Dr. Vishwanath Karad
MIT WORLD PEACE UNIVERSITY | PUNE
TECHNOLOGY, RESEARCH, SOCIAL INNOVATION & PARTNERSHIPS

Unit V File Management

DEPARTMENT OF COMPUTER SCIENCE AND ENGINEERING

Syllabus

- Concept of File, Access methods, File types, File operation, Directory structure, File System structure,
- Allocation methods (contiguous, linked, indexed)
- Disk Management: Disk structure, Disk scheduling - FCFS, SSTF, SCAN, C-SCAN, Disk reliability,
- Disk formatting, Boot-block, Bad blocks

File Management

Files are the central element to most applications
Desirable properties of files:

- **Long-term existence:** Files are stored on disk or other secondary storage and do not disappear when a user logs off.
- **Sharable between processes**: Files have names and can have associated access permissions that permit controlled sharing.
- **Structure**: Depending on the file system, a file can have an internal structure that is convenient for particular applications. In addition, files can be organized into hierarchical or more complex structure to reflect the relationships among files.

Terms in common use when discussing files

1. Fields

- Basic element of data
- Contains a single value
- Characterized by its length and data type
- E.g. Student name

2. Records

- Collection of related fields
- Treated as a unit
- Fixed /variable length

3. File

- Is a collection of similar records
- Treated as a single entity and Have file names
- May implement access control mechanisms

4. Database

- Collection of related data
- Relationships exist among elements
- Consists of one or more files

File System

- The File System is one of the most important part of the OS to a user
- Concerned with secondary storage
- File systems also provide functions which can be performed on files:
  - Create
  - Delete
  - Open
  - Close
  - Read
  - Write

File Organization

- Refers to the **logical structuring** of records
- Determined by the ***way*** in which files are accessed
- Important criteria while choosing a file organization:
  - Short access time
  - Ease of update
  - Economy of storage
  - Simple maintenance
  - Reliability

File Organization

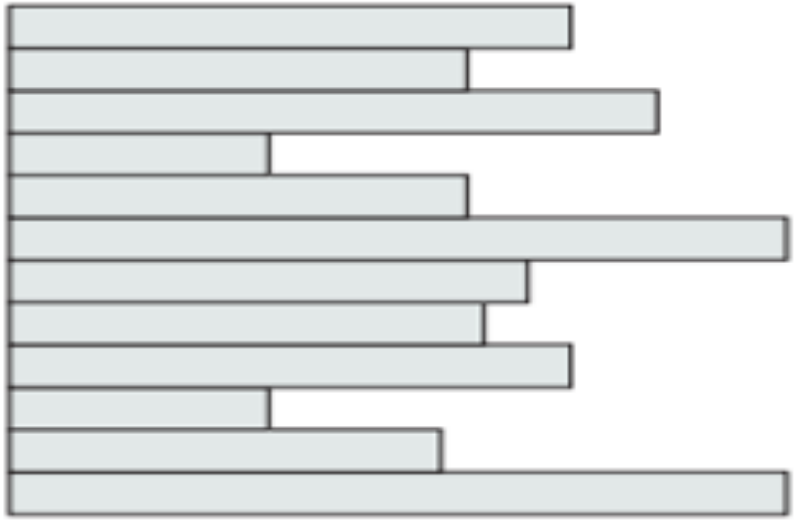
File Organization Types

- Pile
- Sequential file
- Indexed Sequential file
- Indexed file
- Direct or hashed file

File Organization

Pile

- Data are collected in the order they arrive
- No structure
- Purpose is to accumulate a mass of data and save it
- Records may have different fields
- Record access is by exhaustive search
- Useful when the stored data vary in size and structure.
- Easy to update

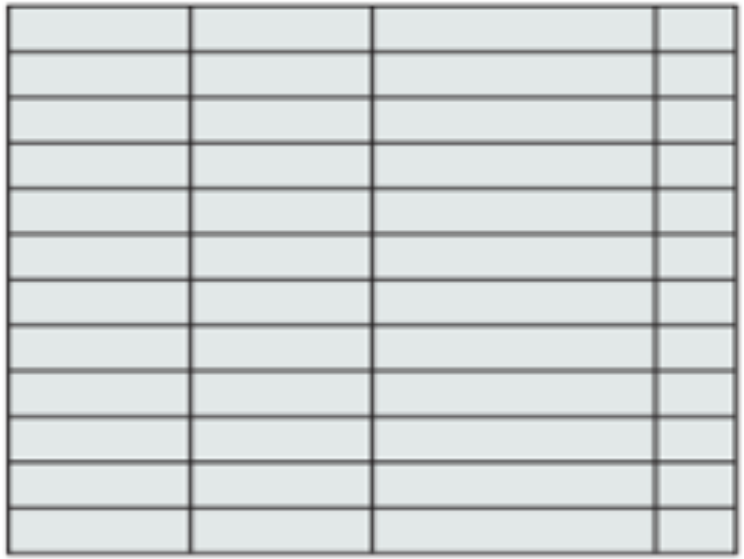


Variable-length records
Variable set of fields
Chronological order

(a) Pile file

Sequential File

- Fixed format used for records
- Records are the same length
- Key field (first field)
  - Uniquely identifies the record
  - Records are stored in key sequence
- Used in batch applications such as billing/payroll
- Easy to store on tape and disk
- Not suitable for interactive applications



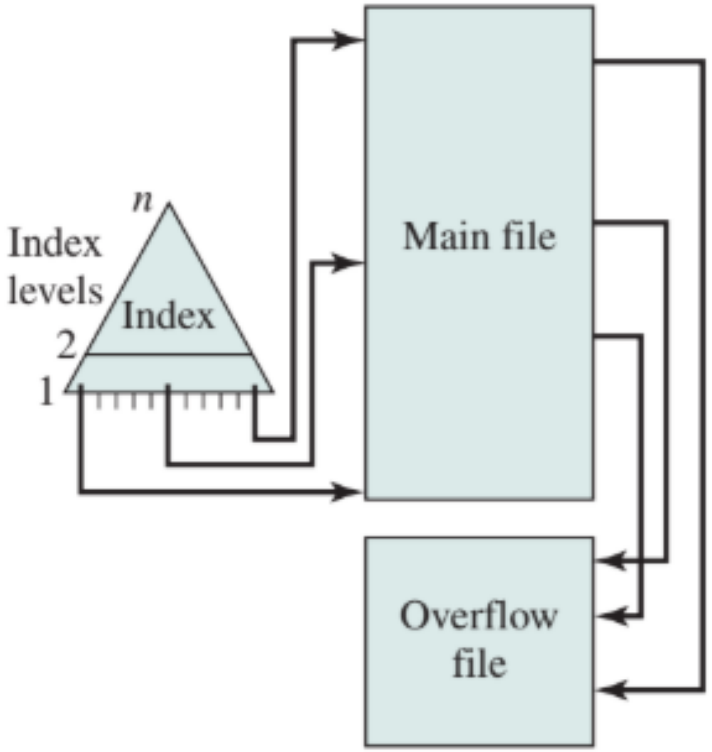
Fixed-length records
Fixed set of fields in fixed order
Sequential order based on key field

(b) Sequential file

File Organization

Indexed Sequential File

- Maintains the key characteristic of the sequential file: records are organized in sequence based on a key field
- An **index** is added to the file to support random access and **overflow file** is also added
- Limitation is processing is limited as it is based on a single field of the file.



(c) Indexed sequential file

Indexed Sequential File Example

Imagine you have a **very long list** of names (1 million students) arranged in **alphabetical order**.
If you want to find “Ranjana,” you’d have to **go through hundreds of thousands of names**
Instead of searching the whole list directly, we create a smaller list called an index , like a table of contents in a book.

•The index stores only some key entries (say, every 1000th name) along with a pointer showing where that section starts in the main file.

Example index- Key A : points to names starting with A

1. First, look in the **index** to find which section target belongs to (e.g., "Ranjana" is under “R”).
2. Use the **pointer** from the index to jump directly to that part of the **main file**.
3. Continue the search there, but now you only search **a small part** instead of the whole file.

Example with numbers:

•Without an index:

You might check **500,000 records** on average before finding the one you want.

•With an index of **1,000 entries**:

- You first check ~**500 entries** in the index.
- Then only ~**500 more records** in the main file.

