Marwadi University

3140702

Operating System

Unit – 7

I/O Management & Disk Scheduling

Topics to be covered

- I/O Management
- Principles of I/O Hardware:
  - I/O devices
  - Device controllers
  - Direct memory access
- Secondary-Storage Structure:
  - Disk structure
  - Disk scheduling algorithm
- RAID

I/O devices

- External devices that engage in I/O with computer systems can be grouped into 3 categories:
  - **Human readable**, suitable for communicating with the computer user.
    Examples: printers, terminals, video display, keyboard, mouse
  - **Machine readable**, suitable for communicating with electronic equipment.
    Examples: disk drives, USB keys, sensors, controllers
  - **Communication**, suitable for communicating with remote devices.
    Examples: modems, digital line drivers

I/O devices

- Devices differ in several areas:
  1. **Data Rate**: there may be differences of magnitude between the data transfer rates
  2. **Application**: the use to which a device is put has an influence on the software
  3. **Complexity of Control**: the effect on the operating system is filtered by the complexity of the I/O module that controls the device
  4. **Unit of Transfer**: data may be transferred as a stream of bytes or characters or in larger blocks
  5. **Data Representation**: different data encoding schemes are used by different devices
  6. **Error Conditions**: the nature of errors, the way in which they are reported, their consequences, and the available range of responses differs from one device to another

Organization of I/O functions

- Three techniques for performing I/O are:

1. **Programmed I/O**, the **processor issues an I/O command on behalf of a process** to an I/O module; that **process then busy waits for the operation to be completed** before proceeding.

2. **Interrupt-driven I/O**, the **processor issues an I/O command on behalf of a process**
   - **if non-blocking – processor continues to execute instructions** from the process that issued the I/O command
   - **if blocking – the next instruction the processor executes is from the OS**, which will put the current process in a blocked state and schedule another process

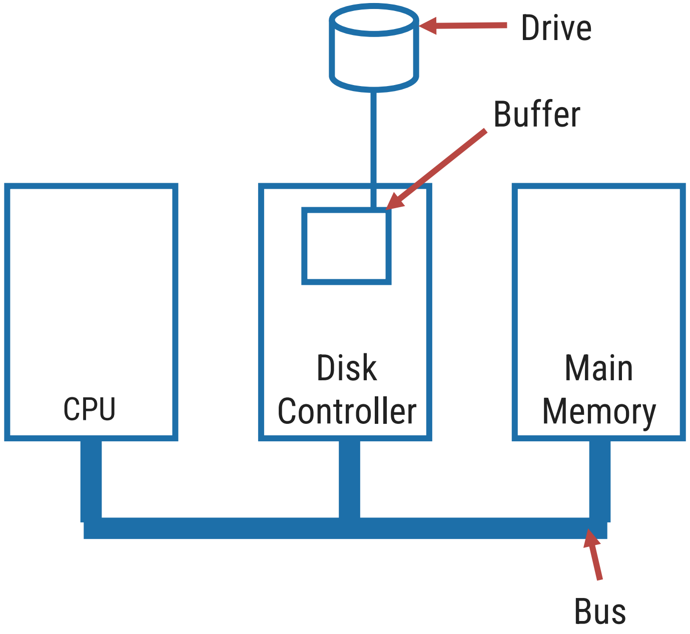
3. **Direct Memory Access (DMA)**, a **DMA module controls the exchange of data** between main memory and an I/O module

Direct Memory Access

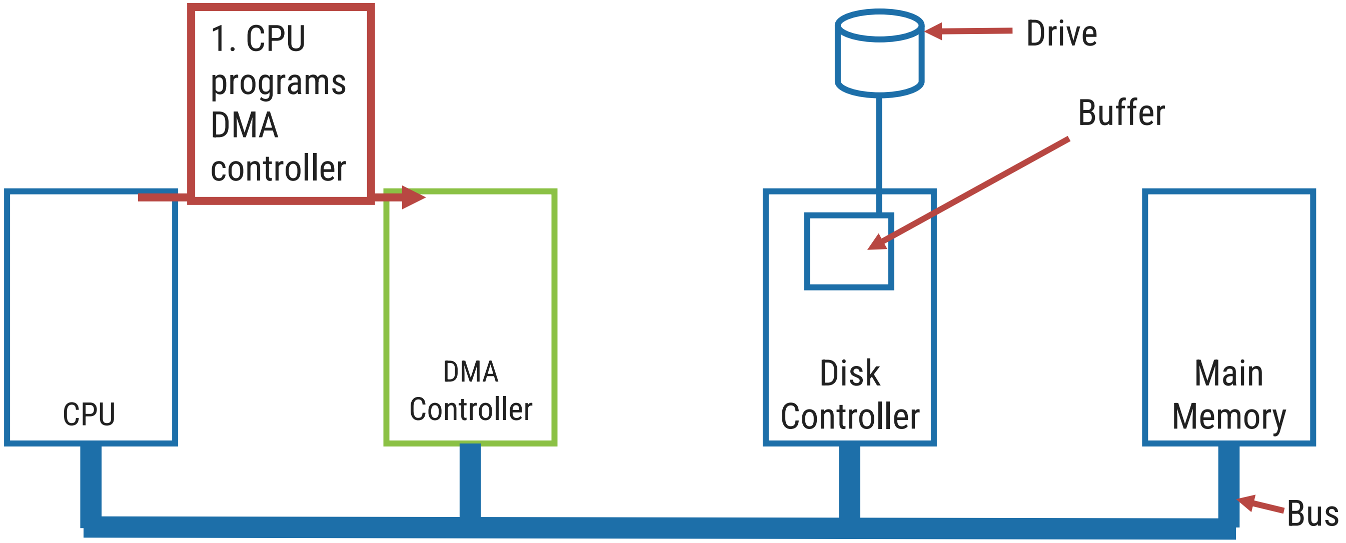
- Feature of computer systems that **allows certain hardware subsystems to access main memory (RAM), independent of the central processing unit (CPU)**.
- Without DMA, when the CPU is using programmed input/output, it is typically fully occupied for the entire duration of the read or write operation and is thus unavailable to perform other work.
- With DMA, the **CPU first initiates the transfer, then it does other operations while the transfer is in progress**, and it finally **receives an interrupt from the DMA controller when the operation is done**.
- This **feature is useful when the CPU needs to perform useful work while waiting for a relatively slow I/O data transfer**.
- Many hardware systems such as **disk drive controllers, graphics cards, network cards and sound cards use DMA**.

Disk read-write Without a DMA

- The **disk controller reads the block from the drive serially**, bit by bit, **until the entire block is in the controller's buffer**.
- Next, it **computes the checksum** to verify that no read errors have occurred.
- Then the **controller causes an interrupt**, so that OS can read the block from controller's buffer (a byte or a word at a time) by executing a loop.
- **After reading every single part of the block** from controller device register, the **operating system will store them into the main memory**.

