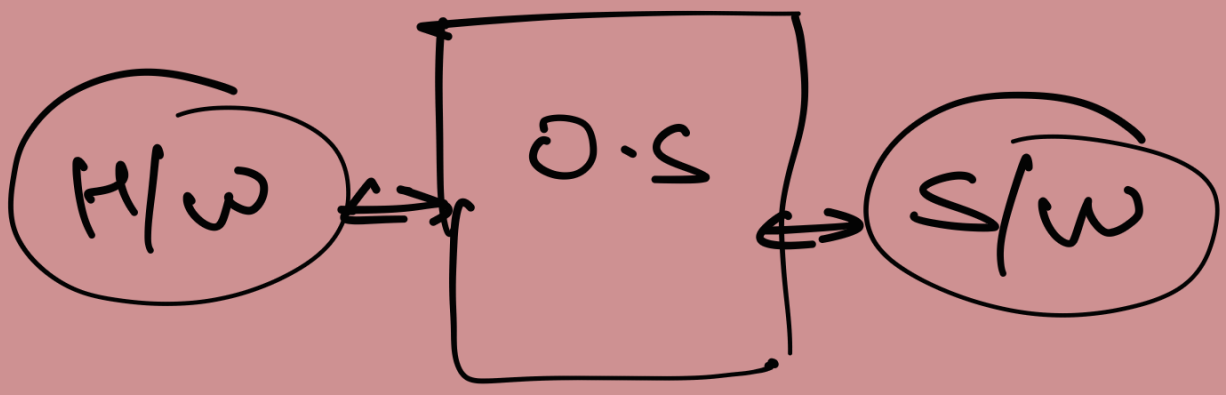
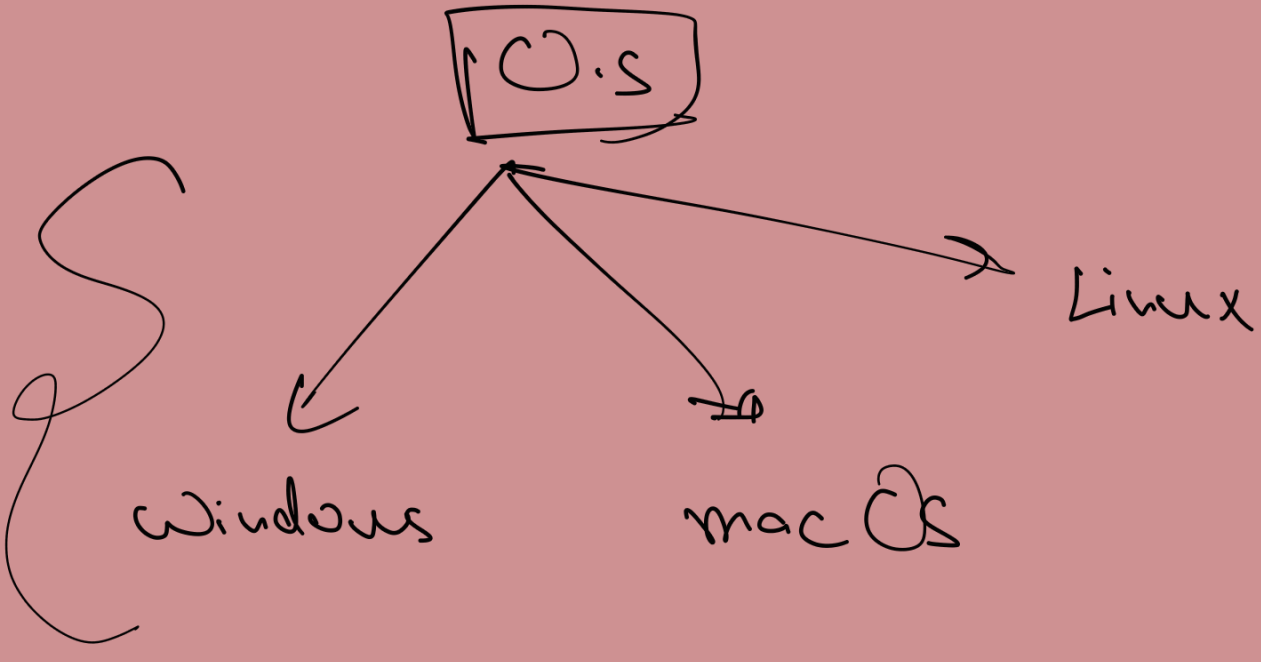


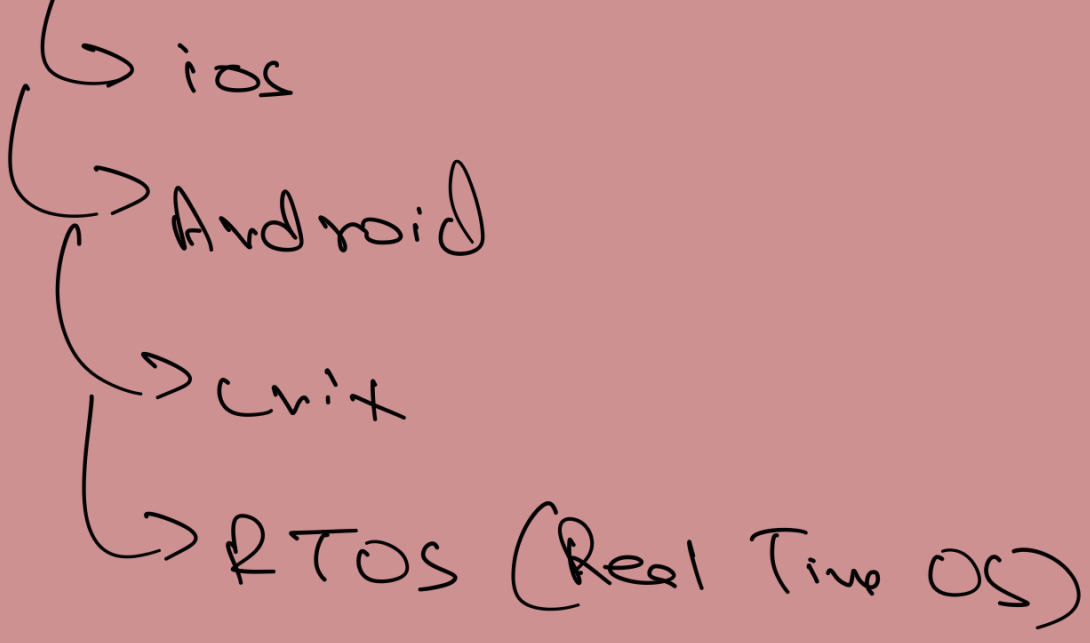
O.S: ① An Interface b/w Computer H/w & user application

② Manges Resources



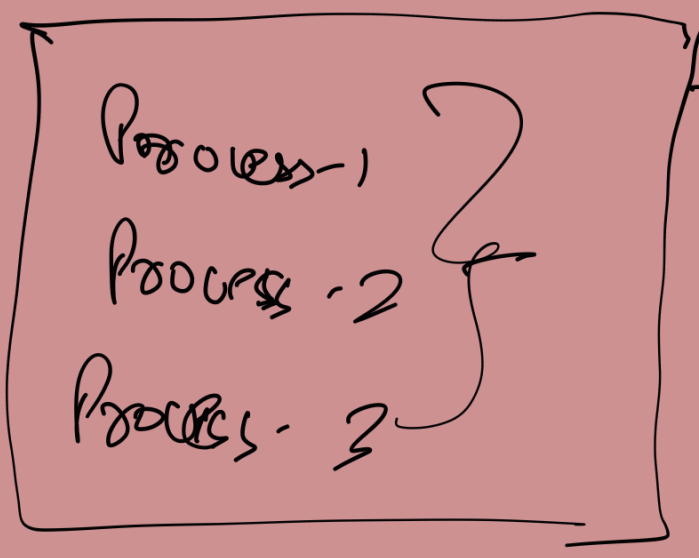
Widely Popular OS ↑

Other than that :-

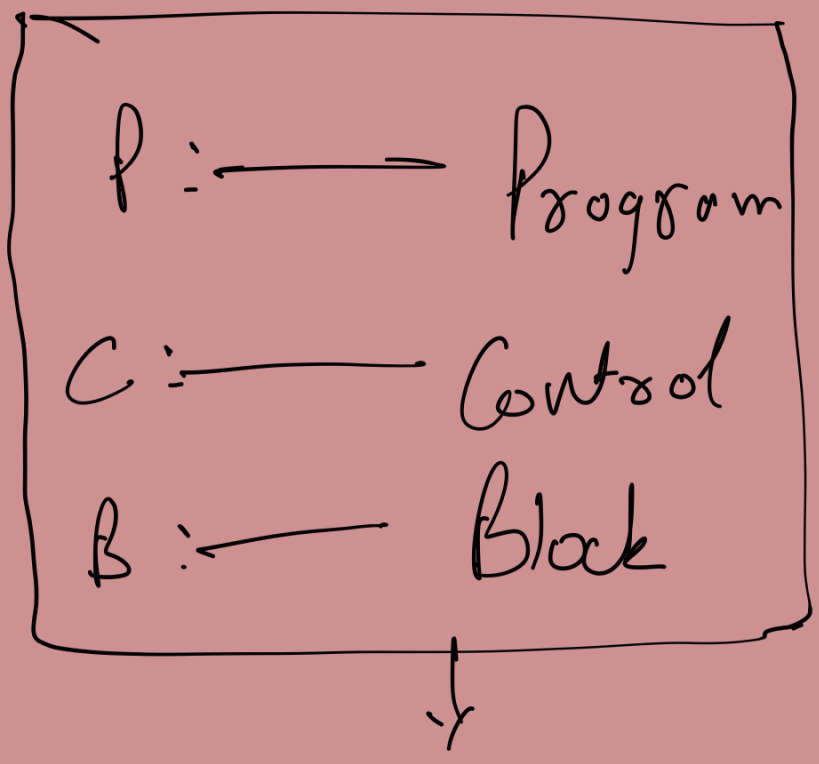


Program :- Set of instructions written in any Programming Language to perform any specific Task.