Indexed File

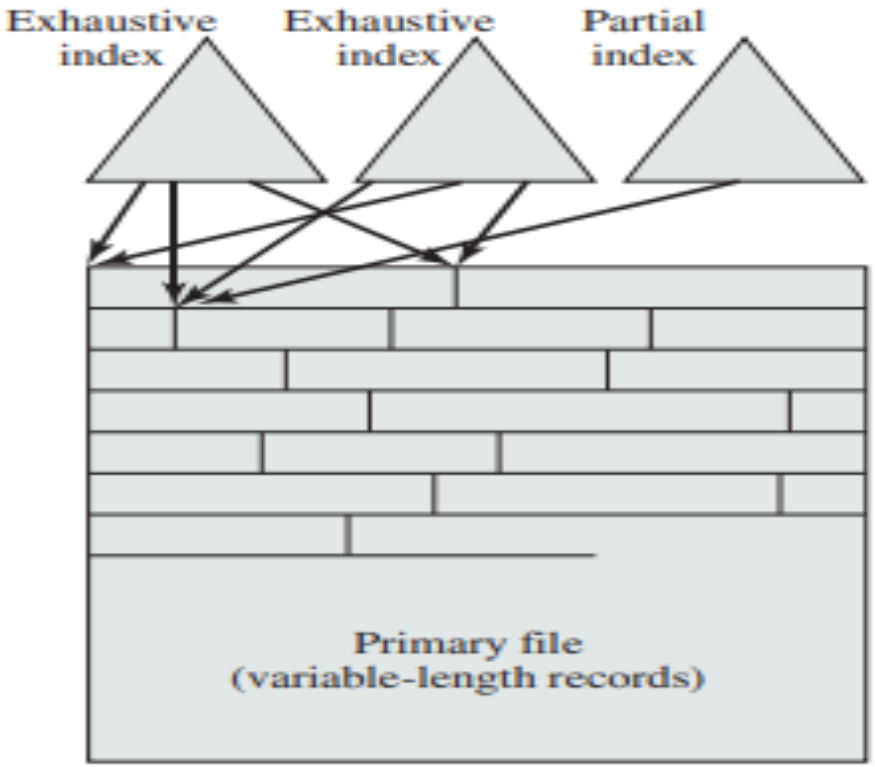
•Multiple indexes are employed for a field.

•Variable length records can be employed

• Two types of indexes are used: **exhaustive index** (one entry per record) **or partial index**(one per block or group).

•When new records are inserted, all **the index files must be updated**.

•Used in airline reservation systems

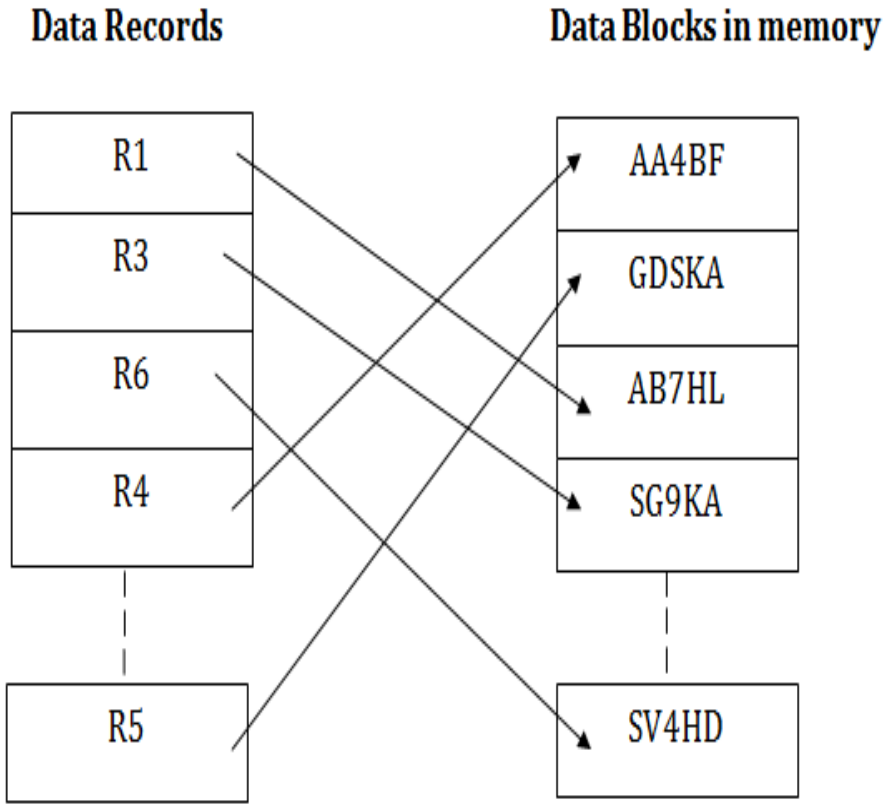


(d) Indexed file

File Organization

Direct or Hash File

- Directly access a block at a known address
- Key field required for each record
- But there is no concept of sequential ordering
- Makes use of hashing on the key value
- Used where rapid access is required such as Directories



File Directories

- Contains information about files
  - Attributes
  - Location
  - Ownership
- Provides mapping between file names and the files themselves
- A directory system should support a number of **operations** including:
  - Search
  - Create files
  - Deleting files
  - Listing directory
  - Updating directory

1. Basic Information

- **File Name-** unique
- **File Type-** text,binary
- **File Organization**

2. Address Information

- **Volume -**Indicates device on which file is stored
- **Starting Address-** on secondary storage
- **Size Used -**Current size of the file in bytes, words, or blocks
- **Size Allocated -** Maximum size of the file

3. Access Control Information

- **Owner :**Owner has control of this file & able to grant/deny access to other users and to change these privileges.
- **Access Permission:** includes the user's name and password for each authorized user.
- **Permitted Actions :**Controls reading, writing, executing, and transmitting over a network

4. Usage Information

- **Date Created**
- **Identity of Creator**
- **Date Last Read Access**
- **Identity of Last Reader**
- **Date Last Modified**
- **Identity of Last Modifier**
- **Date of Last Backup**
- **Current Usage**

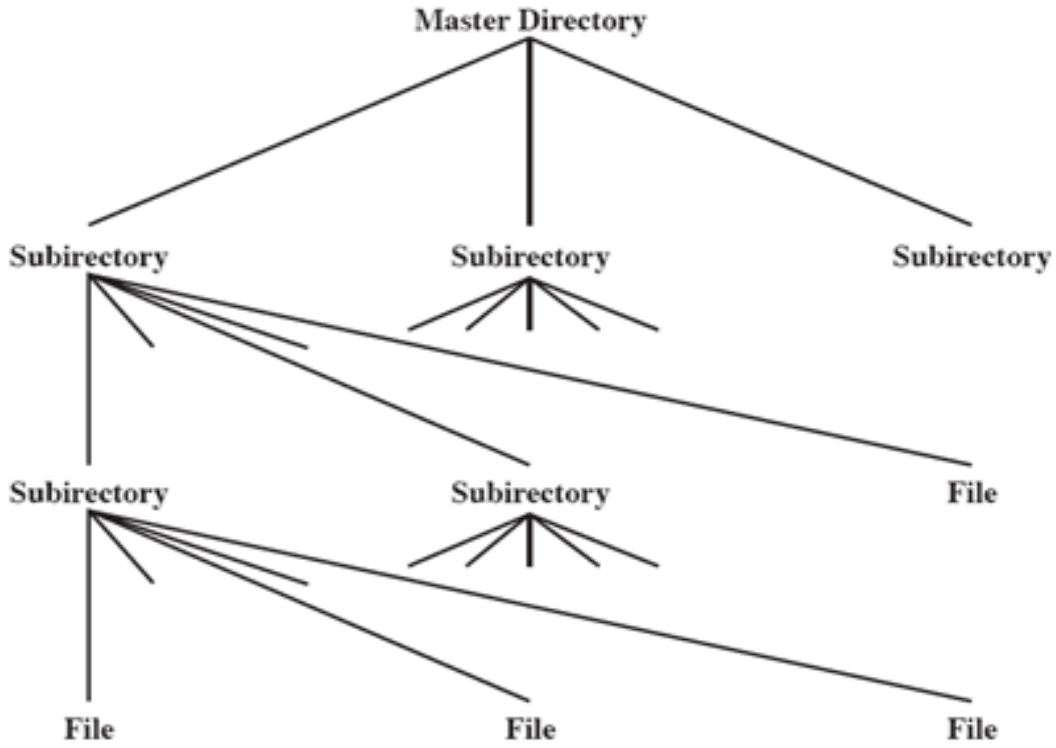
$ chmod 755 your-script-name

The chmod command modifies the permissions of a file or directory on a Linux system. ... The numbers 755 assign **read-write-execute permissions to the user owner and read-execute permissions to group owner and others**.

- **The sums of these numbers give combinations of these permissions:**
- 0 = no permissions whatsoever; this person cannot read, write, or execute the file.
- 1 = execute only.
- 2 = write only.
- 3 = write and execute (1+2)
- 4 = read only.
- 5 = read and execute (4+1)
- 6 = read and write (4+2)
- 7 = read and write and execute (4+2+1)

Hierarchical or Tree-Structured Directory

- Master directory with user directories underneath it
- Each user directory may have subdirectories and files as entries
- **Naming**
  - Users need to be able to refer to a file by name
  - Files need to be named uniquely
  - Tree structure allows users to find a file by following the directory path
  - Duplicate filenames are possible if they have different pathnames



File Sharing

- In multiuser system, allow files to be shared among users
- Two issues
  - Access rights
  - Management of simultaneous access

-User may lock entire file when it is to be updated

-User may lock the individual records during the update

-Mutual exclusion and deadlock are issues for shared access

User Classes

- Owner - Usually the files creator, has full rights
- User Groups - A set of users identified as a group
- Others

File Allocation Methods – Contiguous, Linked, and Indexed”

Definition:

File allocation methods determine how disk blocks are assigned to files.

Purpose:
Efficiently manage storage and access files quickly.

