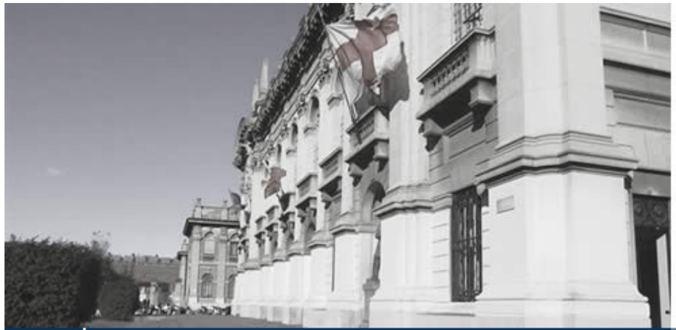


Computing Infrastructure

POLITECNICO DI MILANO





Disk abstraction and HDD

The topics of the course: what are we going to see today?

HW Infrastructures:

System-level: Computing Infrastructures and Data Center Architectures, Rack/Structure;

Node-level: Server (computation, HW accelerators), **Storage (Type, technology),** Networking (architecture and technology);

Building-level: Cooling systems, power supply, failure recovery

SW Infrastructures:

Virtualization: Process/System VM, Virtualization Mechanisms (Hypervisor, Para/Full virtualization)

Computing Architectures: Cloud Computing (types, characteristics), Edge/Fog Computing, X-as-a service

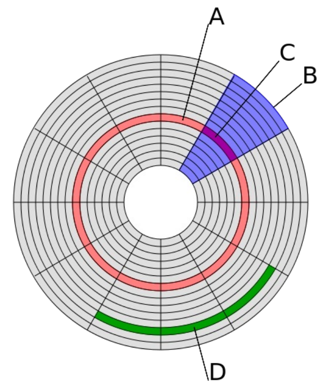
Methods:

Reliability and availability of datacenters (definition, fundamental laws, RBDs)

Disk performance (Type, Performance, RAID)

Scalability and performance of datacenters (definitions, fundamental laws, queuing network theory)

- Disks can be seen by an OS as a collection of *data blocks* that can be read or written independently.
- To allow the ordering/management among them, each block is characterized by a unique numerical address called LBA *(Logical Block Address)*.
- Typically, the OS groups blocks into *clusters* to simplify the access to the disk. Clusters are the minimal unit that an OS can read from or write to a disk.
- Typical cluster sizes range from 1 disk sector (512 B, or 4KB) to 128 sectors (64 KB).



(A) track (B) geometrical sector
(C) track sector (D) cluster

Files

Clusters contains:

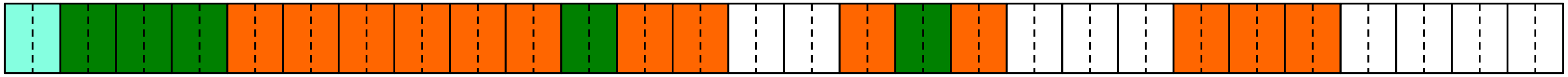
- File data: the actual content of the files
- Meta data: the information required to support the file system.

Meta data contains:

- File names
- Directory structures and symbolic links
- File size and file type
- Creation, modification, last access dates
- Security information (owners, access list, encryption)
- *Links to the LBA where the file content can be located on the disk*

Files

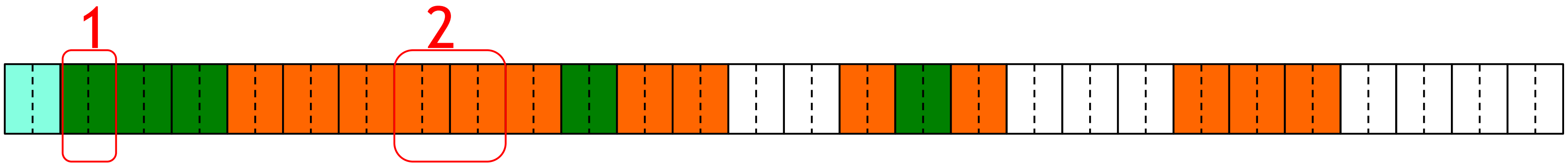
The disk can thus contain several types of clusters:



- Meta data – fixed position (to bootstrap the entire file system)
- Meta data – variable position (to hold the folder structure)
- File data (actual content of the files)
- Unused space (available to contain new files and folders)

Files - reading

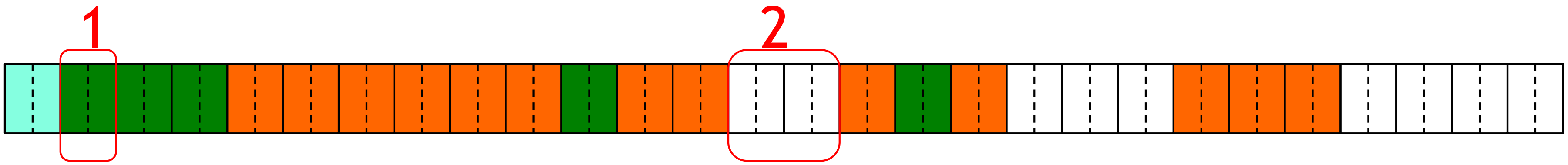
Reading a file requires to:



1. Accessing the meta-data to locate its blocks.
2. Access the blocks to read its content

Files - writing

Writing a file requires to:



1. Accessing the meta-data to locate free space.
2. Write the data in the assigned blocks

Files – writing (2)

Since the file system can only access clusters, the real occupation of space on a disk for a file is always a multiple of the cluster size.

Let us call:

- *s* – the file size
- *c* – the cluster size
- *a* – the actual size on disk

Then, we have:

$$a = ceil(s / c) * c$$

And the quantity ***w = a – s*** is ***wasted disk space*** due to the organization of the file into clusters.

This waste of space is called *internal fragmentation* of files.

Files – writing (2): Example

An example of internal fragmentation:

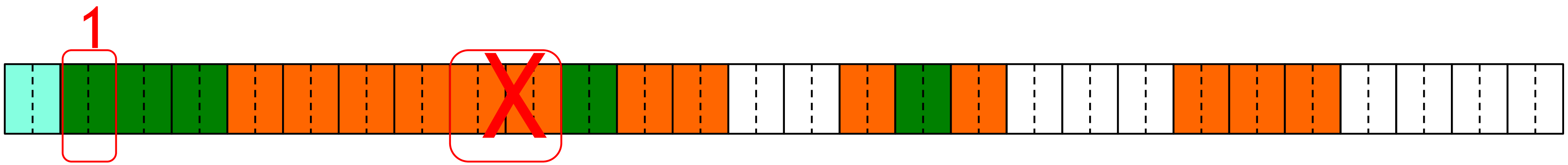
- s - **file size = 27 byte**
- c – **cluster size = 8 byte**
- **actual size on the disk**

*a = ceil(27 / 8) * 8 = ceil(3.375)*8 = 4 *8 =* ***32 byte***

- ***Wasted disk space = 32 – 27 = 5 byte***

Files - deleting

Deleting a file requires:

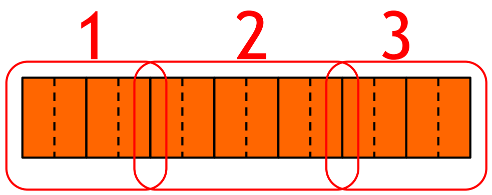


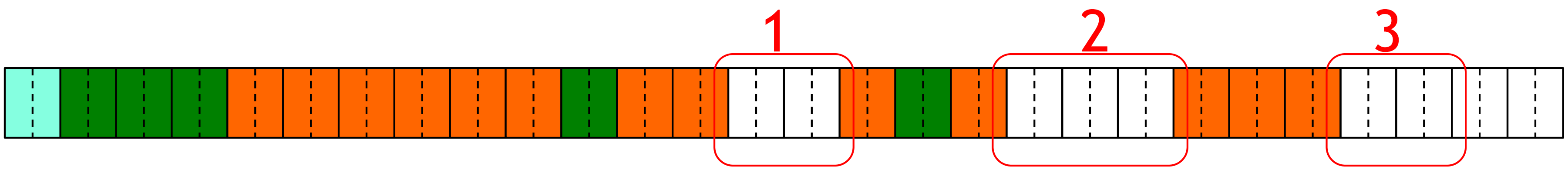
1. Only to update the meta-data to say that the blocks where the file was stored are no longer in use by the O.S.

Deleting a file never actually deletes the data on the disk: when a new file will be written on the same clusters, the old data will be replaced by the new one.

Files – external fragmentation

As the life of the disk evolves, there might not be enough space to store a file contiguously.





In this case, the file is split into smaller chunks that are inserted into the free clusters spread over the disk.