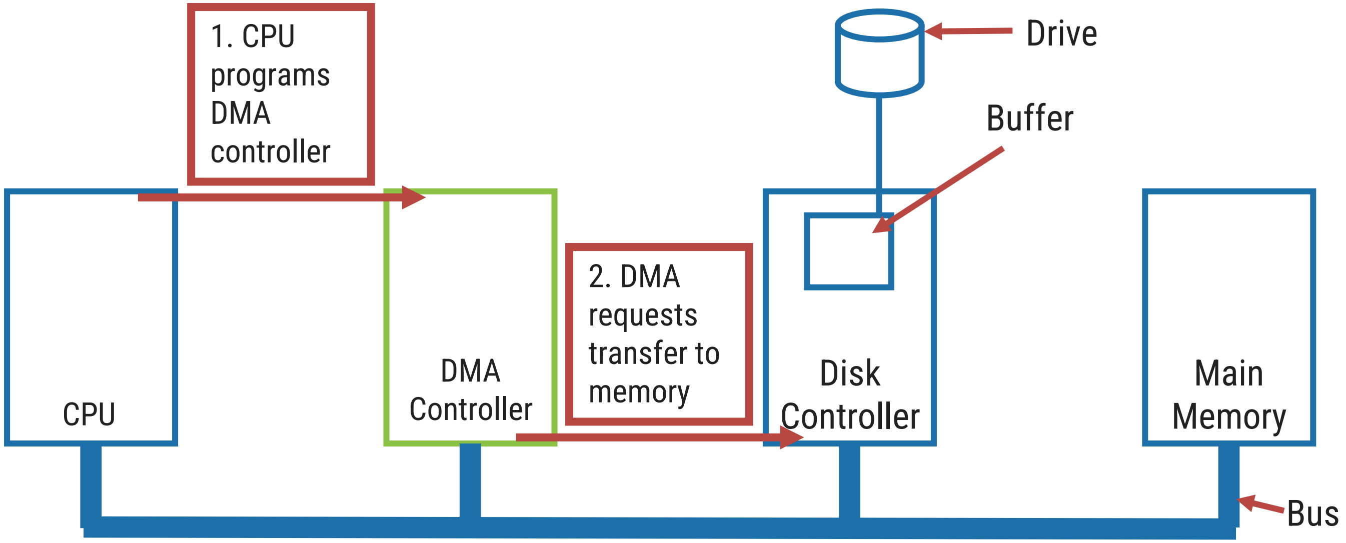


Disk read-write using DMA



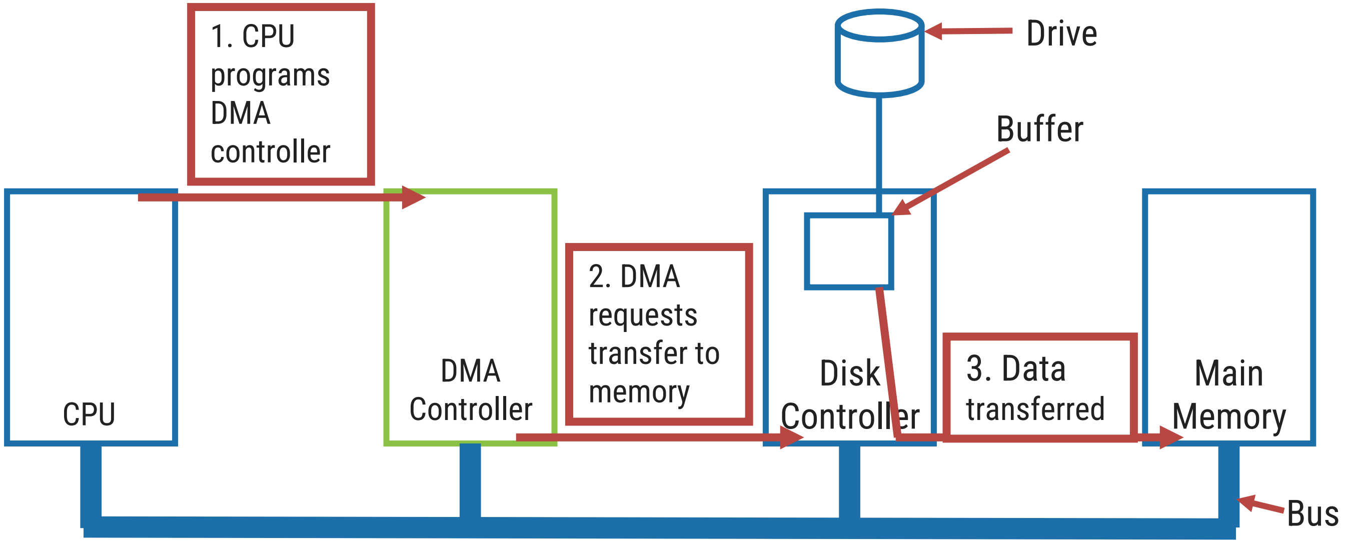
- Step 1: First the CPU programs the DMA controller by setting its registers, so it knows what to transfer where.
- It also issues a command to the disk controller telling it to read data from the disk into its internal buffer and verify the checksum.
- When valid data are in the disk controller’s buffer, DMA can begin.

Disk read-write using DMA


1. CPU programs DMA controller
CPU
DMA Controller
2. DMA requests transfer to memory
Drive
Buffer
Disk Controller
Main Memory
Bus

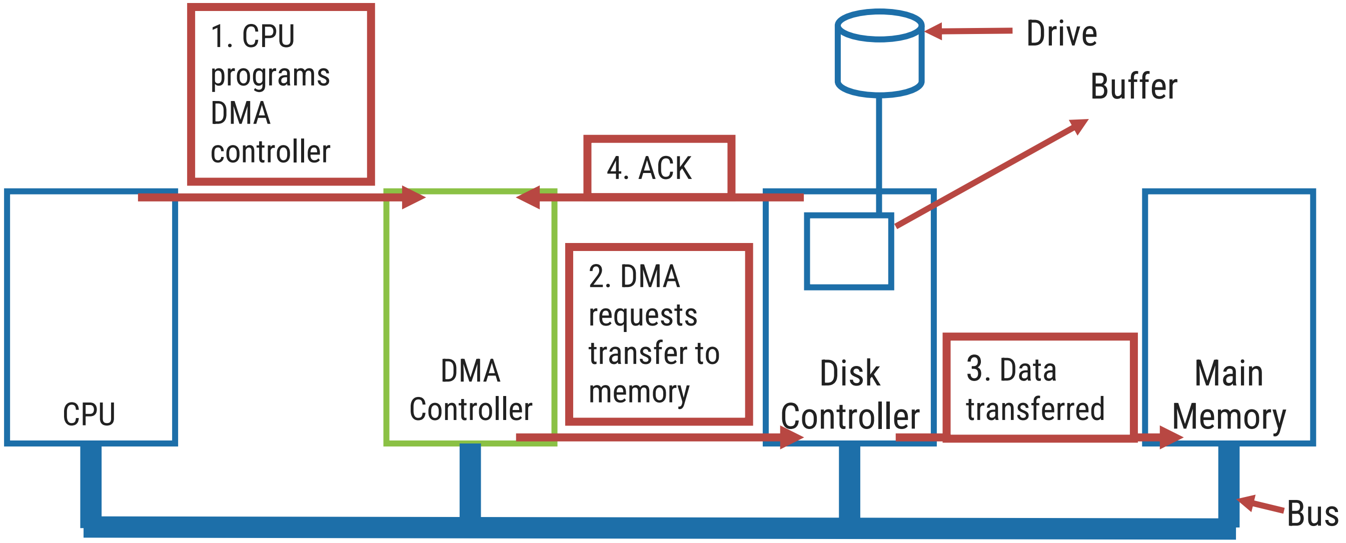
- Step 2: The DMA controller initiates the transfer by issuing a read request over the bus to the disk controller.
- This read request looks like any other read request, and the disk controller does not know (or care) whether it came from the CPU or from a DMA controller.

Disk read-write using DMA



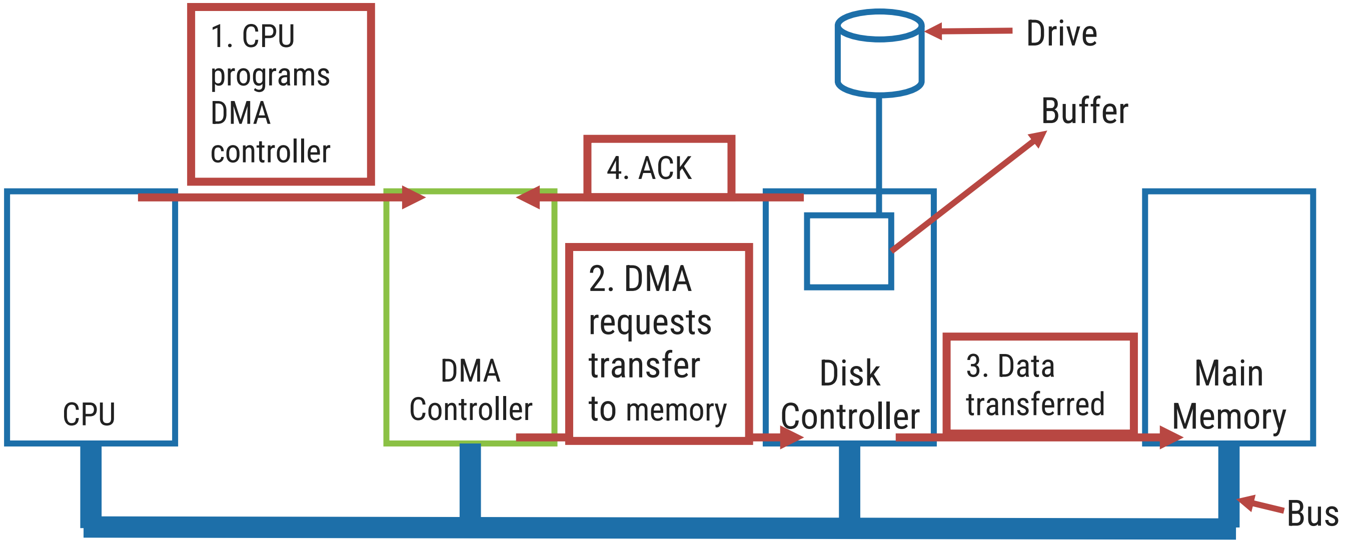
- Typically, the memory address to write to is on the bus' address lines, so when the disk controller fetches the next word from its internal buffer, it knows where to write it.
- Step 3: The write to memory is another standard bus cycle.

Disk read-write using DMA



- Step 4: When the write is complete, the disk controller sends an acknowledgement signal to the DMA controller, also over the bus.
- The DMA controller then increments the memory address to use and decrements the byte count.
- If the byte count is still greater than 0, steps 2 to 4 are repeated until it reaches 0.

Disk read-write using DMA



- At that time, the DMA controller interrupts the CPU to let it know that the transfer is now complete.
- When the OS starts up, it does not have to copy the disk block to memory; it is already there.

Operating System Design issues

- Design objectives: Two objectives are paramount in designing the I/O facility.

1. Efficiency
   - Efficiency is important because I/O operations often **form a bottleneck** in a computing system.
   - Most **I/O devices are extremely slow** compared with main memory and the processor.
2. Generality
   - It is desirable to **handle all devices in a uniform manner**.
   - Applies to the **way processes view I/O devices** and the **way the operating system manages I/O devices** and operations.
   - **Because of the diversity of device characteristics**, it is **difficult in practice to achieve true generality**.
   - What can be done is to **use a hierarchical**, **modular approach** to the design of the I/O function.
   - This approach **hides most of the details of device I/O** in lower-level routines so that user processes and upper levels of the OS see devices in terms of **general functions, such as read, write, open, close, lock, and unlock**.

