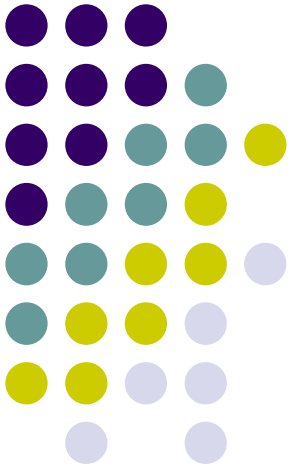
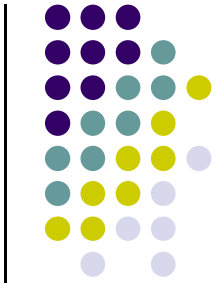
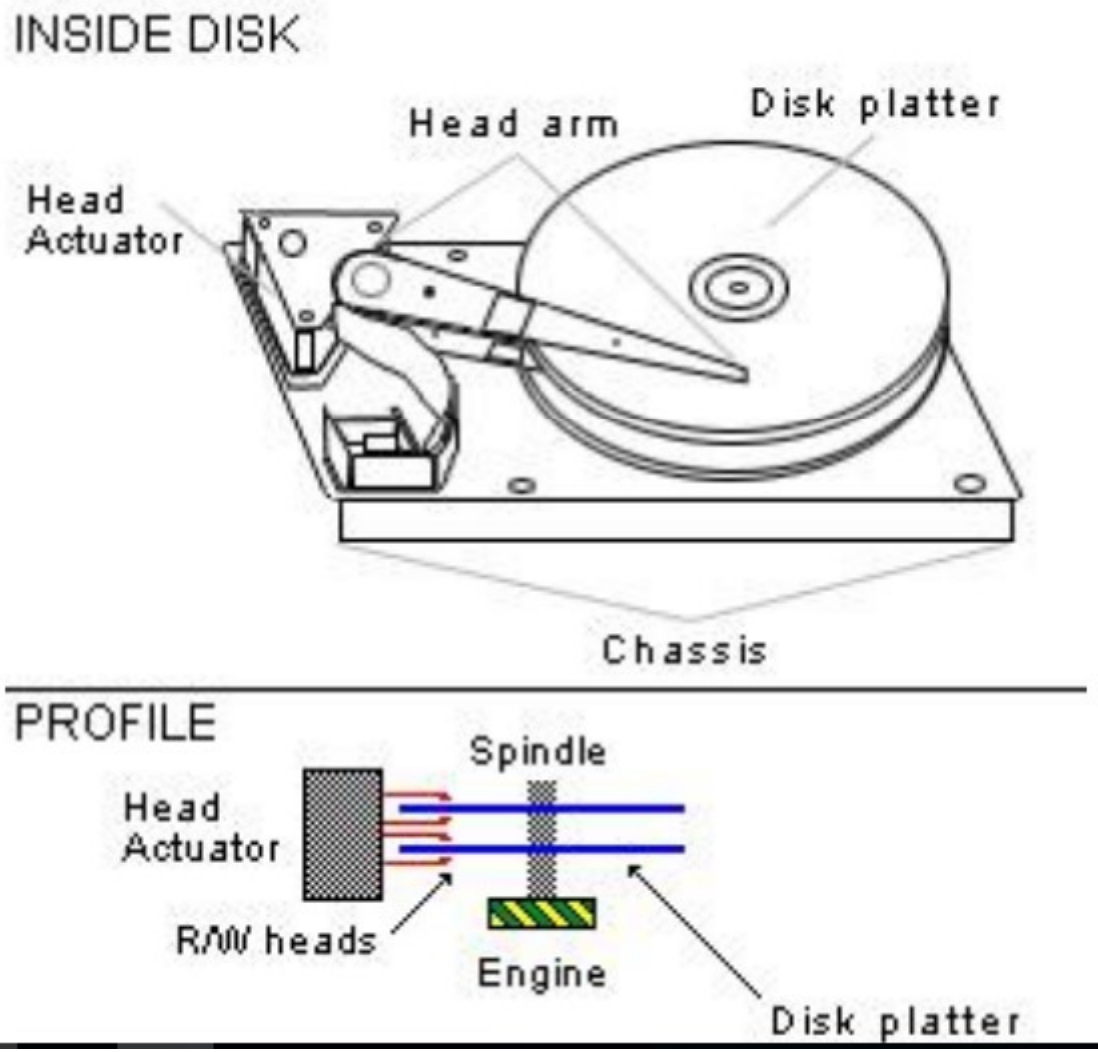
Disk Scheduling

