UNIT 4

Input Output Management

Introduction

- Humans interact with machines by providing information through IO devices.
- Many on-line services are availed through use of specialized devices like printers, keyboards etc.
- Management of these devices can affect the throughput of a system.

Issues in I/O management

- Communication with an IO device is required at the following levels:-

  - The need for a human to input information and output from a computer.

  - The need for a device to input information and receive output from a computer.

  - The need for computers to communicate over network.

Device speeds

- Character oriented input devices operate with speeds of tens of bytes per second.

- Block oriented devices are much faster than character oriented devices. Also note that they operate over a much wider range.

Devices communicate bit or byte streams with a machine using a data bus and a device controller.

EVENT BASED I/O

- A computer system may have to synchronize with some other process to communicate.
- So, one process may actually wait at the point of meet place for the other process to arrive for communication using synchronization signals.
- The process advances further following the synchronizing event.

Event Based I/O

- Processes may use signals to communicate events.
- Processes may wait for asynchronous events to occur.
- Every OS provides a set of mechanisms which may be like polling, or a programmed data transfer, or an interrupt mechanism, or even use DMA with cycle stealing.

IO ORGANIZATION

- Computers employ the four basic modes of IO operation:-

- These are:

  1. Polling
  2. Programmed mode
  3. Interrupt mode
  4. DMA mode

Polling

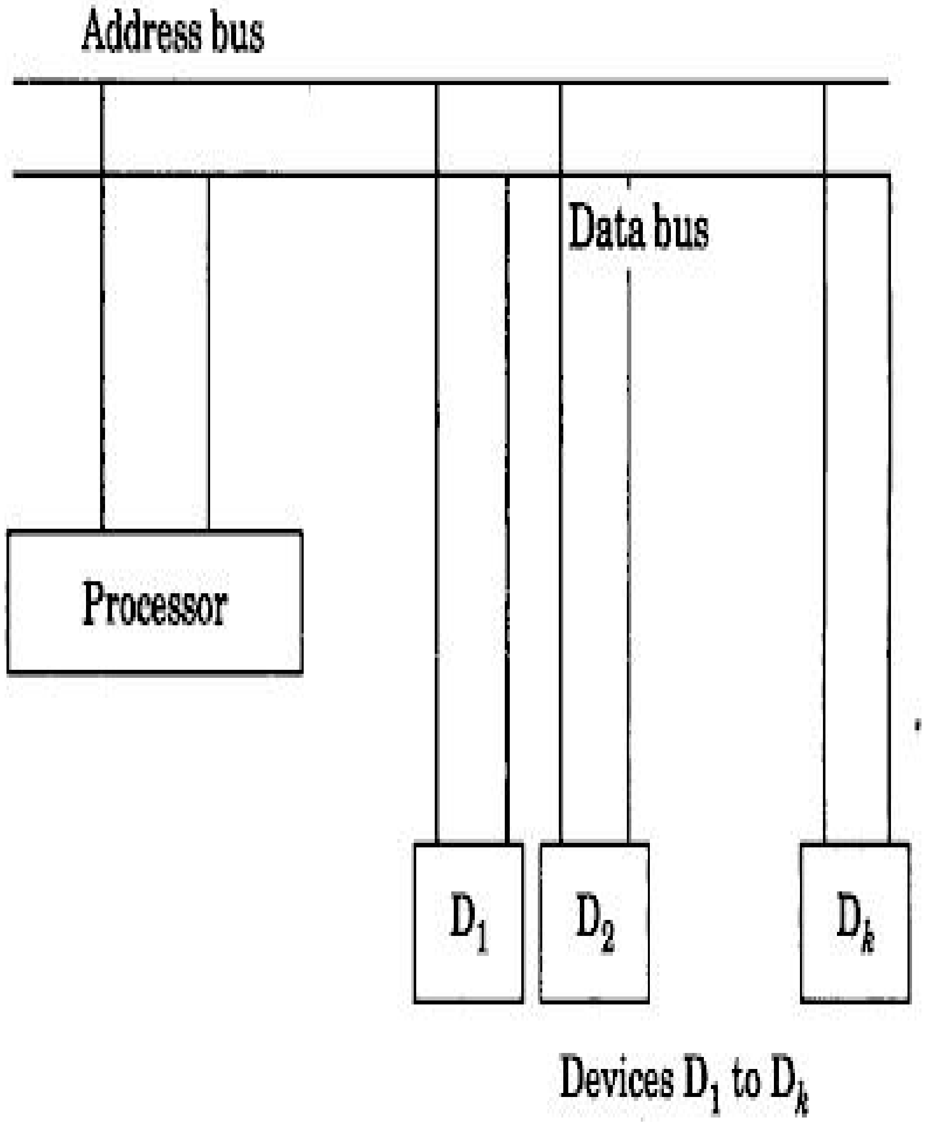
- Polling: Computer interrogates each device in turn to determine if it is ready to communicate.

- Polling as a strategy is also used by systems to interrogate ports on a computer communication network.

POLLING

```
While true
    if device 1 ready to communicate
        then perform IO with device 1

    if device 2 ready to communicate
        then perform IO with device 2
                .
                .
                .
    if device k ready to communicate
        then perform IO with device k
end while
```



POLLING

In hardware, this may typically translate to the following protocol:

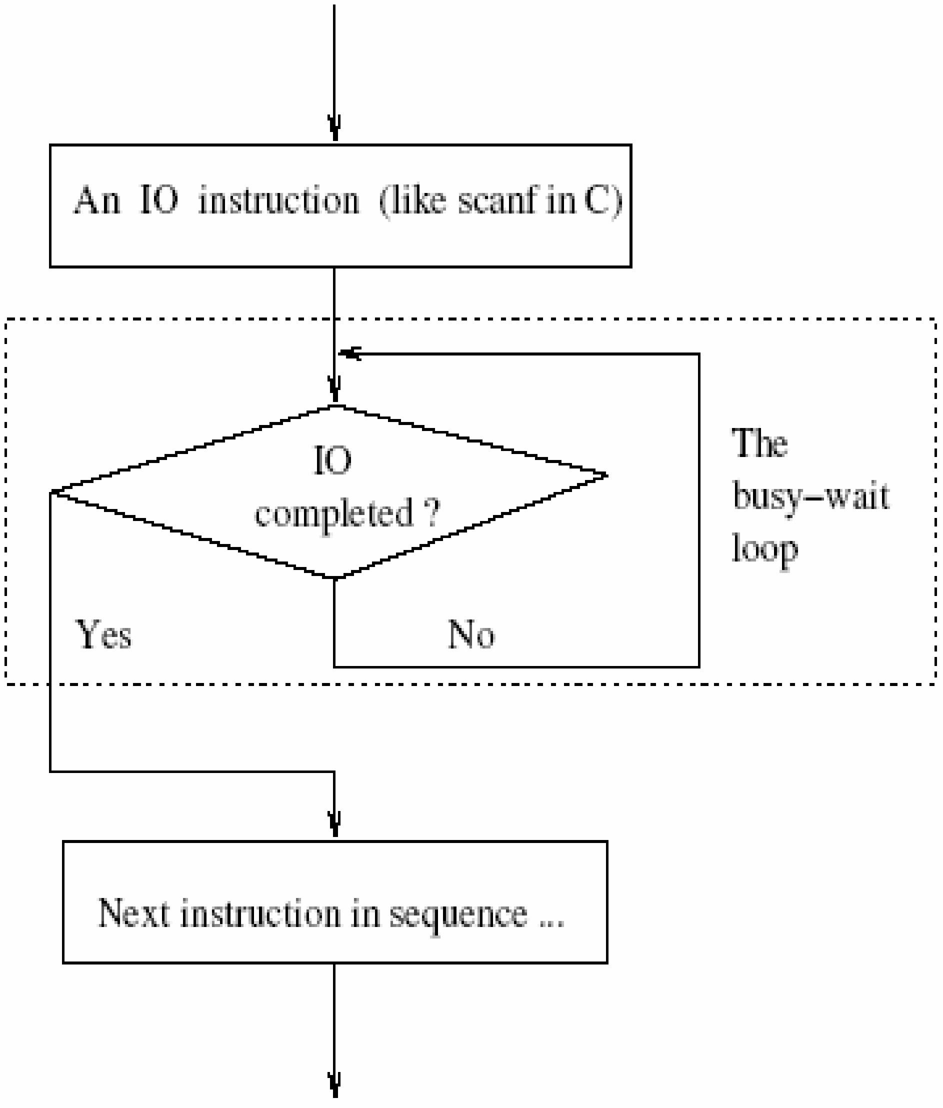
1. Assign a distinct address to each device connected to a bus.
2. The bus controller scans through the addresses in sequence to find which device wishes to establish a communication.
3. Allow the device that is ready to communicate to leave its data on the register.
4. The IO is accomplished. In case of an input the processor picks up the data. In case of an output the device picks up the data.
5. Move to interrogate the next device address in sequence to check if it is ready to communicate.

PROGRAMMED DATA MODE

Programmed mode :

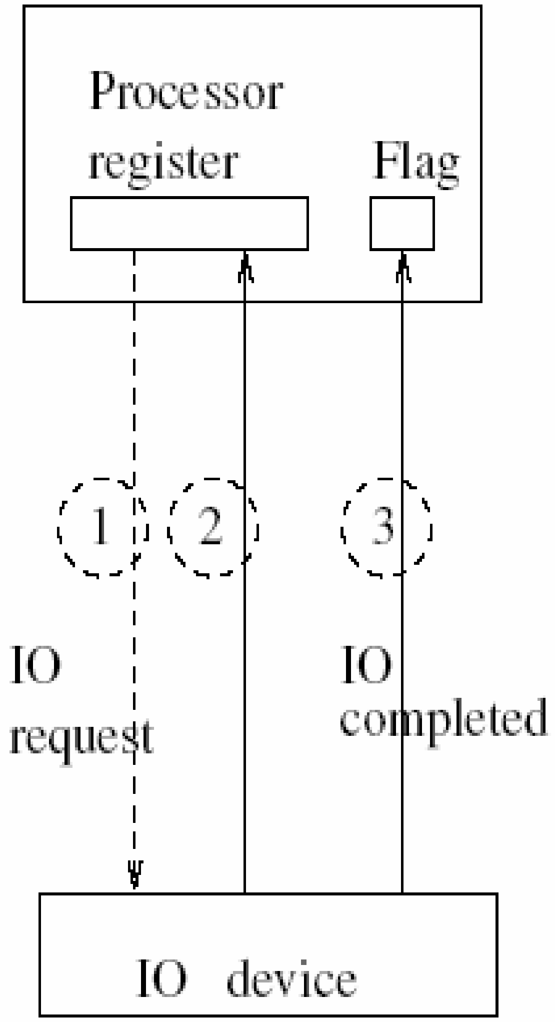
- An IO instruction is issued to a device.
- Now the program busy-waits (idles) till the device IO is completed.
- In other words execution of I/O instruction is in sequence with other program instructions.

PROGRAMMED DATA MODE



An IO instruction (like scanf in C)
IO completed ?
The busy–wait loop
Yes
No
Next instruction in sequence ...

The program description



The hardware support.

Interrupt Mode

- An IO instruction is issued.
- The program advances without suspension till the device is actually ready to establish an IO, when the process is suspended.
- In this mode, an IO instruction is issued and the program advances without suspension.
- Program suspension happens when the device is actually ready to establish an IO.
- An Interrupt Service Routine is executed to achieve device IO.

Interrupt Mode

- Process context is stored to help resume computation later (from the point of suspension).

- Program resumes execution from where it was suspended.

Interrupt Types

- Interrupt processing may require the following contexts :

  <u>Internal Interrupt </u>:

- Source of interrupt is a memory resident process or an event within a processor (due to divide by zero or attempt to execute an illegal instruction).

- Some times malfunctions caused by events

  like divide by zero are called traps.

- Timer interrupts may also occur as in Real Time Operating System.

INTERRUPT TYPES

External Interrupts :

- Source of interrupt is not internal i.e. other than a process or processor related event.
- Can be caused by a device seeking attention of a processor.
- IO device interrupting a program that had sought IO (mentioned earlier) is an external interrupt.

INTERRUPT TYPES

<u>Software Interrupt :</u>

- Most OSs offer two modes of operation – user mode and system mode.
- A system call made by a user program will require a transition from user mode to system mode – this generates a software interrupt.

Interrupt Servicing

Let us see how an interrupt is serviced :-

- Suppose program P is executing an instruction i when an interrupt is raised.
- Assume also an interrupt service routine ISR to be initiated to service the interrupt.

Interrupt Servicing

- The following steps describe how the interrupt service may happen :

1. Suspend the current program P after executing instruction i.

2. Store address of instruction i+1 in P as return address for P – PADDRi+1 which is the incremented program counter value.

This may be stored (RESTORE) in some specific location or a systems’ data structure or in the code area of ISR itself.

Interrupt Servicing

3. A branch unconditionally to interrupt service instructions in ISR happens.

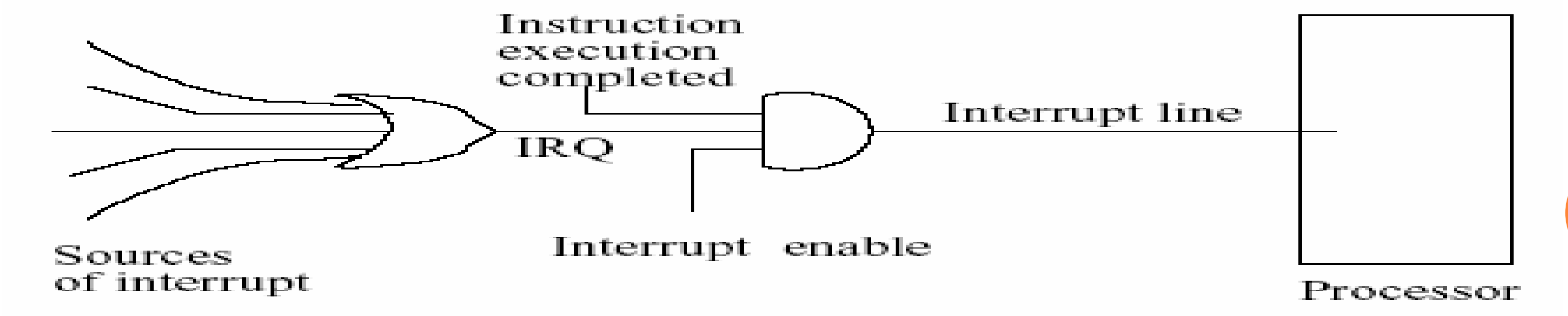
4. Typically, the last instruction in the service routine ISR executes a branch indirect from RESTORE. This restores the program counter a branch indirect instruction from PADDRi+1

- Thus the suspended program P obtains control of the processor again.

