Input Output (IO) Management

Prof. P.C.P. Bhatt

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 IO 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 networks*.

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 -1

- A computer system may have to *synchronize* with some other process to communicate.
- So, one process may actually *wait* at the *point of rendezvous* for the other process to arrive.
- The process advances further following the *synchronizing event*.

Event Based I/O -2

- 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 -1

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.*

Programmed 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.

Interrupt mode

✓ *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*.

DMA mode

- ✓ *DMA mode* :

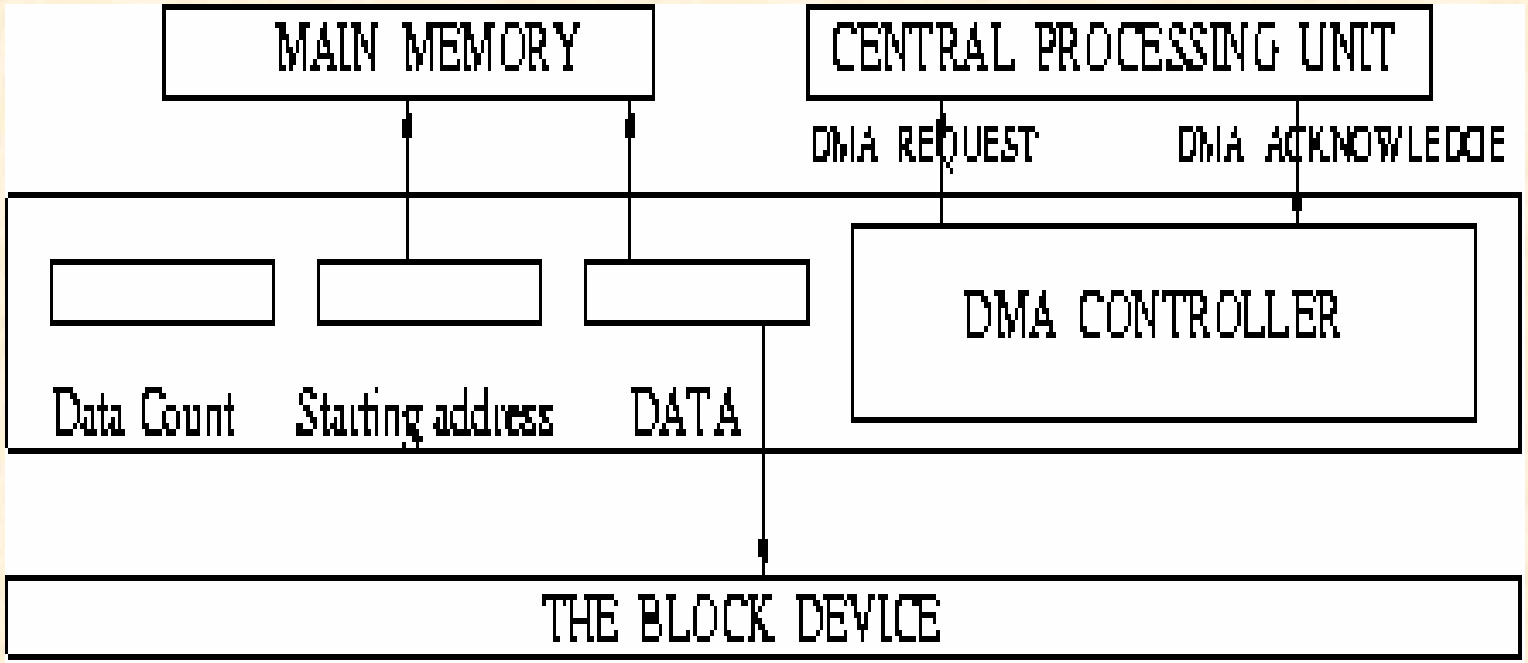
The device requests for an IO for a *block data transfer*.

The execution is briefly suspended and the starting address and size of data block are stored in a *DMA controller*.

IO Modes: some observations

- ✓ Programmed IO is used for *synchronizing information* or when *speed is not critical*, e.g. – in the i386 CPU based PC architecture, there is a notion of *listening to an IO port*.
- ✓ *Interrupt transfer* is suited for a small amount of critical information – *no more than tens of bytes*.
- ✓ *DMA* is used mostly for *block devices*.

Direct Memory Access Mode of Data Transfer



The DMA controller has request, acknowledge, interrupt and read / write lines of control

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.

Interrupt Mode - 1

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 - 2

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

Program *resumes execution* from where it was suspended.

Interrupt Types -1

Interrupt processing may require the following contexts :

Internal Interrupt :

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 *RTOS*s.

Interrupt Types -2

- *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.