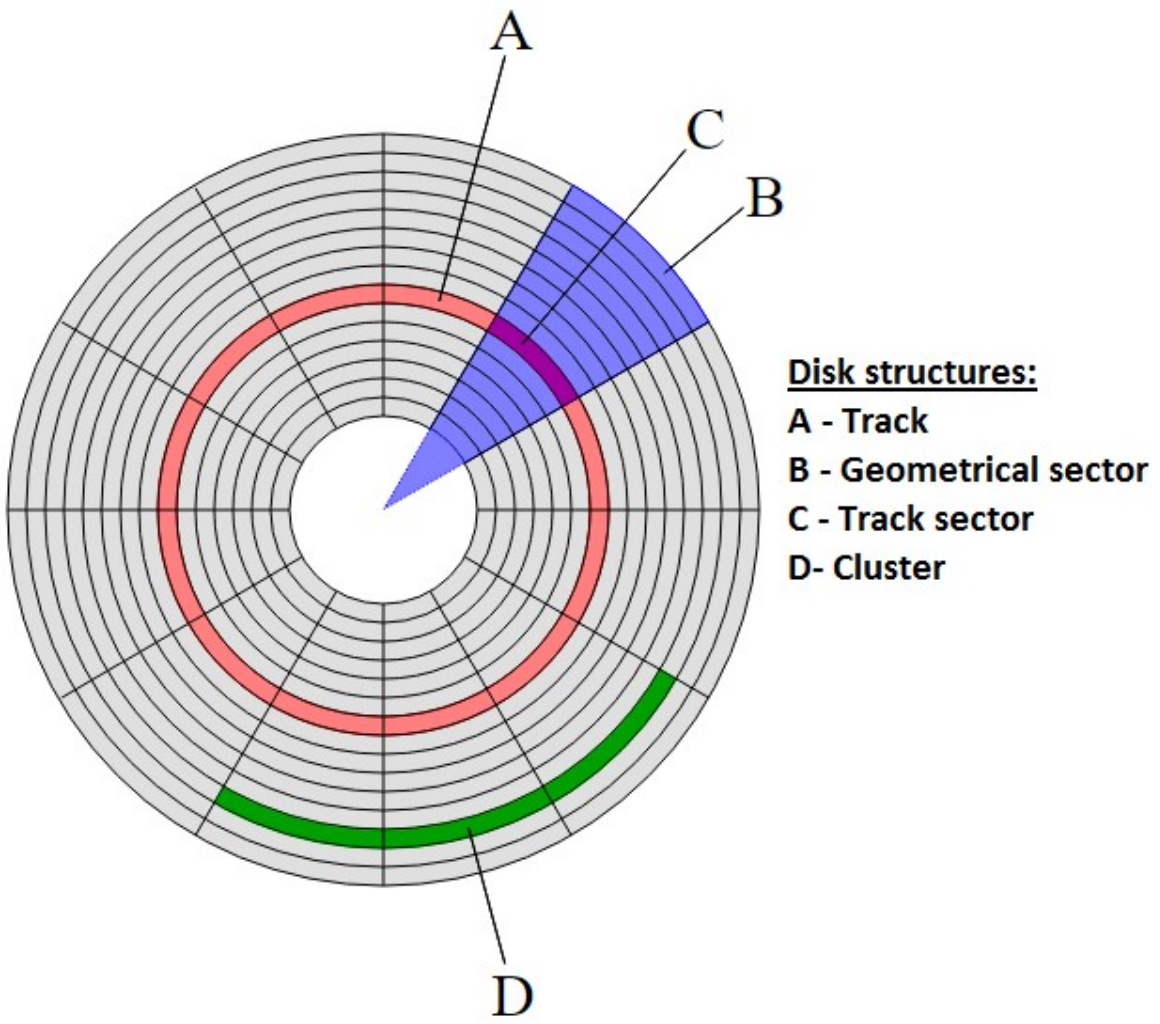
S.Rajarajan





- The increase in the speed of processors and main memory has outstripped that for disk access
- The result is that **disks** are currently at least four orders of magnitude **slower than main memory**.
- This gap is expected to continue into the foreseeable future.
- Thus, the performance of disk storage subsystem is of vital concern, and much research has gone into schemes for improving that performance

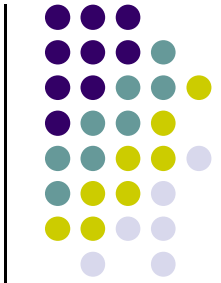

INSIDE DISK
Head arm
Disk platter
Head
Actuator
Chassis
PROFILE
Spindle
Head
Actuator
R/W heads
Engine
Disk platter



- **Track:** one ring
- **Sector:** one pie-shaped piece.
- **Block:** intersection of a track and a sector.

Disk Performance Parameters

- The actual details of disk I/O operation depend on the computer system, the operating system, and the nature of the I/O channel and disk controller hardware.
- When the disk drive is operating, the **disk is rotating at constant speed**.
- To **read or write**, the **head must be positioned at the desired track** and **at the beginning of the desired sector** on that track.
- **Track selection** involves **moving the head** in a movable head system or electronically selecting one head on a fixed-head system.
- On a movable-head system, the time it takes to position the head at the track is known as **seek time**.



- Once the track is selected, the disk controller waits until the appropriate **sector rotates to line up with the head**.
- The time it takes for the beginning of the sector to reach the head is known as **rotational delay**, or rotational latency.
- The sum of the seek time, if any, and the rotational delay equals the **access time**, which is the **time it takes to get into position** to read or write.

- Once the head is in position, the **read or write operation is then performed**. This is the data transfer portion of the operation; the time required for the transfer is the **transfer time**.
- In addition to the access time and transfer time, there are **several queuing delays** normally associated with a disk I/O operation.
- When a process issues an I/O request, it must first wait in a ***queue*** for the device to be available.
- If the device shares a single ***I/O channel*** or a set of I/O channels with other disk drives, then there may be an **additional wait for the channel to be available**.
- At that point, the **seek is performed** to begin disk access.

