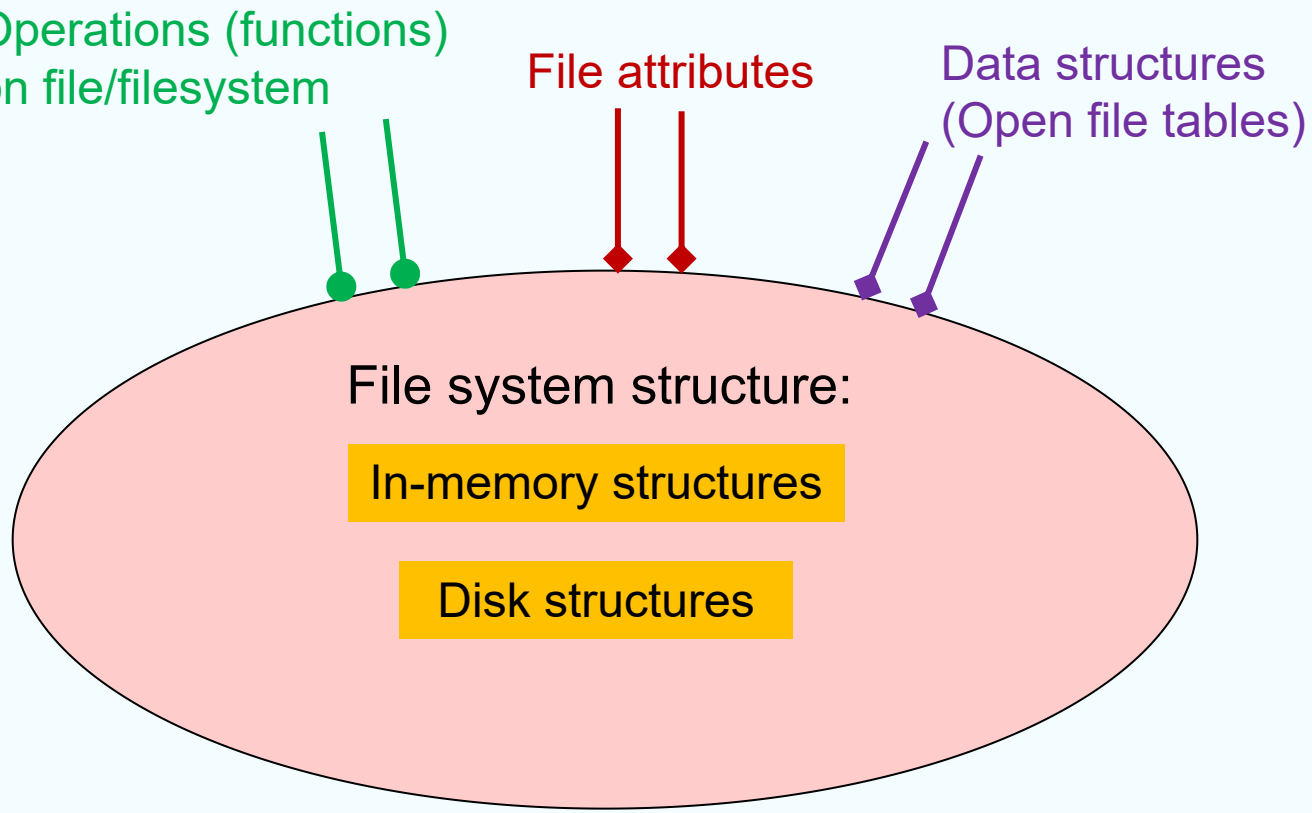
Chapter 10: File-System Interface

Chapter 10: File-System Interface

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

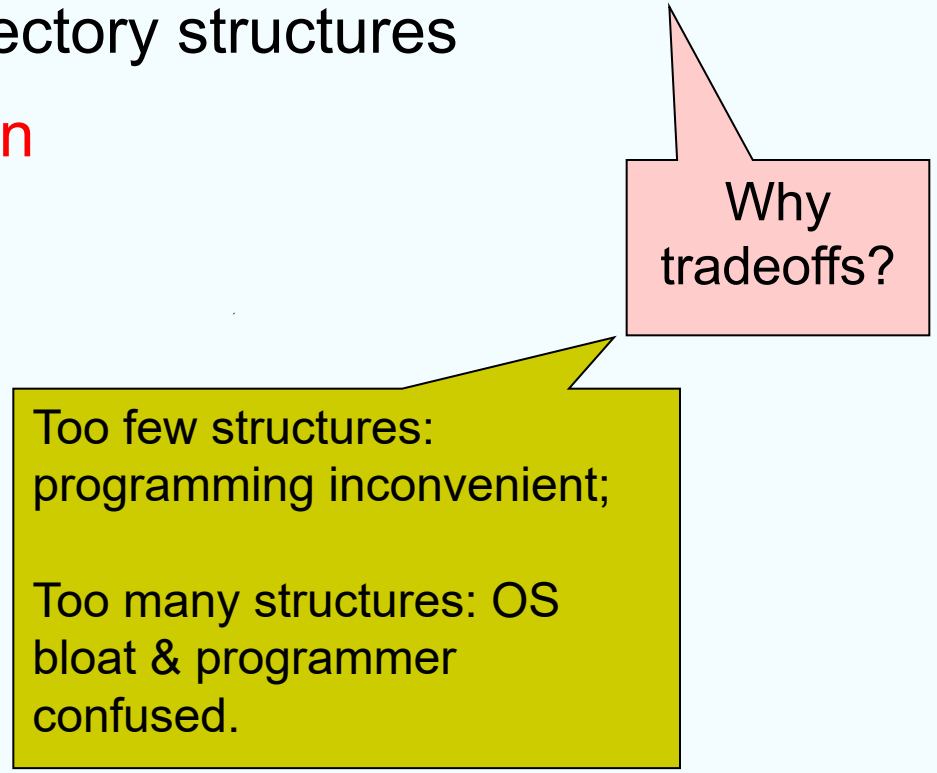
Mind Map for Ch10 & 11

File system interface:



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
- To explore file-system protection



What Is a File System?

