Freie Universität Berlin

Implementierung von Dateisystemen

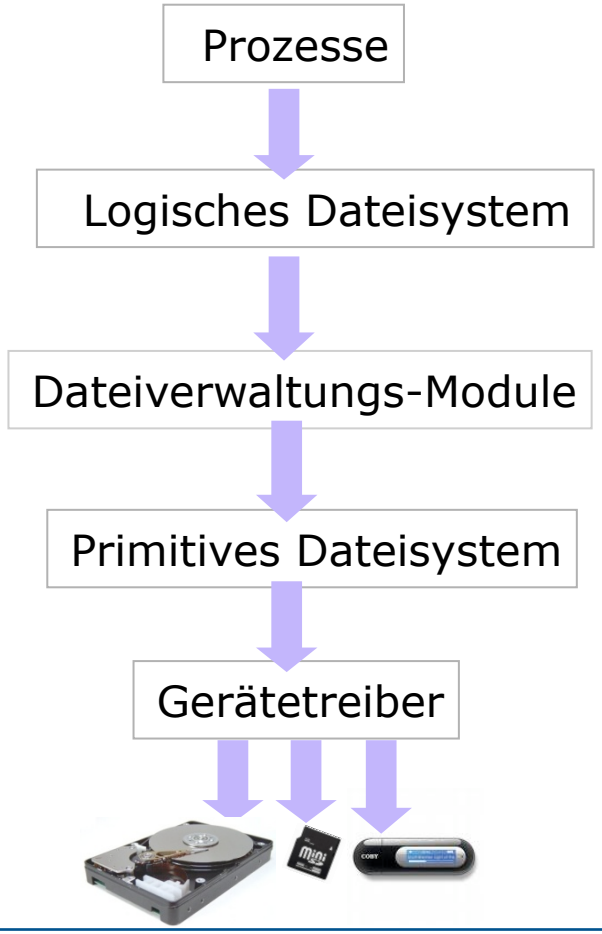
Prof. Dr. Margarita Esponda

WS 2011/2012

Implementierung von Dateisystemen

- Schichten-Architektur des Dateisystems
- Implementierung von Dateisystemen
- Implementierung der Verzeichnisse
- Allokationsmethoden
- Verwaltung von freiem Platz
- Effizienz- und Performance-Probleme
- Wiederherstellung von Information (*Recovery*)
- Log-Strukturen in Dateisystemen
- NFS (*Network File System*)
- WAFL (*Write Anywhere File Layout*)

Schichten eines Dateisystems

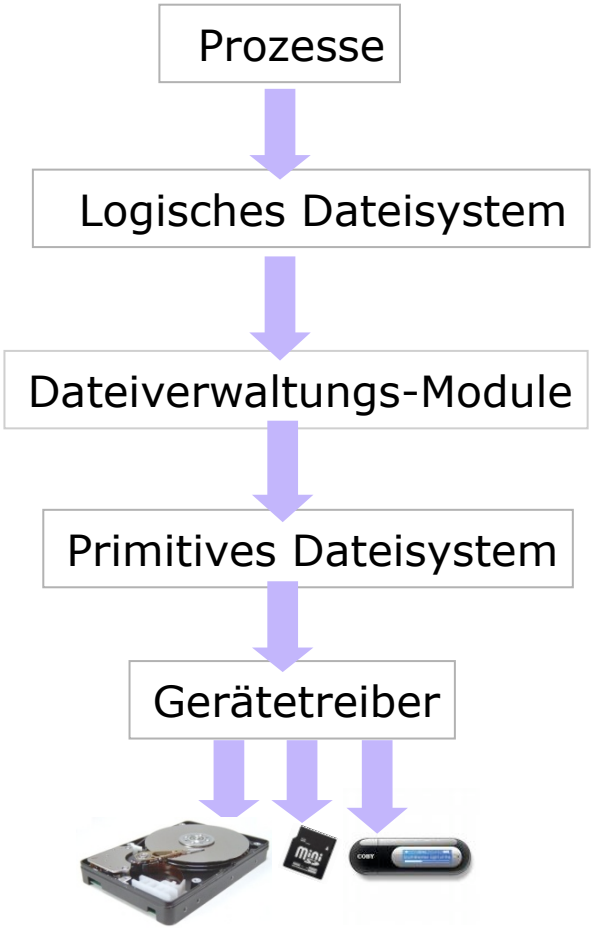


- Metadaten der gesamten Dateisystemstrukturen
- Verwaltet die Verzeichnisstruktur
- FCBs file-control blocks (inode in Linux)
- Schutzmechanismen

- Übersetzung von logischen Adressen in physikalische Blöcke.
- Freiplatz-Manager

- Generische Lese- und Schreibe-Operationen
- Verwaltung von Speicher-Buffers und Caches für Verzeichnisse und Dateiblöcke

Schichten eines Dateisystems



open("Dateiname"); read(pos);

readblock 1 of file 12540
read block at offset 3045

read drive 1, cylinder 20, track 2, sec 2
or read LBA 231230

out 210h, ax

Dateisystem-Architektur

Schichten-Architektur

Vorteile
- Programmcode wird minimiert.
- Mehr Flexibilität

Nachteile
- mehr Overhead innerhalb des Betriebssystems
- Verminderung der Effizienz

Wichtige Entscheidungen sind:

- wie viele Schichten
- Funktionalität innerhalb jeder Schicht

Dateisysteme

Einige Beispiele:

CD-ROMs ISO 9660

Austauschbare Speichermedien

Lokale und verteilte Dateisysteme

UFS UNIX file system

FFS Berkeley Fast File System

FAT, FAT32, VFAT, FAT64 und NTFS Windows NT-Dateisysteme

ext2 und ext3 Linux Extended-Dateisysteme

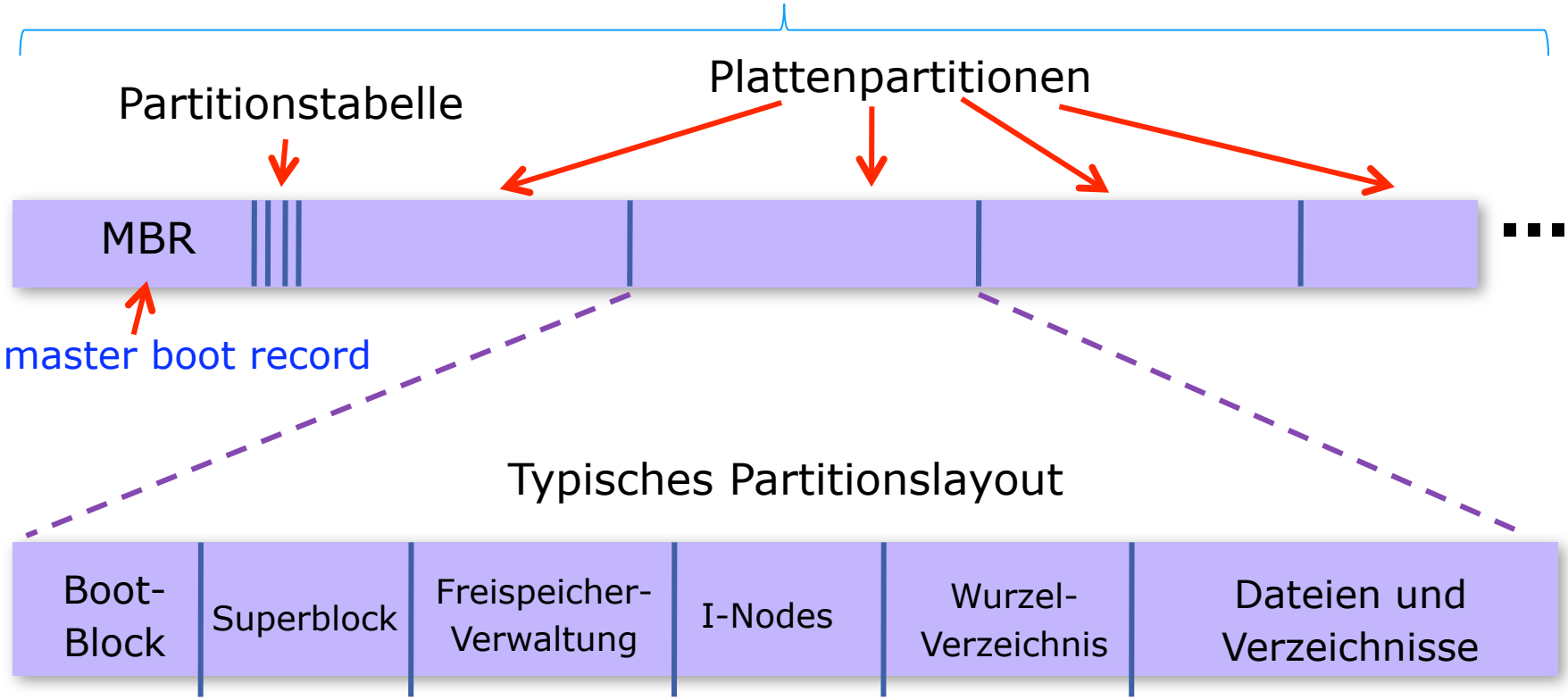
(Verteiltes Dateisystem)

