CPSC 457

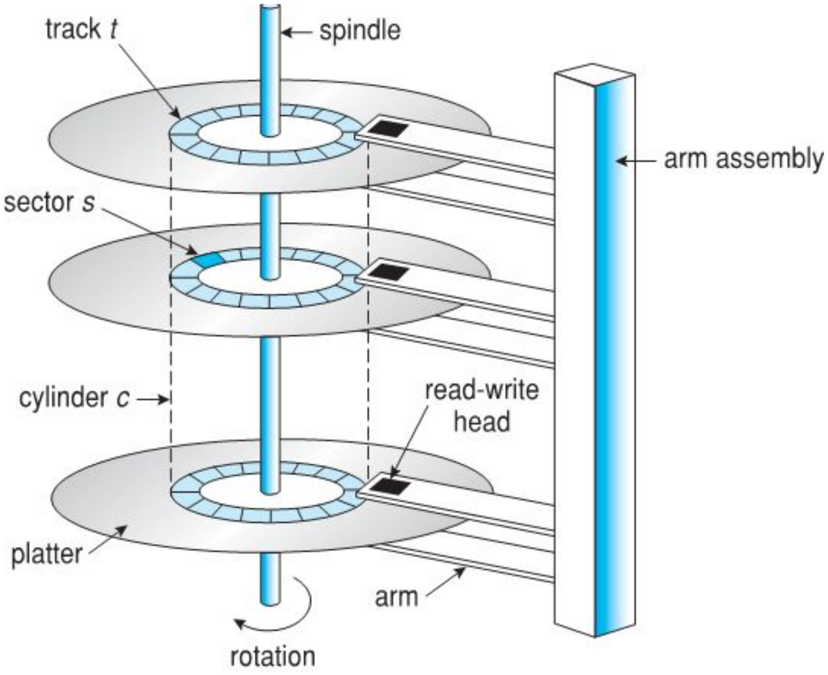
Disks, scheduling, RAID

Contains slides from Mea Wang, Andrew Tanenbaum and Herbert Bos, Silberschatz, Galvin and Gagne

Overview

- disk structure
- disk scheduling
- RAID
- I/O hardware - block and character devices

Magnetic disks

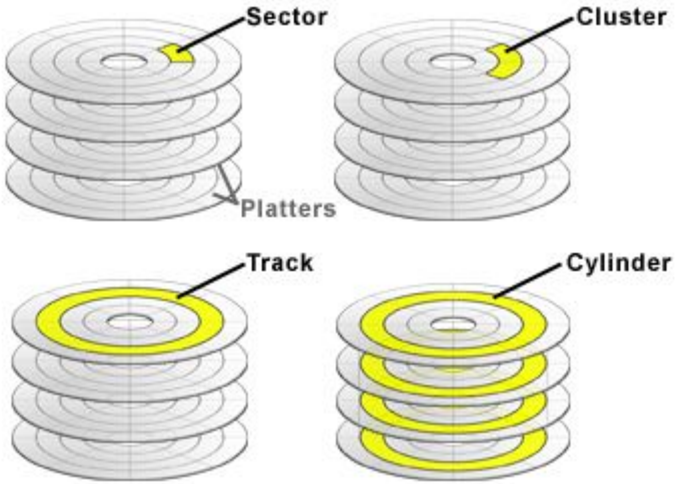


Physical description:

- each disk platter has a flat circular shape (1.8” < diameter < 5.25”)
- platters rotate (5,400 - 15,000 RPMs)
- the read-write heads *fly* just above the surface of each platter
- head crash: the head makes contact with the disk surface, causing permanent damage to the disk
- each head is attached to a disk arm that moves all heads at the same time

Disk space

every track is divided into a bunch of sector } → if we want a bigger chunk we divide the track into clusters



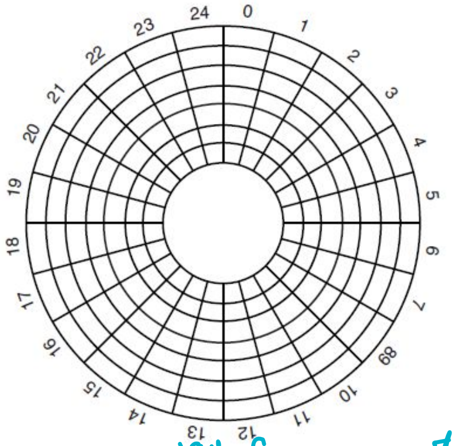
Logical representation:

- the surface of a platter is logically divided into circular tracks
- each track is further divided into sectors
- the set of tracks that are at the same arm position make up a cylinder

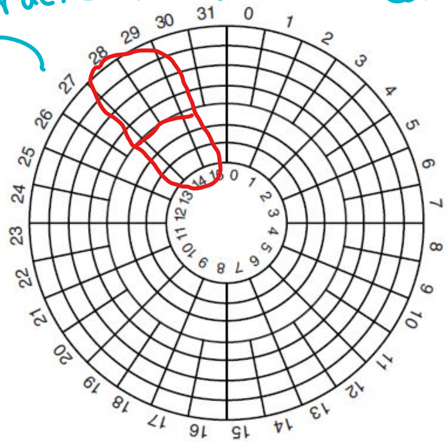
Mapping

- a logical block is the smallest unit of transfer between the disk and the memory, e.g., 512 bytes
- software accesses data on disks only using `write(block #)` and `read(block #)`
- mapping: converting a logical block number into physical disk address that consists of a cylinder number, a head number, and a sector number
- the sectors on disk are mapped to large one-dimensional arrays of logical blocks, numbered consecutively
- on modern disks this mapping is done by an embedded controller because geometry is quite complicated

Comes with circuitry to translate that



modern drive move tracks away from center



Low level format

as a programmer dont have access to these low level bits

- low-level format or physical format: writes low level information to the disk, dividing it into series of tracks, each containing some number of sectors, with small gaps between the sectors
- components of a sector:

Preamble	Data	ECC

a disk sector

Not all sectors are used up 100%.

