Operating Systems ICS 431

Week 14

Ch. 12 Mass-Storage Structure

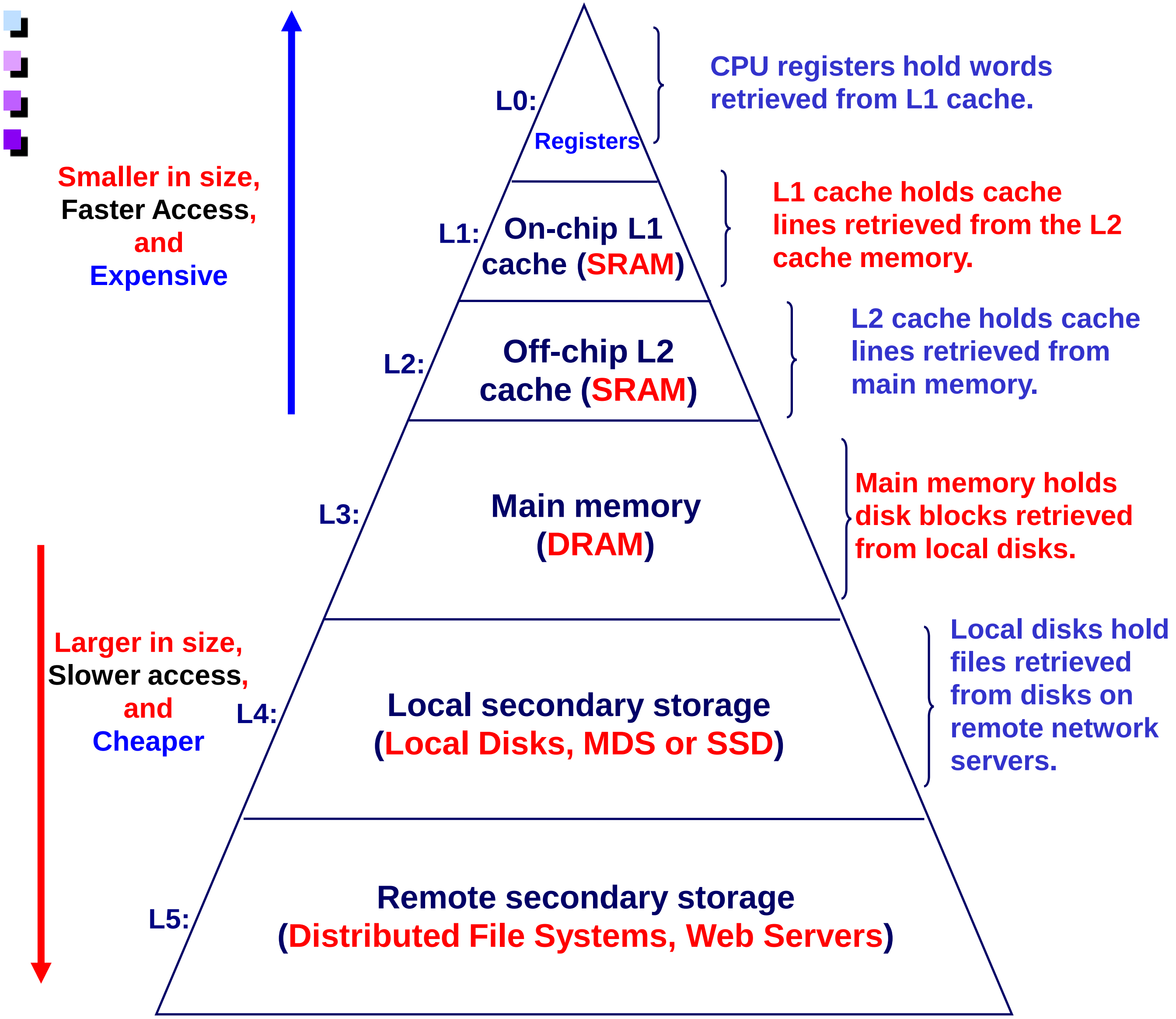
Dr. Tarek Helmy El-Basuny

Ch. 12: Mass-Storage Structure

- Memory System Hierarchy
- The computer system combines many storage technologies to balance the cost and the benefits
- Overview of Mass-Storage Structure
  - The mass-storage formatted to: Tracks, sectors, blocks, cylinders,
- Disk Performance and Parameters
  - Access time = Seek time + Latency time + Data transfer time.
  - Which parameters are static and which are dynamic?
- Disk's Head scheduling will try to improve both access time and transfer rate by scheduling the requests of the disk I/Os in a good order.
- Disk Scheduling Algorithms
  - FIFO, SSTF, Scan (The Elevator Algorithm), C-Scan, C-Lock.
- Selecting a Disk-Scheduling Algorithm
- Disk Formatting: Physical (Low Level) and Logical (High level)
- Disk Partitioning: Can we minimize the seek time through partitioning?
- Important Aspects of Mass Storage management: Reliability, Availability and Integrity.
  - Reliability: Can we detect errors and recover them? To improve the reliability
  - Availability: Is the disks always available to the user if one of them fails?
  - Integrity: we must know exactly what is lost when something goes wrong
- Enhances in disks technology motivated HW engineers to attach many disks to a computer system and allow them to act in parallel as a single logical disk (Redundant Array of Independent Disks, RAID) so that we can improve: Reliability, Availability and Integrity.
- Data Striping, and RAID Levels (0~6).

Memory System Hierarchy

- Combining many technologies to balance costs/efficiency.



Smaller in size, Faster Access, and Expensive

Larger in size, Slower access, and Cheaper

L0: Registers

L1: On-chip L1 cache (SRAM)

L2: Off-chip L2 cache (SRAM)

L3: Main memory (DRAM)

L4: Local secondary storage (Local Disks, MDS or SSD)

L5: Remote secondary storage (Distributed File Systems, Web Servers)

CPU registers hold words retrieved from L1 cache.

L1 cache holds cache lines retrieved from the L2 cache memory.

L2 cache holds cache lines retrieved from main memory.

Main memory holds disk blocks retrieved from local disks.

Local disks hold files retrieved from disks on remote network servers.

SRAM VS. DRAM

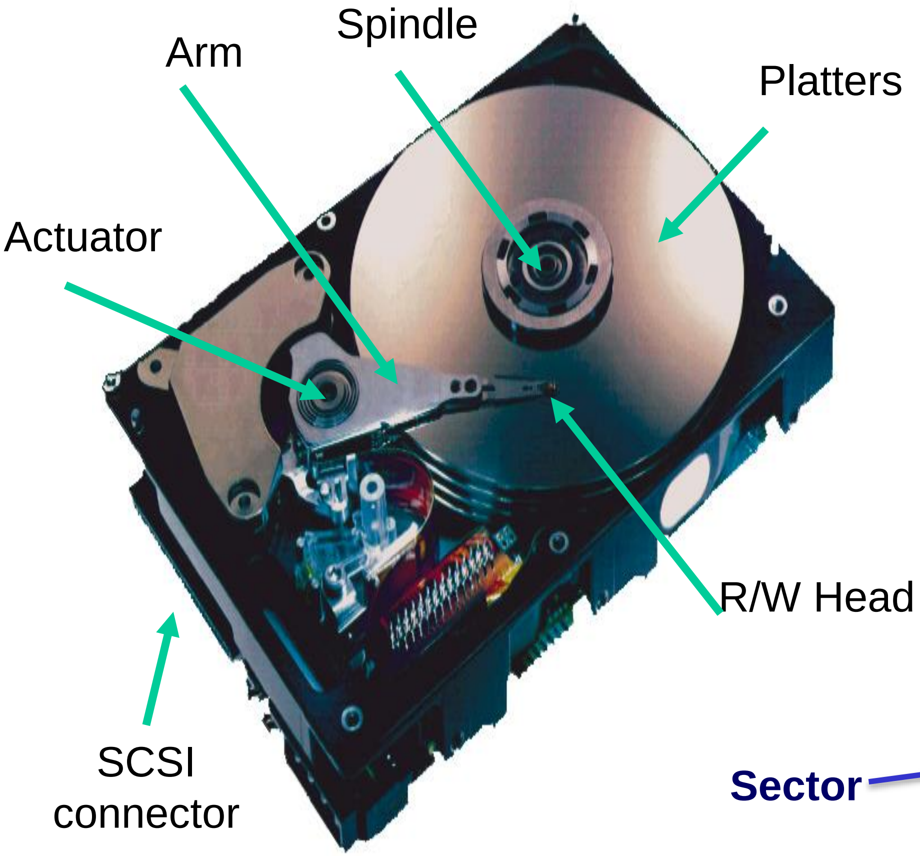
- **SRAM** is on-chip while **DRAM** is off-chip.
- **SRAM** is faster compared to **DRAM**.
- **SRAM** consumes less power than **DRAM** because **DRAM** needs to be refreshed.
- **SRAM** uses more transistors per bit of memory compared to **DRAM**.
- **DRAM** is available in larger storage capacity while **SRAM** is of smaller size.
- **SRAM** is more expensive than **DRAM**.
- Cheaper **DRAM** is used in main memory while **SRAM** is commonly used in cache memory.

Sold-State Disks

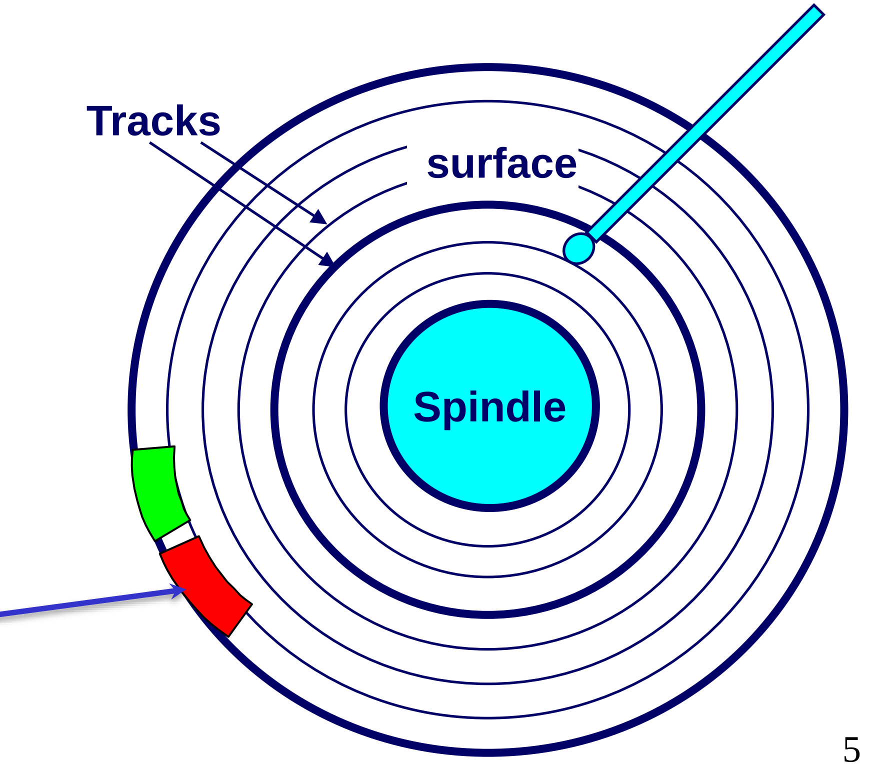
- It uses solid-state memory to store persistent data,
- Reliable in portable environments,
- No noise as it has no moving parts,
- Faster start up,
  - Does not need spin up
- Extremely low read latency,
  - No seek time (25 us per page/4KB)
- Deterministic read performance,
  - The performance does not depend on the location of data.
- Costs more, depending on the unit capacity.

