Agenda

1. Intro to OS
2. Uni v/s Multi Programming
3. Processes
4. CPU Scheduling
5. Scheduling Algorithms

⇒ Operating Systems

→ How the app^n that we will create work under the scenes

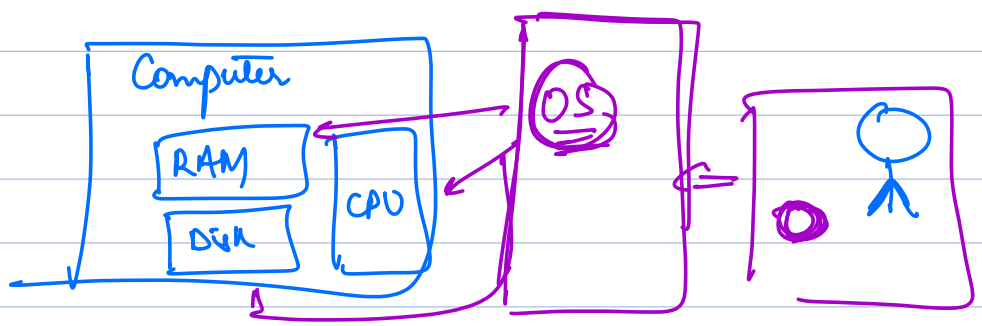
OS

- Program / Process
- Concurrency
- Memory Mgmt

Java

Python → Recorded videos

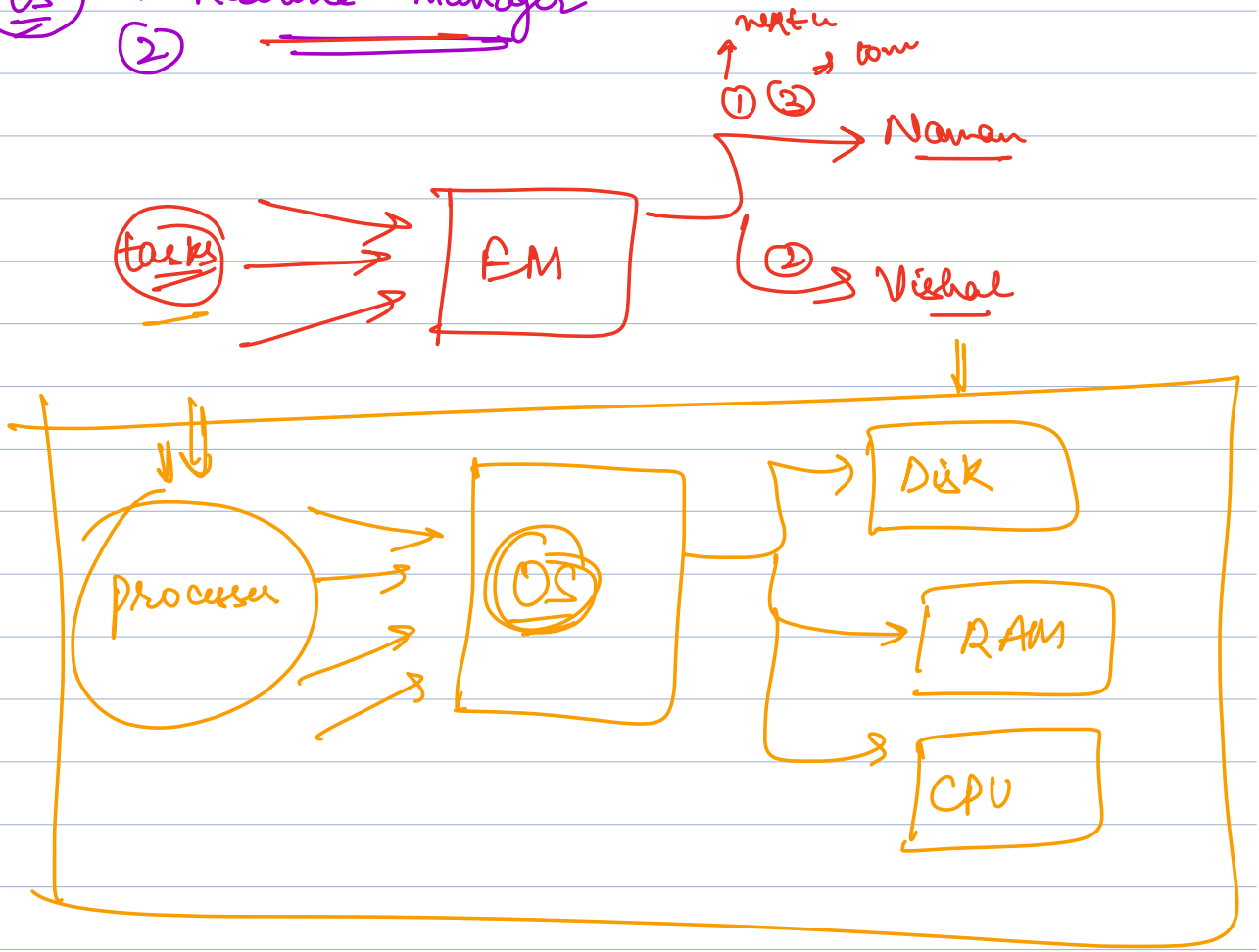
What is an OS



OS ① intermediary that allows us to interact with the computer

OS ⇒ Resource Manager

(2)



