OPERATING SYSTEM

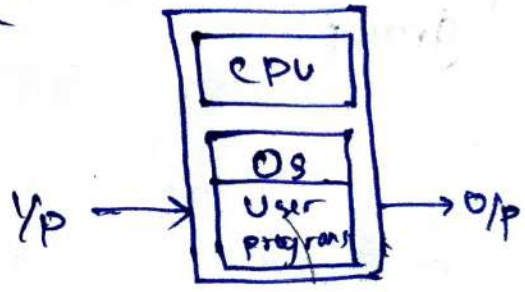
- Operating system is a system software which acts as an interface between user and computer. Purpose of an OS is to provide an environment in which a user can execute programe in convinient and efficient manner.

- Goals of OS
  - Primary : Convinient / User friendly
  - Secondary : Efficient
    - More CPU utilization
    - Less process starvation
    - Higher priority job execution

- Functions of OS

Process Management	Resource Management
File Management	CPU Management
Network Management	Security and Protection

- Evolution of OS
  - In starting mainframe computers
    - → Common input and output devices were tapes and card readers.
    - → Memory was a really big thing that time, these mainframe computers consisted of only CPU and a small memory unit which used to store OS and other user programs. They were not interactive like today.



BY SHIVKARAMAN

* Since these systems were not interactive, the job/process which was to be executed should contain all the required data and inputs in prior i.e before executing. So user creates a job which consists of program, input and control instructions.

* I/p job is given in form of punch cards and o/p is also obtained in form of punch card.

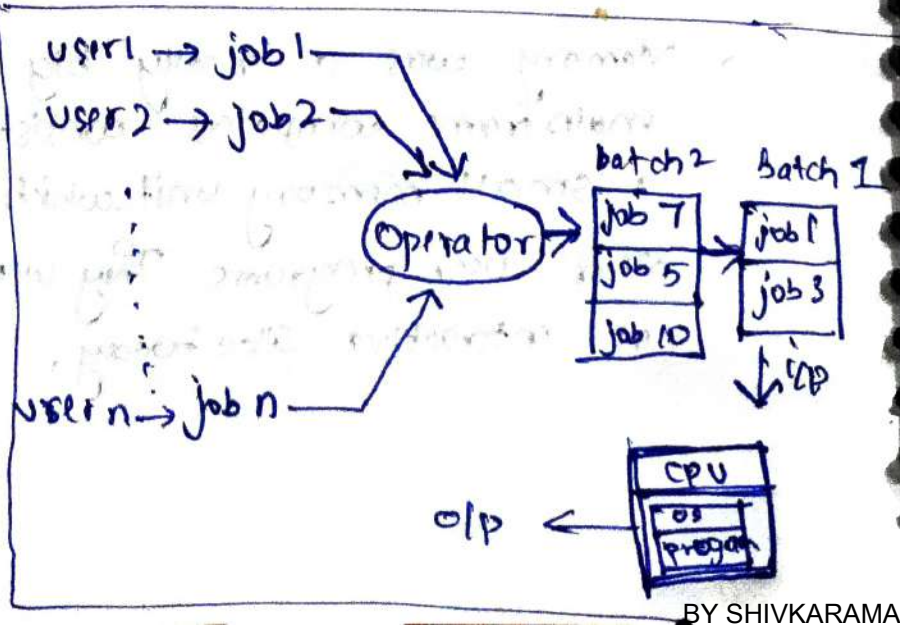
* OS was very simple and was always present in memory and major task was to switch from one job to another.

Disadvantages

* Very slow
* Very limited memory (not even in kb's)
* No speed matching b/w i/p and o/p
* No hierarchy in memory.

1 Batch OS

* Jobs with similar needs are batched together and executed through the processor as a batch/group.



BY SHIVKARAMAN

Advantages

A job gets assigned to CPU only when execution of previous job completes

Disadvantage

→ Memory limitation

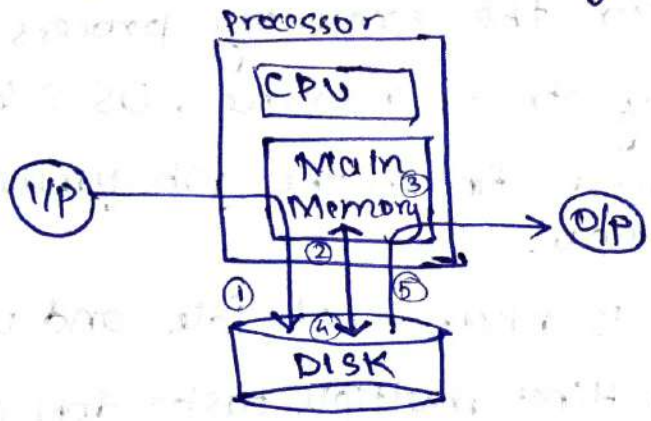
→ Interaction of i/p o/p devices directly with CPU

→ Inneficient CPU utilization

2 Spooling

→ Simultaneous Perepheral Opirations Online (SPOOL)

→ Memory was put in hierarchy → Primary memory
Secondary memory



→ Now CPU interacts directly with main memory and I/P O/P devices interact with only secondary / disk memory.

→ I/P O/P devices are relatively very slow compared to CPU

→ In spooling, the data from i/p o/p devices are stored directly into secondary memory. CPU interacts with ~~main~~ secondary memory via main memory.

→ Now I/O operations and CPU operations can be performed simultaniously i.e overlapping of i/o operations of one job with execution of another job

Advantage

- No interaction b/w i/p o/p device and CPU directly.
- Better CPU utilization

Disadvantage

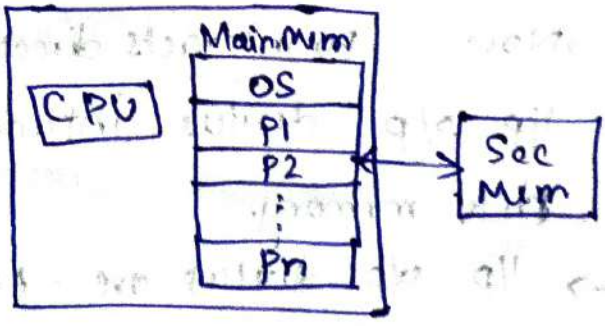
- Multiprogramming is not get possible. i.e CPU is idle when a process / job is waiting for some other resource.

3 Multiprogramming OS

* Main memory consists of jobs waiting for CPU.
* OS selects one of the process and assign it to CPU. Whenever the executing process needs to wait for any other resource, OS selects other ~~resource~~ process from the job queue and assigns it to CPU.
* This way CPU is never ~~kept~~ idle and user gets flavour of getting multiple tasks done at a time.

Advantage

* Max CPU utilization
* May be extended to multiple users



Disadvantage

* Difficult Scheduling algorithms
* Main memory management is required.

4 Multitasking OS

* Multitasking is multiprogramming with time sharing.

BY SHIVKARAMAN

→ Only one CPU but switches between jobs is so quick that the user can interact with each program as it runs. It gives an illusion that all programs are running at same time.

5 Multiprocessing OS

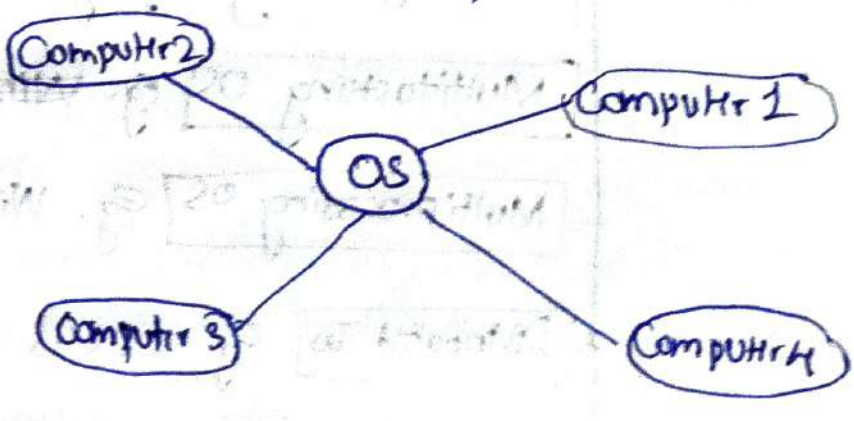
→ Two or more CPU's in a single computer
→ This increases the throughput of the computer as multiple processes can be executed parallely by different processor.

Advantages

- Increased reliability, 1 CPU fails, other can work
- Better throughput
- Less process starvation.

6 Distributed OS

- OS manages different branches of computers, it contains >=1 CPU's, >=1 Main memory, >=1 GPU



* It manages ~~and~~ multiple computers connected via ~~Ethernet~~ network.

* They are "LOOSELY CONNECTED AUTONOMOUS" interconnected computer nodes.

7 Real Time OS (RTOS)

RTOS are used in environments where large no of events, mostly external to the computer system, must be accepted and processed in short period of time with nearly no error.

Eg. Airline Traffic Control system (ATCS), robots etc.

Types of OS

- Single Process OS — Eg: MS DOS
- Batch OS — Eg: ATLAS
- Multiprogramming OS — Eg: THE, Dijkstra
- Multitasking OS — Eg. ~~Windows~~ CTSS, MIT
- Multiprocessing OS — Eg. Windows
- Distributed OS — Eg LOCUS
- RTOS — Eg ATCS

BY SHIVKARAMAN

• Difference Between RAM and ROM

1 RAM (Random Access Memory)

- Volatile Memory (Data lost when power is cut)
- Information stored in RAM can be checked with help of BIOS

Characteristics :
* Much faster Read/write operations
* Widely used in normal operations of a computer for starting and loading OS and applications.

Types

1) DRAM : Dynamic Ram → Must be continiously refreshed otherwise all contents are lost
2) SRAM: Static Ram → No refreshing needed
3) Synchronous DRAM (SDRAM) - Runs at very high clock speed.
4) DDR - Double Data Rate Ram → provides synchronous Random Access Memory

2 ROM (Read only Memory)

- Non volatile Memory
- Computer manufacturer decides the information of ROM and it is permanently stored in memory. It cannot be overwritten by the user.

Characters :

* Data can be written only once
* ROM chips used in start-up process of a modern computer
* It can store only several MB of data

Types :

1) PROM (Programable ROM) - Data written/programed using particular device
2) EPROM (Erasable PROM) - Data can be erased by exposing to UV rays.
3) EEPROM (Electrically Erasable PROM) - It stores and deletes instructions on a special circuit

Parameter	RAM	ROM
Usage	Allows comp to read data quickly to run applications	Stores allpication needed to boot the computer initially
Volatality	Volatine	Non-Volatile
Accessability	Easily accessed	Processor cannot directly access ROM. Data is first copied to RAM and then executed by computer
Read/Write	Both R/W	Only Read
Storage	Used to store temporary info.	Used to store permanent information
Speed	Access speed is faster	Slower
Cost	High	Comparitively low
Chip Size	In GB's	In MB's
Structure	Rectangular structure inserted into mother board	Storage medium that stores permanent data in PC

BY SHIVKARAMAN

- ## SRAM V/S DRAM

1 SRAM

It is a type of Semiconductor memory that uses Bistable Latching circuit to store each bit. Data is stored using six transistor memory cell.

MOSTLY USED AS CACHE MEMORY FOR CPU

Faster than other RAM types

Characterstics :
* Faster
* More expensive than DRAM
* Takes more space than DRAM
* Usage: Level 1 or Level 2 CACHE
* Low power consumption

2 DRAM

* It is a type of RAM which stores each bit of data in separate capacitor within particular IC.
* It is constructed using capacitors and few transistors
* Capacitor charged → bit 1
  discharged → bit 0

MOSTLY USED AS MAIN MEMORY IN COMPUTERS

Characterstics
* Cost-effective
* Slower compared to SRAM
* High power consumption
* Requires to refresh.

BY SHIVKARAMAN

SRAM	DRAM
Slower access time ie faster speed	Higher access time ie slower speed
Costlier	Cheaper
Needs constant power supply but consumes less power	Require more power as info is stored on capacitor
Low package density	High package density
Uses transistors & latches	Uses capacitors and few transistors.
Cache memory is application of SRAM	Mostly found as main memory in computers.
Works on principle of changing direction of current through switches	Works on holding the charges.

- PROM V/S EPROM V/S EEPROM

1. PROM (Programmable ROM)

* It is a memory chip and it is possible to program it only once after creation. Information written becomes permanent once programmed. Data cannot be erased / deleted.

USED IN Earlier computer BIOS system.

* After creating PROM, all bits are 1. the required bits are made 0 by burning.

BY SHIVKARAMAN

2 Erasable PROM (EPROM)

* We can erase and reprogram an EPROM without replacing it. It is possible to erase and write to it by exposing it to UV rays.

3 Electrically Erasable PROM (EEPROM)

It is a memory chip that we can erase and reprogram using electrical charge. It consists of collection of floating ~~plates~~ gate transistors.

Difference

PROM → Read only, no modification

EPROM → can be erased and reused

EEPROM → User modifiable i.e can be erased and reprogrammed through normal electrical voltages.

- **Virtualization V/s Containerization**

Data centers consists of large number of servers. Not all are active at the same time. Say in case traffic is directed to same set of servers more, those servers get busy. The other servers are less loaded, or they even turn totally inactive, ~~thus~~ thereby wasting power, maintanance costs and other allied resources.

To reduce this overhead, virtualization and containerization were introduced.

1 Virtualization

* It is a technology that can simulate your physical hardware and represent it as a separate machine.
* Virtualization is the process of partitioning a physical server into multiple virtual servers. The process of partitioning out using software is called HYPERVISOR. After partitioning, the virtual servers act and perform just as physical server.
* It means basically using same hardware setup more efficiently.
* this is HARDWARE LEVEL VIRTUALIZATION.
* Hypervisor → It is a Virtual Machine Manager (VMM)

Eg. VM Vare, Virtual Box.

Instead of dedicating 1 server and 1 OS to execute 1 dedicated task, the same server is partitioned as 1 server and 2 OS which allows indipendent tasks to run on both of them

2 Containerization

When code is developed in a specific computing environment and transferred to a different environment there is a high possibility of code to result in bugs and error due to missing dependencies, libraries or configuration setting files.

Eg. If a windows based system is ported to linux based system.

* If dependencies are provided along with the code, the potential bugs and errors can be eliminated.

* Containerization is the process of building application code along with the libraries, configuration files, and dependencies for application to run cross-platform. Code is written once and is capable of executing anywhere.

* Here the OS is virtualised ie SOFTWARE LEVEL VIRTUALIZATION

Eg. Docker

What is Container?

A single package of software built using containerization technology is called container.

Key-factor	Virtualization	Containerization.
Technology	One physical machine has multiple OS's and appears as multiple machines.	Application developed in a host environment with same OS and same machine executes bug-free on all cross-platforms
Startup time	Higher than containers	Less
Speed	VMS being a virtual copy of host server, VM's are resource heavy, hence SLOWER	Containers ~~virtualize the OS~~ are faster
Size	Larger	Smaller

Component being virtualised.	Hypervisors virtualize the hardware (use for same hardware)	Containers virtualize the OS (use the same OS)
Cost	Higher	Lower
Benefits	IT Business	Software Developing

- **BIOS v/s UEFI**

Both UEFI and BIOS are low-level s/w that starts when you boot your PC before boting your OS, but UEFI is a more modern solution, supporting larger hard drives, faster boot times, more security features, GUI and cursors.

[1] **BIOS** (Basic Input Output System)

* It is a low-level s/w that resides in a chip on the computers motherboard.
* BIOS loads up when your computer starts and is responsible to wakeup your computer hardware comp; ensure they are functioning properly and ~~boots the~~ runs the boot loader that boots the OS.

1) BIOS goes through a POST (Power-On-Self-Test) before booting OS. It checks to ensure your hardware configuration is valid and working properly.

2) After POST finishes, BIOS looks for MBR (Master Boot Record) and uses it to launch boot-loader which boots the OS.

Limitations

* It can only boot from drivers of 2.1TB or less. 3TB drivers are common now and computer with BIOS cannot boot them.
* It has trouble initializing multiple hardware devices at once which leads to slower boot process.
* It runs in 16-bit processor mode and only has 1MB size to execute in.

2 UEFI

* No limitations as in BIOS. It can boot from drivers of 2.2TB or larger
* It can run in 32-bit or 64-bit mode and has more address space than BIOS ⇒ Faster Boot process.

- GPT v/s MBR

MBR → Master Boot Record
GPT → GUID Partition Table.

We use GPT or MBR to partition a disk

MBR and GPT are two different ways of storing the partitioning information on a drive. This information includes where partition begins and where it ends on physical disk.

1 MBR

* It is a special boot sector located at beginning of a drive. It contains boot loader for the installed OS and information about drive's logic partitions.

Limitations
* MBR works with disks upto 2TB.
* It supports upto 4 primary partitions. If you want more you have to make one of your primary partitions and create logic partitions in it.

2 GPT

* It is associated with UEFI
* GPT replaces MBR's partitioning systems with Partition tables where every partition on the drive has "globally unique identifier" or GUID.
* GPT has no limitations as MBR
* Windows allows upto 128 partions on a GPT
* On MBR, partitioning and boot data is stored in one place. If data is overwritten, you're in trubble. In GPT, multiple copies of data are present accross the disk, it can recover if data is corrupted.

- Important Terminology

1. Compiler : It is a software that typically takes a high level language code as input and converts it into low level language at once. It lists all errors if the input code doesnot follow the rules of the language.

2. ~~Loader~~ Linker

It is a program in system which helps to link object modules of a program into a single object file.

3. Loader

Loader is a program of OS which loads the executable from the disk into the main memory for execution. It allocates memory to executable module in main memory and transfers control to the beginning instruction of the program.

4. Assembler.

Assembler is a program for converting instructions written in low-level assembly code into relocatable machine code and generating along information for the loader.

5 Interpretor

Software by which conversion of high level ~~language~~ instructions is performed line by line to machine level language other than compiler and assembler is called interpretor.

6 System call

System call is a programatic way in which a computer program requests a service from the kernel of OS.

7 Kernel

Kernel is a part of OS that manages operations of a computer and hardware. It basically manages operations of memory and CPU time.

* Kernel acts as a bridge between applications and data processing performed at hardware level using Inter Process Comm and System call.

8 Shell

A shell is a special user program which provides an interface to user to use OS services. Shell accepts user readable commands from user and converts them into something which kernel can understand.

BY SHIVKARAMAN

Shell

- Command Line Shell
  - Command Line Interface in windows, terminal in Ubuntu
- Graphic Shell
  - GUI in windows.

9 Booting

Procedoring of starting computer by loading the kernel is known as booting the system. Here it needs a special program stored in ROM to do this job known as bootstrap loader.

- Diffrence b/w Multi-tasking, Multi-threading, Multiprogramming

Multi-Tasking	Multi-Threading	Multiprogramming	Multiprocessing
Execution of multiple programs simultaniously	Process is divided into several different subtasks called as threads which has own path of execution	Execution of multiple processes using context switching to enhance CPU utilization	Contains two or more CPU's in a single computer.
Concept of more than one process being context switched for each quantum time.	Concept of more than 1 thread. Threads are context switched	Processes are context switched when one process waits for other resources.	Multitasking with diffrent CPU's.
1 CPU	>=1 CPU (Better to have more than 1 CPU)	1 CPU	>1 CPU
Isolation and memory protection	No isolation & memory protection as all threads are of same process	Isolation & Mem protection	Isolation and memory protection

Thread Scheduling

Threads are scheduled for execution based on their priority. Even though threads are executing within the runtime, all threads are assigned processor time slices by the OS.

Thread Context Switching	Process Context Switching
OS saves current state of thread & switches to another of same process.	OS saves current state of process and switches to another process by restoring its state.
Doesnot include switching of memory address space. (But program counter, registers and stack are included)	Includes switching of memory address space.
Fast switching	Slow switching.
CPU cache state is preserved	CPU's cache state is flushed.

cache is refreshed when CPU switches process coz the other process cannot use resource of this process (Process protection) and also the other process may not use the same resource. So since threads belong the same process ⇒ CPU not refreshed.

BY SHIVKARAMAN

- Differenca b/w Monolithic and Micro Kernel

1 Monolithic Structure

It is a very basic Operating System in which file management, memory managem ent, device management and process management are directly controlled within the kernel. Each component of the OS is contained within the kernel.

* A process runs completely in single address space in monolithic kernel.
* Monolithic kernel is a static single binary file.

Advantage : Fast Processing ie high performance.

Disadvantage : * Crash insecure → If any service fails in the kernel, it leads to failure of entire system.

* Entire OS needs to be modified by user to add any new service.
* Bulky in size and needs high memory space.