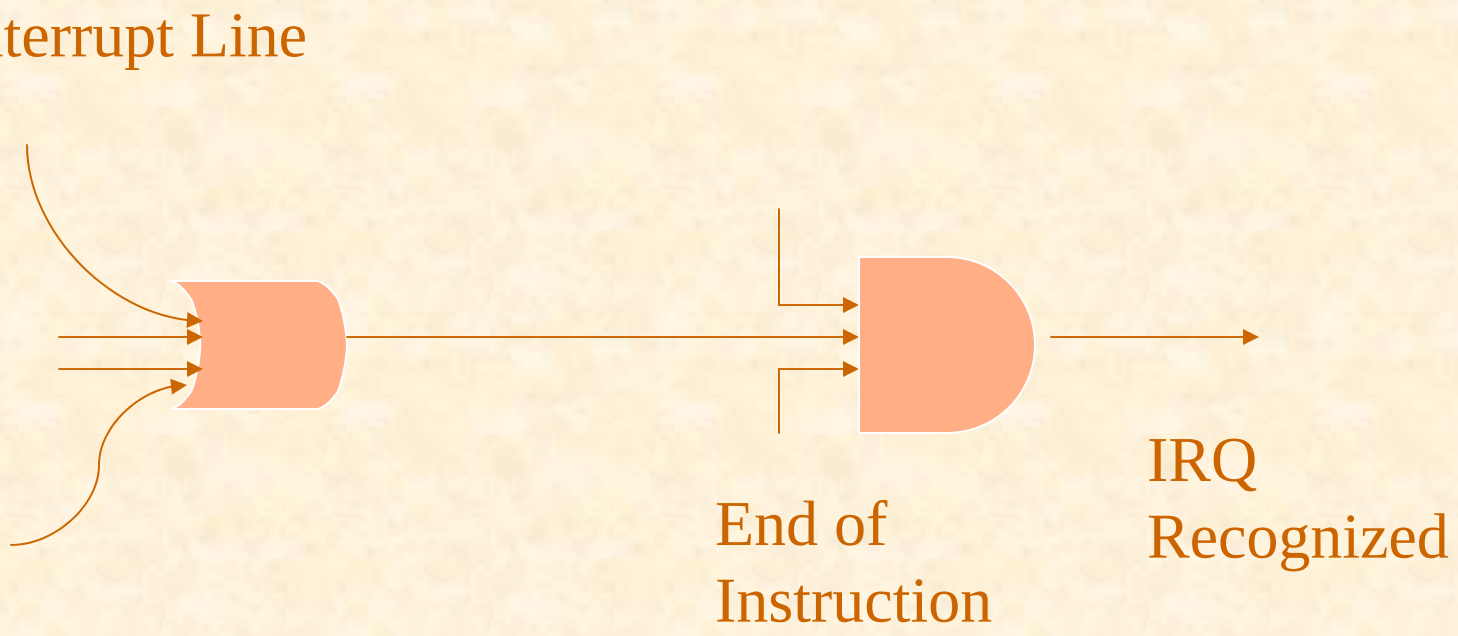
Interrupt Types -3

- *Software Interrupt* :

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*.

Hardware Support to Recognize Interrupt



Interrupt Recognition

Interrupt Servicing -1

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 -2

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* – $PADDR_{i+1}$ 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

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 $PADDR_{i+1}$

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

Identification of Source of Interrupts -1

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

Polling :

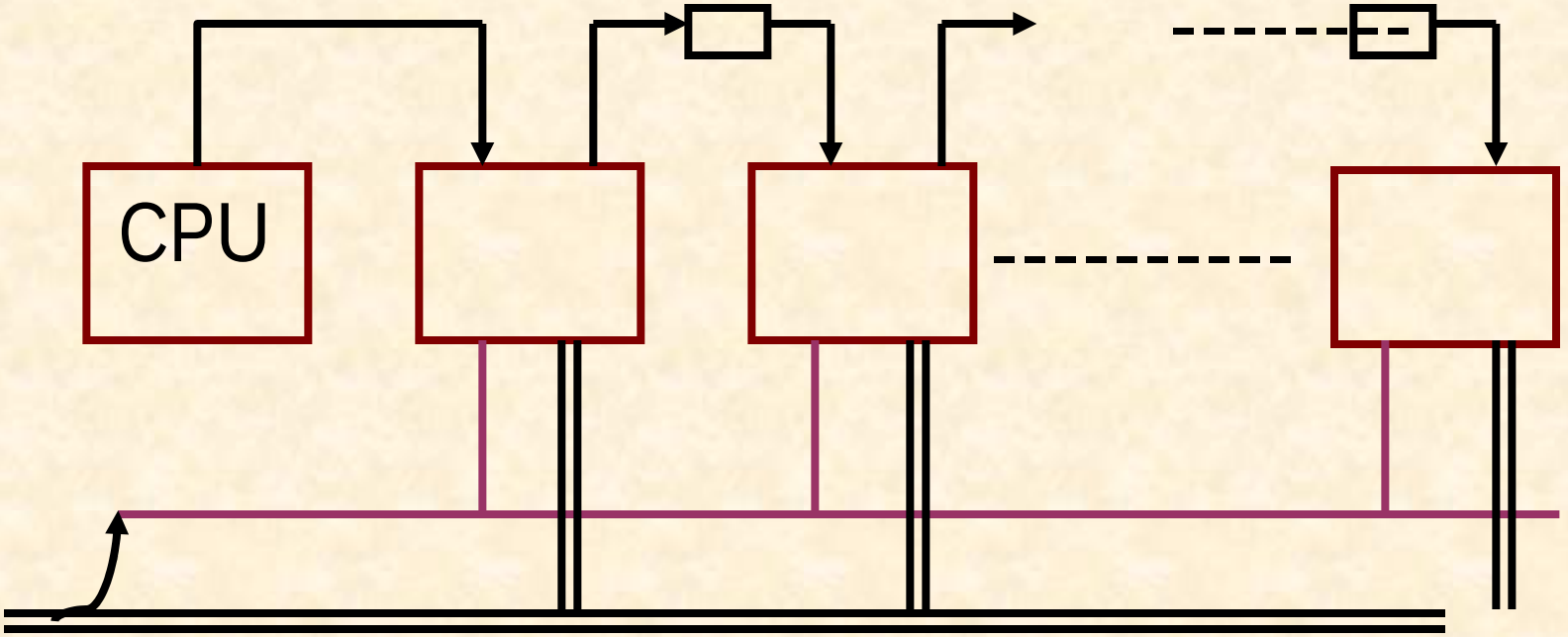
It *interrogates* all the devices in sequence

It is *slow* if there are many devices.

