

File Systems: Interface and Implementation

CSCI 315 Operating Systems Design
Department of Computer Science

Notice: The slides for this lecture have been largely based on those from an earlier edition of the course text *Operating Systems Concepts, 8th ed.*, by Silberschatz, Galvin, and Gagne. Many, if not all, the illustrations contained in this presentation come from this source.



File System Topics

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

File Concept

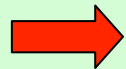
- A file is a named collection of related information recorded on secondary storage.
- **“Contiguous” logical** address space.
- File types:
 - Data:
 - numeric.
 - character.
 - binary.
 - Program (executable).

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 is maintained on the disk.

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.

File Types: Name and Extension

file type	usual extension	function
executable	exe, com, bin or none	read 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, rrf, doc	various word-processor formats
library	lib, a, so, dll, mpeg, mov, rm	libraries of routines for programmers
print or view	arc, zip, tar	ASCII or binary file in a format for printing or viewing
archive	arc, zip, tar	related files grouped into one file, sometimes com- pressed, for archiving or storage
multimedia	mpeg, mov, rm	binary file containing audio or A/V information

Access Methods

- **Sequential Access**

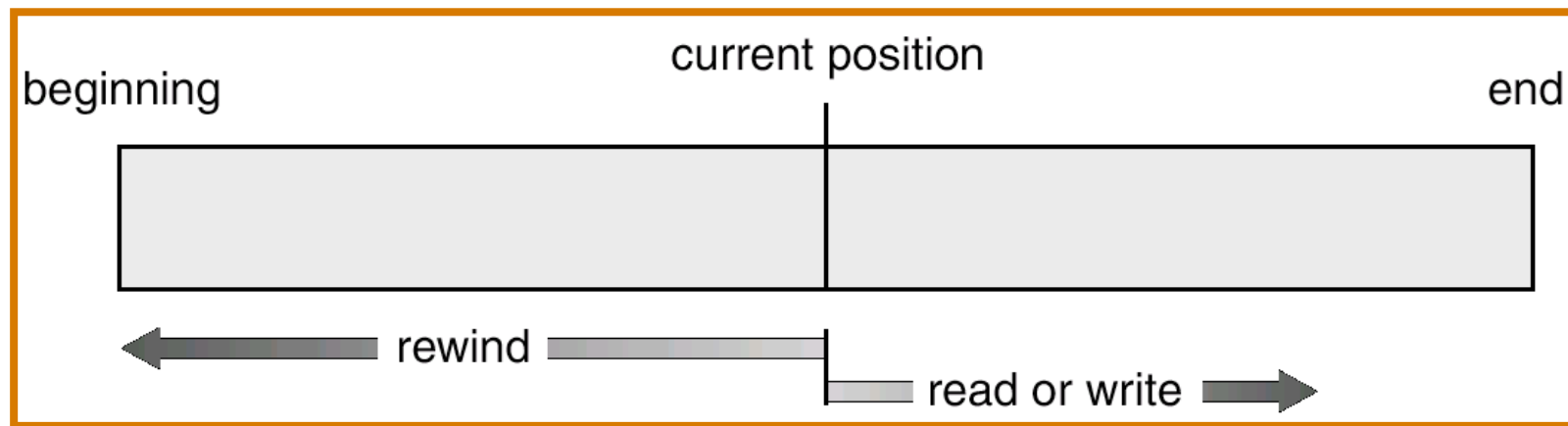
read next
write next
reset
no read after last write
(rewrite)

