# File Concept

- Contiguous logical address space
- File is a named collection of related information that is recorded on secondary storage
- File is an **abstract data type**
- Types:
  - Data
    - numeric
    - character
    - binary
  - Program

# 1. 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, Updating.
- **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

# 2. File Operations

- **Create**
- **Write**
- **Read**
- **Reposition within 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

- 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
  - Access rights:- per-process access mode information (Protection R/RW).
- Open File Locking:-
- Provided by some operating systems and file systems
- 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

# 3. 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 |

# 4. File Structure

- Sequence of words, bytes
- Simple record structure
  - Lines
  - Fixed length
  - Variable length
- Complex Structures
  - Formatted document
  - Re-locatable load file
- Who decides:
  - Operating system
  - Program

# Access Methods

- **1. Sequential Access**

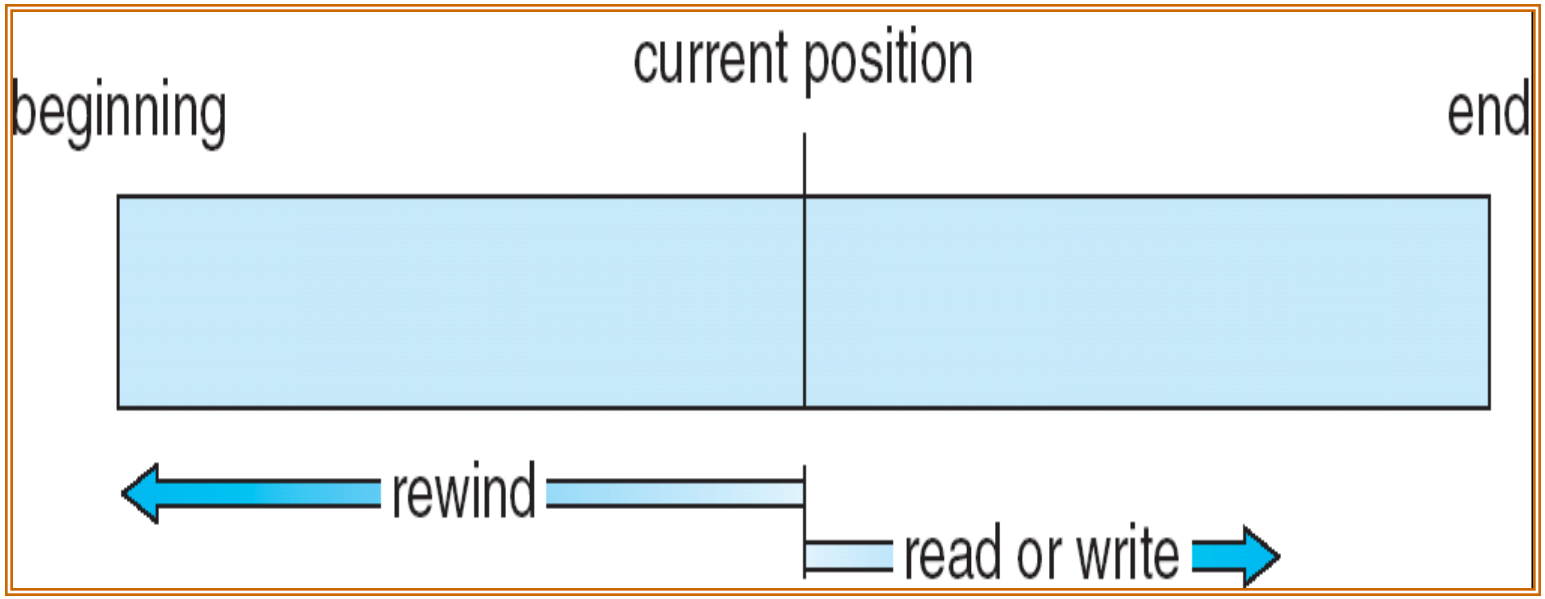
  **read next**
  **write next**
  **reset**
  **no read after last write**
  **(rewrite)**