Types:

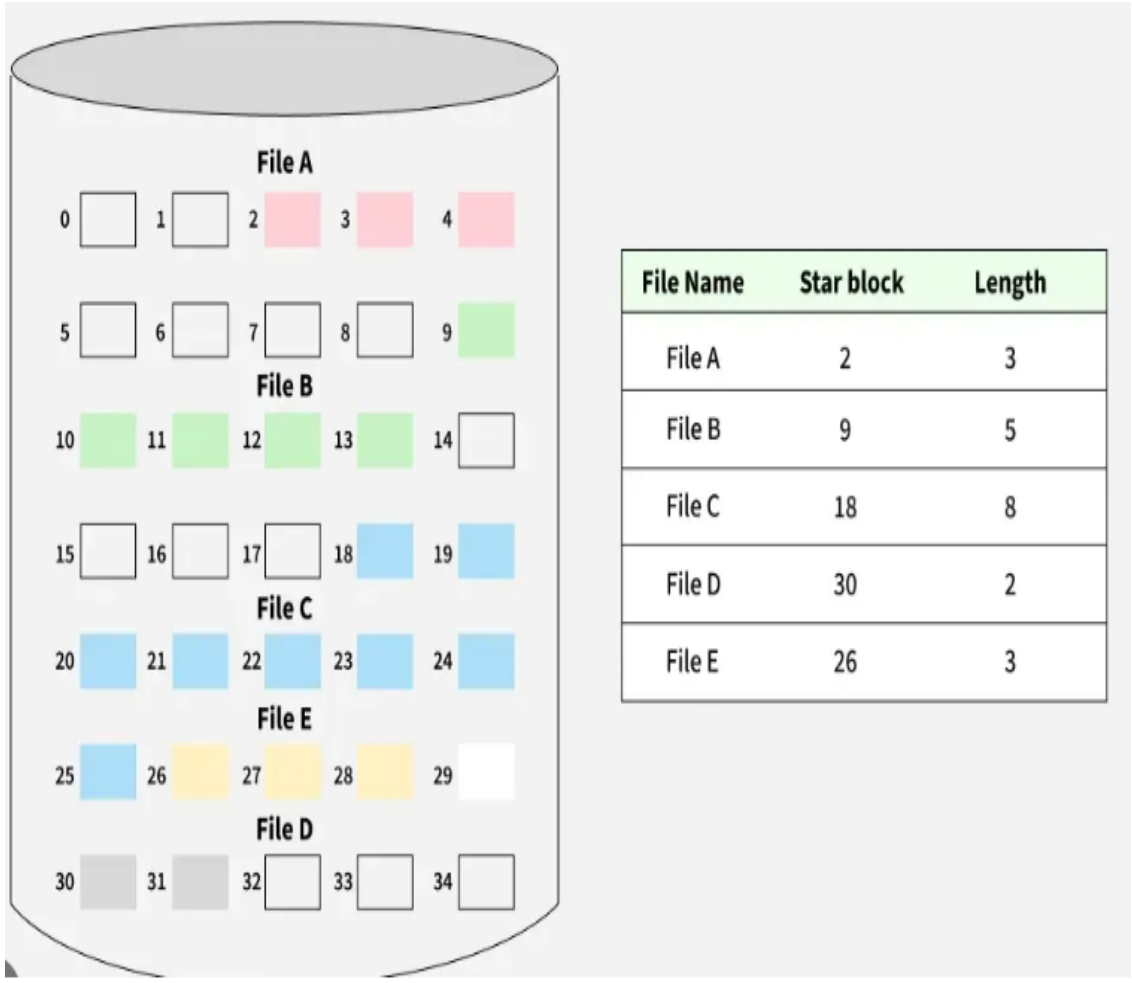
1. Contiguous Allocation
2. Linked Allocation
3. Indexed Allocation

File Allocation Methods – Contiguous,

- An allocation method refers to how disk blocks are allocated for files:
- Each file occupies set of contiguous blocks
  - Best performance in most cases
  - Simple – only starting location (block #) and length (number of blocks) are required
  - Problems include:
    - Finding space on the disk for a file,
    - Knowing file size,
    - External fragmentation, need for **compaction off-line** (**downtime**) or **on-line**

Contiguous Allocation

Each file occupies a set of contiguous (adjacent) blocks on the disk.



File Name	Star block	Length
File A	2	3
File B	9	5
File C	18	8
File D	30	2
File E	26	3

Advantages:

- Fast sequential and direct access
- Simple to implement

Disadvantages:

- External fragmentation

- Difficult to grow files dynamically

Linked Allocation

Each file is a linked list of disk blocks. Each block contains a pointer to the next block.

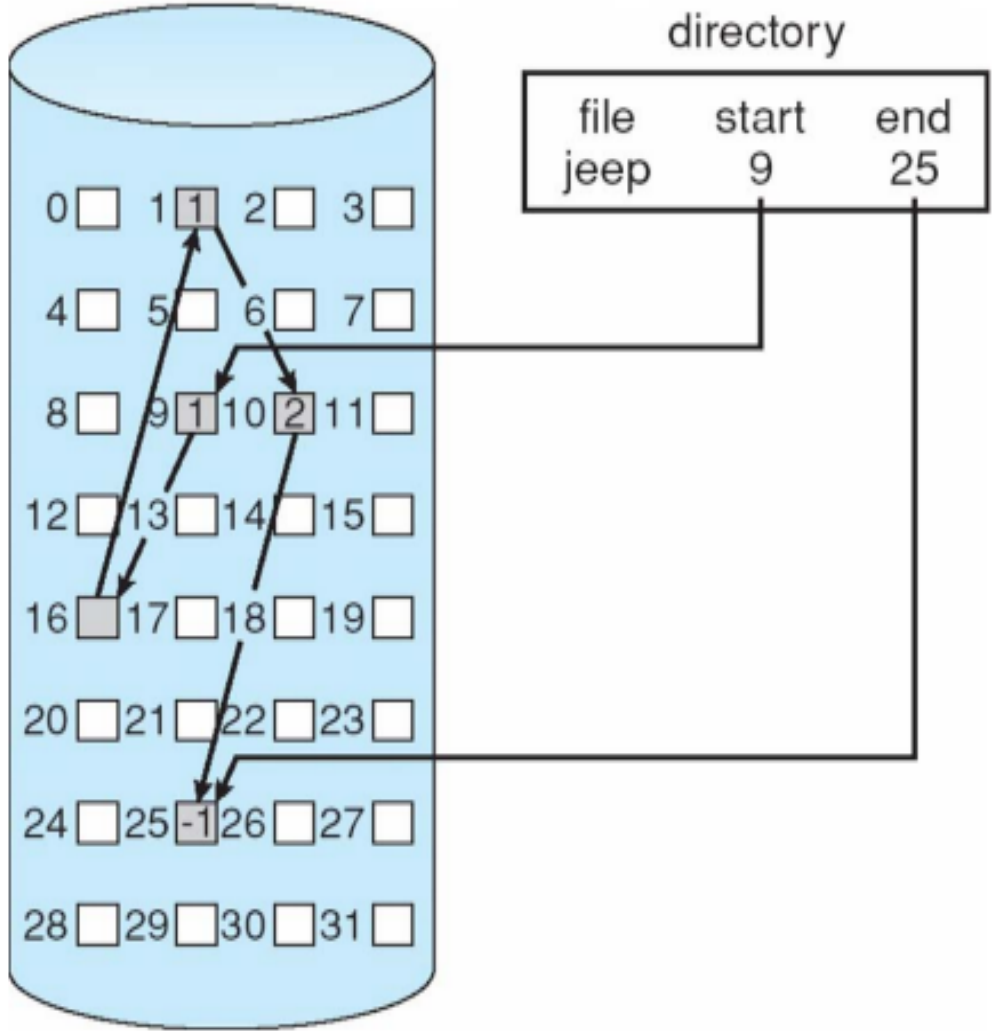
Advantages:

- No external fragmentation
- Easy to grow file size

Disadvantages:

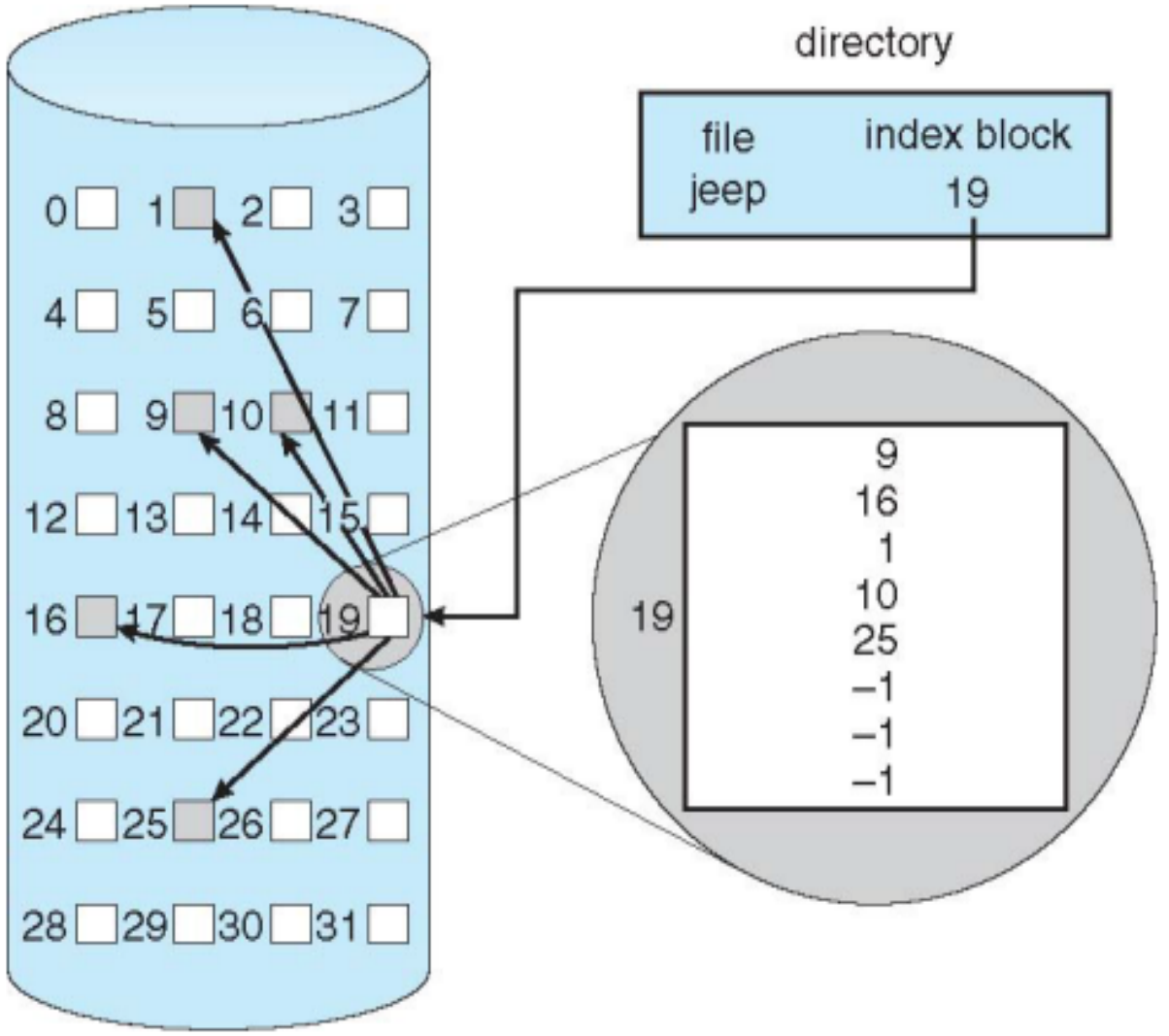
No direct access (only sequential)

- Pointer overhead
- Possible reliability issue if pointer is lost



Indexed Allocation

Each file has an **index block** that contains pointers to all its data blocks.



Advantages:

- Supports both sequential and direct acc
- No external fragmentation

Disadvantages:

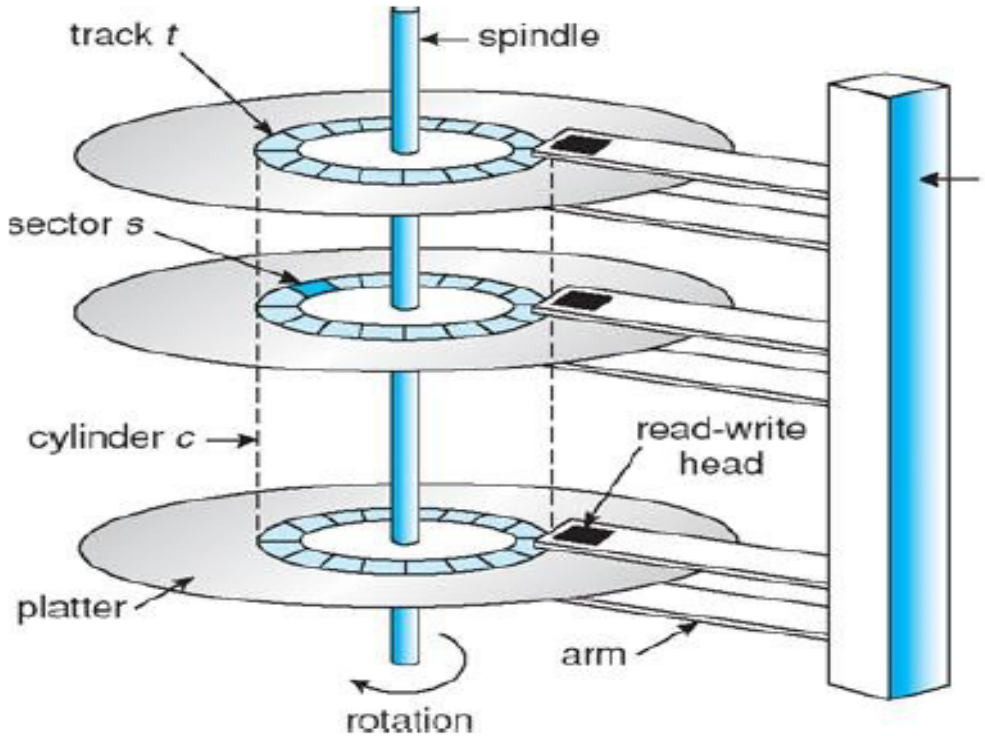
- Overhead of index block

- Small files may waste space

Disk Structure

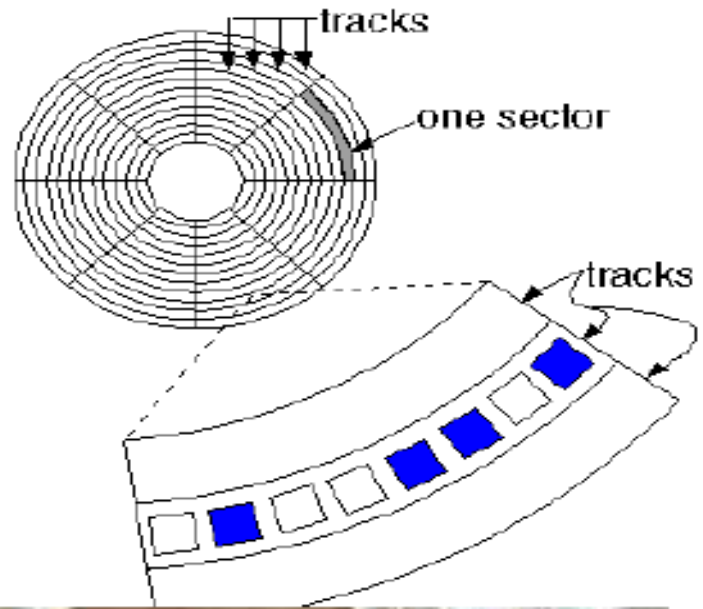
- Disk can be subdivided into **partitions**
- Disks or partitions can be **RAID** protected against failure
- Disk or partition can be used **raw** – without a file system, or **formatted** with a file system
- Partitions also known as minidisks, slices
- Entity containing file system is known as a **volume**
- Each volume containing a file system also tracks that file system's info in **device directory** or **volume table of contents**
- In addition to **general-purpose file systems** there are many **special-purpose file systems**, frequently all within the same operating system or computer

Magnetic Disks



Structure of a magnetic disk

Tracks and Sectors in a platter



Disk Structure

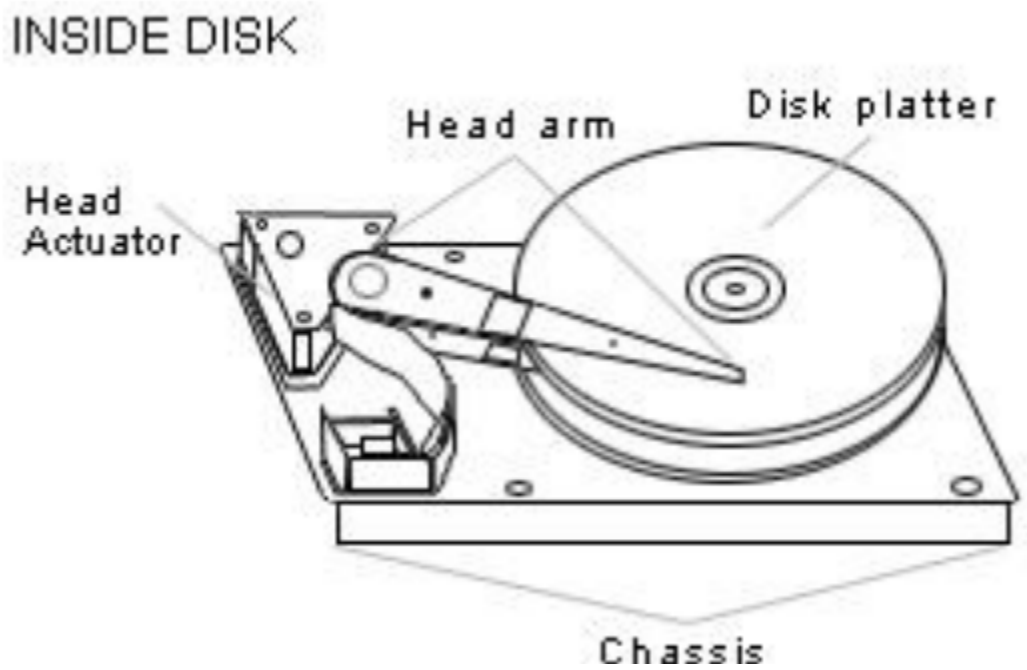
- Modern disk drives are addressed as large one-dimensional arrays of where the logical block is the smallest unit of transfer.
- The size of a logical block is usually 512 bytes, although some disks can be to have a different logical block size, such as 1,024 bytes.
- The one-dimensional array of logical blocks is mapped onto the sectors of the disk sequentially.
- Sector 0 is the first sector of the first track on the outermost cylinder.
- The mapping proceeds in order through that track, then through the rest of the tracks in that cylinder, and then through the rest of the cylinders from outermost to innermost.
- By using this mapping, we can -at least in theory-convert a logical block number into an old-style disk address that consists of a cylinder number, a track number within that cylinder, and a sector number within that track.