- **Direct Access**

read n
write n
position to n
read next
write next
rewrite n

n = relative block number

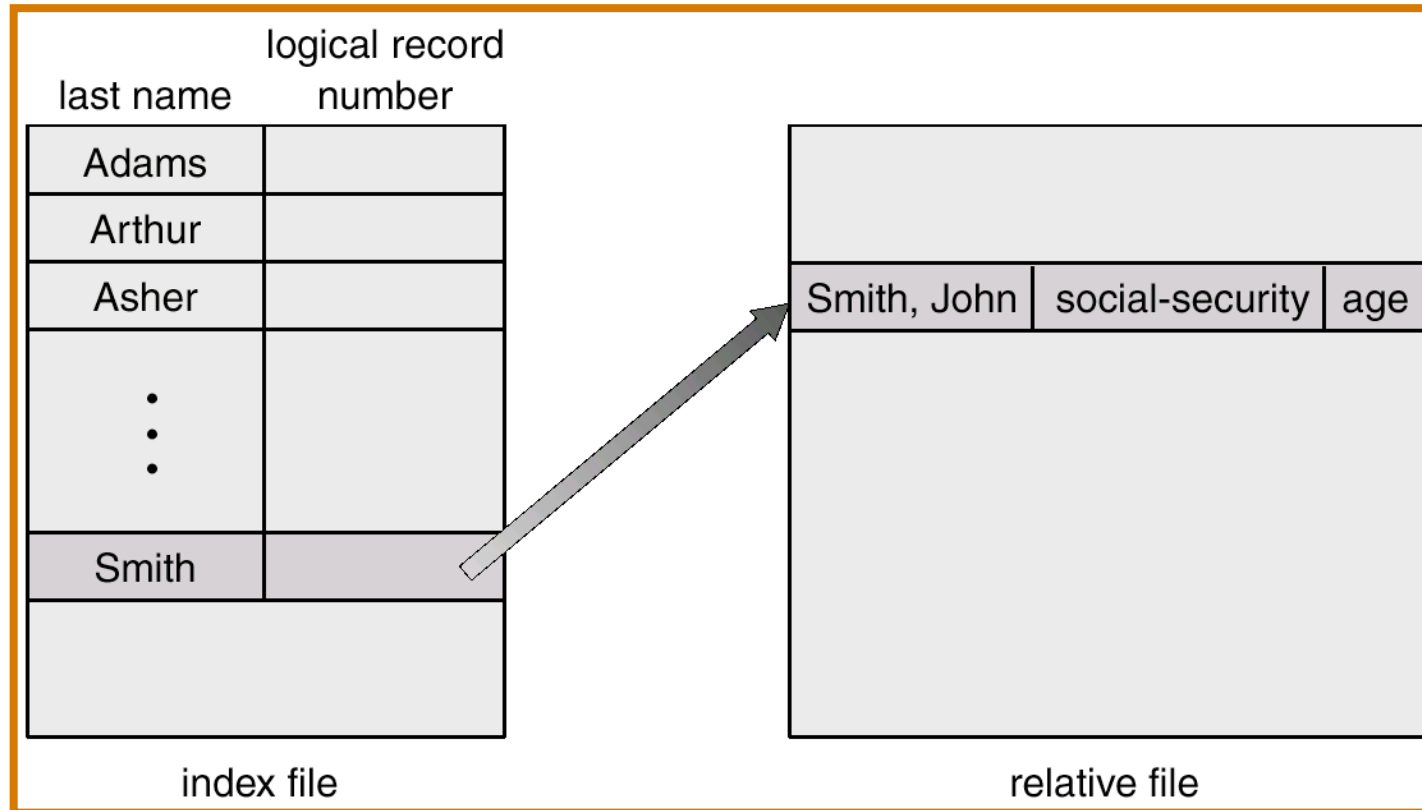
Sequential-access File



Simulation of Sequential Access on a Direct-access File

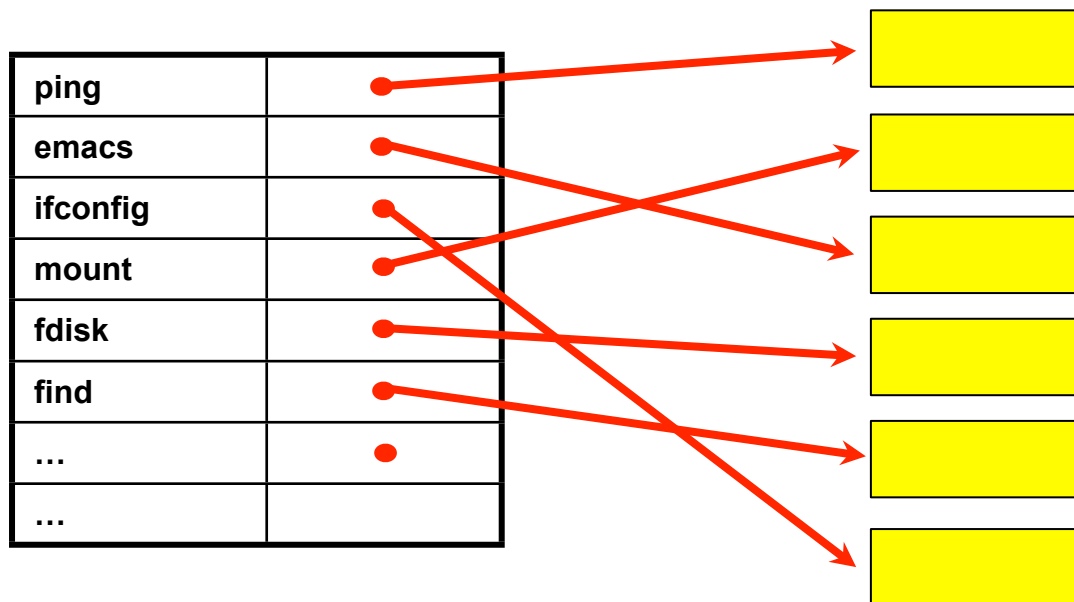
sequential access	implementation for direct access
<i>reset</i>	<i>cp = 0;</i>
<i>read next</i>	<i>read cp;</i> <i>cp = cp+1;</i>
<i>write next</i>	<i>write cp;</i> <i>cp = cp+1;</i>

Example of Index and Relative Files



Directory Structure

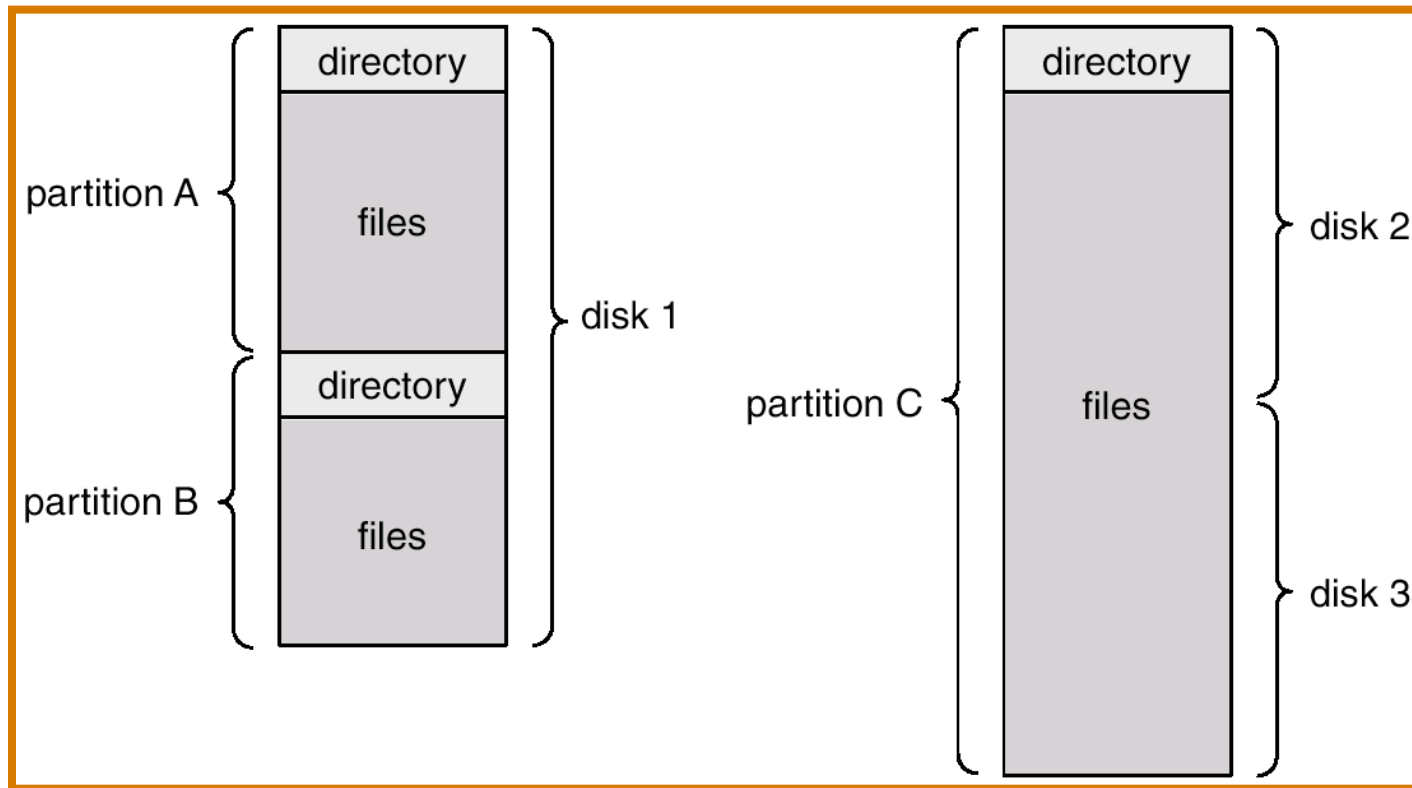
Directory: a symbol table that translates file names into directory entries.



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

Partitions and Directories

(File system organization)



Operations on Directories

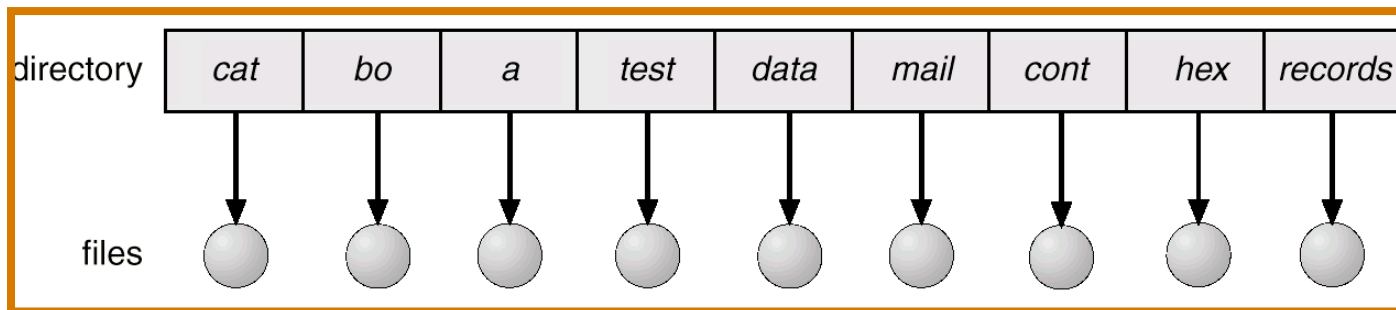
- Search for a file.
- Create a file.
- Delete a file.
- List a directory.
- Rename a file.
- Traverse the file system.

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

A single directory for all users.



