Operating Systems - 2

← Load Balancers

→ Round Robin CPU Scheduling ⇐

→ Sol^n to Starvation

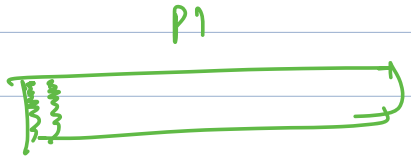
→ Threads → Throughput V/S latency

Theory
- → Intro
- → Multicore V/S Single Core CPU
- → Concurrency V/S Parallelism
- → Code:
  - Hello World
  - Print 1 to 100

Practical
- → Java

ROUND ROBIN CPU SCHEDULING ALGO

- → FCFS
- → SJF



⇒ Most of the processes will be making no progress

→ We want every program that is running to make some progress

RR

q = time quantum

= 2 s → 20 sec → FCFS

→ 0.2 sec → lot of context switching

after every q sec, it will schedule some other process to run.

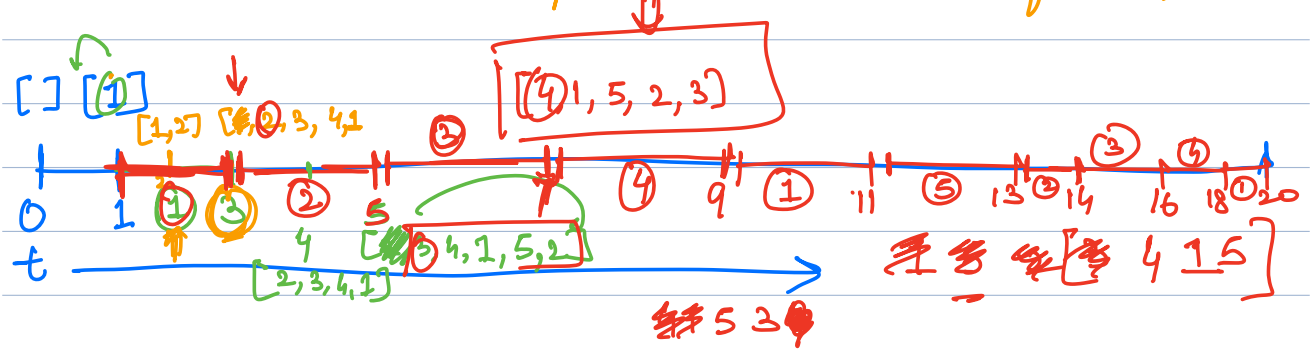
Processes

id	Arrival Time	Time To Complete
1	1	~~6~~ ~~4~~ ~~2~~ 0
2	2	~~3~~ ~~1~~
3	3	~~9~~ ~~7~~ 5
4	3	~~4~~ ~~2~~ 0
5	5	~~11~~ 9

→ Maintains a queue [ ]

→ Whenever a new process enters, it goes to the back of queue

→ Whenever a process finishes 'q' sec of run, it is put at the back of the queue



[4, 1, 5, 2, 3]

