Sistemi Operativi I

Corso di Laurea in Informatica

2023-2024

Gabriele Tolomei

Dipartimento di Informatica

Sapienza Università di Roma

tolomei@di.uniroma1.it

Recap from Previous Lecture

- Mechanical components of magnetic disks cause bottleneck
- To minimize data transfer time from disk we need to minimize seek time and rotational delay
- HW optimizations have a limited impact on the performance gain
- OS can help here!

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

⇩

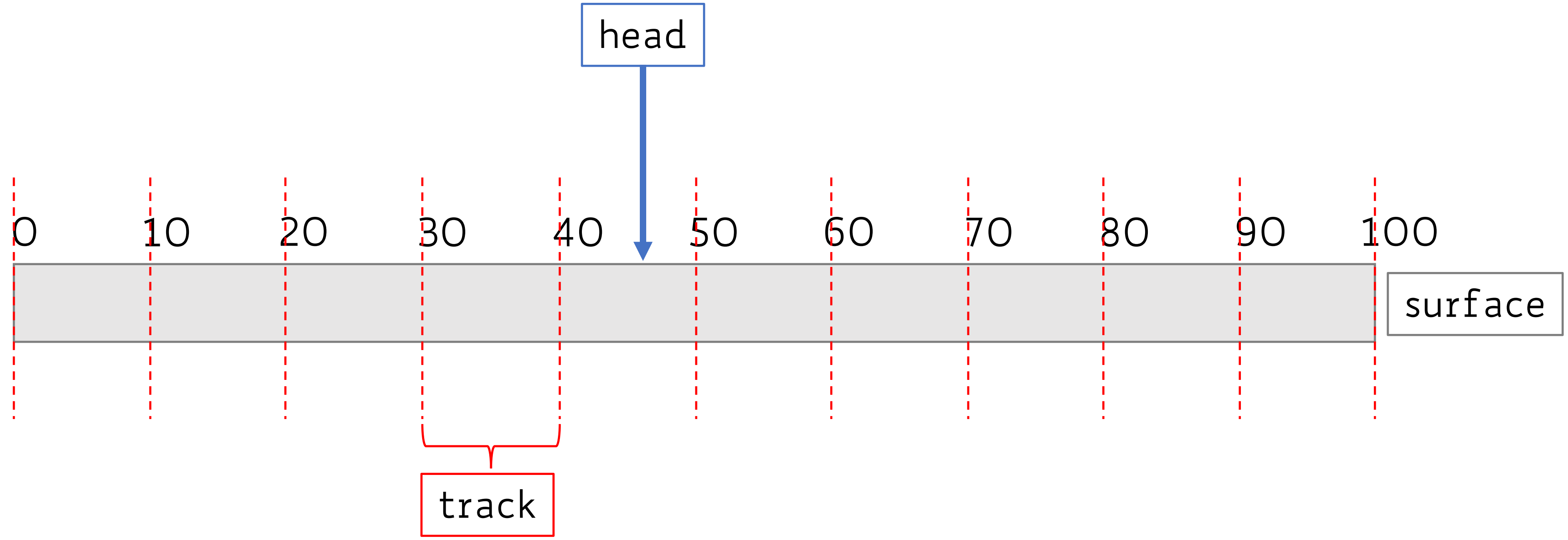
Idea

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

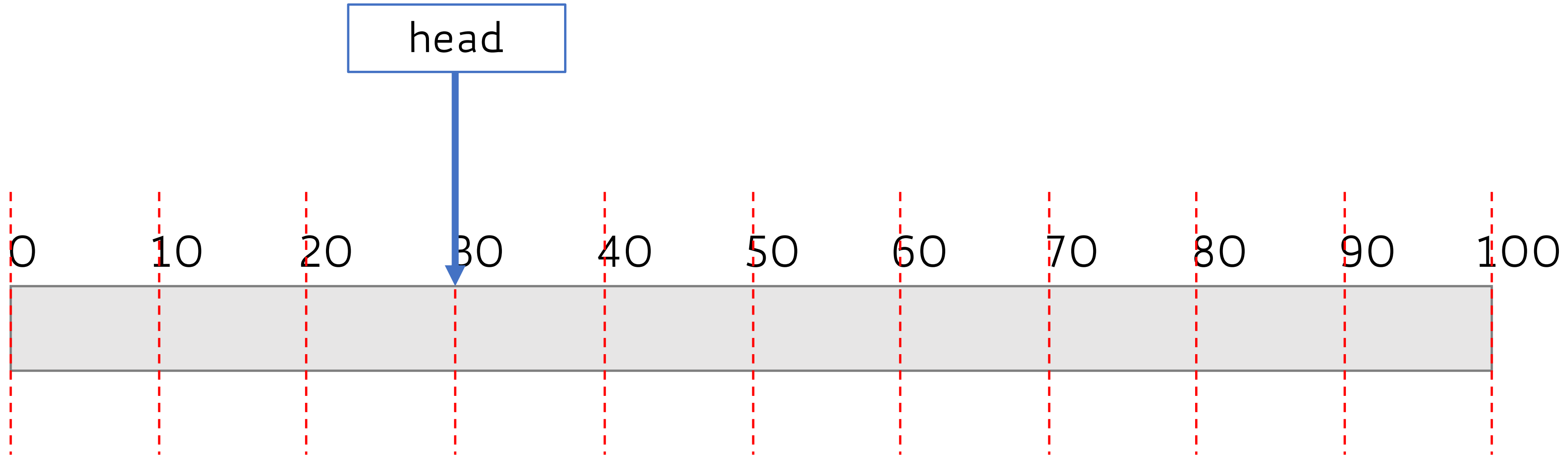
We consider track/cylinder independently of the sector

Disk Scheduling: Model



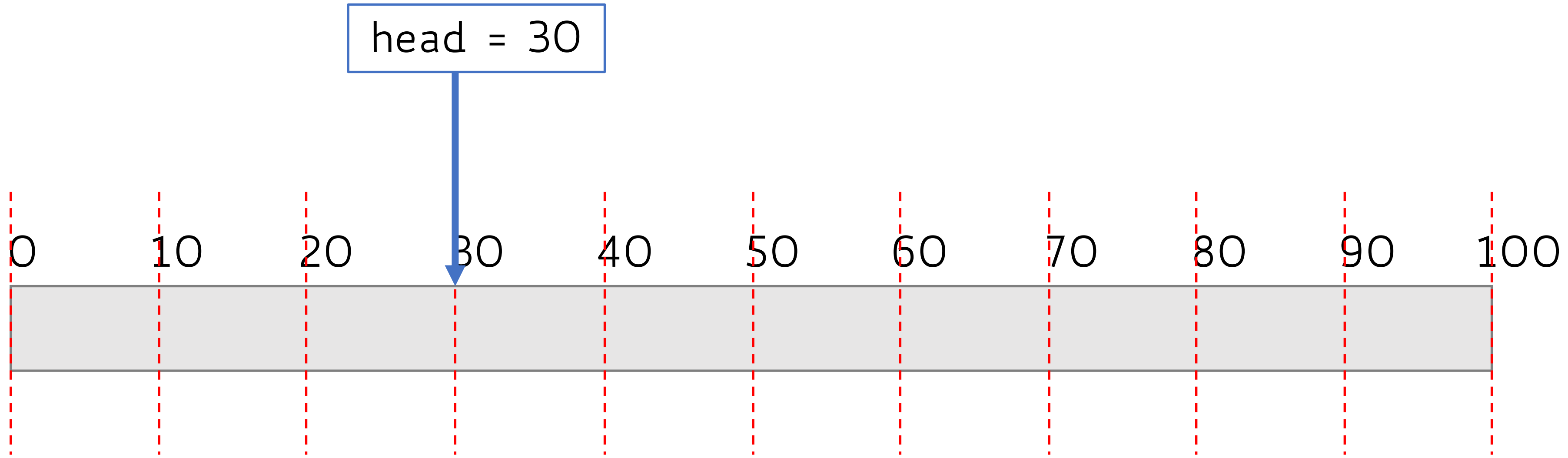
Disk Scheduling: First Come First Served (FCFS)

Serve the requests in the exact same order they arrive



Disk Scheduling: First Come First Served (FCFS)

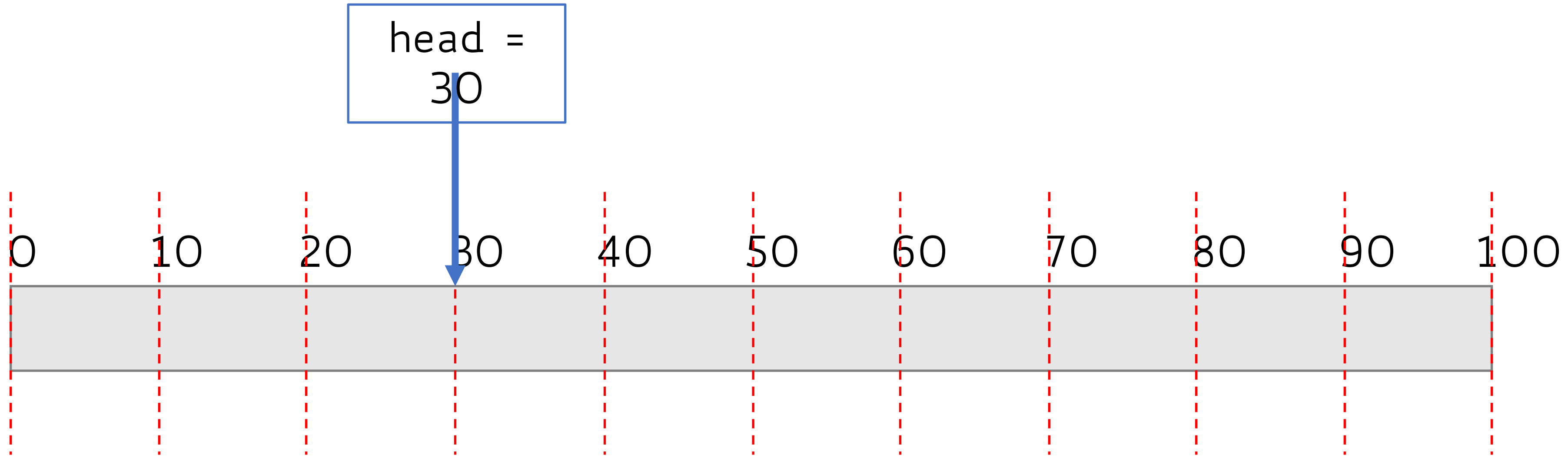
requests = 65, 40, 18, 78



Distance travelled = 0

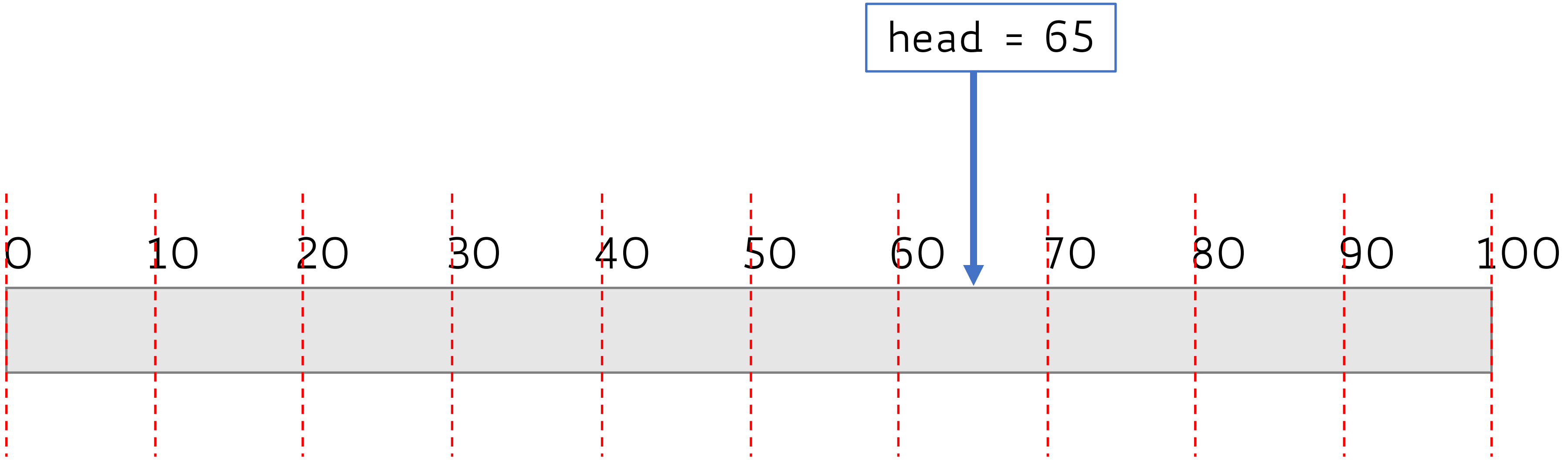
Disk Scheduling: First Come First Served (FCFS)

requests = $\boxed{65}$, 40, 18, 78



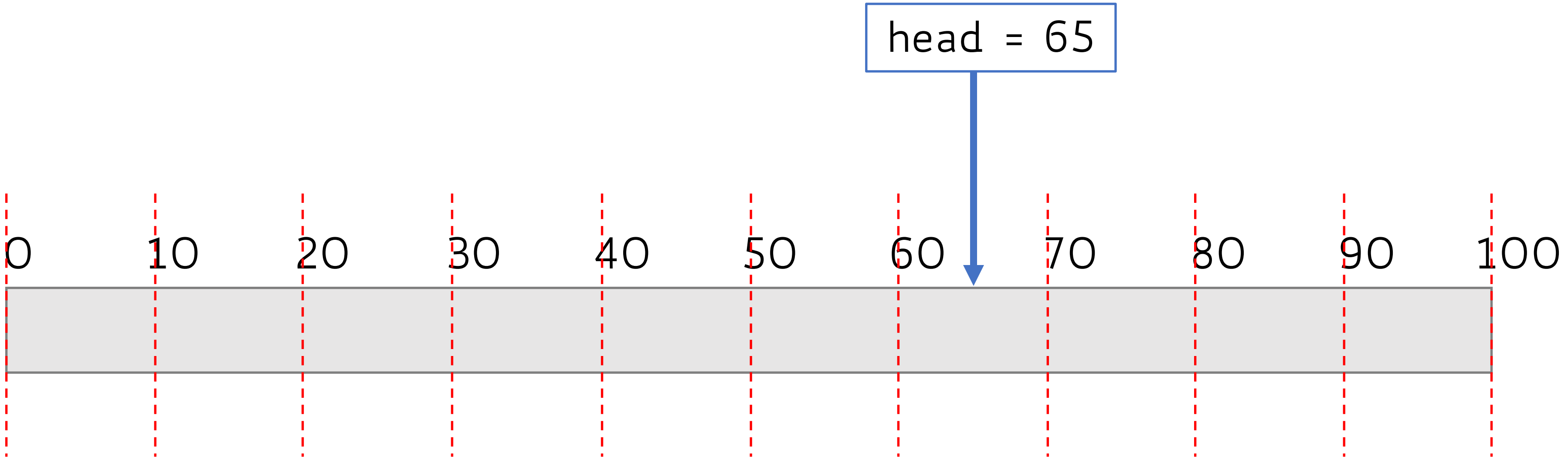
Disk Scheduling: First Come First Served (FCFS)

requests = 65, 40, 18, 78



Disk Scheduling: First Come First Served (FCFS)

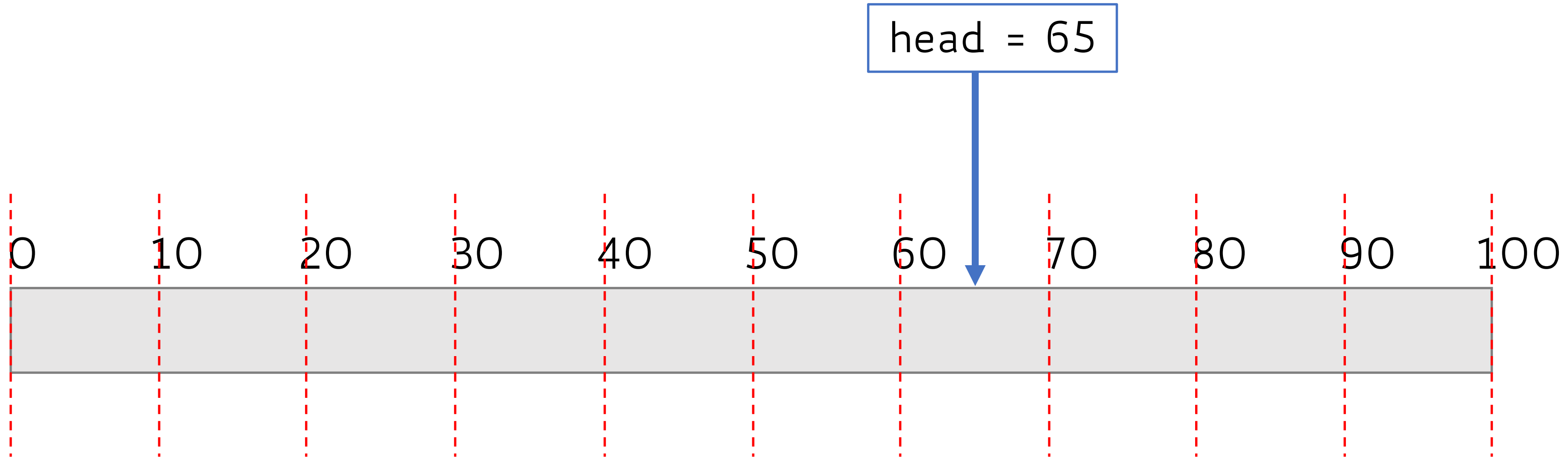
requests = 65, 40, 18, 78



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

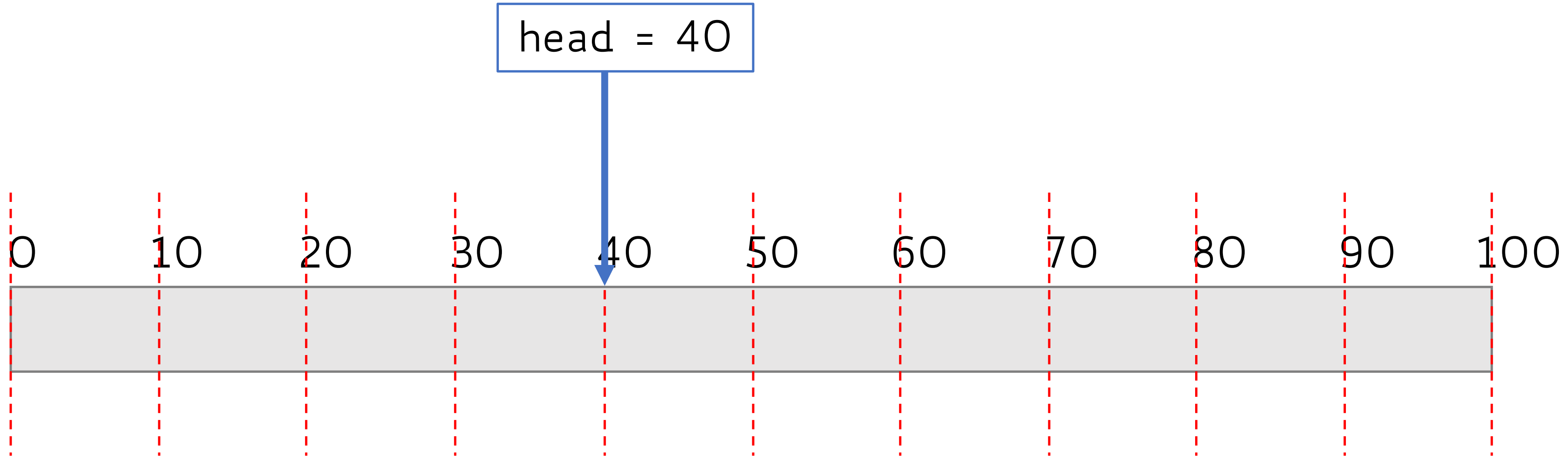
Disk Scheduling: First Come First Served (FCFS)

requests = 40, 18, 78



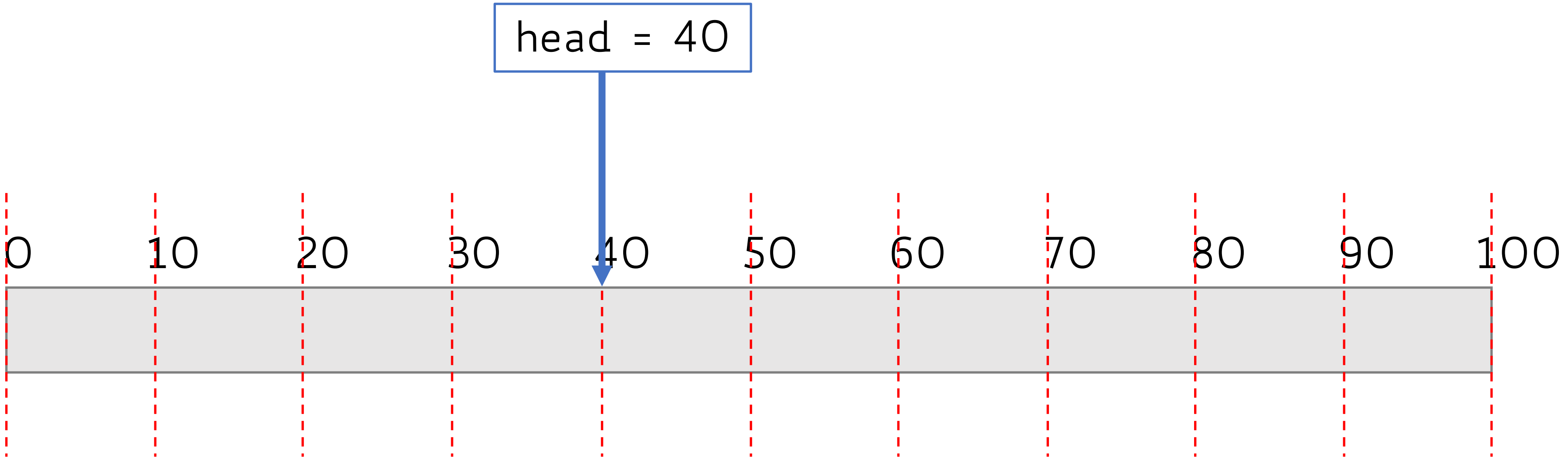
Disk Scheduling: First Come First Served (FCFS)

requests = 40, 18, 78



Disk Scheduling: First Come First Served (FCFS)

