Sistemi Operativi

Corso di Laurea in Informatica

a.a. 2019-2020

SAPIENZA
UNIVERSITÀ DI ROMA

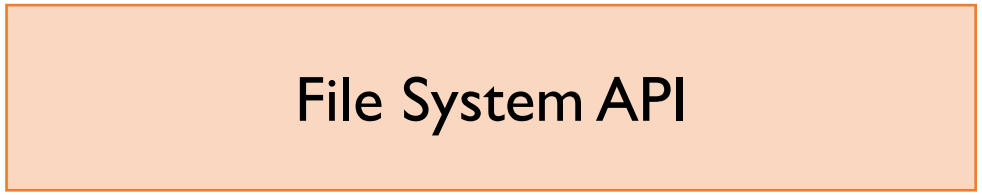
Gabriele Tolomei

Dipartimento di Informatica

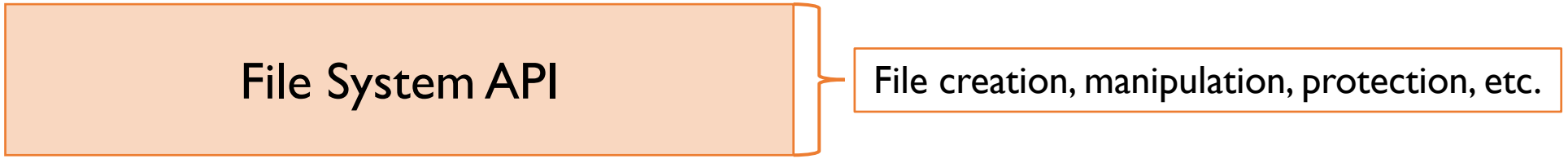
Sapienza Università di Roma

tolomei@di.uniroma1.it

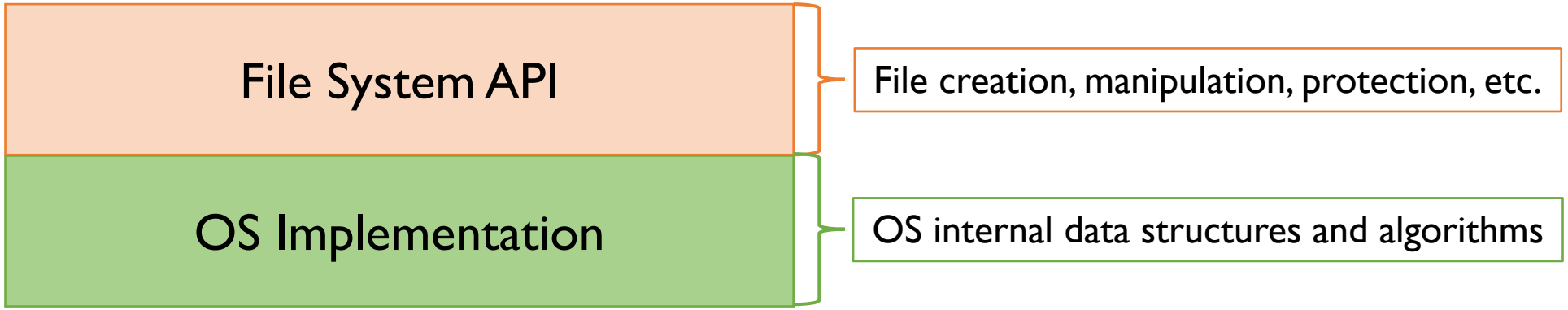
File System's Logical View



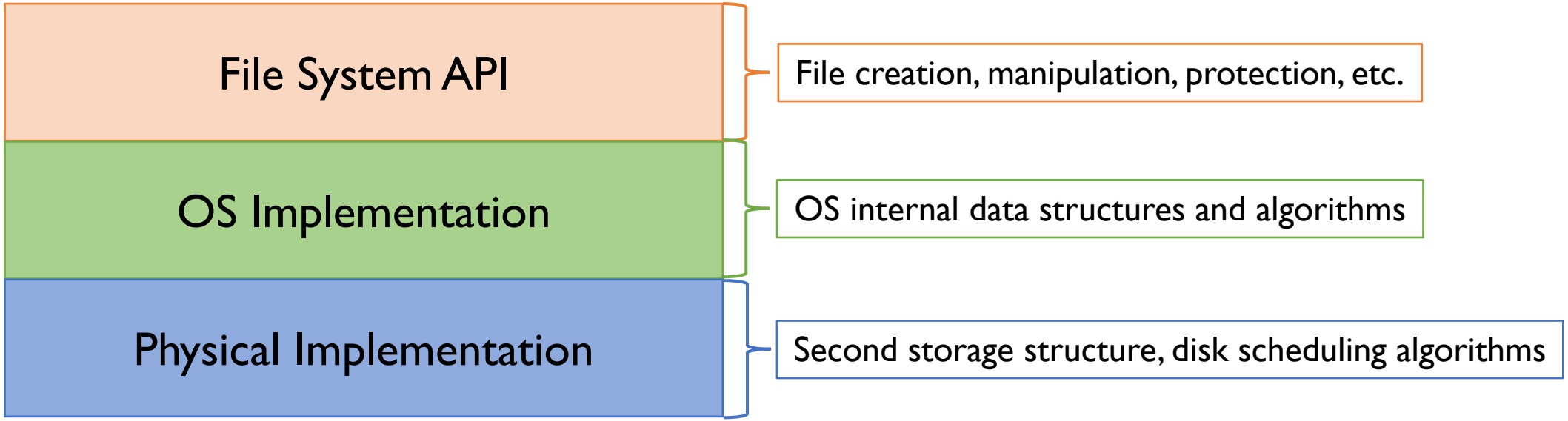
File System's Logical View



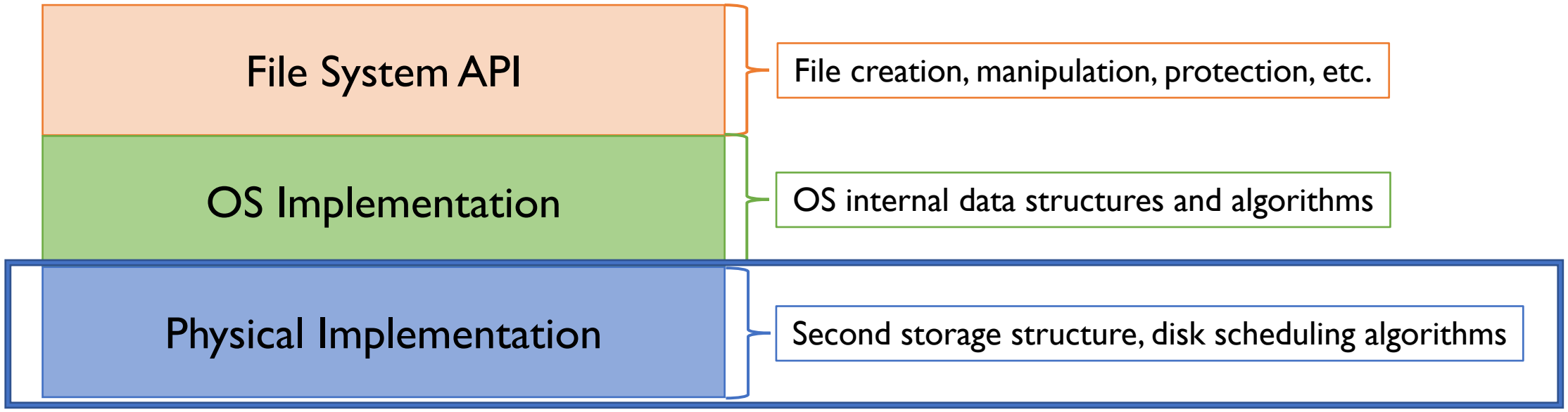
File System's Logical View



File System's Logical View



File System's Logical View



Overview of Mass-Storage Structure

3 categories of mass-storage devices

Overview of Mass-Storage Structure

3 categories of mass-storage devices

Magnetic Disk

Overview of Mass-Storage Structure

3 categories of mass-storage devices

Magnetic Disk

Solid-State Disk

Overview of Mass-Storage Structure

3 categories of mass-storage devices

Magnetic Disk

Solid-State Disk

Magnetic Tapes

Overview of Mass-Storage Structure

3 categories of mass-storage devices

Magnetic Disk

Solid-State Disk

Magnetic Tapes

Secondary Storage

Overview of Mass-Storage Structure

3 categories of mass-storage devices

Magnetic Disk

Solid-State Disk

Magnetic Tapes

Secondary Storage

Tertiary Storage

Overview of Mass-Storage Structure

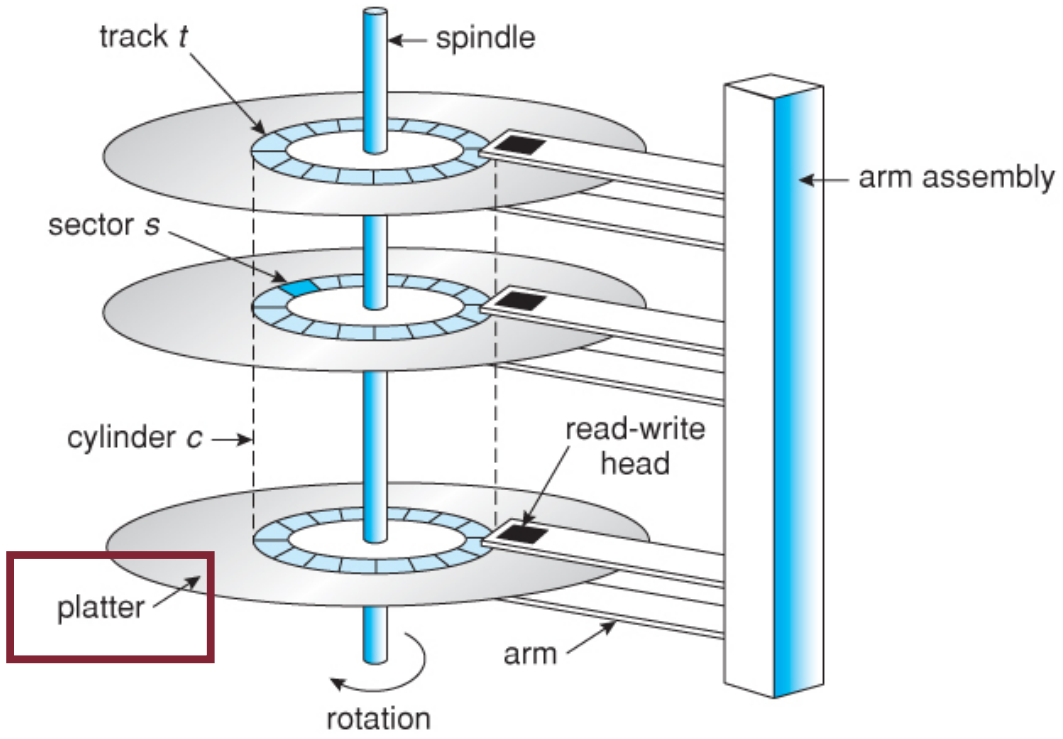
3 categories of mass-storage devices

Magnetic Disk

Solid-State Disk

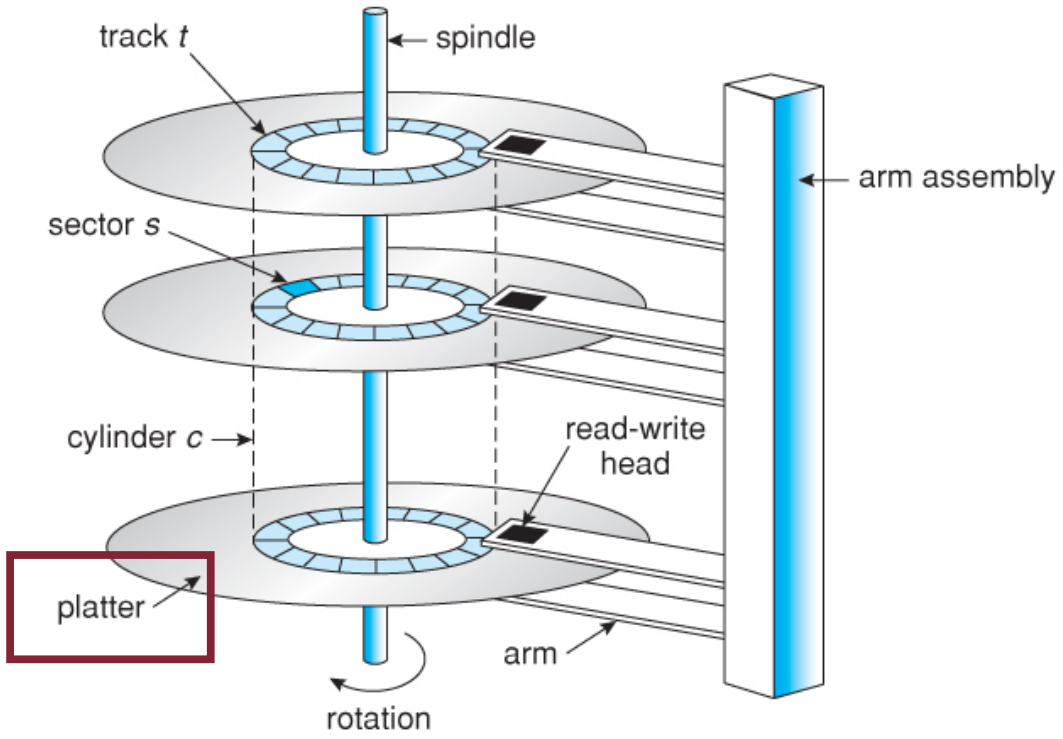
Magnetic Tapes

Magnetic Disks: Structure



One or more **platters** covered with **magnetic** media

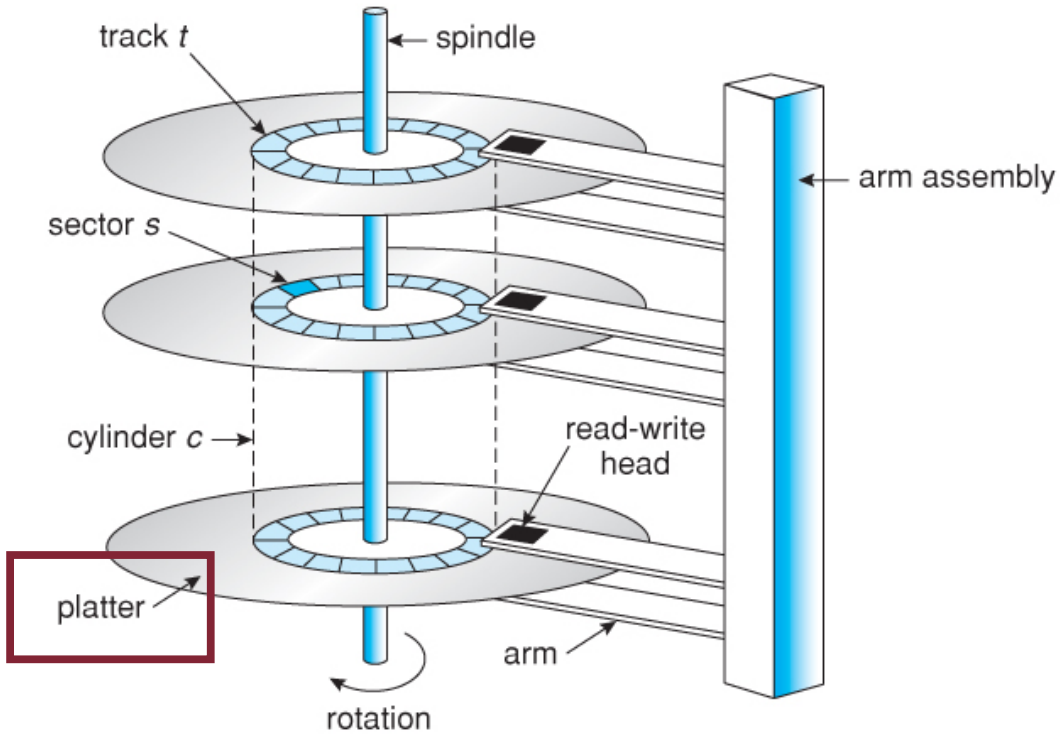
Magnetic Disks: Structure



One or more **platters** covered with **magnetic** media

Hard disk
rigid metal

Magnetic Disks: Structure

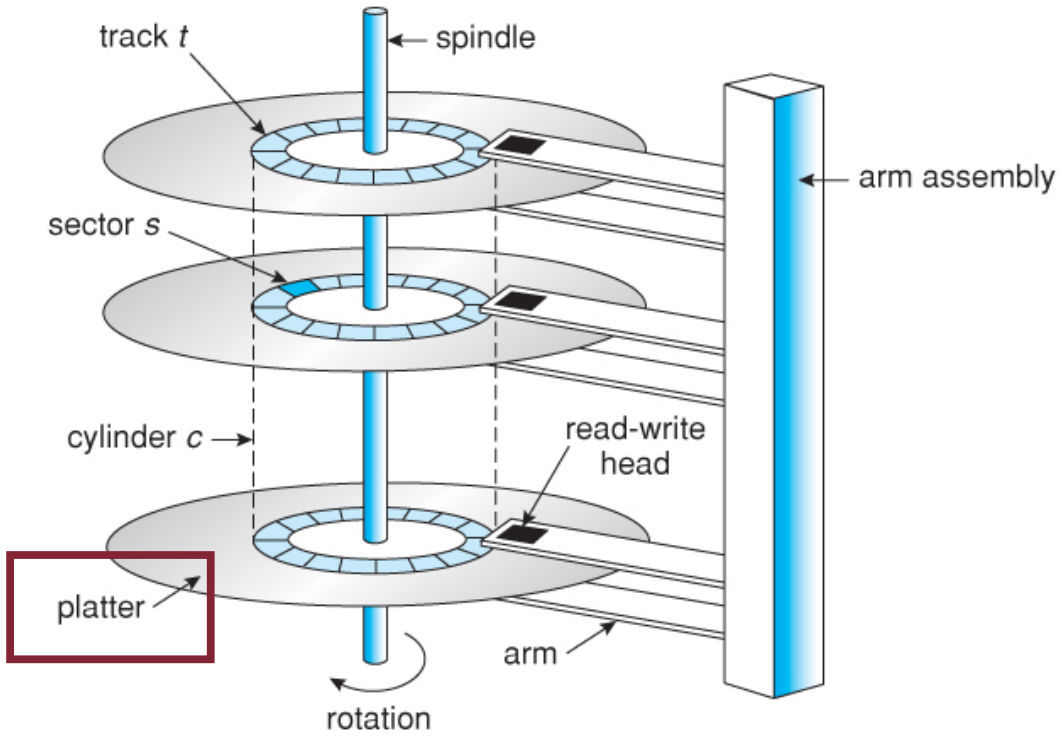


One or more **platters** covered with **magnetic** media

Hard disk
rigid metal

Floppy disk
flexible plastic

Magnetic Disks: Tracks and Cylinders



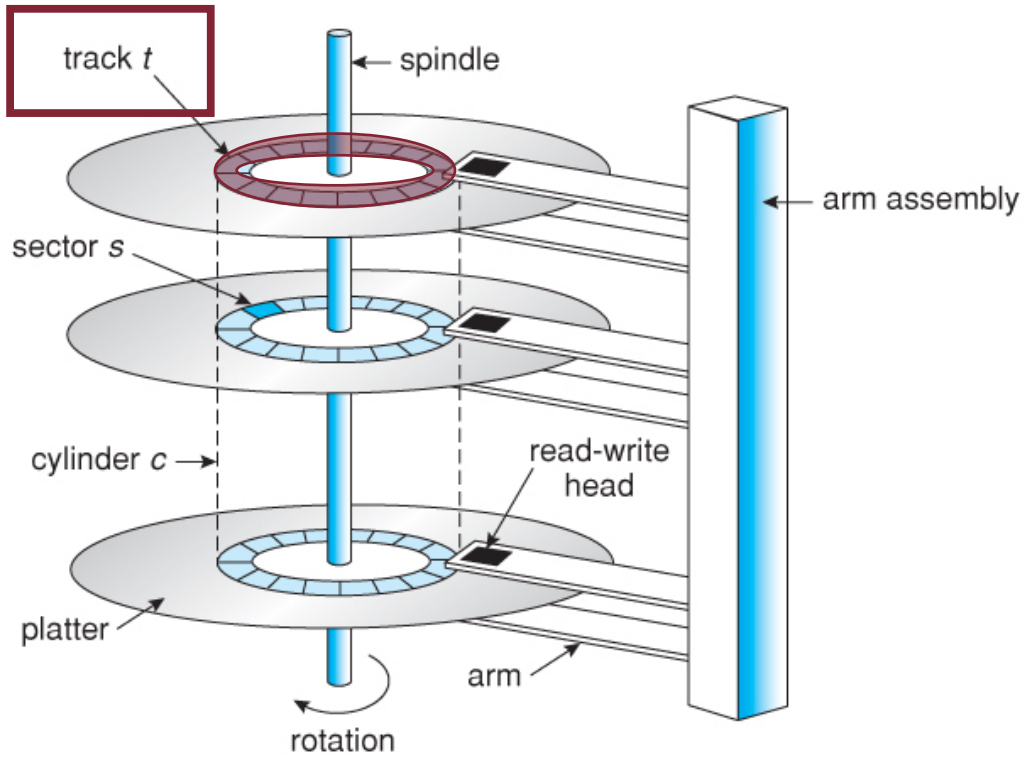
One or more **platters** covered with **magnetic** media

Hard disk
rigid metal

Floppy disk
flexible plastic

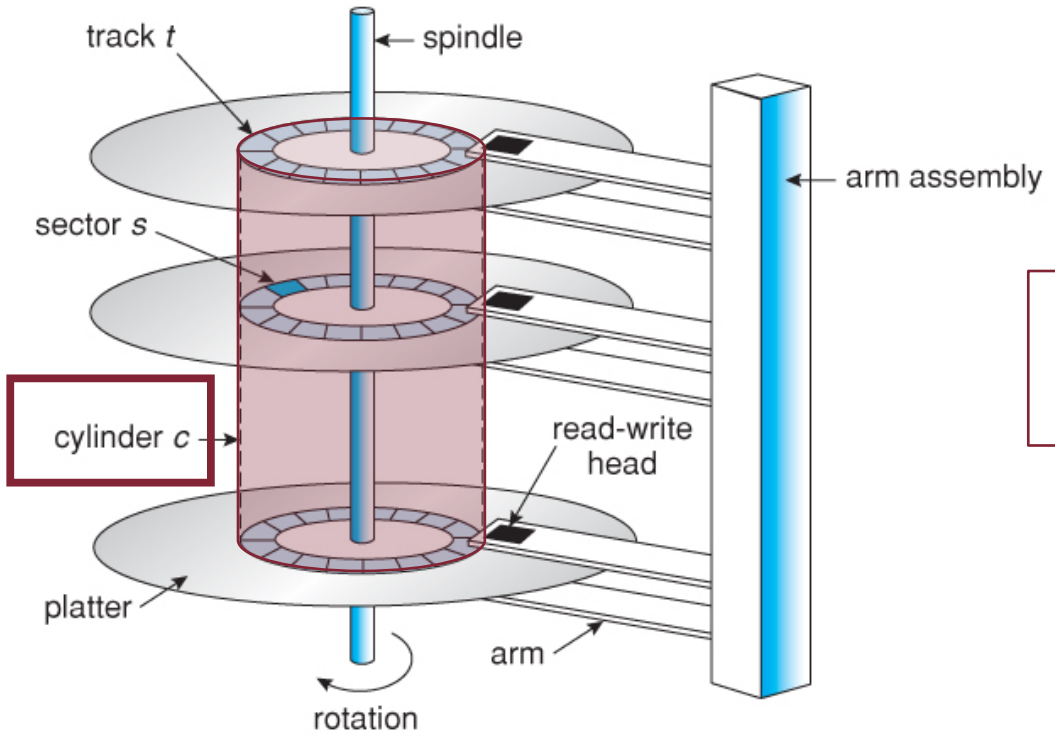
Each platter has **2** working **surfaces**

Magnetic Disks: Tracks and Cylinders



Each surface is divided into a number of concentric rings, called **tracks**

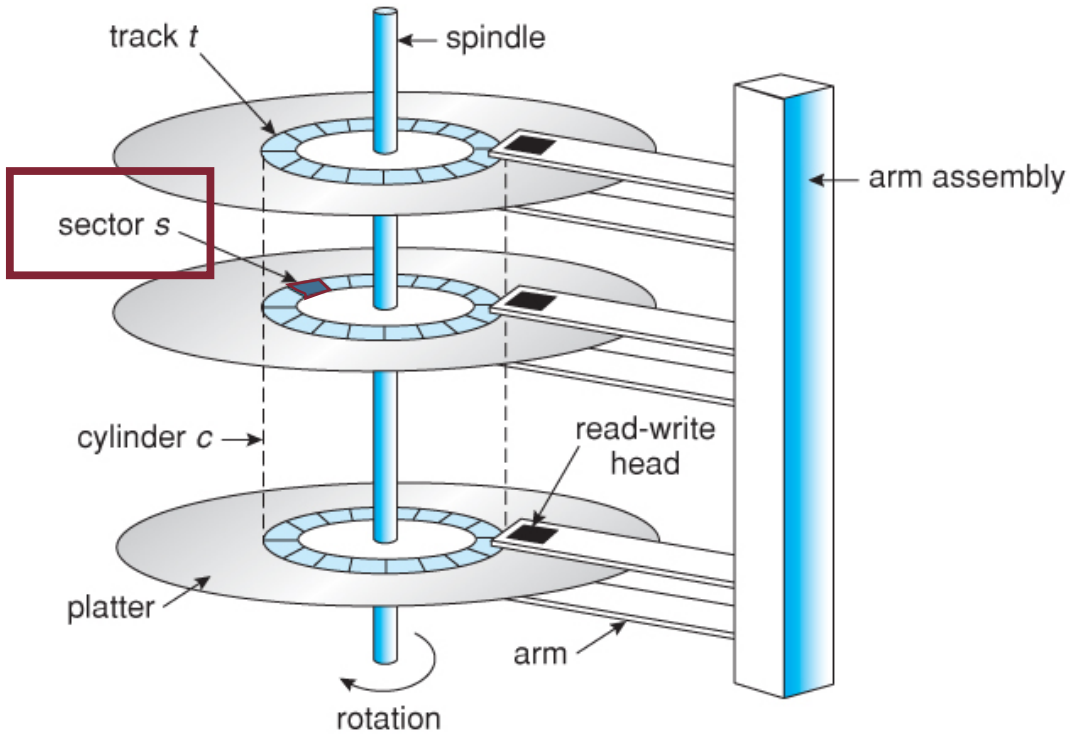
Magnetic Disks: Tracks and Cylinders



Each surface is divided into a number of concentric rings, called **tracks**

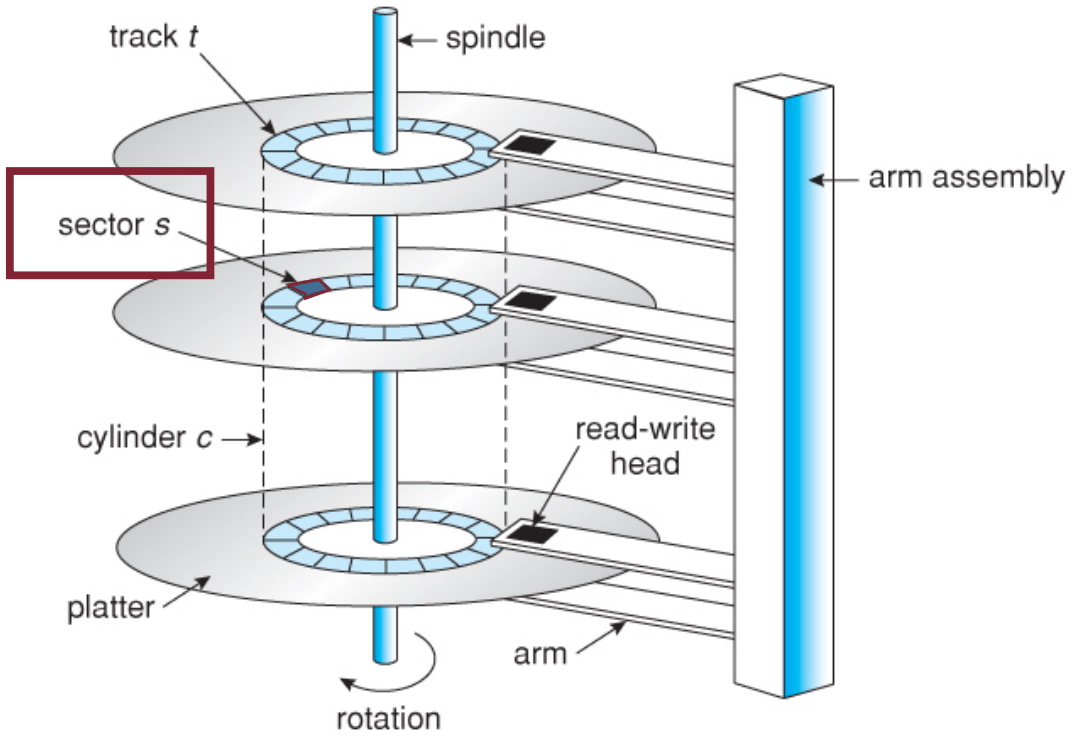
The set of all tracks that are the same distance from the edge of the platter is called a **cylinder**

Magnetic Disks: Sectors



Each track is further divided into **sectors**

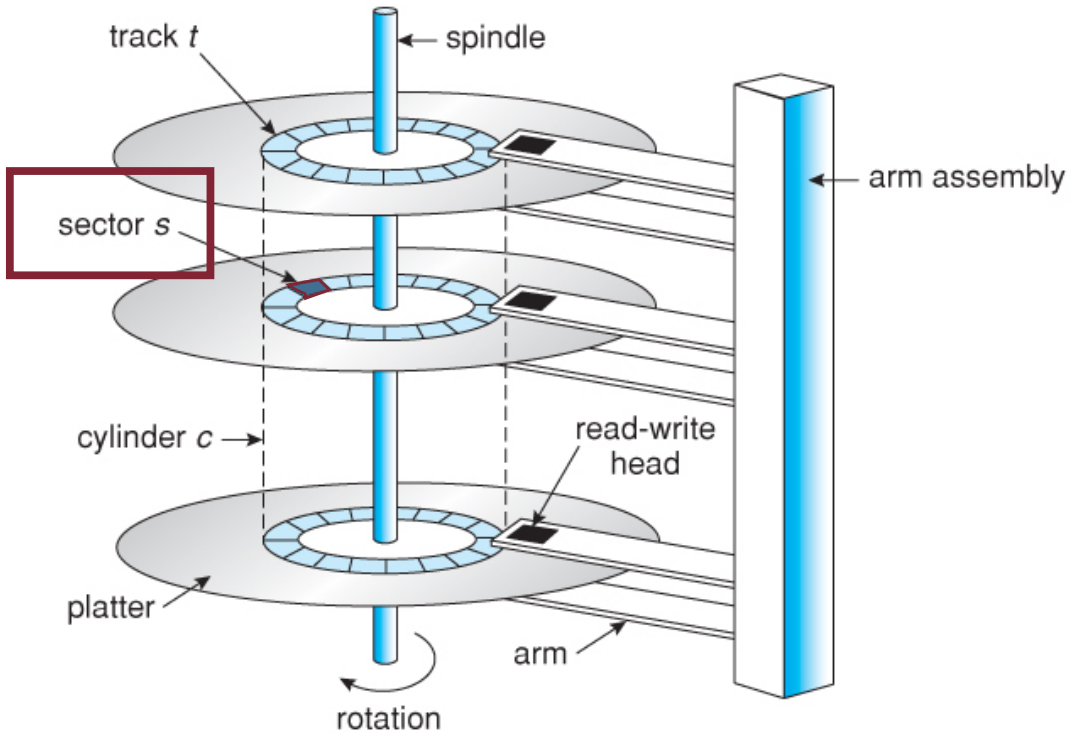
Magnetic Disks: Sectors



Each track is further divided into **sectors**

Each sector usually contains 512 bytes

Magnetic Disks: Sectors

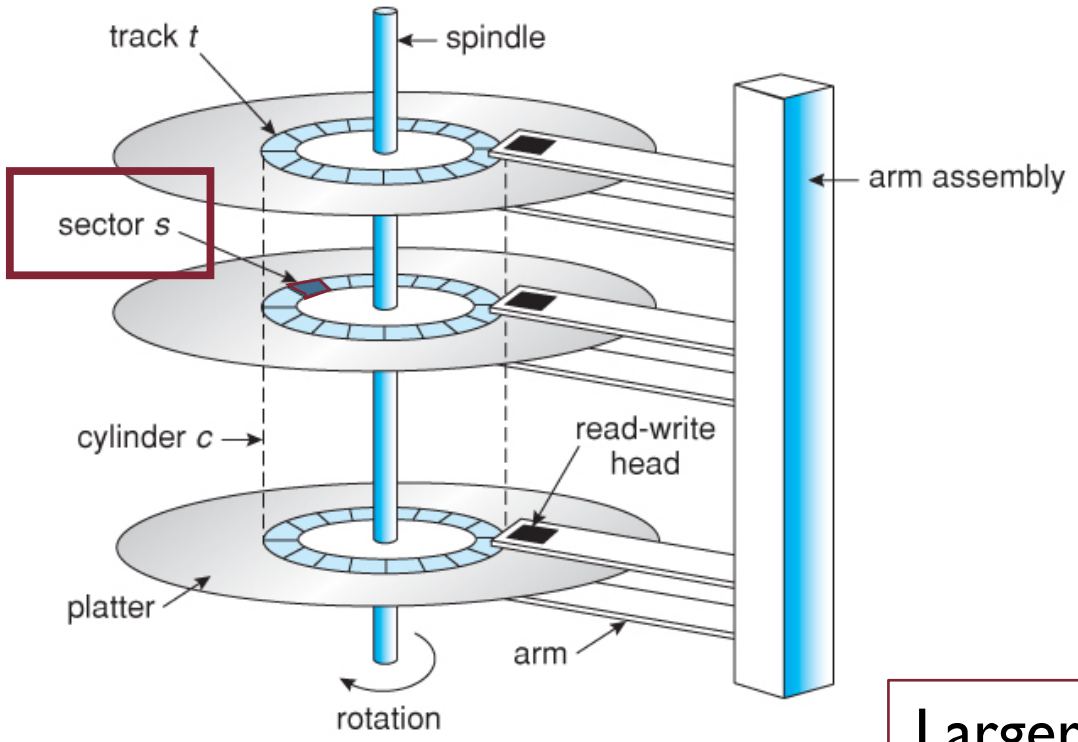


Each track is further divided into **sectors**

Each sector usually contains 512 bytes

Sectors also include a header and a trailer, and checksum information

Magnetic Disks: Sectors



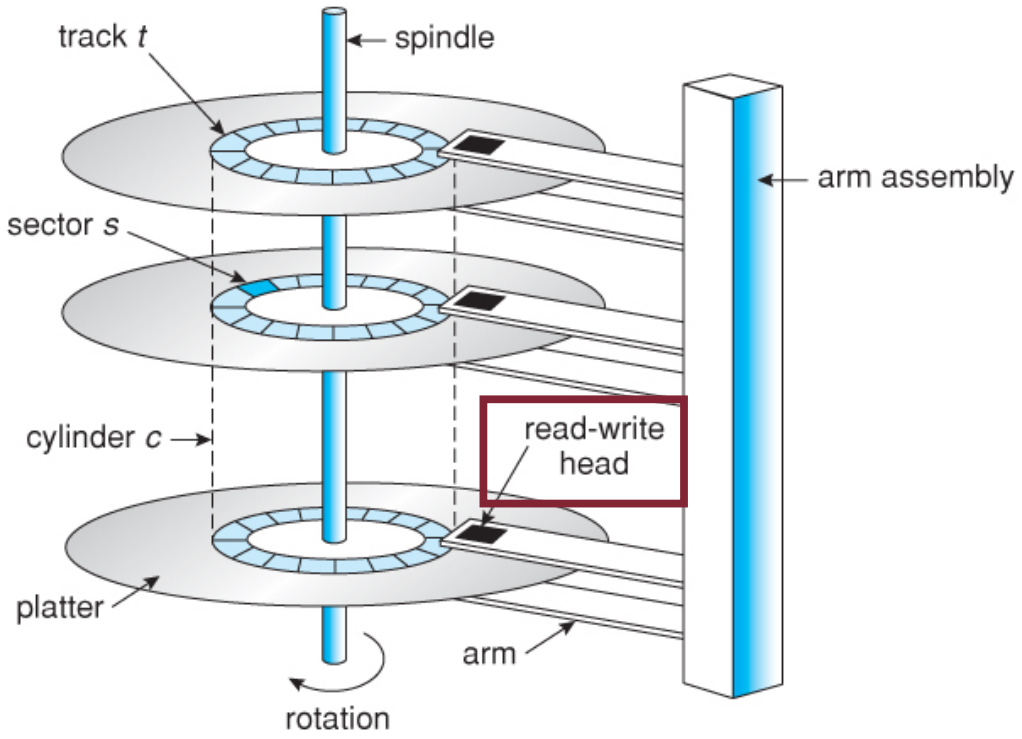
Each track is further divided into **sectors**

