File Systems: Interface and Implementation

CSCI 315 Operating Systems Design
Department of Computer Science

File System Topics

- File Concept
- Access Methods
- Directory Structure
- File System Mounting
- File Sharing
- Protection

File Concept

- A file is a named collection of related information recorded on secondary storage.

- **"Contiguous" logical** address space.

- File types:
  - Data:
    - numeric.
    - character.
    - binary.
  - Program (executable).

File Structure

- None: just a sequence of words or bytes.
- Simple **record** structure:
  - Lines,
  - Fixed length,
  - Variable length.
- Complex Structures:
  - Formatted document,
  - Relocatable load file.
- Can simulate last two with first method by inserting appropriate control characters.
- Who decides:
  - Operating system,
  - Program.

File Attributes

- **Name** – only information kept in human-readable form.

- **Type** – needed for systems that support different types.

- **Location** – pointer to file location on device.

- **Size** – current file size.

- **Protection** – controls who can do reading, writing, executing.

- **Time, date, and user identification** – data for protection, security, and usage monitoring.

⟹ Information about files is kept in the directory structure, which **resides on the disk**.

File Types: Name and Extension

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

File Control Block

file permissions
file dates (create, access, write)
file owner, group, ACL
file size
file data blocks

Users and Passwords

- In Unix, globally readable file `/etc/passwd` maps each user's name to an integer number, to a home directory, and to a shell.

- `/etc/shadow` maps each user to a an encrypted (actually, *hashed*) password. Not readable to prevent "password guessing" attacks.

Groups

- In Unix, `/etc/group` maps each group's name to an integer number and to a collection of users
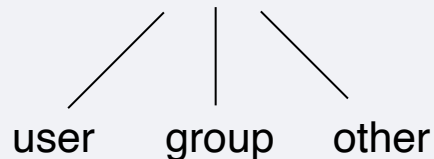
```
chgrp game mine
        |      |
      group  file
```

Access Control: Protection Bits

- Mode of access: **read (r), write (w), execute (x)**
- Three classes of access

		rwx	
a) **owner access** user	$7 \Rightarrow$	1 1 1	(octal $\Rightarrow$ binary)
b) **group access**	$6 \Rightarrow$	**rwx** 1 1 0	
c) **public access** other	$1 \Rightarrow$	**rwx** 0 0 1	

- For a particular file (say *game*) or subdirectory, define an appropriate access.

chmod 761 mine

    user   group   other

chmod u+rwx mine

    user

Access Control (Protection)

- **File owner/creator should be able to control:**
  - what can be done,
  - by whom.

  Discretionary Access Control (DAC)

- **Types of access:**
  - Read,
  - Write,
  - Execute,
  - Append,
  - Delete,
  - List.

Users decide **what** to share and **with whom** to share (no policy): flexible**.**

Users can make bad decisions and share **with the wrong people what they shouldn't be sharing.**

Access Control (Protection)

- **Mandatory Access Control (MAC):**
  - **System policy:** files tied to access levels = (public, restricted, confidential, classified, top-secret).

  - Process also has access level: can read from and write to all files at same level, can only read from files below, can only write to files above.

A policy that guarantees information access rights **for each user** based on their needs is determined in advance and strictly enforced by the system.

Users are locked into the policy and changes in access rights requires a revision of the policy and the action of an administrator.

Access Control (Protection)

- **Role-Based Access Control (RBAC):**
  - **System policy:** defines **"roles"** (generalization of the Unix idea of groups).
  - Roles are associated with access rules to sets of files and devices.
  - A process can change roles (in a pre-defined set of possibilities) during execution.

A policy determines information access rights **for each role** based on needs is determined in advance and is strictly enforced by the system. Users can be granted access to multiple roles. The assignment of users to roles can be changed more easily.

The a change in the rights for each roles requires a change in the policy and actions carried out by an administrator

File Operations

- **Create**.

- **Write**.

- **Read**.

- **Random access**.

- **Delete**.

- **Append**.

- **Truncate** (reset size to 0, keep current attributes).

- **Open(F_i)** – search the directory structure on disk for entry F_i, and move the content of entry to memory.

- **Close (F_i)** – move the content of entry F_i in memory to directory structure on disk.

Open Files

Several pieces of data are needed to manage open files:

- **Open-file table**: tracks open files

- **File pointer:** pointer to last read/write location, per process that has the file open

- **File-open count:** counter of number of times a file is open – to allow removal of data from open-file table when last processes closes it

- **Disk location of the file:** cache of data access information
- **Access rights:** per-process access mode information

Open File Locking

➡ Provided by some operating systems and file systems

- Similar to reader-writer locks
- **Shared lock** similar to reader lock – several processes can acquire concurrently
- **Exclusive lock** similar to writer lock

➡ Mediates access to a file

➡ Mandatory or advisory:

- **Mandatory** – access is denied depending on locks held and requested
- **Advisory** – processes can find status of locks and decide what to do

Access Methods

- **Sequential Access** { *read next*
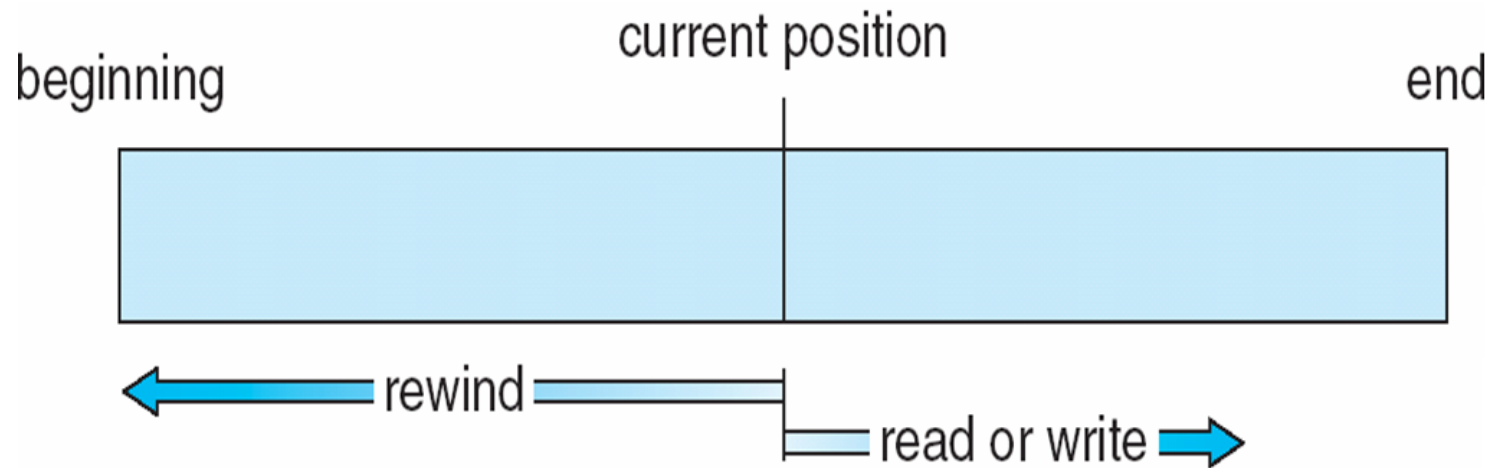 *write next*
  reset
  no *read after last write*
  (rewrite)

- **Direct Access** { *read n*
  write n
  position to n
  read next
  write next
  rewrite n

*n* = relative block number

Sequential-access File

Simulation of Sequential Access on a Direct-access File

sequential access	implementation for direct access
reset	$cp = 0;$
read next	*read cp*; $cp = cp + 1;$
write next	*write cp*; $cp = cp + 1;$

Other Access Methods

- Can be built on top of base methods
- General involve creation of an index for the file
- Keep index in memory for fast determination of location of data to be operated on (consider UPC code plus record of data about that item)
- If too large, index (in memory) of the index (on disk)
- IBM indexed sequential-access method (ISAM)
- Small master index, points to disk blocks of secondary index
- File kept sorted on a defined key
- All done by the OS
- VMS operating system provides index and relative files as another example (see next slide)