Process :- It is an instance of Program in execution.



=> Each process has its own PCB



a block that contains info related to that program like:-

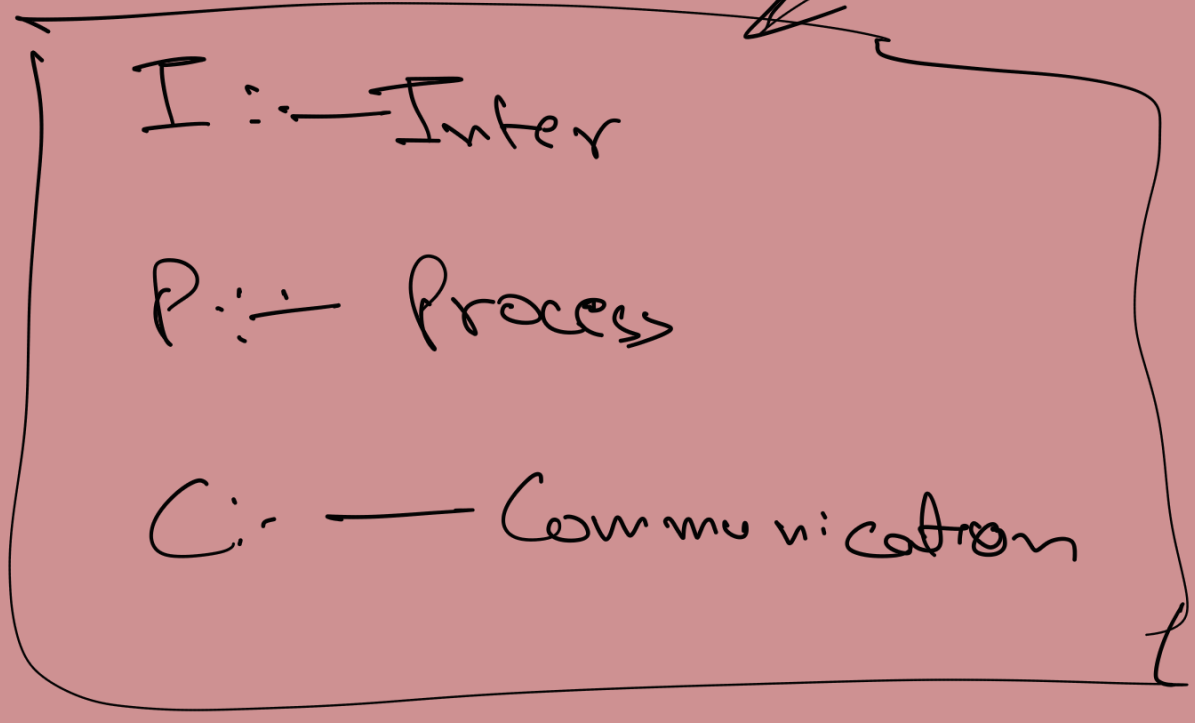
=> amount of M/M

-> Register

=> Program counter .... etc.

NOTE

:) 2 Process need to talk to each other, they use IPC

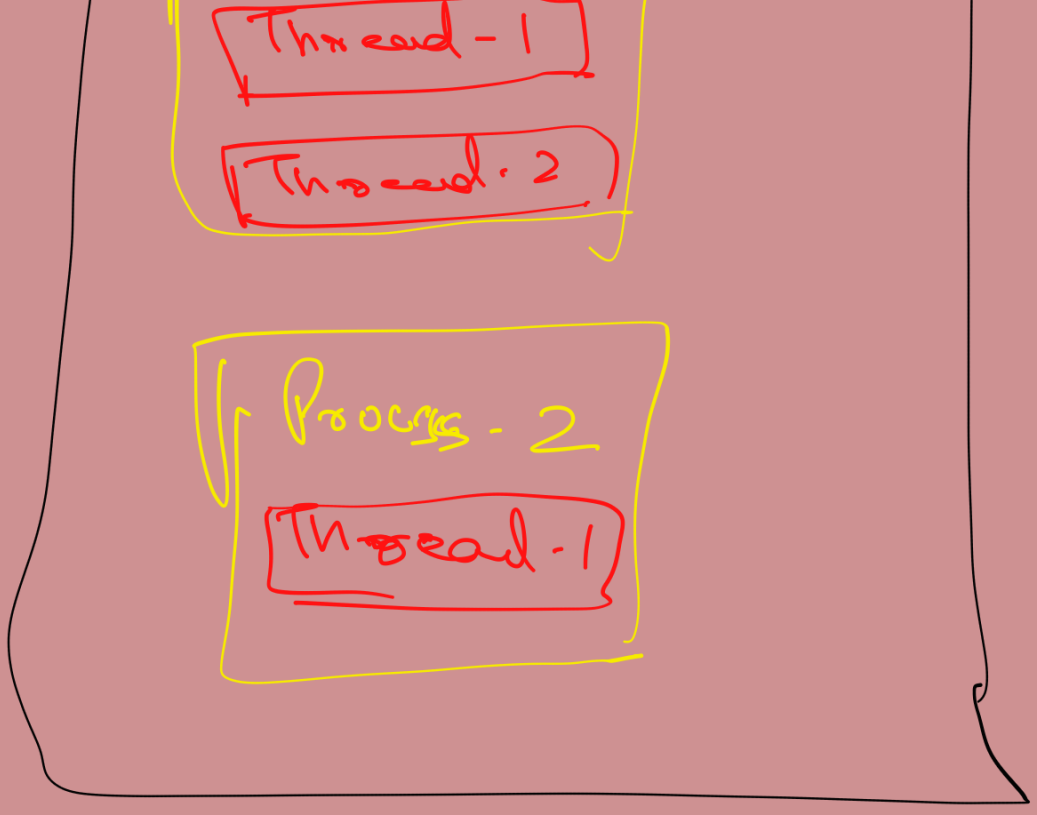


Thread :— Unit of execution with in a process.

→ Seq. of Instructions that can be executed Independently.

→ Share same M/M & resource with process

→ A process can have multiple thread.



MULTIPROCESS :-

When multiple processes are running / executing on a system.

→ benifit

→ Multiple task can be

done simultaneously.

Multi-threading :-

- Execution of Multiple thread within a single process.