Partial ✗

Starvation ✗

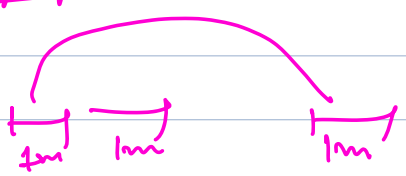
Simple ✓

Eg Party is happening at home

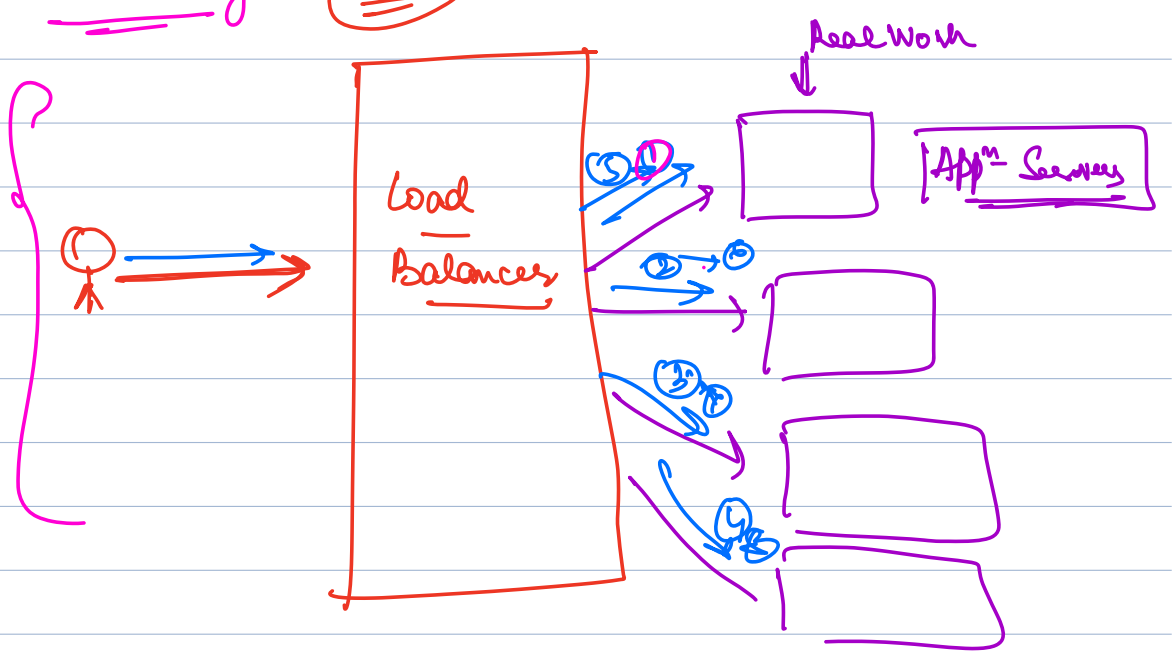
100 people

FCFS

SJF



Load Balancing ⇒ HLD



Balances the load to individual Servers

Throughput and Latency

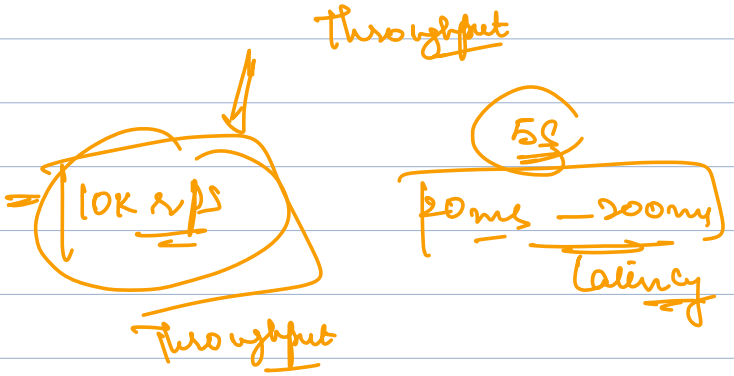
Throughput ⇒ (Efficiency) (Average) ⇒ # processes that are completed per unit time

100 procs ⇒ 20 sec ⇒ $\frac{100}{20}$ = 5 p/sec

Latency ⇒ Average time it takes for 1 process to complete

⇒ time for 1 process ⇒ Completion time − Arrival Time

⇒ measure of perf of an algo



Scaler System

① → 1 million users → Latency → 10sec

10sec

Throughput = 0.1 m user/sec

1 sec → 0.1 mill → 1 sec

2 sec → 0.1 m → 2 sec

3 sec → 0.1 m → 3 sec

4 sec → 0.1 → 4 sec

5 sec → 0.1 → 5 sec

better service

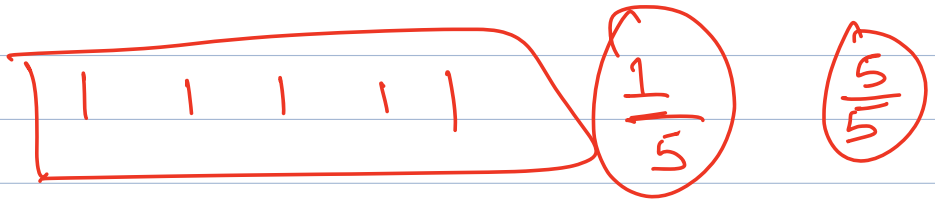
② → 1 million users →

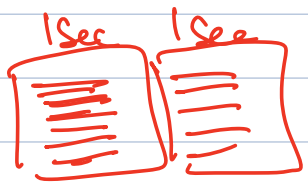
Throughput → 1 million → 0.1 sec

10

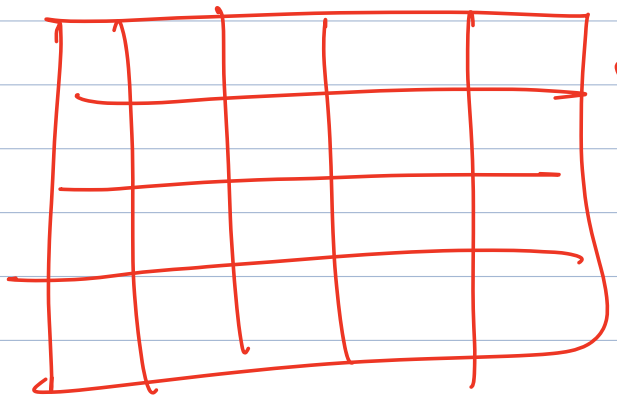
→ Latency → 5 sec

→ 10 sec





Eg Restaurant



50 people

→ 50 people
→ all make order
→ To serve all → 50 min
→ Throughput $\frac{50}{50}$ → 1 min