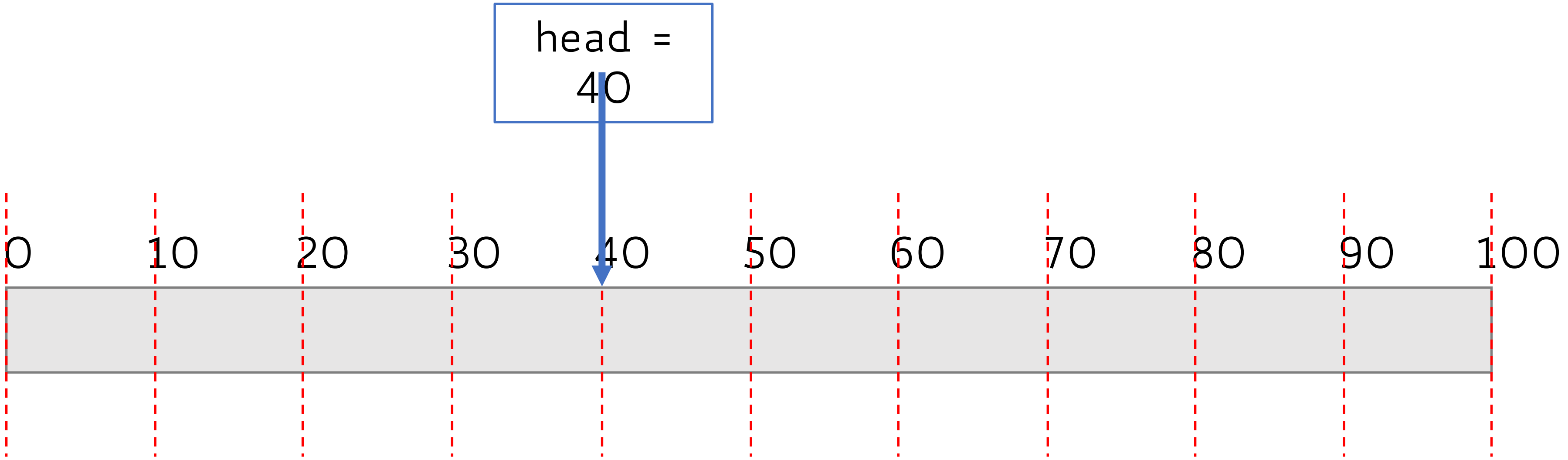
requests = 40, 18, 78



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

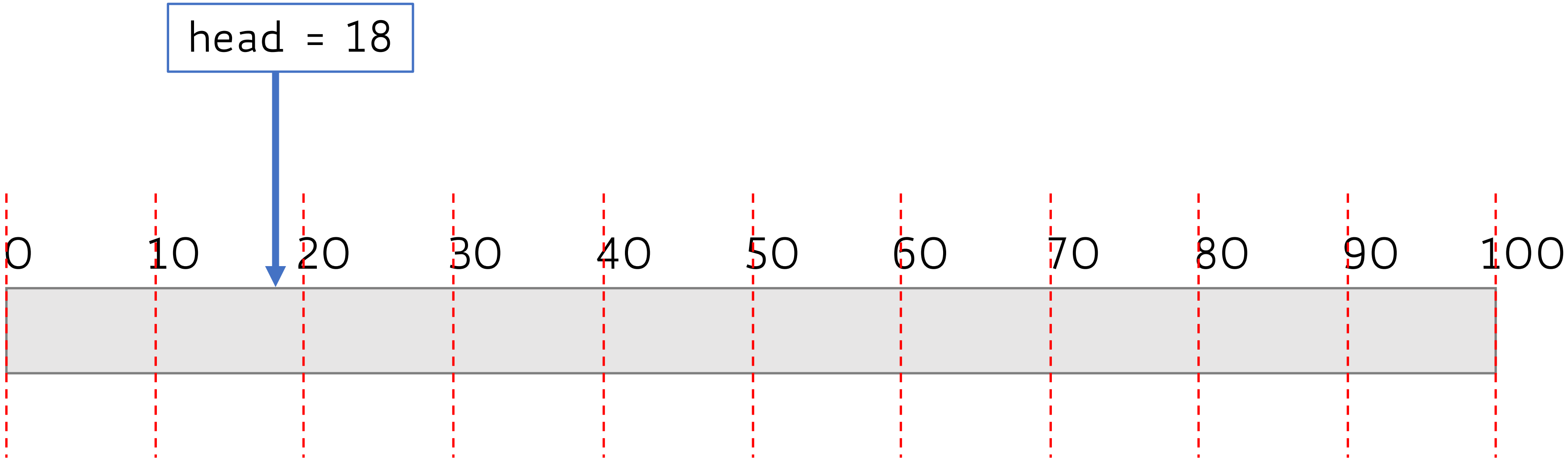
Disk Scheduling: First Come First Served (FCFS)

requests = 18 78



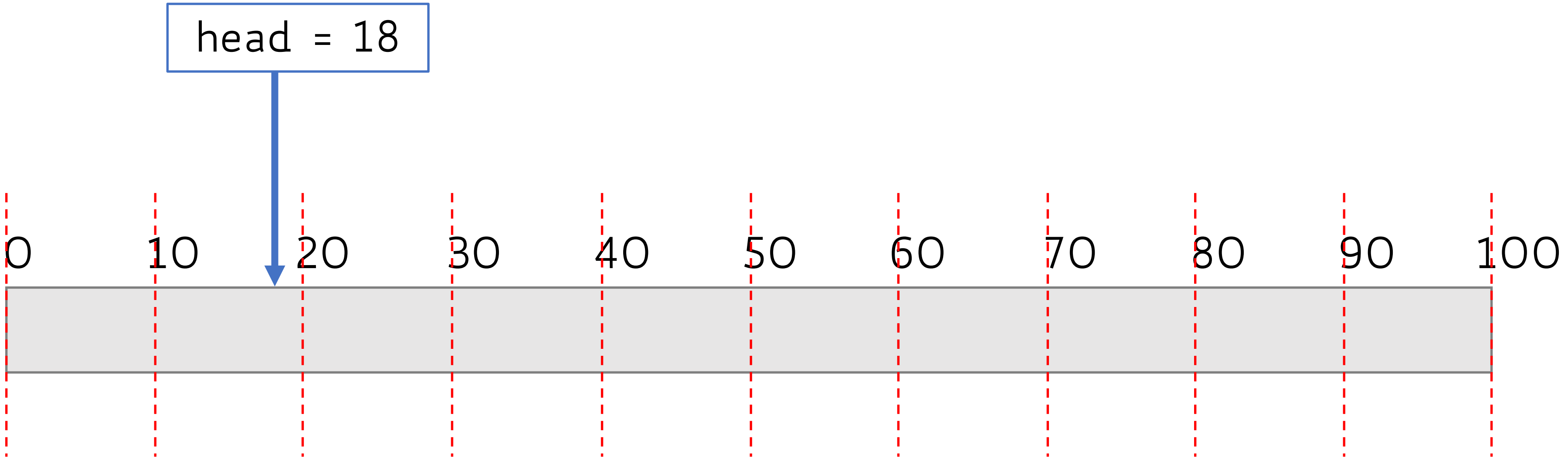
Disk Scheduling: First Come First Served (FCFS)

requests = 18, 78



Disk Scheduling: First Come First Served (FCFS)

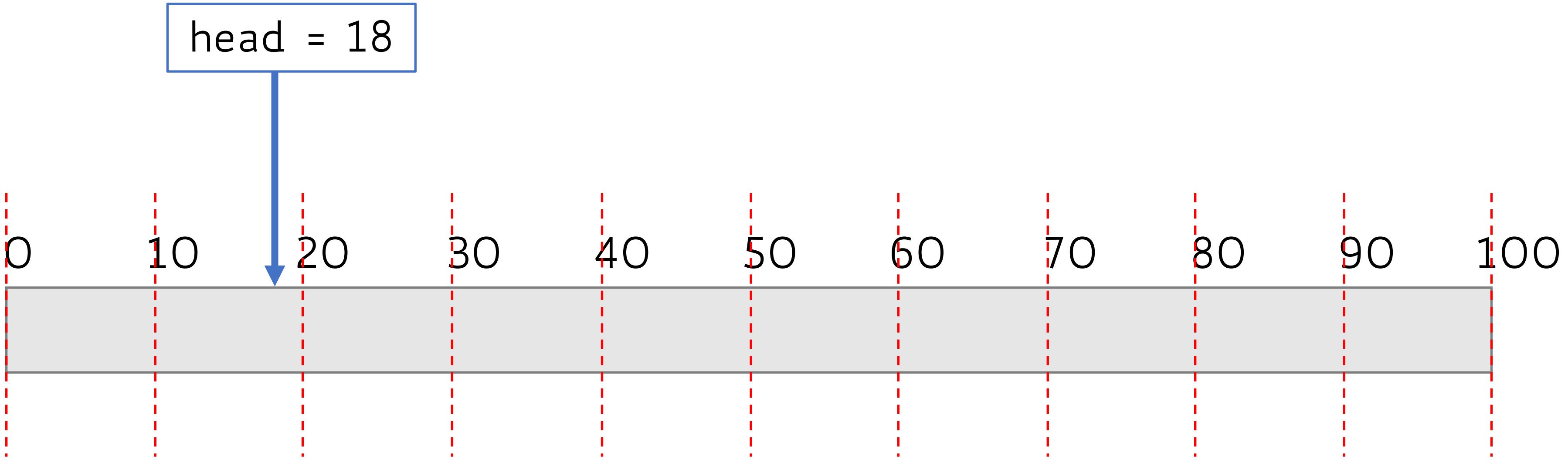
requests = 18, 78



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

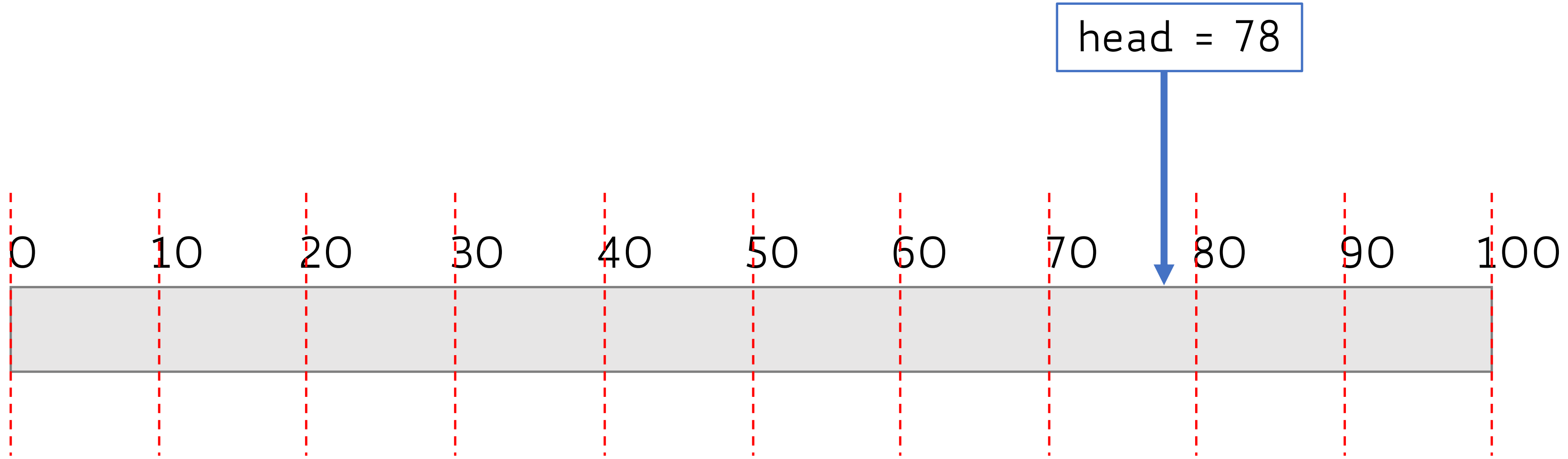
Disk Scheduling: First Come First Served (FCFS)

requests = $\boxed{78}$



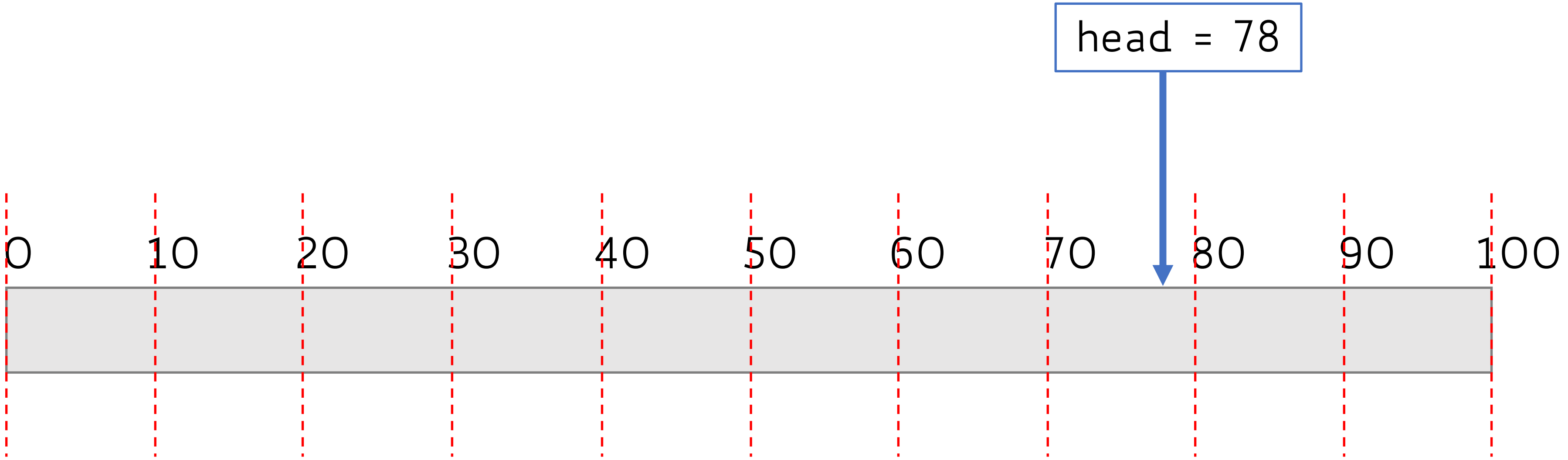
Disk Scheduling: First Come First Served (FCFS)

```
requests = 78
```



Disk Scheduling: First Come First Served (FCFS)

requests = 78



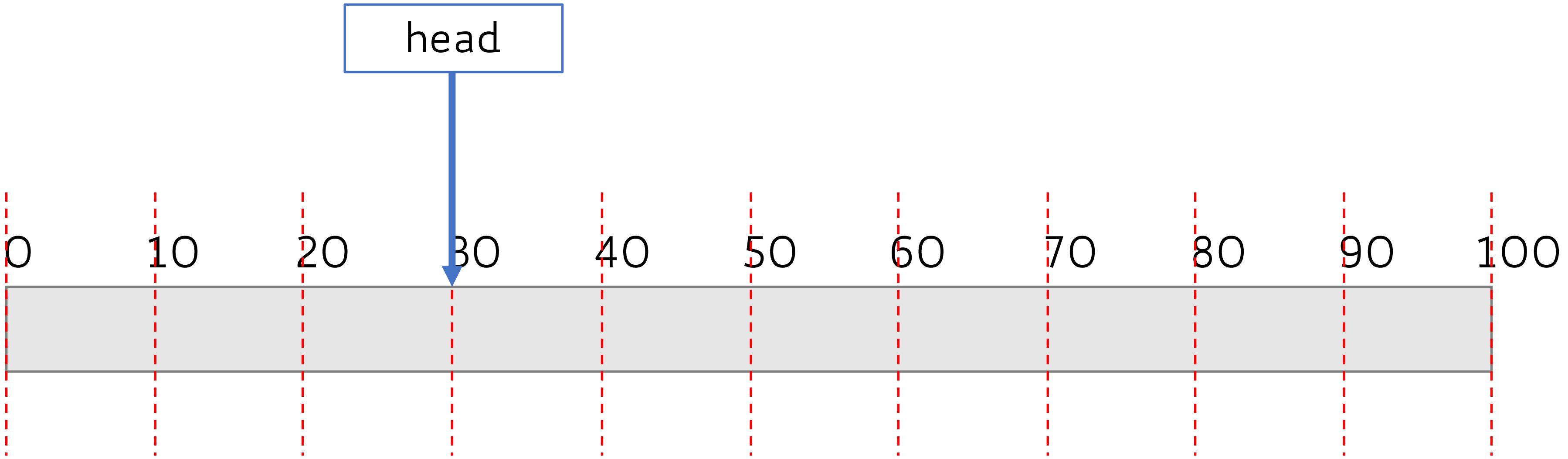
Distance travelled = 82 + |78 – 18| = 142

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

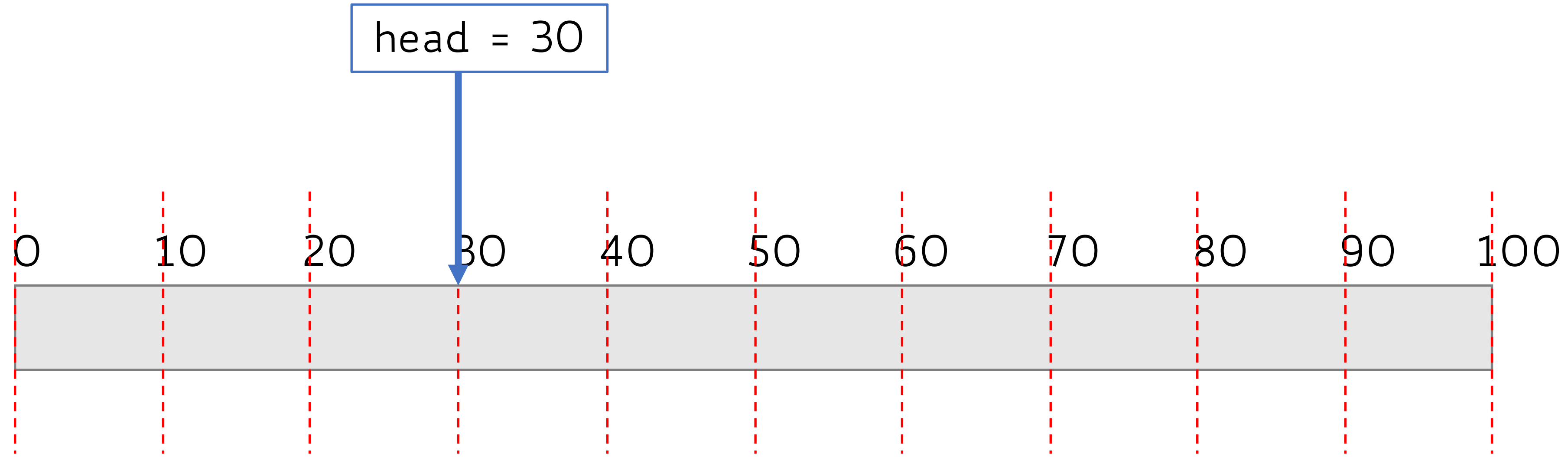
Disk Scheduling: Shortest Seek Time First (SSTF)

Greedily select the next closest request



Disk Scheduling: Shortest Seek Time First (SSTF)

requests = 65, 40, 18, 78

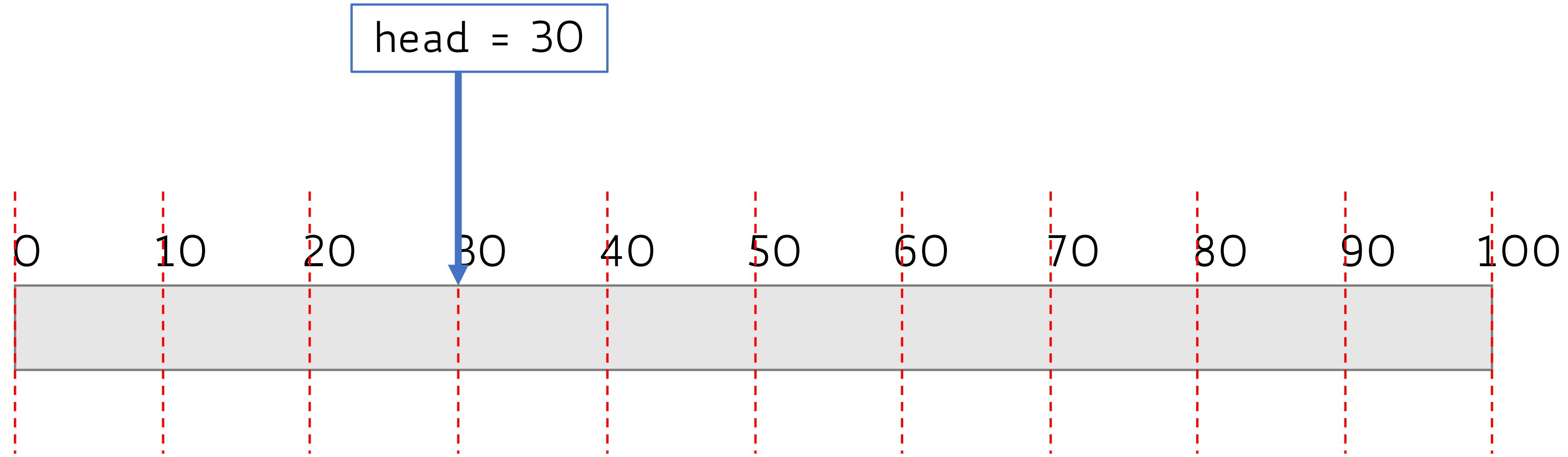


Distance travelled = 0

Disk Scheduling: Shortest Seek Time First (SSTF)

