EDAF35: OPERATING SYSTEMS

MODULE 7
FILE SYSTEMS

CONTENTS

FILE SYSTEMS

- Concepts
- Function/Operations/Interface
- Visible Structures
- Internal Structure
- Space Management
- Performance/Recovery
- Network File Systems

User and application programmer's view

OS developer's view

11, 12, 10 (other ver.)

CHAPTER 10
FILE SYSTEM

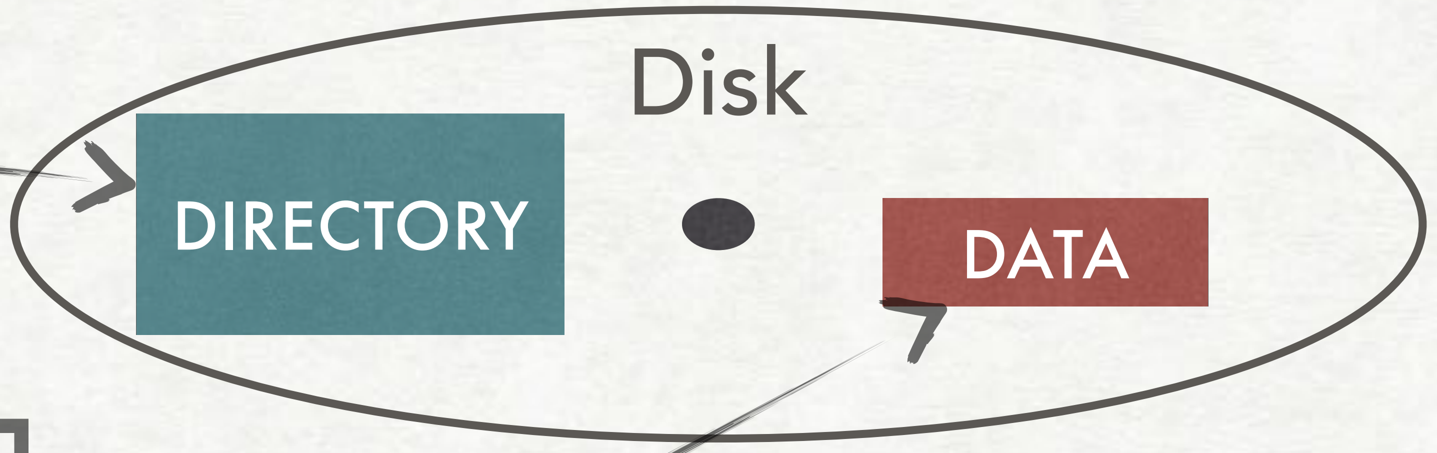
CHAPTER 11
FILE SYSTEM IMPLEMENTATION

CHAPTER 12
MASS STORAGE STRUCTURE

ATTRIBUTES AND OPERATIONS

FILE SYSTEMS

Attributes: describe a file



- name
- identifier
- type (exe, text, gif,…)
- location
- size
- protection
- time, date, owner
- extended attributes (e.g. checksum)

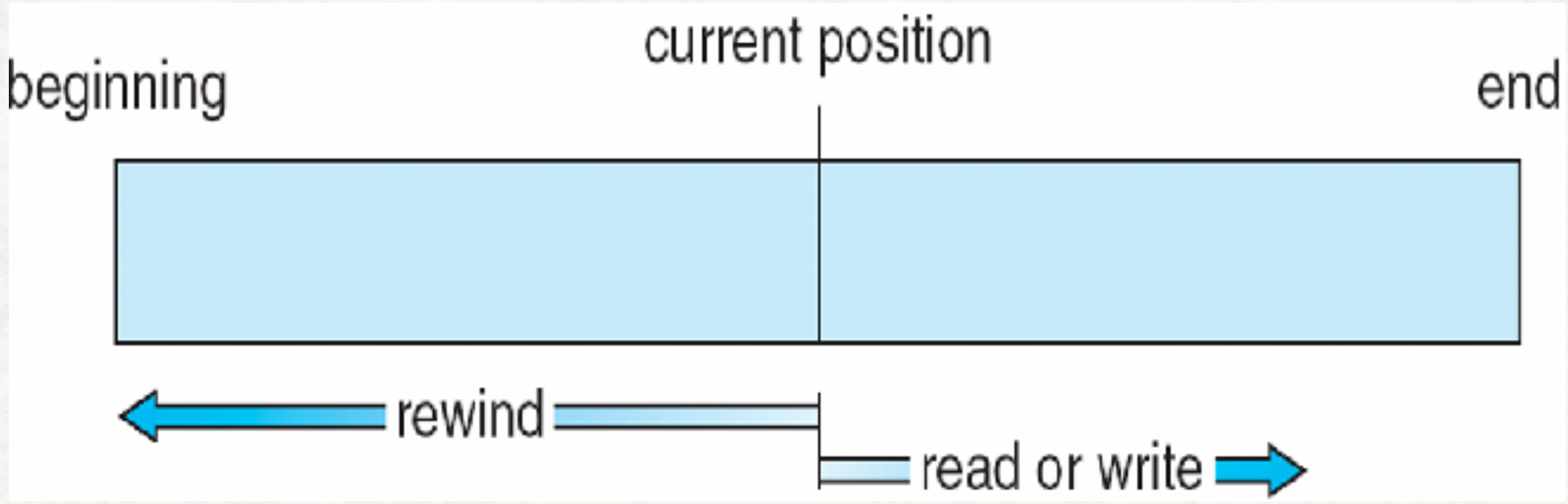
file type	usual extension	function
executable	exe, com, bin or none	ready-to-run machine-language program
object	obj, o	compiled, machine language, not linked
source code	c, cc, java, pas, asm, a	source code in various languages
batch	bat, sh	commands to the command interpreter
text	txt, doc	textual data, documents
word processor	wp, tex, rtf, doc	various word-processor formats
library	lib, a, so, dll	libraries of routines for programmers
print or view	ps, pdf, jpg	ASCII or binary file in a format for printing or viewing
archive	arc, zip, tar	related files grouped into one file, sometimes compressed, for archiving or storage
multimedia	mpeg, mov, rm, mp3, avi	binary file containing audio or A/V information

Operations
(some on directory, some on data)
(some info is on disk, some in memory)

- create
- write/read (at a location)
- reposition — seek
- delete/truncate
- open(f) — search in directory and bring info in memory
- close(f) — move memory info to disk and free memory
- locks - shared vs. exclusive, mandatory vs. advisory

ACCESS METHODS

FILE SYSTEMS



Sequential Access:

read next

write next

rewind (reset)

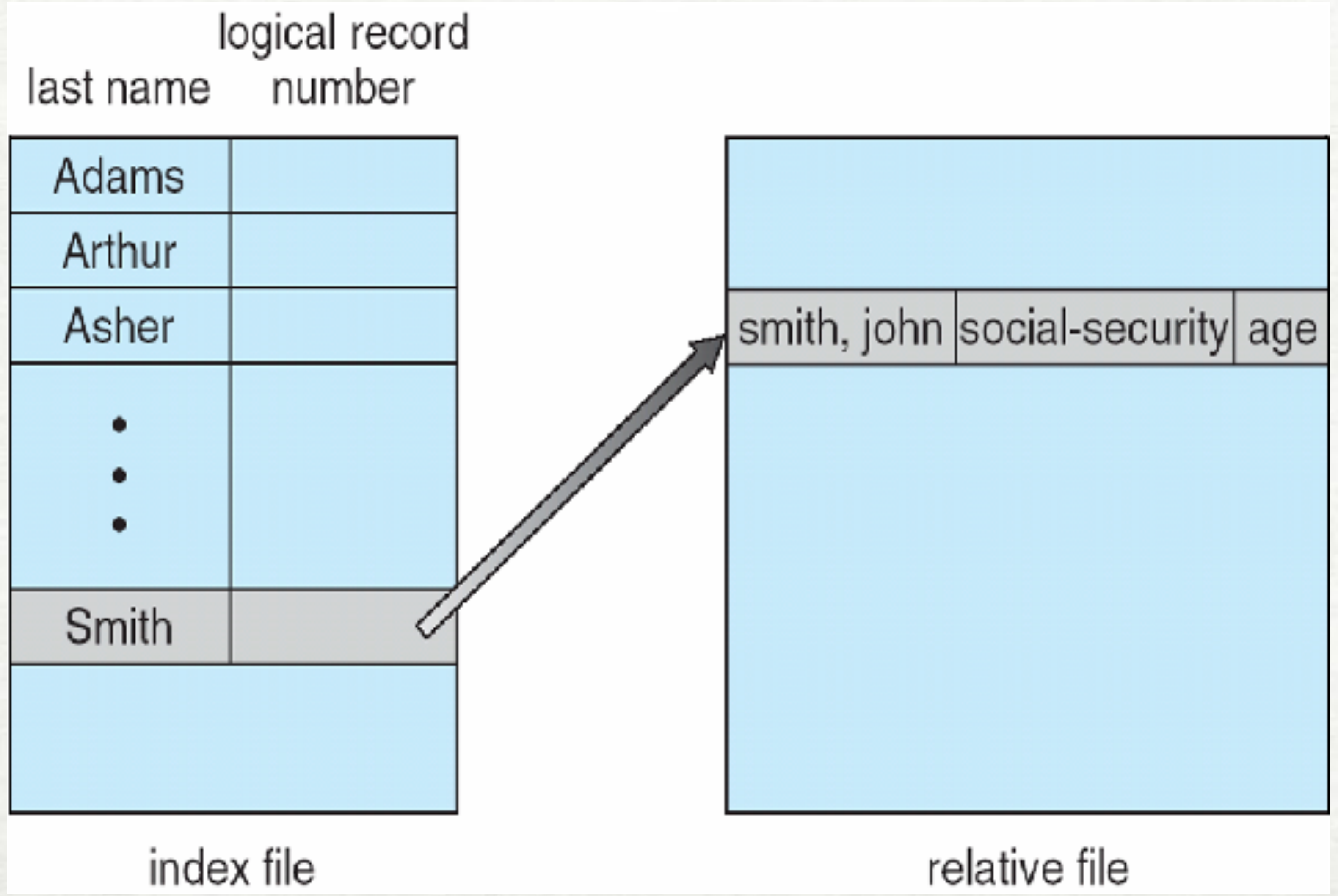
Direct Access:

position at n (relative)

read k records

write k records

OTHERS
(BUILT ON TOP OF THE MORE BASIC ONES)