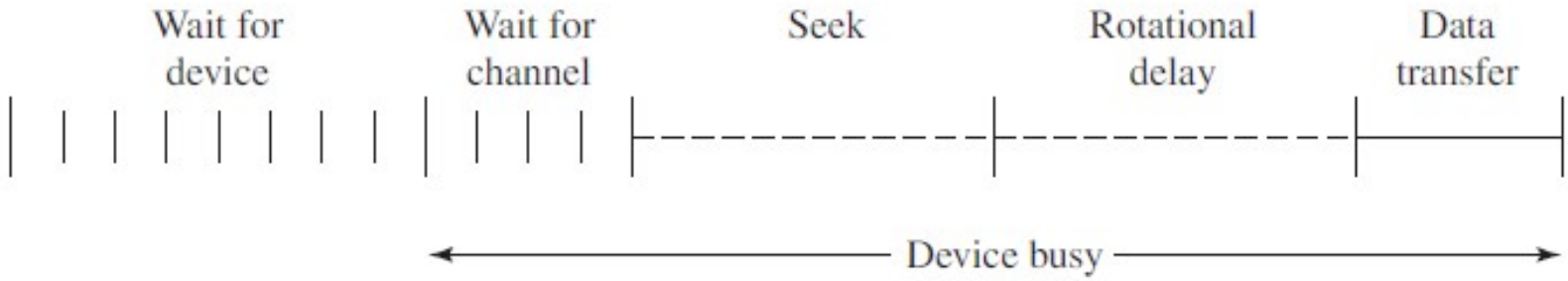
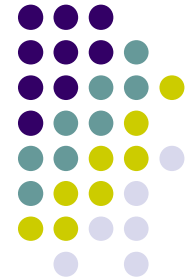


Figure 11.6 **Timing of a Disk I/O Transfer**

Seek Time

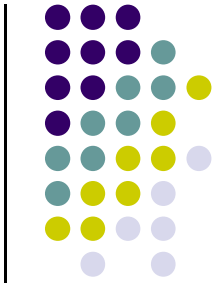
- Seek time is **the time required to move the disk arm head to the required track**.
- This is a difficult quantity to pin down.
- The seek time consists of two key components:
  - the **initial startup time**
  - the **time taken to traverse the tracks** that have to be crossed once the access arm is up to speed
- Unfortunately, the traversal time is not a linear function of the number of tracks but includes a settling time (time after positioning the head over the target track until track identification is confirmed).

Rotational Delay



- Rotational delay is the time required for the addressed area of the disk to rotate into a position where it is accessible by the read/write head.
- Disks rotate at speeds ranging from **3600 RPM** up to, as of this writing, **15,000 RPM**; at this latter speed, there is one revolution per 4 ms.
- Thus, on the average, the rotational delay will be 2 ms.

Transfer Time



- The transfer time to or from the disk depends on the rotation speed of the disk in the following fashion:

$$T = \frac{b}{rN}$$

where

T = transfer time
b = number of bytes to be transferred
N = number of bytes on a track
r = rotation speed, in revolutions per second

Thus the total average access time can be expressed as

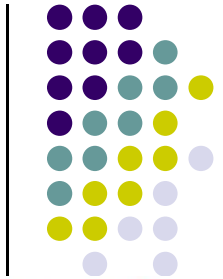
$$T_a = T_s + \frac{1}{2r} + \frac{b}{rN}$$

A Timing Comparison

- With the foregoing parameters defined, let us look at two different I/O operations that illustrate the danger of relying on average values.
- Consider a disk with an
  - average seek time of 4 ms
  - rotation speed of 7500 rpm
  - and 512-byte sectors with 500 sectors per track
- Suppose that we wish to read a file consisting of 2500 sectors for a total of 1.28 Mbytes (5 consecutive tracks).
- We would like to estimate the total time for the transfer.

- First, let us assume that the file is stored as compactly as possible on the disk.
- That is, the file occupies all of the sectors on 5 adjacent tracks (5 tracks X 500 sectors/track = 2500 sectors).
- This is known as *sequential organization*. The time to read the first track is as follows:

Average seek	4 ms
Rotational delay	4 ms
Read 500 sectors	8 ms
	16 ms



Suppose that the remaining tracks can now be read with essentially no seek time. That is, the I/O operation can keep up with the flow from the disk. Then, at most, we need to deal with rotational delay for each succeeding track. Thus, each successive track is read in $4 + 8 = 12\,rmms$. To read the entire file,

$$\text{Total time} = 16 + (4 \times 12) = 64\,\text{ms} = 0.064\,\text{seconds}$$

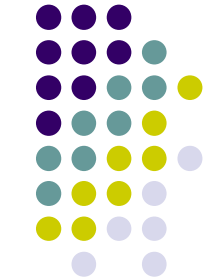
Now let us calculate the time required to read the same data using random access rather than sequential access; that is, accesses to the sectors are distributed randomly over the disk. For each sector, we have:

Average seek	4 ms
Rotational delay	4 ms
Read 1 sector	0.016 ms
	8.016 ms

$$\text{Total time} = 2{,}500 \times 8.016 = 20{,}040\,\text{ms} = 20.04\,\text{seconds}$$

It is clear that the order in which sectors are read from the disk has a tremendous effect on I/O performance. In the case of file access in which multiple sectors

Disk Arm Scheduling Policies



- ***First come, first serve (FCFS):*** requests are served in the order of arrival

