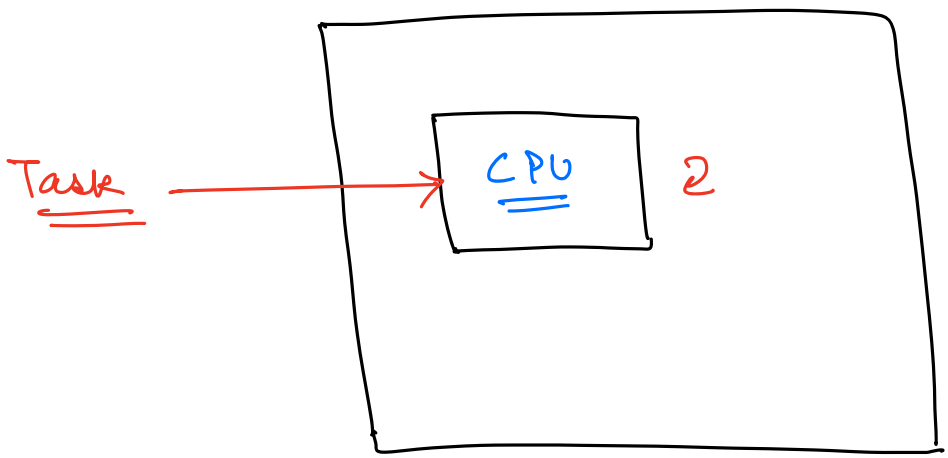
Agenda.

→ Intro to Multithreading
→ Hello World from new thread.
→ Print no's from 1 to 100
→ * Executor framework
→ * Thread Pool.

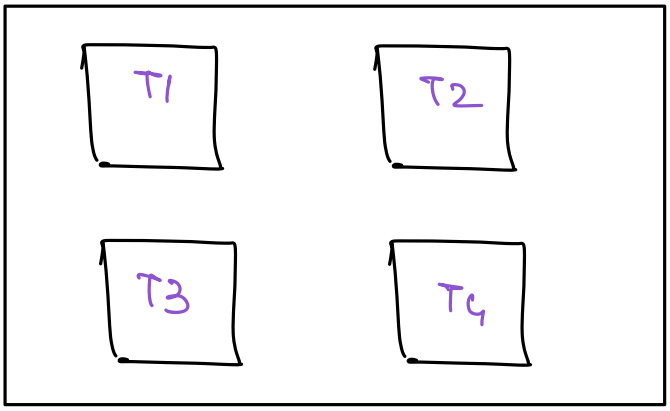
⇒ Multithreading



⇒ CPU executes the task.

⇒ Core → Execution Unit of CPU.

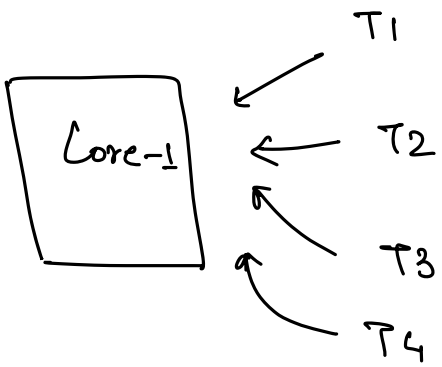
- Quad Core → 4
- Octa Core → 8

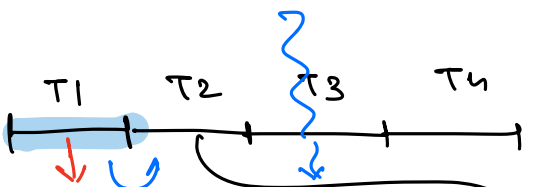


Single Core CPU.

↳ Single task.

↳ Speed = 1 GHz ⇒ 1×10^9 operations / sec.





