Betriebssysteme

Kapitel 7: Files

7.2: Implementierung

Stand: WS 08/09

Prof. Dr. Wolfgang Küchlin

Dipl.-Inform., Dr. sc. techn. (ETH)

Arbeitsbereich Symbolisches Rechnen
Wilhelm-Schickard-Institut für Informatik
Fakultät für Informations- und Kognitionswissenschaften

Universität Tübingen

Steinbeis Transferzentrum
Objekt- und Internet-Technologien (OIT)

Wolfgang.Kuechlin@uni-tuebingen.de
http://www-sr.informatik.uni-tuebingen.de

SR

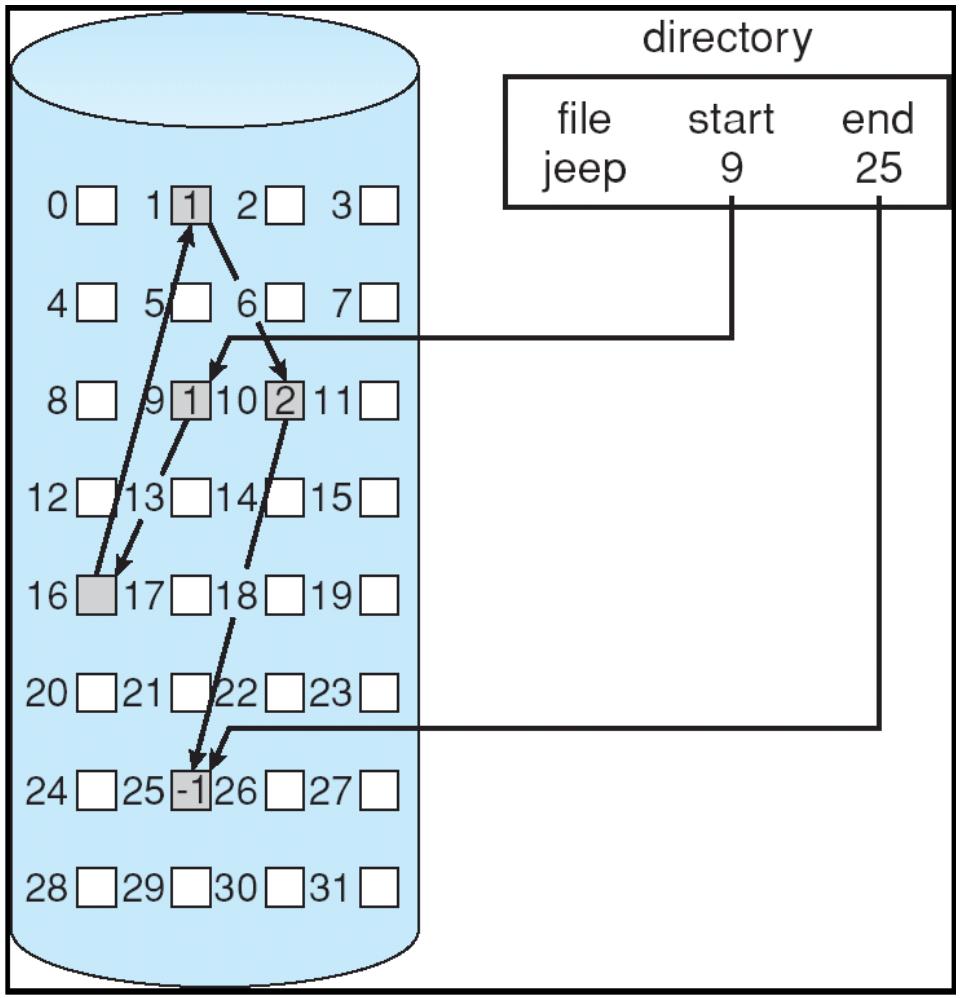
Implementierung von Dateisystemen

- Einfachste Möglichkeit:
  Kontinuierliche Speicherzuweisung.
- Jede Datei besteht aus einer Menge von benachbarten Plattenblöcken.
- Vorteile:
  - Einfache Organisation (nur Startblock und Länge des Files muss bekannt sein).
  - Einfach Abbildung von logischer auf physische Adresse möglich.
  - Mehr oder weniger wahlfreier Zugriff möglich.
- Nachteile:
  - Datei kann nicht wachsen!
  - Verschwendung von Platz (zwischen den Dateien = „externe Fragmentierung“). (Problem dynamischer Platzzuweisung.)