Identification of Source of Interrupts -2

Daisy Chain

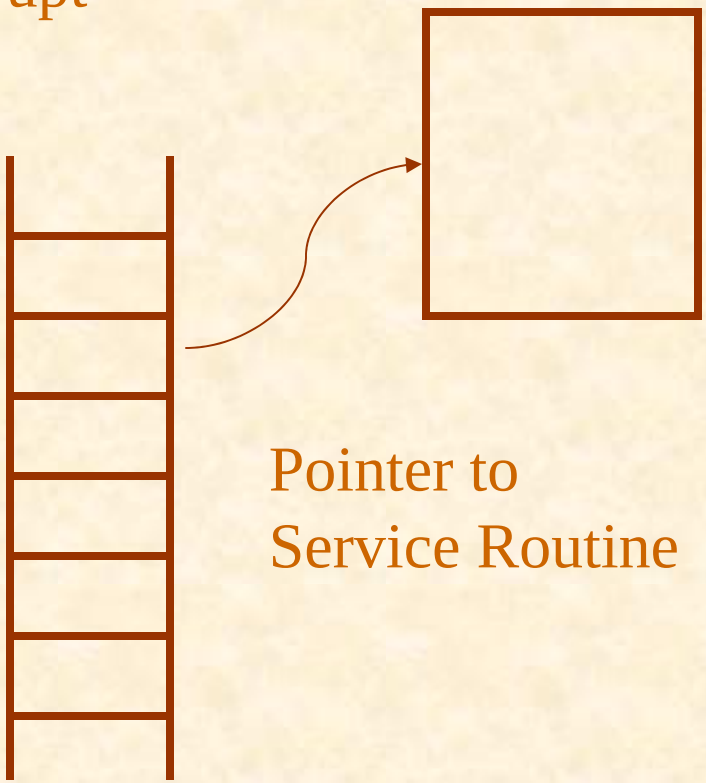
INT Address



Interrupt Line

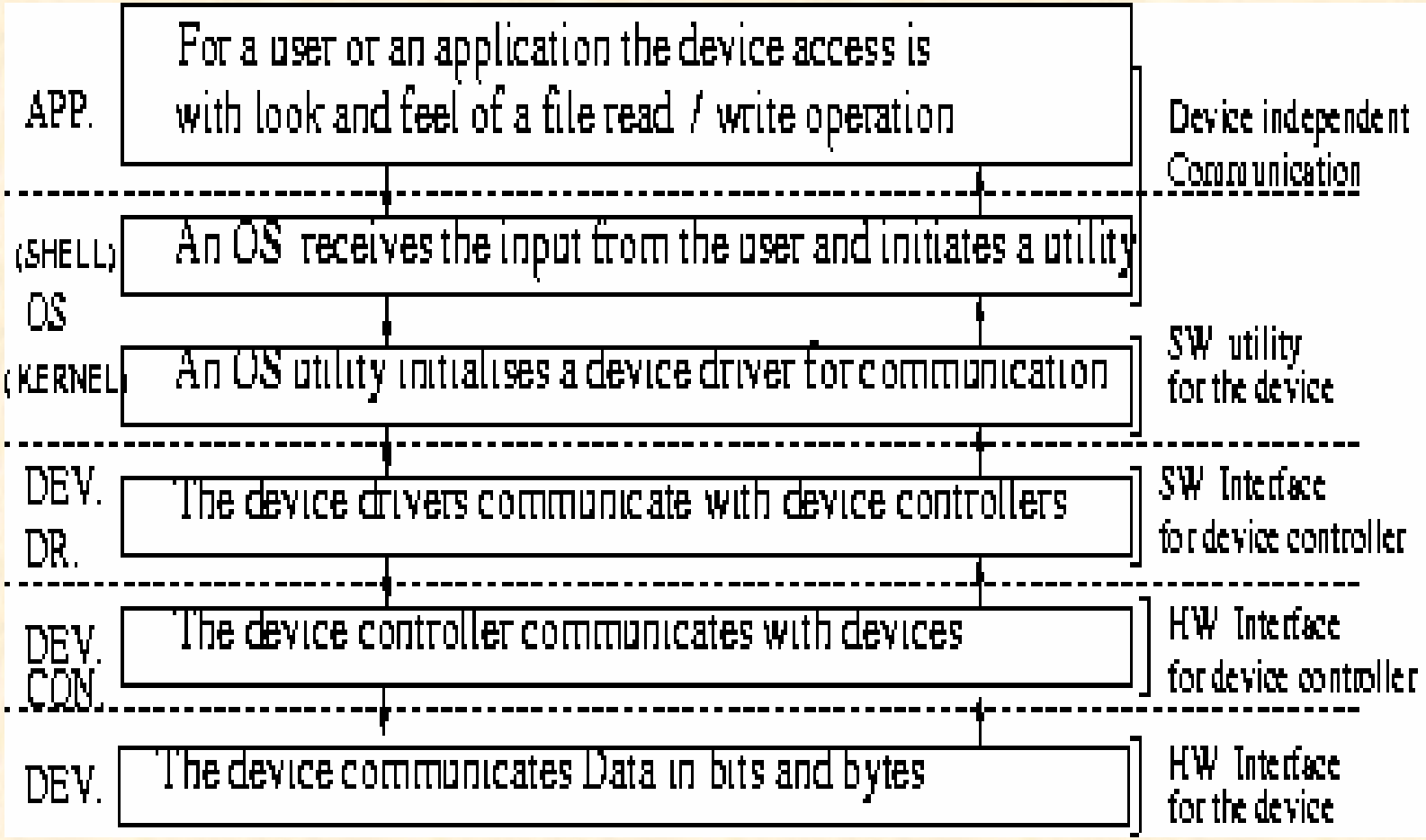
Identification of Source of Interrupts -3