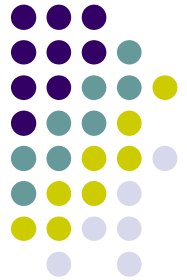
  \+ Fair among requesters

  \- Poor for accesses to random disk blocks

- ***Shortest seek time first (SSTF):*** picks the request that is closest to the current disk arm position

  \+ Good at reducing seeks

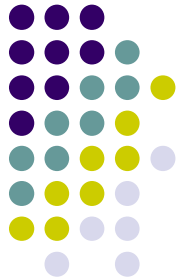
  \- May result in starvation



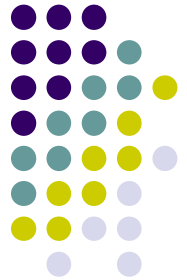
- ***SCAN***: takes the closest request in the direction of travel (an example of elevator algorithm)

  \+ no starvation

  \- a new request can wait for almost two full scans of the disk



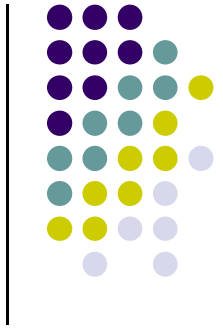
- ***Circular SCAN (C-SCAN):*** disk arm always serves requests by scanning in one direction.
  - Once the arm finishes scanning for one direction
  - Returns to the 0^{th} track for the next round of scanning



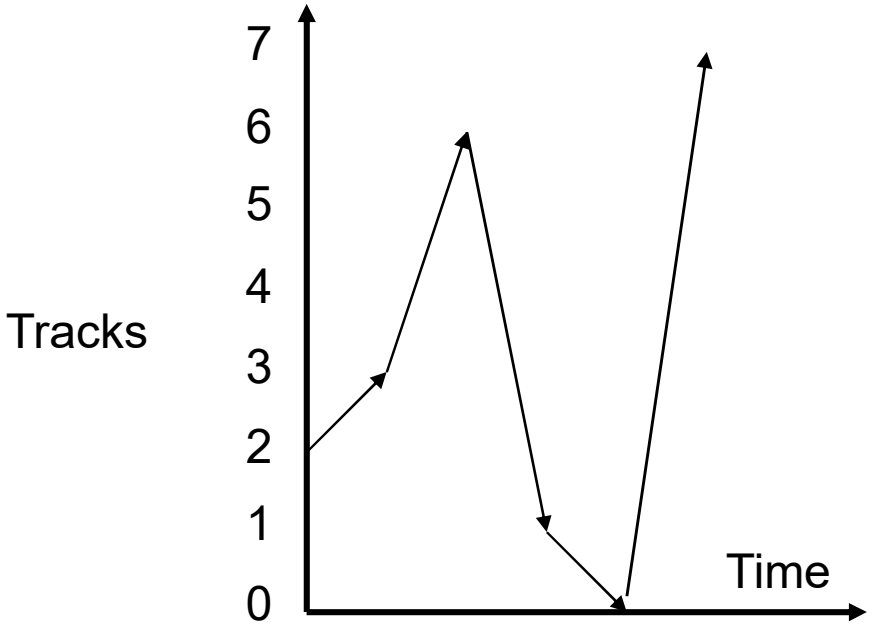
- **Question :** Schedule the disk for the following IO requests of tracks 55, 58, 39, 18, 90, 160, 150, 38, 184 with initial head position of 100 and total tracks of 200.

(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

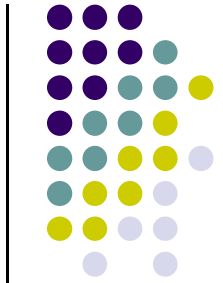
First Come, First Serve



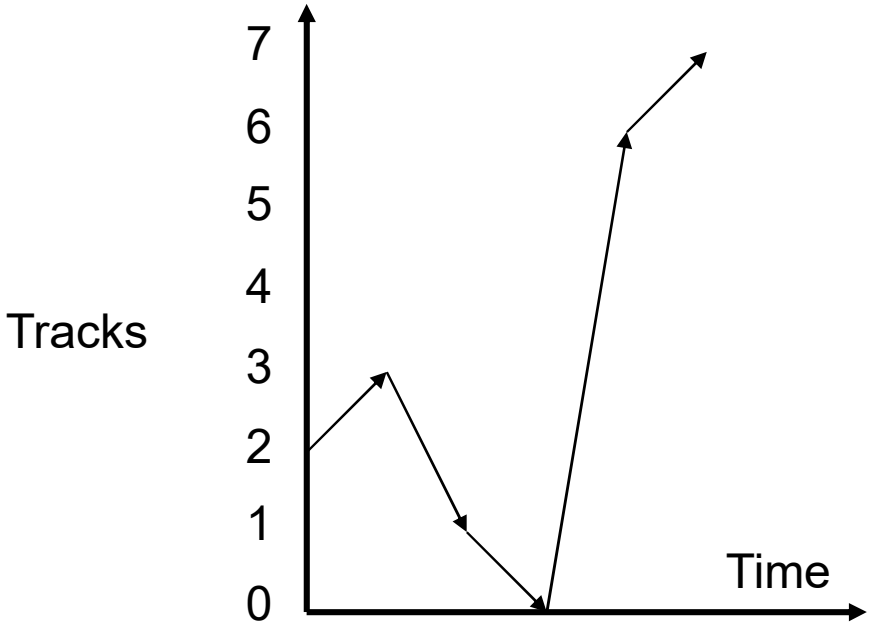
- Request queue: 3, 6, 1, 0, 7
- Head start position: 2
- Total seek distance: 1 + 3 + 5 + 1 + 7 = 17