Each sector usually contains 512 bytes

Sectors also include a header and a trailer, and checksum information

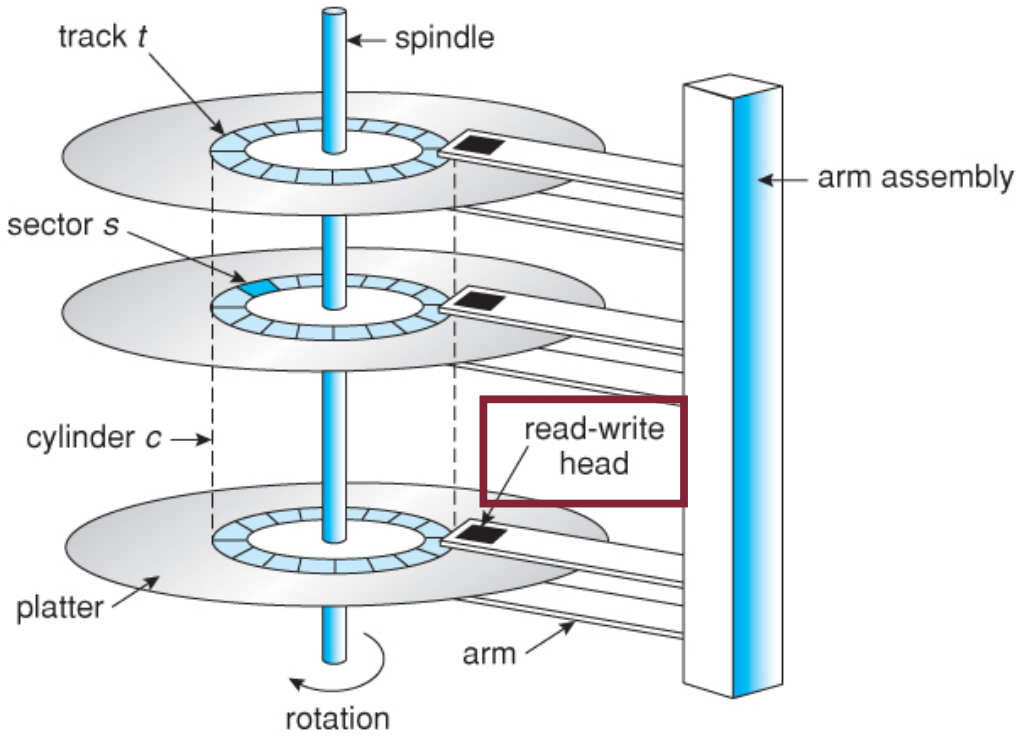
Larger sector sizes reduce the space wasted by headers and trailers, but increase internal fragmentation

Magnetic Disks: Heads



Data on hard drive is read by read-write **heads**

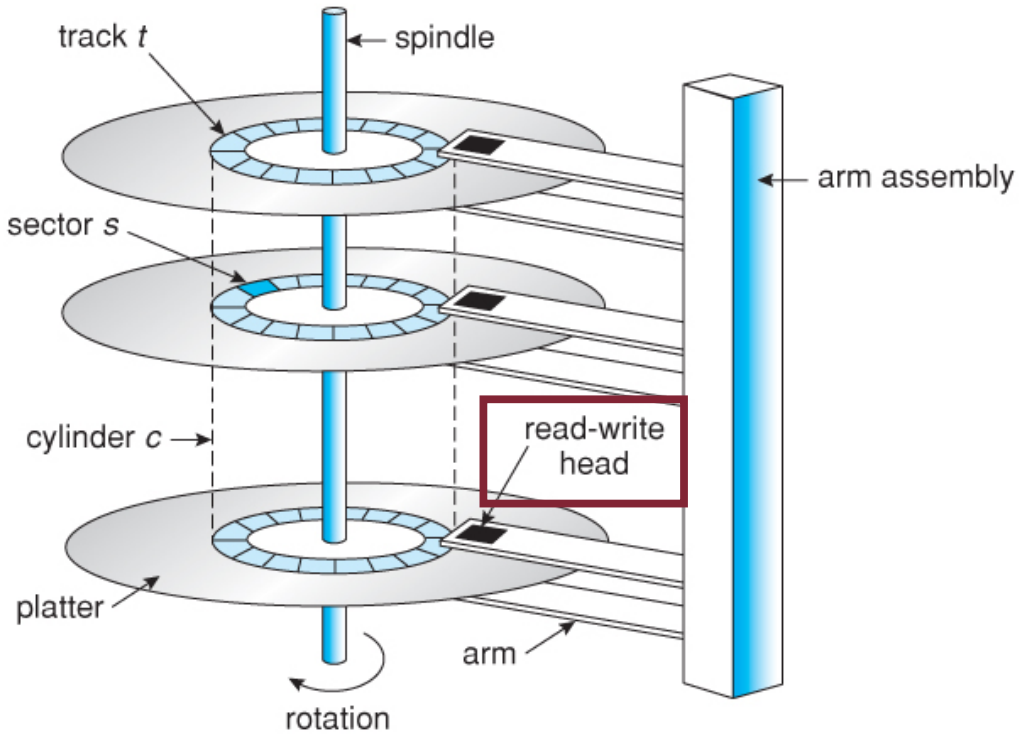
Magnetic Disks: Heads



Data on hard drive is read by read-write **heads**

Standard configuration uses one head per surface

Magnetic Disks: Heads

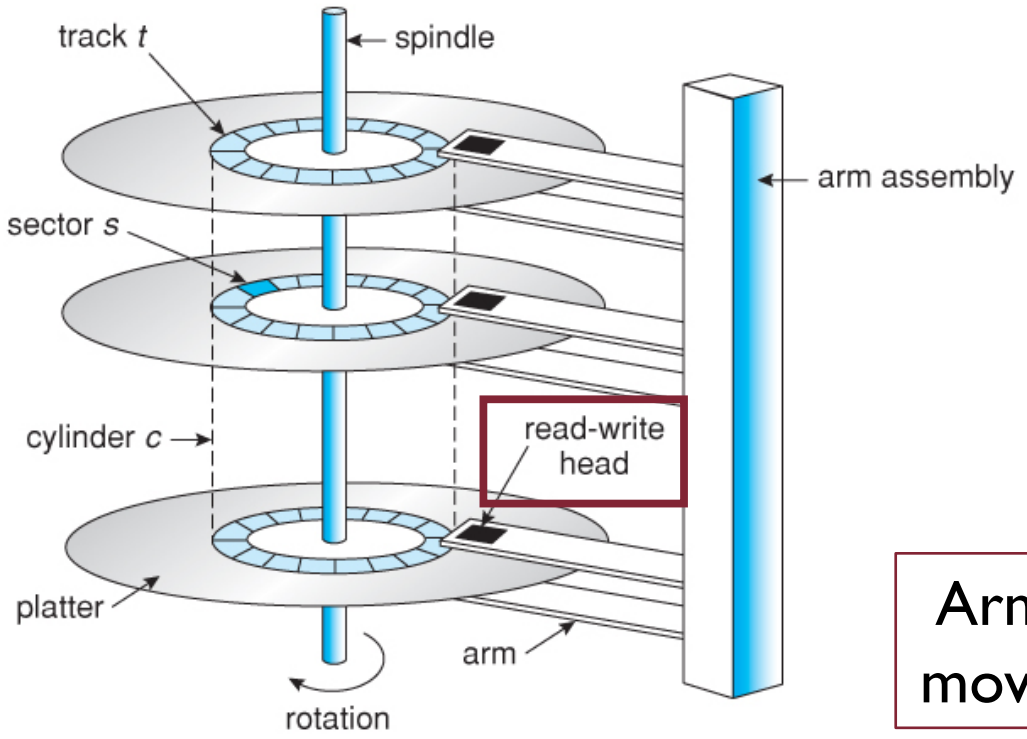


Data on hard drive is read by read-write **heads**

Standard configuration uses one head per surface

Each head is placed on a separate **arm**

Magnetic Disks: Heads



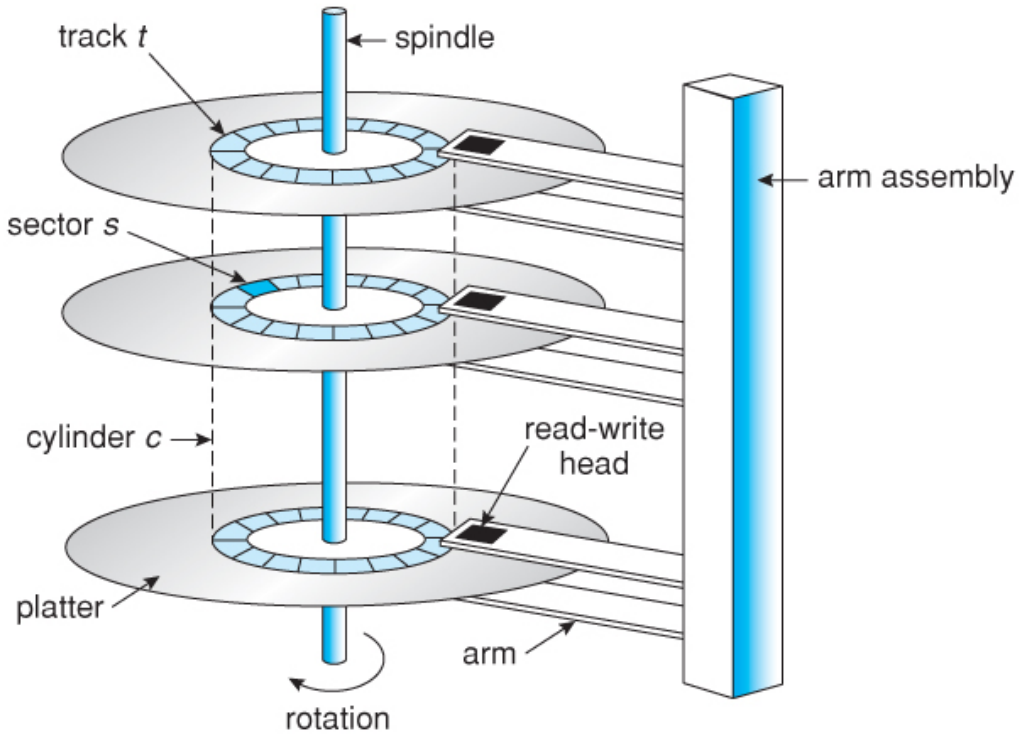
Data on hard drive is read by read-write **heads**

Standard configuration uses one head per surface

Each head is placed on a separate **arm**

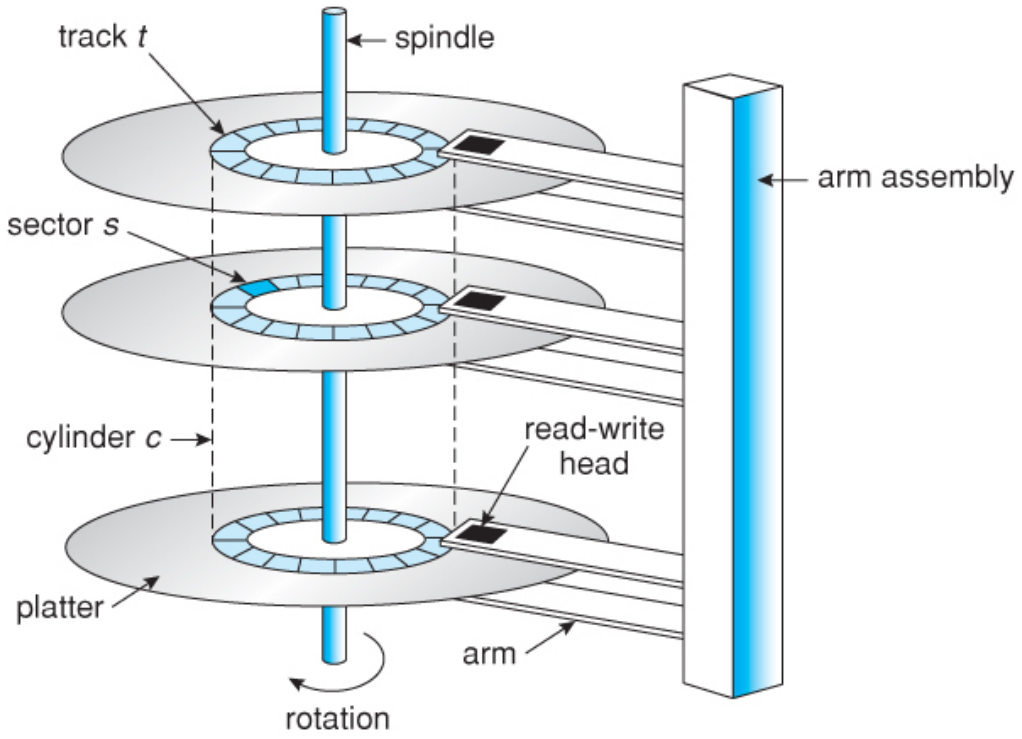
Arms are controlled by a common **arm assembly** moving simultaneously from one cylinder to another

Magnetic Disks: Storage Capacity



H = number of heads (working surfaces)

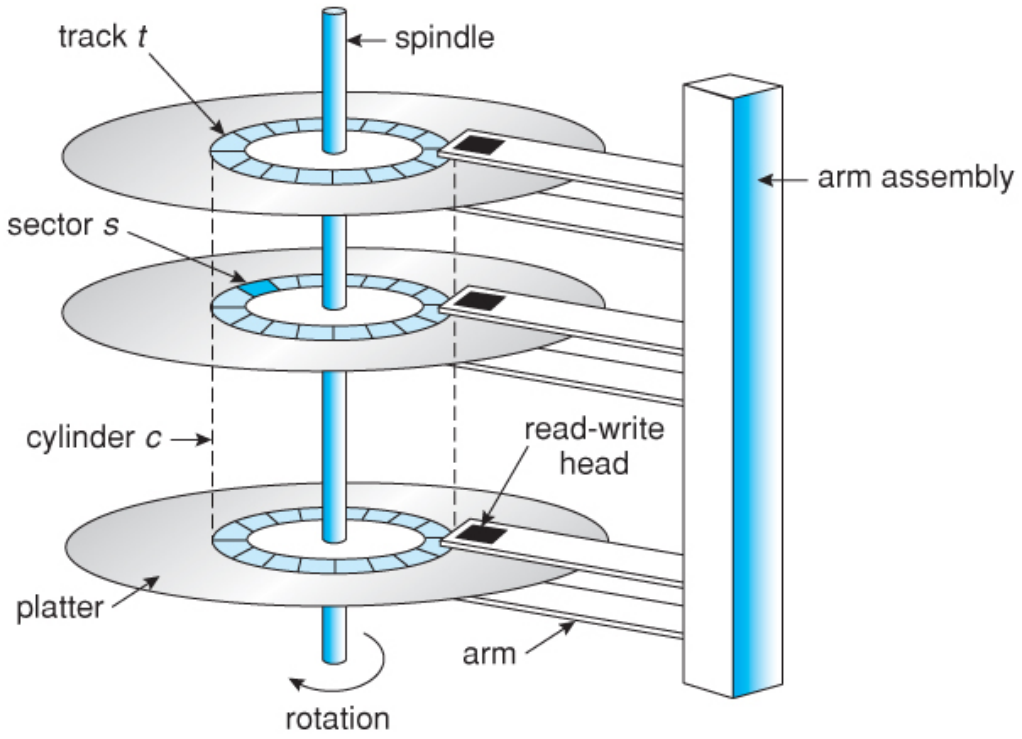
Magnetic Disks: Storage Capacity



H = number of heads (working surfaces)

T = number of tracks per surface

Magnetic Disks: Storage Capacity

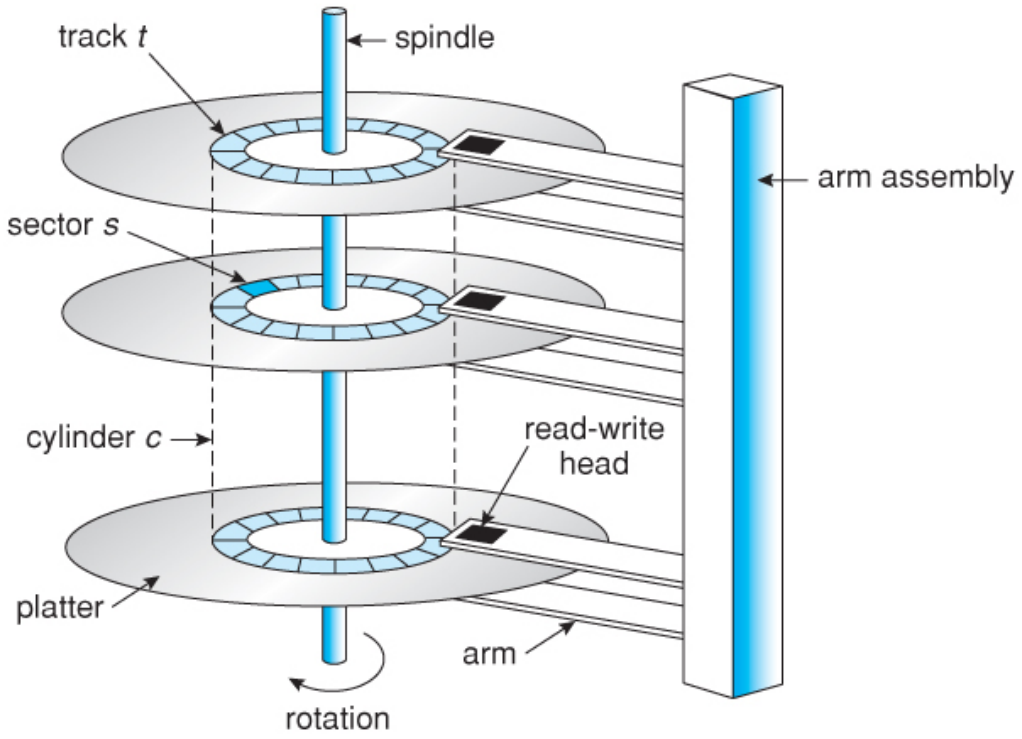


H = number of heads (working surfaces)

T = number of tracks per surface

S = number of sectors per track

Magnetic Disks: Storage Capacity



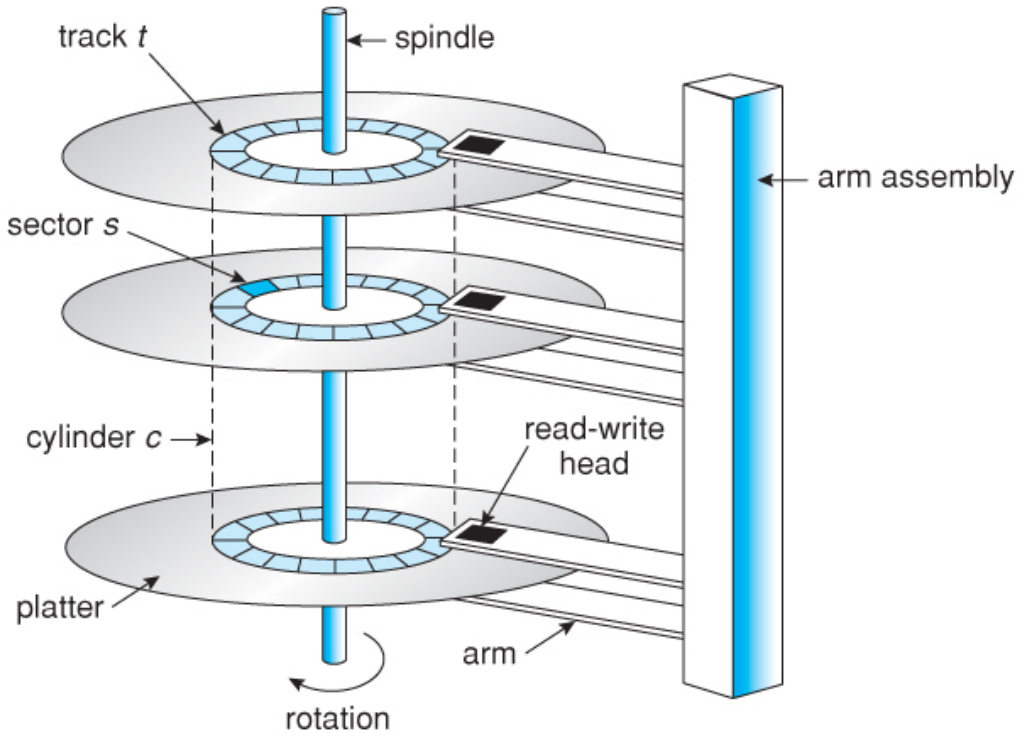
H = number of heads (working surfaces)

T = number of tracks per surface

S = number of sectors per track

B = number of bytes per sector

Magnetic Disks: Storage Capacity



H = number of heads (working surfaces)

T = number of tracks per surface

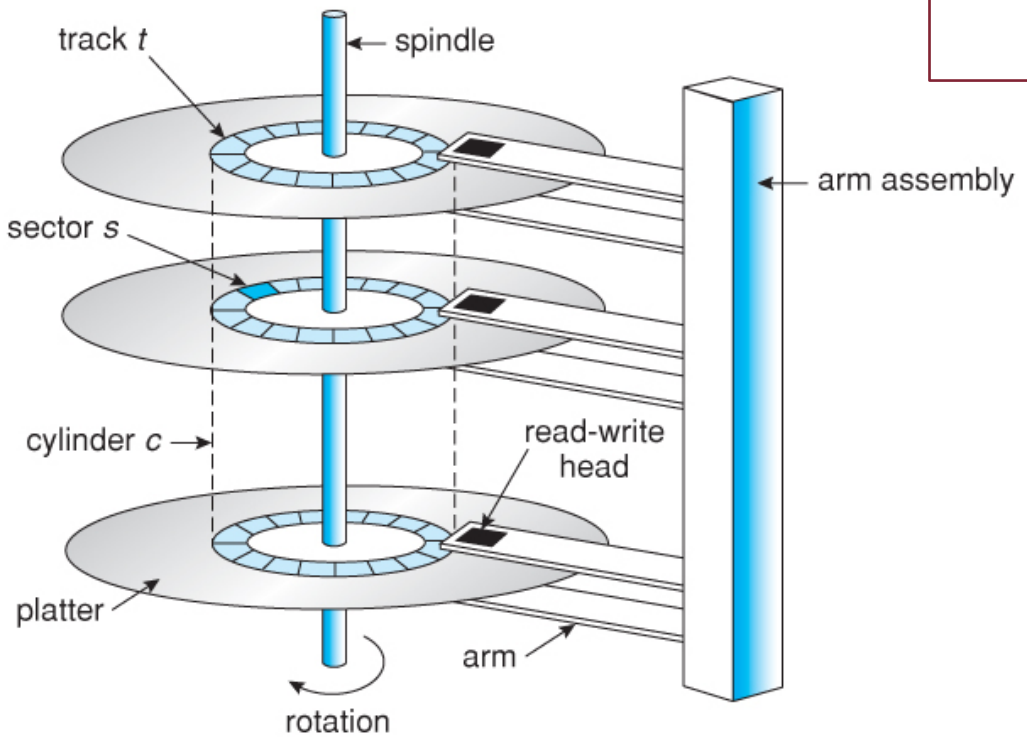
S = number of sectors per track

B = number of bytes per sector

$$C = H * T * S * B$$

OVERALL CAPACITY

Magnetic Disks: Referencing



A physical block of data is specified by the (**head**, **sector**, **cylinder**) number

Magnetic Disks: Data Transfer

- The disk rotates at high speed (e.g., **7,200 rpm** = **120 rps**)

Magnetic Disks: Data Transfer

- The disk rotates at high speed (e.g., **7,200 rpm** = **120 rps**)
- Data transfer from the disk to memory is made of **3 steps**:

Magnetic Disks: Data Transfer

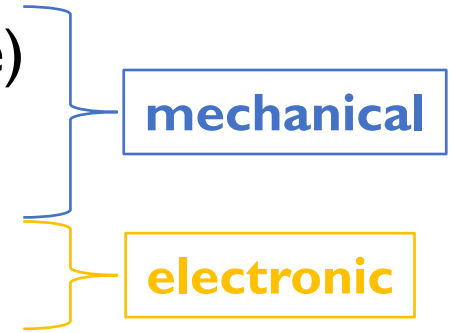
- The disk rotates at high speed (e.g., **7,200 rpm** = **120 rps**)
- Data transfer from the disk to memory is made of **3 steps**:
  - positioning time (a.k.a., seek time or random access time)
  - rotational delay
  - transfer time

Magnetic Disks: Data Transfer

- The disk rotates at high speed (e.g., **7,200 rpm** = **120 rps**)
- Data transfer from the disk to memory is made of **3 steps**:
  - positioning time (a.k.a., seek time or random access time)
  - rotational delay
  - transfer time

Magnetic Disks: Data Transfer

- The disk rotates at high speed (e.g., **7,200 rpm** = **120 rps**)
- Data transfer from the disk to memory is made of **3 steps**:
  - positioning time (a.k.a., seek time or random access time)
  - rotational delay
  - transfer time



Magnetic Disks: Positioning (Seek) Time

- The time required to move the heads to a specific track/cylinder
- Includes the time needed for the heads to settle after the move
- Depends on how fast the hardware moves the assembly arm
- Typically, the slowest step in the entire process

Magnetic Disks: Positioning (Seek) Time

- The time required to move the heads to a specific track/cylinder
- Includes the time needed for the heads to settle after the move
- Depends on how fast the hardware moves the assembly arm
- Typically, the slowest step in the entire process

Bottleneck of overall disk data transfer

Magnetic Disks: Rotational Delay

- The time required for the desired sector to rotate and come under the read-write head

Magnetic Disks: Rotational Delay

- The time required for the desired sector to rotate and come under the read-write head
- Can range from 0 up to one full revolution
  - 0 → the sector is already underneath the head
  - full revolution → the sector is the one before but in the opposite direction

Magnetic Disks: Rotational Delay

- The time required for the desired sector to rotate and come under the read-write head
- Can range from 0 up to one full revolution
  - 0 → the sector is already underneath the head
  - full revolution → the sector is the one before but in the opposite direction
- On average, 0.5 revolutions (r)
  - E.g., for a 7,200 rpm (120 rps) disk this equals to 0.5 r/120 rps **~4 msec**

Magnetic Disks: Rotational Delay

- The time required for the desired sector to rotate and come under the read-write head
- Can range from 0 up to one full revolution
  - 0 → the sector is already underneath the head
  - full revolution → the sector is the one before but in the opposite direction
- On average, 0.5 revolutions (r)
  - E.g., for a 7,200 rpm (120 rps) disk this equals to 0.5 r/120 rps **~4 msec**

The second highest bottleneck

Magnetic Disks: Transfer Time

- The time required to move data (i.e., bytes) electronically from disk to memory

Magnetic Disks: Transfer Time

- The time required to move data (i.e., bytes) electronically from disk to memory
- This is sometimes expressed as **transfer rate** (bandwidth) in bytes per second

Magnetic Disks: Transfer Time

- The time required to move data (i.e., bytes) electronically from disk to memory
- This is sometimes expressed as **transfer rate** (bandwidth) in bytes per second

Data Transfer Time = Seek Time + Rotational Delay + Transfer Time

Sometimes the term **transfer rate** is used to refer to the overall data transfer time

Magnetic Disks: Structure

- Addressed as large one-dimensional arrays of **logical blocks**

Magnetic Disks: Structure

- Addressed as large one-dimensional arrays of **logical blocks**
- Each logical block is the smalles unit of transfer (e.g., **512 bytes**)

Magnetic Disks: Structure

- Addressed as large one-dimensional arrays of **logical blocks**
- Each logical block is the smalles unit of transfer (e.g., **512 bytes**)
- The array of blocks is mapped onto disk sectors sequentially

Magnetic Disks: Structure

- Addressed as large one-dimensional arrays of **logical blocks**
- Each logical block is the smalles unit of transfer (e.g., **512 bytes**)
- The array of blocks is mapped onto disk sectors sequentially
- Sector 0 is the first sector of the first track of the outermost cylinder
  - The mapping proceeds in order through that track
  - Then through the rest of tracks in the same cylinder
  - Then through other cylinders (from the outermost to innermost)

Magnetic Disks: Risks

- Disk heads "fly" over the surface on a very thin cushion of air

Magnetic Disks: Risks

- Disk heads "fly" over the surface on a very thin cushion of air
- If they accidentally contact the disk then a **head crash** occurs

Magnetic Disks: Risks

- Disk heads "fly" over the surface on a very thin cushion of air
- If they accidentally contact the disk then a **head crash** occurs
- Head crash may permanently damage the disk or even destroy it

Magnetic Disks: Risks

- Disk heads "fly" over the surface on a very thin cushion of air
- If they accidentally contact the disk then a **head crash** occurs
- Head crash may permanently damage the disk or even destroy it
- To avoid such a risk, disk heads are "parked" when the computer is turned off

Magnetic Disks: Risks

- Disk heads "fly" over the surface on a very thin cushion of air
- If they accidentally contact the disk then a **head crash** occurs
- Head crash may permanently damage the disk or even destroy it
- To avoid such a risk, disk heads are "parked" when the computer is turned off
- Parking heads means to move them off the disk or to an area where no data is stored

Magnetic Disks: Interfaces

- Hard drives may be removable as floppy disks, and some are even hot-swappable
  - they can be removed while the computer is running
- Disk drives are connected to the computer via the I/O bus
- Some of the common interface formats include:
  - Enhanced Integrated Drive Electronics (EIDE);
  - Advanced Technology Attachment (ATA) and Serial ATA (SATA);
  - Universal Serial Bus (USB);
  - Fiber Channel (FC);
  - Small Computer Systems Interface (SCSI)

Magnetic Disks: Controllers

- The **host controller** is at the computer end of the I/O bus

Magnetic Disks: Controllers

- The **host controller** is at the computer end of the I/O bus
- The **disk controller** is built into the disk itself

