11. File-System Interface

Sungyoung Lee

College of Engineering

KyungHee University

Contents

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

Basic Concept

- Requirements for long-term information storage
  - It must be possible to store a very large amount of information
  - The information must survive the termination of the process using it
  - Multiple processes must be able to access the information concurrently

- File system
  - Implement an abstraction for secondary storage (files)
  - Organize files logically (directories)
  - Permit sharing of data between processes, people, and machines
  - Protect data from unwanted access (security)

Files

- File
  - A named collection of related information that is recorded on secondary storage
    - persistent through power failures and system reboots
  - OS provides a uniform logical view of information storage via files

- File structures
  - Flat: byte sequence
  - Structured:
    - Lines
    - Fixed length records
    - Variable length records

File Concept

- Contiguous logical address space

- Types:
  - Data
    - numeric
    - character
    - binary
  - Program

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
- 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 Attributes

- Attributes or metadata

Attribute	Meaning
Protection	Who can access the file and in what way
Password	Password needed to access the file
Creator	ID of the person who created the file
Owner	Current owner
Read-only flag	0 for read/write; 1 for read only
Hidden flag	0 for normal; 1 for do not display in listings
System flag	0 for normal files; 1 for system file
Archive flag	0 for has been backed up; 1 for needs to be backed up
ASCII/binary flag	0 for ASCII file; 1 for binary file
Random access flag	0 for sequential access only; 1 for random access
Temporary flag	0 for normal; 1 for delete file on process exit
Lock flags	0 for unlocked; nonzero for locked
Record length	Number of bytes in a record
Key position	Offset of the key within each record
Key length	Number of bytes in the key field
Creation time	Date and time the file was created
Time of last access	Date and time the file was last accessed
Time of last change	Date and time the file has last changed
Current size	Number of bytes in the file
Maximum size	Number of bytes the file may grow to

File Operations

- Create
- Write
- Read
- 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

File Operations

- Unix operations

```
int creat (const char *pathname, mode_t mode);
int open (const char *pathname, int flags, mode_t mode);
int close (int fd);
ssize_t read (int fd, void *buf, size_t count);
ssize_t write (int fd, const void *buf, size_t count);
off_t lseek (int fd, off_t offset, int whence);
int stat (const char *pathname, struct stat *buf);
int chmod (const char *pathname, mode_t mode);
int chown (const char *pathname, uid_t owner, gid_t grp);
int flock (int fd, int operation);
int fcntl (int fd, int cmd, long arg);
```

File Types

- Files may have types
  - Understood by file systems
    - device, directory, symbolic link, etc.
  - Understood by other parts of OS or runtime libraries
    - executable, dll, source code, object code, text, etc.
  - Understood by application programs
    - jpg, mpg, avi, mp3, etc.

- Encoding file types
  - Windows encodes type in name
    - .com, .exe, .bat, .dll, .jpg, .avi, .mp3, etc.
  - Unix encodes type in contents
    - magic numbers (e.g., 0xcafebabe for Java class files)
    - initial characters (e.g., #! for shell scripts)

File Types – Name, 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 compressed, for archiving or storage
multimedia	mpeg, mov, rm	binary file containing audio or A/V information

File Access

- Some file systems provide different access methods that specify different ways for accessing data in a file.

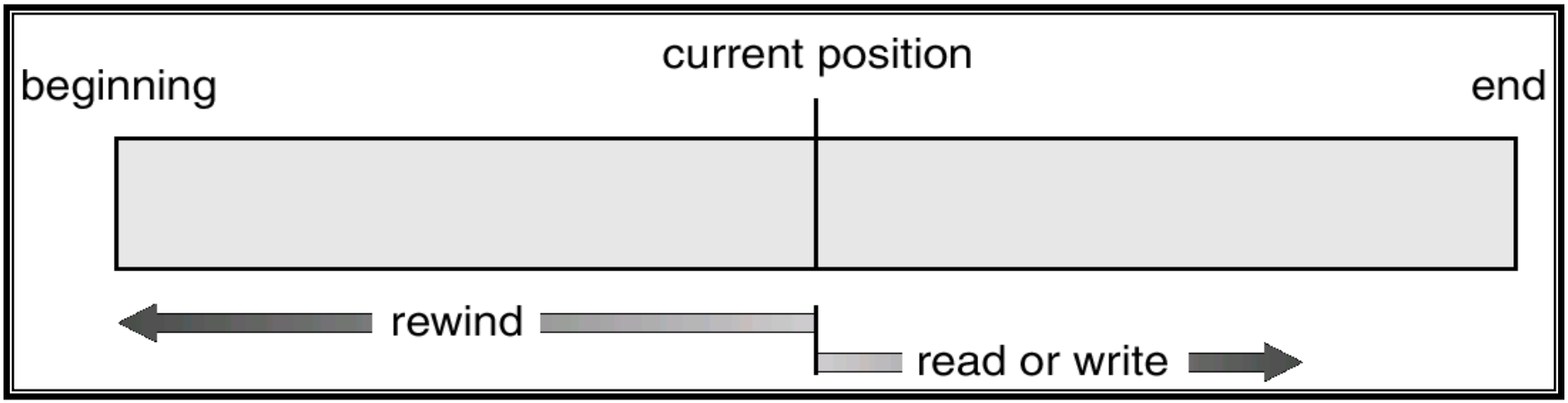
- Sequential access
  - read bytes one at a time, in order
- Direct access
  - random access given block/byte number
- Record access
  - File is an array of fixed- or variable-length records, read/written sequentially or randomly by record #
- Index access
  - File system contains an index to a particular field of each record in a file, reads specify a value for that field and the system finds the records via the index (DBs)

