# Device Management

# Contents

- Classification of IO devices, Controllers, Memory Mapped IO, DMA Operation, Interrupts,
- Goals of IO Software, Handling IO (Programmed IO, Interrupt Driven IO, IO using DMA), IO Software Layers (Interrupt Handlers, Device Drivers)
- Disk Formatting (Cylinder Skew, Interleaving, Error handling),
- RAID,
- Disk Structure, Disk Scheduling (FCFS, SSTF, SCAN, CSCAN, LOOK, CLOOK)

# Principles of I/O hardware

- Different people look at I/O hardware in different ways.
- Electrical engineers look at it in terms of chips, wires, power supplies, motors, and all the other physical components that comprise the hardware.
- Programmers look at the interface presented to the software-the commands the hardware accepts, the functions it carries out, and the errors that can be reported back.

- Conceptually, a simple personal computer can be abstracted to a model resembling that of Fig. 6-1.
- The CPU, memory, and I/O devices are all connected by a system bus and communicate with one another over it.
- Modern personal computers have a more complicated structure, involving multiple buses, which we will look at later. For the time being, this model will be sufficient.
- In the following sections, we will briefly review these components and examine some of the hardware issues that are of concern to operating system designers. Needless to say, this will be a very compact summary. Many books have been written on the subject of computer hardware and computer organization. Two well-known ones are by Tanenbaum and Austin (2012) and Patterson and Hennessy (2013).

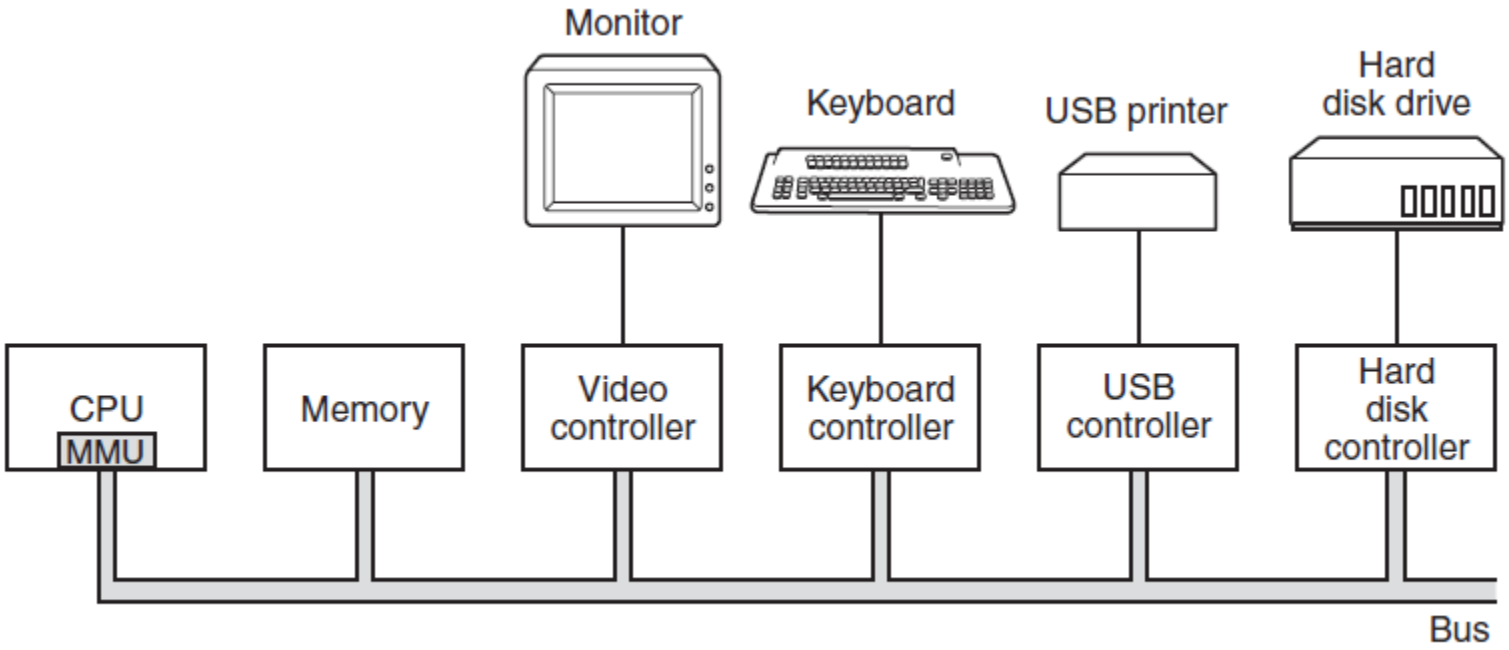



**Figure 6-1. Some of the components( memory, controllers and i/o devices) of a simple personal computer.**

# I/O Devices

- I/O devices can be roughly divided into two categories:
  - **Block devices** and
  - **Character devices**.
- A **block device** is one that stores information in fixed-size blocks, each one with its own address. Common block sizes range from 512 to 65,536 bytes.
- It is possible to read/write each and every block independently in case of block device.
- Hard disks, Blu-ray discs, and USB sticks are common block devices.
- A **character device** delivers or accepts a stream of characters, without regard to any block structure.
- It is not addressable and does not have any seek operation.
- Printers, network interfaces, mice (for pointing), rats (for psychology lab experiments), and most other devices that are not disk-like can be seen as character devices

| Device | Data rate |
|---|---|
| Keyboard | 10 bytes/sec |
| Mouse | 100 bytes/sec |
| 56K modem | 7 KB/sec |
| Scanner at 300 dpi | 1 MB/sec |
| Digital camcorder | 3.5 MB/sec |
| 4x Blu-ray disc | 18 MB/sec |
| 802.11n Wireless | 37.5 MB/sec |
| USB 2.0 | 60 MB/sec |
| FireWire 800 | 100 MB/sec |
| Gigabit Ethernet | 125 MB/sec |
| SATA 3 disk drive | 600 MB/sec |
| USB 3.0 | 625 MB/sec |
| SCSI Ultra 5 bus | 640 MB/sec |
| Single-lane PCIe 3.0 bus | 985 MB/sec |
| Thunderbolt 2 bus | 2.5 GB/sec |
| SONET OC-768 network | 5 GB/sec |

**Figure 6-2. Some typical device, network, and bus data rates.**

# Device Controllers

- A **device controller** is a system that handles the incoming and outgoing signals of the CPU.
- A **device** is connected to the computer via a plug and socket, and the socket is connected to a **device controller**. **Device controllers** use binary and digital codes.
- Many controllers can handle two, four, or even eight identical devices. If the interface between the controller and device is a standard interface, either an official ANSI, IEEE, or ISO standard or a de facto one, then companies can make controllers or devices that fit that interface.
- Many companies, for example, make disk drives that match the SATA, SCSI, USB, Thunderbolt, or FireWire (IEEE 1394) interfaces.

# Memory-Mapped I/O

- **Memory-mapped I/O** uses the same address space to address both **memory** and I/**O** devices.
- The **memory** and registers of the I/**O** devices are **mapped** to (associated with) address values. So when an address is accessed by the CPU, it may refer to a portion of physical **RAM**, or it can instead refer to **memory** of the I/**O** device.
- Each controller has a few registers that are used for communicating with the CPU. By writing into these registers, the operating system can command the device to deliver data, accept data, switch itself on or off, or otherwise perform some action.
- By reading from these registers, the operating system can learn what the device's state is, whether it is prepared to accept a new command, and so on.

- In addition to the control registers, many devices have a data buffer that the operating system can read and write. For example, a common way for computers to display pixels on the screen is to have a video RAM, which is basically just a data buffer, available for programs or the operating system to write into.
- The issue thus arises of how the CPU communicates with the control registers and also with the device data buffers. Two alternatives exist. In the first approach,

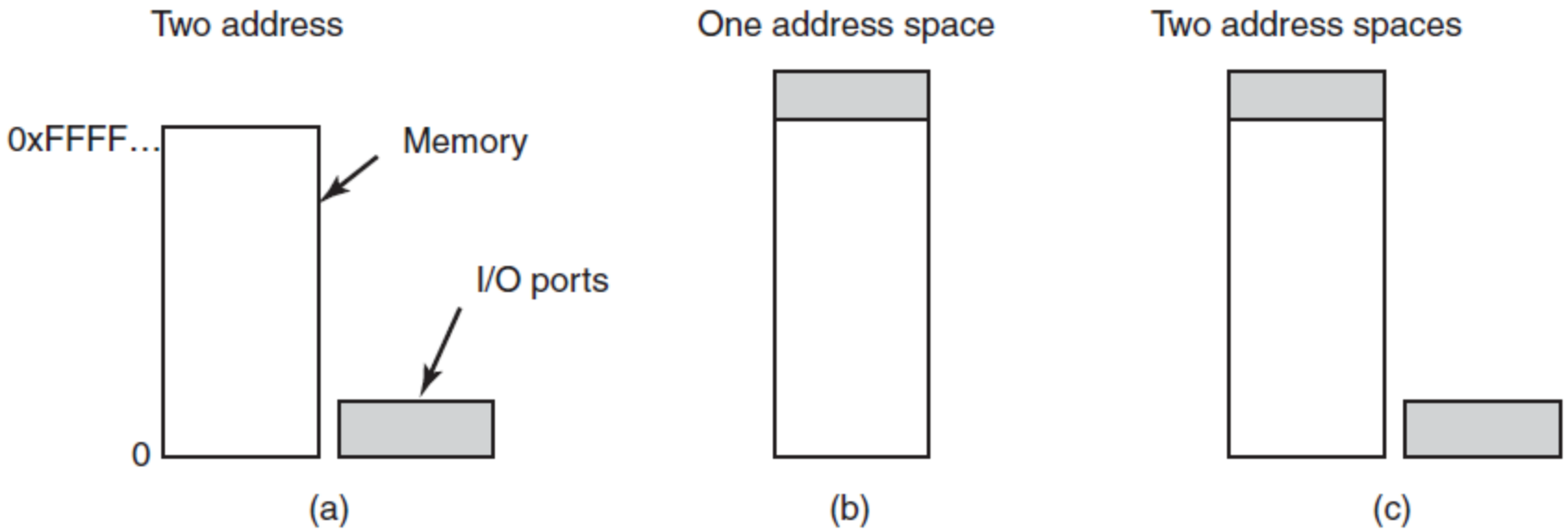



**Figure 6-3. (a) Separate I/O and memory space. (b) Memory-mapped I/O. (c) Hybrid.**

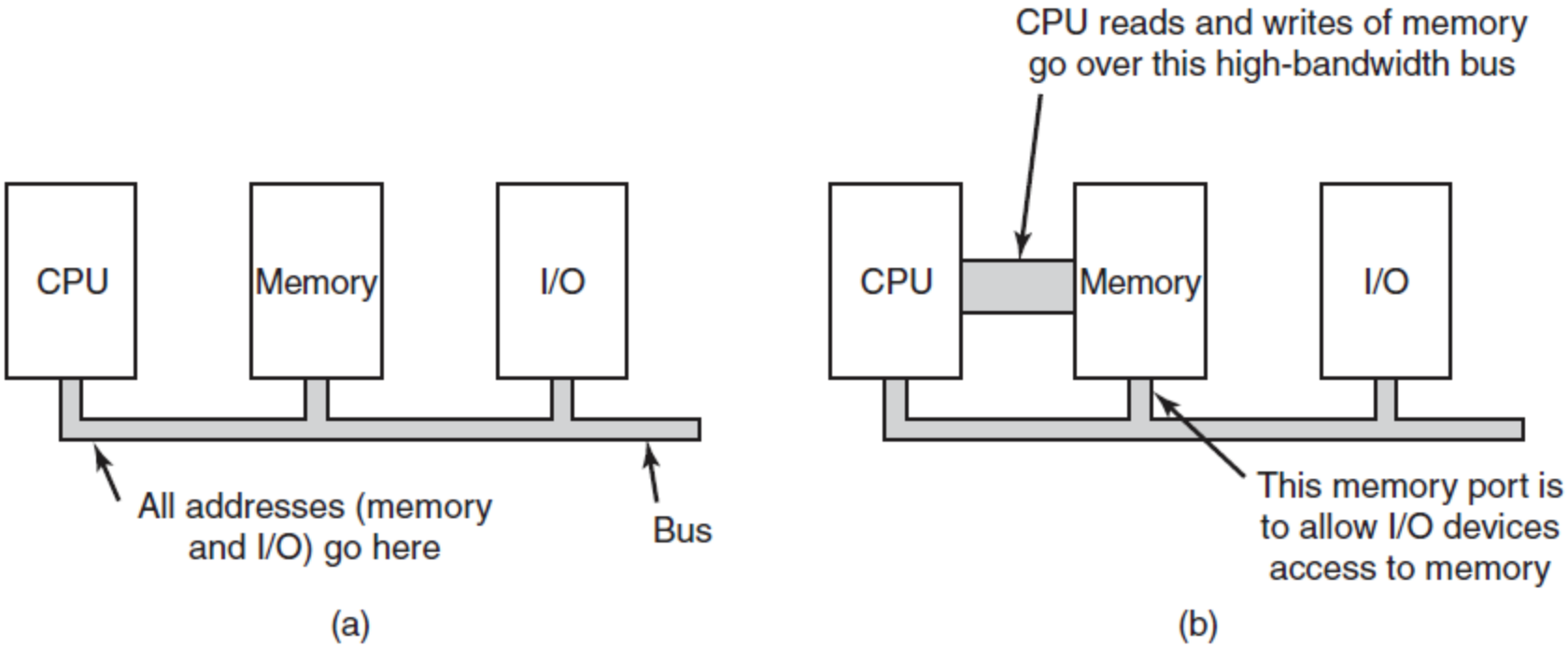



**Figure 6-4. (a) A single-bus architecture. (b) A dual-bus memory architecture.**

# Direct Memory Access

- The CPU can request data from an I/O controller one byte at a time, but doing so wastes the CPU's time, so a different scheme, called **DMA** (**Direct Memory Access**) is often used.

