

# " OPERATING SYSTEM "

1

## Chapter - 1 :- "Introduction & Background"

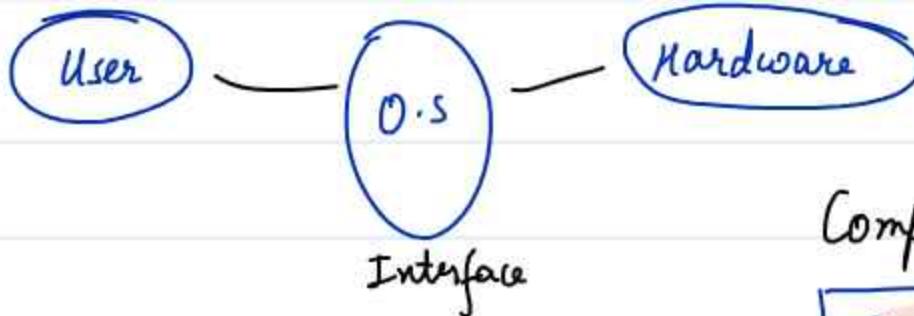
- 1.1 What is Operating System ✓
- 1.2 Function & Goals of Operating System ] → Lec-1
- 1.3 Types of Operating system
- 1.4 Multiprogrammed Operating System ✓
- 1.5 Architectural requirements for multiprogrammed OS
- 1.6 Mode Shifting in Multiprogrammed OS ★
- 1.7 System Calls ★
- 1.8 Fork System Call
- 1.9 Problem Solving

- Textbooks :-
- OS Concepts by Galvin ✓
  - Modern OS by Tanenbaum ✓
  - OS by Stallings ✓

Let's start

What's an OS ?





C.U :-

- Timing / control signals
- controls the operations

→ Sequencing &  
Execution of Micro-ops.

Operations that

are carried out on Data stored in Registers.

`int a, b, c;`

`b=1; c=2;`

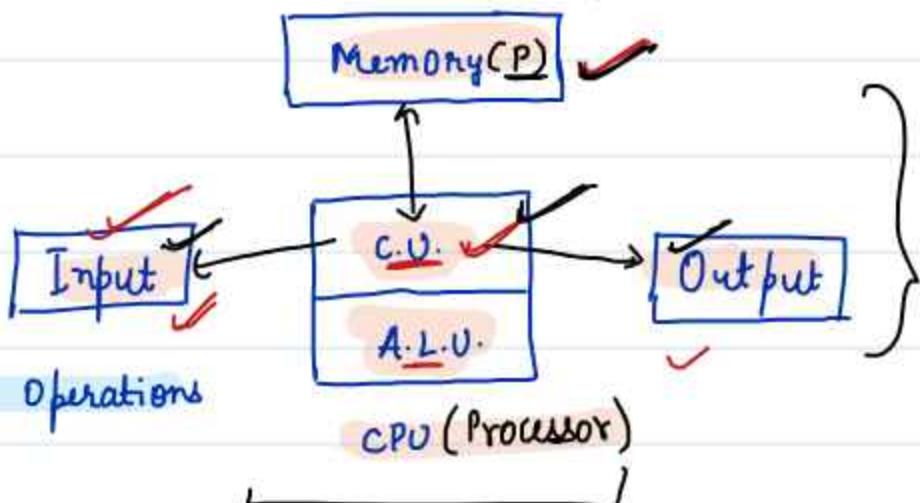
`a = b + c;` → High Level Statement { Macro-operation }



Micro Instruction

1. Load  $R_1, b$
2. Load  $R_2, c$
3. ADD  $R_1, R_2$
4. Store  $A, R_1$

### Components of Hardware



### Architecture of Computer

Proposed by Von Neumann

ALU :- Functional Unit

→ like Arithmetic Functions

→ consists of Adders, Subtractors etc.

Main unit that carries out Arithmetic & Logical Operations

Memory

→ volatile, faster to access, Expensive, Size ↓  
Primary (Main) → RAM, ROM, Cache, Registers

→ Secondary (Auxiliary) → Harddisk, Pendrives

→ Non volatile, slower to access, less expensive, Size ↑

The content doesn't

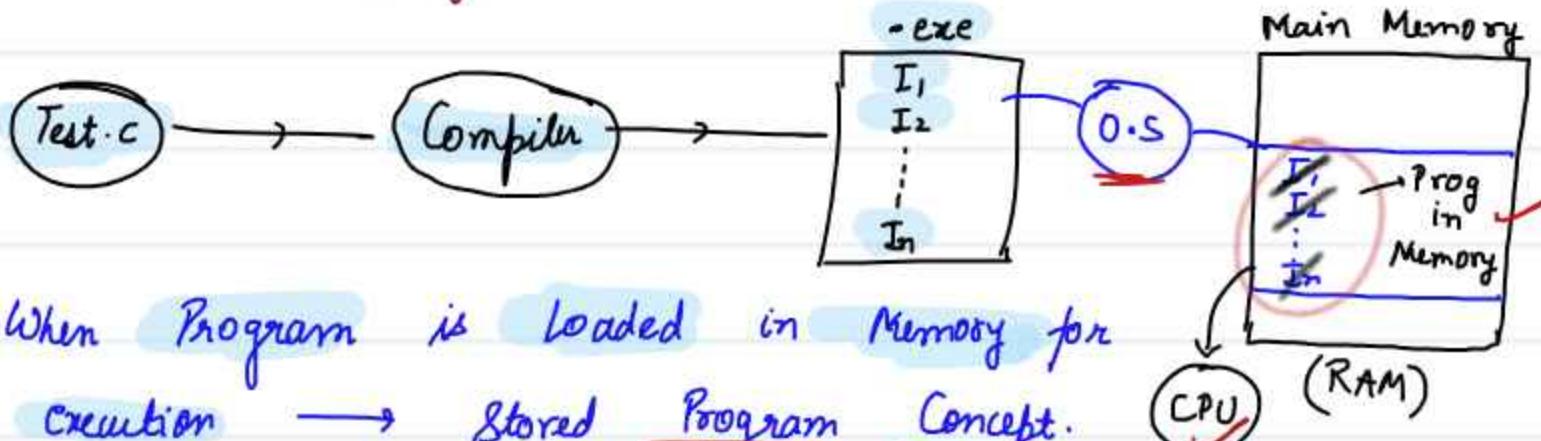
vanishes when the power supply goes off.

Q Where is secondary Memory Then ?

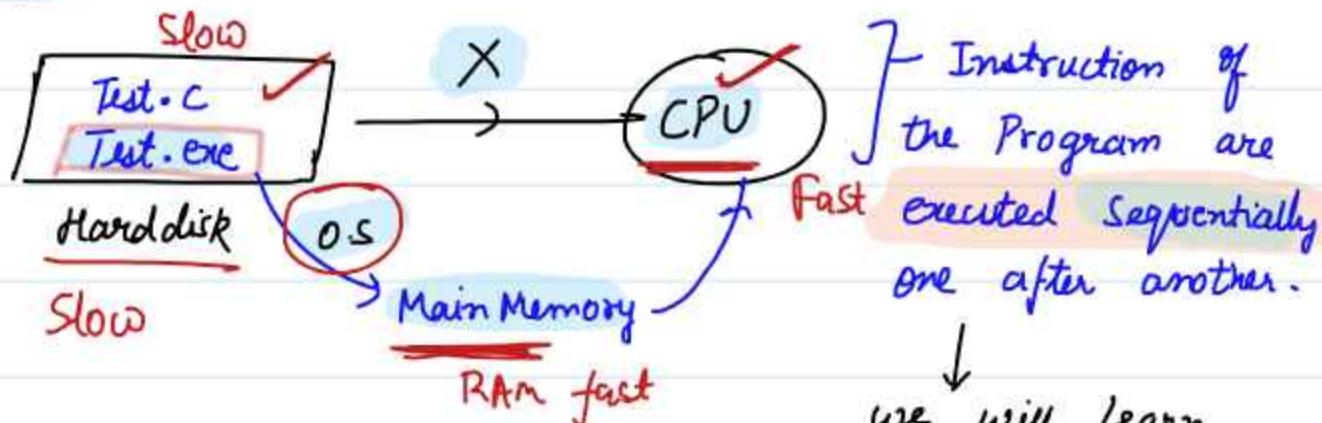
As per Von Neumann Architecture, Secondary Memory

are the part of Input Output Devices.

Q How are programs Executed in the VNA ?



- CPU can't execute the Program Directly from the Hard disk



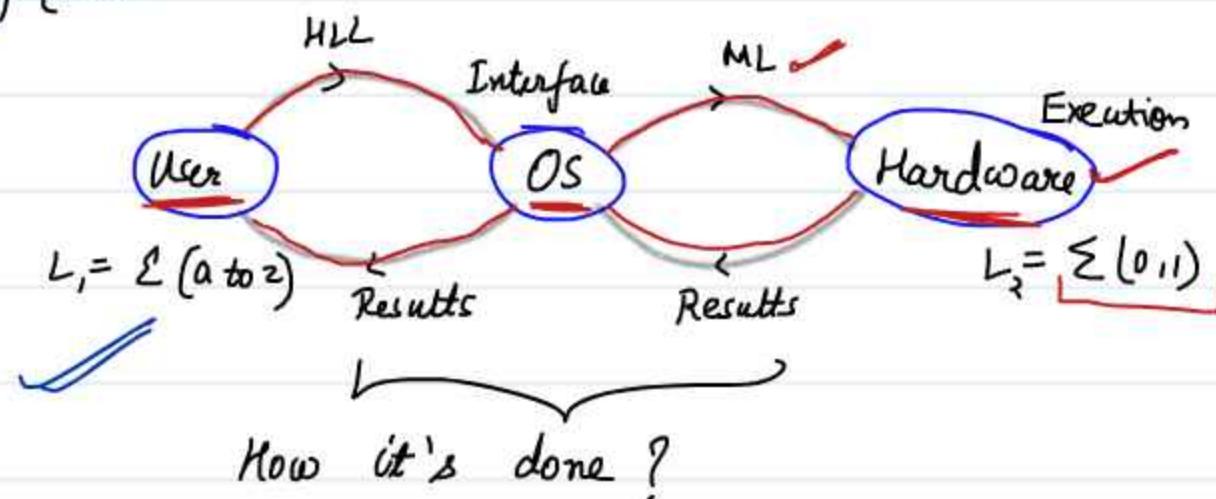
More in COA course

## Fundamental Principles of VNA.

- Stored Program Concept
- Sequential Execution of Instructions.

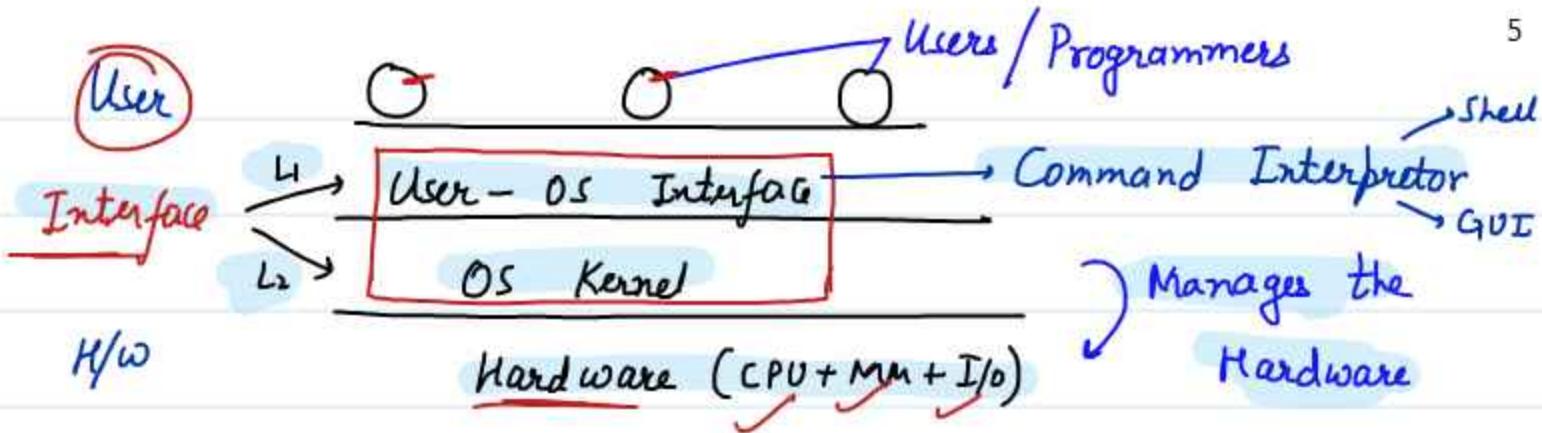


Any program that has to be executed must be stored in Main Memory (MM)



OS consists of several Modules

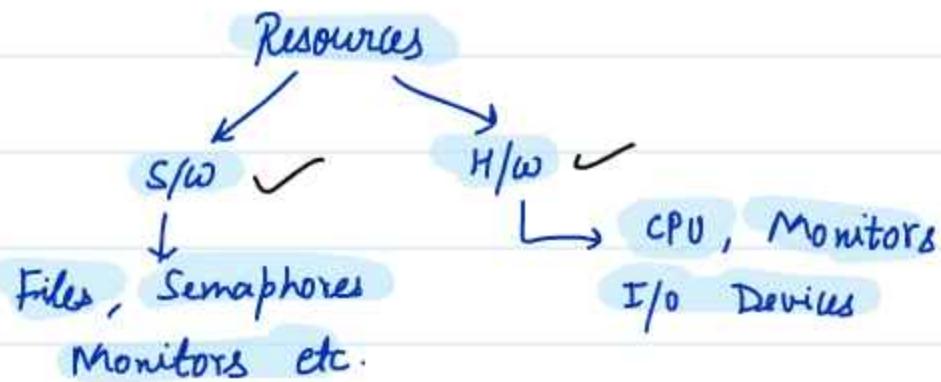




(1) Resource Manager :-



Allocation ✓  
Deallocation ✓  
Management



(2) Control Program → A program that control all the operations of the computer.

(3) Set of utilities to simplify application development.

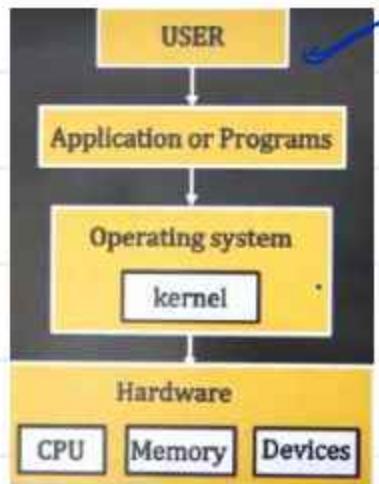
- Takes charge of Hardware
- Create env. / platform that help us to focus on dev. in HL mode.

→ OS will perform work at the back of stage

(4) Acts like a Govt. consists of several managers/modules  
OS is not a single program. OS is service provider

just like the Govt. or is Resource Manager just like Govt.

6



Goals of OS :-

Convinience

(User friendly)

Efficiency

(Utilization of Resources)

Reliability

(Goal should be achieved)

Robustness

(Strong Enough to bear Errors)

Scalability

(Ability to Evolve)

Portability. (ability to work across

diff Platforms)

Functions :-

Processor Management

Memory Management

Error Detection

Security

File Management

# Convincience is not the Primary Goal for all OS #

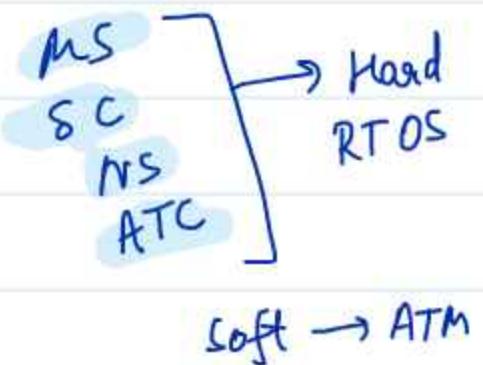
7

→ (a) Real Time System

    → Deadline Strict

} Here Reliability/Eff  
main Goal.

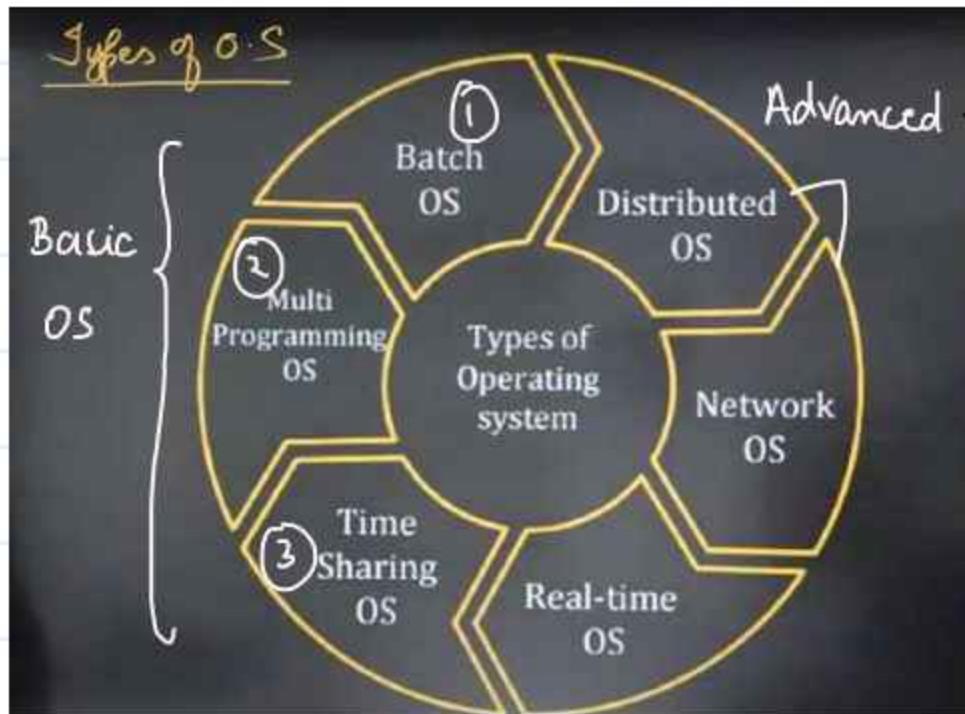
Next lecture :- Types of Os. ✓



→ Details we will study later.

1

## O.S Course - Lec-2



1<sup>st</sup> Generation → No O.S.

2<sup>nd</sup> Generation → Magnetic Tapes (No O.S)

3<sup>rd</sup> Generation → Mag. Disk [Hard disk]

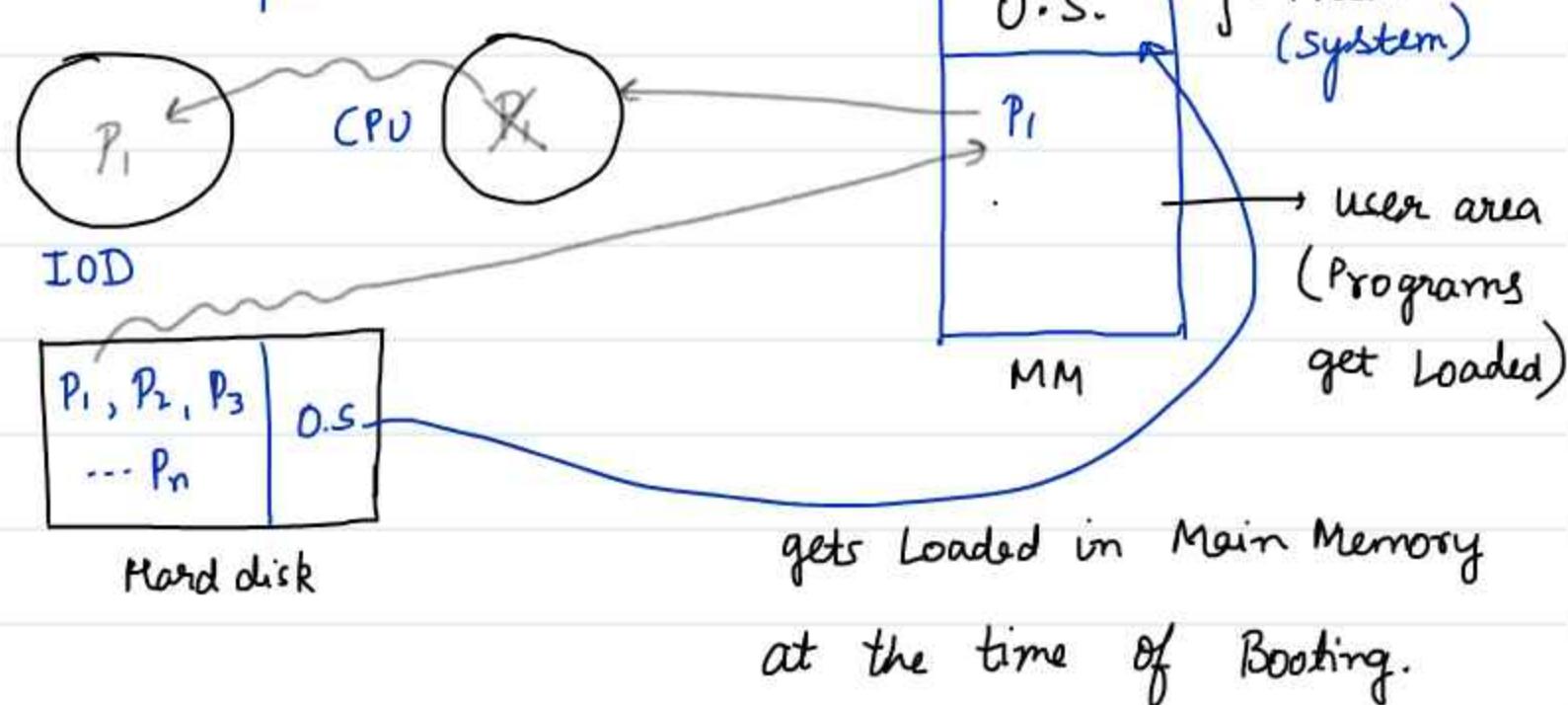
&  
Floppy

Uni  
Programmed      Multi  
Programmed

Uni Programming :- Ability of O.S to hold a single program in Memory

If one CPU at motherboard → UNI processor.

multiple CPU  $\rightarrow$  MULTI



In UniProg O.S.  $\rightarrow$  O.S can Load only a single Prog from Memory.

Less Throughput  
Less Efficiency

↑ moves to IOD then

CPU becomes idle

)

we don't want that. We want efficient CPU utilization.

Throughput = No of Programs Completed in a unit time.

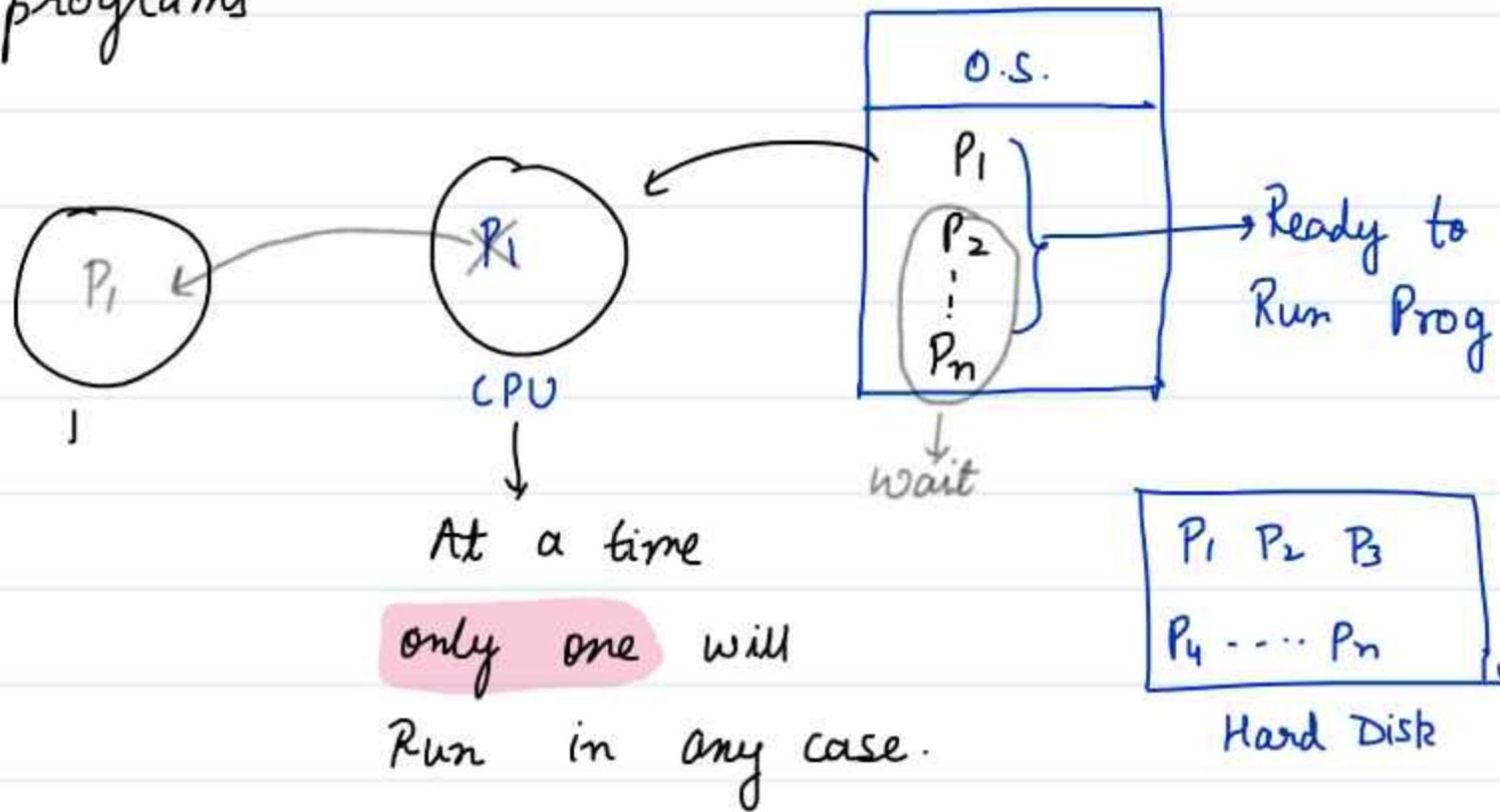
Ex :- MS - DOS [Disk operating system]  $\rightarrow$  1990

↳ Command Based / No GUI

Objective → Maximize CPU utilization

# Multiprogramming O.S. #

↳ O.S. can hold multiple Ready to Run programs

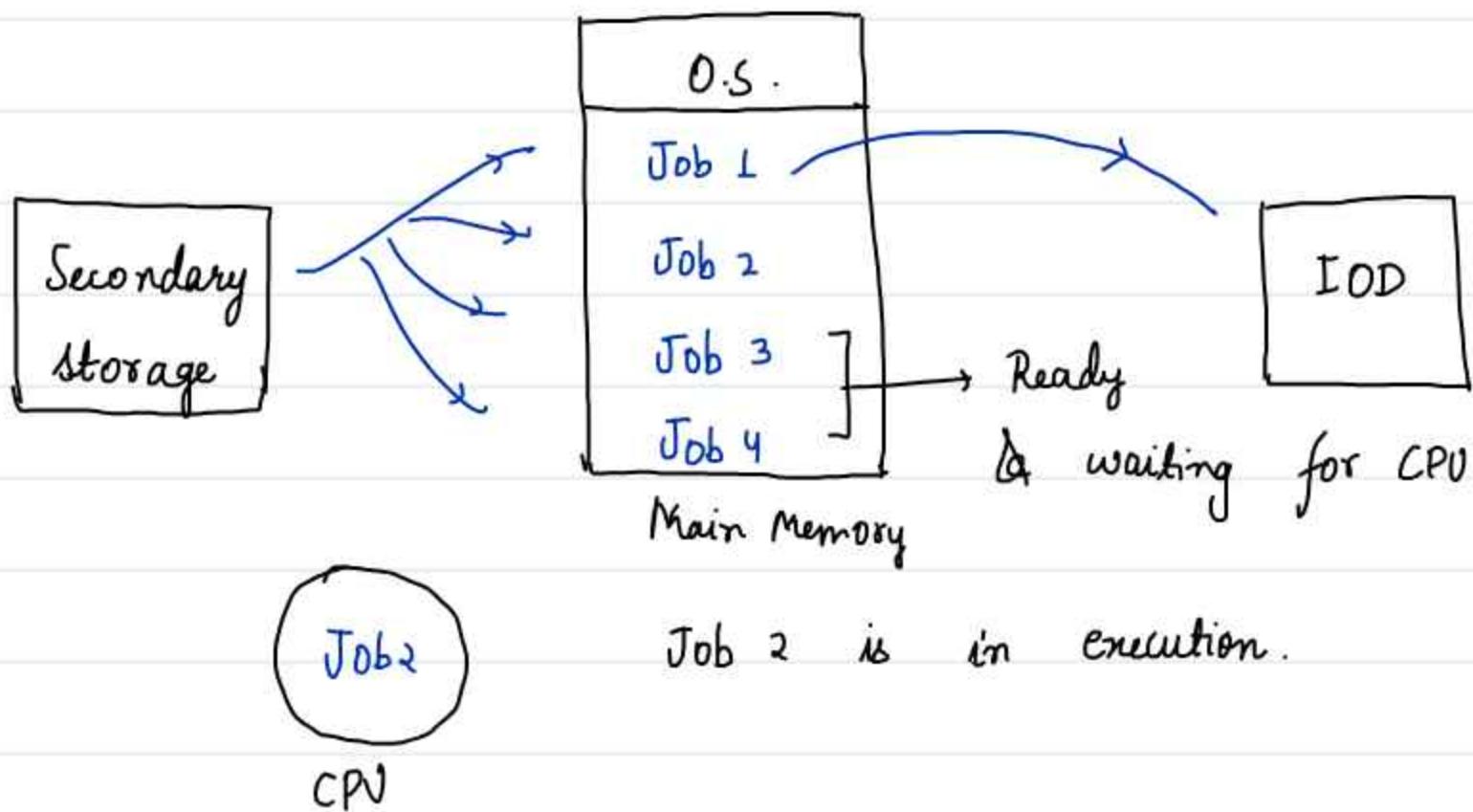


Maximize CPU Utilization, Efficiency, Throughput

Impression of Multiplexing of CPU  
among different programs

Multitasking [ Program  $\approx$  Task ]  
 Unix  
 Windos8

- Schematic View of Multiprogramming



CPU utilization is increased, idleness decreased.

### Types of Multiprogramming

Pre-emptive

Forceful deallocation from CPU

Non Preemptive (WIN 3.0)

Nobody will force the program to leave CPU

## Drawback of MPr:-



It will release the  
CPU voluntarily.

- either all instructions are complete
- Needs I/O.
- System call

- Starvation / Indefinite waiting for other programs.
- Lack of Interactivity / Responsiveness

Pre-emptive → Improves Responsiveness by

(NIN - 95)

!

WIN - 11  
UNIX  
LINUX  
MAC )

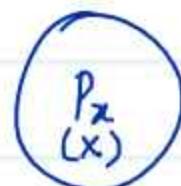
Forcefull Deallocation  
↓ by OS.

Other waiting prog. can get a chance to RUN on CPU.

## Pre-emption



Let's say OS has given 10 sec P<sub>x</sub>. neither completed nor went for I/O.

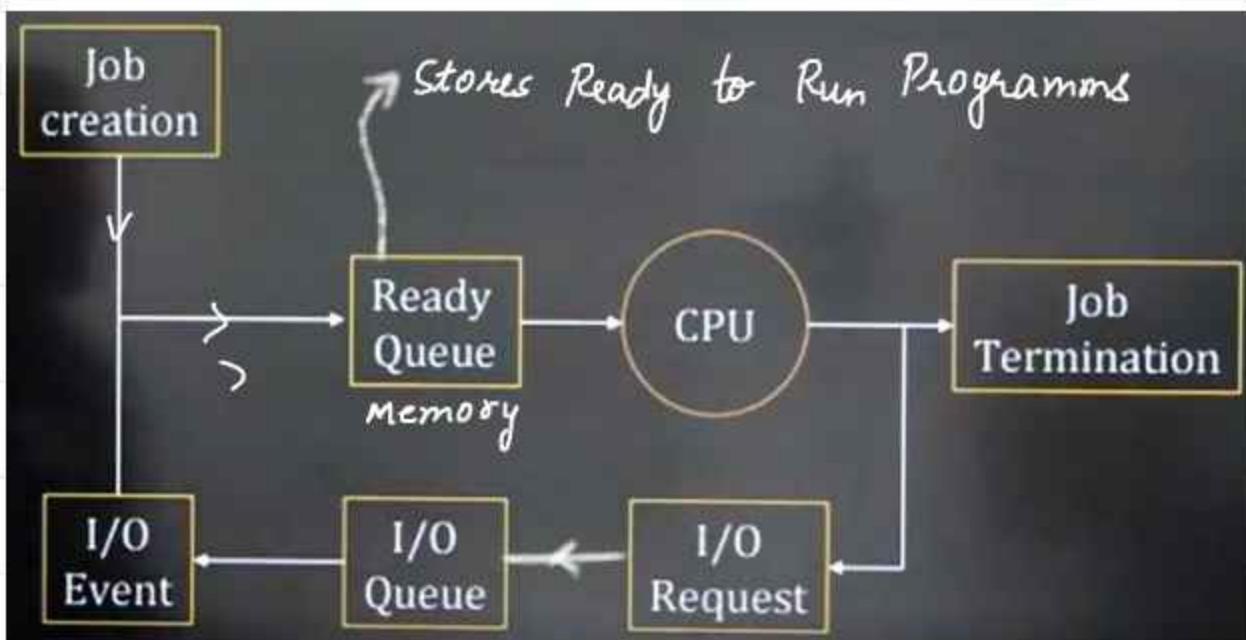


P<sub>y</sub>(Y)

If Y > X  
then OS Removes P<sub>x</sub> from CPU & let P<sub>y</sub> enter the CPU.

These O.S are called as  
Multiprogrammed Time Sharing O.S.

Multitasking O.S :- Pre-emptive based MPr. O.S.



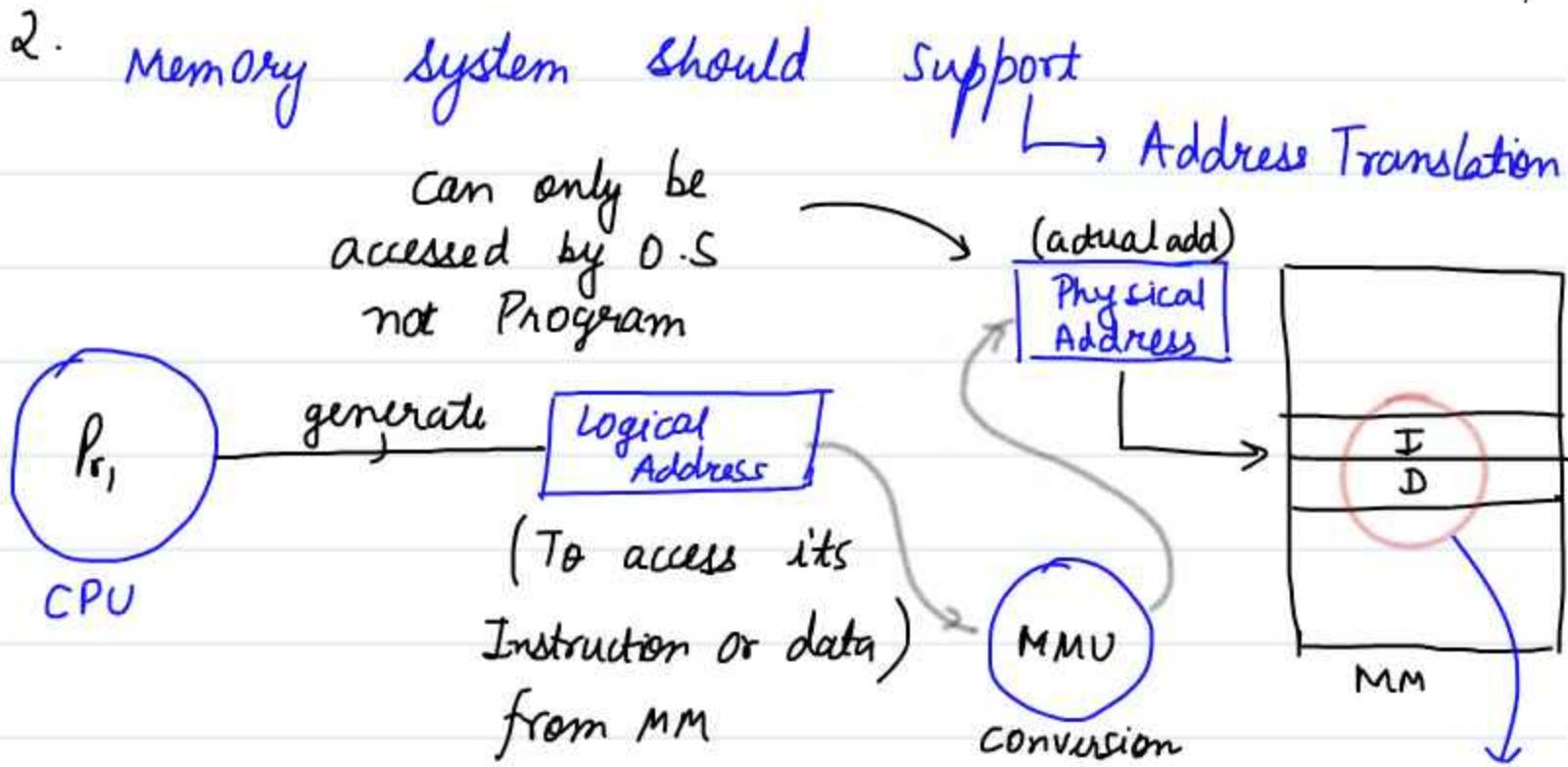
(H/w)

# Architectural Requirement for Implementing a  
Multiprogrammed O.S. #

1. Secondary Storage Device (IO) :- DMA compatible



Efficient data transfer b/w  $\leftrightarrow$  Direct Memory Access  
secondary storage & MM. (Learn More in COA course)



Why two addresses were needed?

Instructions

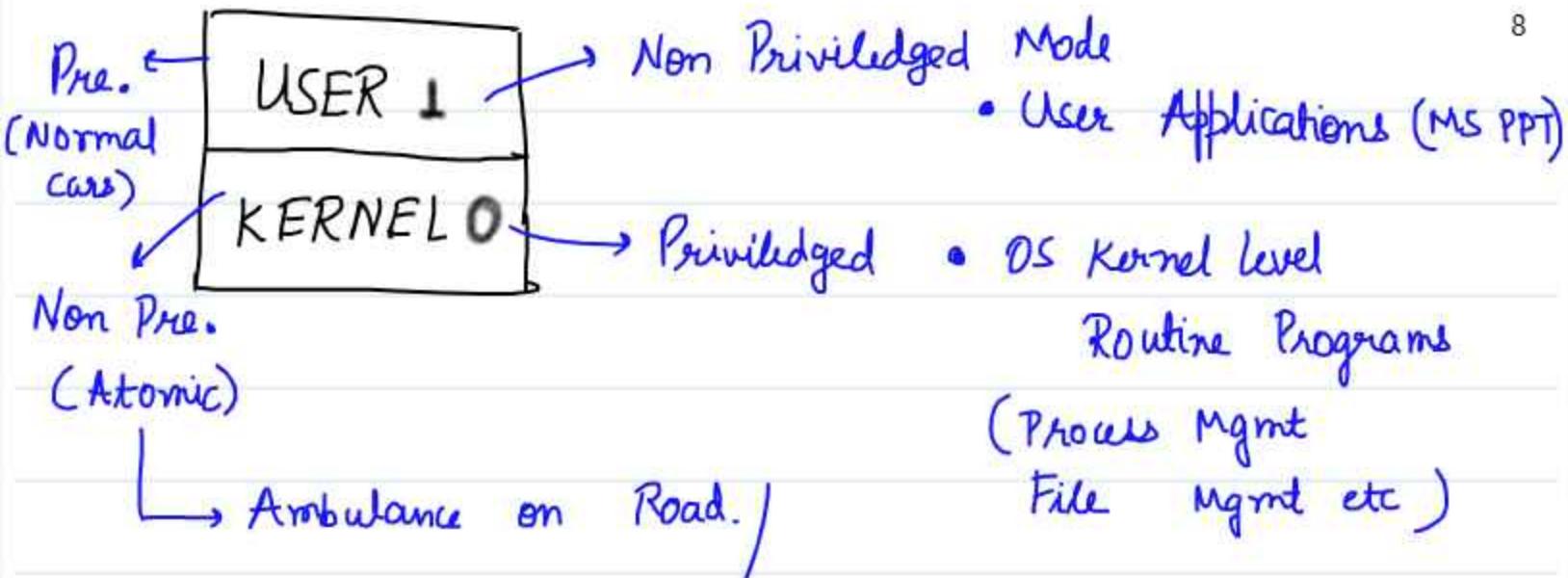
Q Data unit

#### Security:

- With one address (physical), each program would directly access physical memory locations. A buggy program could overwrite memory used by another program, corrupting data and causing crashes. Logical addresses prevent this by creating an abstraction layer. Programs operate with logical addresses, and the OS translates them to physical addresses, ensuring programs don't interfere with each other's memory space.

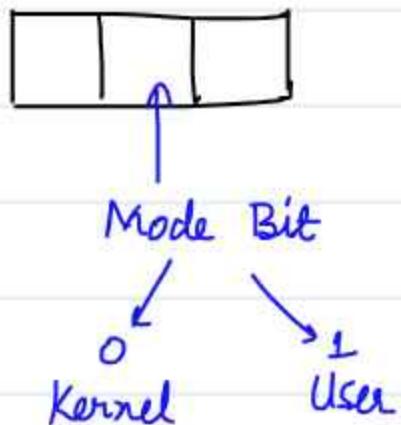
User & Kernel Mode

3. Processor (C.P.U.)  $\Rightarrow$  Dual Mode Operation should be supported



Convoy of Chief Minister

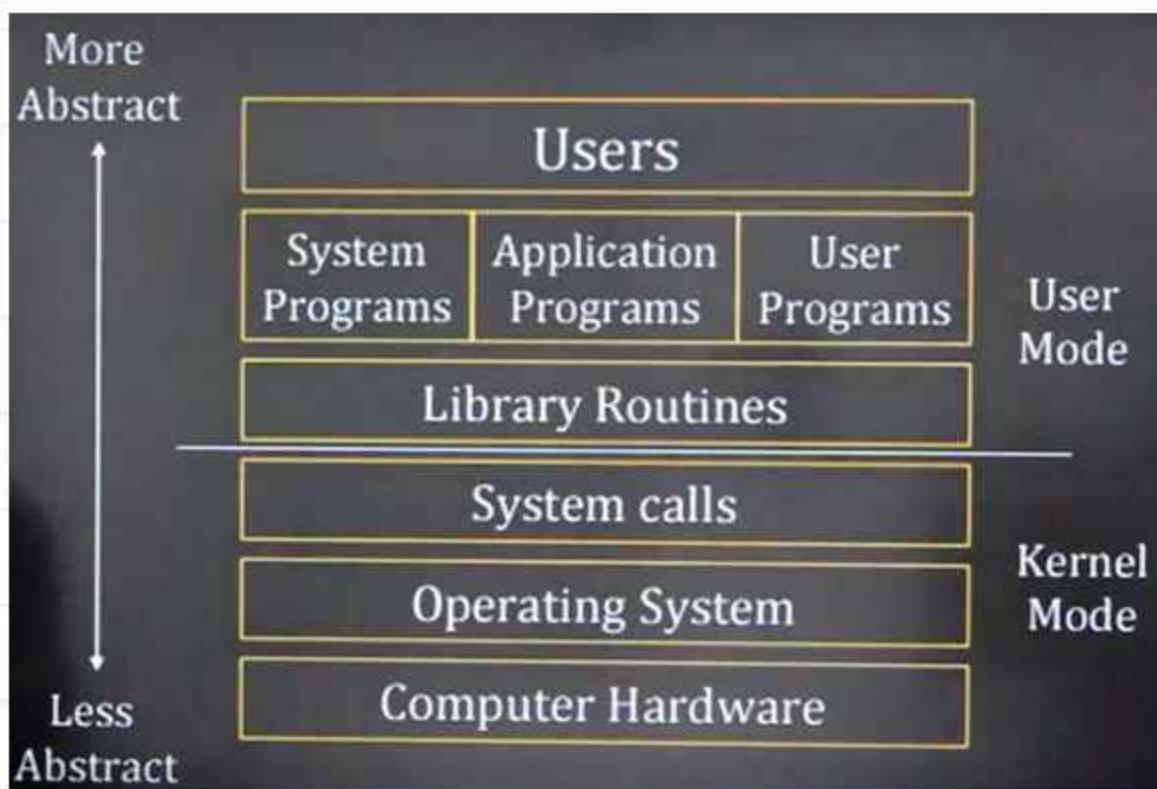
There is a Register in CPU :-



Many times it becomes necessary to shift Modes :-  $U \rightarrow K$  or  $K \rightarrow U$

Eg:-

If some User application wants to avail OS Kernel services then Mode Shifting is Must.



### Kernel Mode

In Kernel mode, the executing code has complete and unrestricted access to the underlying hardware. It can execute any CPU instruction and reference any memory address. Kernel mode is generally reserved for the lowest-level, most trusted functions of the operating system. Crashes in kernel mode are catastrophic; they will halt the entire PC.

### User Mode

In User mode, the executing code has no ability to directly access hardware or reference memory. Code running in user mode must delegate to system APIs to access hardware or memory. Due to the protection afforded by this sort of isolation, crashes in user mode are always recoverable. Most of the code running on your computer will execute in user mode.

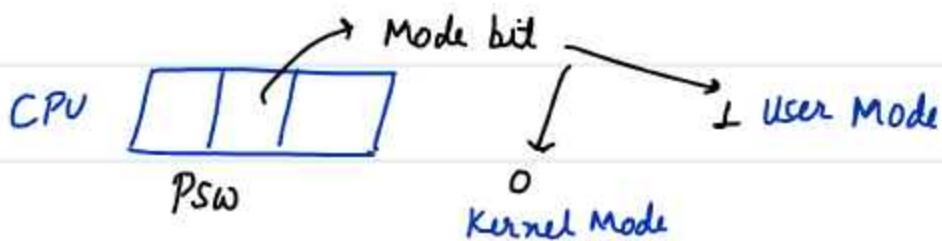
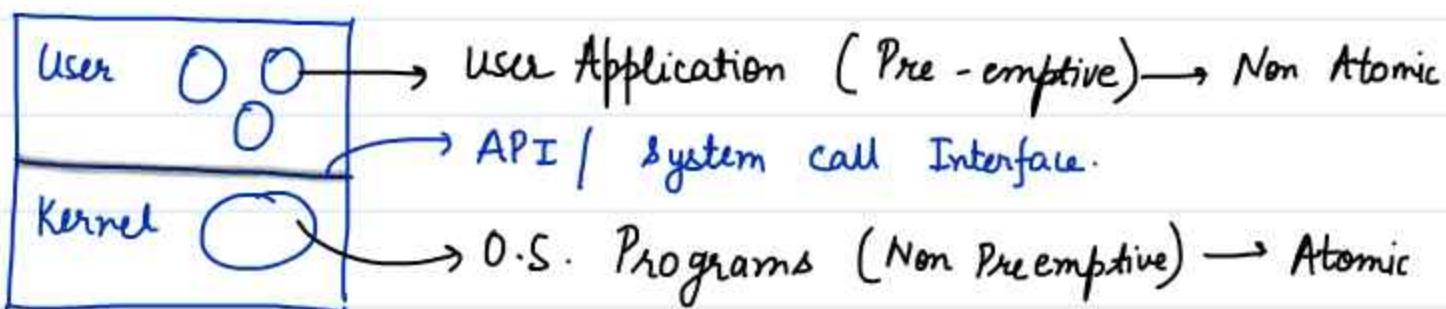
Next class :- How Mode Shifting is done .

Interrupt ?

Fork ?

End of Chapter - 1 [ Intro & Backg.]

## Lecture - 3



OS → service provider

User Programs → service user

Whenever we need some OS service we need to shift from **User Mode** → **Kernel Mode** & after the completion of that service, we need to switch back to **Kernel Mode** → **User Mode**



Mode Shifting is necessary to avail OS services.

API :- Application Programmer's Interface

→ tells us what OS. services we can avail.

## # Implementation of Mode Shifting via API/SCI #

Ex:- main()

```
{ int a, b, c;
```

1. b = 1;

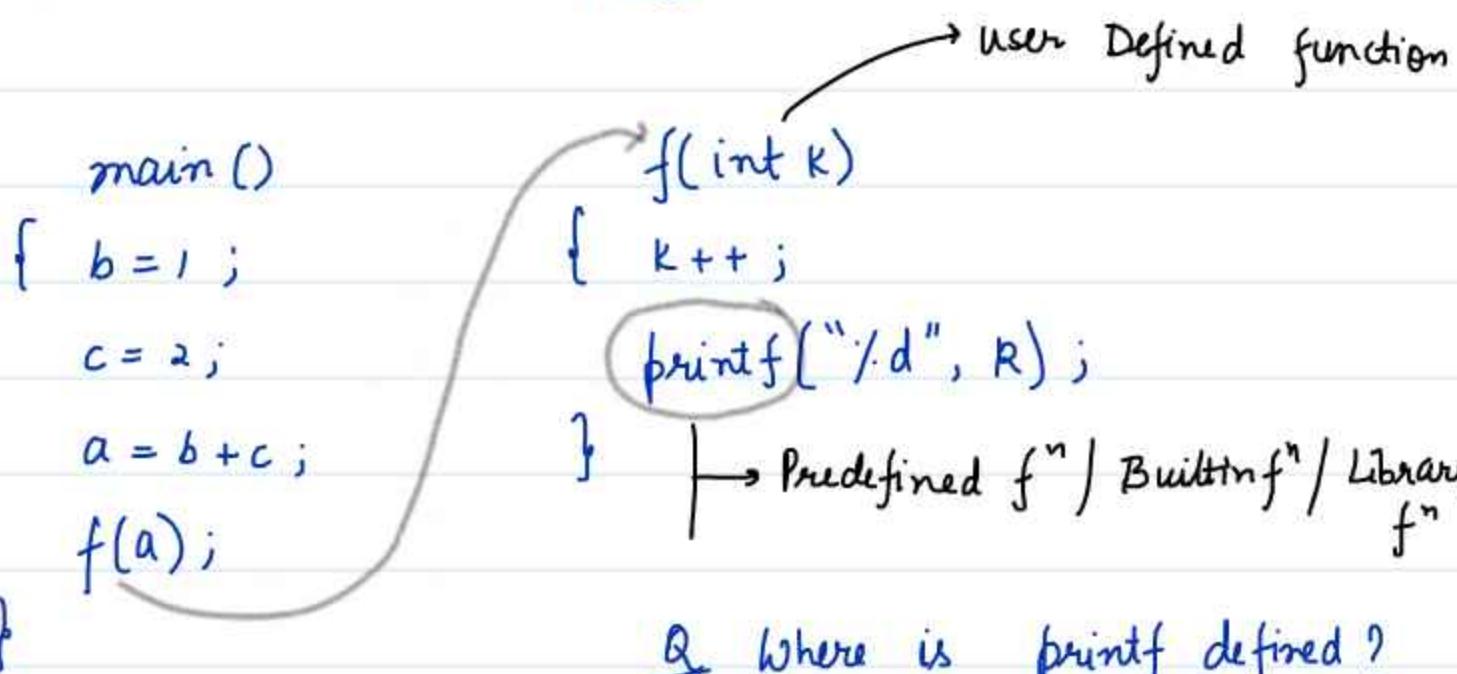
2. c = 2;

3. a = b + c;

```
}
```

]

→ A user program like a  
c program run in kernel  
Mode.



Q Where is printf defined?

→ Header file? ] XX

→ Like stdio.h contains

Learn this more ← Type checking ← f" declaration (prototype)  
in C/C++ DSA course , only not implementation  
Syntactic correctness

proper argument passed or not.

So The Implementation of printf is in Library files.  
(.lib)

Anyway whether it's a predefined f<sup>n</sup> or user defined functions all are executed in **USER MODE**.



simply because they are written by **USER**.

\* Compilers are not the part of O.S. \*

So the above program didn't require Mode Shifting.

```
main()
{
    b = 1;
    c = 2;
    a = b + c;
    f(a);
    fork();
    printf("%d", a);
}
```

f( int k )  
{     k++; }

What is fork & what does it do ?

- Execution of fork
- Results in creating child Processes.
- function (special)
- system call [ f<sup>n</sup> called to O.S.]
- fork is defined / imp. in O.S. Kernel.
- fork is O.S. Routine

How to avail fork() service offered by O.S.?

CT :- Compile Time      BSA = Branch & save Address

main () {

Non Privileged Instruction  
Learn More in COA course.

    |  
    | CT → BSA → This Instruction is executed in UM.

    | f(); CT →

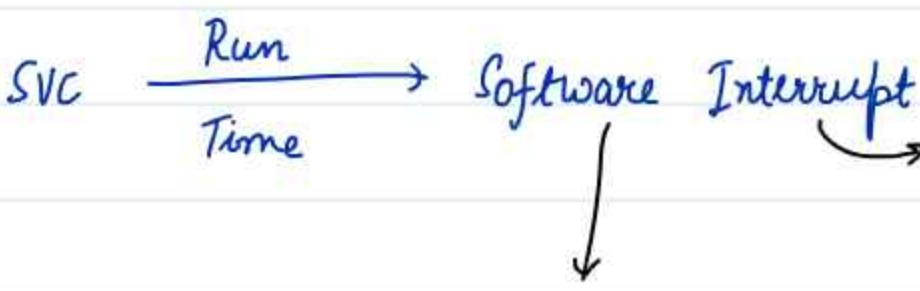
    | fork(); SVC → This Instruction is executed in KM

}

    | Privileged Instructions / soft. Interrupt  
    | SVC = Supervisory call. Inst.

High level Program

CT → Instructions



↓  
Inform O.S. of some activity

Every Intr. has a Routine to serve it → I.S.R.

Intr. service Routine

Interrupt

S/W

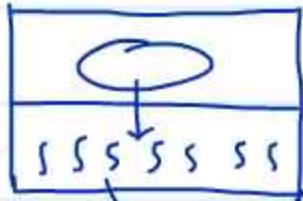
H/W

Ex:- Run  
Time error

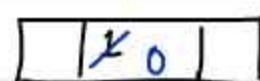
Intr. by  
Hard disk

Like divide by 0.

Activity ↓ by I.S.R.



How to find where is our fork?



Mode  
Bit change

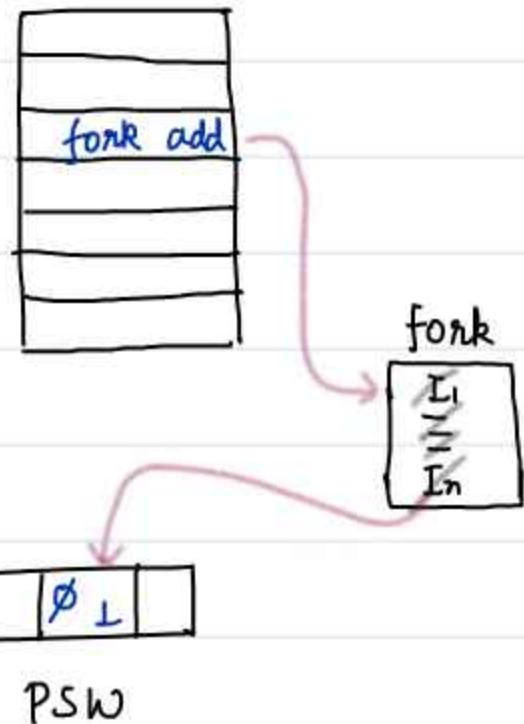
Activity ↓ by I.S.R.

O.S. Maintains a table in Kernel

known as DISPATCH TABLE

↓ (data structure in RAM)

Tells what all services O.S. provides.



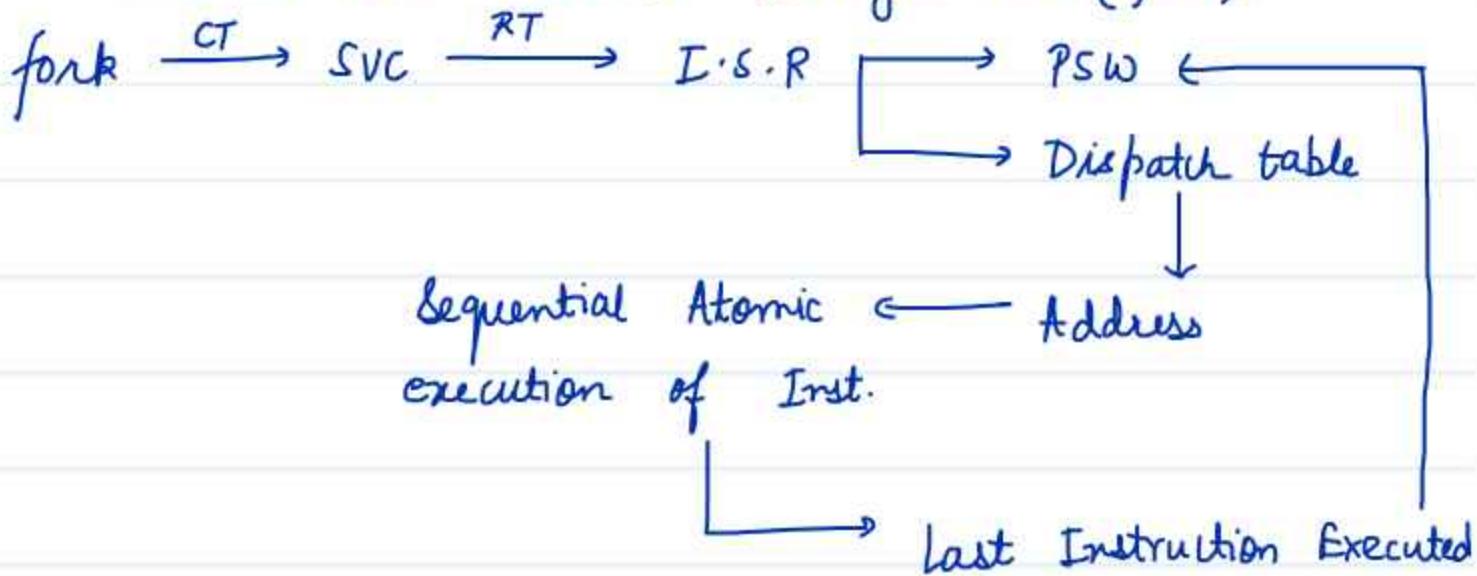
All O.S.s follow this paradigm.

Functions

- User defined
- Built-in (Predefined)
- System calls

UM  
KM

O.S. Routines are accessed through API (fork)



printf Implementation → Library file

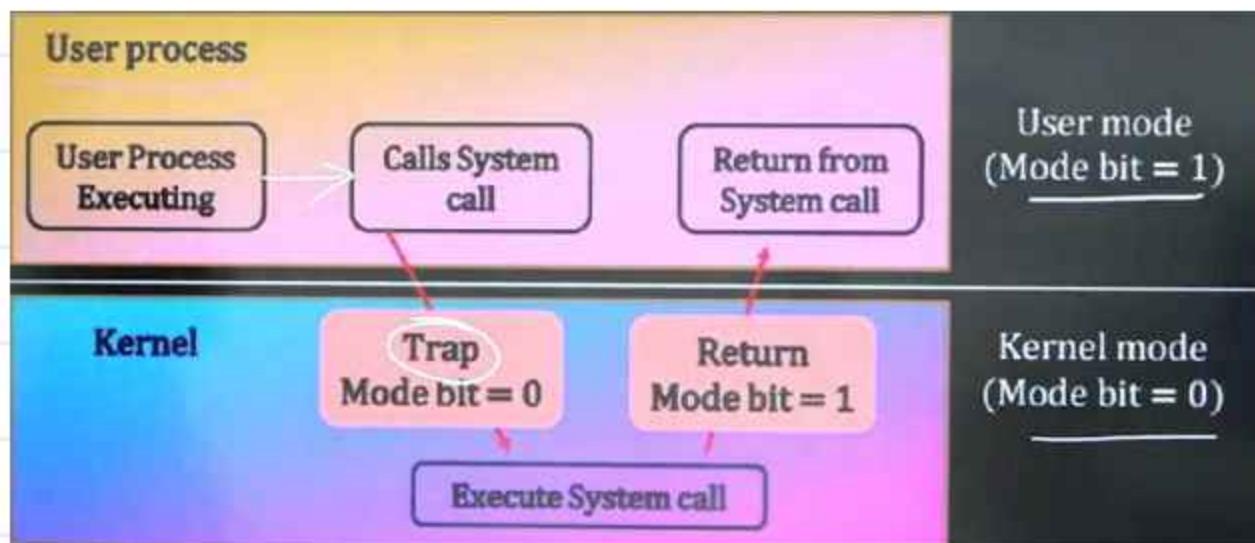
printf() → wants to access the device & device control is with operating system.

{ ; } VM

↓ write(); → system call } → Library fn like printf can also use system calls.

Mode      }

Shifting will occur.



No interrupt is generated during Kernel → User

So we need just a privilege instruction without interp. to switch from Kernel to User Mode.

## Section - 2 - Process Mgmt

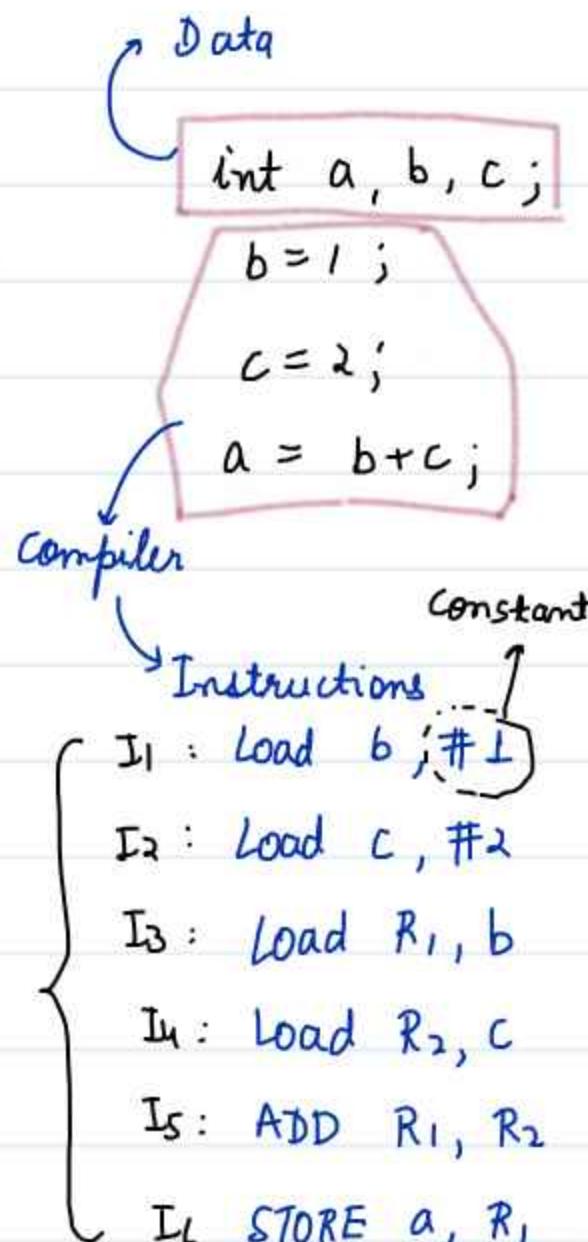
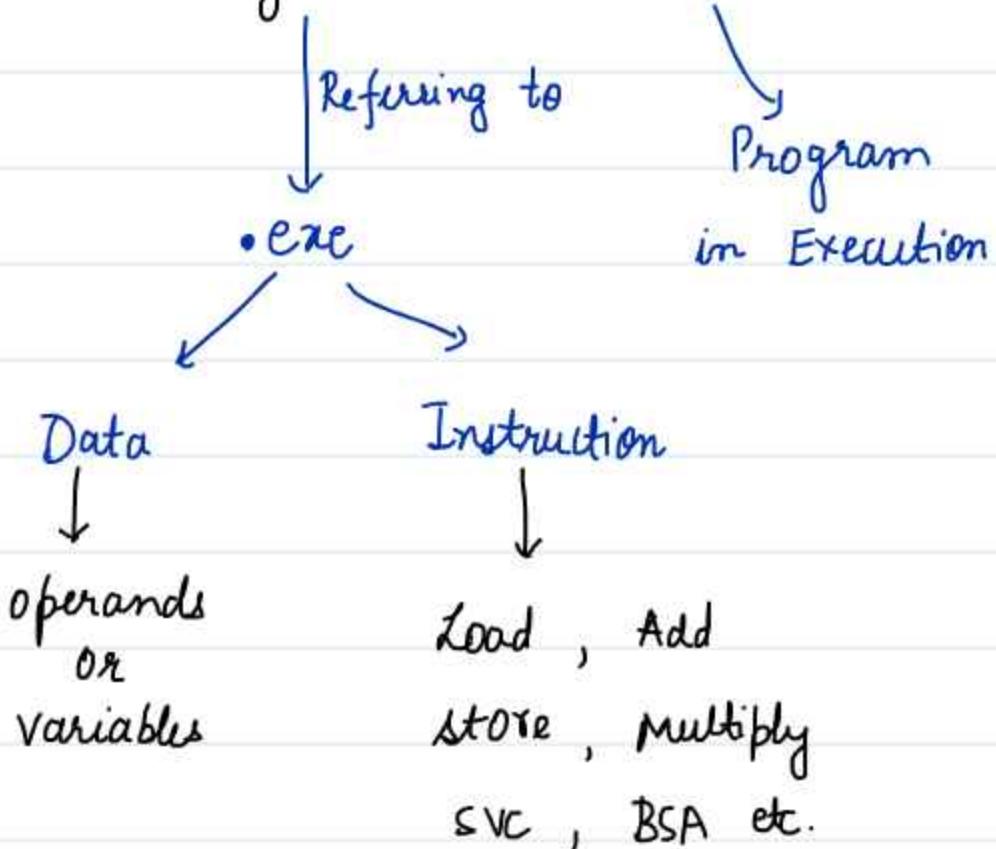
Its instance is

a Process.

### Lecture - 6

when Program is loaded from disk to main memory.

Program Vs Process



so Process is something which is created from program.

Data → Static (Fixed size / Known size)  $\Rightarrow$   $\text{int } a;$  2 bytes

(Varying / Fixed size) → Dynamic (Memory allocated at Runtime)  $\text{int } a[10];$  20 bytes

At what time of program execution, memory allocation occurs of static data?

→ ~~Compile Time ?~~

→ Meant for checking syntactic correctness & generating the code.

what if I just compile the program & don't run it? Memory allocated at CT is a wastage.

Compiler only decide how much.

→ (Before Run Time)  
Load Time ✓ is the correct answer.

When the program is loaded from disk to memory for execution.

int n, A[n];  
scanf ("%d", &n);

Is this valid in C?

↓

NO!

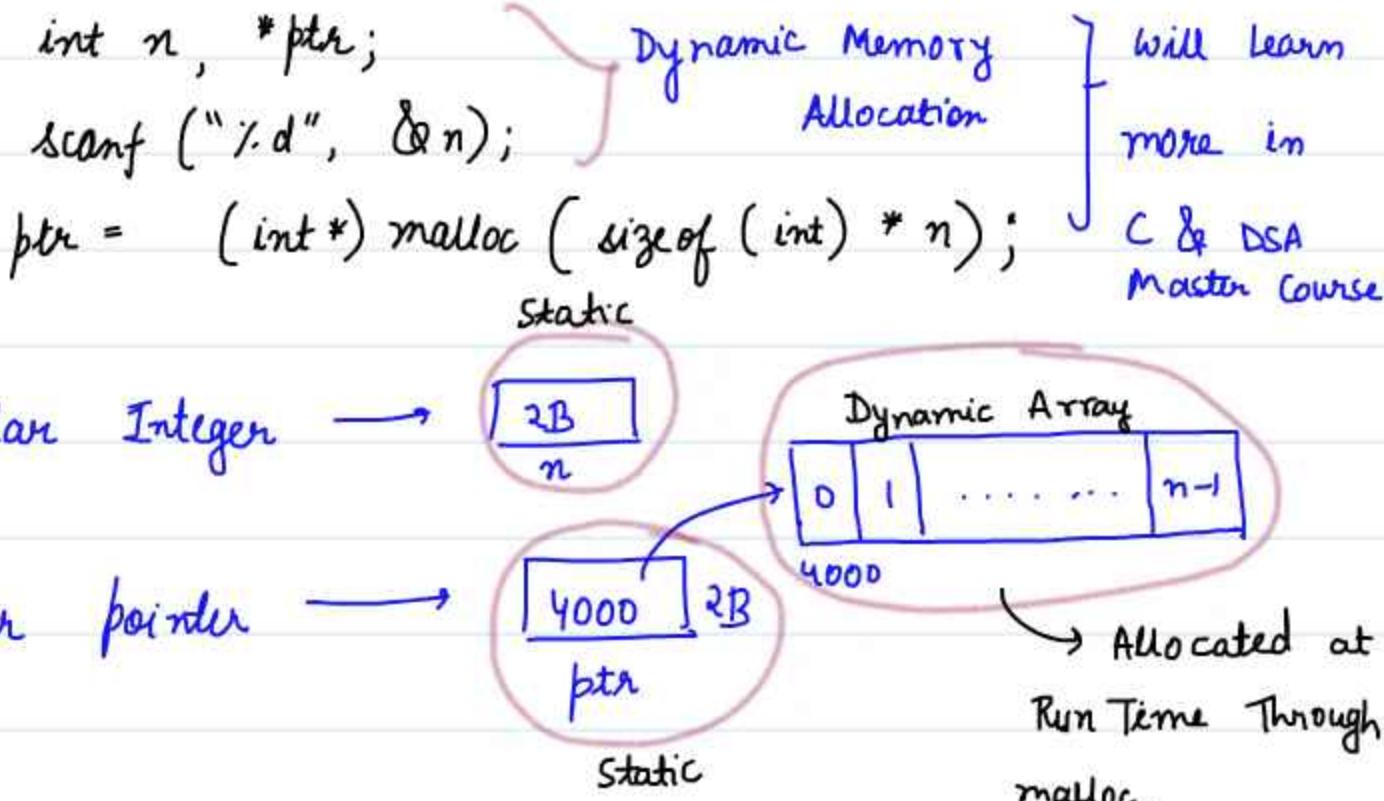
Not allowed in

C

Dynamic array like this

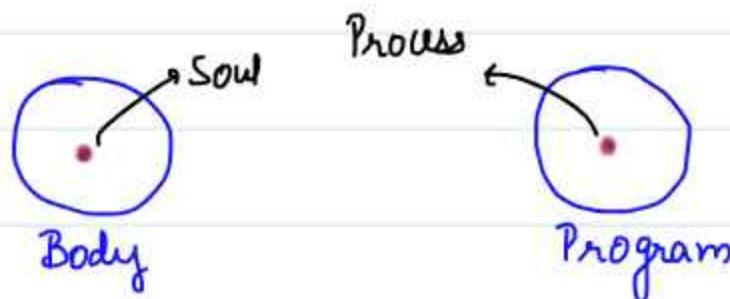
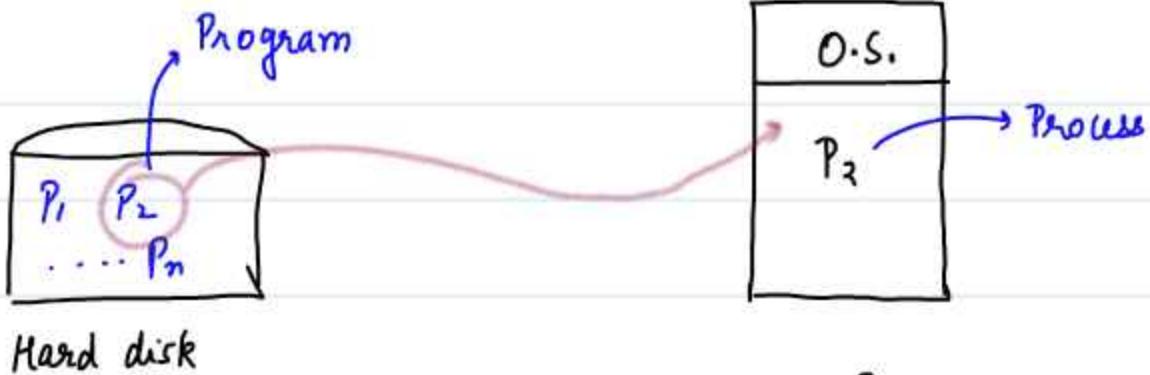
(Allowed in C++)

We can create dynamic array of size  $n$  alternatively :-



As object is to class  
Process is to Program ] Relationship

- Process :-
- Program in Execution
- Program when it get loaded in Memory
- Instance of a Program
- Active Entity / Program is Passive (doesn't use any Resources)
- Always in Memory
- Locus of Control of O.S. → People are the LOC
- Animated Spirit. for the Government



# Let's Learn from Developer's Perspective #

→ Process is an ADT (Abstract Data Type)  
 → simply a Data Structure

Each Data Structure have four parts :-

< Definition, Representation/Implementation, Operations, Attributes >

Ex :- Linked list

Aspect	Description
Definition	A linked list is a linear data structure where each element (node) contains data and a reference to the next.
Representation/Implementation	Implemented using nodes with data and a reference to the next node.
Operations	Insertion, deletion, traversal.
Attributes	Head, tail, size/length, data, next, null.

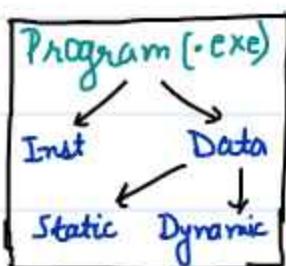
Now in the same way, we need to define Process:-

< Definition, Representation/ , Operations , Attributes >

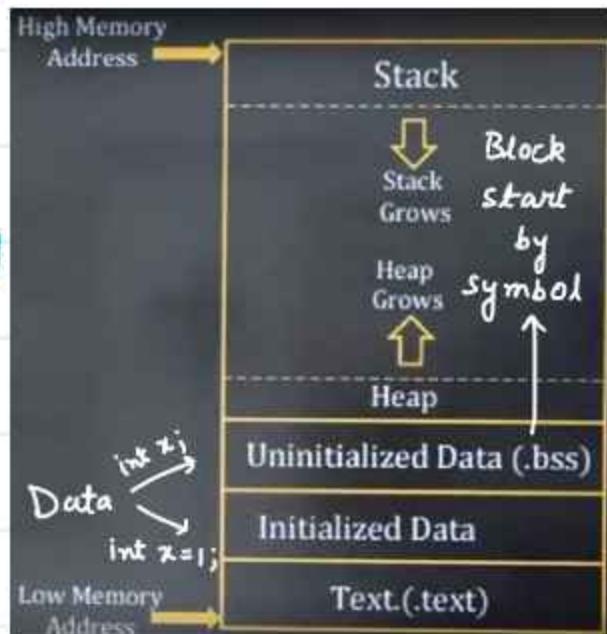
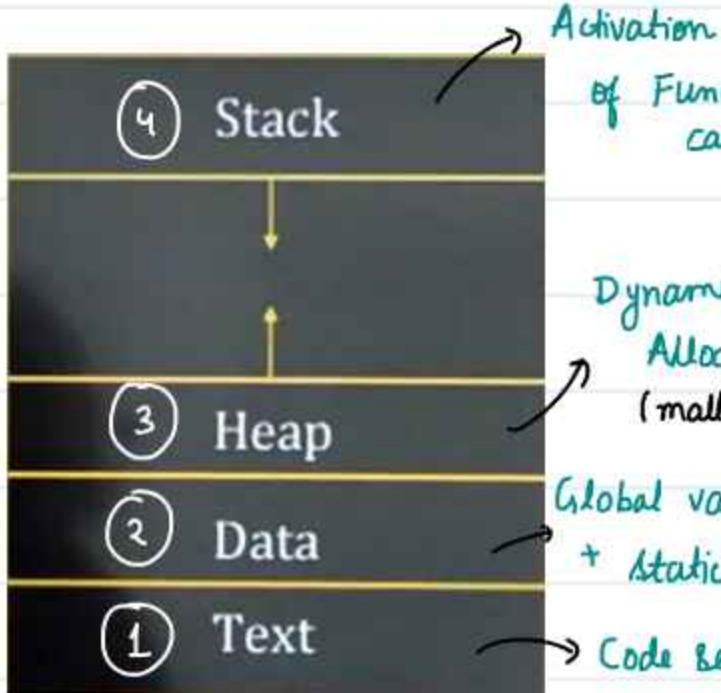
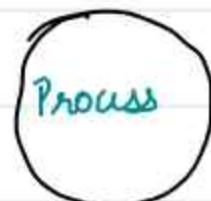


Implementation

How does process look like in Memory?

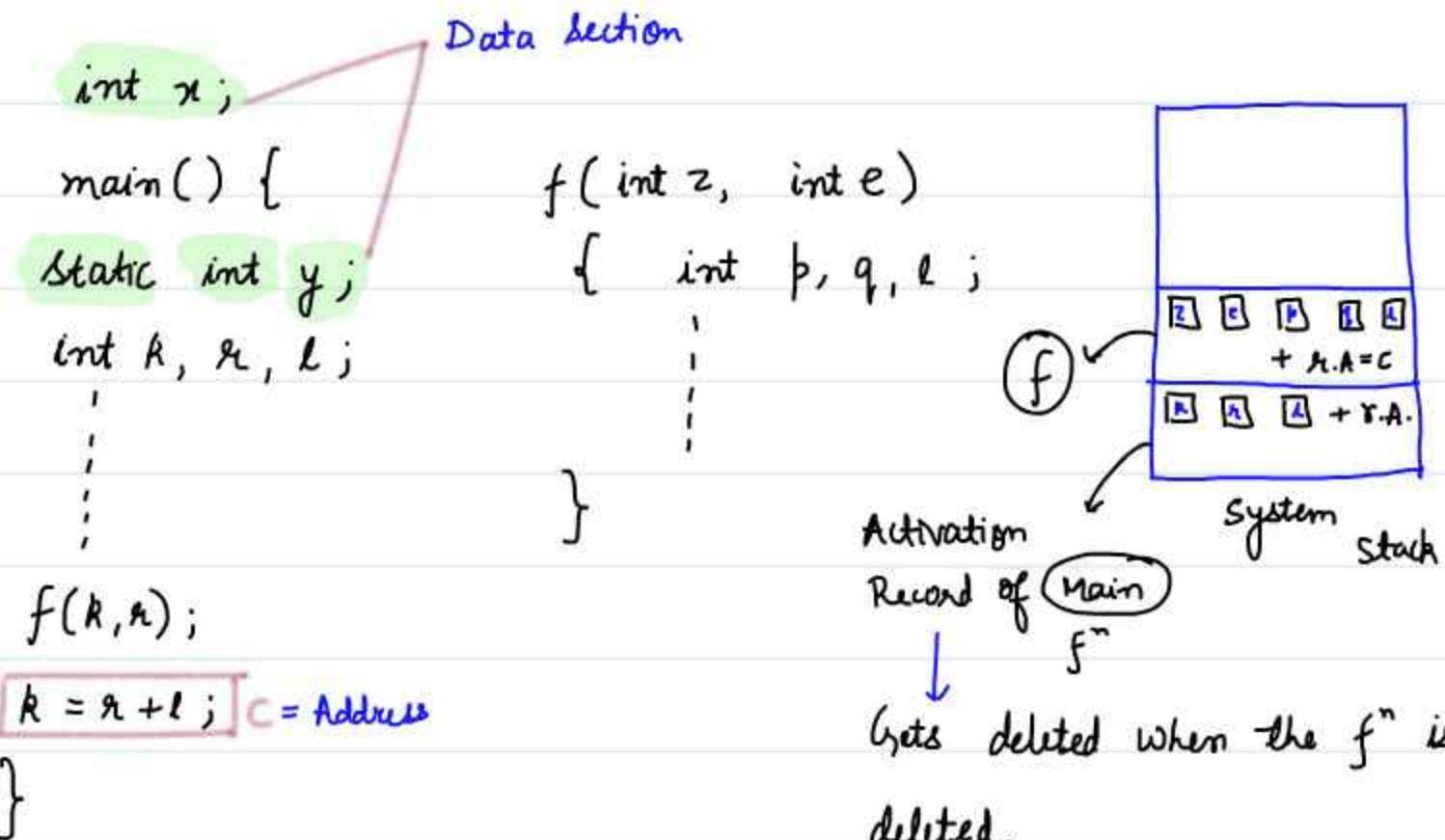


creates



Activation Record doesn't contain the address of the f.

Information & space of Local variables  
Return Address of that function } of a function



## # Process Operations :-

- create() : Resource allocation & resources which are necessary for process are allocated.
- schedule() : the act of selecting process to run on CPU
- execute() / Run() : Executing Inst. from Code section.
- block() / wait() : Process will get blocked when it'll execute system call / I/O opn.

→ `suspend()` :- Sometime we need to move process from memory to disk.

$\vdash \text{resume}() :- \dots$  " Disk to memory.

→ terminate() :- Resource deallocation

## # Attributes :-

Process id

Parent Process id

→ Group id

Identification :-  $\langle \text{Pid} ; \text{PPid} ; \text{gid}, \dots \rangle$

CPU Related :-  $\langle$  PC ; Priority ; State ; Burst Time ....  $\rangle$

Program counter

## Level of Importance

(will point to the next instruction to be executed)

Memory Related :- < size ; Limits ..... > of memory

File Related :- < List of files in use >

## Device Related

11

## Accounting Related etc.

All these

attributes are stored in PCB (Process Control Block)

ID Card of Process	
Pointer	<u>Process State</u>
	<u>Process number(id)</u>
	Process counter
	Registers
	Memory Limits
	List of open files

PCB :- Contains attributes of Process.

Each process has its own PCB.

PCB is stored in Memory.

In the next Lecture, we will learn about :-

1. Scheduling Queues
2. Queuing Diagrams
3. Schedulers & Dispatchers

# Lecture - 8

Pointer	<Process State>
Process number (id)	
Process Counter	
Registers	
Memory Limits	
List of open files	
...	

PCB

PCB ( Process Descriptor) → stored in Memory  
 Total content of PCB = Process Context/  
 Environment

During the Process life time, Process goes from one state to another.

New :- gets created, Resource Allocation

Ready :- Ready to Run on CPU

Running :- Process is executing instructions on CPU.

Block/Wait :- Process need to perform I/O / Execute system call

↳ leaves the C.P.U.

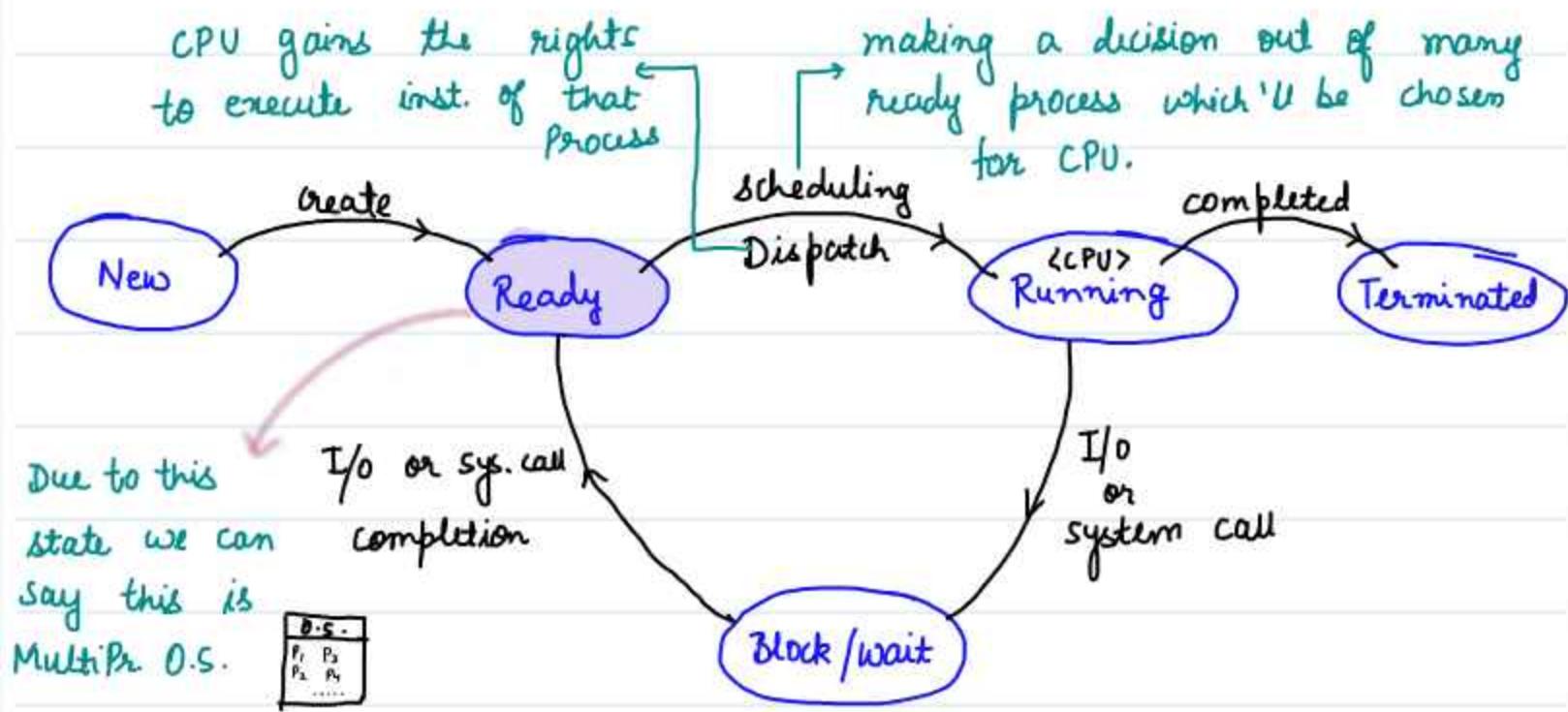
- Reading something from Device
- Writing Something from Device

Terminate :- Process has completed all of its instructions

↳ Resource deallocation

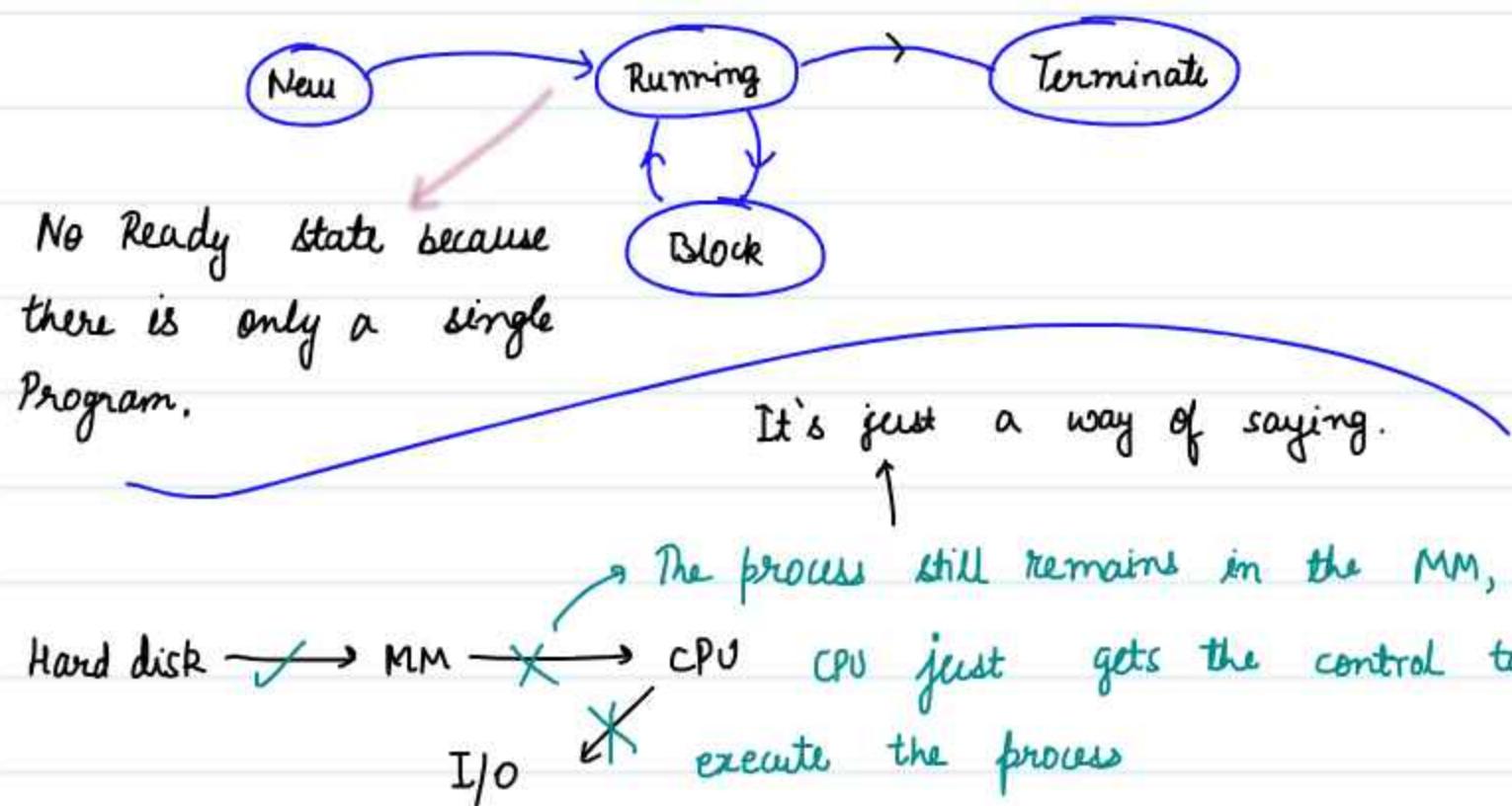
We will represent the transition b/w states with :-

# PROCESS STATE TRANSITION DIAGRAM



Scheduling & Dispatch :- Selecting & giving control

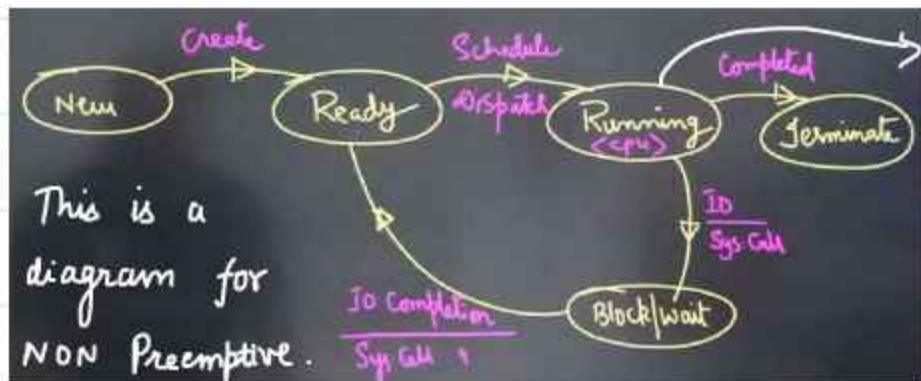
State Transition Diagram for UniProgramming O.S. :-



Every process will complete from Running state.

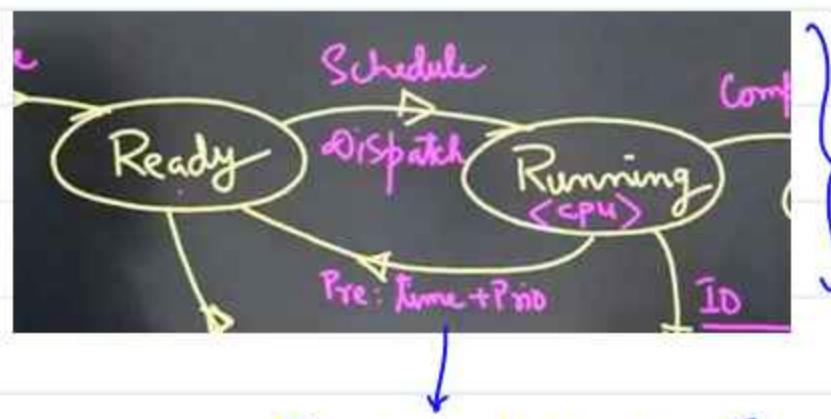
→ MUST REMEMBER

We will write "exit" in the end of program that statement must be executed on CPU to terminate the process.



Process will leave the CPU  
Voluntarily on its own.

- either it completes
- or it needs I/O or sys. call



} Pre-emptive based MultiPr. O.S.

Forcefull deallocation } Now it has become Pre-emptive.

Notes :-

1. Either Process is in Ready / Running / Blocked

New → Entering the MM

It's in the Main Memory.

Terminate → Leaving the MM.

2. There can be multiple process in Ready & Blocked state

but there can be at max 1 Running Process.  
for one CPU.

- Max Process in Ready | Blocked :- Theoretically  $\infty$
- Max Process in Running state :- No of CPU's.

3. Let's say there are 120 Ready Programs in MM but O.S. can accomodate only 100. So OS will transfer the rest 20 from MM to Hard disk to improve performance. → This is SUSPENSION



Take the Example of Classroom.

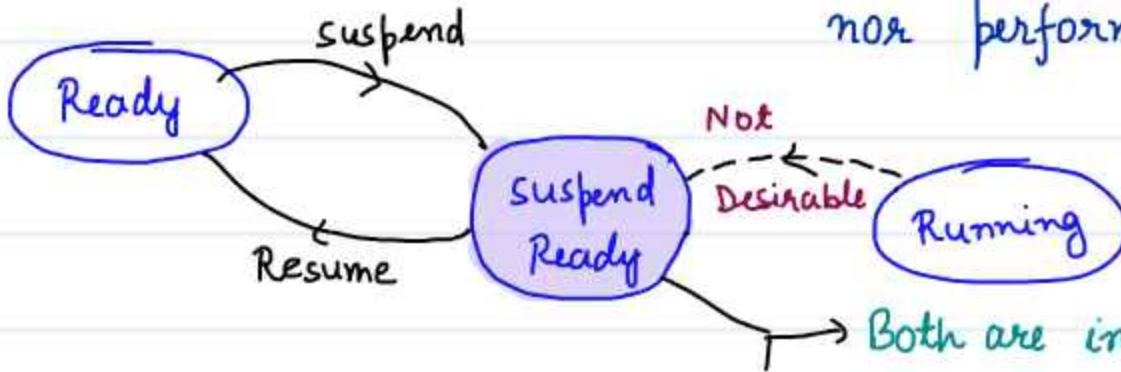
We can suspend from → Ready | Running | Blocked  
(why not New & Term.)



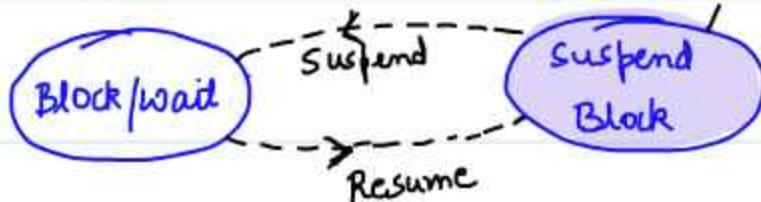
But the most desirable state to suspend is :- READY.

Process is Not in MM in these states

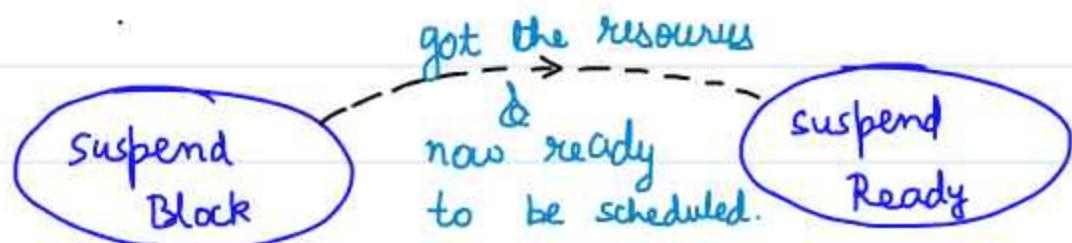
→ Process is neither Running nor performing I/O.



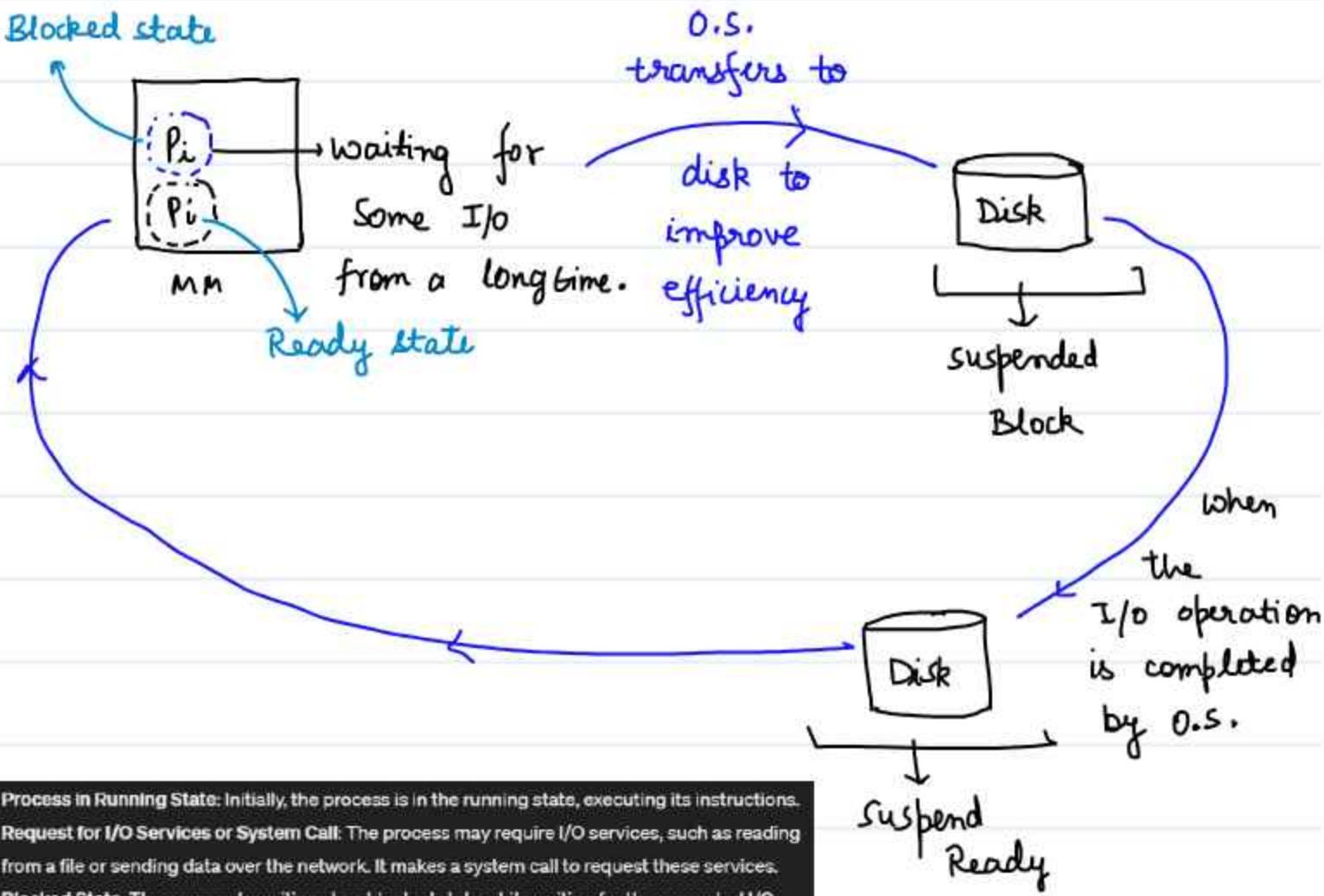
Both are in DISK



Possible but  
Not desirable.

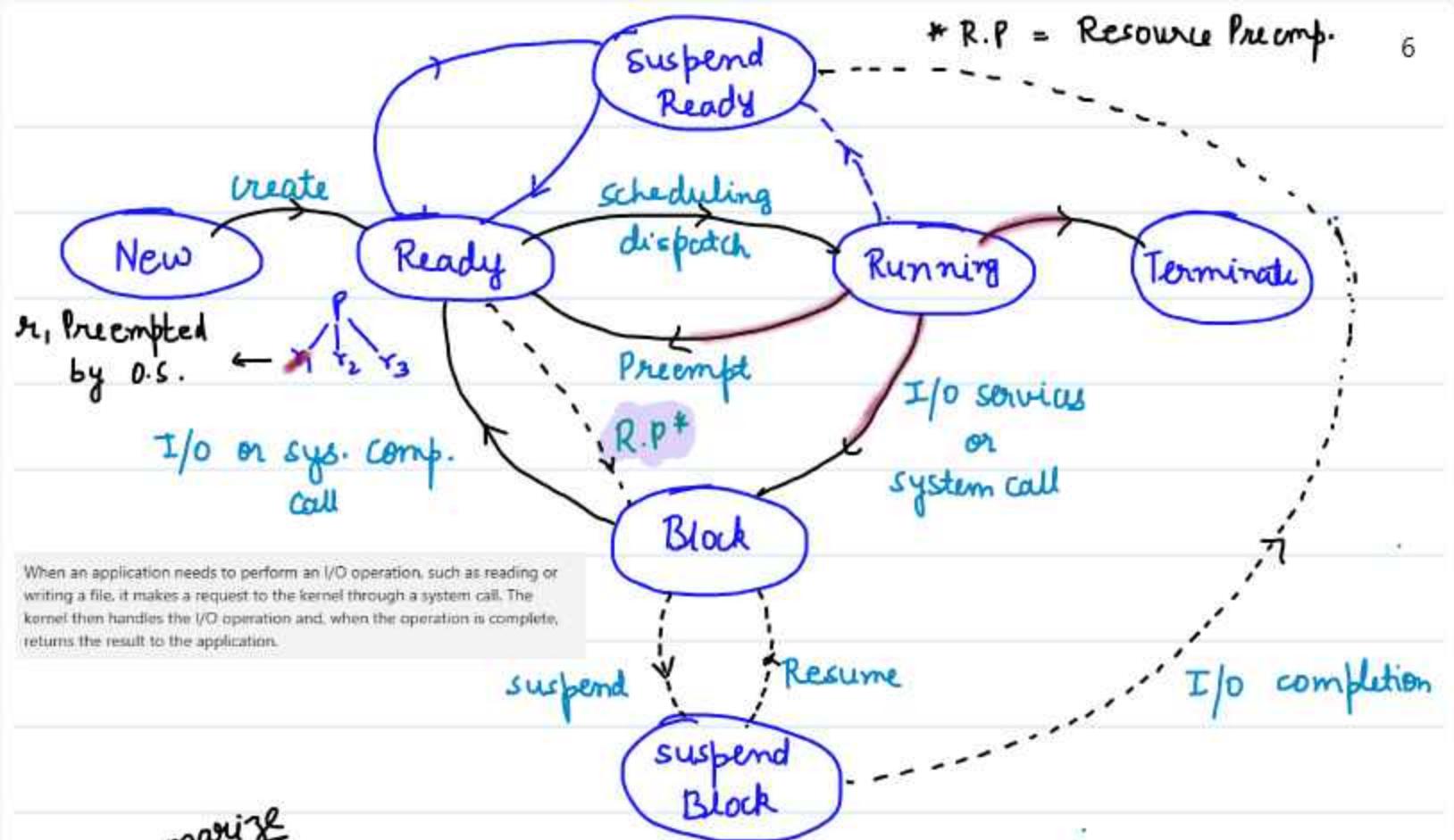


For example, if a process is waiting for data from a file, it will be in the "suspended block" state until the data becomes available. Once the data is ready, the process transitions to "suspended ready" and can be scheduled to run by the operating system.



1. **Process in Running State:** Initially, the process is in the running state, executing its instructions.
2. **Request for I/O Services or System Call:** The process may require I/O services, such as reading from a file or sending data over the network. It makes a system call to request these services.
3. **Blocked State:** The process transitions to a blocked state while waiting for the requested I/O operation to complete. While in this state, the process is not actively executing instructions.
4. **I/O Operation Completion:** The operating system handles the requested I/O operation. For example, if the process requested to read data from a file, the operating system would perform the necessary disk I/O operation to read the data into memory.
5. **Process Resumes:** Once the I/O operation is complete, the process transitions back to the ready state, indicating that it is ready to continue executing. The operating system scheduler will then determine when to schedule the process for execution again.

\* R.P = Resource Preemp.



When an application needs to perform an I/O operation, such as reading or writing a file, it makes a request to the kernel through a system call. The kernel then handles the I/O operation and, when the operation is complete, returns the result to the application.

let's Summarize

Q Why the process is moved from blocked to suspended block?

A process is moved to the "suspended blocked" state due to resource management and multitasking needs.

When a process is in the blocked state, it's waiting for an I/O operation to complete. However, sometimes the operating system needs to free up resources (like memory), or it might need to pause the process for other reasons (like if the user switches to a different application). In these cases, the operating system can move the process from the blocked state to the "suspended blocked" state.

In the "suspended blocked" state, the process is not only waiting for an I/O operation to complete, but it's also been swapped out of main memory and placed onto disk storage. This frees up system resources and allows the operating system to work on other active processes.

Once the I/O operation is complete and the system resources are available again, the process can be moved back into main memory and transitioned to the "ready" state, where it's eligible to continue running on the CPU.

So, the transition to the "suspended blocked" state is a way for the operating system to manage system resources efficiently while ensuring that all processes get their turn to execute.

## Q Why send to suspended ready instead of Ready?

The transition from "suspended blocked" to "suspended ready" instead of directly to "ready" is due to the way operating systems manage memory and resources.

When a process is in the "suspended blocked" state, it's not in the main memory but is swapped out to the disk. This is done to free up main memory for other active processes. When the I/O operation that the process was waiting for is complete, the process becomes "ready" in a logical sense, but it's still not in the main memory. So, it's in the "suspended ready" state.

Before the process can actually be executed (i.e., moved to the "ready" state), it needs to be brought back into the main memory from the disk. This operation is known as "swapping in" or "loading". The decision of when to load the process back into main memory is made by the operating system based on various factors like system load, memory usage, priority of processes, etc.

So, the "suspended ready" state is a way for the operating system to acknowledge that a process is ready to run but hasn't yet been loaded back into main memory. Once the process is back in main memory, it moves to the "ready" state and can be scheduled to run on the CPU.

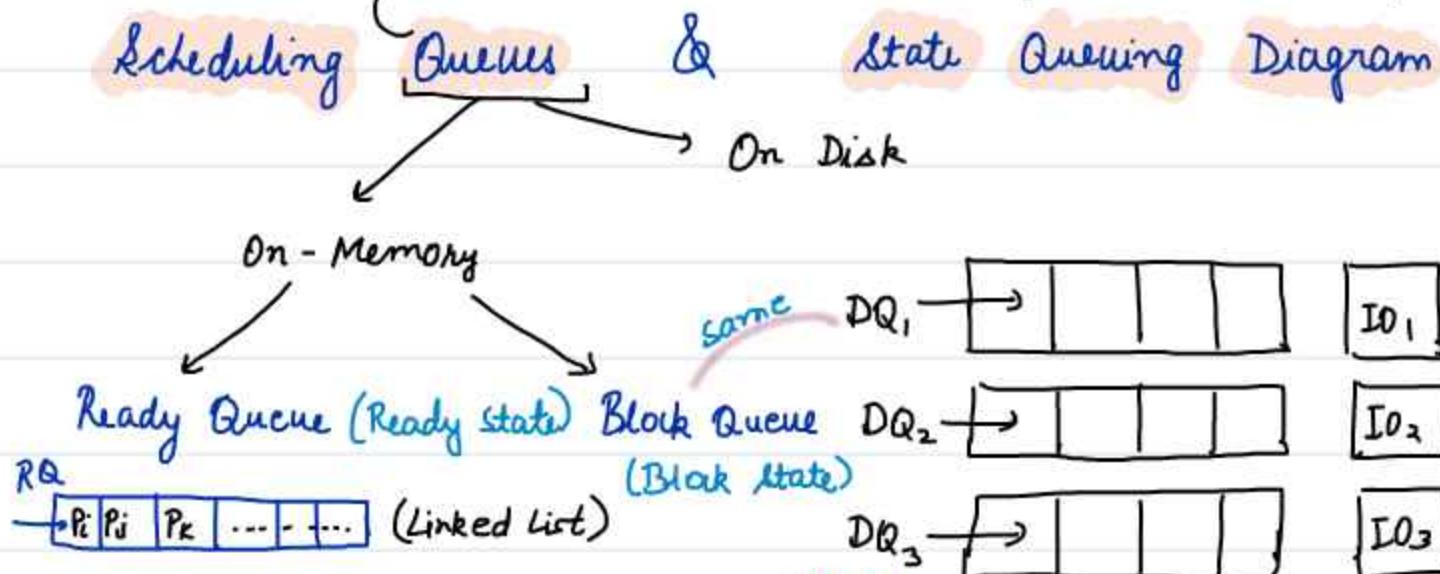
## Q Why resuming the process from suspended block → block instead of sending it to suspended ready & then ready ?

Ans

- **Case 1 (Suspended Block -> Suspended Ready -> Ready):** This case is typically used when the system has enough resources (like memory) to keep the process in main memory, even though it's not currently executing. When the I/O operation is complete, the process can be moved directly to the Ready state because it's already in main memory.
- **Case 2 (Suspended Block -> Block -> Ready):** This case is used when the system needs to free up resources. The process is moved from Suspended Block to Block state when the I/O operation is complete, but the process is still on disk. Once the system resources become available, the process can be swapped back into main memory and moved to the Ready state.

# Lecture - 10

a Data structure which follows FIFO discipline.



Ready Queue contains the list of PCBs of Ready Process

Ready Queue contains the list of PCBs that get blocked onto that device.

Each I/O device will have its own DQ.

Nodes in Llist are PCBs.

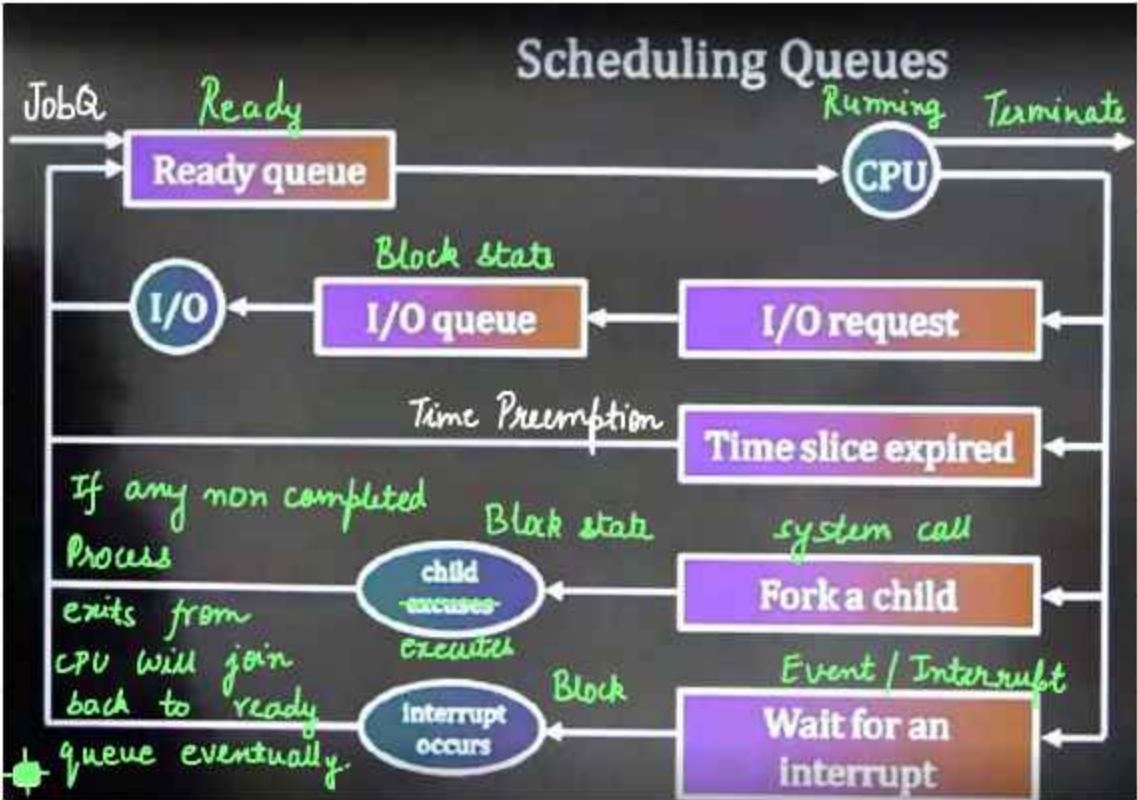
## On-disk Queues

Job Queue  
(New state)

contains Programs ready to be loaded in memory.

Suspend Queue (Repr. both Suspend block & ready)

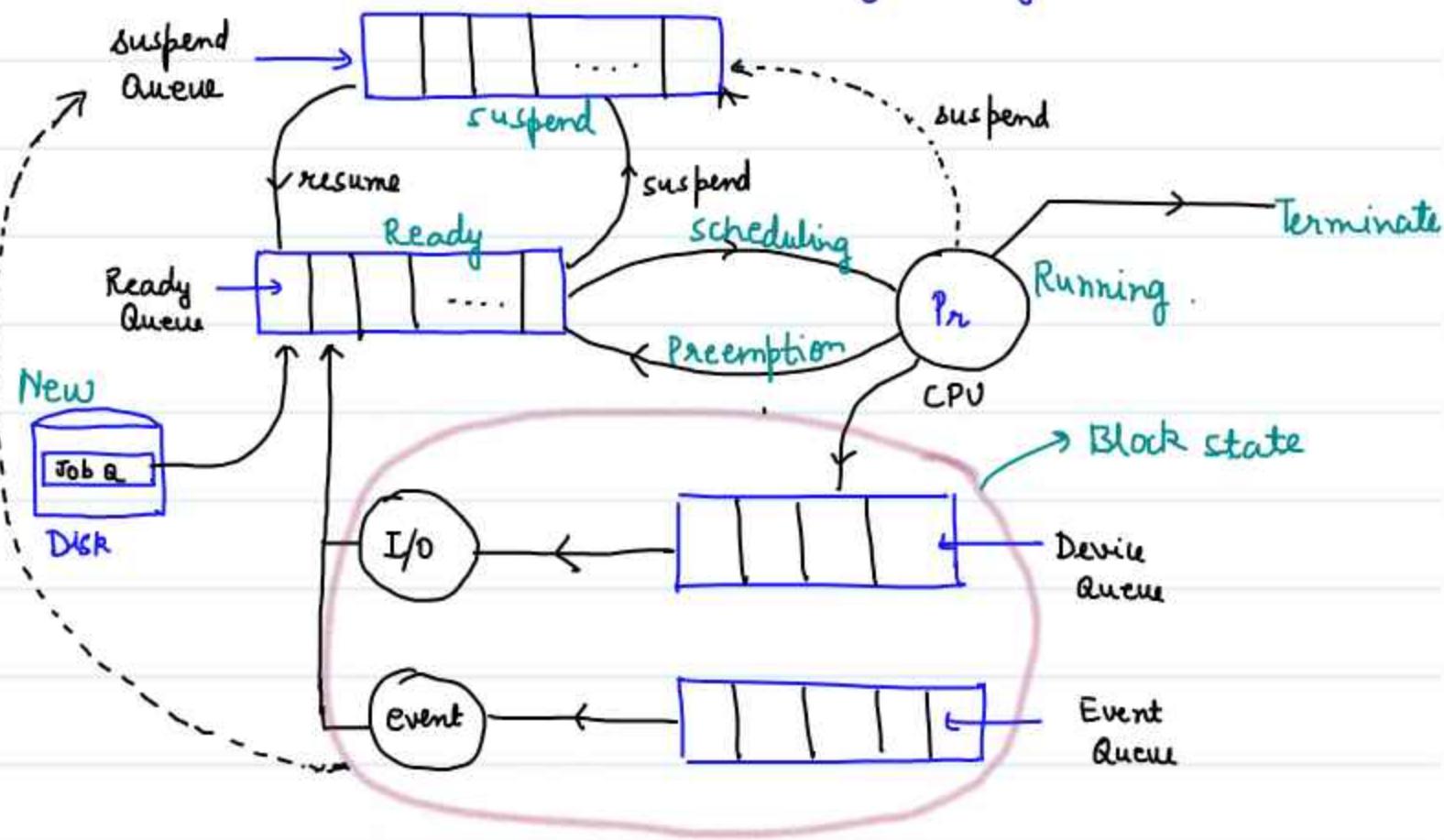
List of Processes that gets suspended from Memory onto disk



fork() → creates a child process

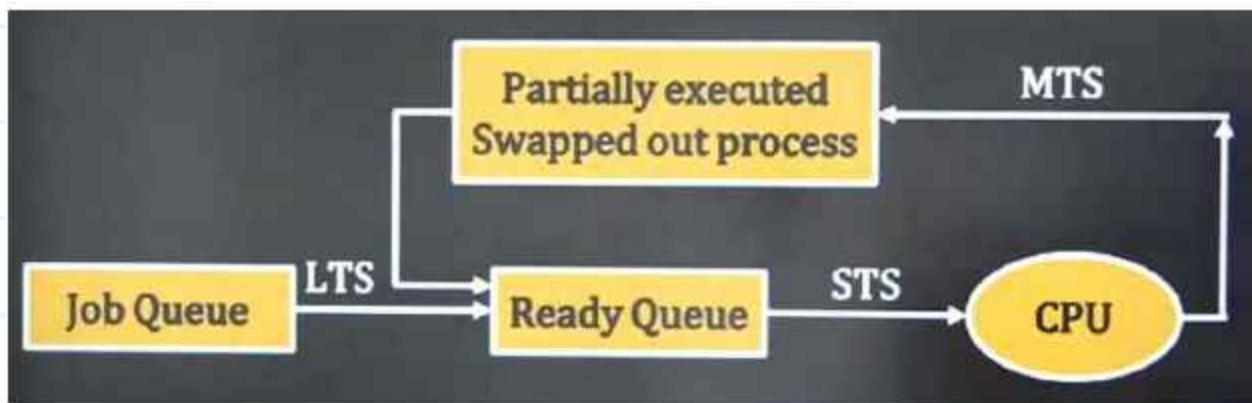
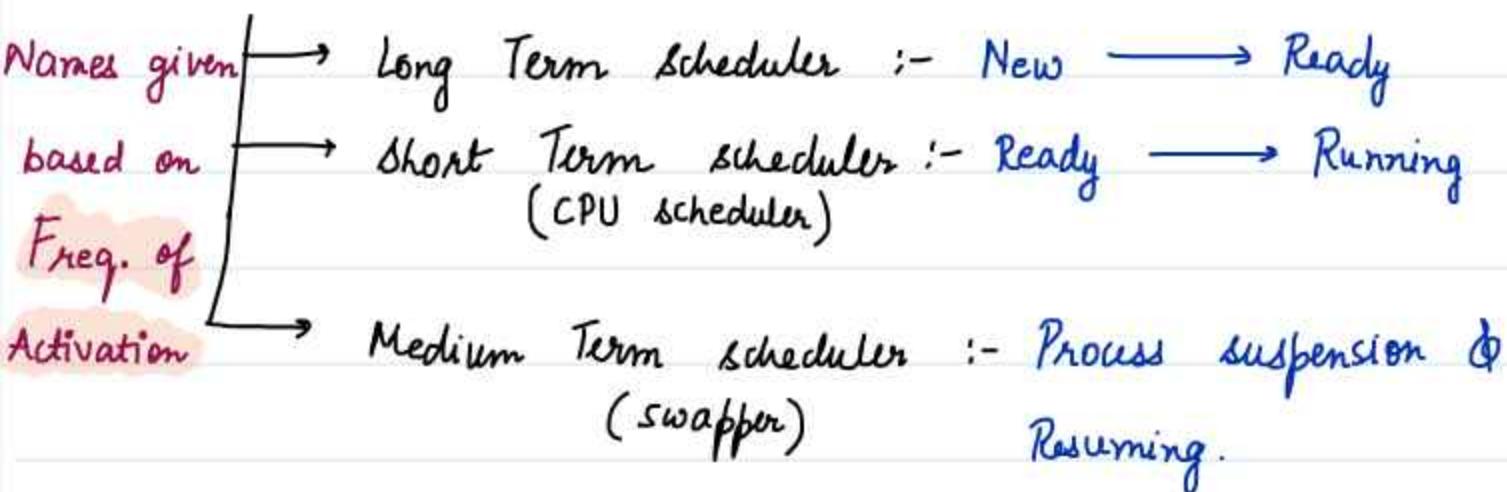


Lets create a State Queuing Diagram :-



# # Schedulers & Dispatchers

- Scheduling means making a decision. In which Q's we have to make a decision :-
- will focus on these three {
- Job Queue
  - Ready Queue
  - Suspend Queue
- Scheduler is the component of Operating system that makes the decision.

