

Betriebssysteme

Kapitel 7: Files

7.2: Implementierung

Stand: WS 08/09

Prof. Dr. Wolfgang Kuchlin

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>**



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.)

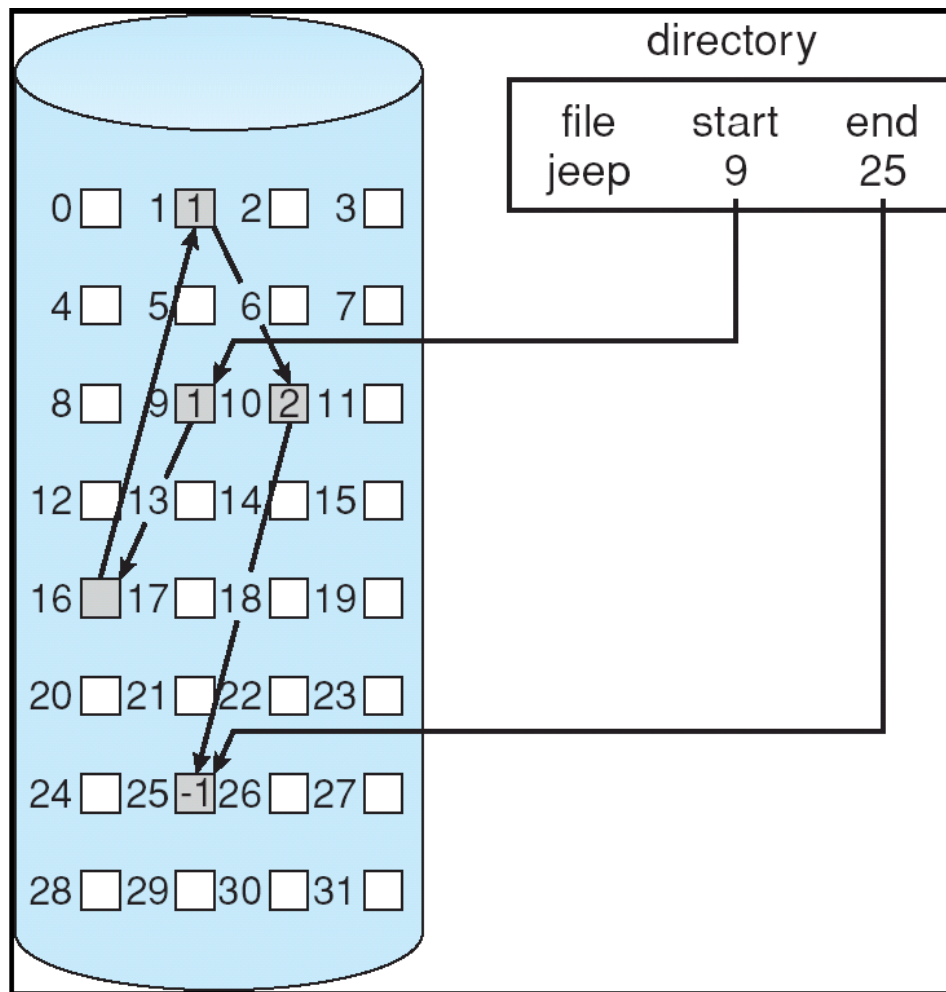


Speicherzuweisung via verketteter Liste

- 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

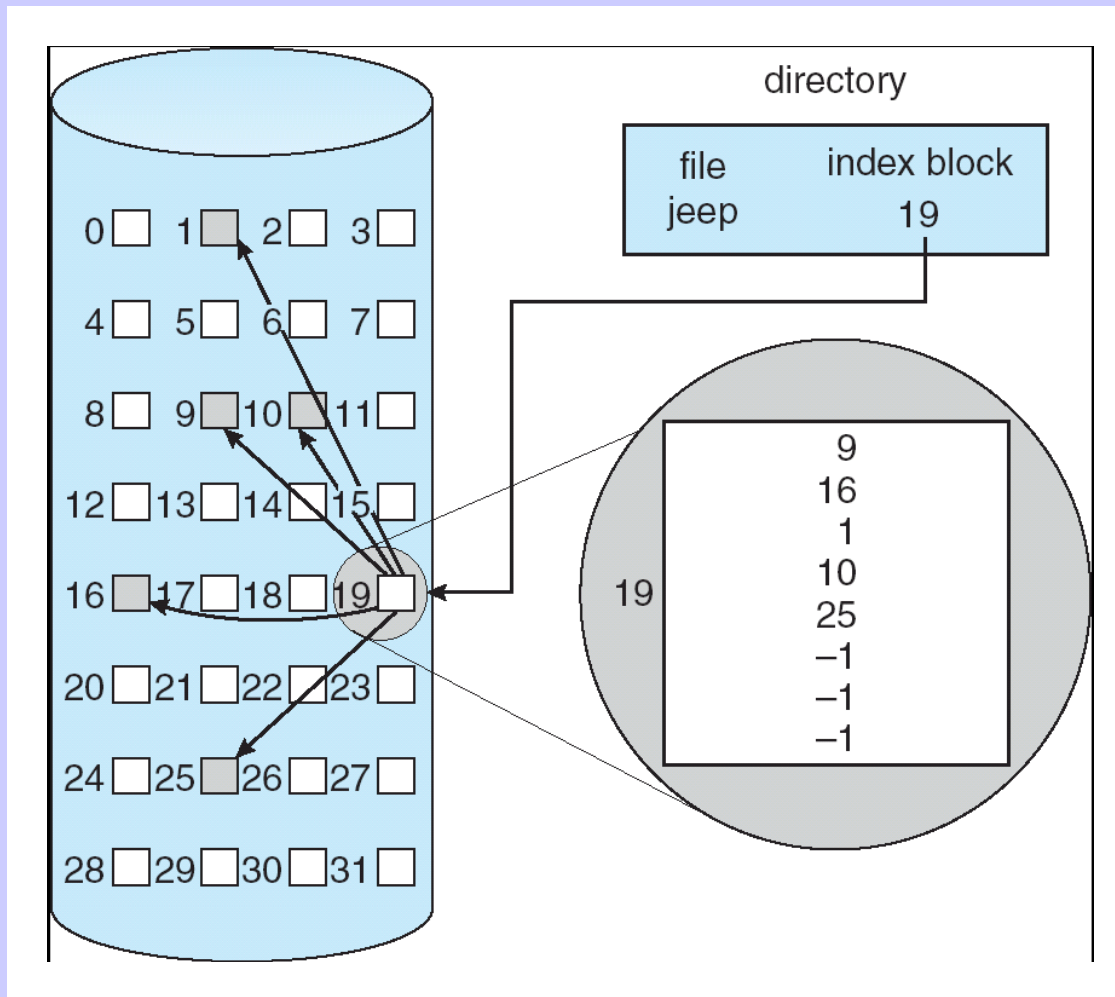


Indizierte Allokierung

- Bei indizierter Allokierung werden alle Pointer in einem **Index Block** zusammengebracht.
- Indizierte Allokierung 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.)