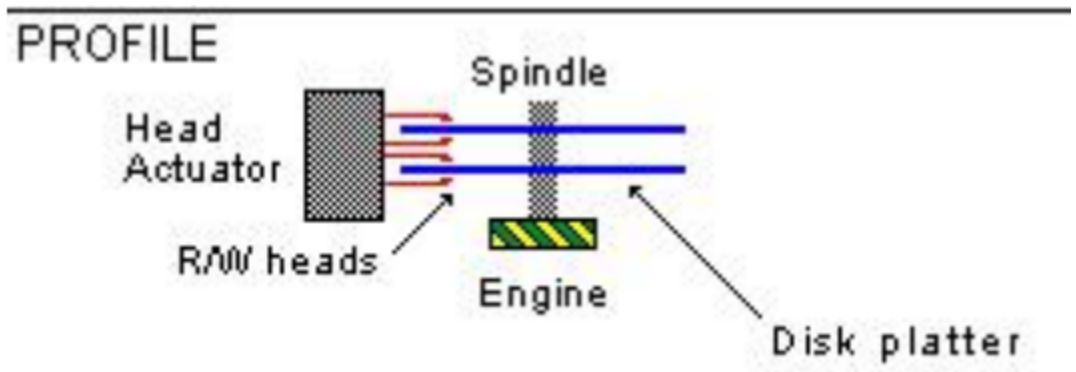
Disk Structure

- First, most disks have some defective sectors, but the mapping hides this by substituting spare sectors from elsewhere on the disk.
- Second, the number of sectors per track is not a constant on same drives.
- On media that use the density of bits per track is uniform.
- The further a track is from the center of the disk, the greater its length, so the more sectors it can hold.

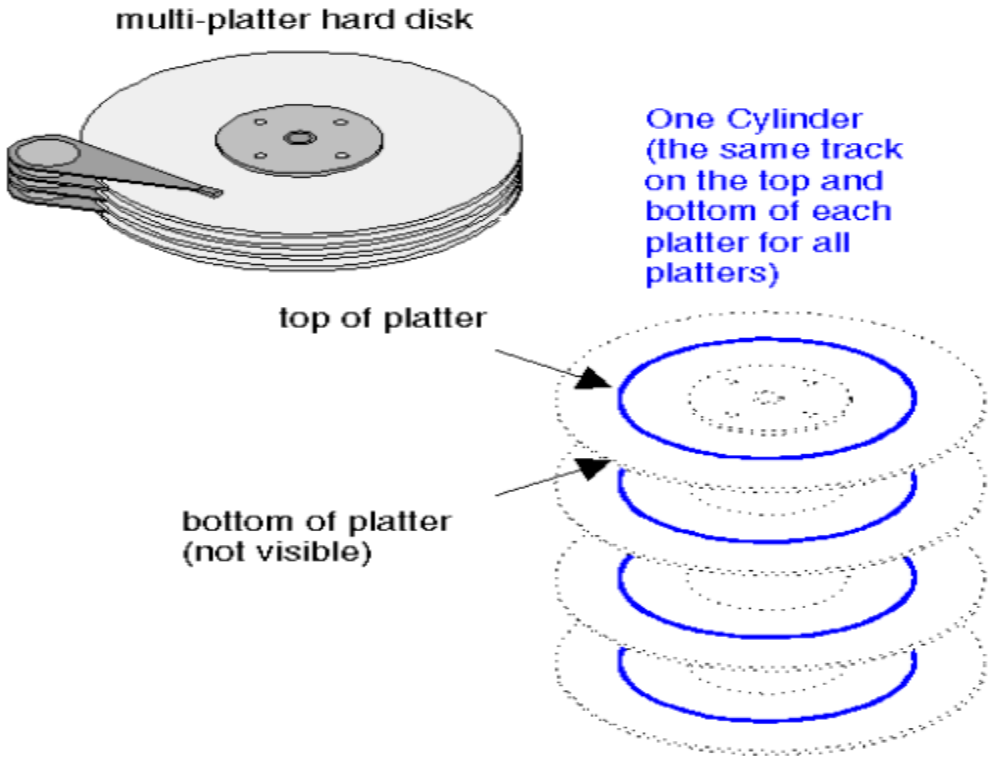
Disk Structure

- When the disk drive is operating, the disk is rotating at constant speed
- Positioning the Read/Write Head
- Track selection involves moving the head to a specific track





- **Each surface is divided into tracks (and sectors) in the same way**. This means that when the head for one surface is on a track, the heads for the other surfaces are also on the corresponding tracks. All the corresponding tracks taken together are called a cylinder.



Cylinder

The cylinder is the aggregate of the same track number on every platter used for recording.

Disk Scheduling

- A process needs two type of time, **CPU time and IO time**.
- For I/O, it requests the Operating system to access the disk.
- However, the operating system must be fare enough to satisfy each request and at the same time, operating system must maintain the efficiency and speed of process execution.
- The technique that operating system uses to determine the request which is to be satisfied next is called Disk scheduling.

Disk Scheduling

- **Seek Time:** is the time taken in locating the disk arm to a specified track where the read/write request will be satisfied.
- **Rotational Latency:** It is the time taken by the desired sector to rotate itself to the position from where it can access the R/W heads.
- **Transfer Time:** It is the time taken to transfer the data.
- **Disk Access Time:** Disk access time is given as,

Disk Access Time = Rotational Latency + Seek Time + Transfer Time

- **Disk Response Time:**It is the average of time spent by each request waiting for the IO operation.

Disk Scheduling Algorithms

- Each algorithm is carrying some advantages and disadvantages. The limitation of each algorithm leads to the evolution of a new algorithm.
- FCFS scheduling algorithm
- SSTF (shortest seek time first) algorithm
- SCAN scheduling
- C-SCAN scheduling

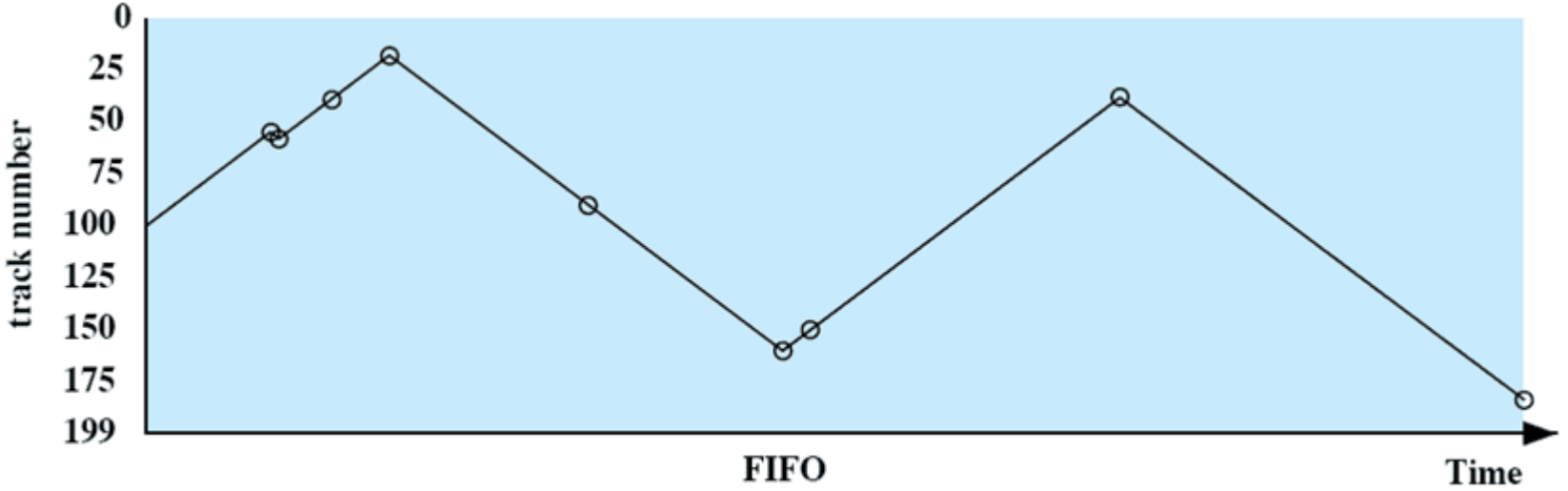
Disk Scheduling Policies

- To compare various schemes, consider a disk head is initially located at track 100.
  - assume a disk with 200 tracks and that the disk request queue has random requests in it

- The requested tracks, in the order received by the disk scheduler, are
  - 55, 58, 39, 18, 90, 160, 150, 38, 184.

