

10. Storage & File Management

9th ed: Ch. (10,) 11, 12

10th ed: Ch. (11,) 13, 14, 15

Objectives

- To understand the nature of mass storage
- To be aware of the challenges of (disk) storage management
- To understand concepts of files, directories and directory namespaces, directory structures, hard- and soft-links
- To know of basic file operations and access control mechanisms
- To be aware of the relationship between paging and block storage in the buffer cache

Outline

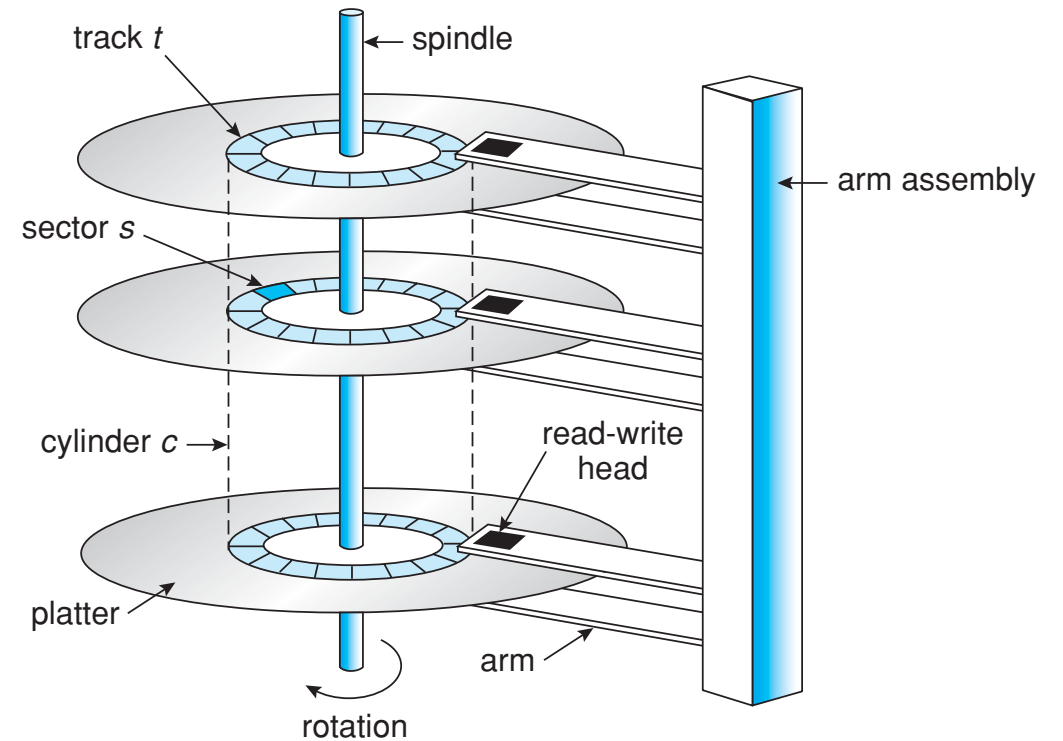
- Mass storage
- Disk scheduling
- Disk management
- Files
- Directories
- Other issues

Outline

- Mass storage
 - Hard disks
 - Solid state disks
- Disk scheduling
- Disk management
- Files
- Directories
- Other issues

Mass storage: Hard disks (HDs)

- Stack of platters
 - Historically 0.85" to 14"
 - Commonly 3.5", 2.5", 1.8"
 - Capacity continually increases but perhaps 30GB – 3TB
- Performance
 - Transfer Rate – theoretical – 6 Gb/sec
 - Effective Transfer Rate – real – 1Gb/sec
 - Seek time 3–12ms with around 9ms common
 - Rotation typically 7200 or 15,000 RPM



Hard disk performance

- **Average latency [secs]**
 $= \frac{1}{2} \text{ Latency} = \frac{1}{2} \times \frac{1}{60} / (\text{rotations/minute}) = 30/\text{RPM}$
- **Access latency [secs]** = Average seek time + average latency
- **Average I/O time [secs]**
 $= \text{Access latency} + \left(\frac{\text{amount to transfer}}{\text{transfer rate}} \right) + \text{controller overhead}$
- E.g., 4kB block, 7200 RPM, 5ms average seek time, 1Gb/sec transfer rate, 0.1ms controller overhead
 - Average latency = $30/7200 = 4.17\text{ms}$
 - Transfer time = $(4096 \text{ B} \times 8^{b/B}) / 1024^3 \text{ b/s} = 0.031\text{ms}$
 - Average I/O time = $5\text{ms} + 4.17\text{ms} + 0.031\text{ms} + 0.1\text{ms} = 9.301\text{ms}$

Mass storage: Solid state disks (SSDs)