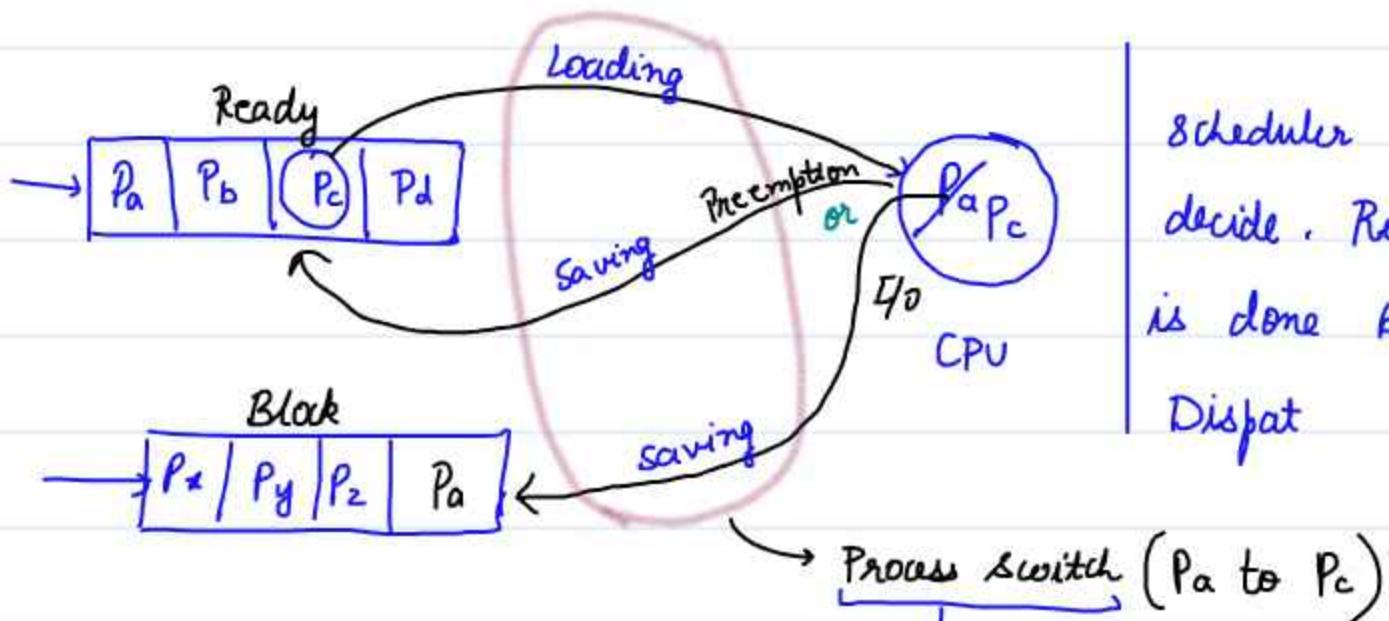


- Long Term Scheduler controls Degree of Multiprogramming
  - ↳ Loads New program in the memory.

- Dispatcher :- Responsible for carrying out the activity of context switching.

works with  
Short Term  
Scheduler

Activity of Loading & saving the process during a process switch on CPU.



Scheduler will decide. Rest is done by Dispatcher

saving the PCB of the CPU leaving process & loading the PCB of the next process.

the context of  $P_c$  is active.

Context switching

Time taken by Dispatcher to do this is Context Switch Time or

Dispatch Latency or CPU Scheduling Overhead

- Dispatcher is not involved in suspension of processes  
there the MTS will work.

In the next lecture :- CPU Scheduling

## Section - 3

## CPU Scheduling

Q Lets say  $t_1$  = User to Kernel Mode Switch time

$t_2$  = Process switch time

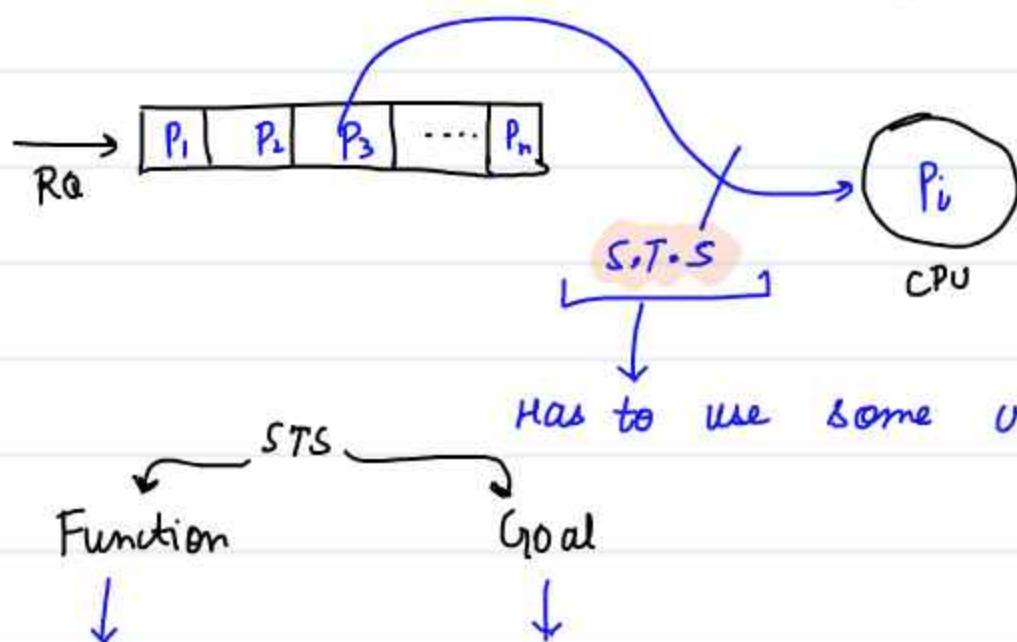
Which one is bigger?

Ans :-  $t_2 > t_1$

Process switching involves mode shifting time

### # CPU Scheduling #

→ Design & Implementation of  
Short Term Scheduler.



Select a process from R.Q. to run on CPU      Maximize CPU utilization, Throughput, Efficiency  
 Minimize waiting Time, TAT, RT → Response Time

# Process Times

1. Arrival Time :- The time at which process enters the Ready Queue for the first time.  
(or submission Time)



2. Waiting Time :- The time spent by the process waiting in the ready queue



3. Burst Time :-



The time spent by the process running on C.P.U.

4. IO Burst Time :-



Time spent by the process waiting for IO operation in blocked state.

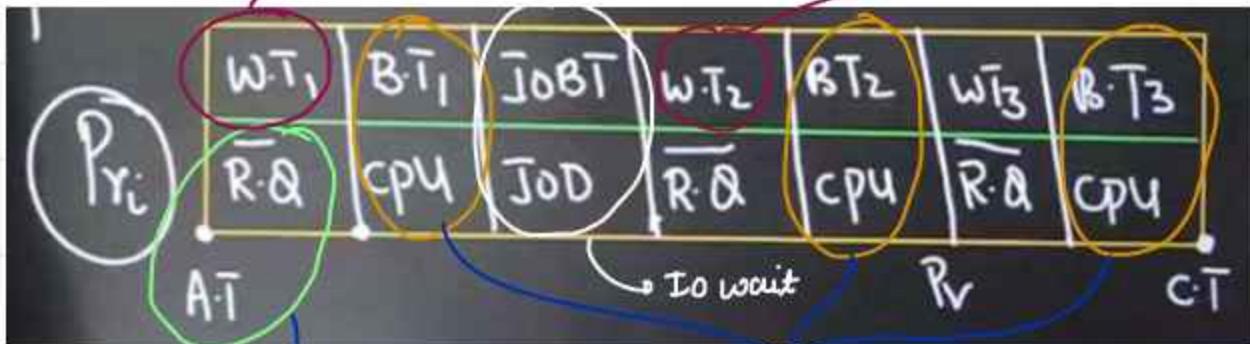
## NOTE

In most modern operating systems, when a process initiates an I/O (Input/Output) operation, it typically doesn't perform the actual I/O operation itself. Instead, it usually initiates the operation and then enters a blocked state, waiting for the operating system to handle the I/O operation on its behalf.

5. Completion Time :-



waiting for CPU for first time second time & so on

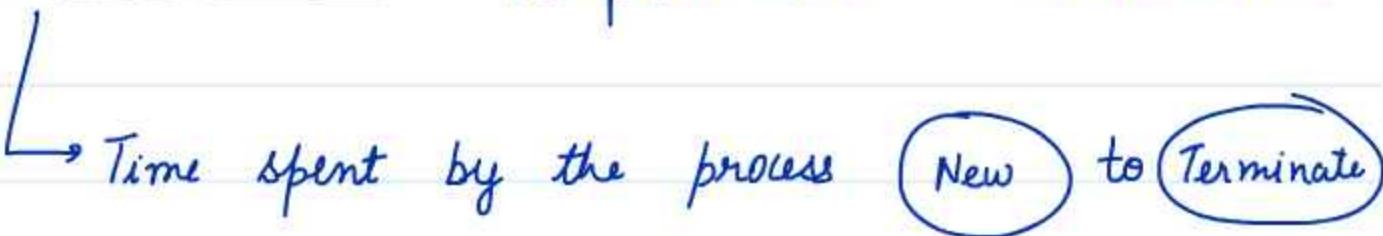


Process arrives in RQ

Process Running on CPU

State Timing Diagram

6. Turn Around Time :- Completion Time - Arrival Time

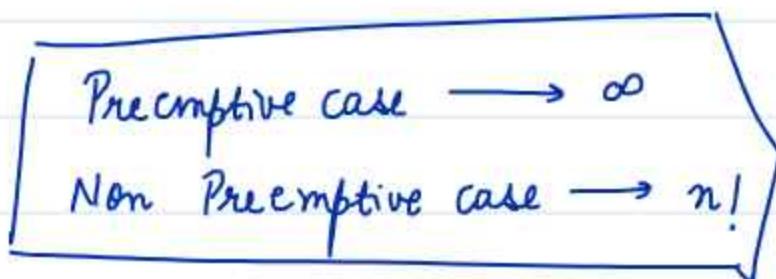


$$\text{Waiting Time} = T.A.T. - (B.T + I.O.B.T)$$

7. Schedule Length :- Total Time taken to complete all  $n$  process as per schedule.

Schedule ?  $\langle P_1, P_2, P_3 \rangle$  ] - A schedule

How many schedules possible with  $n$  process =  $n!$



always?  
↓  
No  
↓  
Then?

$$L = \langle P_1, P_2, P_3 \rangle ?$$

sum of TAT? X

Sum of BT?

- If you add TAT of all process it becomes more than  $L$
- If you add B.T. of all process it'll include IOBT too.

$L = \text{Completion Time of Last Process} - \text{Arrival Time of First Process}$

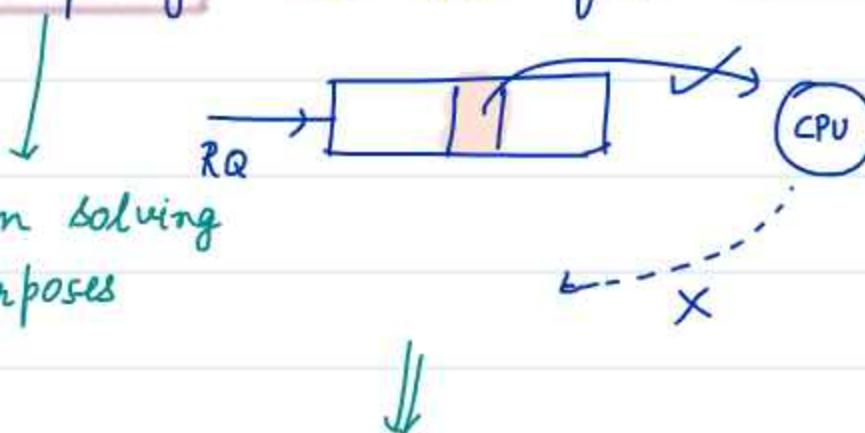
$$= \max(C_i) - \min(A_i)$$

Throughput = No of Processes completed per unit time

$$\eta = \frac{n}{L}$$

Context Switching Time or Scheduling Overhead = ⑤

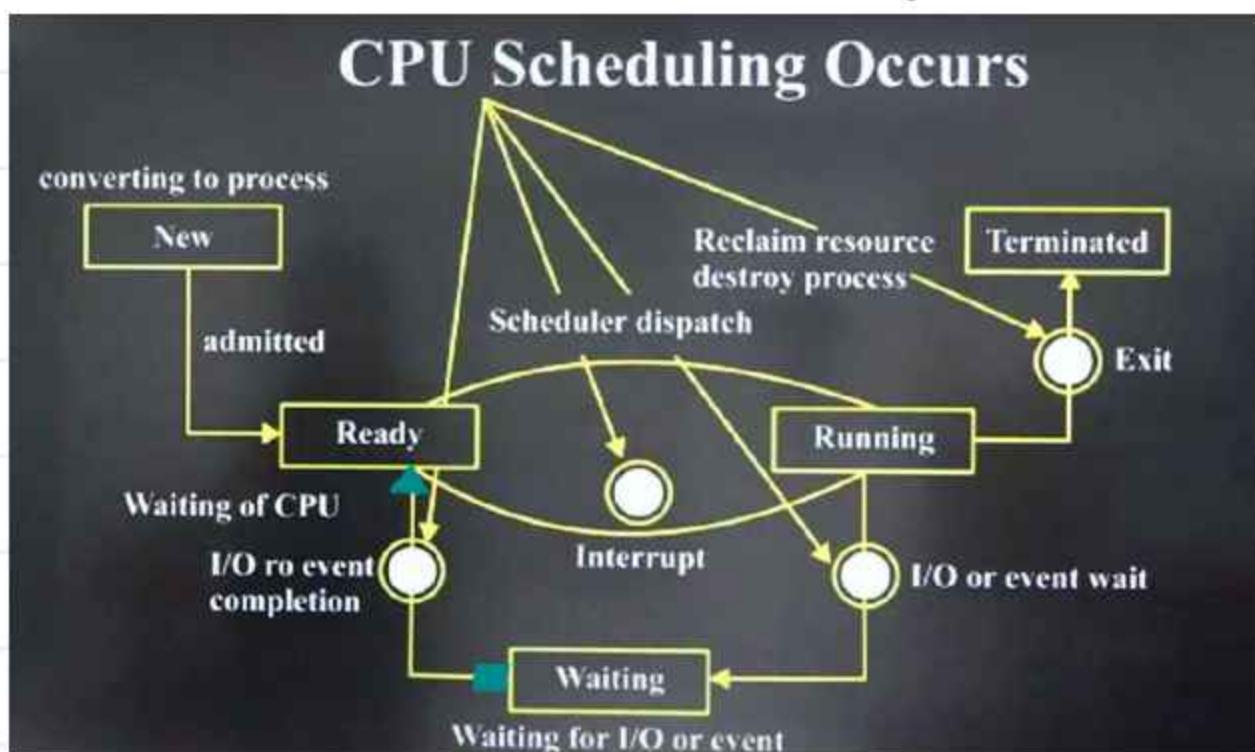
For simplicity we are just including the loading time



Problem solving  
purposes

The time taken by Dispatcher to load the PCB from Ready Queue onto CPU.

Q Where we need scheduling ?



CPU scheduling

Preemptive

Non Preemptive

CPU-bound process: performs lots of computations in long bursts, very little I/O

I/O-bound process: performs lots of I/O followed by short burst of computation

Ideally a system should be the mix of both.

To maximize CPU & I/O utilization

Now we will learn some algo :-

### ① First come First serve (FCFS)

- Selection Criteria :- Arrival Time
- Mode of Operation :- Non Preemptive
- Conflict Resolution :- Lower Process Id

Assumptions :-

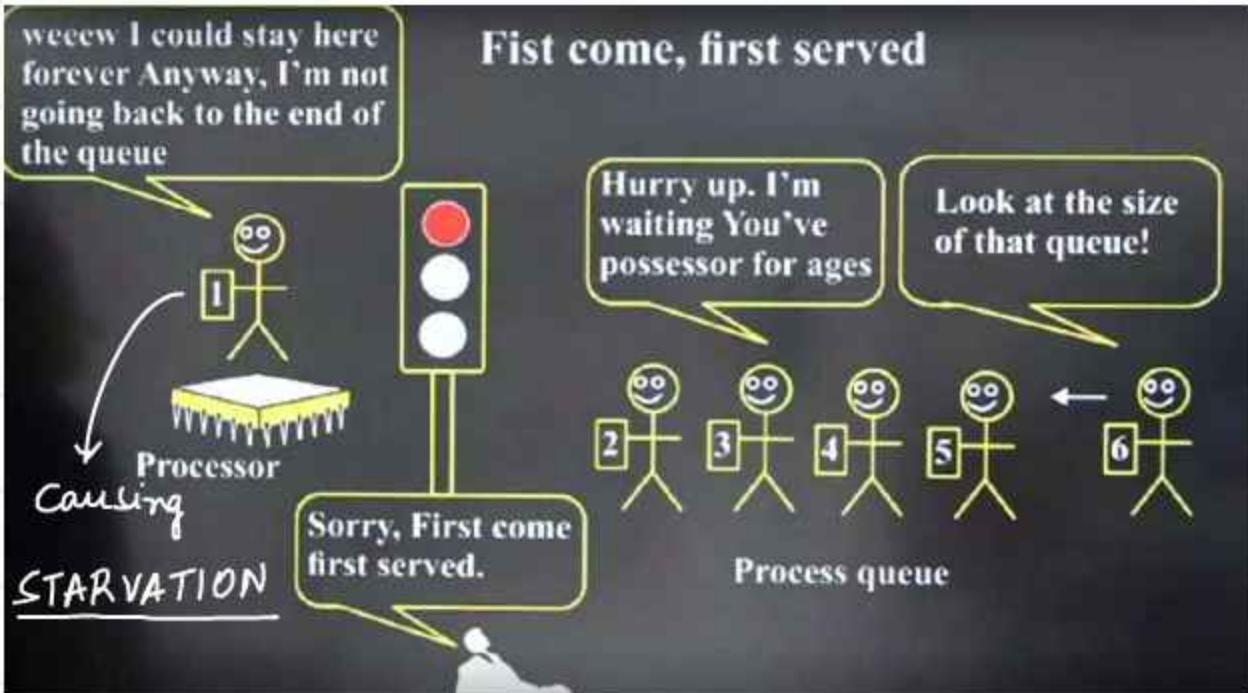
Time is in clock Ticks

No IOBTs

Scheduling overhead = 0

} - Later  
we will take them

What's the problem with the FCFS Scheduling ?

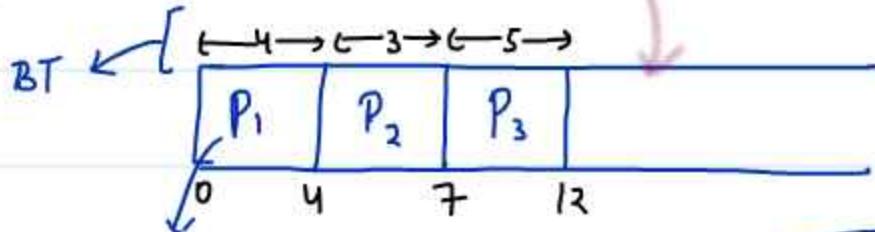


Q Let's Practice :- FCFS Algorithm

P.NO	A.I	B.T	C.T	TAT	WT
1 -	0 - 4 -	-	-	?	
2 -	0 - 3 -	-	-		
3 -	0 - 5 -	-	-		

Shows in what order are we going to schedule

All were present in RQ at t=0



Lowest PID

Gantt chart

$$L = 12 - 0 = 12$$

$$TAT = CT - AT$$

$$WT = TAT - BT$$

How much time

process waited in R.Q.

P.NO	A.I	B.T	C.T	TAT	WT
1 - <sup>min</sup>	0 - 4 -	4	4	4	0
2 -	0 - 3 -	7	7	7	4
3 -	0 - 5 -	12	12	12	7

Q-2

## FCFS Algo

8

P. NO	A.T	B.T	C.T	TAT	W.T
1	0	2	2	2	0
2	0	1	3	3	3
3	2	3	6	4	2
4	3	2	8	5	3
5	5	4	12	7	3

$L = 12$

CT - AT

TAT - BT

Time	Ready Queue
0	P <sub>1</sub> P <sub>2</sub>
2	P <sub>2</sub> P <sub>3</sub>
3	P <sub>3</sub> P <sub>4</sub>

5	P <sub>3</sub> P <sub>4</sub> P <sub>5</sub>
6	P <sub>4</sub> P <sub>5</sub>

↓ Tie Breakers

P <sub>1</sub>	P <sub>2</sub>	P <sub>3</sub>	P <sub>4</sub>	P <sub>5</sub>
0	2	3	6	12

Lower Process Id

Q-3

## FCFS Algo

P. NO	A.T	B.T	C.T	TAT	W.T
1	3	5	8	8	5
2	10	2	12	12	10
3	15	4	16	16	12
4	18	5	23	23	15

$L = 21$

Fill Yourself  
Homework

Time

0

Ready Q

X

3

P<sub>1</sub>

8

X

10

P<sub>2</sub>

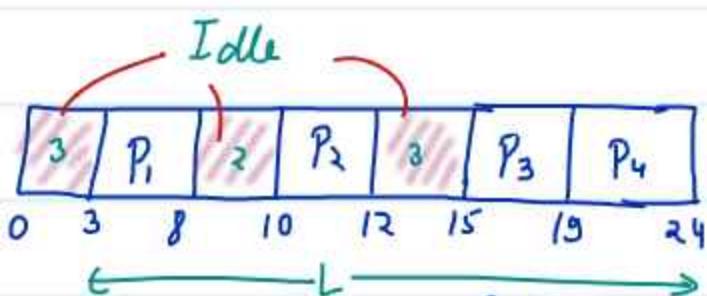
12

X

15

P<sub>3</sub>

18

P<sub>4</sub>

Gantt Chart } Always starts from 0

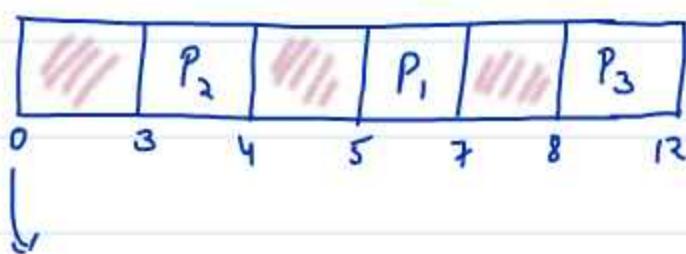
Now I want to calculate % age CPU Idle.

$\rightarrow$  calculated over Schedule Length  $L = \frac{5}{21}$

9

## Q FCFS Algorithm

P.No	A.T	B.T
1	5	2
2	3	1
3	8	4



Time	Ready Q
0	X
3	P <sub>2</sub>
4	X
5	P <sub>1</sub>
6	X
8	✓
12	X

Always starts from 0.

H.W Find % CPU Idleness (FCFS Algo)

P.No	A.T	B.T
1	10	3
2	11	4
3	20	5
4	2	3
5	12	3
6	8	1

$$\% \text{ CPU Idleness} = \frac{4}{23}$$

# CPU Scheduling Part - 2

We'll learn about algorithms if they include  $\delta$  and I/OBT.

- In problems we will be given with process lifecycle  $\langle BT, IGBT, BT \rangle$  &  $\delta = L$  (say)

Ex:-

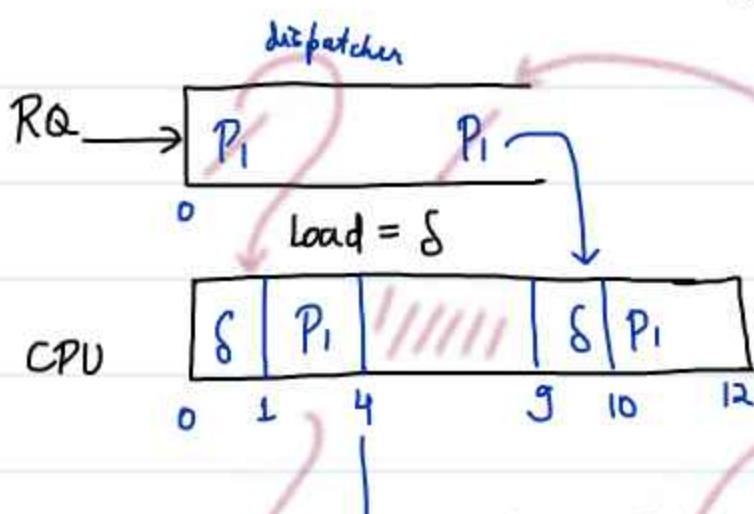
Pno A.T.

Lifecycle

L

0

$\langle 3, 5, 2 \rangle$



Save Time

= negligible

$$\delta = L$$

why not formula?

From chart

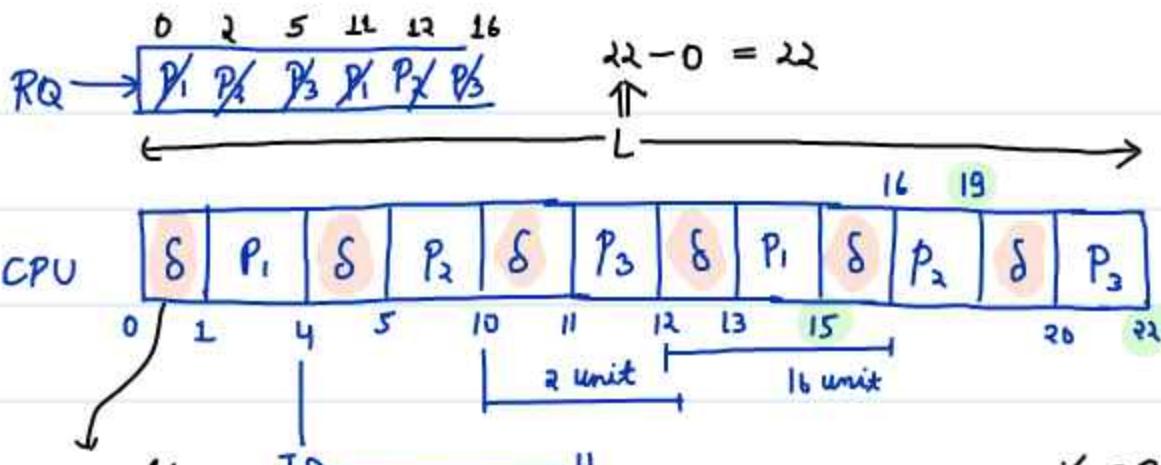
$$CT - AT$$

Q

P.No	A.T	$\langle R.T ; IGBT ; BT \rangle$
1-	0	$\langle 3 ; 7 ; 2 \rangle$
2-	2	$\langle 5 ; 2 ; 3 \rangle$
3-	5	$\langle 1 ; 4 ; 2 \rangle$

$$\begin{aligned} WT &= 0+1 \\ &= 1 \\ WT &= 2+3 \\ &= 5 \\ WT &= 5+3 \end{aligned}$$

$$\begin{aligned} TAT &= H.W \\ CT &= H.W \end{aligned}$$



This is the transit time  
not waiting system has multiple I/O devices time.

I/O operation can be concurrent

W.T. by formula :-

$$\begin{aligned} WT &= TAT - (BT + IOBT) \\ &= 15 - (12) = 3 \times \end{aligned}$$

$WT = TAT - (BT + IOBT + n \cdot \delta)$  we ignored  $\delta$

$\downarrow$  no of times a process

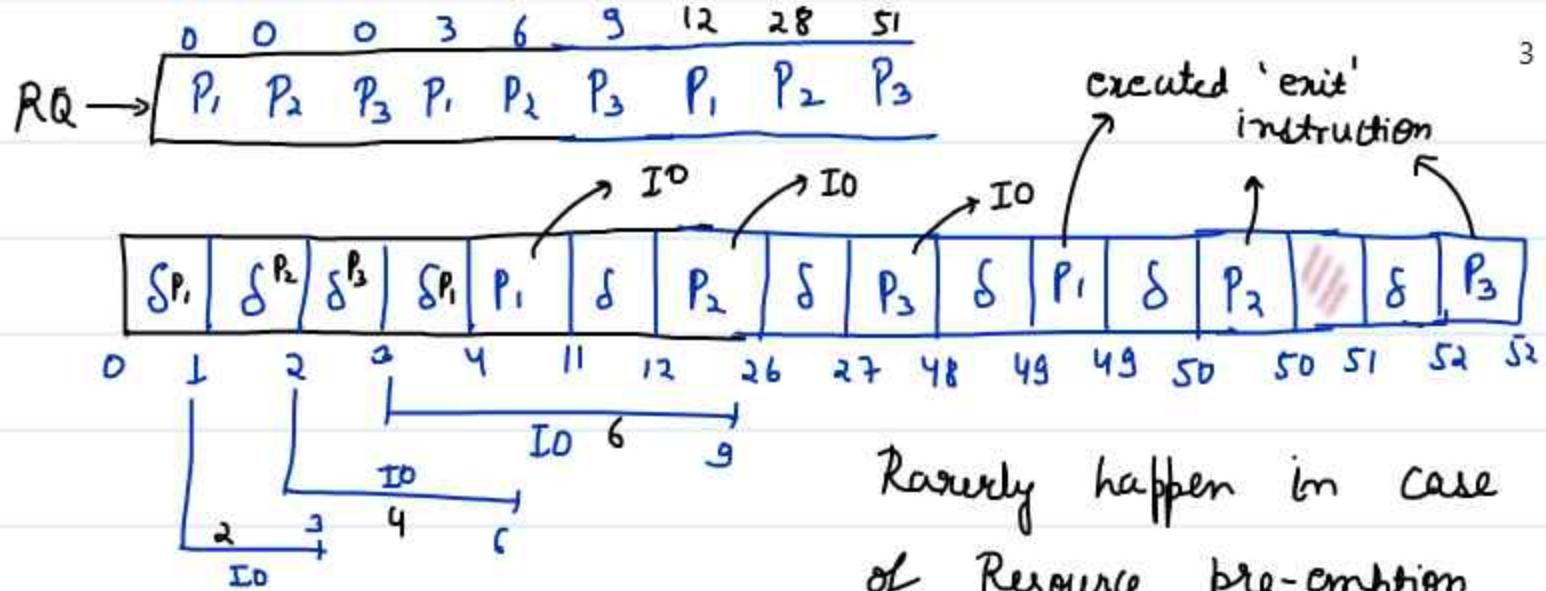
$WT = 15 - (12 + 2) = 1$  get scheduled on CPU

Q  $S.T$   $P.No$   $A.T$   $\left\langle \frac{20Y}{IOBT}, \frac{70X}{BT}, \frac{10Z}{IOBT} \right\rangle$

$10 - 1 - 0 -$	$\left\langle 2 ; 7 ; 1 \right\rangle$
$20 - 2 - 0 -$	$\left\langle 4 ; 14 ; 2 \right\rangle$
$30 - 3 - 0 -$	$\left\langle 6 ; 21 ; 3 \right\rangle$

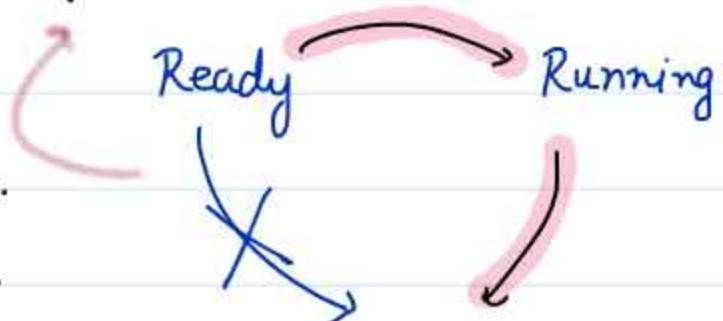
$\delta = L$

"MUST TRY ON YOUR OWN FIRST"



Tips :-

1. Always create Time & RQ.
2. Start Gantt chart from 0.
3. Always Remb. Trans. Diagram
4. Don't include  $\delta$  in WOT (while calculating from chart)

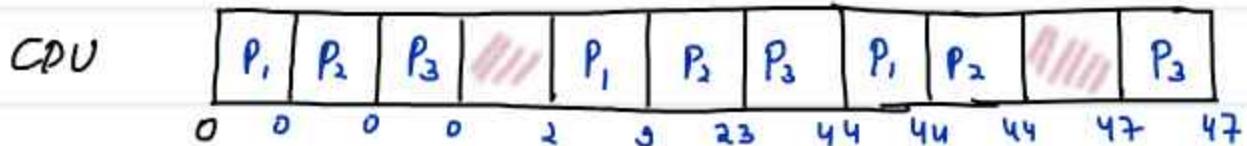
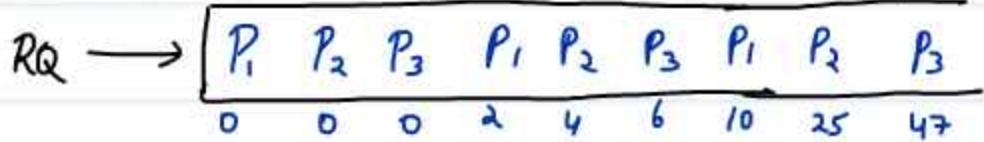


(b) Do the same question with  $\delta = 0$ .

ST	P <sub>No</sub>	A <sub>i</sub>	< 10BT ; BT ; 10BT >		
10	1	0	2	7	1
20	2	0	4	14	2
30	3	0	6	21	3

$$L = 47$$

$$\% \text{CPU Idleness} = \frac{5}{47}$$



(c) For Part (a) calculate % CPU overhead activity

during L. Ans =  $\frac{9}{52}$

$$\% \text{ CPU efficiency} = 100 - \left( \frac{\% \text{ CPU overhead}}{\text{overhead activity}} + \frac{\% \text{ CPU Idleness}}{\text{Idleness}} \right)$$

## # Questions for Homework #

Q-1

$$\delta = 2$$

P.No	A.T	$\langle BT, IOB, BT \rangle$
1-0	-	$\langle 3, 4, 2 \rangle$
2-5	-	$\langle 2, 0, 3 \rangle$
3-20	-	$\langle 1, 15, 6 \rangle$

Q-3

Try Q-1 &

Q-2 assuming  
system has only  
one IO device.

Q-2

$$\delta = 1$$

P.No	A.T	$\langle IOBT, BT, IOBT \rangle$
1-3	-	$\langle 5, 2, 3 \rangle$
2-8	-	$\langle 2, 10, 4 \rangle$
3-12	-	$\langle 6, 2, 1 \rangle$

If one process  
is performing  
IO other has  
to wait.

—————END—————

## CPU Scheduling - 3

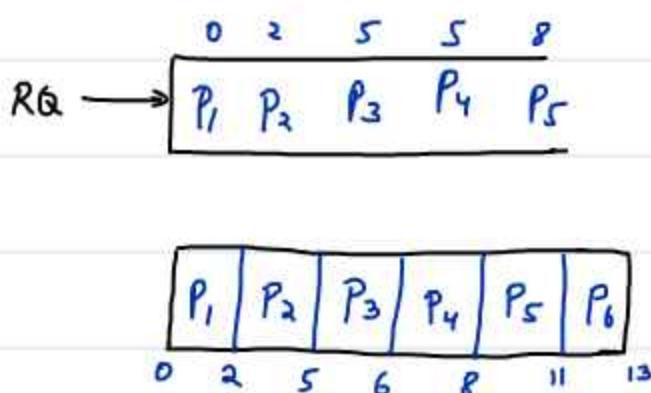
SJF (Shortest Job First) :- Burst Time Based

- Selection criteria :- Shortest Burst Time
- Mode of op. :- Non-Pre
- Conflict Resolution :- Lower P<sub>ID</sub> → ( $P_1 > P_2$  : Preference)

Among the process present in RQ select the process with least Burst Time & schedule it on CPU.

Q

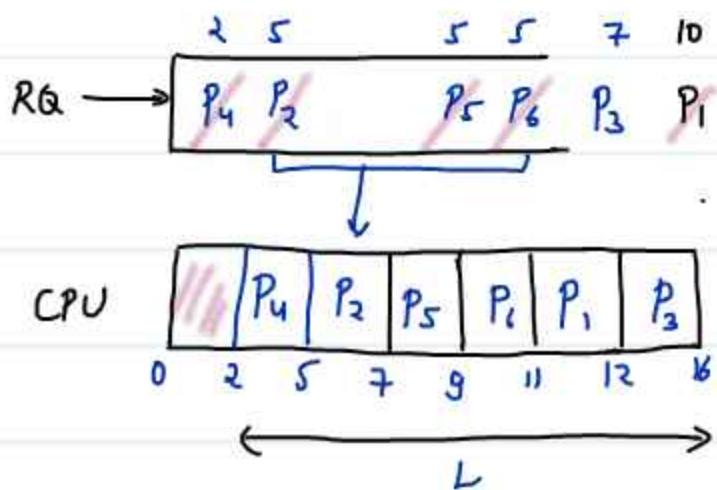
P.No	AT	BT
1	0	2
2	2	3
3	3	1
4	5	2
5	8	3
6	10	2



Burst Time is only compared b/w the processes in Ready Queue.

Q

P.No	A.T	B.T
1	10	1
2	4	2
3	6	4
4	2	3
5	5	2
6	3	2



%CPU Idleness = 0

# # Shortest Remaining Time First / Pre-emptive SJF

- Selection criteria :- Burst Time
- Mode of Operation :- Pre-emptive
- Tie Breaking :- Pre-emption of Running process is based on the availability of strictly shorter process.

→ Out of the processes present in RQ select the one having least B.T. & we will continue to run CPU that process until a new process with shorter B.T. arrives.

Ex:-	P.No.	A.T.	B.T.
	1	0	5 3
	2	2	2

SJF (Non Pre)	P <sub>1</sub>	P <sub>2</sub>
0 5 7		

SRTF	Pn.
0 2 4 7	P <sub>1</sub> P <sub>2</sub> P <sub>1</sub>

If P<sub>2</sub> also had BT = 3 then Remaining BT > New Pro. BT

If No Preemption

At time 2 there is an other process available in RQ which has BT < Remaining BT of currently running process

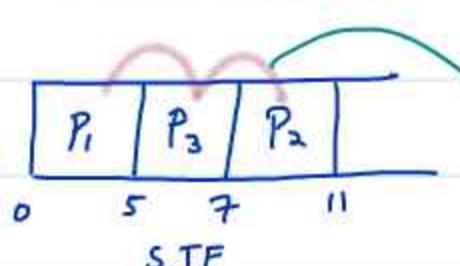
P.NO	A.T	B.T
1	0	5
2	1	4
3	2	2

Which algo has more context switches?

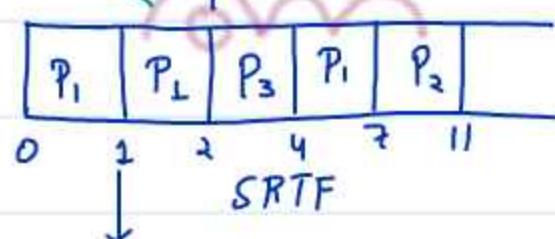
SJF

SRTF

$$\begin{aligned} P_1 &= 3 \\ P_2 &= 2 \end{aligned} \quad ] \text{switch}$$



3 switches  
2 switches



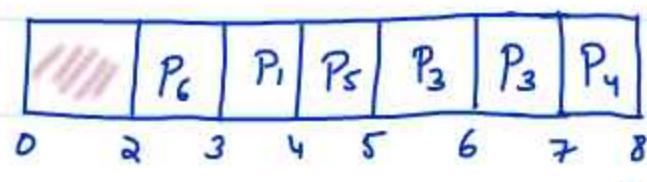
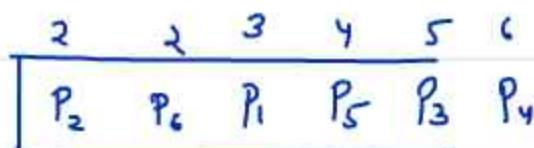
Must Remember

Check everytime whenever a new process comes

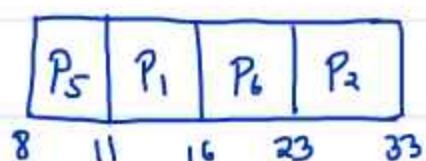
$$\begin{aligned} \text{Remaining } P_1 &= 4 \\ P_2 &= 4 \end{aligned} \quad ] \text{No switch}$$

Q

P.NO	A.T	B.T	SRTF
1	3	6	5 ✓
2	2	10	✓
3	5	2	✗
4	6	1	3
5	4	4	3
6	2	8	7 ✓



No new process is coming



No Preemption

SRTF becomes SJF

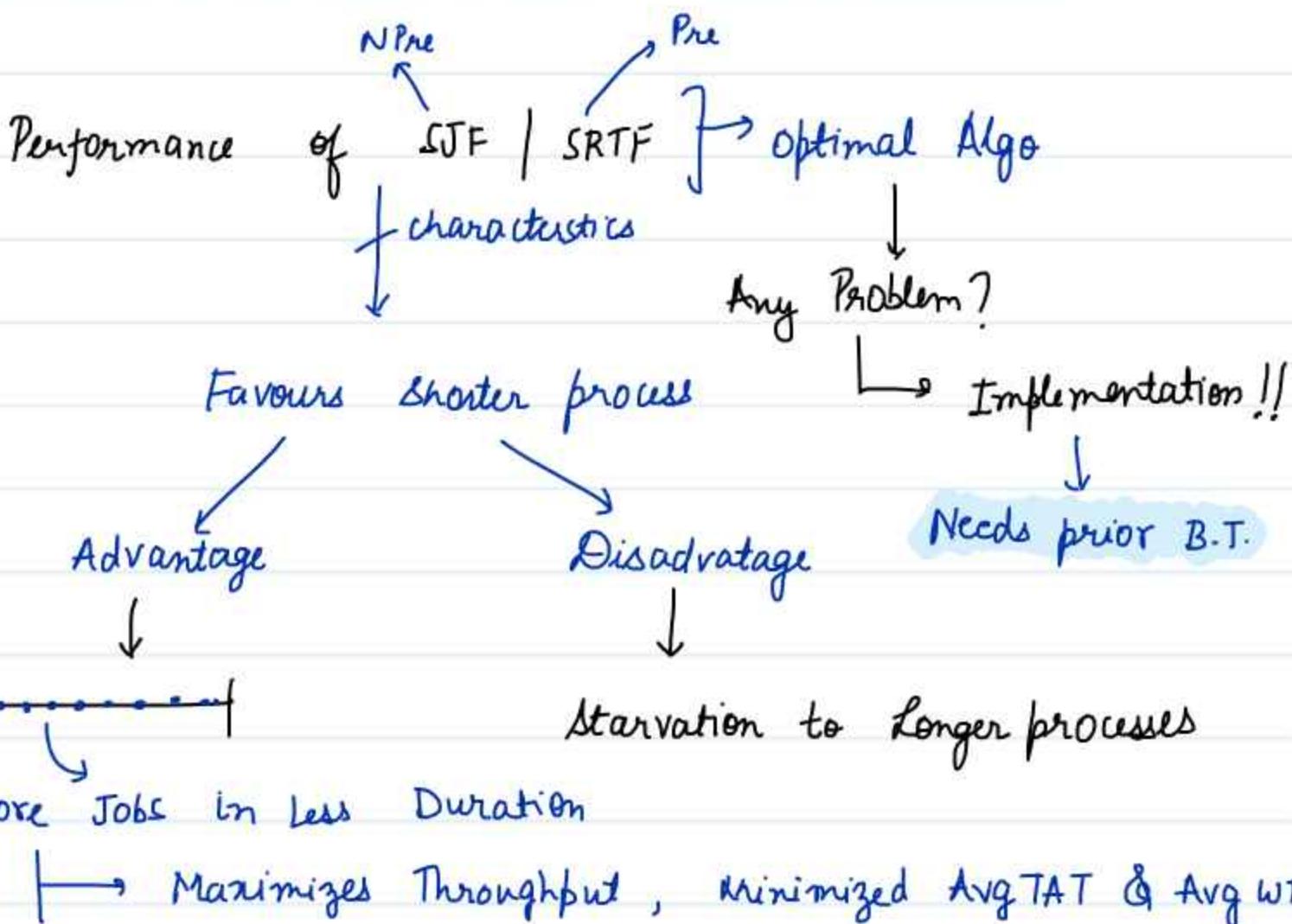
Consider the following processes, with the arrival time and the length of the CPU burst given in milliseconds. The scheduling algorithm used is preemptive Shortest Remaining-Time First (SRTF).

Process	Arrival Time	Burst Time
P1	0	10
P2	3	6
P3	7	1
P4	8	3

The average turnaround time of these processes is 8.25 milliseconds.

Q

Three Processes arrive at time zero with CPU bursts of 16, 20 and 10 milliseconds. If the scheduler has prior knowledge about the length of the CPU bursts, the minimum achievable average waiting time for these three processes in a Non-Preemptive Scheduler (rounded to nearest integer) is \_\_\_\_\_ milliseconds.



Since Burst Time of the Processes are not known a priori

∴ They are not practically implementable.

(SJF/SRTF)

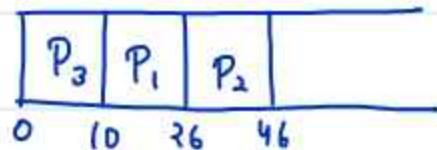
↳ Then why did we study them.

Theoretically SJF is optimal algorithm.



- Used as a benchmark to measure performance of other algorithms.
- SJF can be implemented with Predictive Burst Times

	AT	BT
P <sub>1</sub>	0	16
P <sub>2</sub>	0	20
P <sub>3</sub>	0	10



$$WT = \frac{10 + 26 + 0}{3} = \frac{36}{3} = 12$$

Q Must try first on your own.

Consider the following four processes with arrival times (in milliseconds) and their length of CPU bursts (in milliseconds) as shown below:

Process	P <sub>1</sub>	P <sub>2</sub>	P <sub>3</sub>	P <sub>4</sub>
Arrival time	0	1	3	4
CPU burst time	3	1	3	Z

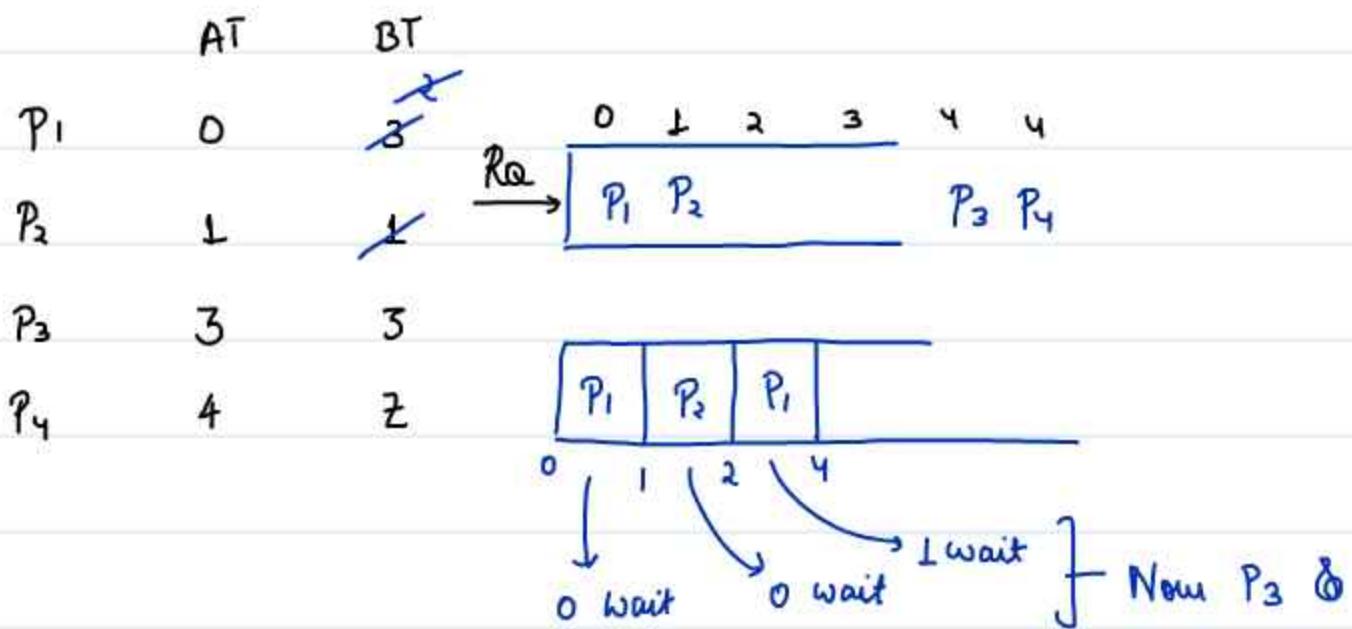
→ SRTF

Algorithm

These processes are run on a single processor using preemptive Shortest Remaining Time First (SRTF) Scheduling Algorithm. If the average waiting time of the processes is 1 millisecond, then the value of Z is

$$\text{Avg wt} = 1$$

$$\text{Total wt} = 4$$



If P<sub>3</sub> gets scheduled first & till 4, P<sub>3</sub> has already

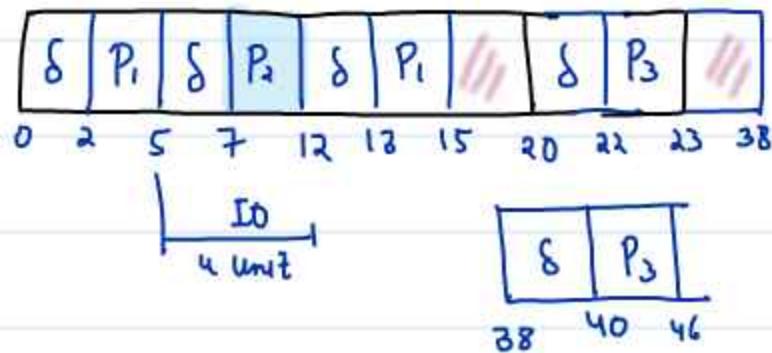
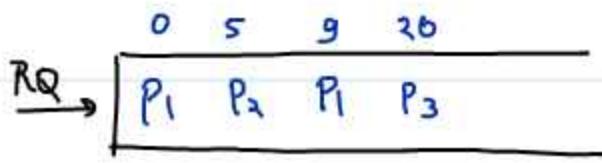
(Z > 3) then P<sub>4</sub> has to wait for 3 unit  $\geq 2$  [ P<sub>4</sub> has to wait for 2 ]

If P<sub>4</sub> gets scheduled first then wt for P<sub>4</sub> = 0 & P<sub>3</sub> = Z ]  $0 + 2 = 2$  it means Z = 2

## Let's Discuss HW Q

$P_{\text{Job}} \quad A\bar{T}$      $\langle BT : \text{Job}, BT \rangle$   
 1 - 0 -  $\langle 3 ; 4 ; 2 \rangle$   
 2 - 5 -  $\langle 2 ; 0 ; 3 \rangle$     *(NO ID)*  
 3 - 20 -  $\langle 1 ; 15 ; 6 \rangle$

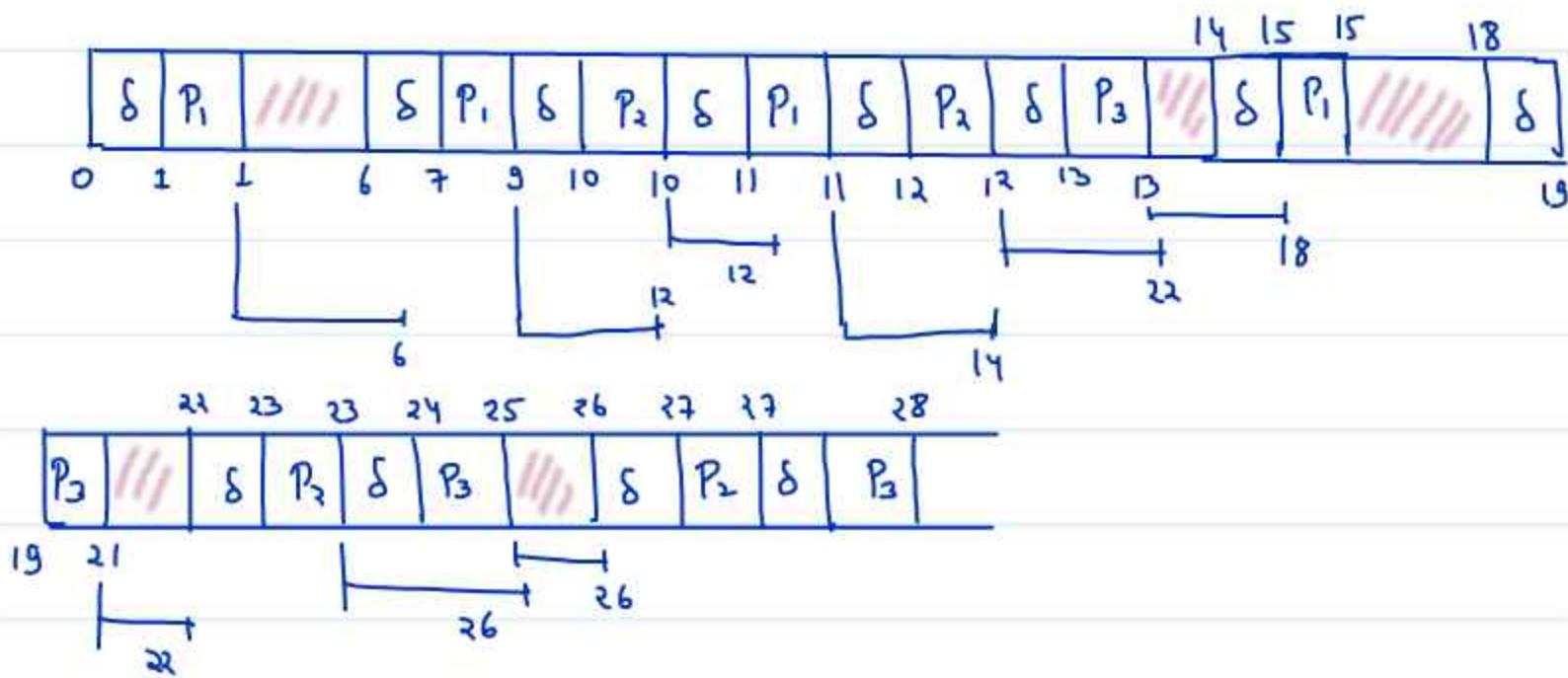
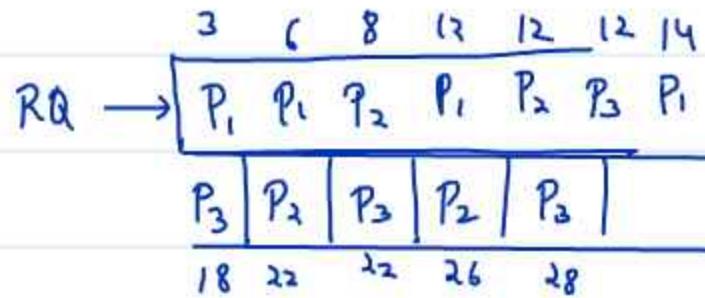
$$\delta = 2 \quad \boxed{\text{FCFS}}$$



Q

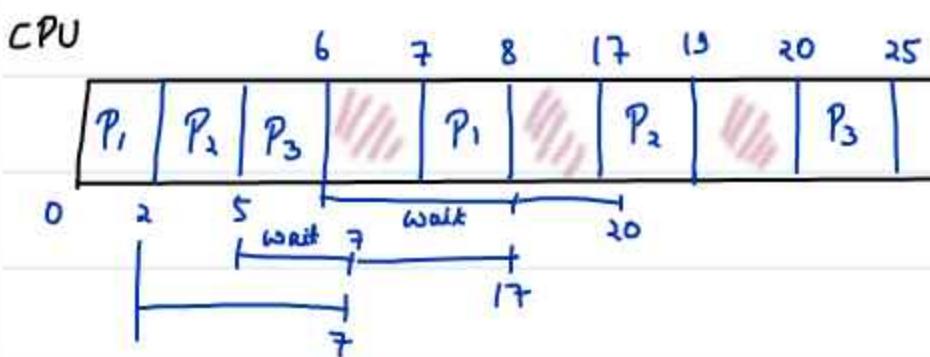
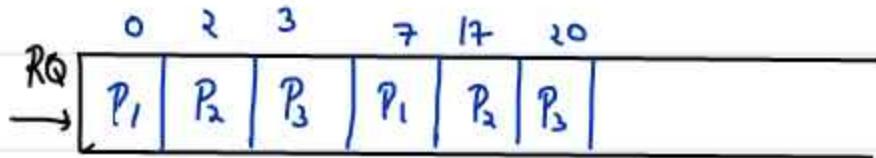
$$\delta = 1$$

$P_{\text{Job}} \quad A\bar{T}$      $\langle \text{Job}BT; BT; \text{Job}BT \rangle$   
 1 - 3 -  $\langle 5 ; 2 ; 3 \rangle$   
 2 - 8 -  $\langle 2 ; 10 ; 4 \rangle$   
 3 - 12 -  $\langle 6 ; 2 ; 1 \rangle$



(c) Non Concurrent I/O  $\delta = 0$  FCFS

<u>P.NO</u>	A.T	B.T	I/O B.	B.T
1 - 0	2	5	3	
2 - 2	3	10	2	
3 - 3	1	3	5	



# CPU Scheduling - 4

Prediction Techniques for CPU Bursts Predictions

Static (Total BT)

Dynamic

W <sub>T1</sub>	B <sub>T1</sub>	J <sub>BT</sub>	WT <sub>2</sub>	B <sub>T2</sub>	?
R.Q. CPU	I/O D	R.Q. CPU	R.Q. CPU	R.Q. CPU	

AT  
While Remaining in PQ we want to predict how much time will be of next burst? Next CPU Burst!

Process Size (Bytes)

$$P_{old} = 101 \text{ kB}$$

$$P_{new} = 100 \text{ kB}$$

Type

Avg BT

$$OS \approx 5s$$

$$\text{Interactive} \approx 10s$$

$$\text{Foreground} \approx 15s$$

$$\text{Background} \approx 30s$$

$P_{new} \rightarrow \text{Type : Bg}$

↓

$$BT \approx 30s$$

Q

How to Predict Dynamically?

Two predict next CPU burst

Exponential Averaging Technique / Aging Algo

$P_i$  : Process

$t_i$  : Completed BT

$T_i$  : Predicted BT

$T_{n+1}$  : Next CPU Burst

$$T_{n+1} = \alpha t_n + (1-\alpha) T_n \quad \alpha \in (0,1)$$

J <sub>1</sub>	①	J <sub>2</sub>	②	J <sub>3</sub>	?	d <sub>3</sub>
W <sub>T1</sub>	B <sub>T1</sub>	J <sub>BT</sub>	WT <sub>2</sub>	B <sub>T2</sub>	J <sub>3</sub>	?
R.Q. CPU	I/O D	R.Q. CPU	R.Q. CPU	R.Q. CPU		

AT  
Pr(t)  
Next CPU Burst!

$$T_{n+1} = \alpha t_n + (1-\alpha) T_n$$

↓

Recurrence Relation

previous predicted burst weightage

weightage given to previous completed burst

$$T_n = \alpha t_{n-1} + (1-\alpha) T_{n-1}$$

$$T_{n+1} = \alpha t_n + (1-\alpha)[\alpha t_{n-1} + (1-\alpha) T_{n-1}]$$

$$T_{n+1} = \alpha t_n + \alpha(1-\alpha)t_{n-1} + (1-\alpha)^2 T_{n-1}$$

↓

find the value & put it

$$T_{n+1} = \alpha t_n + \alpha(1-\alpha)t_{n-1} + \alpha(1-\alpha)^2 t_{n-2} + (1-\alpha)^3 T_{n-2}$$

↓

Keep putting values until  $T_1$

We can't find factorial of any

number until  $F(0) = 1$

GIVEN  $\leftarrow$  [Initial Guess

Given the value of  $\alpha$  &  $T_1$ , one can predict any Next CPU Burst.

Q. Consider a system using Aging Algo, To predict next CPU Burst, Given  $\alpha = \frac{1}{2}$ ,  $T_1 = 10$   
 The process BT's are :- 4, 8, 12, 10  
 Predict the next CPU Burst.

Ans

$$T_{n+1} = \alpha t_n + (1-\alpha) T_n \rightarrow \frac{24+15}{4}$$

$$\frac{40+24+15}{8} = T_5 = \alpha t_4 \xrightarrow{10} + (1-\alpha) T_4 \rightarrow \frac{24+15}{4}$$

$$9.875 \text{ Ans} \quad T_4 = \alpha t_3 \xrightarrow{12} + (1-\alpha) T_3 \xrightarrow{15/2} = \frac{24+15}{4}$$

$$T_3 = \alpha t_2 \xrightarrow{8} + (1-\alpha) T_2 \xrightarrow{7} = 15/2$$

$$T_2 = \alpha t_1 \xrightarrow{4} + (1-\alpha) J_1 \xrightarrow{10} = \frac{1}{2} (t_1 + J_1)$$

The problem of starvation is still continuing in SJF (whether Pre / Non Pre) longer process will starve against shorter process.

## # HRRN (Highest Response Ratio Next)

Selection Criteria :  $\frac{WT + BT}{BT} \rightarrow$  also known as service time  $\frac{(w+s)}{s}$

- Next process to run is the one whose response ratio is highest. Longer process waiting from a long time will have high RR  $\rightarrow$  Problem solved

$WT \uparrow$  then  $RR \uparrow$   
 $BT \downarrow$  then  $RR \uparrow$

• Mode : Non Preemptive

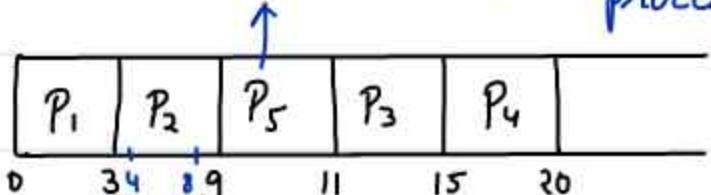
• Tie Breaker : Lower PID

We are favouring short process

Q

P.No	A.T	B.T
1 - 0 - 3		
2 - 2 - 6		
3 - 4 - 4		
4 - 6 - 5		
5 - 8 - 2		

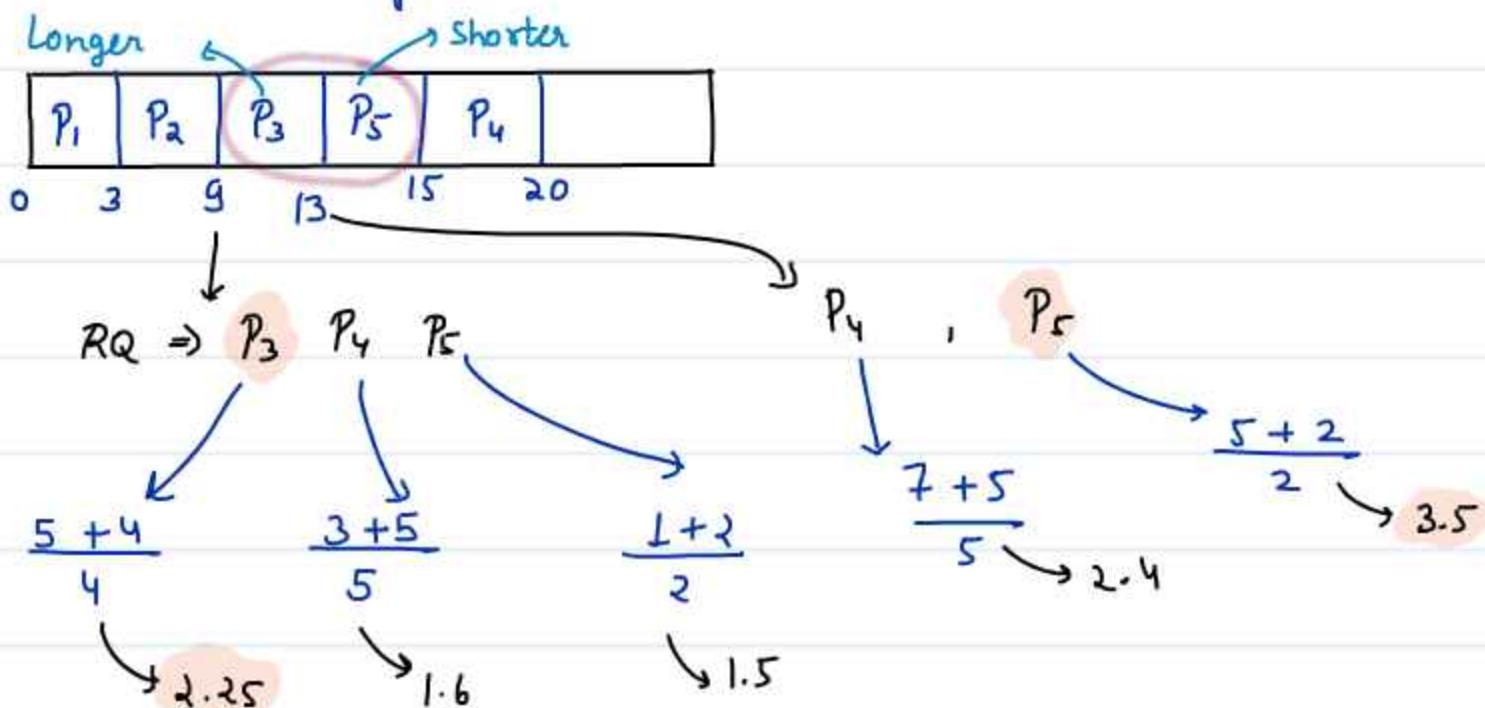
SJF :-



P<sub>5</sub> came at time 8  
P<sub>3</sub> came at time 9  
P<sub>5</sub> got scheduled immediately.

$$\begin{aligned} Wt(P_5) &= 1 \\ Wt(P_3) &= 7 \end{aligned} \quad ] \text{--- } \textcircled{2}$$

Let's solve by HRRN now :-



## # Longest Remaining Time First (SRTF) #

- Selection criteria :- BT
  - Mode of operation :- Pre-emptive
- Not easy as it seems may make mistake

Q

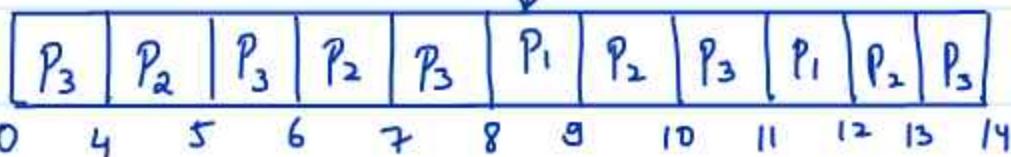
P.No	AT	BT
1	0	2
2	0	4 3 2
3	0	8 4 3 2

Lowest

xx 0 in future.

xx 0

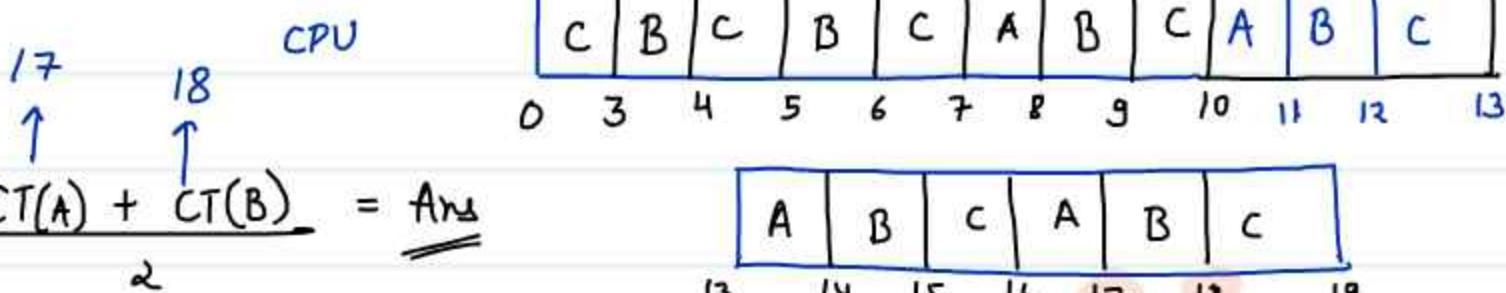
x x 0

Q

Consider 3 processes A, B, C with compute burst times are 4, 6, 9 units. All processes arrive at time zero. Consider the longest remaining time first (LRTF) scheduling algorithm. In LRTF ties are broken by giving priority to the process with lowest process id. Find the average of completion times of A, B?

AT      BT

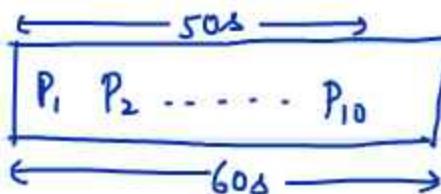
A	0	4	4 3 xx 0
B	0	6 8 4	4 3 xx 0
C	0	9 6 8 4	4 3 xx 0



Q In a system using single processor, a new process arrives at the rate of 10 processes per minute and each such process requires 5 seconds of service time. What is the %CPU utilization?

$$60 \text{ s} \longrightarrow 10 \text{ process} \Rightarrow 6 \text{ s} \longrightarrow 1 \text{ process}$$

$$BT \text{ of each process} = 5 \text{ s}$$



$$\% \text{ CPU utilization} = \frac{50 \times 100}{60}$$

Q Six jobs are waiting to be run. The expected running times are 9, 7, 5, 2, 1 and  $x$  respectively. Where  $5 < x < 7$  and the average completion time is 13. Find the value of  $x$  using SJF algorithm? (Assume all jobs arrive at same time = 0)

AT      BT

$P_1$     0      9

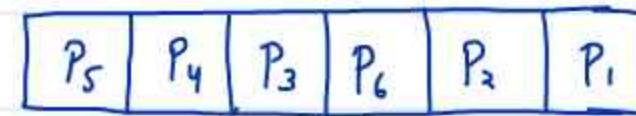
$P_2$     0      7

$P_3$     0      5

$P_4$     0      2

$P_5$     0      1

$P_6$     0       $x$



$$1 + 3 + 8 + (8+x) + (15+x) + (24+x) = 13$$

6

$$x = \frac{78 - 59}{3} = \frac{19}{3}$$

$$= \underline{\underline{3.33}}$$

## # Priority Based Scheduling #

It works exactly like SJF / SRTF except that it looks for Priority instead of B.T.

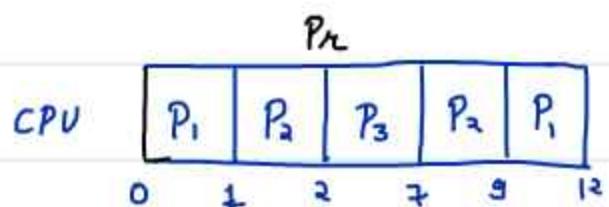
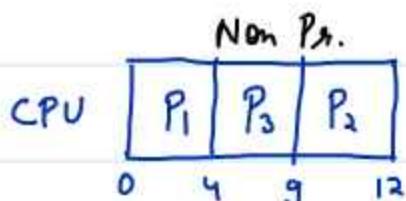
- Selection criteria : Priority (Level of Importance of Process)
- Mode of operation : Non Pre / Pre
- Tie : Lower PID

$$f(\text{Type, Size, Resources...}) = \text{Integer value}$$

Default convention :- Higher Number , Higher Priority

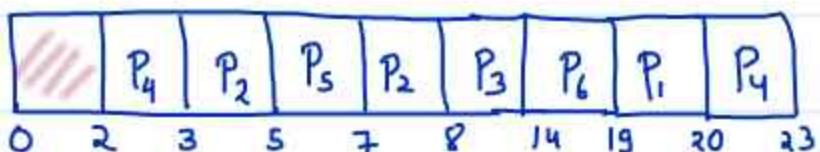
<u>Q</u>	P <sub>no</sub>	P <sub>No</sub>	A.T	B.T
	4	1	0	4 3
	5	2	1	3 2
	8	3	2	5

Solve (Pre & Non Pre)



<u>Q</u>	P <sub>no</sub>	P <sub>No</sub>	A.T	B.T
	4	—	1	4 — 1
	6	—	2	3 — 3
	8	—	3	8 — 6
	3	—	4	2 — 4 2
	7	—	5	5 — 2
	5	—	6	3 — 5

Mode : Pre



$$L = 23 - 2 = 21$$

# CPU Scheduling - 5

Priority Based Scheduling

Priority

Static

Dynamic



Problem  $\Rightarrow$  Starvation to Low Priority Process



use Dynamic Priorities (Aging Algo)



- Low priority process will become High Priority after some time.

Q

Consider a System with Preemptive Priority based Scheduling with 3 Processes P1, P2, P3 having infinite instances of them. The instances of these Processes arrive at regular intervals of 3, 7 & 20 ms respectively. The priority of the Process instances is the inverse of their periods. Each of the Process instance P1, P2, P3 consumes 1, 2 & 4 ms of CPU time respectively. The 1st instance of each Process is available at 1 ms. What is the Completion time of the 1st instance of Process P3?

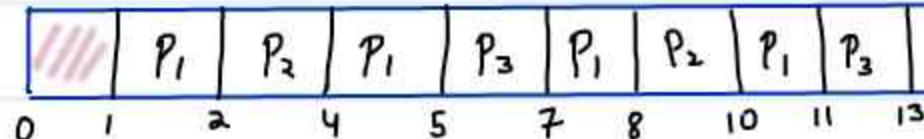
Period      B.T.      Priority

P <sub>1</sub>	3	1	1/3
P <sub>2</sub>	7	2	1/7
P <sub>3</sub>	20	4	1/20

AT of each process = 1 ms

CT of P<sub>3</sub> First = ?

instance



## Aging Algorithm :-

Q

Consider a System using Preemptive Priority based scheduling with dynamically changing priorities. On its arrival a Process is assigned a priority of zero and Running Process Priority increases at the rate of ' $\beta$ ' and Priority of the Processes in the ready  $Q$  increases at the rate of ' $\alpha$ '. By dynamically changing the values of  $\alpha$  and  $\beta$  one can achieve different Scheduling disciplines among the Processes. What discipline will be followed for the following conditions.

1.  $\beta > \alpha > 0$
2.  $\alpha < \beta < 0$

Process arrives  $\rightarrow$  Priority = 0

Process in RQ  $\rightarrow$   $\alpha$  rate  $\uparrow$

Process in Running  $\rightarrow$   $\beta$  rate  $\uparrow$



- (a) if  $\beta > \alpha > 0$  (FCFS)

No Ready Process can't ever preempt the Running process (LCFS)

- (b) if  $\alpha < \beta < 0$   $\rightarrow$   $P_{new}$  has the highest priority then  $P_{ready}$  or  $P_{running}$

## # Round Robin #

→ Used in Preemptive based Multiprogramming time sharing operating system.

→ Improve Interactivity / Responsiveness of sys.

Criteria → A.T. + Time Quantum / Time slice

Mode of Operation → Pre-emptive

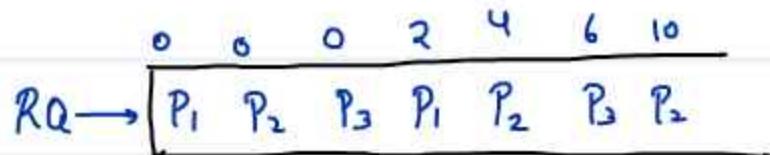
If the Process fails to execute all its instruction in given time quantum then OS will preempt it & it will go to the end of Ready Queue.

TIP :- ALWAYS MAINTAIN THE STATUS OF READY QUEUE  
+ KEEP CUTTING FROM RQ

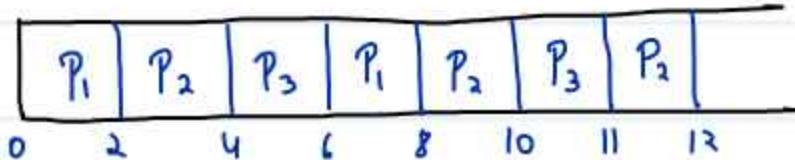
Q

P.No	A.T	B.T
1-	0 -	4 <del>x</del>
2-	0 -	5 <sup>y</sup> 1
3-	0 -	3 <sup>z</sup> 1

$$TQ = 2$$

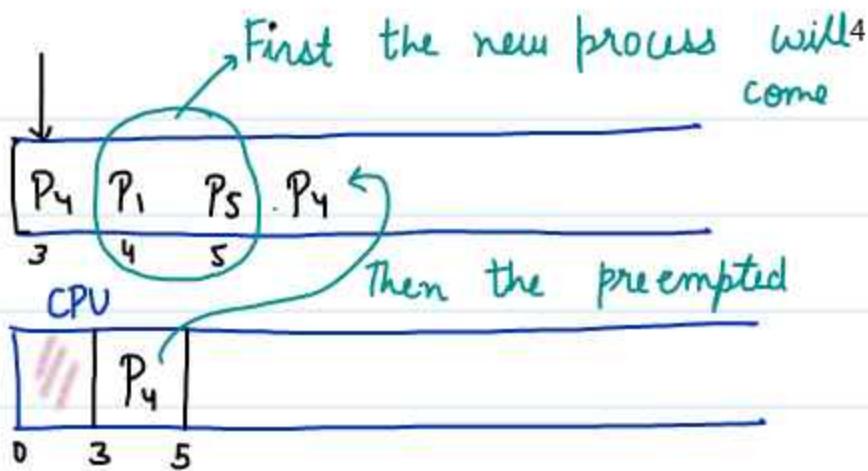


CPU



Q

P.No	A.T	B.T
1	4	1
2	15	5
3	8	6
4	3	3
5	5	4



$$TQ = 2$$

RQ

P4	P1	P5	P4	P1	P3	P5	P3	P2	P3	P2
3	4	5	5	7	8	9	14	15	16	18

CPU

P4	P1	P5	P4	P1	P3	P5	P3	P2	P3	P2
0	3	5	7	9	10	12	14	16	18	20

↓  
Preemption

Q

$$TQ = 2$$

P.No	A.T	B.T	BT; JOBT; BT
1	0	3	<3; 5; 4>
2	2	5	<5; 8; 1>
3	3	2	<2; 2; 2>

1	3	4	X
X	8	X	X
X	X		

P1	P2	P1	P3	P2	P3	P2	P1	P1	P2
0	2	4	6	8	10	12	14	16	18

P1	P2	P1	P3	P2	P3	P2	P1	P1	P2
0	2	4	6	8	10	12	14	16	18

10 → 19 → 20

Q

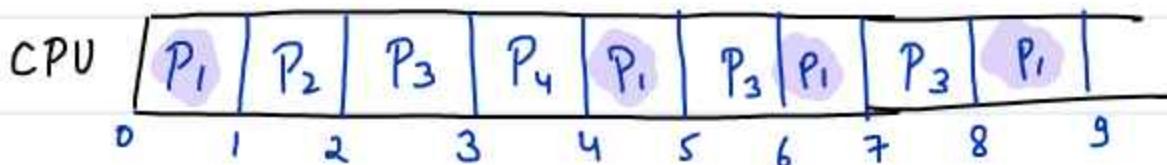
Consider a set of 4 Processes A, B, C, D arriving at time  $0^+$ . Their Burst Time requirements are 4, 1, 8, 1 respectively using Round Robin scheduling with time quantum of 1 unit, The Completion time of Process A is

	AT	BT
P <sub>1</sub>	0	4 3
P <sub>2</sub>	0	1
P <sub>3</sub>	0	8 7
P <sub>4</sub>	0	1

$$TQ = 1$$

$$CT(A) = ? = 9$$

RQ  $\rightarrow$  P<sub>1</sub> P<sub>2</sub> P<sub>3</sub> P<sub>4</sub> P<sub>1</sub> P<sub>2</sub> P<sub>3</sub> P<sub>4</sub>

H.W

Consider a System with 'n' Processes arriving at time  $0^+$  with substantially large Burst Times. The CPU scheduling overhead is 's' seconds, Time Quantum is 'q' seconds. Using Round Robin scheduling, what must be the value of Time Quantum 'q' such that each Process is guaranteed to get its turn at the CPU exactly after 't' seconds in its subsequent run-on CPU.

# Response Time :- Time of admitting the request to the time of generating its result.

→ Ready

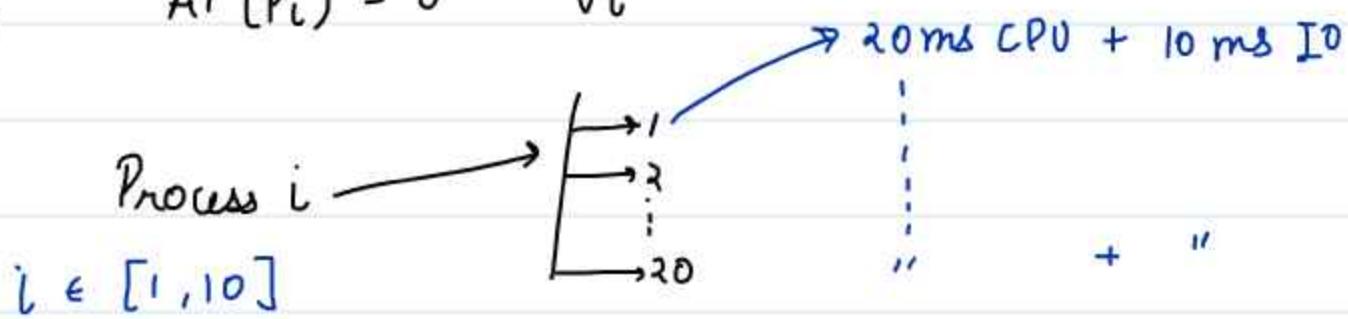
→ end

Q Consider a System using Round Robin Scheduling with 10 Processes all arriving at the time 0. Each Process is associated with 20 identical Request. Each Process request consumes 20 ms of CPU time after which it spends 10 ms of time on I/O, thereafter, initiates subsequent Request. Assuming Scheduling Overhead of 2 ms and Time Quantum of 20 ms, Calculate

- Response time of the 1st request of the 1st Process
- Response time of the 1st request of the last Process
- Response time of the subsequent request of any Process.

Ans

$$AT(P_i) = 0 \quad \forall i$$

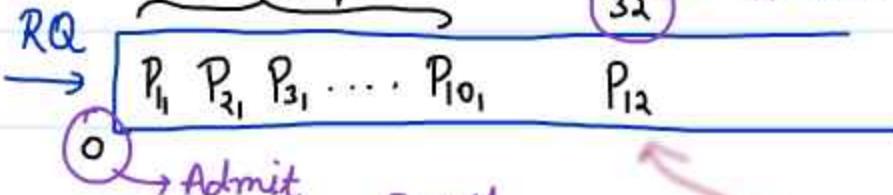


$$\delta = 2$$

$$TQ = 20$$

1<sup>st</sup> Request

32 → Admit



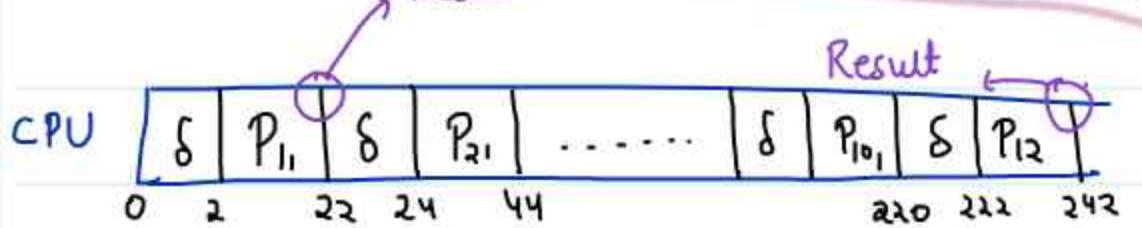
a 22 ms

b 220 ms

c 210 ms

Result

Result



10 32

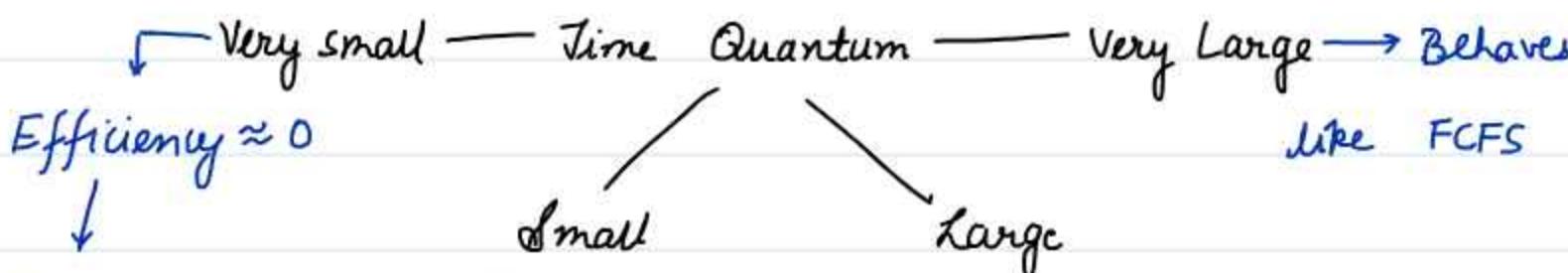
Initiates subseq. Request

# CPU Scheduling - Final

Performance of Round Robin :-

- $T \cdot Q > \max(BT_i)$

- Least Interactive

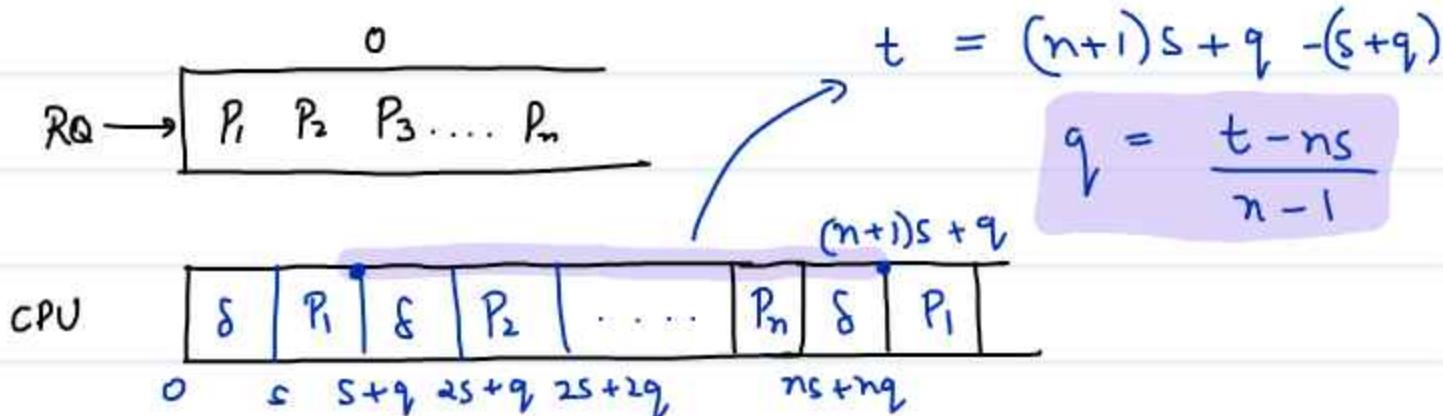


$s$  will take the More Context Switching Less Context Switching

- most of the time
- More Overhead
  - Responsiveness ↑↑
  - Less overhead
  - Interactivity ↓↓

~~Q~~

Consider a System with 'n' Processes arriving at time  $0^+$  with substantially large Burst Times. The CPU scheduling overhead is 's' seconds, Time Quantum is 'q' seconds. Using Round Robin scheduling, what must be the value of Time Quantum 'q' such that each Process is guaranteed to get its turn at the CPU exactly after 't' seconds in its subsequent run-on CPU.



$q = \frac{t - ns}{n - 1}$  Process will get onto CPU exactly after  $t$  seconds

$q \leq \frac{t - ns}{n - 1}$  Process will get onto CPU atleast once within time  $t$

$q > \frac{t - ns}{n - 1}$  Process will get onto CPU after atleast every ' $t$ ' sec

If the value of  $q$  is small  $P_i$  may run multiple time within Time  $t$ .

If the value of  $q$  is large  $P_i$  will wait either  $t$  sec or more than  $t$ .

Q

Consider Processes  $P_1 \& P_2$  arriving in the ready queue at time 0 with following properties.

i)  $P_1$  needs a total of 12 units of CPU time and 20 units of I/O time. After every 3 units of CPU time  $P_1$  spends 5 units on I/O.

ii)  $P_2$  needs a total of 15 units of CPU time and no L/O.  $P_2$  arrives just after  $P_1$ .

Compute the Completion times of  $P_1 \& P_2$  using the following scheduling techniques:

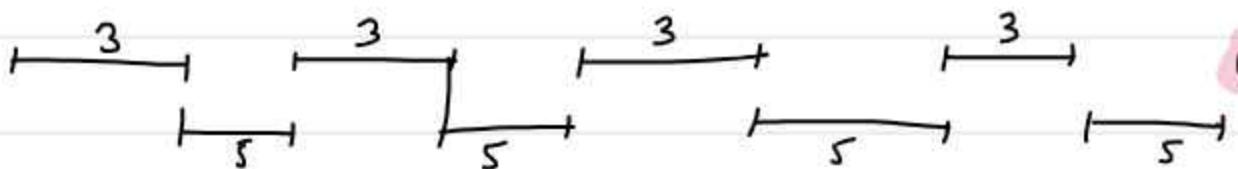
1.SRTF

2.Round Robin with Time Quanta = 4 units

Ans

$P_1 \longrightarrow 12 \text{ units of CPU} + 20 \text{ units of I/O}$

(P<sub>1</sub>)



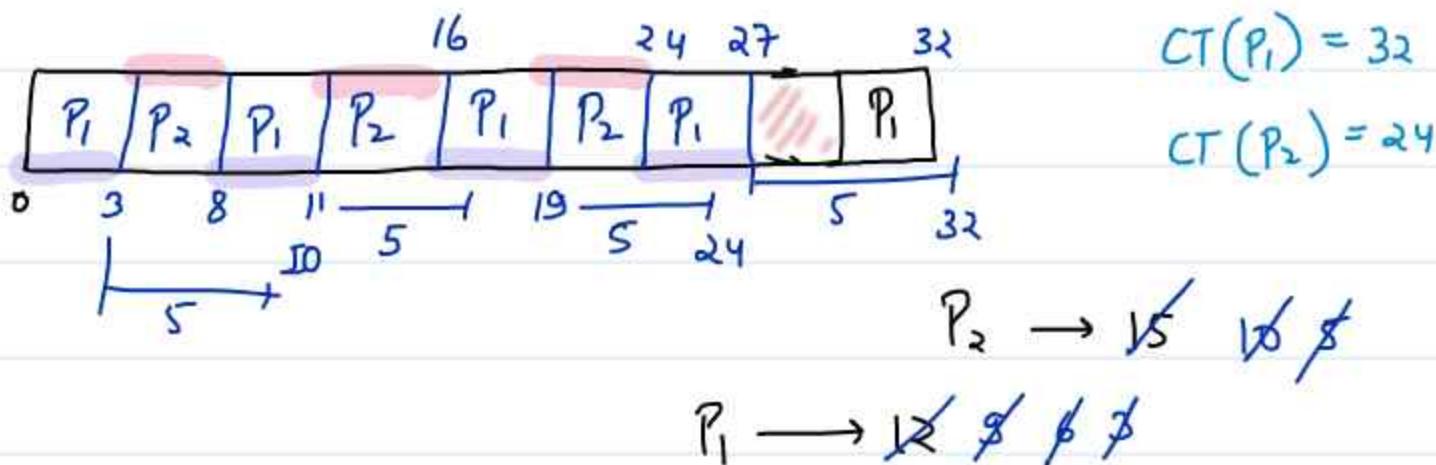
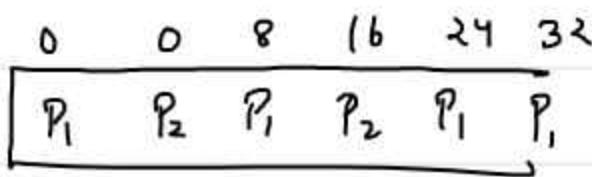
(P<sub>2</sub>)

15 units of CPU + 0 I/O

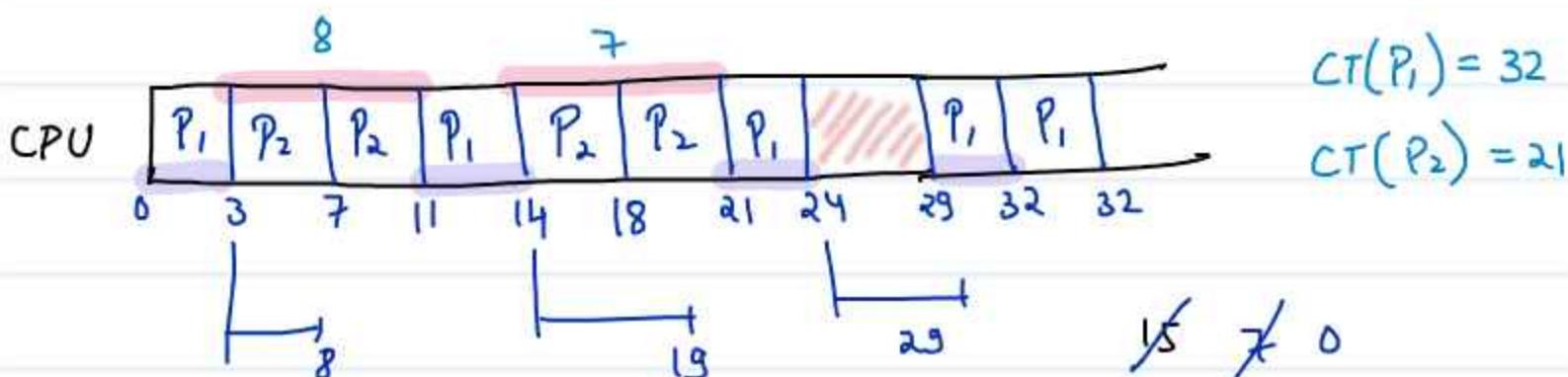
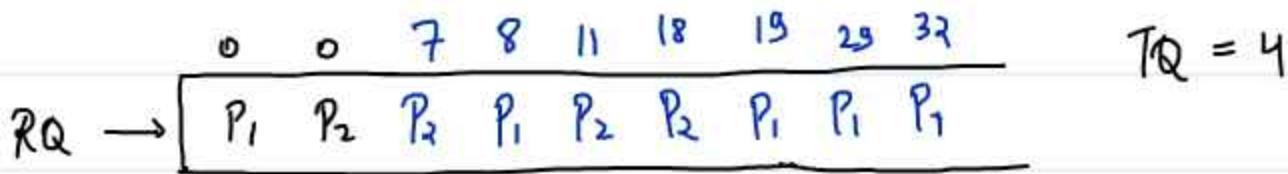
P<sub>1</sub> came first & then P<sub>2</sub>.

(a)

SRTF

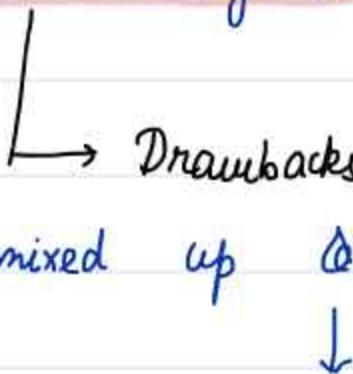


(b)



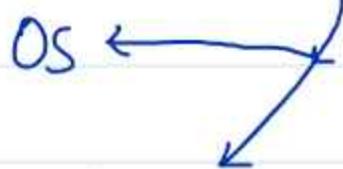
## ☰ Multilevel Queue Scheduling #

All Algo which we have read till now was based on Single Ready Queue System

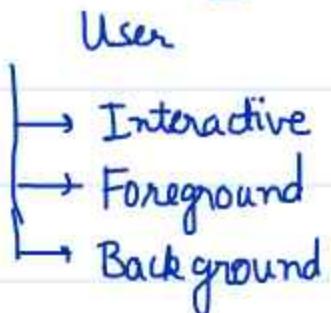


→ Drawbacks :- There are different types of Processes all mixed up & placed in a single Ready Queue.

- High searching & Filtering Time
- Every process is bound to use single scheduling Technique



### MULTIPLE READY QUEUES

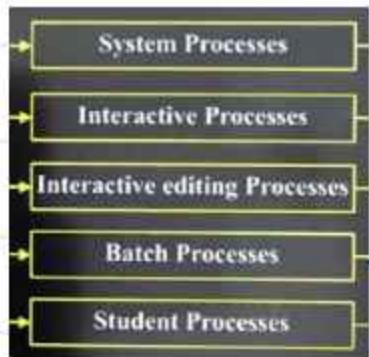


→ Diff Ready Queues for Different Types of Processes

→ Different scheduling algo can be applied to diff Queues

→ Queues are divided based on Priorities

The highest priority processes should be scheduled first then only lower priority processes can be sch.



These 3 higher prio. queues should be empty.

To schedule this

↓ Problem

Starvation to the process lying on the Low Level.

↓ Solution

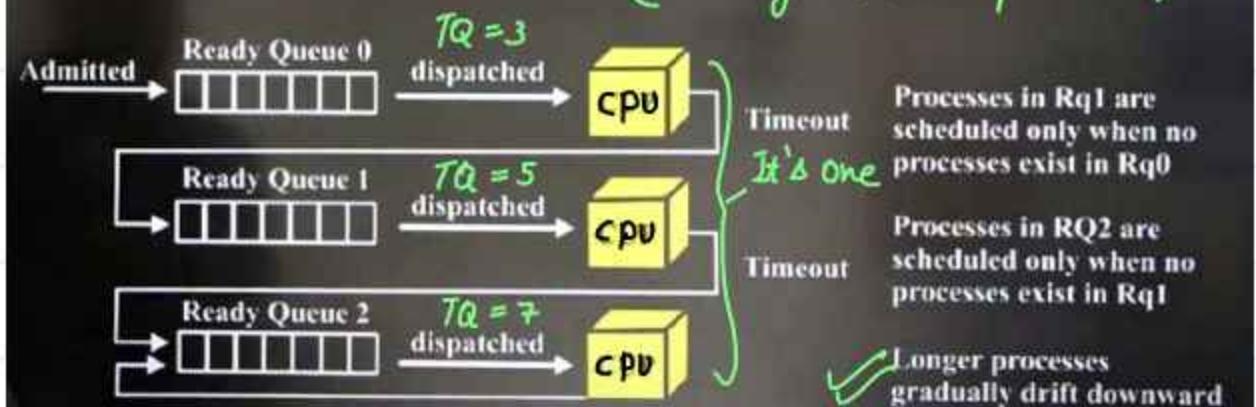
## MULTI LEVEL QUEUE SCHEDULING

Another way to put a preference on short-lived processes

- Penalize processes that have been running longer

Preemptive

(Priority will Keep on ↓)



When process is transferred from RQ0 to RQ1 we will not schedule the process immediately, we will wait until the first queue becomes empty.

Q

Consider a system which has CPU bound process, which require the burst time of 80 seconds, the multilevel feedback queue scheduling algorithm is used and the queue time quantum is '4' seconds' and in each level it is incremented by '10' seconds'. Then how many times the process will be interrupted, and on which queue the process will terminate the execution?

→ no I/O

Ans :-

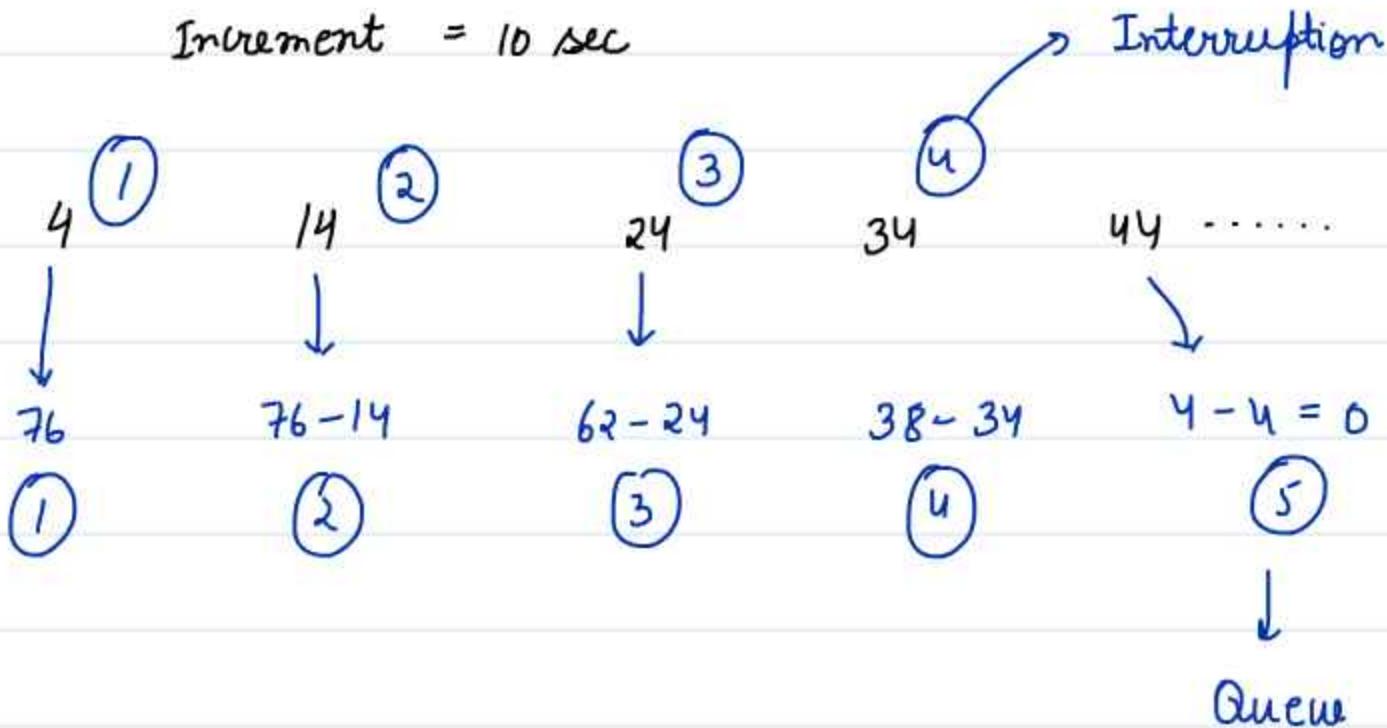
4, 5

Ans

$$TQ = 4 \text{ sec}$$

$$BT = 80 \text{ sec}$$

$$\text{Increment} = 10 \text{ sec}$$



\*———— CPU Scheduling Over ——\*

# IPC & Process Synchronization

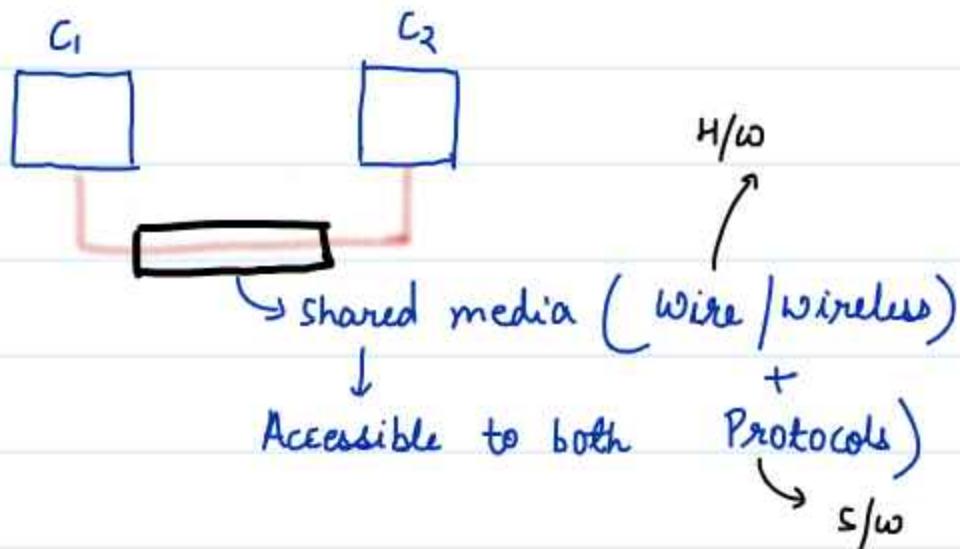


Logical & Reasoning Oriented as CPU Sch.  
was Numerically oriented

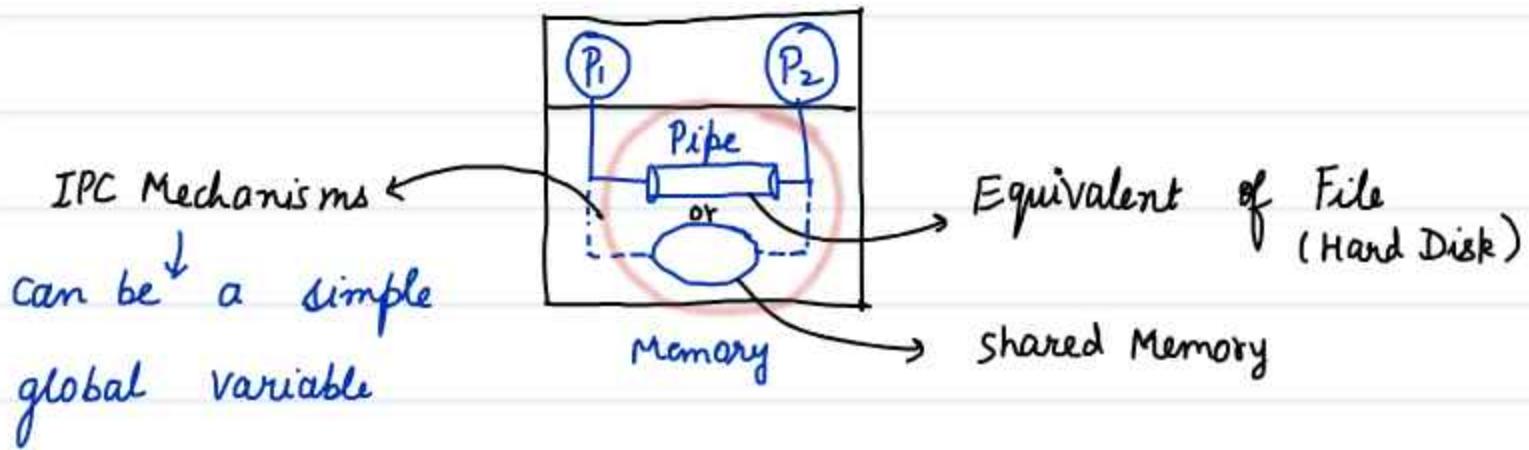
I.P.C. = Inter Process Communication



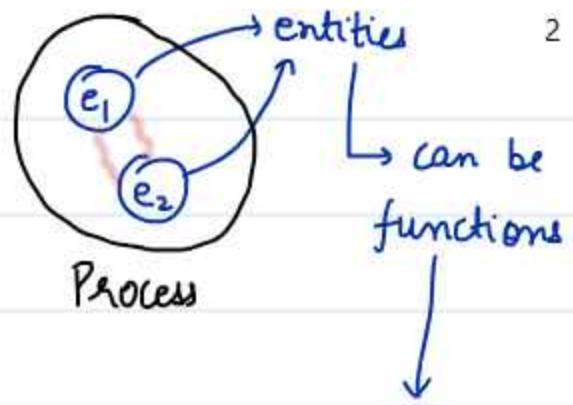
In this world, when two entities wish to communicate, there should be a shared media.



If two process wish to communicate they also needs a shared media.

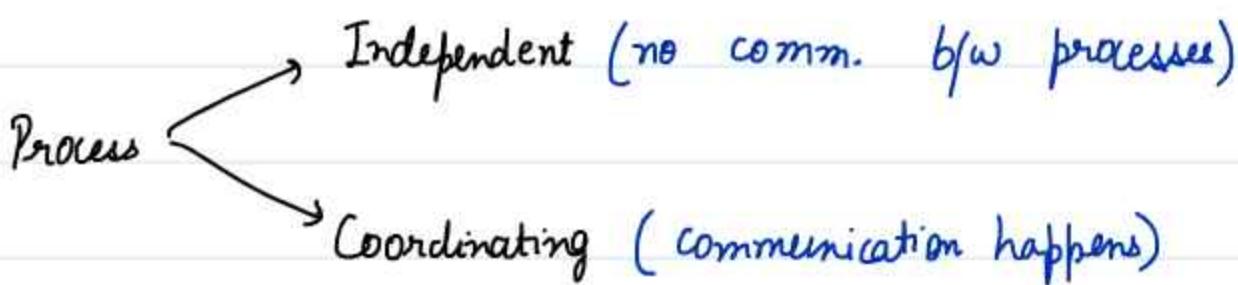


There can be Intra PC too :-



Q How can two  $f^n$  in a program Comm?

1. Parameter Passing
2. Global variables



• Lack of synchronization in IPC Environment will lead to following problems

That's why we need sync

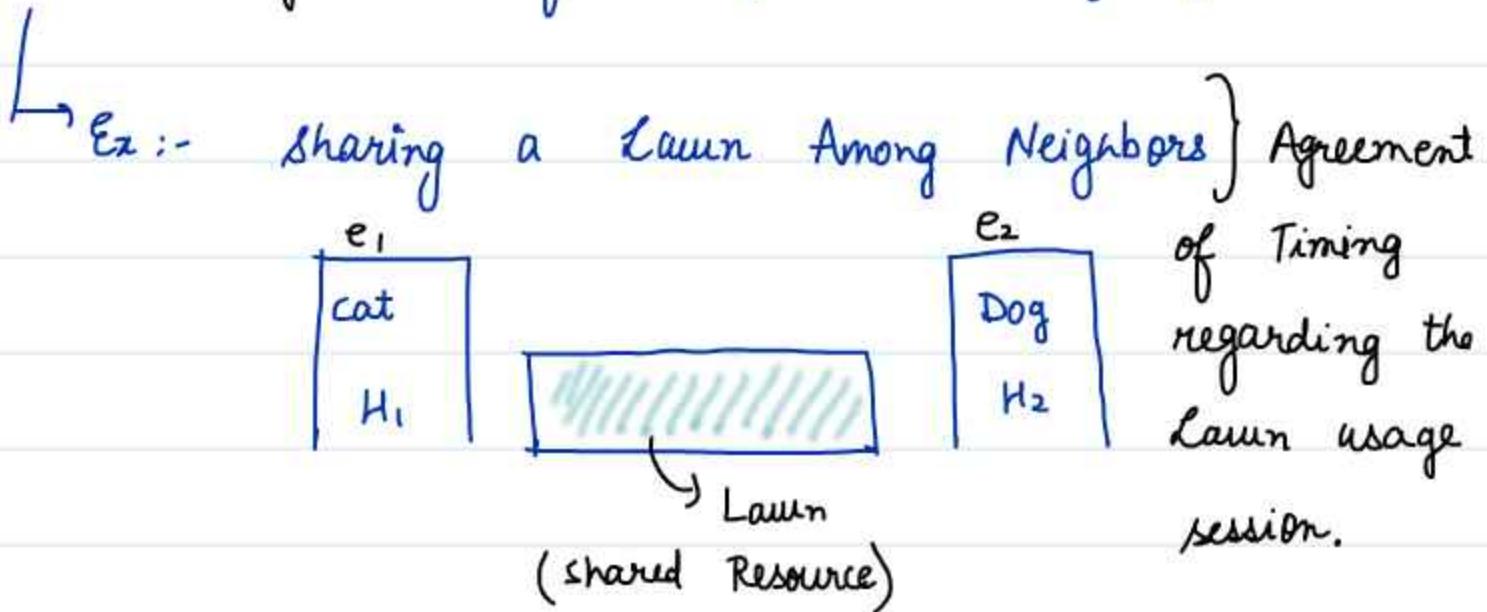
1. Inconsistency / Incorrectness / Wrong Results
2. Data Loss
3. Dead Lock [Lock up] → Infinite Blocking of Processes or

Most undesirable situation of OS

waiting for event

Processes not only get blocked that's never going to happen but they also hold up the resources.

What is sync? Agreement / Understanding b/w entities



Ex :- More Milk Problem



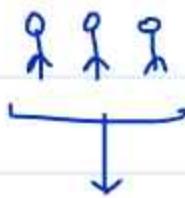
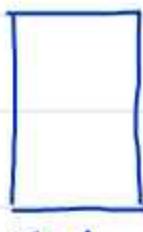
All of them notice one by one  
& goes out to buy milk

↓ Problem

More Milk Problem

↓ solution

Paste it ↓ Note



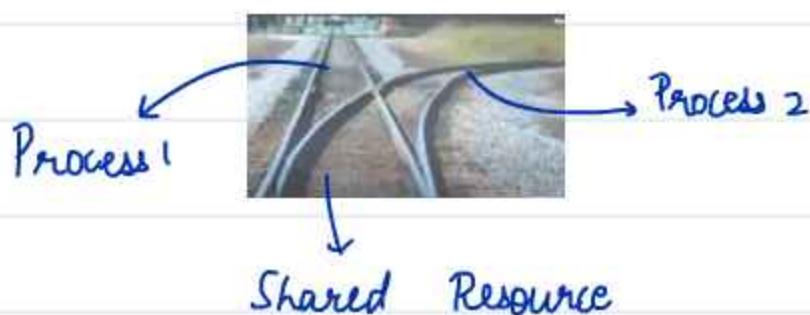
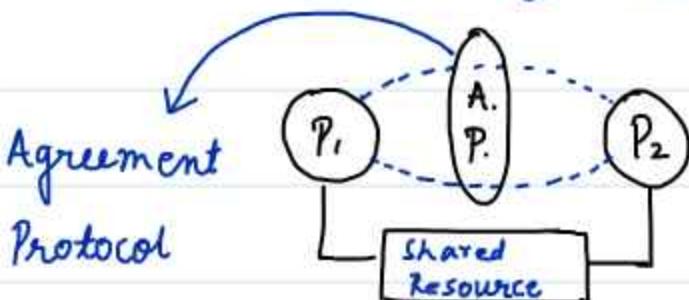
If any one

of us notices that there is  
no milk in the fridge he  
should go out & buy.

If one goes out he paste a note on the fridge preventing  
others to go out.

Process synchronization refers to the coordination and control of concurrent processes in a computer system to ensure they execute properly without interfering with each other. It involves methods and mechanisms to manage access to shared resources, such as data or devices, preventing conflicts and ensuring orderly execution.

Processes sync :- Agreed upon Protocol in IPC environment to avoid Inconsistency , Data Loss or Deadlock.



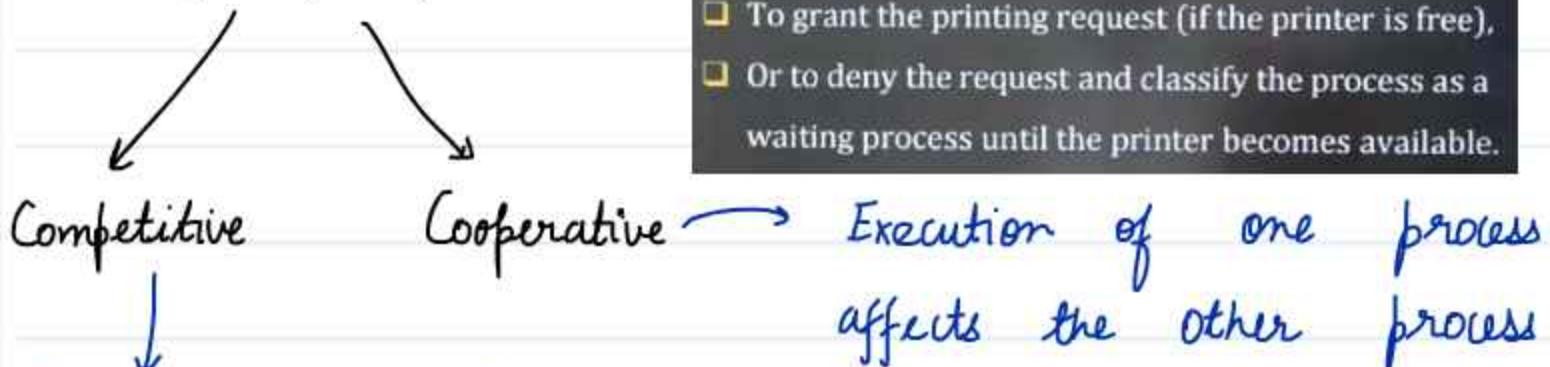
Both Issues a command for printing the file.

Problem May Result print of two diff. Pdfs at the same page  
 ↓ solution

Depending upon whether the printer is already being used by another process or not, the operating system must decide whether:

- To grant the printing request (if the printer is free),
- Or to deny the request and classify the process as a waiting process until the printer becomes available.

## # Types of Sync #



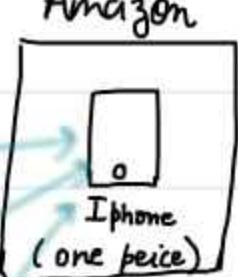
Processes compete for the accessibility of a shared Resource

Ex :- Producer Consumer Problem

Ex :- A ♂

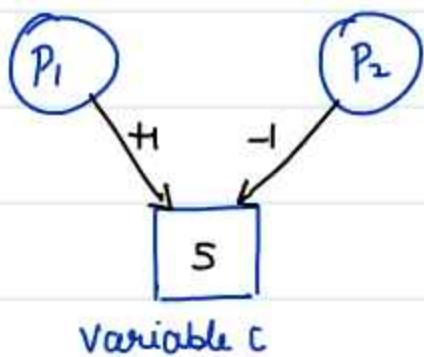
B ♀

C ♂



A, B, & C wants to buy a single piece of iPhone listed on Amazon.

Ex :-



Ex :- 4 people trying to book a single seat in a train

## # Producer Consumer Problem #

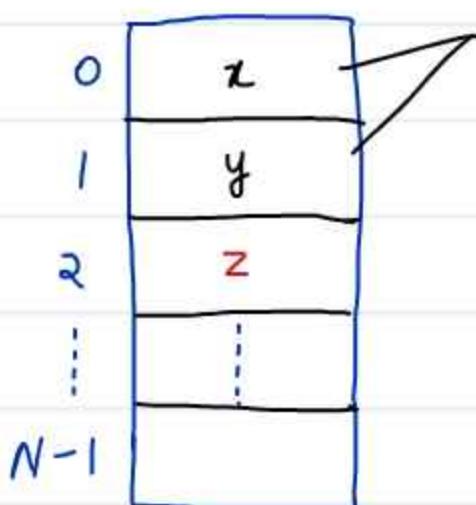
Produces



Consumes



int ~~z~~ 3  
next empty slot



Bounded Buffer

Producer attempts to place the data item onto buffer.

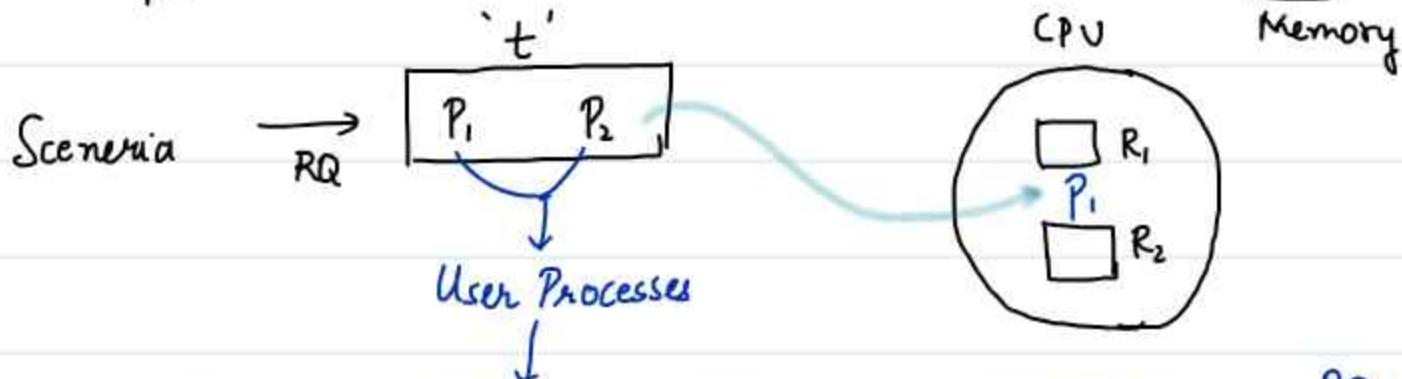
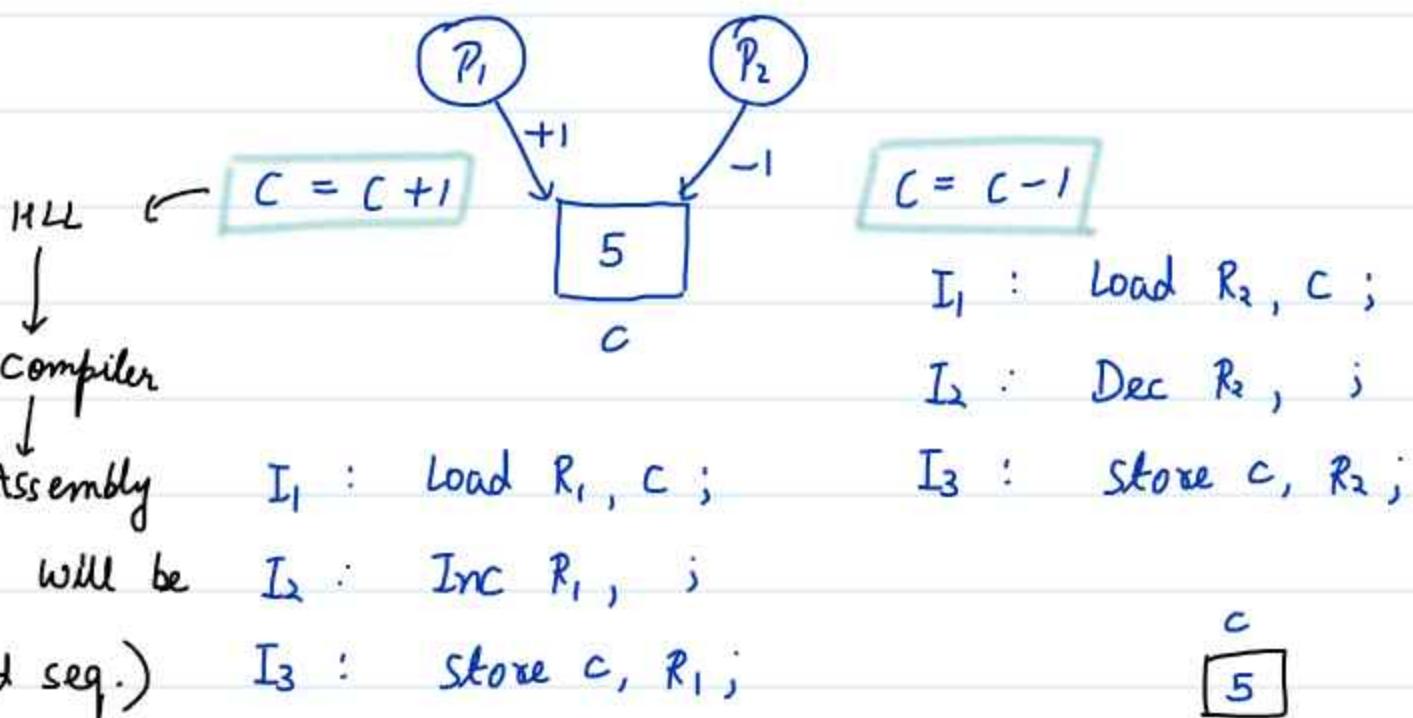
Consumer attempts to consume the data item from Buffer.