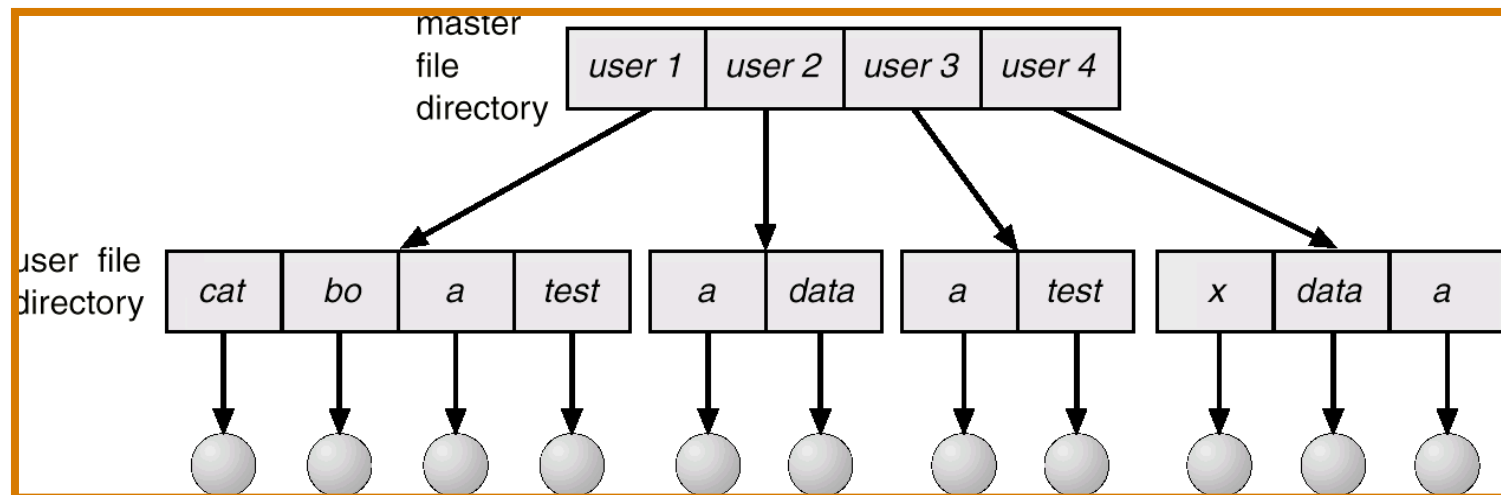
Drawbacks:

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 (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

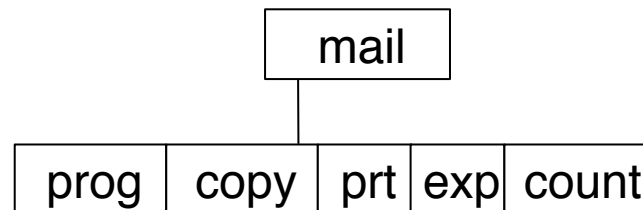
rm <file-name>

- Creating a new subdirectory is done in current directory.

mkdir <dir-name>

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

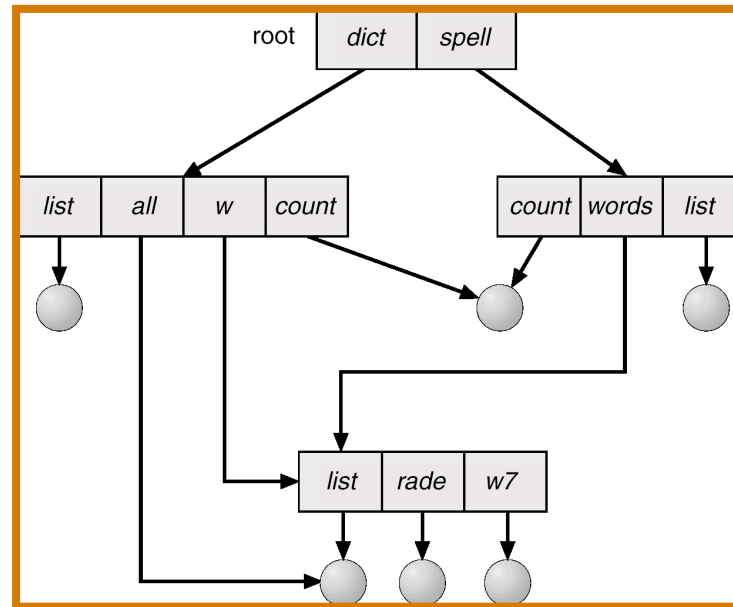
mkdir count



rm -rf . ⇒ doesn't mean “read mail really fast”

Acyclic-Graph Directories

Have shared subdirectories and files.



links: { soft (symbolic)
hard

Unix: In (read man page);
need to keep a reference count on
each file or directory.

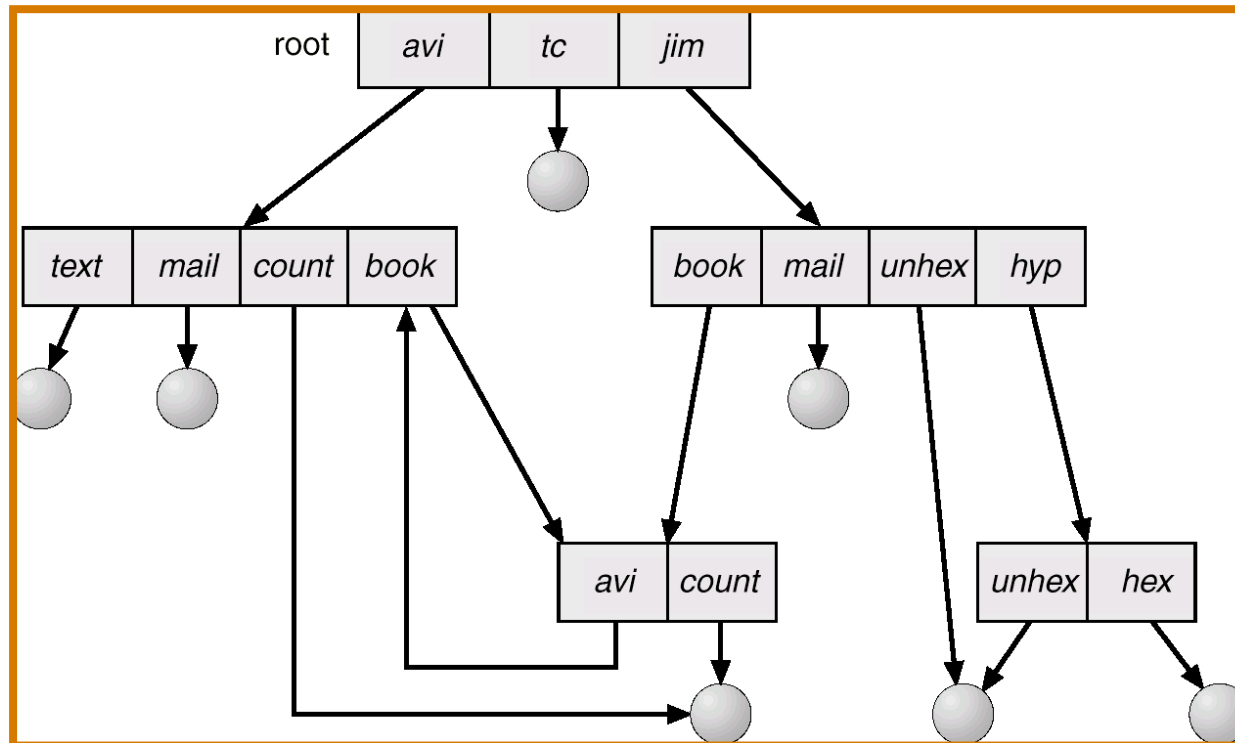
Acyclic-Graph Directories (Cont.)