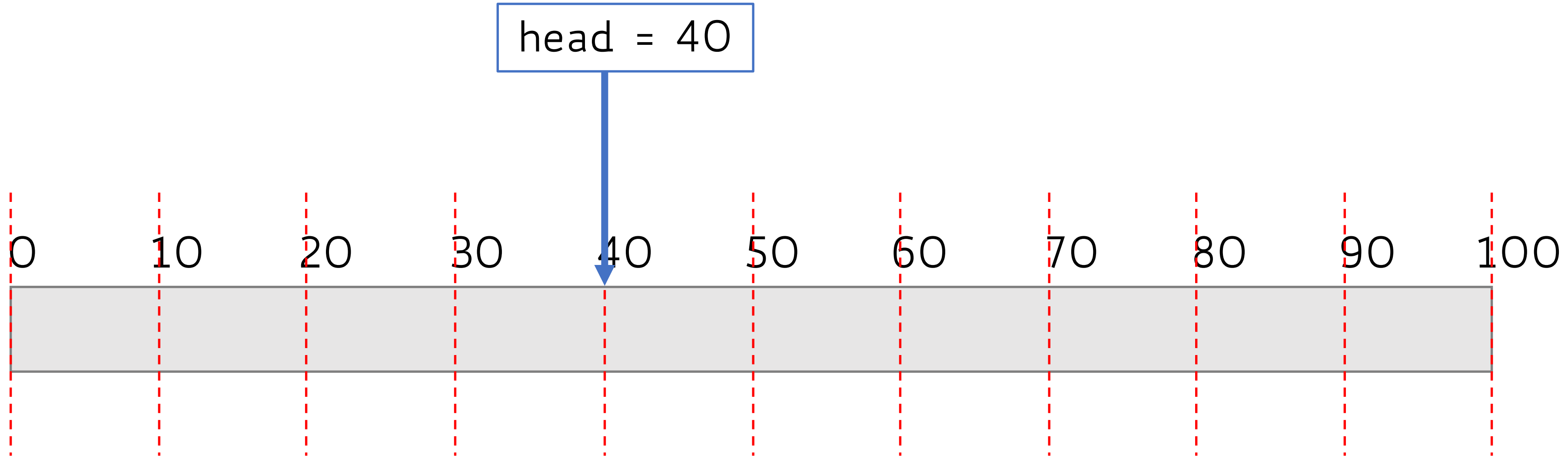
requests = 65, 40, 18, 78

Closest request = 40



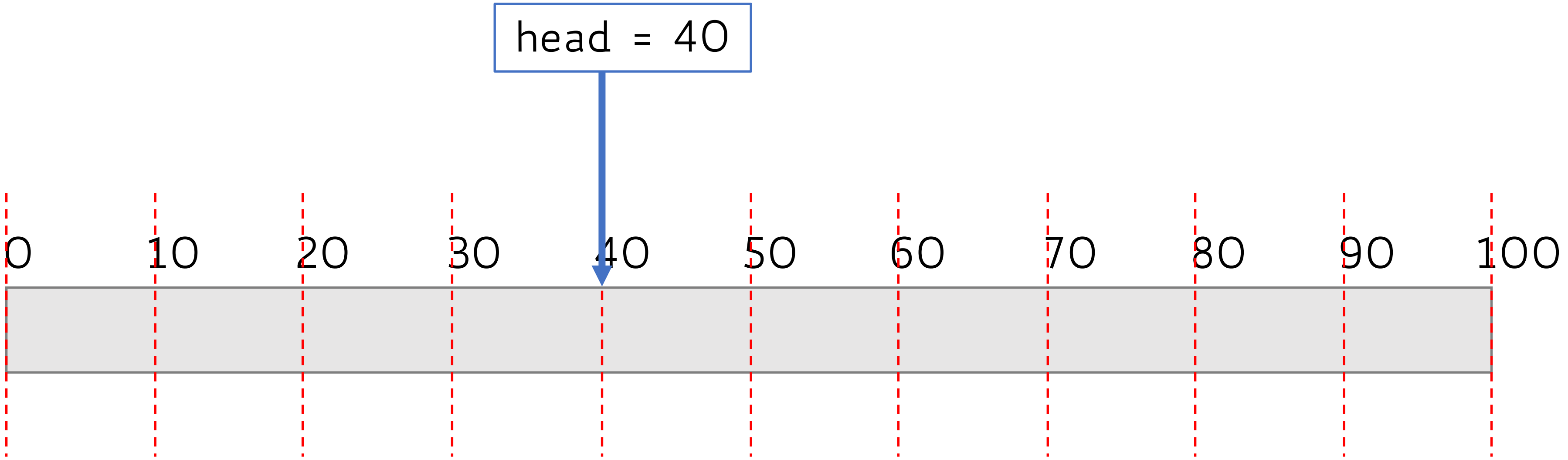
Disk Scheduling: Shortest Seek Time First (SSTF)

requests = 65, 40, 18, 78



Disk Scheduling: Shortest Seek Time First (SSTF)

requests = 65, 40, 18, 78

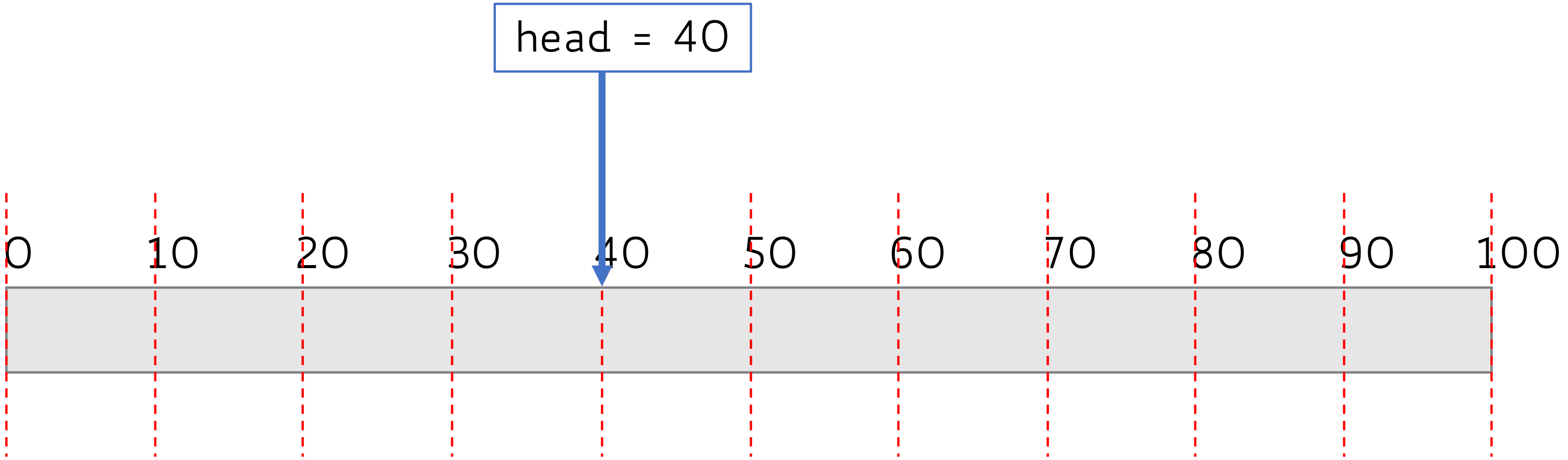


Distance travelled = 0 + |40 – 30| = 10

Disk Scheduling: Shortest Seek Time First (SSTF)

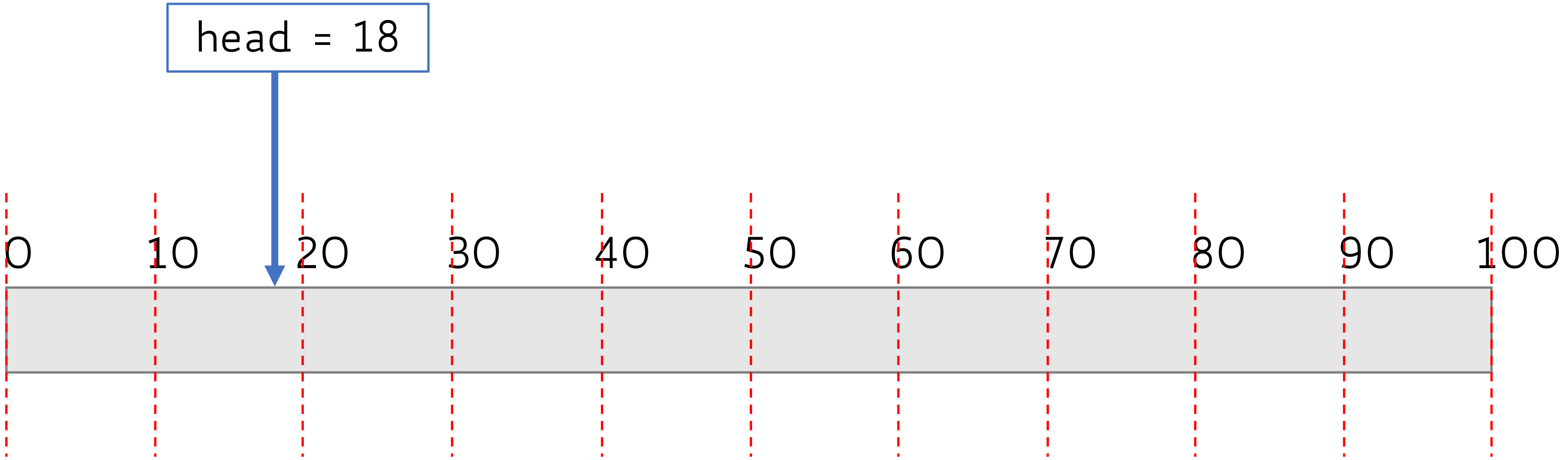
requests = 65, 18, 78

Closest request = 18



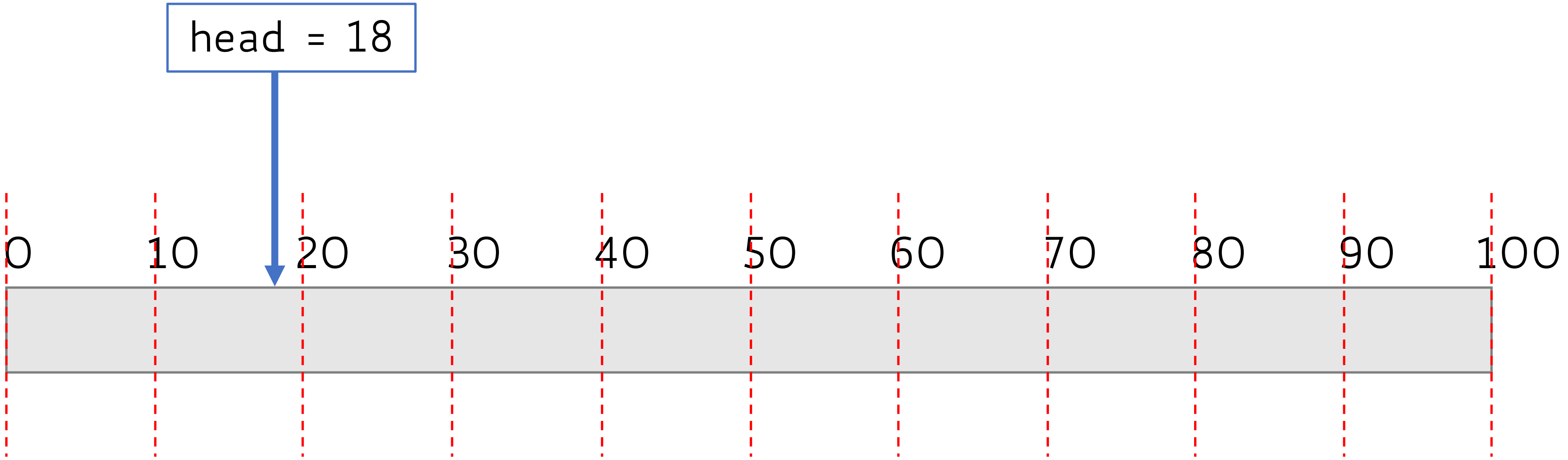
Disk Scheduling: Shortest Seek Time First (SSTF)

requests = 65, 18, 78



Disk Scheduling: Shortest Seek Time First (SSTF)

requests = 65, 18, 78

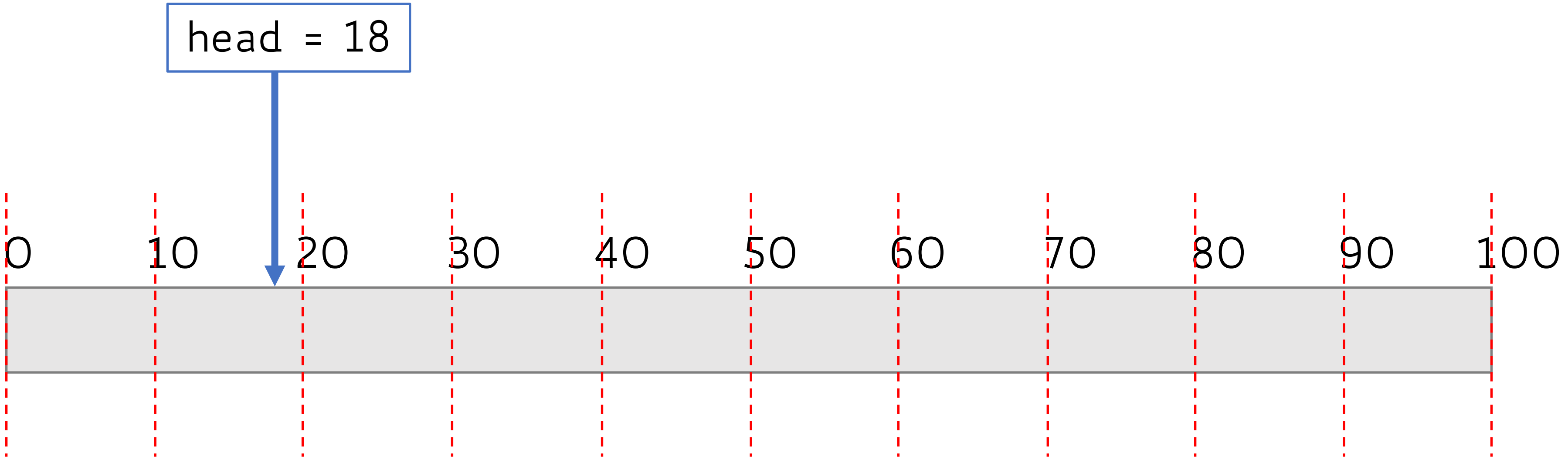


Distance travelled = 10 + |18 – 40| = 32

Disk Scheduling: Shortest Seek Time First (SSTF)

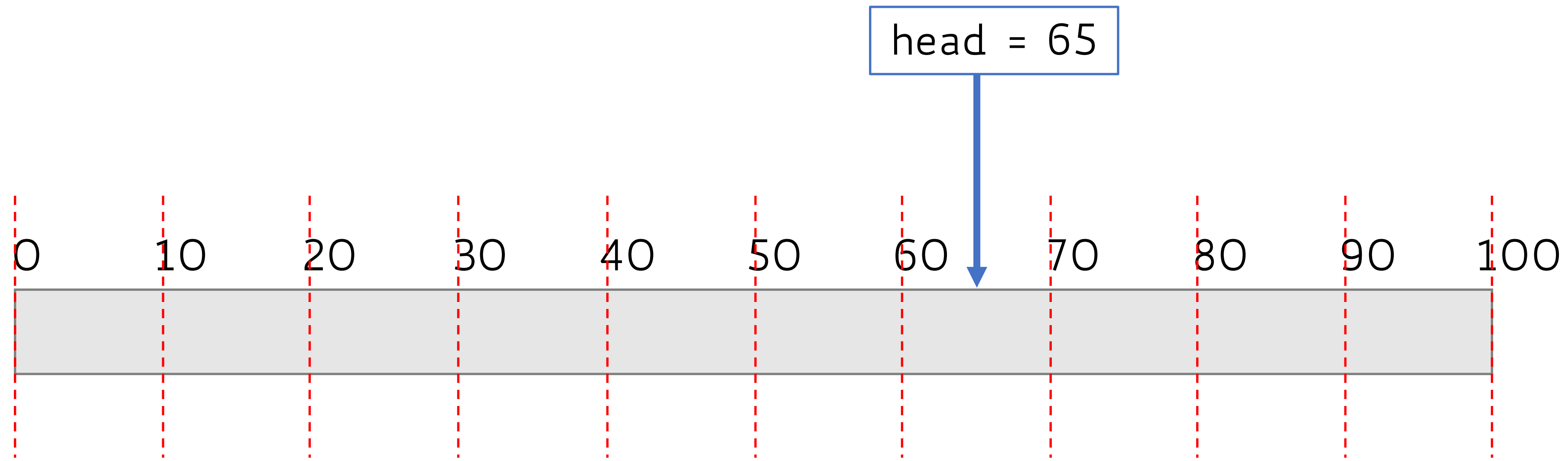
requests = 65, 78

Closest request = 65



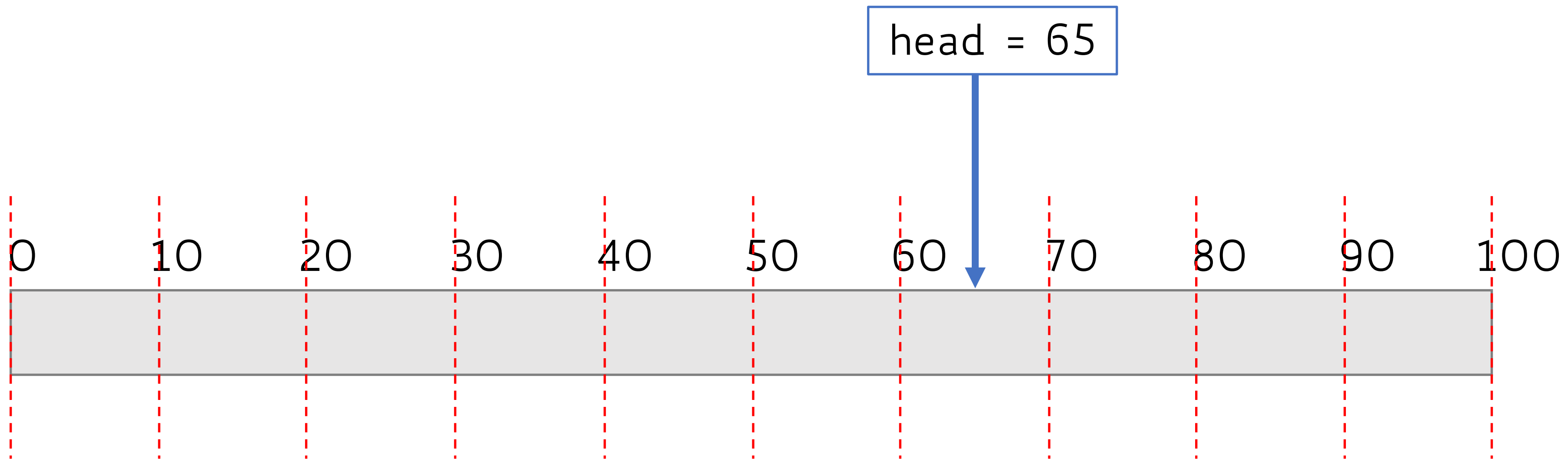
Disk Scheduling: Shortest Seek Time First (SSTF)