- The way that controls how data is stored and retrieved in a storage medium.
  - File naming
  - Where files are placed
  - Metadata
  - Access rules

File Concept

- Contiguous logical address space
- A sequence of bits, bytes, lines, or records. The meaning is defined by the creator and user.
- Types:
  - Data
    - numeric
    - character
    - binary
  - Program
    - Source
    - Object
    - Executable

File Structure

- None - sequence of words, 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
- **Identifier** – unique tag (number) identifies file within file system
- **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 are kept in the directory structure, which is maintained on the disk

File Operations

- File is an **abstract data type**
- **Create**
- **Write –** define a pointer
- **Read –** use the same pointer

  Per-process **current file-position pointer**
- **Reposition within file** (file seek)
- **Delete**
- **Truncate**
- *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

```
Class File{
Public:
    Create();
    Write();
    Read();
    Seek();
    ……
}
```

Open-file table

- Open() system call returns a pointer to an entry in the open-file table
- Per-process table
  - Current file pointer
  - Access rights
  - …
- System-wide table
  - Open count
  - …

Open Files

- Several pieces of data are needed to manage 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 – system doesn't need to read it from disk for every operation.
  - Access rights: per-process access mode information

Open File Locking

- Provided by some operating systems and file systems
- Mediates access to a file (by multiple processes)
- File locks are similar to reader-writer locks
  - Shared lock (reader)
  - Exclusive lock (writer)
- 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

File Locking Example – Java API

```
import java.io.*;
import java.nio.channels.*;
public class LockingExample {
    public static final boolean EXCLUSIVE = false;
    public static final boolean SHARED = true;
    public static void main(String arsg[]) throws IOException {
        FileLock sharedLock = null;
        FileLock exclusiveLock = null;
        try {
            RandomAccessFile raf = new RandomAccessFile("file.txt", "rw");
            // get the channel for the file
            FileChannel ch = raf.getChannel();
            // this locks the first half of the file - exclusive
            exclusiveLock = ch.lock(0, raf.length()/2, EXCLUSIVE);
            /** Now modify the data . . . */
            // release the lock
            exclusiveLock.release();
```

File Locking Example – Java API (cont)

```
                    // this locks the second half of the file - shared
                    sharedLock = ch.lock(raf.length()/2+1, raf.length(),
                                         SHARED);
                    /** Now read the data . . . */
                    // release the lock
                    sharedLock.release();
                } catch (java.io.IOException ioe) {
                    System.err.println(ioe);
                }finally {
                    if (exclusiveLock != null)
                    exclusiveLock.release();
                    if (sharedLock != null)
                    sharedLock.release();
                }
            }
        }
```

File Types – Name, 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 Types

- MS-DOS
- MAC OS X
  - Each file has a creator attribute containing the name of the program that created it.
- UNIX
  - Magic number (executable, shell script, postscript, …)

Access Methods

- **Sequential Access**

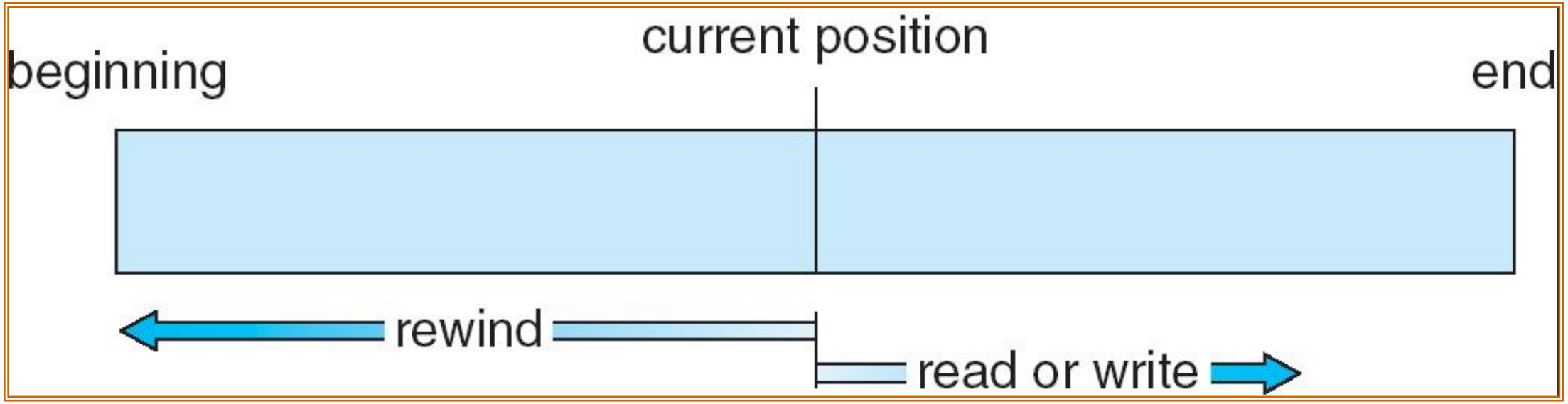
  read next
  write next
  reset
  no read after last write
  (rewrite)

