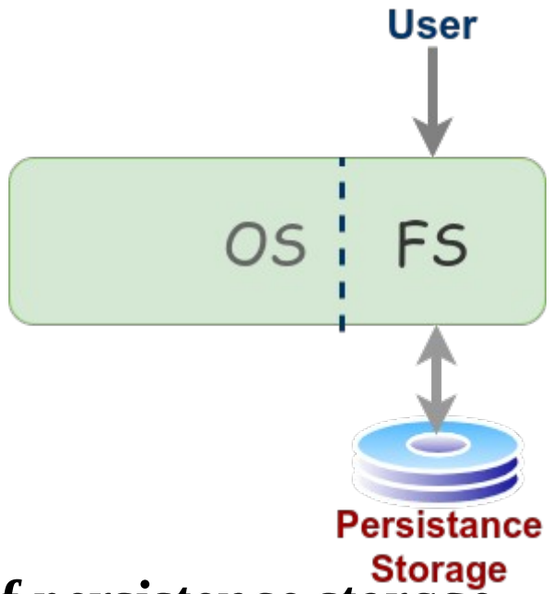
Unit 6 : Storage Management

by:
Manojeet Roy

Table of Contents

Why FS?

- Need of persistence storage.
- Hardware for persistence storage:
  - Magnetic Disk
  - HDD
  - SDD
  - NVMe
- OS requires to manage persistence storage.
- ***FS: OS by FS manages mechanism and policies of persistence storage***
- Types of FS implementations:
  - Simple – Ext2/3/4
  - Complex – XFS, JFS
  - Networked – NFS, CIFS, AFS, Coda
  - Non-Unix – VFAT, NTFS
  - Pseudo – Procfs, Sysfs, Configfs, Debugfs



FS to File

- [**Filesystem (FS)**]:
  - [**DEF1**]: The FS is a system that implements, organizes, manages, and accesses the files and directories on a device's persistent/secondary storage.
    - Empowered by OS, FS is a software by which users and applications organizes and manage their files.
      - ✓ This means whenever we do any operation on file, FS get involved.

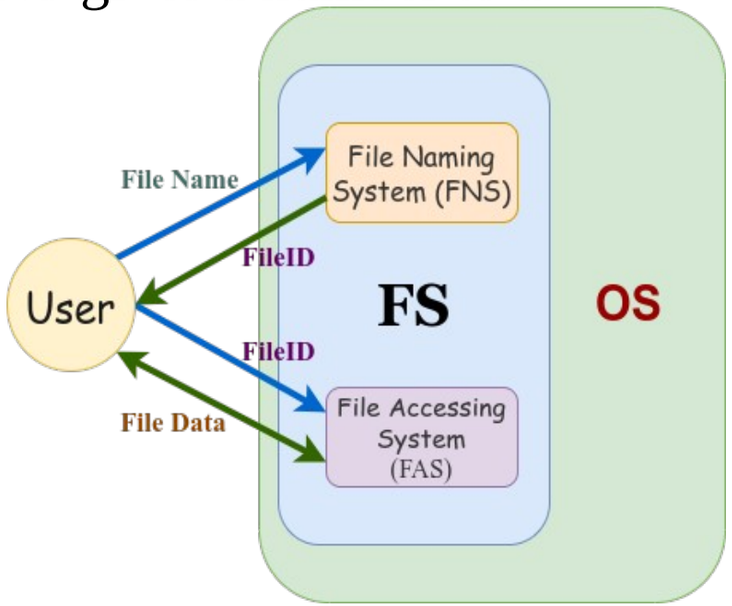
- File (*basic definition as user*):
  - File is a sequenced collection of information that have some structure.

- One hope following basic operations on file:
  - [1]: Right privileged user can operate/access (e.g. create/delete, read/write) on file.
  - [2]: User should able to locate the file in secondary (persistence) memory.

- OS provide above 2 basic ops through FS. Therefore, top level components of FS are:
  - File Naming System (FNS):
  - File Accessing System (FAS):

- **Remark**:
  - Using FNS, user can provide "name" to a file. In return, FNS gives a number to a file called fileID (FID).
  - This FID is used by FAS to access file data.
  - FS also needs to keep information about file (***meta data***)

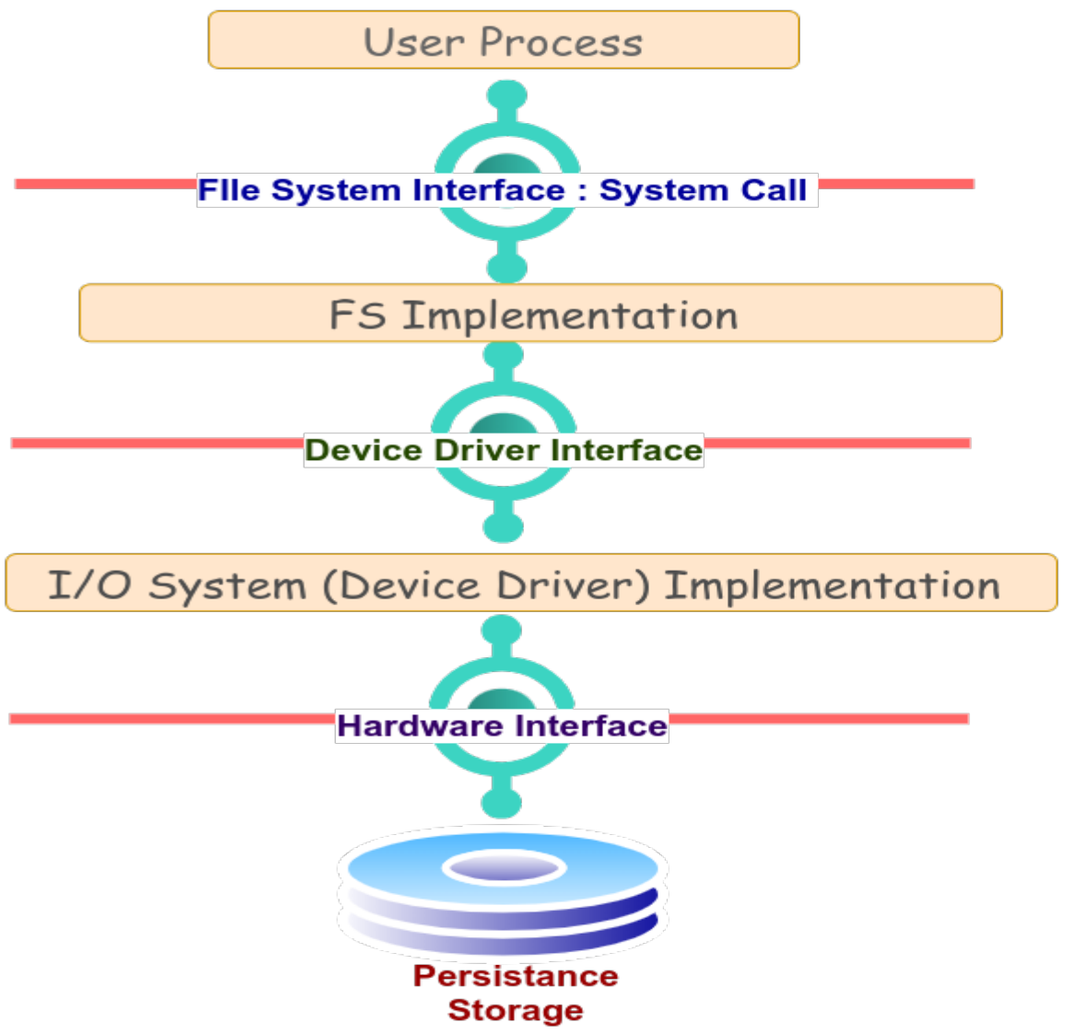
- [**DEF:** file]:
  - *A file is a named sequenced collection of information that have some structure and have some associated meta-information about file itself.*



The 2 main parts of a FS

FS Levels

- User Process interact with FS by using **system call**.
- FS interface abstracts the details of **logical blocks arrangements**
- FS implementation manages reading and writing of blocks using policies of **allocation methods**.
- Disk device driver interface **abstracts the physical details** of the drive and represent disk as logical disk
- I/O Disk driver implementation interact with hardware interface to manages the actual **reading and writing of blocks to the physical disk**



Process sees file as a array of bytes

FS sees HDD as logical disk (array of blocks (sectors))

Disk device driver sees HDD as (cylinder, track, sector)

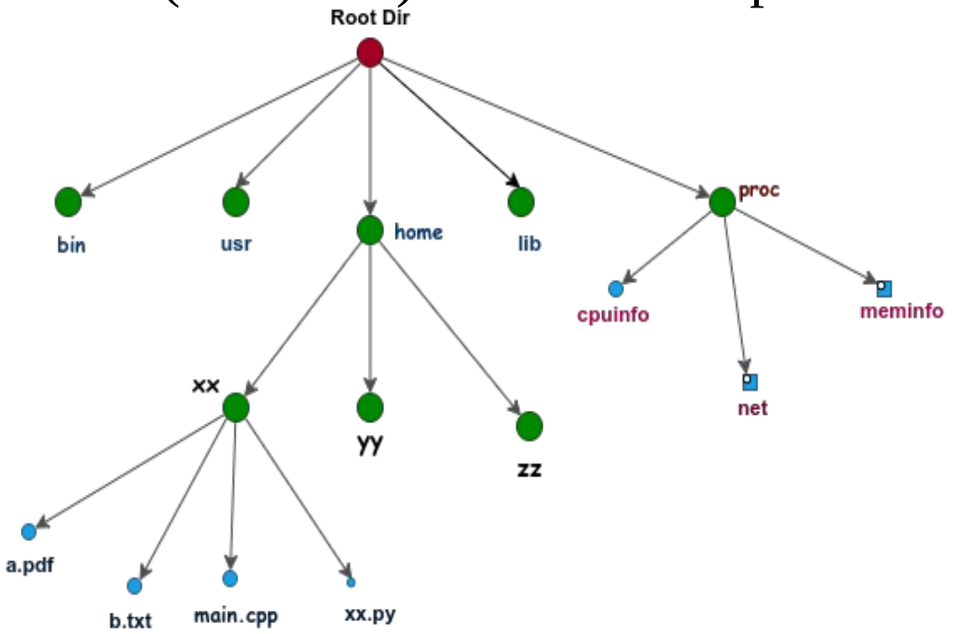
File (Abstract level Interpretation)