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

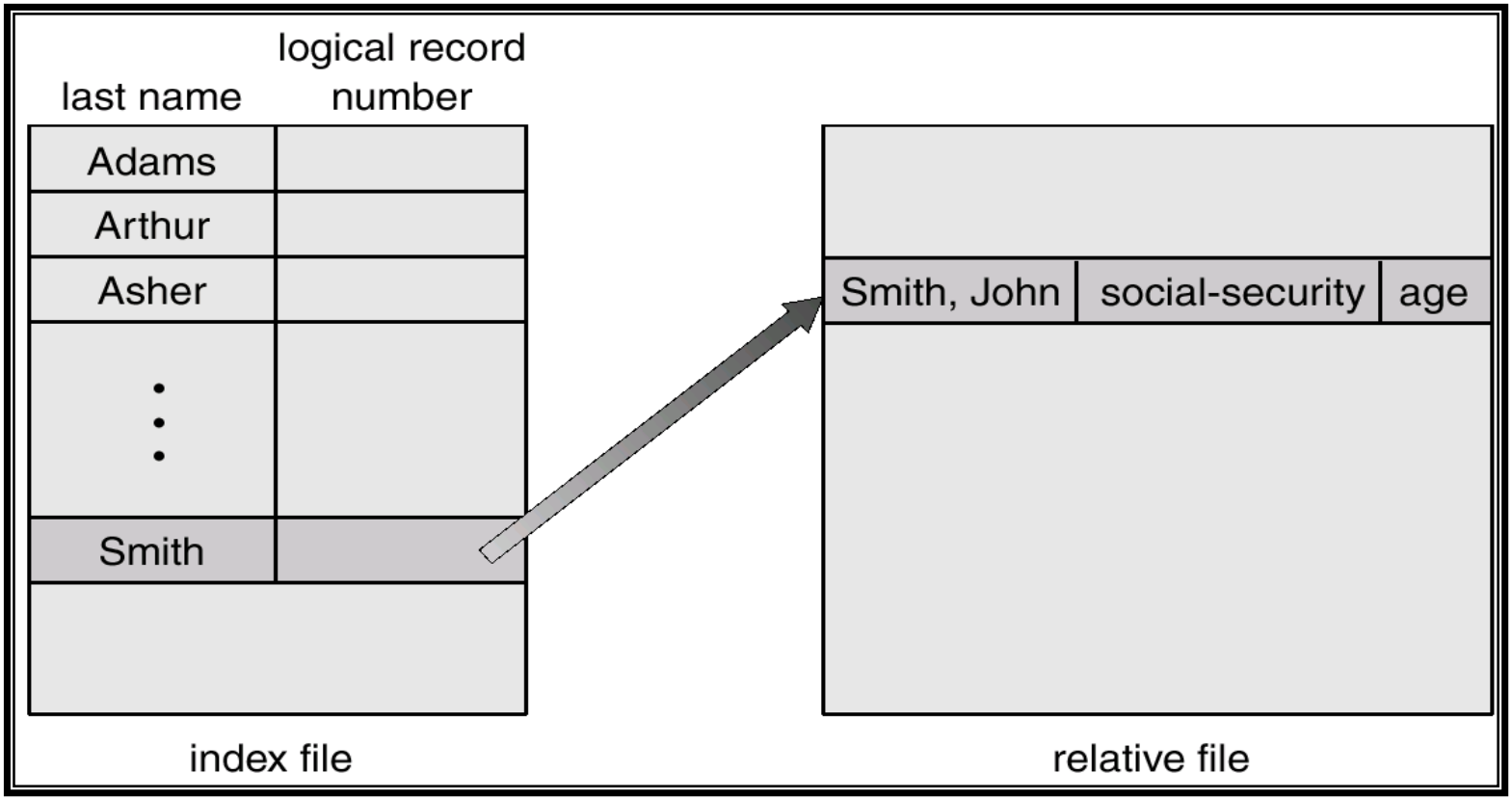
Sequential-access File



Simulation of Sequential Access on a Direct-access File

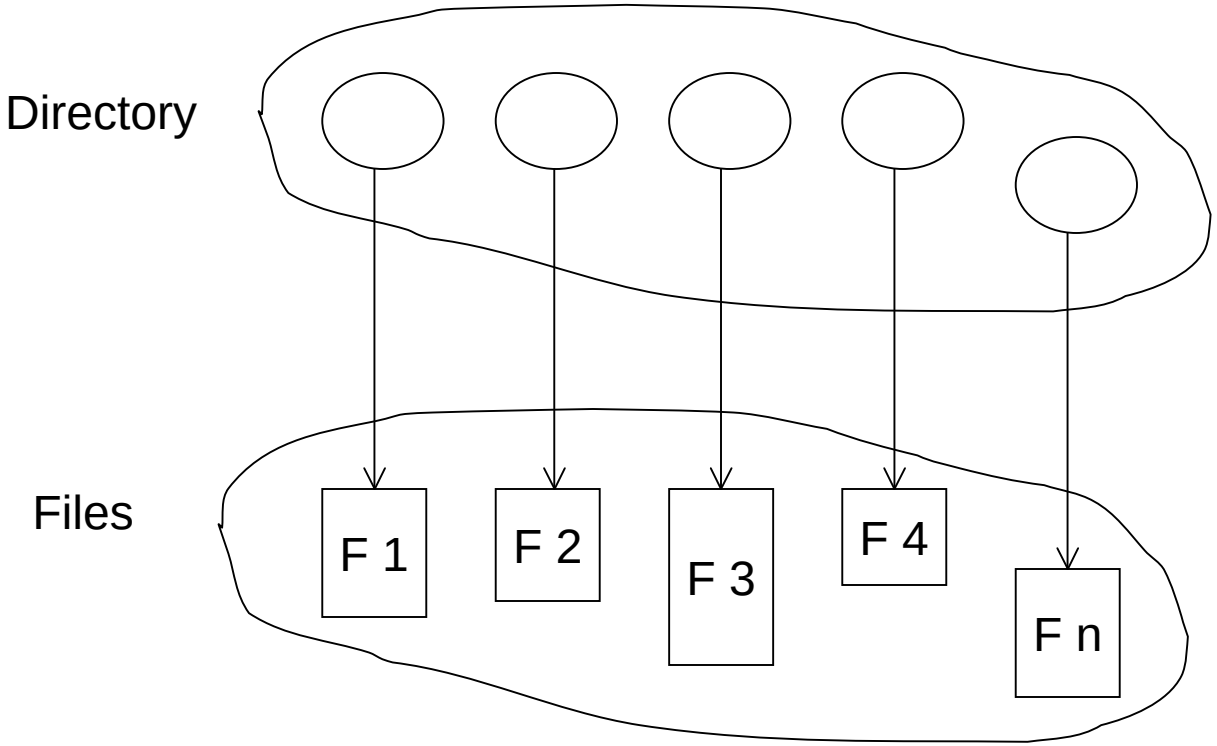
sequential access	implementation for direct access
reset	*cp* = 0;
read next	*read cp*; *cp* = *cp*+1;
write next	*write cp*; *cp* = *cp*+1;