Eg: ~~[illegible]~~, UNIX, Linux

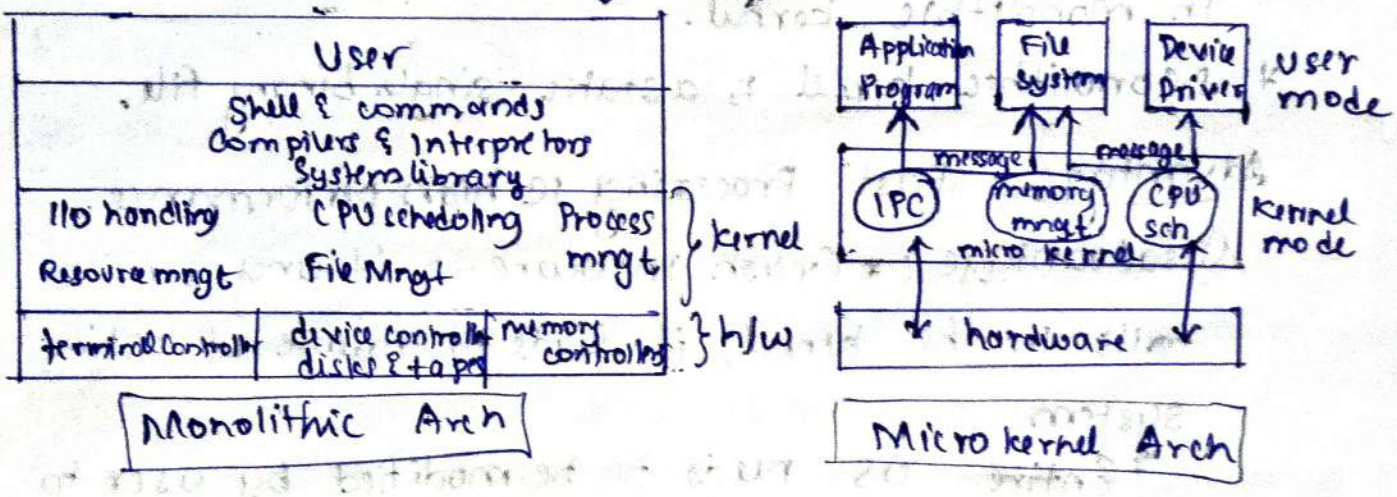
2 Micro kernel

It is an operating system which contains import parts like IPC, basic schedulers, basic memory handling, basic I/O primitives etc, are put into the kernel. Communication happens via message passing

Others are maintained as server process in User Space.

Advantages :
* Crash Resistent
* Portable
* Smaller Size.
* More stable.

Disadvantages
* Overhead in switching b/w user & kernel mode
* Slower processing due to additional message passing



- **Windows Architecture is more monolithic and not microkernel**

This is because the kernel mode protected memory space is shared by OS and device driver code.

According to defination of Microkernel →
OS components and drivers run in their own private address space layed in primitive microkernel.

BY SHIVKARAMAN

Which is not the case in Windows architecture.
All kernel components (IPC, OS, drivers, CPU sch) live in common address space. Therefore there is no protection b/w OS and drivers.

But it still has some attributes of microkernel

- Kernel-mode components dont reach into one another's data structures
- They use formal interfaces to pass parameters and access/or modify data structures.

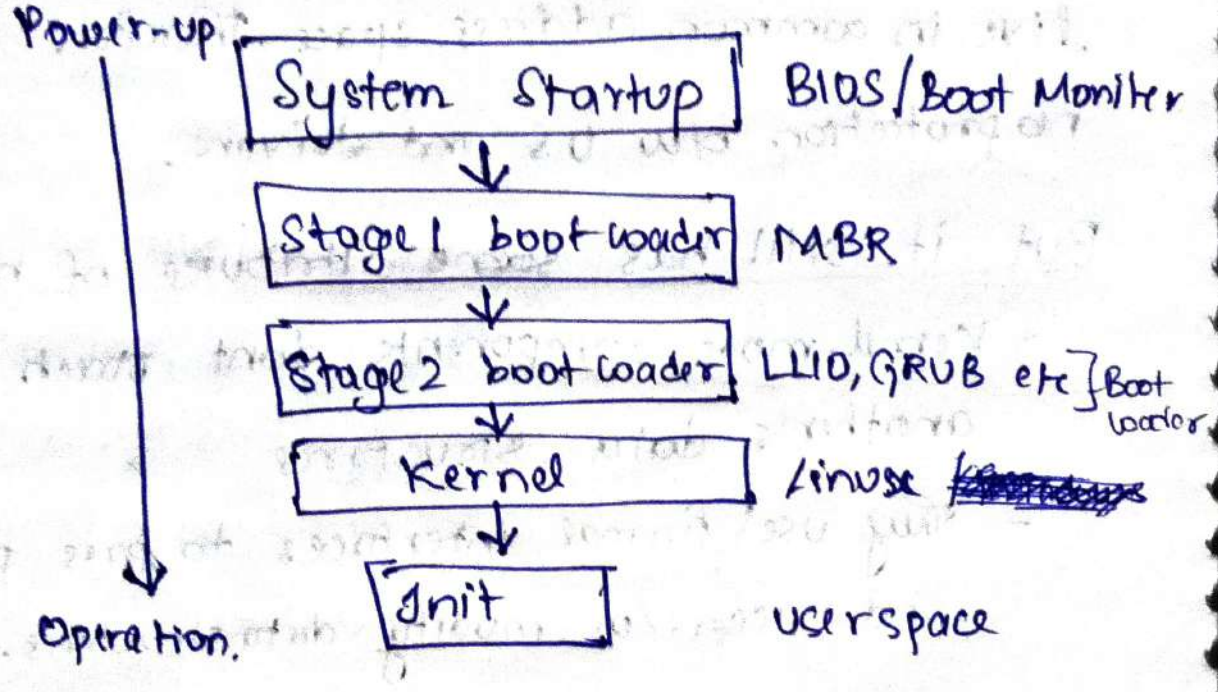
Therefore Windows is neither monolithic nor microkernel. It is a HYBRID. It is neither monolithic where all services run in kernel mode nor microkernel where everything runs in user space

- What happens when you turn on a computer

BIOS loads up when your computer starts and is responsible to wake up your computer hardware components ensuring they are functionin

BY SHIVKARAMAN

properly and runs the bootloader that runs the OS.



1 POST (Power On Self Test)

POST happens each time ~~when~~ you turn on the computer.

- It initialises various hardware devices and ensures all devices operate smoothly.
- POST first checks the BIOS and then checks the CMOS RAM. (Complimentary Metal-oxide s()
- If no errors, it continues to check CPU, other hw devices, such as secondary storage, videocard etc.
- If some errors are found then error message is displayed on screen or number of beep sounds are heard → POST beep codes

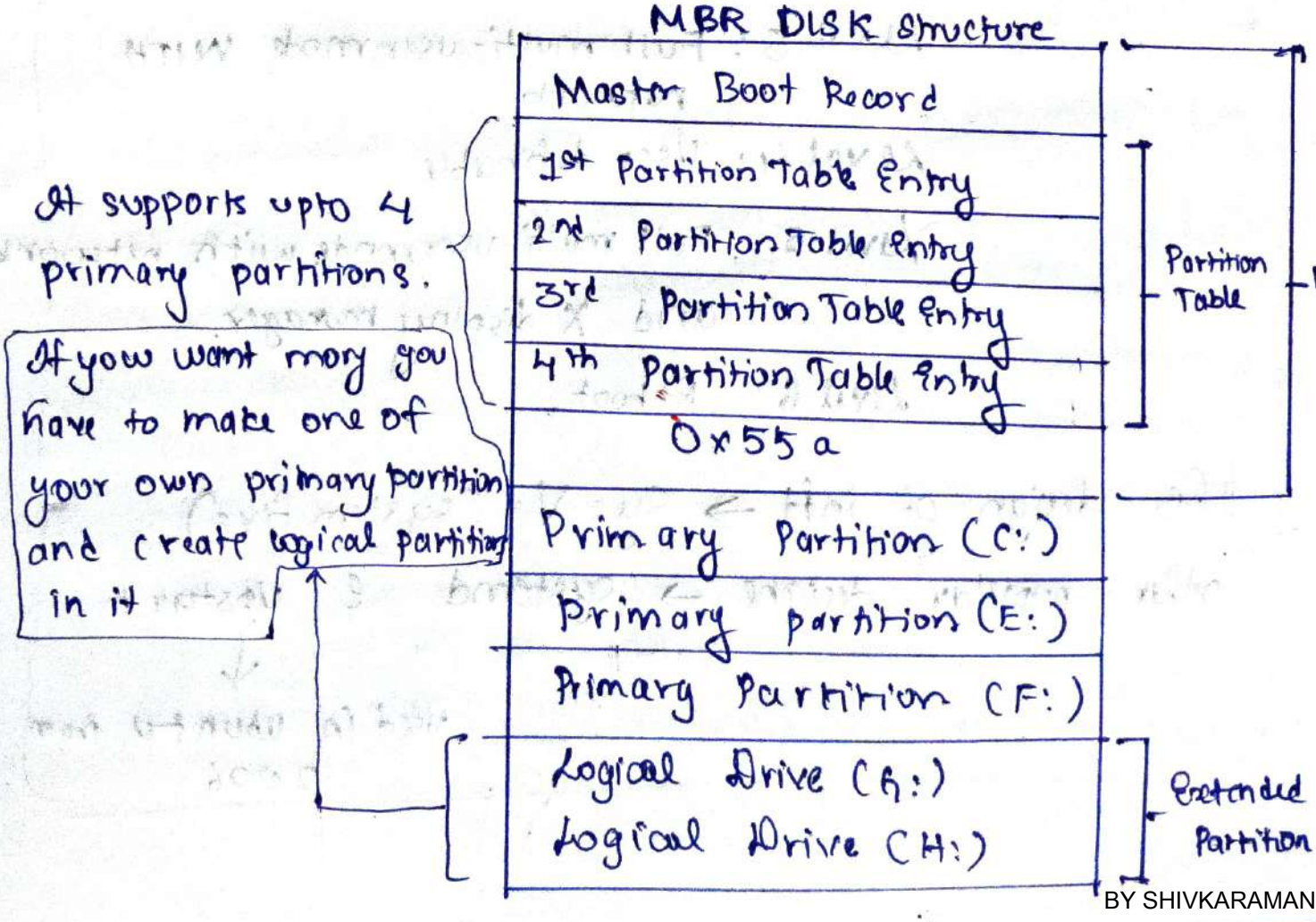
BY SHIVKARAMAN

2 Master Boot Record (MBR)

It is a special boot sector at the beginning of the disk. It contains rest of the code that loads rest of OS, known as boot loader.

Bootstrap loader is stored in computer EPROM, ROM or other non-volatile memory. When computer is turned on, it first runs POST and then searches for MBR [which is present in first sector, first head, first cylinder (cylinder 0, head 0, sector 1)] which contains partition information and bootloader. Bootloader now loads and runs the OS.

MBR DISK structure



BY SHIVKARAMAN

Boot loader is called Grub or LILO in Linux

3 Init

init is last step of kernel boot sequence.

It looks for file /etc/ inittab to see if there is an entry for initdefault. It is used to determine initial run level of system.

Some of Run levels:

Level 0: System Halt

Level 1: Single user mode

Level 2: Full multi-user mode without network

Level 3: Full multi-user mode WITH network

Level 4: User definable

Level 5: Full multi-user mode with network and X display manager.

Level 6: Reboot.

This design of init → Sys V - (system five)

other popular designs → systemd & upstart

↓

used in ubuntu from 2006

- Next step of init is to startup various daemons → background processes that support networking and other services.

X server deamon is one of the most important deamons. It manages display, keyboard and mouse

When X server deamon is started you see a GUI and login screen is displayed.

- **Functions of OS**

1) Process Management
   - CPU scheduling algorithms for processes
   - Suspending and resuming processes
   - ~~Produ~~ Providing mechanisms for process synchronization and process communication.

2) Memory Management
   - Allocating and deallocating memory space as per need.
   - Keeping track of which part of memory are currently being used and by which process.

3) File management
   - Creating and deleting files
   - Creating and deleting directories to organize files
   - Mapping files into secondary storage

4) IO Management

To manage and control I/O operations and I/O devices.

i) Spooling

Matching speed b/w two processes / jobs.

ii) Buffering

Managing speed mismatch ~~b/w~~ within a process

iii) Caching

Memory caching.

PROCESSES CONCEPT

- Difference b/w Process and Program

[1] Program :

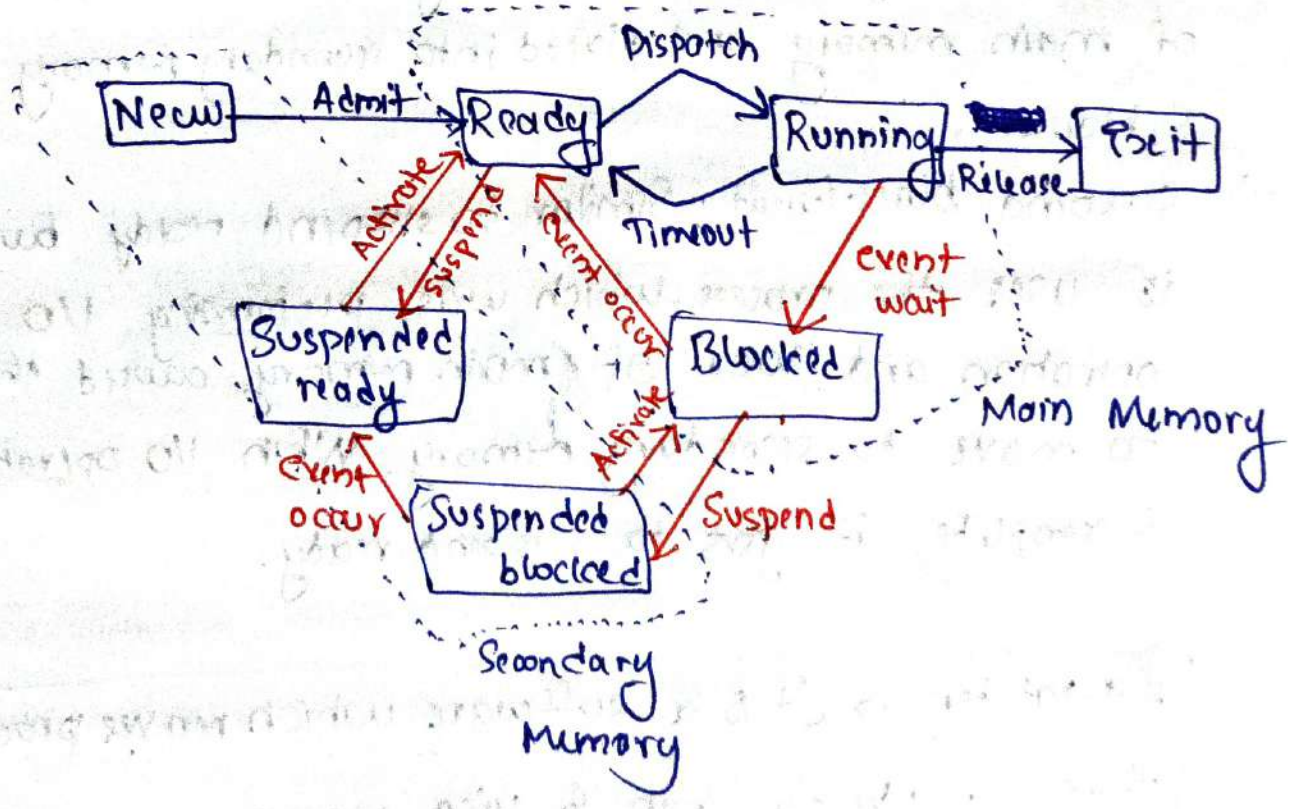
A program is a passive entity which resides in the secondary memory.

[2] Process

A process is a program which is loaded into primary memory (by loader) and is being executed by the processor.

Program	Process
It is a set of instructions written to compute a specific task	It is an instance of executing process.
Passive Entity	Active entity
Present in Secondary memory	Present in primary memory
Static Entity	Dynamic Entity
Has two logical components 1. Code 2. Data	In addition to code & data, process also requires additional info required for management & execution.

- Different states of process



1) **New**: Program present in secondary memory and is ready to be picked up by OS to create process

2) **Ready**: Program is loaded to main memory. Process is ready to run and is waiting for CPU.

3) **Run**: Executing by CPU

4) **Block/Wait**: When it requests for other services such as I/O which requires time. ~~It is paused on~~ (i.e it tries to access critical region), process is kept in wait until requested service occurs

5) **Terminated/completed**: Process is killed and PCB is deleted

6) **Suspend ready**: Process in ready state is swapped out of main memory and placed into secondary memory by scheduler.

7) **Suspend block/wait**: Similar to suspend ready but ~~is~~ uses the process which was performing I/O operation and lack of main memory caused them to move to secondary memory. When I/O operation is complete, it goes to suspend ready.

Dispatcher → It is a software which moves process from ready to run & vice versa

- **Types of Processes**

Processes

CPU bound	I/O bound
* Process is intensive in terms of CPU operations	* Process is intensive in terms of I/O operations

Best system performance can be obtained when there is a combination of CPU bound and I/O bound processes in main memory.

- **Types of Schedulers**

Schedulers

Long term / Job Scheduler	Short term / CPU scheduler	Medium term
* It selects jobs from secondary memory & loads into main memory	* It selects jobs from main memory and dispatches it to be executed by CPU	* Suspension decision is taken by medium term scheduler.
* It makes decision about how many processes should be made to stay in ready state	* It decides which process to be executed next & calls the dispatcher.	* It is used for moving a process from main memory to secondary memory
* Once a decision is taken it stays for long time, this is why it is called as long term scheduler.		
Controls degree of Multiprogramming		

- **Process Control Block (PCB)**

- **Why PCB is used?**

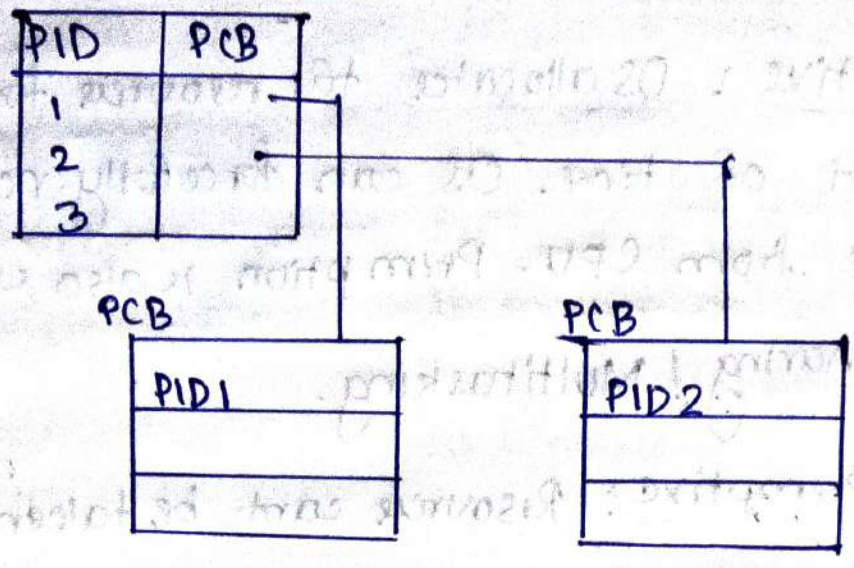
While creating a process, OS performs several operations. To identify the process, it assigns a process identification number (PID) to each process. As OS supports multi-programming it needs to keep track of all processes. For this task the PCB is used. which can be used to track process execution status.

- **PCB structure**

Each PCB contains.....

1) Pointer → It is a stack pointer which is required to be saved when process switches its states.
2) Process State: Current state of process
3) Process Number: PID of process
4) Program Counter: Contains address of next instruction to be executed within the process
5) Register: CPU registers
6) Memory limits: Information about memory management system. (Page tables, segment tables etc)
7) Open file lists: List of files opened for process

Process Table : It is an array of PCB's



- **How does a process look like in memory ?**

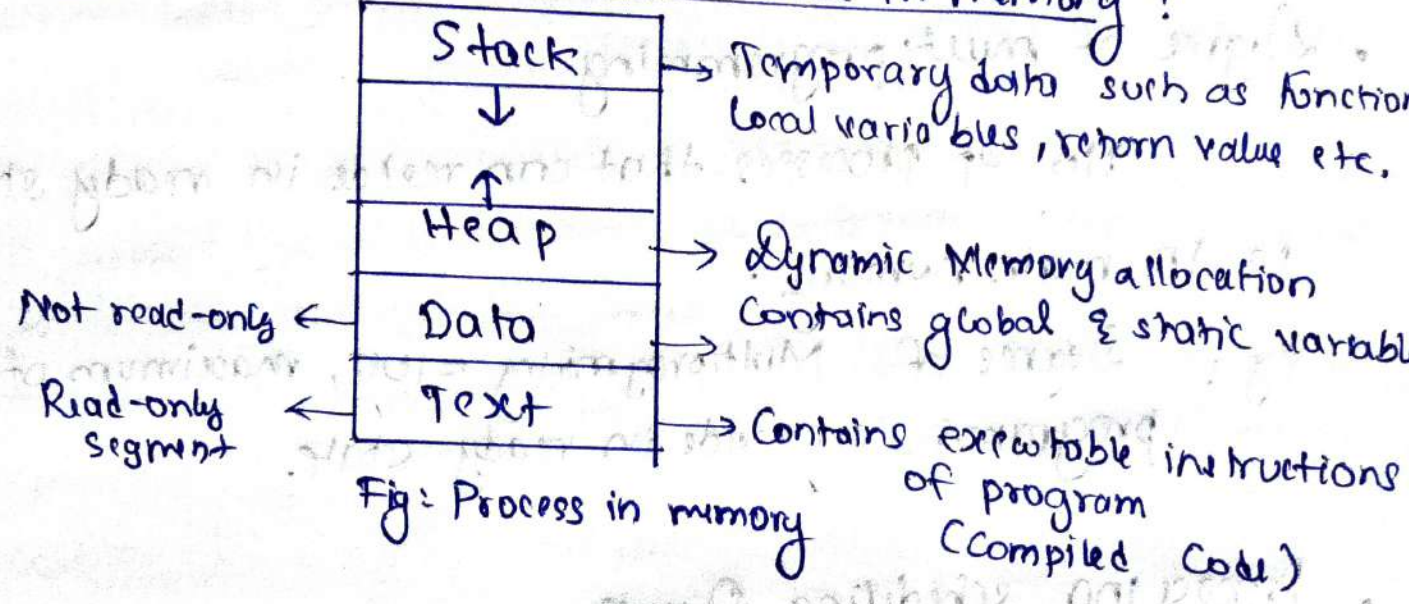


Fig: Process in memory

- **Process Scheduling**

It is activity of process manager that handles removal of running process from CPU and selection of another process ~~from~~ on basis of a particular strategy.