- **Direct (Random) Access**

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

  n = relative block number

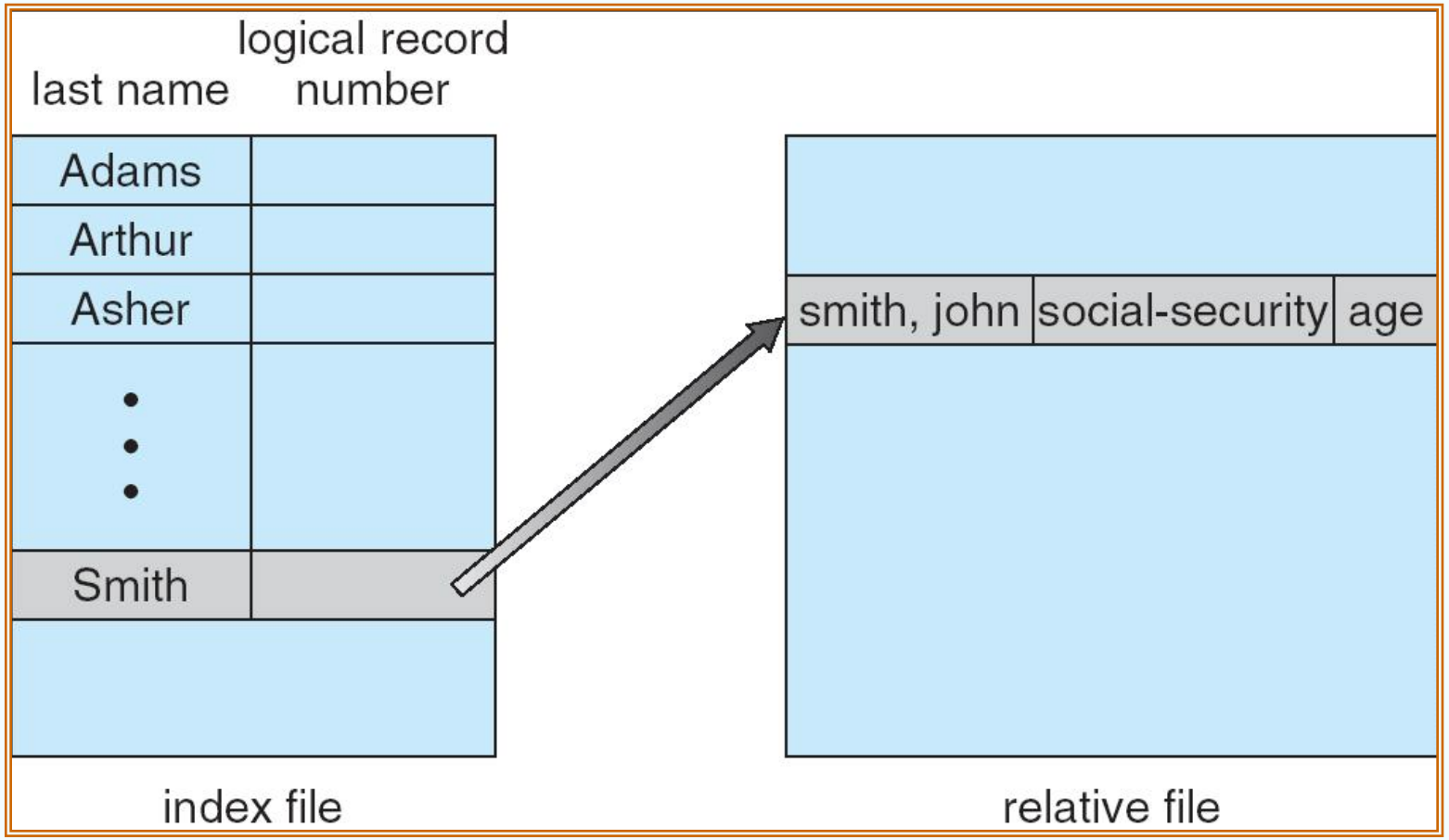
Sequential-access File



Simulation of Sequential Access on a Direct-access File

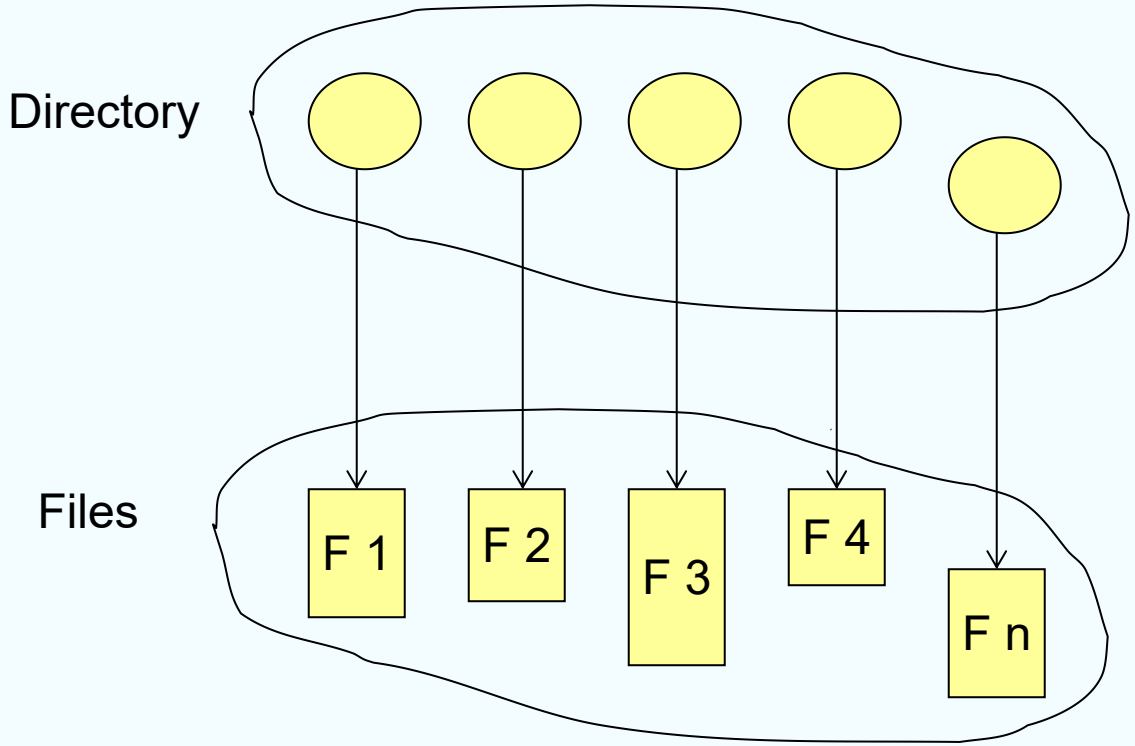
sequential access	implementation for direct access
reset	$cp = 0;$
read next	*read cp*; $cp = cp + 1;$
write next	*write cp*; $cp = cp + 1;$