Index and Relative Files
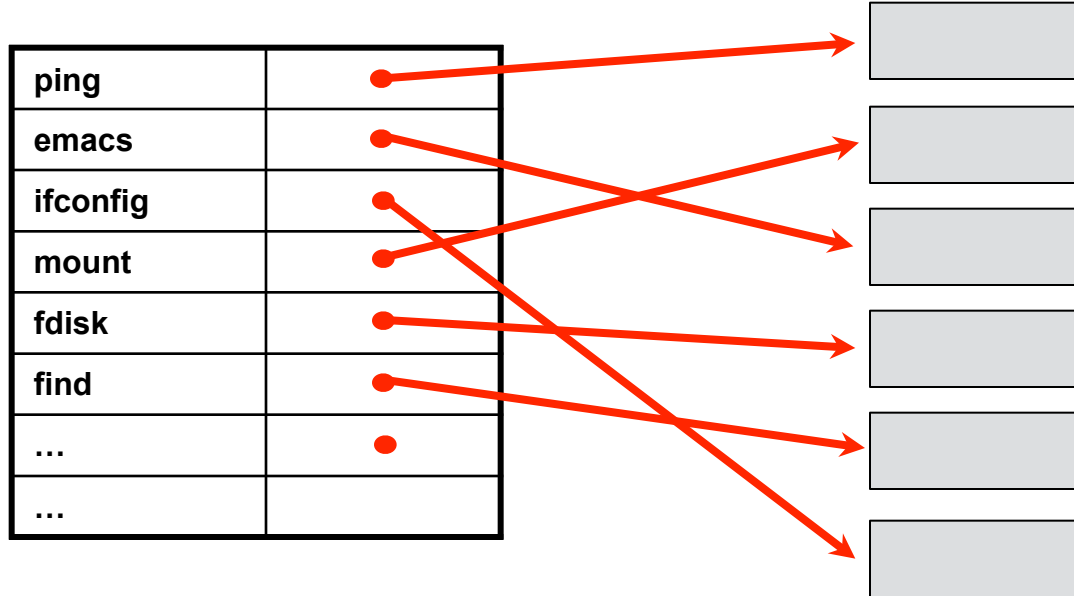

index file

relative file
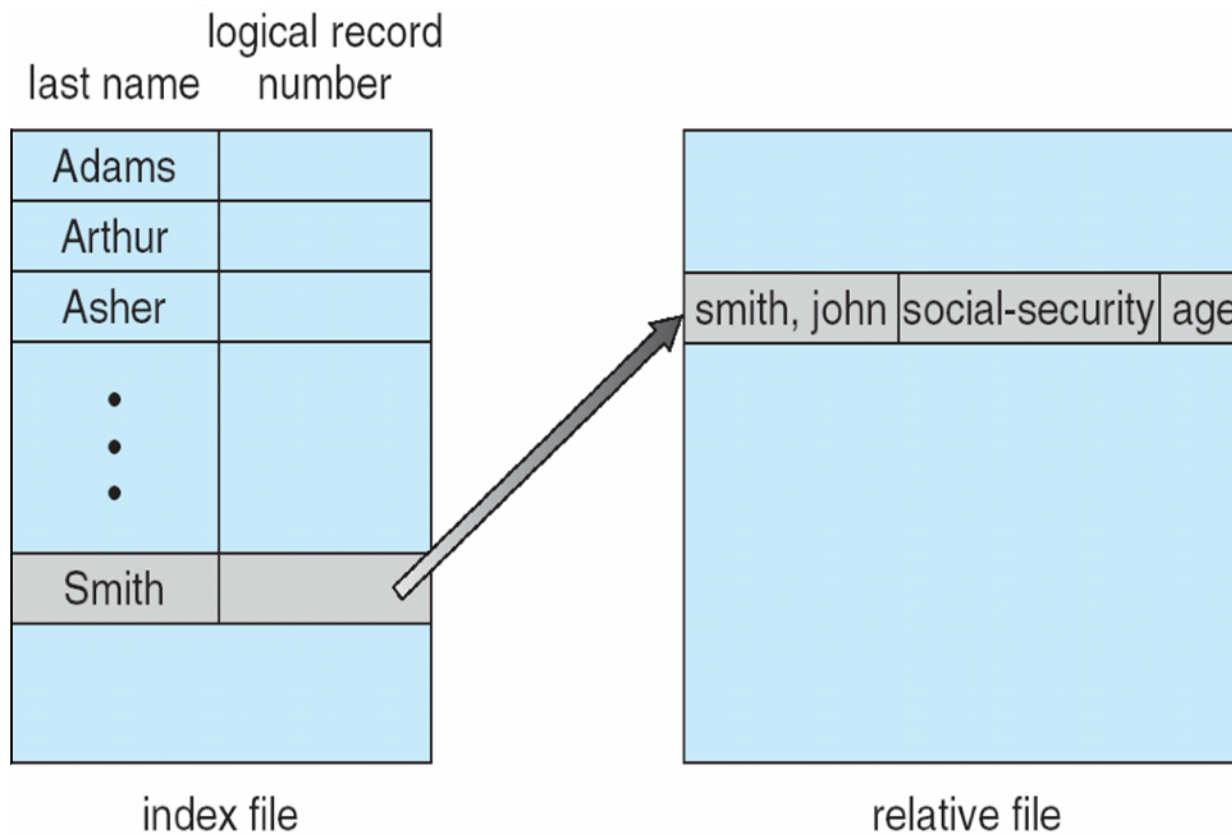
Operations on Directories

- Search for a file.
- Create a file.
- Delete a file.
- List a directory.
- Rename a file.
- Traverse the file system.

mkdir, rmdir, ls, mv, …

Directory Structure

**Directory:** a symbol table that translates file names into directory entries.

Both the directory structure and the files reside on disk.
Backups of these two structures are kept on tapes.

Links

In Unix, the command `ln` allows the user to create symbolic and hard links.

- `symbolic link` - a directory entry that points to another directory entry.

- `hard link` - "another" directory entry that points to a file object that already appeared in the directory.

Links
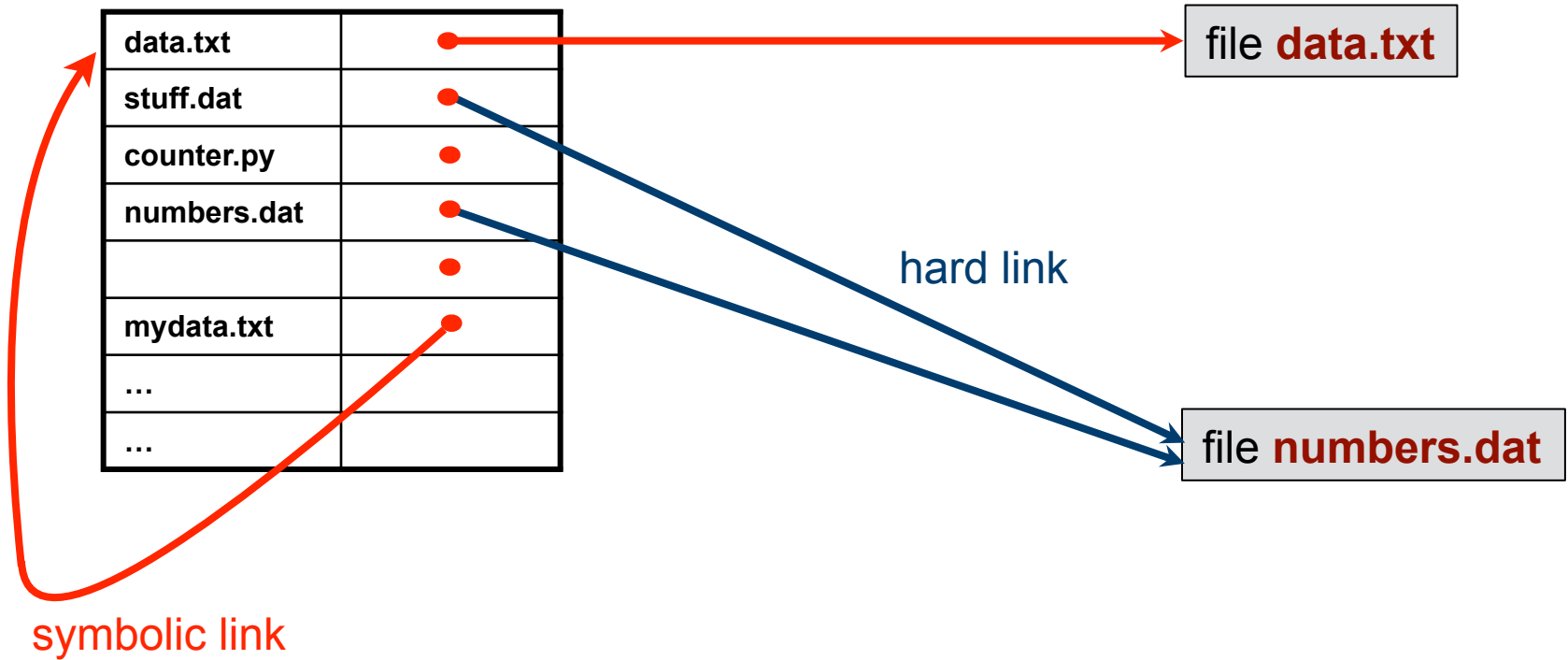
RTFMP Interlude

```
LN(1) User Commands


NAME
      ln - make links between files


SYNOPSIS
      ln [OPTION]... [-T] TARGET LINK_NAME    (1st form)

…

      -s, --symbolic
            make symbolic links instead of hard links
```
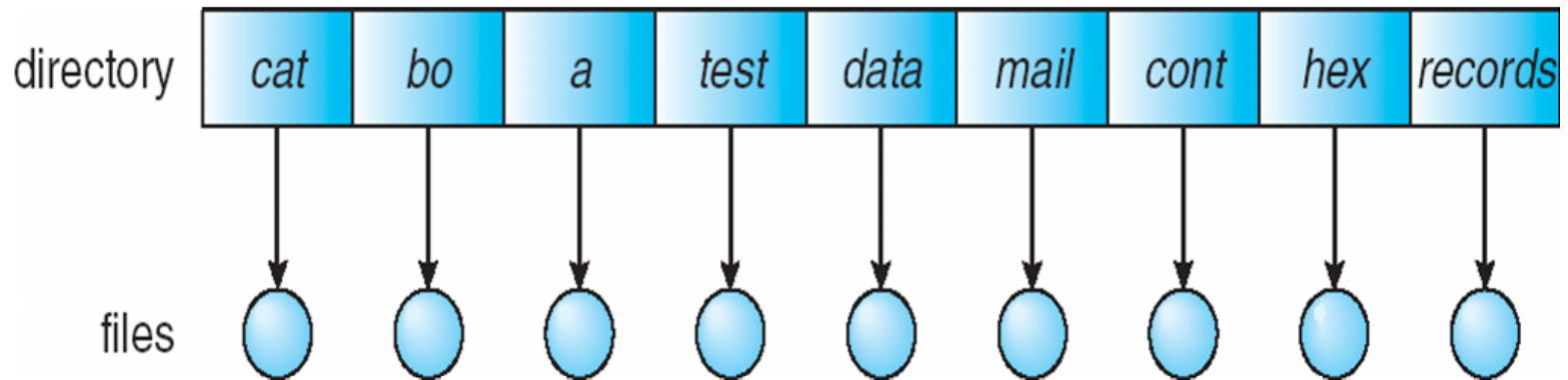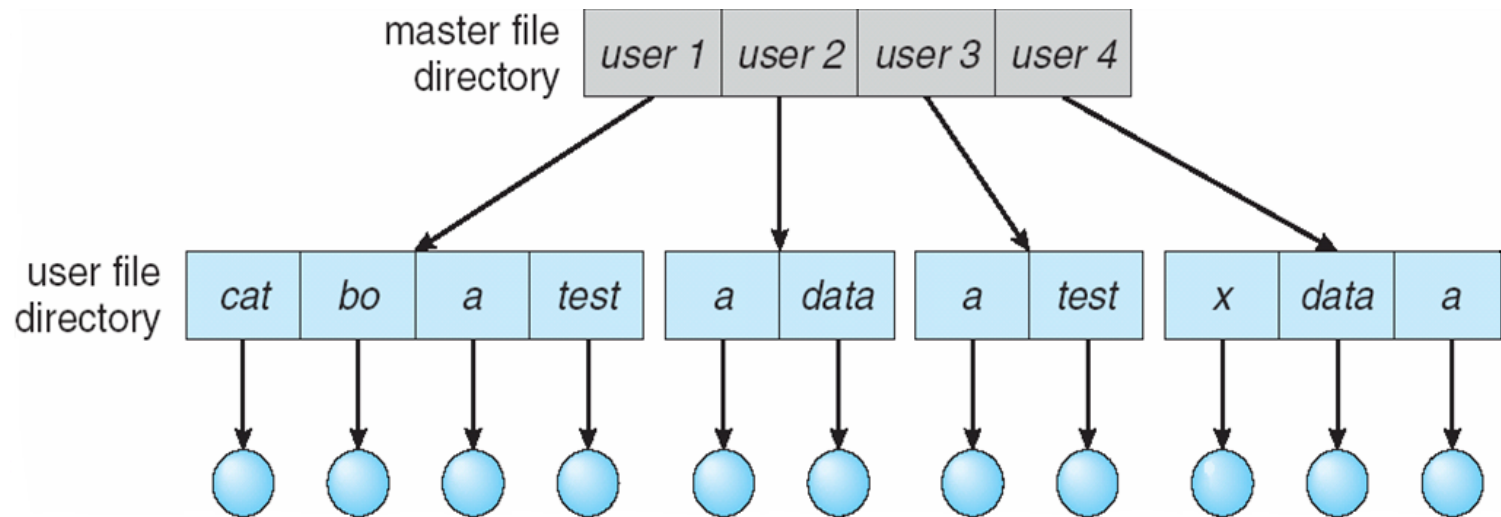
Goals of Directory Logical Organization

- **Efficiency** – locating a file quickly.