It is an essential part of Multiprogramming.

- Category of Multiprogramming

1) Premptive : OS allocates the resource for a fixed amount of time. OS can forcefully remove the process from CPU. Prim ption is also called as time-sharing / Multitasking.

2) Non-Premptive : Resource cant be taken from a process until the process completes execution.

- Degree of multiprogramming

No of processes that can reside in ready state ie in main memory.

Eg : Degree of Multiprogrmming = 100, maximum of 100 programs can reside in ready state.

- Processing scheduling Queues

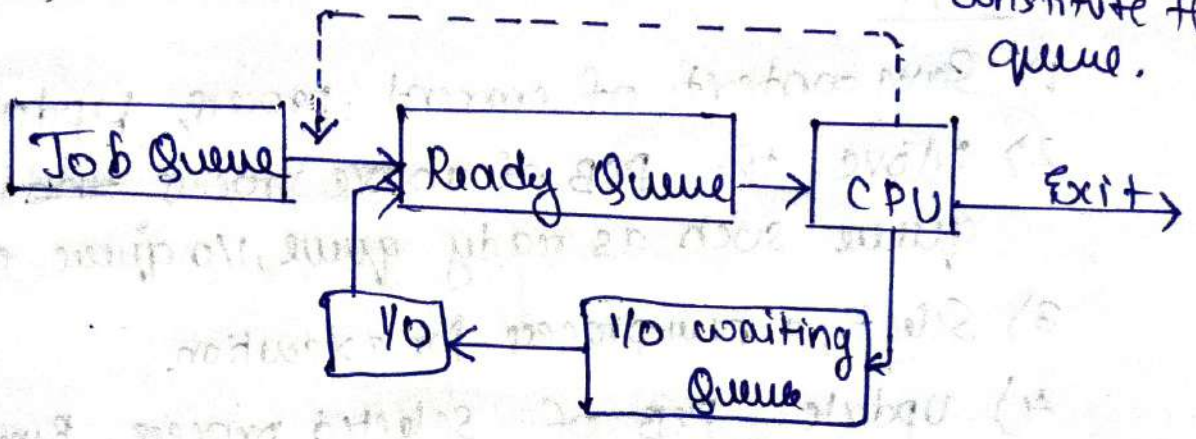
OS maintains all PCB's in scheduling queues WHN state of a process changes, PCB is unlinked from current queue and moved to its next ~~queue~~. state queue.

Scheduling Queue

Job Queue
* It keeps all processes in the system

Ready Queue
* Keeps set of all processes residing in main memory, ready and waiting to execute

Device Queue
* Processes that are blocked due to ~~unavail~~ unavailability of I/O device constitute this queue.



OS can use different policies to manage each queue (FIFO, Round Robin, Priority etc).

Types of process Schedulers

Scheduler

- Long term / Job scheduler
- Short Term / CPU scheduler
- Medium Term Scheduler

- Context switching

It is a mechanism to store and restore state ~~of~~ or content of CPU in PCB so that process execution can be resumed from same point at a later time. This ~~use~~ enables multiple processes to share a single CPU.

Steps

1) Save context of current process, Update the PCB
2) Move the PCB of above process ~~and~~ into relevant queue such as ready queue, I/O queue etc.
3) Select a new process for execution.
4) Update PCB of selected process. Remove the PCB from its current queue. ~~[illegible]~~ and update its state to running
5) Run the process.

- Context switching triggers

1) Multitasking : context switching takes place for every quantum time
2) Interrupt : When there is an interrupt
3) User & kernel mode switching

- Cost of Context switching

It leads to overhead because CPU is kept idle during context switching.

• Inter Process Communication

Processes can be of two types

Independent	Co-operating
* It is not affected by the execution of other processes	* They can affect or be affected by execution of other processes.

Inter Process communication is a mechanism that allows processes to communicate with each other. This increases computational speed, information sharing, convinience and modularity.

1 Shared Memory Systems

* Region of memory that will be shared by the co-operating process is established. Processes share this memory to read/write into it.
* Shared memory is faster once it is set up.
* This is generally preffered when large amount of information must be shared quickly on same computer
* Processes should take care that the two processes will not write the data to shared memory at same time.
* Form of data and position of creating shared memory is decided by process.

Eg: Producer - Consumer Problem. → There are two processes which share memory called as buffer.

Two types of Producer - consumer
- P-C problem with unbounded buffer
- P-C problem with bounded buffer

Buffer

Unbounded Buffer	Bounded buffer
* No practical size on size of buffer	* Fixed buffer size
* Producer keeps producing and consumer consumes (If buffer is empty, consumer waits)	* Producer produces when buffer is not full.
	* Consumer consumes when buffer is not empty

2 Message Passing

* Processes donot share common address space to communicate with each other.
* This type of IPC is used in distributed systems
* The processes use system calls for sending and recieving messages.
* A communication link must be established between the processes before message is passed.
* A message contains 2 parts
  - header → msg type, src id, dst id, msg len etc
  - body → message.

Communication

- Naming (Direct & Indirect)
- Synchronization (Synchronous and Asynchronous)
- Automatic/Explicit buffering

1 Naming

- **Direct Communication**

  * Sender and reciever must know each others name.

  send (P, msg) → send msg to process P

  recieve (Q, msg) → Recieve msg from process Q

Properties

* Links are established automatically
* Link is established with exactly one pair of processes

Type of Addressing

Symmetric	Asymmetric
* Both sender and reciever know each other - send (P, msg) - recieve (Q, msg)	* Only sender names the reciever - send (P, msg) - recieve(id, msg)

- Indirect communication

  * Mailbox/ port is used to send/ recieve message
  * Mailbox is an object to which message can't be sent and recieved. It has a unique ID. Two processes can communicate if they have a shared mailbox.

    - send (A, msg) → send msg to mailbox A
    recieve (A, msg) → recieve msg from mailbox A

  property

  * Link is established b/w process if there is shared mailbox
  * Link can be associated with more than two processes

2 Synchronization

Sending and recieving messages can be either synchronized / asynchronized.

- Blocking (Synchronous) send

  Sending process is blocked until message is recieved by reciever.

- Non-blocking (asynch) send

  Sends message without any waiting

- Blocking (synch) recieve

  Recieving is blocked until message is available

BY SHIVKARAMAN

- Non-blocking (asynch) recieve

Recieves message without any block. Recieved message may be void / null.

3 Automatic / Explicit buffering

When message is passed, temporary queue is created. Such queue can be of 3 capacity

1) Zero-capacity
   * Buffer size is 0
   * Message not stored in queue
   * Sender must block until reciever accepts

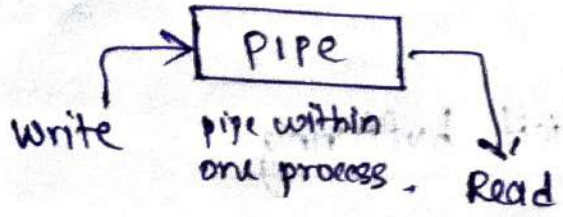
2) Bounded capacity
   * Buffer size is limited.
   * Sender must wait if buffer is full.

3) Unbounded capacity
   * Infinite capacity
   * Sender never blocks.

- IPC - pipe

Pipe is a communication medium between two or more related or interrelated processes. It can be either within one process or between child and parent processes, or between parent, child and grand-child.

Communication is achieved by one process writing into pipe and other process reading from it.



- **Zombie Process:** Process which has completed execution (by exit() sys call) but still has an entry in process table.
- **Orphan Process:** Process whose parent process has finished / terminated, but it is still running.
- **Maximum No of Zombie process a system can handle?**

Since there is entry in process table for each process (even zombie process). Since size of process table is finite, no of zombie processes a system can handle depends on size of ~~zo~~ process table. If zombie process are created in large amount, process table gets filled and program will stop without completing their task.

```
int main ()
{
    int count=0;
    while (fork () > 0) {
        count++;
        cout << count;
}
}
```

→ This program can tell the no of zombie program a system can handle.

BY SHIVKARAMAN

- 32-bit V/S 64 bit OS

32 bit OS	64-bit OS
Each register can hold 32-bits of data	Each register can hold 64-bits of data
It can access 2^{32} unique memory address space i.e 32-bit OS can work with max 4GB RAM	It can access 2^{64} unique memory address space i.e 64-bit OS can work with max 2^{34} GB RAM
It can process/calculate 32 bit data & information	It can process/calculate 64 bits of data and information

Advantages of 64-bit OS over 32-bit OS

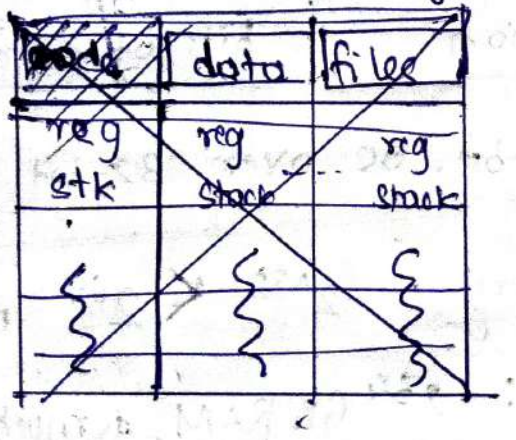
1) Addressable memory : $2^{\wedge 32} < 2^{64}$ memory address
2) Resourse usage : 2^{34} GB RAM is much faster than 4 GB
3) Performance: All calculations take place in registers, so having large registers allows performing larger calculations at the same time.
4) Compatability: 64 bit OS can run both 64-bit & 32-bit programs.

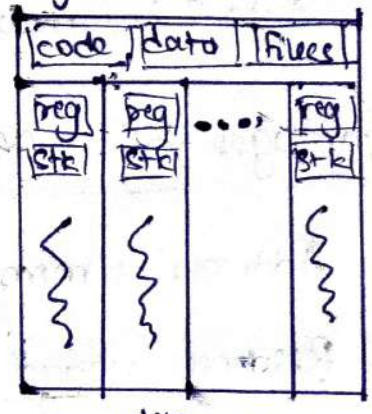
THREADS

- **What is thread?**

  * Thread is basic unit of ~~process~~ CPU utilization
  * It consists its own PC, stack, set of registers.

    PC → address of next instruction
    stack → history of execution
    register → variables
  * They are also known as light weight processes.





multithreaded process

Process	Threads
Program under execution	Segment of a process
Consumes more resources	Consumes less resources
Heavyweight process	Light weight process
Independent data and code segments	Shares data and code segments files etc.
Context switching flushes cache	Context switching does not flush cache.

BY SHIVKARAMAN

- Types of threads

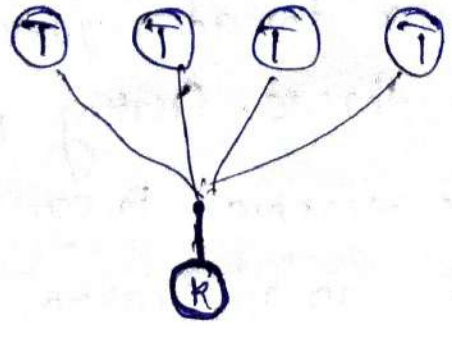
1. User Thread : Supported above kernel and are managed without kernel support. These are threads that application programs use in their programs.
2. Kernel thread : Supported within kernel of OS.

User thread	Kernel Thread
Implimented by users	Implimented by OS
Not Recognised by OS	Recognised by OS
Context switch DOESNOT require hardware support	Context switch requires hardware support.
Designed as dependent threads	Designed as independent threads
Generic in nature and can run in any OS	OS specific.

- Multithreading Models

1. Many to One Model

* Many user threads mapped to single kernel thread



* Thread management is handled by thread library in user space.

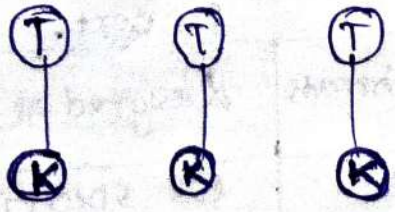
* Disadvantage

* Entire process will be blocked if one thread makes a blocking system call
* As only one thread can access kernel at a time, threads are unable to run in parallel

Eg: GREEN THREAD OF SOLARIS.

2 One-to-One Model

* Creates a separate kernel thread to handle each and every user thread.



* Provides more concurrency than many-to-one model by allowing another thread to run when one thread makes a blocking system call.
* Allows multiple threads to run in parallel

Disadvantages

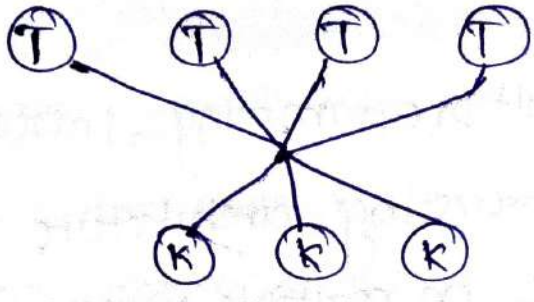
* Creating one ~~the~~ user thread requires creating corresponding kernel thread. Creating many kernel threads is an absolute overhead to system and can burden performance of an application.

* Most implimentations of this System limit no of threads.

Eg: LINUX - , WINDOWS 95 - XP

3 Many to Many model → BEST MODEL

* Multiplexes many user threads to a smaller no of kernel threads.



* Users can create any no of threads
* Blocking system call by one ~~process does~~ thread doesnot block the entire process.
* When multiple user threads are created corresponding kernel threads can run in parallel.

Eg: UNIX, TRU64

- Benefits of Multithreading

1) Responsiveness : One thread can provide rapid response even if other threads are blocked /slowed down.
* It allows a program to run even if a part of it is blocked.

2) Resource sharing : Threads share common code, data

segment, files etc which allows ~~put~~ multiple tasks to be performed simultaniously.

3) Economy

* Allocating memory and resources for process creation is costly job (time and space). It is more economical to create and context switch threads.

4) Scalability

Benefits of multiprogramming increase significantly in case of multiprocessing architecture where threads can run parallelly on multiple processors.

• Examples of multithreaded applications

1) Web Browser : Can download any number of files & web pages simultaniously and still lets you browse efficiently. If a particular webpage cannot be downloaded that is not going to stop browser from downloading other web pages.

2) Web servers : A threaded web server handles each request with a new thread. Each request is assigned a thread from thread pool.

3) Computer games : Various objects like cars, humans, birds are implimented as separate threads. Also

playing background music at the same time is a good example of multithreading.

4) Text Editors: Typing, spelling-checking, formatting are done concurrently by multiple threads.

5) IDE: You can open multiple programs at the same time. It also gives suggestions on completion of command which is a separate thread.

- Optimal No of threads per core

  * If your threads dont do I/O, synchronization etc. and there is nothing else running, 1 thread per core will get you best performance.
  * However that is not gonna happen in reality. Adding more threads usually helps, but after some point they cause performance degradation.
  * So there is no fixed number, it is better to experiment and measure. Performance of parallel programs is not an area where good conclusions can be drawn from first principles alone.

- Why static variables are dangerous in OS?

1) Not very object oriented

They are contrary to object oriented paradigm. In particular, it violates principle of encapsulation

and abstraction of data. In a way you are using as global variables to avoid issue with dealing with scope. However they are not a characterstic of good object oriented code. And they are much difficult to reason about. In OOP each var ~~represents~~ has a state which represents an instance and static variables represent state across instances which is much difficult to unit

2) Object lifetime

Statics have lifetime that matches entire runtime of the program..

PROCESS SCHEDULING

- **Why do we need it?**

Process scheduling is an OS task that schedules processes of different states like ready, waiting etc. Process scheduling allows OS to allocate a time interval of CPU execution for each process. Another important reason to use ~~cpu~~ process scheduling system is that it keeps CPU busy all the time.

- **CPU - I/O burst cycle**

- Process execution consists of a cycle of CPU execution and I/O wait. Processes alternate b/w these two states. Process execution begins with CPU burst that is followed by an I/O burst, followed by CPU burst then another I/O burst, and so on. Eventually a final CPU burst ends with a system request to terminate execution.

- **Dispatcher**

Dispatcher is the module that gives control of CPU to process selected by short-term scheduler. Dispatcher should be as fast as possible, since it is invoked during every process switch.

Time it takes for the dispatcher to stop one process and start another process is called DISPATCH LATENCY.

- **CPU scheduler**

  * Whenever CPU becomes idle, OS must select one of the process in ready queue and assign it to CPU for execution. This is carried by a short-term scheduler / CPU scheduler.

CPU scheduler

Premptive

A process can be forced to leave the CPU and switch to ready queue.

Eg: Unix, Linux, Windows-95 and higher

Non-Premptive

Once CPU has been allocated to a process, process keeps the CPU until it releases the CPU by itself (either by terminating OR by switching to waiting state)

eg: Windows 3.1x

- **Scheduling Criteria**

  - ↑ CPU utilization
  - ↑ Throughput - no of processes completed per unit time
  - ↓ Turnaround time -
  - ↓ Waiting time - time spent in ready queue (only)
  - ↓ Response time

- Terminologies

1) Arrival time (AT) : Time at which process arrives to ready queue

2) Complution time (CT) : Time at which process completed its execution.

3) Burst time (BT) : Time required by process for CPU execution.

4) Waiting Time (WT) : TAT - BT

5) Response time (RT) : First Response time - Arrival time

i.e time from arrival to first response.

6) Turn Around Time (TAT) : CT - AT

- CPU Scheduling Algorithms

Scheduling algorithms are used to select a process for execution.

1 FCFS

- non premptive
- high average waiting time (highest among all algo)
- Convey Effect : Several small processes may need to wait if a large process is given the CPU.

2 SJF

- Process with smallest next CPU burst time is selected
- If two processes have same burst time, one that arrived first is selected.

BY SHIVKARAMAN

- optimal but cannot be implimented.
- It may cause starvation if shorter process keep coming. This problem can be solved using ageing.

Starvation:- Problem that occurs when high priority processes keep coming and low priority processes get blocked for indifinite

ageing : Techneque of gradually increasing priority of processes that wait in system for a long time.

Algorithm

1) Sort all processes according to AT
2) Select the processes that has minimum AT and BT
3) while current process is executing, select process with next min BT in the pool.

Advantages

* Reduces avg WT (Better than FCFS)
* Generally used for Long Term Scheduling.

Disadvantages

* Starvation

3 Priority Scheduling

* CPU is allocated to process with highest priority
* May be premptive or non-premptive
* Advantages
  - avg WT is less than FCFS.

Disadvantage
  - Starvation

Algorithm for priority premptive

1) Sort the processes based on AT
2) Select first process whose AT=0
3) Execute the selected process for 1 unit time.
4) Check priority of next available process
   if priority of curr > others → Execute curr for 1 unit of time
   curr = other of other process
   curr < new process → Execute new process
5) Repeat until all processes in queue are executed.

4 Round Robin

* Processes are cyclically assigned a fixed time slot called as quantum time.
* It is premptive
* If time quantum is
  1) Too large - RR behaves like FCFS
  2) Too small - RR behaves like processor sharing

BY SHIVKARAMAN

* Generally focuses on time sharing technique.
* One of the most widely used algorithm.

Advantages
- Starvation free

~~Disadvantage~~ Disadvantages
- Lot of context switching

5 SJF Premptive / Shortest Remaining Time First

In SRTF, process with smallest remaining BT is selected to execute.

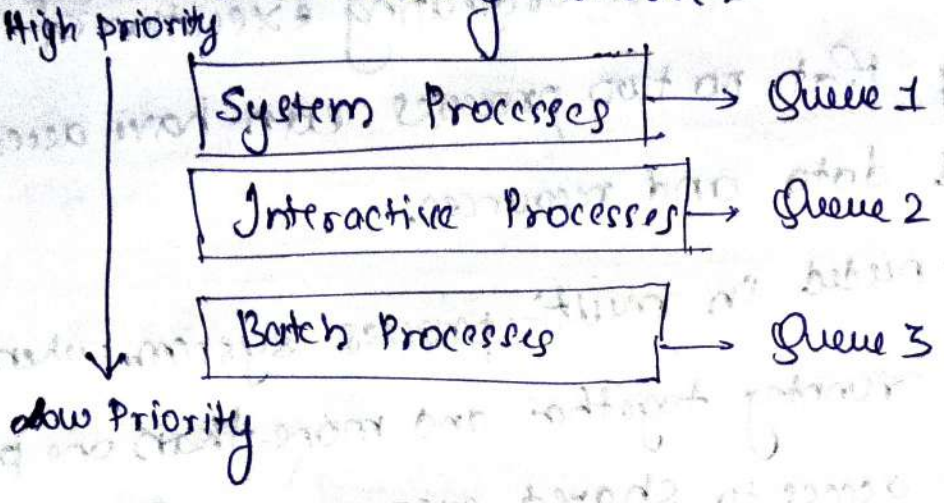
* Context switching is done a lot more times in SRTF than SJF.