What is inside a Hard Disk Drive?

- The Hard Disk contains platters, each with two surfaces.
- Each surface organized into concentric rings called tracks.
- Each track consists of sectors separated by gaps.

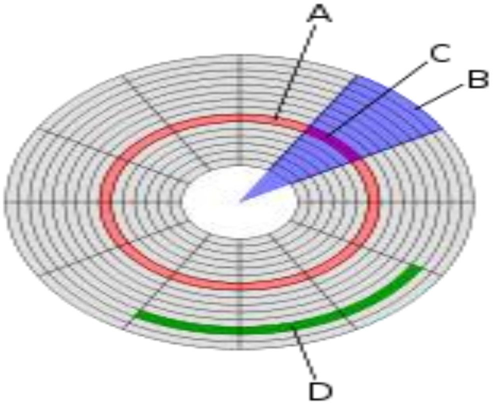


The read/write *head* is attached to the end of the *arm* and flies over the disk surface.



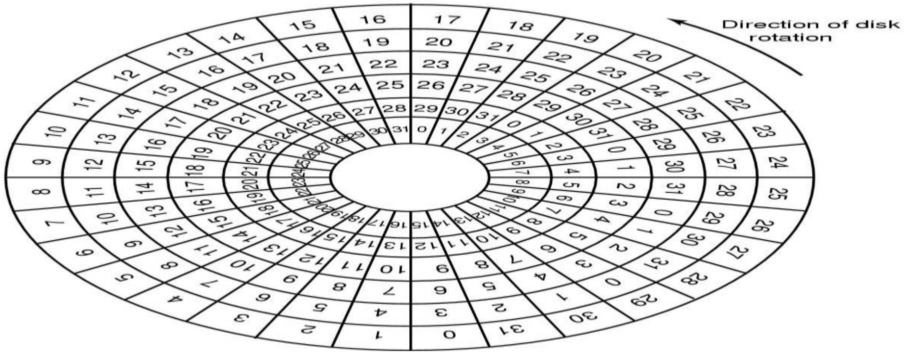
Hard Disk Structure

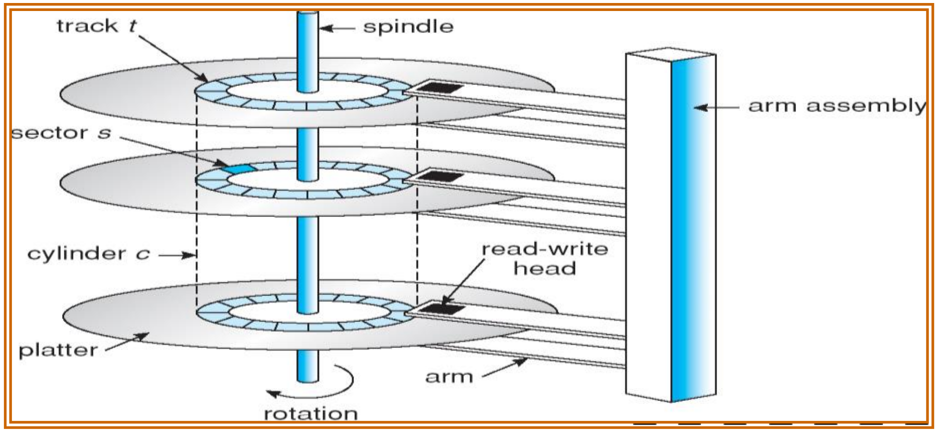
- Tracks: Multiple concentric circles containing data.
- Sectors: Subsections of a track.
- Blocks: Subsection of a sector created during formatting (512-4096 B).
- Cylinder: All tracks of same diameter in the disk pack.
- Moveable heads: One r/w head per surface.
- Seek time: Time to get r/w head to the desired track.
- Head crash: Results from disk head making contact with the disk surface.
- Rotation/latency delay: Waiting time for the desired data to spin under R/W head.
- Block transfer time: Time to read or write a block of data.



- A: Track,
- C: Track Sector
- B: Sector,
- D: Cluster

Disk Data Layout

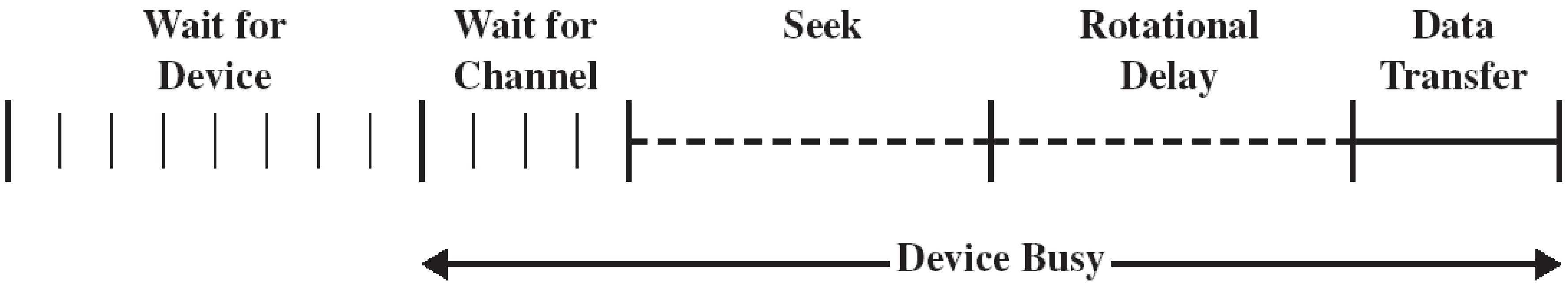




Disk Performance Parameters

- To read or write, the head must be positioned at the desired track and at the beginning of the desired sector.
- Seek time: Time it takes to position the head at the desired track.
- Rotational delay/**latency**: The time it takes for the disk to rotate so that the required sector is above the disk's head .
  - Drives rotate at 60 to 200 times per second.
- Transfer rate is the rate at which data flows between the drive and the MM. Data transfer occurs as the sector moves under the head.
- Access time = Seek time + Latency time (about 15-60ms) + Data transfer time (about1-2ms).
- Disk bandwidth is the total number of bytes transferred, divided by the total time (between the first request for service and the completion of the last transfer).

Timing of a Disk I/O Transfer



Total Average Access Time: $T_a = T_s + 1/(2*r) + b / (r*N)$

Where:

T_s = Average seek time

b = Number of bytes to be transferred

N = Number of bytes on a track

r = Rotation speed in revolutions per second

- Transfer Rate to/from disk depends on the rotation speed of the disk and the track location, **so it will be higher for data on the outer tracks** (where there are more data sectors) and **lower towards the inner tracks** (where there are fewer data sectors).
- In general the Transfer Rate **depends on the #** of bytes on track and rotation speed.

Disk's Head Scheduling

- The OS is responsible for using the mass-storage in an efficient manner.
  - This means having a fast access time and large disk bandwidth.
- The goal is to minimize the seek time (Seek time ≈ seek distance)
- The idea is to improve both **access time** and **bandwidth** by scheduling the requests of the disk I/Os in a good order.
- Whenever a process needs an I/O to or from the disk, it issues a system call to the OS, where the request specifies:
  - Whether this operation is input or output.
  - What is the disk address to read from?
  - What is the memory address to write into?
  - What is the number of bytes to be transferred?
- If the desired disk drive and the controller are available, the request can be serviced immediately, otherwise the request will be queued for that drive.

Disk Scheduling Algorithms

- **Disk's Head Scheduler:** Examines the current contents of the disk's queue and decides which request to be served next.
- The head scheduler must consider **3 important factors:**
  1. Overall performance of the disk
  2. Fairness in treating processes' disk requests
  3. Cost of executing scheduling algorithm

➢ **Several algorithms exist to schedule the disk I/O requests.**

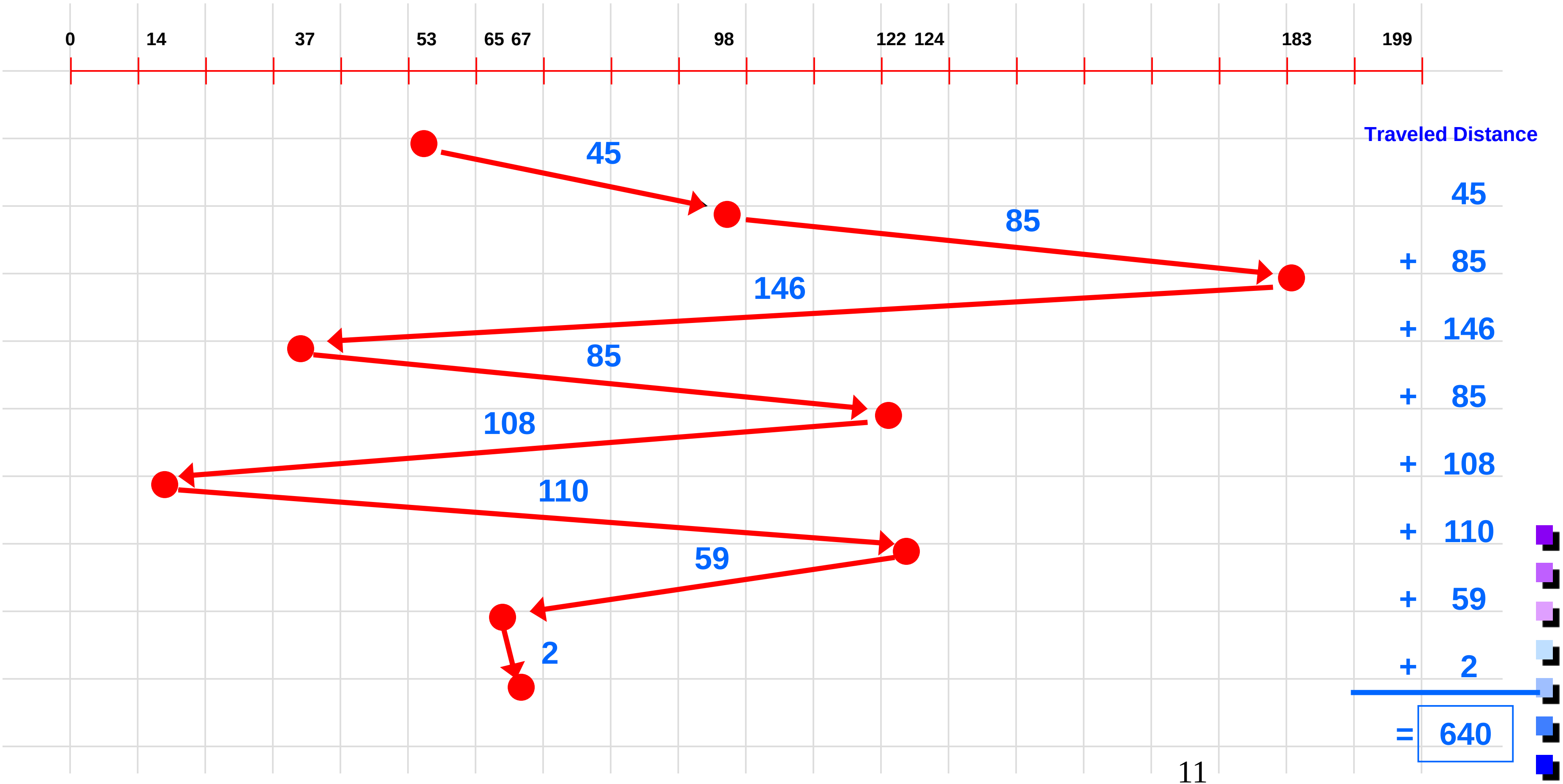
- ➢ First-In First-Out (FIFO),
- ➢ Shortest Seek Time First (SSTF)
- ➢ SCAN: The Elevator Algorithm,
- ➢ C-SCAN, and C-LOOK