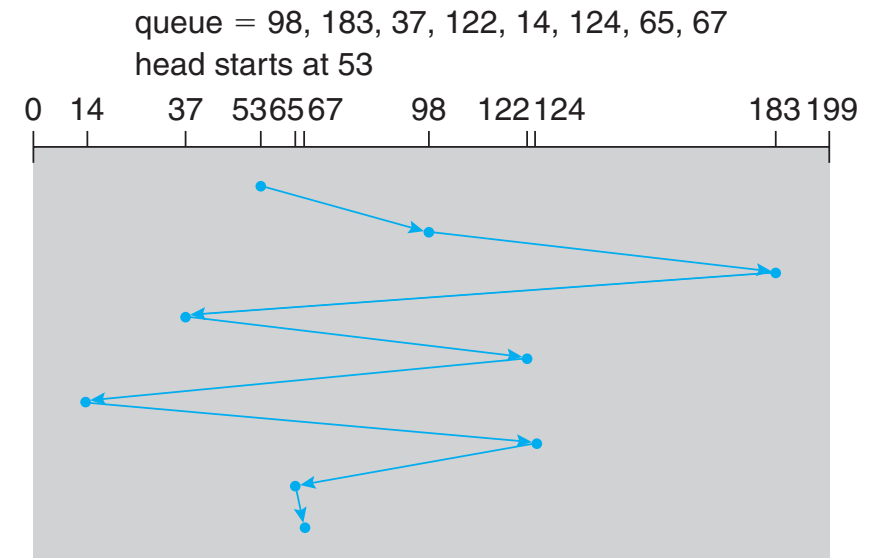
- Non-volatile memory used like a hard drive; many variations
- Pros
 - Can be more reliable than HDDs
 - No moving parts, so no seek time or rotational latency
 - Much faster
- Cons
 - Reads/writes wear out cells leading to unreliability and potentially shorter
 - More expensive per MB
 - Lower capacity

Outline

- Mass storage
- **Disk scheduling**
 - First-Come First-Served
 - Shortest Seek Time First
 - SCAN, C-SCAN
- Disk management
- Files
- Directories
- Other issues

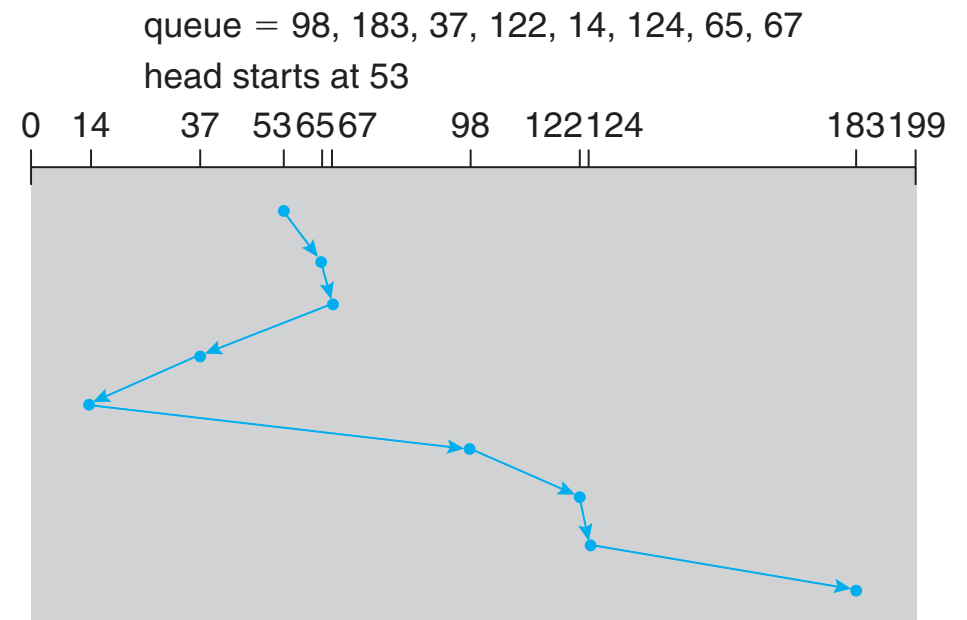
Disk scheduling

- The disk controller receives a sequence of read/write requests from the OS that it must schedule
 - How best to order reads and writes to achieve policy aim?
 - Analogous to CPU scheduling but with very different mechanisms, constraints, and policy aims
 - Many algorithms exist
- Simplest: First-come First-served (FCFS)
 - Intrinsically fair but inefficient
 - E.g., requests for blocks on cylinders are 98, 183, 37, 122, 14, 124, 65, 67



