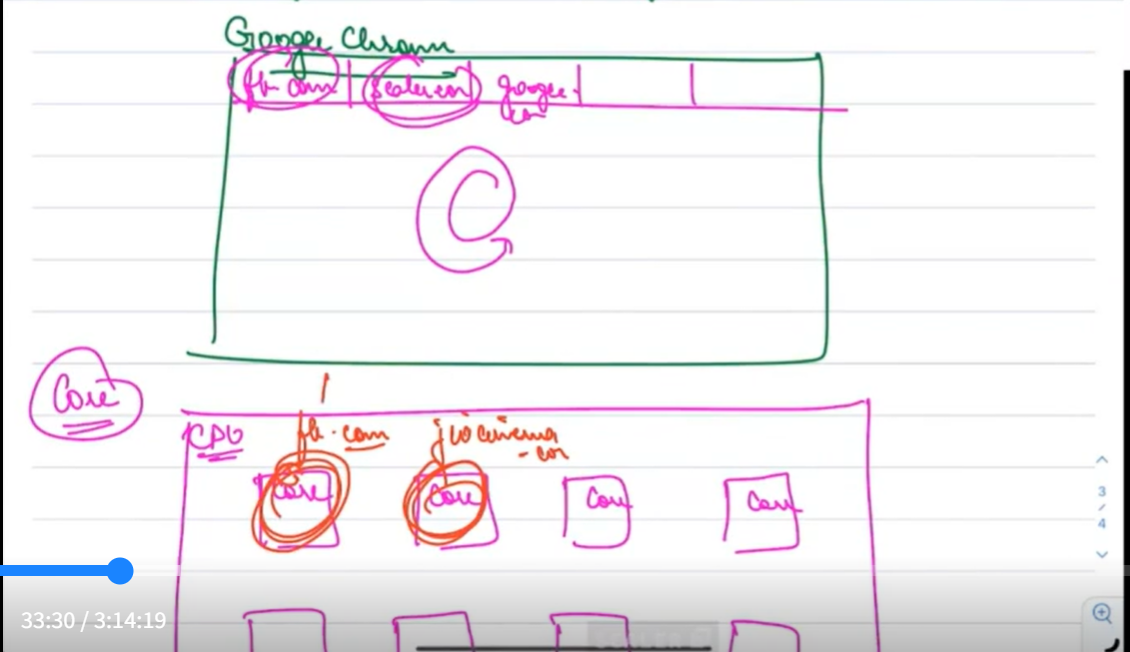
Threads, Multithreading, Concurrency, Parallelism

Program: Something that can be executed that can run to cause a particular action.

When program runs it becomes a process.

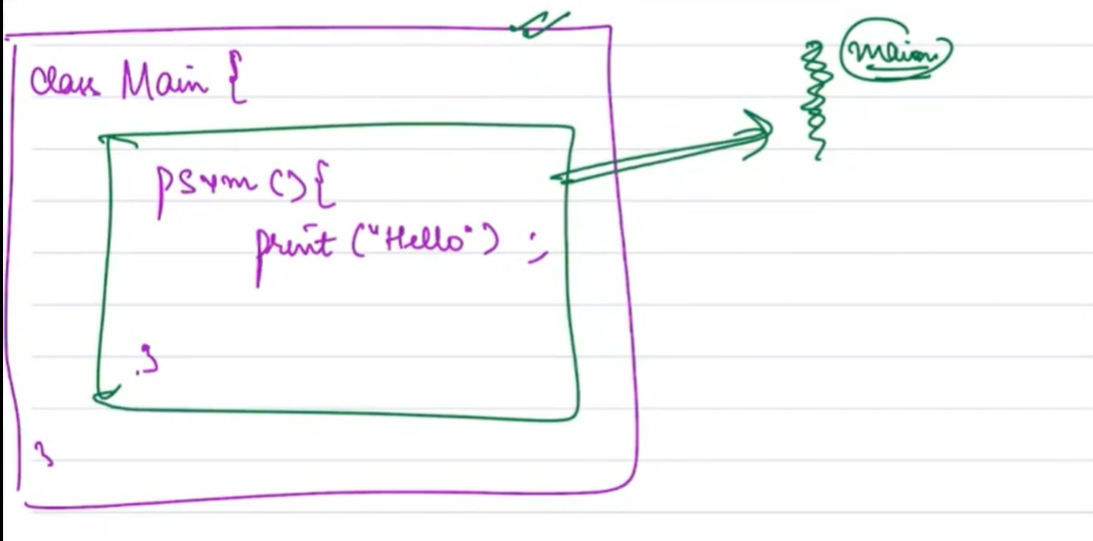
Process executes in CPU, all the data stored in RAM.