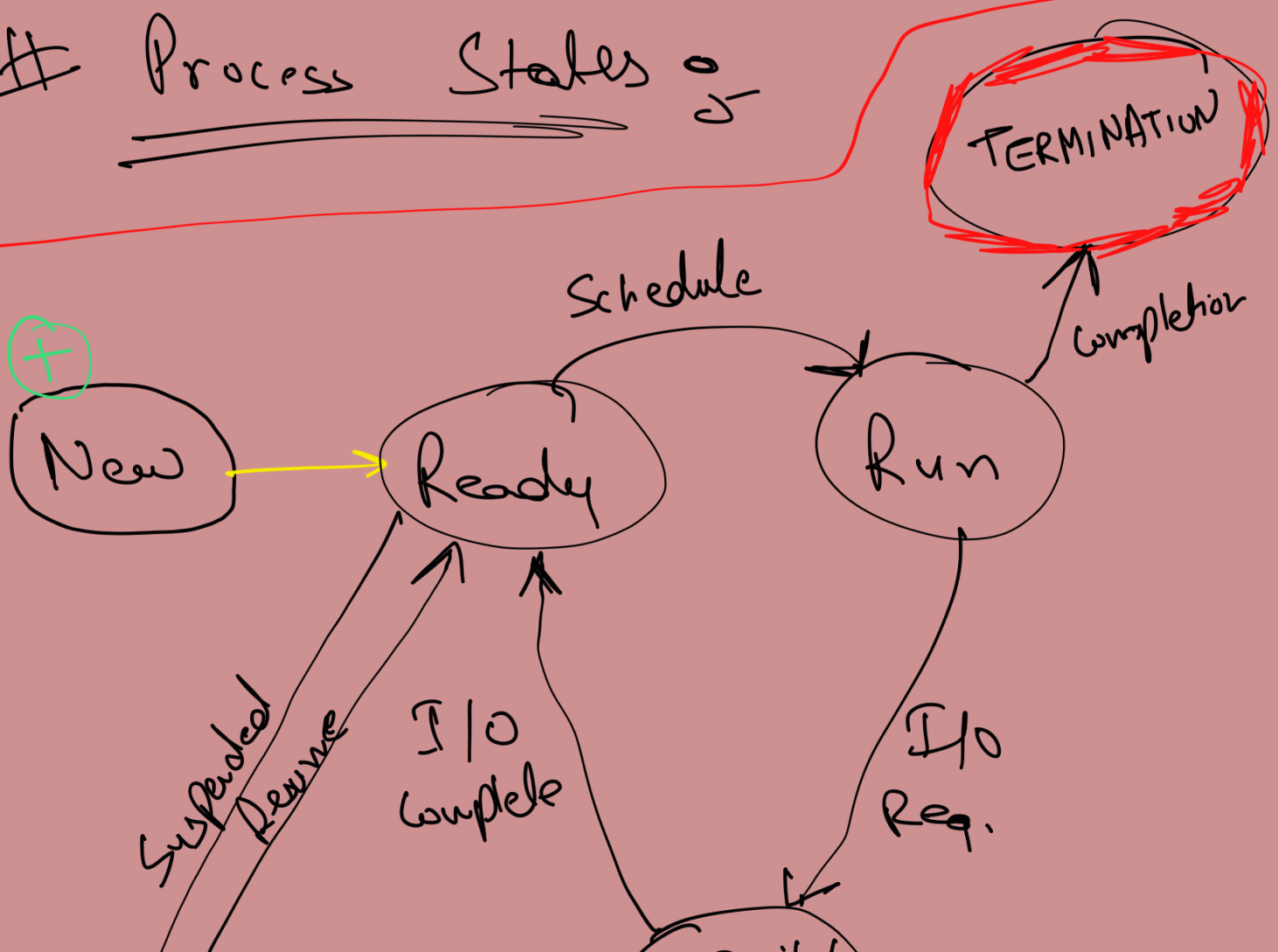
Multi-Programming :-

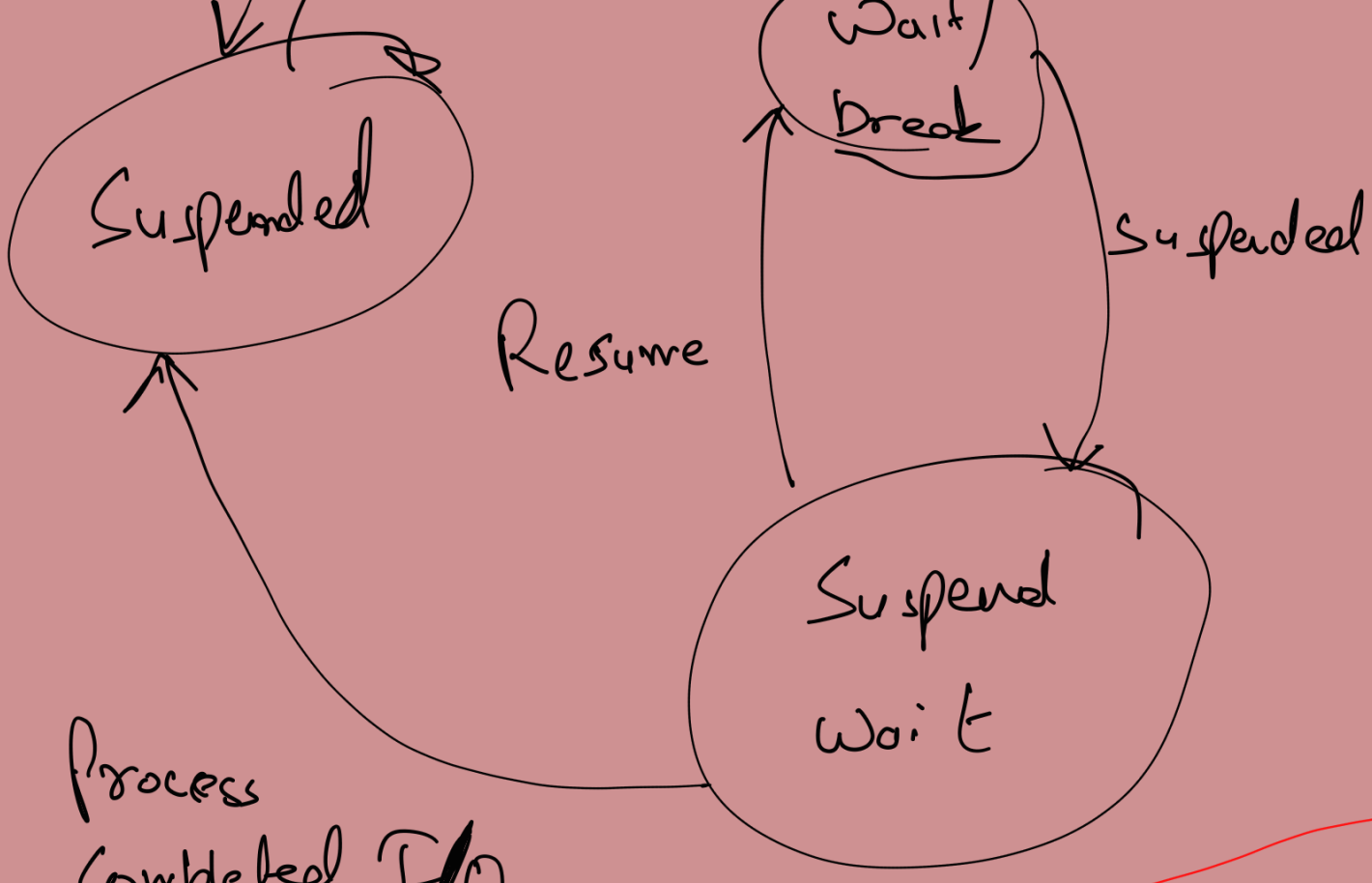
⇒ When multiple programs are loaded into memory at the same time, so that CPU won't stay idle at any moment of time.

Multi-tasking :- (Extended Version of Multi-Programming)

⇒ Multiple task running at same moment of time.

Process States :-





NOTE:- If a high priority request comes, program running can be send to wait/suspended state & can resume later on.

CPU → Scheduling Algorithm:-

① FIFS :- First Come First Serve

Req - come

↓

Req. executed

→ Starvation Problem:-
if a large process comes, past that all other request might get delayed for a longer duration & will feel like, they are not even processing.

#2:- SJN / SJF :-

Shortest Job First / Next:-

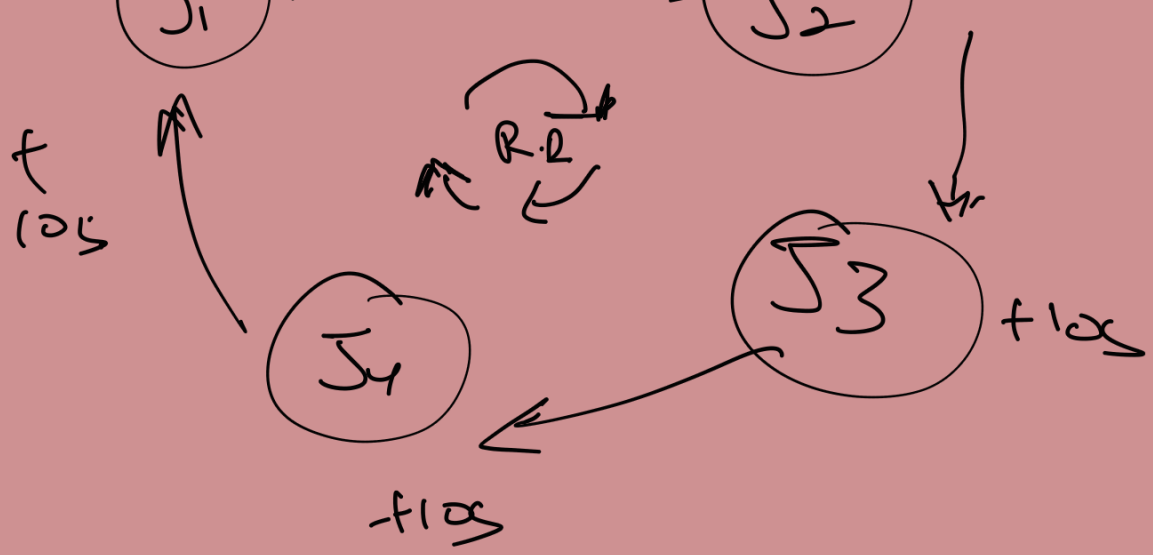
execute the job with min. Burst time

Time req. to execute a Job.

CONS:: > Starvation Problem.

#3. Round Robin (RR) :

allocates a specific time to a job in a round Robin manner.



Cons:-

if Some high priority Job comes (need to comple immediately)

⇒ it will Stop. & end the OS (if that Job is Critical for OS)

Priority Scheduling

⇒ Jobs executed based on Priority.