The effect of splitting a file into non-contiguous clusters is called *external fragmentation*.

As we will see, this can reduce a lot the performance of an HDD.

HARD DRIVES

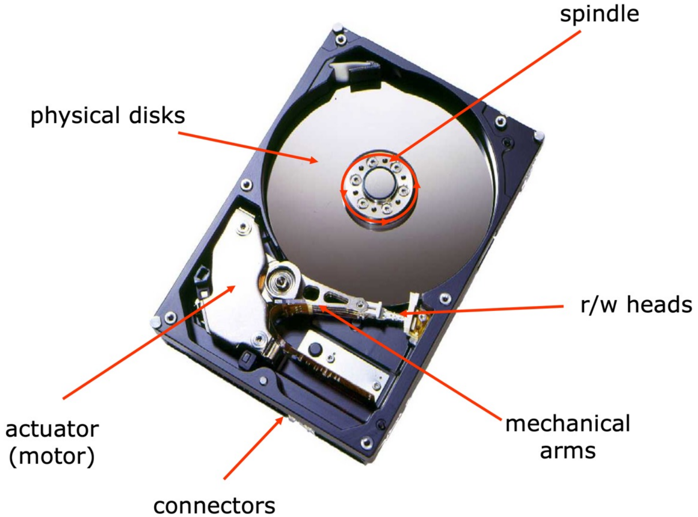
Hard Disk Drives

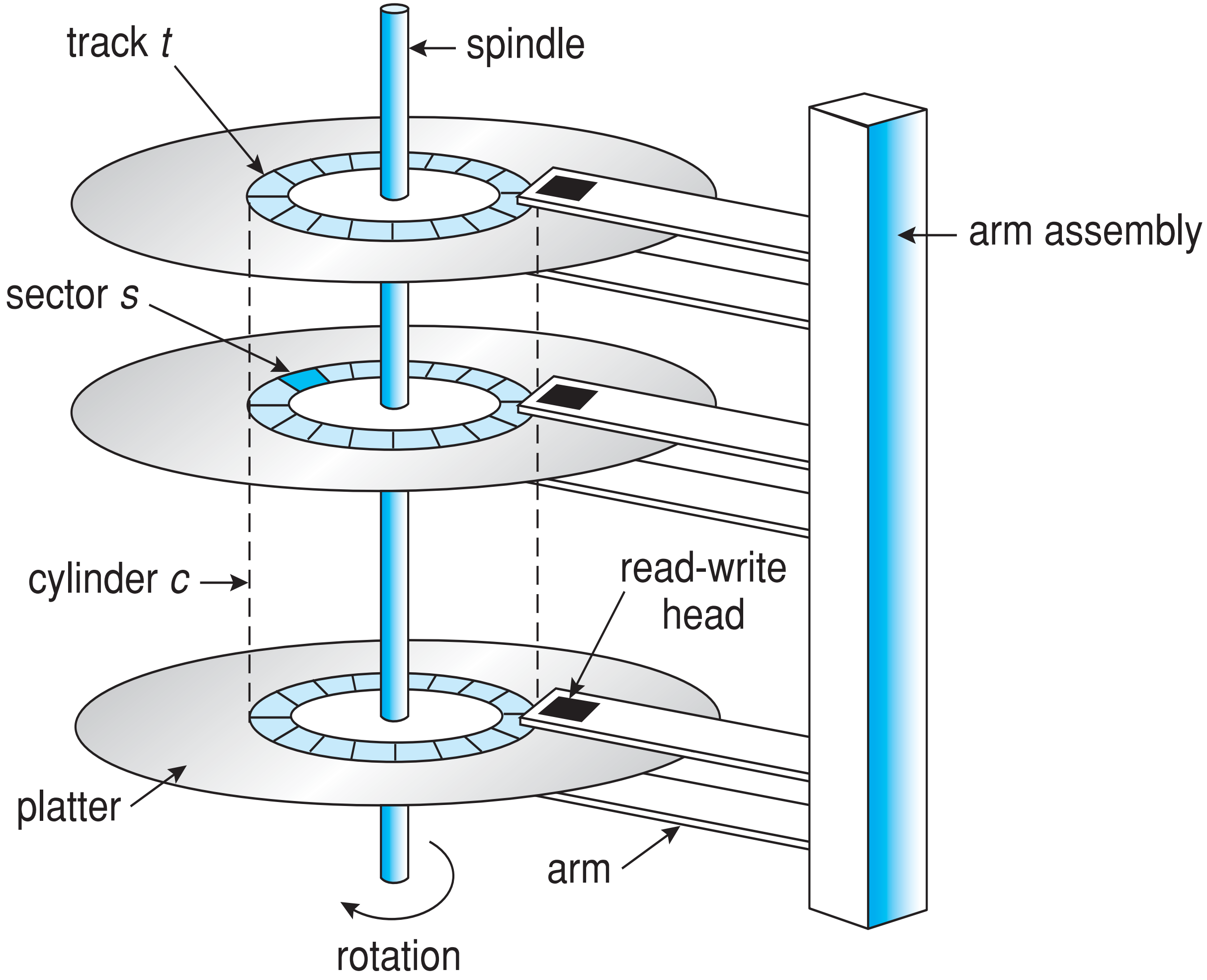
A hard disk drive (HDD) is a data storage using rotating disks (platters) coated with magnetic material.

Data is read in a random-access manner, meaning individual blocks of data can be stored or retrieved in any order rather than sequentially.

An HDD consists of one or more rigid ("hard") rotating disks (platters) with magnetic heads arranged on a moving actuator arm to read and write data to the surfaces.