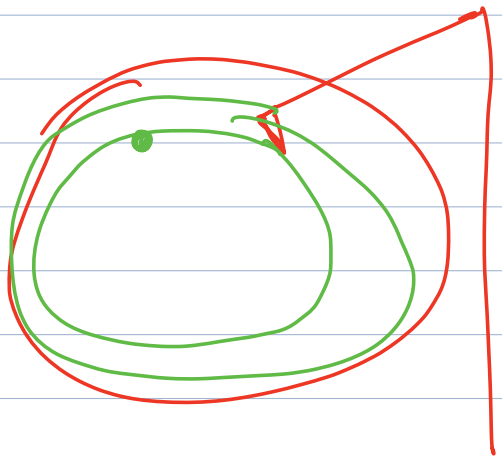
(3) API for app^n developer ⇒ provides a lot of utilities ⇒ disk / RAM / N/W

(a) Print something to command line ⇐ OS

print() Cout cc print (nC)

(b) Sending a N/W req

OS ⇒
- open a port
- connect to a client
- convert data into packets



UNI Programming v/s Multi Programming - OS

One → made to do only one thing
→ it can run only one program
→ ATM Machines
IoT Devices
→ Smart Bulb
→ Washing Mac^n
→ Calculator

Many ⇓
OS that can run more than one programs

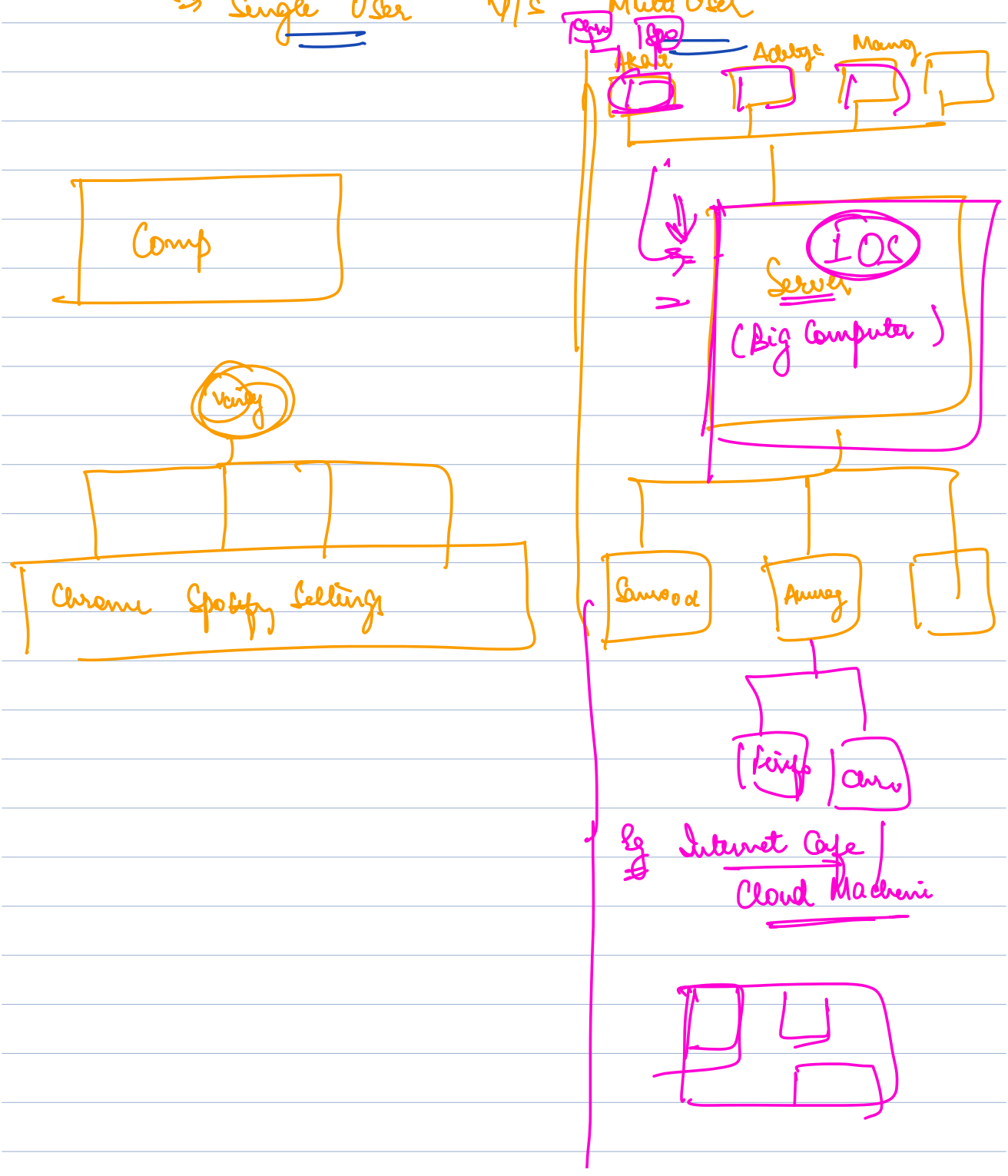
Eg → all the general purpose OS
→ Window, Mac, Android, iOS