Problem :-

least priority Job will starv

Multi-level Queue Scheduling :-

⇒ Use Rand Robin

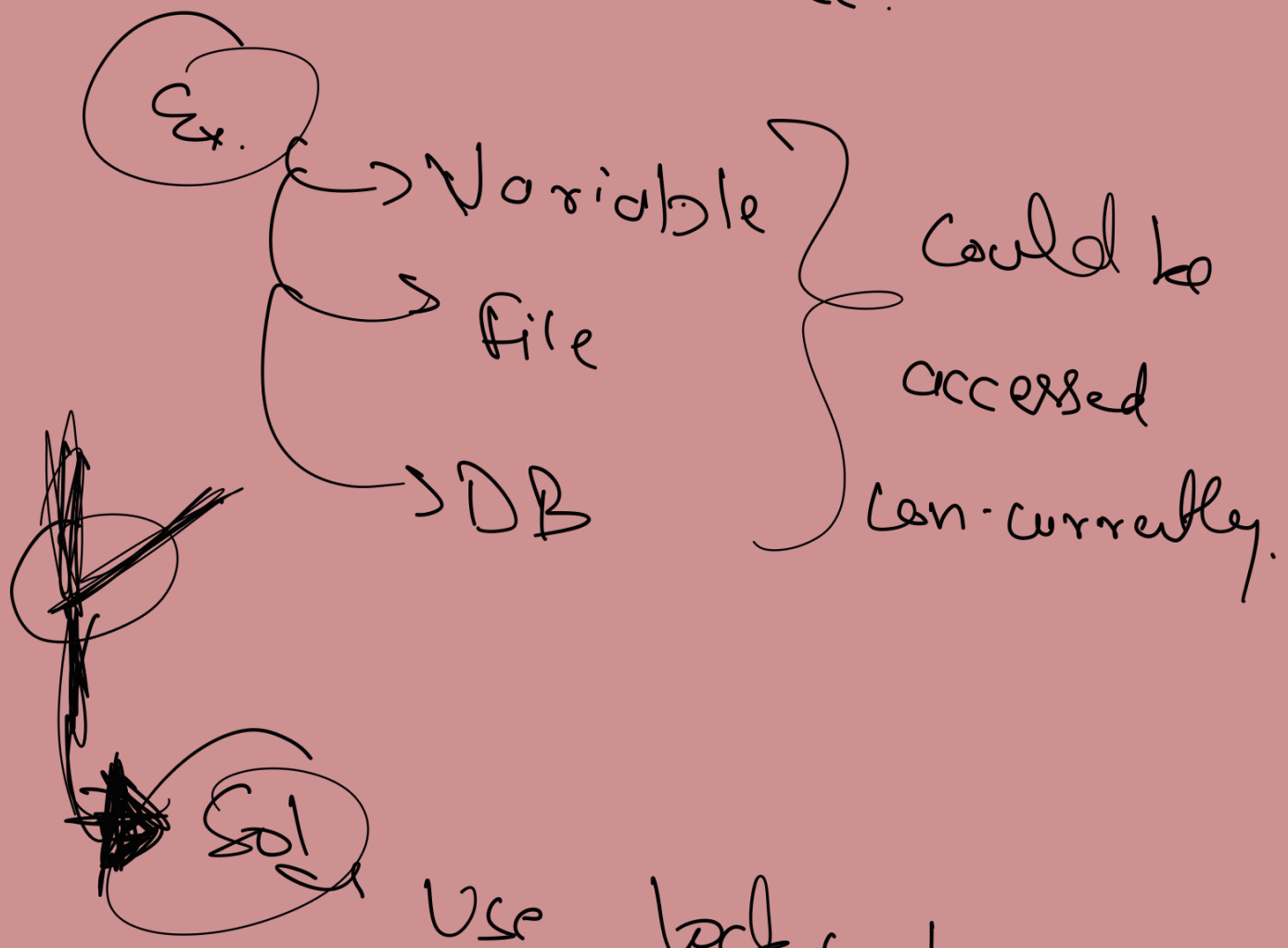
+

Priority based Scheduling

⇒ multiple Job-queue

Critical Section Problem:-

→ It represents a section of code where a process/Thread access the same resource.



Process Synchronization OS

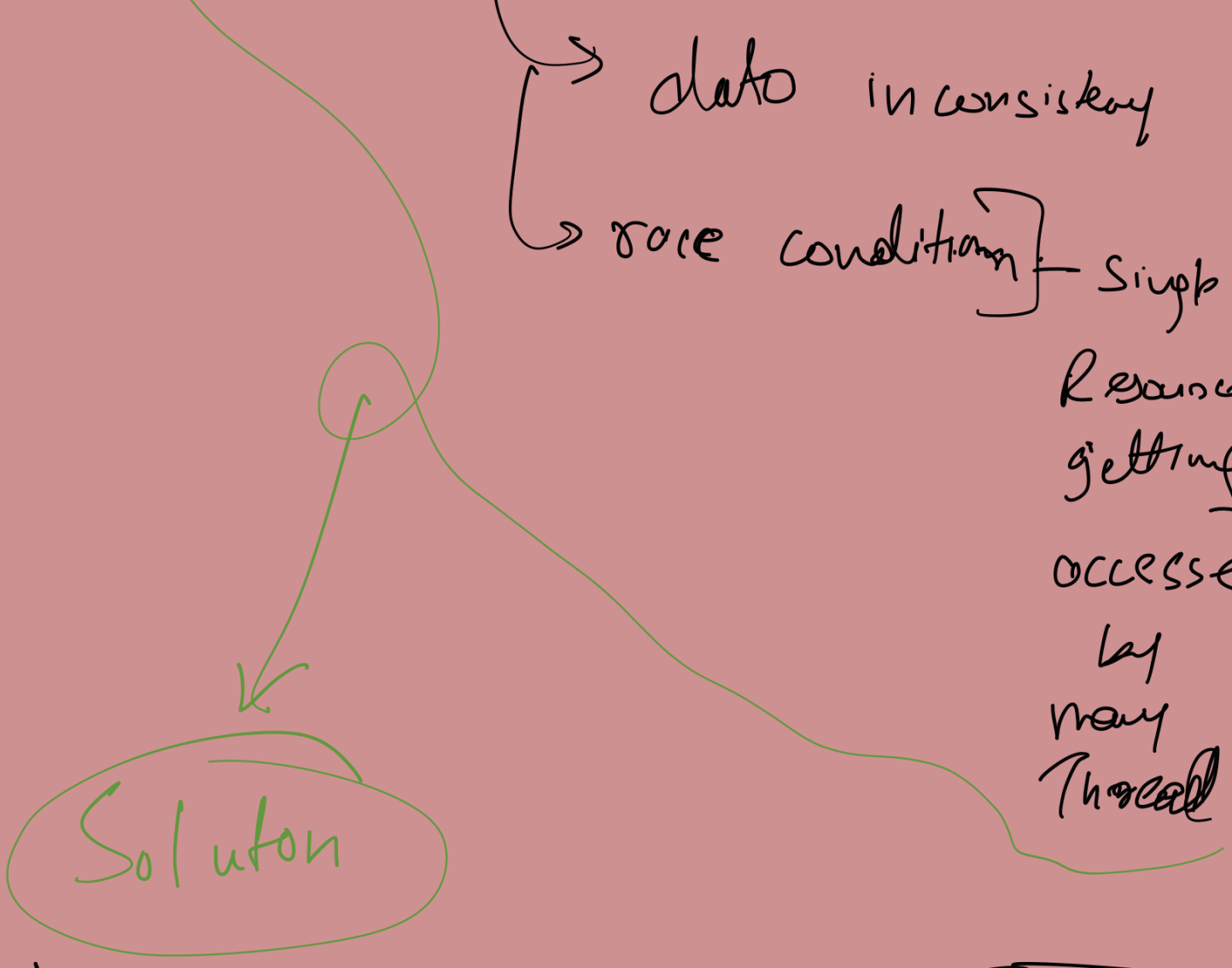
↳ it's like a traffic signal that help regulates the flow of Vehicle on an intersection

Process / Thread can work together

HARM-Lessly

like

↳ Deadlock



For Synchronization

We will things can be done

1. Mutual Exclusion :-

Only one process/Thread can access critical section at a time.

(2) Progress :-

mechanism to allow Thread to make progress into Critical System

(i.e.) →

to avoid situations where all process / threads are blocked indefinately → leading to Deadlock

(3) Bounding waiting :-

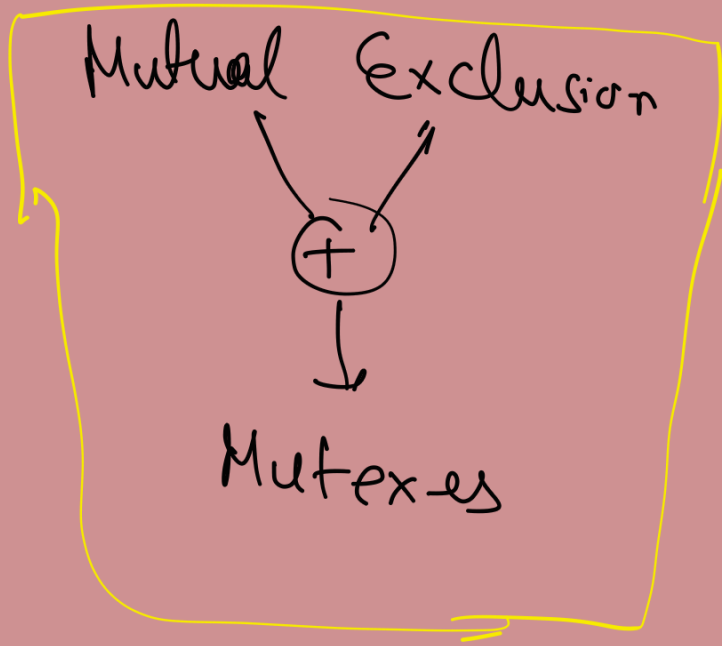
- process should be known that how long it wait to access Critical section.