- **Naming** – convenient to users.
  - Two users can have same name for different files.
  - The same file can have several different names.

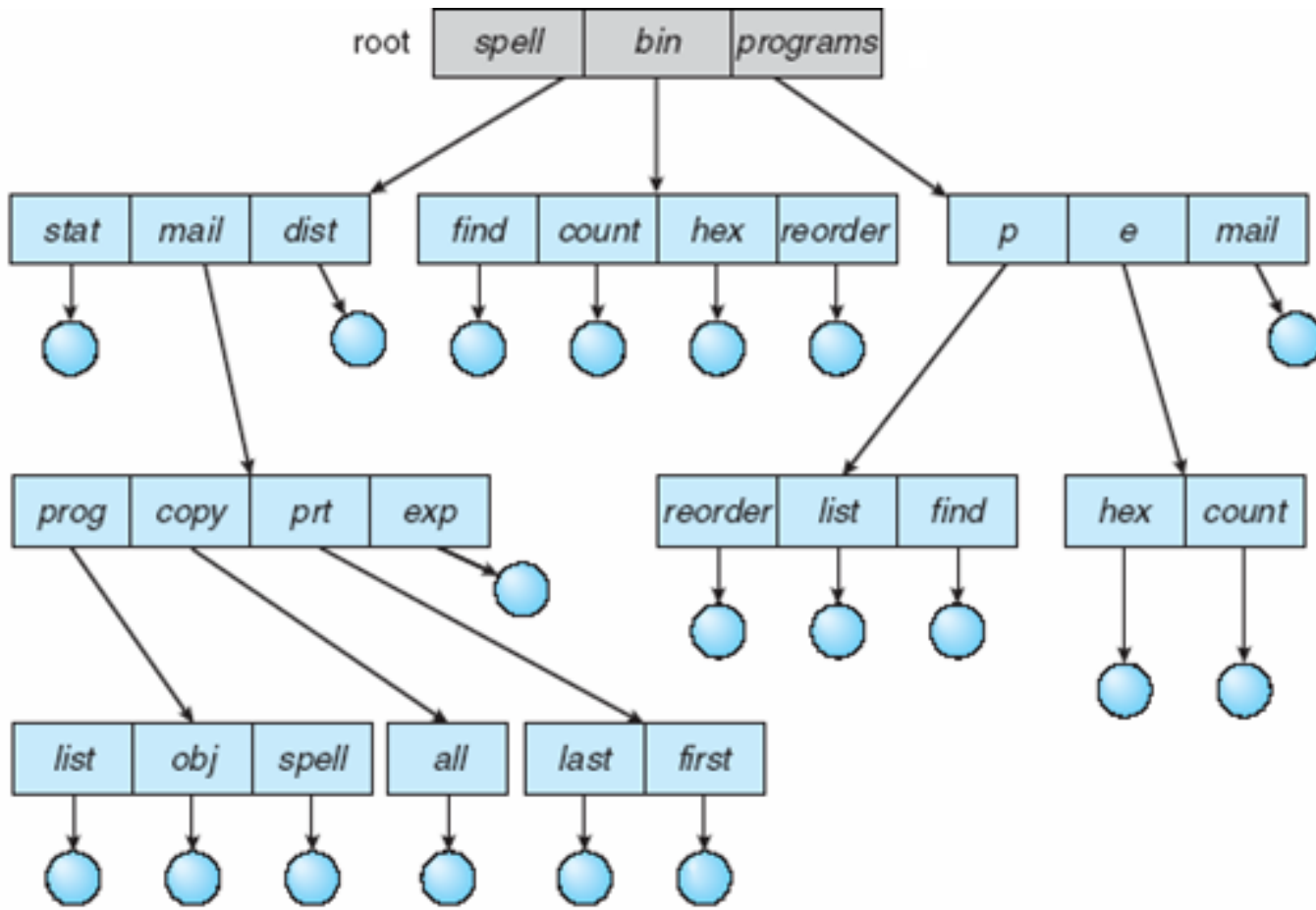- **Grouping** – logical grouping of files by properties, (e.g., all Java programs, all games, …)

