Module 4.1 File Management

Outline

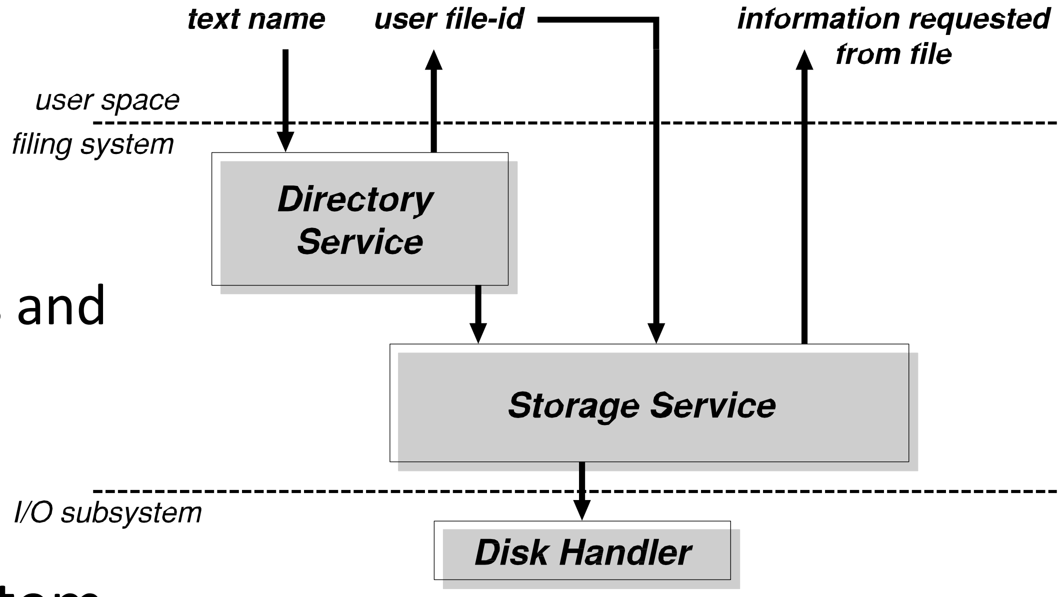
- Files
  - File systems
  - File metadata
  - File and directory operations
- Directories
  - Tree-structured
  - Acyclic-graph structured
  - File system mounting

Files

- The basic abstraction for non-volatile storage:
  - Can be a user or an OS abstraction (convenience vs flexibility)
  - Typically comprises a single contiguous logical address space
- Many different types
  - Data: numeric, character, binary (text vs binary split quite common)
  - Program: source, object, executable
  - “Documents”
- Can have varied internal structure:
  - None: a simple sequence of words or bytes
  - Simple record structures: lines, fixed length, variable length
  - Complex internal structure: formatted document, relocatable object file

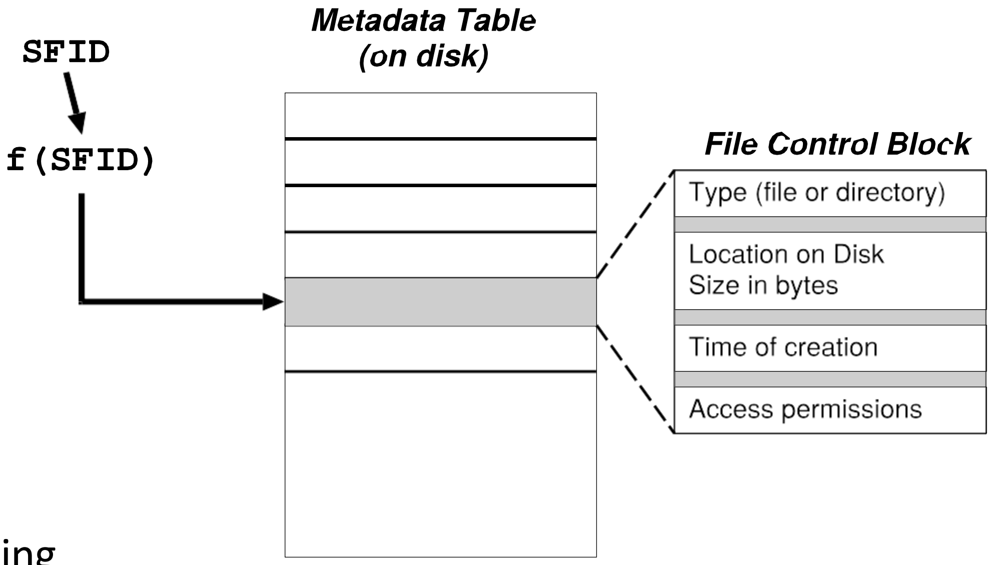
File system

- Consider only simple file systems
  - **Directory service** maps names to file identifiers and metadata, handles access and existence control
  - **Storage service** stores data on disk, including storing directories
- Each partition formatted with a filesystem
  - Logically, a **directory** and some **files**
  - Directory maps human name (*hello.java*) to **System File ID** (typically an integer)
  - Different filesystems implement using different structures



