I/O DEVICE MANAGER

✓Besides providing abstractions like processes, address spaces and files, and OS also controls all of the computer's I/O devices ✓For that, the OS should:

- issue commands to handle the I/O devices
- catch the interrupts
- handle the errors
- provide the interface between devices and rest of the system.

PRINCIPLES OF I/O HARDWARE

✓Different people look at I/O hardware differently

- Electrical engineers look in terms of chips, wires, motors, power supplies and other physical components that make up the hardware
- Programmers look at the interface presented to the software (the commands the hardware accepts, the function it carries out and the errors that can be reported back)

✓We are concerned with programming I/O devices, not designing, building or maintaining them.

I/O DEVICES

✓I/O devices can be roughly divided into :

1 . Block devices

- stores information in a fixed sized block, each one with its own address.
- common block size ranges from 512 bytes to 32,768 bytes
- data transfer takes place in blocks
- block addressable not byte addressable
- e.g. Hard disks, CD-ROMs, USB sticks etc

I/O DEVICES

1 . Character devices

- delivers or accepts a stream of characters, without regard to any block structure

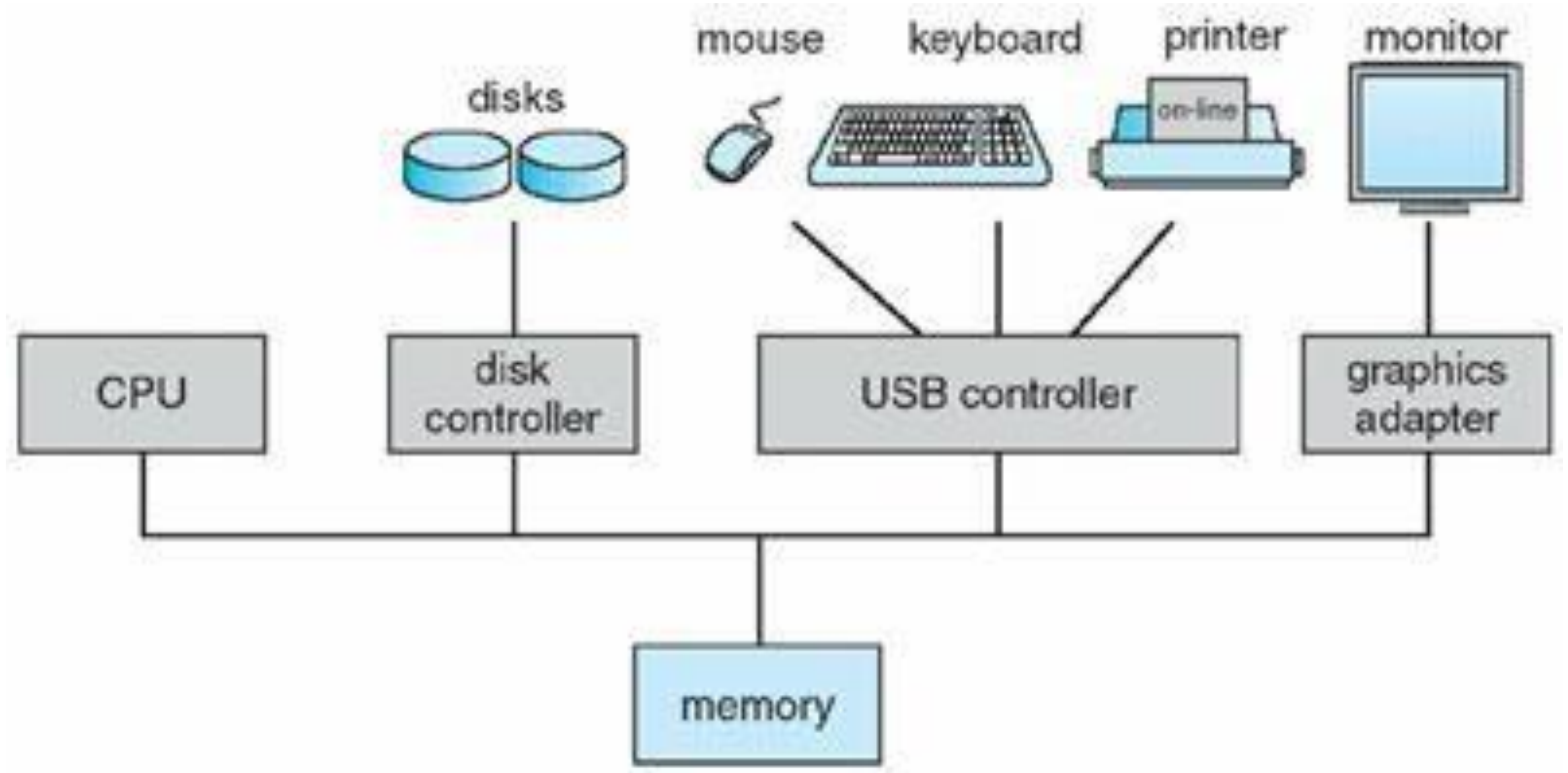
- not addressable

- printers, network interfaces, mice etc.

DEVICE CONTROLLERS

- ✓I/O devices typically consists of two components: electrical and mechanical
- ✓The electronic component is called the device controller or the adapter
- ✓A device controller is a part of a computer system that makes sense of the signals going to, and coming from the CPU
- ✓There are many device controllers in a computer system
- ✓ Any device connected to the computer is connected by a plug and socket, and the socket is connected to a device controller
- ✓In personal computer, device controller usually takes the form of a chip on the parentboard

✓Many controllers can handle two, four or even eight identical devices



✓Device controllers is a piece of hardware that receives commands from the system bus, translates them into device actions and reads/writes the data onto the system bus

✓Each controller has a few registers that are used for communicating with the CPU (control registers) along with a data buffer.

- ✓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.

Task : How does CPU communicate with control registers and device data buffers??

PRINCIPLES OF I/O SOFTWARE

Goals of I/O software

1. **Device independence**

   - it should be possible to write programs that can access any I/O device without having to specify the device in advance.

   - For example, a program that reads a file as input should be able to read a file on a hard disk, a CD-ROM, a DVD, or a USB stick without having to modify the program for each different device.

2. **Uniform naming**

   - The name of the device should simply be a string or an integer and do not depend on the device in any way

PRINCIPLES OF I/O SOFTWARE

Goals of I/O software

3. **Error Handling**

- In general, the error should be handled as close to the hardware as possible - Propagate errors up only when lower layer cannot handle it.

- For example, if controller discovers read error, it should try to correct the error itself. If it cannot, the device driver should handle it.

4. **Synchronous(blocking) Vs. Asynchronous (interrupt driven) transfers**

- It is up to the OS to make the operation that are interrupt driven look blocking to the user programs

PRINCIPLES OF I/O SOFTWARE

Goals of I/O software

5. **Buffering**

- often, data that comes off a device cannot be stored directly to its final destination due to various reasons.
- buffer is the intermediate destination

6. **Dedicated Vs Shared devices**

- devices that can be used by many users at a time is sharable . E.g. Disk (multiple users can open files from the same disk at the same time)
- some devices have to be dedicated to a single person. E.g. tapes

I/O HANDLING

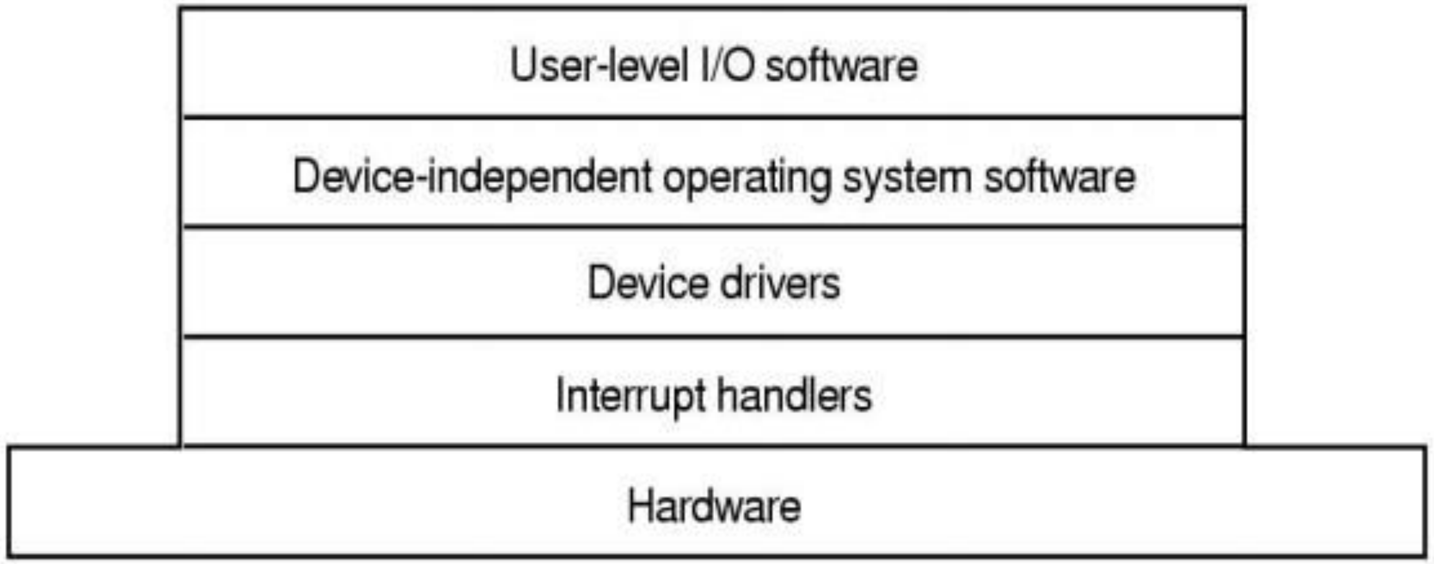
1. Programmed I/O
2. Interrupt driven I/O
3. I/O using DMA

I/O SOFTWARE LAYERS

✓I/O software is organized in four layers.

