und Anwendung)
    - Standard: 4 KiB
    - Eigene Messungen: ?% der Dateien < 4 KiB → FileSizeAnalyzer.java

Master Boot Record (MBR) und Partitionen

- Sektor 0 eines Datenträgers enthält den „Master Boot Record“ (MBR)
- Aufteilung einer Platte in unabhängige Partitionen
- Für jede Partition ist ein unterschiedliches Dateisystem möglich!



Was passiert beim Bootvorgang?

- Das UEFI (*„Unified Extensible Firmware Interface“*, früher BIOS) im ROM liest den MBR ein und führt das dort gespeicherte Programm aus
- Das MBR-Programm ermittelt (i.d.R. aus der GUID-Partitionstabelle) die aktive Partition, lädt deren Bootblock in den Hauptspeicher und führt ihn aus
- Das Bootblock-Programm lädt das Betriebssystem, welches in der Partition enthalten ist.
- Das Betriebssystem holt sich sämtliche Informationen über das Dateisystem aus dem Superblock (Typ des Dateisystems, Anzahl Blöcke, ..)

Wie kommt das Dateisystem auf die Partition?

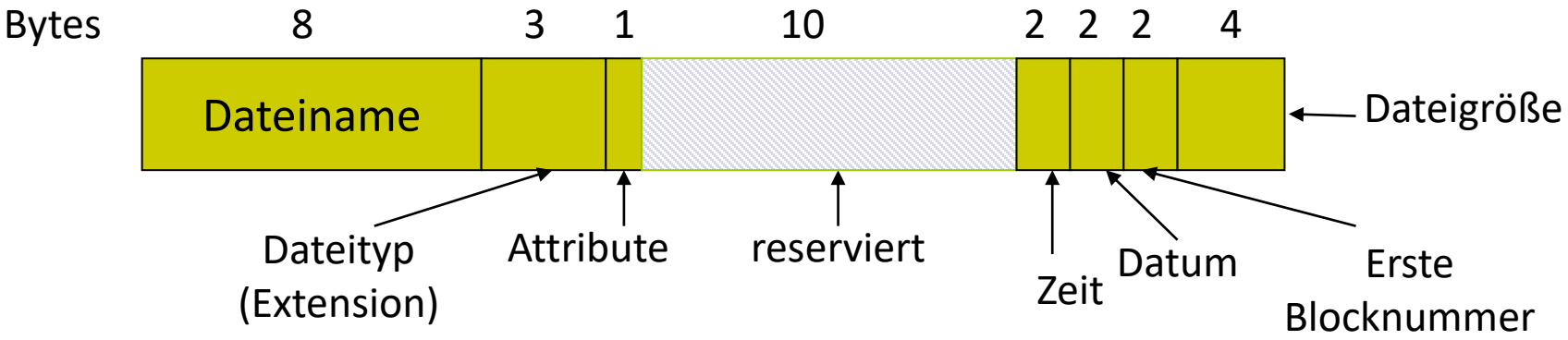
- **High-Level Formatierung** einer Datenträger-Partition („format“-Befehl): Erzeugt
  - Bootblock
  - Superblock
  - Liste der freien Plattenbereiche
  - Wurzelverzeichnis (Root)
  - leeres Dateisystem
  - Dateisystem-Kennzeichen in der Partitionstabelle
- **Low-Level Formatierung** eines Datenträgers (Anlegen von Sektoren etc.): bereits vom Hersteller

Kapitel 5
Externe Geräte & Dateisysteme

1. Externe Geräte
   a) Controller und Driver
   b) Festplattenorganisation
2. Dateisysteme
   a) Dateien und Verzeichnisse
   b) Implementierung von Dateisystemen
   c) Dateisystem-Beispiele
3. Zuverlässigkeit von Dateisystemen
   a) Konsistenz von Dateisystemen
   b) Zuverlässiger Betrieb von Dateisystemen

MS-DOS

- Ab MS-DOS 2.0 (1983): Benutzung einer **FAT**
- Keine Unterscheidung von verschiedenen Benutzern möglich
- **Struktur eines Verzeichniseintrags:**



- Einführung von zusätzlichen Attributen (**r**eadonly, **a**rchive, **h**idden, **s**ystem)
- Adressierung des ersten Plattenblocks:
  max. 2 Byte (16 Bit) zur Verfügung → FAT12 / FAT16
- Ab Windows95:
  - Speicherung von langen Dateinamen in zusätzlichen Verzeichniseinträgen
  - 2 reservierte Bytes werden ab Windows95SE zusätzlich für den zweiten Teil der Startadresse (1. Plattenblock) verwendet → **FAT32**

FAT32

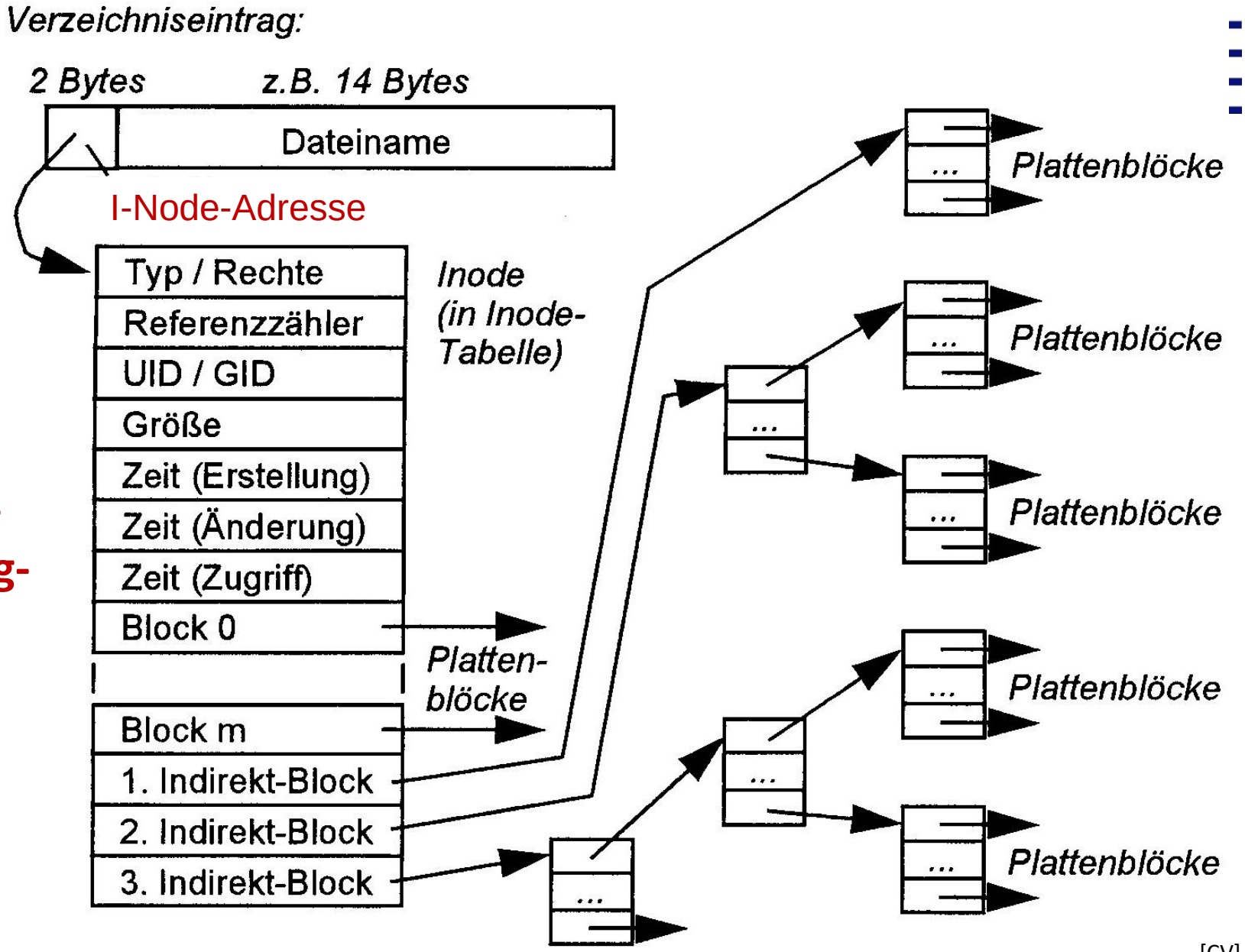
- Für Plattenblockadressen werden nur 28 Bit verwendet:
  → 2^{28} Plattenblockadressen pro Dateisystem möglich
- Größe der FAT: 2^{28} * 4 Byte = 2^{30} = 1 GiB
- Vorteile FAT-32:
  - Einfache Implementierung → die meisten Betriebssysteme können darauf zugreifen
  - Viele externe Geräte verwenden FAT32 (Digitalkamera, MP3-Player, Fernseher, ...)
- Nachteile FAT-32:
  - FAT verbraucht viel Hauptspeicher
  - Maximale Dateigröße: 4 GiB (da im Verzeichniseintrag nur 4 Byte für die Dateigröße vorgesehen sind)
  - Siehe Folie 33
- Erweiterte Version (optimiert für Flash-Speicher): **extFAT**
  - Große Dateien / große Blöcke möglich