Name	SFID
hello.java	12353
Makefile	23812
README	9742

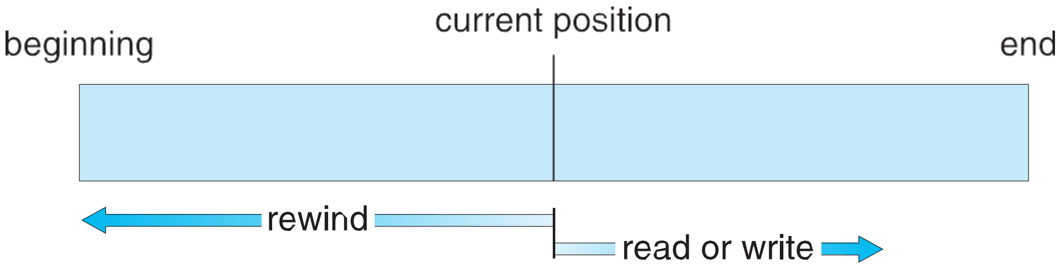
File metadata



- The mapping from SFID to File Control Block (FCB) is filesystem specific
- Files typically have a number of other attributes or metadata stored in directory
  - **Type** – file or directory
  - **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
- OS must also track open files in an **open-file table** containing
  - **File pointer** or **cursor**: last read/written location per process with the file open
  - **File-open count**: how often is each file open, so as to remove it from open-file table when last process closes it
  - **On-disk location**: a cache of data access information
  - **Access rights**: per-process access mode information