Buffering

- **Perform input transfers in advance** of requests being made and **perform output transfers some time after** the request is made is called buffering.
- Types of I/O devices:
  - **Block oriented**: A block-oriented device **stores information in blocks** that are usually of **fixed size**, and **transfers are made one block at a time**.

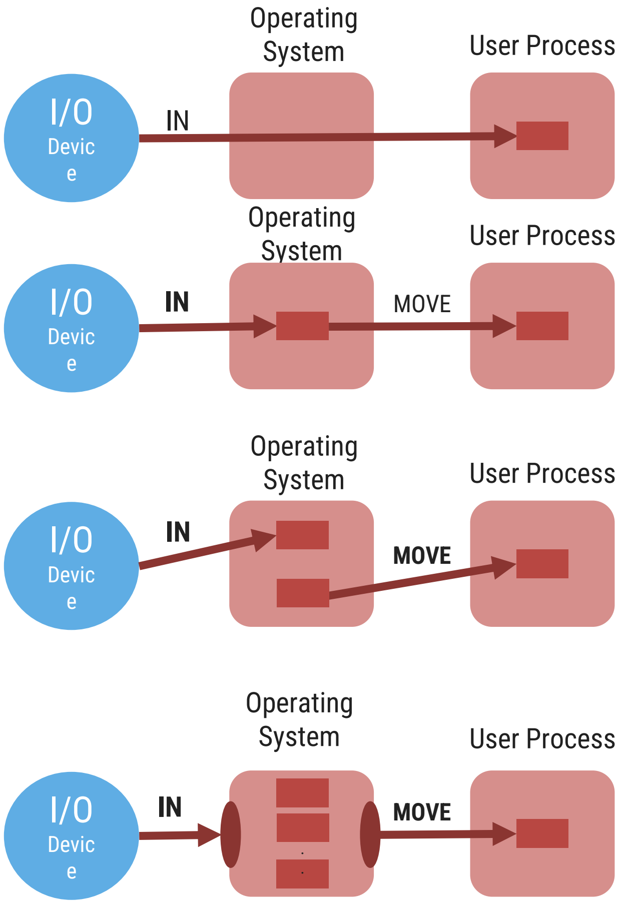
    Generally, it is possible to reference data by its block number.

    Hard disks, floppy disks and optical drives such as DVD-ROM and CD-ROM are examples of block-oriented devices.

  - **Stream oriented**: A stream-oriented device **transfers data in and out as a stream of bytes**, with **no block structure**.

    Terminals, printers, communications ports, keyboard, mouse and other pointing devices, and most other devices that are **not secondary storage** are stream oriented.

I/O buffering



- **Without a buffer**, Operating system **directly accesses the device when it needs**

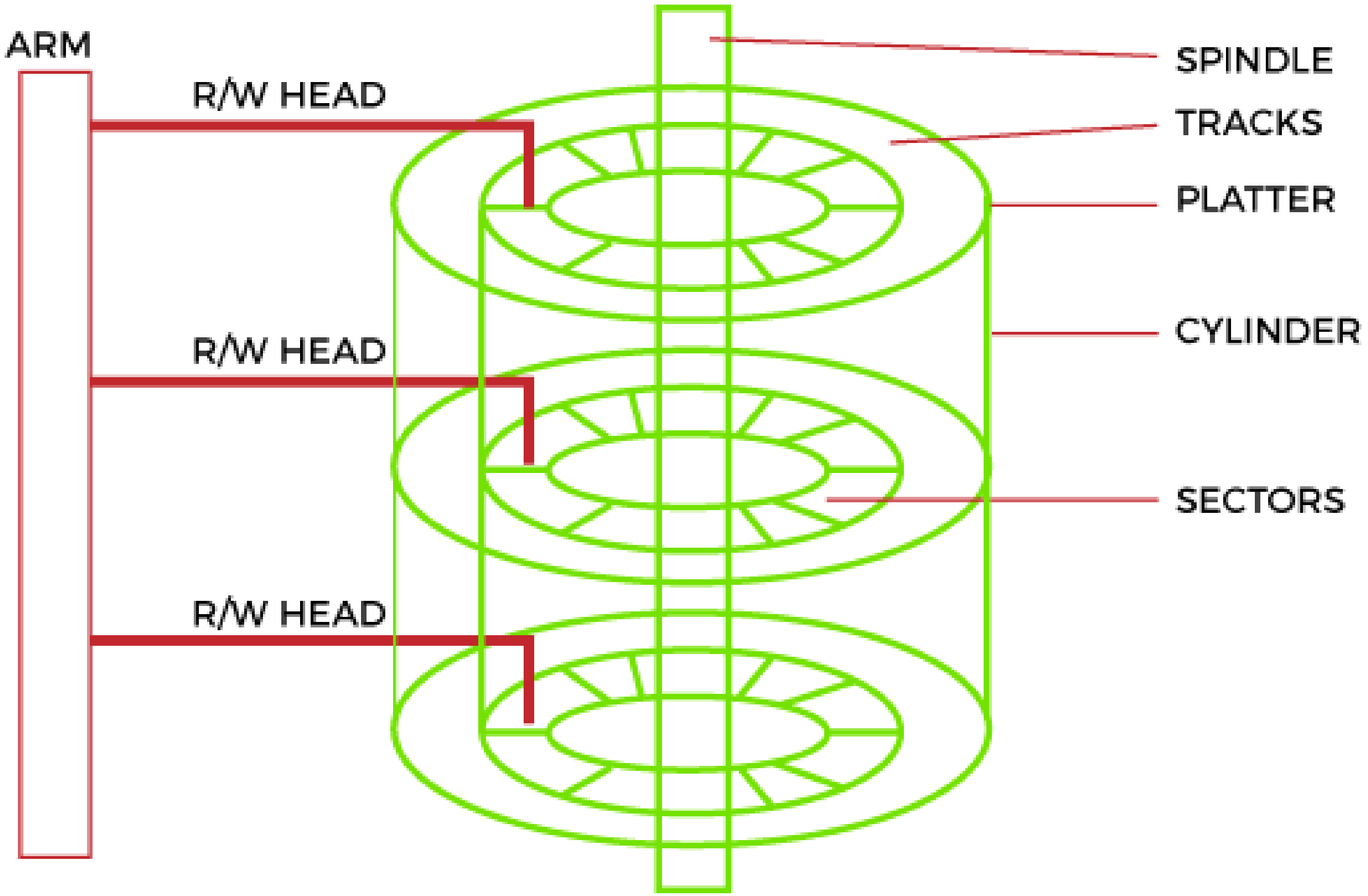
- **Single buffer**, Operating system **assigns a buffer in the main memory for an I/O request**

- **Double buffer**, Operating system **use two system buffers instead of one,** also known as **buffer swapping**. A **process can transfer data to or from one buffer while the operating system empties or fills the other buffer**

- **Circular buffer**, Operating system **uses two or more buffers. Each individual buffer is one unit in a circular buffer. Used when I/O operation must keep up with process**

Disk Architecture

- **Platter → Surface → Track → Sector → Data .**



Disks

- It is relatively **permanent** and can hold **large quantities** of data. Its **access time is slow** compared with that of main memory and magnetic disk. provide the bulk of secondary storage for modern computer system.
- **disk platter:** has a flat circular shape, like a CD. The two surfaces of a platter are covered with a magnetic material. We store information by recording it magnetically on the platters.
- The surface of a platter is logically divided into circular **tracks** which are subdivided into **sectors**. The set of tracks that are at one arm position makes up a **cylinder**.

Disk Access time

- **Seek Time:** Seek time is the time taken to locate the disk arm to a specified track where the data is to be read or write. So, the disk scheduling algorithm that gives minimum average seek time is better.

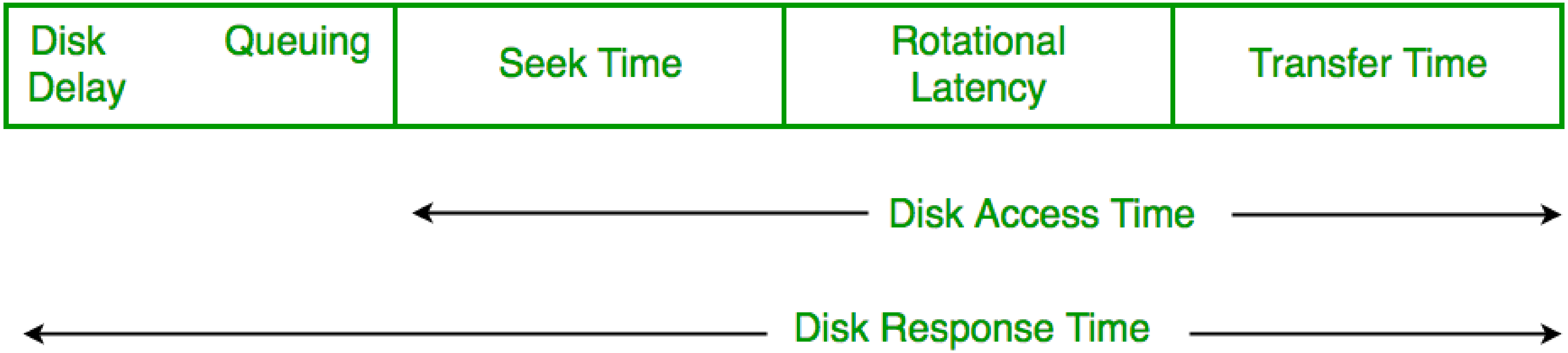
- **Rotational Latency:** Rotational Latency is the time taken by the desired sector of disk to rotate into a position so that it can access the read/write heads. So the disk scheduling algorithm that gives minimum rotational latency is better.

