Module-5

By Dr. Renuka S. Gound

Module-5

- File-System Interface: File Structure, Access methods Sequential
- Access, Direct Access, Other Access Methods
- Implementation: Directory Implementation Linear List, Hash Table,
- Allocation Methods Contiguous Allocation, Linked Allocation, Indexed
- Allocation, Free Space Management Bit-Vector, Linked List,
- Grouping, Counting.
- Mass Storage Structures: Overview, Disk Structure, Disk Scheduling
- FCFS, SSTF, SCAN, CSCAN, LOOK

File Structure

- None - sequence of words, bytes
- Simple record structure
  - Lines
  - Fixed length
  - Variable length
- Complex Structures
  - Formatted document
  - Relocatable load file
- Can simulate last two with first method by inserting appropriate control characters
- Who decides:
  - Operating system
  - Program

File Structure

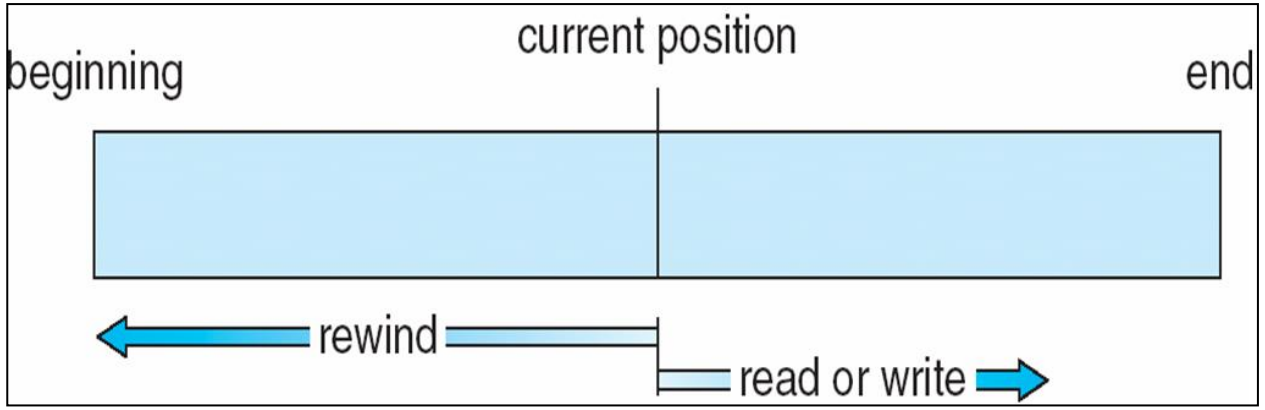
- A File Structure should be according to a required format that the operating system can understand.
- A file has a certain defined structure according to its type.
- A text file is a sequence of characters organized into lines.
- A source file is a sequence of procedures and functions.
- An object file is a sequence of bytes organized into blocks that are understandable by the machine.
- When operating system defines different file structures, it also contains the code to support these file structure. Unix, MS-DOS support minimum number of file structure.

File access mechanism

- File access mechanism refers to the manner in which the records of a file may be accessed. There are several ways to access files –
- Sequential access
- Direct/Random access
- Indexed sequential access
- Other Access Methods

Sequential access

- A sequential access is that in which the records are accessed in some sequence, i.e., the information in the file is processed in order, one record after the other. This access method is the most primitive one. Example: Compilers usually access files in this fashion.



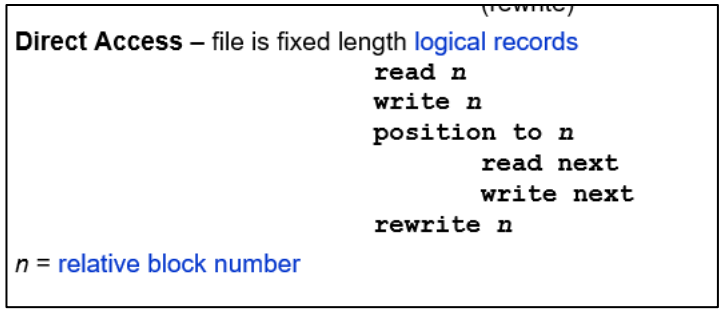
Sequential Access

```
read next
write next
reset
```

no read after last write
(rewrite)

Direct/Random access

- Random access file organization provides, accessing the records directly.
- Each record has its own address on the file with by the help of which it can be directly accessed for reading or writing.
- The records need not be in any sequence within the file and they need not be in adjacent locations on the storage medium.



Indexed sequential access

- This mechanism is built up on base of sequential access.
- An index is created for each file which contains pointers to various blocks.
- Index is searched sequentially and its pointer is used to access the file directly.

Other Access Methods

- Can be built on top of base methods
- General involve creation of an index for the file
- Keep index in memory for fast determination of location of data to be operated on (consider UPC code plus record of data about that item)
- If too large, index (in memory) of the index (on disk)
- IBM indexed sequential-access method (ISAM)
  - Small master index, points to disk blocks of secondary index
  - File kept sorted on a defined key
  - All done by the OS
- VMS operating system provides index and relative files as another example (see next slide)

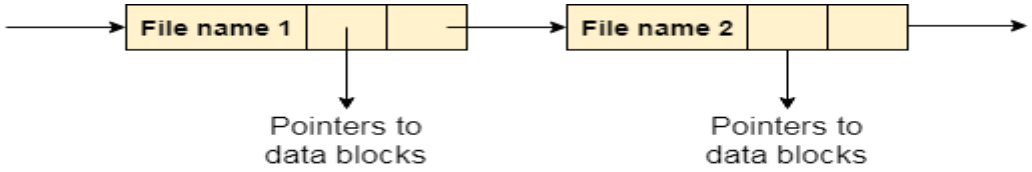
Directory Implementation

- Directory implementation in the operating system can be done using Singly Linked List and Hash table.
- The efficiency, reliability, and performance of a file system are greatly affected by the selection of directory allocation and directory-management algorithms.
- There are numerous ways in which the directories can be implemented. But we need to choose an appropriate directory implementation algorithm that enhances the performance of the system.