Example of Index and Relative Files



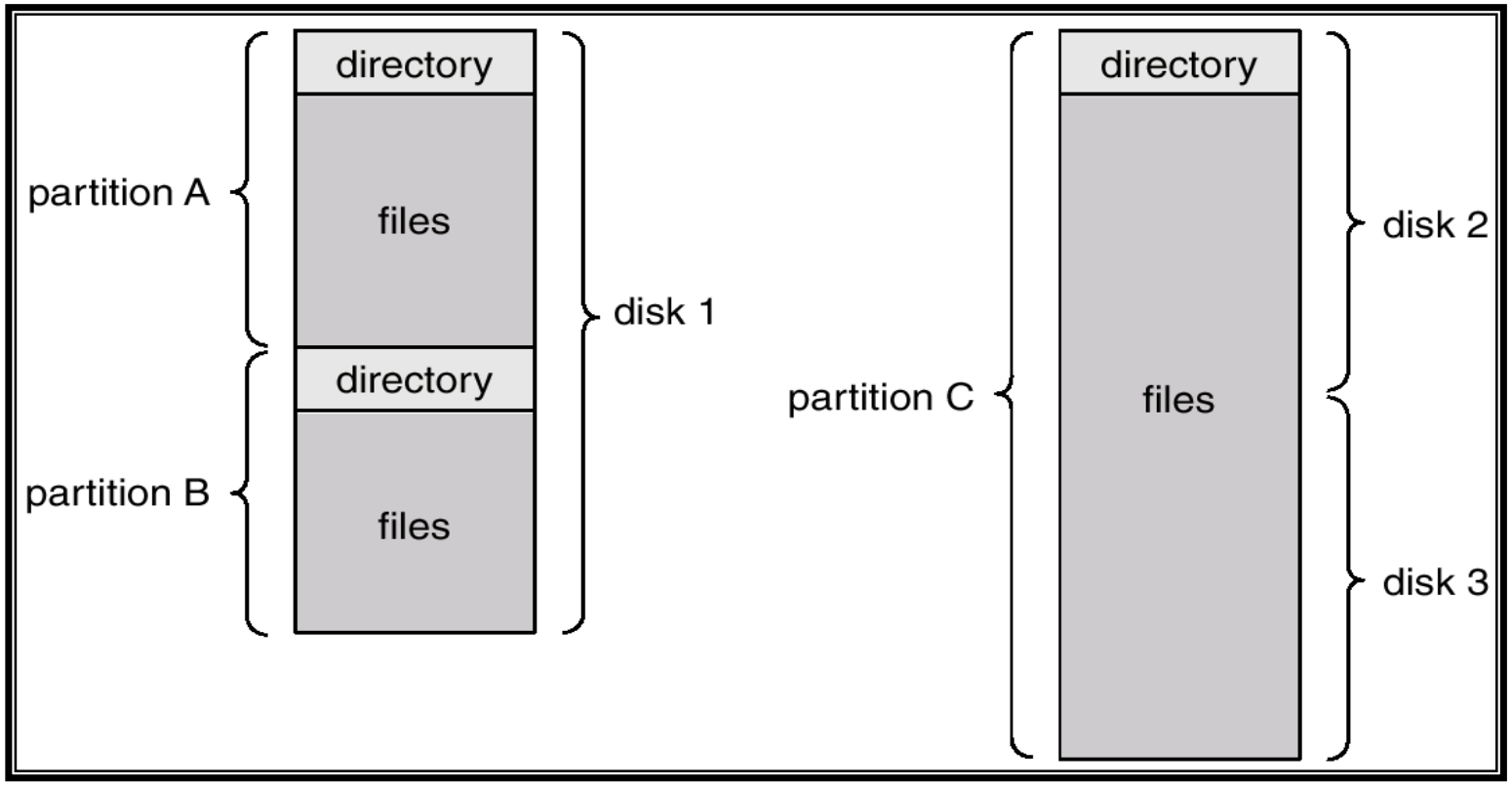
Directory Structure

- A collection of nodes containing information about all files



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

A Typical File-system Organization



Directories

- Directories
  - For users, they provide a structured way to organize files
  - For the file system, they provide a convenient naming interface that allows the implementation to separate logical file organization from physical file placement on the disk

- A hierarchical directory system
  - Most file systems support multi-level directories
  - Most file systems support the notion of a current directory (or working directory)
    - Relative names specified with respect to current directory
    - Absolute names start from the root of directory tree