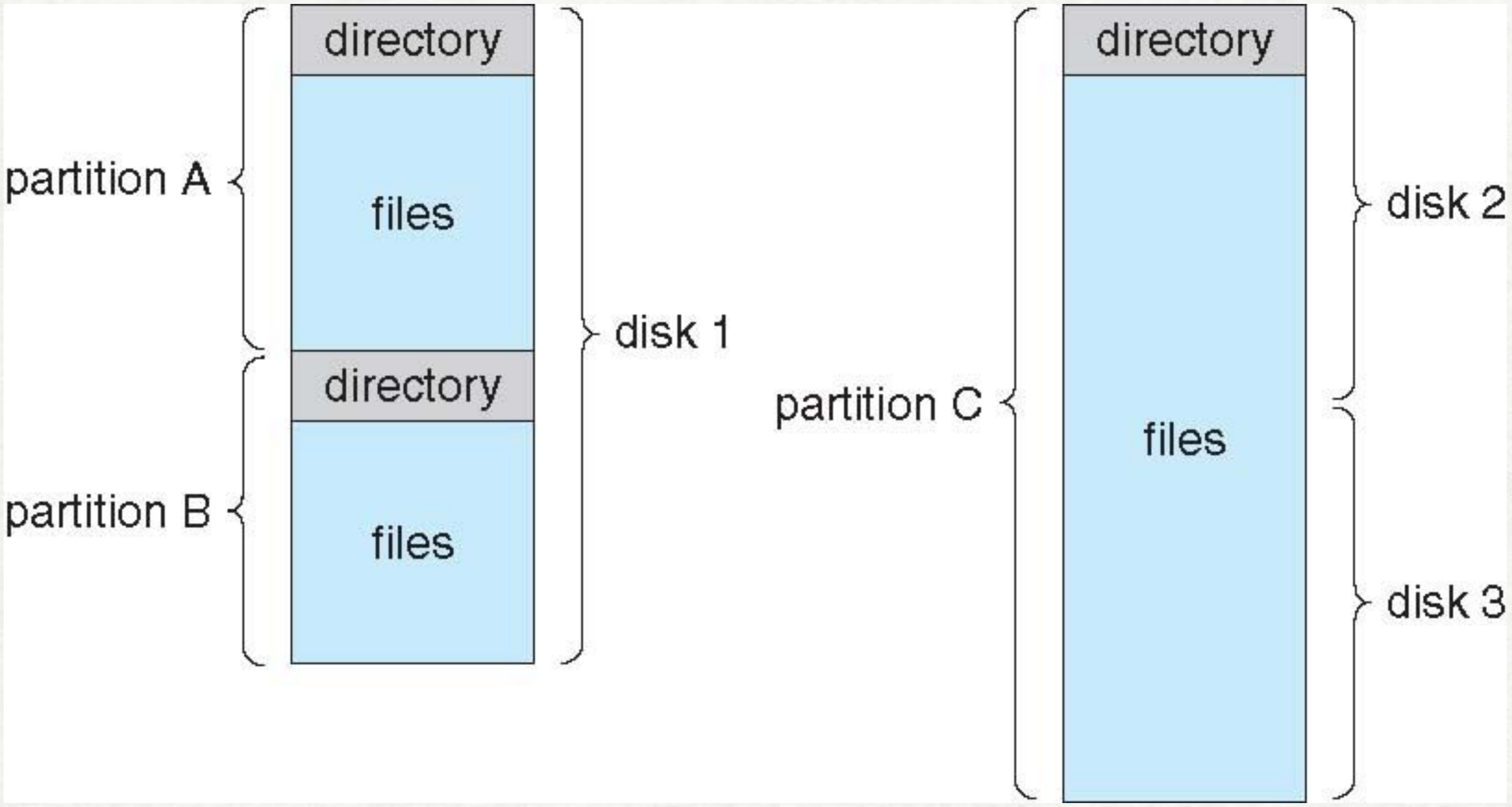
Index/Relative Access:

index file (transparent)

relative file (actual content)

ORGANIZATION

FILE SYSTEMS



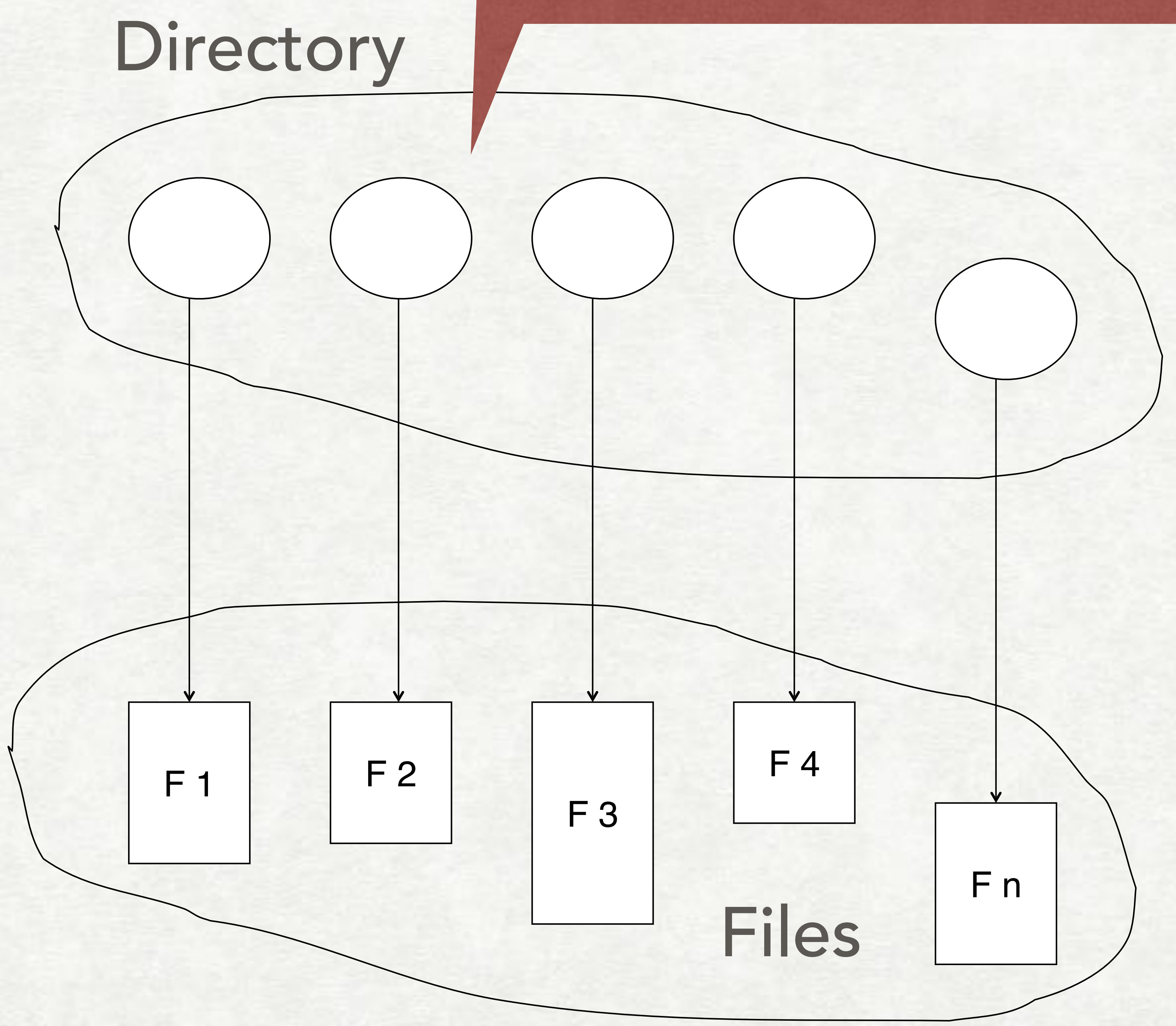
FS occupies a partition
(part of a disk or several disks)