GFS Google File System

Linux unterstützt mehr als 40 Dateisysteme

Dateisystem-Implementierung

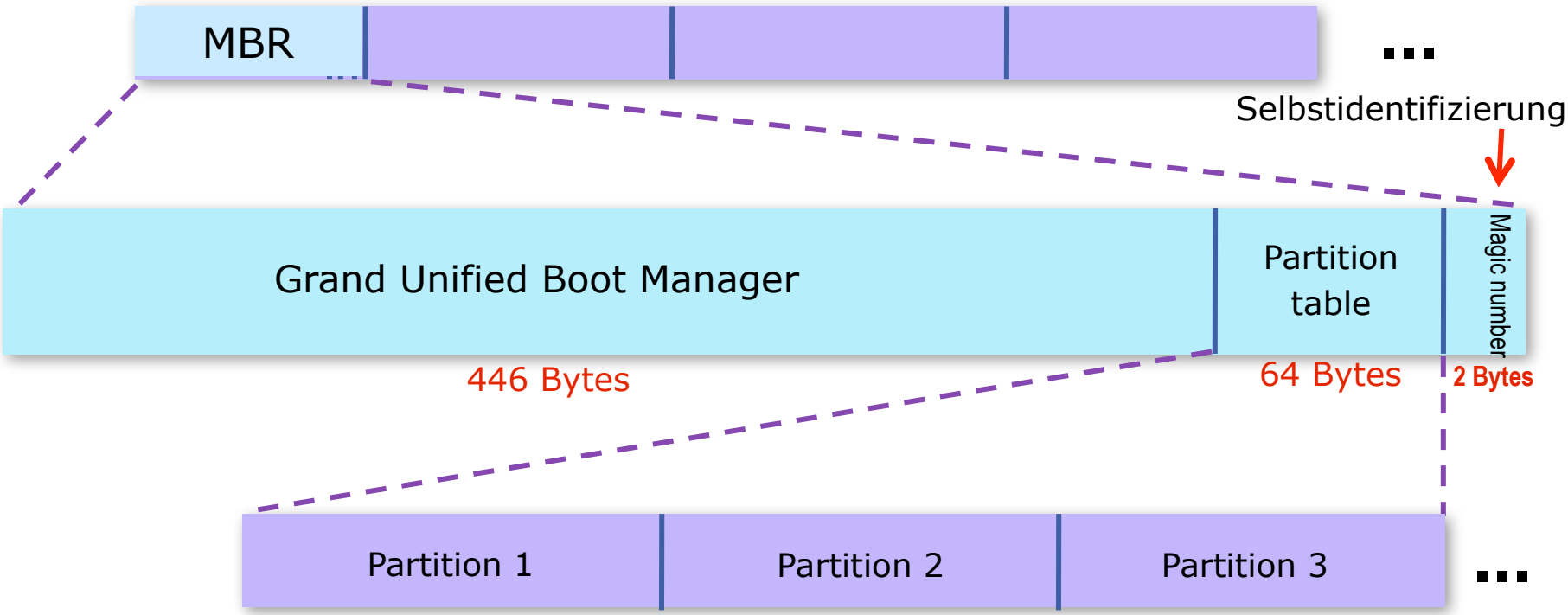
Festplatten-Struktur (Beispiel)



Das Layout der Partitionen kann unterschiedlich sein.

Festplatten-Struktur (Beispiel)

Master Boot Record MBR



MBR Master Boot Record

- Sektor 0
- Im MBR ist die Partitionstabelle, die Anfangs- und Endadresse jeder Partition beinhaltet.
- Eine Partition ist immer als aktiv markiert.
- Wenn der Computer gestartet wird, liest das BIOS den MBR ein und führt ihn aus.
    - Der erste Block der aktiven Partition (Boot-Block) wird gesucht und ausgeführt.
        - Das Programm im Boot-Block lädt das Betriebssystem.
- Jede Partition beginnt immer mit einem Boot-Block, der das eigentliche Betriebssystem lädt.

Superblock oder Volume Control Block

- Wichtige Parameter des Dateisystems sind hier.
- Wird im Speicher geladen, wenn nach Starten des Computers zum ersten Mal auf das Dateisystem zugegriffen wird.
- Hier sind Informationen wie z.B.
  - **Magische Zahl**, um Typ und Größe des Dateisystems sowie andere Verwaltungsinformation zu identifizieren.

Weitere wichtige Information des Dateisystems wie Freispeicher-Verwaltung, I-Nodes, Wurzelverzeichnisse usw. wird je nach Dateisystem unterschiedlich organisiert.

Strukturen des Dateisystems

mount table

Hier sind alle Dateisysteme, die im Betriebssystem aktiv sind.

in-memory directory-structure cache

Information über die vor kurzem verwendeten Verzeichnisse.

system-wide open-file table

Kopie der FCB von jeder Datei, die gerade geöffnet ist.

per-process open-file table

Zeigen auf Einträge im system-wide open-file table, sowie Prozess spezifische Datei-Information.

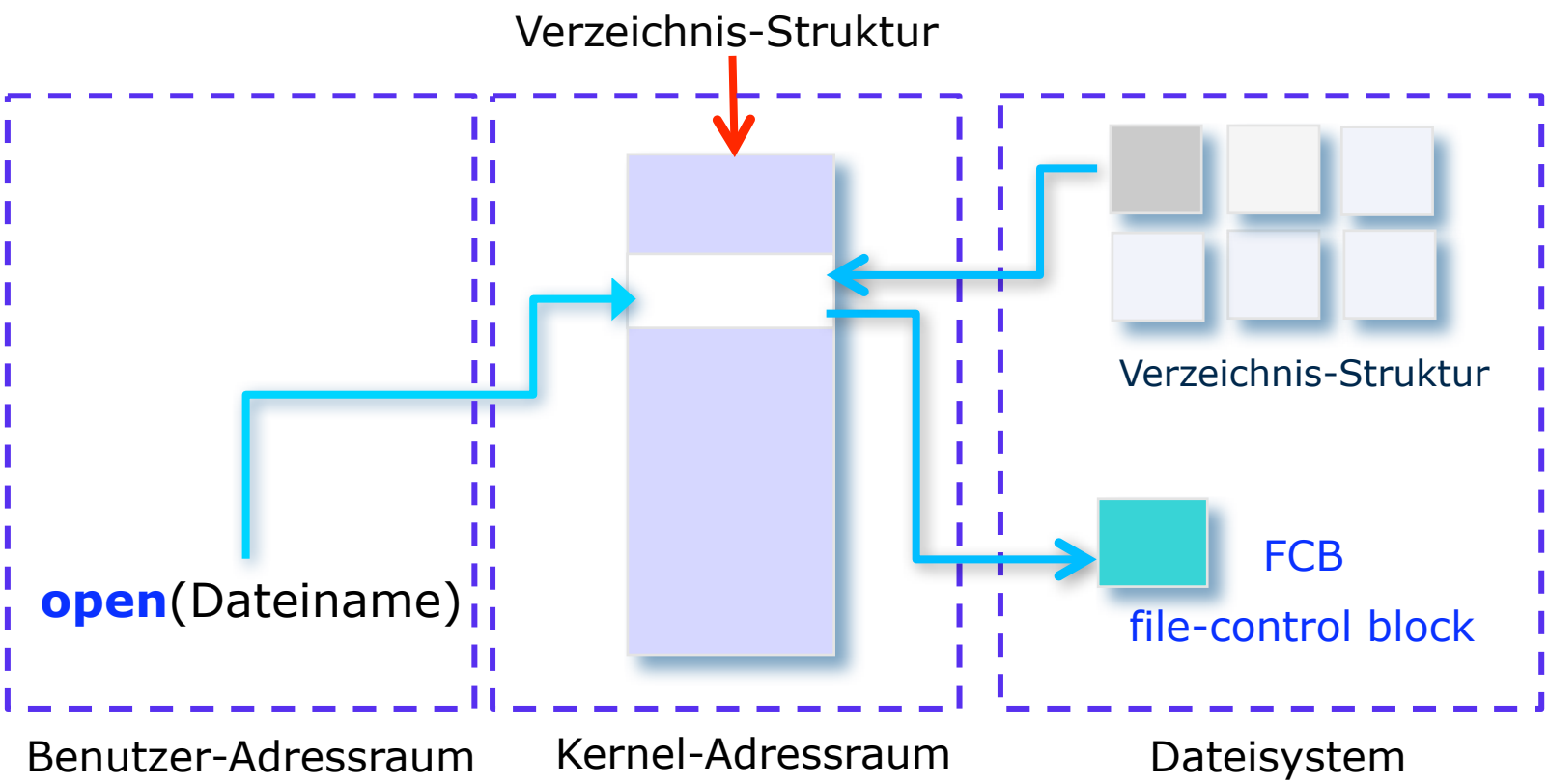
Dateisteuerblock (FCB)