- These algorithms will be used with **electromechanical disks (HDD)** where they have spinning disks and movable heads.
- **They will not be used with solid state disks or electronic disks**.
- These algorithms will be evaluated based on the **head movement distance** that will affect the seek time. We will illustrate that **with the following example**:
- Disk queue (0-199) cylinders with requests for I/O to blocks on cylinders 98, 183, 37, 122, 14, 124, 65, 67, with head at cylinder 53.

First-In First-Out (FIFO)

- Requests will be processed sequentially according to the order.
- Fair to all processes.

Head at 53		98	183	37	122	14	124	65	67



Traveled Distance

45
+ 85
+ 146
+ 85
+ 108
+ 110
+ 59
+ 2
= 640

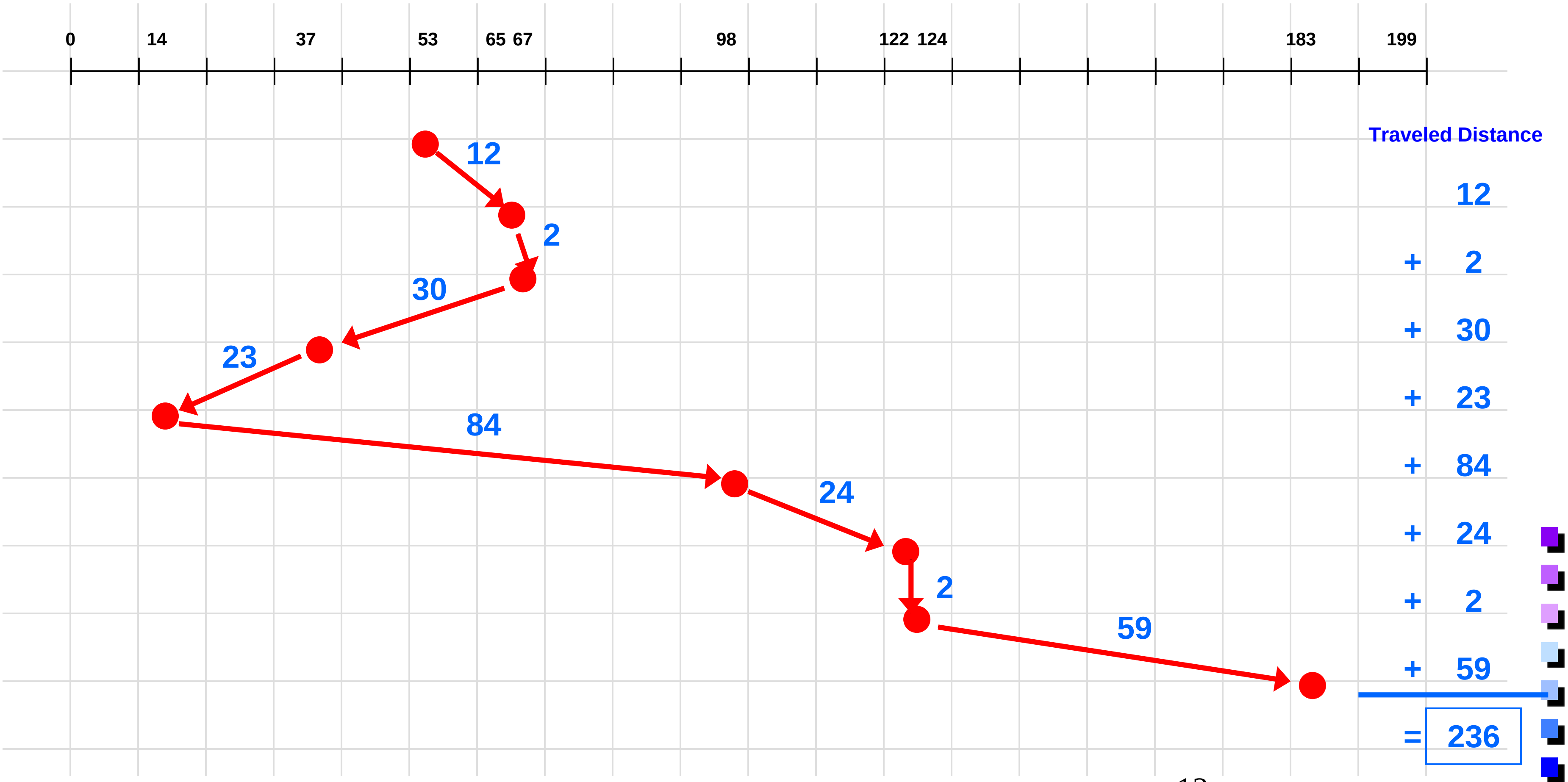
SSTF – Shortest Seek Time First

- When a disk operation finishes, the OS chooses the closest request to the current head position to be processed (the one that minimizes seek time).
- Selects the request with the minimum seek time from the current head position.
- SSTF scheduling is a form of SJF scheduling.
- This algorithm minimizes latency and thus gives the best overall performance.
- It suffers from poor fairness. Requests will get varying responses depending on how lucky they are in being close to the current location of the heads.
- In the worst case, some requests may starve (long delayed).

SSTF – Shortest Seek Time First

- Requests will be processed based on the closest first.
- Some request will starve

Head at 53		98	183	37	122	14	124	65	67	



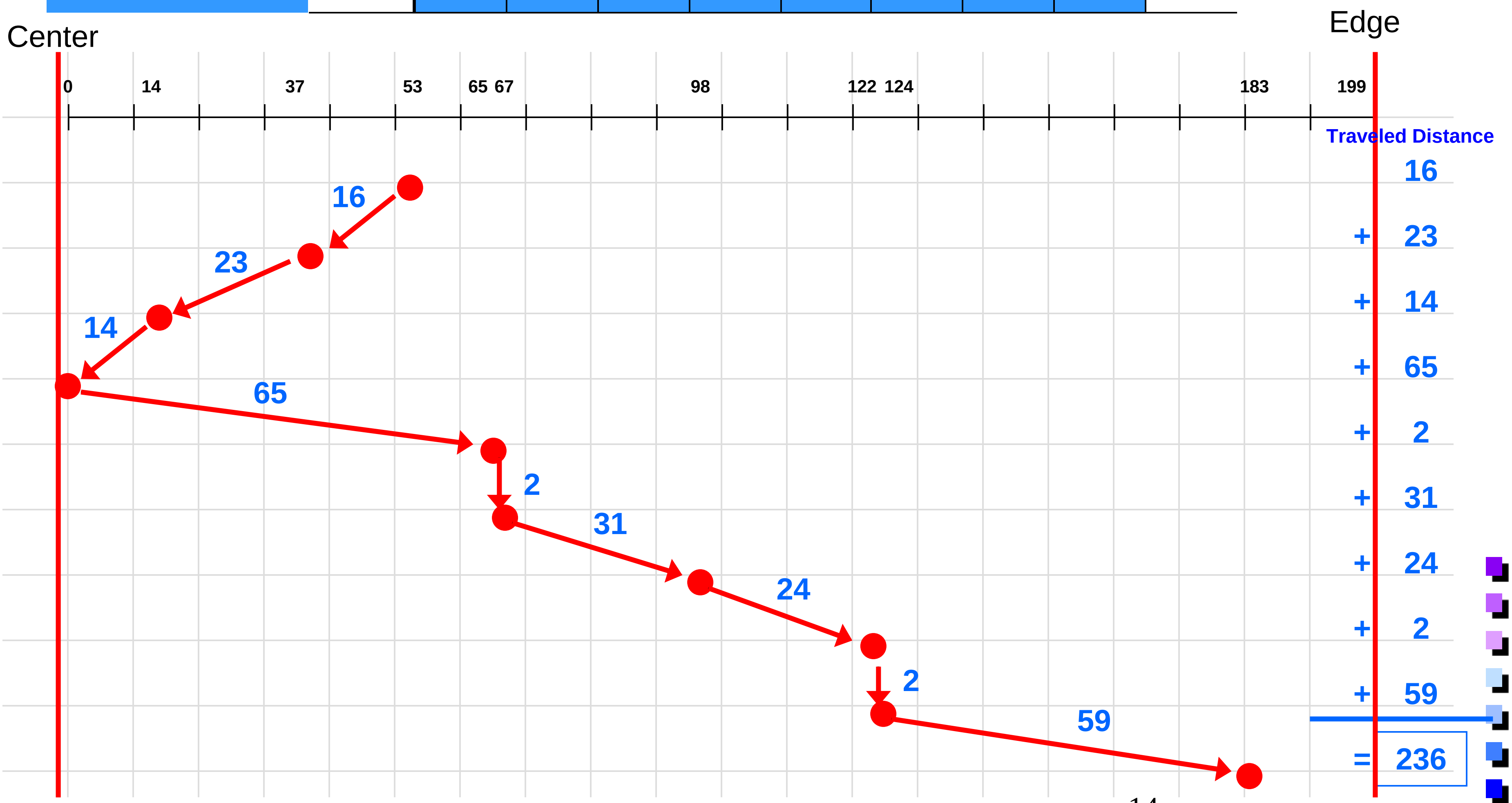
Traveled Distance

12
+ 2
+ 30
+ 23
+ 84
+ 24
+ 2
+ 59
= 236

SCAN: The Elevator Algorithm

- The disk head moves (from the **current position** to the **center** of the disk) serving the closest requests in that direction. When it runs out of requests in the direction it is currently moving, it switches to the opposite (**to the edge**) direction doing the same.