Vectored Interrupt



Pointer to
Service Routine

HW/SW Interface



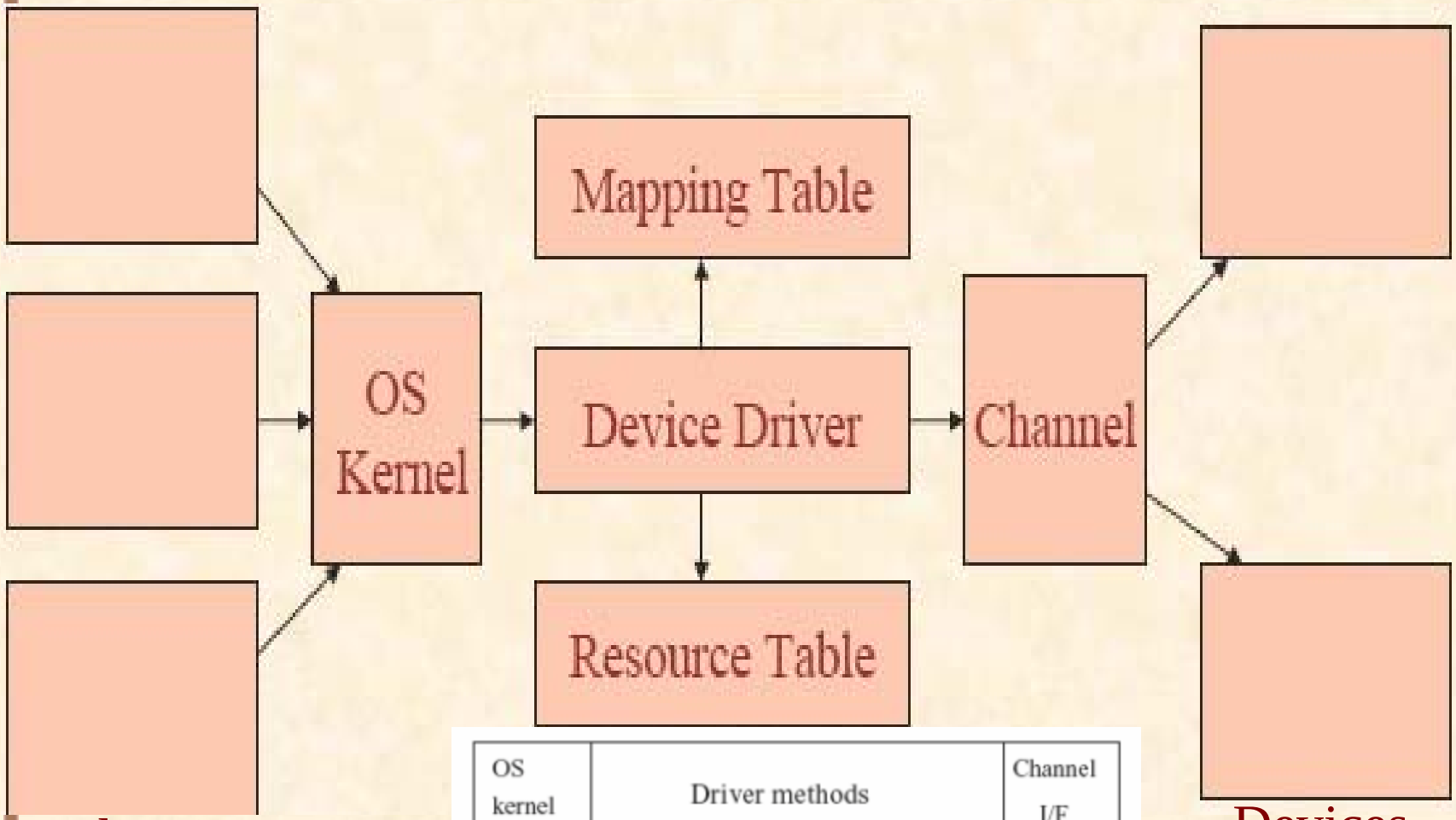
I/O and the Kernel -1

- ✓ Usually, an *OS kernel* resolves IO commands.
- ✓ Following an issuance of an IO command, 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*.

I/O and the Kernel -2

- ✓ 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 -3



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

I/O and the Kernel -4

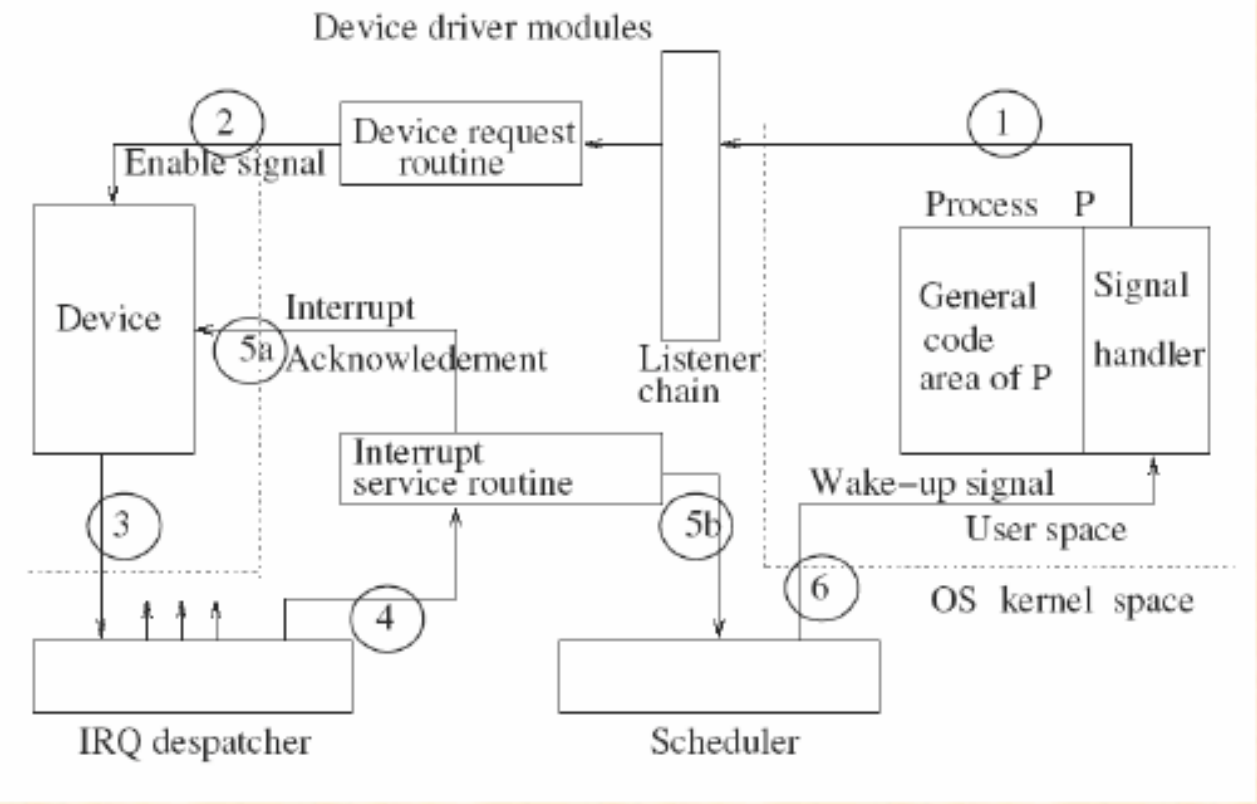
- ✓ Kernel IO subsystem arranges for the following:
  - ➢ Identification of the device driver.
  - ➢ Allocation of buffers.
  - ➢ Reporting of errors.
  - ➢ Tracking the device usage.