Hard Drive anatomy: A Multi-Platter Disk

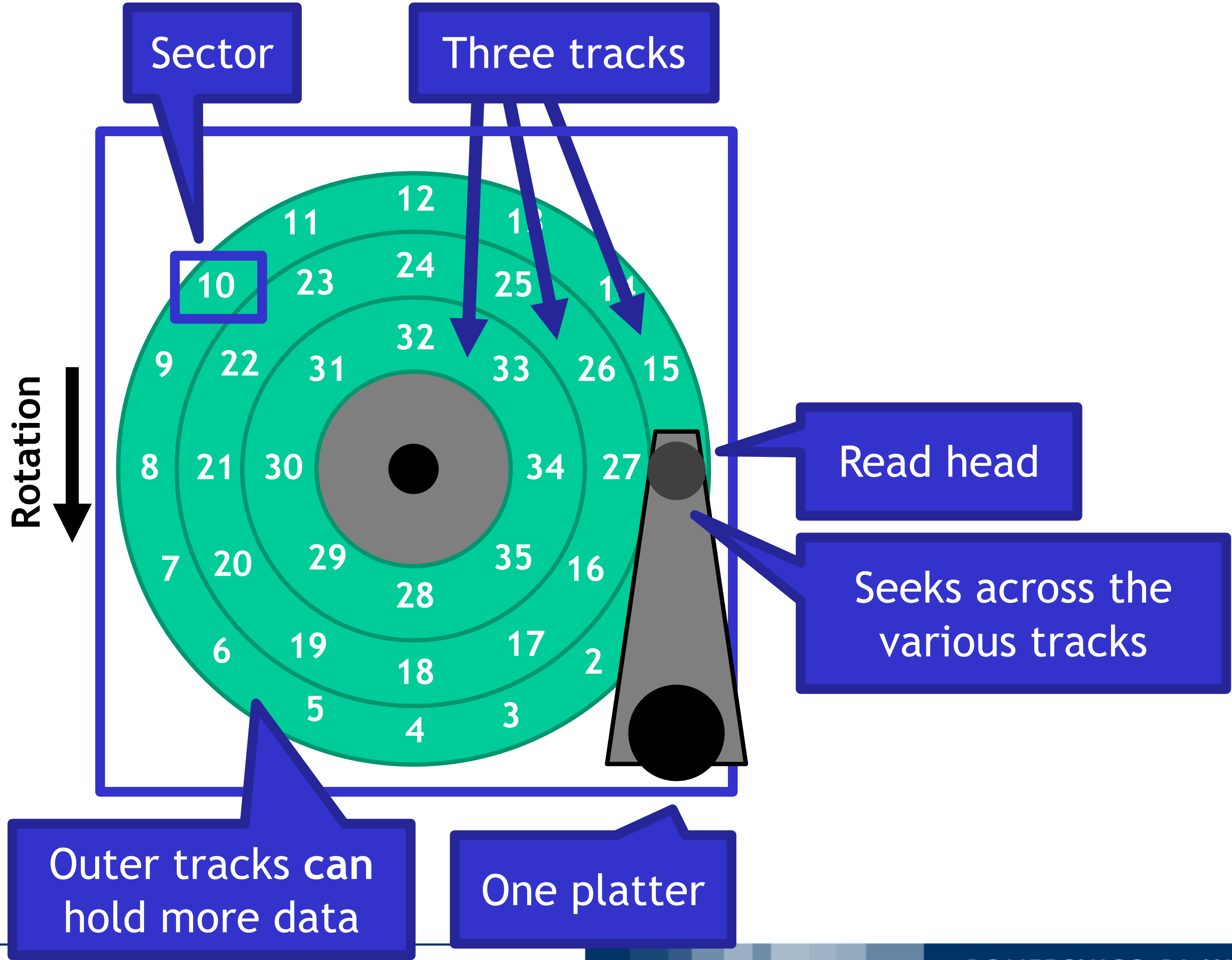


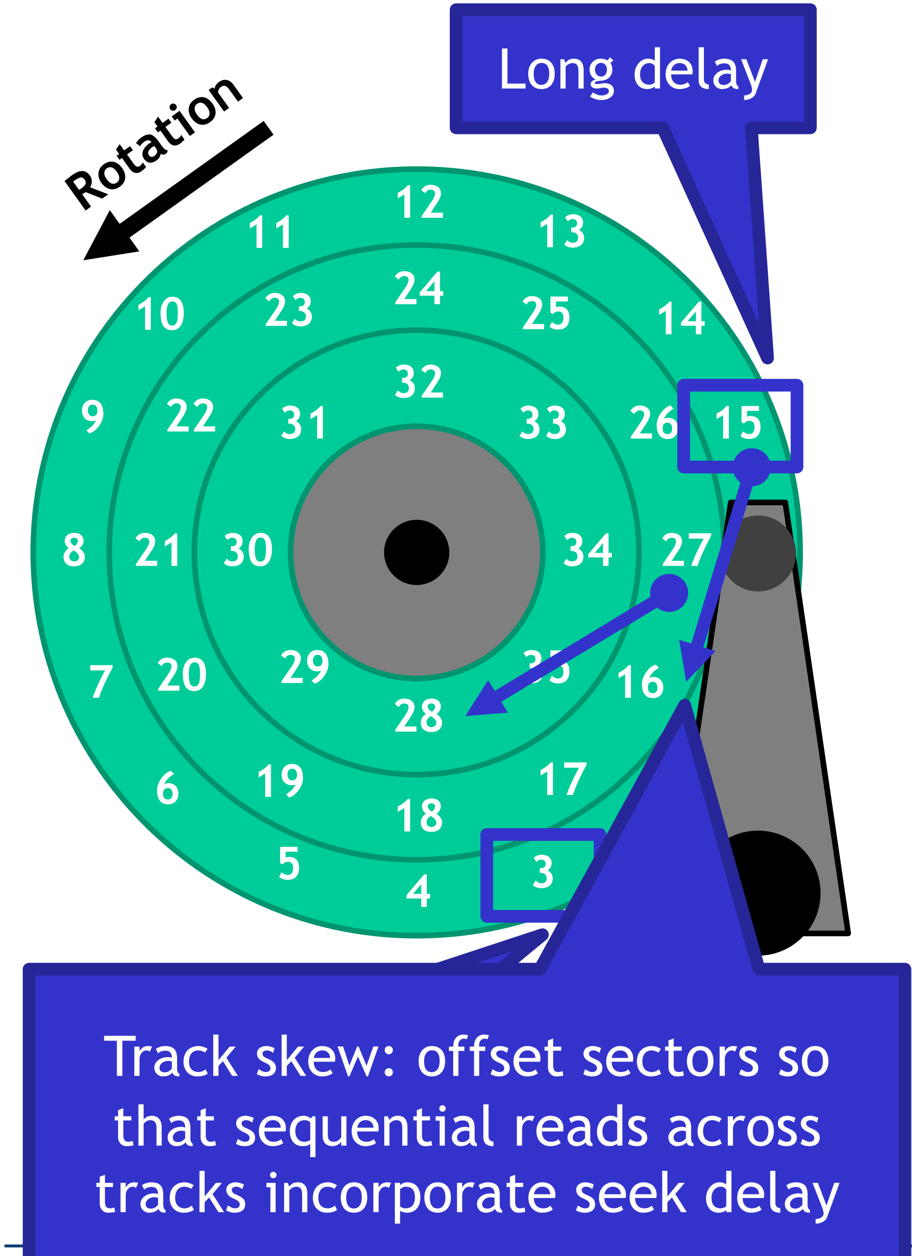


Addressing and Geometry

- Externally, hard drives expose a large number of sectors (blocks)
  - Typically 512 or 4096 bytes
  - Individual sector writes are atomic
  - Multiple sectors writes may be interrupted (torn write)
    - *Torn writes* happens when only part of a multi-sector update are written successfully to disk
- Drive geometry
  - Sectors arranged into tracks
  - A cylinder is a particular track on multiple platters
  - Tracks arranged in concentric circles on platters
  - A disk may have multiple, double-sided platters
- Drive motor spins the platters at a constant rate
  - Measured in revolutions per minute (RPM)

Geometry Example





Four types of delay

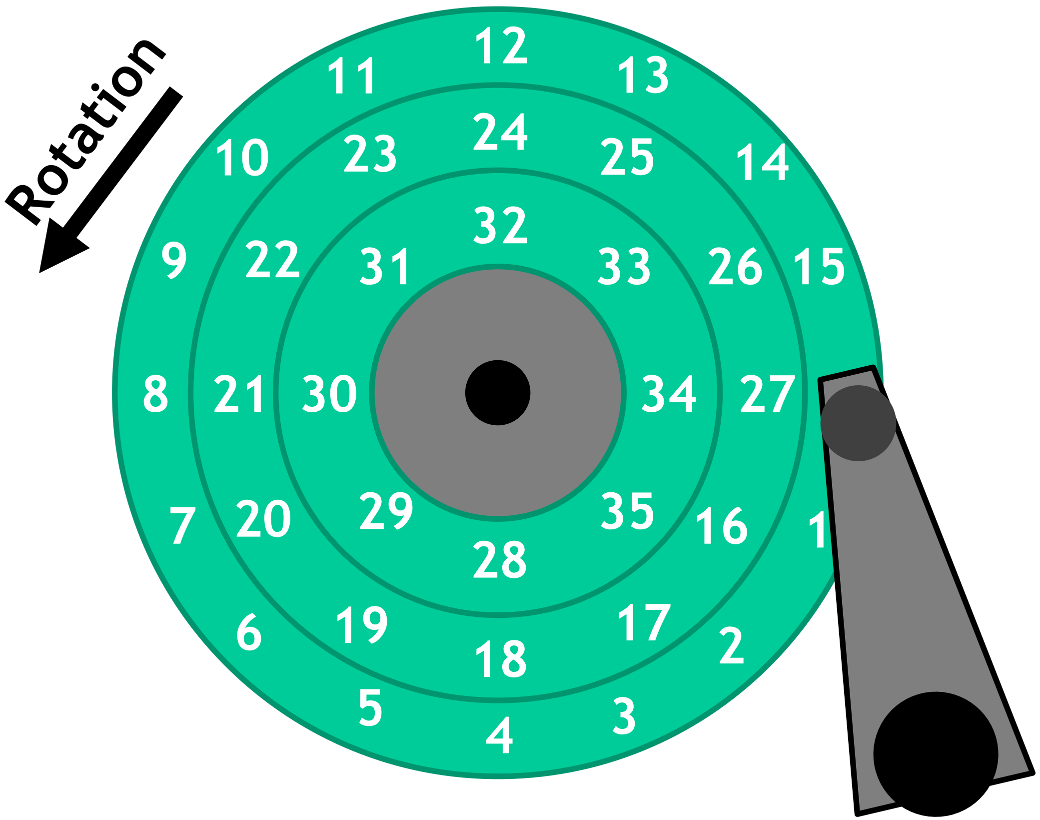
1. Rotational Delay
   - Time to rotate the desired sector to the read head
   - Related to RPM
2. Seek delay
   - Time to move the read head to a different track
3. Transfer time
   - Time to read or write bytes
4. Controller Overhead
   - Overhead for the request management

Rotational Delay

Full rotation delay is R = 1/DiskRPM

- In seconds Rsec = 60*R

$T_{rotation\_AVG}$ = *Rsec/2*



Seek Time

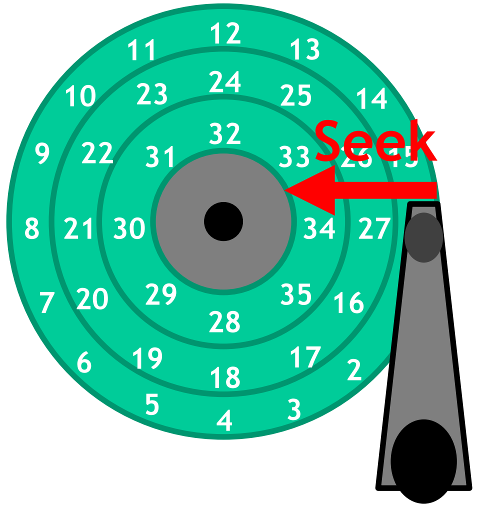
Time to move the head to a different track