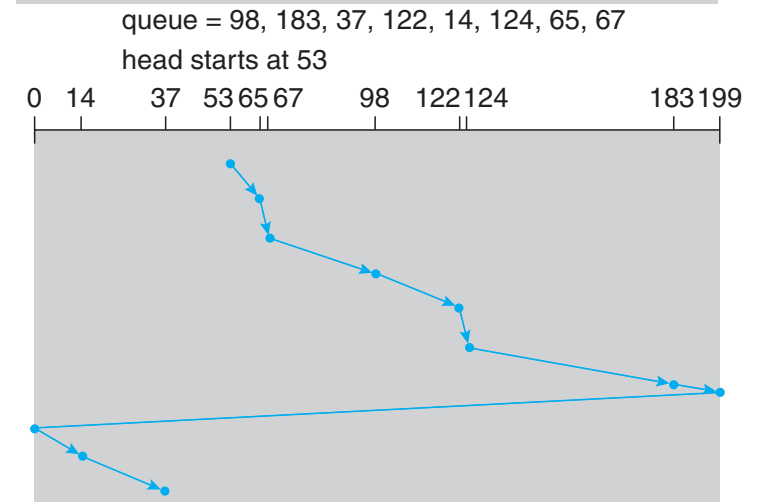
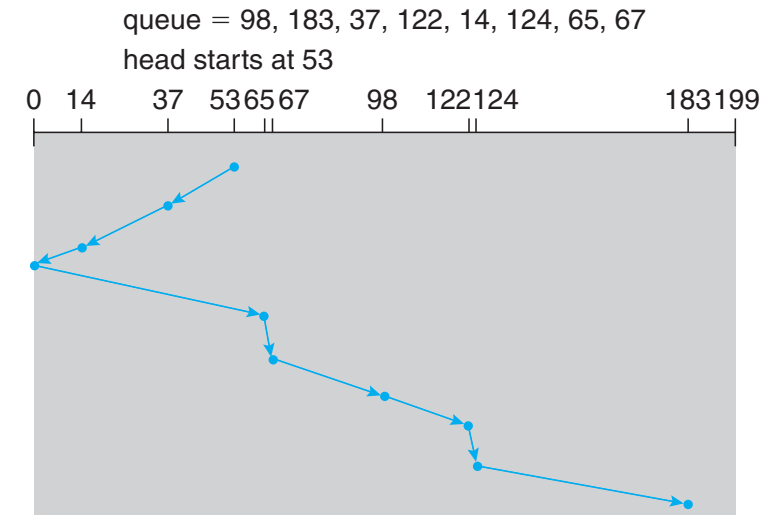
Shortest Seek-Time First (SSTF)

- Service requests based on distance to current head position
 - Next request in queue is that with the shortest seek time
- For this example, involves movement of just 236 cylinders
 - $\frac{1}{3}$ of that required by FCFS
- Somewhat analogous to SJF
 - A big improvement but allows starvation
 - Not optimal: from 53 move to 37 then 14 and then 65 etc – gives movement of 208 cylinders



SCAN and C-SCAN

- **SCAN** or **elevator** algorithm
 - Start at one end of the disk and move to the other end
 - Service everything on the way
- Consider density of requests when changing direction
 - Have just serviced (almost) everything in that vicinity
 - Those furthest away have waited longest so...
- **Circular-SCAN**
 - Return back to the start when reaching the end
 - Cylinders treated as a circular list, wrapping when reaching the end



Outline

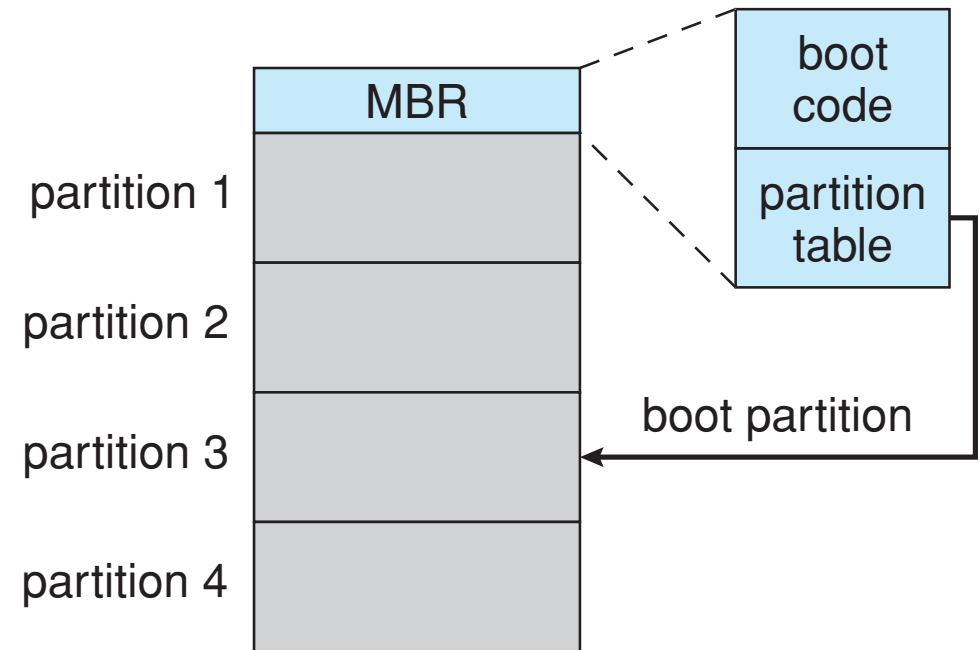
- Mass storage
- Disk scheduling
- **Disk management**
 - Booting from disk
- Files
- Directories
- Other issues