Example of Index and Relative Files



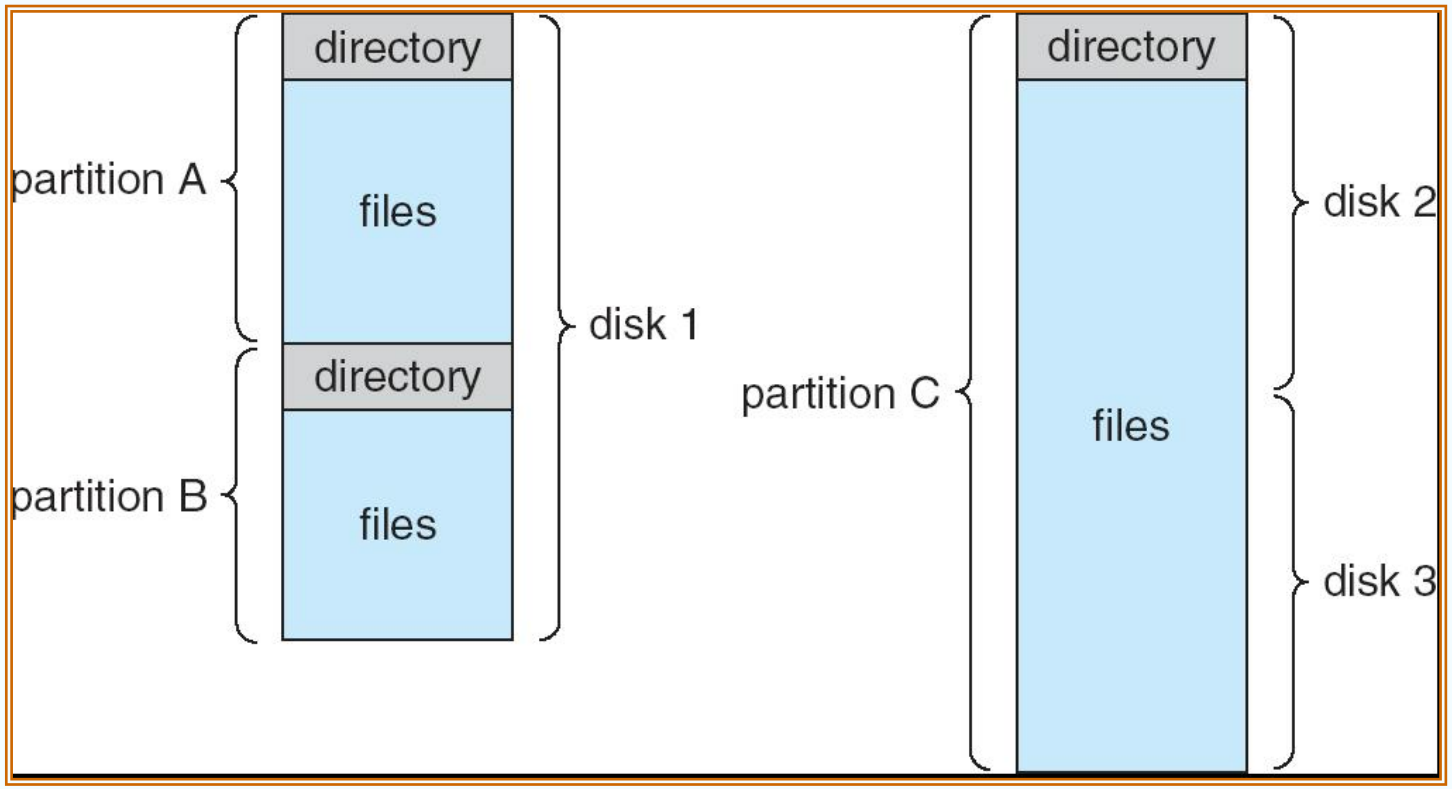
Directory Structure

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



Both the directory structure and the files reside on disk

A Typical File-system Organization



The directory records information about the files in the system – such as name, location, size and type.

Operations Performed on Directory

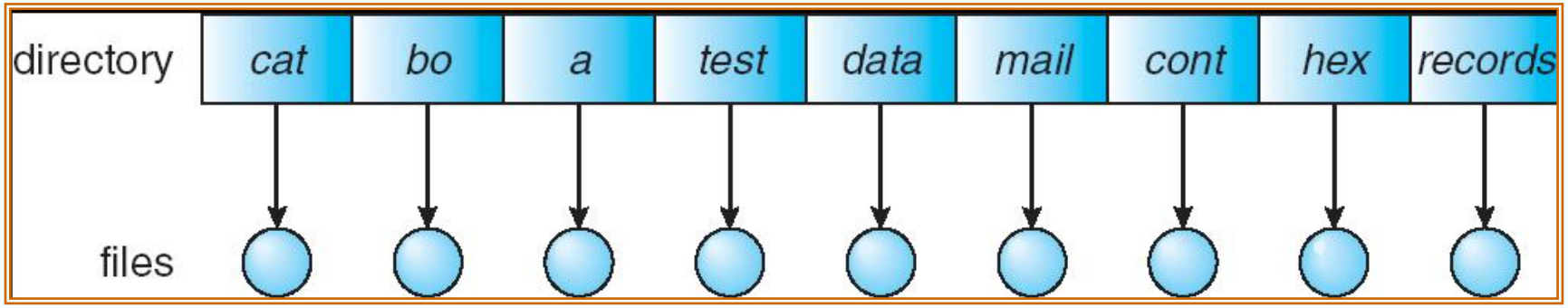
- Search for a file
- Create a file
- Delete a file
- List a directory
- Rename a file
- Traverse the file system – access every dir and file for backing up.