Issues in handling interrupts

<u>Detecting an interrupt</u>

- When a processor has an interrupt enable signal up, then at the end of an instruction cycle we shall recognize an interrupt if the interrupt request (IRQ) line is up.
- Once an interrupt is recognized, interrupt service shall be required.
- Two minor points now arise. One is when is interrupt enabled. The other is how a device which raises the interrupt is identified.



ISSUES IN HANDLING INTERRUPTS

When is interrupt enabled

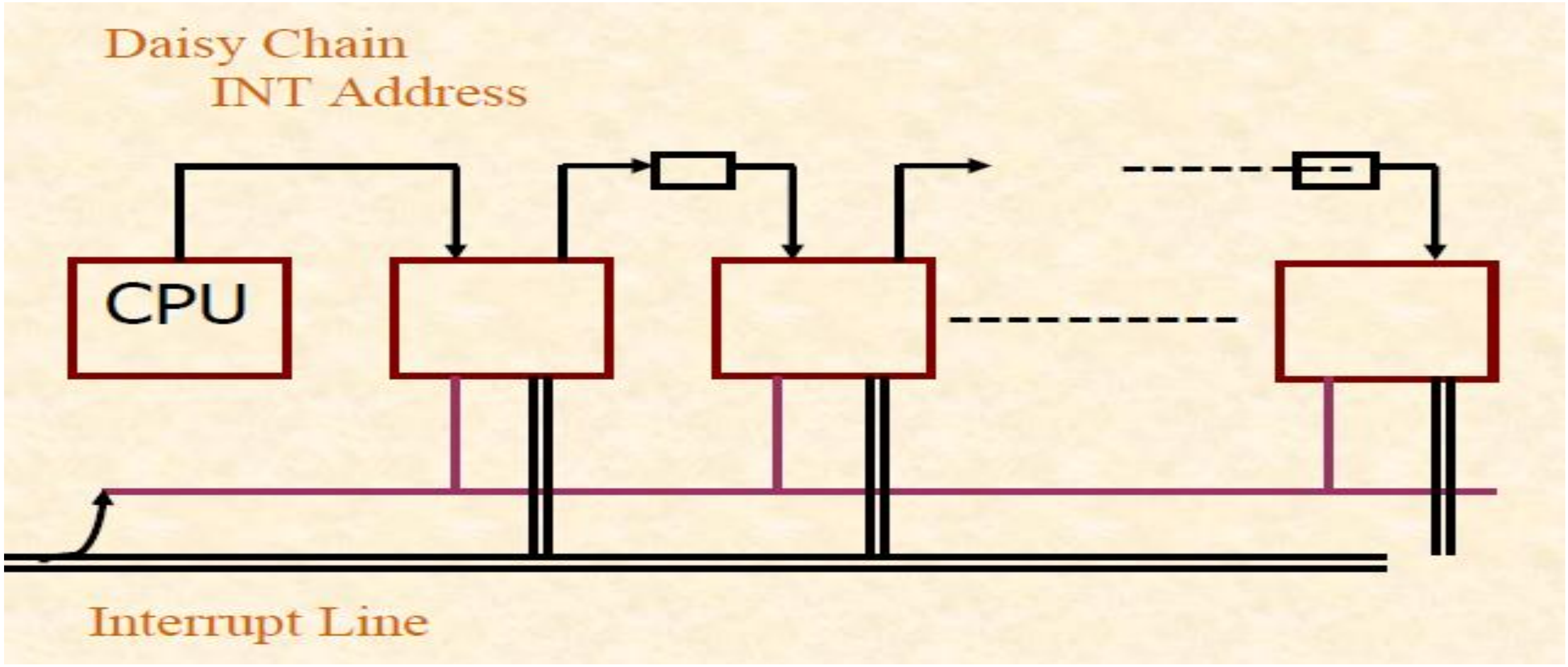
- Sources of interrupt may have assigned priorities.
- If a higher priority device seeks an interrupt, the lower priority device may be denied the per-mission to raise an interrupt.
- Semaphores, cannot be interrupted. Here just processor may disable interrupt.

Identification of Source of Interrupts

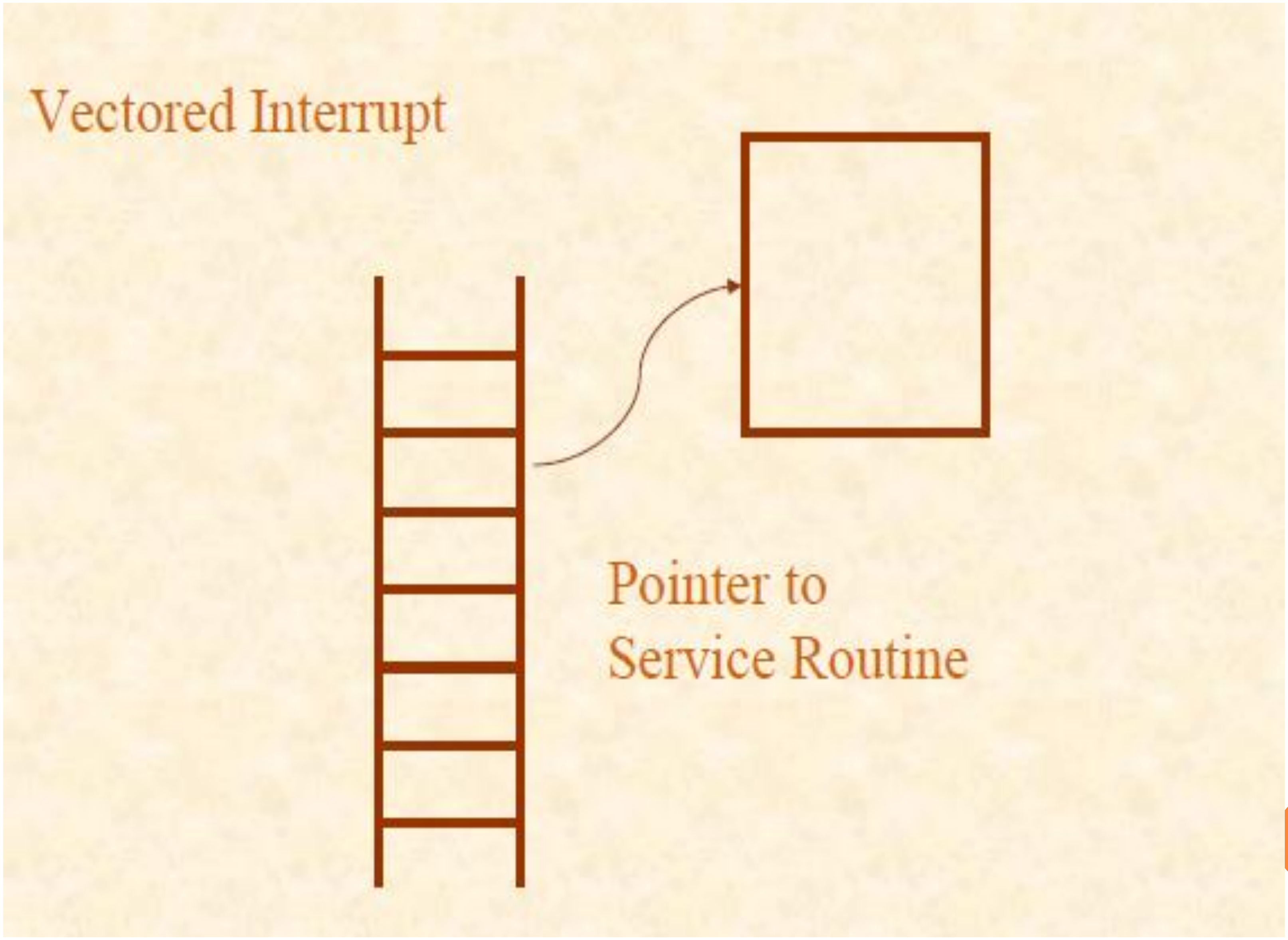
We will see how source of interrupt may be identified in the context of different I/O mechanisms like:

Polling :

- It interrogates all the devices in sequence
- It is slow if there are many devices.



Identification of Source of Interrupts



ISSUES IN HANDLING INTERRUPTS

Interrupt received when a different process is executing.

- Suppose the process Pi initiated an IO. But subsequently it relinquishes the processor to process Pj .
- This may well happen because the process Pi may have finished its time slice or it may have reached a point when it must process a data expected from the device.
- Now let us suppose that priority of process Pi is higher than that of Pj. In that case the process Pj shall be suspended and Pi gets the processor, and the IO is accomplished. Else the process Pj shall continue and the IO waits till the process.
- Pi is able to get the control of the processor. One other way would be to let the device interrupt the process Pj but store the IO information in a temporary buffer (which may be a register) and proceed. The process Pj continues.

Issues in Handling Interrupts

An interrupt during an interrupt service

- Often devices or processes may have priorities. A lower priority process or device cannot cause an interrupt while a higher priority process is being serviced.

- If, however, the process seeking to interrupt is of higher priority then we need to service the interrupt.

Issues in Handling Interrupts

Interrupt Vector (IV)

- Many systems support an interrupt vector (IV).

- An example with four sources of interrupt. These may be a trap, a system call, an IO, or an interrupt initiated by a program. Now we may associate an index value 0 with trap, 1 with system call, 2 with IO device and 3 with the program interrupt.

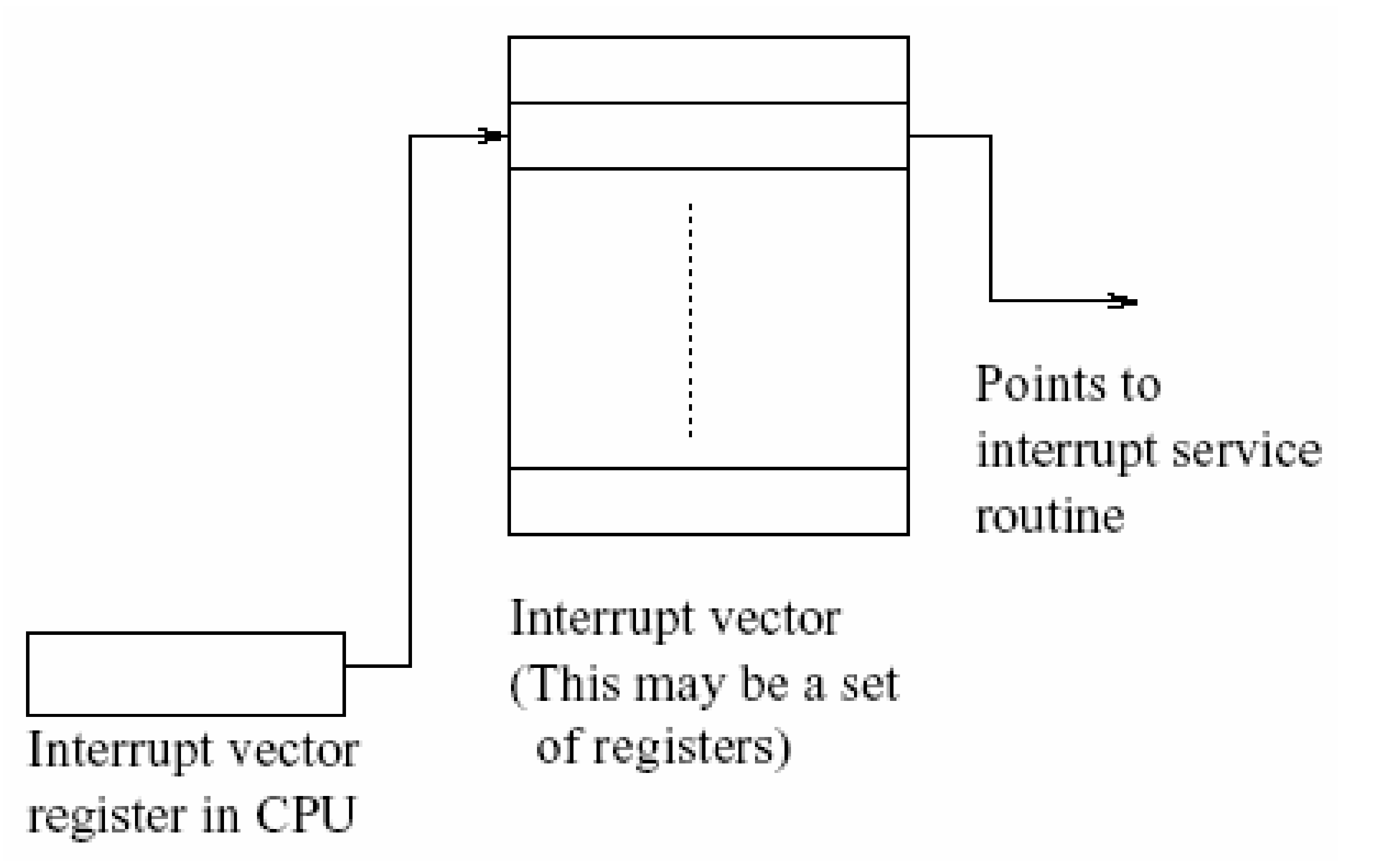
- Note that the source of interrupt provides us the index in the vector. The interrupt service can now be provided as follows:

  - Identify the source of interrupt and generate index i.

  - Identify the interrupt service routine address by looking up IVR(i), where IVR stands for the interrupt vector register. Let this address be ISRi.

  - Transfer control to the interrupt service routine by setting the program counter to
- ISRi.