Advantages
- Short processes are handled very fast

Disadvantages
- potential starvation
-

6 Multilevel Queue Scheduling (MLQ)
- Ready Queue is partitioned into several queues and each queue has its own scheduling algorithm
- A process is permanently assigned to one queue.

* A common division is a foreground (interactive) process and a background (batch) process. These two classes have different scheduling needs. For this kind of situation MLQ scheduling is used.



Advantages

- Low scheduling overhead

Disadvantage

- Starvation
- Inflexible in nature.

7 Multilevel Feedback Queue Scheduling (MLFQ)

* This is like MLQ Scheduling but the processes can move between queues and thus much more efficient than MLQ.

Advantages

- more flexible
- a process waiting too long in a low-priority queue may be moved to higher-priority queue

Disadvantage

- Most complex algorithm

Process Synchronization

- **Process synchronization and why is it needed?**

It is a task of co-ordinating execution of processes in such a way that no two processes can have access to same shared data and resources.

It is mainly needed in multi-process system when multiple processes are running together and more than one process try to gain access to shared resource or any data at the same time

- **Race Condition**

Several processes ~~trying~~ accesing and manipulate same data in concurrent manner due to which outcome of execution differs from actual expected output is known as race condition. Outcome of execution depends on the particular order in which the access of data takes place.

- **Critical Section**

When two or more process try to access the same code segment / any resource, then that segment is called as critical section.

In concurrent programming if one, thread tries to change value of shared data at the same time as another thread tries to read value, the result is unpredictable. The access to such shared variables is to be synchronised.

```
do {
    [Entry section]

        CRITICAL SECTION

    [Exit Section]
} while (true);   Remainder section.
```

A solution to critical section problem must satisfy

1) Mutual Exclusion : 1 process should only be allowed to access the critical section at a time

2. **Progress**: Progress guarantees that if no process is currently in the critical section and some processes are waiting to enter, then the selection of the next process to enter the critical section cannot be postponed indefinitely. In simpler terms, if a process wants to enter the critical section and there's no contention, it should eventually be able to do so.

remainder section can decide on which process should run in critical section next

3) Bounded Wait : After a process has made a request to access the critical section and before the request is approved, a limit on no of times other processes are allowed to enter critical sec. must exist

BY SHIVKARAMAN

ie there must be a maximum bound upto which a process can wait.

It is not always possible to execute bounded wait (basic version of semaphore also fails in bounded wait) so BW is an optional criteria.

But Mutual exclusion and progress are mandatory criteria

TWO PROCESS SOL'S

1 S/w solution 1

* we use 'turn' variable

$\boxed{turn = 1}$ ⇒ 1's turn to access CS

$\boxed{turn = 0}$ → initially

P_0	P_1
while (true) {	while (true) {
while (turn != 0);	while (turn != 1);
CRITICAL SECTION	CRITICAL SECTION
turn = 1;	turn = 0;
remainder section	Remainder section
}	}

2 S/w solution 2

* we use a flag variable

flag[0] = true ⇒ 0 wants to access CS

0	0	→ Initially
flag[0]	flag[1]	

P_0	P_1
while(1){	while(1){
flag[0] = true;	flag[1] = true;
while(flag[1]);	while(flag[0]);
CRITICAL SECTION	CRITICAL SECTION
flag[0] = false;	flag[1] = false;
}	}

1) In the ~~above~~ ~~two~~ ~~solutions~~ s/w solution 1

1) Mutual exclusion is getting satisfied
2) But progress is not satisfied because we are giving ~~a~~ a chance to other process to enter CS after one process finishes without knowing if it really wants to enter CS (what if it doesnot want to?)

Also, one process is selecting which other process can go to CS in remainder section.

Hence it is not a good solution

:-> In s/w solution 2

1) Mutual exclusion is satisfied
2) Progress is also not satisfied. because say initially

flag $\boxed{F|F}$ (0, 1) and P_0 → flag[0] = ~~true~~
↓ context switch to P_1

P_1 → flag[1] = true

Now flag $\boxed{T|T}$ (0, 1)

BY SHIVKARAMAN

P_0	P_1
while (flag[1]);	while (flag[0]);

There will be a deadlock here and both processes will not access the critical section even if they want to

3 Petersons Solution / Humble Solution / Dekkers Algo

* We use both turn and flag[] here

$\boxed{\text{turn} = 0/1 \quad \text{flag } \boxed{F}\boxed{F}}$

P_0	P_1
while (1) {	while (1) {
flag[0] = T;	flag[1] = T;
turn = 1;	turn = 0;
while (turn == 1 && flag[1] == 1);	while (turn == 0 && flag[0] == 1);
CRITICAL SECTION	CRITICAL SECTION
flag[0] = F	flag[1] = F;
}	}

This algorithm ensures

1) Mutual exclusion
2) Progress
3) Bounded wait

But still, this solution only works for 2 process only

BY SHIVKARAMAN

- ~~Semaphores~~

- **Hardware Solution to CS problem**

1 Test and set

* Shared lock variable

lock = 0 → Unlocked

lock = 1 → Locked

```
bool TestAndSet( bool * target) {
    bool val = *target;
    *target = true;
    return val;
}
```

Atomic operation
OR
Un-interrupt able
operation

lock = false Initially

```
do {
    while( TestandSet( &lock) ).
    CRITICAL SECTION
    lock = False;
}
```

process

2 Swap

* Two boolean values key and lock
  → global var

* lock = false → Initially

```
swap (bool *a, bool *b) {
    bool temp = *a;
    *a = *b;
    *b = temp;
}
```

Atomic op

```
do {
    key = true;
    while (key == true) swap(&key, &lock);
    CRITICAL SECTION
    lock = false;
}
```

In the above 2 processes

1) Mutual exclusion is ensured
2) Progress is ensured
3) Bounded Wait is NOT ensured

- <u>Semaphores</u>

A semaphore is an integer variable that is shared between threads.

Semaphore

Binary Semaphore (Mutex lock)

Counting Semaphore

- Semaphores are used to

1) Solve critical section problem (initialised to 1 at begin
2) Order of execution of process
3) Resource management.

wait (S)

```
wait(S) {
    while (S<=0);
    S--;
}
```

used to lock.

signal (S)

```
signal (S) {
    S++;
}
```

used to release lock

```
do {
    wait(S);   // Entry Section / Lock
    CRITICAL SECTION
    signal(S);   // Exit section / Unlock.
    Remainder section
}
```

Disadvantage of semaphore

* Busy Wait i.e Bounded wait is not ensured.

Advantage

1) Ensures Mutual exclusion using wait & signal
2) Progress : Only those processes which want to enter

BY SHIVKARAMAN

Critical section are requesting for it i.e only these processes have (wait, CS, signal) chunk of code.

Also we are not deciding which process is next i.e turn=1 or flag=1. as in prev solutions.

Solution to Busy Wait

* Instead of busy wait, just block the process and add it to waiting queue and it will be restored when other process releases lock (signal).

```
typedef struct {
    int val;
    struct process * list;
}semaphore
```

```
wait (s) {
    s->val--;
    if (s->val <0) {
        Add process to s->list;
        block();
    }
}
```

```
signal (s) {
    s->val++;
    if(s->val <=0) {
        remove process from
                s->list;
        wakeup (P);
    }
}
```

This may overcome busy wait but may lead to deadlock.

Q) A shared variable x is zero, and is operated by 4 processes W, X, Y, Z. Process W and X increment x by 1 and process Y and Z decrement x by 2. Each process before reading performs wait on semaphore s and signal on s after store. If s is initialised to 2. Find what is max possible value of x after all processes complete execution?

	W	X	Y	Z
	wait(s)	wait(s)	wait(s)	wait(s)
Read	R(x)	R(x)	R(x)	R(x)
CS	x++	x++	x--=2	x--=2
write	w(x) signal(x)	w(x) signal(x)	w(x) signal(x)	w(x) signal(x)

$s=2$
$x=0$

If $s=1$, whatever may be the order of execution of process, the answer would be -2 as addition, subtraction is commutative $(1+1-2-2=-2)$

But it is given that $s=2$, $\Rightarrow$ Two processes can ~~execute~~ access the CS simultaniously

$\downarrow$

See video no 32

Lecture 4.9

• Use of semaphore [redacted]

[1] Order of execution of process

Watch video no 33 | VVIP

Lecture 4.10

eg: $P_2 \rightarrow P_1 \rightarrow P_3$: Order of execution we want

How to achieve it?

$S_1 = 0$ $S_2 = 0$ → Initialise two semaphore var's to zero

wait(S_1) P_1 signal(S_2)	P_2 signal(S_1)	wait(S_2) P_3

[2] Resource managemnt

eg: If we have 5 printers ⇒ 5 processes are allowed to run concurrently at a' time.

In C.S problem, we initialize $S=1$ ⇒ only one process can run in C.S.

But HH 5 can run concurrently so we initialize $S=5$

1 Producer Consumer Problem using Semaphore

Problem: Producer produces when buffer is full
or
Consumer consumes when buffer is empty

Solution: Producer produces only when buffer has some empty slots / when buffer is not full
Consumer should not consume when buffer is empty

Solution

We can see

Prod & Cons → As two processes

Buffer → As the critical Section

We use 3 semaphore variables

- E → empty (initialised to n) → No of empty slots
- F → full (initialised to 0 → No of slots filled
- mutex → semaphore variable used to access the critical section / the buffer. (initialised to 1)

Producer

```
do {
    wait(E)
    wait(mutex)
    [ADD ITEM TO BUFFER]
    signal(mutex)
    signal(F)
} while (true)
```

Consumer

```
do {
    wait(F)
    wait(mutex)
    [REMOVE ITEM FROM BUFFER]
    signal(mutex)
    signal(E)
} while (true)
```

BY SHIVKARAMAN

mutex → Takes care that either Prod or Cons is accessing a buffer at a time

~~Takes care that producer produces only when some slots are empty~~

~~Takes care that consumer consumes~~

2 Reader Writer Problem

Reader → can read data

Writer → can write and read data

Problem: When a writer is writing another writer or reader may access the data

Solution: Writer has exclusive access to database

→ If a process is writing, no other process can write or read

→ If atleast one process is reading, no other process can write

We use two semaphores

Wrt → (initially 1)

mutex → (initially 1)

readcount → counts no of readers (normal int variable)

↓

This is a var shared by diff readers ⇒ When using this var, it should be in C.S

Writer

```
do{
    wait(wrt)
    [WRITE TO DB]
    signal(wrt)
}while(true)
```

Reader

```
do{
    wait(mutex)
    readCount ++;
    if(readCount == 1)
        wait(WRITE)
    signal(mutex)

    [READ FROM DB]

    wait(mutex)
    readCount --
    if(readCount == 0)
        signal(wrt)
    signal(mutex)
}while(true)
```

→ If first reader then dont allow any writer to write to DB

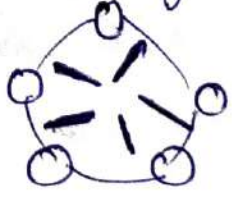
→ If all readers read the DB, we can allow writer to write to DB

- This is not a C.S
  Why? Because multiple readers can read DB simultaniously
- only manipulation of readCount is C.S as two processes cannot manipulate it at same time

3 Dining Philosophers Problem

Here, we are solving the problem using semaphores. So first step is to identify what is the C.S.

Fork/Chop stick → Because it cannot be used by two philosophers at the same time.



No of C.S = No of chop sticks = 5

∴ We need a proper solution to synchronise 5 processes with 5 C.S's.

Solution

- One philosopher can pick one chopstick at a time
- If he has both chopsticks in hand he eats and drops both after finishing.
- Each chopstick → Represented using semaphore
  philosopher grabs a chopstick using wait()
  releases chopstick using signal()

```
semaphore chopstick[5] → all initialised to 1
do {
    wait(chopstick[i])
    wait(chopstick[(i+1)%5])
    [Eat]
    signal(chopstick[i])
    signal(chopstick[(i+1)%5])
} while(true)
```

Assumption
↓
Each philosopher picks right chopstick first and then left

Here there is a possibility of deadlock, say all philosophers pick first chopstick at the same time → Deadlock.

Possible Solutions

- Allow atmost 4 philosophers to be sitting simultaniously at the table
- Allow a philosopher to pick chopstick only if both chopsticks are available
- Odd numbered philosopher picks Left chopstick and then Right
  Even numbered → Reverse
- 6 chopsticks

BY SHIVKARAMAN

Important terms to know

1 Busy Wait

Process synchronization technique in which a process/task waits and constantly checks for a condition to be satisfied before proceeding with its execution.

Eg: wait(s) → In semaphore

2 Spinlock

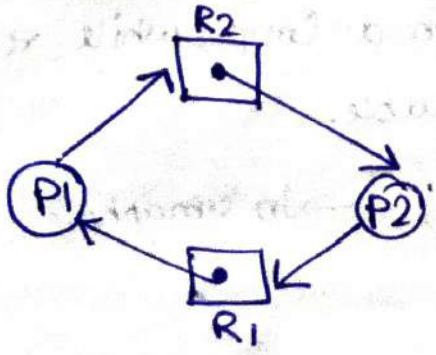
A spinlock is a lock that causes a thread trying to acquire it to wait in a loop while repeatedly checking whether lock is available.

Eg: while (S<0); → In semaphore.

DEADLOCK

* In a multiprogramming system, number of processes compete for limited number of resources and if a resource is not available at that instance, then process enter into ~~deadlock~~ waiting state.

* If a process unable to change its waiting state indefinitely because the resources requested by it are held by another waiting process, then the system is said to be in deadlock.



MULTI-THREADED programs are good candidate deadlock because multiple threads compete for shared resources

What is the effect of deadlock on other processes which are not involved in deadlock?

Only processes involved in deadlock are affected. Most of the OS's use deadlock ignorance (Ostrich algorithm) method to avoid the deadlock in system, while on the other hand other algorithms such as

deadlock prevention / avoidance / detection are expensive to impliment.

If OS has a deadlock prevention or detection system in place, this will have negative impact on performance (slows the system down) because whenever a process / thread requests for a resource, the system will have to check whether granting this request could cause a potential deadlock situation.

- <u>Necessary Conditions for deadlock</u>

[1] <u>Mutual Exclusion</u> : System has atleast one ~~proces~~ resource that is non-sharable and can be used by only one process at a time i.e mutual exclusion.

[2] <u>Hold & wait</u> : A process is holding atleast one resource and requesting additional resources which are held by other resources.

[3] <u>No premption</u> : If a resource cannot be pre-empted from a process ~~from~~ by any other process and it can be released only voluntarily by process holding it.

[4] <u>Circular wait</u> : There exists a set $\{P_0, P_1, P_2 \ldots P_n\}$ of waiting processes such that $P_0 \xrightarrow{\text{waiting}}$ Resource held by P_1, $P_1 \xrightarrow[\text{for}]{\text{waiting}}$ Resource held by P_2, ... $P_n \xrightarrow{\text{waiting}}$ Resource held by P_0

BY SHIVKARAMAN

Deadlock Handling Methods

Deadlock handling
- Prevention
- Avoidance
  - Bankers algo
  - Resource allocation graph
- Detection & Recovery
  - Active approach
  - Lazy approach
- ~~Ignore~~ Ignore

1 Prevention

Designing a system which violates atleast one of 4 conditions of deadlock.

1 Mutual Exclusion violation

If we want to violate this, only way is ~~[illegible]~~ to make the non-sharable resource sharable but if the ~~pro~~ resource is non-sharable, it is the property of hardware and it cannot be changed by anyone.

HENCE MUTUAL EXCLUSION PROPERTY CANNOT BE VIOLATED

Eg: Printer : If two processes are requesting for a printer, printer cannot be shared b/w ~~two~~ the two processes because it can print only ~~data~~ of once process at a time.

BY SHIVKARAMAN

2 Violation of Hold & wait

1) Process is allowed to start execution if and only if it has acquired all the resources.
(less efficient, not implementable)

2) Whenever a process requests for a resource, it doesnot hold any other resources.
(efficient, implimentable)

3) We place a maximum time upto which a process can wait after which process must release all the holding resource and exit.

3 Forceful premption

We allow a process to forcefully prempt the resource held by other process.

- This could be used by high priority / system processes.
- The process which are in waiting state must be selected as victim to snatch its resource instead of process in running state.

4 Circular wait violation

- First give natural no to every resource
- Allow every process to request a resource in either increasing or decreasing order.
- A process can request any no of instances of resource R_i initially and then it can request instances of

BY SHIVKARAMAN

resource R_j if and only if $j > i$.

↓

For increasing order res-allocation.

2 Deadlock Avoidance

To avoid deadlock, we require additional information about how resources are requested, which resources a process will request and use during its lifetime .. etc.

* With this additional information, operating system can decide ~~[illegible]~~ whether for each request, the process should wait or not.

Safe State

A state of a system is called safe if the system can allocate all the resources requested by the process without entering into deadlock.

Safe Sequence

It is the sequence in which processes can be safely executed without entering into deadlock.

1 Bankers Algorithm

If safe sequence can be generated → Safe State
else → Unsafe State

2 Resource Allocation Graph

Claim edge : $P_i \dashrightarrow R_j$: P_i may request resource R_j

Request edge : $P_i \longrightarrow R_j$: P_i requests resource R_j

Assignment edge : $P_i \longleftarrow R_j$: R_j allocated to P_i

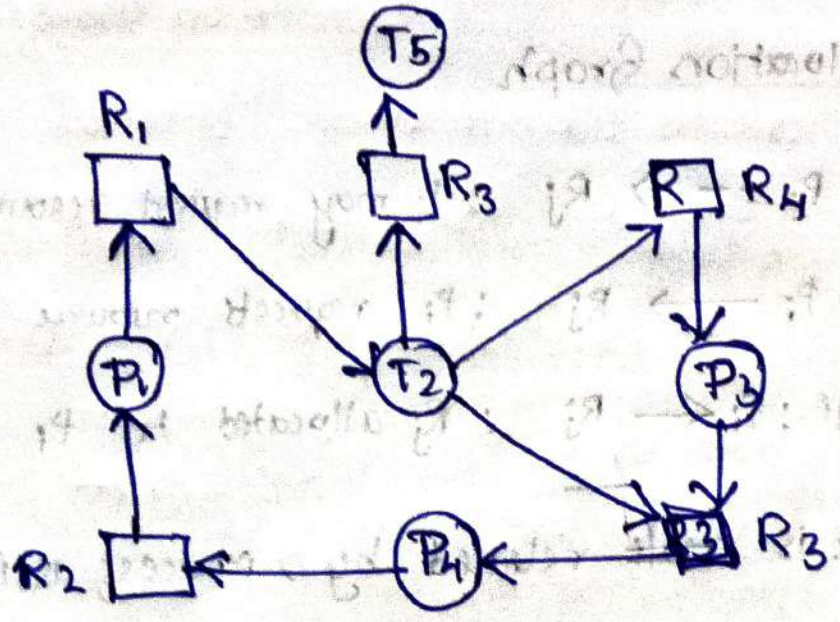
When a resource is ~~reati~~ released by a process, assignment edge reconverts to claim edge (here we assume that the process may request for same resource again in future)

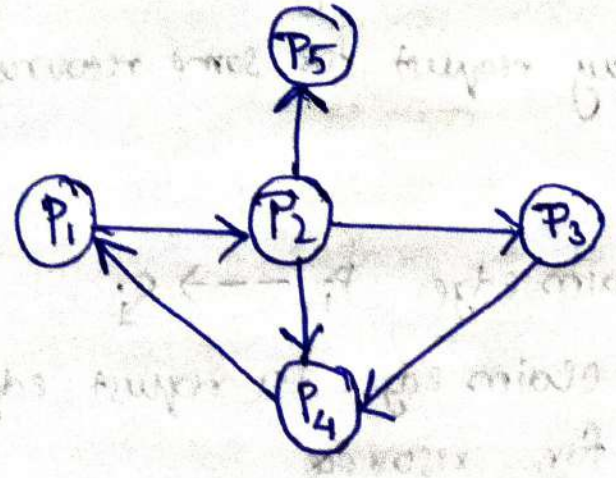
Algorithm

1) Draw claim edge $P_i \dashrightarrow R_j$
2) Convert claim edge to request edge if process requests for resource
3) Convert request edge to assignment edge $P_i \rightarrow R_j$ to $P_i \leftarrow R_j$ if this allocation doesnot result in deadlock. (ie cycle in graph)
4) Convert assignment edge $P_i \leftarrow R_j$ to claim edge $P_i \dashrightarrow R_j$ when process releases the resource.