I ①

at 50th min
→ it serves
order of everyone

② II

→
5th → 10 min
15th → 5 order
20th → 10
50th → all

Lesser latency

5 orders

2 min 3 min 4 min 5 min 6 min

$\frac{2+3+4+5+6}{5}$ →

How to handle starvation

↳ result of priority

⇒ SJF ⇒

⇒ SRTF ⇒

⇒ Priority Scheduling ⇒ lower priority items are prone to starvation.

process id	arrival time	priority	comp
1	1	~~0~~	4
2	3	~~4~~ ~~5~~	2
⇒ 3	4	2	3
⇒ 4	5	~~5~~ ~~6~~	9
5	5	~~2~~ ~~3~~	1

highest priority → 0 → 10

lowest priority ⇒ Starve

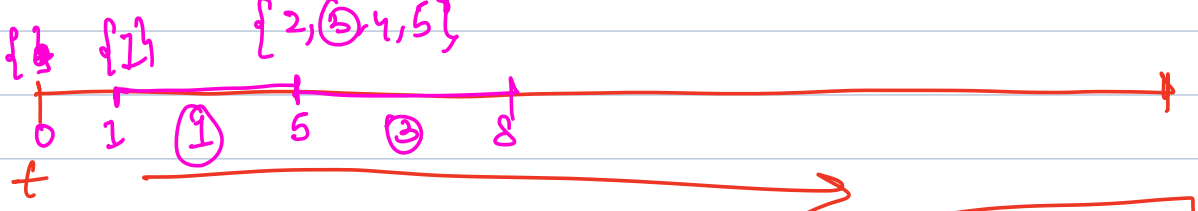
⇒ Aging ⇒ every time a CPU Scheduler runs:

① it will select one process

② increase priority of every other process

⇒ If multiple with same priority ⇒ FCFS

{2, 5, 4, 5}

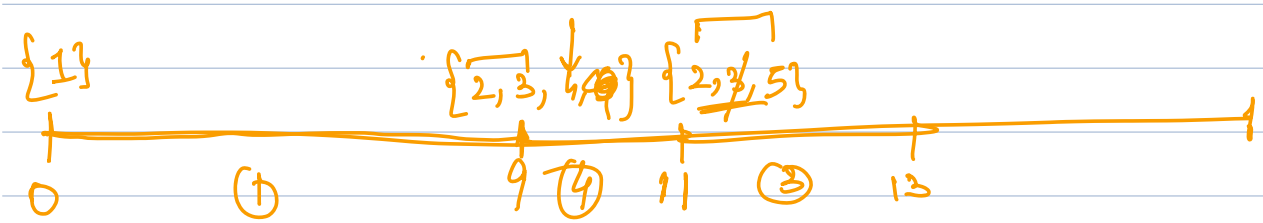


⇒ | priority | + | order |

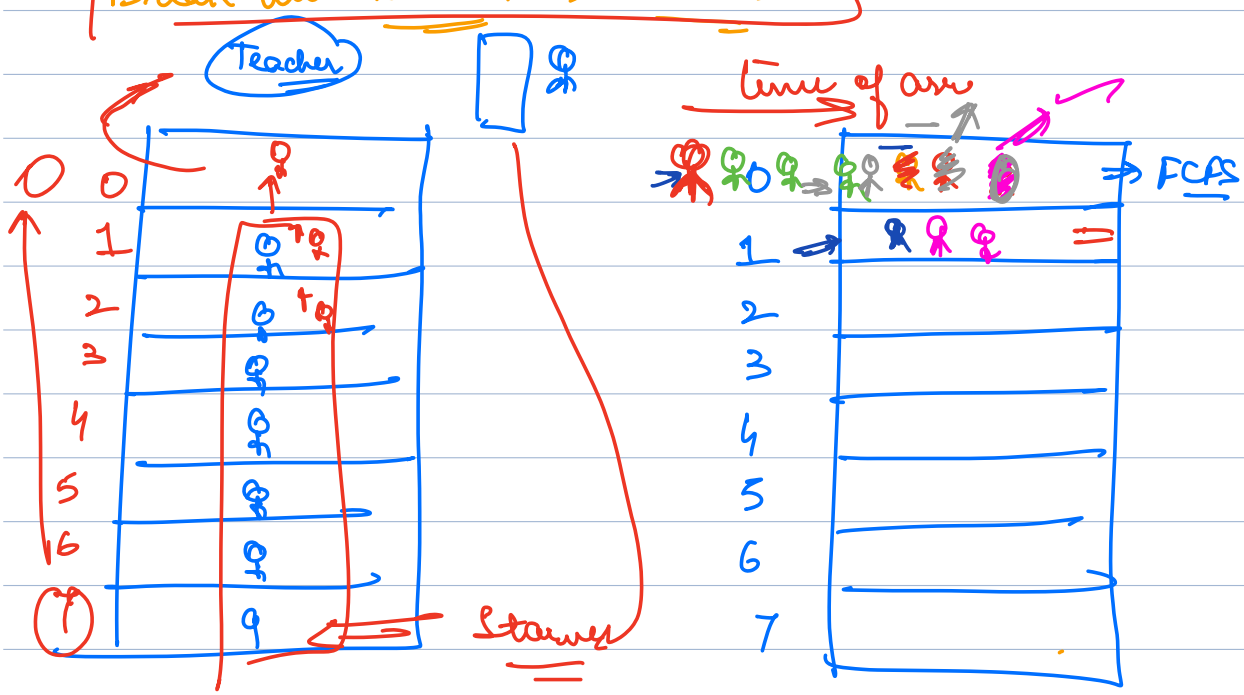
upper bound

processes

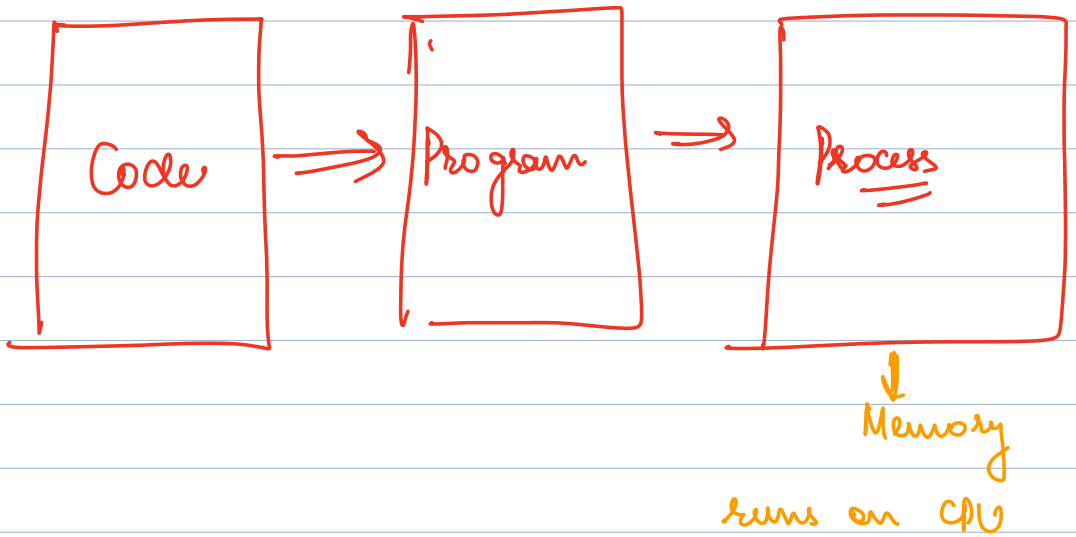
id	arrival time	priority	time to comp
1	0	0	2
2	5	3 2 1	2
3	8	2 1	2
4	9	1	2
5	11	0	2