Speicherzuweisung via verketteter Liste

- Jedes File besteht aus einer verketteten Liste von Blöcken auf der Platte.
- Blöcke können beliebig auf Platte verteilt sein.
- Jeder Block enthält Zeiger auf den nächsten Block.
- Dieses Schema wird beim FAT (**file allocation table**) in MS-DOS, OS/2 und Windows 95 benutzt.

Das Dateisystem



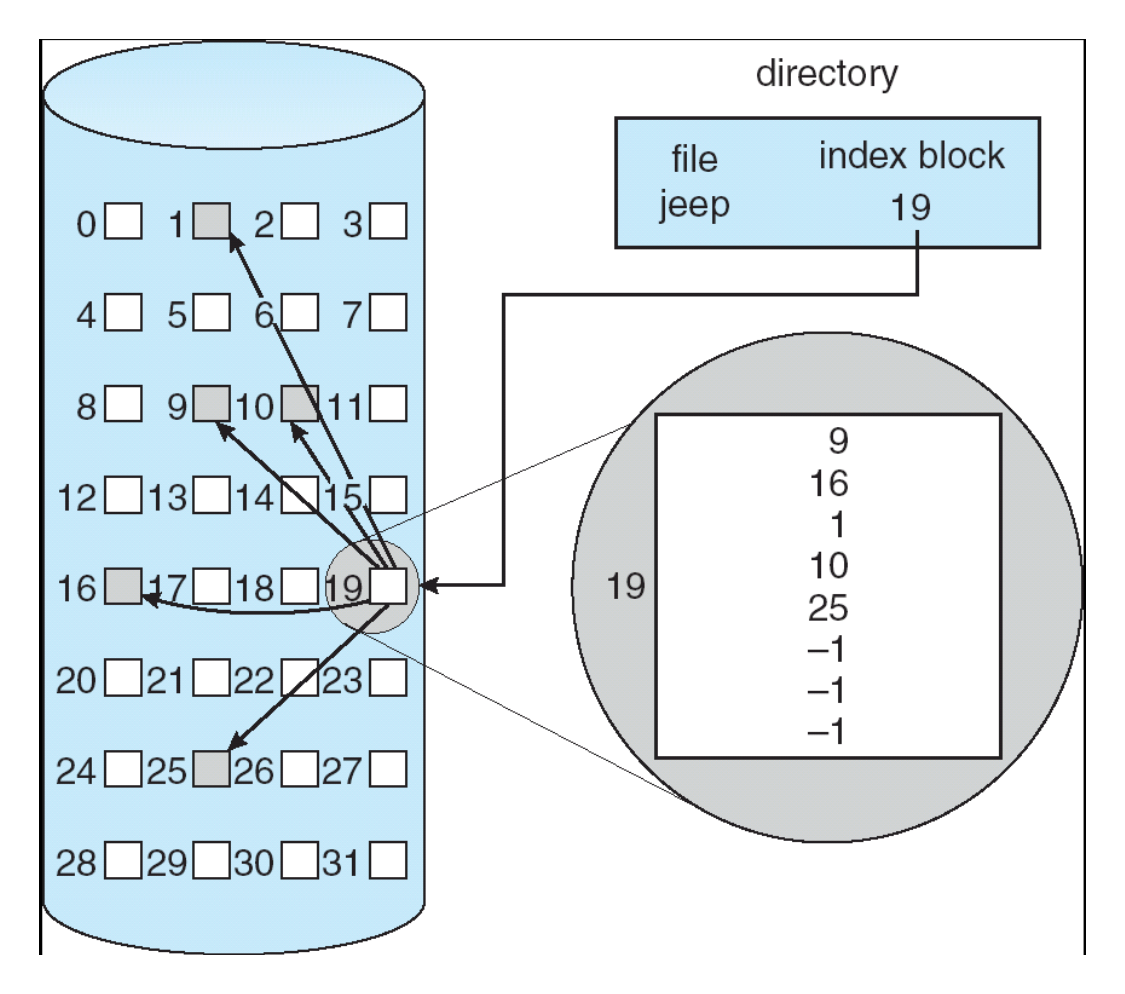
Allocated as needed,
linked together;
e.g. file starts at block 9

