CHAPTER -7

I/O Management &Disk Scheduling

I/O Devices

- One of the important jobs of an Operating System is to manage various I/O devices including mouse, keyboards, touch pad, disk drives, display adapters, USB devices, Bit-mapped screen, LED, Analog-to-digital converter, On/off switch, network connections, audio I/O, printers etc.
- An I/O system is required to take an application I/O request and send it to the physical device, then take whatever response comes back from the device and send it to the application.

I/O Devices

- I/O devices are divided into 3 types :

1. Human Readable

- Computer user can communicate with computer using human readable device. These devices are help to user for giving input to computer and reading result.
- Example : printer, display,keyboard,mouse.

2. Machine Readable

- Machine readable is used for communicating with electronic devices and components.
- Example : USB devices, disk drivers, controller , sensors.

3. Communication

- This type of I/O devices used for communicating with remote devices.
- Example : Modem, digital line drivers

Parameters affect to I/O devices

Data rate

- There may be difference of several orders of magnitude between the data transfer rate

Application

- The use to which a device is put has an influence on the software and policies in the operation system and supporting
- utilities. for ex(A disk used for file required the support of file management software

Complexity of control:

- A printer requires simple control interface. A disk is much more complex

Unit of transfer:

- Data may be transferred as a stream of byte or characters (e.g terimal I/O)

Continue..

Data representation:

- Different data encoding schemes are used by different devices,including difference in character code and parity
- conventions

Error conditions:

- The nature of errors

Organization of I/O Function

- Device management is the part of the os responsible for directly manipulating the hardware devices.
- To start I/O operation the CPU loads appropriate instructions and values into the registers of the device controller via the device driver.
- Following techniques are used for performing I/O :

Programmed I/O:

◦ The processor issues an i/o command on behalf of a process to an i/o module.

Interrupt driven I/O:

◦ The processor issues an i/o command on behalf of a process continues to execute subsequent instruction

Direct memory access:

◦ A DMA module control the exchange of data between main memory and and i/o module

The Evolution of I/O Function

- Peripheral devices are directly controlled by CPU.
- A controller or i/o module is added
- The i/o module is give direct control of memory via DMA
- The i/o module is enhanced to become a separate processor a specialized
- instruction set tailored for i/o
- • The i/o module has a local memory of own in fact a computer own right

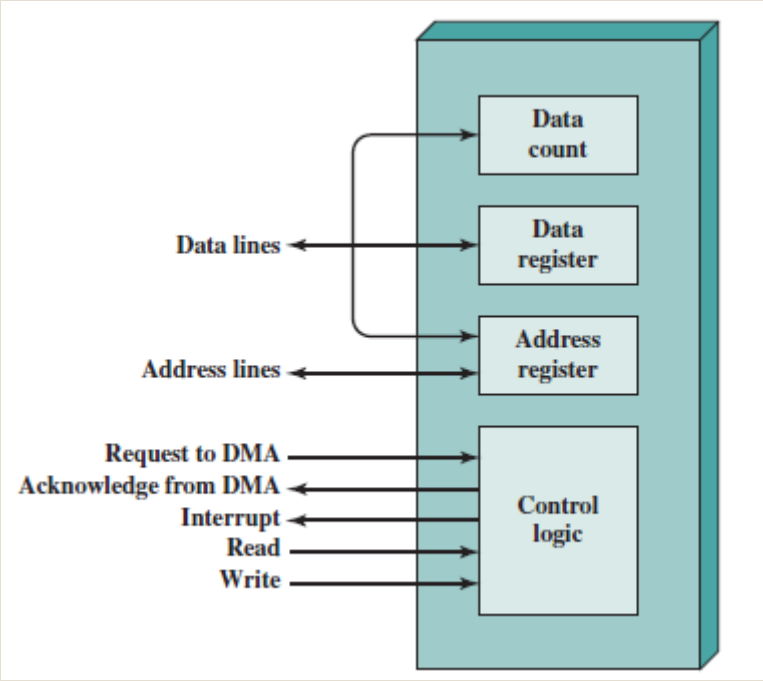
DMA

- DMA stands for "Direct Memory Access" and is a method of transferring data from the computer's RAM to another part of the computer without processing it using the CPU.
- While most data that is input or output from your computer is processed by the CPU, some data does not require processing, or can be processed by another device.
- In these situations, DMA can save processing time and is a more efficient way to move data from the computer's memory to other devices.
- In order for devices to use direct memory access, they must be assigned to a DMA channel.
- Each type of port on a computer has a set of DMA channels that can be assigned to each connected device. For example, a PCI controller and a hard drive controller each have their own set of DMA channels.

Continue…

- For example, a sound card may need to access data stored in the computer's RAM, but since it can process the data itself, it may use DMA to bypass the CPU. Video cards that support DMA can also access the system memory and process graphics without needing the CPU.
- Ultra DMA hard drives use DMA to transfer data faster than previous hard drives that required the data to first be run through the CPU.

DMA block diagram



I/O Buffering

- Input/output (I/O) buffering is a mechanism that improves the throughput of input and output operations.
- It is implemented directly in hardware and the corresponding drivers and is also ubiquitous among programming language standard libraries.
- ***Input Buffering*** is the technique of having the input device read information into the primary memory before processing the request.
- ***Output Buffering*** is the technique of saving information in memory and then writing it to the device while the process continues execution.
- Reasons for buffering
  - Processes must wait for I/O to complete before proceeding
  - Certain pages must remain in main memory during I/O

I/O Buffering

- Block-oriented
  - Information is stored in fixed sized blocks
  - Transfers are made a block at a time
  - Used for disks and USB keys
- Stream-oriented
  - Transfer information as a stream of bytes
  - Used for terminals, printers, communication ports, mouse and other pointing devices, and most other devices that are not secondary storage

Types of I/O Buffering Schemes

- The various I/O buffering techniques are as follows:

1. Single Buffering
2. Double Buffering
3. Circular Buffering

Single Buffering

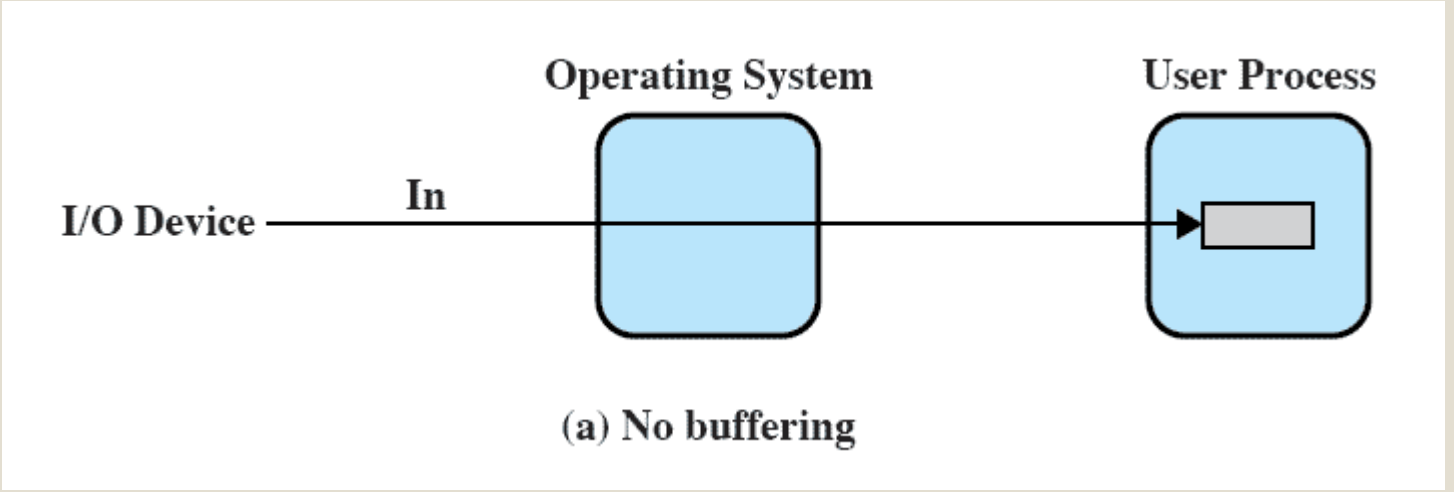
- Operating system assigns a buffer in main memory for an I/O request
- Block-oriented
  - Input transfers made to buffer
  - Block moved to user space when needed
  - Another block is moved into the buffer
  - User process can process one block of data while next block is read in
  - Swapping can occur since input is taking place in system memory, not user memory
  - Operating system keeps track of assignment of system buffers to user processes