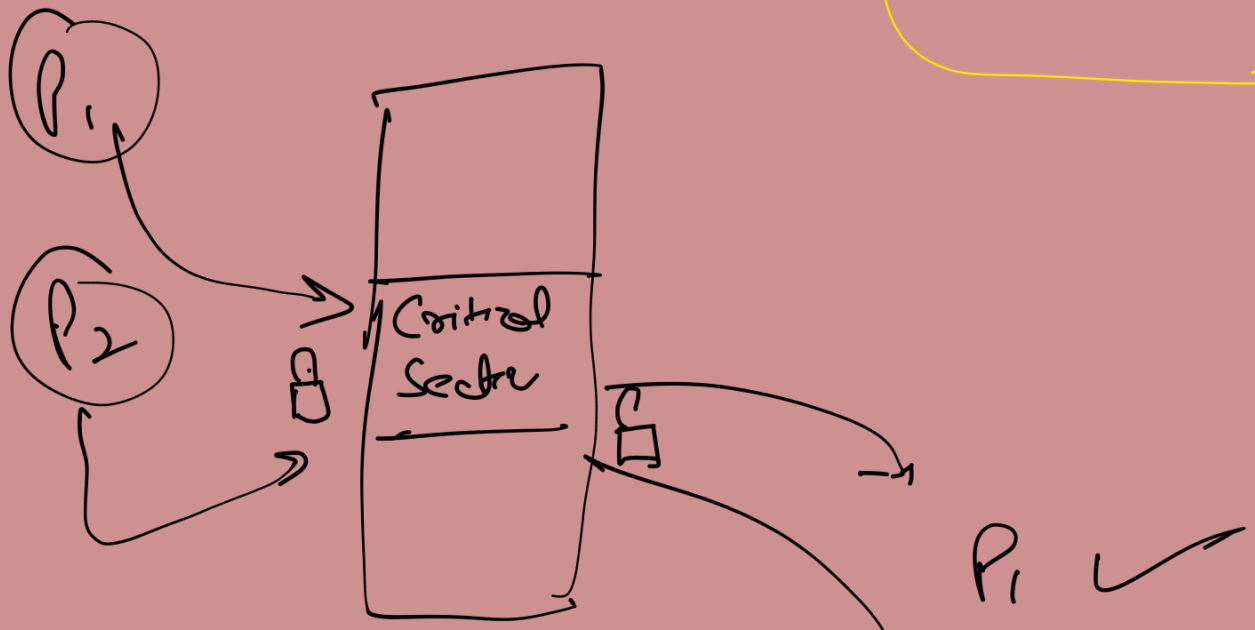
(i.e.) → it should not be like Thread

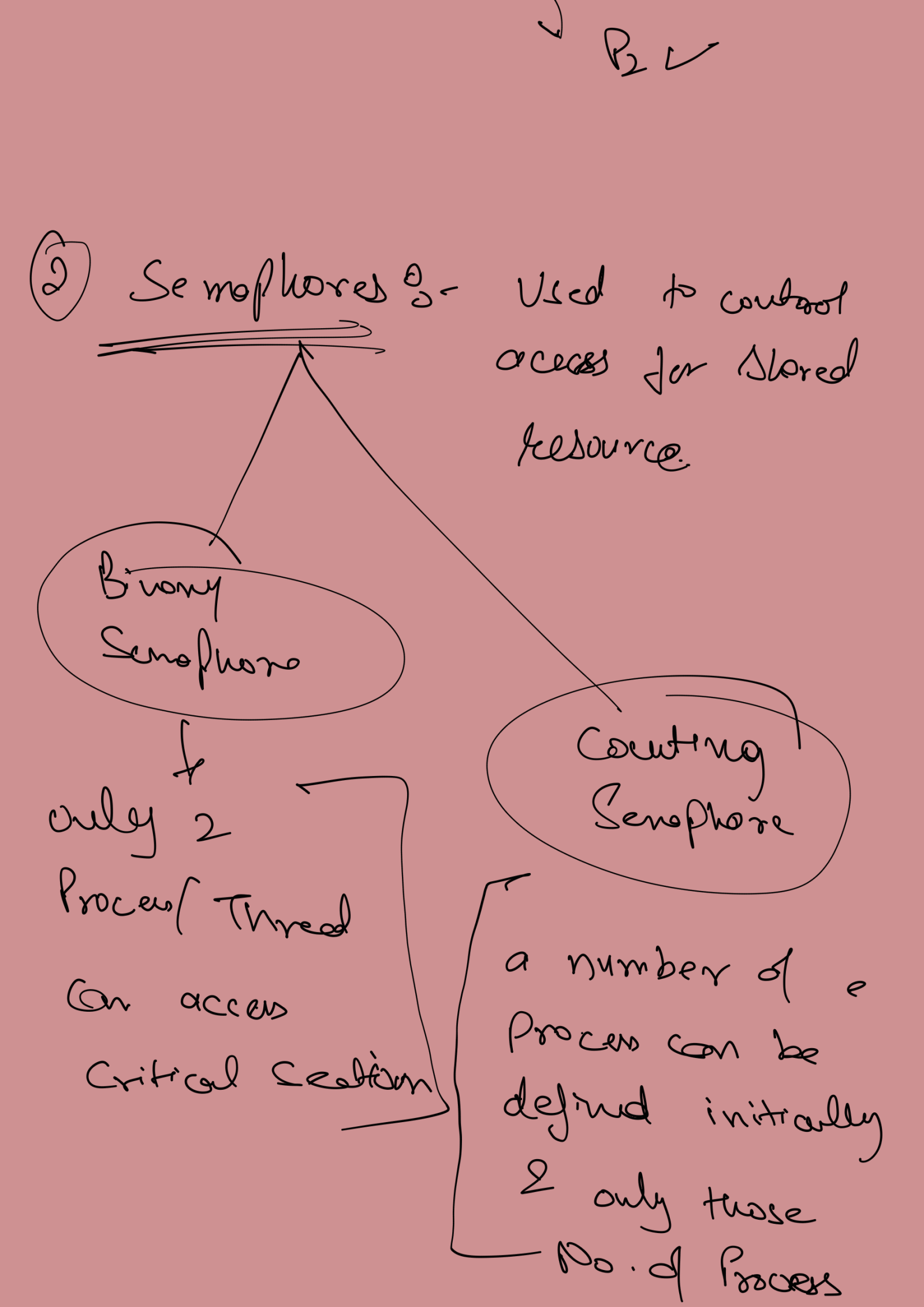
indefinitely to access Critical Section & still not getting it.

How to achieve that?

1. Locks / Mutexes






P2
2 Semaphores :- Used to control access for shared resource.
Binary Semaphore
Counting Semaphore
only 2 Process/ Thread Can access Critical Section
a number of Process can be defined initially & only those No. of Process

Can access critical section at same time

3 Read-Write lock 0/

Reader → allowed as many as can

Write → Mutual exclusion, only one at a time

DEAD-LOCK

Stucked & nothing can be done now.