Silberschatz, Galvin and Gagne, Abb.11.6

Indizierte Allozierung

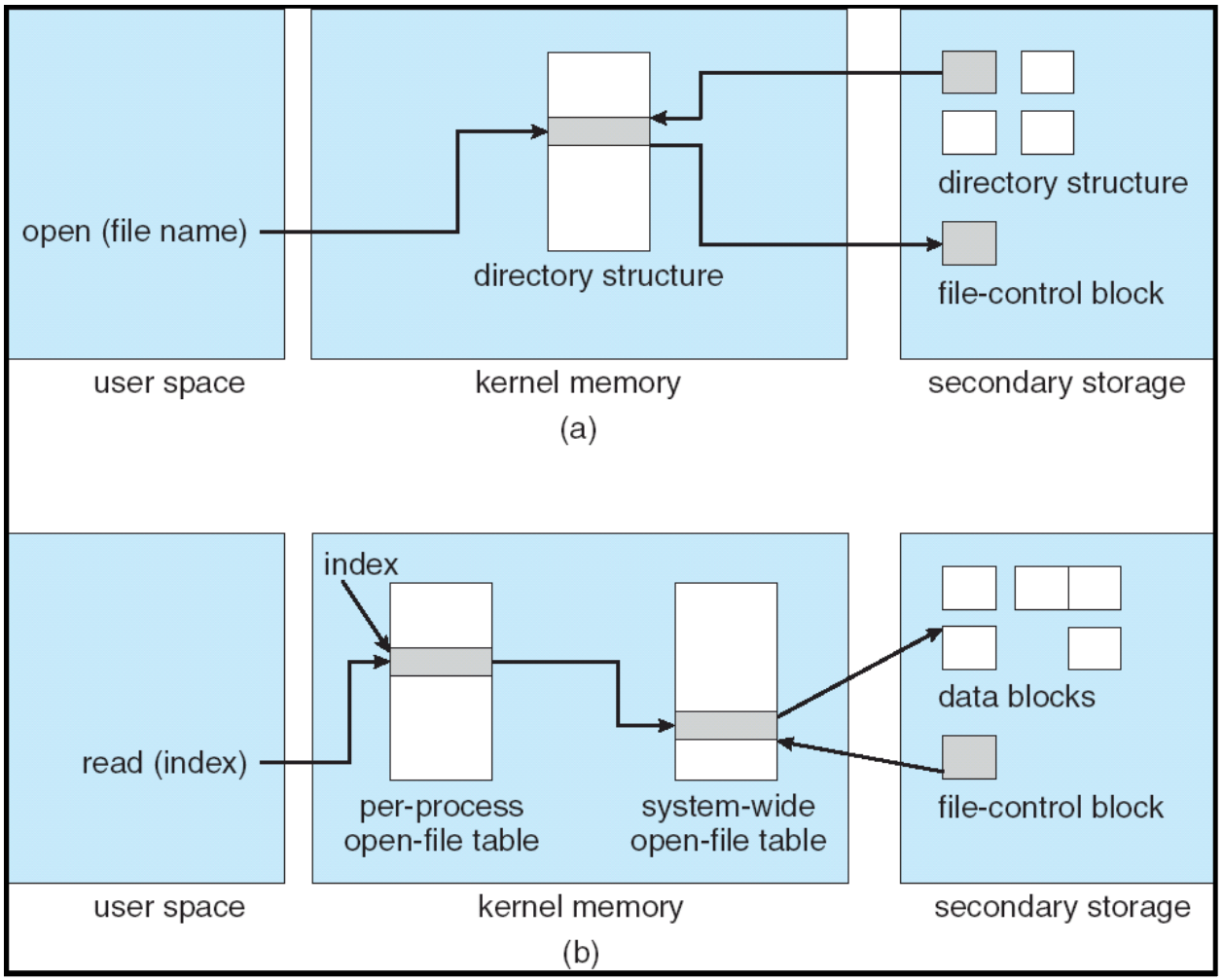
- Bei indizierter Allozierung werden alle Pointer in einem **Index Block** zusammengebracht.
- Indizierte Allozierung kann auch auf mehrere Level verteilt werden, und es kann auch eine Kombination geben.
  - (Schema der direkten Indizierung, indirekte, doppelt indirekte, . . . Indizierung vereinigt, wie z.B. bei UNIX.)

