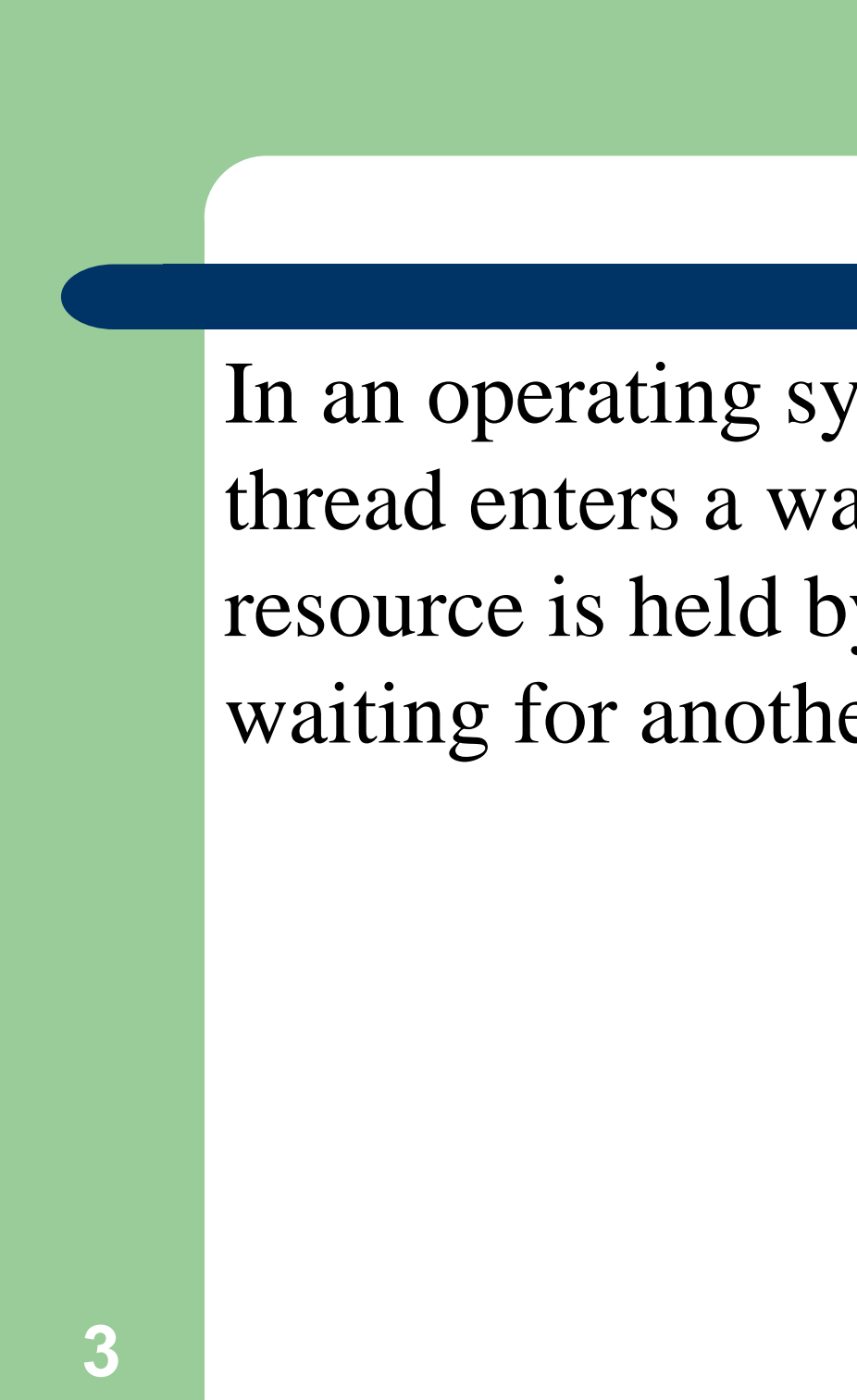


Operating System (OS)