- **2. Direct Access**

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

*n* = relative block number

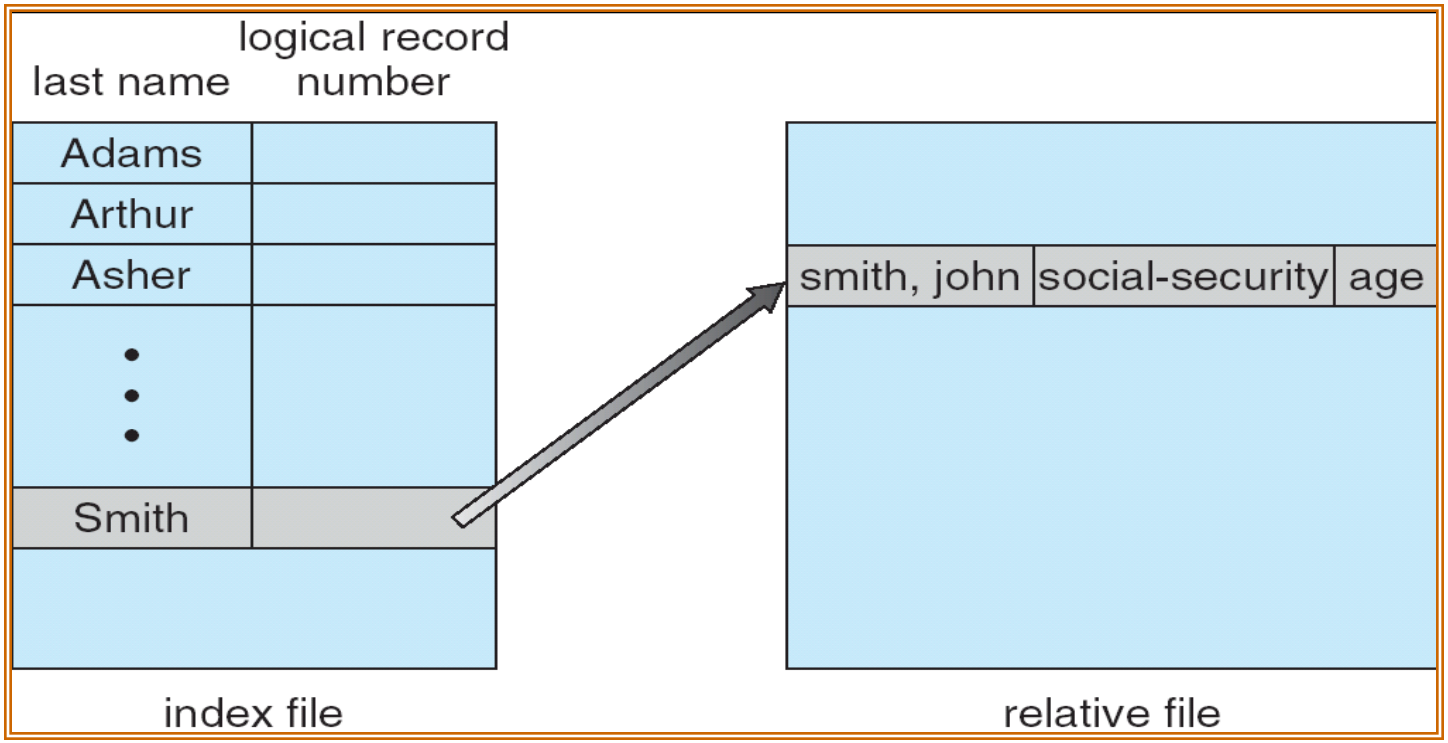
# Sequential-access File



# Simulation of Sequential Access on a Direct-access File

| sequential access | implementation for direct access |
|---|---|
| *reset* | $cp = 0;$ |
| *read next* | *read cp*;<br>$cp = cp + 1;$ |
| *write next* | *write cp*;<br>$cp = cp + 1;$ |