Organize the Directory (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

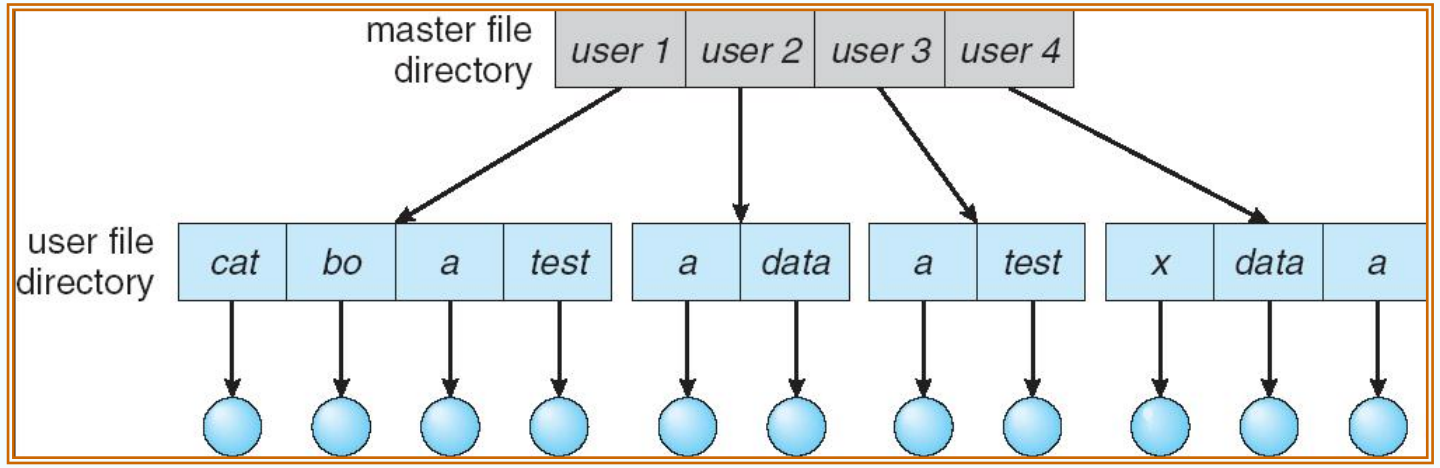


Naming problem

Grouping problem

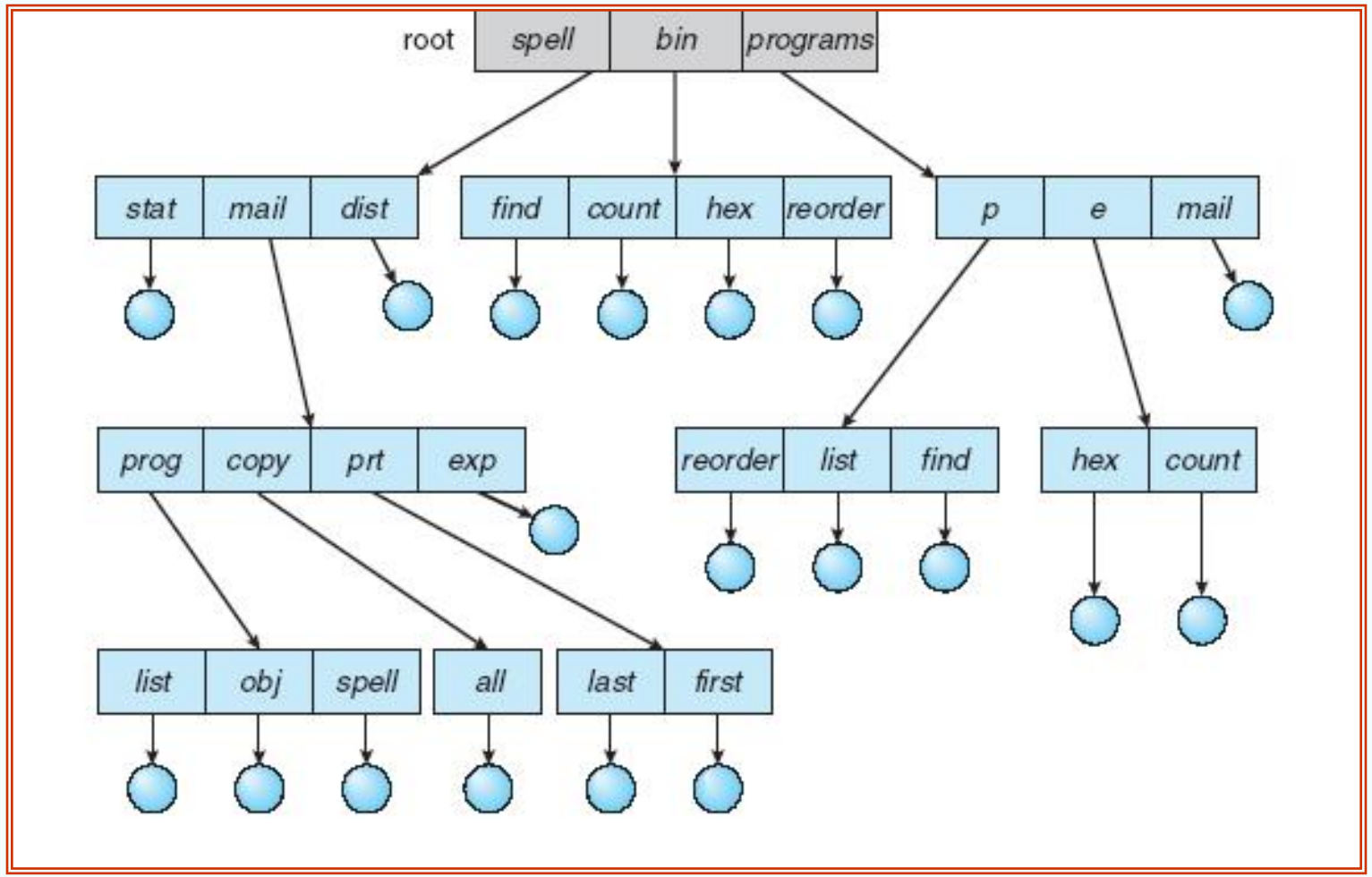
Two-Level Directory

- 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)

- Each directory entry contains a bit defining the entry as file(0) or directory(1).
- 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
- Delete a file

