OPERATING SYSTEMS - 2

→ FCFS → Non Pre
→ SRTF → Pre

Agenda

→ Starvation in CPU Scheduling

→ Round Robin CPU Scheduling

→ Threads → Intro to threads

→ Multicore v/s Single Core

→ Concurrency v/s Parallelism

→ Practical demo of thread

Java → ⇒ Print #s 1 to n each via a separate thread

I. Starvation

PID	Arrival Time	Burst Time
1	0	20
2	1	3
3	2	4
4	3	1
5	4	2
6	7	3

1 2 3

after PID 1 we keep getting processes with very very short burst time.

→ PID 1 will be starving for resources

→ PID 1 is hungry for CPU but because of shorter processes it's unable to get CPU

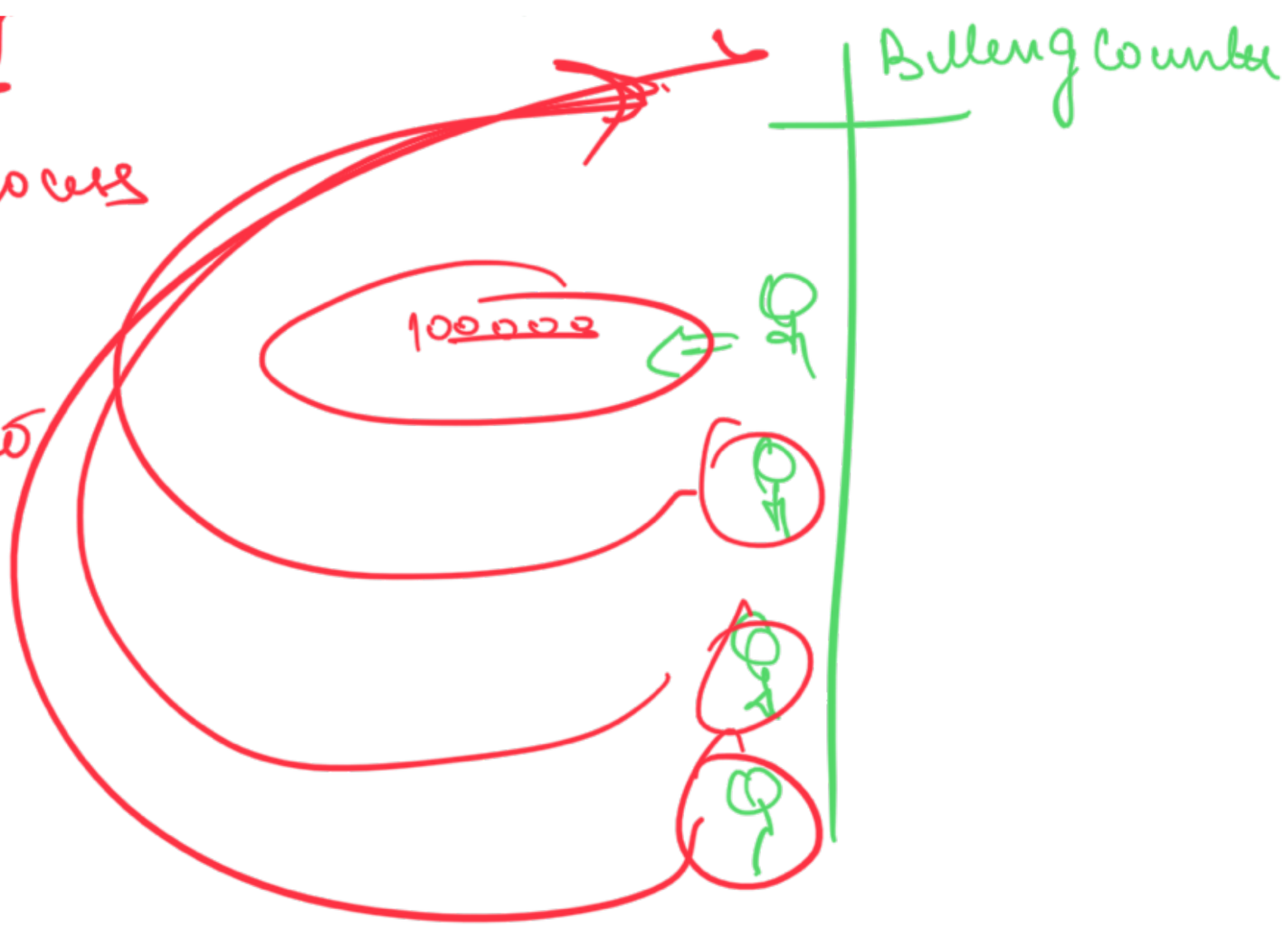
→ Starvation

Supermarket

Scheduling

Priority Scheduling

a lower priority process might have to wait too much to get CPU



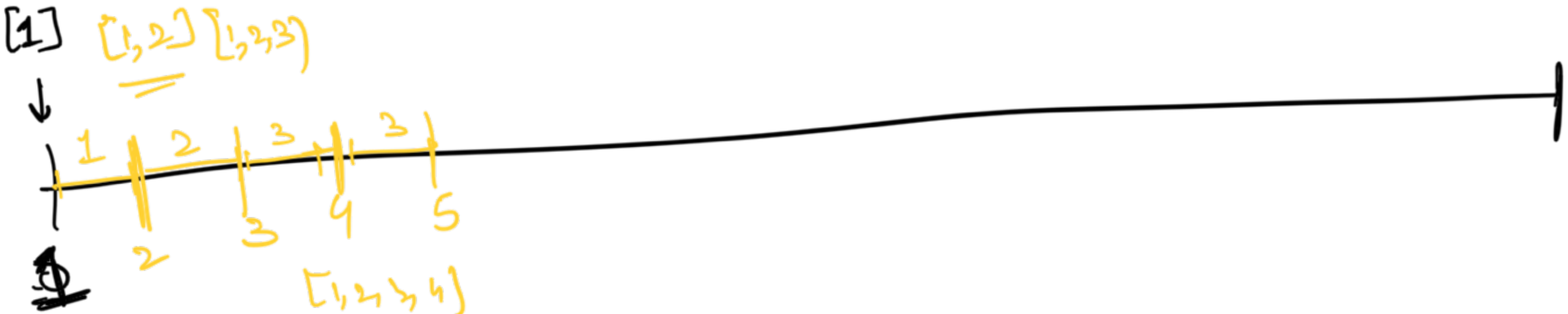
Sol^{n} to Starvation in PS

every time a process is denied a resource its priority by 1.