- Different names (aliasing) for the same file or directory.
- If *dict* deletes *list* \Rightarrow 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

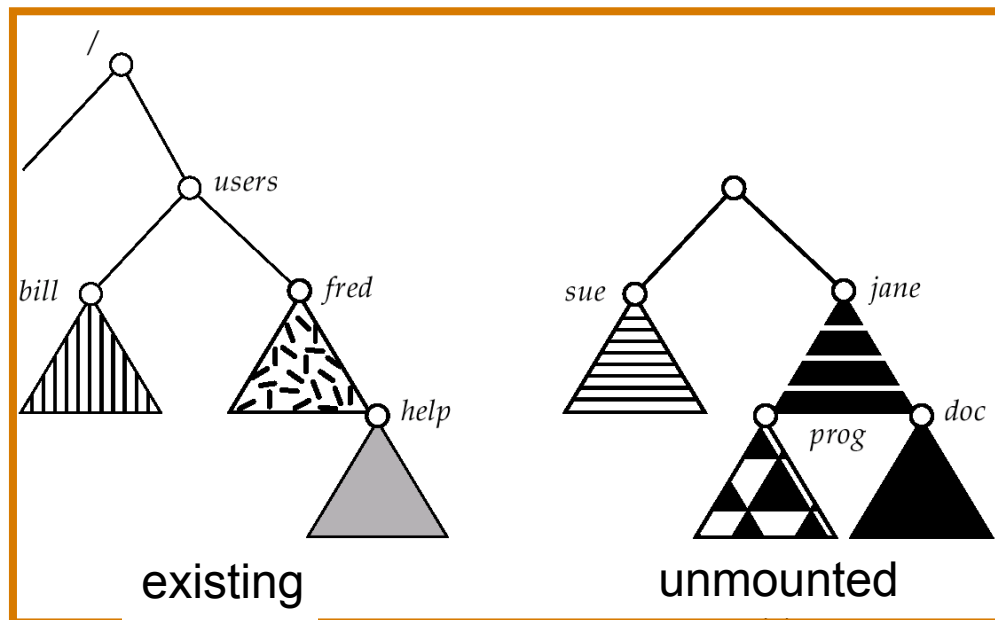


General Graph Directory (Cont.)

- **How do we guarantee no cycles?**
 - Allow only links to file not subdirectories.
 - Garbage collection.
 - Every time a new link is added use a cycle detection algorithm to determine whether it is OK.

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.



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.

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.

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.
- **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.

Access Lists and Groups

- Mode of access: **read, write, execute**
- Three classes of users
 - a) **owner access** RWX
7 \Rightarrow 1 1 1
 - b) **group access** RWX
6 \Rightarrow 1 1 0
 - c) **public access** RWX
1 \Rightarrow 0 0 1
- Ask manager to create a group (unique name), say G, and add some users to the group.
- For a particular file (say *game*) or subdirectory, define an appropriate access.

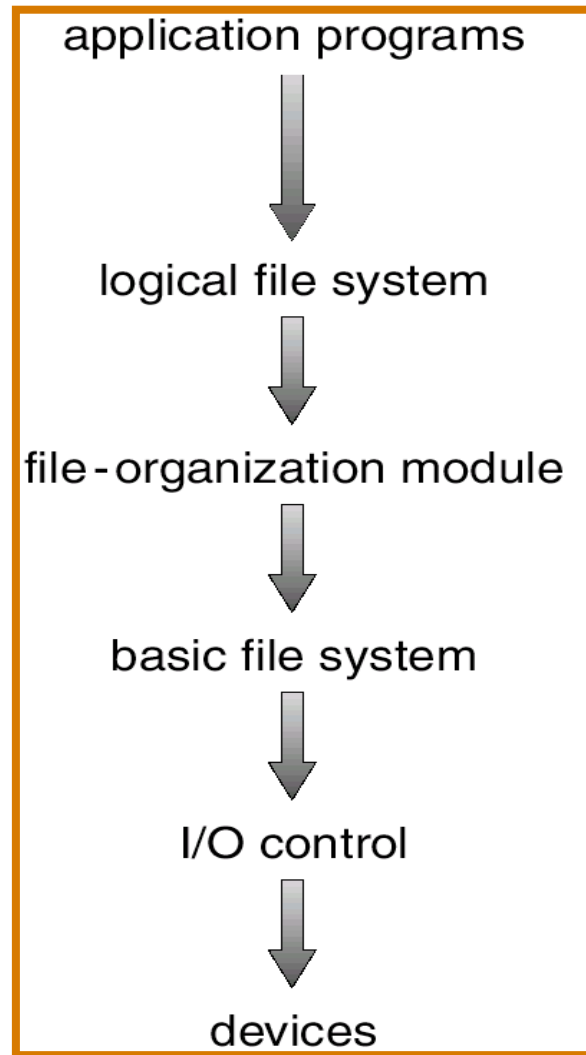
owner group public
 | | |
chmod 761 game

Associate a group with a file: **chgrp G game**

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.