✓ Each layer has a well-defined function to perform and a well-defined interface to the adjacent layers.

✓The functionality and interfaces differ from system to system.



INTERRUPT HANDLER

Steps:

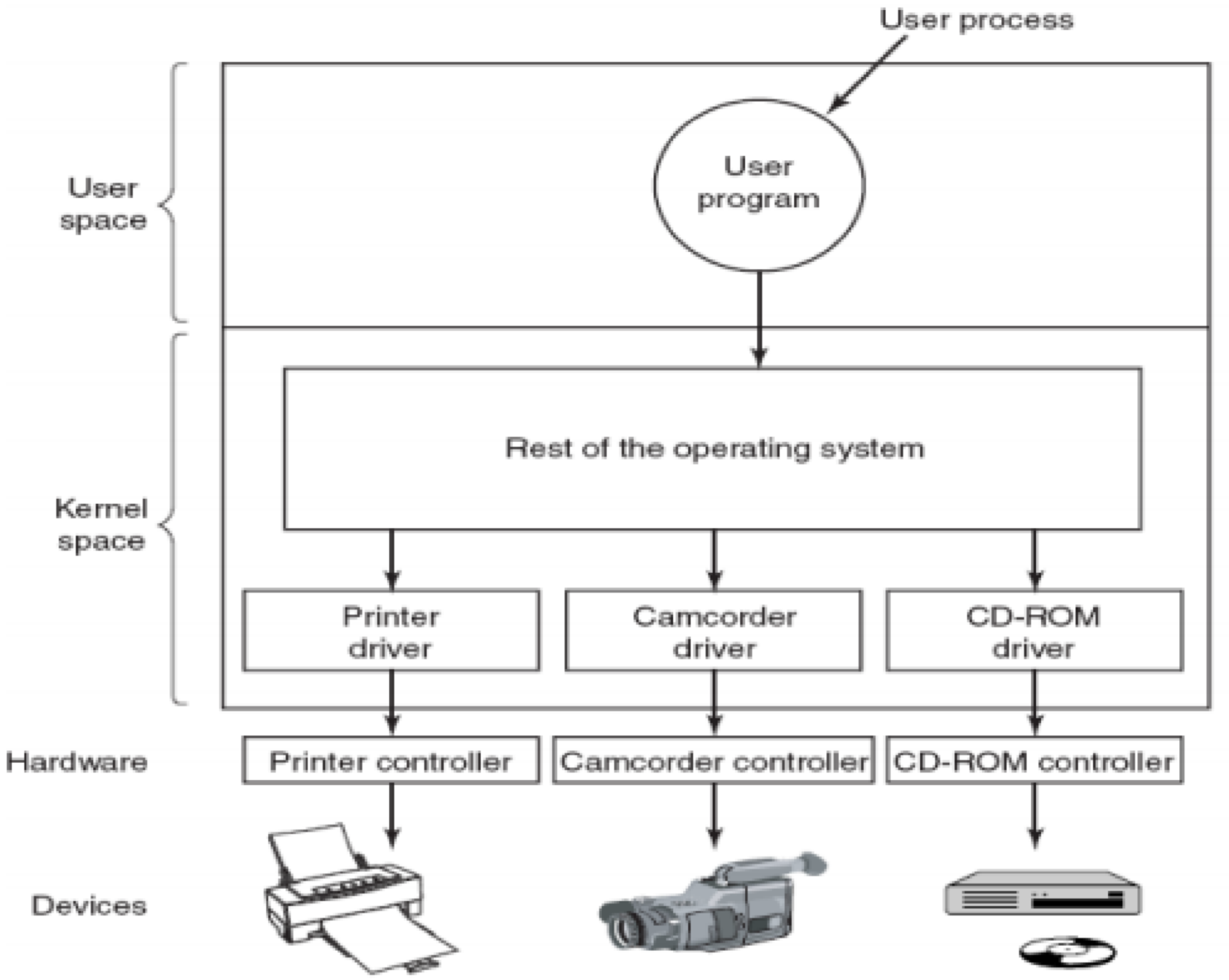
1. Save any registers(including PSW) not already saved by interrupt hardware.
2. Set up a context for the interrupt service procedure.(this may include Page table, TLB)
3. Set up a stack for interrupt service procedure.
4. Acknowledge the interrupt controller.
5. Copy the registers from where they were saved(possibly some stack) to the process table

INTERRUPT HANDLER

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.
10. Start running the new process.

DEVICE DRIVERS

- ✓ Device specific code for controlling the device is called device driver.
- ✓ Written by device's manufacturer and delivered along with the device.
- ✓ Device driver is the software part that communicates to the device controller, giving it commands and accepting response
- ✓ Different devices work differently, so the need for different drivers.
- ✓ A driver provides the software interface to hardware devices, enabling the OS and other programs to access hardware functions without having to now the precise details of the hardware being used
- ✓ Device divers are hardware dependent and operating system specific


User process
User space
User program
Kernel space
Rest of the operating system
Printer driver
Camcorder driver
CD-ROM driver
Hardware
Printer controller
Camcorder controller
CD-ROM controller
Devices

FUNCTIONS OF DEVICE DRIVERS

1. Accept read/write request from device independent I/O software above it.
2. Initialize the device if needed.
3. Manage its power requirements and log events
4. Check whether the input parameters are valid
5. Translate parameters from abstract to concrete terms (e.g, linear block number to CHS(Cylinder Head, Sector) for disk)
6. Check if the device is currently in use. If it is request will be queued for later processing.

WORKING MECHANISM OF DEVICE DRIVERS

- Firstly command sequence is determined.
- After knowing issued command sequences, it starts writing them into controller’s device registers.
- After that, it checks to see if the controller accepted the command and is prepared to accept next command.
- Driver blocks(or waits) till the controller does some work for it.
- It blocks itself until the interrupt comes into unblock it.
- Interrupt is sent to unblock the device driver.
- After operation has been completed, driver checks for error and if everything is all right, the driver passes data to the device independent software.

- Finally, it returns some status information for error reporting back the caller.
- If none of the request is in queue, the driver blocks waiting for next request.

DEVICE INDEPENDENT I/O SOFTWARE

✓Although some of the I/O software is device specific, other parts of it are device independent.

✓The basic function of device-independent software is to perform the I/O functions that are common to all devices and provide uniform interface to user-level software.

✓Functions of device independent I/O software

1. Uniform interfacing for device drivers
2. Buffering
3. Error Reporting
4. Allocating and releasing dedicated devices
5. Providing a device-independent block size

DISK

✓Disks are the information storage media

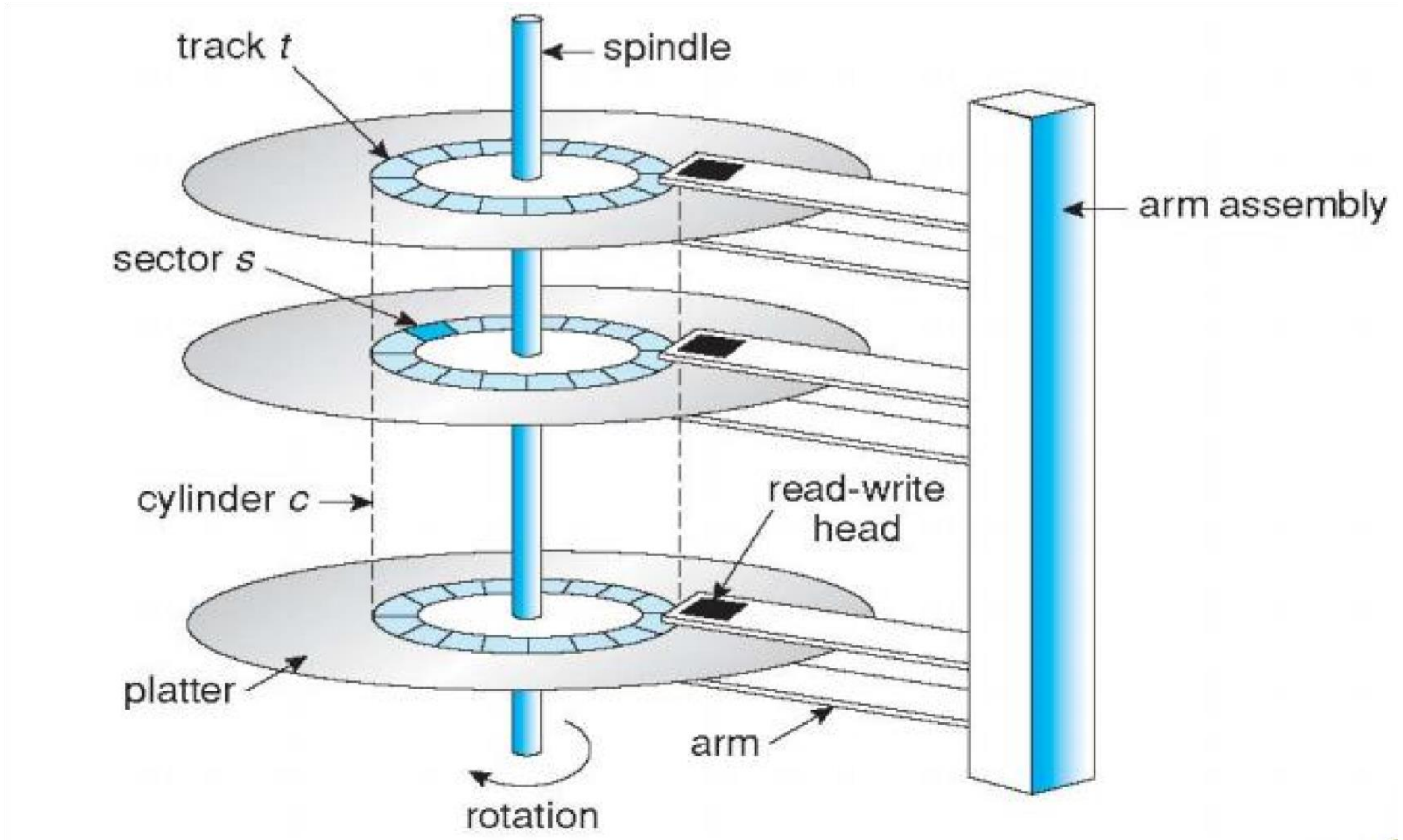
✓Advantages of disks over main memory:

- storage capacity of disk is greater than main memory
- disks are non-volatile (i.e. unlike main memory, the information on disk is not lost even after power off) ✓Disks come in variety of types:
- Magnetic Disks : Hard Disks, Floppy disks - Optical Disks : CD-ROMs, DVDs etc.