Condition → Full Buffer → Producer can't produce.  
Condition → Empty Buffer → Consumer can't consume.

Note - 1 :-	Process	Lack of Syn causes
	Competetive	Inconsistency , Data Loss
	Cooperative	DeadLock

Note-2 :- An application in I.P.C. environment may involve either cooperation / competition or both

# Process Sync Part 2



can get preempted after any instruction

$'t'$   $\rightarrow P_1 : I_1, I_2, P_R$

$\cancel{6}$   $R_1$

$'t_1'$   $\rightarrow P_2 : I_1, I_2, I_3$

$\cancel{4}$   $R_2$

$'t_2'$   $\rightarrow P_1 : I_3$

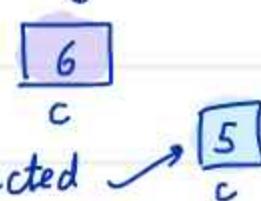
or

$'t_1' \rightarrow P_2 : I_1, I_2, P_R$

$'t_2' \rightarrow P_1 : I_3$

$'t_3' \rightarrow P_2 : I_3$

we expected  $\rightarrow$   $5$   $C$



Whenever a process comes back to CPU it will  
 (Resume ✓) not (Restart ✗).

The above scenario is the clear demonstration of Inconsistency, the process to update last will win.

↓ Then

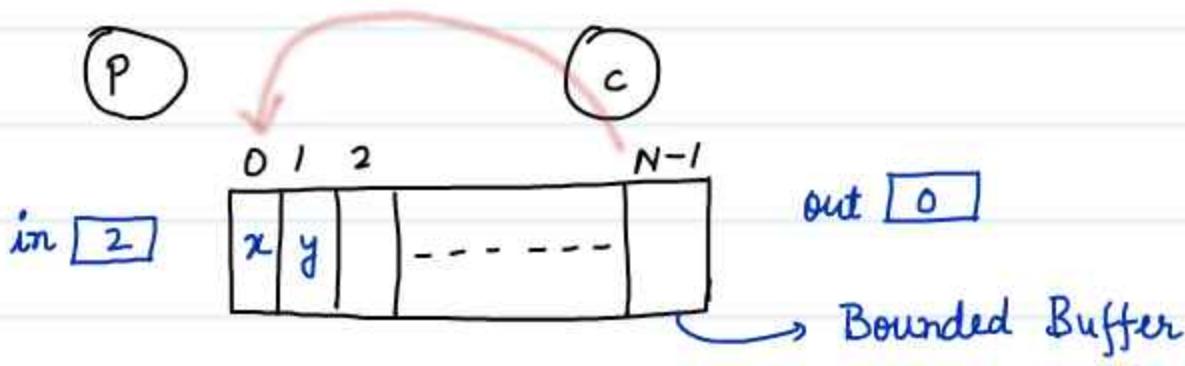
Is it ever possible to get correct value, if yes then in which scenario?

↓

When there is no preemption

But as an end user we want a solution that always gives me correct result whether preemption takes place or not.

## # Implementation of Producer Consumer Problem #



```

#define N 100    int count = 0;      No of data
int Buffer[N];
void Producer(void) {
    int itemp, in = 0;
    while(1)
    {
        a) itemp = ProduceItem();
        b) while(count == N);      Busy waiting
        c) Buffer[in] = itemp;    Loop
        d) in = (in + 1) % N;    Testing the To proceed
        e) count++;
    }
}

```

Produced item is placed in variable itemp.

Function in Library

Busy waiting

Testing the condition again & further again

```

void Consumer(void) {
    int itemc, out = 0;
    while(1)
    {
        a) while(count == 0);      Bounded buffer is
        b) itemc = Buffer[out];
        c) out = (out + 1) % N;
        d) count--;
        e) ConsumeItem(itemc); }
}

```

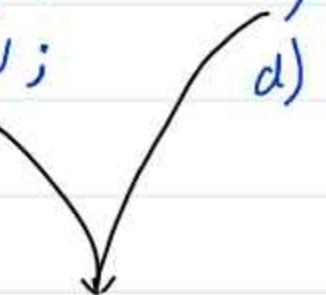
circular.

Producer & Consumer are involved in Cooperative synchronization. because they are getting affected by each other. BUT... There will be competition too to update the shared variable count.

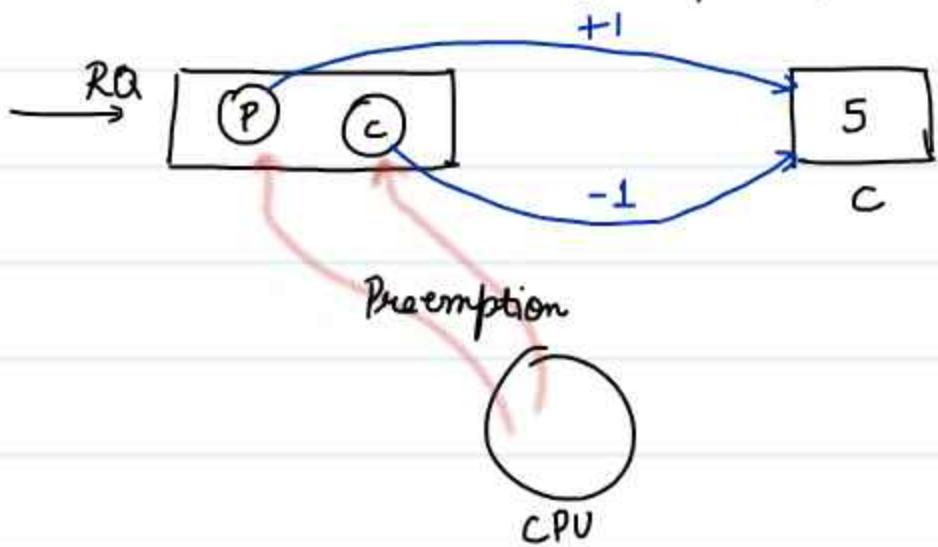


They may try to update it at same time.

- |                                       |                                    |
|---------------------------------------|------------------------------------|
| a) <del>itemp = Produce item();</del> | a) <del>while (count == 0);</del>  |
| b) <del>while (count == N);</del>     | b) <del>itemc = Buffer[out];</del> |
| c) <del>Buffer[in] = itemp;</del>     | c) <del>out = (out + 1) % N;</del> |
| d) <del>in = (in + 1) % N;</del>      | d) <del>count--;</del>             |
| e) <del>count++;</del>                |                                    |



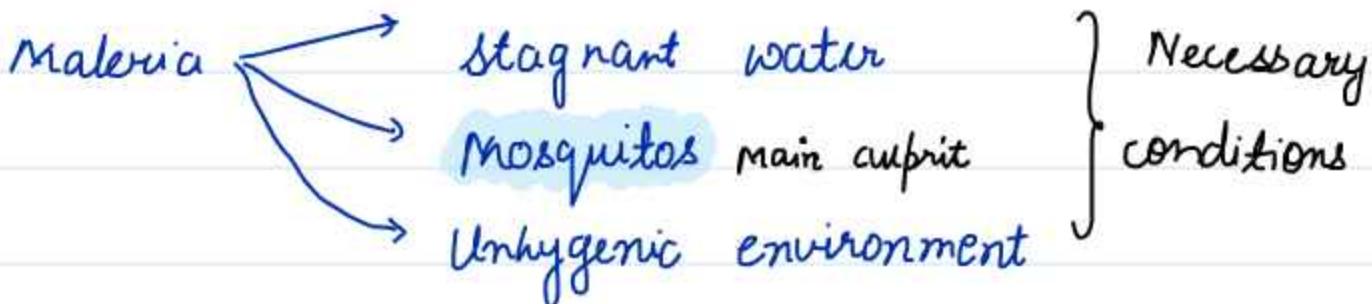
Both get preempted here



} situation is exactly similar to our previous problem that causes inconsistency.

To solve this problem we need a sync. tool.

## # Necessary condition for sync problems #



Sync Prob  $\mapsto$  Critical Section

$\hookrightarrow$  Part of program where shared resource are accessed.



Non critical section

$\hookrightarrow$  No shared resources.

- a) `itemp = ProduceItem(); }`, NCS
- b) `while (count == N); }`
- c) `Buffer[in] = itemp;`
- d) `in = (in + 1) % N;`
- e) `count++;`

CS

$\mapsto$  Race Condition

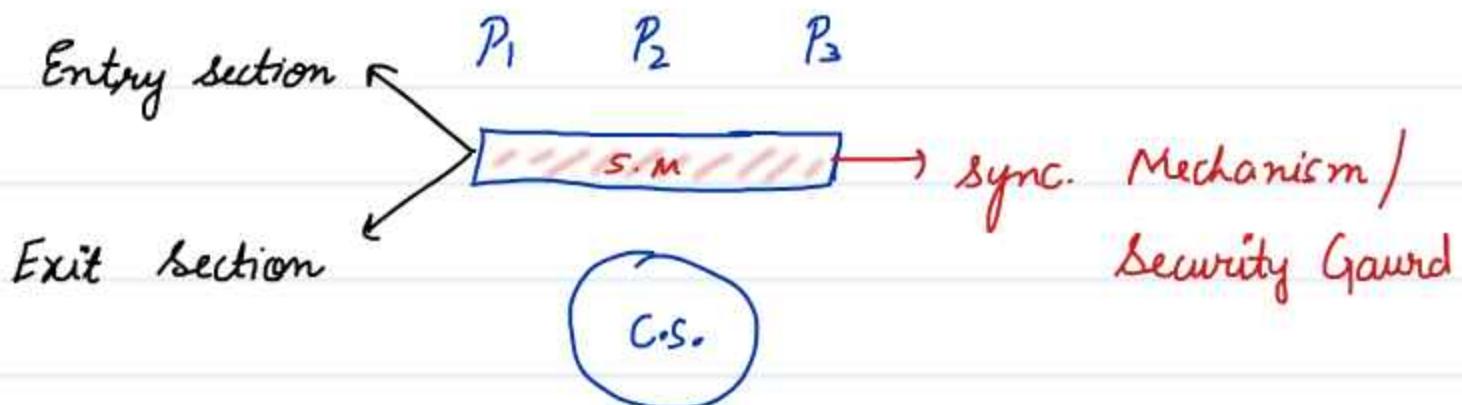
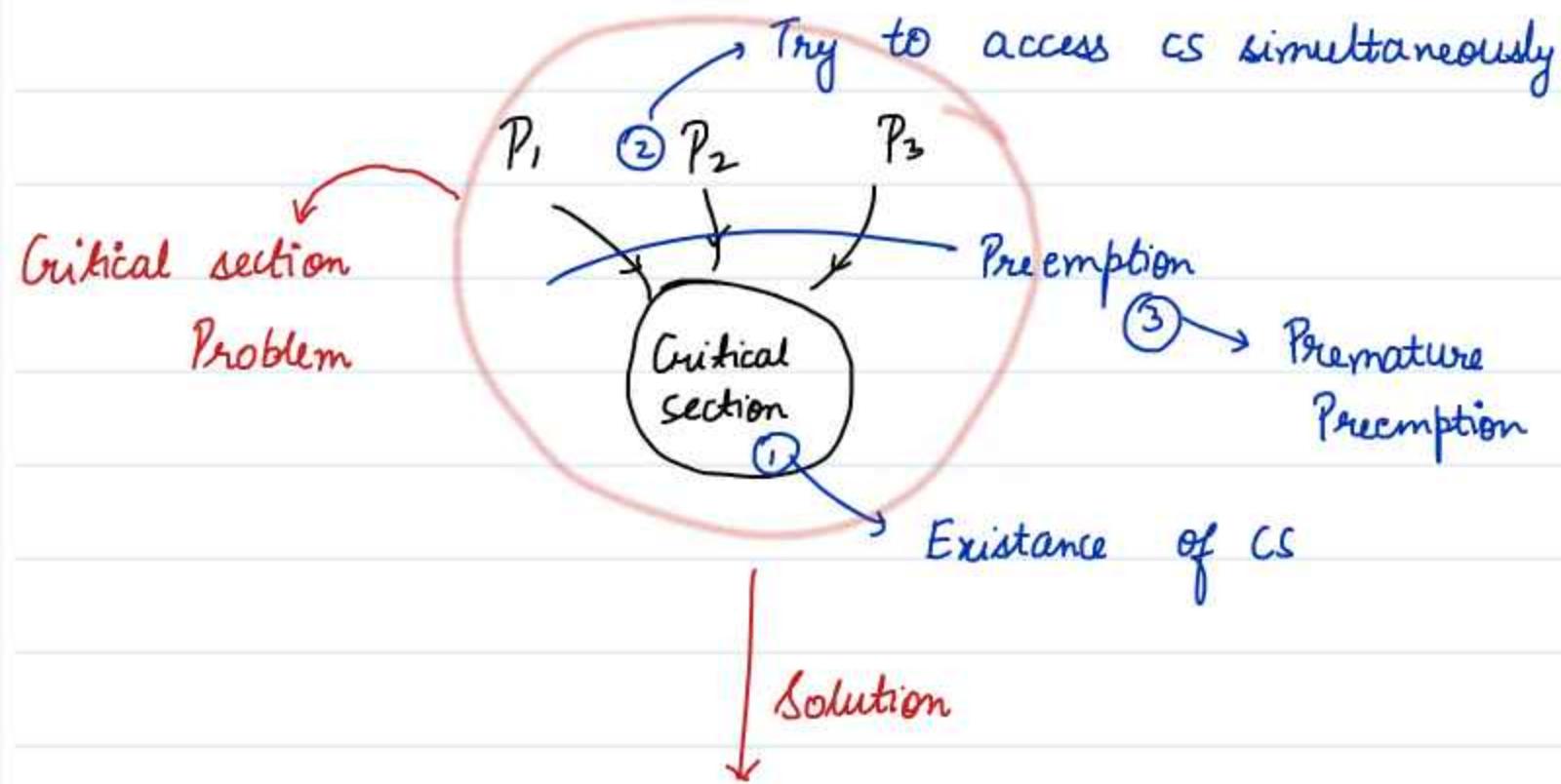
Situation wherein processes are

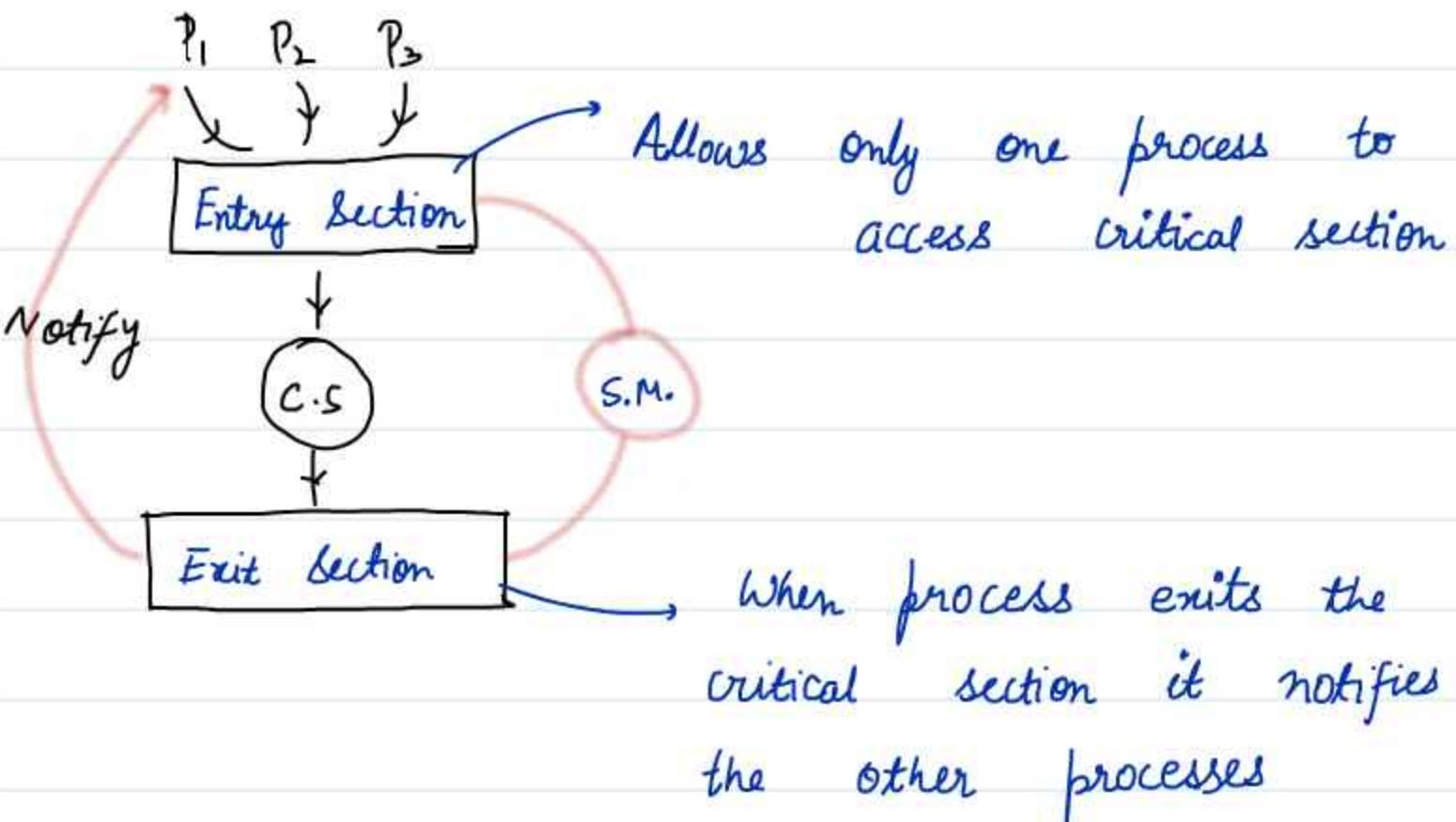
trying to access  $\langle CS \rangle$  &

final result depends on the order they finishes their update.

Main Culprit

$\rightarrow$  PREMPTION (Premature)





< Non Critical Section >

Entry Section  
 < C.S. >  
 Exit Section

} Sync. Mechanism

< Non Critical Section >

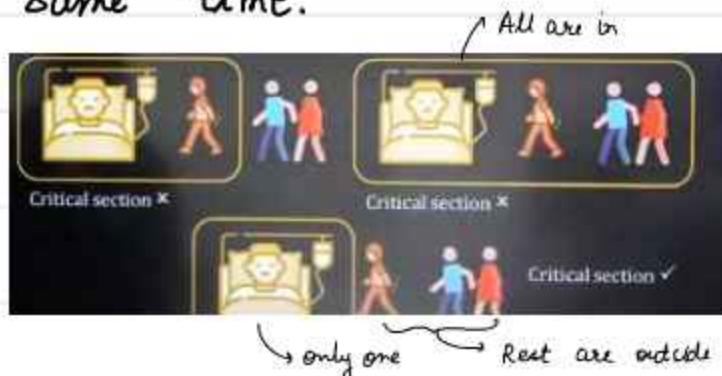
NOTE :- Process running in user mode can get Preempted anywhere, any number of times, after completing any instruction

# Requirements of C.S. Problem → to be satisfied by Sync. Mechanism

## ① Mutual exclusion



No two process should be present in C.S. at the same time.

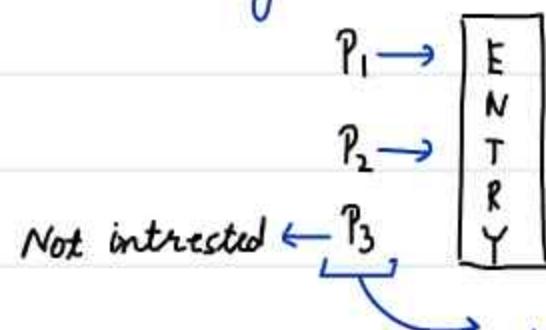


Necessary conditions :-

conditions which are necessary for the problem to happen.

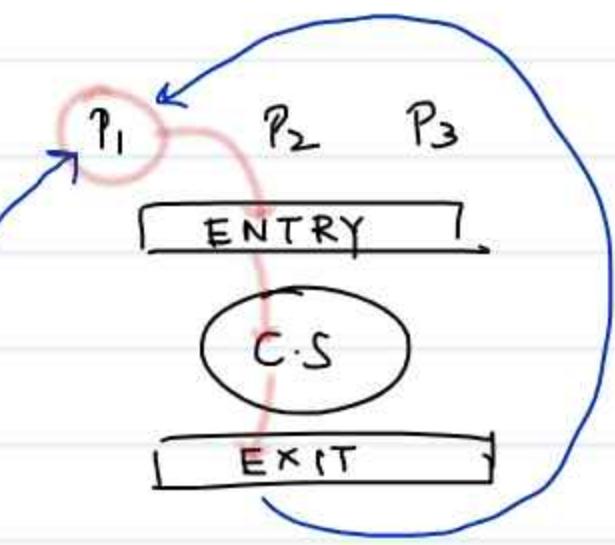
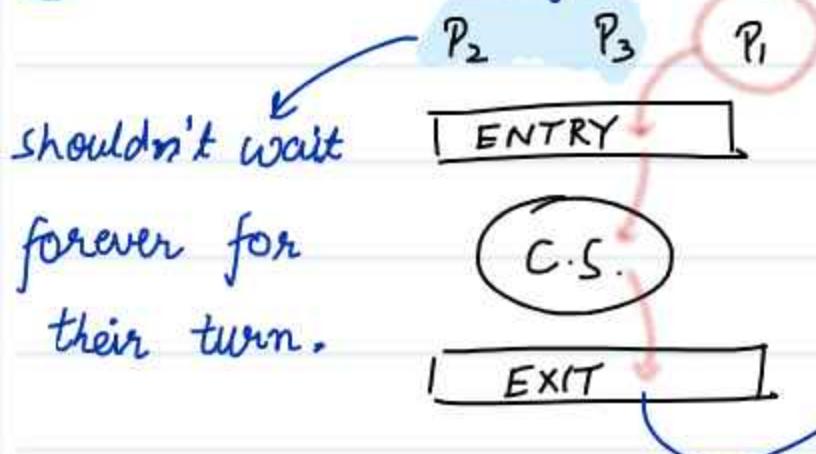
Requirements :- These are to be fulfilled to make sure the solution is correct.

## ② Progress



Has no right to block the progress of others.

## ③ Bounded Waiting



No process has to wait to access C.S. There should be a bound on no. of times a process is allowed to enter C.S. before other process request is granted.

- \* If  $P_2 \& P_3$  are not interested then  $P_1$  shouldn't be blocked otherwise PROGRESS will be violated.

- If Mutual exclusion is not guaranteed then
  - Inconsistency + Loss of Data
- If Bounded waiting " " " " then
  - Starvation
- If Progress " " " " " then
  - unfair sol<sup>n</sup> (Indefinite Postponement)

## # Synchronization Mechanisms #

Busy waiting  
(Spin Lock)

Non Busy waiting  
(Blocking)

subcategorized



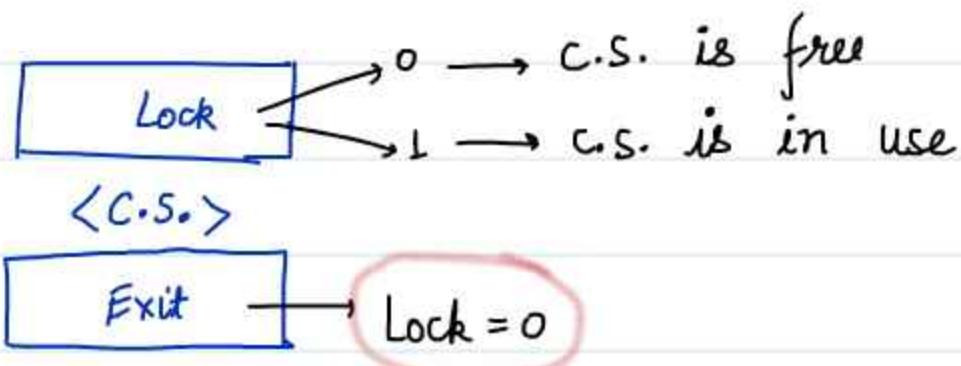
## # Assumptions #

- Process enters  $\langle CS \rangle$  for finite amount of time and come out of it. It never gets stuck in critical section.
- If a process is in Entry section than It means it is interested in  $\langle CS \rangle$ , otherwise it won't enter in Entry section.
- A process is said to have left  $\langle CS \rangle$  only when it has executed its exit section.
- A process can get preempted from CPU while Executing either Entry section or  $\langle CS \rangle$  or Exit section.

# PROCESS SYNC PART - 3

## Lock Variable

- Busy waiting solution
- Software solution Implementable @ VM.
- Multi Process solution



## # Implementation :-

### Entry section

```

Load Ri, Lock
Cmp Ri, #0
JNZ step b
Store lock, #1
  
```

Initially the CS. is free

int lock = 0;

void Process ( int i )

{ while(1) {

a). < Non C.S. >

b) while (lock == 1);

c) lock = 1;

d) < C.S. >

e) lock = 0; }

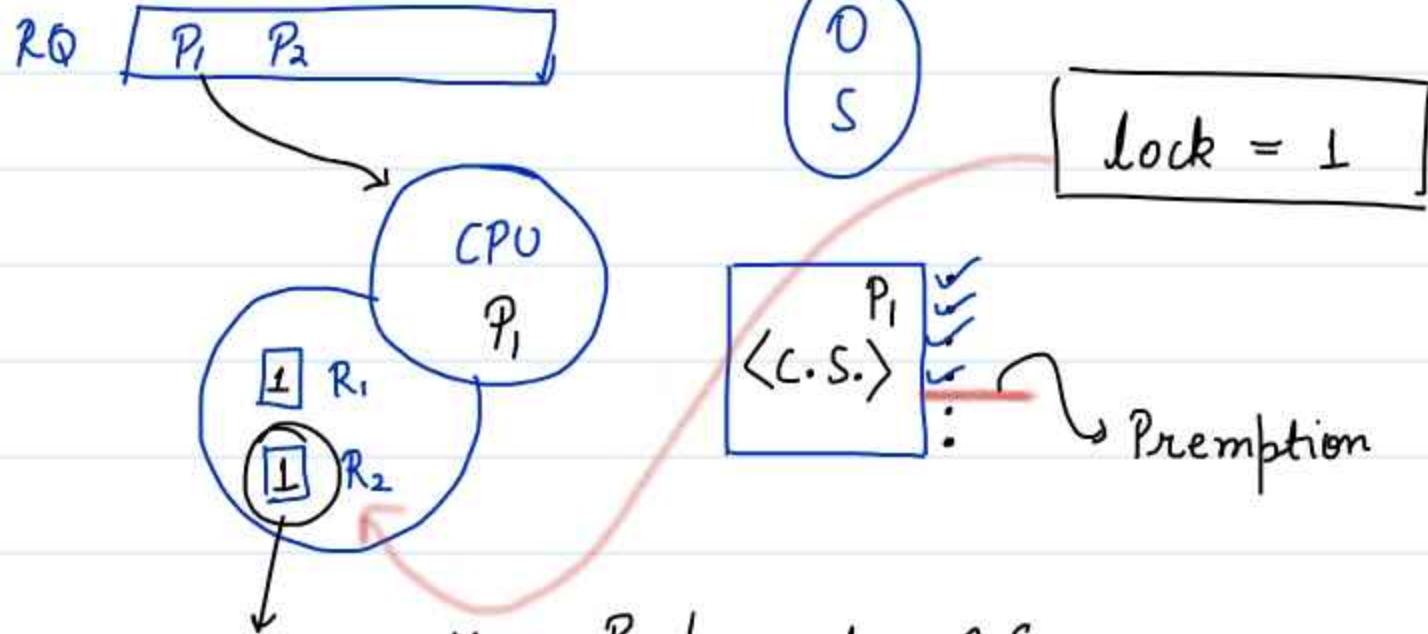
Entry

Exit

Store lock, #0

Exit section

JNZ = Jump if not zero.

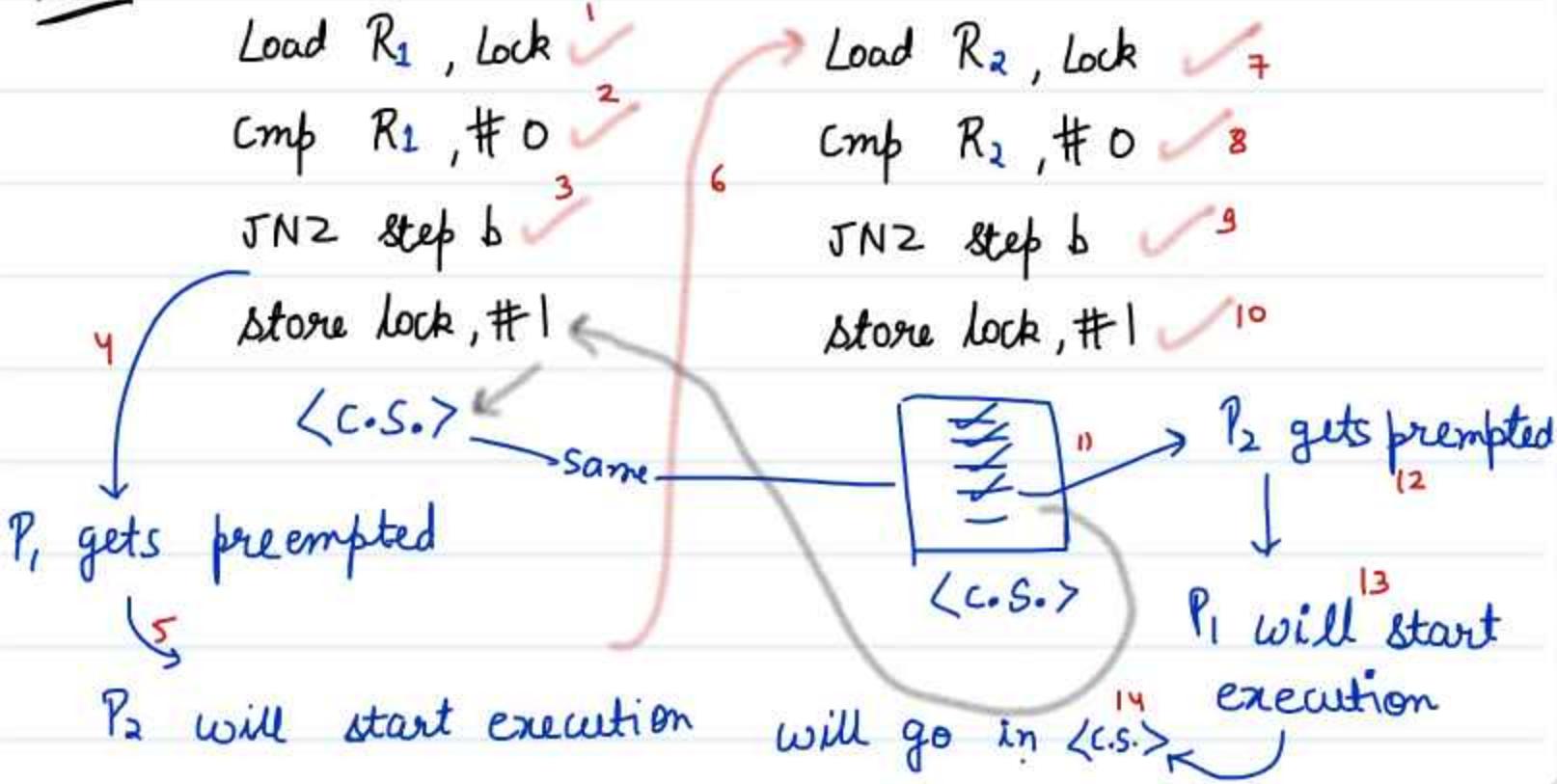


will not allow  $P_2$  to enter C.S.

will be busy waiting until  $P_1$  comes back exits <C.S.> by updating the lock variable

Q Does this Lock guarantees mutual exclusion everytime?

Ans :-



Both processes are in the CS



Mutual Exclusion is violated

Is Progress guaranteed?



→ can non interested process prevent interested process from going into the <CS>.

YES.

let say :-

< Non C.S. >

while ( $lock \neq 0$ );

$lock = 1;$

$P_1 \rightarrow$  Not Interested

$P_2 \rightarrow$  Interested

$P_1$  will be busy here,  
lock value will always be  
zero for  $P_2$ .

Is waiting bounded?

→ a process should not go into <CS>  
when there exist an other interested process.



NO, there is no mechanism to stop  $P_1$ .

Suppose  $P_i$  exits  $\langle C.S. \rangle$  will make  $lock = 0$  and quickly joins  $RQ$  to go into  $\langle C.S. \rangle$  again.

↓  
A process can successfully enter  $\langle C.S. \rangle$  multiple times while others are waiting for their turn to enter  $\langle C.S. \rangle$

- Lock variable is a busy waiting solution leading to wastage of CPU-time / cycles.

Several concurrent processes are attempting to share an I/O device.

In an attempt to achieve Mutual Exclusion each process is given the following structure. (Busy is a shared Boolean Variable)

<code unrelated to device use>

Repeat

until  $busy = \text{false}$ ;

$Busy = \text{true};$

<code to access shared>

$Busy = \text{false};$

<code unrelated to device use>

Non  $\langle C.S. \rangle$

$busy \sim Lock$   
 $false \sim 0$

$true \sim 1$

entry

$\langle C.S. \rangle$

exit

Non  $\langle C.S. \rangle$

} Similar to what we discussed

Which of the following is (are) true of this approach?

I. It provides a reasonable solution to the problem of guaranteeing mutual exclusion. ✗

II. It may consume substantial CPU time accessing the Busy variable. ✓

III. It will fail to guarantee mutual exclusion. ✓

# Strict Alternation #

→ Busy waiting

→ s/w sol" implementable

2 process solution  $\langle P_i | P_j \rangle$

at UM.

Strictly on alternate basis, process takes turn to enter critical section.

```

int turn = rand(0,1);

void Process(0) {
    while(1) {
        a) Non C.S.();
        b) while (turn == 1);
        c) <c.s.>
        d) turn = 1;
    }
}

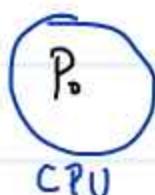
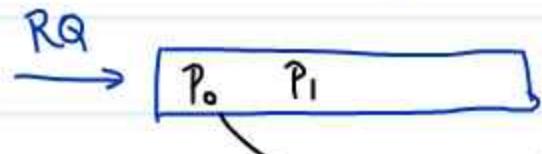
void Process(0) {
    while(1) {
        a) Non C.S.();
        b) while (turn == 0);
        c) <c.s.>
        d) turn = 0;
    }
}

```

Here mutual exclusion is guaranteed

Lock  $\xrightarrow{0}$  CS is free  
 $\xrightarrow{1}$  CS is not free

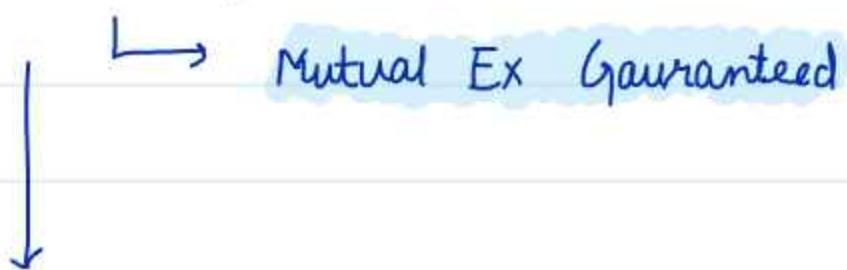
Turn  $\xrightarrow{i}$  It's  $P_i$ 's turn to enter <CS>  
 $\xrightarrow{j}$  It's  $P_j$ 's turn to enter <CS>



turn = 0 (Randomly)

<C.S.>  $\xrightarrow{\text{exit}}$  P<sub>0</sub> gets out

Now  $P_1$  can't enter critical section until  $P_0$  set the value of turn as 1.



can you infer more?

Let's say  $\text{turn} = \text{rand}(0,1)$  gives 1. It means it's  $P_1$ 's turn to enter C.S. but she is not interested to enter critical section.



If she doesn't enter <C.S> then she will never execute the exit section



turn will never be set to 0.



$P_0$  is blocked by a non interested one.



Progress is hindered.

\* Waiting is bounded as there is strict alteration.

	Lock	Turn
Mut.Ex.	X	✓
Progress	✓	X
Bounded waiting	X	✓
Busy waiting	✓	✓
	wastage of CPU cycles	
Number of Process	Multi Process sol"	Two Process sol"

The value of turn need not to be equal to 0/1  
 It can be any  $i, j$ . Let's generalize.

int turn = rand(i, j);

void Process (int i){  
 while (1) {  
   a) Non C.S.();  
   b) while (turn == j);  
   c) C.S.();  
   d) turn = j; } }

void Process (int j) {  
 while (1) {  
   a) Non C.S.();  
   b) while (turn == i);  
   c) C.S.();  
   d) turn = i; } }

Lock & Strict Alternation  $\longrightarrow$  Incorrect solution

H.W.

Q      int turn = rand(i, j)

Void Process ( int i )

{      int j = NOT(i);

while (1)

{      a) Non C.S.();

b) While ( turn != i );

turn = j;

c) <C.S.>

d) turn = j;

MEx ?

Progress ?

Bounded wait?

}

## # PETERSON's SOLUTION #

→ Busy waiting

→ software solution Implementable @ UM

→ 2 - process solution

Combination of Lock variable + Strict Alteration



# define N 2

using good properties of

# define True 1

# define False 0

9

```
int flag[N] = {false}; → array of size N init to  
int turn;  
false.
```

i is the Running flag → Process is interested  
void Process (int i) } Process or not.  
{ int j = NOT(i);      ↓  
true  
false

while (1)  
{

P<sub>i</sub> is interested when interested will change

Initially no one is interested  
the flag value.

a). Non CS(); ↑

b). flag[i] = true;      ↓  
global var.

c). turn = i;

concept of turn will be  
used when both of them  
are interested simultaneously.

d). while (flag[j] == T  
      &&  
      turn == i);



turn = 0 → P<sub>0</sub> updated  
= 1 → P<sub>1</sub> updated

e). <C.S.>

The process to update the  
turn last will wait.

f). flag[i] = false;

→ P<sub>1</sub> :- turn = 1

Exit section } }

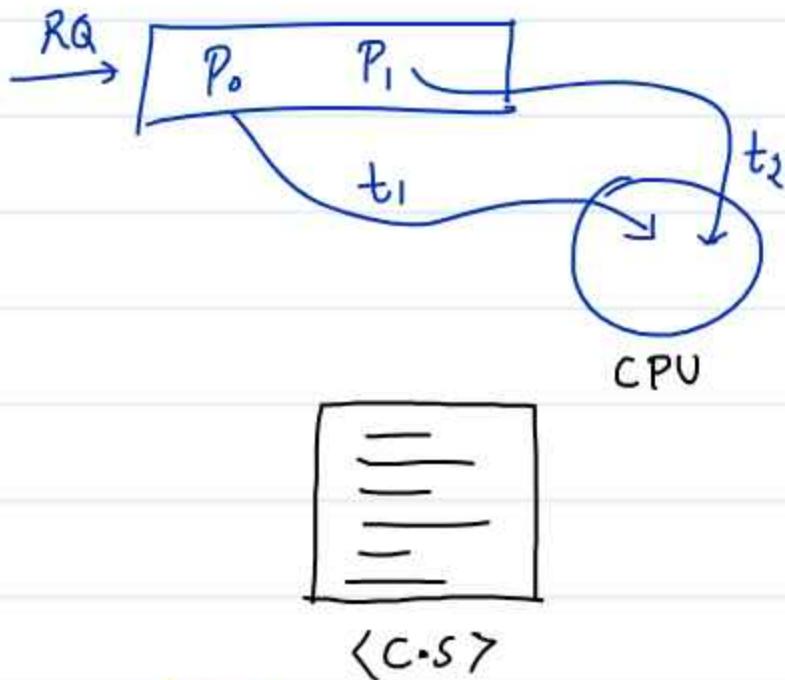
& later it found that  
value of turn is still 1  
it means it was the last.

## SUMMARY



- ① change the flag value to true.
- ② set turn equal to id. the last one to update.
- ③ check if other process is interested & I am the last one to update turn then I should busy wait.

# Now Let's see whether it's a correct solution #



$$\begin{aligned} \text{flag}[0] &= F \\ \text{flag}[1] &= F \end{aligned}$$

$$\begin{array}{|c|} \hline \emptyset \perp \\ \hline \text{turn} \\ \hline \end{array}$$

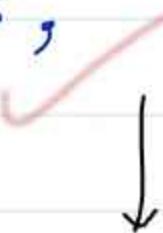
t<sub>1</sub>: (P<sub>0</sub>)<sub>CPU</sub> i=0, j=1 | NCS, flag[0]=T, turn=0, Pre

$t_2 : \left( \begin{array}{l} P_1 \\ CPU \end{array} \right) i=1, j=0 \quad \left| \begin{array}{l} NCS, \text{ flag}[1] = T, \text{ turn} = 1, \end{array} \right.$

11

(Last one to update) busy waiting,  
Pre.

$t_3 : \left( \begin{array}{l} P_0 \\ CPU \end{array} \right) i=0, j=1 \quad \left| \begin{array}{l} \text{No wait, Enters C.S,} \\ (\text{Flag}[j] = T \checkmark) \\ \text{turn} = 0 X \end{array} \right.$



Guarantees Mutual Ex.

When both process are interested-

Now we have a concept of flag. If other process is not interested then it won't let other process busy wait because of her.

current process code {

while ( flag[other process] == T & & .... )

non interested process

False



Guarantees Progress

③ Bounded wait  $\rightarrow$  Guaranteed!

Let's say  $P_0$  completes CS & it quickly completes its rest of the instruction to enter CS again while other process is still waiting



Is this possible?

NO)

We have the concept of turn, if  $P_0$  wants to go again it can't go because if

$\langle C.S. \rangle \checkmark$



Exit section :-  $flag[0] = F$



Entry section :-  $flag[0] = T$   
 $turn = 0$

$\boxed{P \neq 0}$

Busy wait



$while (flag[1] == T \& \& turn = 0)$



can't enter again

$P_0$  was the last one to update

# PROCESS SYNC Part 4

Q int turn = rand(i, j)

turn  $\xrightarrow{i}$   
 $\xrightarrow{j}$

. void Process (int i)  $\rightarrow$  code of Running Process  
 { int j = NOT(i);

while (1)

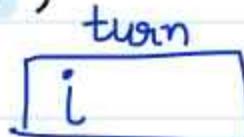
{ a) Non C.S.();

$P_i \rightarrow$  Non CS

b) while (turn != i);

$\downarrow$

turn = j;



while (turn != i)

$\downarrow$

c) <C.S.>

$\downarrow$

d) turn = j;

$P_j \rightarrow$  Non C.S.

$\downarrow$

}

while (turn != i)

turn = j

$\downarrow$

Progress?  $\rightarrow$  Nah  
 $\hookrightarrow$  turn = j

turn = j

$\downarrow$

C.S.

$\downarrow$

Preempt

$\searrow$

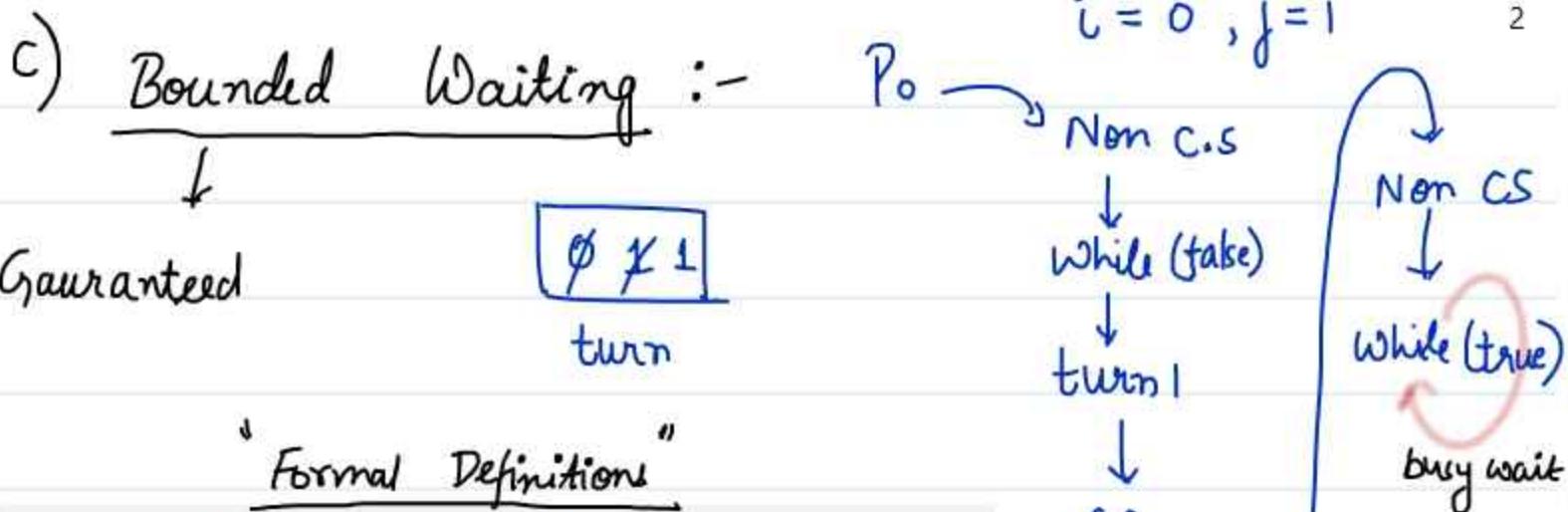
Mut Ex

$P_i \quad P_j$

$P_j$

Not interested for C.S.

$\rightarrow$  will get stuck in while loop as  $P_j$  will never enter to C.S. so turn = i will never happen-



- Mutual Exclusion:** This property ensures that if a process is executing in its critical section, then no other process can execute in their critical section.
- Progress:** This property ensures that if no process is executing in its critical section and there exist some processes that wish to enter their critical section, then the selection of the processes that will enter the critical section next cannot be postponed indefinitely.
- Bounded Waiting:** This property ensures that there exists a bound or limit on the number of times that other processes are allowed to enter their critical sections after a process has made a request to enter its critical section and before that request is granted.

## # DEKKERS ALGORITHM (CORRECT ALGO)

```
void Dekkers_Algorithm(int i)
{
    int j = ! (i);
    while (1)
    {
        (a) Non_CS();
        (b) flag[i] = TRUE;
        (c) while (flag[j] == TRUE)
        {
            if (turn == j)
            {
                flag[i] = FALSE;
                while (turn == j);
                flag[i] = TRUE;
            }
        }
        (d) <CS>
        (e) flag[i] = FALSE;
        turn = j;
    }
}
```

Set coz you are interested

other process turn? → yes

If it's other process' turn.

set yourself as not interested

busy wait coz it's not your turn

Now when you got your turn set yourself as interested

after you've completed C.S. set yourself as uninterested & give turn to other process

comes out when  
 $j = F$

# Q Why Dekkers Algorithm is correct ?

- Provides** {
- Mutual Exclusion:** The algorithm uses two boolean variables, `flag[i]` and `flag[j]`, to indicate if a process wants to enter the critical section. It also uses a variable `turn` to decide which process can enter the critical section when both want to. This ensures that only one process can enter the critical section at a time, providing mutual exclusion.
  - Progress:** If a process doesn't want to enter the critical section, it sets its flag as false. This allows the other process to enter the critical section if it wants to. This ensures that if a process wants to enter the critical section, it eventually will, providing progress.
  - Bounded Waiting:** The `turn` variable ensures that the processes alternate in entering the critical section when both want to enter. This prevents starvation and ensures bounded waiting.

Q Two processes,  $P_1$  and  $P_2$ , need to access a critical section of code. Consider the following synchronization construct used by the processes:

```

/*P1*/           /*P2*/
while (true)      while (true)
{
    wants1 = true;   wants2 = true;
    while (wants2 == true);  while (wants1 == true);
    /* Critical Section */ /* Critical Section */
    wants2 = false;   wants1 = false;
}
/* Remainder section */ /* Remainder section */

```

Here, `wants1` and `wants2` are shared variables/ which are initialized to false.

Mutex ?

Progress ?

Bounded waiting ?

Deadlock ?

↓  
Yes

↓  
Yes

↓  
No

↓  
Yes

No Strict alteration → They never've DL.

Q  
var P = F

var Q = F

**Process X**

```
/* other code for process X */
while (true)
```

varP = true;

while (varQ == true)

{

/\* critical section \*/

varP = false;

}

}

/\* other code for process X \*/

H.W.

**Process Y**

```
/* other code for process Y */
while (true)
```

varQ = true;

while (varP == true)

{

/\* critical section \*/

varQ = false;

}

}

/\* other code for process Y \*/

• Mutex ?

• Deadlock ?

## # Synchronization Hardware #

→ Each Processor supports some special H/w Inst. that are atomic in nature.

Instruction



T.S.L

SWAP

- Busy waiting
- can be used @ UM as Special Instructions
- H/w category
- Multiprocess solution
- Lock Based solution

# T.S.L. #

→ Test & Set Lock

TSL(&lock); < Process executing TSL, returns the current value of lock & sets the value of lock always to true >

```

Bool TSL ( Bool * target )
{
    Bool returnValue;
    returnValue = * target;
    * target = TRUE;
    return (returnValue);
}

```

will execute  
Atomically  
(No Preemption)

return the value of Lock & set it to TRUE.

→ done atomically.

Process will execute T.S.L. by passing & lock.

# Solving C.S. problem using T.S.L. #

Bool lock = F ;

void Process ( int i )

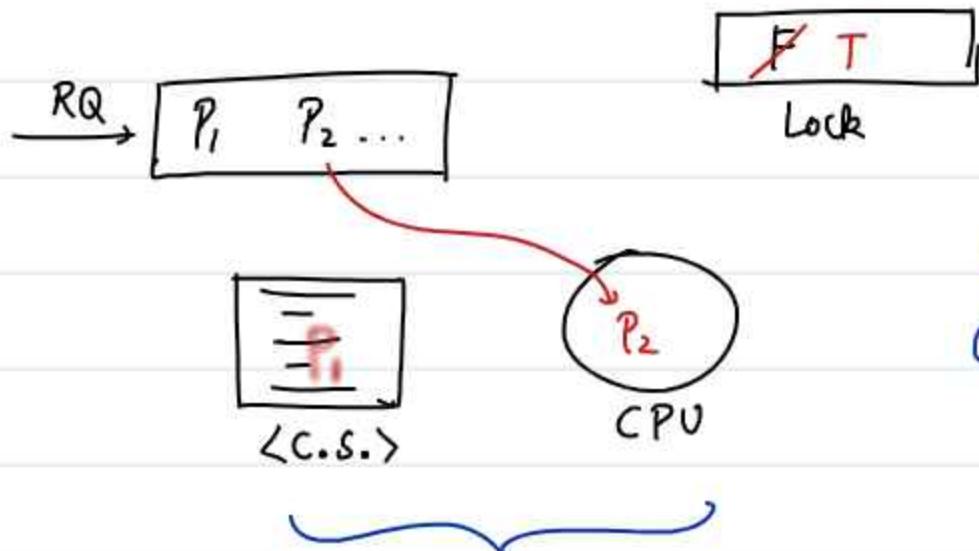
{      while (1)

( a ). Non C.S.

b).  $\text{while } (\text{TSL}(\& \text{lock}) == T);$   $\rightarrow$  Entry section

c).  $\langle C.S. \rangle$

} d). Lock = F ;  $\rightarrow$  Exit section



upgradation of Lock variable. will never change the value of Lock=T.

In Galvin, they have added additional logic in TSL which will make it like Round Robin

Bounded wait X

$P_1$  can go into  $\langle C.S. \rangle$  again & again

## # SWAP Mechanism #

→ Lock & Key Mechanism  
→ Atomic

```

SWPA (Bool * a , Bool * b)
{
    Bool t ;
    1. t = *a ;           3. *b = t; }
    2. *a = *b ;

```

User Process :-       $\text{Bool lock} = F;$

Void Process ( int i )

{       $\text{Bool key} = T;$

while(1) {

a) Non C.S.();

b) while ( $\text{key} == T$ )

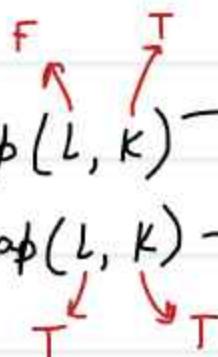
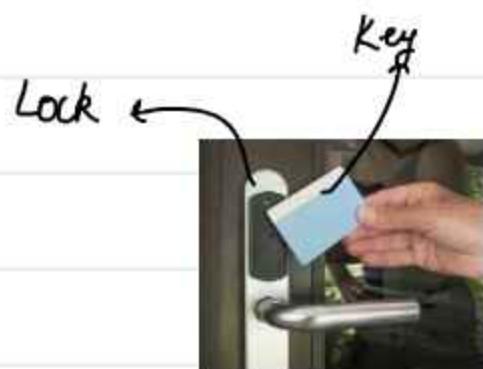
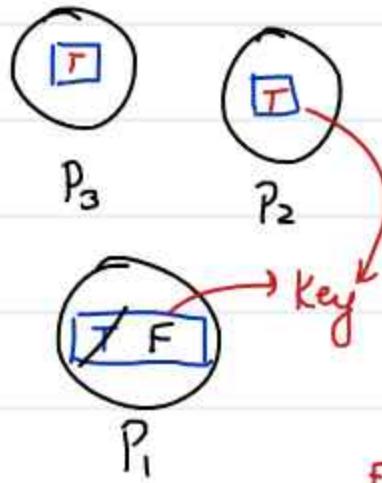
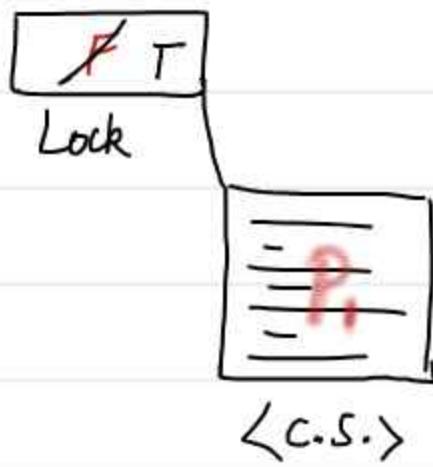
{ SWAP(&lock, &key); }

c) <C.S.>

d)  $\text{lock} = F;$

}

}



$P_1$  : Non C.S., ...  $\text{key} = T \rightarrow \text{swap}(L, K) \rightarrow L = T \xrightarrow[K=F]{} C.S. \rightarrow P_{\text{ex}}$

$P_2$  : Non C.S. ...  $\text{key} = T \rightarrow \text{swap}(L, K) \rightarrow \text{Busy wait}$

Guarantees Mutual Exclusion

Guarantees Progress as no uninterested process will change the value of lock , not blocking the other processes.

In particular, the code could lead to a situation known as **starvation**. If one process quickly finishes its critical section and re-enters it before the other process has a chance to set the lock , it could potentially keep re-entering the critical section indefinitely, while the other process waits unboundedly. This is a limitation of this basic swap-based mutual exclusion algorithm.

Unbounded waiting , if added additional logic then B.W. ✓

Q

The enter\_CS () and leave\_CS () functions to implement critical section of a process are realized using test-and-set instruction as follows:

```
void enter_CS(X)
{
    while (test-and-set(X));
}
void leave_CS(X)
{
    X=0;
}
```

$X = 0 \rightarrow \text{C.S. is free}$

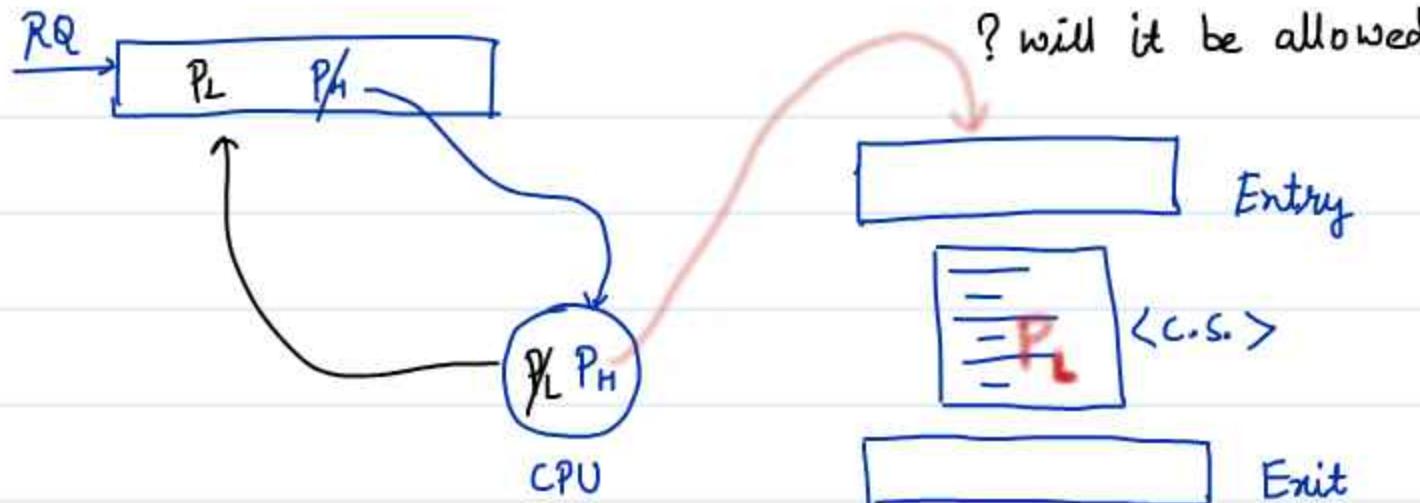
Return the value of  $X$  &  
Set it to  $1 \rightarrow \text{C.S. is busy.}$

In the above solution, X is a memory location associated with the CS and is initialized to 0. Now consider the following statements:

- I. The above solution to CS problem is deadlock-free. ✓
- II. The solution is starvation free. X
- III. The processes enter CS in FIFO order. X → can enter in any order
- IV. More than one process can enter CS at the same time. X

## # Priority Inversion Problem #

- All correct Busy wait solution suffers from Priority Inversion Principle.
- Preemptive Priority Based scheduling



If  $P_H$  is also interested in  $\langle C.S. \rangle$  then?



$P_H$  can't be allowed to enter C.S. until  $P_L$  completes & for completion  $P_L$  needs the CPU.



$P_H$  won't leave CPU because of its ego.



Conflict of Two Principles



Priority

Mutual Exclusion

Both will be adamant.  $\Rightarrow$  DEADLOCK.

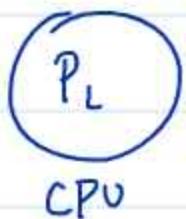
If someone in interview ask you an example of Deadlock, this is the best one. This problem occurred in NASAs "MARS PATH

↓  
sol"

## Priority Inheritance

$P_L$  will inherit the priority of  $P_H$  & vice versa for  $P_H$ .

$$P_L \leftrightarrow P_H$$



Exit sect

 → gets its own priority back
 

H.W.

Fetch\_And\_Add ( $X, i$ ) is an atomic Read-Modify-write instruction that reads the value of memory location  $X$ , increments it by the value  $i$  and returns the old value of  $X$ . It is used in the pseudocode shown below to implement a busy-wait lock.  $L$  is an unsigned integer shared variable initialized to 0. The value of 0 corresponds to lock being available, while any non-zero value corresponds to the lock being not available.

```

AcquireLock(L)
{
  While (Fetch_And_Add (L,1))
    [ L=1; ] → Entry section
    ReleaseLock(L)
    { L=0; } ] Exit section
}
  
```

will  
busy  
wait  
if  $L > 0$

If  $L = 0$  ] → CS is free  
 $L > 0$  ] Not free }

will oscillate b/w 1 & 2.

```

int Fetch-And-Lock (int x, int i)
{
  int rv;
  rv = x;
  x = x + i;
  return (rv);
}
  
```

int lock = 0

while (1) {

while ( Fetch - And - Lock (& lock, 1)

{ L = 1; }

< C.S. >

L = 0;

}

→ Atomic : Returning the value

of lock & inc. by  
 $\perp$  will be done  
 atomically.

→ Entry

→ Exit

If this wouldn't have been there the value overflow would have occurred.



a

→ How?

b

Does it provide Mut Ex?

c

Can it be possible after a moment that no one can go to < C.S. >

d

can it be possible that Lock  $\perp$  & < C.S. > is free

e

starvation for some process?



If you do this correctly, go give yourself a nice treat.

# PROCESS SYNC PART - 5

1

let's Revise

1. Lock variable	ME X	Progress ✓	BW X
2. Strict Alternation	ME ✓	Progress X	BW ✓
3. Peterson & Dekker	ME ✓	Progress ✓	BW ✓
4. TSL	ME ✓	Progress ✓	BW X
5. SWAP	ME ✓	Progress ✓	BW X

H.W.

Fetch\_And\_Add ( $X, i$ ) is an atomic Read-Modify-write instruction that reads the value of memory location  $X$ , increments it by the value  $i$  and returns the old value of  $X$ . It is used in the pseudocode shown below to implement a busy-wait lock.  $L$  is an unsigned integer shared variable initialized to 0. The value of 0 corresponds to lock being available, while any non-zero value corresponds to the lock being not available.

```

AcquireLock(L)
{
    While (Fetch_And_Add (L,1))
        {
            L=1;
        } → Entry section
    ReleaseLock(L)
    {
        L=0;
    } Exit section
}

```

will  
busy  
wait  
if  $L > 0$

If  $L = 0 \rightarrow$  cs is free  
 $L > 0 \rightarrow$  Not free

will oscillate b/w 1 & 2.

```

int Fetch - And - Lock ( int x , int i )
{
    int rv ;
    rv = x ;
    x = x + i ;
    return ( rv ) ;
}

```

int lock = 0

while (1) {

    while ( Fetch - And - Lock (& lock, 1) )

    { L = 1; }

    <C.S.>

    L = 0;

}

→ Atomic : Returning the value

of lock & inc. by 1 will be done

atomically.

    Entry

    Exit

If this wouldn't have been there the value overflow would have occurred.



a

→ How?

b

Does it provide Mut Ex?

c Can it be possible after a moment that no one can go to <C.S.>

d can it be possible that Lock ⊥ & <C.S.> is free

e starvation for some process ?

RQ

P<sub>1</sub> P<sub>2</sub> P<sub>3</sub>

RQ

CS

P<sub>2</sub>  
∅ X Z1

Lock

P<sub>1</sub>

$t_1: P_1 \rightarrow \text{FAA} : \xrightarrow{\text{Lock} = 1} 0, \text{C.S.}, \text{Pre}$   
 (current Lock value)

$t_2: P_2 \rightarrow \text{FAA} : \xrightarrow{\text{Lock} = 1} L, \text{Lock} = 1$   
 $\xrightarrow{\text{Lock} = 2} \text{busy wait}$

$t_3: P_1 \rightarrow \text{C.S.}, L = 0$

$t_4: P_2 \rightarrow \text{FAA} : \xrightarrow{\text{Lock} = 1} 0, \text{C.S.}, L = 0$



Overflow not possible + Guarantees Mut.Ex.

(c)  $t_1: P_1 \rightarrow \text{FAA} : 0, \langle \text{C.S.} \rangle, \text{Pre}$

$\xrightarrow{\text{Lock} = 1}$  when it'll come

$t_2: P_2 \rightarrow \text{FAA} : L, \text{Pre}$

$\xrightarrow{\text{Lock} = 2}$

{ }

while ( Fetch-And-Add (&lock, 1) )

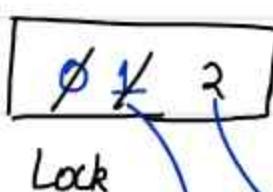
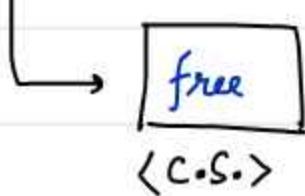
General obs :-

Whenever there are { lock = 1 ; }  
 two stmts in Preemption occurred here

Entry section, preempt  
 after one to get good inferences.

$t_3 : P_1 \rightarrow <\text{C.S.}>$ ,  $L = 0$ , leaves

$t_4 : P_2 \rightarrow L = 1$ , FAA:  $L$  busy wait



$\} P_2$  should logically go into  $<\text{C.S.}>$

as  $P_2$   
starts

By FAA

$t_5 : P_3 \rightarrow$  will always busy wait because  $lock \neq 0$  afterwards.



Have you noticed something?



There will be so many processes which can be preempted after increasing the value of  $L$



OVERFLOW MAY HAPPEN

$int = 2B$



If no of processes are less than

32767 then this mechanism will

Max value = 32767

work correctly. If

no of Process > size of Int

NOTE :- Unless a special focus is given to exceptional case, think Generally!

Ex:- Which of the following Algo suffers from starvation?

Galvin  
Textbook  
Exercise Q

- a) FCFS ? → It inherently doesn't
- b) RR X → Never suffer from
- c) SJF ✓ starvation except
- d) Priority ✓ the case for :-

Infinite Loops

Ans :- (c, d) only.

## # BLOCKING MECHANISMS / NON BUSY #



<c.s.>

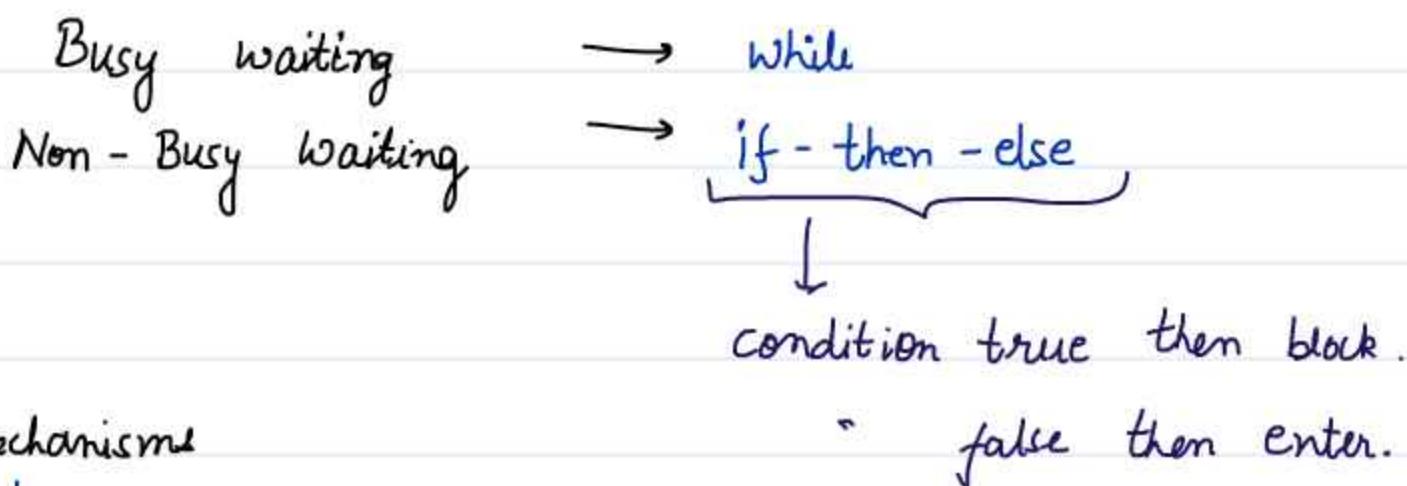
In just one check it can get the idea whether c.s. is free or not, why it has to busy

Avoid wastage of CPU time in form of Loops.

{ wait until the time quantum expires.

wasting its + others time

Process should test the condition only once, if free then enter <C.S.>, otherwise it should immediately release the CPU and should get BLOCKED, so that the process present in <C.S.> should take charge of CPU, quickly execute <C.S.>, come out, execute exit section & unblock the blocked process.



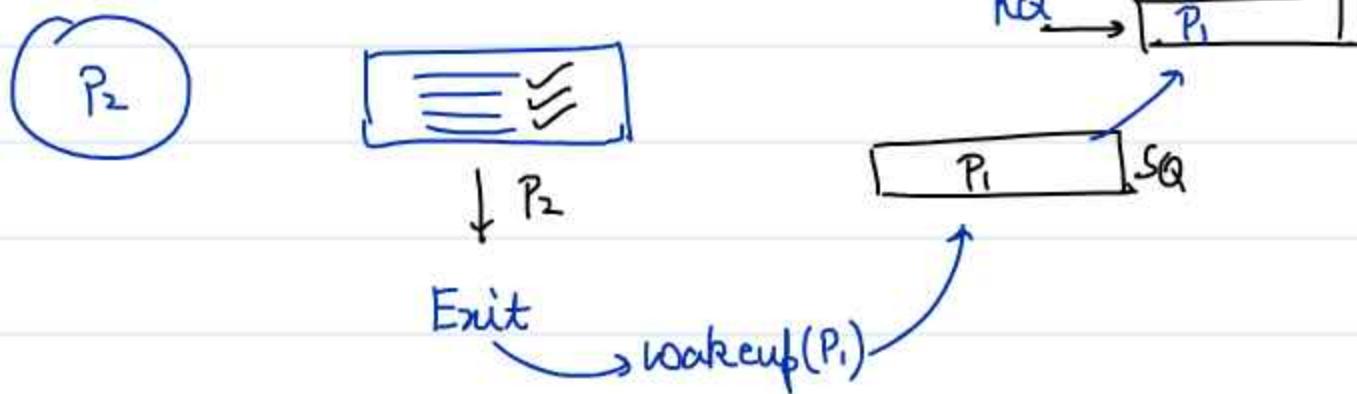
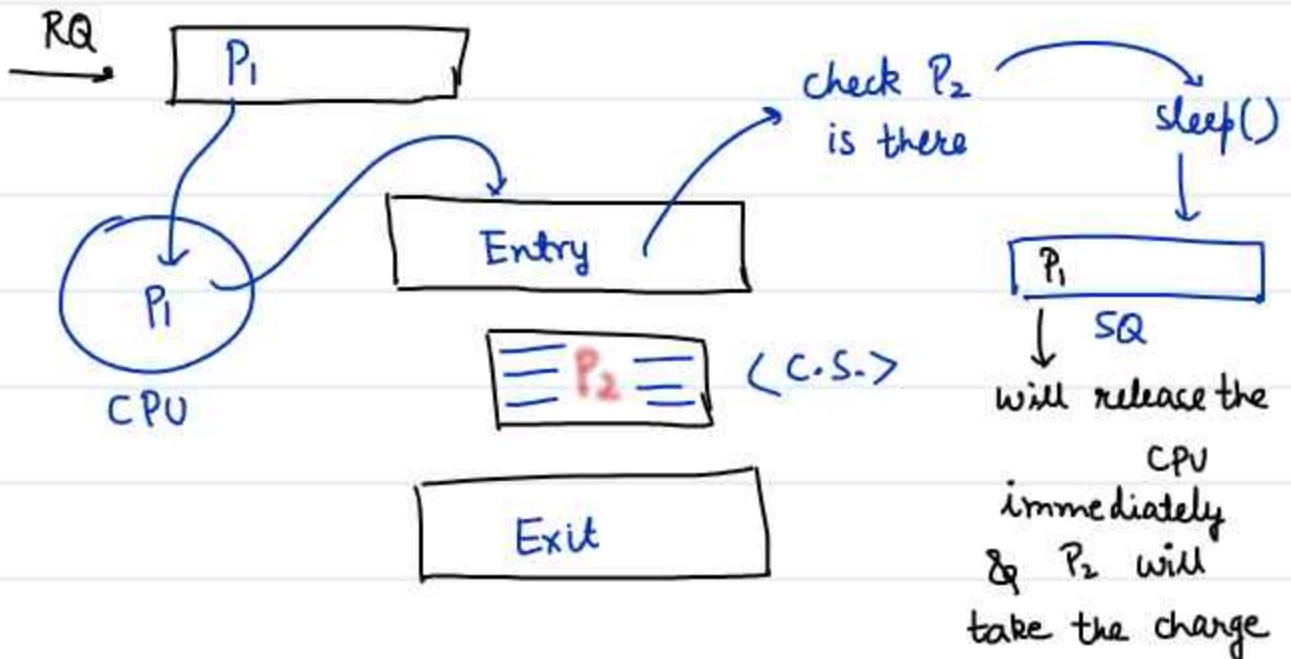
## # Mechanisms

- Sleep & Wakeup
- Semaphores
- Monitors

## # SLEEP & WAKE UP #

- Blocking construct
- Multiprocess sol<sup>n</sup>
- OS Primitives (behaves like system call)

When Process executes `sleep()` → gets Blocked till some other Process wakes her up. (sleep Queue)



## PRODUCER CONSUMER - METHOD II

We are going to convert BW → Non BW  
(Busy waiting)

- \* Make Consumer sleep when buffer is empty.

- Make Producer sleep when buffer is full.
- Wake up Consumer when first item is placed  
(before that it might be sleeping)
- Wake up Producer when an item is consumed from the full buffer, before that it might be sleeping.

```
# define N 100
int buffer [N];
int count = 0;
```

}

global

```
Void Producer (void)
{
    int itemp , in=0 ;
    while (1)
    {
        a). itemp = Produce-item ();
        b). if (count == N) {sleep();}
        c). Buffer [in] = itemp ;
        d). in = (in + 1) % N ;
        e). count++ ;
        f). if (count == 1) {wakeup (consumer)} ;
    }
}
```

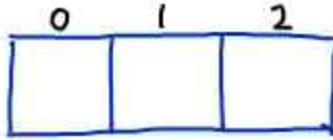
```

void Consumer(void)
{
    int itemc, out = 0;
    while(1) {
        if(count == 0){ sleep(); }
        itemc = Buffer[out];
        out = (out + 1) % N;
        count--;
        if (count == N-1) { wakeup(Producer); }
        Consume-item(itemc); }
    }
}

```

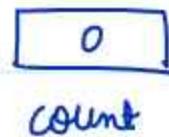
### Lets Analyze

Q-1 As there is cooperation & competition both is there any possibility of  $\rightarrow$  Deadlock or  $\rightarrow$  Inconsistency?



$N=3$

Buffer[3]



count

$\rightarrow$  DEADLY!!!

Preemption

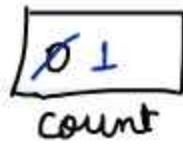
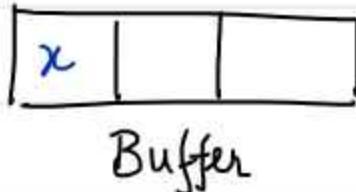
$\text{if } (\text{count} == 0)$

{ sleep(); }



$t_1: C \rightarrow$  if (count == 0)  $\xrightarrow{\text{Preempt}}$  will go to sleep when comes back

$t_2: P \rightarrow$  count  $\neq N \rightarrow$  Produce 1<sup>st</sup> Item  $\rightarrow$  place it

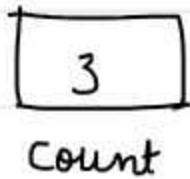
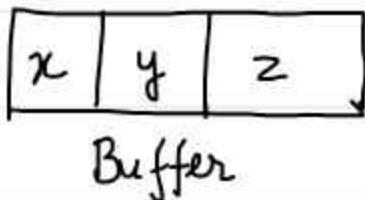


Increment count

(But consumer never slept)  $\downarrow$  wake up consumer  $\leftarrow$  count == 1  $\xrightarrow{\text{yes}}$

P will think it has done its work

goes back & repeat this until buffer is full



if (count == N)  
 $\downarrow$  yes

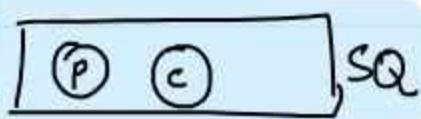
thinking that C

will wake her up as soon as

she consumes an item but P

doesn't know C is going to sleep too.

goes to sleep

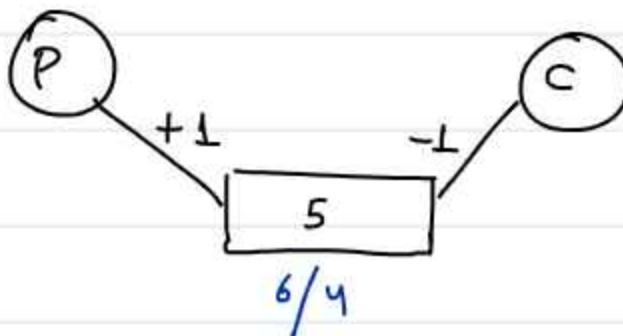


Both processes are sleeping comfortably with assumption that one will wakeup the other.



## DEADLOCK

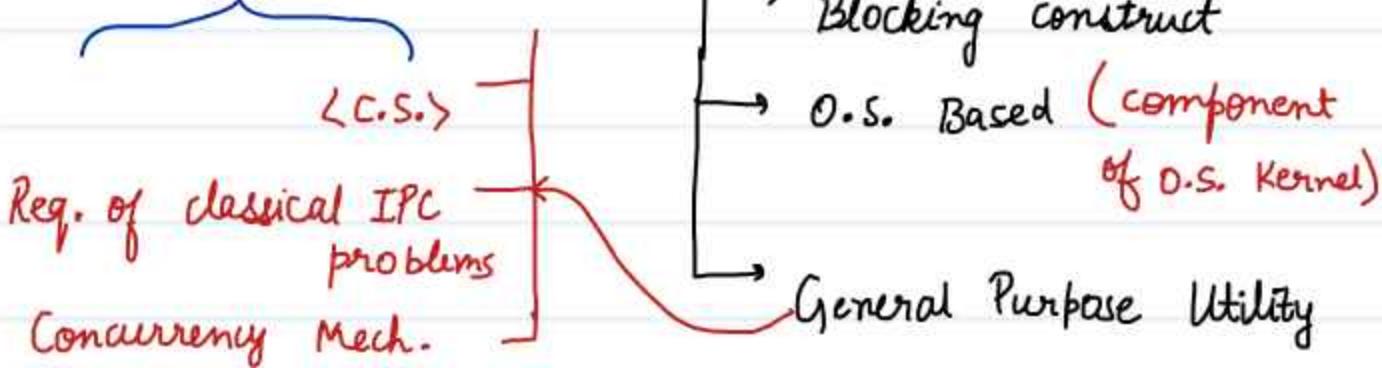
And INCONSISTENCY is also there when P & C tries to update count simultaneously



SOLUTION

SEMAPHORES} Practically in-use tool

Used for several prob.



Semaphore is implemented as an A.D.T. like process.

- have its own implementation
- have its own operations (Atomic)

Down  
 or (Wait)  
 or P()

Up  
 (Signal)  
 v()

Definition :- Variable <ADT : Sem> that takes only Integer values

[BSEM] Binary (MUTEX)

Values: <0,1>

Counting (GENERAL)

<-∞ to +∞>

[CSEM]

↓  
Repr.

# PROCESS    SYNC - 6

# Counting Semaphores (General)     $\langle -\infty \text{ to } +\infty \rangle$

KM {

```
Typedef struct {
    int values ;
    QueueType L ;
} CSEM;
```

Contains the list of PCB's of those processes that gets blocked while performing 'DOWN' operation unsuccessfully.

UM {

```
CSEM s; } after defining the var, we have to
mandatorily initialize it.
↓
initial value?
↓
will depend on application
< si >
```

KM {

```
    DOWN (CSEM s)
    1. s.value = s.value - 1;
    ↓
    we say that result
    is either successful / unsuccessful
    next stmt ✓
    Blocked in Queue ✓
```

2. if ( $s.value < 0$ )

How to identify?

{ Put the process  $\xrightarrow{\text{Queue name}}$  PCB in Queue ( $s.L$ ) &  
Block the Process (sleep)

}

else

{ return; }

After Decrementation if the resultant value of semaphore is negative.  $\Rightarrow$  UNSUCCESSFUL.

If R.V.  $> 0 \Rightarrow$  SUCCESSFUL

$\xrightarrow{\text{to next statement } \langle s_i \rangle}$

Q If  $s = 5$ , how many down ops can

be carried out successfully?

Ans:-

$$s = \cancel{s} / \cancel{y} \cancel{x} \cancel{x} + \cancel{d} - \cancel{x} - 2$$

no of blocked process = 2

$$\downarrow, \downarrow, \downarrow, \downarrow, \downarrow, \downarrow$$

Value became -ve  
hence unsuccessful

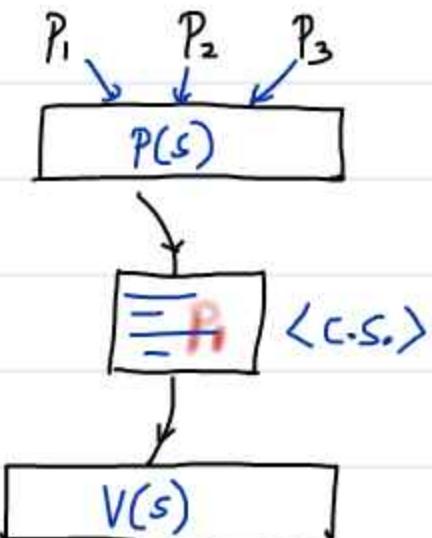
5 down ops

There are

two process in L.

- +ve value of Semaphore indicates that those many down operation can be performed successfully.

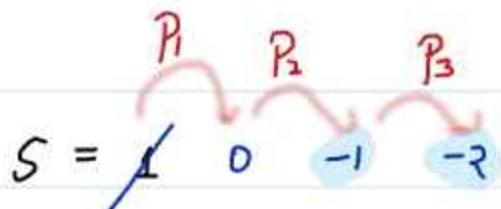
- After carrying out a series of Down operation if the value of semaphore becomes -ve then the magnitude of -ve value indicate the no of blocked process.



$s = \boxed{2}$   $\rightarrow$   $x$  down operations can be performed successfully.

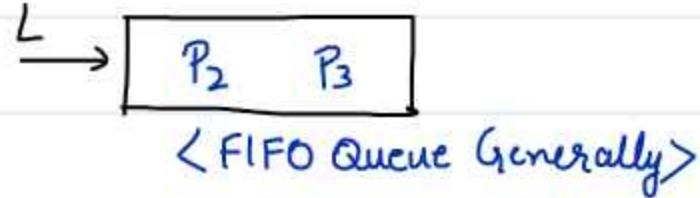
$x = 0$  then

$x = \emptyset \rightarrow \boxed{1} - \boxed{-1} - \boxed{-3}$  ] all three processes will get blocked.



$x = \emptyset \neq \emptyset \neq 0$  ] all three processes

will be successful in performing DOWN operation, they all will enter C.S.



The basic objective of UP operation is to wake up a sleeping process.

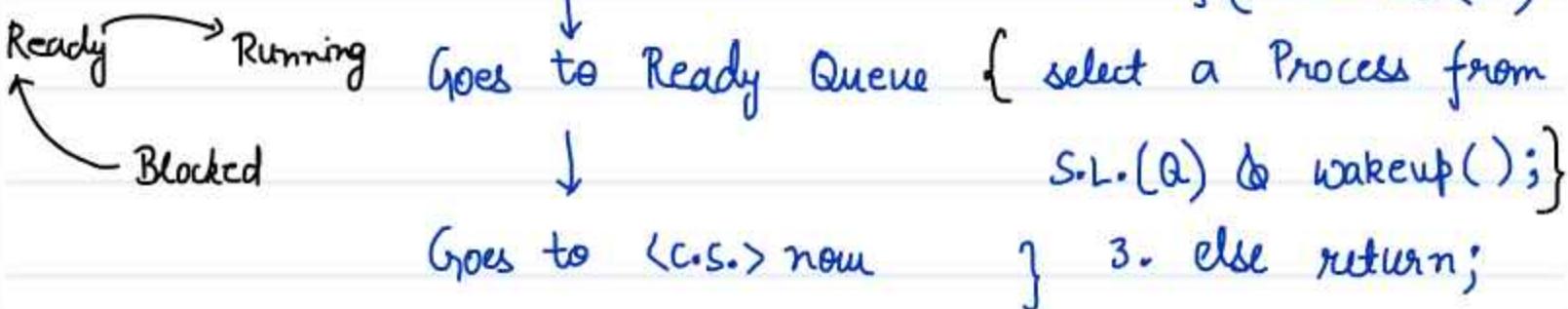
Now you tell what should be the initial value.

$s = \emptyset - \boxed{1} \rightarrow P_3$  is sleeping

$P_2$  is awake now

UP( CSEM s ) KM

- { 1. s.value ++;
- 2. if ( s.value < 0 )



(as it has already completed its Entry section)

4

So  $P_1$  when performing up operation indirectly waking up  $P_2$  & putting it into  $\langle C.S. \rangle$



$P_2$  Perform  $V(S)$  waking up  $P_3$

↓

$P_3$  goes to RQ then C.S then  $V(S)$

No wakeup

∅ +

call if  $S = 0 \rightarrow$  have to wake up one more  $S$  process.

$S$ .value > 0.

if the initial value of  $S = 1$ .

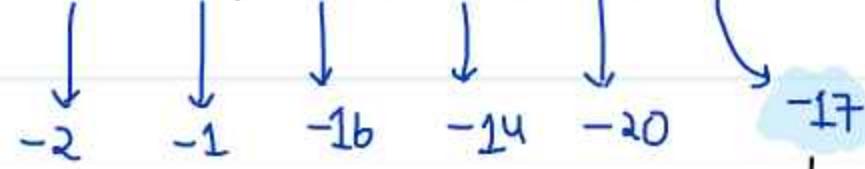


Mutual exclusion is violated  $S > 1$

Q

CSEM  $S = 8$ ;

Operation :- 10P, 1V, 15P, 2V, 6P, 3V



L →  $P_1 P_2 \dots P_{17}$

Q CSEM  $S = ?$

Ops :- 12P, 4V, 6P, 3V, 8P, 1V

Final value = -6

$$\cancel{-6} + \cancel{12} - \cancel{4} + \cancel{6} - \cancel{3} + \cancel{8} - \cancel{1} = \underline{\underline{12}} \text{ Ans}$$

you just took  $x$  & solved it the old way.

Q

Consider a non-negative counting semaphore  $S$ . The operation  $P(S)$  decrements  $S$ , and  $V(S)$  increments  $S$ . During an execution, 20  $P(S)$  operations and 12  $V(S)$  operations are issued in some order. The largest initial value of  $S$  for which at least one  $P(S)$  operation will remain blocked is \_\_\_\_\_.

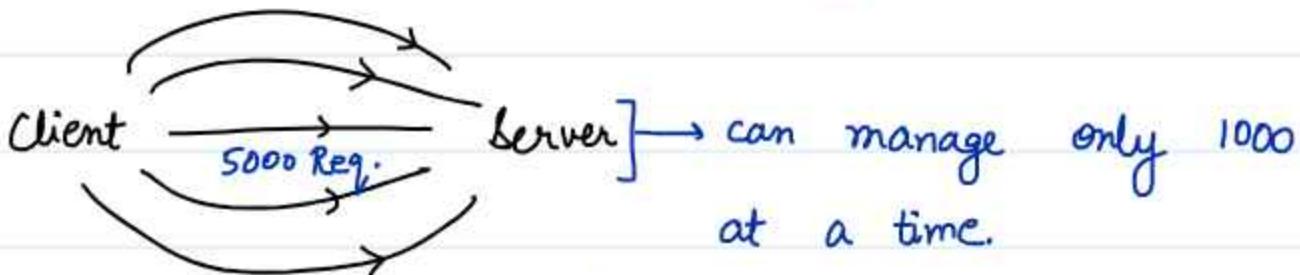
at least one  $P(s)$  will remain blocked that is in the end

$$S \leq -1$$

$$x - 20 + 12 = S \leq -1$$

$$x - 8 \leq -1$$

$$x \leq 7 \quad \text{largest value} = 7$$



problem can be solved using counting sem.  
which ensure only 1000 go & rest wait.

$$S = 1000$$

Every Req. will perform down on  $S$   
& after it gets served it'll perform up then.

# # BINARY SEMAPHORES <0,1> #

Typedef struct {

enum value (0,1); } ]

QueueType L;

} BSEM;

same like that in CSEM.

This variable value of type enum can take only 2 values 0 & 1.



only two possibilities

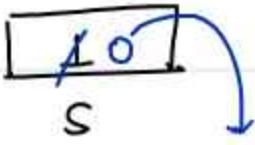
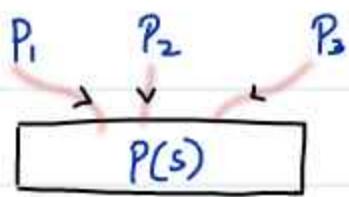
UM

BSEM s;

s = 1;

DOWN(s);

< Next Stmt >



$\langle C.S. \rangle$

- will never take

-ve values  $\Rightarrow$  we  
won't know how

many block processes  
are there.

DOWN (BSEM s)

{ if (s.value == 1)

{ s.value = 0;

return;

}

else { put this process  
(PCB) in S.L.(Q) &

}, Block-it (sleep()); }

V(s)

If there is process in either critical section / block Q then value of S must be 0.

How UP will be performed?

- ↳ if There is no block process then  $S.value = 1$
- ↳ if There are processes in block Q then wakeup a process & send it into <c.s.>

```

UP (BSEM s)           → Block Queue L
{ if (S.L is not Empty) {
    Select a process from L & wakeup();
}
else s.value = 1; }   S = 0 → UP
                        ↗ 0 (Q ≠ empty)
                        ↗ 1 (Q = empty)
} Always success
  
```

\*  $S = 1$   $\Rightarrow$  <c.s.> is free & Q is empty.  
 $\xrightarrow{\text{UP}}$   $s = 1$  (status: success)

$S = 0$   $\Rightarrow$  A process is either in c.s. or Block Q  
Initially  $\xrightarrow{\text{UP}}$   $s = ?$  (status: success)  
First operation

If it's the first of then it logically means Q is empty.

$$S = 1 \quad \checkmark$$

Process never gets blocked while performing UP operation, can get blocked while DOWN, whether it's counting / binary semaphore.

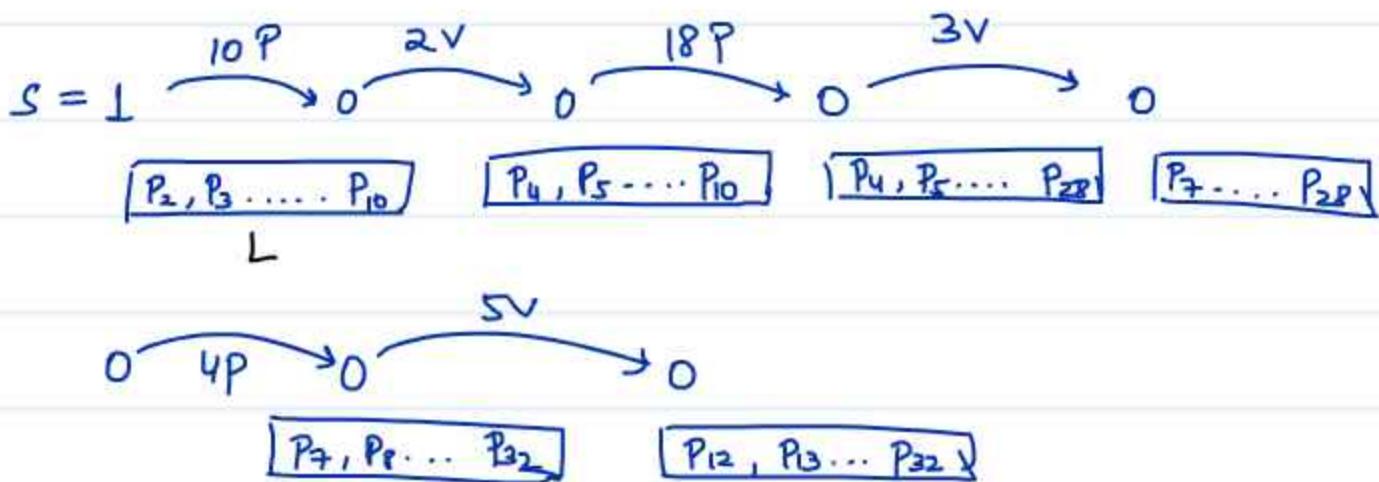
NOTE :- This sleeping Q which we are talking about is dynamic.

Q

BSEM,  $S = 1$ ;

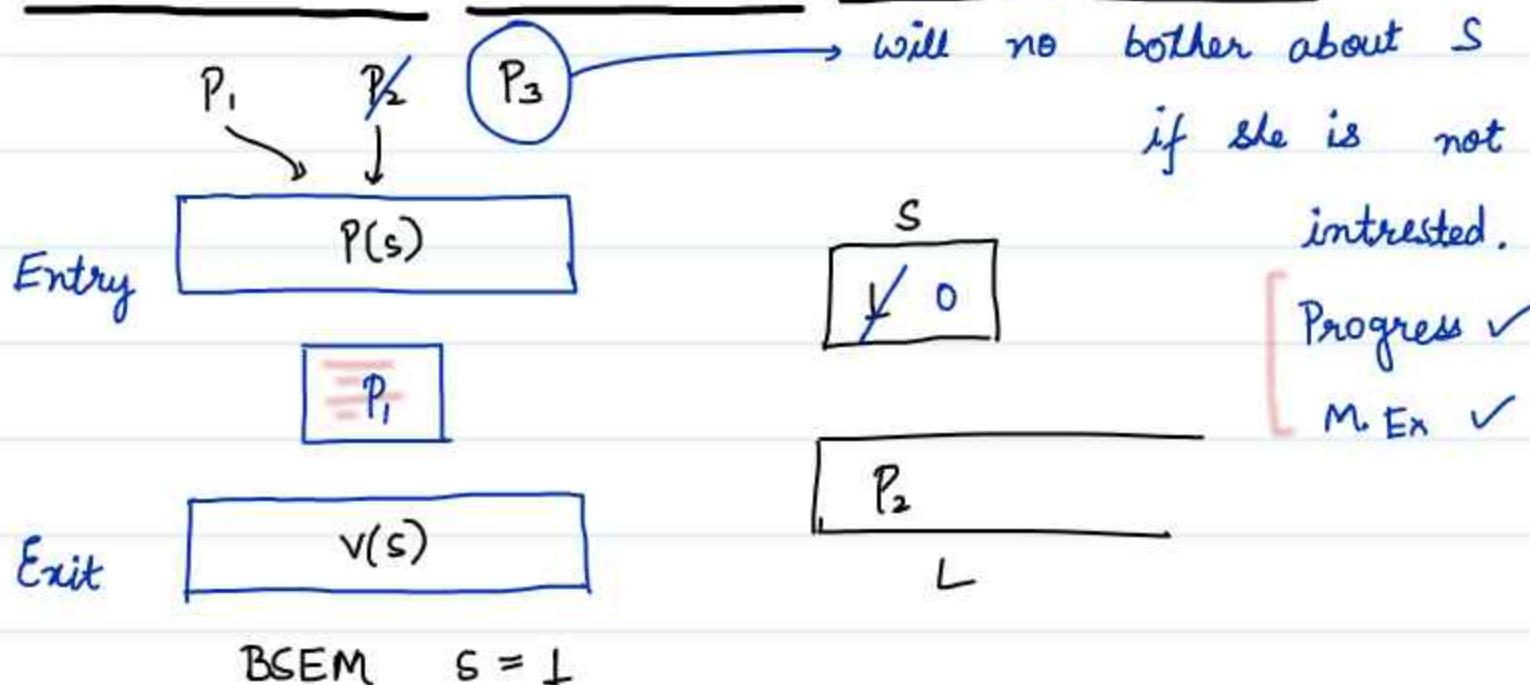
ops :- 10P; 2V; 18P; 3V; 4P; 5V;

- a)  $S = ?$       b) Size of L

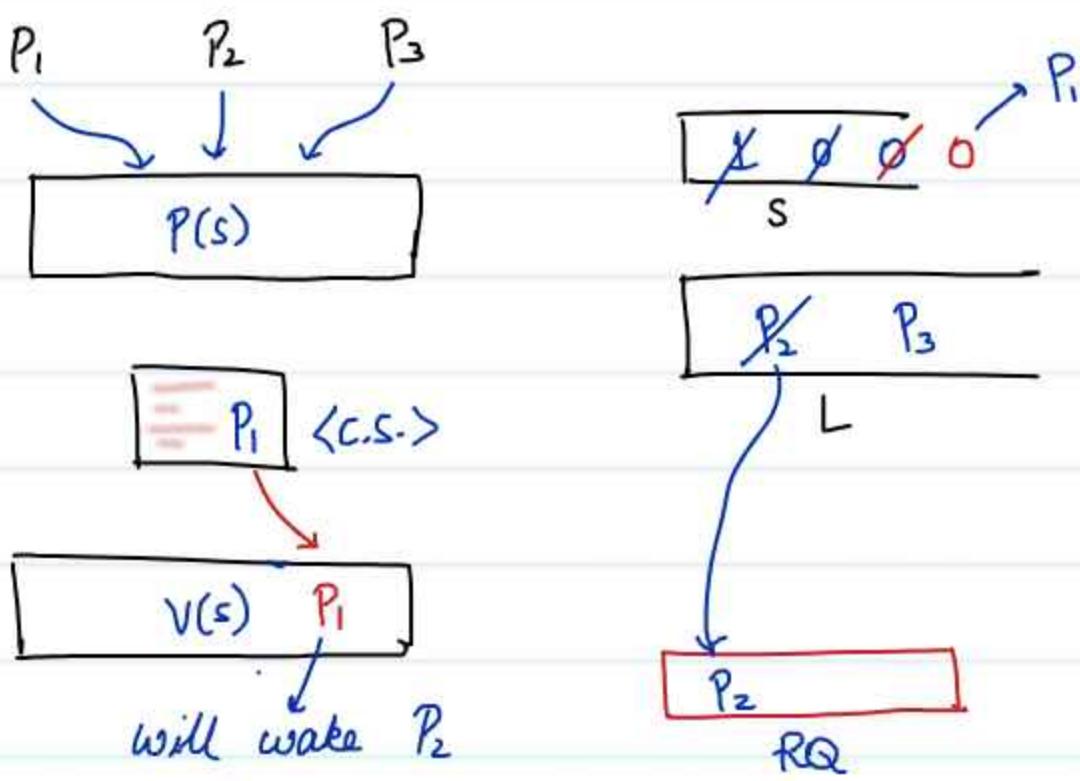


Size = 21       $S = 0$   
 Ans

# PROCESS SYNC PART - 7

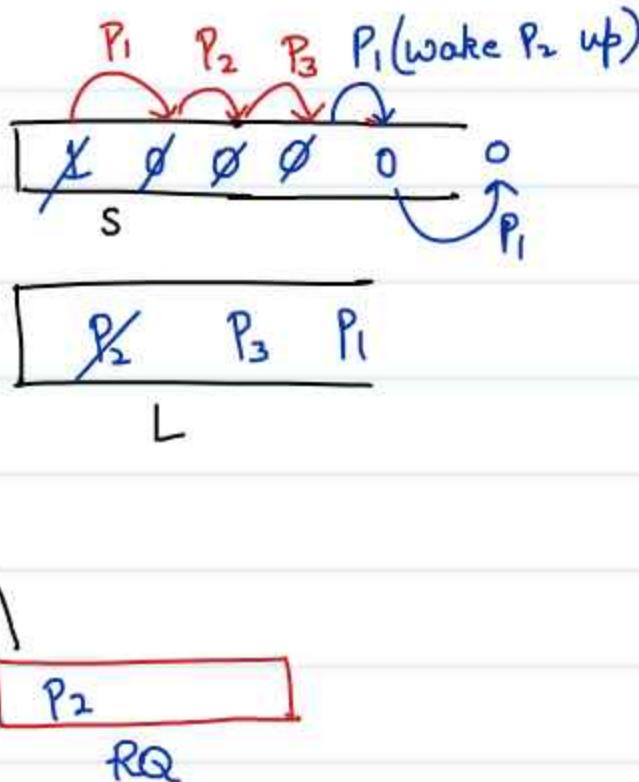
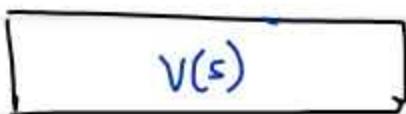
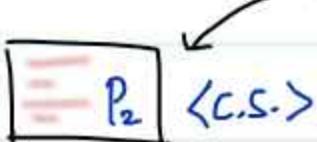
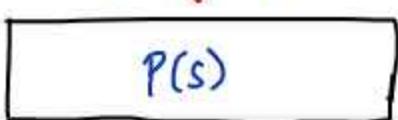


Let's check if P<sub>1</sub> can enter <C.S.> again & again while other interested processes are sleeping

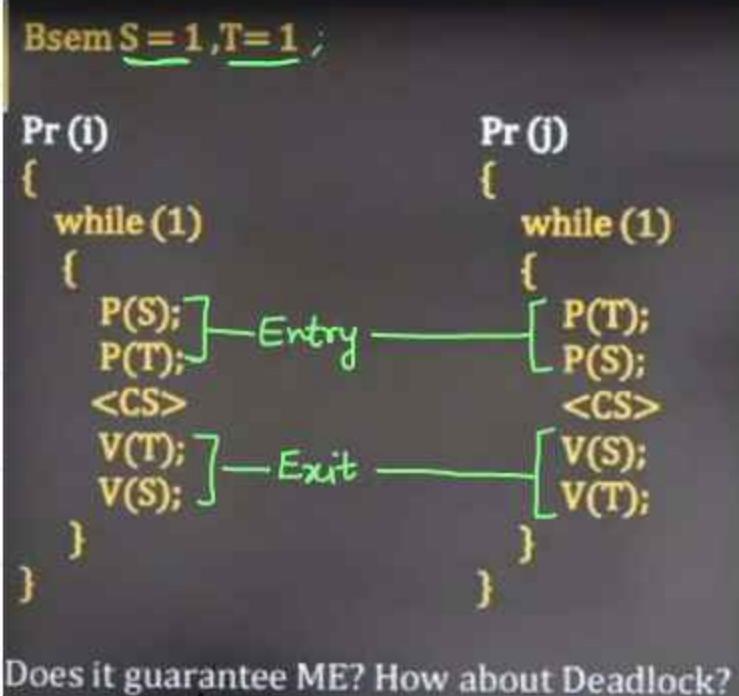


up & try to go to <C.S.> again

tries to go in c.s.  
perform  $P_1$   $P(c)$  but  
 $s=0$  & gets blocked



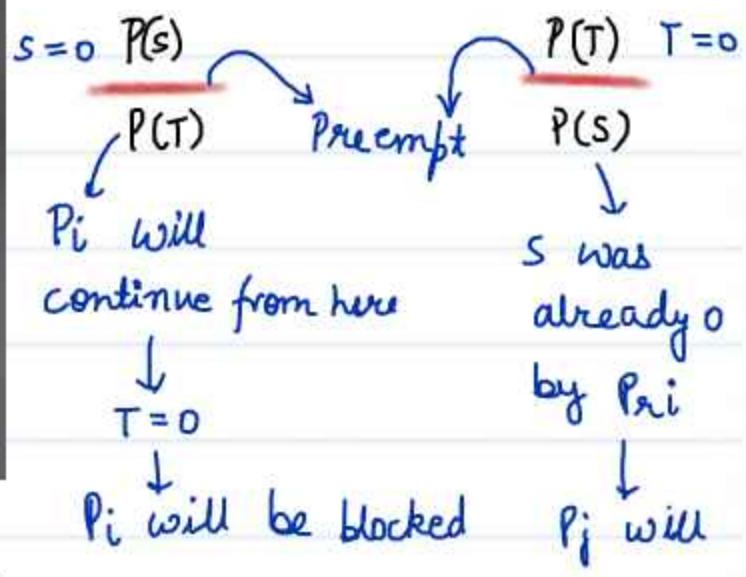
Hence Bounded Waiting is ensured.



can both of them enter <CS> at the same time

Present ✓

Remember the note ?



It's not possible.

Preempt anywhere & check.

Mutual Exclusion ✓

Deadlock X :- Preemption b/w the two down ops.

What change should I make to prevent deadlock.

sequence of both the processes performing down  
should be same. In above question the seq.  
was reverse.

→ doesn't matter for UP ops.

Q

BSem S = 1, T = 0

```

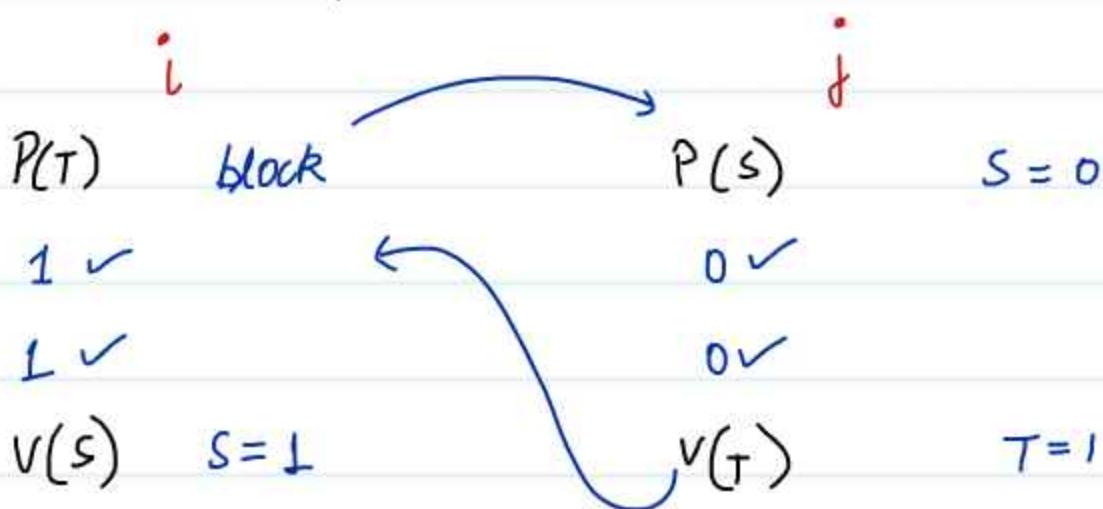
Pr(i)           Pr(j)
{
  while(1)      {
    {
      P(T);      while(1)
      Print('1'); }- P(S); - Entry
      Print('1'); }- Print('0'); }- <c.s.>
      V(S);      Print('0'); }
    }
  }
}
```

Output as  
series

What is the Regular Expression that gets generated?

- b) M/E ?
- c) Progress ?
- d) Bounded wait ?

$$S = 1 \quad T = 0$$



Series :-  $(0011)^+$  → Repeat

ME ✓

Progress X

BW → Implication

of strict alternation

$P_i$  can't enter until

$P_j$  exits from exit section

Q Bsem m[0....4] = {1}; } Array of Binary semaphores

```

Pr(i) i=0,4
{
    while(1)
    {
        P(m[i]);
        P(m[(i+1)%4]);
        <CS>
        V(m[i]);
        V(m[(i+1)%4]);
    }
}
  
```

$P(0)$  →  
 $P(m[0]);$   
 $P(m[1]);$  } stopped  
 $<\text{c.s.}>$   
 $v(m[0]);$   
 $v(m[1]);$

$P(1) \downarrow$   
 $P(m[1]);$  } block  
 $P(m[2]);$   
 $<\text{c.s.}>$   
 $v(m[1]);$   
 $v(m[2]);$

a) What is the maximum no. of processes that can be in <CS>.

b) Deadlock ?

$P(4) \rightarrow P(m[4]);$

$P(m[0]); \quad \} \text{ blocked}$

$P(2) \rightarrow$

$P(m[2]);$

$P(3) \rightarrow$

$P(m[3]); \quad \} \text{ blocked}$

$P(m[3]);$



will pass

$P_0$  &  $P_2$  can be in  $\langle C.S. \rangle$  at the same time.

M/F not guaranteed.

b)

## Deadlock

$$m[0] = \cancel{x} 0$$



C.S.

$$m[1] = \cancel{x} 0$$

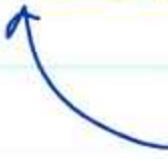
$$m[2] = \cancel{x} 0$$

$$m[3] = \cancel{x} 0$$

$$m[4] = \cancel{x} 0$$

all processes perform their  
first instruction & get preempted

DEADLOCK ✓



will wait on each other for second inst.

BSem S = 1, T = 0, Z = 0

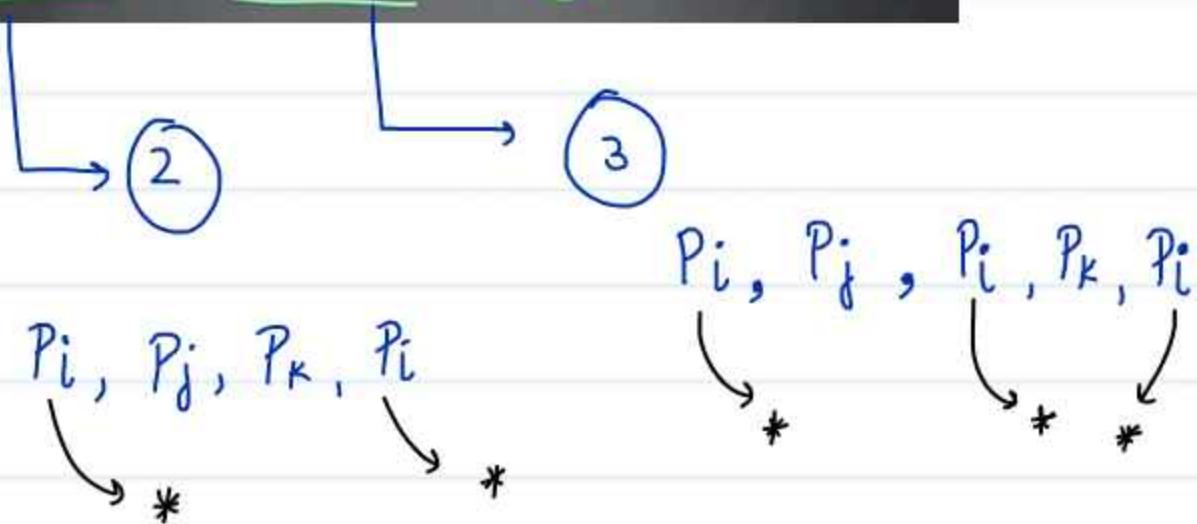
```

Pr (i)           Pr (j)           Pr (k)
{
    While (1)      {
                      P(T); T=0
                      V(S); S=1
    }
    P(S); S=0
    Print (*);
    V(T);
    V(Z);
}
}
}

```

can't  
increase  
more than 1

What is the minimum and maximum no. of "\*" that get printed.



b).  $P_i$       {  
 $\text{while (1)} \downarrow s=0$   
 $\{ P(s) ;$   
 $\text{Print ('*');}$   
 $\text{v(T); } T = \perp \}$   
 $\}$

$S = \perp$

$P_j$       {  
 $P(T) ; \quad T = 0$   
 $v(z); \quad z = 1$   
 $\}$

$T = 0$

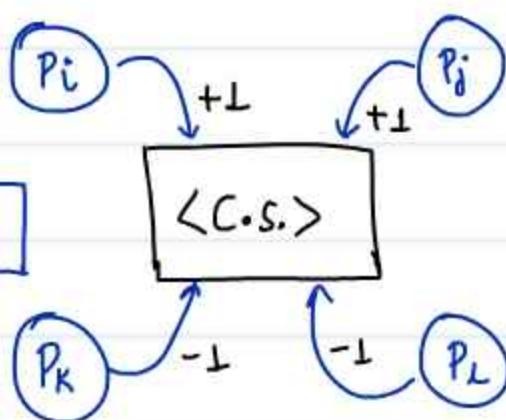
$P_k$       {  
 $P(z) ; \quad z = 0$   
 $v(s); \quad s = \perp$   
 $\downarrow$   
 $P_i$   
 $\downarrow$   
 $s = 0$

$z = 0$

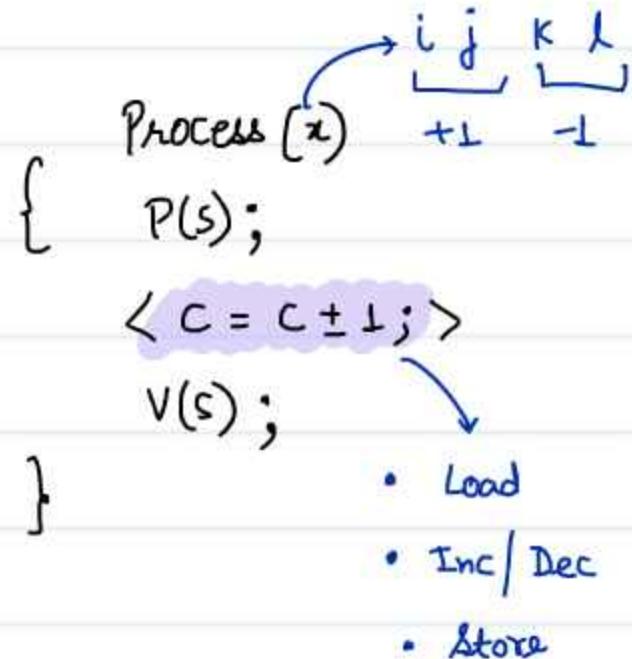
Min = 2 = Max

H.W.

Consider a counting semaphore initialized to 2, there are 4 concurrent processes  $P_i$ ,  $P_j$ ,  $P_k$  &  $P_L$ . The Processes  $P_i$  &  $P_j$  desire to increment the current value of variable C by 1 whereas  $P_k$  &  $P_L$  desire to decrement the current value of C by 1. All Processes perform their update on C under semaphore control. What can be the minimum and maximum value of C after all Processes finish their update.



CSEM S = 2



Process will execute only once.



No While Loop.

Q

Consider Three Processes using four Binary Semaphores a, b, c, d in the order shown below.

Which Sequence is a Deadlock Free sequence?

(I) X: P(a); P(b); P(c);  
Y: P(b); P(c); P(d);  
Z: P(c); P(d); P(a);

(II) X: P(b); P(a); P(c);  
Y: P(b); P(c); P(d);  
Z: P(a); P(c); P(d);

(III) X: P(b); P(a); P(c);  
Y: P(c); P(b); P(d);  
Z: P(a); P(c); P(d);

(IV) X: P(a); P(b); P(c);  
Y: P(c); P(b); P(d);  
Z: P(c); P(d); P(a);

If at all preemption cases , even one of the X,Y,Z processes is able to complete then  $\rightarrow$  DeadLock free.

(I) ~~X: P(a); P(b); P(c);~~  
~~Y: P(b); P(c); P(d);~~  
~~Z: P(c); P(d); P(a);~~

$$a = \cancel{x} \quad b = \cancel{x} \quad c = \cancel{x} \quad d = \cancel{x}$$

$$0 \quad 0 \quad 0 \quad 0$$

- ①  $\rightarrow$  executed
- $\rightarrow$  blocked

All got blocked.

(II) ~~X: P(b); P(a); P(c);~~  
~~Y: P(b); P(c); P(d);~~  
~~Z: P(a); P(c); P(d);~~

$$a = \cancel{x} \quad b = \cancel{x} \quad c = \cancel{x} \quad d = \cancel{x}$$

$$0 \quad 0 \quad 0 \quad 0$$

completed  $\rightarrow$  No Deadlock

Because it's going to release its semaphore later

$$\begin{array}{l} \xrightarrow{\quad} a = 1 \\ \xrightarrow{\quad} c = 1 \\ \xrightarrow{\quad} d = 1 \end{array}$$

Q

Each of a set of  $n$  processes executes the following code using two semaphores  $a$  and  $b$  initialized to 1 and 0, respectively. Assume that  $count$  is a shared variable initialized to 0 and not used in CODE SECTION P.

**<CODE SECTION P >**

```
wait(a); count=count+1;      a = 1      b = 0      count = 0
if(count==n) signal(b);
signal(a); wait(b); signal(b);    n = no of processes
```

**<CODE SECTION Q >**

What does the code achieve ?

- A. It ensures that no process executes CODE SECTION Q before every process has finished CODE SECTION P.
- B. It ensures that two processes are in CODE SECTION Q at any time.
- C. It ensures that all processes execute CODE SECTION P mutually exclusively.
- D. It ensures that at most  $n-1$  processes are in CODE SECTION P at any time.

H.W.

 $P_1$ 

Count  
0

$n = 3 \rightarrow P_1 \quad P_2 \quad P_3$

 $P$ 

Count

 $a = 1$  $b = 0$ 

$a = 0 \rightarrow \text{count} = 1 \rightarrow a = 1 \rightarrow \text{blocked}$

 $P_2$  $P$ 

$a = 0 \rightarrow \text{count} = 2 \rightarrow a = 1 \rightarrow \text{blocked}$

 $P_3$  $P$ 

$a = 0 \rightarrow \text{count} = 3 \rightarrow b = 1 \rightarrow a = 1 \rightarrow b = 0$

q

 $b = 1$ 

**Five Processes invoke Incr( ) and Three Processes invoke Decr( )**

X is a shared variable initialized to 10.

$I_1$ : s value is 1 (Binary semaphore)       $\vee \rightarrow$  min of S  
 $I_2$ : s value is 2 (Counting semaphore)

**I<sub>2</sub> : s value is 2 (Counting semaphore)**

Let  $V_1$  and  $V_2$  be the minimum possible values of the implementation of I1 and I2, then choose the values of x for  $V_1$  and  $V_2$ .

- (A) 12, 7      (B) 7, 7  
 (C) 15, 7      (D) 12, 8

$P_1$     $P_2$     $P_3$     $P_4$     $P_5$

$P_6$     $P_7$     $P_8$

```

    graph LR
        A["x++ ;"] --> B[Load]
        B --> C["Wait(s) ;"]
        B --> D["Inc"]
        B --> E["signal(s) ;"]
        F[Store] --> G["Wait(s) ;"]
        F --> H["x-- ;"]
        F --> I["signal(s) ;"]
        F --> J["Dec"]
    
```

case-1

$$I_1 \quad S^0 \rightarrow \perp \quad S = 1$$

$$P_1 \curvearrowleft x = 11 \quad P_6 \curvearrowleft x = 10 \quad \boxed{10 + 5}$$

### Case -3

1

$S=2 \rightarrow$  Two process can be in  
 $\langle C.S. \rangle$

let them race to update the value of  $x$

& then always make to decreement  
one win

$$P_1 \& P_6$$

↓

P<sub>6</sub> won

$$x = 9$$

$$P_2 \& P_7$$

↓

P<sub>7</sub> won

$$x = 8$$

$$P_3 \& P_8$$

↓

P<sub>8</sub> won

$$x = 7$$

$$P_4 \& P_5$$

↓

Any one win

$$x = 8$$

$$I_1 \rightarrow \boxed{x = 12}$$

$$I_2 \rightarrow \boxed{x = 8}$$

**WRONG**

Is it necessary to store the decreased value of count after only one increment?

let all  $P_1, P_2, P_3, P_4, P_5 \rightarrow$  Increment to  $x = 15$

Now to  $P_6$  stores its value  $\boxed{x = 9} \checkmark$

then  $P_7$  & then  $P_8 \longrightarrow$   $X = ?$  Final Ans

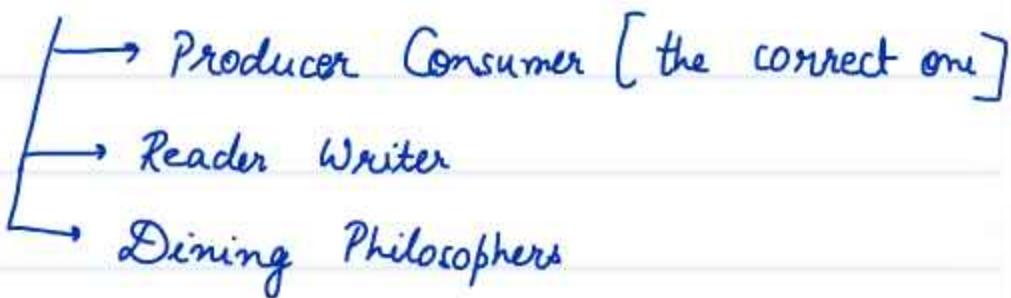
Read for more clarity.

```
// both read X = 10
pass1: incr1() & decr1()      // decr1() still in execution
                                // Final value of X = 11
pass2: incr2() & decr1()      // decr1() still in execution
                                // Final value of X = 12
pass3: incr3() & decr1()      // decr1() writes the final v
                                // Final value of X = 9

// both read X = 9
pass4: incr4() & decr2()      // decr2() writes the final v
                                // Final value of X = 8

// both read X = 8
pass5: incr5() & decr3()      // decr3() writes the final v
                                // Final value of X = 7
```

Next Lecture :- Classical IPC Problems



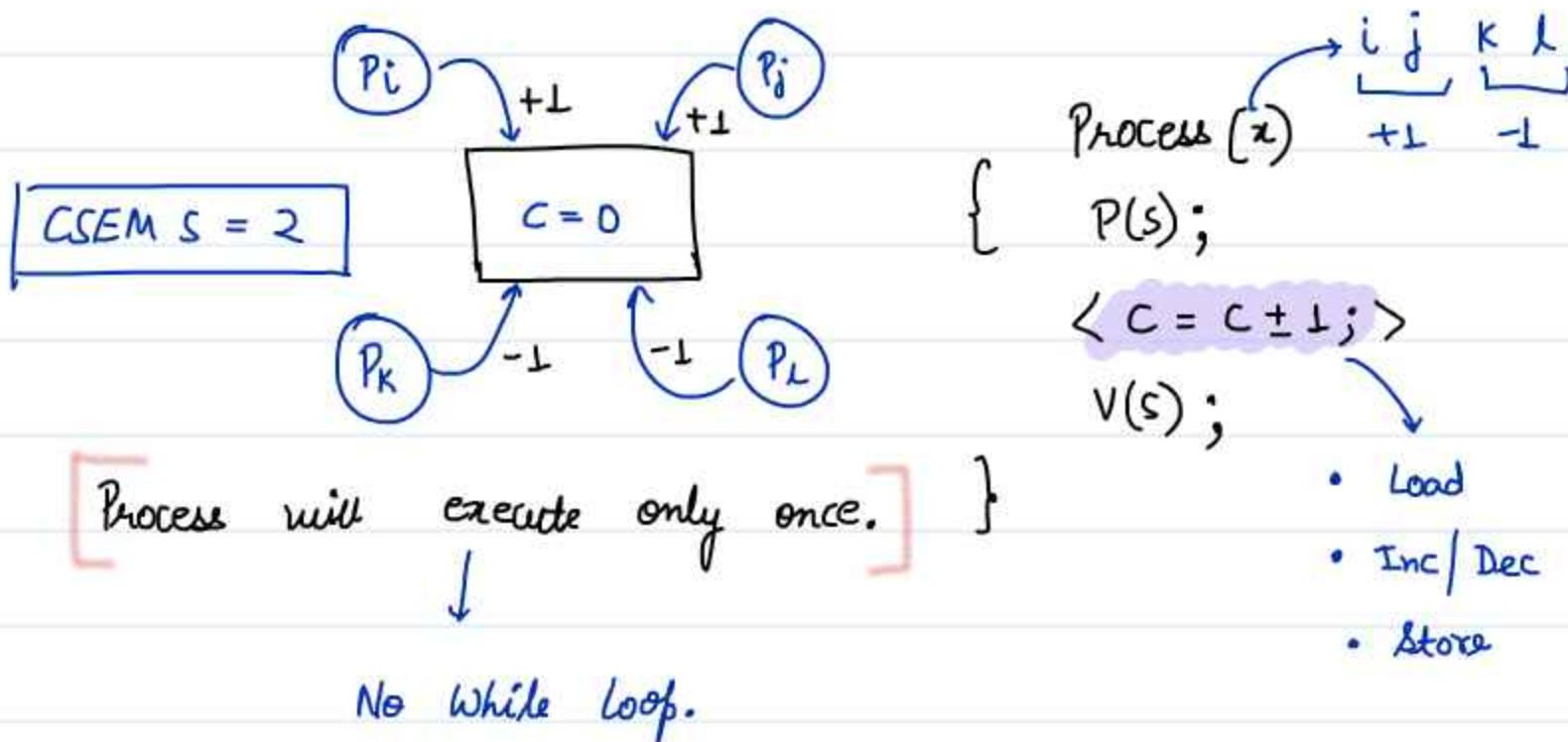
# PROCESS SYNC PART-8

→ classical IPC Problems

Let's start with the H.W.Q.