Multitasking
Program → UNIX (MacOS / Linux / Ubuntu) → Android
Tasks → Window

CATEGORIES OF MULTIPROGRAMMING OS

(1) # users that can work at same time

⇒ Single User V/S Multi User

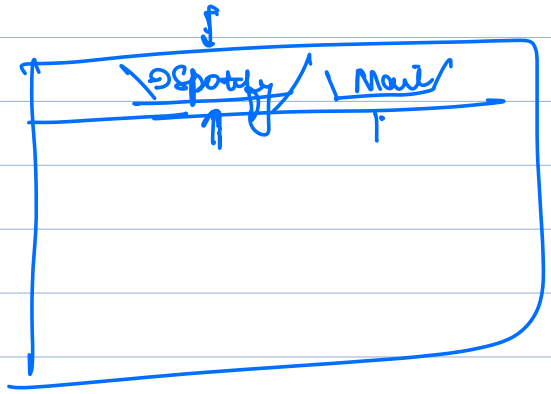


② Preemptive V/S Non Preemptive

How will an OS run more than 1 task

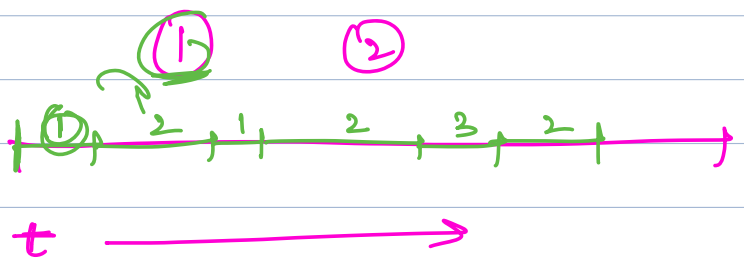
① One after other

② B/W each other Before Completing



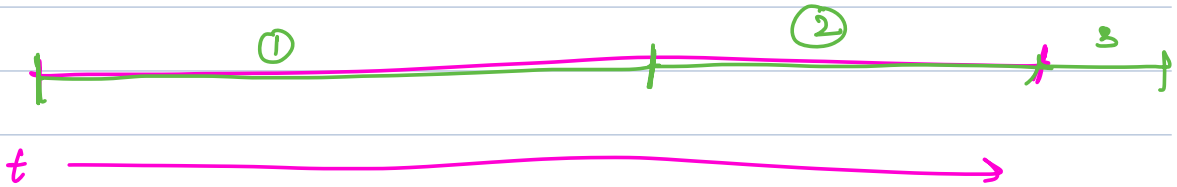
PRE-EMPTIVE

⇒ OS than can move to another by pausing the prev task in b/w.