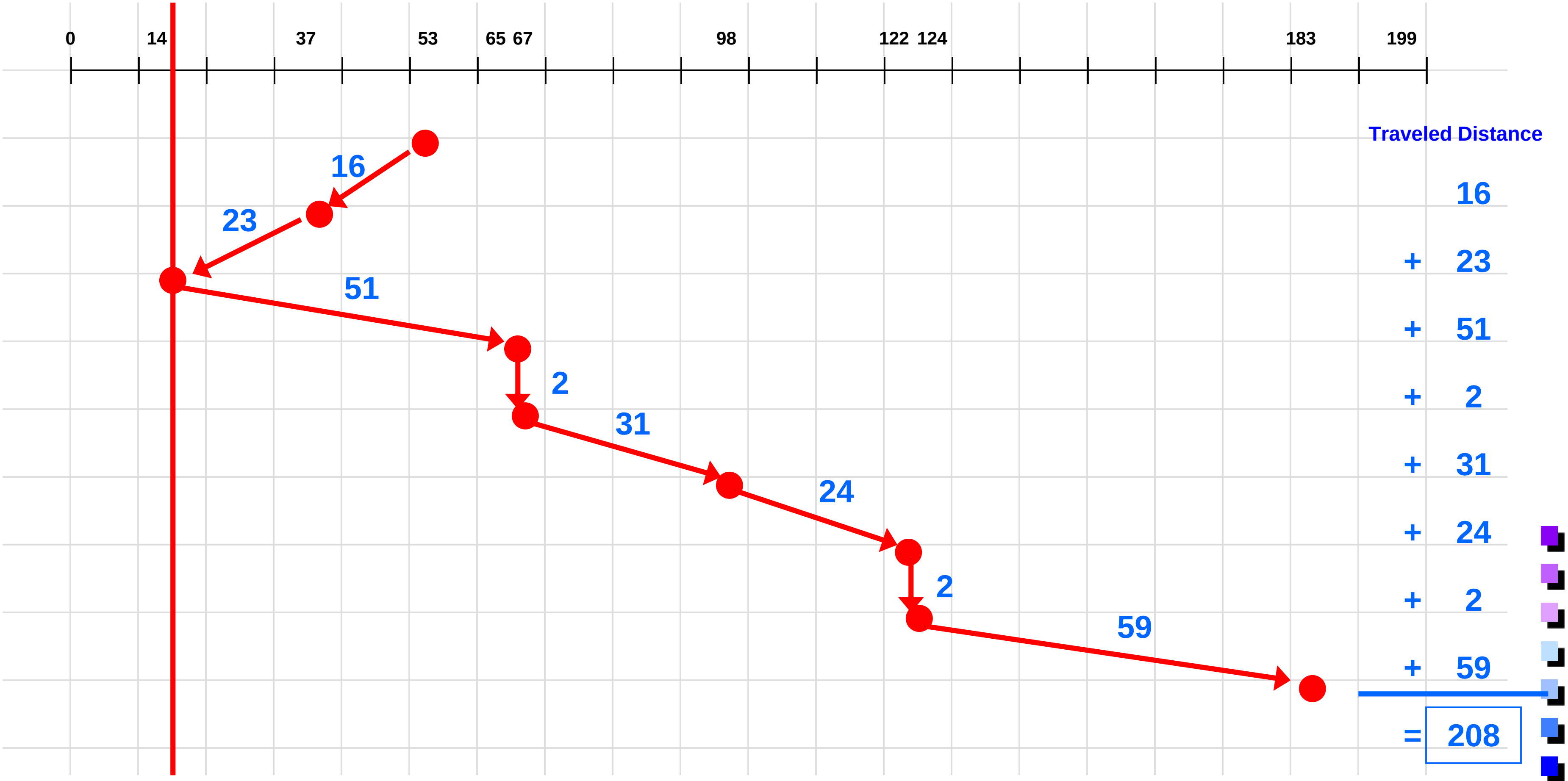
Head at 53		**98**	**183**	**37**	**122**	**14**	**124**	**65**	**67**	←



Some Modification to the Scan

- Reverse the direction immediately, without going all the way to the edge or to the center of the disk.

Head at 53		98	183	37	122	14	124	65	67



Traveled Distance

16
+ 23
+ 51
+ 2
+ 31
+ 24
+ 2
+ 59
= 208

SCAN: The Elevator Algorithm

- This algorithm usually gives fair service to all requests, but in the worst case, it can still lead to starvation.

- While it is satisfying requests on one cylinder, other requests for the same cylinder could arrive.

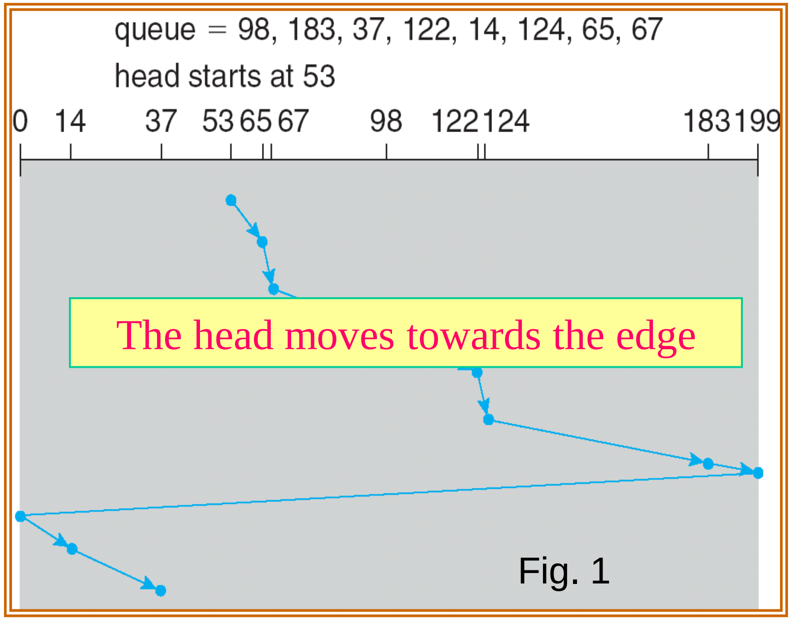
- If enough requests for the same cylinder keep coming, the heads would stay at that cylinder forever, starving all other requests.

- This problem is easily avoided by limiting how long the heads will stay at any cylinder. One simple scheme is to serve only the requests for the cylinder that are already there when the head gets there.

- New requests for that cylinder that arrive while existing requests are being served will have to wait for the next pass.

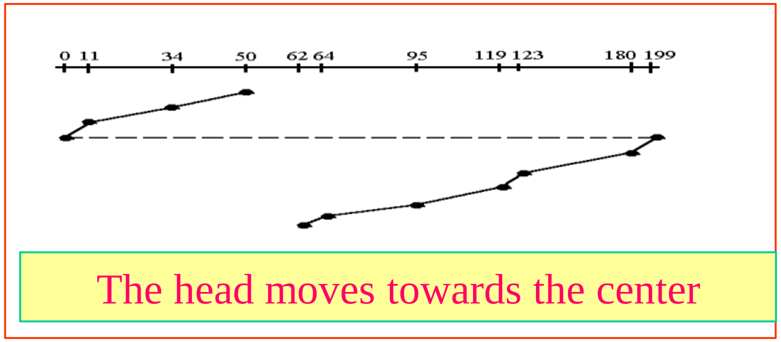
Circular-SCAN (C-SCAN)

- *Circular scanning works just like the elevator to some extent.*
- Provides a more uniform wait time than SCAN.
- The head moves from the current position ***toward the nearest end*** ***or*** **the one that maximizes the rate**, servicing requests as it goes.
- When it reaches the other end, however, *it jumps to the other end* without servicing any requests on the return trip.
- *Once it hits the bottom or top it jumps to, moves in the same direction* servicing requests as it goes.
- ***Keep in mind that the huge jump doesn't count as a head movement***
- Treats the cylinders as a circular list that wraps around from the last cylinder to the first one.



The head moves towards the edge

Fig. 1



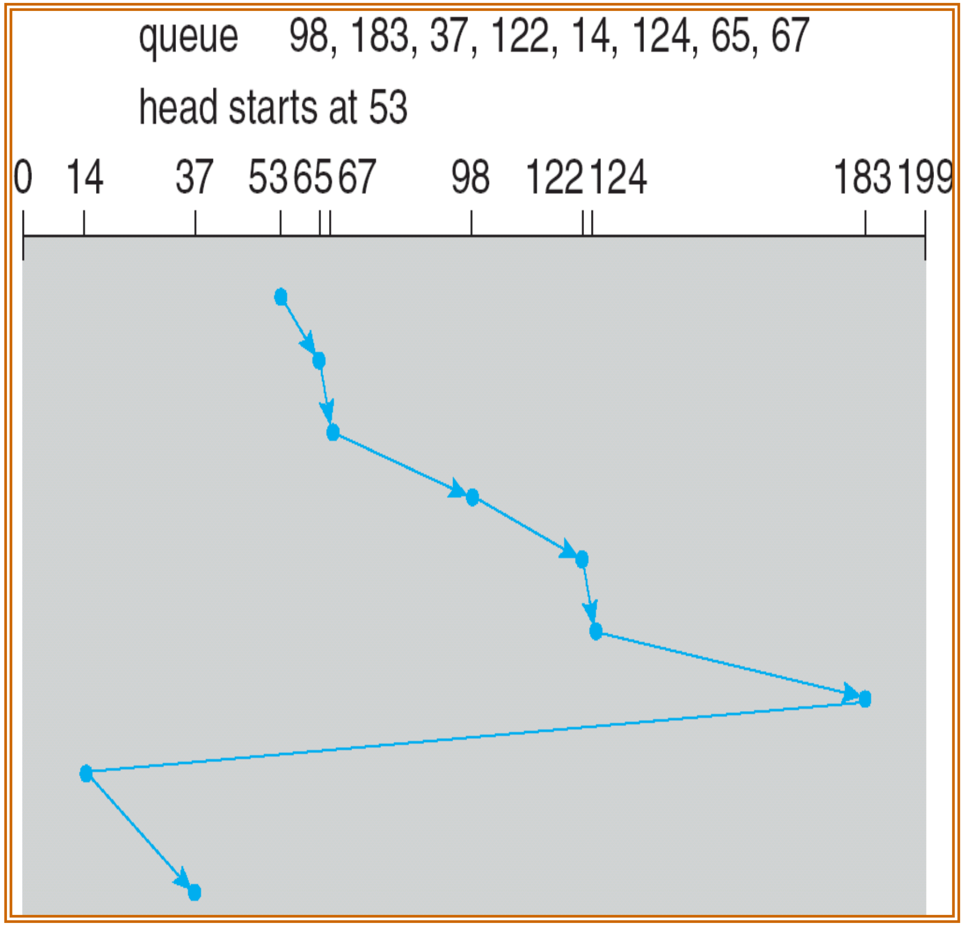
The head moves towards the center

The total head movement for this algorithm is only 187 track

- The total head movement of Fig. 1 is ??? cylinders. Please calculate the head movements.

C-LOOK

- It is a modified version of C-SCAN
- Arm only goes as far as the last request in each direction.
- Then reverses direction immediately, without going all the way to the end or to the center of the disk.