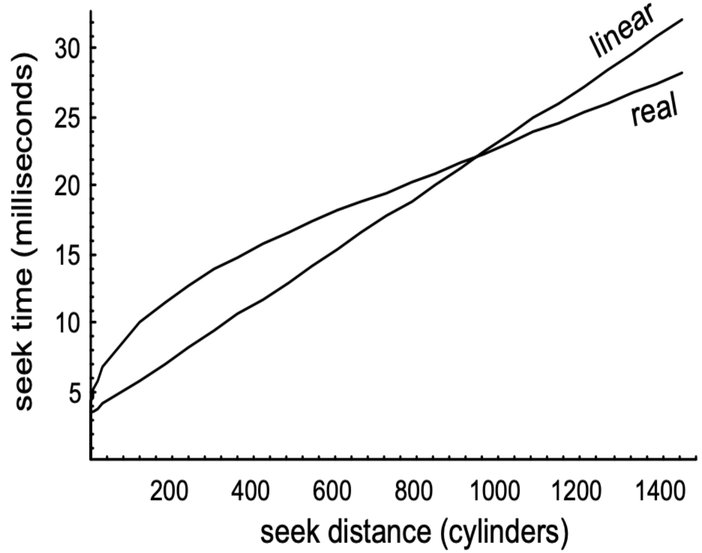
Several Phases:

- Accelleration
- Coasting (constant speed)
- Decelleration
- Settling

T_{seek} modeling consider a linear dependency with the distance

$T_{seek\_AVG} = T_{seek\_MAX} / 3$





Transfer Time and Controller Overhead

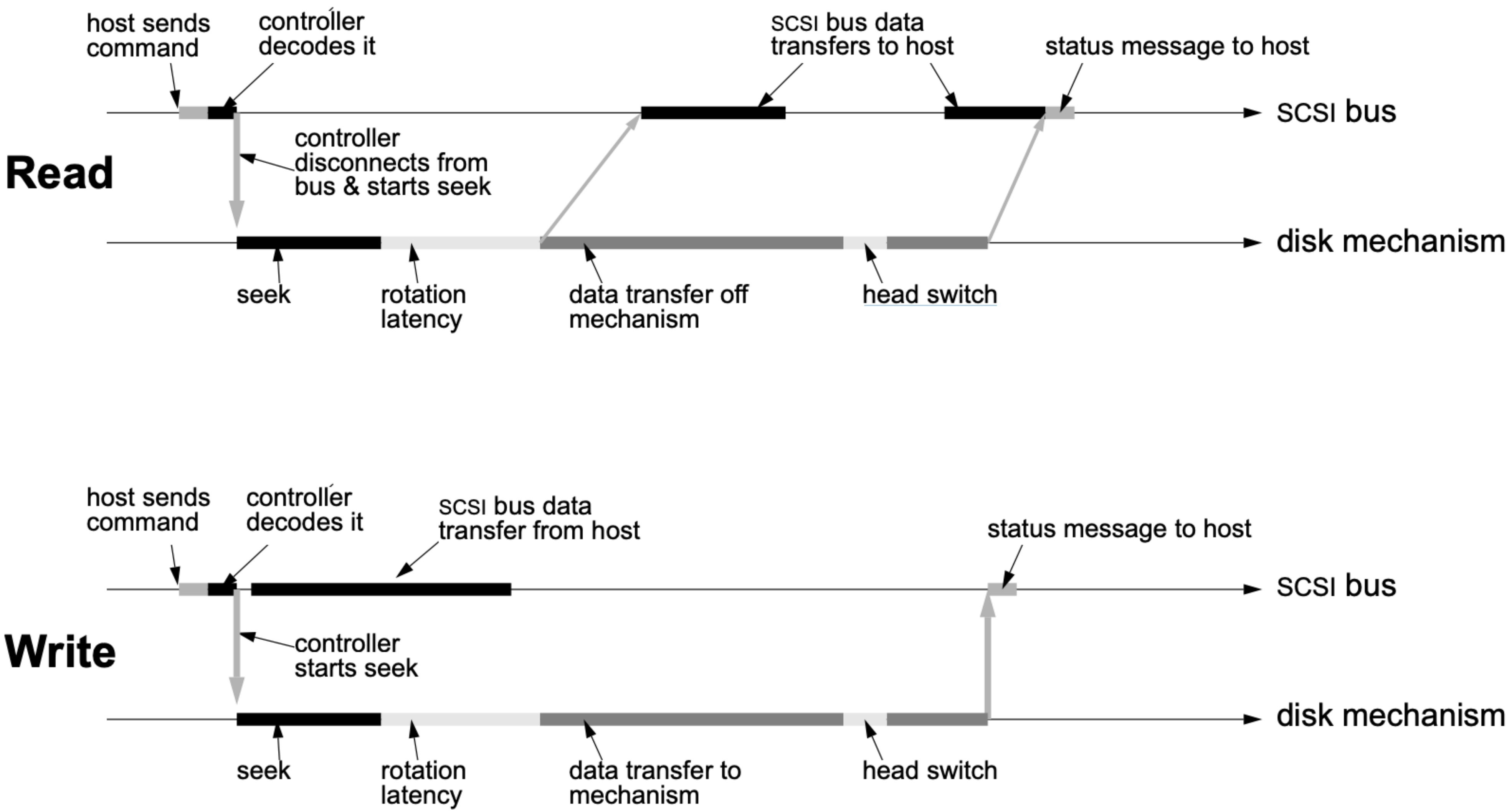
Transfer time

- Final phase of the I/O that takes place
- Time that consider that data is either read from or written to the surface
- Includes the time for the head to pass on the sectors and the I/O transfer
  - rotation speed, storing density

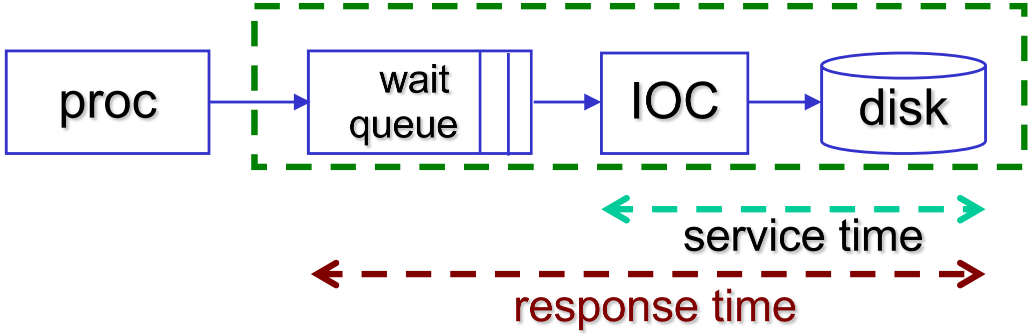
Controller Overhead

- buffer management (data transfer) and interrupt sending time

Read/Write simple protocols



How To Calculate The I/O Service Time for a Disk



Service Time

- $T_{I/O} = T_{seek} + T_{rotation} + T_{transfer} + T_{ovehead}$

Response time

- T_{queue} waiting for the resource + $T_{I/O}$
- Where T_{queue} depends on
  - queue-lenght, resource utilization, mean and variance of disk service time (distribution) and request arrival distribution

Exercise 1: mean service time of a I/O operation

read/write of a sector of 512 Byte = 0.5 KB

- data transfer rate: 50 MB/sec
- rotation speed: 10000 RPM (round per minute)
- mean seek time: 6ms
- overhead controller: 0.2ms

Service time

$$T_{I/O} = T_{seek} + T_{rotation} + T_{transfer} + T_{controller}$$

Exercise 1: mean service time of a I/O operation

read/write of a sector of 512 Byte = 0.5 KB

- data transfer rate: 50 MB/sec
- rotation speed: 10000 RPM (round per minute)
- mean seek time: 6ms
- overhead controller: 0.2ms

mean latency: (60s/min)x1000/(2x10000 rpm) = 3.0ms (time for ½ round)

transfer time: (0.5KB)x1000 / (50x1024KB/s) = 0.01ms

$$\text{mean I/O service time} = \underbrace{6ms}_{\textbf{seek}} + \underbrace{3ms}_{\textbf{latency}} + \underbrace{0.01ms}_{\textbf{transfer}} + \underbrace{0.2ms}_{\textbf{controller}} = 9.21ms$$

Hard Disk Drives

The previous service times considers only the very pessimistic case where sectors are fragmented on the disk in worst possible way

- Files are very small (each file is contained in one block)
- or the disk is very (externally) fragmented

Thus, each access to a sector requires to pay

- Rotational latency, and
- Seek time

In many circumstances, this is not the case:

- files are larger than one block, and
- they are stored in a contiguous way

We can measure the *data locality* of a disk as the percentage of blocks that do not need seek or rotational latency to be found.