Need to efficiently…

CREATE/DELETE/RENAME A FILE

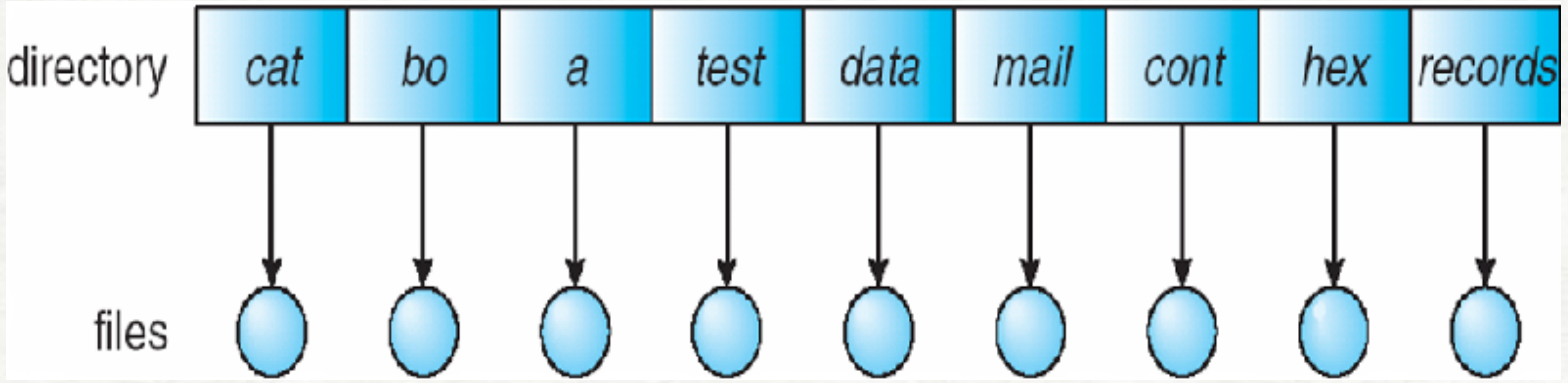
LIST DIRECTORY

SEARCH/TRAVERSE THE FILE SYSTEM



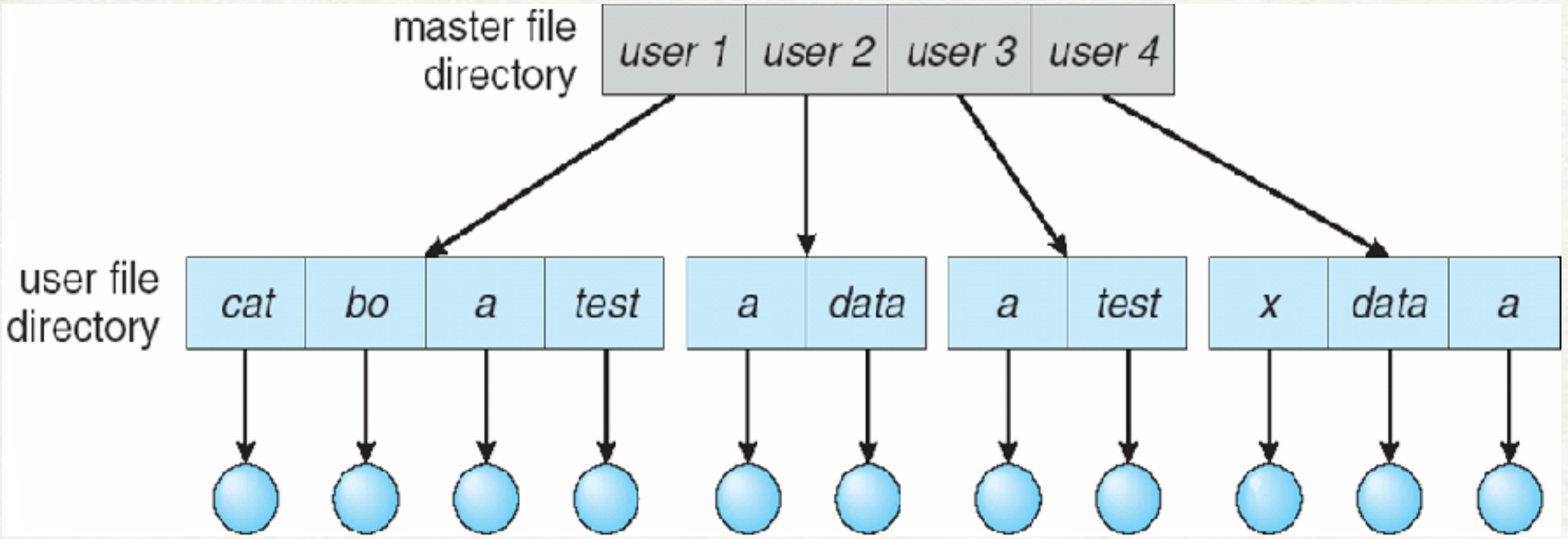
DIRECTORY STRUCTURES

FILE SYSTEMS



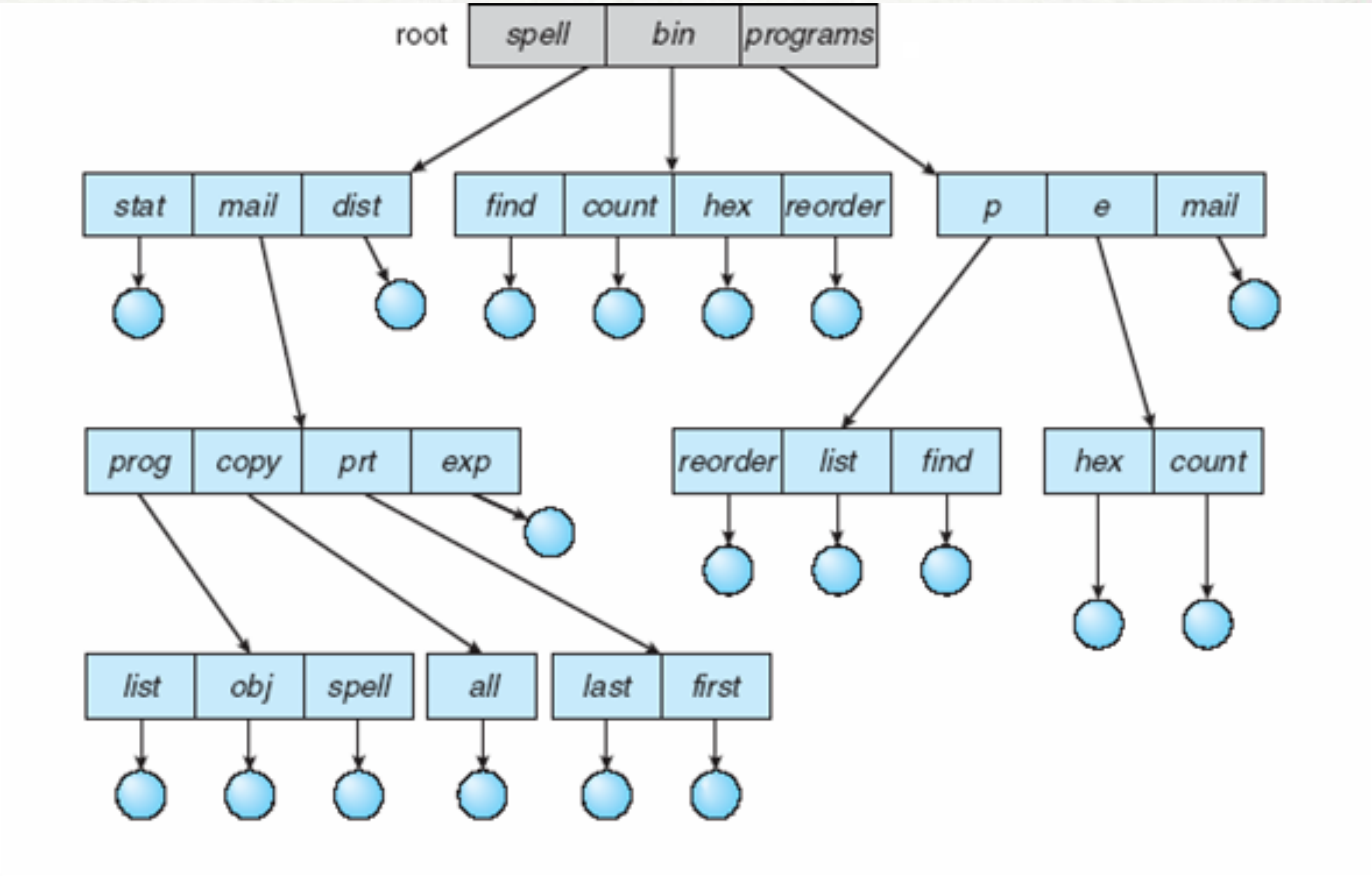
Single Level

issues: naming, grouping



Two Level

issues: grouping, sharing

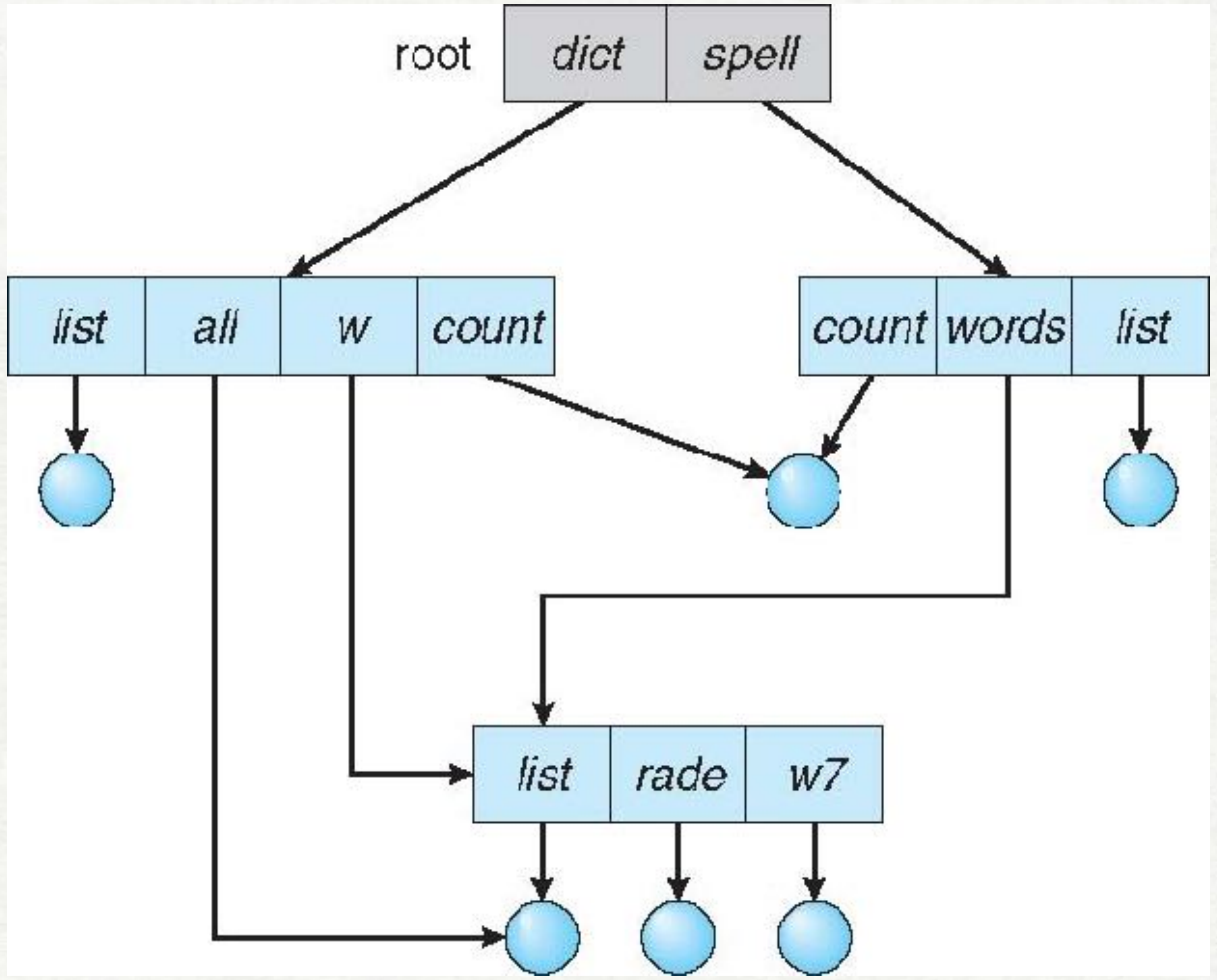


Tree

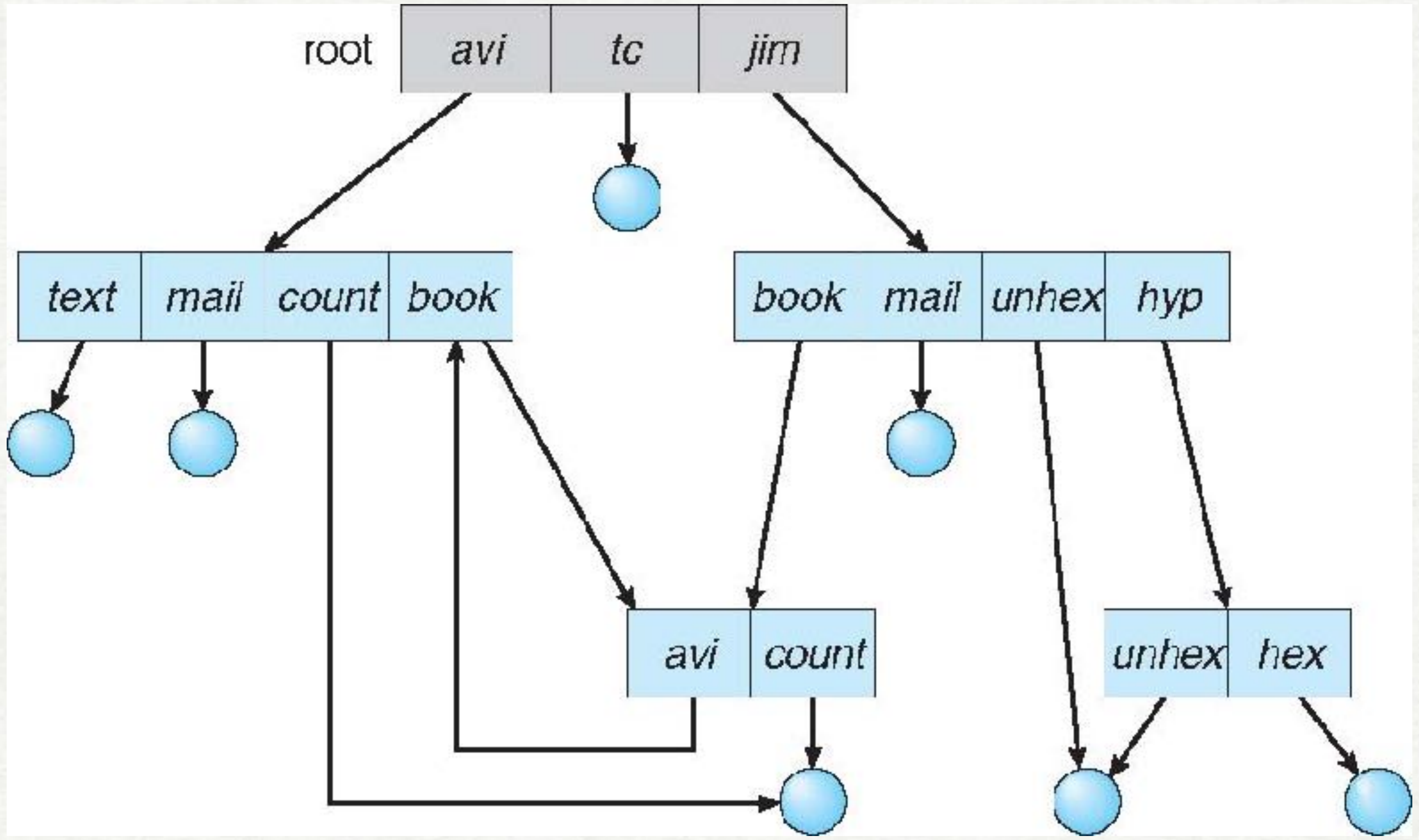
issues: sharing

MODERN DIRECTORY STRUCTURES

FILE SYSTEMS



Acyclic Graphs
issues: deletions

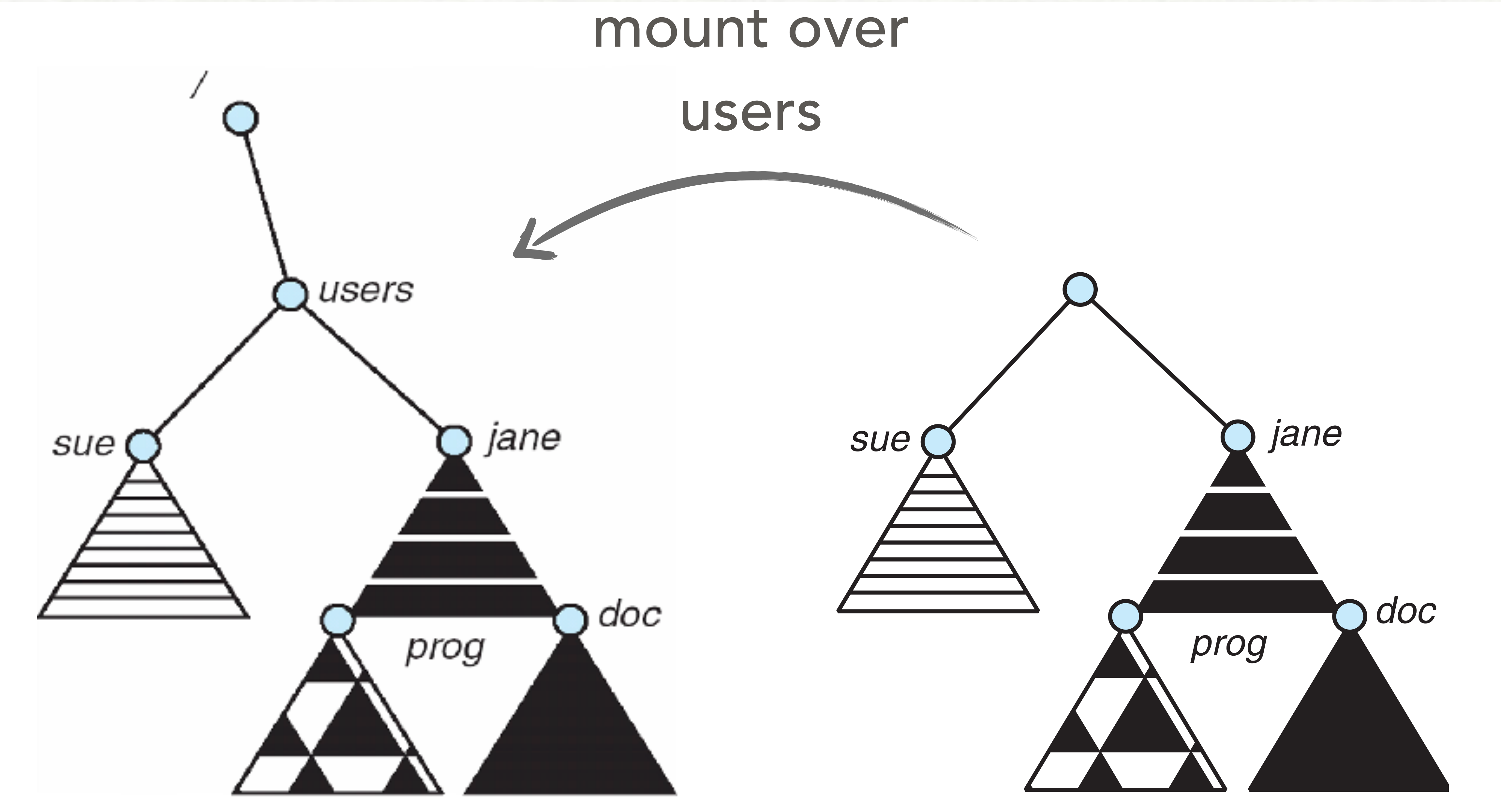


General Graphs
issues: deletions, search

MOUNTING

FILE SYSTEMS

- build a common file system structure out of separate volumes/file systems



(a) (b)

```
>man mount, umount
```

FILE SHARING

FILE SYSTEMS

- multi-user systems: **protection** — extra file attributes: **owner/group**
- **distributed systems** — sharing across a network:
  - manually (ftp), automatically (DFS), semi-auto (WWW)
  - Distributed FS: client-server approach for mounting remote FS (see NFS, IPFS)
  - **helpers**: distributed information systems (see DNS, NIS, CIFS, LDAP)
  - **choices**: consistency semantics (Unix, session, immutable)

PROTECTION

FILE SYSTEMS

- file owner — decides "who can do what?"
  e.g. *read, write, delete, execute, append, list,…*