requests = 65, 78



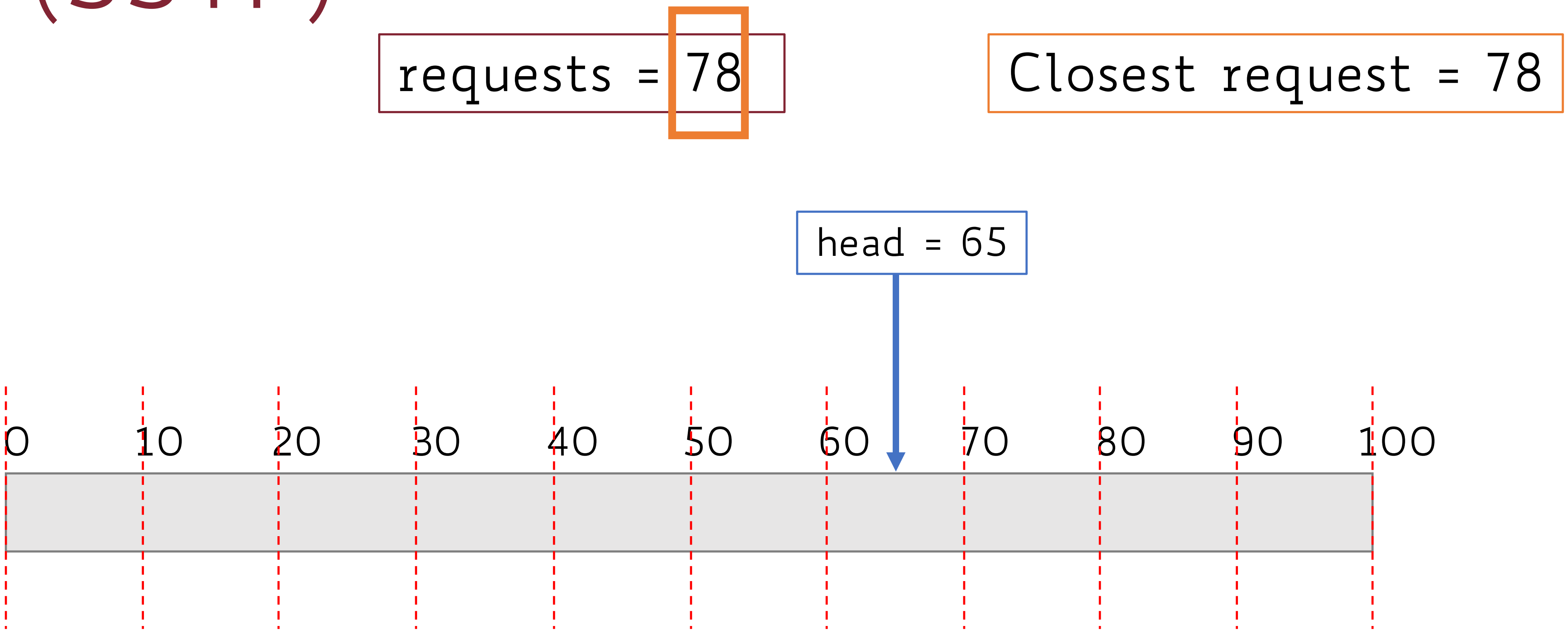
Disk Scheduling: Shortest Seek Time First (SSTF)

requests = 65, 78

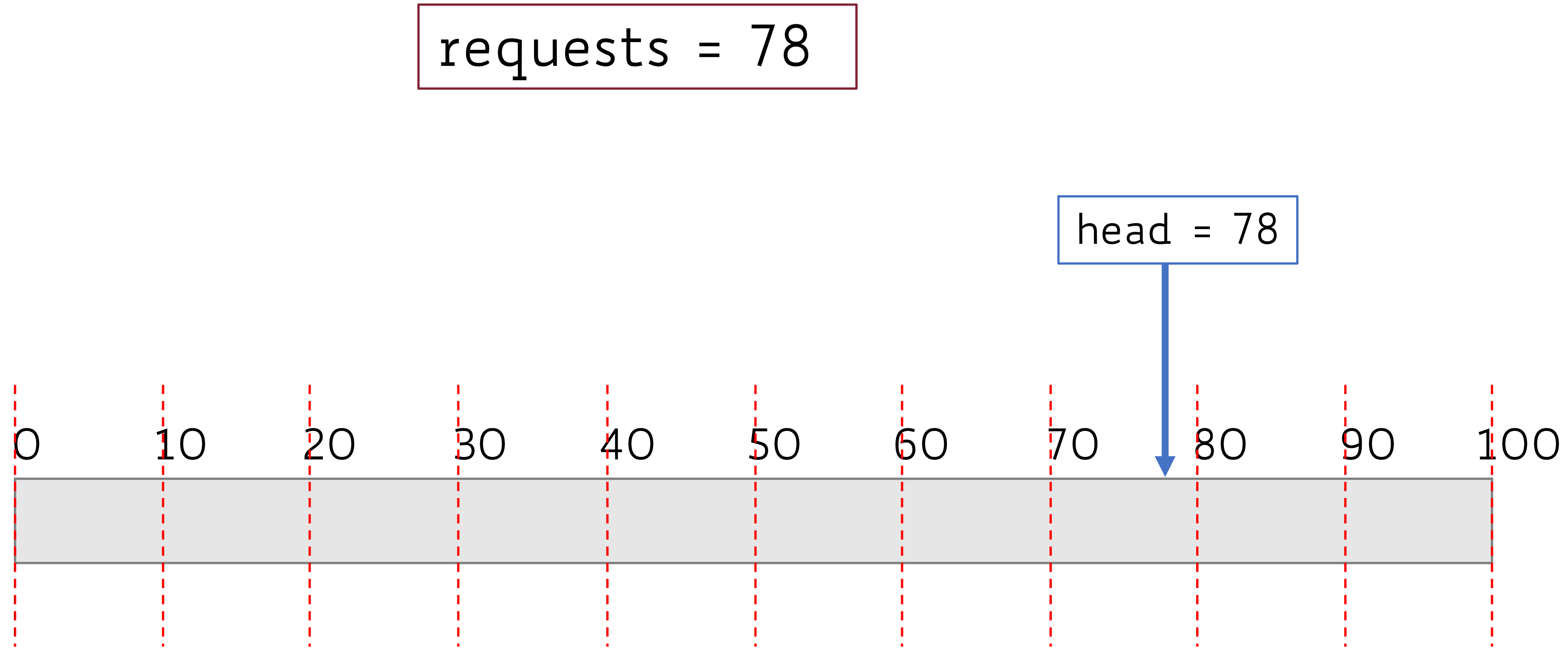


Distance travelled = 32 + |65 – 18| = 79

Disk Scheduling: Shortest Seek Time First (SSTF)

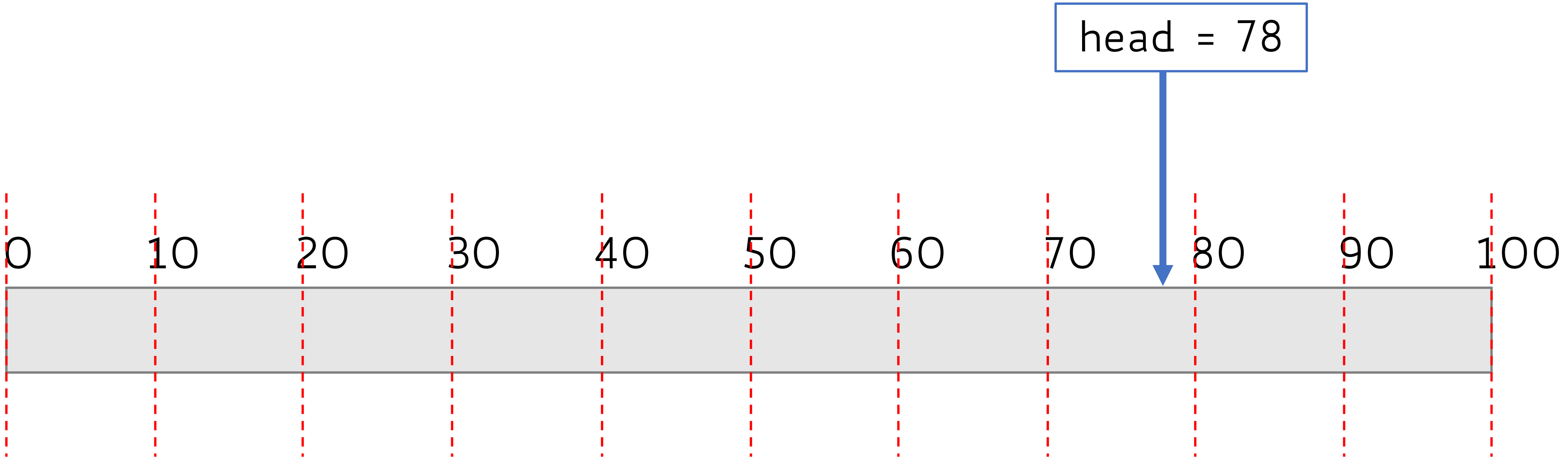


Disk Scheduling: Shortest Seek Time First (SSTF)



Disk Scheduling: Shortest Seek Time First (SSTF)

requests = 78



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

SSTF: Considerations

- Implemented by keeping a sorted list of requests

SSTF: Considerations

- Implemented by keeping a sorted list of requests
- Overhead is negligible w.r.t. the speed of disk

SSTF: Considerations

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

SSTF: Considerations

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

SSTF: Considerations

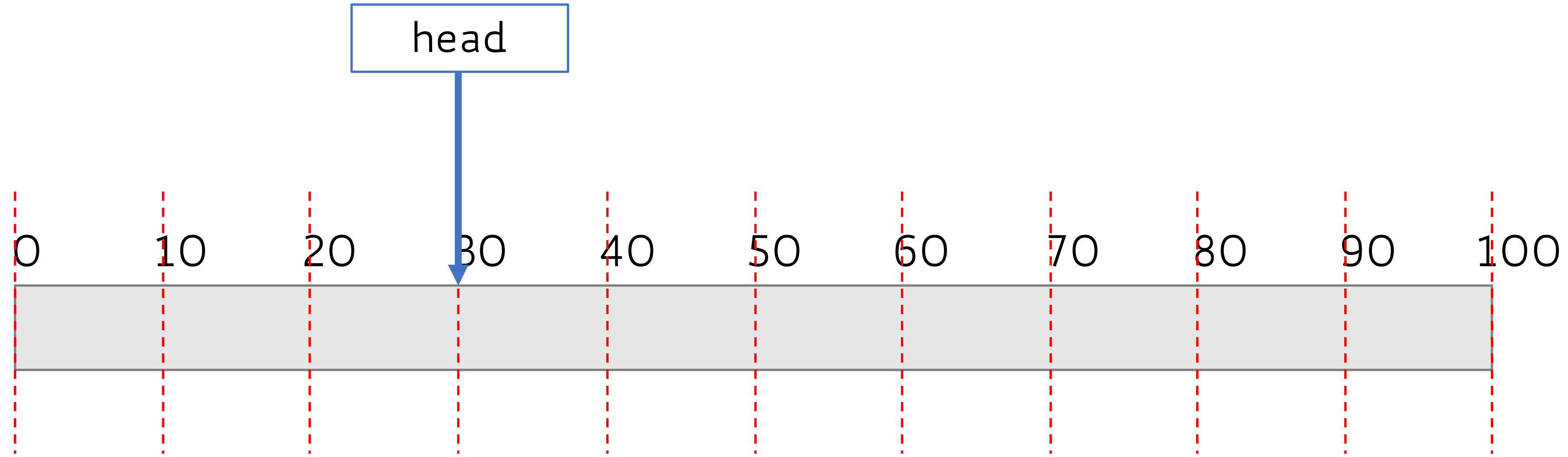
- Implemented by keeping a sorted list of requests
- Overhead is negligible w.r.t. the speed of disk
- It may cause **starvation** (requests too far apart from the head)
- It is **not** overall optimal as it greedily minimizes seek time (locally)
  - Example: head = 50; requests = 10, 20, 30, 40, 61

SSTF only looks one step ahead

Disk Scheduling: SCAN

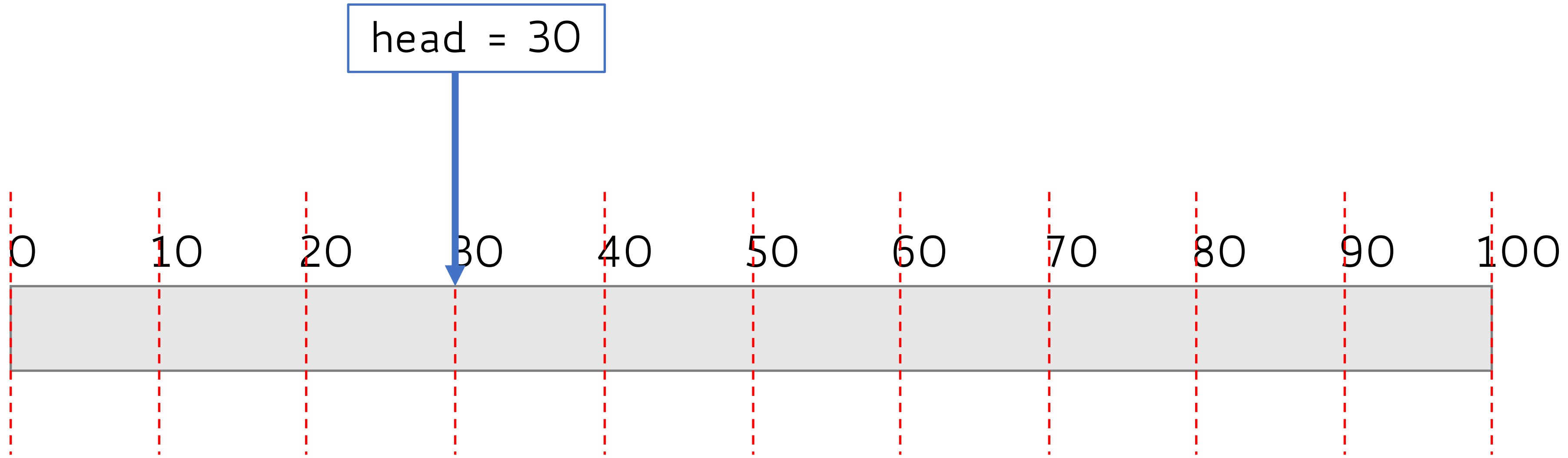
Head moves back and forth across the disk (e.g., track 0-100, 100-0)

Requests are served as the head passes (elevator)