- **Transfer Time:** Transfer time is the time to transfer the data. It depends on the rotating speed of the disk and number of bytes to be transferred.

Disk Access time

- **Disk Access Time:** Disk Access Time is:
- **Disk Access Time** = Seek Time + Rotational Latency + Transfer Time
- **Total Seek Time** = Total head Movement * Seek Time

- **Disk Response Time:** Response Time is the average of time spent by a request waiting to perform its I/O operation. *Average Response time* is the response time of the all requests



Disk Queuing Delay	Seek Time	Rotational Latency	Transfer Time

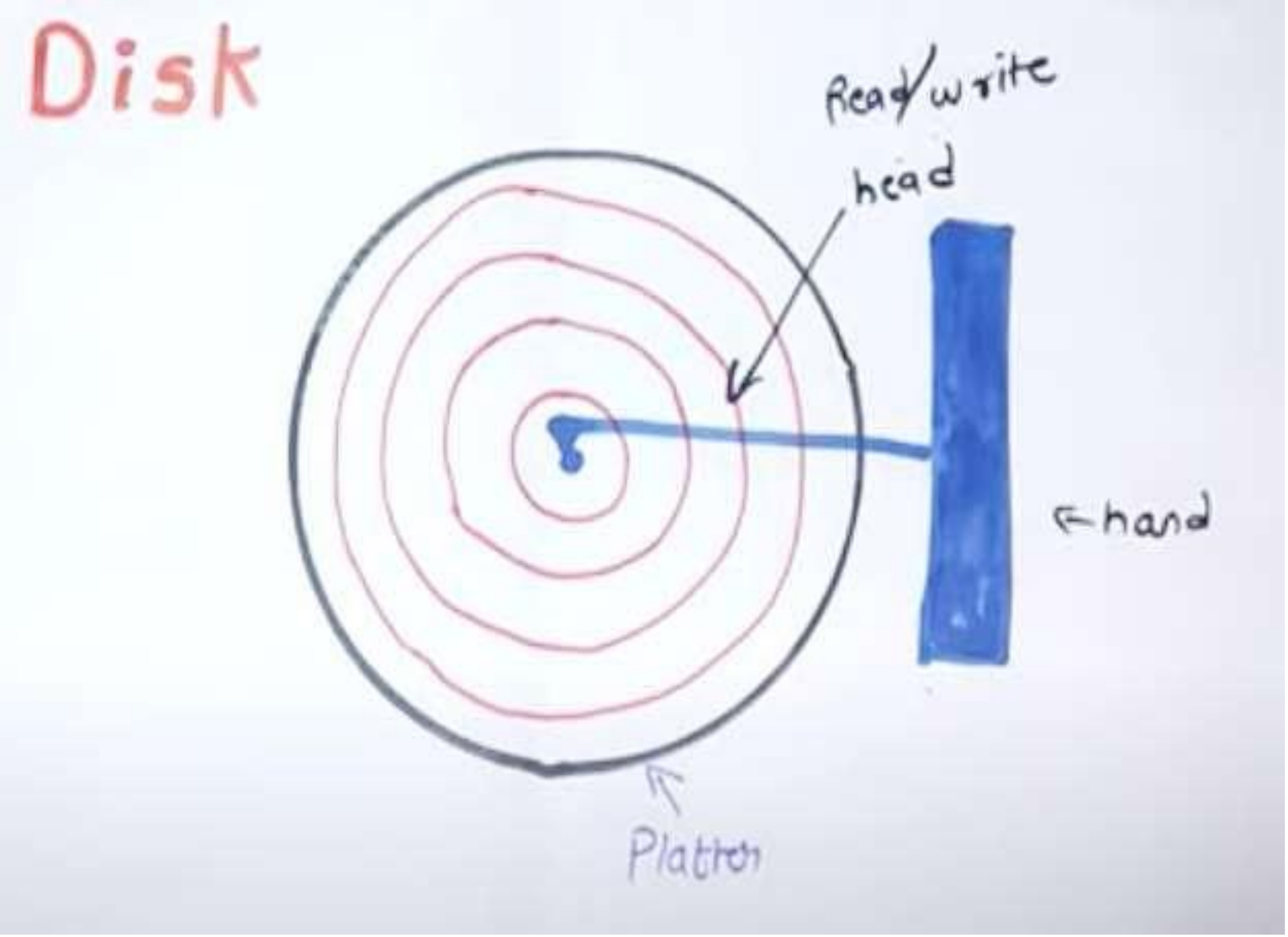
Disk Access Time

Disk Response Time

Disk Arm Scheduling Algorithm

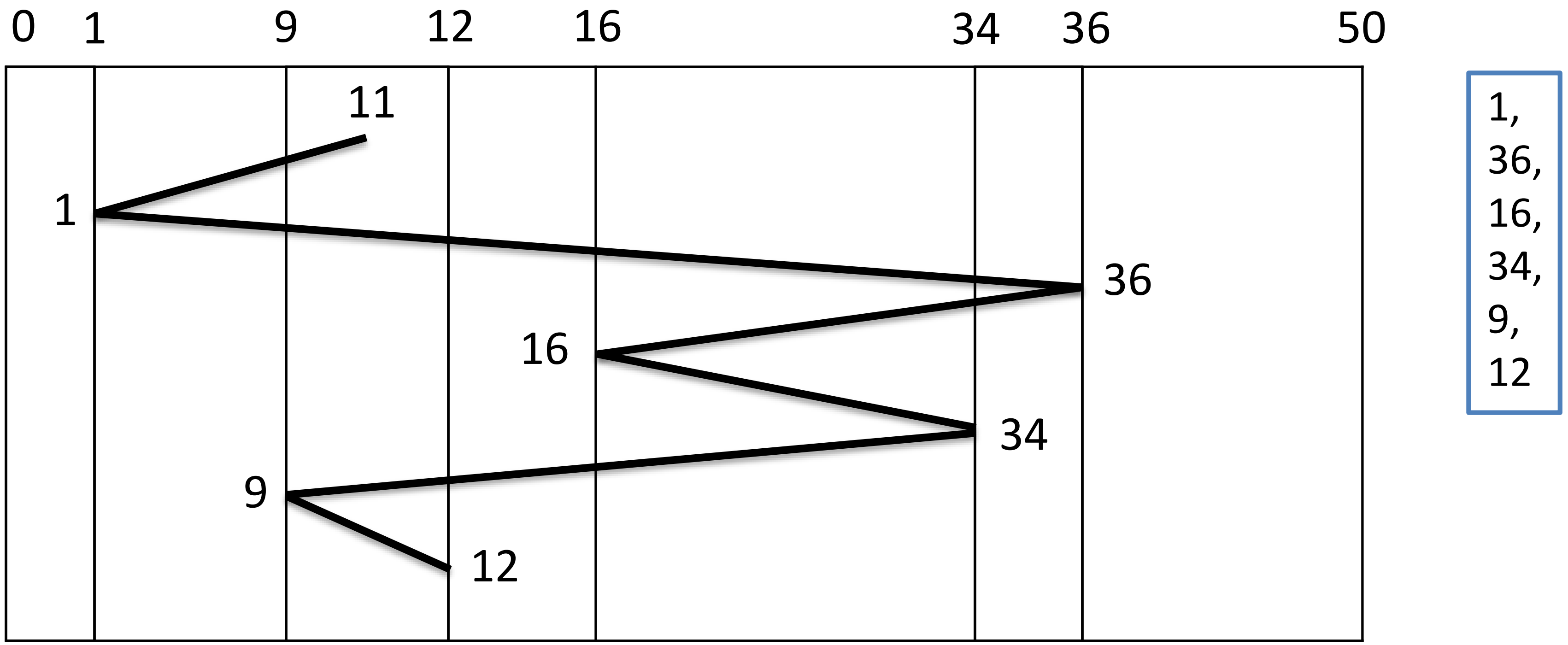
Disk Arm Scheduling Algorithm

- Various types of disk arm scheduling algorithms are available to decrease mean seek time.
  1. FCFS (First come first serve)
  2. SSTF (Shorted seek time first)
  3. SCAN
  4. C-SCAN
  5. LOOK (Elevator)
  6. C-LOOK


Disk
Read/write
head
← hand
Platter

FCSC (First come first serve)