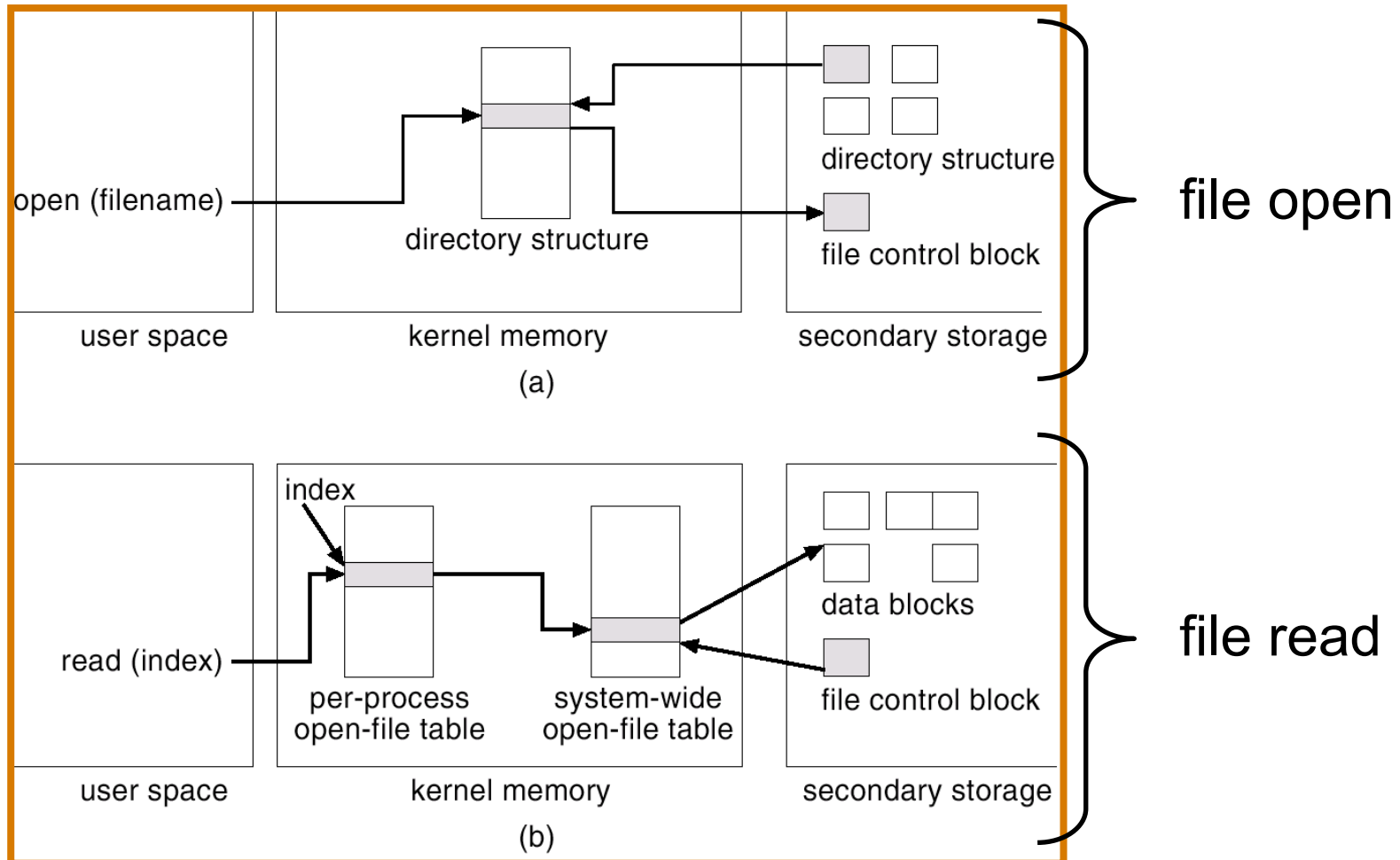
Layered File System



File Control Block

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

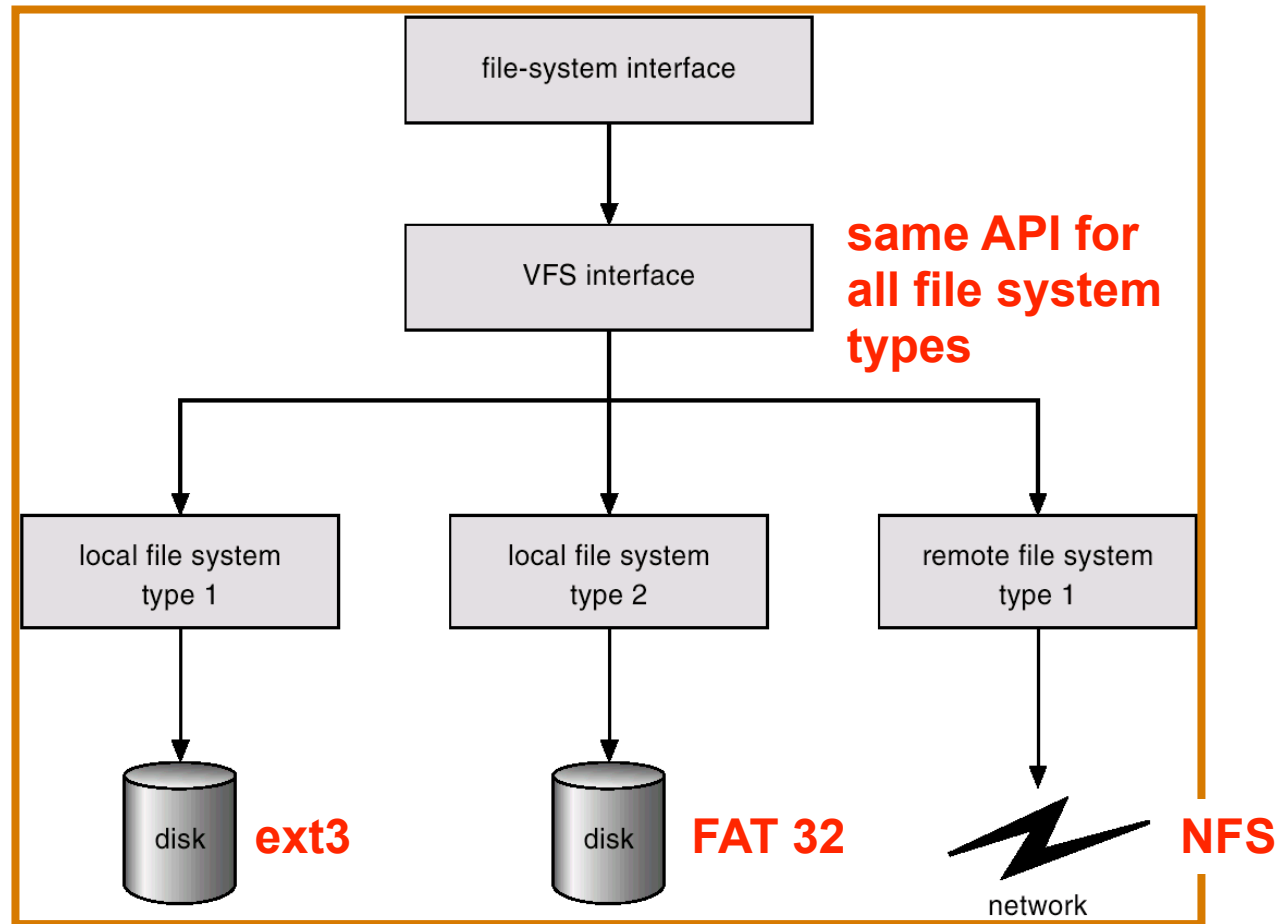
In-Memory File System Structures



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.

Schematic View of Virtual File System



Directory Implementation

The directory is a **symbol table** 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.

Allocation Methods

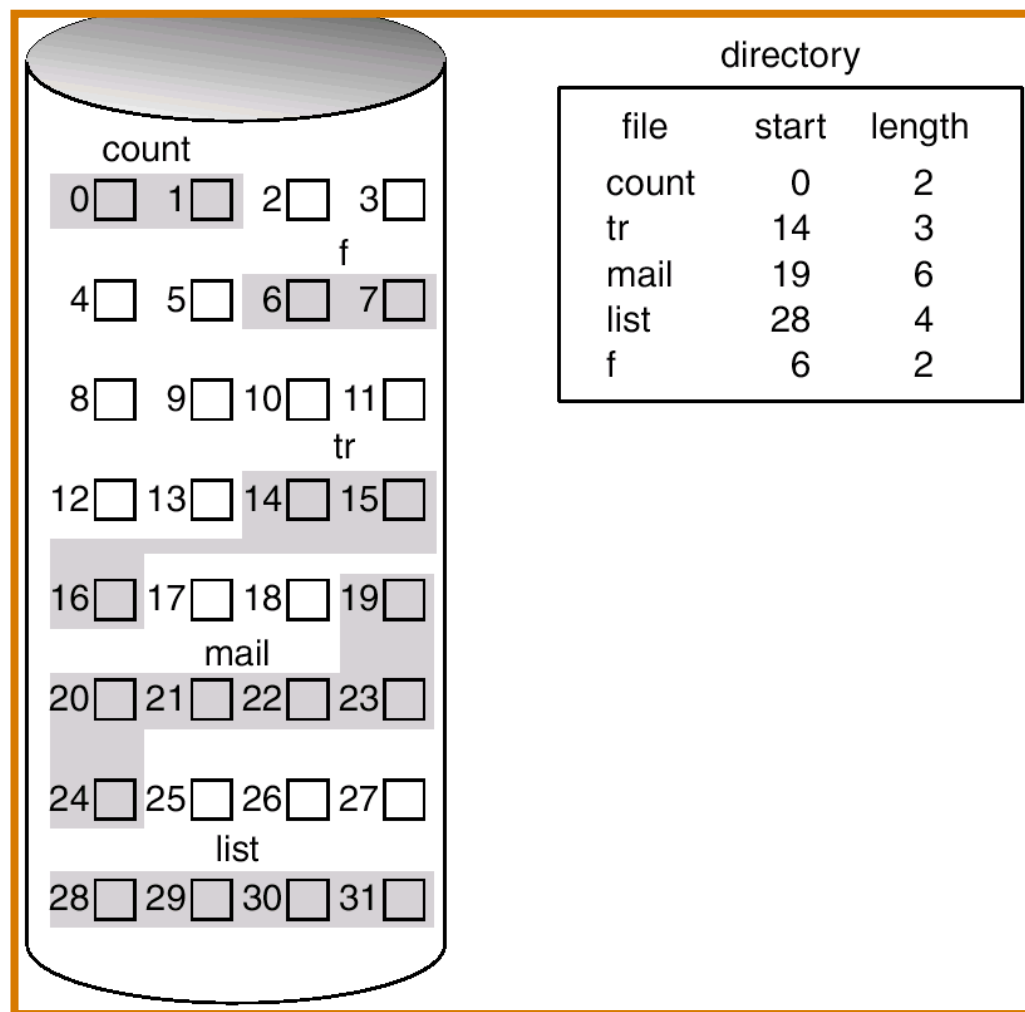
An **allocation method** refers to how disk blocks are allocated for files. We'll discuss three options:

- 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; external fragmentation.
- 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



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 => this leads to **internal 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.