H.W.

Consider a counting semaphore initialized to 2, there are 4 concurrent processes  $P_i, P_j, P_k$  &  $P_L$ . The Processes  $P_i$  &  $P_j$  desire to increment the current value of variable  $C$  by 1 whereas  $P_k$  &  $P_L$  desire to decrement the current value of  $C$  by 1. All Processes perform their update on  $C$  under semaphore control. What can be the minimum and maximum value of  $C$  after all Processes finish their update.



Min value :-

$$C = -2$$

Max value

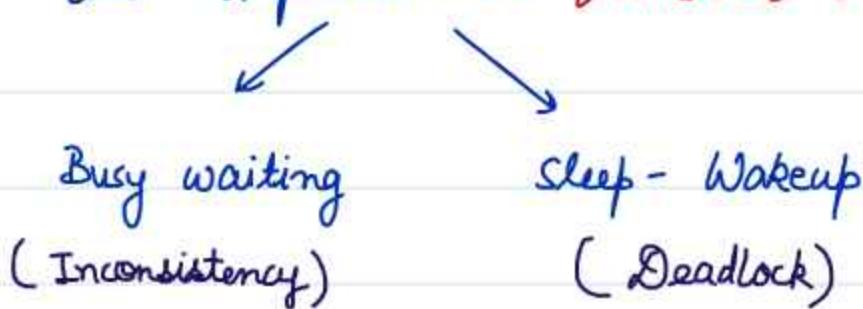
$$C = 2$$

When  
 $CSEM\ S = 2$

When  $S = 1$  then  $\text{Min} = \text{Max} = 0$

# • PRODUCER - CONSUMER PROBLEM •

We have seen two implementation [Incorrect ones]



Let's see the correct implementation using SEMAPHORES :-

```

#define N 100
int Buffer[N];
CSEM Empty = N; < No of empty slots >
CSEM Full = 0; < No of full slots >
BSEM Mutex = 1; < used b/w P & C to ensure
                  mutual exclusion b/w buffer >
  
```

No need of count variable  
condition checking

void Producer (void) {

    int itemp, in = 0;

    while (1) {

        itemp = Produce-item(); } - Non <c.s.>

        DOWN(Empty); DOWN(Mutex); } → Entry section

        Buffer[in] = itemp; in = (in + 1) % N; } → <c.s.>

    }} UP(Mutex); UP(Full); } → Exit section

```
void Consumer ( void ) {
```

```
    int itemc , out = 0 ;
```

```
    while (1) {
```

```
        DOWN ( Full ); DOWN ( Mutex ); ]→ Entry
```

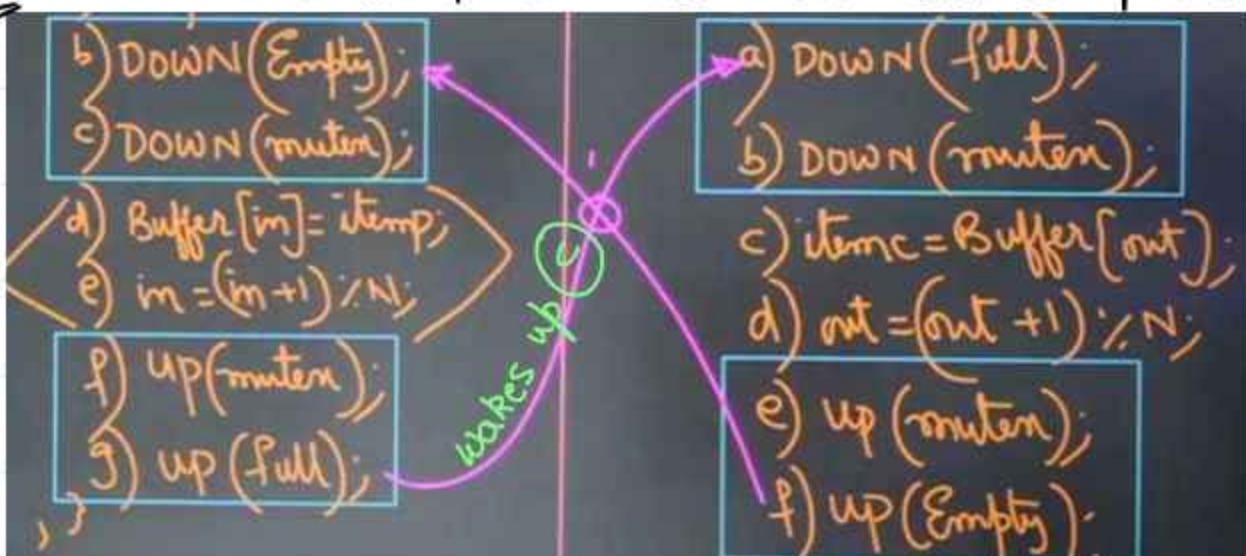
```
        itemc = Buffer [ out ]; out = ( out + 1 ) % N ; ]- < c.s. >
```

```
        UP ( Mutex ); UP ( Empty ); ]→ Exit section
```

```
        Consume - item ( itemc ); ]- Non < c.s. >
```

```
}
```

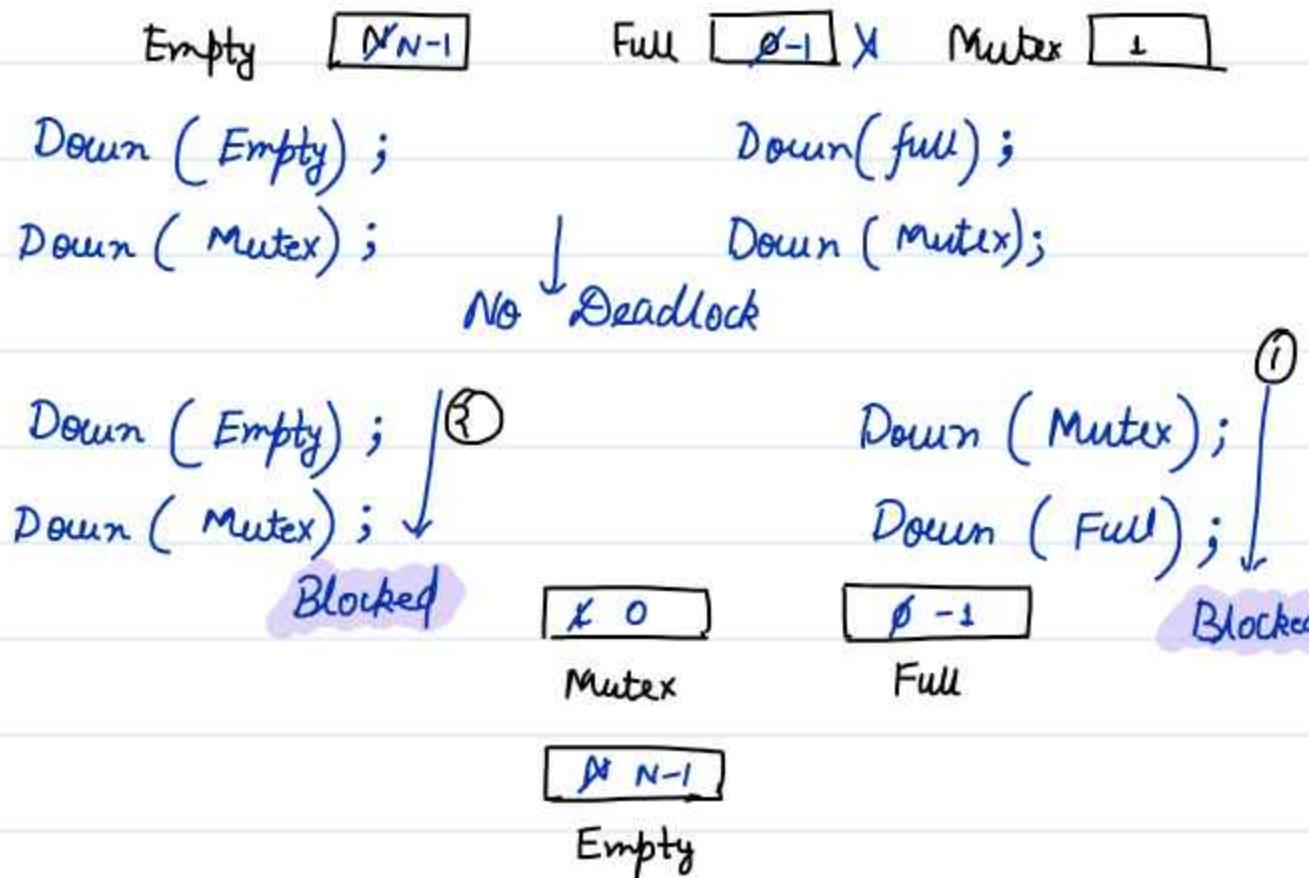
Q How Prod & Consumer are in Co-operation ?



Remember these 5 points :-

1. Buffer full? → Block producer P
2. Buffer empty? → Block consumer C
3. P should wake up C when he is sleeping.
4. C should wake up P when he is sleeping.
5. Mutual Exclusion b/w P & C.

# THE ORDER IS IMPORTANT :-



Remember:- Mutex will always be the 1st of Entry Sec.

suspicious init. values

Q

TRY Consider the following solution to the producer-consumer synchronization problem. The shared buffer size is N. Three semaphores empty, full and mutex are defined with respective initial values of 0, N and 1. Semaphore empty denotes the number of available slots in the buffer, for the consumer to read from. Semaphore full denotes the number of available slots in the buffer, for the producer to write to. The placeholder variables, denoted by P, Q, R and S, in the code below can be assigned either empty or full. The valid semaphore operations are wait() and signal().

Producer:	Consumer:
<pre>do {     wait(P);     wait(mutex);     //Add item to buffer     signal(mutex);     signal(Q); } while(1);</pre>	<pre>do {     wait(R);     wait(mutex);     //Consume item from buffer     signal(mutex);     signal(S); } while(1);</pre>

Which one of the following assignments to P, Q, R and S will yield the correct solution?

- (A) P: full, Q: full, R: empty, S: empty
- (B) P: empty, Q: empty, R: full, S: full
- (C) P: full, Q: empty, R: empty, S: full
- (D) P: empty, Q: full, R: full, S: empty

empty ]  
slots

available for  
consumer

full ]  
for Producer.

Consider the procedure below for the Producer-Consumer problem which uses semaphores:

```

semaphore n = 0;           → no of items
semaphore s = 1;
void producer()           S → mutex
{
    while(true)
    {
        produce();
        semWait(s); ↓ blocked
        addToBuffer();
        semSignal(s);
        semSignal(n);
    }
}                           → n=L
}

```

```

void consumer()
{
    while(true)
    {
        n = 0; semWait(s); ↓ blocked
        semWait(n);
        removeFromBuffer();
        semSignal(s);
        consume();
    }
}

```

Inst. interchanged!

Which one of the following is TRUE?

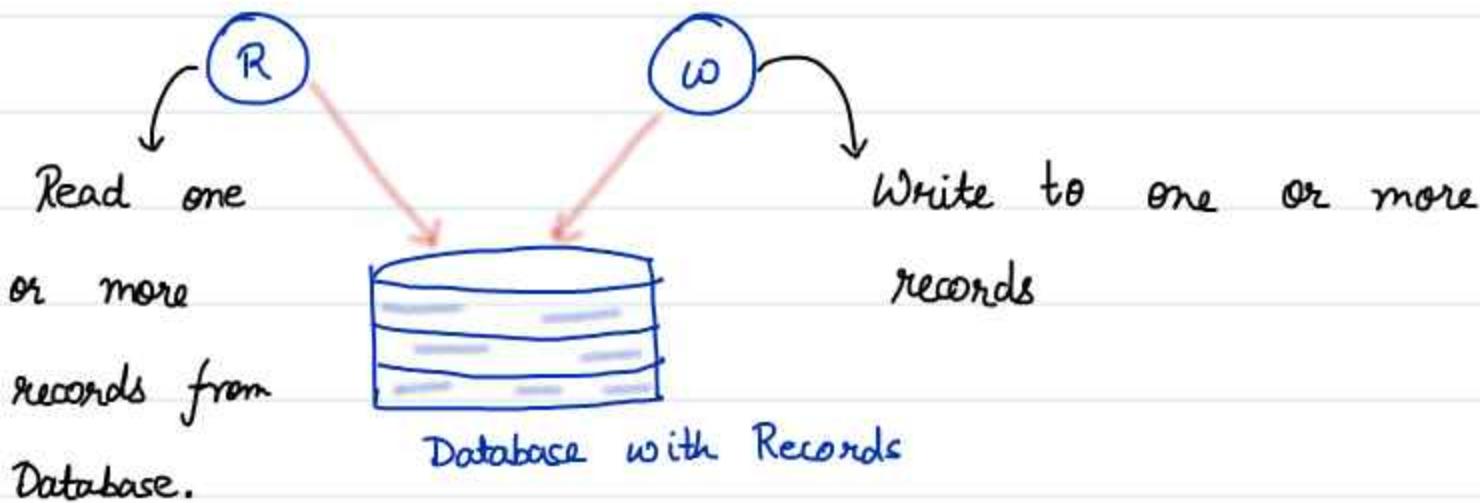
- (A) The producer will be able to add an item to the buffer, but the consumer can never consume it. X why?
- (B) The consumer will remove no more than one item from the buffer. X why?
- (C) Deadlock occurs if the consumer succeeds in acquiring semaphore s when the buffer is empty. ✓ why?
- (D) The starting value for the semaphore n must be 1 and not 0 for deadlock-free operation. X why?

Mutex at I<sub>1</sub>

To solve DL

Swap

## • READER WRITER PROBLEM •



||

Reader & Writer Should not be able to access Database at the same time.

BSEM mutex = 1;

```

{ P(mutex);
<DB>
v(mutex); }
}

```

} Problem solved

BUT there can be multiple readers & writers.

(i) Reading is not a problem.



(ii) Reading + writing will be a problem.

(iii) Writing will be a problem.



At a time multiple writers X

multiple readers ✓

reader & writer X

# First R (w) problem : Starvation to writers #

$t_1 : r_1 \& w_1 \rightarrow$  choose any

$t_1 : w_1$

$r_1$ : wait

$t_2 : r_1 \checkmark \quad w_1$ : wait

$t_2 : w_2 \times$

$t_3 : r_2 \checkmark$

$t_3 : w_3 \times$

} wait

$t_4 : r_3 \checkmark$

$t_4 : w_4 \times$

writer starves

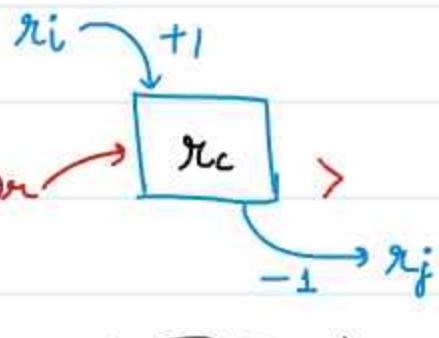
# # Implementation of First R & W using Semaphores #

int rc = 0; < Readers count >

BSEM mutex = 1; < used by R for >

BSEM db = 1;

< used by R & W to  
access db as C.S. >



Multiple reader may  
want to access rc  
for  $\uparrow$  &  $\downarrow$ .

Void writer (void) {

while () {

DOWN (db);

< DB - write >

UP (db);

}

$\downarrow$

Inconsistency

Void Reader (void) {

while () {

a) DOWN (Mutex); First one

b)  $rc++$ ; Locks.

c) if ( $rc == 1$ ) DOWN (dB);

d) UP (Mutex);

e) < DB - Read >

If the first reader checks  
that dB has writer already  
there, it will get blocked.

$\downarrow$

Rest all readers will be blocked

& if it sees no one is

mutex =  $\neq 0$   
 $dB = 0$

there in dB it can go there  
 & simply lock it for  
 writers, readers can enter  
 & no need to lock everytime.



first Reader is the Leader  
 of all other Readers.

TIP :-

- Down Mutex before accessing rc & before making it up } }
- lock if it's the first reader &  
 unlock if it's the last one.

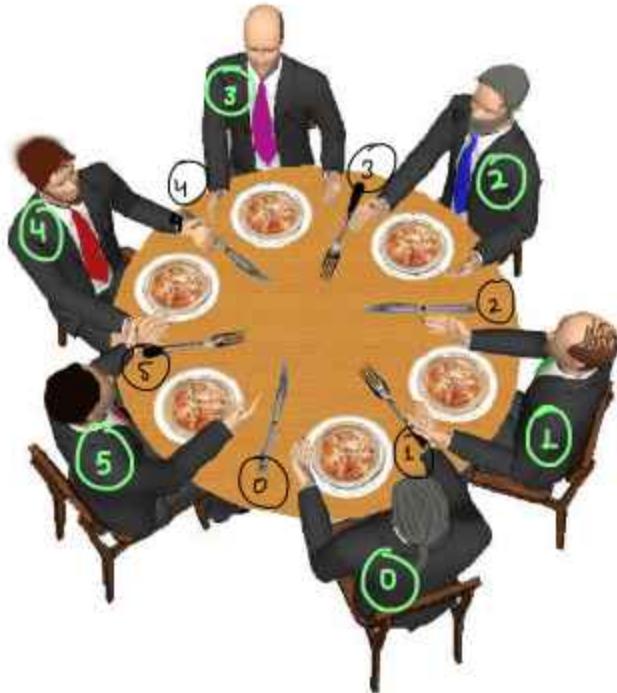
↓  
 leader will be blocked  
 at instruction c)  
 & Rest others at  
 instruction a).

f) DOWN(mutex);  
 g) rc --;  
 h) if ( $rc == 0$ ) UP(dB);  
 i) UP(mutex); }

The Last  
 one unlocks.

REVISE

## # DINING PHILOSOPHERS #



$$N \geq 2$$

Philosophers eating noodles  
a knife/fork is shared b/w  
these two.

Whenever Philosopher become hungry  
he will pick the left fork &  
then the right if successful then  
starts eating.

# define N 6

void Philosopher (int i)

{ while (1)

{ a). Think(i);

b). Take\_fork(i);

c). Take\_fork((i+1)% N);

d). eat(i);

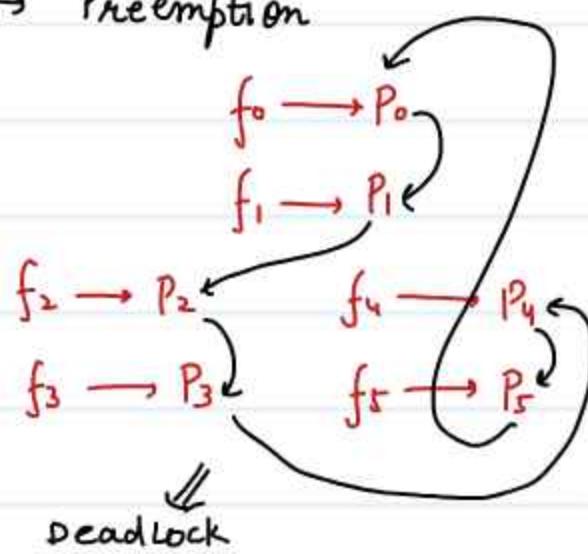
e). put\_fork(i);

f). put\_fork((i+1)% N);

}

Deadlock ?

- All Hungry
- Preemption



Deadlock

Q Max concurrency?

without DL for  $N=5$  what's the max no of Philosophers that can be eating?

For  $N=5$

$$\left. \begin{array}{l} P_0 : f_0, f_1 \checkmark \\ P_1 : f_1 X \end{array} \right\} \stackrel{2}{=}$$

For  $N=3$

$$P_2 : f_2, f_3 \checkmark$$

↓

$$P_3 : f_3 X$$

The ans would be 3.

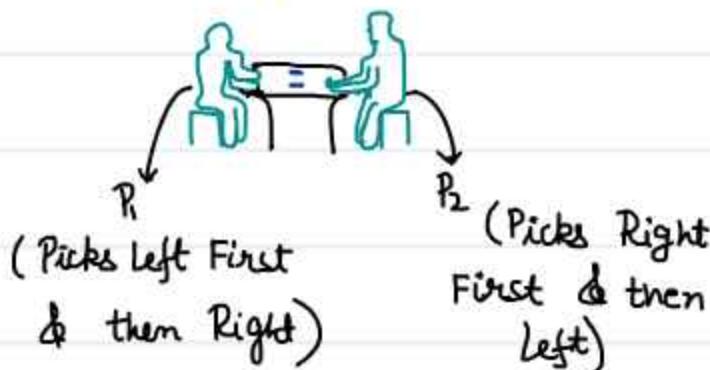
$$P_4 : f_4, f_0 X$$

## # How to Prevent Deadlock #

Semaphore Based

(Put semaphore control  
on taking & releasing  
the fork)

Non Semaphore Based



will discuss later  
(MONITORS)

this fork is right one to P<sub>2</sub>

P<sub>1</sub> picks the Left fork

Semaphore free

Deadlock free soln

P<sub>2</sub> tries for right one get

blocked. P<sub>1</sub> picks right one

eat & release forks then  
 $P_2$  eats.

↓  
 Out of  $N$  philosophers :-

Even Numbered  $\rightarrow L - R$       OR      Let  $(N-1) \rightarrow (L)$  First then  $(R)$   
 Odd Numbered  $\rightarrow R - L$       &  $L \rightarrow (R)$  First then  $(L)$

Q Fill the blanks up.

First Reader-Writer using Semaphore with Busy Waiting;

int  $R = 0, W = 0;$

Bsem mutex = 1;

Void Reader (Void)

```
{
    L1: P(mutex);
    if ( $W == 1$ )
    {
        _____;
        goto L1;
    }
    else
    {
        R = R + 1;
        _____;
    }
```

<DB\_READ>

```
P(mutex);
    R = R - 1;
V(mutex);
}
```

Void Writer (Void)

```
{
    L2: P(mutex);
    if (_____)
    {
        V(mutex);
        goto L2;
    }
    else
    {
        W = 1;
        V(mutex);
    }
}
<DB_WRITE>
P(mutex);
    W = 0;
V(mutex);
}
```

H.W.

# PROCESS SYNC PART - 9

## H.W. Question

First Reader-Writer using Semaphore with Busy W.

```
int R = 0, W = 0;
```

```
BSEM mutex = 1;
```

**Void Reader (Void)**

```
{
    L1: P(mutex);
    if(W == 1)
    {
        V(mutex);
        goto L1;
    }
    else
    {
        There is no
        writer in DB
        R = R+1;
        V(mutex) → Why?
    }
}
```

<DB\_READ> Multiple

P(mutex); readers are allowed

R = R - 1;

V(mutex); the second should perform L1.

**Void Writer ( Void)**

```

    writer already present. L2: P(mutex);
    if(____)
    {
        V(mutex);
        goto L2;
    }
    else
    {
        W=1;
        V(mutex);
    }
}
```

<DB\_WRITE>

P(mutex);

W=0;

V(mutex);

Some reader/s

already

present

$R \geq 1$

bb

$W == 1$

a writer is  
already  
present.

# Homework Exercise #

Possibility of

Deadlock in

Sleeping Barber  
problem.



H.W.

# Sleeping Barber Problem



↓ HINT

When the customer  
doesn't vacate the  
only chair even after  
the haircut.

- There is one barber, and n chairs for waiting customers
- If there are no customers, then the barber sits in his chair and sleeps (as illustrated in the picture)
- When a new customer arrives and the barber is sleeping, then he will wake up the barber
- When a new customer arrives, and the barber is busy, then he will sit on the chairs if there is any available, otherwise (when all the chairs are full) he will leave.

→ Tanenbaum  
Textbook

→ Think how this lead to Deadlock

## # CONCURRENCY Vs SEQUENTIALITY #

begin {

S<sub>1</sub> :  $a = b + c;$ S<sub>2</sub> :  $d = e \times f;$ S<sub>3</sub> :  $k = a + d;$ S<sub>4</sub> :  $l = k \times 10;$ 

} end

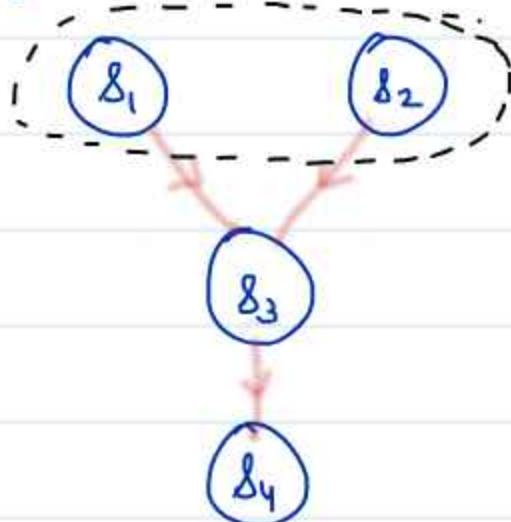
I<sub>1</sub> : Load R<sub>1</sub>, bI<sub>2</sub> : Load R<sub>2</sub>, cI<sub>3</sub> : Load R<sub>1</sub>, R<sub>2</sub>I<sub>4</sub> : Store a, R<sub>1</sub>

will also execute seq.

Sequential  
execution

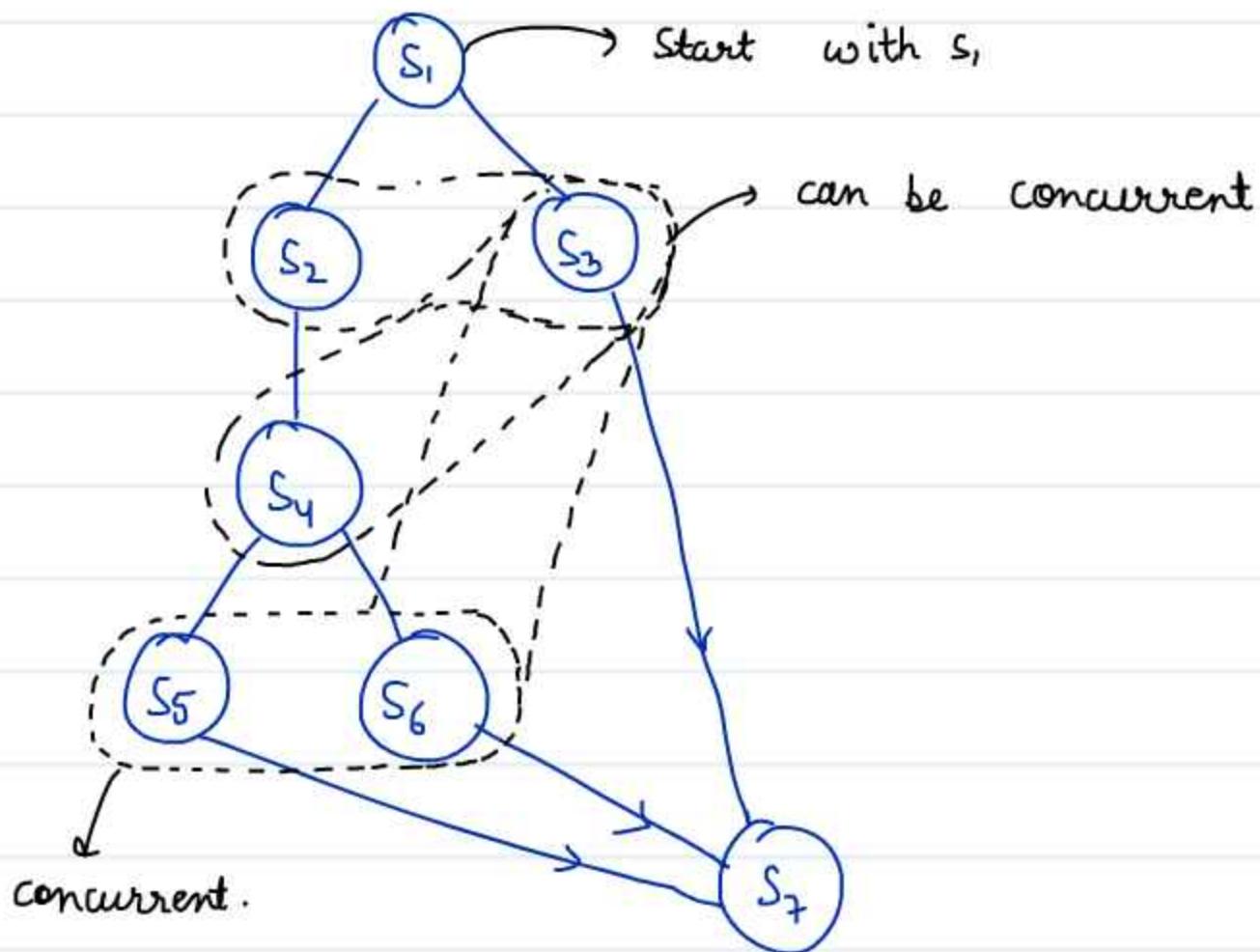
But if permissible which stmt can execute concurrently?

→ only  $s_1$  &  $s_2$  < No Dependency



can be shown using  
Precedence graph  
(directed)

Lets take another example :-



# CONCURRENCY

# PARALLELISM

A System is said to be **Concurrent** if it can support Two or more actions in **Progress** at the same time

A system is said to be **Parallel** if it can support Two or more actions Executing **simultaneously**

**Concurrency** is about dealing with lots of things at once.

Parallelism is about doing Lots of things at once.

↓  
Multiprogramming

↓  
multiprocessing

Two or more actions

Simultaneous execution

can go along together / progress together.

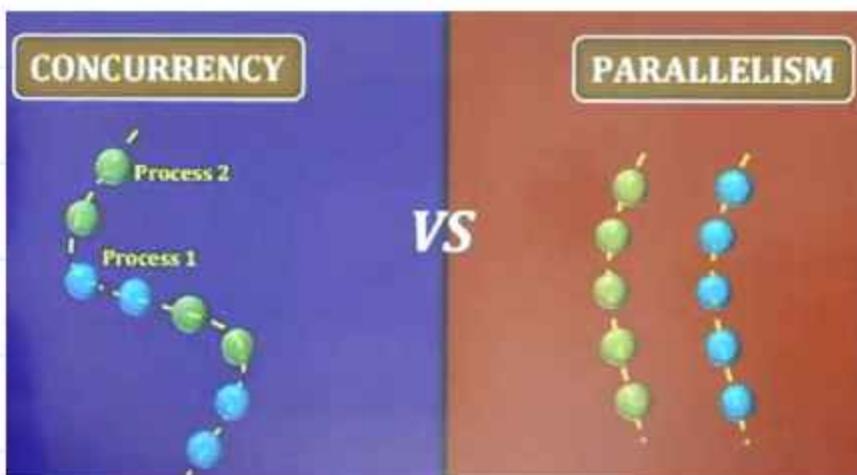
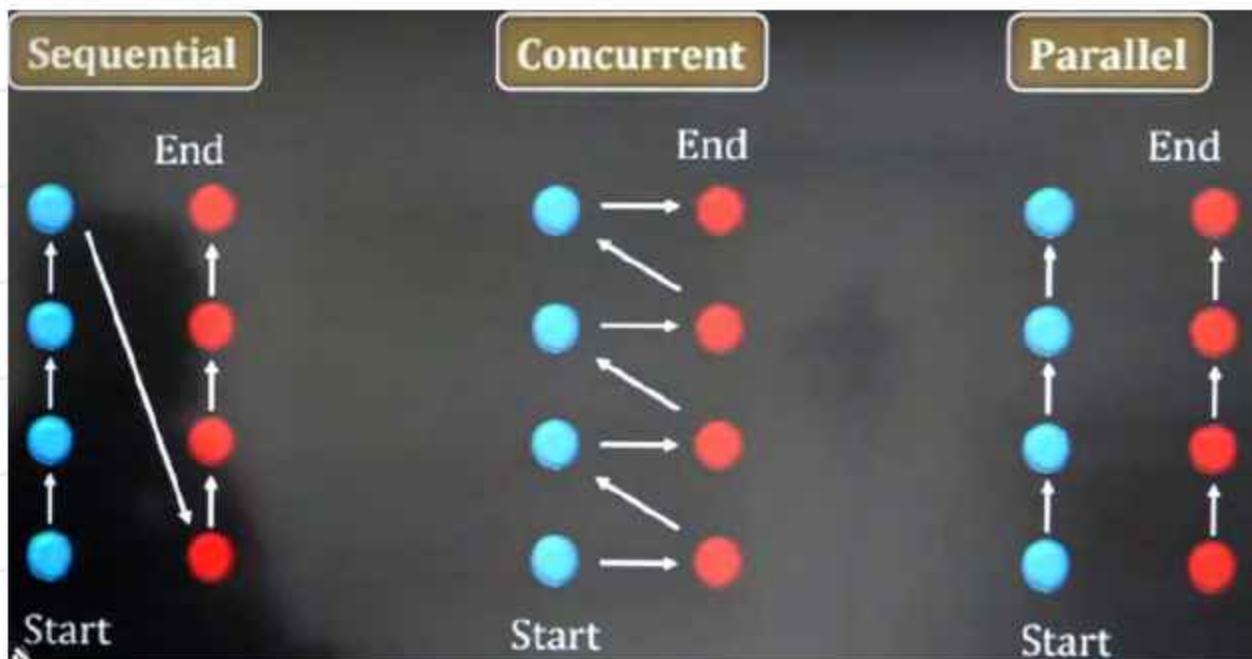
Ex:- • one person eating snacks.

• one person watching lectures

• one person listening to Lofi beats

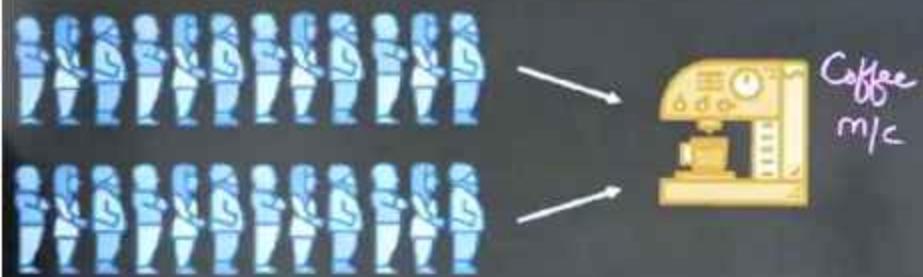
↓  
"at the same time"

- Concurrency leads to Parallelism
- Parallelism can be derived through Concurrency

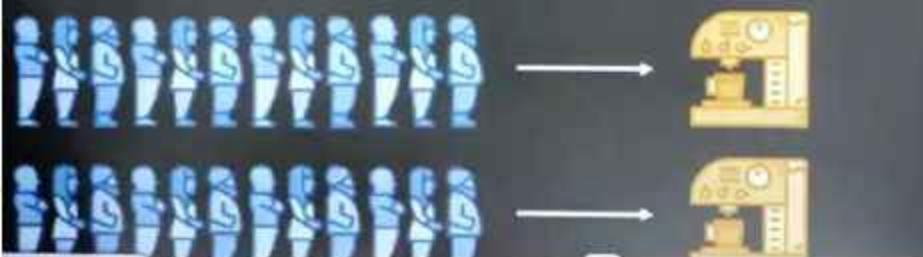


### An analogy

Concurrent = Two queues one coffee machine

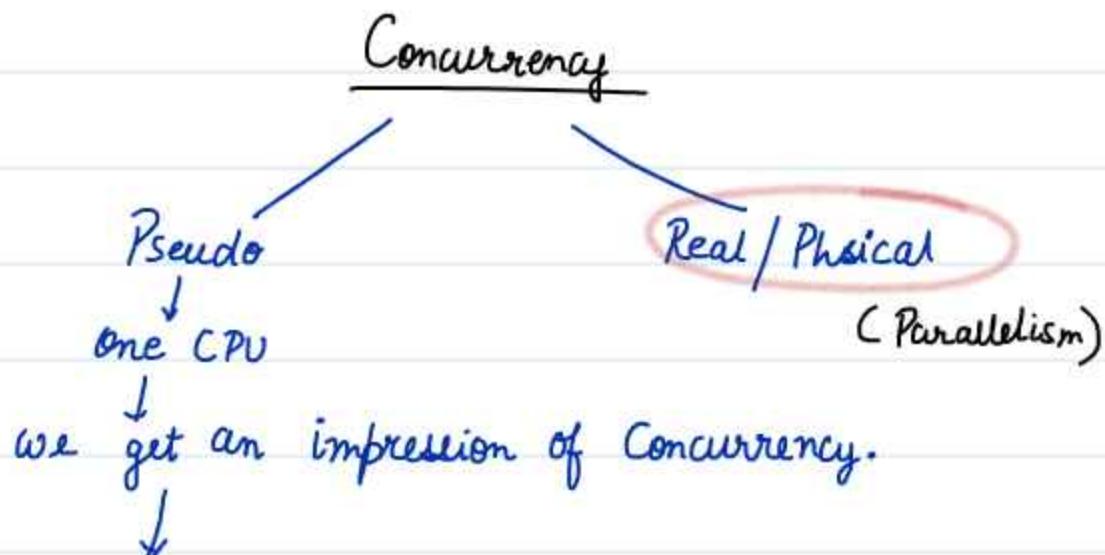


Parallel = Two Queues two coffee machine



Interleaved execution in Multiprogramming }  
 Concurrency  
 ↗ Sometimes P<sub>1</sub> Sometimes P<sub>2</sub> .....

Parallelism is possible with multiple CPU's / core.



Interleaved execution among Processes.

$s_i : a = b + c$        $s_j : d = e * f$  } can be executed  
concurrently.

$s_i : a = b + c$        $s_j : d = b * c$  } ?

YES  
↓

Even though they have shared resources  
but they are only reading the value of b & c.

If output of one becomes the input for another then those two statements won't be concurrent.

For every stmt two sets can be defined:

$s \rightarrow R(s) \rightarrow$  Set of all var whose values are read.  
 $s \rightarrow w(s) \rightarrow$  " " Whose values are updated.

$$S_1 : \begin{array}{c} a = b + c \\ \textcircled{a} \quad \textcircled{b+c} \end{array} \rightarrow R(s)$$

$\xrightarrow{\hspace{1cm}}$

$$S_1 : \begin{array}{c} a = b + c \\ \textcircled{a} \quad \textcircled{b+c} \end{array} \rightarrow w(s)$$

$$S_2 : a += ++b * --c$$

$$a = a + ( \xrightarrow{\hspace{1cm}} ++b * --c )$$

int  $x$ ;

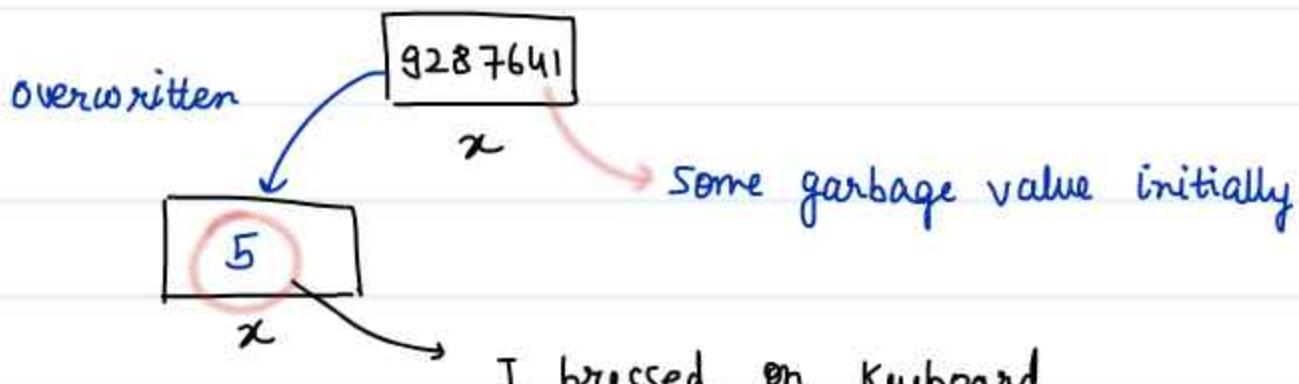
$$R(s) = \emptyset$$

$S_3 : \text{scanf}(\%d, \&x);$        $w(s) = \{a, b, c\}$  First Increase its value by 1 & then use

$$R(s) : \emptyset$$

$$w(s) : \{x\}$$

Ans. Did you think of opposite ans?



$\text{Read} = \emptyset$  because no need to read 9287641.

If  $s : \text{print}(x)$  then  $\text{Read} = \{x\}$  &  $\text{Write} = \emptyset$

For  $s_i$  &  $s_j$  to be concurrent

$$\left[ \begin{array}{l} w(s_i) \cap R(s_j) = \emptyset \\ R(s_i) \cap w(s_j) = \emptyset \\ w(s_i) \cap w(s_j) = \emptyset \end{array} \right] \rightarrow \text{Bernstein}$$

Concurrency condition

Similar to Reader Writer Problem

## # CONCURRENCY MECH #

Not from the textbooks

C, C++, Java → Sequential language

- PARBEGIN - PAREN ( Cobegin - Coend )
  - Parallel
  - Concurrent

begin

$s_1;$   
 $s_2;$   
 $s_3;$

end

sequential



$s_0$

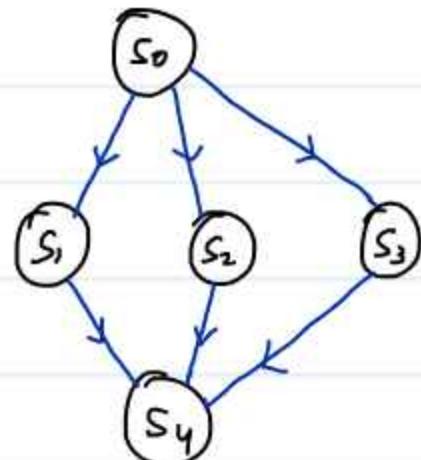
parbegin

$s_1;$   
 $s_2;$   
 $s_3;$

paren

$s_4$

parallel



$s_1$ ,

Parbegin

$s_2, s_3 ;$

$s_4 ;$

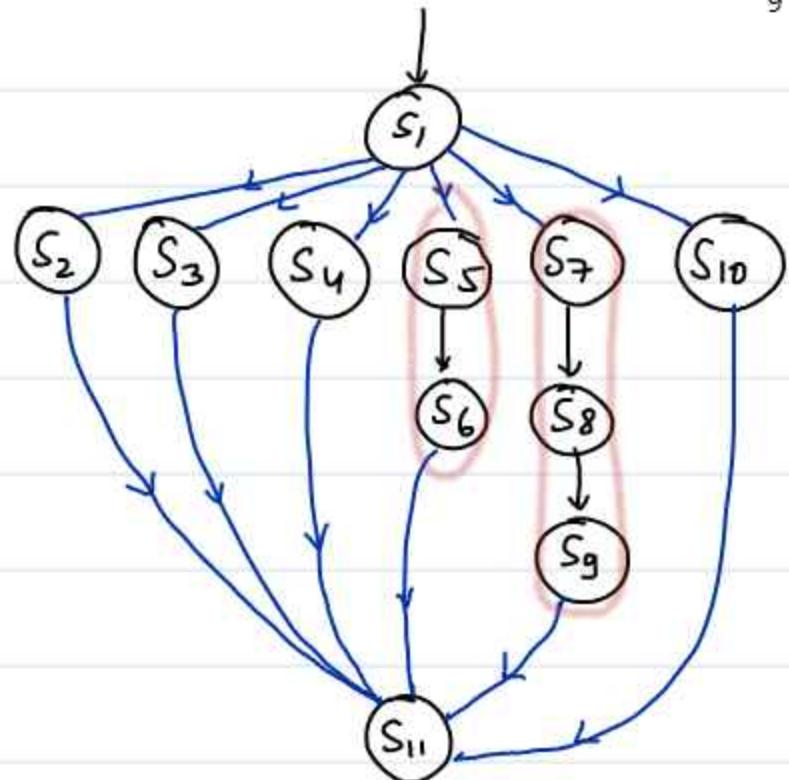
begin  $s_5 ; s_6 ;$  end

begin  $s_7 ; s_8 ; s_9 ;$  end

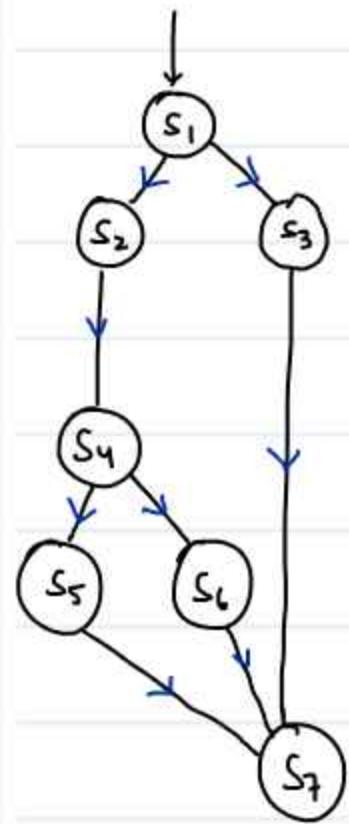
$s_{10} ;$

Parend

$s_{11} ;$



Q. Write program for the given graph:-



$s_1 ;$

Parbegin {

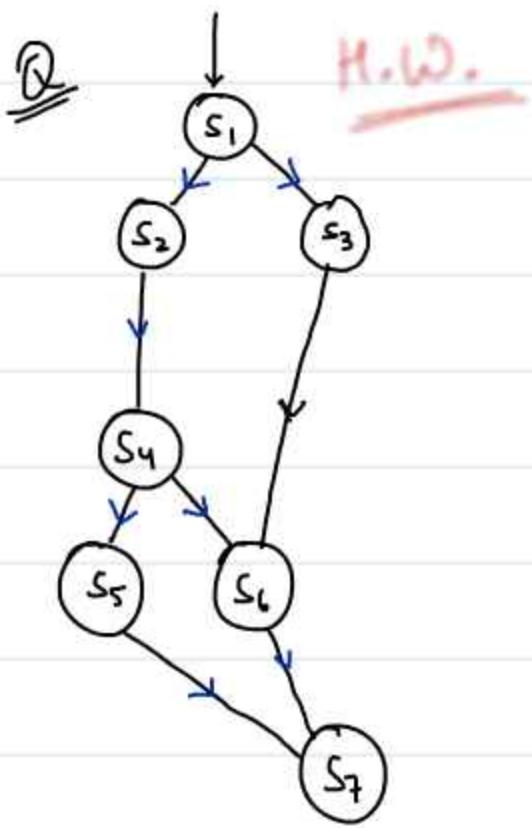
$s_3 ; \text{ Begin} \{ s_2 ;$

$s_4 ;$

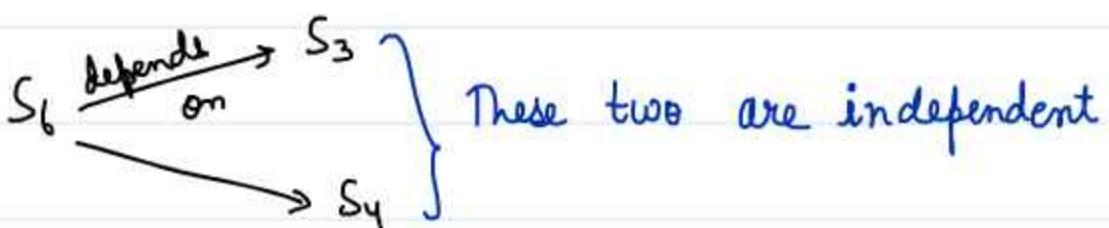
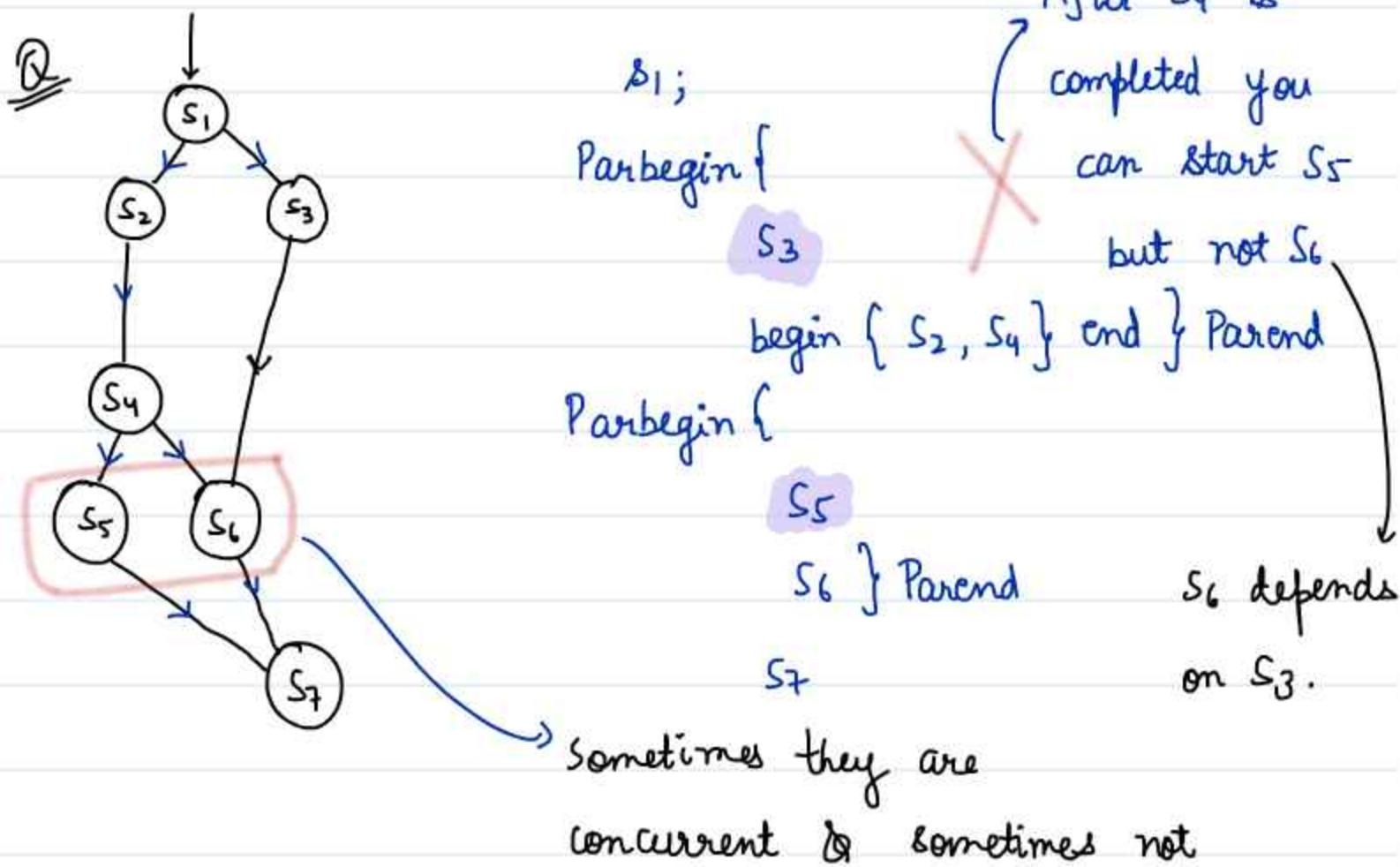
Parbegin {  $s_5 ; s_6 ;$  } Parend  
} End

} Parent

$s_7 ;$



# PROCESS - SYNC - FINAL PART



Program says you can't complete  $S_5$  before completing  $S_3$ , but graph says opposite.

The program would be (✓) if I put an edge from  $S_3$  to  $S_5$ . But now it is (✗)

Sol<sup>n</sup> :- Non Implementable

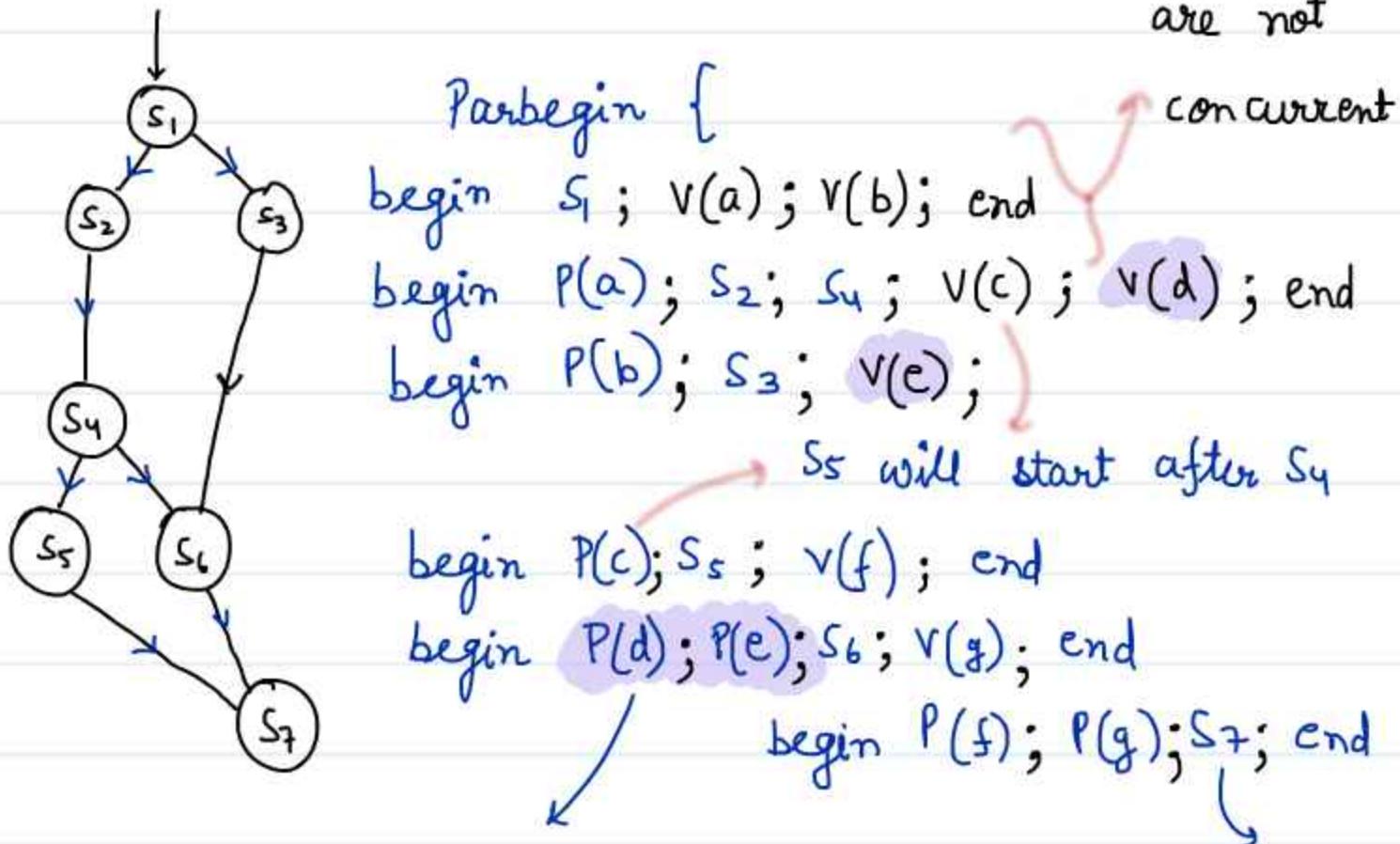
(with Parbegin & Paren alone)



Such graphs are Implementable using  
SEMAPHORES

BSEM a, b, c, d, e, f, g = {0}       $s_1, s_2 \& s_3$

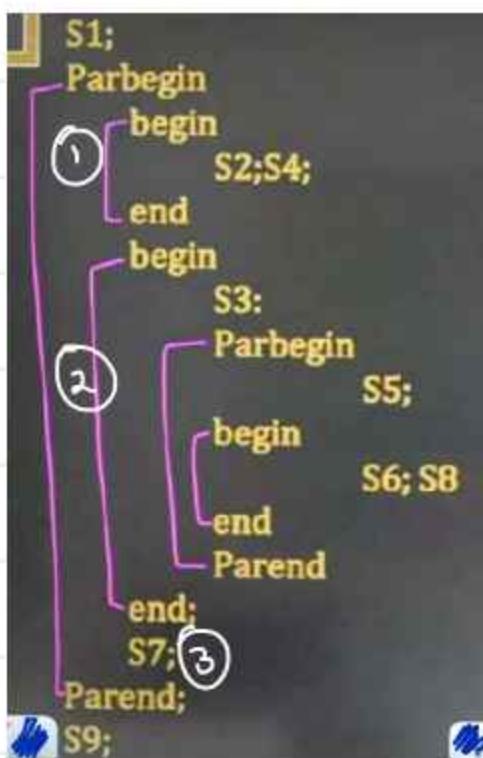
are not



H.W.

Try writing the same program with &lt; 7 sem.

Q



Q

```

void P(void) {
    A;
    B;
    C;
}
  
```

void Q(void)

```

{
    D;
    E;
}
  
```

P first then Q ✓

Q first then P ✓

Interleaved Execution ✓

```

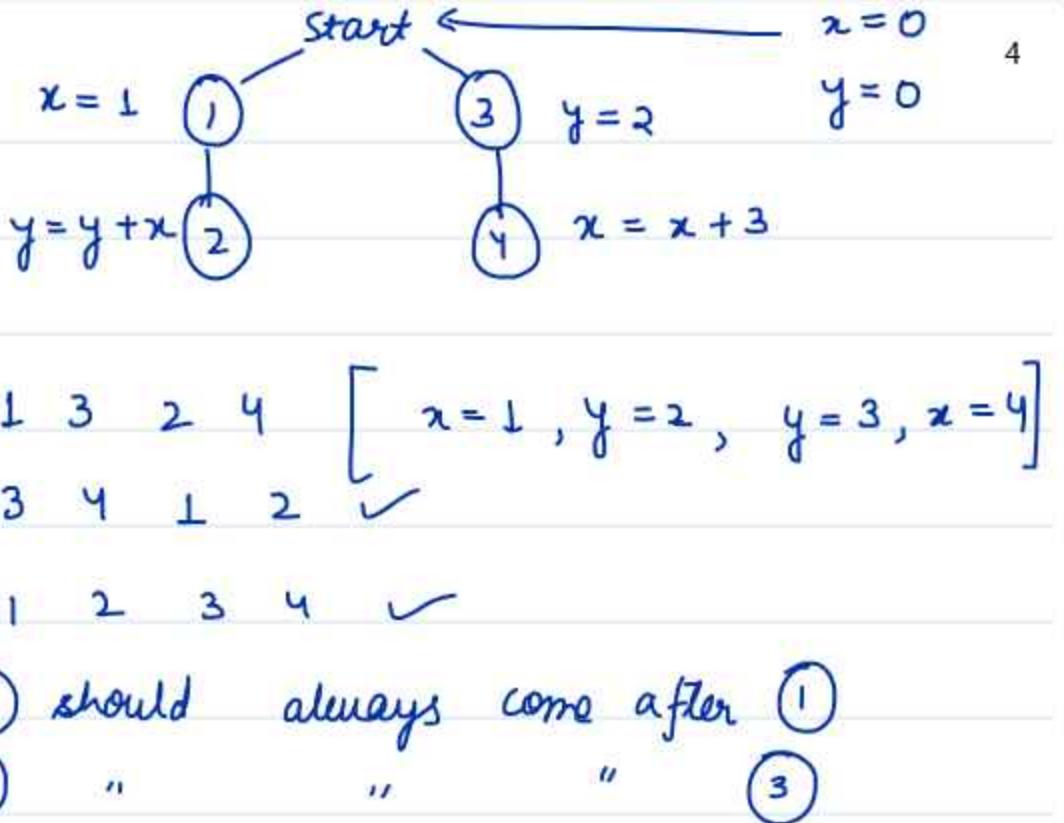
main()
{
    Parbegin
        P();
        Q();
    Paren
}
  
```

Select the valid sequences :-

- |             |              |
|-------------|--------------|
| 1. ABCDE ✓  | 2. DEABC ✓   |
| 3. ADBEC ✓  | 4. AEBDC ✗   |
| 5. DCCEBA ✗ | E before D ? |

```

Q
int x = 0, y = 0;
Cobegin
begin
  1: x = 1;
  2: y = y + x;
end
begin
  3: y = 2;
  4: x = x + 3;
end
Coend
Final values of x & y
I) x = 1; y = 2
II) x = 1; y = 3
III) x = 4; y = 6
  
```



i)  $x = 1 \quad y = 2 \quad \cancel{x} \quad \text{Not possible}$

ii)  $x = 1 \quad y = 3 \quad \checkmark \quad [3 \ 4 \ 1 \ 2]$

iii)  $x = 4 \quad y = 6$

↳ Stmt 4      ↳ Stmt 2

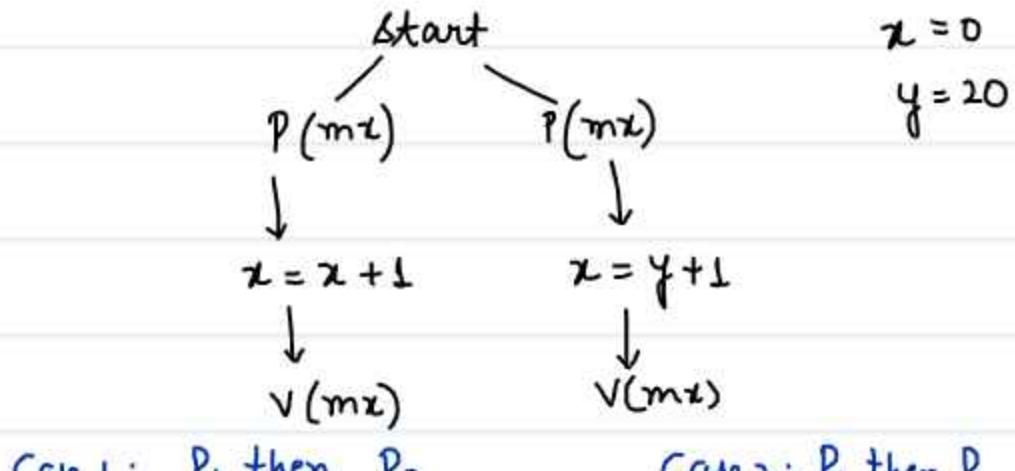
[ 1    3    4    2 ]

then                  then

Q  

```

int x = 0, y = 20;
Bsem mx = 1; my = 1;
Cobegin
begin
  P(mx);
  x = x + 1;
  V(mx);
end
begin
  P(mx);
  x = y + 1;
  V(mx);
end
Coend
Final possible values of x 21, 22
  
```



If  $P(mx)$  &  $V(mx)$  were not there then  $x = ?$

HINT :- Think Low Level.

## (Atomic) "FORK & JOIN" (Not the System call)

Syntax of Fork  $\Rightarrow$  fork L ;

Label

Execution of fork results in starting two computations concurrently.

- (i) which is Immediately after fork
- (ii) which is at Label 'L'.

int count = 2 ;

L,

fork L ;

8<sub>2</sub>

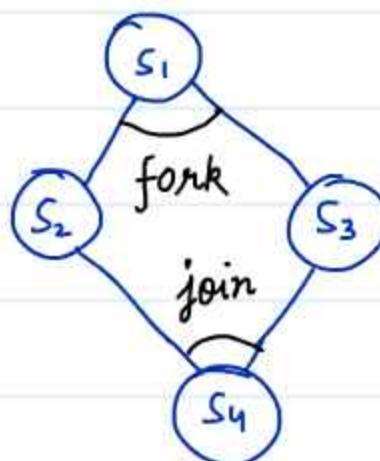
goto x ;

L : S<sub>3</sub>

X : Join (count);

S<sub>4</sub>; no of stmts

which S<sub>4</sub> is joining



Join (int count)

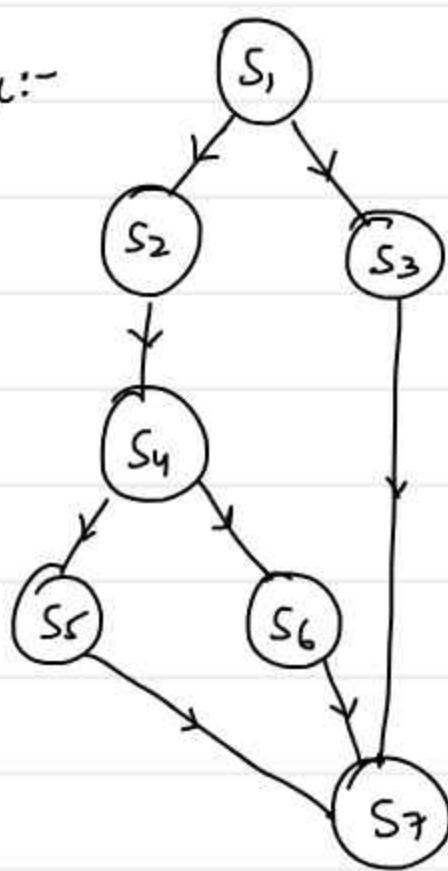
- 1. count = count - 1;
- 2. if (count != 0) then exit
- 3. else return;

After  $s_2$  &  $s_3$  the control should come to join out of  $s_2$  &  $s_3$  the one which completes at last (that is going to make count 0) that will start  $s_4$ .



$s_4$  starts only after completion of  $s_2$  &  $s_3$

Ex:-



$s_1$   
fork L

$s_2$   
 $s_4$   
fork K

$s_5$   
 $s_6$   
goto Z

L:  $s_3$

goto Z

K:  $s_6$  [no need to write join]

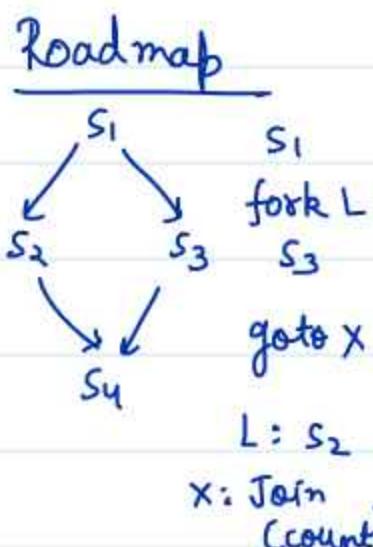
Z: Join(count);

$s_7$

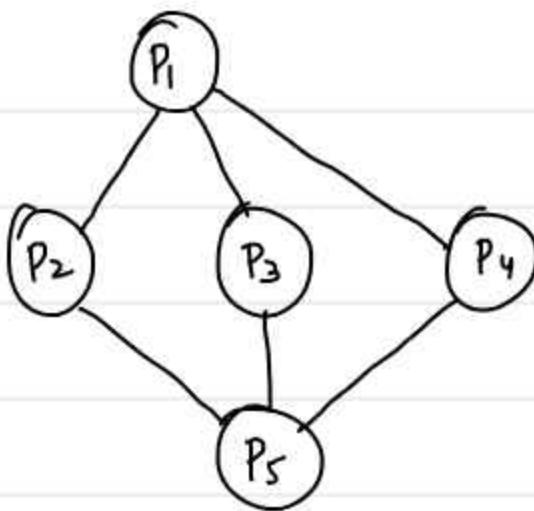
the next stmt is

join itself.]

int count = 3



X: Join  
(count)

QP<sub>1</sub>

fork L

int count = 3

P<sub>2</sub>

goto K

L : fork X

P<sub>3</sub>

goto K

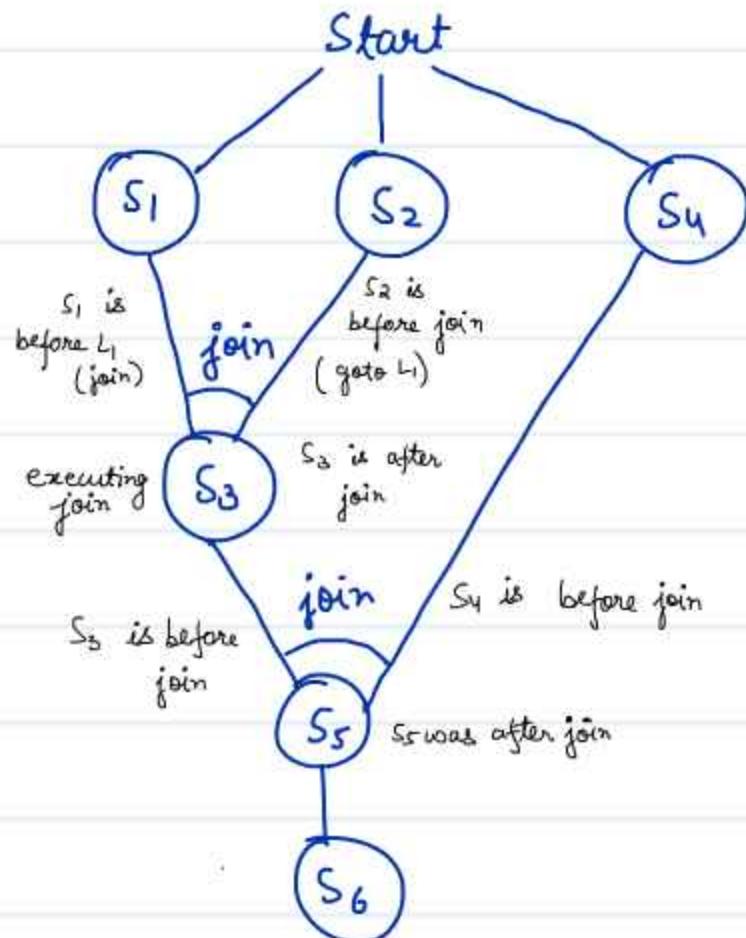
X : P<sub>4</sub>

K : Join(count)

P<sub>5</sub>Q

$N = 2$   
 $M = 2$   
 fork L3;  
 fork L4;  
 S1; } Three start will start concurrently.  
 S3; } executing join  
 L1: Join N  
 L2: Join M  
 S5: goto next;  
 L3: S2;  
 goto L1;  
 L4: S4;  
 goto L2;  
 next: S6;

Draw the Precedence Graph



Q Consider the following pseudocode, where S is a semaphore initialized to 5 in line#2 and counter is a shared variable initialized to 0 in line#1. Assume that the increment operation in line #7 is not atomic.

```

1. int counter = 0
2. Semaphore S = init(5);
3. void parop(void)
4. {
5.     wait(S);
6.     wait(S);
7.     counter++;
8.     signal(S);
9.     signal(S);
10. }
    
```

$S = 5 \checkmark$   
counter = 0

$T_1 \quad T_2 \quad T_3 \quad T_4 \quad T_5$

Load  
Increment  
Store

prompt  
after  
one wait

If five threads execute the function **parop** concurrently, which of the following program behavior(s) is/are possible?

There is a deadlock involving all the threads  $\checkmark$

The value of counter is 5 after all the threads successfully complete the execution of **parop**  $\checkmark$

The value of counter is 1 after all the threads successfully complete the execution of **parop**  $\checkmark$

The value of counter is 0 after all the threads successfully complete the execution of **parop**  $\times$

$P_1$  : Load, Increment, Pre

$P_2, P_3, P_4, P_5 \rightarrow$  complete

$P_1$  : Store

counter  
 $\emptyset \times \perp$

Reg  $\checkmark$

The value of counter can be 1, 2, 3, 4, 5

depends on how process executes.

H.W.

(b) Let's initialize semaphore = 1 & remove one wait & signal

sem = 1

wait(s)

count++

signal(s)

What are the possible values of count?

Q

```
int count = 0;
void test()
```

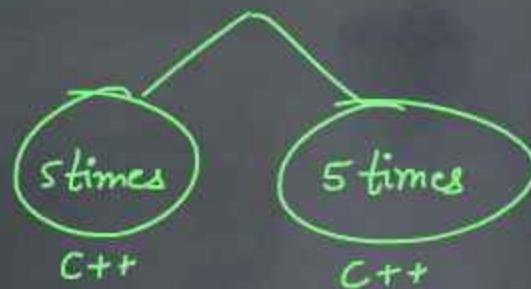
```
{
    int i, n = 5;
    for(i = 1; i <= n; ++i)
        count = count + 1; ] → 5 times
}
```

```
main()
{
    Parbegin
        test();
        test();
    Parenend
}
```

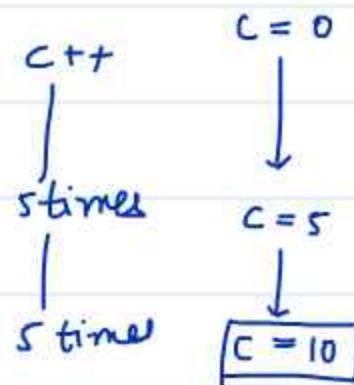
What is the minimum and maximum value of count?

Min ≠ 5

Min ≠ L



sequential execution



Now tell what will be the min value?



H.W.

=====

$\text{turn} = 0$

### K-W. Problem

$t_1$ :  : NCS ; while ( $\text{turn} \neq 0$ ) ; Pre  
CPU

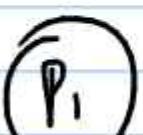
$i = 0 \ j = 1$

$t_2$ :  : NCS , while ( $\text{turn} \neq 1$ ) ; BW  
 $i = 1 \ j = 0$

### Case-2

$\text{turn} = 0 \perp$

$t_1$   NCS ; while ( $\text{turn} = 1$ ) ;  $\text{turn} = 1$  ; CS ✓  
 $i = 0 \ j = 1$

$t_2$   NCS ; while ( $\text{turn} = 0$ ) ;  $\text{turn} = 0$  CS ✓  
 $i = 1 \ j = 0$



Mutual exclusion

is not guaranteed.

# DEADLOCK

# PART - I



Two or more processes are waiting for the happening of the event that is never going to happen.

↓  
Consequences

- Throughput / Efficiency Drops.
- Ineffective Utilization of Resources.
- Deadlock is therefore undesirable.

Deadlock



Blocking for Infinite time  
(Forever)

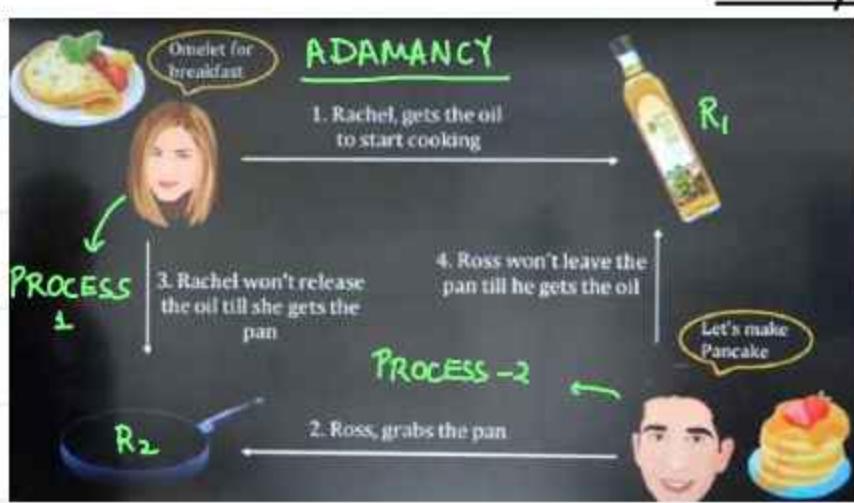
Vs

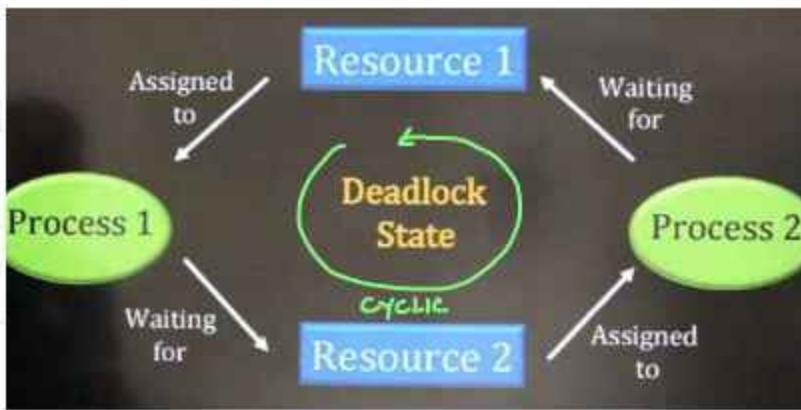
Starvation



Blocking for indefinite time

## Examples





## # System Model #

What kind of system we are assuming for study of deadlock.

$n$ : no of Processes

$$\langle P_1, P_2 \dots P_n \rangle$$

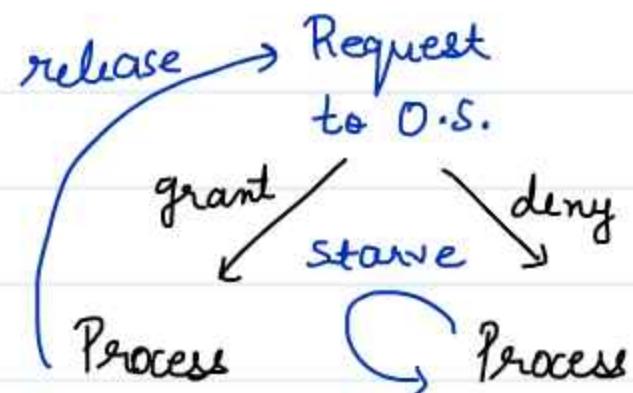
$m$ : Resources

$$\langle R_1, R_2 \dots R_m \rangle$$

h.w                    s.w

Single Instance

Multi Instance  
(copies)

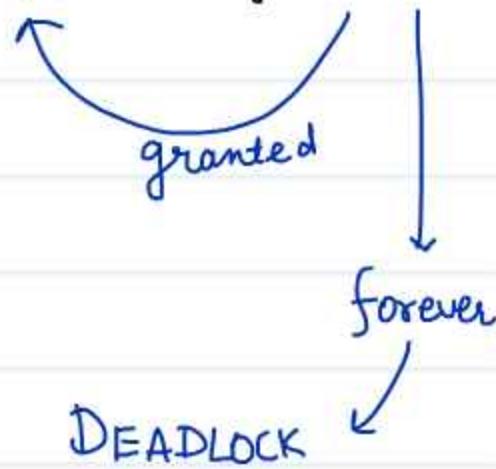


use Resources      gets blocked

Ex:- Semaphore

If one s then single Instance

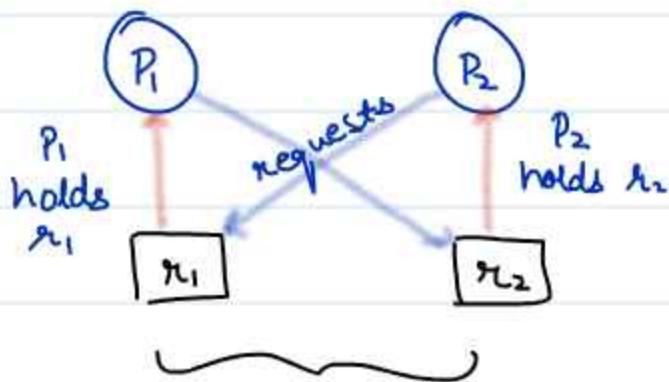
If multiple s then Multi Instance



# NECESSARY CONDITIONS

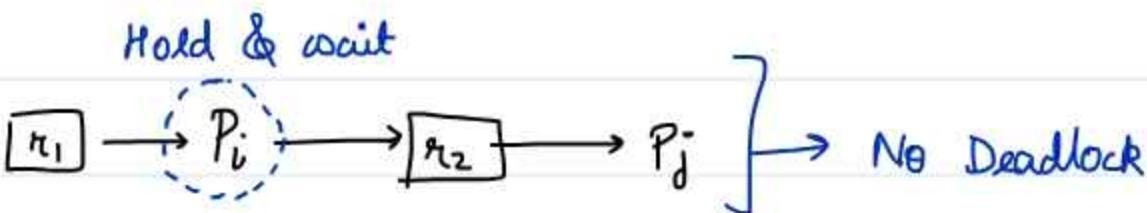
- ① There should be Critical Section / some shared resources  
Deadlock will never happen among Independent processes
- ② Process should hold some resource & should wait for another resource.

Hold & wait ~~⇒~~ Deadlock



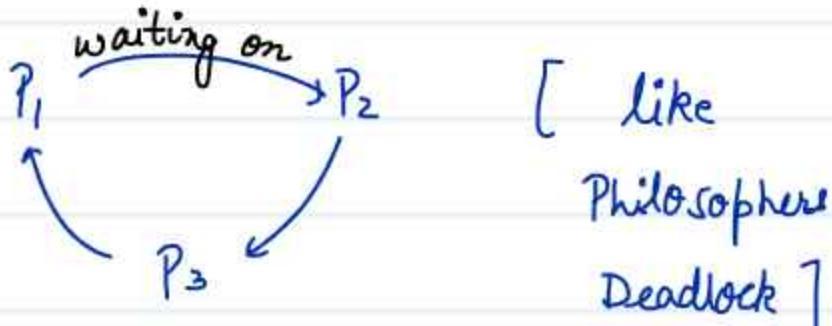
Example for Deadlock

Deadlock  ⇒ Hold & wait



- ③ No forcefull Snatching of Resource [ No Preemption ]

- ④ Circular Wait



[ like  
Philosophers  
Deadlock ]

If these 4 conditions are present then there is a Possibility of Deadlock.

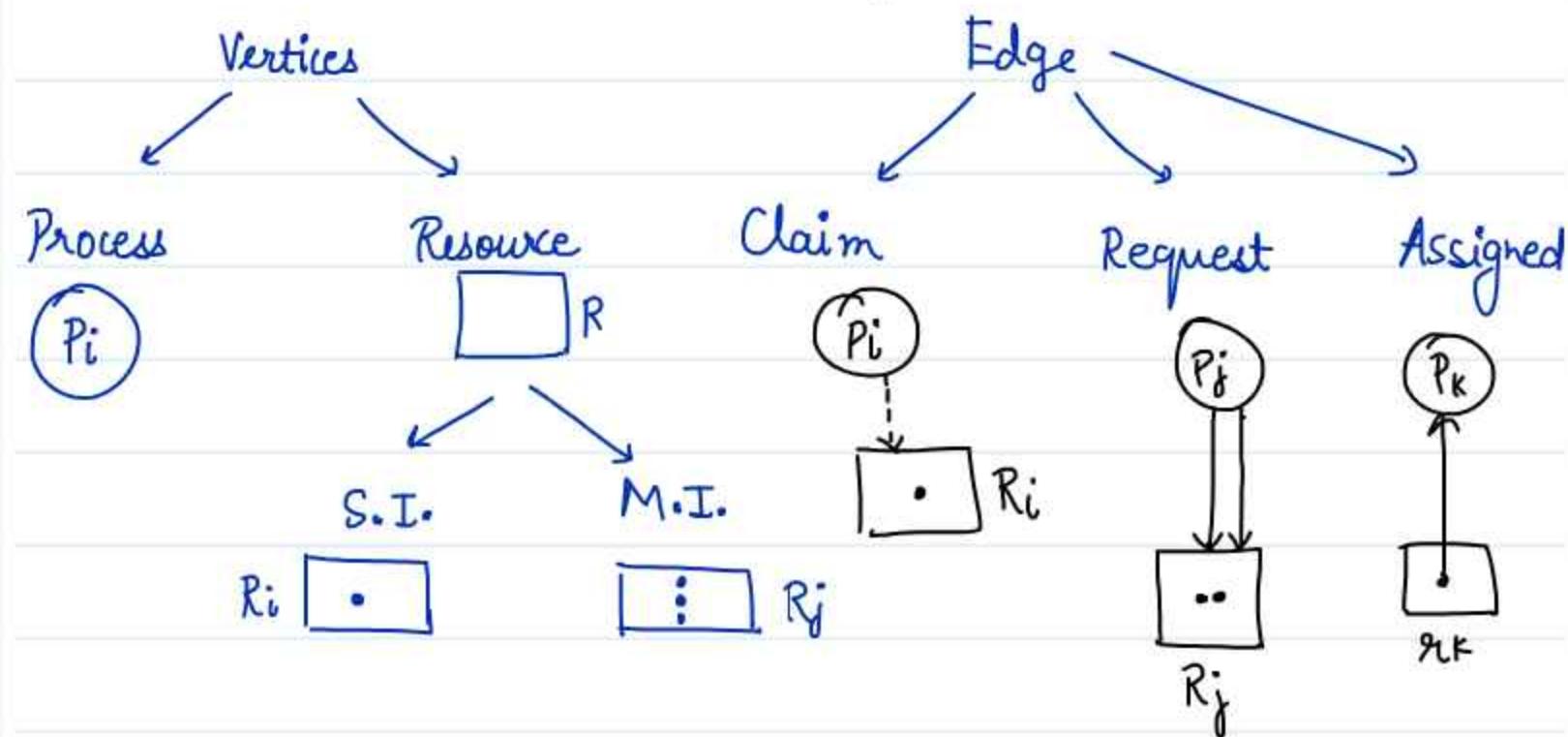


If there is Deadlock then these 4 conditions MUST be present.

## • RESOURCE ALLOCATION GRAPH •

(MultiGraph)

$(V, E)$



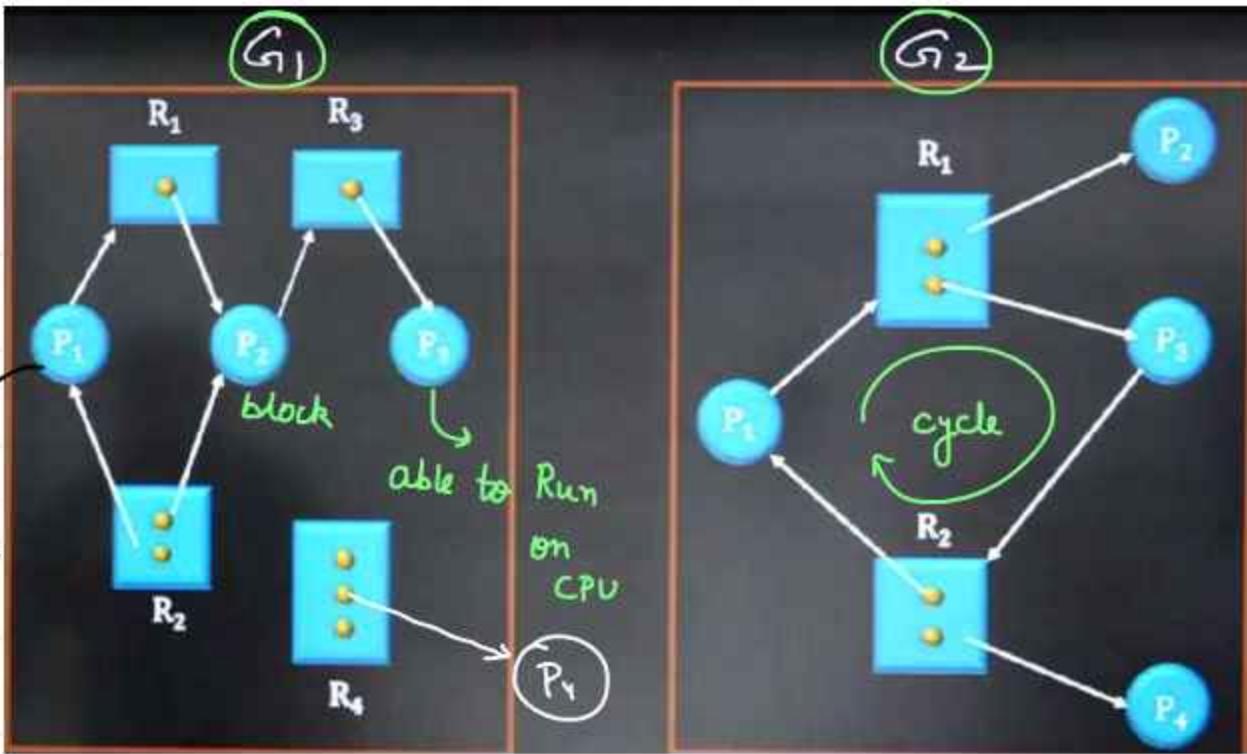
No of requests / assignment

That many edges.

Claim :- May Request in future

Request :- Actually Requested

Assigned :- Resource allocated to process.



Not the case of Deadlock

[ Suppose P<sub>3</sub> makes request of R<sub>2</sub> ]

↳ gets blocked

P<sub>1</sub> → blocked

P<sub>2</sub> → blocked

P<sub>3</sub> → blocked

cycle is only  
necessary condition

Cycle → Deadlock X

Deadlock → cycle ✓

+

Cycle: P<sub>1</sub> - R<sub>1</sub> - P<sub>2</sub> - R<sub>3</sub> - P<sub>3</sub> - R<sub>2</sub> - P<sub>1</sub>

↓

Process involved in cycle are blocked → Deadlock

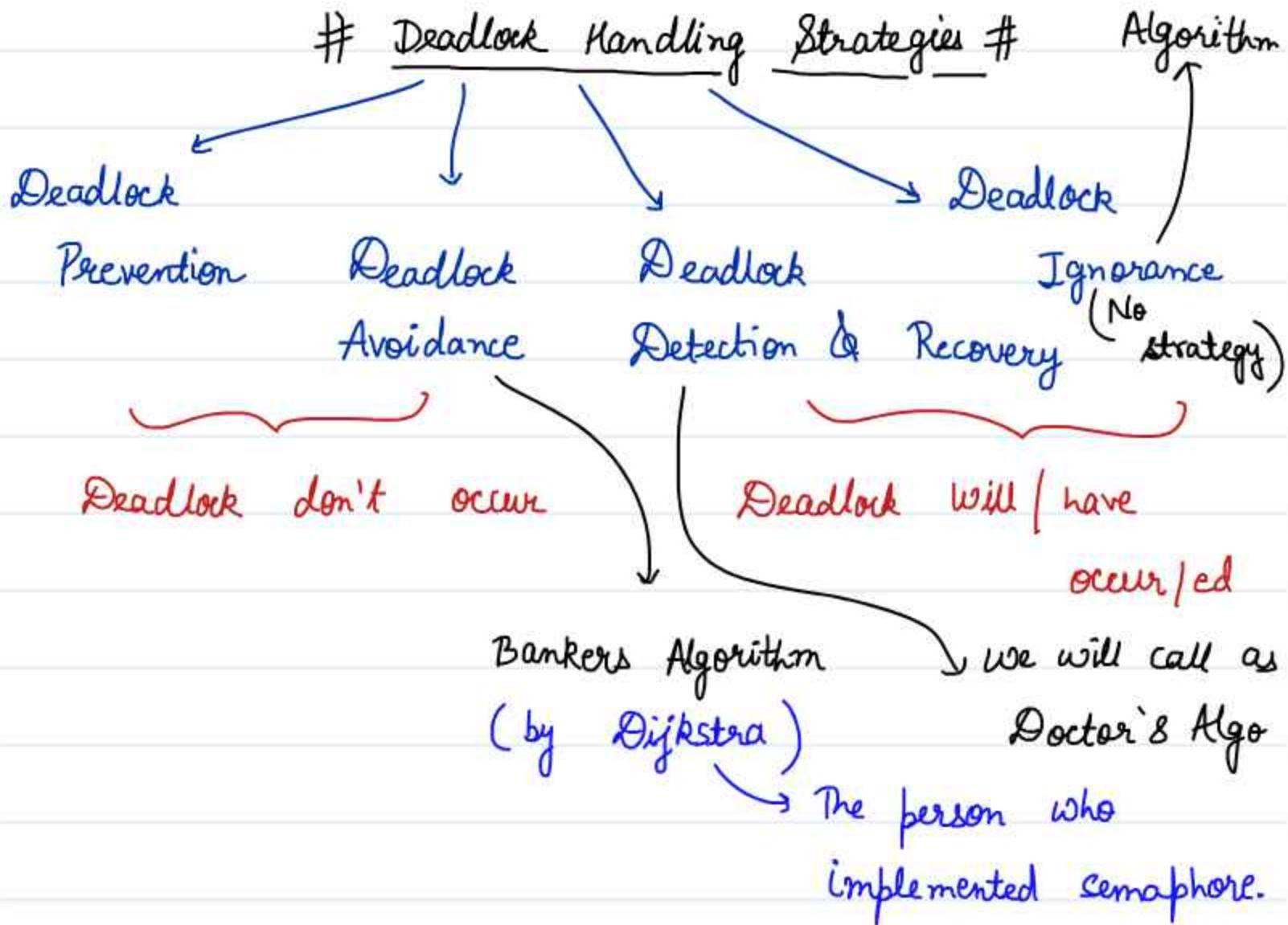
(No need for all the process in system  
to get blocked)

⑥ cycle can be broken if  $P_2 \& P_4$  release resources

In future if  $P_2$  asks for  $R_4$

if  $P_1$  asks for  $R_1$

Ostrich



- Ostrich Algorithm (No strategy)

↳ system will go into deadlock

In the worst case it'll get hanged.



→ In that case, we press **RESTART** button.

Software / System may get  crashed

Fortunately / unfortunately Windows / UNIX - LINUX uses Ostrich Algorithm because deadlocks occurs very rarely & cost of prevention is high.

Day to day O.S. don't require deadlock prevention we have to use that in those O.S. in which data & time are critical.



you can't lose 1B of data, even for 1ms system shouldn't be down.

Ex:- ATC, Missile control, Satellite CS.

## \* DEADLOCK PREVENTION 4

→ Negating one / more of necessary conditions.

① Critical section :- Remove it?



Can you imagine a multiprogrammed O.S. without a shared resource?



NO it's practically impossible.



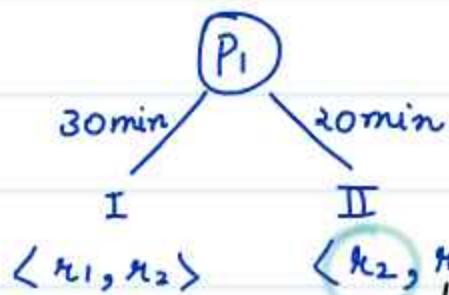
Non-Dissatisfiable

② Hold & Wait :- Instead of Hold & wait



Hold or wait

Process must request & be allocated all resources prior to its start.



After the completion of Phase I

< $r_1, r_2$ > process will release  $r_1$ , hold  $r_2$  & request for  $r_3$  (if not available then wait).

Holding &

waiting

Protocol 1

To avoid this case process will ask for all  $\langle r_1, r_2, r_3 \rangle$  before the beginning of phase I.



Drawbacks :-

① Starvation (if wait)

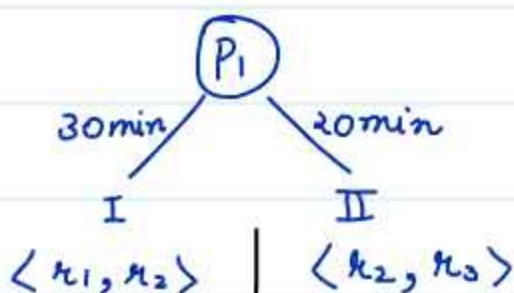
② Inefficiency (if hold)

In phase I

why holding  $r_3$  for 30 min

Protocol-2

Process must release all resource before making a fresh / new request.

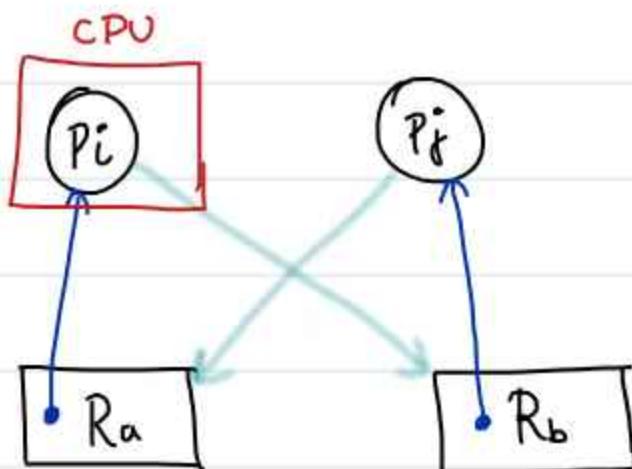


- No inefficiency
- Starvation

After Phase-I process will release  $r_1$  &  $r_2$  & will ask for  $r_2$  &  $r_3$ . No Holding of Resources but there is no guarantee that it will get resource  $r_2$  immediately.

3

No Preemption :- Preemption of Resources from Processes.



Forccful  
Running process  
Self  
selfless  
should not get blocked. attitude.

Allow  $P_i$  to forcefully take away  $R_b$  without bothering what will happen to  $P_j$ .

Running process got the right to snatch away resources of ready processes.

Forceful

Self { Running Process  $P_i$  finds out that  $R_b$  is not available, instead of snatching  $R_b$  it'll release  $R_a$  thinking that others might be in need of my resources.

Both strategies prevent deadlock.

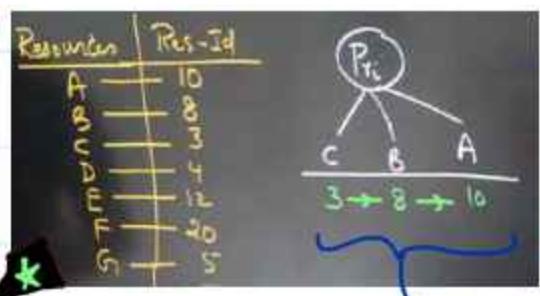
Problem → Starvation

(4)

Circular Wait → Prevented by total order relation among processes & resources.

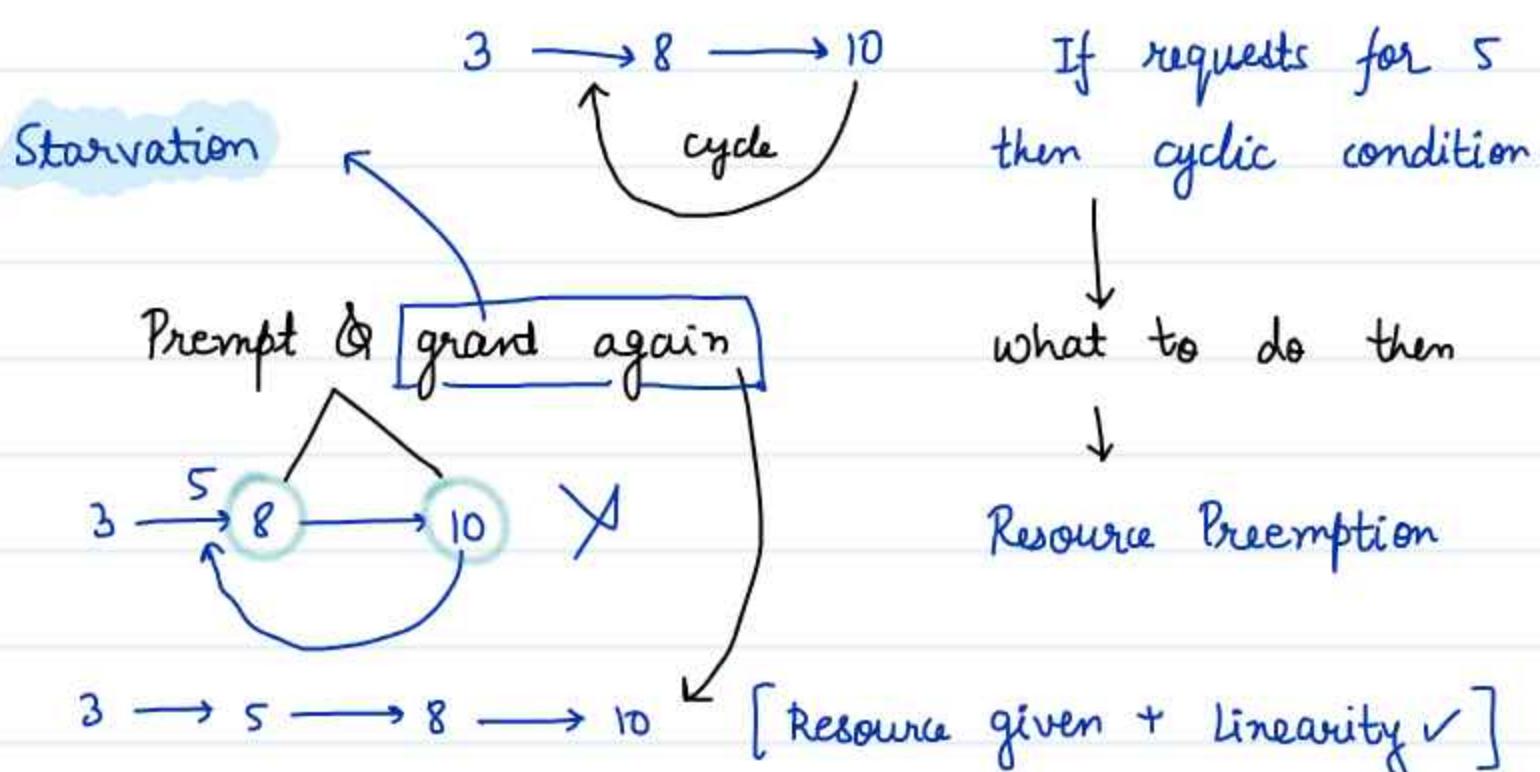
Assign unique number to each resource

Never allow a process to request a lower numbered resource than the last one allocated.



$P_{r_i}$  can only request for those resources whose id  $> 10$

Implementing linearity to avoid circularity.



# DEADLOCK : PART- II

- If RAG has resource of multi instance type then cycle is only a necessary condition

cycle  $\xrightarrow[\text{may not}]{\text{may or}}$  Deadlock

only Multi instance :- Cycle is Necessary

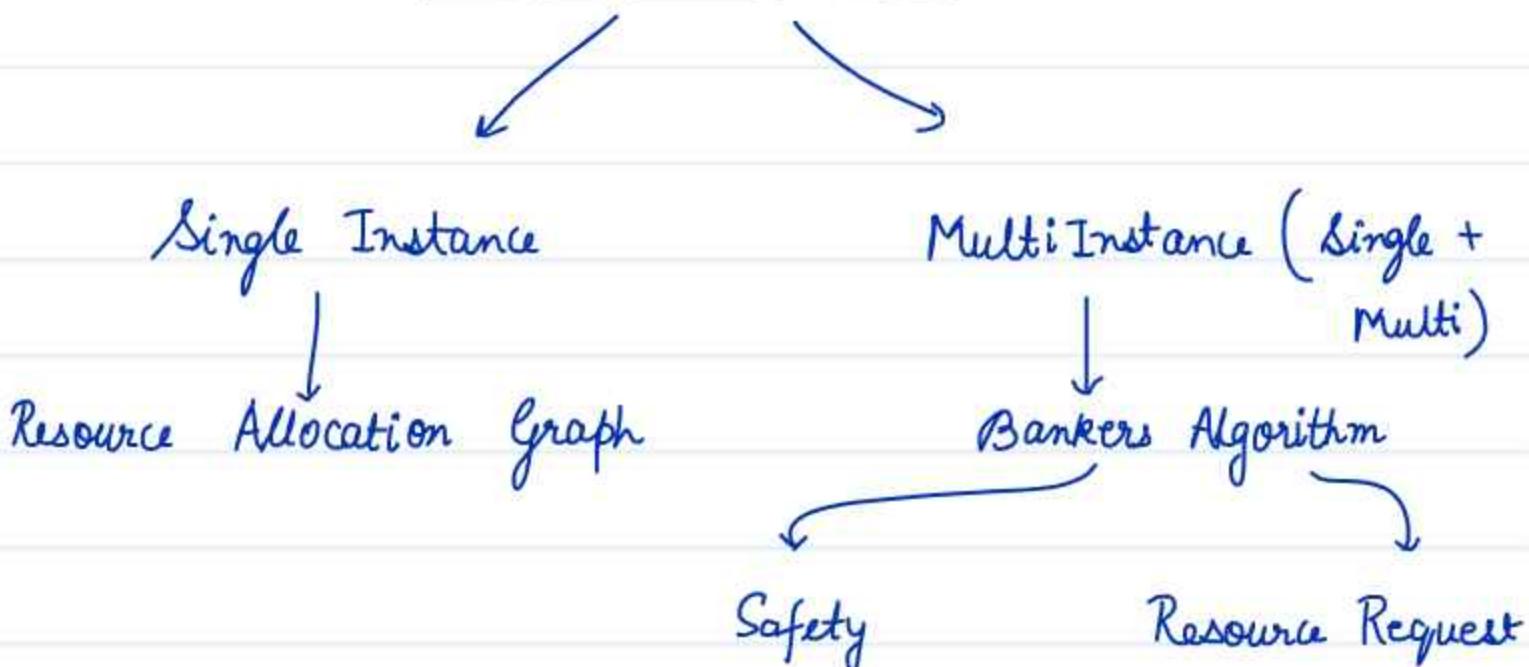
single/ Multi instance :- cycle is Necessary

only Single Instance :- cycle is Necessary

& Sufficient

cycle  $\Rightarrow$  Deadlock

## Deadlock Avoidance



Both Algorithms are based on A priori Knowledge.

### # Resource Allocation graph Algorithm #

→ Single Instance Resource

Every process need to tell which resources she needs beforehand to operating system.

→ grants / deny request based on the resulting system state.

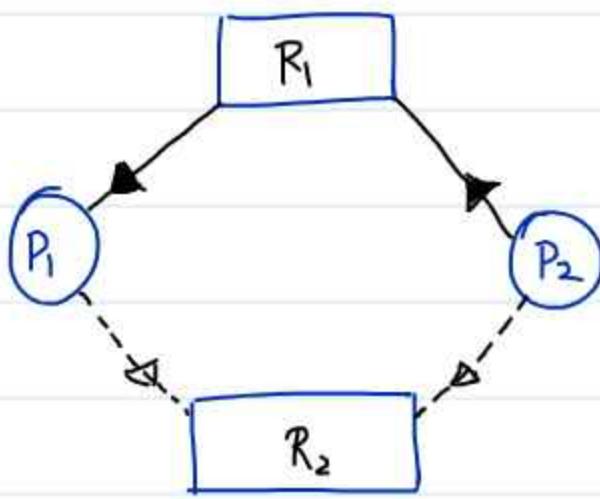
- Resources are claimed a priori in the system
  - If process  $P_i$  starts executing then all claim edges must appear in
  - If  $P_i$  requests resource  $P_j$  then the request is granted only if Resource edge  $\xrightarrow{\text{convert}}$  Assigned edge doesn't lead to a cycle in RAG<sub>i</sub>, otherwise the process is blocked.
- |             |                              |
|-------------|------------------------------|
| safe        | unsafe                       |
| ↓           | ↓                            |
| No Deadlock | Warning / likelihood of D.L. |

In RAG, no cycle  $\longrightarrow$  safe state  
 cycle  $\longrightarrow$  unsafe state

The base objective of RAG is to always operate the system into safe state.

If Req. Edge  $\xrightarrow{\text{conversion}}$  Assigned Edge  
 doesn't lead to a cycle in RAG.

Q



• current state?

$\hookrightarrow$  safe

• When will system gets into unsafe state?

$\hookrightarrow$  If R<sub>2</sub> is granted to P<sub>2</sub>

• When will Deadlock happen?

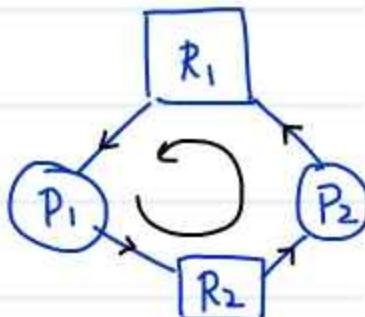
$\hookrightarrow$  When P<sub>1</sub> also request for R<sub>2</sub>

P<sub>1</sub> holds R<sub>1</sub>, may req. for R<sub>2</sub>.

P<sub>1</sub> requests R<sub>1</sub>, may req. for R<sub>2</sub>.

NOTE:- Claim edge is also part of the graph.

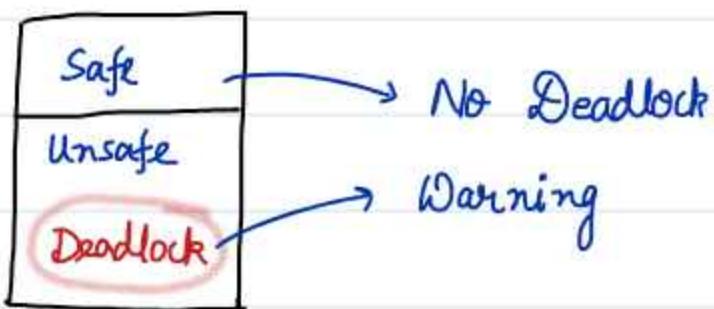
Deadlock {



Safe  $\rightarrow$  unsafe  $\rightarrow$  Deadlock

RAGA operates to keep system in safe mode.

In the above case, the claim → Request ✓  
 → Assigned X



System state

\* \* \* BANKER'S ALGORITHM \* \* \*

(Avoidance)

(Multi-instance)

→ Safety Algorithm  
 → Resource Request Algorithm

1  $n$ : no of Processes ( $P_1, P_2 \dots P_n$ )

2  $m$ : no of Resources ( $R_1, R_2 \dots R_m$ )

3 Maximum  $[1 \dots n, 1 \dots m]_{n \times m}$

$$\underbrace{\max[i, j]}_{\downarrow} = k$$

Process  $P_i$  demands (apriori) 'k' copies of  $R_j$

④ Allocation  $[1 \dots n, 1 \dots m]_{n \times m}$

$$\text{Alloc}[i, j] = a$$

$$a \leq k$$

allocation

demand

⑤ Need  $[1 \dots n, 1 \dots m]_{n \times m}$

$$\text{Need}[i, j] = b$$

$$\text{Need} = \text{Demand} - \text{Alloc}$$

$$b = k - a$$

⑥ Request  $[1 \dots n, 1 \dots m]_{n \times m}$

$$c \leq b$$

$$\text{Req}[i, j] = c \quad @ \text{time } t.$$

Request made by process  $i$  at time  $t \leq \text{Need}$

⑦ Total  $[1 \dots m]$        $\text{Total}[j] = z$

There are  $z$  copies of  $R_j$  in the system.

⑧ Available  $[1 \dots m]$        $\text{Avail}[j] = e$

There are  $e(R_j)$  free available @ time 't'.

$$\text{Total} = 100$$

$$\text{Allocated to Processes} = 70$$

$$\text{Available} = 30$$

$e \in \mathfrak{F}$

$$\text{Available} = \text{Total} - \sum_{i=1}^n \text{Alloc}_i$$

The state of the system defined by all these params at any time  $\rightarrow$  either safe / unsafe.

$$n = 5 \quad m = 1 \quad R = 21 \text{ total}$$

Pid	Max(R)	Alloc(R)	Need(R)	Avail(R)
P <sub>1</sub>	10	5	5	21 - 19 = 2
P <sub>2</sub>	8	4	4	
P <sub>3</sub>	12	5	7	In this case the
P <sub>4</sub>	5	2	3	system is unsafe
P <sub>5</sub>	6	3	3	
		19		

Safety Algo :- System is said to be safe if the need of all process can be satisfied with the available resources, otherwise unsafe.

$$\text{Available} = 2 \rightarrow \text{Unsafe}$$

If Available

Initially :- 22

Pid	Max(R)	Alloc(R)	Need(R)	Avail(R)	safe seq
P <sub>1</sub>	10	5	5	3 0	< P <sub>4</sub>
P <sub>2</sub>	8	4	4	2	P <sub>5</sub>
P <sub>3</sub>	12	5	7	1	P <sub>3</sub>
P <sub>4</sub>	5	2	3	9	P <sub>2</sub>
P <sub>5</sub>	6	3	3	12	P <sub>5</sub> >

Finally 22

There can be multiple safe sequences. All processes must be satisfied

# DEADLOCK : PART-3

## Bankers Algorithm

Total:  $\langle 10, 5, 7 \rangle$

$T_D$	Allocation			Max			Available			Need		
	A	B	C	A	B	C	A	B	C	A	B	C
P0	0	1	0	7	5	3	3	3	2	7	4	3
P1	2	0	0	3	2	2	5	3	2	1	2	2
P2	3	0	2	9	0	2	7	4	3	6	0	0
P3	2	1	1	2	2	2	7	4	5	0	1	1
P4	0	0	2	4	3	3	10	4	7	4	3	1

$\langle 10, 5, 7 \rangle$

Available = Total - Allocated

Seq.:-

single entry  
for each resource

Need = Max - Allocated

matrix

got all my resources back.

safe sequence

P<sub>1</sub>  
P<sub>3</sub>  
P<sub>4</sub>  
P<sub>2</sub>  
P<sub>0</sub>

(ii) Resource Request Algorithm ( $P_i$ ,  $Req_i$ ,  $Alloc_i$ ,  $Need_i$ ,  $Avail$ )

- { 1.  $Req_i \leq Need_i$
- 2.  $Req_i \leq Avail$
- 3. [ Assume to have satisfied the  $Req_i$  ]

$$a) \text{Avail} = \text{Avail} - \text{Req}_i$$

$$b) \text{Need}_i = \text{Need}_i - \text{Req}_i$$

$$c) \text{Alloc}_i = \text{Alloc}_i + \text{Req}_i$$

4. Run Safety Algorithm

5. If system is safe then grant the req<sub>i</sub>  
else deny Req<sub>i</sub> & block the process P<sub>i</sub>

}

Ex:- T<sub>1</sub>: P<sub>1</sub> → Req<sub>1</sub> <1, 0, 2>

T <sub>0</sub>	Allocation			Max			Available			Need		
	A	B	C	A	B	C	A	B	C	A	B	C
P <sub>0</sub>	0	1	0	7	5	3	3	3	2	7	4	3
P <sub>1</sub>	2	0	0	3	2	2				1	2	2
P <sub>2</sub>	3	0	2	9	0	2				6	0	0
P <sub>3</sub>	2	1	1	2	2	2				0	1	1
P <sub>4</sub>	0	0	2	4	3	3				4	3	1

$$1). \text{Req}_i \leq \text{Need}_i \quad <1, 0, 2> \leq <1, 2, 2>$$

$$2). \text{Req}_i \leq \text{Avail} \quad <1, 0, 2> \leq <3, 3, 2>$$

3). Assume req<sub>1</sub> to be satisfied

$$a. \quad <3, 3, 2> \longrightarrow <2, 3, 0> (\text{Avail})$$

- b.  $\langle 1, 2, 2 \rangle \longrightarrow \langle 0, 2, 0 \rangle$  (Need)  
 c.  $\langle 2, 0, 0 \rangle \longrightarrow \langle 3, 0, 2 \rangle$  (Allocation)

4). Run safety Algorithm

$T_0$	Allocation			Max			Available			Need		
	A	B	C	A	B	C	A	B	C	A	B	C
P0	0	1	0	7	5	3	3	3	2	7	4	3
P1	2	0	0	3	2	2	2	3	0	1	2	2
P2	3	0	2	9	0	2	5	3	2	6	0	0
P3	2	1	1	2	2	2	7	4	3	0	1	1
P4	0	0	2	4	3	3	7	4	5	4	3	1

$\langle P_1, P_3, P_4, P_0, P_2 \rangle$



$$\begin{array}{r} 7 \\ 5 \\ 5 \end{array}$$

$$\langle 10, 5, 7 \rangle$$

Safe state

$t_2: P_4 \rightarrow \text{Req}_{P_4} \langle 3, 3, 0 \rangle \quad X$

~~Reg > Need~~

$t_3: P_0 \rightarrow \text{Req}_{P_0} \langle 0, 2, 0 \rangle$

• Request should be granted only if the resulting state is safe.

If granted then by remaining resources you won't be able to satisfy any process's need.  $\Rightarrow$  UNSAFE.

Process	Allocation	Max	Available	Need = Max - Alloc	Safe seq
	A B C D	A B C D	A B C D	A B C D	P <sub>3</sub>
P <sub>0</sub>	0 0 1 2	0 0 1 2	1 5 2 0	0 0 0 0	✓ P <sub>4</sub>
P <sub>1</sub>	1 0 0 0	1 7 5 0	1 1 5 2	0 7 5 0	✓ P <sub>2</sub>
P <sub>2</sub>	1 3 5 4	2 3 5 6	1 1 6 6	1 0 0 2	✓ P <sub>1</sub>
P <sub>3</sub>	0 6 3 2	0 6 5 2	2 1 4 10	0 0 2 0	✓ P <sub>0</sub>
P <sub>4</sub>	0 0 1 4	0 6 5 6	3 14 11 10	0 6 4 2	

$$\text{Allocation} + \text{Avail} = \underline{\underline{3 \ 14 \ 12 \ 12}}$$

↓                    ↓  
 2, 9, 10, 12      1, 5, 2, 0

Alloc  
+ Avail  
Avail +

Q TRY  
 An operating system uses the *Banker's algorithm* for deadlock avoidance when managing the allocation of three resource types X, Y, and Z to three processes P<sub>0</sub>, P<sub>1</sub>, and P<sub>2</sub>. The table given below presents the current system state. Here, the *Allocation* matrix shows the current number of resources of each type allocated to each process and the *Max* matrix shows the maximum number of resources of each type required by each process during its execution.

	Allocation			Max		
	X	Y	Z	X	Y	Z
P <sub>0</sub>	0	0	1	8	4	3
P <sub>1</sub>	3	2	0	6	2	0
P <sub>2</sub>	2	1	1	3	3	3

There are 3 units of type X, 2 units of type Y and 2 units of type Z still available. The system is currently in a safe state. Consider the following independent requests for additional resources in the current state:

$$X, Y, Z = \langle 3, 2, 2 \rangle = \text{Avail}$$

REQ1: P<sub>0</sub> requests 0 units of X, 0 units of Y and 2 units of Z  
 REQ2: P<sub>1</sub> requests 2 units of X, 0 units of Y and 0 units of Z

$$P_0 \rightarrow \langle 0, 0, 2 \rangle$$

$$P_1 \rightarrow \langle 2, 0, 0 \rangle$$

	Allocation			Max		
	X	Y	Z	X	Y	Z
P0	0	0	1	8	4	3
P1	3	2	0	6	2	0
P2	2	1	1	3	3	3

$$\text{Need} = \text{Max} - \text{Alloc}$$

$$\begin{matrix} X & Y & Z \\ 8 & 4 & 2 \\ 3 & 0 & 0 \\ 1 & 2 & 2 \end{matrix}$$

$$\begin{matrix} \text{Avail: } & \langle 3, 2, 1 \rangle \\ & 0 \\ 3, 2, 0 \end{matrix}$$

$$P_2 \quad 5 \quad 3 \quad 1$$

$$P_0 \rightarrow \text{Req}_0: \langle 0, 0, 2 \rangle$$

$$\text{Req} \leq \text{Need} \vee P_1 \quad 8 \quad 3 \quad 1$$

$$\text{Req} \leq \text{Avail} \vee P_0 \times$$

Safe state?

→ No

	Allocation			Max		
	X	Y	Z	X	Y	Z
P0	0	0	1	8	4	3
P1	2	2	0	6	2	0
P2	2	1	1	3	3	3

$$\text{Need} = \text{Max} - \text{Alloc}$$

$$\begin{matrix} X & Y & Z \\ 8 & 4 & 2 \\ 3 & 0 & 0 \\ 1 & 2 & 2 \end{matrix}$$

$$\begin{matrix} \text{Available} \\ \langle 3, 2, 2 \rangle \end{matrix}$$

$$P_1 \quad \langle 1, 2, 2 \rangle$$

$$P_1 \rightarrow \langle 2, 0, 0 \rangle$$

$$\text{Req} \leq \text{Available} \quad P_2 \quad \langle 3, 3, 3 \rangle$$

$$\text{Req} \leq \text{Need}$$

Safe state?

$$P_0 \quad \langle 8, 5, 4 \rangle$$

↓  
yes

Finally

# Deadlock Detection & Recovery #

When to Apply

Safe DL

When do you go to Doctor?

