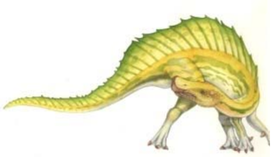
File-System Basics

- File Concept
- Access Methods
- Disk and Directory Structure



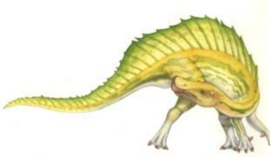
Objectives

- To explain the function of file systems
- To describe the interfaces to file systems
- To discuss file-system design tradeoffs, including access methods, file sharing, file locking, and directory structures

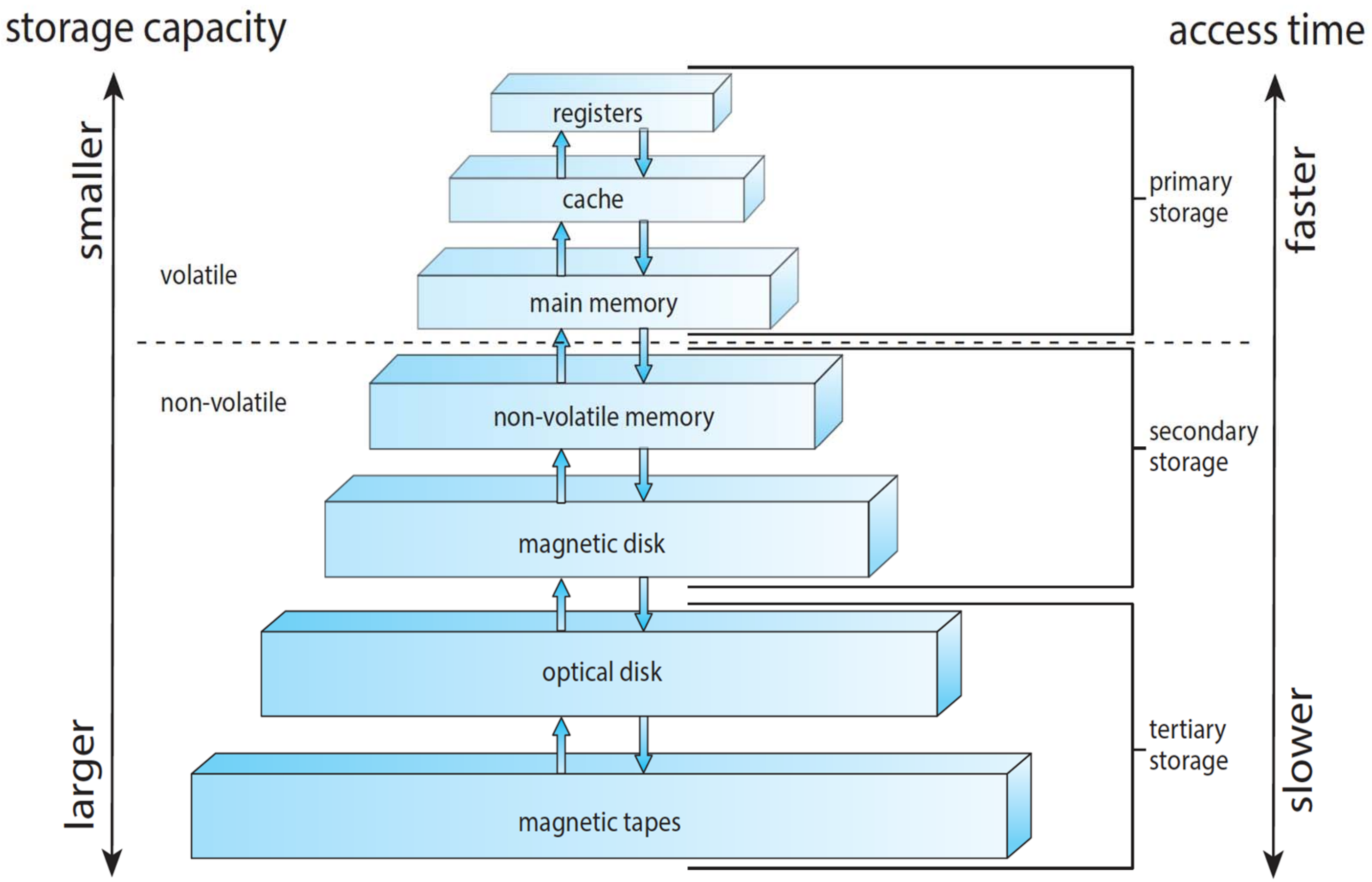
Files: OS Abstraction

- Files: another OS-provided abstraction over hardware resources

OS Abstraction	Hardware Resource
Processes Threads	CPU
Address space	Memory
Files	Disk



Storage-Device Hierarchy

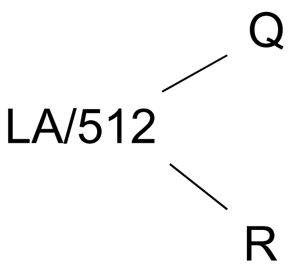


File Concept

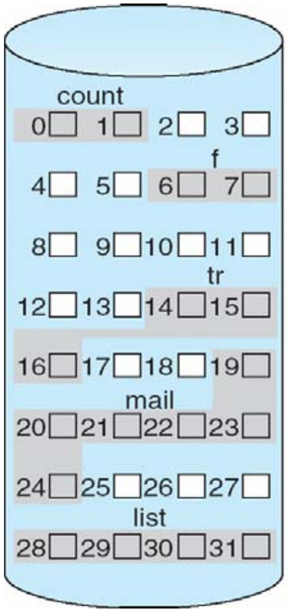
- The file system consists of two distinct parts:
  - A collection of files, each storing related data
  - A directory structure, which organizes and provides information about all the files in the system.
- File: Contiguous logical address space, mapped by the OS onto physical devices.
- Types:
  - Data
    - Numeric, character, binary
  - Program
- Contents (many types) is defined by file's creator
  - text file,
  - source file,
  - executable file

Contiguous Allocation

- Mapping from logical to physical (assume block size if 512)