Linear List

- In this algorithm, all the files in a directory are maintained as singly lined list. Each file contains the pointers to the data blocks which are assigned to it and the next file in the directory.
- **Characteristics**

1. When a new file is created, then the entire list is checked whether the new file name is matching to a existing file name or not. In case, it doesn't exist, the file can be created at the beginning or at the end. Therefore, searching for a unique name is a big concern because traversing the whole list takes time.
2. The list needs to be traversed in case of every operation (creation, deletion, updating, etc) on the files, therefore, the systems become inefficient.

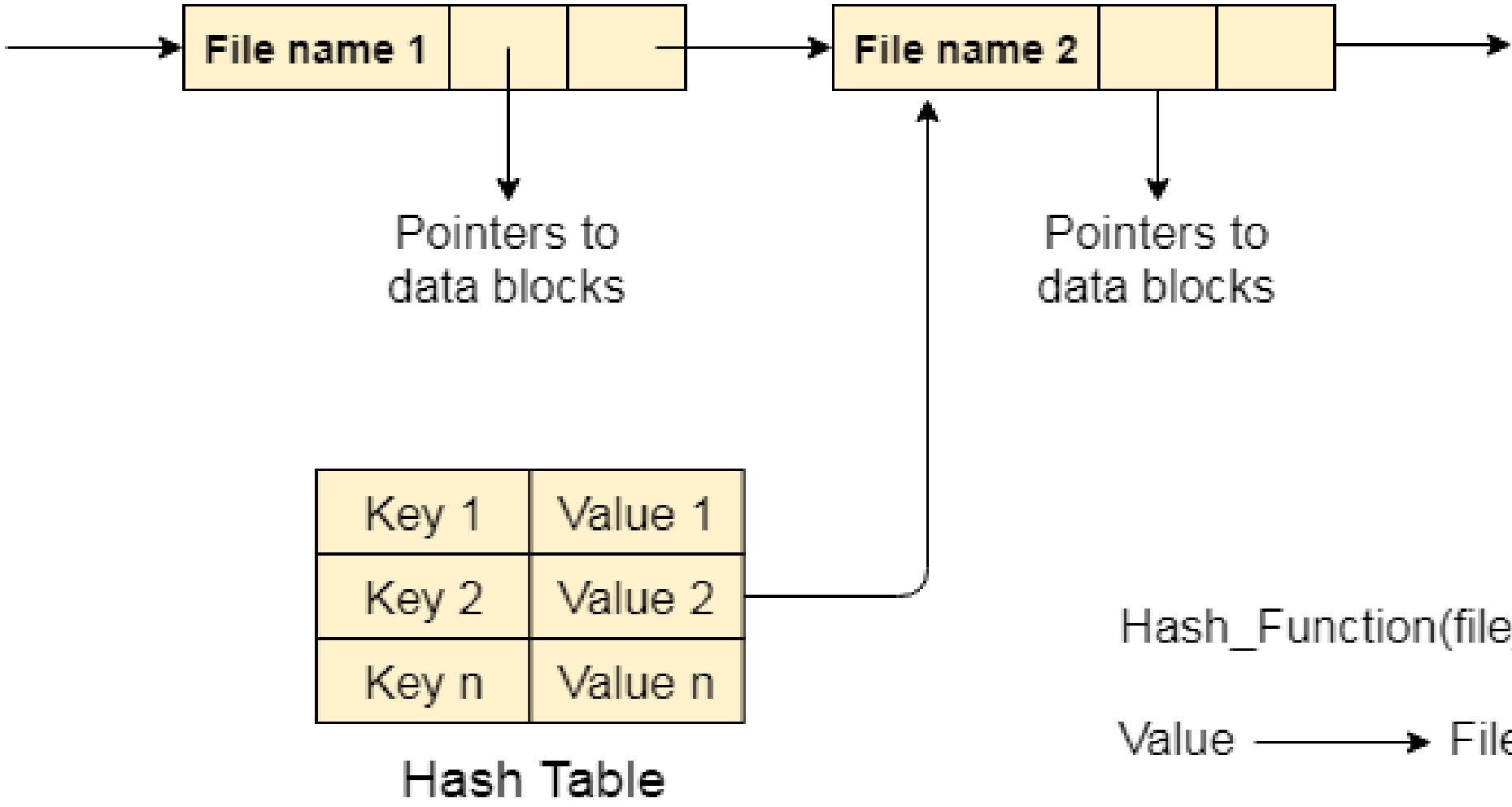


Linear List

Hash Table

- To overcome the drawbacks of singly linked list implementation of directories, there is an alternative approach that is hash table. This approach suggests to use hash table along with the linked lists.
- A key-value pair for each file in the directory gets generated and stored in the hash table. The key can be determined by applying the hash function on the file name while the key points to the corresponding file stored in the directory.
- Now, searching becomes efficient due to the fact that now, entire list will not be searched on every operating. Only hash table entries are checked using the key and if an entry found then the corresponding file will be fetched using the value.

Hash Table


File name 1
File name 2
Pointers to data blocks
Pointers to data blocks
Key 1
Value 1
Key 2
Value 2
Key n
Value n
Hash Table
Hash_Function(file_name) = key
Value
File

Allocation Methods

- There are various methods which can be used to allocate disk space to the files. Selection of an appropriate allocation method will significantly affect the performance and efficiency of the system. Allocation method provides a way in which the disk will be utilized and the files will be accessed.

Contiguous Allocation

- If the blocks are allocated to the file in such a way that all the logical blocks of the file get the contiguous physical block in the hard disk then such allocation scheme is known as contiguous allocation.
- In the image shown below, there are three files in the directory. The starting block and the length of each file are mentioned in the table. We can check in the table that the contiguous blocks are assigned to each file as per its need.