Processes

PID	Arrival Time	Priority	Burst Time
1	1	7	~~2~~ 1
2	2	3	~~2~~ 1
3	3	2	~~2~~ 1
4	4	4	2
5	5	3	2

P0
P1
P2



Processes

PID	Arrival Time	Priority	Burst Time
1	1	6	1
2	2	0	1
3	3	1	2
4	4	1	1
5	5	2	1

[1] [1,2,3] [1, 2, 3, 4, 5)

1 2 3 4 2 5 6 4 8

1

[1, 2] [1, 2, 3, 4]

⇒ we increase the priority by 1 till it becomes ⇒ (1)

Q^n ⇒ If every 1's we are checking which process should run next

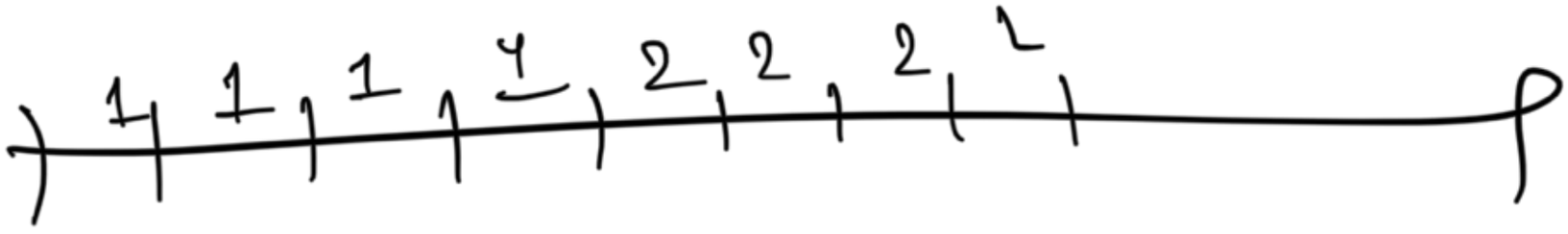
PID	Priority
1	~~20~~ ~~19~~ ~~18~~ 15 ⇒ (1)

How long at max will PID 1 have to wait

⇒ We have created an upper bound on an amount of time one process has to wait

PID	Priority
1	~~10~~ 9 ← 1 ≤
2	~~10~~ 9 ≤

PID	Priority
1	1
2	1



(1) Starvation ⇐

(2) Algos that can lead to starvation

→ SRTF

→ Priority Scheduling

Fix :

Inc priority of other everytime they don't get picked till their prio reaches 1

(1 - 10)

~~HIGH_PRIORI.~~

HIGH_PRIOR

LOW_PRIOR

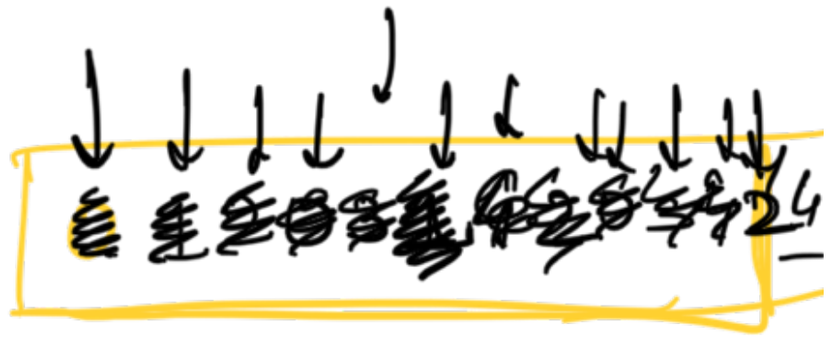
1:1 Mapping

Linux Kernel Development

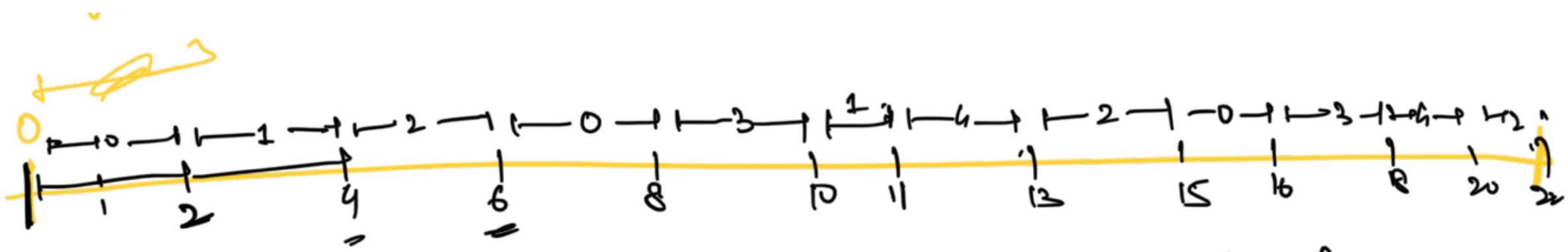
Round Robin Policy for CPU Scheduling

q = Time Quantum = 2 sec

PID	Arrival Time	Burst Time	Remaining Time
0	0	5	5 3 1
1	1	3	3 1
2	2	6	6 4 2
3	3	4	4 2
4	5	5	5 3 1