Beispiel indizierte Allocation



Silberschatz, Galvin and Gagne, Abb.11.8

Indexed Allocation - Mapping

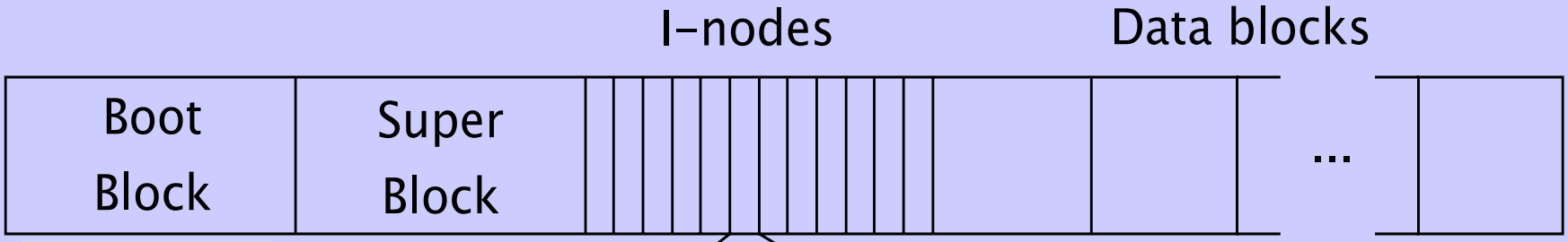


open („filename“):
- durchsuche directory structure, lade sie dazu in den kernel
- kopiere gefundenen FCB in den Kern mit Verweis auf Disk-FCB

read (index):
- vom Index (fd) aus finde FCB und hole ihn von disk in kernel
- Aus read-position berechne gesuchten data-block
- hole data block von disk