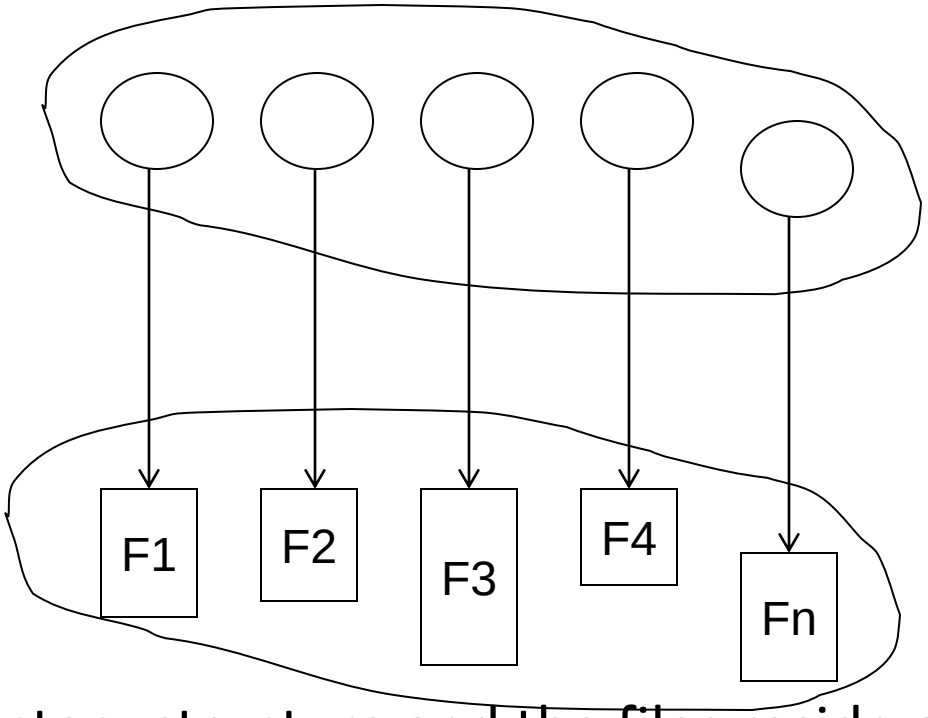
# 3.Other Access methods




Example of Index and Relative Files

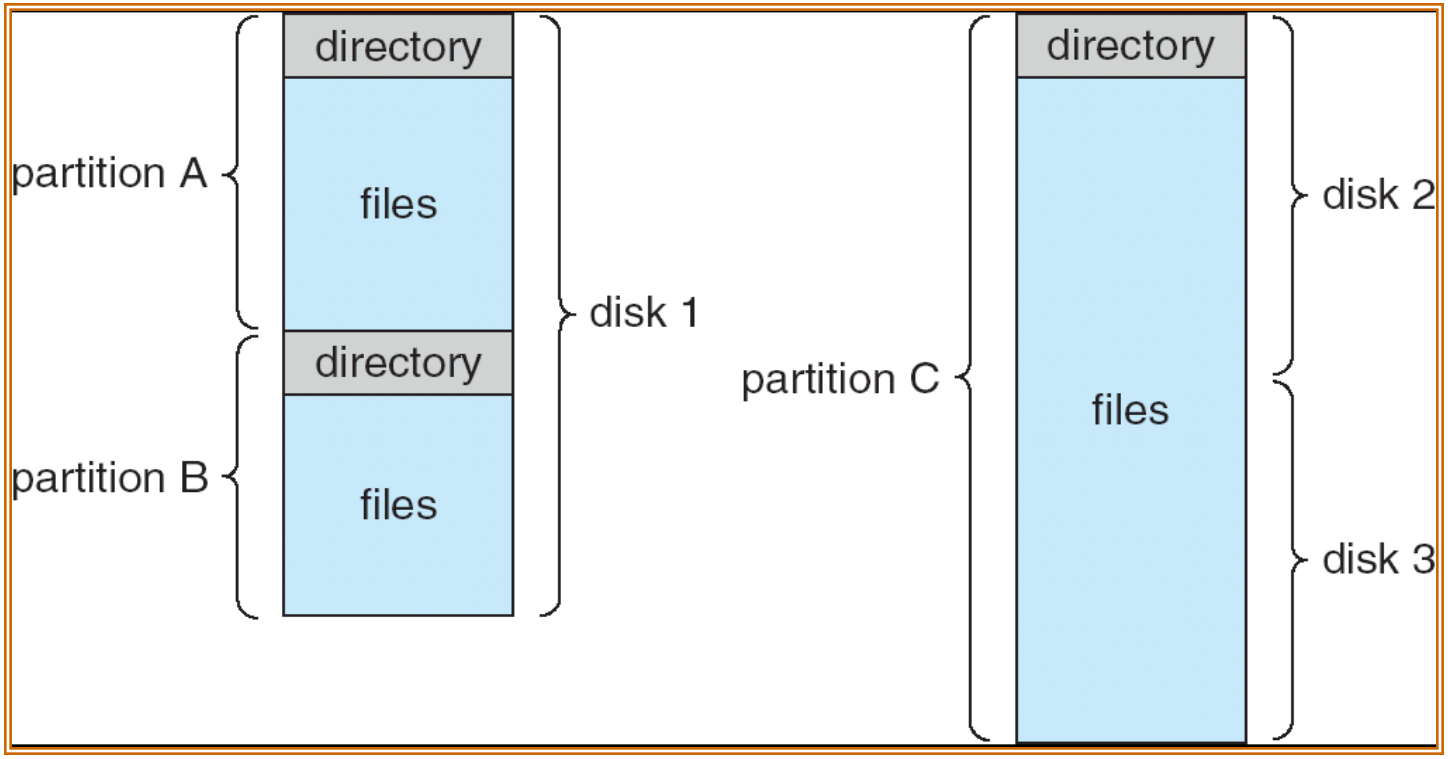
# Directory Structure

- A collection of nodes containing information about all files
- Directory