- UNIX
  - user/group/others — read/write/execute
  - >man chmod, chgrp, chown

```
> chmod 644 program.c
> chmod ug+x atool
```

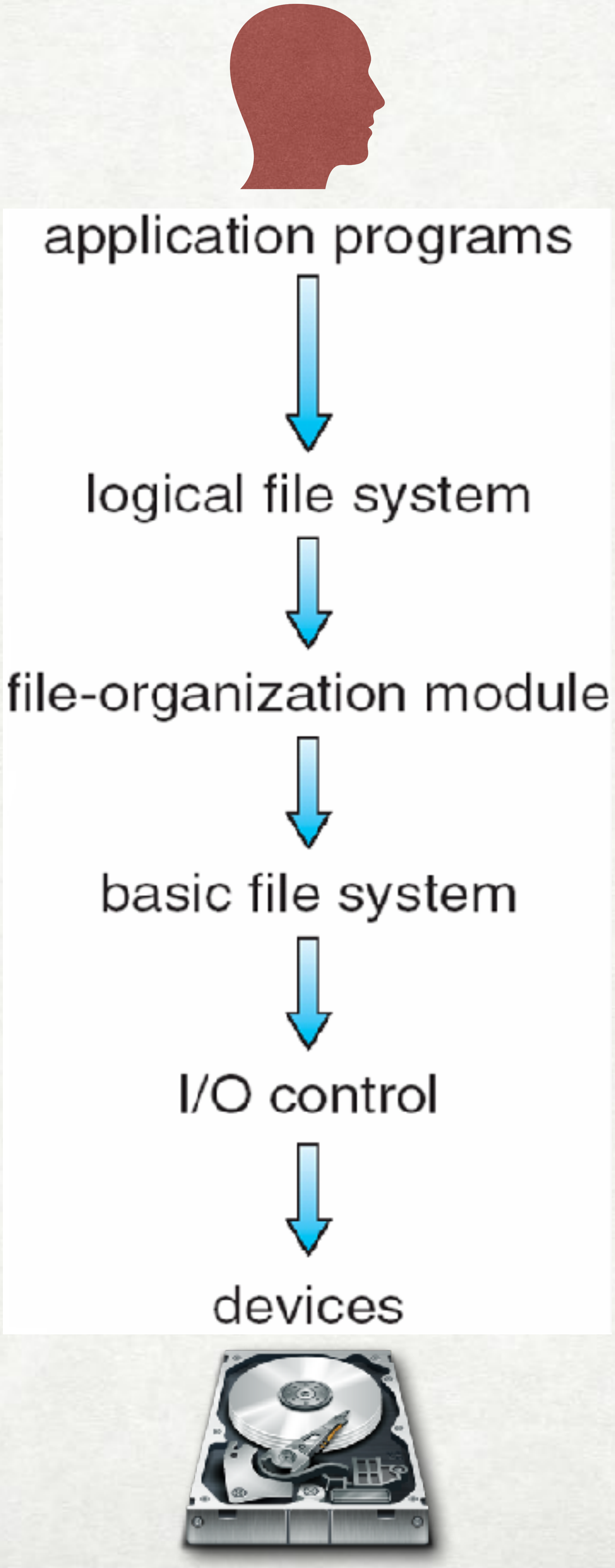
```
>ls -al
```

-rw-rw-r--	1	pbg	staff	31200	Sep 3 08:30	intro.ps
drwx------	5	pbg	staff	512	Jul 8 09.33	private/
drwxrwxr-x	2	pbg	staff	512	Jul 8 09:35	doc/
drwxrwx---	2	pbg	student	512	Aug 3 14:13	student-proj/
-rw-r--r--	1	pbg	staff	9423	Feb 24 2003	program.c
-rwxr-xr-x	1	pbg	staff	20471	Feb 24 2003	program
drwx--x--x	4	pbg	faculty	512	Jul 31 10:31	lib/
drwx------	3	pbg	staff	1024	Aug 29 06:52	mail/
drwxrwxrwx	3	pbg	staff	512	Jul 8 09:35	test/

FILE SYSTEMS IMPLEMENTATION

LAYERED STRUCTURE

FILE SYSTEM IMPLEMENTATION



- gradual translation from a unified view to specific device operations
- hides low level details, caches/buffers, timing
- easy to retarget other devices
- allows for several FS and devices in a single system

SEE ALSO PARTITIONS AND MOUNTING IN THE TEXT BOOK

INTERNAL STRUCTURES

FILE SYSTEMS IMPLEMENTATION

IN MEMORY (VOLATILE)	ON DISK (PERSISTENT)

In memory (volatile):

- Mount table:
  info about mounted volumes
- Directory structure cache
- System-wide open-file table
- Per-process open-file table
  pointer to system entry, position in file
- Buffers/Caches for data