Ein typischer FCB *file-control block* beinhaltet folgende Information:

- Zugriffsrechte
- Datum (create, access, modify)
- Datei (owner, group) ACL (access control list)
- Dateigroße
- Dateiblöcke oder Zeiger auf Dateiblöcken

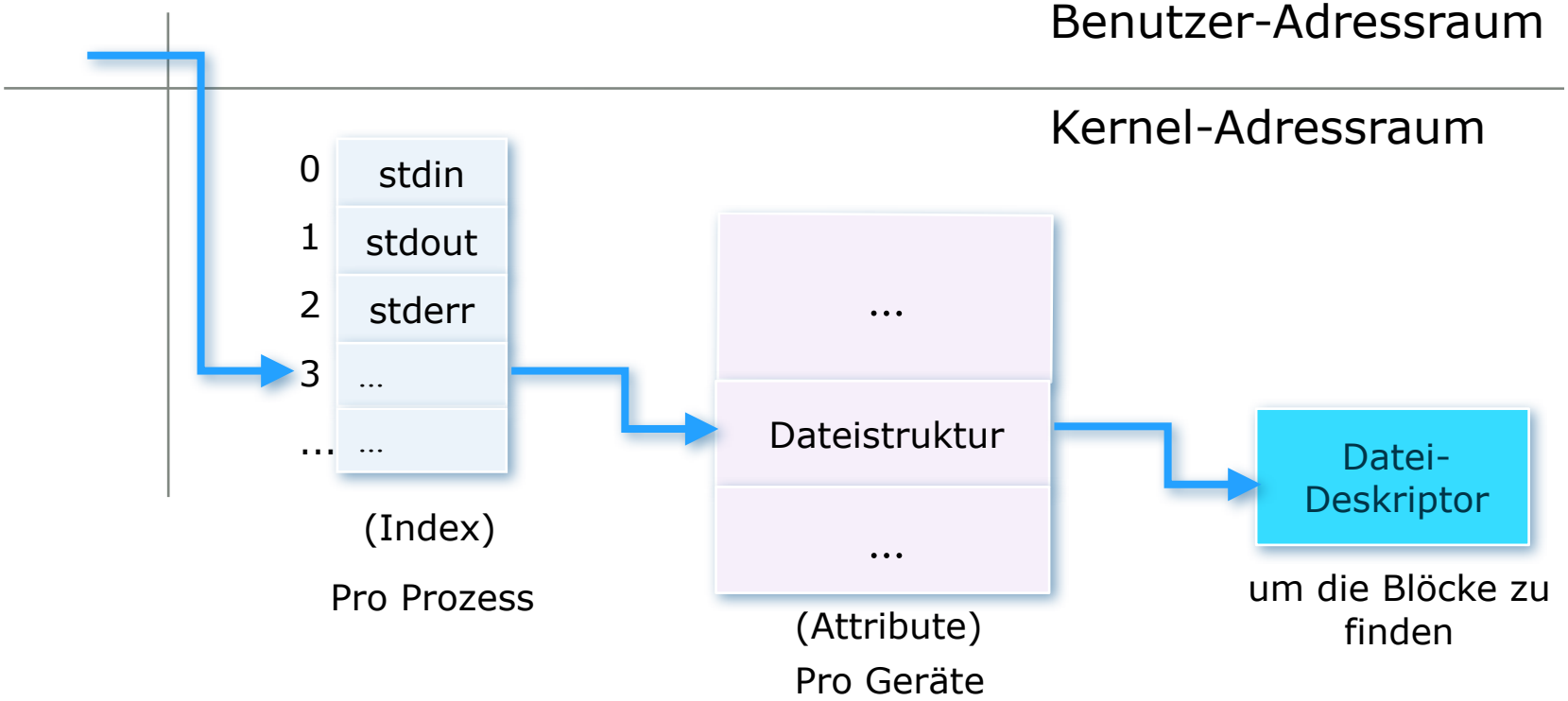
Wenn eine neue Datei erzeugt wird, wird das logische Dateisystem eingeschaltet und eine neue FCB erzeugt. Oft gibt es im Speicher eine Liste freier FCBs, die im voraus erzeugt werden.