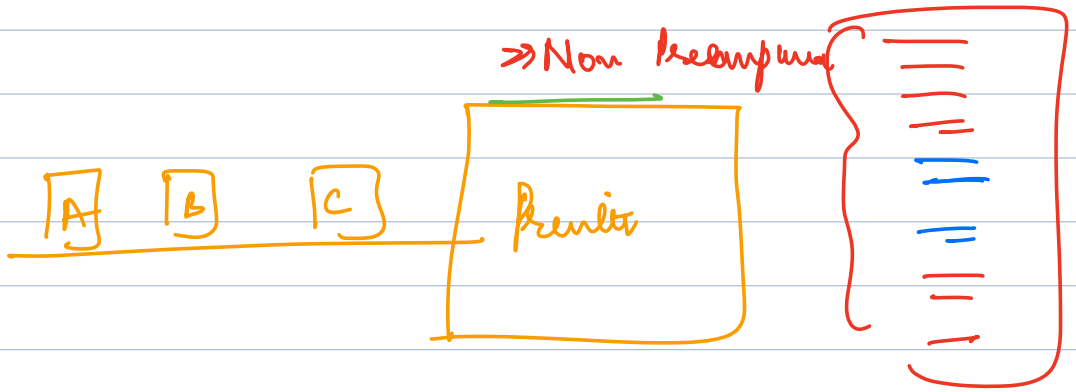
NON PREEMPTIVE OS

① ②

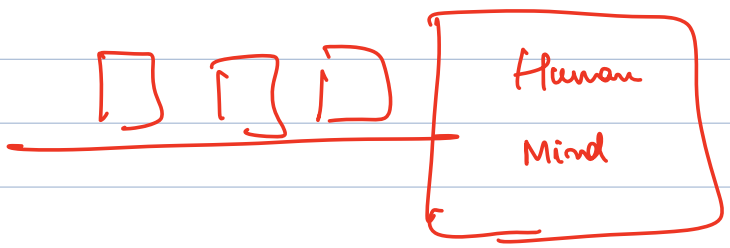


→ Can't stop a task in b/w

→ move to another task only after the prev task is completely done



⇒ Non preemptive



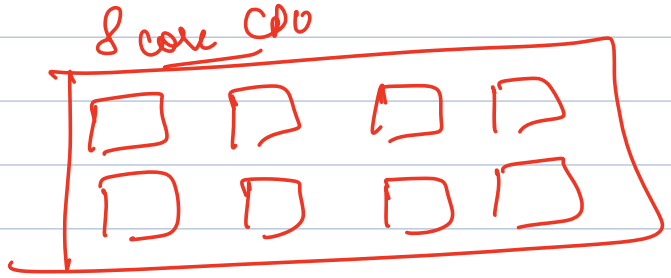
Preemptive

Quad Core / Octa Core CPU

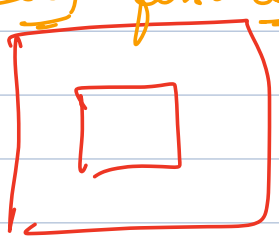
1 core ⇒ 1 Brain of CPU

1 Brain can only do 1 thing at 1 time

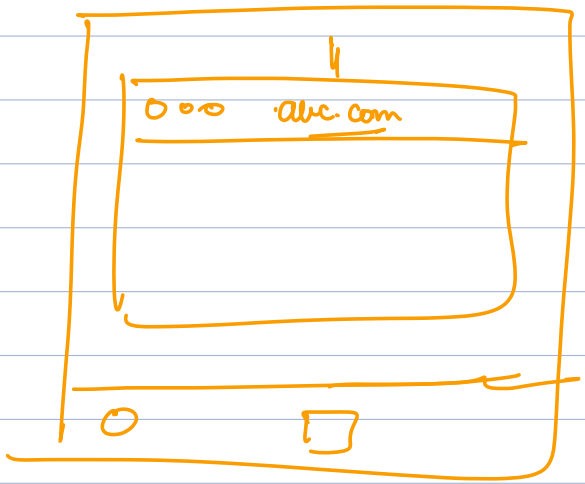
8 core CPU



200y first comp



⇒ Single core



OS

Non Preemptive → Load Website | Show Mouse Move

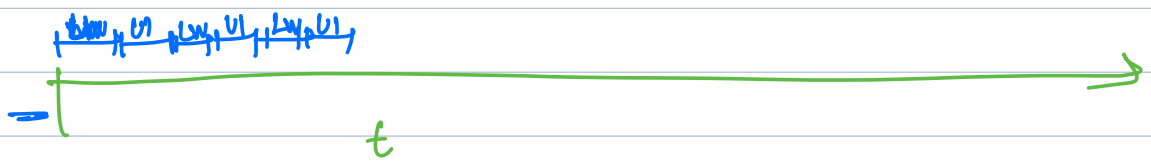
Preemptive OS