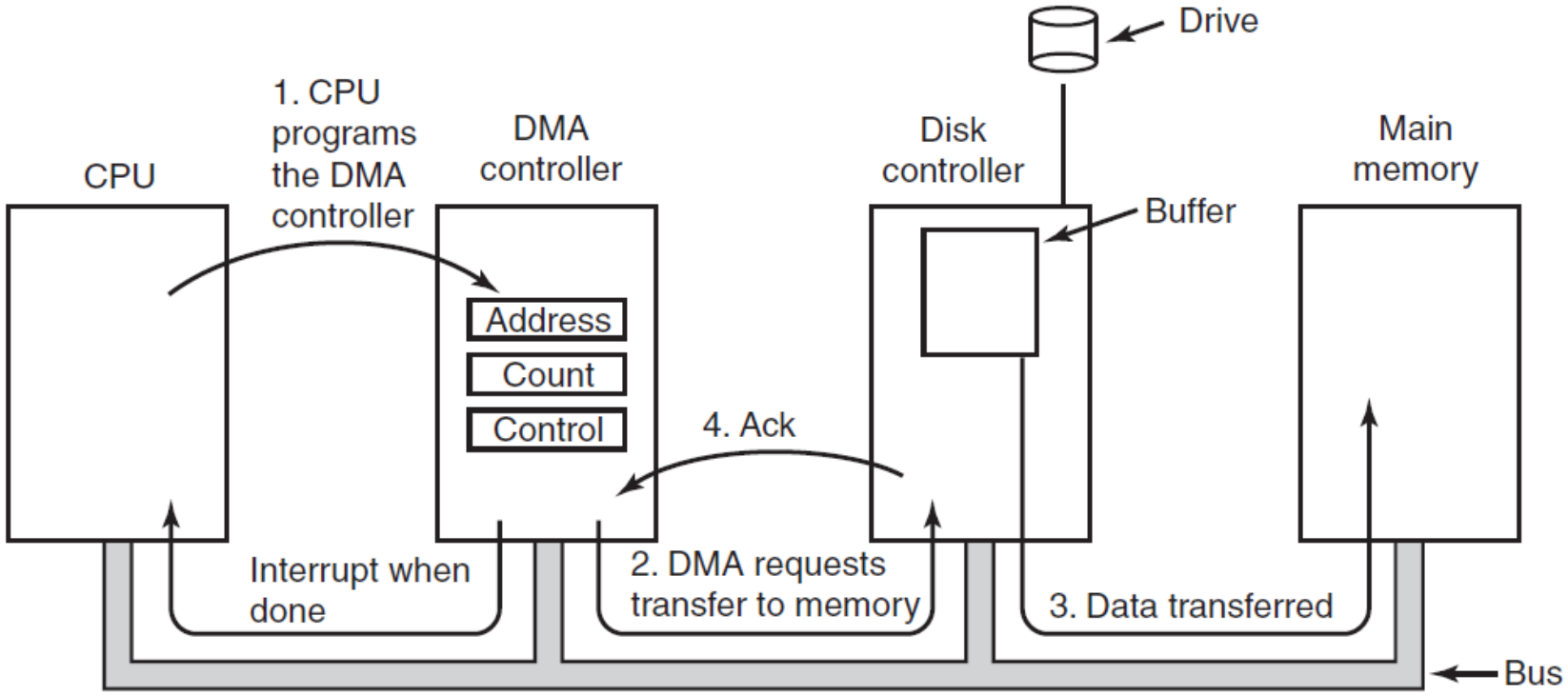



**Figure 6-5. Operation of a DMA transfer.**

- The operating system can use only DMA if the hardware has a DMA controller, which most systems do.
- Sometimes this controller is integrated into disk controllers and other controllers, but such a design requires a separate DMA controller for each device.
- DMA contains several registers that can be written and read by the CPU. These include a memory address register, a byte count register, and one or more control registers.
- The control registers specify the I/O port to use, the direction of the transfer (reading from the I/O device or writing to the I/O device), the transfer unit (byte at a time or word at a time), and the number of bytes to transfer in one burst.

# Interrupts Revisited

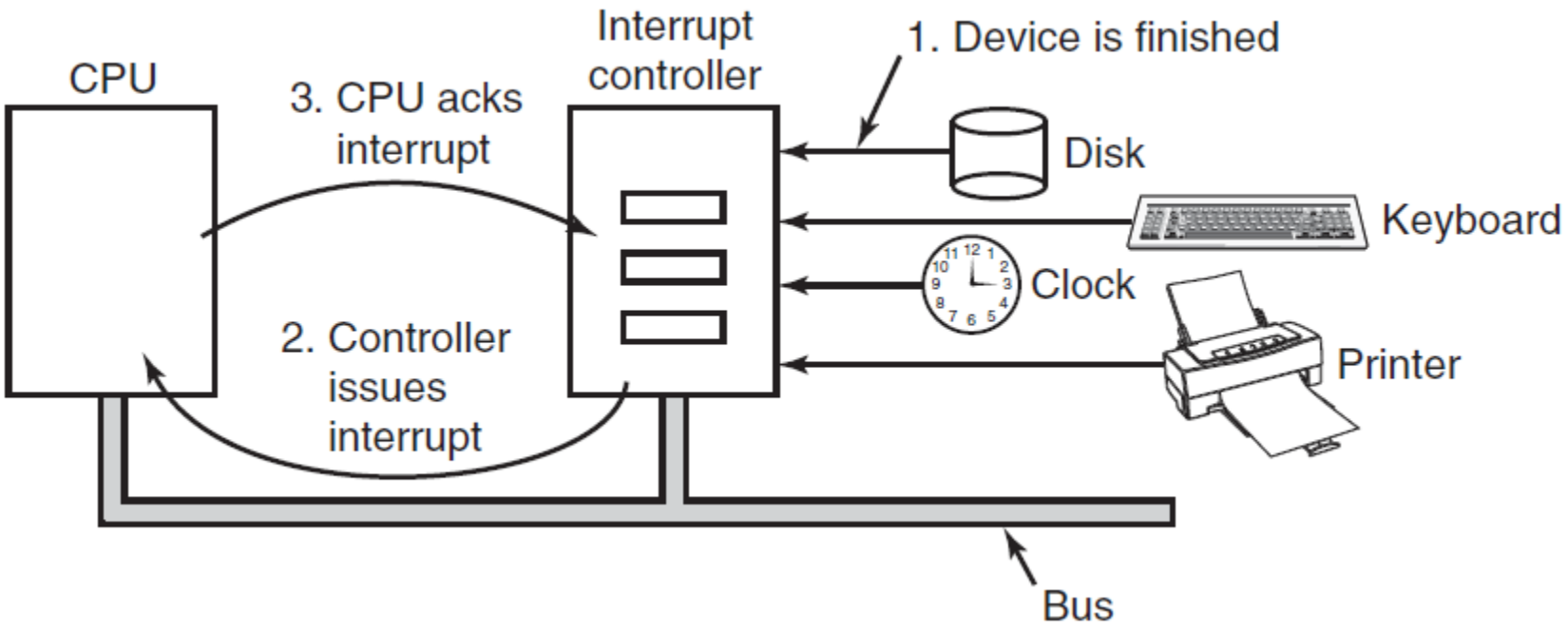



**Figure 6-6. How an interrupt happens. The connections between the devices and the controller actually use interrupt lines on the bus rather than dedicated wires.**

- An interrupt that leaves the machine in a well-defined state is called a **precise interrupt** (Walker and Cragon, 1995).

Properties of a precise interrupt:

1. PC (Program Counter) is saved in a known place.
2. All instructions before the one pointed to by the PC have fully executed.
3. No instruction beyond the one pointed to by the PC has been executed.
4. Execution state of the instruction pointed to by the PC is known.

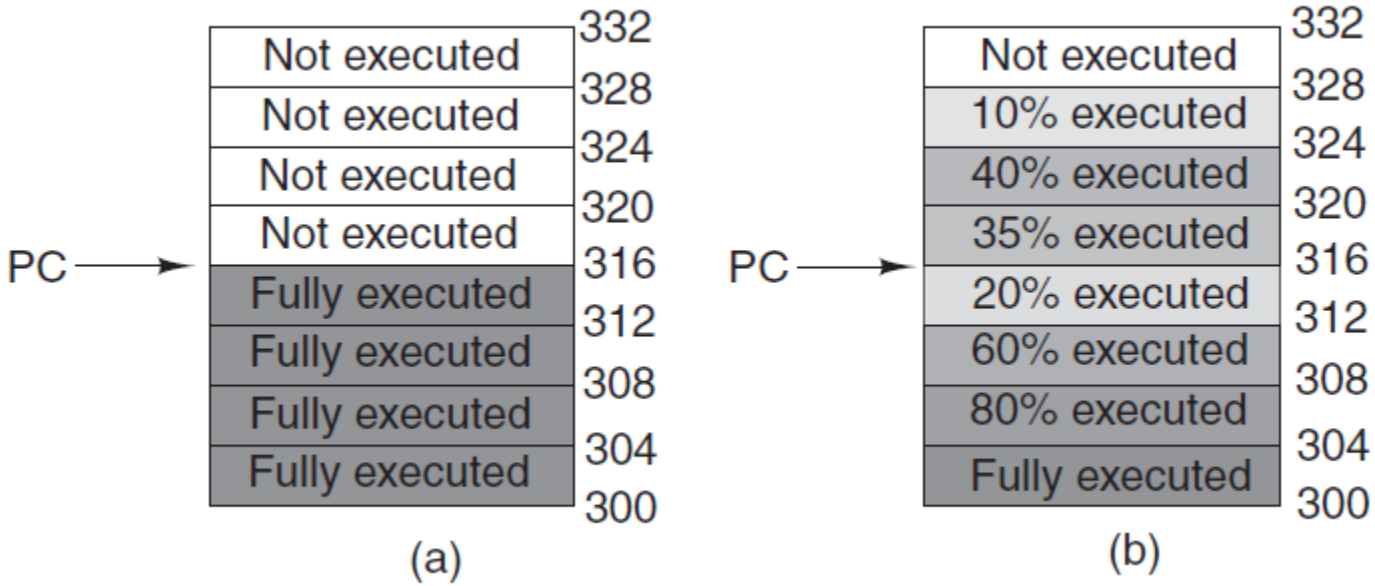



**Figure 6-7. (a) A precise interrupt. (b) An imprecise interrupt.**

- Fig. 5-7(a) illustrates a precise interrupt. All instructions up to the program counter (316) have completed and none of those beyond it have started (or have been rolled back to undo their effects).
- An interrupt that does not meet these requirements is called an **imprecise interrupt** and makes life most unpleasant for the operating system writer, who now has to figure out what has happened and what still has to happen.
- Fig. 5-7(b) illustrates an imprecise interrupt, where different instructions near the program counter are in different stages of completion, with older ones not necessarily more complete than younger ones.
- Machines with imprecise interrupts usually vomit a large amount of internal state onto the stack to give the operating system the possibility of figuring out what was going on.