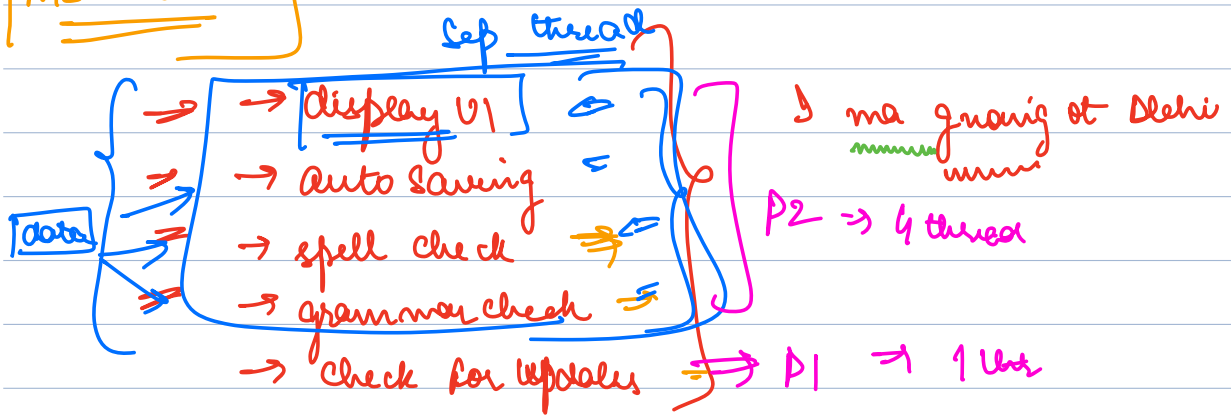
Break till 10:35 PM ⇒ THREADS



THREADS



MS-Word =

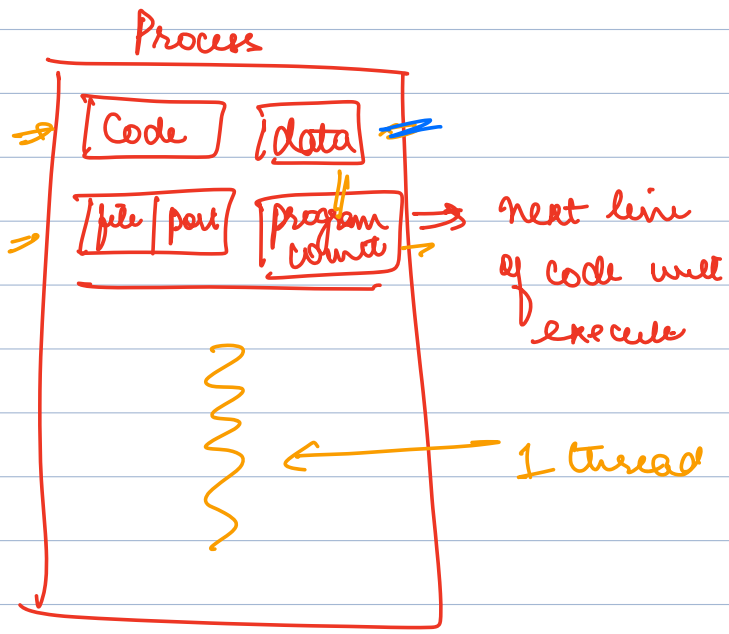


Thread

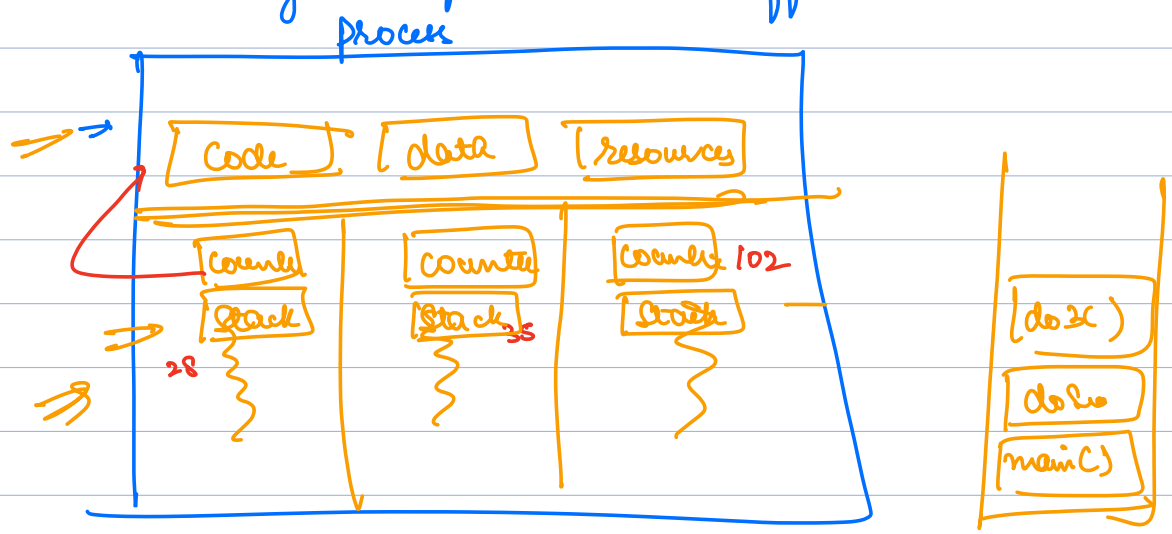
- ⇒ Basic / fundamental unit of CPU exec^{n}
- ⇒ CPU executes only threads
- ⇒ whenever anything is running on CPU ⇒ it is a thread
- ⇒ Process will atleast have 1 thread

```
main() {

    print (Hello);

}
```

→ main Thread

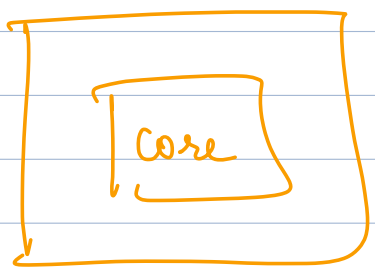


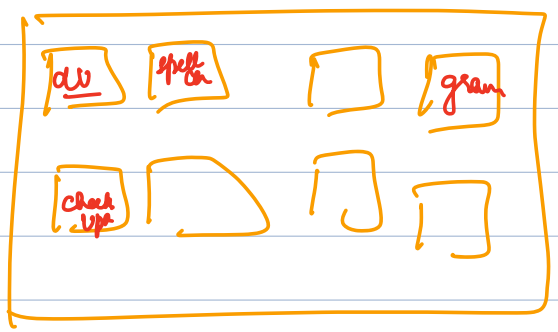
Because each process has its own data, data sharing b/w processes is difficult.