- Illustration shows total head movement of ??? cylinders. Please calculate the head movements. About 30 cylinders less than the C-Scan.

Selecting a Disk-Scheduling Algorithm

- **What is the best disk's head scheduling algorithm?**
- The answer depends on too many factors, i.e. types of file, number of requests, allocation method, etc.

- SSTF is common and has a natural appeal if the process **needs high data transfer rate**.
- SCAN and C-SCAN perform better for **systems that place a heavy load on the disk**.
- Suppose that the queue has just one or two requests, then all algorithms behave the same as FCFS.
- Selecting the best algorithm **can be biased by the file-allocation method**, contiguous, linked, indexed.
- In general, either **SSTF or C-LOOK** is a reasonable choice for the default/normal processes.
- **Recommended actions:**
- **The disk-scheduling algorithm should be written as a separate module of the OS, allowing it to be replaced with a different algorithm if necessary or to be updated.**
- **The location of directories and index blocks is also important.** Opening a file needs to search the directory on the disk and then reading the file needs to access the disk again. Caching the directory and disk blocks in main memory can also help in reducing the head movement, specially for reading operation.

Disk Formatting

- Two formatting processes are required before we can write data to a HDD:

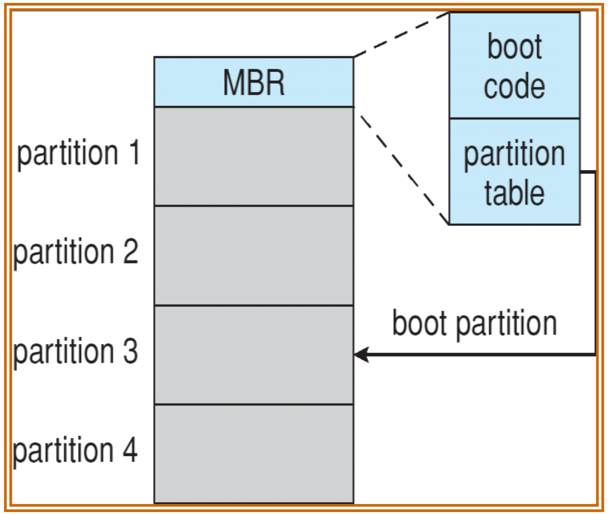
1. Low-level (Physical) formatting: usually performed at the factory
   - ➔ Marking out cylinders and tracks for a blank hard disk, and then dividing tracks into multiple sectors.
   - ➔ Sequentially numbers the tracks and sectors on the disk,
   - ➔ Identifies each track and sector,
   - ➔ Disk is physically prepared to hold data.

2. High-level (Logical) formatting: usually performed by the OS.
   - ➔ Determines how the OS uses a disk and labels logical bad sectors.
   - ➔ Builds data structure to keep track of directories and files (initializes FAT, and directory structure tables)
   - ➔ Builds the data structures used by the OS to identify the logical drive or partition's contents .
   - ➔ Sets the bootable partition.

Disk Partitioning and MBR

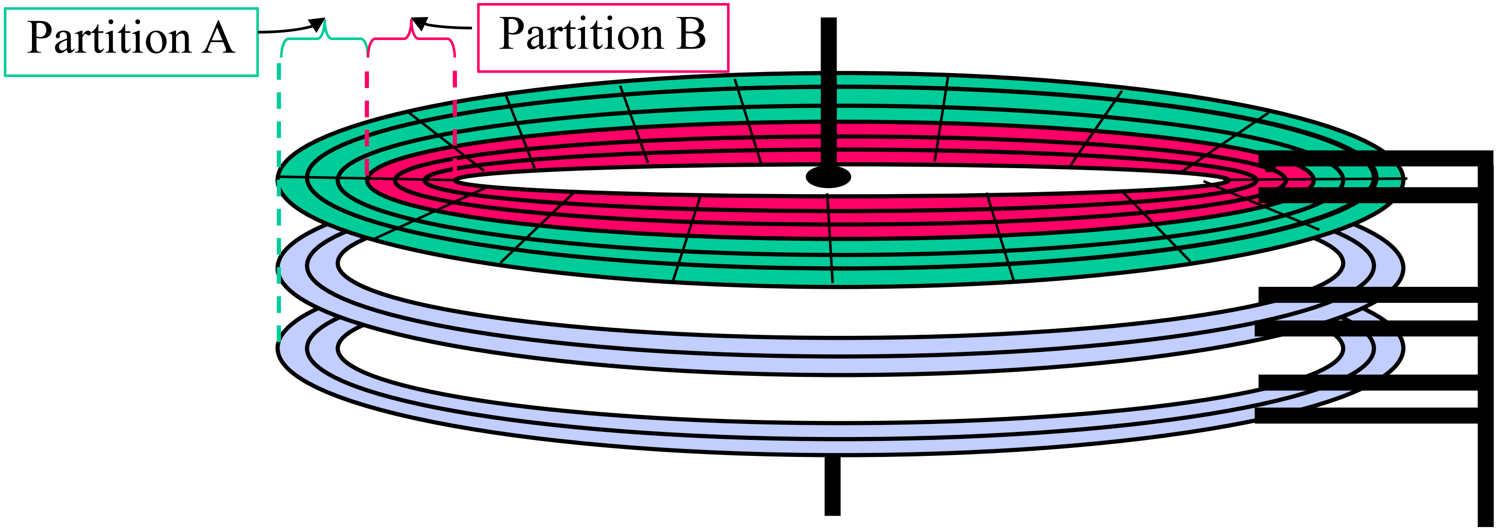
- Creating partitions on a hard drive enables it to support separate file systems, each in its own partition. In fact, each partition is like a separate disk.
- **Master Boot Record** (**MBR**) is the first sector of a hard drive (cylinder 0, track 0, sector 1):
  - MBR Identifies how and where an OS is located so that it can be loaded into the computer's MM.

- What happens while turning on the computer?
  - BIOS executed and loads the MBR.
  - MBR also contains information about the HD's partitions.
  - There is a flag to identify the boot partition.
  - MBR then loads OS kernel into the MM.



Disk Partitioning and the Seek Time

- If seek distances can be minimized then the overall performance will be improved and the importance of the disk-head scheduling algorithm will be narrowed.
- Disks are typically partitioned **to minimize the largest possible seek time**.
  - A partition is a collection of cylinders
  - Each partition is logically a separate disk
- Swap-space (virtual memory): The disk space that is used as an extension to the main memory.
- Swap-space can be stamped out of the normal file system or, more commonly, it can be in a separate disk partition.
- Knowing that the transfer rate depends on the position of reading or writing, do you think virtual memory should be selected from outer or inner partitions?



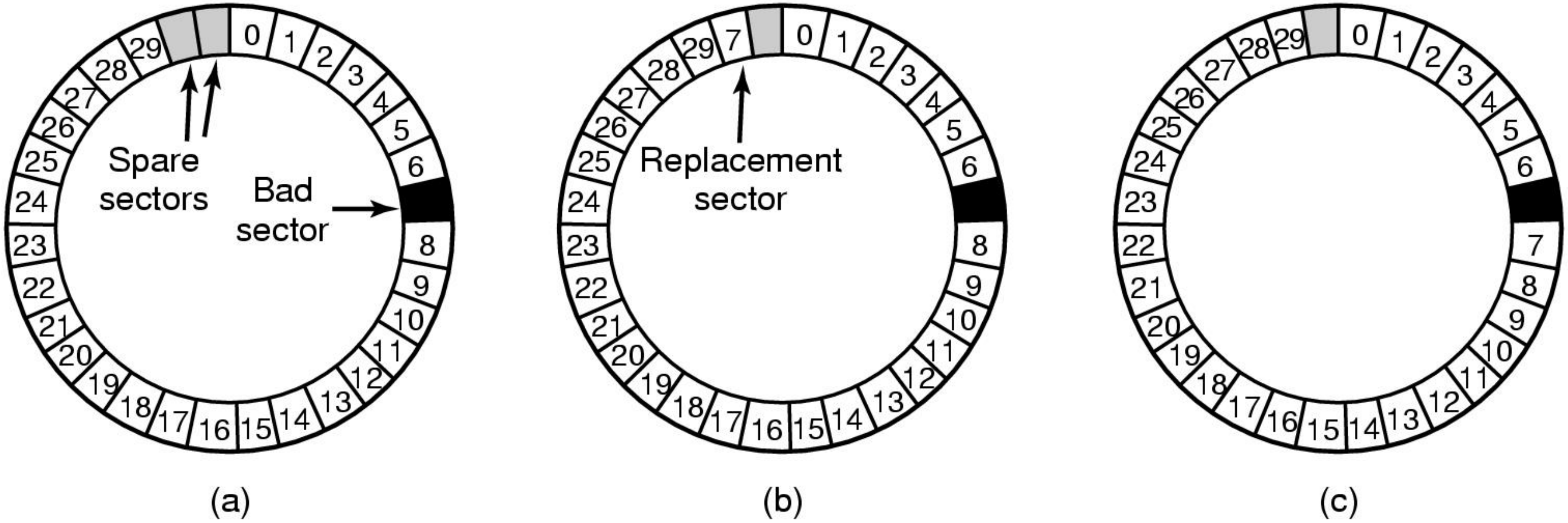
Mass Storage Performance

- Three important aspects of Mass Storage management:
  - Reliability: Can we **retrieve** the information in case of a disk failure.
    - Storing extra information that is not needed normally but that can be used in case of failure of a disk. i.e.
    - Redundancy is the main hack to increase the reliability.
    - Error Correcting Code (ECC) and various correction schemes improve the reliability (but cannot improve Availability).
  - Availability: Is the system always available to the user if something fails? **Availability is improved by adding redundant disks**.
    - If a single disk fails, the required information can be accessed from the redundant disk.
    - Capacity penalty to store redundant information.
    - Bandwidth penalty to update both disks.
  - Data Integrity
    - Can we know exactly what is lost when something goes wrong.

Handling Bad Sectors on the HDD

A. To improve both **reliability** and **availability** of the HDD, the OS reserves spare sectors to:

- Substitute a spare for the bad sector (sector sparing) or
- Shift all sectors to bypass bad one (sector forwarding)



Reliability improve by using ECC (Parity Bit)

- A parity bit can be used to detect and correct a one-bit error ?
  - Suppose you have a binary number, represented as a collection of bits: <b3, b2, b1, b0>, e.g. 0110
- We can add one extra bit (parity) to reflect filliping of any bits.
- Odd and even parity:
  - Count the number of bits that are (ones), see if it's odd or even.
    - EVEN parity: if the number of 1 bits is even set the parity bit to 0.
    - ODD parity: if the number of 1 bits is odd set the parity bit to 1.
  - Parity(<b3, b2, b1, b0 >) = P0 = $b0 \otimes b1 \otimes b2 \otimes b3$
  - Parity(<b3, b2, b1, b0, P0>) = 0 if all bits are intact
  - Parity(0110) = 0, Parity(01100) = 0
  - Parity(11100) = 1 => ERROR!
  - Parity can detect a single bit error, but can't tell you which of the bits got flipped.