Disk management

- **Low-level or physical formatting**
 - Divides a disk into sectors that the disk controller can read and write
 - Each sector can hold header information, plus data, plus error correction code (ECC)
 - Usually 512 bytes of data but can be selectable
- **Logical formatting** to make a file system required before disk can hold files
 - OS needs to record its own data structures on the disk so it can find files
 - Partition the disk into one or more groups of cylinders, each treated as a logical disk
 - To increase efficiency most file systems group blocks into clusters
- **Disk I/O done in blocks**
- **File I/O done in clusters**
 - Some applications, e.g., databases, will prefer “raw” block access

Booting from disk

- OS needs to know where to start looking
 - BIOS (or similar) is “firm-coded” to e.g., read first block of first disk
- First block contains bootloader program, which is executed
- Bootloader knows enough to start reading in the right blocks to read the filesystem starting with the **partition table**
 - Sometimes need to **chain-load** to get enough code to parse more complex filesystems
- Allows for handling of bad blocks
 - E.g., by **sector sparing** where spare good blocks logically substitute for bad ones



Outline

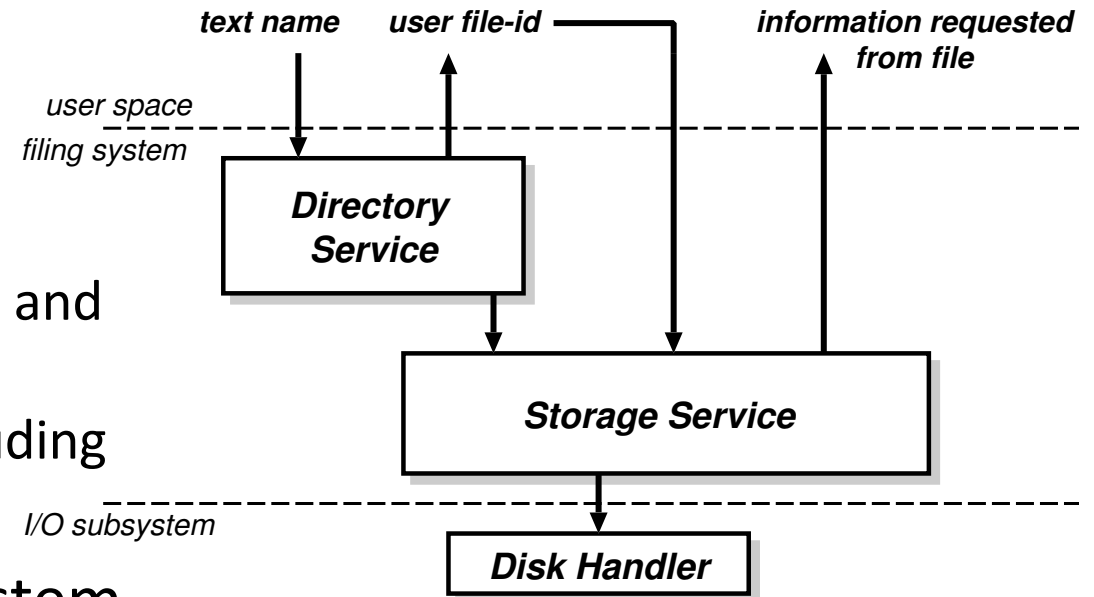
- Mass storage
- Disk scheduling
- Disk management
- **Files**
 - File systems
 - File metadata
 - File and directory operations
- Directories
- Other issues