- 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



# Operations Performed on Directory

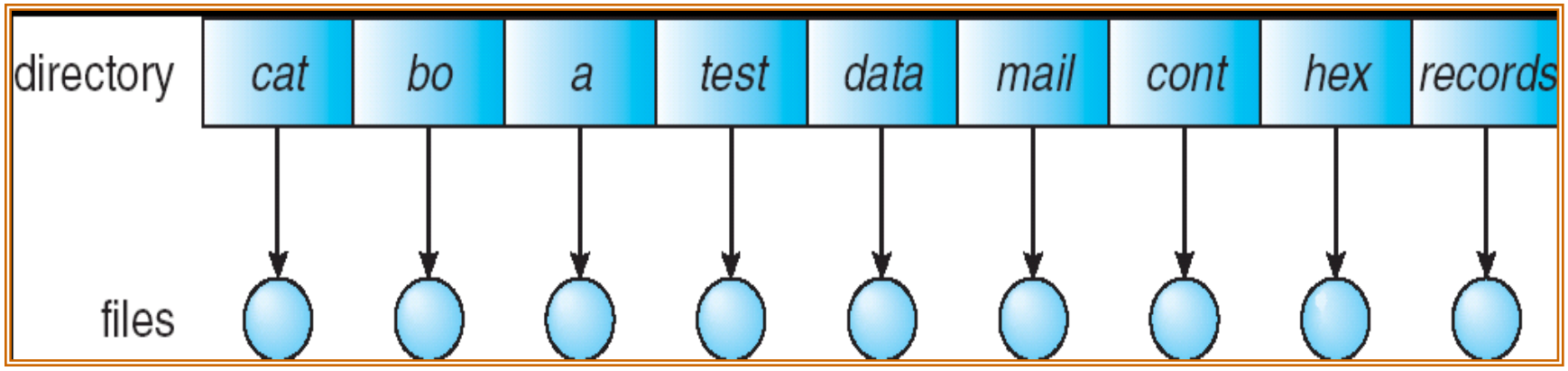
- Search for a file
- Create a file
- Delete a file
- List a directory
- Rename a file
- Traverse the file system