First-in, first-out (FIFO)

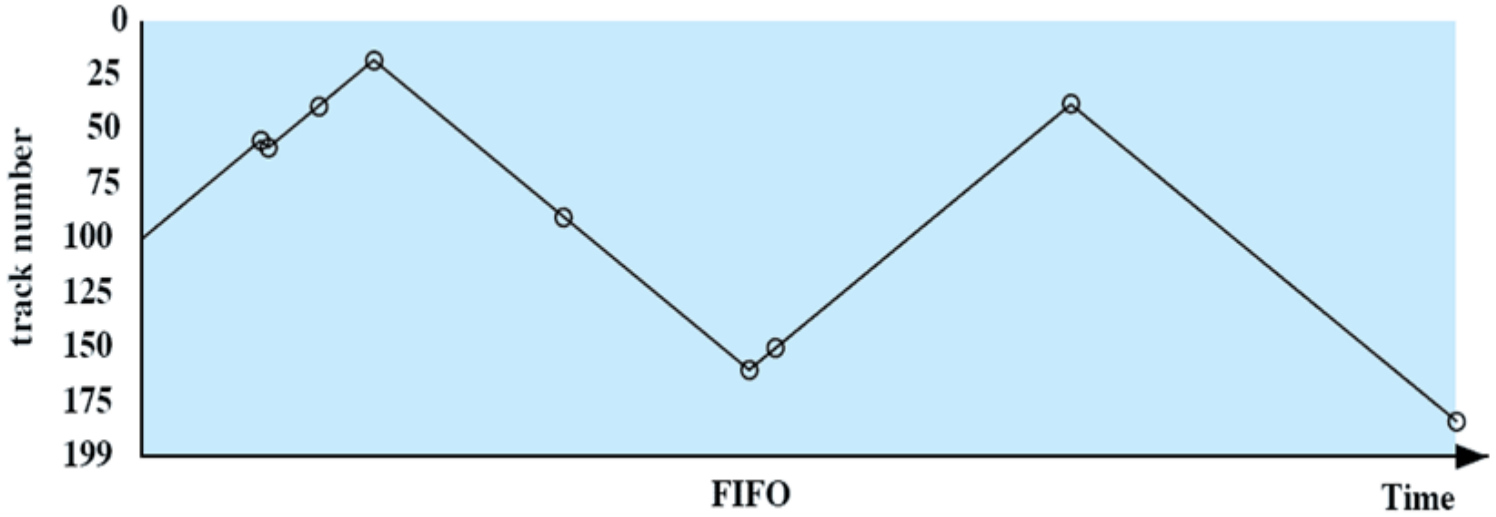
- Processes requests sequentially in the order received
- Fair to all processes
- Approaches random scheduling in performance
- The requested tracks, in the order received by the disk scheduler, are
  - 55, 58, 39, 18, 90, 160, 150, 38, 184.



First-in, first-out (FIFO)

(a) FIFO
(starting at track 100)

Next track accessed	Number of tracks traversed
55	45
58	3
39	19
18	21
90	72
160	70
150	10
38	112
184	146
Average seek length	55.3



Number of tracks traversed by the head

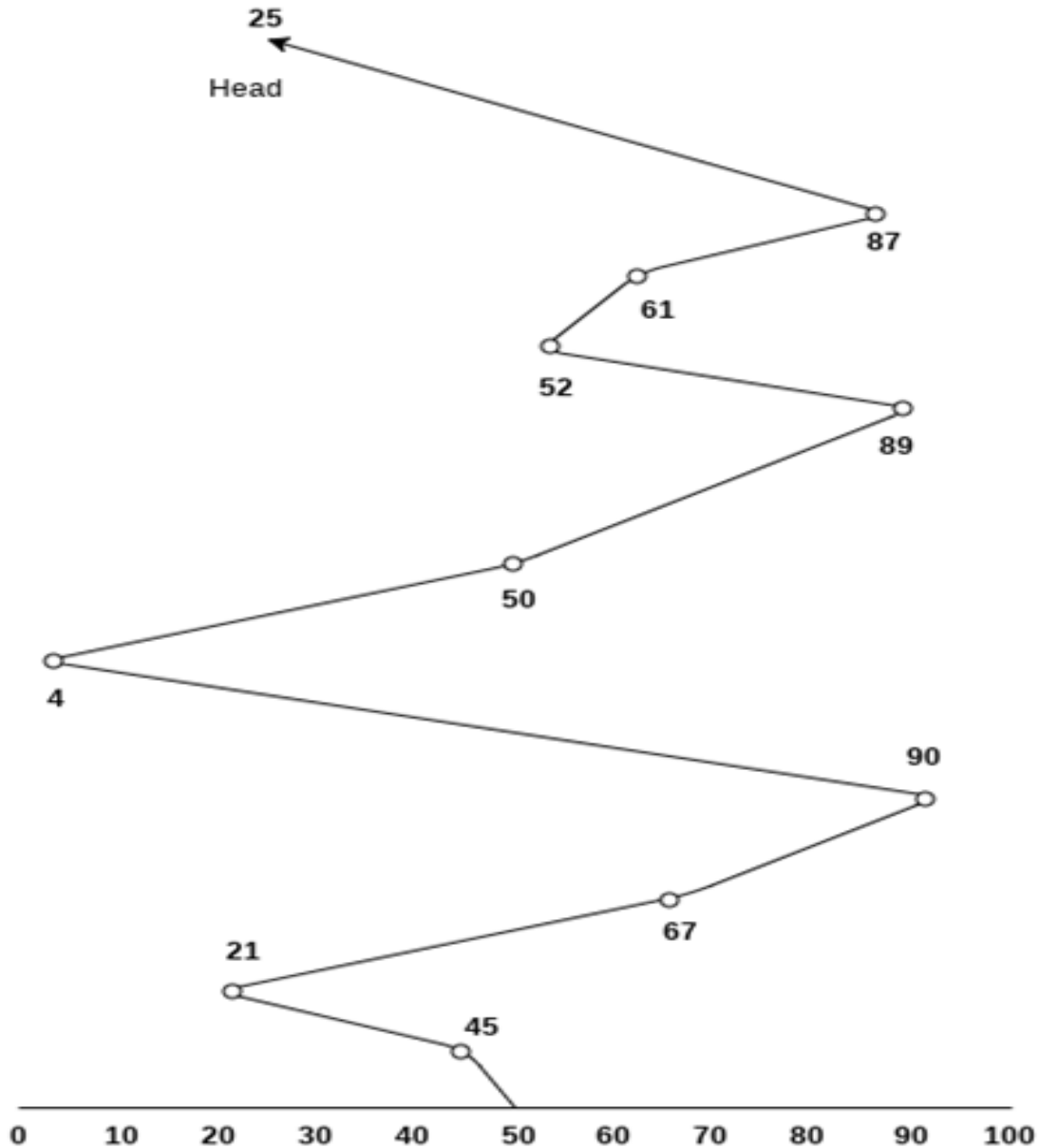
= |55-100|+ |58-55|+|39-58|+|18-39|+|90-18|+|160-90|+|150-160|+|38-150|+|184-38|

= 498

Average seek length= 498/9=55.3

FCFS

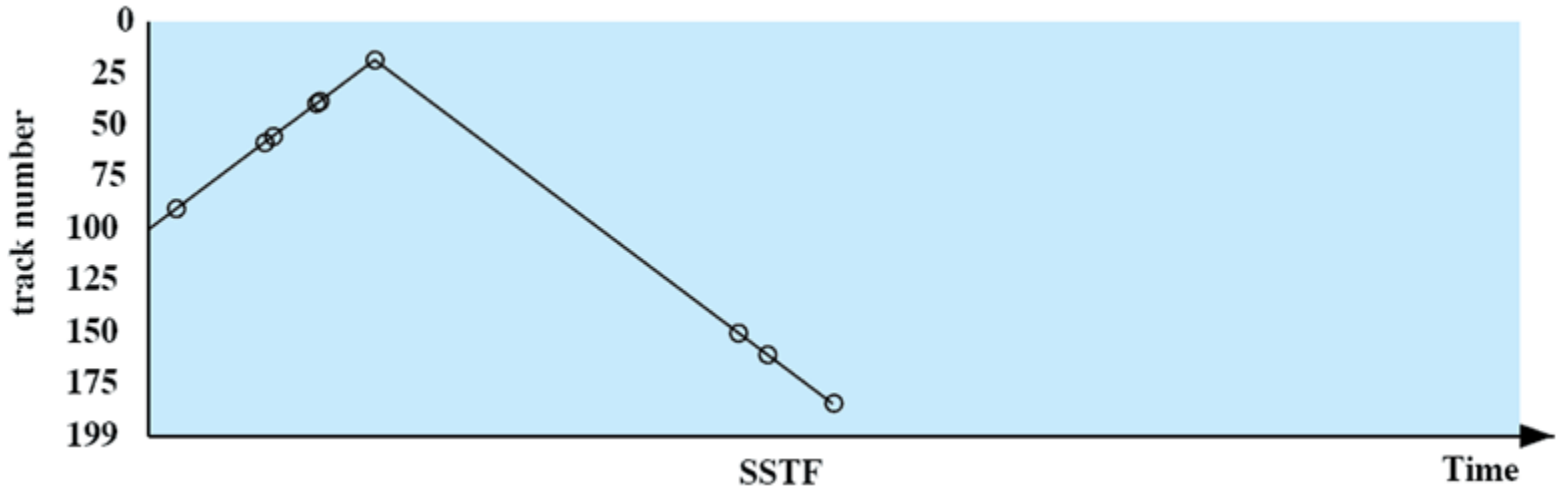
- Consider the following disk request sequence for a disk with 100 tracks 45, 21, 67, 90, 4, 50, 89, 52, 61, 87, 25
- Head pointer starting at 50 and moving in left direction. Find the number of head movements in cylinders using FCFS scheduling.