1000 operations

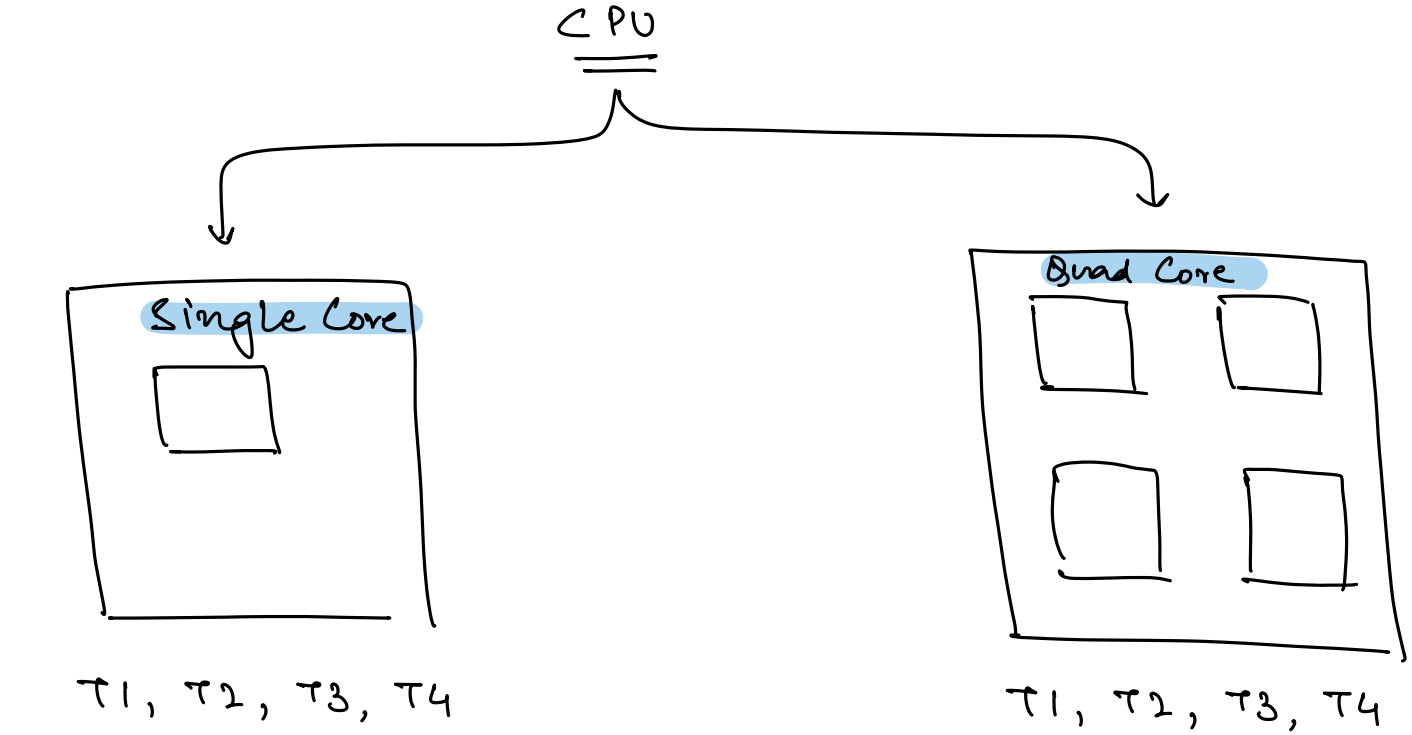
10^6 ope"

↳ 1 ms.

$10^9 \longrightarrow$ 1 sec

$1 \longrightarrow \frac{1}{10^9}$

$10^3 \longrightarrow \frac{10^3}{10^9}$ sec → 1 ms. 10^{-6} sec



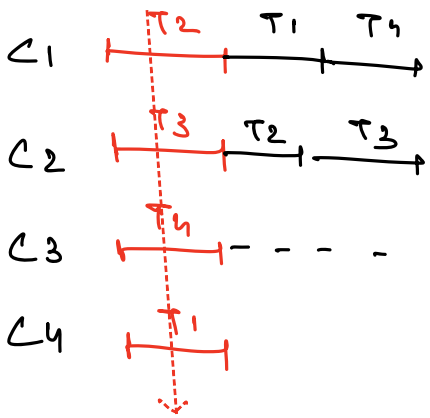
⇓

Concurrent System.