DISK HARDWARE

- ✓A disk consists of several platters
- ✓Each disk platter has a flat circular shape, like a CD
- ✓The two surfaces of a platter are covered with a magnetic material
- ✓A read-write head "flies" just above each surface of every platter
- ✓The heads are attached to a **disk arm** that moves all the heads as a unit
- ✓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**
- ✓There may be thousands of concentric cylinders in a disk drive, and each track may contain hundreds of sectors

DISK HARDWARE DISK I/O



- ✓**Seek** : Position head over desired cylinder/track. Seek time depends on how fast the hardware moves the arm.
- ✓**Rotational Delay** : time for the sector to rotate underneath the head. Rotational delay depends on how fast the disk spins
- ✓**Transfer time** : time to move the bytes from disk to memory

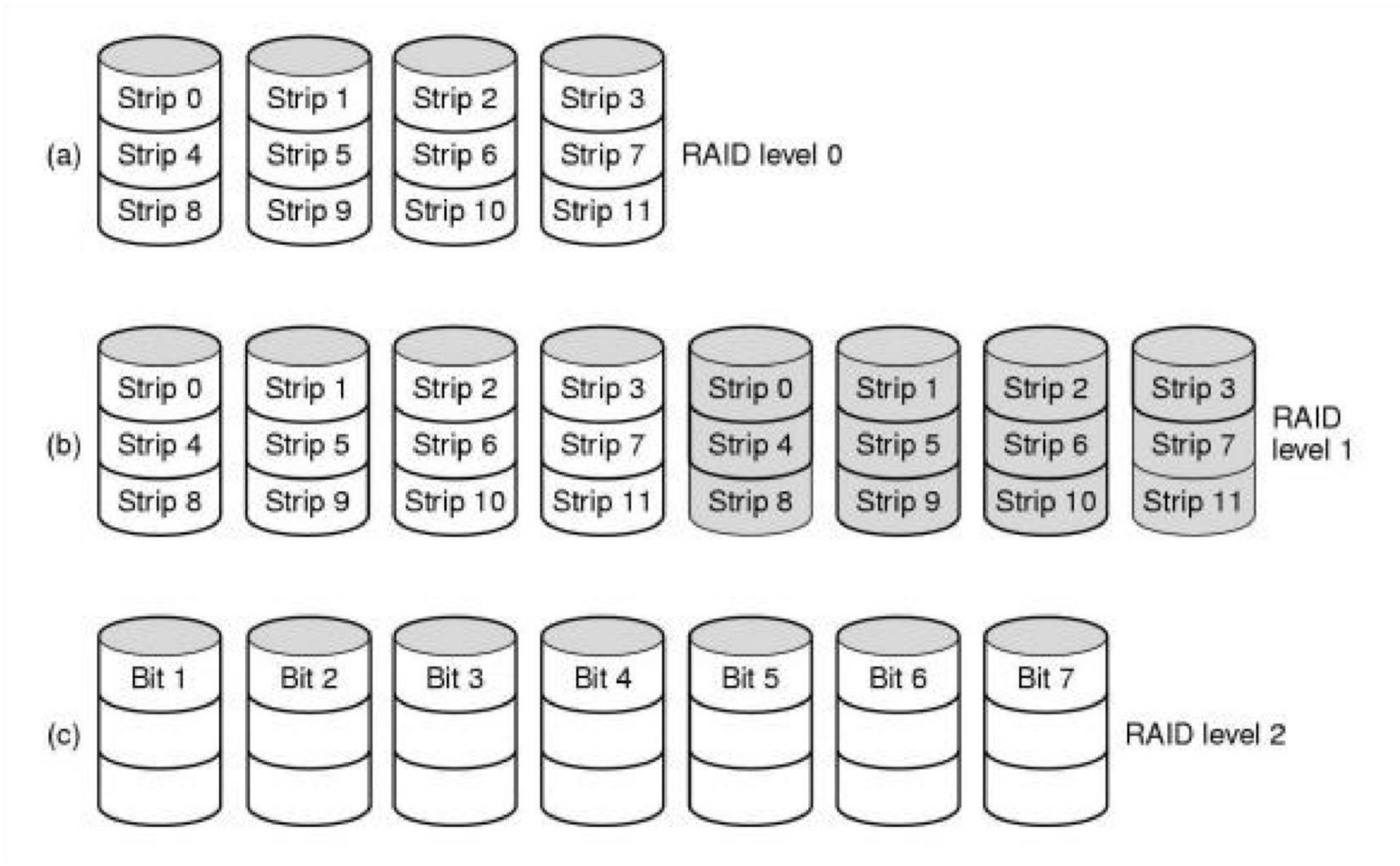
- ✓**Disk I/O time = seek time + rotational delay + transfer time**

RAID LEVELS

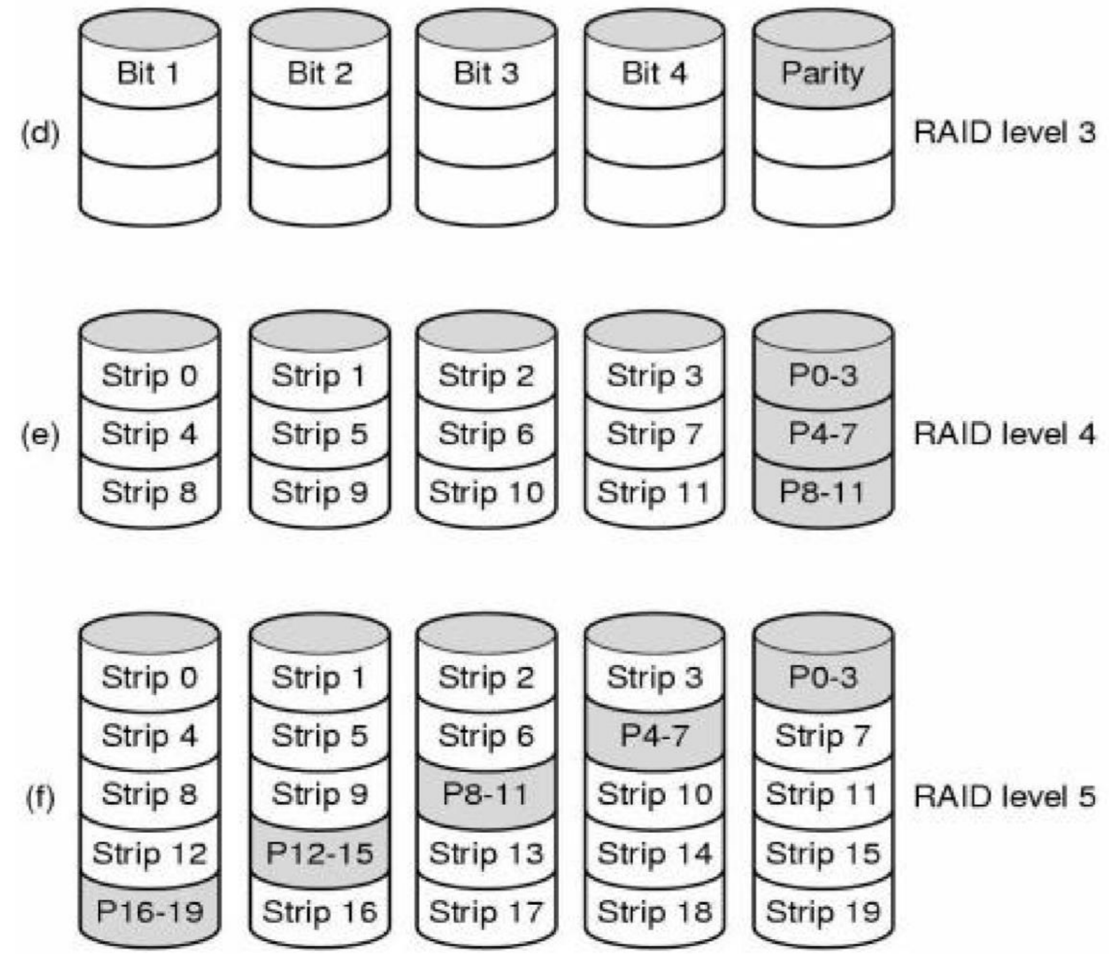
RAID

- ✓RAID: Redundant Array of Independent Disks
- ✓Collection of inexpensive disks which appears as a single disk to the software
- ✓In addition to appearing like a single disk to the software, all RAIDs have the property that the data are distributed over the drives, to allow parallel operation
- ✓Different schemes for doing this is called RAID levels. In other words, RAID levels refers to the organization of data in the disks in the RAID.
- ✓We have RAID level 0 through 5 with some few other minor levels