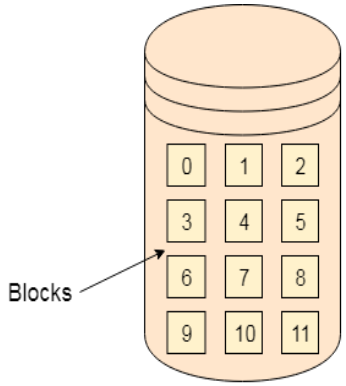
Contiguous Allocation

- **Advantages**

1. It is simple to implement.
2. We will get Excellent read performance.
3. Supports Random Access into files.

- **Disadvantages**

1. The disk will become fragmented.
2. It may be difficult to have a file grow.



Hard Disk

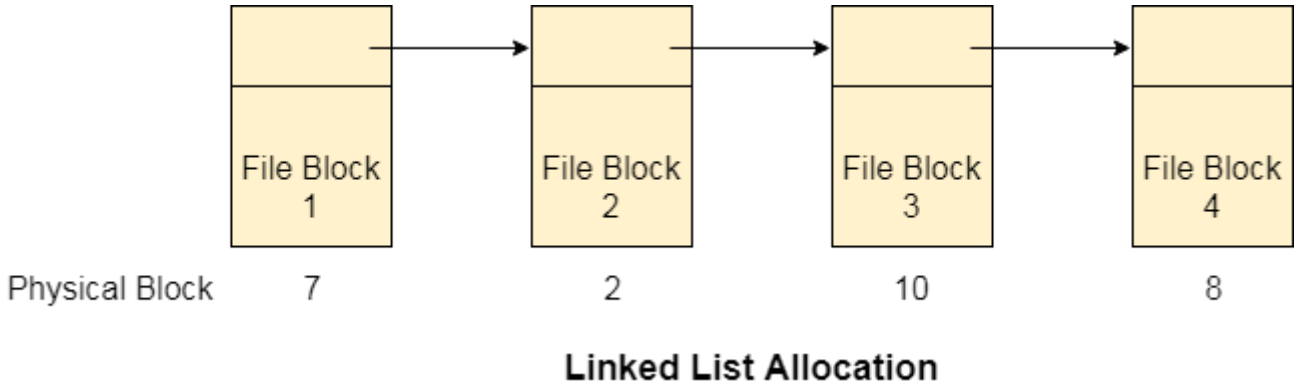
File Name	Start	Length	Allocated Blocks
abc.text	0	3	0,1,2
video.mp4	4	2	4,5
jtp.docx	9	3	9,10,11

Directory

Contiguous Allocation

Linked Allocation

- Linked List allocation solves all problems of contiguous allocation. In linked list allocation, each file is considered as the linked list of disk blocks. However, the disks blocks allocated to a particular file need not to be contiguous on the disk. Each disk block allocated to a file contains a pointer which points to the next disk block allocated to the same file.



Linked List Allocation

Linked Allocation

Advantages

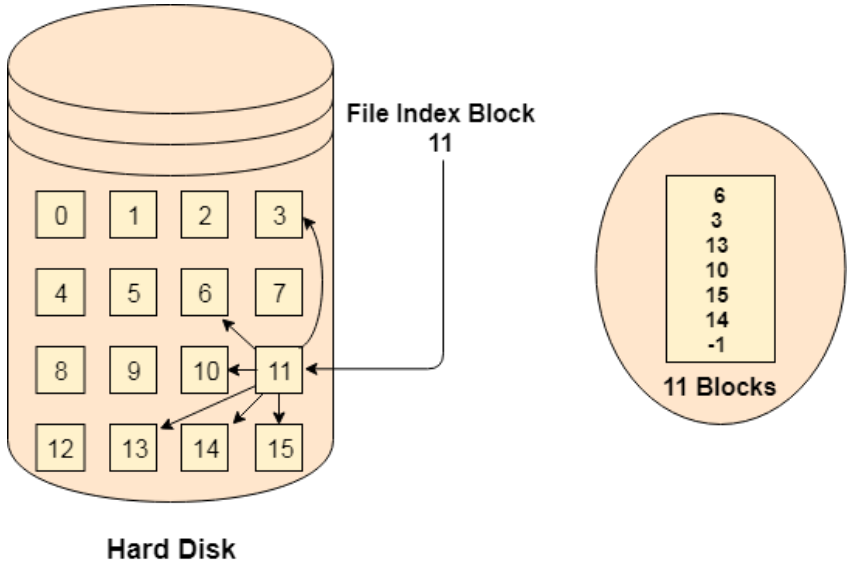
- There is no external fragmentation with linked allocation.
- Any free block can be utilized in order to satisfy the file block requests.
- File can continue to grow as long as the free blocks are available.
- Directory entry will only contain the starting block address.

Disadvantages

- Random Access is not provided.
- Pointers require some space in the disk blocks.
- Any of the pointers in the linked list must not be broken otherwise the file will get corrupted.
- Need to traverse each block.

Indexed Allocation Scheme

Instead of maintaining a file allocation table of all the disk pointers, Indexed allocation scheme stores all the disk pointers in one of the blocks called as indexed block. Indexed block doesn't hold the file data, but it holds the pointers to all the disk blocks allocated to that particular file. Directory entry will only contain the index block address.



Indexed Allocation Scheme

Advantages

- Supports direct access
- A bad data block causes the lost of only that block.

Disadvantages

- A bad index block could cause the lost of entire file.
- Size of a file depends upon the number of pointers, a index block can hold.
- Having an index block for a small file is totally wastage.
- More pointer overhead

Free Space Management

- A file system is responsible to allocate the free blocks to the file therefore it has to keep track of all the free blocks present in the disk. There are mainly two approaches by using which, the free blocks in the disk are managed.
- 1. Bit Vector
- In this approach, the free space list is implemented as a bit map vector. It contains the number of bits where each bit represents each block.
- If the block is empty then the bit is 1 otherwise it is 0. Initially all the blocks are empty therefore each bit in the bit map vector contains 1.

Free Space Management

- 2. Linked List
- It is another approach for free space management. This approach suggests linking together all the free blocks and keeping a pointer in the cache which points to the first free block.