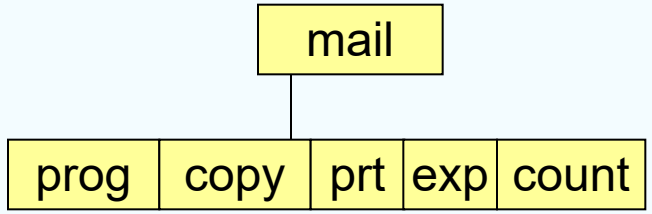
  rm <file-name>

- Creating a new subdirectory is done in current directory

  mkdir <dir-name>

  Example: if in current directory /mail

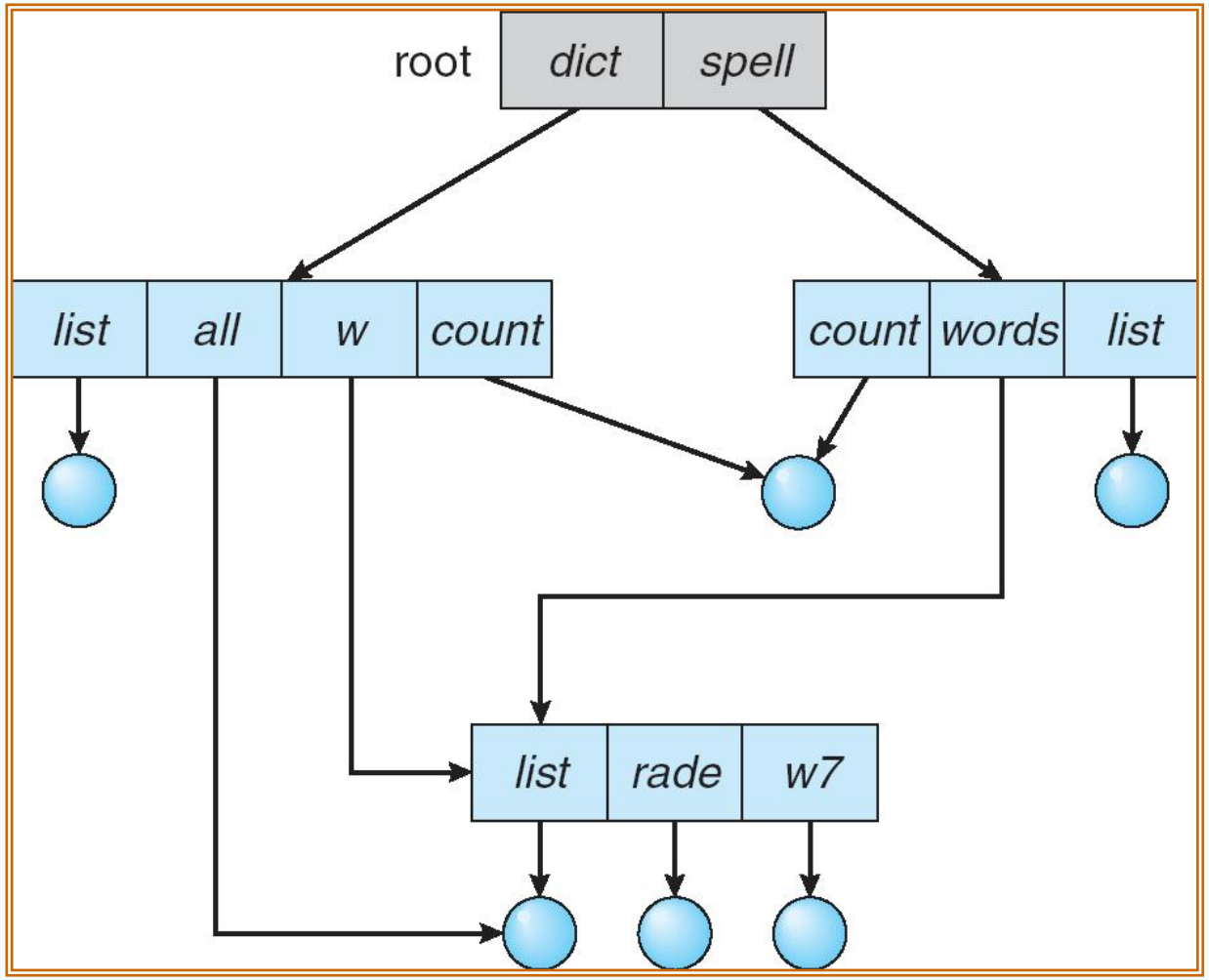
  mkdir count



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

Acyclic-Graph Directories

- Requirement for file sharing
- Have shared subdirectories and files



Acyclic-Graph Directories (Cont.)

- Two different names (aliasing)

- If *dict* deletes *count* $\Rightarrow$ dangling pointer

  Solutions:

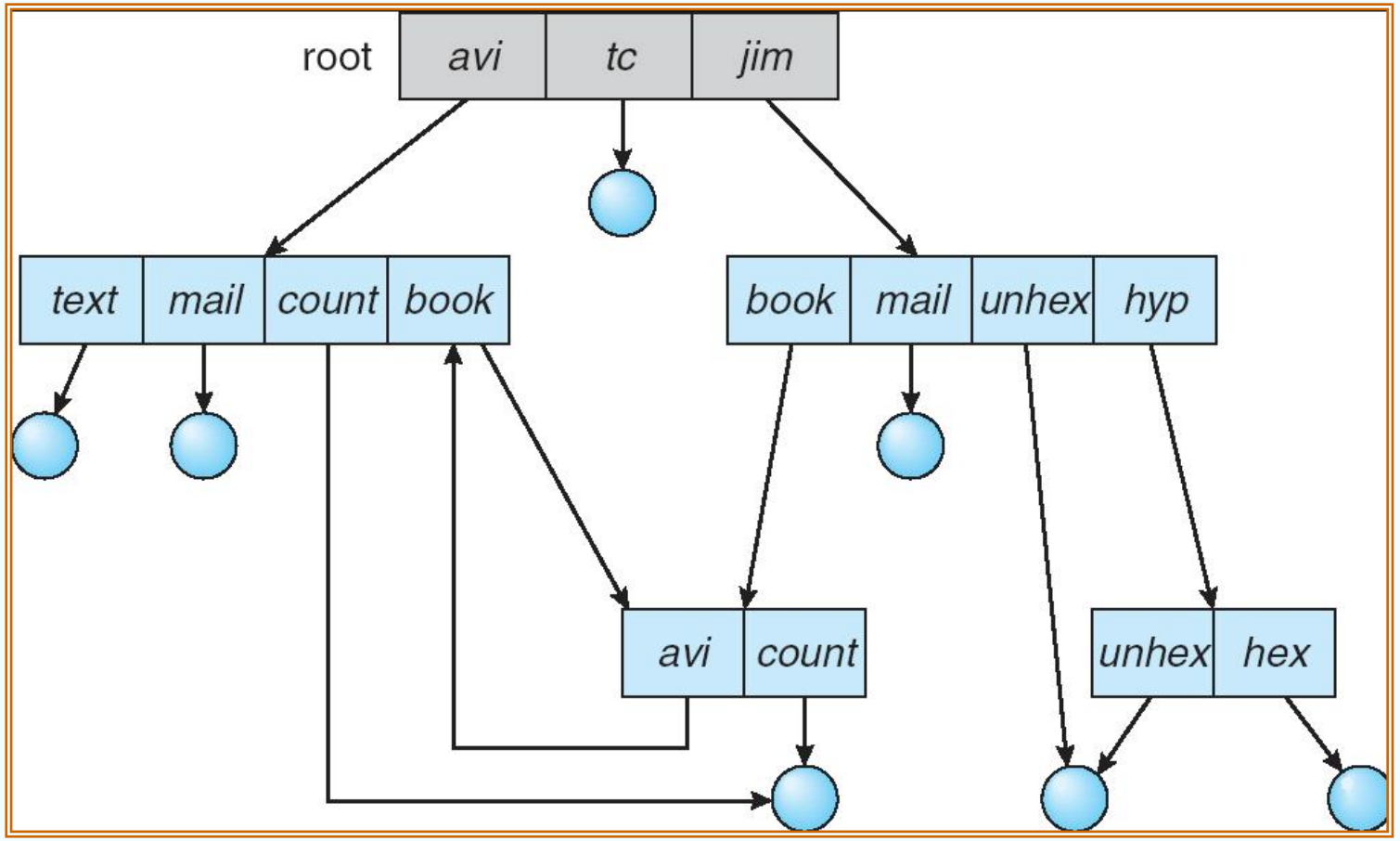
  - Backpointers (keep a list of references to a file), so we can delete all pointers

    But: Large, variable size reference list is a problem

- New directory entry type
  - **Link** – another name (pointer) to an existing file
  - **Resolve the link** – follow pointer to locate the file

General Graph Directory

A serious problem with acyclic-graph is to ensure no cycles.



General Graph Directory (Cont.)