RAID LEVELS



(a)
Strip 0
Strip 1
Strip 2
Strip 3
Strip 4
Strip 5
Strip 6
Strip 7
Strip 8
Strip 9
Strip 10
Strip 11
RAID level 0
(b)
Strip 0
Strip 1
Strip 2
Strip 3
Strip 4
Strip 5
Strip 6
Strip 7
Strip 8
Strip 9
Strip 10
Strip 11
Strip 0
Strip 1
Strip 2
Strip 3
Strip 4
Strip 5
Strip 6
Strip 7
Strip 8
Strip 9
Strip 10
Strip 11
RAID level 1
(c)
Bit 1
Bit 2
Bit 3
Bit 4
Bit 5
Bit 6
Bit 7
RAID level 2


Bit 1
Bit 2
Bit 3
Bit 4
Parity
(d)
RAID level 3
Strip 0
Strip 1
Strip 2
Strip 3
P0-3
Strip 4
Strip 5
Strip 6
Strip 7
P4-7
Strip 8
Strip 9
Strip 10
Strip 11
P8-11
(e)
RAID level 4
Strip 0
Strip 1
Strip 2
Strip 3
P0-3
Strip 4
Strip 5
Strip 6
P4-7
Strip 7
Strip 8
Strip 9
P8-11
Strip 10
Strip 11
Strip 12
P12-15
Strip 13
Strip 14
Strip 15
P16-19
Strip 16
Strip 17
Strip 18
Strip 19
(f)
RAID level 5

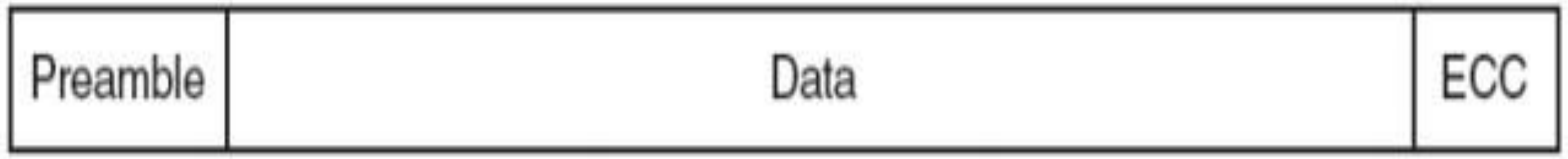
DISK FORMATTING

✓Disk formatting is the process of preparing a data storage device such as a hard disk drive, floppy disk or USB flash drive for initial use.

✓After manufacture, there is no information whatsoever in the disk

✓Each platter receives a **low-level format** done by the software

✓The format consists of a series of concentric tracks, each containing some number of sectors, with short gaps between the sectors ✓Format of the sector is:



DISK FORMATTING

- ✓Preamble starts with a certain bit pattern the allows hardware to recognize the start of the sector. It also contains the cylinder, sector number and some other information
- ✓The size of data portion depends on formatting, most disk use 512 byte sectors
- ✓ECC contain redundant information that can be used to recover from the read errors
- ✓When the controller writes a sector of data during normal I/O, the ECC is updated with a value calculated from all the bytes in the data area.
- ✓When the sector is read, the ECC is recalculated and is compared with the stored value. If the stored and calculated numbers are different, this mismatch indicates that the data area of the sector has become corrupted and that the disk sector may be bad.
- ✓ The controller automatically does the ECC processing whenever a sector is read or written.

DISK FORMATTING

- ✓After low level formatting, comes the mid level formatting called partitioning which involves creating **master boot record.**
- ✓The third part of the process, usually termed "high-level formatting" is done to all the partition, most often refers to laying down a boot block, free storage administration (bitmap or linked list), root directory and an empty file system.

Cylinder skew

- ✓Position of sector 0 on each track is offset from the previous track
- ✓This offset is cylinder **skew**, which is done to improve the performance

Q. How does skew improve the performance ?

DISK FORMATTING

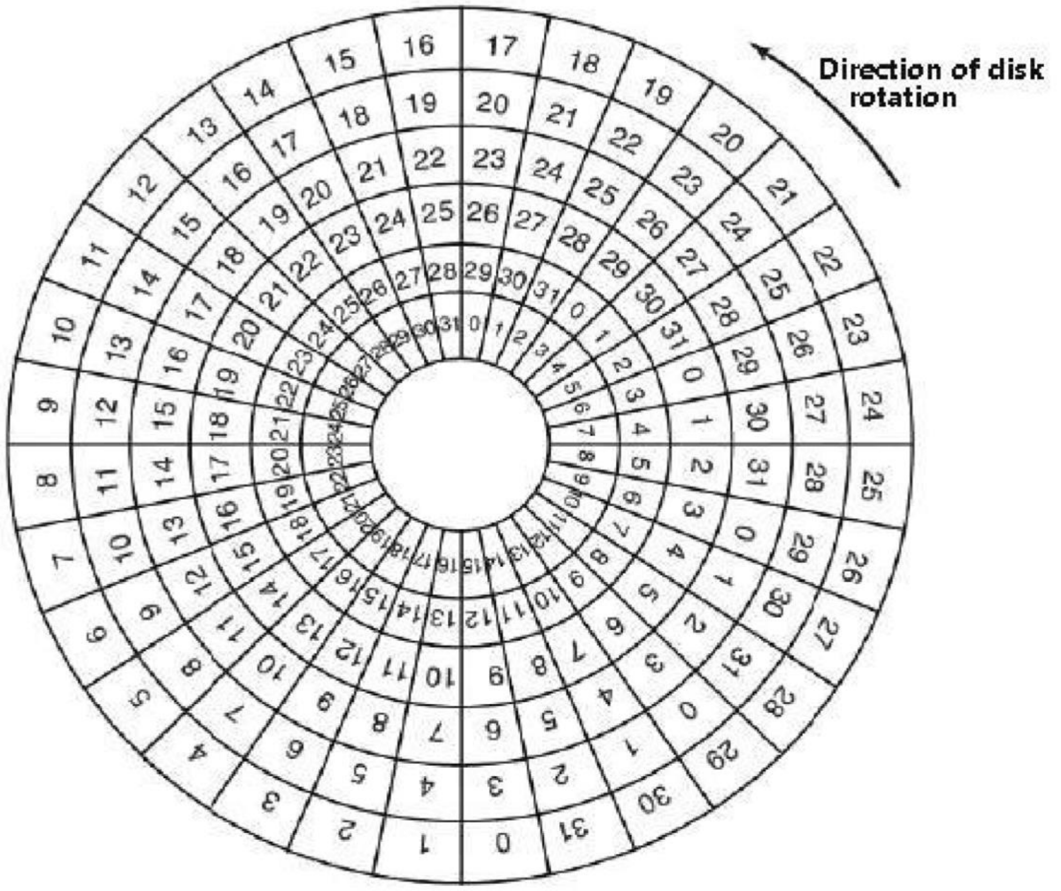


Figure 2. An illustration of cylinder skew.

DISK FORMATTING

Interleaving

✓Consider, a controller with a one-sector buffer that has been given a command to *read two consecutive sectors*.

✓After reading the first sector from the disk and doing the ECC calculation, the data must be transferred to main memory.

✓While this transfer is taking place, the next sector will fly by the head.

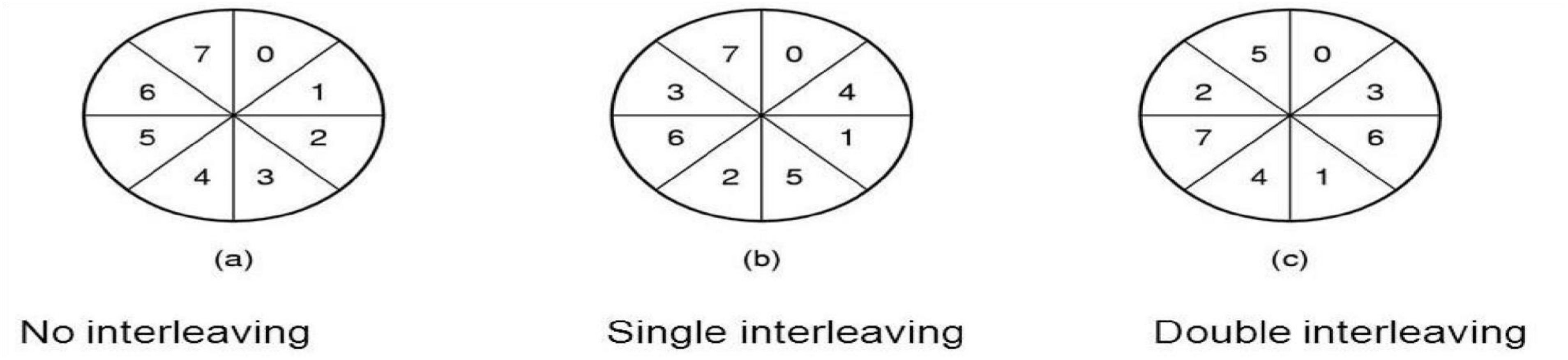
✓When the copy to memory is complete, the controller will have to wait almost an entire rotation time for the second sector to come around again.