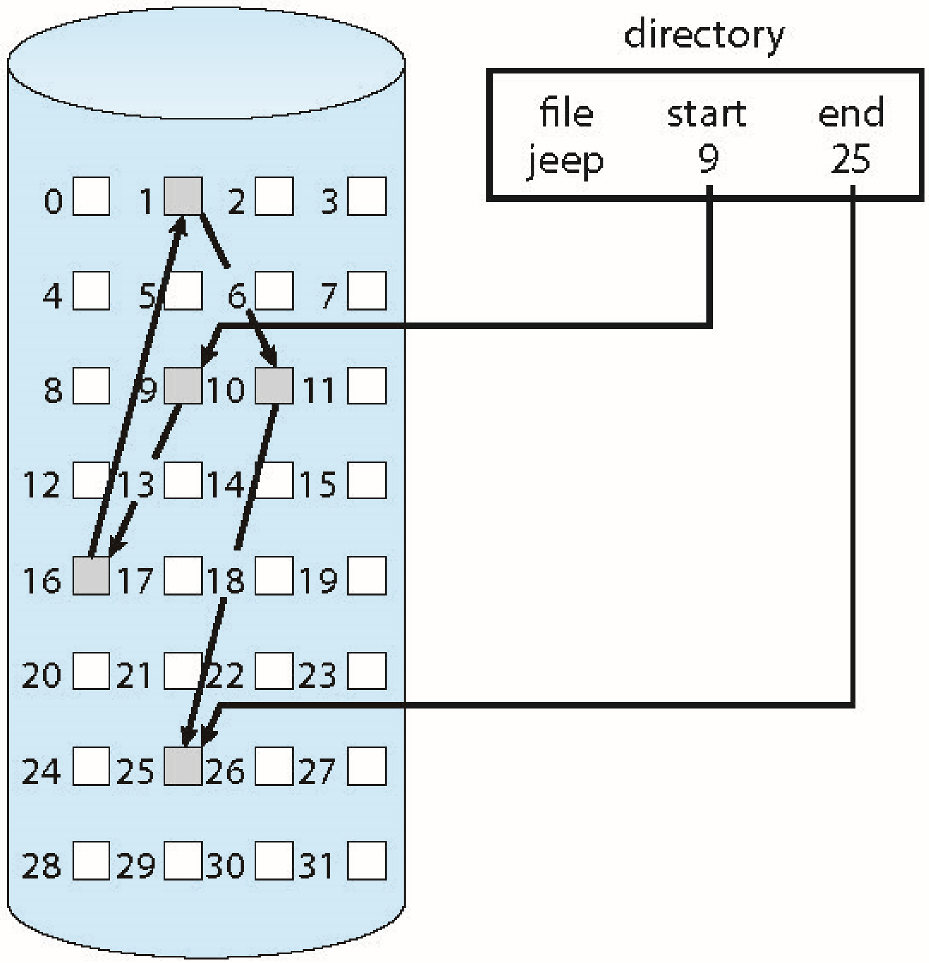
- Block to be accessed = “starting address” + Q
- Displacement into block = R