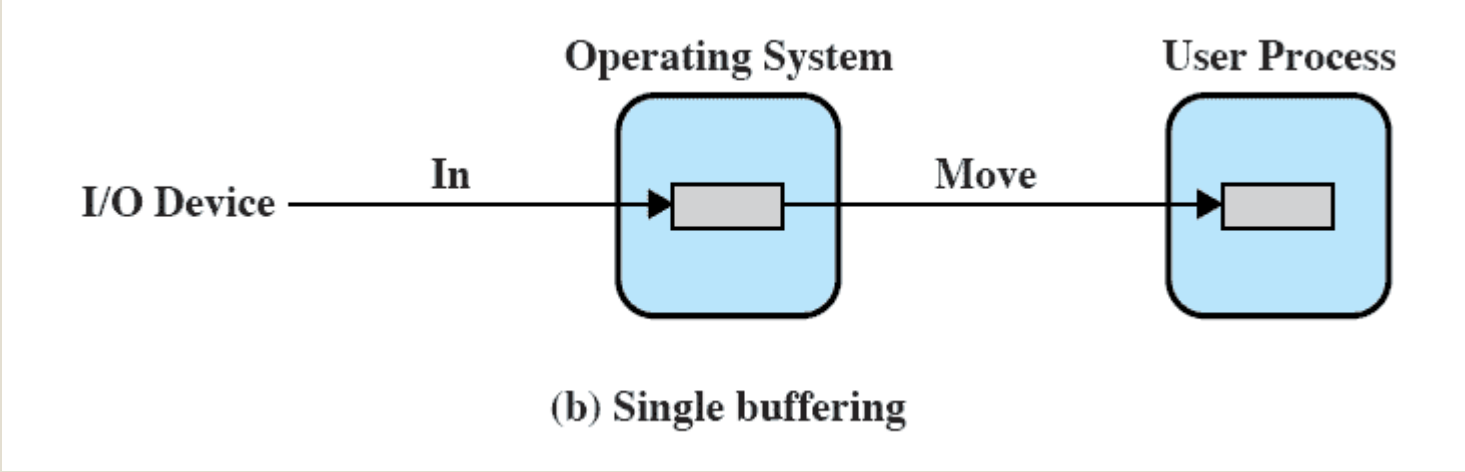
Single Buffering

Stream-oriented

- Used a line at time
- User input from a terminal is one line at a time with carriage return signaling the end of the line
- Output to the terminal is one line at a time

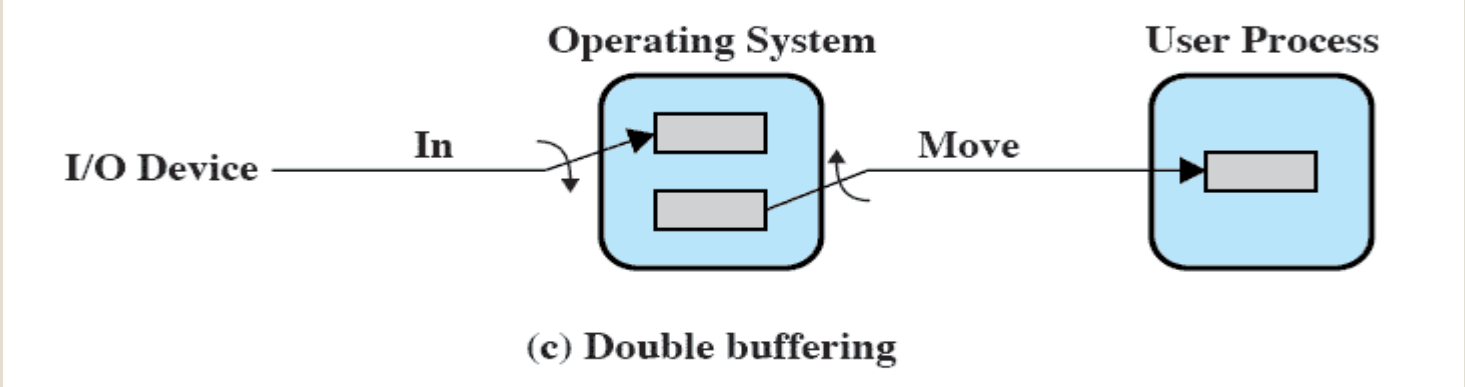
Single Buffering





Double Buffering

- Use two system buffers instead of one
- A process can transfer data to or from one buffer while the operating system empties or fills the other buffer

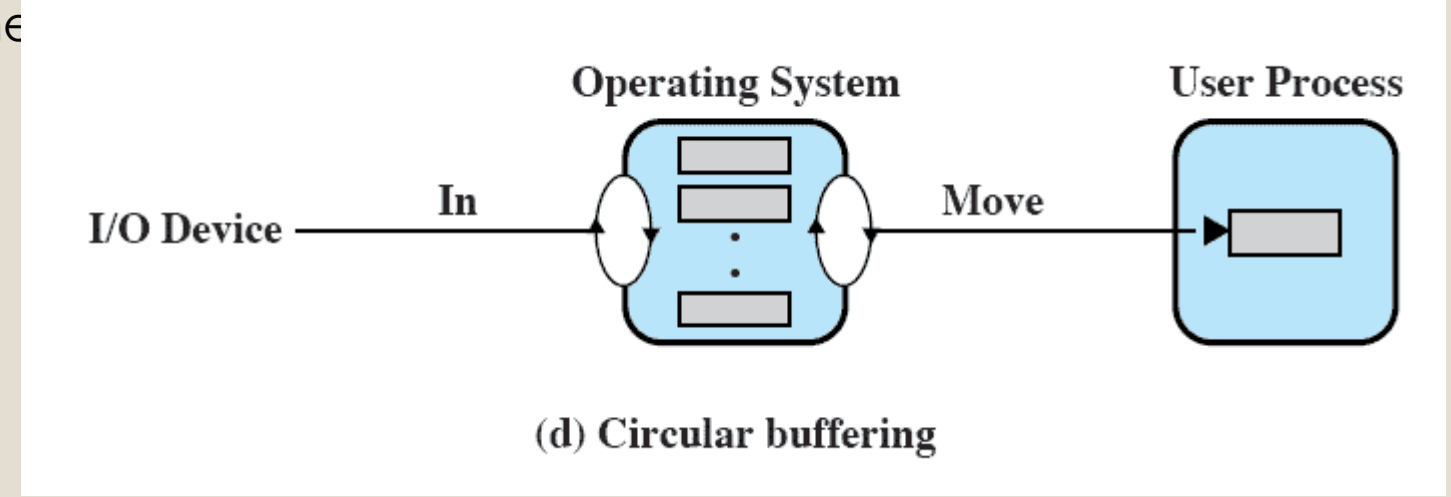


Double Buffering

- one buffer is for driver or controller to store data while waiting or it to be retrieved by higher level of hierarchy.
- Other buffer is to store data from the lower level module.
- Double buffering is also called as buffer swapping.