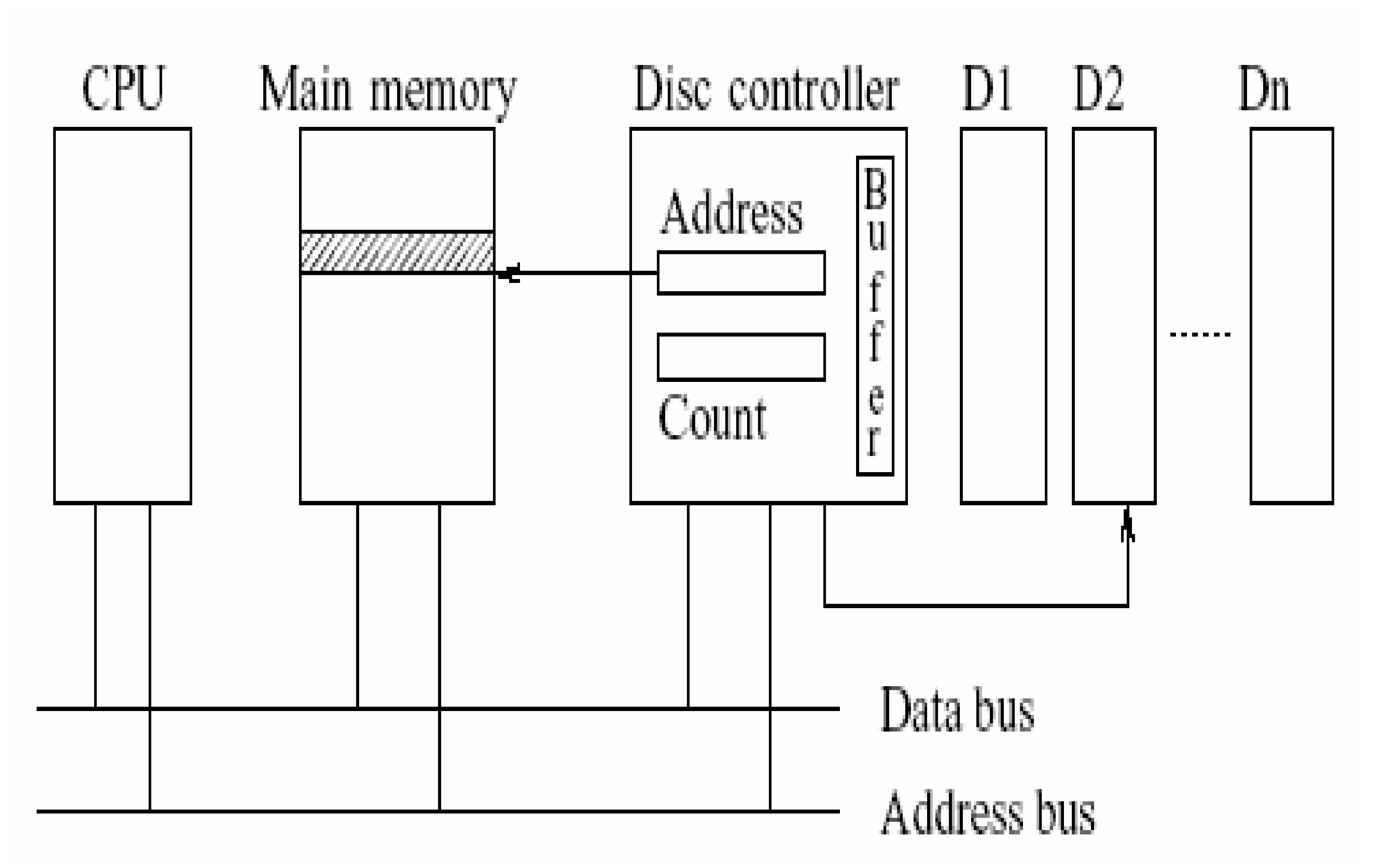
Issues in handling interrupts



DMA MODE OF DATA TRANSFER

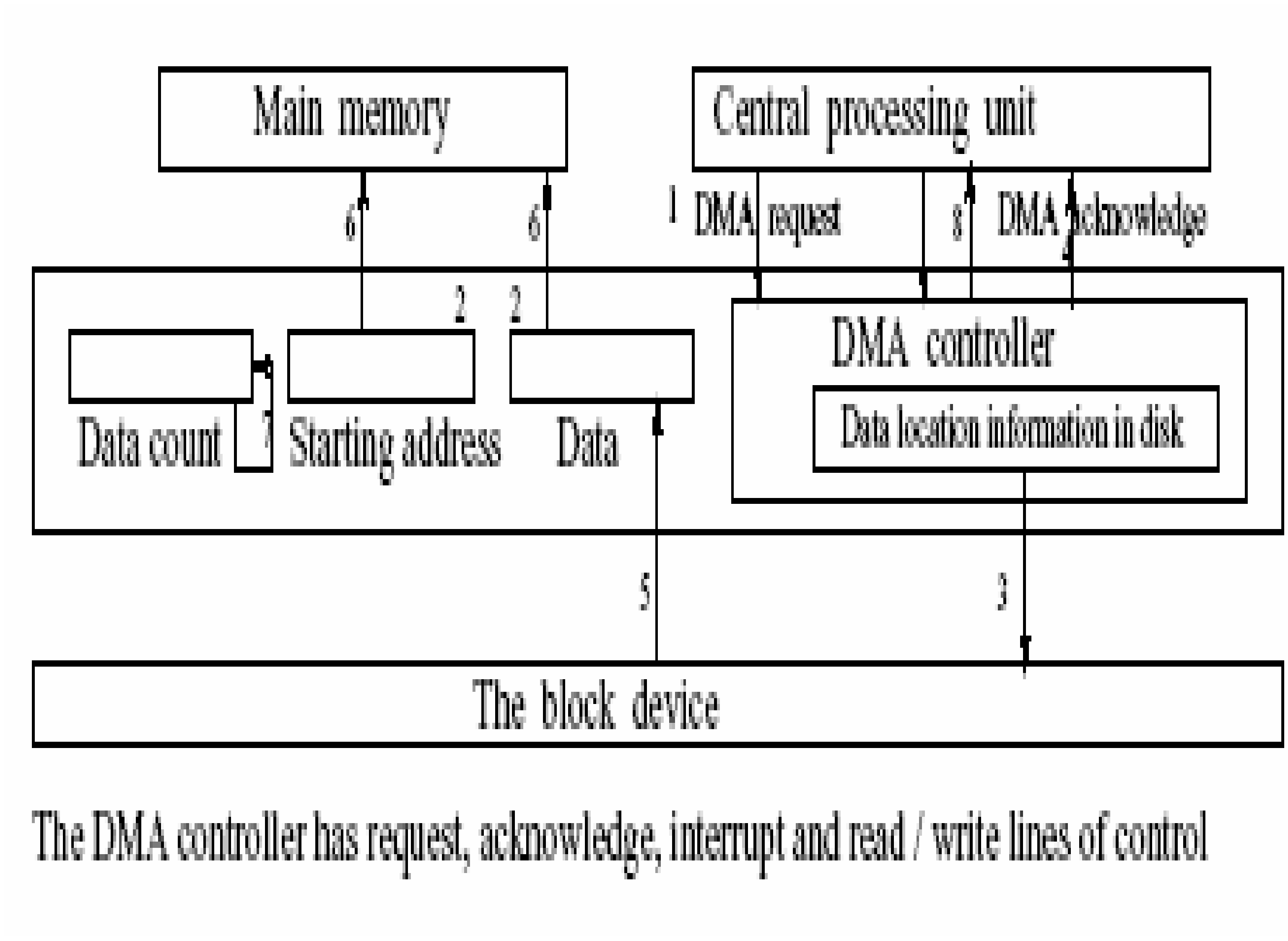
- This is a mode of data transfer in which IO is performed in large data blocks.
- Disks communicate in data blocks of sizes like 512 bytes or 1024 bytes.
- DMA ensures access to main memory without processor intervention or support. Such independence from processor makes this mode of transfer extremely efficient.
- When a process initiates a direct memory access (DMA) transfer, its execution is briefly suspended (using an interrupt) and the starting address and size of data block are stored in a DMA controller.
- Following the DMA set up, the program resumes from the point of suspension.
- The device communicates with main memory stealing memory access cycles in competition with other devices and processor.

DMA MODE OF DATA TRANSFER



DMA : Hardware Support

DMA MODE OF DATA TRANSFER



DMA : Direct memory access mode of data transfer.

IO Modes: some observations

- Programmed IO is used for synchronizing information or when speed is not critical.
- Interrupt transfer is suited for a small amount of critical information – no more than tens of bytes.
- DMA is used mostly for block devices.

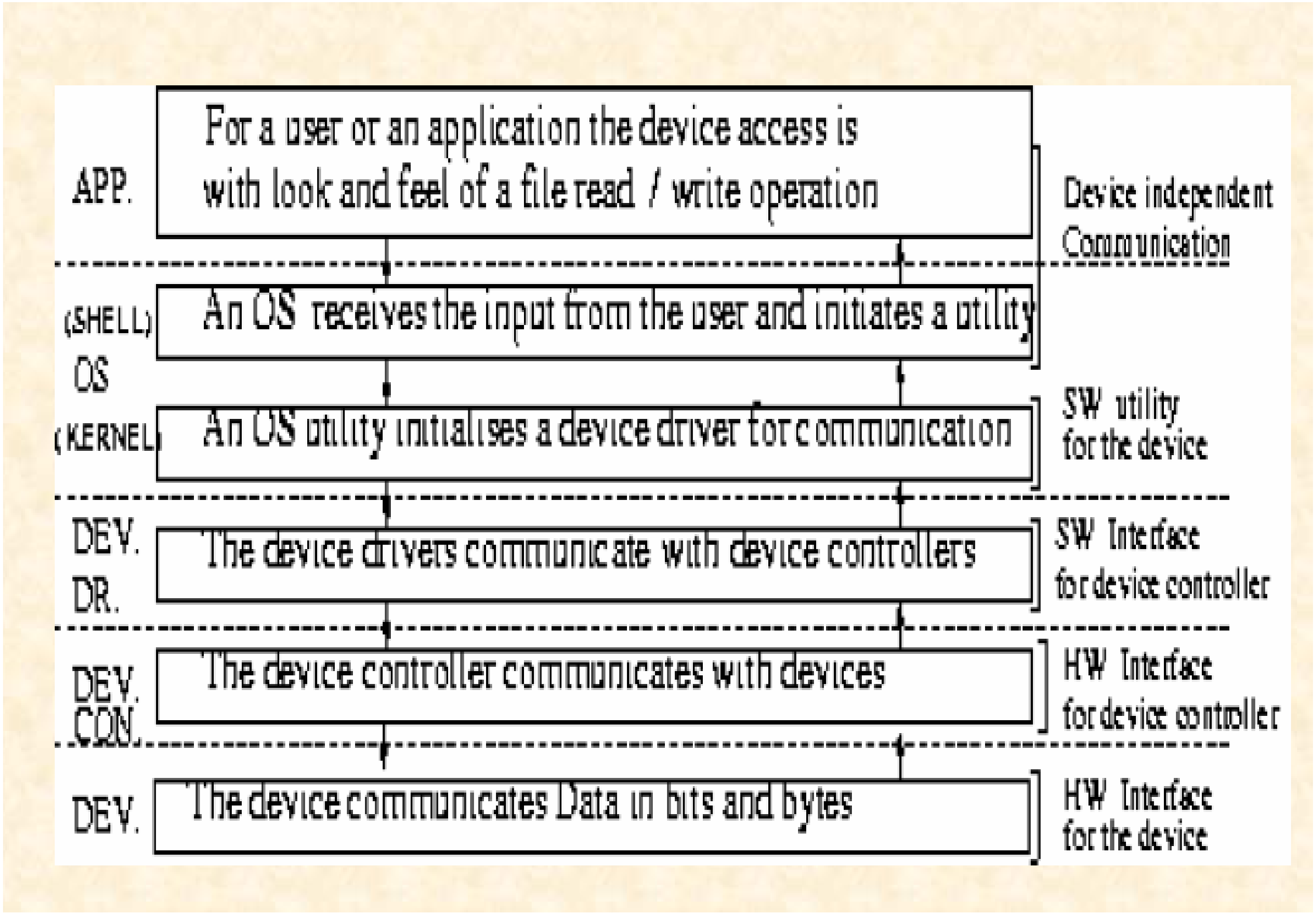
NETWORK ORIENTED TRAFFIC

- Network oriented traffic can be handled in DMA mode.
- Network traffic usually corresponds to getting information from a disc file at both ends and network traffic is bursty.
- In all the above modes, OS makes it look as if we are doing a read or a write operation on a file.

HARDWARE SOFTWARE INTERFACE

- IO management requires that a proper set-up is created by an application on computer system with an IO device.
- An IO operation is a combination of HW and SW instructions.
- Following the issuance of an IO command, OS kernel resolves IO commands.
- Then the kernel communicates with individual device drivers; which in turn communicate with IO devices.
- The application at the top level communicates only with the kernel.

Hardware software interface



APP.
For a user or an application the device access is with look and feel of a file read / write operation
Device independent Communication
(SHELL) OS
An OS receives the input from the user and initiates a utility
(KERNEL)
An OS utility initialises a device driver for communication
SW utility for the device
DEV. DR.
The device drivers communicate with device controllers
SW Interface for device controller
DEV. CON.
The device controller communicates with devices
HW Interface for device controller
DEV.
The device communicates Data in bits and bytes
HW Interface for the device

I/O AND THE KERNEL

- Each IO request from an application generates the following:-

  - Naming or identification of the device to communicate.

  - Providing device independent data to communicate.

I/O AND THE KERNEL

- Kernel IO subsystem arranges for the following:-

  - Identification of the device driver.

  - Allocation of buffers.

  - Reporting of errors.

  -Tracking the device usage(is the device free, who is the current user, etc.)

I/O and the Kernel

- The device driver transfers the kernel IO request to set device controller with the following information:-

  - Identify read/write request.

  - Set controller registers for data transfer (Data count = 0; where to locate data, how much data to transfer, etc).

  - Keep track when the data has been transferred(when fresh data is to be brought in).