- It is a basic concept in FS
- Usual Examples: txt, docx, pdf, bin, exe files, But they are more …….
- Concept of file is an abstraction of “sink for” and/or “source of” data/information.
  - This concept also provides abstraction for all devices, attached to computer.
  - [**NOTE**]: Linux/Unix abstracts everything as a file. For example
    - [SCREEN]:
      - It is a write-only file i.e. act as sink for data
      - It do not store information and only display (write) information on screen
    - [KEYBOARD]:
      - It is read-only file i.e. act as source of data
    - [PRINTER]
      - It is a write-only file i.e. act as sink for data
- Remark:
  - Every running program starts with 3 files
    - [Stdin] : This file is responsible to captures the input from keyboard
    - [StdError]: This file is responsible to send error signals output to console
    - [Stdout]: This file is responsible to send output to console

File Naming System (FNS)

Most OS follows “Hierarchical Naming System” (HNS) policy in which

- Filenames are structured as a tree (shown in Fig) with following characteristics
  - Nodes are directories
  - Leafs are files
- HNS named files with structured multipart filenames
  - A filename represents path from root directory to the file being named
  - Special Character (“\” or “/”) are used to separate names in the pathname.



For Linux:
Using HNS policy, we can naming and identification (location) of a file (e.g. /home/xx/main.cpp) for the given structural organization

File Naming System (FNS)

- [Pathname]:
  - It consist of series of component names separated by separator character (“\” or “/”)
  - Main issues related to component names
    - [1]: What subset of character are allowed to populate component-names
    - [2]: What would be maximum length of component name.
  - Types of pathname
    - Absolute path name
    - Relative path name

Concept of Namespace in CS

- [**General Interpretation**]:
  - A **namespace** is a container (or context) that holds a set of identifiers (e.g. name of a file or variables/objects in programming) that can be uniquely identified.
- [**OS specific Interpretation**]
  - With respect to FS, namespace refers to the file system hierarchy such that each directory again acts as a namespace for its contents to uniquely identify them.

Concept of Namespace in CS

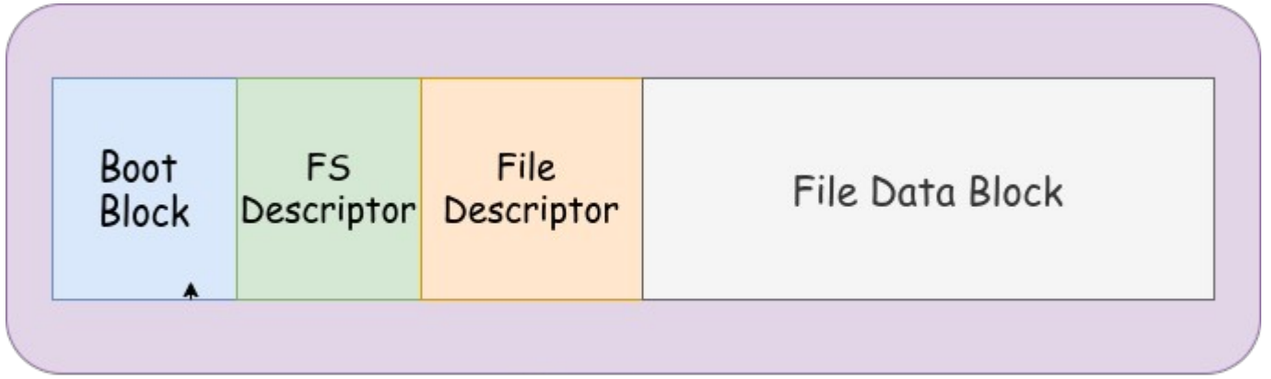
Example: Suppose, a file result.txt full path is /home/user/documents/result.txt, then

- **Wrt Home Directory Namespace:**
  - The file result.txt can be uniquely identified by its full path /home/user/documents/report.txt.
  - This full path specifies the unique location of the file within the entire file system starting from the root directory /.
- **Wrt Documents Directory Namespace:**
  - Within the documents directory namespace (context), result.txt is uniquely identified as result.txt.
  - This means The name result.txt is unique within the documents directory
- **[Observation]:**
  - The identification of the file report.txt can vary based on the namespace context i.e.
  - Global (File System) Namespace: /home/user/documents/report.txt
  - Local (Documents Directory) Namespace: report.txt

- *More discussion on namespace when we study directory and its structure ……..*

File system structure

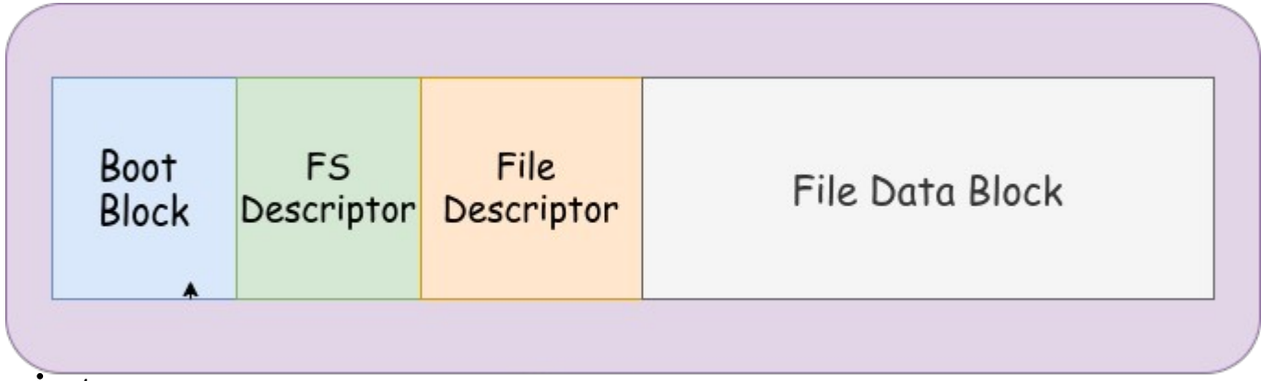
- Most FS are stored on logical disk.
  - Physically, this might be single/part/multiple physical disk
- I/O transfers between memory (RAM) and disk is performed in units of blocks
  - This is done to improve data transfer performance.
  - One Block == 1 or more sectors
  - **32B** ≤ Size of sector ≤ **4096B** (Usually: **512B**)



Basic FS Layout

File system structure (contd …)

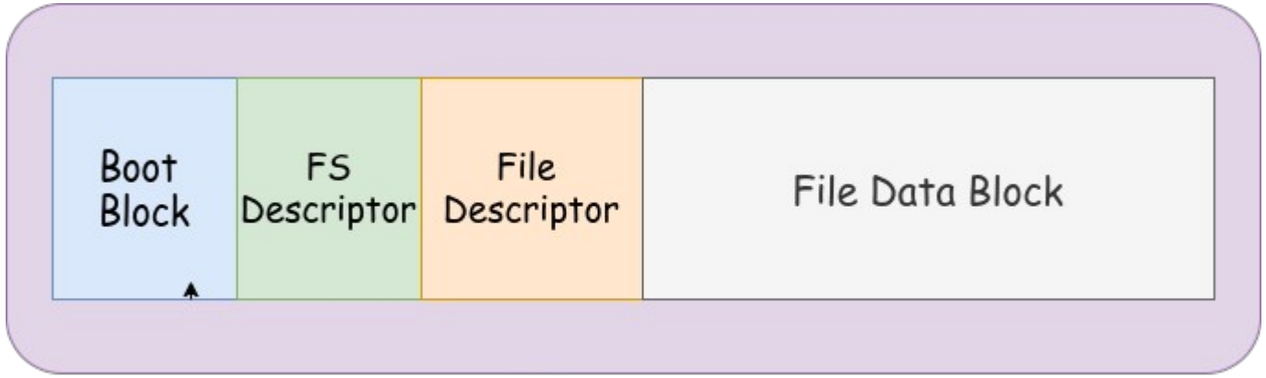
- Building blocks of file system structure
  - **Boot Block**:
    - contains info needed by system to boot OS
  - **FS descriptor** :
    - Descriptor for entire FS
    - Content of FS descriptor area:
      - Total size of FS (in blocks)
      - Size of file descriptor area
      - Location of root directory file descriptor
      - Timestamp: when FS created, last modified, last used, etc.
      - Other FS meta data
    - In Unix it is called as Superblock



Basic FS Layout

File system structure (contd …)

- **File Descriptor:**
  - Descriptor for files that record all meta info about file, example
    - Location information of file in file data block, etc.
    - In Unix, file descriptor is called as **inode** (index node)
- **File Data Block:**
  - Contains actual data blocks for files



Basic FS Layout

File attributes

- Common attributes of files are:
  - Filename
  - File Type
  - Location
  - Size
  - Protection/Permission
  - Time stamp
  - Ownership
  - Directory Information

File operations

- Common file operations:
  - Create: Allocate space for a new file.
  - Read: Fetch data from a file.
  - Write/Append: Add data to a file.
  - Delete: Remove a file from the file system.
  - Seek: Move to a specific location in a file.
  - Open/Close: Prepare file for use/end file usage