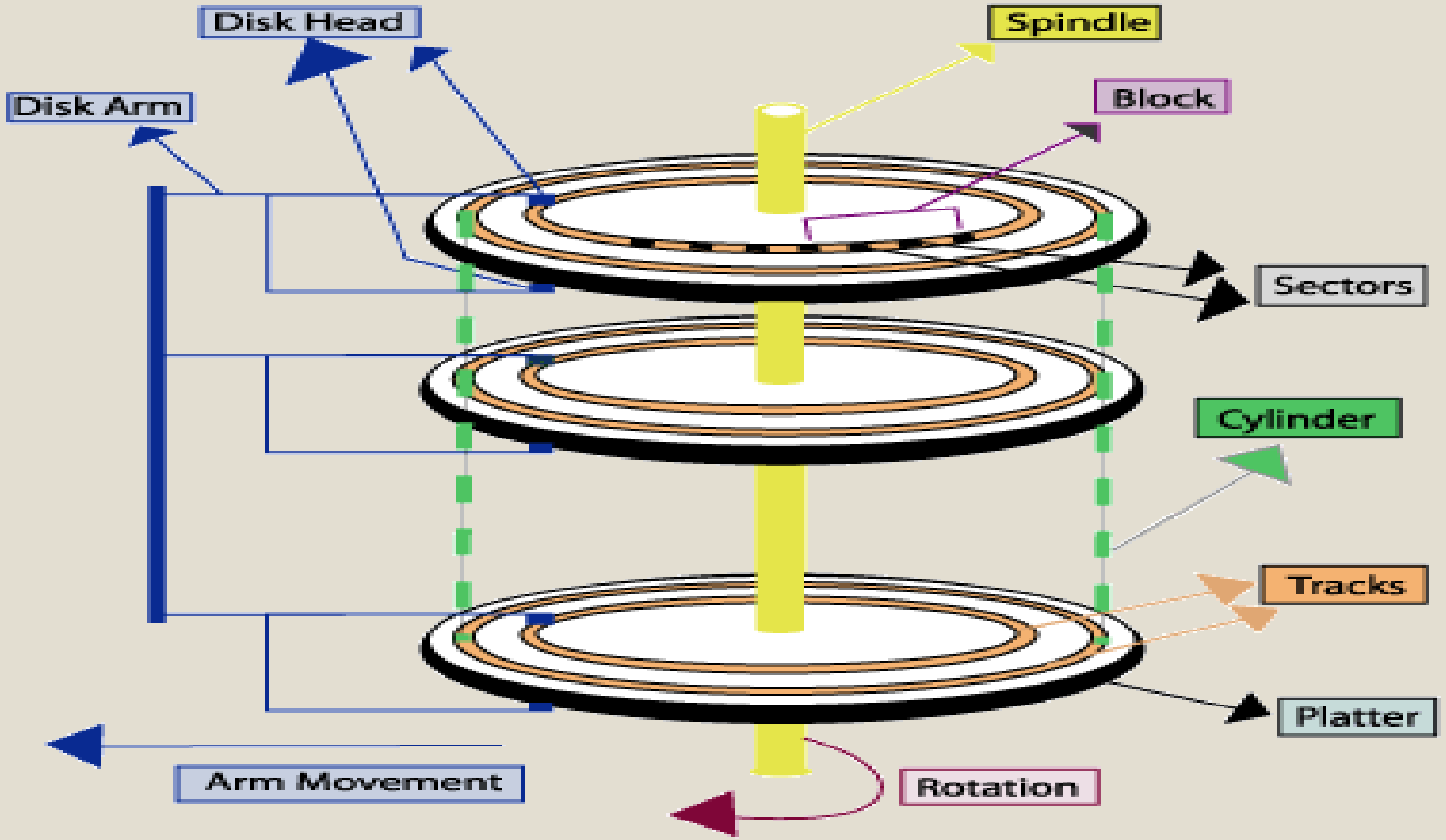
Circular Buffering

- More than two buffers are used
- Each individual buffer is one unit in a circular buffer
- Used when I/O operation must keep up with process
- In this, producer can not pass the consumer because it would overwrite buffers before they had consume



(d) Circular buffering

Disk Structure



Disk Performance Parameters

- To read or write, the disk head must be positioned at the desired track and at the beginning of the desired sector
- Following parameters are used for disk performance :
  - 1. Seek time
    - Time it takes to position the head at the desired track.
  - 2. Rotational delay or rotational latency
    - Time its takes for the beginning of the sector to reach the head.

  3. Access Time
    - The sum of seek time and rotational delay equals to the access time

Disk Performance Parameters

- To read or write, the disk head must be positioned at the desired track and at the beginning of the desired sector
- Following parameters are used for disk performance :
  - **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.

Continue..

- **Disk Access Time:** Disk Access Time is the sum of seek time and rotational delay.
- **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. *Variance Response Time* is measure of how individual request are serviced with respect to average response time. So the disk scheduling algorithm that gives minimum variance response time is better.

Disk Scheduling

- **Disk scheduling** is done by operating systems to schedule I/O requests arriving for the disk. Disk scheduling is also known as I/O scheduling.

Disk scheduling is important because:

- Multiple I/O requests may arrive by different processes and only one I/O request can be served at a time by the disk controller. Thus other I/O requests need to wait in the waiting queue and need to be scheduled.
- Two or more request may be far from each other so can result in greater disk arm movement.
- Hard drives are one of the slowest parts of the computer system and thus need to be accessed in an efficient manner.

Disk Scheduling Algorithms

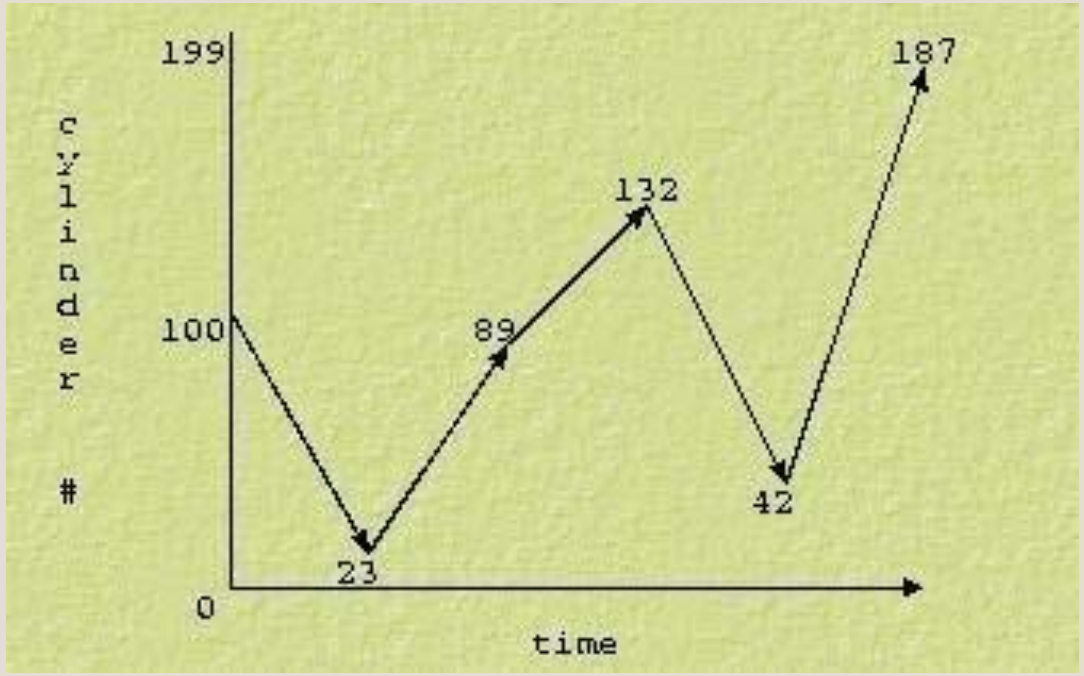
- There are total 6 different algorithms for disk scheduling :
- **1. FIFO (First In First out)**
- **2. SSTF (Shortest Seek Time First)**
- **3. SCAN**
- **4. C-SCAN**
- **5. LOOK**
- **6. C-LOOK**