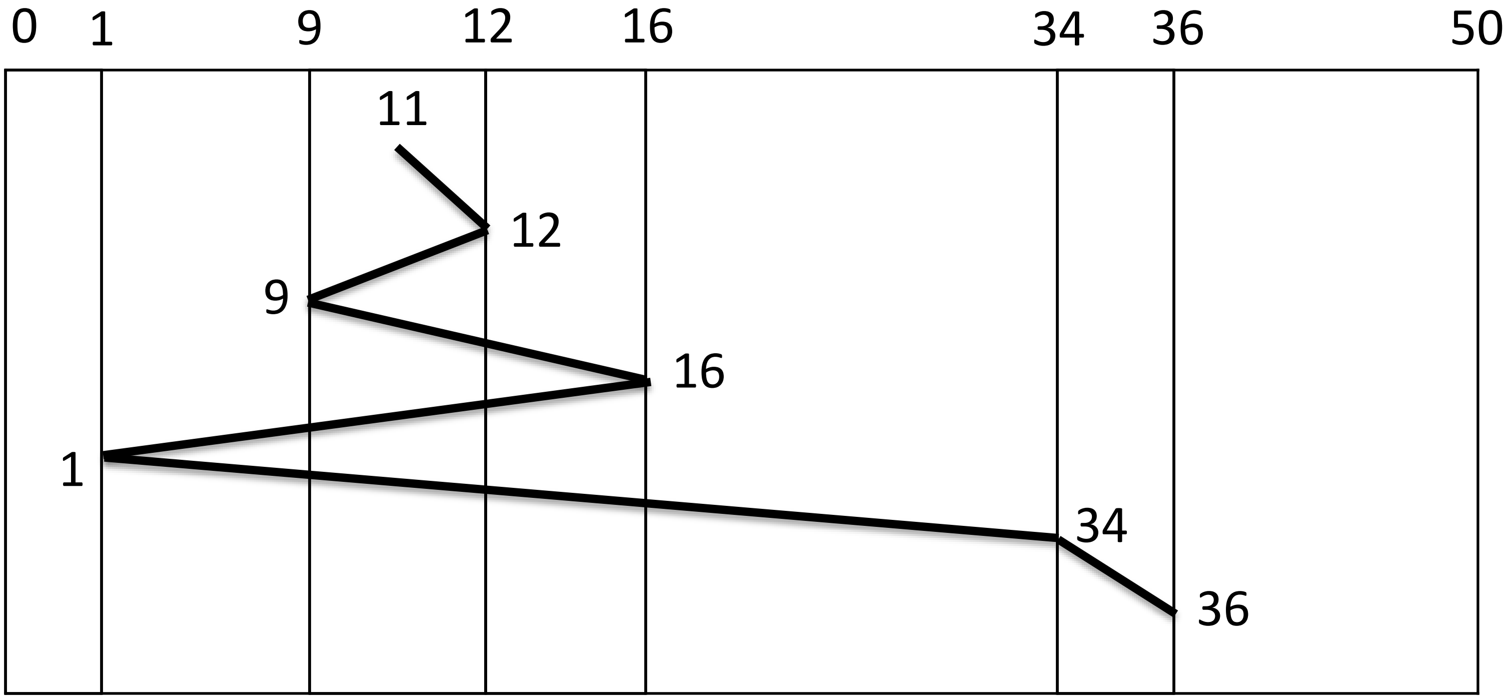
- Here **requests are served in the order of their arrival**.



- Disk movement will be 11, 1, 36, 16, 34, 9 and 12.
- Total cylinder movement: (11-1) + (36-1) + (36-16) + (34-16) + (34-9) + (12-9) = 111

SSTF (Shortest seek time first)

- We can minimize the disk movement by **serving the request closest to the current position** of the head.



1,
36,
16,
34,
9,
12

- Disk movement will be 11, 12, 9, 16, 1, 34, 36.
- Total cylinder movement: (12-11) + (12-9) + (16-9) + (16-1) + (34-1) + (36-34) = 61

Disk Scheduling Algorithm

Q1 Track no given= 82, 170,43,140,24,16,190

- A disk contains 0-199 tracks
- Current position of r/w head is 50
- Calculate the total no of track movement of your r/w head?

Solution-

FCFS= and **SSTF=**

SCAN=

C-SCAN

LOOK

C-LOOK

Disk Scheduling Algorithm

Q1 Track no given= 82, 170,43,140,24,16,190

- A disk contains 0-199 tracks
- Current position of r/w head is 50 direction is towards larger value of track
- Calculate the total no of track movement of your r/w head?

Solution-

FCFS=642 and **SSTF=208**

SCAN=

C-SCAN

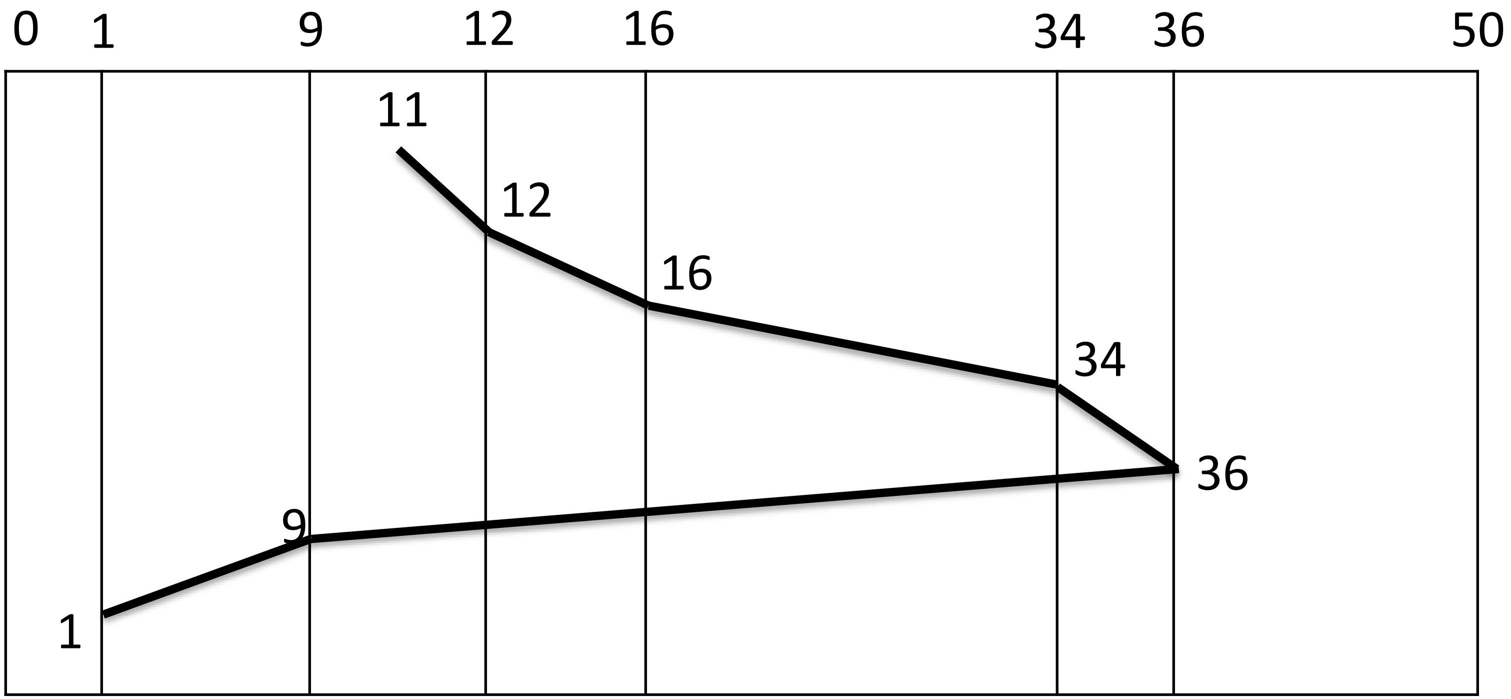
LOOK

C-LOOK

LOOK(Elevator)

- **Keep moving in the same direction until there are no more outstanding requests pending in that direction, then algorithm switches the direction**.
- After switching the direction the arm will move to handle any request on the way. Here **first go it moves in up direction then goes in down direction**.
- This is also called as **elevator algorithm**.
- In the elevator algorithm, the **software maintains 1 bit: the current direction bit, which takes the value either UP or DOWN**.

LOOK (Elevator)



