Intro to MultiThreading and Concurrency

2 weeks

Agenda

1. Intro to threads, processes
2. Intro to concurrency, parallelism
3. How to design a multithreaded program
4. Create a simple thread ⇒ HelloWorld
5. Executors, Thread Pools ⇒ Number Printer
6. Callables and Futures ⇒ Merge Sort
7. Data Synchronization Problem ⇒ Adder Subtractor

Practical

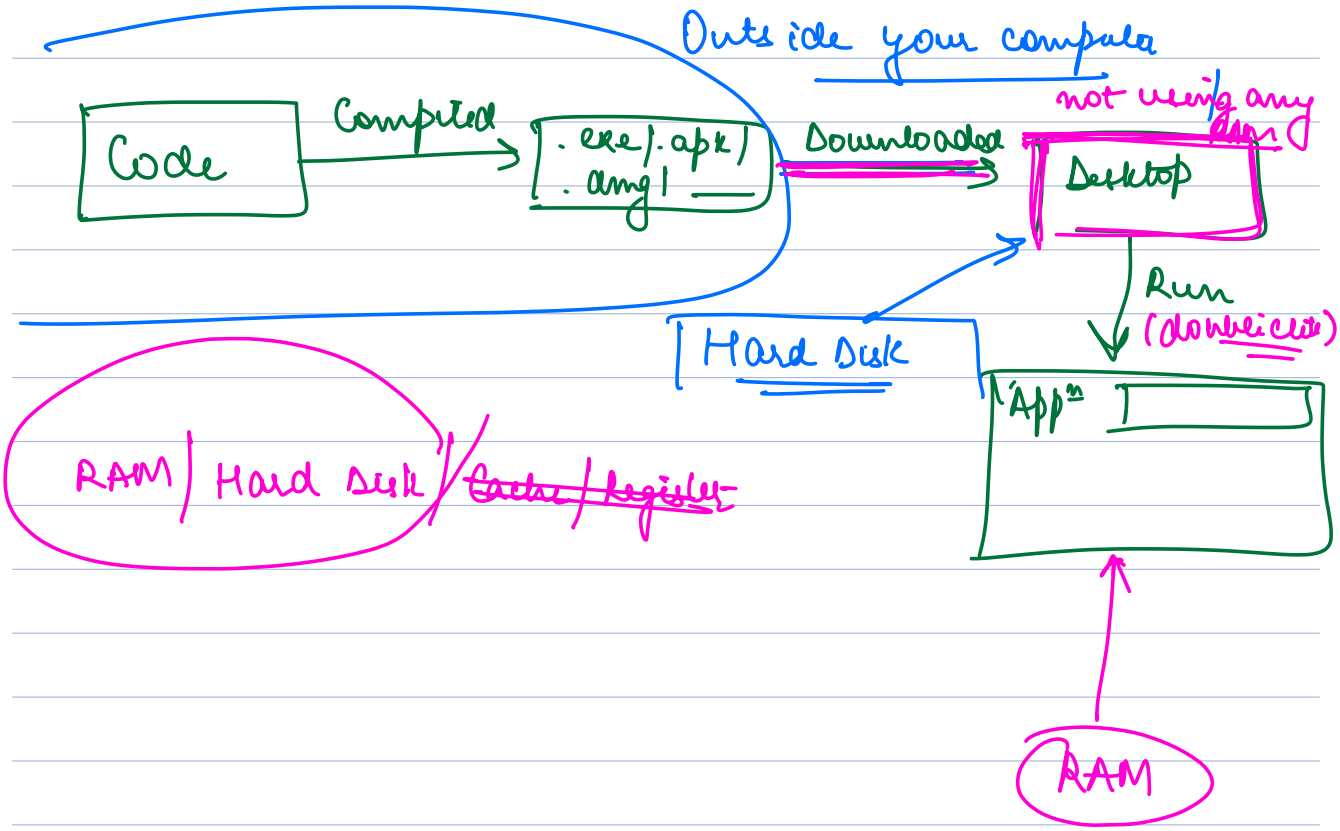
Not going to cover

- → Mutex / locks / Semaphores
- → Fork / Join
- → Atomic Data Types
- → Concurrent Data Structures
- → Completable Futures / Barriers

Intro to processes and threads

journey of Google Chrome on laptop

0. Downloaded (.exe, .dmg, .apk)
1. Installed
2. Opened



→ Disk → Double Click → RAM (Memory)