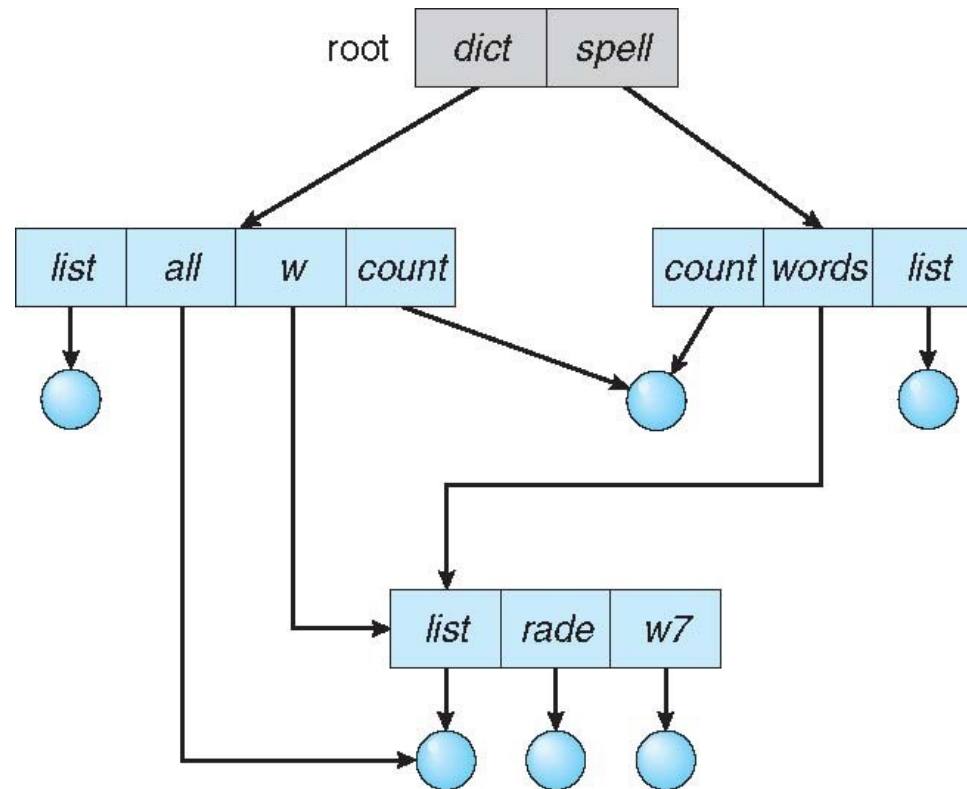# Single-Level Directory

Two-Level Directory

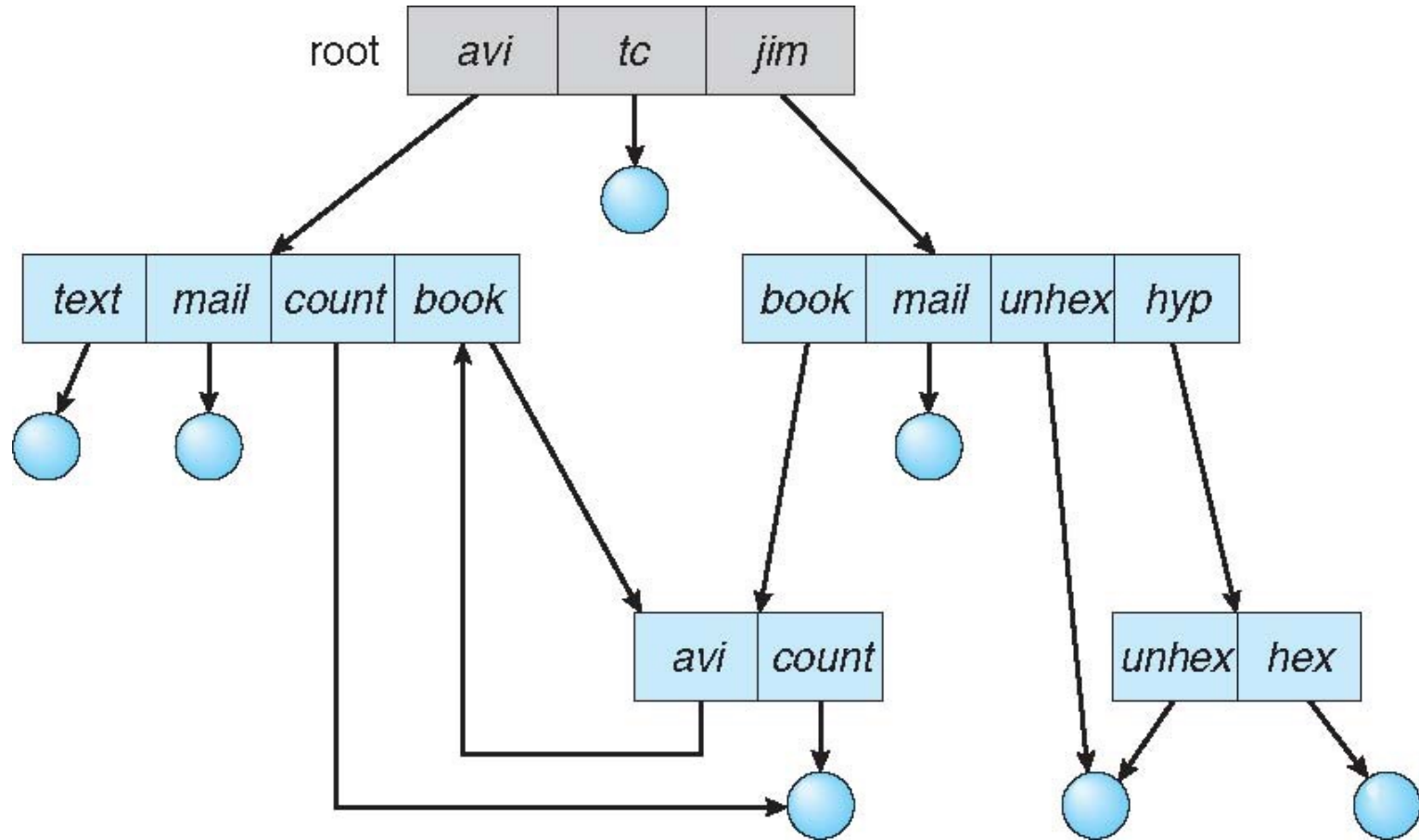# Tree-Structured Directories

Acyclic-Graph Directories

General Graph Directory

Single-Level Directory

A single directory for all users.
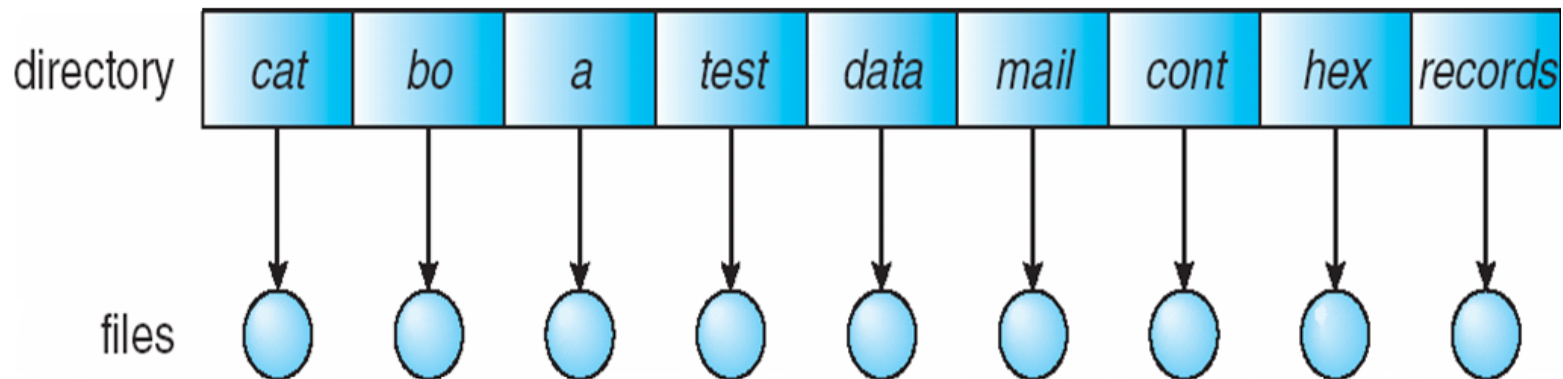
Drawbacks:

Naming problem

Grouping problem

Two-Level Directory

A separate directory for each user.

- Path name.
- Can have the same file name for different user.
- Efficient searching.
- No grouping capability.

Tree-Structured Directories

Tree-Structured Directories (Cont.)

- Efficient searching.

- Grouping Capability.

- Current directory (working directory):
  - **cd** /spell/mail/prog,
  - **type** list.

Tree-Structured Directories (Cont.)
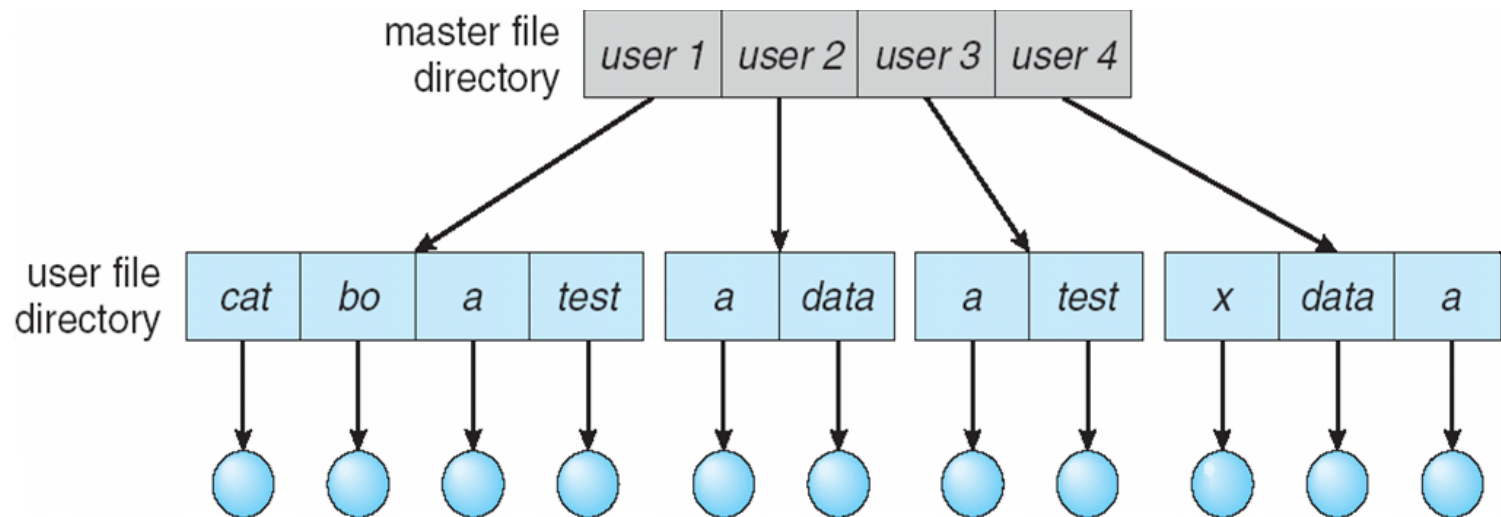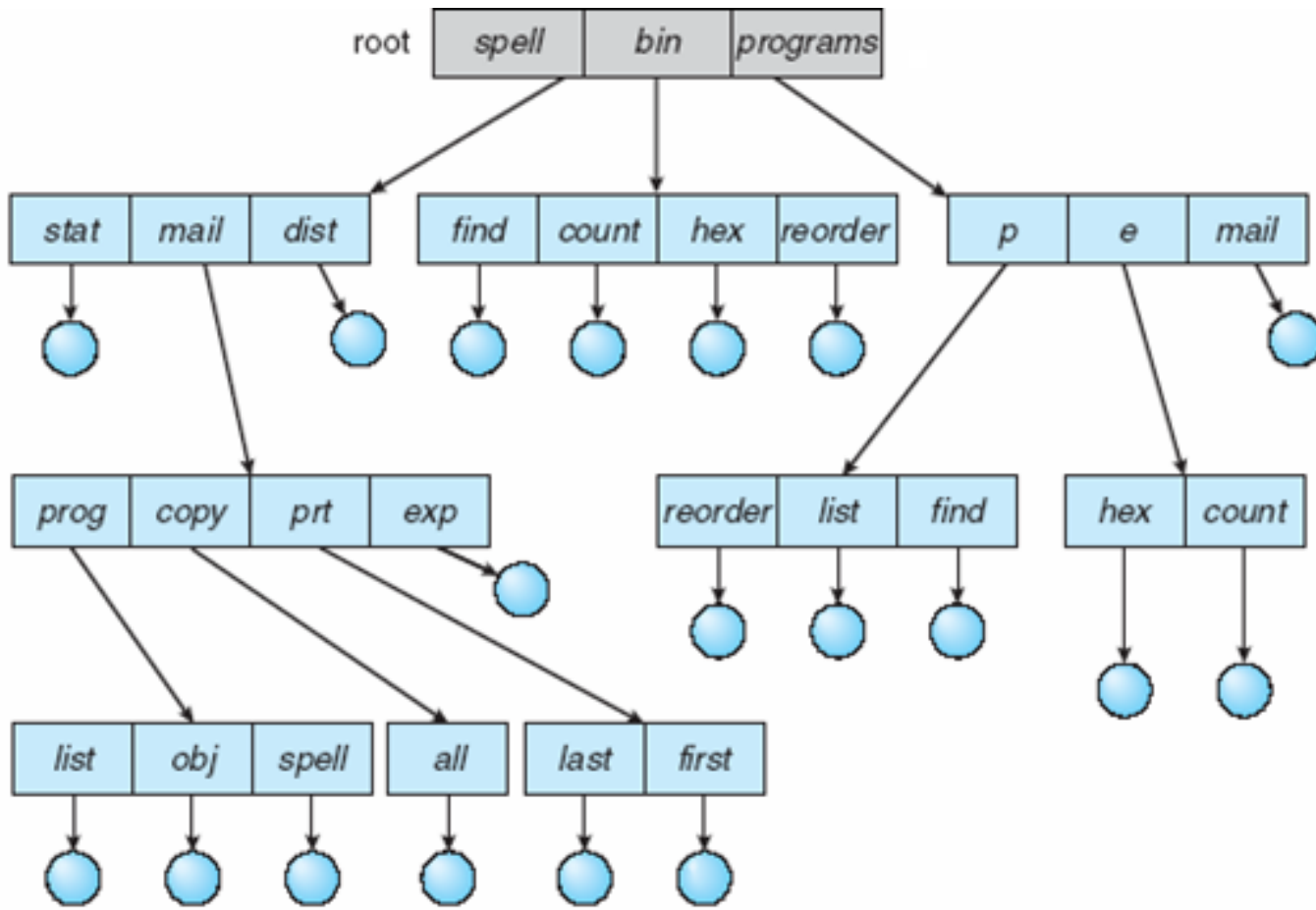
- **Absolute** or **relative** path name.
- Creating a new file is done in current directory by default.
- Delete a file

  rm <file-name>

- Creating a new subdirectory is done in current directory.

  mkdir <dir-name>

Example: if in current directory **/mail**

  mkdir count

```
           ┌──────┐
           │ mail │
           └──────┘
              │
  ┌──────┬──────┬─────┬─────┬───────┐
  │ prog │ copy │ prt │ exp │ count │
  └──────┴──────┴─────┴─────┴───────┘
```
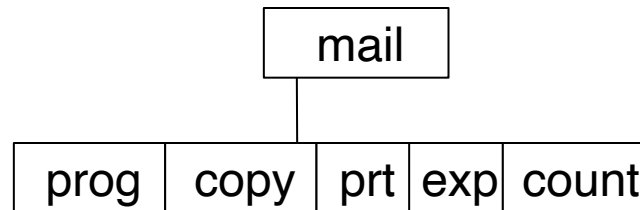
rm -rf . ⇒ doesn't mean "read mail really fast"