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

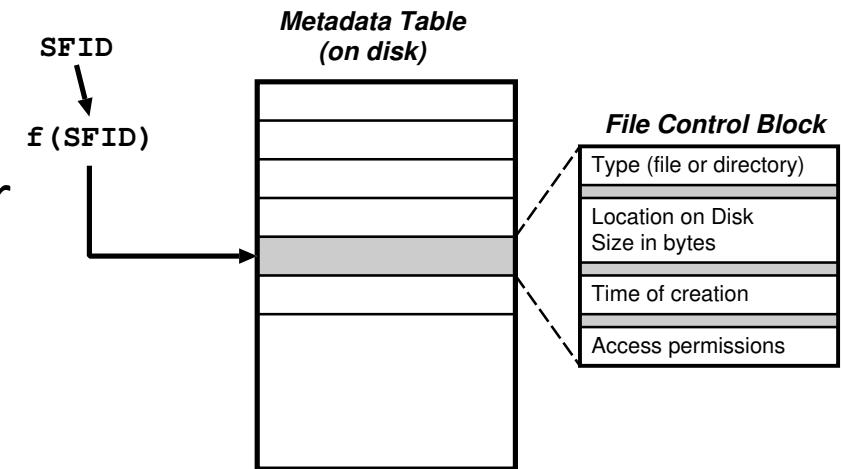
- Consider only simple file systems
 - **Directory service** maps names to file identifiers and metadata, handles access and existence control
 - **Storage service** stores data on disk, including storing directories
- Each partition formatted with a filesystem
 - Logically, a **directory** and some **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