- Therefore, all the free blocks on the disks will be linked together with a pointer. Whenever a block gets allocated, its previous free block will be linked to its next free block.

Free Space Management

3. Grouping – This approach stores the address of the free blocks in the first free block. The first free block stores the address of some, say n free blocks. Out of these n blocks, the first n-1 blocks are actually free and the last block contains the address of next free n blocks. An advantage of this approach is that the addresses of a group of free disk blocks can be found easily.

4. Counting – This approach stores the address of the first free disk block and a number n of free contiguous disk blocks that follow the first block. Every entry in the list would contain:

Address of first free disk block

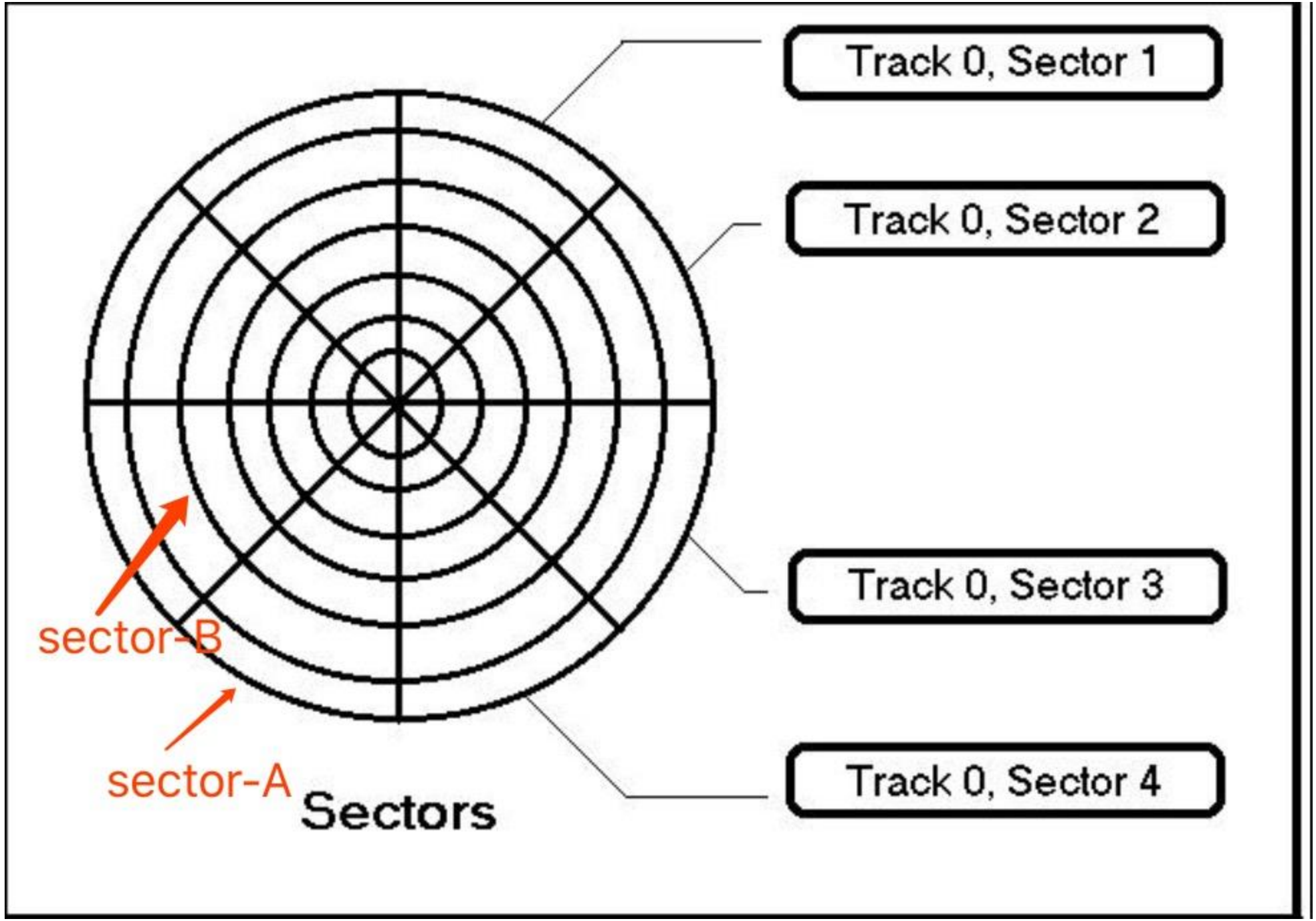
A number n

Disk Scheduling Algorithms

- Disk scheduling is done by operating systems to schedule I/O requests arriving for the disk. Disk scheduling is also known as I/O Scheduling.

Importance of Disk Scheduling in Operating System

1. Multiple I/O requests may arrive by different processes and only one I/O request can be served at a time by the disk controller. Thus other I/O requests need to wait in the waiting queue and need to be scheduled.
2. Two or more requests may be far from each other so this can result in greater disk arm movement.
3. Hard drives are one of the slowest parts of the computer system and thus need to be accessed in an efficient manner.


Track 0, Sector 1
Track 0, Sector 2
Track 0, Sector 3
Track 0, Sector 4
sector-B
sector-A
Sectors

Key Terms Associated with Disk Scheduling

- Seek Time: Seek time is the time taken to locate the disk arm to a specified track where the data is to be read or written. So the disk scheduling algorithm that gives a minimum average seek time is better.
- Rotational Latency: Rotational Latency is the time taken by the desired sector of the disk to rotate into a position so that it can access the read/write heads. So the disk scheduling algorithm that gives minimum rotational latency is better.
- Transfer Time: Transfer time is the time to transfer the data. It depends on the rotating speed of the disk and the number of bytes to be transferred.
- Disk Access Time:
- Disk Access Time = Seek Time + Rotational Latency + Transfer Time
- Total Seek Time = Total head Movement * Seek Time

Disk Scheduling Algorithms

- **FCFS (First Come First Serve)**
- FCFS is the simplest of all Disk Scheduling Algorithms. In FCFS, the requests are addressed in the order they arrive in the disk queue. Let us understand this with the help of an example.