Magnetic Disks: Controllers

- The **host controller** is at the computer end of the I/O bus
- The **disk controller** is built into the disk itself
- The CPU issues commands to the host controller (typically via memory-mapped I/O ports)

Magnetic Disks: Controllers

- The **host controller** is at the computer end of the I/O bus
- The **disk controller** is built into the disk itself
- The CPU issues commands to the host controller (typically via memory-mapped I/O ports)
- Data is transferred between the magnetic surface and onboard **cache** by the disk controller

Magnetic Disks: Controllers

- The **host controller** is at the computer end of the I/O bus
- The **disk controller** is built into the disk itself
- The CPU issues commands to the host controller (typically via memory-mapped I/O ports)
- Data is transferred between the magnetic surface and onboard **cache** by the disk controller
- Finally, data is transferred from that cache to the host controller and the motherboard memory at electronic speeds

Minimize Data Transfer Time

- Mechanical components of magnetic disks cause bottleneck
  - Seek Time
  - Rotational Delay

Minimize Data Transfer Time

- Mechanical components of magnetic disks cause bottleneck
  - Seek Time
  - Rotational Delay
- To minimize data transfer time from disk we need to minimize those

Minimize Data Transfer Time

- Mechanical components of magnetic disks cause bottleneck
  - Seek Time
  - Rotational Delay
- To minimize data transfer time from disk we need to minimize those
- How?
  - Smaller disks → lower seek time, since arms have to travel smaller distance
  - Fast-spinning disks → lower rotational delay

Minimize Data Transfer Time

- Mechanical components of magnetic disks cause bottleneck
  - Seek Time
  - Rotational Delay
- To minimize data transfer time from disk we need to minimize those
- How?
  - Smaller disks → lower seek time, since arms have to travel smaller distance
  - Fast-spinning disks → lower rotational delay

Hardware Optimization

Minimize Data Transfer Time

- How can the OS help minimize data transfer time?
- Schedule disk operations so as to minimize head movement
- Lay out data on disk so that related data are located on close tracks
- Place commonly-used data on a specific portion of the disk
- Pick carefully the block size contained on each sector:
  - Too small → more seeks are needed to transfer the same amount of data
  - Too large → more internal fragmentation and space wasted

Disk (Head) Scheduling

Setting

The OS is getting constantly read/write disk requests from a bunch of processes, each one having its set of open files

Disk (Head) Scheduling

Setting

The OS is getting constantly read/write disk requests from a bunch of processes, each one having its set of open files

Idea

Permute the order of disk requests from the original order of arrival, so as to reduce the length and number of disk seeks

Disk (Head) Scheduling

Setting

The OS is getting constantly read/write disk requests from a bunch of processes, each one having its set of open files

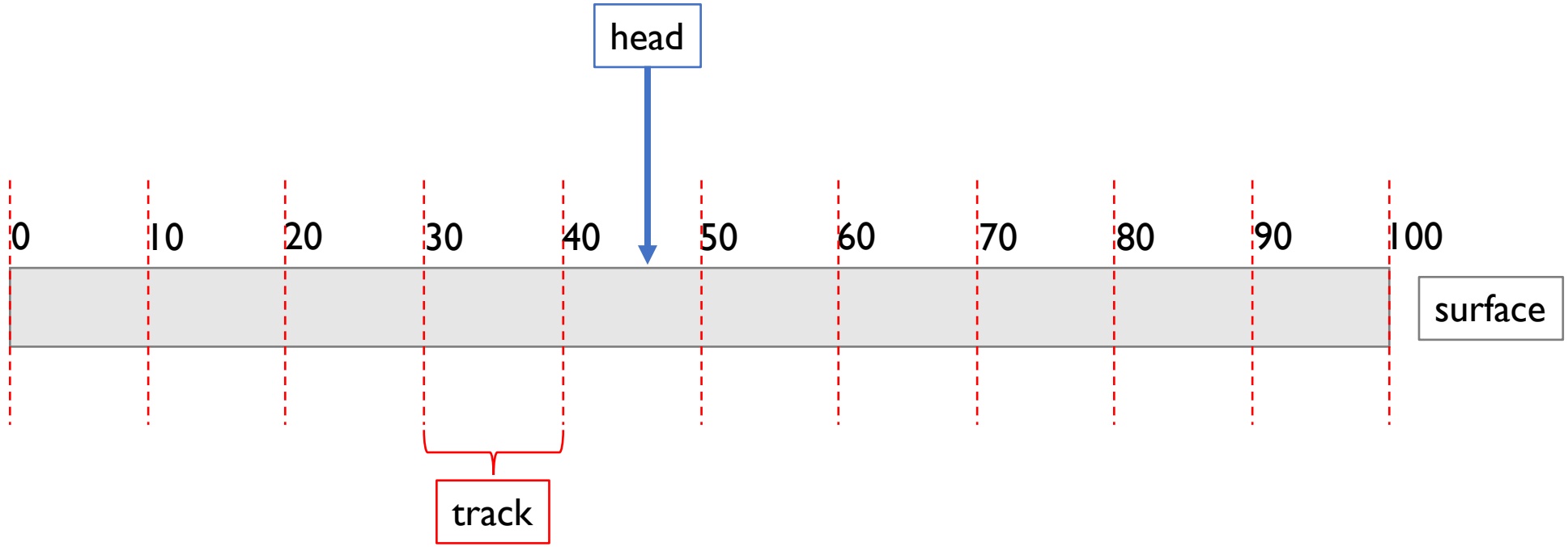
Idea

Permute the order of disk requests from the original order of arrival, so as to reduce the length and number of disk seeks

We only try minimizing the seek time not the rotational delay

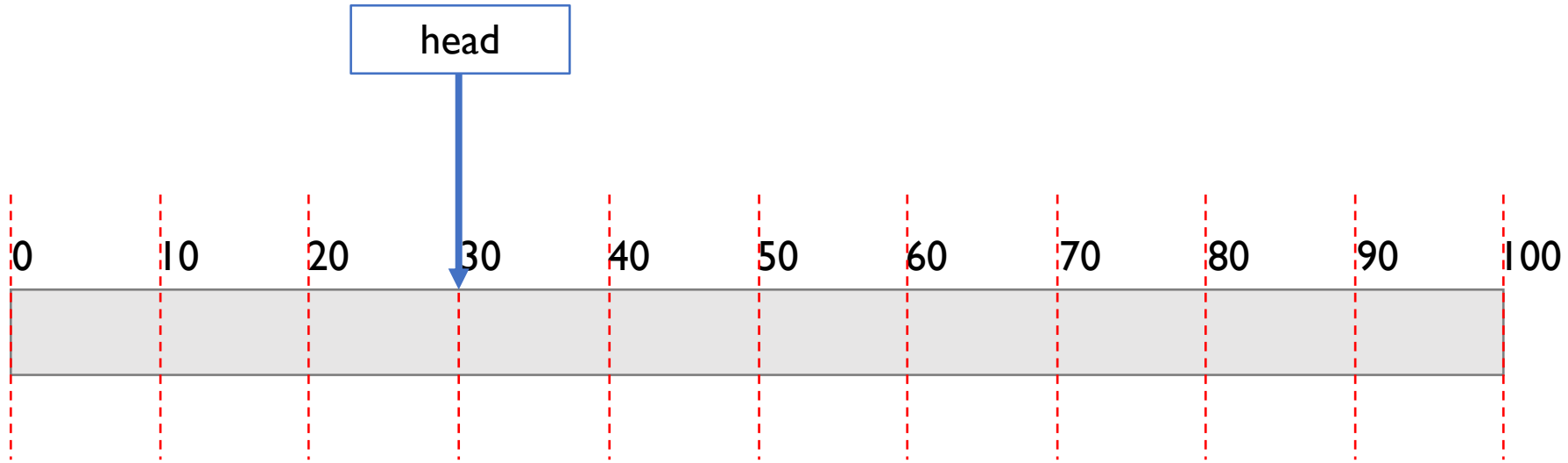
We consider track/cylinder independently of the sector

Disk Scheduling: Model



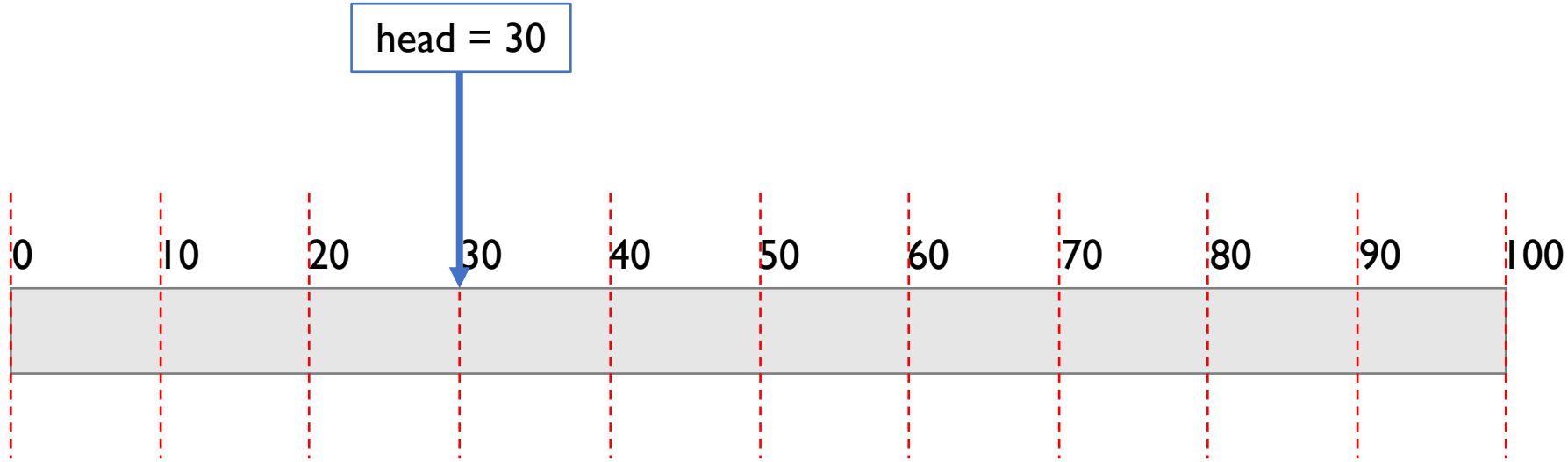
Disk Scheduling: First Come First Served (FCFS)

Serve the requests in the exact same order they arrive



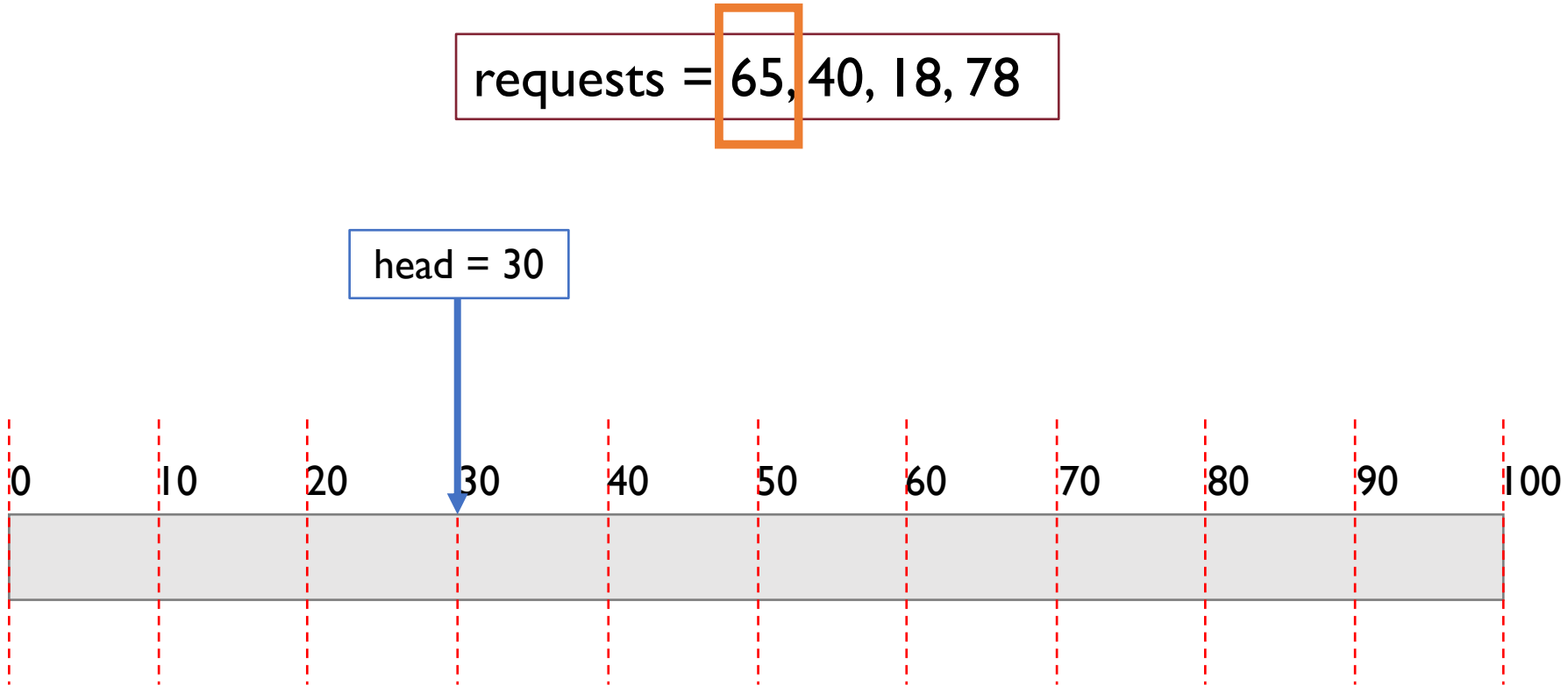
Disk Scheduling: First Come First Served (FCFS)

requests = 65, 40, 18, 78



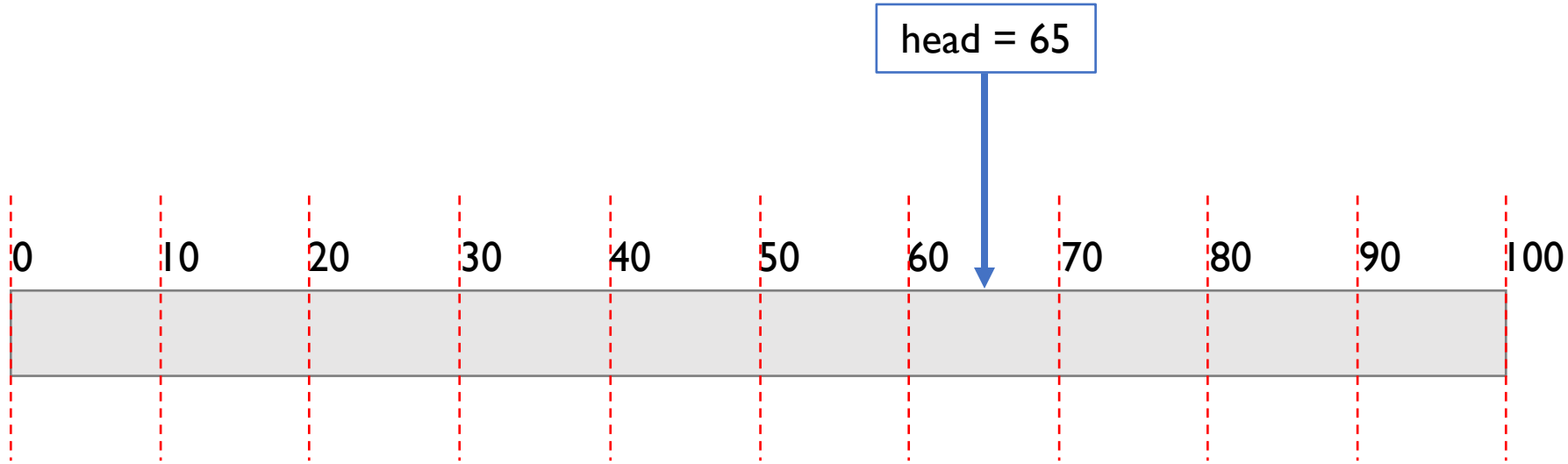
Distance travelled = 0

Disk Scheduling: First Come First Served (FCFS)



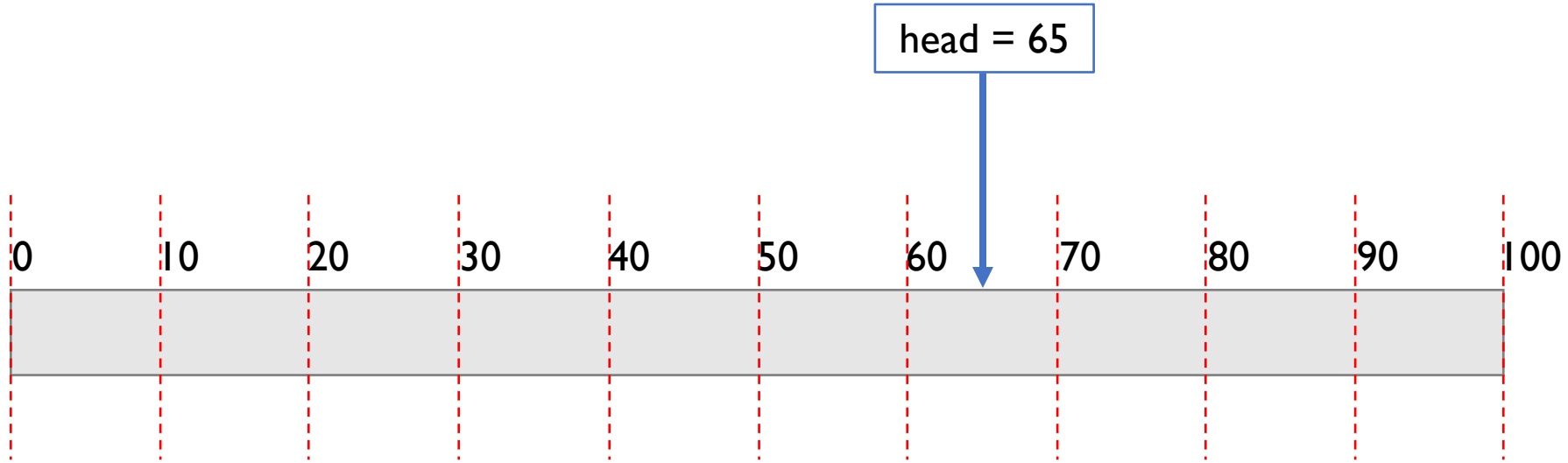
Disk Scheduling: First Come First Served (FCFS)

requests = 65, 40, 18, 78



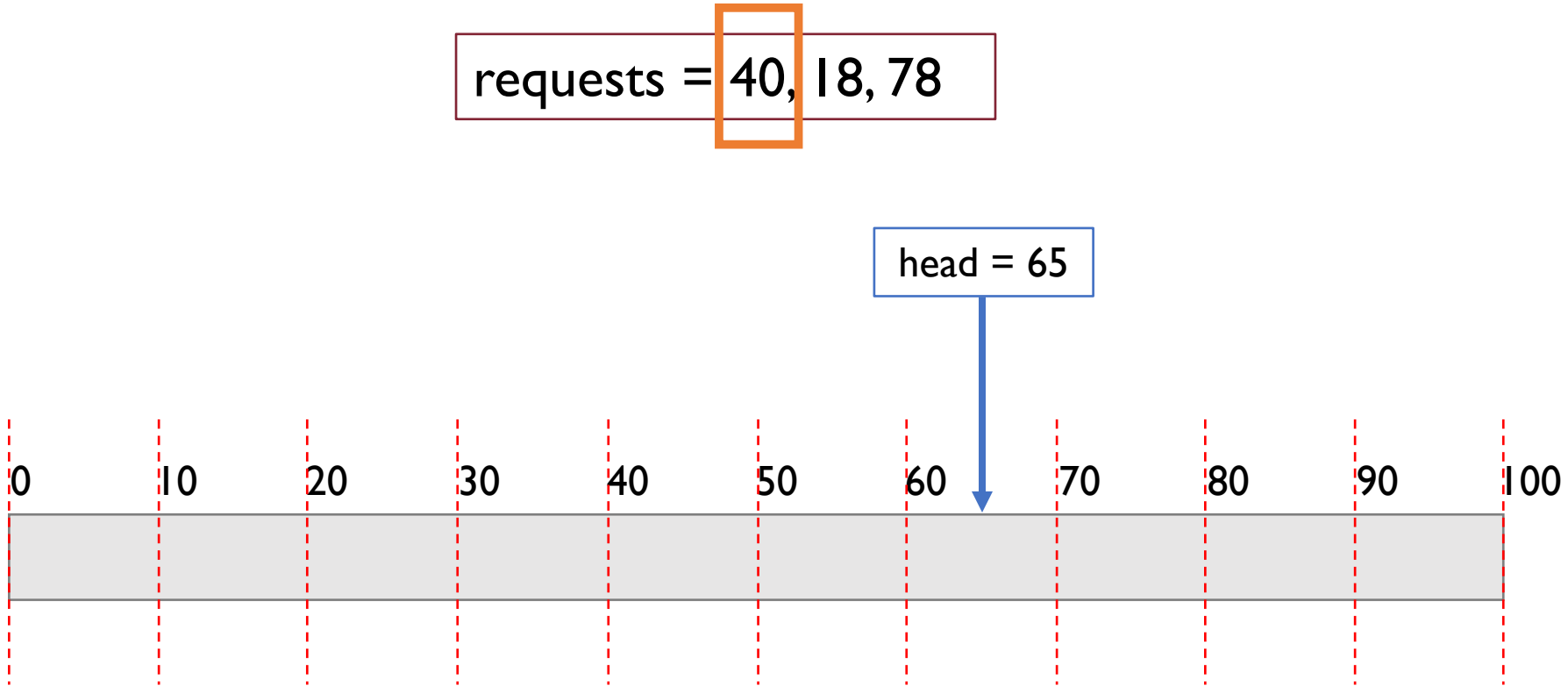
Disk Scheduling: First Come First Served (FCFS)

requests = 65, 40, 18, 78



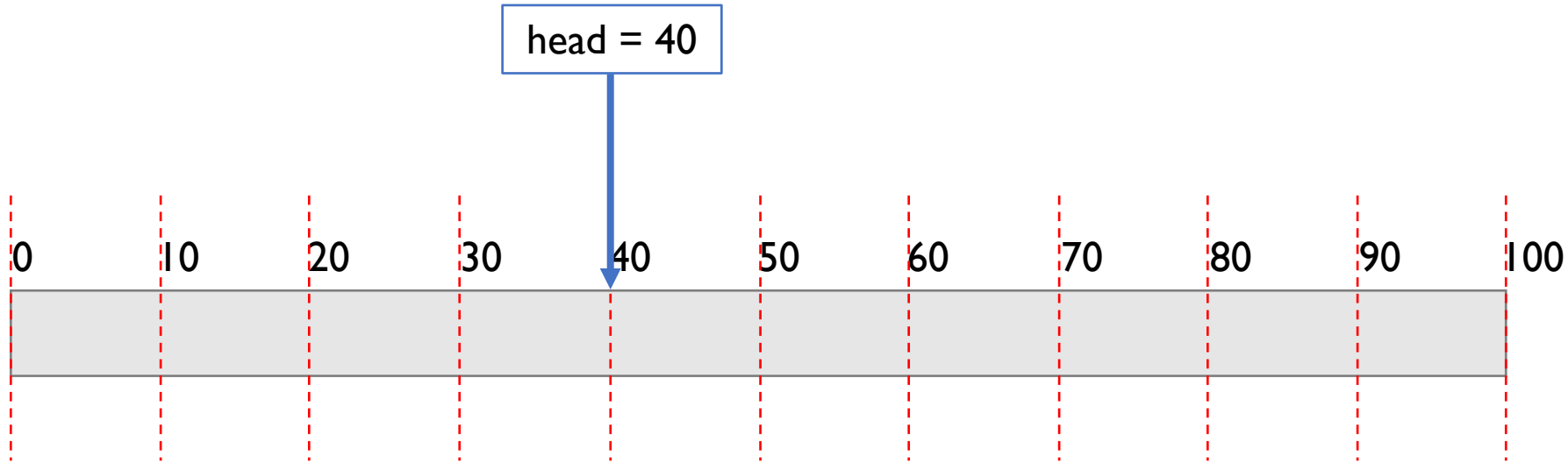
Distance travelled = 0 + |65 – 30| = 35

Disk Scheduling: First Come First Served (FCFS)



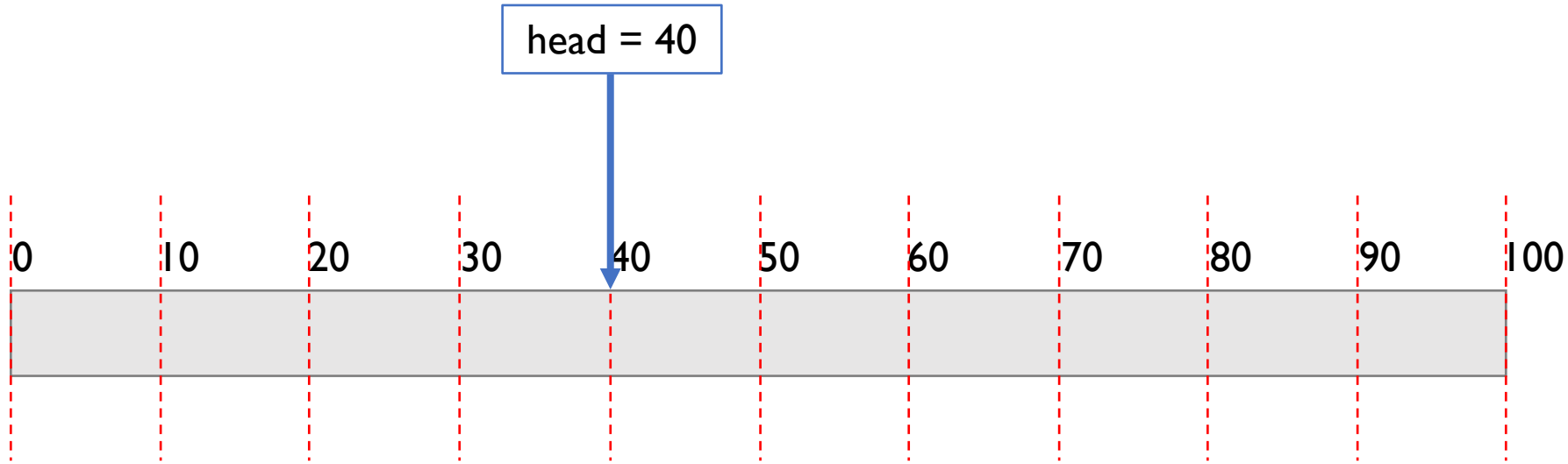
Disk Scheduling: First Come First Served (FCFS)

requests = 40, 18, 78



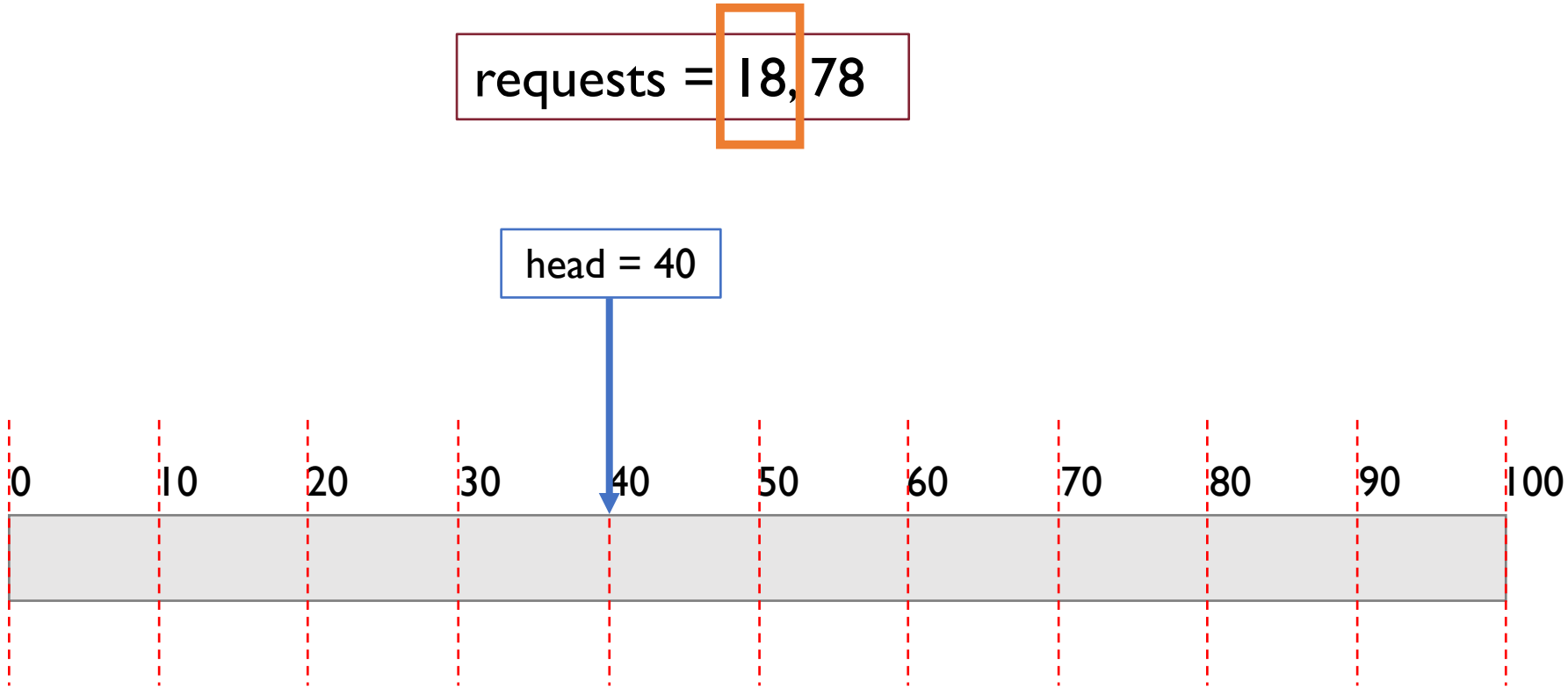
Disk Scheduling: First Come First Served (FCFS)

requests = 40, 18, 78



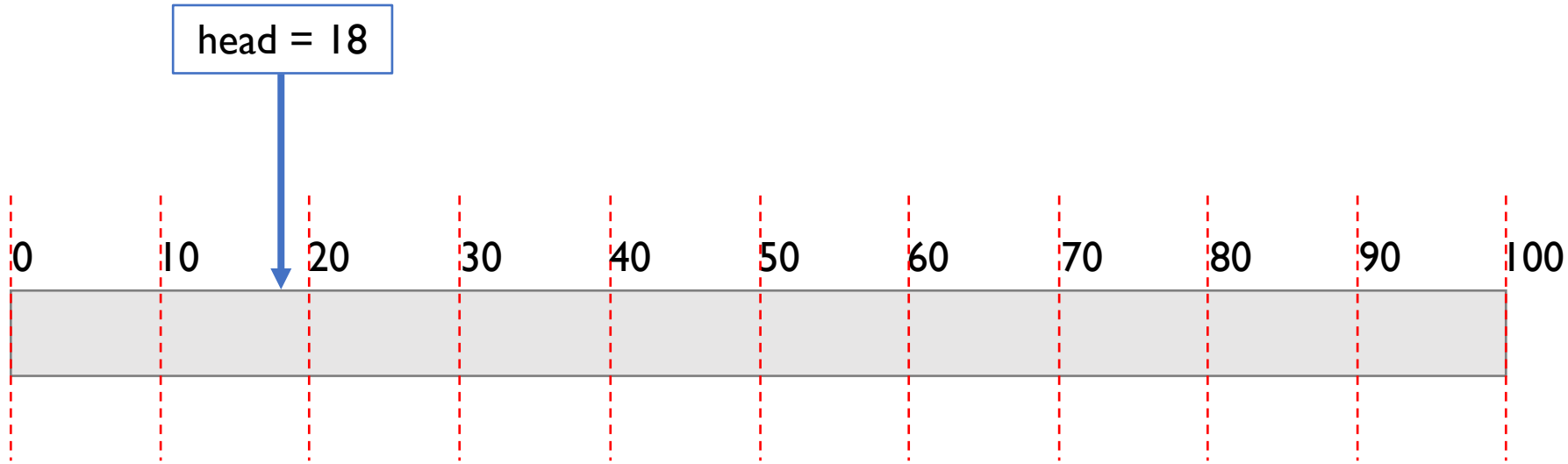
Distance travelled = 35 + |40 – 65| = 60

Disk Scheduling: First Come First Served (FCFS)