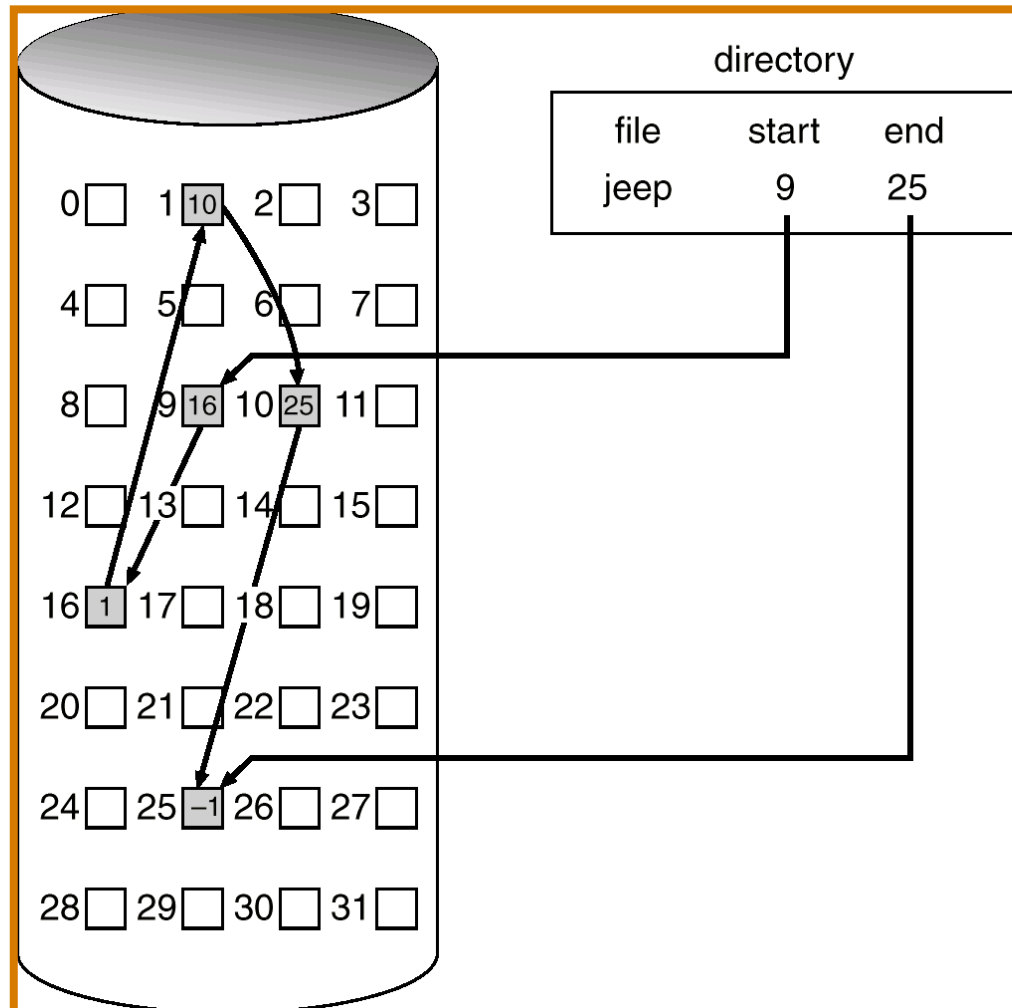
Simple: need only starting address.

Overhead: each block links to the next.

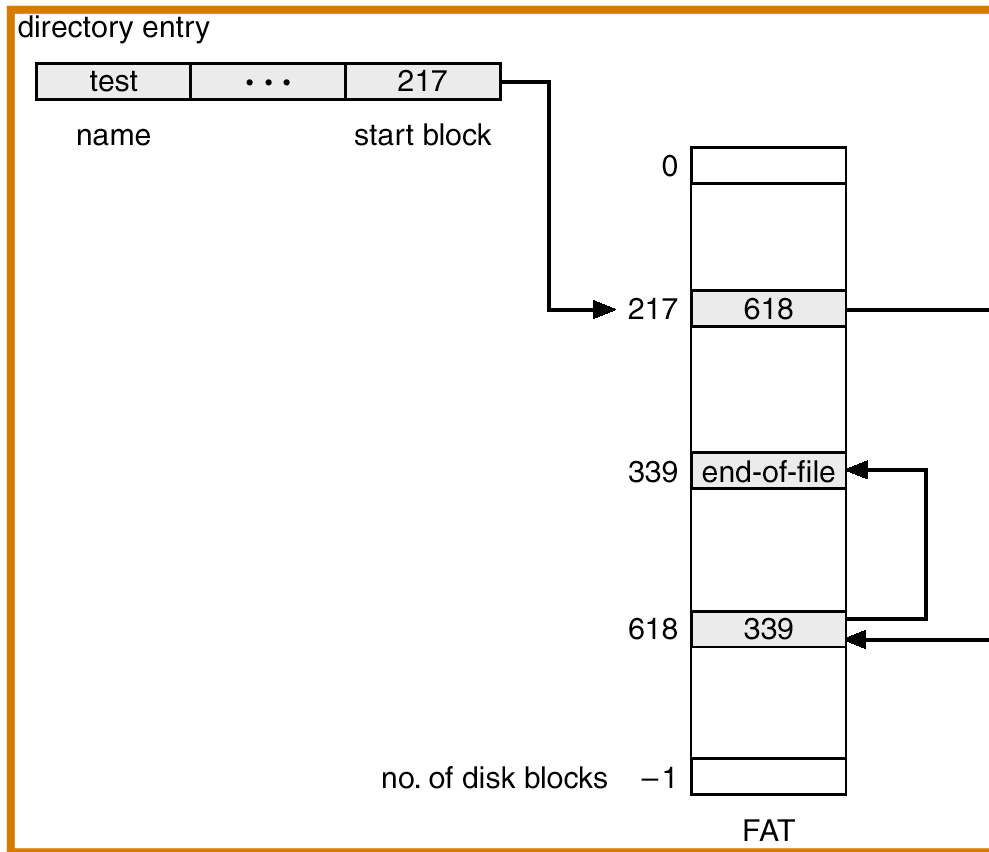
Space cost to store pointer.

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)



Simple and efficient: One entry for each block; indexed by block number. The table 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.

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*.