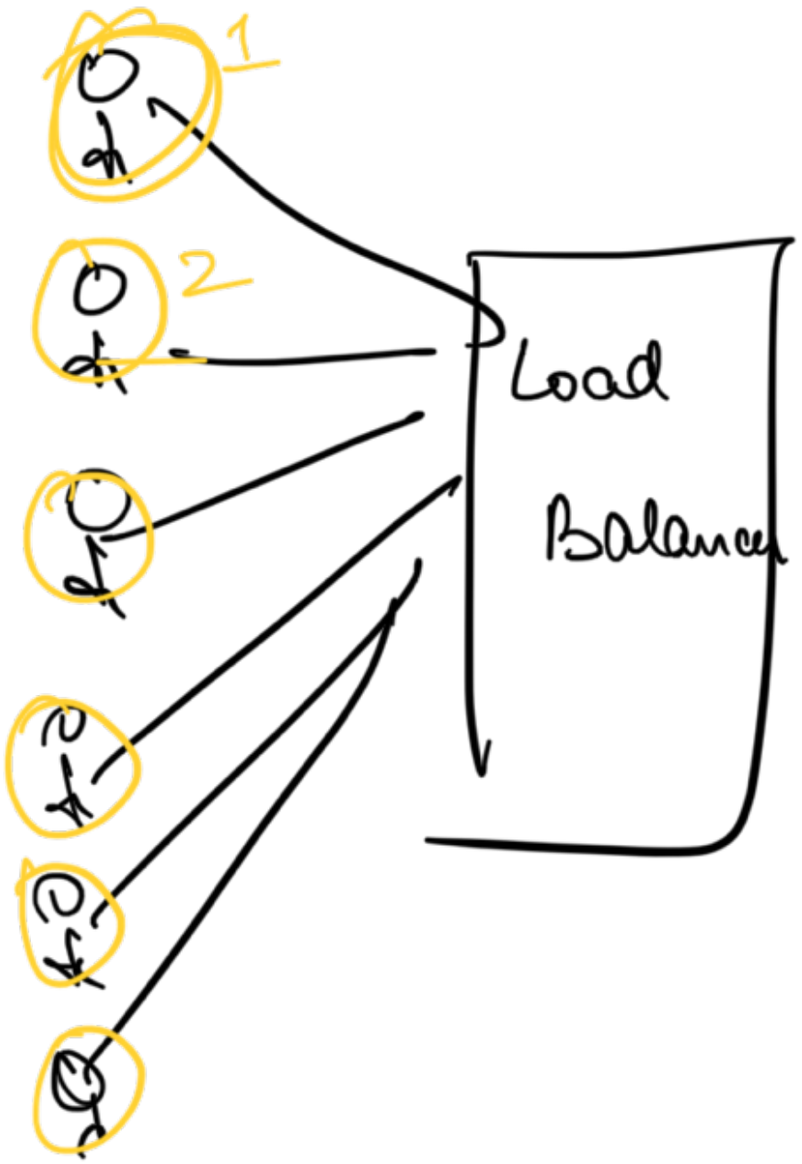
I will give any process max time of q. After that I will allot CPU to other process in cyclic order