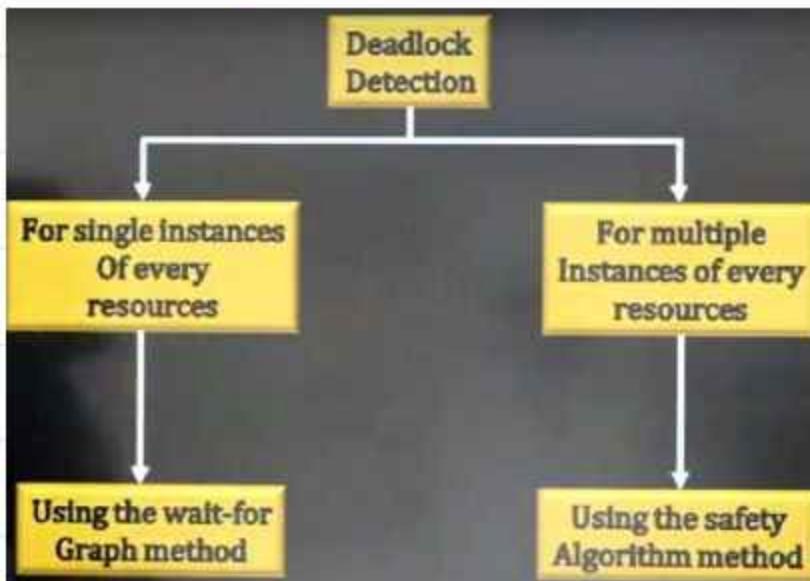
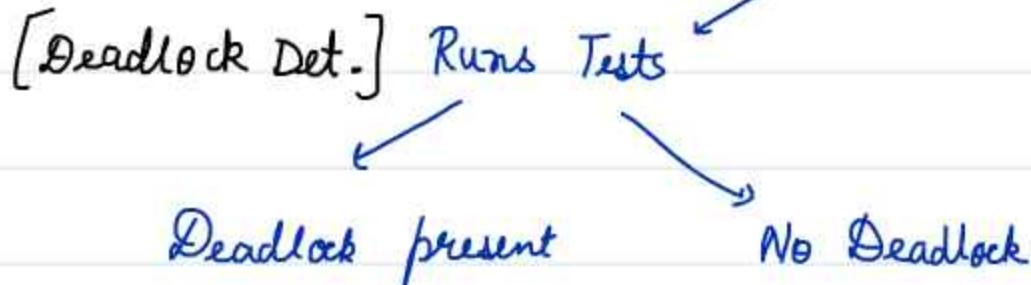
When you are not feeling well.

Based on the symptoms he tells, doctor estimates the problem by tests (diagnosis).

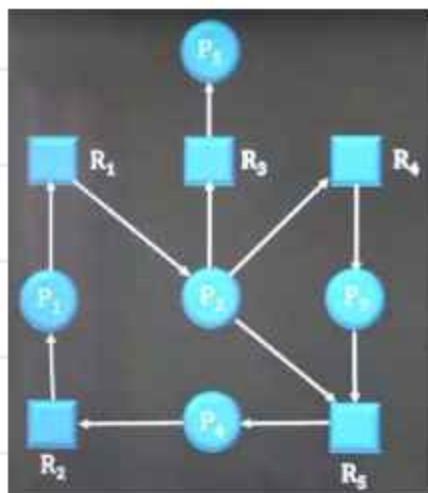
O.S. notices the symptoms like:

- Under utilization of CPU

- Majority processes are getting blocked.

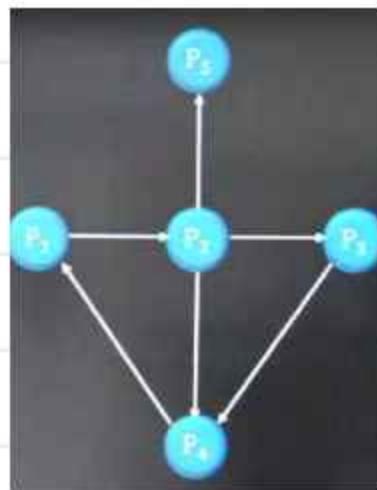


## # Single Instance #



RAG

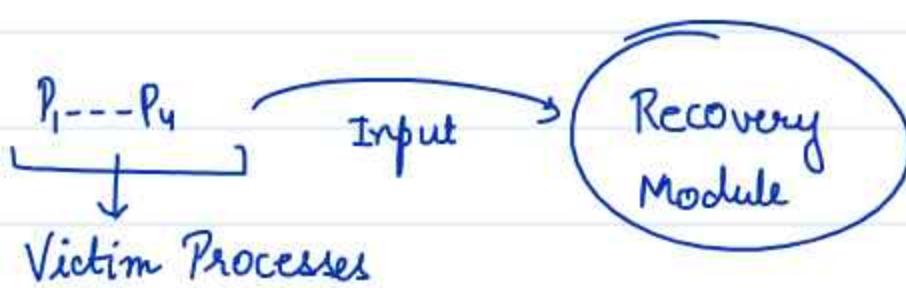
wait for Graph



WFG

(contains only Processes)

After WFG, Run cycle detection algorithm

If single instance :- cycle  $\Rightarrow$  Deadlock

## # Multi Instance #

	Allocation			Request			Available		
	A	B	C	A	B	C	A	B	C
P0	0	1	0	0	0	0	0	0	0
P1	2	0	0	2	0	2			
P2	3	0	3	0	0	0			
P3	2	1	1	1	0	0			
P4	0	0	2	0	0	2			

Total copies &lt;7, 2, 6&gt;

How many processes  
are blocked?

3

P<sub>1</sub>, P<sub>3</sub>, P<sub>4</sub>3  
5

majority blocked

System is safe iff request of all processes is satisfiable with Avail copies in some order.

→ Exactly like safety Algo.

otherwise ~~unsafe~~ deadlock.

	Allocation			Request			Available				
	A	B	C	A	B	C	A	B	C		
P0	0	1	0	0	0	0	0	0	0		
P1	2	0	0	2	0	2	$P_0 \rightarrow P_3$	$P_3 \downarrow$	$P_1 \leftarrow P_4$		
P2	3	0	3	0	0	0					
P3	2	1	1	1	0	0	↙ P2				
P4	0	0	2	0	0	2	↙ P1				

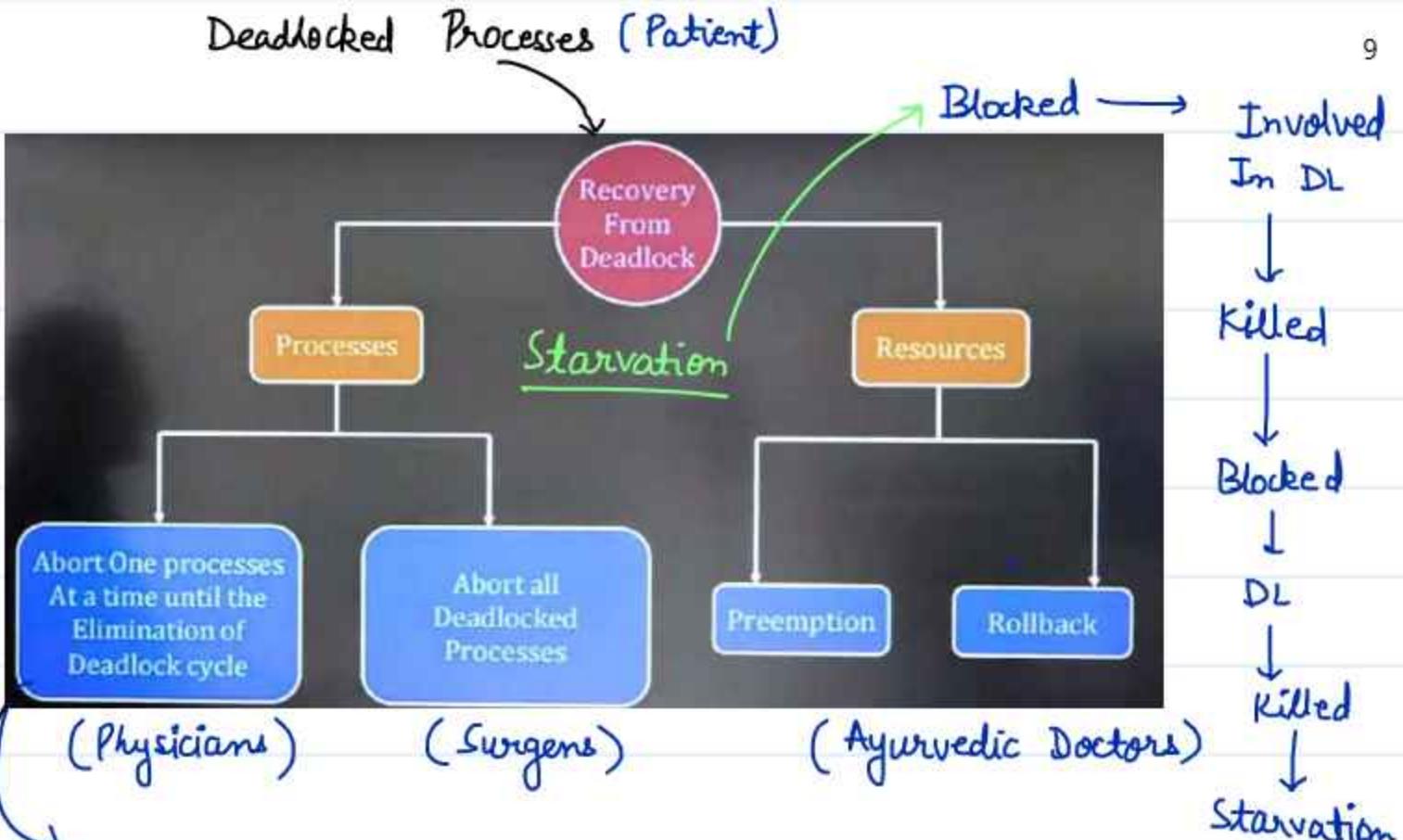
Req. of all processes can be satisfied in some order.  
 ↑  
 Safe state

Even though we got a symptom, but still we are saying no problem because the result of the test is negative.

Now let's say  $P_2$  makes a new Req.  $\langle 0, 0, 1 \rangle$

$\checkmark \underbrace{\langle P_0, P_1, P_2, P_3, P_4 \rangle}_{\text{blocked}} \Rightarrow \text{Deadlock}$

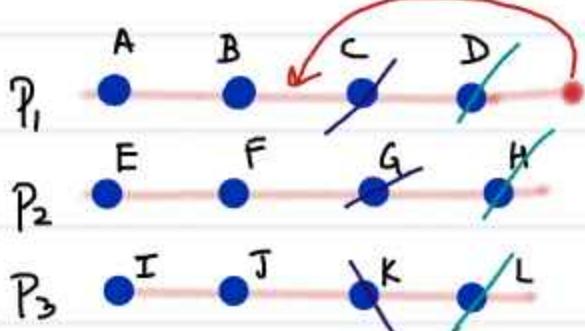
$\langle P_1, \dots, P_4 \rangle$  Deadlock  $\xrightarrow[\text{Input}]{\quad}$  Recovery



Which process to kill? : The one who has just started

Will have to apply detection algo again & again.

- Resource Preemption :- we will keep preempting the latest resources add them to available pool & run detection algo after each preemption, will keep preempting until cycle is broken.



Rollback & Req. again  
 $\langle D, H, L \rangle \xrightarrow{\text{Detection}} \text{yes}$   
 $\langle D, H, L, C, G, K \rangle \xrightarrow{\text{Detection}}$

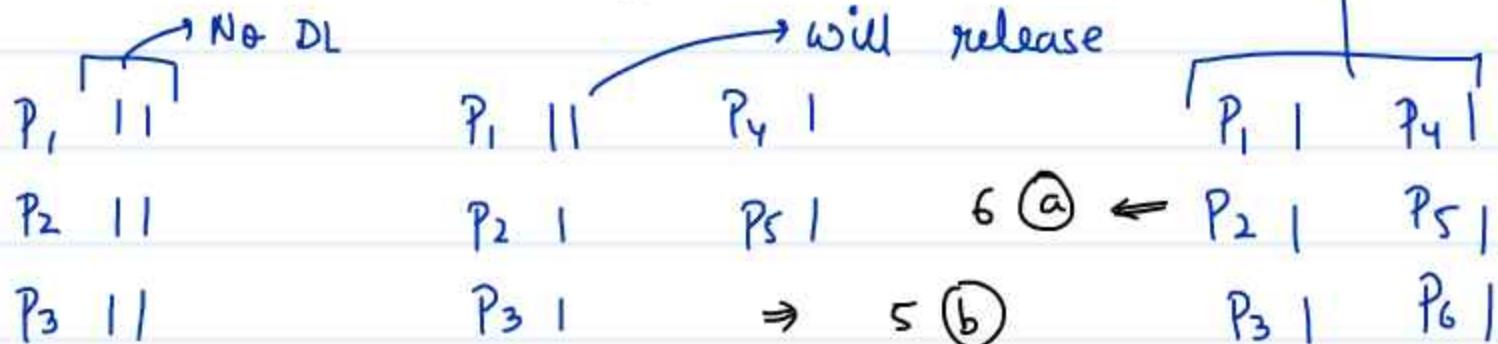


- **Prevention:** This approach is about taking proactive measures to stop problems before they occur. It's a **proactive** attitude because it involves planning ahead and taking action to prevent issues.
- **Avoidance:** This is about steering clear of potential problems. It's a **reactive** attitude because it involves responding to the possibility of issues by avoiding them.
- **Detection & Recovery:** This approach involves recognizing problems when they occur and then recovering from them. It's also a **reactive** attitude because it deals with issues after they've happened.
- **Ignorance:** This is when no action is taken, either to prevent, avoid, or detect and recover from problems. It's an **inactive** attitude because it involves doing nothing in response to potential or actual issues.

$$\underline{Q} \quad \text{Total processes} = n \quad \boxed{\dots} \quad R$$

Each process need two copies of R to complete.

- $\text{Min}(n)$  to cause Deadlock
- $\text{Max}(n)$  for Deadlock freedom



Q $n = 3 \quad (P_1, P_2, P_3)$  $R = ?$  $P_i \rightarrow \alpha(R)$ 

- Max R to cause DL.
- Min R to ensure DL free.

$P_1$	1
$P_2$	1
$P_3$	1

Max = 3 to cause Deadlock

$P_1$	11
$P_2$	1
$P_3$	1

Min = 4 to ensure DL free

Q $n - \text{Process} \quad R = 6 \text{ copies}$  $P_i \rightarrow \beta(R)$ 

$P_1$	11
$P_2$	11
$P_3$	11

Min = 3 to cause deadlock

$P_1$	$P_2$
-------	-------

Max = 2 to ensure DL free

Q

Process

 $P_1$  $P_2$  $P_3$  $P_4$  $P_5$ 

Max Demand

10

8

5

12

6

 $n = 5$  $R$  $\xrightarrow{\quad}$  Max to cause DL $\swarrow$ 

Min

to ensure DL freedom

$P_1$	$P_2$	$P_3$	$P_4$	$P_5$		
9	7	4	11	5	]	$\rightarrow 36 = \text{Max}(R)$

$+1$

$$\xrightarrow{\hspace{1cm}} 37 = \text{Min}(R)$$

$\langle L \dots 36 \rangle \rightarrow \text{DL}$

$\langle 37 \dots \rangle \rightarrow \text{Deadlock free}$

To solve such questions always remember Dining philosopher.

Q

H.W.

→ Safe or not.

Allocation				Request				Available				
	A	B	C	D	A	B	C	D	A	B	C	D
P1	1	0	0	0	0	1	0	0	2	0	0	0
P2	0	1	0	0	0	0	1	0				
P3	0	0	1	0	0	0	0	1				
P4	0	1	0	1	1	0	0	0				
P5	0	0	0	1	0	0	0	0				

Q

Which of the following statements is/are TRUE with respect to deadlocks?

- a) Circular wait is a necessary condition for the formation of deadlock.
- b) In a system where each resource has more than one instance, a cycle in its wait-for graph indicates the presence of a deadlock.
- c) If the current allocation of resources to processes leads the system to unsafe state, then deadlock will necessarily occur.
- d) In the resource-allocation graph of a system, if every edge is an assignment edge, then the system is not in deadlock state.

H.W.

Consider the following snapshot of a system running  $n$  processes. Process  $i$  is holding  $x_i$  instances of a resource  $R$ , for  $1 \leq i \leq n$ . Currently, all instances of  $R$  are occupied. Further, for all  $i$ , process  $i$  has placed a request for an additional  $y_i$  instances while holding the  $x_i$  instances it already has. There are exactly two processes  $p$  and  $q$  such that  $y_p = y_q = 0$ . Which one of the following can serve as a necessary condition to guarantee that the system is not approaching a deadlock?

$$\min(x_p, x_q) < \max_{k \neq p, q} y_k$$

$$x_p + x_q \geq \min_{k \neq p, q} y_k$$

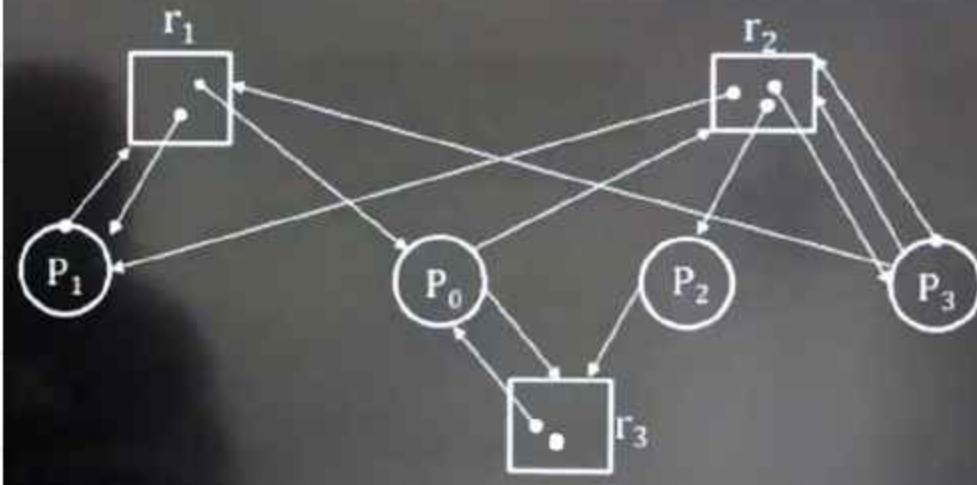
$$\max(x_p, x_q) > 1$$

$$\min(x_p, x_q) > 1$$

MUST TRY

H.W.

Consider the Following Resource Allocation Graph. Find if the System is in Deadlock State.



# MEMORY MANAGEMENT - PART I

R.W

Consider the following snapshot of a system running  $n$  processes.

Process  $i$  is holding  $x_i$  instances of a resource  $R$ , for  $1 \leq i \leq n$ .

Currently, all instances of  $R$  are occupied. Further, for all  $i$ ,

process  $i$  has placed a request for an additional  $y_i$  instances while holding the  $x_i$  instances it already has. There are exactly two processes  $p$  and  $q$  such that  $y_p = y_q = 0$ . Which one of the following can serve as a necessary condition to guarantee that the system is not approaching a deadlock?

- a)  $\min(x_p, x_q) < \max_{k \neq p, q} y_k$
- b)  $x_p + x_q \geq \min_{k \neq p, q} y_k$
- c)  $\max(x_p, x_q) > 1$
- d)  $\min(x_p, x_q) > 1$

MUST TRY

Hold

$$P_i \leftarrow x_i(R)$$

$$i \in [1, n]$$

Req

$$P_i \rightarrow y_i(R)$$

$$y_p = y_q = 0$$

Pid	Alloc	Req	Avail
$P_1$	$x_1$	$y_1$	0
$P_2$	$x_2$	$y_2$	
$P_3$	$x_3$	$y_3$	
⋮	⋮	⋮	
$P_p$	$x_p$	0 ✓	
$P_q$	$x_q$	0 ✓	
⋮	⋮	⋮	
$P_n$	$x_n$	$y_n$	

$x_p \& x_q$  are not requesting more

Not app.

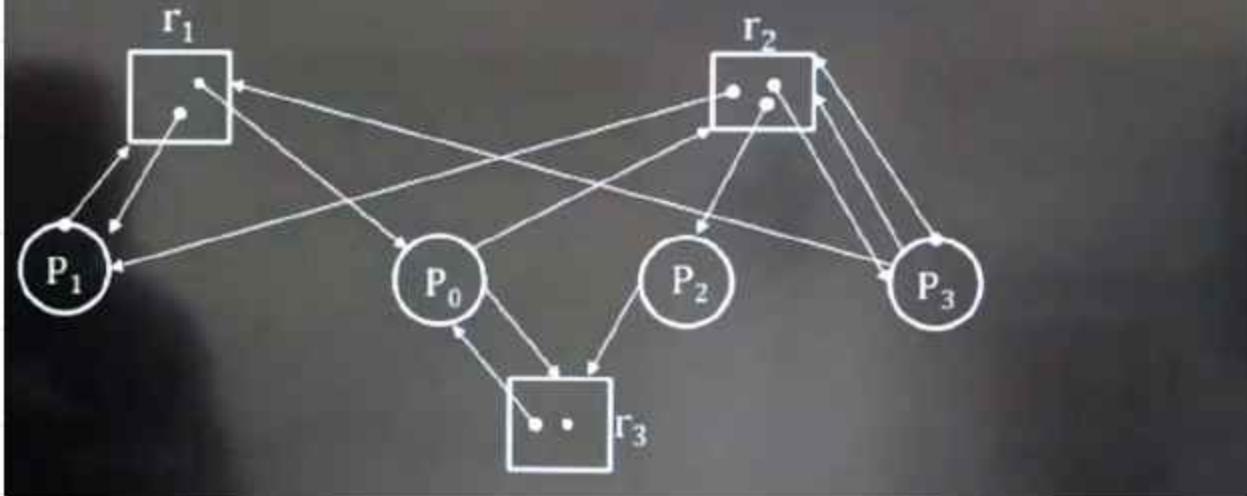
$x_p + x_q \geq \min(y_i)$  Deadlock  
 $i \neq p, q$

$x_p + x_q < \min(y_k) \rightarrow$  Deadlock  
 $k \neq p, q$

$x_p + x_q > \max(y_i) \rightarrow$  Safe

Q

Consider the Following Resource Allocation Graph. Find if the System is in Deadlock State.



which Prevention ~~Avoidance~~ Detection ?

~~single~~

~~Multi~~

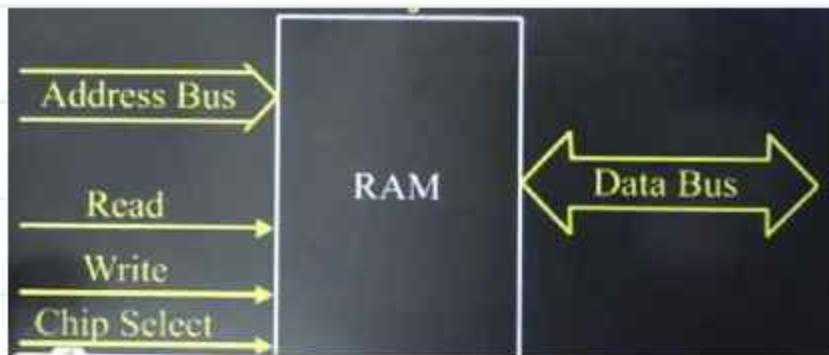
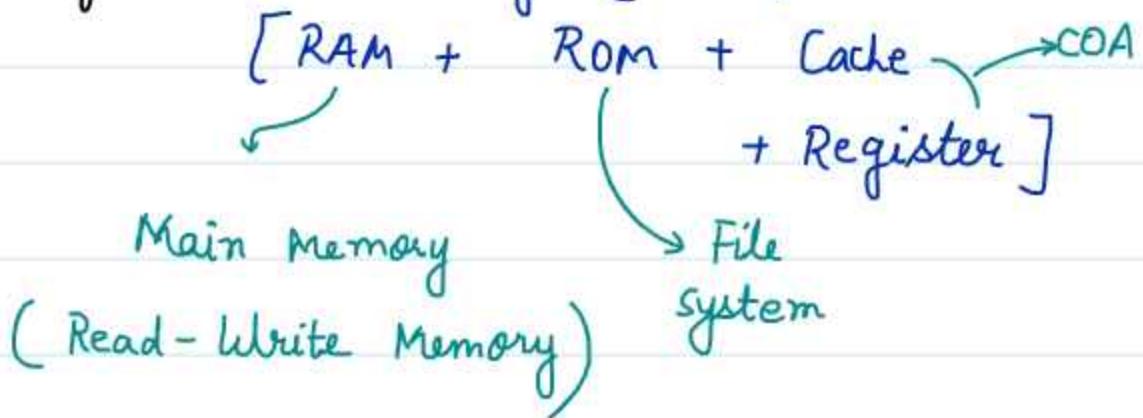
Matrix form

We need to convert this  
RAG → Matrix

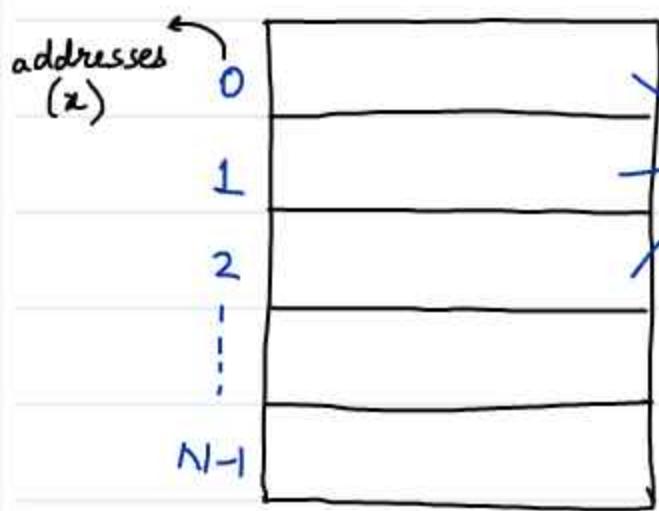
$\langle r_1, r_2, r_3 \rangle$

Pid	Alloc	Req	Avail	
P <sub>0</sub>	1, 0, 1	0, 1, 1	0, 0, 1	
P <sub>1</sub>	1, 1, 0	1, 0, 0	0, 1, 1	
P <sub>2</sub>	0, 1, 0	0, 0, 1	1, 1, 2	{ P <sub>2</sub> P <sub>0</sub> P <sub>1</sub> P <sub>3</sub> }
P <sub>3</sub>	0, 1, 0	1, 2, 0	2, 2, 2	
			2, 3, 2	✓
				Safe

Memory → Primary (Main)



# Linear One Dimensional View of Memory



Words / Locations / cells

$\times$   $m$  bits  $m$

Word

content stored  
Data + Inst.

$m = \text{width of word} / \text{word length}$   
(bits / bytes)

Total no of words =  $N$

$x = \text{Fixed length address (bits)}$

Memory is specified using

$$= \text{No of words} \times \text{Width of word}$$

$(N \times m)$

$n$ : Size of Address in bits

$m$ : Word length

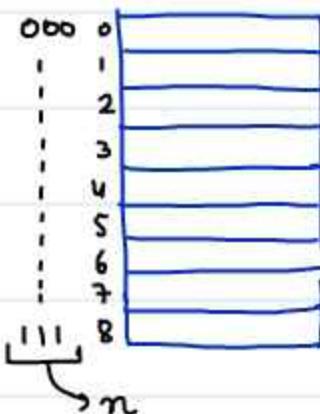
$N$ : Total no. of words

$$\boxed{2^n \text{ words} = N}$$

$$2^{\text{size of address}} = \text{Total no of words}$$

• Relation b/w  $N$  &  $n$  :-

$$N = 8 \text{ words}$$



$$n = 3 \text{ bits}$$

$$10 \text{ bits} \longrightarrow 1 \text{ KB}$$

$$20 \text{ bits} \longrightarrow 1 \text{ MW}$$

$$30 \text{ bits} \longrightarrow 1 \text{ GW}$$

Ex:-  $N = 500 \text{ GB}$        $n = 39 \text{ b}$

$$2^9 = \frac{2^{10}}{2} \quad \xleftarrow[2]{1000}$$

$$N = 1000 \text{ KB} \quad n = 20 \text{ b}$$

$$N = (\log_2 M) B$$

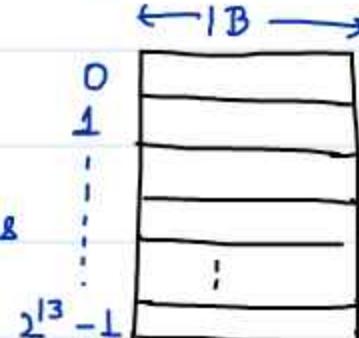
$$2^n = N = \log_2 M$$

$$n = \log_2(\log_2 M)$$

$$n = \log_2(\log_2 \frac{M}{B})$$

• Relation b/w  $N$ ,  $n$  &  $m$  :-

**Ex:-**  $n = 13 \text{ bits}$  } 13 bits are needed to refer 1 word  
 $m = 8 \text{ bits} = 1B$  } of memory  
 $N = 2^{13} = 8KB$  } word length  
 $= 8Kw$  = Total no of words

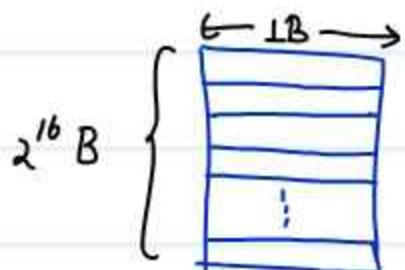


capacity of memory in terms of  $B$  &  $w$ .

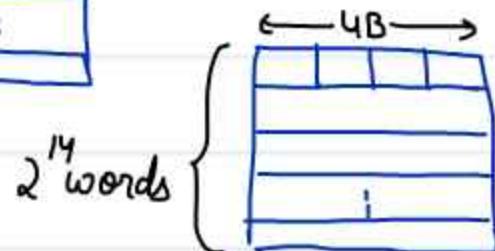
**Ex:-**  $n = 18 \text{ bits}$        $N = 256 Kw \rightarrow \text{word view}$   
 $m = 16 \text{ bits}$        $= 512 KB \rightarrow \text{byte view}$

By default you can take one word as one byte.

**Ex:-**  $N = 64 KB$        $n = \underline{\quad} (w)$   
 $m = 4B$        $\underline{\quad} (B)$



16 bits =  $n$  (byte view)



14 bits =  $n$   
 (word view)

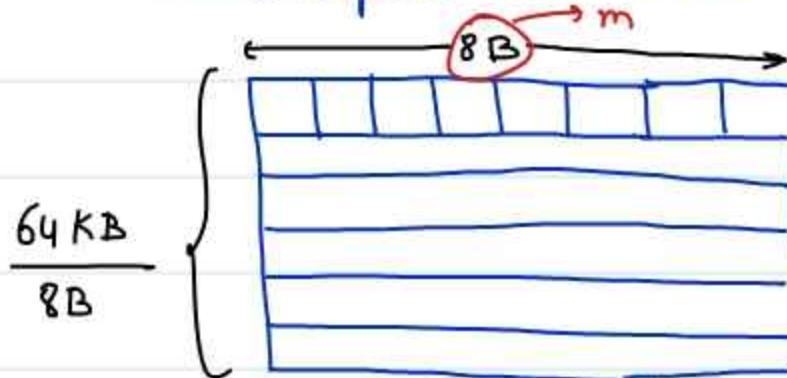
$N \sim \text{words/}^m$   
 Bytes  
 $m \sim \text{bits/}$   
 Bytes  
 $n \sim \text{bits}$

Byte view means one word is 1B

Word view means one word is given no of bytes

$N$  = Total no of words

depends on view



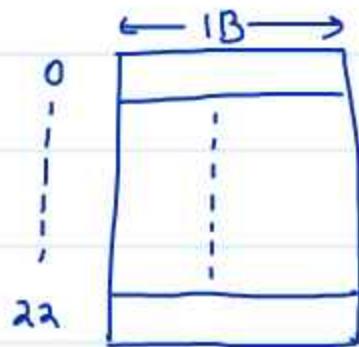
If I say 1 word = 8B

$n$  = 13 bits required to address each word.

$$\frac{2^6 \times 2^{10} \text{ B}}{2^3 \text{ B}} = 2^{13} \text{ words}$$

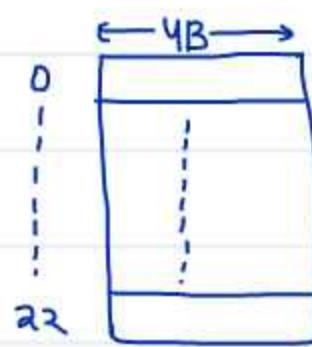
Q  $n = 23$  bits  $N = \underline{\quad} \text{W}$

$m = 32$  bits  $N = \underline{\quad} \text{B}$



$$n = 23 \text{ bits}$$

$$2^{23} \text{ B} = N \text{ (byte)}$$



$$2^{23} \times 2^2 \text{ B} = N \text{ (word)}$$

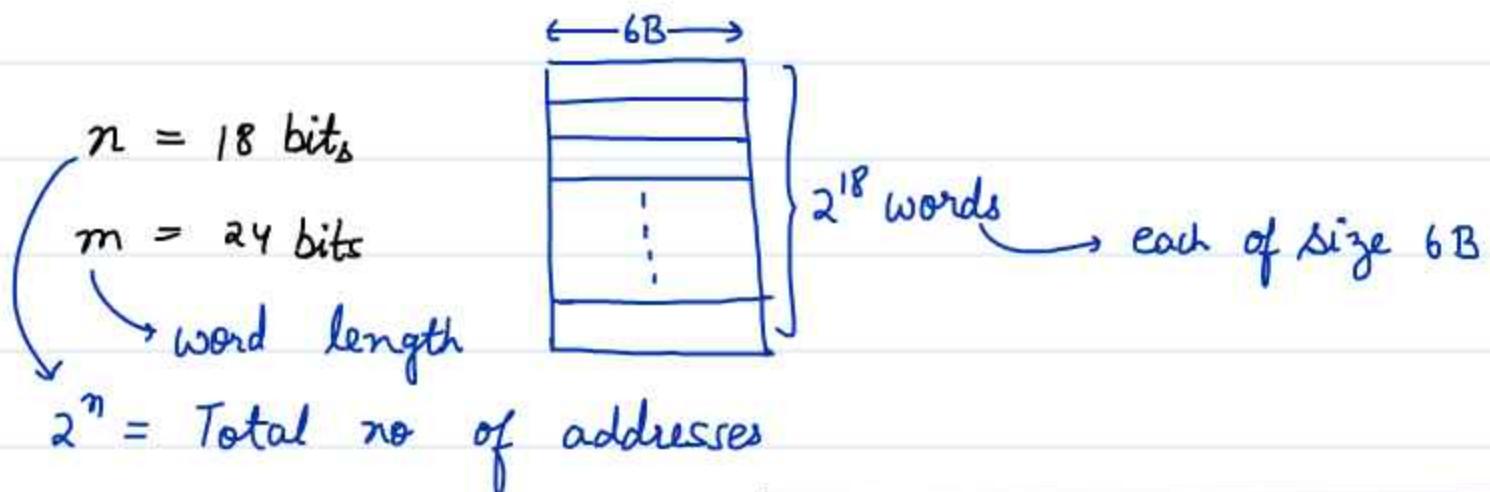
$N = 2^n \text{ W}$   
 $n = \log_2 N \text{ b}$

H.W.

Q  $N = 8G \text{ bit}$   $m = 64 \text{ bit}$   $N = \underline{\quad} \text{W} \& \underline{\quad} \text{B} ?$

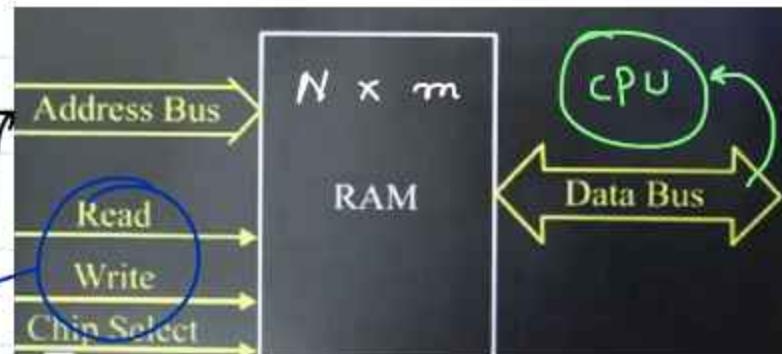
# MEMORY MANAGEMENT - PART II

1



Any memory word you want to refer, pass the address to

control signal  
0 →      1 →



01 :- Write    10 :- Read    11 :- X

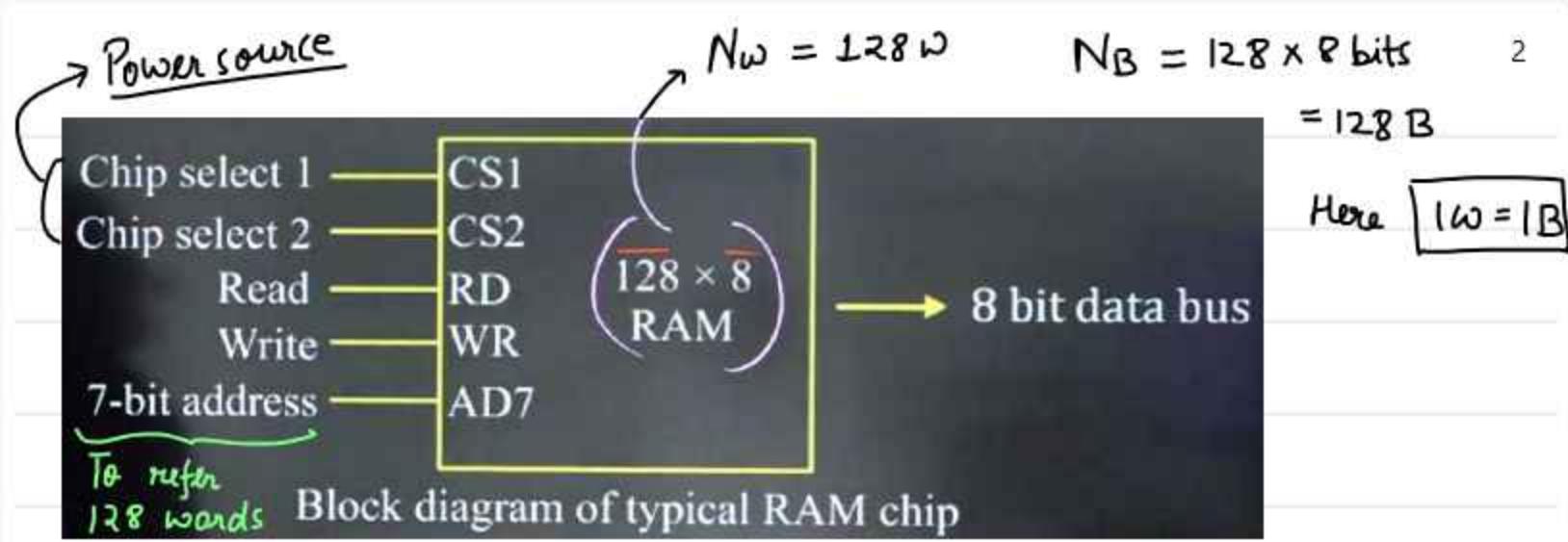
Bus :- Group of wires ( $\perp$  wire can carry  $\perp$  bit)

Read → Ops → Write

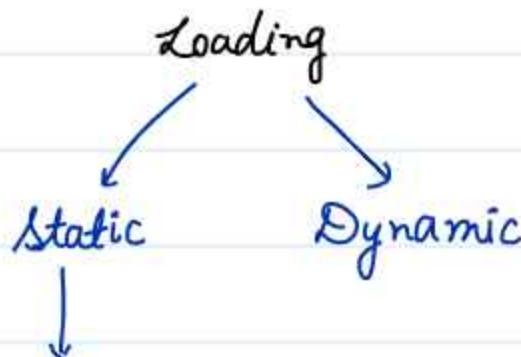
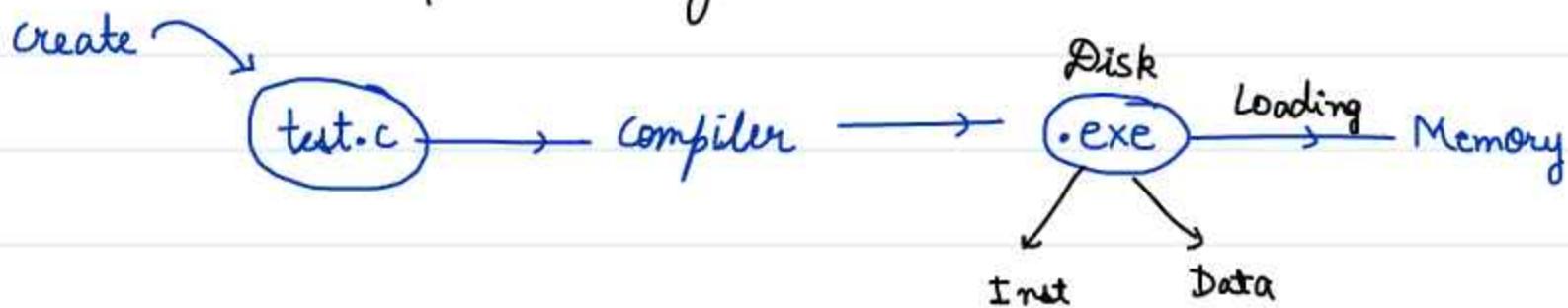
Chip select :- Power on button  
→ should be 1

To perform any op, pass the address to address bus then that word in memory will be activated, pass the control signal either 10 or 01. Then from databus (Read) (Write)

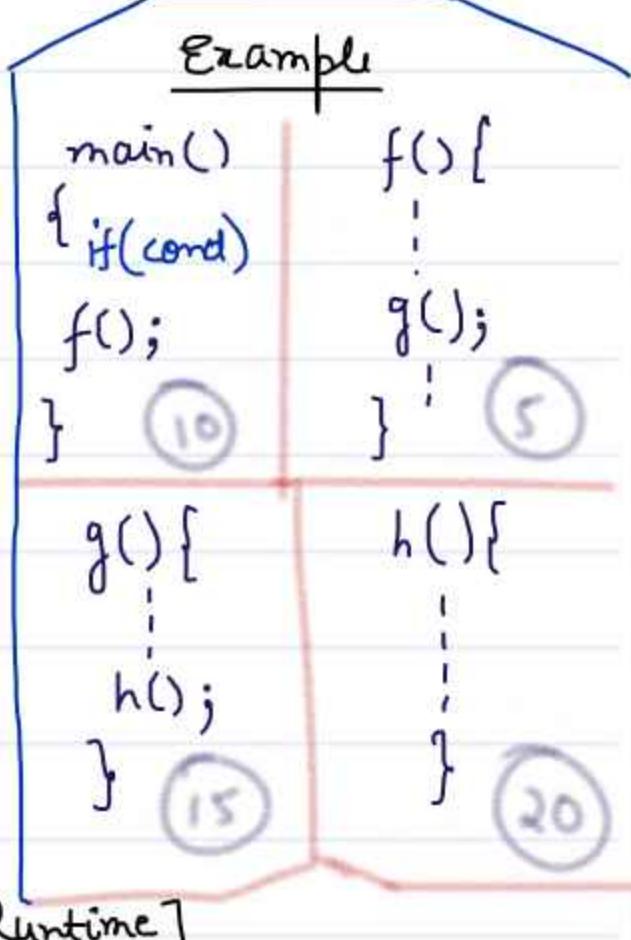
content (data/ Inst.) will reach the CPU.



## # Loading Vs Linking #



→ Drawback :- Possible wastage of space (Ineffective utilization)  
 [Case of : Condition [Known at Runtime]]



→ Advantage :- Faster execution



→ For start

Time efficient , Space inefficient [40 kB req.]

- If want to save memory then go for → DYNAMIC



[10kb req] No wastage of memory ← Loading on Demand  
but

↓ slow execution

For the start

- Linking :- Process of resolving external references

# include < stdio.h >

main()

✓  
✓ {  
✓ |  
compiller → |

f(); → BSA(—)

f()  
{  
|  
}  
|

Compilation → starts  
from first line.

Execution → starts  
from main.

}      scanf();      }  
                → BSA(—) references      → unresolved

- Every function call  
is like a branch/  
goto statement

When the compiler first time confronts  $f()$  it will generate BSA(jump to  $f()$ )

## address

but compiler doesn't know

this so it will leave it

## blank : Unresolved References

extern int x;

1

Present externally in  
some other file

↓

Blank again

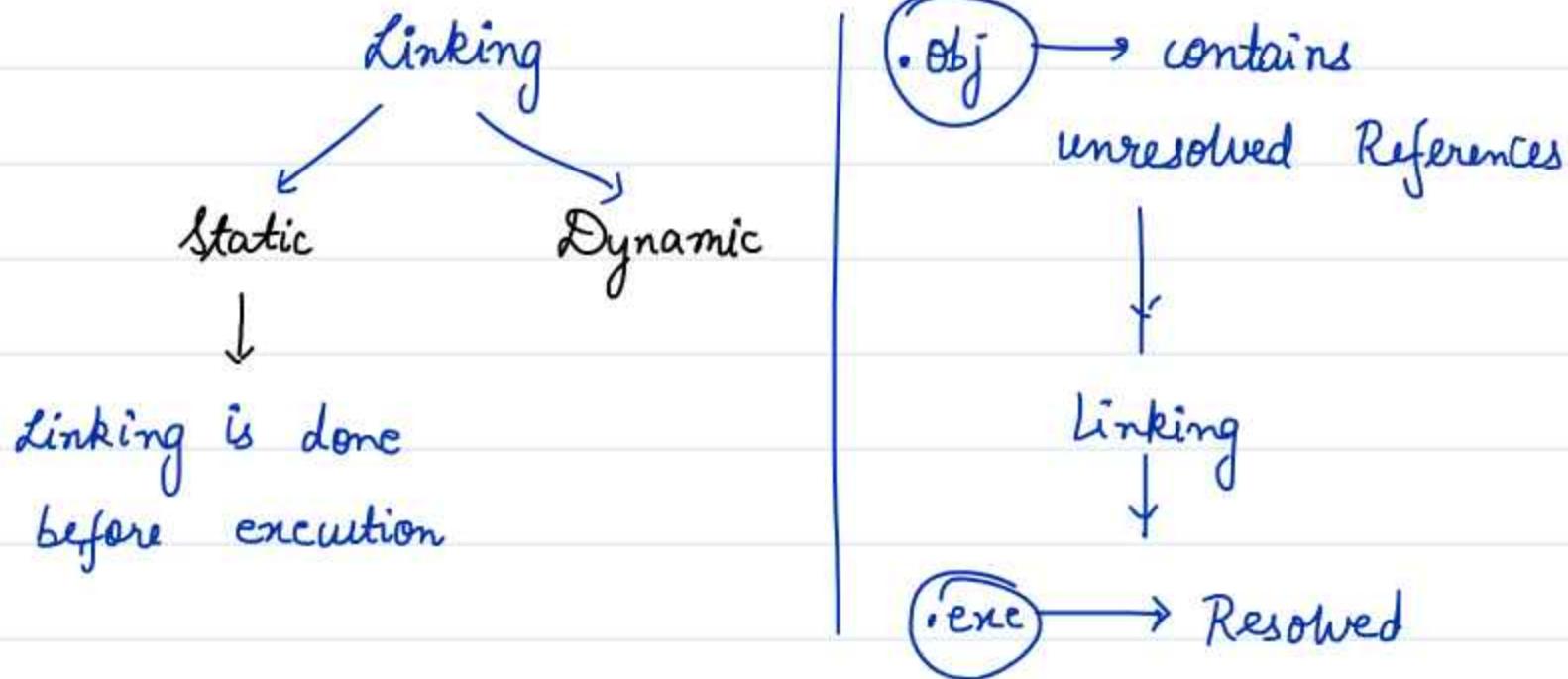
Compiler will leave so many blanks while compiling the code, blanks for  $f$ , external variables / objects

$x: f() \{$

```
main() {
```

$$B \subseteq A(x) \quad f();$$

Module of O.S. who's going to  
fill up these blanks are Linker.

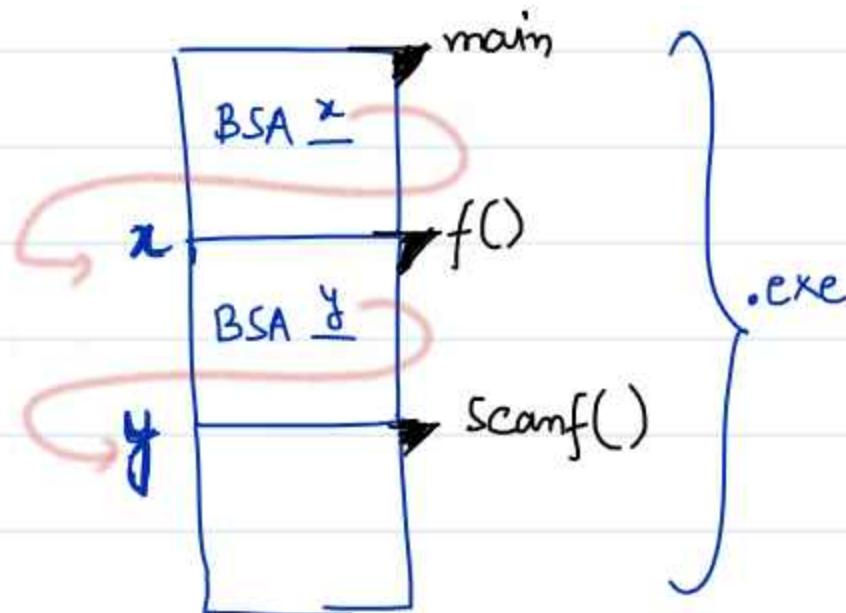


main()

```

{ ;
;
f();
}
    
```

f(){  
    ;  
    scanf();  
}



Drawback :- Space Inefficiency

Linker acts like Tailor



→ stitches (links) different pieces  
of clothes to make a meaningful fabric.

## # Dynamic Linking : Linking @ RT

↓  
Every unresolved reference will be associated

with a small piece of code  $\Rightarrow$  STUB  
(executable)

compilation

linking

will going to activate linker  
at runtime.

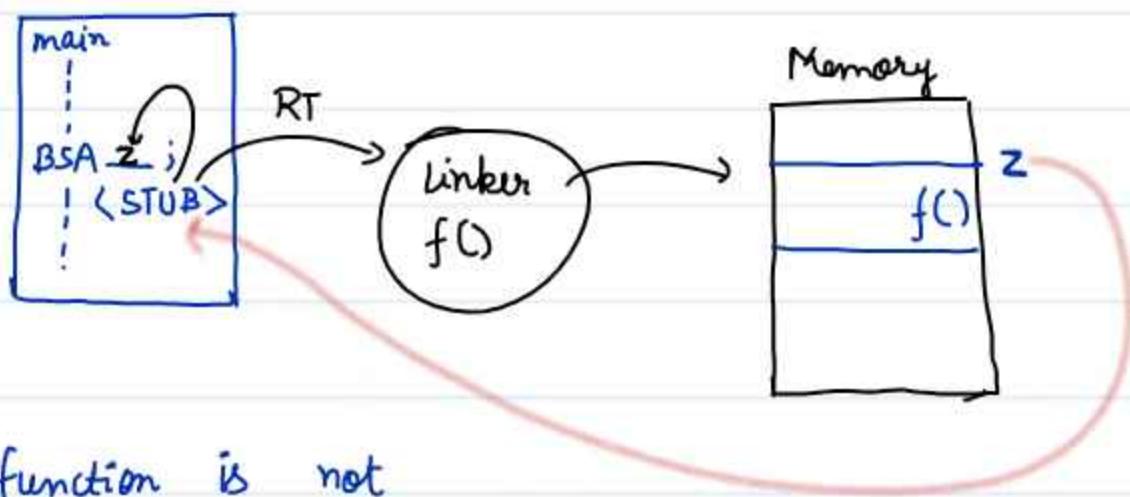
Loading



Execution

linker will find the address &  
will provide it to stub

provide that address to BSA.



If function is not present in memory then linker will look for compiler library on disk. If found then loader will load that module in memory.

All modern system uses :- Dynamic Linking & Dynamic Loading

Modules / Libraries which

are linked @ RT

by Dynamic Linking

are generally called as

**.dll** : dynamic link libraries

Used in Windows

- Space eff + Time inefficient
  - Done during runtime.
  - Code Reusability
- Ex:- `scanf()`, no need to give each function its own copy of `scanf()` unlike static linking.

• Flexibility to change



Modules are separate



we will load a single copy of `scanf` into memory & diff function can call it.



REUSABILITY

They are not linked statically.



Can change the library implementation without affecting the application

library functions

(`scanf`, `printf` etc)

Static library

Dynamic library (.dll)

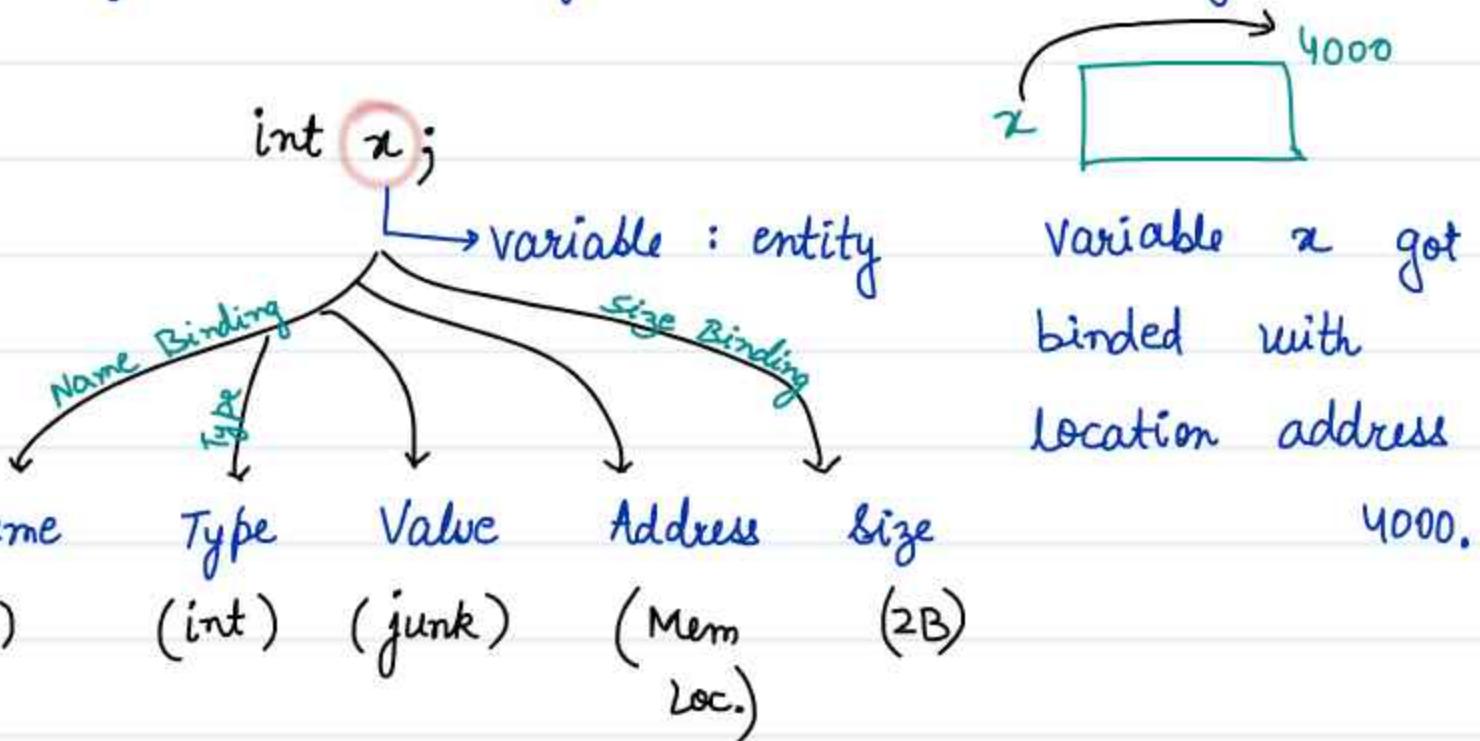
- Drawback :-
1. Time inefficiency
  2. Less security

(In static, the thing is complete, have to do nothing at runtime, no loophole concept)

### # Address Binding #

Association of program instruction & data units to mem. locations (addresses) is address binding.

Binding : Association of attributes with entity



Q The capacity of a memory unit is defined by the number of words multiplied by the number of bits/word. How many separate address and data lines are needed for a memory of 4K x 16?

GATE

PYQ.

- a. 10 address, 16 data lines
- b. 11 address, 8 data lines
- c.  12 address, 16 data lines
- d. 12 address, 12 data lines

$$\begin{matrix} N \times m \\ \text{no of words} \end{matrix} \rightarrow \text{bits/words}$$

→ Simplest  
Problem

$$\begin{matrix} 2^R \times 16 \\ \text{no of words} \end{matrix} \rightarrow \text{word length}$$

$$\text{Data Lines} = \text{word length}$$

$$\text{Address} = 12$$

$$= 16$$

# MEMORY MANAGEMENT - III

## # Address Binding (A.B.)

- Job of linker is to find addresses of these external objects /  $f^n$  / variables.
- A.B.: - Association of program instructions & data units to memory locations (Addresses)

int a, b, c;

$a = 1;$        $\Rightarrow I_1 : \text{Store } a, \#1$

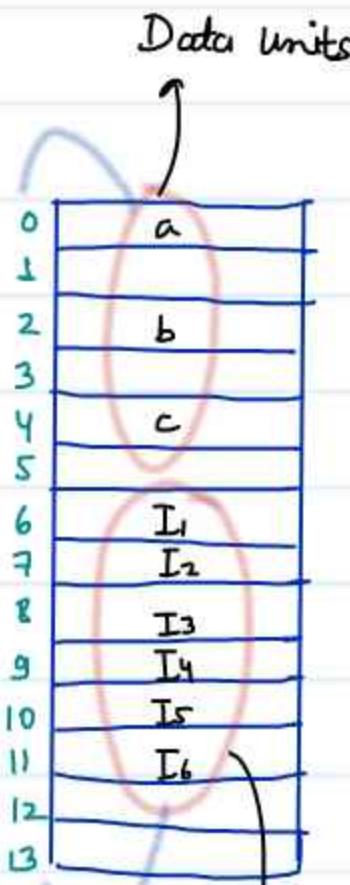
$b = 2;$        $I_2 : \text{Store } b, \#2$

$c = a + b;$      $\Rightarrow I_3 : \text{Load } R_1, a$

$I_4 : \text{Load } R_2, b$

$I_5 : \text{Add } R_1, R_2$

$I_6 : \text{Store } C, R_1$



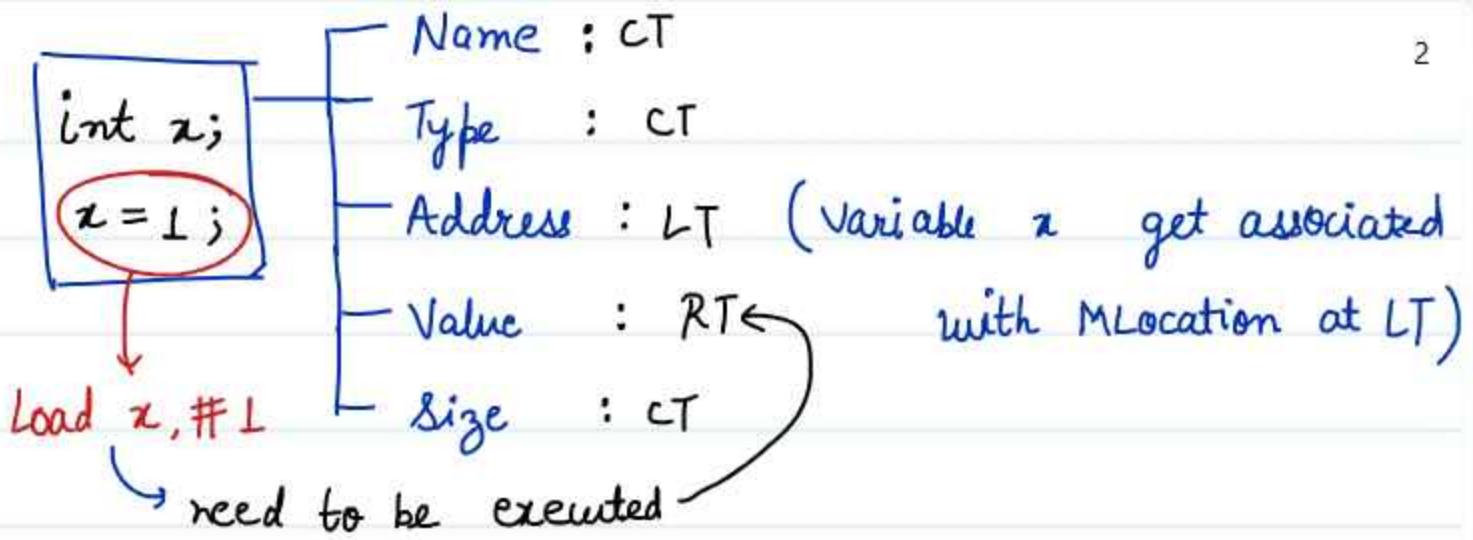
Binding Instructions & data units

to memory locations / addresses.

Prog. Instruction

Binding Time :- Time at which binding takes place.

F → CT, LT, RT



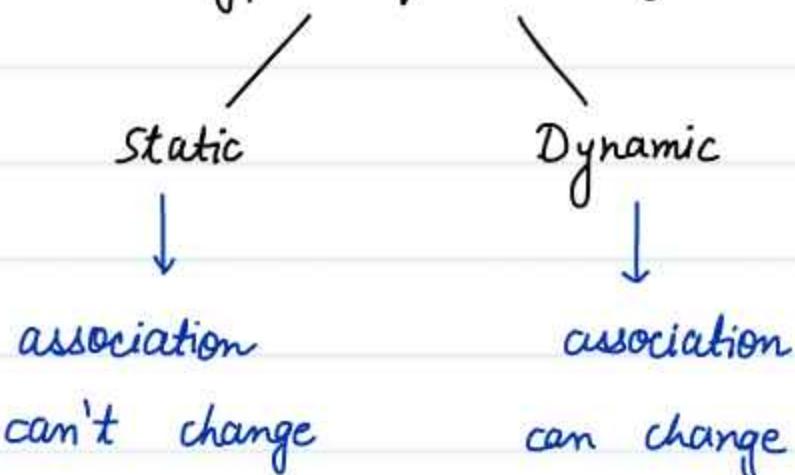
## Types of Binding

Ex:-

Entity : Person

Name Binding : Static

Age Binding : Dynamic



Name	Type	Address	Value	Size
Static	Static	static	Dynamic	Static

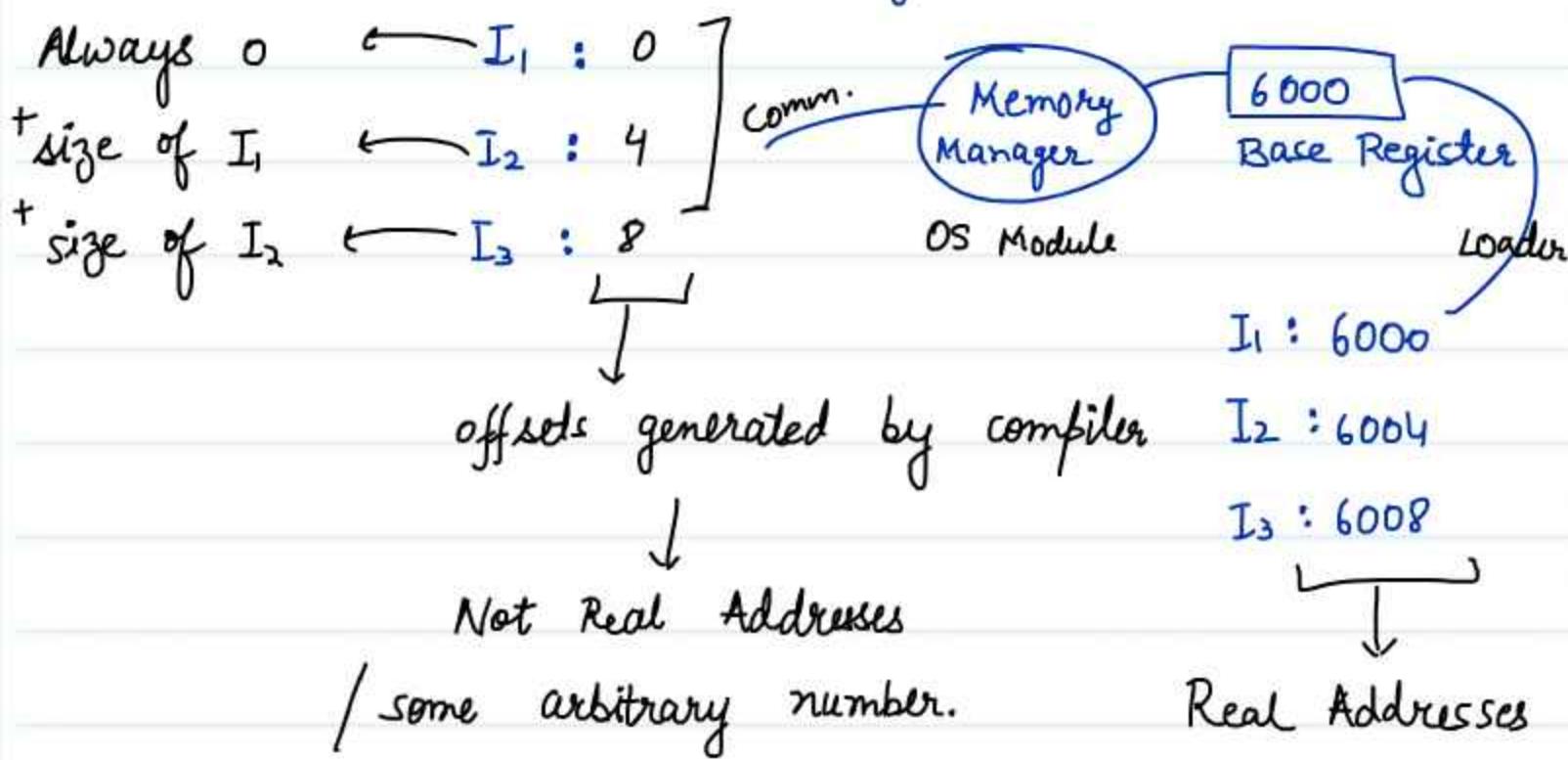
## # Types of Address Binding #

a) CT.

- If addresses of Data units & Instructions are decided by the compiler then it's Compile Time Binding

→ Purely static

b) LT :- Compiler will not generate / associate the addresses , loader will generate .

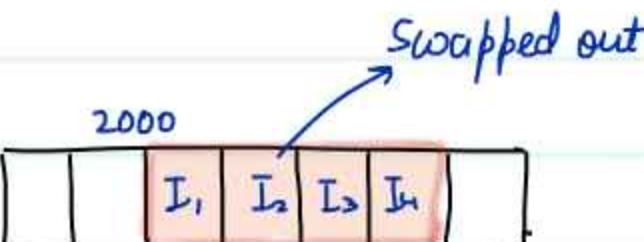


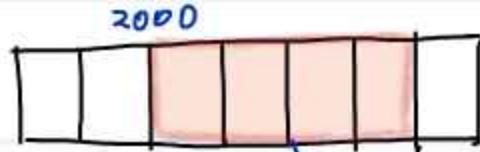
CT :- compiler associates direct addresses.

LT :- Loader associates direct addresses.

Both are static

During execution, the instruction & data unit addresses don't change.



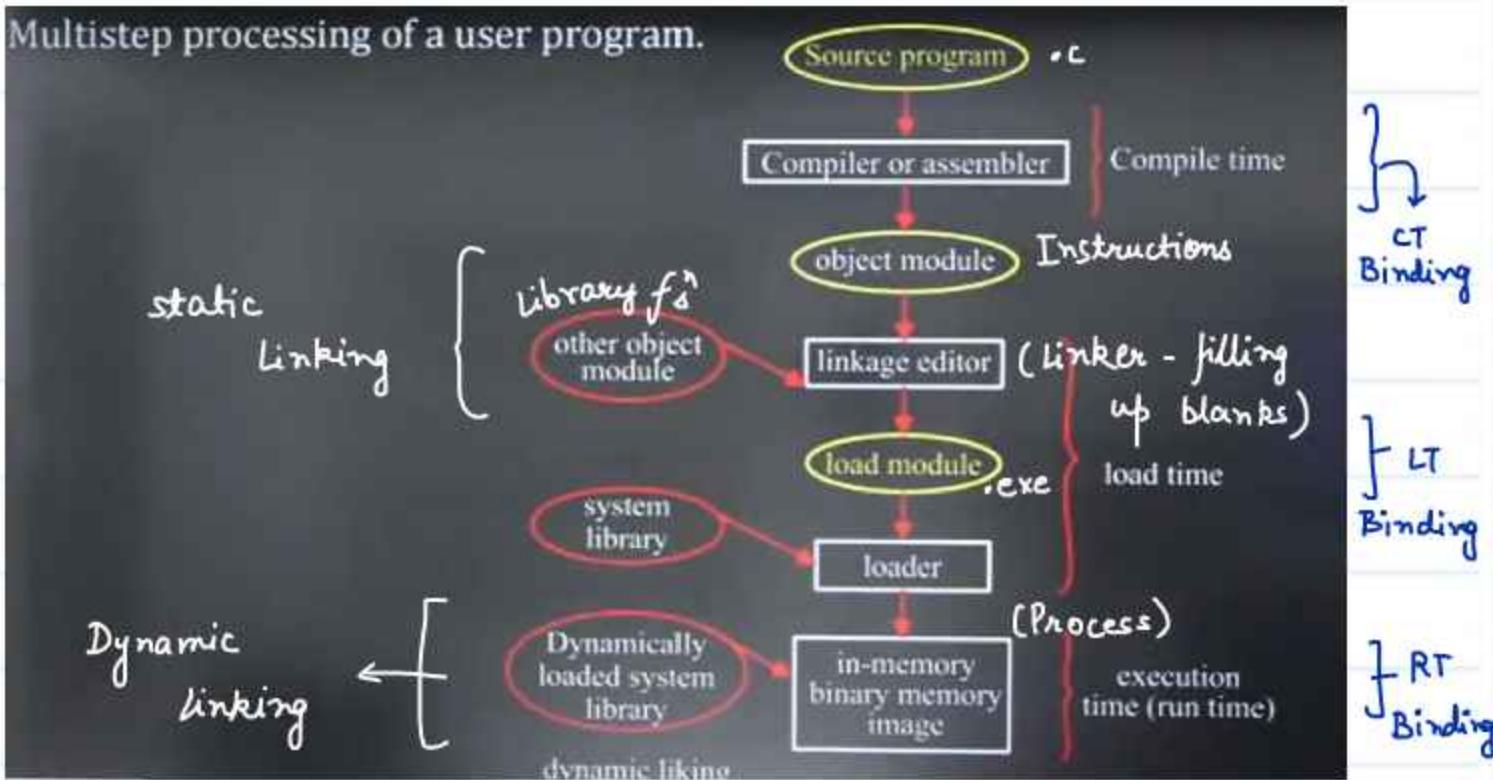


Static :- Addresses can't change  
 → will remain empty  
 when swapped in will be assigned same addresses.

c) Run Time Address Binding :- Done by Loader itself but , when swapped in can be loaded to different place. → Addresses changed during execution.

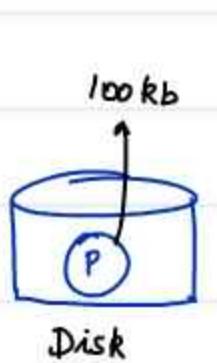
### Relocation Flexibility

Multistep processing of a user program.

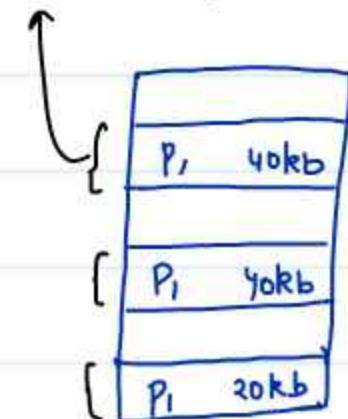


# Memory Management Techniques

Contiguous  
< Centralized >



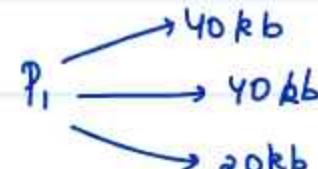
Non Contiguous  
< Distributed >



Whole program  
at one place

- Overlays
- Partitions
  - Variable
  - Fixed
- Buddy Systems

Old Techniques



- Paging
- Segmentation
- Segmented Paging
- Demand Paging

MM started fundamentally from these.

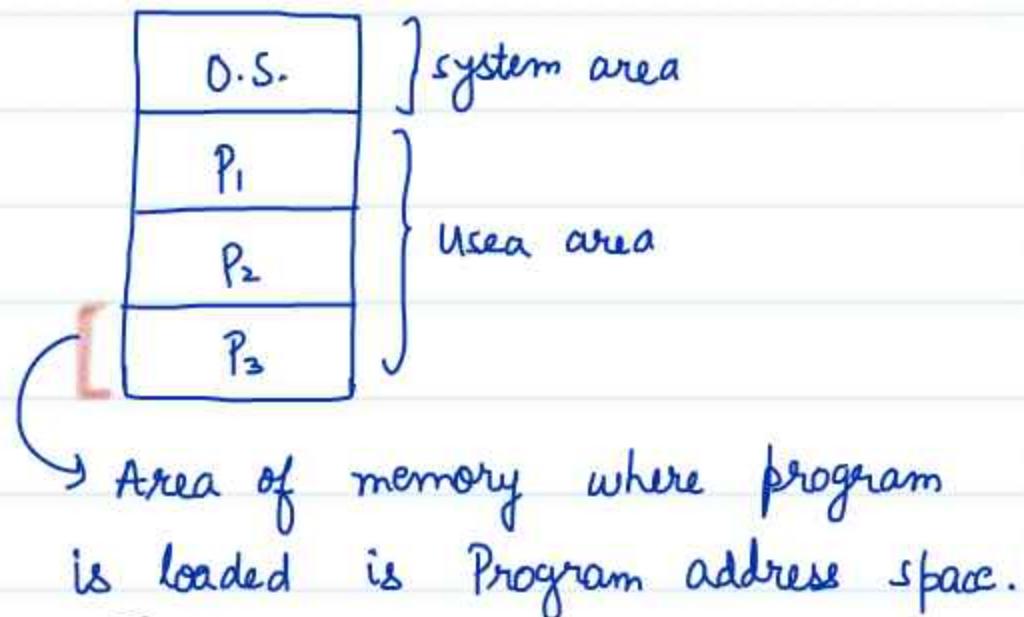
# Functions & Goals of Mem. Manager #

1. Allocation :- Before loading the process you have

to first allocate the memory.

6

2. Protection :-



Should get's its I/O  $\leftarrow$  [P<sub>3</sub>'s address space.  
executed in its own address space only, should not  
interfere with P<sub>2</sub>. NO TRESPASSING.

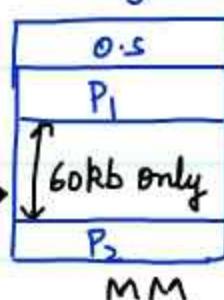
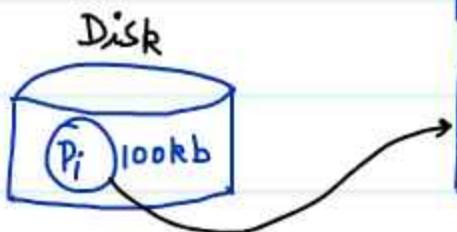
3. Free space Management

4. Address Translation

5. De-allocation.

Goals

- Minimize Wasteage (Utilize effectively)
  - Fragmentation
- Ability to manage larger program in small Memory areas



By stored program concept you need 100 kb in memory.

possible by V.M & overlays.

7



This means O.S. can load more prog. in Mem.



Degree of Multiprogramming Increases



Efficiency & Throughput Increases

## # Overlays #

2-Pass Assembler  
2 stage

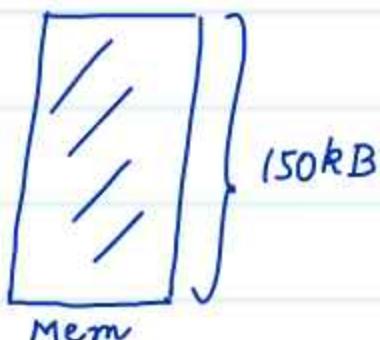
Pass-1 → 70 kB

Pass-2 → 80 kB

NOTE:- Pass-1 & Pass-2 are independent. But we need (at a time) ← {

Symbol Table	→ 30 kB
Common Routines	→ 20 kB
Overlay Loader	→ <u>10 kB</u>
<u>210 kB</u>	

To assembly any program using this assembler you need 210 kB mem.

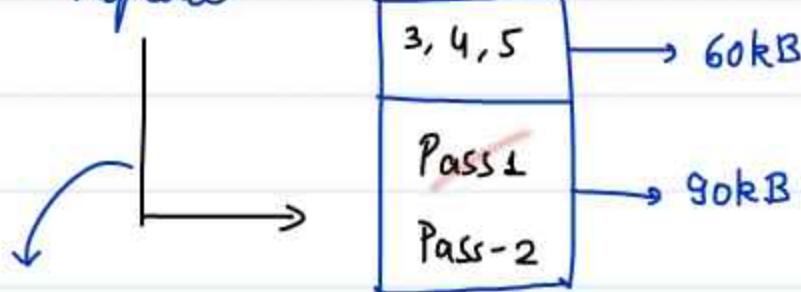


Main Idea :- Complete Pass-1 first then Replace it with Pass-2

as both are Independent.

8

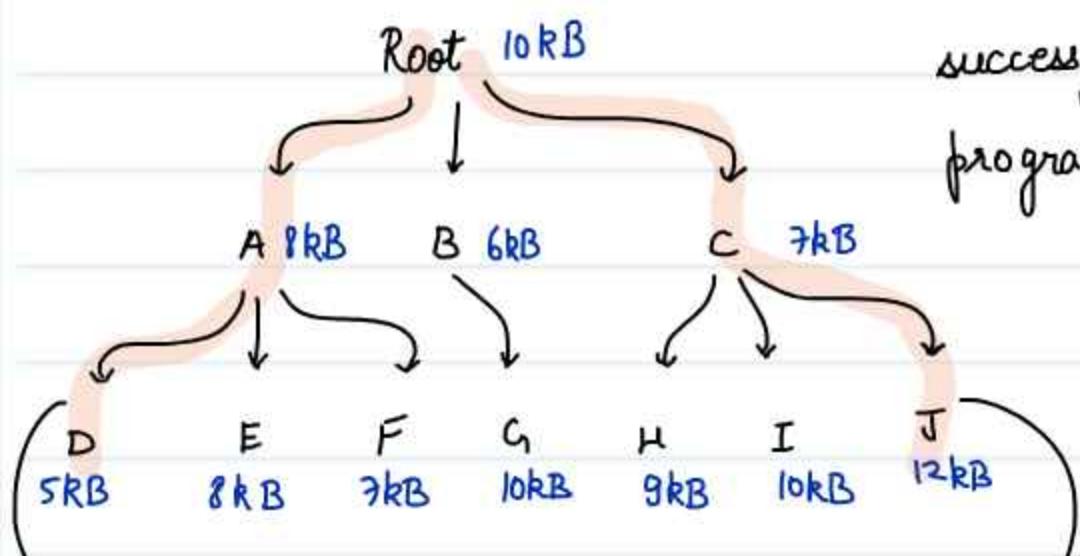
Overlay :- Replace



possible only if program is divisible into Independent modules.

Q. Program expressed as overlay tree  
↳ Size = 92 kB

Min memory required to successfully create this program using overlays?



$$\text{Min Memory Req} = \max(\text{Pass-1 to Pass-7})$$

Pass-1

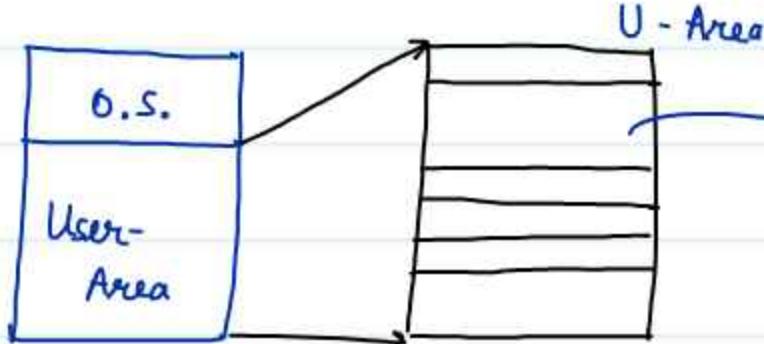
Pass 7

= 29 kB

If you have 29 kB then you can execute any pass.

# # Partitioning #

→ Fixed Partition :- (MFT)



Multiprogramming with fixed tasks.

will be divided into fixed number of partitions

1 Part = 1 Prog

Size of partitions can be diff as program sizes are. | may be of diff. sizes.      1 : 1 Relation

Depends on Architecture

Limit Registers

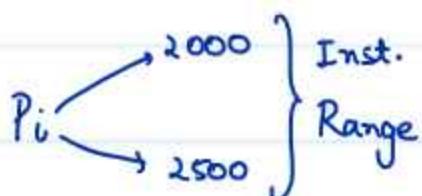
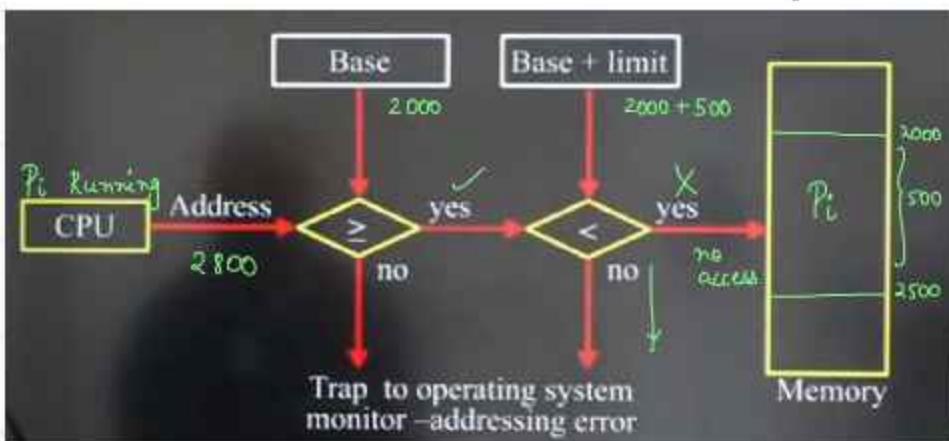


base + Limit

starting & end addresses.

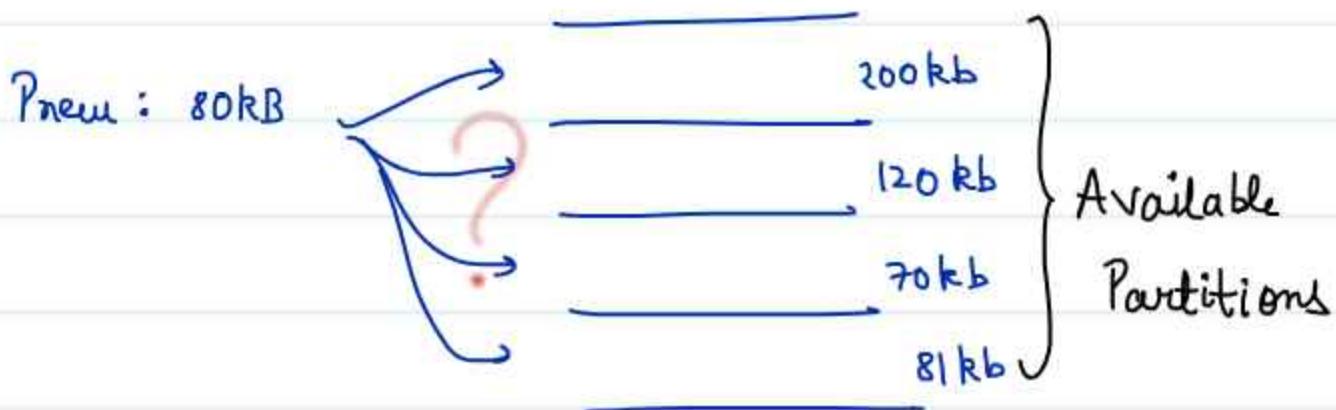
done at O.S. booting time.

used for Partitioning.

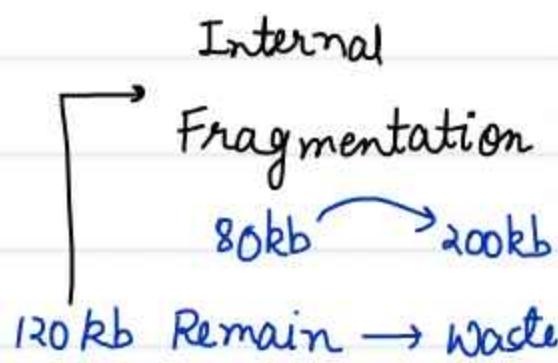


→ Base + limit architecture to ensure protection.

As we have CPU scheduling Techniques, we have technique for partition allocation too.



### ① Partition Allocation Policy :-



- First Fit :- First free big enough.  
(search from first partition)
- Best Fit :- least Internal fragmentation  
smallest free big enough.
- Next Fit :- Works like first fit  
(Search begins from last allocation)
- Worst Fit :- Largest free big enough.

(saves search time)

may work  
faster than First fit

Search is CYCLIC

## Performance of Fixed Partition → static in Nature

- Internal fragmentation :- ✓
  - External fragmentation :- X (wastage of memory outside the partition)
  - Degree of Multiprog :-  
(No of Part) Limited
  - Max Process size :- Limited
  - Partition allocation policy :- Best Fit
- B/w partition there is no free space.
- Max (Part. size)

## # Variable Partition #

Multi prog with variable tasks.  
(Dynamic Partitioning)

Process Req :- < 35K , 115K , 315K , 15K , 120K ..... >  
to  
P<sub>1</sub> P<sub>2</sub> P<sub>3</sub> P<sub>4</sub> P<sub>5</sub>

Initially no partition , that memory is



P <sub>1</sub>	35 kB
P <sub>2</sub>	115 kB
P <sub>3</sub>	315 kB
P <sub>4</sub>	15 kB
P <sub>5</sub>	120 kB
Free hole	250 kB

P<sub>2</sub> & P<sub>4</sub> have all its instruction executed  
So they'll leave the memory.

P <sub>1</sub>	35 kB
Freehole 1	115 kB
P <sub>3</sub>	315 kB
Freehole 2	15 kB
P <sub>5</sub>	120 kB
Free hole <sup>3</sup>	250 kB

Note new pr.  
P<sub>6</sub> comes with  
req = 13 kB

P <sub>1</sub>	35 kB
P <sub>6</sub>	13 kB
Freehole 1	102 kB (new)
P <sub>3</sub>	315 kB
Freehole 2	15 kB
P <sub>5</sub>	120 kB
Free hole <sup>3</sup>	250 kB

Dynamic Partitioning

Allocation policy

FF → F <sub>Hole 1</sub>	NF → F <sub>Hole 3</sub>
BF → F <sub>Hole 2</sub>	WF → F <sub>Hole 3</sub>

Performance Issues :-

Internal Frag X

External Frag ✓

$$P_{new} = 280 \text{ kB}$$

250 kB	15 kB	115 kB
FH3	FH2	FH1

380 kB available

But we are still telling process, mem is not available.

Size of Total Free mem > Process Req,

but still denied :- External Fragmentation

Degree of Multiprogramming :- Flexible

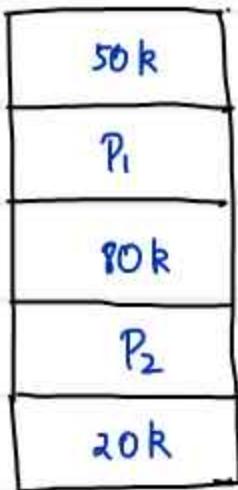
Max Process Size :- Flexible

Partition Alloc Policy :- Worst Fit

(BF may create small free holes which may not accomodate new processes)

create bigger free holes that may acc. new processes.

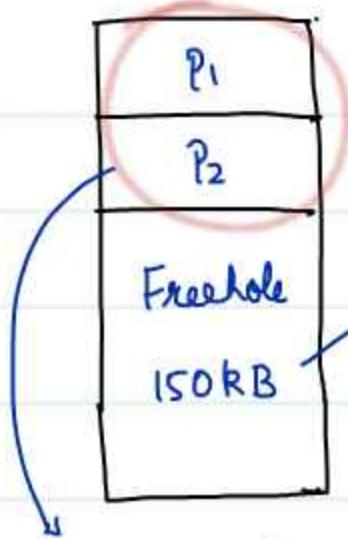
Q How to solve the problem of external frag.?



- Preq: 85K  $\longrightarrow$  Denied
- Even if 150k is available but not contiguously.

we can make it

Memory contiguous, by relocating  $P_1$  &  $P_2$  to one end of memory & creating



bigger freehole.

→ COMPACTON

a)



Merging of available free holes.

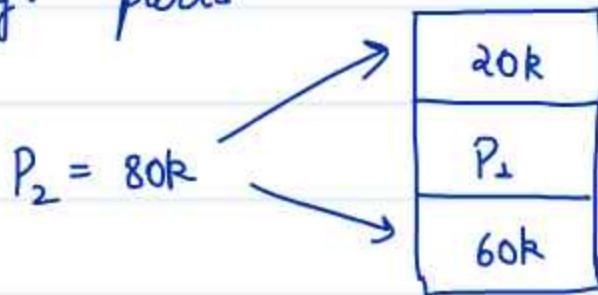
→ Time Consuming operation

Relocated

→ Must support Run Time Address Binding

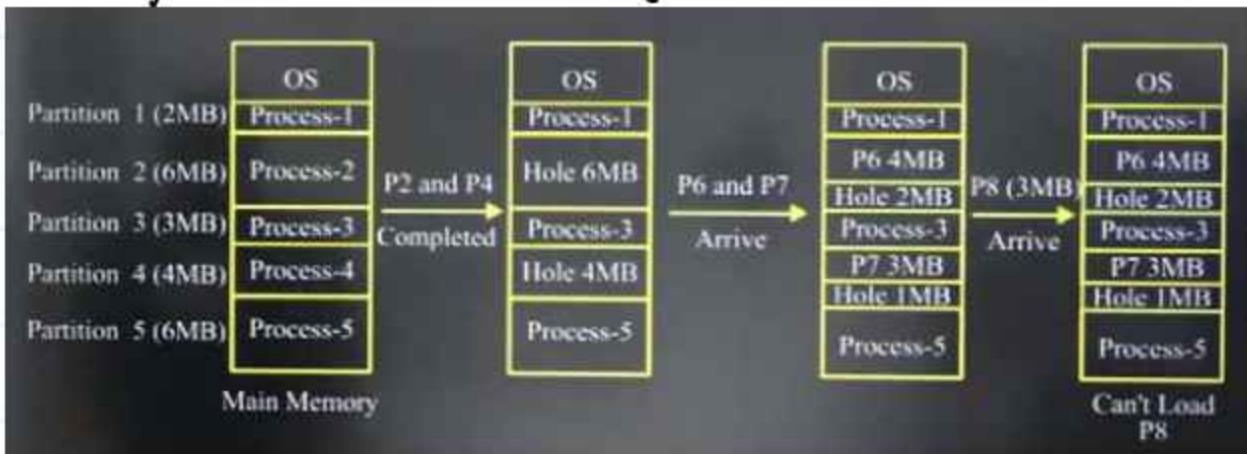
→ Undesirable solution

b) Non-Contiguous Allocation :- Store program into non contg. parts



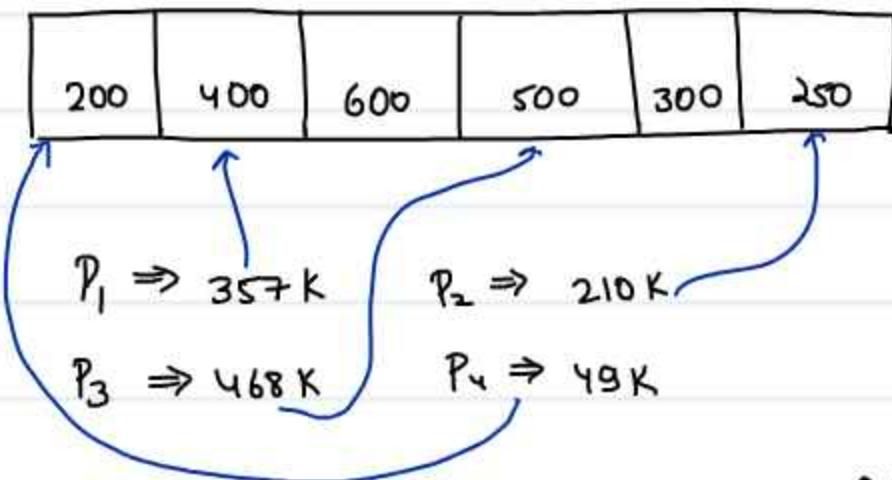
NOTE :- Compaction is not automatic but if two freeholes are adjacent then they are automatically merged.

## Ex of Variable Partitioning



Q

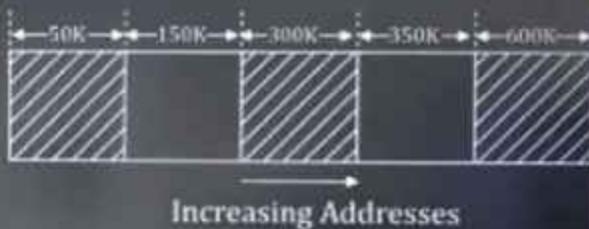
Consider a Memory System having 6 Partitions of sizes 200K; 400K; 600K; 500K; 300K; 250K. There are 4 Processes of sizes: 357K; 210K; 468K; 49K. Using **Best Fit Allocation Policy**, what Partitions are not allocated/ remains Unallocated?



Q

Consider the following Memory Map in which blank regions are not in use and hatched regions are in use. Using Variable Partitions with no Compaction:

The sequence of requests for blocks of sizes 300K, 25K, 125K, 50K can be satisfied if we use:



Either first fit or best fit policy (any one)

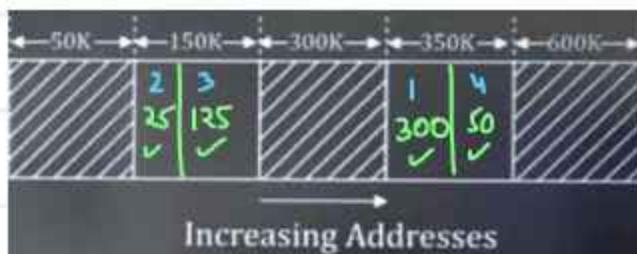
First fit but not best fit policy

Best fit but not first fit policy

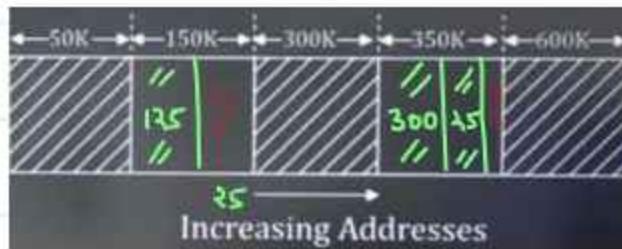
Ans:-

Req:- 300K, 25K, 125K, 50K

## (1) First Fit



## (2) Best Fit



50K can't be accommodated.

Q

Consider a System with Memory of size 1000KBytes. It uses Variable Partitions with no Compaction. Presently there are 2 partitions of sizes 200K & 260K respectively.

(i) What is the allocation request of the Process which would always be denied? → Easy one.

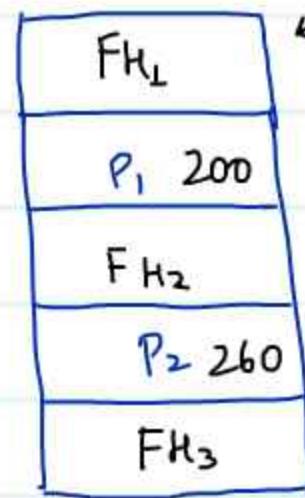
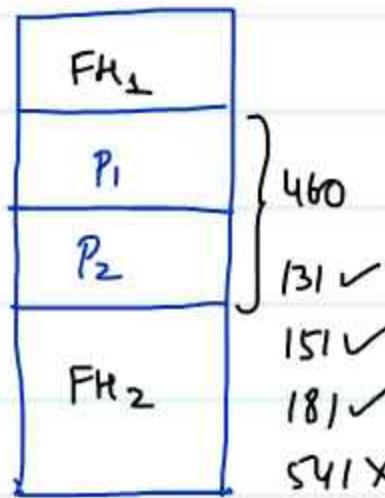
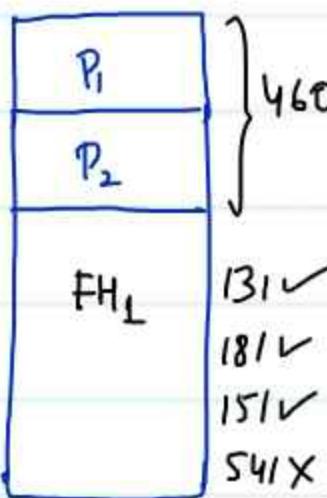
- A. 131 K
- B. 151 K
- C. 181 K
- D. 541 K

★

(ii) The smallest Allocation Request which could be denied is:

- A. 131 K
- B. 151 K
- C. 181 K
- D. 541 K

cases possible

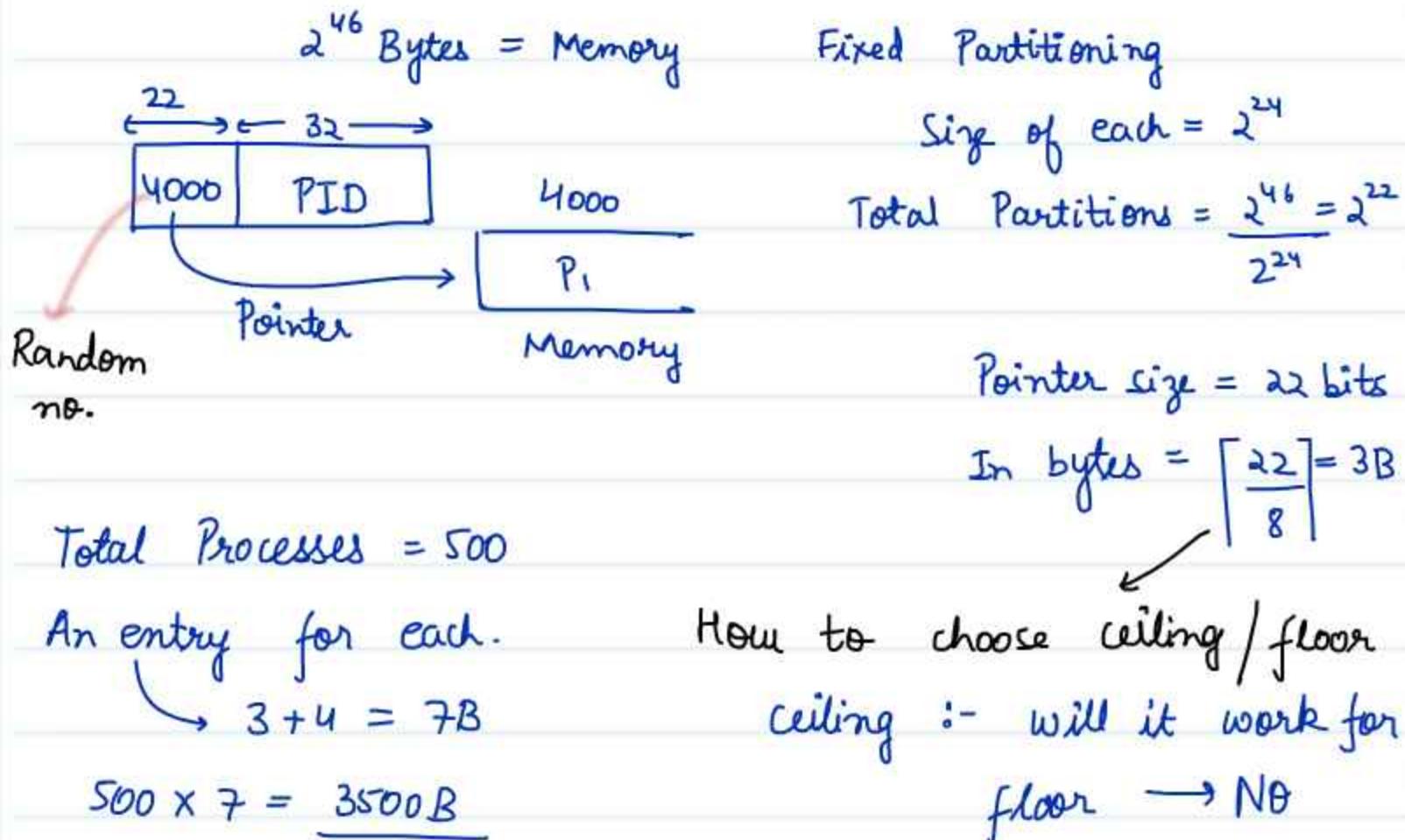


131✓  
151✓  
181X  
541X

Q Consider a System having Memory of size  $2^{46}$  Bytes, uses Fixed Partitioning. It is divided into fixed size Partitions each of size  $2^{24}$  Bytes. The OS maintains a Process Table with one entry per Process. Each entry has, two fields: First, is a pointer pointing to Partition in which the Process is loaded and Second, Field is Process ID(PID). The Size of PID is 4Bytes.

Calculate

- The Size of Pointer to the nearest Byte.
- Size of Process Table in Bytes if the System has 500 Processes.



Q

Consider a System Using Variable Partition with no Compaction

Free holes	4K; 8K; 20K; 2K
Program size	2K; 14K; 3K; 6K; 10K; 20K; 2K
Time for Execution	4; 10; 2; 1; 4; 1; 8

Using Best Fit Allocation Policy and FCFS CPU Scheduling

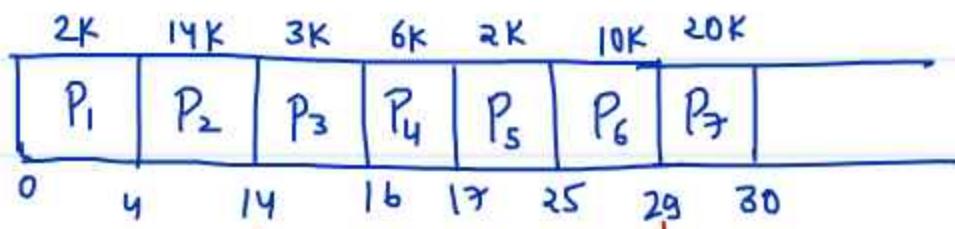
Technique, Find the Time of Loading & Time of Completion of each program. The Burst Times are in Seconds.

FH <sub>1</sub>	3 3K	4K
FH <sub>2</sub>	1K	
FH <sub>3</sub>	5 2K	8K
	6K	
FH <sub>4</sub>	2 14K	20K
	4 6K	
	1 2K	2K

Graph to ensure that they are not adjacent otherwise they will coalesce (merge).



Schedule them



→ Here 10K will come.

will come into memory as soon as space become available.

FH <sub>1</sub>	3 3K	4K
FH <sub>2</sub>	1K	
FH <sub>3</sub>	5 2K	8K
	6K	
FH <sub>4</sub>	6 10K 4K	20K
	4 6K	
	1 2K	2K

at this time 20K will come

Load Time :-

$$P_1 \rightarrow 0$$

$$P_6 \rightarrow 14$$

$$P_2 \rightarrow 0$$

$$P_7 \rightarrow 29$$

$$P_3 \rightarrow 0$$

FH <sub>1</sub>	3 3K	4K
FH <sub>2</sub>	1K	
FH <sub>3</sub>	5 2K	8K
	6K	
FH <sub>4</sub>	7 20K	20K ✓
	1 2K	2K

FH <sub>1</sub>	3 3K	4K
FH <sub>2</sub>	1K	
FH <sub>3</sub>	5 2K	8K
	6K	
FH <sub>4</sub>	7 20K	20K ✓
	1 2K	2K

FH <sub>1</sub>	3 3K	4K
FH <sub>2</sub>	1K	
FH <sub>3</sub>	5 2K	8K
	6K	
FH <sub>4</sub>	7 20K	20K ✓
	1 2K	2K

CT → See from Gantt Chart.



Consider allocation of memory to a new process. Assume that none of the existing holes in the memory will exactly fit the process's memory requirement. Hence, a new hole of smaller size will be created if allocation is made in any of the existing holes. Which one of the following statements is TRUE?

- a) The hole created by next fit is never larger than the hole created by best fit X
- b) The hole created by worst fit is always larger than the hole created by first fit X
- c) The hole created by first fit is always larger than the hole created by next fit may be equal
- d) The hole created by best fit is never larger than the hole created by first fit

# MEMORY MANAGEMENT - 4

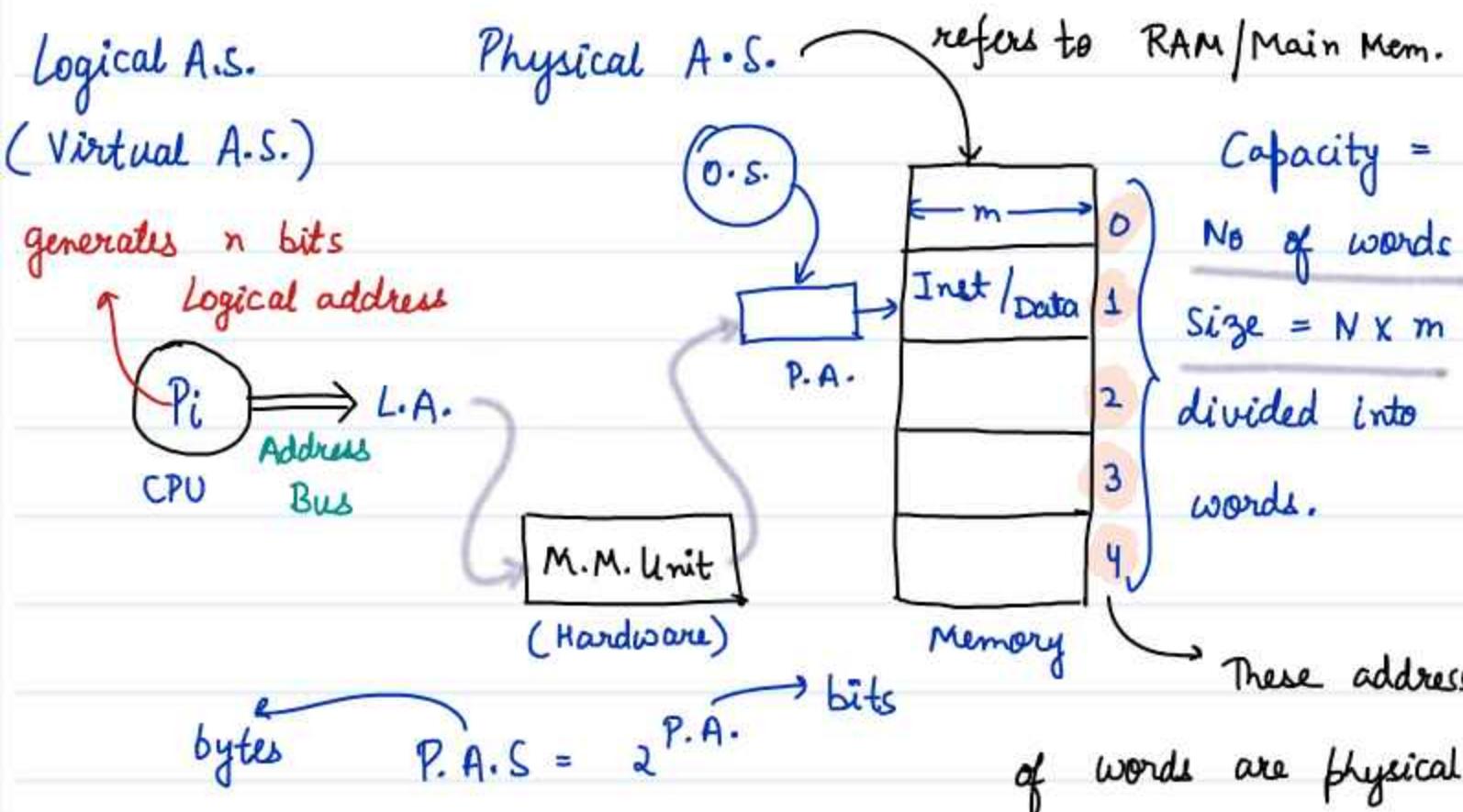
## # Non Contiguous Allocation #

↳ to prevent external fragmentation

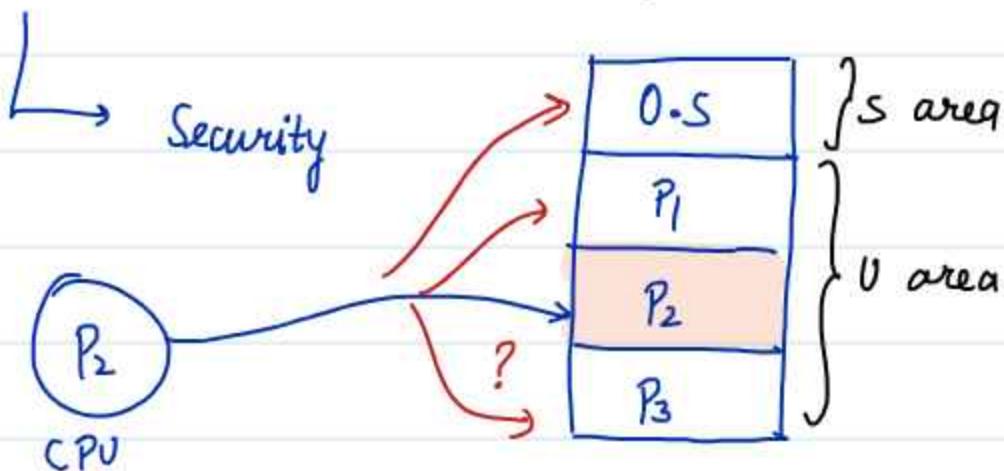
Compaction → undesirable

- Time Consuming operation
- Require programs to exhibit RT Address Binding.

Address Space :- A set / group / space of words associated with addressee.



Q Why doesn't Process directly access Physical address?

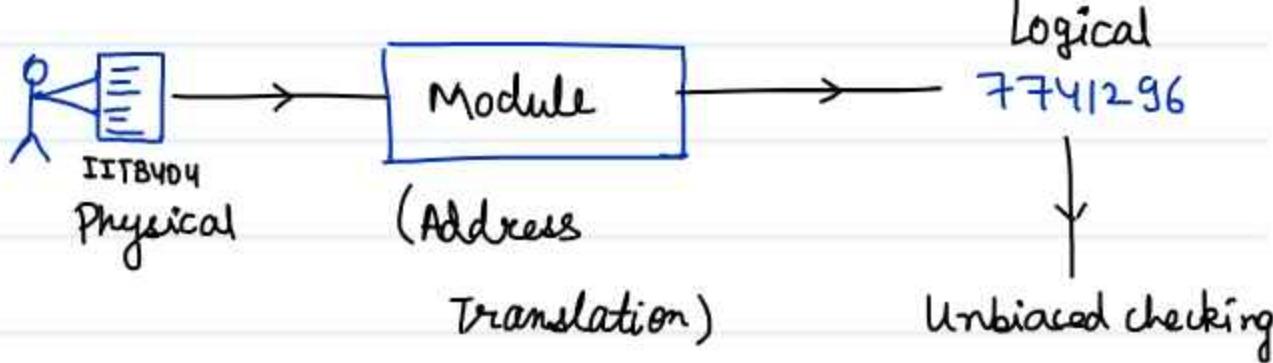


If we allow  $P_2$  to access P.A. directly, then it can also access P.A. of  $P_1$  &  $P_2$ . Who is stopping  $P_2$  to access P.A. of O.S.?

↓ O.S. says

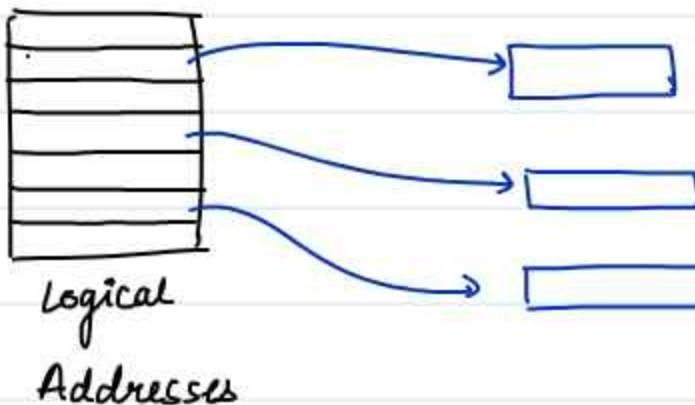
Let's the process generate whatever they want I have the mapping table. I will map & make space accessible so that you can't directly interfere.

Analogy :-



O.S. maintains a table :-

Mapping of all  
Logical addresses a  
process can generate ) to ( Physical  
Addresses.



Each L.A.

corresponds to a  
word in MM.

OR

Each Logical Address correspond to a Physical Address.

Function of MMU :- Address Translation

- Program assume that I'm in Logical Memory  
O.S. will give it an illusion

↳ Formally : Abstraction Layer

# Design & Implementation of NCG Techniques #

↳ Organization of L.A.S. , P.A.S. & M.M.U.

## # Simple Paging #

say

LAS = 8KB

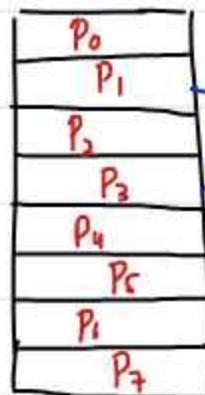
PAS = 4KB

L.A. = 13 bits

P.A. = 12 bits

a)

Organization of L.A.S. / V.S.

Divided into equal sized unit  
called pages.

Pages

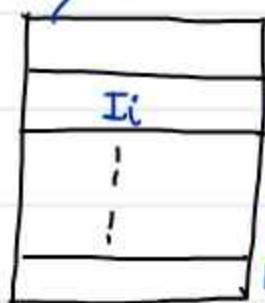
Page size = 1KB

8 pages  $\Leftarrow$  No of Pages =  $\frac{\text{L.A.S}}{\text{P.S.}}$   $N = 2^P$

3 bits  $\Leftarrow$  Page Number ( $P$ ) =  $\log_2(\text{No. of Pages})$

$P = \log_2(N)$

word in a Page



1 Page

Page Offset ( $d$ ) =  $\log_2(\text{Page Size})$   
or Displacement

Page size =  $2^d$

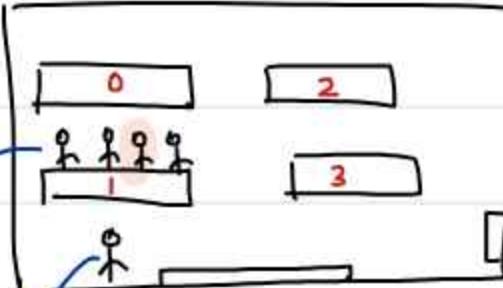
How many bits need to refer 1B from  
that 1KB page?  $\rightarrow$  10 bits

That's page offset

Analogy :-

Students

Teacher



Classroom

Teacher doesn't know the names of the student, so he/she will call as :-

(10) page offset

Third student from

Classroom

MM

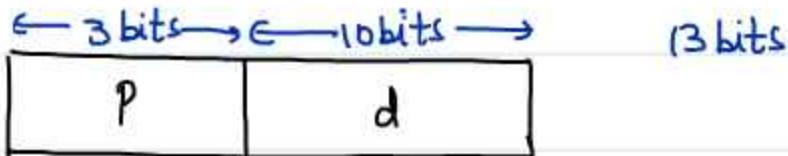
Page

(01)

first bench.

That's exactly how logical addressing work.

### Logical Address Format



logical Address

Q

$$LAS = 32 \text{ MB}$$

$$P.S. = 4 \text{ KB}$$

$$N = ? \quad 2^{13} \text{ pages}$$

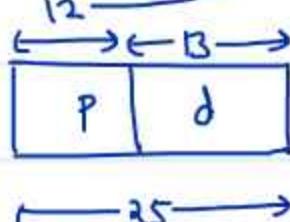
$$P = ? \quad 25 - 12 = 13 \text{ bits}$$

$$d = ? \quad 12 \text{ bits}$$

$$LA = 25 \text{ bits}$$

$$d = 13 \text{ bits}$$

$$N = ? \quad 2^{12} \leftarrow$$



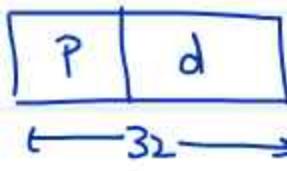
$$2^{12} \times P.S. = 2^{32}$$

$$P.S. = 2^{12}$$

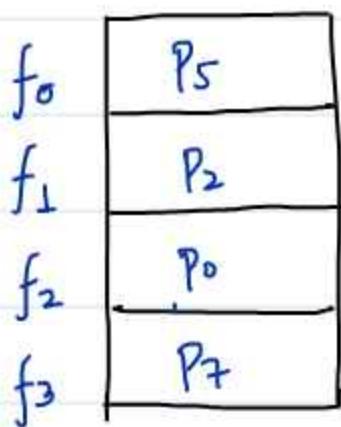
$$N = 2^K \rightarrow 2^{11} \text{ pages}$$

$$LA = 32 \text{ bits}$$

$$P.S. = ? = 2^{12}$$



## b) Organization of P.A.S. (4KB)



P.A.S. is divided into equal size units known as frames (Page frames)

↓

Each frame hold one page.

- Meant for holding a page.
- Any page can be stored in any frame

Frame Size = Page Size

$f$	$d$
-----	-----

$$\text{No of Frames } (M) = \frac{\text{P.A.S.}}{\text{FS}}$$

$$\text{Frame No. } (f) \text{ bits} = \log_2 M$$

$$\text{Frame off} = \text{Page offset} = d \text{ (same)}$$

Q LA = 31 bits , PA = 20 bits , PS = 4KB

$$LAS = 2^{31} B \quad PS = 2^{12} B$$

$$N = ? \quad M = ? \quad P = ? \quad f = ? \quad d = ?$$

$N = 2^{19}$        $M = 2^8$        $P = 19 \text{ bits}$        $f = 8 \text{ bits}$        $d = 12 \text{ bits}$

Q System has 4K Pages & 1K frames. PAS = ?