Das Öffnen einer Datei

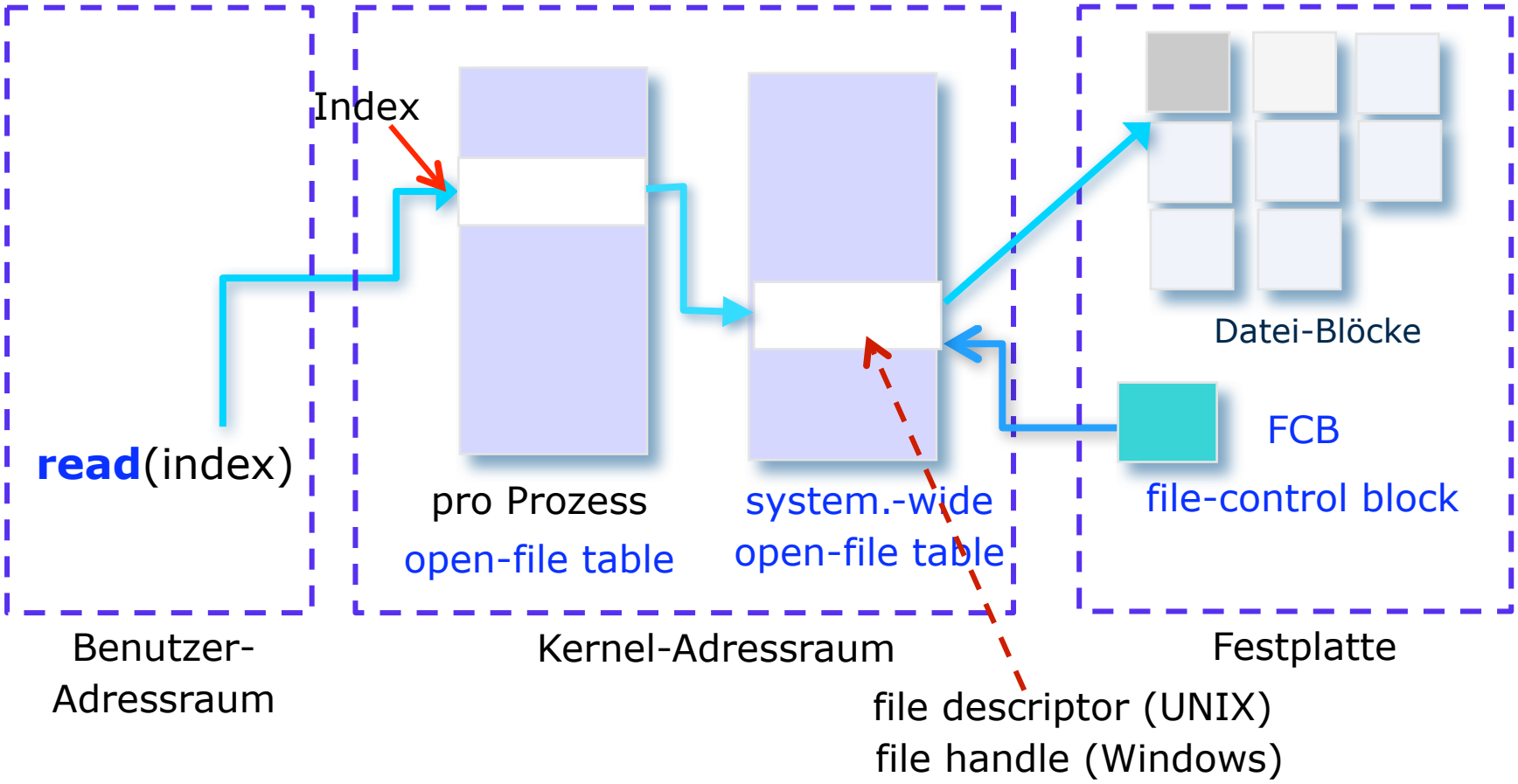


Unix open()

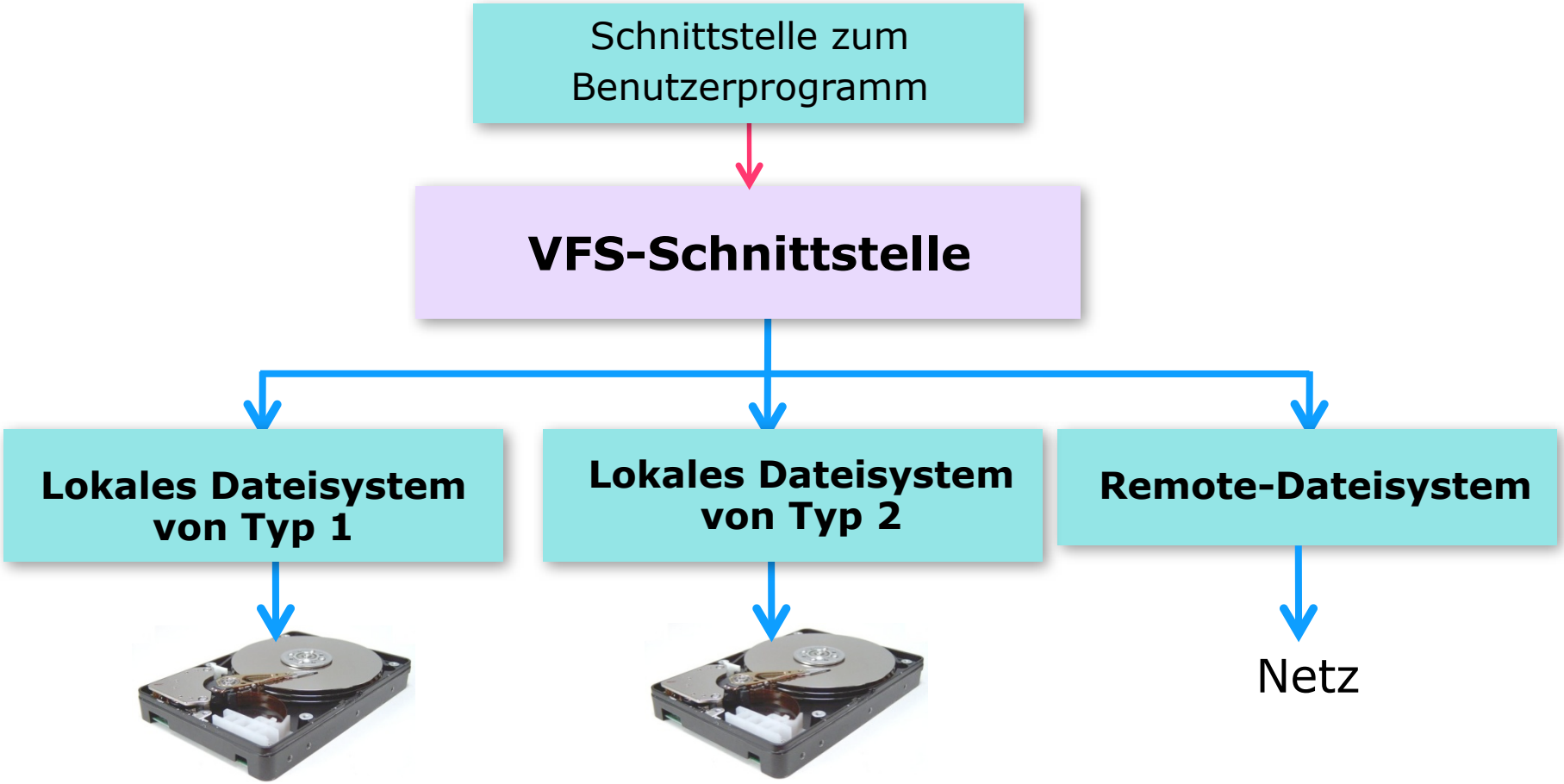
```
int fid = open("Dateiname", flags);
read(fid, …);
```



Das Lesen in einer Datei



Dateisystem



VFS-Architektur in Linux

Wichtige Datenstrukturen sind:

inode-Objekt	Eine Datei
file-Objekt	Eine geöffnete Datei
superblock-Objekt	Ein komplettes Dateisystem
dentry-Objekt	Verzeichnis-Eintrag

Für jedes dieser Objekte definiert das **Linux-VFS** eine Reihe von Operationen (Schnittstelle), die implementiert werden müssen.

Jeder Objekttyp hat einen Verweis auf eine Funktionstabelle.

Die **Funktionstabelle** beinhaltet die Liste der Adressen auf den Funktionen, die die entsprechenden Operationen der Objekte implementieren.

VFS-Architektur in Linux

Hier ist eine gekürzte Version der bekanntesten Funktionen der Linux-API für das **file**-Objekt:

int open (. . .)	öffnet eine Datei
ssize_t read(. . .)	liest aus einer Datei
ssize_t write(. . .)	schreibt in eine Datei
int mmap(. . .)	memory-map von einer Datei

Für jeden neuen Dateisystems-Typ müssen alle Funktionen des Datei-Objekts implementiert werden, die für den ***struct file*-Datentyp** spezifiziert sind.

/usr/include/linux/fs.h

Implementierung von Verzeichnissen

Lineare Liste

Eine lineare Liste von Namen mit Verweisen auf die Blockdaten.

- sehr einfach zu implementieren
- sehr ineffizient
  - O(n) für Such-Operationen