→ Hard Disk
→ SSD

→ Permanent
→ 1 TB
→ Slower

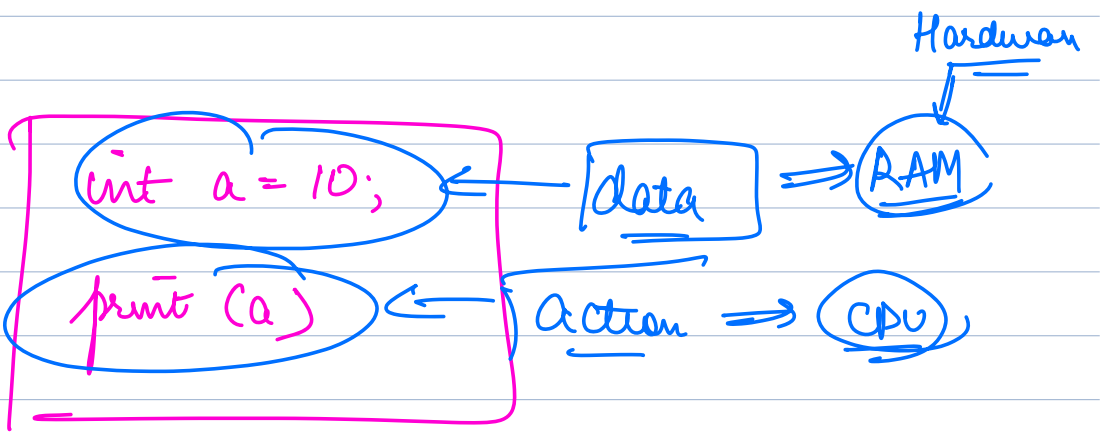
→ Volatile
→ 16 GB
→ Faster

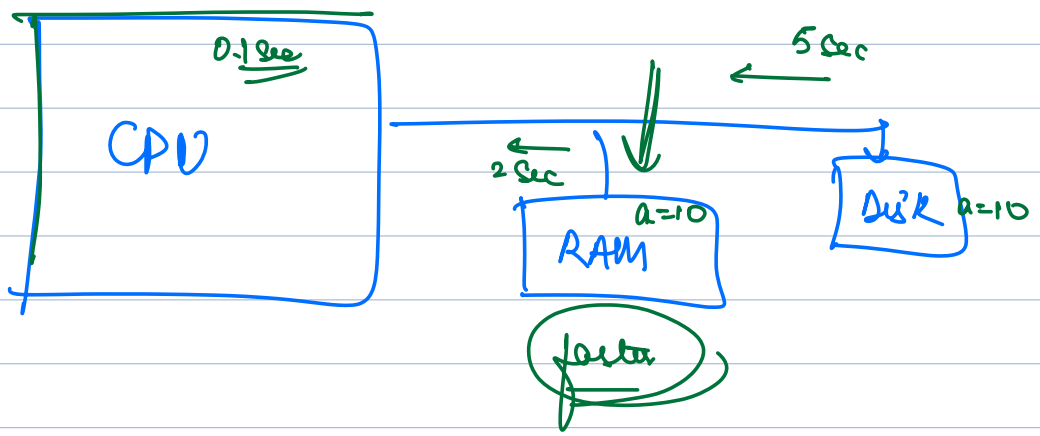
CPU → Central Processing Unit

→ Every code runs on CPU

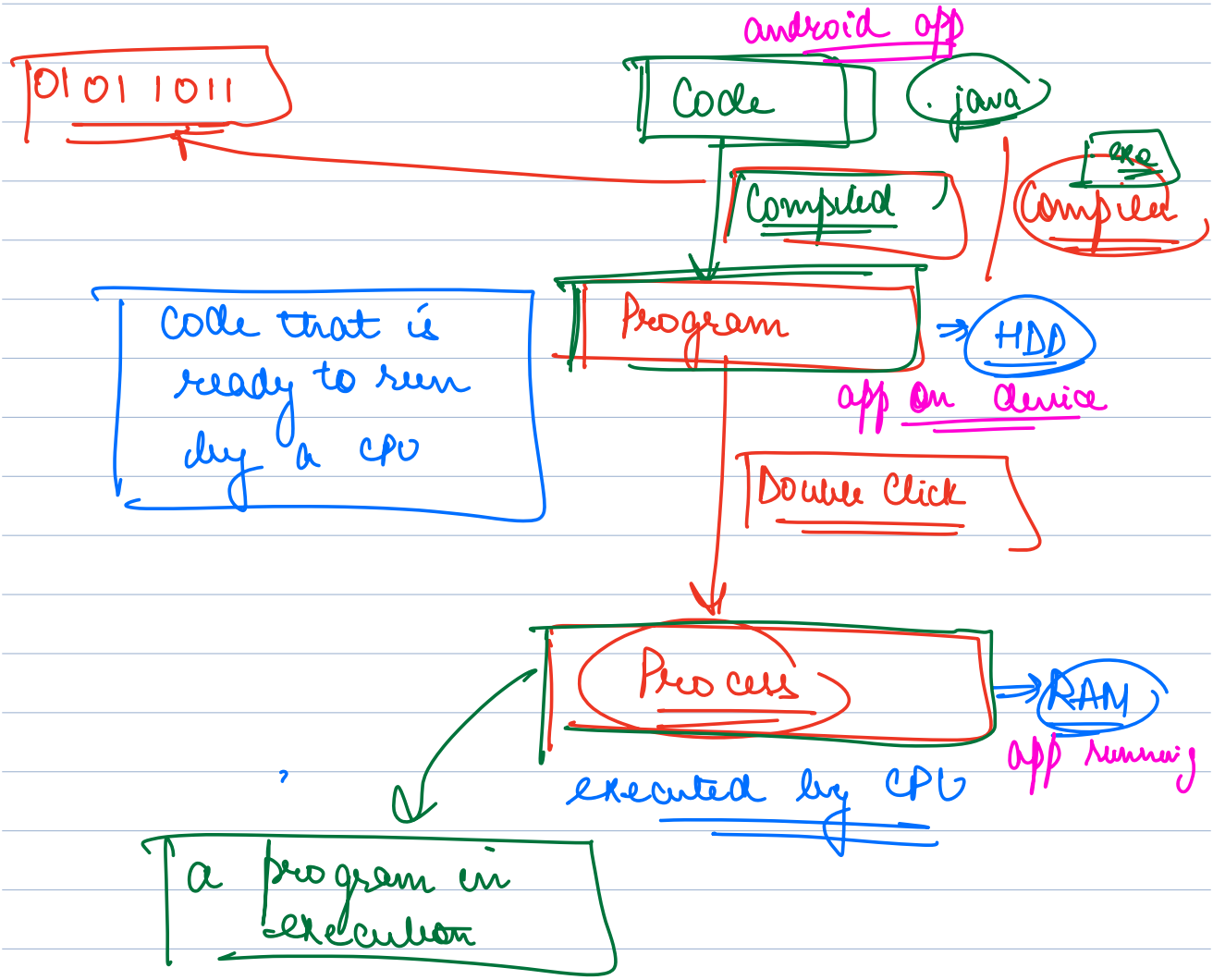
Code

Action
(to do something)
(behaviour)

