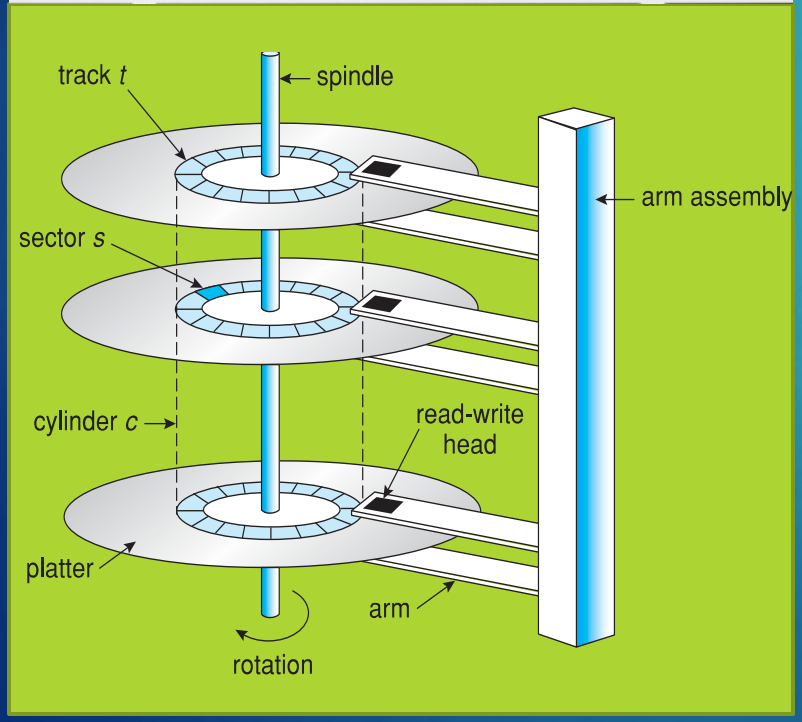
V. File Systems

SGG9: chapters 11.1-11.7, 12.1-12.6
SGG10: chapters 13, 14.1-14.6 and 15.1-15.4

- Files, directories, sharing
- Partitions, allocations, free space

TDIU11: Operating Systems

Ahmed Rezine, Linköping University



File Concept

- Primary memory is volatile: need secondary storage for long-term storage
- A *disk* is a linear sequence of numbered blocks
  - With 2 operations: write block *b*, read block *b*
    - **Physical / logical blocks**
  - Low level of abstraction,
- Portability across different storage devices
- Solution: OS provides the ***file*** abstraction
  - Smallest piece of secondary storage known to the user
  - Attributes (Name, id, size, …)
  - Organized in a *directory* of files
  - API (operations on files and directories)

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
- Many variations, including extended file attributes such as file checksum

Basic File Operations

- File is an **abstract data type.** System calls for:
  - **Create:** space must be found, and an entry created in directory
  - **Write –** at **write pointer** location
  - **Read –** at **read pointer** location
  - **Reposition within file – file seek.**
  - **Delete:** release all file space and delete directory entry.

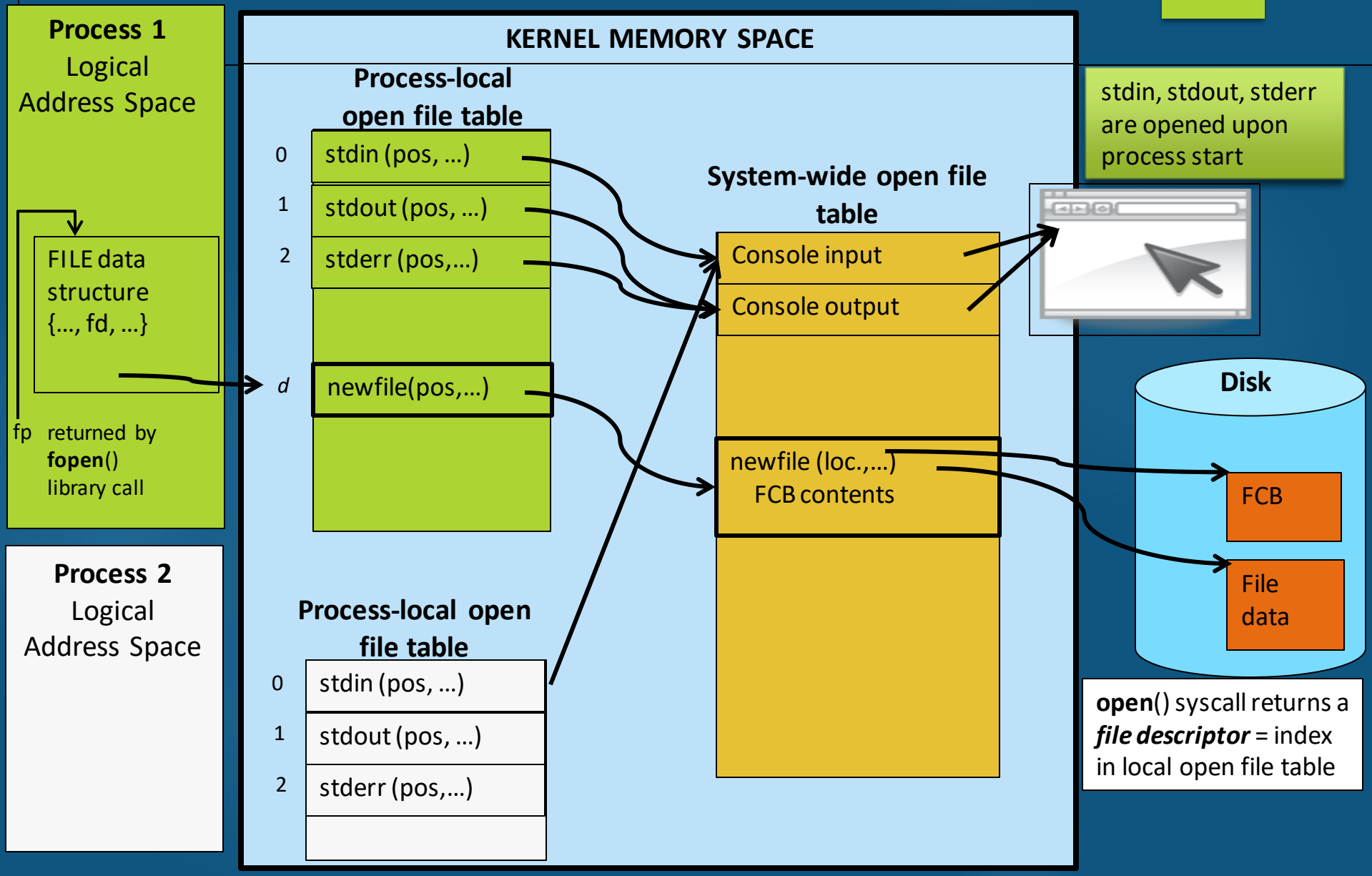
- Instead of searching each time, keep information in a table
  - ***Open(F_i)*** – search the directory structure on disk for entry ***F_i***, and move the content of entry to memory
  - ***Close (F_i)*** – move the content of entry ***F_i*** in memory to directory structure on disk

Open Files

Several pieces of data are needed to manage open files:

- **Open-file table** tracks open files (Both system wide and per process)
- 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