✓**This problem can be eliminated by a technique called interleaving**

DISK FORMATTING

✓The idea is to number the sectors in interleaved fashion rather than sequentially

✓Interleaving gives the controller some breathing time between consecutive sectors in order to copy the buffer to main memory✓We may also implement double interleaving if the copying is really slow



DISK ARM SCHEDULING

✓We know that

Disk I/O time = seek time + rotational delay + transfer time

Q. How can you make disk access faster??

✓For most disk, seek time dominates the other two times.

✓So, reducing the mean seek time can improve system performance time substantially.

✓*The idea is to permute the order of the disk requests form the order that they arrive from the users to an order that reduces the length and number of seeks*

✓We will look at several disk arm scheduling algorithms for doing so

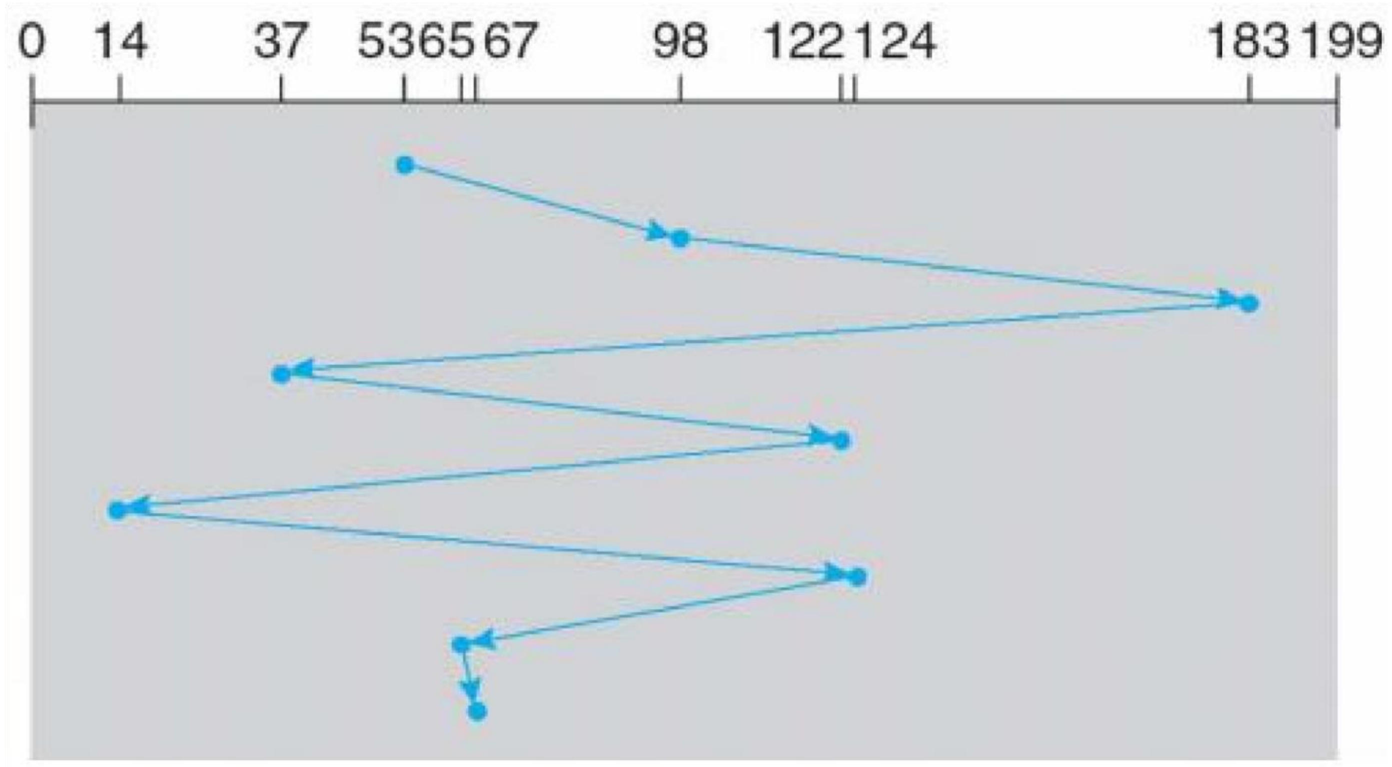
FIRST COME FIRST SERVED (FCFS)

- Process requests as they come -
Fair (no starvation)

Q. Suppose that a disk drive has 200 cylinders, numbered from 0 to 199. The drive is currently serving a request at cylinder 53. The queue of the pending requests is 98, 183, 37, 122, 14, 124, 65, 67. Starting from the current head position, what is the total distance (in cylinders) that the disk arm moves to satisfy all the pending requests if the controller is using FCFS scheduling algorithm??

queue = 98, 183, 37, 122, 14, 124, 65, 67
head starts at 53

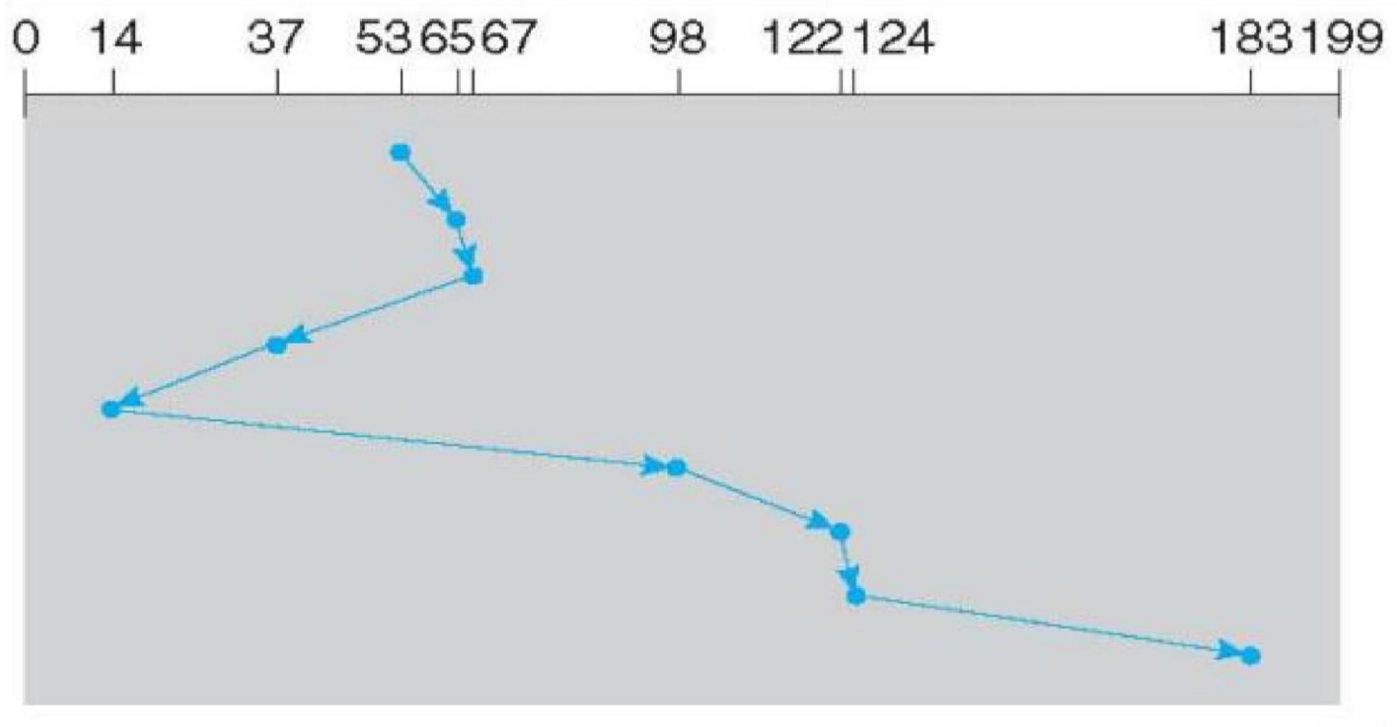
FIRST COME FIRST SERVED (FCFS)



Total Distance travelled = 45+85+146+85+108+110+59+2 = 640 cylinders
Average distance travelled = 640/8 = 80 cylinders

SHORTEST SEEK TIME FIRST (SSTF)

- always go to the next closest request
- May lead to starvation
- greedy, not always optimal
- Q. in slide 36 but with SSTF algorithm