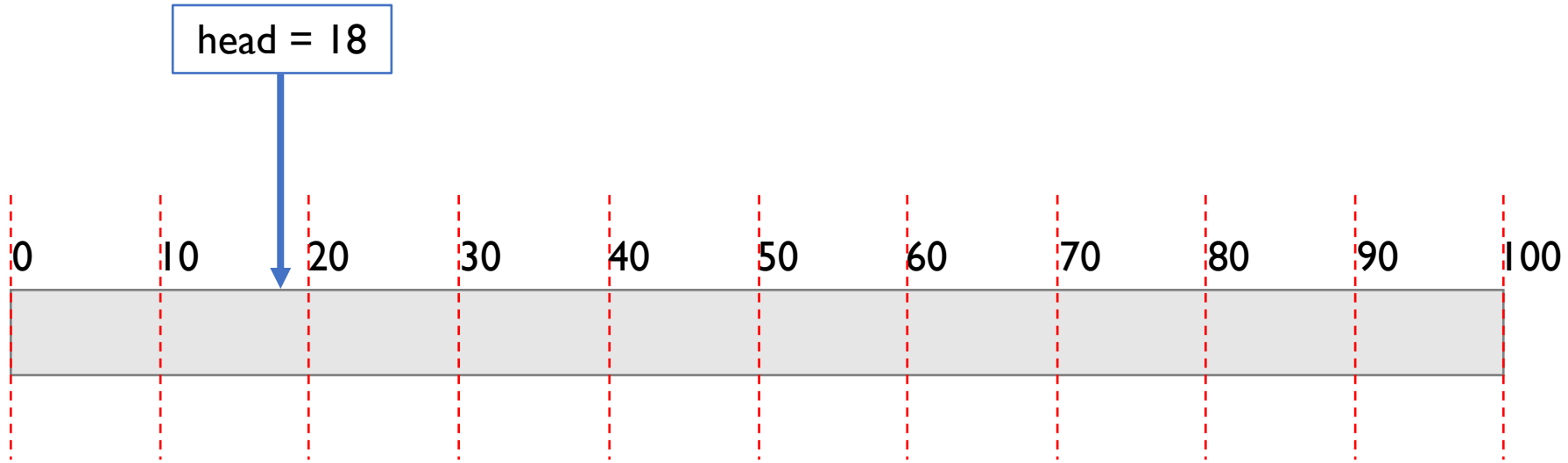
Disk Scheduling: First Come First Served (FCFS)

requests = 18, 78



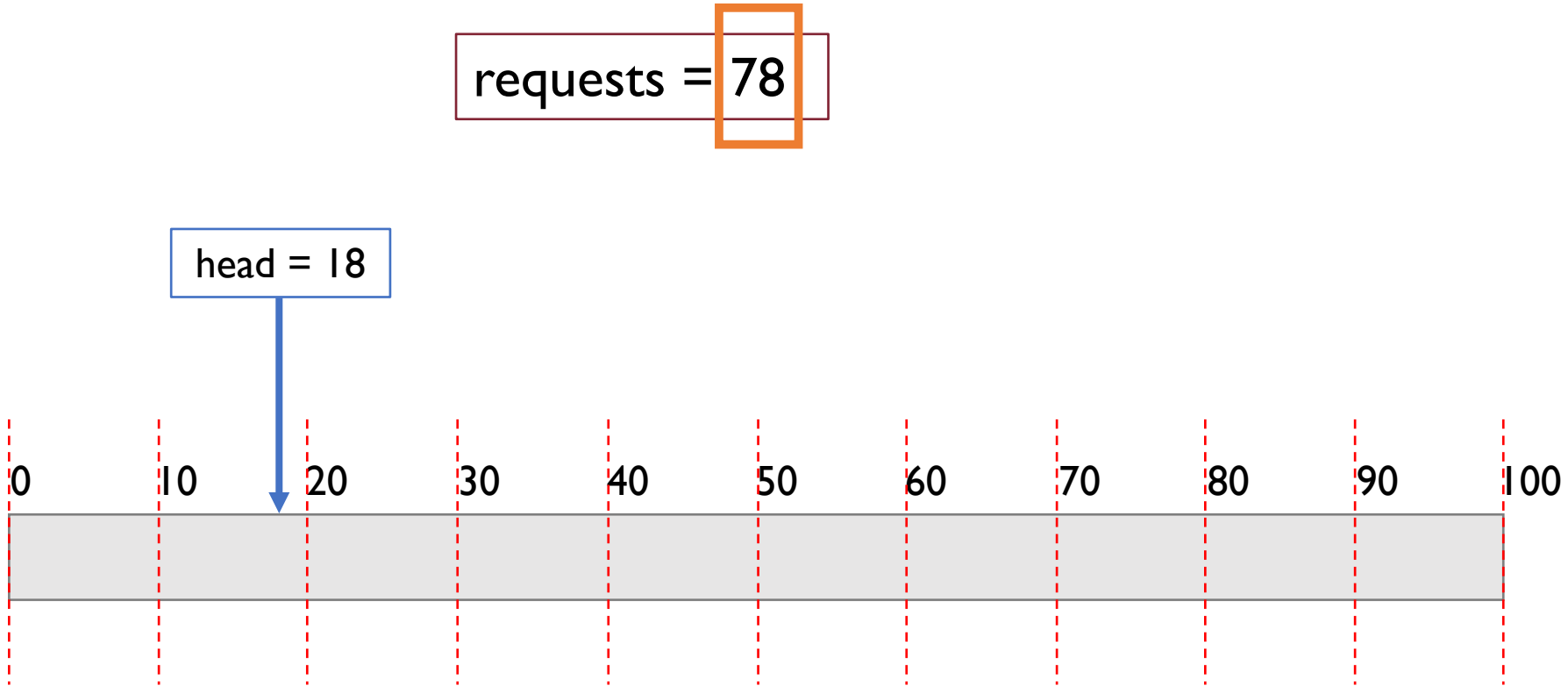
Disk Scheduling: First Come First Served (FCFS)

requests = 18, 78

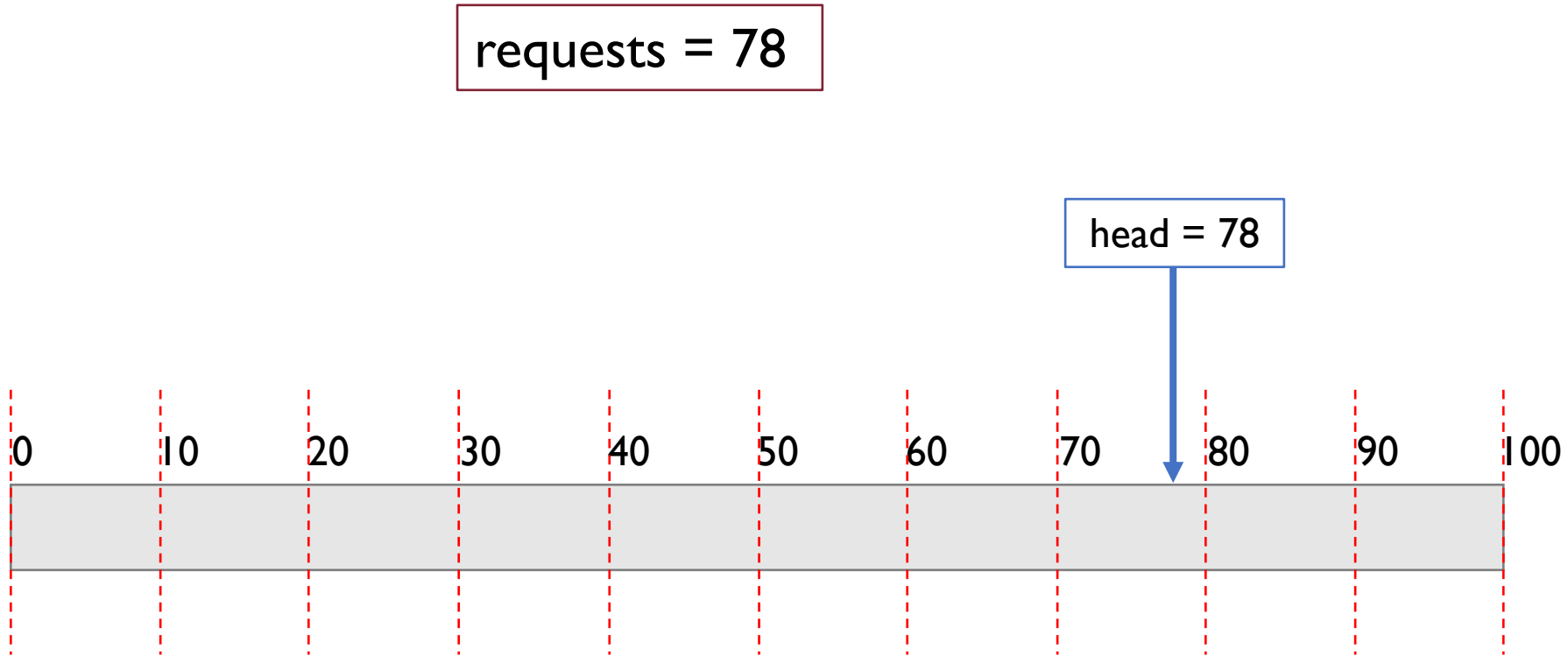


Distance travelled = 60 + |18 – 40| = 82

Disk Scheduling: First Come First Served (FCFS)

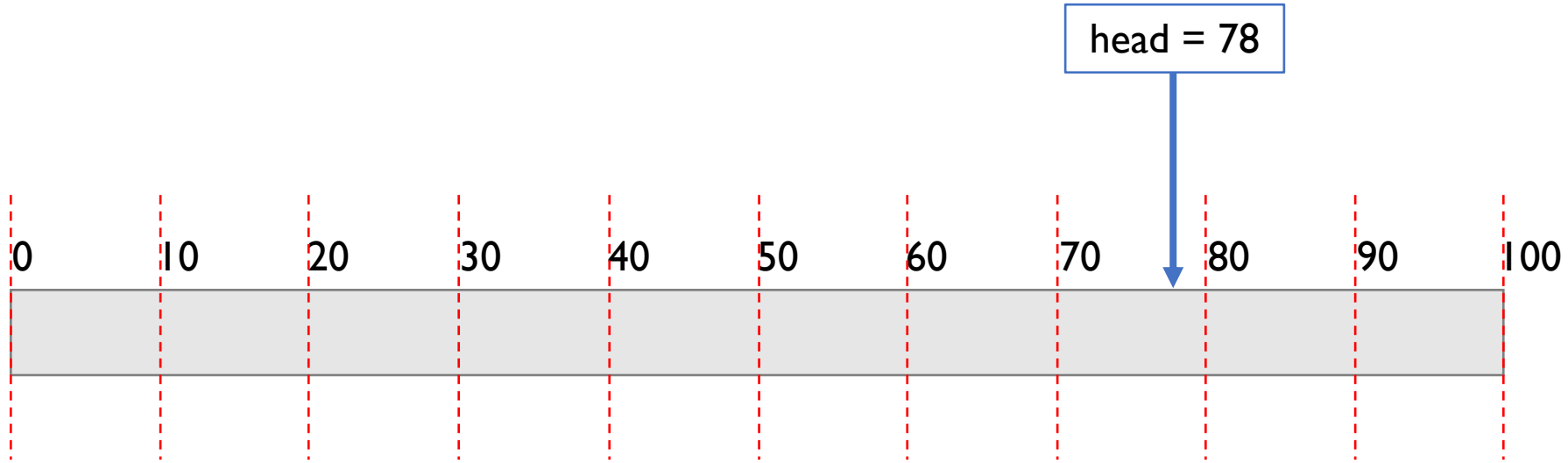


Disk Scheduling: First Come First Served (FCFS)



Disk Scheduling: First Come First Served (FCFS)

requests = 78



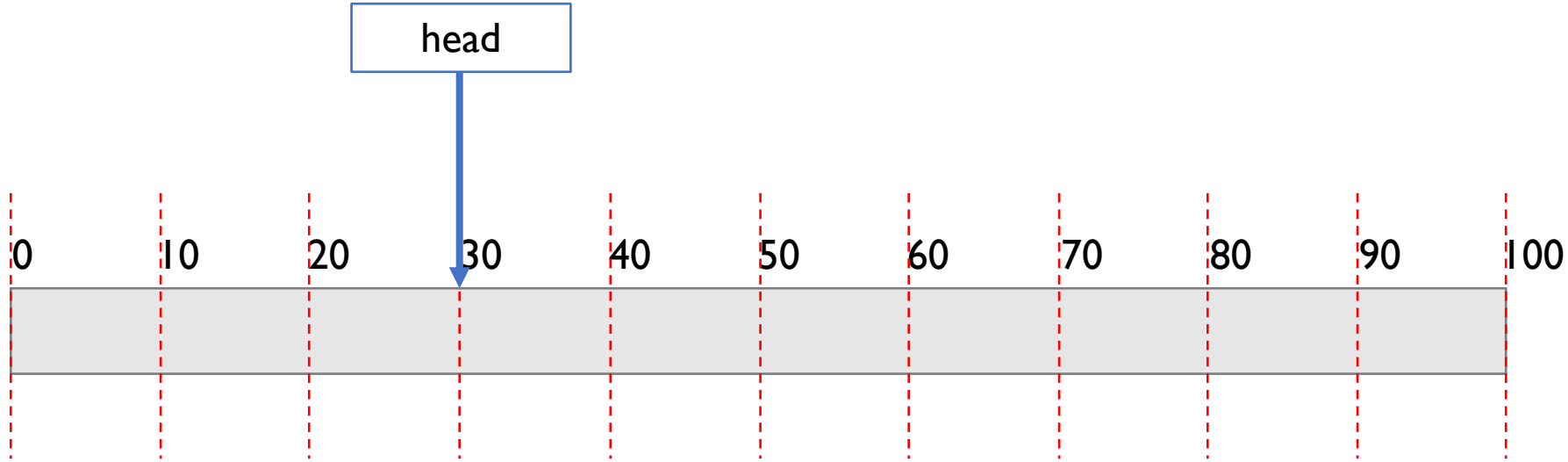
Distance travelled = 82 + |78 – 18| = **142**

FCFS: Considerations

- Easy to implement and fair
- Works quite well when systems are underloaded
- Performance may quickly deteriorate as requests increase
- Used also by SSD drives because accesses there do not require any mechanical movement
  - It's like random access to main memory

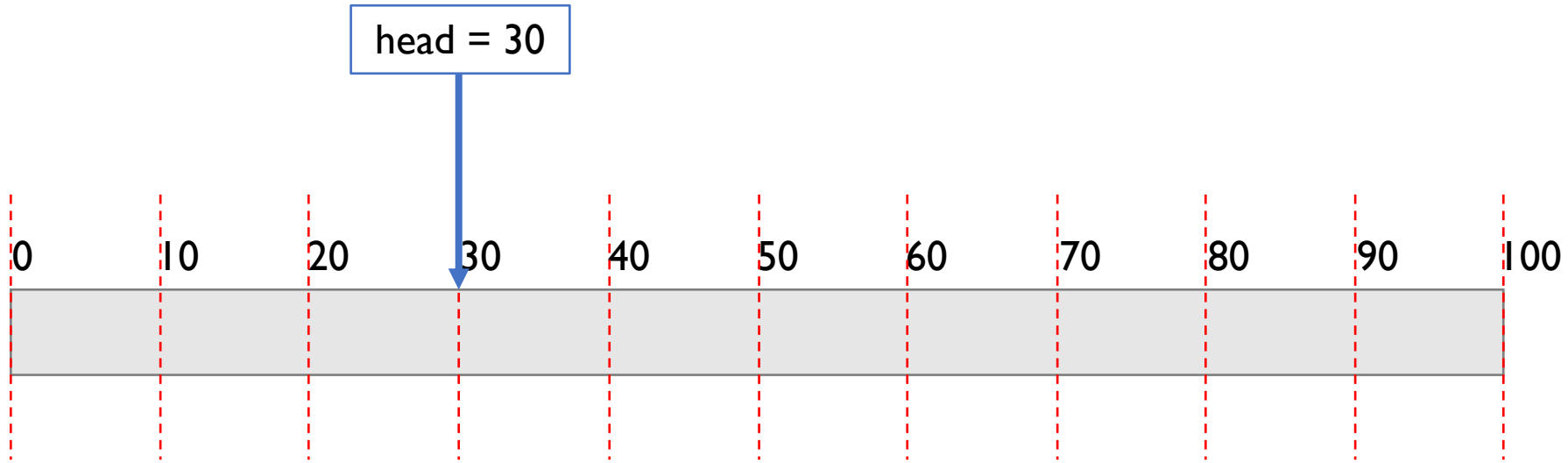
Disk Scheduling: Shortest Seek Time First (SSTF)

Greedily select the next closest request



Disk Scheduling: Shortest Seek Time First (SSTF)

requests = 65, 40, 18, 78

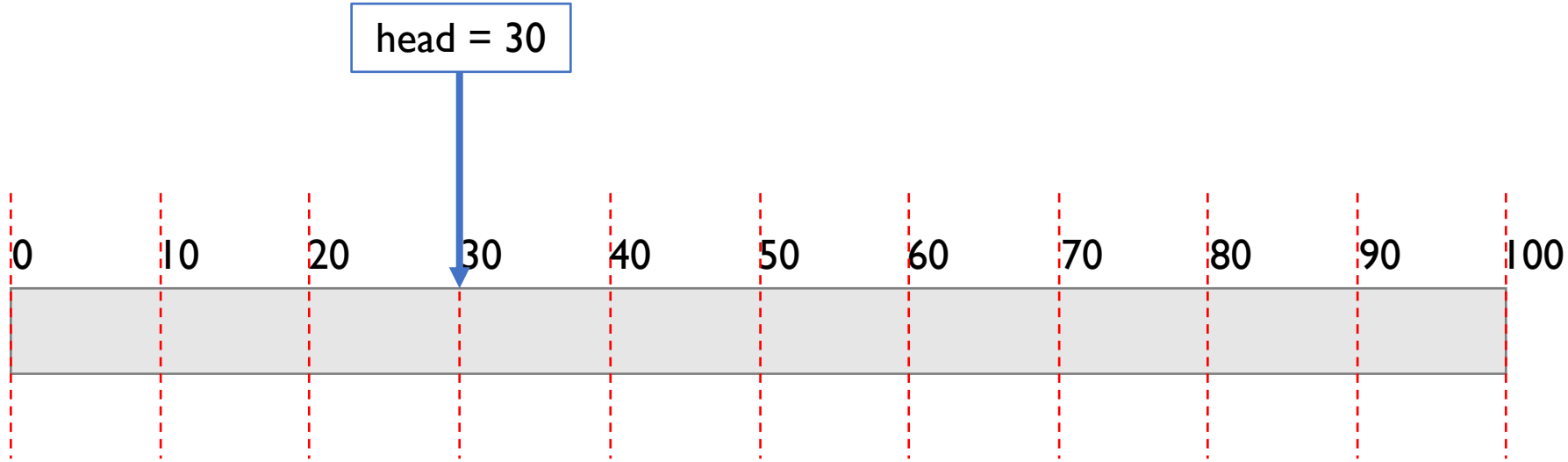


Distance travelled = 0

Disk Scheduling: Shortest Seek Time First (SSTF)

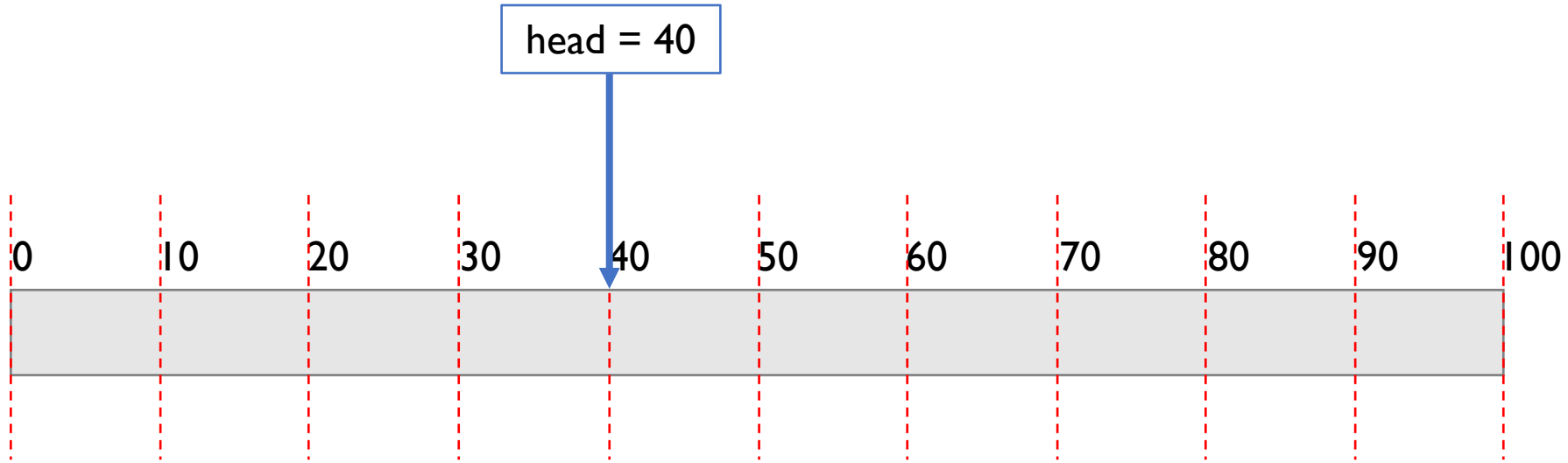
requests = 65, 40, 18, 78

Closest request = 40



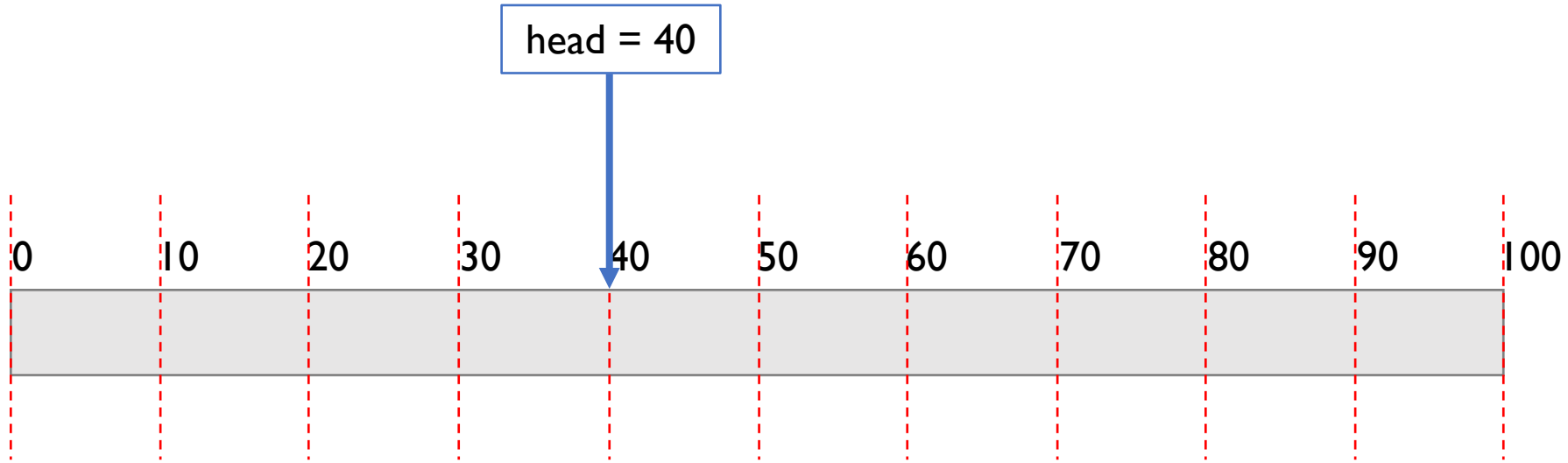
Disk Scheduling: Shortest Seek Time First (SSTF)

requests = 65, 40, 18, 78



Disk Scheduling: Shortest Seek Time First (SSTF)

requests = 65, 40, 18, 78

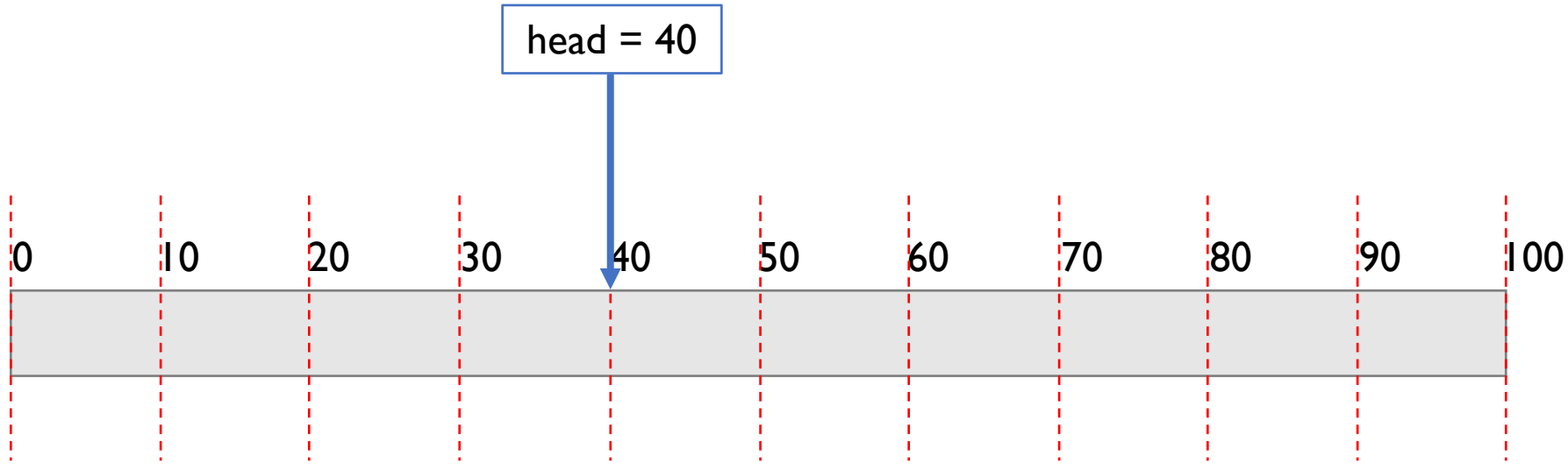


Distance travelled = 0 + |40 – 30| = 10

Disk Scheduling: Shortest Seek Time First (SSTF)

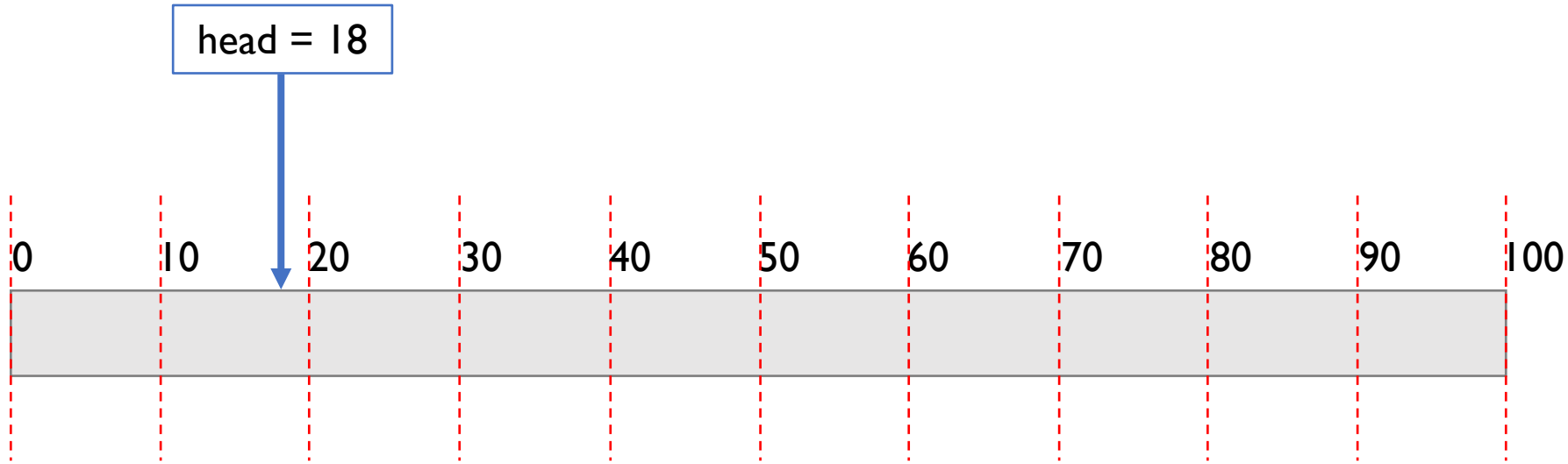
requests = 65, 18, 78

Closest request = 18



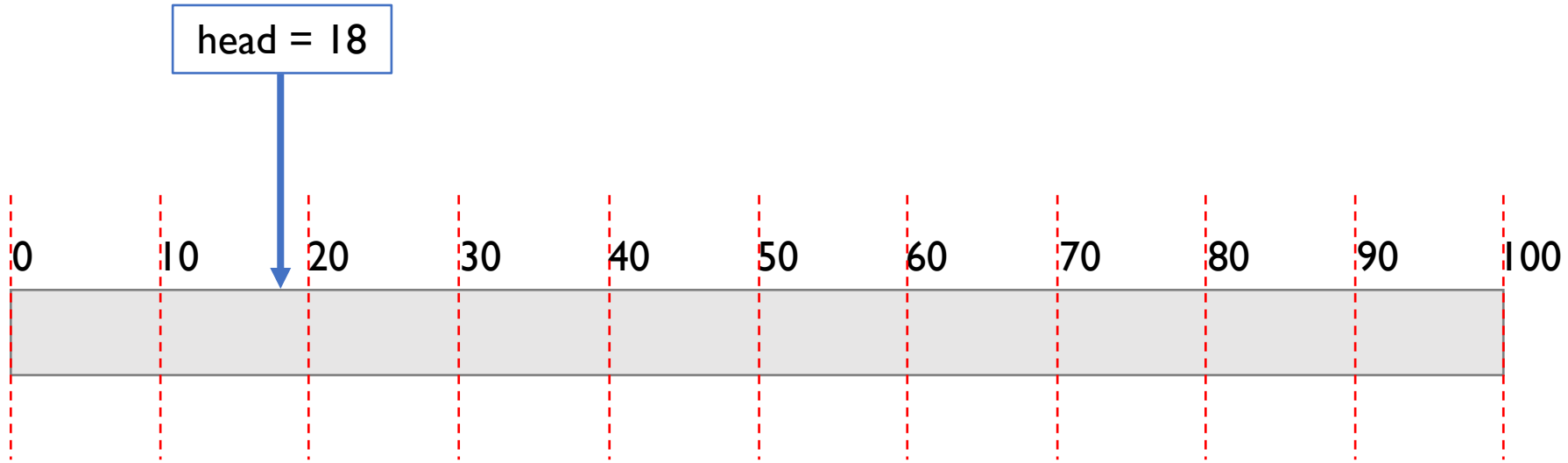
Disk Scheduling: Shortest Seek Time First (SSTF)

requests = 65, 18, 78



Disk Scheduling: Shortest Seek Time First (SSTF)

requests = 65, 18, 78

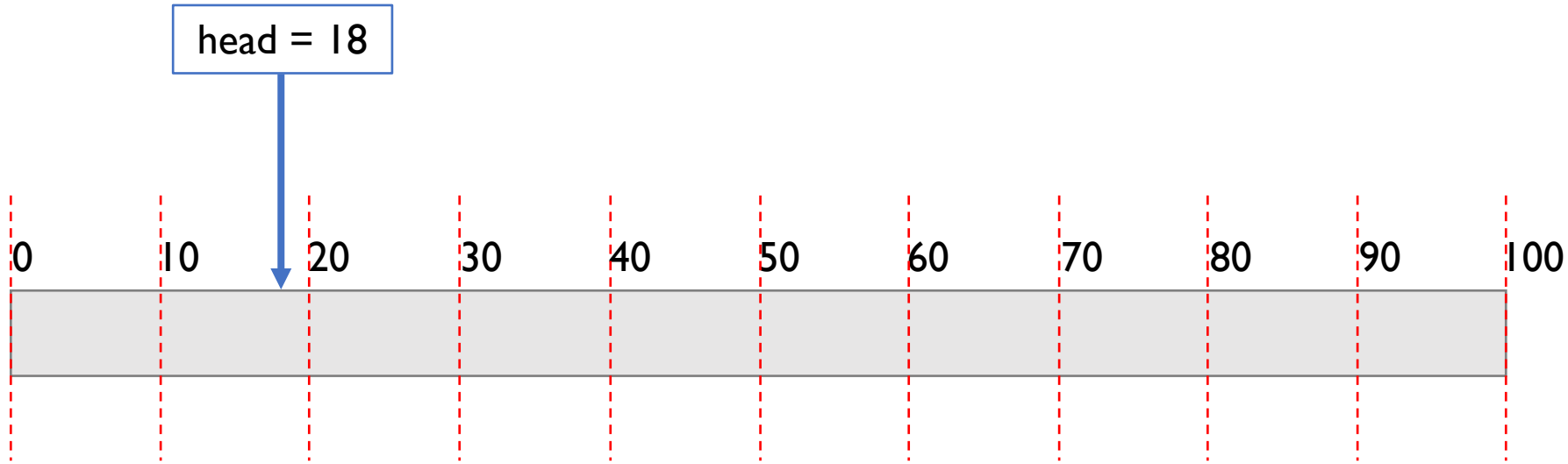


Distance travelled = 10 + |18 – 40| = 32

Disk Scheduling: Shortest Seek Time First (SSTF)