File descriptors and open file tables

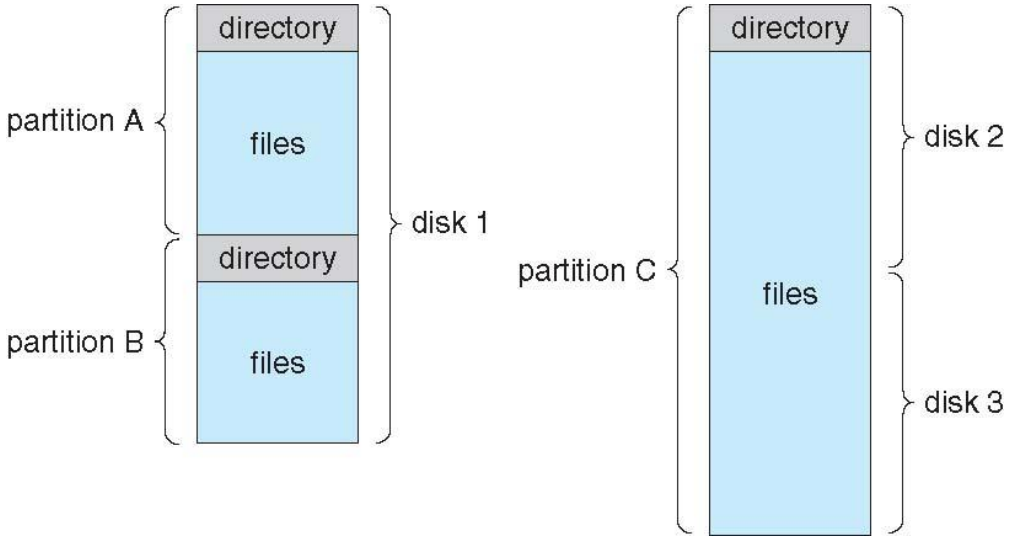


Open File Locking

- Provided by some operating systems and file systems
  - Like reader-writer locks
  - **Shared lock** like reader lock – several processes can acquire concurrently
  - **Exclusive lock** like writer lock
- Mediates access to a file
- Mandatory or advisory:
  - **Mandatory** – access is denied depending on locks held and requested (usually adopted by windows)
  - **Advisory** – processes can find status of locks and decide what to do (usually adopted by Unix)

Disk Structure

- Disk can be subdivided into partitions.
- Disk or partition can be used raw – without a file system, or formatted with a file system
- Entity containing file system known as a volume
- Each volume containing file system also tracks that file system's info in device directory or volume table of contents: names, location, size and type… for all files

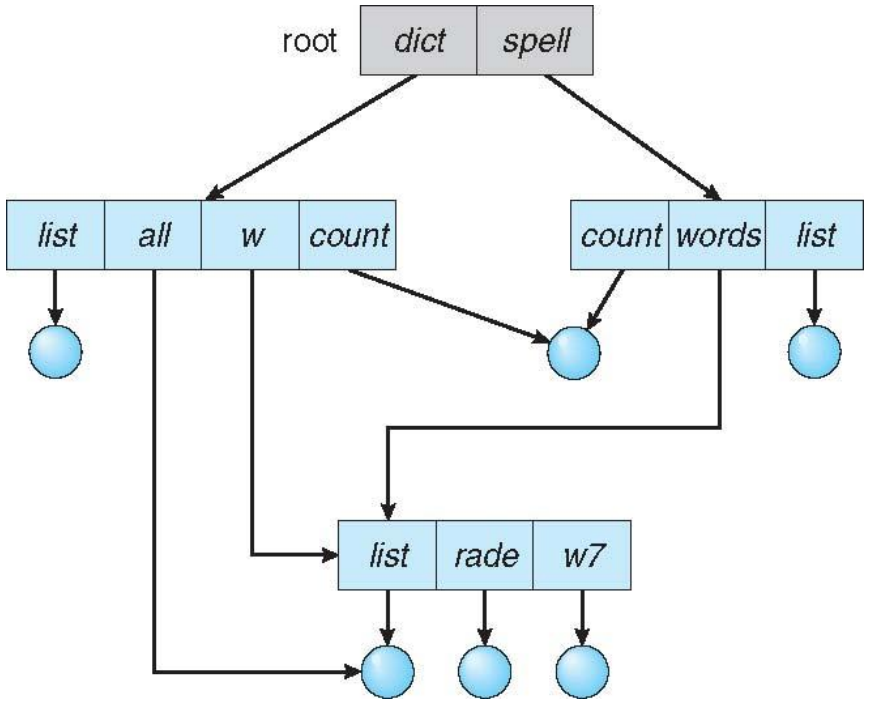


Operations Performed on Directory

- Directory could be viewed as a table from file names to their entries.
  - Search for a file, or for all files matching a pattern
  - Create and add a file
  - Delete a file
  - List all files, and their entries, in a directory
  - Rename a file
  - Traverse the file system

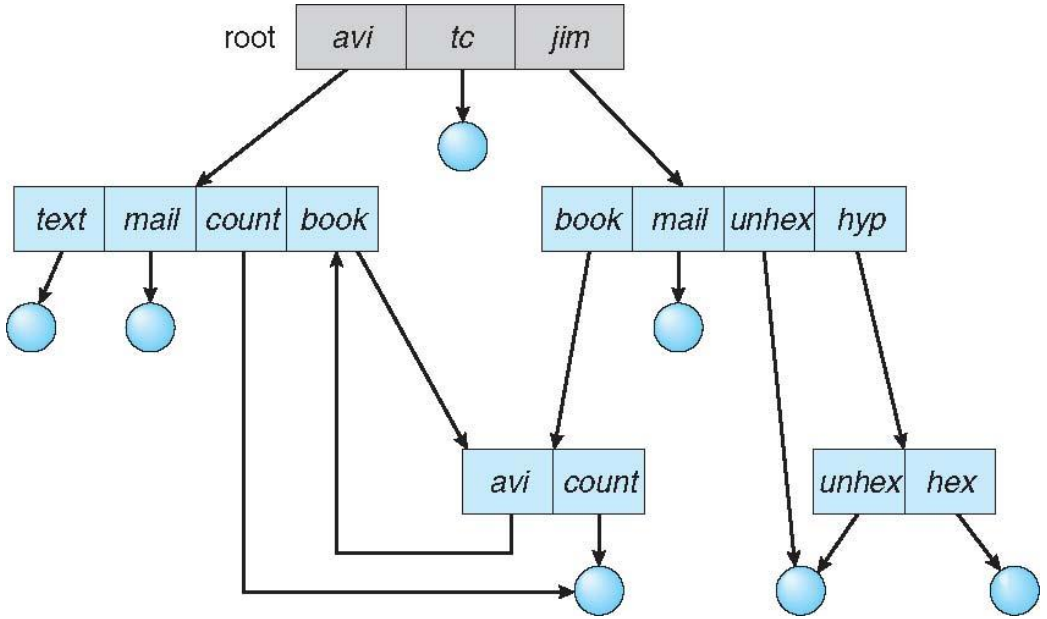
Acyclic-Graph Directories

- Aliasing: two different names
- If ***dict*** deletes ***count:***
  - dangling pointer
- Solutions:
  - Backpointers, so we can delete all pointers, but variable size records a problem
  - Entry-hold-count solution
- Hard links/symbolic links



General Graph Directory

- Efficiency and correctness problems: **Garbage collection**
- How do we guarantee no cycles?
  - Allow only links to file not subdirectories
  - Every time a new link is added use a cycle detection algorithm to determine whether it is OK



File-System Structure

file permissions
file dates (create, access, write)
file owner, group, ACL
file size
file data blocks or pointers to file data blocks

File system (FS) resides on disks. Organized into layers.

Logical file system directory management. Translates file name into file number, file handle, location, protection, etc.