File types

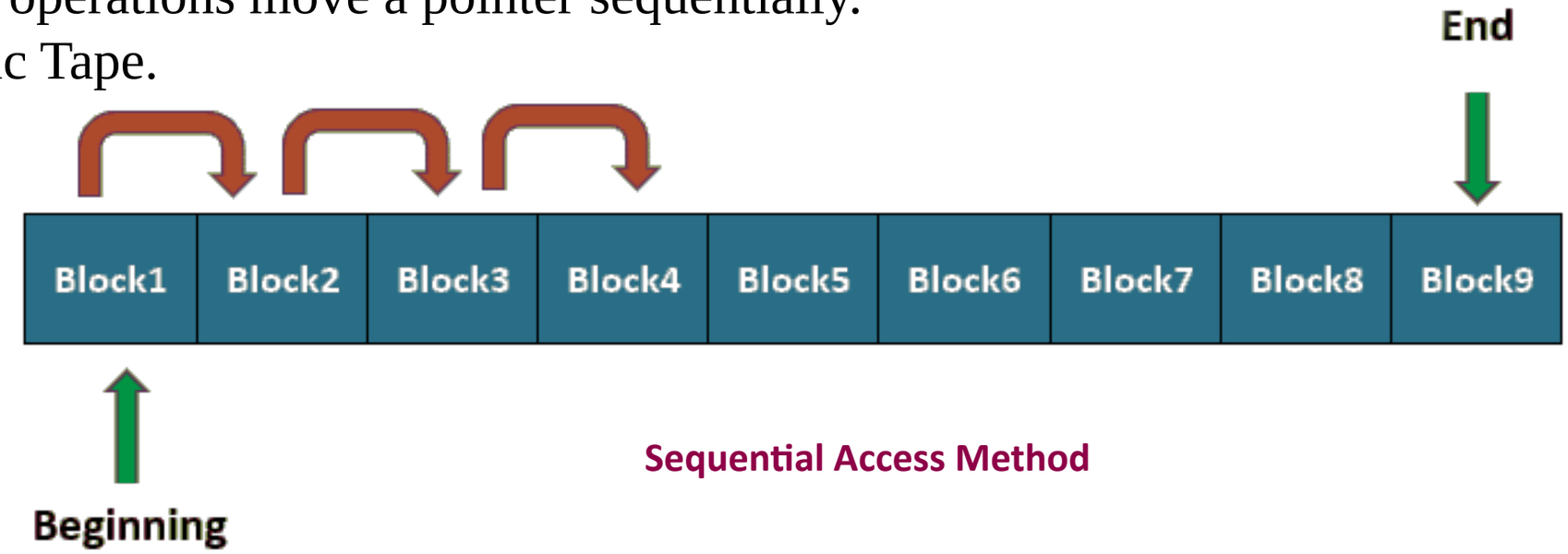
File Type	Extensions	Description	Examples
Executable	exe, com, bin	Ready-to-run machine language programs	Windows applications, Linux binaries
Object	obj, o	Compiled machine language, not yet linked	Intermediate files in the compilation process
Source Code	c, cc, java, pas, asm, a	Source code in various programming languages	C source files, Java source files, Pascal programs
Batch	bat, sh	Commands for the command interpreter	Windows batch files, Unix shell scripts
Text	txt, doc	Textual data and documents	Notepad files, Word documents
Word Processor	wp, tex, rtf	Various word processor formats	Word processor documents, LaTeX files, Rich Text Format files
Library	lib, a, so, dll	Libraries of routines for programmers	Dynamic Link Libraries (DLLs), Static libraries (.lib)
Print or View	ps, pdf, jpg	ASCII or binary file in a format for printing or viewing	PDF files, PostScript files, JPEG images
Archive	arc, zip, tar	Related files grouped into one file, sometimes compressed	ZIP files, TAR archives
Multimedia	mpeg, mov, rm, mp3, avi	Binary files containing audio or A/V information	MPEG videos, MP3 audio files, AVI videos

File access method

Contents of file can be accessed using different methods as listed below:

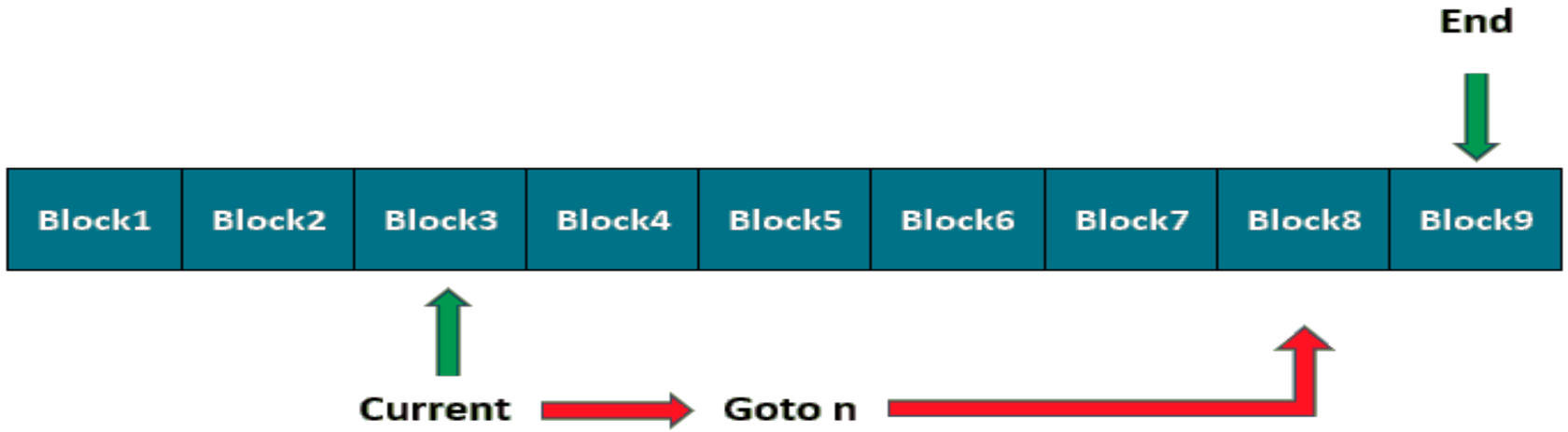
❑ Sequential access

- **Accessing Way:** In order, one record after another
- **Characteristics:**
  - Simple and easy to implement.
  - Suitable for text files and log files.
  - Read and write operations move a pointer sequentially.
- **Example:** Magnetic Tape.



❑ Direct access:

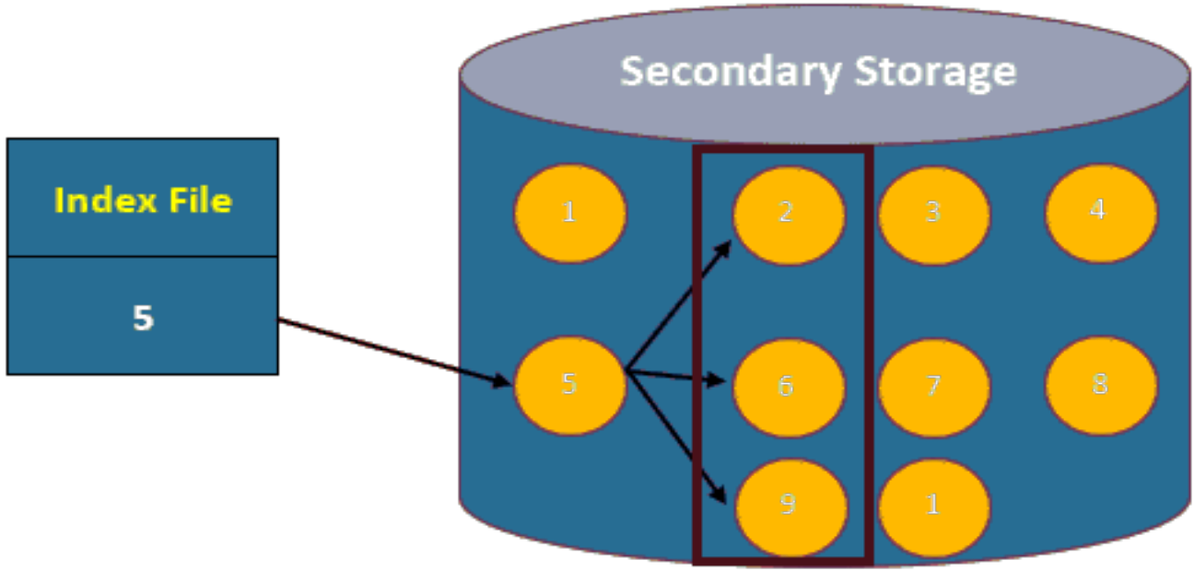
- **Accessing Way:** In any order using a relative address or index, skipping the previous data blocks.
- **Characteristics:**
  - Allows fast access to any part of the file.
  - Suitable for databases and large files where quick access is needed.
  - Each block or record has a unique identifier.
- **Example:** Accessing a specific customer record in a database.



Direct Access Method

❑ Indexed access:

- **Accessing Way:** In any order, but accessed using particular key(s) or index to find the location of data blocks.
- **Characteristics:**
  - Combines benefits of both sequential and direct access.
  - Index file points to data blocks, enabling quick access.
  - Suitable for large files with varied access patterns.
- **Example:** Accessing chapters in an e-book using a table of contents.