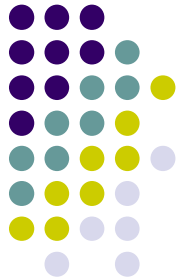
Shortest Seek Distance First



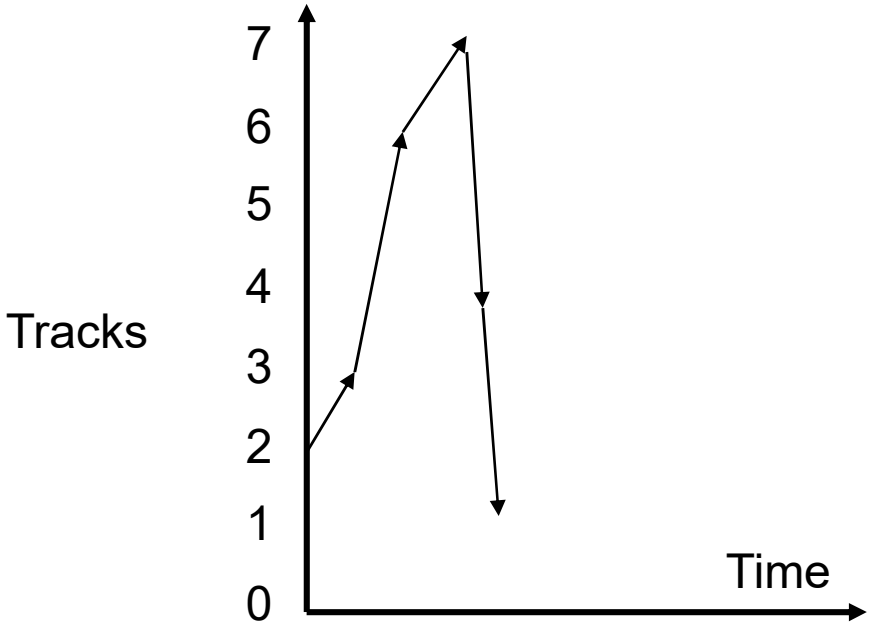
- Request queue: 3, 6, 1, 0, 7
- Head start position: 2
- Total seek distance: 1 + 2 + 1 + 6 + 1 = 10



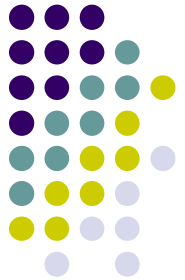
SCAN



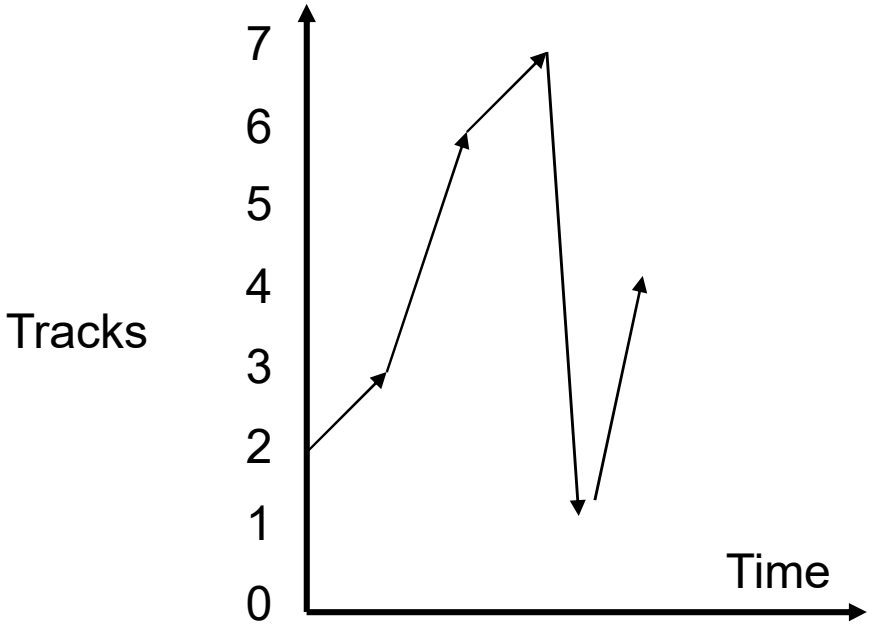
- Request queue: 3, 6, 1, 4, 7
- Head start position: 2
- Total seek distance: 1 + 3 + 1 + 3 + 3 =11



C-SCAN



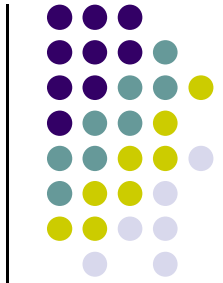
- Request queue: 3, 6, 1, 4, 7
- Head start position: 2
- Total seek distance: 1 + 3 + 1 + 6 + 3 = 14



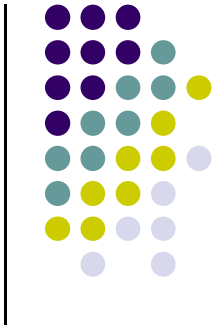
Look and C-Look

- Similar to SCAN and C-SCAN
  - the arm goes only as far as the final request in each direction, then turns around
    - Look for a request before continuing to move in a given direction.