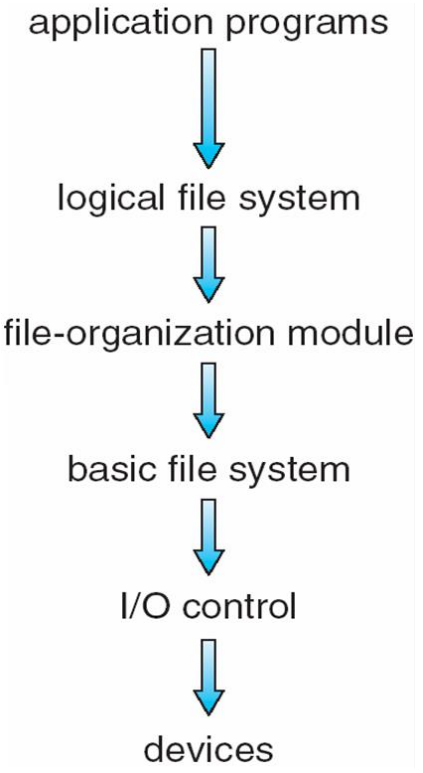
- Maintains **File control blocks** (**FCBs**) or **inodes** in UNIX.

File organization module understands files, logical addresses, and physical blocks. Translates logical block # to physical block #. Manages free space, disk allocation

Basic file system issues generic commands to the appropriate device driver. Manages memory buffers and caches (allocation, freeing, replacement)

Device drivers manage I/O devices at the I/O control layer.

Layering useful for reducing complexity and redundancy (e.g., several FSs) but adds overhead and can decrease performance.



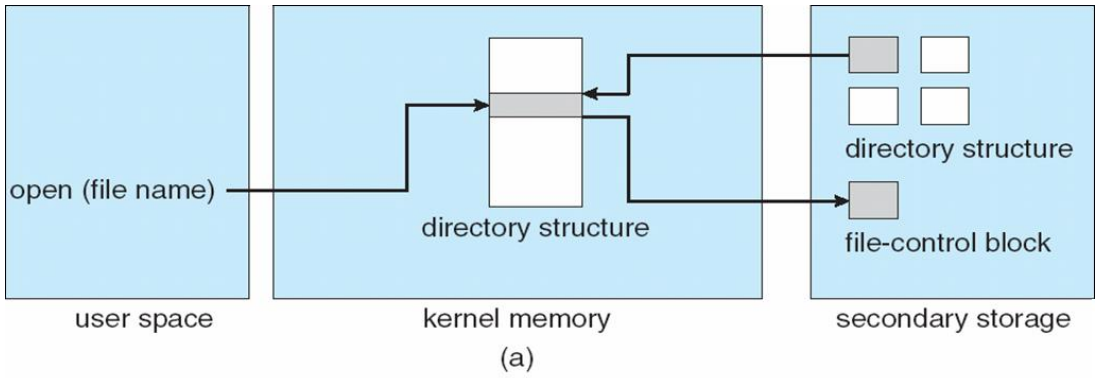
File-System Implementation

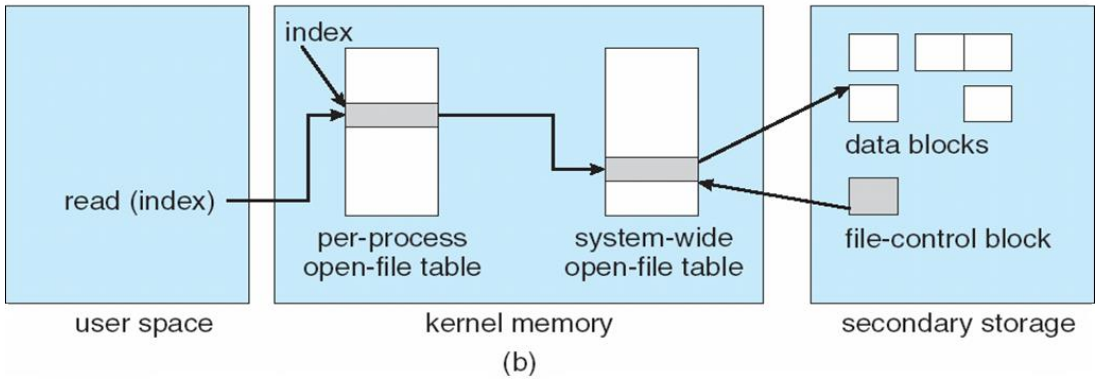
System calls at the API level are implemented with the help of on disk and in-memory structures, example:

- **Boot control block:** info needed by system to boot OS from a volume: Needed if volume contains OS, usually first block of volume
- **Volume control block (superblock, master file table)** contains volume details: Total # of blocks, # of free blocks, block size, free block pointers or array
- Directory structure organizes the files for each file system: Names and their inode numbers
- Per file control block (FCB) with a unique identifier and many details about the file.

In-Memory File System Structures

- In memory for FS management and performance. Data loaded at mount time:
  - Mount table storing file system mounts, mount points, file system types
  - Copy of the directory structure of recently accessed directories
  - System-wide open-file table: copy of FCB of open files
  - Buffers hold FS blocks when read/written to disk





Partitions and Mounting

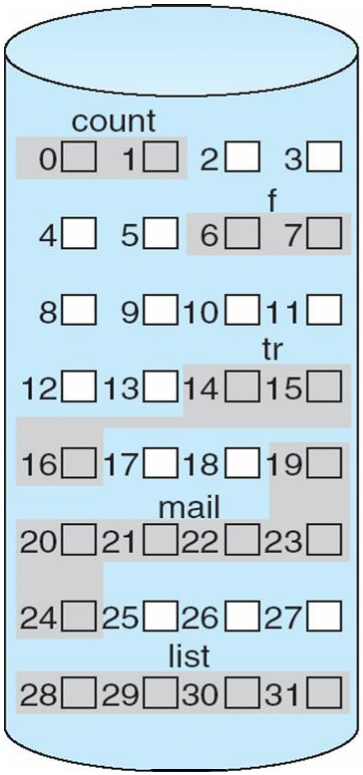
- Partition can be a volume containing a file system (“cooked”) or **raw** – just a sequence of blocks with no file system
- Boot block can point to boot volume or boot loader set of blocks that contain enough code to know how to load the kernel from the file system. System does not know FS code yet.
- **Root partition** contains the OS, other partitions can hold other Oses, other file systems, or be raw
  - Mounted at boot time
  - Other partitions can mount automatically or manually
- At mount time, file system reads device directory and consistency is checked

Directory Implementation

- **Linear list** of file names with pointer to the data blocks
  - Simple to program
  - Time-consuming to execute
    - Linear search time
    - Could keep ordered alphabetically via linked list or use B+ tree
- **Hash Table** – linear list with hash data structure
  - Decreases directory search time

Allocation Methods- Contiguous

- An allocation method refers to how disk blocks are allocated for files: **Contiguous, linked and indexed**.
- **Contiguous allocation** – each file occupies set of contiguous blocks
  - Best performance in most cases
  - Simple – only starting location (block #) and length (number of blocks) are required
  - Supports both sequential and direct access
  - Problems include finding space for file, knowing file size beforehand, external fragmentation, need for **compaction off-line** (**downtime**) or **on-line**