Silberschatz, Galvin and Gagne, Abb.11.3

UNIX Filesystem: Plattenorganisation



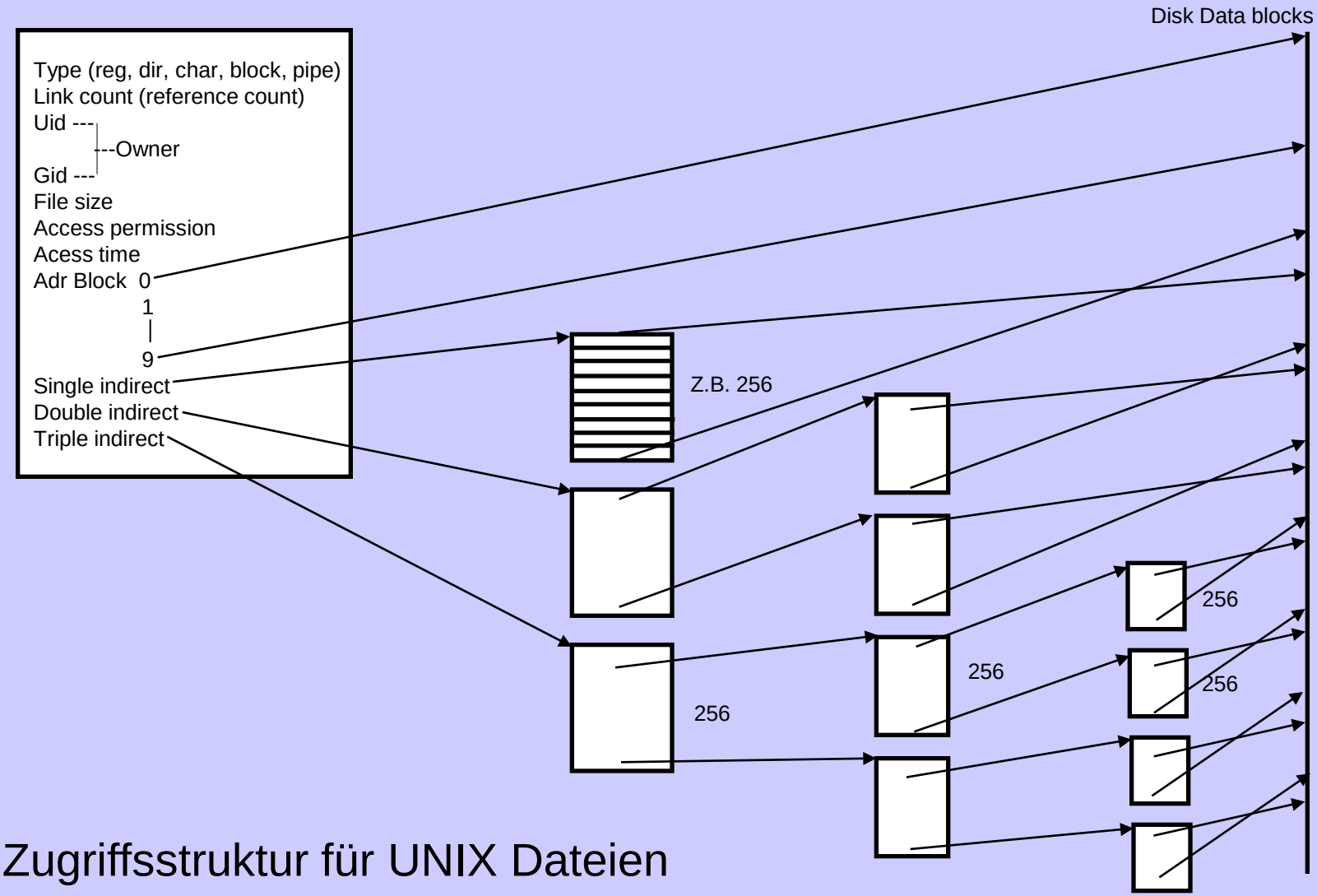
8 KB space for bootstrap program
Unused by UNIX FS

1

Verwaltung System:
- Wie viele I-nodes?
- Wie viele data blocks?
- Freiliste der blocks.
- Freiliste der I-nodes.