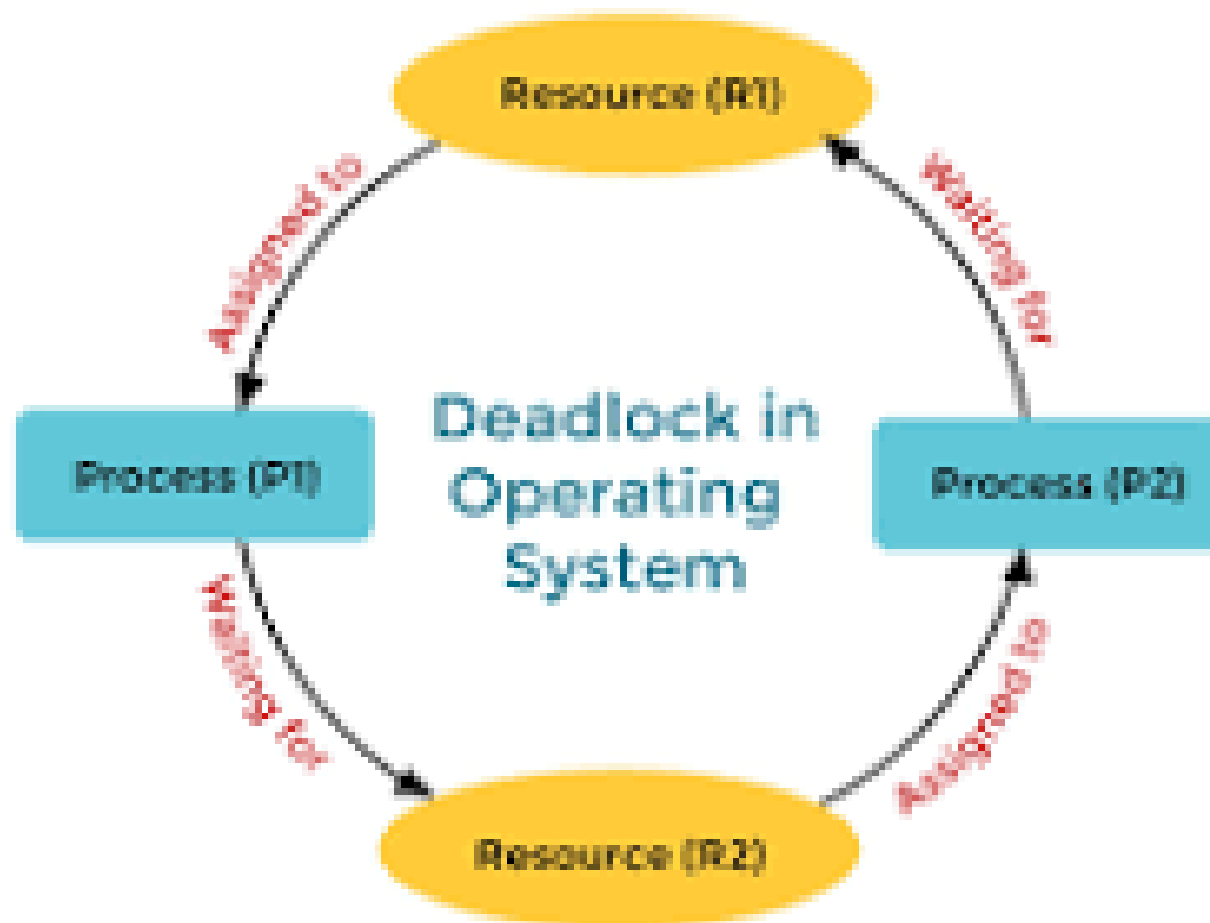


Deadlock

- A process in operating system uses resources in the following way.
- Requests a resource
- Use the resource
- Releases the resource
- A *deadlock* is a situation where a set of processes are blocked because each process is holding a resource and waiting for another resource acquired by some other process.

A green vertical bar is on the left side of the slide. A blue horizontal bar with rounded ends is positioned above the text.

In an operating system, a deadlock occurs when a process or thread enters a waiting state because a requested system resource is held by another waiting process, which in turn is waiting for another resource held by another waiting process



necessary conditions for deadlock

- **Deadlock can arise if the following four conditions hold simultaneously (Necessary Conditions)**
- ***Mutual Exclusion:*** Two or more resources are non-shareable (Only one process can use at a time)
- ***Hold and Wait:*** A process is holding at least one resource and waiting for resources.
- ***No Preemption:*** A resource cannot be taken from a process unless the process releases the resource.
- ***Circular Wait:*** A set of processes waiting for each other in circular form.

- **Methods for handling deadlock**

There are three ways to handle deadlock

- 1) Deadlock prevention or avoidance:**

- Prevention:**

- The idea is to not let the system into a deadlock state. This system will make sure that above mentioned four conditions will not arise. These techniques are very costly so we use this in cases where our priority is making a system deadlock-free. One can zoom into each category individually, Prevention is done by negating one of the above-mentioned necessary conditions for deadlock

- Prevention can be done in four different ways:
 1. Eliminate mutual exclusion
 2. Allow preemption
 3. Solve hold and Wait
 4. Circular wait

deadlock solution

- **Avoidance:**

Avoidance is kind of futuristic. By using the strategy of “Avoidance”, we have to make an assumption. We need to ensure that all information about resources that the process will need is known to us before the execution of the process. We use Banker’s algorithm (Which is in turn a gift from Dijkstra) to avoid deadlock.

- In prevention and avoidance, we get the correctness of data but performance decreases.

- **2) Deadlock detection and recovery:** If Deadlock prevention or avoidance is not applied to the software then we can handle this by deadlock detection and recovery. which consist of two phases:
 - In the first phase, we examine the state of the process and check whether there is a deadlock or not in the system.
 - If found deadlock in the first phase then we apply the algorithm for recovery of the deadlock.
 - In Deadlock detection and recovery, we get the correctness of data but performance decreases.

Recovery from Deadlock

1. Manual Intervention:

- When a deadlock is detected, one option is to inform the operator and let them handle the situation manually. While this approach allows for human judgment and decision-making, it can be time-consuming and may not be feasible in large-scale systems.

2. Automatic Recovery:

- An alternative approach is to enable the system to recover from deadlock automatically. This method involves breaking the deadlock cycle by either aborting processes or preempting resources

Recovery from Deadlock: **Process Termination:**

1. Abort all deadlocked processes:
 - This approach breaks the deadlock cycle, but it comes at a significant cost. The processes that were aborted may have executed for a considerable amount of time, resulting in the loss of partial computations. These computations may need to be recomputed later.