I/O and the Kernel -5

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

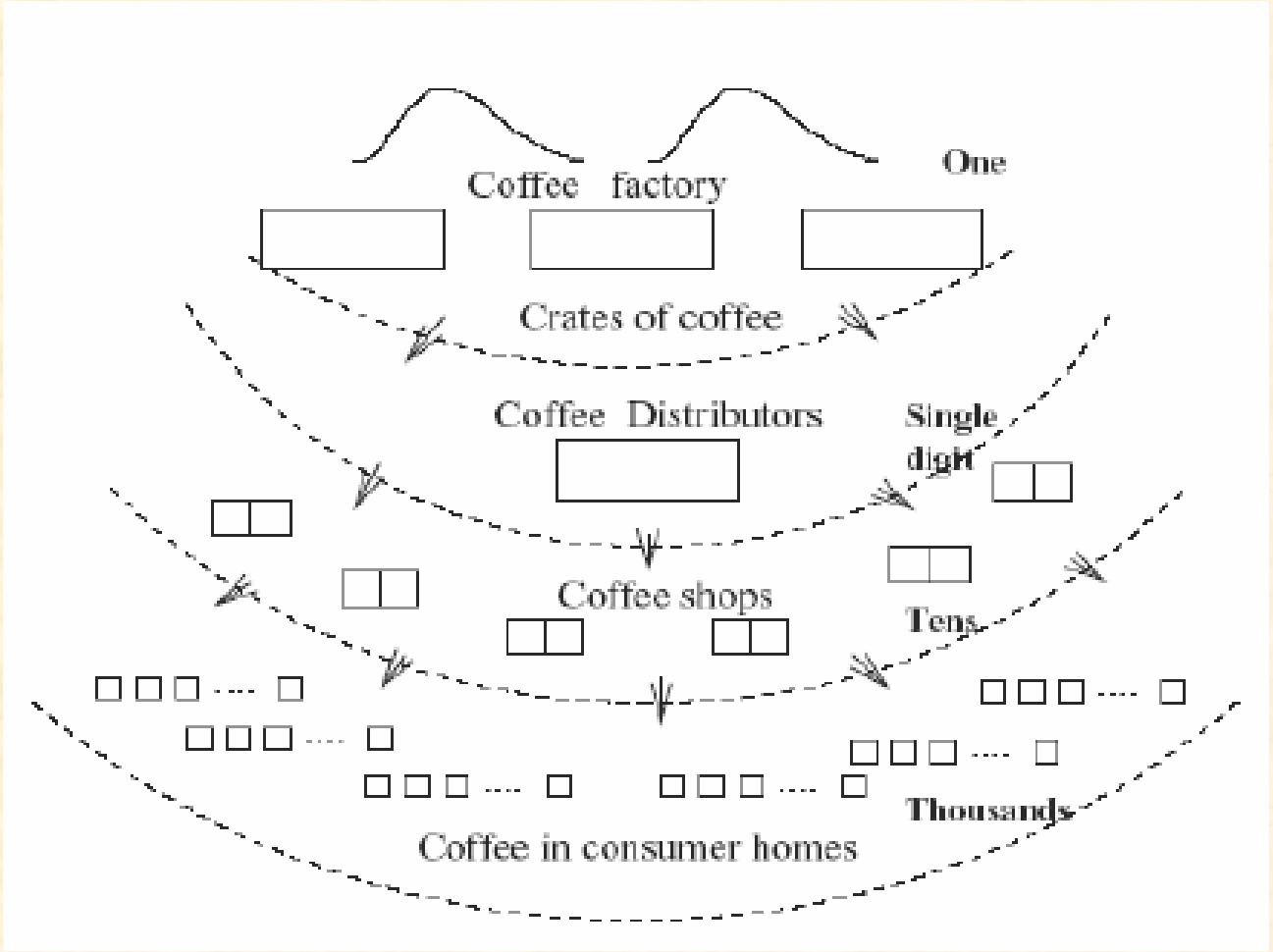
Device Driver Operation

The figure below shows the sequence which device drivers follow to handle interrupts:



Management Buffer -1

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



Management of Buffers - 2

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.

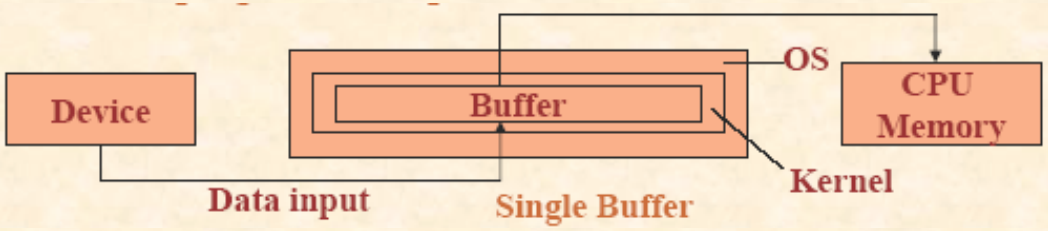
Management of Buffers - 3

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.