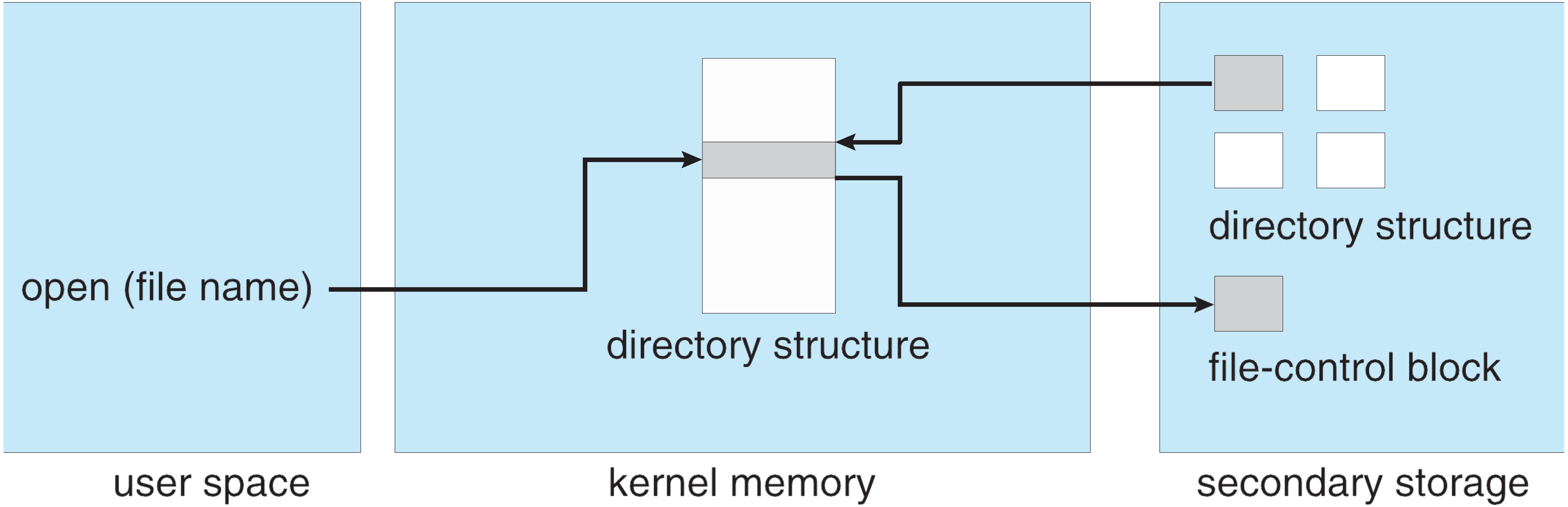
File and directory operations

- A file as an **abstract data type** (**ADT**) over some (possibly structured) bytes
- **Directory operations** to manage lifetime of a file
  - **Create** allocates blocks to back the file
  - **Open**/**Close** handle to the file, typically including OS maintained current position (**cursor**)
  - **Delete** returns allocated blocks to the free list
  - **Stat** retrieves file status including existence reads and returns file metadata
- **File operations** to interact with file
  - **Write** provided data at cursor location
  - **Read** data at cursor location into provided memory
  - **Truncate** clips length of file to end at current cursor value
- Access pattern:
  - **Random access** permits **seek** to move cursor without reading or writing
  - **Sequential access** permits only **rewind** to move cursor back to beginning

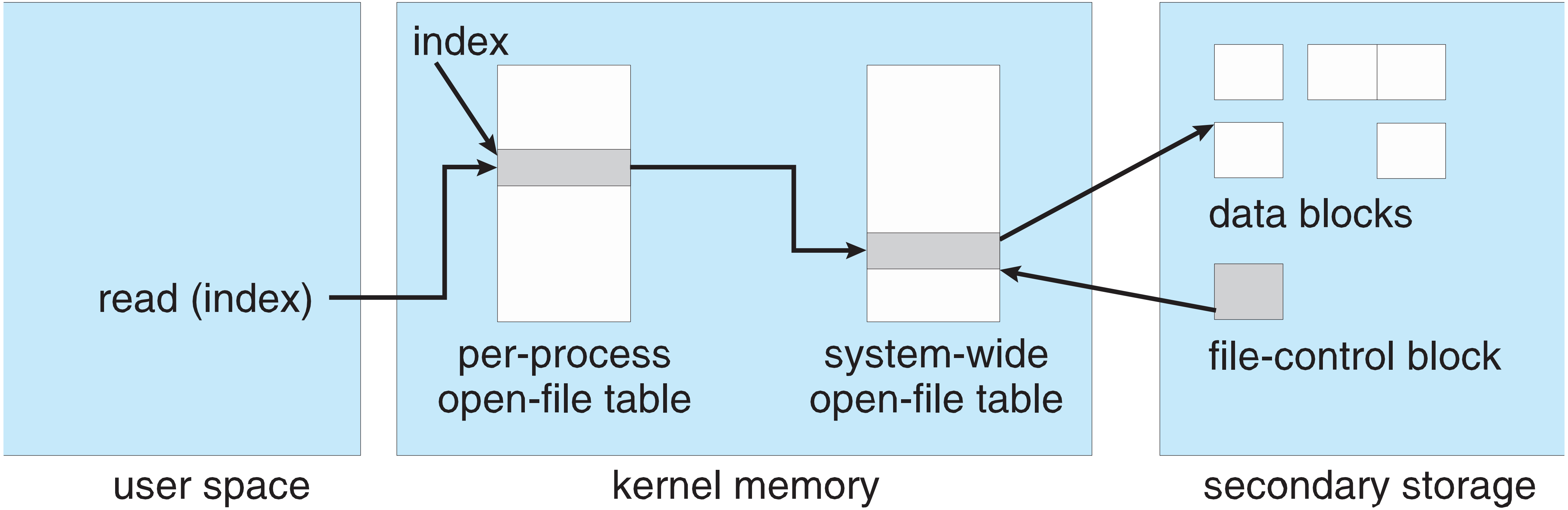


Opening a file

- In-memory directory structure previously read from disk resolves file name to a file control block