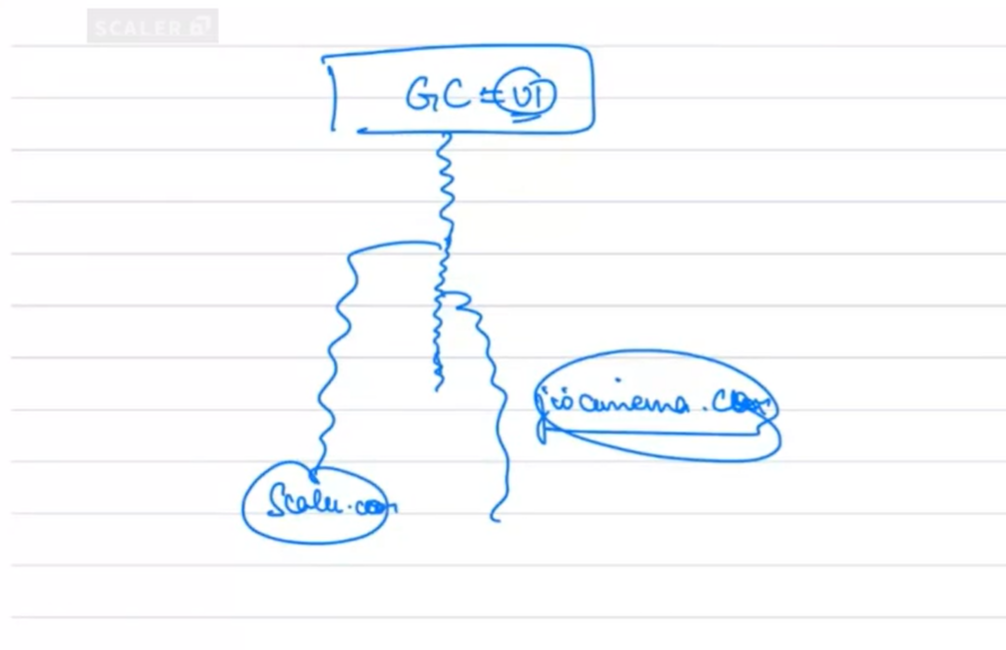
These days CPU has multi core. Core is something like a brain.



Thread----Basic unit of CPU execution. Each CPU core executes a thread.

Every process consists of at least 1 thread.



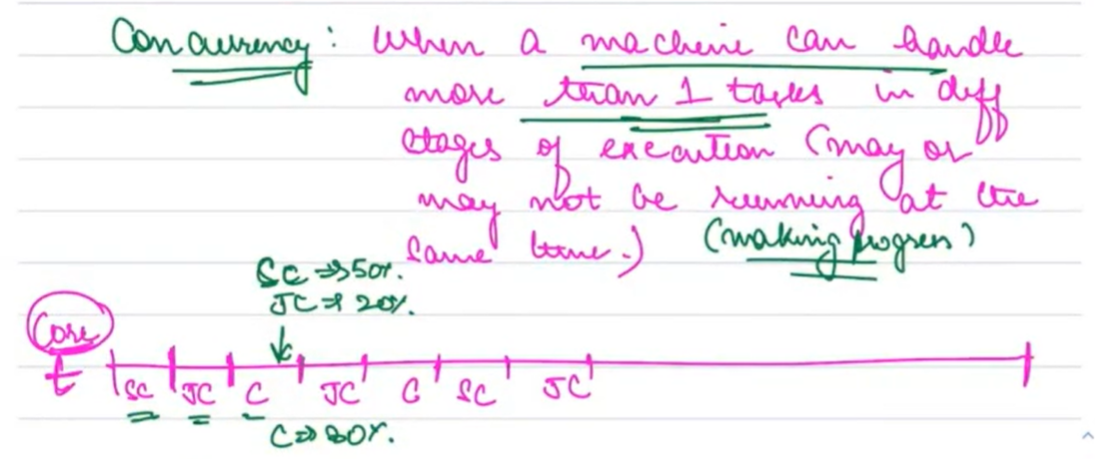


The below description is for a CPU with a single core:

Time Slicing----CPU maintains a task queue. Every item in the queue is allotted a time for execution after which the next item in the queue gets executed with it’s allotted time and the same thing continues. So actually, at any given point of time it performs only one task, however, it looks to the end user that the CPU is running multiple threads, but in reality it is running only one thread at a time.

Concurrency vs Parallelism:

Concurrency When a machine can handle more than one task in different stages of execution(may or may not be running on the same time).



Parallelism: When a machine can handle more than one task making progress at the same time.