# Principles of I/O Software

**Goals of the I/O Software**

- A key concept in the design of I/O software is known as **device independence.**
- Device independence is the goal of **uniform naming**. The name of a file or a device should simply be a string or an integer and not depend on the device in any way.
- issue for I/O software :
- **Error handling**.
- **Synchronous** (blocking) vs. **asynchronous** (interrupt-driven) transfers.
- **Buffering**.
- sharable vs. dedicated devices: Some I/O devices, such as disks, can be used by many users at the same time. No problems are caused by multiple users having open files on the same disk at the same time.

- Other devices, such as printers, have to be dedicated to a single user until that user is finished.
- Fundamentally input/output can be performed in one of the following three ways: (Handling I/O):

**Programmed I/O**

- Programmed input–output is a method of data transmission, via input/output, between a central processing unit and a peripheral device, such as a network adapter or a Parallel Advanced Technology Attachment(PATA) storage device

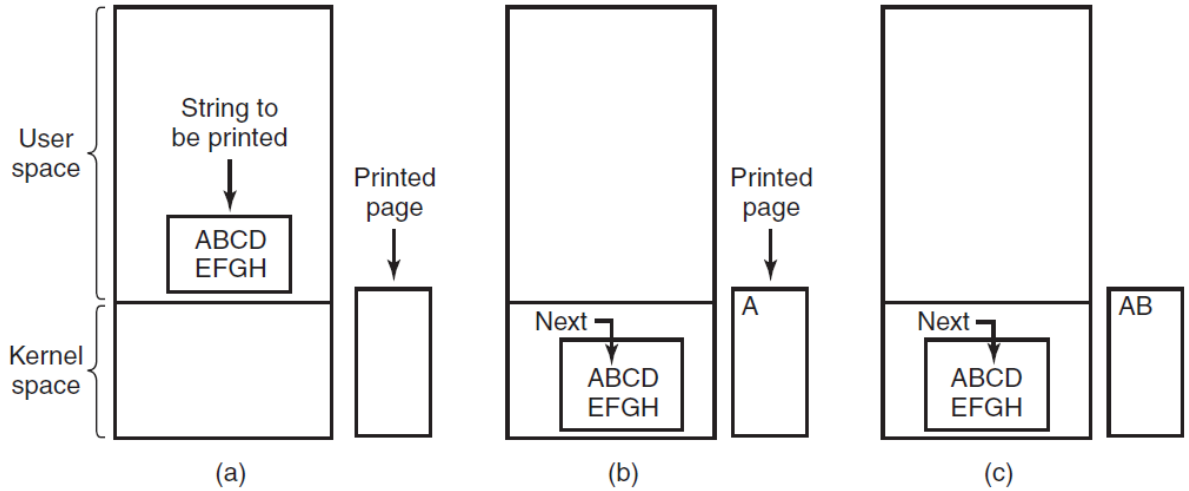



**Figure 6-8. Steps in printing a string.**

```
copy from user(buffer, p,count);                /* p is the ker nel buffer */
for (i = 0; i < count; i++) {                   /* loop on every character */
while (*pr inter status reg != READY) ;         /* loop until ready */
*pr inter data register = p[i];                 /* output one character */
}
retur n to user( );
```

**Figure 6-9. Writing a string to the printer using programmed I/O.**

# Interrupt-Driven I/O

```
copy_from_user(buffer, p, count);
enable_interrupts( );
while (*printer_status_reg != READY) ;
*printer_data_register = p[0];
scheduler( );
```

(a)

```
if (count == 0) {
    unblock_user( );
} else {
    *printer_data_register = p[i];
    count = count – 1;
    i = i + 1;
}
acknowledge_interrupt( );
return_from_interrupt( );
```

(b)

**Figure 6-10. Writing a string to the printer using interrupt-driven I/O. (a) Code executed at the time the print system call is made. (b) Interrupt service procedure for the printer.**

## I/O using DMA

- Disadvantage of interrupt-driven I/O is that an interrupt occurs on every character. Interrupts take time, so this scheme wastes a certain amount of CPU time. A solution is to use DMA.
- DMA is programmed I/O, only with the DMA controller doing all the work, instead of the main CPU. This strategy requires special hardware (the DMA controller) but frees up the CPU during the I/O to do other work.

```
copy_from_user(buffer, p, count);
set_up_DMA_controller( );
scheduler( );
```

(a)

```
acknowledge_interrupt( );
unblock_user( );
return_from_interrupt( );
```

(b)

**Figure 6-11. Printing a string using DMA. (a) Code executed when the print system call is made. (b) Interrupt-service procedure.**

# I/O Software layers

- I/O software is typically organized in four layers, as shown in Fig. 5-11. Each layer has a well-defined function to perform and a well-defined interface to the adjacent layers.

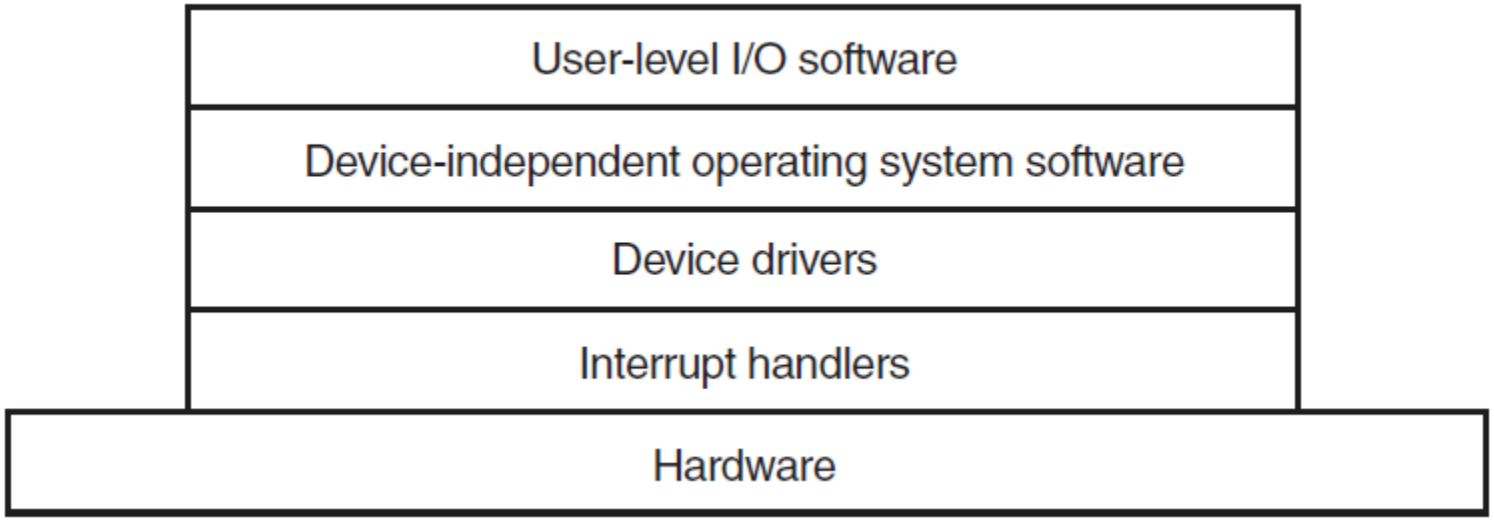



**Figure 6-12. Layers of the I/O software system.**

The four input/output software layers that are listed above are:

**Interrupt handlers**

1.Save registers not already been saved by interrupt hardware.

2. Set up a context for the interrupt service procedure.

3. Set up a stack for the interrupt service procedure.

4. Acknowledge the interrupt controller. If there is no centralized interrupt controller, reenable interrupts.

5. Copy the registers from where they were saved to the process table.

6. Run the interrupt service procedure.

7. Choose which process to run next.

8. Set up the MMU context for the process to run next.

9. Load the new process' registers, including its psw(program status word).

10. Start running the new process.

# Device Drivers

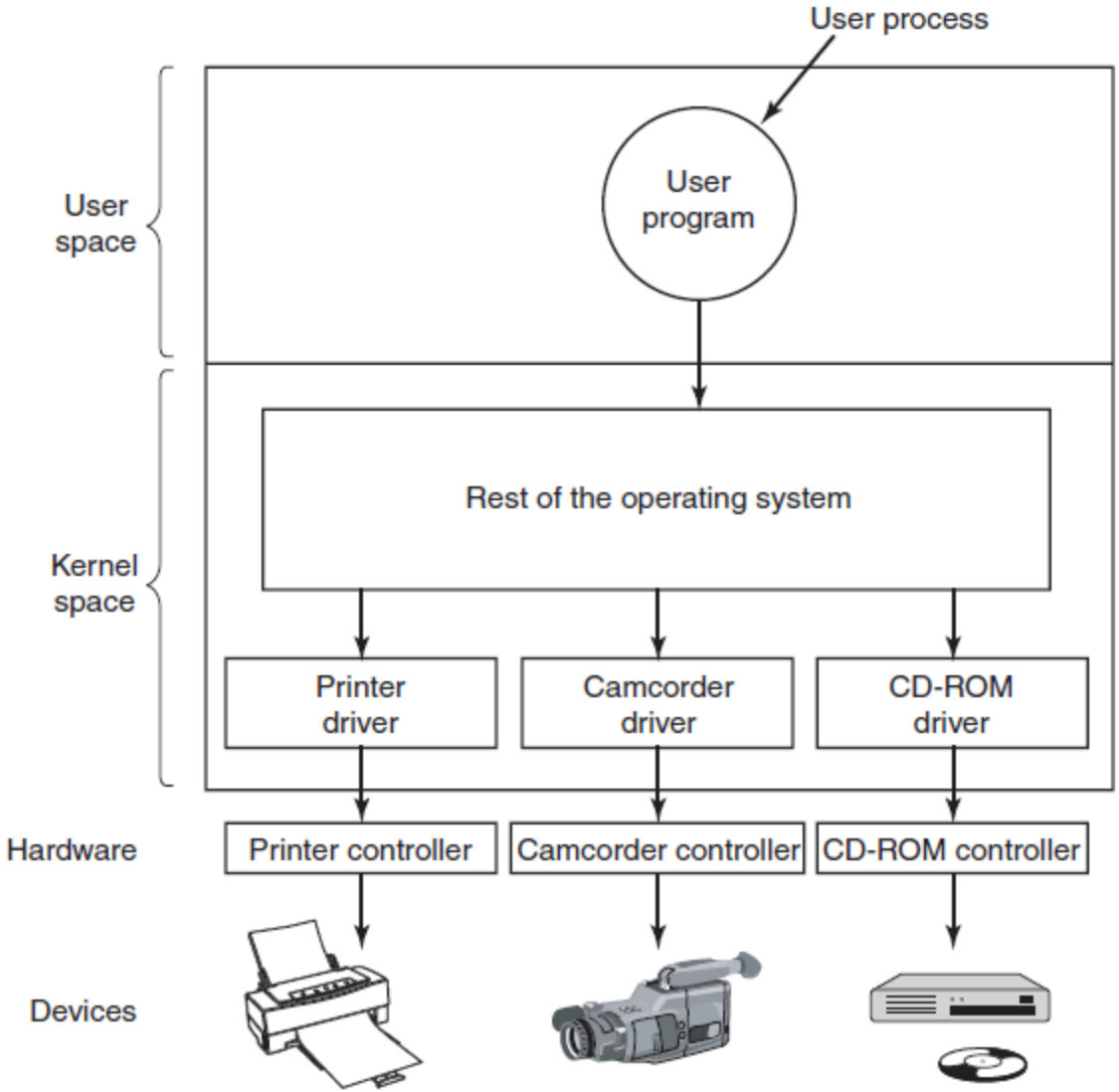



**Figure 6-13. Logical positioning of device drivers. In reality all communication between drivers and device controllers goes over the bus.**