DEVICE DRIVER

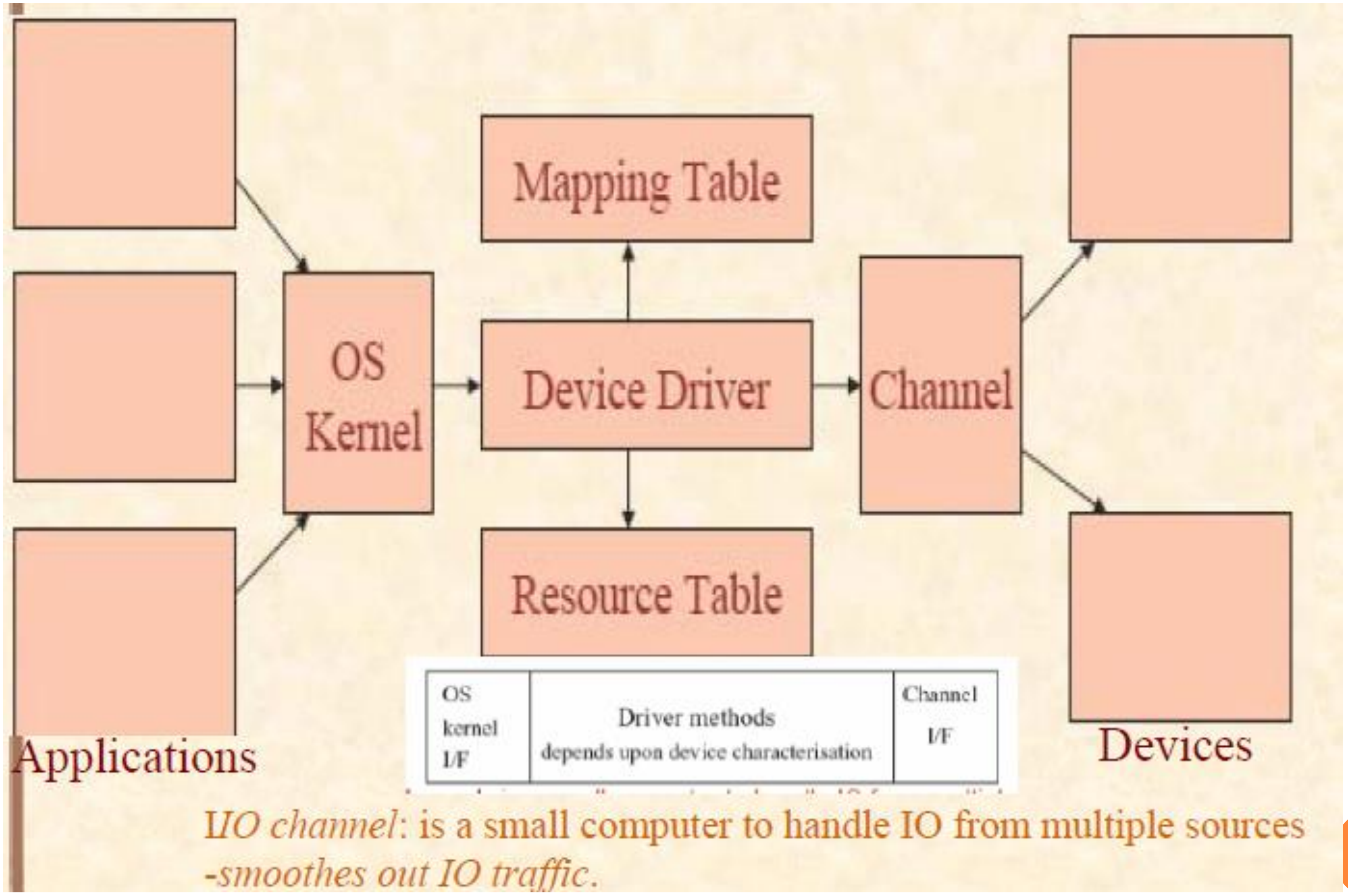
- A device driver is a specialized software which is specifically written to manage communication with an identified class of devices.

- The devices of different makes may differ in speed, the sizes of buffer and the interface characteristics, etc.

- Therefore requires different drivers.

- Device drivers present a uniform interface to the OS.

DEVICE DRIVER



OS kernel I/F	Driver methods depends upon device characterisation	Channel I/F

I/O channel: is a small computer to handle IO from multiple sources *-smoothes out IO traffic.*

Device Driver Interface

Device Driver

- In a general scenario, n applications may communicate with m devices using a common device driver.

- In that case the device driver employs a mapping table to achieve communication with a specific device.

- Drivers may also need to use specific resources (like a shared bus).

- If more than one resource is required, a device

  driver may also use a resource table. Sometimes a device driver may need to block a certain resource for its exclusive use by using a semaphore 1.

Device Driver

- The device driver methods have device specific functionalities using standard function calls.

- Function calls : *open(), close(), lseek(), read(), write ()*
- These calls may even be transcribed as *hd-open(), hd-close(), etc. to reflect their use in* the context of hard-disk operations.

- The user views device communication as if it were a communication with a file so an arbitrary amount of

  data.

- The device driver has to be device specific. It cannot choose an arbitrary sized data transfer. The driver must manage fixed sizes of data for each data transfer.

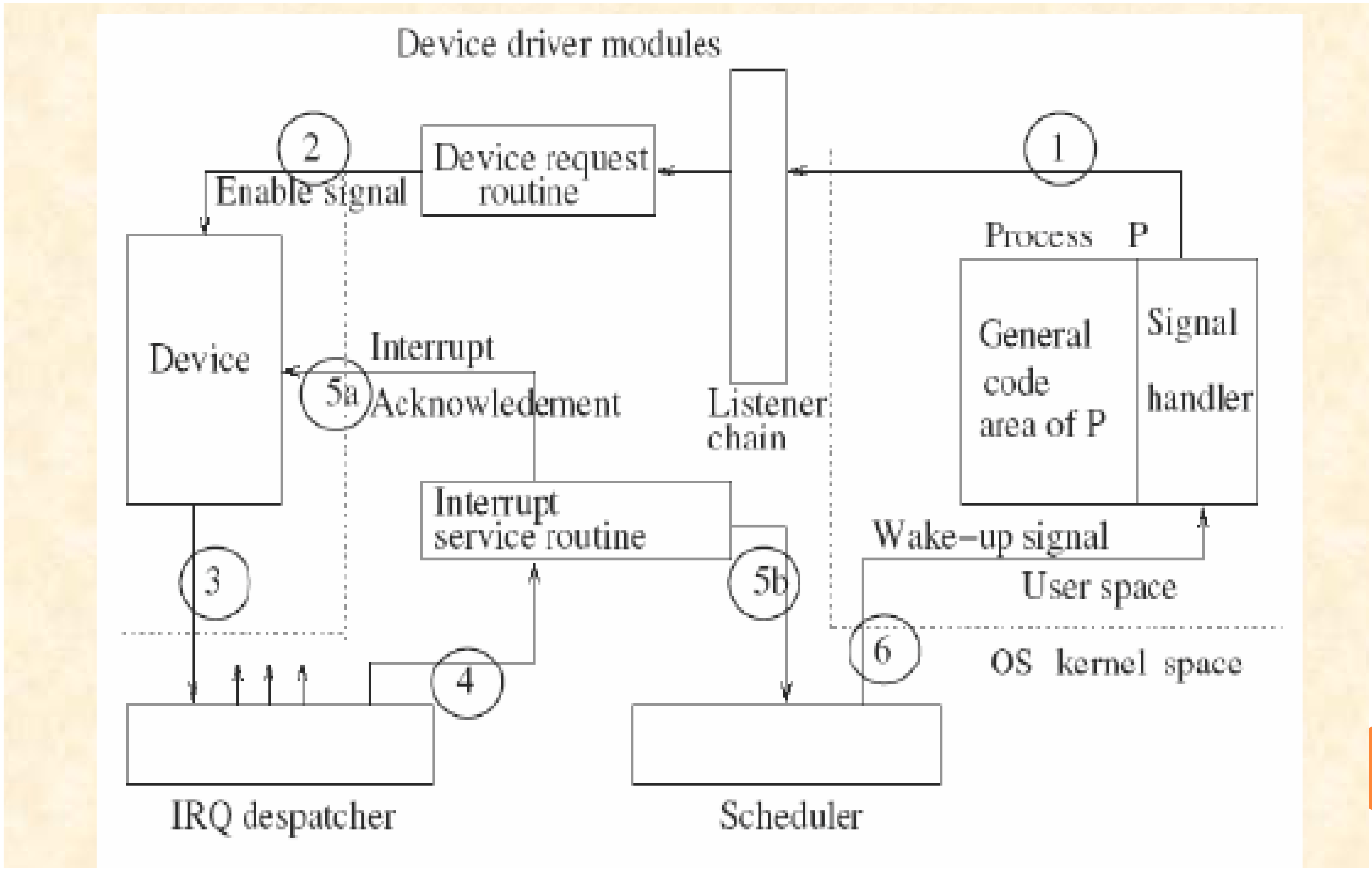
- With n applications communicating with m devices the device driver methods assume greater levels of complexity in buffer management.

Handling interrupt using device driver

- Register with listener chain of the driver.
- Enable the device.
- Interrupt request handling.
- Interrupt acknowledge and interrupt servicing.
- Schedule to complete the communication.
- Generate a wake up signal.

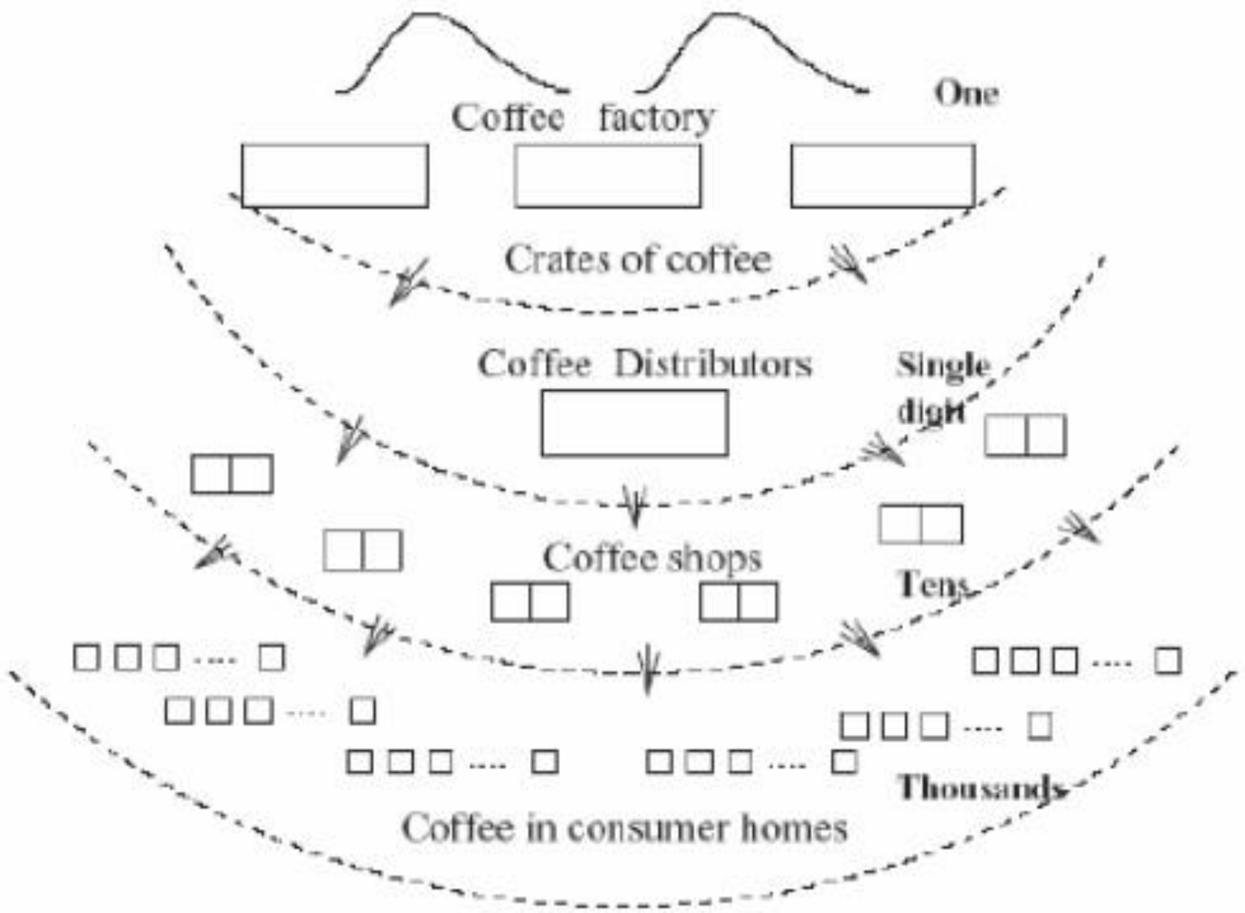
DEVICE DRIVER OPERATION

The figure below shows sequence which the device driver follow to handle interrupt :-



Management Buffer

Example in the figure shows how at different stages the buffer sizes may differ.



Management of Buffers

- Buffers are usually set up in the main memory or in caches.
- Caches are used to obtain enhanced performance.
- Buffers absorb mismatch in the data transfer rates of the processor or memory on one side and the device on the other.

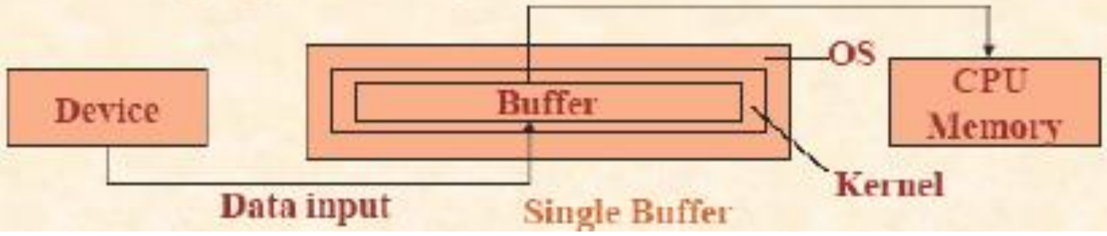
Management of Buffers

Assume we are seeking input of data from a device. The various buffering strategies are as follows:

Single buffer : The *device first fills a buffer*.