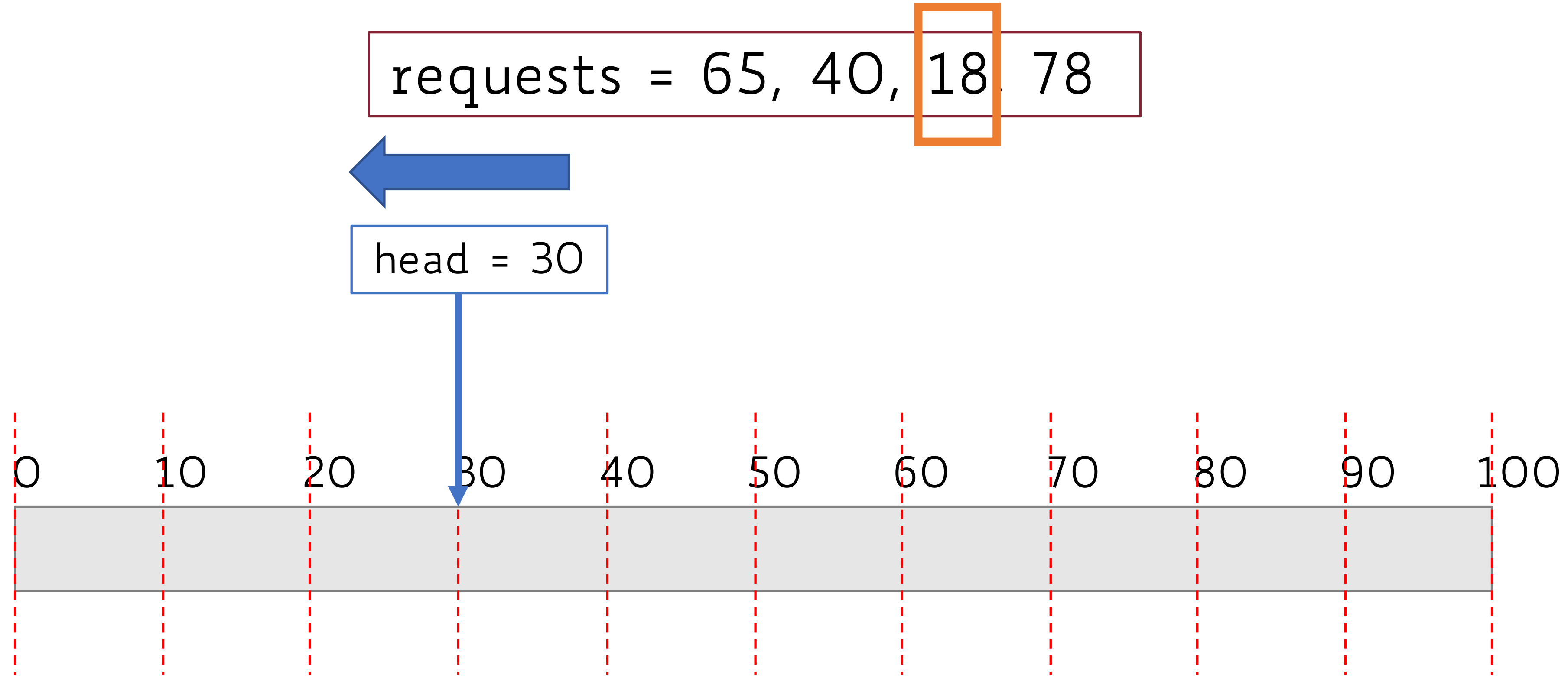
Disk Scheduling: SCAN

requests = 65, 40, 18, 78



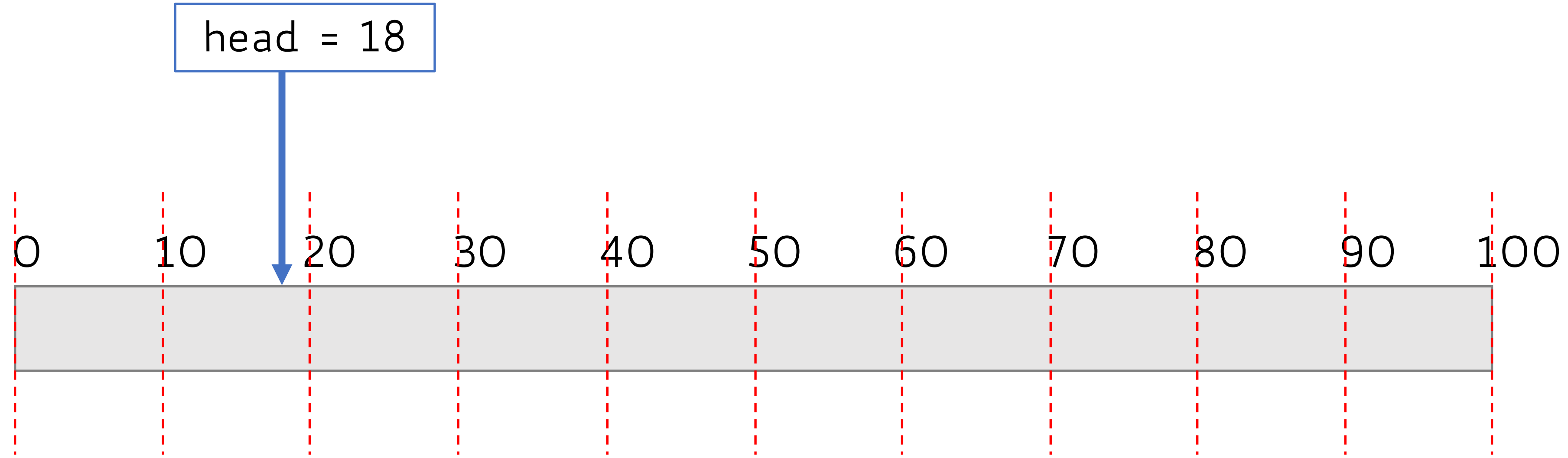
Distance travelled = 0

Disk Scheduling: SCAN



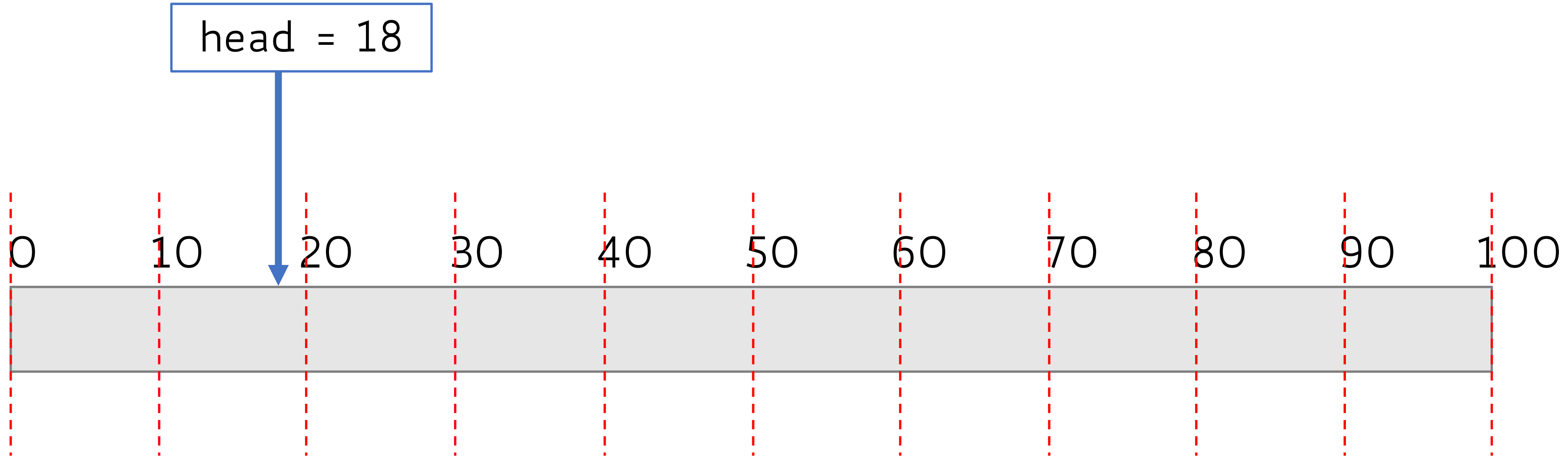
Disk Scheduling: SCAN

requests = 65, 40, 18, 78



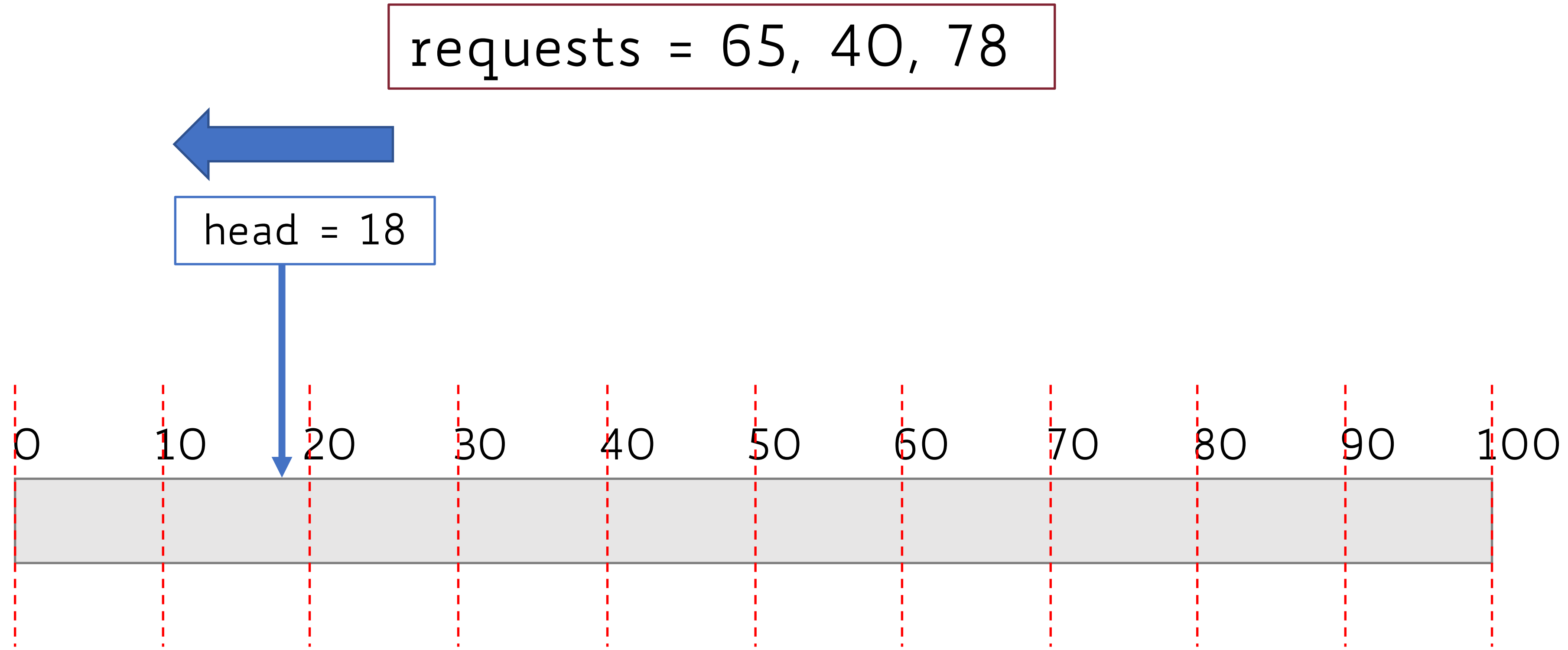
Disk Scheduling: SCAN

requests = 65, 40, 18, 78



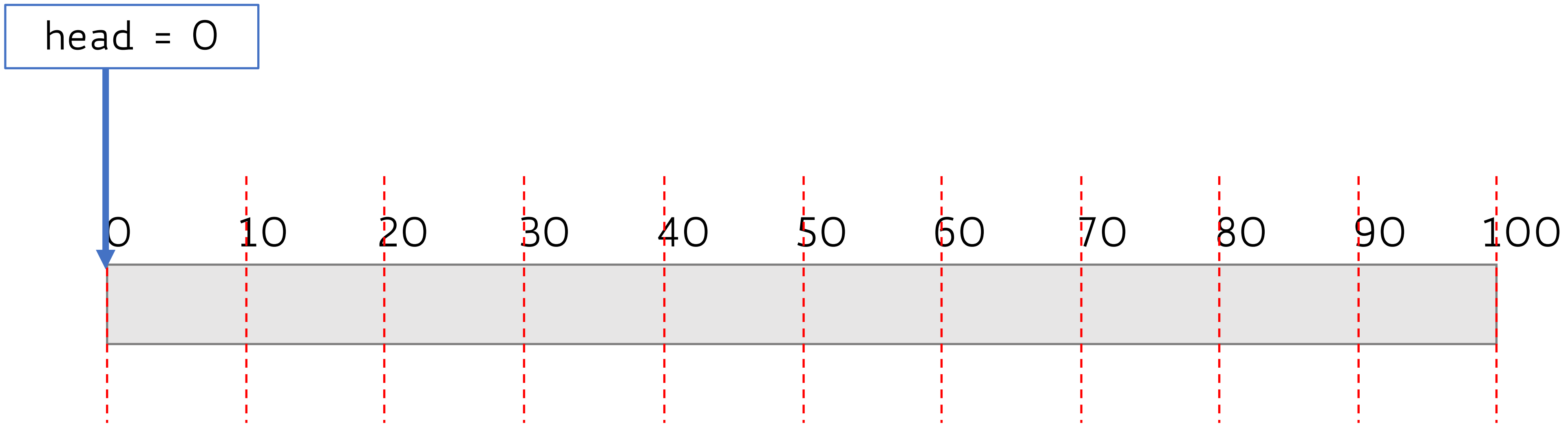
Distance travelled = 0 + |18 – 30| = 12

Disk Scheduling: SCAN



Disk Scheduling: SCAN

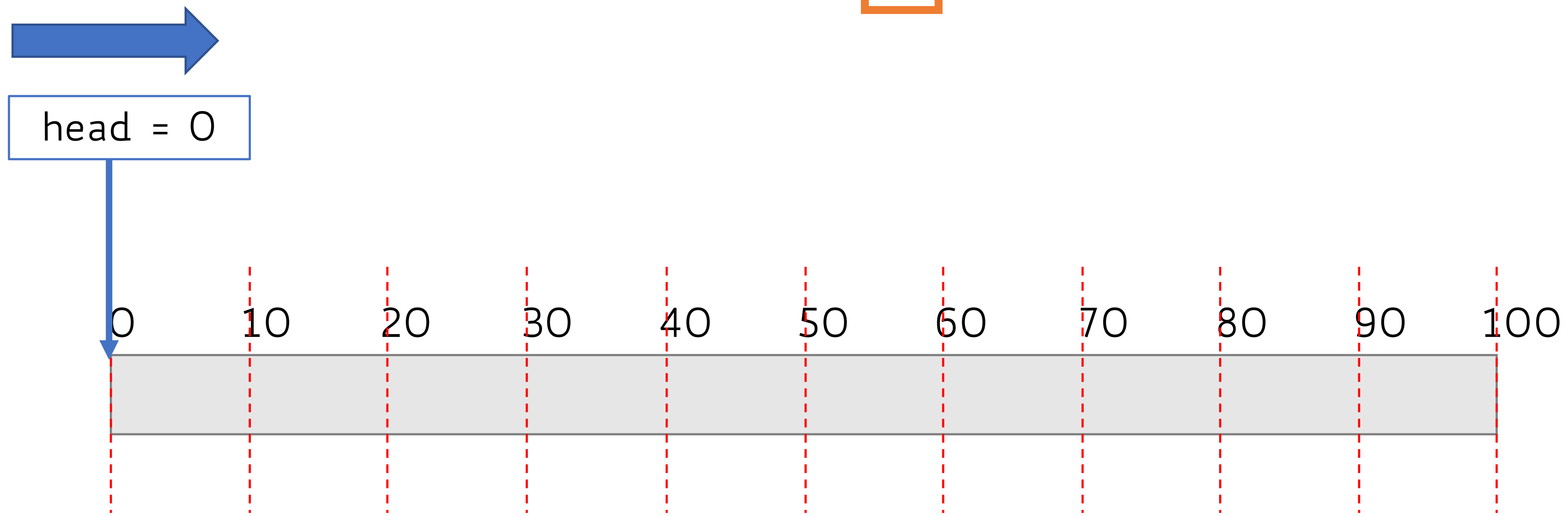
requests = 65, 40, 78



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

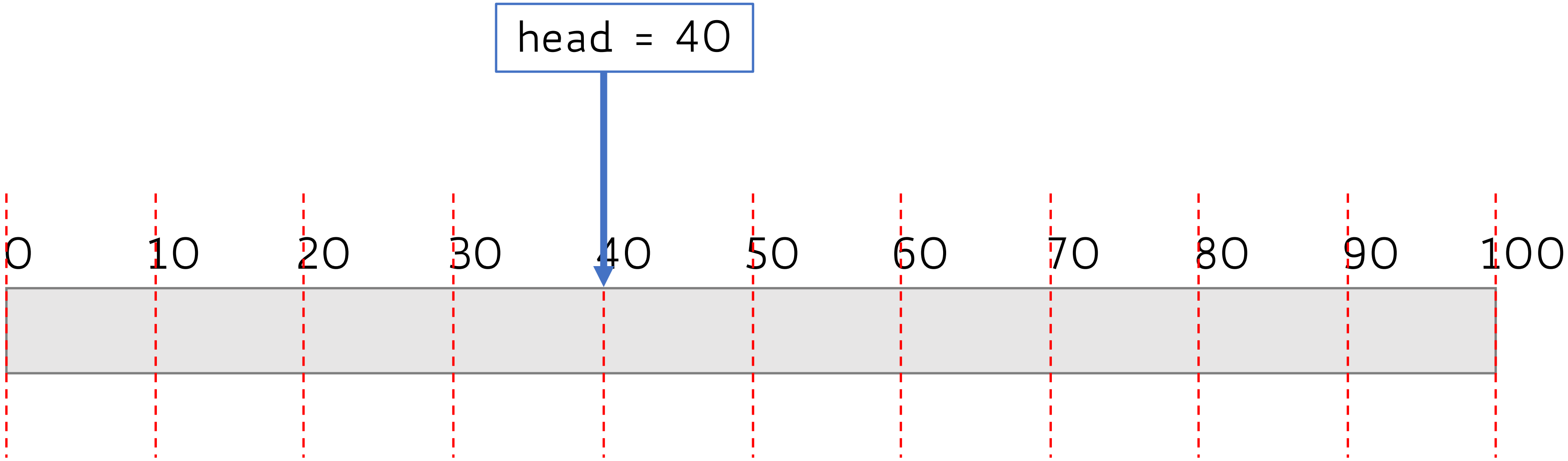
Disk Scheduling: SCAN

requests = 65, $\boxed{40}$, 78



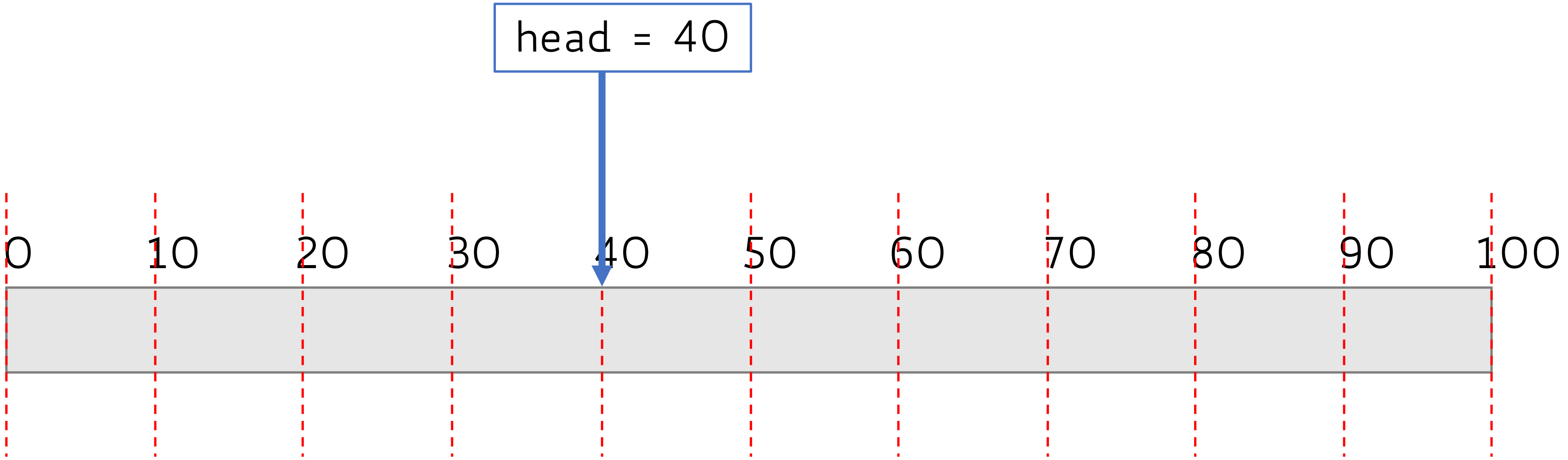
Disk Scheduling: SCAN

requests = 65, 40, 78



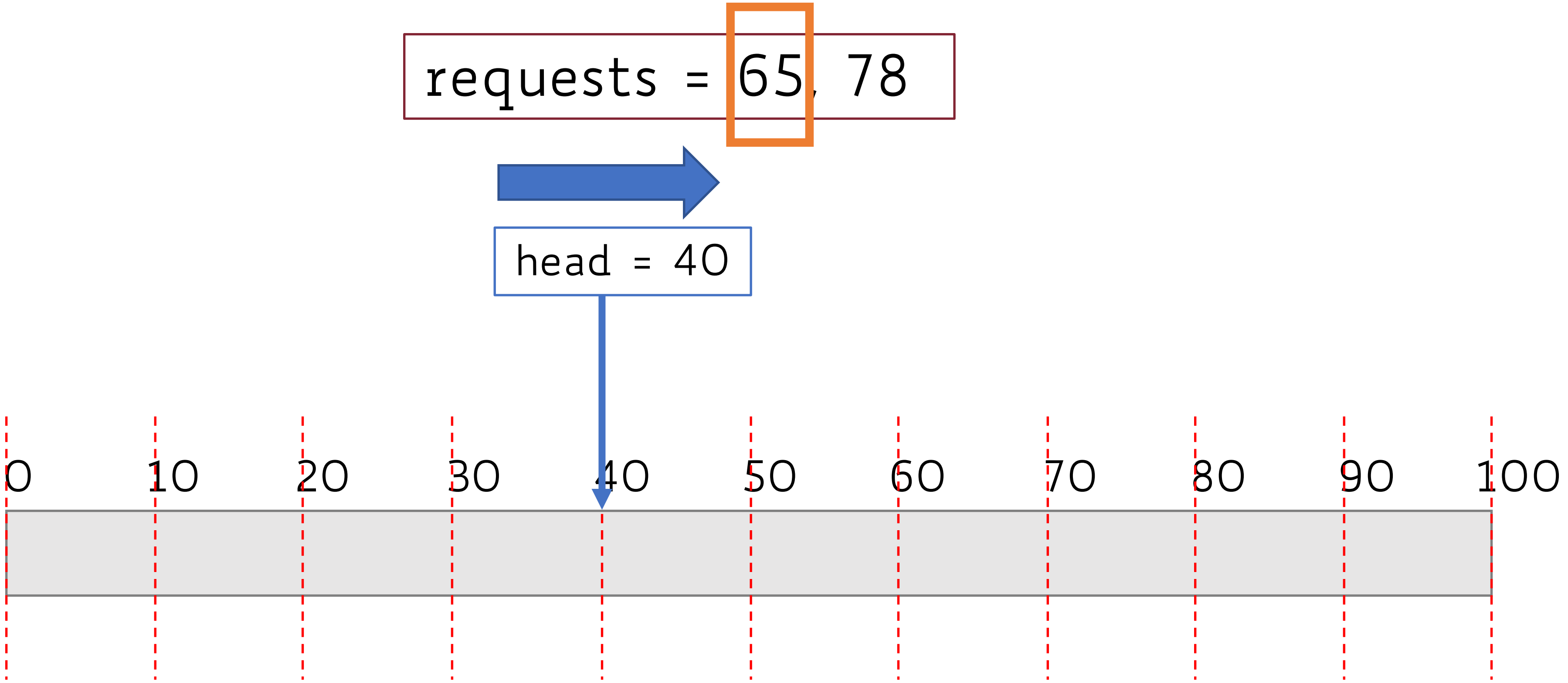
Disk Scheduling: SCAN

requests = 65, 40, 78



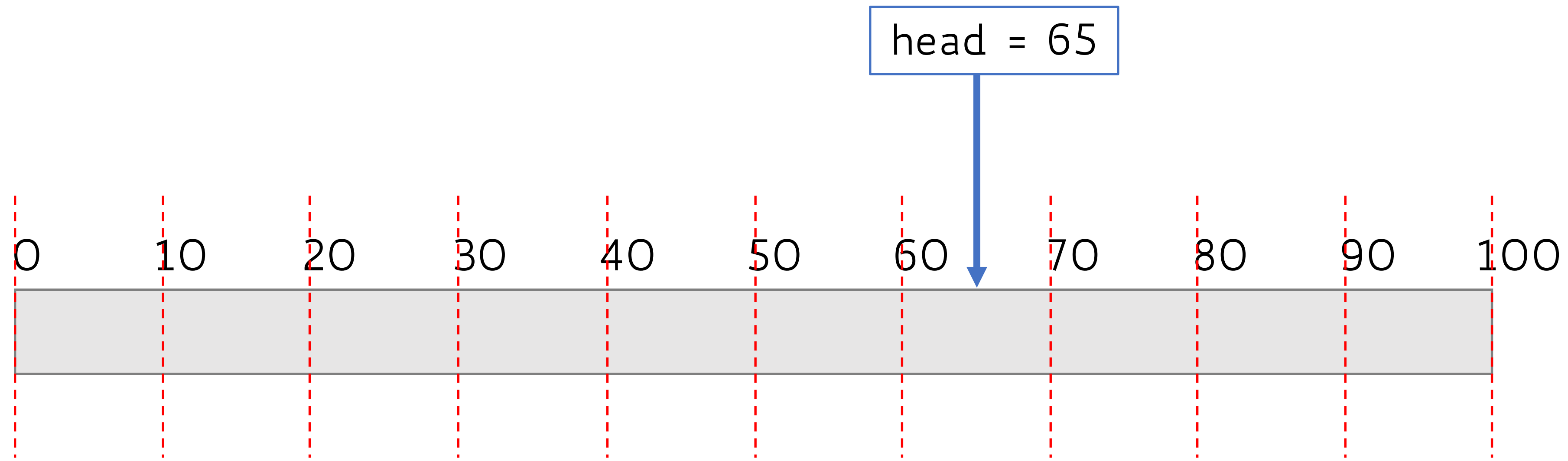
Distance travelled = 30 + |40 – 0| = 70

Disk Scheduling: SCAN



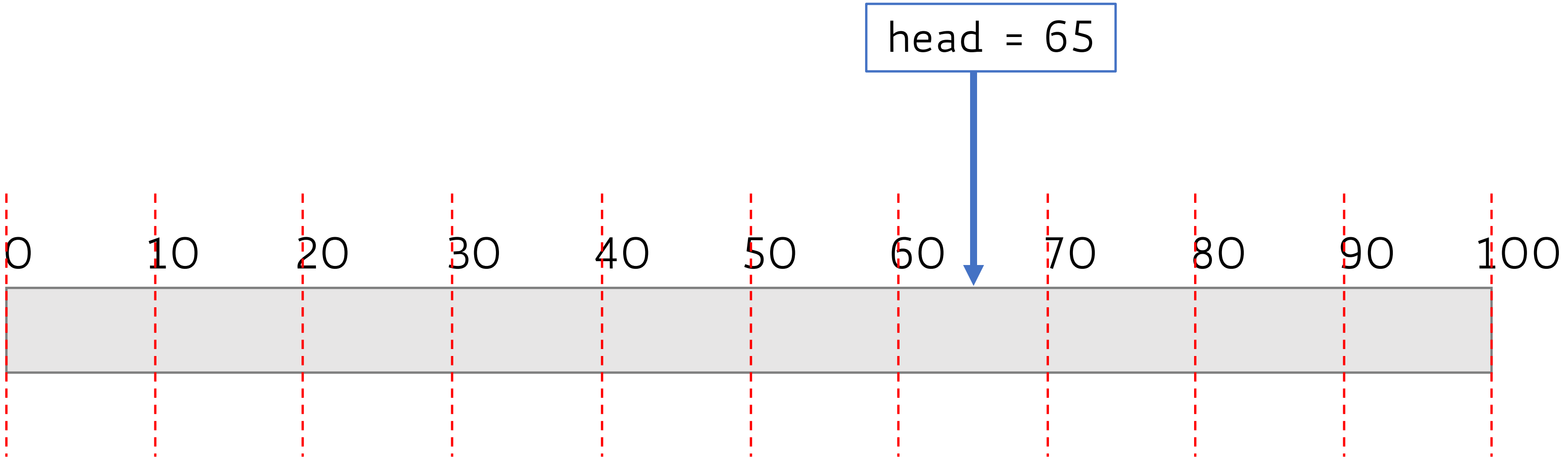