directory

file	start	length
count	0	2
tr	14	3
mail	19	6
list	28	4
f	6	2

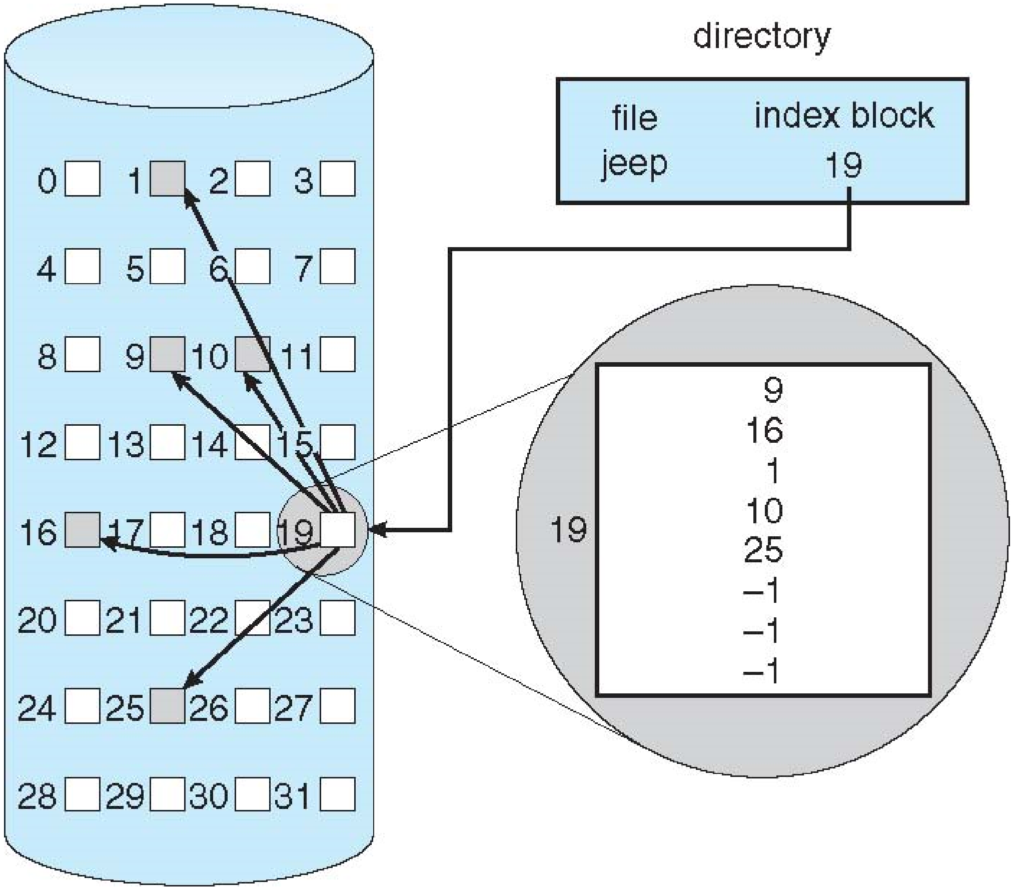
Linked Allocation



directory

file	start	end
jeep	9	25

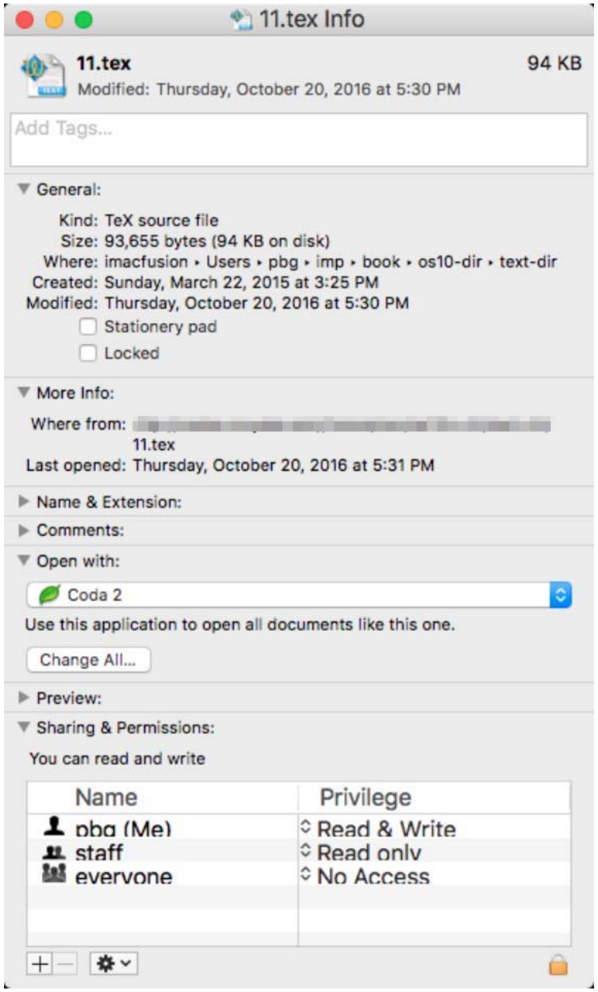
Indexed Allocation



File Attributes

- **Name** – only information kept in human-readable form
- **Identifier** – unique tag (number) identifies file within file system
- **Type** – needed for systems that support different types
- **Location** – pointer to file location on the device (disk)
- **Size** – current file size
- **Protection** – controls who can do reading, writing, executing
- **Time, date, and user identification** – information kept for creation time, last modification time, and last use time.
  - Useful for data for protection, security, and usage monitoring
- Information kept in the directory structure (on disk), which consists of “inode” entries for each of the files in the system.

File info Window on Mac OS X



11.tex Info

11.tex 94 KB
Modified: Thursday, October 20, 2016 at 5:30 PM

Add Tags...

▼ General:

Kind: TeX source file
Size: 93,655 bytes (94 KB on disk)
Where: imacfusion ▸ Users ▸ pbg ▸ imp ▸ book ▸ os10-dir ▸ text-dir
Created: Sunday, March 22, 2015 at 3:25 PM
Modified: Thursday, October 20, 2016 at 5:30 PM

□ Stationery pad
□ Locked

▼ More Info:

Where from: [redacted]
11.tex
Last opened: Thursday, October 20, 2016 at 5:31 PM

▶ Name & Extension:

▶ Comments:

▼ Open with:

Coda 2

Use this application to open all documents like this one.

Change All...

▶ Preview:

▼ Sharing & Permissions:

You can read and write

Name	Privilege
pbg (Me)	Read & Write
staff	Read only
everyone	No Access

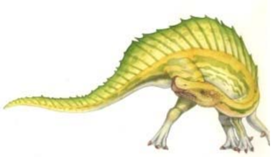
File Operations

- **Create**
- **Write –** at **write pointer** location
- **Read –** at **read pointer** location
- **Reposition within file - seek**
- **Delete**
- **Truncate**
- ***Open(F_i)*** – search the directory structure on disk for inode entry ***F_i***, and move the content of the entry to memory
- ***Close (F_i)*** – move the content of inode 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**: Keeps information (inode data) about all the open files
- **File pointer** : A pointer to last read/write location, per process that has the file open
- **File-open count**: A counter of the 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**: Many file operations require the system to modify data within the file. The information needed to locate the file on disk is kept in memory so that the system does not have to read it from disk for each operation.
- **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 according to the lock state.
- Access policy:
  - **Mandatory** – access is denied depending on locks held and requested; OS ensures locking integrity.
  - **Advisory** – processes can find status of locks and decide what to do

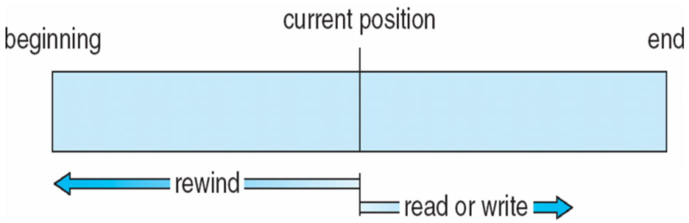
File Structure

- None - sequence of words, bytes
- Simple record structure
  - Lines
  - Fixed length
  - Variable length
- Complex Structures
  - Formatted document
  - Relocatable load file
- Who decides:
  - Operating system
  - Program

Access Methods

Sequential Access (based on tape model)

- Information in the file is processed in order, one record after the other.
- General structure

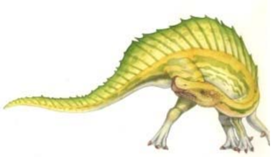


- Operations:
  - **read_next ()** – reads the next portion of the file and automatically advances a file pointer.
  - **write_next () –** append to the end of the file and advances to the end of the newly written material (the new end of file).
  - **reset** – back to the beginning of the file.