Hashtabelle

Viel effizienter als lineare Listen

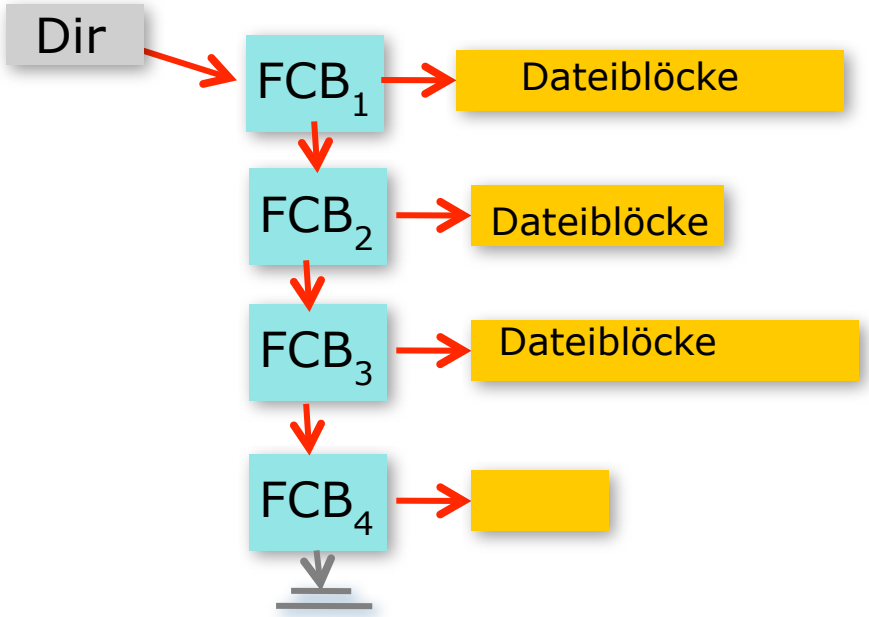
Probleme

- Kollisionen
- Feste Tabellengröße

Eine Alternative sind *chained-overflow hash tables*

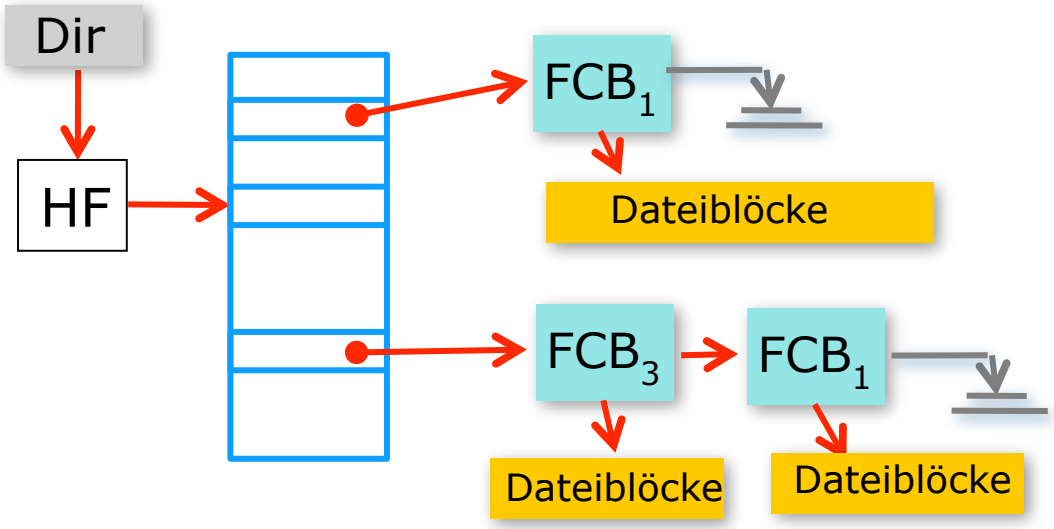
Implementierung von Verzeichnissen

Lineare Liste



Implementierung von Verzeichnissen

Hashtabelle



Allokationsmethoden

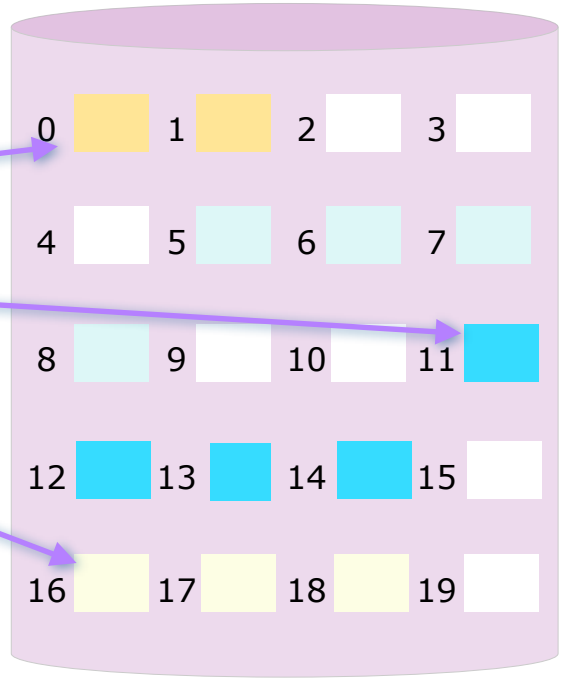
Drei grundlegenden Belegungs-Mechanismen

- Zusammenhängende Belegung
- Belegung durch verkettete Listen
- Belegung durch verkettete Listen mit einer Tabelle im Arbeitsspeicher (Indexed allocation)

Zusammenhängende Belegung

Verzeichnis

Datei	Start	Länge
a	0	2
first	11	4
list	16	3
b	5	4



Festplattenspeicher

Zusammenhängende Belegung

Vorteile

- Einfachste aller Methoden
- Sehr schneller direkter Zugriff auf die Daten
- Für die Lokalisierung der Dateiblöcke brauchen wir nur Anfangsblock und Größe der Datei zu wissen.
- Lese-Operationen können sehr effizient implementiert werden.
- Gute Fehlereingrenzung

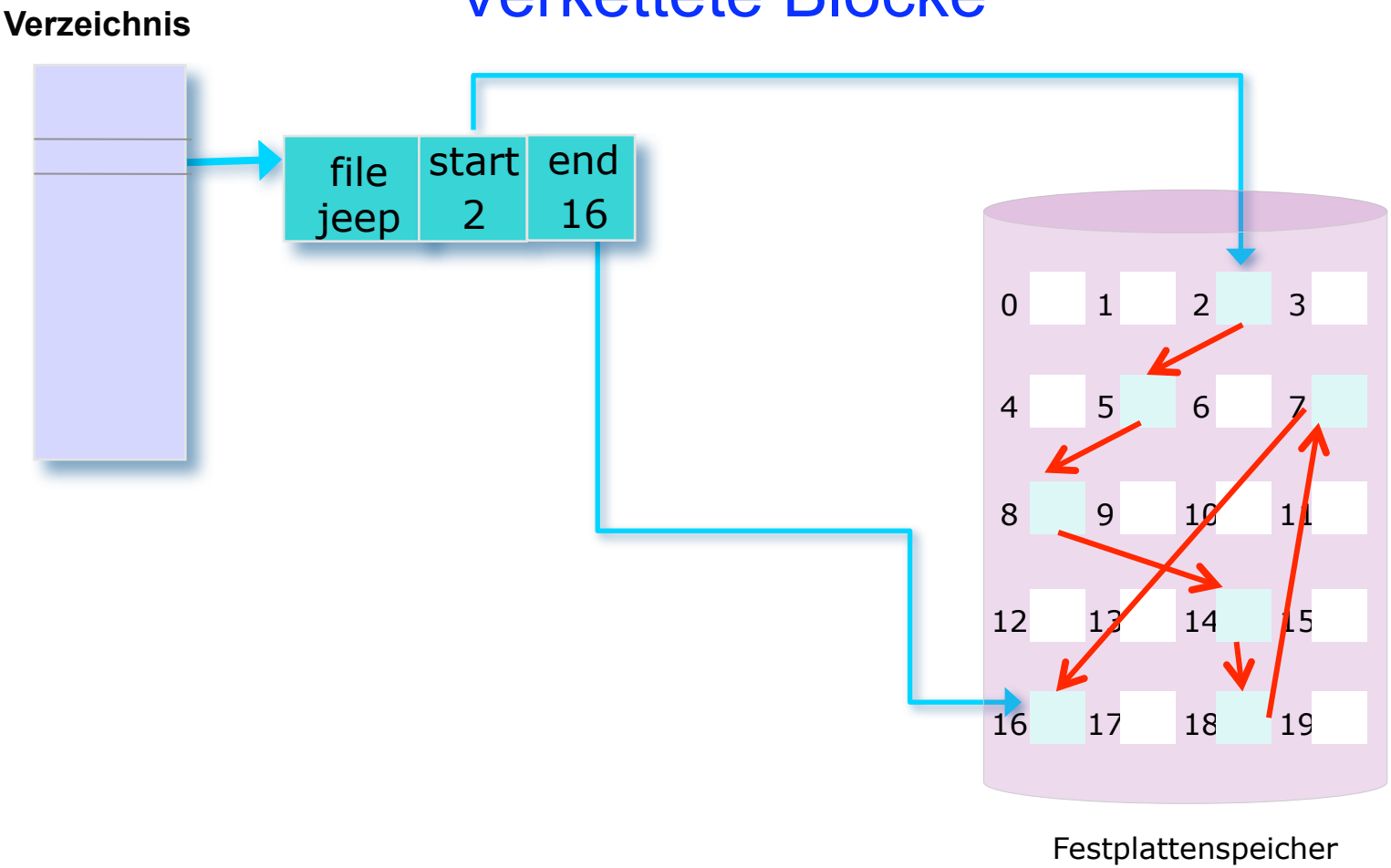
Zusammenhängende Belegung

Probleme

- Dynamische Dateigrößen sind ein Problem
- Im Laufe der Zeit wird die Platte fragmentiert.
- Platz zu finden für neue Dateien ist ein Problem
  - Verwaltung von freien Speicherplätzen notwendig
- Regelmäßige Kompaktifizierung notwendig
  - Platte hin- und zurück kopieren
  - On-line Defragmentierung

Modifizierte zusammenhängende Belegung mit extends (UNIX UFS)

Verkettete Blöcke



Verkettete Blöcke

Jede Datei wird als verkettete Liste von Plattenblöcken gespeichert.