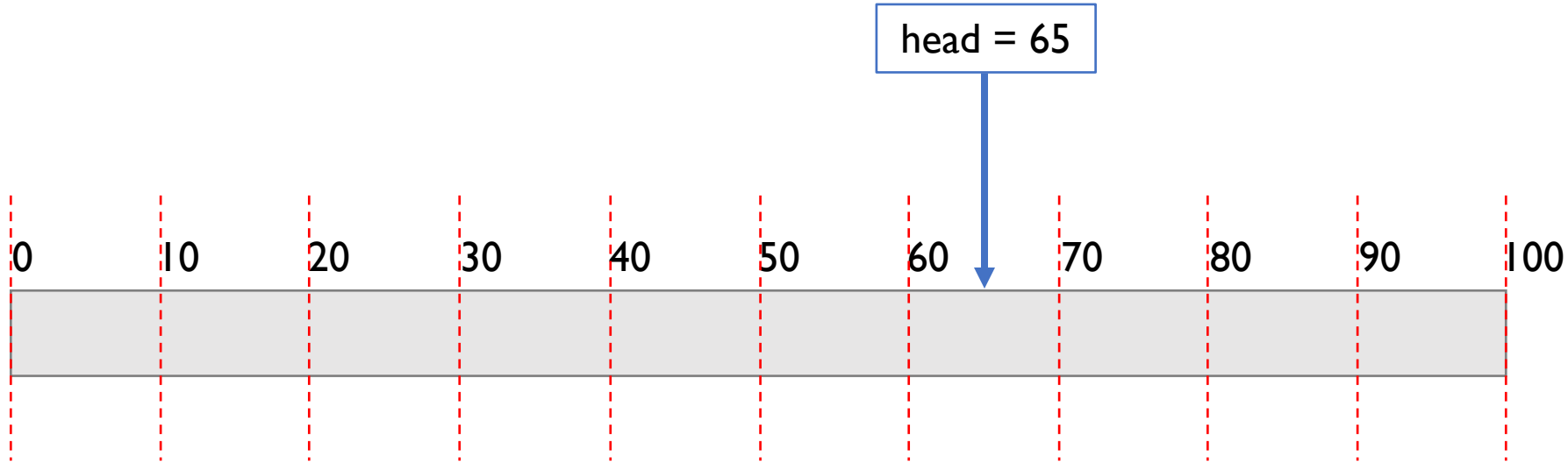
requests = 65, 78

Closest request = 65



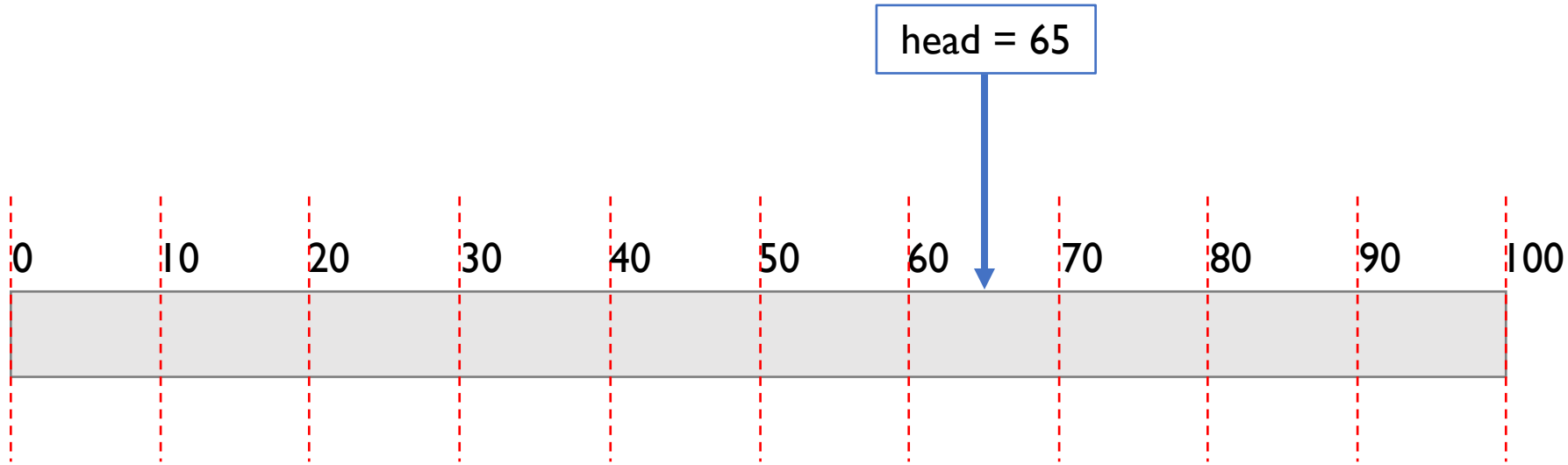
Disk Scheduling: Shortest Seek Time First (SSTF)

requests = 65, 78



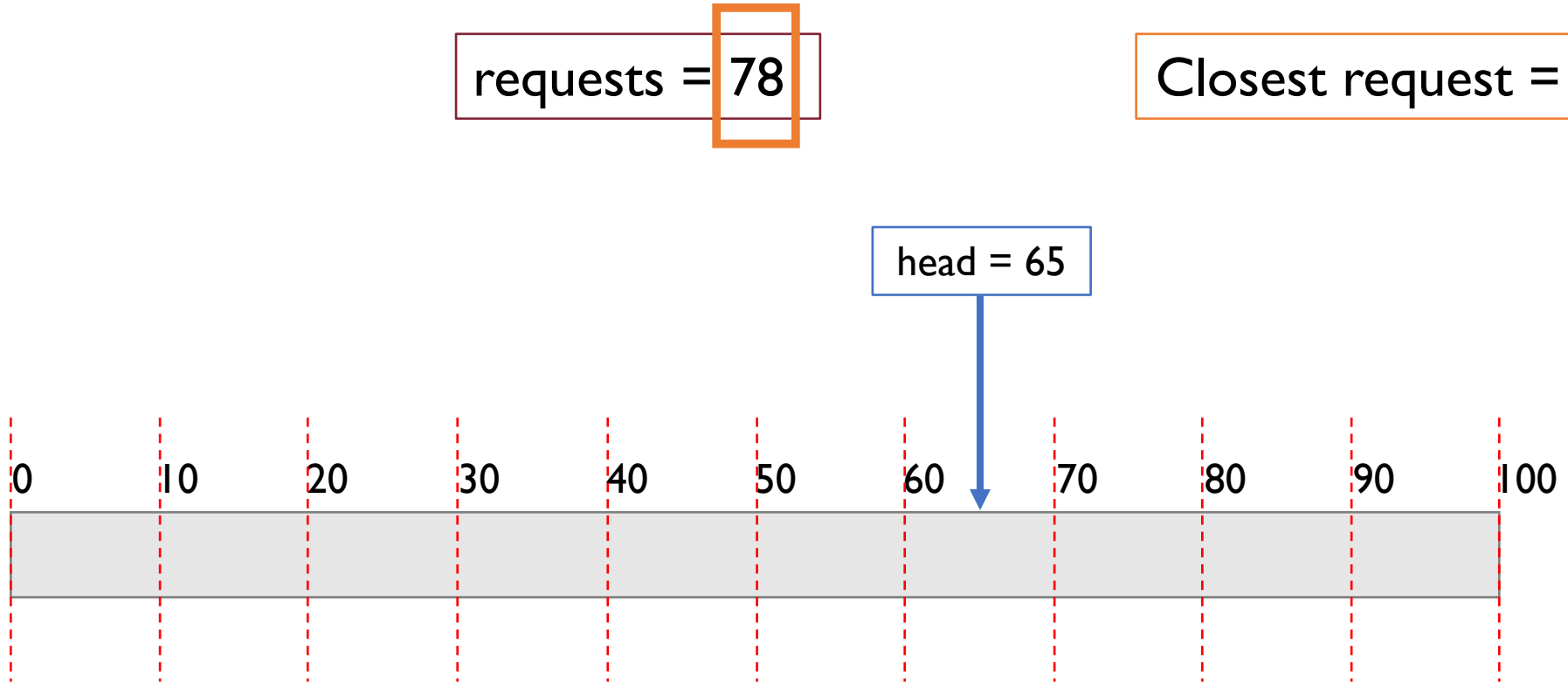
Disk Scheduling: Shortest Seek Time First (SSTF)

requests = 65, 78

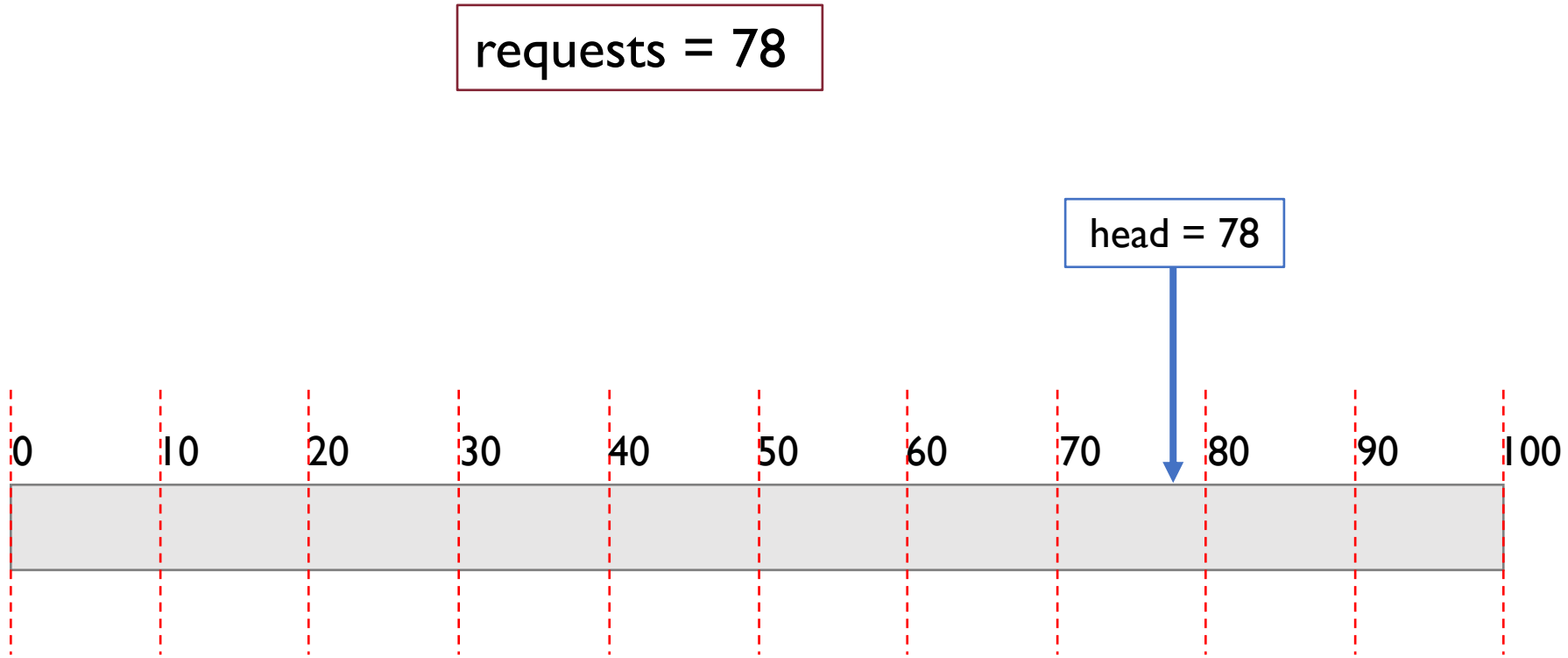


Distance travelled = 32 + |65 – 18| = 79

Disk Scheduling: Shortest Seek Time First (SSTF)

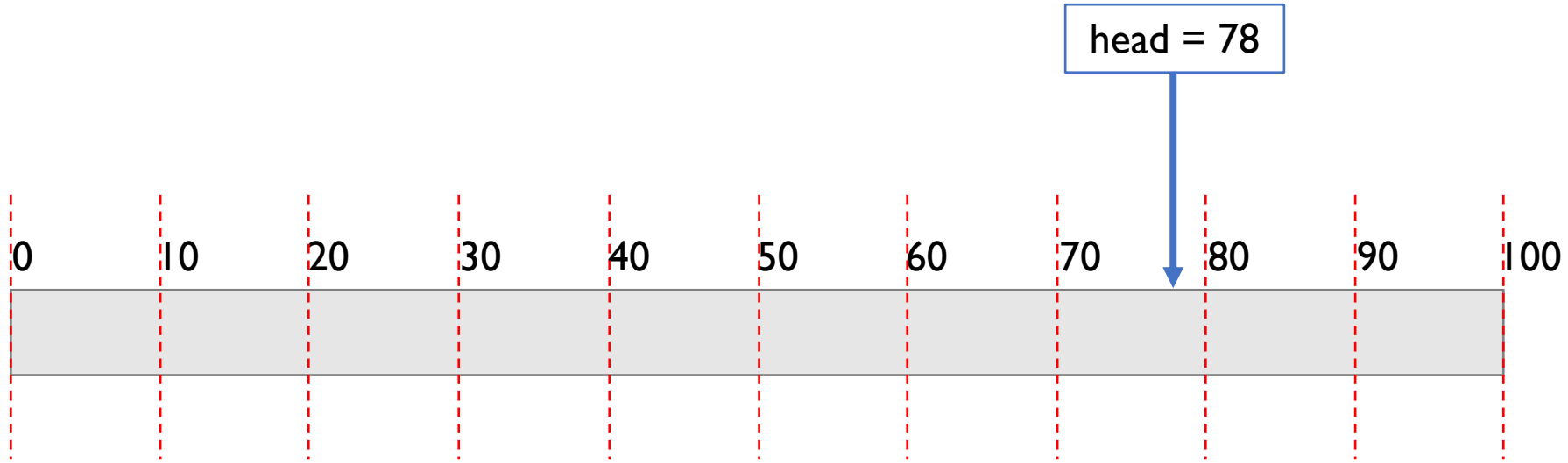


Disk Scheduling: Shortest Seek Time First (SSTF)



Disk Scheduling: Shortest Seek Time First (SSTF)

requests = 78



Distance travelled = 79 + |78 – 65| = **92**

SSTF: Considerations

- Implemented by keeping a doubly-linked list of requests (sorted)

SSTF: Considerations

- Implemented by keeping a doubly-linked list of requests (sorted)
- Overhead is negligible w.r.t. the speed of disk operations

SSTF: Considerations

- Implemented by keeping a doubly-linked list of requests (sorted)
- Overhead is negligible w.r.t. the speed of disk operations
- It may cause **starvation** (requests too far apart from the head)

SSTF: Considerations

- Implemented by keeping a doubly-linked list of requests (sorted)
- Overhead is negligible w.r.t. the speed of disk operations
- It may cause **starvation** (requests too far apart from the head)
- It is **not** overall optimal as greedily minimizes seek time (locally)
  - Example: head = 50; requests = 10, 20, 30, 40, 61

SSTF: Considerations

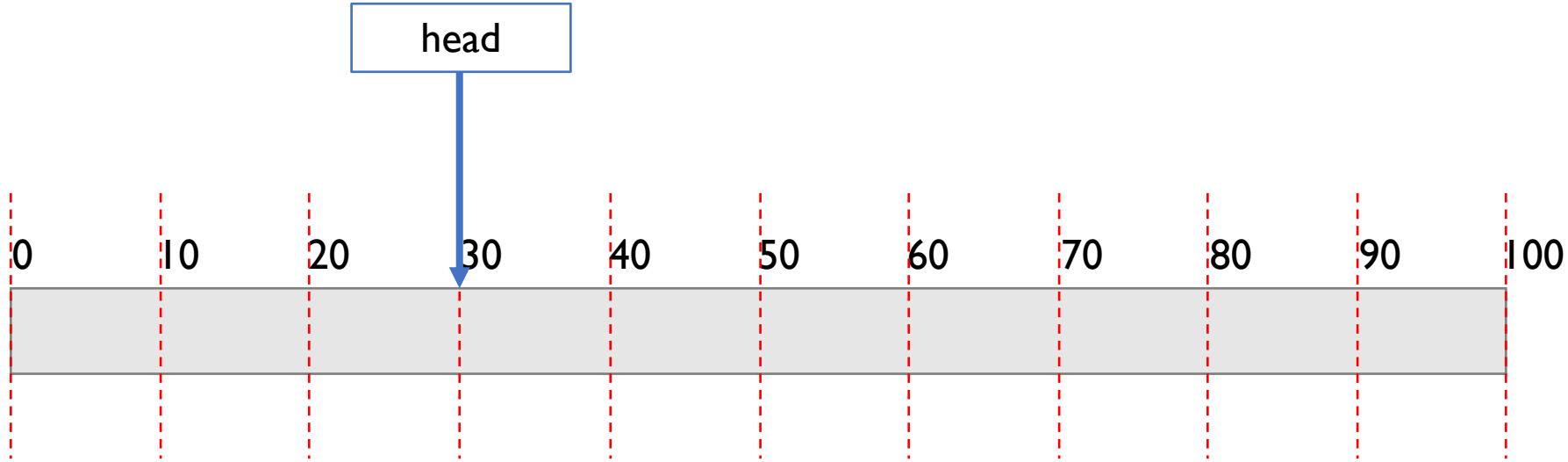
- Implemented by keeping a doubly-linked list of requests (sorted)
- Overhead is negligible w.r.t. the speed of disk operations
- It may cause **starvation** (requests too far apart from the head)
- It is **not** overall optimal as greedily minimizes seek time (locally)
  - Example: head = 50; requests = 10, 20, 30, 40, 61

SSTF only looks one step ahead

Disk Scheduling: SCAN

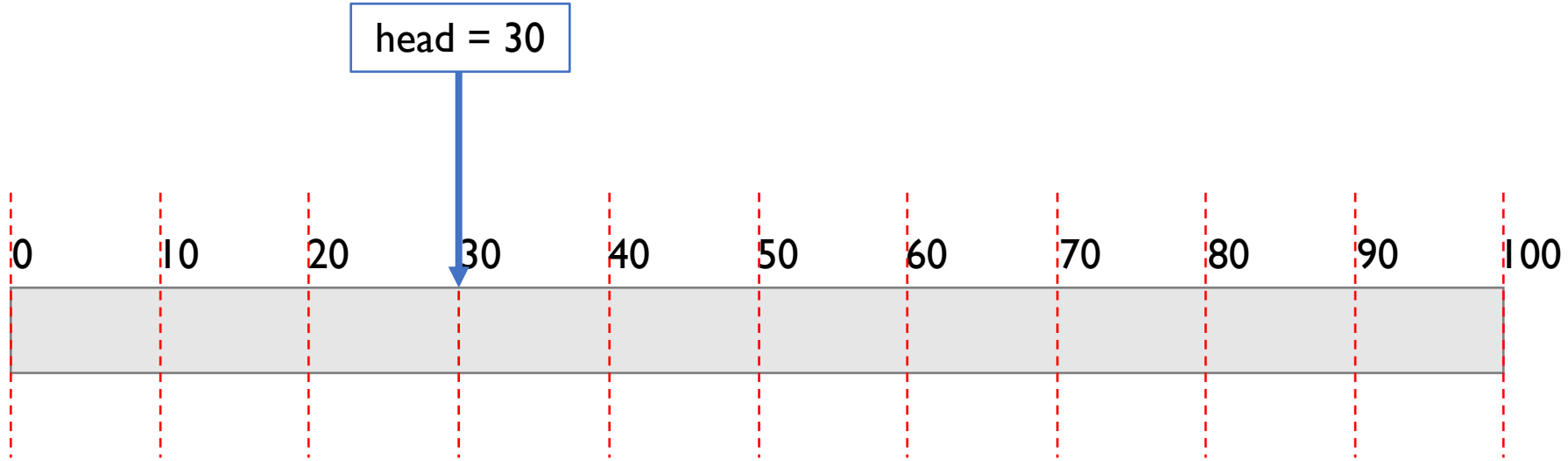
Head moves back and forth acoss the disk (e.g., track 0 to 100, 100 to 0, etc.)

Requests are served as the head passes (**elevator**)