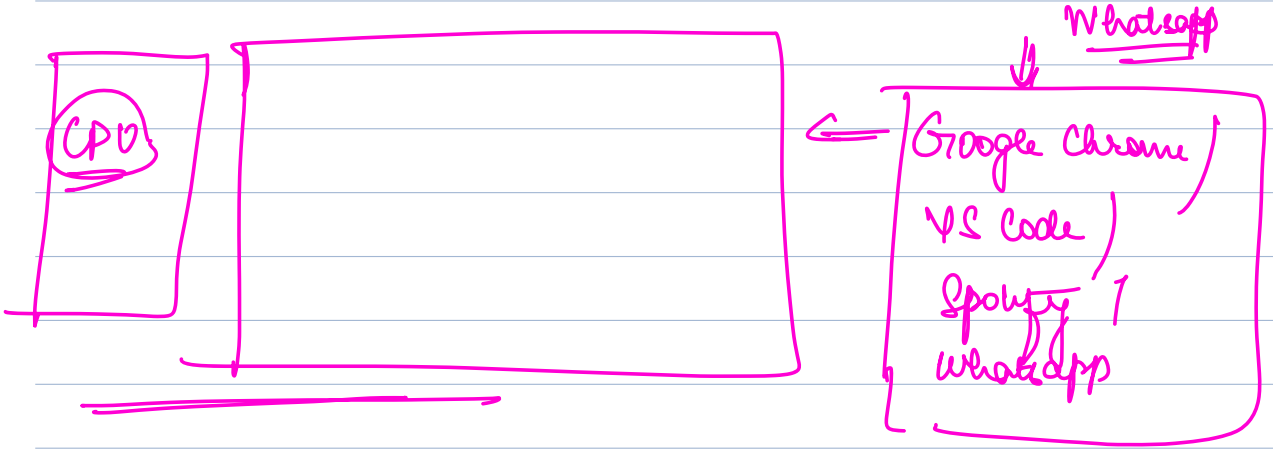




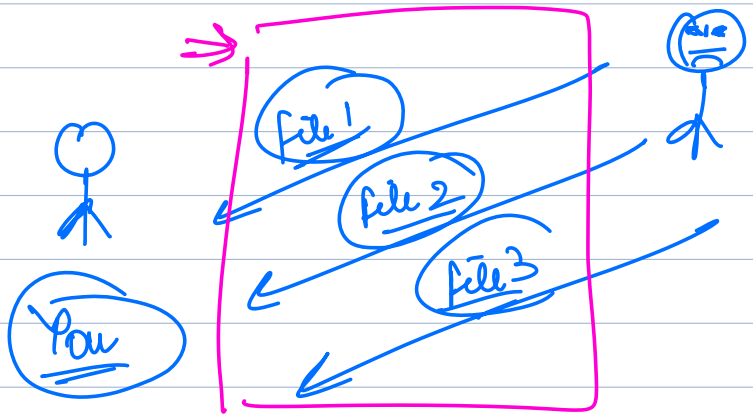
work ⇒ 2 mins
info ⇒ 60 mins
→ 62 Mins



⇒ Process → some work for the CPU to do



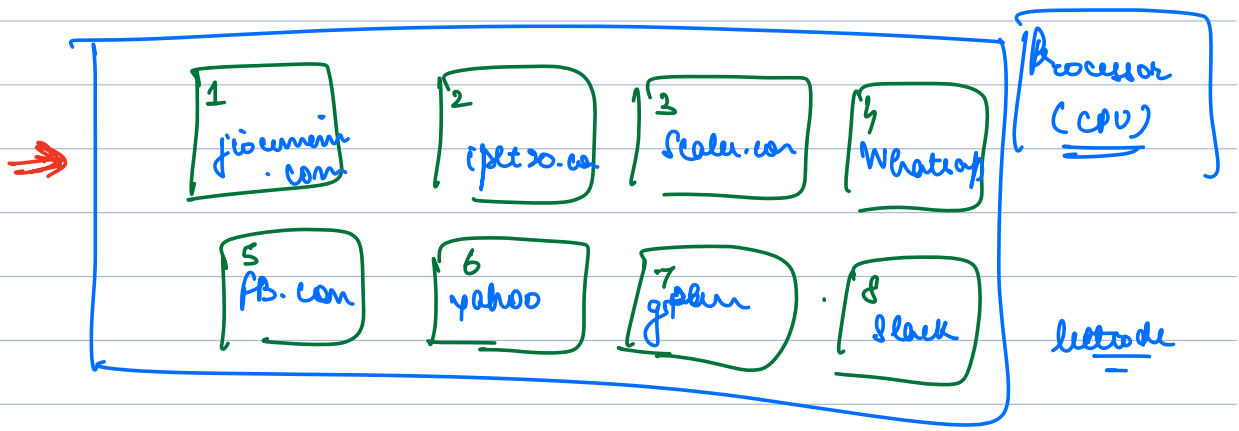
CPU - Process V/S Employee - Task

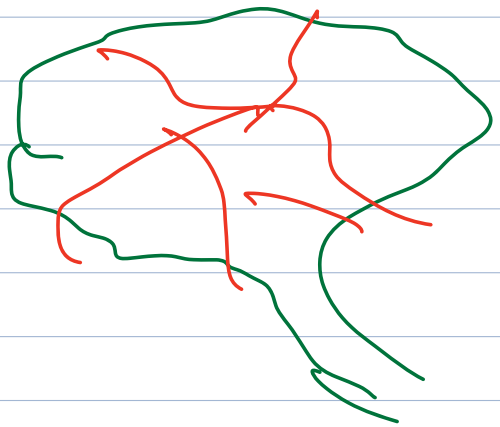


Brain ⇒ One Thing at a time

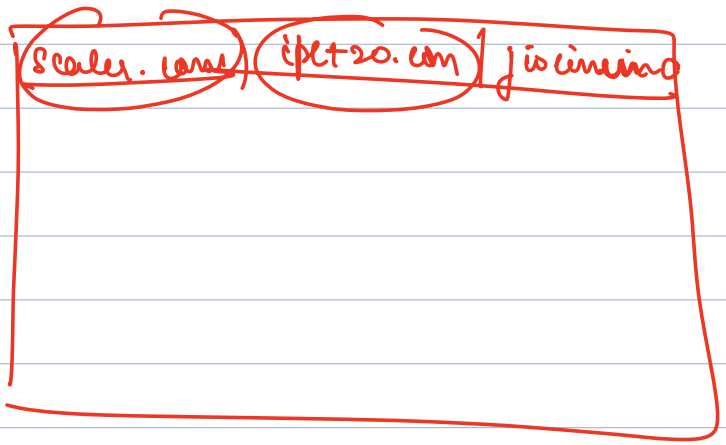
CPU ⇒ only 1 thing at a time

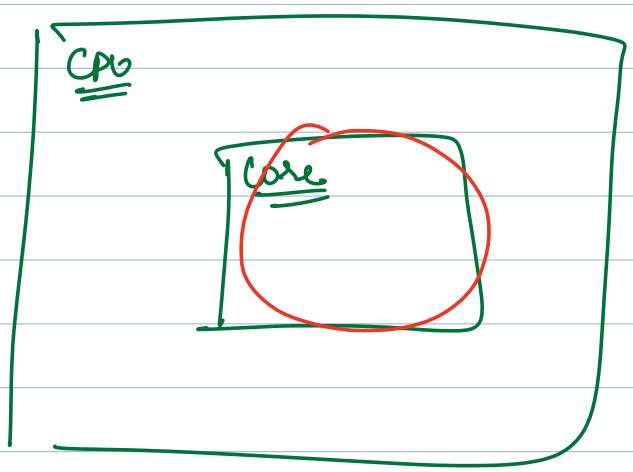
↳ CPU Core ⇒ Real Brain of a computer

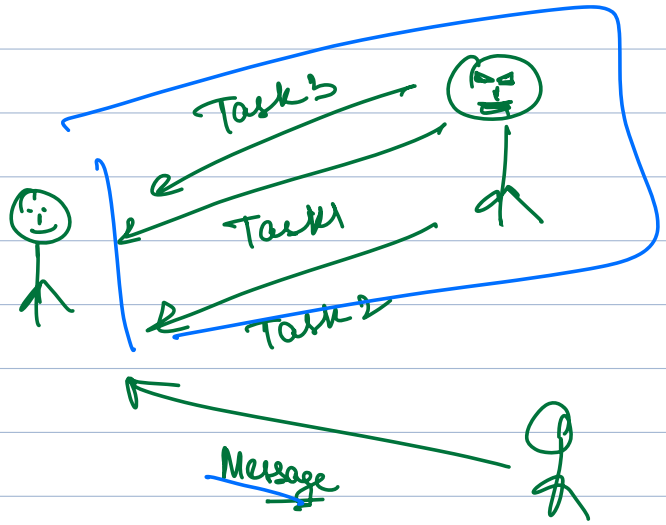




Core ⇒ Independent Brain
⇒ Execute 1 thing at a time







cores < # tasks

↳ Context Switching

2.4 GHz ⇒ 2.4 x 10^9 cycles/sec

Concurrency v/s Parallelism

① Multiple Tasks

Serial
Simple System

SITUATION 1 → 1 core / very focussed / only one thing at a time

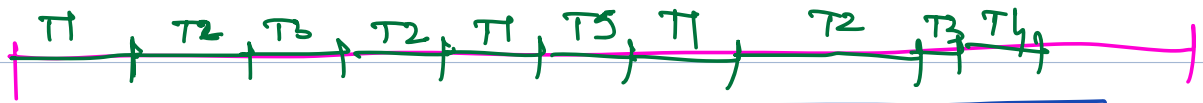
⇒ T1, T2, T3, T4, T5

how many tasks are in process at 1 time
⇒ Only one

Concurrent

SITUATION 2 → 1 core / distracted / multiple tasks half done at one time

T1, T2, T3, T4, T5



how many tasks in process at one time ⇒ >1
how many tasks actually making prog at ⇒ 1

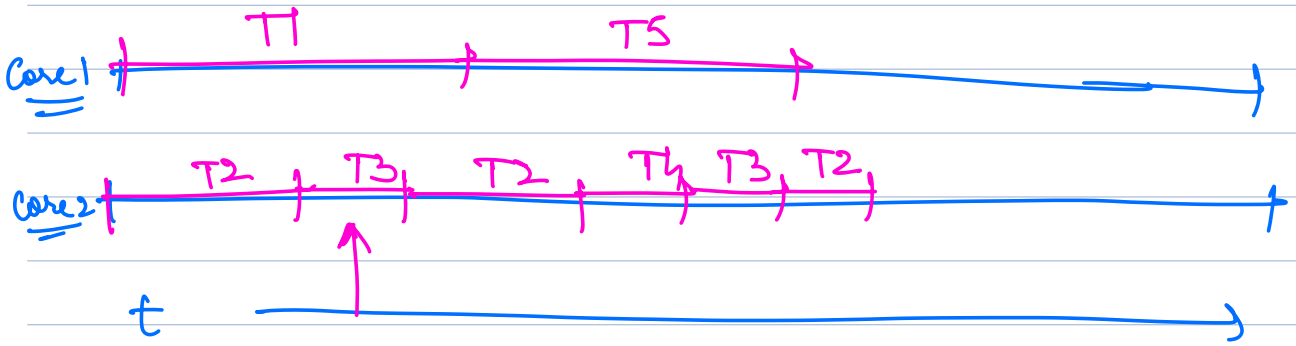
w team

Parallelism & Concurrent

SITUATION 3

Multi Cores / focussed or Distracted

T1, T2, T3, T4, T5



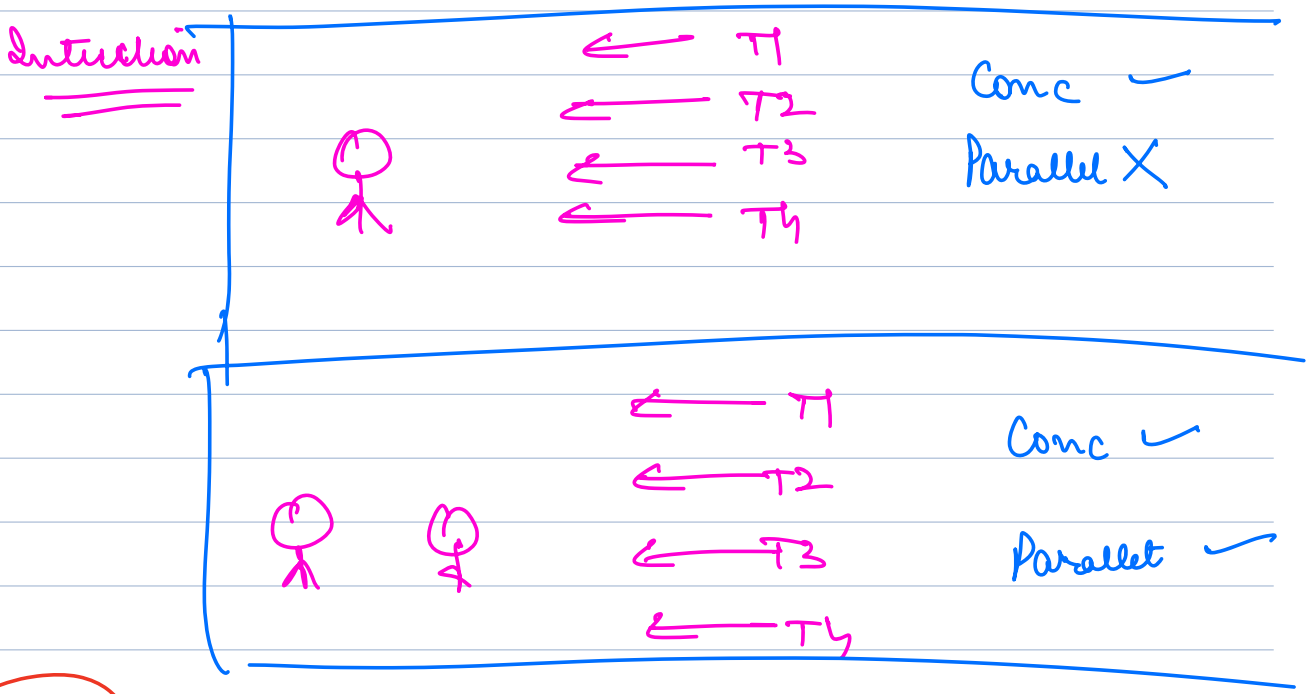
how many tasks in process → >1
how many tasks making progress → >1

Concurrent → more than 1 thing at diff stages of exec^n at the same time, not necessarily making prog together.
(May be at same time / May Not)

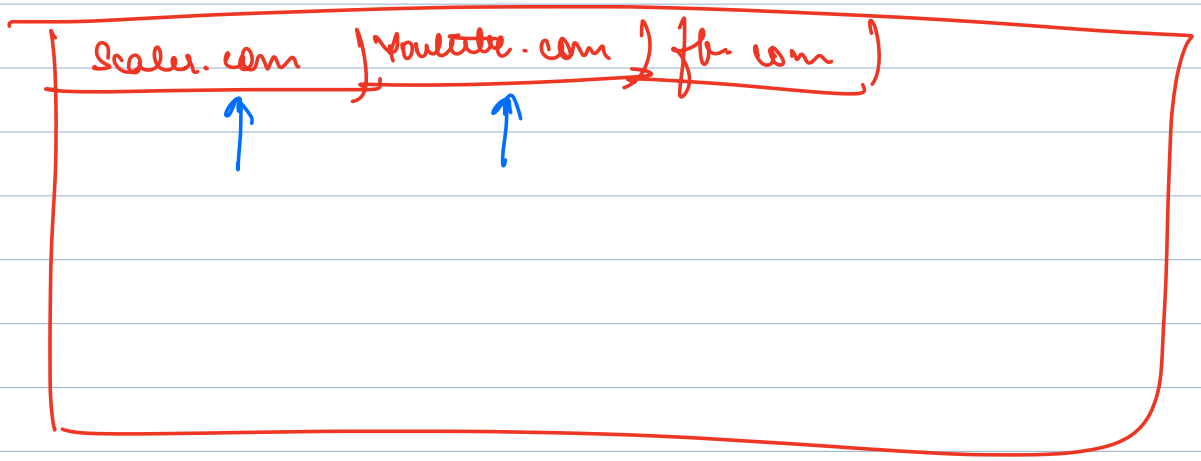
Parallelism → more than 1 thing at diff stages of exec^n and also making program