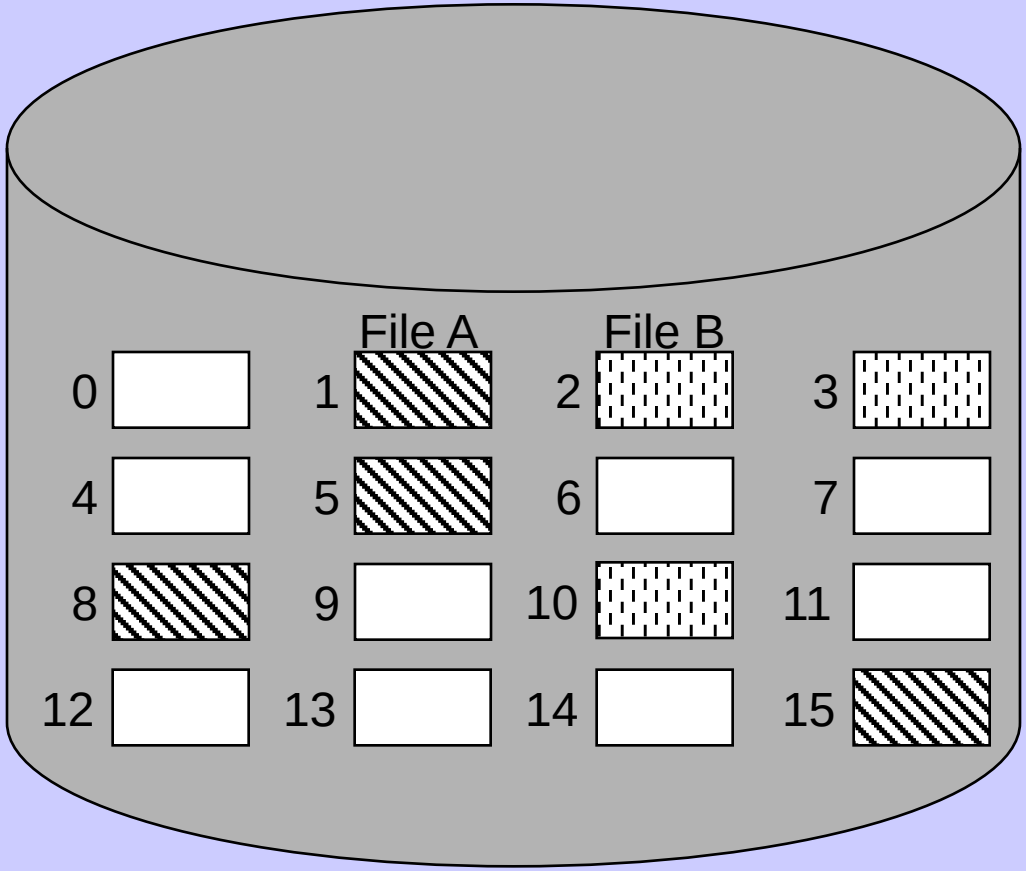
Verwaltung Datei
- Zugriffsrechte
- Eigentümer
- Adressen Dateiblöcke
- #2: root dir.
- #1: bad block handling

UNIX: Organisation von I-node und Datei



Zugriffsstruktur für UNIX Dateien

Organisation von I-node und Datei



I-node File A	
…	
Adr Block0	1
Adr Block1	8
Adr Block2	15
Adr Block3	5
…	

File Datenstrukturen im Kern

- I-nodes existieren sowohl auf Disk als auch im HSP, wenn die Datei offen ist. Oben gezeigt sind die disk I-nodes.
- HSP I-nodes haben zusätzliche Felder, u.a.
  - die Position des zugehörigen disk Inode Platzes
  - ein „dirty bit“
  - sowie Zeiger für die Liste freier I-nodes und
  - Zeiger für Verkettung in der Hash-Tabelle der HSP-Inodes
- Eine Schloß- Variable schützt den I-node während eines Systemaufrufs vor dem Zugriff durch andere Prozesse.
- Das Dateisystem errechnet den gefragten Block aus der Dateiposition (in Byte), dem Dateianfang und der Blockgröße.
  - Mit 32 bit Wort können 4 GB adressiert werden.
  - Mit 1 KB Blöcken und 32 bit Blockadressen können im obigen Schema über 16 GB gespeichert werden.