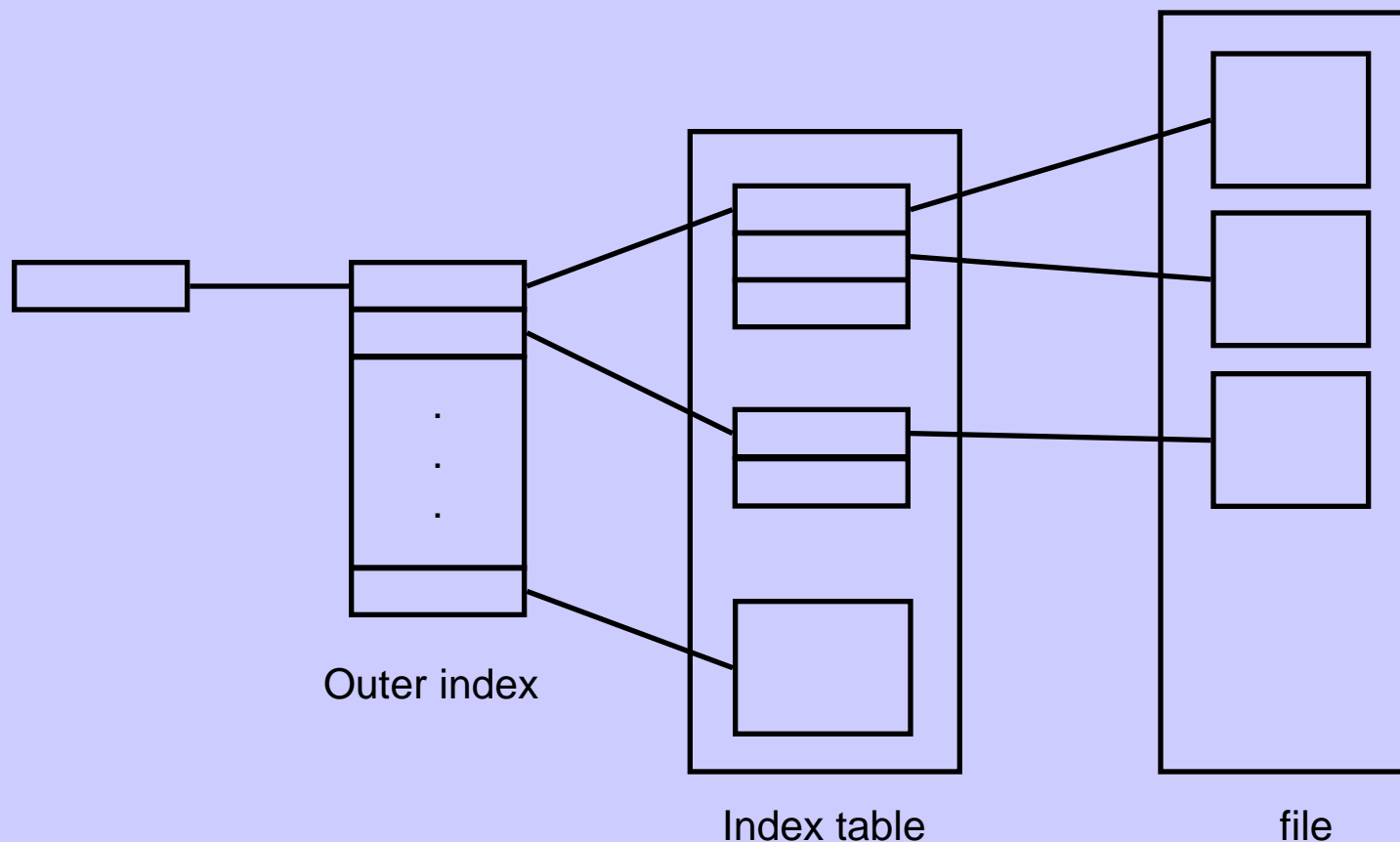
Beispiel indizierte Allocation



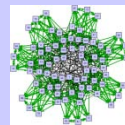
Silberschatz, Galvin and Gagne, Abb.11.8