If something is parallel, is it conc ✓

If something is conc, is it parallel → Not always true



Thread → Google Chrome → Process



till now we were assigning a process to a core

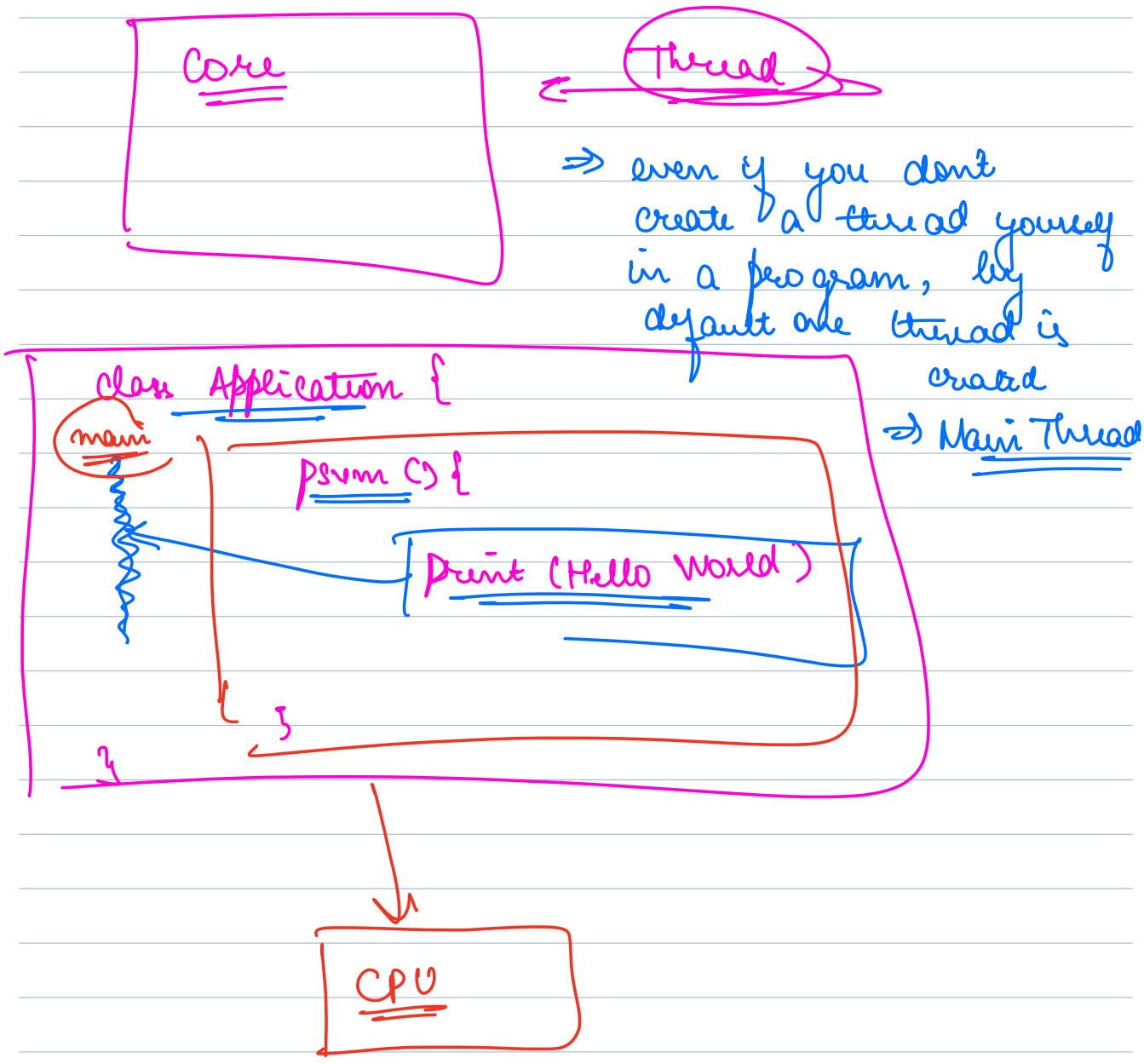
⇒ within one process as well I may want an ability to have multiple things happening at the same time

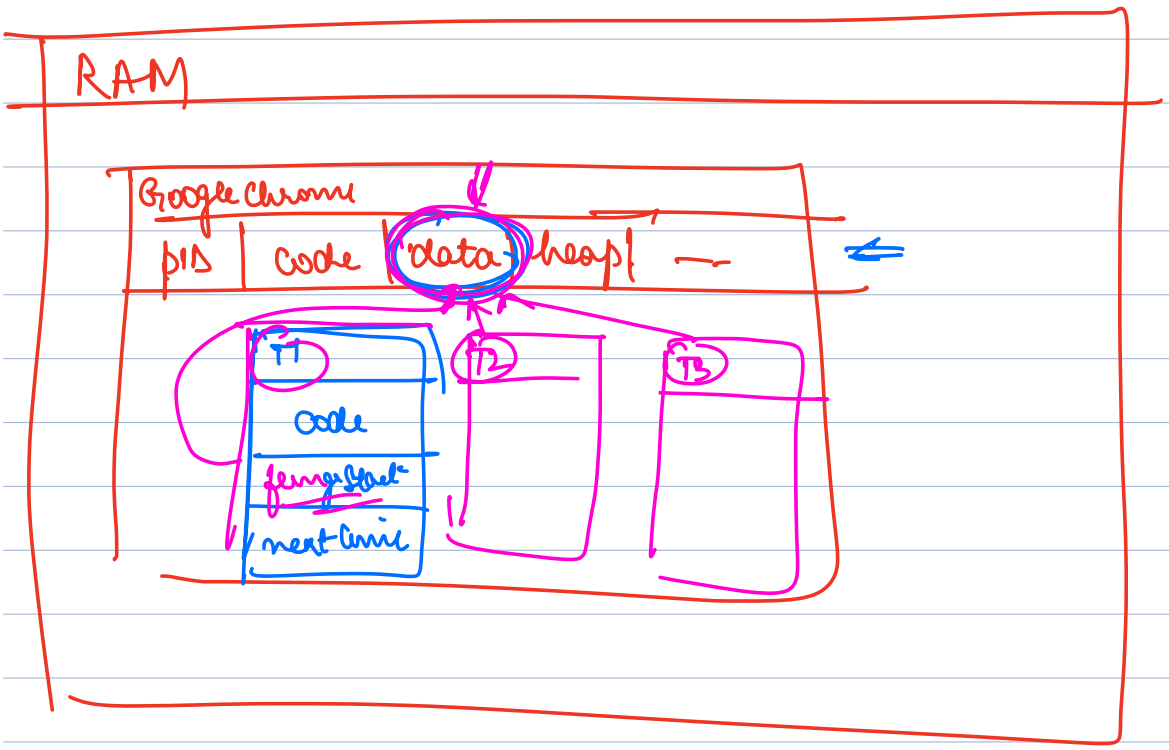
⇒ divide process into threads

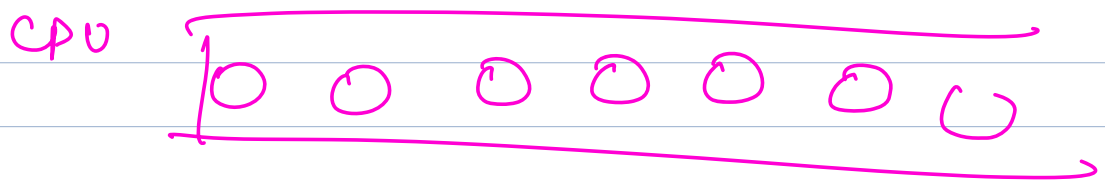
⇒ A CPU core always executes a thread

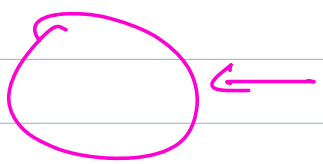
⇒ Smallest unit of CPU exec^n

⇒ what a CPU executes







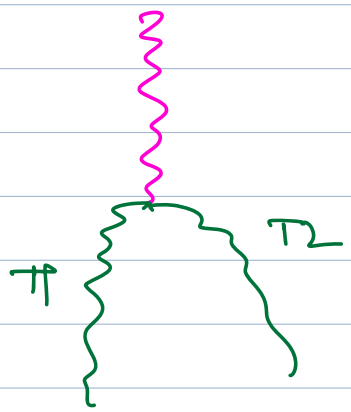


← CPU Scheduler

Thread ⇒ Unit of CPU exec^{n}

Whenever you have to create a multithreaded prog, never think of thread as a thread. Instead think in terms of "Task"

⇒ task that you want to do parallel



"Data Synchronization"

Multi Tasking v/s Multi Program

Same

Tasks: Windows.
Program: Unix

HOW TO CREATE A MULTITHREADED PROG

Qm: Create a simple program that prints hello world, but from a sep thread.

⇒

DEFINE TASK

(1) Think what are tasks to be done in parallel

→ Print Hello World

(2) For each of the task in (1) create a class for them.

→ Name that class as what task it performs

→ HelloWorldPrinter

```
class HelloWorldPrinter {
}
```

(3) To mark something as a thread (task to be done parallely), implement Runnable in that class.