Disk Scheduling: SCAN

requests = 65, 78



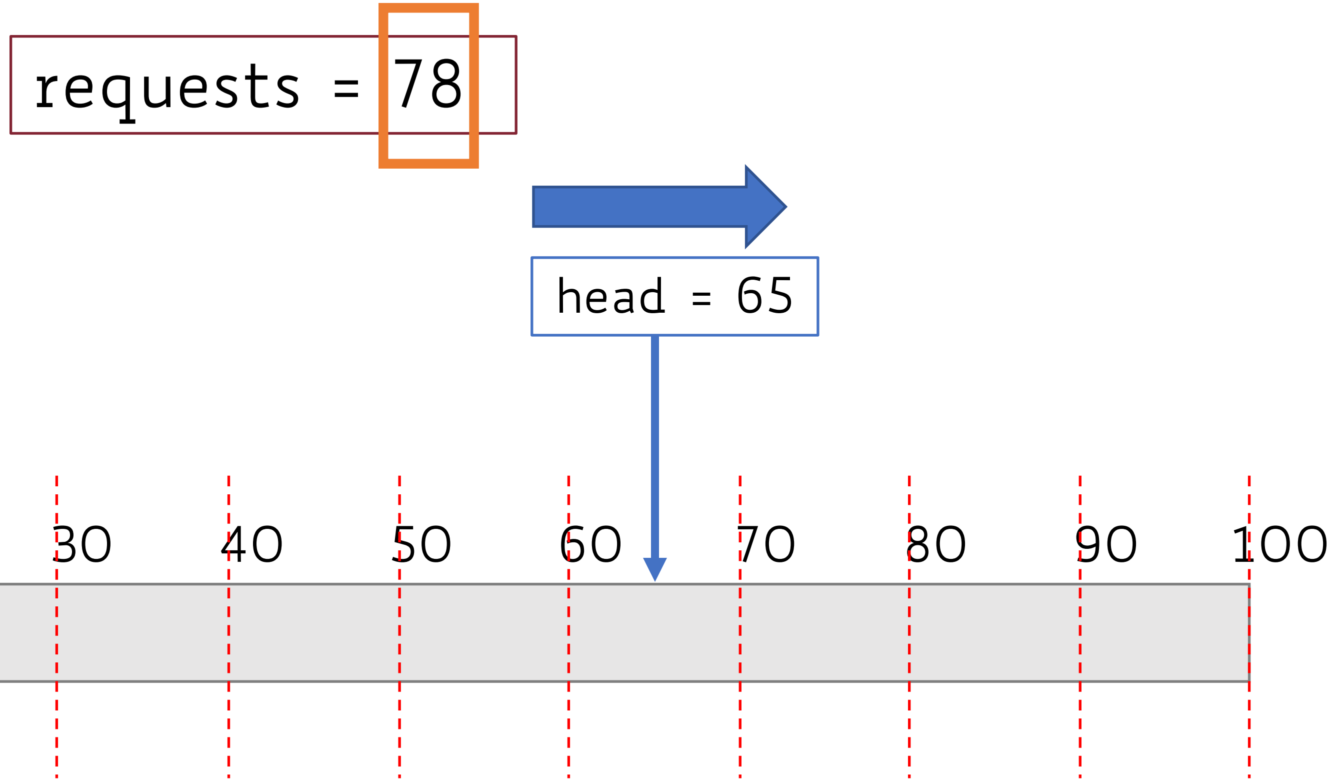
Disk Scheduling: SCAN

requests = 65, 78

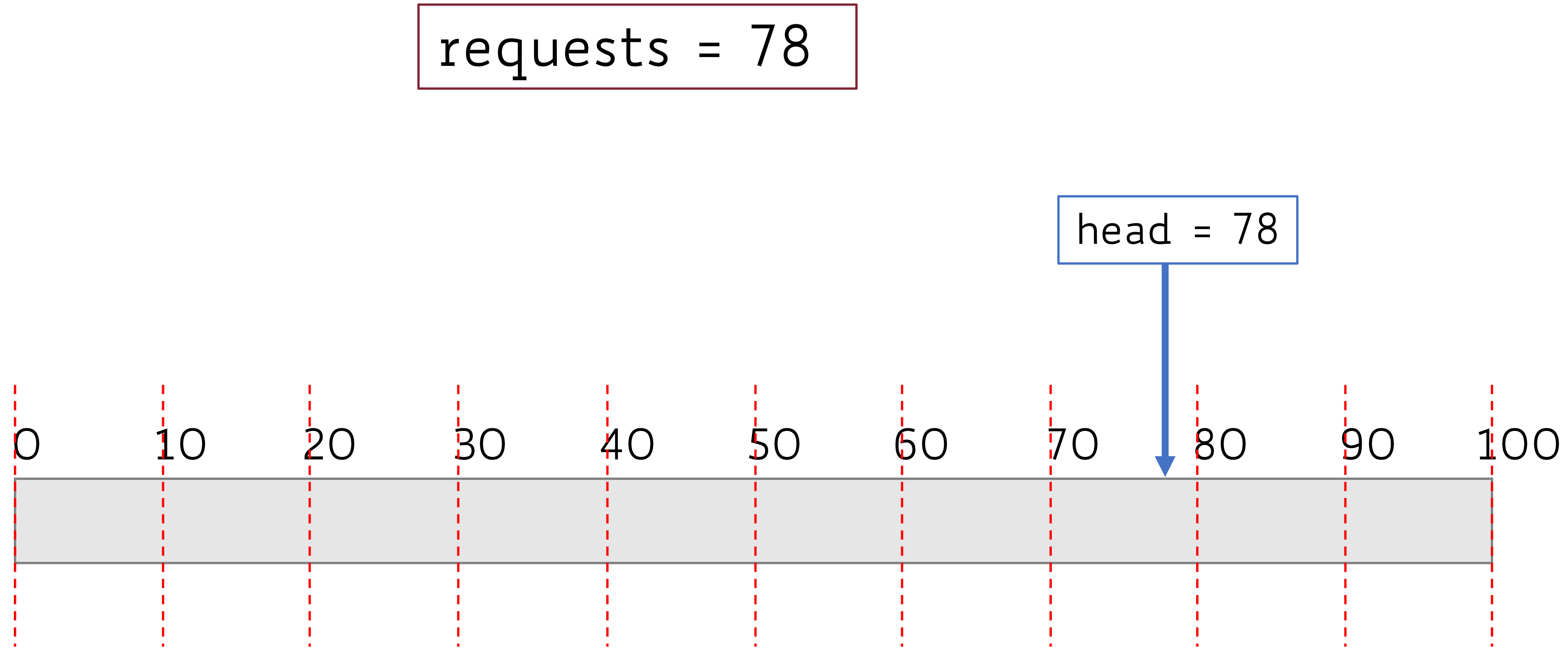


Distance travelled = 70 + |65 – 40| = 95

Disk Scheduling: SCAN

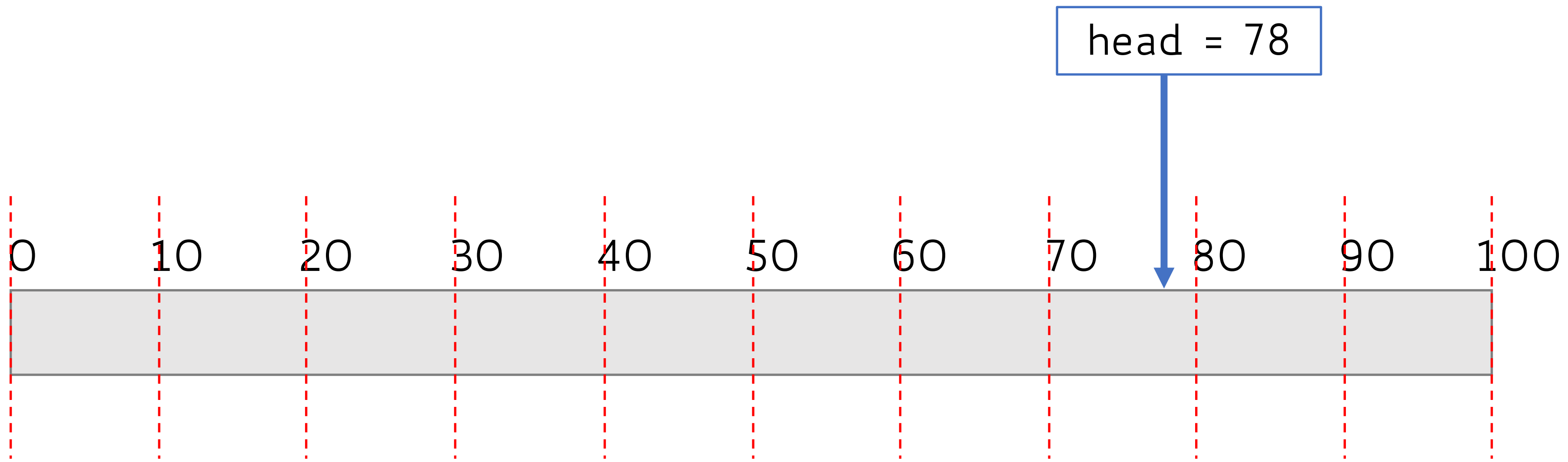


Disk Scheduling: SCAN



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
- Seek time might be larger with SCAN than SSTF when:
  - New requests are coming in immediately after SCAN has passed
  - 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
- Seek time might be larger with SCAN than SSTF when:
  - New requests are coming in immediately after SCAN has passed
  - 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 results in longer waiting time

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!

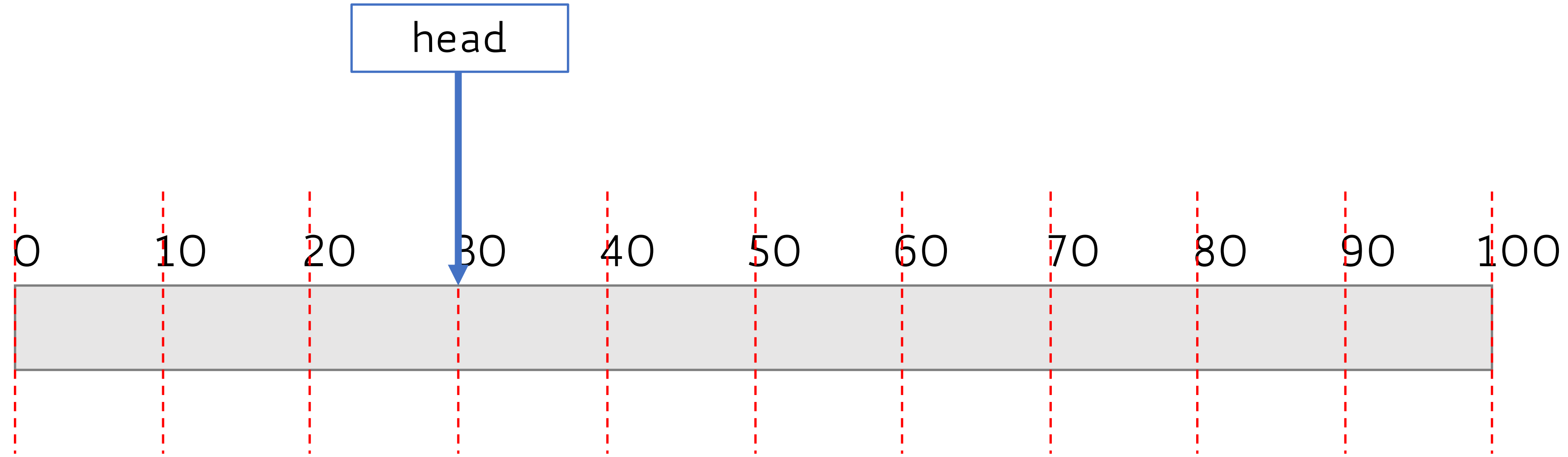
Appendix: Arm Speed

- 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

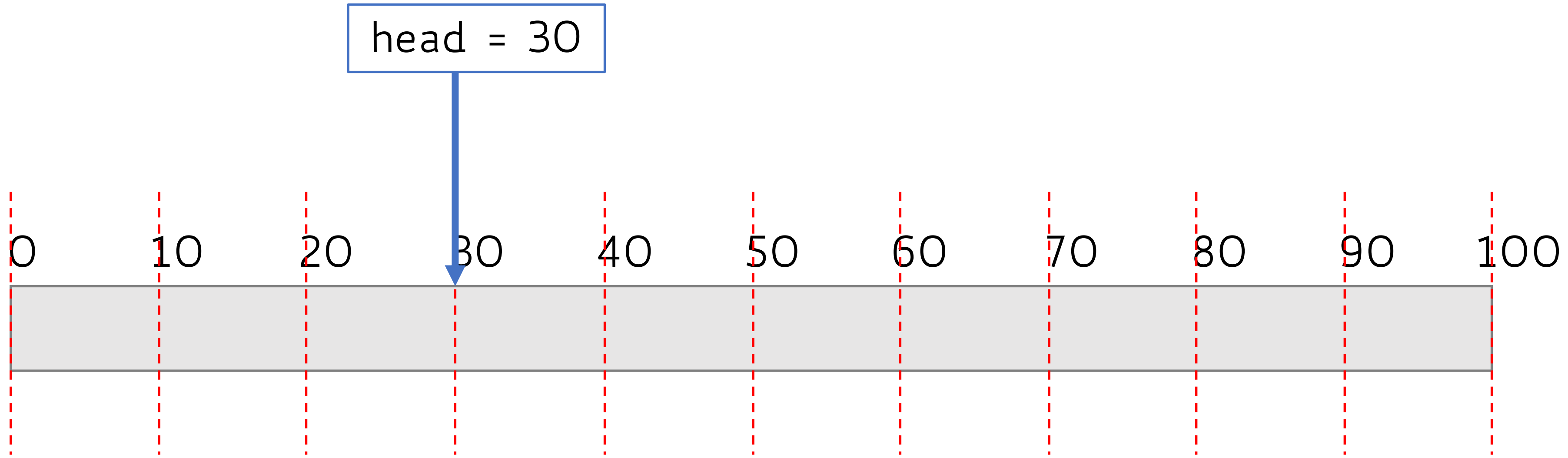
Head makes circular scan of the disk (requests are in a circular queue)