  - preamble: starts with a special bit sequence, cylinder number, sector number, etc.
  - data: depends on the format (eg. 512 bytes)
  - error correction code - redundant information to detect read errors
- the formatted capacity is about 20% lower than the unformatted capacity

Disk management

high level format → can only be done if it has been low level format

- in order to use a disk to hold files, the OS needs to record its own data structures on the disk
  - partition the disk into one or more regions, each treated as a logical disk
  - logical formatting or "making a file system" on a partition
    - abstracting blocks into files and directories
- OS can allow raw disk access for applications that want to do their own block management, and want to keep OS out of the way (databases for example)
- methods such as sector sparing can be used to handle bad blocks
  - either at OS level, or at lower level

wen format drive the drive can save some sectors unuresed and when the drive detects that some sectors are getting corrupt it can move the data to one of those saved sectors that are not corrupt

Disk scheduling

introduces the biggest latency time

- the time required for reading or writing a disk block is determined by several factors
- the most important one, and the one we'll focus on is the seek time
  - seek time — the time to move the arm holding the heads to the correct cylinder
- other factors that we'll not discuss include:
  - rotational delay — the time for the correct sector to rotate under the head
  - disk bandwidth — the actual data transfer rate
    - calculated for a set of requests
    - total bytes transferred divided by the total time taken to service all requests

Disk scheduling

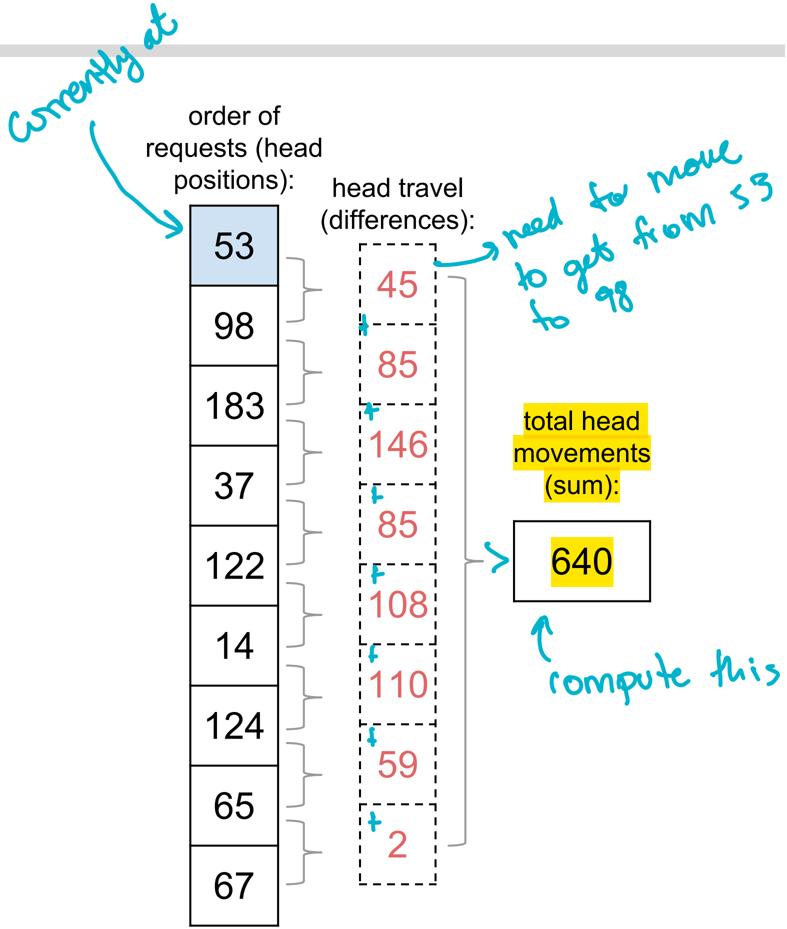
- the requests for disk I/O are appended to the **disk queue**
- OS maintains separate queues of requests for each disk
- OS can improve the overall I/O performance by reordering disk I/O requests, with the goal of minimizing the total head movement
- we will look at 6 different algorithms:
  - FCFS scheduling → not efficient because of the slow seek time
  - SSTF scheduling
  - elevator scheduling
    - SCAN, C-SCAN, LOOK, C-LOOK

FCFS scheduling

- First-Come-First-Served scheduling
- requests are processed in the same order they are received
- FCFS is intrinsically fair
- but it generally does not provide the fastest overall service
- example:

  head starts at cylinder 53

  queue = 98, 183, 37, 122, 14, 124, 65, 67



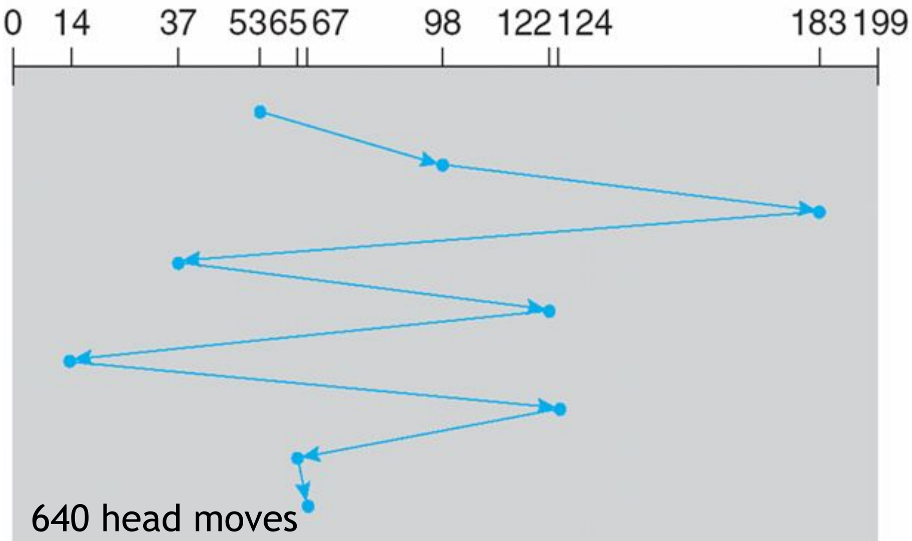
FCFS scheduling

INEfficient

- Example:

  queue = 98, 183, 37, 122, 14, 124, 65, 67

  head starts at 53



SSTF scheduling

STARVATION

- Shortest-Seek-Time-First
- selects the next request that would result in the shortest seek time from the current head position, i.e. picks the 'closest' request next
- seek time = distance to move the heads
- may cause starvation of some requests
- Example:

  head starts at 53

  queue = 98, 183, 37, 122, 14, 124, 65, 67

order of requests (head positions): 53, 65, 67, 37, 14, 98, 122, 124, 183

head travel (differences): 12, 2, 30, 23, 84, 24, 2, 59

total head movements (sum): 236

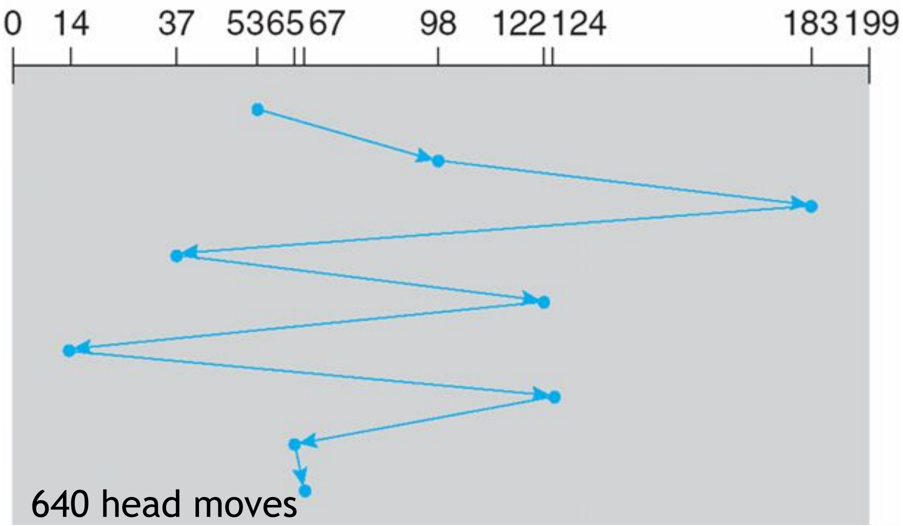
SSTF scheduling

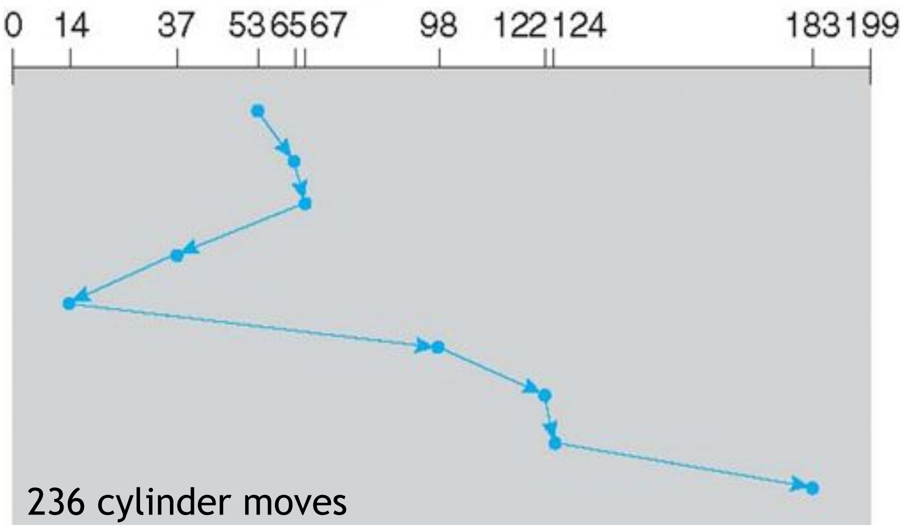
- Example:

  head starts at 53

  queue = 98, 183, 37, 122, 14, 124, 65, 67

FCFS	SSTF





SCAN (elevator) scheduling

Not fair to requests at either ends of the disk

like a train going through its stops

- the head continuously scans back and forth across the disk and serves the requests as it reaches each cylinder
- head moves all the way to first/last cylinder before turning back
- requests at either end tend to wait the longest
- Example:

  head starts at 53, direction is downwards

  queue = 98, 183, 37, 122, 14, 124, 65, 67

down direction

change direction

up direction

order of requests (head positions):	head travel (differences):
53	
37	16
14	23
0	14
65	65
67	2
98	31
122	24
124	2
183	59