In this algorithm, we check if allocating a resource to a process ~~do~~ results in a cycle in graph (including dashed edge) and then allocate the resource.

3 Wait-for Graph





(If wait-for graph contains cycle → Deadlock.)

↓

Not true when resource has more than one instance

3 Deadlock detection and Recovery

When process requests for a resource, it is allocated to the process without any safety check. If ~~deadlock~~ system goes in deadlock, then we take required steps.

Deadlock detection

Active approach

Here we simply invoke the algorithm at defined intervals

Passive Approach

whenever CPU utilization drops significantly, we go for detection.

- Recovery from deadlocks

1) Process termination
   - Abort all deadlocked process
   - Abort one process at a time until deadlock cycle is eliminated.

2) Resource preemption

Resources are pre-empted from the processes involved in deadlock, preempted resources are allocated to other processes.

4 Deadlock Ignorance (Ostrich Algorithm)

* When storm approaches, an ostrich puts his head in the sand and pretend that there is no problem at all.
* Similarly, when deadlock occurs, ignore the deadlock and pretend that deadlock never occured.
* This is reasonable if
  - → Deadlock occurs very rarely
  - → Cost of prevention is very high.

BY SHIVKARAMAN

Memory Management Strategies

- **Important points**
  - * CPU can directly access registers and main memory
  - * Protection of memory space is handled by hardware
  - * OS loads Base and Limit registers
  - * Mapping from Logical to Physical address is done by Memory Management Unit (MMU)

- **Memory management**

CPU ⇌ Main Memory ⇌ Secondary Memory

Space of main memory is limited and cannot store lot of processes at once, so CPU cannot switch b/w many processes efficiently ⇒ We need proper memory management to load programs from secondary memory to main memory when CPU requires.

Criteria:	larger size
	low access time
	less cost per unit memory

- **Logical v/s Physical address space**

[1] **Logical address:**

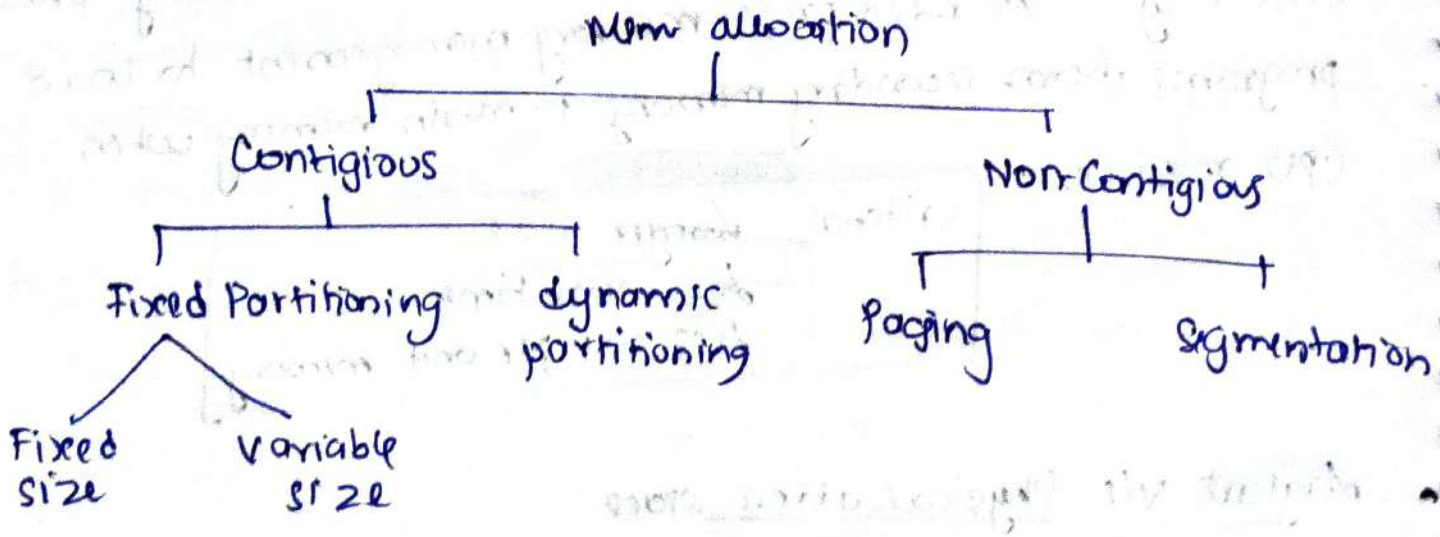
- * Generated by CPU while program is running
- * It is a virtual address and doesnot physically exist

* Logical address space → Has all logical addresses generated by CPU
* Memory management unit (MMU) is used to link logical address to physical address.
* User can view logical address.

2 Physical address

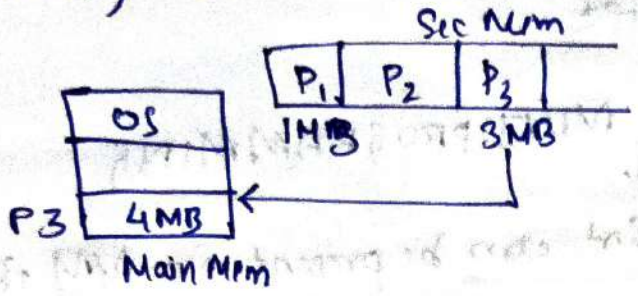
* It is location in memory unit
* It is set of physical address mapped to the corresponding logical addresses.
* User can never view physical address of program but can indirectly access it using logical address
* It is computed by MMU

- Memory Allocation



1 Contigious Memory allocation

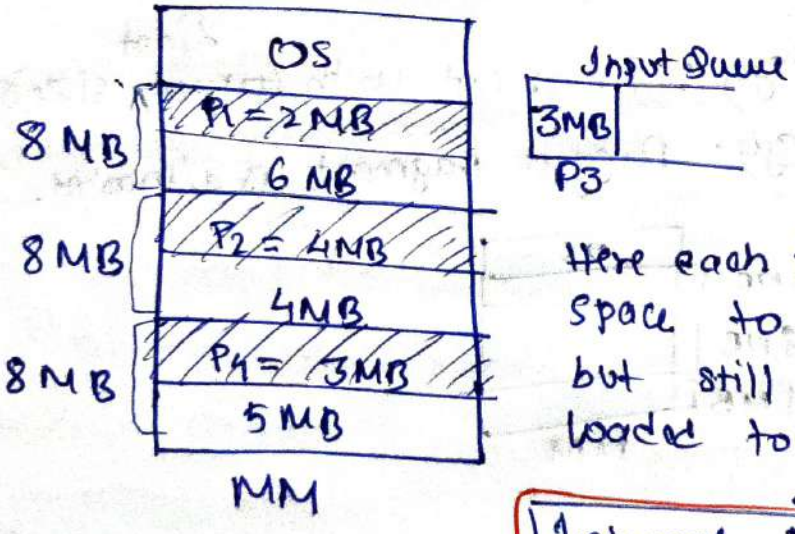
Process is loaded from secondary memory to main memory in contigious fashion (No splitting of process)



- **Fixed Partitioning**

Free space in MM is divided in partitions of fixed size. Each partition can hold atmost 1 process.

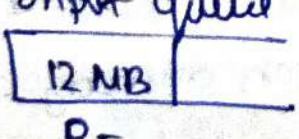
1 Equal size partitioning



Here each fragment has enough space to occupy waiting process but still the process cannot be loaded to MM

↓

Internal Fragmentation



In MM, total (6MB + 4MB + 5 MB) 15MB of space is free, but still process is waiting

↓

External Fragmentation

BY SHIVKARAMAN

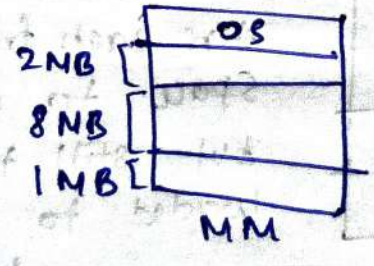
Drawbacks

1) Internal & External fragmentation
2) If size of process > size of fragment → Process cannot be executed
3) FIXED DEGREE OF MULTIPROGRAMMING
   ↓
   No of process that can be present in MM is fixed

2 Variable size Partitioning

Memory is allocated using

1) First fit strategy : First fragment which can hold the process is allocated
2) Best fit strategy : Fragment with ~~largest~~ least size is allocated
3) Worst fit strategy : Biggest fragment is allocated.

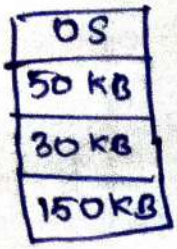


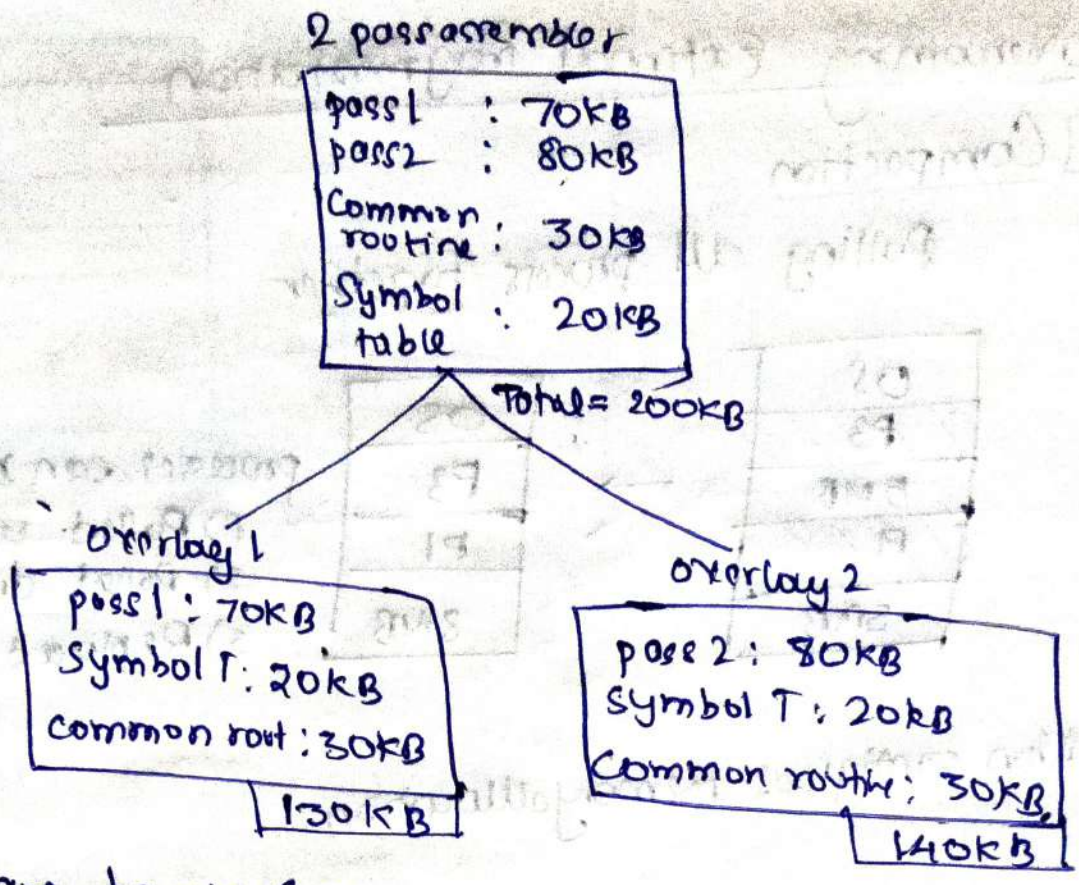
Overlays

If size of process > largest fragment, we divide the process into overlays such that each overlay can fit into partitions and each overlay is independent of each other.

BY SHIVKARAMAN

Eg





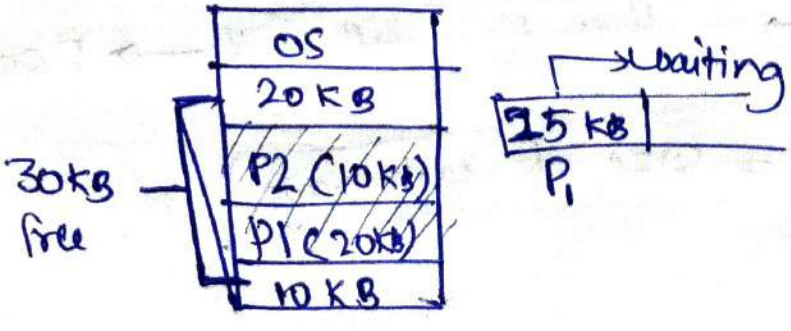
Overlays are bought & executed 1 after other

2 Dynamic Partitioning

Based on size of process, a new partition is created and allocated to process.

* No internal fragmentation
* Degree of multiprogramming is not fixed
* Size of process is not limited by size of partition

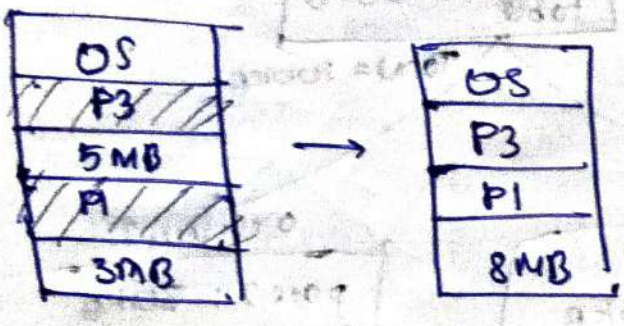
Drawback : External Fragmentation



Overcoming External fragmentation

1 Compaction

Pulling all process together



processes can be
1) Pulled up
2) Pulled down
3) Reload

2 Non contigious memory allocation

2 Non contigious Memory Allocation

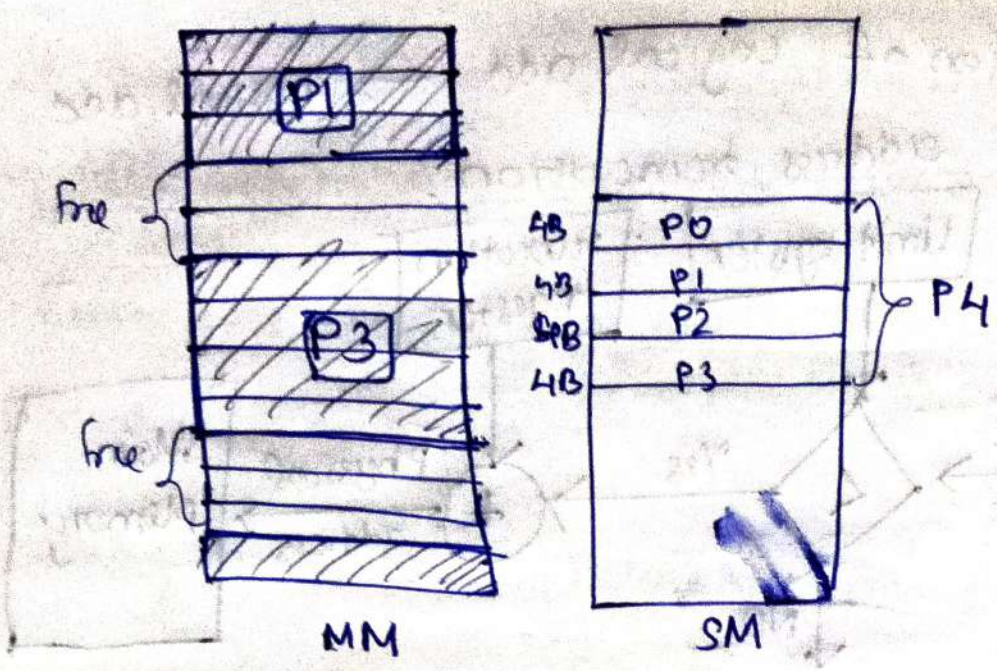
It is a type of memory allocation in which a process is split into different parts and the parts are allocated in memory.

- **Paging**

Secondary memory → Divided into fixed size → (called) PAGES

Primary memory → " " " " → FRAME

Size of each frame = size of each page



Size of page = 4B

We see that 5 frames are free in MM ⇒ Pages of P4 can loaded in MM (order doesnot matter)

↓

NO External Fragmentation → Problem Solved

* Say size of a process is 13B i.e 3.25 pages ⇒ When process is loaded to MM, there would be free space ~~(0.75~~

↓

Internal Fragmentation → Solution → Reduce page size (from size

- **Address Translation, Address Binding**

* CPU always generates a logical address i.e address wrt SM. Since CPU interacts directly with MM, logical address has to be translated to physical address (addr of MM).

* The translation of logical add → Physical add is called as address translation.

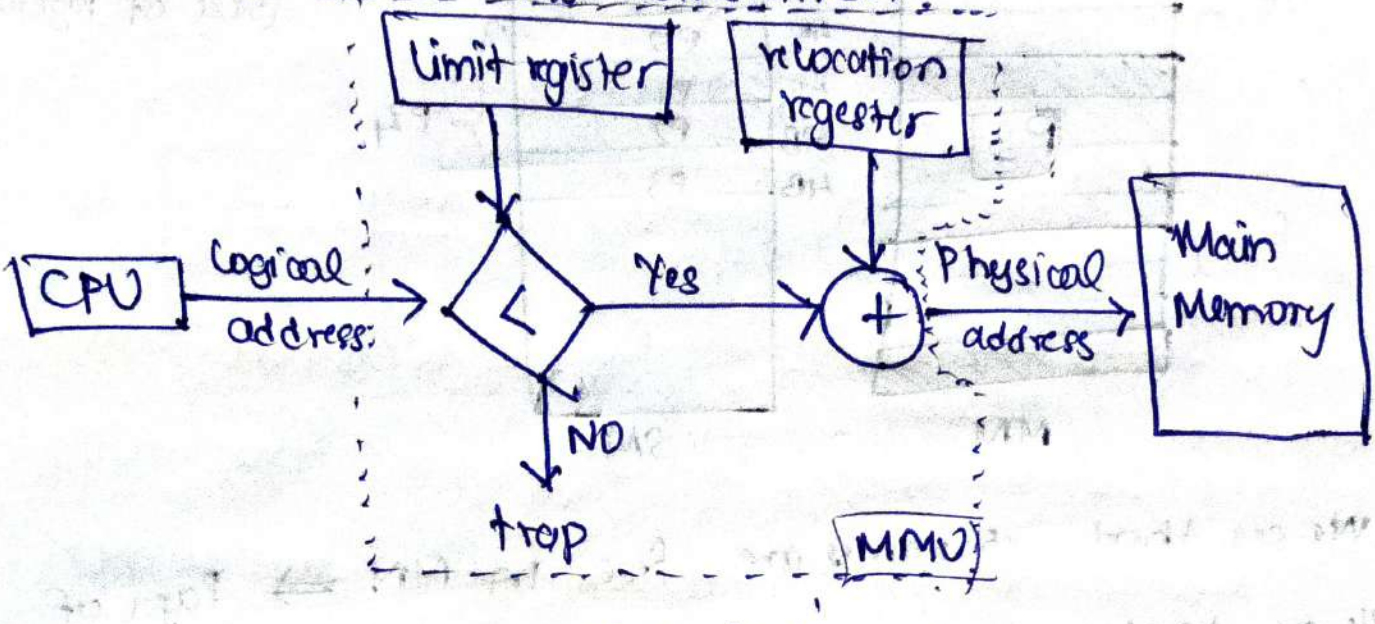


Fig: Address TRANSLATION

Program instructions, data etc from sec memory are mapped to physical memory → This is called as ADDRESS BINDING.

Program —Has→ Symbolic address
MM —Has→ absolute address

Conversion of [sym add → abs add] → Takes place at compile time → [COMPILE TIME BINDING]

↓

This happens only when compiler knows where in MM the program will be loaded.