Indexed Access Method

File system mounting

- Issues related to FS Structure with one logical disk (*discuss in last lecture*)
  - Keeping backup is hard
  - Failure in any disk brings down entire system
- [**Possible Solution**]:
  - Different OS vendor adopted different strategy
    - [**MS/DOS Strategy**]:
      - Partition the disk
      - Give a letter name to each disk (e. g. C, D ... .)
        - This means pathname would be for example C:\usr\bin
    - [**Unix/Linux Strategy**]

File system mounting

- [**Unix/Linux Strategy**]
  - Use Concept of Mounting
    - Using this concept one can mount given FS into the directory of current OS
    - Mounting requires 2 arguments
      - **[1]** Mounting Point:
        - A directory in the FNS of current OS
      - **[2]** FS which needs to be mounted
    - [**NOTE**]: Mounting operation allow user to combine any number of FS into a single directory-tree.

File system mounting ……

Linux FS Mounting

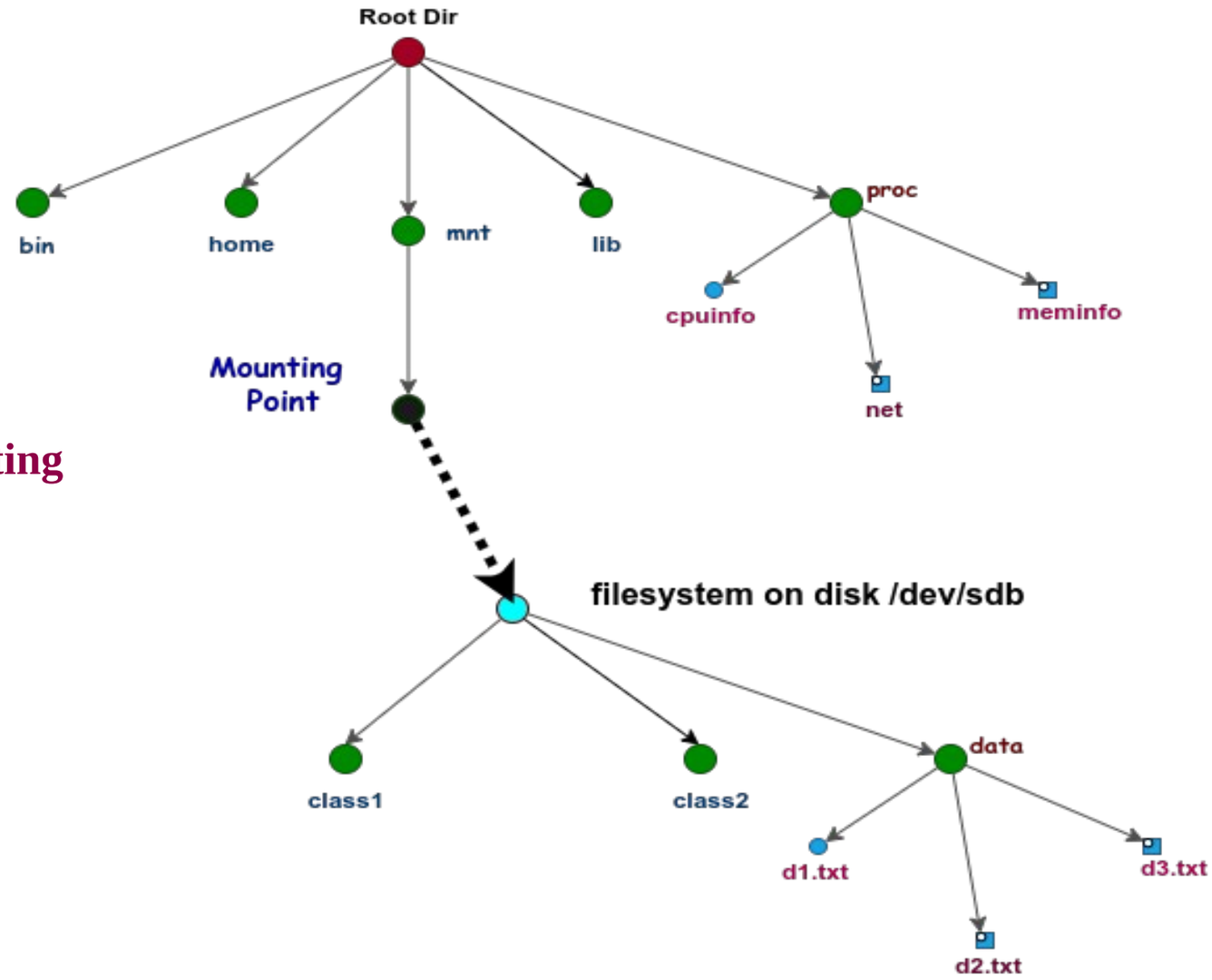
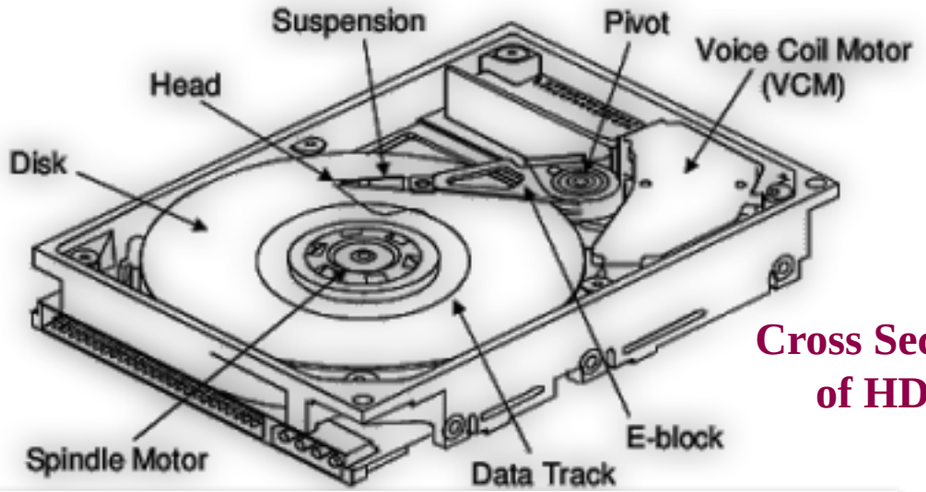
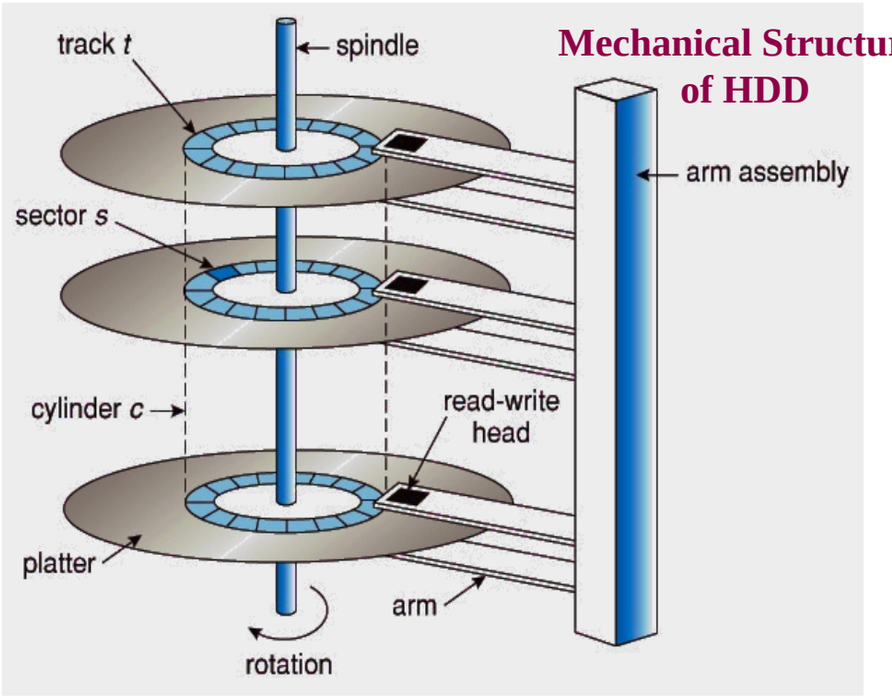