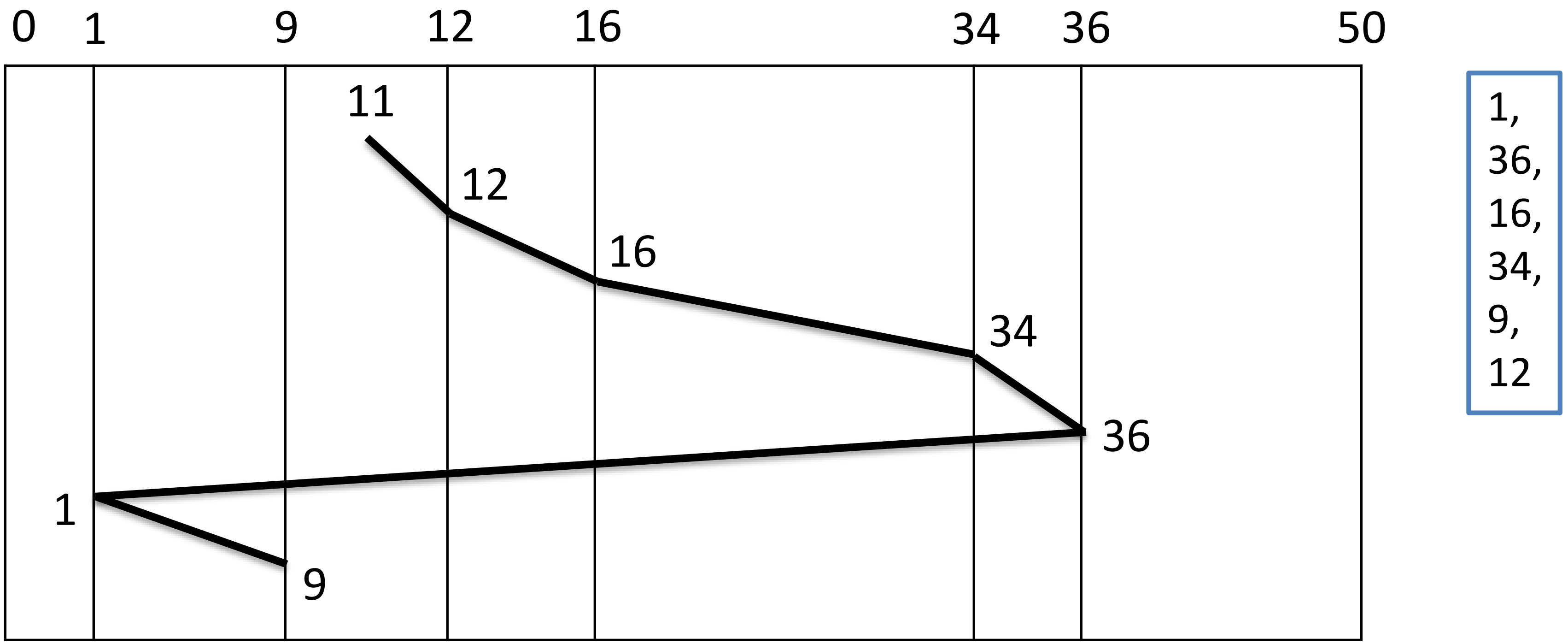
1,
36,
16,
34,
9,
12

- Disk movement will be 11, 12, 16, 34, 36, 9, 1.
- Total cylinder movement: (12-11) + (16-12) + (34-16) + (36-34) + (36-9) + (9-1)=60

C-LOOK

- **Keep moving in the same direction until there are no more outstanding requests pending in that direction, then algorithm switches direction**.
- **When switching occurs the arm goes to the lowest numbered cylinder** with pending requests and **from there it continues moving in upward direction again**.

C-LOOK

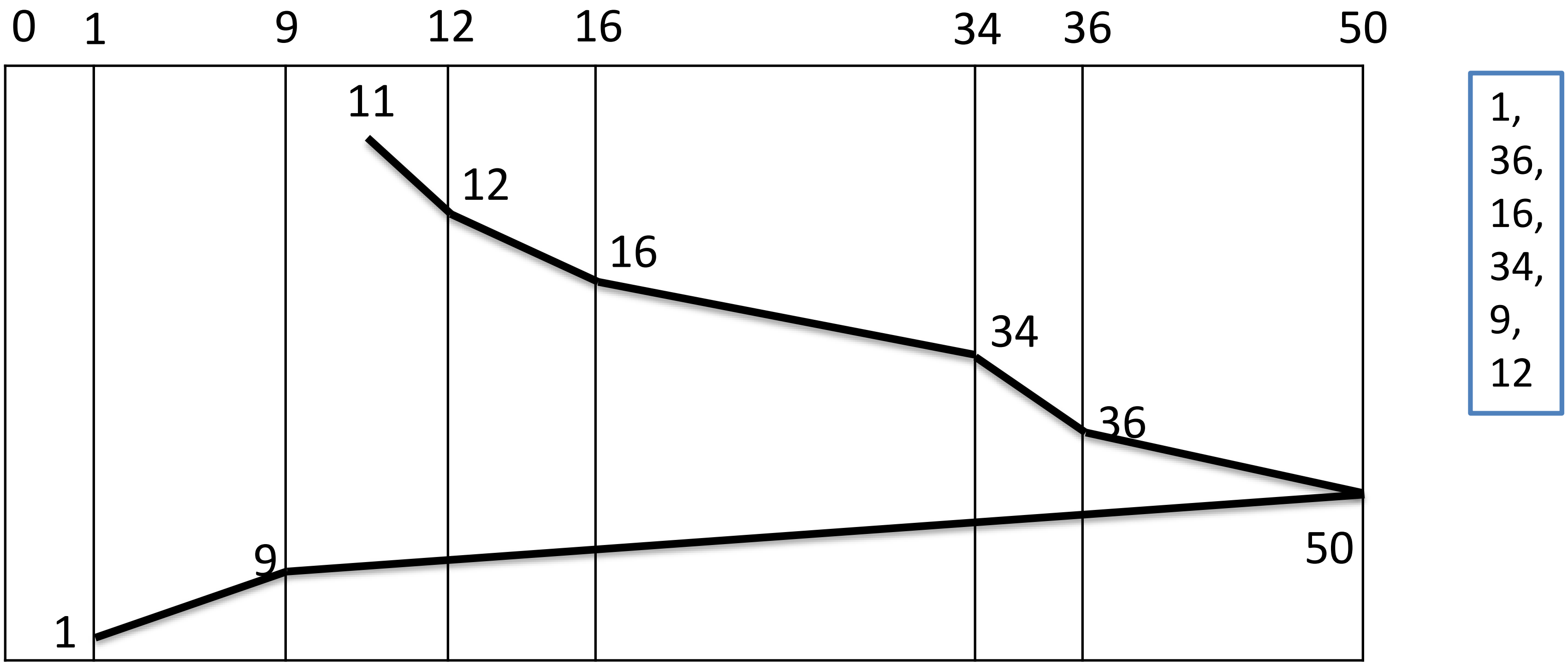


- Disk movement will be 11, 12, 16, 34, 36, 1, 9.
- Total cylinder movement: (12-11) + (16-12) + (34-16) + (36-34) +(36-1)+(9-1)=68

SCAN

- **From the current position disk arm starts in up direction and moves towards the end, serving all the pending requests until end**.
- At that end arm **direction is reversed (down) and moves towards the other end serving the pending requests on the way**.

SCAN

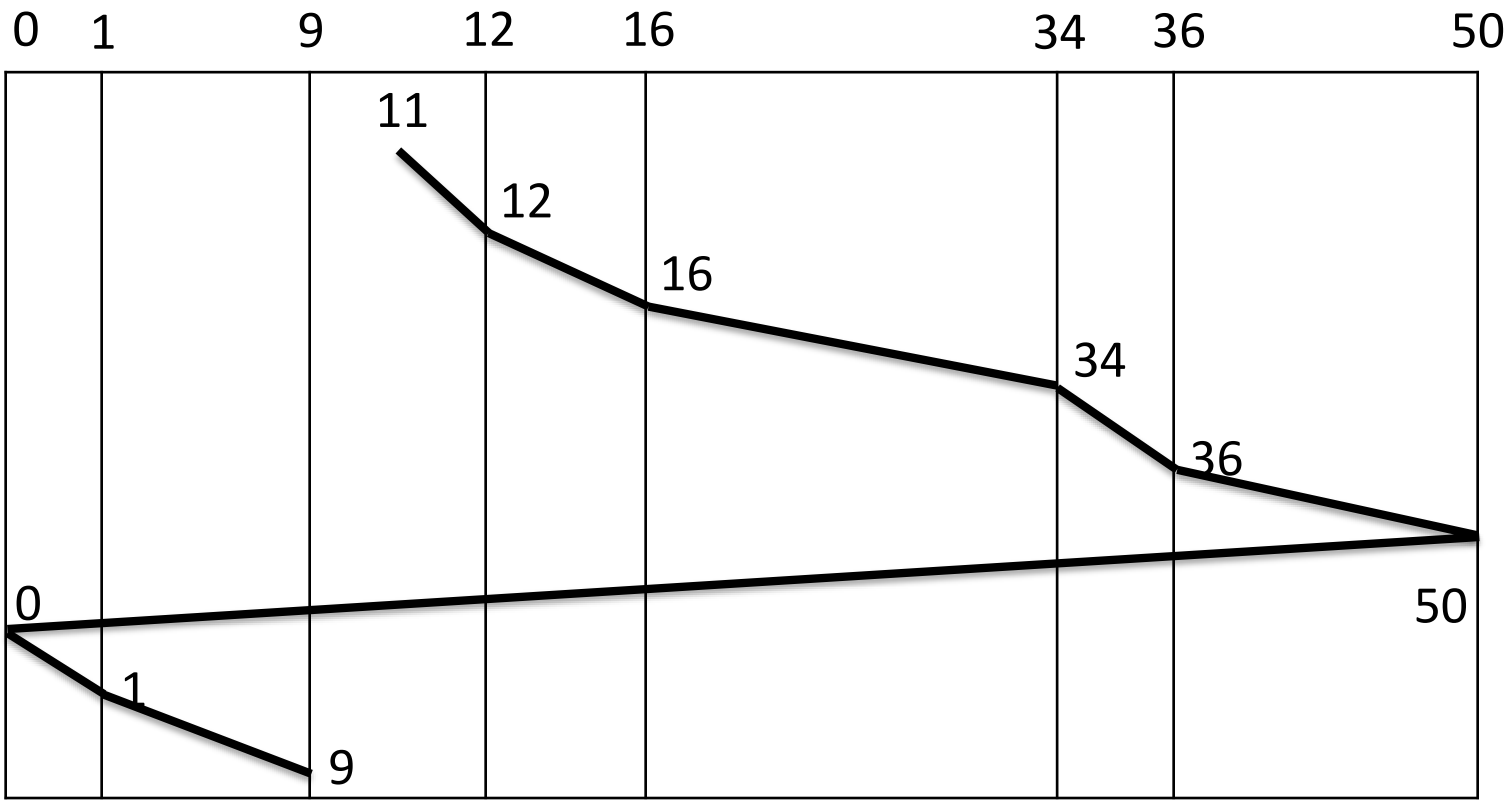


- Disk movement will be 11, 12, 16, 34, 36, 50, 9, 1.
- Total cylinder movement: (12-11) + (16-12) + (34-16) +(36-34) +(50-36) + (50-9) + (9-1) = 88

C-SCAN

- From the **current position disk arm starts in up direction and moves towards the end, serving request until end**.
- At the end the arm **direction is reversed (down), and arm directly goes to other end and again continues moving in upward direction**.