Example

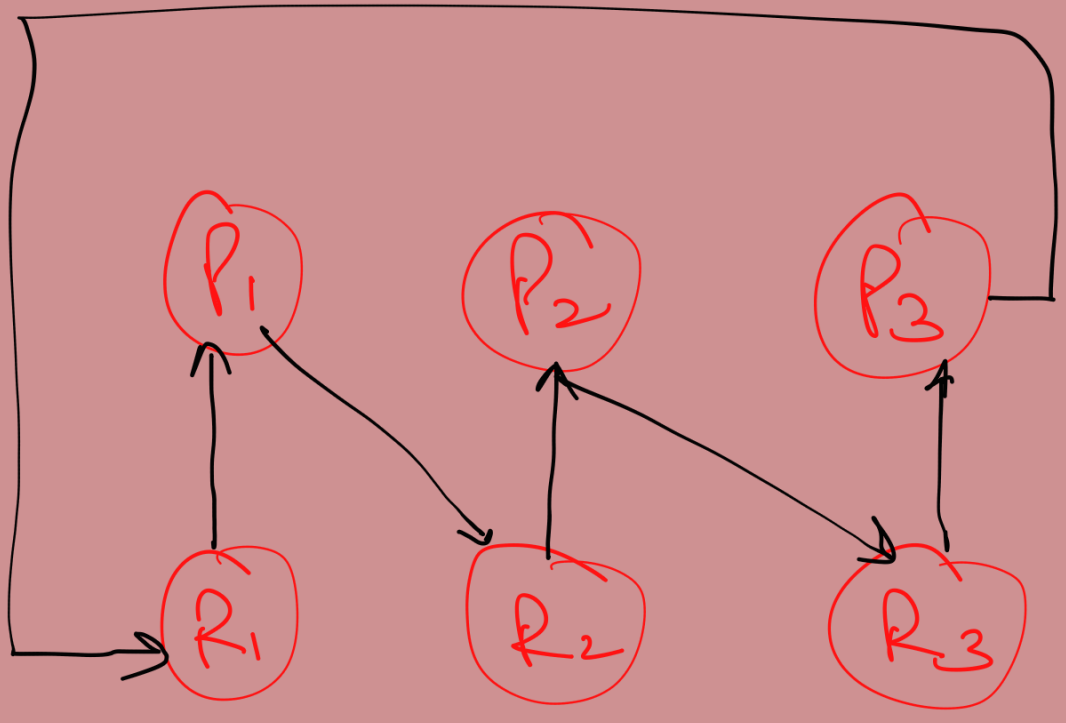
need Job

Fresher

Company

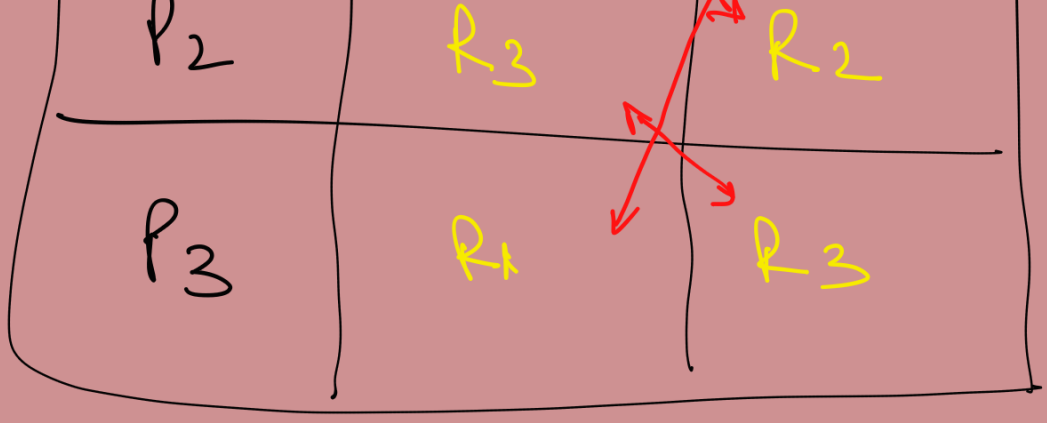
I Need experience?

Deadlock.



Process	Need	Has
P1	R2	R1

None



Then Can work.

Necessary Condition for Deadlock:-

All condition should be there.

1. Mutual Exclusion
2. Hold & wait
3. No Preemption
4. Circular wait

4 Conditions REQUIRED

Deadlock Handling Technique :-

(1) Deadlock Prevention :-

↳ Says that try removing one of the four necessary condition of deadlock.

(2) Deadlock avoidance :-

↳ use of some resource allocation algorithm, so that deadlock should not occur.

Ex :-

↳ Banker's Algo

(3) Deadlock Detection :-

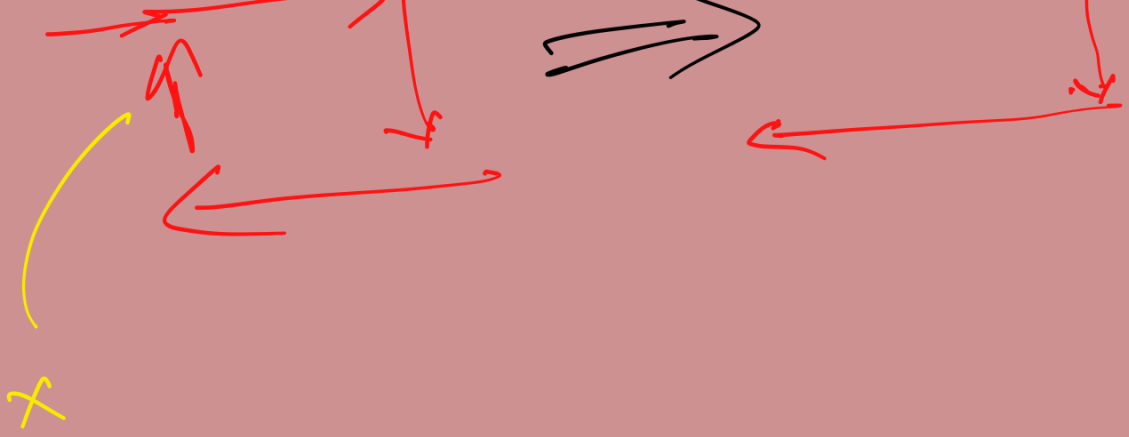
Says check periodically whether deadlock occured or not:

if occured → Take necessary steps (like process preemption)

(4) Deadlock Recovery:-

Says, Terminate one or more process → which can help removing deadlock.

Eg:



now after removing this deadlock can be recovered.

(5) Deadlock Ignorance :-

→ says :⇒ IGNORE only.

like in Windows's system crash

↓

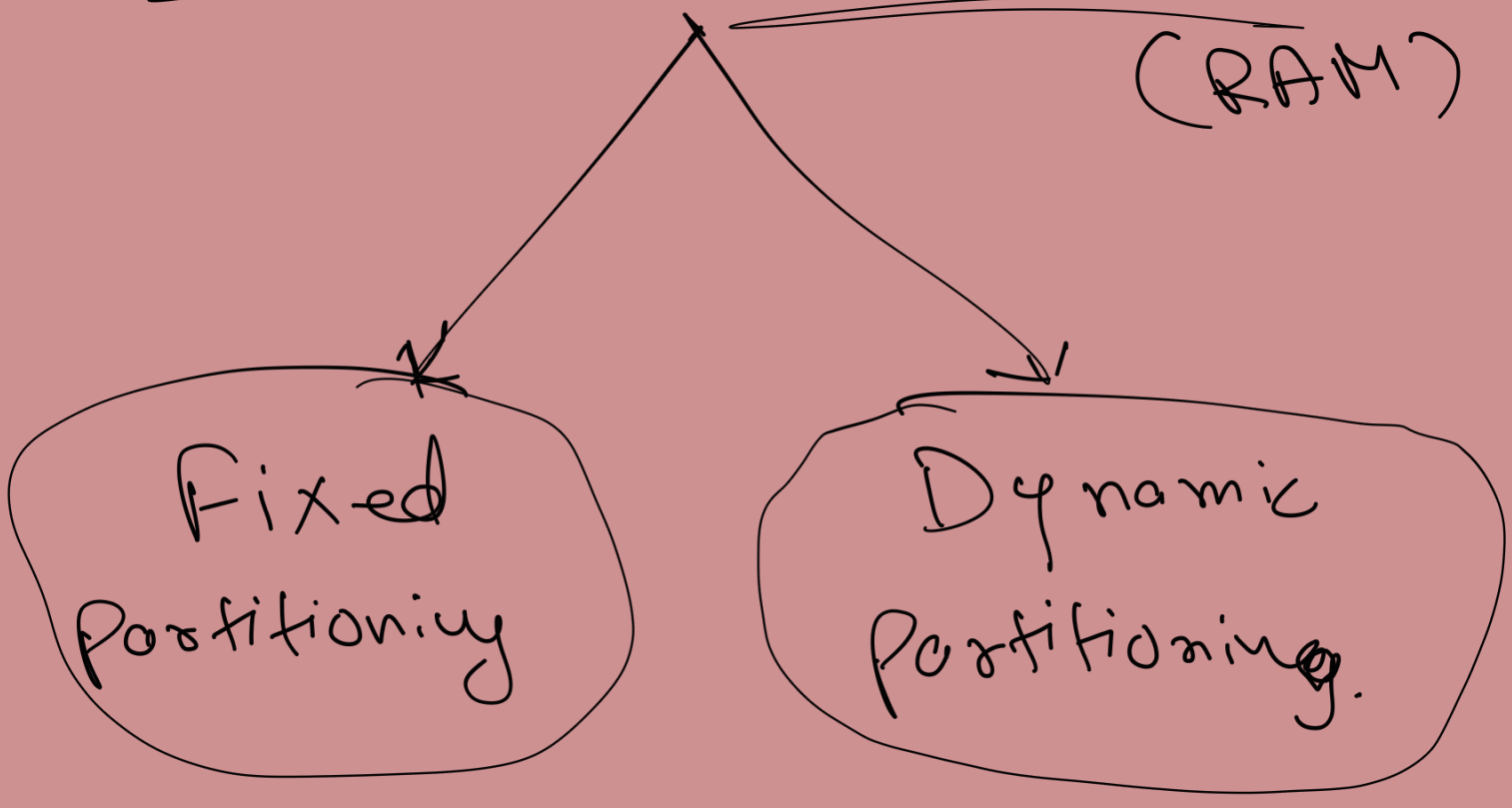
means, deadlock occured

↓

Do Nothing.