Directory Internals

- A directory is …
  - typically just a file that happens to contain special metadata
    - Only need to manage one kind of secondary storage unit
  - directory = list of (file name, file attributes)
  - attributes include such things as:
    - size, protection, creation time, access time,
    - location on disk, etc.
  - usually unordered (effectively random)
    - Entries usually sorted by program that reads directory

Information in a Device Directory

- Name
- Type
- Address
- Current length
- Maximum length
- Date last accessed (for archival)
- Date last updated (for dump)
- Owner ID (who pays)
- Protection information (discuss later)

Operations Performed on Directory

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

Directory Operations

- Unix operations
  - Directories implemented in files
    - Use file operations to manipulate directories
  - C runtime libraries provides a higher-level abstraction for reading directories
    - DIR *opendir (const char *name);
    - struct dirent *readdir (DIR *dir);
    - void seekdir (DIR *dir, off_t offset);
    - int closedir (DIR *dir);
  - Other directory-related system calls
    - int rename (const char *oldpath, const char *newpath);
    - int link (const char *oldpath, const char *newpath);
    - int unlink (const char *pathname);

Organize the Directory (Logically) to Obtain

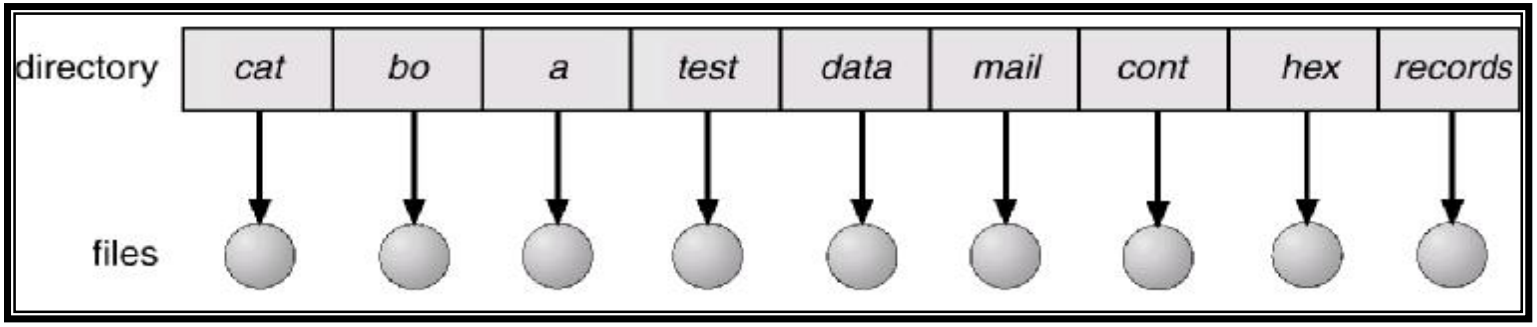
- **Efficiency** – locating a file quickly