C-SCAN



1,
36,
16,
34,
9,
12

- Disk movement will be 11, 12, 16, 34, 36, 50, 0, 1,9.
- Total cylinder movement: (12-11) + (16-12) + (34-16) +(36-34) +(50-36) + (50-0) + (1-0)+ (9-1) = 98

Disk Scheduling Algorithm

Q- Track no given= 55,58,39,18,90,160,150,38,184

- A disk contains 0-199 tracks
- Current position of r/w head is 100 and direction is towards larger value of track
- Calculate the total no of track movement of your r/w head for FCFS, SSTF, SCAN,C-SCAN, LOOK AND C-LOOK?

Disk Scheduling Algorithm

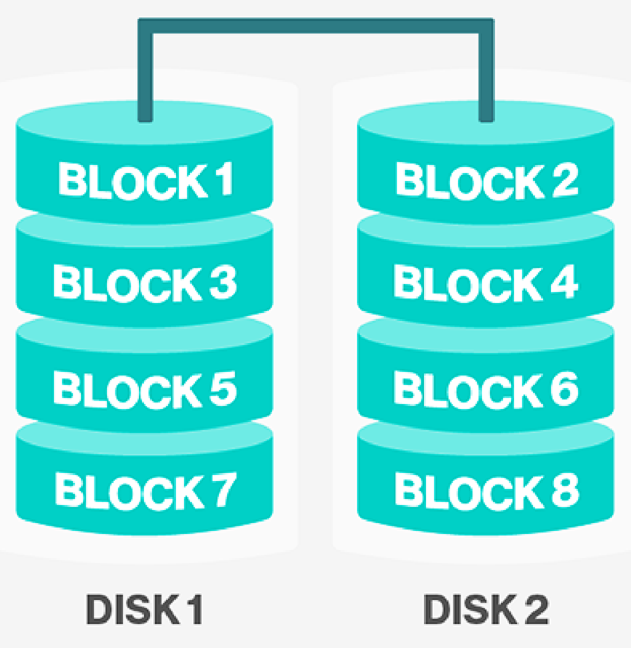
- Solution
- **FCFS = 498**
- **SSTF = 248**
- **SCAN = 280**
- **C-SCAN = 388**
- **LOOK = 250**
- **C-LOOK = 322**

RAID

- RAID (**Redundant Array of Independent Disks**)
- RAID is a **data storage virtualization technology that combines multiple physical disk drive components into a single logical unit for the purposes of data redundancy, performance improvement, large storage capacity or all**.
- **Data is distributed across the drives** in one of several ways, referred to as RAID levels, depending on the required level of redundancy and performance.
- All RAID have the property that the **data are distributed over drives, to allow parallel operation**.
- There are **7 levels of RAID**.

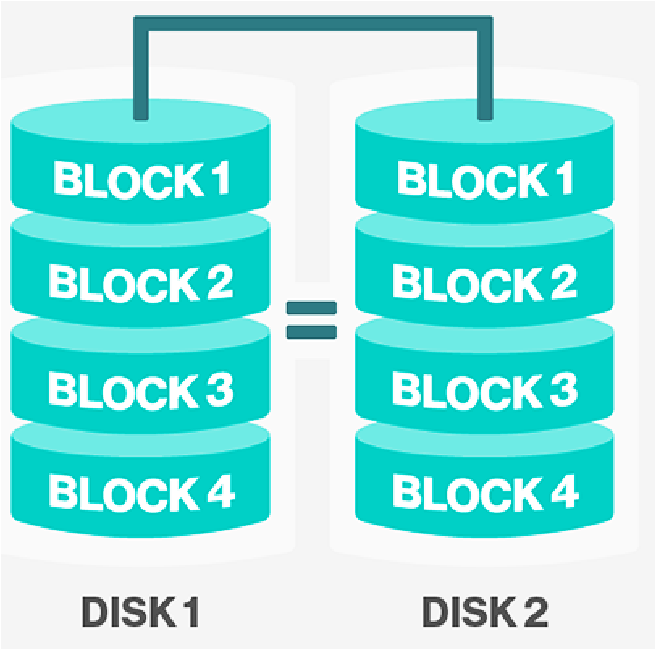
RAID 0 (Striping)

- It **splits data (file) into blocks of data**.
- **Stripe the blocks across disks** in the system.
- In the diagram to the right, the **odd blocks are written to disk 1 and the even blocks to disk 2**.
- It is **easy to implement**.
- **No parity calculation** overhead is involved.
- It **provides good performance** by spreading the load across many channels and drives.
- It **provides no redundancy or error detection**.
- **Not true RAID because there is no fault tolerance**. The failure of just one drive will result in all data in an array being lost.
- After certain amount of drives, performance does not increase significantly.
- It **requires minimum 2 drives** to implement.



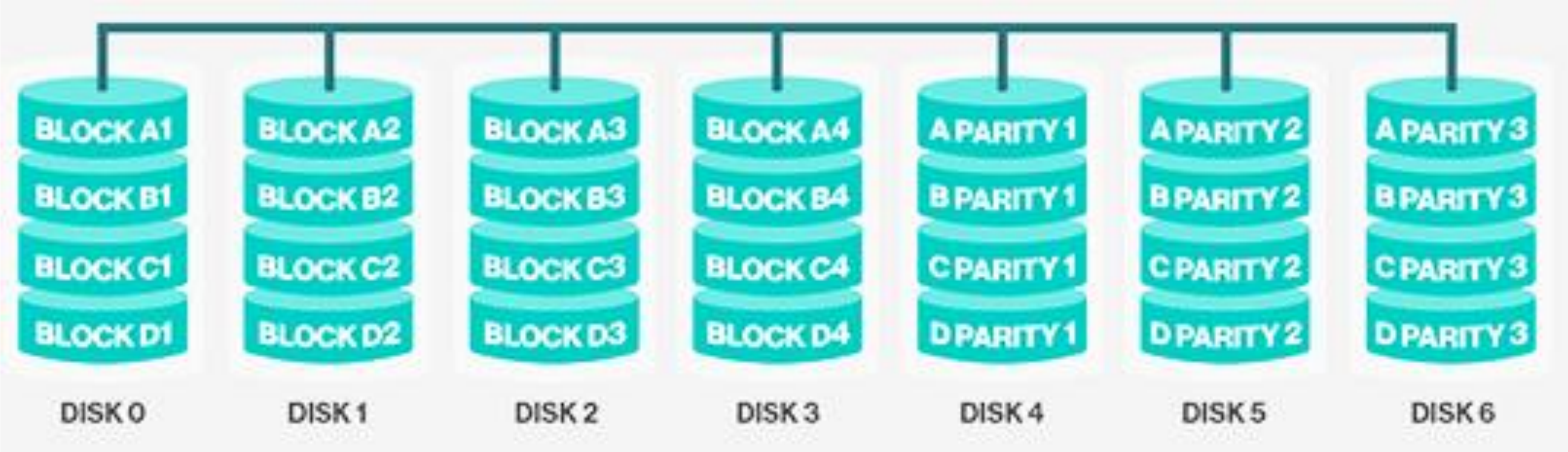
RAID 1 (Mirroring)

- A **complete file is stored on a single disk**.
- A **second disk contains an exact copy of the file**.
- It **provides complete redundancy** of data.
- **Read** performance can be **improved**
  - ✓ same file data can be read in parallel
- **Write** performance **suffers**
  - ✓ must write the data out twice
- **Most expensive** RAID implementation.
- It **requires twice as much storage space**.
- In case a **drive fails, data do not have to be rebuild**, they just have to be copied to the replacement drive.
- The main **disadvantage is that the effective storage capacity is only half of the total drive capacity** because all data get written twice.