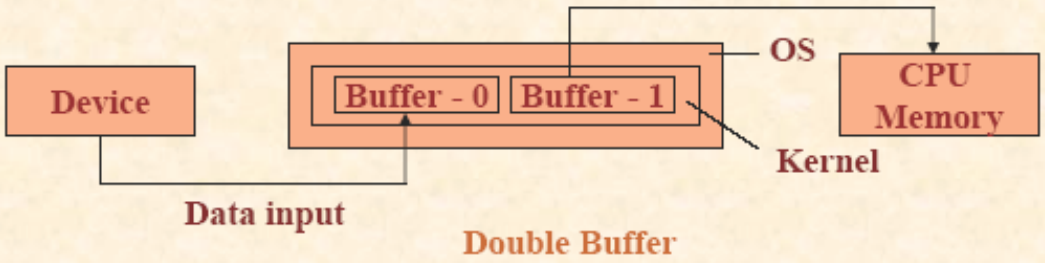
Management of Buffers - 4

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 - 5

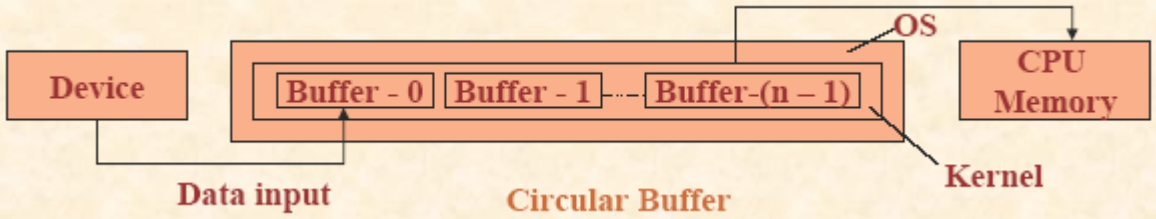
Circular buffer : One can say that double buffer is a circular queue of size 2.

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

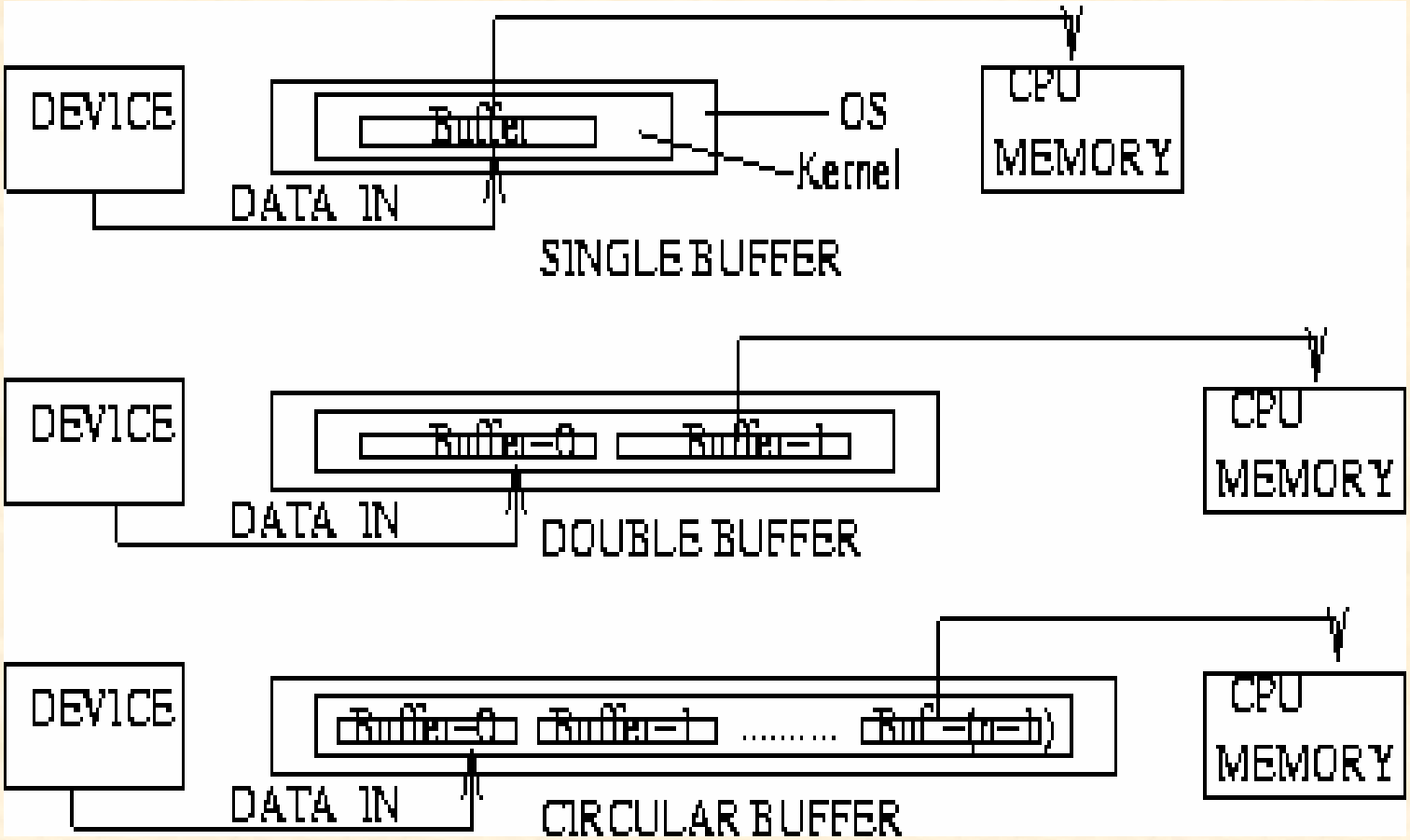
Kernel accesses filled out buffers in the same order that these are filled out.