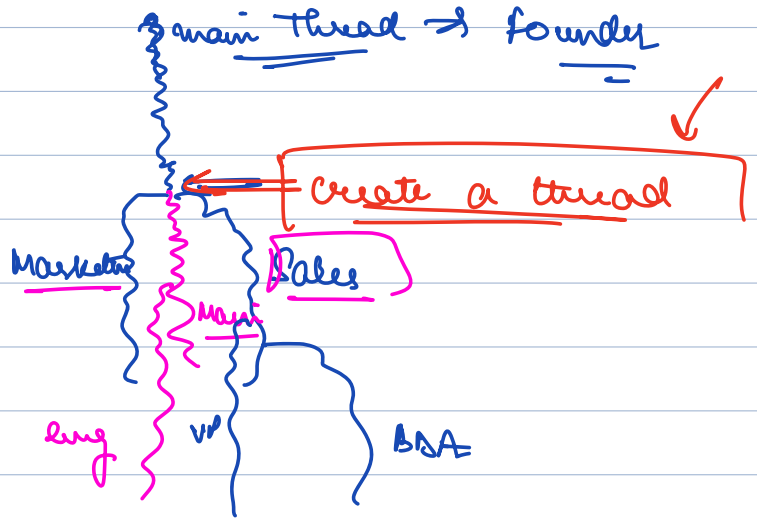
```
class HelloWorldPrinter implements Runnable {
}
```

④ Implement run() method in those classes.

```
class HelloWorldPrinter implements Runnable {
    @Override
    void run() {
        print("Hello World");
    }
}
```

⑤ In run() & Write code that does what you want to do in parallel.

ACTUALLY EXECUTE TASK



→ from the place where you want to branch out the task in parallel.

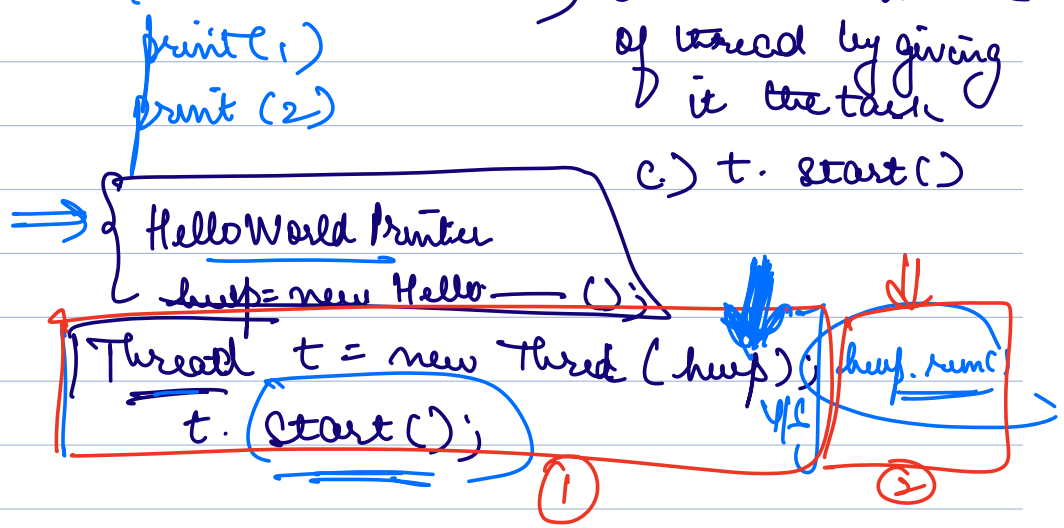
Main {

psvm() }

a) Create an instance of task

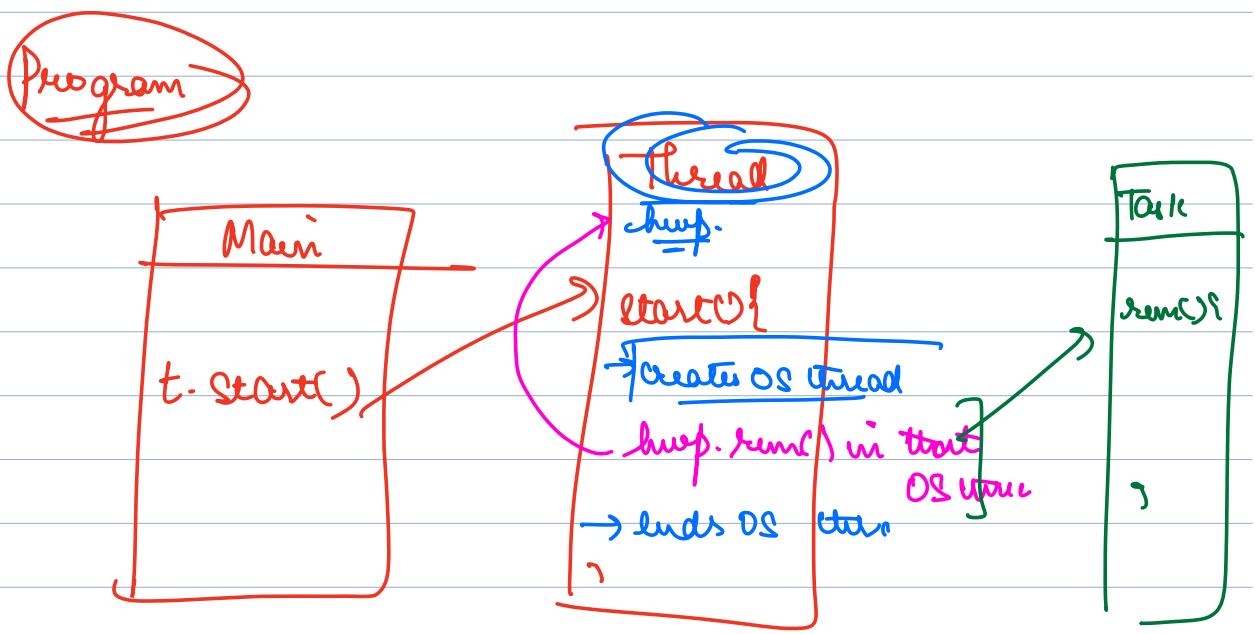
b) Create an instance of thread by giving it the task

c) t.start()



Run() V/S Start()

(1) Run() is there in task.



every learner → Interviews

→ Day to Day Job

12 months

1st yoe

BDEM 2

6 mos

SDE 1

→ Programming language (any)
→ DSA → 6 months ≈ 900 Qns
→ SQL → 1 month
→ LLD → 2.5 months
↳ OOP
↳ DP, DP
↳ Machine coding
→ HLD → 1.5 months
↳ CAP Theo
↳ Hashing
↳ NoSQL
↳ Queue
→ Projects → 1 month
↳ Payment G/W
↳ Auth Layer
↳ Deployment & Cloud

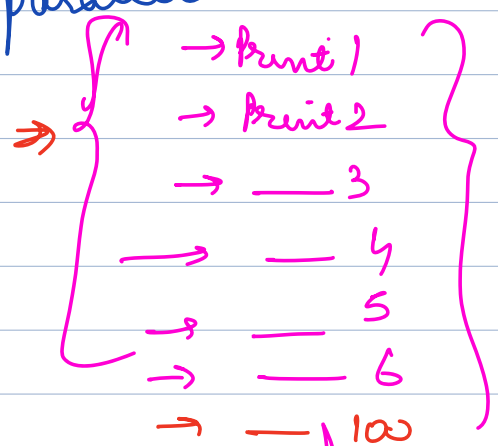
Fullstack

Backend

→ Algo Qns } 5/day
→ HW Qns } 5/day
→ TA Support
→ 1:1 Mentorship
↳ Mock Interview
→ Placement Support
↳ ≈15 offers/day
≈ 25 calls

Q2 Implement a multithreaded program which prints numbers (1 to 100) but each from a separate thread

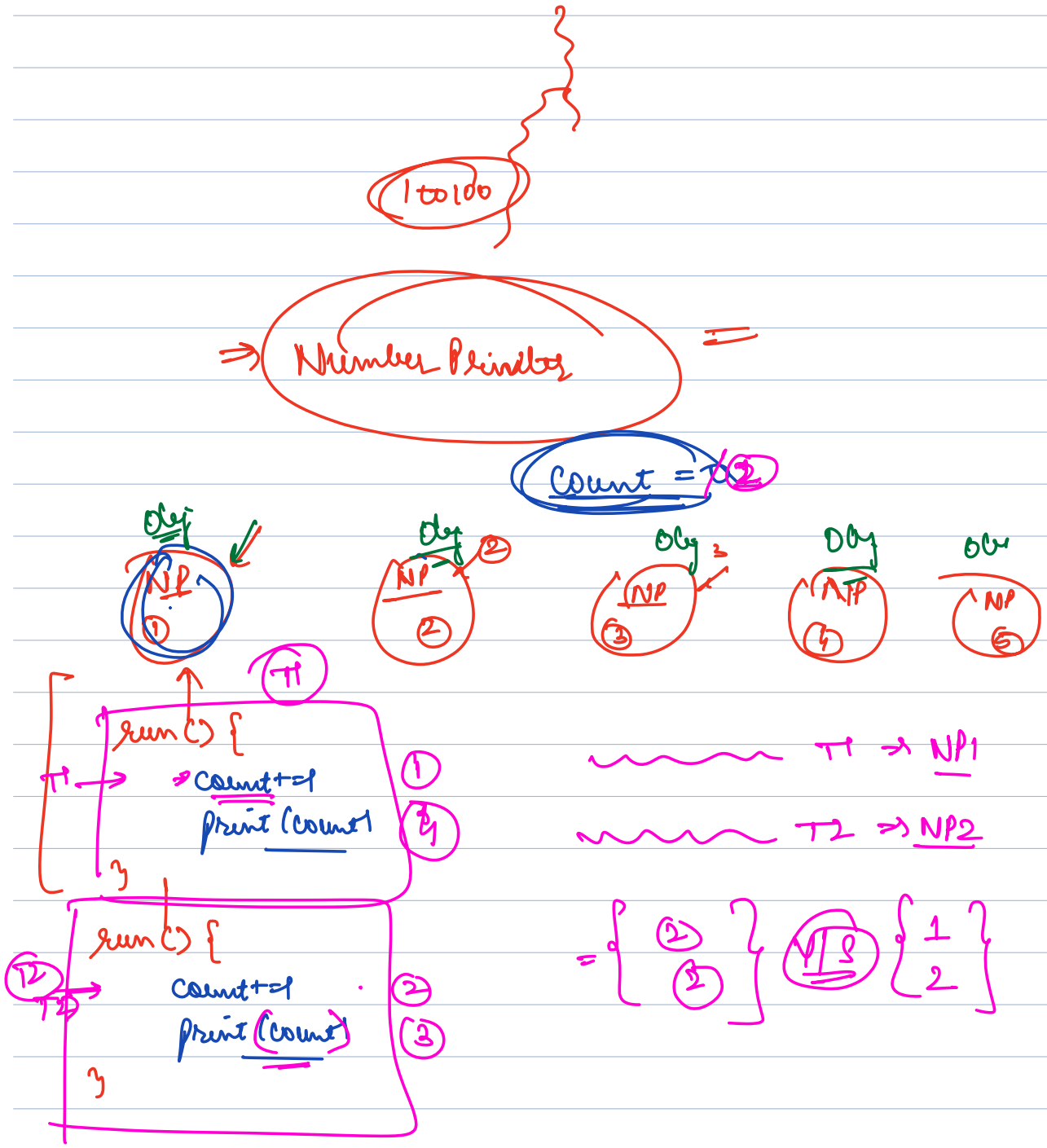
(1) Find the task that you want to do in parallel.