UNIX V7

UNIX-V7-Dateisystem (1979) definiert die Grundprinzipien

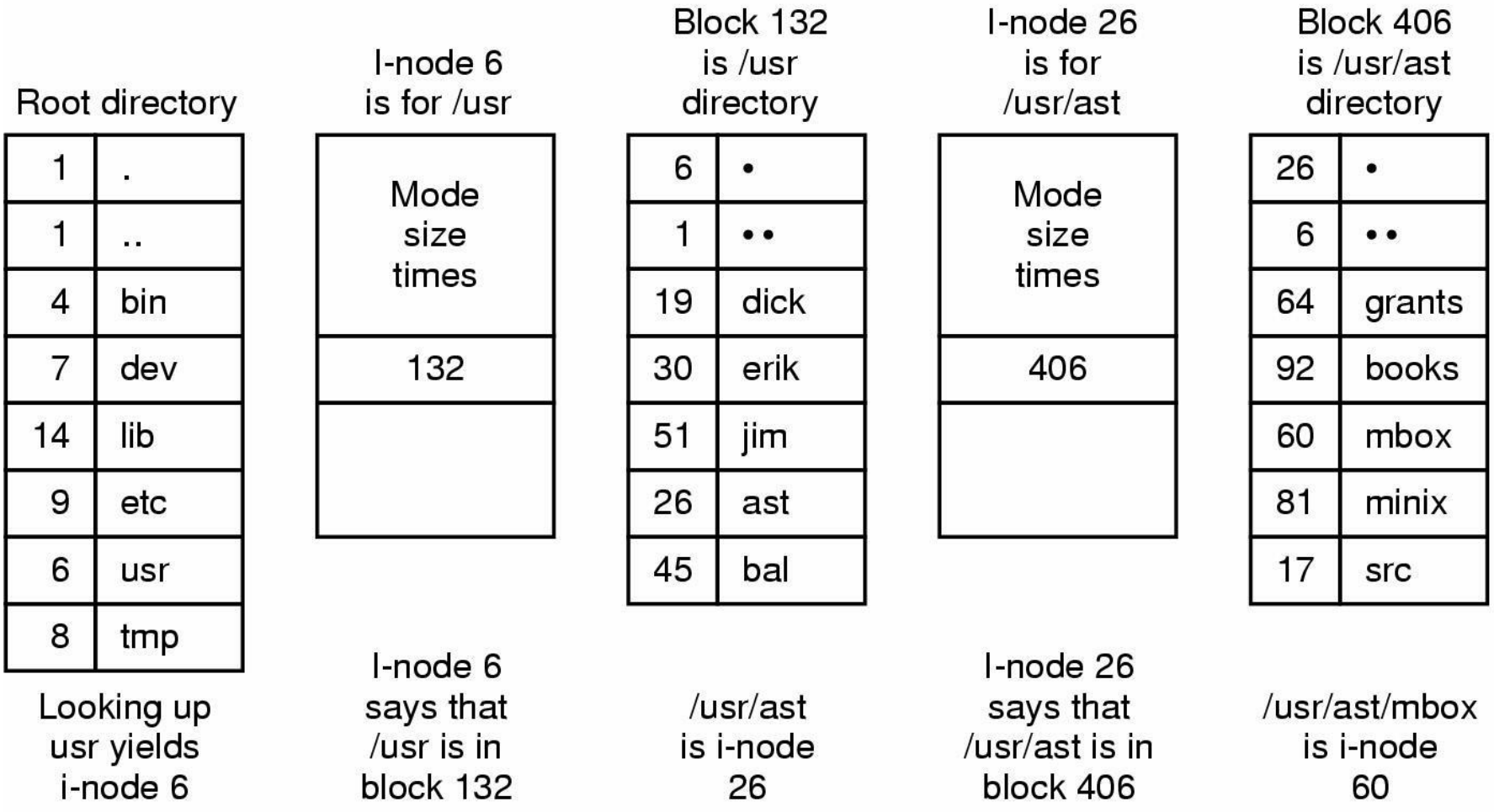
- Dateien sind byte-orientiert (**Streams**), die Verwaltung erfolgt über **I-Nodes**
- Jede Partition enthält einen Boot-Block, einen Superblock, eine I-Node-Tabelle und einen Bereich für die Datenblöcke
  - In einem Eintrag in der I-Node-Tabelle sind die Informationen über eine Datei gespeichert (I-Node):
    - Datei-Attribute
    - Die Zeiger für die ersten m Plattenblöcke werden direkt im I-Node gespeichert (m=10 oder 12)
    - Bei großen Datei verweisen „Indirekt-Blöcke“ auf weitere Zeigertabellen für jeweils n Blöcke ($n \geq 256$)
- Jede Partition (jedes Dateisystem) kann in das Hauptverzeichnis an jeder Stelle des Teilbaums eingehängt werden (mount)

UNIX V7: Verzeich-niseintrag- und I-Node-Struktur



[CV]

Beispiel: Zugriff auf /usr/ast/mbox (UNIX V7)



[AT]

Unix: Links auf Dateien

`$ ln datei1 linkdateiname1`

erzeugt **Hard Link** = neuer Verzeichniseintrag **`linkdateiname1`** mit derselben I-Nodeadresse wie **`datei1`**

`$ ln -s datei1 linkdateiname1`

erzeugt **Soft Link** = neue eigene Spezial-Datei **`linkdateiname1`** vom Typ „Link“ (mit eigenem I-Node), in die der Dateipfad der Zieldatei **`datei1`** als Inhalt eingetragen ist

Ergebnis in beiden Fällen: `datei1` ist unter dem Namen `linkdateiname1` erreichbar.