Disk Scheduling Algorithms

- **1. FCFS:** FCFS is the simplest of all the Disk Scheduling Algorithms. In FCFS, the requests are addressed in the order they arrive in the disk queue.
- **Advantages**:
- Every request gets a fair chance
- No indefinite postponement
- **Disadvantages**:
- Does not try to optimize seek time
- May not provide the best possible service

Example

- disk queue with requests for I/O to blocks on cylinders **23, 89, 132, 42, 187**
- disk head initially at **100**



Continue..

- **Number of track traversed are** : (‘|’ this is mode sign)

|100-23| + |23-89| + |89-132| + |23-89| + |132-42| + |42-187|

=77+66+43+90+145

=421

Average Seek length = Total no. of track traversed / total requested tracks

= 421/ 5

= 84.2

- You can calculate average seek length for all algorithm using above formula.

Disk Scheduling Algorithm

- **SSTF:** In SSTF (Shortest Seek Time First), requests having shortest seek time are executed first. So, the seek time of every request is calculated in advance in the queue and then they are scheduled according to their calculated seek time. As a result, the request near the disk arm will get executed first. SSTF is certainly an improvement over FCFS as it decreases the average response time and increases the throughput of system.

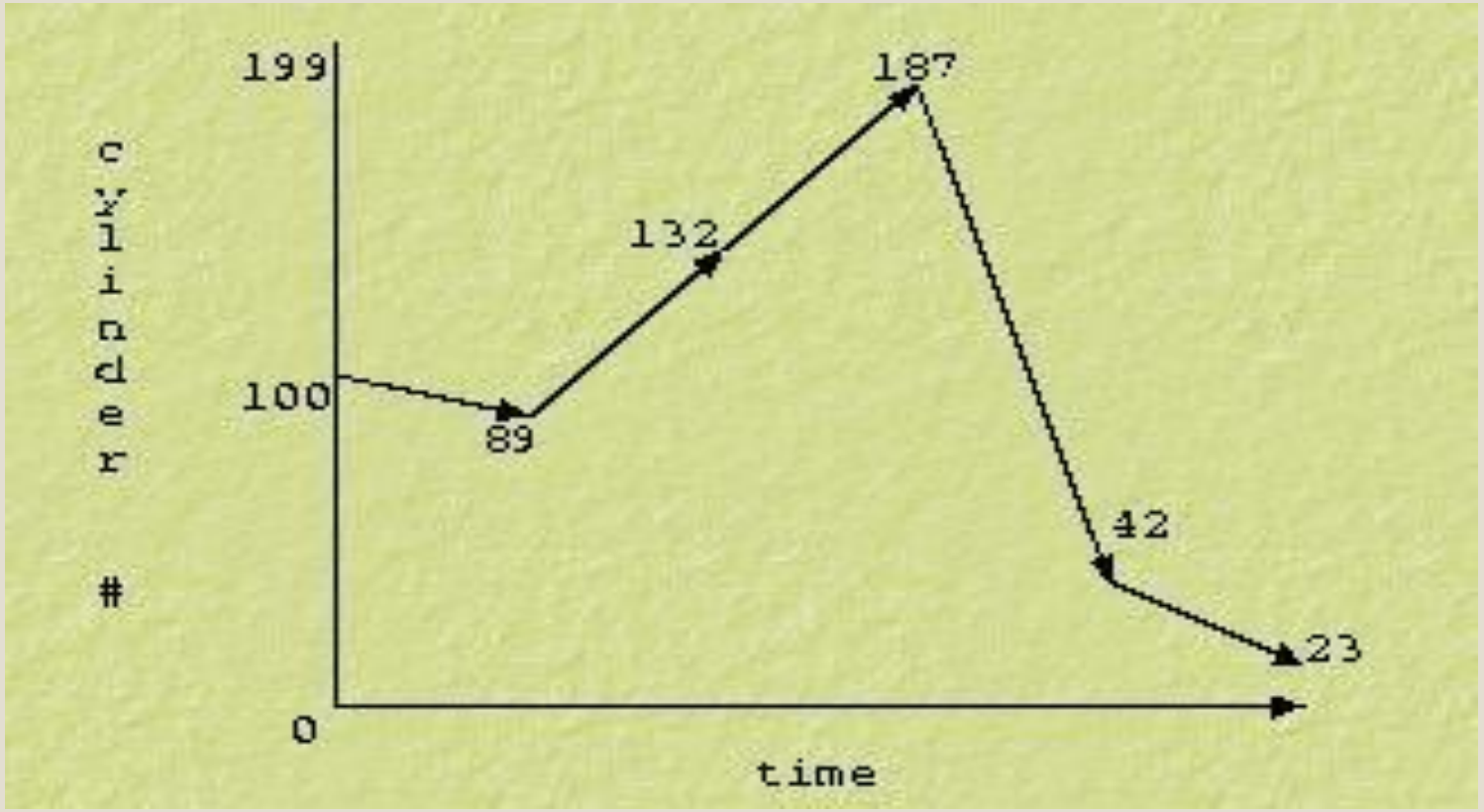
Advantages:

- Average Response Time decreases
- Throughput increases

Disadvantages:

- Overhead to calculate seek time in advance
- Can cause Starvation for a request if it has higher seek time as compared to incoming requests
- High variance of response time as SSTF favours only some requests

disk queue with requests for I/O to blocks on cylinders 23, 89, 132, 42, 187
disk head initially at 100



- Total tracks = 11+43+55+145+19=273

Disk Scheduling Algorithm

- **SCAN:** In SCAN algorithm the disk arm moves into a particular direction and services the requests coming in its path and after reaching the end of disk, it reverses its direction and again services the request arriving in its path. So, this algorithm works as an elevator and hence also known as **elevator algorithm.** As a result, the requests at the midrange are serviced more and those arriving behind the disk arm will have to wait.