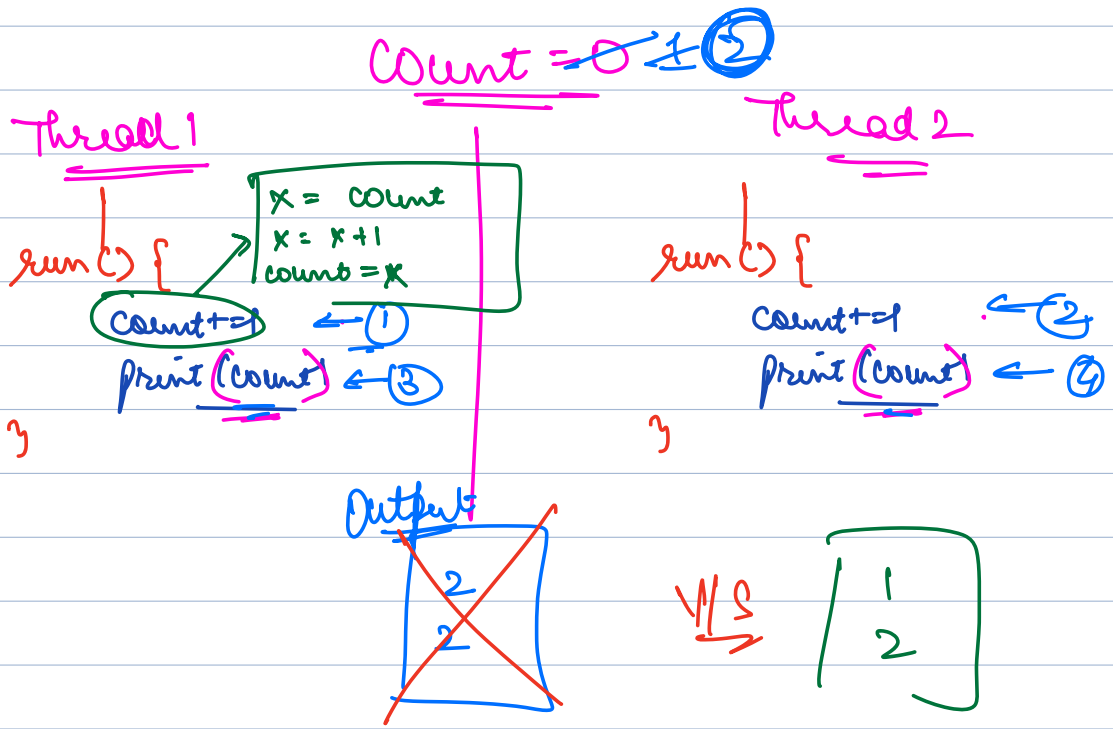


→ 100 tasks

But all of them are similar.

↳ each is printing a number


1 to 100
Number Printing
=
Count = 0
Obj
NP
1
Obj
NP
2
2
Obj
NP
3
3
Obj
NP
4
Obj
NP
5
T1
run() {
Count++
print (Count)
}
1
4
T1
T2
run() {
Count++
print (Count)
}
2
3
T2
T1 → NP1
T2 → NP2
= { 2 2 } VTS { 1 2 }



⇒ Data Synchronization

---

1. Task ⇒ Number Printer
2. class Number Printer { }
3. class Number Printer implements Runnable { }
4. class Number Printer implements Runnable {

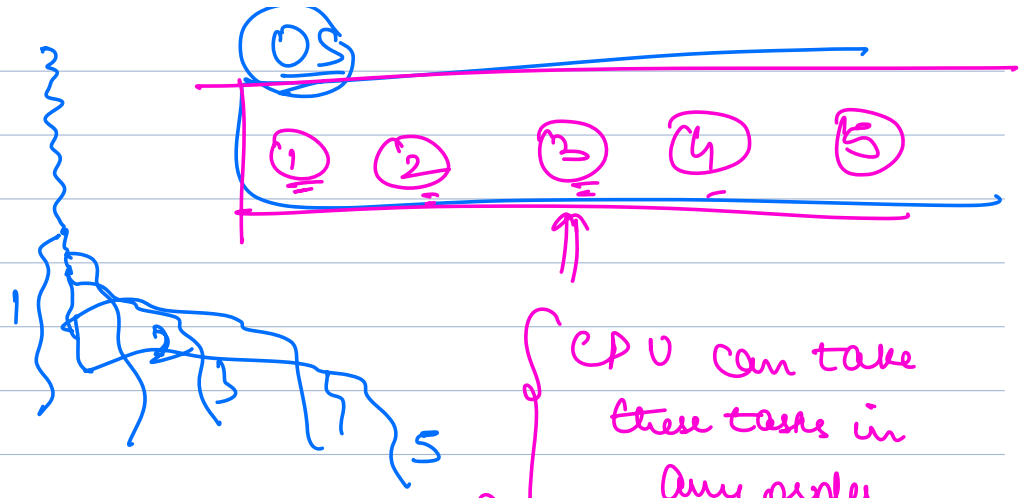
⇒
```
run() {

}
```

}

(5)

```
class NumberPrinter implements Runnable {
    int numToPrint;
    NumberPrinter(numToPrint) {
        this.numToPrint = numToPrint
    }
    run() {
        print(numToPrint)
    }
}
```

```
Main {
    psvm(){
        (1);
    }
}
```