New Process goes at the back

Pick New Process from front

Current Scheduled process gets added at the end after q Sec

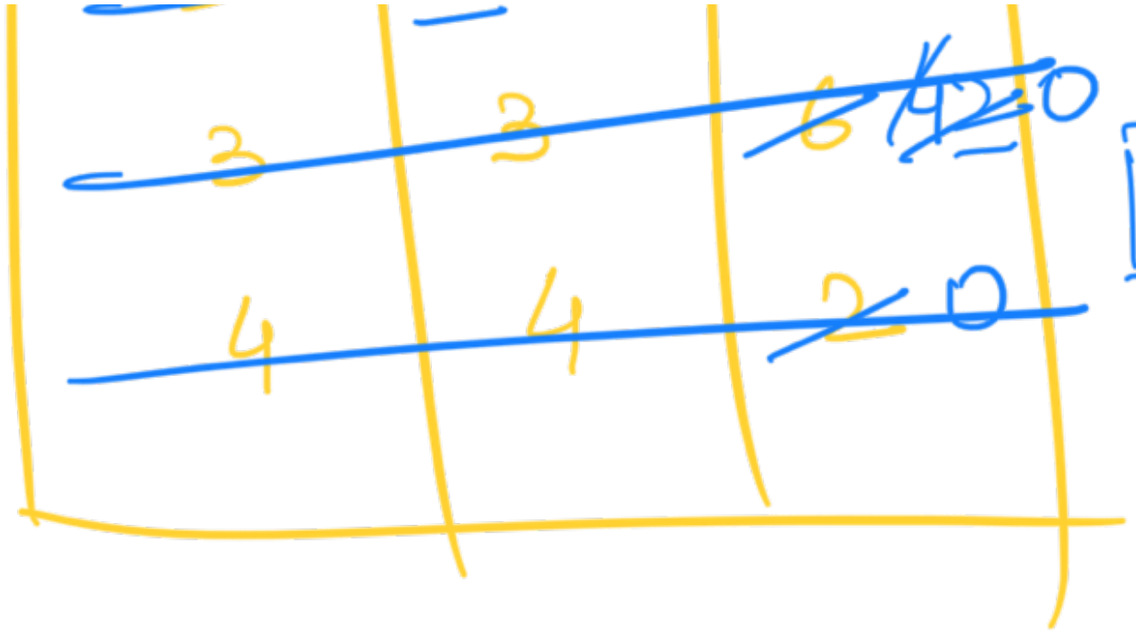


PID	Arrival Time	Burst Time
1	1	~~5~~ 3 ~~1~~ 0
2	2	~~3~~ ~~1~~ 0

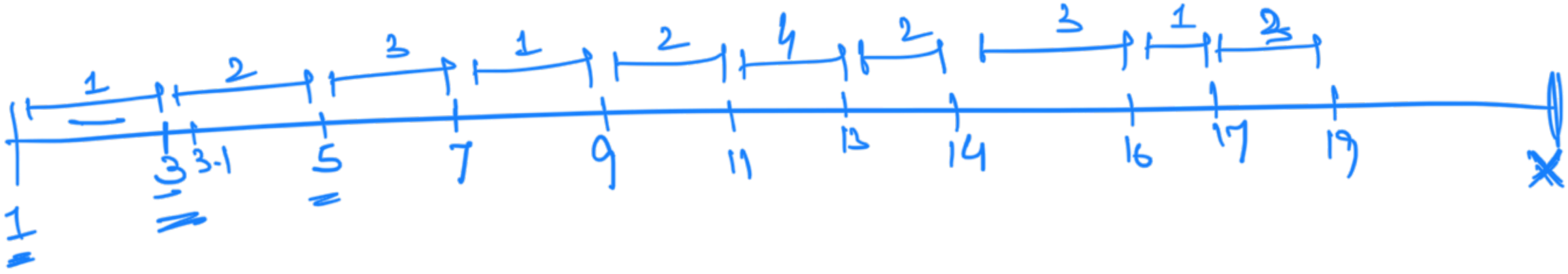
q ⇒ 10 sec ⇒ FCFS

q = 0.001 s

1



q should be neither too
less nor too big

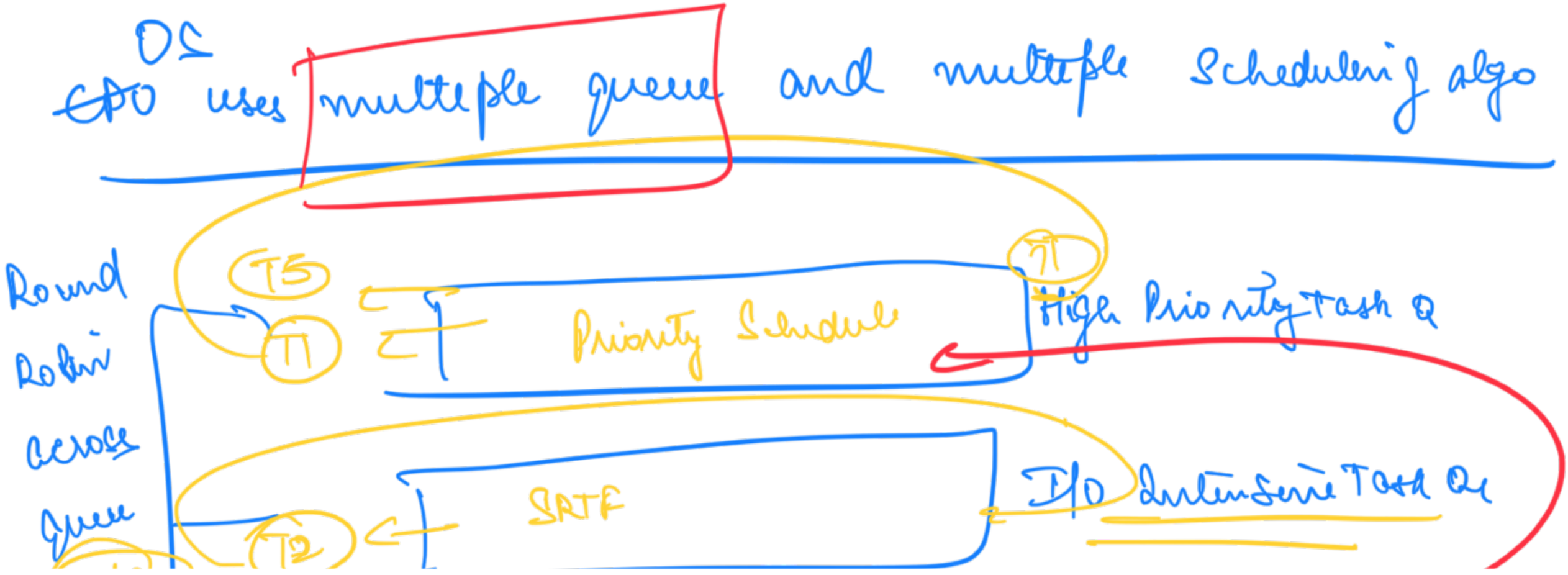


Assumption

1. No I/O / Interrupt
2. No overhead for context Switch

OS

CPU uses multiple queue and multiple Scheduling algo



Class CPU Scheduler {

private Map<String, Queue> ;

private int q = 2 sec ;