Scheduling Algorithms



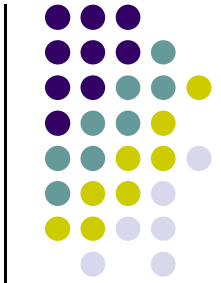
- **FCFS**
  - IO requests serviced based on seniority
- **Priority based**
  - Based on a pre-assigned priority
- **Shortest Seek Time First**
  - Select the disk IO request that requires the least movement of disk arm from current position
- **Scan(Elevator)**
  - Arm moves in one direction satisfying all requests in that en route, until it reaches last track and moves in reverse direction servicing requests
- **Look**
  - Same as Scan except that moves only till the last request



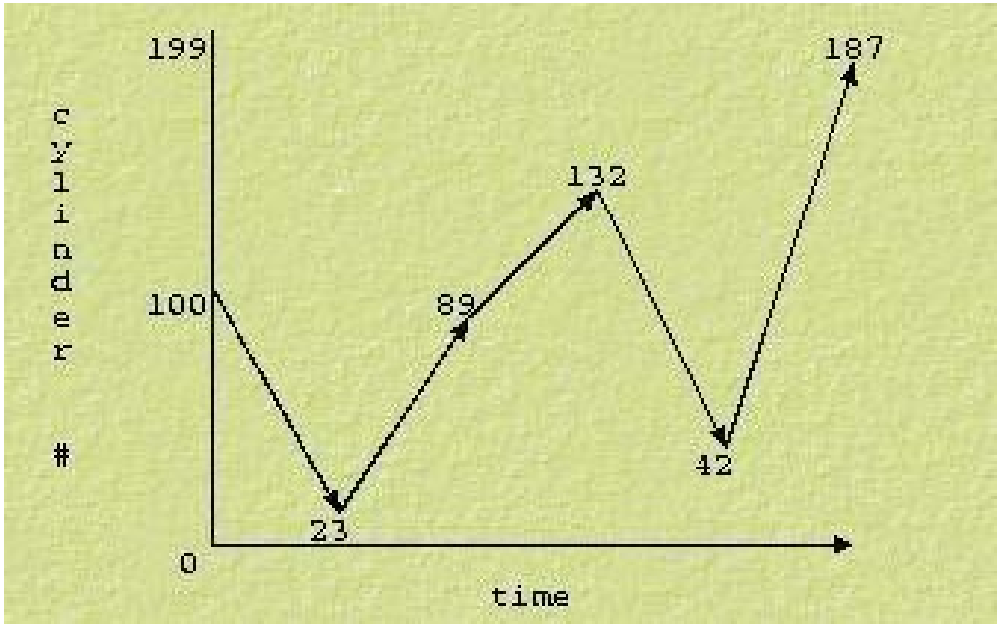
- **C-Scan**
  - Same as Scan, but after reaching the last track in one direction, jumps straight to the first track again and start to move again
- **C-Look**

- **Example:** disk queue with requests for I/O to blocks on tracks / cylinders 23, 89, 132, 42, 187
  With disk head initially at 100

FCFS

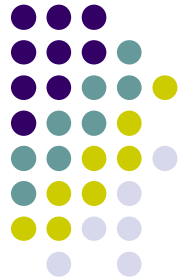


- **Given sequence** 23, 89, 132, 42, 187



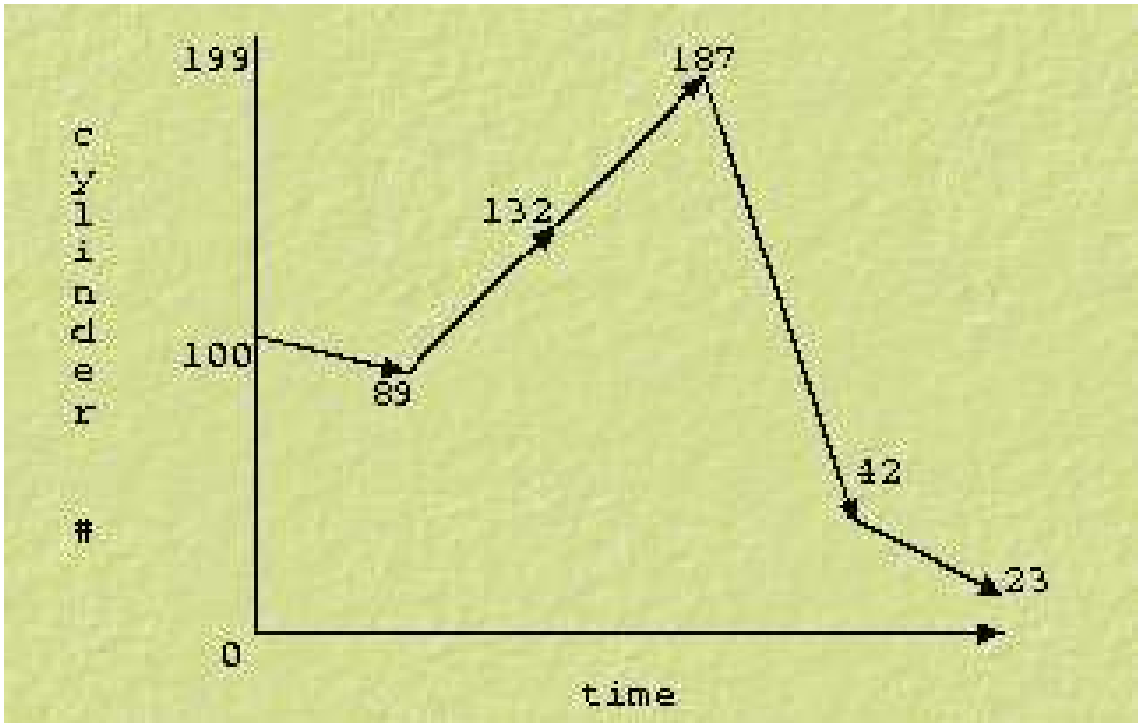
- Total track movement 77+66+43+90+145=421