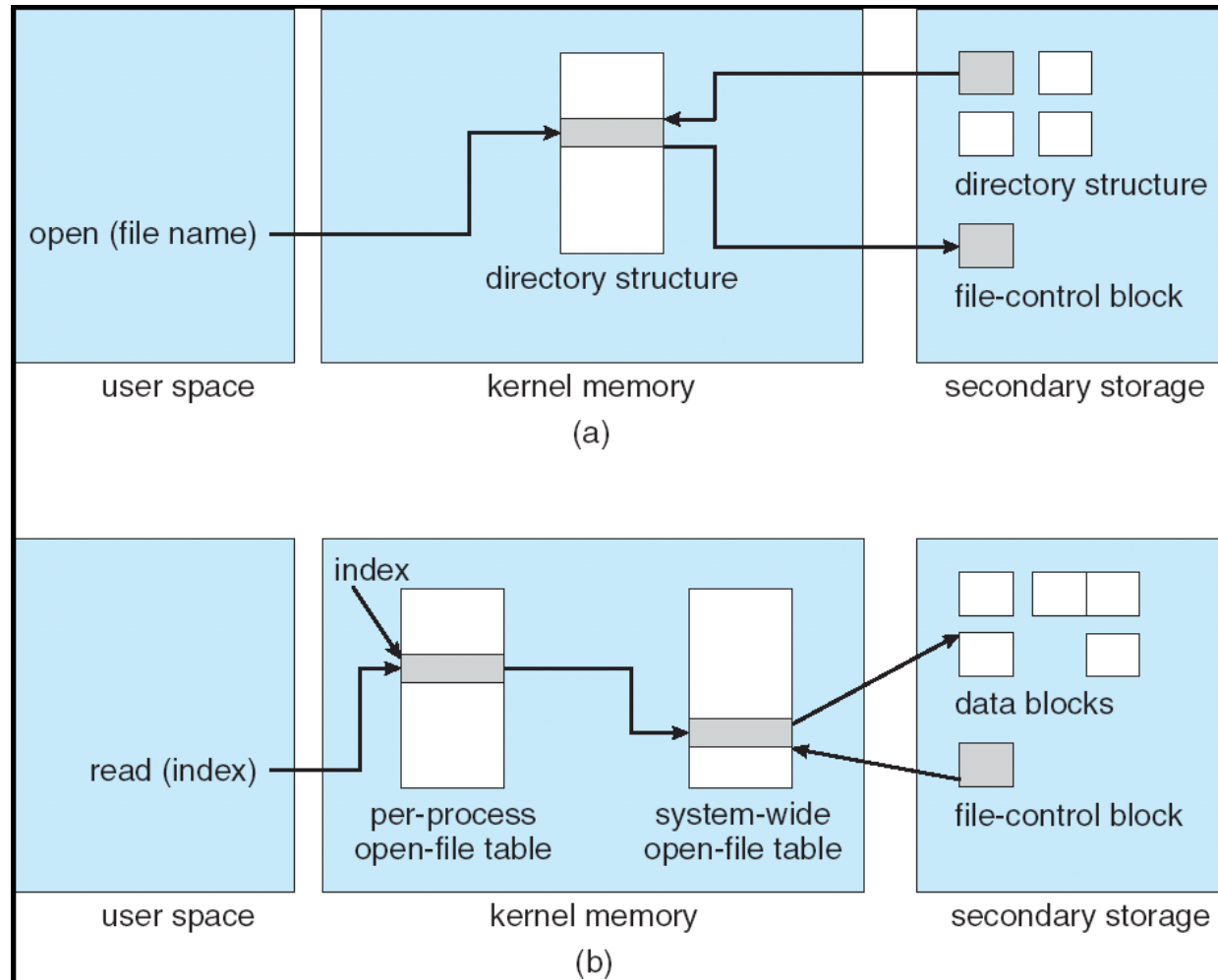
Multi-level Indexed Allocation (Prinzip)



Operating System Concepts, 11.2, Silberschatz and Galvin © 1998



Indexed Allocation - Mapping



open („filename“):