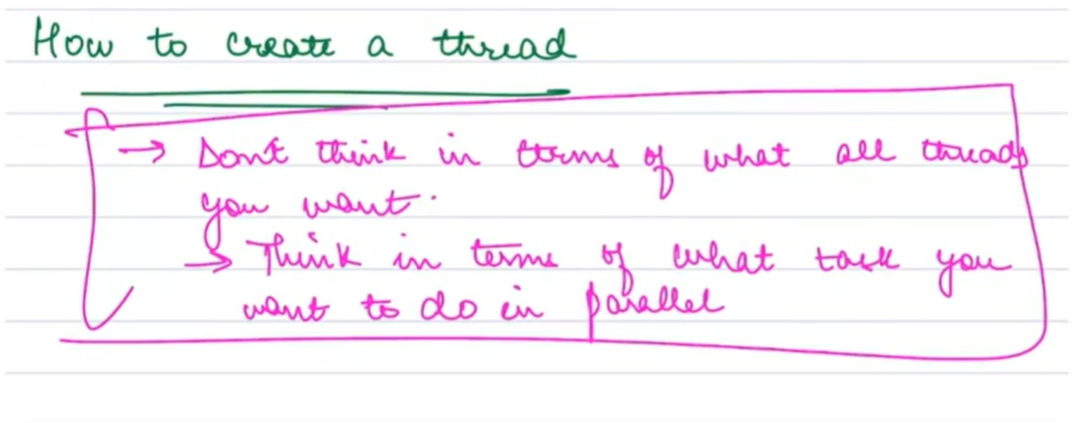
Note: A single core machine can’t handle more than one task at the same time.

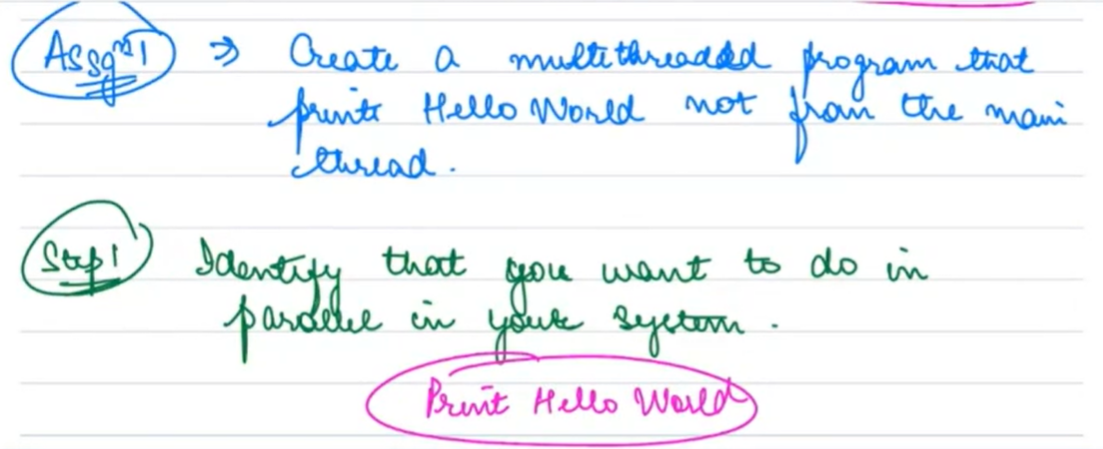


If something is parallel, it is concurrent.

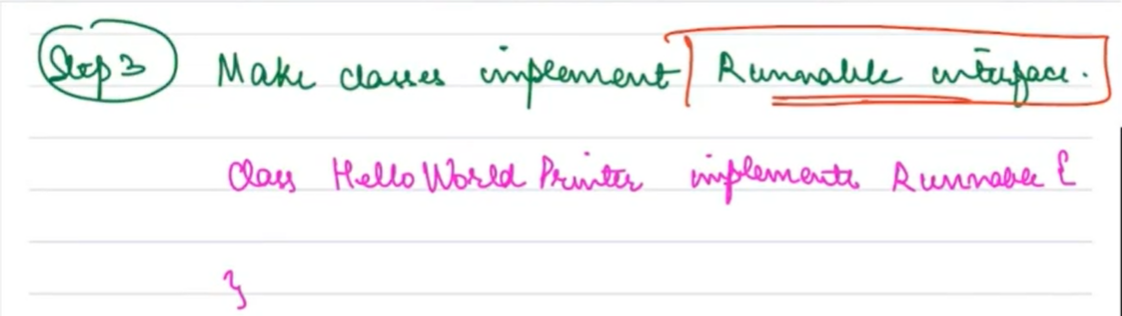
If something is concurrent, it may or may not be parallel.