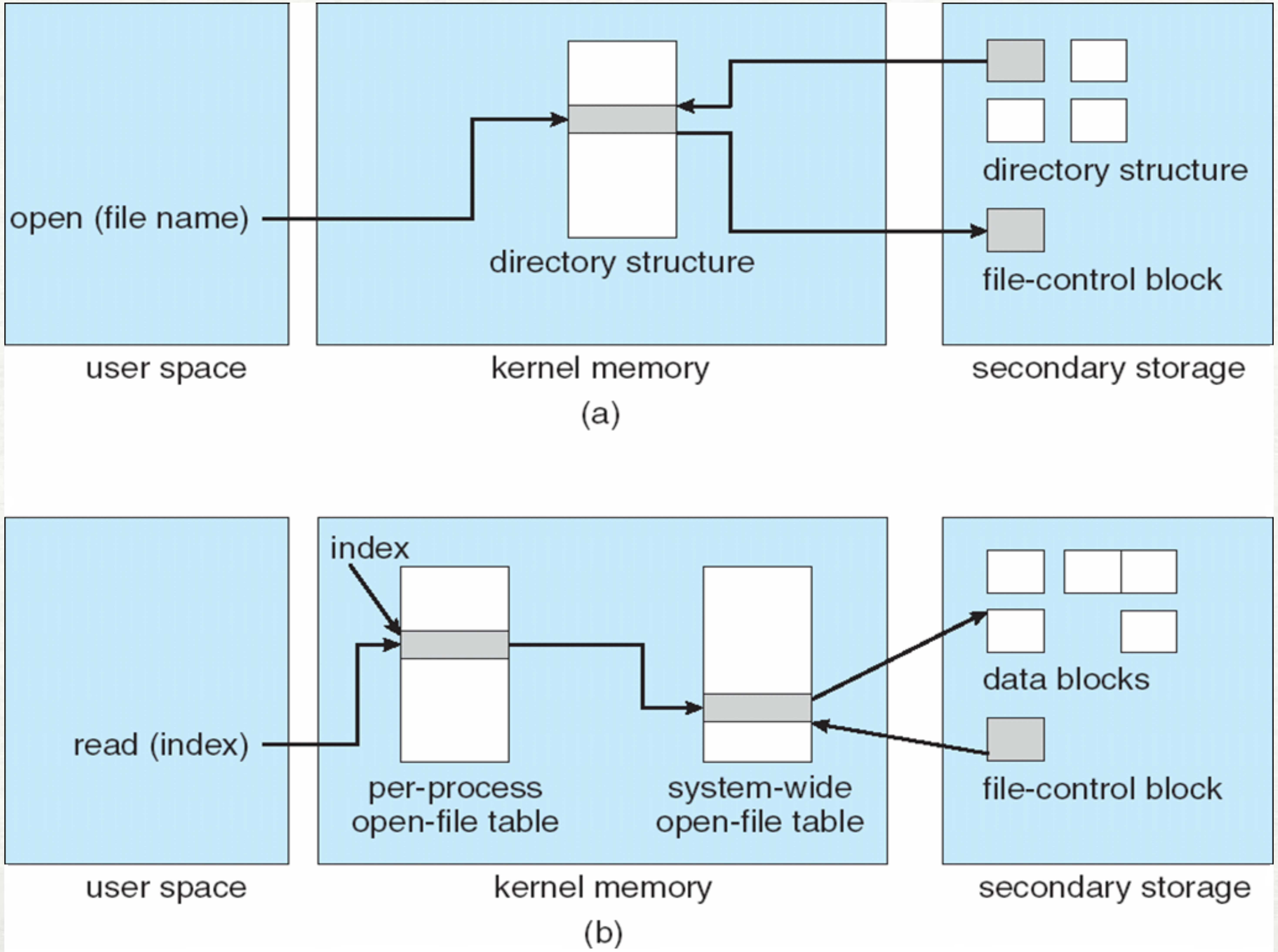
On disk (persistent):

- Boot control block/volume:
  how to boot an OS from there
- Volume control block/volume:
  blocks, free/used, counts, pointers
- Directory structure/FS:
  file names, pointers to FCB
- File control blocks (FCB)/file:
  file permissions, dates, owner, pointers (inode)

most common, others are possible

OPEN AND READ OPERATIONS

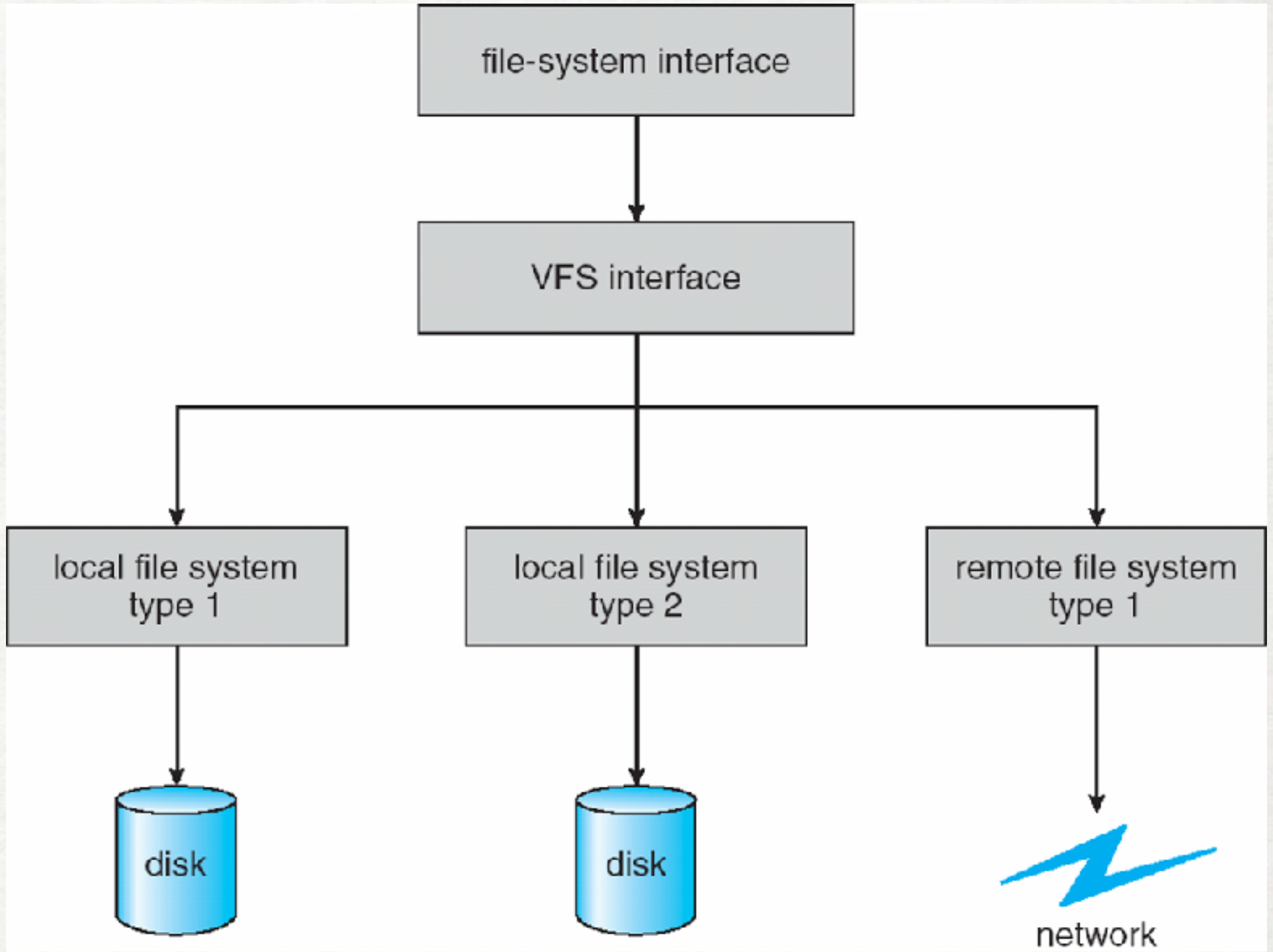
FILE SYSTEMS IMPLEMENTATION



VIRTUAL FILE SYSTEMS

FILE SYSTEMS IMPLEMENTATION

- VFS exposes a generic API towards different FS
- local and networked FS supported:
  vnode — network wide FCB (inode-like)
- VFS redirects requests to the right FS



VFS in **Linux**:

4 objects – inode, <u>file</u>, superblock, dentry – each with their API calls

API calls for <u>file</u> – open, close, read, write, mmap

DIRECTORY IMPLEMENTATION

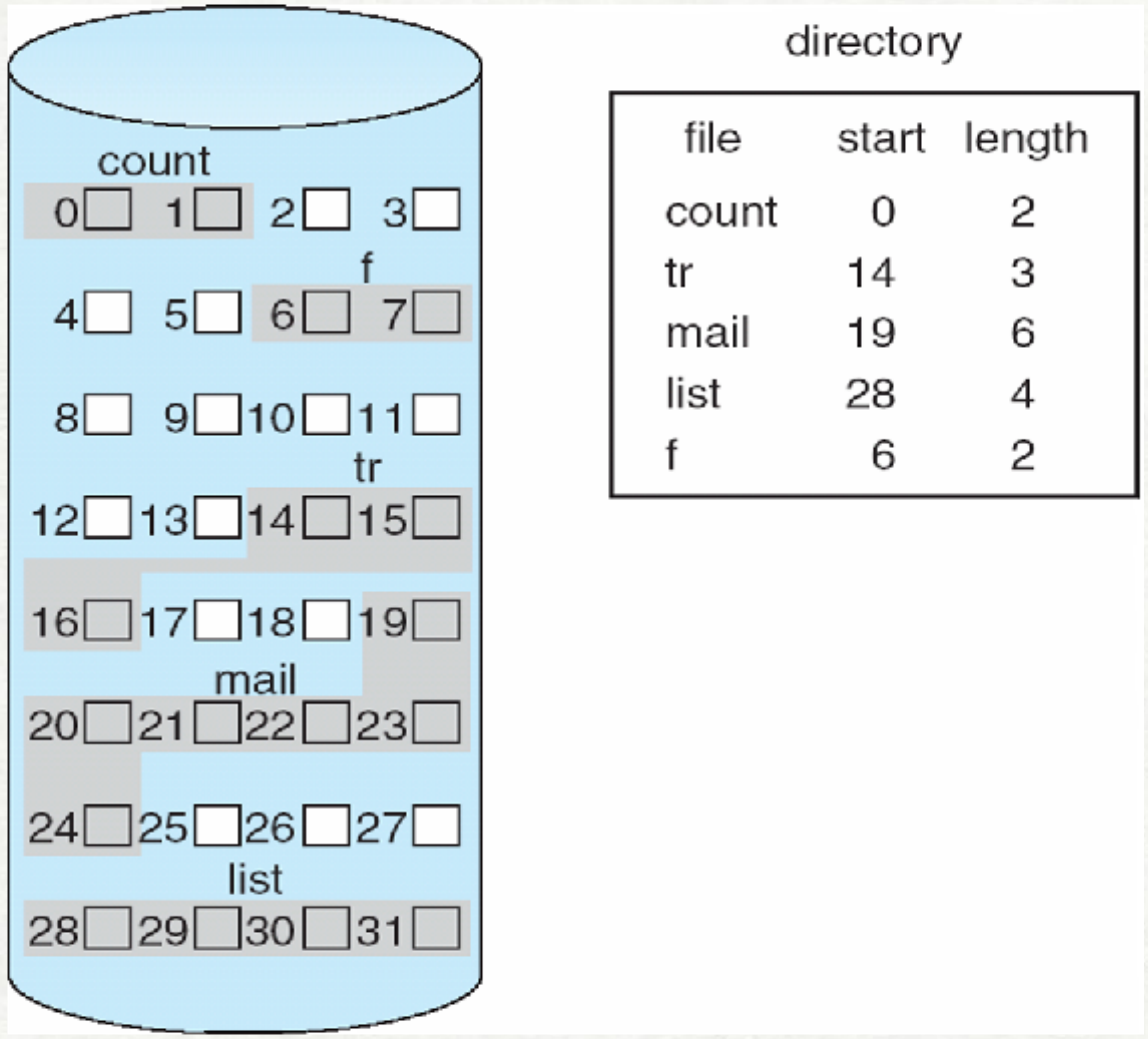
FILE SYSTEMS IMPLEMENTATION

- **Linear list** — of file names, pointer to data block(s)
    - variations to speed up search: ordered, trees, etc.
- **Hash table** — name based hash, as above
    - variations to handle collisions: chaining