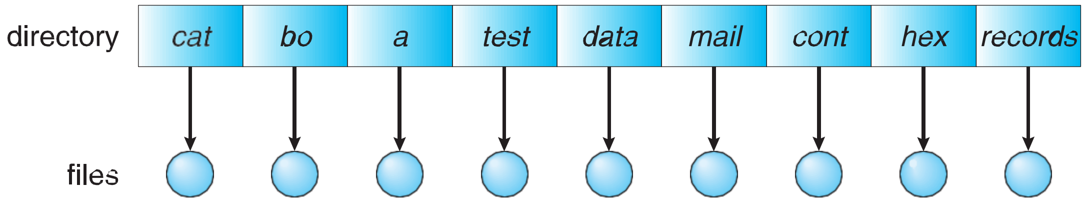
Reading a file

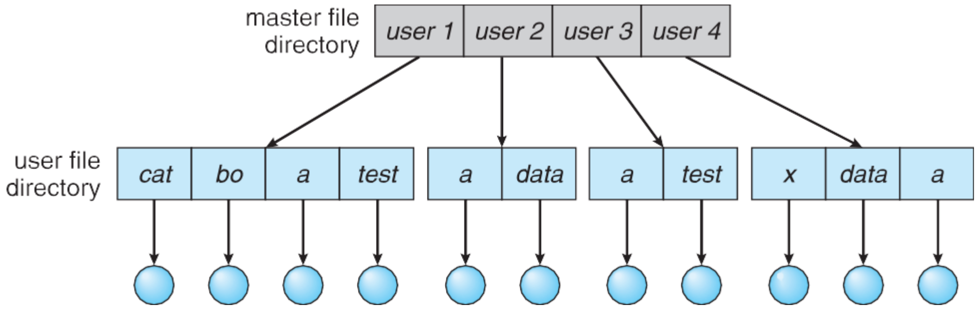


- Using per-process open-file table, index (file handle or file descriptor) resolves to system-wide open-file table containing file-control block which resolves to actual data blocks on disk

Directories

- Implementations must provide
  - **Grouping**, to enable related files to be kept together
  - **Naming**, for user convenience so different files can have the same name and one file can have many names
  - **Efficiency**, to find files quickly
- **Single-level directory** is simplest
  - Naming and grouping problems though
- **Two-level directory** is next (FAT)
  - Same names for different users via paths
  - Efficient searching but no grouping





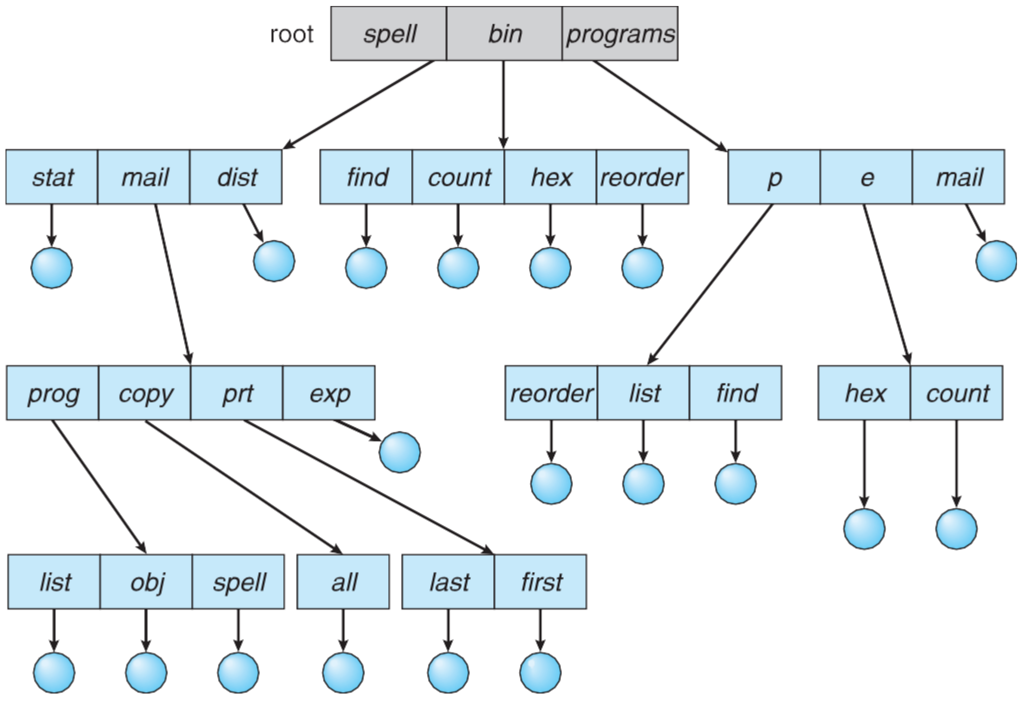
Tree-structured directories

- Provide naming convenience, efficient search, and grouping
- Introduce notion of **current working directory** (**CWD**)