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



Additional Considerations

Spooling in Printers

Consider a printer connected to a machine and *several users* wanting to use it.

To avoid *print clashes*, all the print requests are

SPOOLED and thus the requests are queued.

OS *maintains* and *schedules* all print requests.

We can examine *print queue status* with *lpq* and *lpstat*

commands in Unix.

Additional Considerations

Clocks and Management of Time -1

Clocks : CPU has a system clock. OS uses this clock to provide a variety of system and application based services such as :

- Maintaining time of day (*date command in Unix*)
- Scheduling a program run at a *specified time* during systems' operation (*cron* command)

Additional Considerations

Clocks and Management of Time -2

•Providing *over runs* by processes in preemptive scheduling

- important for real-time systems.

- Keeping track of *resource utilization* or *reserving resource use.*
- Performance related measurements (like *timing IO, CPU activity*).

Additional Considerations

Identifying Devices

Addressing a device :

•Most OSs reserve some address space for use as exclusive addresses for devices (like DMA controllers, timer, serial ports).

•Each of them have *fixed ranges of addresses* so that *they communicate with the right ports of data.*

Additional Considerations

Caching -1

- A cache is an *intermediate level fast storage*.
- Caches are regarded as *fast buffers*.
- Caching may be between *disk* and *memory* or *memory* and *CPU*.
- It helps in overcoming the *latency during an instruction fetch*.
- It helps in *higher locality of reference* when used for data.

Additional Considerations

Caching -1

•As for main memory to disk caches, one use is in *disk rewrites*.

•This technique is used almost always to *collect all write requests* over a short period of time and subsequently it is actually written into disc.

•Caching is always done to *enhance the performance of systems*.

Additional Considerations

IO channels : an IO channel is primarily a small computer to basically handle IO from multiple sources - *smoothes out IO traffic*.

OS and CDE : OS provides several terminal oriented facilities for operations in a *Common Desktop Environment*.

IO kernel provides all *screen management functions* within the frame work of a CDE.

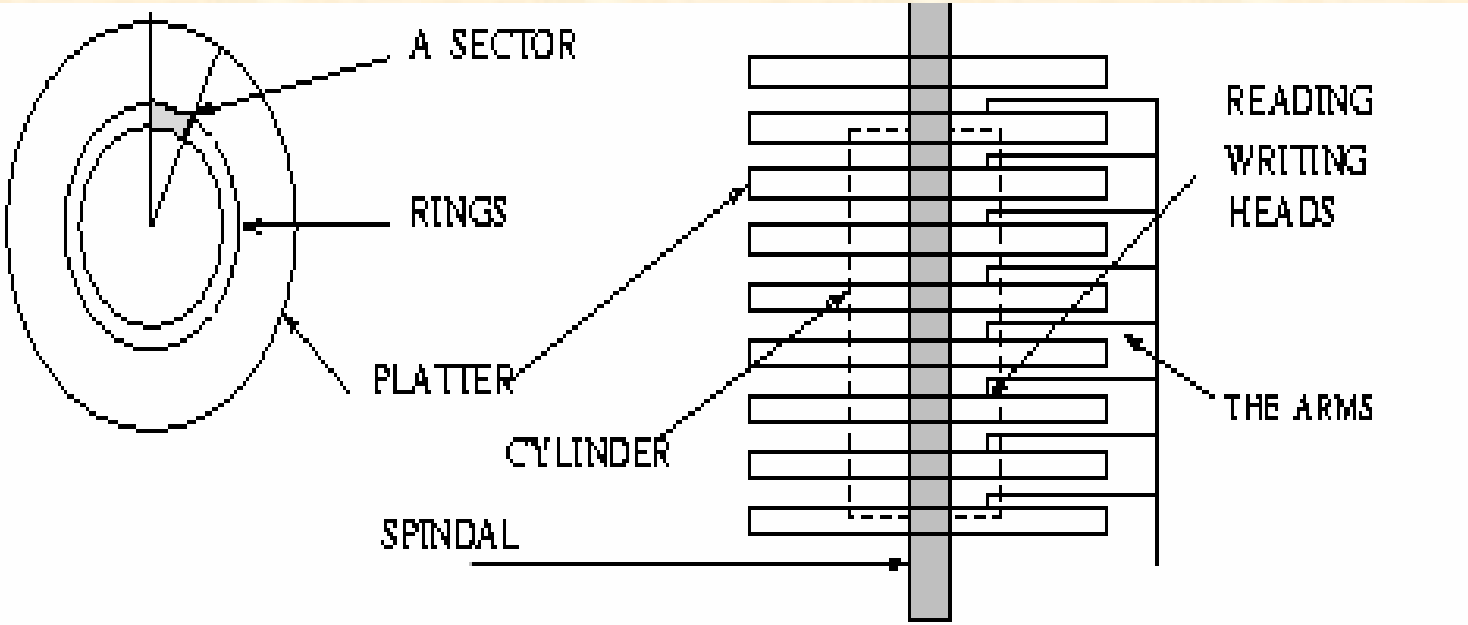
Motivation for Disc Scheduling

Primary memory is *volatile* and secondary memory is *non-volatile*.

Primary memory *loses its information* when power goes off; unlike secondary memory.

The most common secondary storage is a *disc*.