{P, T} Schedule() {

{P1, 2}

LT: P5 P6 P7 P8

1 5 10

RR 2s

HPT: PQ: 1 2 3

LPT: FCFS: 4 5 6

ST: RR: 8 9 7

1 1,2

2 4,2

3 7,3

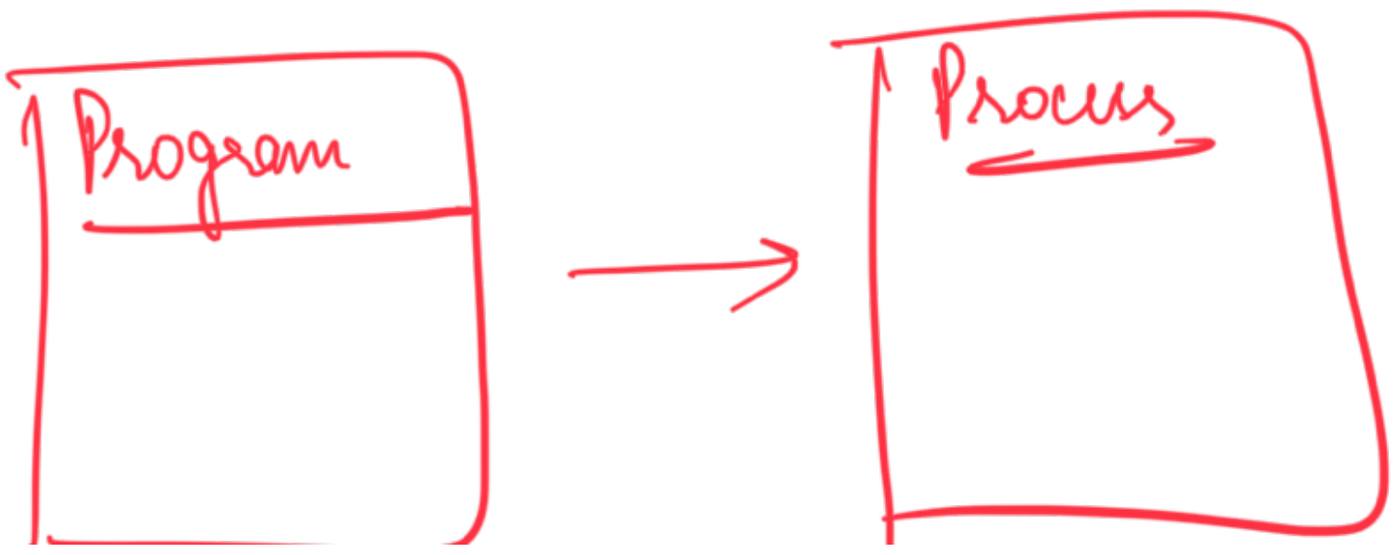
10^6 < 1s

10^9 1s

Threads

Intro to Threads & Concurrency

Threads



Doing multiple things at the same time:

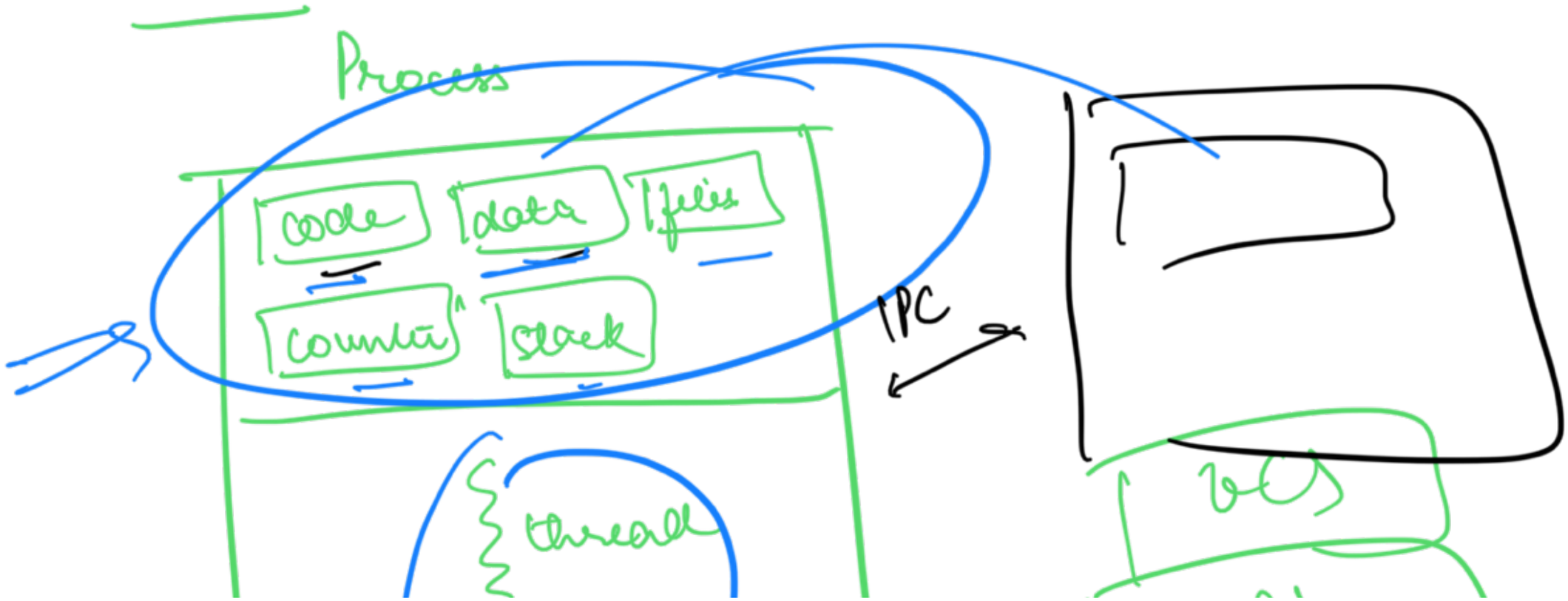
1. Taking input & displaying it
2. Spell Check
3. Grammar Check
4. [illegible]