Acyclic-Graph Directories

Have <u>shared</u> subdirectories and files
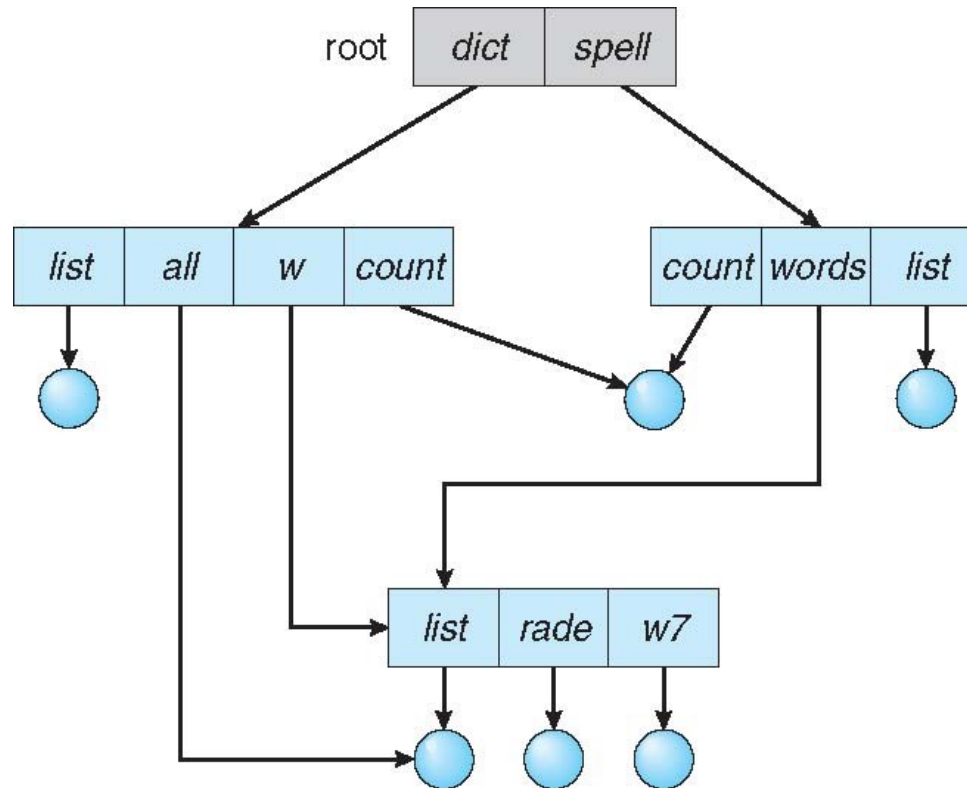
links: { soft (symbolic)

hard

Unix: **In** (read man page);

need to keep a reference count on each file or directory.

Acyclic-Graph Directories (Cont.)

- Different names (<u>aliasing</u>) for the same file or directory.
- Must be careful with removals to avoid dangling pointer.

  Solutions:

  – Backpointers, so we can delete all pointers. Variable size records a problem.
  – Backpointers using a daisy chain organization.
  – Entry-hold-count solution.

General Graph Directory

General Graph Directory (Cont.)

- **How do we guarantee no cycles?**
  - Allow only links to files, never to ~~subdirectories~~.
  - Garbage collection.
  - Every time a new link is added use a cycle detection algorithm to determine whether it is OK.

Disk Structure

- Disk can be subdivided into *partitions* which are logical divisions inside a physical drive

- Partitions are also known as minidisks or slices

- A disk or partition can be used *raw* (without a file system) or *formatted with a file system*

- Once a partition is formatted with a file system, it is seen a *volume*

- As well as *general-purpose file systems* there are many *special-purpose file systems*, frequently all within the same operating system or computer

- Disks or partitions can be protected against failure by *Redundant Array of Independent Disks* (RAID)

Why should one partition a drive?

- To keep a disk organized systematically

- To create different compartments for different operating systems (for instance: dual boot your machine with Windows and Linux)

- To create a compartment for *data* that needs to be backed up

- To create compartments for reliability: if one gets corrupted, others may stay intact

Partitions and Directories
(File system organization)
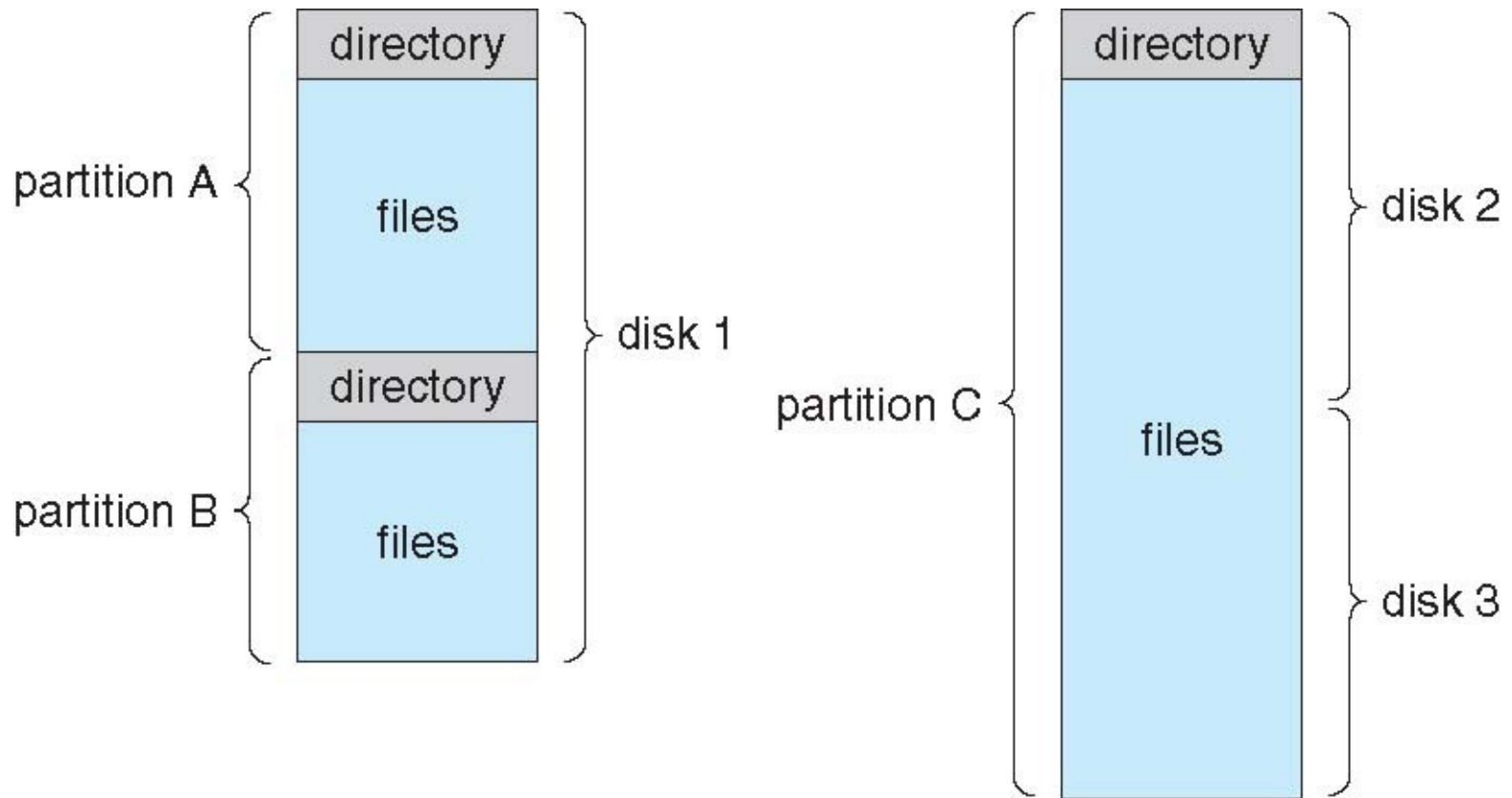
Practical information for those interested in Linux: Partitioning Recommendations

Directory Implementation

The directory is a **<u>symbol table</u>** that maps file names to pointers that lead to the blocks comprising a file.

- **Linear list** of file names with pointer to the data blocks:
  - simple to program, but…
  - time-consuming to execute.
- **Hash Table:**
  - decreases directory search time,
  - *collisions* – situations where two file names hash to the same location,
  - fixed size.