LA = 32 bit

$$\text{No of Pages} = 2^{12}$$

$$\text{Page size} = \frac{\text{LAS}}{N} \quad \text{LAS} = 2^{32}$$

$\downarrow 2^{20}$

$$\text{Offset}(d) = 20 \text{ bits}$$

$$\text{No of Frames} = 1K$$

$$\text{Offset} = 20 \text{ bits}$$

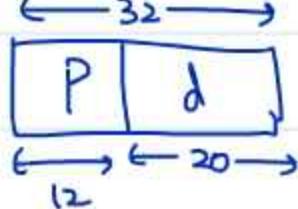
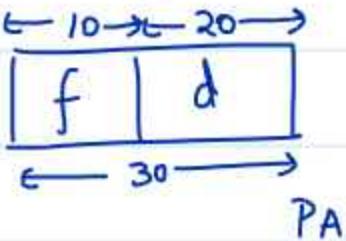
$\downarrow$

$$\frac{\text{PAS}}{\text{No of Frames}} = \text{Frame size} = 2^{20}$$

$2^{20}$  words in each frame

$$\begin{aligned} \text{PAS} &= 2^{20} \times 2^{10} \\ &= 2^{30} \end{aligned}$$

Equal to Page size



No of pages & frames

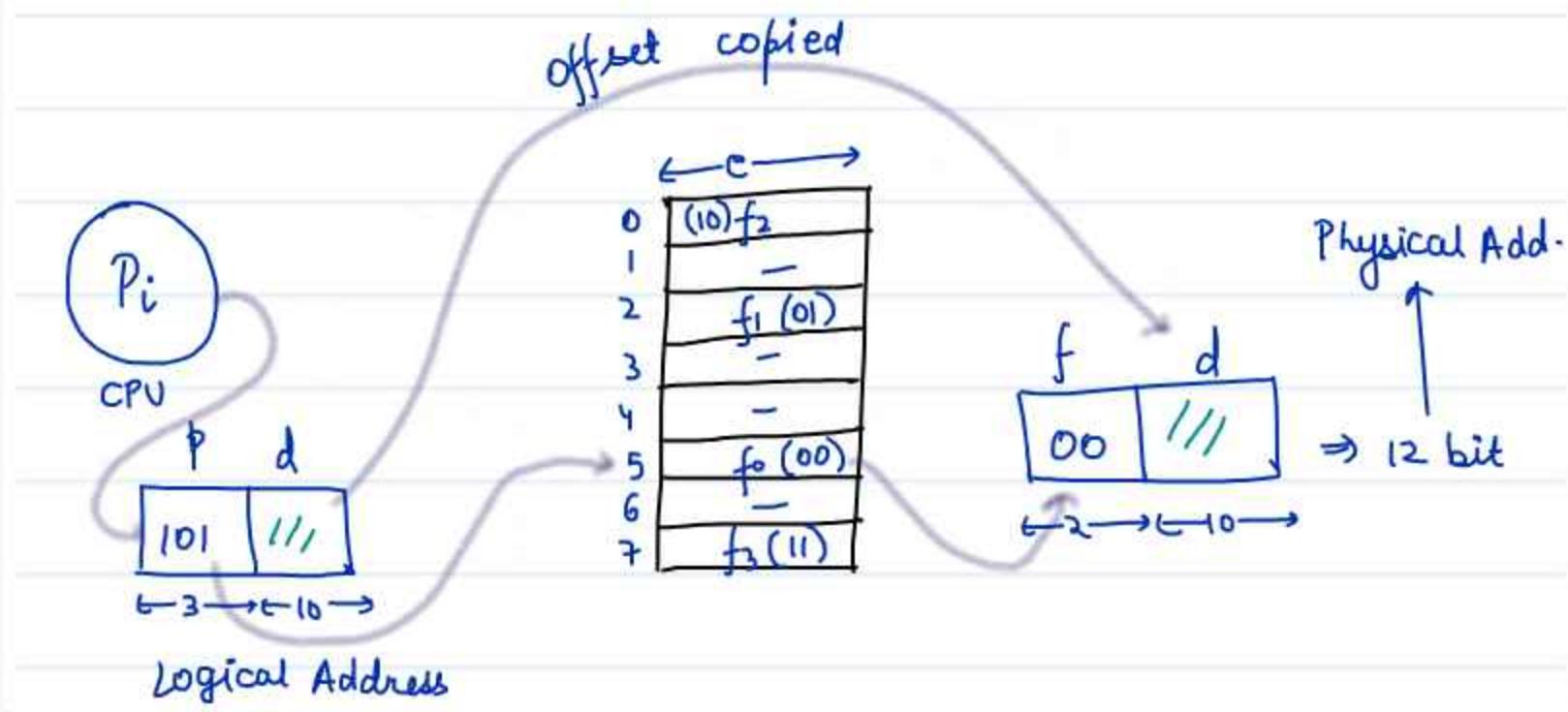
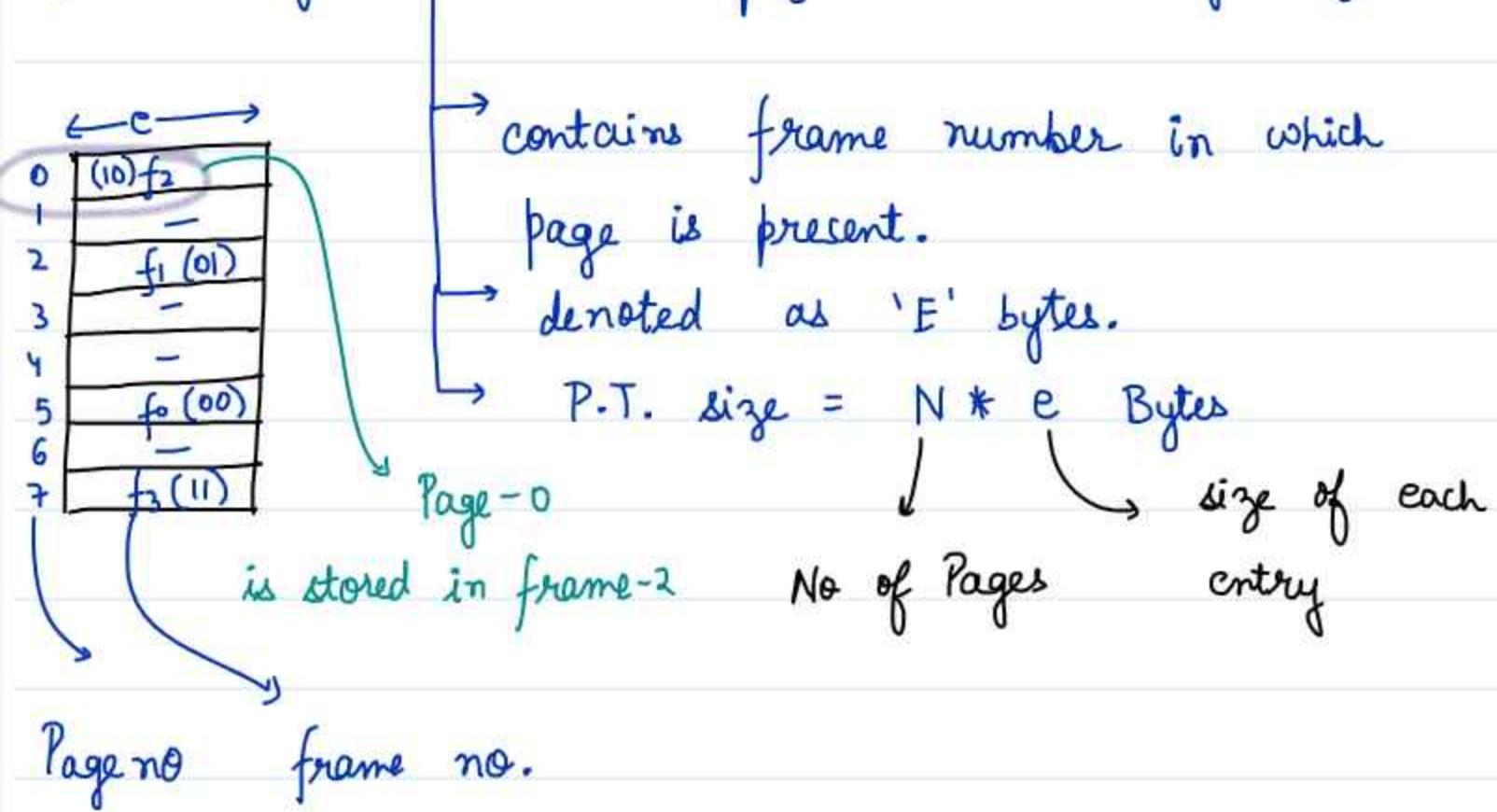
logical  $\rightarrow$  LA, LAS, P, f, d  $\rightarrow$  offset

Physical  $\rightarrow$  PA, PAS, M, f, d  
Address  $\checkmark$   $\downarrow$  space  $\rightarrow$  page / frame no

c) Organization of M.M.U. [ Page - Table / Page Map Table / Address Trans. Table ]

① Each process has its own Page table.

- ② Page Tables are stored in memory.
- ③ Page Tables are organized as set of entries known as page table entries.
- ④ No of entries in page table = No of Pages



# MEMORY MANAGEMENT - 5

Let's revise :-

$$LAS = 8 \text{ KB}$$

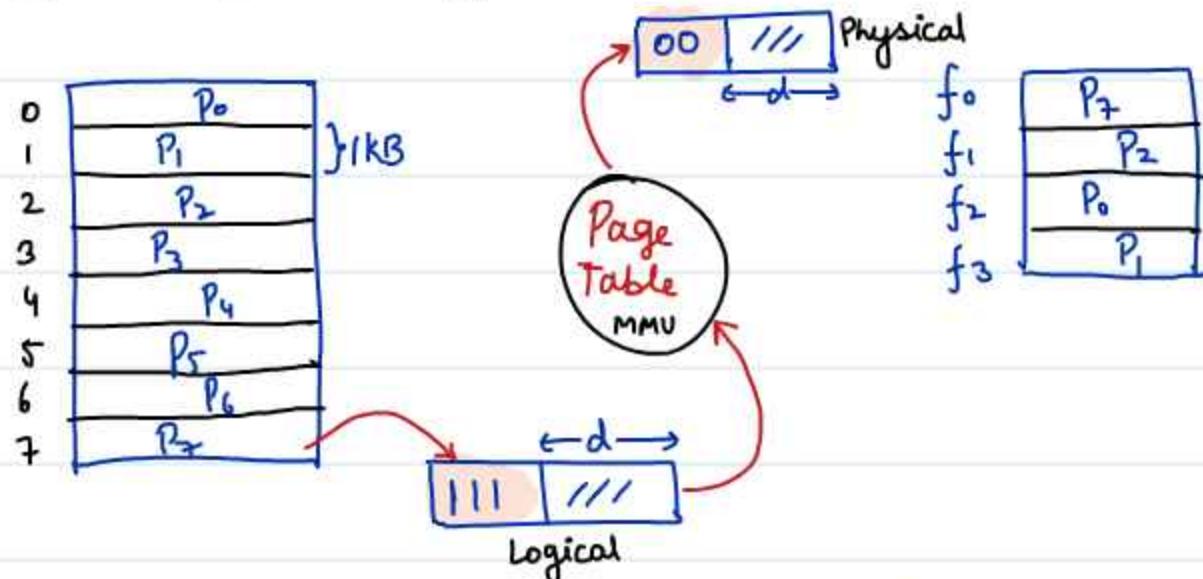
$$PAS = 4 \text{ KB}$$

$$PS = 1 \text{ KB}$$

$$LA = 13 \text{ bits}$$

$$PA = 12 \text{ bits}$$

$$d = 10 \text{ bits}$$



$$N_{\text{pages}} = \frac{LAS}{PS}$$

number

$$P = \log_2 N_{\text{pages}}$$

bits

$$d = \log_2 PS$$

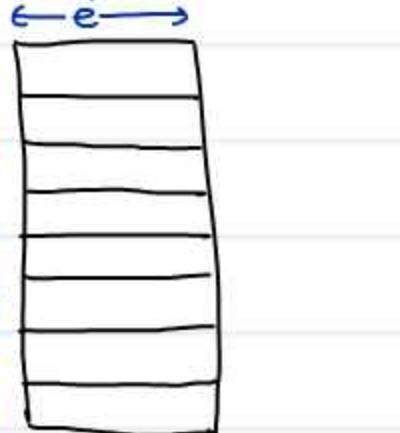
bits

$$M_{\text{frames}} = \frac{PAS}{PS}$$

frames

$$f = \log_2 (M_{\text{frames}})$$

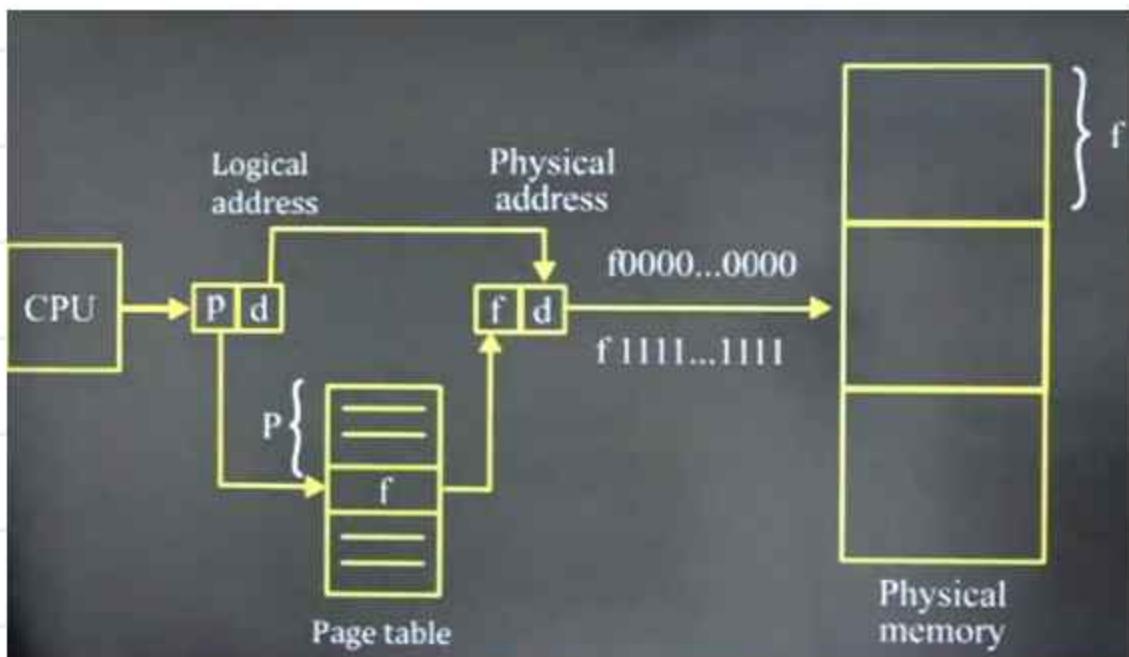
$$d = \log_2 PS$$



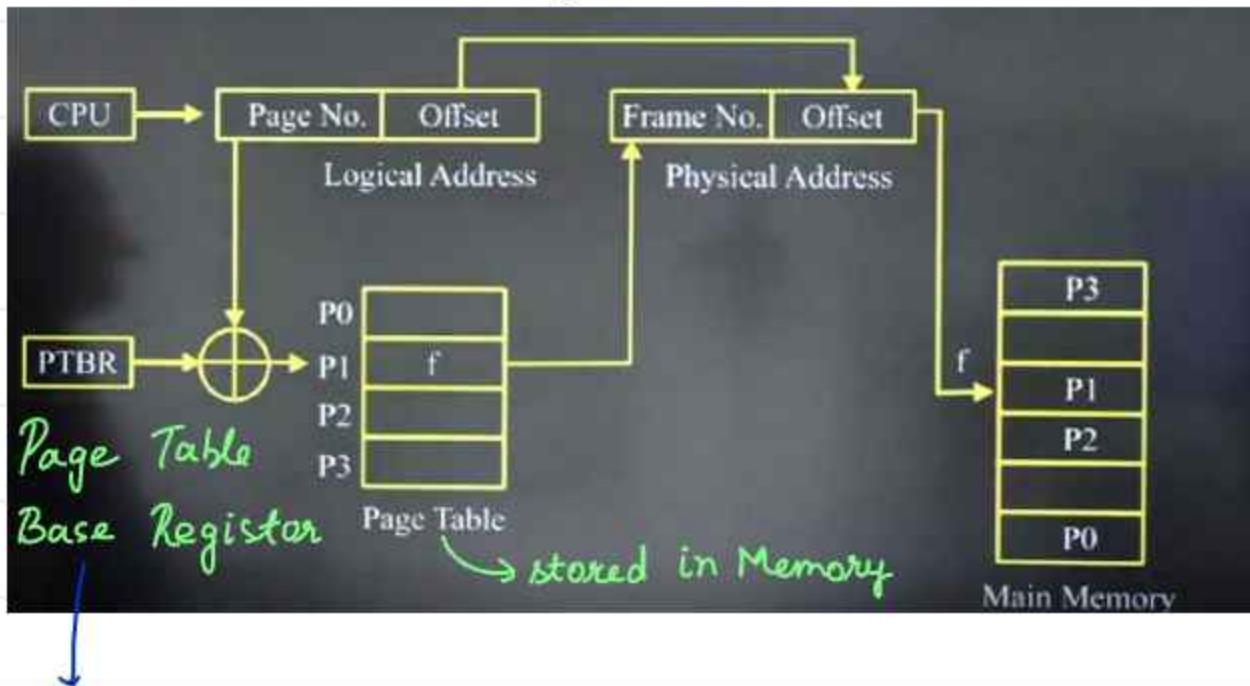
Page Table

$$\text{Size} = N \times e$$

No of bits  
needed to refer.  
a words within a  
Page.



"How Paging Works"



The address in Memory where the page table is stored.

Q

A Computer System using **Paging Technique** implements an 8KB Page with a Page Table of Size 24MB. The Page Table Entry is 24 bits. What is the length of Virtual Address in this System? PQA

Page size = 8 KB

$\hookrightarrow d = 13 \text{ bits}$

$N \times e = 24 \text{ MB}$

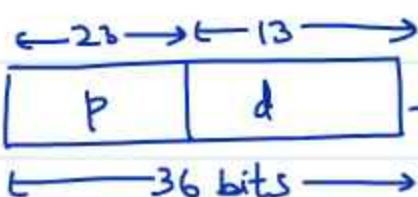
24 bits

No of Pages

$$\text{Total VM} = 8 \text{ KB} \times N \rightarrow \frac{24 \times 2^{20}}{3}$$

$$= 2^3 \times 2^{10} \times 2^3 \times 2^{20} = 2^{36}$$

length = 36 bits

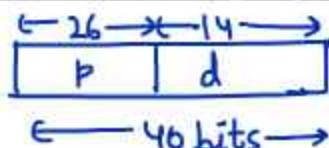


Logical Address

Q

Consider a computer system with 40-bit virtual addressing and page size of sixteen kilobytes. If the computer system has a one-level page table per process and each page table entry requires 48 bits, then the size of the per process page table is megabytes.

$$\text{V.A. / L.A.} = 40 \text{ bits}$$



$$\text{P.S.} = 16 \text{ KB} = 2^{14} \text{ B}$$

$$d = 14 \text{ bits}$$

$$e = 6 \text{ B}$$

$$p = 26 \text{ bits}$$

$$\text{Size} = N \times e \rightarrow 6 \text{ B}$$

$$= 2^6 \times 6 \text{ MB}$$

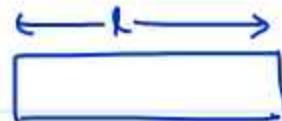
$$\text{no of pages} = 2^{26}$$

$$= \underline{\underline{384 \text{ MB}}}$$

Q

A Computer System using Paging technique has a Virtual Address of length T. The Number of Pages in the Address Space are 'Z'. There are 'H' Frames in PAS. Calculate the number of bits in Page Offset and the size of PAS

V.A. =  $l$  bits



$$N_p = z$$

$$M_f = H$$

$$z \times P.S. = 2^l$$

$$P.S. = \frac{2^l}{z}$$

$$\begin{aligned} d &= \log_2(2^l/z) \\ &= l - \log_2 z \end{aligned}$$

$$d = ?$$

$$\log_2(P.S.)$$

$$PAS = ?$$

$$M_f \times \text{size of frame}$$

$$\text{No of frames}$$

$$PAS = H \times \frac{2^l}{z}$$

$$\begin{aligned} \text{Size of} \\ \text{Page} \\ \frac{2^l}{z} \end{aligned}$$

Q

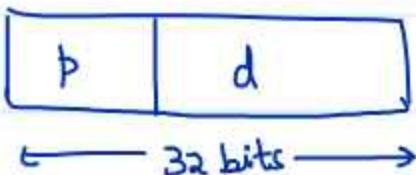
Consider a System using Simple Paging Technique with Logical Address (LA) of 32 bits. Page Table Entry (PTE) of 32 bits. What must be the Page Size in bytes, such that the Page Table of the Process Exactly fits in one Frame of Memory (PAS)?

$$LA = 32 \text{ bits}$$

$$e = 32 \text{ bits} = 4B$$

$$P.S. = ?$$

$$\text{Size of Page Table} = \text{Page size}$$



$$\frac{2^{32}}{\text{Page size}} = \text{Page size} \quad ?$$

128 KB

$$(Page \text{ size})^2 = 2^{32} \times 4B$$

$$\begin{aligned} \text{Page size} &= 2^{16} \times 2B \\ &= 2^{17} B \end{aligned}$$

Q       $VA = 32 \text{ bits}$        $P.S. = 4KB$        $P.A.S = 64MB$

$P.T.S = ?$

$$VA = p \quad d$$

$$\log_2 N = 20$$

$$\log_2 P.S. = \log_2 2^{12} = 12 \text{ bits}$$

$$N = 2^{20} \text{ pages}$$

$$M_f \times P.S. = 64$$

$$M_f = \frac{2^6 \times 2^{20}}{2^{12}} = 2^4$$

$$N \times e \rightarrow 2^{20}$$

$$\log_2 M_f = 14 \text{ bits} \approx 2B$$

$$P.T.S. = 2^{20} \times 2B = \underline{\underline{2MB}}$$

## # Performance of Paging #

a) Timing Issue :- Page Table is stored in Memory

a) M.M. Access Time = 'm' (nanoseconds)

Amount of time CPU takes to read / write the word from memory.

b) Effective Memory Access Time = '2m'

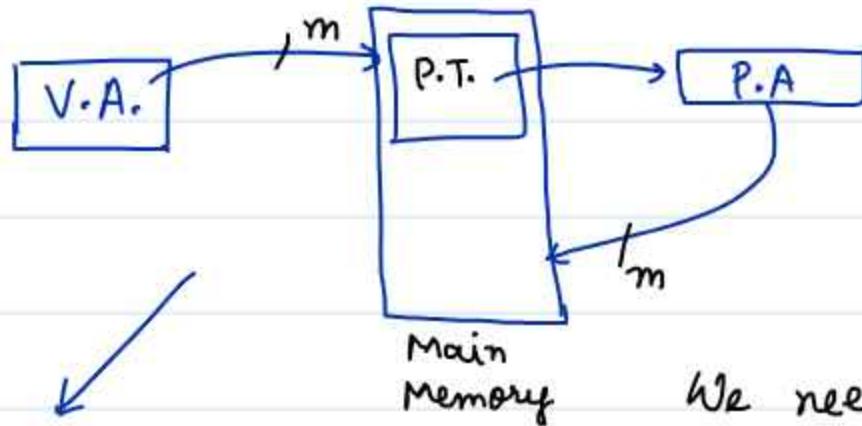
(Actual Time Taken)

Time taken to access Page table

$m + m$

Actual mem access

Process generates



This increases the execution time

We need to reduce access



time

We want to make ' $2m$ ' close to ' $m$ '.

Let's first focus on reducing P.T. m.

Faster, Costly, small + (ft search) ← CACHE MEMORY

By

Will first check  
in cache if ↗  
not available we  
will go to Page Table.

Some entries of P.T. we store in  
Cache memory, which has a shorter  
access time :-  $c < m$

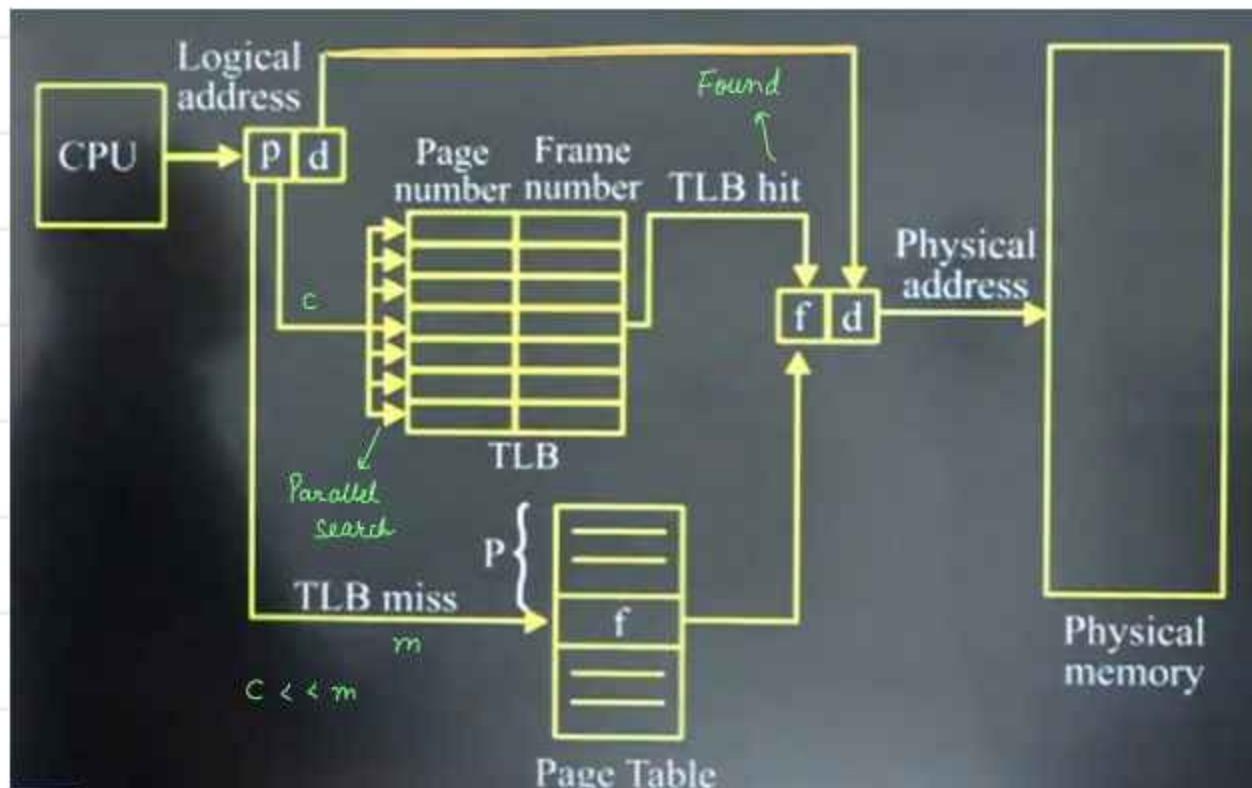
CACHE is known here by name T.L.B



Translation Lookaside Buffer.

This is How it works with TLB :-

7



Use  
of  
TLB is  
only justified  
only if the  
Hit Ratio  
is high.

Entry of TLB :-

$$\text{Log}_2 N \text{ pages} + \text{Log}_2 M \text{ frames}$$

If TLB Hit :-  $C + m < 2^m$

TLB Miss :-  $C + m + m$

$$\text{Hit Ratio} = \frac{\text{Hits}}{\text{Total}} (x)$$

$$\frac{\text{Miss} + \text{Hits}}{\text{Total}} = \frac{1}{\text{Total}}$$

$$\text{Miss Ratio} = \frac{\text{Miss}}{\text{Total}} (1 - x)$$

Effective Memory Access Time using TLB =  
 $x(C+m) + (1-x)(C+2m)$

# Paging with TLB & Physical Address Cache (PAC)

Reduce Address Translation Time

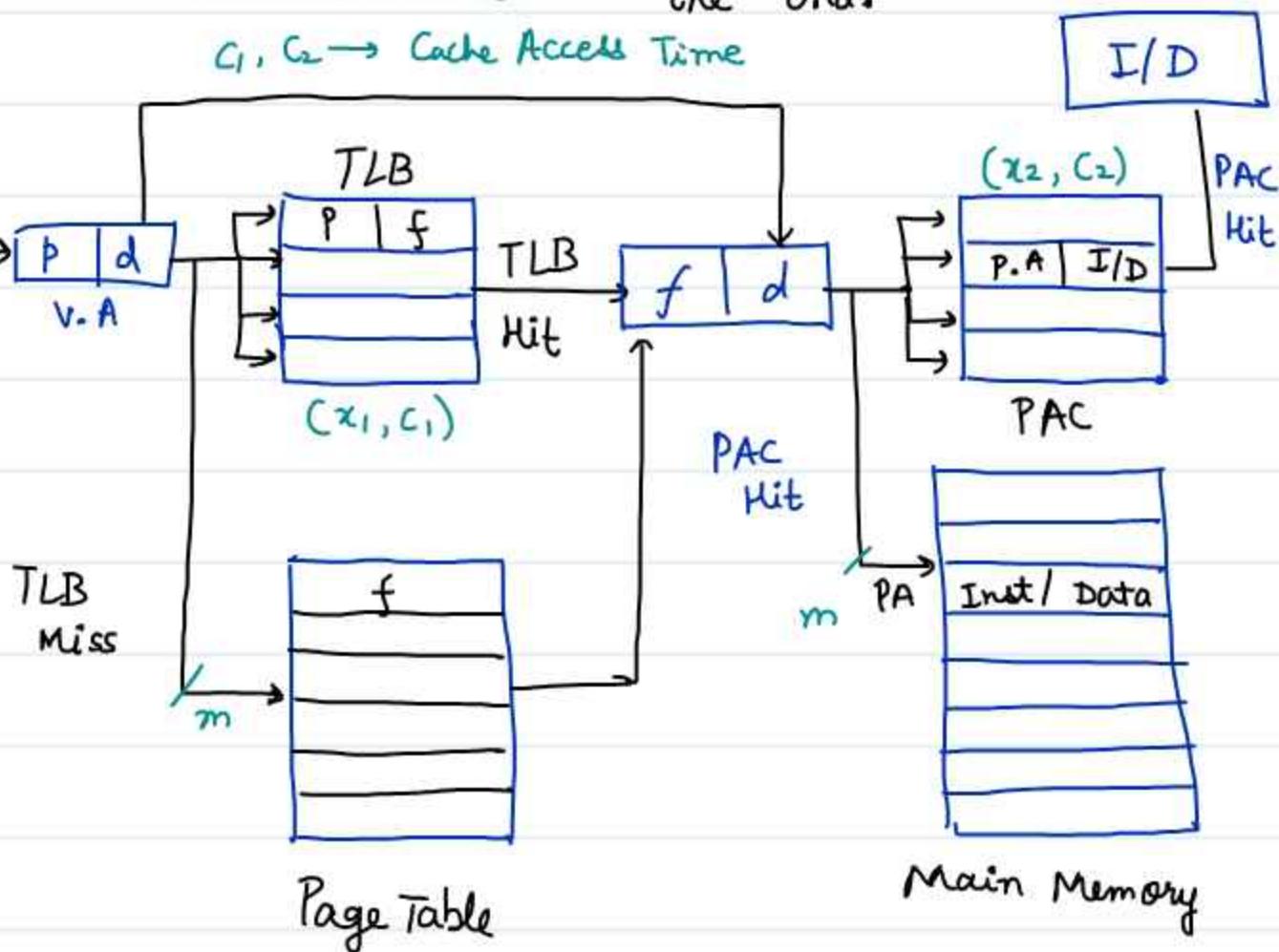
Reduce Final Access Time

That means we

used to access MM in the end.

$x \rightarrow$  Hit Ratio

$C_1, C_2 \rightarrow$  Cache Access Time



$$\begin{aligned}
 \text{EMAT (TLB \& PAC)} = & \quad x_1 [C_1 + x_2 C_2 + (1-x_2)(C_2+m)] \\
 & + \\
 & (1-x_1)[C_1 + m + x_2 C_2 + (1-x_2)(C_2+m)]
 \end{aligned}$$

Q

Consider a System using Paging with TLB. What Hit Ratio is required to reduce the EMAT from 'D' ns to 'Z' ns using TLB. Assume that TLB access time is 'K' ns.

$$D = 2m$$

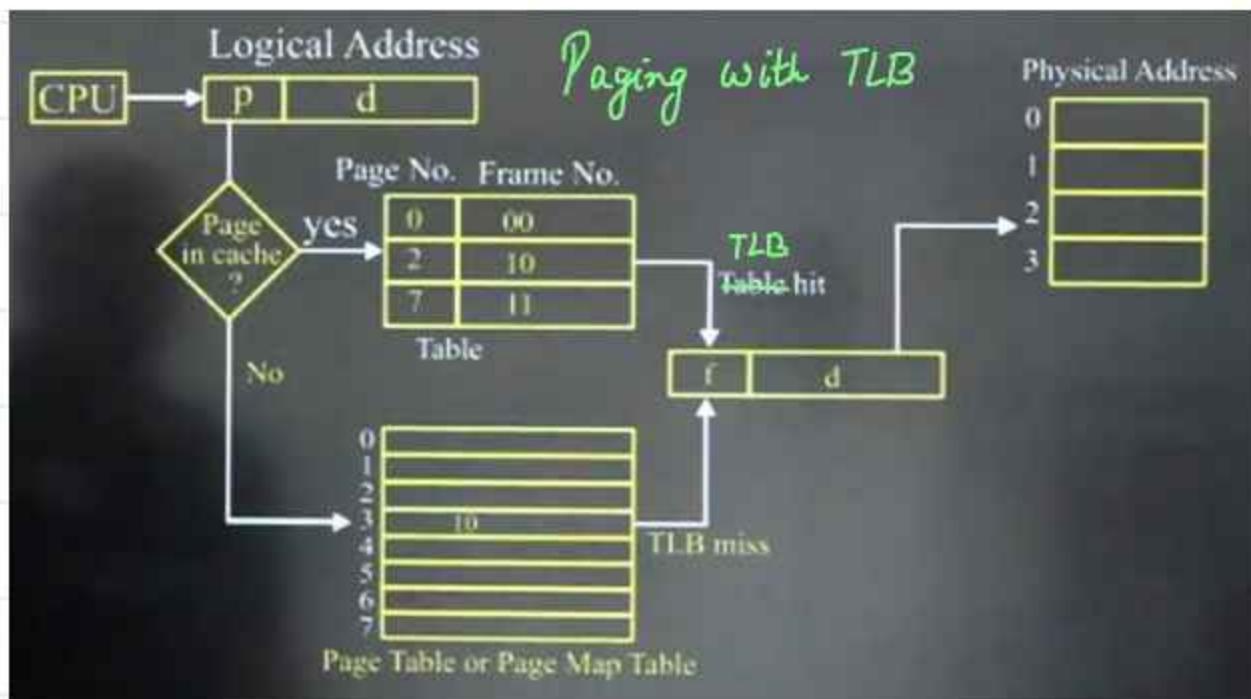
$$m = D/2$$

$$z = \alpha(c+m) + (1-\alpha)(c+m+m)$$

$$z = \alpha \left( K + \frac{D}{2} \right) + (1-\alpha) (K+D)$$

$$z = K\alpha + \frac{D\alpha}{2} + K + D - K\alpha - D\alpha$$

$$z - (K+D) = -\frac{D\alpha}{2} \Rightarrow \alpha = \frac{[(K+D)-z]}{\frac{D}{2}}$$



b) Space Consumption Issue :-  $\text{Page Table} = N \times e$   
size

$$LA = 32 \text{ bits}$$

$$P.S = 4 \text{ KB}$$

Approximate Page Table size  $N = \frac{2^{32}}{2^{12}} = 2^{20}$

if  $e = 4B$  then  $P.T.S. = N \times e$   
 $= 2^{20} \times 4B = 4MB$

4MB for Page Table of a Process

let's say there are 100 Process  $\longrightarrow$  400MB of MM  
just to store Page Tables.



We need to reduce space cons.

## # Reduce Page Table size #

$$P.T.S. = N \times e$$

$$PTS \propto \underset{\text{Page size}}{\frac{N}{\text{L.A.S.}}} \rightarrow \text{Fix}$$

$$PTS \propto N \propto \frac{1}{P.S.} \Rightarrow$$

$$\boxed{P.T.S. \propto \frac{1}{P.S.}}$$

We can Reduce Page Table size by Increasing the  
Page size but there are side effects too.

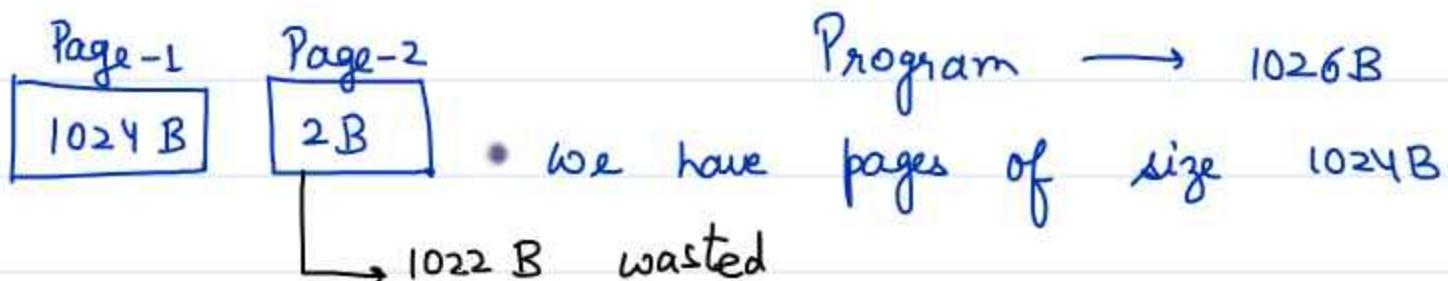
H.W.

# Memory Management - 6

$$LA = 32 \text{ bits} \quad PS = 4 \text{ KB}$$

$$PTS \propto N \propto \frac{1}{PS} \Rightarrow PTS \propto \frac{1}{PS}$$

① Increase: Page size  $\Rightarrow$  Internal Fragmentation



$$IF = 1022 \text{ B}$$

- We have pages of size 2 B



- Page size  $\uparrow$  then IF  $\uparrow$
- Page size  $\downarrow$  then IF  $\downarrow$

Then what should be optimal page size?

lets say VAS = 'S' Bytes      Page size = 'P'

PTE = 'e' Bytes

Optimal ( $P$ ) = ?  $\left[ \begin{array}{l} \xrightarrow{\text{IF}} \text{Page Table size} \\ \xrightarrow{\text{P.T.S.}} \end{array} \right] \text{Min}$

$$\text{P.T.S.} = \left[ \left( \frac{S}{P} \right) * e \right] \text{ Bytes}$$

I.F.  $\Rightarrow$  Happens in Last Page of every process  
 $\hookrightarrow$  Let's say  $\frac{P}{2}$  goes to waste.

$$O = \text{Total overhead} = \left[ \left( \frac{S}{P} \right) e + \frac{P}{2} \right] \Rightarrow \text{Min}$$

$$\frac{dO}{dP} = 0 = -\frac{1}{P^2} \times se + \frac{1}{2} = 0 \Rightarrow P^2 = 2Se$$

$$P = \sqrt{2Se}$$

If  $P/n$  goes to waste then  $\Rightarrow P = \sqrt{nse}$

V-A-S. entry size

~~Page size~~  $\xrightarrow{\text{Impacts}}$  P.T.S.  
~~Page size~~  $\xrightarrow{\text{Impacts}}$  I.F.

Q

A Machine has a 32-bit Address Space and an 8KB Page. The Page Table is entirely in hardware, with one 32-bit word per entry. When a Process starts, the Page Table is copied to the hardware from memory, at one word every 100 nsec. If each Process runs for 100 msec (including the time to load the page table), what fraction of the CPU time is devoted to loading the Page Tables?

$$N = \text{VAS} / \text{P.S.} = 2^{32} / 2^{13} = 2^{19}$$

$$e = 4B$$

$$\text{Speed} = 4B / 100\text{nsec}$$

$$\text{Process Time} = \text{B.T.} + \text{P.T. Load Time} = 100 \text{ ms}$$

$$\% \text{ CPU for Loading} = \frac{\text{Load Time}}{\text{Process Time}} \times 100$$

$$\text{Page Table Size} = 2^{19} \times 2^2 = 2^{21} \text{ B}$$

$$2^2 \text{ B} \longrightarrow 100 \text{ ns}$$

$$LT = 2^{19} \times 100 \text{ ns}$$

$$2^{21} \text{ B} \longrightarrow ?$$

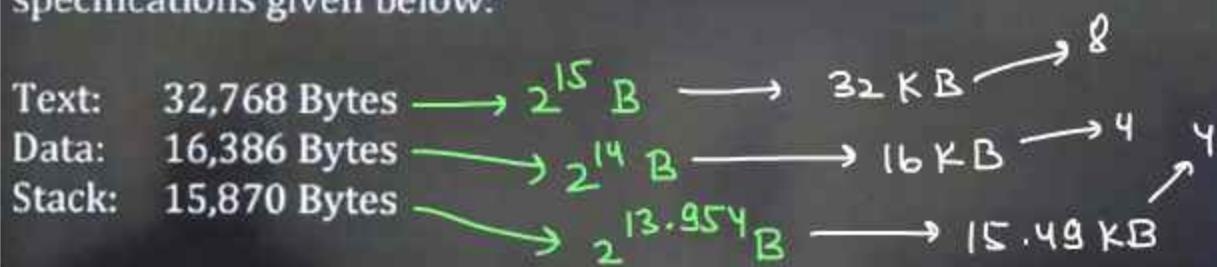
$$\frac{100}{2^2} \times 2^{21} = 2^{19} \times 100 \text{ ns}$$

$$\text{Process Time} = 100 \text{ ms}$$

$$\% \text{ CPU} = \frac{2^{19} \times 100 \text{ ns}}{100 \times 10^6 \text{ ns}}$$

$$\frac{2^9}{10^3} \times 100 \approx 50\% \Leftrightarrow \frac{2^9 \times 2^{10}}{10^6} \rightarrow K \rightarrow 10^3$$

Q Consider a System using Paging technique with an Address Space of 65,536 Bytes. The Page Size in this System is 4096 Bytes. The Program Consists of Text, Data and Stack Sections as per the specifications given below:



C  
A  
R  
E  
F  
U  
L

A Page of the program contains portion of only one section i.e either Text or Data or Stack.

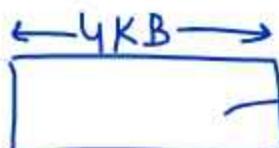
- (a) Does the Program Fit in the given Address Space? (Y/N)
- (b) What is the Maximum Page Size in Bytes such that the Program Fits in the Given Address Space?

can't  
approx in  
these  
cases

Ans :-

$$\text{Space} = 65,536 \text{ B} = 2^{16} \text{ B} \quad P.S = 4 \times 2^{10} \text{ B} \\ = 4 \text{ KB}$$

$$N = \frac{2^{16}}{2^2} = 2^4 = 16 \text{ pages}$$



Page

Text or Data or Stack

Text
Data
Stack

No. of Pages for Text = 8 pages

No. of Pages for Data = 5 pages ?

No. of Pages for stack = 4 pages

Answer:  $y = 14.000176099486$

$$\log_2(16386) = 14.000176099486$$

$$2^{14.000176099486} = 16386$$

can't write

it as 14.

16386	=	<input type="text"/>
2		
<input type="button" value="Calculate"/> <input type="button" value="Clear"/>		

lots of IF but  
5 Pages will be  
Required.



Total 17 pages required but we have only 16.  
⇒ NO.

Instead of 4KB if P.S = 4B then ?

$$N = \frac{2^{16}}{2^2} = 2^{14} \text{ pages} \quad \left. \begin{array}{l} \text{Text} = 8192 \\ \text{Data} = 4096 \\ \text{Stack} = 3968 \end{array} \right\} 16256$$

YES ⇐ 16384 ✓

b) Max P.S. for Program to fit :-

$$\text{space} = 64 \text{ KB } (2^{16} \text{ B}) \quad N = \frac{2^{16}}{2^x} = 2^{16-x}$$

$$\text{Text : } 32768 / 2^x = 2^{15-x}$$

$$\text{Data : } 16386 / 2^x = 2^{(14+)-x}$$

$$\text{Stack : } 15870 / 2^x = 2^{13.95-x}$$

$$2^{15-x} + 2^{14.0001-x} + 2^{13.95-x} = 2^{16-x}$$



↓  
can't solve like that.

### # Hashed Paging #

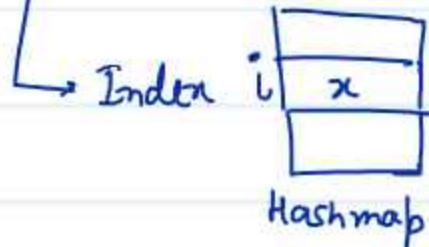
To Associate a smaller P.T.  
 organize data with a process.  
 in such a way such that  
 searching becomes easy.

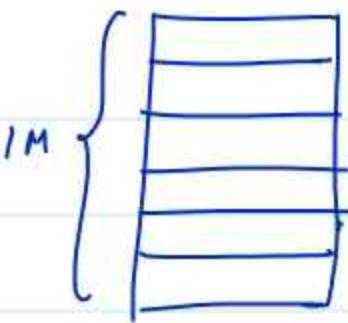
We will design a page table using Hashing. Table  
 ↗ Hashed Page Table

VA = 32 bits

P.S. = 4KB

N<sub>pages</sub> = 1M pages





No of Pages in Page Table  
= No of Pages in Memory

Page Table

Practically, there are 1M pages in the address space but process uses only around 5 pages, rest are generally unused.



But according to Conventional Paging whole 1M should be in Memory.

Ex:- If I have %.10 function then there will be 10 entries in my hash table.



Challenge : Collision  $f(I_1) = f(I_2)$   
where  $I_1 \neq I_2$

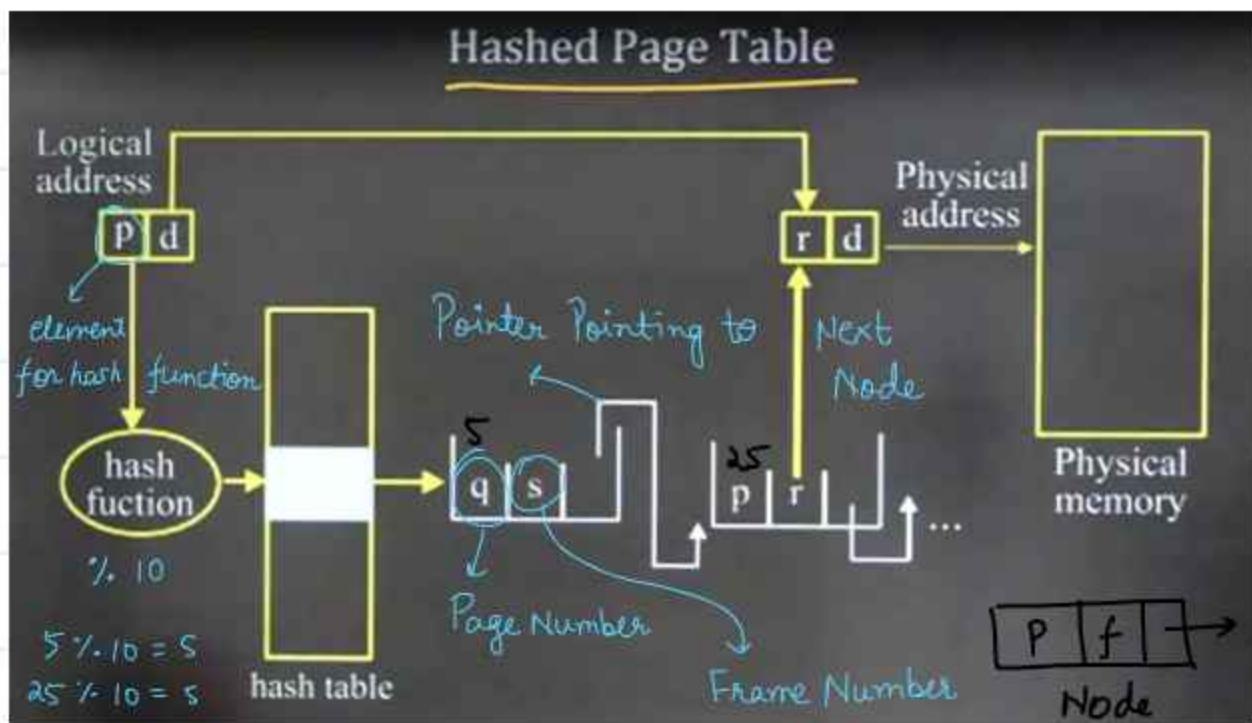
Collision Resolution Technique

→ Probing  
→ Chaining

Here to resolve collision, we will use Chaining



Implemented using linked list



So instead of 1M entries, we have a hash Table with 10 entries & linked list associated to them.



Space Optimized, but Time Inefficiency



$O(n)$  searching time of LL.

NOTE :- It's a trade off b/w space & Time in Von Neuman Architecture.

Time for searching  $I_1$  &  $I_2$  is different but same in traditional.

a) ↑ page size

b) Hashing

c) MULTI LEVEL PAGING / Hierarchical Paging  
 (Recursive Paging)

→ Paging on Page Table (Index of an Index)

Simple Paging :- Overcome the Problem of External Fr. that happens in Variable Partition.

Paging with TLB :- Reduce EMAT.

Hashed Paging :- Associate smaller PT @ cost of Search time.

MULTI LEVEL Paging :- Associate smaller PT

$$VA = 32 \text{ bits} \quad PS = 4KB \quad e = 4B \quad PTS = 4MB$$

When do we say Page Table size is small ?

When page table fits in one frame of memory.

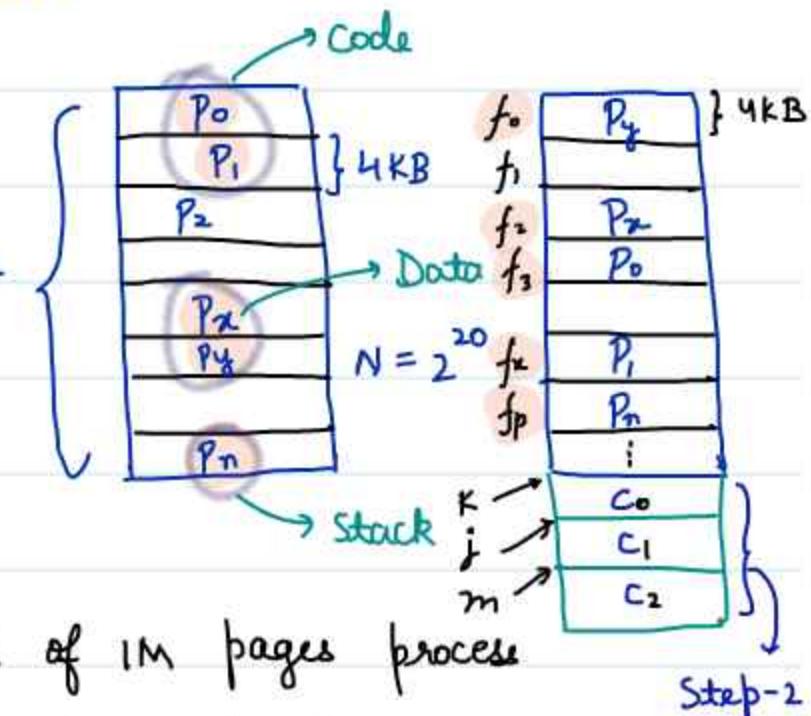
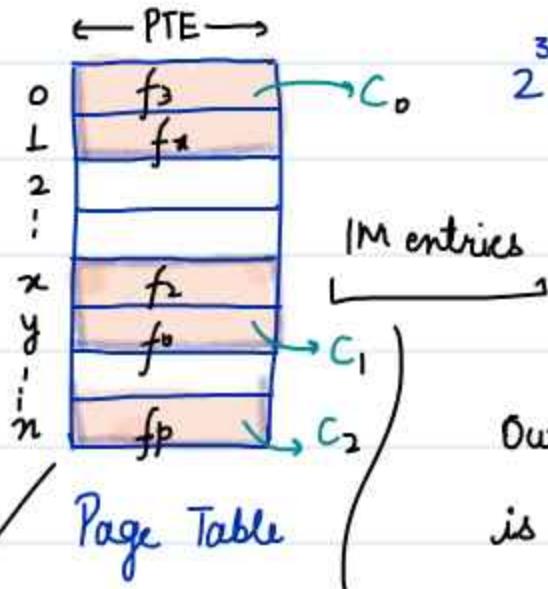
Paging as Concept generally involve 3 steps :-

- Divide the Address space into pages  
 → group of words associated with addresses .

- \* Storing the pages in Physical Address space.  
(PAS)
  - \* Accessing the chunks of A.S. in P.A.S. via  
or through page table.

$$V_A = 32 \text{ bits} \quad P.S. = 4 \text{ KB}$$

P.S. = 4KB

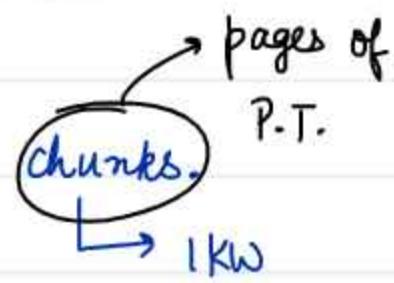


3 chunks are useful

To reduce the size of Page Table, you have to apply Paging as a concept to Page Table.

~~Stop -1~~

Page Table is divided into



No of chunks = 1K

~~Stop - 2~~

→ Store chunks

into memory

In same memory you will store pages of VAS & P.T.

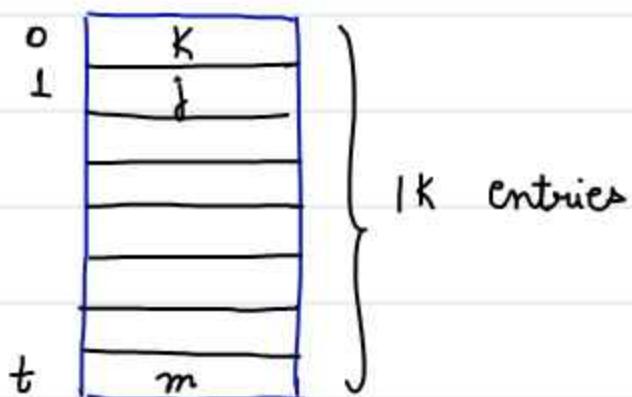
only 3 are useful.

Step-3

10

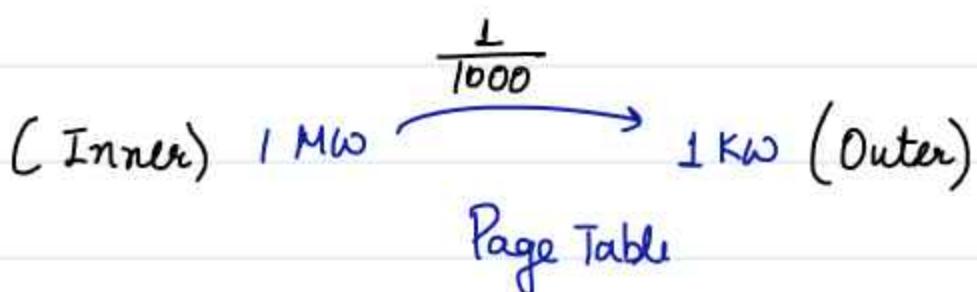
Access the inner page table through outer Page table.

First Level PT



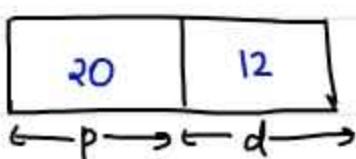
Process will be associated with Outer PT & 3 chunks of inner P.T. } 3K

Earlier it was associated with 1M entry P.T.

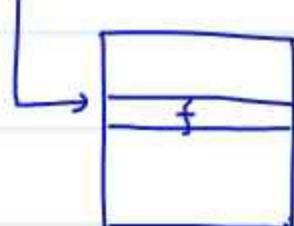


LA = 37 bits

Ps = 4 KB



No of Pages



No of entries in Inner PT

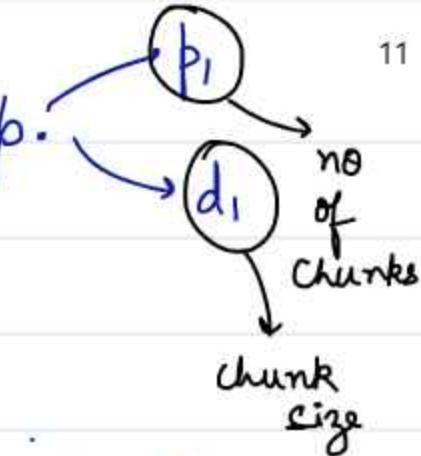
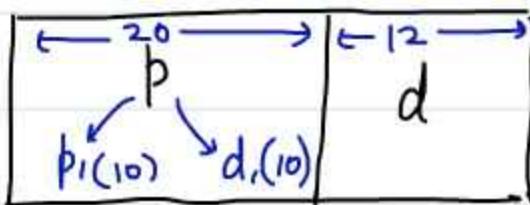
No of Pages  
p

Page size  
d

Earlier we applied paging on IA

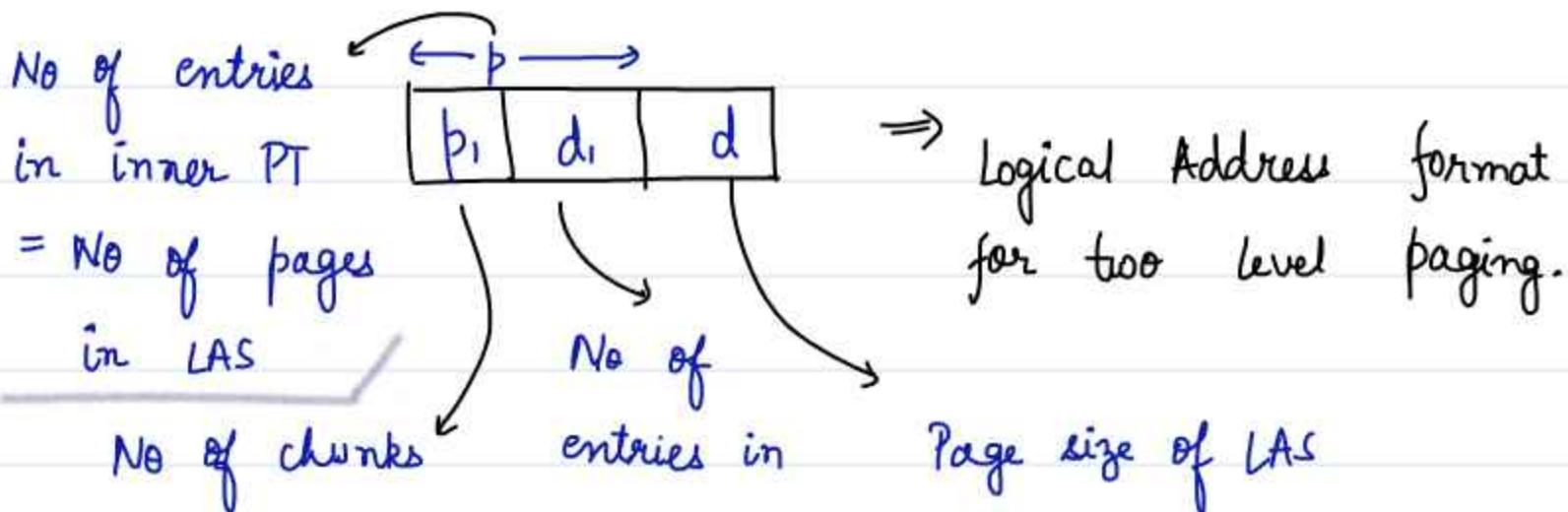
No of Pages  
p  
Page size  
d

Now we are applying paging on p.



No of Entries in Inner PT = No of pages in LAS

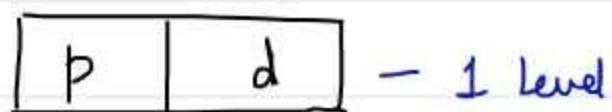
No of Entries in Outer PT = No of Chunks in Inner PT



No of chunks  
of Inner Page table  
=

Entries of outer PT

Multi Level Paging :-



$p_1$

- 1 Level

$d_1$

- 2 Level

$d_2$

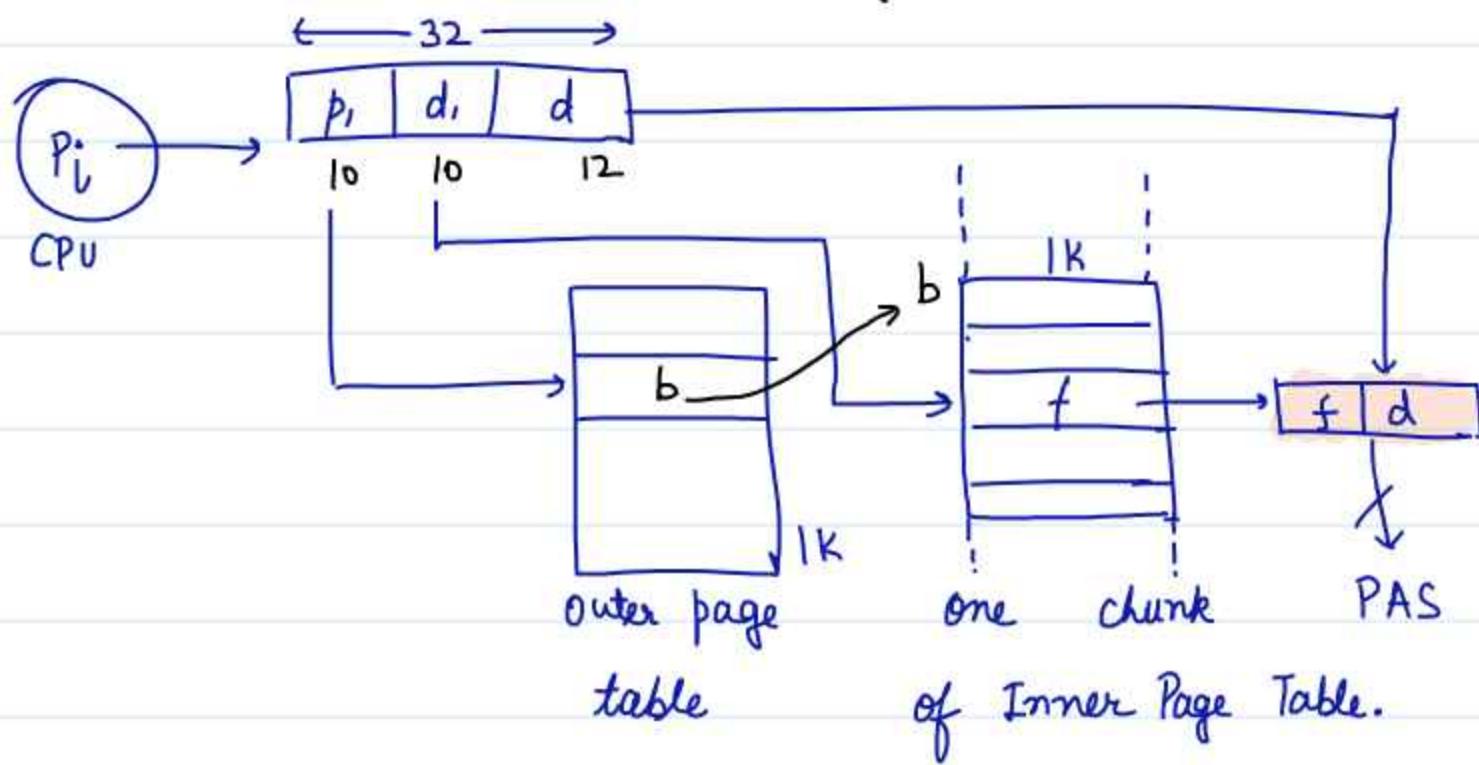
- 3 Level

!

We did paging coz LAS was too big to fit in one frame of MM, but Inner page table can easily fit in one frame of MM.

When Inner page table became too big to fit in one frame of memory, we did paging creating outer page table which can easily fit in one frame of MM.

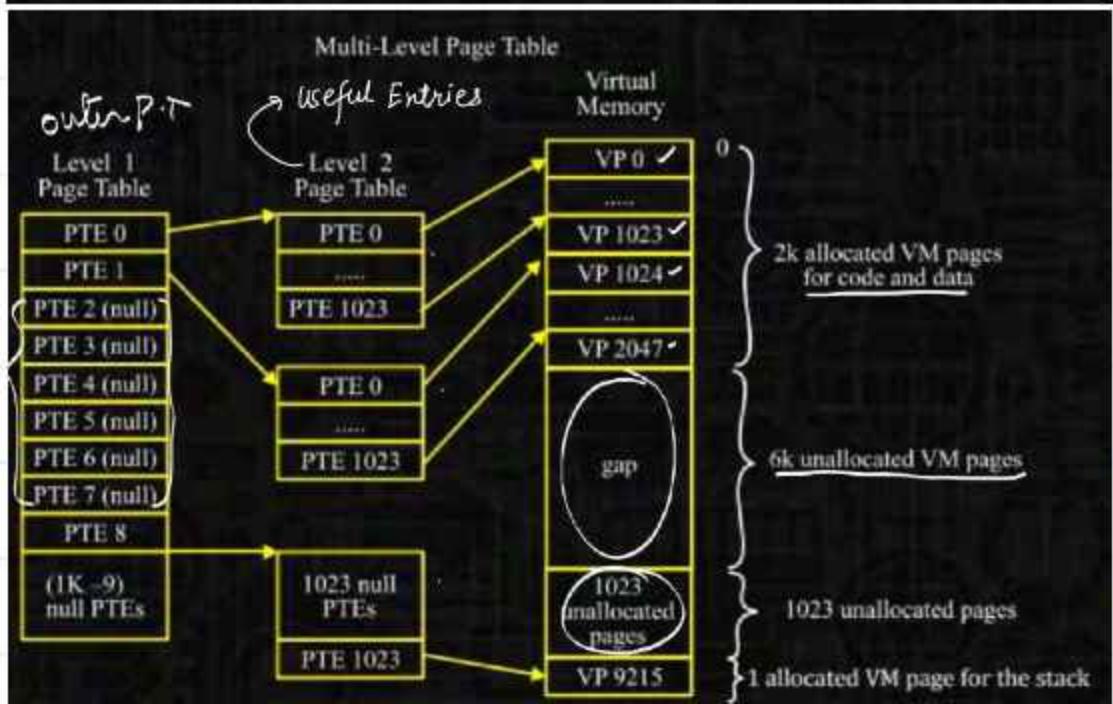
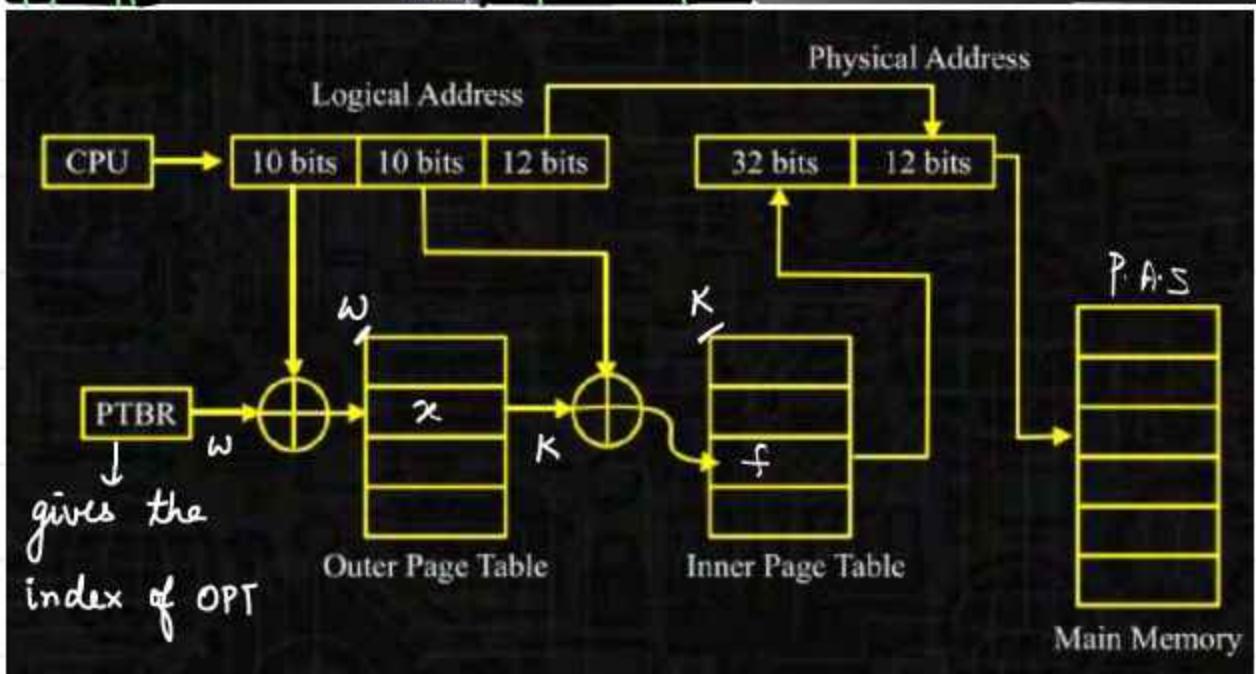
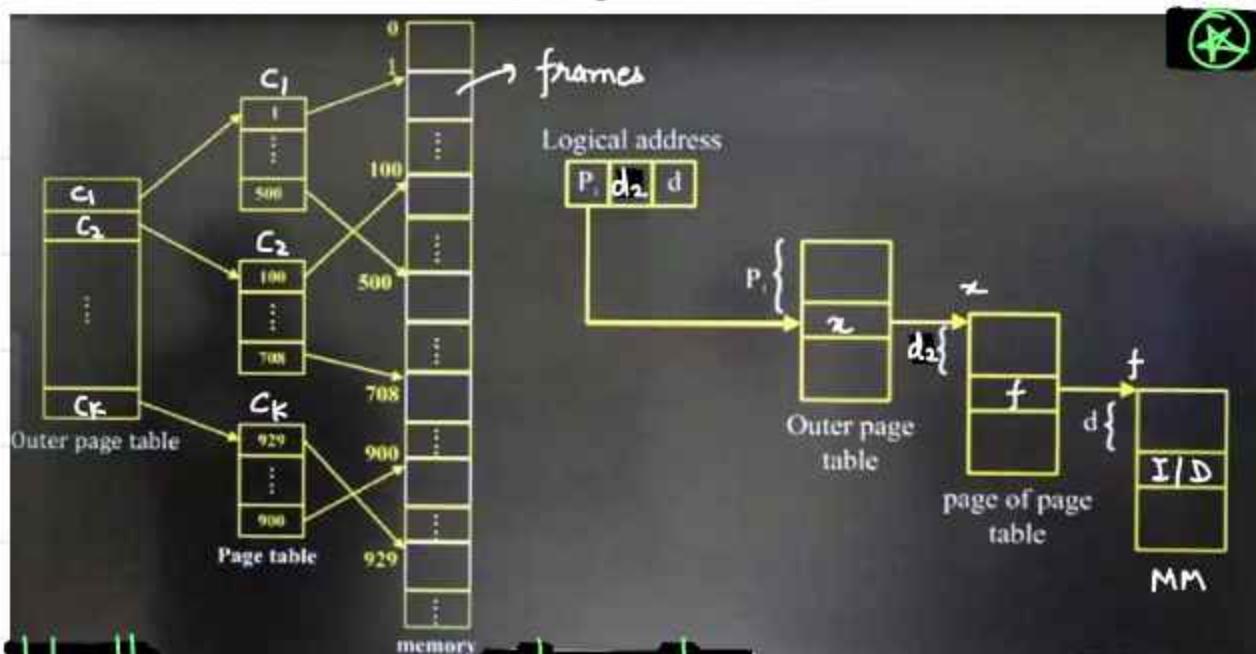
This is the crux of two level paging.



- (L)  $p_1 \rightarrow$  address of chunk of Inner Page table
- (A)  $d_1 \rightarrow$  address of frame of Main memory
- $d \rightarrow$  address of the data / Inst in the frame

Try to Remember these diagrams below & Then we will see another example.

# = REMEMBER =



Paging involves 3 steps

- Dividing the address space into pages<sup>14</sup>.
- Storing the pages in frames.
- Accessing the pages thru page table.

MOTIVE :- optimizing space , sacrificing time.

EMAT / :-  $2m \quad (m + m)$

One level Paging

EMAT / :-  $3m \quad (m + m + m)$

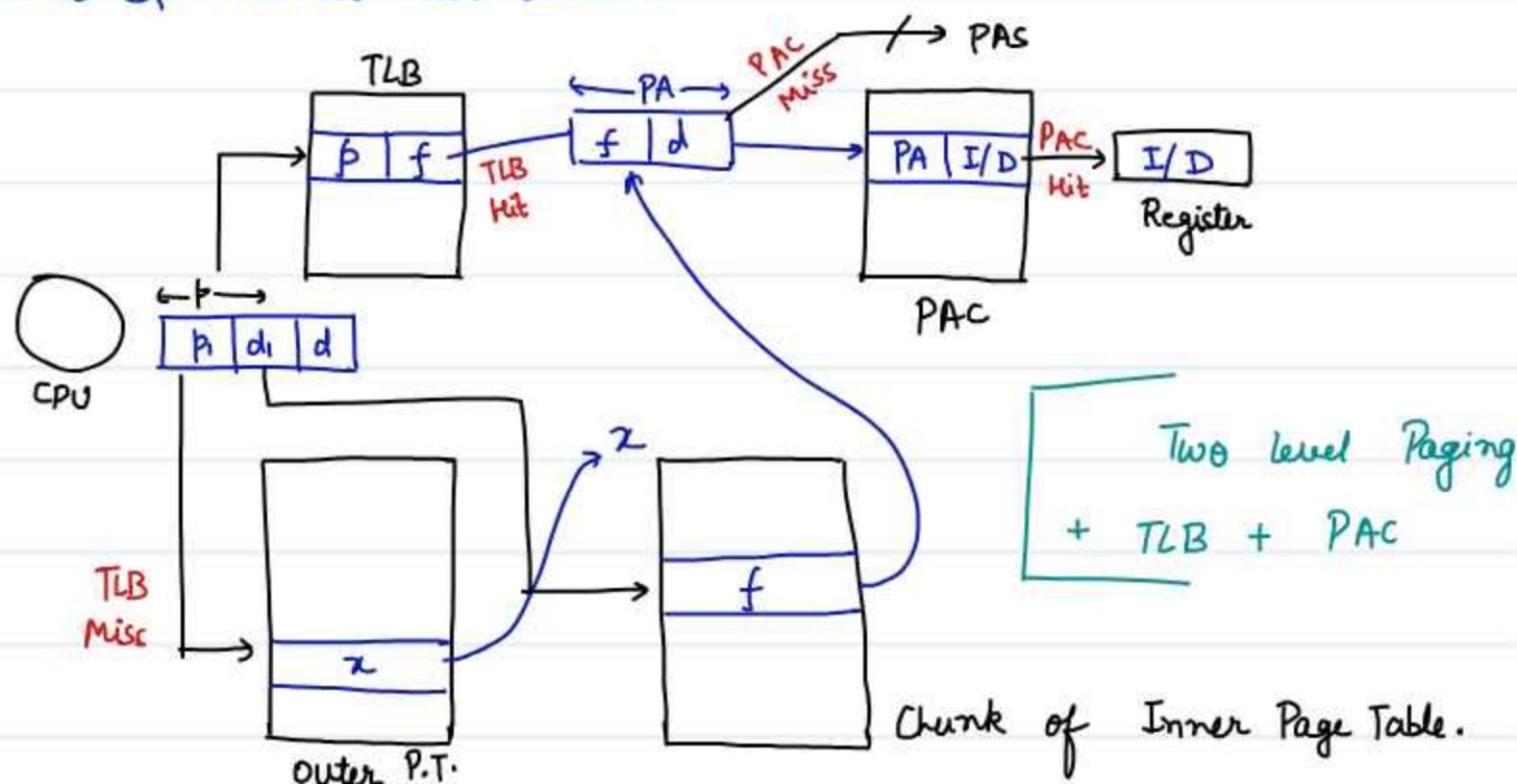
Two level

EMAT / :-  $(n+1)m$

n-level



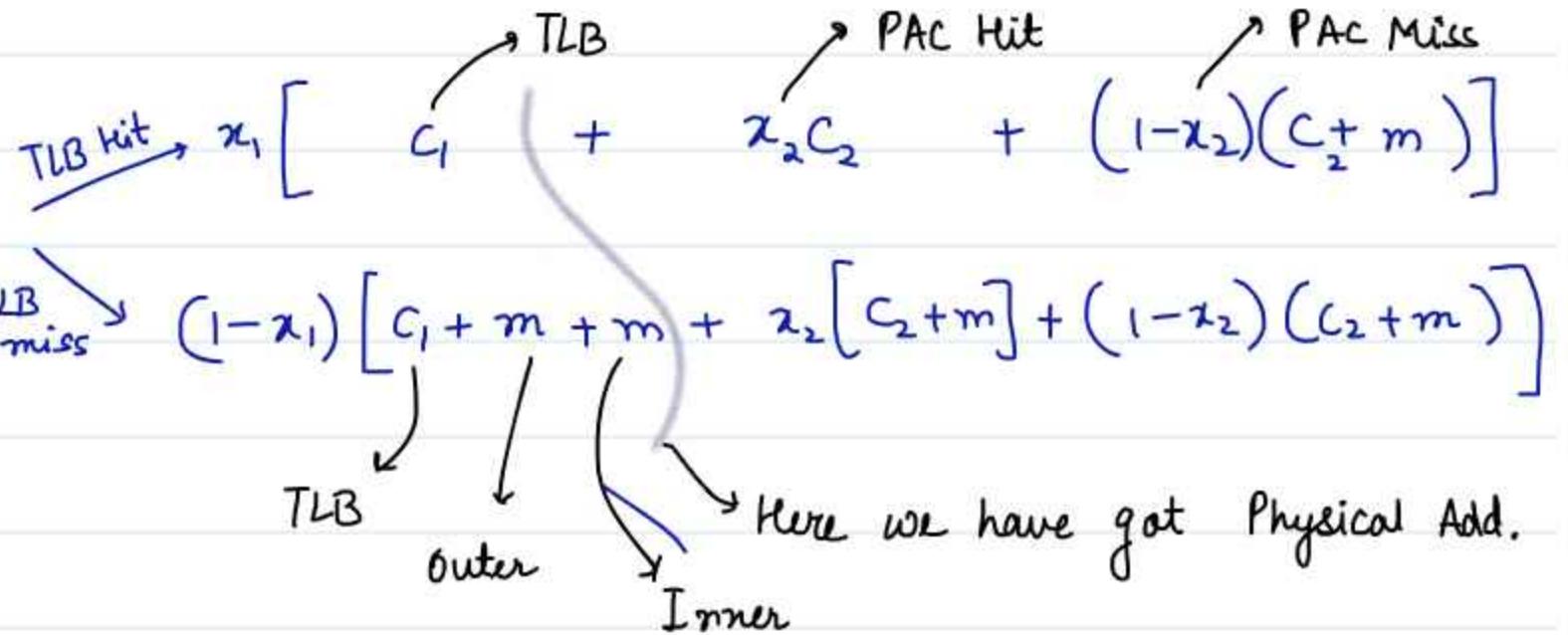
TLB & PAC in Two level :-



$$MMAT = m, TLBAT = C_1, TLB \text{ Hit Ratio} = x_1$$

$$PACAT = C_2, PAC \text{ Hit Ratio} = x_2$$

$$EMAT = ?$$



Ex-2

- Physical address space = 256 MB
- Logical address space = 4GB
- Frame size = 4KB
- Page table Entry = 2B

LAS  $\rightarrow$  Frame size



Level-1 Paging

$$N = \frac{LAS}{PS} = \frac{4GB}{4KB} = \frac{2^{30}}{2^{10}} = 1M$$

Dividing & trying to store it in 1 frame.

$$\text{Size of Page Table} = N \times e = 1M \times 2B - 2MB$$

Level-2  $\Leftarrow$

Paging

$>$  Frame size (4KB)

Size of outer Page Table = No of chunks  $\times$  entry

↓  
2B

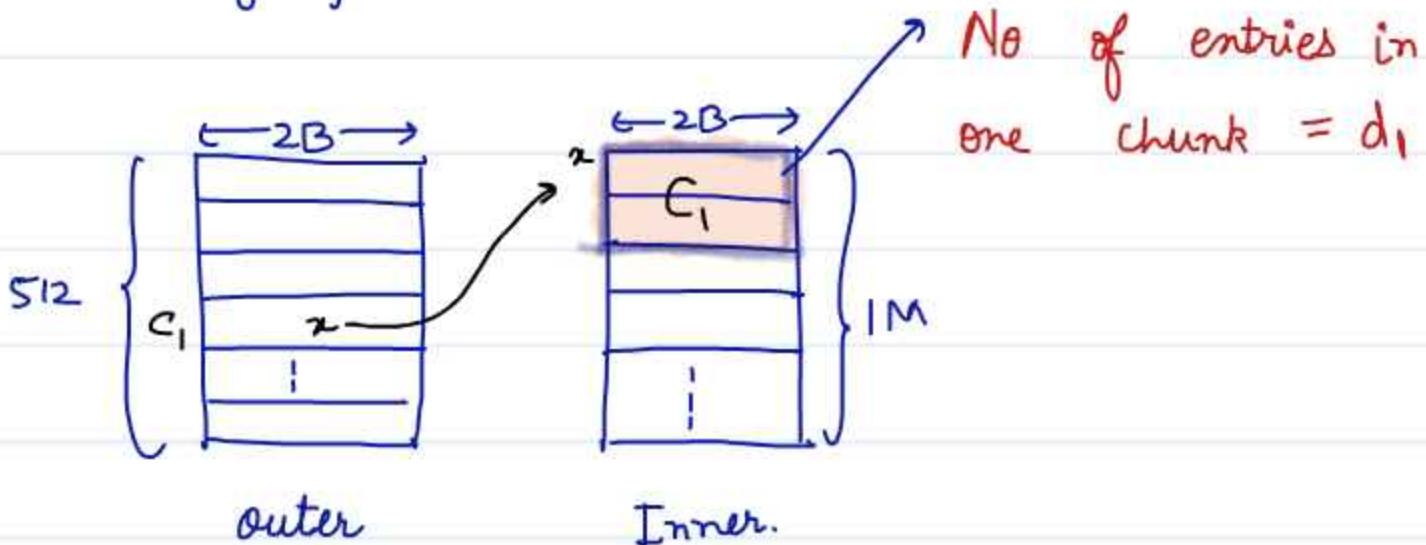
Divide such that one chunk can be stored in 1 frame.  
 Chunk size = 4KB stored in 1 frame.

$$\frac{2\text{MB}}{4\text{KB}} = 512$$

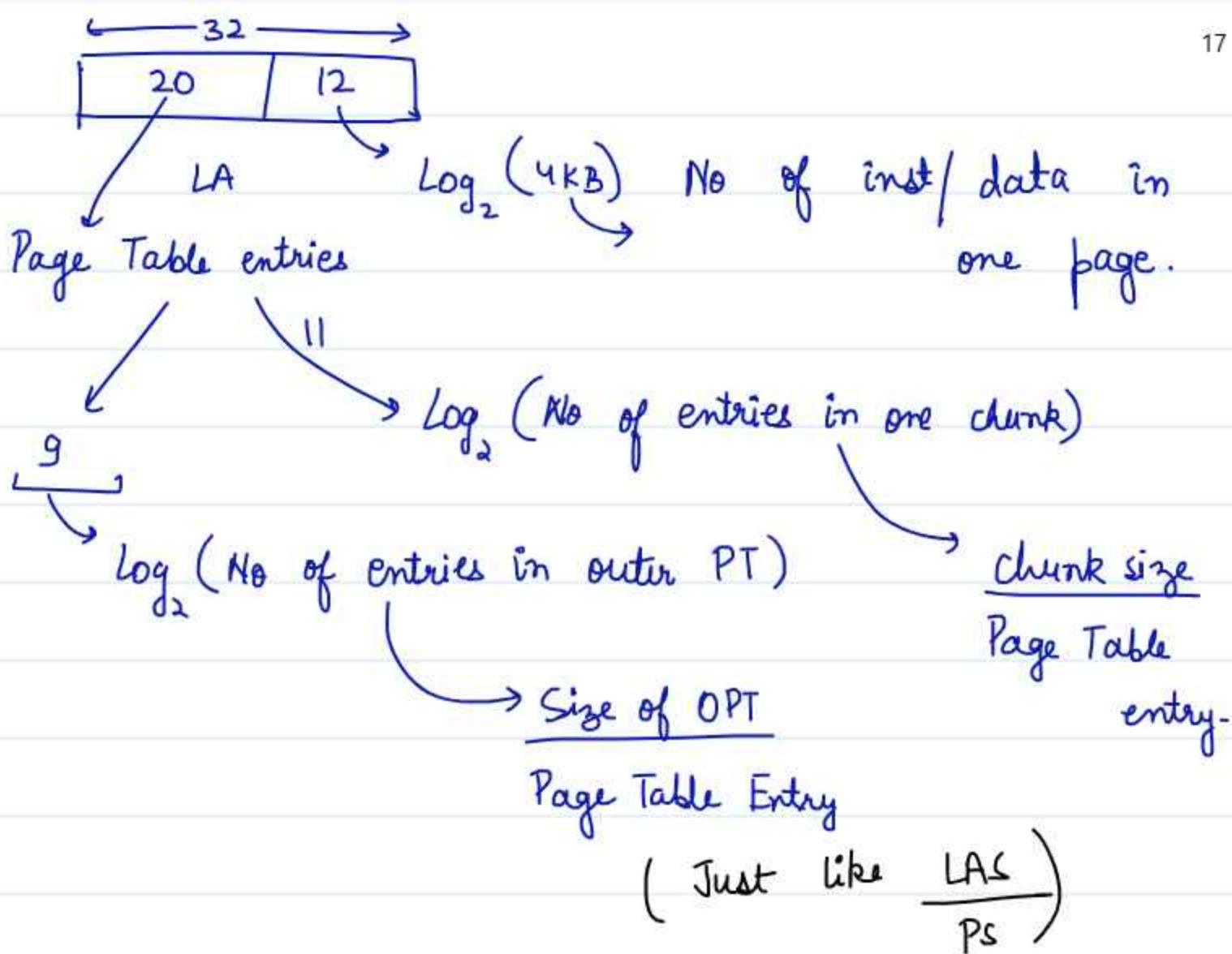
$$\text{Size} = 2^9 \times 2B = 1\text{KB}$$

Now  $1\text{KB} < \text{Frame size } (4\text{KB})$  ✓

Outer page table can be fit in one frame. No need to go further.



- Entry / <sub>OPT</sub> = Entry / <sub>IPT</sub> = e
- Frame size = Page size  $\neq$  Chunk size

Q

In the context of operating systems, which of the following statements is/are correct with respect to paging?

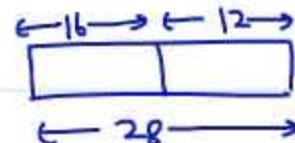
- Page size has no impact on internal fragmentation. X
- Paging helps solve the issue of external fragmentation. ✓
- Paging incurs memory overheads ✓ ] How?
- Multi-level paging is necessary to support pages of different sizes X

we didn't talk about this, we did MLP to optimize overheads.

Q Consider a System using 2 Level Paging Architecture. The Top level 9 bits of the Virtual Address are used to index into the outer Page Table. The next 7 bits of the Address are used to index into next level Page Table. If the size of Virtual Address is 28 bits. Then

- How large are the Pages and How many are there in Virtual Address Space?
- If P.T.E at both levels is 32 bits in size then what is the Space Overhead needed to translate Virtual Address to Physical Address of an Instruction or Data Unit?

$$p_1 = 9 \text{ bits} \quad d_1 = 7 \text{ bits}$$



$$e = 4B$$

space overhead ?

$$(i) \begin{cases} \text{Page size} = 2^{12} B = 4KB \\ N = 2^{(p_1 + d_1)} = 2^{16} \end{cases}$$

To convert logical Address to physical how much space is needed

$$(2^9 \times 4B) \leftarrow \begin{matrix} \text{Size of outer page table} \\ + \end{matrix}$$

Q. Is Page size = chunk size ?

NO, Not necessarily :- page size of the inner page table is not necessarily equal to the page size of the VAS. The page size of the inner page table depends on the number of entries it has, which in turn depends on the size of the VAS and the page size 1 2. However, the frame size in the main memory is equivalent to the page size 1 3. This is because the memory management unit (MMU) maps a page in the virtual memory to a page frame in the physical memory

Page of Inner Page Table

$$(2^7 \times 4B)$$

No of entries in entry one chunk size

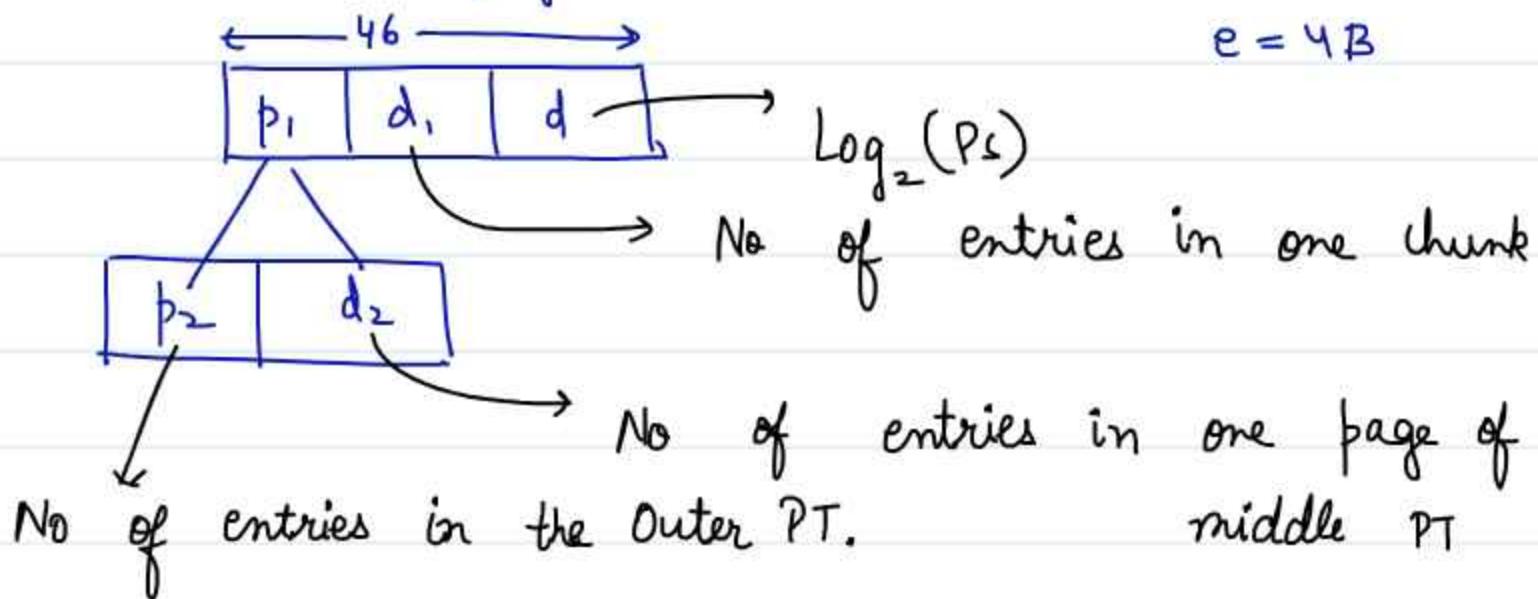
Q Is Page Table entry size same of OPT & IPT ?

Yes, the size of a page table entry is typically the same for both the outer and inner page tables 1 2 3 . This is because each entry in both tables contains the base address of a page frame 1 2 3 . The size of each entry is determined by the number of bits required to address each frame in the main memory 1 2 3

Q

Consider a Computer System using 3 Level Paging Architecture with a uniform Page Size at all levels of Paging. The size of Virtual Address is 46 bits. Page Table Entries at all levels of Paging is 32 bits. What must be the Page Size in Bytes such that the Outer Page Table exactly fits in one frame of Memory. Assume Page Size is power of 2 in Bytes. Show the Virtual Address format indicating the number of bits required to access all the three levels of Page Tables and the Page offset of Virtual Address Space.

3 level Paging architecture



$$\text{Page size} = \text{chunk size} = \text{Page of outer PT}$$

$$d = d_1 = d_2$$

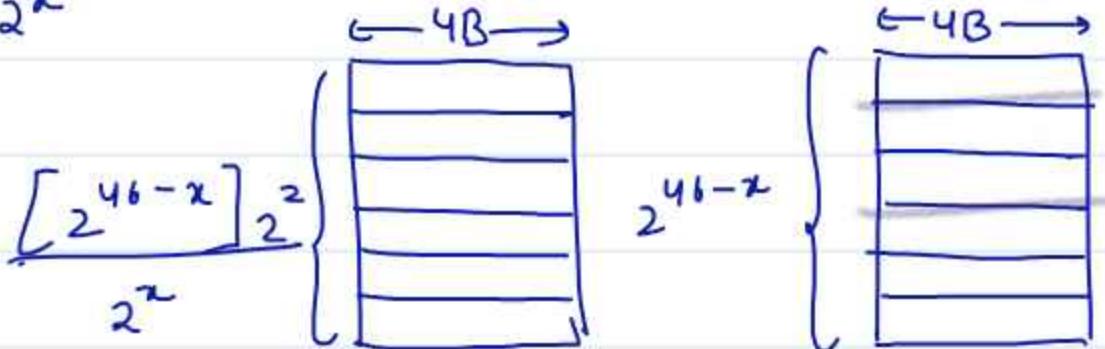
$Ps$  in Bytes [ Outer PT size = Frame size ]

$$\text{No of entries} \times 4B = \text{Page size} = 2^d$$

$$p_2 \times 4B = 2^d$$

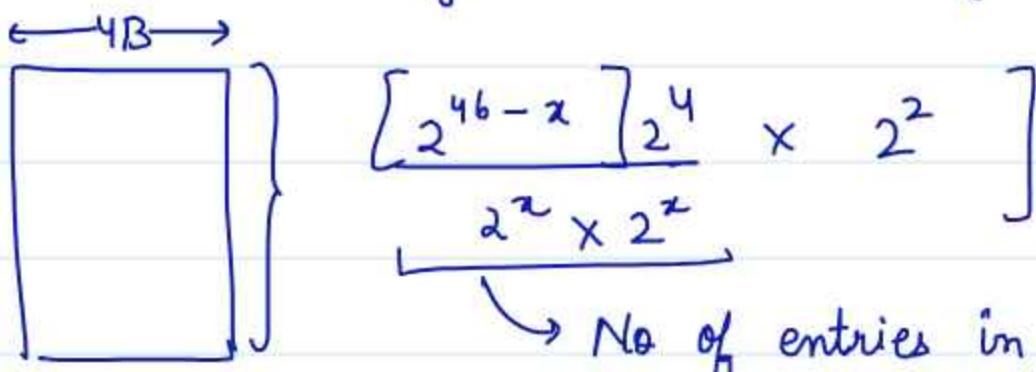
$$p = p_1 + d_1$$

$$\frac{2^{46}}{2^x} = p = \text{No of entries in Page table-1}$$



Page Table-2

Page Table-1



$$2^{52-3x} = 2^x \Rightarrow x = 13$$

let's generalize to solve things quickly :-

$$VAS = 2^5 \text{ Bytes} \quad e = 2^c \text{ Bytes} \quad \text{Page size} = 2^x$$

$$\hookrightarrow VA = 5 \text{ bits}$$

$$\text{Levels of Paging} = 1$$

Size of outer page table = ?

$$\frac{2^c \times \frac{2^{s+c-x}}{2^x}}{2^{s+2c-2x}} = \frac{2^c}{2^x}$$

l level :-  $2^{s+lc-1x}$

$$\text{Table 1} :- \frac{2^{s-x} \times 2^c}{2^{s+c-x}}$$

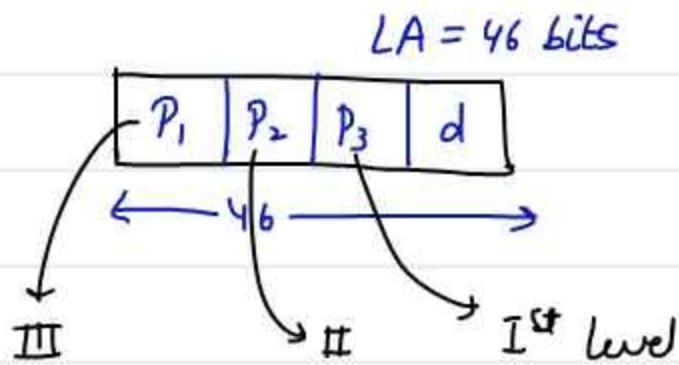
Ex :- VA = 46 bits

PTE = 4 Bytes

PS =  $2^x$  Bytes

l = 3

OPT = PS



Method - 2

$$d = x$$

Page size constant  $\left\{ \begin{array}{l} P_1 = P_2 = P_3 = y \\ 3y + x = 46 \end{array} \right.$

$$3y + y + x = 46$$

$y = 11$
$x = 13$

$$x = y + 2$$

Outer Page Table = Page size  
 $= 2^y \times 4B$

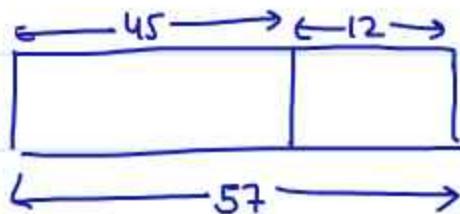
$$2^x = 2^{y+2} B$$

Q

Consider a Computer System with 57 bit Virtual Addressing, Using Multi-Level Tree Structured Page Tables with L Levels for Virtual to Physical Address Translation. Page size is 4KB and Page Table Entry is of 8 Bytes at all levels.

The value of L is \_\_\_\_\_.

$$e = 8B$$



L level Paging.

$$PS = 4KB$$

$$= 12 \text{ bits}$$

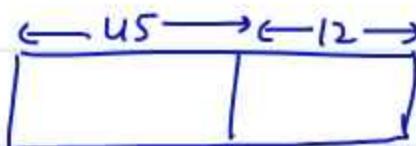
Size of outermost Table = Frame size

$$2^{57 + 12 - 12} = 4KB$$

$$2^{57 + 12(3-12)} = 2^{12}$$

$$57 + 12(-9) = 12$$

$$9L = 57 - 12 = 45 \Rightarrow L = 5$$



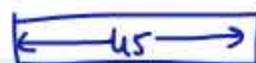
Method-2

$$PS = 4KB \quad PTE = 8B$$

No of entries that can be stored in one

$$\text{page} = 2^{12}/2^3 = 2^9$$

Every page table has  $2^9$  entries, 9 bits to access.

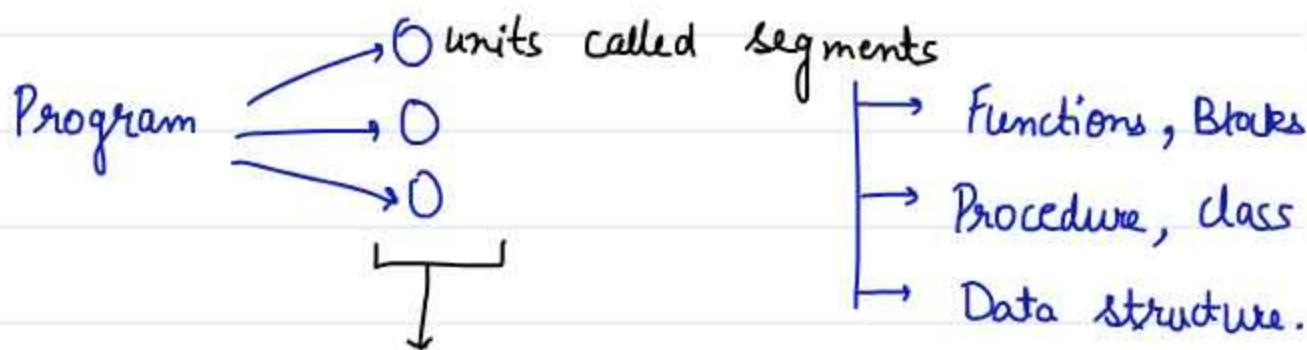


5 levels

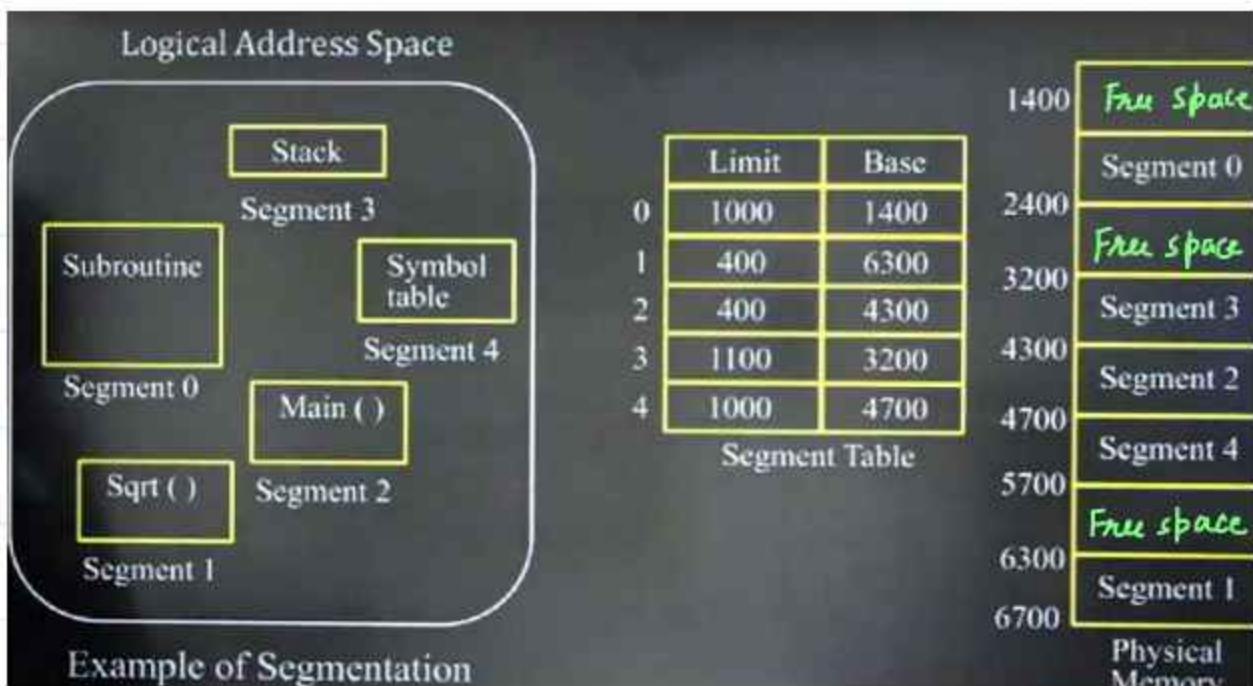
# MEMORY MANAGEMENT - 7

## # Segmentation #

- Paging doesn't preserve user's view of memory allocation.
- As per user's view of memory allocation



- These segments are assumed to be stored at their entirety @ Non Contiguous locations in memory.



Concept is similar to paging but Pages → same Seg → diff Size

Seg. may be of same size.

- The whole segment is in one place in the memory but segments are at non contiguous locations.

↓  
This view is not preserved by Paging.

- Main() → 5KB & Page size = 1KB then main f" itself will be distributed non contiguously.

But in segmentation, no matter how big the segment is, will be stored entirely at one location.

- There is no concept of frames in segmentation wherever free hole is present store the segment. It's like variable partitions.
- There we had page tables, here we have segment tables (S.T.) :-

$$\text{No of entries in ST} = \text{No of segments}$$

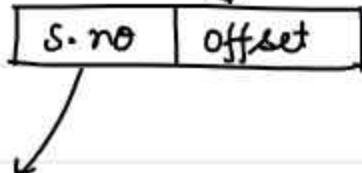
S.T. are stored in MM.

Limit	Base
1000	1400

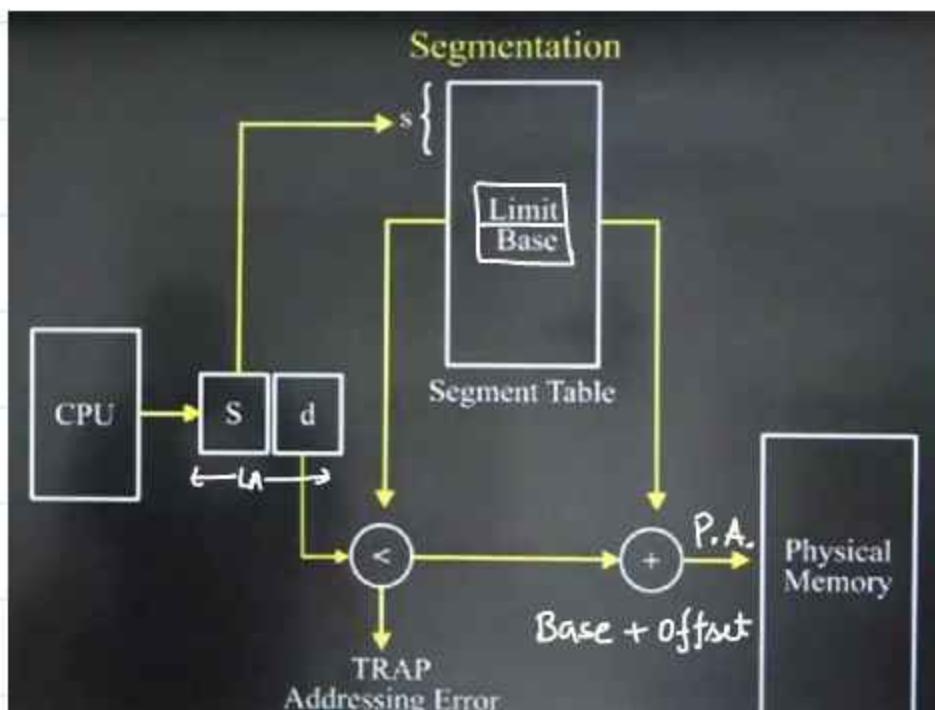
Starting address

Size ← →

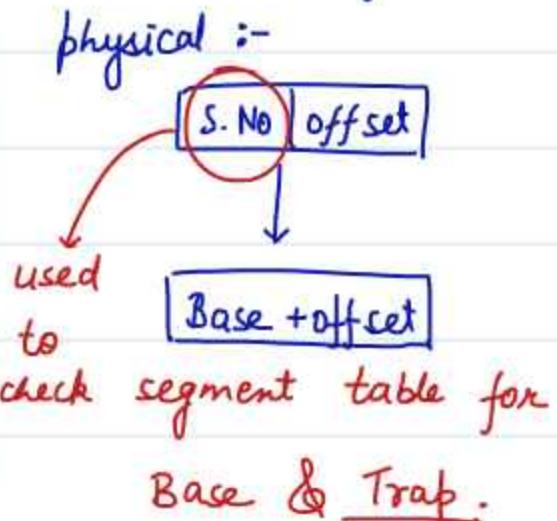
Logical Address  $\Rightarrow$



Out of these 1000 words  
which word do you  
want to read  
(offset  $\leq 1000$ )



To convert logical to physical :-



used to  
check segment table for  
Base & Trap.

Trying to access the word beyond the segment.

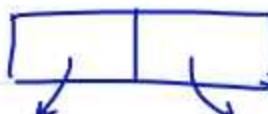
Q

Consider the following segment table:

Segment	Base	Length
0	1219	600
1	3300	14
2	90	100
3	2327	580
4	1952	96

What are the physical addresses for the following logical addresses?

- a) 0, 4302 > 600  $\rightarrow$  TRAP  
 b) 1, 15 > 14 : TRAP  
 c) 2, 50 : 90 + 50  
 d) 3, 400 : 2327 +  $^{400}$  E) 4, 112 : TRAP

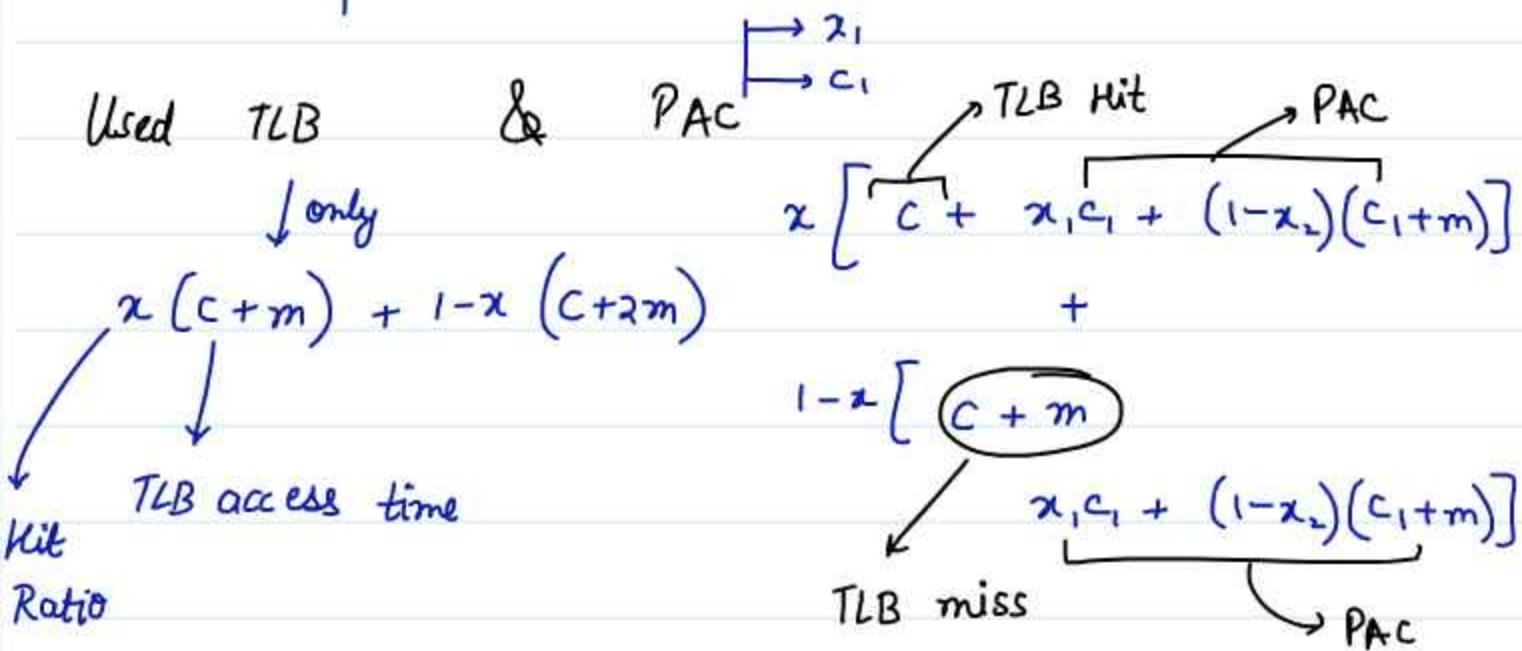


segment no. offset

P.A. :  $[B + off]$

# Performance  $\begin{cases} \rightarrow \text{Time} \\ \rightarrow \text{Space} \end{cases}$

Time :-  $2m$  (  $m$  for segment table +  
 $m$  for main memory )  
 $\rightarrow \textcircled{+} \& \textcircled{<} \text{ takes negligible time.}$



Space :- If segment table becomes too large apply  
 paging on segment table.

Segmented Paging → select some P.S. & divide ST into pages.  
 → store the pages in frames  
 → Access them through page tables.

## PAGING Vs SEGMENTATION wrt Fragmentation

↓  
 Internal at Last Page

↓  
 External at PAS

	Int F	Ext F
Paging Segment	✓ (Last Page)	X
	X	✓ In PAS

organized like variable partition

## "External Fragmentation in Segmentation"

Less ← Compaction /

Diagram

Defragmentation

Segmented Paging

- Reduces size of S.T.
- Reduces Ext. Frag.

fork system call

Threads & MT

Monitors

Inverted, Shared

& segmented Paging

will study in the

last section of course

## # VIRTUAL MEMORY #

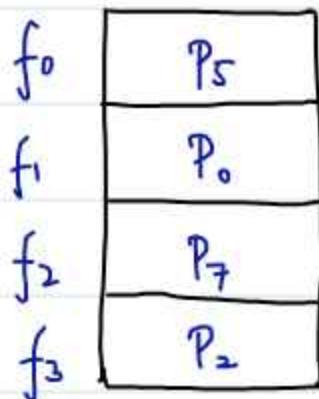
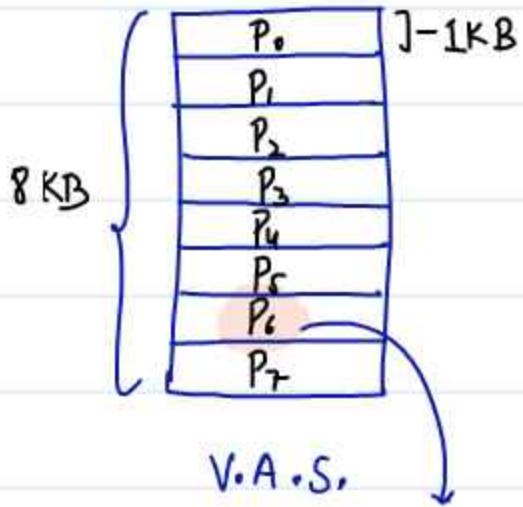
→ gives an illusion to programmer that huge amount of memory is available for

executing program that's larger than available physical memory.

$$V.A.S = 8KB$$

$$P.A.S = 4KB$$

$$P.S. = 1KB$$



what if I want to refer P<sub>6</sub>?

V.M. is implemented

through ↗

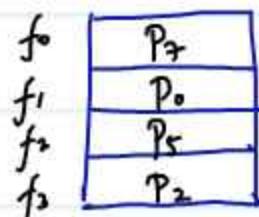
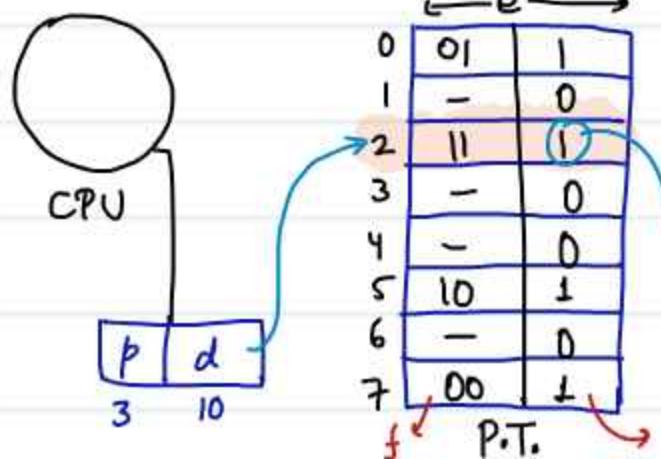
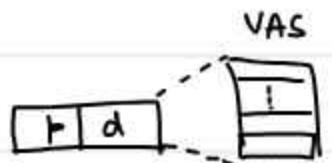


How will this work?

## # Demand Paging #

Loading the pages on demand from disk to Mem.

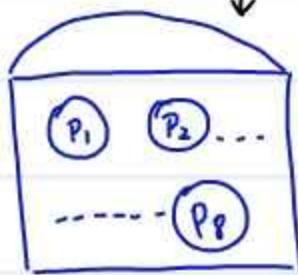
- Virtual Memory is mapped on disk.

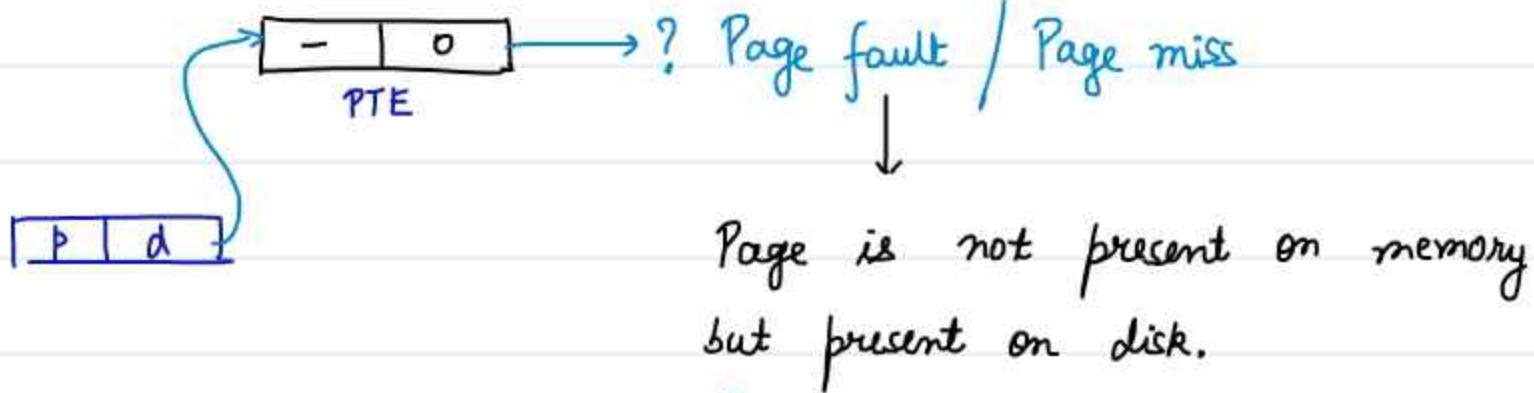
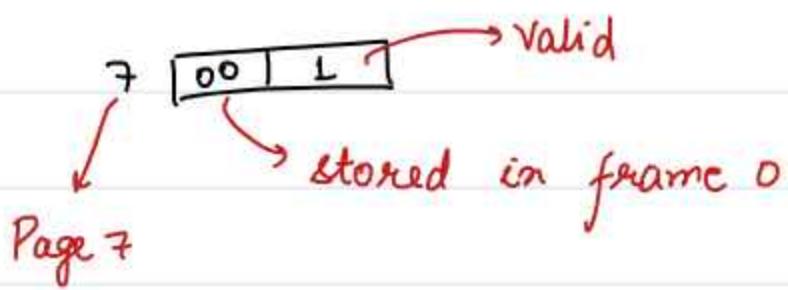


Page hit  $[11|d] = PA$

PAS

Valid / Invalid





Page is not present on memory  
but present on disk.

What will happen now?

- ① Process causing page fault will get blocked. ] → we have to read the page from disk

↓  
Mode shifting

↓

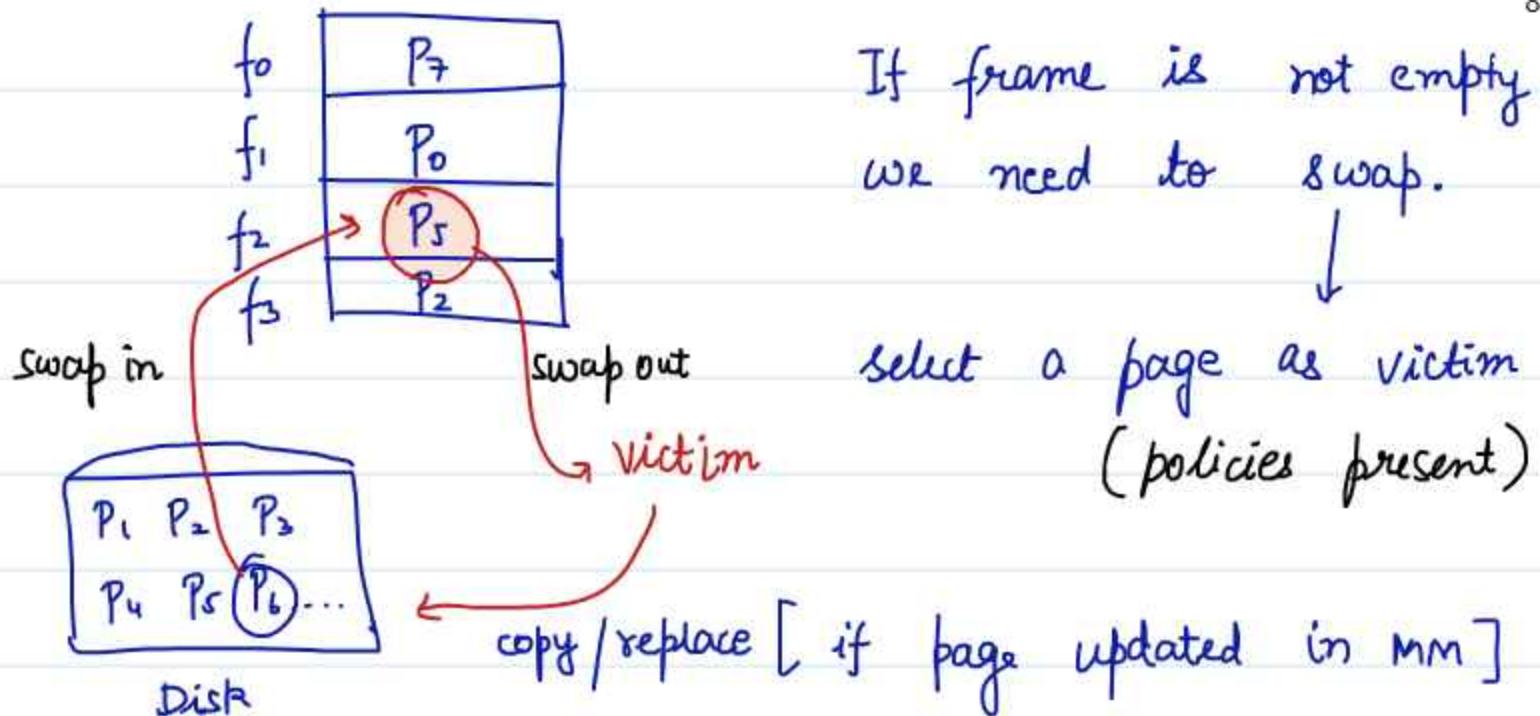
I/O operation

↓

- ② VMM will be on CPU taking Block

charge, summoning Disk Manager to find the required page & handover the copy of that page to VMM

- ③ Now VMM will try to save that copy to MM
- unblock the Pi ← save ← frame is empty ←
- Ready state → continue ? ← frame is not empty ←



This is how demand paging is done, all OS follow this approach.

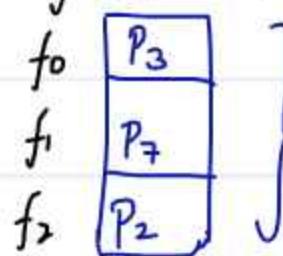
win, MAC, unix

## Demand Paging

Pure (default)

Executing of the process will start with empty frames.

## Prefetch Demand Paging



} we start with loaded pages

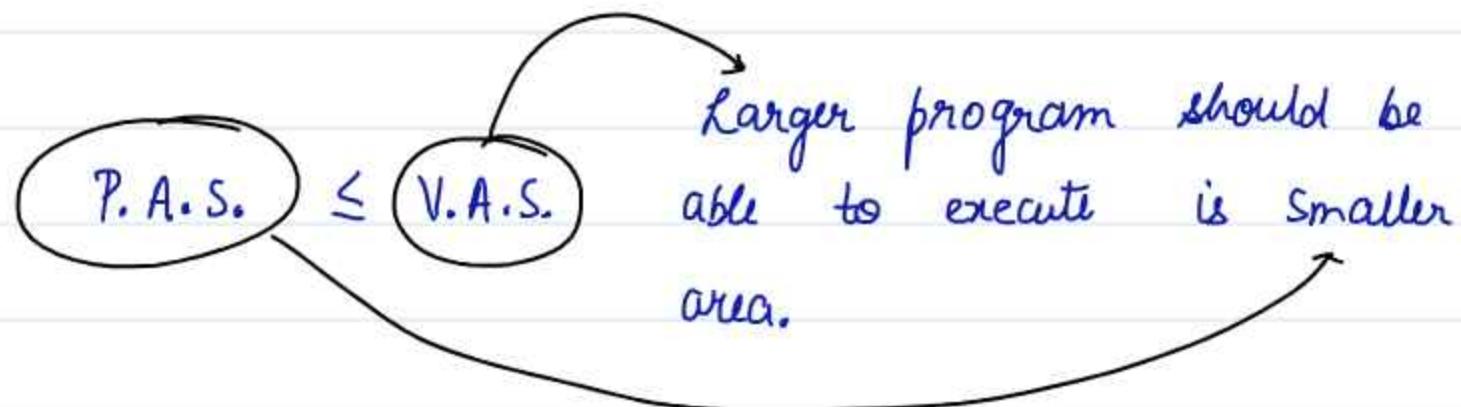
NOTE :- Page fault is like an interrupt result from changing the mode from user to Kernel.