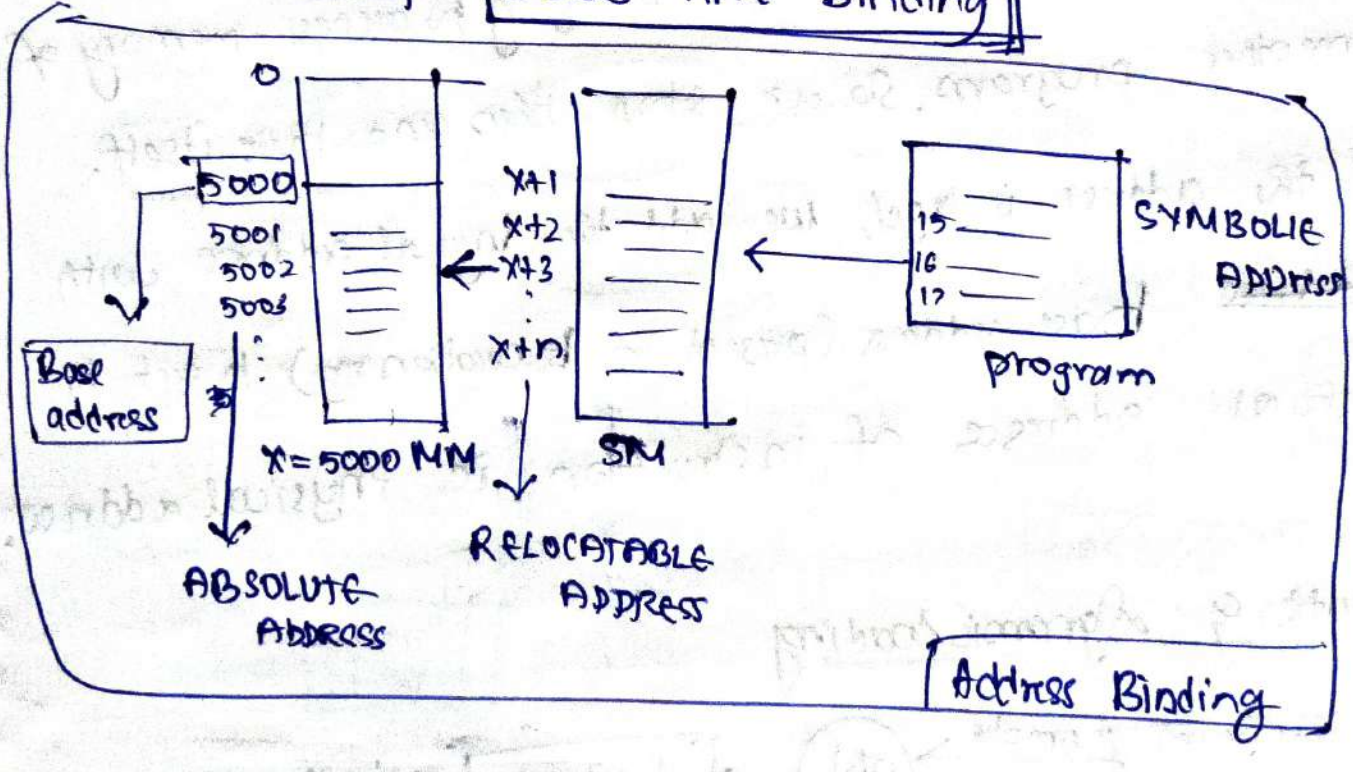
* If compiler doesnot know where memory is free in MM, Sym add → Relocatable address

* Relocatable address → Every address is relative to starting address of program.

* During Load Time, relocatable address $\xrightarrow{\text{converter}}$ Absolute address

↓

~~Comp~~ Load Time Binding



* If during runtime, process is removed from main memory and loaded back with a different base address.

↓

RUNTIME BINDING

Relocatable address → Abs address

At runtime

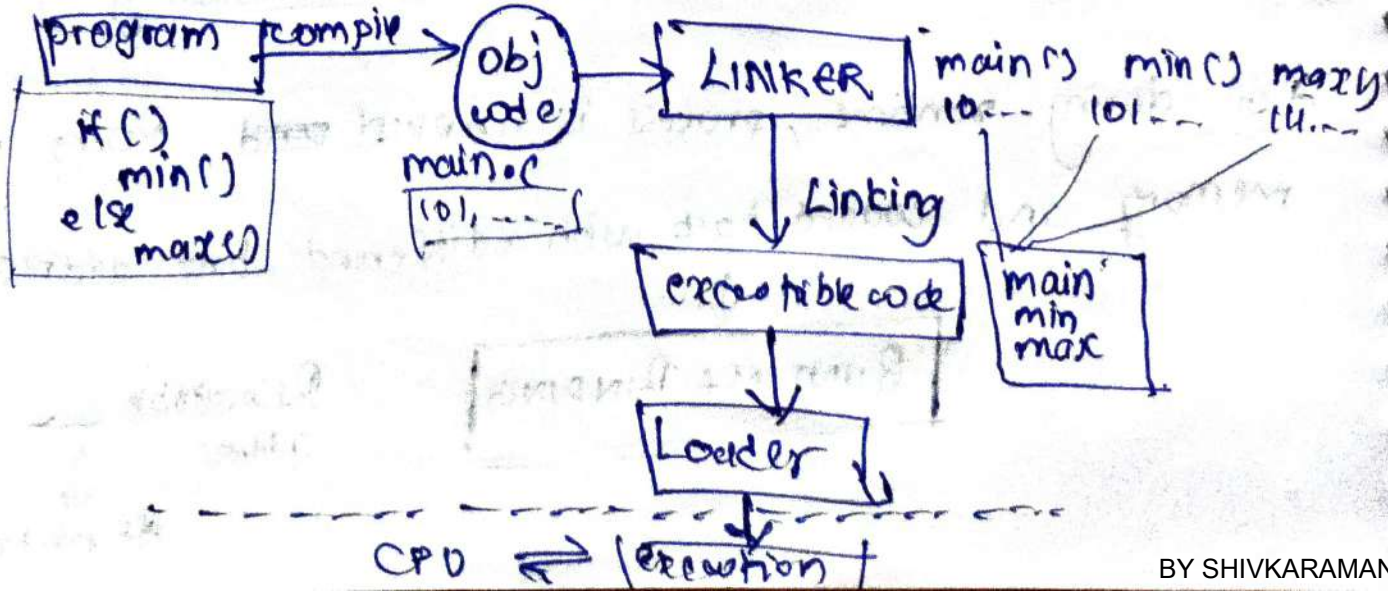
BY SHIVKARAMAN

MMU

- **Limit Register** : It contains range of local address the current process can access.
- **Relocation register** : It contains base address

1) CPU generates logical address (relocatable address)
2) Using limit register, we check if the logical address generated lies in range of the process. If yes we continue else it implies that CPU is trying to access memory of some other program. So we stop there and there itself.
3) If the address is legal, we add the logical address with ~~address~~ base address (present in relocation reg) to get the actual address of instruction. ie Physical address.

- **Static & Dynamic Loading**

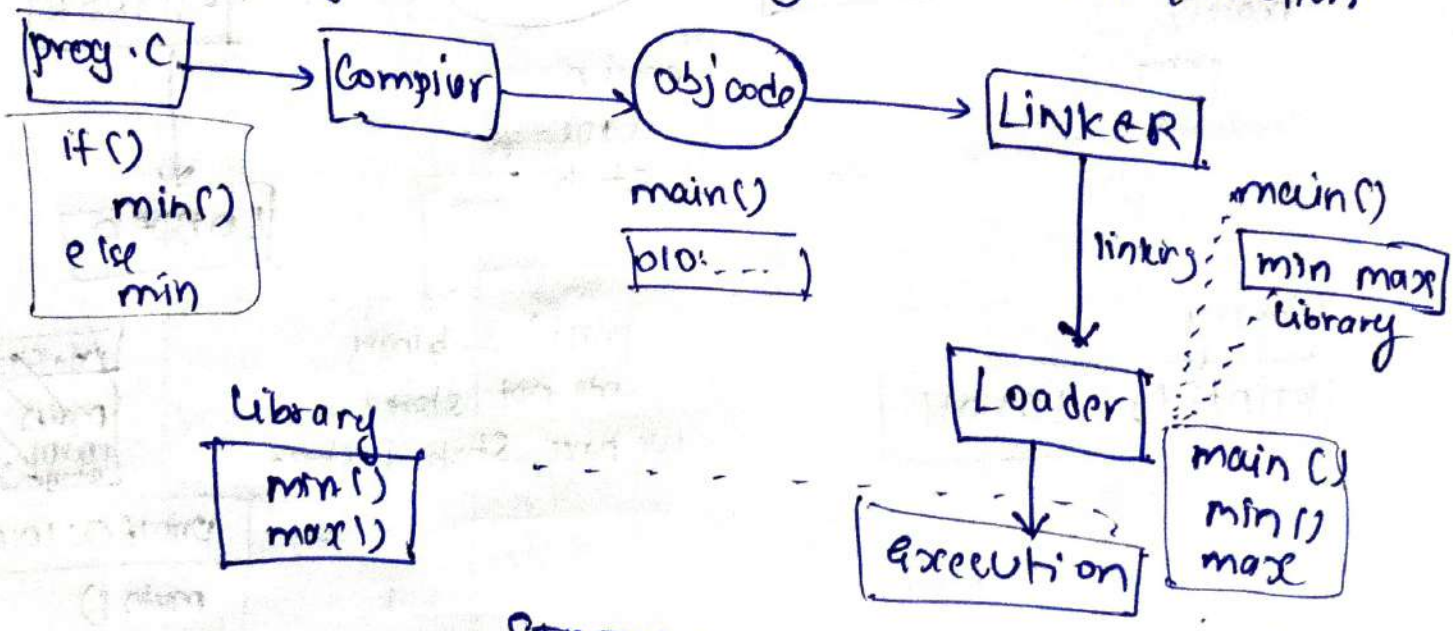


* We either access min() or max() in this program, not both. If condition $\xrightarrow{\text{true}}$ min() else max().
So we are unnecessarily loading one function even if it is not used $\longrightarrow$ **STATIC LOADING**

* Instead, we only load main() initially, and if( ) is true, we load min(), else we load only max()
i.e. function which is required is only loaded $\rightarrow$ **DYNAMIC LOADING**

- <u>Static and Dynamic Linking</u>

~~Static~~ Library $\rightarrow$ Contains frequently used routines / function



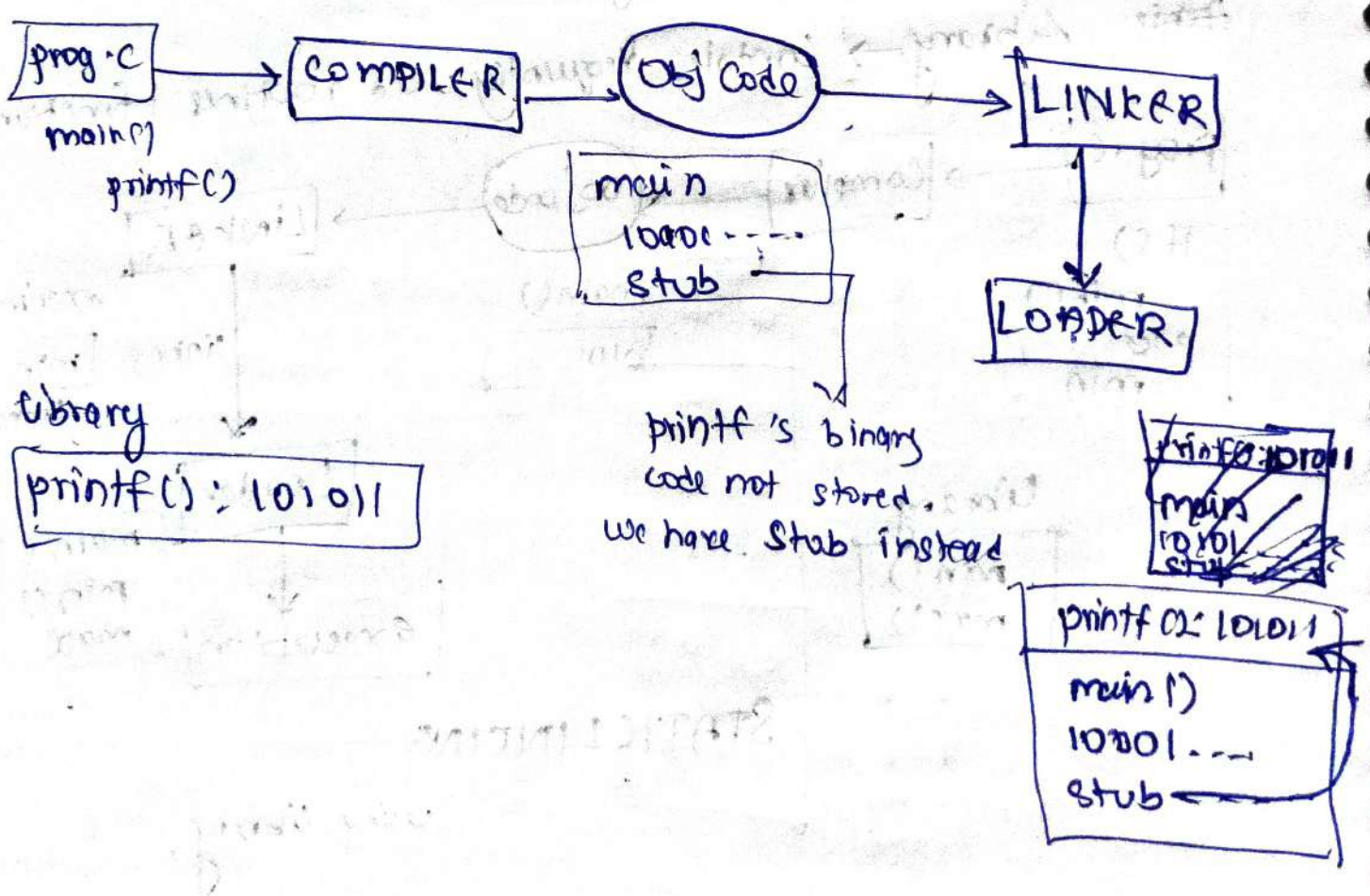
STATICLINKING $\rightarrow$ using library.

* Say our main program has printf() statement, Obviously it is mostly used in all programs. Say there are 'n' processes in MM and all of them use printf().
So instead of linking code of printf() into object module, we keep single piece of code of printf() in MM and make it accessable to every other process

↓

DYNAMIC LINKING

Such libraries → **SHARED LIBRARIES**



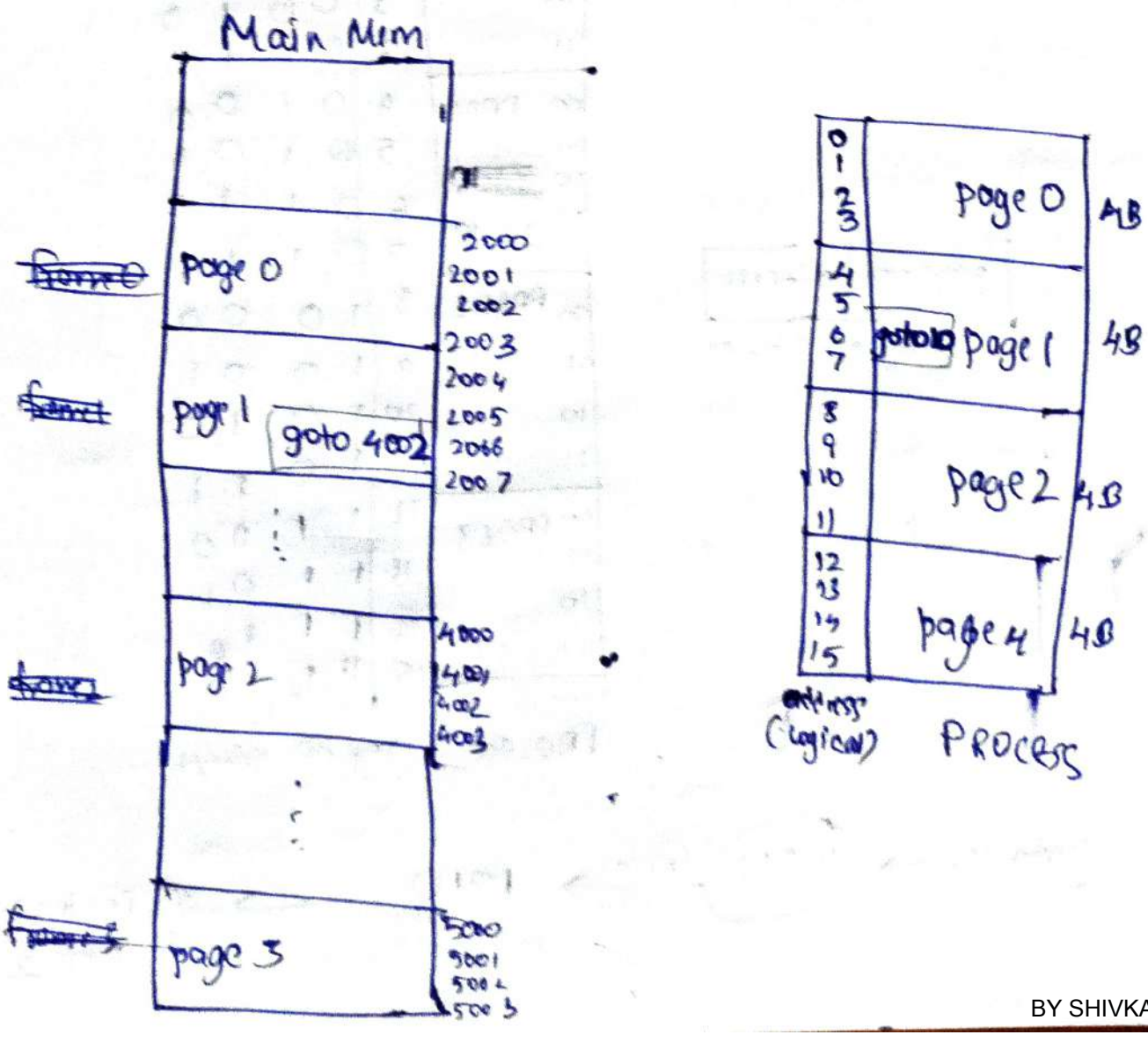
Address Translation in Paging

Relocatable Add/ Logical Add → Absolute address/ Physical address

Here, it is in terms of base address of each frame

eg: Let size of process = 16 bytes
size of page = 4 bytes

process = 16 bytes = 2^4 bytes ⇒ 4 bits
page = 4 bytes = 2^2 bytes ⇒ 2 bits.



We need to find physical address of 10 for $\boxed{\text{goto 10}}$

1) Find base address of frame corresponding to page 1 $= 4000$

2) Relocative address of 10 wrt 8 $= 2 \rightarrow \boxed{\text{Offset}}$

3) Add offset to base address of frame of page 1

$= 4000 + 2$

$= 4002$

The above is actually implemented as follows

page no	offset
2 bytes	2 bytes

offset			logical address			
00	PAGE 0	0	0	0	0	0
01		1	0	0	0	1
10		2	0	0	1	0
11		3	0	0	1	1
00	PAGE 1	4	0	1	0	0
01		5	0	1	0	1
10		6	0	1	1	0
11	goto 10	7	0	1	1	1
00	PAGE 2	8	1	0	0	0
01		9	1	0	0	1
10		10	1	0	1	0
11		11	1	0	1	1
00	PAGE 3	12	1	1	0	0
01		13	1	1	0	1
10		14	1	1	1	0
11		15	1	1	1	1

PROCESS : page no offset

goto 10 → CPU $\xrightarrow{\text{logical address}}$ 1010 → 3rd instruction of page 2

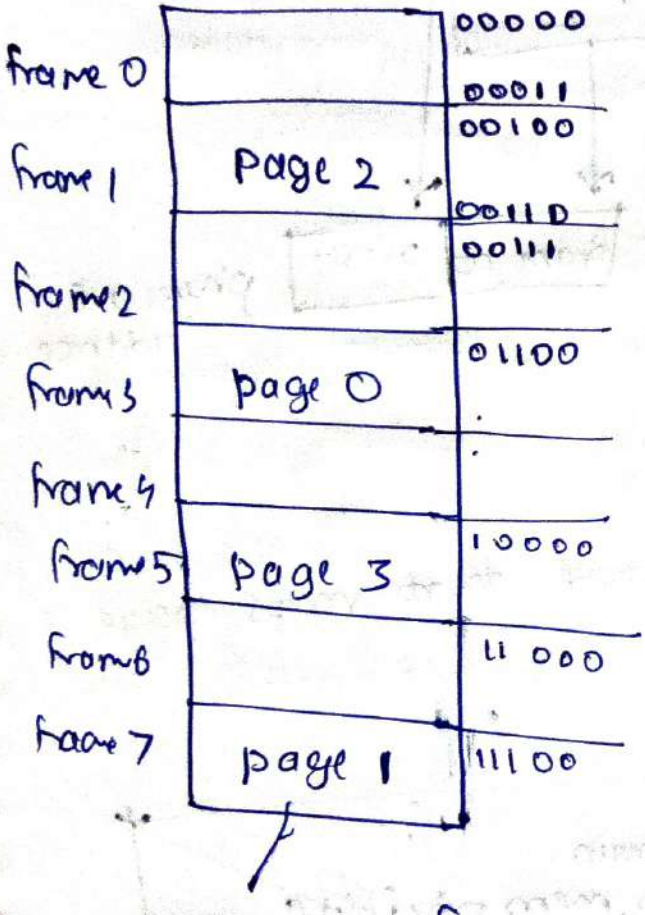
page 2 offset 2

Let size of MM = 32 B = 2^5 bytes ⇒ 5 bytes

$$\text{No of frames} = \frac{\text{Size of MM}}{\text{size of frame}}$$

$$= \frac{32\ B}{4\ B}$$

$$= 8$$



Page table

page	frame
0	3
1	6
2	1
3	4

Each process has 1 page table

No of entries = No of pages

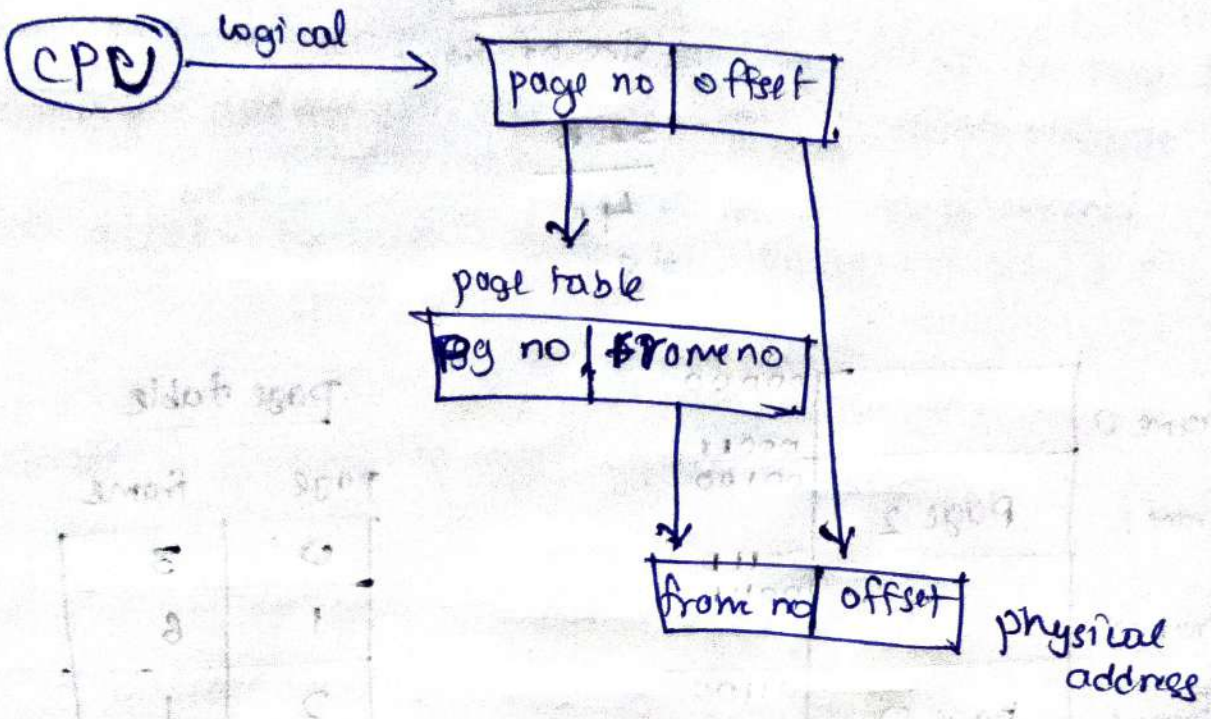
Used to find physical address of given logical address

~~frame = 5 bytes~~ MM = 2^5 bytes ⇒ 5 bytes

~~frame = 5 bytes~~ frame = 2^2 bytes = 2 bytes → OFFSET

frame no	offset
3 bytes	2 bytes

Logical to physical



Page table: It is a data structure that maps page no with frame number.