Memory Management :-

(RAM)



#1 FIXED Partitioning :-

↳ divide M/M into fixed segments (say 4 MB)

Cons:-

① P_1 require 2 MB only)

↓

{ 2MB of M/M is wasted.

② P_2 require 8 MB.
2 allocated in 4MB.

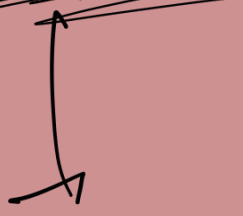
↓

Can't ever run

ends up with

FRAGMENTATION

FRAGMENTATION



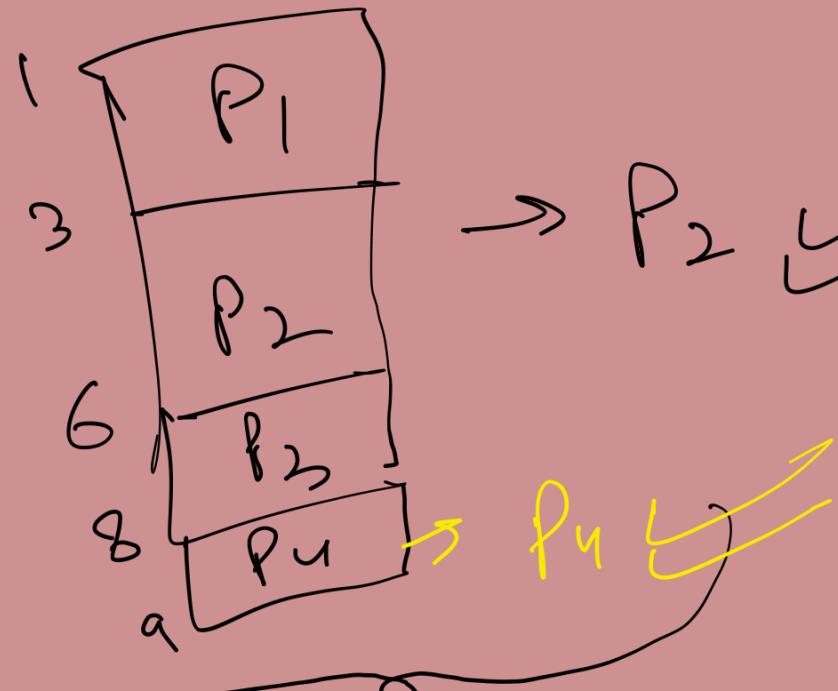
means,

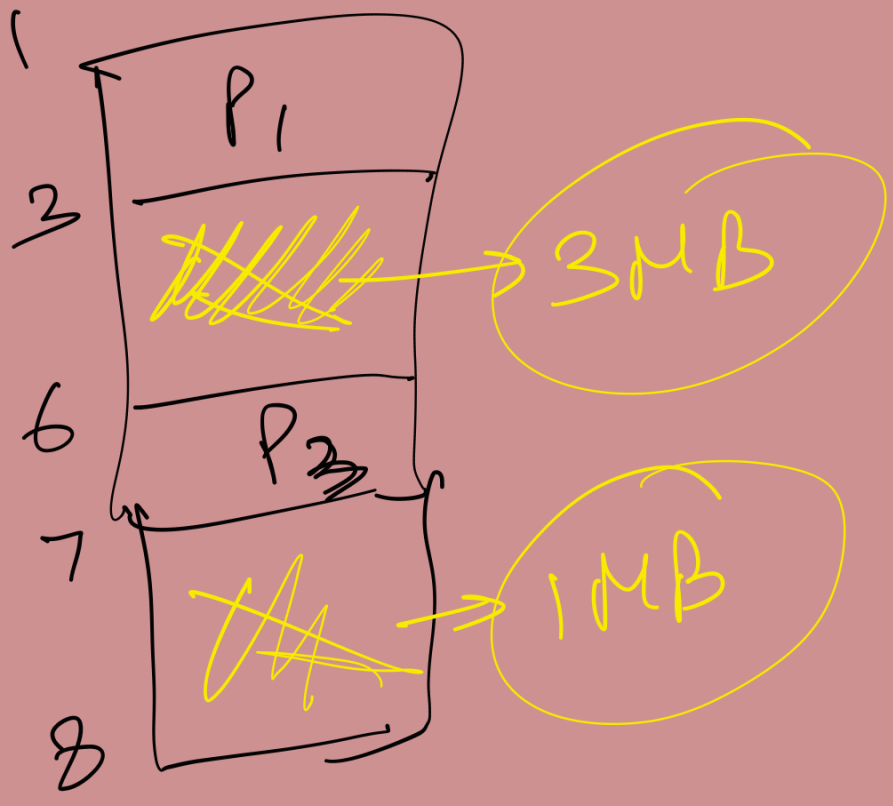
even though I have space still I can't use it.

#2 Dynamic Partitioning:-

↳ give M/M space as required by the process

if a proces has completed the task, then M/M of that particular area is deacated & now it becomes rigid or only that sized proces can used it now.





Now, if P5 comes & ask for 4MB of M/M

↓

FAIL

even though I have it, I can't give it to them

Types of Dynamic Partitioning

First space available greater than or equal to required

Iterate all M/M space & Then allocate the best M/M block

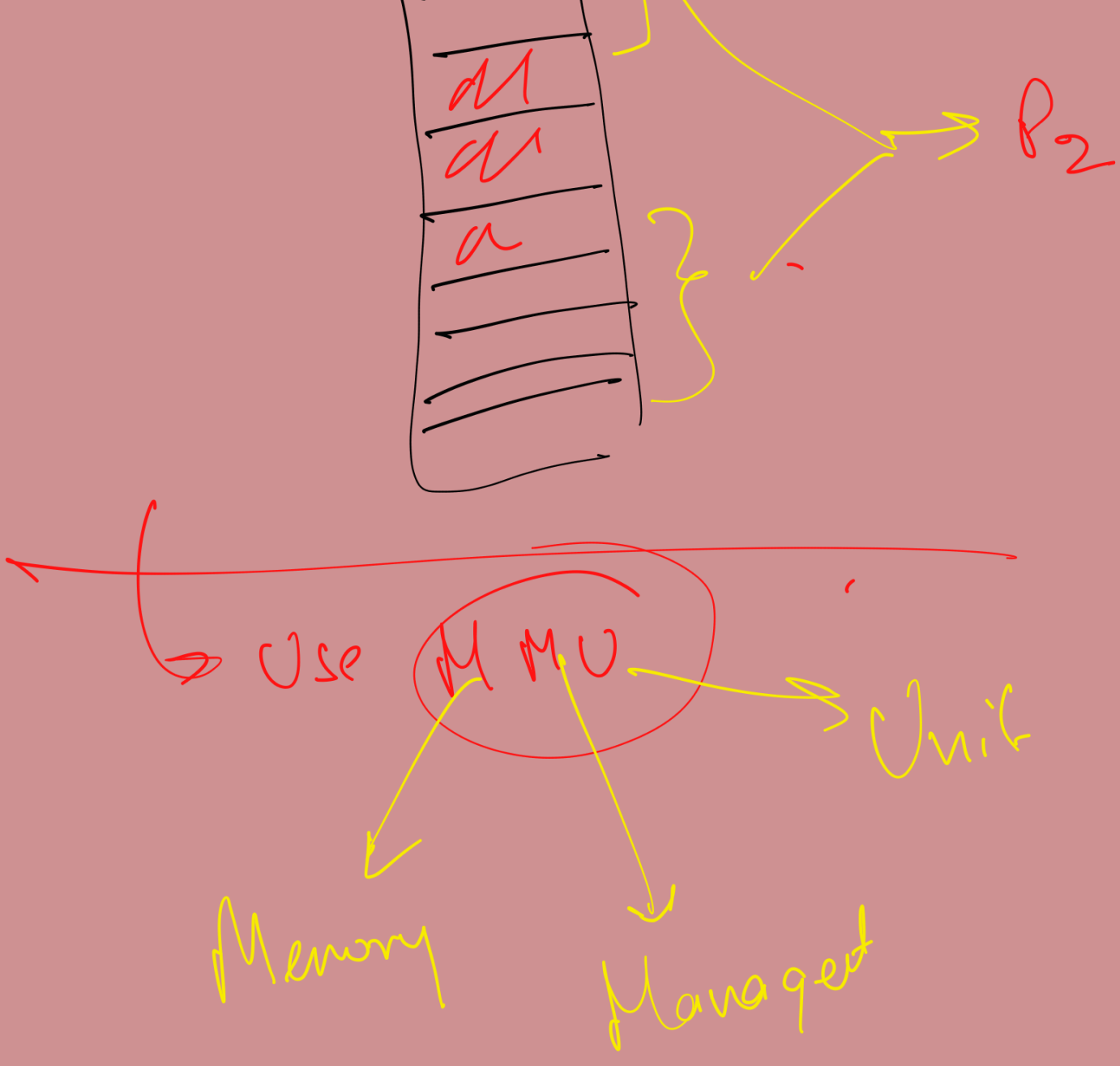
allocate the largest available space to Process.

space
allocate it

to
process.

PAGING :-

↳ Divide M/M into Pages of fixed size. & now no need to store Process in a Continous M/M block.