Nur die Plattenadresse des ersten und letzten Blocks wird in dem Verzeichniseintrag gespeichert.

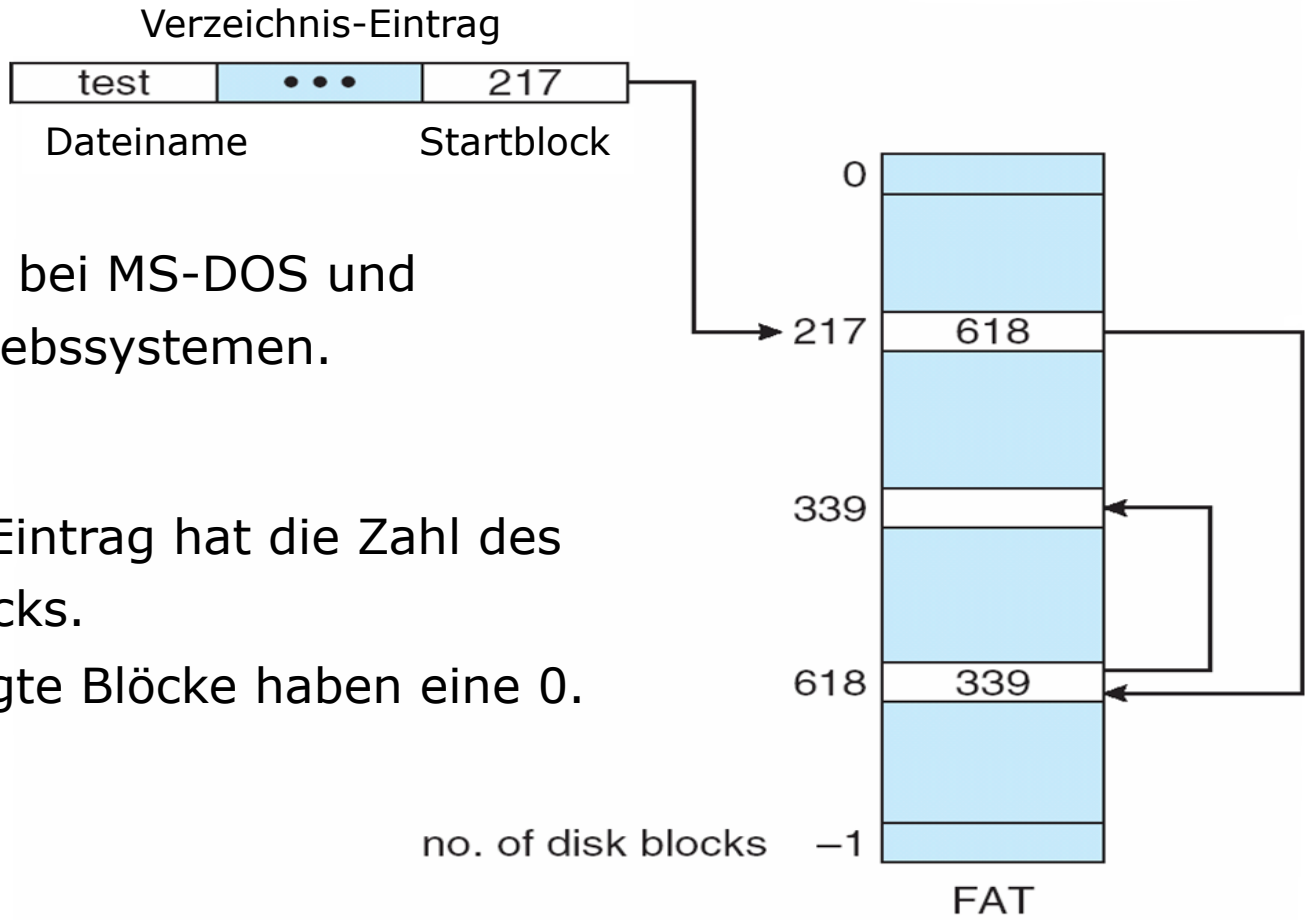
Vorteile

- Keine externe Fragmentierung
- Sequenzieller Zugriff ist kein Problem

Probleme

- Schlechter wahlfreier Zugriff auf Dateiinhalte
- Jeder Verweis verursacht einen neuen Plattenzugriff
- Overhead für das Speichern der Verkettung
  - Lösung: clusters aus mehren Blöcken)
- Erhöhter Aufwand bei Dateizugriffen
- Schlechte Fehlereingrenzung

File Allocation Table FAT



Verwendet bei MS-DOS und OS/2 Betriebssystemen.

Ein Datei-Eintrag hat die Zahl des ersten Blocks.

Nicht belegte Blöcke haben eine 0.

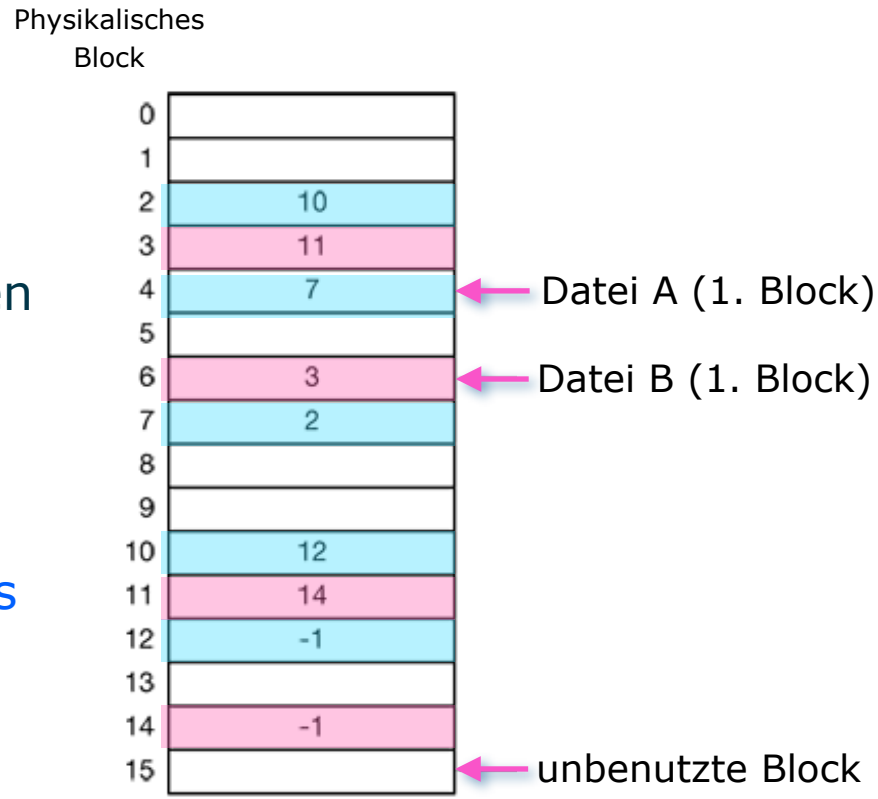
File Allocation Table FAT

Vorteile

- Dateien können sehr leicht und effizient vergrößert werden

Nachteile

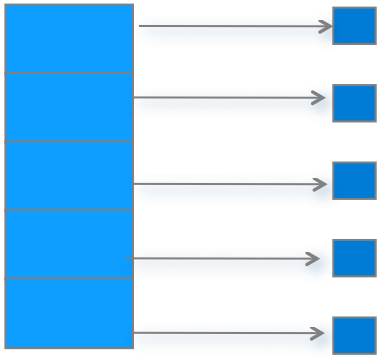
- Interne Fragmentierung
- schlecht für random accesses
- fehleranfällig