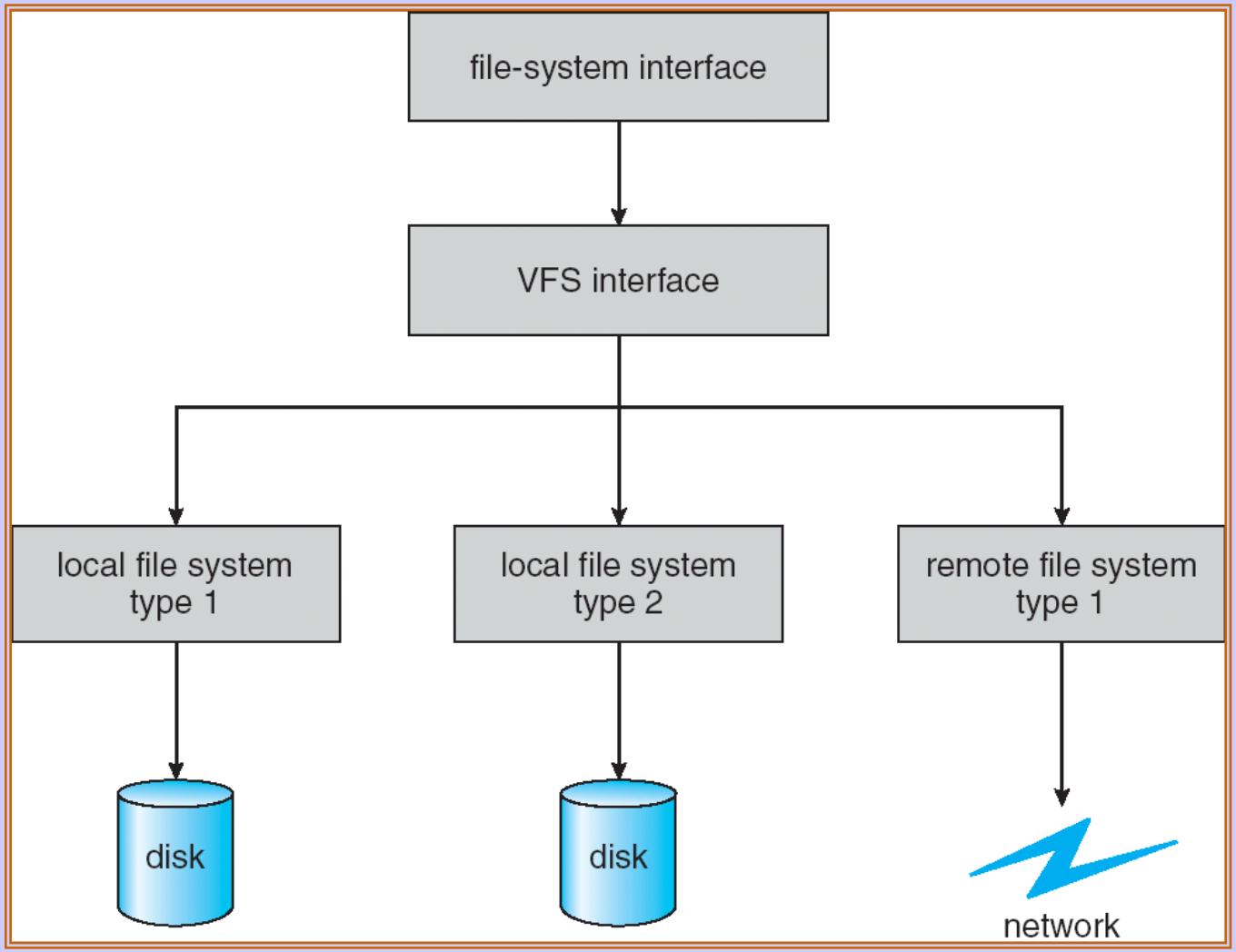
File Datenstrukturen im Kern

- Zusätzlich zur Repräsentation auf der Platte existieren zu jeder benutzten Datei noch weitere Datenstrukturen im Kern, die den Zugriff regeln und erleichtern. Der inode auf der Platte wird in einen Eintrag der Inode-Tabelle im Kern (**in-core inode table**) geladen (**caching**).
- Um Daten auf Platten mit denen im Hauptspeicher zu synchronisieren, gibt es die Möglichkeit zur expliziten Synchronisation: **fsync**. Auch ohne explizite Synchronisation wird regelmäßig der Eintrag auf der Platte mit dem im HS synchronisiert
- Jeder Zugang zur Datei (vgl. **open**) wird durch einen Eintrag in der File-Tabelle des Kerns vermerkt mit Zugangsart (read/write/read-write) und momentaner Lese/Schreibposition (**position**) (vgl. **lseek**).

Virtual File Systems

- Virtual File Systems (VFS) provide an object-oriented way of implementing file systems.
  - VFS provides an interface, which is implemented differently by each file system.
  - VFS provides globally unique file handles across different file systems.
  - the appropriate implementation of **read(filehandle)** is selected according to the type of filehandle
- VFS allows the same system call interface (the API) to be used for different types of file systems.
- The API is to the VFS interface, rather than any specific type of file system.

Schematic View of Virtual File System



Silberschatz, Galvin and Gagne, Abb.11.4

The traditional UNIX File System

- The original UNIX-FS was developed at Bell Labs
  - 1 file system per disk partition
    - typically 150MB total, 4 MB Inodes, 146 MB data
  - blocks of 512 Bytes, file size < 1GB, disk layout see above
  - Inode with 8—13 direct block addr, 1 single, 1 double, 1 triple indirect
    - indirect 512 Byte block contains 128 block addresses (4 Bytes each)
- Problems
  - Long seek from file Inode to data, data blocks not consecutive
  - Transfers of 512 Bytes each, no read ahead
  - No block bulk transfers even for consecutive blocks
  - 6x Performance degradation when free list gets scrambled