  cd /spell/mail/prog

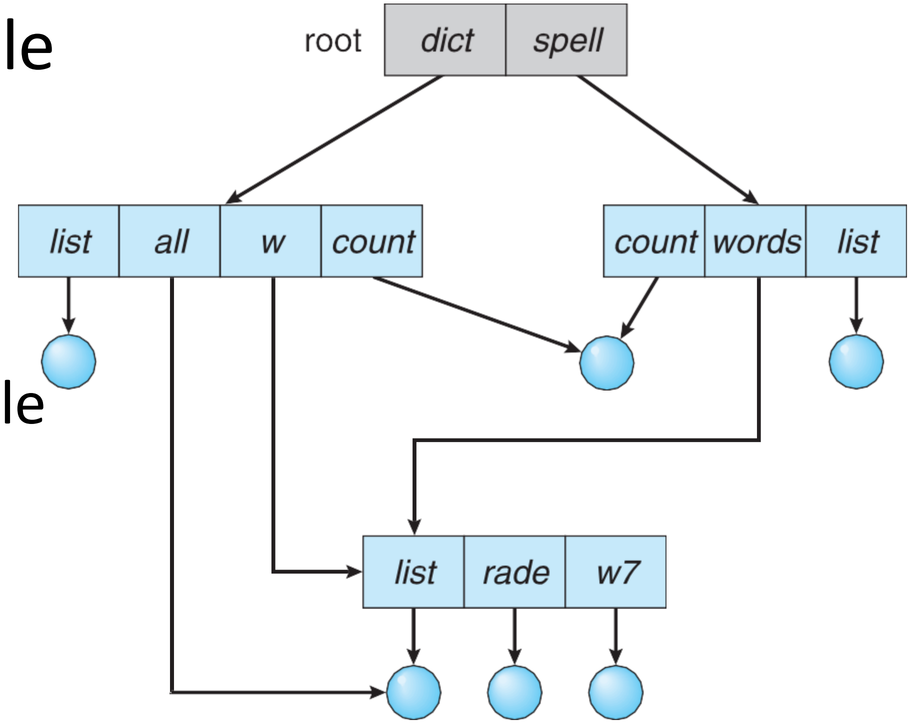
  type list

- Gives rise to **absolute** or **relative** path names
  - Name is resolved with respect to the CWD
- Other operations also typically carried out relative to CWD



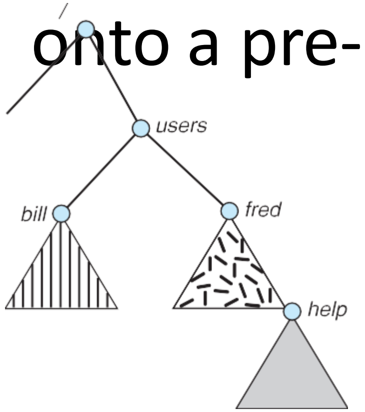
Acyclic-graph structured directories

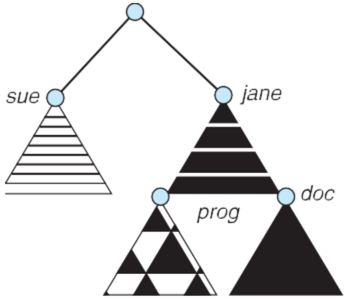
- Generalise to a DAG so can share subdirectories and files
  - Allows files to have two different absolute names (**aliasing**)
- Need to know when to actually delete a file
  - Use back-references or reference counting
  - Compare soft- and hard-links in Unix
- Need to know how to account storage
  - Which user "owns" the storage backing the file
  - For deletion and generally for permissions
- Need to avoid creating cycles
  - Forbid links to subdirectories



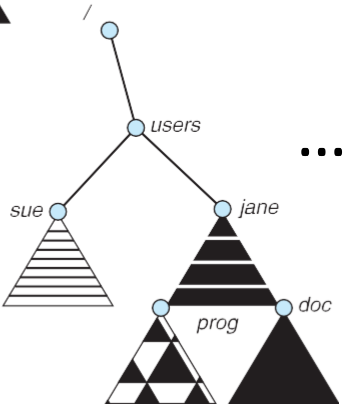
File-system mounting

- Filesystems must be **mounted** at a **mount-point** before access, e.g., onto a pre-existing file-system...





...an unmounted filesystem in another partition



...is mounted, overlaying the *users* subdirectory