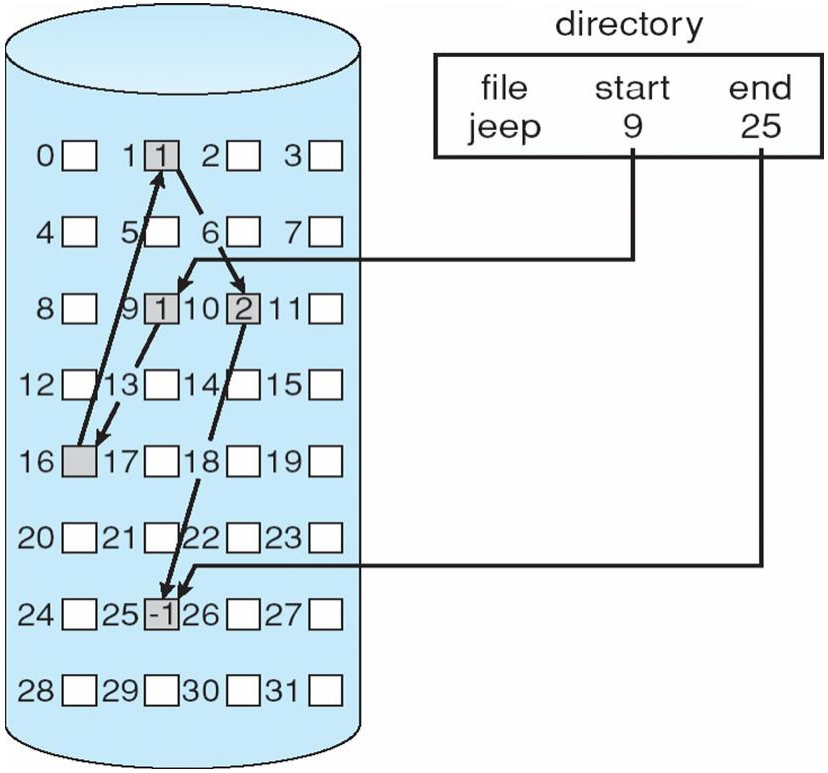


directory

file	start	length
count	0	2
tr	14	3
mail	19	6
list	28	4
f	6	2

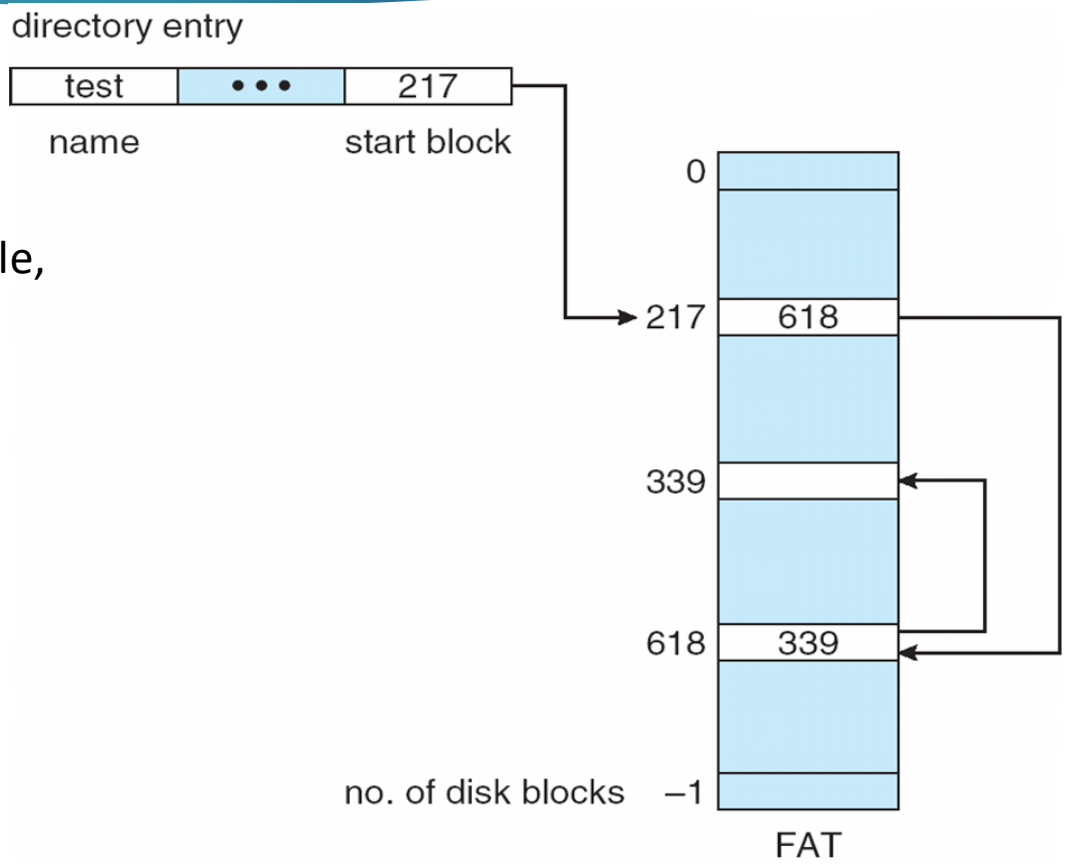
Allocation Methods - Linked

- **Linked allocation** – each file a linked list of blocks.
  - No external fragmentation. No need for compaction.
  - Each block contains pointer to next block
  - Reliability can be a problem
  - Locating a block can take many I/Os and disk seeks
  - Improve efficiency by clustering blocks into groups but increases internal fragmentation



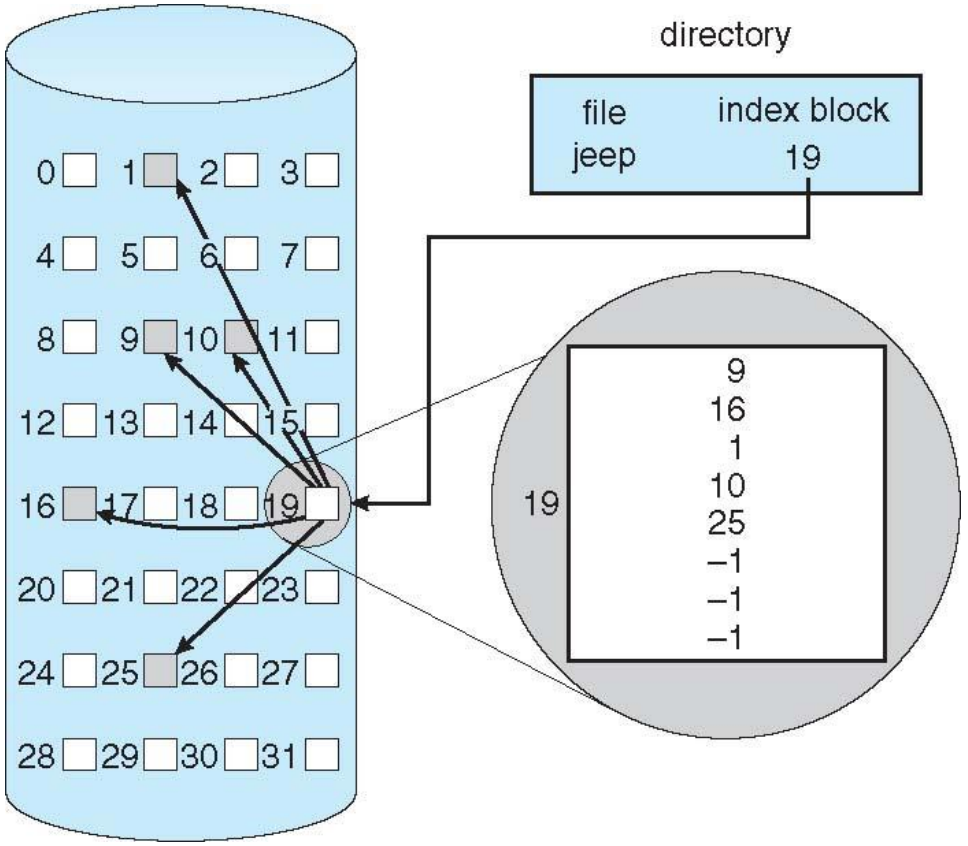
Allocation Methods - Linked

- FAT (File Allocation Table) variation
  - Beginning of volume has table, indexed by block number
  - Much like a linked list, but faster on disk and cacheable
  - New block allocation simple