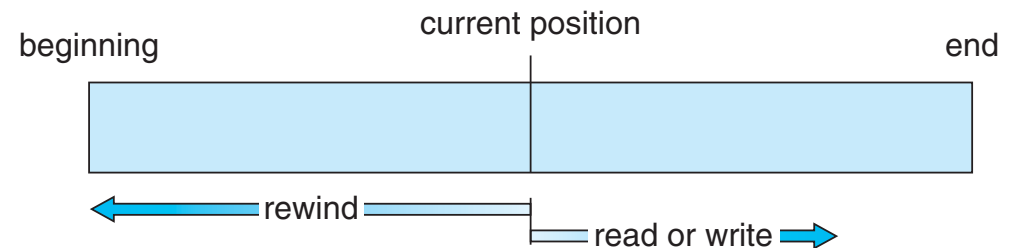
File metadata

- 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



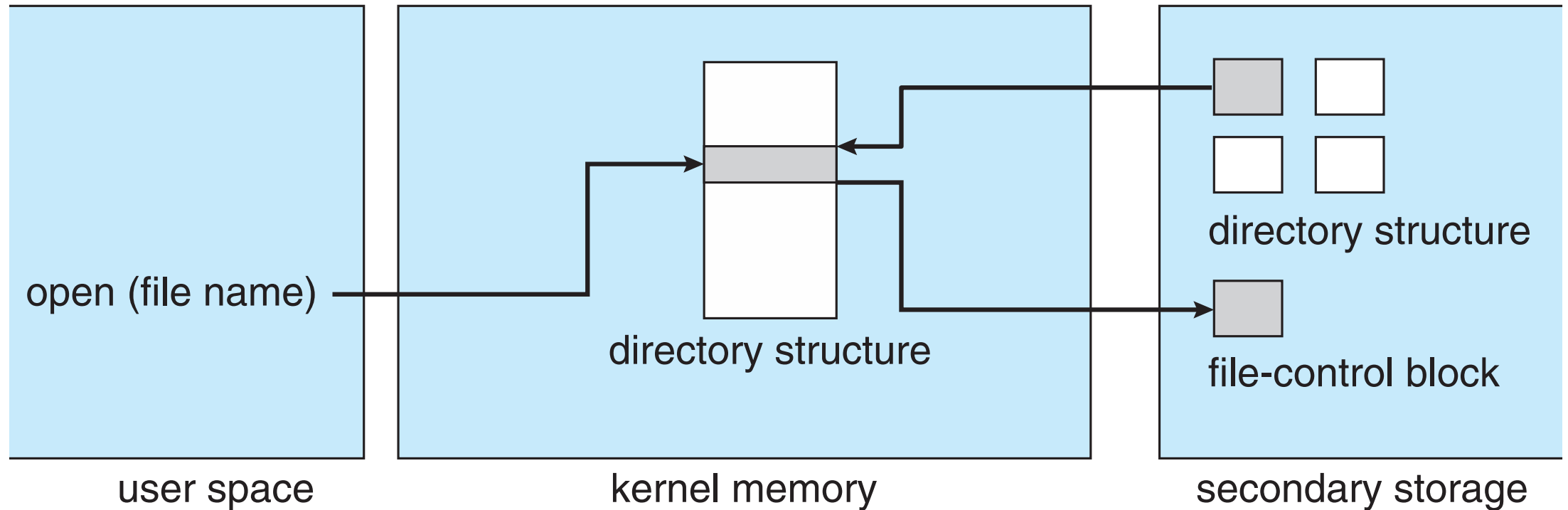
File and directory operations

- 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

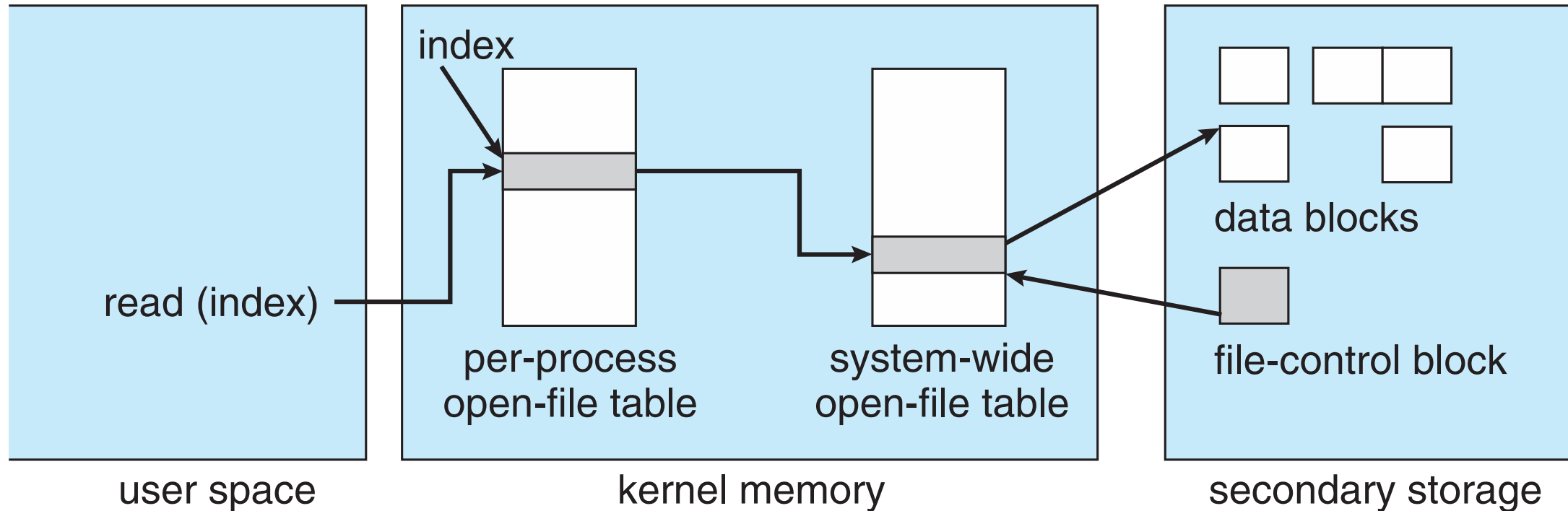


Opening a file

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



Reading a file



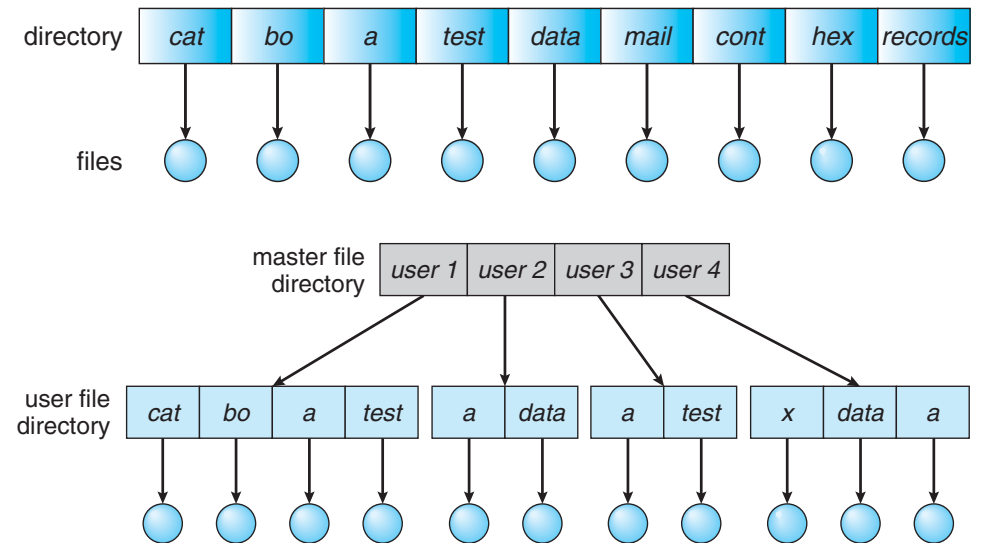
- Using per-process open-file table, index (file handle or file descriptor) resolves to system-wide open-file table containing file-control block which resolves to actual data blocks on disk

Outline

- Mass storage
- Disk scheduling
- Disk management
- Files
- **Directories**
 - Tree-structured
 - Acyclic-graph structured
 - File system mounting
- Other issues