## Device-Independent I/O Software

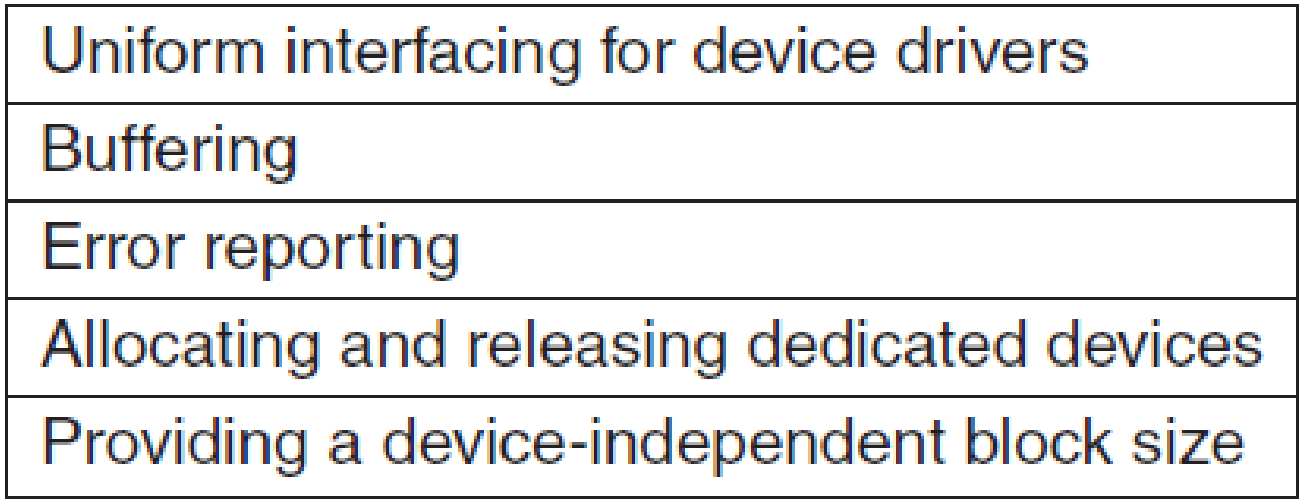



**Figure 6-14. Functions of the device-independent I/O software.**

## User-Space I/O Software

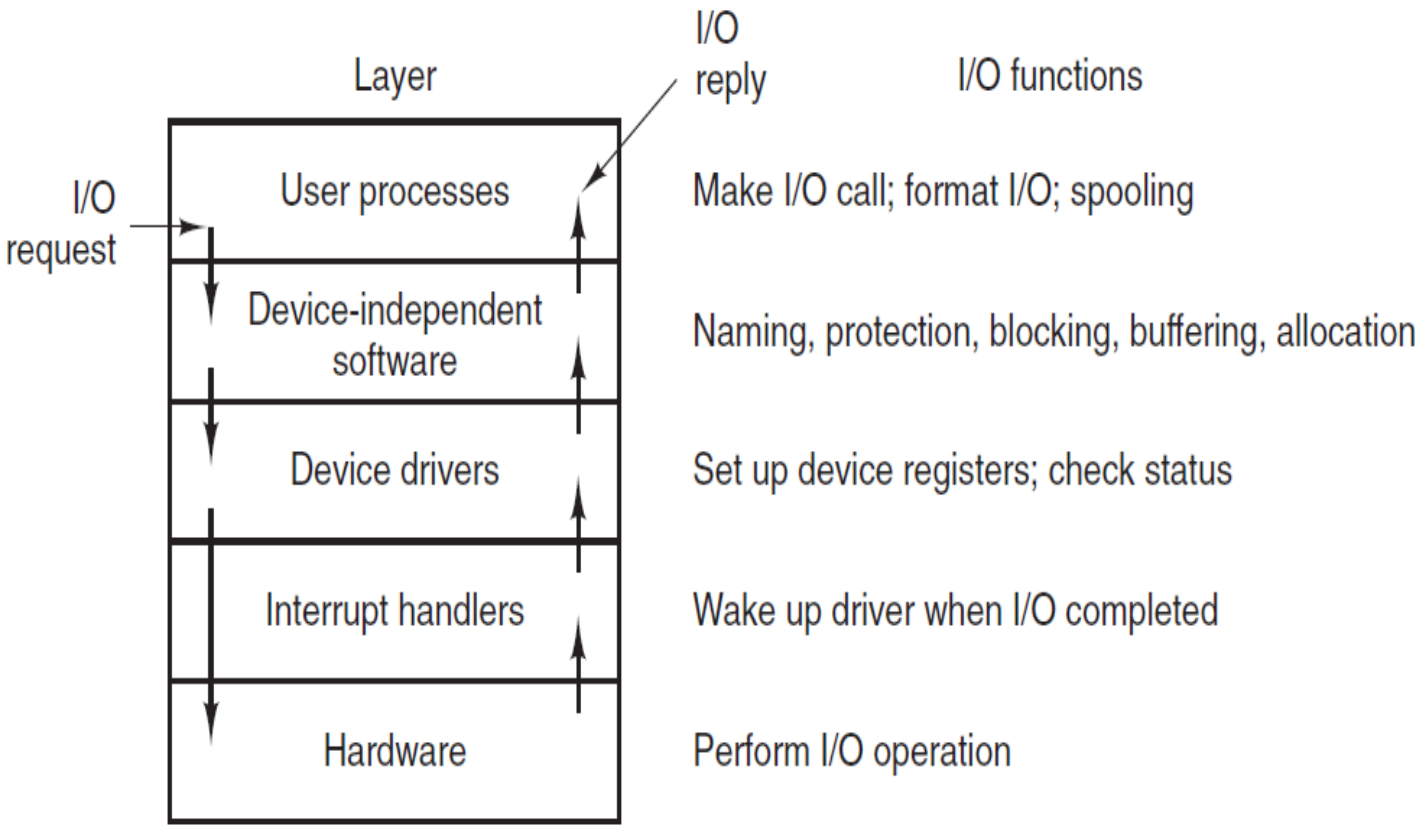



**Figure 6-15. Layers of the I/O system and the main functions of each layer.**

# Disks

## Disk Hardware

- **Magnetic Hard Disks**
  - Most common type of disk.
  - Read and write speeds are similar.
  - Ideal for secondary memory uses such as: Paging (virtual memory), File systems, General data storage
- **Optical Disks**
  - Examples: DVDs, Blu-ray discs, Commonly used for distribution of : Programs, Data, Movies

**Disk Formatting Overview**

- A hard disk consists of a stack of aluminum, alloy, or glass platters typically 3.5 inch in diameter (or 2.5 inch on notebook computers).
- On each platter is deposited a thin magnetizable metal oxide.
- Disk formatting prepares a magnetic disk for use by an operating system.
- It involves organizing the disk into **tracks**, **sectors**, and sometimes **cylinders**.
- There are two major types of formatting:
  - **Low-level formatting:** Physically divides the disk into sectors and tracks (usually done at the factory).
  - **High-level formatting:** Creates the file system structures (like the root directory and file allocation tables).

- **Preamble**: Contains a special bit pattern to mark the start of a sector and includes cylinder and sector numbers.
- **Sector Size**: Most disks use 512 bytes per sector.
- **ECC (Error Correction Code) Field**:
  - Stores redundant information to help recover from read errors.
  - Size varies; 16-byte ECC fields are common.
- **Spare Sectors**:
  - Hard disks include extra sectors to replace any defective sectors caused by manufacturing faults.


**Figure 5-21. A disk sector.**

**Cylinder Skew**

- Cylinder skew is a performance optimization for reading data sequentially across tracks on different platters (cylinders).
- When reading consecutive tracks, the disk's read/write head must move to the next track, which takes a little time (seek time).
- During this head movement, the disk continues to spin.
- Cylinder skew offsets the starting sector of each track to account for the time it takes to move the head, ensuring that when the head arrives at the next track, the first sector to be read is just arriving under the head.
- This avoids missing the beginning of the track and having to wait an entire rotation.
- Example: If track 0, sector 1 starts at 0°, track 1, sector 1 may start at 20°.

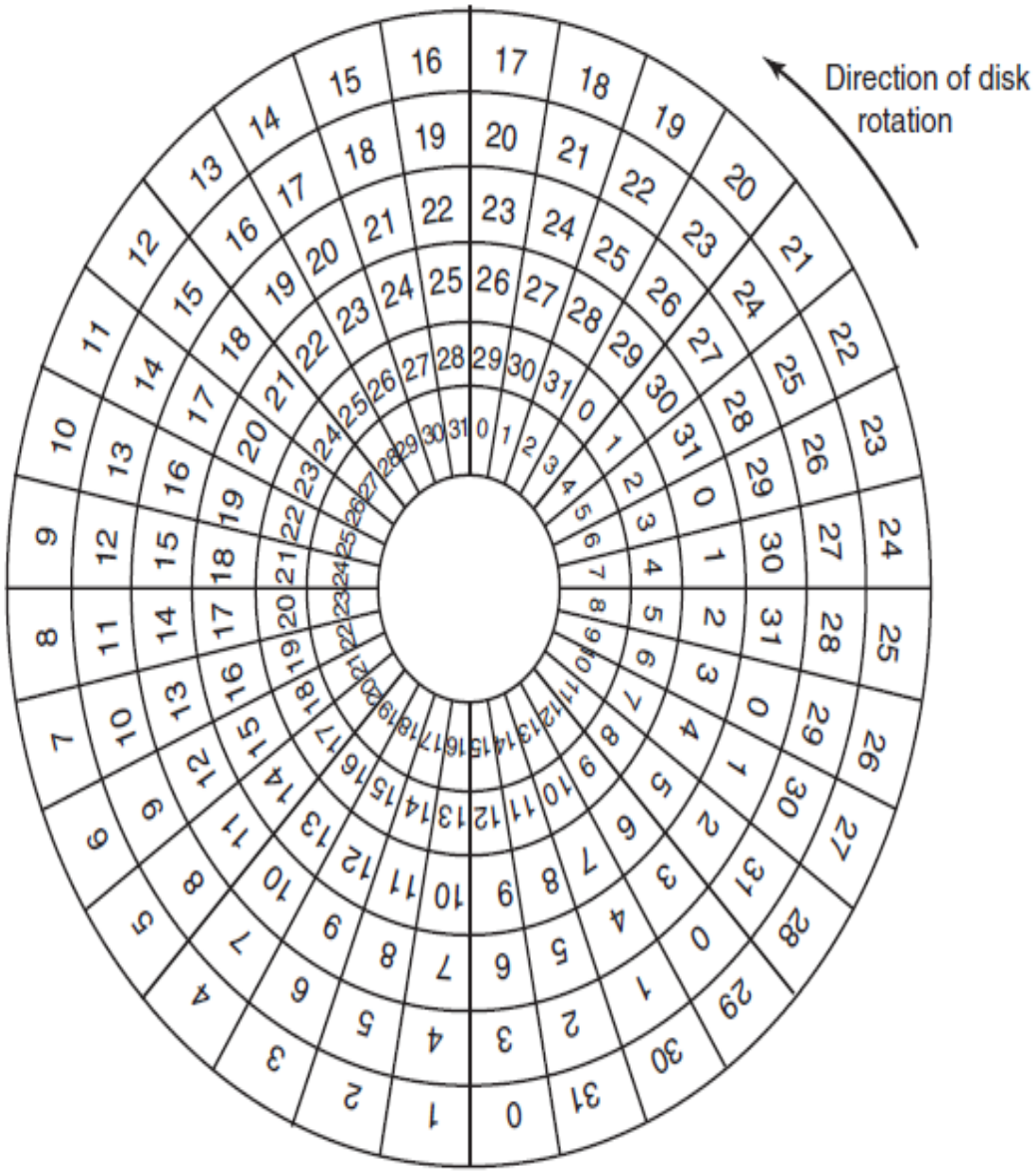



**Figure 5-22. An illustration of cylinder skew.**

**Interleaving**

- Interleaving (disk management technique) determines how sectors are physically arranged on a disk track to match the processing speed of the computer.
- Early computers couldn’t process data fast enough to read sequential sectors on spinning disks without missing some.
- For example, Sectors are spaced out (e.g., in a 2:1 interleave, they are arranged as in: 0, 3, 6, 1, 4, 7... ) so that the system has time to process one sector before the next one arrives under the read head.
- Modern drives and controllers are fast enough that interleaving is rarely needed today, but it was important in early computing.

***Issue:***

- When a sector is read, the disk controller first copies it to its buffer, and then it must be transferred to main memory.
- During this transfer, the system may not be ready to read the immediately adjacent next sector, causing it to miss that sector and wait for a full disk rotation.

***Solution-Interleaving:***

- Instead of placing sectors consecutively, sectors are spaced out on the track (e.g., 0, 3, 6, 1, 4...) so that by the time the system is ready for the next sector, it is just arriving under the read head.

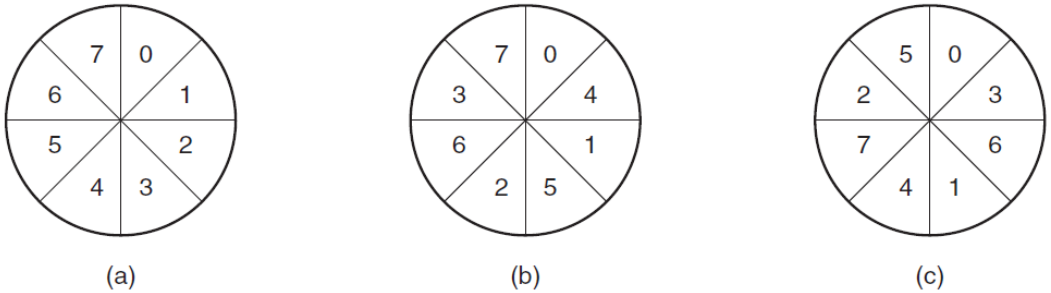



**Figure 5-23. (a) No interleaving. (b) Single interleaving. (c) Double interleaving.**

**Error Handling**

- Error handling during formatting is crucial because disks can have defects (bad sectors).
- During formatting, the disk is scanned for defects. Bad sectors are marked so that the file system does not use them.
- Two approaches:
  - **Physical formatting:** The disk controller finds bad sectors during low-level formatting and marks them so they're never used.
  - **Logical formatting (software level):** The operating system or file system can maintain a list of bad sectors, skipping them during data storage.