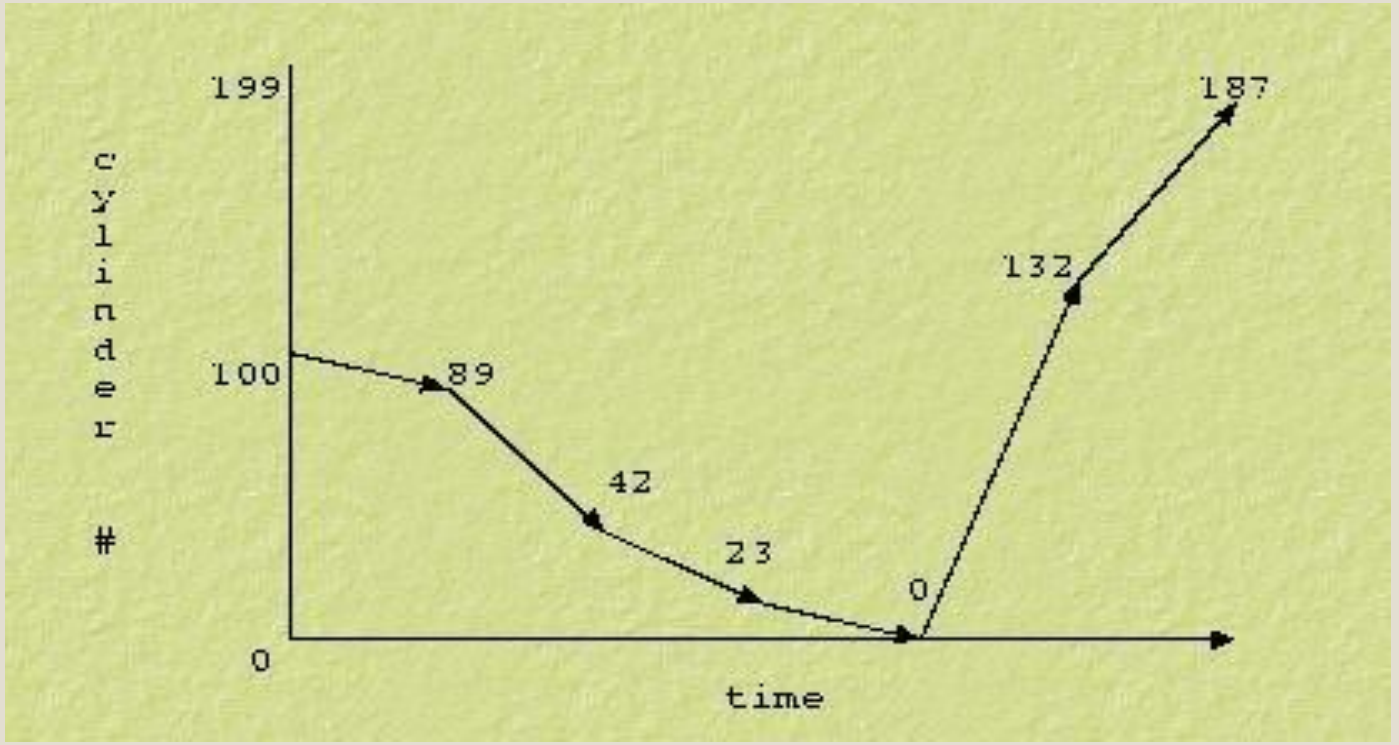
Advantages:

- High throughput
- Low variance of response time
- Average response time

Disadvantages:

- Long waiting time for requests for locations just visited by disk arm

disk queue with requests for I/O to blocks on cylinders 23, 89, 132, 42, 187
disk head initially at 100



- Total Tracks = 11+47+19+23+132+55=287

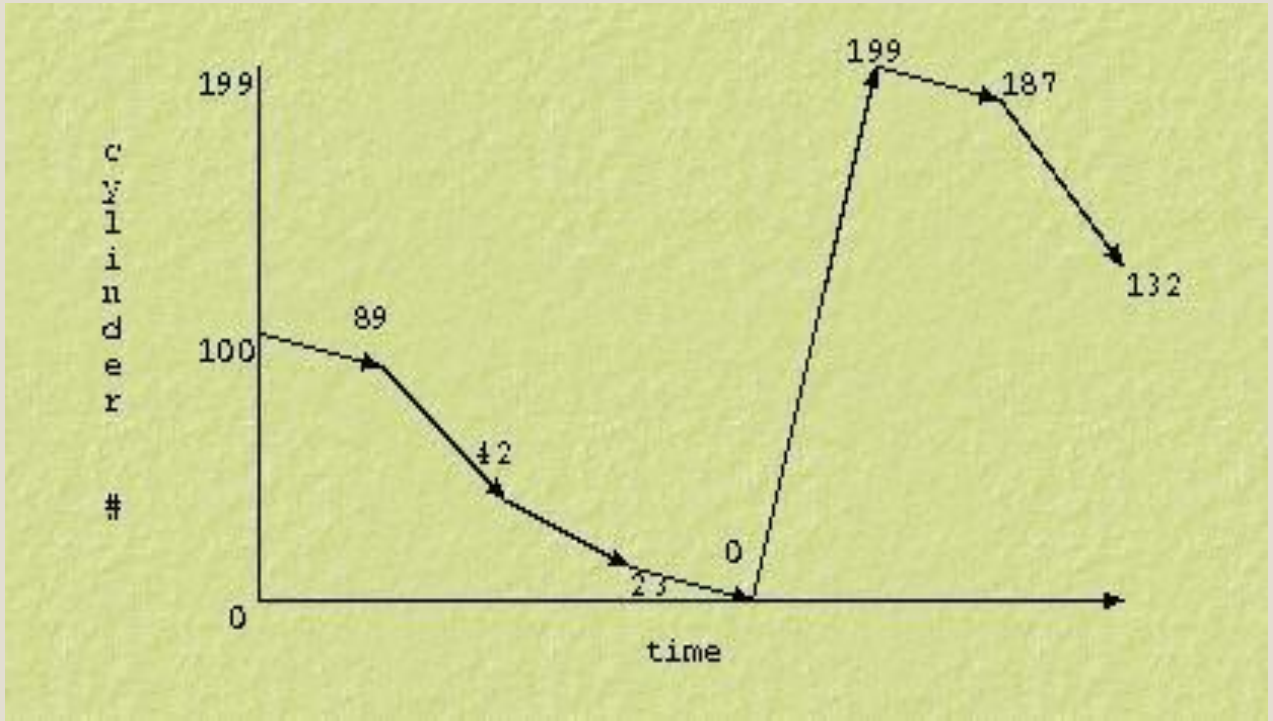
Disk Scheduling Algorithm

- **C-SCAN**: In SCAN algorithm, the disk arm again scans the path that has been scanned, after reversing its direction. So, it may be possible that too many requests are waiting at the other end or there may be zero or few requests pending at the scanned area.
- These situations are avoided in *CSCAN* algorithm in which the disk arm instead of reversing its direction goes to the other end of the disk and starts servicing the requests from there. So, the disk arm moves in a circular fashion and this algorithm is also similar to SCAN algorithm and hence it is known as C-SCAN (Circular SCAN).

Advantages:

- Provides more uniform wait time compared to SCAN

disk queue with requests for I/O to blocks on cylinders 23, 89, 132, 42, 187
disk head initially at 100