total head movements (sum):
236

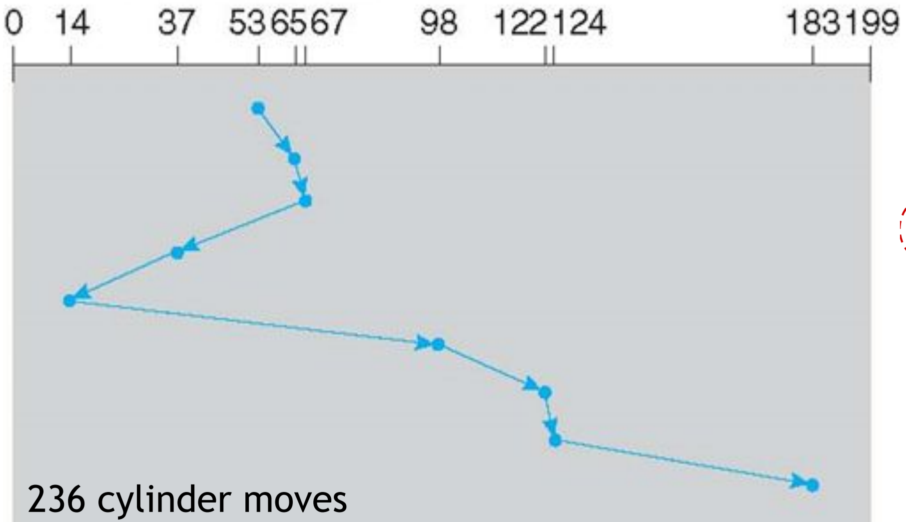
SCAN (elevator) scheduling

- Example:

  head starts at 53, direction is downwards

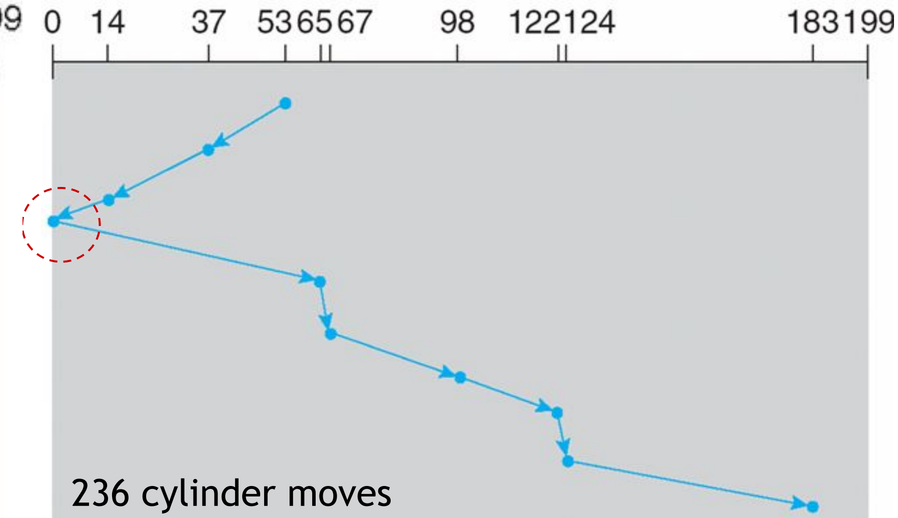
  queue = 98, 183, 37, 122, 14, 124, 65, 67

SSTF



236 cylinder moves

SCAN



236 cylinder moves

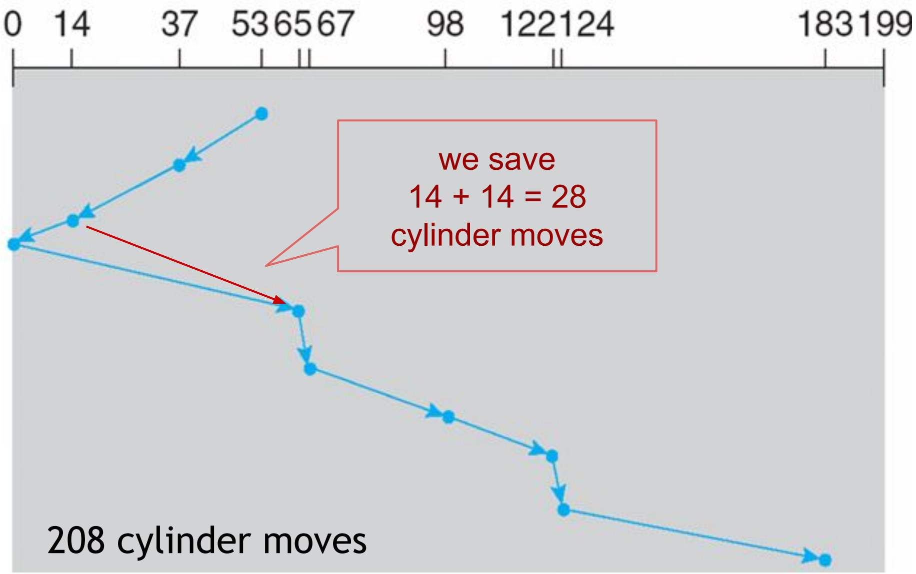
LOOK scheduling

only goes as far as required

- nearly identical to SCAN, but head does not move all the way to first/last cylinder before turning back
- instead it only goes as far as necessary
- results in the same request order as SCAN, but less overall head movement
- Example:

  queue = 98, 183, 37, 122, 14, 124, 65, 67

  head starts at 53, direction is downwards



C-SCAN scheduling

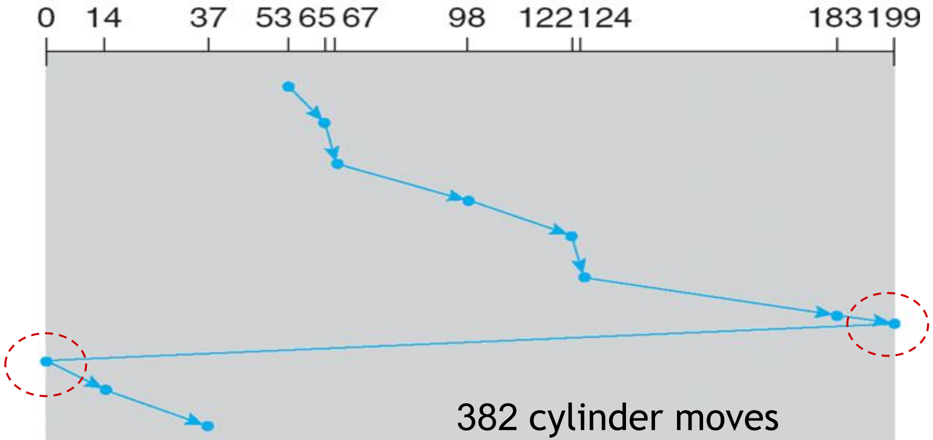
when you reach the ends of the disk so move the head all the way to the other end without ~~too~~ servicing any requests