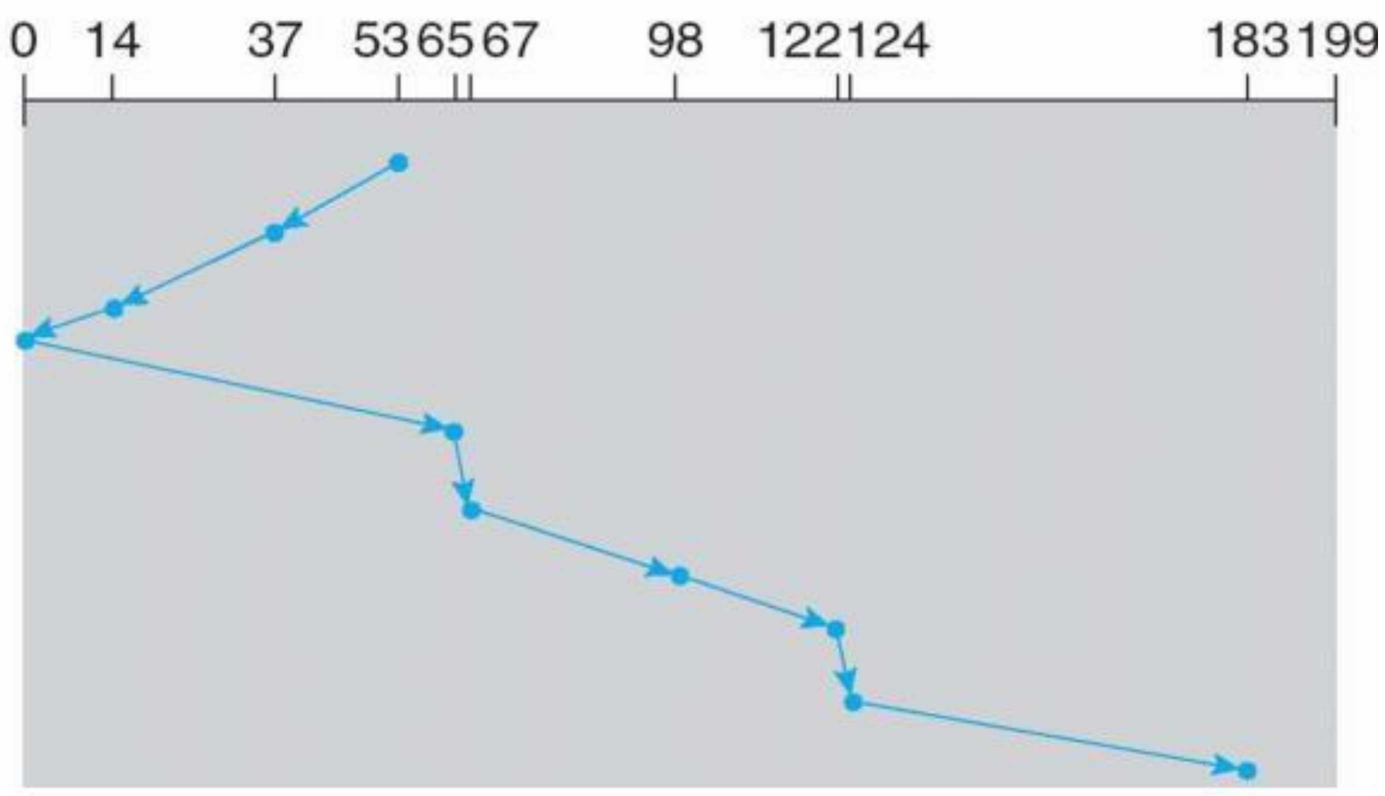


Total Distance travelled = 236 cylinders

ELEVATOR ALGORITHM (SCAN)

- ✓Move head in one direction; Services requests in track order until it reaches the last track, then reverses direction
- ✓Head moves back and forth across the disk , servicing requests as it passes them (0 to N, N to 0, 0 to N……)
- ✓Better than FIFO, usually worse than SSTF
- ✓Avoids starvation
- ✓Suppose that a disk drive has 200 cylinders, numbered from 0 to 199. The drive is currently serving a request at cylinder 53. The queue of the pending requests is 98, 183, 37, 122, 14, 124, 65, 67. The previous request was at cylinder 88. Starting from the current head position, what is the total distance (in cylinders) that the disk arm moves to satisfy all the pending requests if the controller is using SCAN algorithm??

SCAN

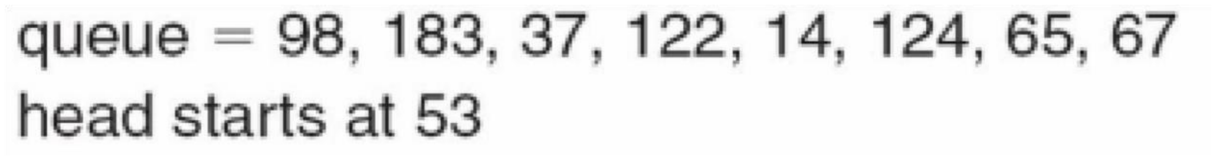


Total Distance travelled = 208 cylinders

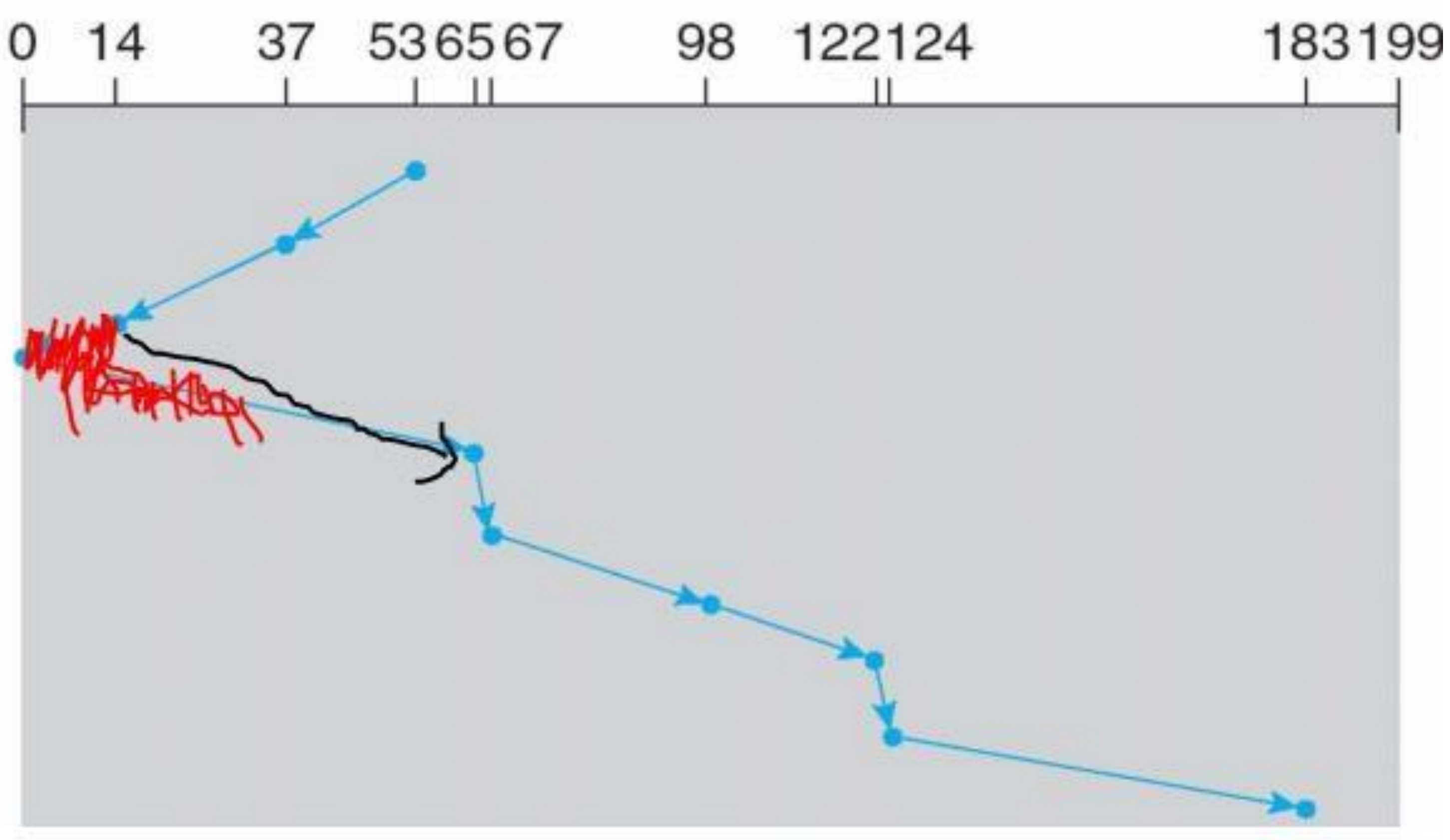
LOOK ALGORITHM

✓Similar to SCAN but does not go all the way to the edge of the disk each time, but just as far as the last request

Q.