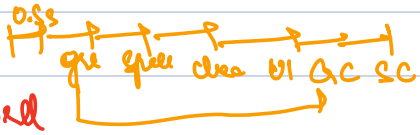


⇒ all threads share data of the parent process

⇒ every CPU core is executing a thread
⇒ thread scheduling

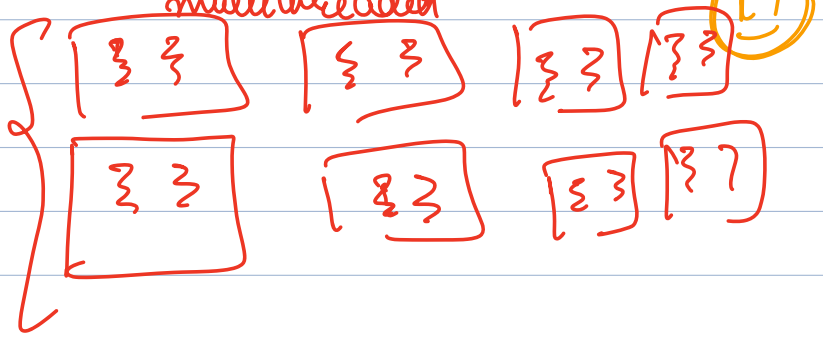






MS Word

multithreaded



1998
→ 4 core
→ 6 core
→ 8 core

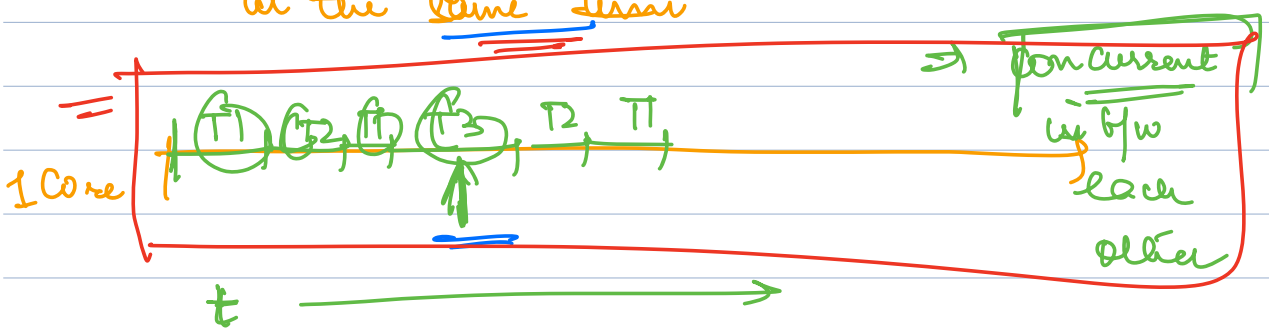
8 core → 16 Hyper Threading

CONCURRENCY V/S PARALLELISM

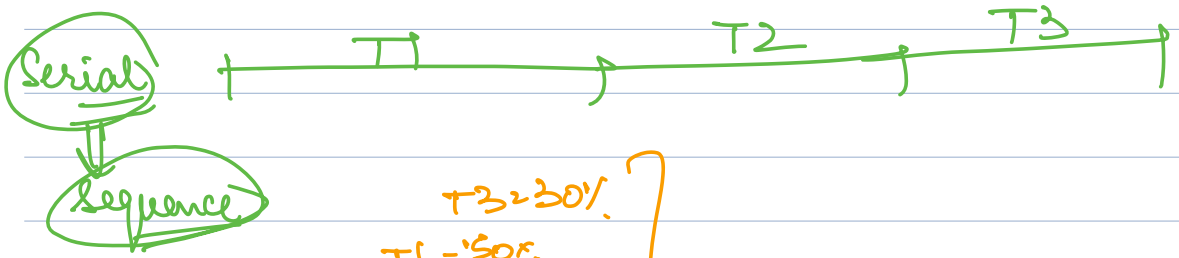
CONCURRENCY

→ Multiple threads at the same time in different stages of execution

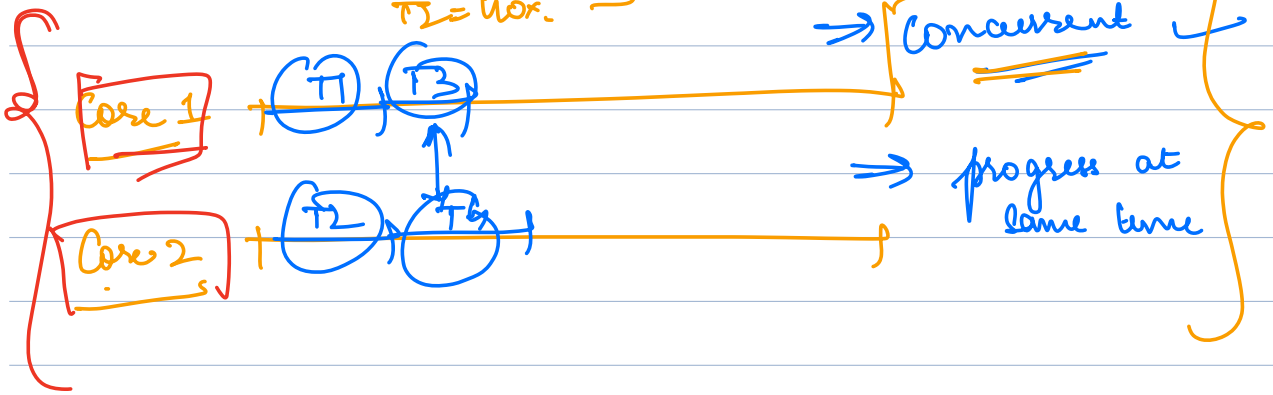
→ They may or may not be making progress at the same time