Advantages of FCFS

Here are some of the advantages of First Come First Serve.

- Every request gets a fair chance
- No indefinite postponement
- Disadvantages of FCFS

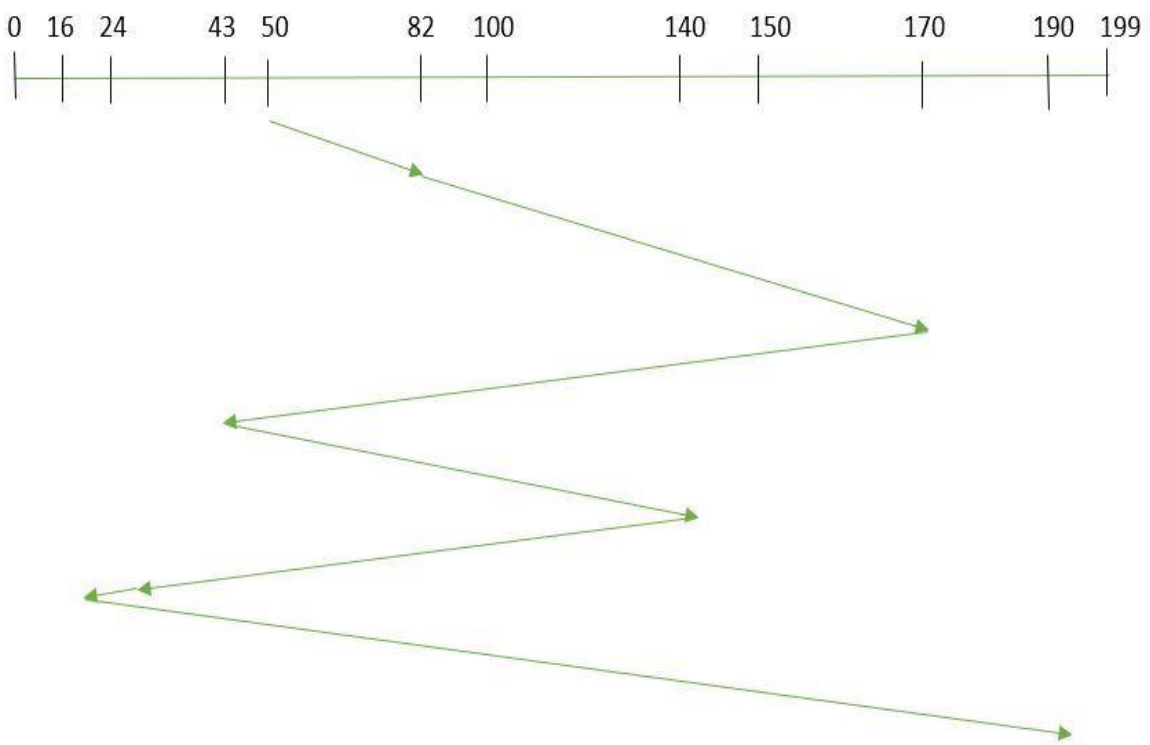
Here are some of the disadvantages of First Come First Serve.

- Does not try to optimize seek time
- May not provide the best possible service

Example:

- Suppose the order of request is- (82,170,43,140,24,16,190)
- And current position of the Read/Write head is: 50
- So, total overhead movement (total distance covered by the disk arm) =

(82-50)+(170-82)+(170-43)+(140-43)+(140-24)+(24-16)+(190-16) =642



Example:

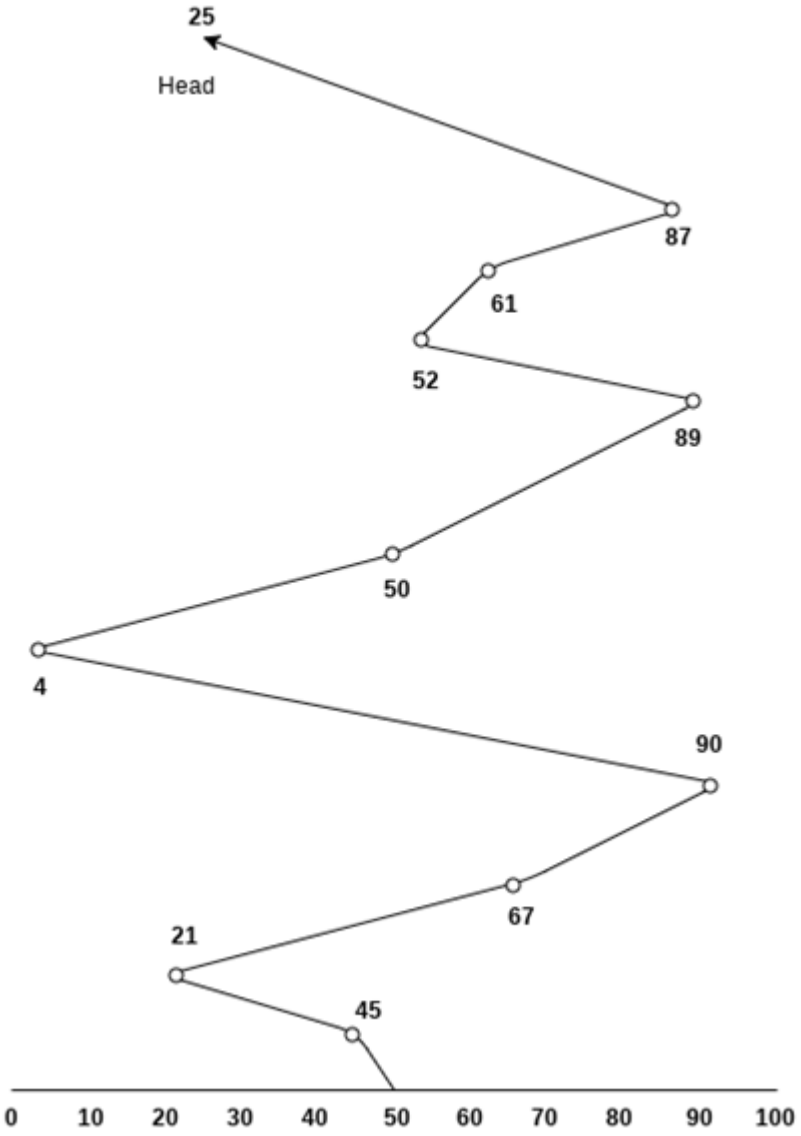
- Consider the following disk request sequence for a disk with 100 tracks 45, 21, 67, 90, 4, 50, 89, 52, 61, 87, 25
- Head pointer starting at 50 and moving in left direction. Find the number of head movements in cylinders using FCFS scheduling.