SSTF Scheduling



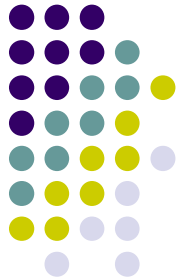
- Like SJF, select the disk I/O request that
- requires the least movement of the disk arm
- from its current position, regardless of
- direction
- reduces total seek time compared to FCFS.
- Disadvantages
  - **starvation** is possible; stay in one area of the disk
  - if very busy
  - switching directions slows things down
- Not the most optimal

Given sequence 23, 89, 132, 42, 187



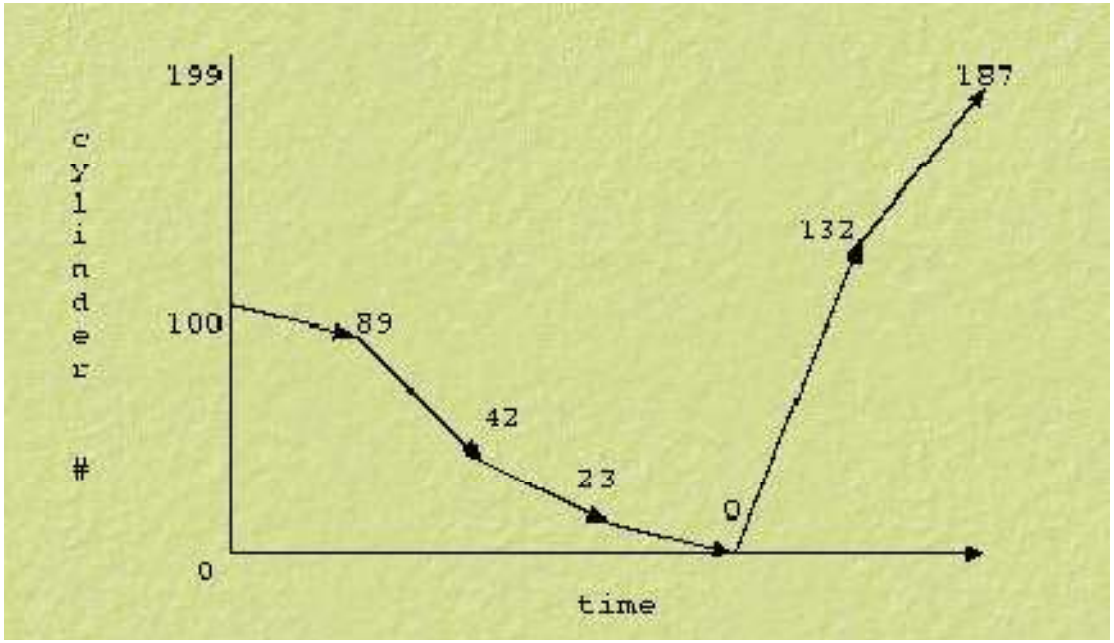
11+43+55+145+19=273

SCAN



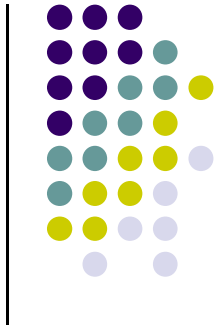
- go from the outside to the inside servicing requests and then back from the outside to the inside servicing requests
- Sometimes called the elevator algorithm
- Reduces variance compared to SSTF.
- Drawback - If a request arrives in the queue
  - just in front of the head serviced immediately
  - Just behind delayed until next cycle