2. Abort one process at a time:

- Instead of aborting all deadlocked processes simultaneously, this strategy involves selectively aborting one process at a time until the deadlock cycle is eliminated. However, this incurs overhead as a deadlock-detection algorithm must be invoked after each process termination to determine if any processes are still deadlocked.

Factors for choosing the termination order:

- – The process's priority
- – Completion time and the progress made so far
- – Resources consumed by the process
- – Resources required to complete the process
- – Number of processes to be terminated
- – Process type (interactive or batch)

Recovery from Deadlock: **Resource Preemption:**

1. Selecting a victim:

- Resource preemption involves choosing which resources and processes should be preempted to break the deadlock. The selection order aims to minimize the overall cost of recovery. Factors considered for victim selection may include the number of resources held by a deadlocked process and the amount of time the process has consumed.

2. Rollback:

- If a resource is preempted from a process, the process cannot continue its normal execution as it lacks the required resource. Rolling back the process to a safe state and restarting it is a common approach. Determining a safe state can be challenging, leading to the use of total rollback, where the process is aborted and restarted from scratch.

3. Starvation prevention:

- To prevent resource starvation, it is essential to ensure that the same process is not always chosen as a victim. If victim selection is solely based on cost factors, one process might repeatedly lose its resources and never complete its designated task. To address this, it is advisable to limit the number of times a process can be chosen as a victim, including the number of rollbacks in the cost factor.

3) Deadlock ignorance: If a deadlock is very rare, then let it happen and reboot the system. This is the approach that both Windows and UNIX take. we use the ostrich algorithm for deadlock ignorance.

- In Deadlock, ignorance performance is better than the above two methods but the correctness of data.

Safe State:

- A safe state can be defined as a state in which there is no deadlock. It is achievable if:
- If a process needs an unavailable resource, it may wait until the same has been released by a process to which it has already been allocated. if such a sequence does not exist, it is an unsafe state.
- All the requested resources are allocated to the process.

Disk Scheduling Strategies

- **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

- 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.
- Two or more requests may be far from each other so this can result in greater disk arm movement.
- Hard drives are one of the slowest parts of the computer system and thus need to be accessed in an efficient manner.

- *Disk Scheduling Algorithms*
 - *FCFS (First Come First Serve)*
 - *SSTF (Shortest Seek Time First)*
 - *SCAN (Elevator Algorithm)*
 - *C-SCAN (Circular SCAN)*
 - *LOOK*
 - *C-LOOK*
 - *RSS*
 - *LIFO (Last-In First-Out)*
 - *N-Step SCAN*
 - *F-SCAN*

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:**

$\text{Disk Access Time} = \text{Seek Time} + \text{Rotational Latency} + \text{Transfer Time}$

$\text{Total Seek Time} = \text{Total head Movement} * \text{Seek Time}$

- **Disk Response Time:** Response Time is the average time spent by a request waiting to perform its I/O operation. The average *Response time* is the response time of all requests. *Variance Response Time* is the measure of how individual requests are serviced with respect to average response time. So the disk scheduling algorithm that gives minimum variance response time is better.