Table of Contents

Disk Structure

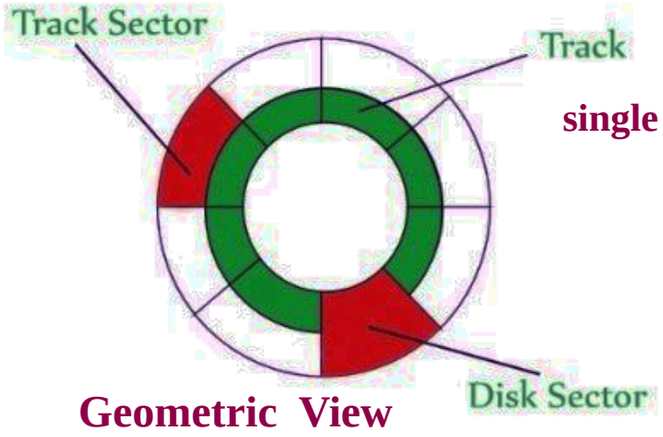


Cross Section of HDD

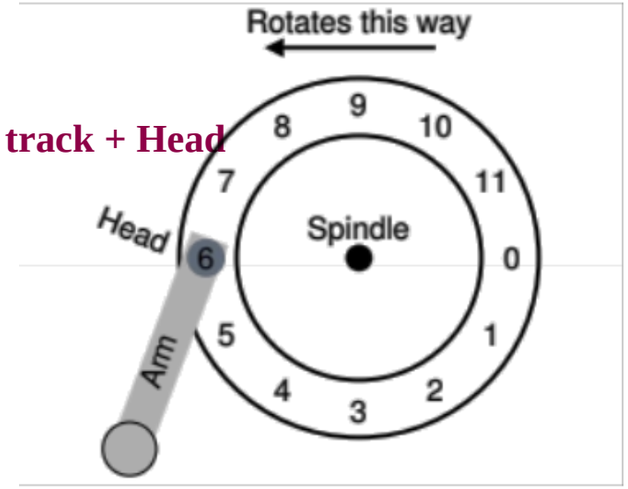


Mechanical Structure of HDD

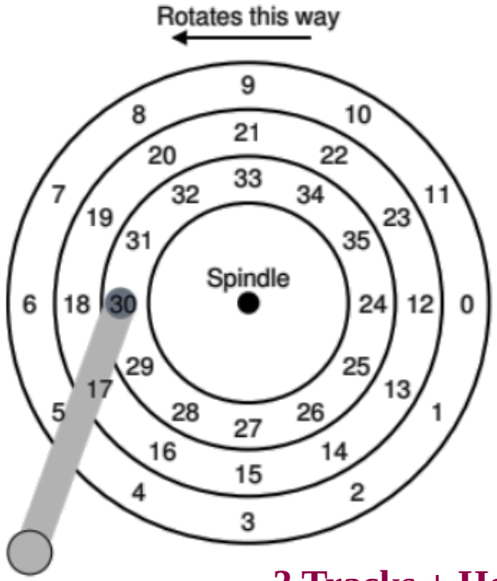
Disk Structure : Simplified

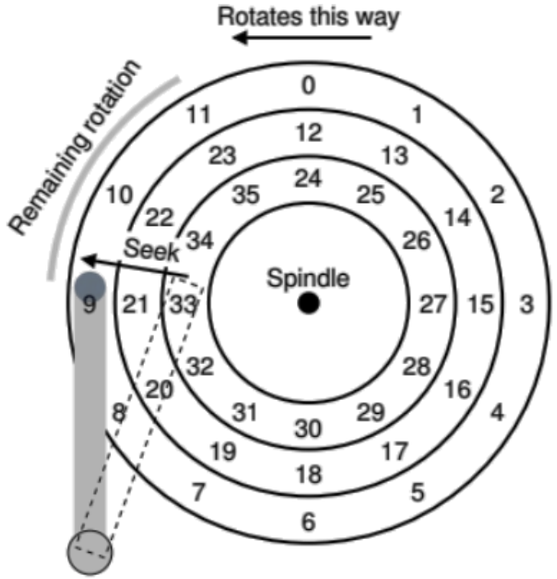


Geometric View



single track + Head





3 Tracks + Head + Seek (Right)

R/W process: (3 components)

- **SEEK**: Positioning R/W head over correct track
- **ROTATION**: Waiting for desired sector to rotate under head
- **Transfer**: Do R/W

Remark

- When arm goes from one track to another then due to rotational effect we need to consider ***Track Skew***
- Outer tracks covers more distance then inner track.

Block mapping problem

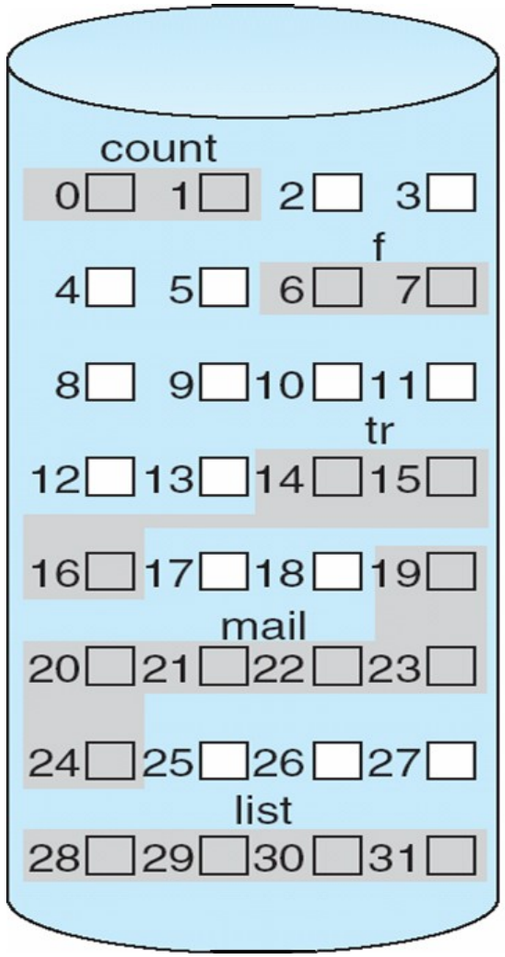
Disk Allocation Methods

- An allocation method refers to how disk blocks are allocated for files:
- Some of the commonly used methods are:
  - **Contiguous Allocation**
  - **Linked Allocation**
  - **Indexed Allocation**

Contiguous Allocation

- Allocates a single contiguous set of blocks to a file.
- **Characteristics:**
  - Simple and easy to implement.
  - Provides fast access to file data.
  - Can lead to external fragmentation.
  - Difficult to find a contiguous block of space for large files.
- **Advantages:**
  - Fast read/write access due to sequential block placement.
- **Disadvantages:**
  - Can cause external fragmentation.
  - Difficult to grow files if adjacent blocks are occupied.

Example Contiguous Allocation



directory

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

Linked Allocation

Definition: Each file is a linked list of disk blocks, where each block contains a pointer to the next block.

Characteristics:

- No external fragmentation.
- Only the block pointer is stored in each block.
- Suitable for sequential access, not random access.

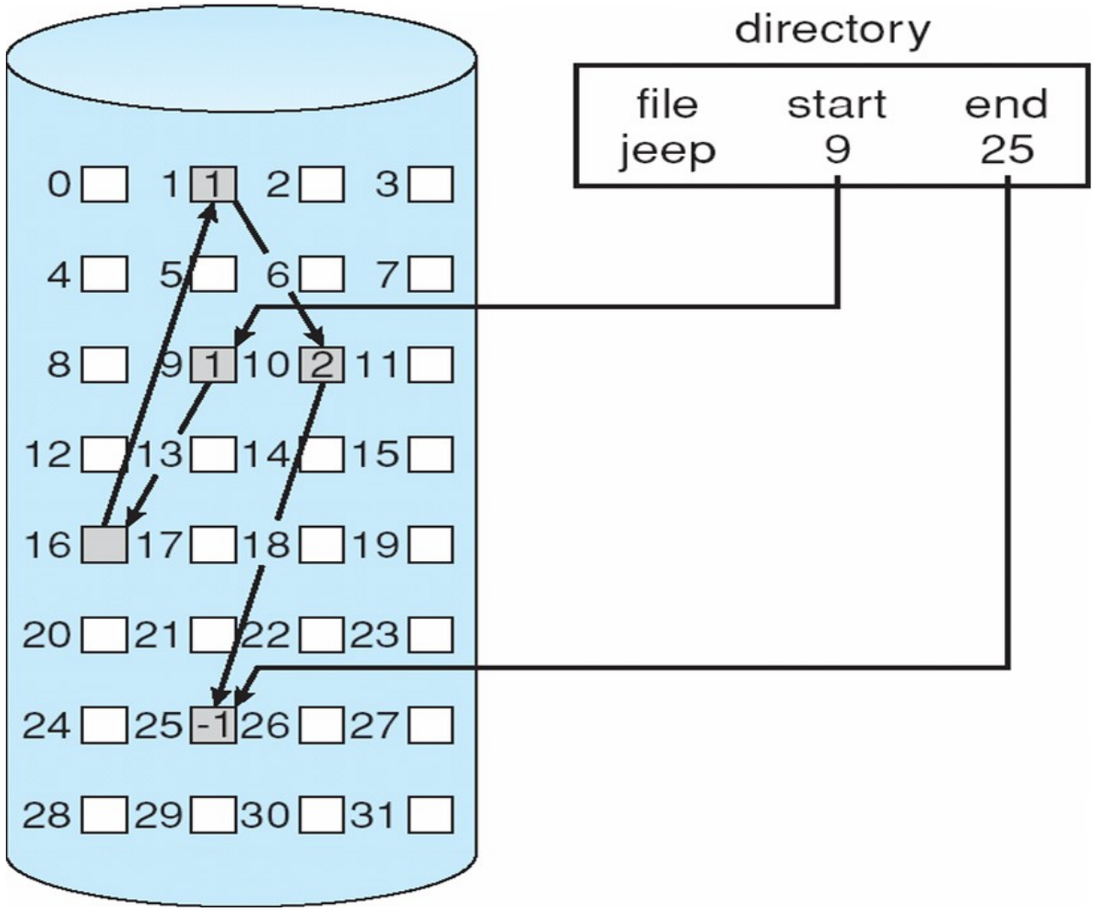
Advantages:

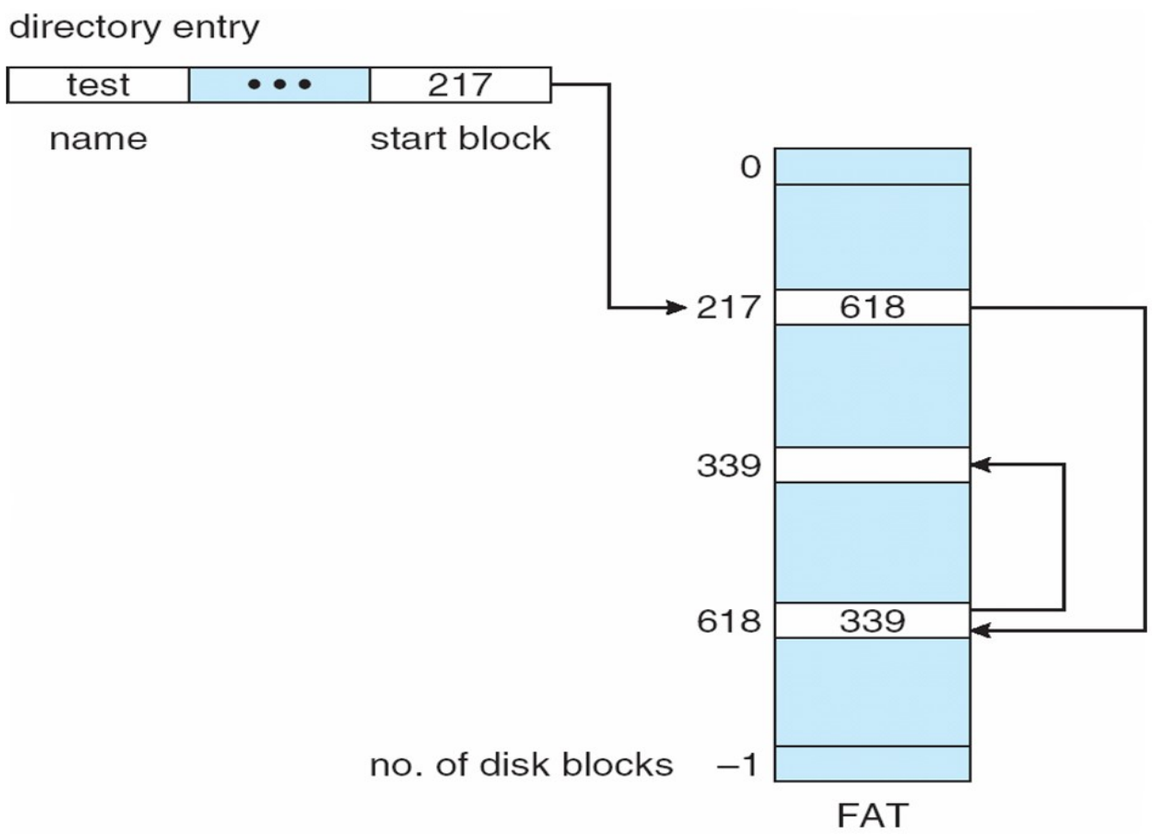
- Simple to implement.
- No need for contiguous space, reducing fragmentation.

Disadvantages:

- Overhead of storing pointers.
- Slower access times due to pointer traversal.

Example Linked Allocation





Indexed Allocation

- **Definition:** Uses an index block to keep track of the blocks allocated to a file.
- **Characteristics:**
  - An index block contains pointers to all other blocks used by the file.
  - Supports both sequential and random access efficiently.
- **Advantages:**
  - No external fragmentation.
  - Efficient access for both sequential and random reads.
- **Disadvantages:**
  - Overhead of index block storage.
  - Limited by the size of the index block.

Example Indexed Allocation

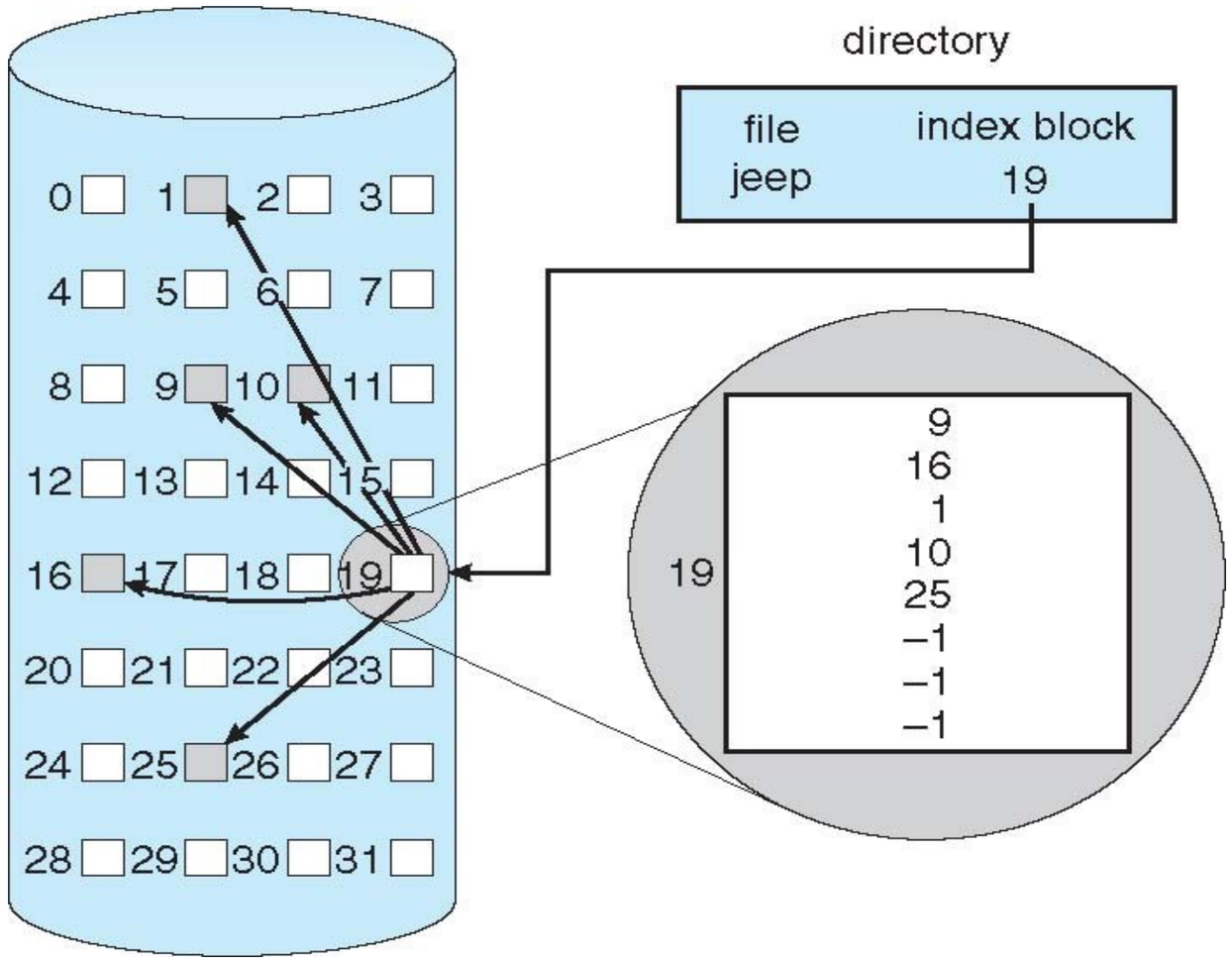


Table of Contents

Disk Scheduling Algorithms

- The operating system is responsible for using hardware efficiently — for the disk drives, this means having a fast access time and disk bandwidth.
- Access time has two major components
  - *Seek time* is the time for the disk are to move the heads to the cylinder containing the desired sector.
  - *Rotational latency* is the additional time waiting for the disk to rotate the desired sector to the disk head.
- 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 Algorithms

- Several algorithms exist to schedule the servicing of disk I/O requests.
- We illustrate them with a request queue (0-199).

98, 183, 37, 122, 14, 124, 65, 67

Head pointer 53

FCFS (First Come First Served)

FCFS algo serves disk I/O requests in the order they arrive as shown in figure.

Advantages

- **Easy** to implement
- [**Fairness**]: Every request has same priority
- Preferred when **simplicity** is crucial despite its potential inefficiency

Disadvantages

- Do not consider underline hardware characteristics
- If requests are far apart we face long waiting times due to high seek time