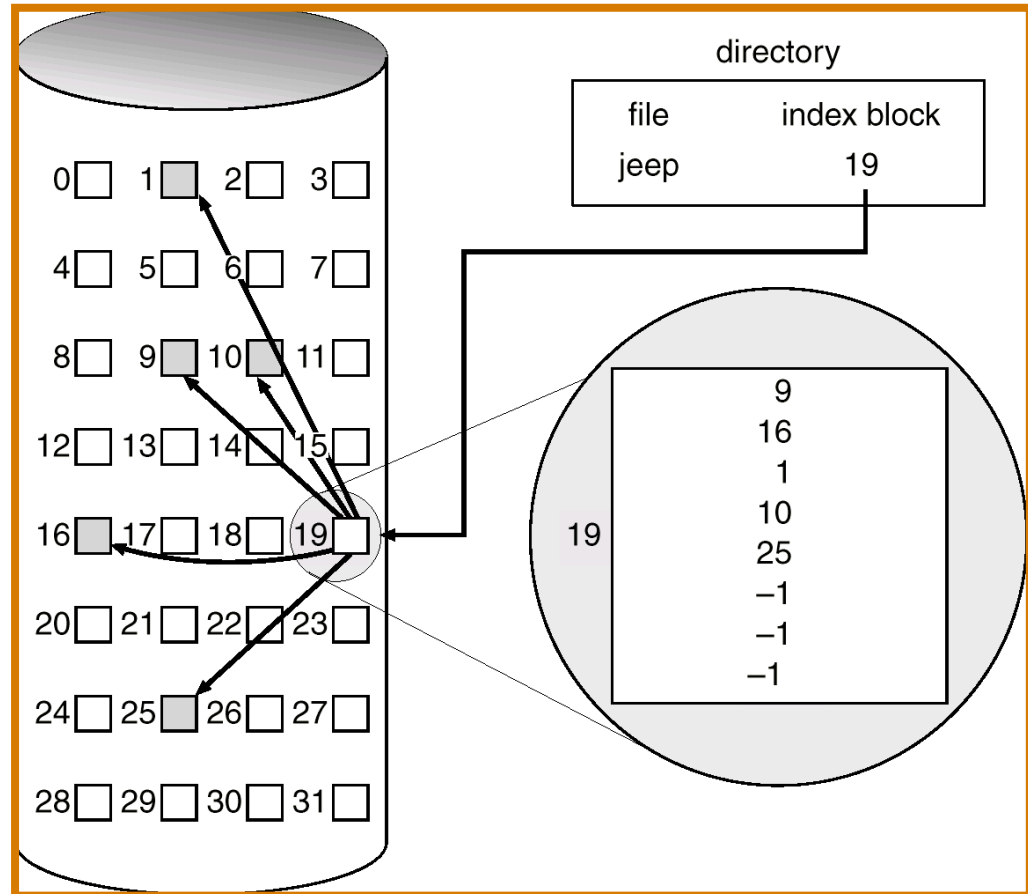
One index block *per file*.

Random access comes easy.

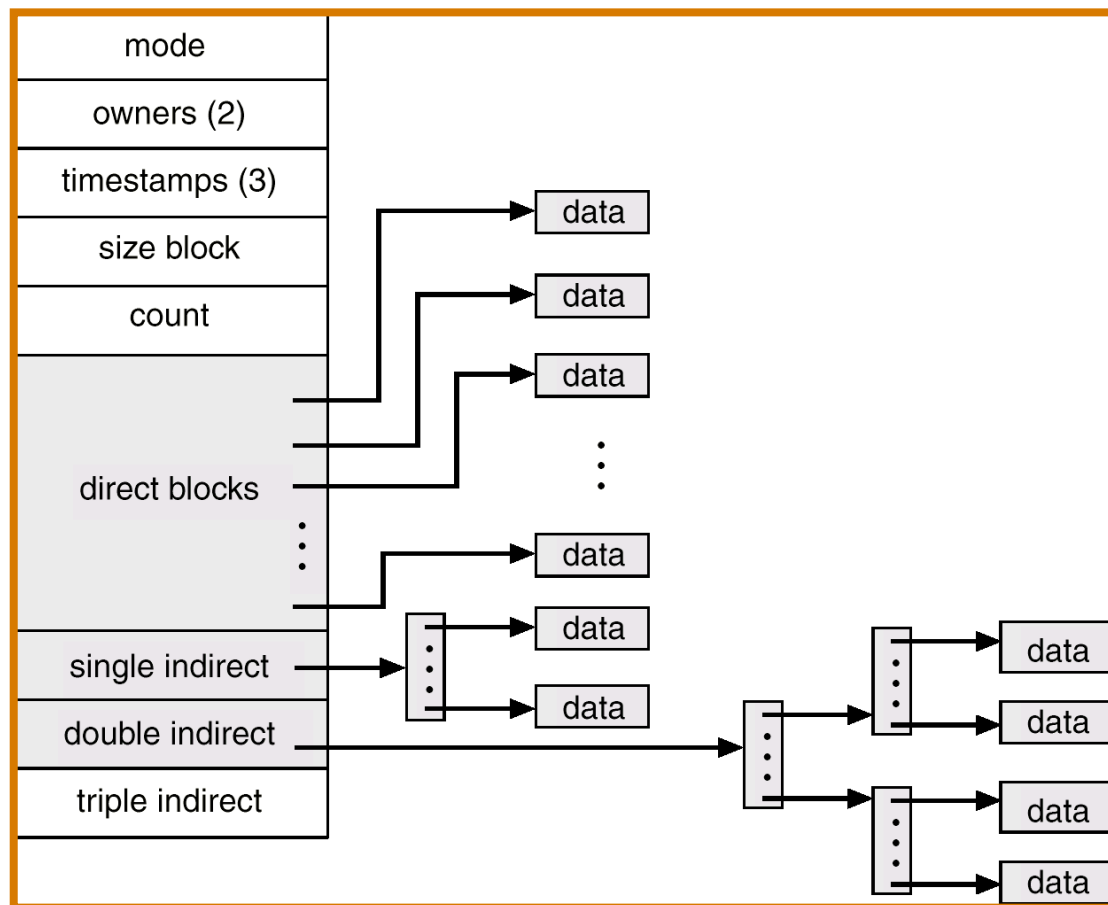
Dynamic access without external fragmentation, but have overhead of index block.

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

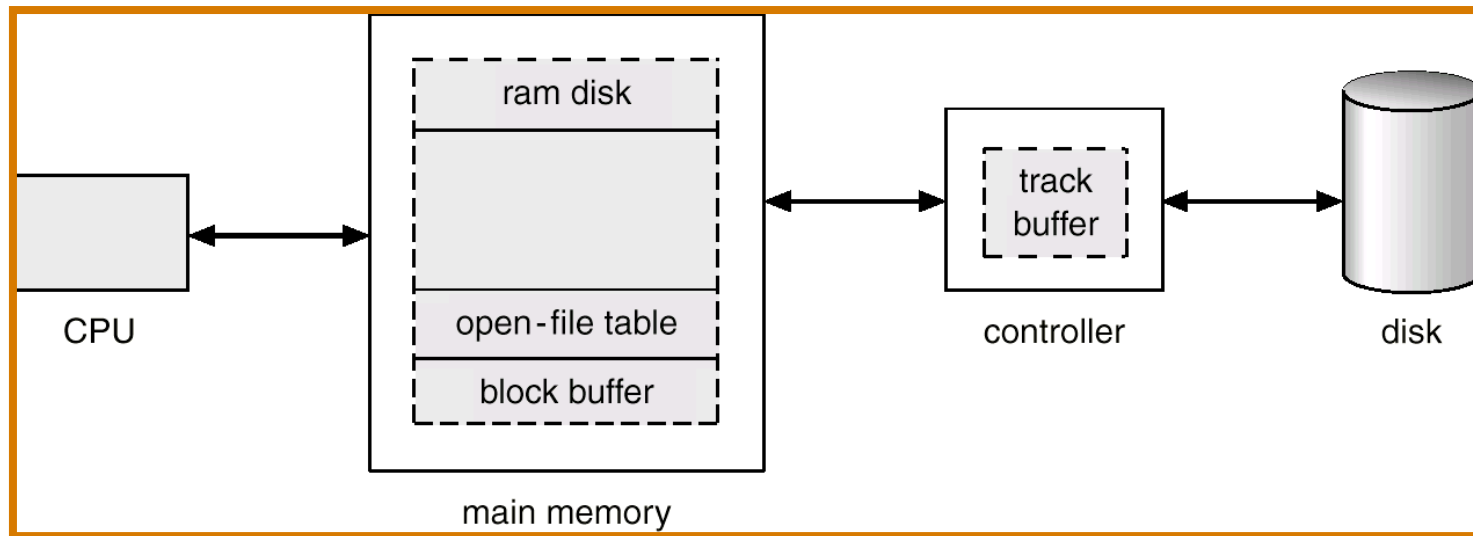
- **Bit map** (1 bit per disk block)
 - internal fragmentation
- **Linked list** (free list)
 - external fragmentation
- **Grouping**
 - first free block has address of n free blocks (the last of which has the address of the next n free blocks and so on)
- **Counting**
 - like linked list, but each node 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.