0.05s | 0.05s | 0.05s | 0.05s | 0.05s

LW | | LW | M1 | LW | M1

CPU

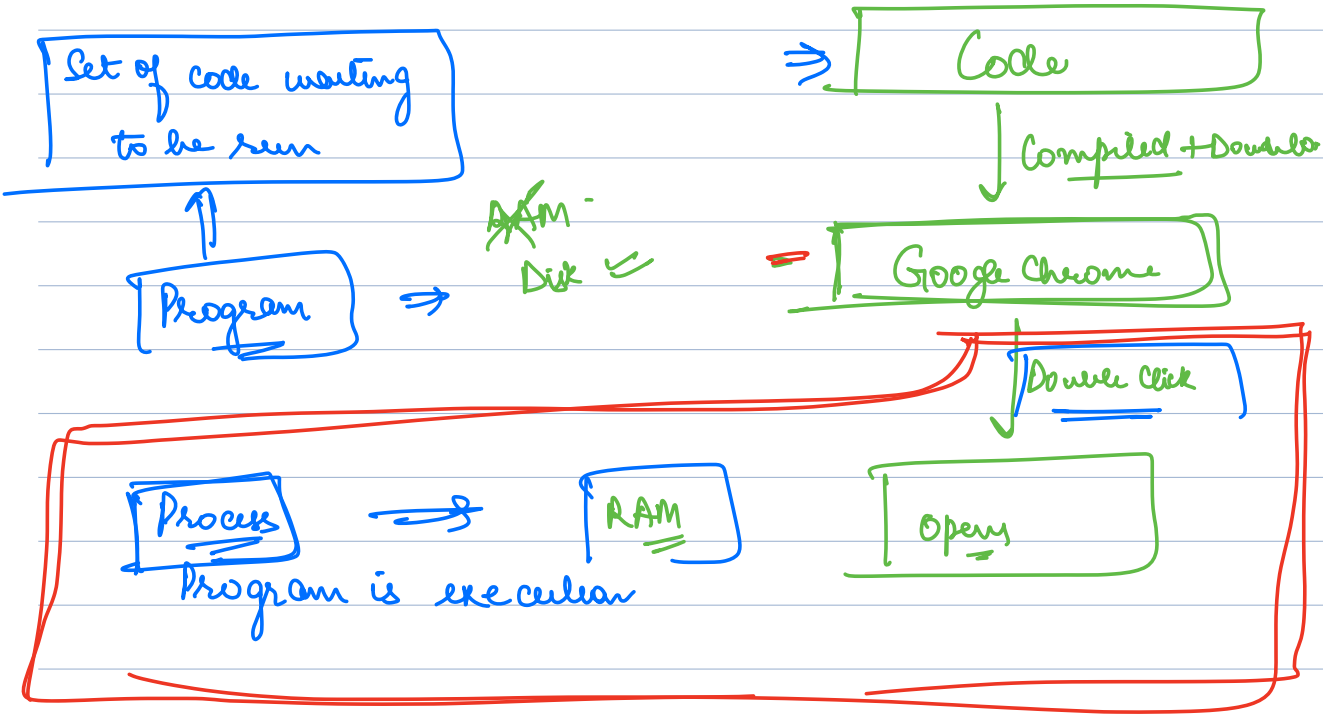
Preemptive OS



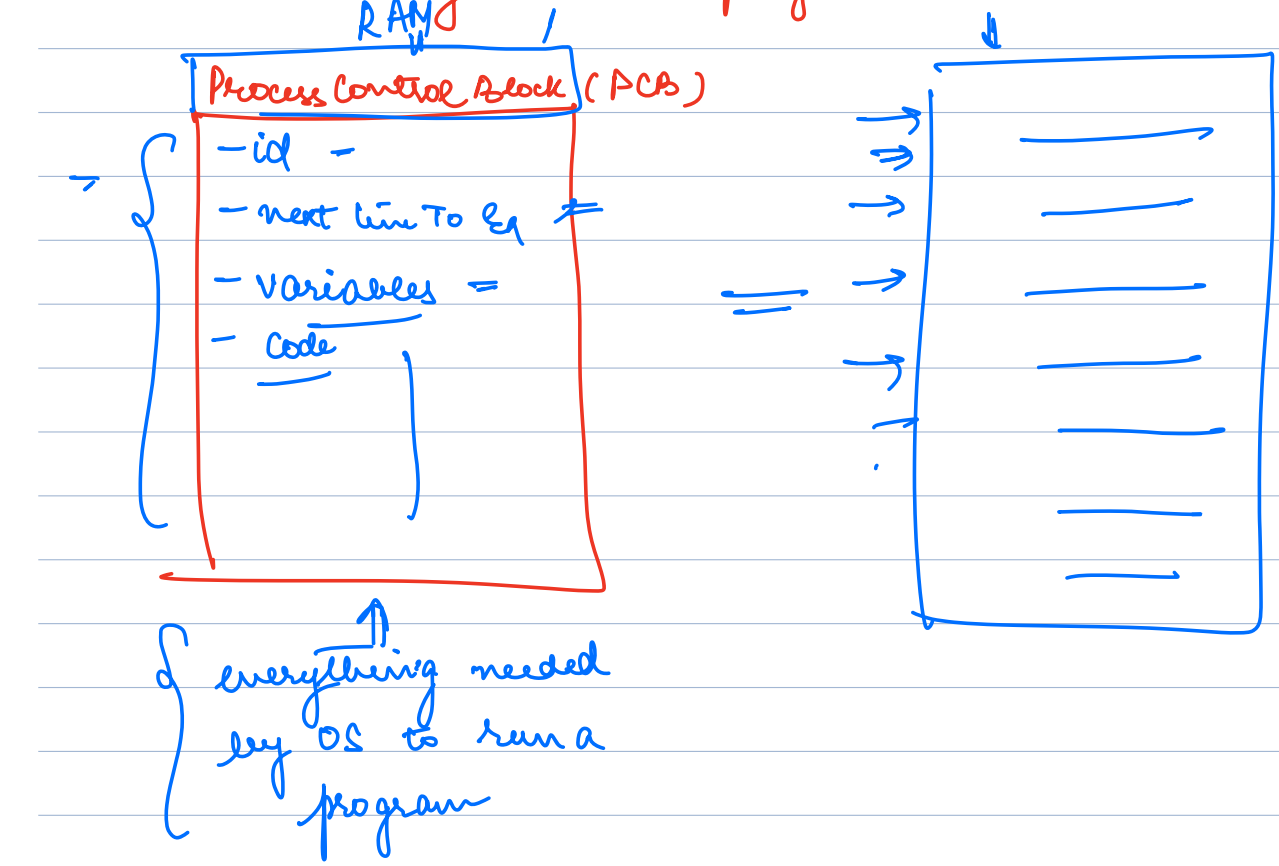
PROCESS

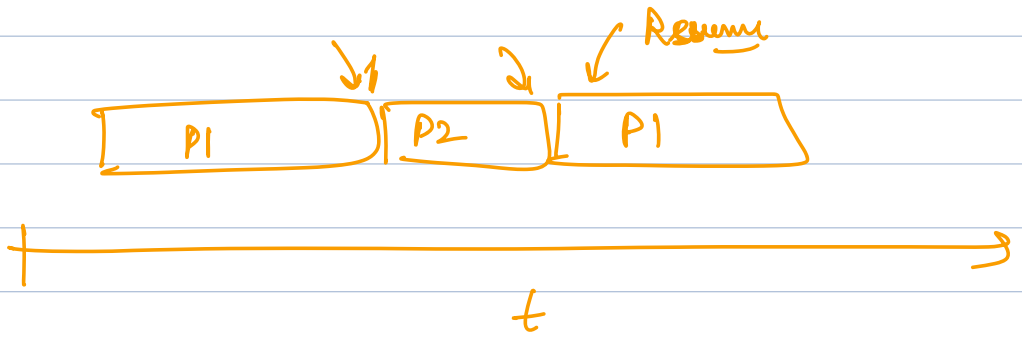
Multiprog == Multitasking == Multiprocessing

Prog ≠ Process



Process is nothing but a program that is executing





→ For an OS to successfully execute a prog, it will need to know multiple things:

1. value of all variable
2. line of code
3. id of process

PCB → in RAM

Types of Processes

1. I/O Bound Processes : Spend most of their time doing I/O
2. CPU Bound Processes
   → Mostly calculation