- same as SCAN in one direction
- but after reaching last cylinder, head repositions to the first cylinder, and no requests are processed during this time
- achieves more uniform wait time than SCAN
- Example:

  queue = 98, 183, 37, 122, 14, 124, 65, 67

  head starts at 53, direction is upwards

fixes unfairness



order of requests (head positions):	head travel (differences):
53	
65	12
67	2
98	31
122	24
124	2
183	59
199	16
0	199
14	14
37	13

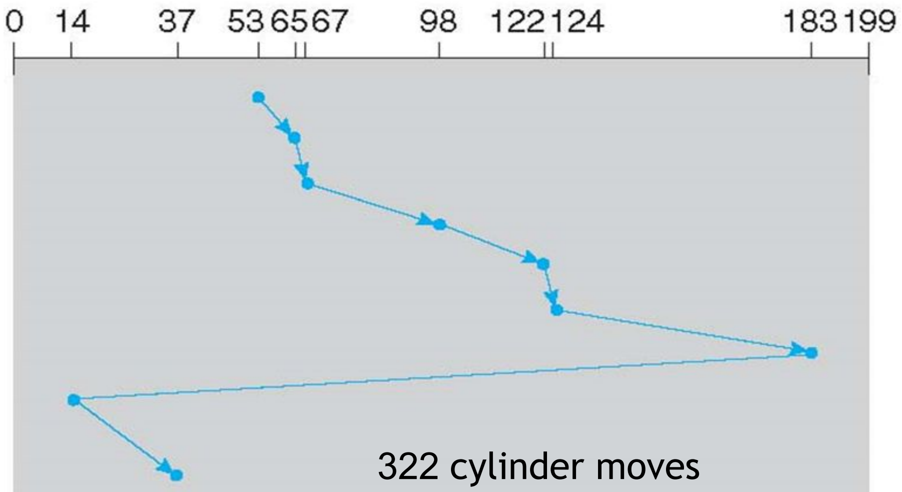
total head movements (sum): 382

C-LOOK scheduling

Only goes as far as it needs to

- small optimization of C-SCAN, head only goes as far as needed by the next request (same optimization as SCAN → LOOK)
- Example:
  queue = 98, 183, 37, 122, 14, 124, 65, 67
  head starts at 53, direction is upwards

* fixes unfairness and adds small optimization *



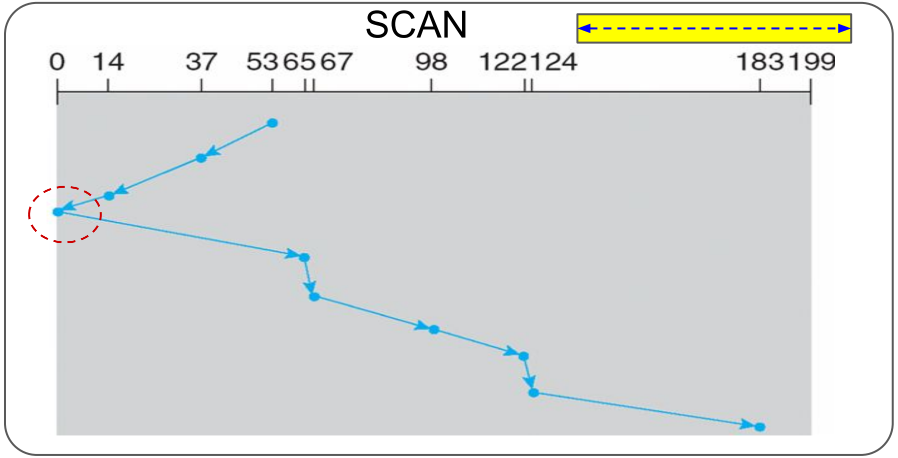
order of requests (head positions):
53
65
67
98
122
124
183
14
37

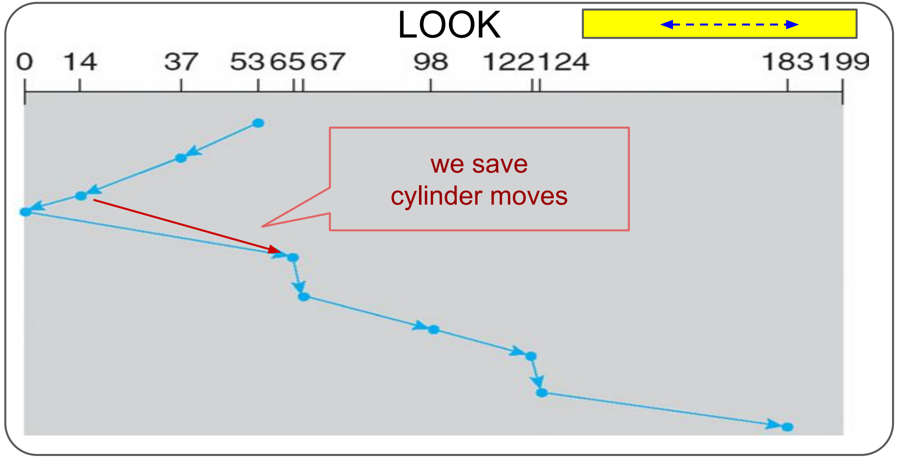
head travel (differences):
12
2
31
24
2
59
169
23