Total Tracks = 11+47+19+23+199+12+55=366

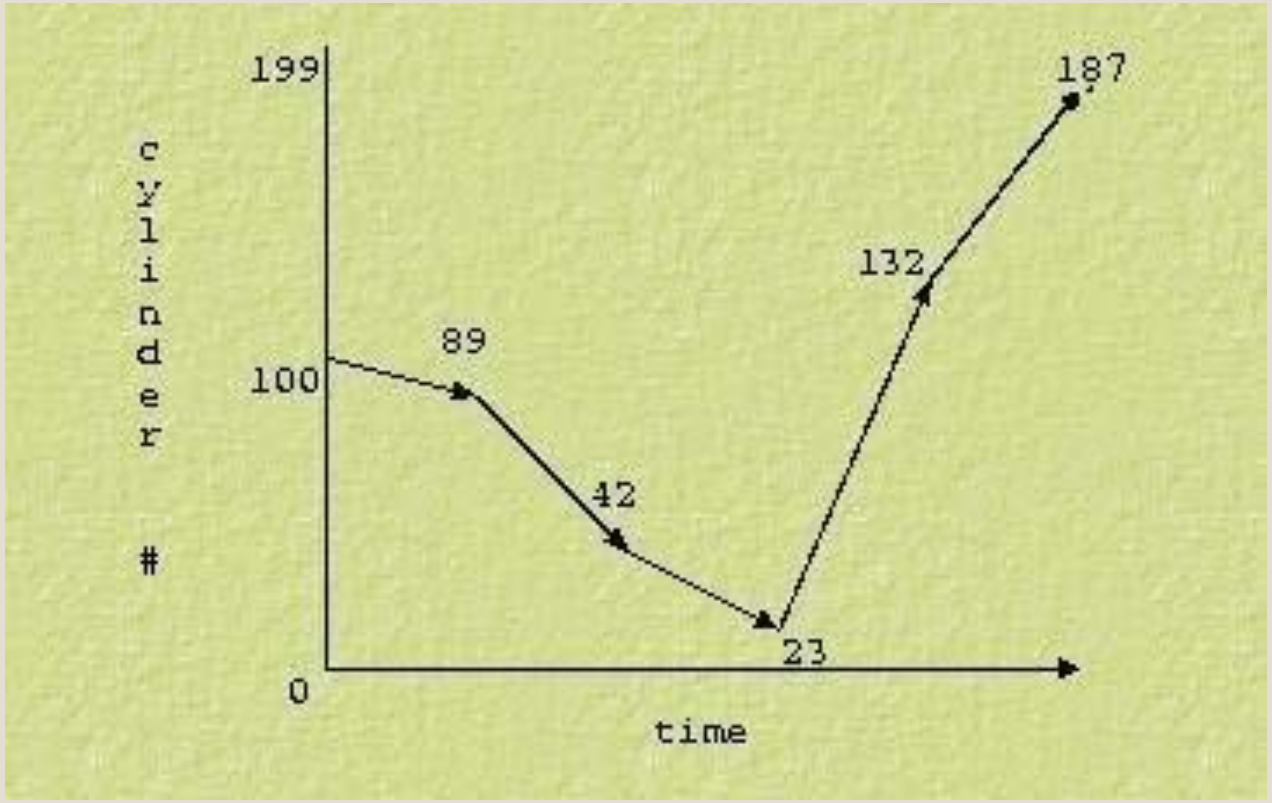
Disk Scheduling Algorithm

- **LOOK:** It is similar to the SCAN disk scheduling algorithm except for the difference that the disk arm in spite of going to the end of the disk goes only to the last request to be serviced in front of the head and then reverses its direction from there only. Thus it prevents the extra delay which occurred due to unnecessary traversal to the end of the disk.

- **CLOOK:** As LOOK is similar to SCAN algorithm, in similar way, CLOOK is similar to CSCAN disk scheduling algorithm. In CLOOK, the disk arm in spite of going to the end goes only to the last request to be serviced in front of the head and then from there goes to the other end's last request. Thus, it also prevents the extra delay which occurred due to unnecessary traversal to the end of the disk.

LOOK algorithm example :

disk queue with requests for I/O to blocks on cylinders 23, 89, 132, 42, 187
disk head initially at 100



Total Tracks = 11+47+19+109+55=241

C-LOOK Example

- C-LOOK Example : consider requested track are 23,89,132,42,187. starting track at 100.

Next Track Accessed	Number of track traversed
89	11
42	47
23	19
187	164
132	55

Average Seek Length = 11+47+19+164+55/5 = 296/5 = 59.2

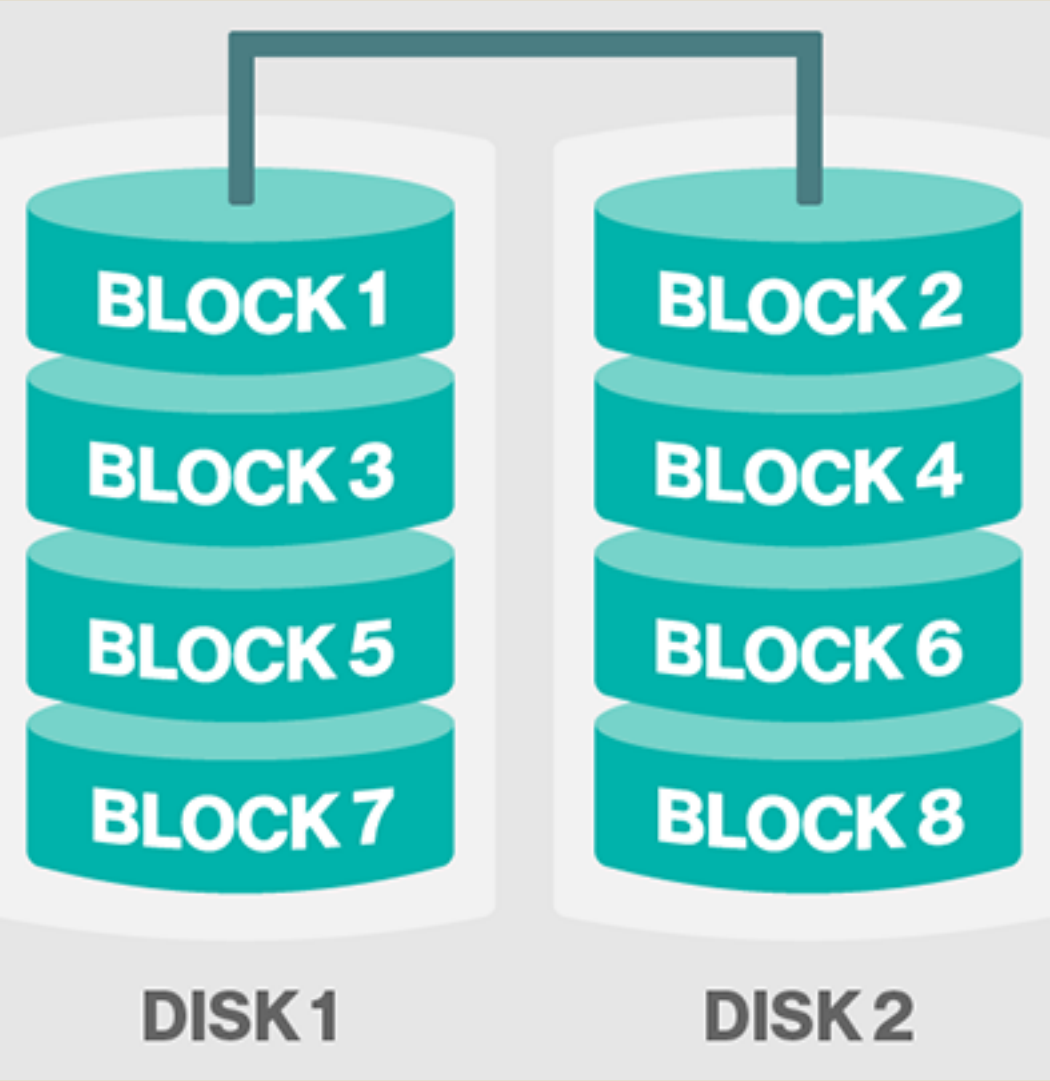
- You can refere below link for algorithm example
- http://www2.cs.uregina.ca/~hamilton/courses/330/notes/io/node8.html