⇒ Context Switching. ↑

⇒ App^n

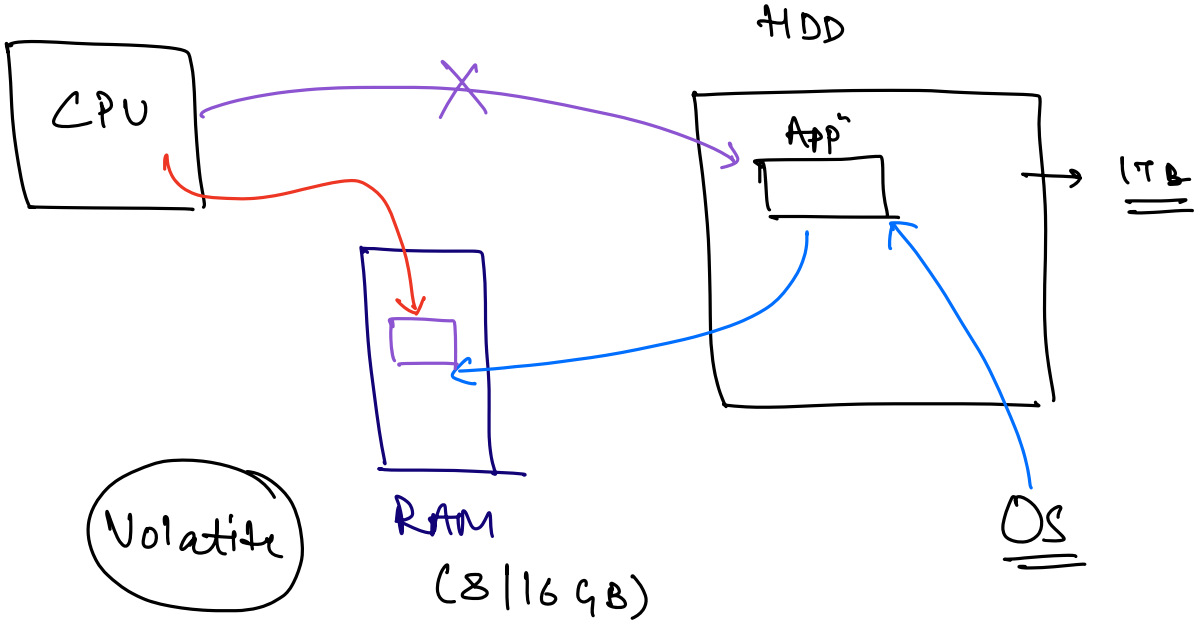
↳ Download executable file



⇓

Parallelism.

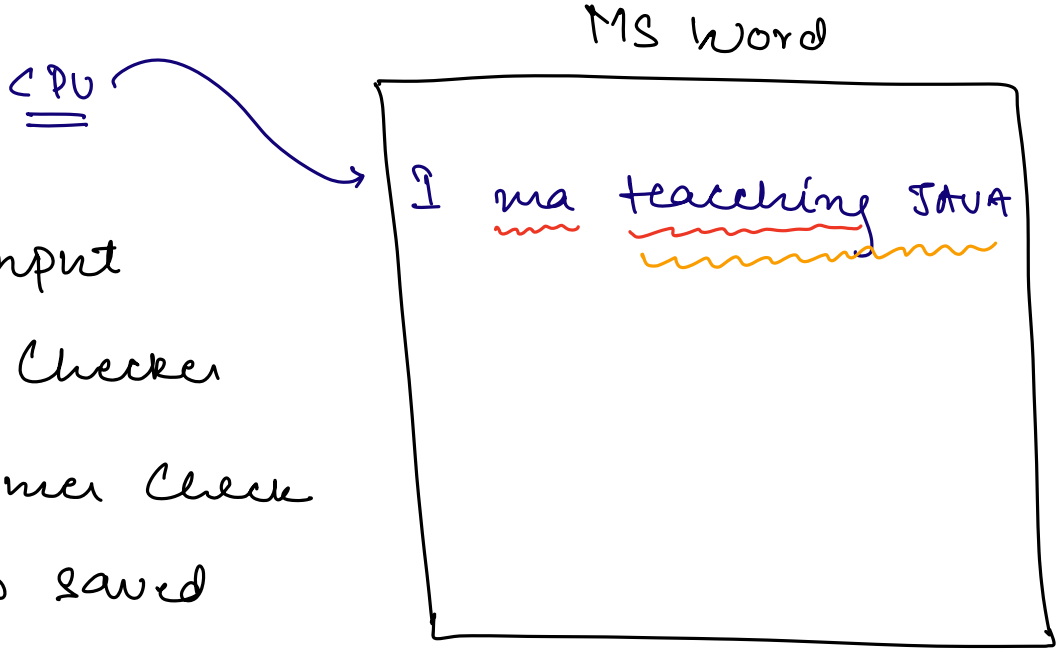
⇒ Context Switching ↓



Can CPU directly talk to HDD ? NO.