Access Methods

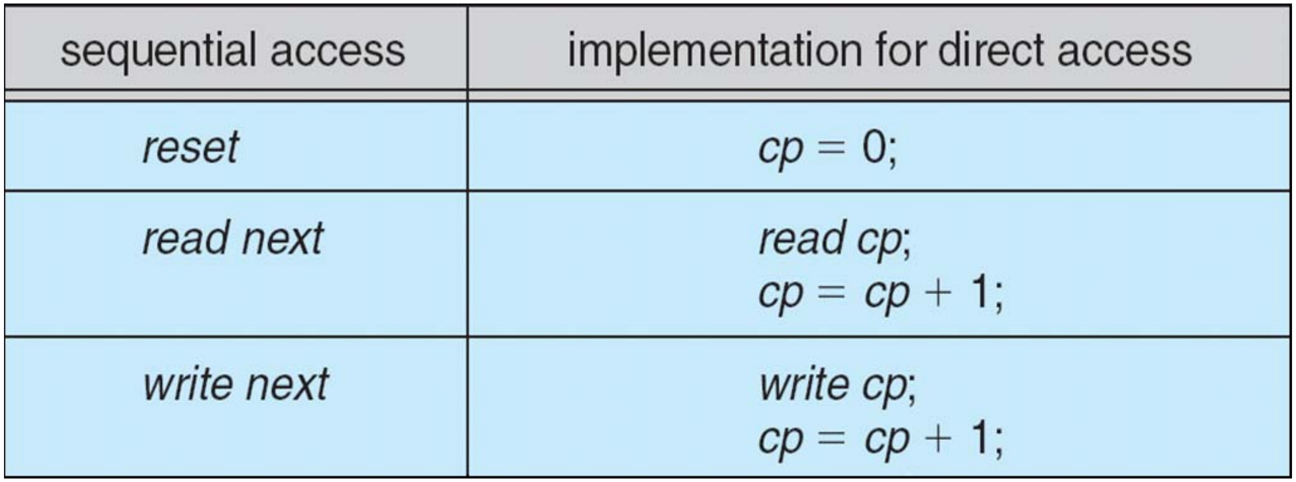
Direct Access (based on disk model)

- File is made up of fixed-length *logical records* that allow programs to read and write records rapidly in no particular order.
- File is viewed as a numbered sequence of blocks or records. For example, can read block 14, then read block 53, and then write block 7.
- Operations:
  - **read(n)** – reads relative block number n.
  - **write(n)** – writes relative block number n.
- Relative block numbers (to the beginning of the file) allow OS to decide where file should be placed



Simulation of Sequential Access on Direct-access File

cp is a pointer to the next block to be read/written

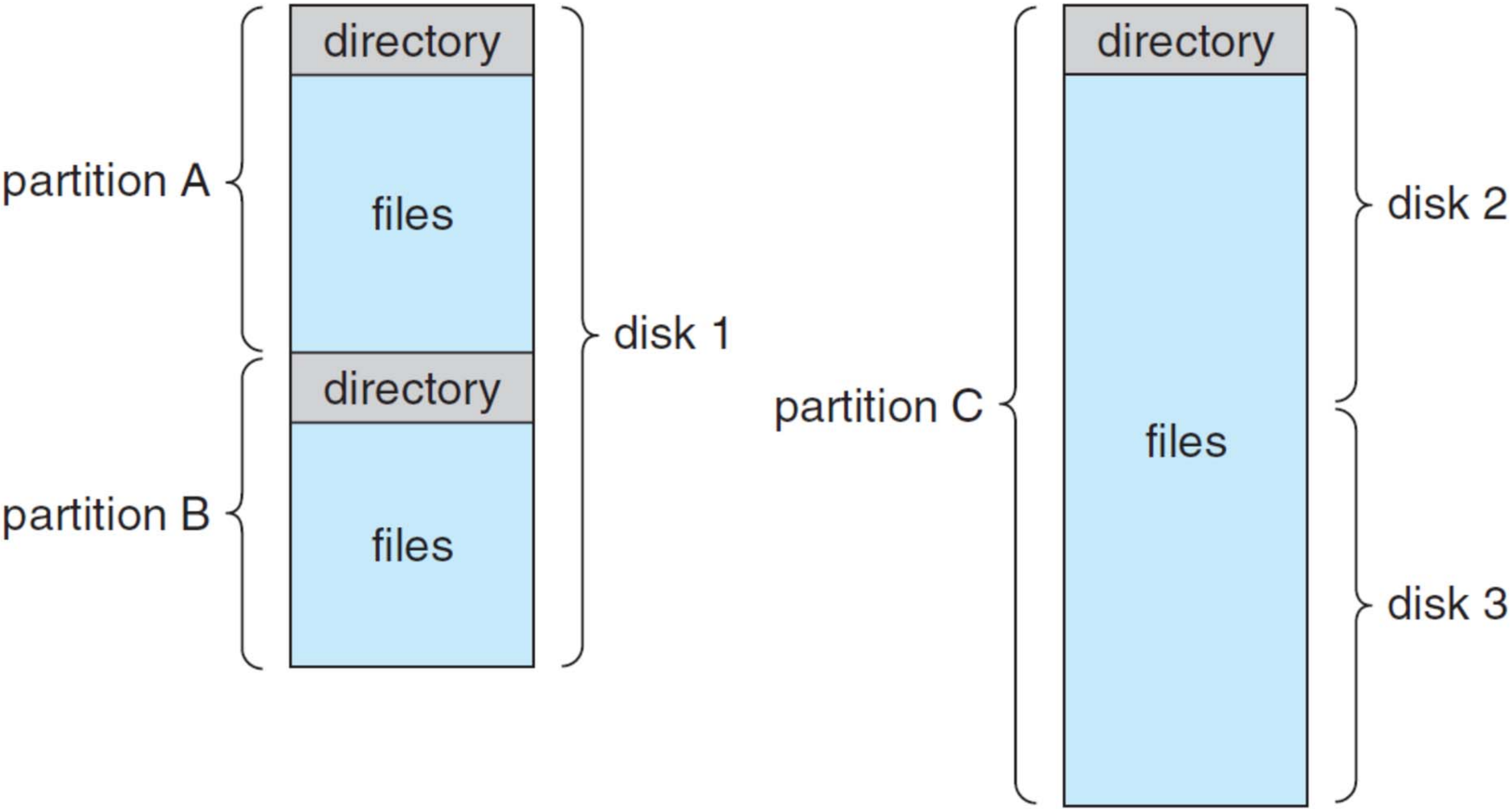


sequential access	implementation for direct access
reset	*cp* = 0;
read next	*read cp*; *cp* = *cp* + 1;
write next	*write cp*; *cp* = *cp* + 1;