Exercise 2: data locality (see the values of exercise 1)

Calculate the average time for read/write a sector of 512 Byte = 0.5 KB

- CONSIDERING A DATA LOCALITY = 75%
- data transfer rate: 50 MB/sec
- rotation speed: 10000 RPM (round per minute)
- mean seek time: 6ms
- overhead controller: 0.2ms

Average Service time

$$T_{I/O} = (1\text{-}DL)*(T_{seek} + T_{rotation}) + T_{transfer} + T_{controller}$$

Exercise 2: data locality (see the values of exercise 1)

Calculate the average time for read/write a sector of 512 Byte = 0.5 KB

- CONSIDERING A DATA LOCALITY = 75%
- data transfer rate: 50 MB/sec
- rotation speed: 10000 RPM (round per minute)
- mean seek time: 6ms
- overhead controller: 0.2ms

  - data locality l=75%: seek+RotLatency affect only 25% of the operations
  - T = (1 – l) * (Ts + Trl) + Tc + Tt

    $(6.0 + (0.5 \times 60 \times 10^{3} / 10000))$ x $0.25 + (0.5\text{ KB} / 50\text{MB} \times 2^{10}) + 0.2$
  - mean time of one I/O op. = $(0.25 \times (6+3)) + 0.01 + 0.2 = 2.46$ ms

Exercise 3: influence of “not optimal” data allocation

time to transfer a file of 1MB

(10 blocks of 1/10 MB “not well” distributed on disk) for each block

(values as in ex.1)

Case A (locality = 100%):

- 1 initial seek: 6 ms
- 1 tot. latency: 3 ms
- 1 global transfer 1 MB: $(1/50) \times 1000 = 20$ ms
- total time: 29 ms

Case B (locality = 0%):

- 1 seek: 6 ms
- 1 rot. latency: 3 ms
- 1 partial transfer (1/10): 2 ms
- total time: $(6 + 3 + 2) \times 10 = 110$ ms (+ 279% !)

(controller times not considered)

Many disks incorporate caches (track buffer)

- Small amount of RAM (8, 16, or 32 MB)

Read caching

- Reduces read delays due to seeking and rotation

Write caching

- Write back cache: drive reports that writes are complete after they have been cached
  - Possibly dangerous feature. Inconsistent state if power goes off before the write back event
- Write through cache: drive reports that writes are complete after they have been written to disk

Today, some disks include flash memory for persistent caching (hybrid drives)

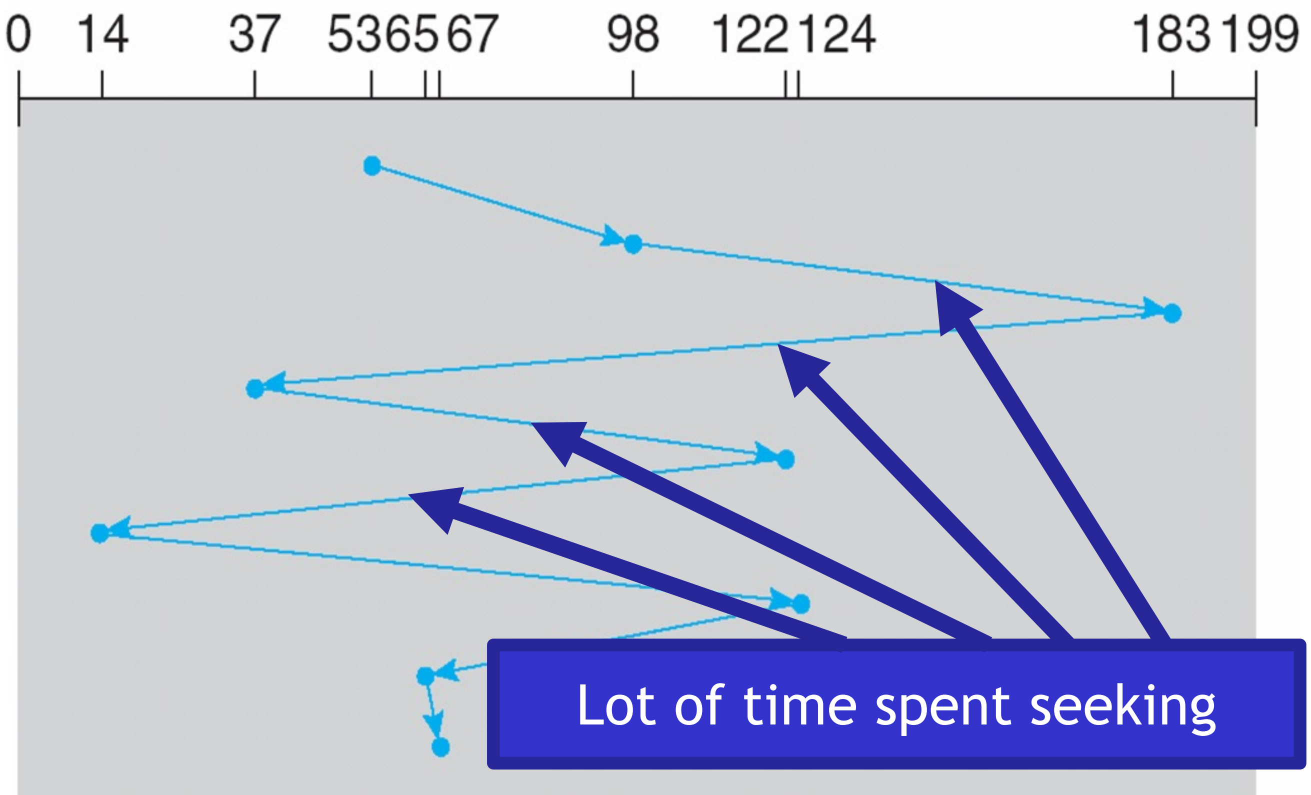
Disk Scheduling

- Caching helps improve disk performance
- But it can't make up for poor random accesses
- Key idea:
  - if there are a queue of requests to the disk, they can be reordered to improve performance
  - Estimation of the request length is feasible knowing the position on the disk of the data
  - Several scheduling algorithms
    - First come, first serve (FCFC)
    - Shortest seek time first (SSTF)
    - SCAN, otherwise know as the elevator algorithm
    - C-SCAN, C-LOOK, etc.

FCFS Scheduling - First come, first serve

- Most basic scheduler, serve requests in order

- Head starts at block 53
- Queue:
  - 98,
  - 183,
  - 37,
  - 122,
  - 14,
  - 124,
  - 65,
  - 67