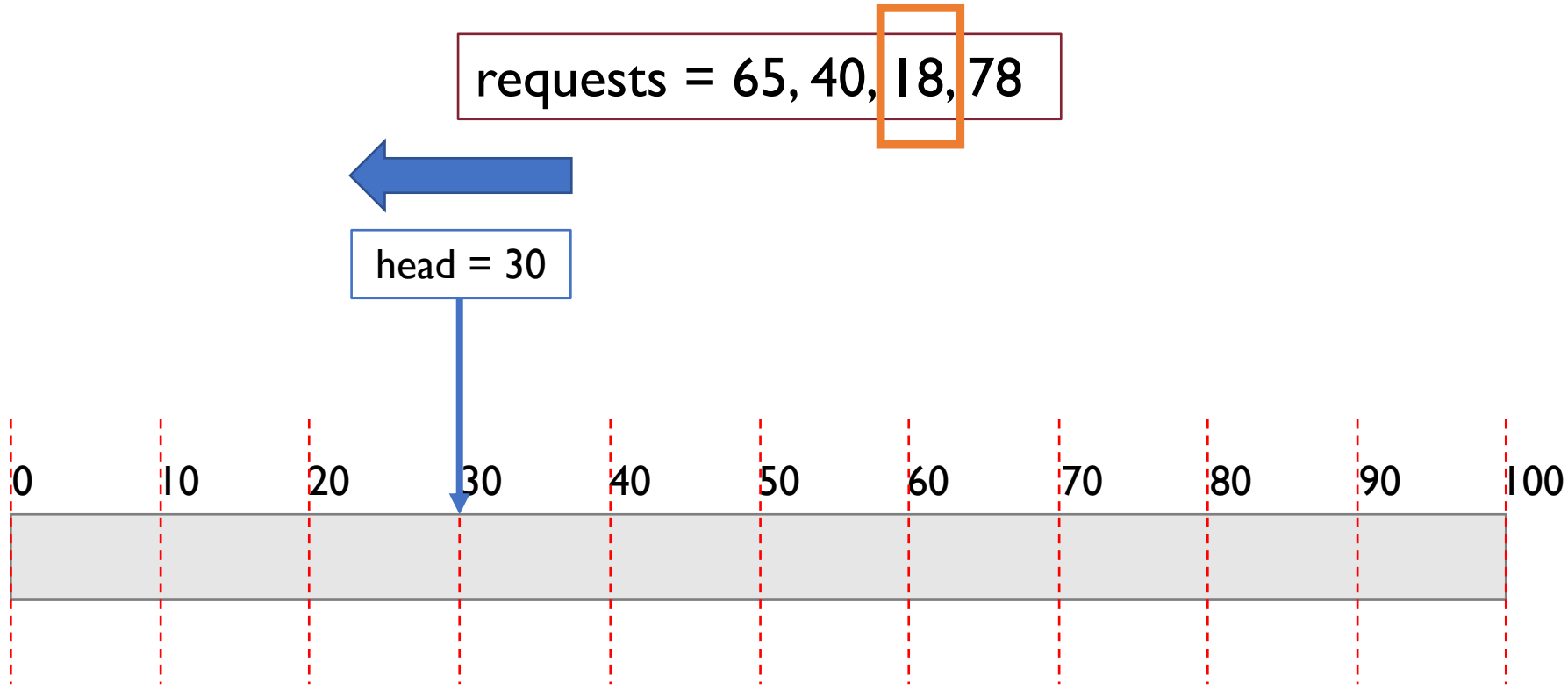
Disk Scheduling: SCAN

requests = 65, 40, 18, 78



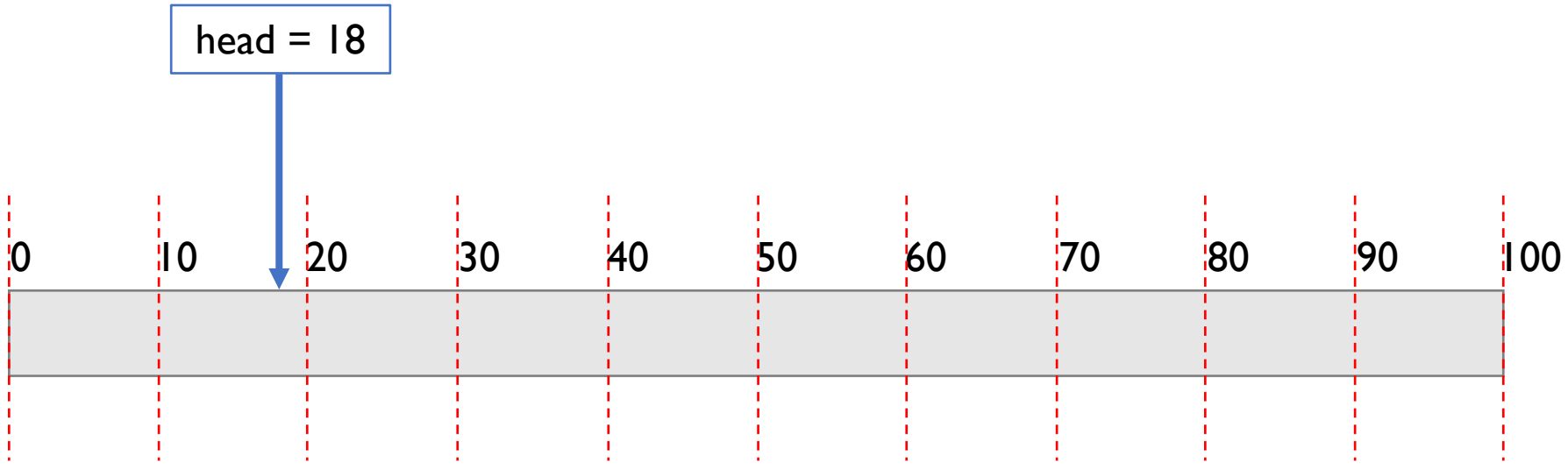
Distance travelled = 0

Disk Scheduling: SCAN



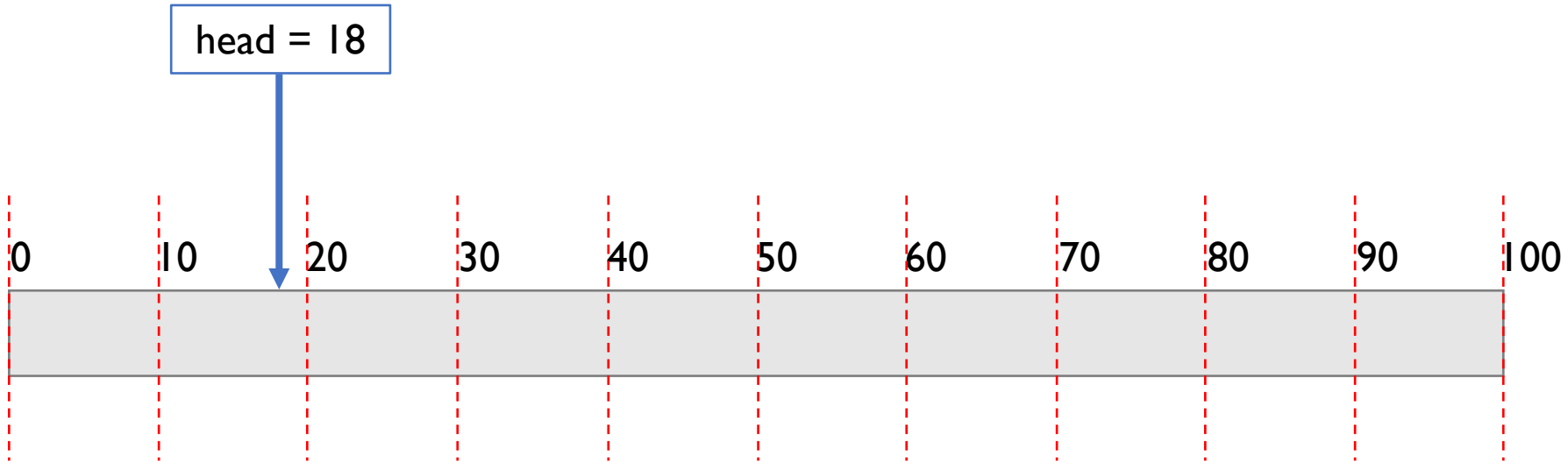
Disk Scheduling: SCAN

requests = 65, 40, 18, 78



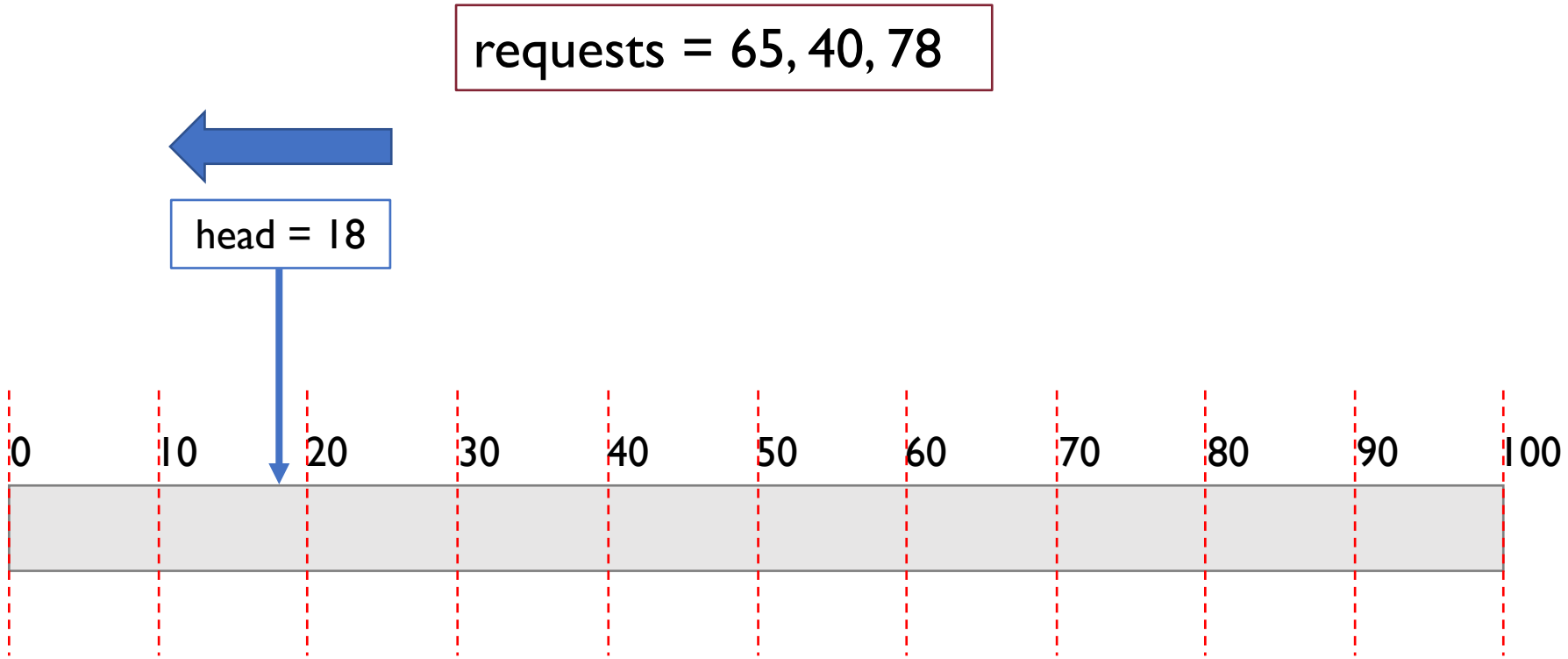
Disk Scheduling: SCAN

requests = 65, 40, 18, 78



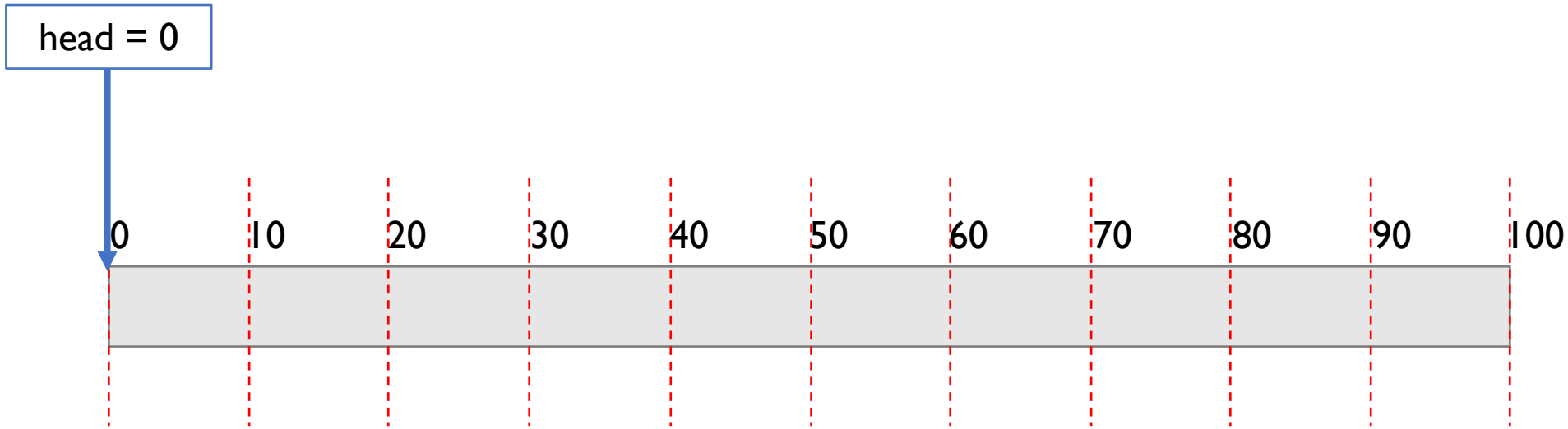
Distance travelled = 0 + |18 – 30| = 12

Disk Scheduling: SCAN



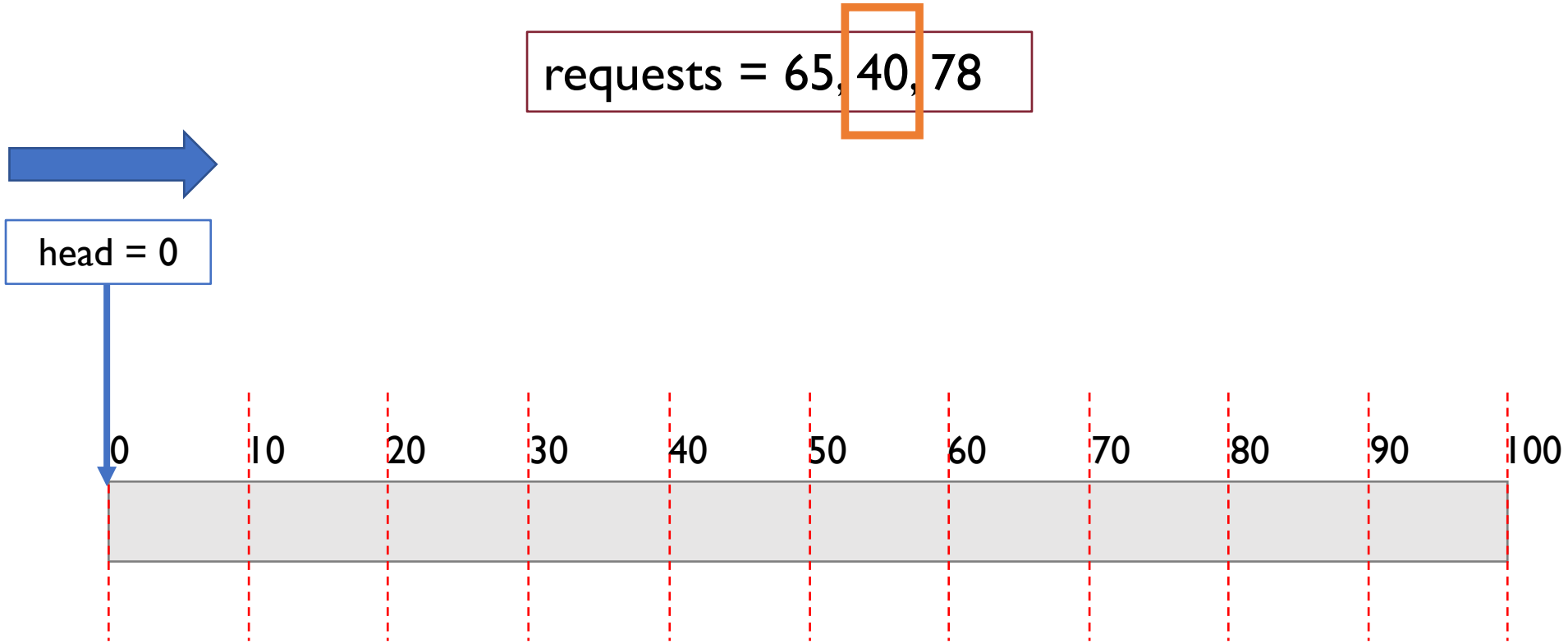
Disk Scheduling: SCAN

requests = 65, 40, 78

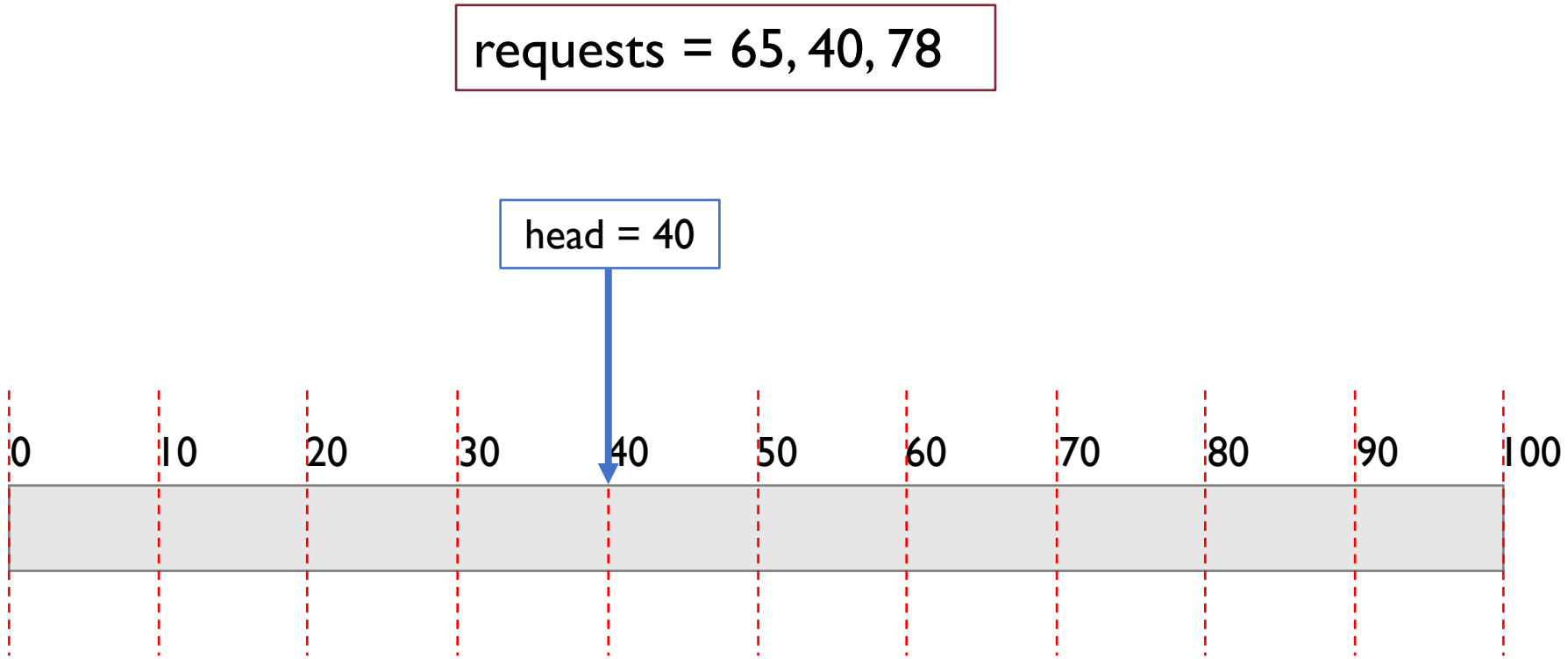


Distance travelled = 12 + |0 – 18| = 30

Disk Scheduling: SCAN

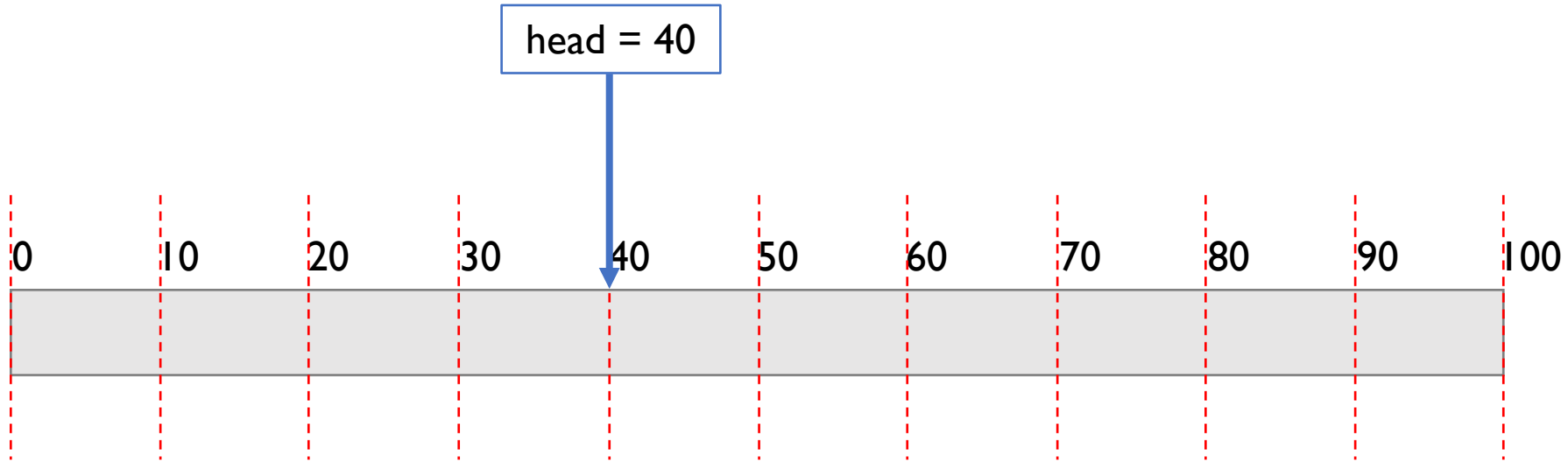


Disk Scheduling: SCAN



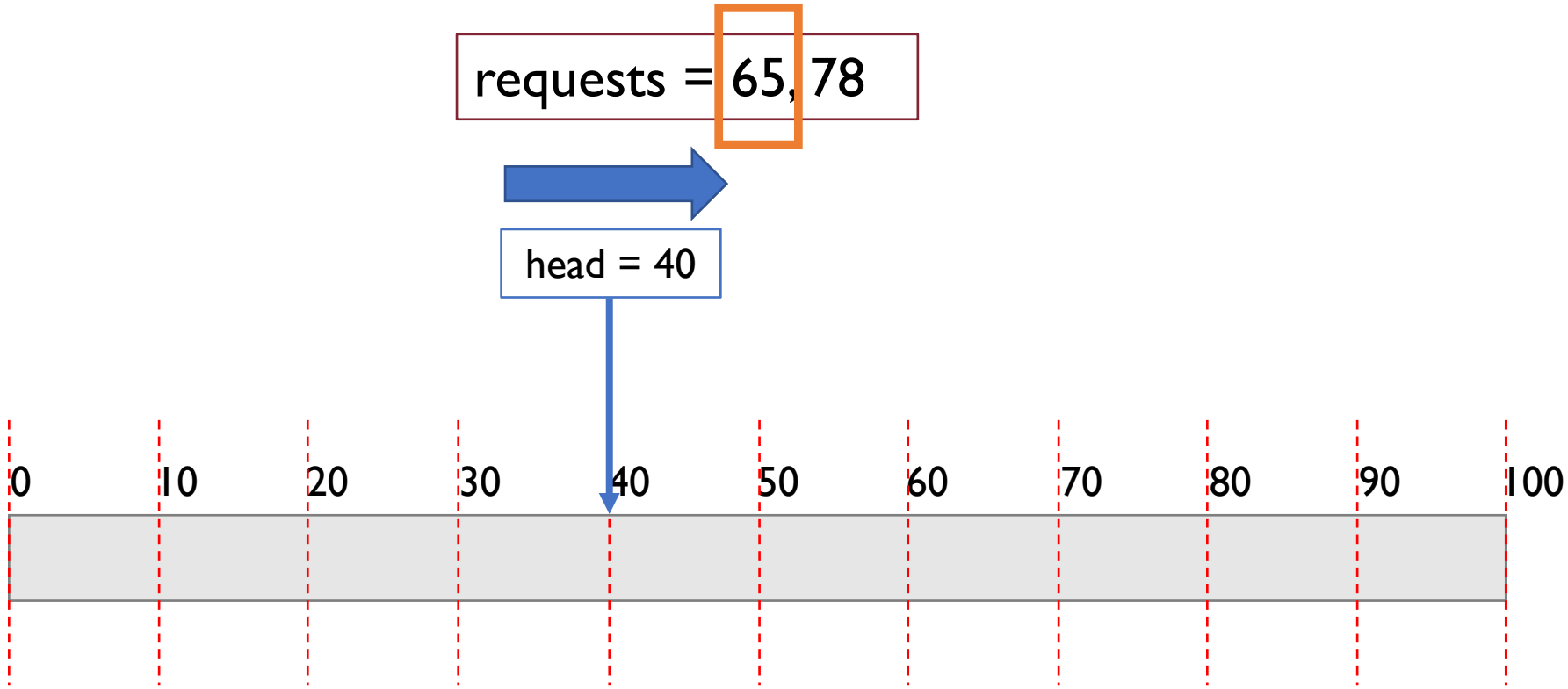
Disk Scheduling: SCAN

requests = 65, 40, 78



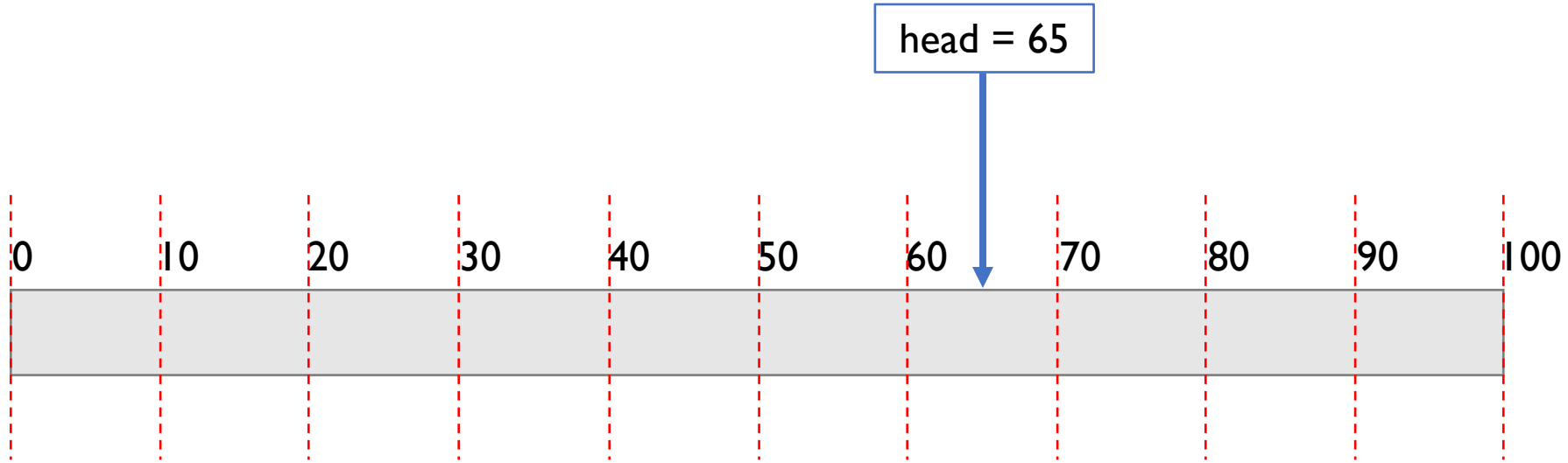
Distance travelled = 30 + |40 – 0| = 70

Disk Scheduling: SCAN



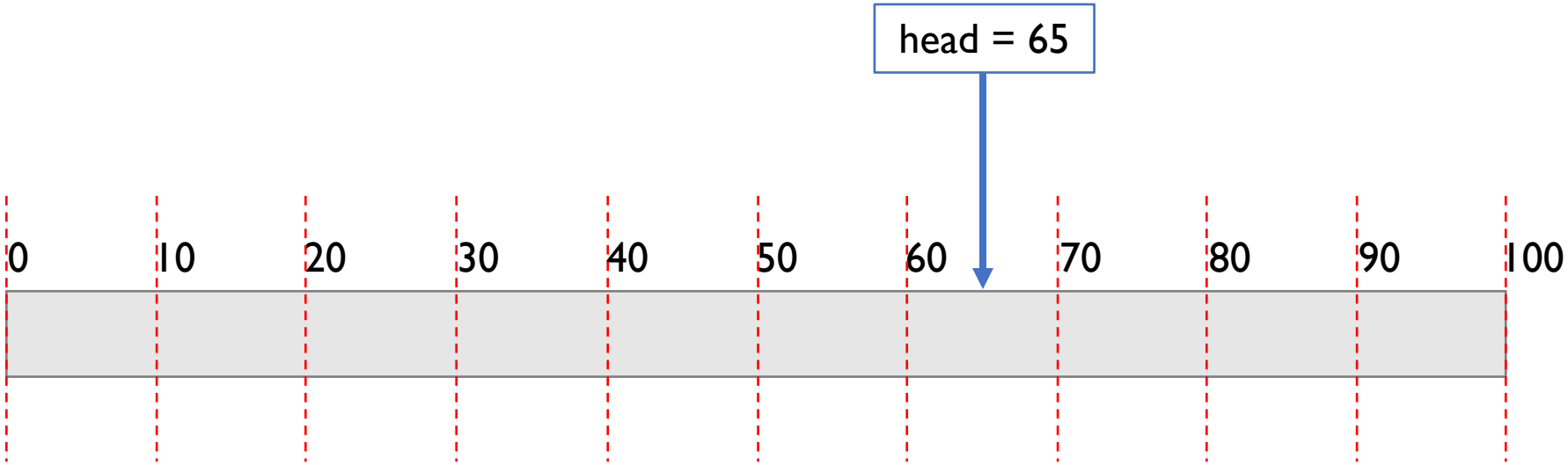
Disk Scheduling: SCAN

requests = 65, 78



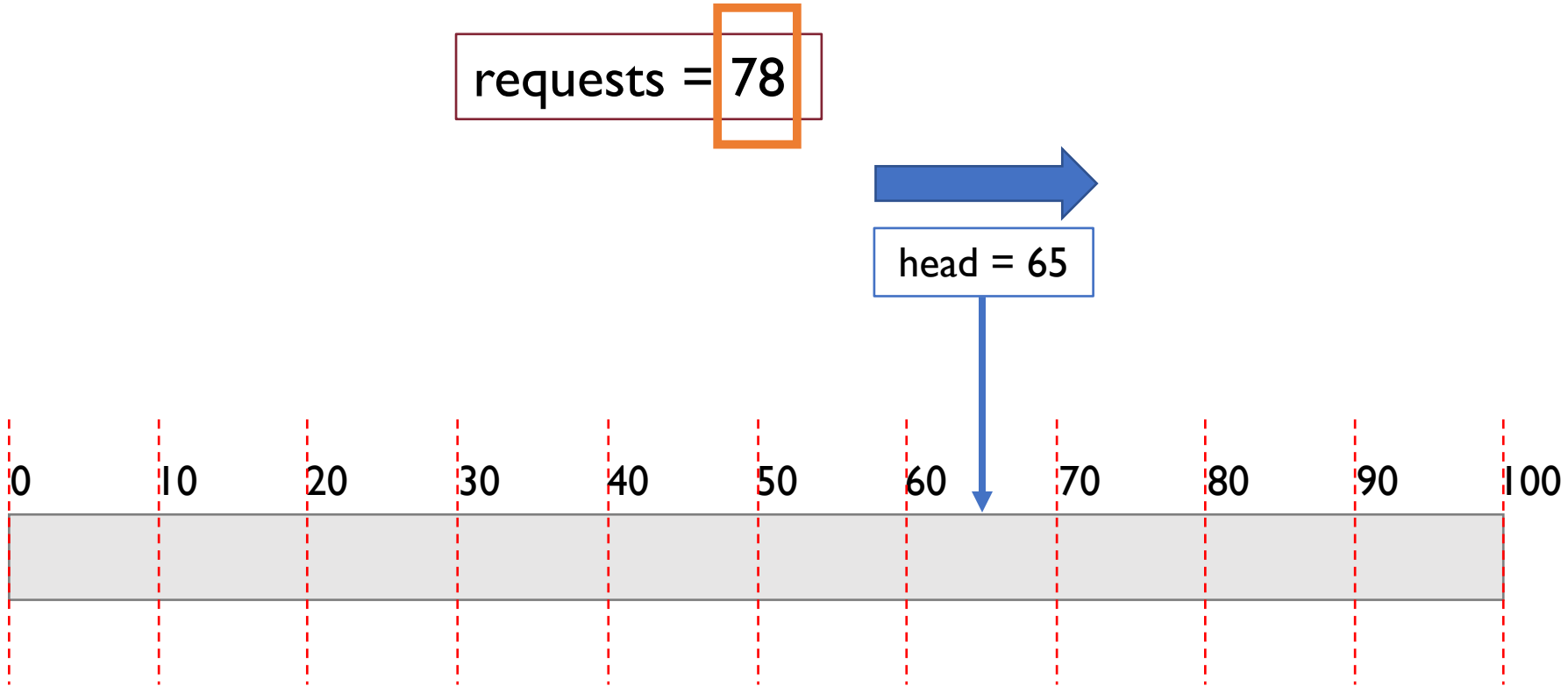
Disk Scheduling: SCAN

requests = 65, 78

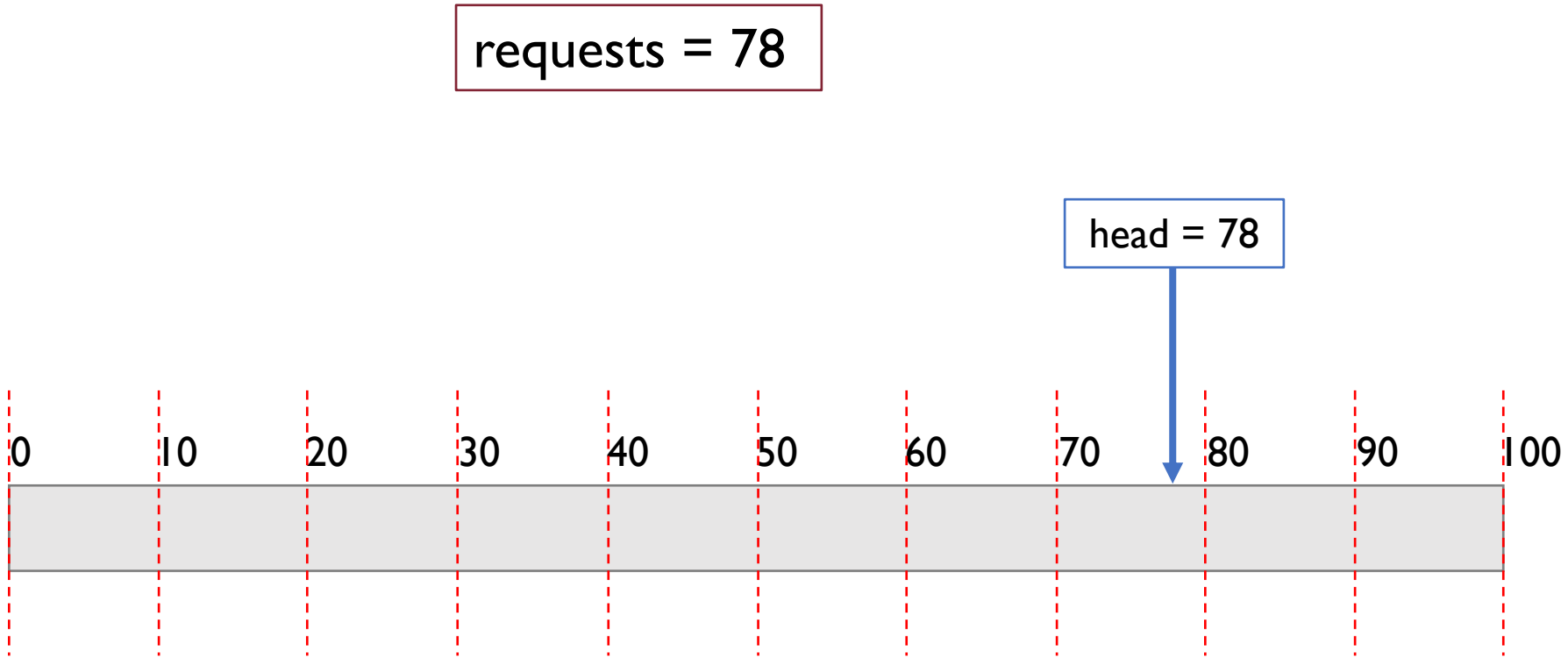


Distance travelled = 70 + |65 – 40| = 95

Disk Scheduling: SCAN

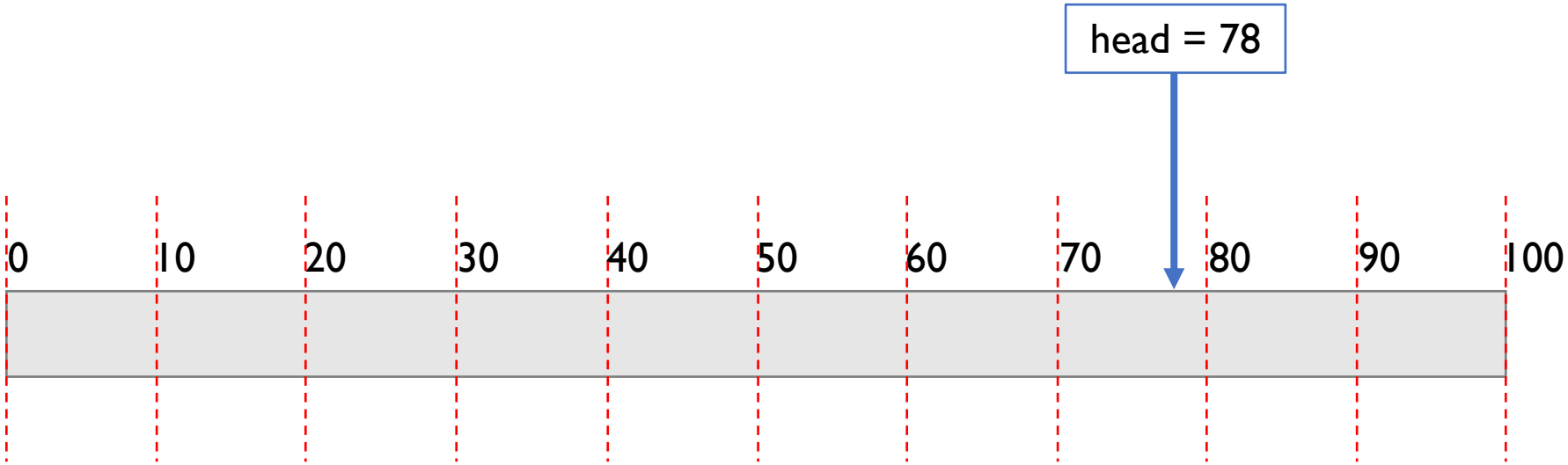


Disk Scheduling: SCAN



Disk Scheduling: SCAN

requests = 78



Distance travelled = 95 + |78 – 65| = **108**

SCAN: Considerations

- Requires to keep a sorted list of requests

SCAN: Considerations

- Requires to keep a sorted list of requests
- Simple optimization (LOOK)
  - Do not go all the way to the edge of the disk each time
  - Just go as far as the last request to be served
  - In the example: no need to go from 18 to 0! Just stop at 18 (first request)
  - Total distance goes from **108** down to **72** (saving 18-0 and 0-18 movements)

SCAN vs. SSTF

- SCAN does not suffer from possible starvation as SSTF does
  - We are guaranteed each request will be served eventually

SCAN vs. SSTF

- SCAN does not suffer from possible starvation as SSTF does
  - We are guaranteed each request will be served eventually
- However, seek time might be larger with SCAN than SSTF when:
  - New requests are coming in immediately after SCAN has passed
  - In such cases, SSTF will serve "closer" request first, whilst SCAN will complete the whole pass to the opposite edge before going back
  - E.g., SCAN is going upwards and just passes track 50, then request 49 comes in

SCAN vs. SSTF

- SCAN does not suffer from possible starvation as SSTF does
  - We are guaranteed each request will be served eventually
- However, seek time might be larger with SCAN than SSTF when:
  - New requests are coming in immediately after SCAN has passed
  - In such cases, SSTF will serve "closer" request first, whilst SCAN will complete the whole pass to the opposite edge before going back
  - E.g., SCAN is going upwards and just passes track 50, then request 49 comes in
- SCAN may prioritize serving more recent requests than older ones

Appendix: Arm Speed

- We assumed the time it takes to the head to move from track to track is a linear function of the traversed tracks
  - i.e., moving from track 10 to 20 takes as twice as much time than moving from track 10 to 15 (10 vs. 5 tracks traversed)

Appendix: Arm Speed

- We assumed the time it takes to the head to move from track to track is a linear function of the traversed tracks
  - i.e., moving from track 10 to 20 takes as twice as much time than moving from track 10 to 15 (10 vs. 5 tracks traversed)
- However, this is not the case! Remember that this is all mechanical!

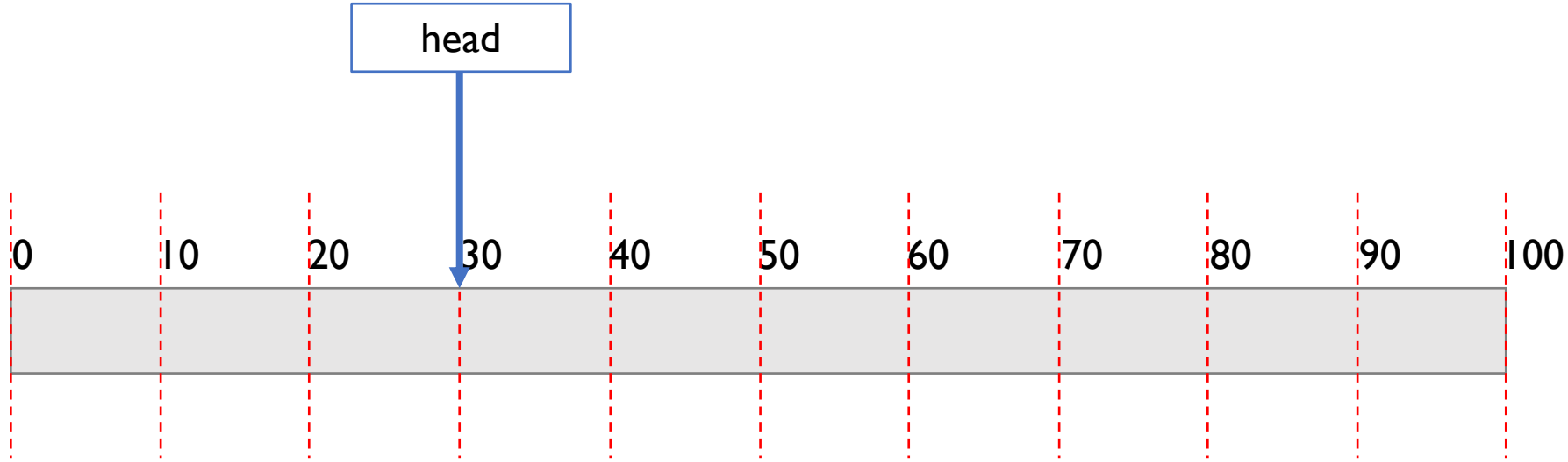
Appendix: Arm Speed

- We assumed the time it takes to the head to move from track to track is a linear function of the traversed tracks
  - i.e., moving from track 10 to 20 takes as twice as much time than moving from track 10 to 15 (10 vs. 5 tracks traversed)
- However, this is not the case! Remember that this is all mechanical!
- Disk arms are subject to acceleration and deceleration, and their speed is not constant (as opposed to rotational speed)

Disk Scheduling: C-SCAN

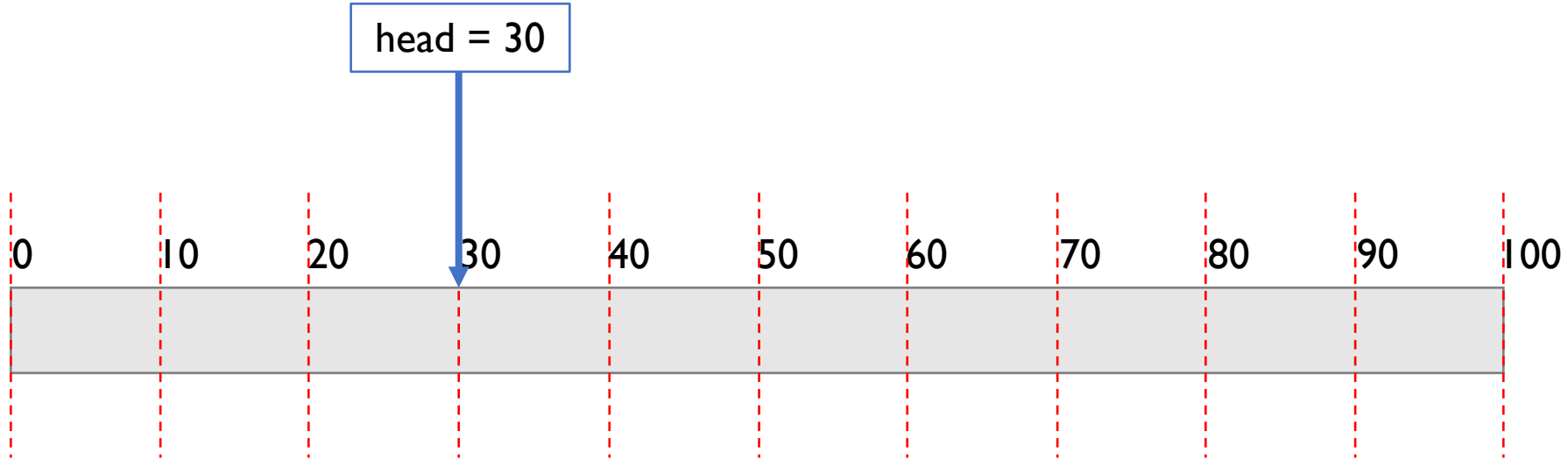
Head makes circular scan of the disk (requests are in a circular queue)