- **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

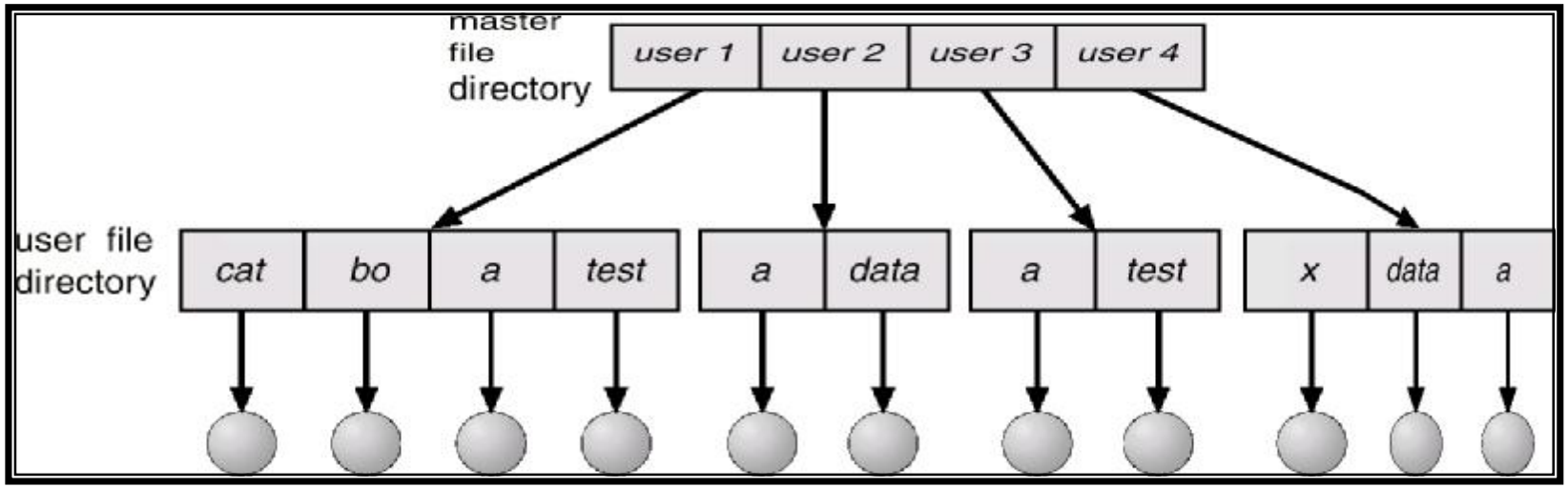


Naming problem

Grouping problem

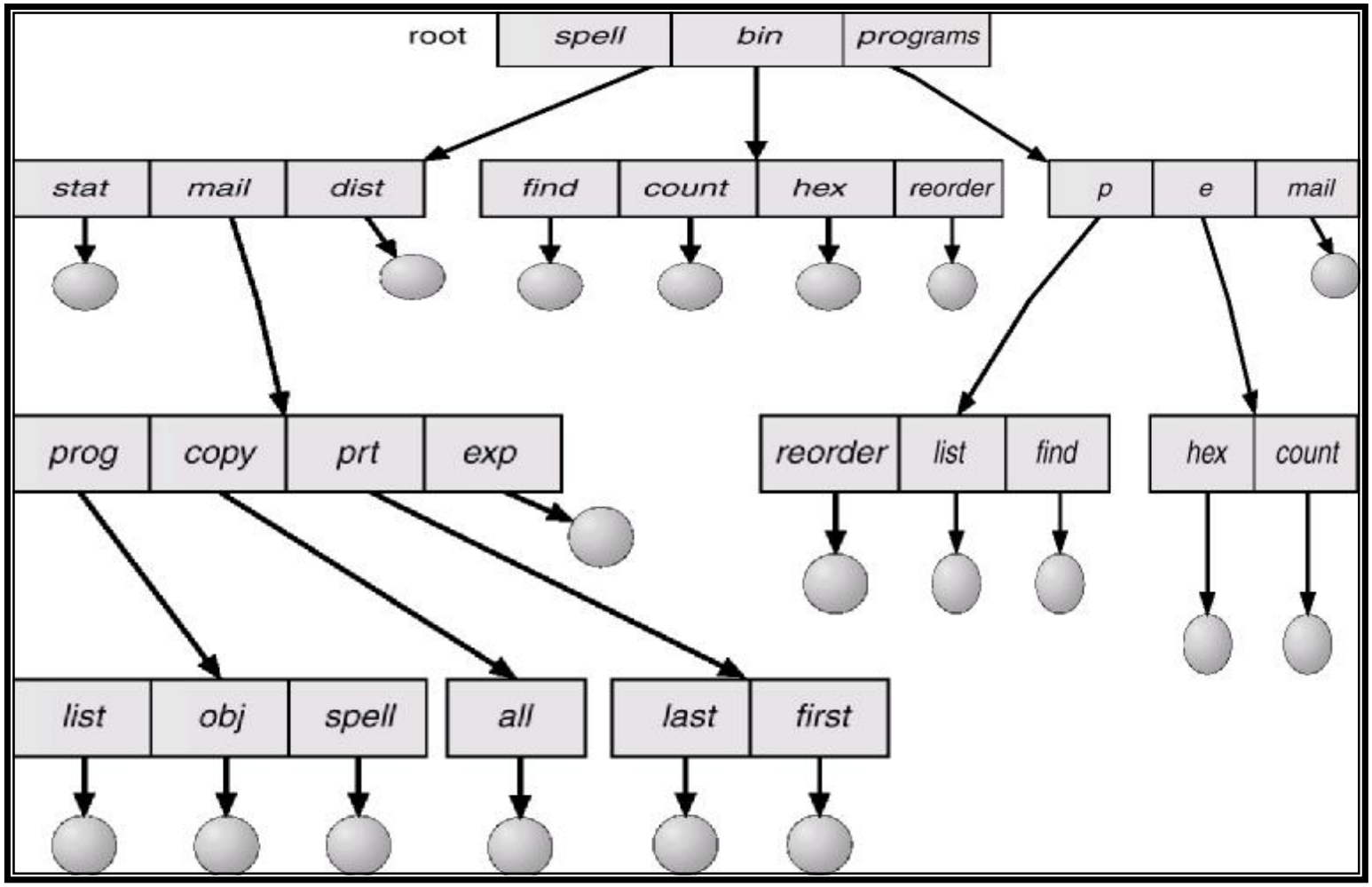
Two-Level Directory

- 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'd)

- Efficient searching
- Grouping Capability
- Current directory (working directory)
  - **cd** /spell/mail/prog
  - **type** list

Tree-Structured Directories (Cont'd)

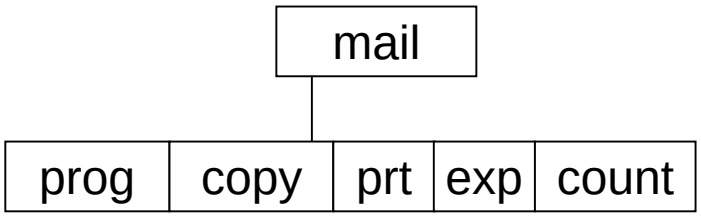
- **Absolute** or **relative** path name
- Creating a new file is done in current directory
- Delete a file