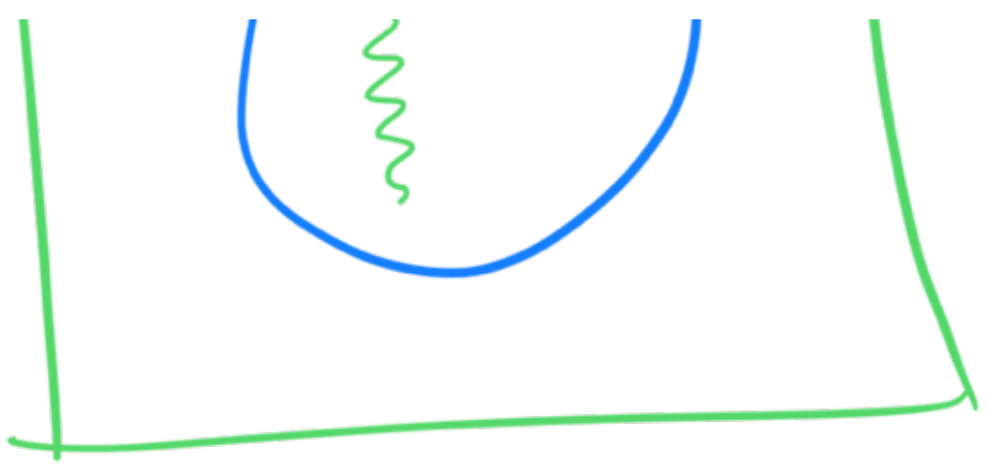
⑤ Autosave

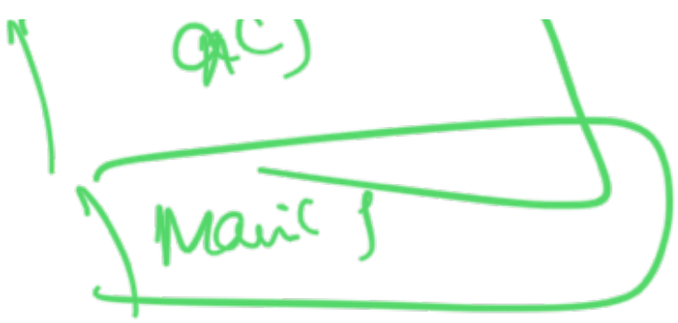
⑥ Check for Updates

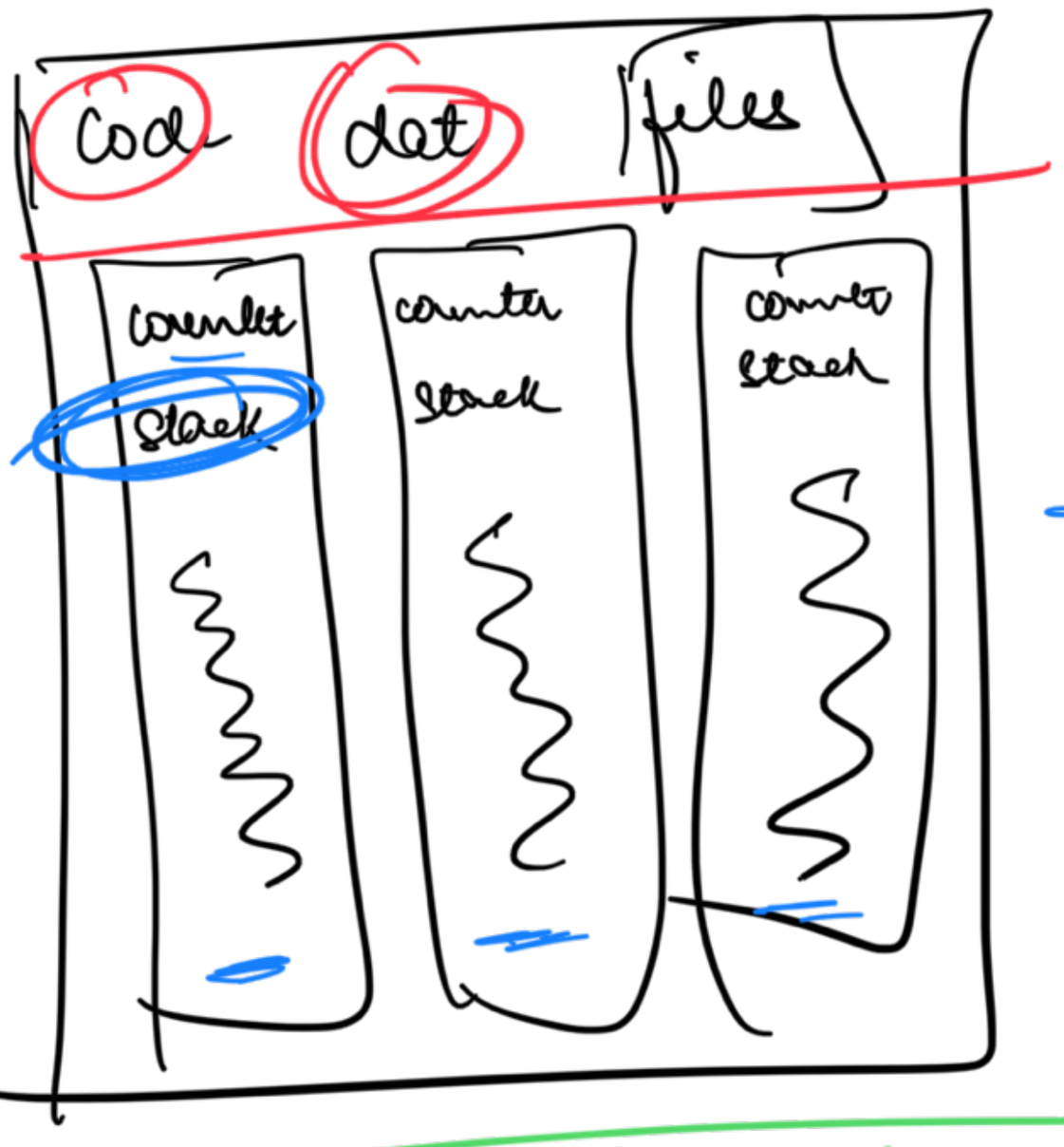
Threads : Basic unit of CPU

Execution / Utilization









Process v/s Thread

→ data sharing: IPC is difficult
data is shared by default

→ Size: Creating a Thread is faster
↳ lightweight process

Core: | computing core

Octa Core CPU

64 sec

1 thread

64 s

2 thr

32 se

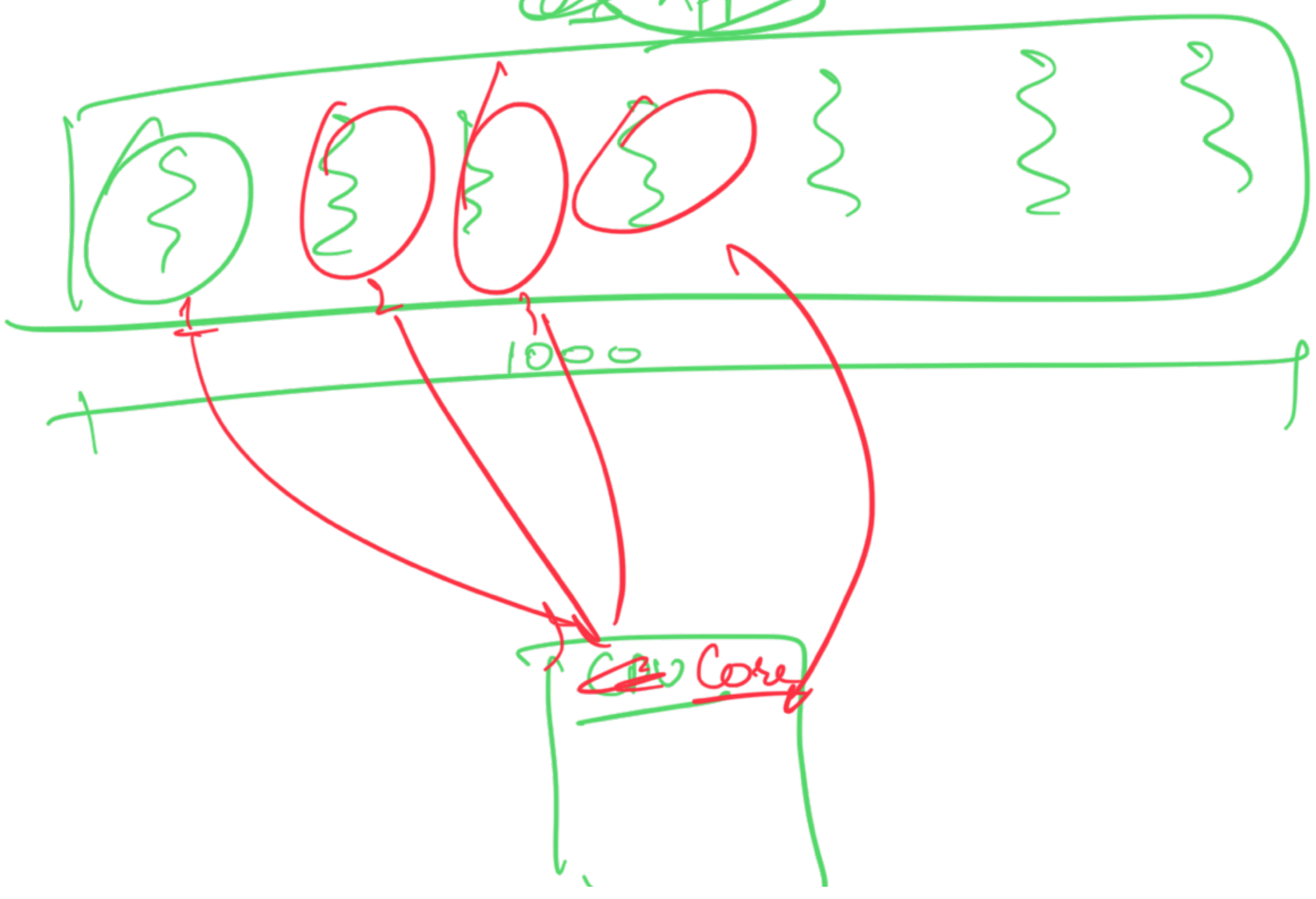
4 thread

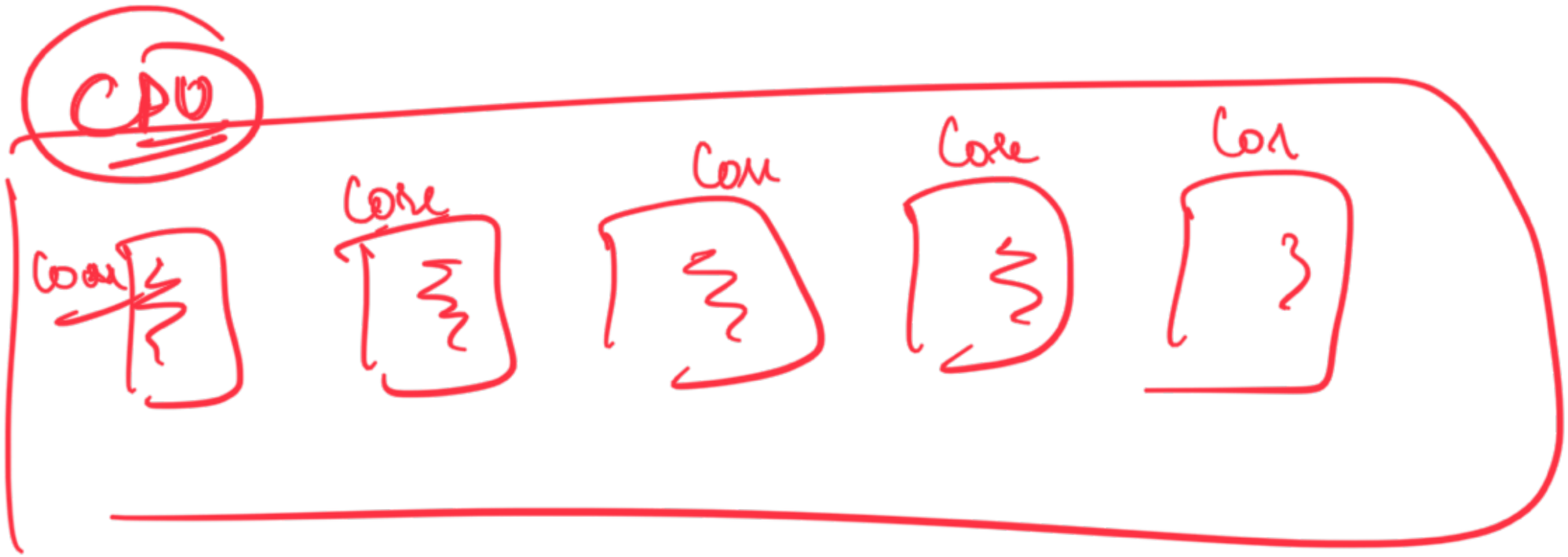