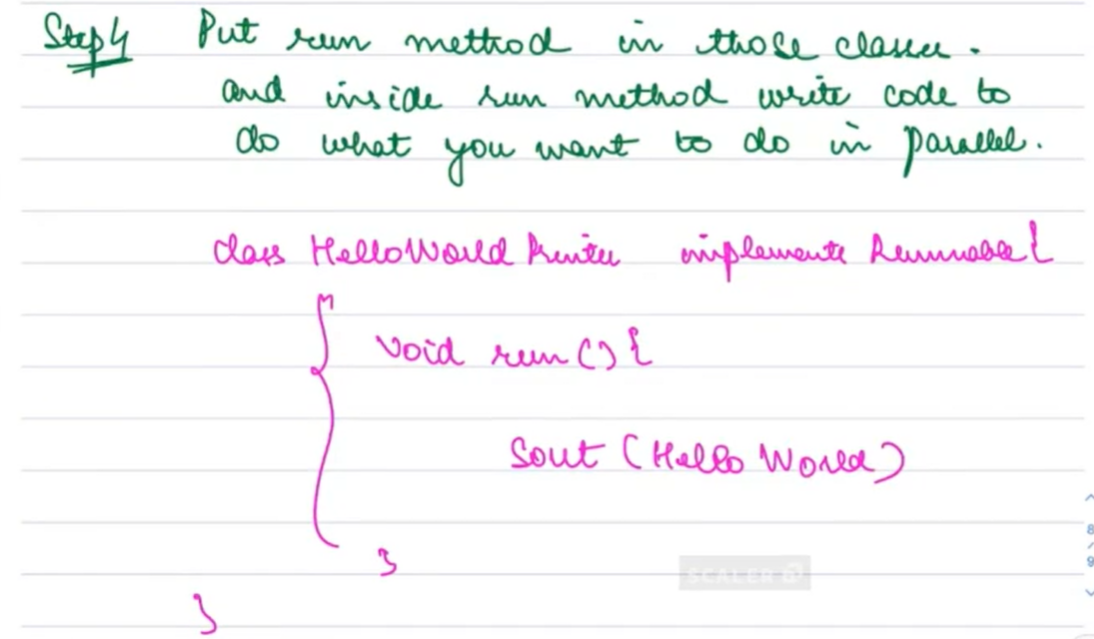
How to create a thread?