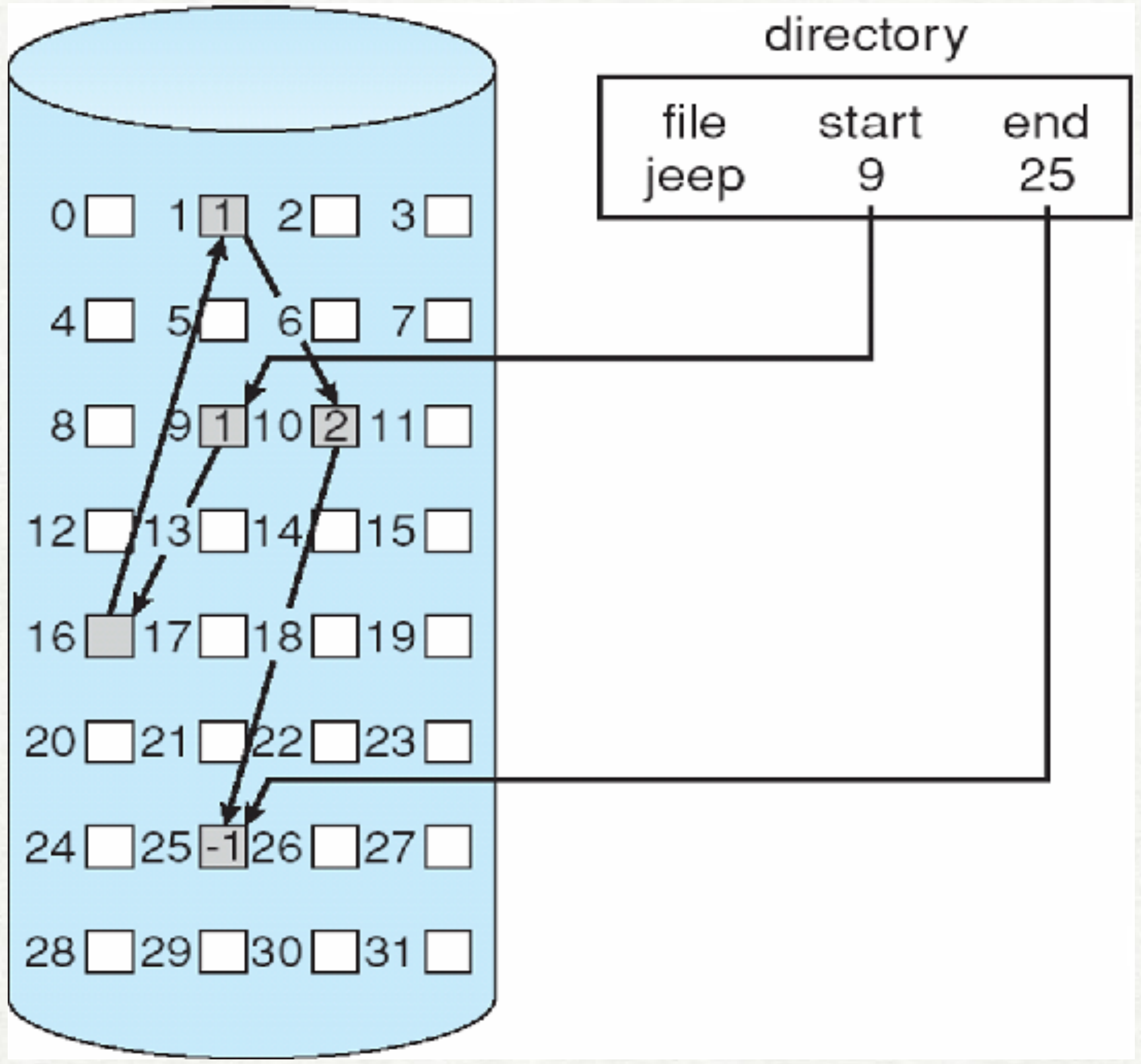
DISK BLOCK ALLOCATION

FILE SYSTEMS IMPLEMENTATION

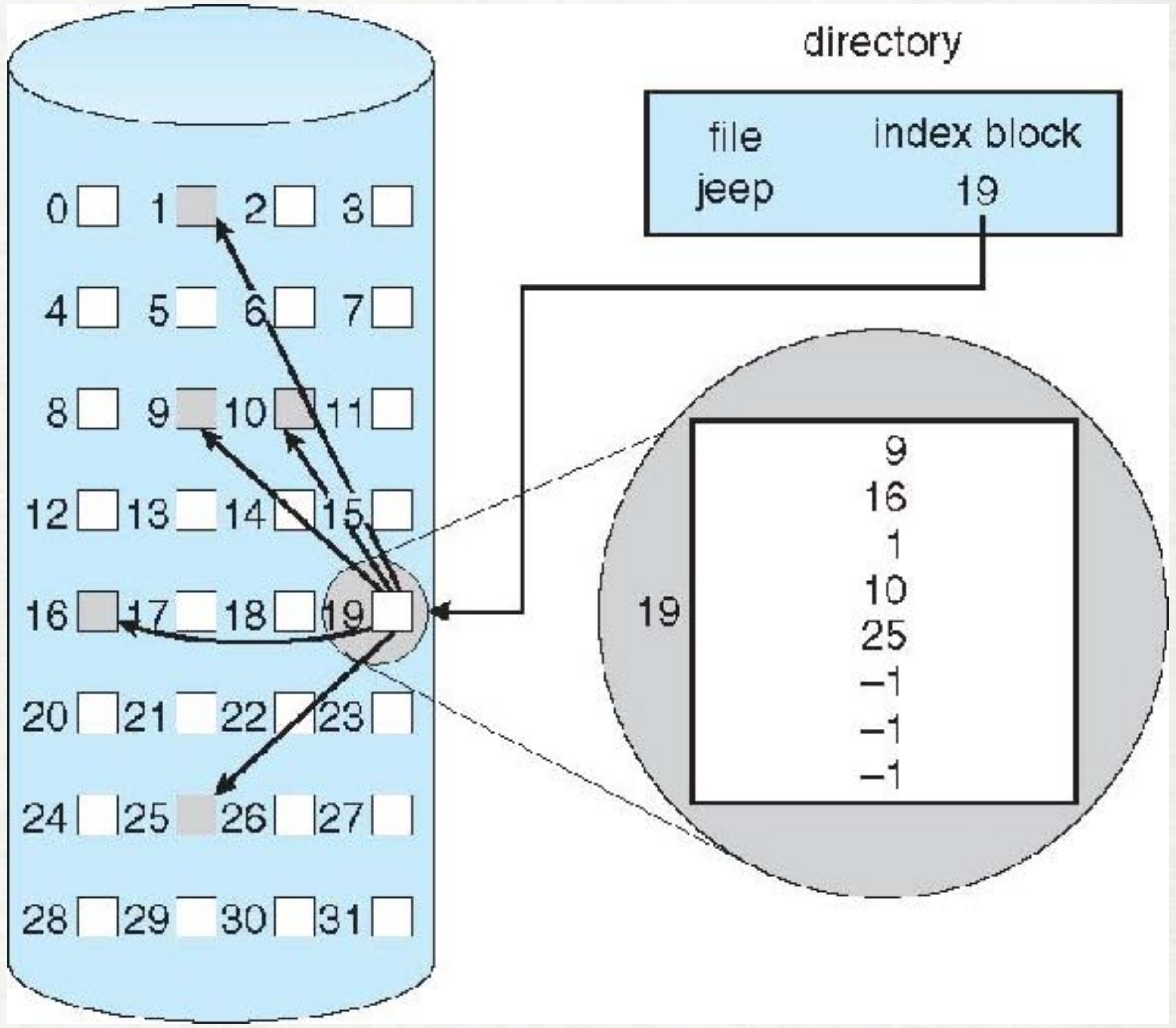
CONTIGUOUS



LINKED



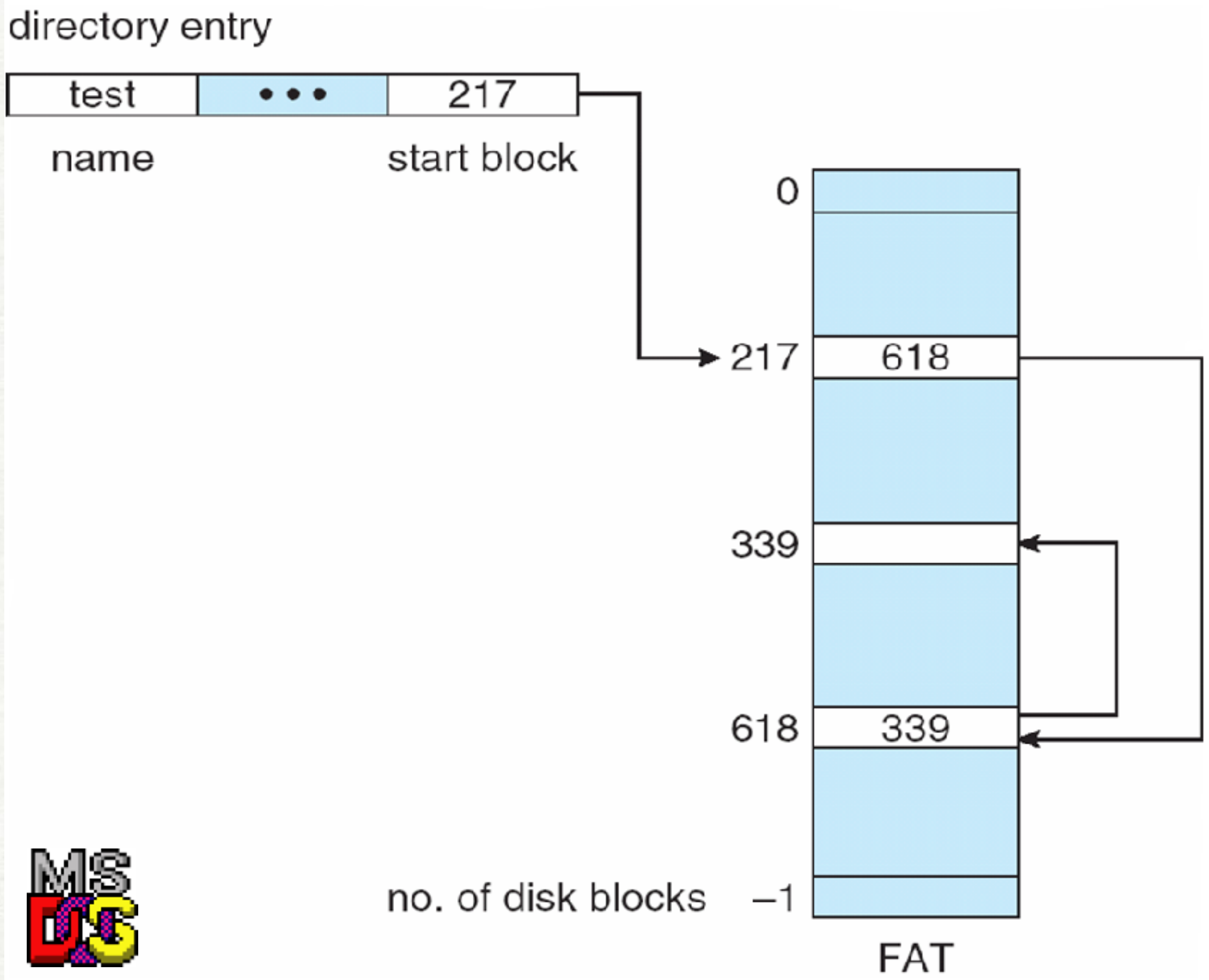
INDEXED



- Each have variations
- Pros and Cons?