# 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, …)

# 1. Single-Level Directory

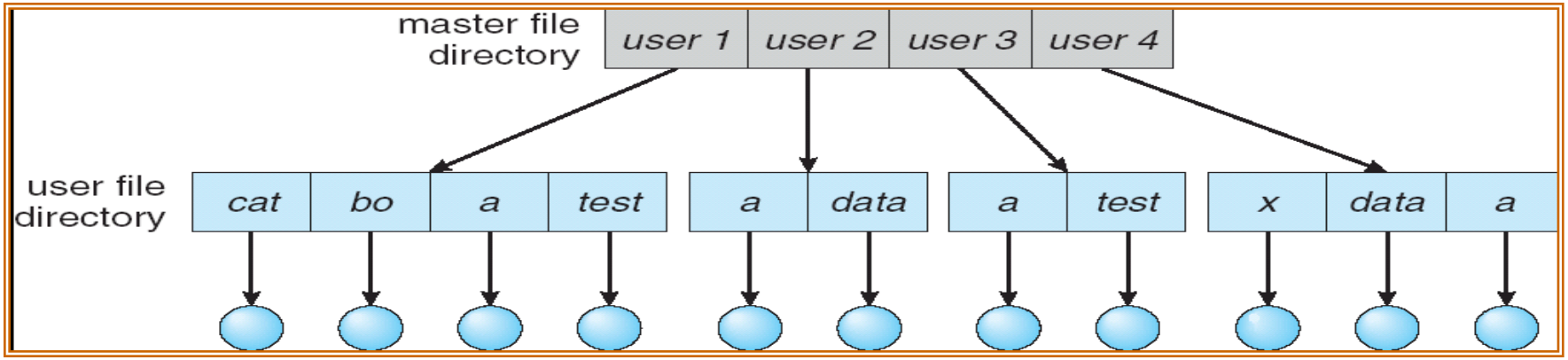
- A single directory for all users




- Drawback:-
  - Naming problem
  - Grouping problem

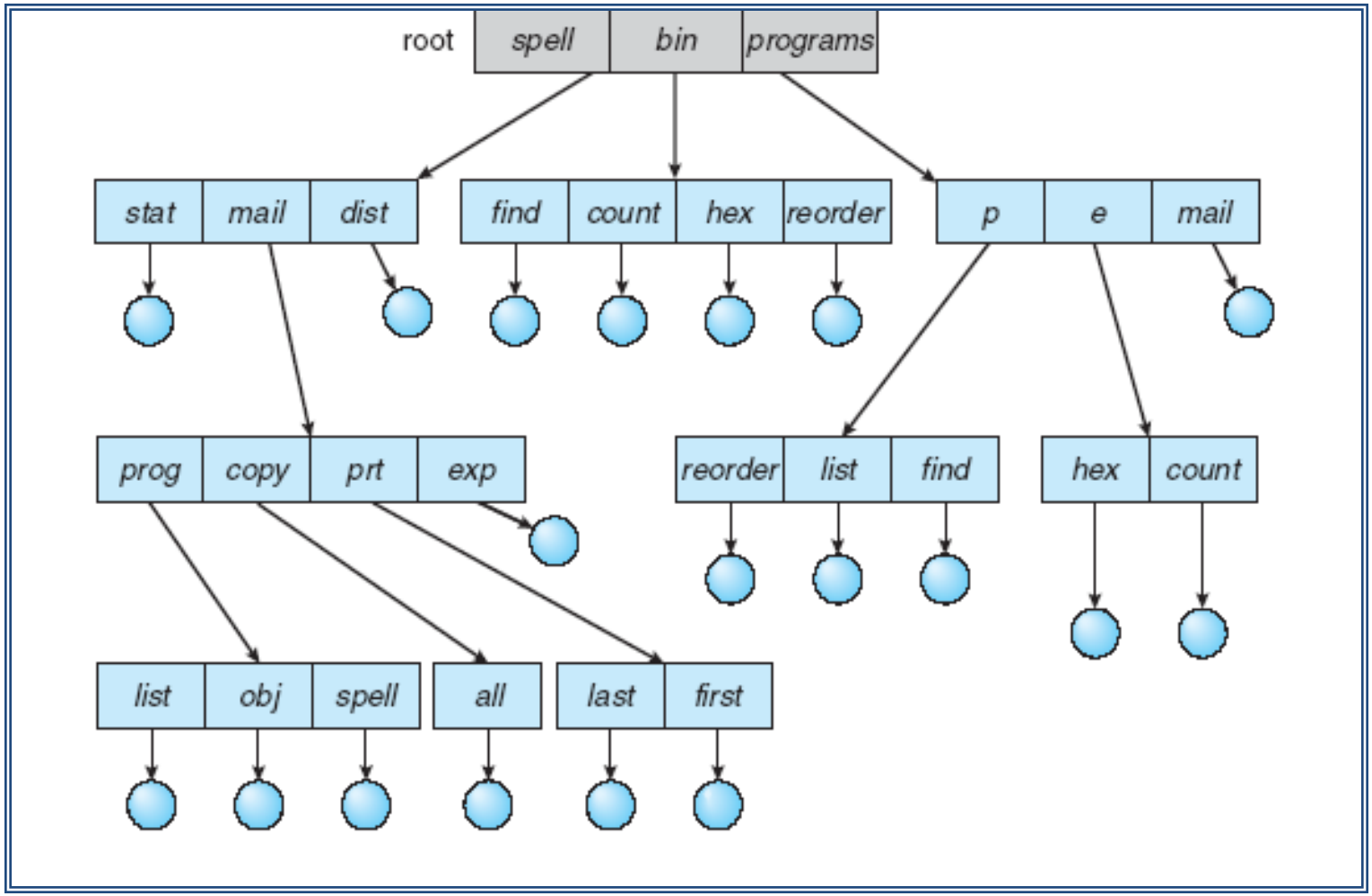