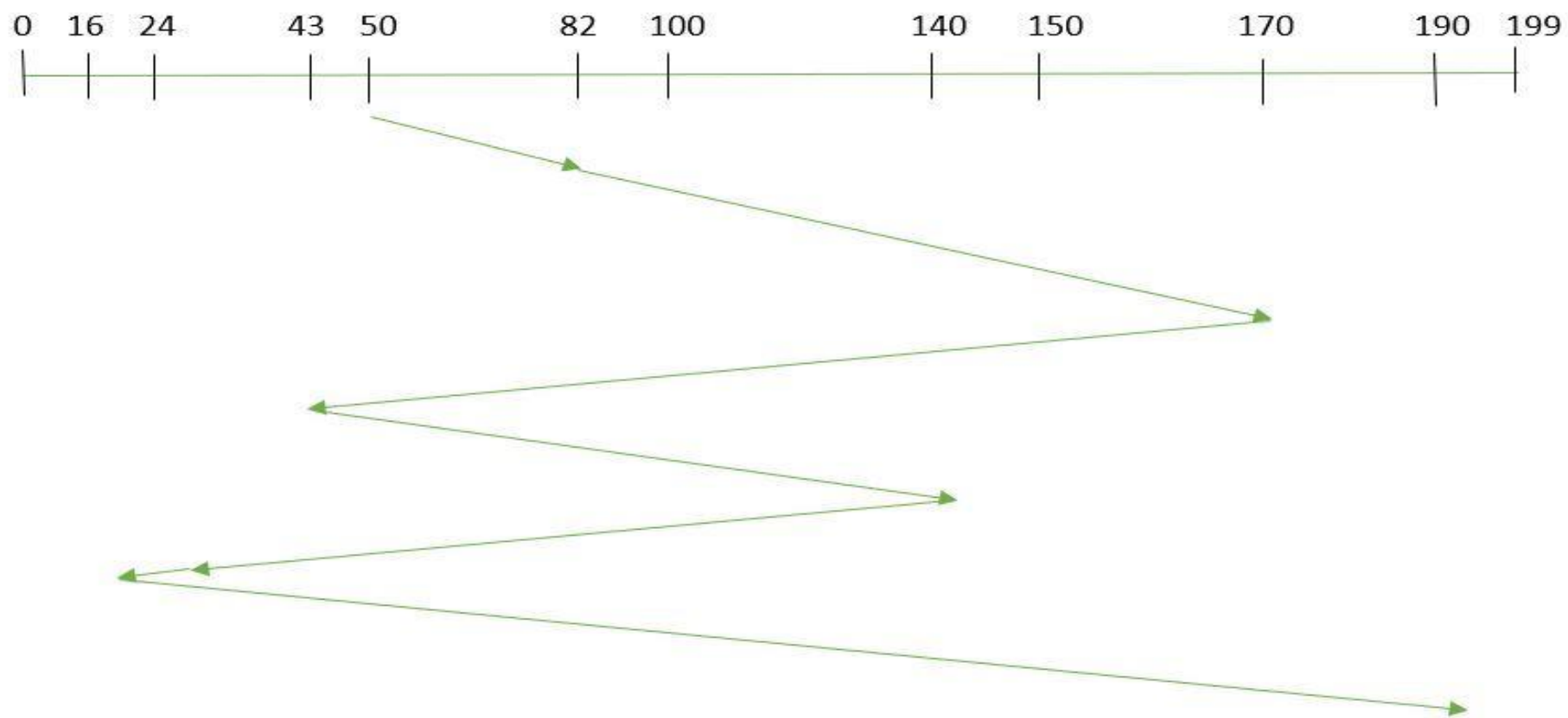
Disk Scheduling Algorithms

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.

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) =

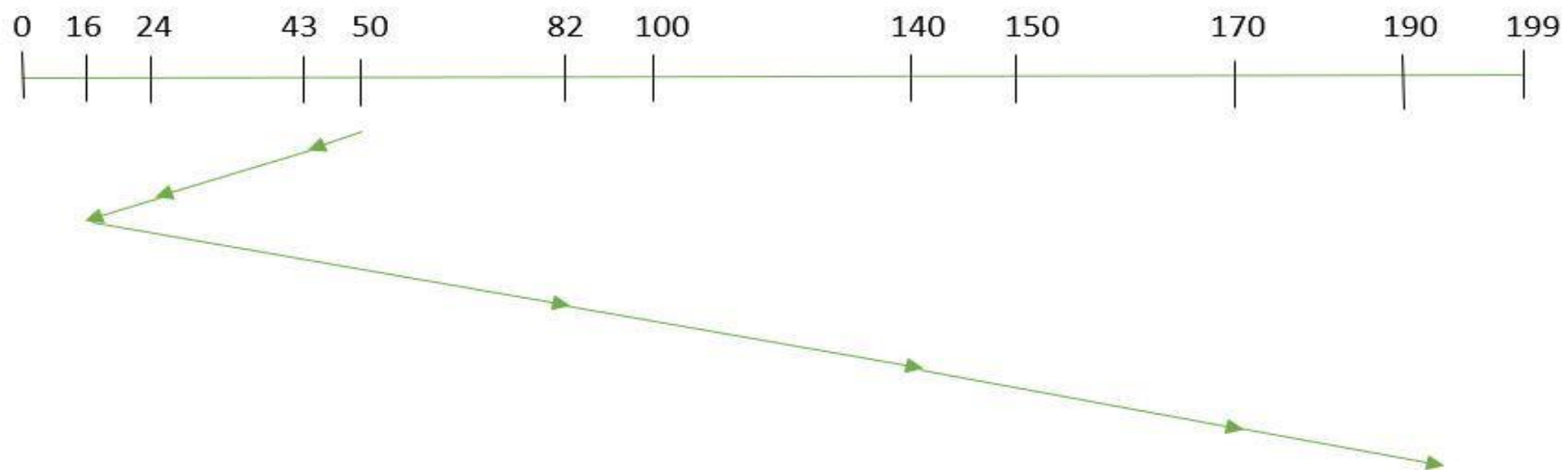
$$(82-50)+(170-82)+(170-43)+(140-43)+(140-24)+(24-16)+(190-16) = 642$$

- 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.

Example: order of request is- (82,170,43,140,24,16,190)