    - long seek after each data block transfer
  - Transfer rate 2% of max disk transfer rate (20KB/sec per arm)

The UNIX Fast File System (UNIX FFS)

- UNIX-FFS was introduced for 4.2 BSD
  - M. McKusick, W. Joy, S. Leffler, R. Fabry. A Fast File System for UNIX. ACM Transactions on Computer Systems (ACM ToCS), Vol. 2, No. 3, Aug. 1984, pp. 181-197.
- First Berkeley improvement
  - 1024 Byte data blocks, 2x performance improvement
    - 1024 B transfers, fewer seeks due to fewer indirections (more data in direct blocks)
- Remaining Problems
  - Transfer rate 4% of max disk transfer rate
  - Long seek from file Inode to data, data blocks not consecutive
  - No block bulk transfers even for consecutive blocks
  - Performance degradation when free list gets scrambled
    - From 175 KB/sec down to 30 KB/sec

The Organization of UNIX FFS

- Fault protection
  - Replication of the super block (read-only data)
- Block size 4096 Bytes minimum
  - Access 2^{32} Bytes with only 2 levels indirection
  - block size stored in super-block (configurable at start-up)
  - Blocks divided into 2, 4 or 8 consecutive fragments
- Cylinder groups
  - one or more consecutive cylinders in a partition with
  - copy of superblock
  - own static set of inodes (one per 2048 Bytes data)
  - Bit map of free fragments instead of free list of data blocks
  - Book-keeping info at increasing offset, therefore distributed over the platters of a disk pack (fault protection, hot spot avoidance)

The Organization of UNIX FFS

- Layout optimization for file blocks
  - parameterized by disk geometry and speed
  - optimized placement of consecutive blocks
- Performance improvement up to 10x compared to old FS
- Functional enhancements
  - long file names (up to 255 Bytes)
  - file locking (shared / exclusive locks, file granularity)
  - symbolic links
    - symbolic link is separate file containing arbitrary pathname to be followed
    - hard link is reference to specific inode on the same file system
  - Quotas on inodes and disk blocks per user