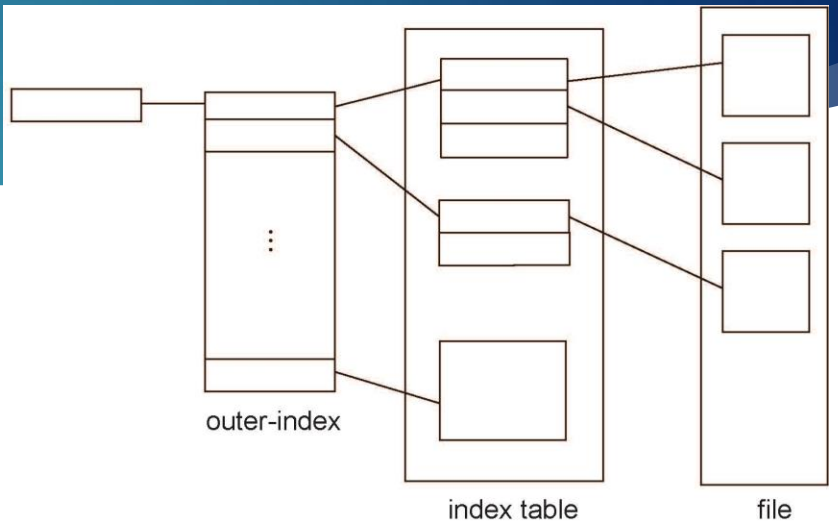


Allocation Methods - Indexed

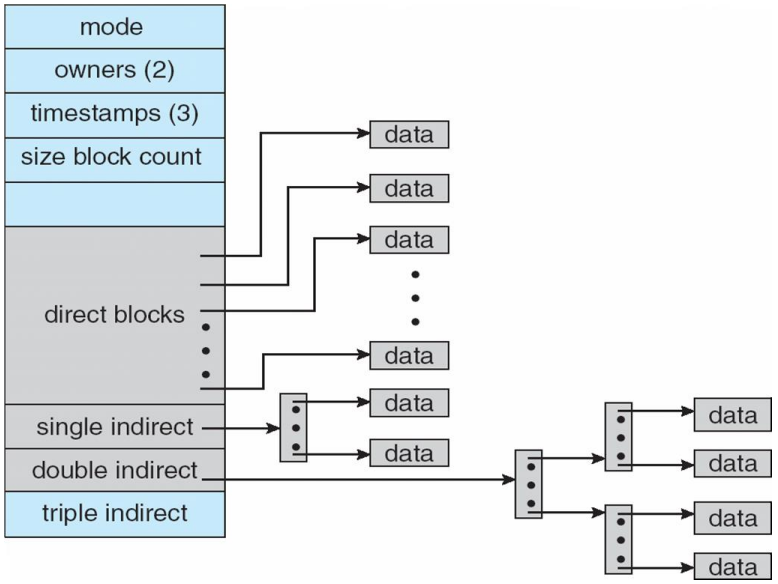
- Each file has its own **index block**(s) of pointers to its data blocks. Like the paging scheme.
- Supports direct access without external fragmentation.
- More overhead than linked allocation pointers.
- Index blocks can be cached, but blocks can be spread all over a volume



Indexed Allocation (Cont.)



- A block of 512 bytes can store 128 4-bytes pointers for a maximum file size of only 128 x 512 bytes= 64 KB.
- Linked scheme – Link blocks of index table (no limit on size). Keep last word to point to next index block.
- Two-level index (4K blocks could store 1,024 four-byte pointers in outer index -> 1,048,567 data blocks and file size of up to 4GB)
- Combined scheme (Unix based file systems): say 12 (direct),1 (single indirect),1 (double indirect),1 (triple indirect)



Performance

- Best method depends on file access type
  - Contiguous great for sequential and random
- Linked good for sequential, not random
- Some systems require to declare access type at creation -> select either contiguous for direct (but require max length) or linked for sequential
- Indexed more complex
  - Single block access could require 2 index block reads then data block read
  - Clustering can help improve throughput, reduce CPU overhead

Free-Space Management

- File system maintains **free-space list** to track available blocks/clusters
- **Bit vector** or **bit map** (***n*** blocks):
- CPUs have instructions to return offset within word of first "1" bit
- Bit map requires extra space:

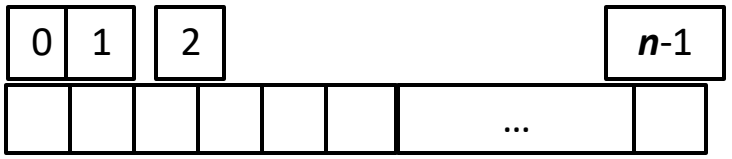
block size = 4KB = 2^{12} bytes

disk size = 2^{40} bytes (1 terabyte)

n = $2^{40}/2^{12} = 2^{28}$ bits (or 32MB)

if clusters of 4 blocks -> 8MB of memory

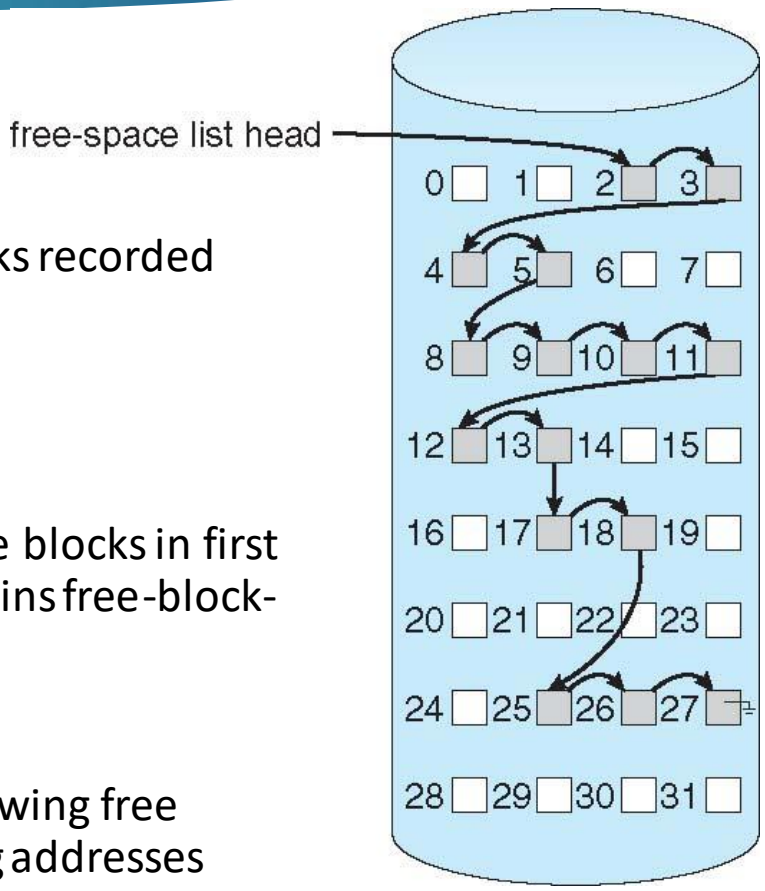
- Easy to get contiguous files



$$\text{bit}[i] = \begin{cases} 1 \Rightarrow \text{block}[i] \text{ free} \\ 0 \Rightarrow \text{block}[i] \text{ occupied} \end{cases}$$

Linked Free Space List on Disk

- Linked list (free list):
  - Cannot get contiguous space easily
  - No waste of space
  - No need to traverse the entire list if free blocks recorded like in FAT



- Grouping:

  Modify linked list to store address of next n-1 free blocks in first free block, plus a pointer to next block that contains free-block-pointers

- Counting:

  Keep address of first free block and count of following free blocks. Free space list then has entries containing addresses and counts

Recovery

- **Consistency checking** – compares data in directory structure with data blocks on disk, and tries to fix inconsistencies
  - Can be slow and sometimes fails
- Use system programs to **back up** data from disk to another storage device (magnetic tape, other magnetic disk, optical)
- Recover lost file or disk by **restoring** data from backup