queue = 98, 183, 37, 122, 14, 124, 65, 67
head starts at 53

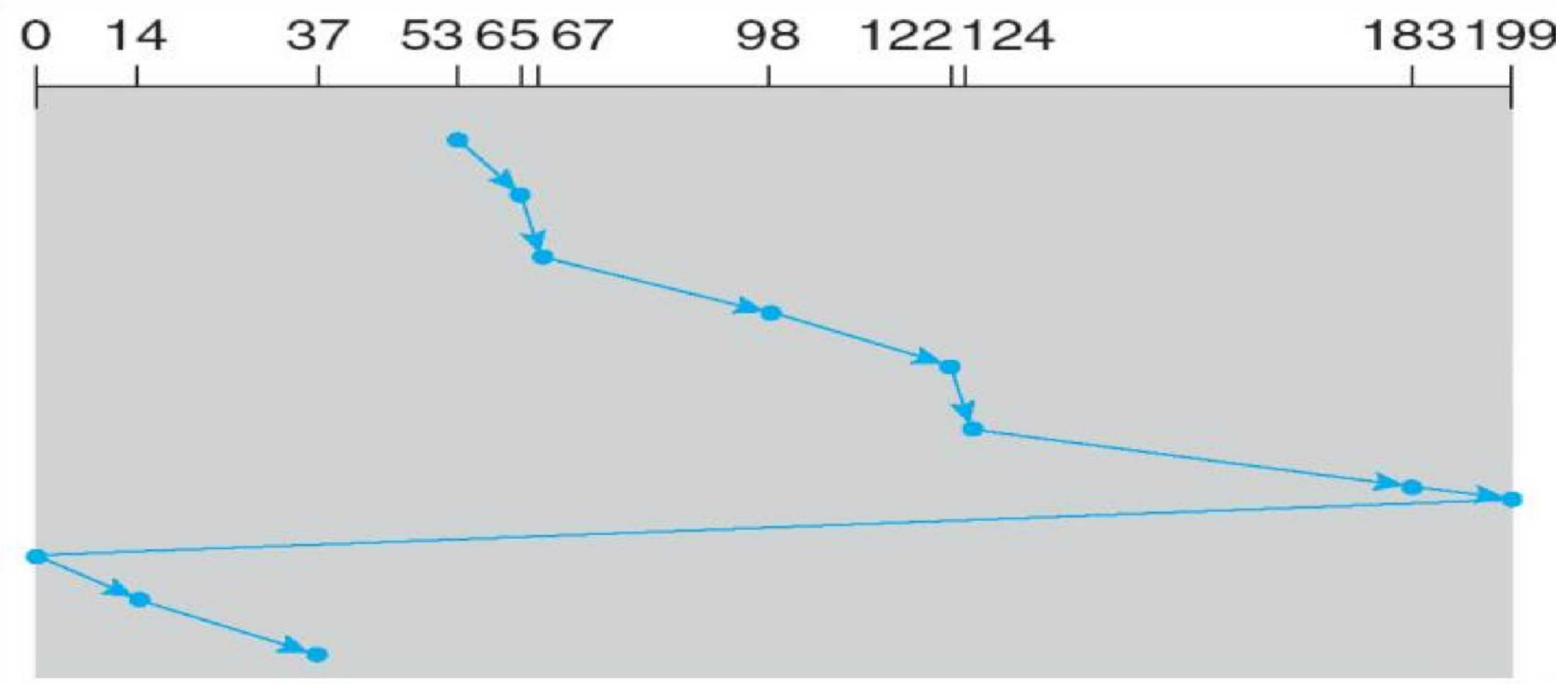
LOOK ALGORITHM



C-SCAN (CIRCULAR SCAN)/MODIFIED ELEVATOR

- ✓The head moves from one end of the disk to the other, servicing requests as it goes
- ✓When it reaches the other end, however, it immediately returns to the beginning of the disk, without servicing any requests on the return trip
- ✓Treats the cylinders as a circular list that wraps around from the last cylinder to the first one
- ✓Provides a more uniform wait time than SCAN

C-SCAN (CIRCULAR SCAN)/MODIFIED ELEVATOR

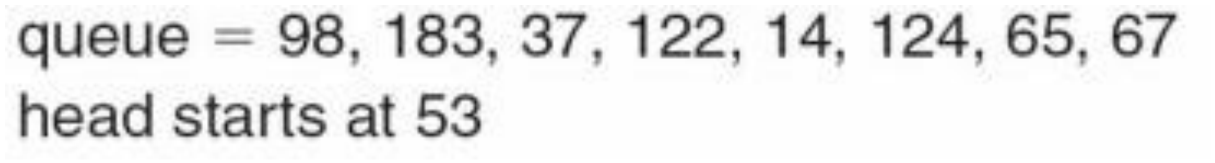


C-LOOK

✓LOOK a version of SCAN, C-LOOK a version of C-SCAN

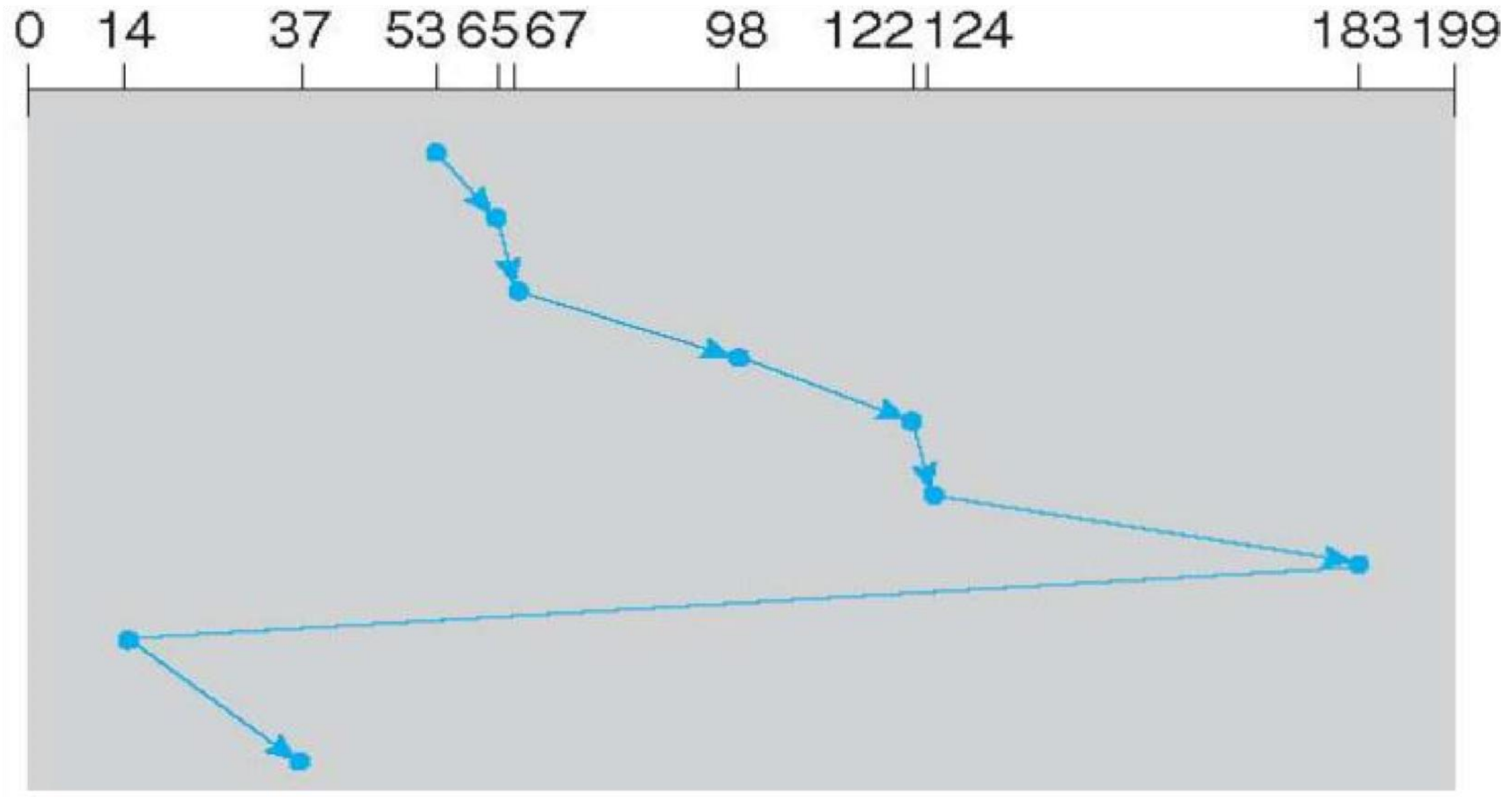
✓Arm only goes as far as the last request in each direction, then reverses direction

immediately, without first going all the way to the end of the disk Q.



queue = 98, 183, 37, 122, 14, 124, 65, 67
head starts at 53

C-LOOK



SELECTING A DISK SCHEDULING ALGORITHM

- ✓SSTF common, natural appeal
- ✓SCAN and C-SCAN perform better if heavy load on disk
- ✓Performance depends on number and types of requests
- ✓Requests for disk service influenced by file-allocation method
- ✓Disk-scheduling should be separate module of OS, allowing replacement with different algorithm if necessary

QUESTION !!

Suppose that the head of a moving head disk has 200 tracks numbers 0-199, is currently serving the request at track 143 and has just finished a request at track 125. The queue of requests is kept in FIFO order 86, 147, 91, 177, 94, 150, 102, 175, 130.

What is the total number of head movements needed to satisfy these requests for the following disk scheduling algorithm?

FCFS, SSTF, SCAN, LOOK, C-SCAN.

FIFO (FCFS) scheduling algorithm

Here the head is move in the order 143, 86, 147, 91, 177, 94, 150, 102, 175, 130

= |86-143|+|147-86|+|91-147|+|177-91|+|94-177|+|150-94|+|102-150|+|175-102|+|130-175|

= 565 cylinders

Average head movements =565/9=**62.77 cylinders**

SSTF scheduling algorithm

The head will move in the following order

143-147-150-130-102-94-91-86-175-177

Total number of head movement is:

= |147-143|+|150-147|+|130-150|+|102-130|+|94-102|+|91-94|+|86-91|+|175-86|+|175-177|

= 4+3+20+28+8+3+5+89+2

=162 cylinder

Average head movements=162/9=18 cylinders

SCAN Scheduling Algorithm

The head is move in order

143, 147, 150, 175, 177, 199, 130, 102, 94, 91, 86

Total number of head movements are:

= |147-147 |+ |150-147| + |175-150| + |177-175| + |199-177| + |130-199| + |102-130| + |94-102| + |91-94| + |86-91|

=169 cylinders

Average head movements =169/9=18.77 cylinders

LOOK Scheduling Algorithm

The head is move in order

143, 147, 150, 175, 177, 130, 102, 94, 91, 86

Total number of head movements is:

= |147-147 |+ |150-147|+|175-150|+|177-175|+|130-177| + |102-130| +|94-102| + |91-94| + |86-91|

=125 cylinders

Average head movements =125/9=13.88 cylinders

C-SCAN Scheduling Algorithm

The head is move in order

143, 147, 150, 175, 177, 199, 0, 86, 91, 94, 102, 130

Total number of head movements are:

= |147-147 |+ |150-147| + |175-150| + |177-175| + |199-0| + |0-86| + |91-86| + |94-91| + |102-94| + |130-102|

=385 cylinders

Average head movements =385/9=18.77 cylinders

ERROR HANDLING

- ✓Manufacturing defects introduce *bad sectors*, the sectors that don’t correctly read back the value just written to them
- ✓Two general approaches to deal with bad sectors:
  1. Deal with them in the controller
  2. Deal with them in the OS
- ✓In the first approach, before the disk is shipped from the factory, it is tested and the list of bad sectors is written onto disk.
- ✓For each bad sector, one of the spares is substituted for it
- ✓There are two ways to do this substitution