3. Doing both I/O and computational work
   → Games

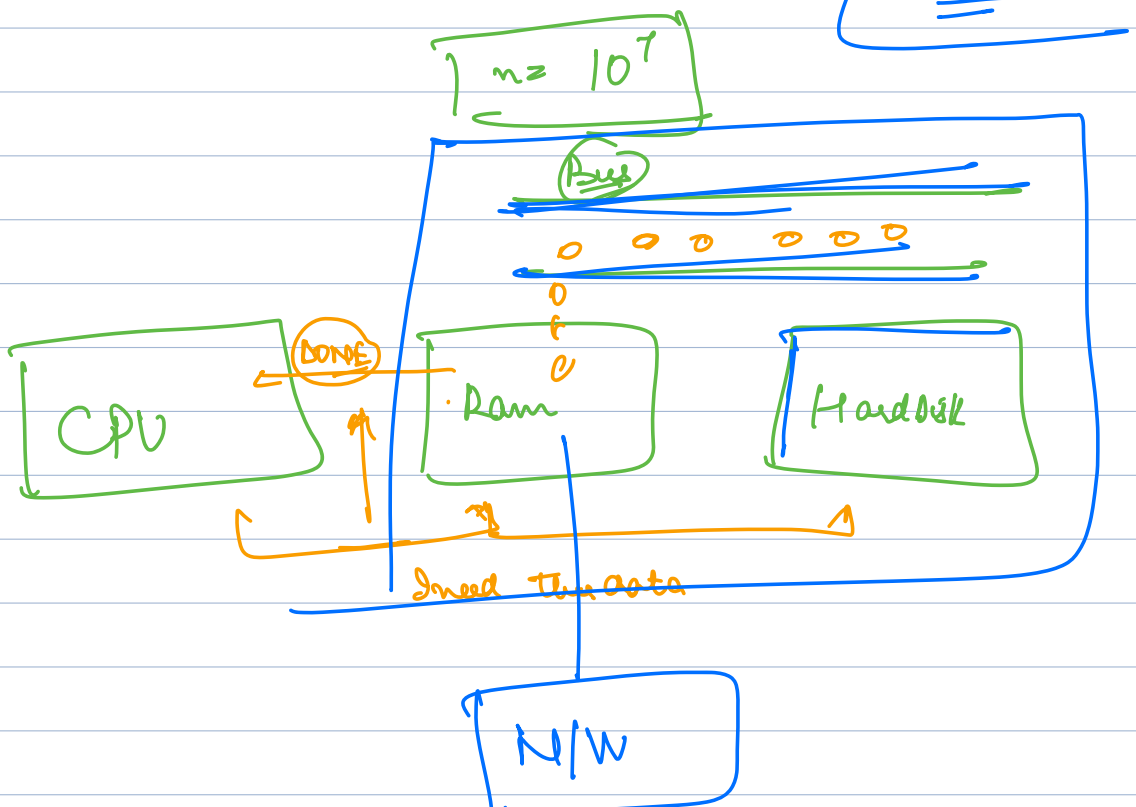
I/O → Input Output
- → Network access
- → Disk access (reading / storing)
- → Printing / Fetching from CL / Keyboard / Mou

→ I/O is very very very very slow

→ When a process is waiting for I/O to happen, CPU is not doing anything useful

String s = sc.nextInt();

2.8 GHz CPU | 2.8×10^9 ops/sec ⇒ COA



When an I/O bound process is running, CPU may be spending a lot of time idle.

How I/O happens

→ dev calls the

Signal sent to OS by an app's to tell it wants to do I/O

raises an OS interrupt

Can at this time CPU do something useful?