Each time the head reaches an end it is reset to the opposite end



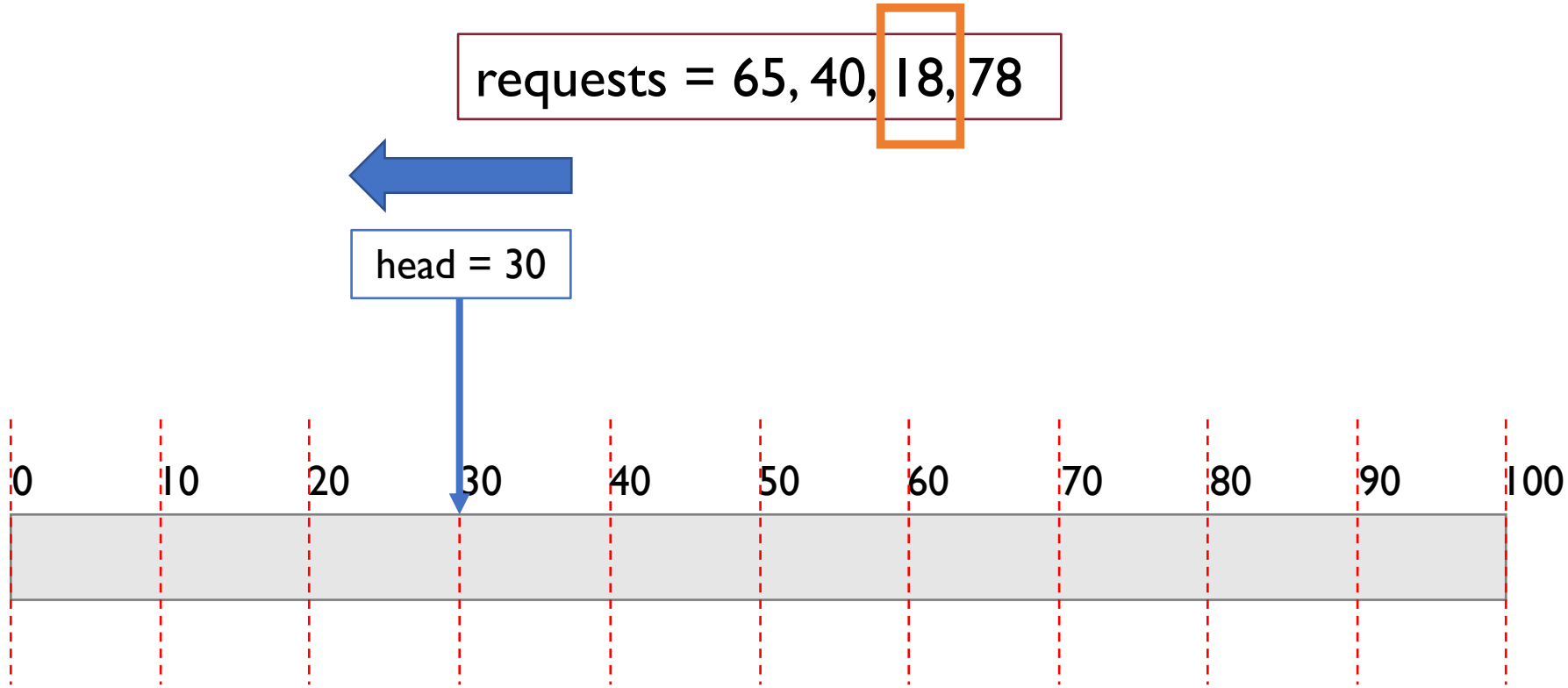
Disk Scheduling: C-SCAN

requests = 65, 40, 18, 78



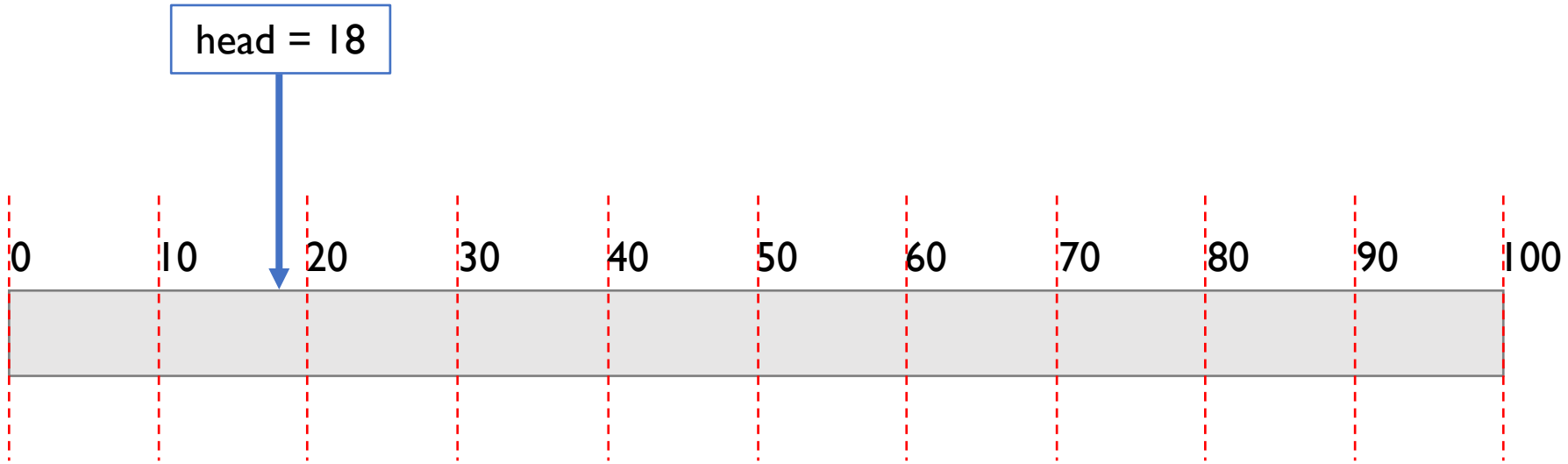
Distance travelled = 0

Disk Scheduling: C-SCAN



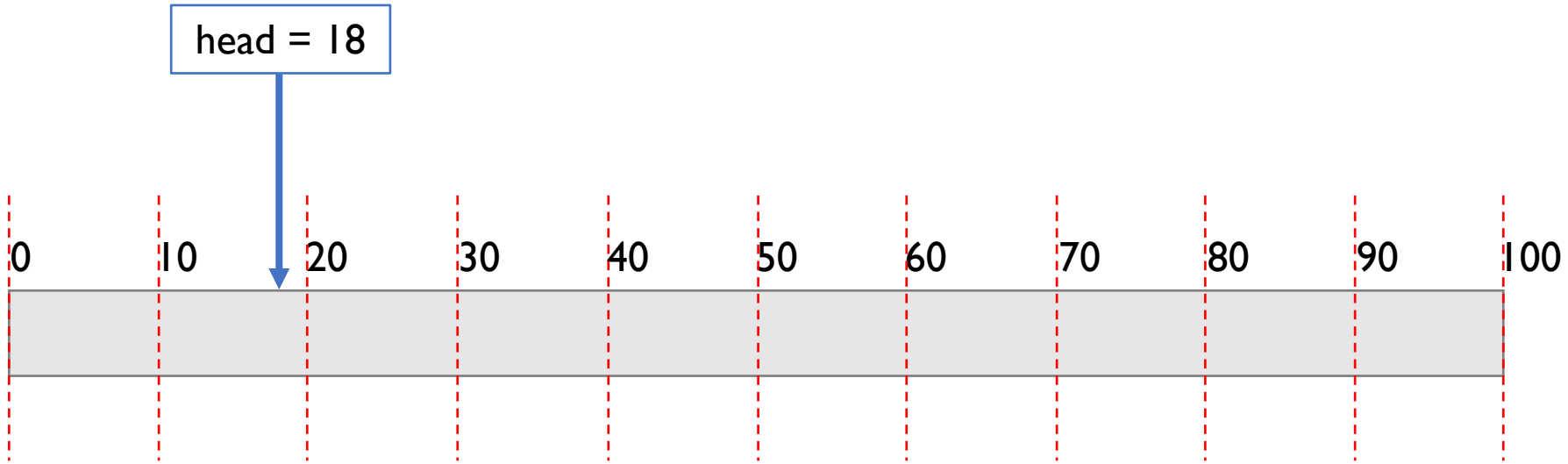
Disk Scheduling: C-SCAN

requests = 65, 40, 18, 78



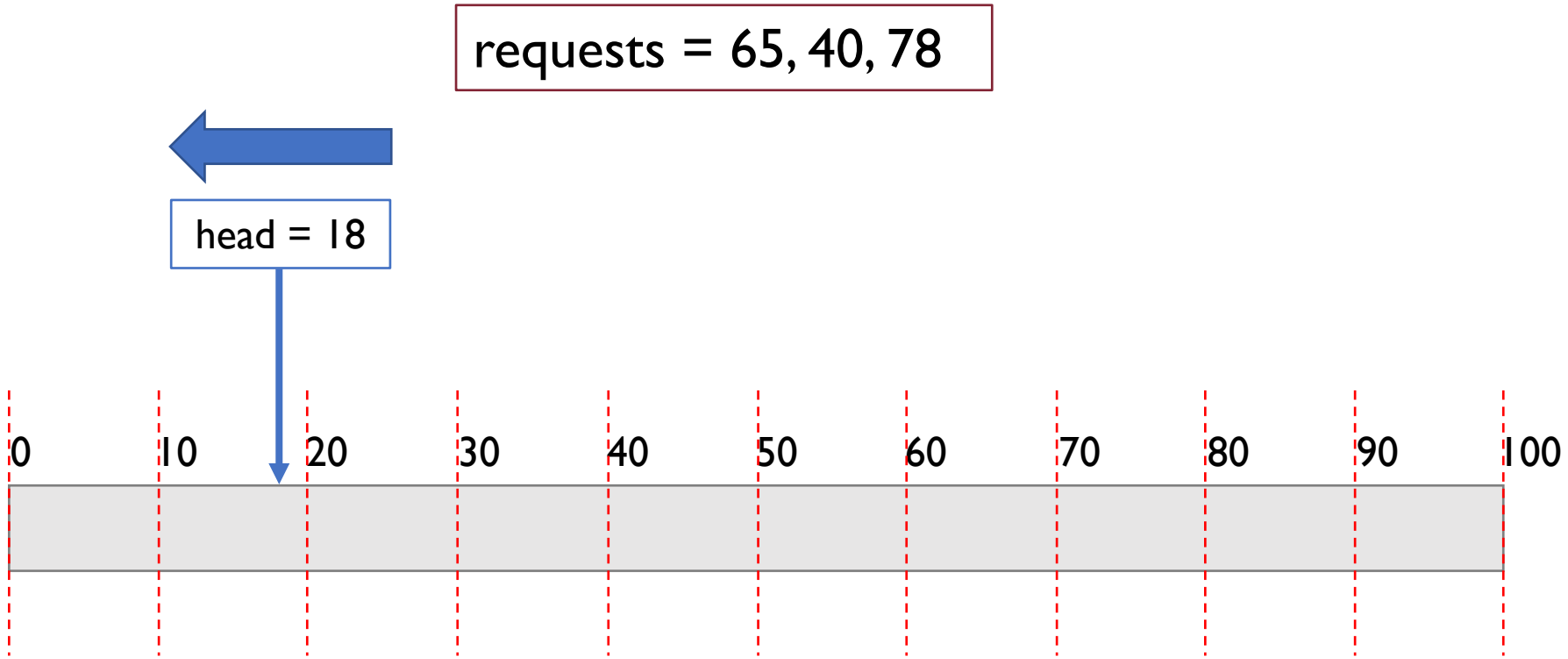
Disk Scheduling: C-SCAN

requests = 65, 40, 18, 78



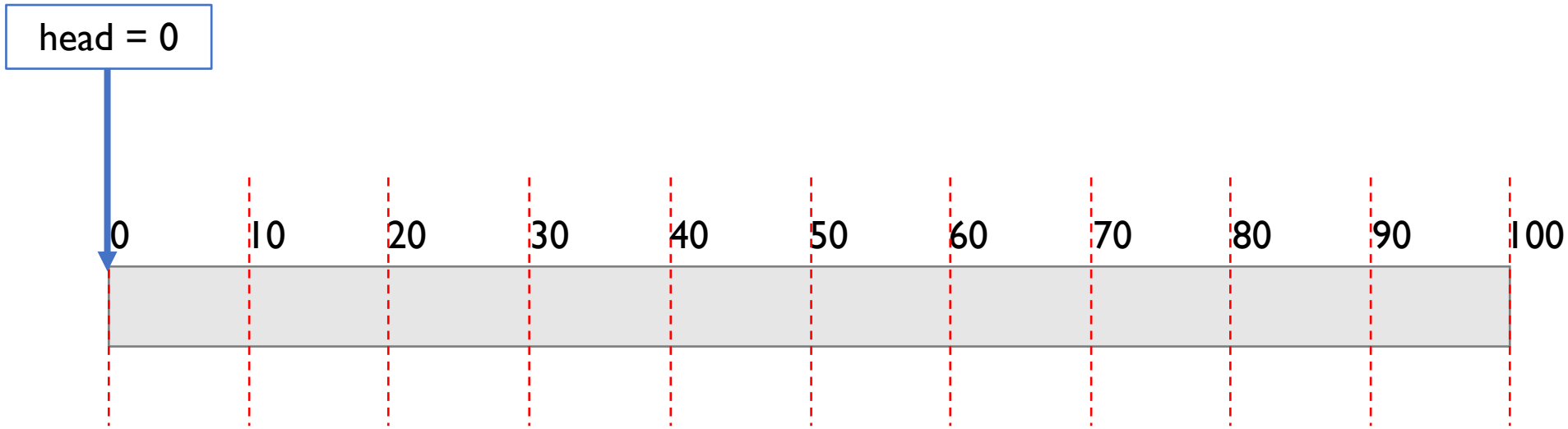
Distance travelled = 0 + |18 – 30| = 12

Disk Scheduling: C-SCAN



Disk Scheduling: C-SCAN

requests = 65, 40, 78

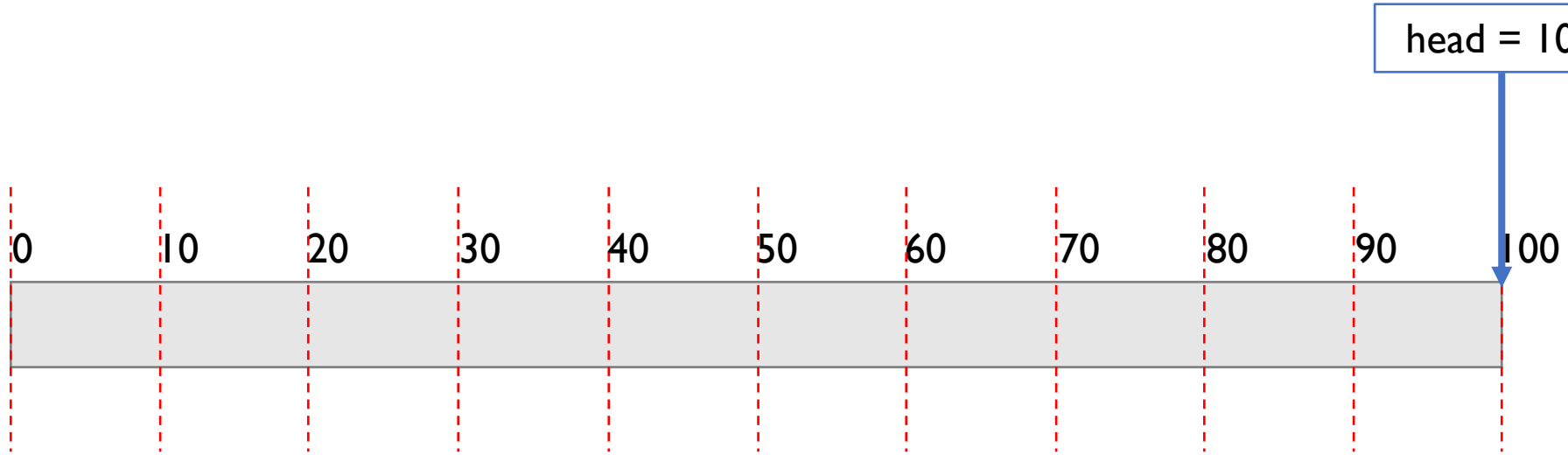


Distance travelled = 12 + |0 – 18| = 30

Disk Scheduling: C-SCAN

requests = 65, 40, 78

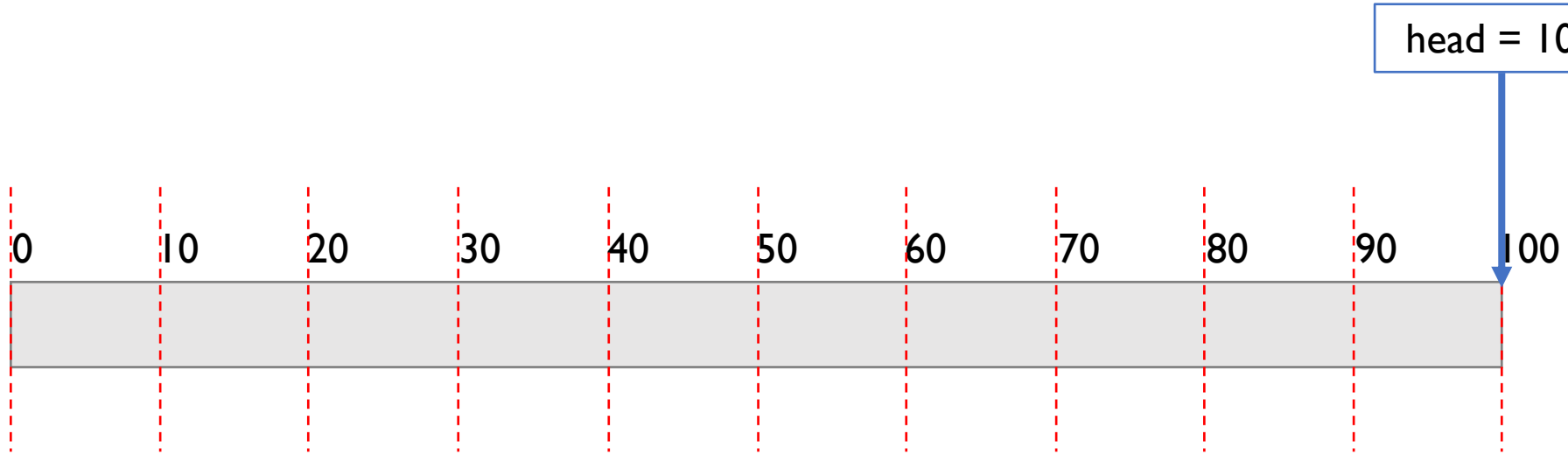
Head reset



Disk Scheduling: C-SCAN

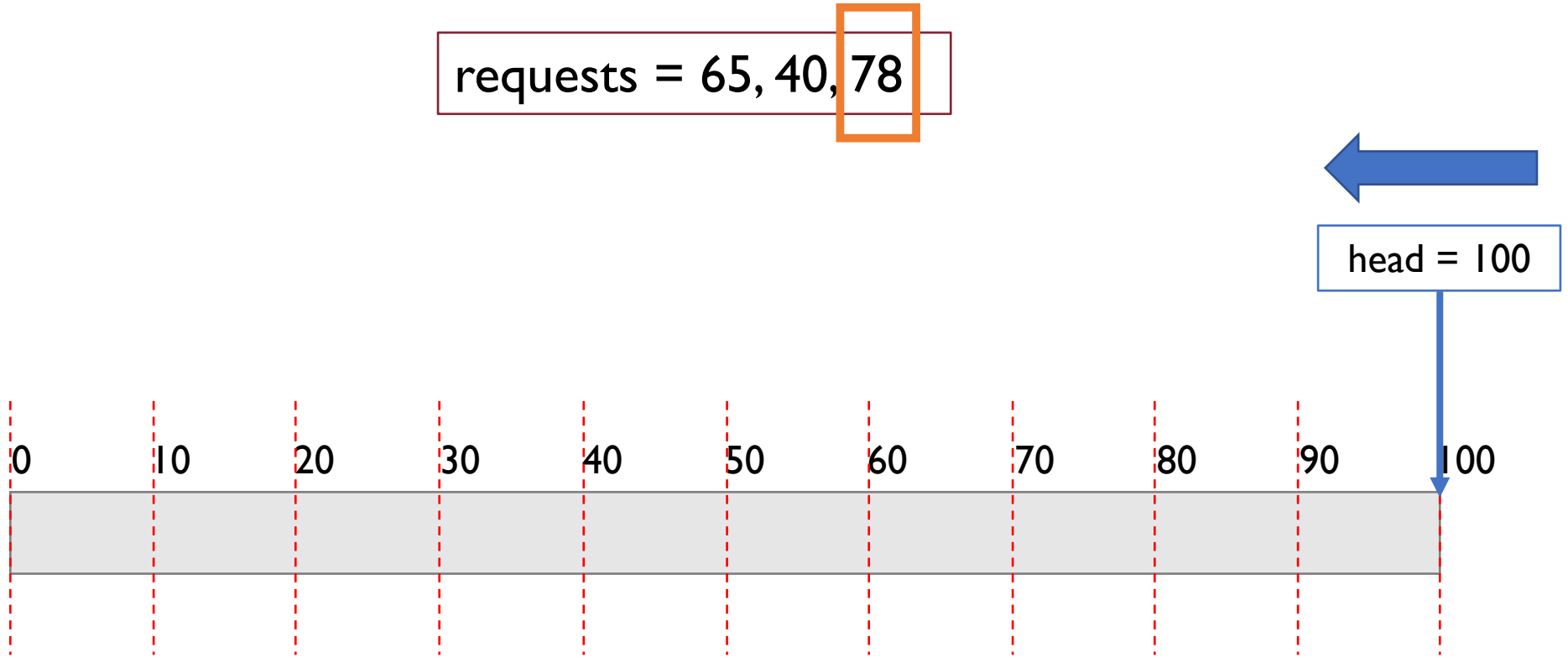
requests = 65, 40, 78

Head reset



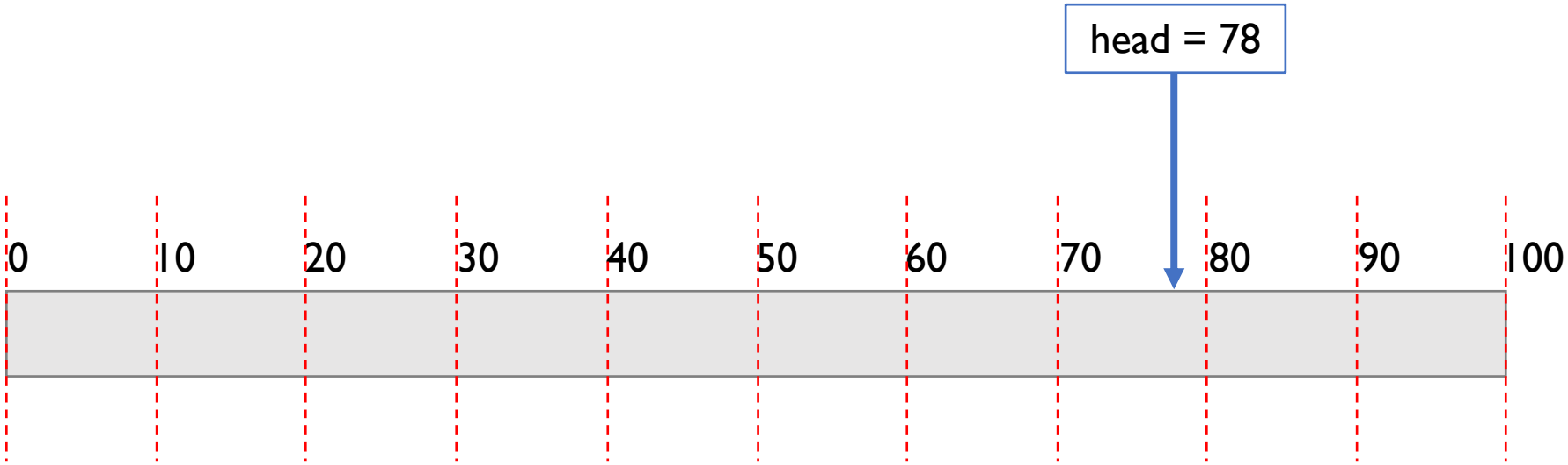
Distance travelled = 30 + |100 – 0| = 130