# 2. Two-Level Directory

- Separate directory for each user




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

# 3. Tree-Structured Directories



# Tree-Structured Directories (Cont)

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

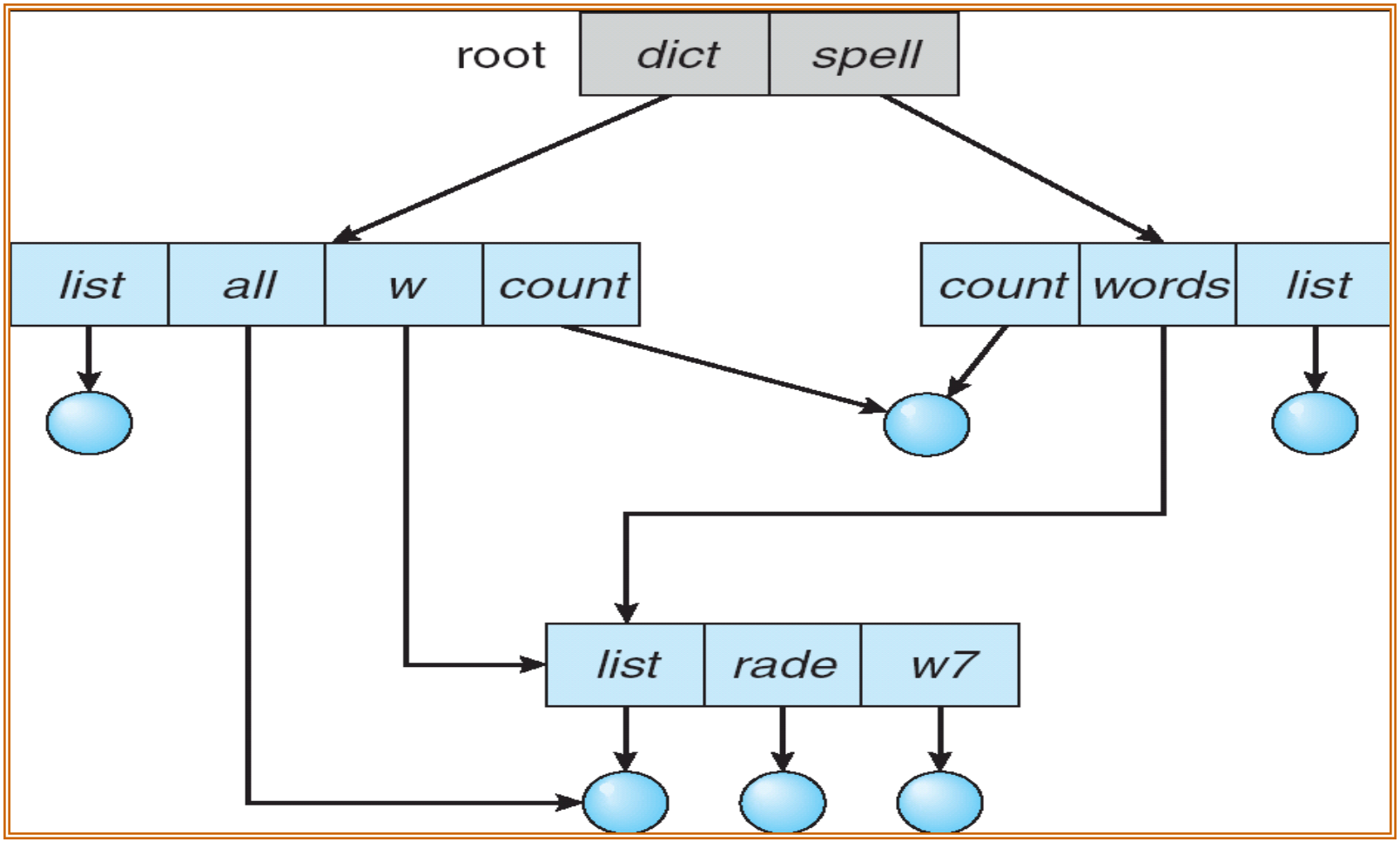
  mkdir count

Deleting "mail" ⇒ deleting the entire sub-tree rooted by "mail"

- DB:- No File sharing

# 4. Acyclic-Graph Directories

- Have shared subdirectories and files



- Two different names (aliasing)