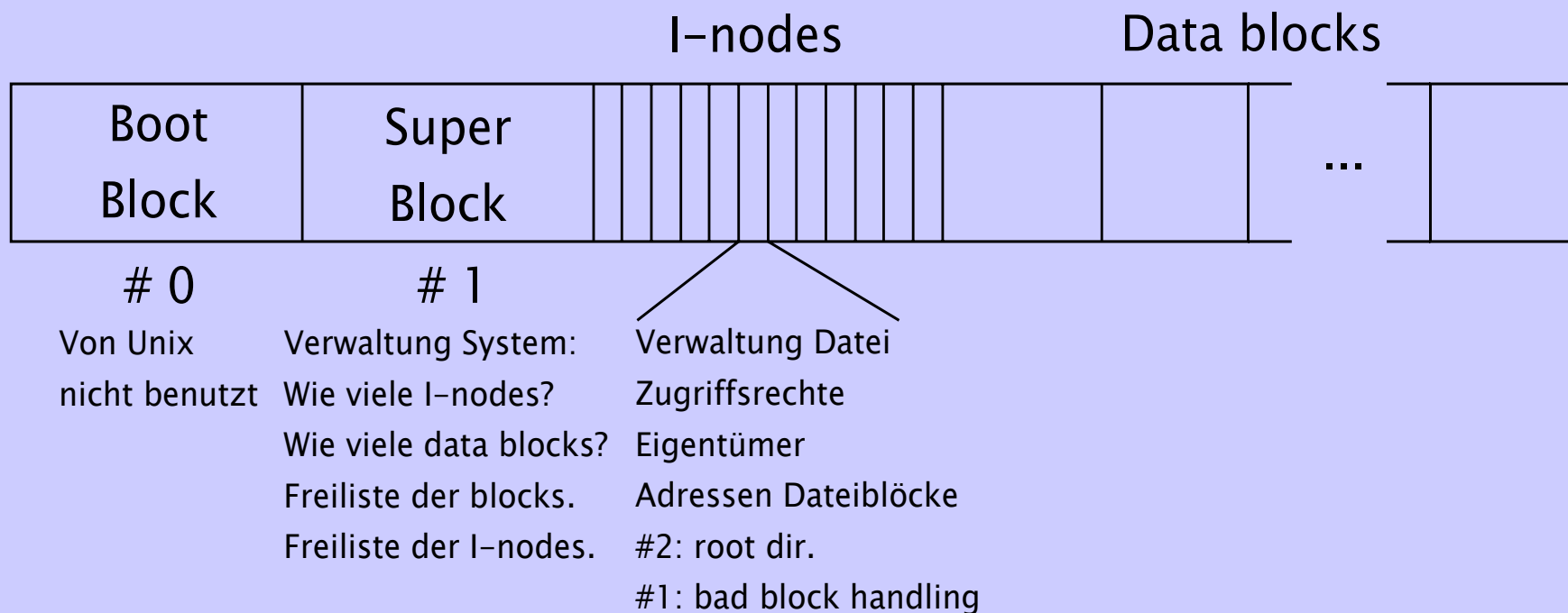
- durchsuche directory structure auf disk
- kopiere gefundenen FCB in den Kern mit Verweis auf Disk-FCB