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



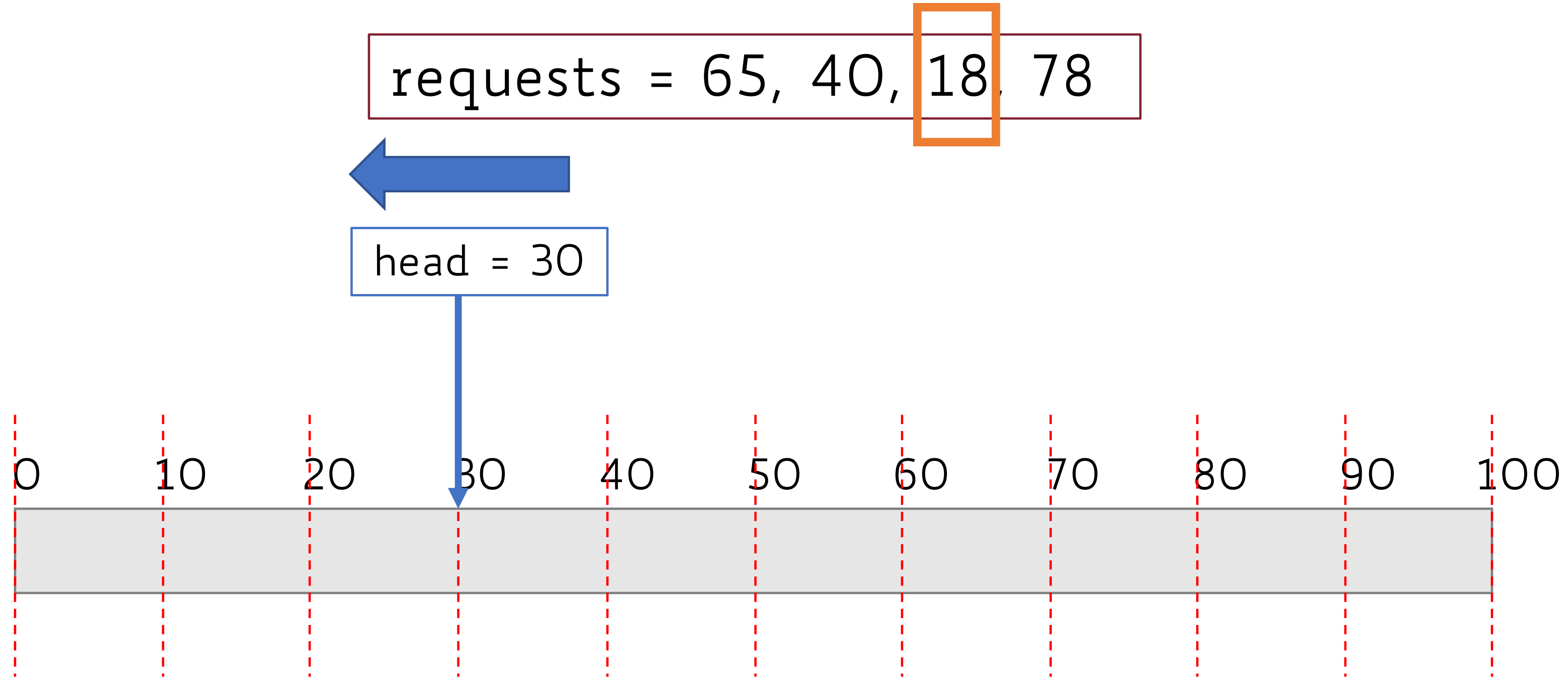
Disk Scheduling: C-SCAN

requests = 65, 40, 18, 78



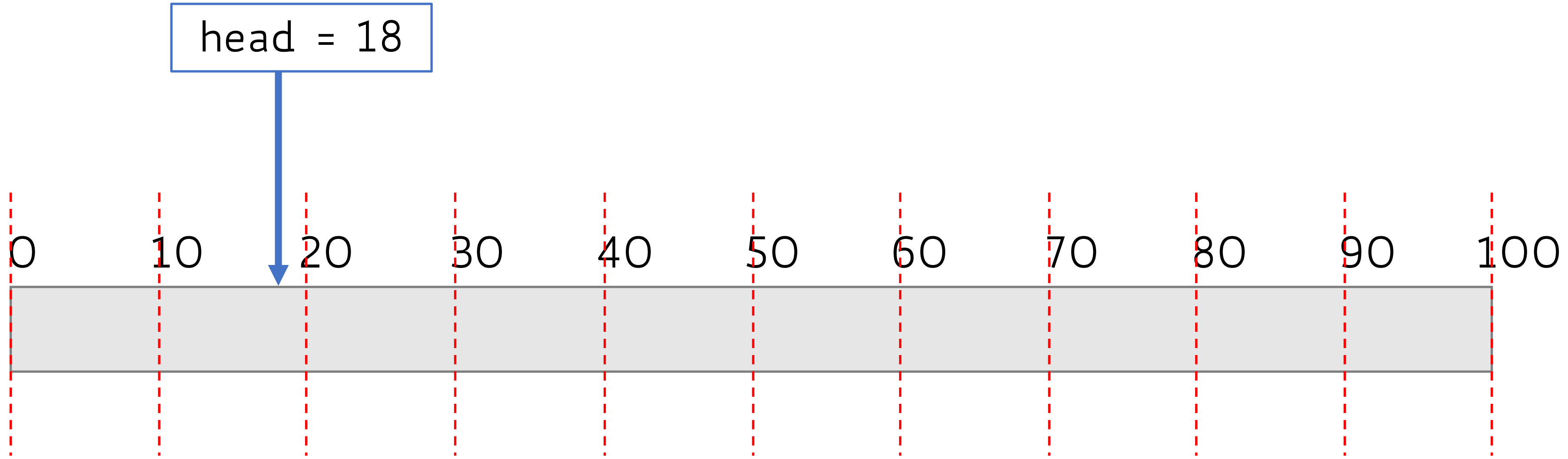
Distance travelled = 0

Disk Scheduling: C-SCAN



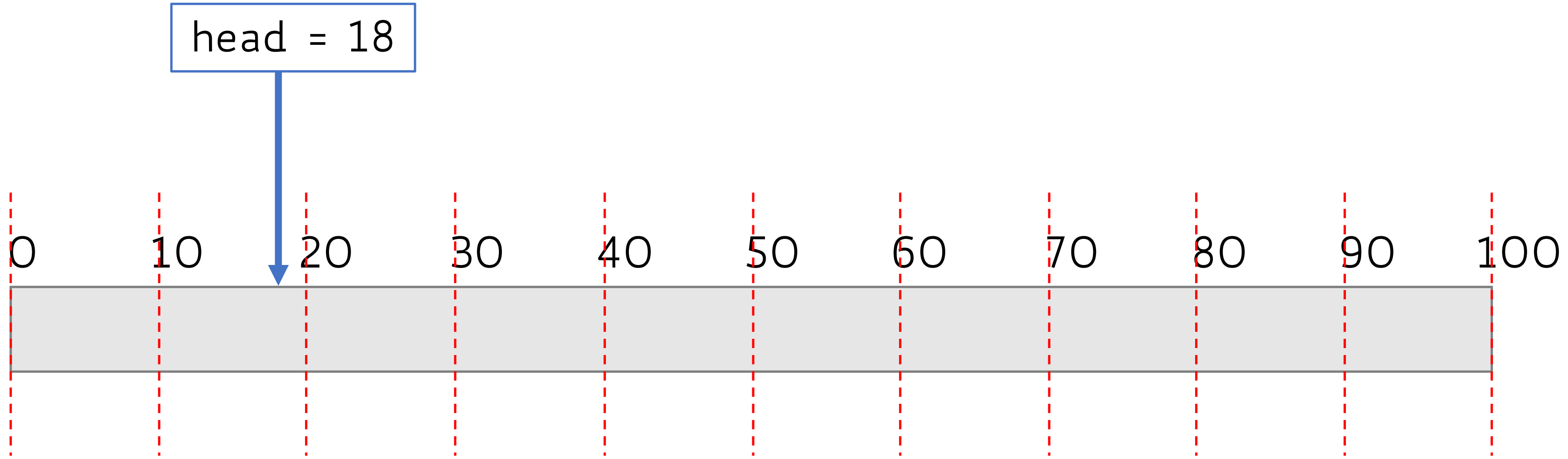
Disk Scheduling: C-SCAN

requests = 65, 40, 18, 78



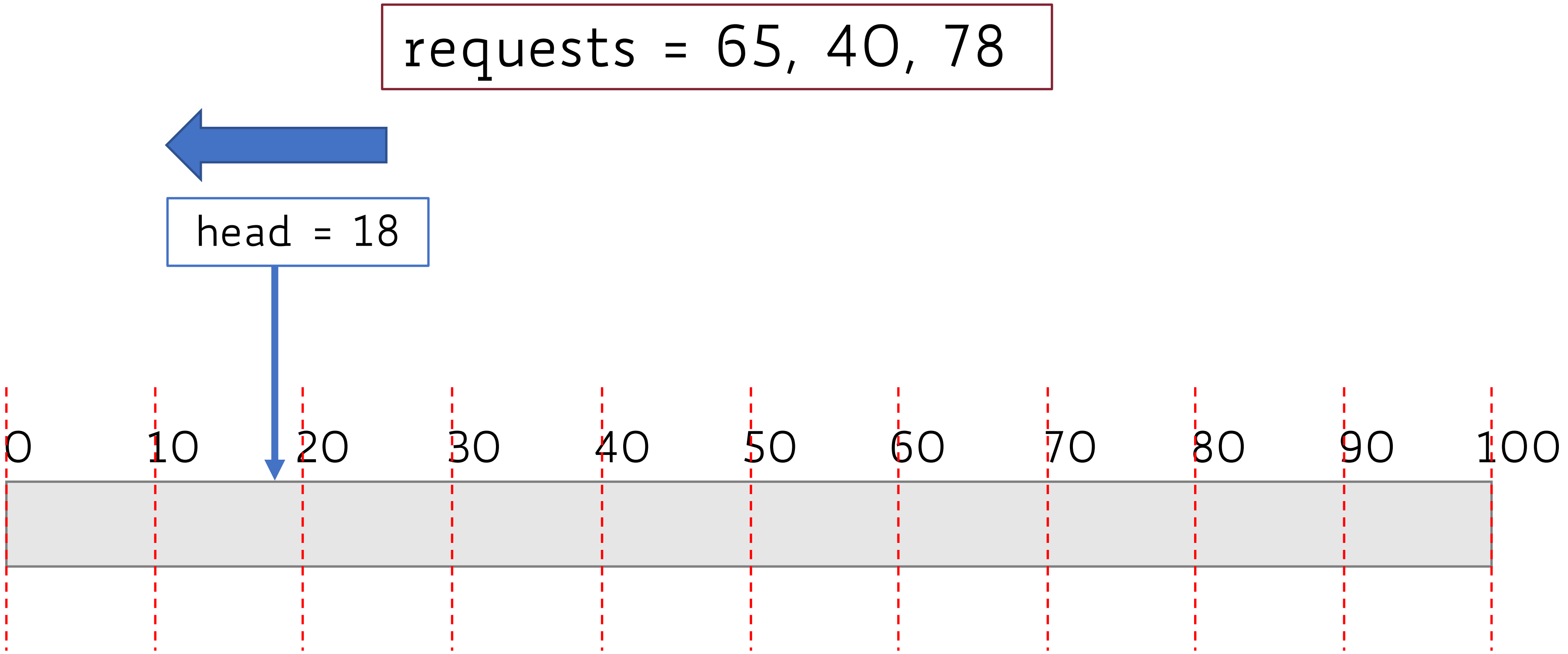
Disk Scheduling: C-SCAN

requests = 65, 40, 18, 78



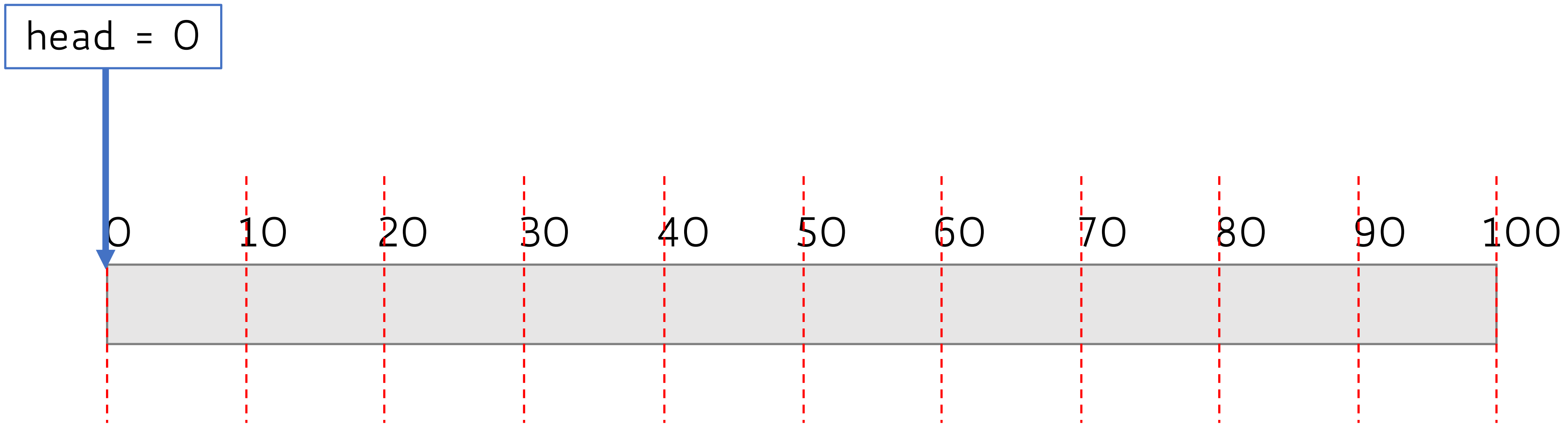
Distance travelled = 0 + |18 – 30| = 12

Disk Scheduling: C-SCAN



Disk Scheduling: C-SCAN

requests = 65, 40, 78

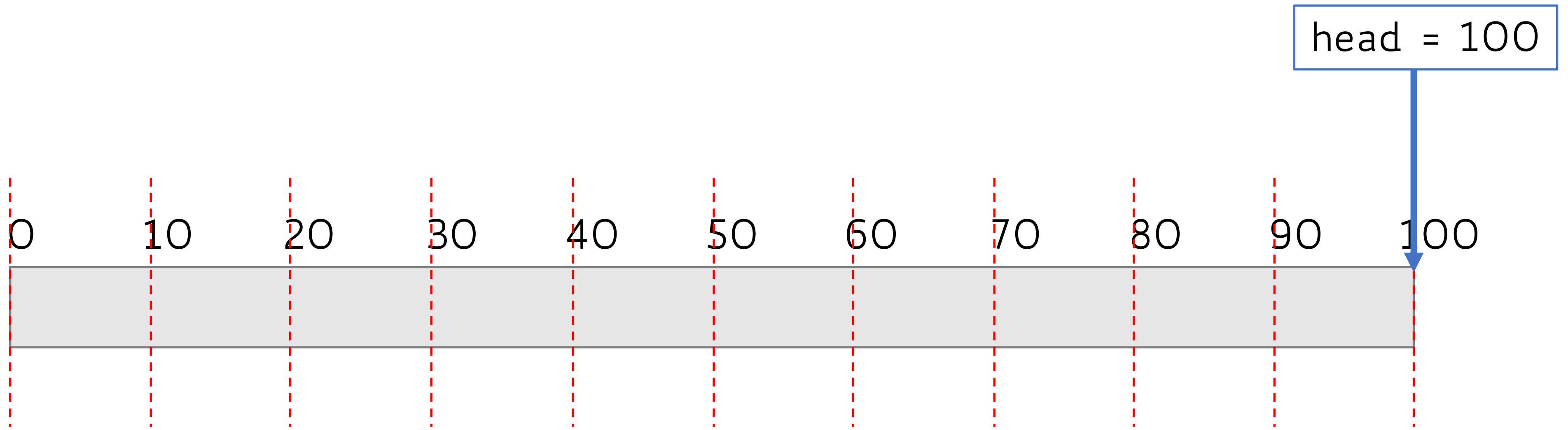


Distance travelled = 12 + |0 – 18| = 30

Disk Scheduling: C-SCAN

requests = 65, 40, 78

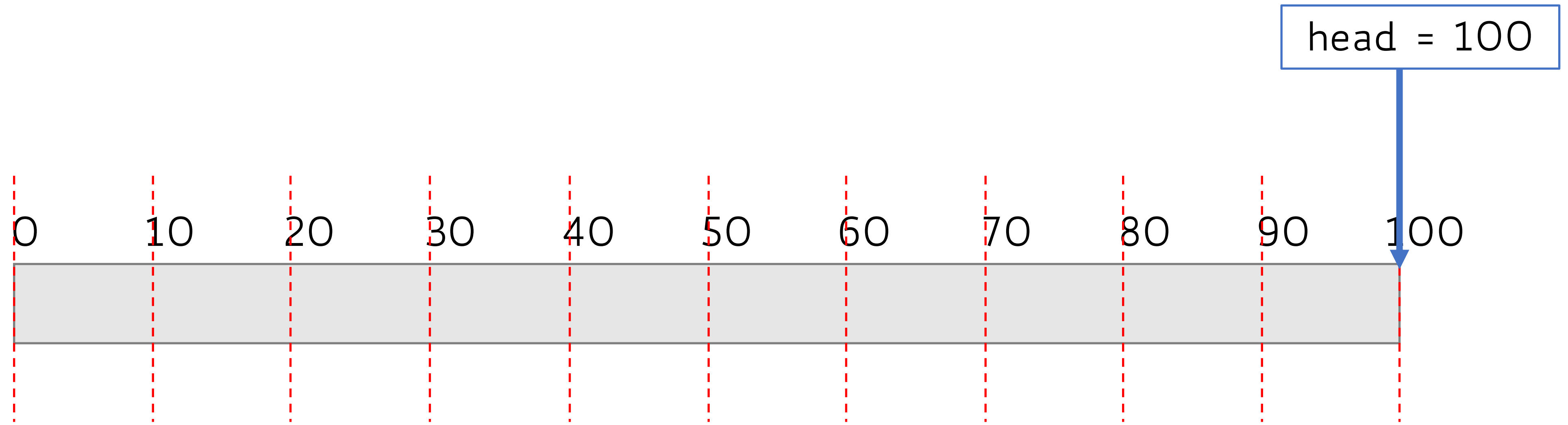
Head reset



Disk Scheduling: C-SCAN

requests = 65, 40, 78

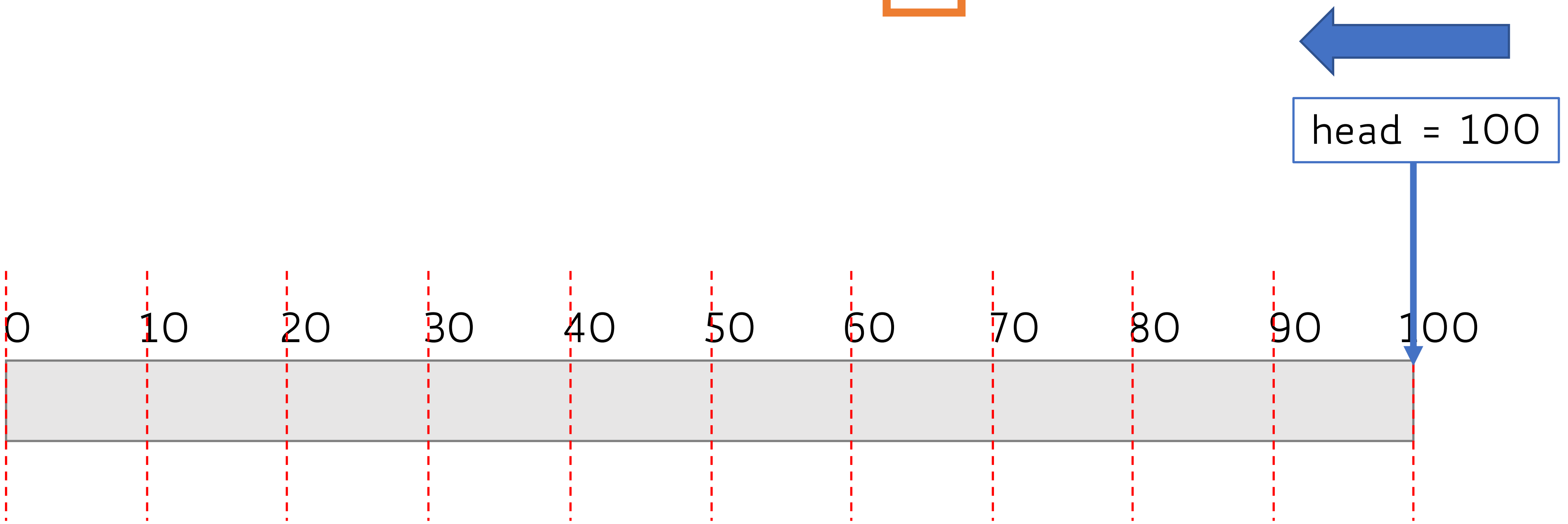
Head reset



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

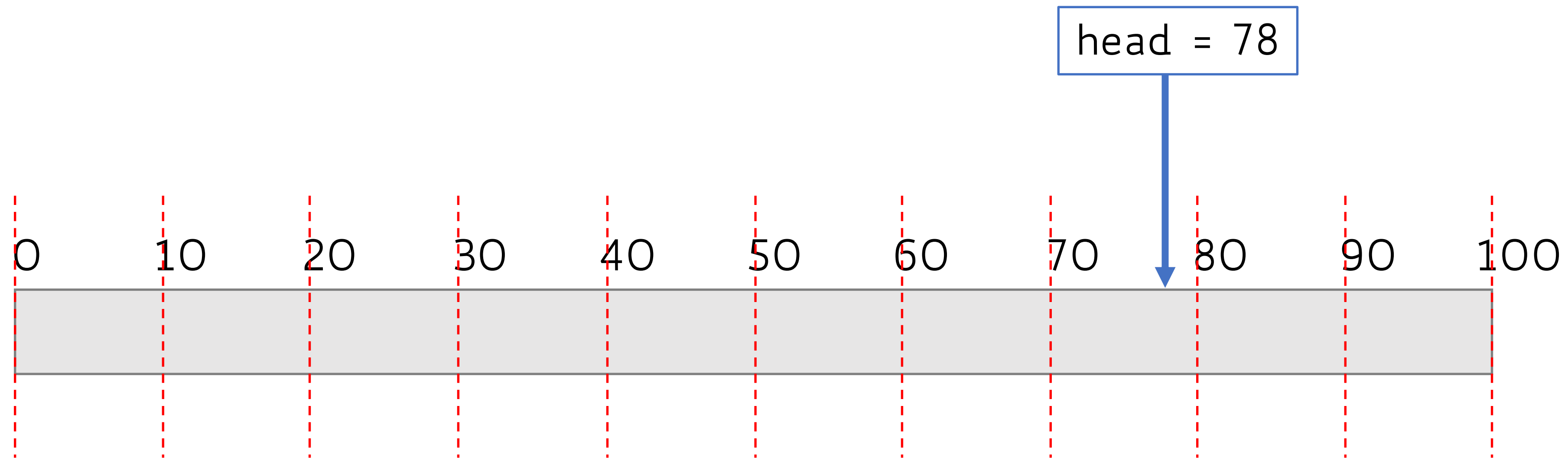
Disk Scheduling: C-SCAN

requests = 65, 40, $\boxed{78}$



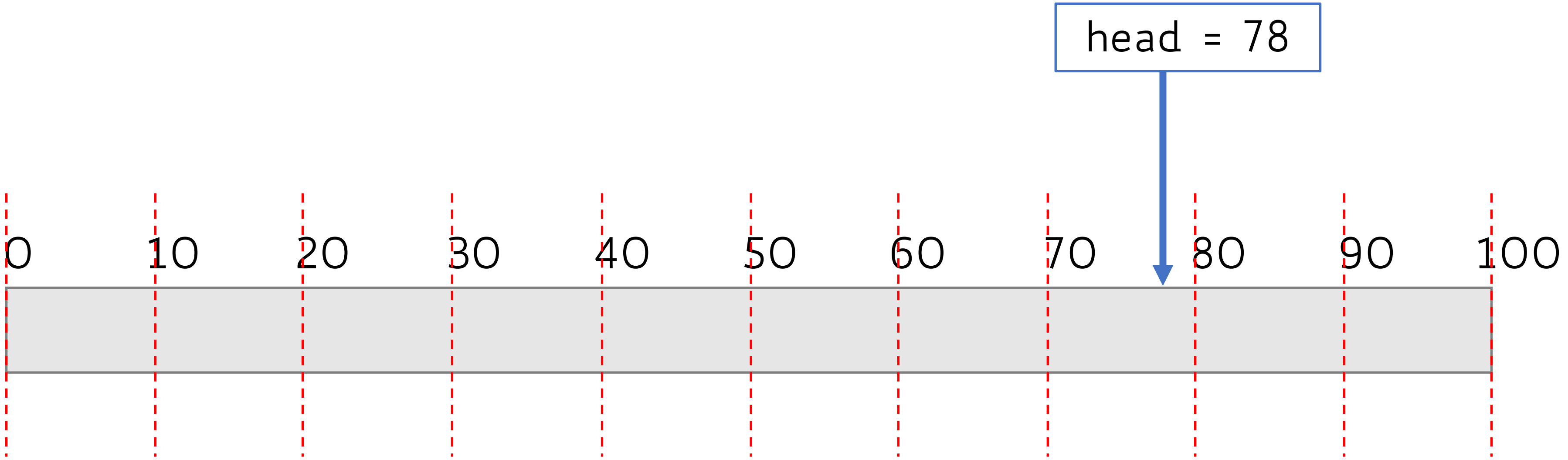
Disk Scheduling: C-SCAN

requests = 65, 40, 78



Disk Scheduling: C-SCAN

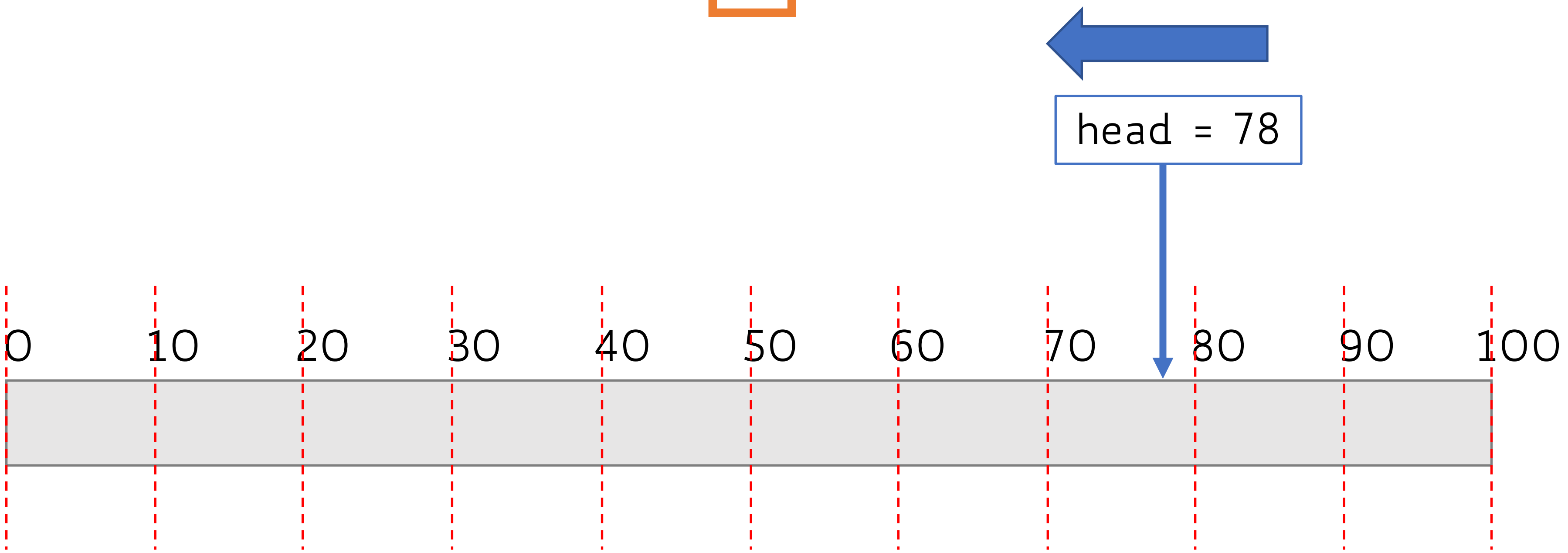
requests = 65, 40, 78



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

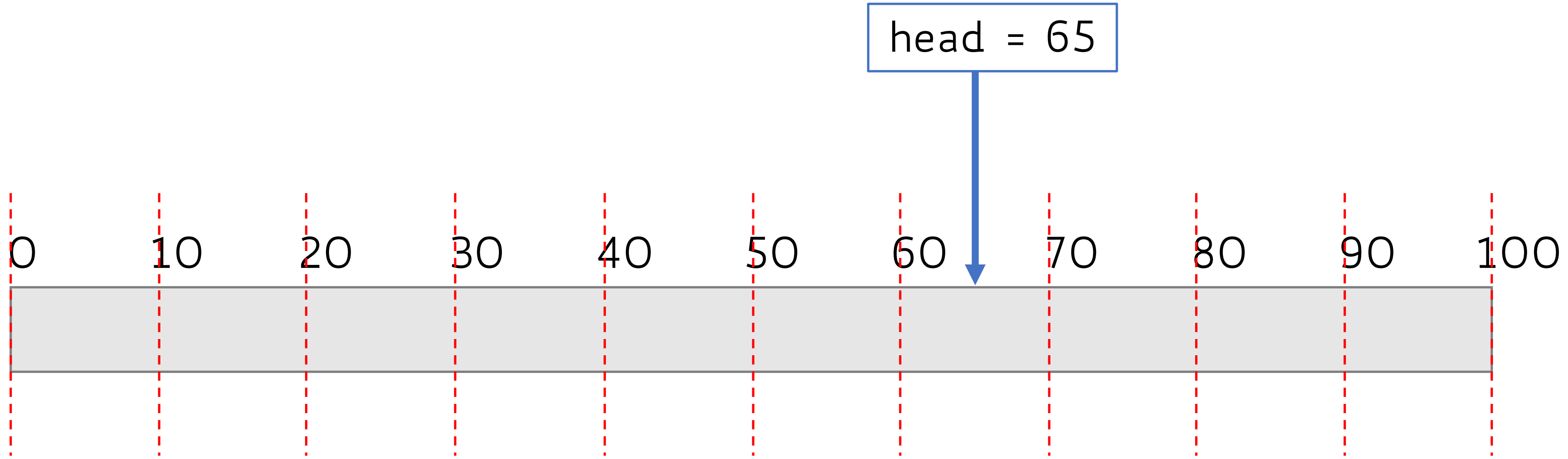
Disk Scheduling: C-SCAN

requests = $\boxed{65}$, 40



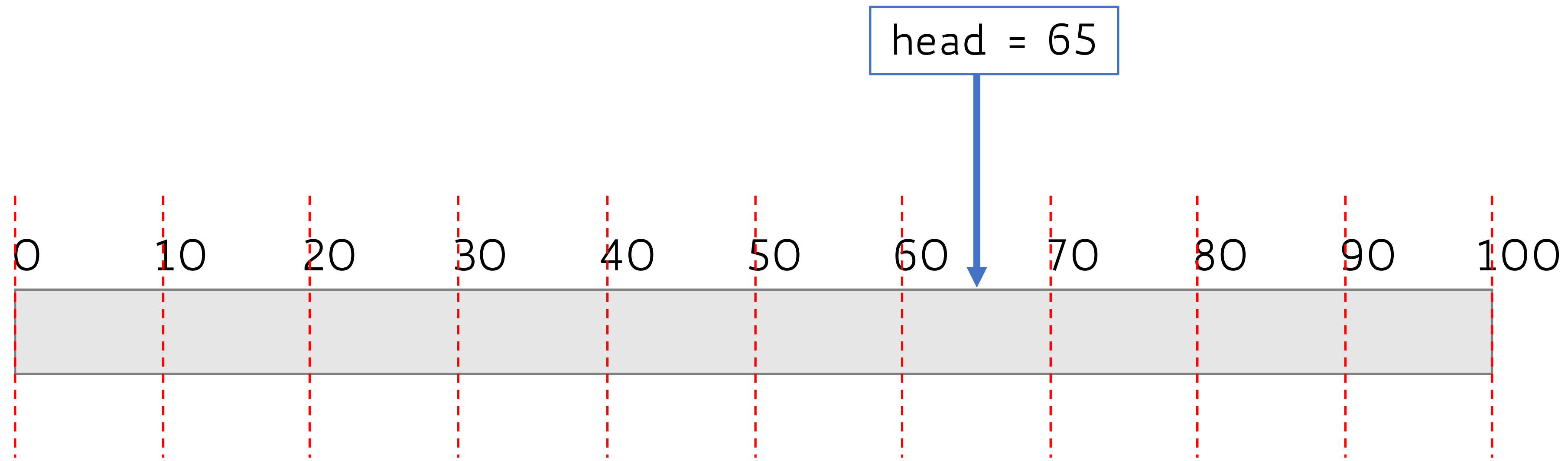
Disk Scheduling: C-SCAN

requests = 65, 40



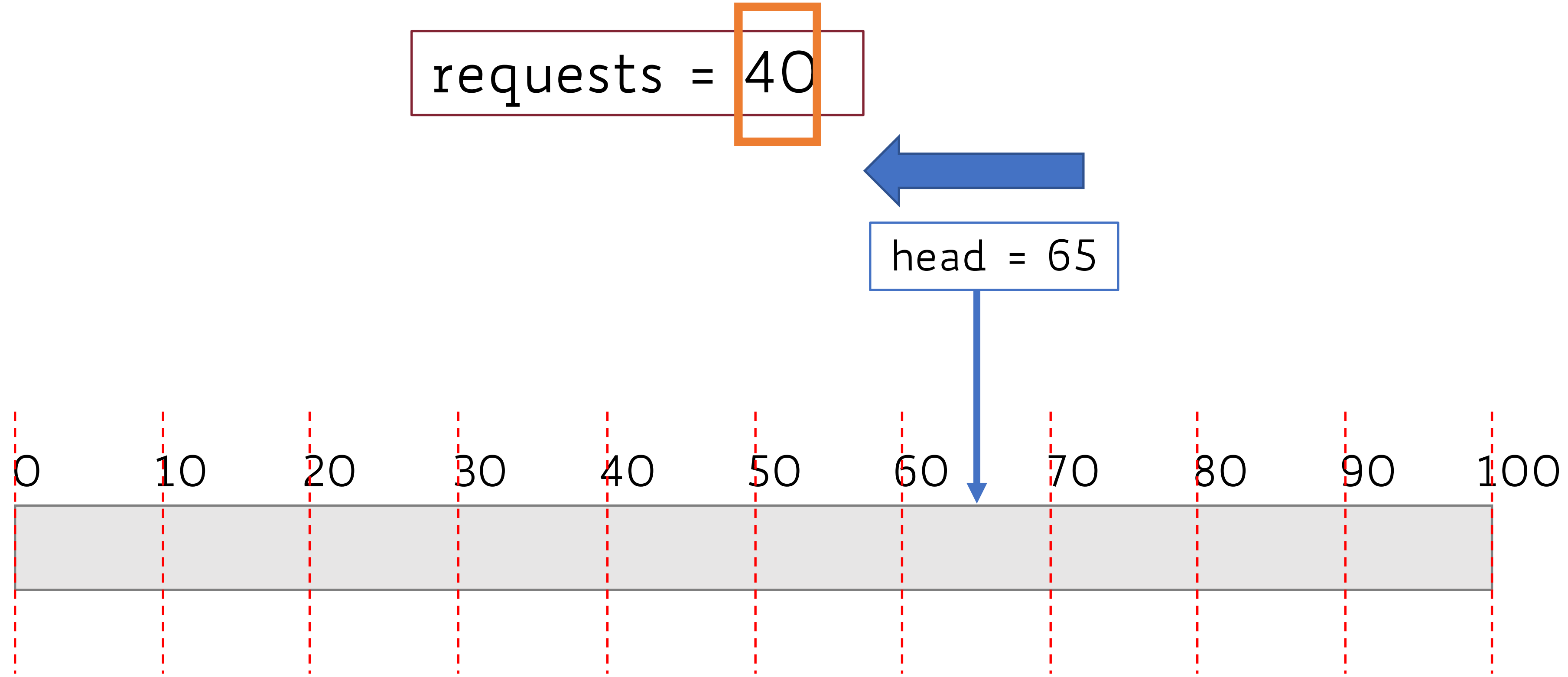
Disk Scheduling: C-SCAN

requests = 65, 40



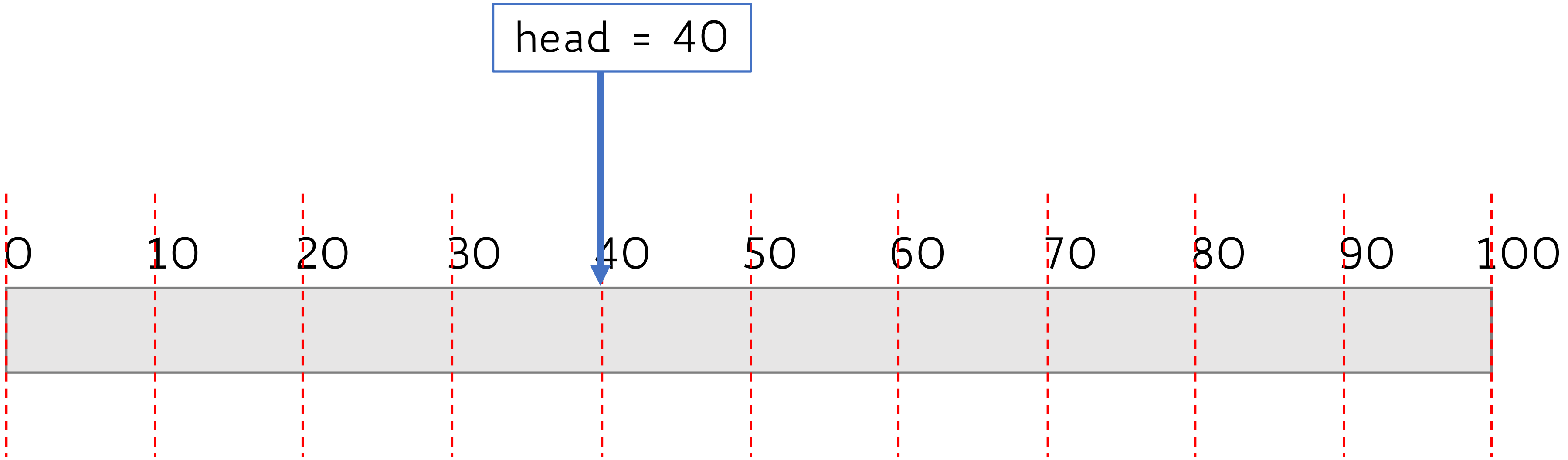
Distance travelled = 152 + |65 – 78| = 165

Disk Scheduling: C-SCAN



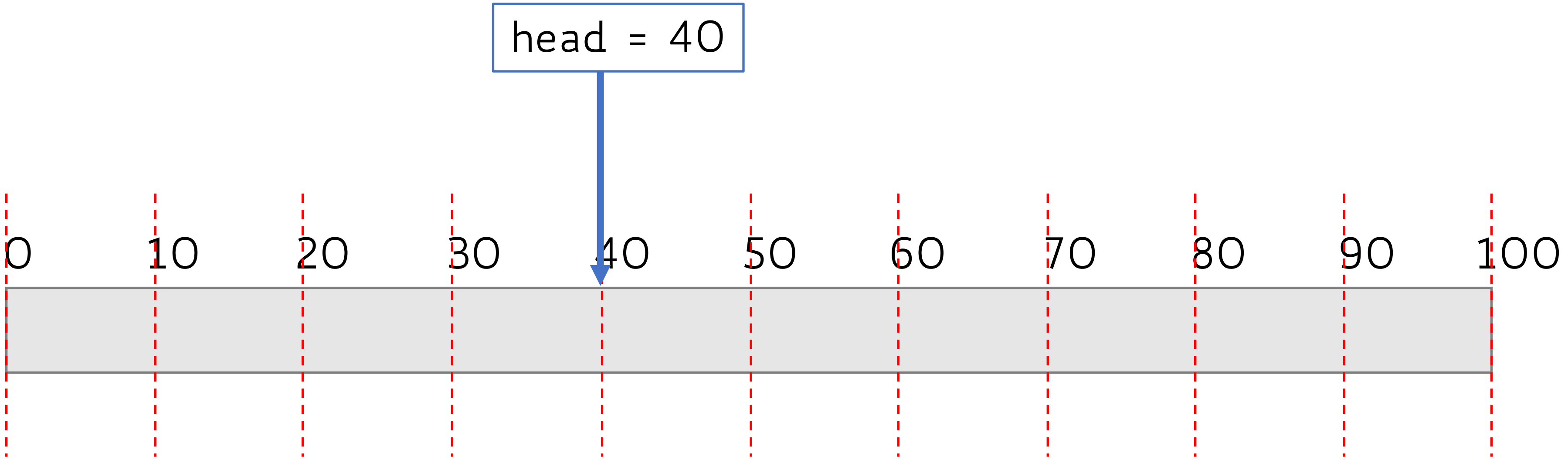
Disk Scheduling: C-SCAN

requests = 40



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 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 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: 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

Today, CPUs are so fast that interleaving is not used anymore!

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)

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**

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

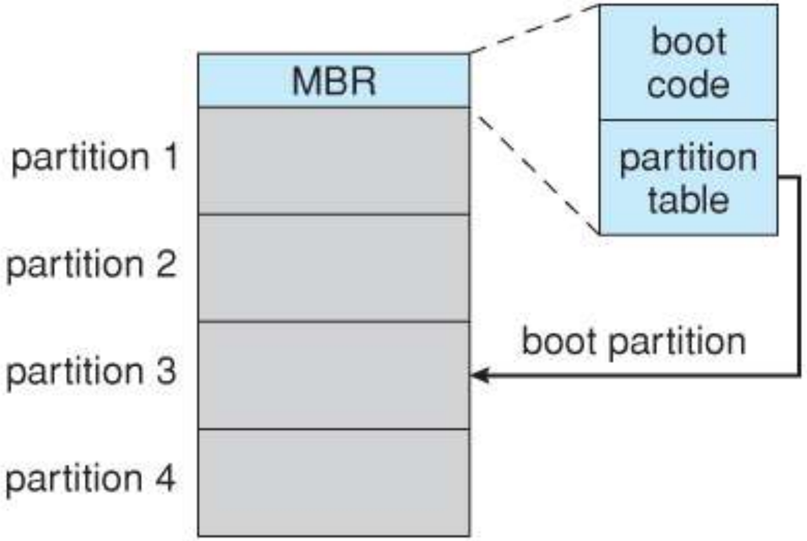
This loads that sector into memory, and transfer control over to it