Efficiency and Performance

- **Efficiency dependent on:**
 - disk allocation and directory algorithms
 - types of data kept in file's directory entry
- **Performance**
 - disk cache – separate section of main memory for frequently used blocks
 - free-behind and read-ahead – techniques to optimize sequential access
 - improve PC performance by dedicating section of memory as virtual disk, or RAM disk.

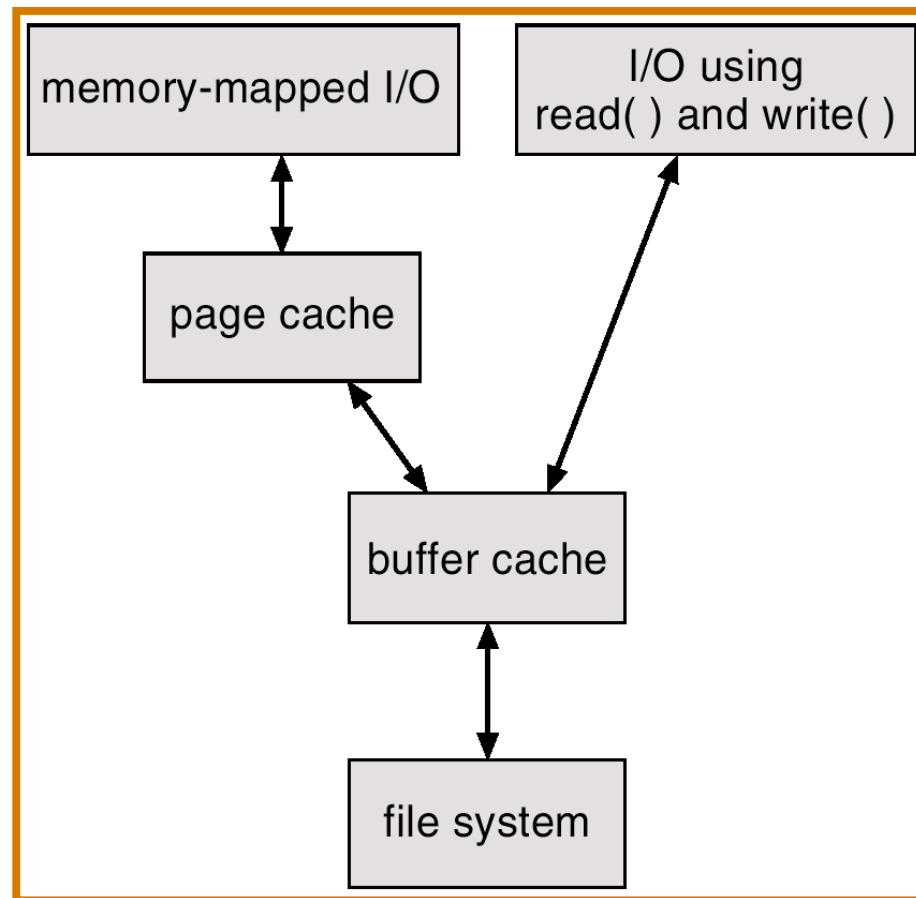
Various Disk-Caching Locations



Page Cache

- A **page cache** caches pages rather than disk blocks using virtual memory techniques.
- Memory-mapped I/O uses a page cache.
- Routine I/O through the file system uses the buffer (disk) cache.
- This leads to the following figure.

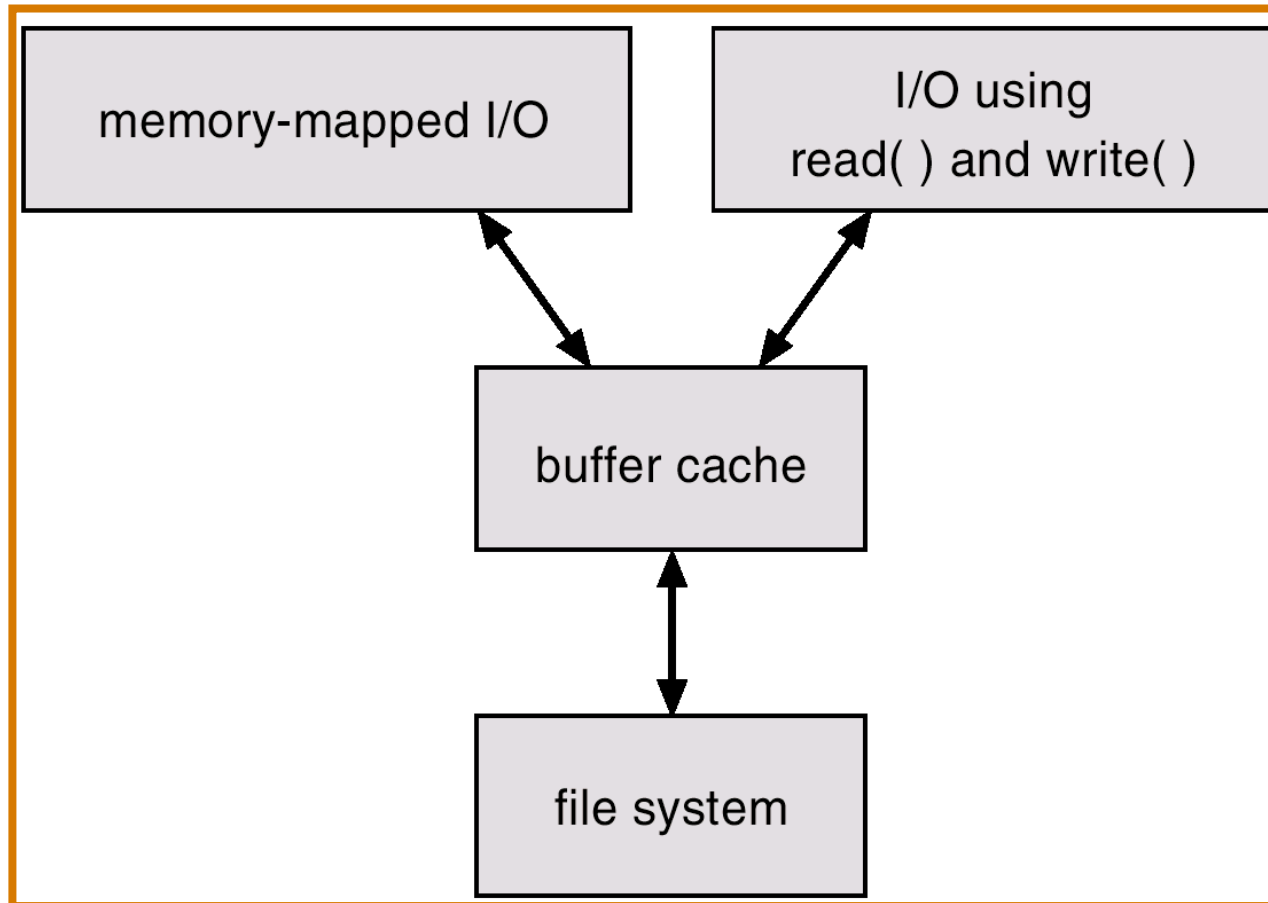
I/O Without a Unified Buffer Cache



Unified Buffer Cache

- A unified buffer cache uses the same page cache to cache both memory-mapped pages and ordinary file system I/O.

I/O Using a Unified Buffer Cache



Recovery

- **Consistency checking** – compares data in directory structure with data blocks on disk, and tries to fix inconsistencies.
- 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. However, 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.
- If the file system crashes, all remaining transactions in the log must still be performed.