→ CPU wait (Sits idle) till the time OS satisfies that interrupt

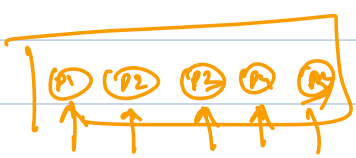
→ Run other task

Switch to another process

Which process

P6

P1



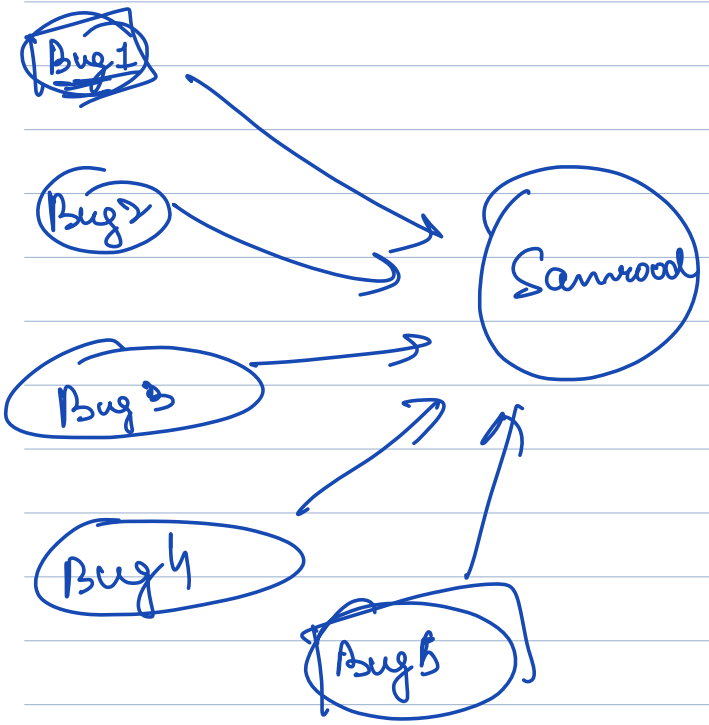
→ CPU/Scheduler

OS

⇒ decides what process run next

Problems due to CPU Scheduling

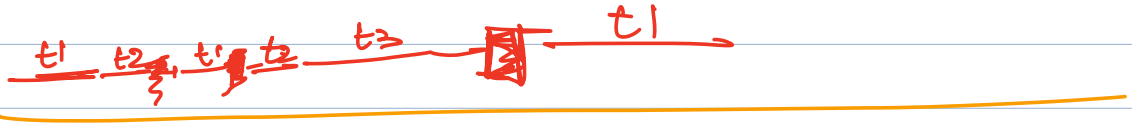
CONTEXT SWITCHING



When you move from task A to B, when you come back to task A you have to spend some extra time recollecting where you were.

⇑

Context Switch



t →

⇒ If preemptiveness happens too fast, CPU may spend a lot time doing context switch than actually doing work ⇒ which may be bad for efficiency

CPU Scheduling Algos

Decide what process should CPU run.

⇒ Theoretical Algo

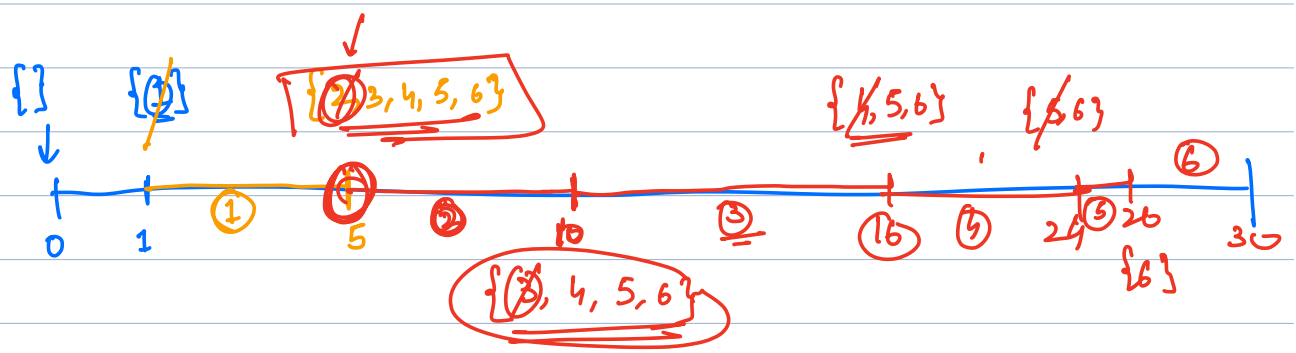
(1) FCFS (First Come First Served)

PID	time of entry	length
1	1	4
2	2	5
3	3	6
4	4	8
5	4	2
6	5	4

t0 — P1
t2 — P2
t3 — P3
t7 — P6

→ Non preemptive algo

→ Whenever a prev process finishes, the algo checks which is the first process waiting in the queue. It will run that process next.