  rm <file-name>
- Creating a new subdirectory is done in current directory

  mkdir <dir-name>

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

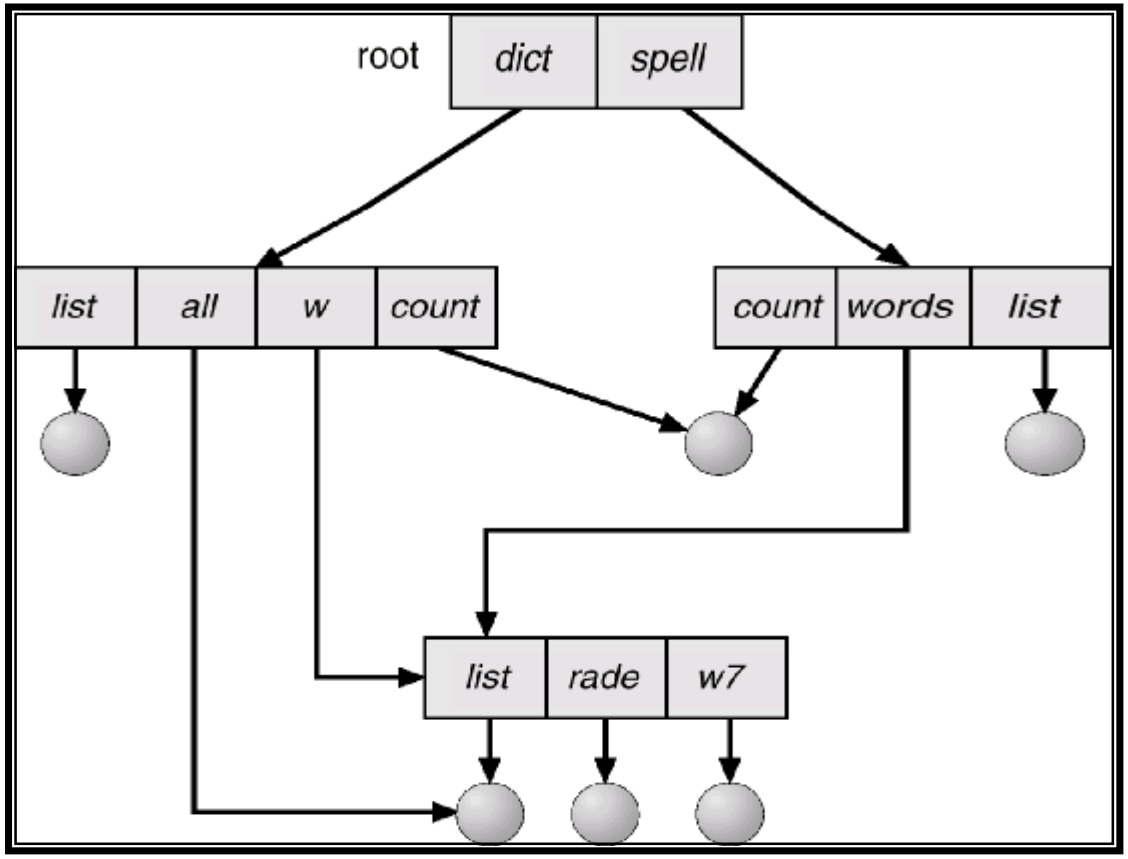
  mkdir count



Deleting "mail" ⇒ deleting the entire subtree rooted by "mail"

Acyclic-Graph Directories

- Have shared subdirectories and files



Acyclic-Graph Directories (Cont'd)

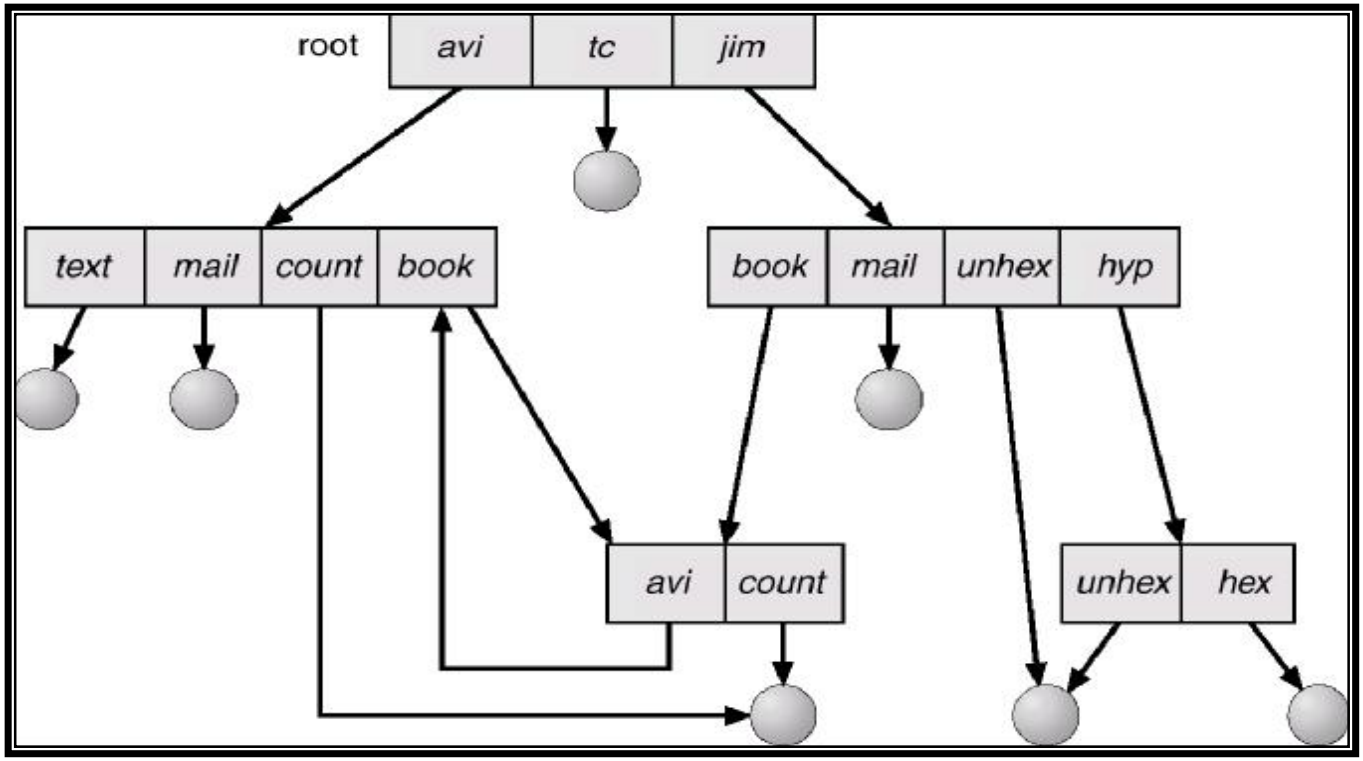
- Two different names (aliasing)

- If *dict* deletes *list* ⇒ 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'd)