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

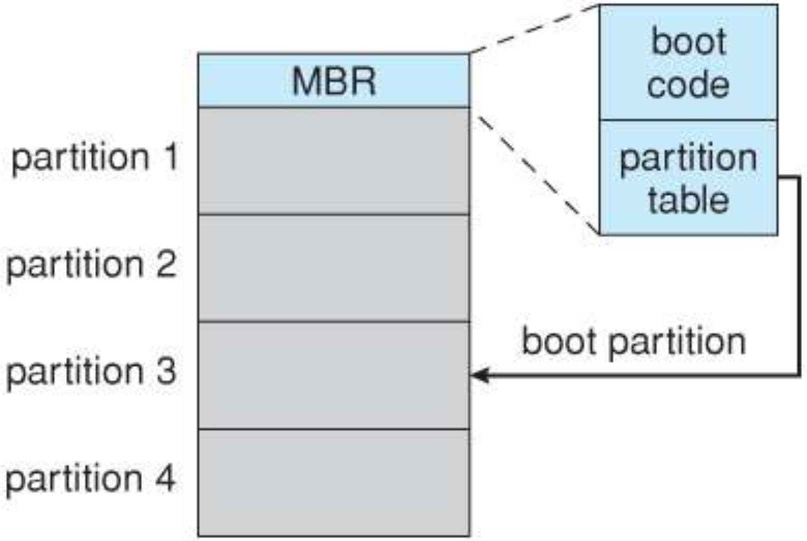
This loads that sector into memory, and transfer control over to it

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



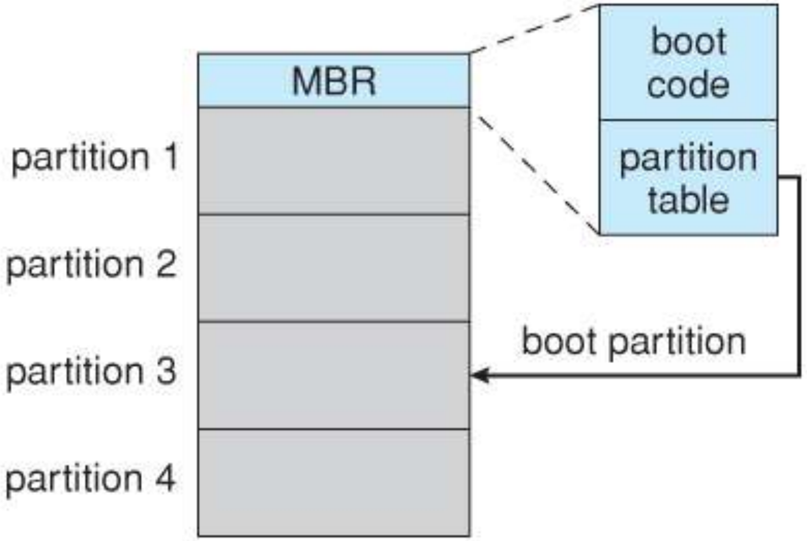
This loads that sector into memory, and transfer control over to it

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

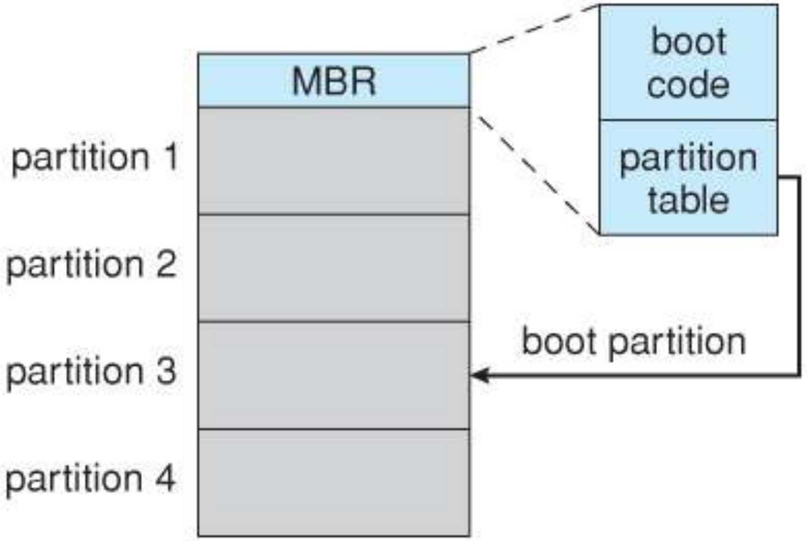
The first sector is known as the Master Boot Record (MBR), and contains a very small amount of code and the partition table

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 control is transferred 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
  - 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, switching the definition to **Independent** rather than **Inexpensive** 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**

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

Example: Disk Failure

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

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

- 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**

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

- Associate with each disk a random variable $X_{i,t}$
  - $X_{i,t} \sim \mathit{Bernoulli}(p)$ 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**

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

- 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.

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

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.

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

$$\boxed{T \sim \text{Binomial}(N, p)}$$

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.

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

$$T \sim \text{Binomial}(N, p)$$

$$E[T] = Np$$

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

$$\boxed{p^N}$$

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

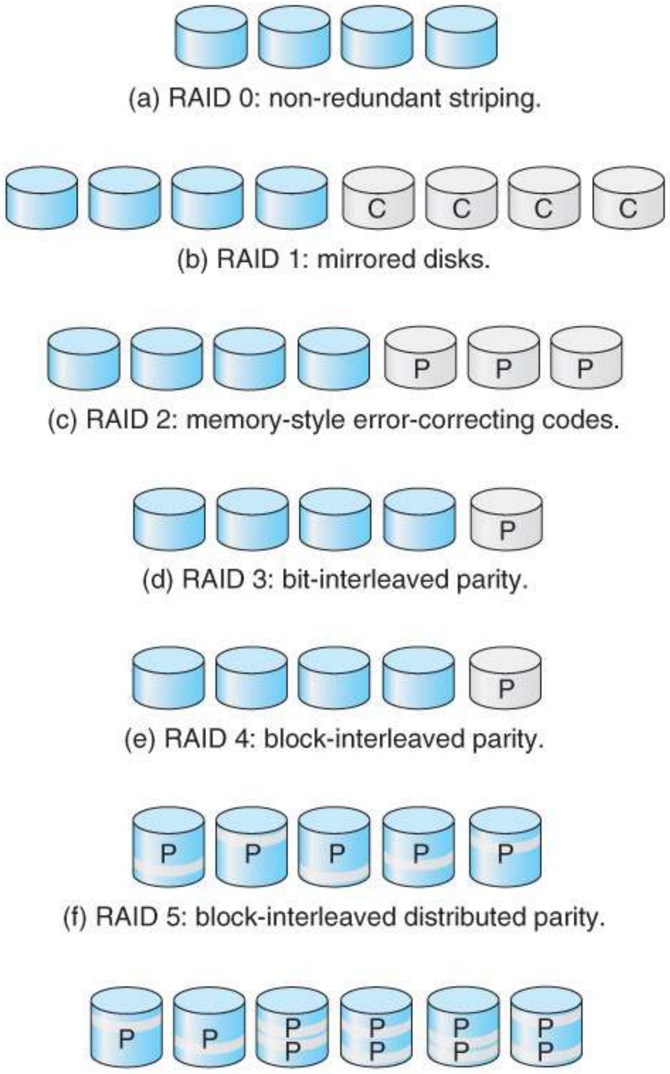
RAID Levels

- 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 devices efficiently is crucial
- I/O requests can be re-arranged so as to reduce mechanical movements of magnetic drives → **disk scheduling algorithms**
- Redundancy allows to cope with disk failure in critical application domains