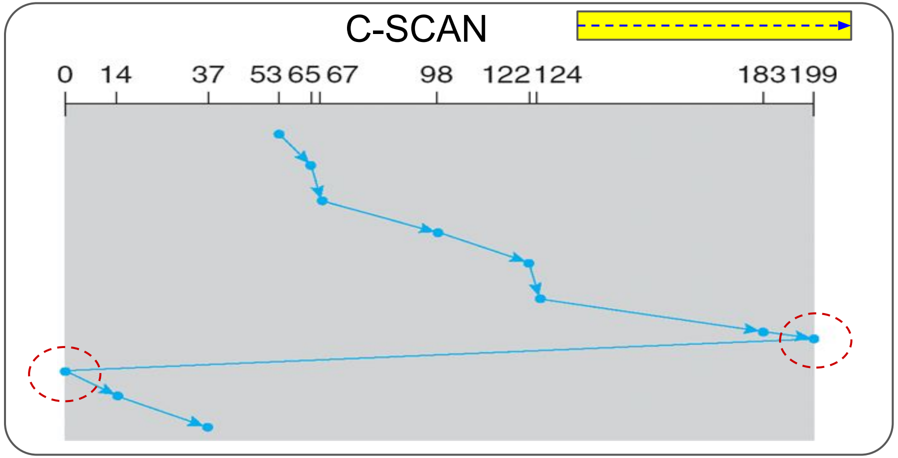
total head movements (sum):
322

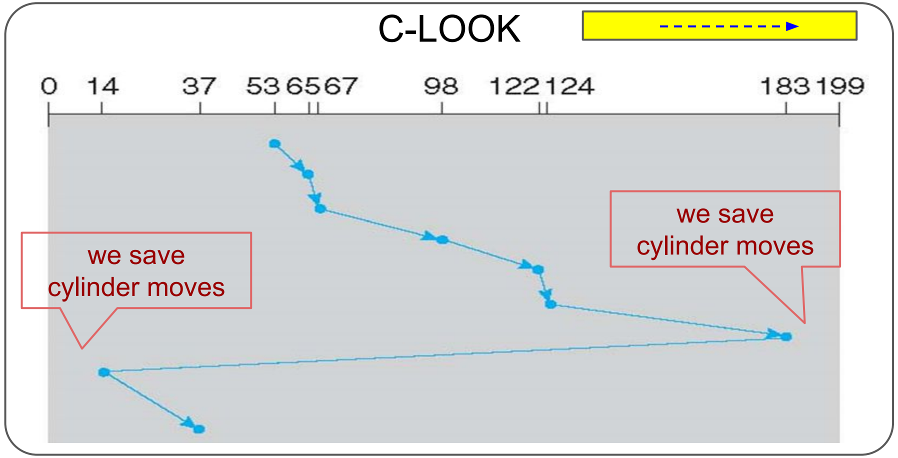
Elevator scheduling

ON THE FINAL (LABELING)









Disk scheduling

- the performance of a scheduling algorithm depends on:
  - the number and types of requests
  - the file-allocation method
  - the location of directories and index blocks
- either SSTF or LOOK is a reasonable choice for a default algorithm
  - C-LOOK if we need more consistent wait times ⤳ Basler
- other scheduling algorithms also consider:
  - rotational latency
  - priority of the task - requests belonging to higher priority process receive higher priority

    eg. requests related to demand paging should receive higher priority
  - prioritize read over write, since read requests usually block processes
  - examples: completely fair queuing (CFQ) & deadline scheduler on Linux

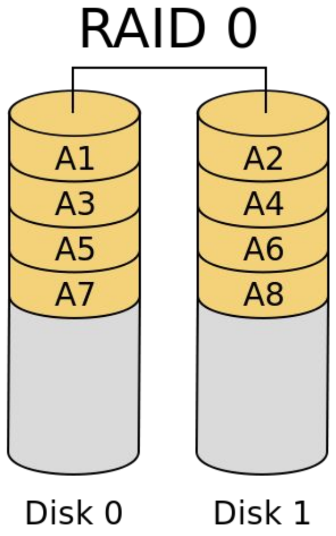
RAID

independent

- RAID – redundant array of ~~inexpensive~~ disks
- multiple disk drives provide reliability via redundancy, increasing the mean time to failure
- can also improve performance through parallelization of requests
- accessed as one big disk (increased capacity)
- can be implemented via dedicated hardware, or in software, or a combination
- can think of it as an abstraction of multiple disks, presented as a single disk (opposite of partitioning)

RAID 0 – striped volume

- uses a group of disks as one unit
- purpose: highest performance for read & write
- consecutive logical blocks distributed across all disks,
  ideally contents of every file are evenly distributed over all disks
- offers no redundancy — a single disk failure leads to entire RAID failure
  actually reduces reliability
- with N disks, read & write performance can be up to
  N times higher than with a single disk, because both
  read & write requests can be parallelized
- often used for high-performance temporary storage,
  where data loss is tolerable, eg:
  for storing temporary data, `/tmp` or `/scratch`

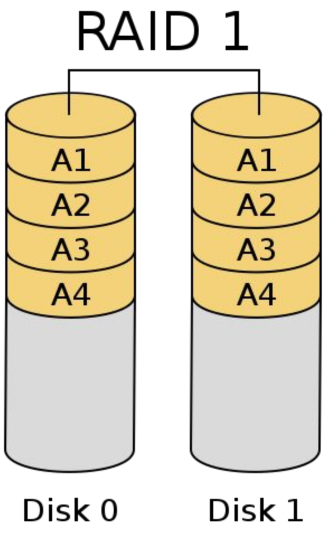


Images from:
https://en.wikipedia.org/wiki/Standard_RAID_levels

RAID 1 – mirrored disks

- keeps 1 or more duplicates of a disk
- purpose: very high reliability & fast read performance
- with N disks, it is tolerant to N-1 simultaneous disk failures
    - RAID continues to work in degraded mode
    - RAID software usually notifies the operator
    - failed disk can be removed & rebuilt from the surviving disks
- with N disks, read performance can be up to