- **Error detection and correction codes (EDC/ECC):**
  - Extra bits are added to each sector to allow the controller to detect and sometimes correct errors automatically.
  - Common algorithms: parity bits, CRC (Cyclic Redundancy Check), Reed-Solomon codes.

# RAID

- **RAID (Redundant Array of Inexpensive/ Redundant Array of Independent Disks)**
- RAID is a storage technology that combines multiple physical disks into logical units to improve performance, data redundancy, or both.
- It typically uses two or more drives, including HDDs or SSDs.
- Unlike **SLED (Single Large Expensive Disk)**, which risks total data loss on failure, RAID distributes data across disks, reducing the impact of individual disk failures.
- However, as the number of disks increases, so does the risk of failure-making RAID essential for reliable and efficient storage.

- A typical reliable disk has a failure rate of 1 in 100,000 hours.
- In an array of N disks, the overall reliability becomes: **Reliability = (100,000 hours) / N**
- With 100 disks, Reliability = 100,000 / 100 = 1,000 hours ≈ 41.66 days**,** A failure is expected roughly every 42 days!

Solution: Redundancy via RAID

**RAID levels:**

**RAID 0:** striping

**RAID 1:** mirroring

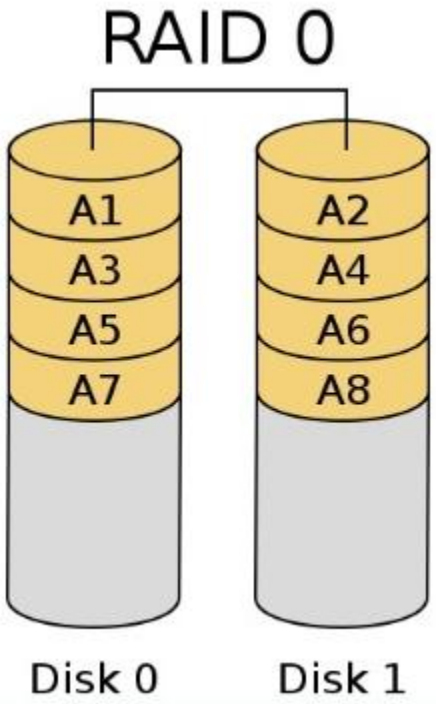
**RAID 5:** striping with parity

**RAID 6:** striping with double parity

**RAID 10:** combining mirroring and striping

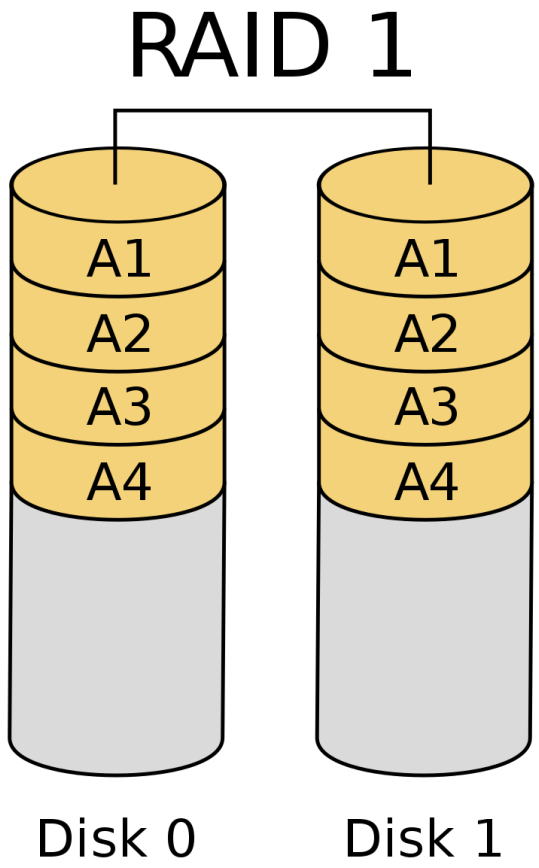
## RAID level 0 : Striping

- RAID Level 0 is block-level striping and non-redundant disk array
- Files are striped across disks, no redundant information
- Best I/O performance achieved when data is striped across multiple controllers with only one drive per controller.
- High read throughput but no fault-tolerance
- Best write throughput (no redundant info to write)
- Any disk failure results in data loss; sometimes a file, sometimes the entire volume



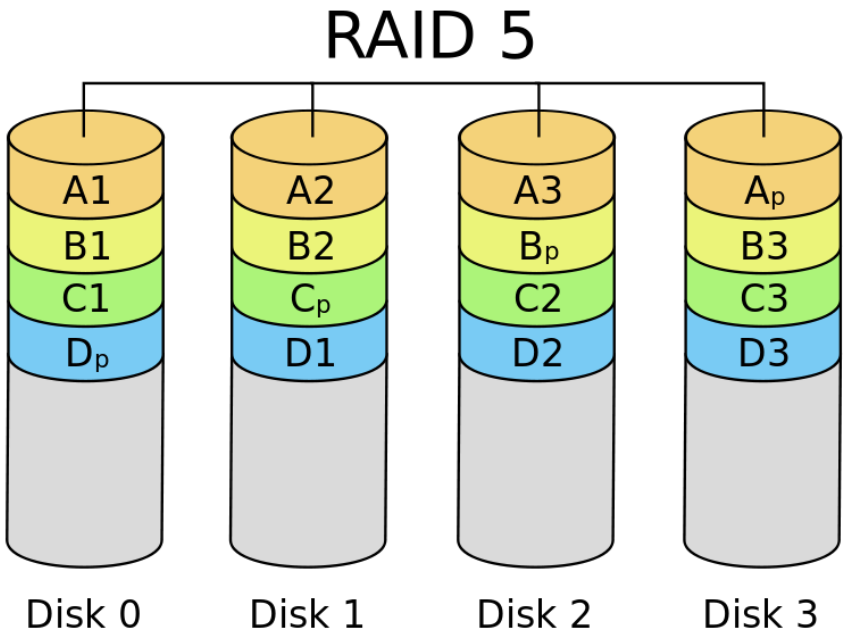
**RAID level 1 : Mirroring**

- RAID Level 1 is mirrored disks with no parity
- Files are striped across (half) the disks
- Data is written to multiple (two) places – data disks and mirror disks
- Best fault-tolerance
- On failure, just use the surviving disk(s)
- Factor of N (2x) space expansion



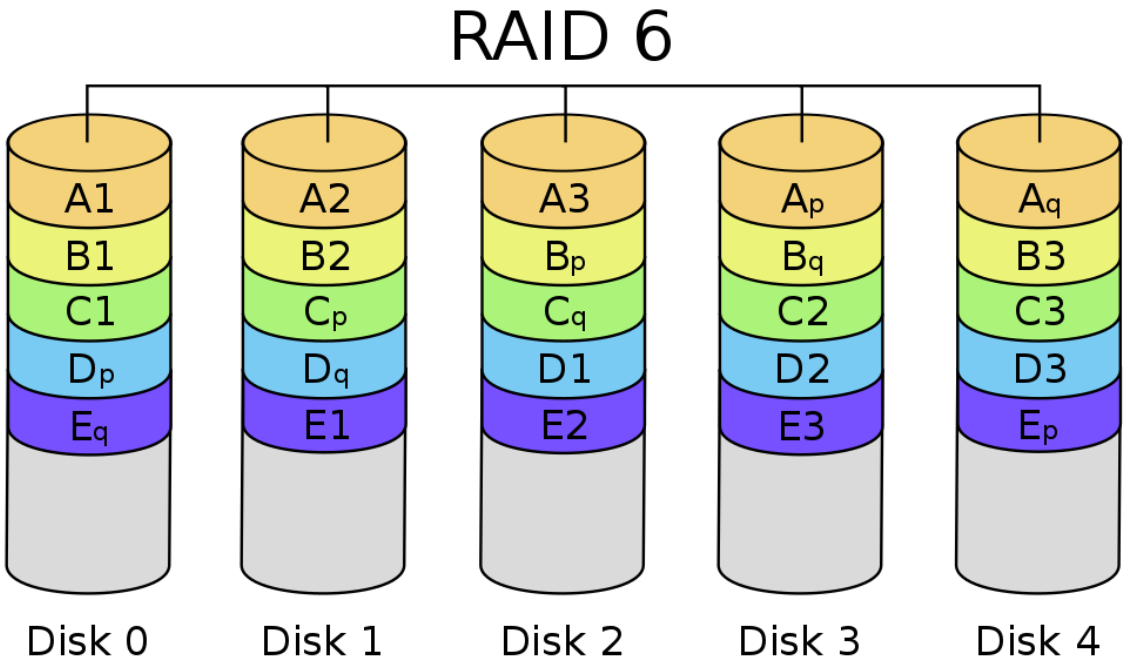
## RAID 5: striping with parity

- The parity information is distributed over all the disks instead of storing them in a dedicated disk.
- Parity for blocks in the same rank is generated on writes, recorded in a distributed location and checked on reads.
- No more a bottleneck as the parity stress evens out by using all the disks to store parity information
- No possibility of losing data redundancy since one disk does not store all the parity information.
- Can only handle up to a single disk failure



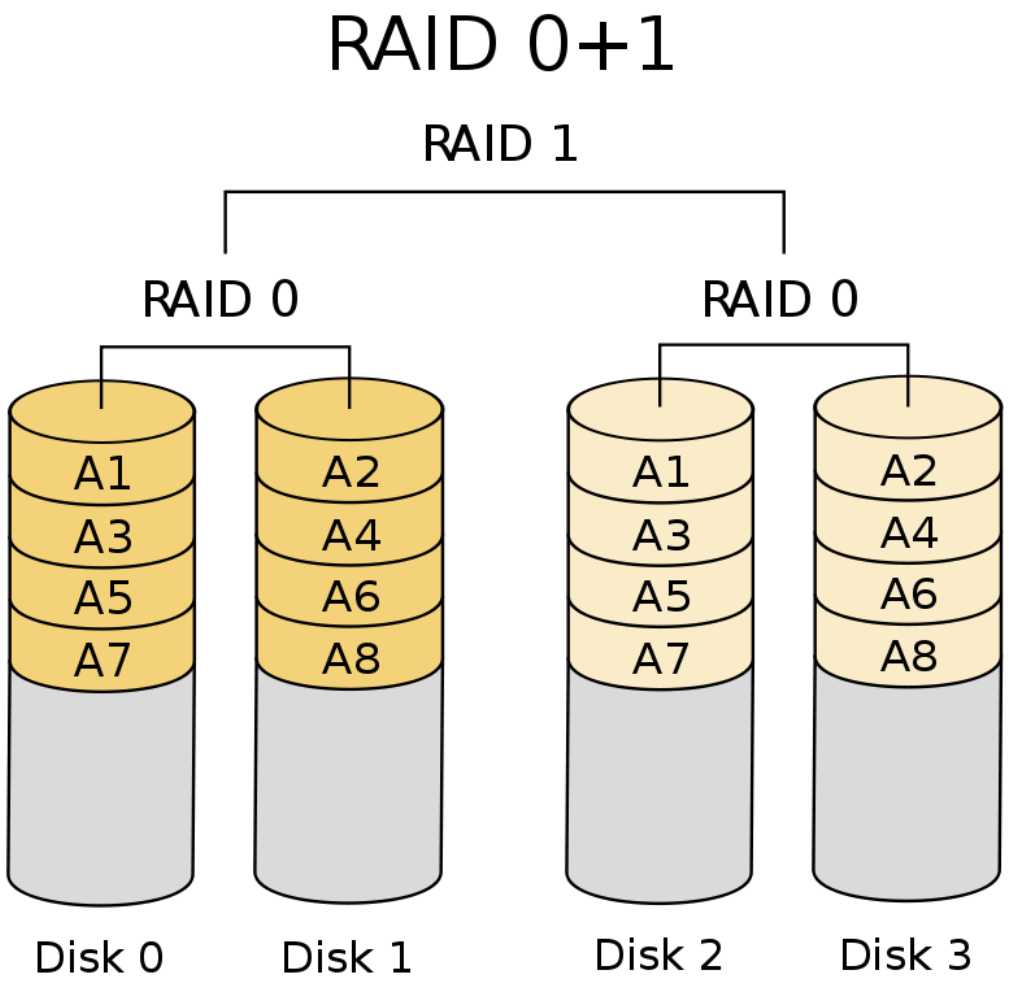
**RAID 6: striping with double parity**

- Block –level striping with double distributed parity.
- This increases the fault tolerance for up to two drive failures in the array.
- Each disk has two parity blocks which are stored on different disks across the array.
- RAID 6 is a very practical infrastructure for maintaining high availability systems.
- Large parity overhead