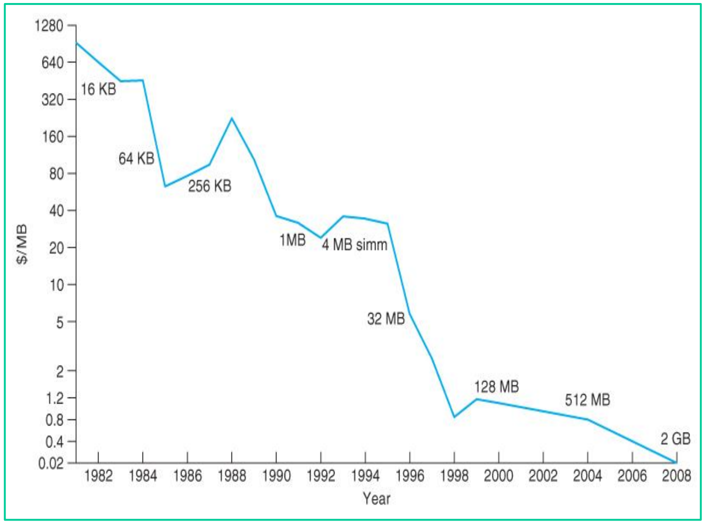
Reliability improve by using ECC (Hamming Code)

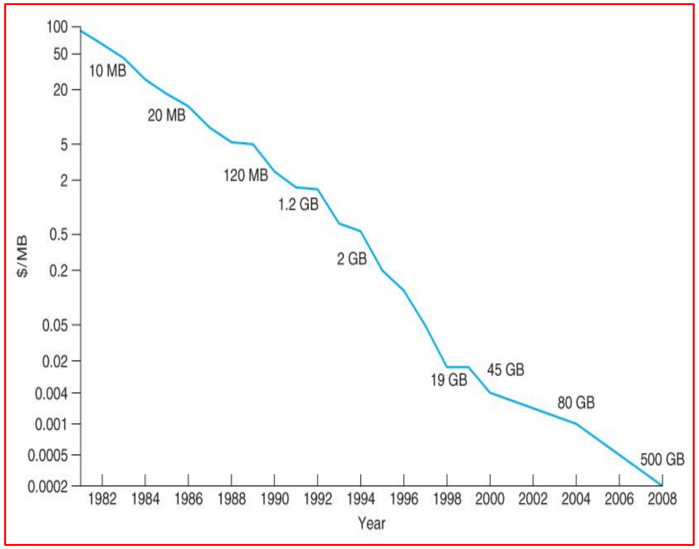
- Hamming code can detect double bits error and detect & correct single bit errors in a message. Hamming Code is composed on 3 bits as following:
  - $h0 = b0 \otimes b1 \otimes b3$
  - $h1 = b0 \otimes b2 \otimes b3$
  - $h2 = b1 \otimes b2 \otimes b3$
  - If the code is (1101)
  - $h0\ (<1 \otimes 0 \otimes 1>) = 0$
  - $h1\ (<1 \otimes 1 \otimes 1>) = 1$
  - $h2\ (<1 \otimes 1 \otimes 0>) = 0$
  - The hamming code is <010>
  - Message <1101>, with hamming = <b3, b2, b1, h2, b0, h1, h0> = <1100110>
  - If a bit is flipped in the message, e.g. <11**1**0110>
  - Hamming of the **error** message (<1111>) = <h2, h1, h0> = <111>
  - Now the difference between the two hamming codes <010>, <111> is <101>
  - The Error has occurred in bit # **5**.

Disks Technology Trends

- Disks are getting smaller in size
  - Smaller → spin faster; smaller distance for head to travel; and lighter weight.
- Disks are getting denser
  - More bits/square inch → small disks with large capacities.
- Disks are getting faster
  - Seek time, rotation latency: 5-10% faster/year
  - Bandwidth: 20-30% larger/year
- Disks are getting cheaper
- Overall:
  - Disk capacities are improving much faster than performance.
- A fixed disk drive is likely to be more reliable than a removable disk.
- So it is now economically feasible to attach many disks to a computer system and allow them to act in parallel as a single logical disk.
- Having a large number of disks in a system improves the rate of R/W and the reliability.

Price per Megabyte of DRAM





Price per Megabyte of Magnetic Hard Disk

RAID Technology

- Because disks are getting small and cheap, so it’s easy to put lots of disks (10s to 100s) in one box to increase the storage, performance, and availability.
- RAID is a **Redundant Array of Inexpensive/**Independent **Disks** viewed by the **OS as a Single Logical Expensive Disk** (SLED).
- ➢ With parallel access to multiple disks, the OS can improve the transfer rate.
- The RAID box with a RAID controller looks just like a SLED to the computer.
- Redundant copies and Error Correction Code (ECC) are striped across the disks to increase reliability and availability.

RAID Levels

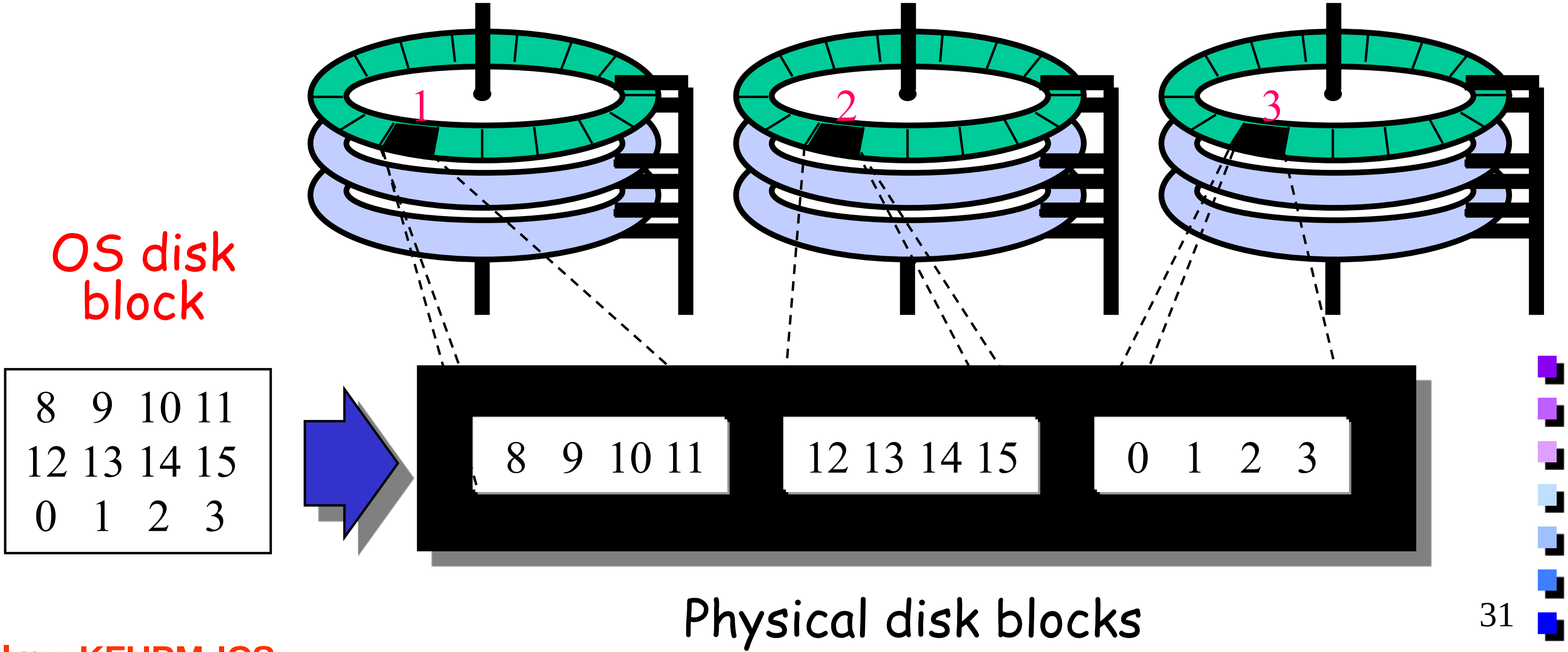
- Redundancy can be implemented either by:
  - Duplicating/mirroring: replicates **data** on more than one disk.
  - Striping: is a technique that breaks up a file and spreads the data across all the disk drives in a RAID group.
- Seven RAID levels (0~6) were introduced, because of:
  - Cost
  - Performance
  - Fault-tolerance characteristics
  - Different storage needs
- They were used in different environments for different applications:
  - Banking systems,
  - Storage systems,
  - Responsive services,

Improvement of Disk Performance

- **Data Striping:**
  - Bit-level striping: Strip the bits of each byte across multiple disks.
    - No. of disks can be a multiple of 8 or divides of 8.
  - Byte-level stripping: Strip the bytes across multiple disks.
  - Block-level striping: Blocks of a file are striped across multiple disks; with n disks, block i goes to disk (i mod n)+1
- Every disk participates in every access
  - Number of I/O per second is the same as a single disk
  - Number of data read/written per second is improved
- Provide high data-transfer rates.
  - Throughput is increased through a larger effective block size and through the ability to perform parallel I/O.

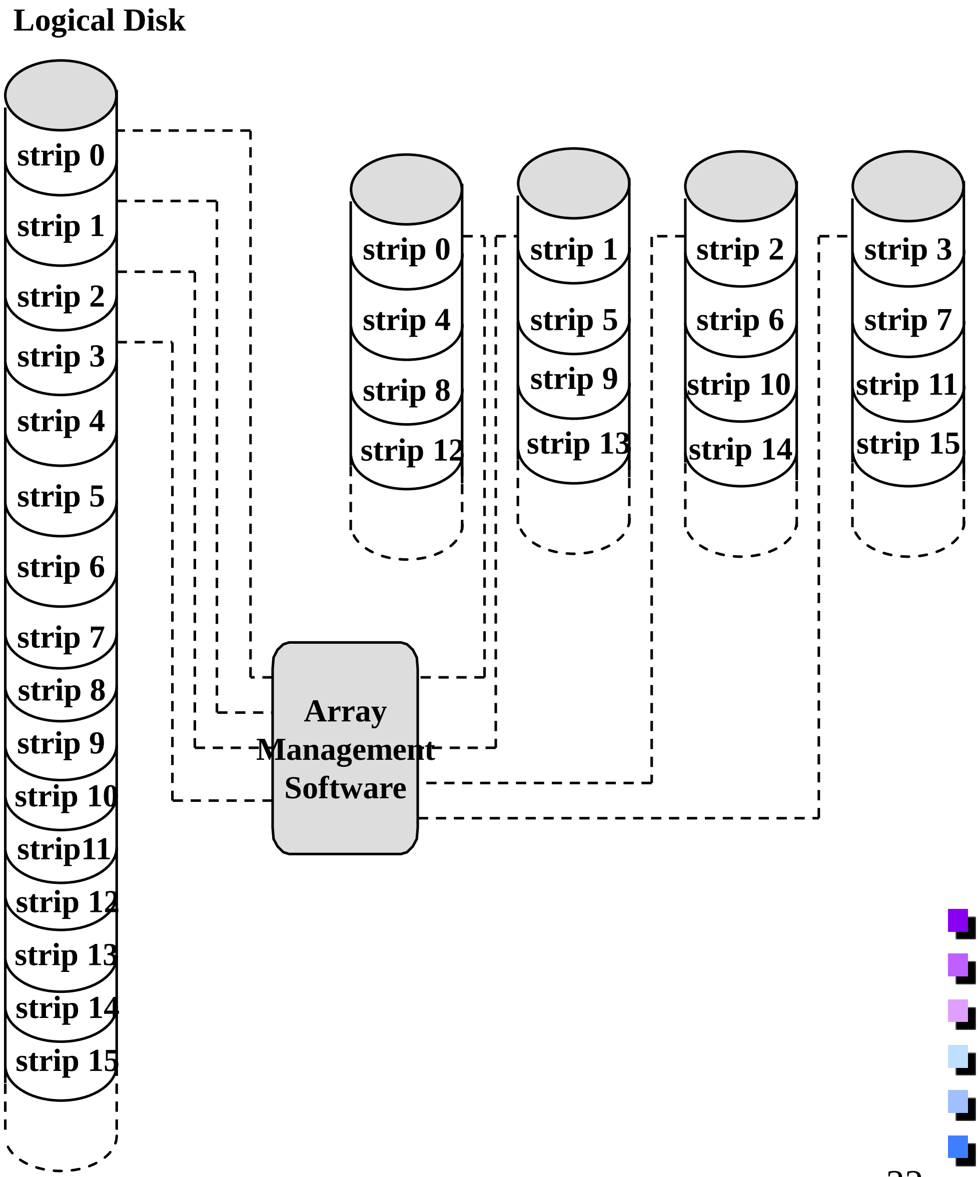
Disk Striping (RAID-0)