- 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

Pathname Translation

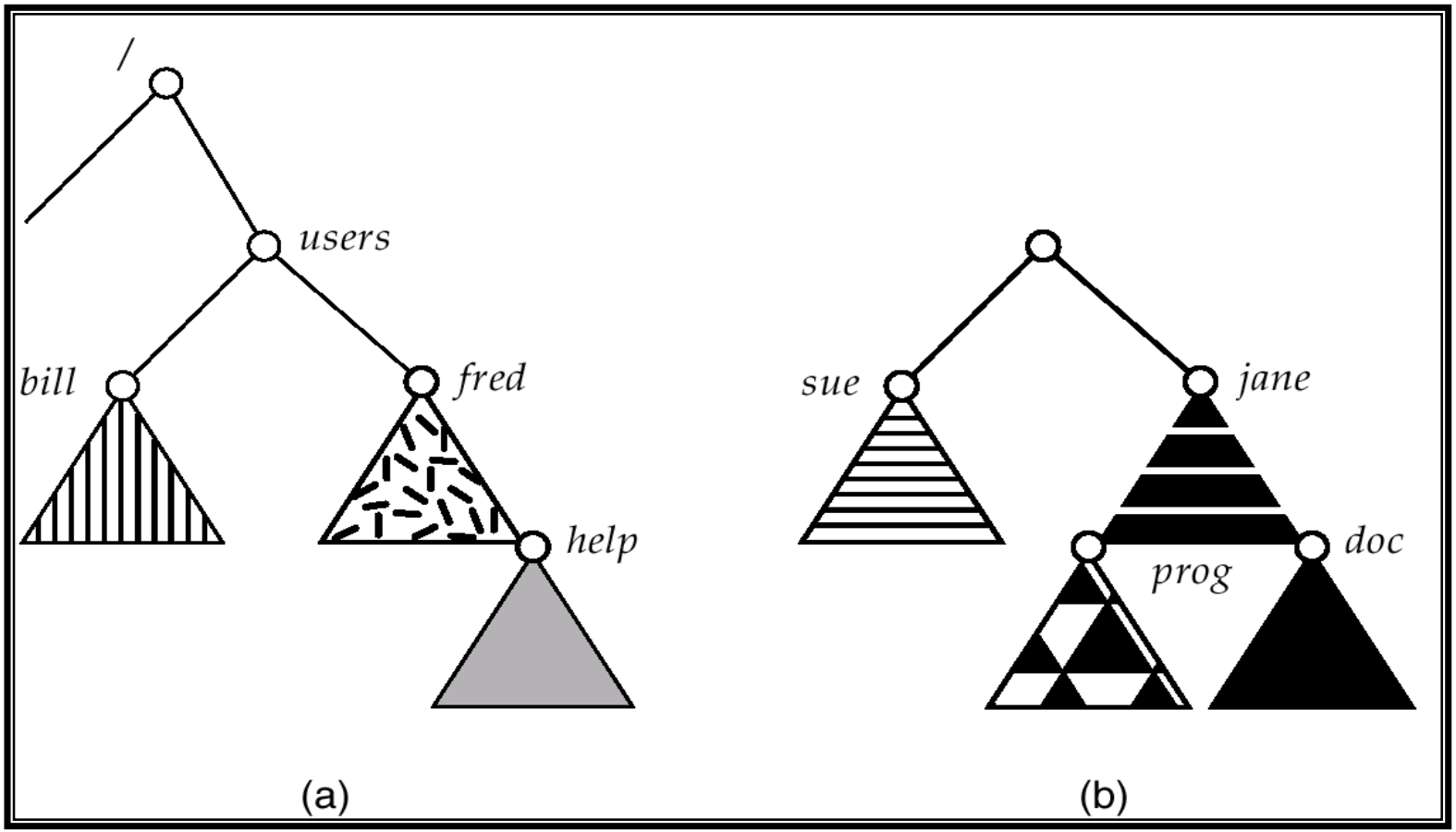
- open(“/a/b/c”, …)
  - Open directory “/” (well known, can always find)
  - Search the directory for “a”, get location of “a”
  - Open directory “a”, search for “b”, get location of “b”
  - Open directory “b”, search for “c”, get location of “c”
  - Open file “c”
  - (Of course, permissions are checked at each step)

- System spends a lot of time walking down directory paths
  - This is why open is separate from read/write
  - OS will cache prefix lookups to enhance performance
    - /a/b, /a/bb, /a/bbb, etc. all share the “/a” prefix