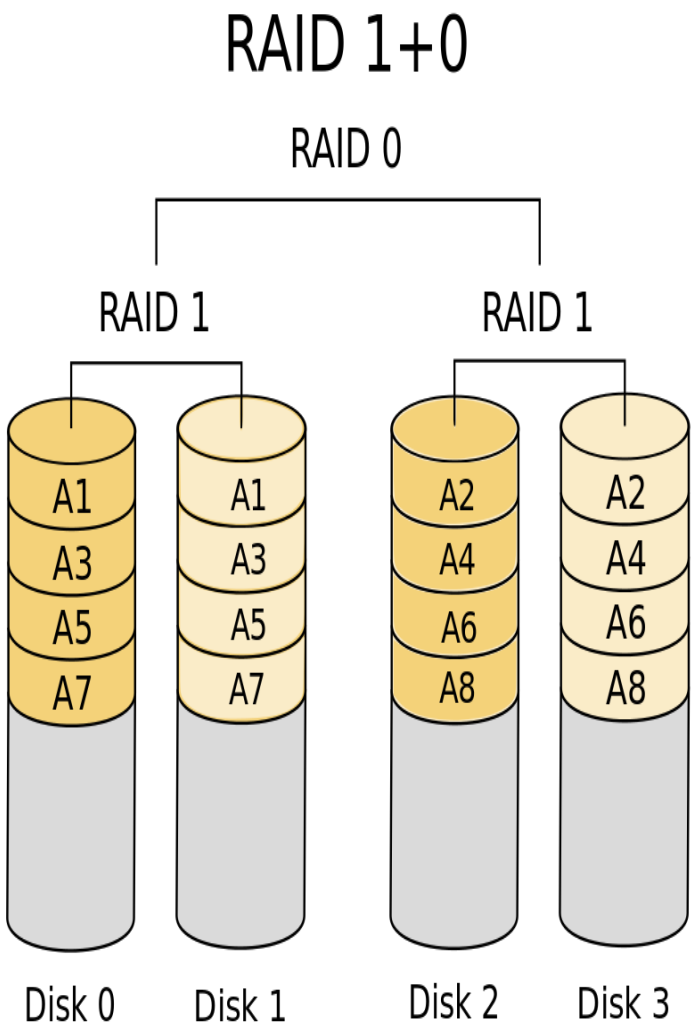
**RAID 01 (RAID 0+1)**

- RAID level using a mirror of stripes, achieving both replication and sharing of data between disks.
- The usable capacity of a RAID 01 array is the same as in a RAID 1 array made of the same drives, in which one half of the drives is used to mirror the other half.



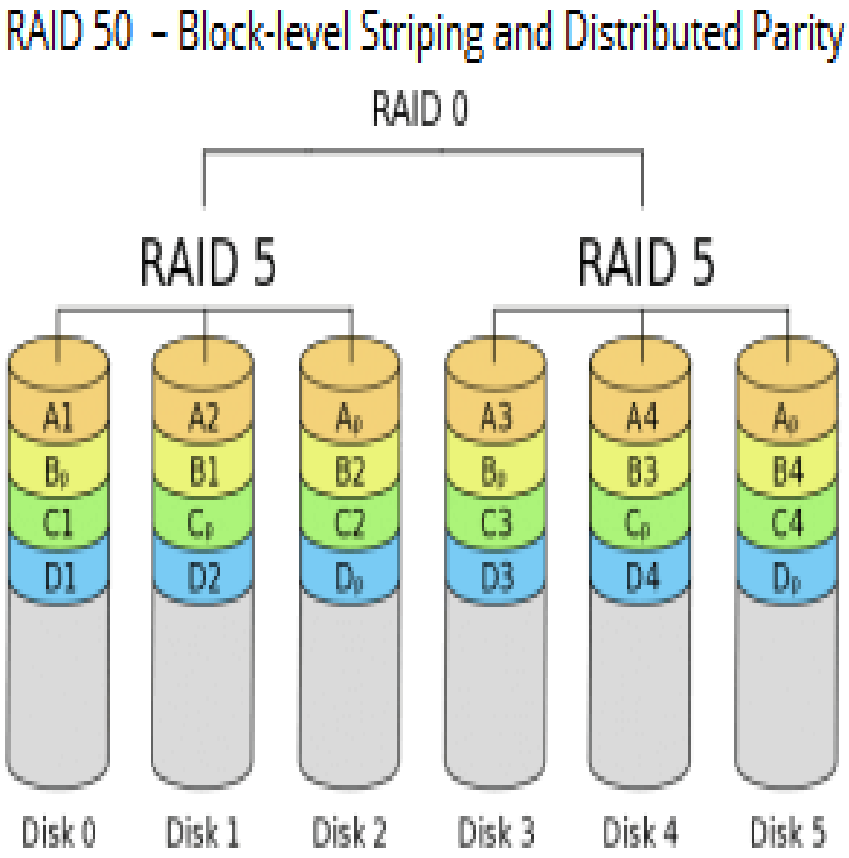
**RAID 10: Combining Mirroring And Striping**

- Also called **RAID 1+0** and sometimes **RAID 1 & 0**
- RAID 10 combines both RAID 1 and RAID 0 by layering them in opposite order.
- In this setup, multiple RAID 1 blocks are connected with each other to make it like RAID 0.
- This is a nested or hybrid RAID configuration.
- It is used in cases where huge disk performance (greater than RAID 5 or 6) along with redundancy is required.
- Cost per unit memory is high since data is mirrored



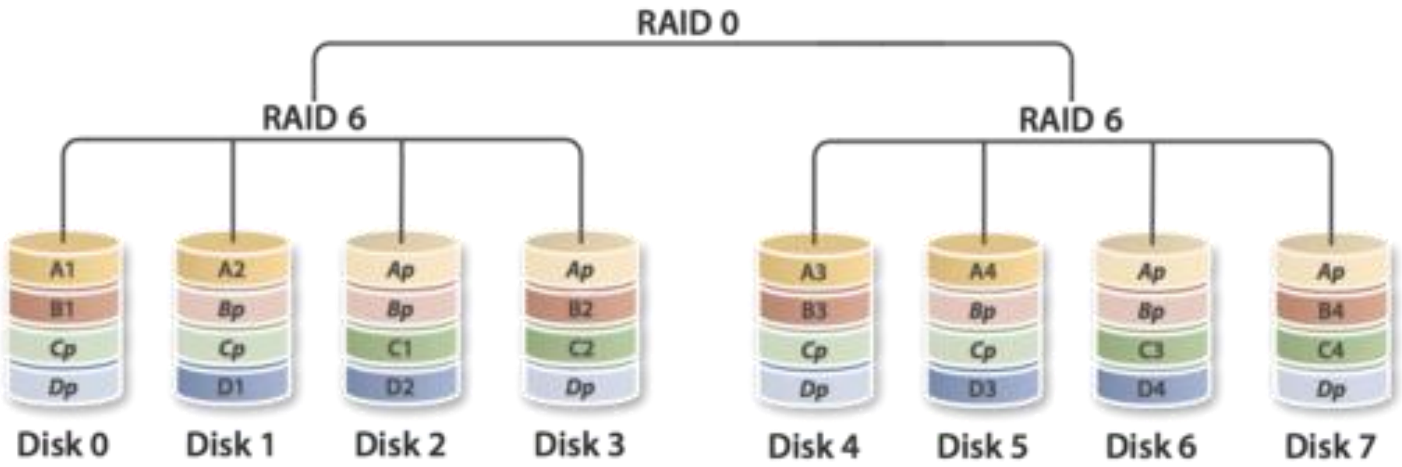
**RAID 50:**

- **RAID 50 a**lso called **RAID 5+0**
- combines the straight block-level striping of RAID 0 with the distributed parity of RAID 5**.**
- As a RAID 0 array striped across RAID 5 elements, minimal RAID 50 configuration requires six drives.
- This takes advantage of the distributed parity of the RAID 5 level with the extra speed gained by using the data striping of RAID 0.



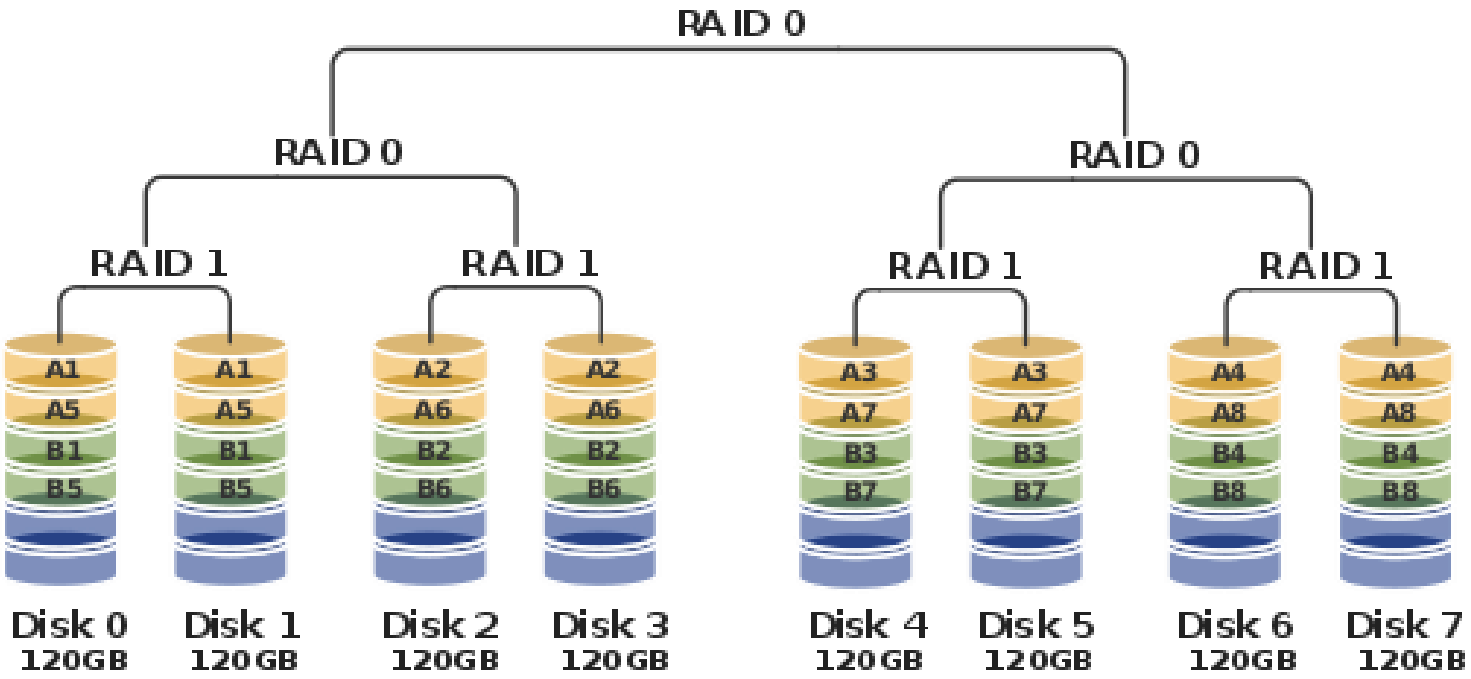
**RAID 60 (RAID 6+0)**

- **RAID 60**, also called **RAID 6+0**,
- Combines the straight block-level striping of RAID 0 with the distributed double parity of RAID 6, resulting in a RAID 0 array striped across RAID 6 elements. It requires at least eight disks



**RAID 100 (RAID 10+0)**

- **RAID 100**, sometimes also called **RAID 10+0**, is a stripe of RAID 10s.
- This is logically equivalent to a wider RAID 10 array, but is generally implemented using software RAID 0 over hardware RAID 10. Being "striped two ways", RAID 100 is described as a "plaid RAID"



# Disk Arm Scheduling Algorithms

- How long it takes to read or write a disk block. The time required is determined by three factors:

1. Seek time (the time to move the arm to the proper track or cylinder).

2. Rotational delay (how long for the proper sector to appear under the reading head).

3. Actual data transfer time(Actual time to transfer data once the read/write head is in position).

**Disk Scheduling Algorithms**

- FCFS Algorithm
- SSTF Algorithm
- SCAN Algorithm
- C-SCAN Algorithm
- LOOK Algorithm
- C-LOOK Algorithm

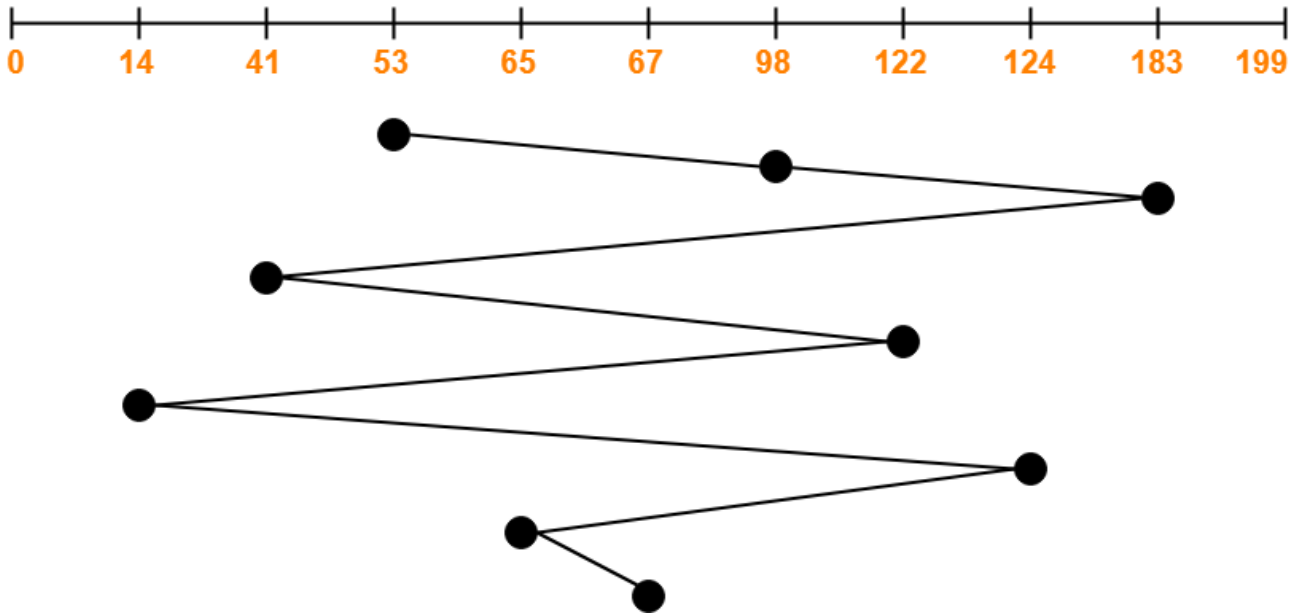
# FCFS Scheduling

- Services I/O requests strictly in the order they arrive.
- It is the simplest disk scheduling algorithm to implement.
- There is no starvation-every request will eventually be serviced.

Drawbacks:

- Does not optimize seek time or reduce head movement.
- Requests may come from different processes in a scattered order, leading to inefficient and excessive disk head movement.
- As a result, overall disk performance can be poor compared to more advanced algorithms

Q. A disk contains 200 cylinders ,the track sequence is 98, 183, 41, 122, 14, 124, 65, 67 and current position of R/W head is 53 calculate total no of arm movement of head.




Total head movements incurred while servicing these requests

= (98 – 53) + (183 – 98) + (183 – 41) + (122 – 41) + (122 – 14) + (124 – 14) + (124 – 65) + (67 – 65)= 45 + 85 + 142 + 81 + 108 + 110 + 59 + 2

= 632

## SSTF Scheduling

- SSTF stands for **Shortest Seek Time First**.
- This algorithm services that request next which requires least number of head movements from its current position regardless of the direction.
- It breaks the tie in the direction of head movement.

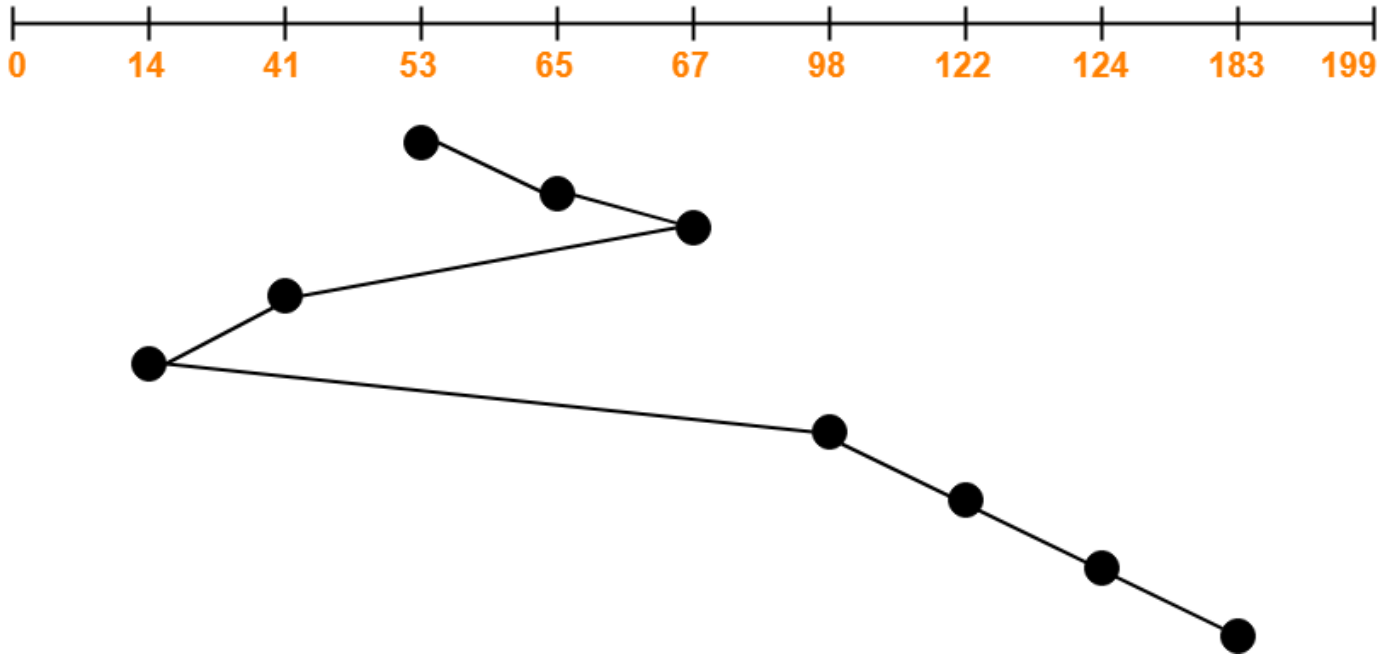
Advantages:

- It reduces the total seek time as compared to FCFS.
- It provides increased throughput.
- It provides less average response time and waiting time.

Disadvantages:

- There is an overhead of finding out the closest request.
- The requests which are far from the head might starve for the CPU.
- It provides high variance in response time and waiting time.
- Switching the direction of head frequently slows down the algorithm.

Q. Suppose that a disk drive has the cylinder numbered from 0 to 199. The head is currently at cylinder number 53. The queue for services of cylinder is as 98, 183, 41, 122, 14, 124, 65, and 67. What is the total head movement in each of the following disk algorithm to satisfy the requests?




Total head movements incurred while servicing these requests
= (65 – 53) + (67 – 65) + (67 – 41) + (41 – 14) + (98 – 14) + (122 – 98) + (124 – 122) + (183 – 124)= 12 + 2 + 26 + 27 + 84 + 24 + 2 + 59
= 236

## SCAN Scheduling

- Also called the **elevator** algorithm.
- The disk arm moves in one direction (inward or outward), servicing all pending requests along the way.
- Once it reaches the end (either the innermost or outermost cylinder), it reverses direction and continues servicing requests in the opposite direction.
- Mimics the operation of an elevator: it goes to one end before reversing, serving requests along its path in both directions.

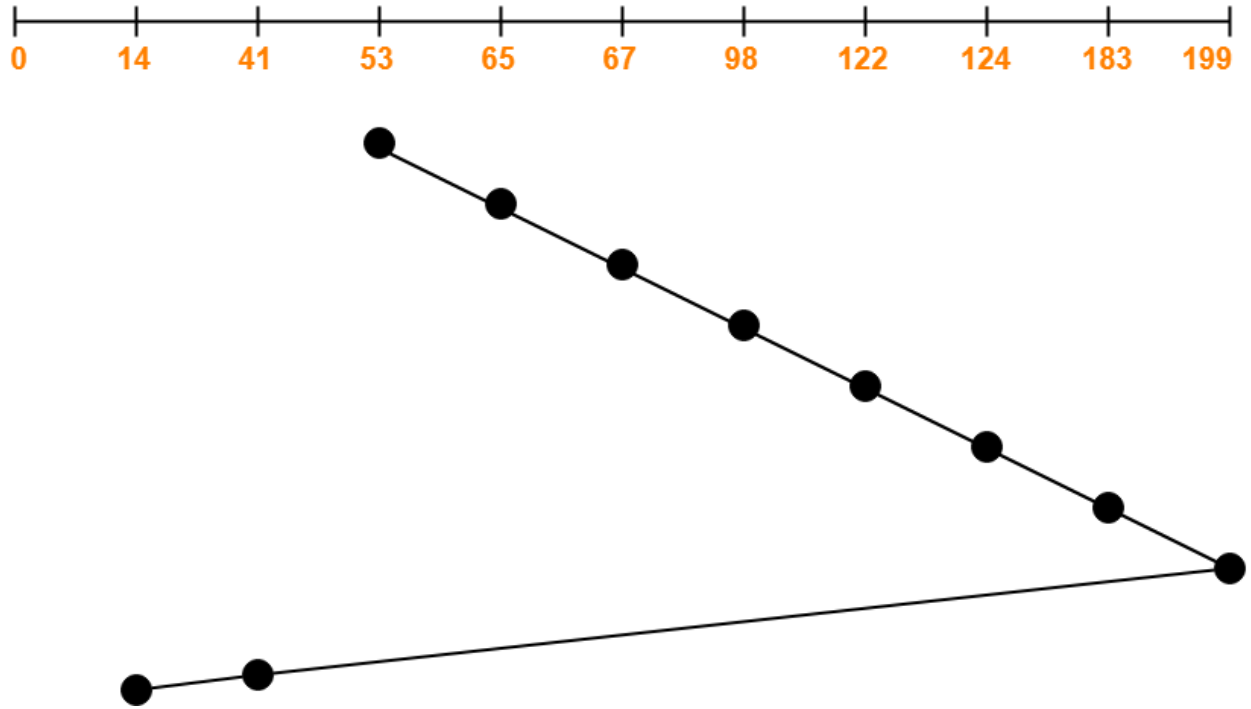
Advantages:

- High throughput: Services multiple requests efficiently in each sweep.
- Low variance in response time: More uniform wait times compared to FCFS.
- Better average response time than FCFS in heavy load conditions.

Disadvantages:

- Long waiting time for requests that arrive just after the head has passed their position.
- Slightly more complex to implement than simpler algorithms like FCFS or SSTF.

Q. Consider a disk queue with requests for I/O to blocks on cylinders 98, 183, 41, 122, 14, 124, 65, 67. The head is initially at cylinder number 53 moving towards larger cylinder numbers on its servicing pass. The cylinders are numbered from 0 to 199.




Total head movements incurred while servicing these requests = (65 – 53) + (67 – 65) + (98 – 67) + (122 – 98) + (124 – 122) + (183 – 124) + (199 – 183) + (199 – 41) + (41 – 14)

= 12 + 2 + 31 + 24 + 2 + 59 + 16 + 158 + 27

= 331

# C-SCAN Scheduling

- Circular-SCAN Algorithm is an improved version of the **SCAN Algorithm**.
- Head starts from one end of the disk and move towards the other end servicing all the requests in between.
- After reaching the other end, head reverses its direction.
- It then returns to the starting end without servicing any request in between. The same process repeats.