- If cycles allowed
  - Repeated search the same object
  - File deletion problem (count <>0 even if unused)
- 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

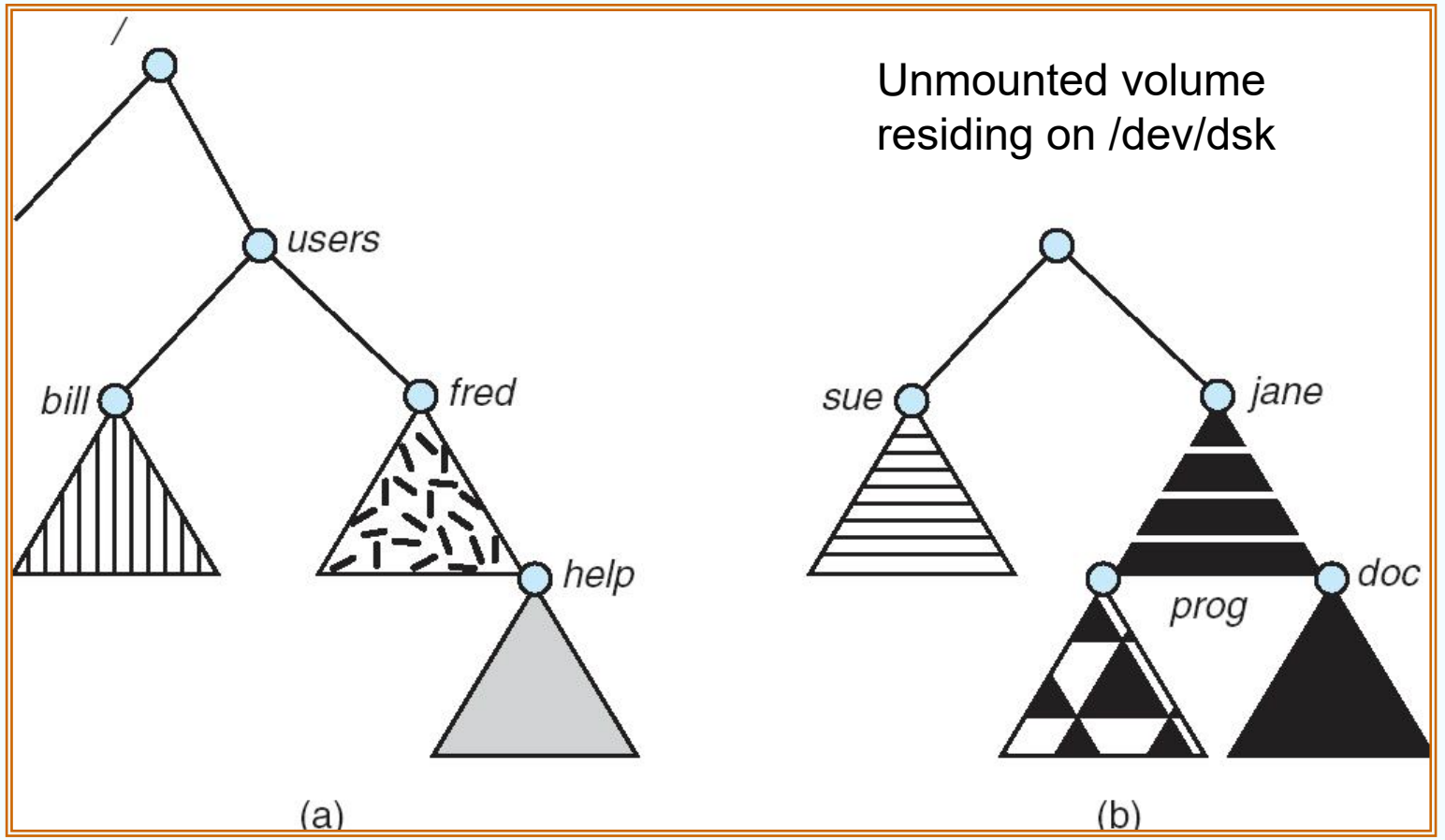
Soft (Symbolic) Link vs. Hard Link

- A soft link is a separate file that points to the original file by storing its path.
- A hard link is an additional name for an existing file. It increases the file's link count, which is a count of how many names (links) a file has.
- The soft link has its own inode (FCB), and its data contains the path to the linked file, not the file data itself.
- Hard Link: Both the original file and the hard link point to the same inode (FCB).
- Soft links can span file systems since they are simply paths to other files.
- Hard links cannot span file systems; one cannot create a hard link for a directory to prevent the creation of cycles.

File System Mounting

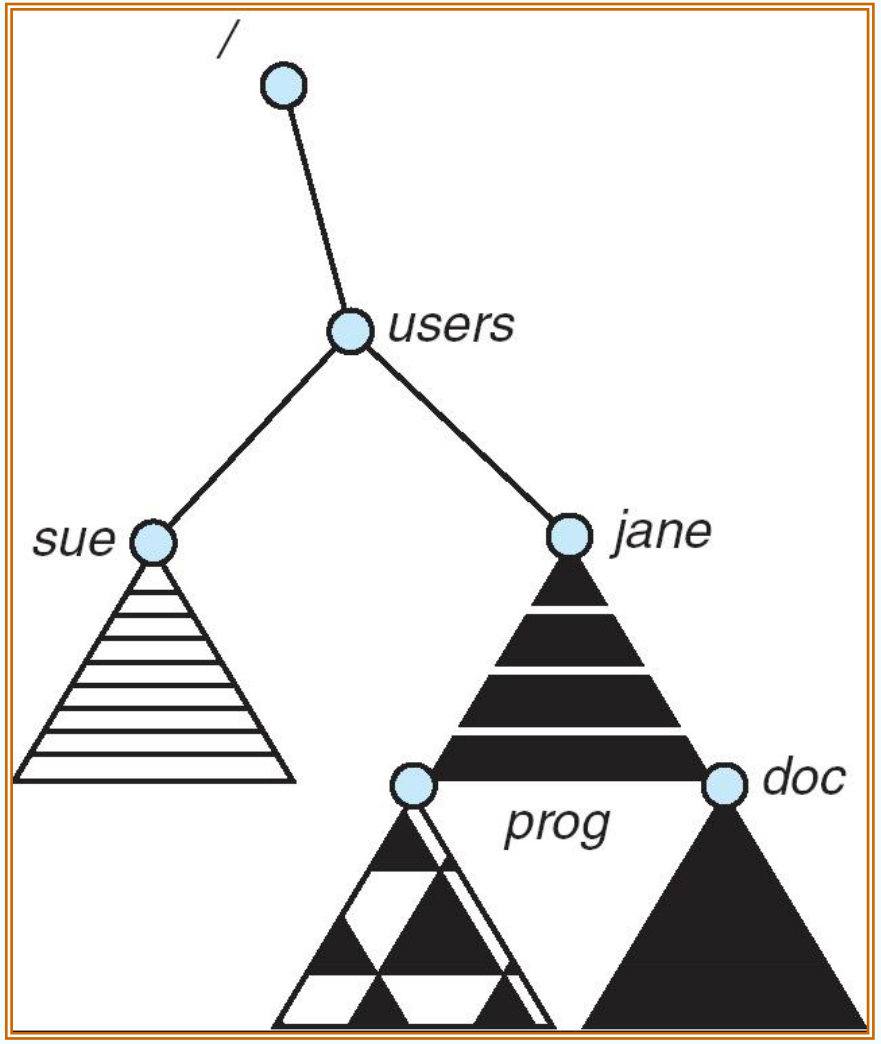
- A file system must be **mounted** before it can be accessed
- An un-mounted file system (i.e. Fig. 10-12(b)) is mounted at a **mount point**