Log Structured File Systems

- **Log structured** (or **journaling**) file systems record each metadata update to the file system as a **transaction**
- All transactions are written to a log
  - A transaction is considered committed once it is written to the log (sequentially)
  - However, the file system may not yet be updated
- The transactions in the log are asynchronously written to the file system structures
  - When the file system structures are modified, the transaction is removed from the log
- If the file system crashes, all remaining transactions in the log must still be performed
- Faster recovery from crash, removes chance of inconsistency of metadata

V. B. Mass Storage Systems

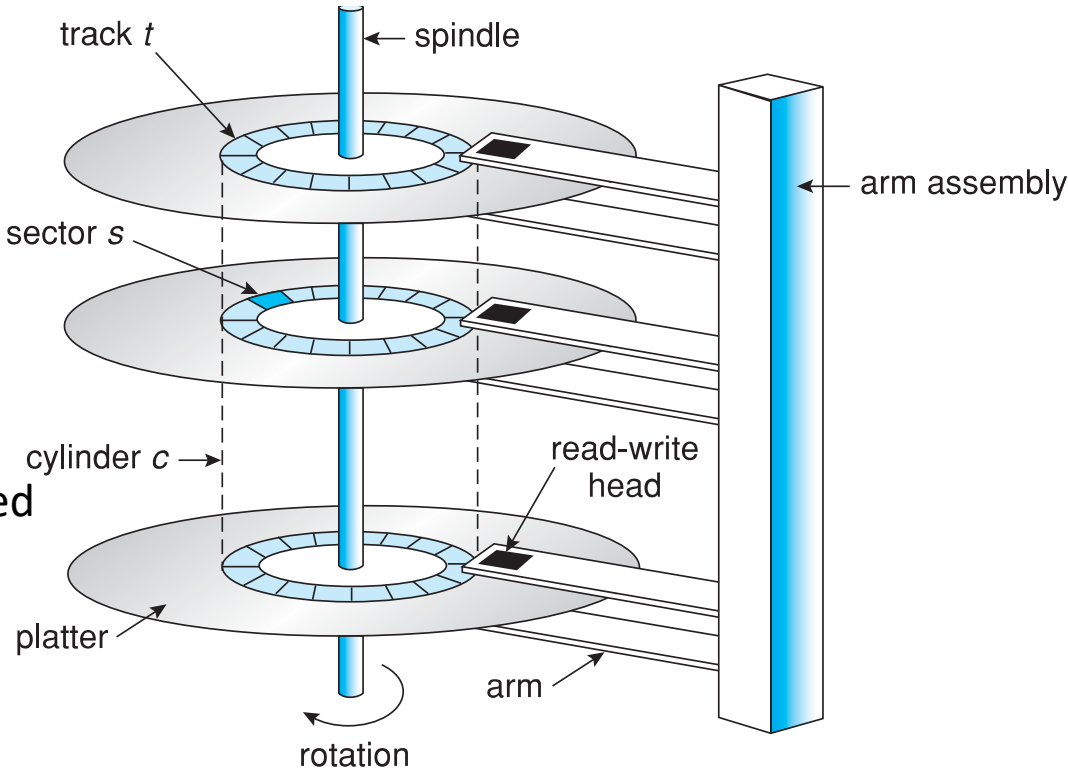
SGG9: chapters 10
SGG10: chapters 11

- hard disks, structure, disk scheduling

Overview of Mass Storage Structure

Magnetic disks bulk of secondary storage:

- Drives rotate at 60 to 250 times per second
- **Transfer rate** is rate at which data flow between drive and computer
- **Positioning time** (**random-access time**) is time to move disk arm to desired cylinder (**seek time**) and time for desired sector to rotate under the disk head (**rotational latency**)



Hard Disks

- Platters from .85” to 14”: commonly 3.5”, 2.5”
- Up to 10TB per drive
- Performance
  - Transfer Rate – around 1Gb/sec
  - Seek time from 3ms to 12ms – 9ms common for desktop drives
  - Average seek time measured or calculated based on 1/3 of tracks
  - Latency based on spindle speed
    - 1 / (RPM / 60) = 60 / RPM
  - Average latency = ½ latency

Spindle [rpm]	Average latency [ms]
4200	7.14
5400	5.56
7200	4.17
10000	3
15000	2

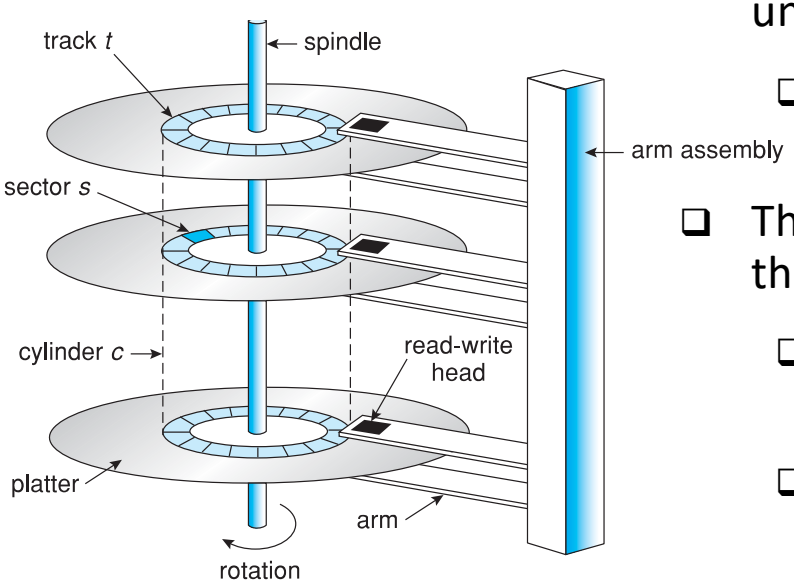
Hard Disk Performance

- **Access Latency** = **Average access time** = average seek time + average latency
  - For fastest disk 3ms + 2ms = 5ms
  - For slow disk 9ms + 5.56ms = 14.56ms
- Average I/O time = average access time + (amount to transfer / transfer rate) + controller overhead
- For example, to transfer a 4KB block on a 7200 RPM disk with a 5ms average seek time, 1Gb/sec transfer rate with a .1ms controller overhead =
  - 5ms + 4.17ms + 0.1ms + transfer time =
  - Transfer time = 4KB / 1Gb/s * 8Gb / GB * 1GB / 1024^2KB = 32 / (1024^2) = 0.031 ms
  - Average I/O time for 4KB block = 9.27ms + .031ms = 9.301ms

Solid-State Disks

- Nonvolatile memory used like a hard drive
  - Many technology variations
- Can be more reliable than HDDs
- More expensive per MB
- May have shorter life span
- Less capacity
- But much faster
- No moving parts, so no seek time or rotational latency

Disk Structure



- Disk drives are addressed as large 1-dimensional arrays of **logical blocks**, where the logical block is the smallest unit of transfer
  - Low-level formatting creates **logical blocks** (typically 512B)
- The 1-dimensional array of logical blocks is mapped into the sectors of the disk sequentially
  - Sector 0 is the first sector of the first track on the outermost cylinder
  - Mapping proceeds in order through that track, then the rest of the tracks in that cylinder, and then through the rest of the cylinders from outermost to innermost