- Disk striping (RAID-0)
  - Disk blocks are striped and stored on separate disks.
  - Provides for higher disk bandwidth through a larger effective block size
- If the rotation of the disks can be synchronized so that all have the same rotational latency for any read/write operation then we have what is known as RAID.

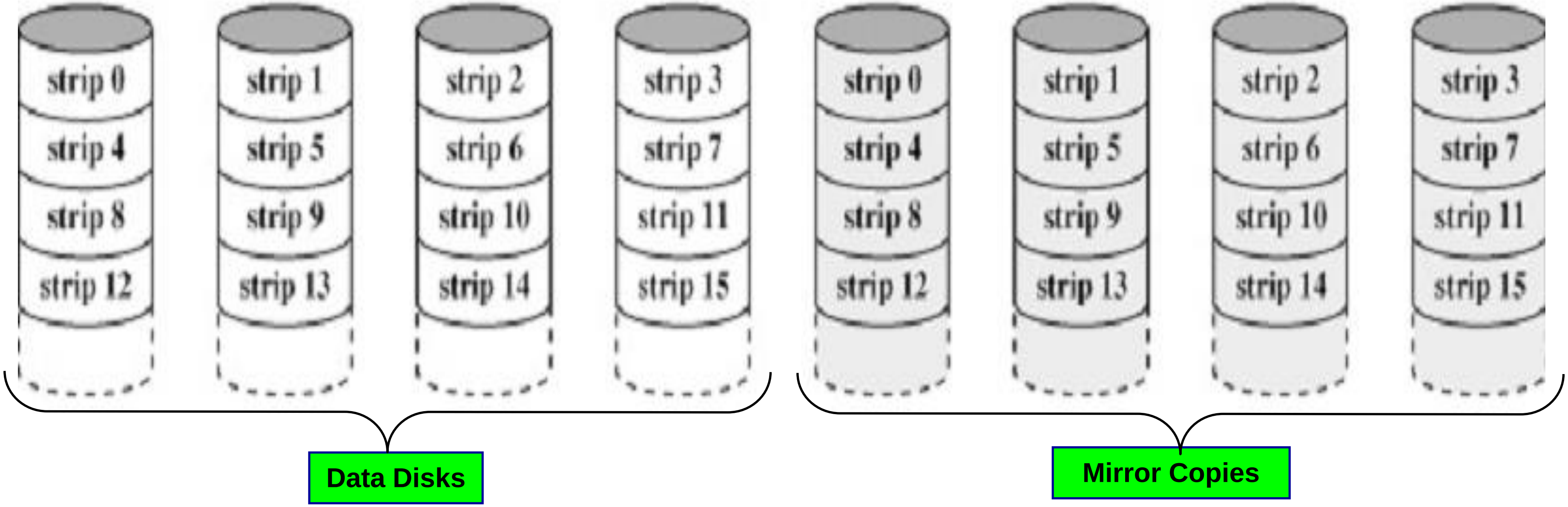


Data Mapping for RAID Level 0

- In RAID Level 0 files are striped across disks, no redundant info which makes the reliability lower than with single disk (***if any disk in the array fails, we have a problem***)
- High read throughput/data transfer rate
  - Strips 0, 1, 2, and 3 can be accessed in parallel
- Best write throughput (no redundant info to write)
  - High I/O request rate
  - Servicing multiple I/O requests in parallel, e.g., requests for strip 0 and strip 3 can be serviced simultaneously.
- Makes disks share the load, improving Latency: less queuing delay – a queue for each disk.

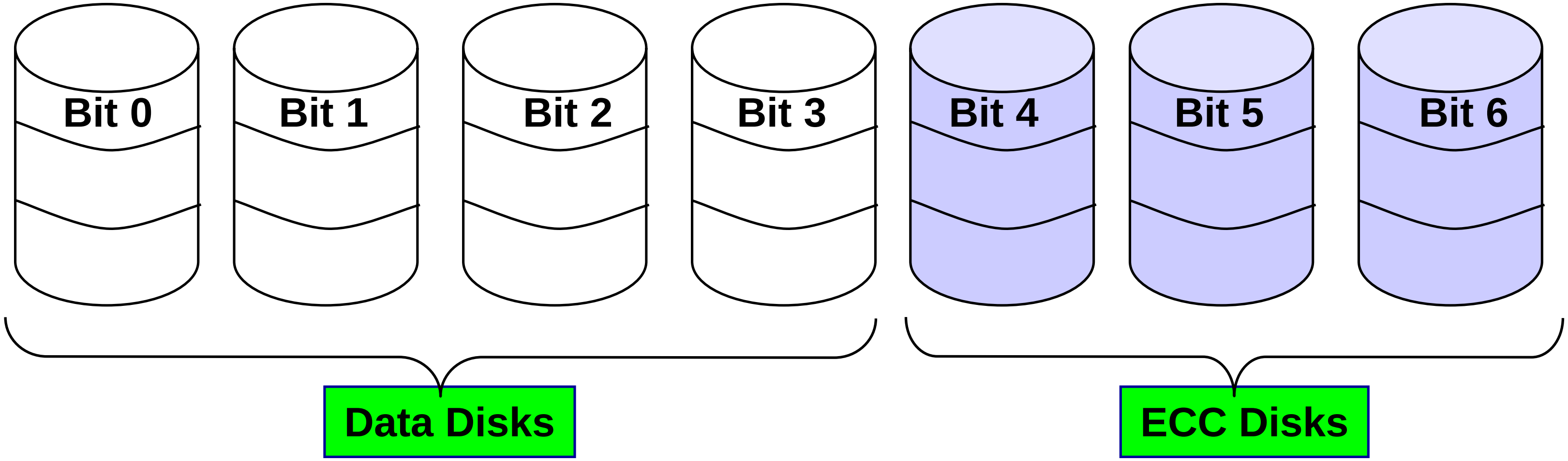


RAID Level 1



- Redundancy is achieved by using mirroring (duplication of data on other disks) where each disk is fully copied on its “shadow”.
- Files written to both, if one fails, flags it and gets the data from the mirror.
- Expensive due to the need of mirror disks.
- A read can be serviced by either disks (the least access time).
- A write requires updating two strips, Logical write = two physical writes.
- Recovery is simple.

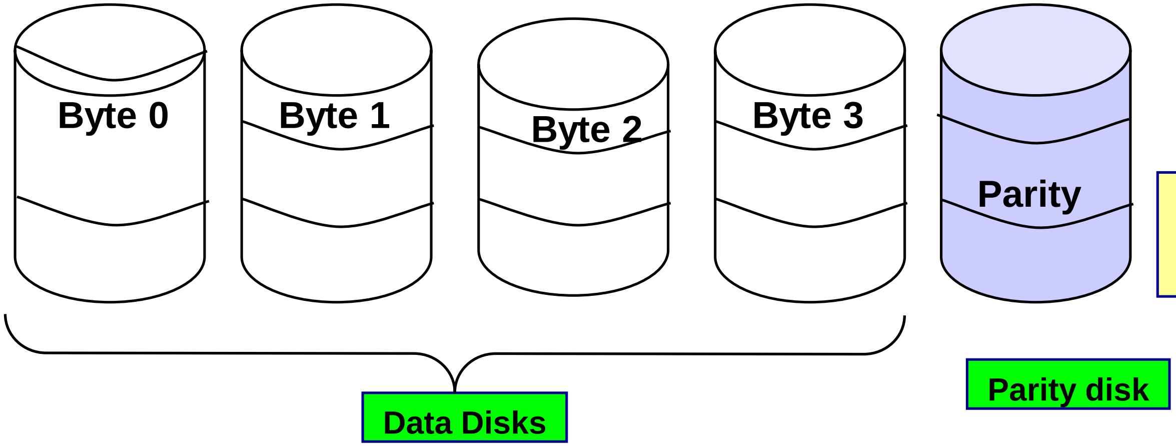
RAID Level 2



- Bit-level striping with (Hamming ECC) for error correction & recovery.
- All member disks participate in the execution of every I/O request. (Gives high transfer rate but not I/O request rate).
- **Spindles and heads are all synchronized to the same position**.
- Requires smaller number of disks compared to RAID level 1.
- It uses parity checks, associate a parity bit with each byte in the memory, even parity or odd parity.
- In RAID level 2, 3 **parity bits** are used to reconstruct the damaged byte.

Raid Level 3

- **Striping at the byte level** and stores dedicated parity bits on a separate disk drive
- Because RAID 3 combines parity and striping with stored parity bits on a dedicated disk,
- The disks must spin in sync, so sequential read/write (R/W) operations achieve good performance.
- A read accesses all the disks.
- A write accesses all disks plus the parity disk.
- On a disk failure, read data from the remaining disks plus parity disk to compute the missing data.
- High throughput for transferring large amounts of data

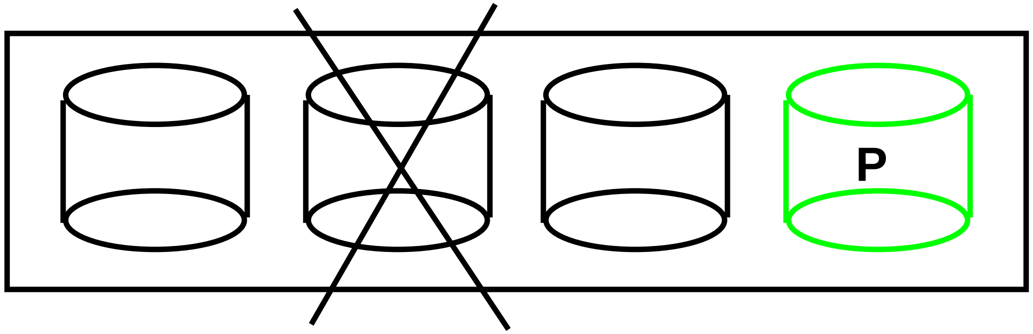


Single parity disk can be used to detect and correct errors.