Information Storage
Organization on Discs - 1



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 - 2

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 -1

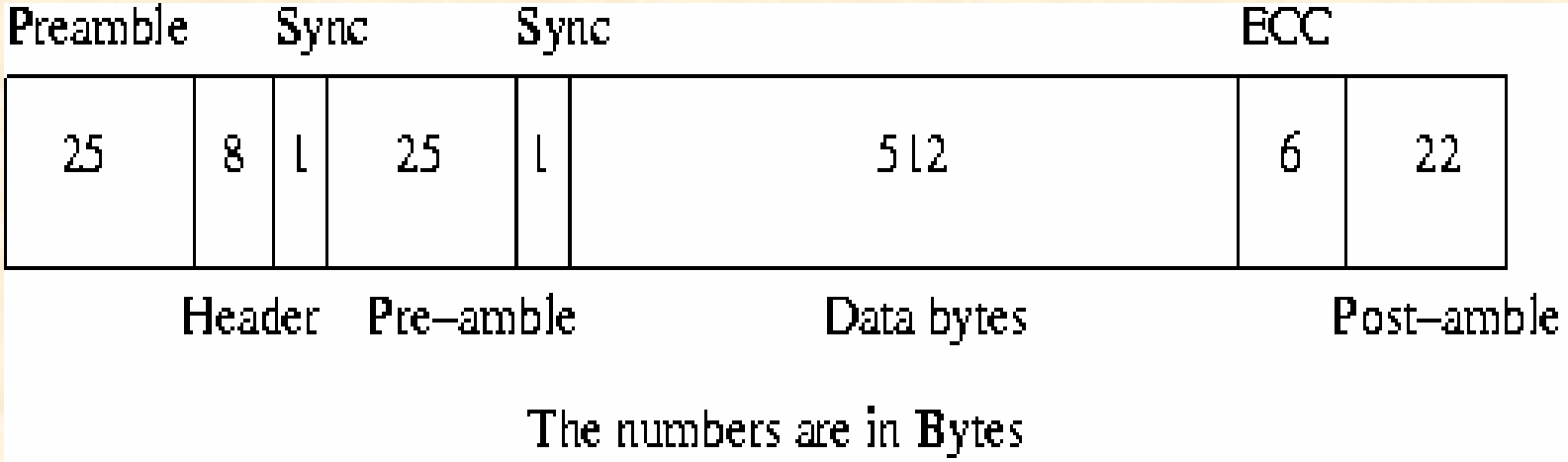
Rotational delay is due to the waiting time for a sector in rotation to come under the read or write head.

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 -1

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 -1

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 -1

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

The service would be in the sequence :

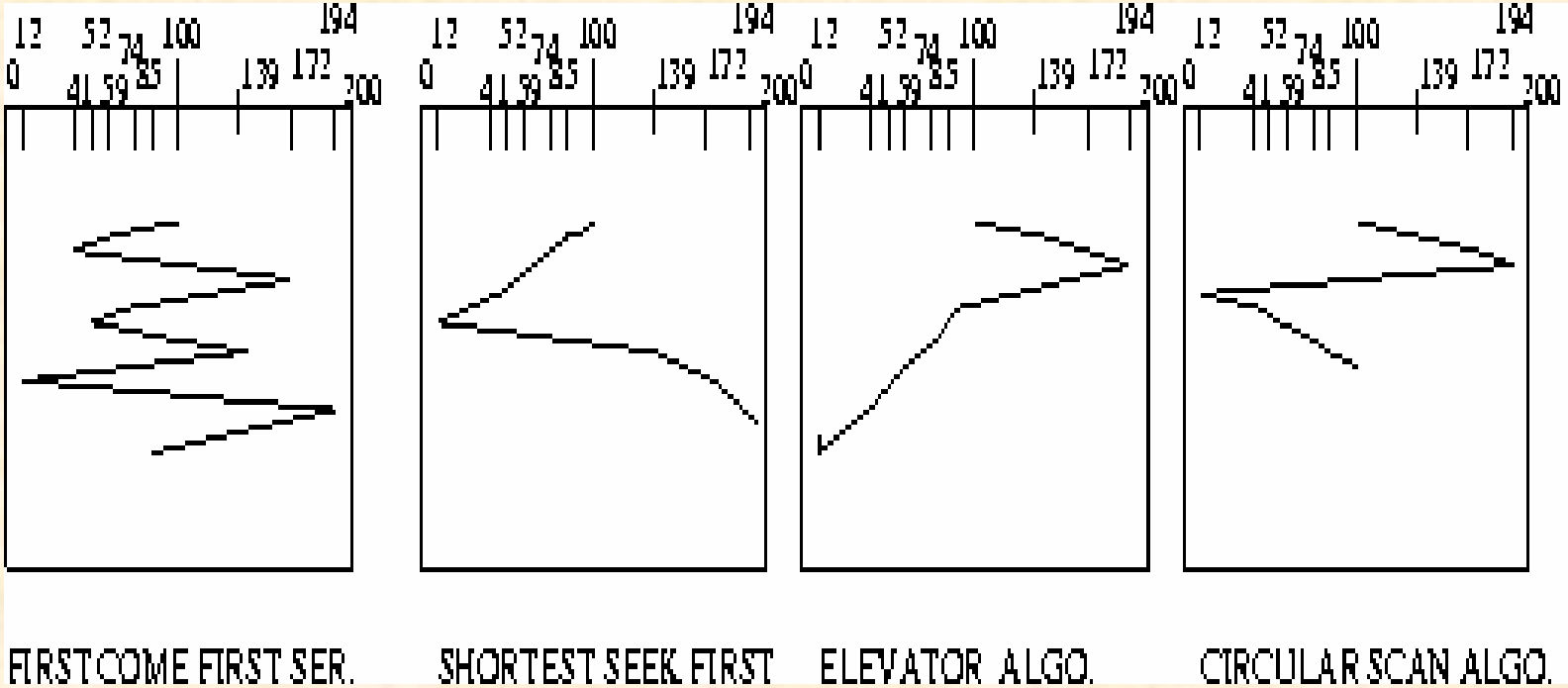
59, 41, 172, 74, 52, 85, 139, 12, 194 and 87.

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

Comparison of Policies -2

Disc Scheduling Policies

Assume that the arm is located at cylinder number 100.



Comparison of Policies -3

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

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

Elevator 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 -4

Circular Span : 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*.