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



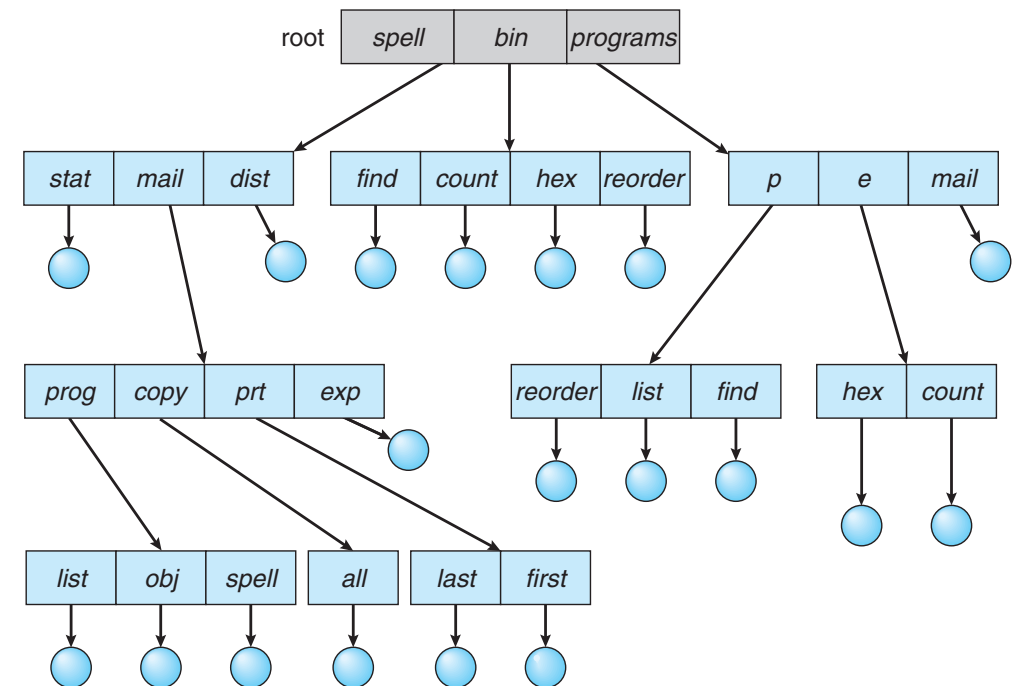
Tree-structured directories

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

```
cd /spell/mail/prog
```

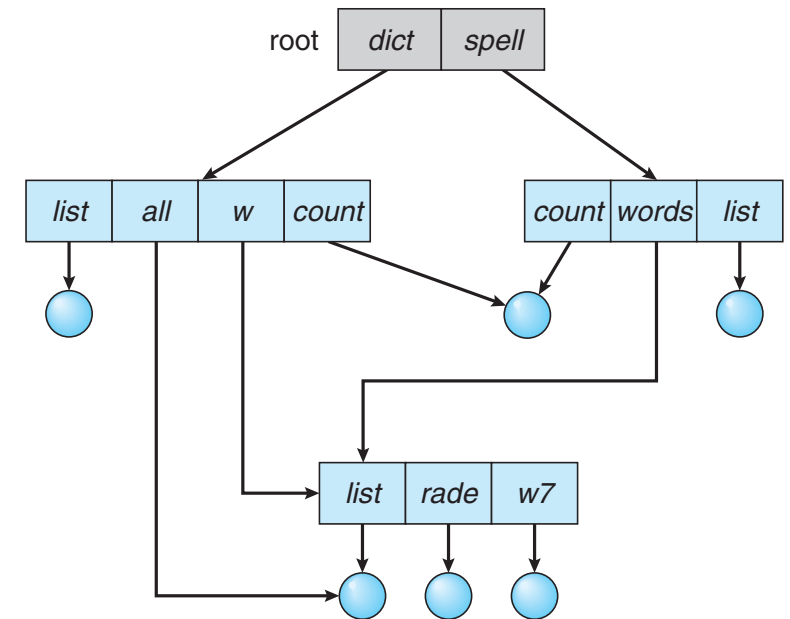
```
type list
```

- 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

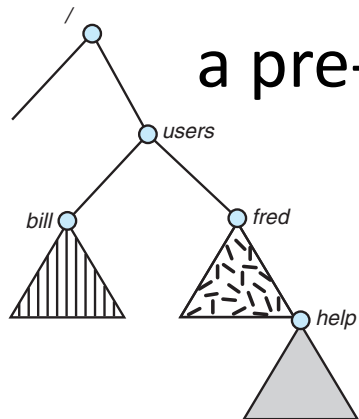
- Generalise to a DAG so can share subdirectories and files
 - Allows files to have two different absolute names (**aliasing**)
- Need to know when to actually delete a file
 - Use back-references or reference counting
 - Compare soft- and hard-links in Unix
- Need to know how to account storage
 - Which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