MMU Uses → Page table

for

Managing Pgs

Virtual M/M :-

↳ in simple terms

a process need 10 MB of M/M

but for working it only need

5 MB.

↓

hence load only required size of M/M & ignore the rest

↓

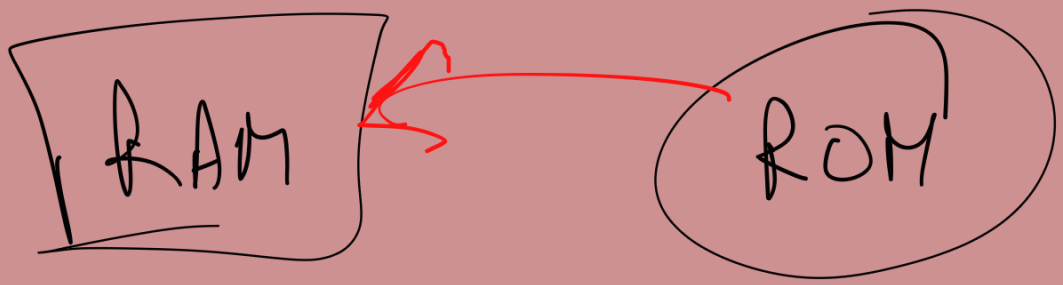
in terms

a Concept that lets a computer use more M/M that it actually

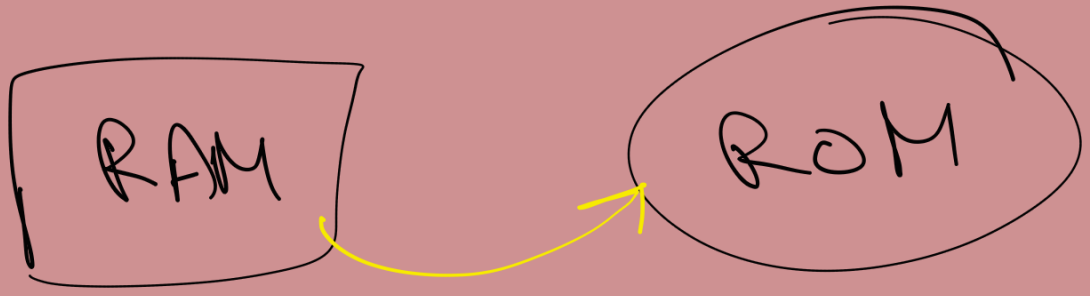
has.

It creates an imaginary M/M space by combining RAM & ROM

when Program need it



→ when not Required



PAGE - Replacent Algorithm :-

in Paging also, not all Pages are kept in Main M/M, & only required pages are kept there.

So

how to prevent Page Fault

Page Replacent Algo's are required.

Page req. Not found in Main M/M

① FIFO Page replacement Algorithm

(Queue like Algo)

Same store pages till the M/M has capacity & once the capacity exceeds the capacity, replace the oldest page with the required page.

→ Easy to implement but not used much.

#2 OPTIMAL Page Replacement:-

↓

Says remove that page that is required late in future.

↳ Best Algo but not feasible to implement.

↳ its like predicting future.

→ Can be used for benchmarking

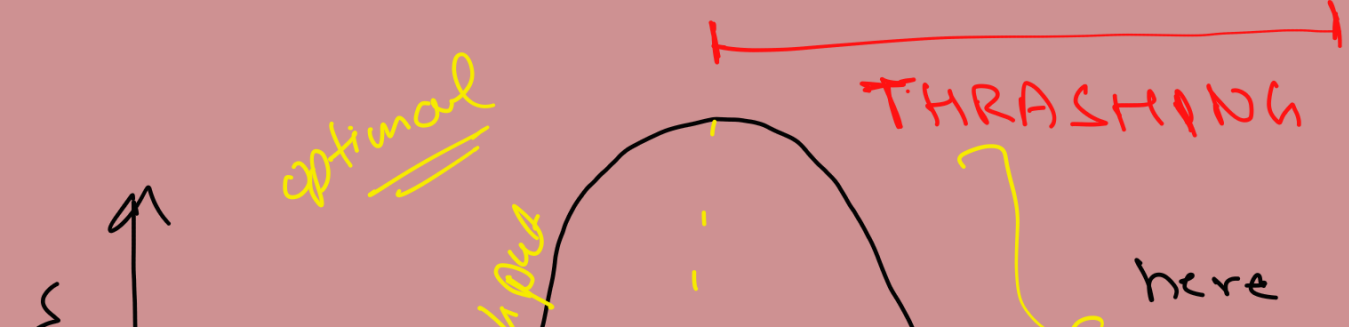
#3 Least Recently used (LRU):-

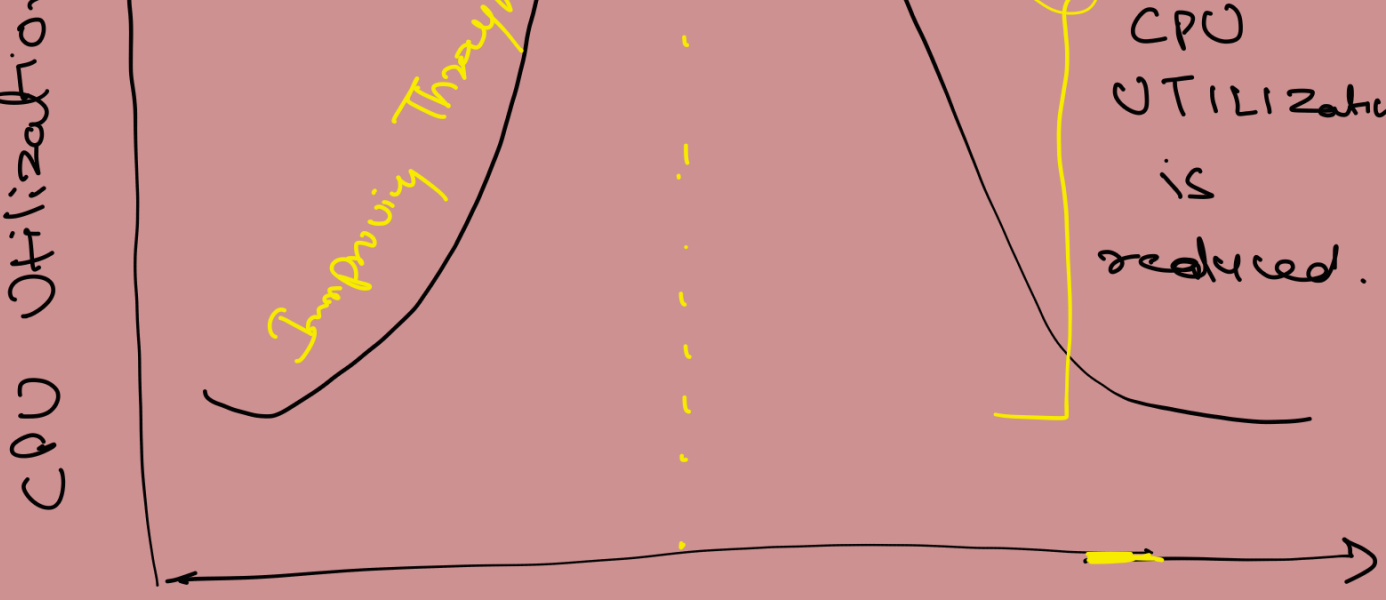
↳ Remove that page from Main M/M which was used least recently.

THRASHING :-

↳ a concept where computer system spends more time in page replacent rather than executing useful work.

→ it happens because degree of multiprogroming is increased a-lot.





Solution:- ① Increase RAM
② Reduce degree of Multi-Progrm

SEGMENTATION:

→ MOTO → To reduce Page faults.

↓

here Program is not divided by OS. → Users decidy

↳ Say a user progam has

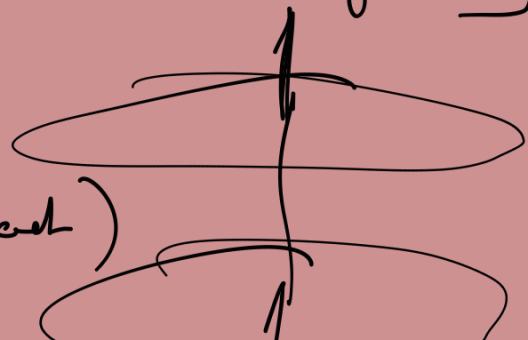
8 LOC & it is divided into 2 pages of 4 LOC each.

&

Why to load these two pages when required even if we know we need to run it together.?

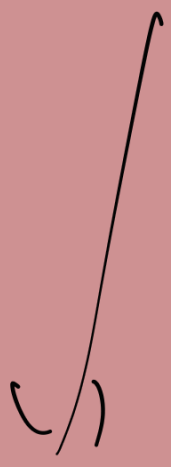
hence load all pages in one go

DISK Management :- { For Secondary Memory }



Seek Time :- (Forward & back)

Rotational Latency :-
(Rotation)



SSD :- Solid State Drive

The token to go to sector where data need to be read/write

Notes prepared from YT-channel Coding Shuttle by Anuj - Bhaiya

&

Video Done :- O.S in one Shot.