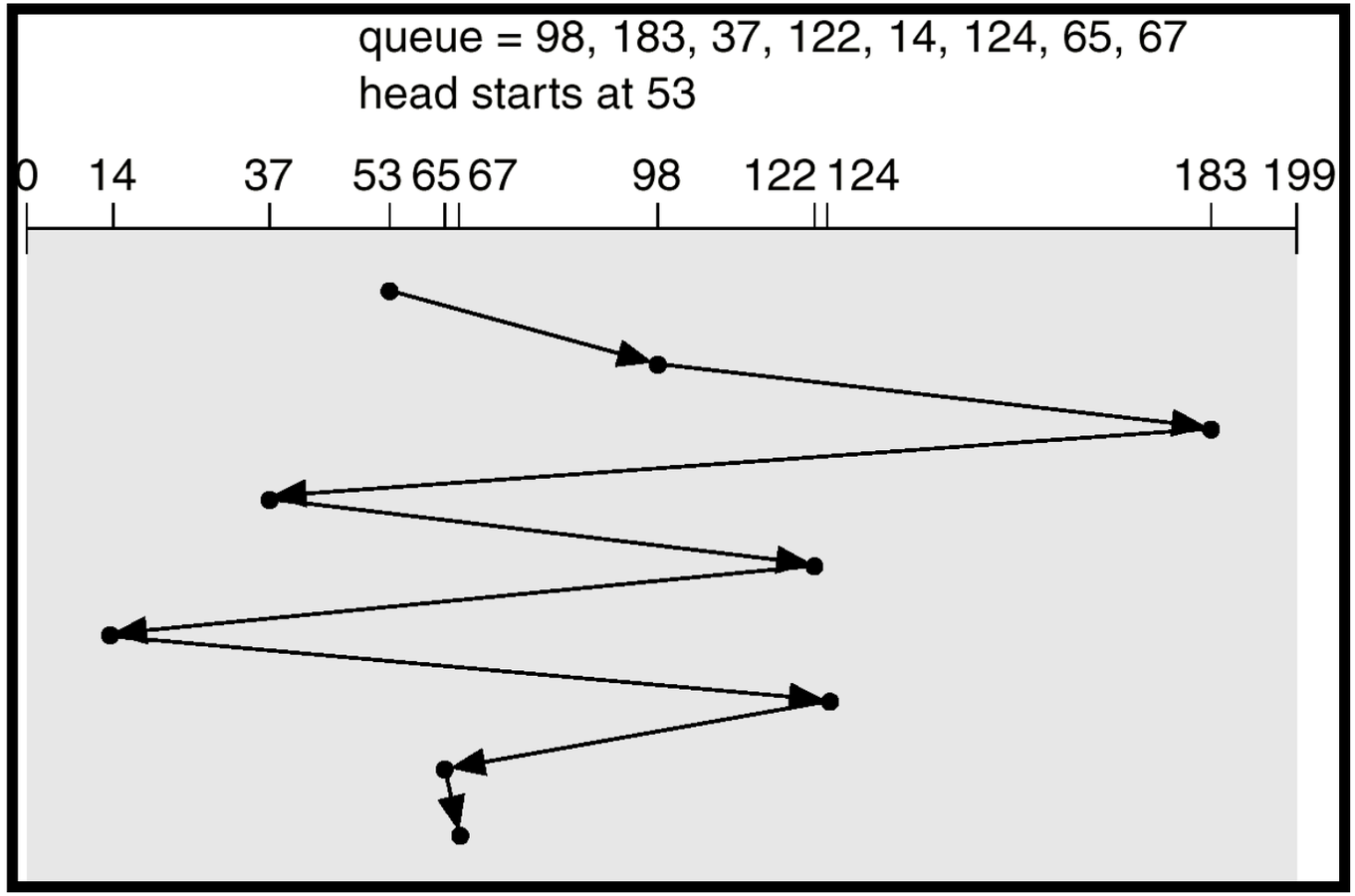


Illustration shows total head movement of 640 cylinders.

SSTF (Shortest Seek time first)

- Pick the request from the queue with the minimum seek time from the current head position as shown in Figure.

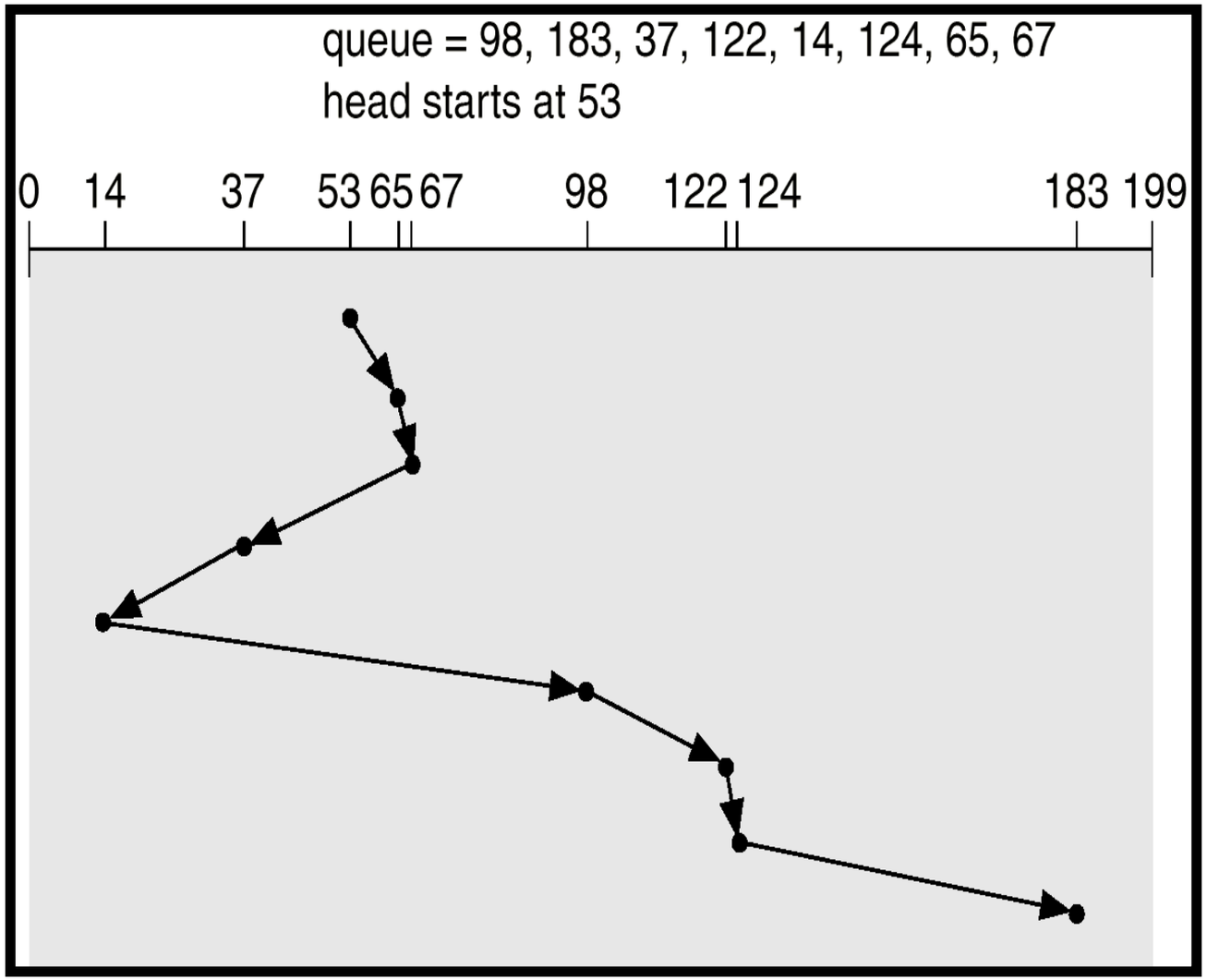
Advantages:

- By definition reduces total seek time
- More efficient than FCFS in minimizing seek time.

Disadvantages:

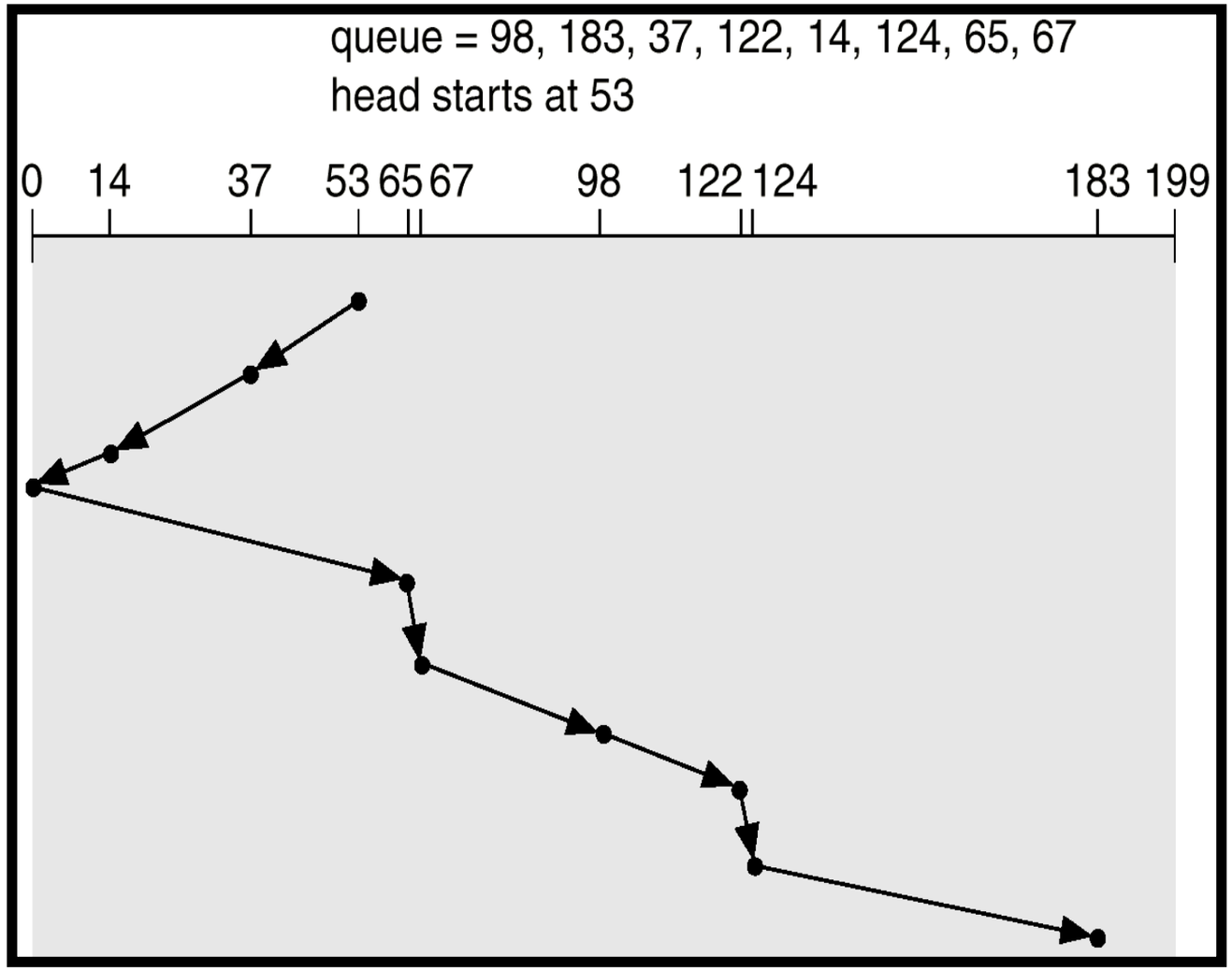
- Can cause starvation and leads to unfairness
- More complex to implement than FCFS.