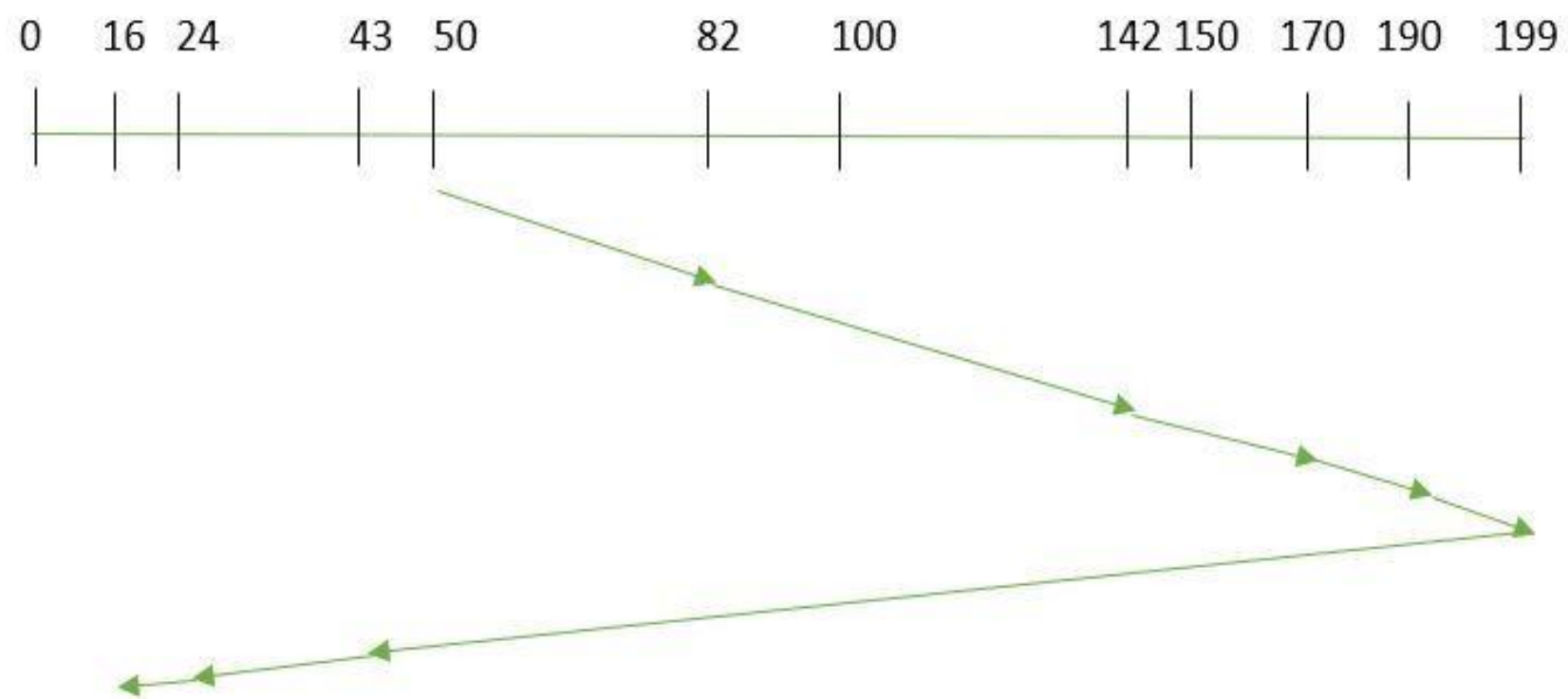
And current position of Read/Write head is: 50



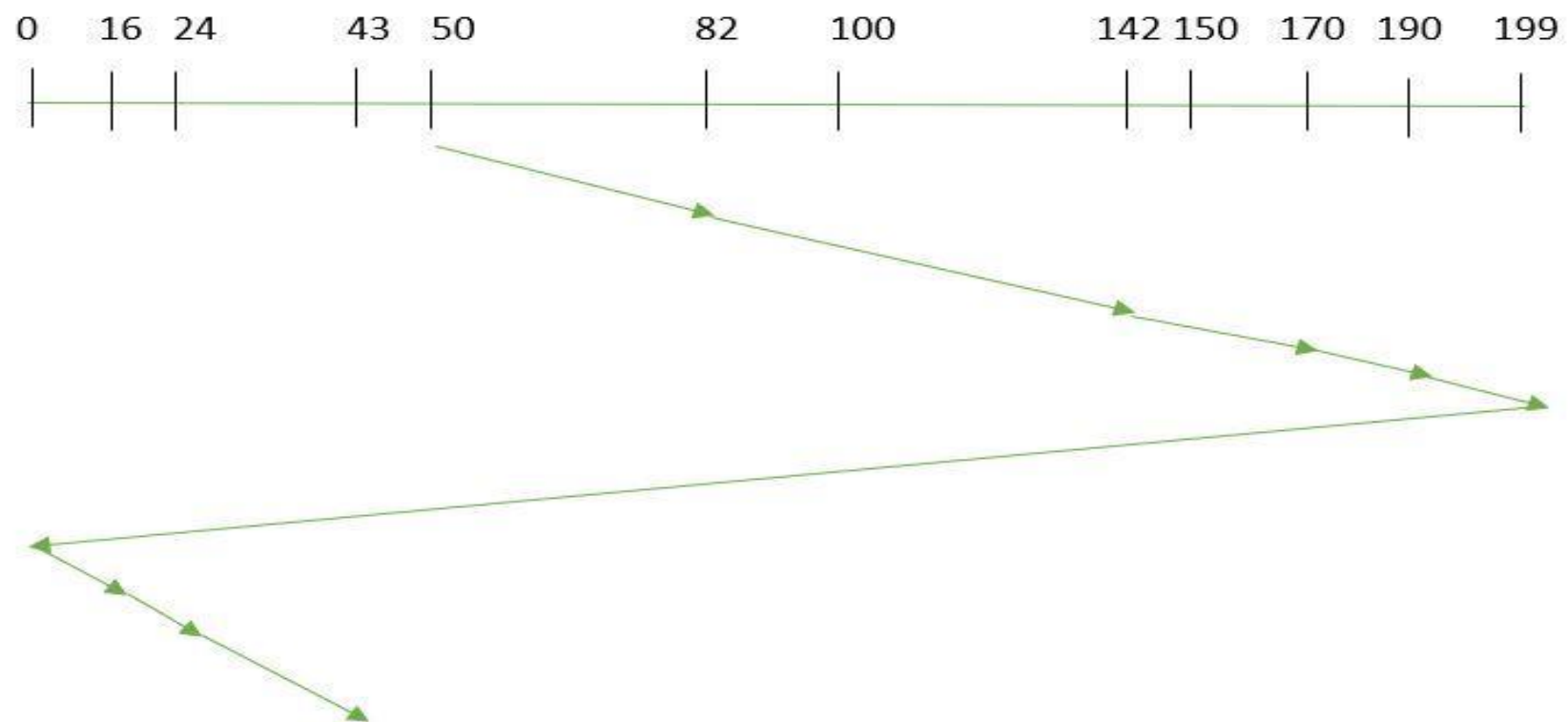
total overhead movement (total distance covered by the disk arm) =