→ Unterschiede zwischen Hard- und Soft Link?

Tipp: **`$ ls -li`** gibt die I-Nodeadresse zu jedem Dateinamen aus!

Linux-Dateisysteme

- MINIX (Tanenbaum)
  - erstes Linux-Dateisystem mit starken Beschränkungen
  - Dateigröße ≤ 64 MiB, Dateinamen ≤ 14 Zeichen
- ext
  - Dateien bis 2 GiB, Dateinamen bis 255 Zeichen, langsamer als MINIX
- ext2
  - Dateien bis 2 TiB, bessere Performance, Linux-Standard ab 1993
- ext3
  - Erweitert ext2 um Journaling (Logging), Linux-Standard ab 2001
- **ext4**
  - Maximale Dateigröße: 1 EiB (2^{60} = 1.152.921.504.606.846.976 Byte)
  - Unterstützung für SSDs, höhere Sicherheit, Linux-Standard ab 2008
- **vfs**
  - Virtuelles Dateisystem, dient zum Einhängen (mounten) unterschiedlicher realer Dateisysteme in einen Verzeichnisbaum

Windows: NTFS ("New Technology File system")

- Version 1.0 für Windows NT 1993
- Aktuell: Version 3.1 ab Windows XP 2001

- I-Node-basiert (Unix/Linux), aber eigene Begrifflichkeiten:
  - Partition ↔ Volume
  - I-Node-Tabelle ↔ Master File Table
  - I-Node ↔ Master File Table – Eintrag
  - I-Node-Adresse ↔ Master File Table – Index
  - (Platten-)Block ↔ Cluster

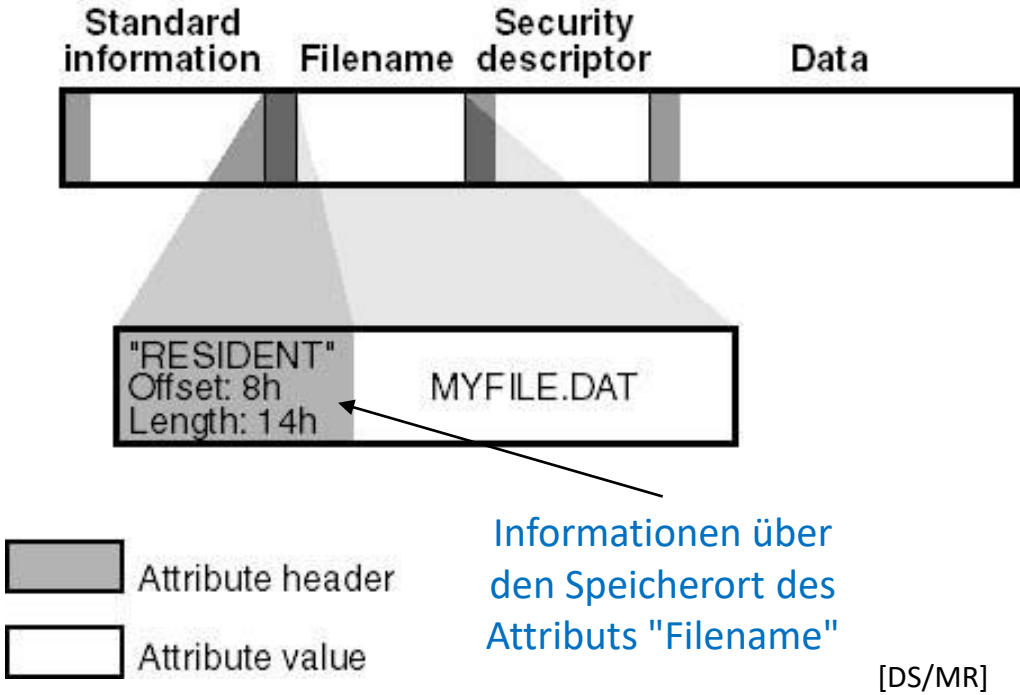
- Verwendung auch unabhängig von Windows möglich

Struktur eines NTFS-Volumes

1. Boot Block
2. Master File Table (MFT)
   - **Ein MFT-Eintrag** hat einen MFT-Index und speichert für eine Datei:
     - Dateiname
     - ... alle anderen Dateiattribute
     - Dateiinhalt, i.a. durch Zeiger auf Cluster
3. Bereich für Systemdateien
   - MFT – Kopie
   - Log Datei – für Recovery
   - Cluster Bitmap – Liste freier Cluster
   - Attribut-Definitionen (Typcode, Name)
   - ...
4. Bereich für Benutzerdateien
   - Die freien / belegten Benutzer-Cluster

Master File Table – Einträge

- Jeder Eintrag hat eine feste Länge *(1 KiB)*
- Mehrere Einträge pro Datei möglich *(für große Dateien)*
- Jeder Eintrag ist eine Liste von Paaren (Attributheader, Wert)
  - Ein Attribut wird durch einen Typcode identifiziert und hat einen Wert
  - Der Attributheader liefert Informationen über
    - den Typcode des Attributs
    - Länge und Speicherort des Wertes
  - Der Wert jedes Attributs ist ein Bytestrom



[DS/MR]

Konsequenzen der Verwendung von Bytesströmen für alle Attribute

- Die Daten einer Datei sind Wert des Attributs $DATA
- Standardmäßig wird bei Dateioperationen nur auf das Attribut $DATA zugegriffen (read, write, getSize, …)
- Durch Definition neuer Attribute können „versteckt“ weitere Informationen als Bytestrom in einer Datei gespeichert werden!
- Anwendungen:
  - Beliebige Speicherung von Meta-Daten für eine Datei (Eigenschaften, Beschreibung)
  - Speicherung mehrerer Versionen (Word)
  - Speicherung in mehreren Formaten (Bilder, Audio, Video)
- Beispiel: *myfile.dat:streamname* (→ Demo!)

NTFS-Implementierung von Dateien

- Kleine Dateien: residente Speicherung der Daten als Bytestrom in der MFT
- Mittelgroße Dateien: Verweis auf zusammenhängend belegte Cluster (Plattenblöcke) in mehreren „Runs“ =
  - Virtuelle Clusternr. (n-ter Block der Datei, beginnend bei 0)
  - Logische Clusternr. (Adresse des Start-Blocks)
  - Anzahl Cluster (Anzahl Blöcke dieses Runs)

  Beispiel für eine Datei mit 3 Runs und 12 Clustern (Blöcken):

  (0,1355,4) (4,1588,3) (7,2033,5)
- Große Dateien / stark fragmentierte Dateien:
  - Mehrere MFT-Einträge, falls zu viele Runs für einen Eintrag vorhanden sind
  - Runs werden in eigenen Clustern (Plattenblöcken) gespeichert, falls zu viele MFT-Einträge nötig sind

NTFS-Implementierung von Verzeichnissen

Erinnerung: Ein Verzeichnis ist selbst eine (spezielle) ***Datei***

Ein Verzeichniseintrag für eine Datei besteht aus

- Dateireferenz (Master File Table - File Reference):
  - 48-Bit MFT-Index (~ I-Nodenummer)
  - 16-Bit Sequenznummer *(gelöschte Dateien können so trotz Wiederverwendung der Indexnr. wiederhergestellt werden)*
- Dateiname ← Kopie des MFT-Eintrags
- Änderungsdatum ← Kopie des MFT-Eintrags
- Größe ← Kopie des MFT-Eintrags