*[**Starvation in SSFT**]: When certain disk I/O requests may be indefinitely delayed as shorter seek time requests keep on arriving and taking precedence.*



SCAN (Elevator Algorithm)

- 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.



Advantages:

- More efficient than FCFS and SSTF in terms of overall seek time.
- Provides a more uniform wait time.

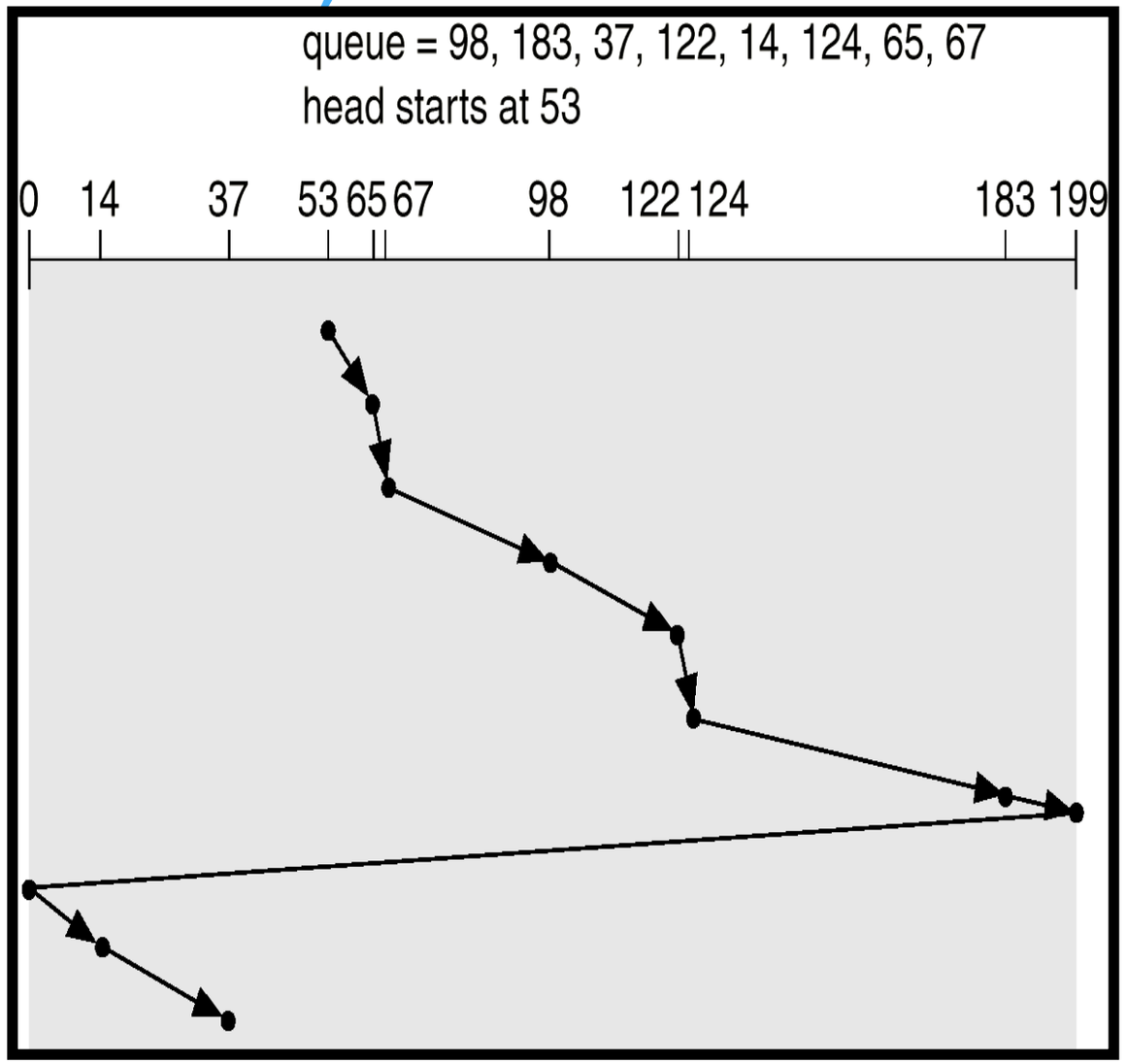
Disadvantages:

- Can still cause long wait times for requests just missed on the sweep.
- More complex to implement than FCFS.

Illustration shows total head movement of 208 cylinders.

C-SCAN (Circular SCAN)

- C-SCAN moves the disk arm towards one end of the disk, servicing requests, and then immediately returns to the beginning without servicing any requests on the return trip.
  - Treats the cylinders as a circular list that wraps around from the last cylinder to the first one.



Advantages :

- Provides a more uniform wait time compared to SCAN.
- Reduces the likelihood of starvation compared to SSTF.

Disadvantages :

- Can be less efficient than SCAN due to the not serviced return trip.
- More complex to implement than FCFS and SSTF

C-LOOK (Circular LOOK)

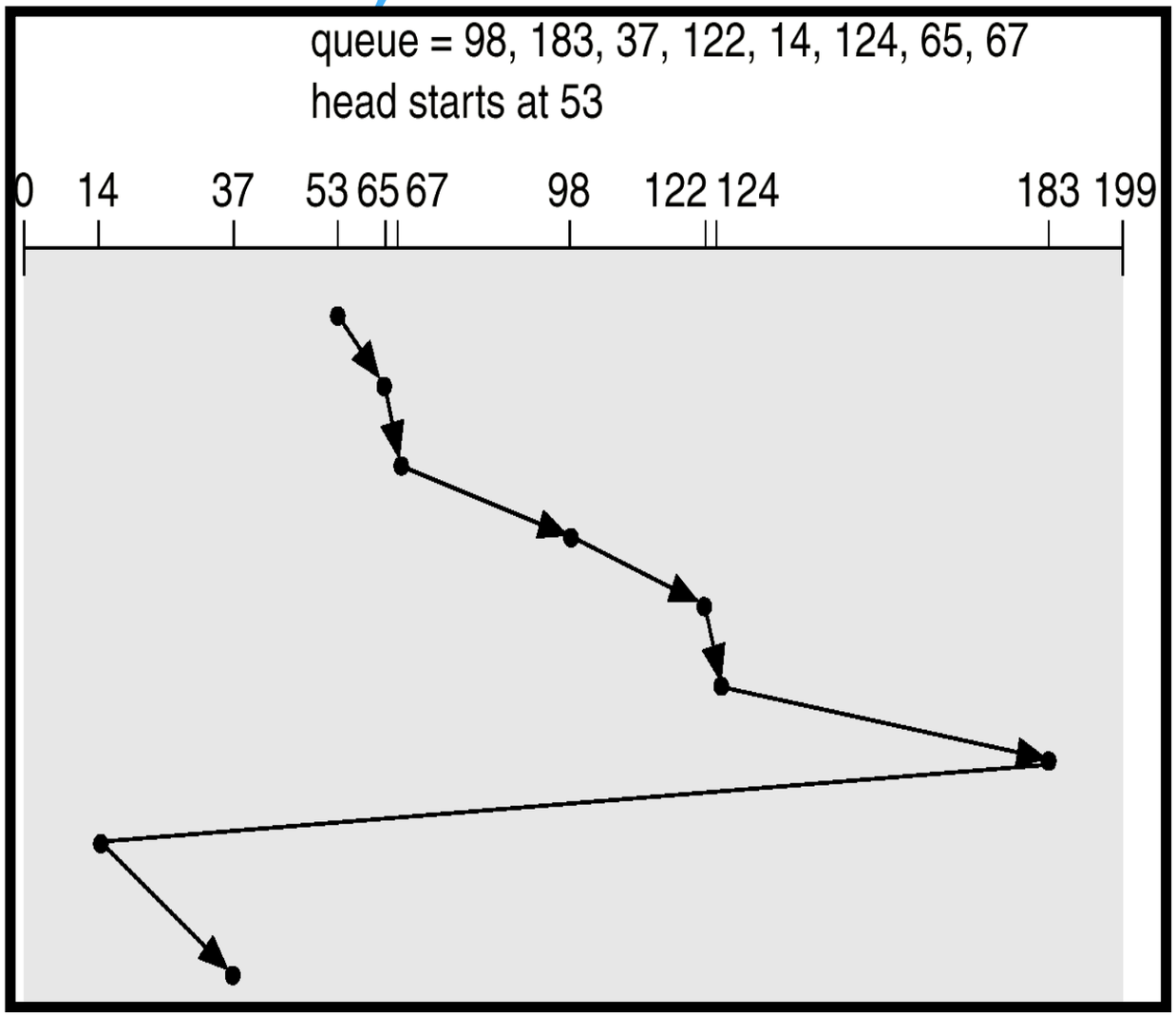
C-LOOK is similar to C-SCAN but only goes as far as the last request in each direction before reversing.

Advantages:

- More efficient than C-SCAN as it avoids unnecessary traversal.
- Provides a uniform wait time similar to C-SCAN.

Disadvantages :

- Complex to implement.
- May have slightly longer seek times than SSTF for some requests.



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.
- Performance depends on the number and types of requests.
- Requests for disk service can be influenced by the file-allocation method.
- 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.

Reference Material

- Crowley, Charles. Operating systems: a design-oriented approach. McGraw-Hill Professional, 1996.
- Arpaci-Dusseau, Remzi H., and Andrea C. Arpaci-Dusseau. Operating systems: Three easy pieces. Arpaci-Dusseau Books, LLC, 2018.
- Silberschatz, A., Galvin, P. B., & Gagne, G. (2006). Operating System Concepts, Windows XP update. John Wiley & Sons.

UPES
UNIVERSITY OF TOMORROW