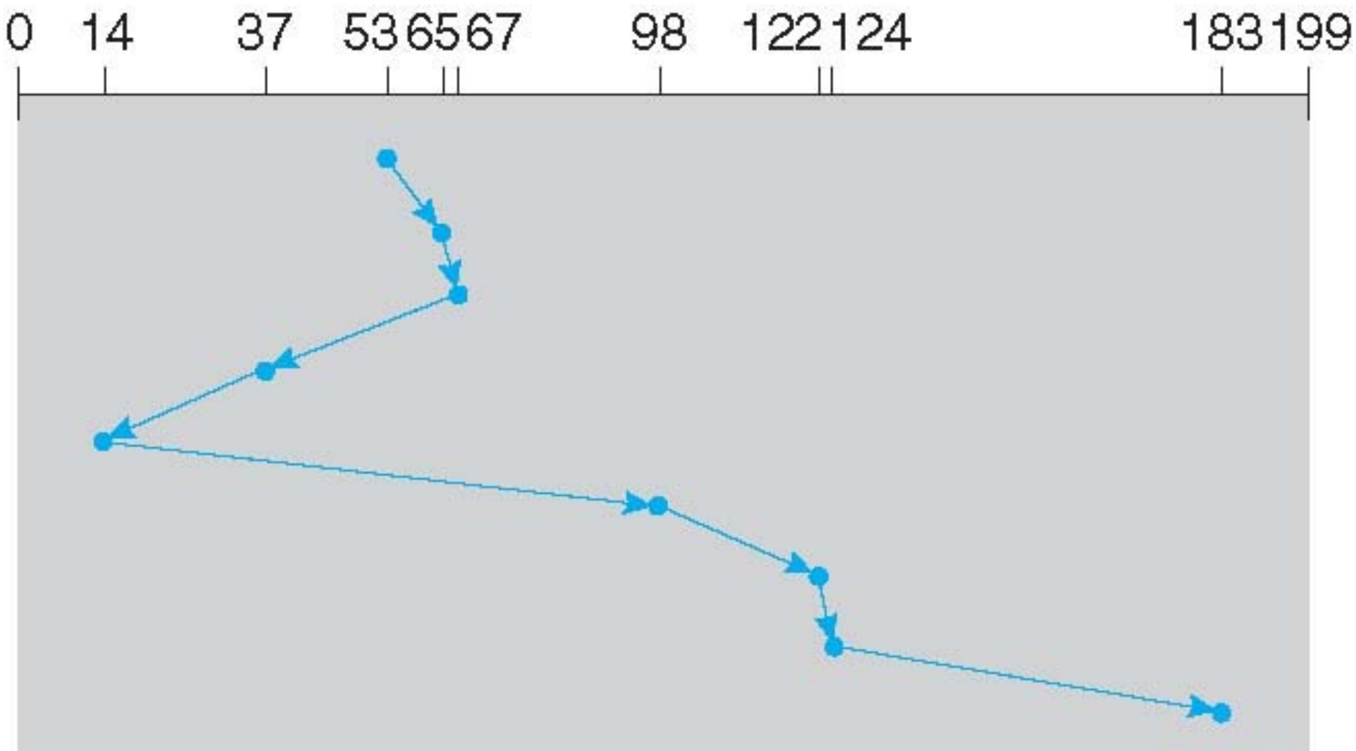


- Total movement: 640 cylinders

SSTF Scheduling - Shortest seek time first

- Idea: minimize seek time by always selecting the block with the shortest seek time

- Head starts at block 53
- Queue:
  - 98,
  - 183,
  - 37,
  - 122,
  - 14,
  - 124,
  - 65,
  - 67

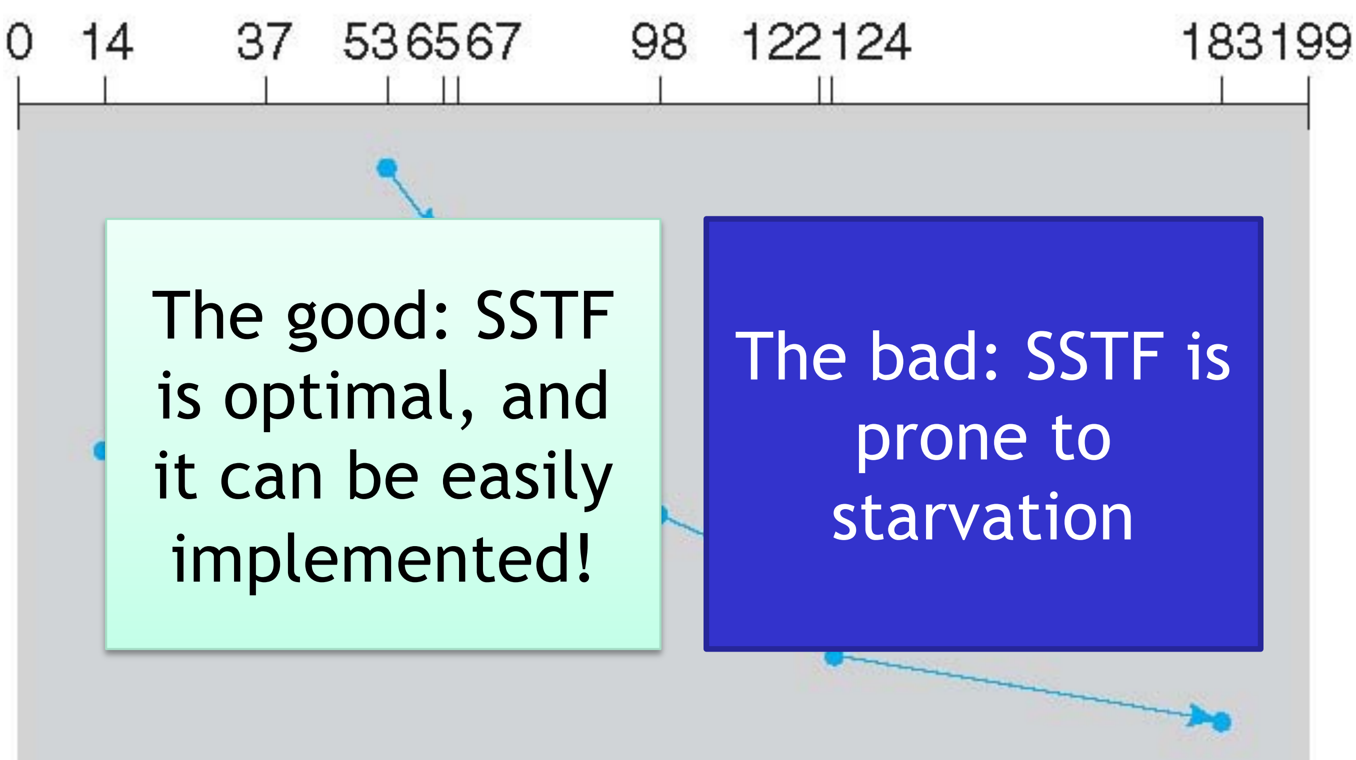


- Total movement: 236 cylinders

SSTF Scheduling - Shortest seek time first

- Idea: minimize seek time by always selecting the block with the shortest seek time

- Head starts at block 53
- Queue:
  - 98,
  - 183,
  - 37,
  - 122,
  - 14,
  - 124,
  - 65,
  - 67



The good: SSTF is optimal, and it can be easily implemented!