16

8 s

16 thread

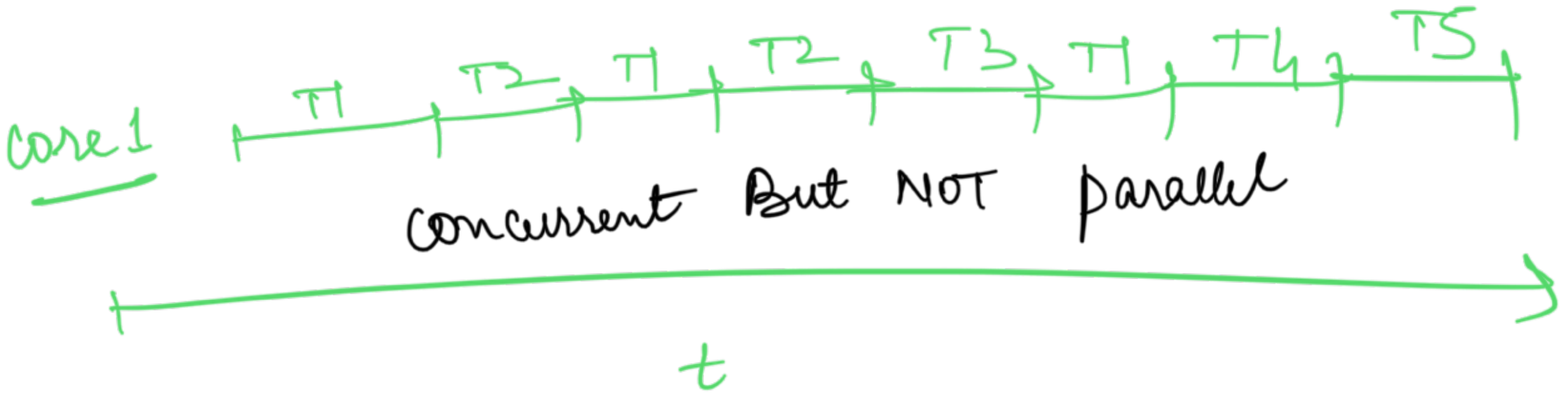
App




CPU
Core
Core
Core
Core
Core
Unit
GPU
Games

Concurrency v/s Parallelism

Imagine a Single core CPU Batch Multiprogram System



more than one task is at diff state of execution but not necessarily making prog at the same time

Parallel: Multiple tasks making progress at the same time

Is a parallel system Concurrent ?

Yes

Vice Versa True ?

Threads (Next Class)

→ Node a multithreaded app/m

Project Build