Spezielle Speichertechnik für große Verzeichnisse:

- Speicherung der Verzeichniseinträge in Clustern (Plattenblöcken) unter Verwendung eines B+ - Baums zur Zugriffsoptimierung

Weitere NTFS – Eigenschaften

- Ein Volume (Partition) kann sich über mehrere physikalische Platten verteilen – max. Größe: 2^{64} Byte
- Dateioperationen sind atomare Transaktionen
  - Journaling (alle Befehle einer Transaktion werden vor Ausführung der Transaktion in eine Log-Datei geschrieben)
  - Rollback oder Fortsetzung bei Wiederherstellung, falls Transaktion nicht abgeschlossen wurde

Kapitel 5
Externe Geräte & Dateisysteme

1. Externe Geräte
   a) Controller und Driver
   b) Festplattenorganisation
2. Dateisysteme
   a) Dateien und Verzeichnisse
   b) Implementierung von Dateisystemen
   c) Dateisystem-Beispiele
3. Zuverlässigkeit von Dateisystemen
   a) Konsistenz von Dateisystemen
   b) Zuverlässiger Betrieb von Dateisystemen

Zuverlässigkeit von Dateisystemen

- Zuverlässigkeit = **Schutz vor Datenverlust** durch
  - technische Probleme
  - Dummheit

  im Sinne von „fehlertolerant“
- Wichtiges (teures) Ziel in der Praxis:
  - (Hoch-) Verfügbarkeit der Daten / Dienste
- Themen:
  - **Konsistenz von Dateisystemen** (Reparatur)
  - **Zuverlässiger Betrieb von Dateisystemen** (Zugriffsorganisation, Plattenspiegelung, Datensicherung / Backup)

Kapitel 5
Externe Geräte & Dateisysteme

1. Externe Geräte
    a) Controller und Driver
    b) Festplattenorganisation
2. Dateisysteme
    a) Dateien und Verzeichnisse
    b) Implementierung von Dateisystemen
    c) Dateisystem-Beispiele
3. Zuverlässigkeit von Dateisystemen
    a) Konsistenz von Dateisystemen
    b) Zuverlässiger Betrieb von Dateisystemen

Problemstellung

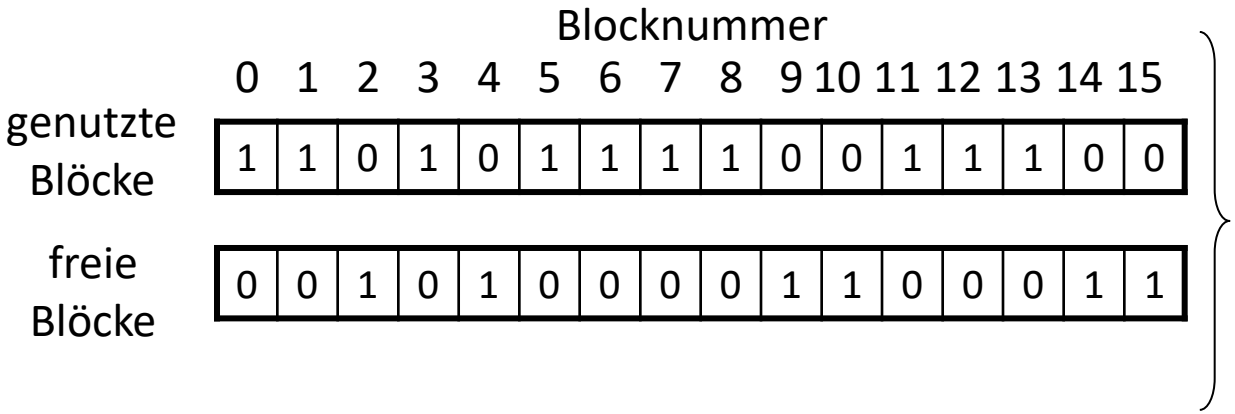
- Systemabsturz (Prozessorausfall)
  - ➔ Daten im Cache noch nicht auf die Platte geschrieben
  - ➔ Inkonsistenz des Dateisystems
    - FAT / MFT / I-Nodetabelle entspricht nicht dem Zustand auf der Platte!
- Tools zur Prüfung und Reparatur eines Dateisystems:
  - FAT: **scandisk**
  - NTFS: **chkdsk**
  - UNIX: **fsck**
- *Alle Tools haben eine ähnliche Funktionsweise, sind jedoch spezifisch auf das Dateisystem zugeschnitten!*

fsck – Prüfalgorithmen (UNIX)

- Konsistenzüberprüfung anhand von **Blöcken**
  - Tabelle für benutzte Blöcke aufbauen anhand der aktuellen I-Node-Informationen
    - Blocknummer referenziert in I-Node: erhöhe Zähler
  - Tabelle für freie Blöcke aufbauen anhand der aktuellen Freiliste
    - Blocknummer eingetragen in Freiliste: erhöhe Zähler
  - Prüfung: Vergleiche Tabellen!
    - Konsistenz gegeben, wenn für jeden Block eine 1 in genau einer der beiden Tabellen eingetragen ist!

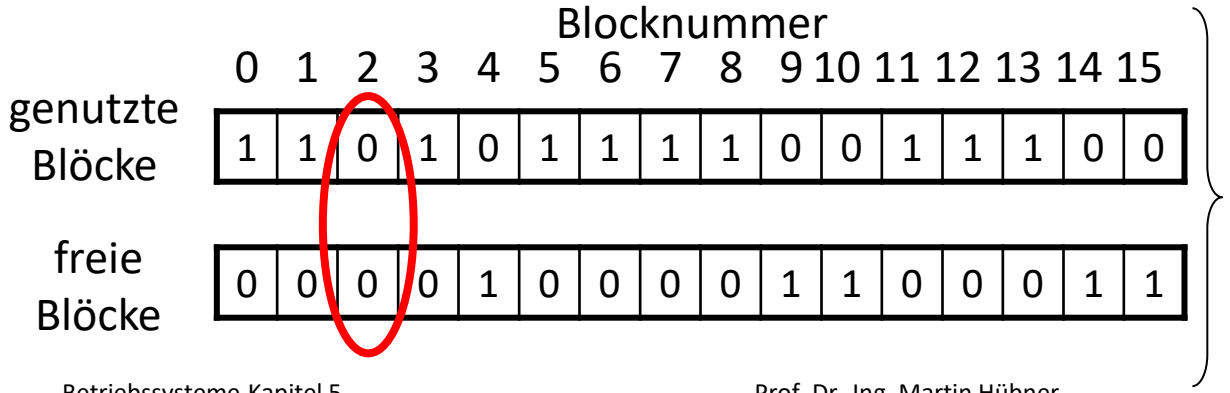
fsck-Konsistenzprüfungs – Fälle (1)

Fall 1: Dateisystem ist konsistent ("ok")



Alles OK:
Genau eine ‚1'
für jeden Block vorhanden:
entweder in 1. Tabelle
oder in 2. Tabelle (xor)