read (index):

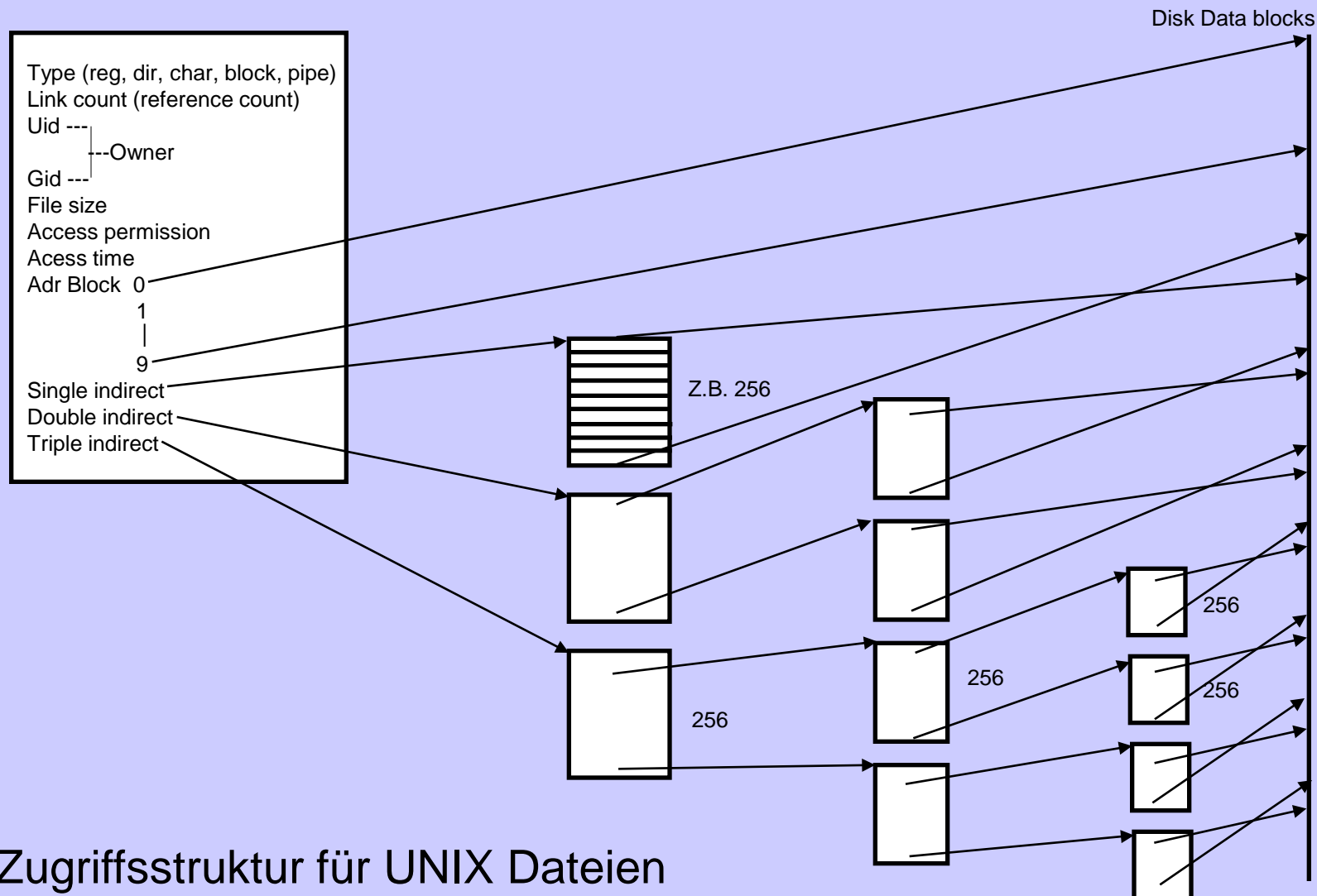
- vom Index aus finde FCB im Kern
- Aus read-position berechne gesuchten data-block
- hole data block