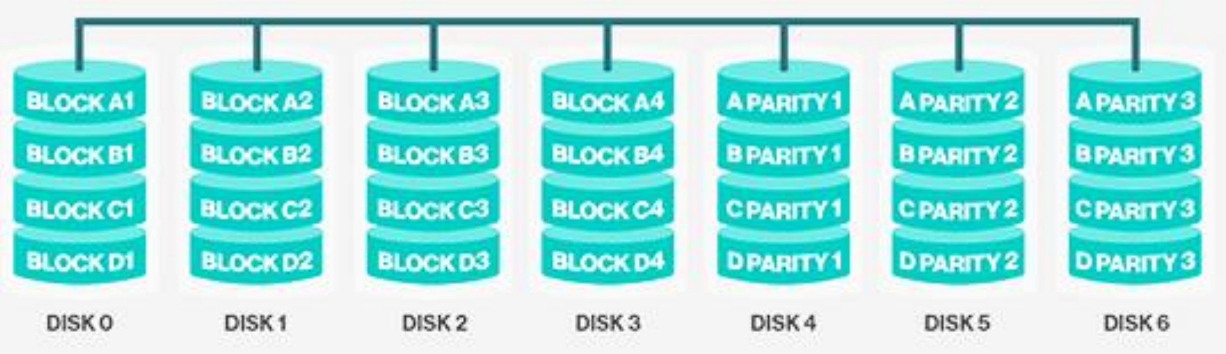
RAID 2 (Bit-level striping using a Hamming Code parity)

- It **stripes data at bit level (rather than block level) with some disks storing error checking and correcting (ECC) information**.
- It **uses ECC (Error Correcting Code)** to monitor correctness of information on disk.
- A **parity disk is then used to reconstruct corrupted or lost data**.



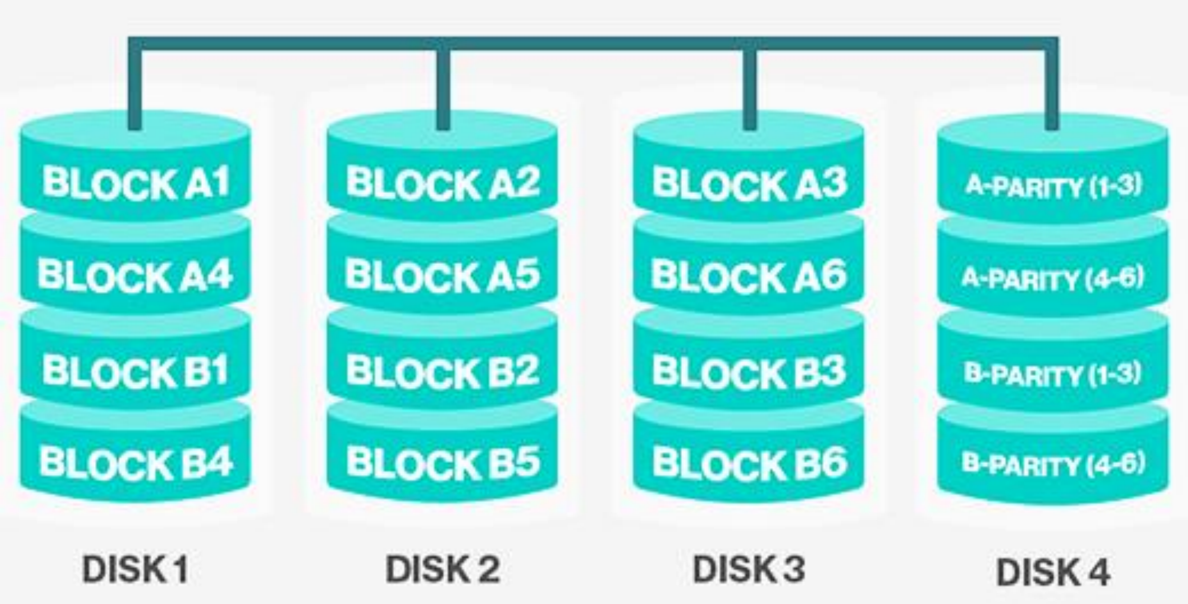
RAID 2 (Bit-level striping using a Hamming Code parity)

- Imagine **splitting each byte into a pair of 4-bit nibbles, then adding Hamming code to each one to form a 7-bit word, of which bit 1, 2 and 4 were parity bits**.
- In RAID 2 **each of seven drives needs synchronized** in terms of arm position and rotational position, and then it would be possible to write the 7-bit Hamming coded word over the seven drives.
- Here, **losing one drive did not cause problem**, which can be handled by Hamming code on the fly.
- Big problem is performance
  - must have to **read data plus ECC code** from other disks
  - for a **write, must have to modify data, ECC, and parity disks**



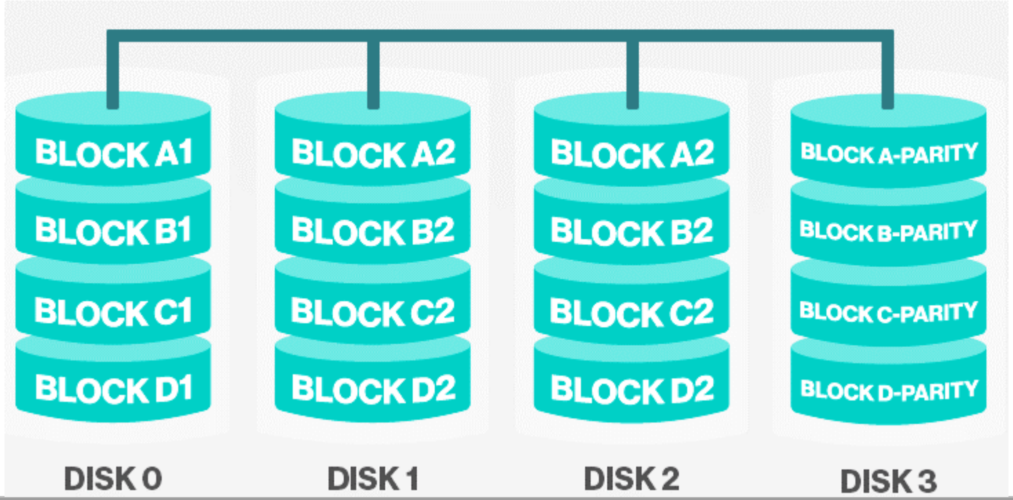
RAID 3 (Byte-level striping with a dedicated parity)

- It uses **striping and dedicates one drive for storing parity information**.
- Here **single parity bit is computed for each data word and written to a parity drive**.
- The **embedded ECC information is used to detect errors**.
- As in RAID level 2 the **drives must be exactly synchronized**.



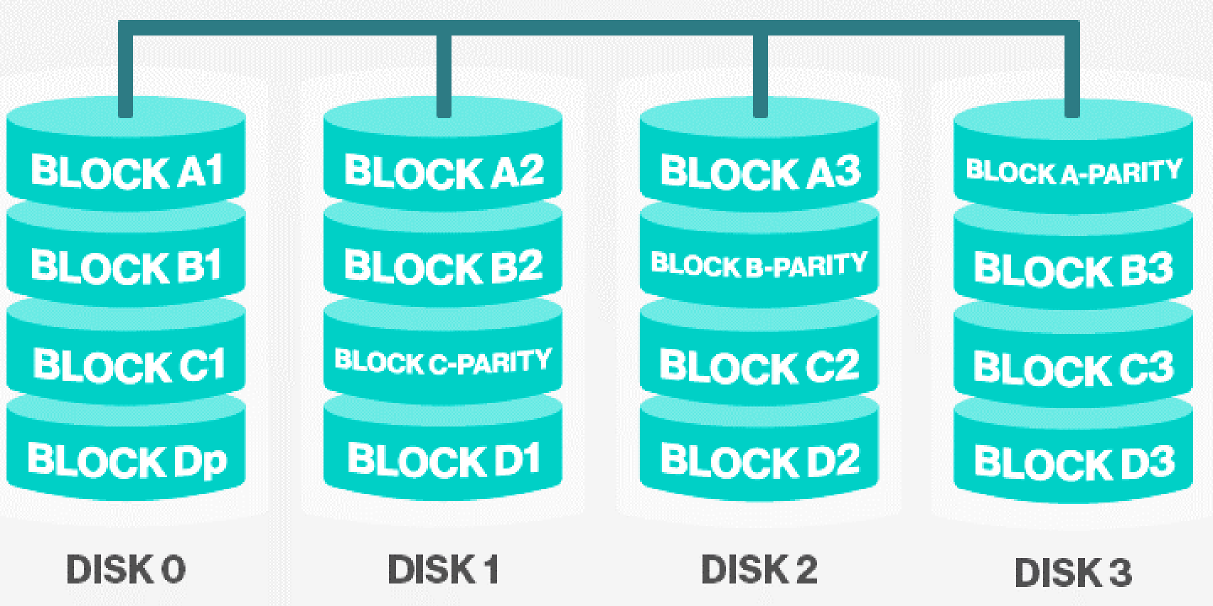
RAID 4 (Block-level striping with dedicated parity)

- It **uses large stripes (block level striping), which means you can read records from any single drive**.
- They **do not require synchronization of drives**.
- RAID 4 is like RAID 0, with strip-for-strip parity written onto an extra drive.
- If **a drive crashes, the lost bytes can be recomputed from the parity drive** by reading the entire set of drives.
- This design protects against the loss of a drive but performs poorly for small updates, if one sector is changed, it is necessary to read all the drives in order to recalculate the parity.
- It **creates heavy load** on parity drive.



RAID 5 (Block-level striping with distributed parity)

- This level is **based on block-level striping with parity**.
- The **parity information is striped across each drive**.
- As with RAID level 4, there is a heavy load in the parity drive, it may become bottleneck.
- This **bottleneck can be eliminated in RAID level 5 by distributing the parity bits uniformly over all the drives**.



THANK
YOU