In-Memory File System Structures

file open

file read
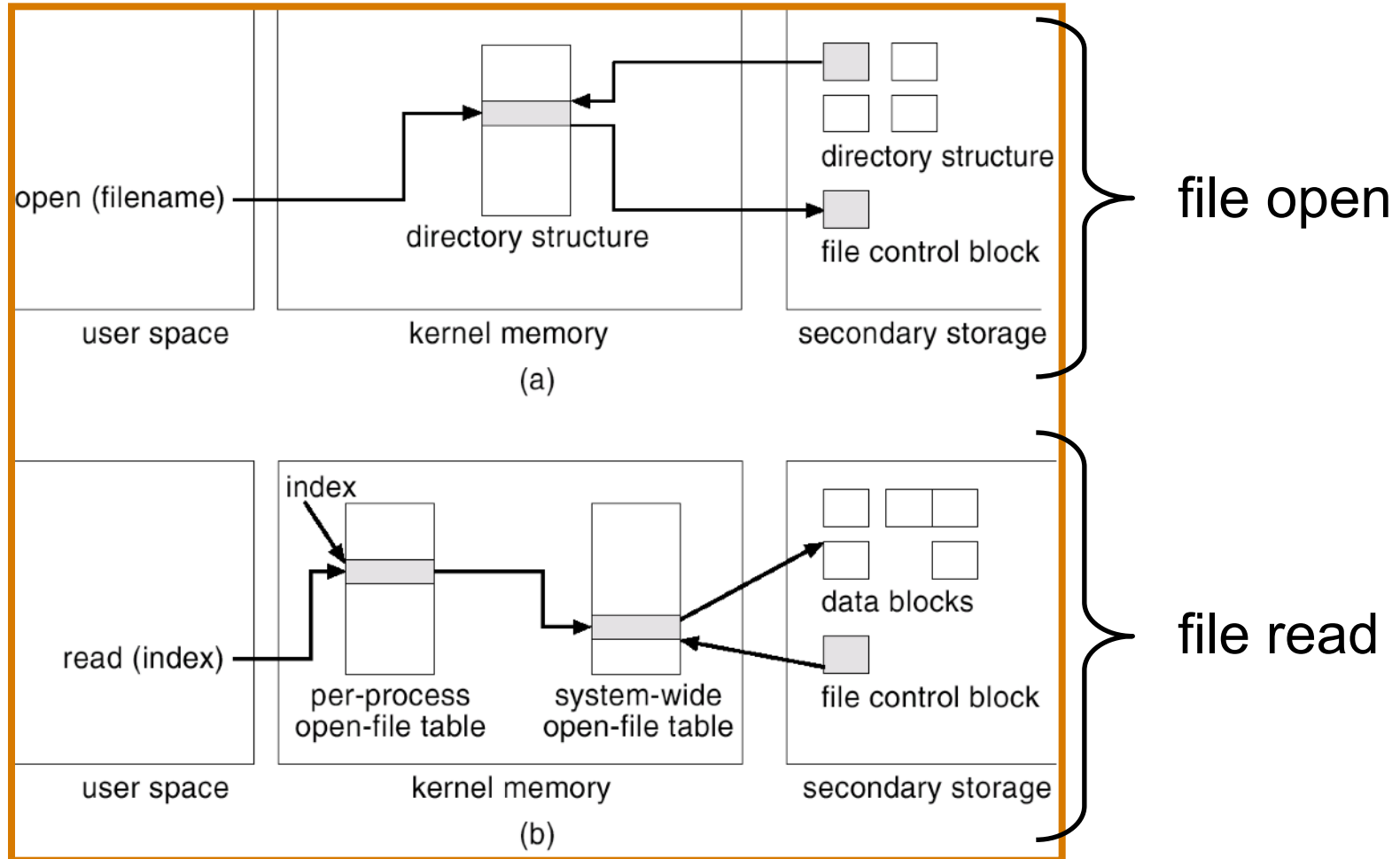
Virtual File Systems

- **Virtual File Systems (VFS)** provide an object-oriented way of implementing file systems.

- 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.

File System Mounting

- A file system (partition) must be mounted before it can be accessed. Mounting allows one to attach the file system on one device to the file system on another device.

- A unmounted file system needs to be attached to a mount point before it can be accessed.
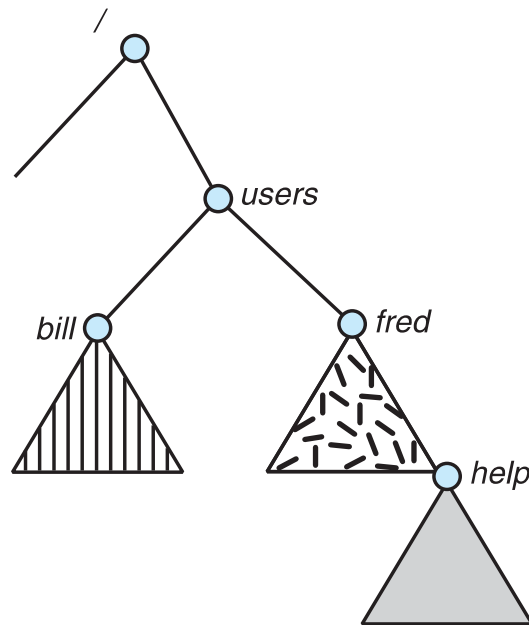
(a)

existing
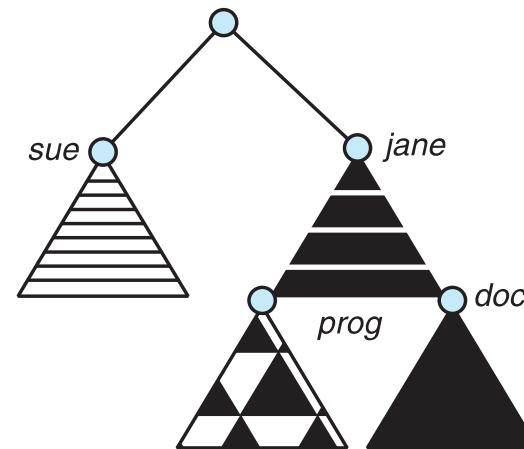
(b)

unmounted

File Sharing

- Sharing of files on multi-user systems is desirable.

- Sharing may be done through a *protection* scheme.

- On distributed systems, files may be shared across a network.

- Network File System (NFS) is a common distributed file-sharing method.

File-System Structure

- File structure:
  - Logical storage unit,
  - Collection of related information.
- File system resides on secondary storage (disks).
- File system is organized into layers.
- **File control block** – storage structure consisting of information about a file.

Layered File System
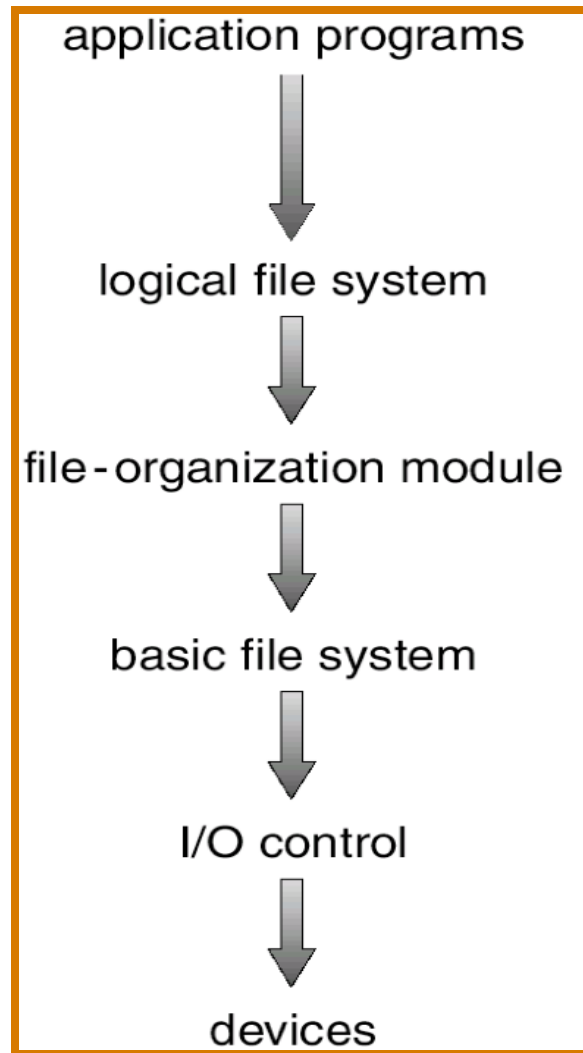
Schematic View of a Virtual File System
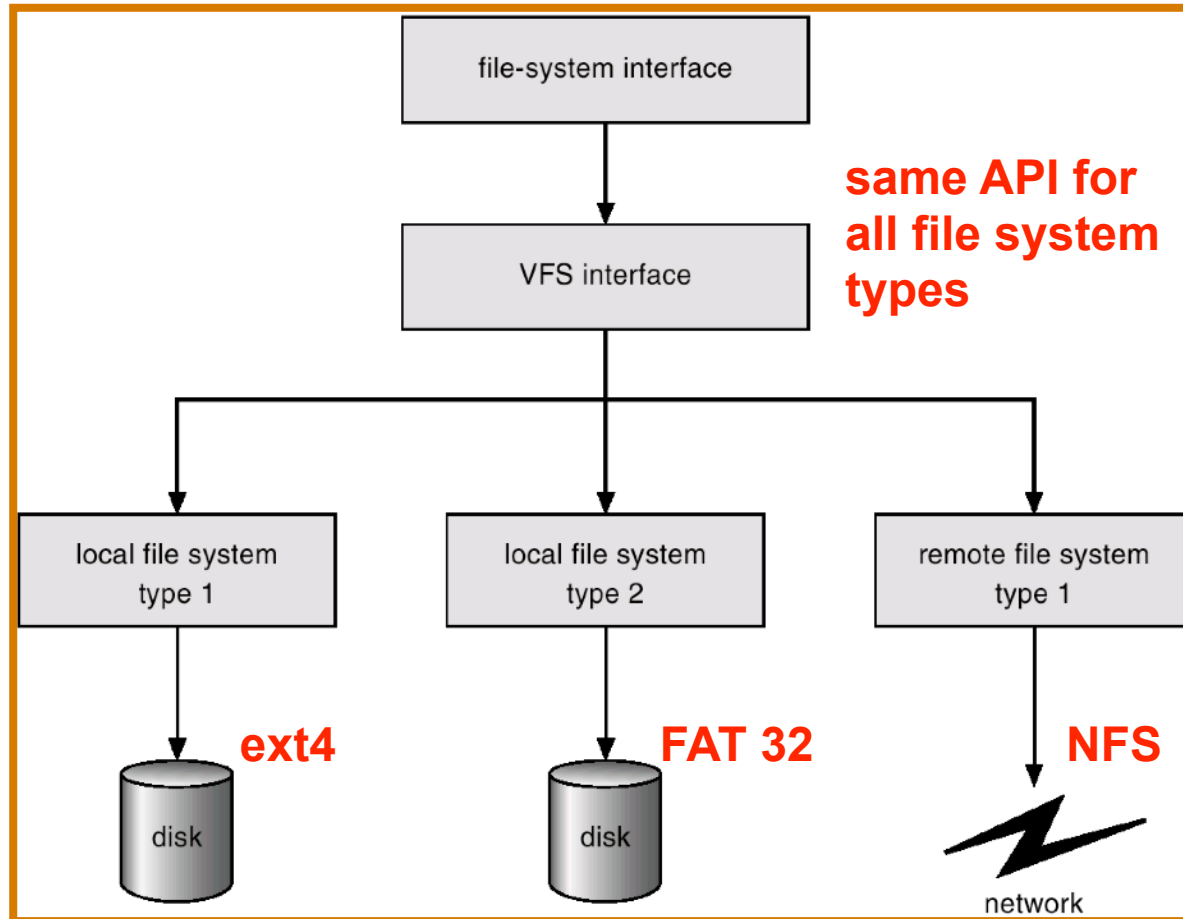
Disk Allocation Methods

An **allocation method** refers to how disk blocks are allocated for files. We'll discuss three options:

➡️ Contiguous allocation,

➡️ Linked allocation,

➡️ Indexed allocation.