Disk Scheduling

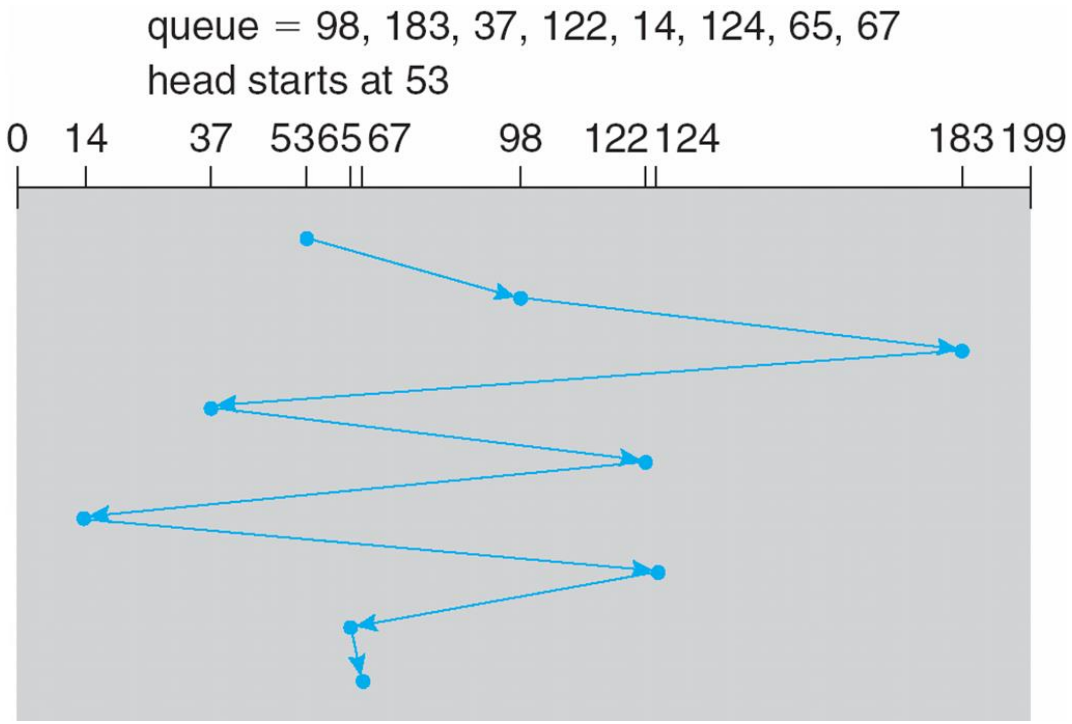
- The operating system is responsible for using hardware efficiently — for the disk drives, this means having a fast access time and disk bandwidth
- Minimize seek time
- Seek time ≈ seek distance
- Disk **bandwidth** is the total number of bytes transferred, divided by the total time between the first request for service and the completion of the last transfer

Disk Scheduling (Cont.)

- There are many sources of disk I/O request
  - OS, System processes, Users processes
- I/O request includes input or output mode, disk address, memory address, number of sectors to transfer
- OS maintains queue of requests, per disk or device
- Idle disk can immediately work on I/O request, busy disk means work must queue
  - Optimization algorithms only make sense when a queue exists

Disk Scheduling (Cont.)

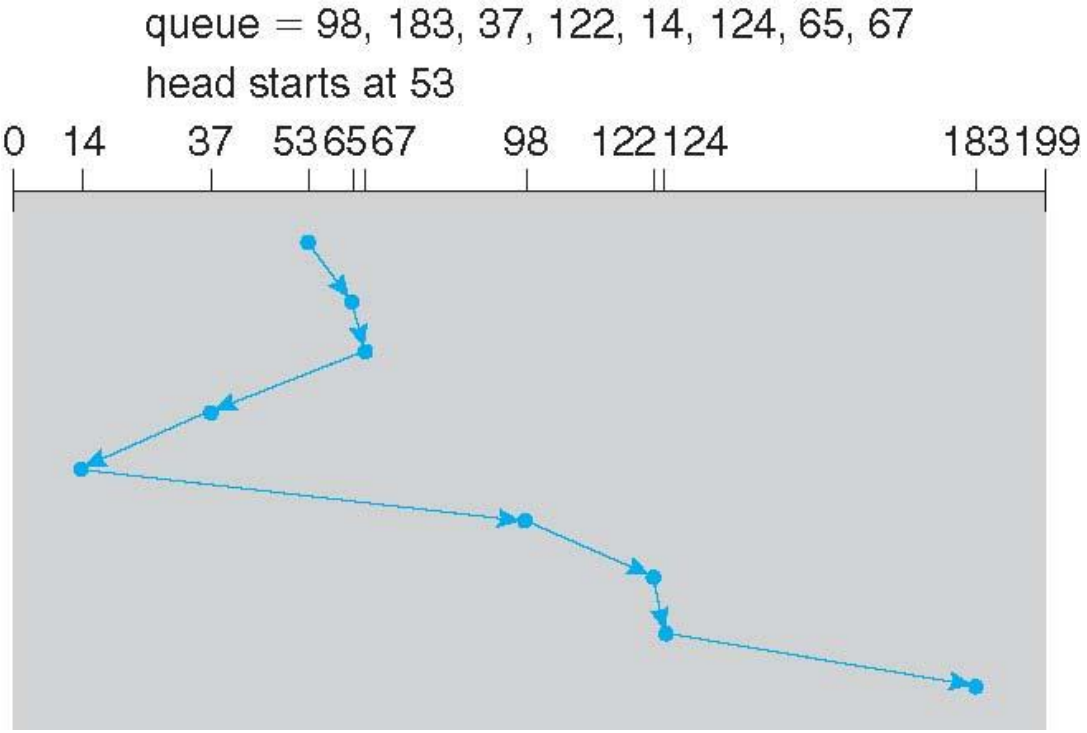
- Note that drive controllers have small buffers and can manage a queue of I/O requests (of varying "depth")
- Several algorithms exist to schedule the servicing of disk I/O requests
- The analysis is true for one or many platters


queue = 98, 183, 37, 122, 14, 124, 65, 67
head starts at 53
0
14
37
53
65
67
98
122
124
183
199

FCFS

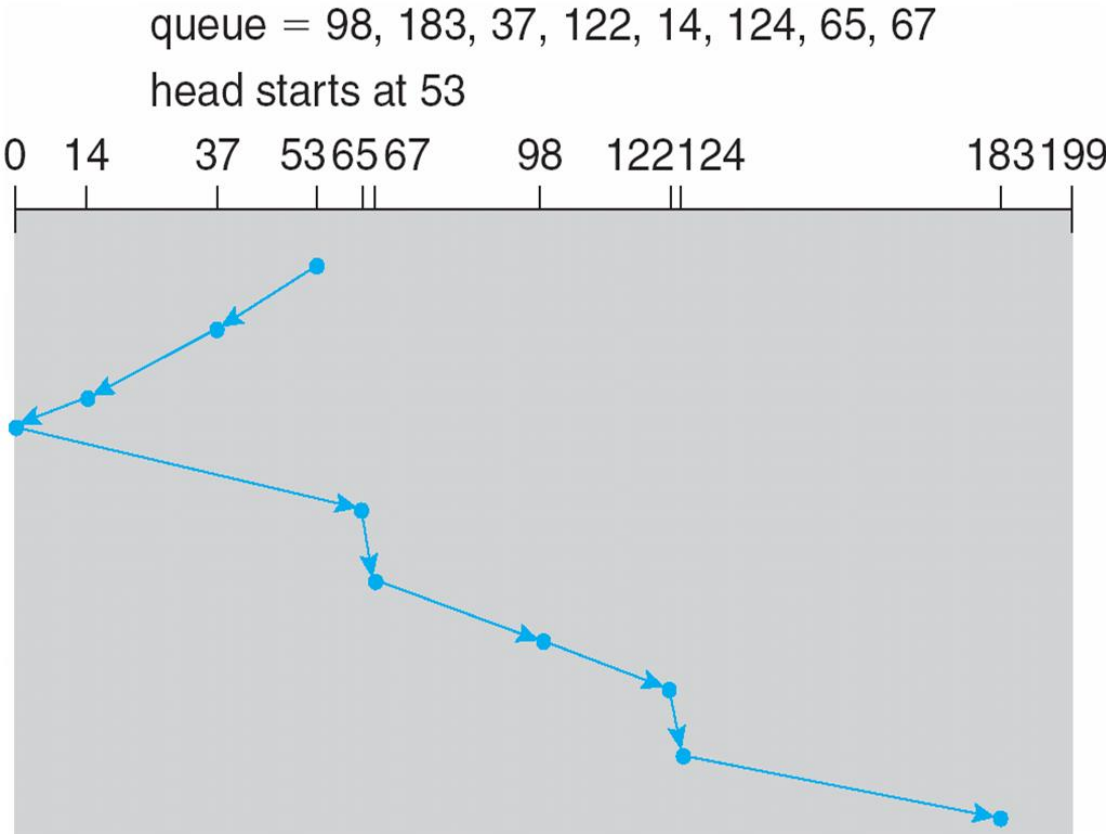
SSTF

- Shortest Seek Time First selects the request with the minimum seek time from the current head position
- SSTF scheduling is a form of SJF scheduling; may cause starvation of some requests
- Illustration shows total head movement of 236 cylinders