Other Access Methods

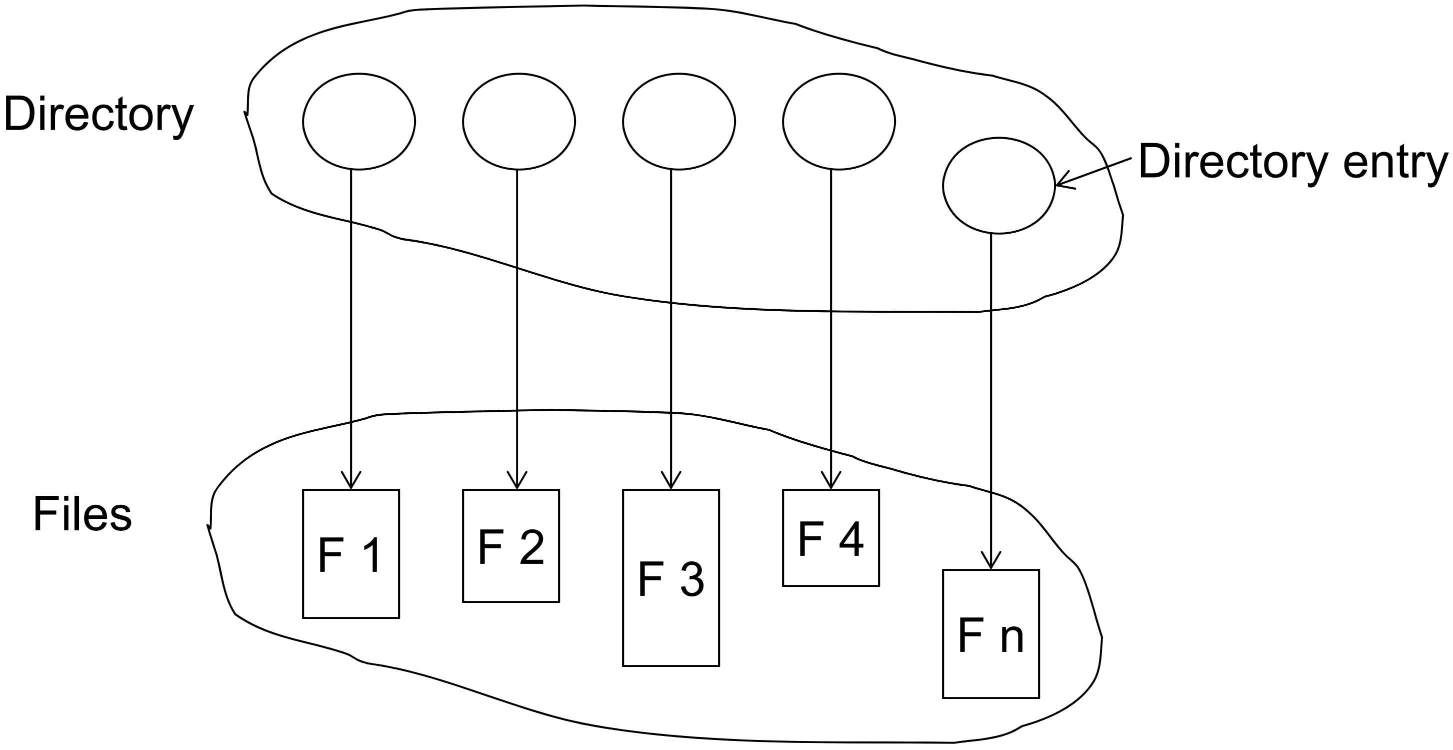
- Can be built on top of the direct-access methods
- Generally -- involve creation of an index for the file, containing pointers to the various blocks.
- Keep index in memory for fast determination of location of data to be operated on
- For example, a retail-price file might list the universal product codes (UPCs) for items, with the associated prices: 10-digit UPC + 6-digit price = a 16-byte record. If the disk has 1,024 bytes per block, we can store 64 records per block. A file of 120,000 records would occupy about 2,000 blocks (2 million bytes). By keeping the file sorted by UPC, we can define an index consisting of the first UPC in each block. This index would have 2,000 entries of 10 digits each, or 20,000 bytes, and thus could be kept in memory.
- If too large, keep index (in memory) of the main index (on disk)

A Typical File System Organization



Directory Structure

- The directory can be viewed as a symbol table that translates file names into their directory
- A collection of nodes containing information about all files



Both the directory structure and the files reside on disk

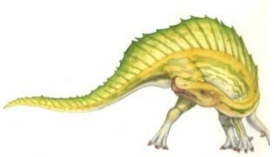
Operations Performed on Directory

- Search for a file
- Create a file
- Delete a file
- List a directory
- Rename a file
- Traverse the file system: access every directory and every file within a directory structure.