⇒ because of speed difference

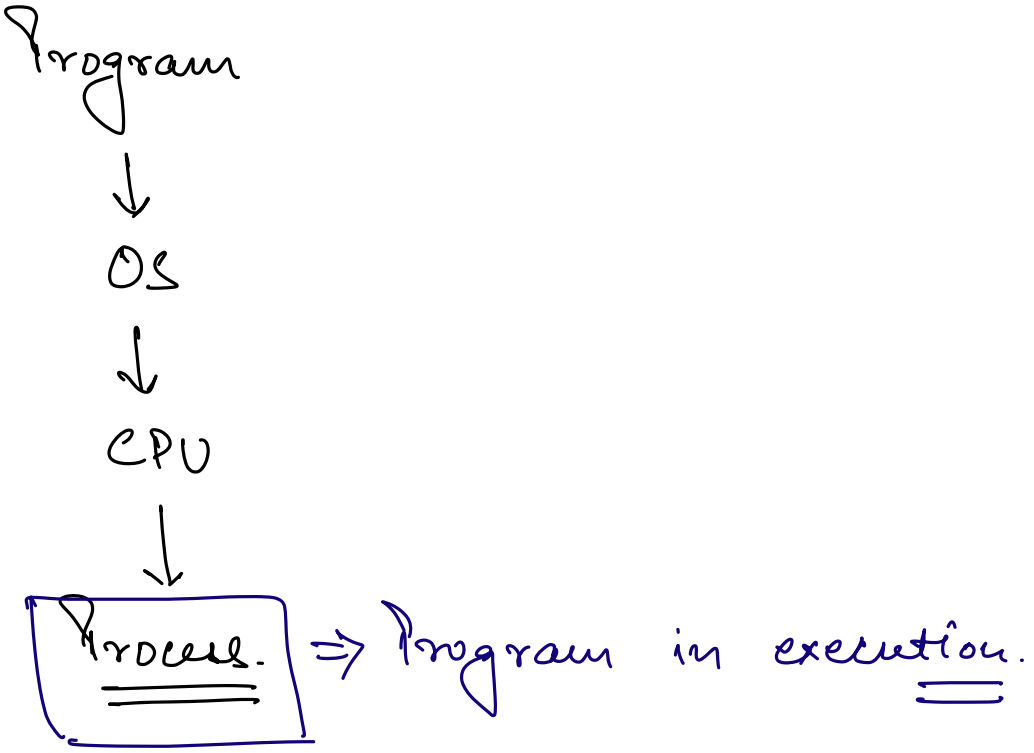
MS Word.



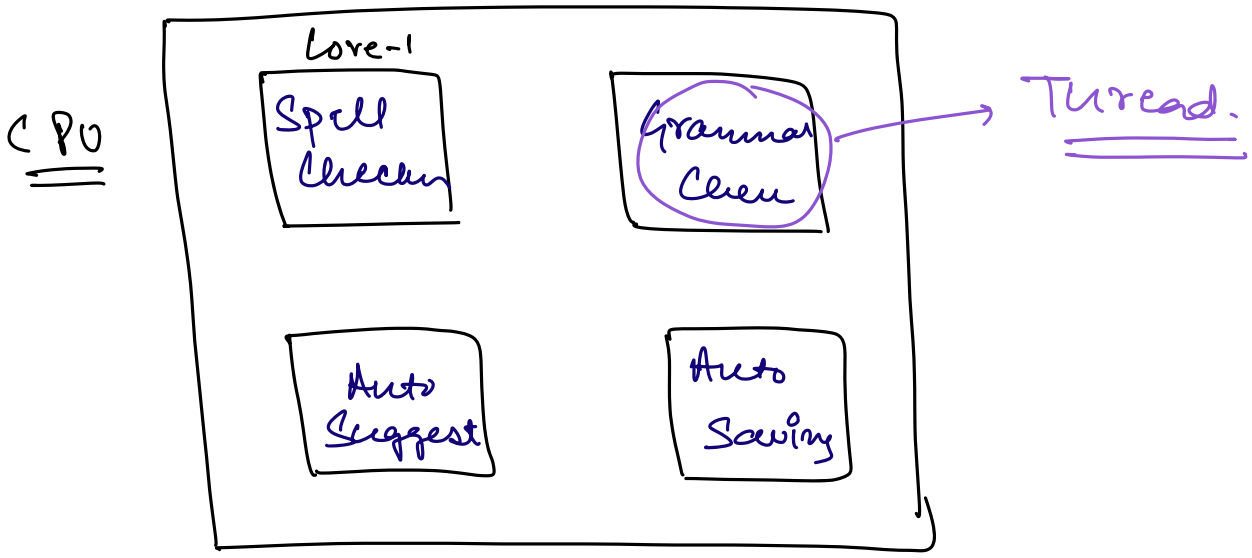
1) User input
2) Spell Checker
3) Grammer Check
4) Auto saved
5) Suggestions.