Number of cylinders moved by the head

= (50-45)+(45-21)+(67-21)+(90-67)+(90-4)+(50-4)+(89-50)+(61-52)+(87-61)+(87-25)

= 5 + 24 + 46 + 23 + 86 + 46 + 49 + 9 + 26 + 62

= 376



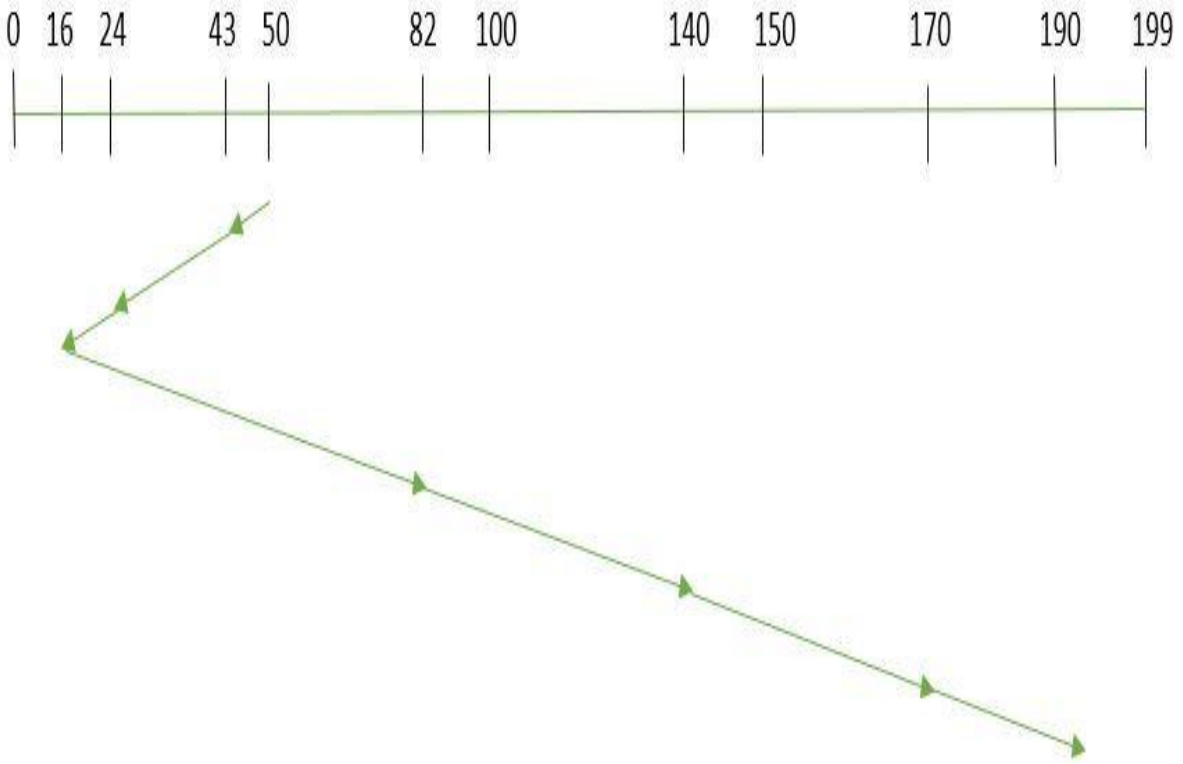
SSTF (Shortest Seek Time First)

- In SSTF (Shortest Seek Time First), requests having the shortest seek time are executed first. So, the seek time of every request is calculated in advance in the queue and then they are scheduled according to their calculated seek time.
- As a result, the request near the disk arm will get executed first. SSTF is certainly an improvement over FCFS as it decreases the average response time and increases the throughput of the system. Let us understand this with the help of an example.