(a) Existing. (b) Unmounted Partition



Mount Point

```
$ mount  /dev/dsk  /users
```



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 Sharing – Multiple Users

- **User IDs** identify users, allowing permissions and protections to be per-user
- **Group IDs** allow users to be in groups, permitting group access rights

File Sharing – Remote File Systems

- Uses networking to allow file system access between systems
  - Manually via programs like FTP
  - Automatically, seamlessly using **distributed file systems**
  - Semi automatically via the **world wide web**
- **Client-server** model allows clients to mount remote file systems from servers
  - Server can serve multiple clients
  - Client and user-on-client identification is insecure or complicated
  - **NFS** is standard UNIX client-server file sharing protocol
  - **CIFS** is standard Windows protocol
  - Standard operating system file calls are translated into remote calls
- Distributed Information Systems **(distributed naming services)** such as LDAP, DNS, NIS, Active Directory implement unified access to information needed for remote computing

File Sharing – Failure Modes

- Remote file systems add new failure modes, due to network failure, server failure
- Recovery from failure can involve state information about status of each remote request
- Stateless protocols such as NFS include all information in each request, allowing easy recovery but less security

File Sharing – Consistency Semantics

- **Consistency semantics** specify how multiple users are to access a shared file simultaneously
  - Similar to Ch 6 process synchronization algorithms
    - Tend to be less complex due to disk I/O and network latency (for remote file systems) – slow speed
  - Andrew File System (AFS) implemented complex remote file sharing semantics
  - Unix file system (UFS) implements:
    - Writes to an open file visible immediately to other users of the same open file
    - Sharing file pointer to allow multiple users to read and write concurrently
  - AFS has session semantics
    - Writes only visible to sessions starting after the file is closed

Protection

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

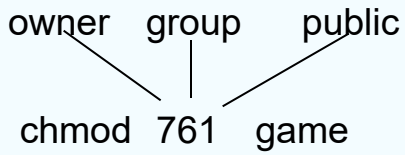
- Types of access
  - **Read**
  - **Write**
  - **Execute**
  - **Append**
  - **Delete**
  - **List**

Access Lists and Groups

- Mode of access: read, write, execute
- Three classes of users

			RWX
a) **owner access**	7	⇒	1 1 1
b) **group access**	6	⇒	RWX 1 1 0
c) **public access**	1	⇒	RWX 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.



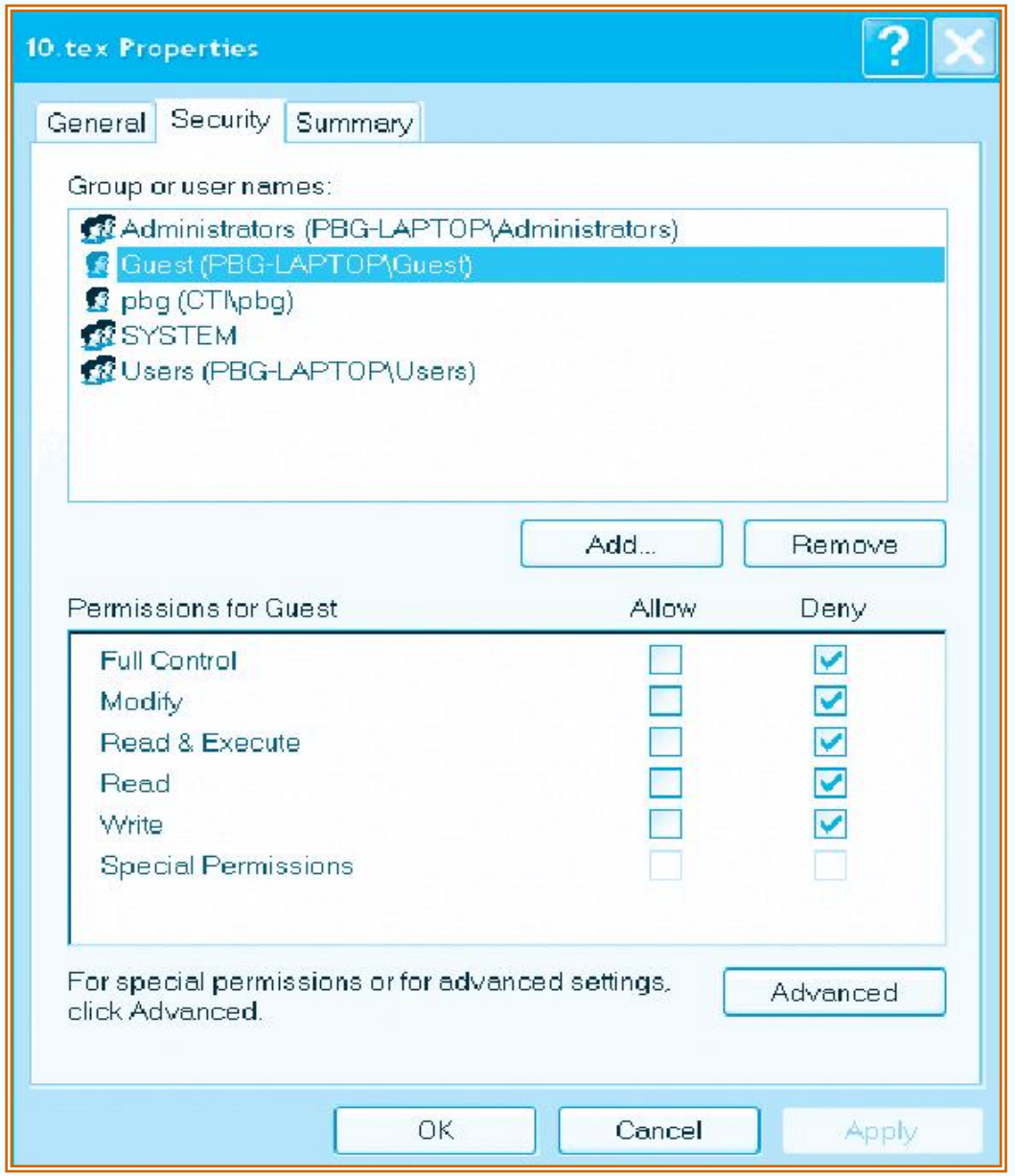
Attach a group to a file

chgrp G game

A Sample UNIX Directory Listing

-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/

Windows XP Access-control List Management



End of Chapter 10