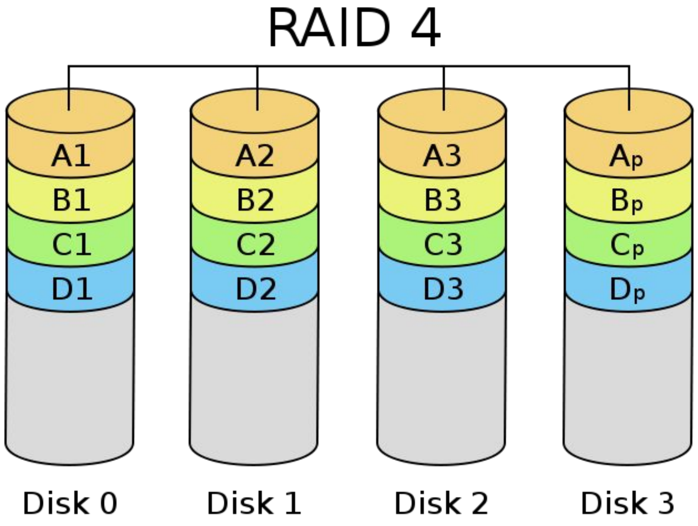
  N times higher than with a single disk
- write performance is that of a single disk
- with N disks, only 1 disk worth of space used to store data!!!



Images from:
https://en.wikipedia.org/wiki/Standard_RAID_levels

RAID 4 – striping with dedicated parity

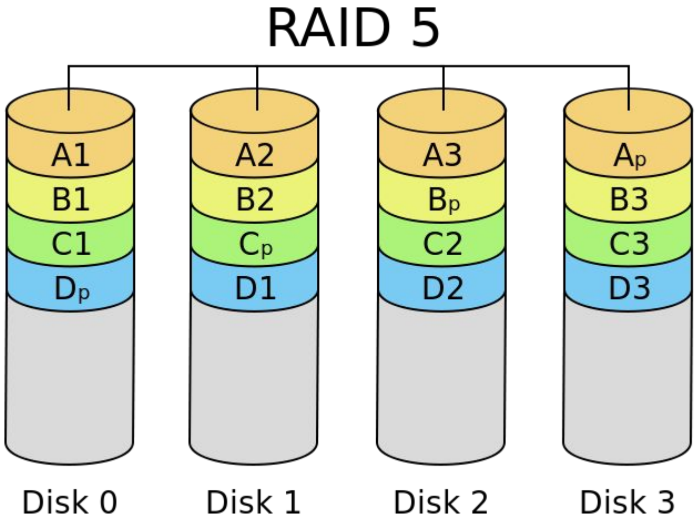
- one disk dedicated to contain parity information, computed eg. using XOR
- purpose: reliability & fast read performance
- tolerant of a single disk failure
- with N disks, only N-1 are used for data
- not very common
  - write is slow, since parity disk is a bottleneck
  - parity disk also wears out faster than the other disks in the array



Images from:
https://en.wikipedia.org/wiki/Standard_RAID_levels

RAID 5 – striping with distributed parity

- similar to RAID 4, but parity information is distributed among all disks
- purpose: reliability, fast read and write performance, although not as fast as RAID 0
- tolerant of a single disk failure
- with N disks, only N-1 space is used for data
- quite common when we need both performance and basic redundancy

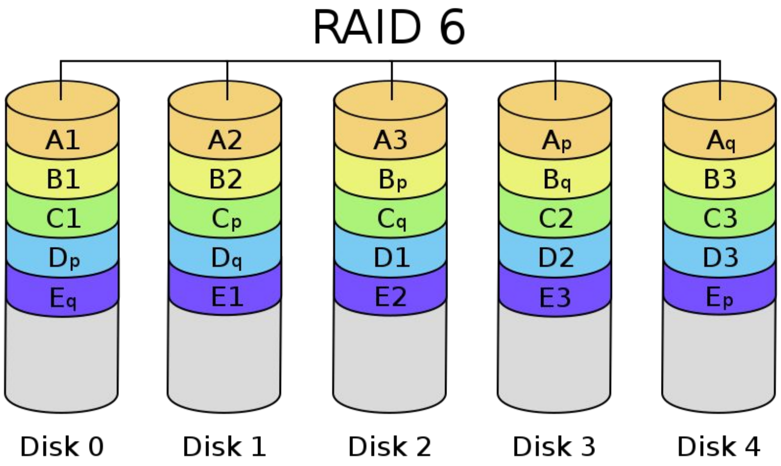


Images from: https://en.wikipedia.org/wiki/Standard_RAID_levels

RAID 6 – striping with double distributed parity

- similar to RAID 5, but doubles the amount of parity (parity computation more complicated)
- purpose: reliability, fast read/write performance
- tolerant of 2 simultaneous disk failures
- with N disks, only N-2 space is used for data
- usage: same as RAID 5, but data is very important

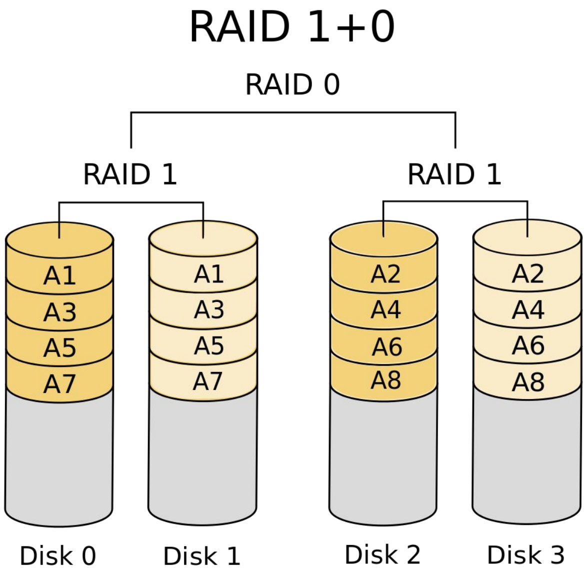
- what about RAID 2 and RAID 3?
  - not used, obsolete