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

- SCAN
- In the SCAN algorithm the disk arm moves in a particular direction and services the requests coming in its path and after reaching the end of the disk, it reverses its direction and again services the request arriving in its path. So, this algorithm works as an elevator and is hence also known as an elevator algorithm. As a result, the requests at the midrange are serviced more and those arriving behind the disk arm will have to wait.

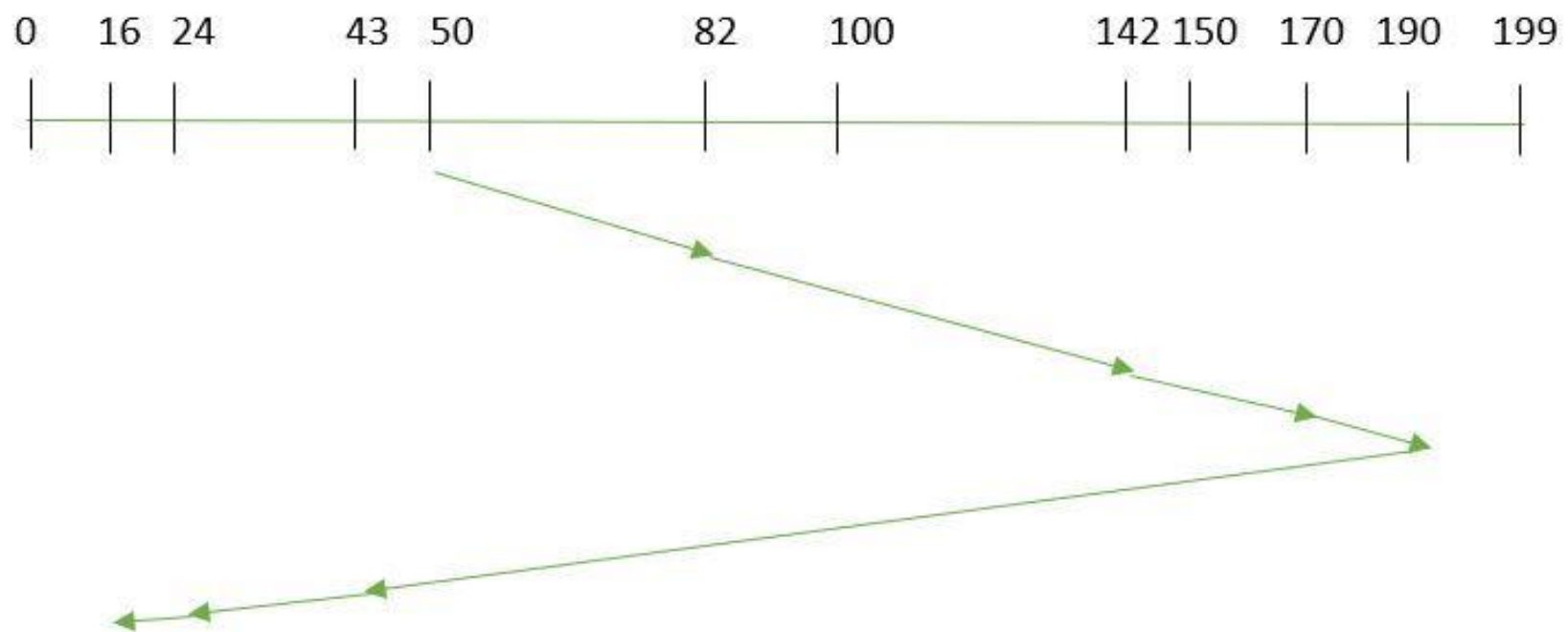


- C-SCAN
- In the SCAN algorithm, the disk arm again scans the path that has been scanned, after reversing its direction. So, it may be possible that too many requests are waiting at the other end or there may be zero or few requests pending at the scanned area.
- These situations are avoided in the CSCAN algorithm in which the disk arm instead of reversing its direction goes to the other end of the disk and starts servicing the requests from there. So, the disk arm moves in a circular fashion and this algorithm is also similar to the SCAN algorithm hence it is known as C-SCAN (Circular SCAN).