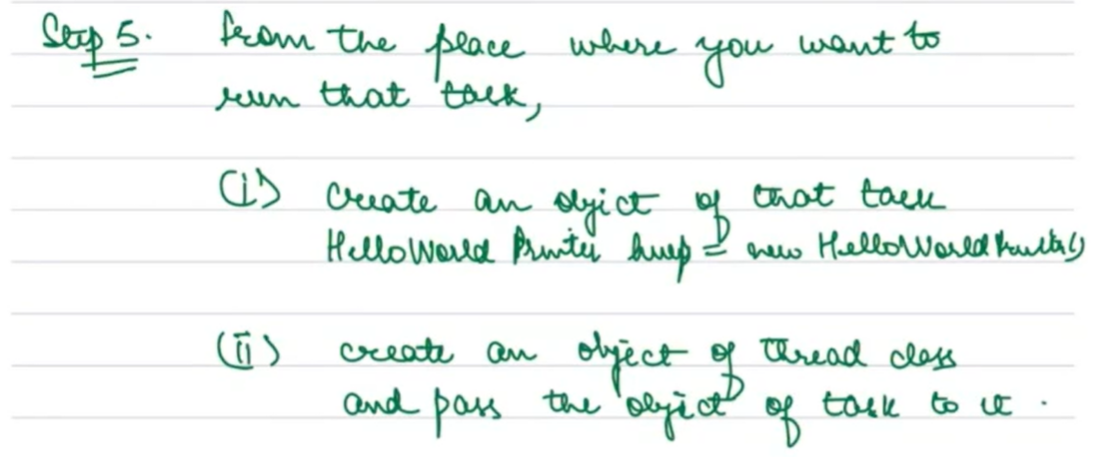


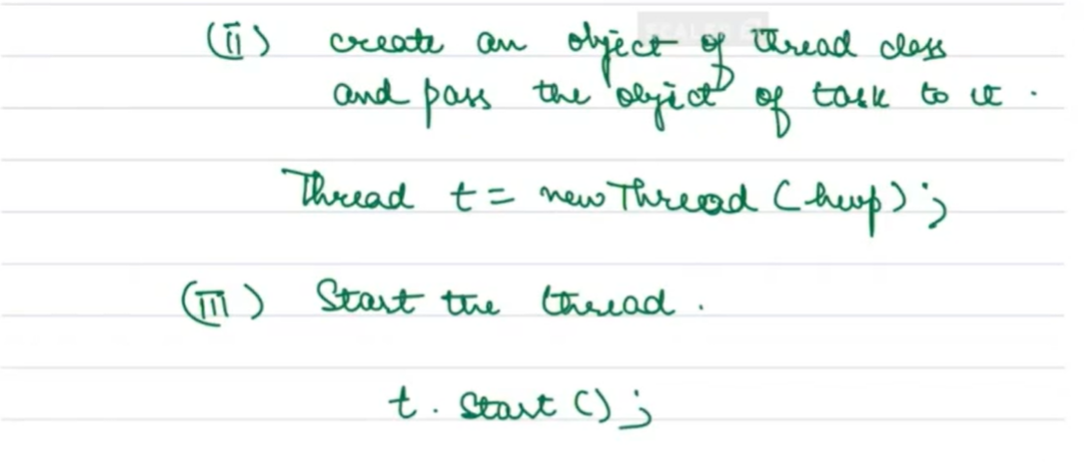


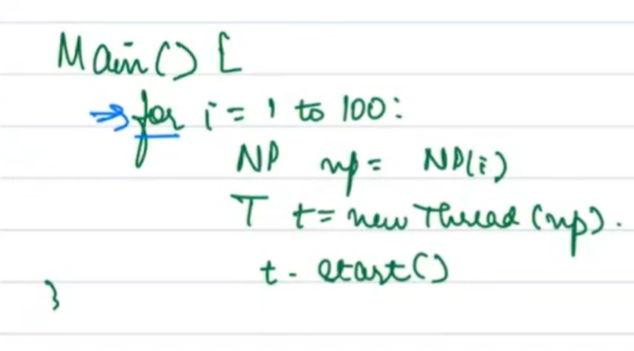


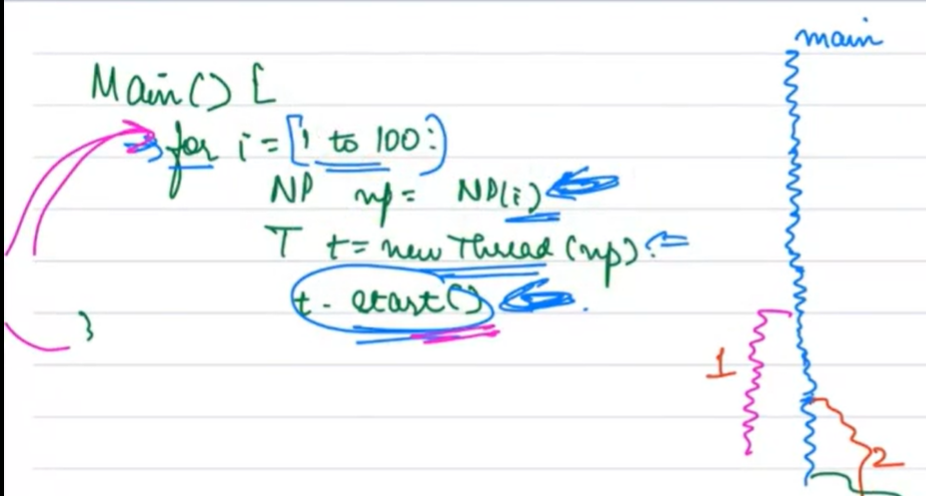










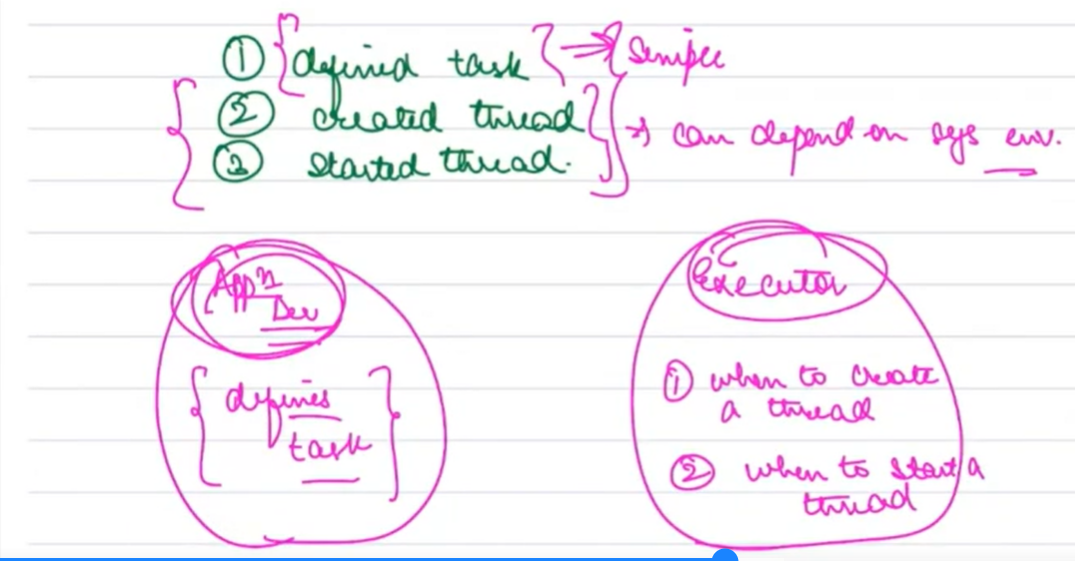


Context Switching----- CPU does not perform task at this time.

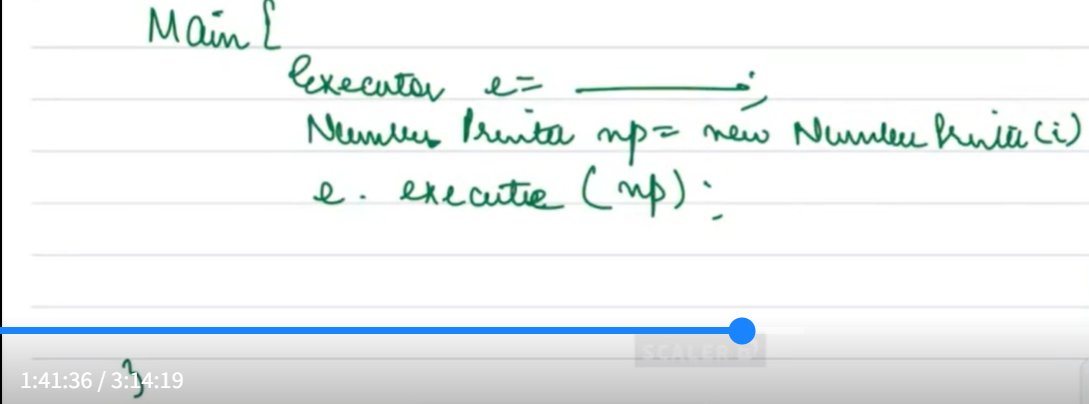
Max Number of threads<=2 times the number of cores(Rule of thumb ,Generally holds but not always true)