RAID Level 3: Parity Disk

10010011
11001101
10010011
. . .

logical record



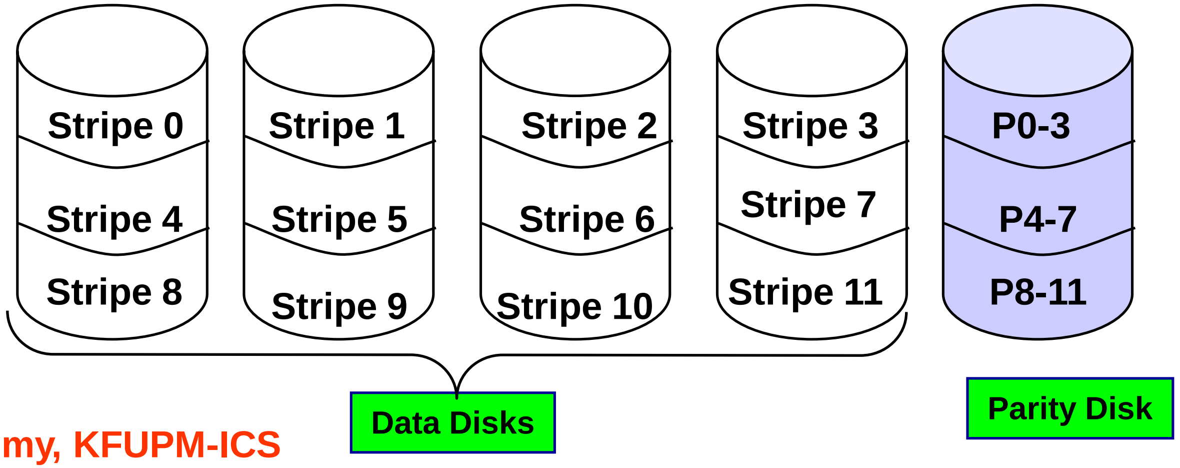
Striped physical records →

1	1	1	1
0	1	0	1
0	0	0	0
1	0	1	0
0	1	0	1
0	1	0	1
1	0	1	1
1	1	1	1

25% capacity cost for parity in this configuration (1/N)

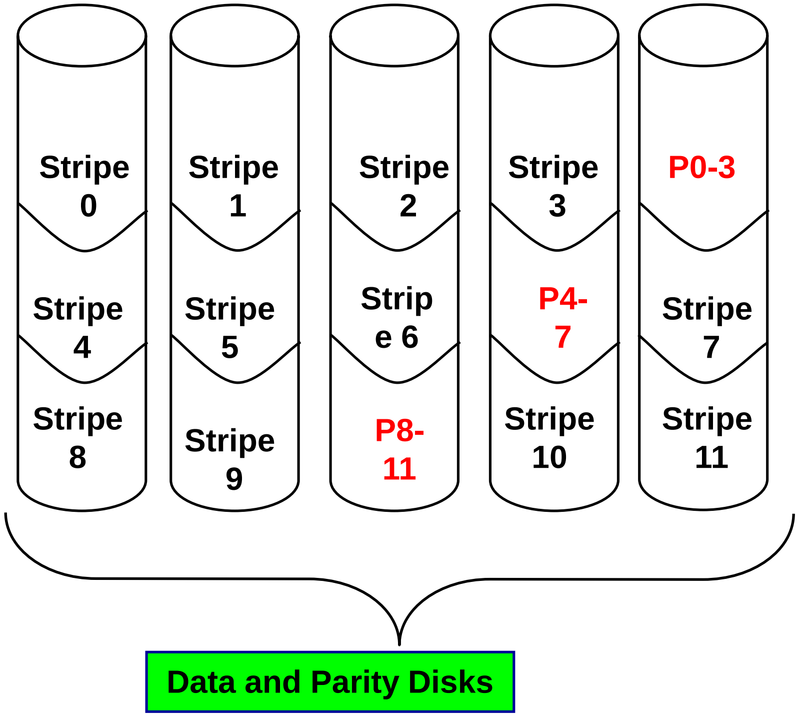
Raid Level 4

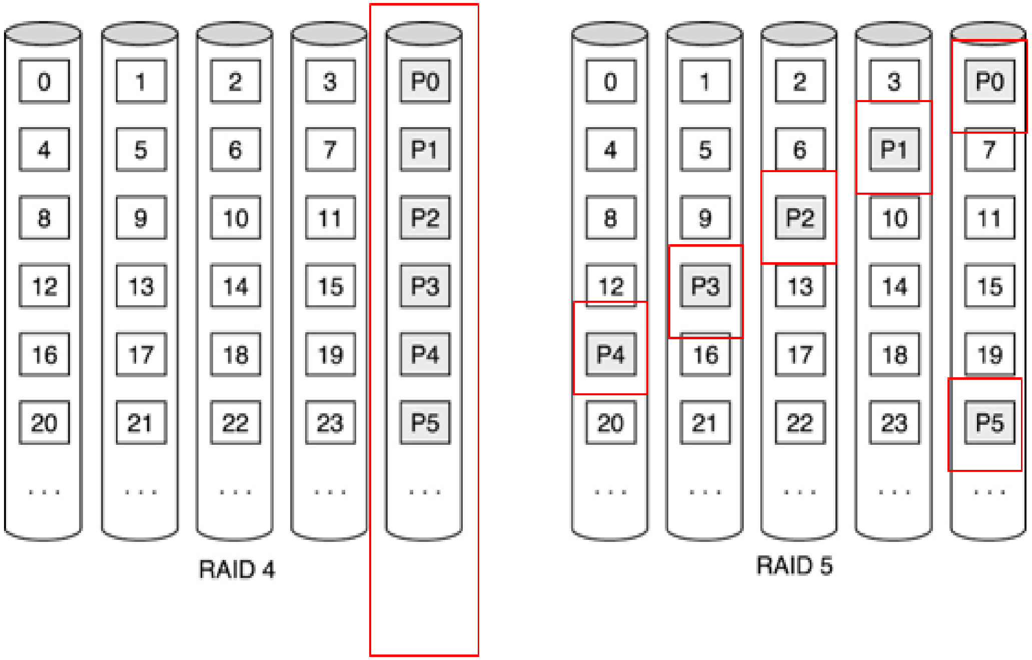
- Block-interleaved parity
  - RAID 4 is very similar to RAID 3. The main difference is the way of sharing data. They are divided into blocks and written on disks.
  - One disk is a parity disk, keeps parity blocks
  - Parity block at position X is the parity for all blocks whose position is X on any of the data disks
  - A read accesses only the data disk where the data is there.
  - A write must update the data block and its parity block.
  - Can recover from an error on only one disk.
  - Note that with N disks we have N-1 data disks and only one parity disk, but can still recover when one disk fails
  - But write performance worse than with one disk (all writes must read and then write the parity disk)



Raid Level 5

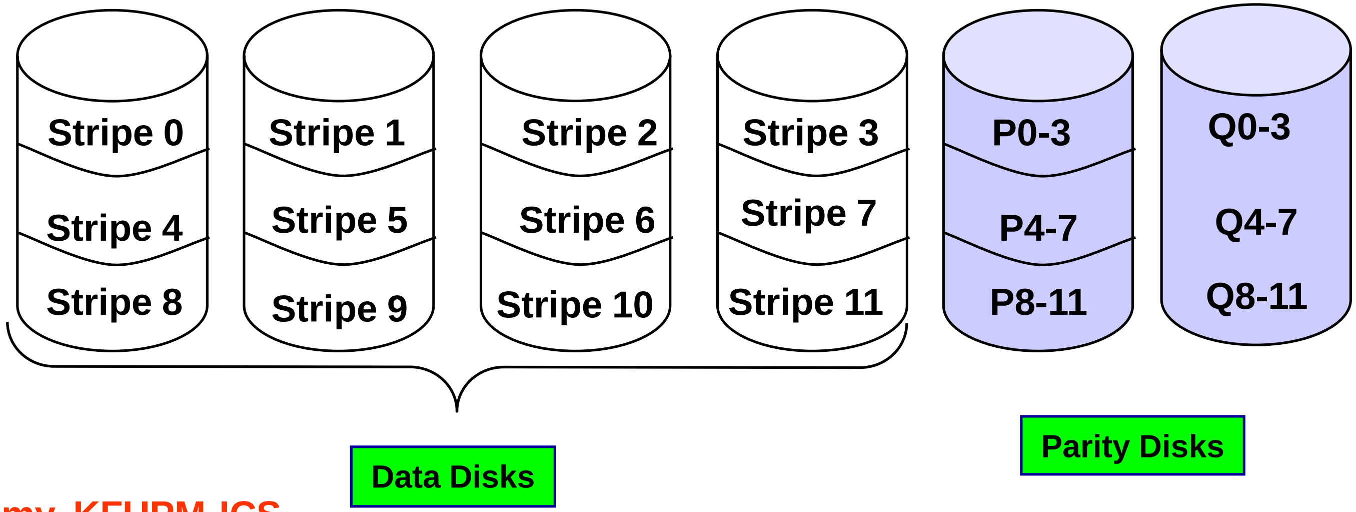
- Like RAID 4, but parity blocks distributed to all disks
- Read accesses only the data disk where the data is
- A write must update the data block and its parity block
    - But now all disks share the parity update load





Raid Level 6

- Level 5 with an extra parity bit
- Two different (P and Q) check blocks
  - Each protection group has
    - N-2 data blocks
    - One parity block
    - Another check block (not the same as parity)
- Can recover when two disks are lost
  - Think of P as the sum and Q as the product of D blocks
  - If two blocks are missing, solve equations to get both back
- More space overhead (only N-2 of N are data)
- More write overhead (must update both P and Q)
  - P and Q still distributed like in RAID 5



Selecting a RAID Level

- If a disk fails, the time to rebuild its data can be significant and will vary with the RAID level used.
- RAID level 0 is used in high-performance applications where data loss is not critical.
- Rebuilding is easiest for RAID level 1. Simply copy the data from another disk.
- **RAID level 1 is popular for applications that require high reliability with fast recovery**.
- The combination of RAID levels 0 and 1 (RAID 0 + 1) is used for applications where performance and reliability are very important, for e.g. banking system's databases.
- Due to RAID 1's high space overhead, RAID level 5 is often preferred for storing large volumes of data.
- RAID level 6 is not supported currently by many implementations, but should offer better reliability than level 5.

The End!!

Thank you

Any Questions?