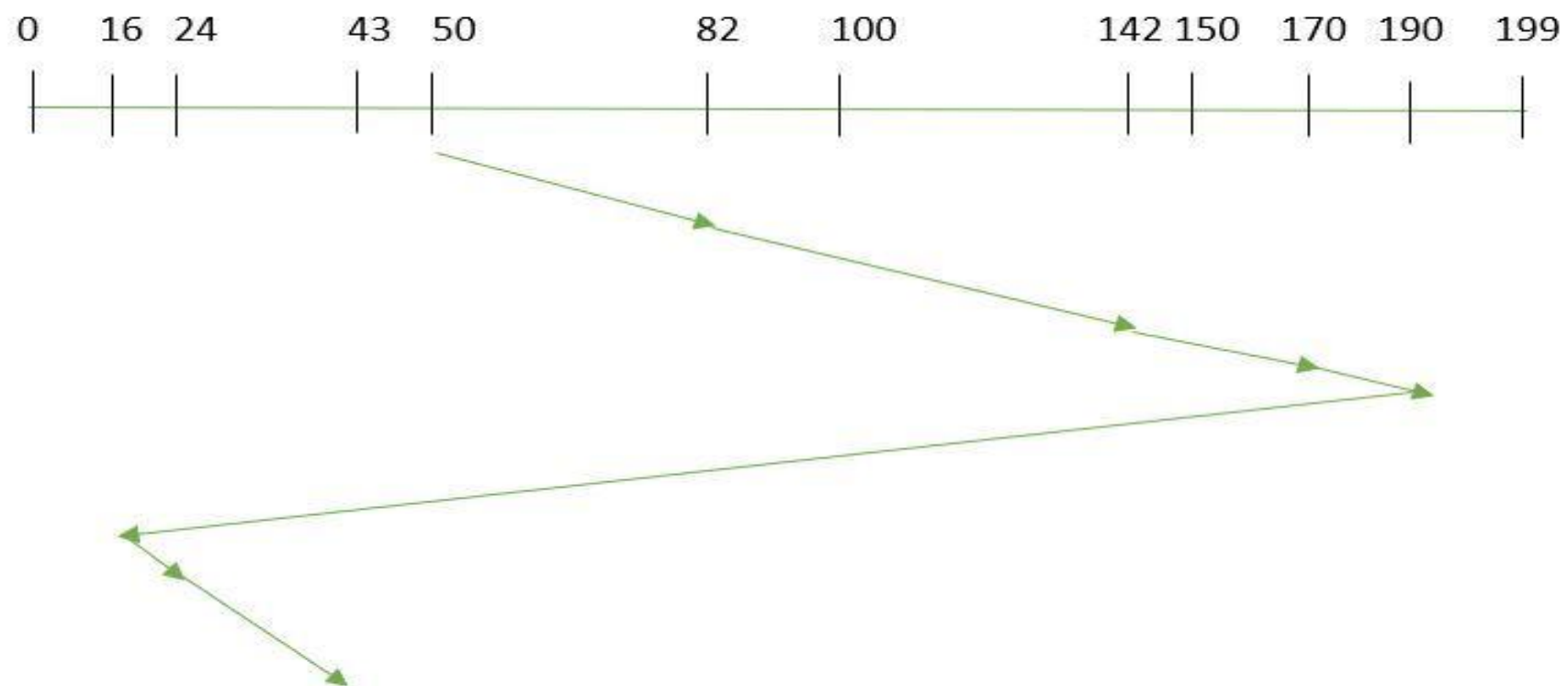
LOOK

LOOK Algorithm is similar to the SCAN disk scheduling algorithm except for the difference that the disk arm in spite of going to the end of the disk goes only to the last request to be serviced in front of the head and then reverses its direction from there only. Thus it prevents the extra delay which occurred due to unnecessary traversal to the end of the disk.



C-LOOK

- As LOOK is similar to the SCAN algorithm, in a similar way, C-LOOK is similar to the CSCAN disk scheduling algorithm. In CLOOK, the disk arm in spite of going to the end goes only to the last request to be serviced in front of the head and then from there goes to the other end's last request. Thus, it also prevents the extra delay which occurred due to unnecessary traversal to the end of the disk.



Rotational optimization

In early hard drives, the dominant component of access time was seek time. So, older systems concentrated on disk scheduling algorithms for minimizing seek times. Today's hard disks exhibit seek times and average latencies of the same order of magnitude. Rotational optimization can improve the performance in case there are a number of requests to small pieces of data randomly distributed throughout the disk's cylinders. But the processes that access data sequentially tend to access entire tracks of data and thus do not benefit much from rotational optimization. Recently developed strategies attempt to optimize disk performance by reducing rotational latency

- *Seek time* is measured defines the amount of time it takes a hard drive's read/write head to find the physical location of a piece of data on the disk. *Latency* is the average time for the sector being accessed to rotate into position under ahead, after a completed seeks.

What is Rotational Latency?

- The disk is divided into many circular tracks, and these tracks are further divided into blocks known as sectors. To access data, the actuator arm moves the read-write head over the platter to a particular track while the platter spins to position the requested sector under the read-write head. The time taken by the platter to rotate and position the data under the read-write head is called *rotational latency*.
- This latency depends on the rotation speed of the spindle and is measured in *milliseconds*. The average rotational latency for a disk is half the amount of time it takes for the disk to make one revolution.