Note: I/O intensive tasks and CPU intensive tasks can have different thread requirements.

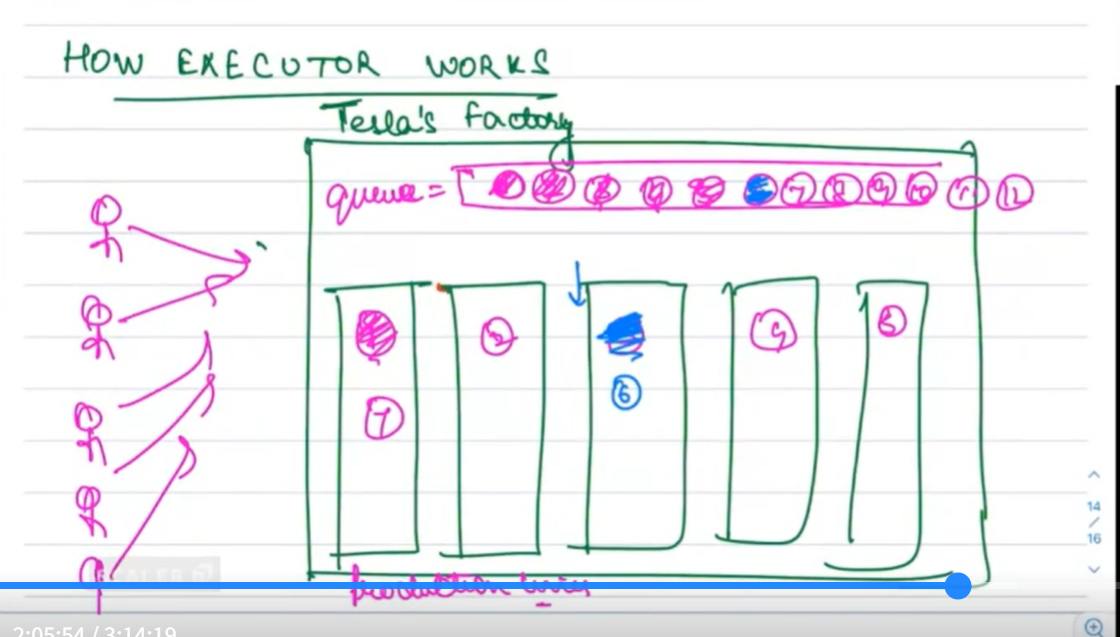




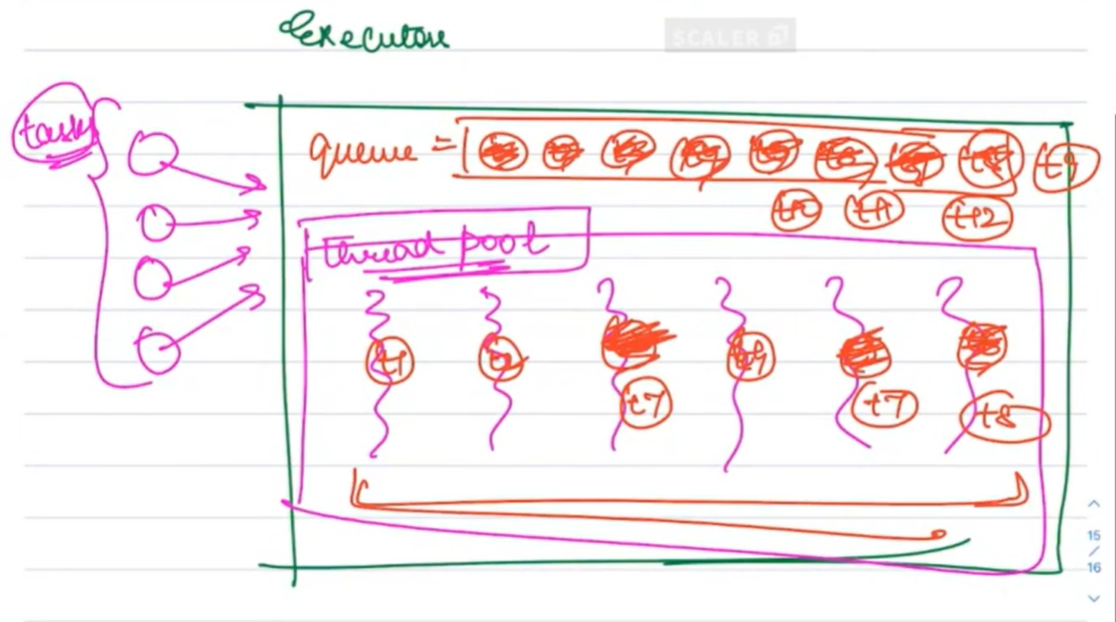
In typical production environment, it is difficult to determine when a thread should be created and when a thread should be started. This is taken care by executor. Defining the task is taken care by the Application Developer.