ERROR HANDLING

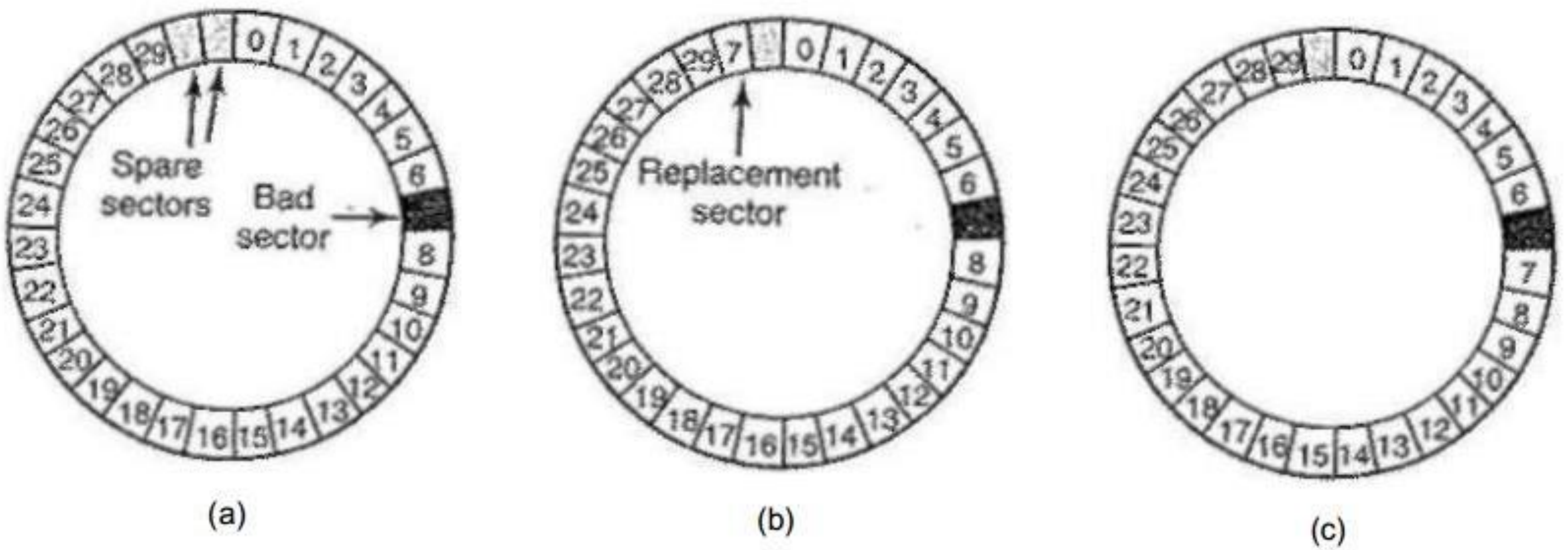


Figure 5-30. (a) A disk track with a bad sector, (b) Substituting a spare for the bad sector, (c) Shifting all the sectors to bypass the bad one.

ERROR HANDLING

✓Handling bad sectors by OS also involves remapping the sectors in software

✓Besides bad sectors, we also have to deal with **seek error**

✓Seek problems are caused by mechanical problems in the arm

✓The controller keeps track of the arm position internally

✓To perform a seek, it issues a series of pulses to the arm motor, one pulse per cylinder, to move the arm to the new cylinder.

✓When the arm gets to its destination, the controller reads the actual cylinder number from the preamble of the next sector.

✓If the arm is in the wrong place, a seek error has occurred.

- ✓Most hard disk controllers correct seek errors automatically, but most floppy controllers just set an error bit and leave the rest to the driver.

STABLE STORAGE

✓Disks sometimes make errors. Good sectors can suddenly become bad sectors. Whole drives can die unexpectedly.

✓RAIDs protect against a few sectors going bad or even a drive falling out. However, they do not protect against write errors laying down bad data in the first place.

✓They also do not protect against crashes during writes corrupting the original data without replacing them by newer data.

✓What is achievable is a disk subsystem that has the following property: when a write is issued to it, the disk either correctly writes the data or it does nothing, leaving the existing data intact. Such a system is called **stable storage** and is implemented in software

✓The goal is to keep the **disk consistent** at all costs

STABLE STORAGE

Assumptions:

1. The model assumes that when a disk writes a block (one or more sectors), either the write is correct or it is incorrect and this error can be detected on a subsequent read by examining the values of the ECC fields.

2. The model also assumes that a correctly written sector can spontaneously go bad and become unreadable.

3. The model also assumes the CPU can fail, in which case it just stops.

- ✓ Stable storage uses a pair of identical disks with the corresponding blocks working together to form one error-free block.
- ✓ In the absence of errors, the corresponding blocks on both drives are the same.
- ✓ Either one can be read to get the same result.

STABLE STORAGE

Operations:

1. **Stable writes :** A stable write consists of first writing the block on drive 1, then reading it back to verify that it was written correctly. If it was not written correctly, the write and reread are done again up to n times until they work. After n consecutive failures, the block is remapped onto a spare and the operation repeated until it succeeds, no matter how many spares have to be tried. After the write to drive1 has succeeded, the corresponding block on drive 2 is written and reread, repeatedly if need be, until it, too, finally succeeds.

2. **Stable Read:** A stable read first reads the block from drive 1. If this yields an incorrect ECC, the read is tried again, up to *n* times. If all of these give bad ECCs, the corresponding block is read from drive 2. Given the fact that a successful stable write leaves two good copies of the block behind, and our assumption that the probability of the same block spontaneously going bad on both drives in a reasonable time interval is negligible, a stable read always succeeds.

STABLE STORAGE

3 . Crash Recovery

✓After a crash, a recovery program scans both disks comparing corresponding blocks. If a pair of blocks are both good and the same, nothing is done.

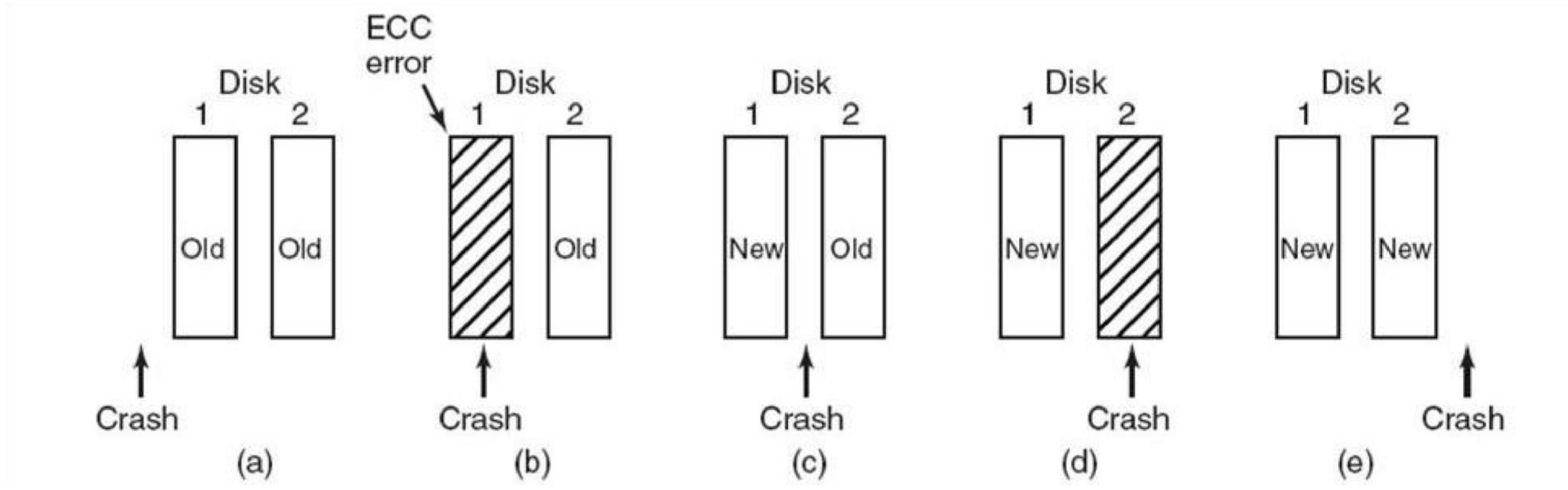
✓If one of them has an ECC error, the bad block is overwritten with the corresponding good block.

✓If a pair of blocks are both good but different, the block from drive 1 is written onto drive 2

STABLE STORAGE

Q. what about CPU crashes during stable write ?

A. It depends on precisely when the crash occurs. There are five possibilities



STABLE STORAGE

- ✓In Fig.(a), the CPU crash happens before either copy of the block is written. During recovery, neither will be changed and the old value will continue to exist, which is allowed.
- ✓In Fig.(b), the CPU crashes during the write to drive 1, destroying the contents of the block. However the recovery program detects this error and restores the block on drive 1 from drive 2. Thus the effect of the crash is wiped out and the old state is fully restored.
- ✓In Fig.(c), the CPU crash happens after drive 1 is written but before drive 2 is written. The point of no return has been passed here: the recovery program copies the block from drive 1 to drive 2. The write succeeds.
- ✓Fig.(d) is like Fig.(b): during recovery, the good block overwrites the bad block. Again, the final value of both blocks is the new one.
- ✓Finally, in Fig.(e) the recovery program sees that both blocks are the same, so neither is changed and the write succeeds here too.