In implementation of VM

P.A.S

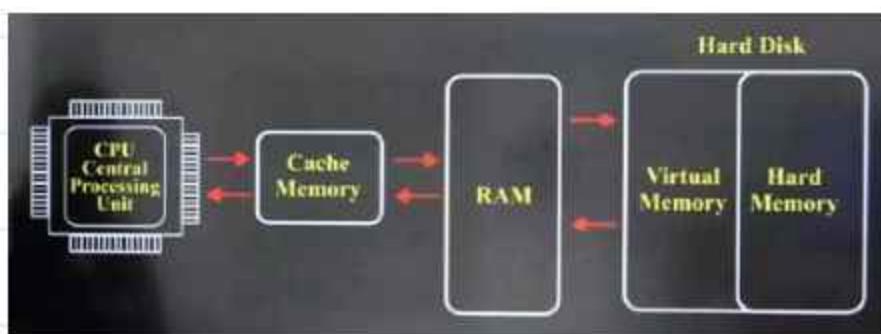
V.A.S (VM)

Disk A.S.

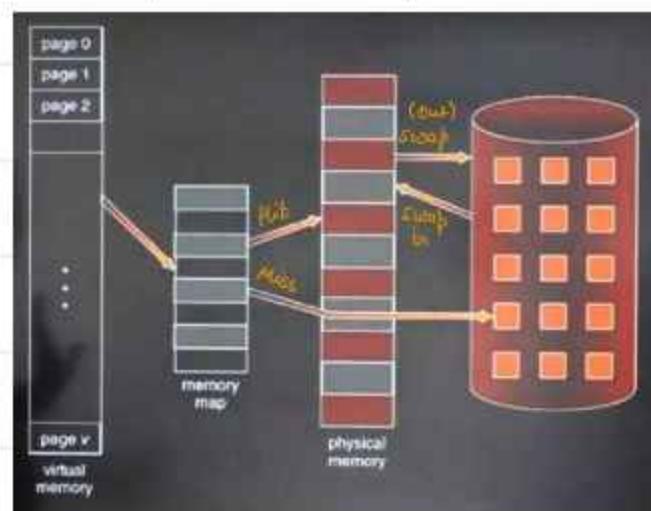
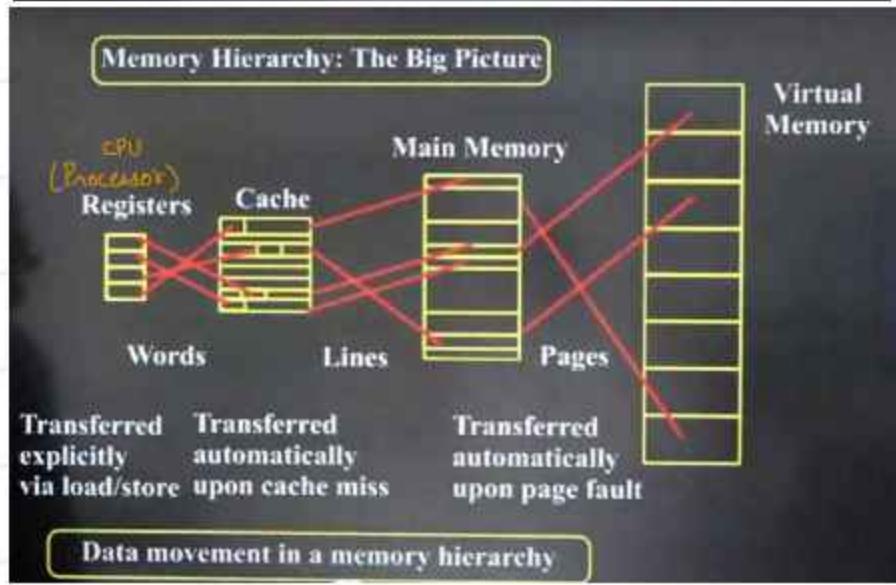


$PAS \leq VAS \leq D.Add\ space$  ] size of VM is limited by disk size

stored on disk, so DAS should be large enough.

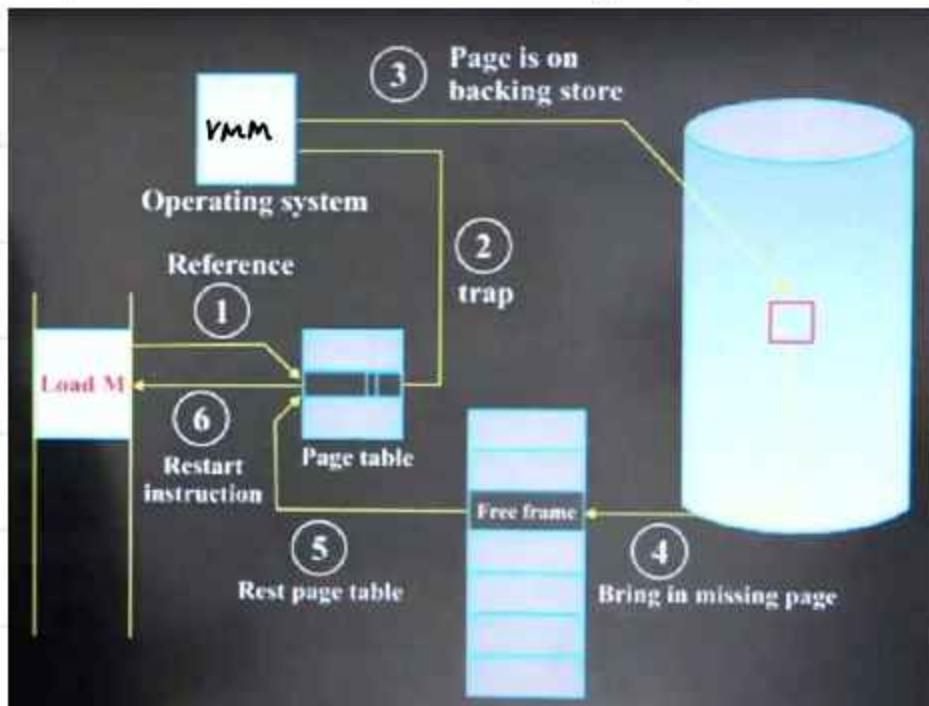


Conceptual View of VM



# Steps Taken to Serve Page fault.

10



$PFST$  :- Amount  
of time taken by  
OS to serve the  
page fault.

↳ Generally in ms

Memory AT ~ ns

## # Performance of VM #

$$① \text{ Time : } MMAT = m$$

$$PFST = s \quad [s \gg m]$$

$$\text{Page fault rate} = p \in [0, 1]$$

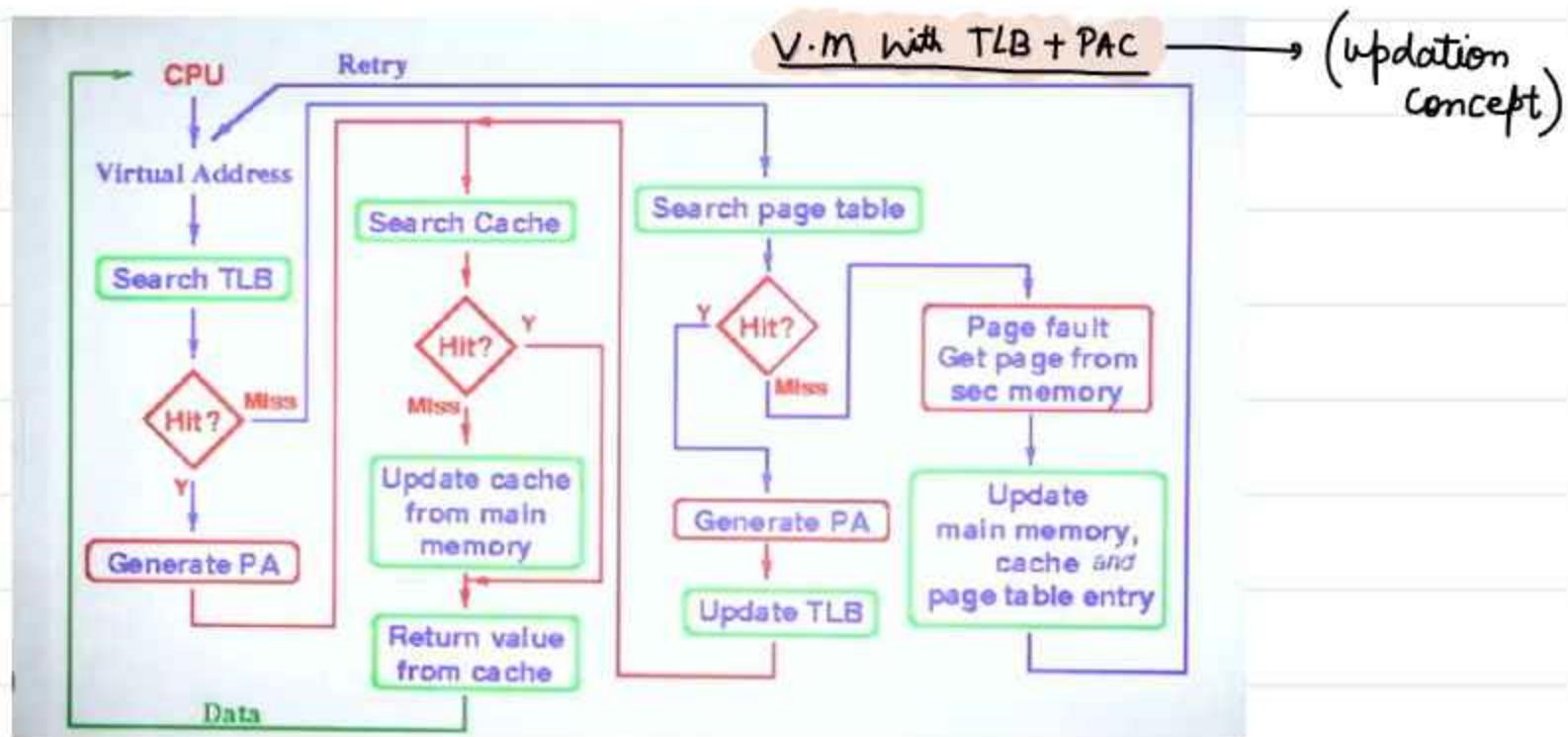
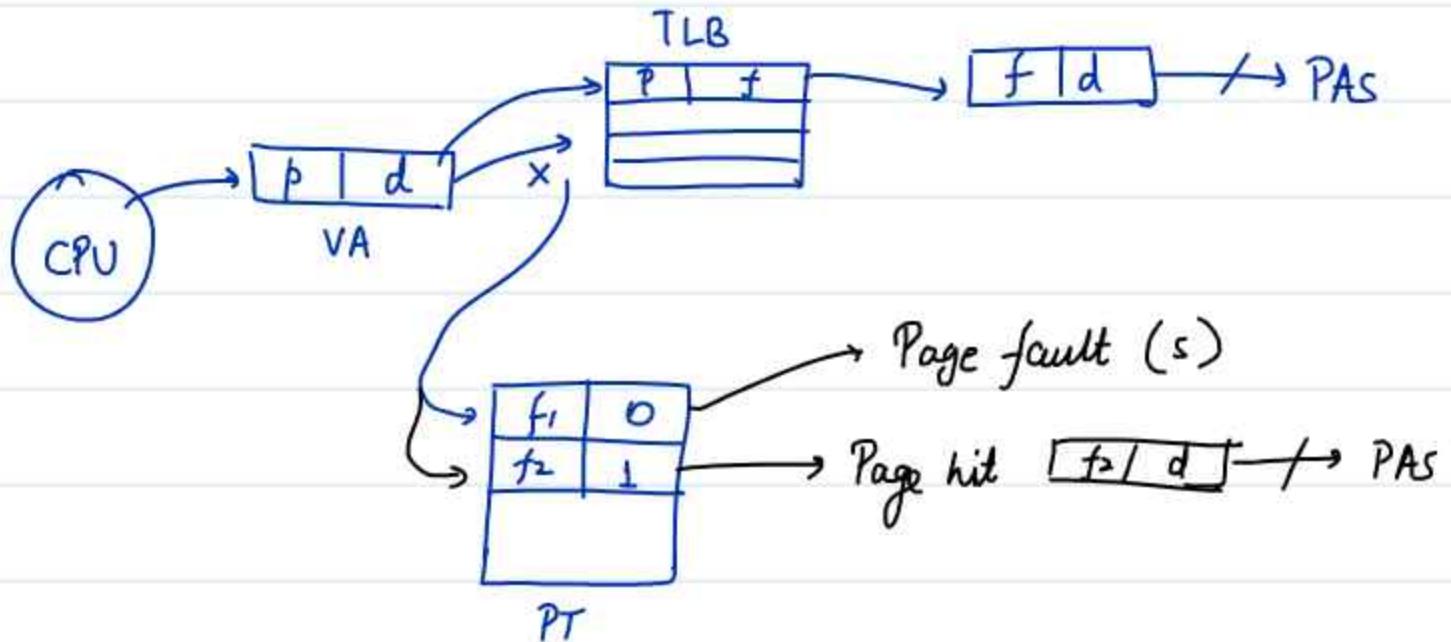
$$\text{Page hit Rate} = 1 - p$$

$$\begin{aligned} EMAT / \\ \text{Demand Paging} &= (1-p) \times m + p \times s \\ &\quad \text{Hit} \qquad \qquad \qquad \text{Miss} \end{aligned}$$

If I used TLB + Page swap then :-

$$\begin{aligned} EMAT / \text{demand paging} &= x(c + m) + 1 - x [c + (1-p)2m + p(mts)] \end{aligned}$$

$$\begin{aligned}
 x &\rightarrow \text{TLB Hit} & c+m &\rightarrow \text{TLB Hit + PAS} \\
 1-x &\rightarrow \text{TLB Miss} & (1-\phi)2m &\rightarrow \text{Page Hit + PAS} \\
 && \phi(m+s) &\rightarrow \text{Page miss + PFST}
 \end{aligned}$$



NOTE :- One level Page table is less desirable becoz of the large memory overhead in maintaining P.T.

# MEMORY MANAGEMENT - 8

Q

Suppose the time to service a Page fault is on the average 10 milliseconds, while a Memory Access takes 1 microsecond. Then a 99.99% Hit ratio results in Average Memory Access Time

$$\text{AMAT} = \frac{p \times s}{100} + \frac{(1-p) \times m}{100}$$

$$= \frac{0.01 \times 10\text{ms}}{100} + \frac{0.9999 \times 1\text{\mu s}}{100}$$

$$= 1\text{\mu s} + 0.9999 \times 1\text{\mu s}$$

$$= \underline{1.9999\text{\mu s}}$$

Q

If an Instruction takes 'i' microseconds and a Page Fault takes an additional 'j' microseconds, the Effective Instruction Time if on the average a page fault occurs every 'k' instructions is \_\_\_\_\_.

Instruction = i

Page fault rate =  $1/k$

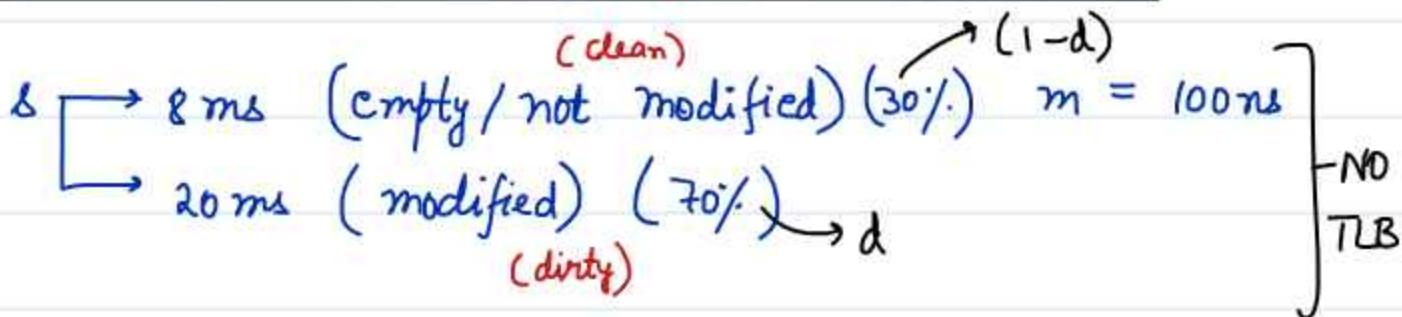
Page fault =  $i+j$  ] - every k

$$\begin{array}{l} 1 \rightarrow i \\ 2 \rightarrow i \\ \vdots \\ k^{\text{th}} \rightarrow i+j \end{array} \quad \left. \right] \rightarrow \frac{i+i+\dots+i+j}{k} \Rightarrow \frac{ki+j}{k}$$

$$= i + \frac{j}{k}$$

OR can also do like :-  $\frac{1}{k}(i+j) + \left(1 - \frac{1}{k}\right)i$

Q Assume that we have a Demand-Paged memory. It takes 8 milliseconds to service a page fault if an empty frame is available or if the replaced page is not modified, and 20 milliseconds if the replaced page is modified. Memory-access time is 100 nanoseconds. Assume that the page to be replaced is modified 70 percent of the time. What is the acceptable page-fault rate for an effective access time of no more than 2000 nanoseconds?



$$\text{Page fault rate} = ? \quad \text{EMAT} < 2000 \text{ ns}$$

$$\text{EMAT} = p \times \left[ 8 \times 0.3 + 20 \times 0.7 \right] + (1-p) 100 \text{ ns}$$

$$p [2.4 + 14] \times 10^6 \text{ ms} + (1-p) 100 \text{ ns} < 2000 \text{ ns}$$

$$16.4p \times 10^4 + (1-p) < 20$$

$$164000p - p < 19$$

$$p < \frac{19 \times 10^{-3}}{164} \quad \boxed{0.1} \quad 10^{-4}$$

$$\boxed{p \approx 0.01 \%}$$

Q Consider a process executing on an operating system that uses demand paging. The average time for a memory access in the system is M units if the corresponding memory page is available in memory, and D units if the memory access causes a page fault. It has been experimentally measured that the average time taken for a memory access in the process is X units. Which one of the following is the correct expression for the page fault rate experienced by the process?

$$(D - M)/(X - M)$$

$$m = M$$

$$(X - M)/(D - M)$$

$$\text{Page fault} = D \text{ (total)}$$

$$(D - X)/(D - M)$$

$$\text{Avg. Time taken} = X \text{ unit}$$

$$(X - M)/(D - X)$$

$$X = (1 - p)m + p \times D$$

$$X = m - mp + Dp$$

$$\frac{X - m}{D - m} = p$$

Q Consider a paging system that uses 1-level page table residing in main memory and a TLB for address translation. Each main memory access takes 100 ns and TLB lookup takes 20 ns. Each page transfer to/ from the disk takes 5000 ns. Assume that the TLB hit ratio is 95%, page fault rate is 10%. Assume that for 20% of the total page faults, a dirty page has to be written back to disk before the required page is read in from disk. TLB update time is negligible. The average memory access time in ns (round off to 1 decimal places) is \_\_\_\_\_.

$m = 100 \text{ ns}$

$m = 100 \text{ ns}$

$c = 20 \text{ ns}$

Page Transfer = 5000 ns  
(one side)

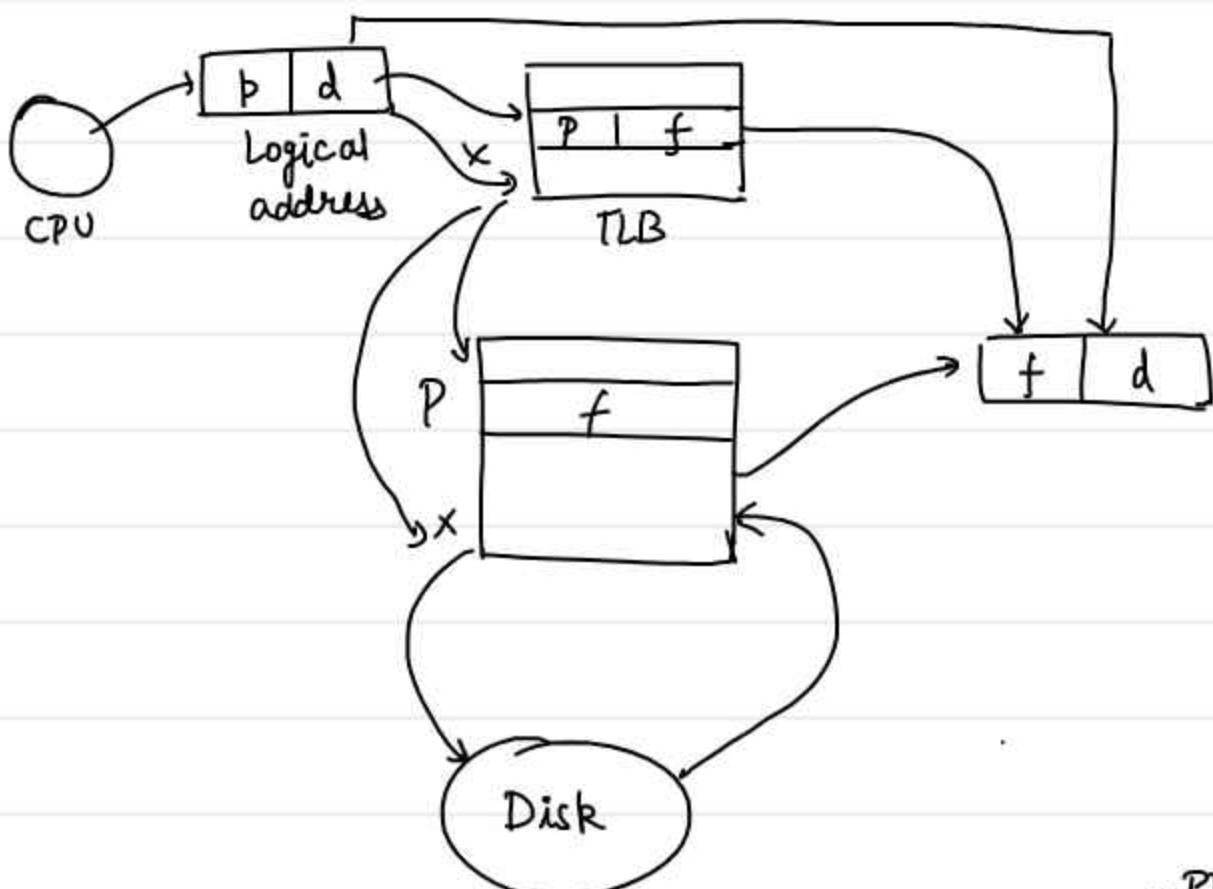
$h = 0.95$

$p = 0.1 \text{ (page fault)}$

TLB hit

$d = 0.2 \text{ (dirty)}$

AMAT = ?



$$\text{EMAT} := x[c+m] + (1-x)[c + (1-\beta)(m+m) + \beta[m + d \times 2t + (1-d)t]]$$

Annotations for the EMAT formula:

- $x[c+m]$ : **TLB hit**
- $(1-x)$ : **TLB miss**
- $c$ : **Page fault**
- $(1-\beta)(m+m)$ : **Page table**
- $\beta[m + d \times 2t + (1-d)t]$ : **transfer time**
- $\beta$ : **Page dirty**
- $m$ : **memory**
- $d$ : **Page table**
- $t$ : **transfer time**

$$\begin{aligned}
 & 0.95 [20 + 100] + 0.05 [20 + (1-0.1)(200) + 0.1 \\
 & \quad [100 + 0.2 \times 10k \text{ ns}]] \\
 & 170 \times 0.95 + 0.05 [20 + 180 + 610] \\
 & \quad \downarrow \\
 & 114 + 0.05 [810] = \underline{\underline{154.5 \text{ ns}}} \\
 & 40.5
 \end{aligned}$$

Page clean :- 1 Disk operation

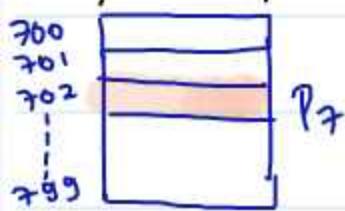
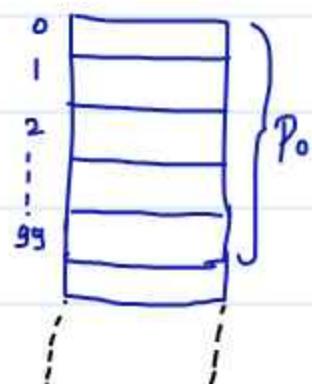
Page dirty :- 2 Disk op.

# # Page Replacement #

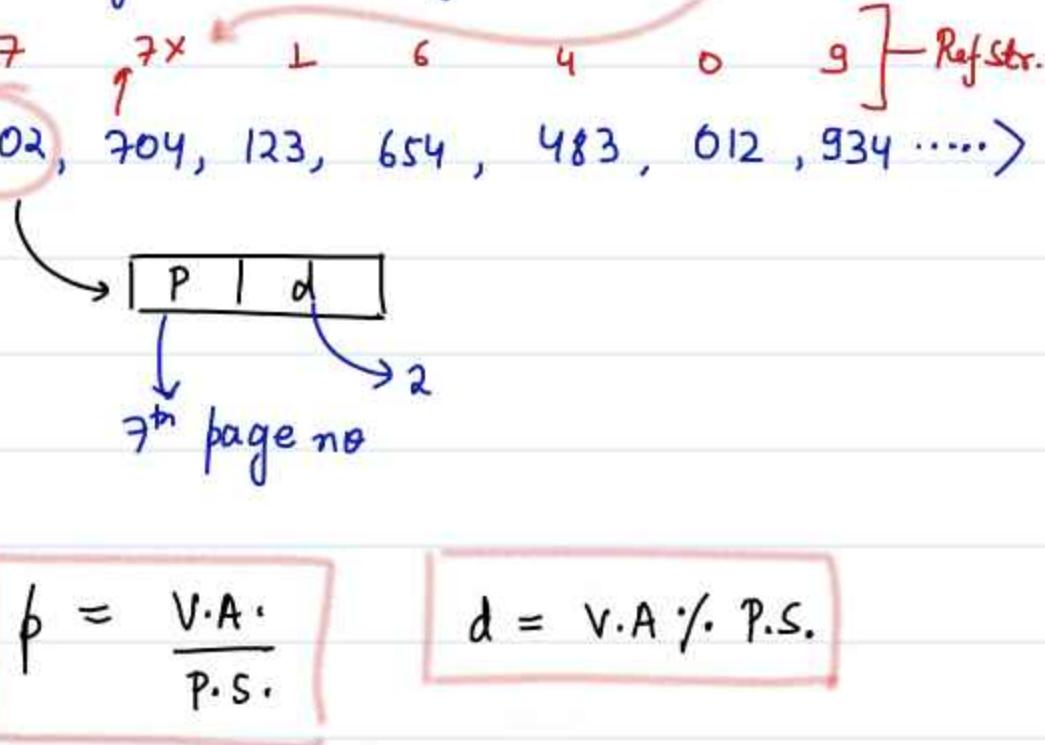
a) Reference string :- set of successively unique pages referred in the given list of VA's.

V.A. gen :-  $\langle 702, 704, 123, 654, 483, 012, 934 \dots \rangle$

$$P.S = 100 B$$



Page Table



Why successfully unique?

we referred a page, page fault occurred, we brought that page into memory, now we are referring that again, page fault can't occur.

Ex:-

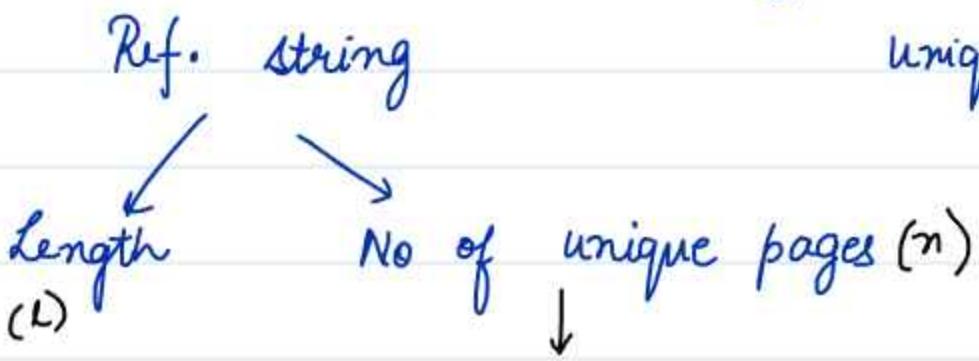
$P.S = 100 B$

V.A's :-  $\langle 654, 203, 122, 608, 706, 205, 986, 555 \dots \rangle$

Ref Str.  $\langle 7, 7x, L, 6, 4, 0, 9 \rangle$

Ex :-  $L = 8$

6



unique = 6

indicate size of process in terms  
of no of pages.

b) Frame Allocation Policies :-

$n$  : no of Processes

$s_i$  : Demand (frames) for Process  $i$

$D$  : Total demand  $\sum_{i=1}^n s_i$

$M$  : Available frames

$a_i$  : Allocated frames to  $P_i$

$$a_i \leq s_i$$

Ex :-  $n = 5$   $\langle P_1, P_2, \dots, P_5 \rangle$ ;  $M = 40$

$P_i$      $s_i$     Equal Alloc.     $\rightarrow$  Should be used

$P_1$     10              8 } when all processes

$P_2$     5              8 } Not have almost equal

$D = 80$      $P_3$     35              8 } Justified demand, in this

$P_4$     18              8 } case, demand is

$P_5$     12              8 } wtf! varying.

$M > D$   
 If Available > Demand then there is no point of allocation policy (decision making)

(ii) Proportionate Allocation :-  $a_i = \left( \frac{s_i}{D} \right) \times M$

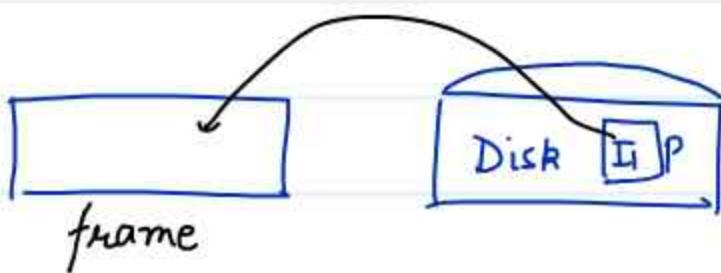
	$P_1$	$P_2$	$P_3$	$P_4$	$P_5$
$s_i$	10	5	35	18	12
$a_i$	5	3	17	9	6

(iii) 50% Rule :- Whatever the process ask give  $\frac{1}{2}$  of that

Minimum no of frames allocated to process = ?

process can't execute without them.

process should be able to execute minimum one Instruction.



So for one instruction shouldn't we need just 1 frame?



Addressing mode tells us how to  
read operands

### Instruction architecture

operands can be distributed in multiple pages.



so it depends on instruction architecture.

**Q.** The minimum number of page frames that must be allocated to a running process in a virtual memory environment is determined by

A. The Instruction set Architecture  
 B. Page size  
 C. Physical memory size  
 D. Number of processes in memory

Dots  $\Rightarrow$  Page Fault

Q Ref string :-

(7, 0, 1, 2, 0, 3, 0, 4, 2, 3, 0, 3, 2, 1, 2, 0, 1, 7, 0, 1)

$$l = 20$$

$$n = 6$$

Pure DP

7
0
1

} now  
who will

become victim?



FIFO

stored on disk in form  
of six pages.

It'll ask for 6 frames.



Given = 3 frames

<del>7</del>	2	2	<del>2</del>	4	4	<del>4</del>	0	0	<del>0</del>	7	7	7
0	<del>0</del>	3	3	<del>3</del>	2	2	<del>2</del>	1	1	<del>1</del>	0	0
1	1	<del>1</del>	0	0	<del>0</del>	3	3	<del>3</del>	2	2	<del>2</del>	1

No of Page Faults  $\Rightarrow 15$  } when no of page = 3

$$\text{Page faults rate} = \frac{15}{20} = \underline{\underline{\frac{3}{4}}}$$

W.W. (If no of pages  $\Rightarrow 4$  then rate will  $\downarrow$ )

# Memory Management - 9

Q Ref string :-

$\langle 7, 0, 1, 2, 0, 3, 0, 4, 2, 3, 0, 3, 2, 1, 2, 0, 1, 7, 0, 1 \rangle$

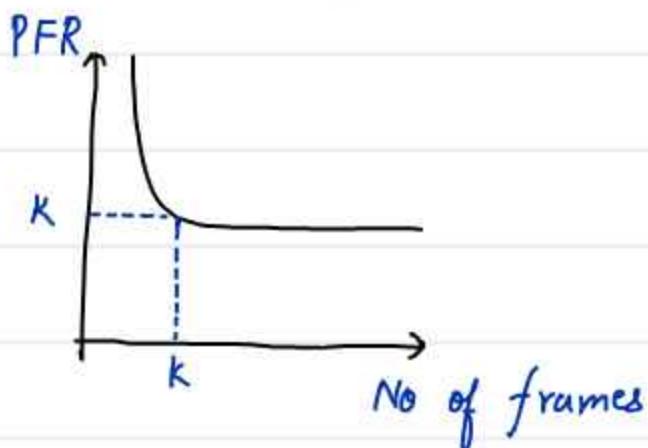
a)  $l = 20$        $n = 6$

FIFO  $\rightarrow$  3 frames  $\Rightarrow 15$       4 frames  $\Rightarrow 10$

b)  $\langle 1, 2, 3, 4, 1, 2, 5, 1, 2, 3, 4, 5 \rangle$        $l = 12$ ;  $n = 5$

3 frames  $\Rightarrow 9$

4 frames  $\Rightarrow 10$  !!



It increased against the natural characteristics.

ANAMOLY

In case-a if I took 6 frames  $\rightarrow$  6 page faults

(Initial ones due to Pure DP)

7 frames  $\rightarrow$  6

8 frames  $\rightarrow$  6

The above's anomaly was discovered by Bellady.

Bellady's Anomaly :- As the no of frames allocated to process ↑, page fault also ↑ sometimes.



only FIFO & LIFO Based!

Reason :- STAC property of Replacement

# Optimal Replacement #

↳ Generates min Page faults

Ref string :-

<7, 0, 1, 2, 0, 3, 0, 4, 2, 3, 0, 3, 2, 1, 2, 0, 1, 7, 0, 1>

- ① Replace that page which will not be used for the longest duration of time.

7	2	2	2	2	2	7
0	0	0	4	0	0	0
1	1	3	3	3	1	1

Page faults = 9

(3 frames)

Now for 4 frames :-

<7, 0, 1, 2, 0, 3, 0, 4, 2, 3, 0, 3, 2, 1, 2, 0, 1, 7, 0, 1>

<del>F</del>	3	<del>3</del>	1	1
0	0	0	0	0
1	<del>1</del>	4	4	4
2	2	2	<del>2</del>	7

Page faults = 8  
(4 frames)

N.B.: For second string using optimal  
3 frames  $\rightarrow$ ? (7)  
4 frames  $\rightarrow$ ? (6)  
does it suffer from Belady's Anomaly.

BUT HOW WILL YOU LOOK INTO FUTURE?



we use this  
algorithm as a  
benchmark.

which pages are you going  
to refer.



Like SJF, it's not implementable.

Let's shift this algo 180° :- # Least Recently Used



Replace that page which has not  $\leftarrow$  been used for  
the longest duration in the past.

Q Ref string :-

(7, 0, 1, 2, 0, 3, 0, 4, 2, 3, 0, 3, 2, 1, 2, 0, 1, 7, 0, 1)

7	2	2	4	4	4	0	1	1	1
0	0	0	0	0	3	3	3	0	0
1	1	3	3	2	2	2	2	2	7

Page Faults /<sub>3</sub> = 12

Page faults /<sub>4</sub> = 8

# Most Recently Used #

Replace the page which has just been used

Q Ref string :-

(7, 0, 1, 2, 0, 3, 0, 4, 2, 3, 0, 3, 2, 1, 2, 0, 1, 7, 0, 1)

7	7	7	7	7	7	7	7	7	7	7	7	7	0
0	0	3	0	4	4	4	4	4	4	4	4	4	4
1	2	2	2	2	3	0	3	2	1	2	0	1	1

Page Faults /<sub>3</sub> = 16

## # Counting Algo :-

Criteria :- Count of Ref.

when swapped out

&amp; then swapped in

set count = 1

a) LFU :- Least Frequently used

b) MFU :- Most Frequently used

&lt;7, 0, 1, 2, 0, 3, 0, 4, 2, 3, 0, 3, 2, 1, 2, 0, 1, 7, 0, 1&gt;

~~A<sup>L</sup>~~  
0<sup>L</sup>  
1<sup>L</sup>

→ same count  
use FIFO

2<sup>L</sup>  
0<sup>L+1</sup>  
~~A<sup>L</sup>~~

2<sup>L</sup>  
0<sup>L+1</sup>  
~~3<sup>L</sup>~~

4<sup>L</sup>  
0<sup>3</sup>  
~~3<sup>L</sup>~~

4<sup>L</sup>  
0<sup>3</sup>  
~~2<sup>L</sup>~~

3<sup>L</sup>  
0<sup>4</sup>  
~~2<sup>L</sup>~~

when swapped in count=1

3<sup>2</sup> 3<sup>2</sup> 3<sup>2</sup> 3<sup>2</sup> 3<sup>2</sup>  
0<sup>4</sup> 0<sup>5</sup> 0<sup>5</sup> 0<sup>6</sup> 0<sup>6</sup>  
~~A<sup>L</sup>~~ ~~1<sup>L</sup>~~ ~~1<sup>L</sup>~~ ~~7<sup>L</sup>~~ L<sup>1</sup>.

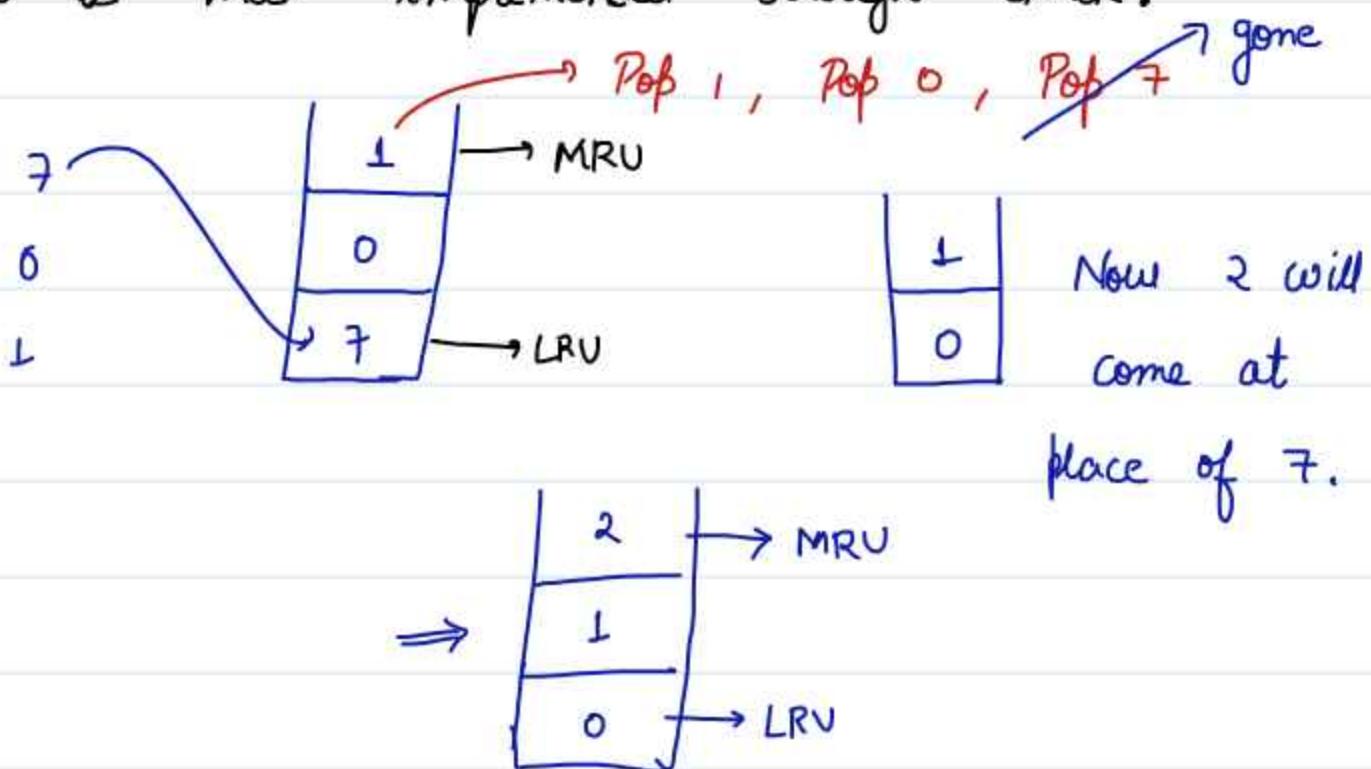
Page faults = 13

Similarly do for MFU :- Page faults = 15 (say)

NOTE :- Most of the time it's found that LRU is closer to optimal in terms of performance.

Many OS either implement LRU / variations of LRU.

LRU is most implemented through stack.



# LRU Approximations :- These algo are not really LRU but they approximate to the behaviour of LRU.

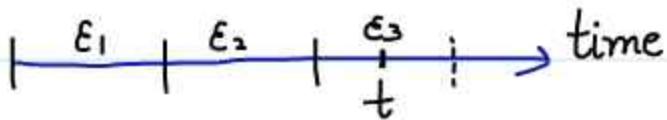
< Reference bit >

→ Each page in the PT will be associated with Reference bit R.



- a) Reference bit (talk about current epoch)
- 0 → Page is not referred so far
  - 1 → atleast once | during present epoch.

Epoch  $\Rightarrow$  Time Quantum (duration of time)



P	F	VLI	TOL	R	
0	a	1	2	1	
1	c	1	3	0	
2	d	1	0	1	
3	-	0	-	-	
4	c	1	4	1	
5	b	1	1	0	

→ Page fault

5 pages (0, 1, 2, 4, 5)

$P_2$ ,  $P_4$   
first      last

$P_0$ ,  $P_2$  &  $P_4$  has  
been referred in this  
epoch

LRU:- has not been referred to the longest  
duration in the past

→  $R=0$  matches with Stmt.

scan the page table right from the first  
entry & as  $R=0$  we get victimize it.



Page 1 will be victimized.

If no  $R=0$  then Ref bit algo fails.

when one epoch gets completed then  $\Rightarrow$  set  $R=0$ .

b) Additional Reference bits :- Each page is associated with more than one Ref bits (say 8).

	Previous history				current epoch			
	$R_1$	$R_2$	$R_3$	$R_4$	$R_5$	$R_6$	$R_7$	$R_8$
$P_i$	1	0	0	1	0	1	1	1
$P_j$	0	0	1	1	1	0	1	1
$P_k$	0	0	1	1	0	1	1	1
$P_l$	(X)	cause Page Fault				3	2	1

who will be victimized?

Rare case :- when all reference bits are 1.

This algo will also fail.

When current epoch gets completed  $\Rightarrow$  Shift Left of

$\downarrow$   
all  $R_8 \Rightarrow 0$

$R_1 \Rightarrow$  Lost

## # Second Chance (Clock Algorithm) #

Criteria :- Time of loading + Reference bit

P	f	V/I	T.o.L	R
0	e	1	2	1
1	a	1	3	0
2	b	1	0	X0
③	-	0	-	-
4	c	1	1	0
5	d	1	4	1

- The page which gets loaded first see its R value if 0 , victimize it. if R value is 1 , Set it to 0.  
→  $P_4$  will be victimized.

→ after one cycle go again in same order.

→  $P_2$  will be victimized.

P	f	V/I	T.o.L	R
0	e	1	2	X0
1	a	1	3	X0
2	b	1	0	X0
③	-	0	-	-
4	c	1	1	X0
5	d	1	4	X0

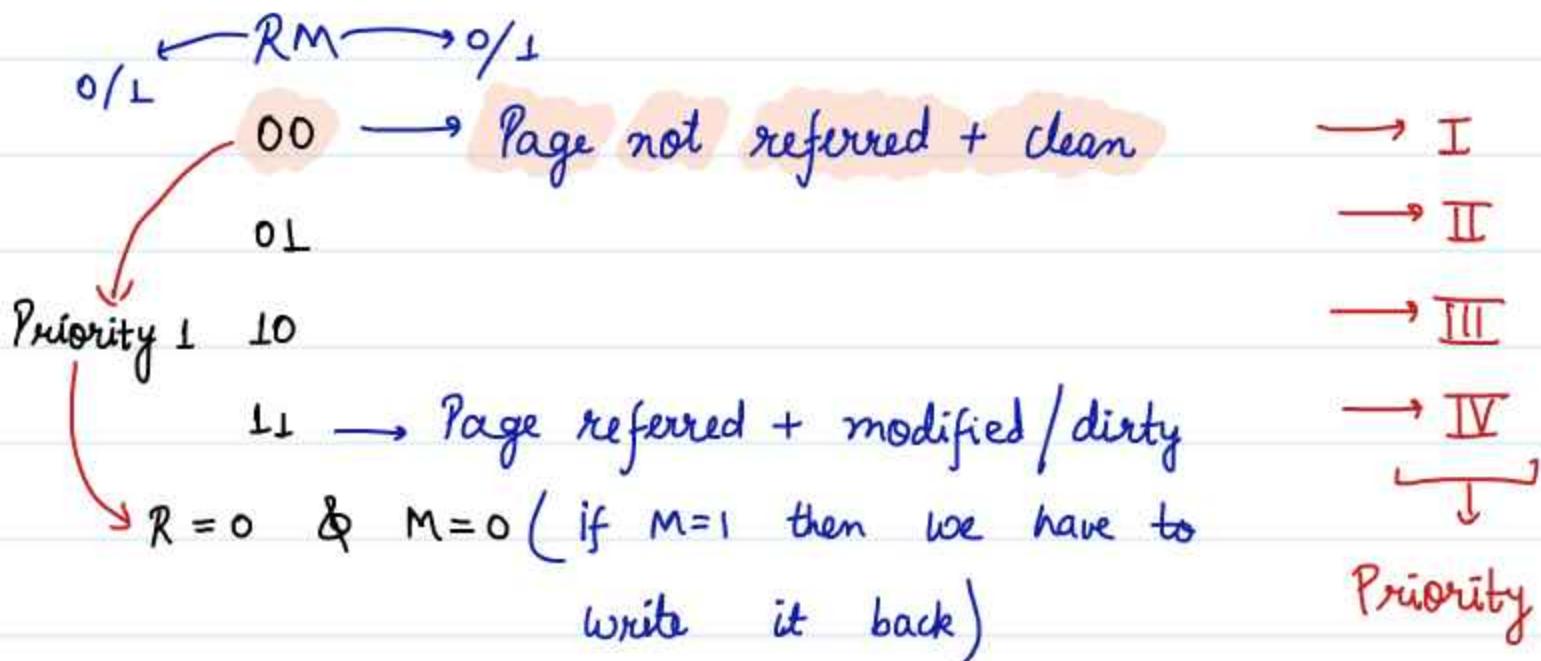
→ In this case reference bit would have been failed.

When all pages R value is 1 then FIFO page gets selected.

It'll also have the problem of bellady's anomaly.

MEMORYMANAGEMENT - Final (10)

- a) Reference bits
- b) Additional Reference bits
- c) Second chance / clock Algo ( TOL + R )
  - FIFO when all R values are 1's.
  - Belady's Anomaly
- d) Enhanced second chance ( NRU - Not Recently used)
  - Criteria = Reference bit + Modified bit (Dirty bit)



PTE Structure

P	f	V/I	T	L	R	M
0	c	1	4	1	1	
1	a	1	3	0	0	
2	-	0	-	-	-	
3	b	1	0	1	1	
4	d	1	2	1	0	
5	e	1	1	0	1	

- (i) FIFO : P<sub>3</sub>
- (ii) R : P<sub>1</sub>
- (iii) Sec chance : P<sub>5</sub>
- (iv) Enh Sec chance : P<sub>1</sub>

Q.

Consider a System with V.A.S = P.A.S =  $2^{16}$  Bytes. Page Size is 512 Bytes. The size of Page Table entry is 32 bits. If the Page Table Entry contains besides other information 1 V/I bit, 1 Reference, 1 Modified bit, 3 bits for Page Protection. How many bits can be assigned for storing other attributes of the Page. Also compute Page Table Size in Bytes?

$$V.A.S = P.A.S = 2^{16} B$$

$$\text{Page size} = 2^9 B$$

$$e = 4B$$

$$\text{No of entries} = \frac{2^{16}}{2^9} = 2^7$$

$$\text{Page Table size} = 2^2 \times 2^7 = 2^9 B = 512 B$$

$32 - 6 = 26$  bits

→ 7 bits for frame number  
 → 19 bits for rest other attributes

Q.

Consider a virtual memory system with FIFO page replacement policy, for an arbitrary page access pattern, increasing the number of page frames in main memory will

- A Always decrease the number of page faults.
- B Always increase the number of page faults
- C Sometimes increase the number of page faults
- D Never affect the number of page faults.

→ Belady's Anomaly

Q.

A memory page containing a heavily used variable that was initialized very early and is in constant use is removed when  
 ↗ brought first

- A. LRU page replacement algorithm is used
- B. FIFO page replacement algorithm is used
- C. LIFO page replacement algorithm is used
- D. None of the above

Q.

Recall that Belady's anomaly is that the page-fault rate may increase as the number of allocated frames increases. Now, consider the following statements:

S1: Random page replacement algorithm (where a page chosen at random is replaced) suffers from Belady's anomaly

S2: LRU page replacement algorithm suffers from Belady's anomaly

Let's say by coincidence it worked by FIFO

→ ?

→ No it doesn't

suffer.

Q Consider a process having ref string  $L$  in which  $n$  unique pages occur,  $z$  frames are allocated to process, calculate lower bound & upper bound of no of page faults.

Ans

Max  $\Rightarrow l$  [if  $z=1$ ] } every reference will cause page fault.

Min  $\Rightarrow n$  [if  $z \geq n$ ]  $\rightarrow$  [Pure DP]

$\Rightarrow 0$  [Prefetch DP]

Like deadlock it's undesirable feature of OS.

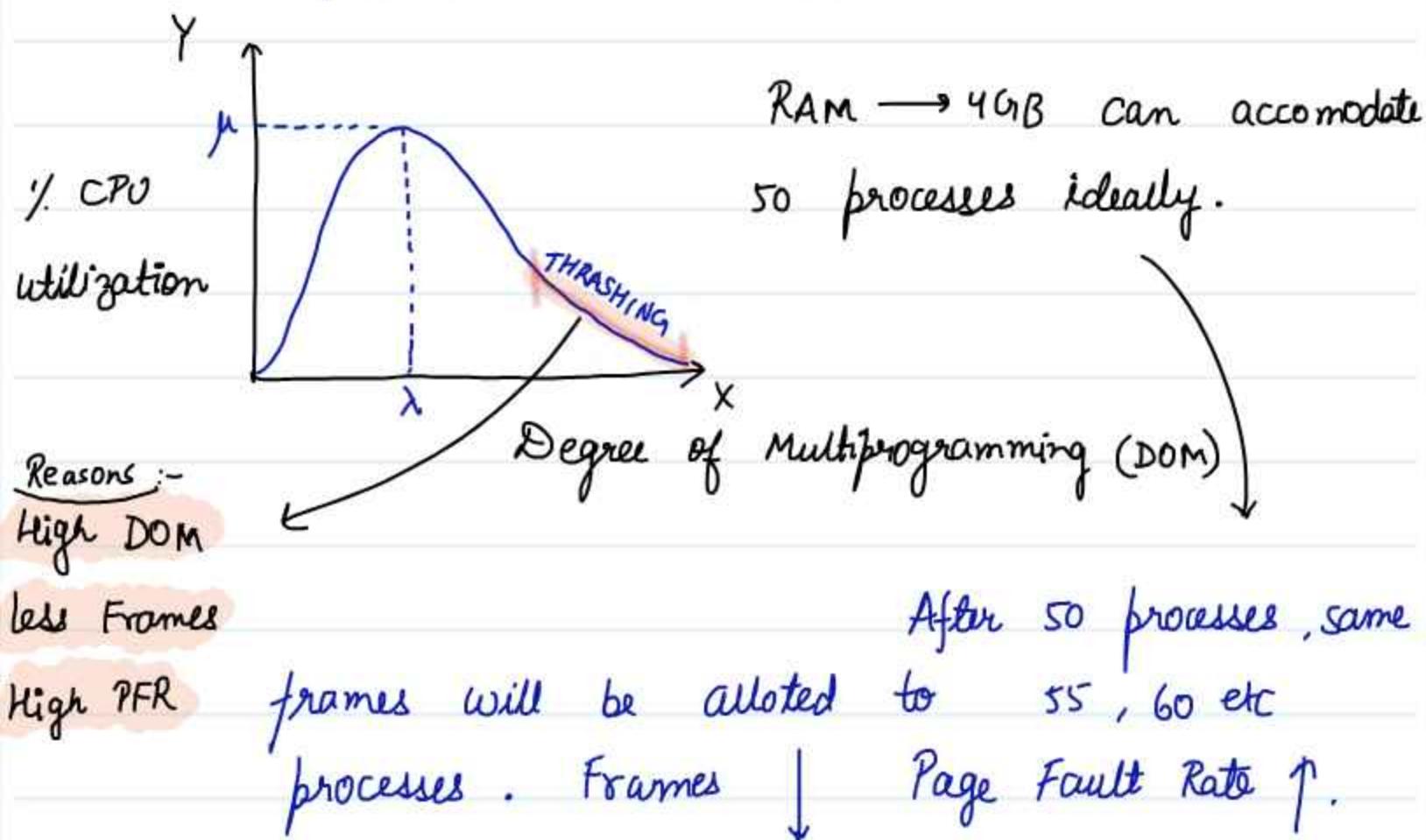
4

## # Thrashing #

↳ Excessive / High paging activity [High PFR]

The act of causing page fault leading to loading & saving pages on the disk.

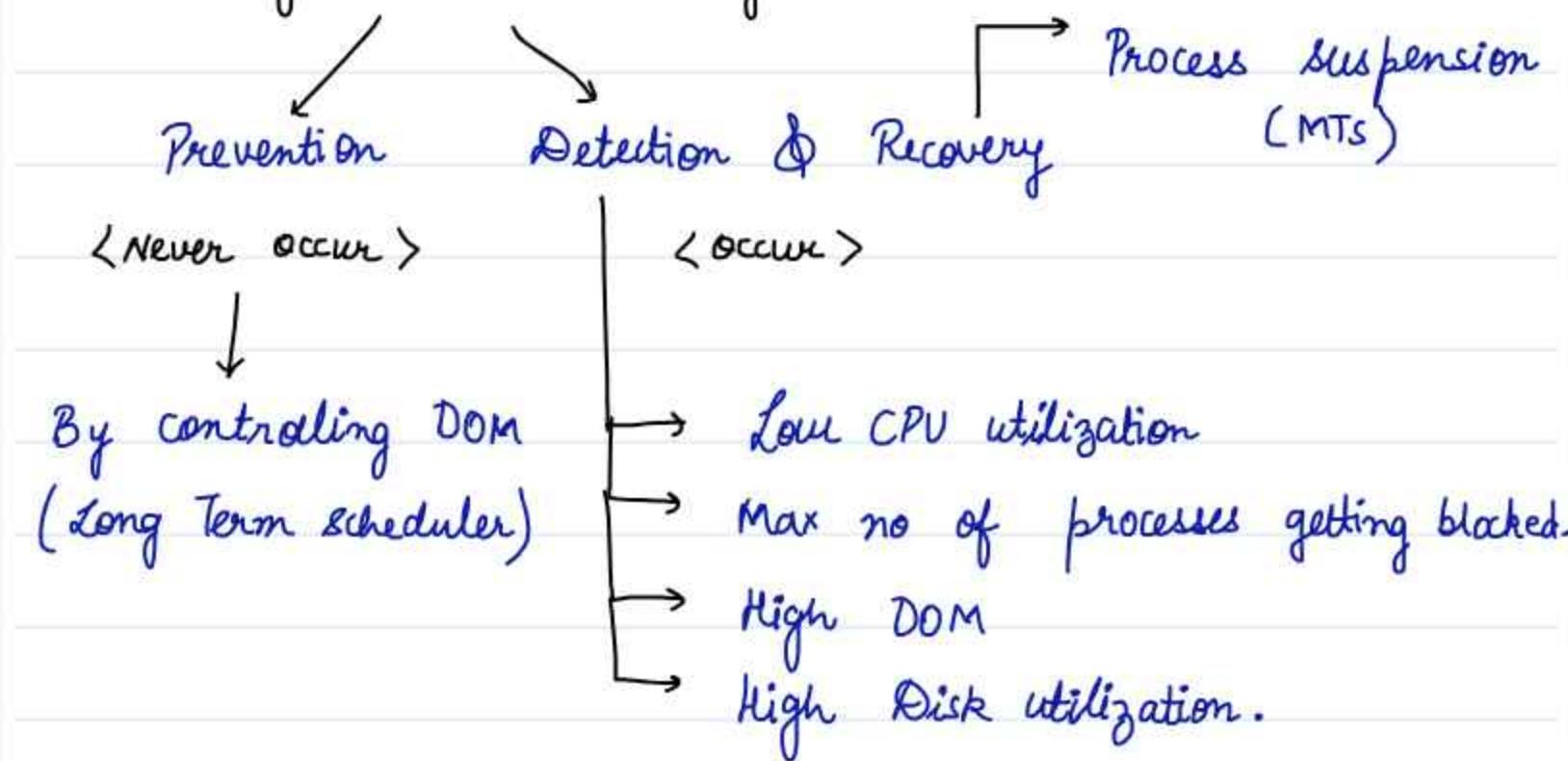
Most of the time process will be spending time on their page fault service (gets blocked)



- If OS is not using good replacement technique PFR will increase.

- If Page size  $\Rightarrow$  small  $\Rightarrow$  More Pages  $\Rightarrow$  High PF.  
 $\Rightarrow$  Large  $\Rightarrow$  less Pages  $\Rightarrow$  less PF.

- Thrashing control strategies.



- ~~•~~ How can our (we as programmers not os dev) programming / data structure can affect thrashing?

Case-I

int A[1...128][1...128];

1. for i  $\leftarrow$  1 to 128  
 for j  $\leftarrow$  1 to 128  
 $A[j,i] = 1;$

$P_1 \leftarrow a_{11} \ a_{12} \dots a_{1128}$   
 $P_2 \checkmark a_{21} \ a_{22} \dots a_{2128}$

2. for i  $\leftarrow$  1 to 128  
 for j  $\leftarrow$  1 to 128  
 $A[i,j] = 1;$

Page size = 128 words

& we are following Row major order

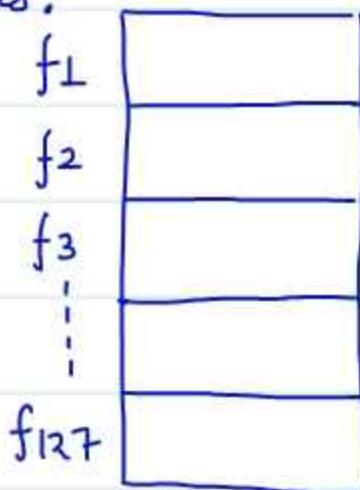
128 pages  $\leftarrow$  no of pages needed to store matrix on disk.

$A[1, 1]$   
to  
 $A[1, 128]$

$A[2, 1]$   
to  
 $A[2, 128]$

$A[128, 1]$   
to  
 $A[128, 128]$

Based on these 128 pages, it'll ask for 128 frames.



Pure DP ✓

FIFO ✓

$\rightarrow$  127 frames allocated.

How many total page faults will occur with ① & ②?

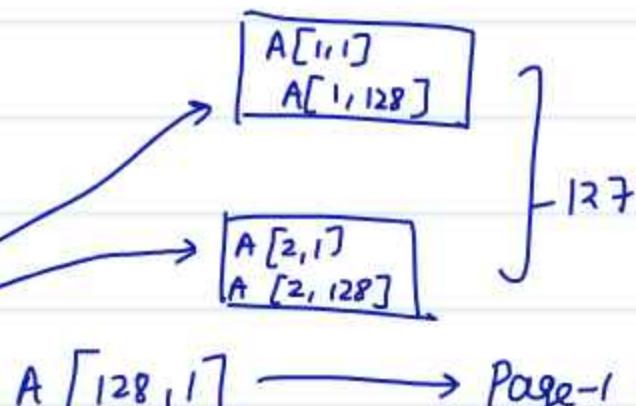
for  $i \leftarrow 1$  to 128

for  $j \leftarrow 1$  to 128

$A[j, i] = 1;$

$A[1, 1]$

$A[2, 1]$



$A[128, 1] \longrightarrow$  Page-1 will be Replaced

$f_1$	$P_1 P_{128}$
$f_2$	$P_2 \text{ } (P_1)$
$f_3$	$P_3 P_2$
$\vdots$	$\vdots$
$f_{127}$	$P_{127} P_{126}$

$A[1,2]$  was present in first page but got replaced that's why  $P_2$  was replaced.

Each value of  $i \Rightarrow 128$  page faults

Total  $\Rightarrow (128)^2$  Page faults.

②

for  $i \leftarrow 1$  to  $128$

for  $j \leftarrow 1$  to  $128$   
 $A[i,j] = 1;$

$A[1,1]$
$A[1,2]$ ...
$A[1,128]$

one page fault

Each value of  $i \Rightarrow 1$  PF

Total PF  $\Rightarrow \underline{128}$

We stored in RMO but  $A[j,i]$  is CMO  $\rightarrow$  Increased PF

" " " &  $A[i,j]$  is also RMO  $\Rightarrow$  Low PF

Program 1 is 128 times slower than Program 2.

Program 2 follows LOCALITY OF REFERENCE

(i) P.S. = 256 w

(ii) P.S. = 64 w

Case-2Arrays

Vs

LinkedList

} which is better in

case of demand  
page environment?

much possibility  
that complete array  
in single page

each node  
will be in different  
page

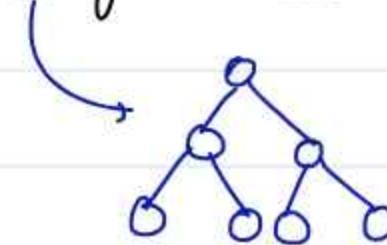
↓  
Lesser PF↓  
Higher PF

} may

Case-3Linear searchBinary search

↓  
Like an array

May f  
↓  
Less PF

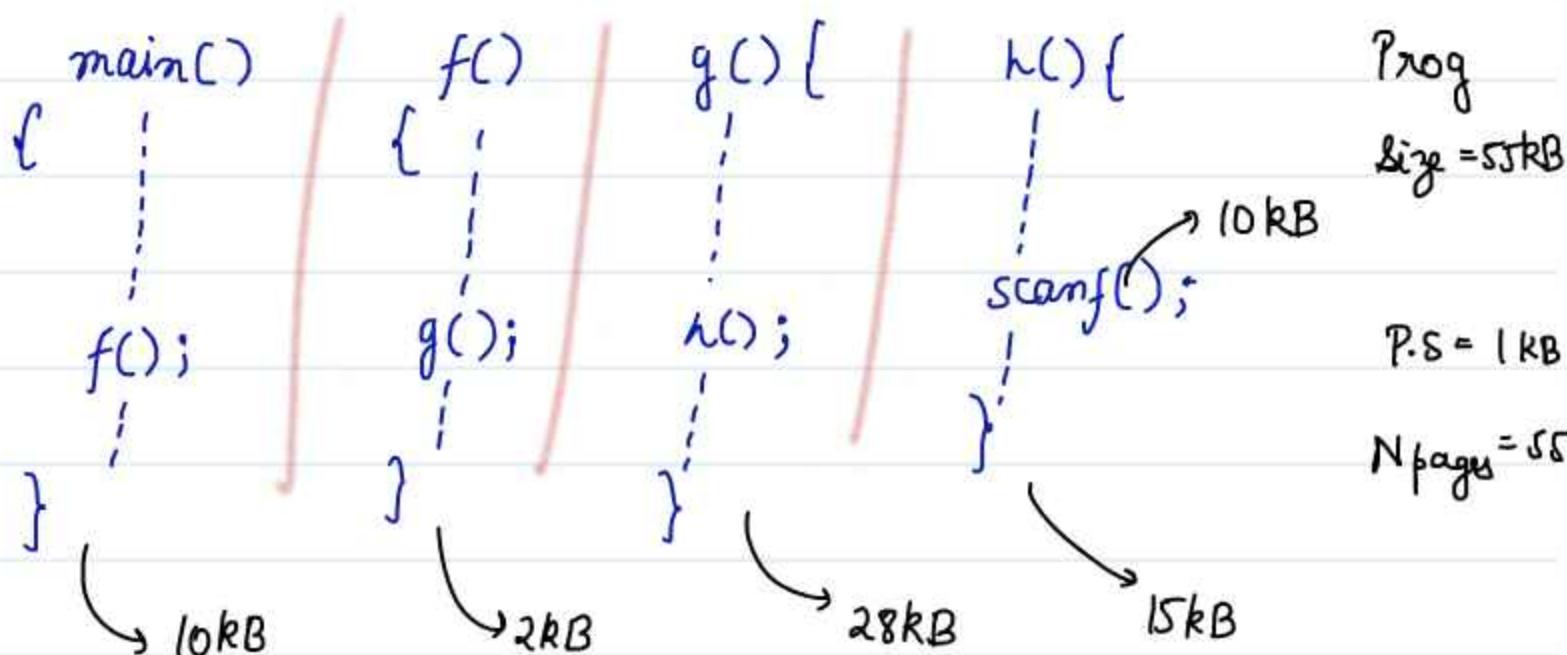


Higher (each node will  
be in diff page)

A programming tech or D.S. is said to be good  
in demand page env. iff it satisfies Locality model.  
should refer those pages only which are loaded in memory.

## # Working Set Strategy (Model) :-

- To minimize page fault rate & also utilize memory effectively.
- Works on principle of locality of reference.



Instead of static demand of 55 frames →  
see which locality & demand accordingly

dynamic mem allocation

When in main 10KB → 10 pages req,

when f() → 2 pages, doesn't need that 10 old pages of main.

whatever be the size of locality ask for those many frames only ⇒ Less PFR & Better mem. utilization.

# Working set strategy

- control page fault.
- Utilize memory effectively.
- uses Locality of Reference

Dynamic Frame allocation

Estimate the size of locality in which the program is executing & demand those many frames.

$$\langle P_{R_i} \rangle = \langle 4, 5, 8, 2, 4, 5, 8, 20, 22, 21, 19, 20, 18, 19 \\ 54, 58, 56, 54, 53, 51, 52, 58, 56, \dots \rangle$$

at time t

Working set window (WSW) = {set of unique pages referred in the Ref. string during the past  $\Delta$  references}

WSW → Sliding window

$\Delta$  integer

$$WSW_0^t = \langle 19, 51, 52, 53, 54, 56, 58 \rangle$$

$\Delta = 10$

unique ones

WSW size = 7 ] → Right out we are in locality which is defined in terms of 7 pages

ask OS to give 7 frames

Is the system thrashing or not by this framework?

No of processes = 'n'  $\langle P_1, \dots, P_n \rangle$

Total available frames = M

Demand of each process for frames =  $s_i = \sum_{\Delta=10}^t w_s w_s^t$

Total demand @ time t =  $\sum_{i=1}^n s_i = D$

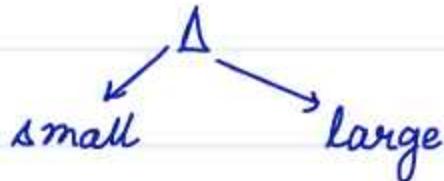
I:  $D \approx M$  No Thrashing

II:  $D < M$  No Thrashing, you can still ↑ DOM.

III:  $D > M$  System is Thrashing.

The whole success of WS strategy depends on  $\Delta$ .

Let's say the ideal value is  $\Delta = 10$



$\Delta = 2$   
leads to More PF

$\Delta = 10$

Ineffective mem utilization

Q-

Let the Page Reference and the Delta ( $\Delta$ ) be "c c d b c e c e a d" and 4, respectively. The initial Working Set at time  $t = 0$  contains the pages {a, d, e}, where 'a' was referenced at time  $t = 0$ , 'd' was referenced at time  $t = -1$ , and 'e' was referenced at time  $t = -2$ . Determine the total number of page faults and the average number of page frames used by computing the working set at each reference.

Ref string =  $\langle c, c, d, b, c, e, c, e, a, d \rangle$ ;  $\Delta = 4$

Initial WS =  $\{e, d, a\} \rightarrow$  at  $t=0$   
 $t=-2 \quad t=-1 \quad t=0$

Page faults = 5  
 Page frames used

$t =$   $\nearrow^{-2} \nearrow^{-1} \nearrow^0 \nearrow^1 \nearrow^2 \nearrow^3 \nearrow^4 \nearrow^5 \nearrow^6 \nearrow^7 \nearrow^8 \nearrow^9 \nearrow^{10}$   
 $\langle e, d, a, c, c, d, b, c, e, c, e, a, d \rangle$

unique

$$WSW|_{time=1} = \{e, d, a, c\}_{(4)}$$

$$WSW|_{t=5} = \{d, b, c\}_{(3)}$$

$$WSW|_{time=2} = \{d, a, c\}_{(3)}$$

$$WSW|_{t=6} = \{d, b, c, e\}_{(4)} \\ t=7 \rightarrow (3)$$

$$WSW|_{time=3} = \{a, c, d\}_{(3)}$$

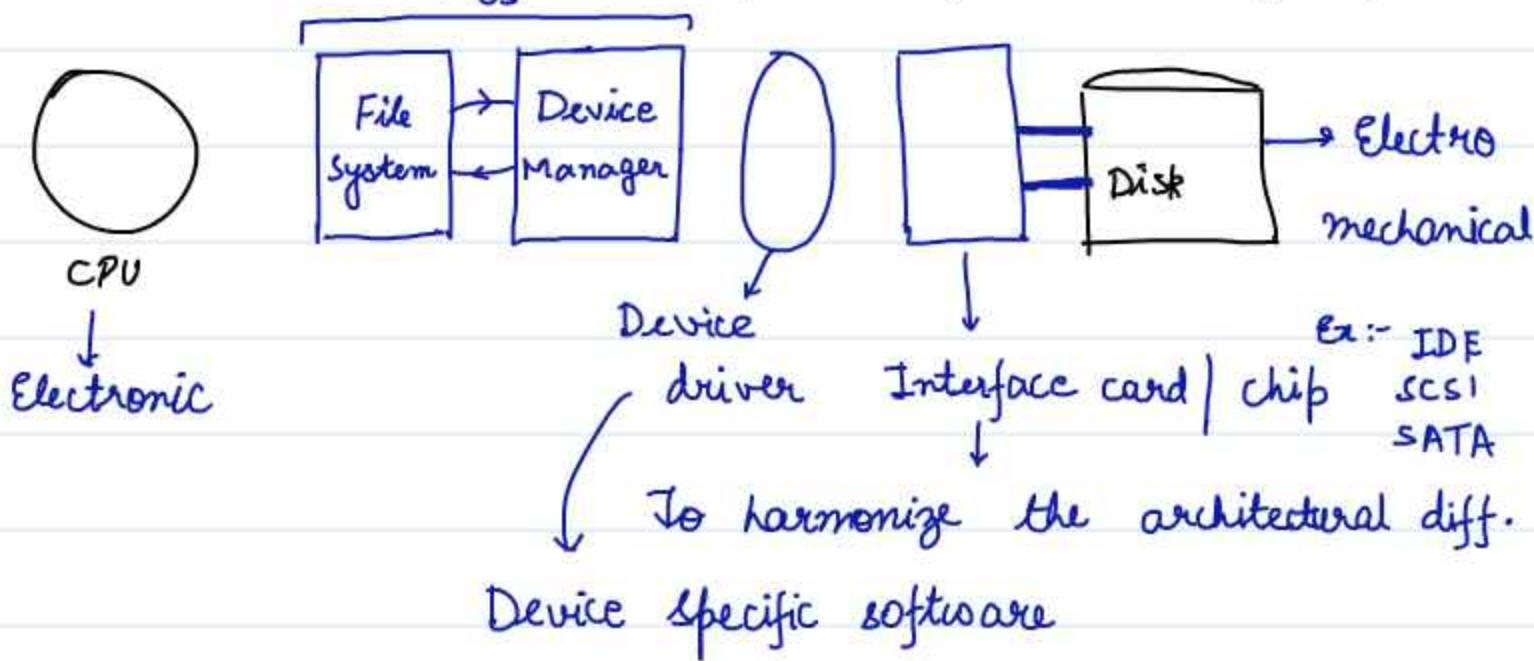
$$WSW|_{t=8} = \{c, e\}_{(2)} \\ t=9 \rightarrow (3)$$

$$WSW|_{t=4} = \{c, d, b\}_{(3)}$$

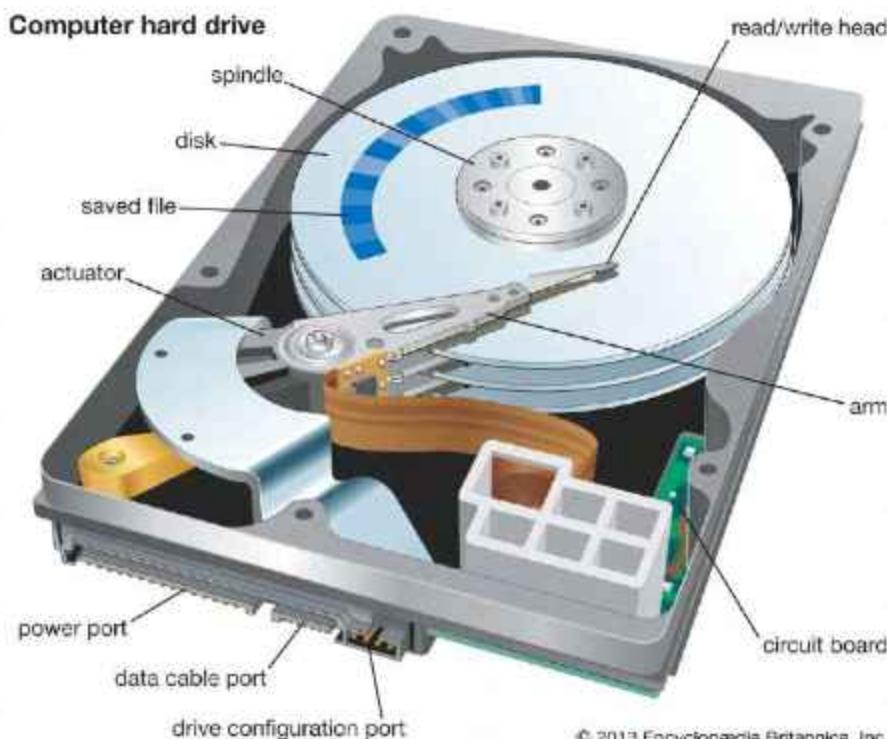
$$WSW|_{t=10} = \{c, e, a, d\}_{(4)}$$

# File System - Part - 1

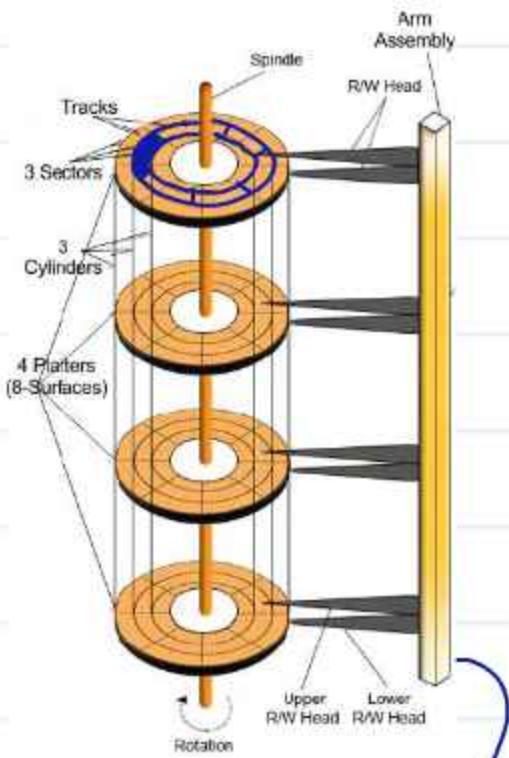
The only visible part of Operating system.



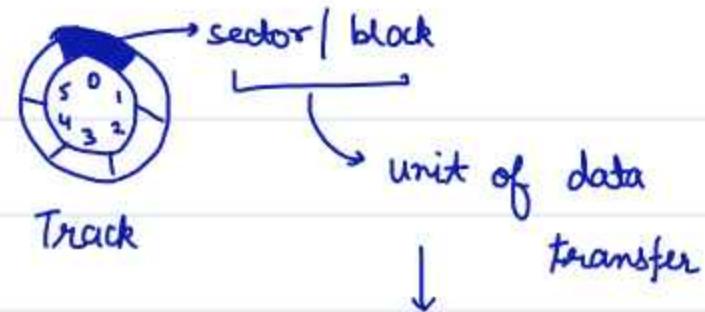
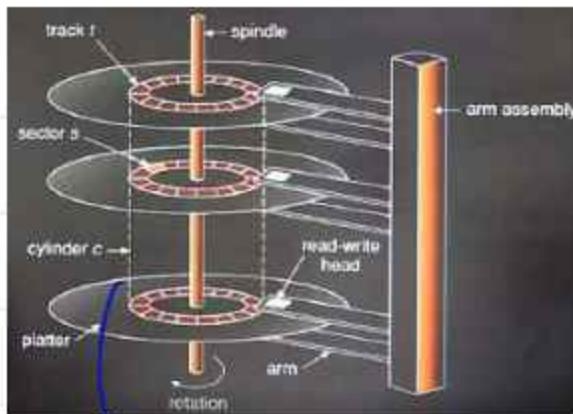
## # Disk Physical structure #



© 2013 Encyclopaedia Britannica, Inc.



can move forward/backward

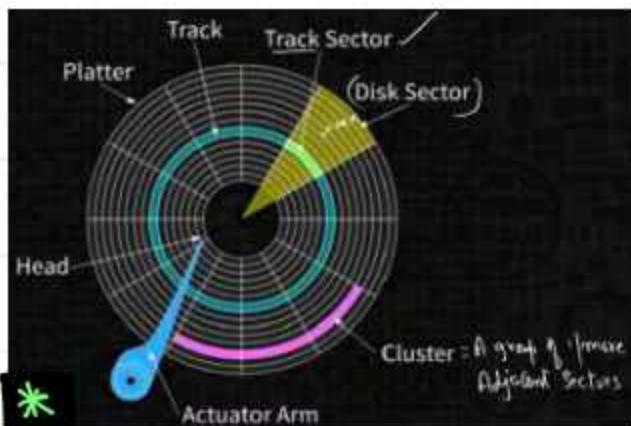


generally measured in bytes.

If we view the same track on platter, we get a cylinder.  
 ↓

$$\text{No of Tracks} = \text{No of cylinder}$$

same sector on all the tracks  $\Rightarrow$  Disk sector



Disk  $\rightarrow$  Platters  $\rightarrow$  2 surfaces

↳ Tracks

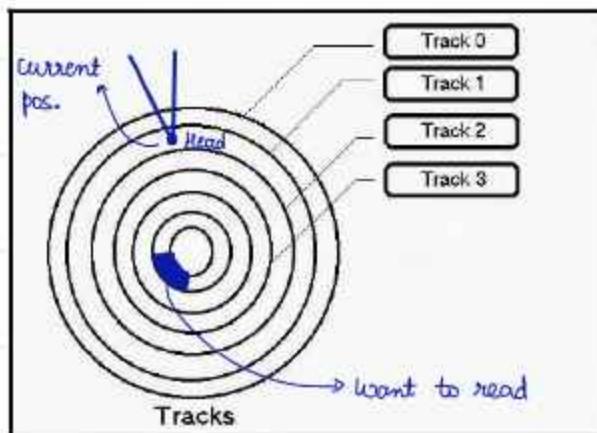
↳ Sectors

sector number (Bits)      sector size (Bytes)

Disk I/O Time  $\Rightarrow$   
(one sector)

Track 1 to Track 5

↓  
Seek time



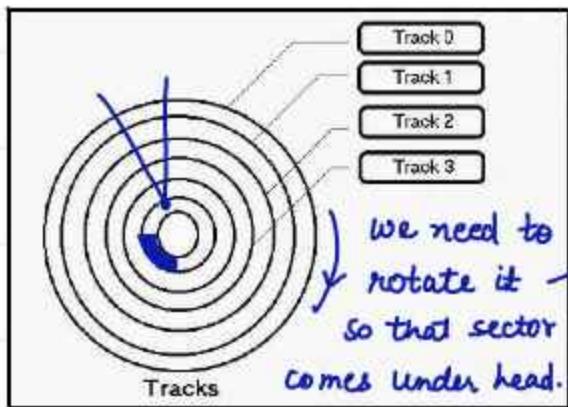
one track to another  
 ↓  
1 seek.

Head have to make 4 seeks.

Track - Track Time  $\Rightarrow$  Time taken by RW head to move from one track to another.

Seek Time  $\Rightarrow$  Total seeks  $\times$  TTT

$$| \text{Track to reach} - \text{current track no} | \times \text{TTT}$$



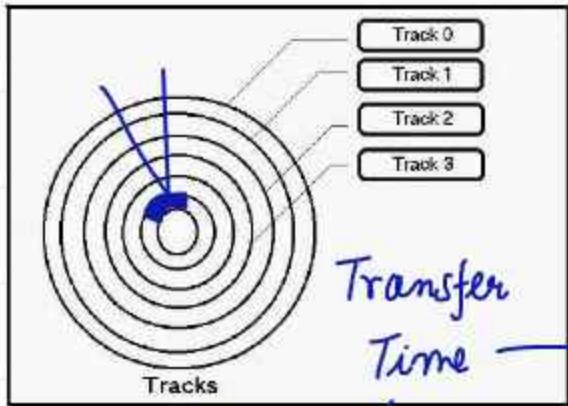
Time Taken for this  $\Rightarrow$  Rotational Latency.

$$\text{Generally taken as } \frac{R}{2}$$

Time taken for full rot.

$$( \text{rpm} )$$

$$\frac{\text{rotation}}{\text{minute}}$$



Amount of time

taken by RW head & transfer the data to disk buffer.

$$\text{Sector size } \left( \frac{S \cdot R}{Z} \right)$$

Track size

$$\text{Disk I/O Time} = \text{Seek Time} + \text{Latency} + \text{Transfer Time}$$

$$\begin{matrix} \text{Rotational} \\ \text{Time} \end{matrix}$$

Disk transfer data from rate :- Data Transfer Rate

$$\text{Bytes} \leftarrow \left( \frac{Z}{R} \right) \text{ kB/sec}$$

ms

**Q. 1**

Consider the following Disk Specifications:

Number of Platters = 16  
 Number of Tracks/Surface = 512  
 Number of Sectors/Track = 2048  
 Sector offset = 12 bits  
 Average Seek Time = 30 ms  
 Disk RPM = 3600

Calculate the Following:

- A. Unformatted Capacity of Disk.
- B. IO-Time/Sector
- C. Data Transfer Rate
- D. Sector Address

unformatted capacity

$$\frac{1 \times 16 \times 512 \times 2 \times 2048 \times 2^{12} \text{ B}}{\text{No of tracks} \times \text{sector size}}$$

↓

No of sectors

$$128 \text{ GB} \leftarrow 2^{37} \text{ B} \leftarrow 1 \times 2^4 \times 2^9 \times 2 \times 2^{10} \times 2 \times 2^{12}$$

$$\frac{\text{IO Time}}{\text{sector}} \Rightarrow \text{seek time} + \text{Rotational latency time} + \text{Transfer time}$$

↓

30 ms

$R/2 \Rightarrow 8.3 \text{ ms}$

K

$$K = \frac{\text{Track size}}{\text{sector size}} \times R$$

$$3600 \text{ rotation} \rightarrow 60 \text{ sec}$$

1 rot → ?

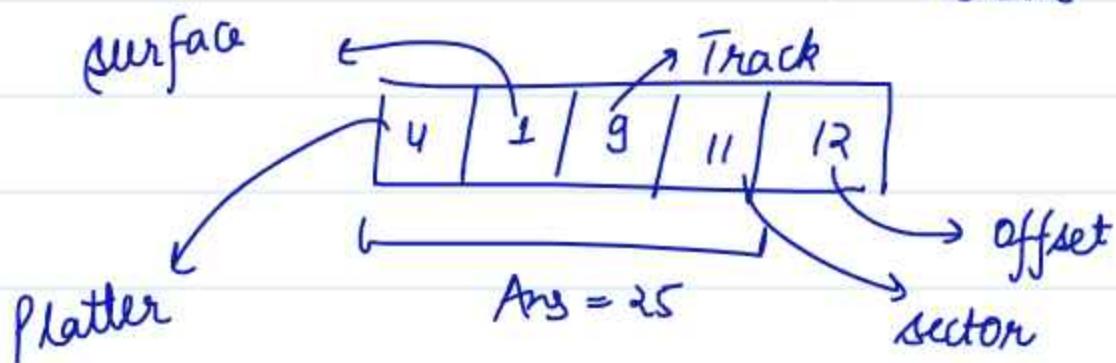
$$R = \frac{60}{3600} \times \frac{1}{2} = 16.67 \text{ ms}$$

$$\text{Data transfer rate} = \left(\frac{\pi}{R}\right) \text{ kB/sec} \Rightarrow 0.5 \text{ GB/sec}$$

$$(8 \text{ MB}) \quad 1 \text{ Track} \longrightarrow 1 \text{ Rotation} \quad (16.6 \times 10^{-3} \text{ s})$$

? ← 18

Sector Address in bits  $\rightarrow \log_2 (\text{No of sectors})$   
 $\Rightarrow 25 \text{ bits}$



Q

Consider a Disk with the following Specifications:

Number of surfaces = 64

Outer diameter = 16 cm

Inner diameter = 4 cm

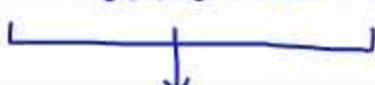
Inter Track space = 0.1 mm

Max Density = 8000 bits/cm

Calculate the Unformatted Capacity of Disk.

Inner most Track

< outer most Track

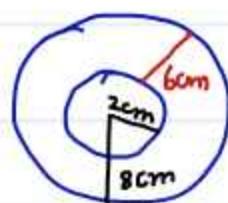


But then also size

is same How?



Surfaces = 64



$$\frac{\text{No of tracks}}{\text{surface}} = \frac{60 \text{ mm}}{0.1 \text{ mm}} = 600$$

By using varying density.

Track size ?

Track length in cm × Density

But we had max density is given



$$\text{Inner Track length in cm} = 2\pi r \xrightarrow{r=2\text{cm}} \\ = 4\pi \text{ cm}$$

$$\text{Track cap.} = 4\pi \times 1\text{KB} = 4\pi \text{ KB}$$

$$\text{Surface cap.} = 600 \times \underbrace{4\pi \text{ KB}}_{12.56 \text{ KB}}$$

$$\text{Disk capacity} = \text{surface cap.} \times 64 \text{ surfaces} \\ = \underline{\underline{0.48 \text{ GB}}}$$

~~RW-2~~

How long does it take to load a 64 Kbytes Program from a disk whose Average Seek time is 30 ms Rotation time is 20 ms, Track Size is 32 Kbytes, Page Size is 4 Kbytes. Assume that Pages of the Program are distributed randomly around the disk. What will be the % saving in time if 50% of the Pages of program are Contiguous?

64 KB program  $\longrightarrow$  Load  
 Seek time = 30 ms

} Randomly distributed

Rotation time = 20 ms

Track size = 32 KB

No of pages =  $\frac{64 \text{ KB}}{4 \text{ KB}}$

PS = 4 KB

$\Rightarrow$  16 pages

1 Page Load Time  $\Rightarrow$  30 ms + 10 ms +  $\frac{4 \text{ KB}}{32 \text{ KB}} \times 20 \text{ ms}$

1 Rot  $\longrightarrow$  1 Track  
 ?  $\longleftarrow$  1 Page ]

$$= 30 + 10 + 2.5 \text{ ms} = 42.5 \text{ ms}$$

$$16 \text{ page Load Time} = 42.5 \times 16 = 680 \text{ ms}$$

Case-2 :- 50% pages of the program are contiguous

8 pages are contiguous  
 8 pages are randomly distributed

$$42.5 \times 8 = 340$$

All pages are at same track.

$$30 + 10 + 20 = 60$$

$$\text{seek Time} + \frac{R}{2} + 2.5 \times 8$$



one time only

Total :-

$$340$$

$$60$$

$$\underline{400}$$

$$\text{Case -1} \quad \text{Time} = 680$$

$$\text{Case -2} \quad \text{Time} = 400$$

$$(400 - 680)$$

$$\% \text{ Saving} = \frac{280}{680} \times 100 \approx 41\%$$

Rotational latency is taken into account only once when the pages are contiguously distributed over one track because the read/write head of the disk needs to wait for the disk to rotate to the correct position only at the beginning of the read operation.

Once the first page on the track is being read, the rest of the pages on the same track can be read in one continuous sweep as the disk rotates. This is because the pages are laid out contiguously on the disk, so there's no need for the disk to rotate multiple times to different positions to read each page.

Therefore, the time taken for the disk to rotate to the correct position (rotational latency) is only incurred once, at the beginning of the read operation. After that, the read/write head can continue reading the rest of the pages on the track without additional rotational delay. This is why rotational latency is taken only once when the pages are contiguously distributed over one track.

In case of  
contiguous distribution



Q

An Application requires 100 libraries at startup. Each library requires 1 disk access. Seek Time is 10 ms, Disk RPM is 6000. All 100 libraries are at random locations. 50% of Libraries require transfer time of  $\frac{1}{2}$  Rotation, while for the remaining 50% it is negligible. How long does it take to load all 100 libraries?

Random [ 100 libraries ]  $\rightarrow$  100 disk access

$$\text{seek time} = 10 \text{ ms}$$

$$\text{Disk RPM} = 6000$$

$$R = 10 \text{ ms}$$

$$60 \text{ Sec} \longrightarrow 6000 \text{ Rotation}$$

$$\frac{60 \times 1000}{6000} \leftarrow 1 \text{ Rotation ms}$$

$$50 \text{ libraries} \Rightarrow TT = \frac{R}{2} = 5 \text{ ms}$$

$$50 \text{ libraries} \Rightarrow TT \approx 0$$

]  
set-1  
]  
set-2

$$\text{Time}_{\text{set-1}} = 50 (10\text{ms} + 5\text{ms} + 5\text{ms}) = 1000\text{ms}$$

9

$$\text{Time}_{\text{set-2}} = 50 (10\text{ms} + 5\text{ms} + 0) = 750\text{ms}$$

$$\text{Total Time} = \underline{1.75\text{ sec}}$$

## # Logical Structure of Disk [Formatting Processes]

Analogical to college library

↓  
Placing infrastructure onto disk.

library place ~ Disk

Books ~ Data

• Library without book shelves

~ Raw Disk

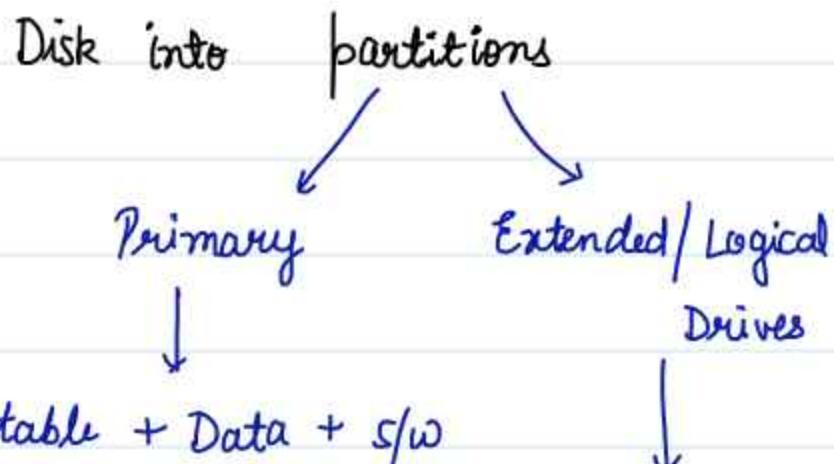
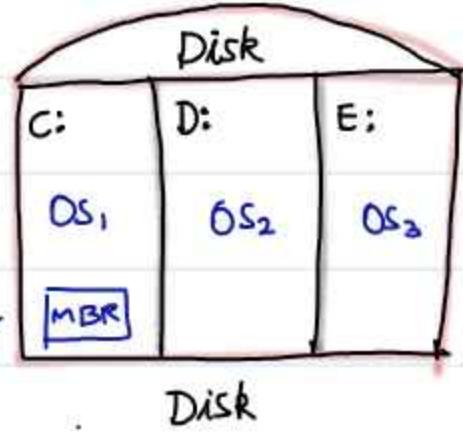
• Library with furniture

~ Formatted Disk

If I just dump thousands of books in library without having proper book shelves etc then searching becomes horrible.

- Effective storage
- Faster retrieval

Multiboot computer ] Generally done at comp Labs

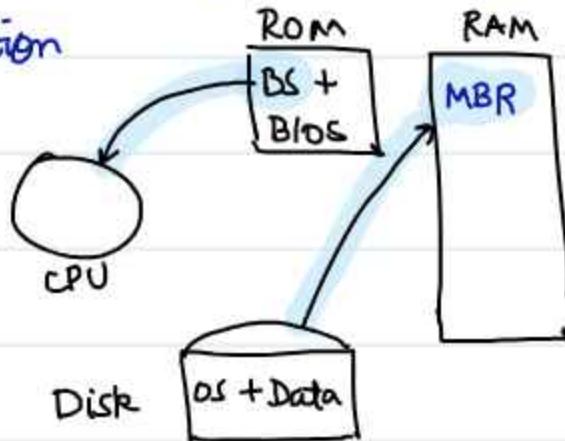


MBR :- on the first primary partition  
(Master Boot Record)

Partition Table

↓

Boot Loader



Information about all the partition

All are the devices electronically active ?  
(Hardware test)

Steps :- 1. Switch on

2. POST < Power on self Test >

3. BIOS < Basic Input - output system (ROM)

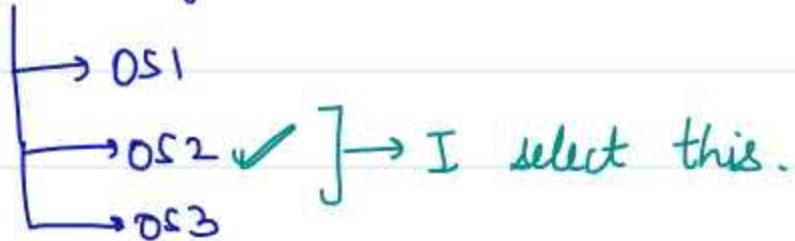
Initializes I/O devices including DISK

small booting program

4. Bootstrap .

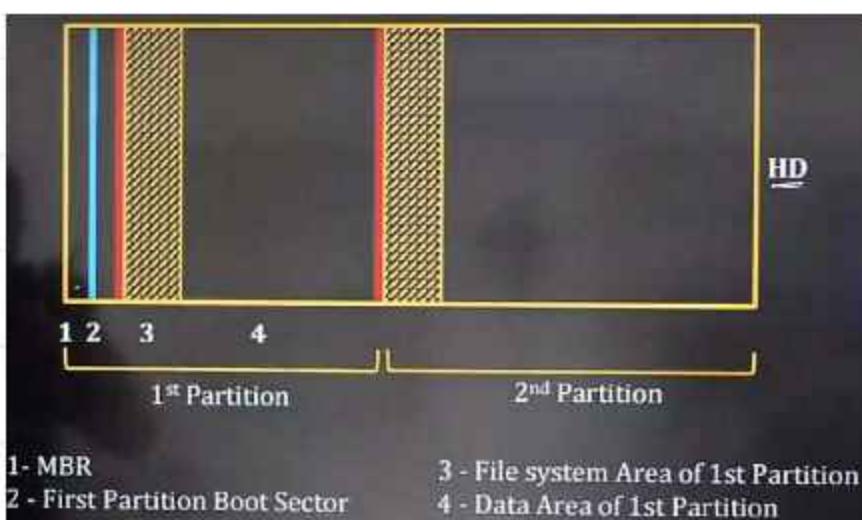
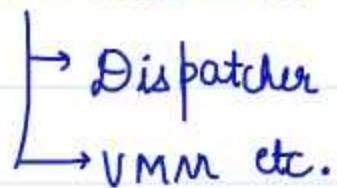
→ Loads MBR into RAM & handover the control to boot loader.

5. Boot Loader reads partition table & display options if available.



6. Boot loader will load the kernel program from disk into memory.

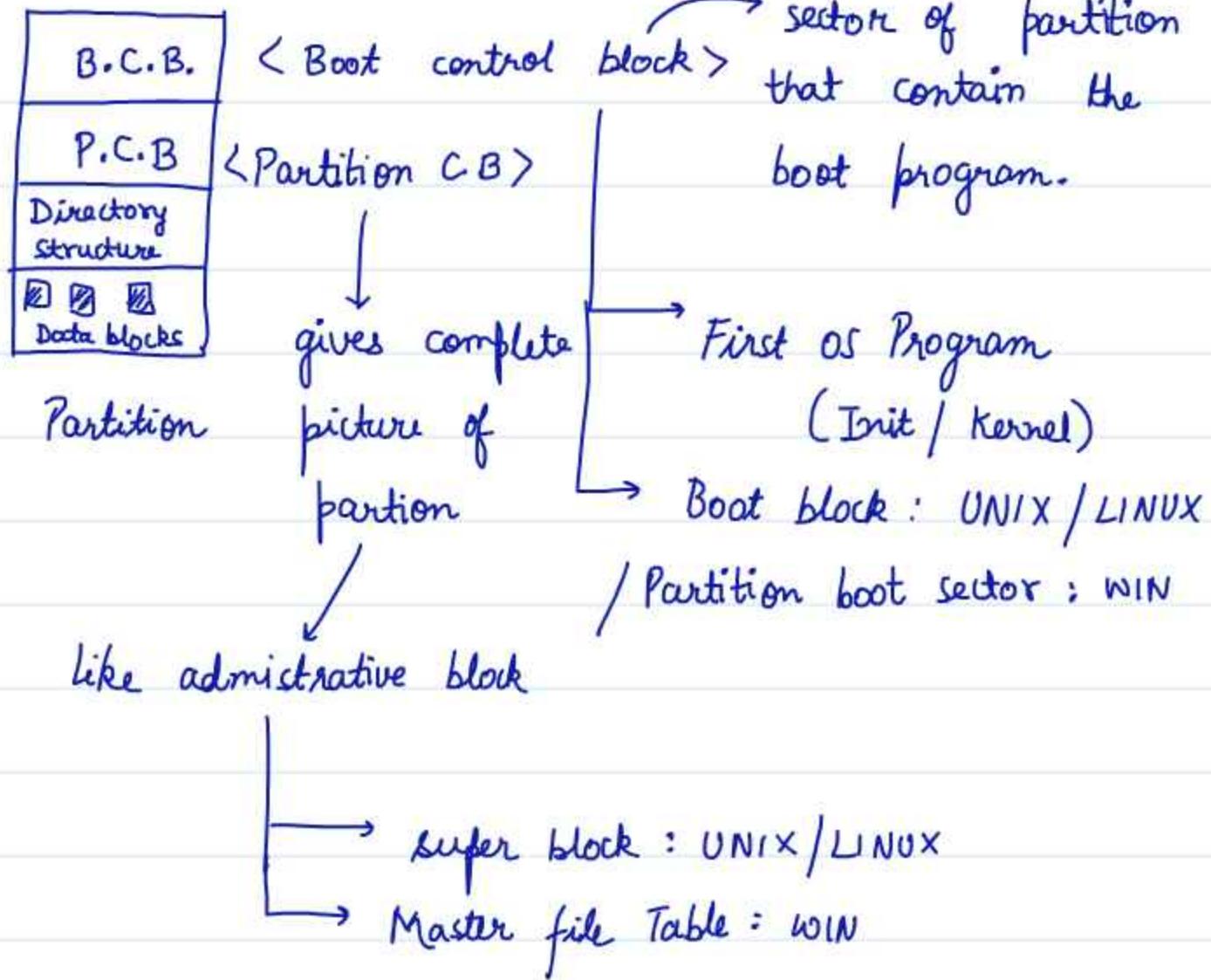
execution of kernel will load other OS modules into memory.



Partition structure



what infrastructure're embossed on the disk  
partition after formating



Data blocks contains data for application & other o.s. files.

# File Vs Directory #

- collection of logically related records / entity
- abstract Data Type (def., Rep/str., ops, attributes)

Ref/structure

→ Flat (series of bytes)

→ Record (series of records)

↳ Hierarchical Tree (B+, B-Trees)  
(higher level implementation)

Operation (create, open, read, write, seek, truncate etc)

Attributes (Name, extension, type, owner, mode, size, date & time, permissions, block in use)

↳ In which block

the file data is stored  
(data blocks in partition str)

# Attributes of file are stored in File control block.

Every partition contains  
directory structure.

UNIX / LINUX

WINDOWS

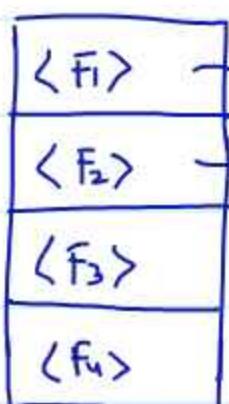
(I-Node)

(Directory Entry)

# Directory #

↳ special file which also contains data about other files.

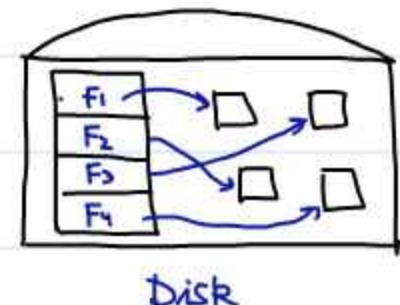
→ contains metadata of files (not actual data of files)



→ Directory entry

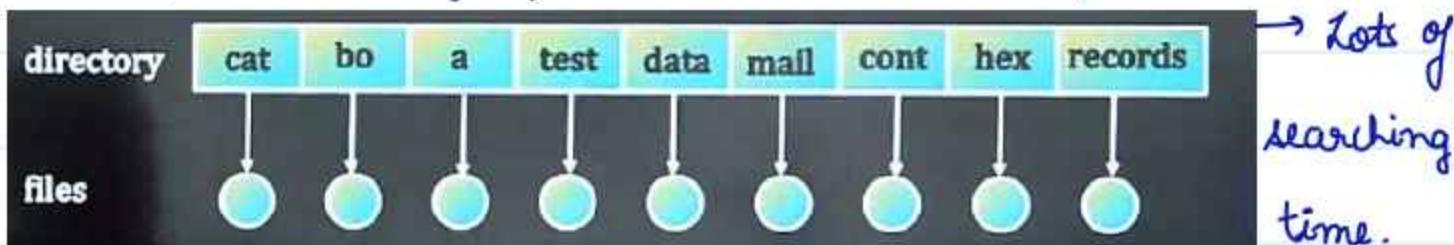
linear 1D array

Directory



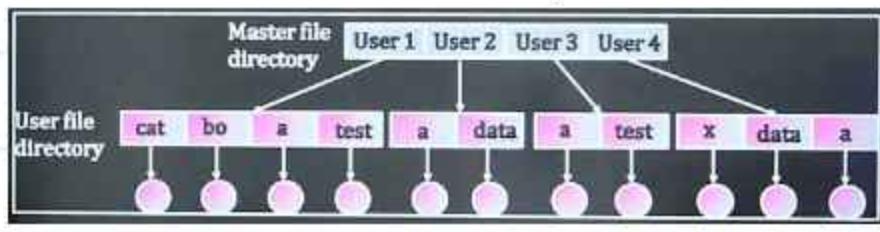
## # Directory structures #

a) Single directory for all users :- simple to Implement



→ Name conflict problem  
→ Bad organization

b) Two Level Directory :- separate directory for each user.

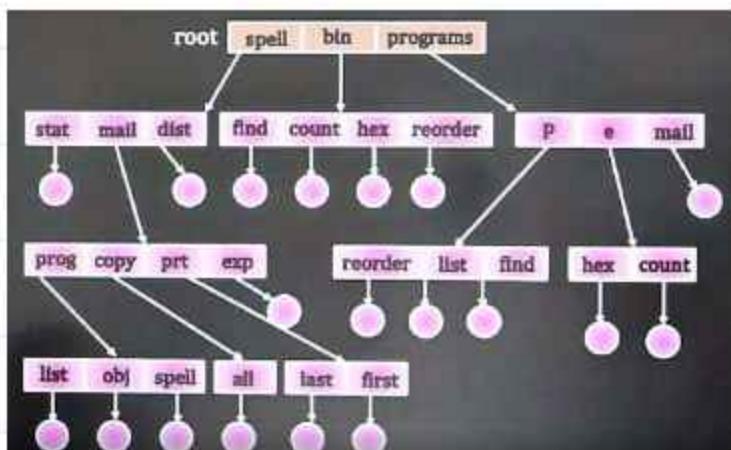


- Efficient searching
- path involved

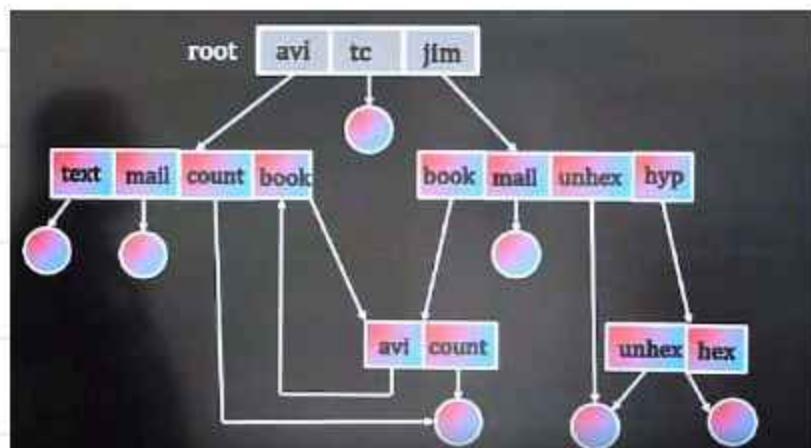
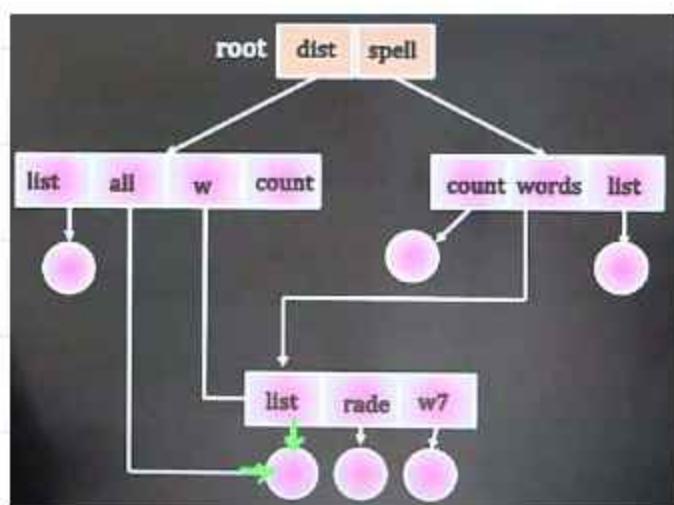
- Can have same file name for different user.
- No grouping capability.

wanted to keep some file separate.

### c) Multi Level Directory (subdirectories within other directories)



Tree structure



General Graph Directory  
(can have cycles)

Directed Acyclic Graph  
(For file sharing)

Link count  
(How many links are pointing to shared file)

To search a file we may have to traverse the directory for that → DFS & → BFS

Directory as a file support operations.



special op on directory which is not perf.  
on regular files

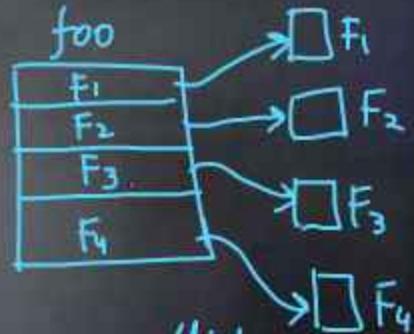
Traverse

Q. 5

(MSQ)

Consider a linear list based directory implementation in a file system. Each directory is a list of nodes, where each node contains the file name along with the file metadata, such as the list of pointers to the data blocks. Consider a given directory foo. Which of the following operations will necessarily require a full scan of foo for successful completion?

- A. Opening of an existing file in foo X
- B. Creation of a new file in foo ✓
- C. Renaming of an existing file in foo ↗ name conflict
- D. Deletion of an existing file from foo X



# File Management - 2

# File system    Implementation #

↳ Layered File system :-

Application programs



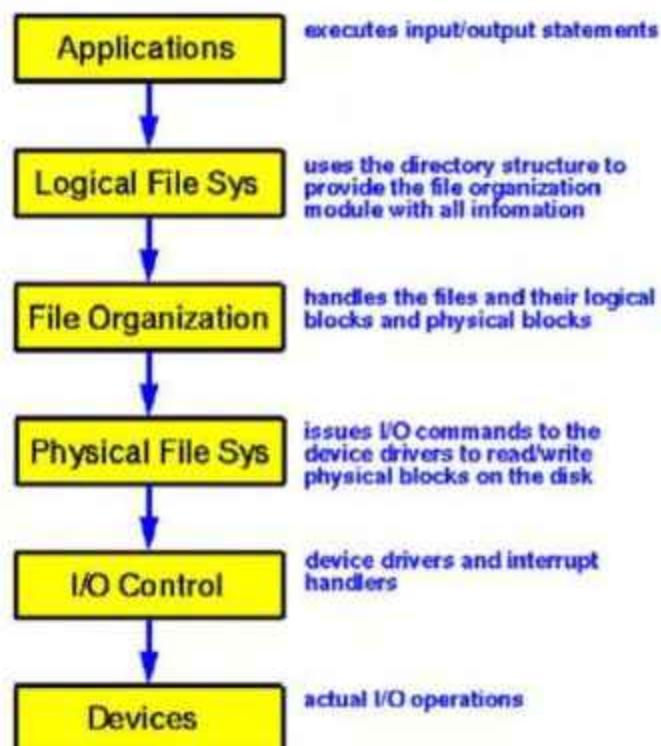
Logical file system

File organization Module



Basic file system

} core part



I/O control (Device driver +  
Interrupt Handler)

Devices (Hardware + controller  
disk cards)

**The I/O control level** consists of device drivers and interrupt handlers to transfer information between the main memory and the disk system. A device driver can be thought of as a translator. Its input consists of high-level commands such as "retrieve block 123." Its output consists of low-level, hardware-specific instructions that are used by the hardware controller, which interfaces the I/O device to the rest of the system.

**The Basic File System** needs only to issue generic commands to the appropriate device driver to read and write physical blocks on the disk. Each physical block is identified by its numeric disk address (for example, drive 1, cylinder 73, track 2, sector 10). This layer also manages the memory buffers and caches that hold various file-system, directory, and data blocks.

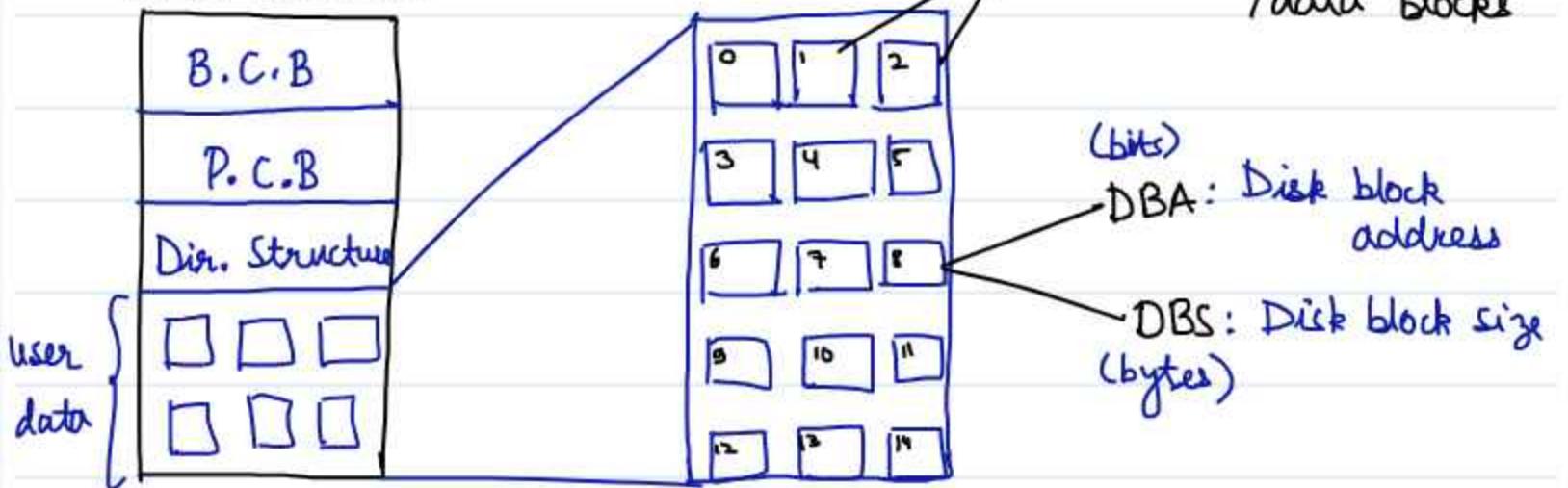
**The File-organization module** knows about files and their logical blocks, as well as physical blocks. By knowing the type of file allocation used and the location of the file, the file-organization module can translate logical block addresses to physical block addresses for the basic file system to transfer. The file-organization module knows about files and their logical blocks, as well as physical blocks. By knowing the type of file allocation used and the location of the file, the file-organization module can translate logical block addresses to physical block addresses for the basic file system to transfer. a translation is needed to locate each block, the file-organization module also includes the free-space manager, which tracks unallocated blocks and provides these blocks to the file-organization module when requested.

**The Logical File System** manages metadata information. Metadata includes all of the file-system structure except the actual data (or contents of the files). The logical file system manages the directory structure to provide the file-organization module with the information the latter needs, given a symbolic file name. It maintains file structure via file-control blocks. A file control block (FCB) (an anode in UNIX file systems) contains information about the file, including ownership, permissions, and location of the file contents.

## # Allocation Methods :-

Methods for allocating disk space to files ,  
directories or any other D.S..

## Disk Partition

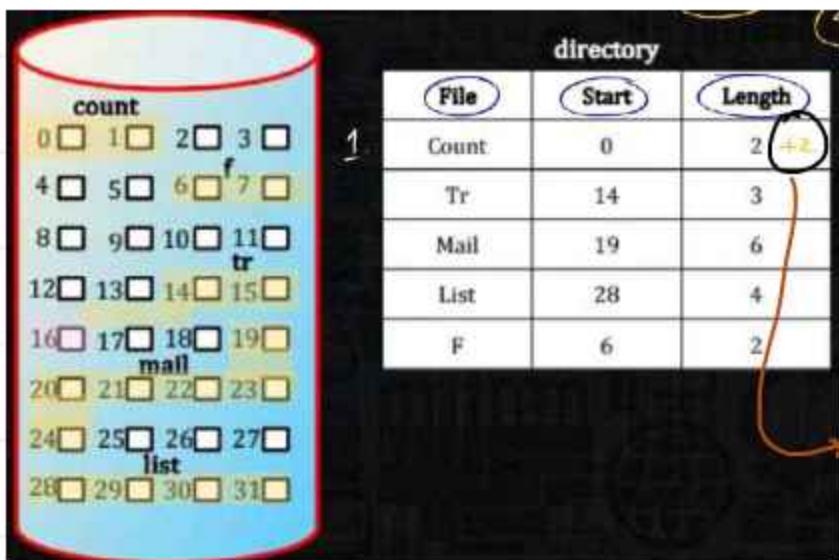


Ex:- DBA = 16 bits DBS = 1 KB  
 $\downarrow$   
 $2^{16}$  blocks

$$\text{Max possible file size} = \text{Disk size } 2^{16} \times 1 \text{ KB} = 64 \text{ KB}$$

$$= (2^{\text{DBA}}) * \text{DBS}$$

Methods for allocating the blocks to file → Allocation methods



Contiguous

non

## Index

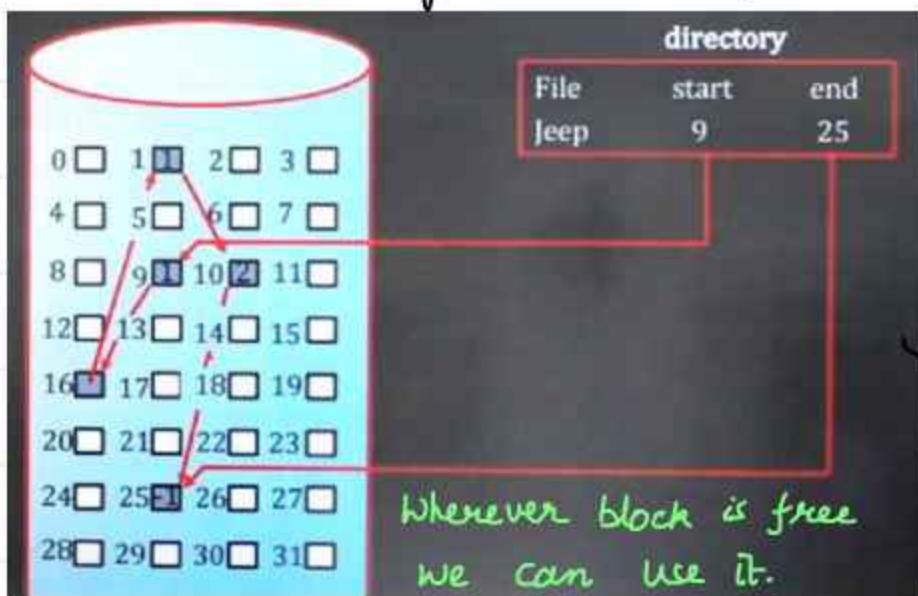
Performance :- 1. Int. frag. (Last block)

2. Ext. Fragmentation ✓
  3. Increasing file size (Inflexible)  
+ 2 blocks may not be available.

#### 4. Type of Access :- Sequential / random or direct

4

Non-contiguous (linked)



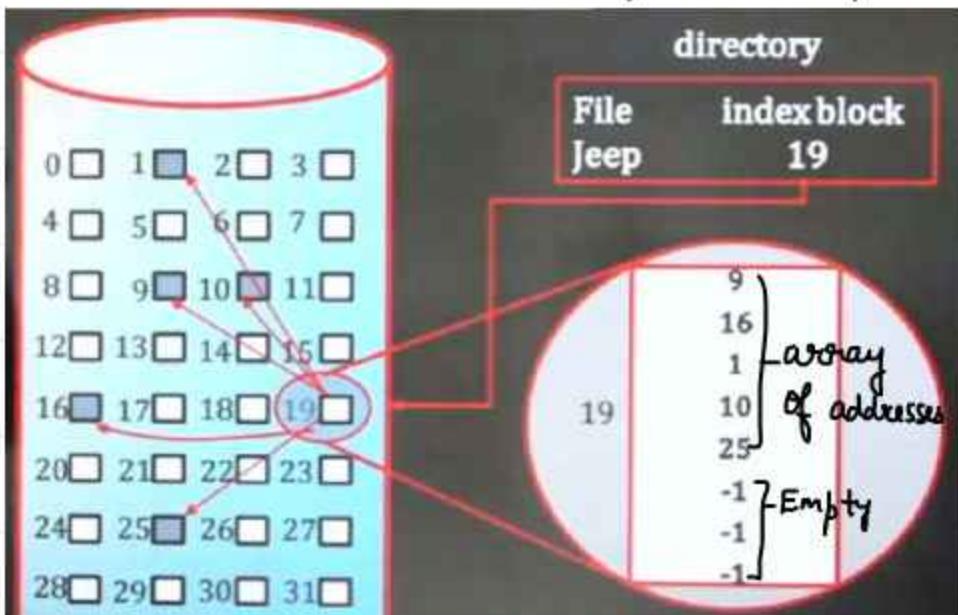
↓  
Like an array  
↓  
Faster

→ creating a linked list  
of data blocks.

1. Internal Frag ✓ (Last block)
2. External Frag X      needs ending address
3. Increasing File size ✓ (Flexible) → address
4. Types of access → only sequention → slow
5. Performance → Pointers consume disk space (overhead)
6. Vulnerability of Pointers      can get broken  
↓  
will truncate the file.

### 3. Indexed Allocation of Disk spaces

5



- doesn't contain data.
- Each file has Index block.
- Index block holds address of data blocks of file.

Not block to block links , index itself is pointing to all blocks.

Eg:- DBA = 16 bits      DBS = 1 KB      Max file size with 1 index block = ?

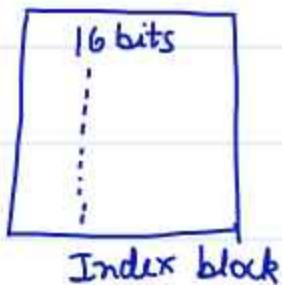
$2^{16}$  blocks

1 KB

$$\begin{aligned} \text{File size} &= 2^9 \times 1 \text{ KB} \\ &= 0.5 \text{ MB} \end{aligned}$$

$$16 \text{ bits} \times x = 2^{10} \times 2^3 \text{ bits}$$

$$2^4 \text{ bits} \times x = 2^{13} \text{ bits}$$



$$x = 2^9$$

→ no of blocks addresses Index block can store.

Q File system support a DBS = 4KB . Each block can hold 512 addresses . Size of DBA in bits.

$$2^{15} \text{ bits} = x \text{ bits} \times 2^9 \Rightarrow x = 64 \text{ bits}$$

- Can I increase file size → YES ✓

- Performance :-  
Internal Frag → ✓  
Take more than one index block.