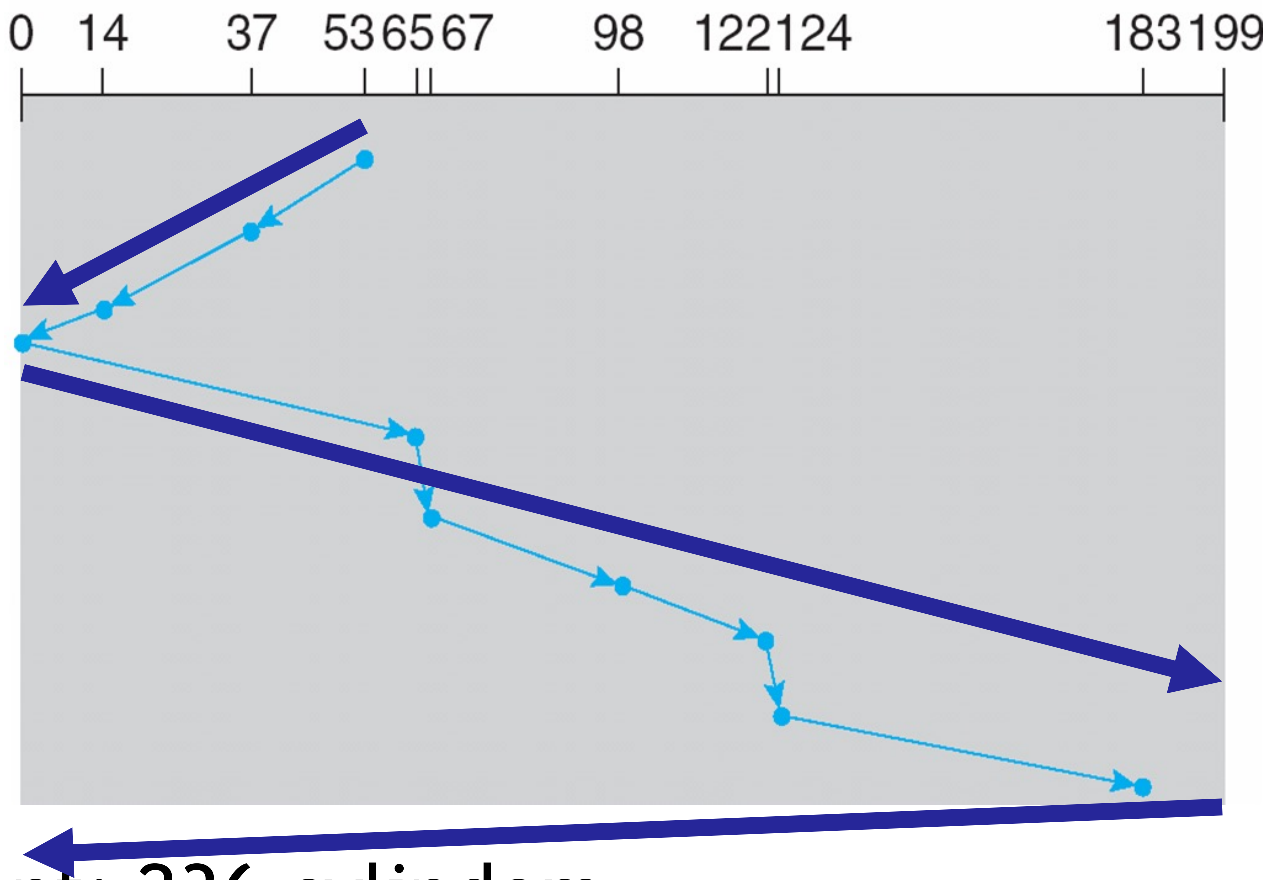
The bad: SSTF is prone to starvation

- Total movement: 236 cylinders

SCAN Example

- Head sweeps across the disk servicing requests in order

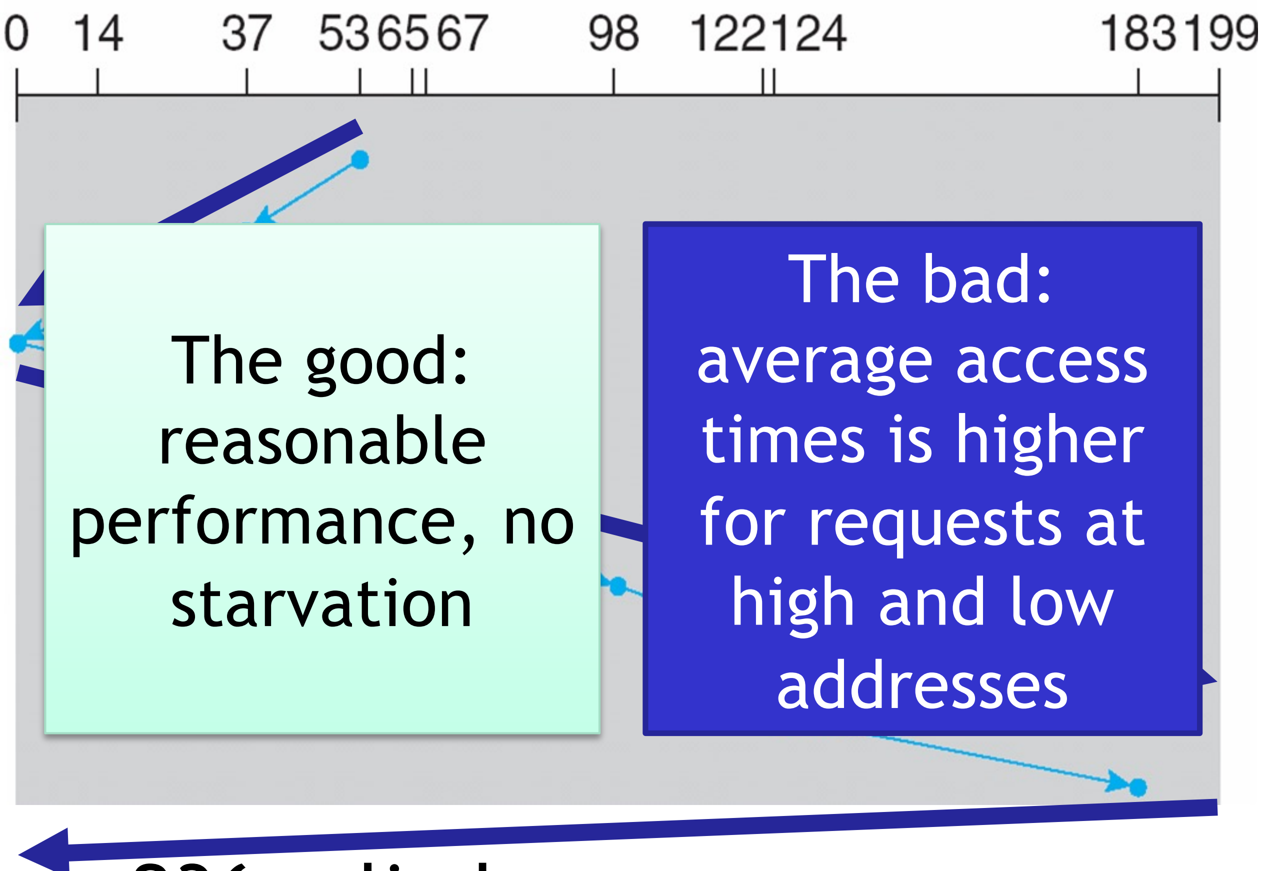
- Head starts at block 53
- Queue:
  - 98,
  - 183,
  - 37,
  - 122,
  - 14,
  - 124,
  - 65,
  - 67



- Total movement: 236 cylinders

- Head sweeps across the disk servicing requests in order

- Head starts at block 53
- Queue:
  - 98,
  - 183,
  - 37,
  - 122,
  - 14,
  - 124,
  - 65,
  - 67



The good: reasonable performance, no starvation

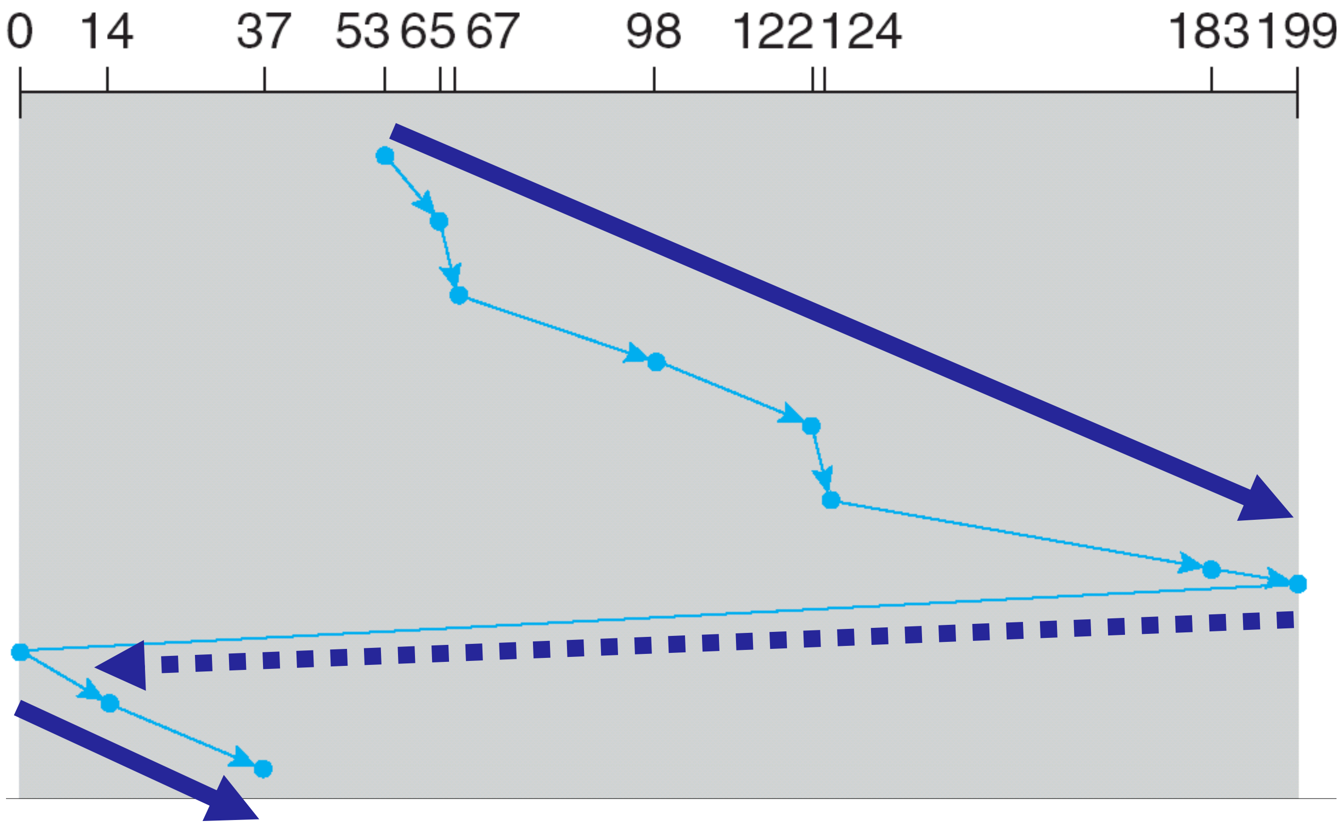
The bad: average access times is higher for requests at high and low addresses

- Total movement: 236 cylinders

C-SCAN Example

- Like SCAN, but only service requests in one direction (Circular SCAN)