25
Head
87
61
52
89
50
4
90
67
21
45
0
10
20
30
40
50
60
70
80
90
100

- Number of cylinders moved by the head
- = (50-45)+(45-21)+(67-21)+(90-67)+(90-4)+(50-4)+(89-50)+(89-52)+(61-52)+(87-61)+(87-25)
- = 5 + 24 + 46 + 23 + 86 + 46 + 39 + 37+9 + 26 + 62
- = 403

Shortest Service Time First

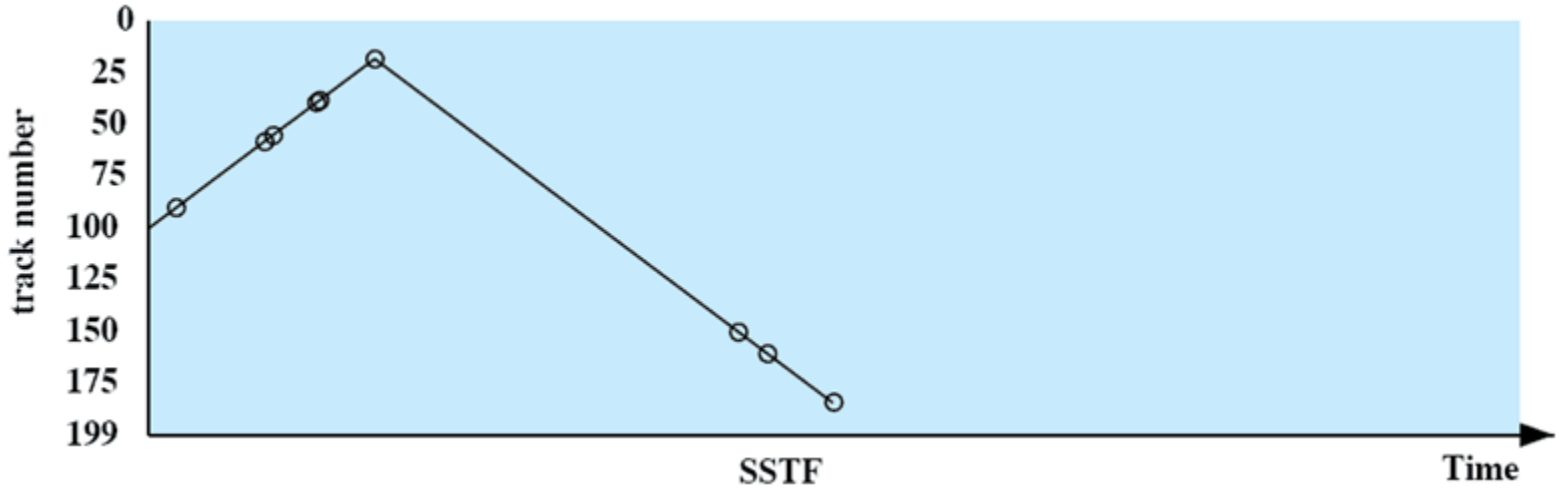
- Select the disk I/O request that requires the least movement of the disk arm from its current position
- Always choose the minimum seek time
- The requested tracks, in the order received by the disk scheduler, are
  - 55, 58, 39, 18, 90, 160, 150, 38, 184.



Shortest Service Time First

(b) SSTF
(starting at track 100)

Next track accessed	Number of tracks traversed
90	10
58	32
55	3
39	16
38	1
18	20
150	132
160	10
184	24
Average seek length	27.5



Shortest Service Time First

Disadvantages

- It may cause starvation for some requests.
- Switching direction on the frequent basis slows the working of algorithm.
- It is not the most optimal algorithm.

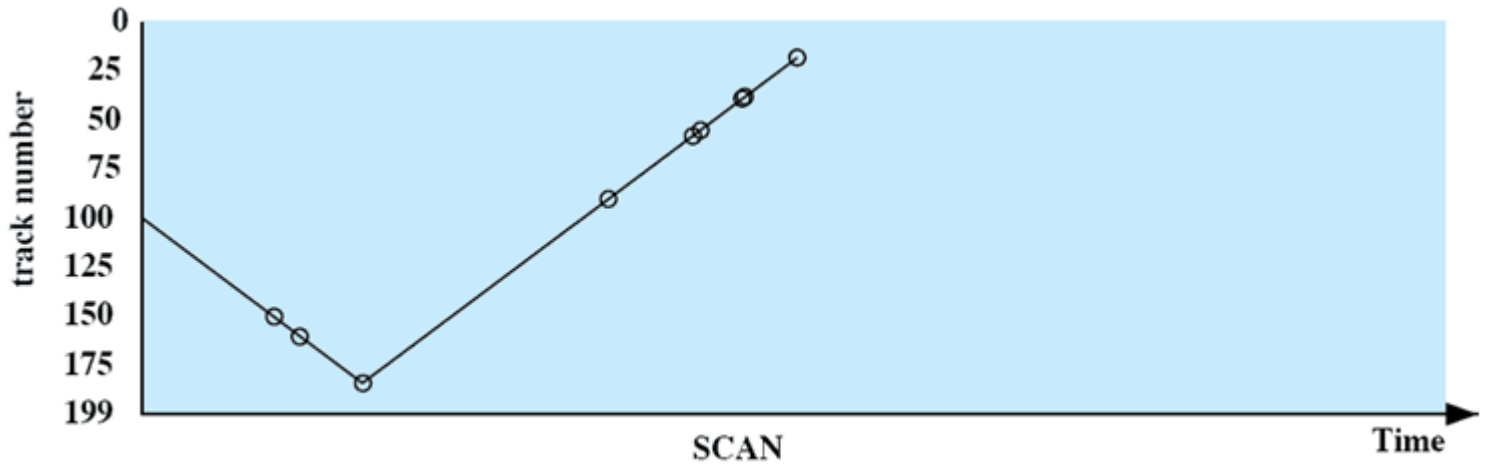
SCAN

- 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
- But note that if requests are uniformly dense, largest density at other end of disk and those wait the longest

SCAN

55, 58, 39, 18, 90, 160, 150, 38, 184.

- Arm moves in one direction only, processing all outstanding requests until it reaches the last track in that direction & then the direction is reversed



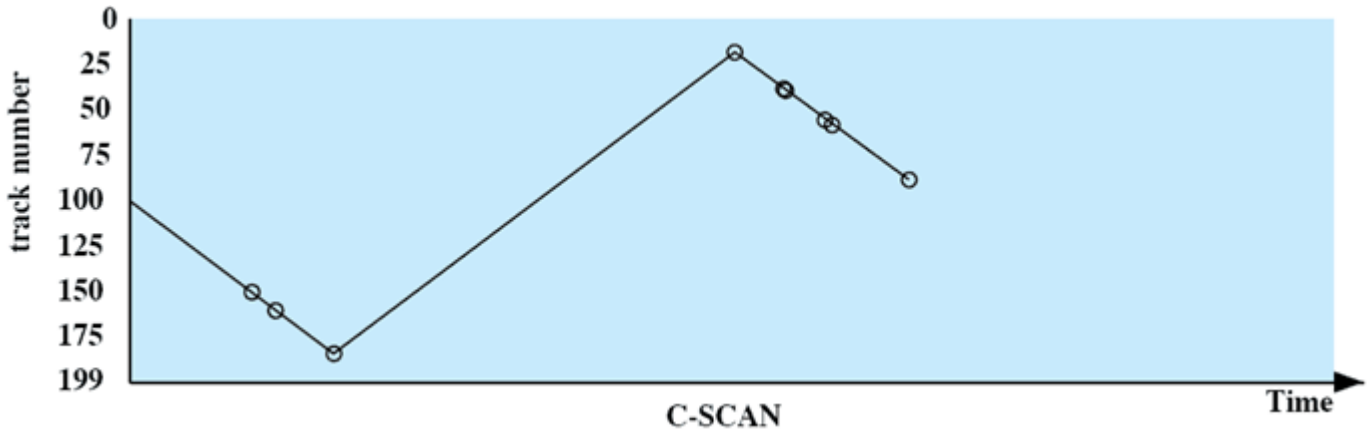
(c) SCAN (starting at track 100, in the direction of increasing track number)	
Next track accessed	**Number of tracks traversed**
150	50
160	10
184	24
90	94
58	32
55	3
39	16
38	1
18	20
Average seek length	27.8

C-SCAN

- Provides a more uniform wait time than SCAN
- The head moves from one end of the disk to the other, servicing requests as it goes
- • When it reaches the other end, however, it immediately returns to the beginning of the disk, 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
- Total number of cylinders?

C-SCAN
55, 58, 39, 18, 90, 160, 150, 38, 184.