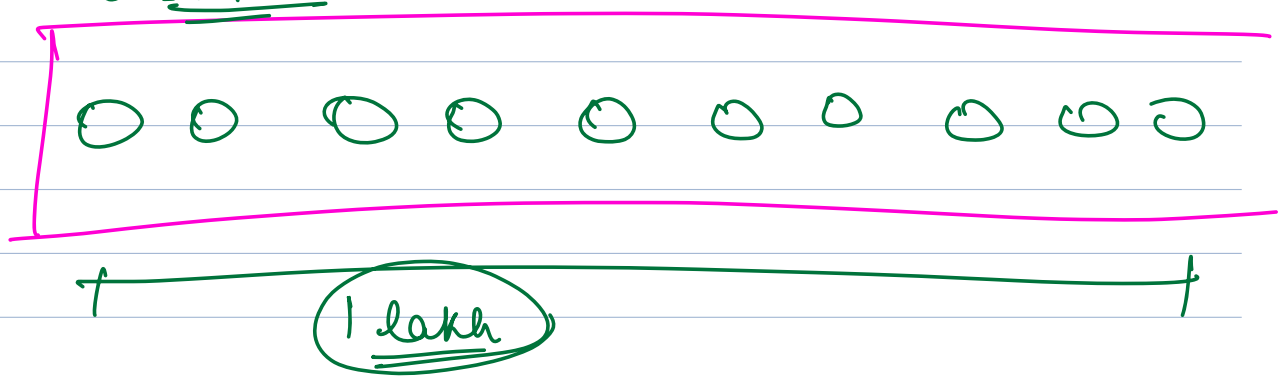


CPU can take these tasks in any order

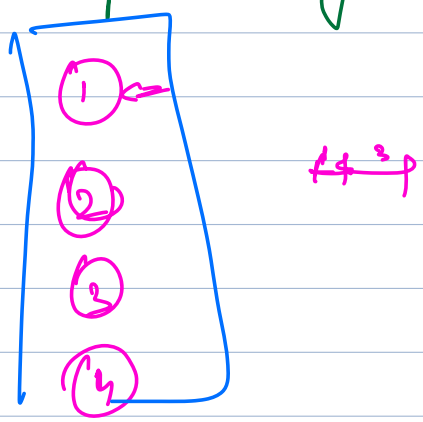
⇒ leads to out of order printing

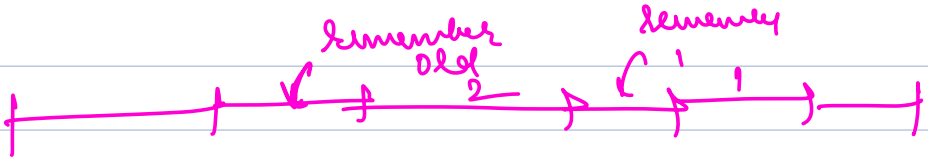
CPU / OS

task Queue

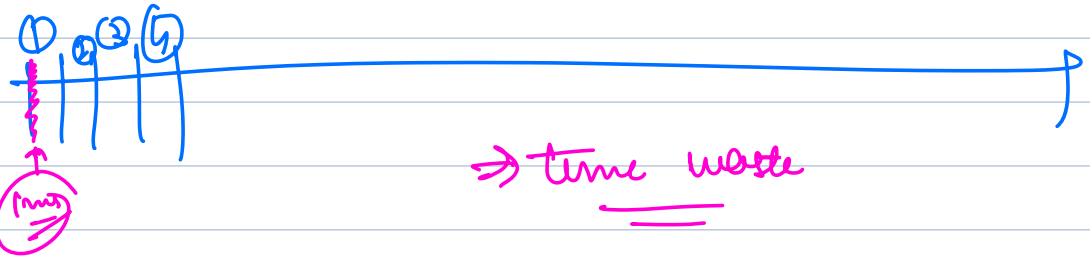


If I end up creating so many threads.



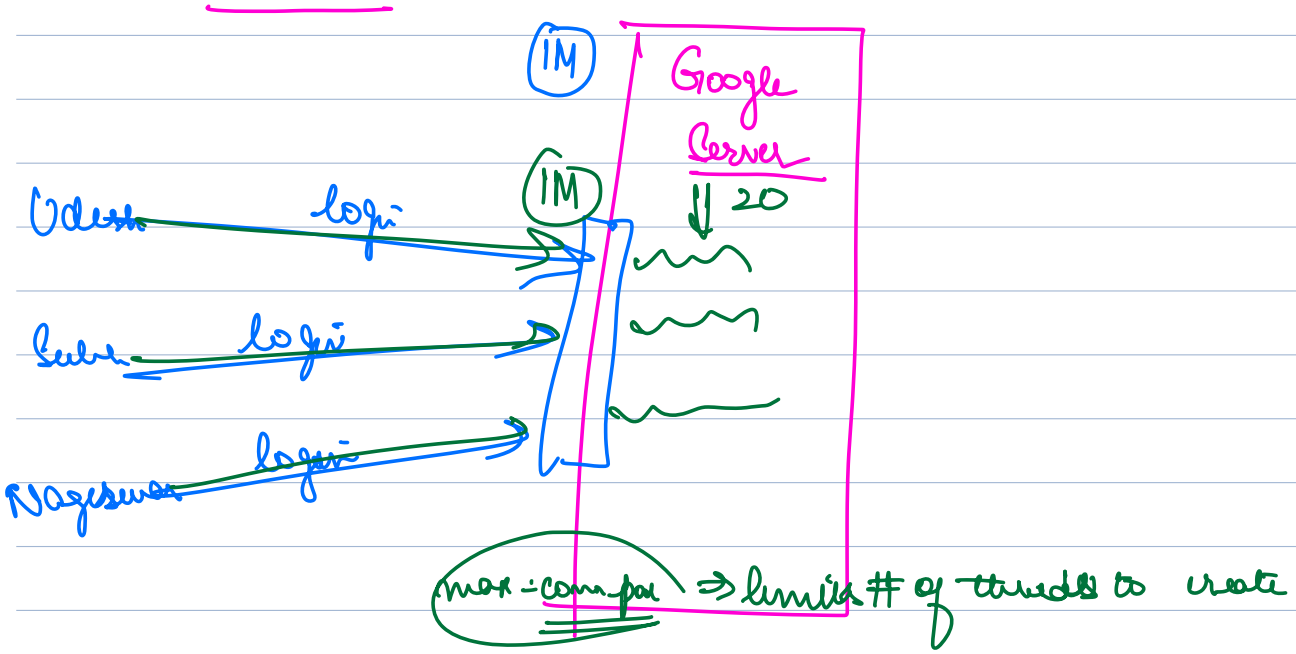


Context switching : time that is spent by a CPU remembering/bringing back state of previous task



⇒ time waste

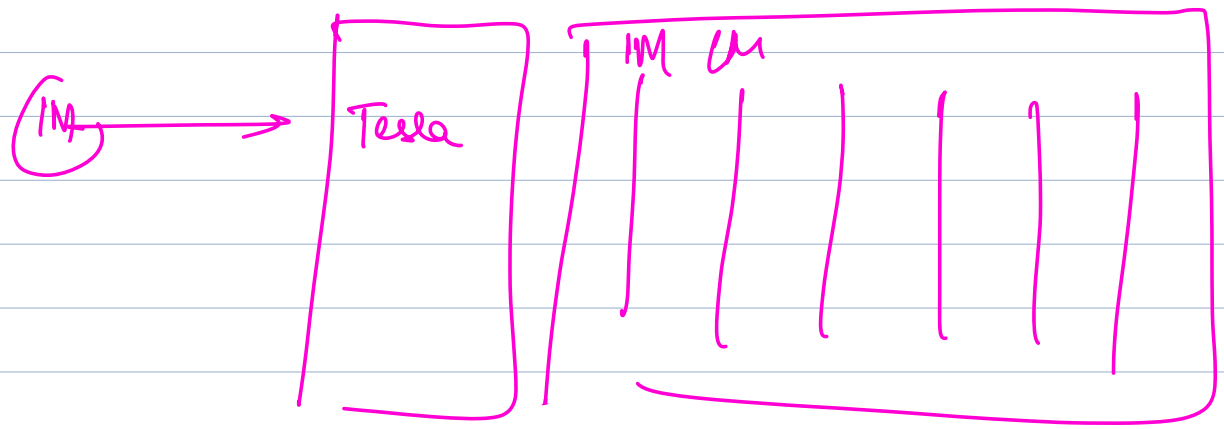
Backend Server



every req is served via a separate

max_conn_pool = 20

Tesla : Car Manufacturing Company

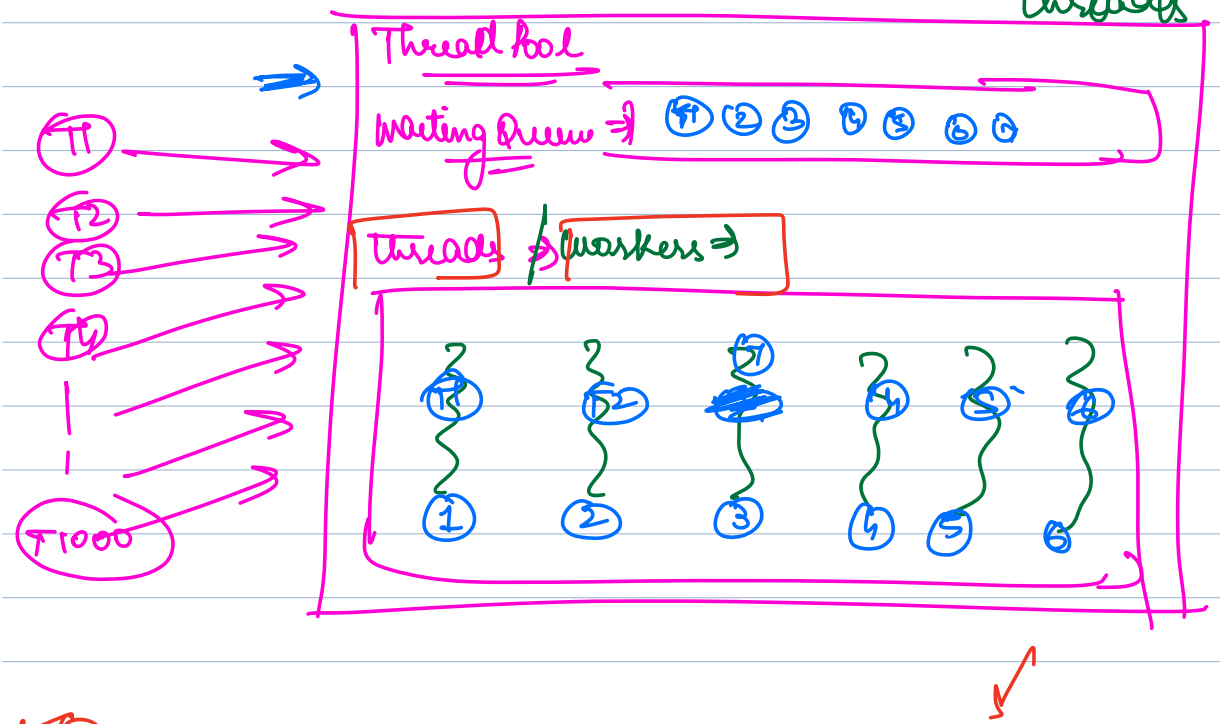


even if IM orders ⇒ 6 cars at same time

exactly same thing is done by Thread Pools

⇓

Factory for threads



① What should be logic of thread pool ?