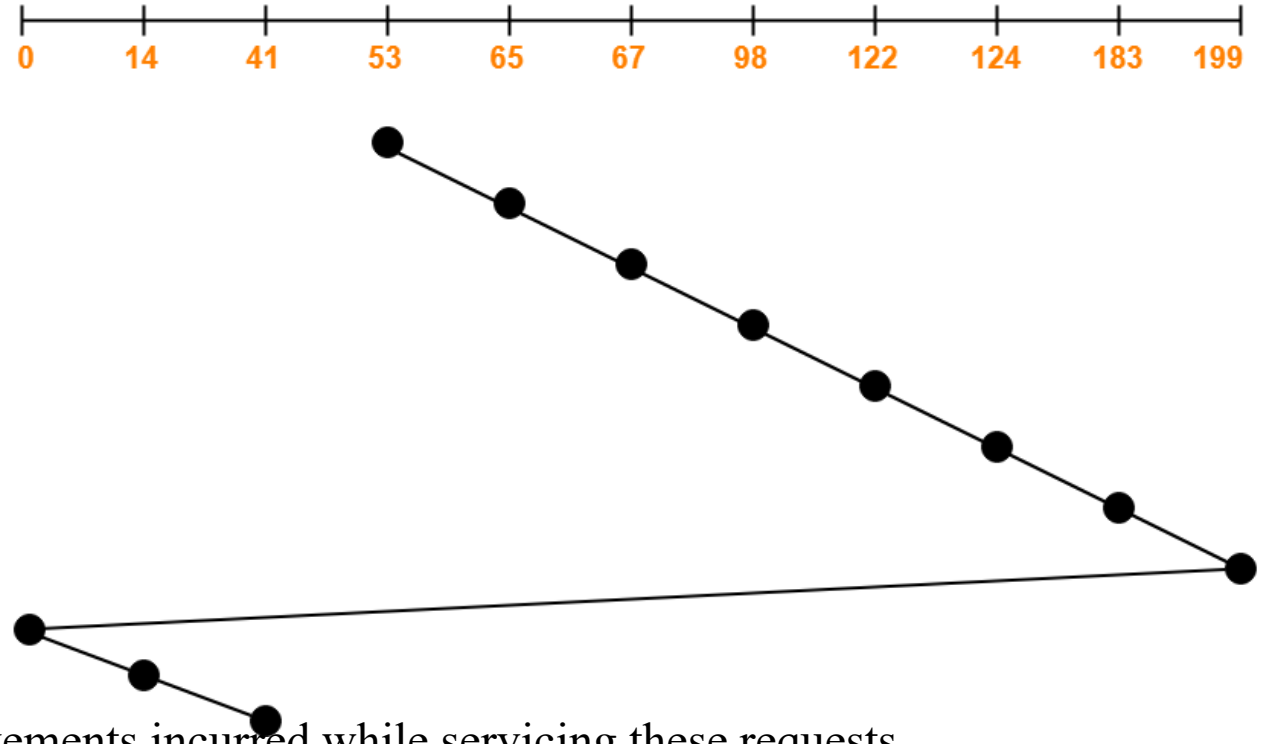
**Advantages:**

- The waiting time for the cylinders just visited by the head is reduced as compared to the SCAN Algorithm.
- It provides uniform waiting time.
- It provides better response time.

**Disadvantages:**

- It causes more seek movements as compared to SCAN Algorithm.
- It causes the head to move till the end of the disk even if there are no requests to be serviced.

Q. Suppose that a disk drive has the cylinder numbered from 0 to 199. The head is currently at cylinder number 53 moving towards larger cylinder numbers on its servicing pass. The queue for services of cylinder is as 98, 183, 41, 122, 14, 124, 65 and 67. What is the total head movement in each of the following disk algorithm to satisfy the requests?




Total head movements incurred while servicing these requests
= (65 – 53) + (67 – 65) + (98 – 67) + (122 – 98) + (124 – 122) + (183 – 124) + (199 –183) + (199 – 0) + (14 – 0) + (41 – 14)
= 12 + 2 + 31 + 24 + 2 + 59 + 16 + 199 + 14 + 27
= 386

# LOOK Scheduling

- LOOK Algorithm is an improved version of the **SCAN Algorithm.**
- Head starts from the first request at one end of the disk and moves towards the last request at the other end servicing all the requests in between.
- After reaching the last request at the other end, head reverses its direction.
- It then returns to the first request at the starting end servicing all the requests in between.
- The same process repeats.

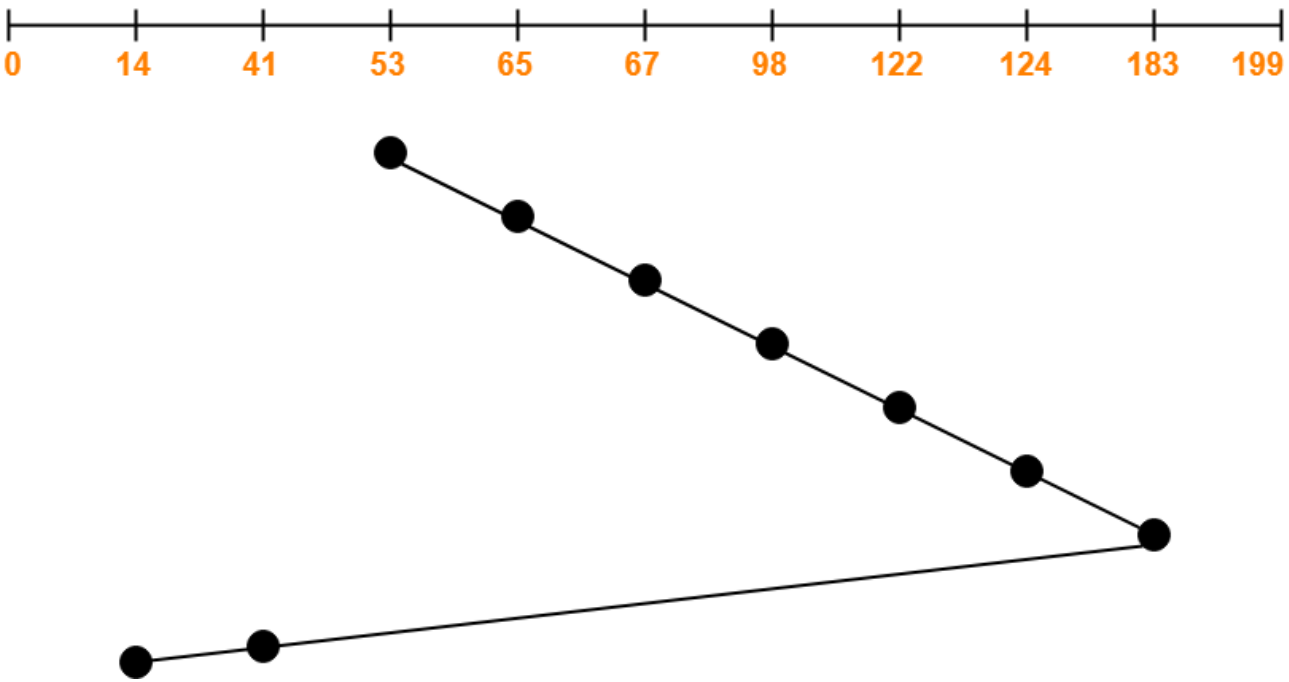
Advantages:

- It does not cause the head to move till the ends of the disk when there are no requests to be serviced.
- It provides better performance as compared to SCAN Algorithm.
- It does not lead to starvation.
- It provides low variance in response time and waiting time.

Disadvantages:

- There is an overhead of finding the end requests.
- It causes long waiting time for the cylinders just visited by the head.

Q. Consider a disk queue with requests for I/O to blocks on cylinders 98, 183, 41, 122, 14, 124, 65, 67. The head is initially at cylinder number 53 moving towards larger cylinder numbers on its servicing pass. The cylinders are numbered from 0 to 199.




Total head movements incurred while servicing these requests

= (183 – 53) + (183 – 14)

= 130 + 169

= 299

# C-LOOK Scheduling

- C-LOOK is an optimized version of the LOOK algorithm.
- Like LOOK, it services requests in one direction until the last request is reached.
- However, instead of reversing direction to continue servicing, C-LOOK jumps directly back to the first request on the opposite end, without servicing requests on the return trip.
- The disk head moves in one direction (e.g., from the lowest to highest cylinder), servicing all requests along the way.
- Upon reaching the last request in that direction, the head jumps back to the first pending request at the opposite end.
- The process repeats in this circular fashion.

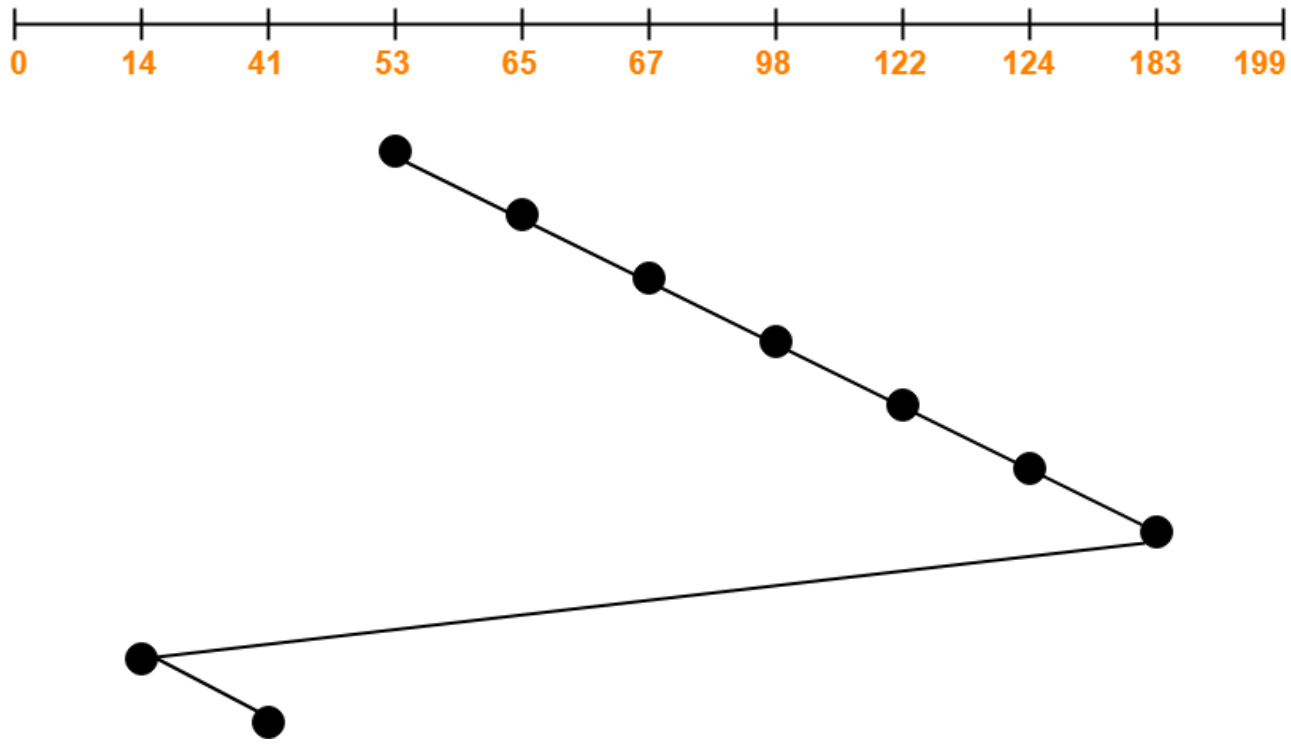
Advantages:

- Avoids unnecessary head movement to the physical ends of the disk.
- Reduces waiting time for recently serviced cylinders.
- Improves performance compared to standard LOOK.
- Prevents starvation by ensuring all requests are eventually serviced.
- Provides low variance in response and waiting times.

Disadvantages:

- Slight overhead in tracking the end (furthest) requests to decide jump points.
- Jumping back adds a small-time cost (though no actual servicing occurs during the return).

Q. Consider a disk queue with requests for I/O to blocks on cylinders 98, 183, 41, 122, 14, 124, 65, 67. The head is initially at cylinder number 53 moving towards larger cylinder numbers on its servicing pass. The cylinders are numbered from 0 to 199.




Total head movements incurred while servicing these requests
= (183 – 53) + (183 – 14) + (41 – 14)
= 130 + 169 + 27
= 326

- **Inward / Innermost Track**: Has the **lowest cylinder number.**
- **Outward/ Outermost Track:** Has the **highest cylinder number.**

Q. Find the seek time using SCAN, C-SCAN, Look and C-Look disk scheduling algorithms for processing the following request queue: 35, 70, 45, 15, 65, 20, 80, 90, 75, 130. Suppose the disk has tracks numbered from 0 to 150 and assume the disk arm to be at 30 and moving outward.