Important Formula

Rotational Latency = (Angle between current sector and required sector) / (Rotational frequency)

And

Average Rotational Latency = $(1/2) * \text{One rotation time}$

Example

Suppose, Speed = 2400 RPM (Rotation Per Minute)

Then, 2400 Rotation = 1 min

1 Rotation = $60 \text{ sec} / 2400$ // 1 min = 60 sec

1 Rotation = $1/40 \text{ sec}$

One Rotation Time = $1/40 \text{ sec}$

Average Rotational Latency = $(1/2) * (1/40)$

= $1/80 \text{ sec}$

- What is Disk Access Time?
- Disk Access Time is defined as the total time required by the computer to process a read/write request and then retrieve the required data from the disk storage.
- There are two components in disk access time. The first component is *seek time* which occurs when the read and write arm seeks the desired track. The second component is latency or wait *time* which occurs when the head write arm waits for the desired sector on the track to spin around.
- Access to the data on disks is measured in terms of milliseconds. However, this is much slower than the processing speeds of CPUs. Although I/O is still slow, it cannot match the speed improvements of modern processors. Disk Access Time is divided into two parts:
 - Access Time
 - Data Transfer Time

Formula

we can calculate the disk access time by using the following formula.

Disk Access Time = Access Time + Data Transfer Time

OR

Disk Access Time = Seek time + Rotational delay + Transfer time + Controller overhead + Queuing delay

Average disk access time is calculated as

Average disk access time = Average seek time + Average rotational delay + Transfer time + Controller

System Consideration

Make sure that your system meets the minimum operating system requirements. In a production environment with numerous concurrent users, multiple processors help increase performance. In addition to using faster processors to speed up most operations, there are other areas to consider when you want to improve operating system performance.

Caching and Buffering

Caching :

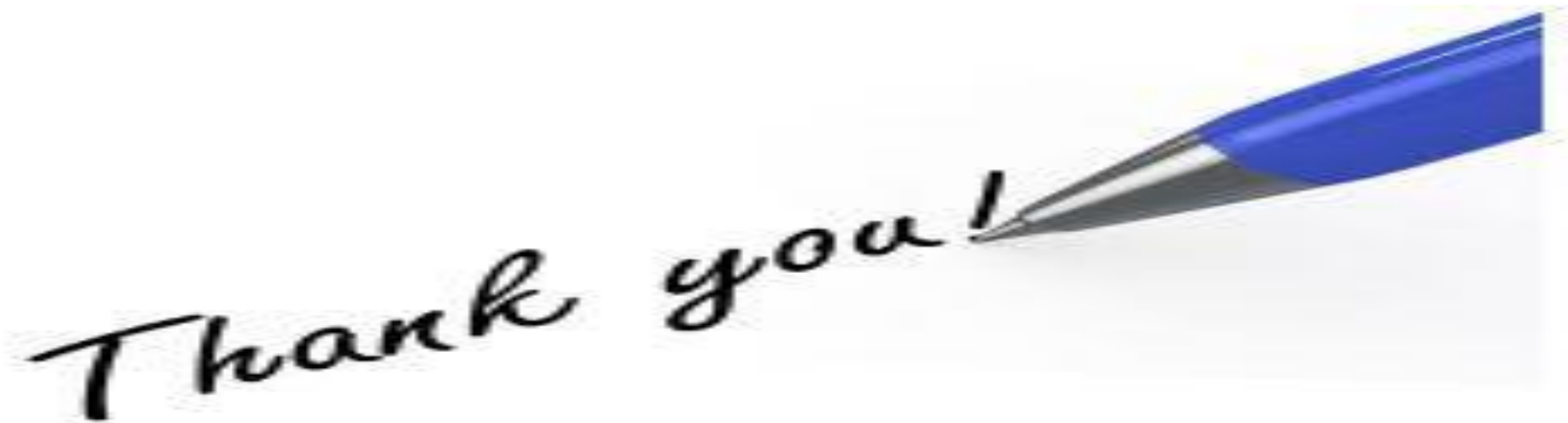
Caching is storing data in a separate disk (very fast speed disk). The data which is to be used many times results in wastage of time if it is in hard disk, but storing the data in cache reduces this time wastage. cache is used in system to speed up the access of data frequently used.

Buffering :

In computer system when the speed in which data is received and the speed in which data is processed are different, then there we use the buffer. Buffer is a memory space which stores the input data and pass it on to the system according to this speed in this way there is no need to hold the input device until it is processed. simply the data will be stored in buffer and then used by the system. The buffer can be of any type, hardware or software, but in general software buffer are used widely.

Assignment 3

1. Explain necessary conditions for deadlock.
2. Define Bankers algorithms with example.
3. What are steps taken for deadlock recovery.
4. Explain any 3 Disk Scheduling Strategies with example.
5. Differentiate between Caching and Buffering.

A blue ballpoint pen is shown in the process of writing the words "Thank you!" in a cursive script on a white, rectangular card. The pen is positioned at the end of the word "you!", with its tip touching the paper. The card is placed on a light-colored surface, and the background is a soft, out-of-focus white.

Thank you!