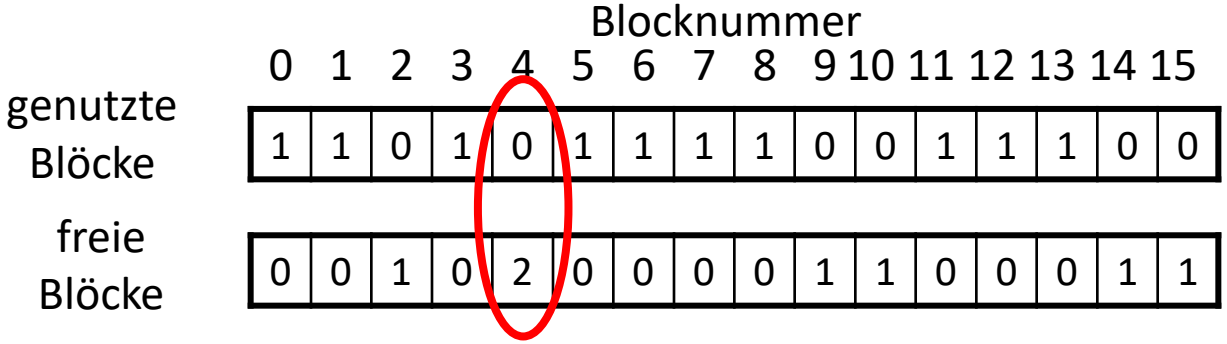
Fall 2: Dateisystem ist inkonsistent wegen fehlenden Blocks *("missing block")*



Inkonsistentes Filesystem:
fehlender Block
Behebung:
fsck fügt Block
in die Freiliste ein
(oder Block wird neue Datei in "Lost&Found")

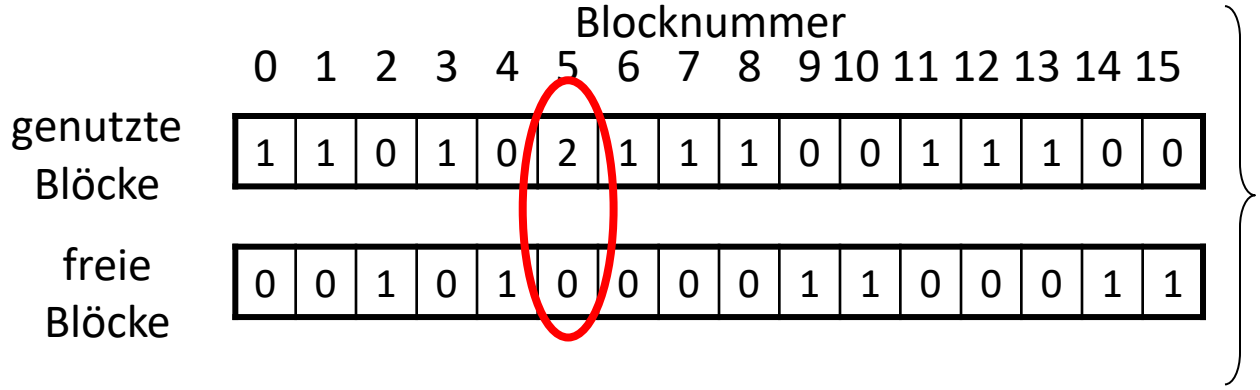
fsck-Konsistenzprüfungs – Fälle (2)

Fall 3: Dateisystem ist inkonsistent wegen doppelt vorhandenen Blocks in der Freiliste



Inkonsistentes Filesystem:
doppelter Block in Freiliste
Behebung:
fsck passt die Freiliste an

Fall 4: Dateisystem ist inkonsistent, da derselbe Block in mehreren Dateien verwendet wird



Inkonsistentes Filesystem:
Block in mehreren Dateien
Behebung:
1. fsck belegt neuen freien Block, kopiert Inhalt von Block 5 dorthin und passt einen I-Node an
2. Fehler wird Nutzer angezeigt

fsck – Prüfalgorithmen (UNIX)

- Konsistenzüberprüfung anhand von **Dateien**
  - Tabelle von Zählern für jede Datei
  - Rekursiver Durchlauf durch den Verzeichnisbaum: erhöhe Zähler für jede in einem Verzeichnis referenzierte Datei
  - Vergleich der Tabelle mit den Referenz-Zählern in den I-Nodes! Bei Abweichung:
    - Referenz-Zähler im I-Node anpassen

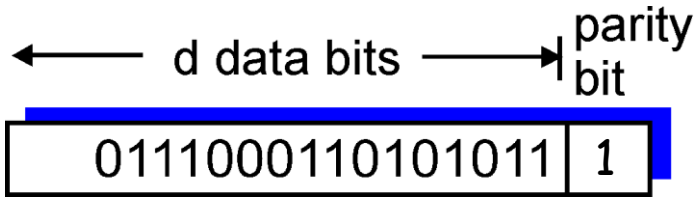
Kapitel 5
Externe Geräte & Dateisysteme

1. Externe Geräte
   a) Controller und Driver
   b) Festplattenorganisation
2. Dateisysteme
   a) Dateien und Verzeichnisse
   b) Implementierung von Dateisystemen
   c) Dateisystem-Beispiele
3. Zuverlässigkeit von Dateisystemen
   a) Konsistenz von Dateisystemen
   b) Zuverlässiger Betrieb von Dateisystemen

Fehlererkennung und -korrektur durch Paritätsprüfung

Single Bit Parity:

Detect single bit errors



Berechnung der Parität:
Verknüpfe alle Datenbits mit XOR

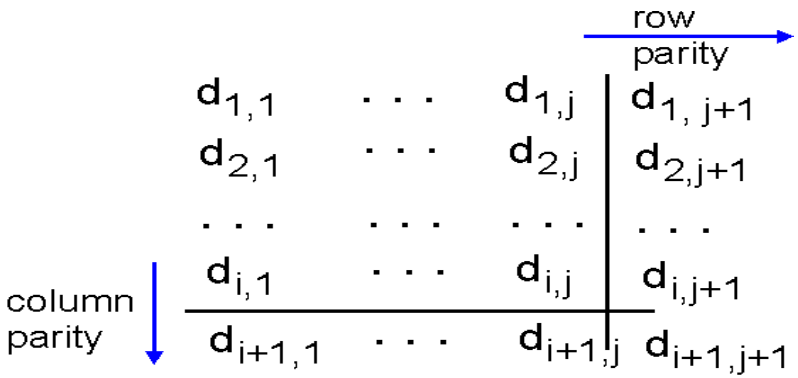
Ergebnis:

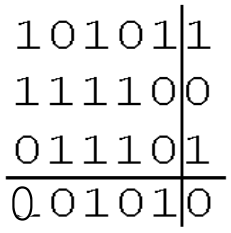
0 → gerade Anzahl Einsen

1 → ungerade Anzahl Einsen

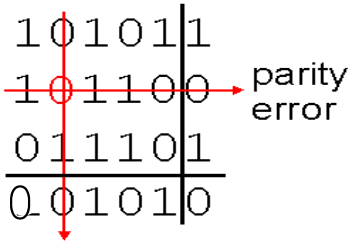
Two Dimensional Bit Parity:

Detect *and correct* single bit errors





no errors

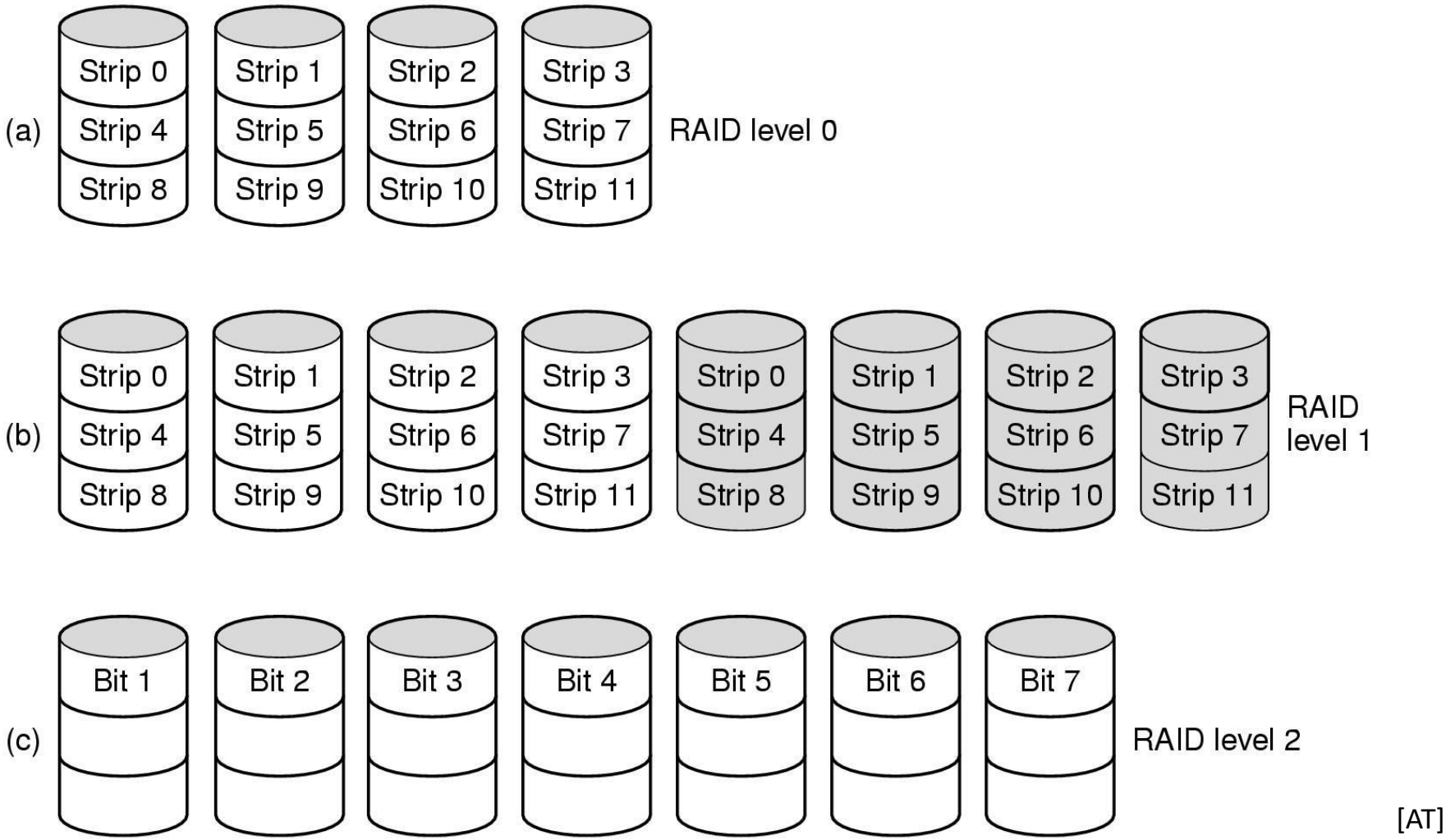


correctable single bit error

RAID

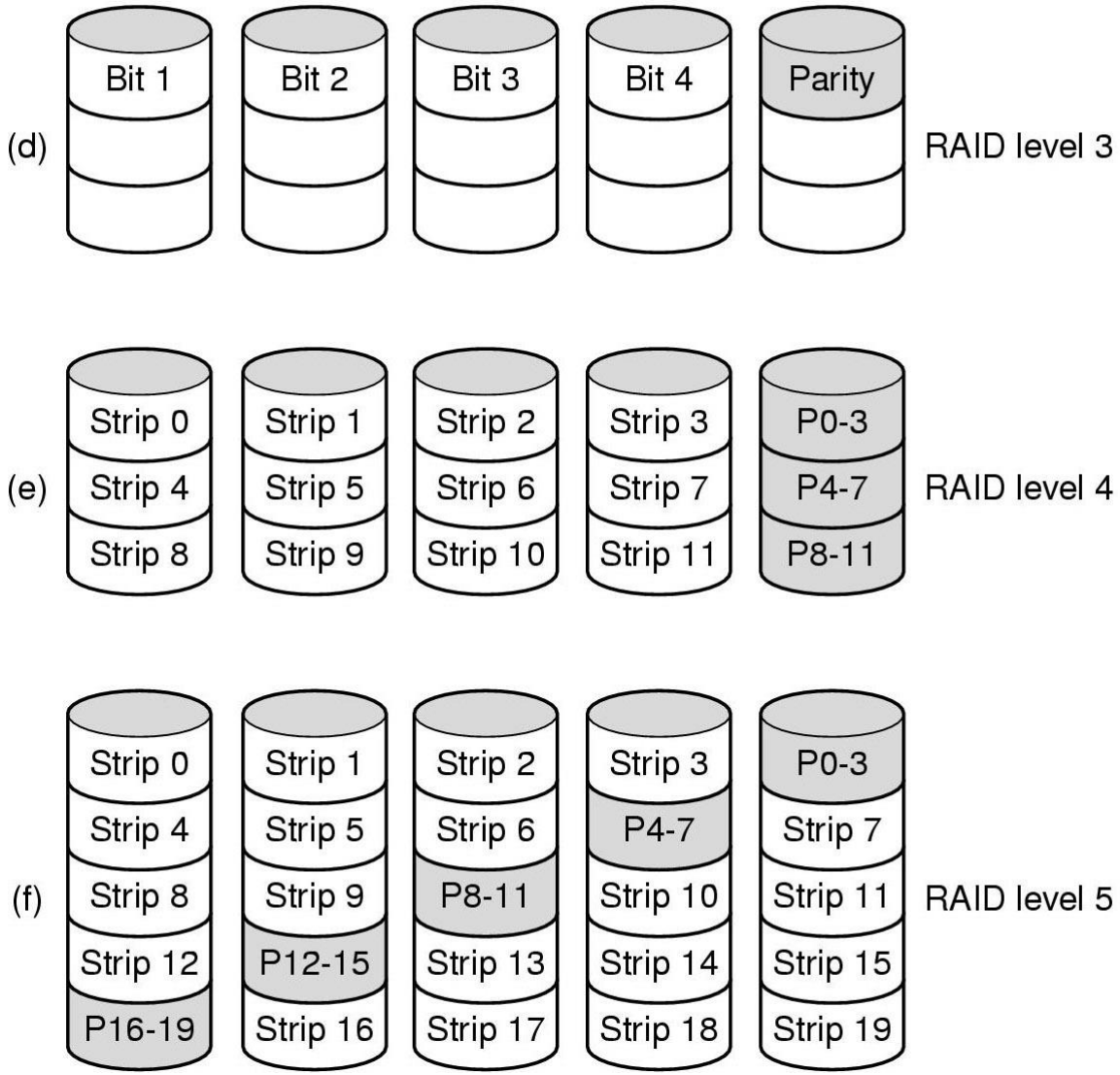
- RAID = „Redundant Array of Inexpensive Disks“
- Daten werden auf mehrere (billige) Platten verteilt
- Ziele:
  - Hardware – Ausfall von einzelnen Platten absichern
  - Lesegeschwindigkeiten durch parallelen Zugriff erhöhen
- Schnittstelle: RAID-Controller – für das Betriebssystem von „normalem“ Plattencontroller nicht zu unterscheiden
- Betriebs-Varianten: RAID 0 – RAID 6