- Restricts scanning to one direction only
- When the last track has been visited in one direction, the arm is returned to the opposite end of the disk and the scan begins again



(d) C-SCAN

(starting at track 100, in the direction of increasing track number)

Next track accessed	Number of tracks traversed
150	50
160	10
184	24
18	166
38	20
39	1
55	16
58	3
90	32
Average seek length	35.8

Performance Compared

Comparison of Disk Scheduling Algorithms

(a) FIFO (starting at track 100)		(b) SSTF (starting at track 100)		(c) SCAN (starting at track 100, in the direction of increasing track number)		(d) C-SCAN (starting at track 100, in the direction of increasing track number)	
Next track accessed	**Number of tracks traversed**	**Next track accessed**	**Number of tracks traversed**	**Next track accessed**	**Number of tracks traversed**	**Next track accessed**	**Number of tracks traversed**
55	45	90	10	150	50	150	50
58	3	58	32	160	10	160	10
39	19	55	3	184	24	184	24
18	21	39	16	90	94	18	166
90	72	38	1	58	32	38	20
160	70	18	20	55	3	39	1
150	10	150	132	39	16	55	16
38	112	160	10	38	1	58	3
184	146	184	24	18	20	90	32
Average seek length	55.3	**Average seek length**	27.5	**Average seek length**	27.8	**Average seek length**	35.8

MIT-WPU
।। विश्वशान्तिर्ध्रुवं ध्रुवा ।।

Disk Scheduling Algorithms – Seek Time Problems

FCFS, SSTF, SCAN, and C-SCAN with Examples and Solutions

Problem 1: FCFS (First Come First Serve)

Given: Disk queue: 98, 183, 37, 122, 14, 124, 65, 67

- • Initial head position = 53

Solution Steps:

- 1. Process requests in the given order.
- 2. Head movements:

Step	From	To	Distance
1	53	98	45
2	98	183	85
3	183	37	146
4	37	122	85
5	122	14	108
6	14	124	110
7	124	65	59
8	65	67	2

- 53→98→183→37→122→14→124→65→67
- 3. Total Seek Distance = 640 tracks
- 4. Average Seek Time = 640 / 8 = 80 ms

Problem 2: SSTF (Shortest Seek Time First)

- Given:
- • Same queue, initial head = 53

- Solution Steps:
- 1. Choose the nearest request at each step.
- 2. Head movements:
- 53→65→67→37→14→98→122→124→183
- 3. Total Seek Distance = 236 tracks
- 4. Average Seek Time = 29.5 ms

Problem 3: SCAN (Elevator Algorithm)

- Given:
- • Disk size = 0–199, head = 53, moving right first

- Solution Steps:
- 1. Order: 14, 37, 65, 67, 98, 122, 124, 183
- 2. Move right to end, then reverse.
- 3. Total Seek Distance = 315 tracks
- 4. Average Seek Time = 39.3 ms

Problem 4: C-SCAN (Circular SCAN)

- Given:
- • Disk size = 0–199, head = 53

- Solution Steps:
- 1. Move right till end, then jump to 0.
- 2. Head movements:
- 53→65→67→98→122→124→183→199→0→14→37
- 3. Total Seek Distance = 365 tracks
- 4. Average Seek Time = 45.6 ms

Problem 1: Calculating Disk Access Time

- Given:
- • Seek Time (Ts) = 10 ms
- • Rotational Latency (Tr) = 5 ms
- • Transfer Time (Tt) = 2 ms

- Solution:
- Disk Access Time = Ts + Tr + Tt
- = 10 + 5 + 2 = 17 ms

- ⏱️ Total Disk Access Time = 17 ms

Problem 2: Finding Average Disk Response Time

- Given:
- • Request 1 Access Time = 18 ms
- • Request 2 Access Time = 22 ms
- • Request 3 Access Time = 20 ms

- Solution:
- Average Disk Response Time = (18 + 22 + 20) / 3
- = 60 / 3 = 20 ms

- 📈 Average Disk Response Time = 20 ms

Disk Formatting

- A new **magnetic disk is a blank slate**: It is just a platter of a magnetic recording material.
- Before a disk can store data, it must be **divided into sectors** that the disk controller can read and write. This process is called **low-level formatting, or physical formatting.**
- Low-level formatting **fills the disk with a special data structure for each sector**. The data structure for a sector typically consists of a header, a data area (usually 512 bytes in size), and a trailer.
- The header and trailer contain information used by the disk controller, such as a sector number and an **error-correcting code** (ECC)
- When the controller writes a sector of data during normal I/O, the ECC is updated with a value calculated from all the bytes in the data area.
- When the sector is read, the ECC is recalculated and is compared with the stored value.
- If the stored and calculated numbers are different, this **mismatch indicates that the data area of the sector has become corrupted** and that the disk sector may be bad

Disk Formatting

- The controller automatically does the **ECC processing whenever a sector is read or ,written.**
- Most hard disks are low-level-formatted at the factory as a part of the manufacturing process.
- For many hard disks, when the disk controller is instructed to low-Level-formatting, it can also be told how many bytes of data space to leave between the header and trailer of all sectors.
- It is usually possible to **choose among a few sizes, such as 256, 512, and 1,024 bytes**.
- Formatting a disk with a larger sector size means that fewer sectors can fit on each track; but it also means that fewer headers and trailers are written on each track and more space is available for user data.
- Some operating systems can handle only a sector size of 512 bytes.

Disk Formatting

- To use a **disk to hold files**, the operating system still **needs to record its own data structures** on the disk.
- The first step is to **partition the disk into one or more groups** of cylinders.
- The operating system can treat each partition as though it were a separate disk.
- Eg, one partition can hold a copy of the operating system's executable code, while another holds user files.
- After partitioning, the second step is **logical formatting (or creation of a file system)**. In this step, the operating system stores the initial file-system data structures onto the disk.
- These data structures may include maps of free and allocated space (a FAT or inodes) and an initial empty directory.

Disk Formatting

- To increase efficiency, most file systems group blocks together into larger chunks, frequently called clusters.
- Disk I/O is done via blocks, but file system I/O is done via clusters, effectively assuring that I/O has more sequential-access and fewer random-access characteristics.
- Some operating systems give special programs the ability to use a disk partition as a large sequential array of logical blocks, without any file-system data structures.
- This array is sometimes called the raw disk, and I/O to this array is termed raw I/O. For example, some database systems prefer raw I/O because it enables them to control the exact disk location where each database record is stored.
- Raw I/O bypasses all the file-system services, such as the buffer cache, file locking, prefetching, space allocation, file names, and directories.

Boot Block

- For a computer to start running-for instance, when it is powered up or rebooted-it must have an i**nitial program to run.**
- This initial **bootstrap program** tends to be simple. It **initializes all aspects of the system**, from CPU registers to device controllers and the contents of main memory, and then starts the operating system.
- To do its job, the **bootstrap program finds the operating- system kernel on disk, loads that kernel into memory, and jumps to an initial address to begin the operating-system** execution.
- For most computers, the **bootstrap is stored in read-only memory (ROM)**. This location is convenient, because ROM needs no initialization and is at a fixed location that the processor can start executing when powered up or reset.

Boot Block

- And, since ROM is read only, it cannot be infected by a computer virus. The problem is that changing this bootstrap code requires changing the ROM hardware chips.
- For this reason, most systems store a **tiny bootstrap loader program in the boot ROM** whose only job is to **bring in a full bootstrap** program from disk.
- The full bootstrap program can be changed easily: A new version is simply written onto the disk.
- The **full bootstrap program** is stored in "**the boot blocks**" at a fixed location on the disk. A disk that has a **boot partition is called a boot disk or system disk**.
- The code in the boot ROM instructs the disk controller to read the boot blocks into memory (no device drivers are loaded at this point) and then starts executing that code.
- The full bootstrap program is more sophisticated than the bootstrap loader in the boot ROM; it is able to load the entire operating system from a non-fixed location on disk and to start the OS running

Boot Block

- Example the boot process in Windows 2000-
- The Windows 2000 system places its boot code in the first sector on the hard disk (which it terms the master boot record, or MBR).
- Windows 2000 allows a hard disk to be divided into one or more partitions; one partition, identified as the boot partition, contains the operating system and device drivers.
- Booting begins in a Windows 2000 system by running code that is resident in the system's ROM memory.
- This code directs the system to read the boot code from the MBR. In addition to containing boot code, the MBR contains a table listing the partitions for the hard disk and a flag indicating which partition the system is to be booted from.
- Once the system identifies the boot partition, it reads the first sector from that partition (which is called the boot sector) and continues with the remainder of the boot process, which includ.es loading the various subsystems and system services.

Bad Blocks

- As **disks have moving parts** and small tolerances, they are **prone to failure**. Sometimes the failure is complete; in this case, the disk needs to be replaced and its contents restored from backup media to the new disk.
- one or more sectors become defective. Most disks even come from the factory with bad blocks. Depending on the disk and controller in use, these blocks are handled in a variety of ways.
- On simple disks, such as some disks with IDE controllers, bad blocks are handled manually. For instance, the MS-DOS format command performs logical formatting and, as a part of the process, scans the disk to find bad blocks.
- If format finds a bad block, it writes a special value into the corresponding FAT entry to tell the allocation routines not to use that block.
- If blocks go bad during normal operation, a special program (such as chkdsk) must be run manually to search for tIle bad blocks and to lock them away as before.
- Data that resided on the bad blocks usually are lost.

Bad Blocks

- More sophisticated disks, such as the SCSI disks used in high-end PCs and most workstations and servers are smarter about bad-block recovery.
- The controller maintains a list of bad blocks on the disk.
- The list is initialized during the low-level formatting at the factory and is updated over the life of the disk.
- Low-level formatting also sets aside spare sectors not visible to the operating system.
- The controller can be told to replace each bad sector logically with one of the spare sectors.
- This scheme is known as sector sparing or forwarding.

Bad Blocks

- A typical bad-sector transaction might be as follows:
- The operating system tries to read logical block 87.
- The controller calculates the ECC and finds that the sector is bad. It reports this finding to the operating system.
- The next time the system is rebooted, a special command is run to tell the SCSI controller to replace the bad sector with a spare.
- After that, whenever the system requests logical block 87, the request is translated into the replacement sector's address by the controller.