V/S Serializable ⇒ tasks happen one after other



T3 = 30%
T1 = 50%
T2 = 40%



PARALLELISM

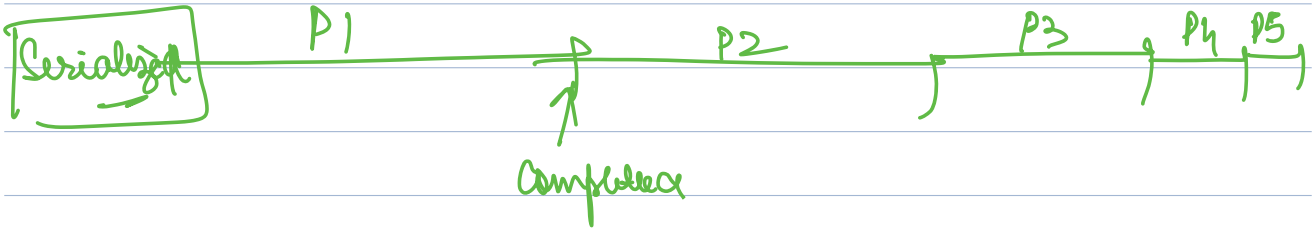
⇒ a system where more than 1 task can make progress at same time

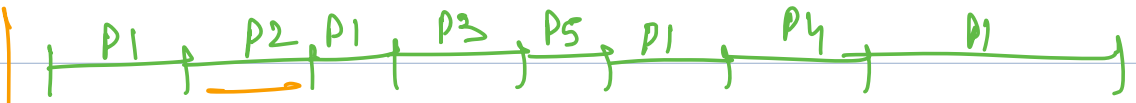
Concurrent system may not always be parallel

Parallel System ⇒ always concurrent

Single core ⇒ can't be parallel
⇒ can be concurrent

P1, P2, P3, P4, P5





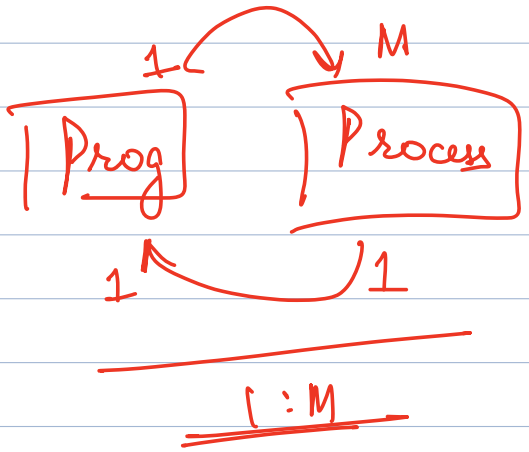
Concurrent ⇒ More than 1 task in diff stages of completion

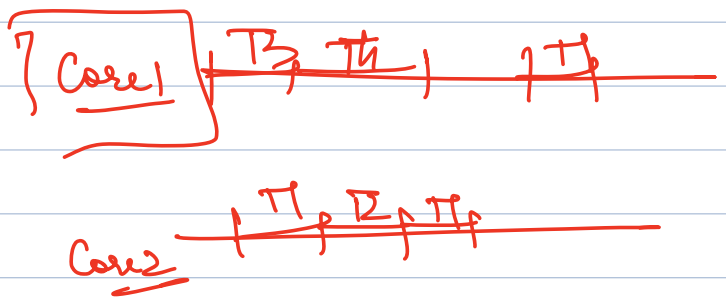
⇒ But at one time 1 or more than one like entry