Next the device driver *hands in its control to the kernel* to input the data in the buffer.

Once the buffer has been used up, the device fills it up again for input.



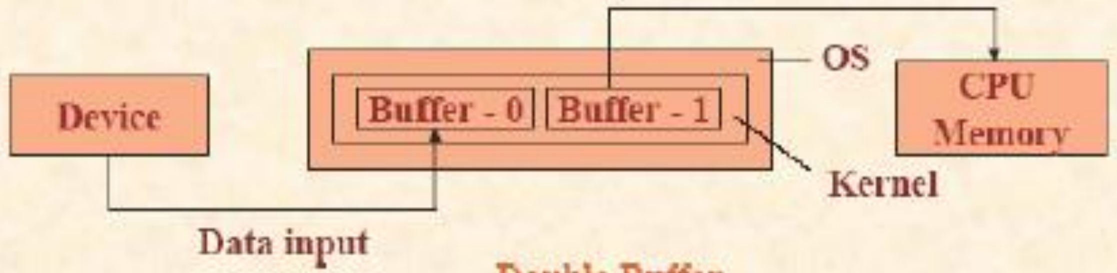
Management of Buffers

Double buffer : Here there are *2 buffers*.

Device driver starts by filling *buffer-0* and then hands it to the kernel to be *emptied*.

In the meanwhile it starts to fill *buffer-1*.

The roles are switched when *buffer-1* is filled out.



Double Buffer

MANAGEMENT OF BUFFERS

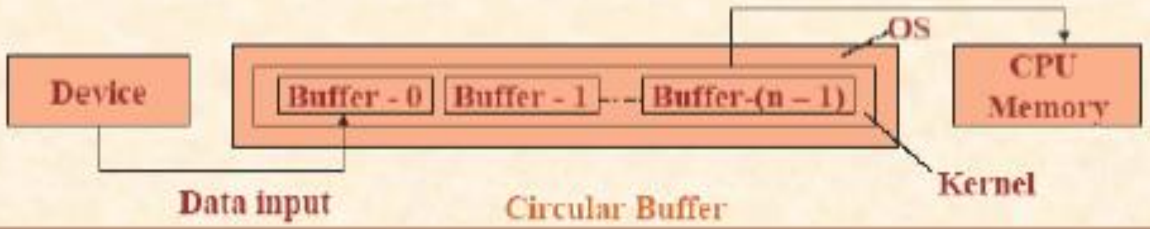
Circular buffer : One can say that double buffer is a circular queue of size 2.

We can have many buffers in a *circular queue*.

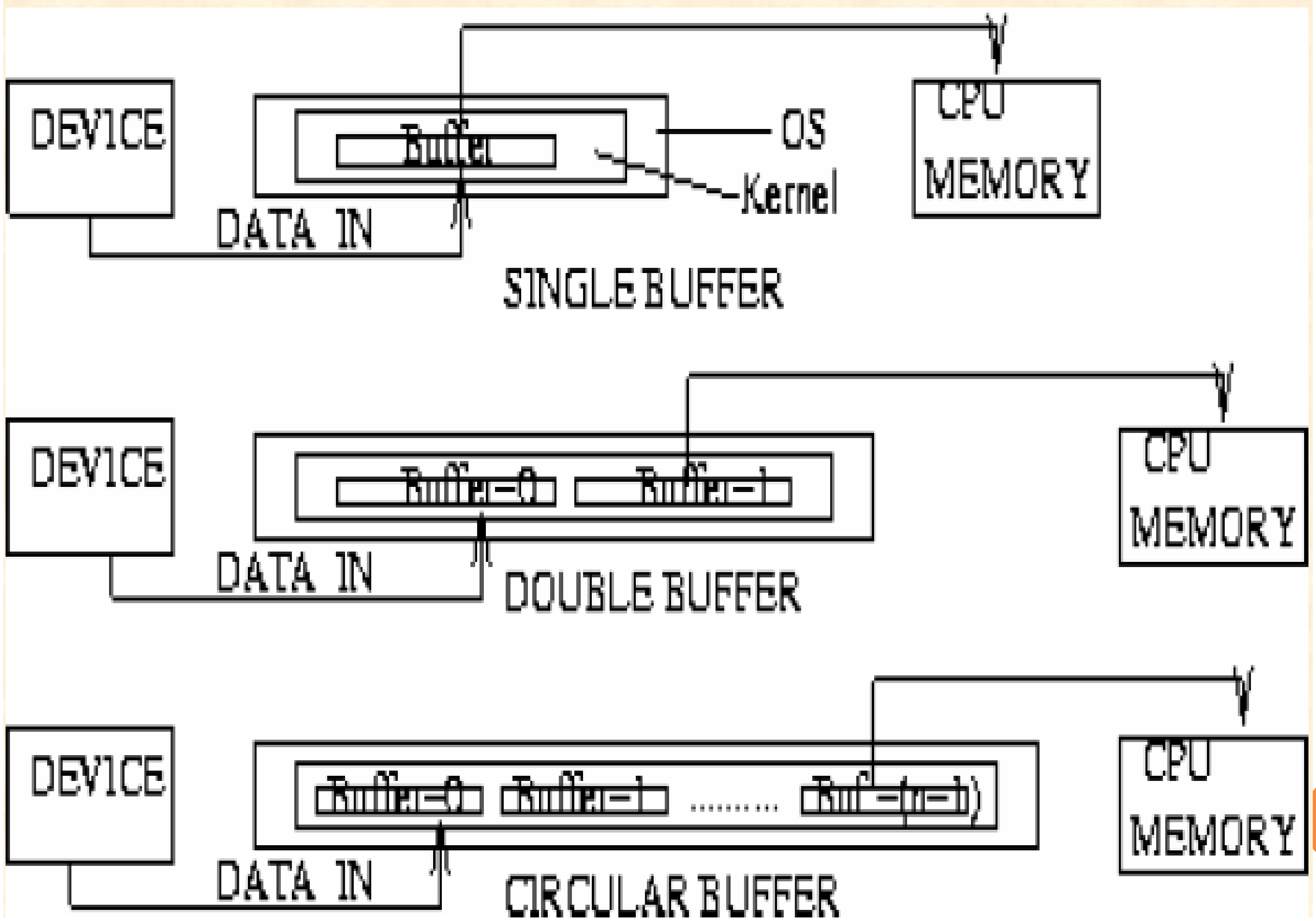
Kernel accesses filled out buffers in the same order that these are filled out.

Note that buffer management requires *management of a queue* data structure.

One must have *pointers to the head and tail of this queue* to determine if it is *full* or *empty*.

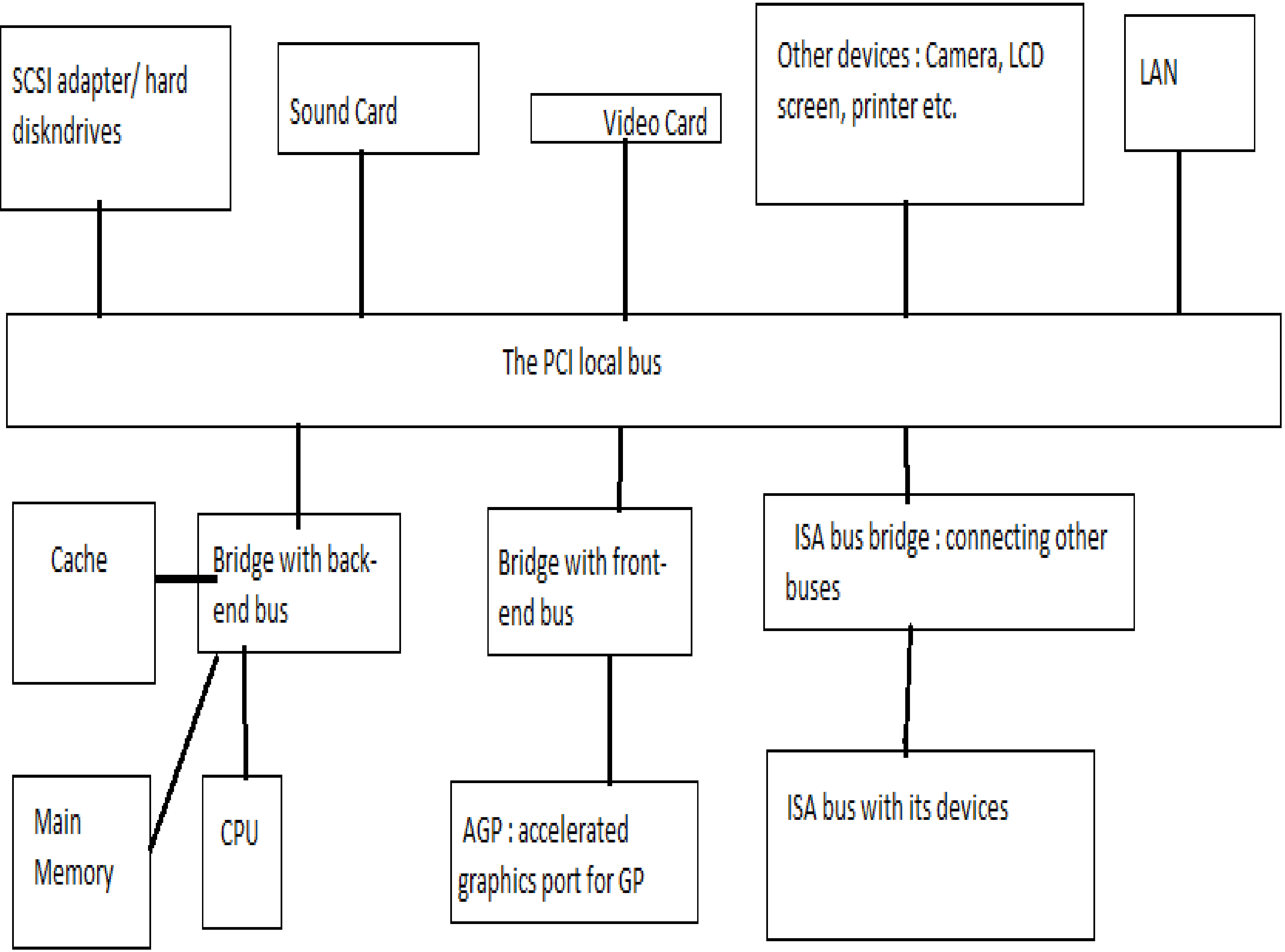


Buffering Schemes



PCI Bus

- Peripheral Controller Interface.
- System architects recognized the limitations of the internal buses.
- Limitation : bus structure was too closely tied to the system architecture.
- Microprocessor architecture supports a front end bus (local bus) and a back end bus (system bus).
- Front end bus : used for communication with devices like disks, video & sound cards, network cards & bridge that connect other buses.
- Back end bus : high speed communication between CPU, main memory & caches.



SCSI adapter/ hard diskndrives
Sound Card
Video Card
Other devices : Camera, LCD screen, printer etc.
LAN
The PCI local bus
Cache
Bridge with back-end bus
Bridge with front-end bus
ISA bus bridge : connecting other buses
Main Memory
CPU
AGP : accelerated graphics port for GP
ISA bus with its devices

PCI Bus : Modes of communication

- 3 primary modes supported by every PCI bus are :-
- Burst mode : An address is set up & multiple data sets are communicated over the bus to the designated address.
- Avoids overheads, by storing the address for communication.
- Similar to cache bursting for memory transfers.

PCI Bus : Modes of communication

- Bus Master mode : PCI supports direct communication between two entities connected to the bus.

- Closest to DMA mode.

- Also referred as 1st party DMA.

- Because here instead of 3rd party sets up communication, the bus directly offers the connection.

PCI Bus : Modes of communication

- High bandwidth mode : Accelerated with graphics port (AGP).
- Useful for streaming IO.
- High bandwidth easily achieved with 64 bit operation.

PCI Bus : Modes of communication

- Point to point mode : PCI express, which operates with 64 bit data transfers at 66 MHz, may also operate like a switch for point to point communication.

- Beyond the usual shared bus operations.

- Shared bus operations avoid direct links.

- But point to point mode requires direct links (not shared) to be set up between the devices.