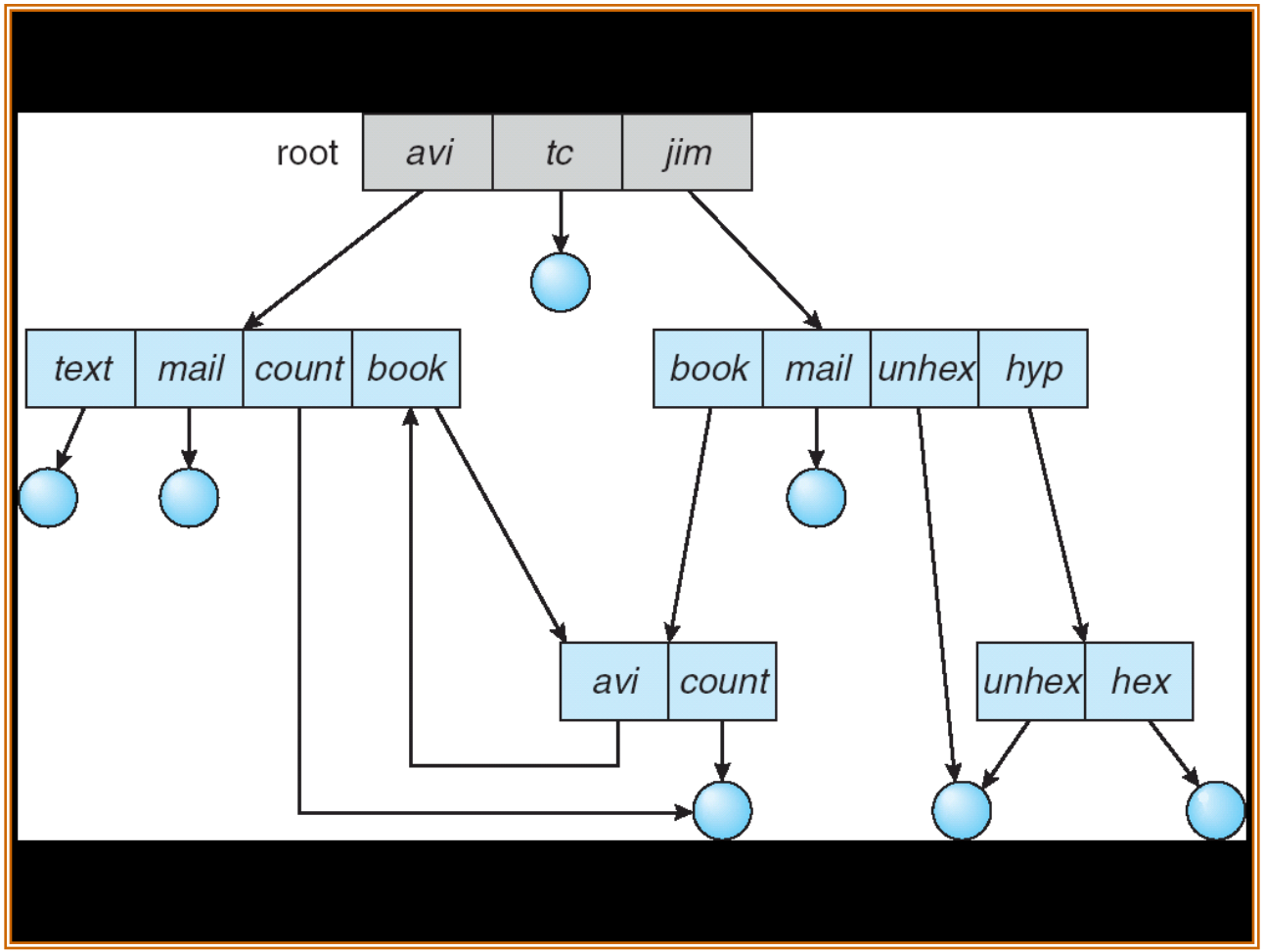
- If *dict* deletes *list* $\Rightarrow$ dangling pointer
- One possibility is to remove the file whenever anyone delete it, but this action may leave dangling pointers to the now –nonexistent file.

  Solutions:
  - Backpointers, so we can delete all pointers
    Variable size records a problem
  - Backpointers using a daisy chain organization
  - Entry-hold-count solution
- New directory entry type
  - **Link** – another name (pointer) to an existing file
  - **Resolve the link** – follow pointer to locate the file
- **Problems:-**

  **1. Deleting**

  **2. Backup storage and path**

# 5.General Graph Directory



- 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 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***
- **1. 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
  - Standard operating system file calls are translated into remote calls
- 2. **Distributed Information Systems (distributed naming services)** such as DNS, 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 Chapter process synchronization algorithms
    - Tend to be less complex due to disk I/O and network latency (for remote file systems)
  - 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

- Information must be protected from a *physical damages and improper access.*
- i.e. Reliability and protection.
- Protection mechanisms provide *control access* by limiting the type of file access that can made.
- File owner/creator should be able to control:-
  - what can be done
  - by whom

1. Types of access:-
   - **Read:-** Read from the file
   - **Write:-** Write the file
   - **Execute:-** Loads the files into memory and execute it.
   - **Append:-** Write a new information at the end of the file
   - **Delete:-** Delete the file and free its space for possible reuse
   - **List:-** List the name and attributes of the file.

- 2. Access Control:-
- Mode of access:- read, write, execute
- Three classes of users

  a) **owner:-**User Who created the file

  b) **group:-** Set of users who are sharing the file and need similar access is a group

  c) **public access/Universe:-** All other users in the system constitute the universe.
- 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.

# Windows XP Access-control List Management