~~Implimentation of page table~~

~~We keep the page table in main memory itself and index~~

~~* In page table, index → acts as page no~~

~~entry at index → frame to which page is mapped~~

page table

0	frame 3
1	frame 7
2	frame 1
3	frame 9

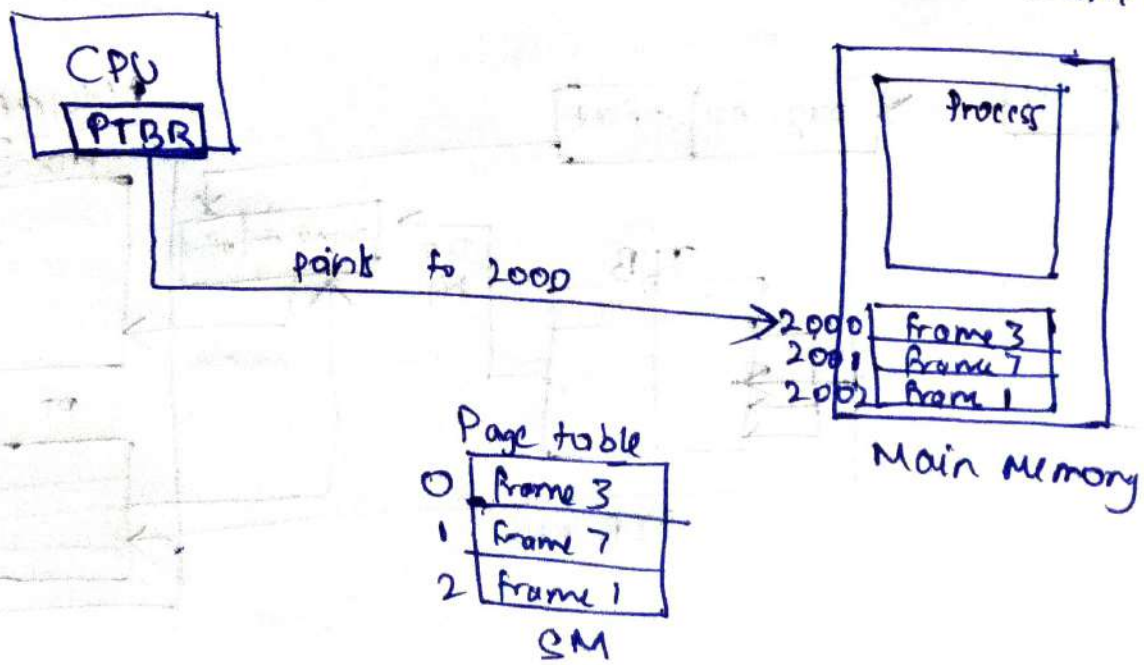
• Page table Access & Storage

1) One option is to use set of dedicated registers for the page table. Here each register is loaded when program is loaded into memory.

Problem: For each context switch, we have to reload all regesters → This increases overall time of context switch effecting performance of system. Also this can be used only when size of PT is small, but in most of the cases, PT are big.

2) Solution to above approach is to keep the PT in MM itself and use a single register (called Page Table Base Register) to store the address of page table in memory.

In page table → index → acts as page no
value at index → Associated frame no



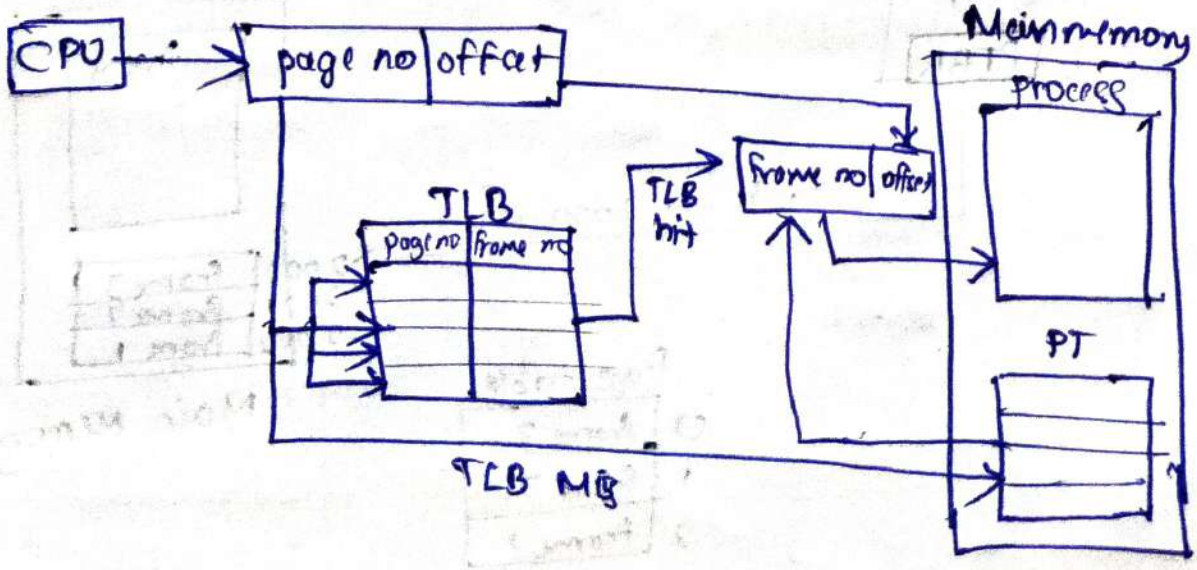
* Context switching is fast because only one register needs to be changed/reloaded.

* **Problem** : Memory access ~~time~~ is ~~to~~ is slow, because every memory access now requires two memory access ~~[illegible]~~

1) To access the page table and get frame number
2) To access the byte within the process.

3) The solution to this problem is to use very special high-speed memory device called as translation-look aside buffer (TLB). It is a hardware cache for the PT and contains few frequently accessed entries of PT.

* It is extremely fast because it can search an entire table for key in parallel.



BY SHIVKARAMAN

No of permissions required

1) For TLB hit → only one permission i.e to access memory of process.

2) For TLB miss → Two permissions
   1) To access page no and get frame no
   2) To access memory of process

* **TLB hit ratio:** No of times, desired info is found in TLB.

Q) Given: Access time of MM = 100ms, TLB = 9ns, TLB hit ratio = 80%

Calculate effective memory access time.

$\boxed{\text{TLB not used} = 2 \times 100 = 200\text{ns}}$

TLB used:

TLB hit = 9 + 100 = 109 ns

TLB miss = 9 + 2×100 = 209 ns

Effective Mem Access Time $= \frac{80}{100} \times 109 + \frac{20}{100} \times 209$

$\boxed{\text{TLB used} = 129\text{ns}}$

- **Multilevel Paging**

2 level paging

P1	P2	offset
outer PT	actual PT	

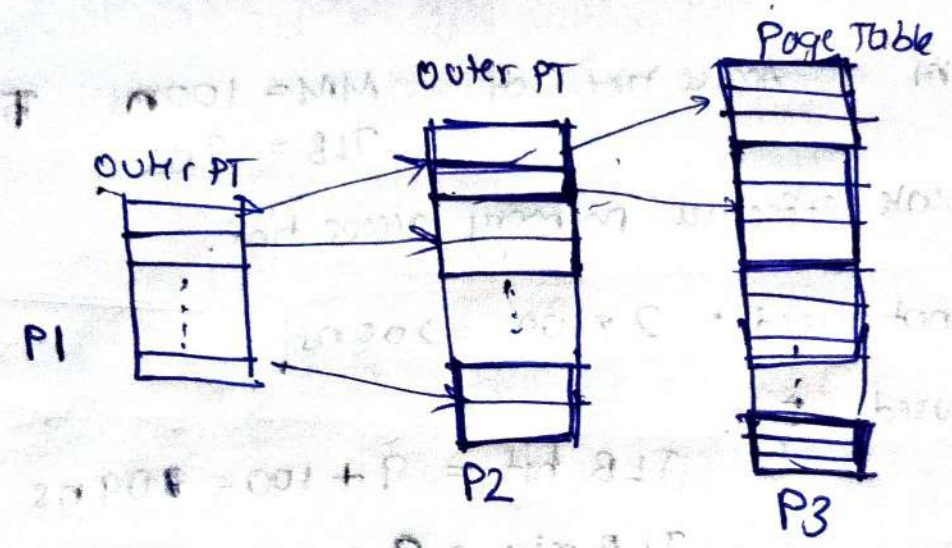
3 level paging

P1	P2	P3	offset

P1 → page no of outermost PT

P2 → offset of 1st outer PT

P3 → Actual PT



Eg: Logical address: 32 bits

Page size = 4KB

Page table entry size = 4B

1K = 2^{10}
1M = 2^{20}
1G = 2^{30}

- Process size = 2^{32}

Page size = 2^{12}

No of pages = $\frac{2^{32}}{2^{12}} = 2^{20}$

Page Table

Size of PT = size of each page × No of pages

= 4×2^{20}

BY SHIVKARAMAN

$\boxed{\text{Size of PT} = 4\text{ MB}}$

Total no of frames required for PT $= \dfrac{\text{Size of PT}}{\text{Size of 1 frame}}$

$= \dfrac{4\text{MB}}{4\text{KB}}$

$= 2^{10}$

$= 1024$ frames

Which is too much and we should multilevel paging until page table fits in 1 frame of MM ie PT size = 4KB.

Outer PT

No of pages in actual PT = ~~No of~~ 1024

$= 2^{10}$

No of entries of outer PT $= 2^{10}$

Size of outer PT = size of 1 PT entry × No of entries

$= 4B \times 2^{10}$

$= 4\text{ KB}$

Logical address

Offset :- Size of page $= 4\text{KB} = 2^{12} \Rightarrow 12$ bytes

PT1 : No of pages of PT1 $= 2^{10} \Rightarrow 10$ bytes

PT2 : No of pages of PT2 $= 2^{10} \Rightarrow 10$ bytes

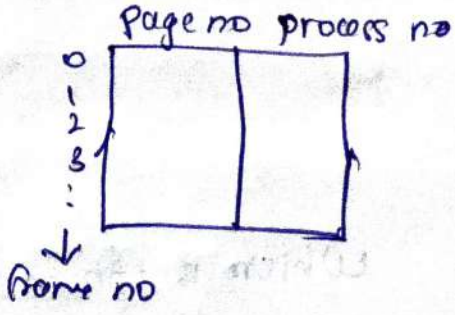
• Implimentation of PT

Implimentation

Hash Tables

* Used when logical address > 32 bits
* Uses hash function for given page no

Inverted PT

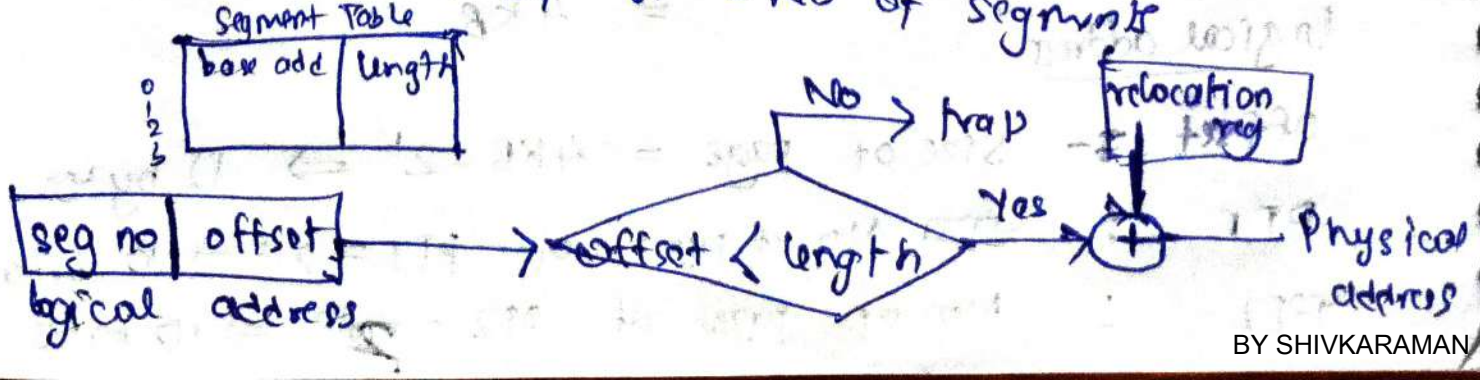


• Segmentation

* Program → Divided into different segments
* Division is according to the user view of memory and size of each partition may vary.
* Different segments can be placed in different parts of memory

SEGMENT TABLE → keeps track of where each segment is kept

* No of entries of ST = No of segments



BY SHIVKARAMAN

- Paging increases context switch time. How & Why?

The OS should keep track of each individual process's process table, updating it when ever the process's page gets moved in and out of memory and applying the correct page table when processing system calls for a particular process.

This all increases the overhead involved when swapping processes in and out of CPU. This increases the context switching time.

Virtual Memory Mgnt

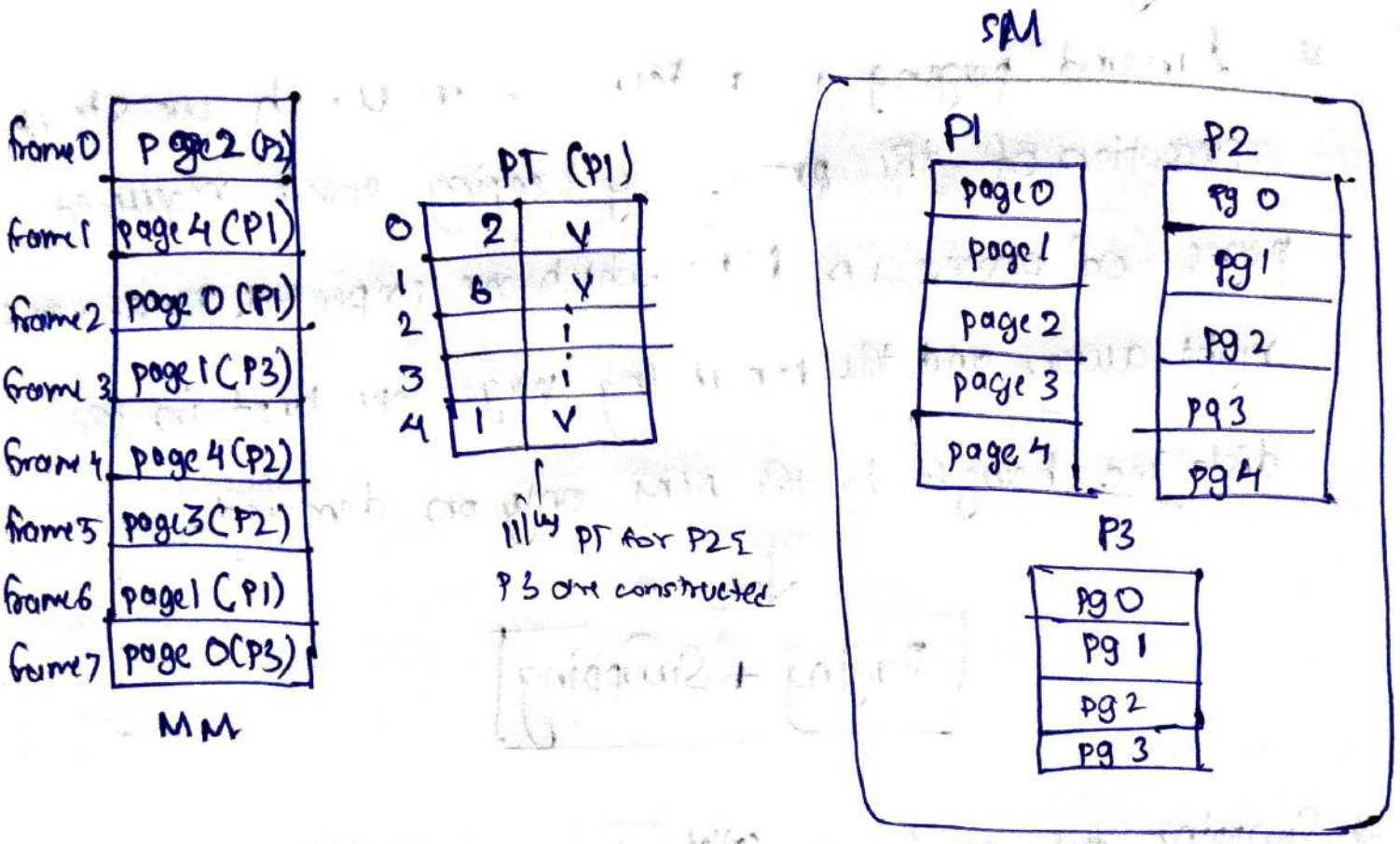
Virtual memory provides an illusion to the programmer that a program / process whose size is larger than the main memory can also be executed. It is a storage scheme that provides an illusion of having a very big main memory. Here secondary memory is ~~as~~ used ~~as~~ as if it was a part of main memory.

It is a technique of running a process without keeping the entire process in MM. We load only required pages into MM and keeping rest of pages in SM itself.

Advantages

1) User can run / execute programs larger than MM
2) Increased degree of multiprogramming
3) Less I/O would be required to load or swap user programs into memory, so each program would run faster
4) Allows files and memory to be shared b/w two or more process through page sharing.

* In page table, we have an additional entry ie VALID/INVALID bit. which tells if the page is loaded to MM or not.



- When desired page is not present in MM $\xrightarrow{\text{called}}$ PAGE FAULT

* To bring a page to MM, we first check if a frame is free in MM or not.
  - If yes, we can just bring that page to free frame and update the page table
  - If no frame is free, we have to choose 1 frame from MM which is to be replaced. We can decide which frame to choose for replacement using PAGE REPLACEMENT ALGORITHM.

- **Demand Paging**

  * Virtual memory is implimented using Demand Paging.

  * Demand paging is a technique in which we start execution of the process by keeping some required pages of process in MM which are expected to be used right away and the remaining pages are kept in the disk and brought to the MM only on demand.

↓

$\boxed{\text{Paging + Swapping}}$

* Swapping only on demand $\xrightarrow{\text{called}}$ $\boxed{\text{Lazy Swapper}}$

- **Pure Demand Paging**

  Pure demand paging is a technique in which we start execution of the process with NO PAGES of process in physical memory and every page of the process is brought to physical memory only on demand.

- Performance - Of Demand paging

p: probability of Page fault

$\Rightarrow (1-p)$: probability of no page fault

WKT, access time when no page fault → only memory access time

If there is a page fault, it will notify the OS and
1) the state of process is saved
2) we bring our desired page from SM → MM and update PT (using page replacement algo if no frame is free)
3) If the page we bought to MM is modified, we need to write back that page to SM and update the page table

Time req for all the processes

$$\therefore \text{Effective Access Time (EAT)} = (1-p) \times \text{Memory access time} + (p) \times \text{Page fault time}$$

Page fault time is usually in 'ms' and memory access time is usually in 'ns'.

$$\boxed{\text{EAT} \propto \text{Page fault Time}}$$

Goal → To keep page fault as low as possible.