Example

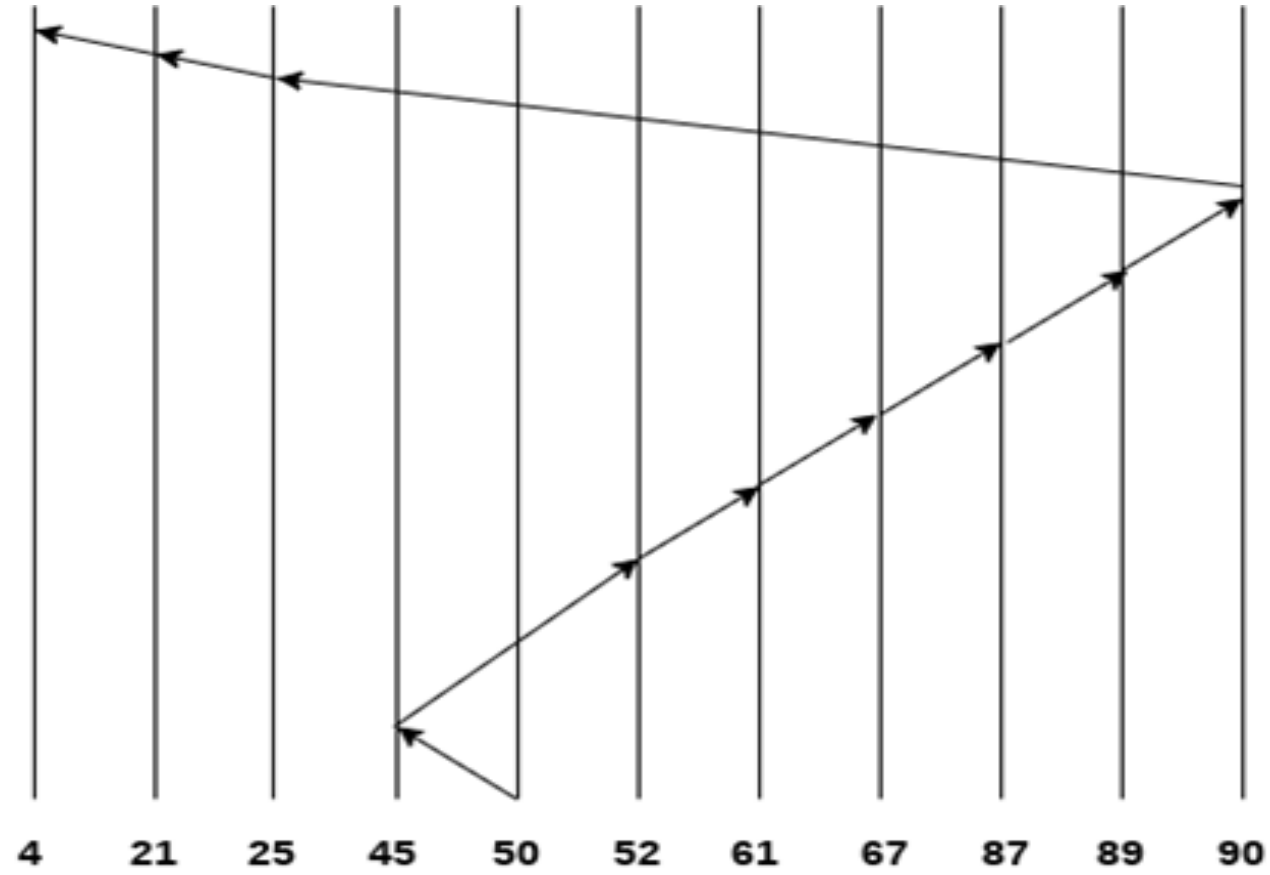
- Suppose the order of request is- (82,170,43,140,24,16,190)
- And current position of Read/Write head is: 50
- So, total overhead movement (total distance covered by the disk arm) =

(50-43)+(43-24)+(24-16)+(82-16)+(140-82)+(170-140)+(190-170) =208



Example

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



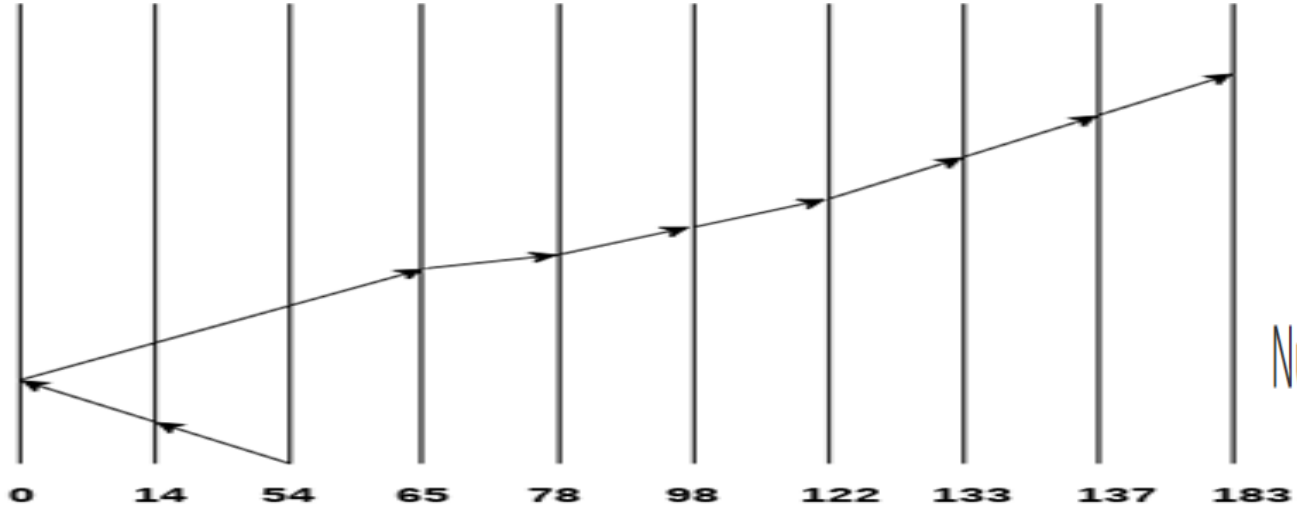
Number of cylinders = 5 + 7 + 9 + 6 + 20 + 2 + 1 + 65 + 4 + 17 = 136

SCAN and C-SCAN algorithm

- **Scan Algorithm**
- It is also called as Elevator Algorithm. In this algorithm, the disk arm moves into a particular direction till the end, satisfying all the requests coming in its path,and then it turns backand moves in the reverse direction satisfying requests coming in its path.
- It works in the way an elevator works, elevator moves in a direction completely till the last floor of that direction and then turns back.