→ Executor Framework

→ Thread Pools

→ Print N #s using diff Threads

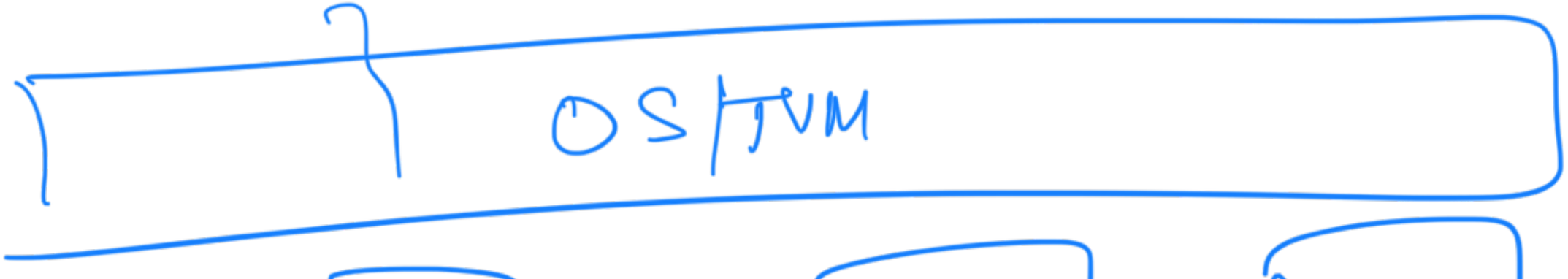
→ Merge Sort using multiple threads.

Next to Next Class: Sync Problems

Conc: Can you eat multiple food items -


Rice
Ram


Logical
JVM


OS/JVM


Physical

Flask

Python Lang Modules

Frameworks

Data Eng

Python

Asyncio

Kotlin Coroutine

Project Coon

Go routines

Saiel =

Nov Intermedia

7 weeks - 8 weeks

→ Google Docs
- ↳ CRDT (Conflict free Replica
- ↳ OT (Operational transf)

4 classes