Disk Scheduling: C-SCAN



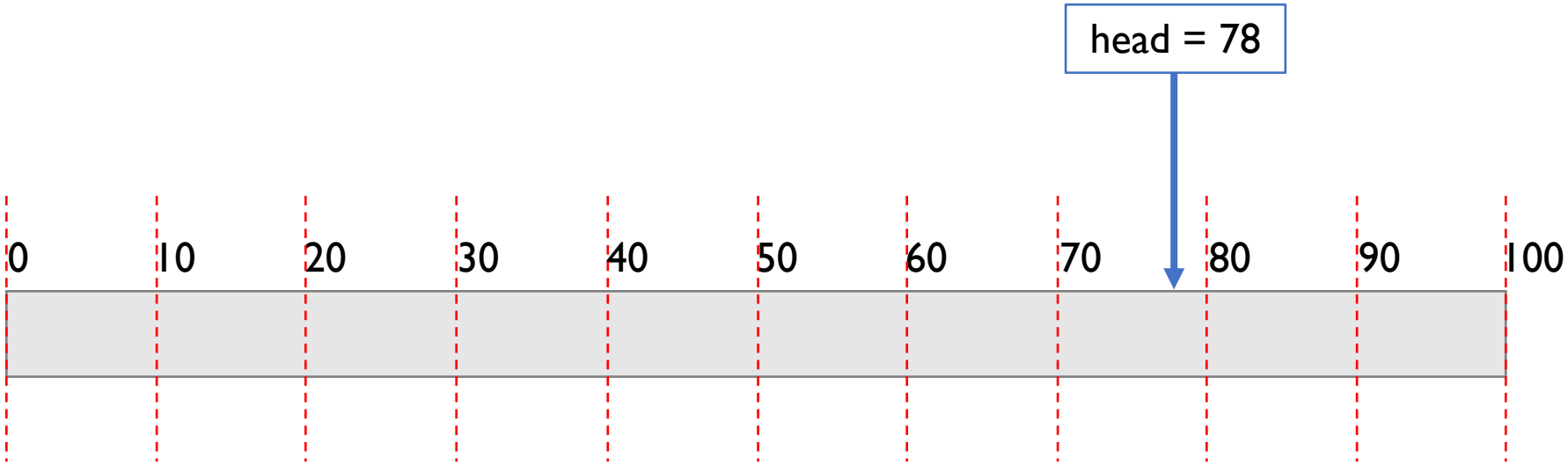
Disk Scheduling: C-SCAN

requests = 65, 40, 78



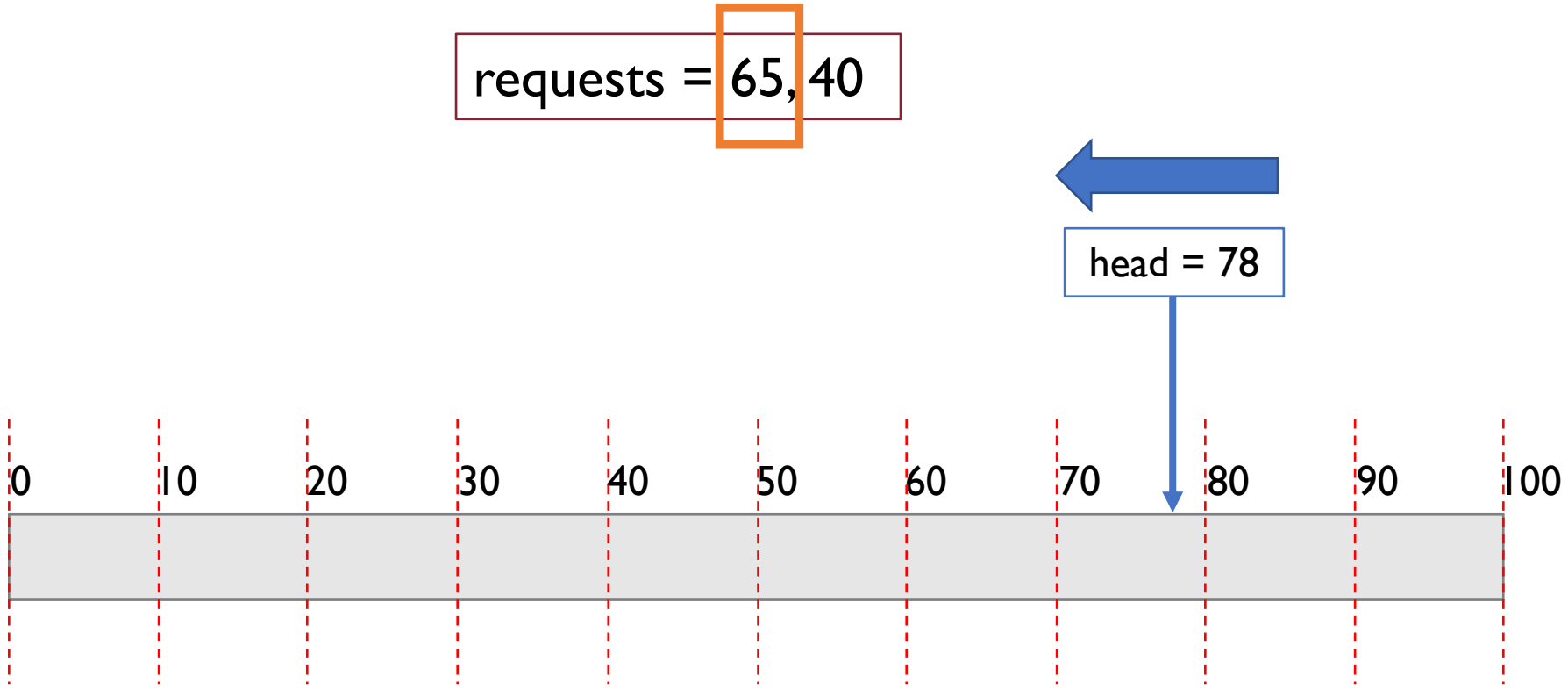
Disk Scheduling: C-SCAN

requests = 65, 40, 78

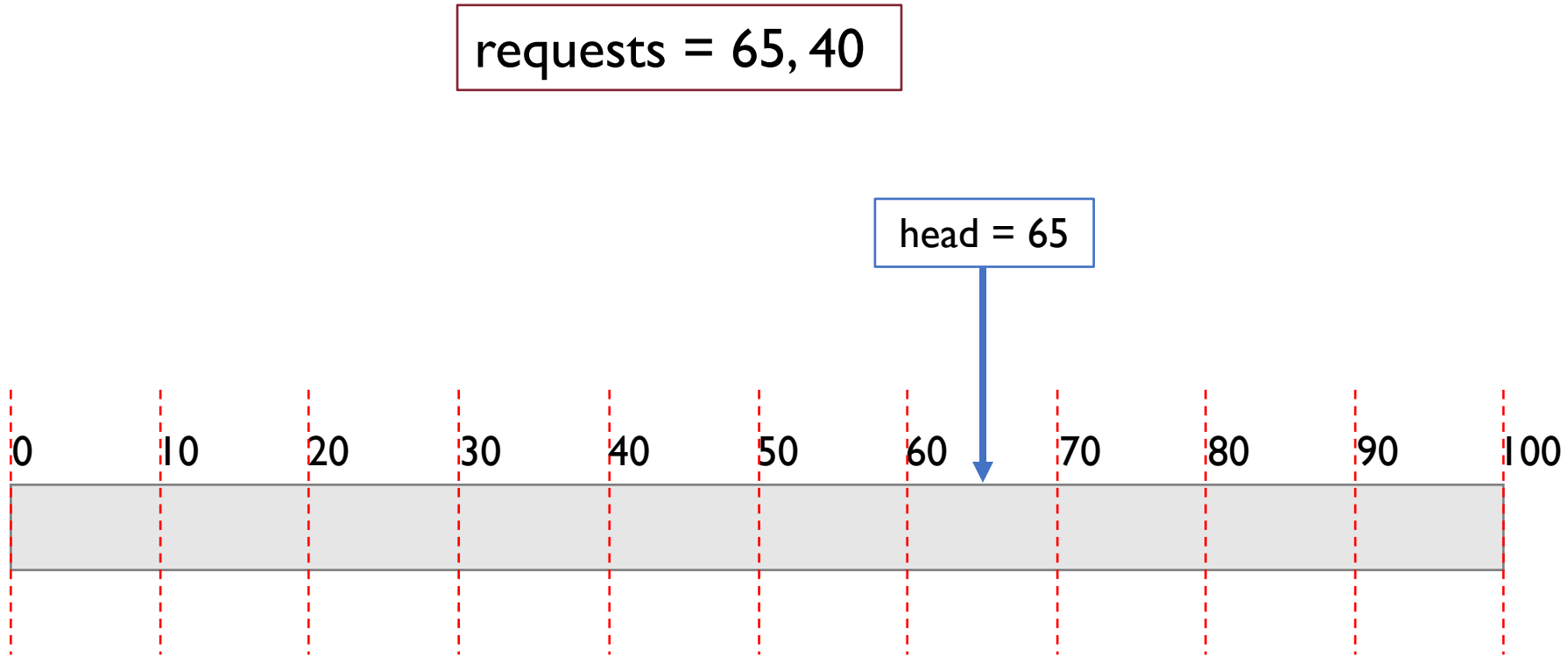


Distance travelled = 130 + |78 – 100| = 152

Disk Scheduling: C-SCAN

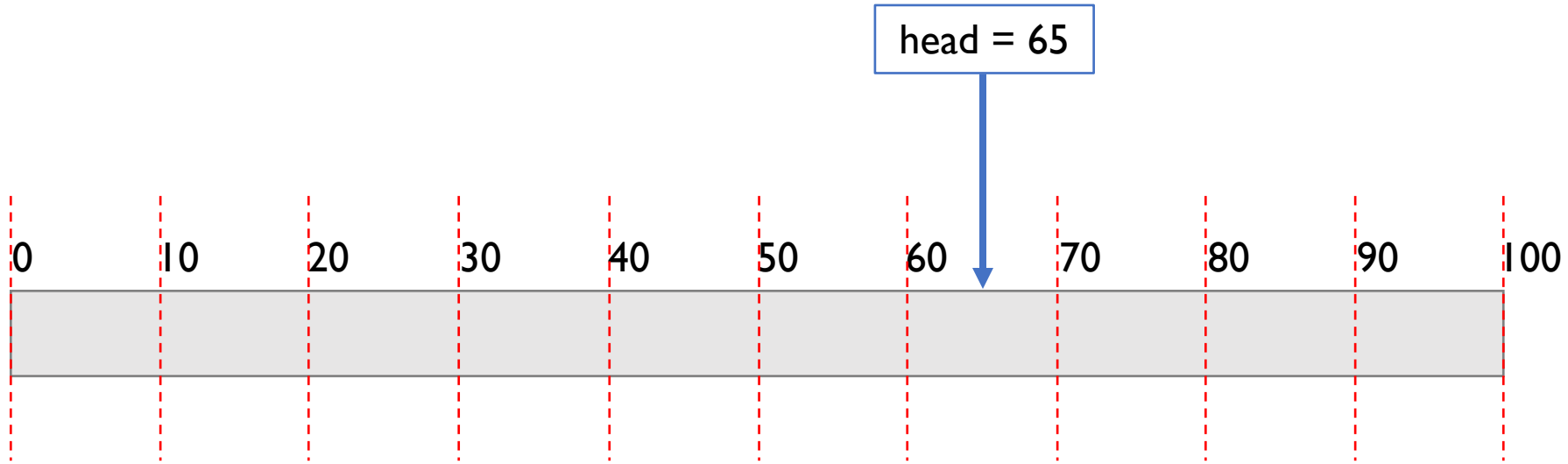


Disk Scheduling: C-SCAN



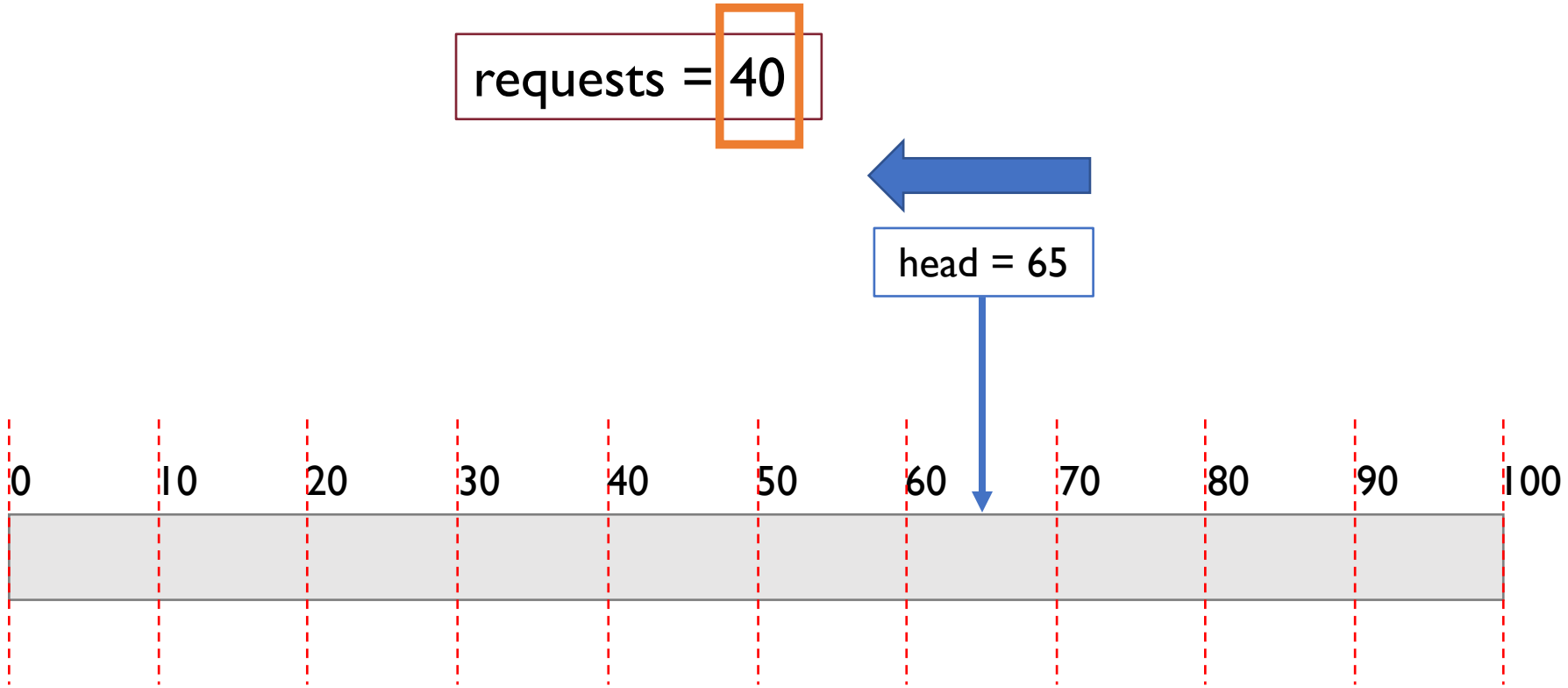
Disk Scheduling: C-SCAN

requests = 65, 40

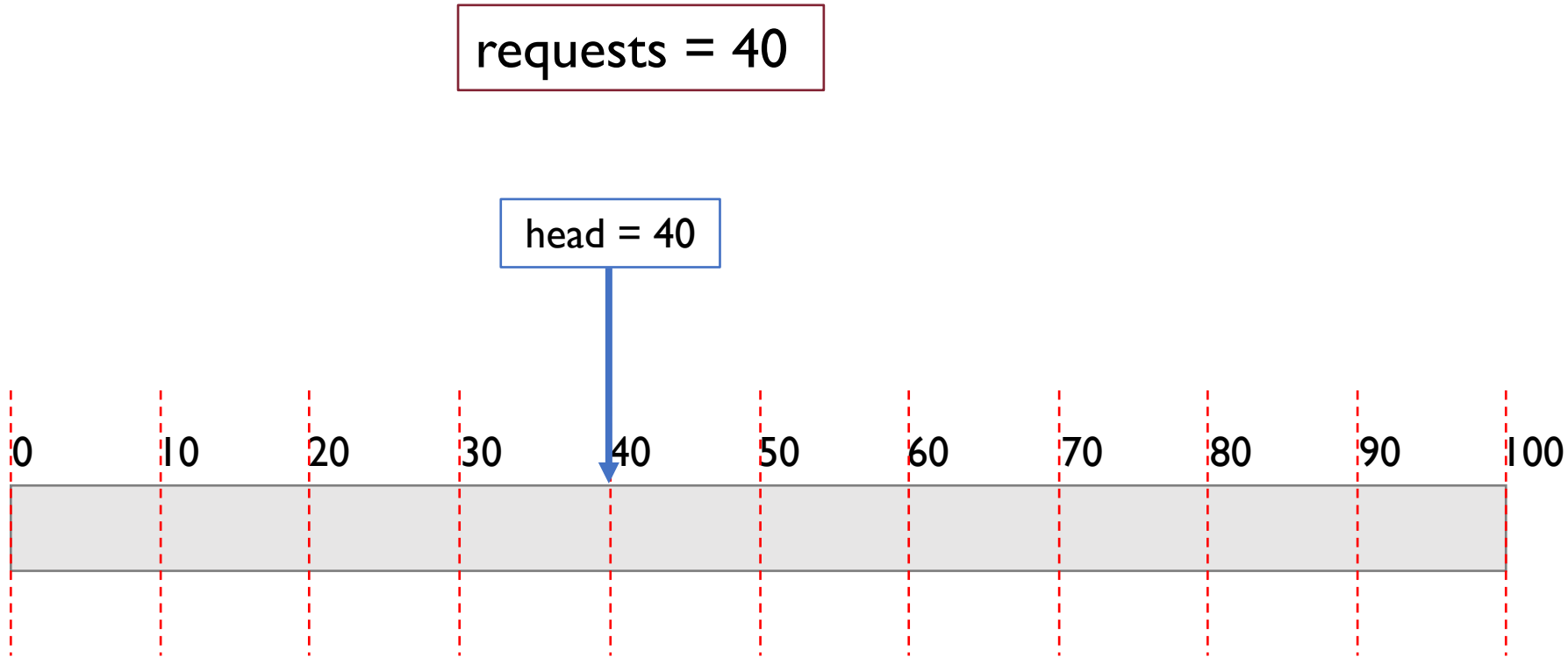


Distance travelled = 152 + |65 – 78| = 165

Disk Scheduling: C-SCAN

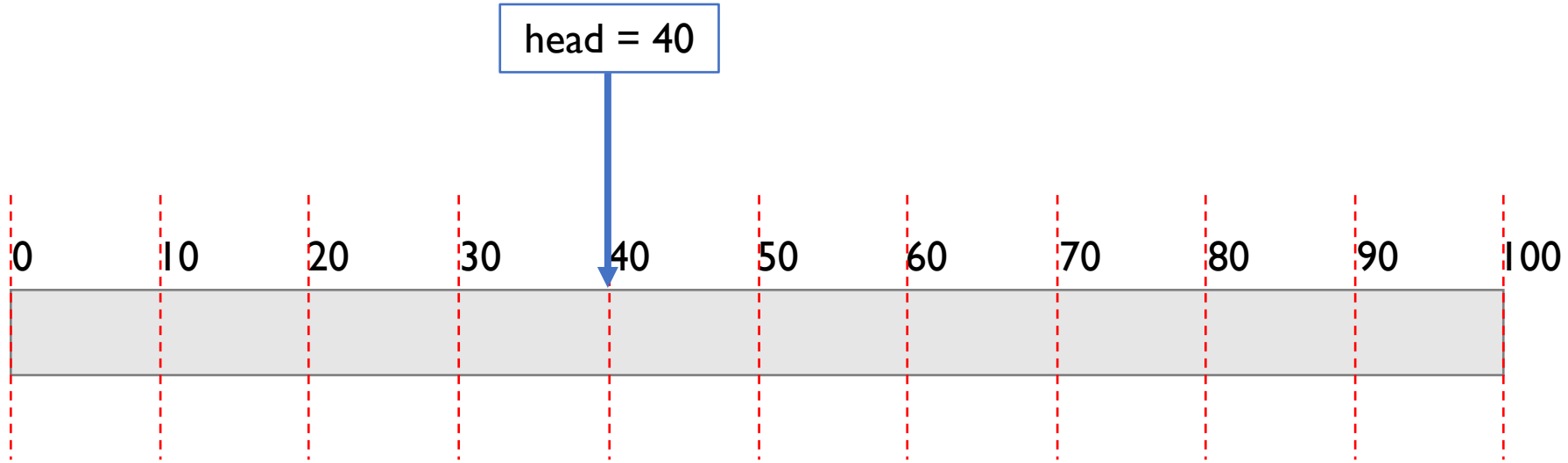


Disk Scheduling: C-SCAN



Disk Scheduling: C-SCAN

requests = 40



Distance travelled = 165 + |40 – 65| = **190**

C-SCAN: Considerations

- C-LOOK: similar optimization to LOOK
  - In the example:
    - no need to go from 18 to 0! Just stop at 18 (first request)
    - no need to restart the head position to the last one (100)! Just reset it to 78
  - Total distance goes from **190** down to **110** (saving 18-0/0-18 and 78-100/100-78 movements)

C-SCAN: Considerations

- C-LOOK: similar optimization to LOOK
  - In the example:
    - no need to go from 18 to 0! Just stop at 18 (first request)
    - no need to restart the head position to the last one (100)! Just reset it to 78
  - Total distance goes from **190** down to **110** (saving 18-0/0-18 and 78-100/100-78 movements)
- C-SCAN does not prioritize serving more recent requests

C-SCAN: Considerations

- C-LOOK: similar optimization to LOOK
  - In the example:
    - no need to go from 18 to 0! Just stop at 18 (first request)
    - no need to restart the head position to the last one (100)! Just reset it to 78
  - Total distance goes from **190** down to **110** (saving 18-0/0-18 and 78-100/100-78 movements)
- C-SCAN does not prioritize serving more recent requests
- Avoid start/stop of mechanical head movements

Where Are Those Algorithms Implemented?

- Disk scheduling algorithms are typically implemented on the disk controller itself
- Disk drives are shipped with one of these algorithms ready
- More complex scheduling algorithms can be designed (e.g., optimizing the overall access time)
- Complex logic should be instead moved to the disk driver (OS kernel)

Improving Disk Performance: Interleaving

Problem

Contiguous allocation of files on disk blocks only makes sense if the OS can react to one disk request and issue the next one before the disk spins over the next block

Improving Disk Performance: Interleaving

Problem

Contiguous allocation of files on disk blocks only makes sense if the OS can react to one disk request and issue the next one before the disk spins over the next block

Idea: Interleaving

Do not allocate blocks contiguously, but just leave temporarily contiguous blocks few blocks away from each other (e.g., 2 or 3) so as to accommodate rotational speed

Improving Disk Performance: Interleaving

Problem

Contiguous allocation of files on disk blocks only makes sense if the OS can react to one disk request and issue the next one before the disk spins over the next block

Idea: Interleaving

Do not allocate blocks contiguously, but just leave temporarily contiguous blocks few blocks away from each other (e.g., 2 or 3) so as to accommodate rotational speed

Filesystem-level optimization

Improving Disk Performance: Read-Ahead

Read-Ahead

Read blocks from the disk ahead of process requests and store them on the buffer (cache) of the disk controller

Improving Disk Performance: Read-Ahead

Read-Ahead

Read blocks from the disk ahead of process requests and store them on the buffer (cache) of the disk controller

Goal

Reduce the number of seeks, as several blocks that are on the same track are read even if not explicitly requested (locality)

Improving Disk Performance: Read-Ahead

Read-Ahead

Read blocks from the disk ahead of process requests and store them on the buffer (cache) of the disk controller

Goal

Reduce the number of seeks, as several blocks that are on the same track are read even if not explicitly requested (locality)

Virtual Memory vs. Disk Block Pre-fetching

The order of disk accesses more predictable (more locality) than memory accesses

Improving Disk Performance: Read-Ahead

Read-Ahead

Read blocks from the disk ahead of process requests and store them on the buffer (cache) of the disk controller

Goal

Reduce the number of seeks, as several blocks that are on the same track are read even if not explicitly requested (locality)

Virtual Memory vs. Disk Block Pre-fetching

The order of disk accesses more predictable (more locality) than memory accesses

Pre-fetching can be done both at the Disk-level and Filesystem-level

Disk Management: Formatting

- Before a disk can be used, it has to be **low-level formatted**

Disk Management: Formatting

- Before a disk can be used, it has to be **low-level formatted**
- This means laying down all of the headers and trailers marking the beginning and ends of each sector

Disk Management: Formatting

- Before a disk can be used, it has to be **low-level formatted**
- This means laying down all of the headers and trailers marking the beginning and ends of each sector
- Headers and trailers contain the linear sector numbers and **error-correcting codes** (**ECC**), for detection/fixing purposes

Disk Management: Formatting

- Before a disk can be used, it has to be **low-level formatted**
- This means laying down all of the headers and trailers marking the beginning and ends of each sector
- Headers and trailers contain the linear sector numbers and **error-correcting codes** (**ECC**), for detection/fixing purposes
- ECC is done with every disk read/write, and if damage is detected and recoverable, the disk controller handles itself a **soft error**

Disk Management: Formatting

- Once the disk is formatted, the next step is to partition the drive into one or more separate partitions

Disk Management: Formatting

- Once the disk is formatted, the next step is to partition the drive into one or more separate partitions
- Must be done even if the disk is used as a single large partition, so that the partition table is written at the beginning of the disk

Disk Management: Formatting

- Once the disk is formatted, the next step is to partition the drive into one or more separate partitions
- Must be done even if the disk is used as a single large partition, so that the partition table is written at the beginning of the disk
- After partitioning, then the filesystems must be **logically formatted** (more on this later)

Disk Management: Boot Block

Computer ROM contains a **bootstrap** program (OS independent) with just enough code to find the first sector on the first hard drive on the first controller

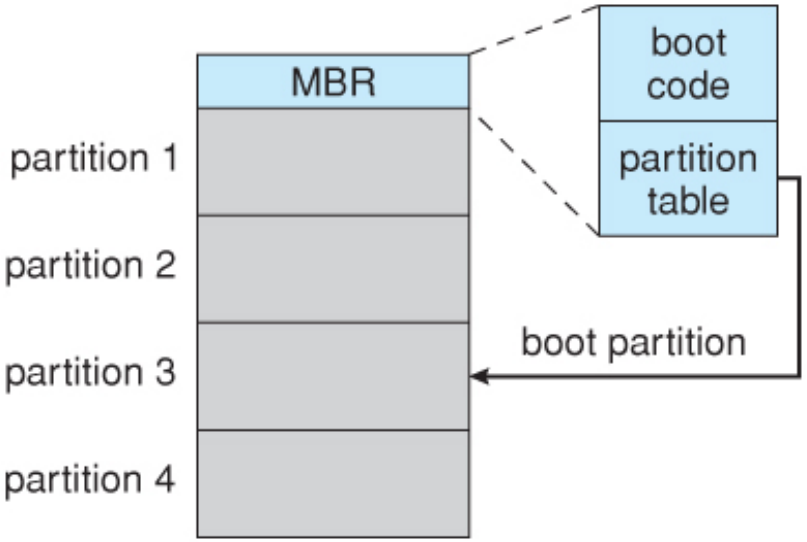
Disk Management: Boot Block

Computer ROM contains a **bootstrap** program (OS independent) with just enough code to find the first sector on the first hard drive on the first controller

This loads that sector into memory, and transfer control over to it

Disk Management: Boot Block

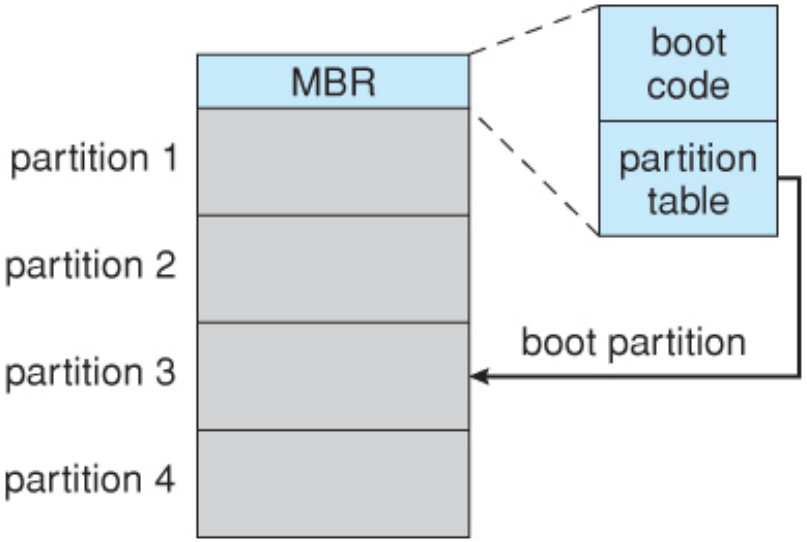
Computer ROM contains a **bootstrap** program (OS independent) with just enough code to find the first sector on the first hard drive on the first controller



This loads that sector into memory, and transfer control over to it

The first sector is known as the **Master Boot Record** (**MBR**), and contains a very small amount of code and the **partition table**

Disk Management: Boot Block



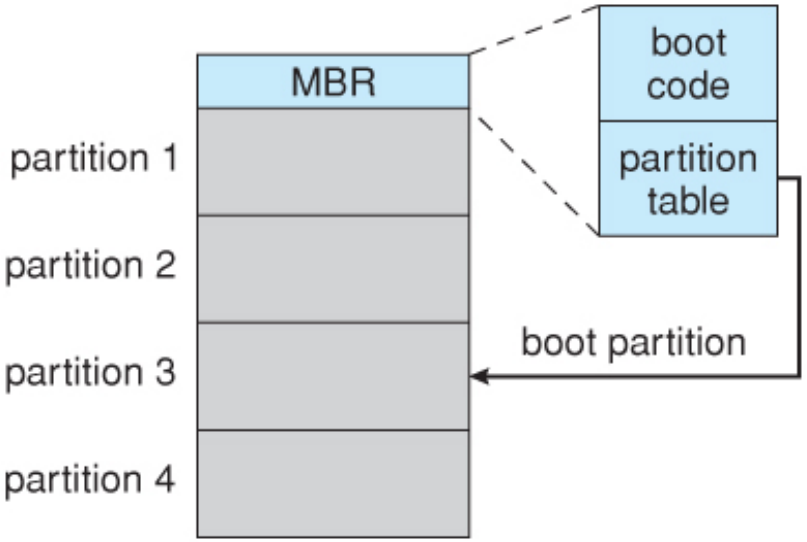
Computer ROM contains a **bootstrap** program (OS independent) with just enough code to find the first sector on the first hard drive on the first controller

This loads that sector into memory, and transfer control over to it

The first sector is known as the **Master Boot Record** (**MBR**), and contains a very small amount of code and the **partition table**

The partition table tells how the disk is logically partitioned, and indicates which partition is the **active** or **boot**(**able**) one

Disk Management: Boot Block



Computer ROM contains a **bootstrap** program (OS independent) with just enough code to find the first sector on the first hard drive on the first controller

This loads that sector into memory, and transfer control over to it

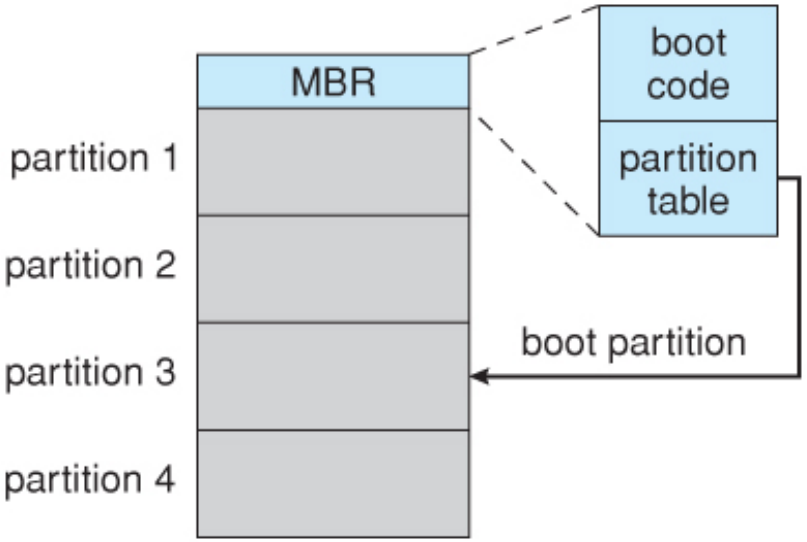
The first sector is known as the **Master Boot Record** (**MBR**), and contains a very small amount of code and the **partition table**

The partition table tells how the disk is logically partitioned, and indicates which partition is the **active** or **boot**(**able**) one

The boot program then looks to the active partition to find an operating system

Disk Management: Boot Block

Computer ROM contains a **bootstrap** program (OS independent) with just enough code to find the first sector on the first hard drive on the first controller



This loads that sector into memory, and transfer control over to it

The first sector is known as the **Master Boot Record** (**MBR**), and contains a very small amount of code and the **partition table**

The partition table tells how the disk is logically partitioned, and indicates which partition is the **active** or **boot**(**able**) one

The boot program then looks to the active partition to find an operating system

Once the kernel is found, it is loaded into memory and then control is transferred over to the OS, which initializes all important kernel data structures and system services

Overview of Mass-Storage Structure

3 categories of mass-storage devices

Magnetic Disk

Solid-State Disk

Magnetic Tapes

Solid-State Disks (SSDs): Overview

- SSDs use memory technology as a small fast hard disk
- Specific implementations may use either flash memory or DRAM chips protected by a battery
- SSDs have no moving parts so they are much faster than traditional hard drives
- Access blocks directly by referencing block number

Solid-State Disks (SSDs): Overview

- SSDs use memory technology as a small fast hard disk
- Specific implementations may use either flash memory or DRAM chips protected by a battery
- SSDs have no moving parts so they are much faster than traditional hard drives
- Access blocks directly by referencing block number

No need for disk scheduling

Solid-State Disks: Overview

- Read operations are very fast
- Write operations are slower as they need a slower erease cycle (cannot overwrite directly)
- Unreferenced blocks instead of overwriting (garbage collection)
- Limited number of writes per block (over lifetime)
- SSD controller needs to count how many times a block gets overwritten, so as to keep this balanced

Solid-State Disks: Overview

- SSDs are more expensive than hard drives, generally not as large, and may have shorter life spans
- SSDs are especially useful as a high-speed cache of hard-disk information that must be accessed quickly
- For example, they can use to store:
  - File system meta-data, e.g., directory and inode information (more on this later)
  - The OS bootloader and some application executables, but no vital user data

Solid-State Disks: Overview

- SSDs are also used in laptops to make them smaller, faster, and lighter
- Since SSDs are so much faster than traditional hard disks, the throughput of the I/O bus can become a limiting factor
- Some SSDs are therefore connected directly to the system PCI bus

Overview of Mass-Storage Structure

3 categories of mass-storage devices

Magnetic Disk

Solid-State Disk

Magnetic Tapes

Magnetic Tapes

- Primarily used for backups
- Accessing a particular spot on a magnetic tape can be slow
- No random/direct access, only sequential!
- After reading or writing starts, access speeds are comparable to disk drives
- Capacities of tape drives can range from 20 to 200 GB
- Today replaced by disks

Advanced Topic: RAID Structure

- **RAID → Redundant Array of Inexpensive Disks**

Advanced Topic: RAID Structure

- **RAID → Redundant Array of Inexpensive Disks**
- Use a group of cheap hard drives together with some form of duplication instead of one or two larger and more expensive

Advanced Topic: RAID Structure

- **RAID → Redundant Array of Inexpensive Disks**
- Use a group of cheap hard drives together with some form of duplication instead of one or two larger and more expensive
- **Goal:** increase reliability or speed up operations, or both

Advanced Topic: RAID Structure

- **RAID → Redundant Array of Inexpensive Disks**
- Use a group of cheap hard drives together with some form of duplication instead of one or two larger and more expensive
- **Goal:** increase reliability or speed up operations, or both
- Today RAID systems employ large possibly expensive disks as their components, switching the definition to **Independent** disks

Disk Failure

- Real-world systems may require **multiple disks**
  - E.g., Think about Google or Facebook, just to name a few

Disk Failure

- Real-world systems may require **multiple disks**
  - E.g., Think about Google or Facebook, just to name a few
- The more disks a system has, the greater the likelihood that one of them will go bad at any given time

Disk Failure

- Real-world systems may require **multiple disks**
  - E.g., Think about Google or Facebook, just to name a few
- The more disks a system has, the greater the likelihood that one of them will go bad at any given time
- Hence, increasing disks on a system actually decreases the **Mean Time To Failure** (**MTTF**) of the system

Example: Disk Failure

- Suppose we have a system with *N* disks

Example: Disk Failure

- Suppose we have a system with *N* disks
- Each disk has a MTTF = 10 years **~4,000 days**

Example: Disk Failure

- Suppose we have a system with N disks
- Each disk has a MTTF = 10 years **~4,000 days**

$$p = P(\text{disk}_i \text{ fails}) = 1/4{,}000 = 0.00025$$

Example: Disk Failure

- Suppose we have a system with N disks
- Each disk has a MTTF = 10 years **~4,000 days**

$$p = P(\text{disk}_i \text{ fails}) = 1/4{,}000 = 0.00025$$

- Associate with each disk a random variable $X_{i,t}$

Example: Disk Failure

- Suppose we have a system with N disks
- Each disk has a MTTF = 10 years **~4,000 days**

$$p = P(\text{disk}_i \text{ fails}) = 1/4{,}000 = 0.00025$$

- Associate with each disk a random variable $X_{i,t}$
  - $X_{i,t} \sim Bernoulli(p)$ outputs 1 (failure) with probability $p = 0.00025$ and 0 (working) with probability $(1-p) = 0.99975$

Example: Disk Failure

- Suppose we have a system with N disks
- Each disk has a MTTF = 10 years **~4,000 days**

$$p = P(\text{disk}_i \text{ fails}) = 1/4{,}000 = 0.00025$$

- Associate with each disk a random variable $X_{i,t}$
  - $X_{i,t} \sim Bernoulli(p)$ outputs 1 (failure) with probability $p = 0.00025$ and 0 (working) with probability $(1-p) = 0.99975$
  - Assume for semplicity p is the same for all disks and independent from each other

Example: Disk Failure

What is the **expected number of failures** in a certain day t,
given that the probability of **one** disk failing is p?"

Example: Disk Failure

What is the **expected number of failures** in a certain day t,
given that the probability of **one** disk failing is p?"

Under the (simplified) assumption that $X_{i,t}$ are all i.i.d.

Example: Disk Failure

What is the **expected number of failures** in a certain day t,
given that the probability of **one** disk failing is p?"

Under the (simplified) assumption that $X_{i,t}$ are all i.i.d.

$$T = X_{1,t} + X_{2,t} + \ldots + X_{N,t}$$

Example: Disk Failure

What is the **expected number of failures** in a certain day t,
given that the probability of **one** disk failing is p?"

Under the (simplified) assumption that $X_{i,t}$ are all i.i.d.

$$T = X_{1,t} + X_{2,t} + \ldots + X_{N,t}$$

$$T \sim Binomial\ (N, p)$$

Example: Disk Failure

What is the **expected number of failures** in a certain day t,
given that the probability of **one** disk failing is p?"

Under the (simplified) assumption that $X_{i,t}$ are all i.i.d.

$$T = X_{1,t} + X_{2,t} + \ldots + X_{N,t}$$

$$T \sim Binomial\ (N, p)$$

$$E[T] = Np$$

Example: Disk Failure

- A single-disk failure on a day is a quite a rare event (0.025% chance)

Example: Disk Failure

- A single-disk failure on a day is a quite a rare event (0.025% chance)
- Things are not so infrequent when we deal with several disks:
  - 1 (expected) failure per day with N = 4,000 disks
  - 100 (expected) failures per day with N = 400,000 disks

Improvement of Reliability via Redundancy

- **Mirroring** → copy the same data onto multiple disks

Improvement of Reliability via Redundancy

- **Mirroring** → copy the same data onto multiple disks
- Data won't be lost unless **all** copies were damaged simultaneously

Improvement of Reliability via Redundancy

- **Mirroring** → copy the same data onto multiple disks
- Data won't be lost unless **all** copies were damaged simultaneously
- This is a much lower probability than for a single disk to fail!

Improvement of Reliability via Redundancy

- **Mirroring** → copy the same data onto multiple disks
- Data won't be lost unless **all** copies were damaged simultaneously
- This is a much lower probability than for a single disk to fail!
  - Probability of *N* i.i.d. events, each having probability *p* to occur

Improvement of Reliability via Redundancy

- **Mirroring** → copy the same data onto multiple disks
- Data won't be lost unless **all** copies were damaged simultaneously
- This is a much lower probability than for a single disk to fail!
  - Probability of N i.i.d. events, each having probability p to occur

$$p^N$$

Improvement of Performance via Parallelism

- Mirroring also improves performance, particularly with respect to read operations
- Every block of data is duplicated on multiple disks, and read operations can be satisfied from any available copy
- Multiple disks can be reading different data blocks simultaneously in parallel
- Writes could also be speed up through careful scheduling algorithms, but it would be complicated in practice

Improvement of Performance via Parallelism

- Another way of improving disk access time is with **striping**
- This means spreading data out across multiple disks that can be accessed simultaneously
- Striped disks are logically seen as a single storage unit

RAID Levels

- Mirroring provides reliability but is expensive and possibly space wasting

RAID Levels

- Mirroring provides reliability but is expensive and possibly space wasting
- Striping improves performance, but does not improve reliability

RAID Levels

- Mirroring provides reliability but is expensive and possibly space wasting
- Striping improves performance, but does not improve reliability
- A number of different schemes combine try to balance between the 2

RAID Levels

- Mirroring provides reliability but is expensive and possibly space wasting
- Striping improves performance, but does not improve reliability
- A number of different schemes combine try to balance between the 2
  - RAID Level 0 → striping only, no mirroring

RAID Levels

- Mirroring provides reliability but is expensive and possibly space wasting
- Striping improves performance, but does not improve reliability
- A number of different schemes combine try to balance between the 2
  - RAID Level 0 → striping only, no mirroring
  - RAID Level 1 → mirroring only, no striping

RAID Levels

- Mirroring provides reliability but is expensive and possibly space wasting
- Striping improves performance, but does not improve reliability
- A number of different schemes combine try to balance between the 2
  - RAID Level 0 → striping only, no mirroring
  - RAID Level 1 → mirroring only, no striping
  - …
  - RAID Level 6 → striping + mirroring + parity bit

RAID Levels

(a) RAID 0: non-redundant striping.

(b) RAID 1: mirrored disks.

(c) RAID 2: memory-style error-correcting codes.

(d) RAID 3: bit-interleaved parity.

(e) RAID 4: block-interleaved parity.

(f) RAID 5: block-interleaved distributed parity.

(g) RAID 6: P + Q redundancy.

Summary

- Disks are slow devices compared to CPUs (and main memory)
- Manage those device efficiently is crucial
- Adding complexity to the OS in exchange for better I/O performance
- The same applies to other I/O devices