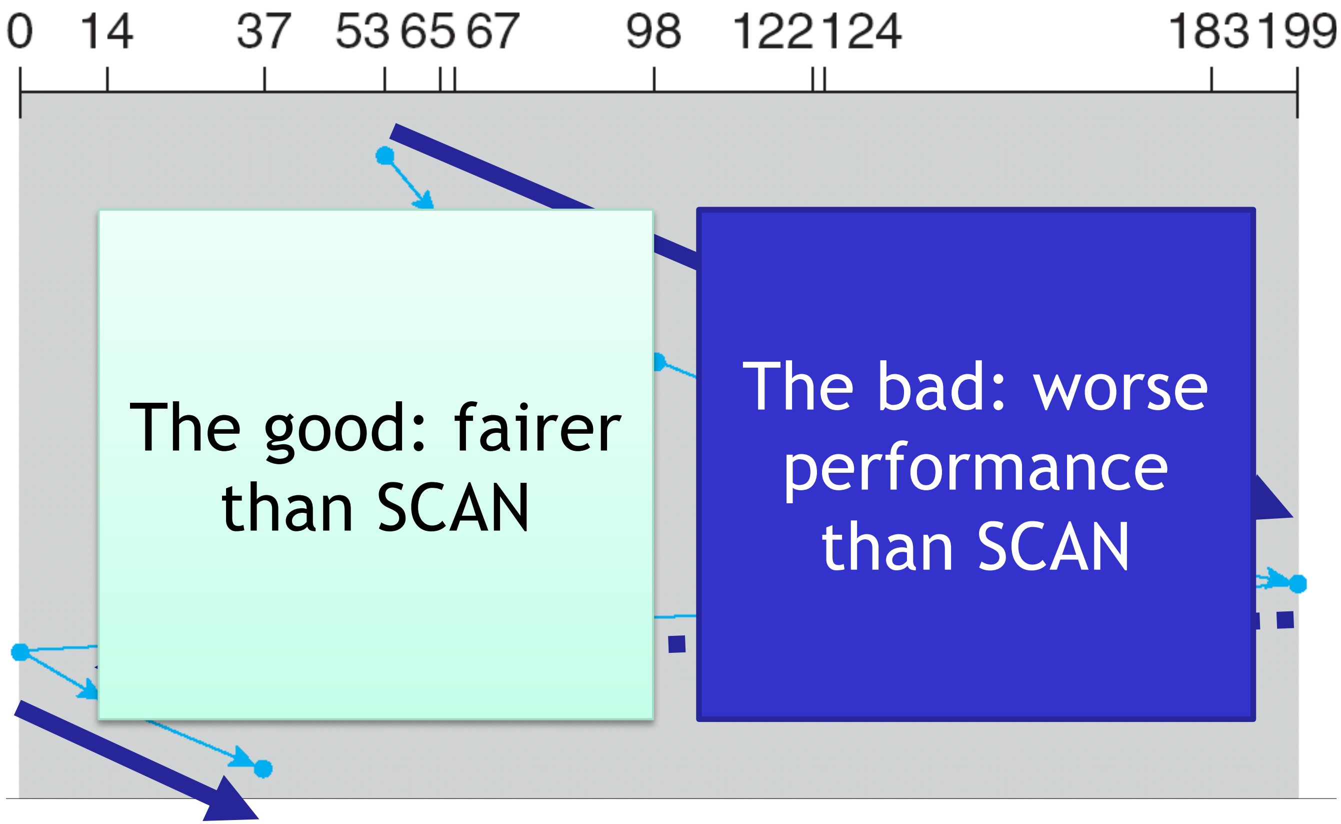
- Head starts at block 53
- Queue:
  - 98,
  - 183,
  - 37,
  - 122,
  - 14,
  - 124,
  - 65,
  - 67



- Total movement: 382 cylinders

- Like SCAN, but only service requests in one direction (Circular SCAN)

- Head starts at block 53
- Queue:
  - 98,
  - 183,
  - 37,
  - 122,
  - 14,
  - 124,
  - 65,
  - 67

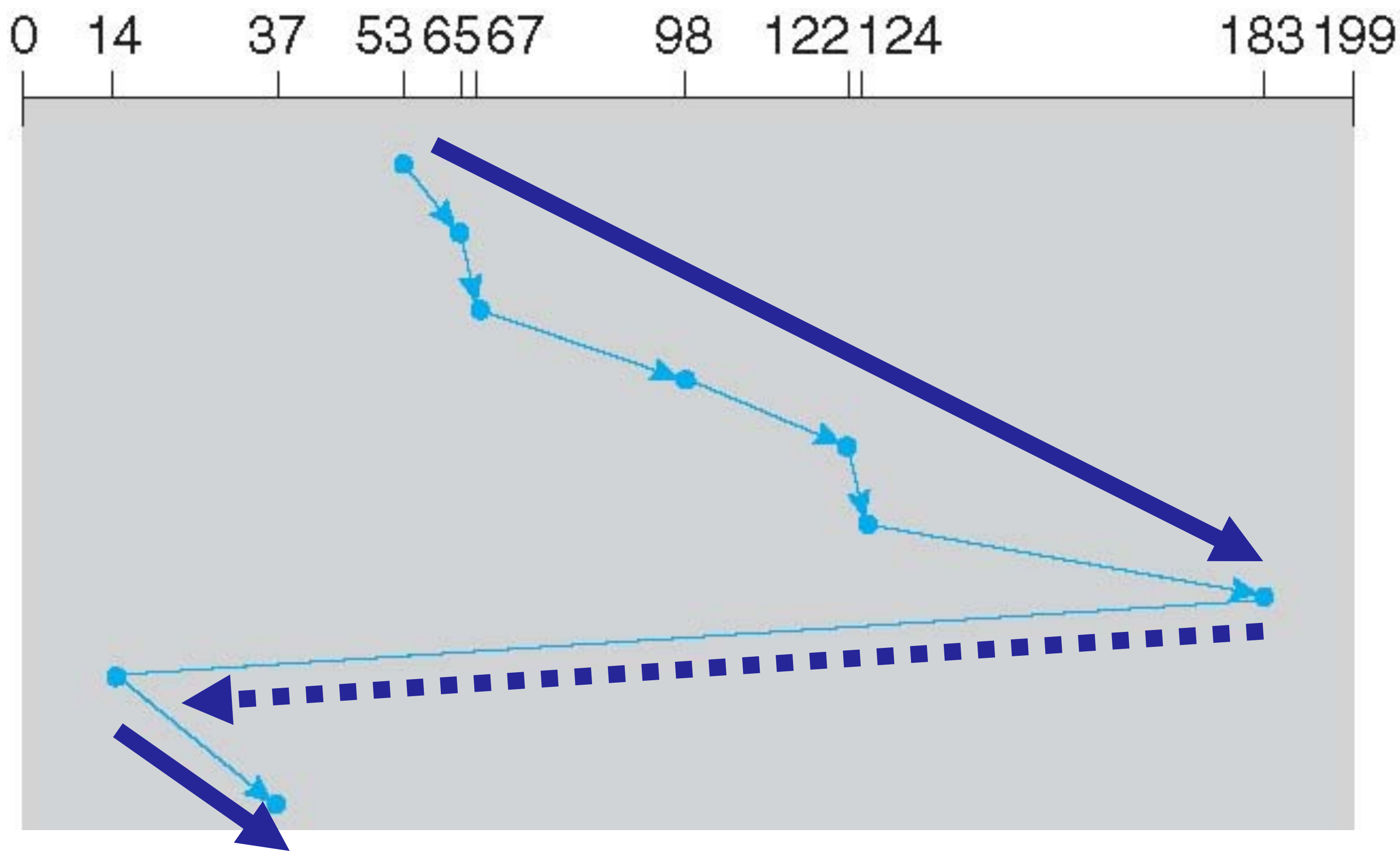


- Total movement: 382 cylinders

C-LOOK Example

- C-SCAN Variant that peeks at the upcoming addresses in the queue
  - Head only goes as far as the last request

- Head starts at block 53
- Queue:
  - 98,
  - 183,
  - 37,
  - 122,
  - 14,
  - 124,
  - 65,
  - 67



- Total movement: 322 cylinders

Implementing Disk Scheduling

Where should disk scheduling be implemented? OS or DISK?

- OS scheduling
  - Requests re-ordering by LBA
  - However, the OS cannot account for rotation delay
- On-disk scheduling
  - Disk knows the exact position of the head and platters
  - Can implement more advanced schedulers
  - But, requires specialized hardware and drivers
- Disk Command Queue
  - Available in all modern disks
  - Queue where a disk stores pending read/write requests
    - Called Native Command Queuing (NCQ)
  - Disk may reorder items in the queue to improve performance
- Joint OS & on-disk scheduling can bring to problems
  - E.g. "NCQ vs. I/O Scheduler: Preventing Unexpected Misbehaviors "