Example

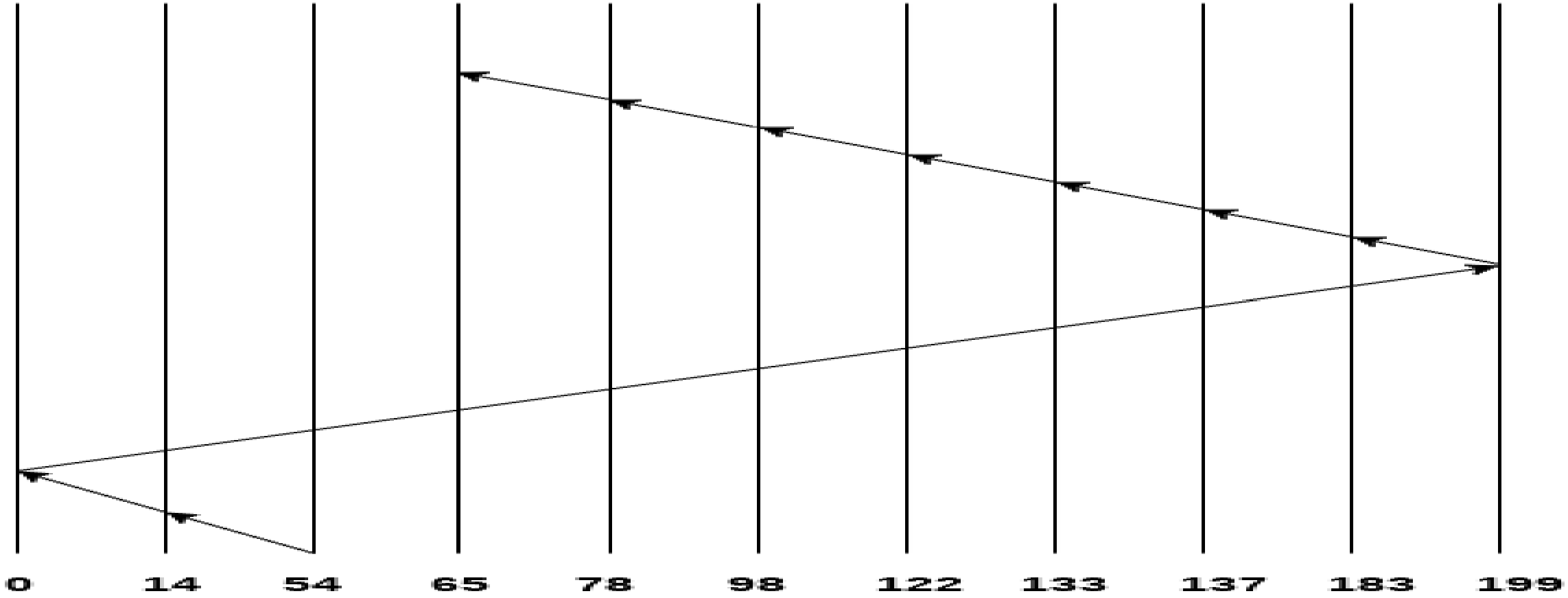
- Consider the following disk request sequence for a disk with 100 tracks
- 98, 137, 122, 183, 14, 133, 65, 78
- Head pointer starting at 54 and moving in left direction. Find the number of head movements in cylinders using SCAN scheduling.



Number of Cylinders = 40 + 14 + 65 + 13 + 20 + 24 + 11 + 4 + 46 = 237

C-SCAN algorithm

- In C-SCAN algorithm, the arm of the disk moves in a particular direction servicing requests until it reaches the last cylinder, then it jumps to the last cylinder of the opposite direction without servicing any request then it turns back and start moving in that direction servicing the remaining requests.
- Example
- Consider the following disk request sequence for a disk with 100 tracks
- 98, 137, 122, 183, 14, 133, 65, 78
- Head pointer starting at 54 and moving in left direction. Find the number of head movements in cylinders using C-SCAN scheduling.



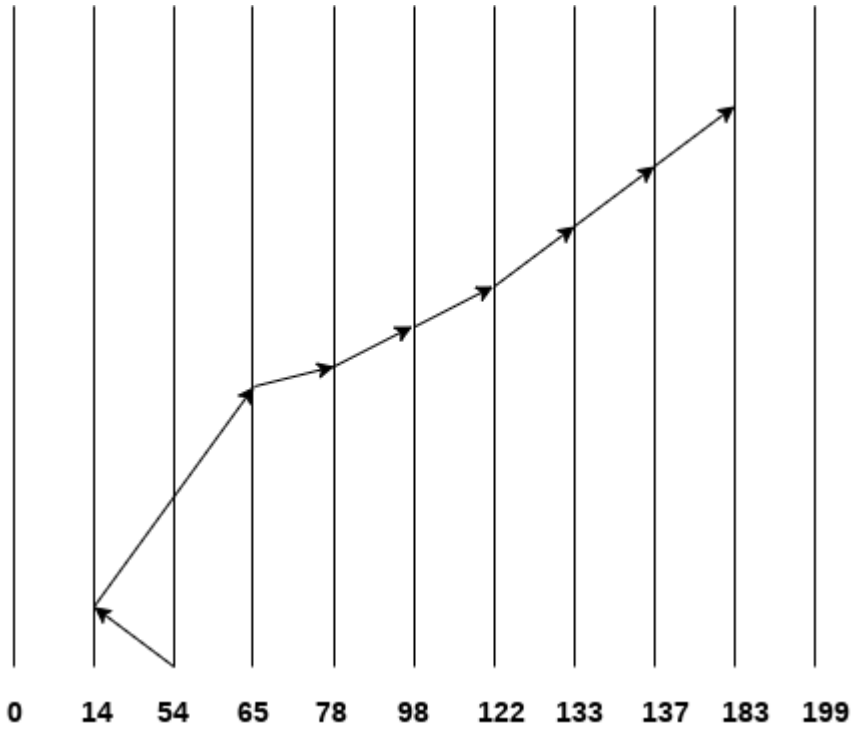
No. of cylinders crossed = 40 + 14 + 199 + 16 + 46 + 4 + 11 + 24 + 20 + 13 = 387

Look Scheduling

- It is like SCAN scheduling Algorithm to some extant except the difference that, in this scheduling algorithm, the arm of the disk stops moving inwards (or outwards) when no more request in that direction exists. This algorithm tries to overcome the overhead of SCAN algorithm which forces disk arm to move in one direction till the end regardless of knowing if any request exists in the direction or not.

Example

- Consider the following disk request sequence for a disk with 100 tracks
- 98, 137, 122, 183, 14, 133, 65, 78
- Head pointer starting at 54 and moving in left direction. Find the number of head movements in cylinders using LOOK scheduling.



Number of cylinders crossed = 40 + 51 + 13 + +20 + 24 + 11 + 4 + 46 = 209