RAID-Varianten 0 - 2



[AT]

RAID-Varianten 3 - 5



[AT]

RAID-Varianten

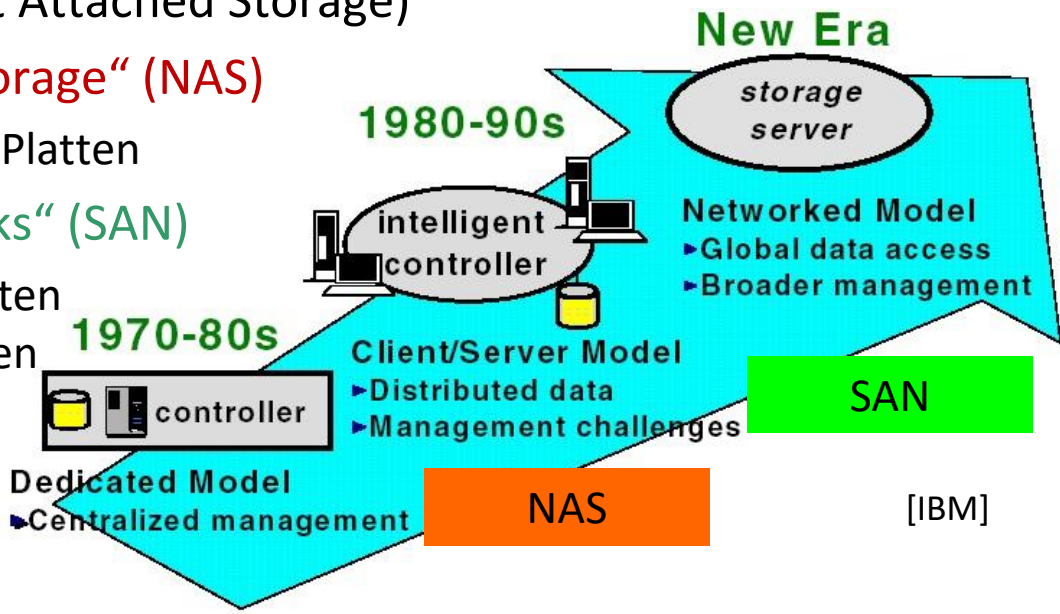
- **RAID 0**
  - Aufteilung der Dateien in *Strips* (Streifen) und Verteilung über alle N Platten → schnelleres Lesen durch parallelen Zugriff!
- **RAID 1**
  - wie RAID 0, aber vollständige Spiegelung (2 * N Platten nötig)
  - jeder Strip wird auf zwei Platten geschrieben
  - hohe Sicherheit, aber viele Zusatzplatten nötig
- **RAID 2**
  - Aufteilung der Dateien auf Bitebene mit mehreren Parity-Bits ("Hamming"-Codes) zur Fehlerkorrektur → N + 3 Platten
  - Bit-Synchronisation aufwändig
- **RAID 3**
  - wie RAID 2, aber nur mit einem Parity-Bit → N + 1 Platten

RAID-Varianten

- **RAID 4**
  - wie RAID 3, aber auf Basis von Strips → N + 1 Platten
  - Parity-Platte ist Engpass, da bei jeder Schreiboperation nötig!
- **RAID 5**
  - wie RAID 4 (N + 1 Platten), aber die Parity-Bits sind über alle Platten verteilt
  - der Ausfall **einer** (beliebigen) Platte kann kompensiert werden (Wiederherstellung dauert aber ggf. lange)
  - (relativ) billige Lösung für große Datenmengen
  - zusätzlich Absicherung: „Hot Spare“ – Platte als Reserve
- **RAID 6**
  - wie RAID 5, nur mit zwei zusätzlichen Parity-Platten (N + 2)
  - der Ausfall von **zwei** (beliebigen) Platten kann kompensiert werden

Organisation des Festplattenzugriffs

- 1. Stufe: Controller und Platten einem (großen) Server direkt zugeordnet (**DAS** – Direct Attached Storage)
- 2. Stufe: „Network Attached Storage“ (NAS)
  - Spezielle Fileserver mit lokalen Platten
- 3. Stufe: „Storage Area Networks“ (SAN)
  - Zusammenfassung der Festplatten in spezialisierten Serversystemen



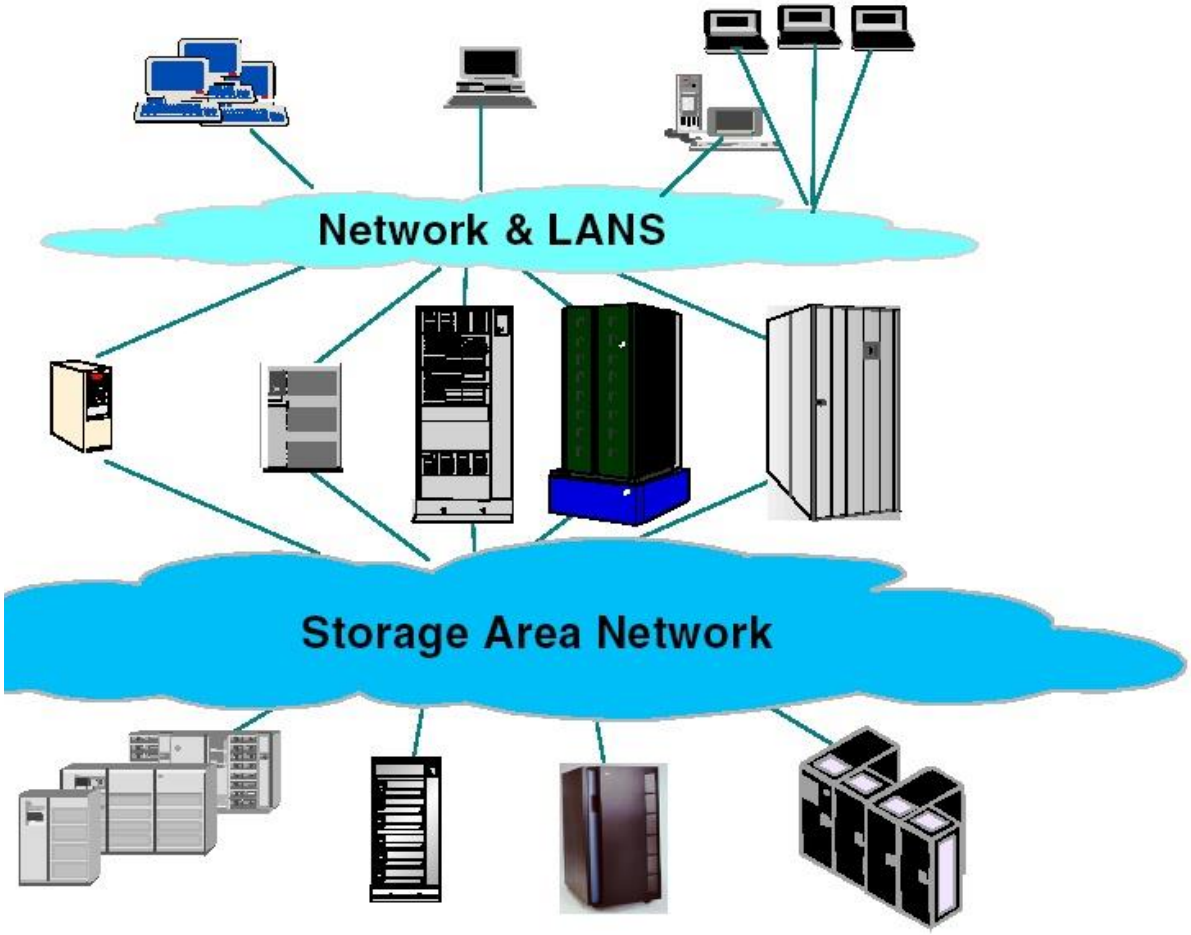
[IBM]