Contiguous Allocation

- Each file occupies a set of **contiguous blocks** on the disk.

- Simple: only starting location (block #) and length (number of blocks) are required.

- Suitable for **sequential** and **random** access.

- Wasteful of space: dynamic storage-allocation problem.

- Files cannot grow unless more space than necessary is allocated when file is created (clearly this strategy can lead to **internal fragmentation**).

Contiguous Allocation of Disk Space

directory

file	start	length
count	0	2
tr	14	3
mail	19	6
list	28	4
f	6	2
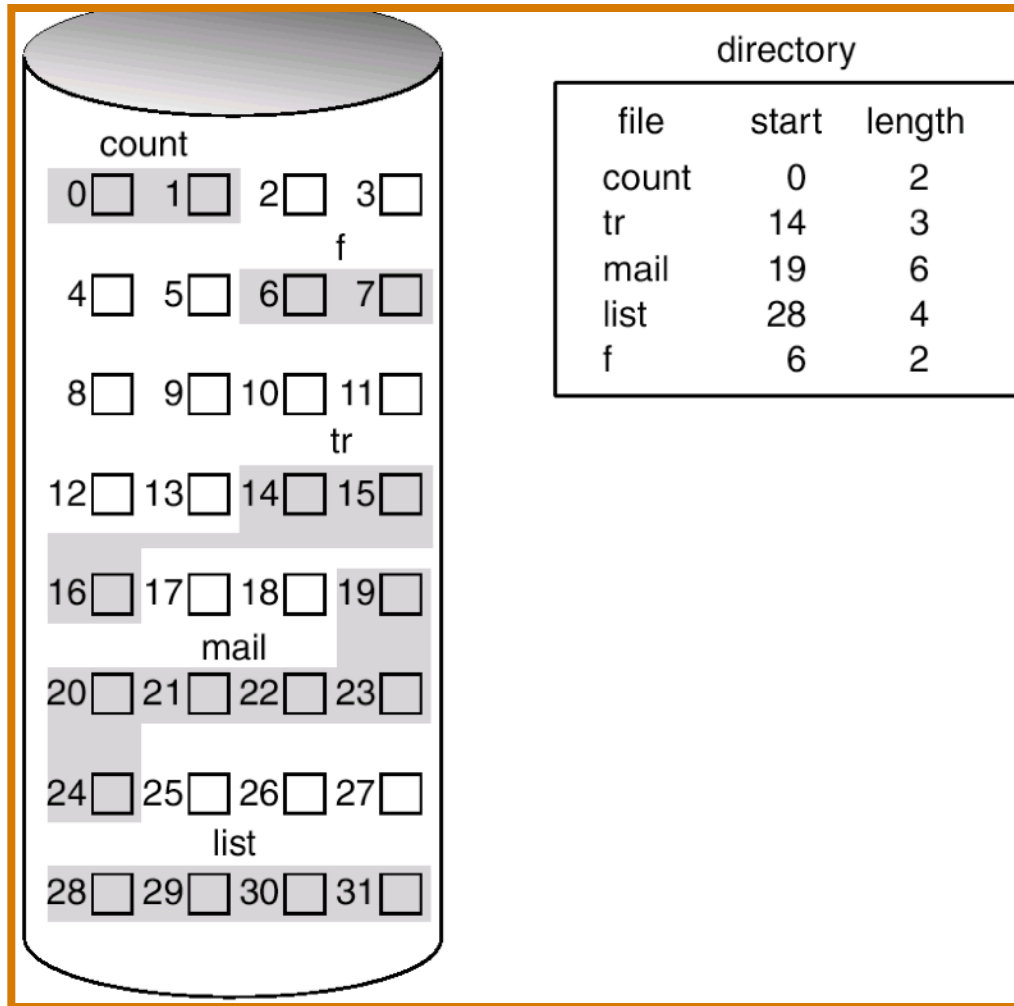
To deal with the dynamic allocation problem (external fragmentation), the system should periodically **compact** the disk.

Compaction may take a long time, during which the system is effectively **down**.

To deal with possibly growing files, one needs to pre-allocate space larger than required at the initial time which leads to **internal fragmentation**.

Does this disk allocation, **in general**, suffer from **internal fragmentation**?
Does this method suffer from **external fragmentation**?

Extent-Based Systems

- Many newer file systems (i.e. Veritas File System) use a modified contiguous allocation scheme.

- Extent-based file systems allocate disk blocks in **extents**.

- An **extent** is a contiguous set of blocks. Extents are allocated for each file. A file consists of one or more extents.

- Extents can be added to an existing file that needs space to grow. A block can be found given by the location of the first block in the file and the block count, plus a link to the first extent.

Linked Allocation

Each file is a linked list of disk blocks.

Simple: need only starting address.

Overhead: each block links to the next.

Space cost to store pointer.

Time cost to read one block to find the next.

Internal fragmentation, but not external.

Sequential access comes naturally, random does not.
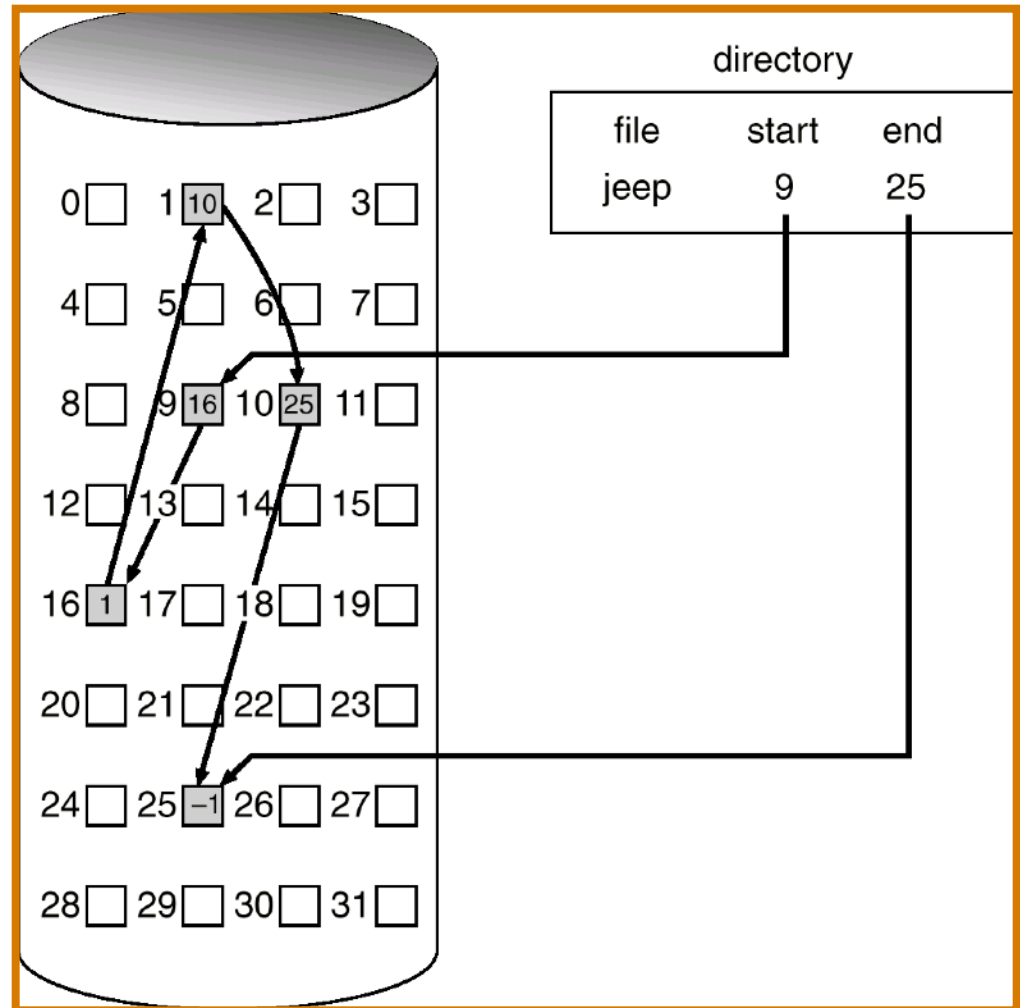
File-Allocation Table (FAT)

directory entry

test	· · ·	217

name   start block

0

217   618

339   end-of-file

618   339

no. of disk blocks   −1

FAT
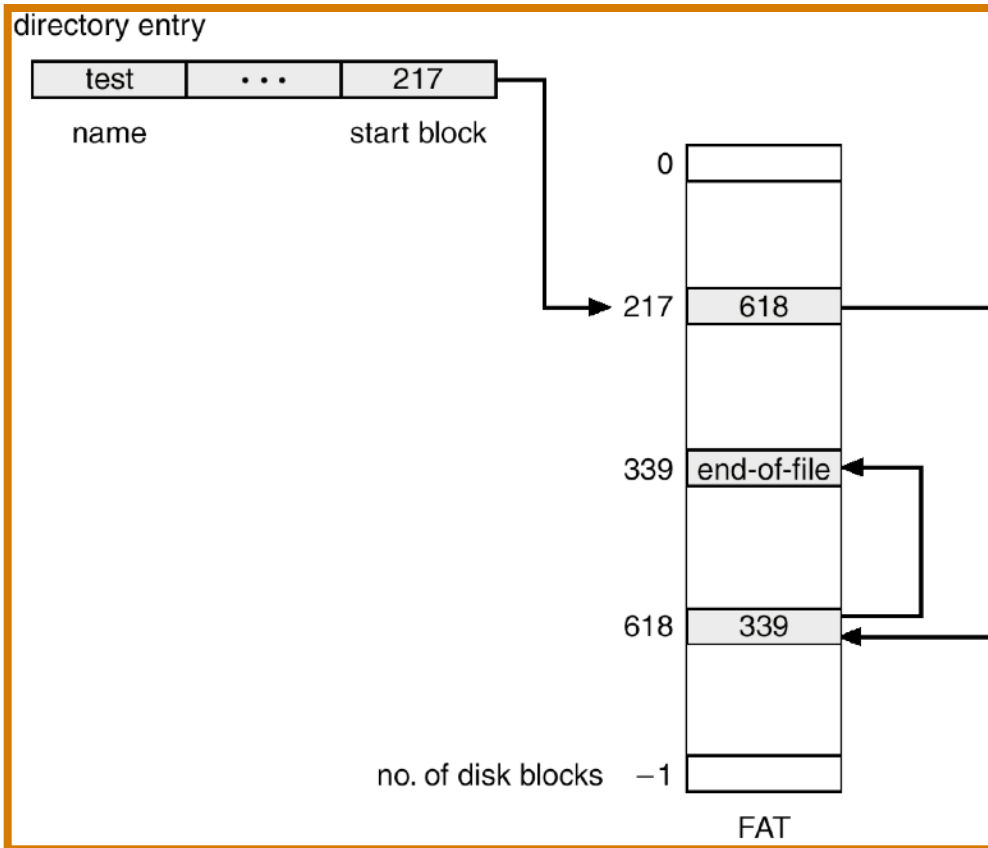
Simple and efficient: One entry for each block; indexed by block number. The table is implements the list linking the blocks in a file.