- **Swap space** : A B Section of hard disk used for implementing Virtual Memory.

- **Page Replacement ALgorithme**

1 FIFO (belady's anomaly)

* Oldest page in memory is selected for replacement
* Easy to implimnt

f3			1	1	1	1	0	0	0	3	3	3	3	2	2
f2		0	0	0	0	3	3	3	2	2	2	2	1	1	1
f1	7	7	7	2	2	2	2	4	4	4	0	0	0	0	0
	*	*	*	*	Hit	*	*	*	*	*	Hit	*	*	*	Hit

Reference String: 7, 0, 1, 2, 0, 3, 0, 4, 2, 3, 0, 3, 1, 2, 0

Hit = 3

Fault = 12

$$\text{Hit ratio} = \frac{3}{15} = 0.20 = 20\%$$

$$\text{Fault ratio} = \frac{12}{15} = 80\%$$

~~Let's try the same question with 4 frames~~

Beladys Anomaly

It is a phenomenon in which increasing no of frames increases the no of page faults for a given reference string.

But logically if we think, increasing no of frames should decrease no of page faults.

[2] Optimal Page Replacement Algorithm

* Replace the page which will not be used in longest dimension of time in FUTURE ~~in we replace~~ the

			2	2	2	2	2	2	2	2	2	2	2	2
		1	1	1	1	1	4	4	4	4	4	4	4	4
	0	0	0	0	0	0	0	0	0	0	0	0	0	0
7	7	7	7	7	3	3	3	3	3	3	3	1	1	1
*	*	*	*	Hit	*	Hit	*	Hit	Hit	Hit	Hit	*	Hit	Hit

Ref : 7 0 1 2 0 3 0 4 2 3 0 3 1 2 0

Hit = 8

Fault = 7

$$\text{Hit ratio} = \frac{8}{15} \times 100 = 53.3\%$$

$$\text{Fault ratio} = \frac{7}{15} \times 100 = 46.6\%$$

BY SHIVKARAMAN

3 Least Recently Used (LRU) algorithm

* Replace the least recently used page in PAST

			2	2	2	2	2	2	2	2	2	2	2	2	2	2	2	2	2
		1	1	1	1	1	4	4	4	4	4	4	1	1	1	1	1	1	1
	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
7	7	7	7	7	3	3	3	3	3	3	3	3	3	3	3	3	7	7	7
7	0	1	2	0	3	0	4	2	3	0	3	2	1	2	0	1	7	0	1
*	*	*	*	Hit	*	Hit	*	Hit	Hit	Hit	Hit	Hit	*	Hit	Hit	Hit	*	Hit	Hit

Hit = 12
Miss = 8

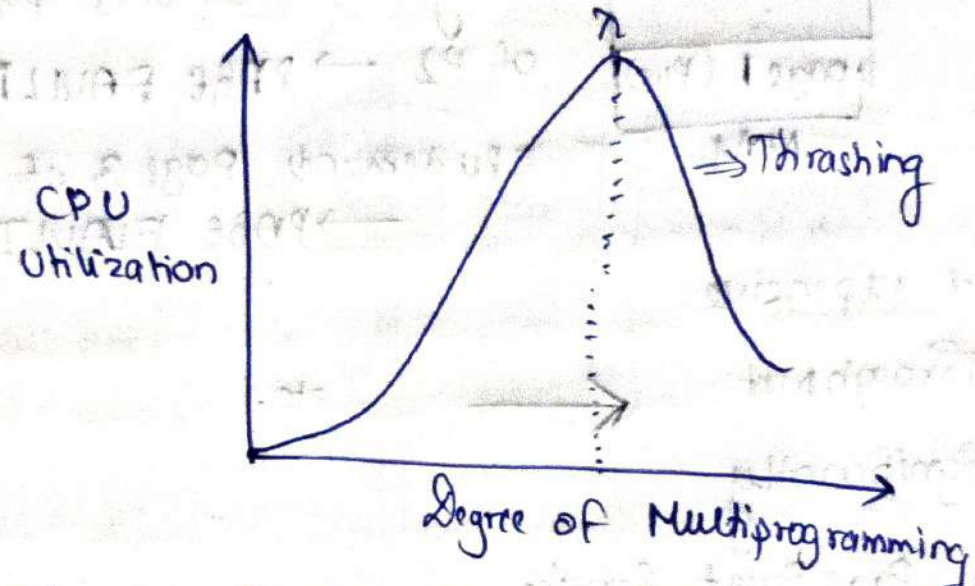
$$\text{Hit ratio} = \frac{12}{20} \times 100 = 60\%$$

$$\text{Miss ratio} = \frac{8}{20} \times 100 = 40\%$$

BEST PAGE REPLACEMENT ALGO → Optimal Page Replacement Algorithm

- Thrashing [VVIP] [Most Asked]

It is a condition in which system is spending a major portion of its time ~~syst~~ servicing the page faults.



Thrashing is directly linked with degree of multiprogramming

Less no of process in MM → CPU sits idol when process demands for some I/O operation and goes to block state

↓

Throughput decreases

↓

We need to increase the no of process in MM but size of ~~pro~~ MM is limited.

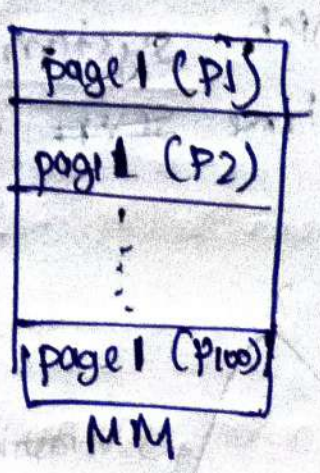
↓

So we use demand paging and bring 1 page (say) of 100 processes into MM → Degree of Multiprogramming increased.

BY SHIVKARAMAN

* But this is a BIG PROBLEM

say



High degree of Multiprogramming is achieved but say CPU demands page 2 of P2 → PAGE FAULT

CPU demands page 3 of process 3 → PAGE FAULT

⋮

etc.

This is a time expensive process. Throughput decreases significantly ←

CPU busy in Page Fault Service

<u>Solution</u>

1) Increase MM size → Not an easy task and expensive

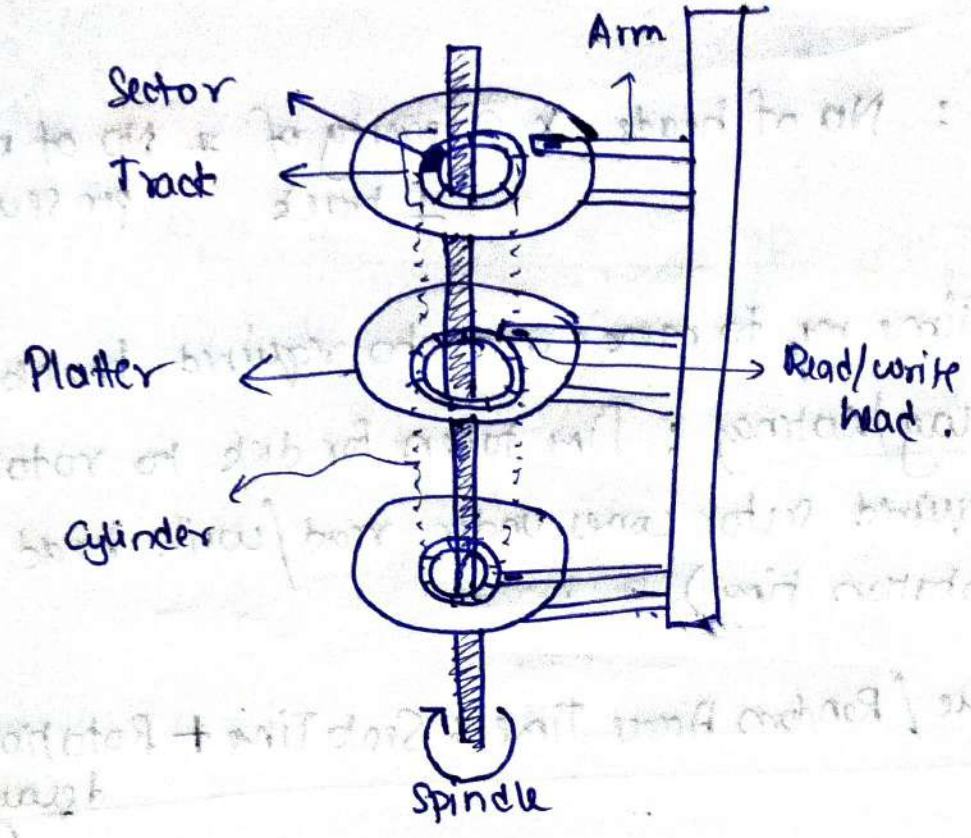
2) Use priority based Replacement algorithm

3) Allocate exact no of frames that are actually required.

BY SHIVKARAMAN

Storage Management

Disk Archetecture



* Information stored magnetically
* Has multiple platter which has flat circular shape like CD
* Two surfaces of platter are covered with magnetic material
* Surface of platter → logically divided into CIRCULAR TRACKS

CIRCULAR TRACKS → subdivided into SECTORS

* Basic unit of storage → SECTOR
* Set of tracks make up a cylinder.

* Disk rotates at constant speed.

Important terminology

1) Transfer Rate : No of heads * Capacity of 1 track * No of rotation per second

2) Seek Time : Time req to move arm to required track

3) Rotational Delay / Latency : Time taken for disk to rotate so that required sector comes under read / write head ($= 1/2 \times$ rotation time)

3) Positioning time / Random Access Time : Seek Time + Rotational delay

No of heads

2 * No of platters * No of tracks in each platter * No of sectors * Data size in each sector

Total no of tracks

Total no of sectors

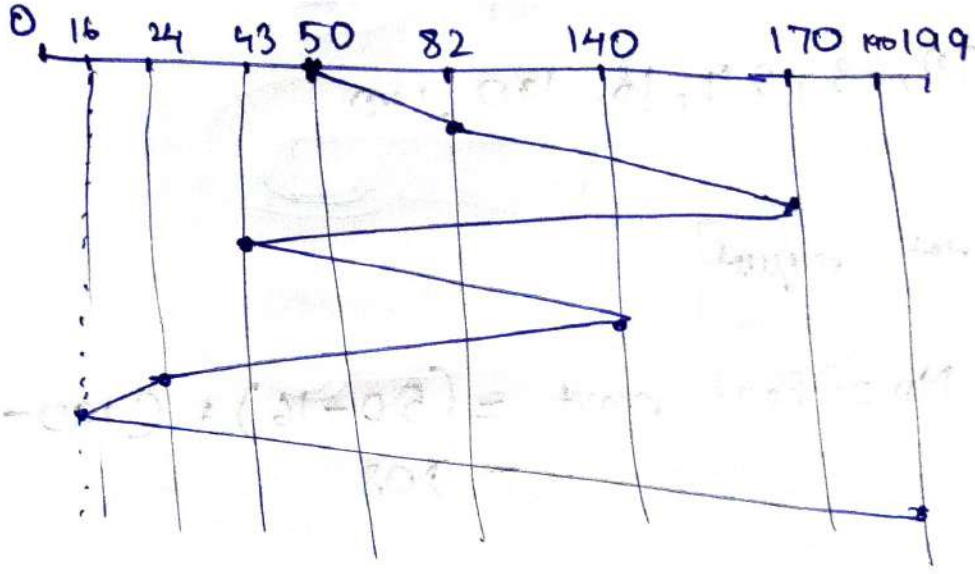
Size of disk

Disk Scheduling Algorithms

Goal → Minimise Seek Time

1) FCFS

initial pos = 50

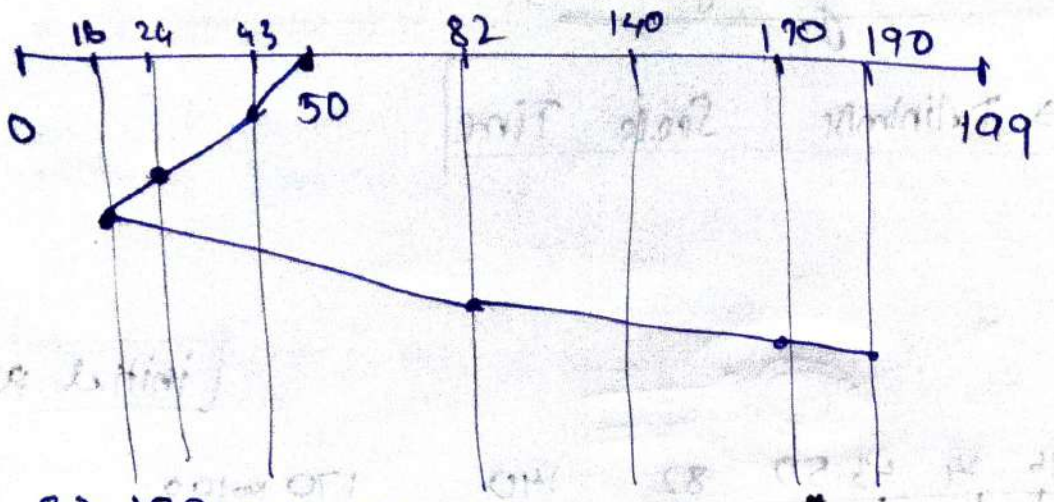


Request Queue = 82, 170, 43, 140, 24, 16, 190

No of head movements = (170 − 50) + (170 − 43) + (140 − 43) + (140 − 16) + (199 − 16)

= 651

2) SSTF (Shortest seek time First)



82, 170, 43, 24, 16, 190, 140

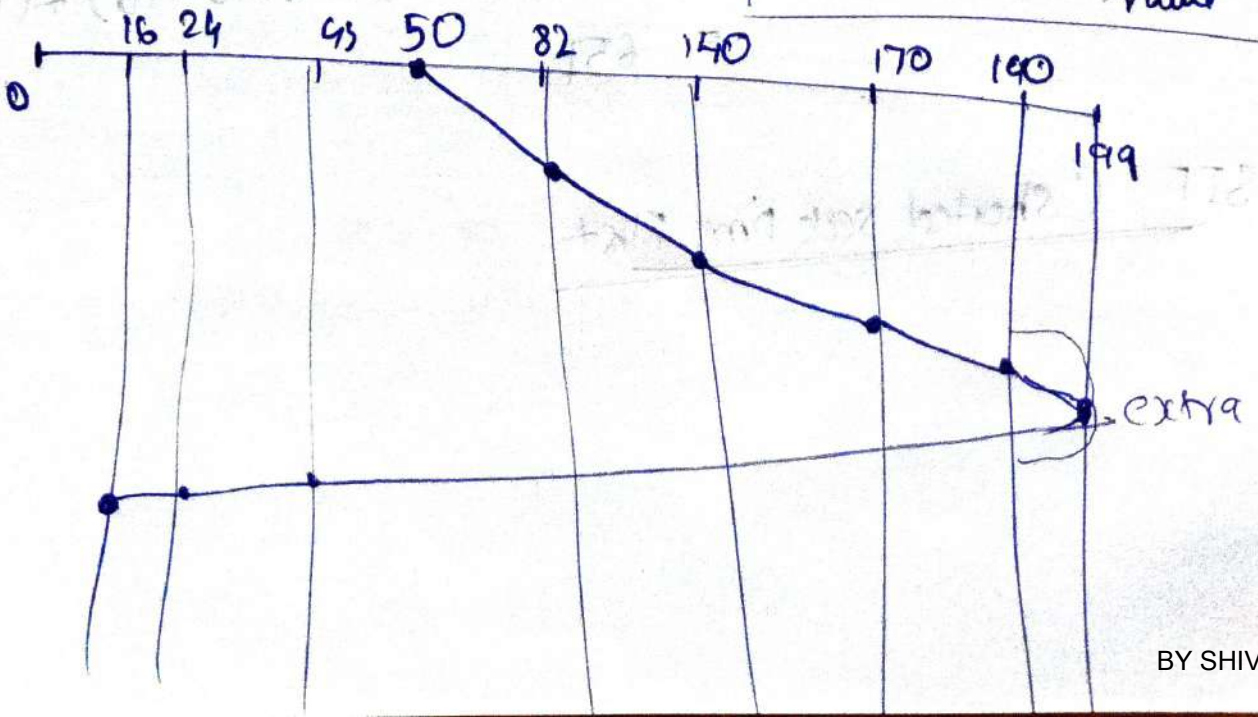
Go to nearest request

No of head movt $= (50-16) + (190-16)$
$= 208$

3 SCAN algo

Given the direction, go along the direction until largest request is processed and also move till last value (even if not in request queue)

Given; move towards larger value

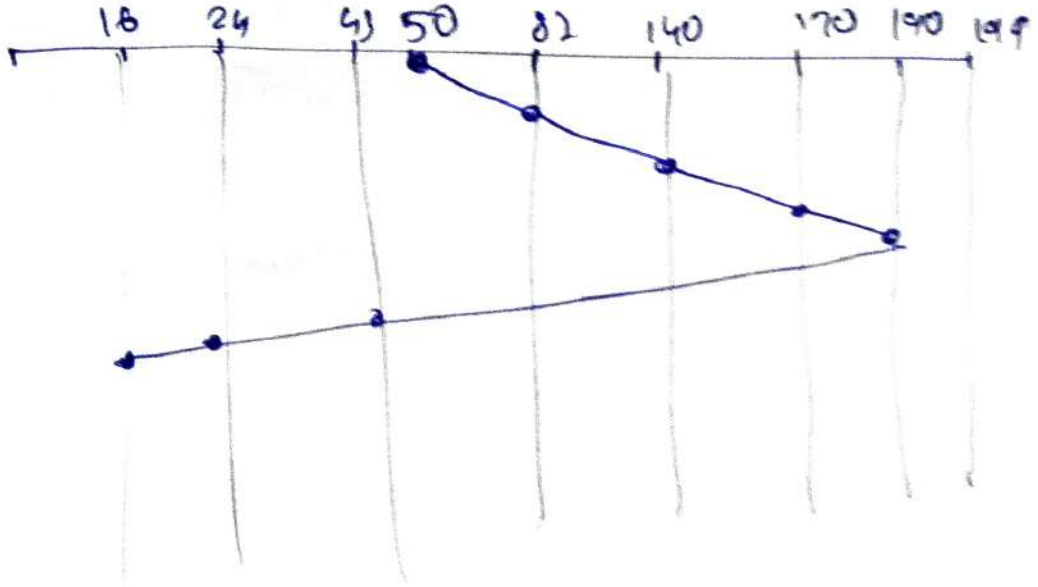


No of head movt = (199 − 50) + (199 − 16)

= 332

4 LOOK algo

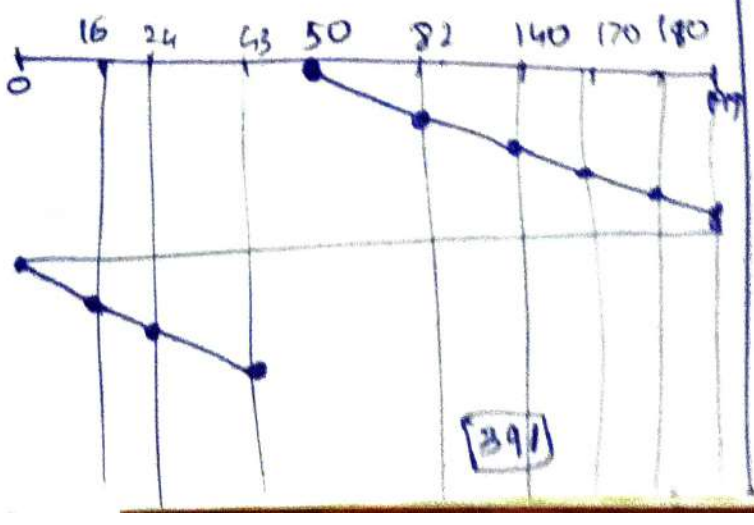
Given the direction, go along the direction until largest is processed and DO NOT Move till last value of large.



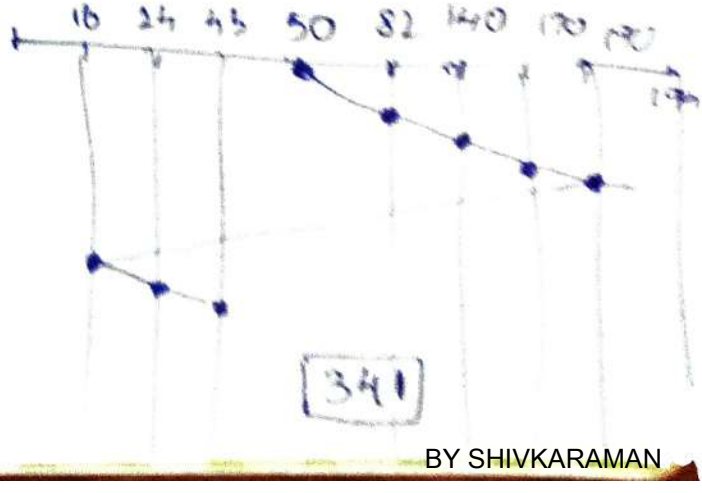
No of head movt = (190 − 50) + (190 − 16)

= ~~314~~ 314

5 C-SCAN



6 C-LOOK



BY SHIVKARAMAN