Indizierte Tabellen

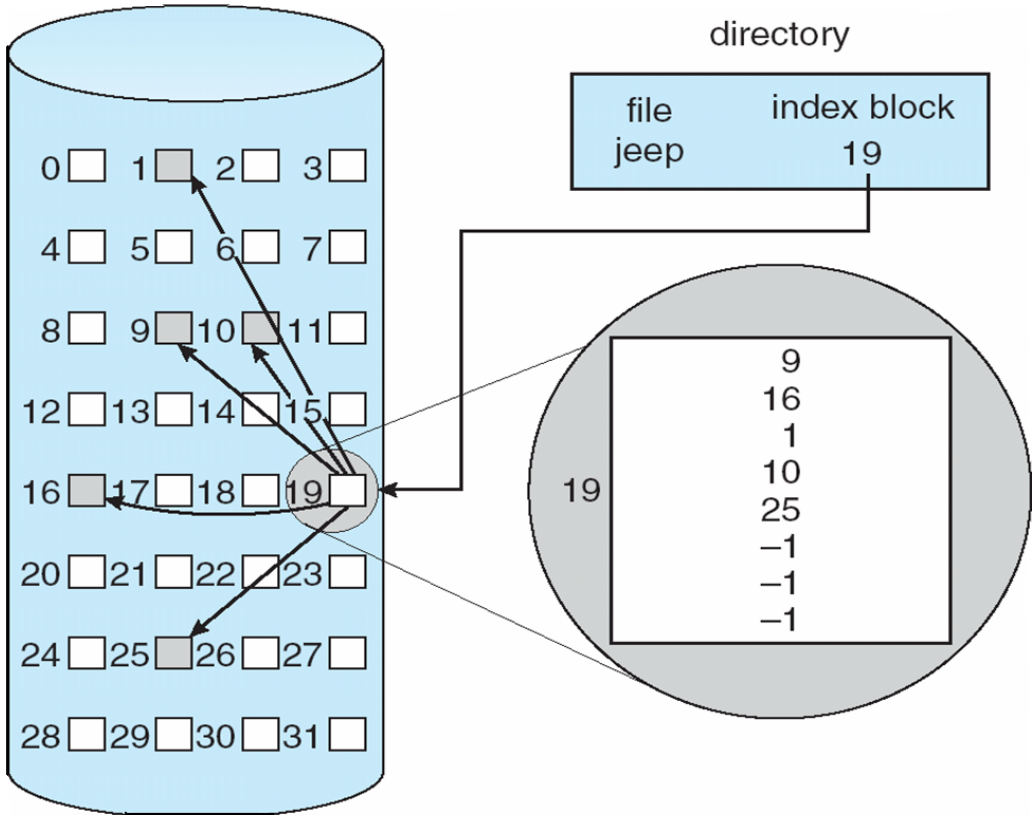
- jede Datei hat einen Indexblock
- wahlfreier Zugriff ist möglich
- keine externe Fragmentierung
- dynamischer Zugriff
- Speicher-Overhead für die zusätzlichen Indextabellen
- der Index kann im Cache gespeichert werden

Indextabelle



Indizierte Tabelle

Beispiel:



Bildquelle: Silverschatz, Galvin, Gagne

Verkettete Listen mit indizierter Tabelle

Fragen

Wie groß soll ein Indexblock sein?