Growing a file is easy: find a free block and link it in.

Random access is easy.

The FAT should be cached in memory.

If the FAT is not cached in memory, a considerable number of disk seeks happens.

Used by MS-DOS and OS/2.

Indexed Allocation

Brings all pointers together into an *index block.*

One index block *per file*.

Random access comes easy.

Dynamic access without external fragmentation, but have overhead of index block.

Wasted space: how large should an index block be to minimize the overhead?
- linked index blocks
- multilevel index
- combined scheme
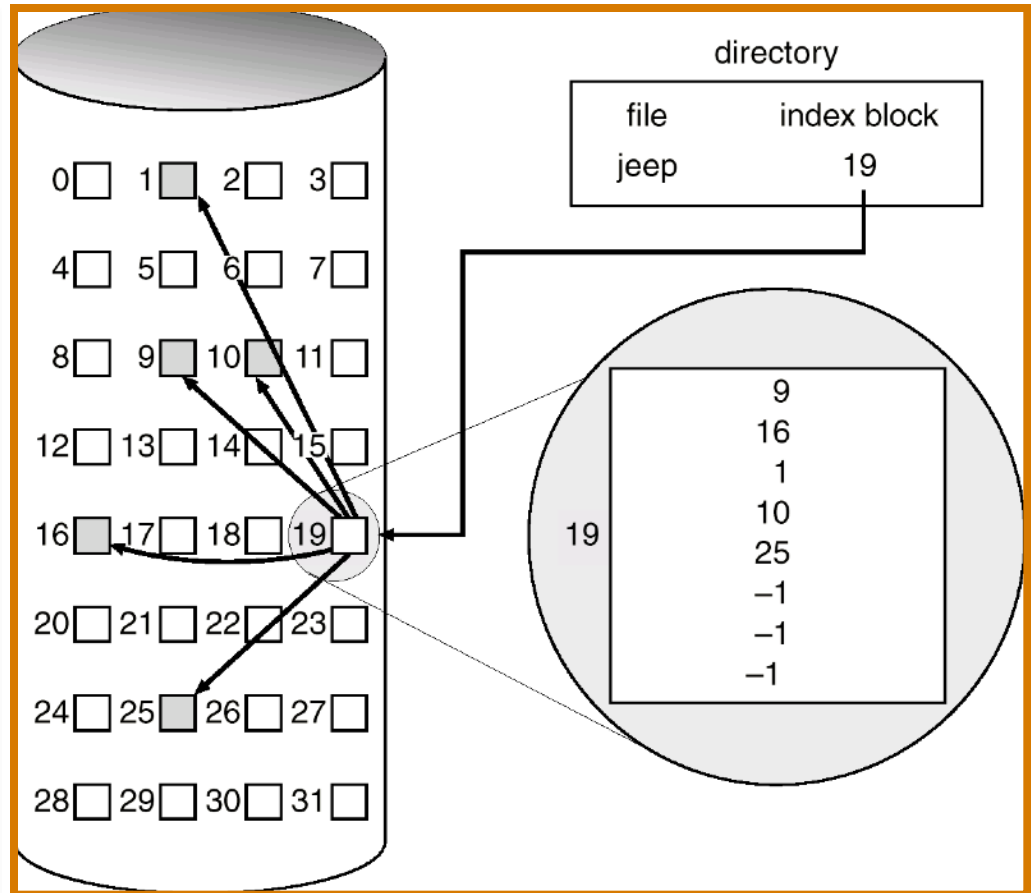
Combined Scheme: UNIX
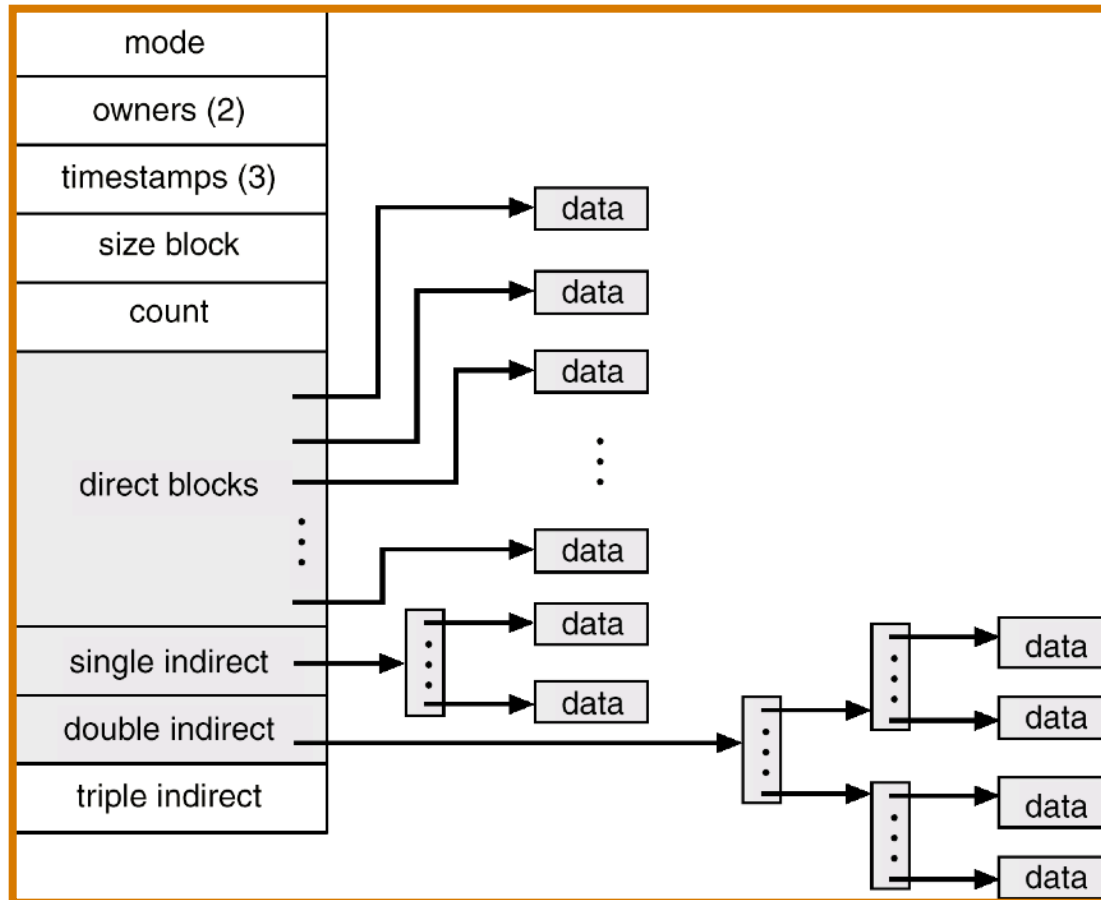
If file is small enough, use only **direct blocks** pointers.

If number of blocks in file is greater than the number of direct block pointers, use **single**, **double**, or **triple indirect**.

Additional levels of indirection increase the number of blocks that can be associated with a file.

Index blocks can be cached in memory, like FAT. **Access to data blocks, however, may require many disk seeks**.

Free-Space Management

- **Bit vector** (1 bit per disk block)
- **Linked list** (free list)
- **Grouping**
  - like linked list: first free block has n block addresses (the n-1 addresses are free blocks, the nth is the address of a block with the next bundle of addresses)
- **Counting**
  - like linked list, but each node points to a cluster of contiguous, free blocks

The OS can cache in memory the free-space management structures for increased performance. Depending on disk size, this may not be easy.

Bit Vector

Bit Vector (or Bit Map)

101010111111010011111000001111...

| 0 | 1 | 2 | 3 | 4 | 5 | 6 | 7 | 8 | 9 |

| 10 | 11 | 12 | 13 | 14 | 15 | 16 | 17 | 18 | 19 |

| 20 | 21 | 22 | 23 | 24 | 25 | 26 | 27 | 28 | 29 |

...

Linked List

Grouping

first free block

1

0	1	2	3	4	5	6	7	8	9
	2								
	3								
	4								
	5								
	11								

10	11	12	13	14	15	16	17	18	19
	12								
	13								
	14								
	15								
	16								
	17								
	20								

20	21	22	23	24
21				
22				
25				

...

Grouping

first free block

| 1 |

| 0 | 1 | 2 | 3 | 4 | 5 | 6 | 7 | 8 | 9 |

Block 1 contains:
2
3
4
5
(11)

| 10 | 11 | 12 | 13 | 14 | 15 | 16 | 17 | 18 | 19 |

Block 11 contains:
12
13
14
15
16
17
(20)

| 20 | 21 | 22 | 23 | 24 |

Block 20 contains:
21
22
(25)

...

Counting

first free block

1, 5

0	1	2	3	4	5	6	7	8	9
	12, 7								

10	11	12	13	14	15	16	17	18	19
		25, 4							

20	21	22	23	24	25	26	27	28	29
					31, 10				

…

Counting

first free block

1, 5

0	1	2	3	4	5	6	7	8	9
	12, 7								

10	11	12	13	14	15	16	17	18	19
		25, 4							

20	21	22	23	24	25	26	27	28	29
					31, 10				

...

Recovery

- **Consistency checking** – compares data in directory structure with data blocks on disk, and tries to fix inconsistencies. (Takes time!)

- **What events or failures can cause file system inconsistency?**

- **Philosophy:** Allow structures to break and provide ways to repair them.

- Use system programs to *back up* data from disk to another storage device (floppy disk, magnetic tape).

- Recover lost file or disk by *restoring* data from backup.

Log-Structured File Systems

- **Log-structured** (or journaling) file systems record each update to the file system as a **transaction**.

- All transactions are written to a **log**. A transaction is considered **committed** once it is written to the log (the file system may not yet be updated).

- The transactions in the log are asynchronously written to the file system. When the file system is modified, the transaction is removed from the log.

- After a system crash, all transactions that remained in the file system log are performed.