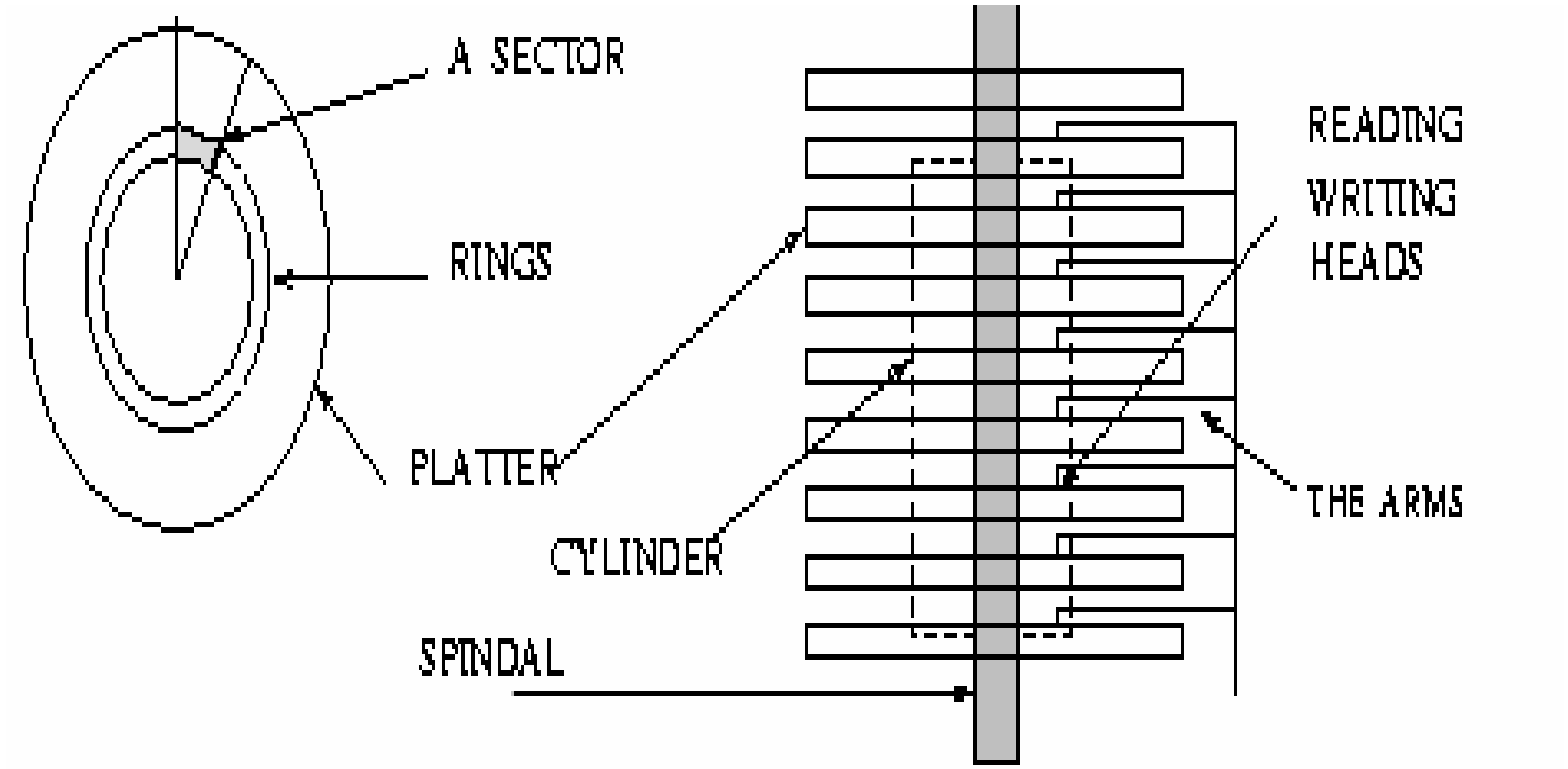
PCI Bus : Modes of communication

- Interrupt transfers : PCI bus has internal interrupt control at 4 levels identified as #A through #D.
- When used these levels need to be mapped to the IRQ priorities at four distinct levels.

Points related to device communication

- PCI provisions handshake & acknowledgement signals for reliable network device communication.
- ISA continues to be used for serial & parallel port transfers. Several devices ISA compatibility. Usage of bus bridge allows PCI to connect to devices.
- Conflicts in communication due to two devices seek to communicate with same device. But PCI bus has embedded arbitration logic to resolve conflicts.

Information Storage Organization on Discs



NOTE THAT THE RINGS ON THE DISC PLATTERS ON THE SPINDLE FORM A CYLINDER SINCE ALL HEADS ARE ON A PARTICULAR RING AT THE SAME TIME SO IT IS EASY TO ORGANISE INFORMATION ON A CYLINDER. THE INFORMATION IS STORED IN THE SECTORS THAT CAN BE IDENTIFIED ON THE RINGS. SECTORS ARE SEPARATED FROM EACH OTHER. ALL SECTORS CAN HOLD EQUAL AMOUNT OF INFORMATION

INFORMATION STORAGE ORGANIZATION ON DISCS

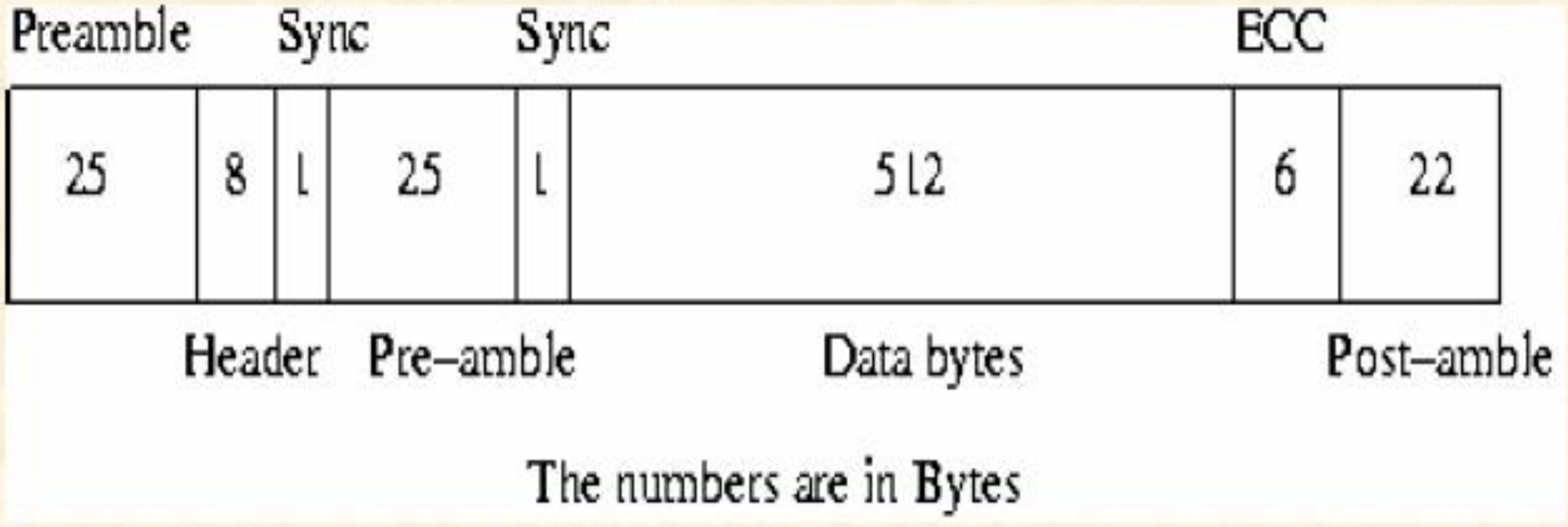
- A disc has several platters each with several rings or tracks.
- Rings are divided into sectors where information is actually stored.
- Sequentially related information is organized into cylinders.
- Information on different cylinders has to be accessed by moving the arm – seek latency.

Delays in Information Retrieval

- Rotational delay is due to the waiting time for a sector in rotation to come under the read or write head.
- The motivation for disc scheduling comes from the need to keep both the delays to a minimum.
- A sector stores a lot of other information in addition to a block of information.

Delays in Information Retrieval

Information Storage in Sectors.



Note that we require 10% of extra information for a 512 byte data. Clearly, for larger block sizes this constant over head becomes less significant. However, larger block sizes would require larger buffers.

Scheduling Disk Operations

- A user communicates with files (program, data, system utilities etc.) stored on discs. All such communications have the following components.
- The IO is to read from, or write into, a disc.
- The starting address for communication in main memory.
- The amount of information to be communicated.
- The starting address in disc and current status of the transfer.

A Scenario for Information Retrieval

- Consider a scenario of one process with one request to access data; a disc access request leads finally to the cylinder having that data.

- When multiple requests are pending on a disc, accessing information in a certain order becomes essential – disc access scheduling.
- ‘;
- For example, if there are 200 tracks on each platter, pending requests may come in the order – 59, 41, 172, 74, 52, 85, 139, 12, 194 and 87.

Comparison of Policies

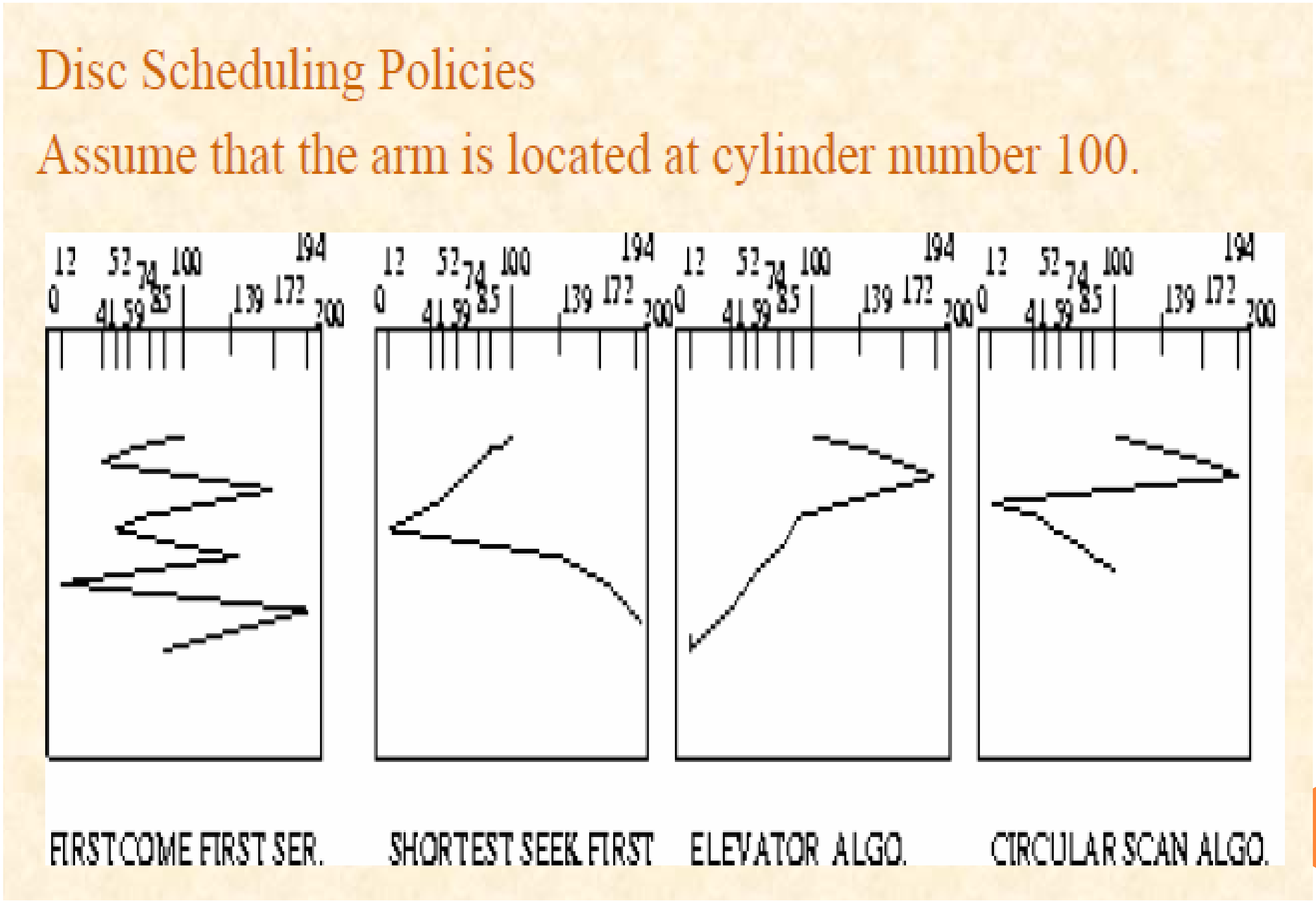
- **FCFS Policy :** The service is provided strictly in the sequence in which the requests arrived.

- The service would be in the sequence :-

  59, 41, 172, 74, 52, 85, 139, 12, 194 and 87.

- In order to compare the effect of implementing a certain policy, we analyze the arm movements – captures the basic disc activity.

Comparison of Policies


Disc Scheduling Policies

Assume that the arm is located at cylinder number 100.

Comparison of Policies

- **Shortest Seek First** **:** Disc access is in the order:

  87, 85, 74, 59, 52, 41, 12, 139, 172, 194.

- **Elevator (SCAN) Algorithm** **:** Assume an initial movement is in a certain direction. Disc access is in the following order :

  139, 172, 194, 87, 85, 74, 59, 41, 12.

Comparison of Policies

- **Circular Scan** : This scan policy is done in one

  direction and wraps around. The disc access will be in the following order:

  139, 172, 174, 12, 41, 52, 59, 74, 85, 87.

- From these graphs we find that FCFS is not a very good policy; Shortest Seek First and Elevator algorithm seem to perform well as these have least arm movements.

USB (Universal Serial Bus) - Need

- In 1990s, major problem with PCs was difficult in connections to a variety of devices(mouse, keyboards, joysticks for game, modems, printers, scanners, speakers, mobile phones, web cams, LAN cards etc.
- Each device have its own connectors with a designated pin configuration to support voltage levels, control signals & its customized cable etc.
- Also its own IO protocol(for initiating & maintaining connection) & drivers.

USB (Universal Serial Bus)

- Offers a common cable & connector configuration (blind mating) to connect the wide range of devices.