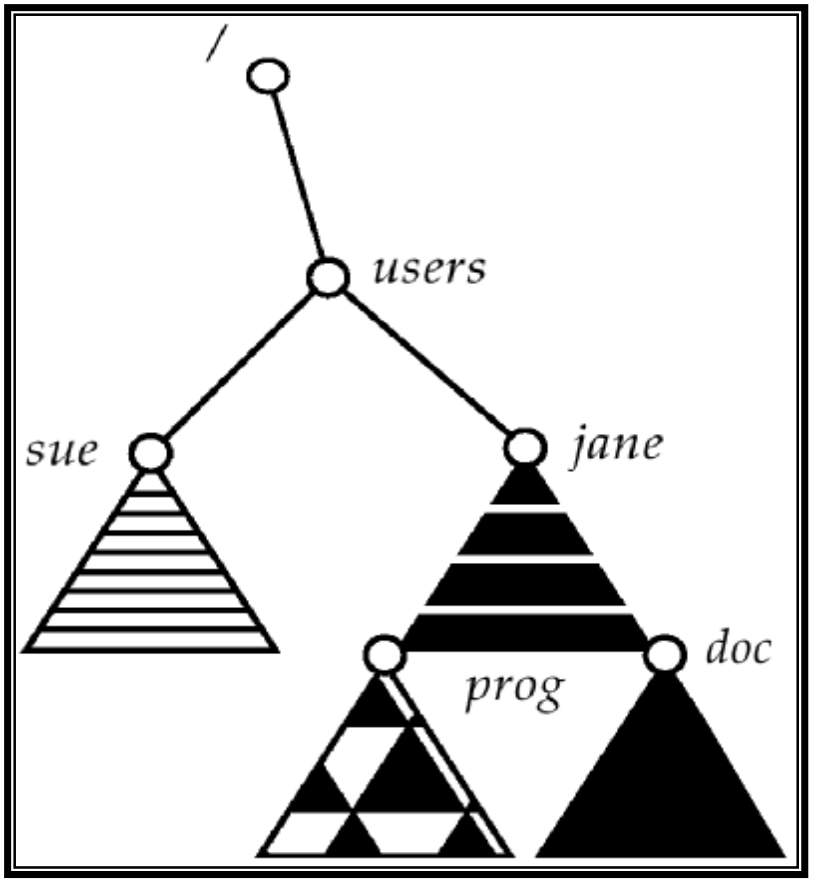
File System Mounting

- A file system must be **mounted** before it can be accessed
- A unmounted file system (i.e. Fig. 11-11(b)) is mounted at a **mount point**
- *Example*
  - Windows: to drive letters (e.g., C:\, D:\, …)
  - Unix: to an existing empty directory (= mount point)

(a) Existing (b) Unmounted Partition



Mount Point



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

Consistency Semantics

- UNIX semantics
  - Writes to an open file are visible immediately to other users that have this file open at the same time.
  - One mode of sharing allows users to share the pointer of current location into the file.
    - via fork() or dup()
- AFS session semantics
  - Writes to an open file are not visible immediately
  - Once a file is closed, the changes made to it are visible only in sessions starting later
- Immutable-shared-files semantics
  - Once a file is declared as shared by its creator, it cannot be modified

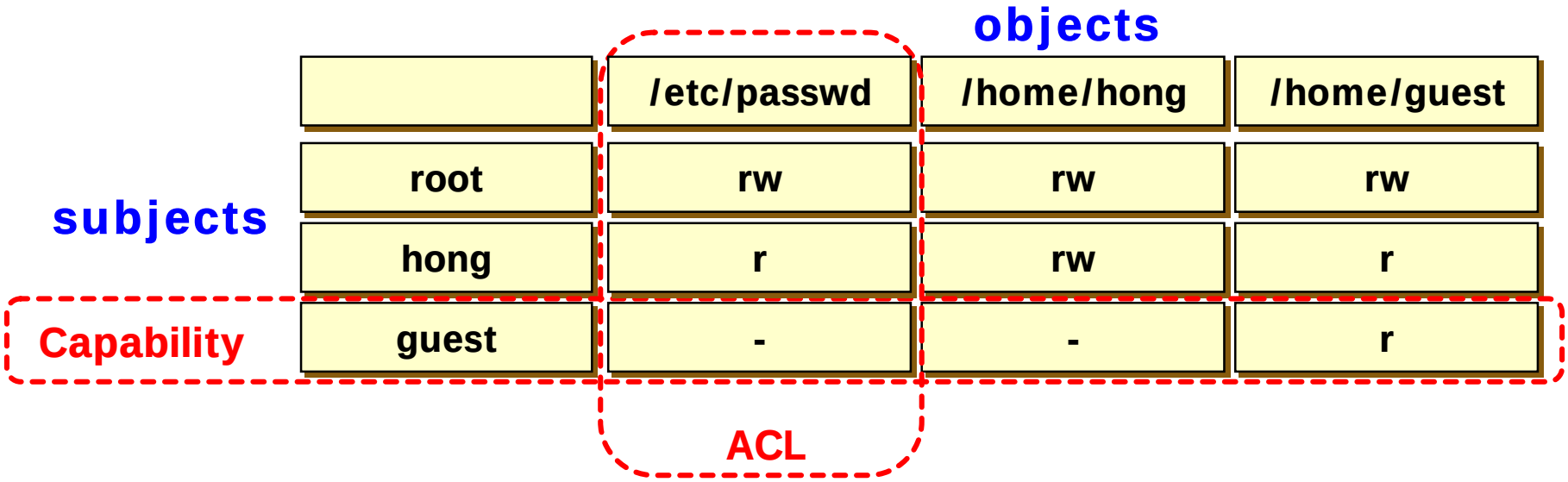
Protection

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

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

Protection

- Representing protection
  - Access control lists (ACLs)
    - For each object, keep list of subjects and their allowed actions
  - Capabilities
    - For each subject, keep list of objects and their allowed actions



Protection (Cont'd)

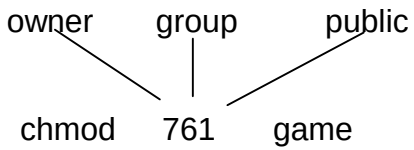
- ACLs vs. Capabilities
  - Two approaches differ only in how the table is represented
  - Capabilities are easy to transfer
    - They are like keys; can hand them off
    - They make sharing easy
  - In practice, ACLs are easier to manage
    - Object-centric, easy to grant and revoke
    - To revoke capabilities, need to keep track of all subjects that have the capability – hard to do, given that subjects can hand off capabilities
  - ACLs grow large when object is heavily shared
    - Can simplify by using "groups"
    - Additional benefit: change group membership affects all objects that have this group in its ACL

Access Lists and Groups

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

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

File Locking

- Advisory lock on a whole file
  - int flock (int fd, int operation)
    - LOCK_SH(shared), LOCK_EX(exclusive), LOCK_UN(unlock)
- POSIX record lock
  - discretionary lock: can lock portions of a file
  - If a process dies, its locks are automatically removed
  - int fcntl (int fd, int cmd, struct flock *lock);
    - cmd: F_GETLK, F_SETLK, F_SETLKW
    - struct flock { type, whence, start, len, pid };
- System V mandatory lock
  - A file is marked as a candidate by setting the setgid bit and removing the group execute bit
  - Must mount the file system to permit mandatory file locks
  - Use the existing flock()/fcntl() to apply locks
  - Every read() and write() is checked for locking