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

- 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 0

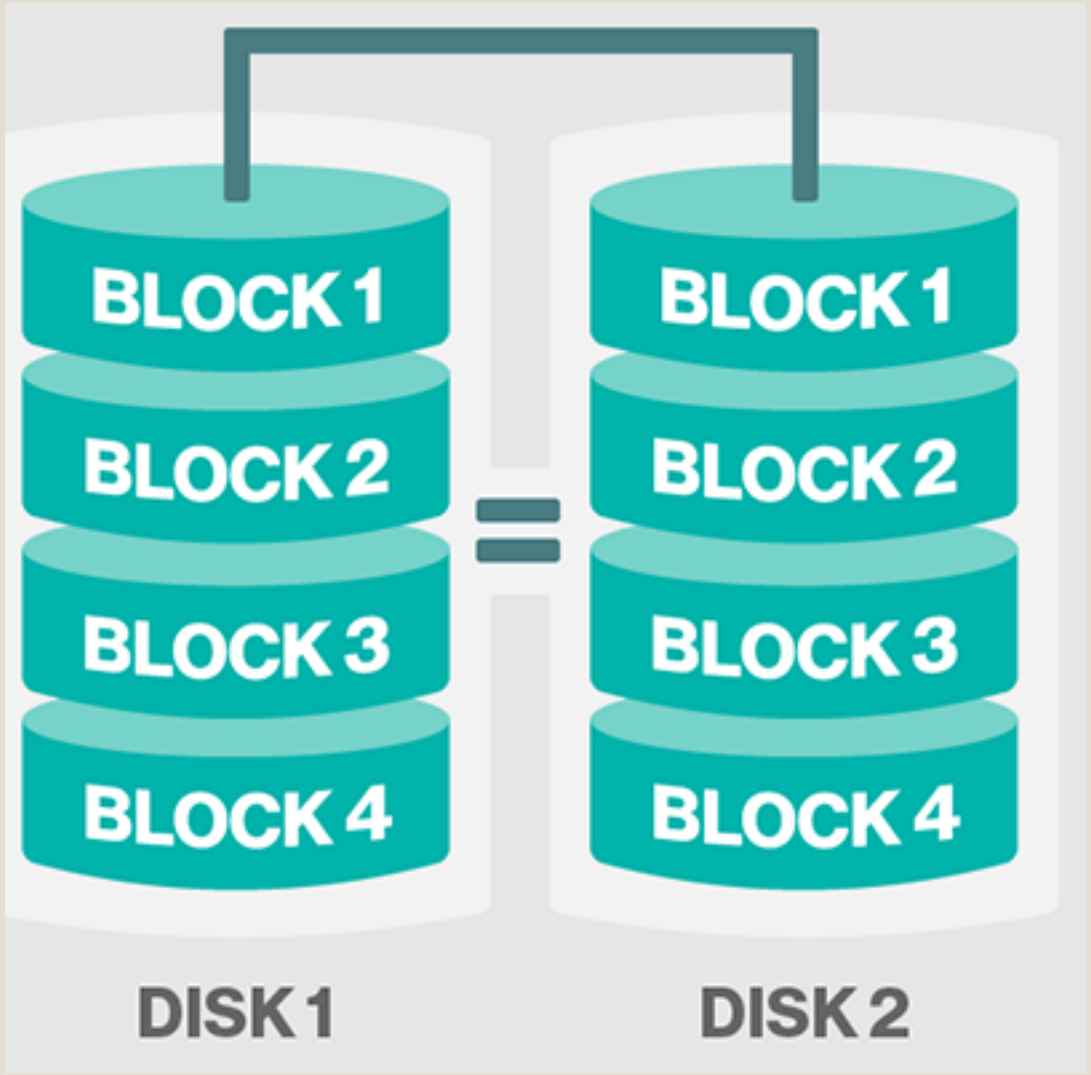
It splits data (file) into blocks of data.

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.

RAID 1

- 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.
- Software RAID 1 solutions do not always allow a hot swap of a failed drive.
- That means the failed drive can only be replaced after powering down the computer it is attached to.
- For servers that are used simultaneously by many people, this may not be acceptable. Such systems typically use hardware controllers that do support hot swapping.

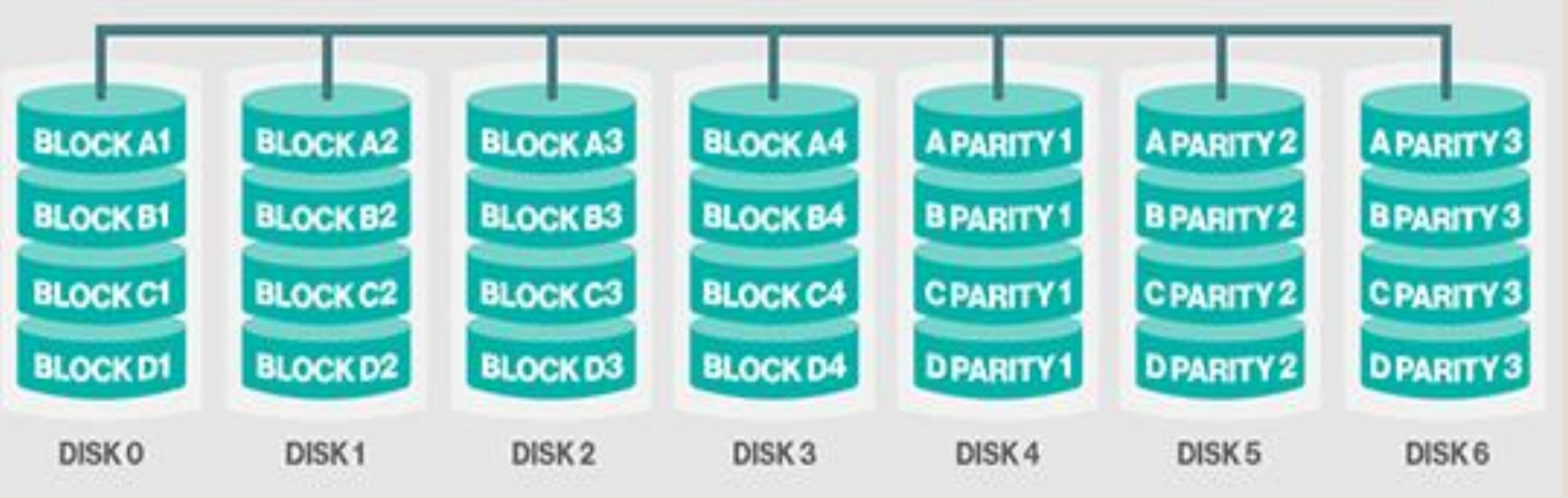


RAID1

- 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

RAID 2

- It stripes data at bit level (rather than block level) with dedicated Hamming-code parity.
- 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.

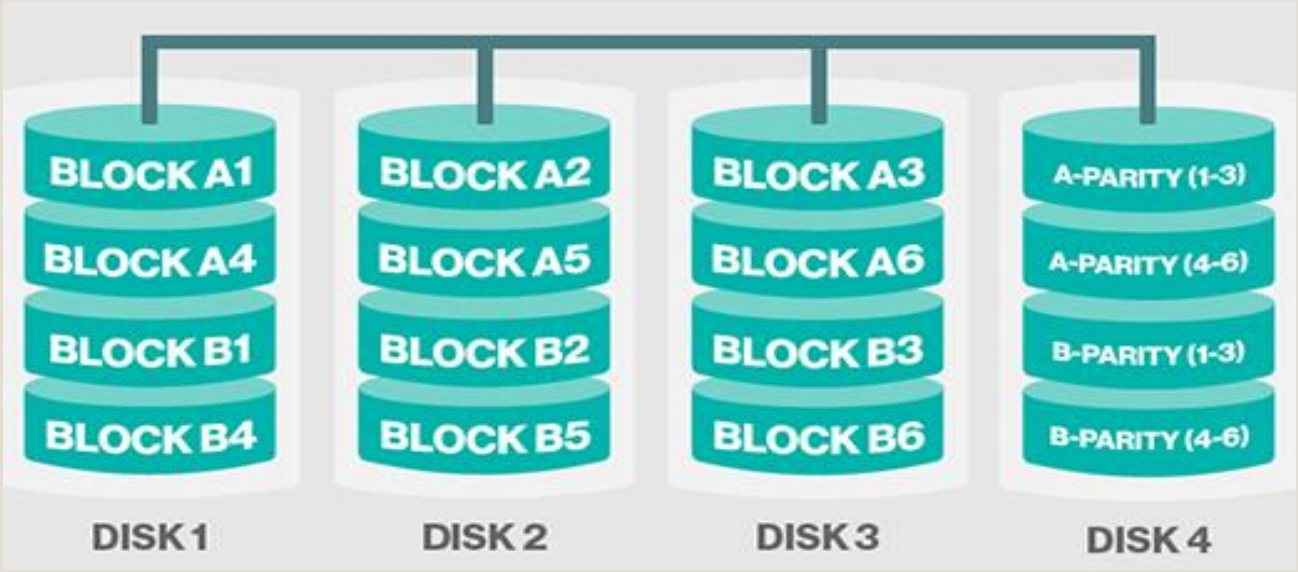


RAID 2

- 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 this RAID level 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