Directory Organization

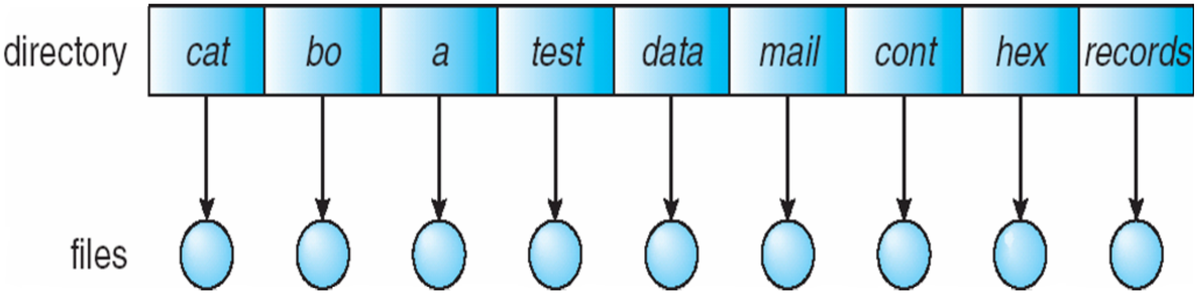
The directory is organized logically to obtain

- 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

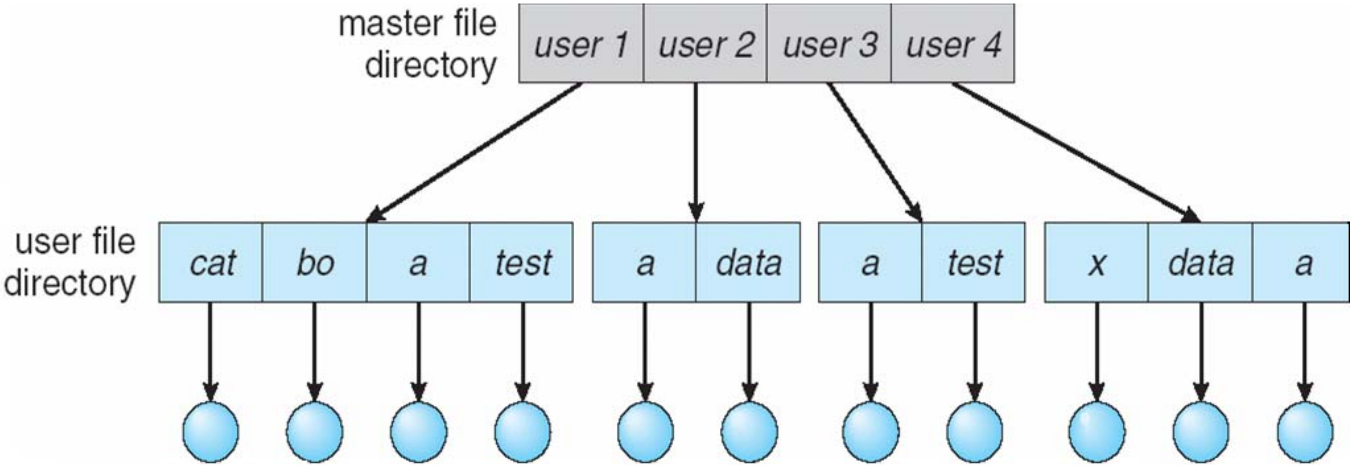
- A single directory for all users



- Naming problem: unique name rule is violated
- Grouping problem

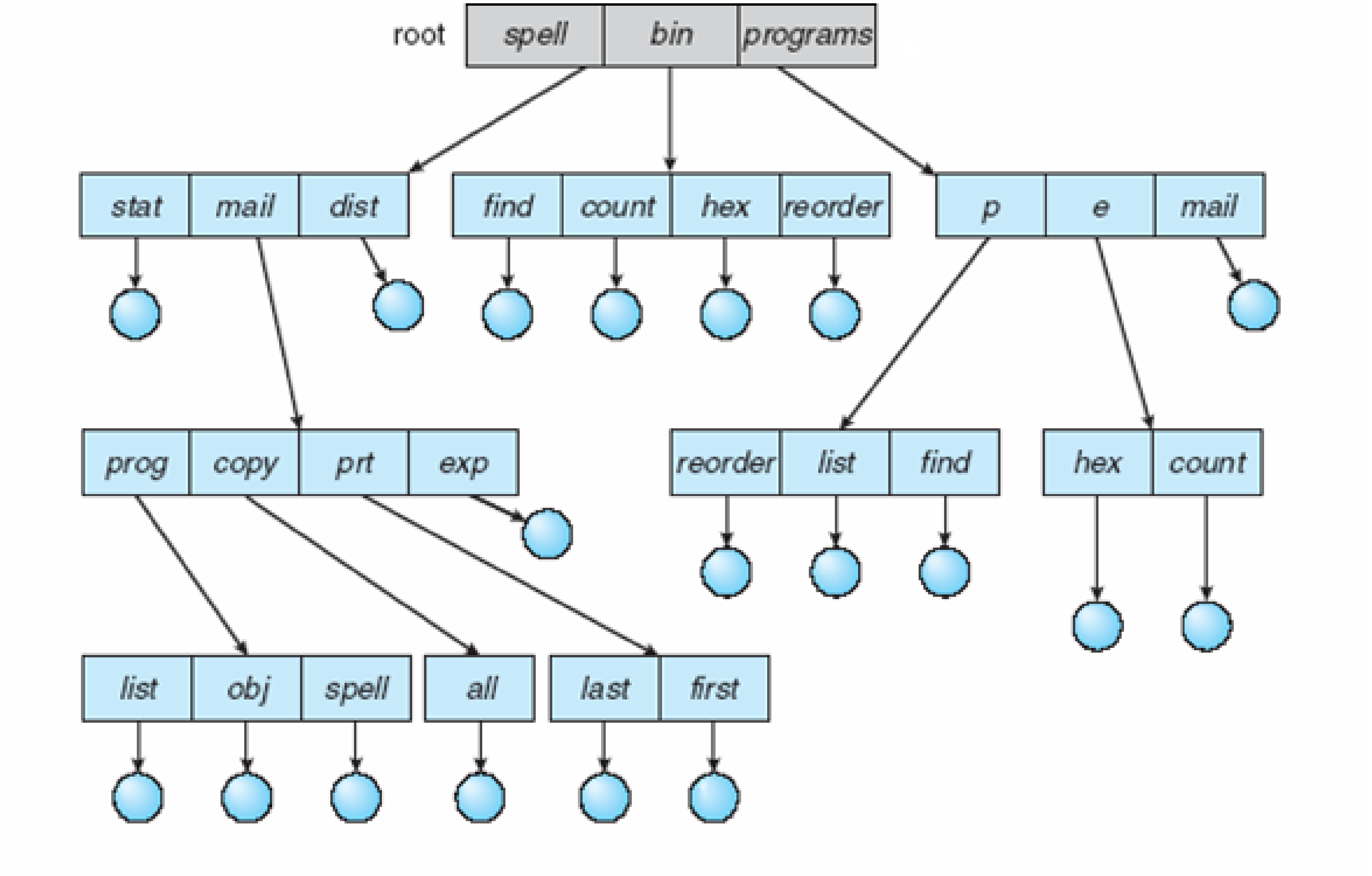
Two-Level Directory

- Separate directory for each user



- Can have the same file name for different user
- Efficient searching
- User isolation: difficult for file sharing
- No grouping capability

Tree-Structured Directories



Tree-Structured Directories (Cont.)