Silberschatz, Galvin and Gagne, Abb.11.3

UNIX Filesystem: Plattenorganisation



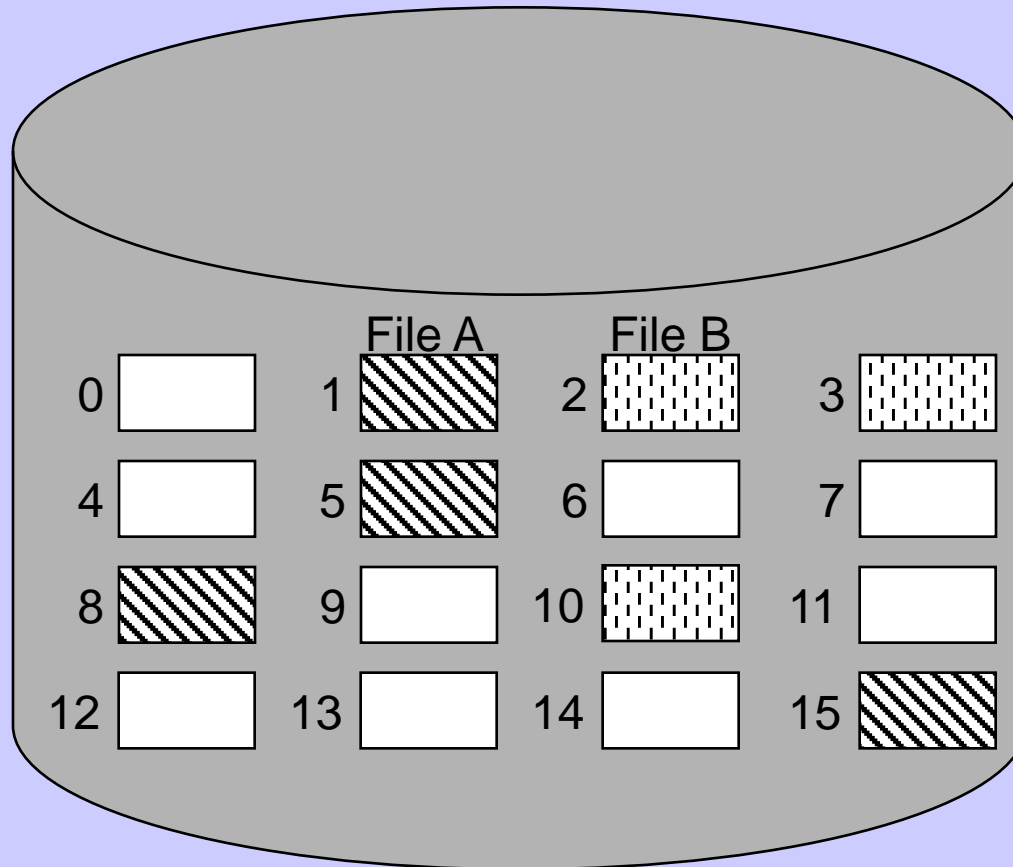
UNIX: Organisation von I-node und Datei



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**).