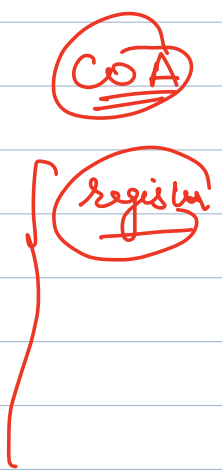
Parallel ⇒ more than 1 making prog

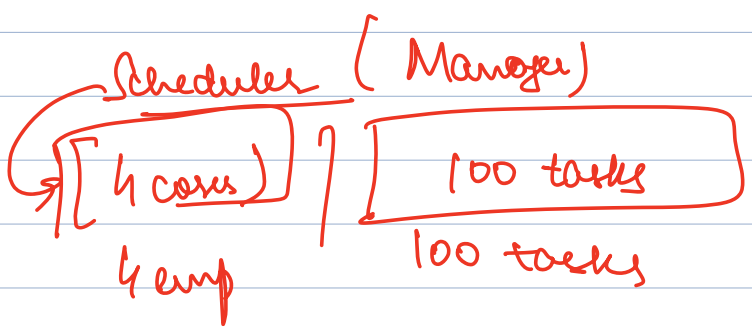
→ CPU = 4 cores → 2.8 GHz =

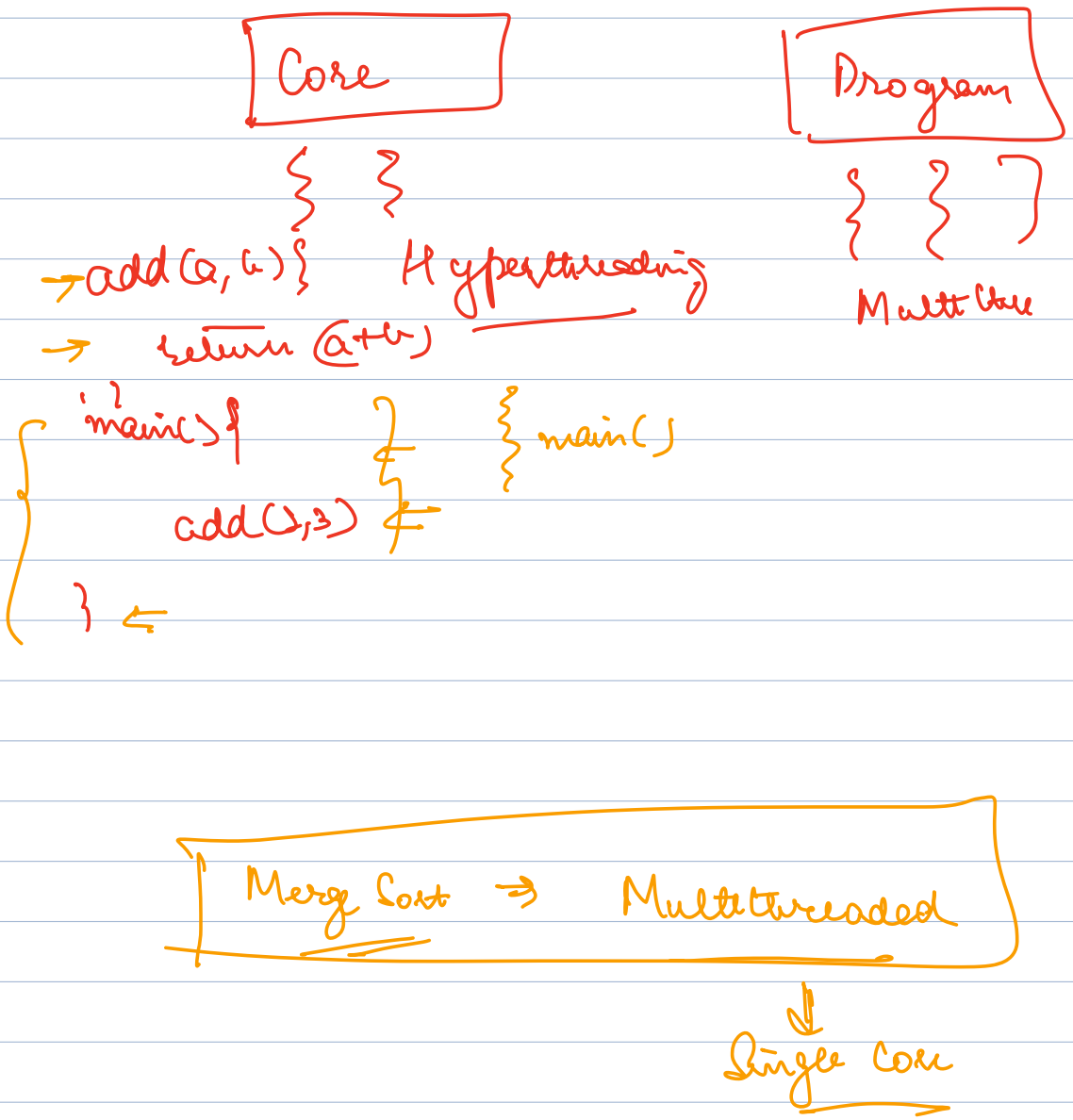
GPU = 1000 cores → 400 MHz ← Gaming


1
M
Prog
Process
1
1
1 : M


Core1
T3
T4
T5
Core2
T1
T2
T6


CoA
register


Scheduler
(Manager)
[4 cores]
[100 tasks
4 emp
100 tasks



Merge Sort ⇒ Multithreaded

↓

Single Core