- Efficient searching
  - Users can create their own subdirectories and organize their files
  - Every file in the system has a unique path name
- Grouping Capability
- Current directory (working directory)
  - **`cd /spell/mail/prog`**
  - **`type list`**

Tree-Structured Directories (Cont)

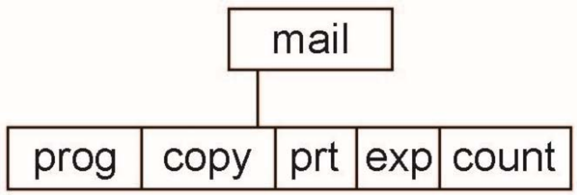
- **Absolute** or **relative** path name
- Creating a new file is done in current directory
- Delete a file

```
rm <file-name>
```

- Creating a new subdirectory is done in current directory

```
mkdir <dir-name>
```

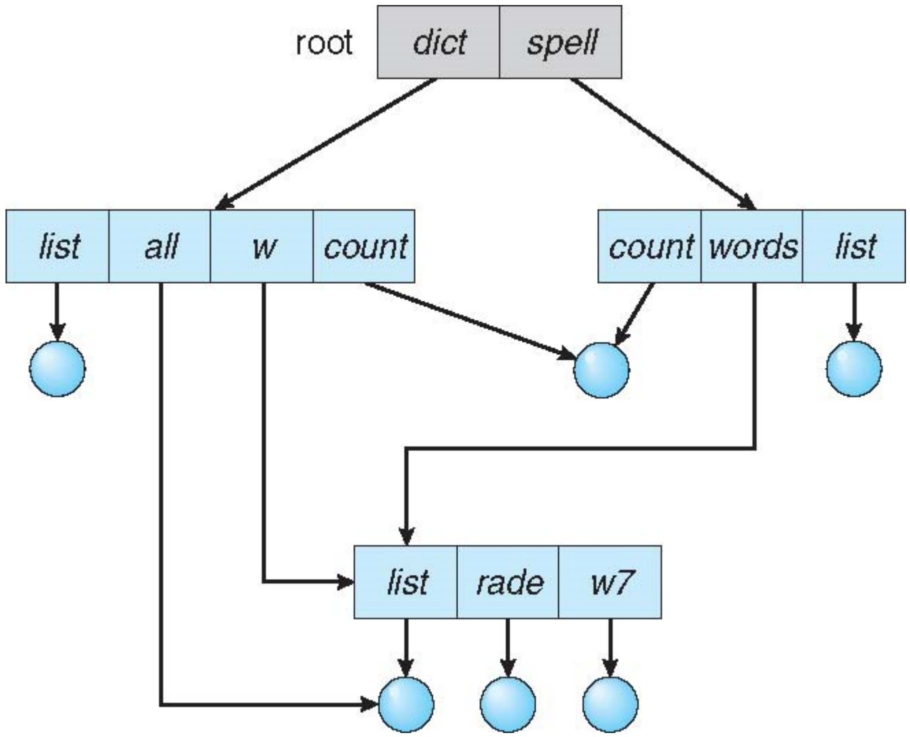
- Example: Suppose the current directory is -- “`/mail`“
  - `mkdir count`



- Deleting “mail” ⇒ deleting the entire subtree rooted by “mail”

Acyclic-Graph Directories

Have shared subdirectories and files



Acyclic-Graph Directories (Cont.)

- Files/subdirectories have two different names (aliasing)
  - Only one actual file exists, so any changes made by one person are immediately visible to the other.
- How do we implement shared files and subdirectories?
  - Can duplicate the *inode* information about the shared file/subdirectory in each of the subdirectories that “point’ to the shared structure.
    - If some information changes about the file it had to be updated in several places.
  - Have a new directory entry type:
    - **Link** – another name (pointer) to an existing file or subdirectory.

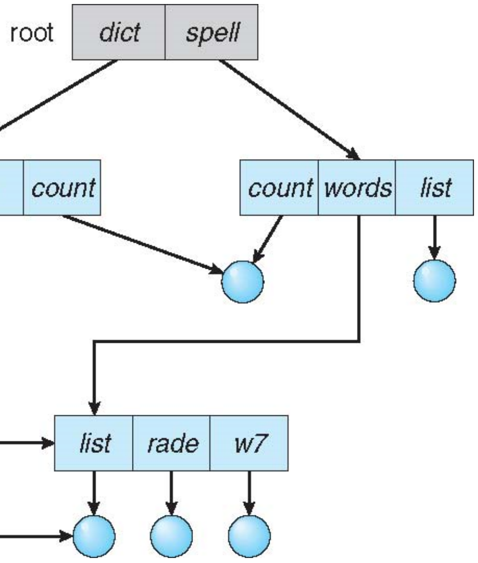
Acyclic-Graph Directories -- Links

- A link may be implemented as an **absolute** or a **relative** path name.
- When a reference to a file is made, the directory is searched. If the directory entry is marked as a link, then the name of the real file is included in the link information.
- We **resolve** the link by using that path name to locate the real file.
- Links are easily identified by their format in the directory entry (or by having a special type on systems that support types) and are effectively indirect pointers.
- The operating system ignores these links when traversing directory trees to preserve the acyclic structure of the system.