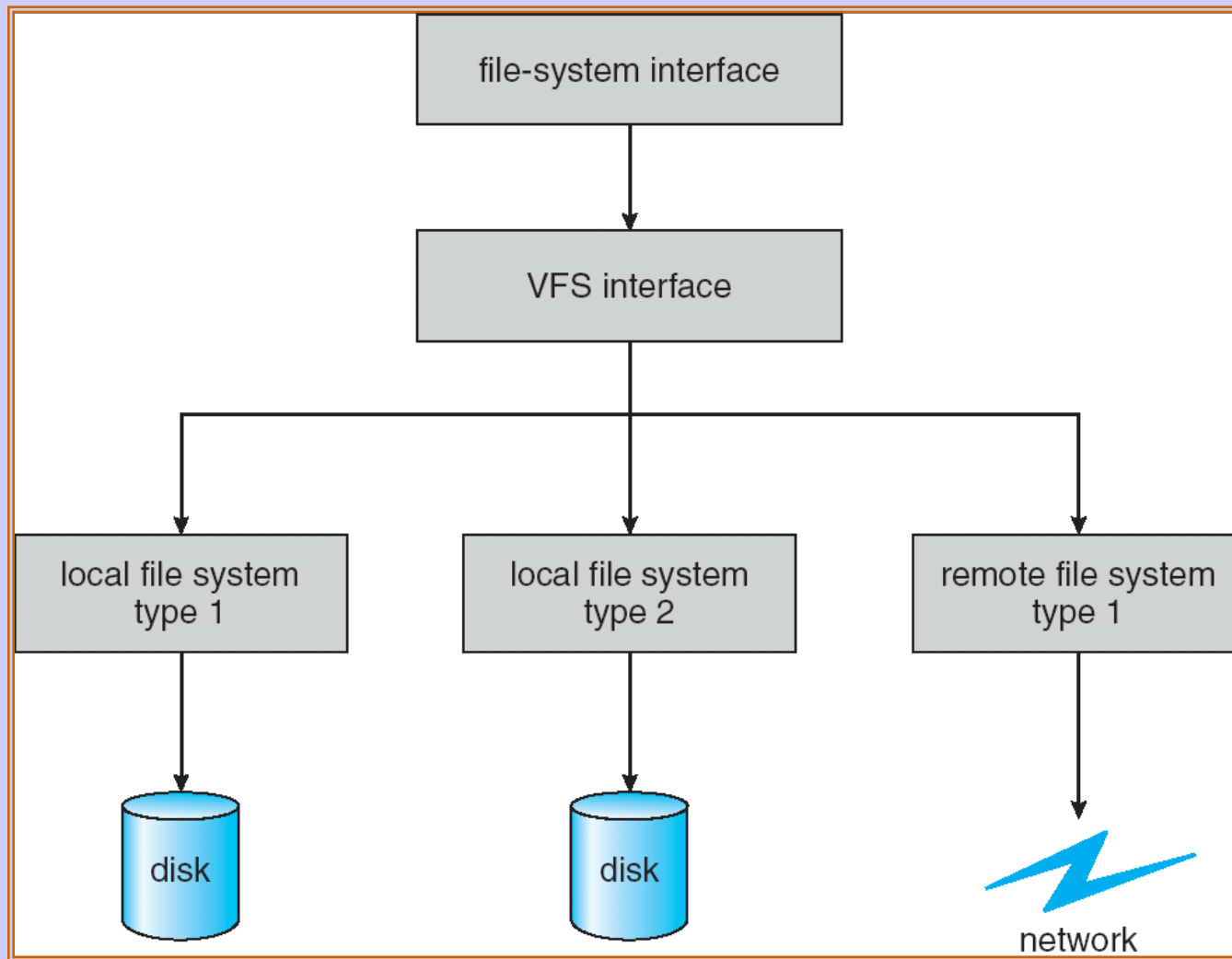


Virtual File Systems

- 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