External Frag → X (Non cont)

Increasing File size → ✓ (Flexible)

Type of Access → Both seq & random

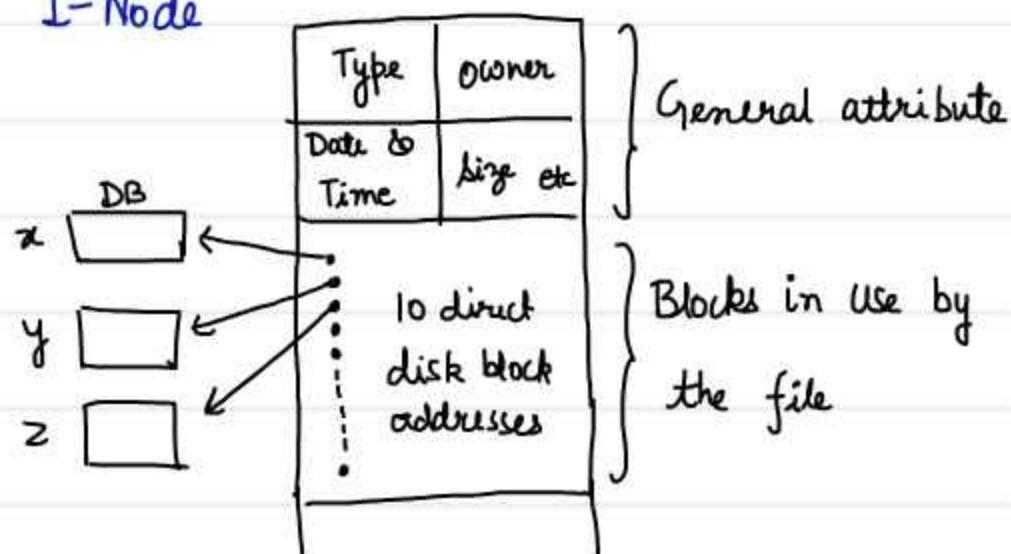
Reliability → More than linked as no pointers  
but if Index block....

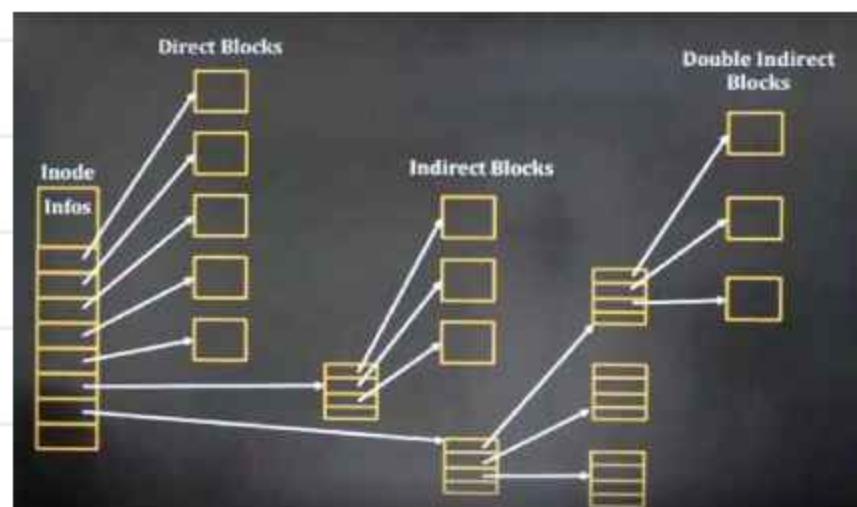
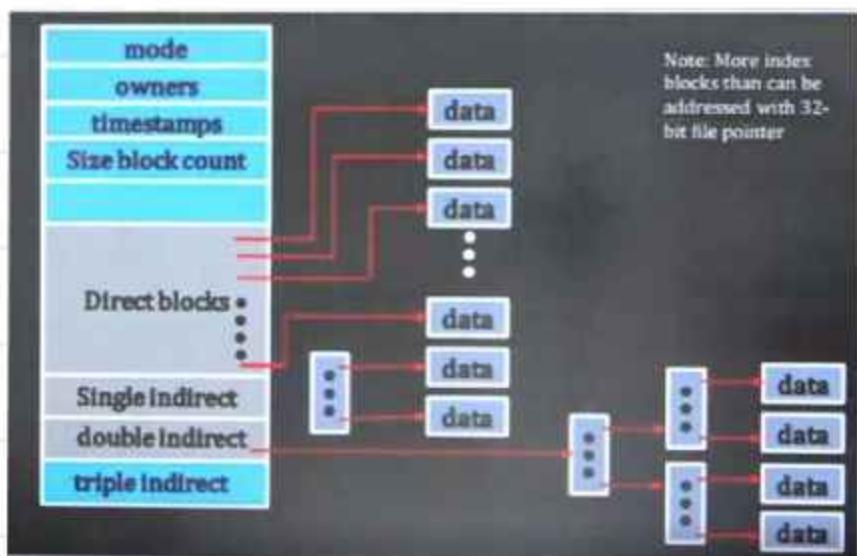
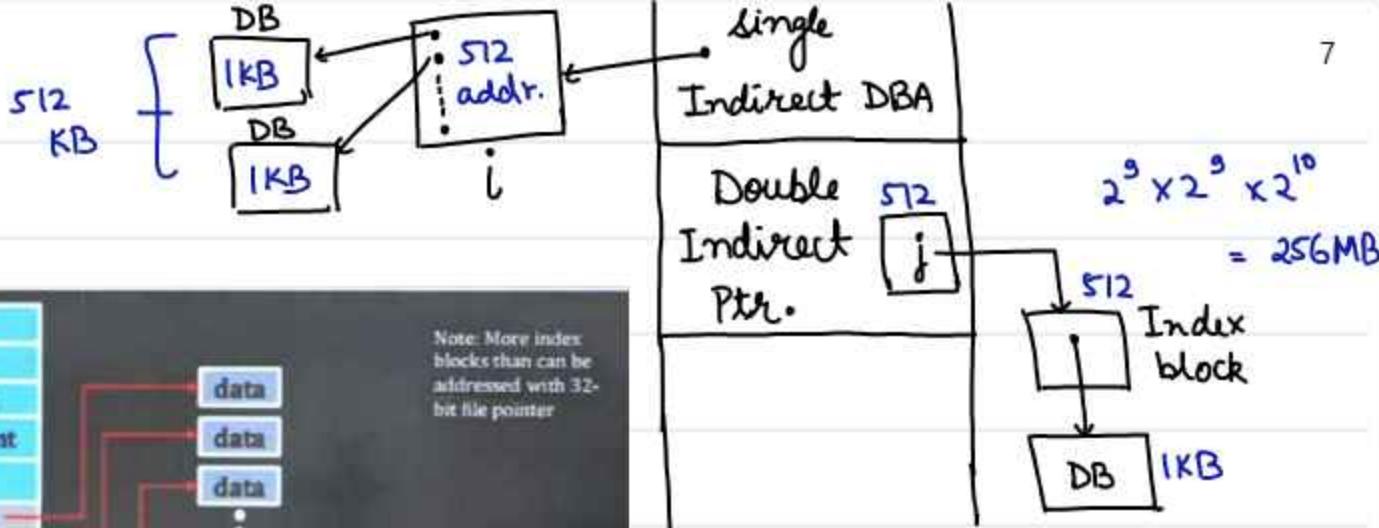
Space overhead → some disk space will be consumed for Index blocks.

## # Case study #

1. UNIX / LINUX :- Each file node is associated with I-Node

File Name	I-Node
gym.c	23





## 2. DOS / Windows

Directory  $\Rightarrow$

File Name	other Attr.	Block Info	
gym.c	-----	First DBA	Last DBA
		5	8

How to get info. of

F.A.T / Master File Table  $\leftarrow$  other tables  
 ↘ File allocation table

0
1 <2>
2 <4>
3 <7>
4 <6>
5 start <3>
6 <?> end
7 <1>
8 --- -1 -----

Entries = No of blocks



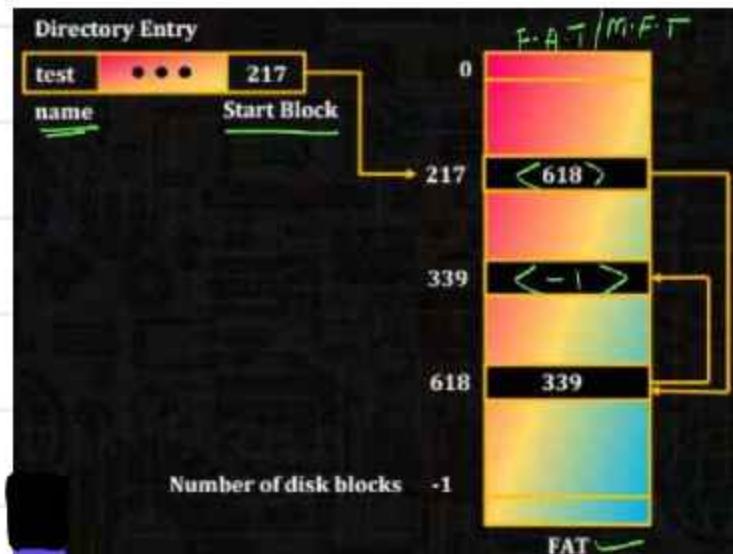
contains address of next data block.

- Entry size = DBA

<5, 3, 7, 1, 2, 4, 6, 8> } Info of all blocks

Like Linked Allocation

→ Tabular based / Array based



Consider a Unix I-node structure that has 8 direct Disk Block Addresses and 3 Indirect Disk Block Addresses, namely Single, Double & Triple.

Disk Block Size is 1Kbytes & each Block can hold 128 Disk Block Addresses.

Calculate

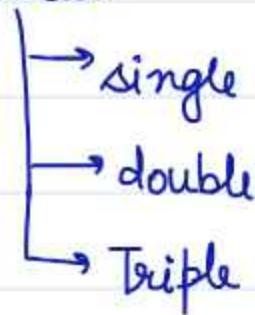
- Maximum File Size with this I-Node Structure?
- Size of Disk Block Address?
- Is this File Size possible over the given Disk?

8 direct DB addresses, 3 Indirect

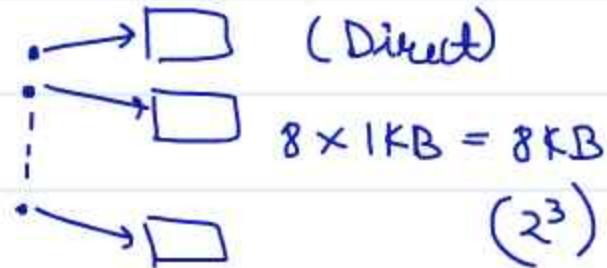
$$DBS = 1KB$$

$$8 \text{ Kbytes} = 128 \times \text{DBA}$$

$$\frac{\text{bites } 2^13}{2^7} = 2^6 \text{ bites} = \text{DBA}$$

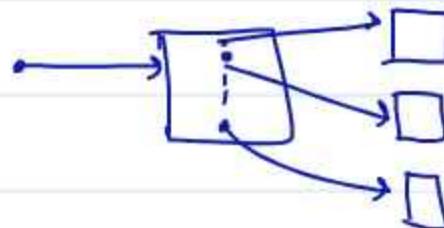


Maximum File size  $\Rightarrow$



+ Indirect

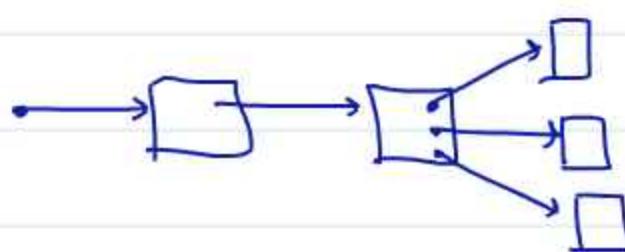
(single)



$$128 \times 1KB = 128KB$$

$(2^7)$

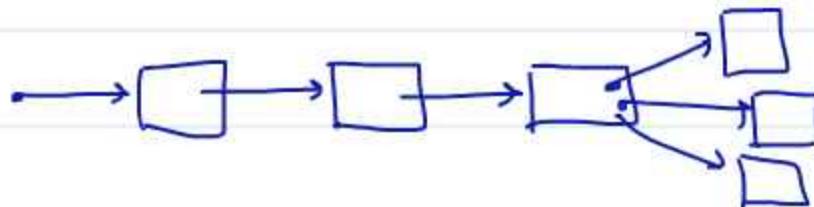
double



$$128 \times 128KB$$

$2^{14}KB$

triple



$$2^{21}KB$$

$$\text{Max size} = (2^{21} + 2^{14} + 2^7 + 2^3) KB \approx 2GB$$

(c) Disk size =  $2^{64}$  blocks  $\times$  1KB block size  
 $\gg 2GB$

✓ yes

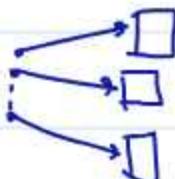
Q The index node (inode) of a Unix-like file system has 12 direct, one single-indirect and one double-indirect pointers. The disk block size is 4 kB, and the disk block address is 32- bits long. The maximum possible file size is \_\_\_ GB (rounded off to 1 decimal place)

12 direct , 1 single indirect  
1 double indirect

DBS = 4KB

DBA = 32 bits

Max possible file size = ?



$$12 \times 4\text{KB} = 48\text{KB}$$

$$4\text{B} \xrightarrow{32 \text{ bit}} \times x = 4\text{KB} \Rightarrow x = 2^{10}$$

$$2^{10} \times 4\text{KB} = 4\text{MB}$$

$$2^{10} \times 2^{10} \times 4\text{KB} = 4\text{GB}$$

+

$\sim 4\text{GB}$



The Data Blocks of a very large file in the Unix File System are allocated using

Contiguous allocation

Linked allocation

Indexed allocation

An extension of indexed allocation.

Q Using a Larger Block size in a Fixed Block Size File System leads to

- Better Disk Throughput but Poorer Disk Space Utilization. No of bytes you can read with one disk access Int. frag
- Better Disk Throughput and Better Disk Space Utilization
- Poorer Disk Throughput but Better Disk Space Utilization
- Poorer Disk Throughput and Poorer Disk Space Utilization

Q A FAT (File allocation table) based file System is being used and the total overhead of each entry in the FAT is 4 bytes in size. Given a  $100 \times 10^6$  bytes disk on which the file System is stored and data block size is  $10^3$  bytes, the maximum size of a file that can be stored on this disk in units of  $10^6$  bytes is \_\_\_\_.

$$\begin{array}{l}
 \text{Entry size} = 4B \\
 \text{No of blocks in FAT} = \frac{100 \text{ shiba}}{10^3} = 0.1 \text{ shiba} \\
 \text{DS} = 100 \times 10^6 \text{ B (say)} \\
 \text{DBS} = 10^3 \\
 \text{Max File size} = \text{Given Disk size} - \text{FAT size} \\
 \qquad\qquad\qquad\downarrow \qquad\qquad\qquad\downarrow \\
 \qquad\qquad\qquad 100 \text{ shiba B} \qquad\qquad\qquad 0.4 \text{ shiba B} \\
 \qquad\qquad\qquad = 99.6 \text{ shiba B}
 \end{array}$$

# File Management - 3

Q

Consider a File System that stores 128 Disk Block Addresses in the index table of the Directory. Disk Block Size is 4 Kbytes. If the file size is less than 128 Blocks, then these addresses act as direct Data Block addresses.

However, if the File Size is more than 128 Blocks, then these 128 addresses in the Index table point to next level Index Blocks, each of which contain 256 Data block addresses. What is the Max File Size in this File System?

128 DBAs

DBS = 4KB

→ Min file size.

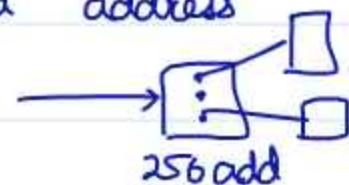
File size < 512 KB



128 address act as  
direct data address

File size > 512 KB ]

128 address



$$\text{Max File size} = 128 \times 256 \times 4\text{KB}$$

$$2^7 \quad 2^8 \quad 2^{12} = 2^{27} = 128\text{ MB}$$

Q

A File System with a One-level Directory structure is implemented on a

disk with Disk Block Size of 4 Kbytes. The disk is used as follows:

Disk Block 0 ✓ : Boot Control Block

Disk Block 1 ✓ : File Allocation Table, consisting of one (10-bit) entry per

Data Block, representing the Data Block Address of the next Data Block in the files.

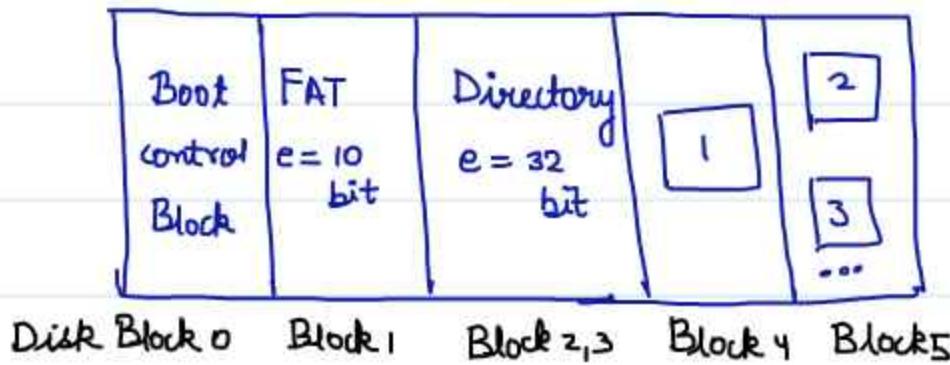
Disk Block [2, 3] : Directory with 32-bit entry per File.

Disk block 4 : Data block 1.

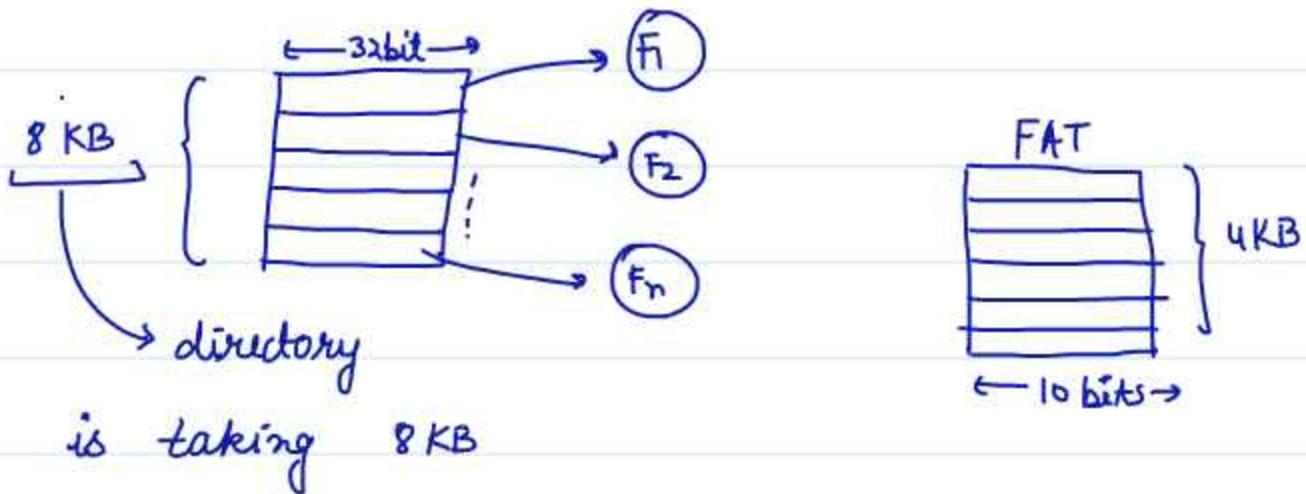
Disk Block 5 : Data Block 2,3 etc;

(a) What is the Maximum possible number of Files?

(b) What is the Maximum Possible File size in Bytes?



Max no of files = ?  
Max file size = ?



$$\text{No of files} = \text{No of entries} = \frac{8 \text{ KB}}{4 \text{ B}} = 2\text{K}$$

$$\text{FAT entry} = 10 \text{ bit} = \text{DBA}$$

$$\begin{aligned} \text{Max file size} &= \text{Disk size} - (\text{control overhead}) \\ &\downarrow \\ & (2^{\text{DBA}}) \text{ DBs} - (\text{Block 0, 1, 2, 3}) \end{aligned}$$

$$= (2^{10}) \times 4 \text{ KB} - 16 \text{ KB}$$

$$= 2^{12} \text{ KB} - 16 \text{ KB} = \underline{4080 \text{ KB}}$$

# # Disk Free space Management Algo #

Disk size = 20 MB

No of blocks = ?

DBS = 1 KB

$$\frac{\text{Disk size}}{\text{Block size}} = \frac{20 \text{ MB}}{1 \text{ KB}}$$

DBA = 16 bit

Block size

or

$$2^{\text{DBA}} = 2^{16} = 64 \text{ K}$$

Max possible

To refer 20K blocks, how many bits are needed?

$$\log_2(2^{12} \times 5) = 12 + \log_2(5) = 12 + 2.3 = 14.3$$

$\approx 15$  bits

or can do directly

K needs 10 bits

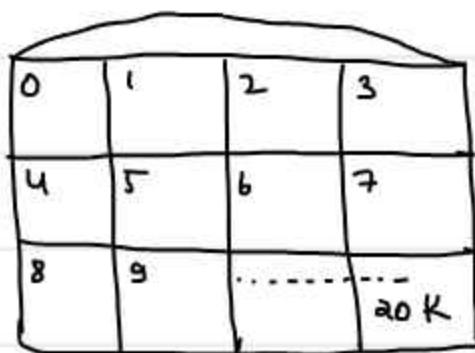
20 needs  $\underbrace{5 \text{ bits}}$ , 15 bits

We need 15 bits but are using 16 bits

Max possible disk size =  $2^{16} \times 1 \text{ KB} = 64 \text{ MB}$

Given disk size = 20 MB

Given disk size  $\leq$  Max possible  
 $(2^{PBA})$  DBs



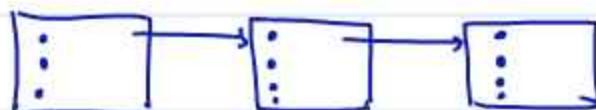
1) Linked List of free blocks



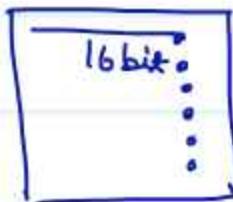
Initially chain of 20k free blocks.

If I want to allocate some block to file then delete 10 blocks from LL & allocate.

2) Free List :- Linked list of blocks containing addresses of free blocks



How many blocks of free list are needed to store 20k free DB's.



$$16 \text{ bit} \times x = 1 \text{ KB}$$

$$x = 512$$

512 addresses in one block

$$512 \times y = 20 \text{ K}$$

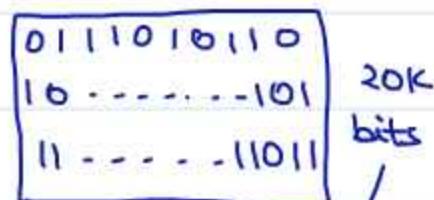
$$y = \underline{40}$$

### 3) Bit Map / Vector

Keeps track of all blocks unlike free list.

Associate a binary bit with each block.

$0$  → free  
 $1$  → In-use



How many blocks are needed to store our bit map (20K bits)

$$\text{DBS} = 1\text{KB} \quad \frac{20 \text{ K bits}}{2^3 \text{ 1K bits}} = \frac{20}{8} \Rightarrow 3 \text{ blocks}$$

FREE LIST → Faster } keeps track of only free blocks, no need to search

### 4) Counter Method :- When we have contiguous free blocks

start Free DBA No of CG free block

Start Free DBA	No of CG free block	
5	45	→ FF
85	200	→ WF
325	125	
600	25	→ BF

Let's say we have a file of 20 blocks now which blocks to allocate?  
First fit, Best fit, Worst fit

→ After allocation we need to update the tuple.

**Q** Consider a Disk with 'B' Blocks, 'F' of which are free. Disk Block Address is 'D' bits, Disk Block Size is 'X' Bytes.

(A) Calculate

- (i) Given Disk Size.
- (ii) Maximum possible Disk Size.
- (iii) Relation between 'B' & 'D'.

(B) What is the condition in which Free List uses less space than Bit Map?

$$\text{Total blocks} = B$$

$$\text{Free blocks} = F$$

**A** DBA = D bits      DBS = X bytes

a) Given disk size =  $(B * X)$  bytes

b) Max possible size =  $(2^D) * X$  bytes

c) Relation b/w B & D bits :-  $B \leq 2^D$

**B** Free list uses less space than Bit map :-

$$F \times D < B$$

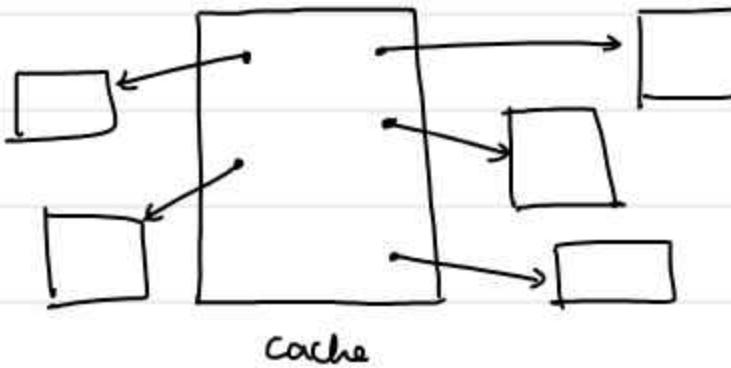
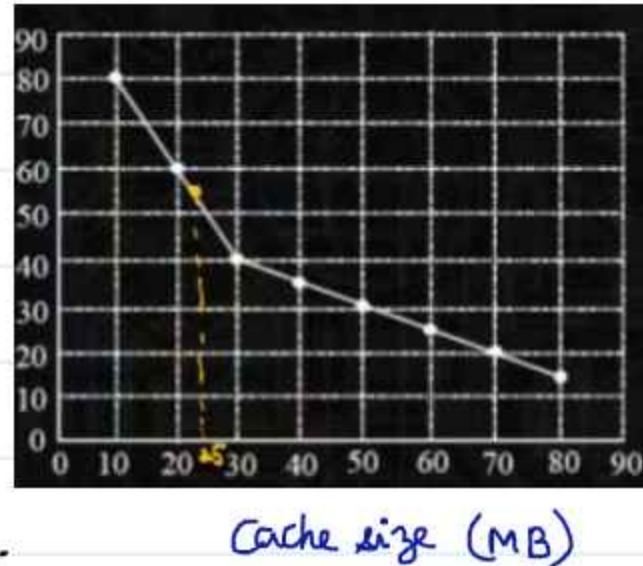
**Q** The beginning of a free space Bit-Map looks like this after the Disk Partition is first formatted: 1000 0000 0000 0000 (the first block is used by the Root Directory). The System always searches for free blocks starting at the lowest numbered block, so after writing file A, which uses 6 blocks, the bitmap looks like this: 1111 1110 0000 0000. Show the Bit-Map after each of the following additional actions as HEX Code:

File A	1000 0000 0000 0000	HEX: 8000
File B is written, using 5 blocks	1111 1110 0000 0000	• 1000 0000 0000 0000
File deleted	1111 1110 0000 0000	• 1111 1110 0000 0000
File C is written, using 8 blocks	1111 1111 1111 0000	• 1111 1110 0000 0000
File B is deleted.	1111 1111 1111 0000	• 1111 1111 1111 0000

- b) 81F0  
1000 0001 1111 0000
- c) FFFC  
1111 1111 1111 1100
- d) FE0C  
1111 1110 0000 1100

Q A File System uses an in-memory cache to cache disk blocks. The miss rate of the cache is shown in the figure. The latency to read a block from the cache is 1 ms and to read a block from the disk is 10 ms. Assume that the cost of checking whether a block exists in the cache is negligible. Available cache sizes are in multiples of 10 MB.

The smallest cache size required to ensure an average read latency of less than 6ms is \_\_\_\_\_ MB.



$$\text{Latency}_{\text{cache}} = 1 \text{ ms}$$

$$\text{Latency}_{\text{disk}} = 10 \text{ ms}$$

$$\text{Read Latency} < 6 \text{ ms} \quad \text{Cache size} = \alpha \times 10 \text{ MB}$$

$$\rho \times 10 + (1-\rho)1 < 6 \quad . \quad \text{miss \% rate} = \rho$$

$$10\rho + 1 - \rho < 6$$

$$9\rho < 5 \Rightarrow \rho < \frac{5}{9} \approx 55\%$$

According to graph  $\Rightarrow$  30 MB

~~Q~~ The amount of Disk Space that must be available for Page storage is related to Maximum number of Processes 'N', the number of Bytes in Virtual Address Space 'B' and the number of Bytes in RAM 'R'. Give an expression for the worst case Disk Space required.

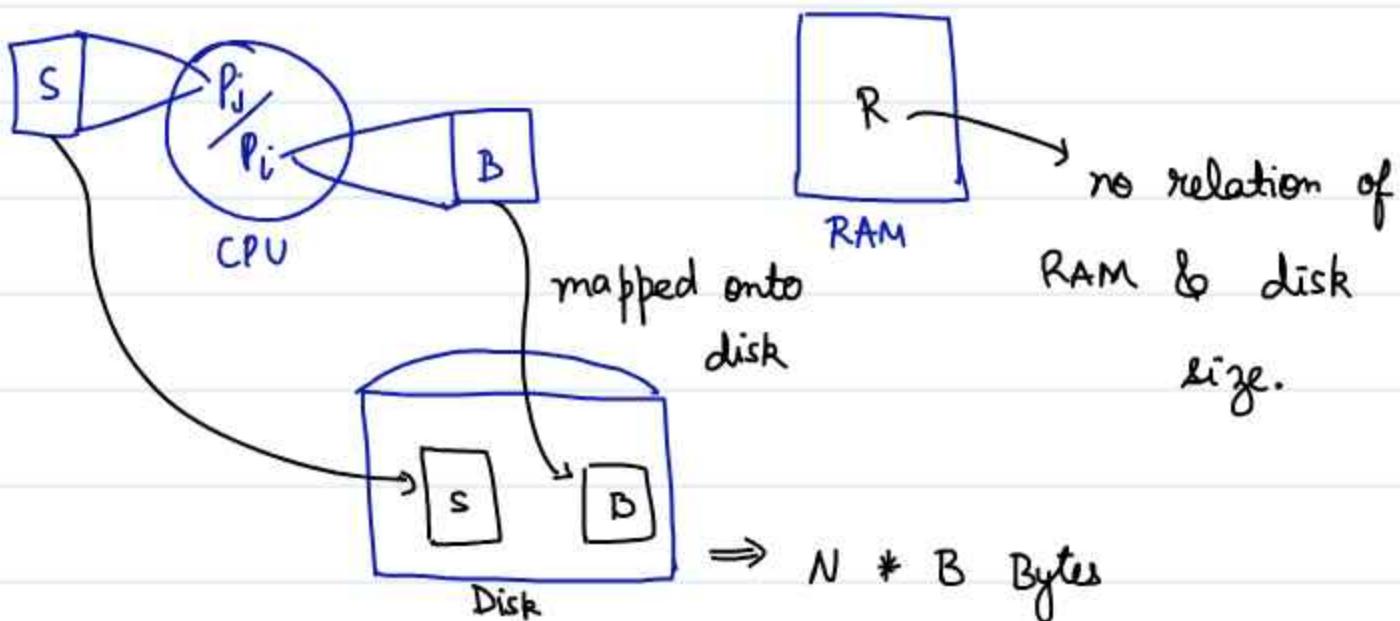
No of processes = N

VAS size = B bytes

worst case disk space

Mem size = R bytes

required = ?



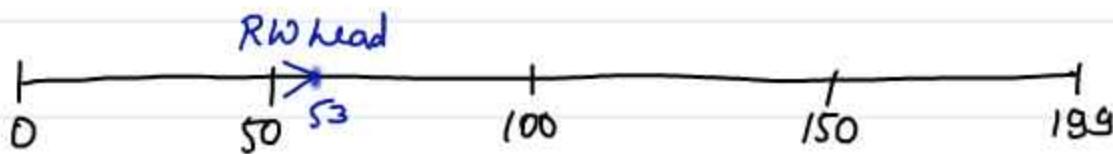
## # Disk Scheduling #

- Just like CPU can serve one process at a time similarly the disk can only serve one I/O req.
- Just like short term scheduler, disk scheduler also exist.

let's move directly to the disk scheduling techniques

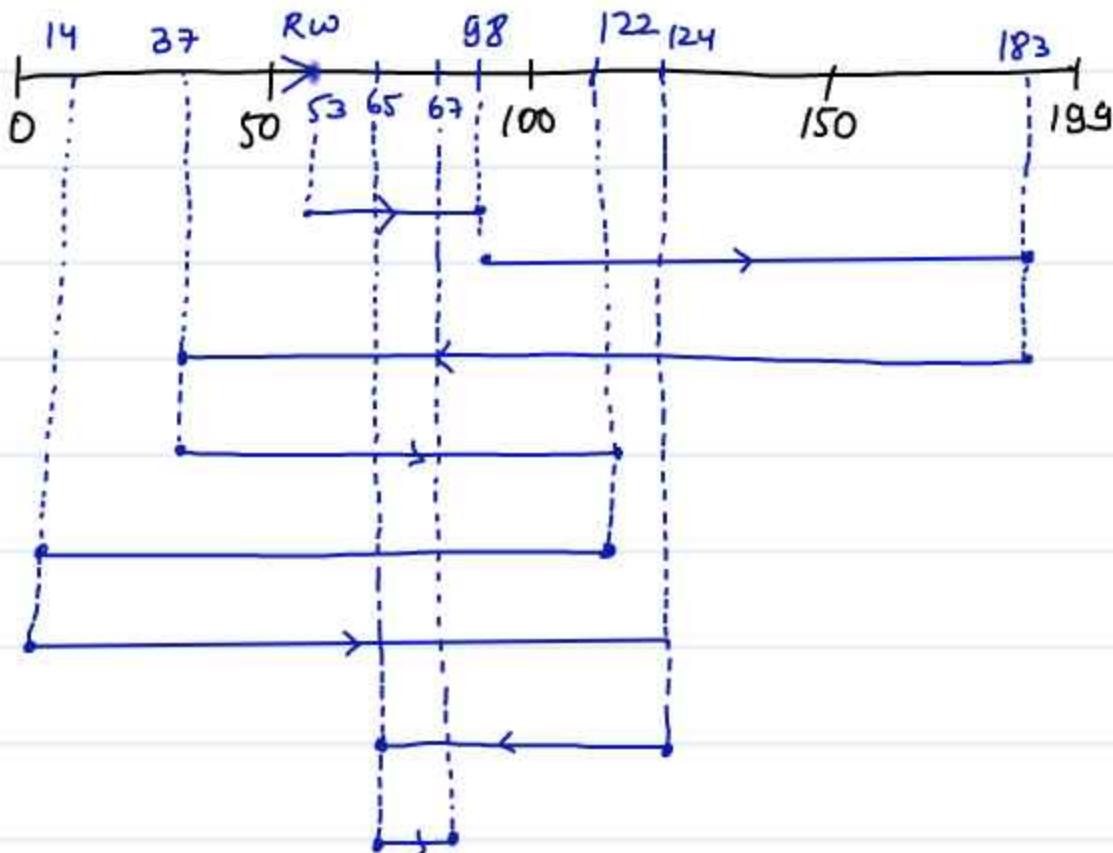
Process Requests : 98, 183, 37, 122, 14, 124, 65, 67

→ Track no / cylinder no.



200 Tracks are distributed evenly on the surface.  
(0-199)

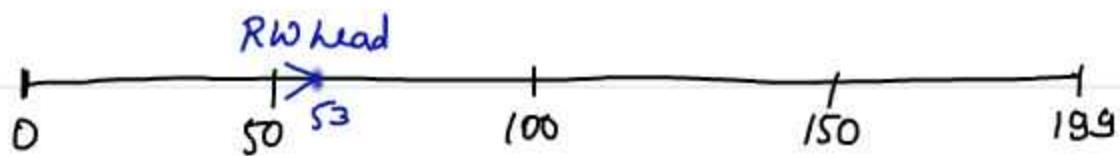
a) FCFS :-



$$\text{Total seeks} = 640$$

$$\text{Avg seek} = \frac{\text{Total}}{\text{No of req.}} = \frac{640}{8} = 80$$

⑥ Shortest Seek Time First (Nearest/closest track next)



$S_3 \rightarrow 65 \rightarrow 67 \rightarrow 37 \rightarrow 14 \rightarrow 98 \rightarrow 122 \rightarrow 124 \rightarrow 183$

Total seeks = 236

⑦ SCAN (Elevator/Lift)

↳ Top to bottom & Bottom to Top serving

For scan, apart from knowing the position of RW head, should also know the dir<sup>n</sup> (Def: →)

331  
Seeks [  $53 \rightarrow 65 \rightarrow 67 \rightarrow 98 \rightarrow 122 \rightarrow 124 \rightarrow 183 \rightarrow 199$  ]  
 $14 \leftarrow 37 \leftarrow$  ]

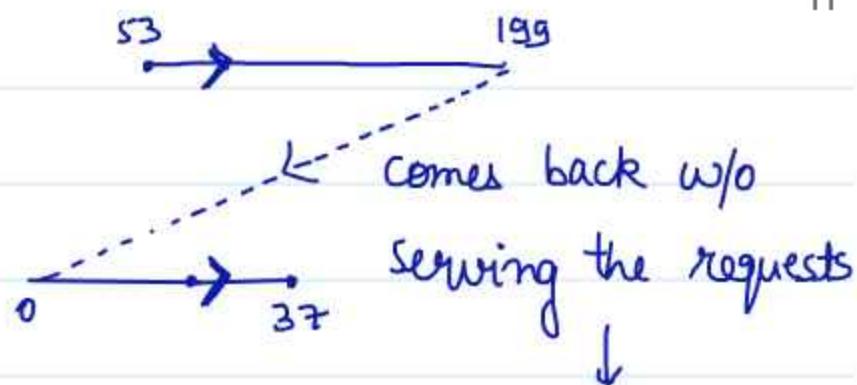
To take backward turn, need to go to last.



Drawback [Extra seeks] + [starvation to the processes in opp dir<sup>n</sup>]

⑧ LOOK :- Instead of going to last, take turn at last request, rest exactly like SCAN. Seeks = 299

e) C-SCAN (circular) :-

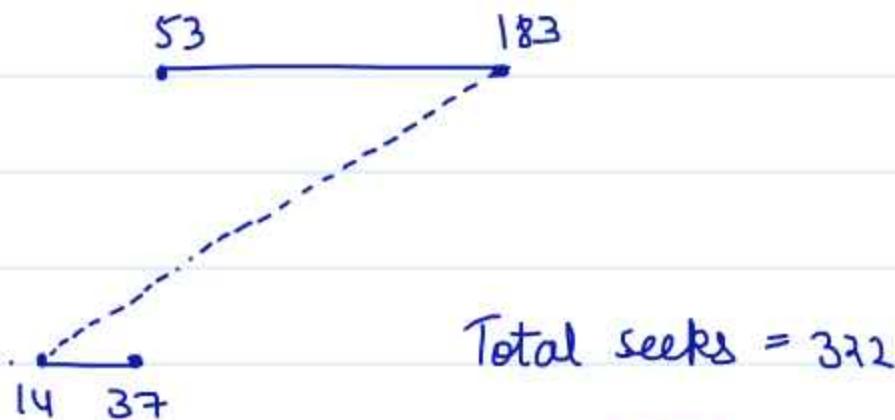


comes back w/o  
serving the requests  
↓

$$382 = (199 - 53) + (199 - 0) + (37 - 0) \quad \text{seeks counted}$$

f) C-LOOK (circular) :- Don't go to the extremes

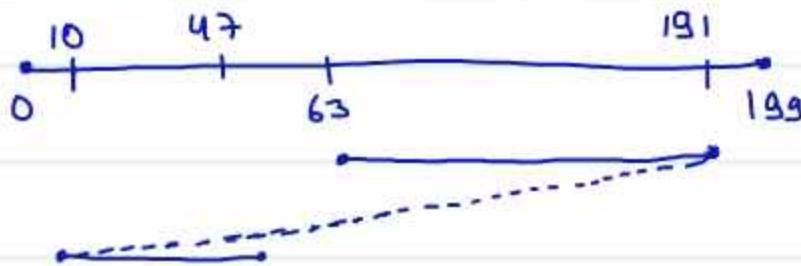
Instead of going to 199 & 0 it goes to 183 & 14.



Total seeks = 372

Q

Consider a disk queue with requests for I/O to blocks on cylinders 47, 38, 121, 191, 87, 11, 92, 10. The C-LOOK scheduling algorithm is used. The head is initially at cylinder number 63, moving towards larger cylinder numbers on its servicing pass. The cylinders are numbered from 0 to 199. The total head movement (in number of cylinders) incurred while servicing these requests is



$$(191 - 63) +$$

$$(191 - 10) +$$

$$346 = (47 - 10)$$

Same as  
Track no.

**Q**

Disk requests come to disk driver for cylinders 10, 22, 20, 2, 40, 06 and 38, in that order at a time when the disk drive is reading from cylinder 20. The seek time is 6 msec per cylinder.

Compute the total seek time if the disk arm scheduling algorithm is:

- (a) First come first served.
- (b) Closest cylinder next.

**A**

Consider a storage disk with 4 platters (numbered as 0, 1, 2 and 3), 200 cylinders (numbered as 0, 1, .., 199), and 256 sectors per track (numbered as 0, 1, .., 255). The following 6 disk requests of the form [sector number, cylinder number, platter number] are received by the disk controller at the same time:

[120, 72, 2], [180, 134, 1], [60, 20, 0] [212, 86, 3], [56, 116, 2], [118, 16, 1]

Currently the head is positioned at sector number 100 of cylinder 80, and is moving towards higher cylinder numbers. The average power dissipation in moving the head over 100 cylinders is 20 milliwatts and for reversing the direction of the head movement once is 15 milliwatts.

Power dissipation associated with rotational latency and switching of head between different platters is negligible. The total power consumption in milliwatts to satisfy all of the above disk requests using the Shortest Seek Time First disk scheduling algorithms is \_\_\_\_\_

**Ans:-**

Platters  $\rightarrow$  (0, 1, 2, 3)

• 200 tracks on each platter

sector      track

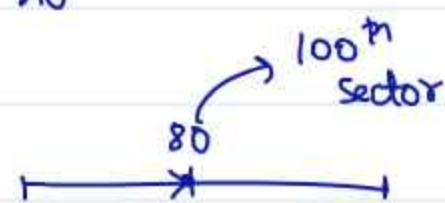
• 256 sectors / track

Req. made :- [120, 72, 2]  $\rightarrow$  Platter no

[180, 134, 1]

[60, 20, 0]

[212, 86, 3]



Single RW head

[56, 116, 2]

100 track seek  $\rightarrow$  20mW

for multiple  
platters,

[118, 16, 1]

1 seek  $\rightarrow$   $\frac{1}{5}$ mW

Reversing the dir<sup>n</sup>  $\rightarrow$  15mW

Power dissipation

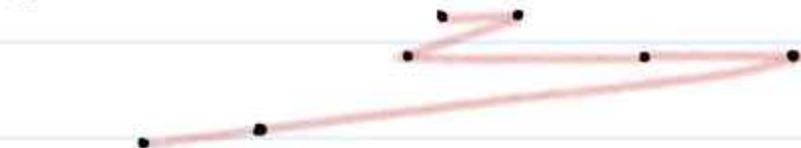
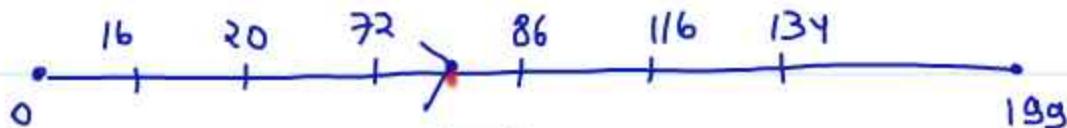
Rot Latency

Switching heads

$\int \rightarrow 0$

sector & diff platter thing ] IGNORE

72, 134, 20, 86, 116, 16



80  $\rightarrow$  86  $\rightarrow$  72  $\rightarrow$  116  $\rightarrow$  134  $\rightarrow$  20  $\rightarrow$  16

$$6 \quad 14 \quad 44 \quad 18 \quad 114 \quad 4 = 200$$

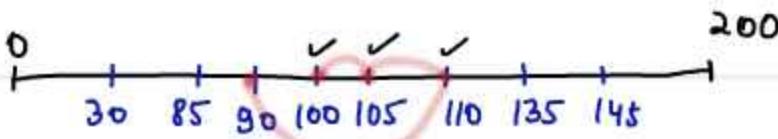
$$200 \times \frac{1}{5} \text{ mW} + 15 \text{ mW} = 85 \text{ mW}$$

$$+ 15 \text{ mW}$$

$$+ 15 \text{ mW}$$

Q

Suppose a disk has 201 cylinders, numbered from 0 to 200. At some time the disk arm is at cylinder 100, and there is a queue of disk access requests for cylinders 30, 85, 90, 100, 105, 110, 135 and 145. If Shortest-Seek Time First (SSTF) is being used for scheduling the disk access, the request for cylinder 90 is serviced after servicing \_\_\_\_\_ number of requests.

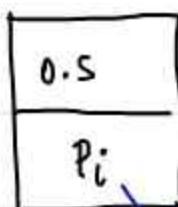


100 itself is a

request  $\Rightarrow$  3 Ans

Q. Consider an operating System capable of loading and executing a single sequential user Process at a time. The disk head scheduling algorithm used is First Come First Served (FCFS). If FCFS is replaced by Shortest Seek Time First (SSTF), claimed by the vendor to give 50 % better benchmark results, what is the expected improvement in the I/O performance of user programs?

uni prog.



} Memory

0%.

→ Doesn't matter whether you use FCFS or SSTF

The scheduling topic is insignificant for UNIPROGRAMMING OS.

Q. Consider a Disk with the Following pertinent details:

Track-Track Time = 1 ms

Current Head Position = 65

Direction: moving towards higher tracks with highest Track being 199.

Current Clock time = 160 ms

Consider the Following Data Set:

Calculate the Time of Decision, Pending Requests, Head Position,

Selected Request, Seek Time using

FCFS, SSTF, SCAN, LOOK, CSCAN,

C-LOOK Algorithms. H.W

Serial No.	Track No	Time of Arrival
1	12	65 ms
2	85	80 ms
3	40	110 ms
4	100	120 ms
5	75	175 ms

FCFS

1	Time of Decision	160	213	286
2	Pending Requests	1, 2, 3, 4	2, 3, 4, 5	3, 4, 5
3	Head Position	65	12	85
4	Selected Requests	Req 1   Track no 12	Req 2   Track = 85	Req 3   40
5	Seek Time	53 seeks x 1 ms	73 ms	25 ms

# Miscellaneous Topics - 1

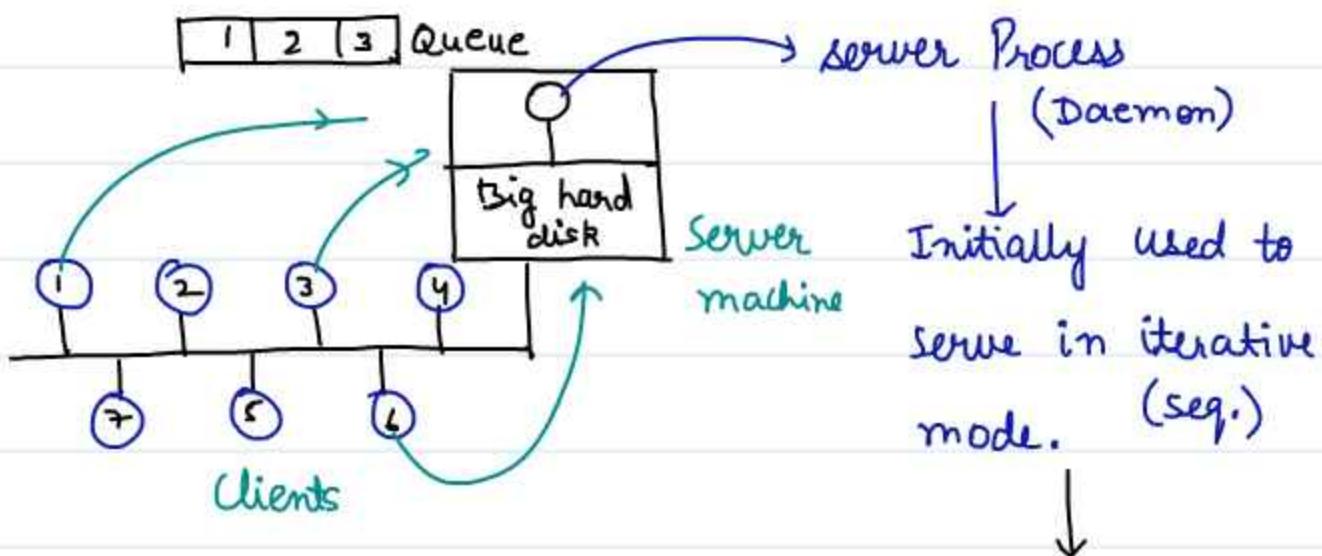
# Threads & Multi-Threading #

↳ Light weight process



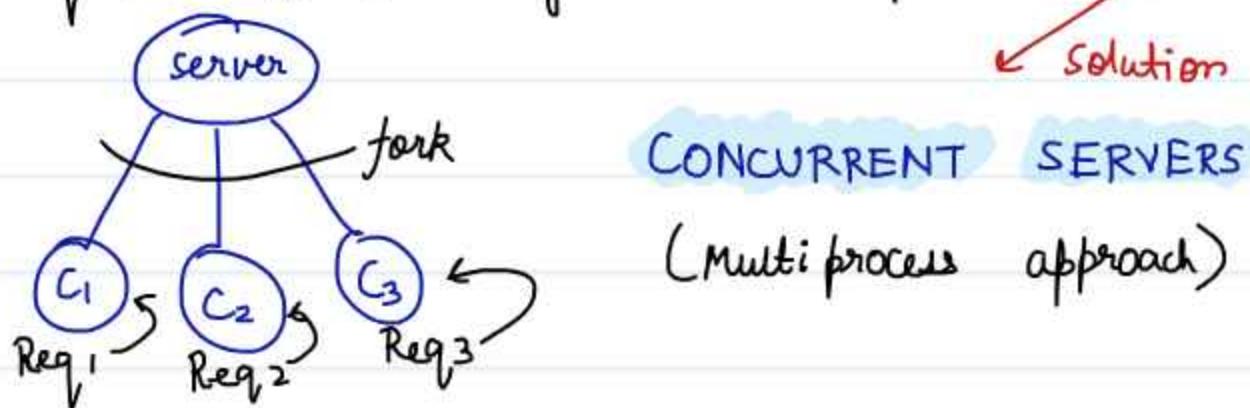
Program in execution state

Client Server Environment



Requests will be assigned to child processes. Starvation / delay

↳ Solution



server observes that there are 3 request pending

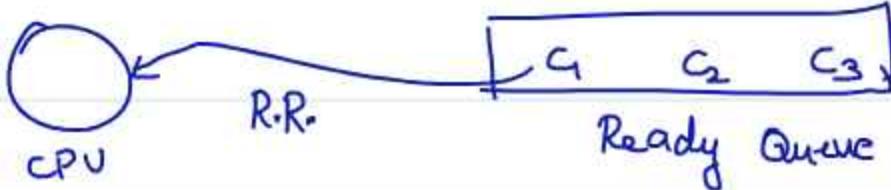
using fork

(exact replica of original)



creates 3 child processes.

Independent



schedule them in such a way that they get an impression that their request is being served.

↓ Severe Drawback!

### Redundancy of code

Code of server process will be copied as it is in the child process.

→ Functionality of all children will be same.

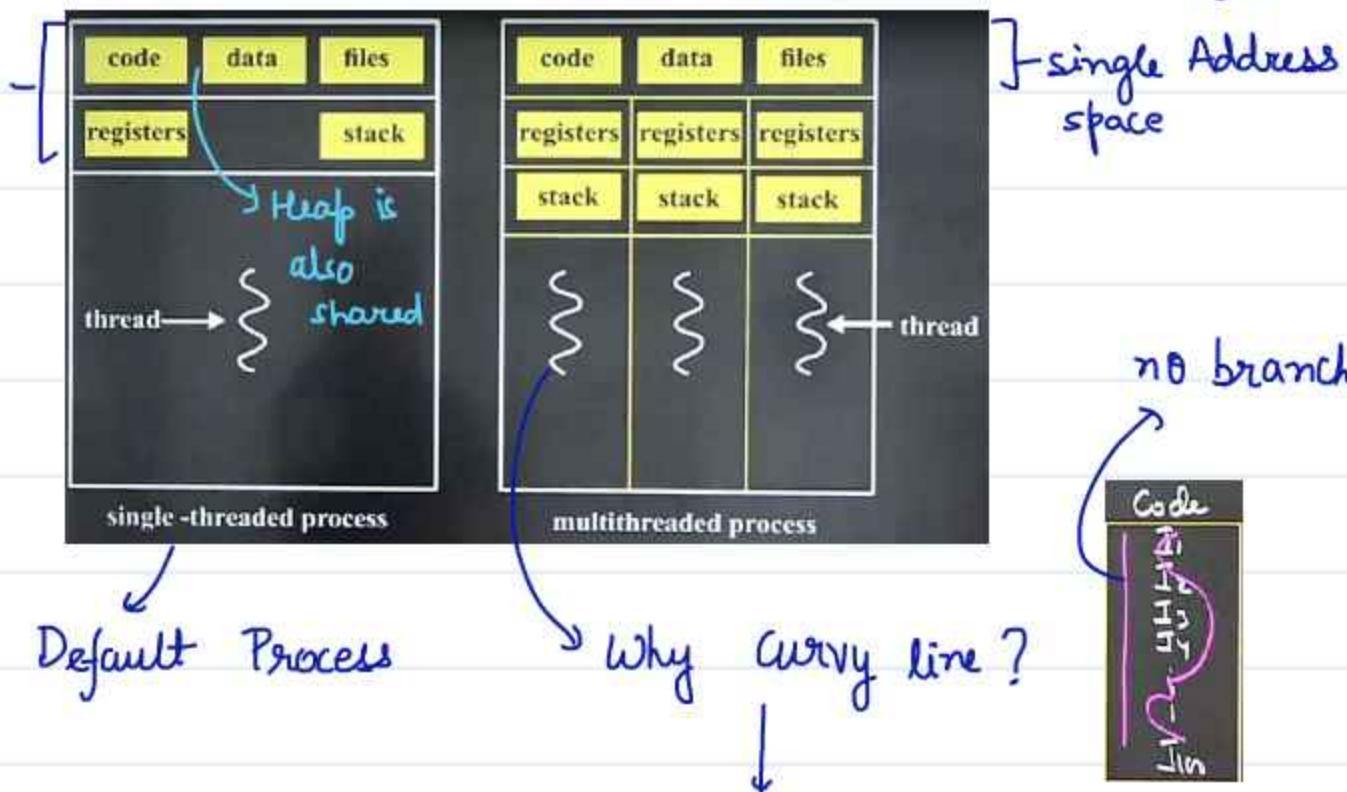
can't we have an architecture in which I will have one section of code & the resources which are separately needed should be given to each process.

→ Memory & Resource wastage will be avoided.

Thread → subprocess which will share all the (light weight) common resources.

Main Reason for Multithreading → Economy  
(Resource sharing)

Address space



Default Process

Why Curvy line?

During the execution of inst. there may be a branch inst. / jump.

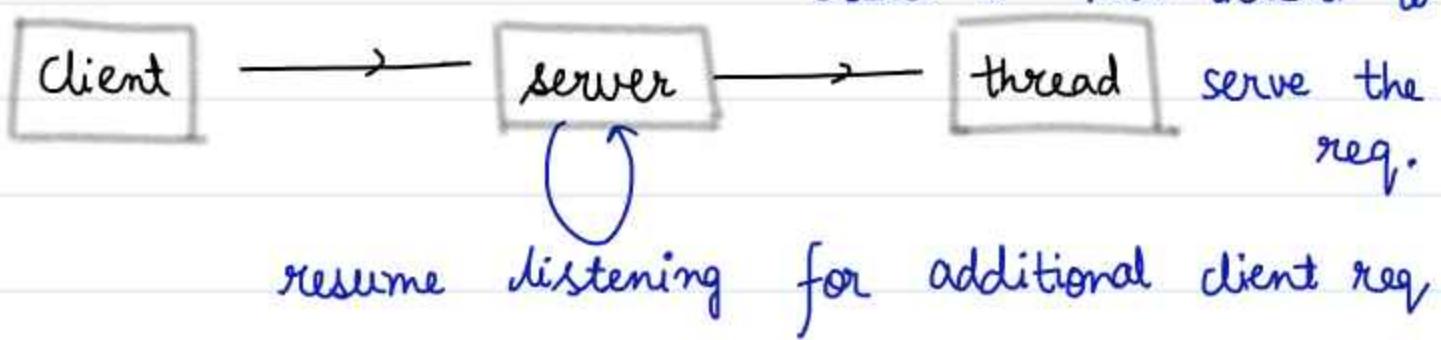
Each thread have its own stack (its own  $f^n$ ) & register.



Concurrent server

→ Multi process approach

Multi - threaded approach



## # Benefits of Multi-threaded program #

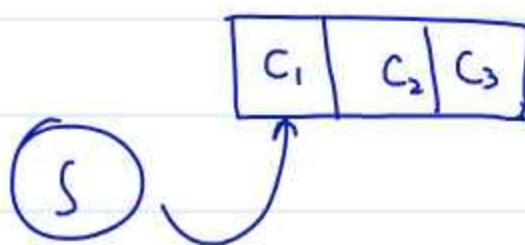
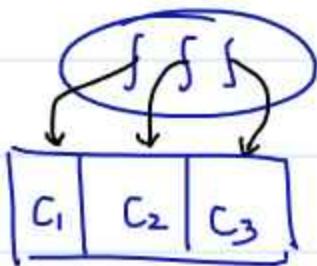
- Responsiveness:** Multithreading an interactive application may allow a program to continue running even if part of it is blocked or is performing a lengthy operation, thereby increasing responsiveness to the user. This quality is especially useful in designing user interfaces. For instance, consider what happens when a user clicks a button that results in the performance of a time-consuming operation. A single-threaded application would be unresponsive to the user until the operation had completed. In contrast, if the time-consuming operation is performed in a separate thread, the application remains responsive to the user.
- Resource sharing:** Processes can only share resources through techniques such as shared memory and message passing. Such techniques must be explicitly arranged by the programmer. However, threads share the memory and the resources of the process to which they belong by default. The benefit of sharing code and data is that it allows an application to have several different threads of activity within the same address space.
- Economy:** Allocating memory and resources for process creation is costly. Because threads share the resources of the process to which they belong, it is more economical to create and context-switch threads. Empirically gauging the difference in overhead can be difficult, but in general it is significantly more time consuming to create and manage processes than threads. In Solaris, for example, creating a process is about thirty times
- Scalability:** The benefits of multithreading can be even greater in a multiprocessor architecture, where threads may be running in parallel on different processing cores. A single-threaded process can run on only one processor, regardless how many are available. We explore this issue further in the following section.

(5) Improved Performance due to less context switch  
 ↓  
 Each thread will have its own control block (TCB)  
 just like PCB

→ size of TCB < PCB

Thread switching is faster than process switching.

6. can utilize multicore architecture & achieve parallelism.



Let's summarize :-

- Thread is like light weight copy of process that executes independently.
- Threads share the same address space.
- Each Thread has a separate Program counter  
↳ may run over diff parts of program.
- Each Thread has a separate stack for independent f" calls.

Parent P forks a child C

- ❖ P and C does not share any memory
- ❖ Need complicated IPC mechanism to communicate
- ❖ Extra copies of code, data in memory

Parent p execute two threads T1 and T2

- ❖ T1 and T2 share parts of the address space
- ❖ Global Variables can be used for communication
- ❖ Smaller memory footprint

**Threads Vs Processes**

- A thread has no data segment or heap.
- A thread cannot live on its own it need to be attached to a process.
- There can be more than one thread in a process. Each thread has its own stack.
- If a thread dies, its stack is reclaimed.
- A process has code, heap, stack, and other segments
- A process has at-least one thread.
- Heads within a process share the same code, files.
- If a process dies, all thread die.

Parallelism: single process can effectively utilise multiple CPU cores

- ❖ Understand the difference between concurrency and the parallelism
- ❖ Concurrency: running multiple threads/ processes at the same time, even on single CPU core, by interleaving their execution
- ❖ Parallelism: running multiple threads/process in parallel over different CPU cores

# # Types of Threads #

User Level



Threads that are created & managed @ VM without any O.S. support

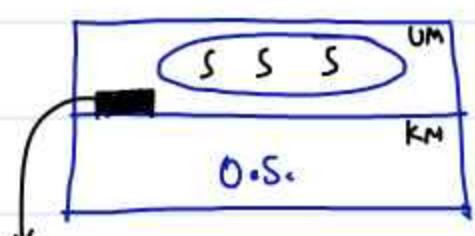


Kernel Level



Threads that are directly created & managed by O.S.  
KTL → part of process but managed independently by process.

Flexibility given to users



] we managed the creation @ user level with some package/library.

some package / library

We call this as Transparency.

Benefits :-

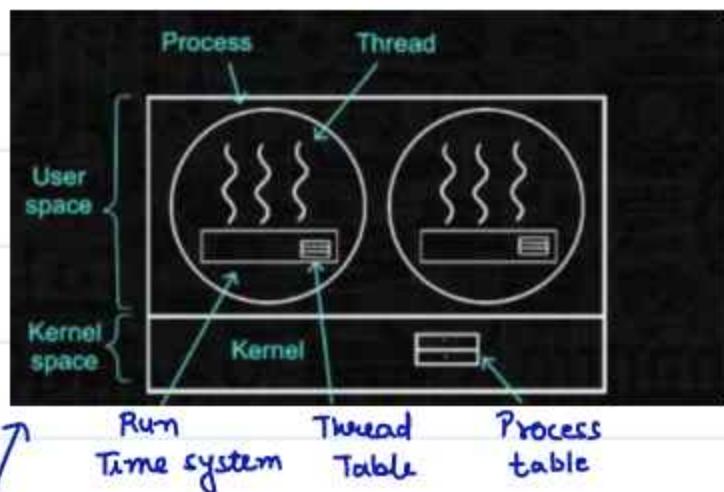
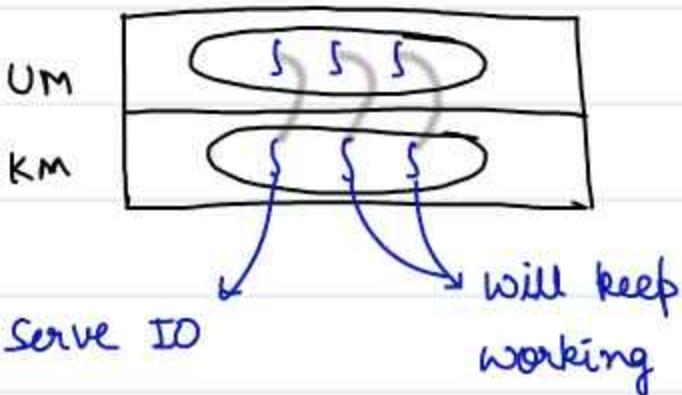
1. O.S. is not able to see those threads, it will see it as a process. ] → good as well as bad.
2. Flexibility ( scheduling, assigning threads etc).
3. Faster Context switching (No need of mode shifting)
  - ↳ Less overhead
4. Portability (can run on diff O.S.)

Drawback :- → Lack of True parallelism even if multiple processors are available.

1. Scheduling & management is the responsibility of user. → Increasing complexity.

2. Blocking Issues :- If one threads need I/O, OS will block the process (all threads)

→ will be solved if O.S. kernel is multithreaded.

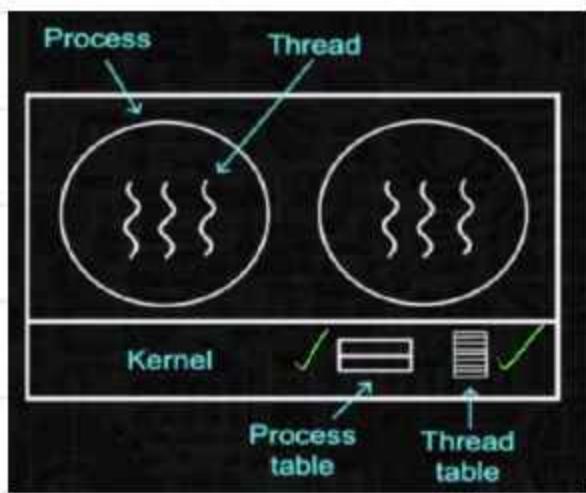


## # Kernel Level #

1. Direct OS mgmt. Kernel level →

2. True parallelism

KTL can be executed on multiple processors parallelly



os is managing the processes & threads

3. Blocking Handling :- Unlike UTL , if one KLT gets blocked , others keep running
4. Managed completely by OS → Less complexity to user.
5. Slower context switching than UTL.

#### Comparison:

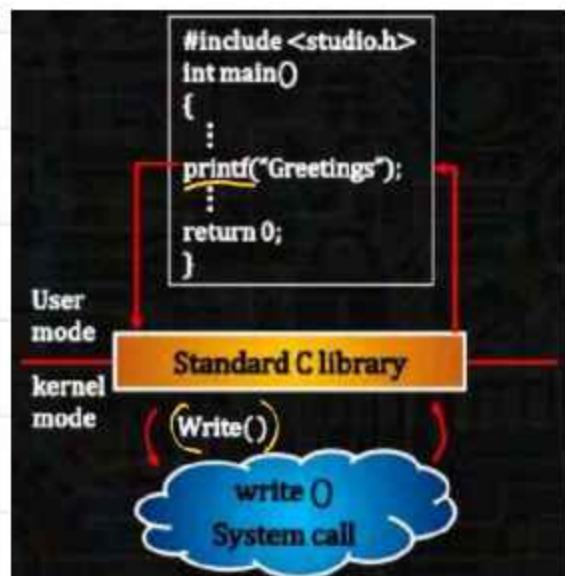
- User-Level Threads: Faster context switching, custom scheduling, less overhead, but blocking issues ~~or~~ no true parallelism.
- Kernel-Level Threads: Better blocking handling, true parallelism, better OS integration, but higher overhead ~~or~~ slower context switching.



# Miscellaneous Topic - 2

## # System call & Fork #

→ How are parameters passed from VM to KM?



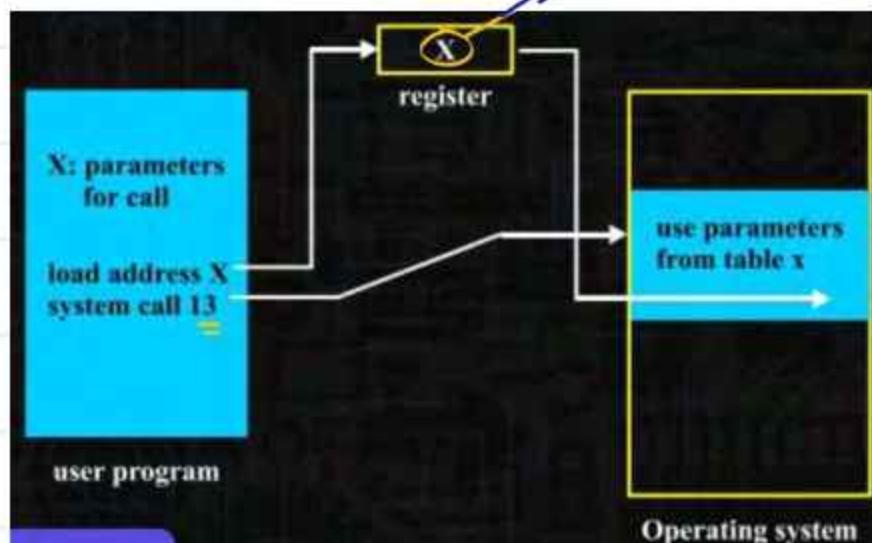
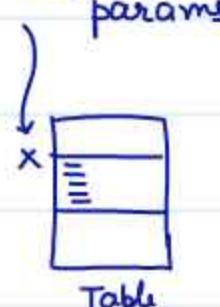
→ Registers  
→ Memory → pass address  
→ Stack

### System Call Parameter Passing

- Often, more information is required than simply identity of desired system call  
Exact type and amount of information vary according to OS and call
- Three general methods used to pass parameters to the OS
  - ❖ Simplest: pass the parameters in registers
    - In some cases, may be more parameters than registers
  - ❖ Parameters stored in a block or table, in memory, and address of block passed as a parameter in a register
    - This approach taken by Linux and Solaris
  - ❖ Parameters placed, or pushed, onto the stack by the program and popped off the stack by the operating system
    - Kernel
- Block and stack methods do not limit the number or length of parameters being

Address of the block of memory

cont.



We should know some basic system calls related to process mgmt, file management & directory mgmt.

## Process Management

2

Call	Description
<code>pid = fork()</code>	Create a child process identical to the parent
<code>pid = waitpid(pid, &amp;statloc, options)</code>	Wait for a child to terminate
<code>s = execve(name, argv, environp)</code>	Replace a process' core image
<code>exit(status)</code>	Terminate process execution and return status

## File Management

Call	Description
<code>fd = open(file, how, ...)</code>	Open a file for reading, writing or both
<code>s = close(fd)</code>	Close an open file
<code>n = read(fd, buffer, nbytes)</code>	Read data from a file into a buffer
<code>n = write(fd, buffer, nbytes)</code>	Write data from a buffer into a file
<code>position = lseek(fd, offset, whence)</code>	Move the file pointer
<code>s = stat(name, &amp;buf)</code>	Get a file's status information

## Some System Calls For Directory Management

Call	Description
<code>s = mkdir(name, mode)</code>	Create a new directory
<code>s = rmdir(name)</code>	Remove an empty directory
<code>s = link(name1, name2)</code>	Create a new entry, name2, pointing to name1
<code>s = unlink(name)</code>	Remove a directory entry
<code>s = mount(special, name, flag)</code>	Mount a file system
<code>s = umount(special)</code>	Unmount a file system

## Miscellaneous

Call	Description
<code>s = chdir(dirname)</code>	Change the working directory
<code>s = chmod(name, mode)</code>	Change a file's protection bits
<code>s = kill(pid, signal)</code>	Send a signal to a process
<code>seconds = time(&amp;seconds)</code>	Get the elapsed time since Jan 1,1970

## # Fork System call #

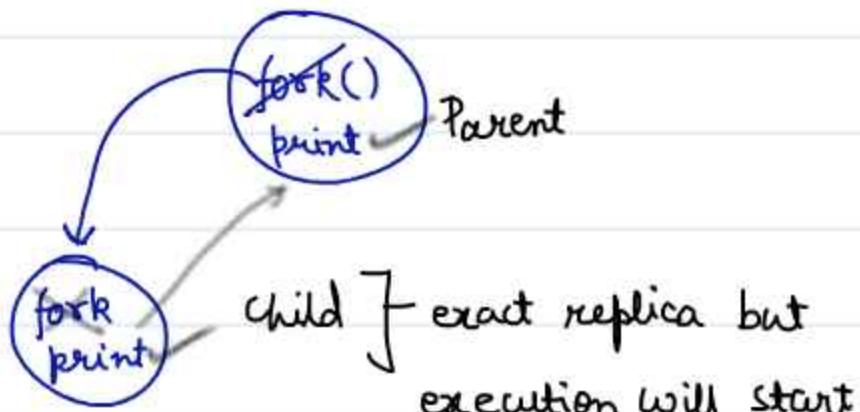
→ creates a child process

→ The code of child will be an exact replica of parent process.

→ Execution in child will start from next stmt after fork till end of program.

seperate area of memory , child & parent are independent , no master-slave relationship

```
① main () {
    fork ();
    printf ("Hello");
}
```



Control flow :-

from next stmt

fork () : Parent

after fork().

printf() : child

→ Hello

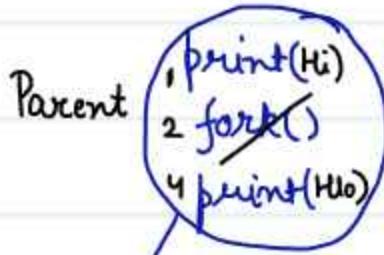
printf() : parent

Hello

] OUTPUT

② main ()

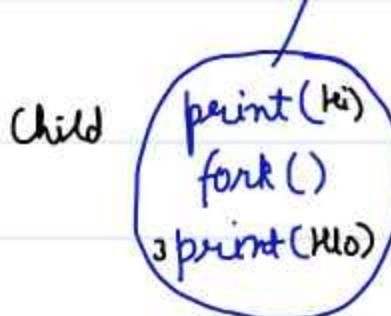
```
{   printf (" Hi "); }
```



```

fork();
print("Hello");
}

```

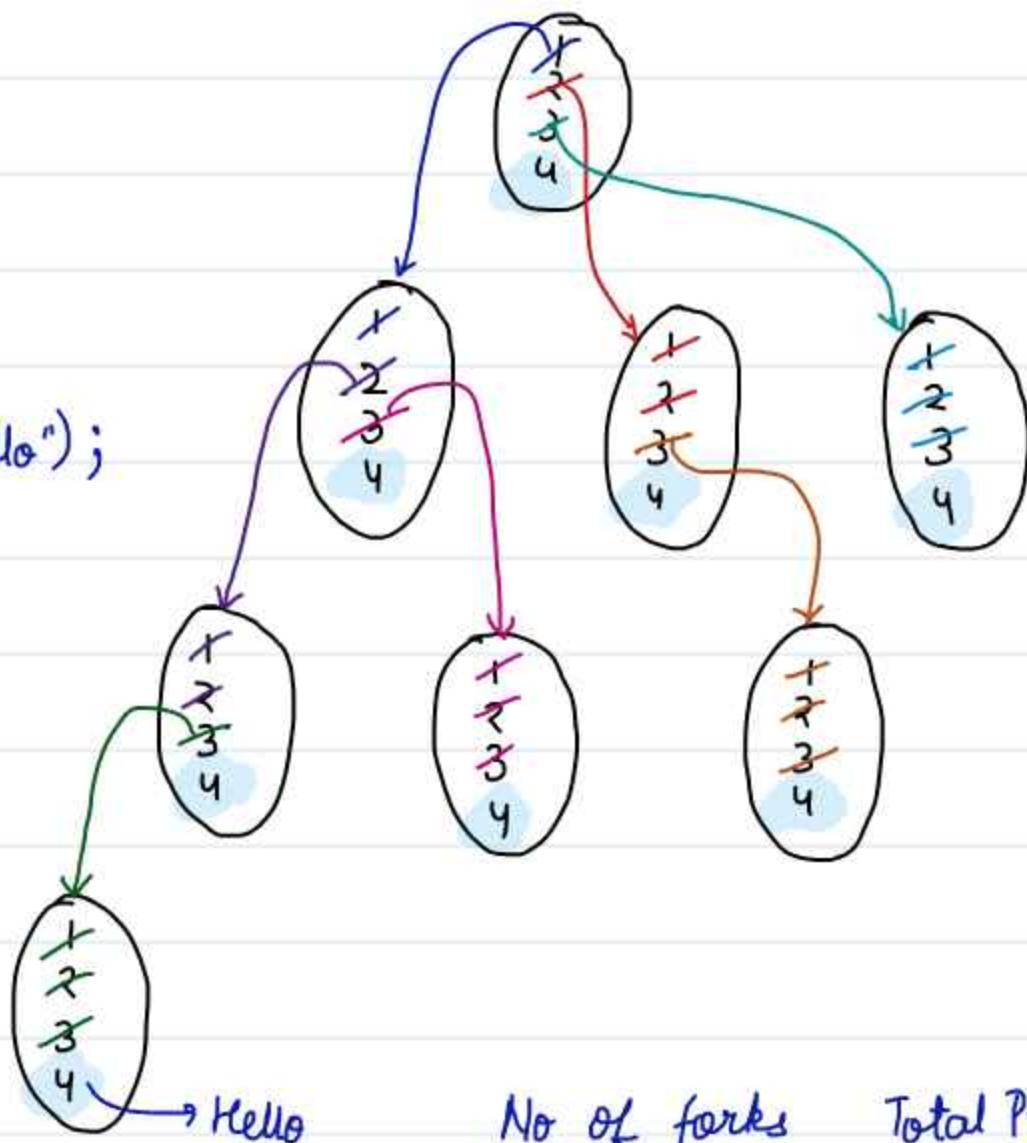


③ main()

```

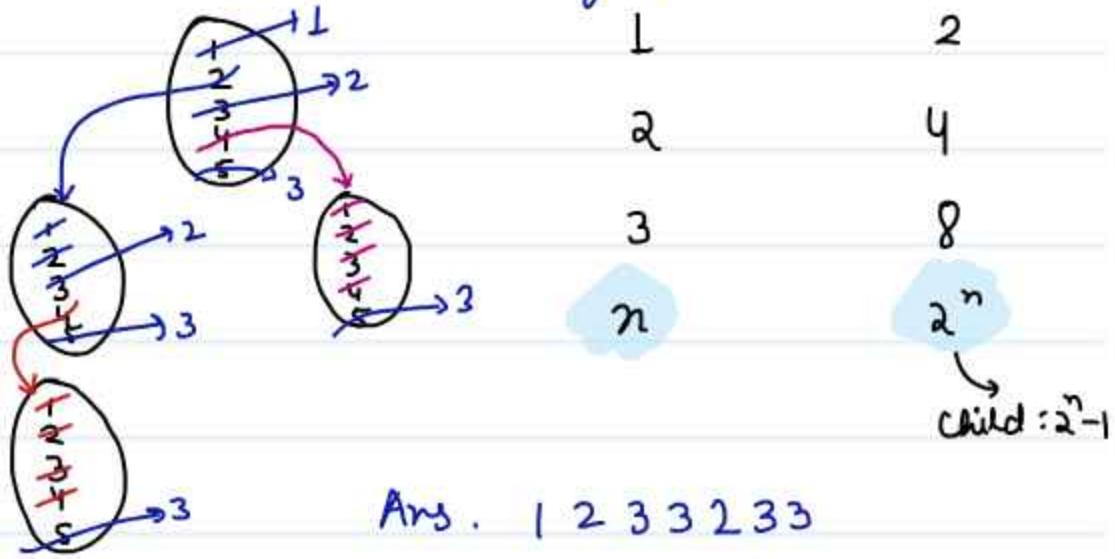
{
1. fork();
2. fork();
3. fork();
4. printf("Hello");
}

```

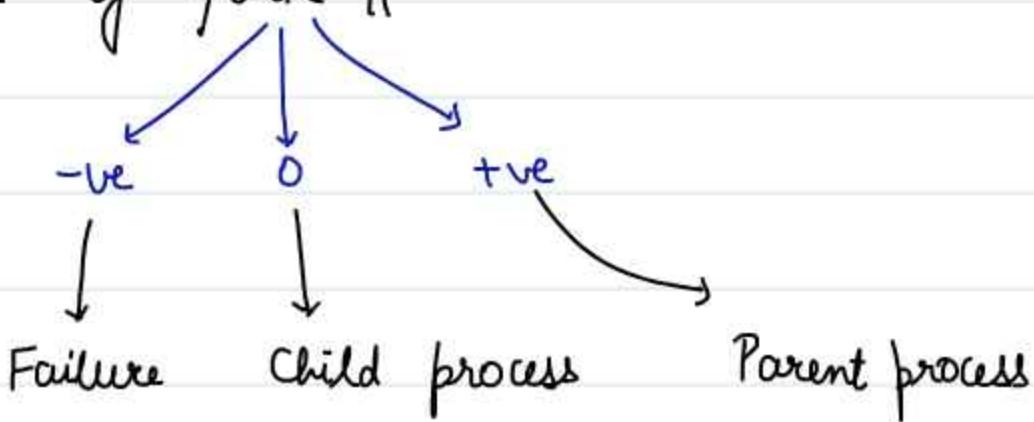


No of forks      Total Process

④ main() {  
1 print("1");  
2 fork();  
3 print("2");  
4 fork();  
5 print("3");



# Return value of fork #



main()

int ret;

ret = fork();

if (ret == 0)

{

// child process code

}

else {

// parent process code

}

== // child + parent

}

Q

main()

{ int a=5, b=3, c ;

c = ++a \* b++;

1 ret  
ret = fork();  
if (ret == 0){  
} else {  
} == ✓

0 ret  
~~ret = fork();~~  
if (ret == 0){  
} else {  
} == ✓

OS

fork routine

pf(a,b,c);

child [ if( fork == 0) { a=a+5;  
b = b+3; pf(ab); c=c-1;} ]

else { b+=a+=2; pf(a,b);  
c=c-1; } ↗ parent

pf(c); } ↗ both

a 86 b 134 c 0518

$$C = 6 \times 3 = 18$$

child :-

print → 6, 4, 18 ①

$$a = 6 + 5 = 11;$$

$$b = 4 + 3 = 7;$$

parent :-

② print → 11, 7

$$b+ = a+ = 2$$

$a = a + 2 \Rightarrow 8$

C = 17

print(a,b) → 8,12 ④

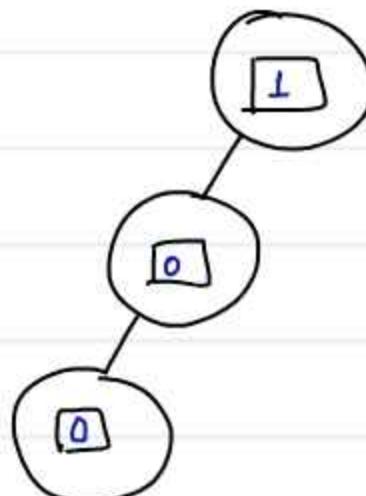
$$C \rightarrow C = 17$$

point(c) → 17 ⑤

Q

```
main()
{
    int i, n;
    for ( i=1 ; i <=n ; ++i )
        if ( fork() ==0 )
            print( "*" );
}
```

Child of child is  
also child to parent



will be executed  $2^n - 1$  times

→ no of child processes

Q

```

int i, n;
for ( i=1 ; i <=n ; ++i) → n
{
    if ( i % 2 == 0 )
        fork(); →  $\frac{n}{2}$ 
}

```

$$2^{\frac{n}{2}} - 1$$

Q

```

main()
{
    int i, n;
    for ( i=1 ; i <=n ; ++i) →  $2^n - 1$  child process
        if ( fork() == 0 );
            print("*"); } not in for loop
    }

```

$\downarrow$

$2^n - 1$  times

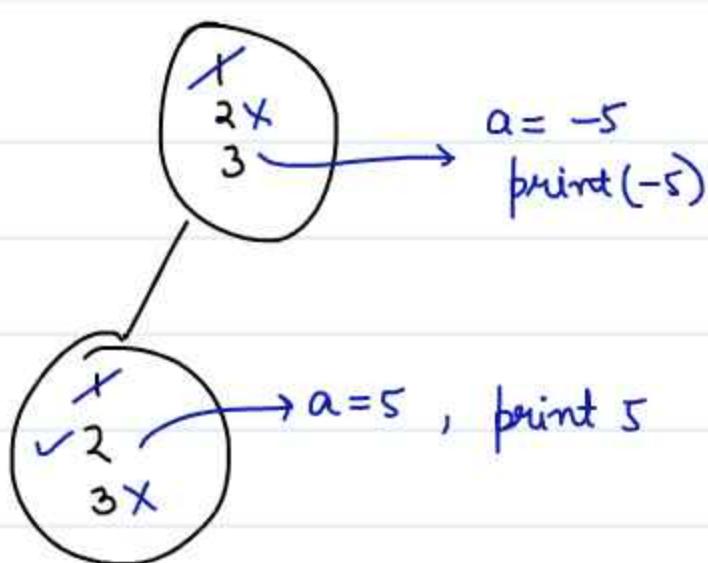
↓ time by parent also

$$\text{Total} := 2^n - 1 + 1 = \underline{2^n}$$

Q main()

1. int a;	3. else	What's the
2. if ( fork() == 0 )	{ a = a-5;	relation b/w u
{ a = a+5;	x: print(a);	& z.
u: print(a); }	}	

Let's take  $a = 0$  initially



$$u = 5$$

$$x = -5$$

$$u = x + 10$$

of variable in child & parent.

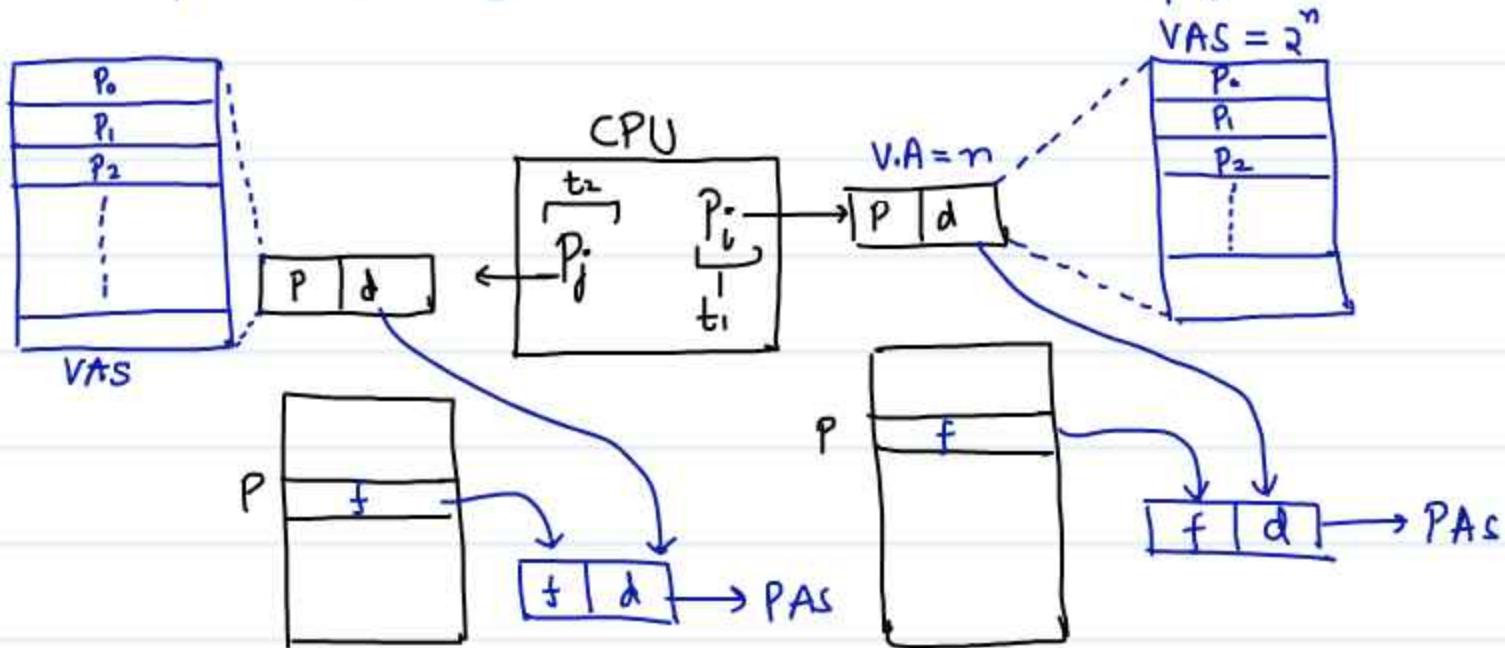
NOTE :- Physical addresses will be different, virtual addresses will be same.

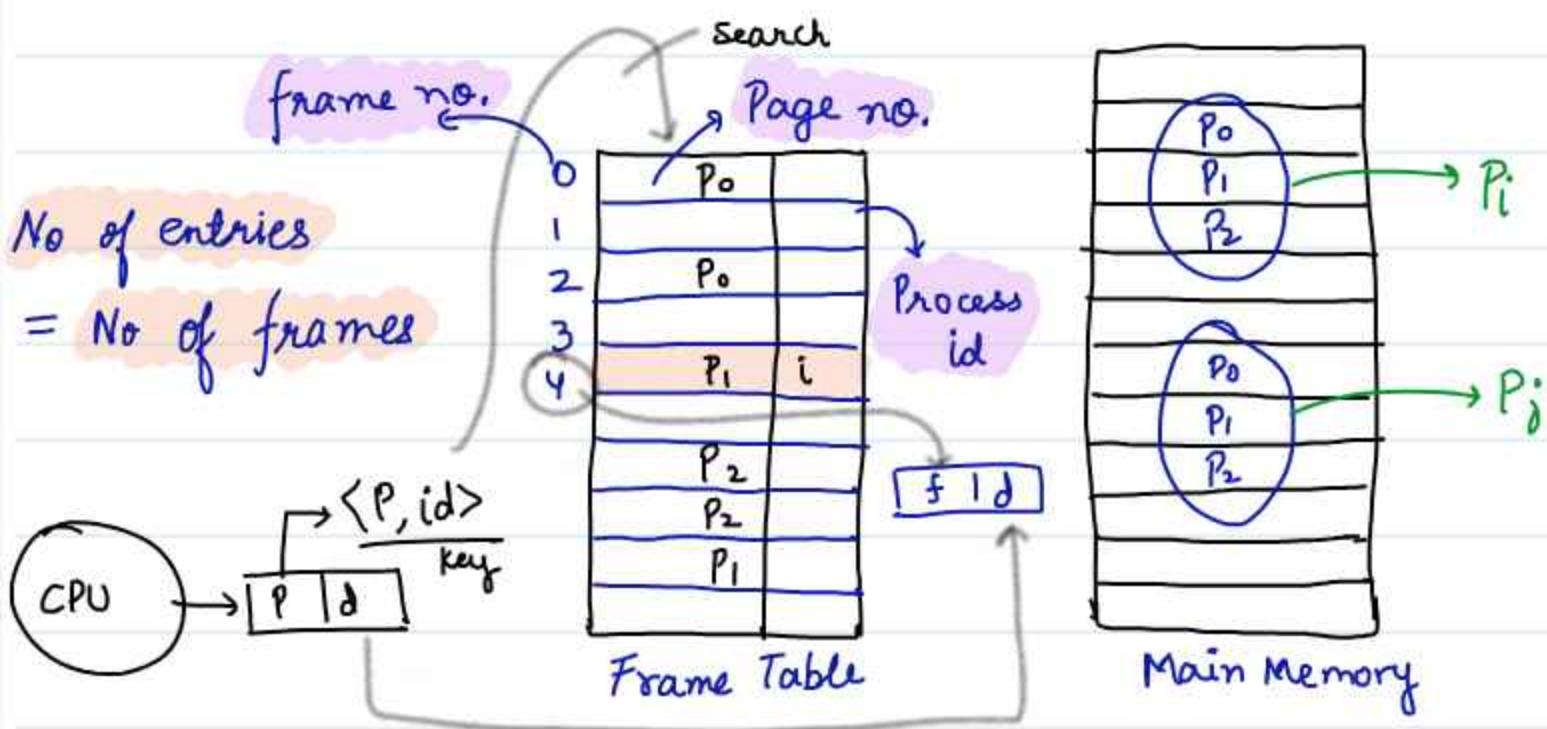
on printing we get V.A.

## # Inverted Page Table < Reverse Paging > #

→ Reduce space overhead of P.T. in Paging

Each process will have its own VAS & page table.





Pages of all process are being mapped on same PAS.  
Can't we design a page table based on PAS?

Some kind of global P.T.

In inverted paging one global page table is used by all the process. In traditional approach 100 processes will have 100 page tables.

To save space, sacrifice Time!

search:  $O(n)$

Traditional :  $O(1)$

Q

$$V.A = 34 \text{ bits}$$

$$P.A = 23 \text{ bits}$$

Size of PT

$$P.S. = 8 \text{ KB}$$

$$P.T.E = 32 \text{ bits}$$

- a) Trad b) Inv.

a)  $VAS = 2^{34} B$  No of pages = No of entries  
 $P.S. = 8 \text{ KB}$

$$= \frac{2^{34}}{2^{13}} = 2^{21}$$

Table size = No of entries  $\times e = 2^2 \times 2^{21} = 8 \text{ MB}$   
 per process

b) No of entries = No of frames  $\frac{2^{23}}{2^{13}} = 2^{10}$

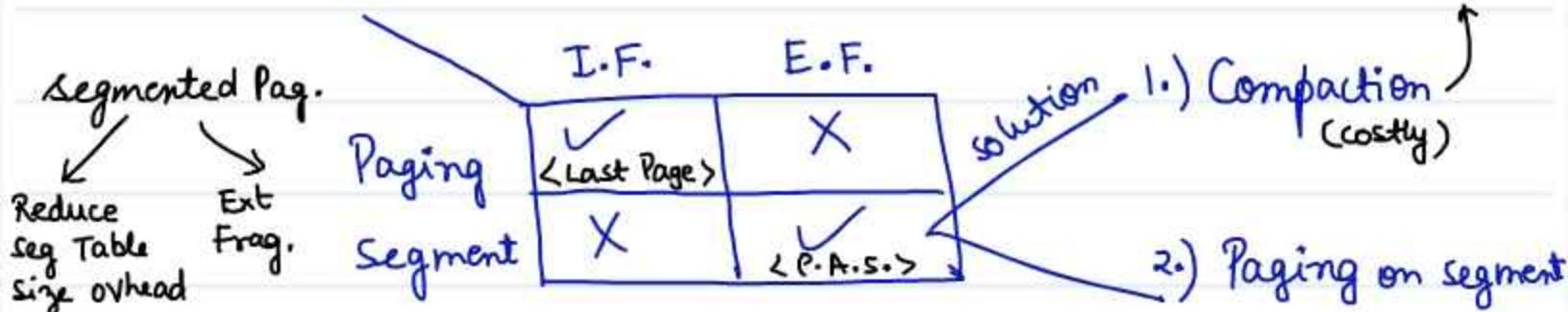
$$\text{Table size} = 2^{10} \times 2^2 B = 2^{18} B = 256 \text{ KB}$$

Let say there are 100 processes

- a) Trad = 800 MB
- b) Inv = 256 KB

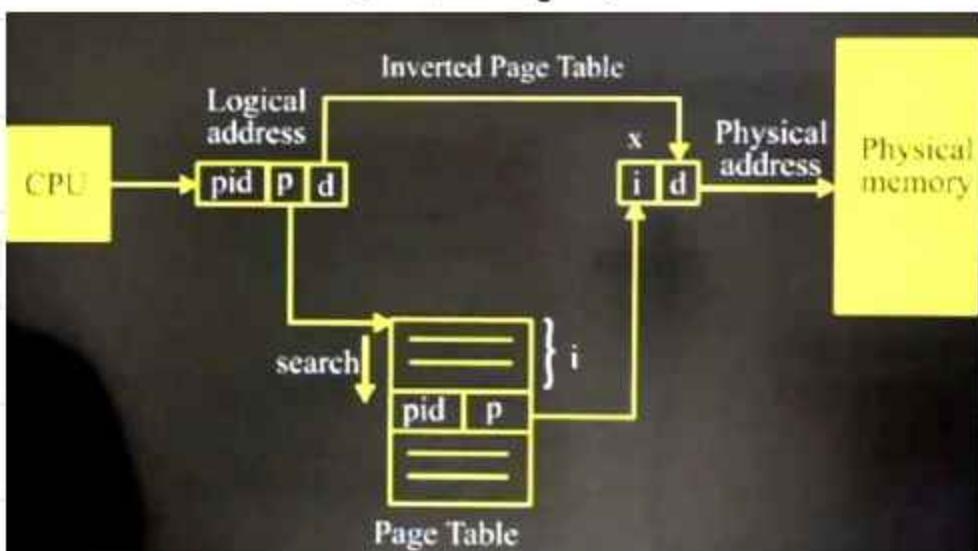
## # Paging Vs Segmentation #

Requires run time address binding



# Miscellaneous Topic - 3

## Inverted Page Table



- Uses global page table.
- Amazing space optimization

Which one of the following statements is FALSE?

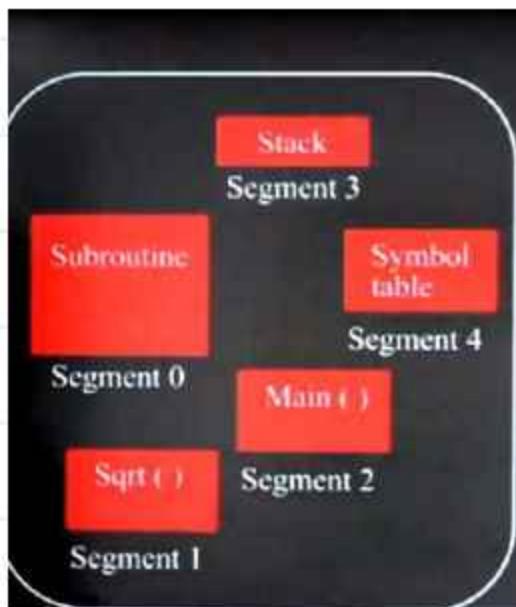
- A. The TLB performs an associative search in parallel on all its valid entries using page number of incoming virtual address.
- B. If the virtual address of a word given by CPU has a TLB hit, but the subsequent search for the word results in a cache miss, then the word will always be present in the main memory.
- C. The memory access time using a given inverted page table is always same for all incoming virtual addresses.
- D. In a system that uses hashed page tables, if two distinct virtual addresses V1 and V2 map to the same value while hashing, then the memory access time of these addresses will not be the same.

solution

- A. The TLB performs an associative search in parallel on all its valid entries using the page number of the incoming virtual address.
- This statement is true. The Translation Lookaside Buffer (TLB) indeed performs an associative search, meaning it compares the incoming virtual address with all entries simultaneously to find a match quickly.
- B. If the virtual address of a word given by CPU has a TLB hit, but the subsequent search for the word results in a cache miss, then the word will always be present in the main memory.
- This statement is true. If there is a TLB hit, the virtual address is correctly translated to a physical address. Even if there is a cache miss, meaning the data is not in the CPU's cache, the data can still be found in the main memory.
- C. The memory access time using a given inverted page table is always the same for all incoming virtual addresses.
- This statement is false. In an inverted page table, the time to access memory can vary because the system may need to search through multiple entries to find the correct physical address corresponding to the given virtual address. This is unlike traditional page tables where the virtual address directly indexes into the table.
- D. In a system that uses hashed page tables, if two distinct virtual addresses V1 and V2 map to the same value while hashing, then the memory access time of these addresses will not be the same.
- This statement is true. When hashed page tables are used, different virtual addresses might hash to the same value, causing a collision. The system would then need additional steps to resolve the collision, potentially leading to different access times for these addresses.

# # Segmented Paging Architecture #

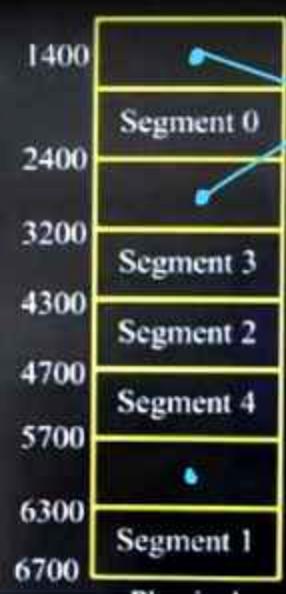
→ overcome the problem of External frag.  
 Associate a smaller translation table with process by applying paging on segment table



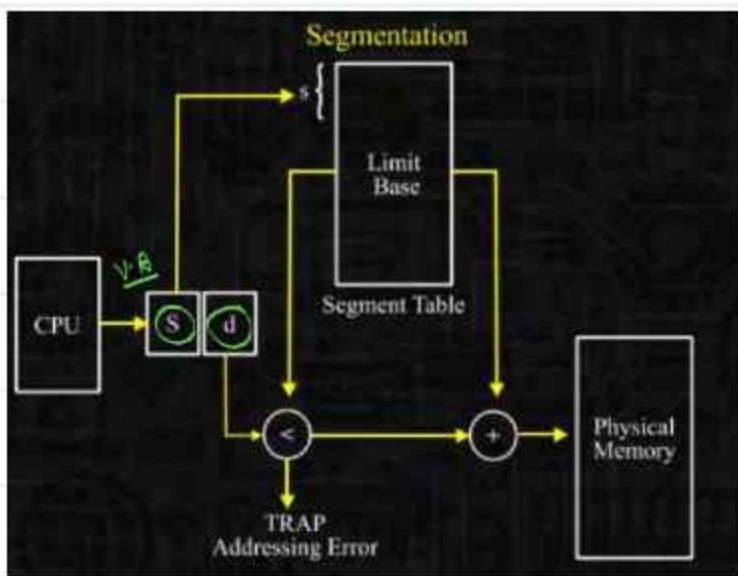
Segmentation

	Limit	Base
0	1000	1400
1	400	6300
2	400	4300
3	1100	3200
4	1000	4700

Segment Table



Physical mem.



d value < offset  
 Yes  
 Add to Base  
 ↓  
 P.A.

Trap

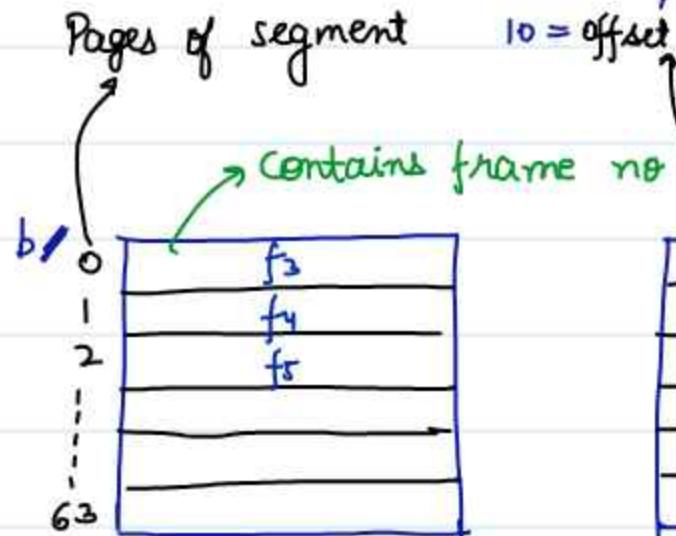
# # Paging on Segment #

$$\langle s, d \rangle = \langle 18, 16 \rangle$$

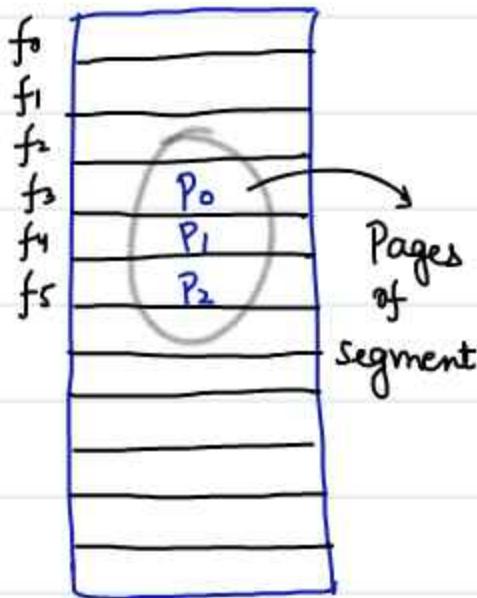
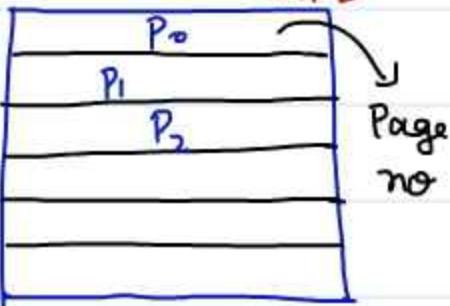
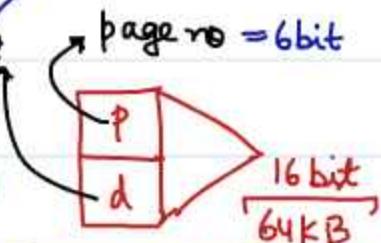
$$LA = 34 \text{ bit}$$

$$\text{Max segment size} = 2^{16} = 64 \text{ KB}$$

Let page size of segment =  $1 \text{ KB}$  = Frame size of MM



$10 = \text{offset}$

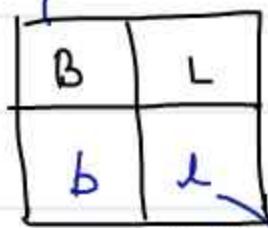
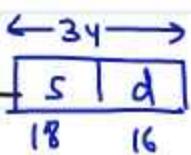
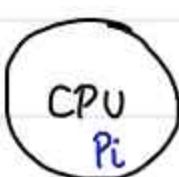


Page Table

of segment

Same no  
of entries

Segment



seg-table

Now seg is no more stored in memory as it is.

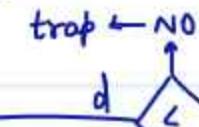
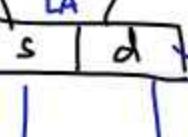
$B$  = Add of

Page Table of  
segmentation

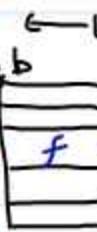
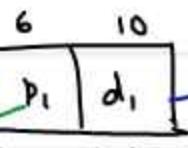
length of segment

entries in seg table =  $2^{18}$

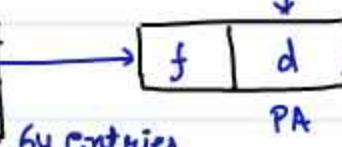
$2^{16}$  = segment size



Yes



64 entries



$\nearrow$  PAs

Entries in seg. table =  $2^{18}$ ] - Large 😒

To reduce this Apply Paging on ST.

- ⊕ 256 K entries segment table will be divided into pages of size = 1 KB
- ⊕ Total 3 tables in architecture :-
  - a) First → segment table
  - b) Page Table 1 → To access pages of segment table
  - c) Page Table 2 → To access pages of segment

Virtual address = segment no + Page no + offset

Program divided → segment table → PT with → MM  
into segments with PT addresses frame addresses



## Summary :-

- a) Paging is used for overcoming Ext. Fr.
- b) To remove Ext. Fr. from segmentation
  - ↓
  - Apply paging on segment
- c) Divide each segment into pages, store pages in MM & used P.T. to access those pages.
- d) Each segment will have its own page table & addresses of these page table will be in the segment table.
- e) If the segment table becomes large then apply paging on segment table.
- f) To access these pages of segment table we have another table.

Q Consider a System using Segmented-Paging Architecture with V.A.S = P.A.S =  $2^{16}$  Bytes. The VAS is divided into 8 equal sized nonoverlapping Segments. Paging is applied on Segment, with a Page size being a power of 2 in Bytes. The Page Table Entry size of the Page Tables of the Segment is 16 bits. What must be the Page Size of the Segment such that the Page Table of the Segment exactly fits in one frame of Memory.

$$V.A.S = P.A.S = 2^{16} \text{ Bytes}$$

$$\text{divided into 8 segments} = 2^{16}/2^3 = 2^{13} \text{ B}$$

$$P.S = 2^x \text{ B}$$

$$\text{Page table entry size} = 2 \text{ B}$$

$$\text{Page table of segment} = \text{Frame size}$$

$$\text{No of entries} \times \text{Entry size} = 2^B$$

$$2^{14-x} = 2^x$$

$$\text{No of segments}$$

$$14-x = x$$

$$x = 7$$

Q Consider a System using Segmented-Paging Architecture. Paging is applied on Segment. The System maintains a 256 entry Page-Table per Segment. The Page Size of Segment is 8Kbytes. The Virtual Address Space supports 2K Segments. Page Table Entry Size is 16 bits while the Segment Table entry is 32 bits in size.

Calculate

- Size of Virtual Address Space
- Address Translation Space Overhead in Bytes.
- The number of levels of Memory accesses required for Address Translation.

No of pages = 256

Page size

= 8KB

Seg size

=  $256 \times 8 \text{ KB}$

$$\text{seg size} = 2^8 \times 2^3 \text{ KB} = 2^{11} \text{ KB} = \underline{2 \text{ MB}}$$

$$\text{No of segments} = 2^K$$

$$\text{Page Table entry} = 2B$$

$$\text{Segment table entry} = 4B$$

a) VAS = No of seg  $\times$  seg size

$$2^K \times 2 \text{ MB} = 4 \text{ GB}$$

No of segments  
↑

b) Address Translation space overhead = Seg. Table size

+

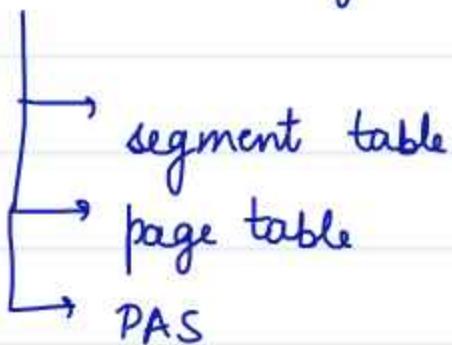
$$2^K \times 4B + 256 \times 2B$$

Page Table size  
↓

$$2 \times 2^2 \times 2^{10} B + 256 \times 2B \quad \text{No of entries} = \frac{\text{segment size}}{\text{Page size}}$$

$$8 \text{ KB} + 512 \text{ B} = \underline{8.5 \text{ KB}}$$

c) Three levels of memory access



To avoid separate copy of pages.

# Shared Pages # → To optimize memory.

- Many application need compilers / editors

### Shared code

One copy of read-only (reentrant) code shared among processes (i.e., text editors, compilers, window systems).

Shared code must appear in same location in the logical address space of all processes

→ Just like  
the concept  
of threads.

### Private code and data

Each process keeps a separate copy of the code and data

The pages for the private code and data can appear anywhere in the logical address space

Simple idea : Instead of having separate copies of same pages, applications can share them.

VAS
ed1
ed2
ed3
data1

Process - 1

3
4
6
1

Process - 1

VAS
ed1
ed2
ed3
data2

Process - 2

VAS
ed1
ed2
ed3
data3

Process - 3

PAS
0
1
2
3
4
5
6
7
8
9

PAS

data1
data2
ed1
ed2
ed3
data2

### # Monitors < synchronization - Mechanism >

Collection of procedures, Variables & other D.S. that are all grouped together in a special kind of

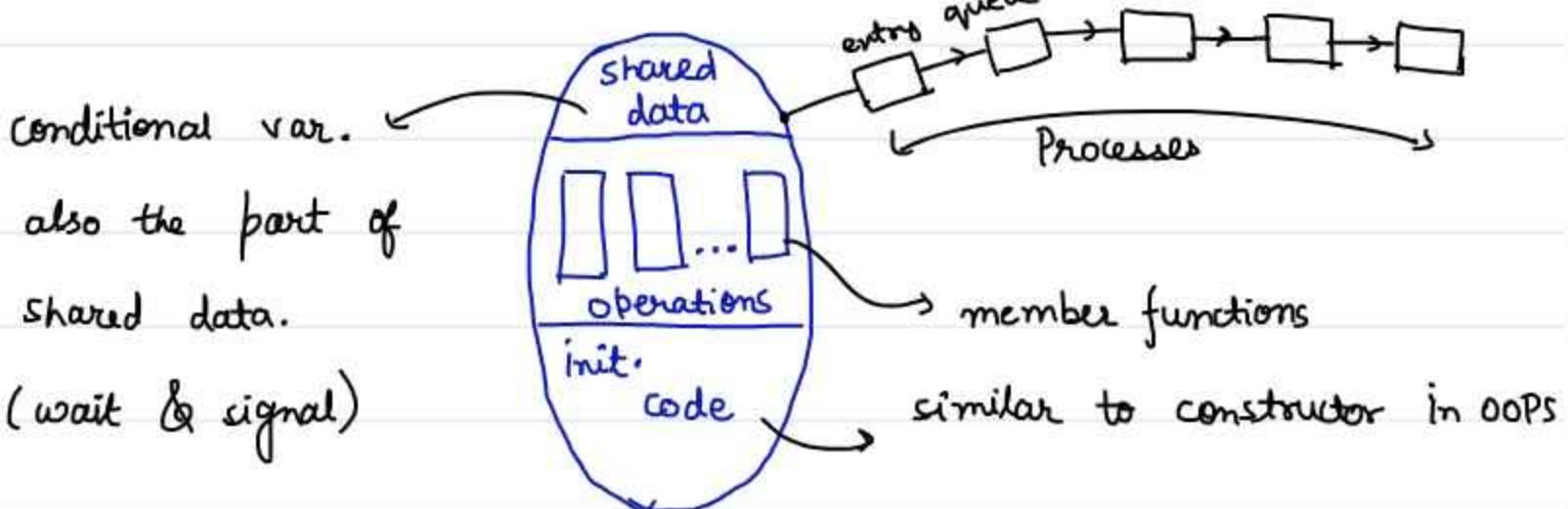
module / package.

Like a class in OOPS.

Procedures that run outside the monitor can't access monitors internal variables and data structures. They can only activate monitor member functions.

Monitors also include special condition variables with wait & signal operation.

only one process can be active in monitor at a time.



Monitor monitor\_name

```
{
    // Shared variable declaration
    function P1(-, -, -) { ..... }
    function P2(-, -, -) { ..... }
    function Pn(-, -, -) { --- }
```

} similar to class

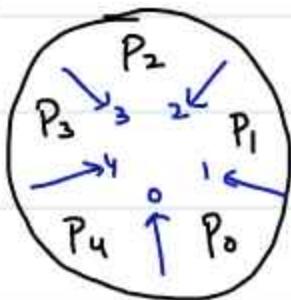
at last

```
initialization code (...) {
    !
}
```

Deadlock free solution

Ex:- Dining philosopher Implementation :-

Monitor DiningPhilosopher {



enum { Thinking, Hungry, Eating } state [5];

condition self [5]; // philosopher successfully taking fork?

Void pickup (int i)

{ state [i] = Hungry;

test (i); // can I really eat , is my LN & RN  
if ( state (i) != Eating ) aren't eating ?

}

self (i). wait();  
→ If I am unsuccessful in  
blocked. taking fork

Void Putdown (int i){

state [i] = Thinking; → change state to thinking

test ((i+4)%5); → will wake up left neighbour

test ((i+4)%5); → " " Right N.

}

Void test (int i){

if (((state (i+4)%5) != Eating) &&

(state (i) == Hungry) && ((state (i+1)%5) != Eating))

L

R

```
{ state[i] = Eating
  self(i) = Signal(); }
```

```
initialization_code() {
    for (int i=0; i<5; i++)
        state[i] = Thinking; }
```

## Dining philosophers dp;

- The monitor `DiningPhilosophers` encapsulates the states and synchronization for the philosophers.
- Each philosopher can be in one of three states: THINKING, HUNGRY, or EATING.
- The `pickup` method is used to attempt to start eating.
- The `putdown` method is used to stop eating and let others eat.
- The `test` method checks if a philosopher can start eating and signals them if they can.
- The initialization code sets all philosophers to the THINKING state initially.
- The instance `dp` is used to work with the monitor.

## condition

cpp

Copy code

```
condition self[5];
```

This declares an array of condition variables, `self`, one for each philosopher. These condition variables will be used to make philosophers wait and be signaled when they can start eating.

## Putup

This method is called by philosopher `i` when they want to pick up the chopsticks and start eating.

- `state[i] = HUNGRY;` sets the state of philosopher `i` to `HUNGRY`.
- `test(i);` checks if philosopher `i` can start eating.
- `if (state[i] != EATING) { self[i].wait(); }` makes philosopher `i` wait if they are not in the `EATING` state after the test.

## Putdown

This method is called by philosopher `i` when they are done eating and want to put down the chopsticks.

- `state[i] = THINKING;` sets the state of philosopher `i` to `THINKING`.
- `test((i + 4) % 5);` checks if the left neighbor of philosopher `i` can start eating.
- `test((i + 1) % 5);` checks if the right neighbor of philosopher `i` can start eating.

## Test

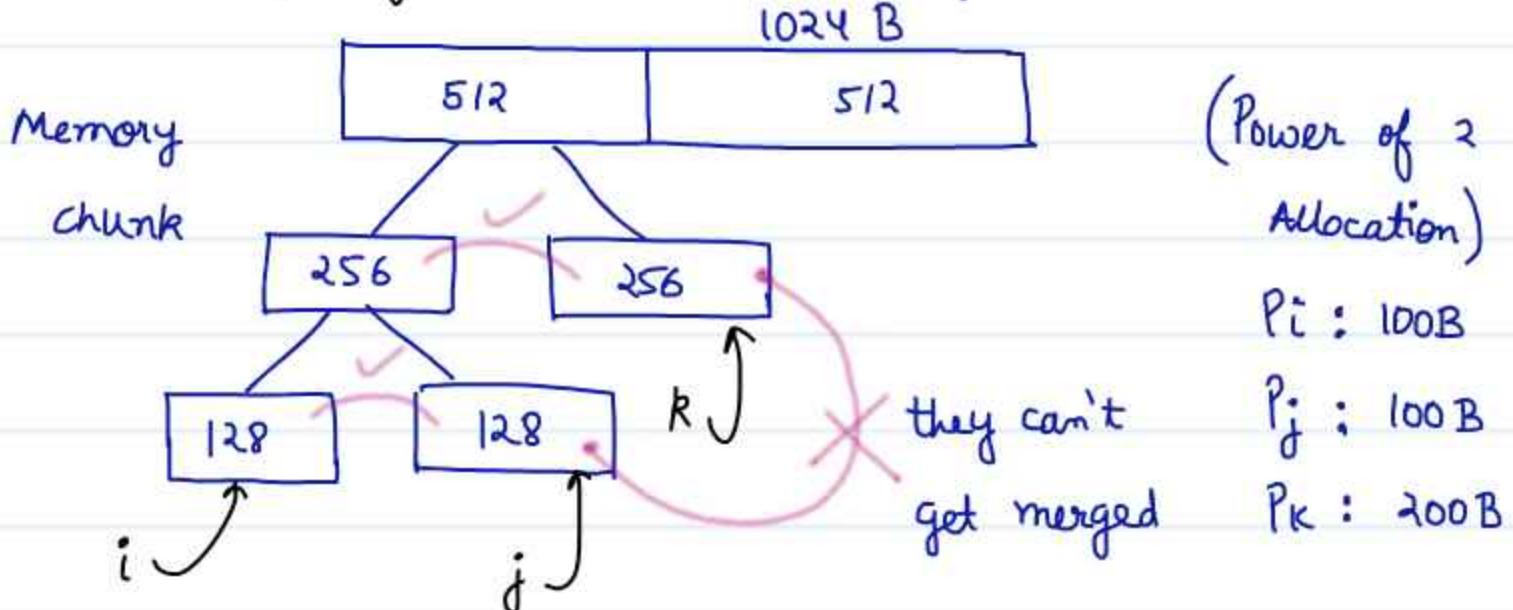
This method checks if philosopher `i` can start eating.

- `if (state[(i + 4) % 5] != EATING && state[i] == HUNGRY && state[(i + 1) % 5] != EATING)` checks if the left and right neighbors of philosopher `i` are not eating and philosopher `i` is hungry.
- `state[i] = EATING;` sets the state of philosopher `i` to `EATING`.
- `self[i].signal();` wakes up philosopher `i` if they were waiting.

## Initialization code .

- `for(int i=0;i<5;i++) state[i] = THINKING;` initializes the state of all philosophers to `THINKING`.

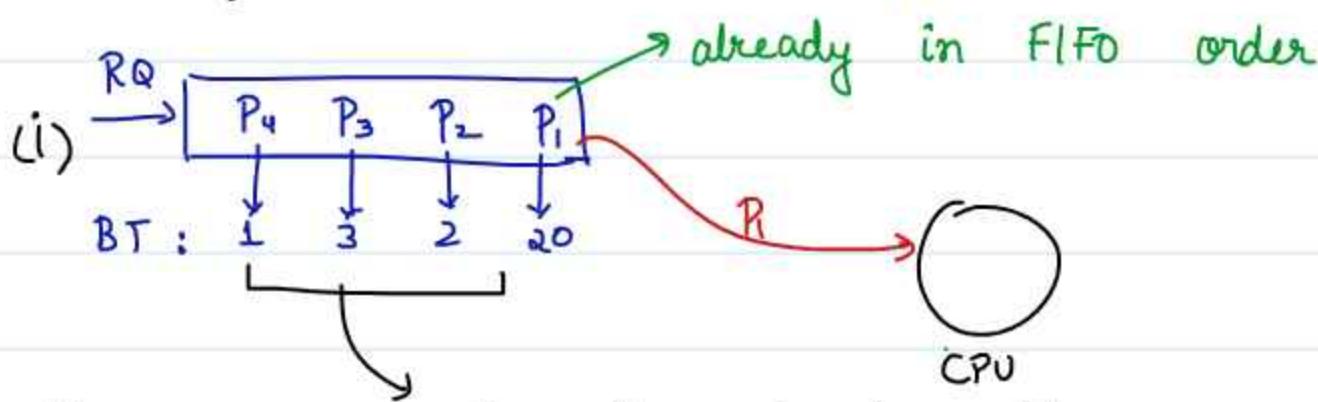
# # Buddy System [Variation of variable partition] #



In variable part. we are creating part. of exactly 100B, here we will create in power of 2  $\Rightarrow$  smallest buddy that can accommodate 100B.

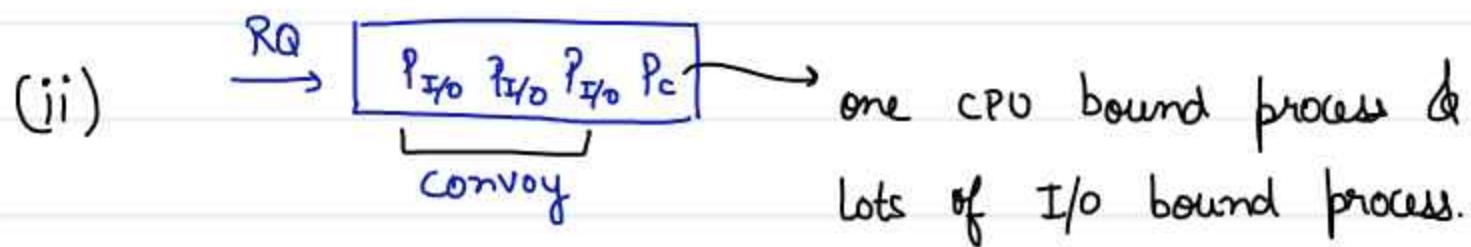
Two adjacent buddies can get merged. } at same level

### # Convoy Effect (FCFS scheduling) #



this convoy will be waiting for long time

when a long process occupies CPU causing waiting for other short process.



Convoy has to wait for  $P_C$  to complete, when  $P_C$  complete convoy uses very little CPU goes to I/O complete the work goes back to RQ finds that  $P_C$  is again with CPU-II<sup>nd</sup> time  $\hookrightarrow$  CPU & I/O can't be used effectively.