② How to create a thread pool

Executors

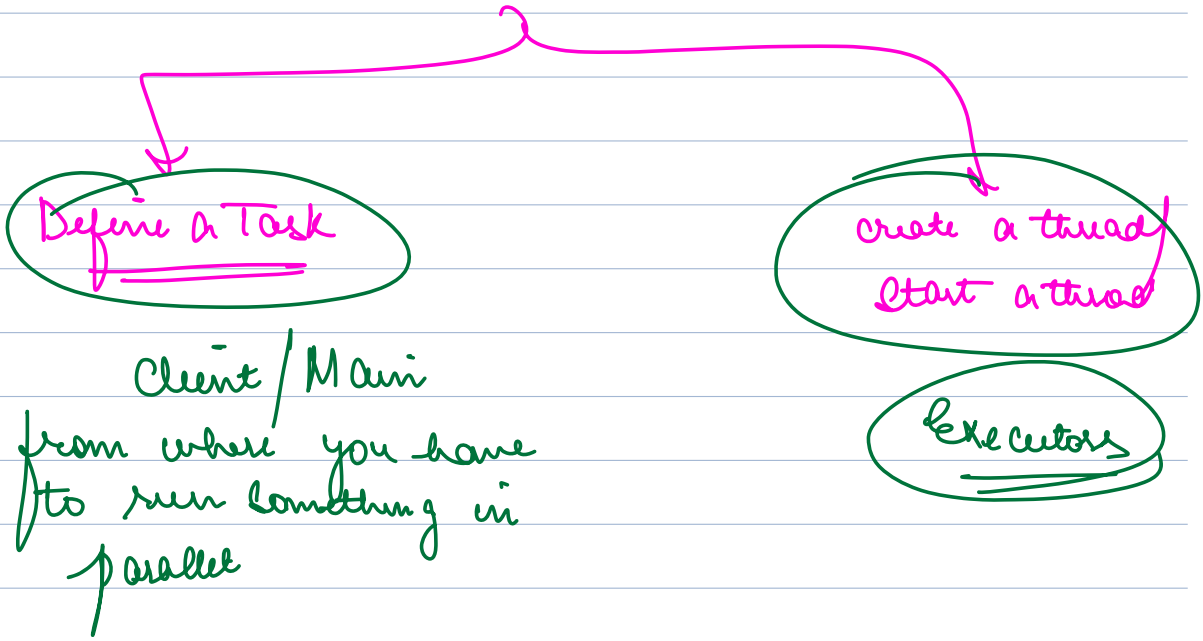
2 types of things that are done to run a task in a thread

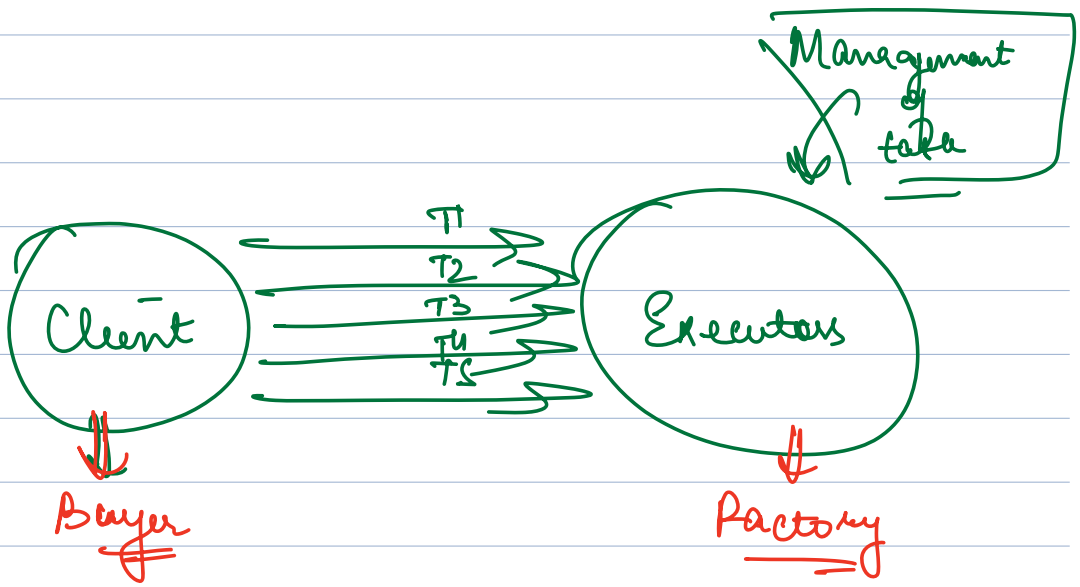
① Create Task

② Created a Thread

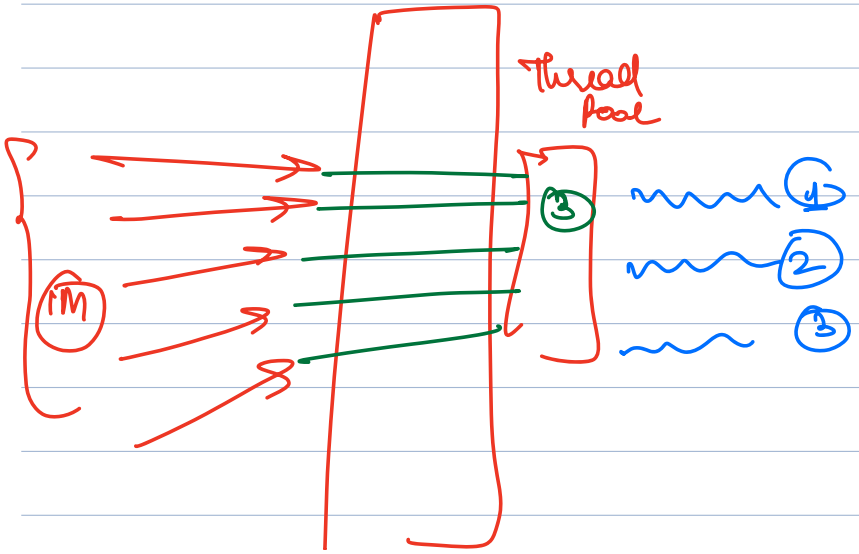
③ Started Thread

dependent on a lot of factors





Nginx | AWS LB | Spring Boot



How many threads?

↳ 2^x # cores

↳ # cores

depends on type of task

- I/O intensive task
  - → Accessing disk
  - → Printing / Input
  - → N/W issues
- CPU intensive task

→ Why threads: use CPU to fullest

I/O → CPU is not being used

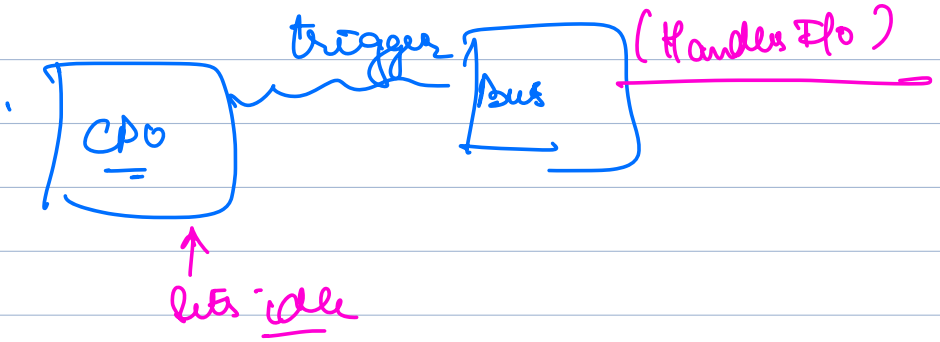
→ not the most optimal use of CPU

→ if a task is currently doing I/O I should give some other task to CPU

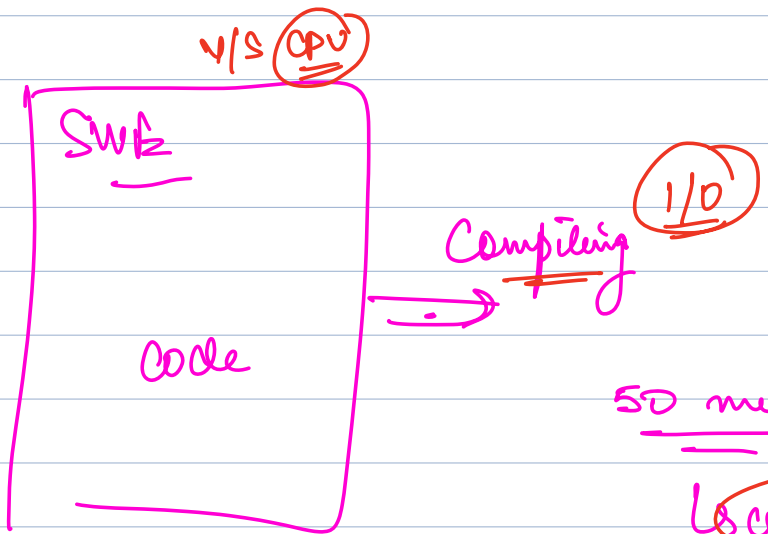
8 core

→ CPU just starts I/O, after that I/O happens independently, completes

I/O → N/W req
→ Printing / Taking Input
→ Storing something to disk



→ when I/O is happ → CPU is free
→ can I give some other task to CPU



50 mins
idle
Chai Sutta
Do some other work

I/O intensive tasks → More
∴ when one is doing i/O

Other task
4-Core → 8 Core

CPU can do other

CPU intensive → less → # cores → 2^ # core

2 threads / core

Non-Scaler

- Udemy
- Concurrency in Practice
- Machine Coding Case Studies.
  - Dist Cache
  - Pub Sub Queue
  - Rate Limiter