23, 89, 132, 42, 187



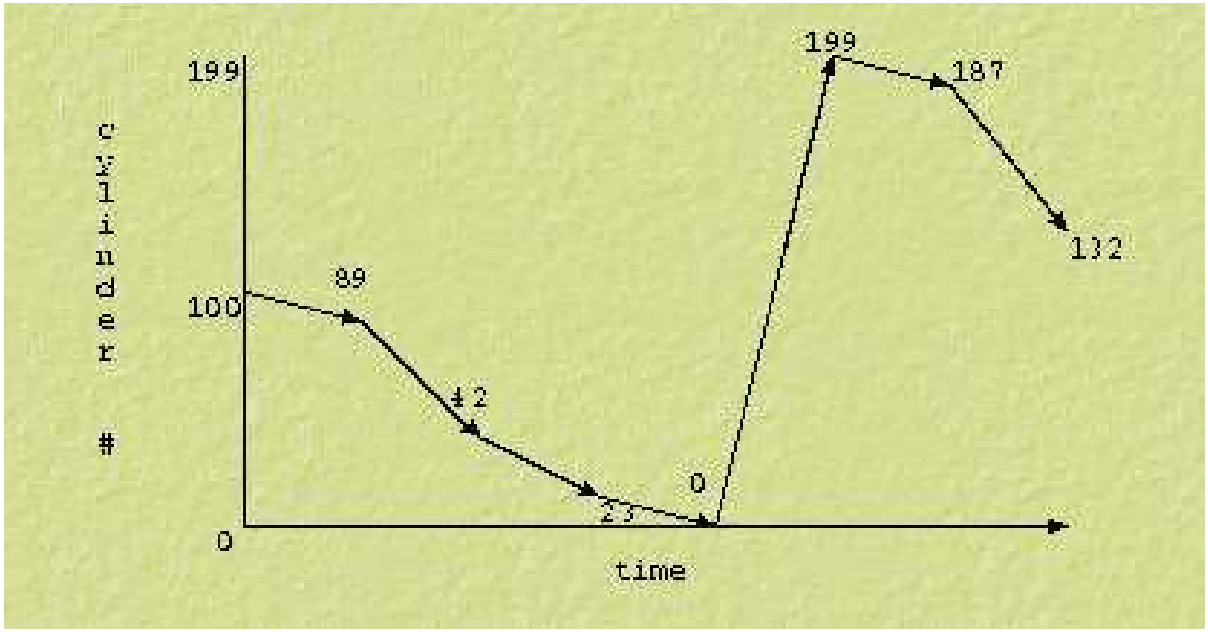
- 11+47+19+23+132+55=287

C-SCAN



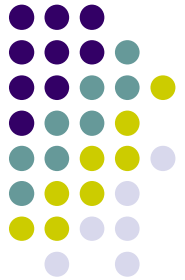
- Circular SCAN
- Moves inwards servicing requests until it reaches the innermost cylinder; then jumps to the outside cylinder of the disk without servicing any requests.
- Why C-SCAN?
  - Few requests are in front of the head, since these cylinders have recently been serviced.
  - Hence provides a more uniform wait time.

- 23, 89, 132, 42, 187



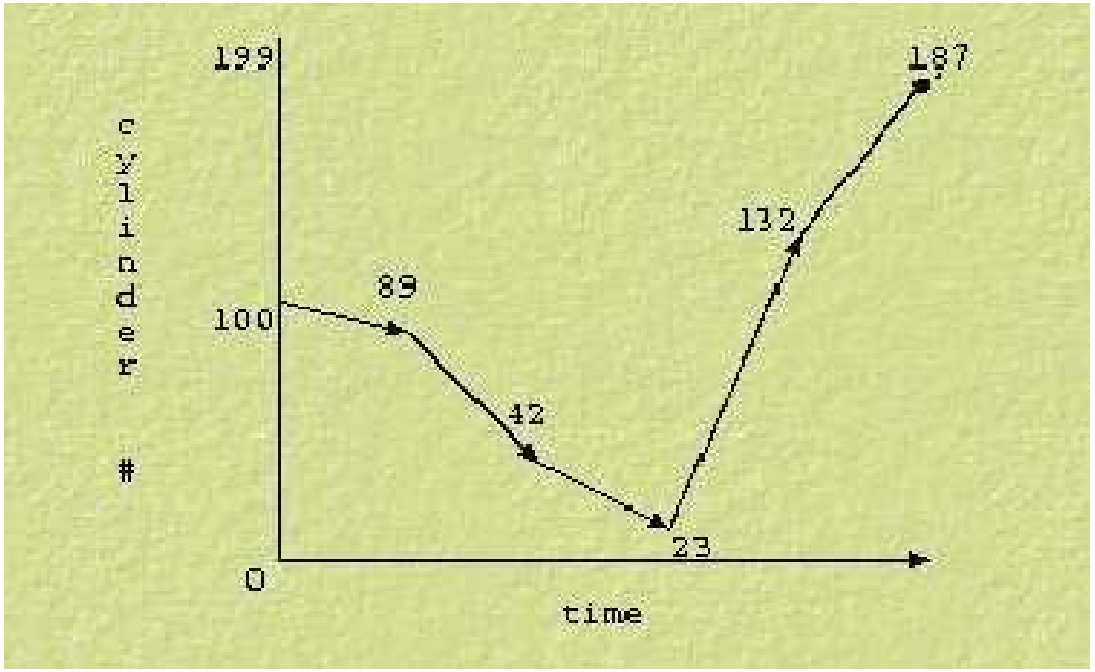
- 11+47+19+23+199+12+55=366

LOOK



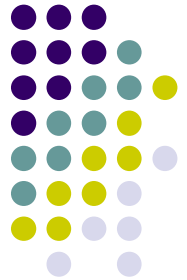
- like SCAN but stops moving inwards (or outwards) when no more requests in that direction exist

23, 89, 132, 42, 187



- 11+47+19+109+55=241
- Note : Compared to SCAN, LOOK saves going from 23 to 0 and then back. Most efficient for this sequence of requests

Which one to choose?



- Performance depends on number and type of requests.
- SSTF over FCFS.
- SCAN, C-SCAN for systems that place a
- heavy load on the disk, as they are less likely
- to cause starvation.
- Default algorithms, SSTF or LOOK