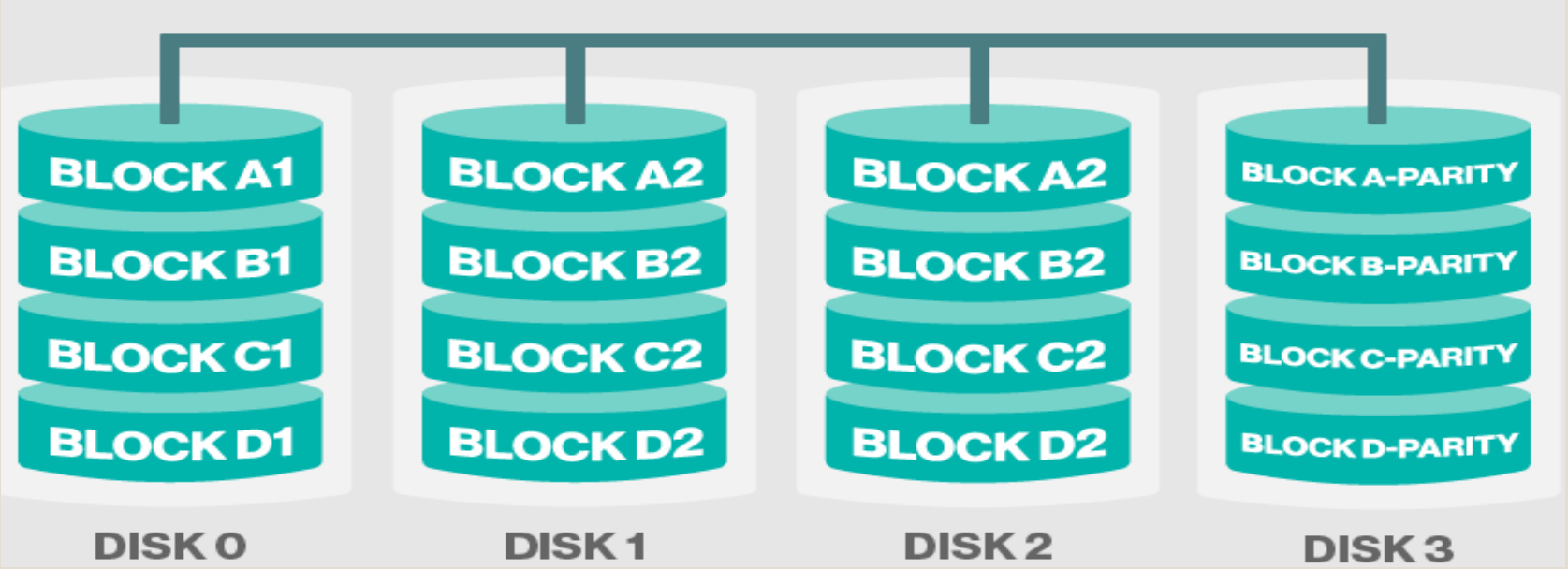
RAID 3

- This technique 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

- This level uses large stripes (block level striping), which means you can read records from any single drive.
- They do not require synchronization of drives.

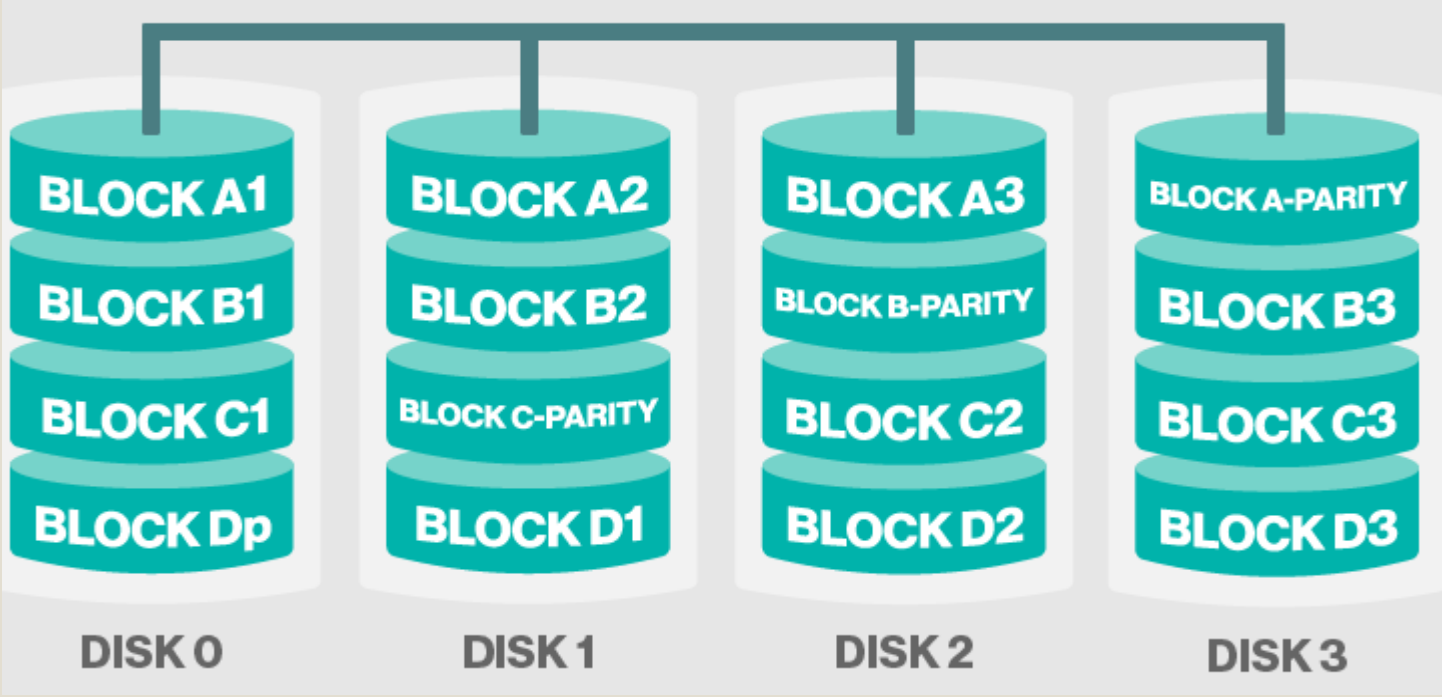


RAID 4

- RAID level 4 is like RAID level 0, with strip-for-strip parity written onto an extra drive, for example, if each strip is k bytes long, all strips are EXCLUSIVE ORed together, resulting in a parity strip k bytes long.
- 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

- This level is based on block-level striping with parity.
- The parity information is striped across each drive.



RAID 5

- 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.