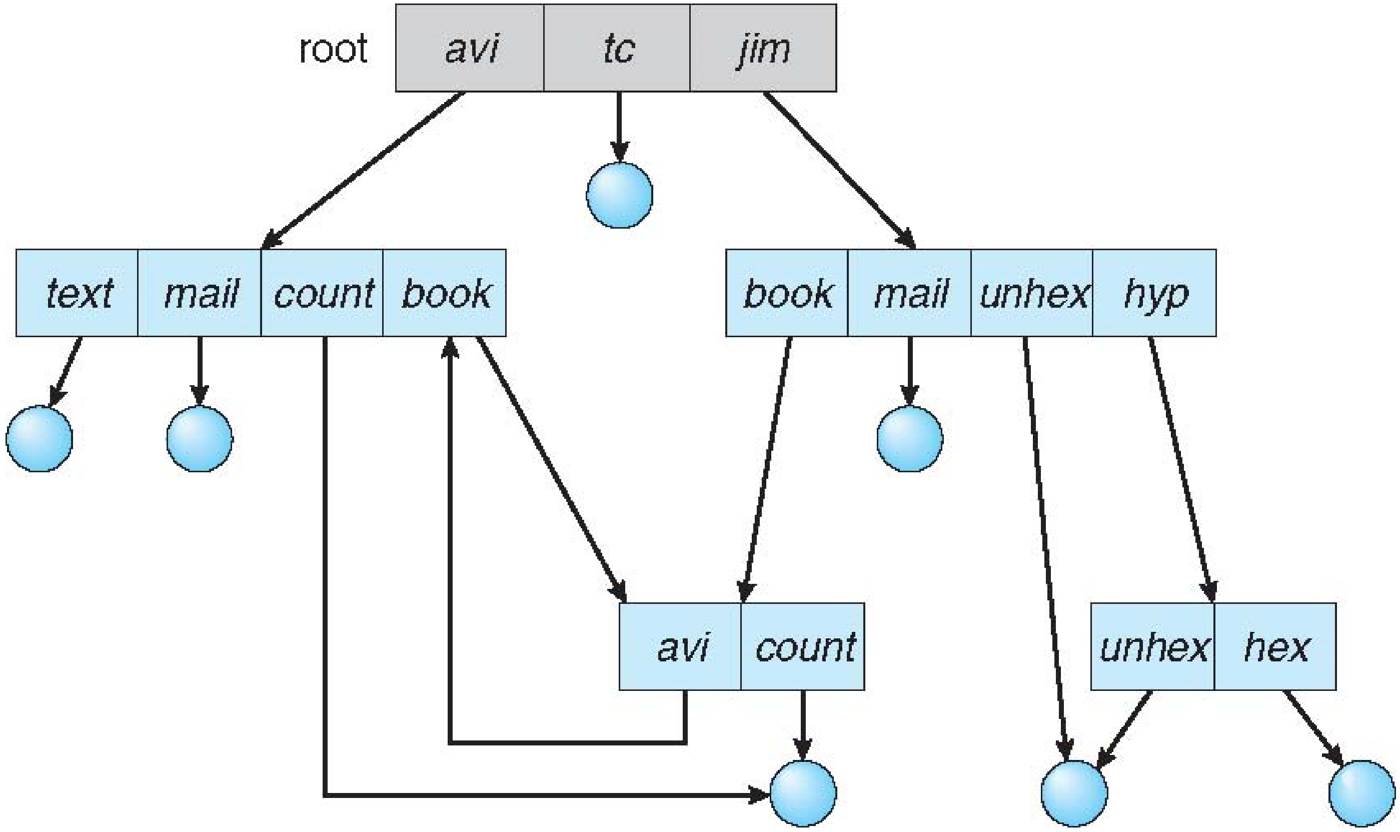
Acyclic-Graph Directories -- Deletion

- If ***dict /count*** is deleted ⇒ dangling pointer

  Solutions:
  - Backpointers -- so we can delete all pointers.
    - Keep a list of all references to the file
    - Variable and large size records a problem
  - Entry-hold-count solution



General Graph Directory

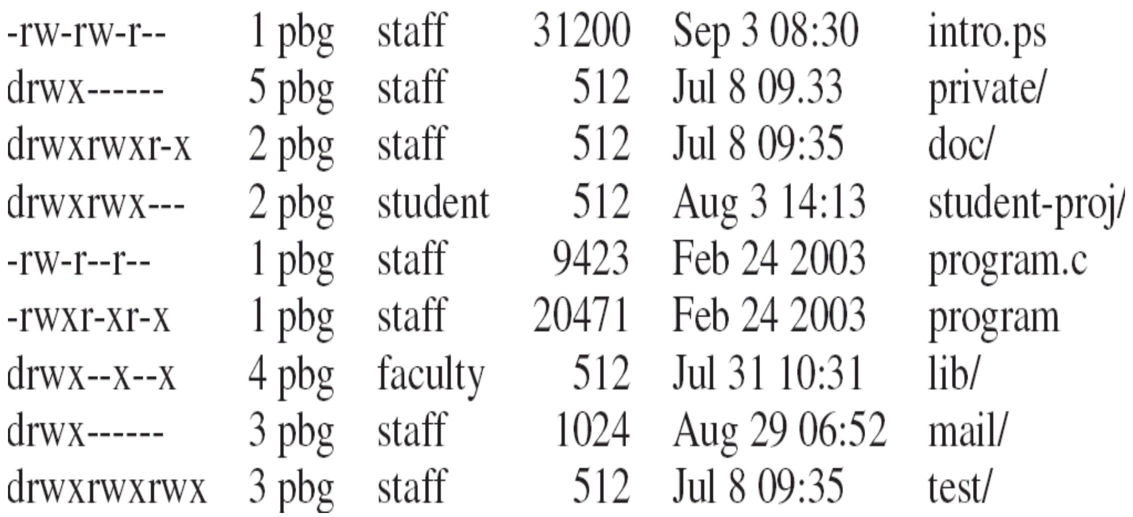


General Graph Directory (Cont.)

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

A Sample UNIX Directory Listing

- Example:



```
-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/
```

- Explanation:
  - “into.ps” is a file owned by “pbg” with group ”staff”
  - “lib” is a subdirectory owned by “pbg” with group “faculty”