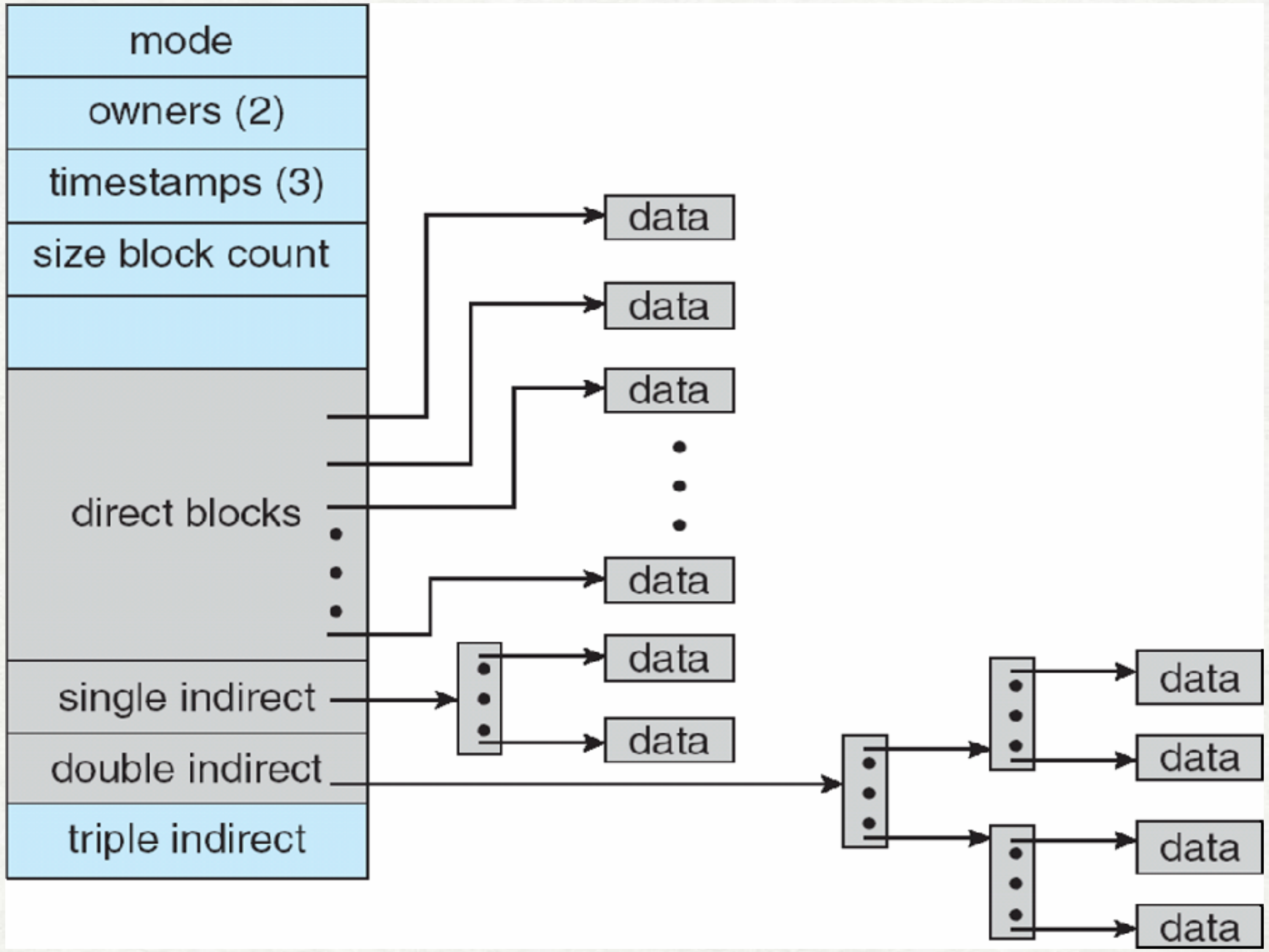
EXAMPLE: FILE-ALLOCATION TABLE (FAT)

- linked allocation
- list/pointers separate from data blocks
- initially 12-bits addresses
- cache the FAT for improved access speed



EXAMPLE: UNIX UFS INODE

- combined scheme
- 4KB blocks,
  32-bit addresses
- more blocks can be linked than a 32-bit file pointer can address!
- *Exercise*: compute the maximum file size



FREE SPACE MANAGEMENT

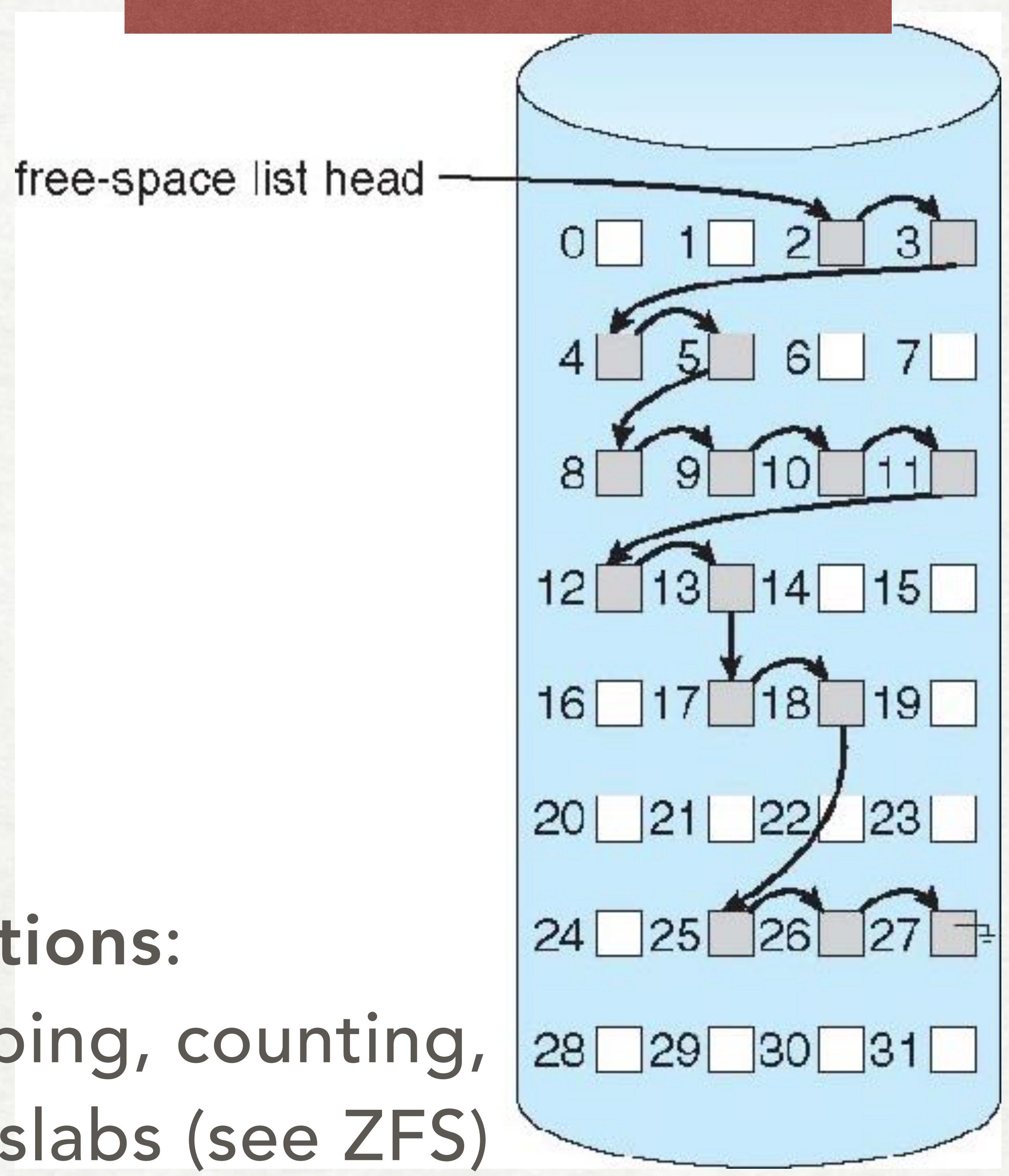
FILE SYSTEMS IMPLEMENTATION

BIT VECTOR OR BITMAP

$$\text{bit}[\boldsymbol{i}] = \begin{cases} 1 \Rightarrow \text{block}[\boldsymbol{i}] \text{ free} \\ 0 \Rightarrow \text{block}[\boldsymbol{i}] \text{ occupied} \end{cases}$$

- can get big: 256MB for 1TB disk, 4KB blocks
- variations exist

LINKED LIST



- **variations:** grouping, counting, metaslabs (see ZFS)
- pros and cons?