② SRTF (Shortest Remaining Time First)

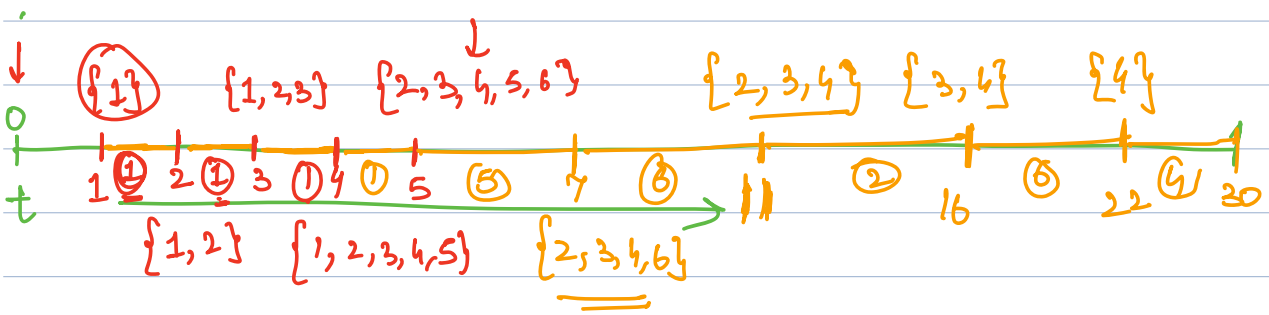
① Preemptive Algo : it may stop a process in between

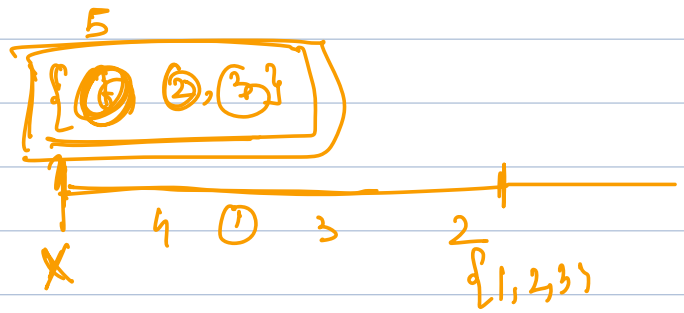
② Runs :

- ① When a prev process finishes
- ② When a new process comes into system

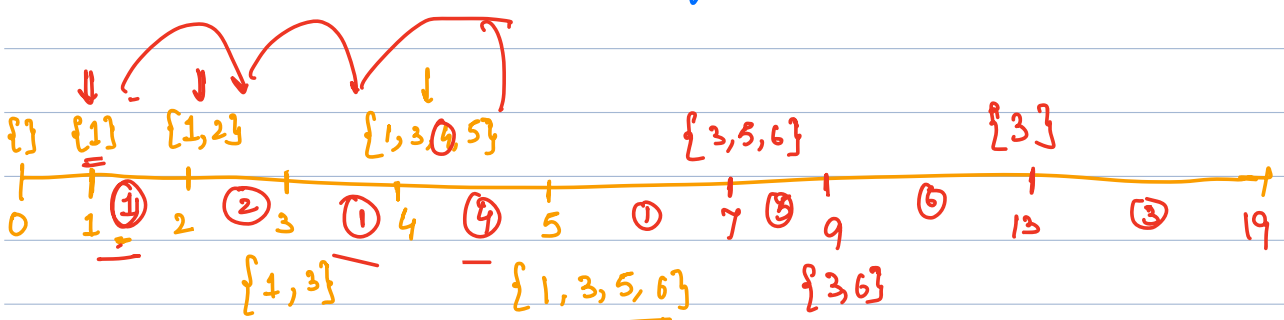
③ Whenever the algo runs, it chooses the process that has least time left for completion.

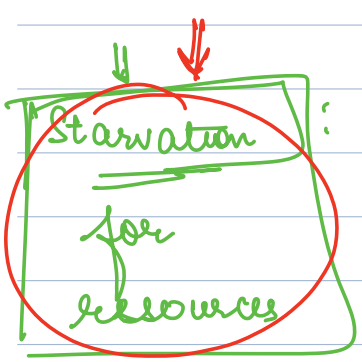
PID	time of entry	length	rem-time
1	1	4	4 3 2 1 0
2	2	5	5 6
3	3	6	6
4	4	8	8 0
5	4	2	2 0
6	5	4	4





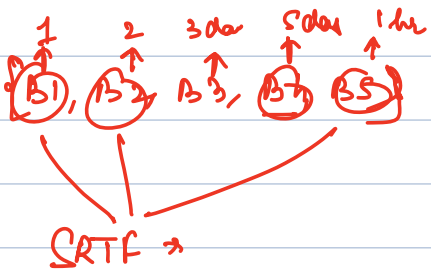
PID	time of entry	length	rem-time
1	1	4	4 3 2 0
2	2	1	1 0
3	3	6	6
4	4	1	1 0
5	4	2	2 0
6	5	4	4 0





: A process with larger length will rarely get time to run.

Adv: Processes will start getting completed earlier



{B4, B3, B1, B2, B5}

Adv of FCFS : Simple