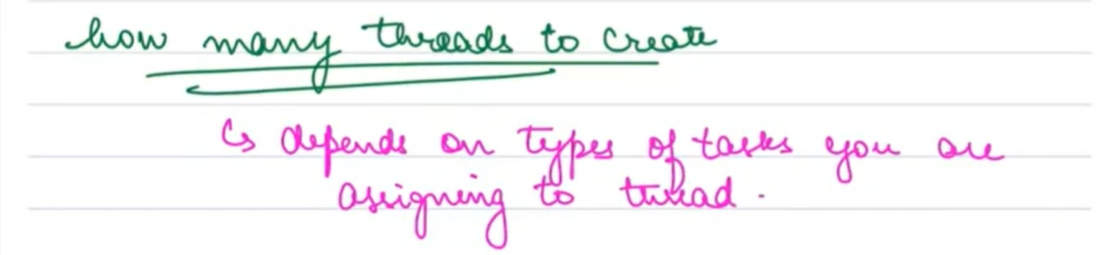
Need of an executor: Simplifies the work as it takes the responsibility of creating and starting thread.

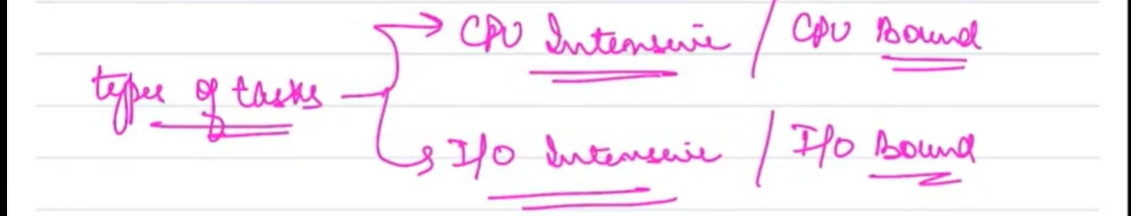


Executor maintains a thread pool. Executor also has a queue.



Example may be a database connection pool for the above screenshot.