RECOVERY

FILE SYSTEMS IMPLEMENTATIONS

- failures happen!
  consistency checking — is the metadata correct? (e.g. UNIX fsck)

- *manual recovery*: back up & restore (on failure)

- *partial recovery*: checksums, duplicates (hw support, RAID arrays)

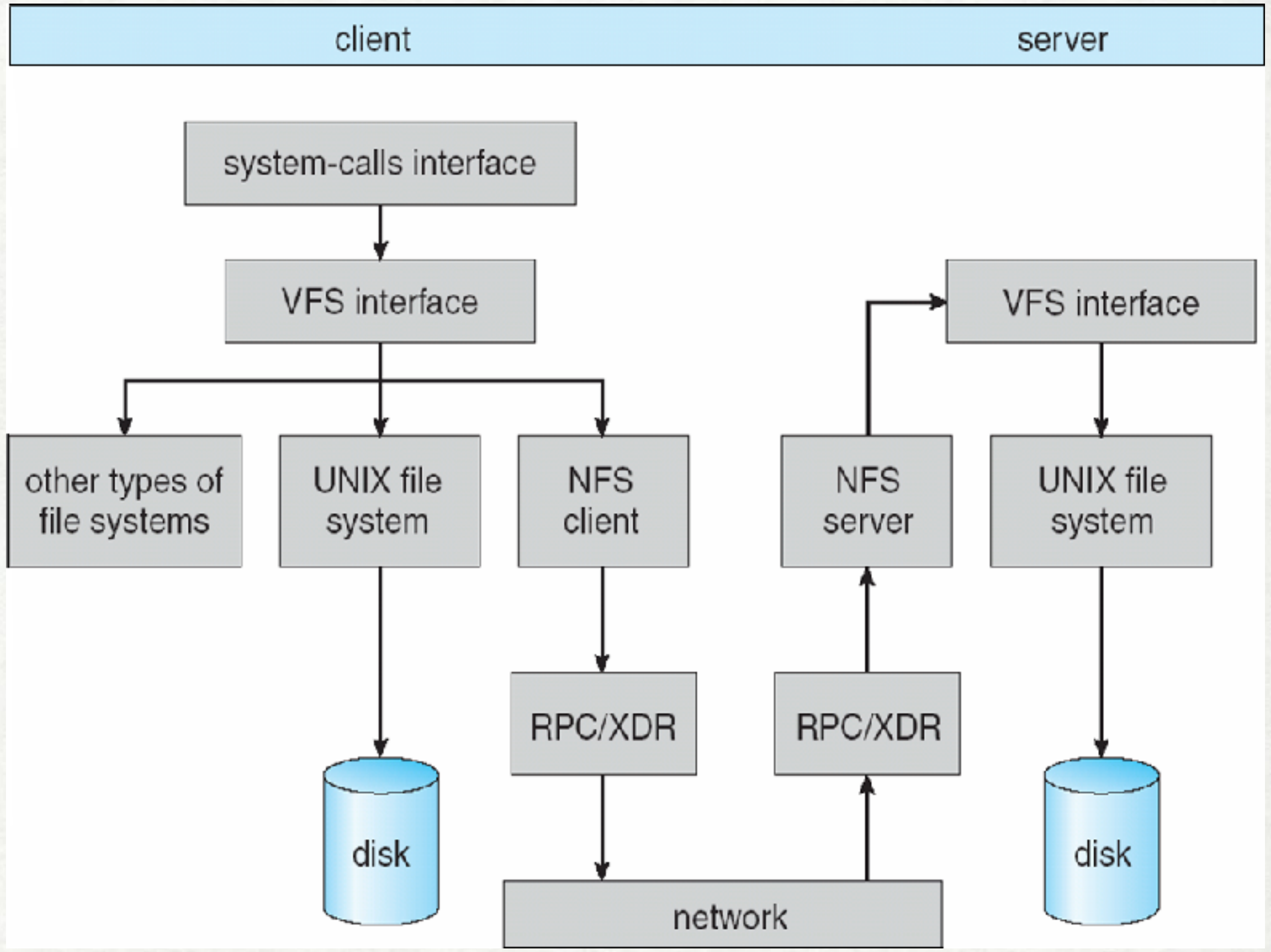
- *automated recovery*: **log-structured file systems** (journaling)
  - record metadata operations as transactions
  - a sequential log besides regular files
  - replay log (commit or rollback) on crash or to bring FS up to date

SEE ALSO **WAFL**
IN THE TEXTBOOK

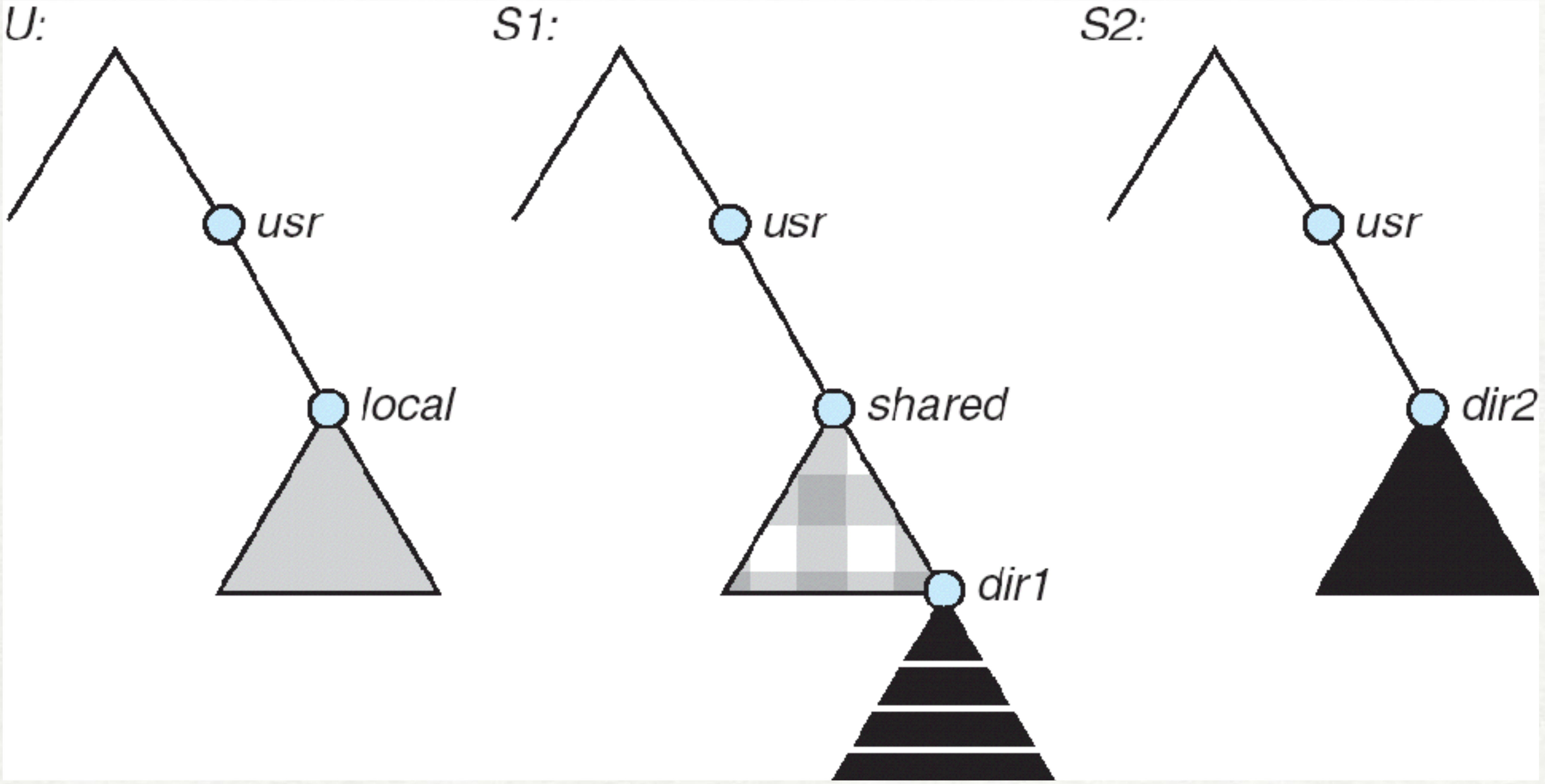
NETWORK FILE SYSTEM (NFS)

FILE SYSTEMS IMPLEMENTATION

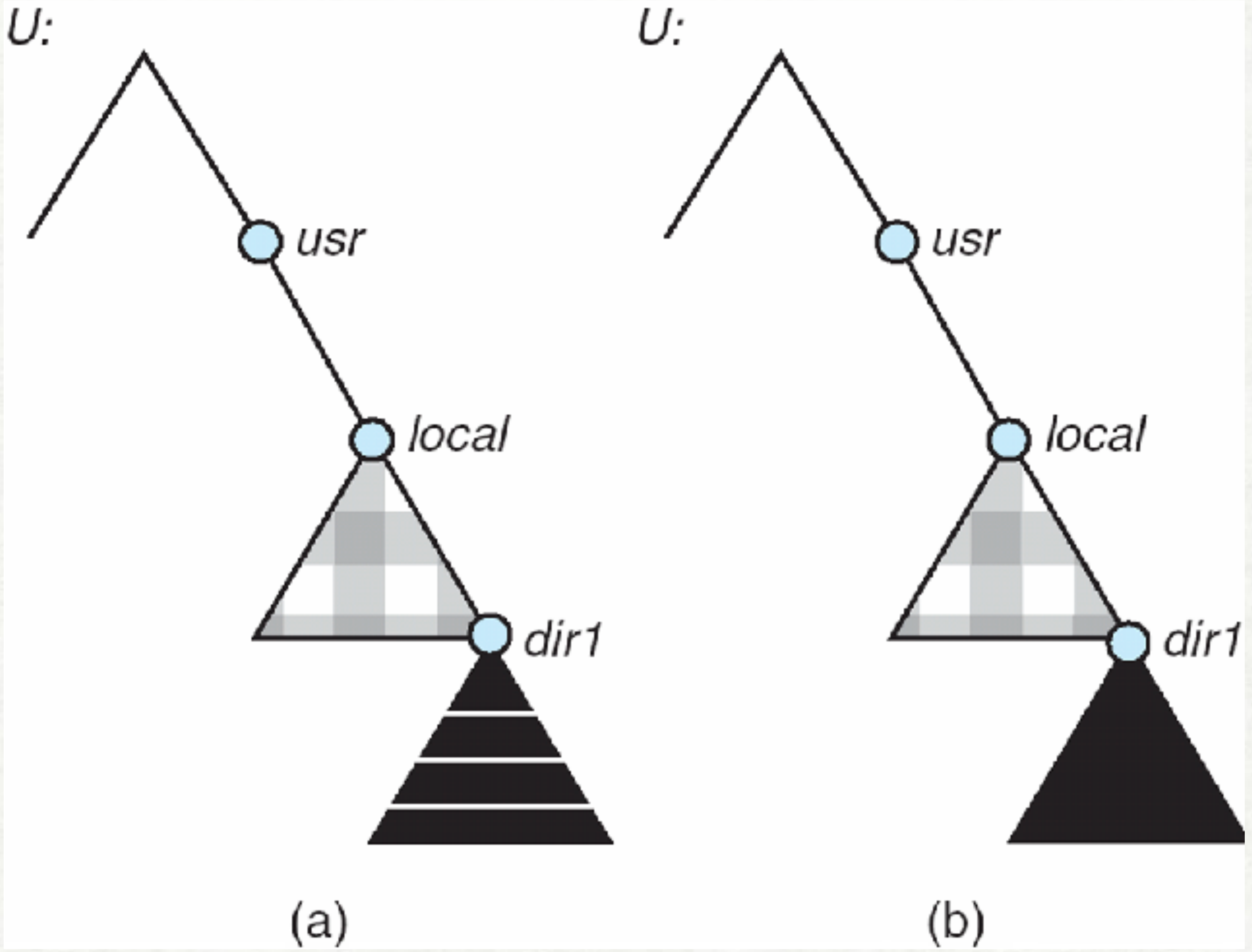
- SUN NFS — specification and implementation for remote files across LAN, WAN
- client-server architecture, using TCP/UDP networking
- two RPC/XDR-based protocols: ***mount*** and ***file access***
- stateless servers (incl. NFS v3)
- no concurrency-control mechanism (external locks assumed)



EXAMPLE: MOUNTING IN NFS



Three networked machines: `U, S1, S2`



Mounts
`S1:/usr/shared` over
`U:/usr/local`

Cascading mounts
`S2:/usr/dir2` over
`U:/usr/local/dir1`

END OF MODULE 7