- Aktuell: „Hyper Converged Infrastructures“ (HCI)
  - Virtualisierte Server, virtualisiertes Netzwerk, virtualisierte Festplatten
  - Physikalisch: An jedem Server sind die Festplatten (SSDs) direkt angeschlossen (DAS -- *siehe 1. Stufe ...)*

Storage-Area Network (SAN)

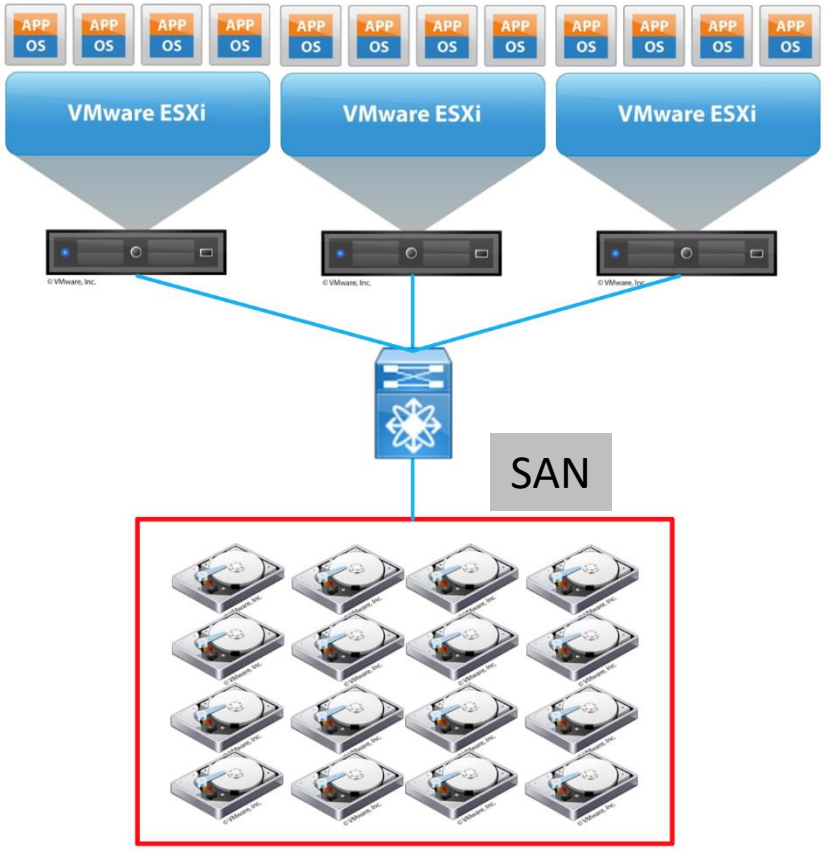
Typische Eigenschaften:

- Flexible Zuordnung von Festplatten(-Servern) zu Applikations-Servern
- Zugriff über ein schnelles, separates Netzwerk (Fiber Channel / iSCSI / Gigabit-Ethernet)
- Zentrales Speichermanagement und zentrale Wartung
- Hochverfügbarkeit durch „zuverlässigen Speicher“ (ggf. RAID 1)

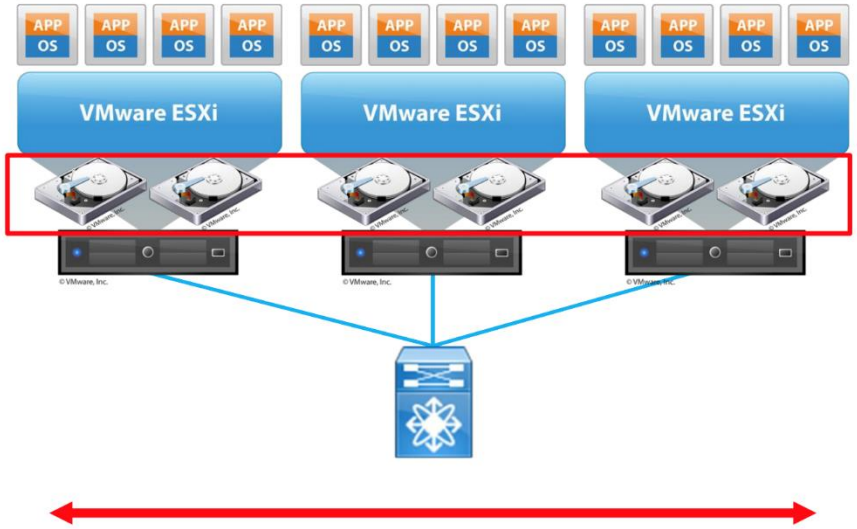


[IBM]

Hyper Converged Infrastructure (HCI)



Converged infrastructure



Distributed File System

z.B. CEPH

Hyper-Converged infrastructure

Datensicherung / Versionierung von Dateien durch „Snapshot“-Technologie

- Snapshot erzeugen: „Einfrieren“ des Dateistands durch Kopieren der aktuellen I-Nodetabelle („Snapshot“)
- Bei anschließenden Änderungen an einem Plattenblock (im Hauptspeicher): Zurückschreiben auf die Platte als **neuer** Block ("Copy-on-Write" – Technik) mit Anpassung des I-Nodes in der aktuellen I-Nodetabelle (neue Plattenblockadresse)
- Der alte Block wird nicht überschrieben, auf ihn kann über den Snapshot ("alte" I-Nodetabelle) konsistent zugegriffen werden
- Vorteile:
  - Garantiert konsistenter Dateisystem-Zustand (auch im Snapshot)
  - Nur die neu geänderten Blöcke müssen kopiert werden
  - **Snapshots können im laufenden Betrieb angelegt werden (Datensicherung, Versionierung, ..!)**

Organisation der Datensicherung

- Was soll gesichert werden?
  - Daten, die auf keinem anderen Medium verfügbar sind
- Wann soll gesichert werden?
  - periodisch / auf Anforderung
  - vollständig / inkrementell (nur Veränderungen)
- Wie soll gesichert werden?
  - Medium (Festplatte, USB-Stick, Cloudspeicher, Band, ...)
  - unkomprimiert / komprimiert
- „Einfrieren“ des Dateisystems nötig („Snapshot“)?
- Aufbewahrung der Sicherungsmedien?

Ende des 5. Kapitels: Was haben wir geschafft?

1. Externe Geräte
   a) Controller und Driver
   b) Festplattenorganisation
2. Dateisysteme
   a) Dateien und Verzeichnisse
   b) Implementierung von Dateisystemen
   c) Dateisystem-Beispiele
3. Zuverlässigkeit von Dateisystemen
   a) Konsistenz von Dateisystemen
   b) Zuverlässiger Betrieb von Dateisystemen