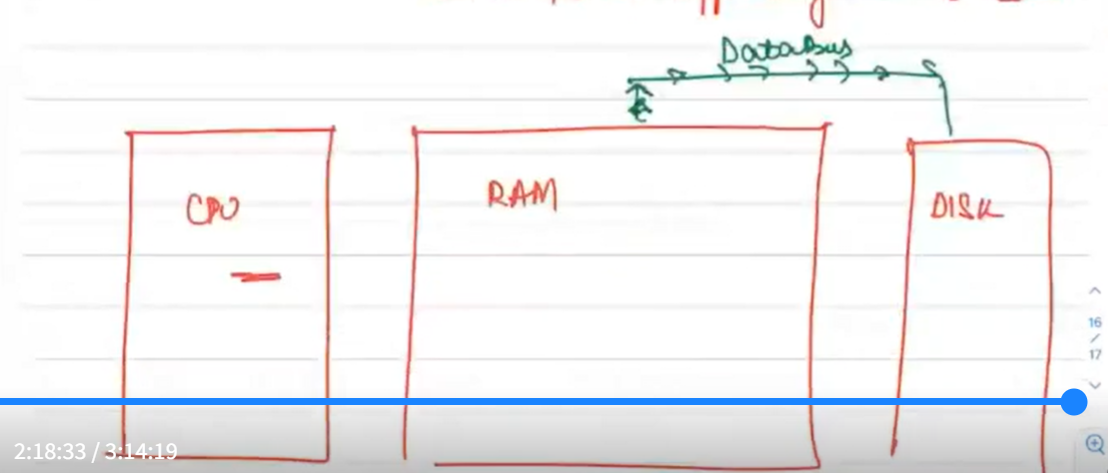




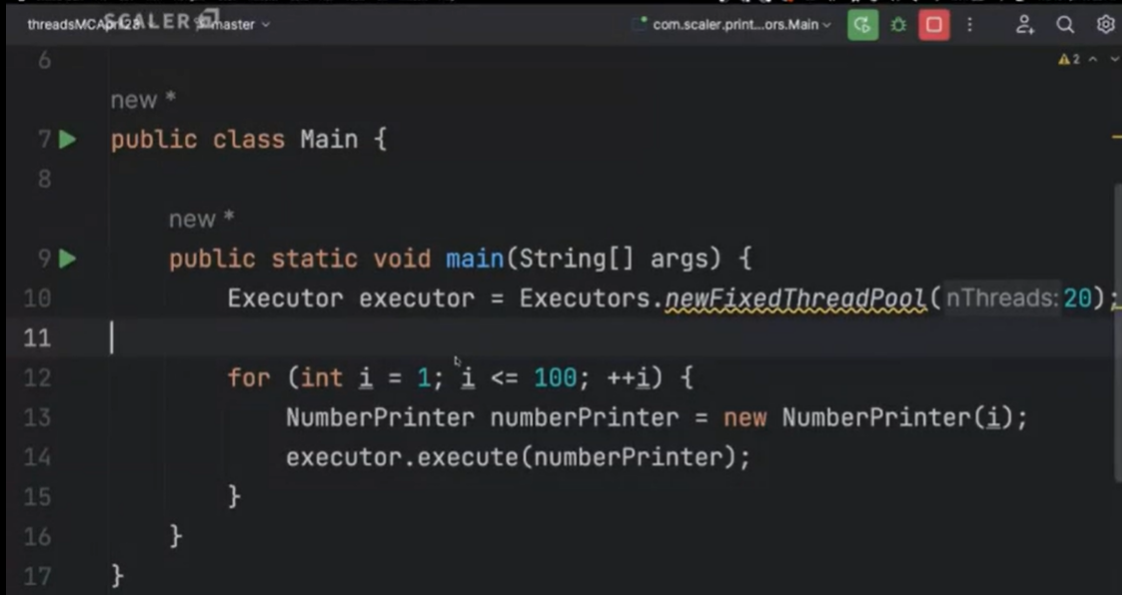
I/O----sending data/ receiving data outside the RAM is I/O.eg----printing in console is I/O, accessing disk, accessing network are all I/O. When I/O is happening, CPU is idle. So some other tasks can be given to CPU. So, in case of I/O, number of tasks can be more. Max number of tasks<=4\*Number of cores or depends on the situation.

In CPU intensive tasks, number of tasks need to be less. Here, Max number of tasks<=Number of cores.

Data Bus does the I/O operations.



CPU tasks----Anything which is not I/O task.

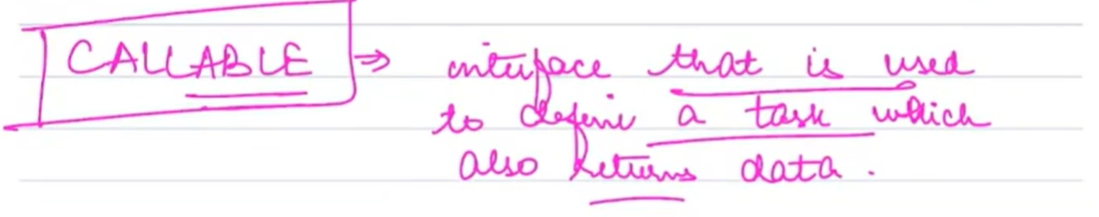


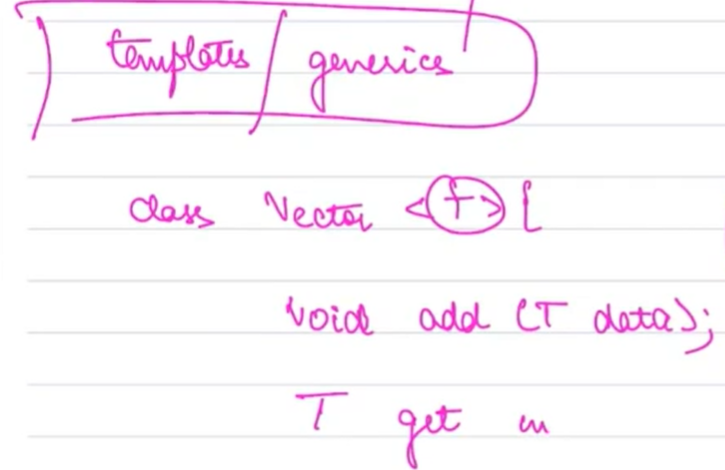
In the above screenshot, the executor restricts the number of threads to 20.

List<Threads>----known as Workers

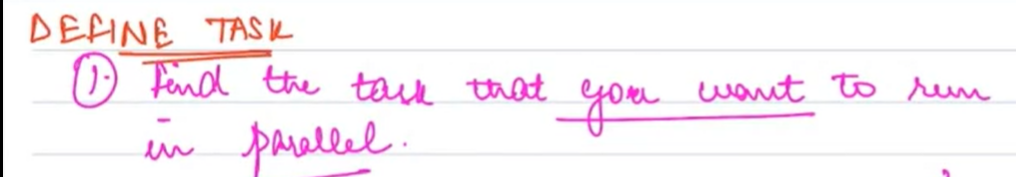
Queue<tasks>-----known as tasklist.

Scheduler is controlled by OS and not by executor.





Callable:





Future is like a promise.