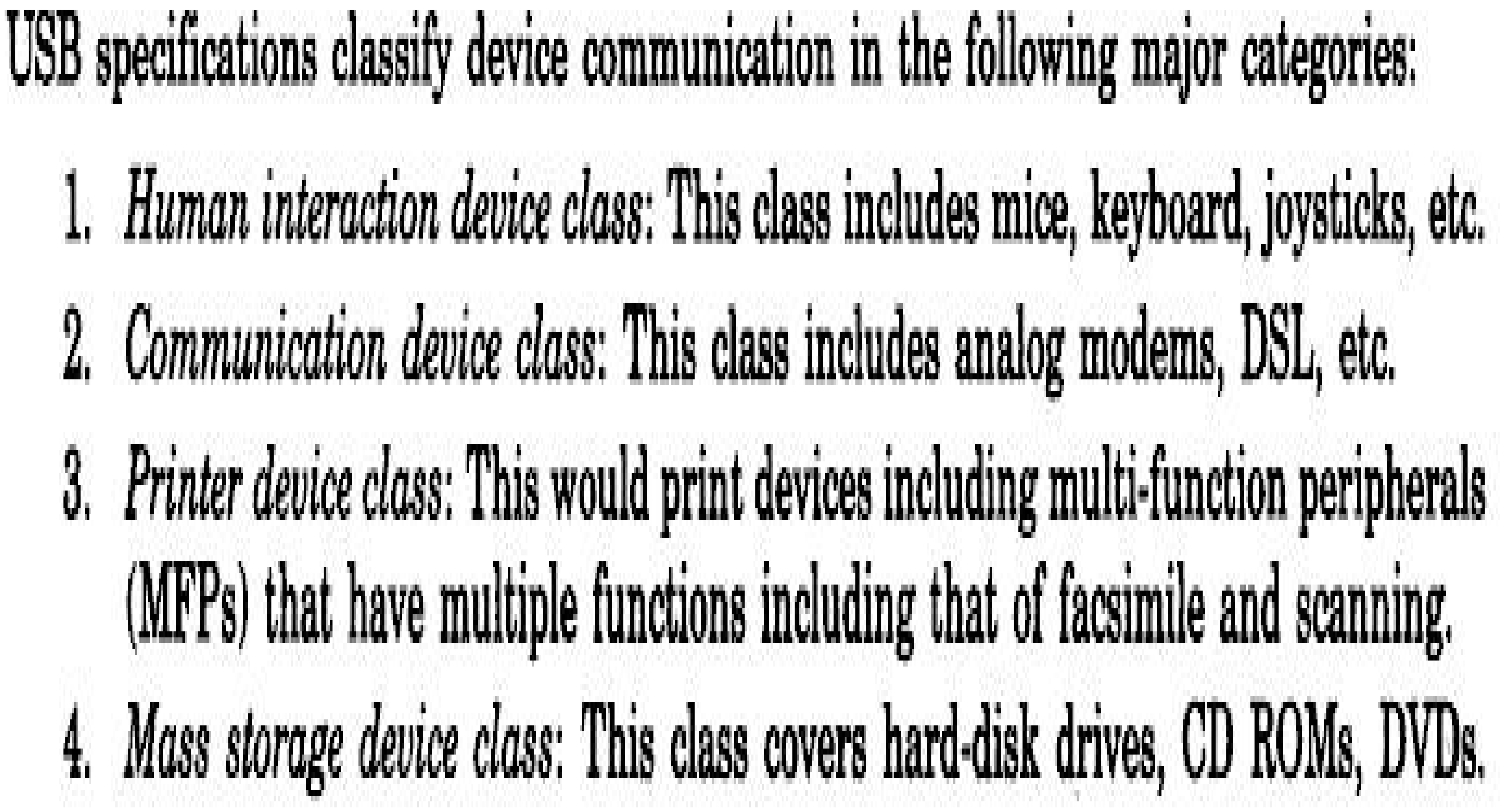
- Means a slots on PC can be flexibly used.

- The industry reckons a USB interface to be “hot swappable”- which implies that one can plug-out one device & plug-in another device in that slot, without rebooting the PC.

USB : A BRIEF HISTORY

Application's environment	Speed	USB Specifications	Some Observations
Interactive i/p as in games	Low 10-100 kbps	USB 1.1(announced in 95, but released in 96, popular till 2000)	Low cost interface for human I/F devices like mouse, keyboards, joysticks
Bulk transfer for data services as required in ISDN, audio etc	Medium 1.5-12 Mbps	USB 1.1(Intel announced USB host controller & Philips announced USB audio for isochronous communications with consumer electronics devices)	Useful when latency is to regularly guaranteed like in speech & audio.
Bulk transfer at high speed as required in LANs & video data	High 50-480 Mbps	USB 2.0(announced in Apr 2000)	Useful for data transfers with very low latency(flash memory devices, projection devices)
Bulk transfer at ultra high speed	UHS 1-5 Gbps	USB 3.0 (announced Nov 2008)	Handles variety of real time media data at 500 Mbps or more

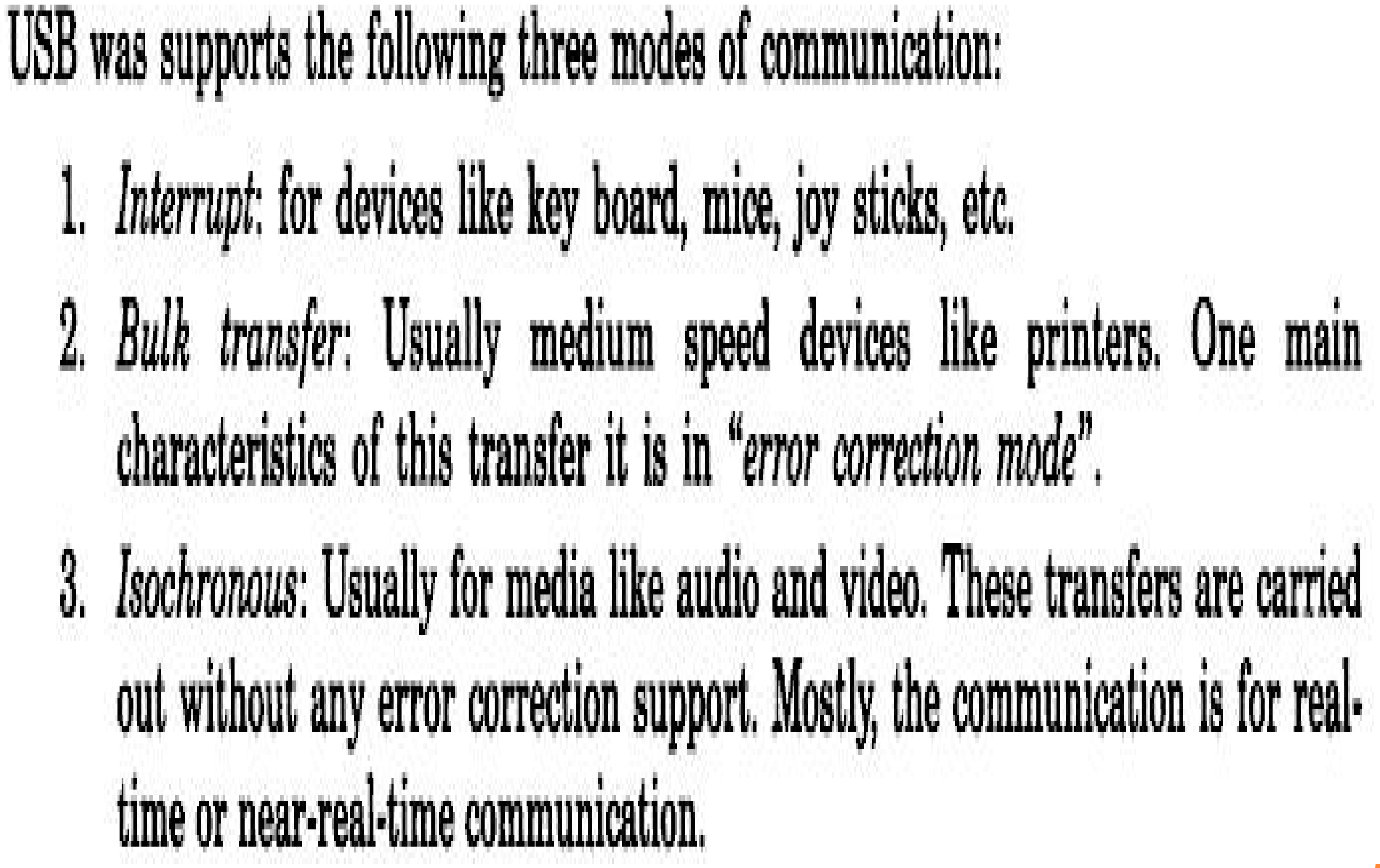
USB : Classification of devices



USB specifications classify device communication in the following major categories:

1. *Human interaction device class:* This class includes mice, keyboard, joysticks, etc.
2. *Communication device class:* This class includes analog modems, DSL, etc.
3. *Printer device class:* This would print devices including multi-function peripherals (MFPs) that have multiple functions including that of facsimile and scanning.
4. *Mass storage device class:* This class covers hard-disk drives, CD ROMs, DVDs.

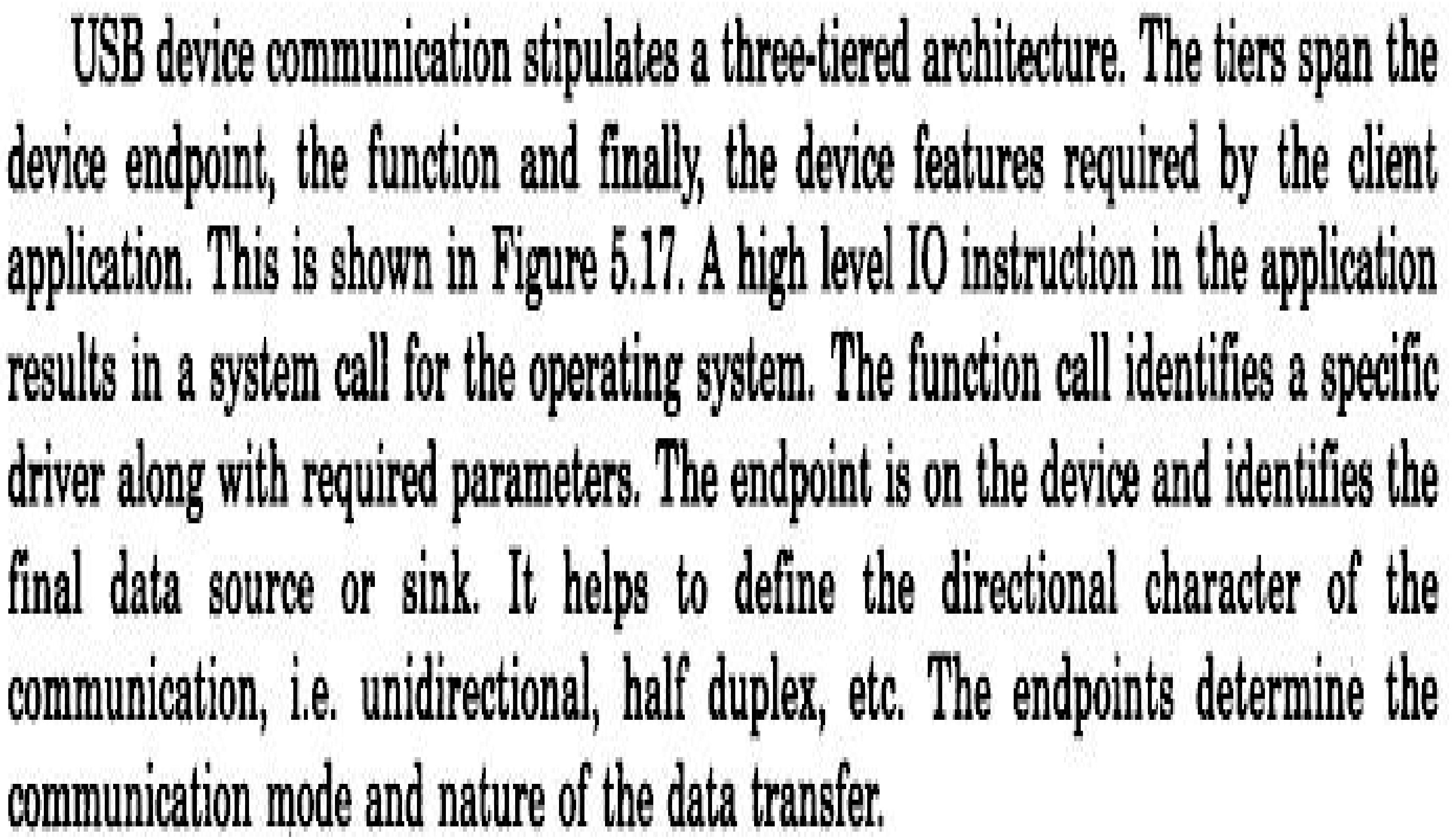
USB : Modes of communication



USB was supports the following three modes of communication:

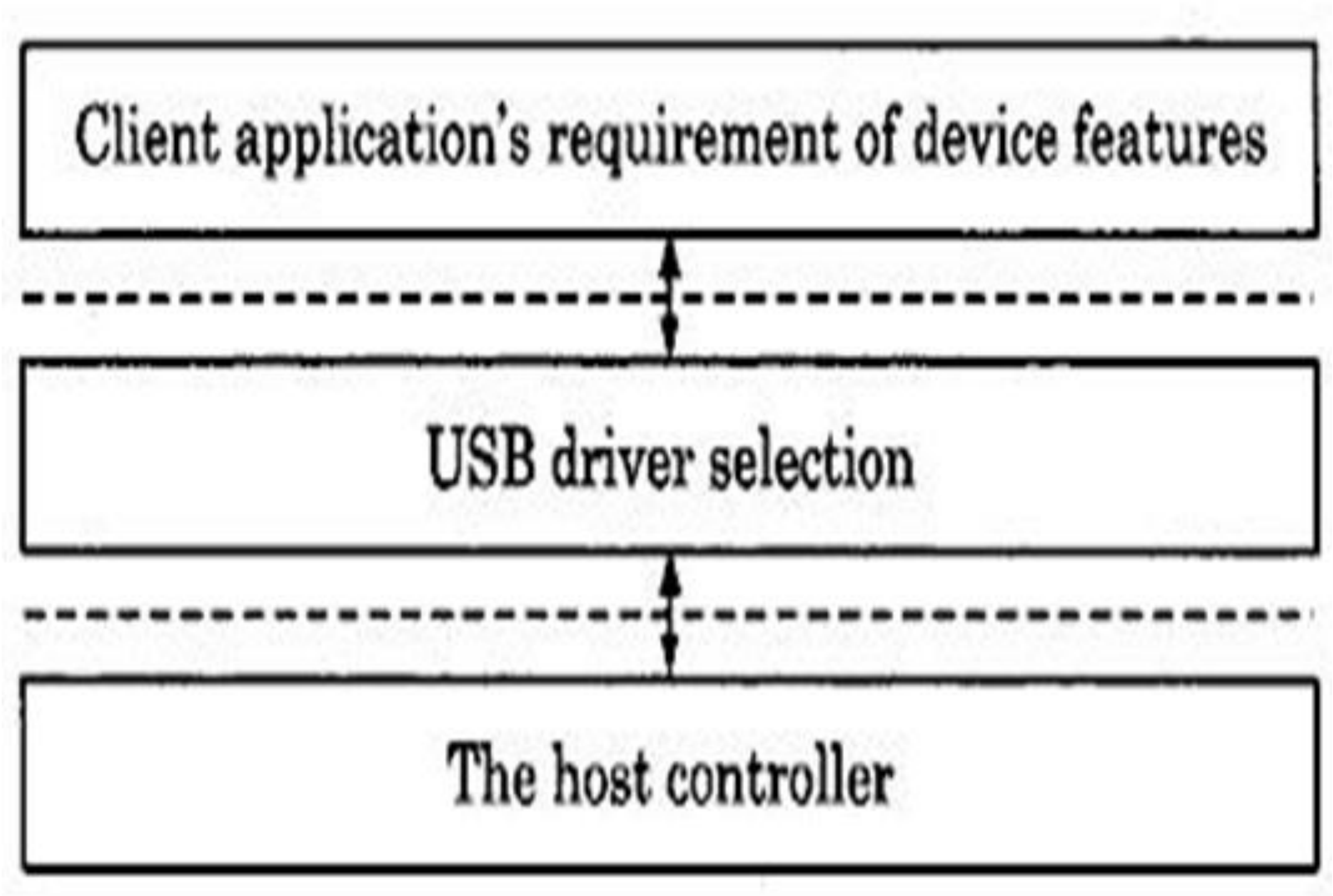
1. *Interrupt*: for devices like key board, mice, joy sticks, etc.
2. *Bulk transfer*: Usually medium speed devices like printers. One main characteristics of this transfer it is in "*error correction mode*".
3. *Isochronous*: Usually for media like audio and video. These transfers are carried out without any error correction support. Mostly, the communication is for real-time or near-real-time communication.