Images from:
https://en.wikipedia.org/wiki/Standard_RAID_levels

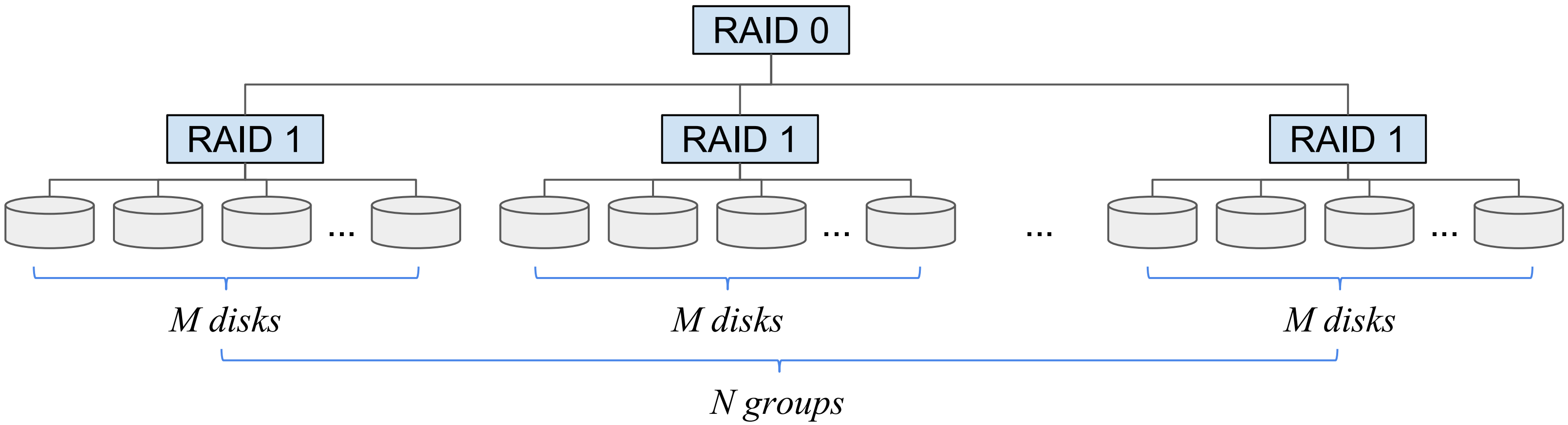
RAID 1+0 – striped mirrors

- aka RAID 10, is an example of a hybrid/nested RAID
  - nests RAID 1 in RAID 0 configuration
  - simplest form: 4 disks, 2 groups of 2
- purpose: very fast & very reliable
  - combines advantages of RAID 0 and RAID 1
- in simplest form (4 disks), it can survive at least 1 disk failure, and if lucky 2 failures
- common for high-performance uses where data cannot be lost, eg. databases, email server
- can tune redundancy to 3, 4, 5 … simultaneous failures



Images from:
https://en.wikipedia.org/wiki/Nested_RAID_levels

RAID 10 – striped mirrors

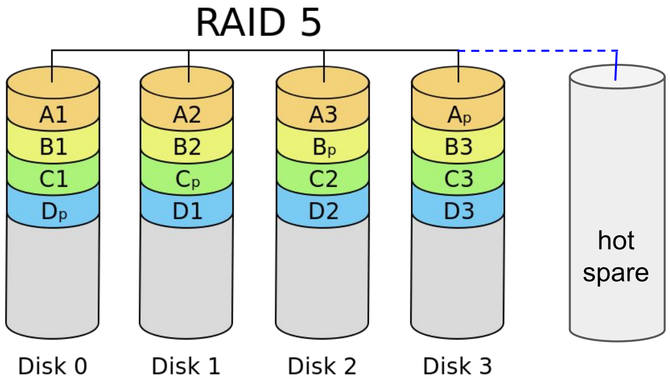


- consider RAID 10 that has N groups of RAID 1, and each group has M disks

  i.e. total number disks = M * N
- can survive at least M - 1 simultaneous disk failures, but potentially up to N*(M-1) failures
- read performance potentially up to N * M of a single disk, write performance is N times higher
- only N disks worth of space used for data out of N * M, so it is a very expensive RAID configuration
- note: other nested RAIDs are also possible,eg. RAID 5+0, RAID 6+0, RAID 10+0

Hot Spares

- a small number of hot-spare disks can be left unallocated
- these automatically replace a failed disk
  - data is rebuilt onto them
  - time spent in degraded mode is minimized
- hot-spares are not used until failure occurs
- can be added to any RAID that supports redundancy



Images from:
https://en.wikipedia.org/wiki/Standard_RAID_levels

I/O Devices

- block devices:
  - store information in fixed-size blocks (eg.512 bytes to 32KB)
  - each block has its own address
  - data transferred in units of one or more entire blocks
  - read or write can be done in any order
  - e.g., hard disk, CD-ROMs, USB
- character devices:
  - delivers or accepts a stream of characters, without regard to any block structure
  - not addressable, and no seek operations
  - e.g., printer, network interface, mouse, keyboard
- other devices:
  - clocks (also known as timers)

Summary

- disk structure
- disk scheduling
- RAID
- I/O hardware - block and character devices

Reference: 5.1 - 5.4 (Modern Operating Systems)

12.1 - 12.5, 13.1 - 13.4 (Operating System Concepts)

Review

- Which one of the following disk scheduling algorithms could lead to starvation among requests?
  - FCFS
  - SSTF
  - SCAN
- Which RAID configuration cannot survive a single disk failure?
  - RAID 0
  - RAID 1
  - RAID 5
  - RAID 10
- Keyboard is a character device.
  - True or False

Questions?