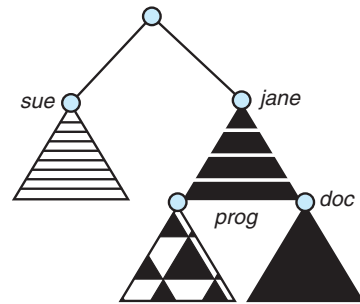
File-system mounting

- Filesystems must be **mounted** at a **mount-point** before access

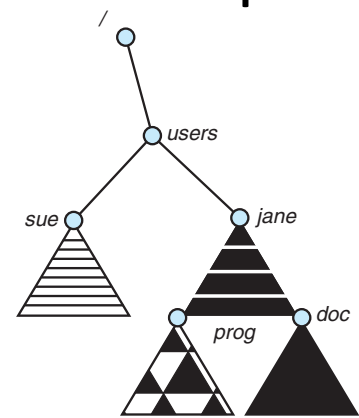
- E.g.,  a pre-existing file-system



...an unmounted filesystem in another partition



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



Outline

- Mass storage
- Disk scheduling
- Disk management
- Files
- Directories
- **Other issues**
 - Consistency
 - Efficiency
 - Buffer cache

Consistency issues

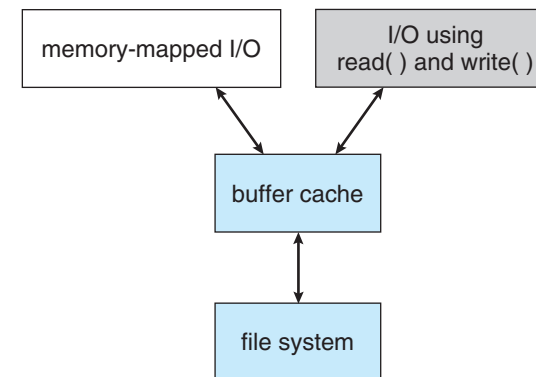
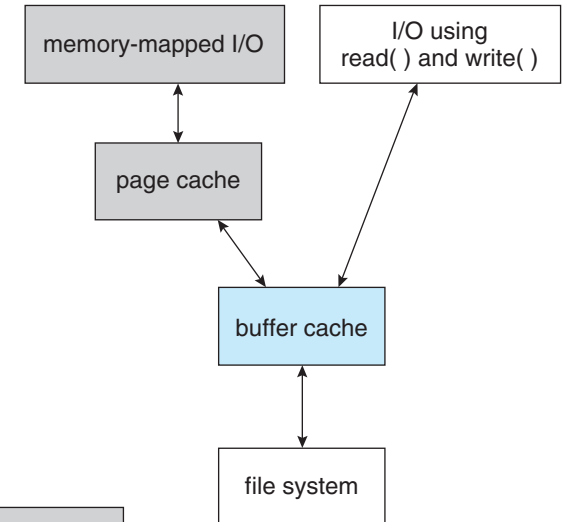
- Arise without multiple threads!
- E.g., Deleting a file uses the *unlink* system call
 - Invoked from the shell as *rm <filename>*
- Implementation must
 - Check if user has sufficient permissions on the file (write access)
 - Check if user has sufficient permissions on the directory (write access)
 - If ok, remove entry from directory
 - Decrement reference count on inode
 - If reference count is now zero, free data blocks and inode
- If the system crashes, must check the entire filesystem (*fsck*)
 - Check if any block is unreferenced, and mark free
 - Check if any block double referenced, and update reference counts

Efficiency and performance

- Efficiency depends on, e.g,
 - Disk allocation and directory algorithms
 - Similar challenges to memory of allocation, fragmentation, compaction
 - Types of metadata in directory entries
 - E.g., file creation time vs last written time vs last accessed time
 - Pre-allocation or as-needed allocation of metadata structures
 - Fixed-size or varying-size data structures
- Performance measures include
 - Keep data and metadata close together
 - Create a buffer cache, a separate part of memory for often used blocks
 - Synchronous writes sometimes requested by apps or needed by OS
 - Require no buffering / caching – writes must hit the disk before acknowledgement
 - Asynchronous writes more common, can be buffered, are faster

Buffer caches

- Not unified
 - **Page cache** caches pages not disk blocks, using virtual memory techniques and addresses
 - Memory-mapped I/O uses a page cache while routine I/O through the file system uses the **buffer (disk) cache**
- Unified
 - A single **buffer cache** uses a single page cache for both memory-mapped I/O and normal disk I/O



Summary

- Mass storage
 - Hard disks
 - Solid state disks
- Disk scheduling
 - First-Come First-Served
 - Shortest Seek Time First
 - SCAN, C-SCAN
- Disk management
 - Booting from disk
- Files
 - File systems
 - File metadata
 - File and directory operations
- Directories
 - Tree-structured
 - Acyclic-graph structured
 - File system mounting
- Other issues
 - Consistency
 - Efficiency
 - Buffer cache