THREE TIER ARCHITECTURE OF USB

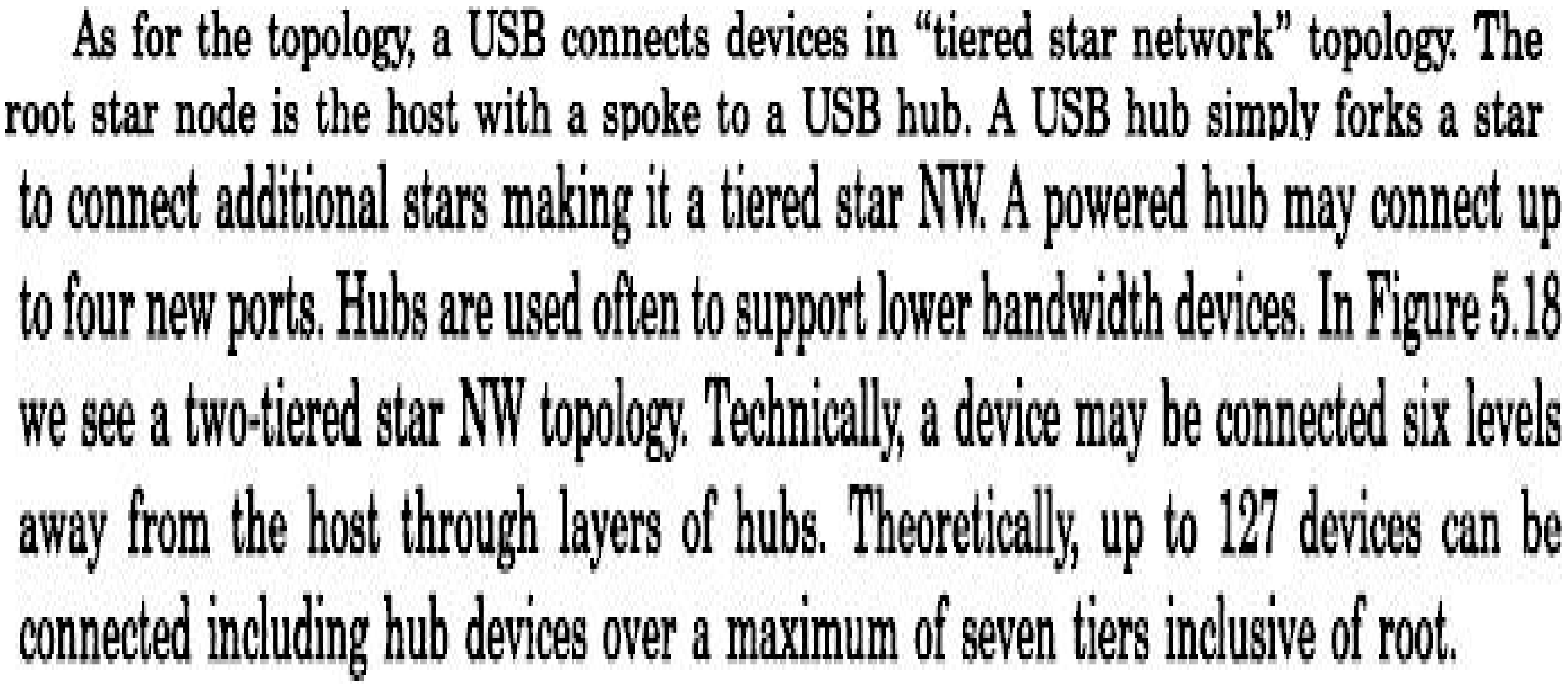


USB device communication stipulates a three-tiered architecture. The tiers span the device endpoint, the function and finally, the device features required by the client application. This is shown in Figure 5.17. A high level IO instruction in the application results in a system call for the operating system. The function call identifies a specific driver along with required parameters. The endpoint is on the device and identifies the final data source or sink. It helps to define the directional character of the communication, i.e. unidirectional, half duplex, etc. The endpoints determine the communication mode and nature of the data transfer.

Three tier architecture of USB

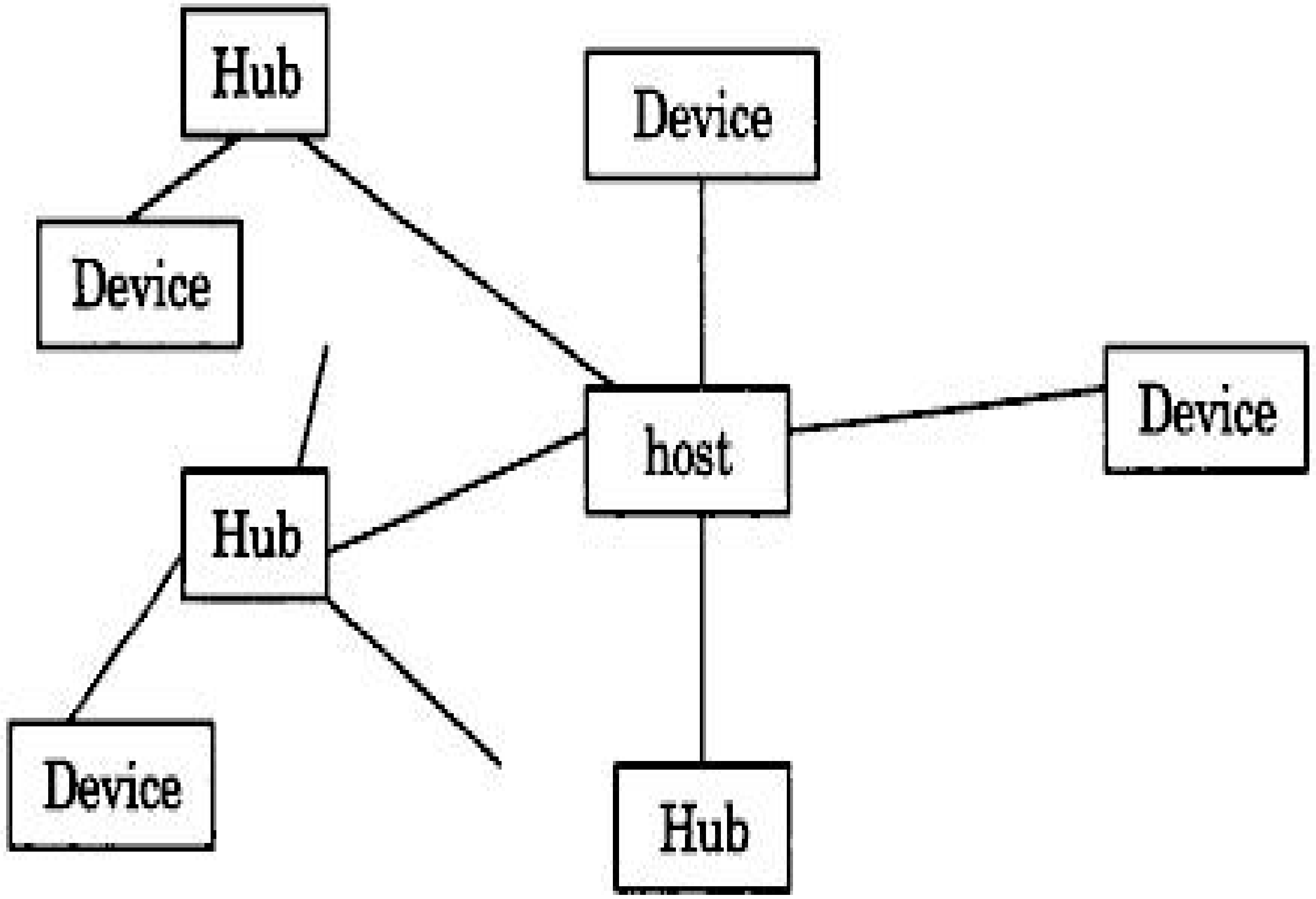


USB : Tiered star network topology

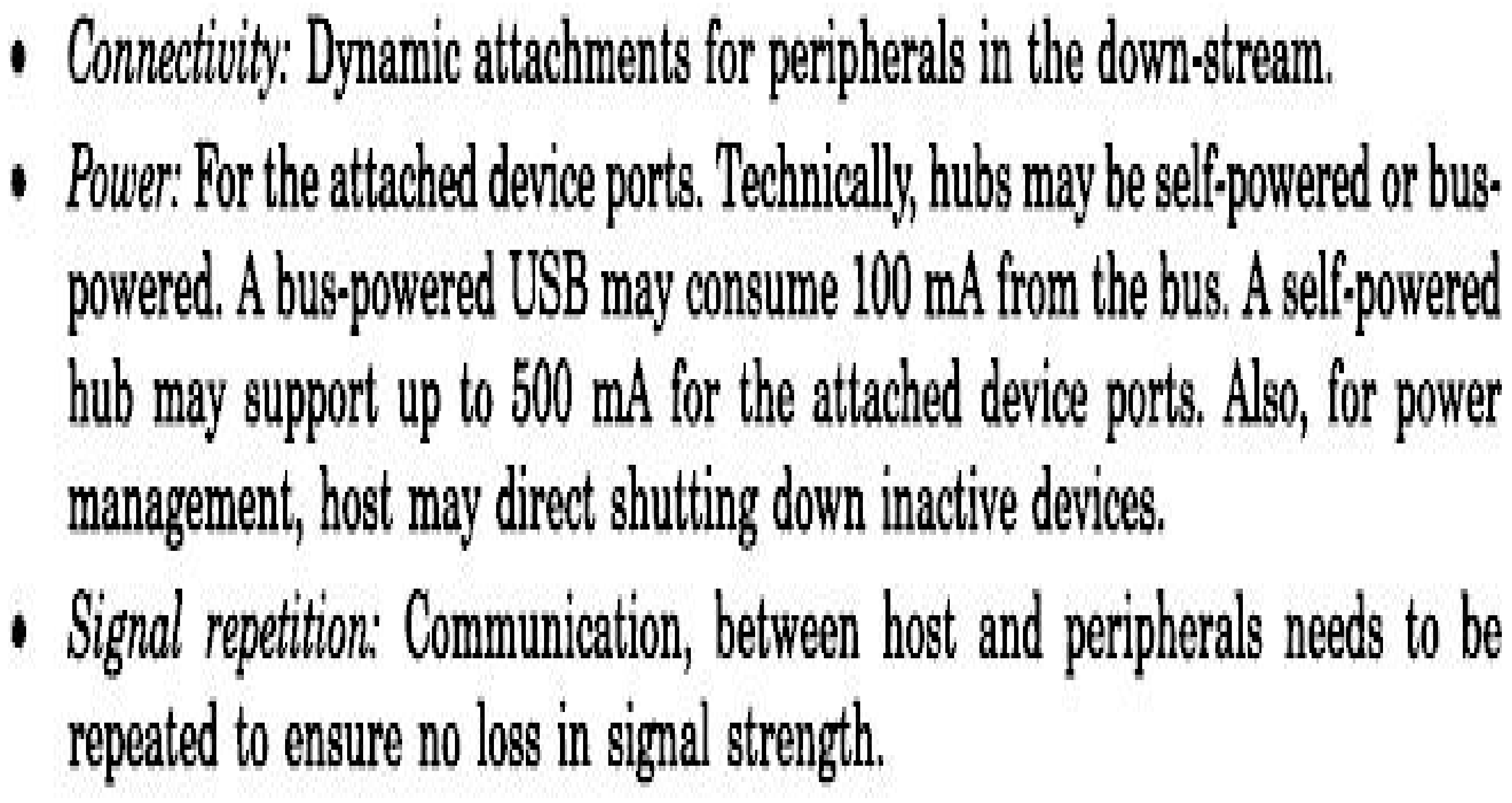


As for the topology, a USB connects devices in "tiered star network" topology. The root star node is the host with a spoke to a USB hub. A USB hub simply forks a star to connect additional stars making it a tiered star NW. A powered hub may connect up to four new ports. Hubs are used often to support lower bandwidth devices. In Figure 5.18 we see a two-tiered star NW topology. Technically, a device may be connected six levels away from the host through layers of hubs. Theoretically, up to 127 devices can be connected including hub devices over a maximum of seven tiers inclusive of root.

USB : Tiered star network topology



USB HUB PROVISIONS AS FOLLOWS :



- *Connectivity:* Dynamic attachments for peripherals in the down-stream.
- *Power:* For the attached device ports. Technically, hubs may be self-powered or bus-powered. A bus-powered USB may consume 100 mA from the bus. A self-powered hub may support up to 500 mA for the attached device ports. Also, for power management, host may direct shutting down inactive devices.
- *Signal repetition:* Communication, between host and peripherals needs to be repeated to ensure no loss in signal strength.