- zu groß-> Speicher-Overhead
- zu klein -> Probleme mit großen Dateien

Lösungen

Linked scheme

- Mehrere Indexblocks werden verkettet

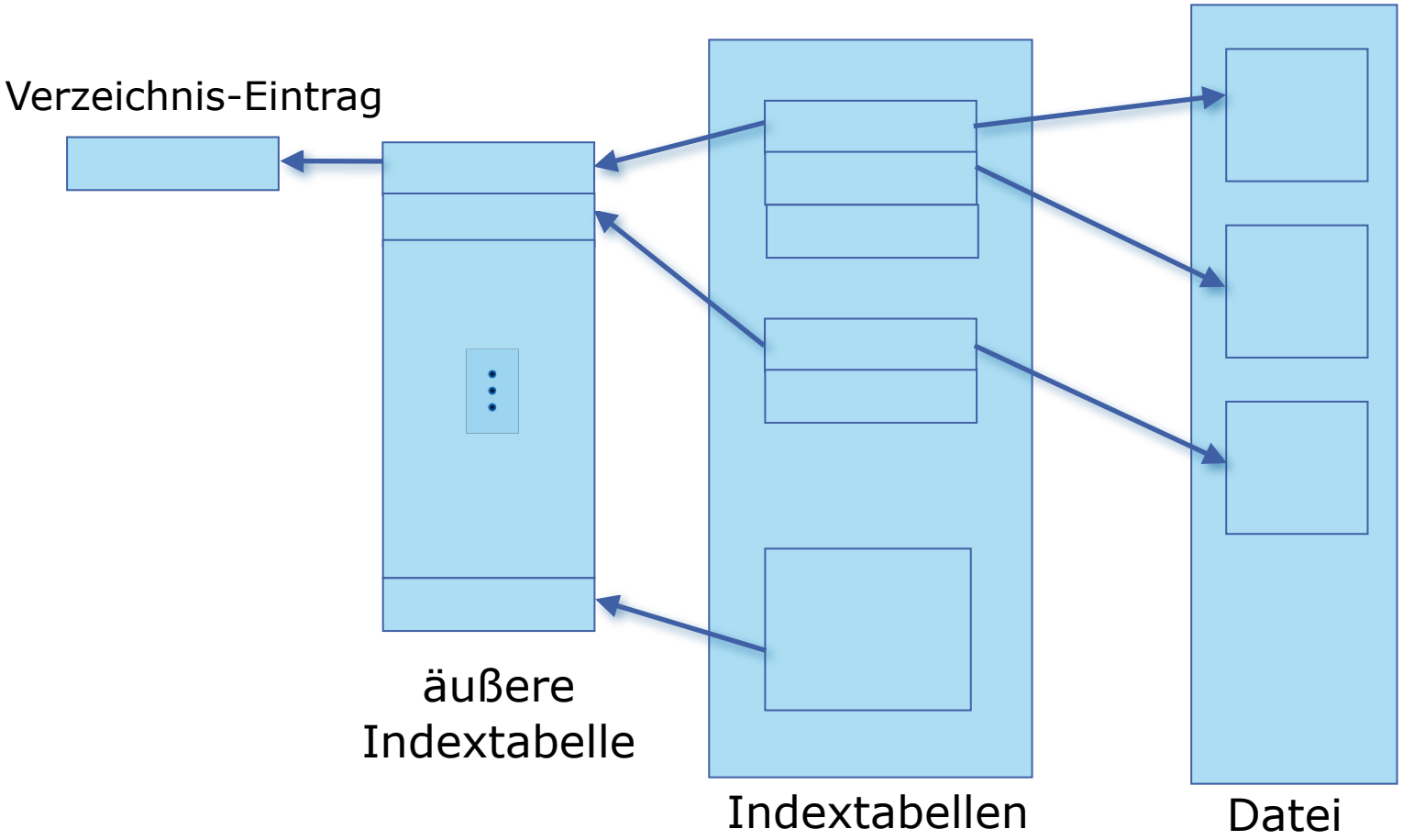
Multilevel index

- Zweistufige Indexblocktabelle wird verwendet

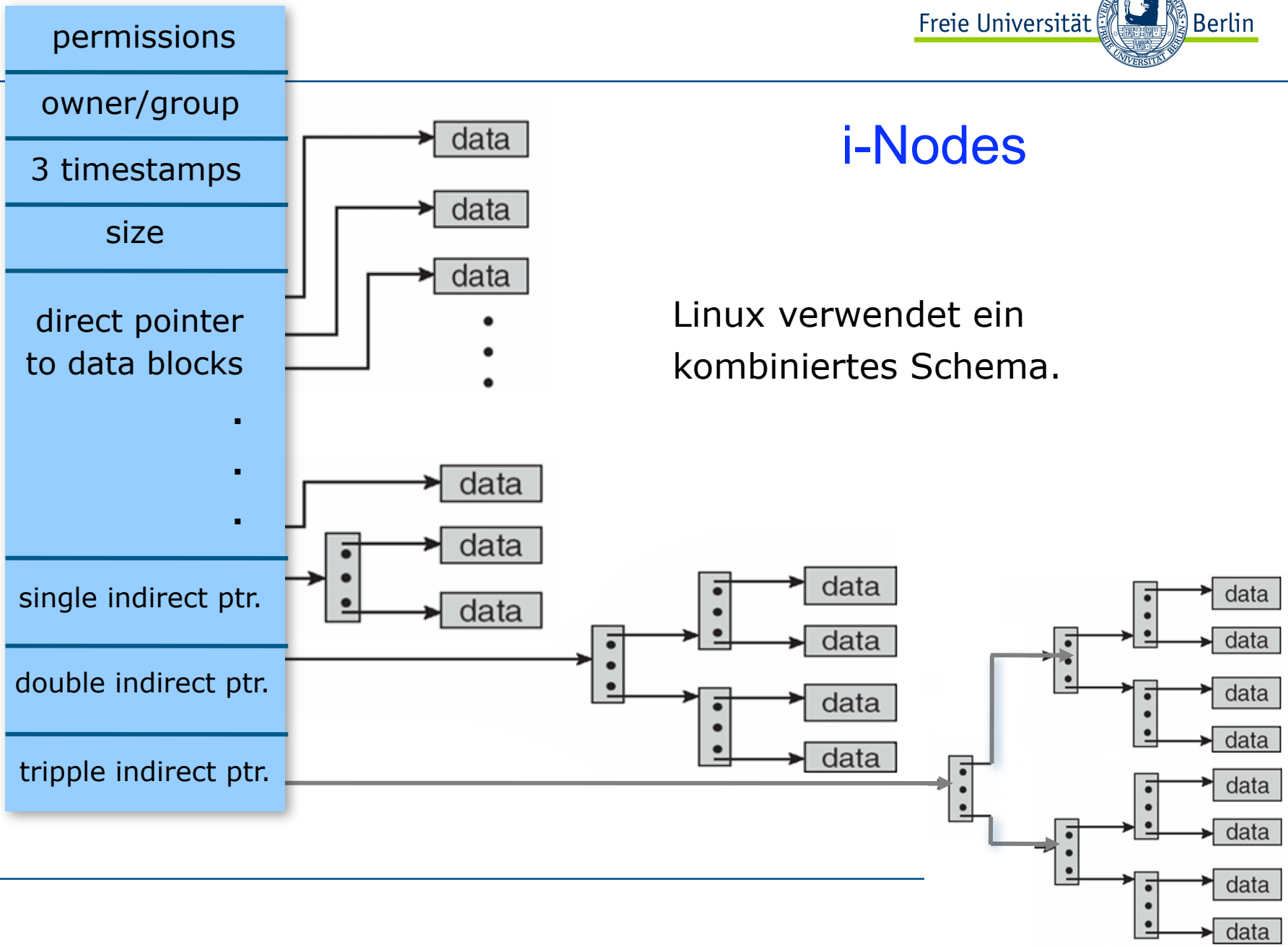
Combined scheme

- Für kleine Dateien wird nur eine einfache Indextabelle verwendet. Wenn aber die Dateien groß werden, verwandelt sie sich in eine mehrstufige Indextabelle.

Multilevel index



i-Nodes



Linux verwendet ein kombiniertes Schema.

i-Nodes

Vorteile

- schneller Zugriff für kleine Dateien
- keine externe Fragmentierung

Probleme

- Interne Fragmentierung
- Die maximale Dateigröße ist begrenzt

i-Nodes

ext2-Dateisystem-Grenzen im Linux

Blockgröße:	1 kB	2 kB	4 kB	8 kB
max. Dateigröße:	16 GB	256 GB	2048 GB	2048 GB
max. Dateisystemgröße:	2047 GB	8192 GB	16384 GB	32768 GB

Das Dateisystem begrenzt die Anzahl von Unterverzeichnissen in einem gegebenen Verzeichnis auf 32.768 Stücke.

Linux/Unix-Dateisysteme

Dateisystem	maximale Dateigröße	maximale Dateisystemgröße
Ext4	16 TByte	1024 PByte
Ext3	2 TByte	16 TByte
JFS	4 PByte	32 PByte
ReiserFS 3	8 TByte	16 TByte
XFS	8192 PByte	8192 PByte
ZFS (Solaris)	16.384 PByte	16.384 PByte

Verwaltung des freien Speichers

- Bit-Vektor
- Verkettete Listen
- Grouping
  - Freie Blöcke werden in Indexblöcken zusammengefasst
  - Der letzte Block hat einen Zeiger auf den nächsten Indexblock
- Counting
  - Zeiger auf zusammenhängende Blöcke + Anzahl der Blöcke wird gespeichert
- Space Maps Sun ZFS Dateisystem

Verwaltung des freien Speichers

Bit-Vektor 00001110111 … 00000011

0 ⇒ block[i] ist frei

1 ⇒ block[i] ist belegt

- Vereinfacht das Erzeugen von Dateien aus zusammenhängenden Blöcken.
- Zusätzlicher Speicher auf der Festplatte wird benötigt.

Beispiel

Blockgröße = 4096 Bytes

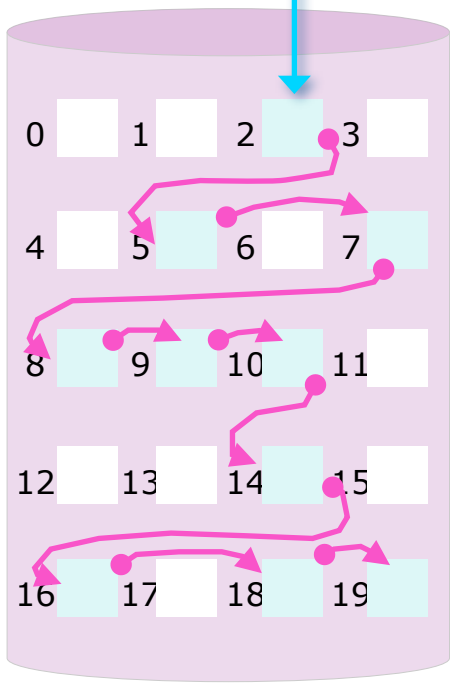
Partitionsgröße = 20 Gigabytes

Bit-Vektor = 640 KBytes

Verwaltung des freien Speichers

Kopfzeiger der Liste von freie Blöcken

- Keine Verschwendung von Speicherplatz. Die Verkettung findet in den freien Blöcken selbst statt.
- Schwierig, zusammenhängenden Speicherbereich zu finden.



Festplattenspeicher

Lokalität?

ab hier!

Verwaltung von freiem Platz

Grouping

- Eine Verbesserung der Verwaltung mit verketteten Listen.
- Speichert die Adressen von **n** freien Blöcken (Indextabelle) im ersten freien Block. Im Block **n** befindet sich die Adresse der nächsten Gruppe von Blöcken.
- Eine große Anzahl von freien Blöcken kann schnell gefunden werden.

Verwaltung von freiem Platz

Counting

- Speichert die Anfangsadresse und Größe von **n** zusammenhängenden Blöcken.
- Die Einträge können anstatt in einer Liste in einem **B**-Baum gespeichert werden.

Verwaltung von freiem Platz

Space Maps

- Sun's ZFS – Dateisystem
- große Dateimengen, Verzeichnisse und Dateisystem-Hierarchien
- verwendet *metaslabs*
- eine Platte kann mehrere hundert *metaslabs* haben
- jedes *metaslab* wird einem *space map* zugeordnet
- der *space map* ist einer log-Struktur, die die gesamte Aktivität in zeitlicher Reihenfolge speichert
- bei *alloc-* oder *dealloc*-Operationen in einer *metaslab,* wird der *space map* als balancierter Baum geladen, indexiert beim *offset.* Die log-File wird in der Baumstruktur gespielt
- komprimiert zusammenhängende freie Blöcke der *space map*
- zum Schluss wird die Liste freier Bereiche der Festplatte als Teil der Transaktion aktualisiert