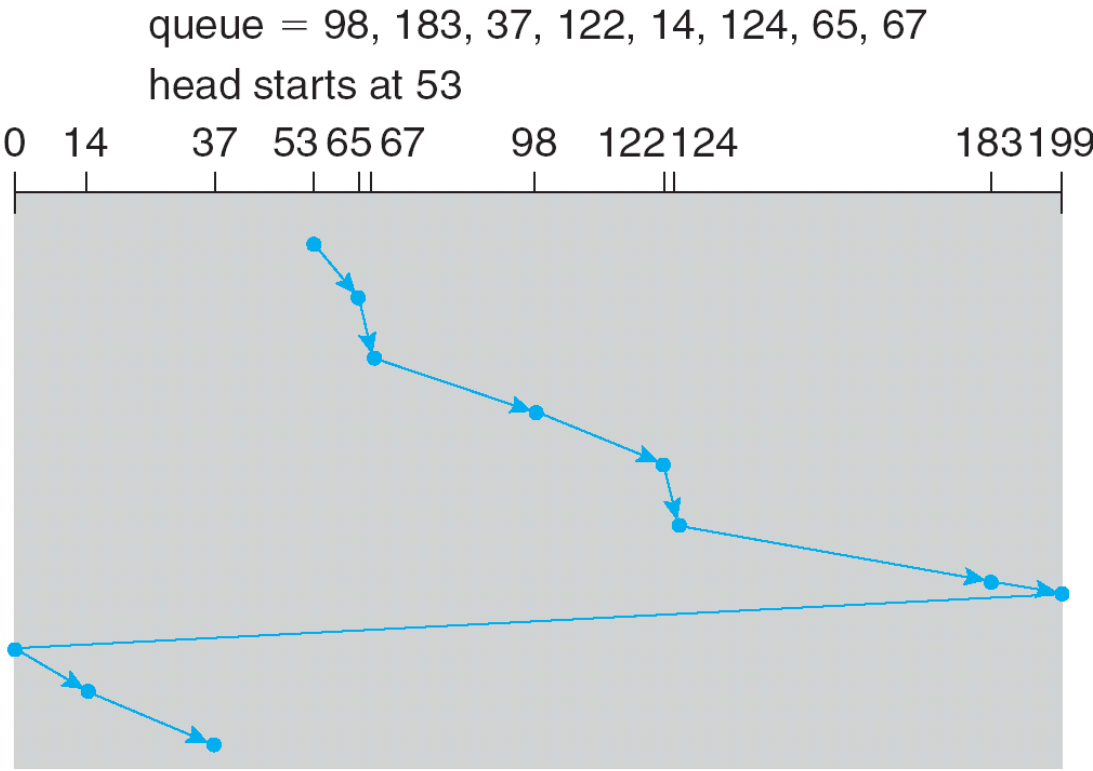
SCAN

- The disk arm starts at one end of the disk, and moves toward the other end, servicing requests until it gets to the other end of the disk, where the head movement is reversed, and servicing continues.
- **SCAN algorithm** Sometimes called the **elevator algorithm**
- Illustration shows total head movement of 208 cylinders



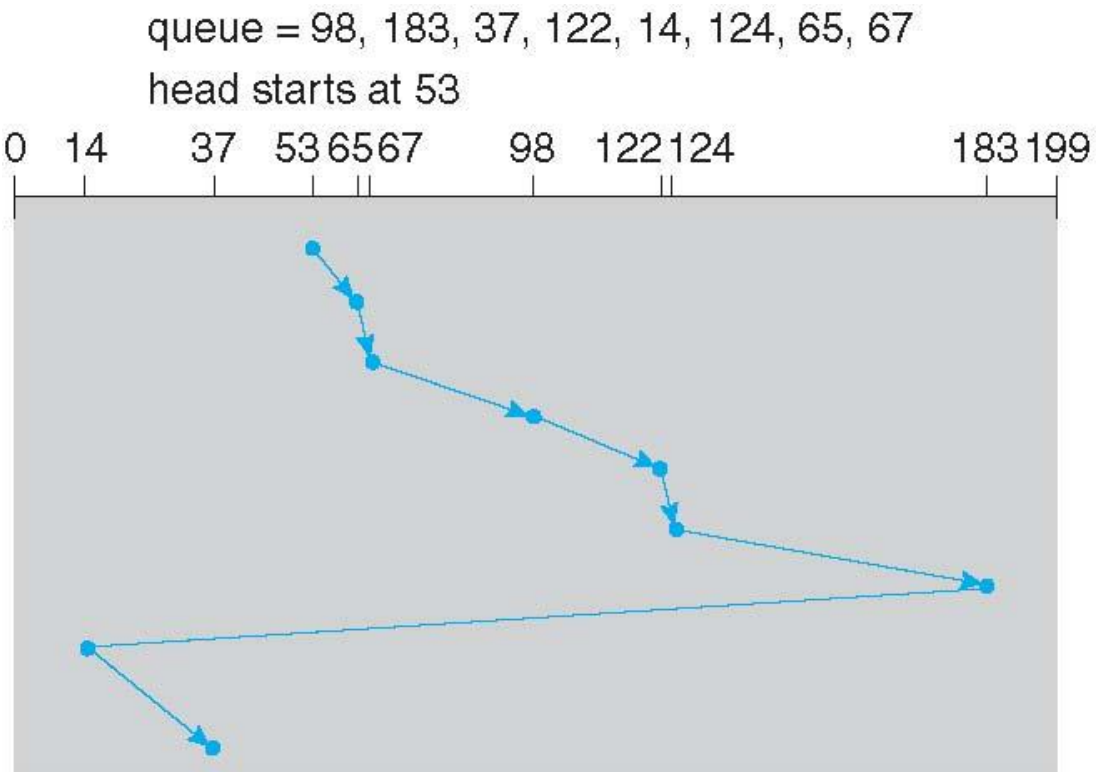
C-SCAN

- Provides a more uniform wait time than SCAN
- The head moves from one end of the disk to the other:
  - When it reaches the other end, however, it immediately returns to the beginning of the disk
- Treats the cylinders as a circular list that wraps around from the last cylinder to the first one



C-LOOK

- LOOK a version of SCAN, C-LOOK a version of C-SCAN
- Arm only goes as far as the last request in each direction, then reverses direction immediately, without first going all the way to the end of the disk
- Total number of cylinders?



Selecting a Disk-Scheduling Algorithm

- SSTF is common and has a natural appeal
- SCAN and C-SCAN perform better for systems that place a heavy load on the disk: Less starvation
- Requests for disk service can be influenced by the file-allocation method and metadata layout
- The disk-scheduling algorithm should be written as a separate module of the operating system, allowing it to be replaced with a different algorithm if necessary
- Either SSTF or LOOK is a reasonable choice for the default algorithm
- What about rotational latency?
  - Difficult for OS to calculate
- How does disk-based queueing effect OS queue ordering efforts?