⇒ Parallely.

MS Word. → Multiple task.



⇒ Every process will have at least task.



▷ main() {

↓ x = 1 ;

↓ y = 2 ;

↓ z = x + y ;

↓ print (z) ;

}

Sequentially.

MS Word

Thread :-1 | Thread :-2 | Thread :-3

↓ | ↓ | ↓

SpellChecker { | GrammarCheck { | Suggestions {

}

OS

Core-1 | Core-2 | Core-3

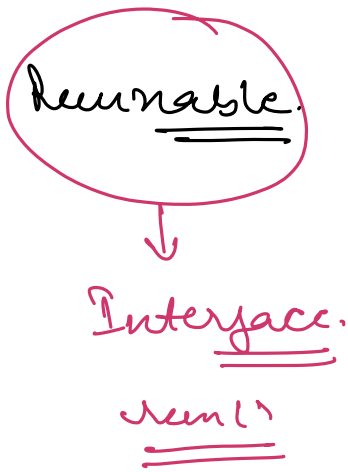
Q: Print Hello World from a Separate thread.

Step-I: Create task Class for the task that we want to execute in a separate thread.

Class HelloWorldPrinter {

}

II) Make task Class implement Runnable.



Class HelloWorldPrinter implements Runnable {

}

III) Implement run() method.

Class HelloWorldPrinter implements Runnable {

    void run() {
        Sout (Hello World)
    }

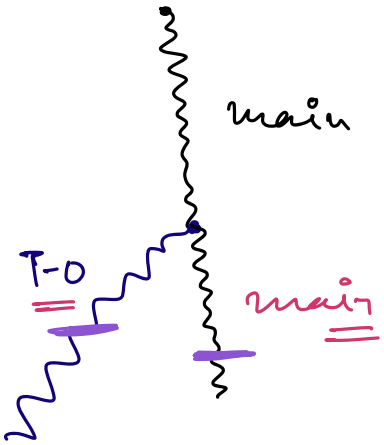
}

Write the code that we want to execute in a separate thread.

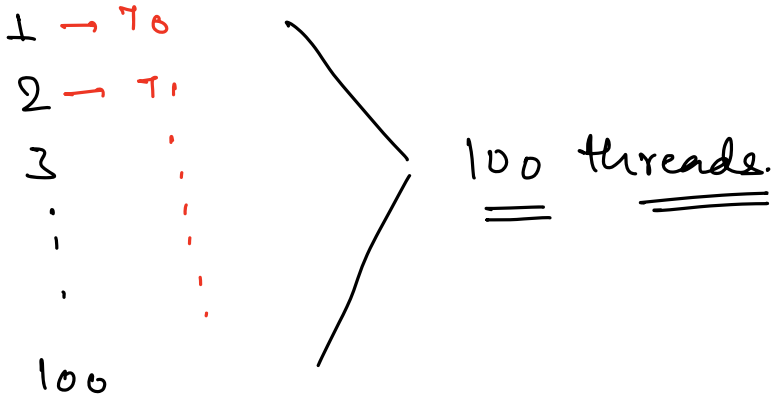
IV) Create a thread object & assign the task to the thread.

V) Start the thread.

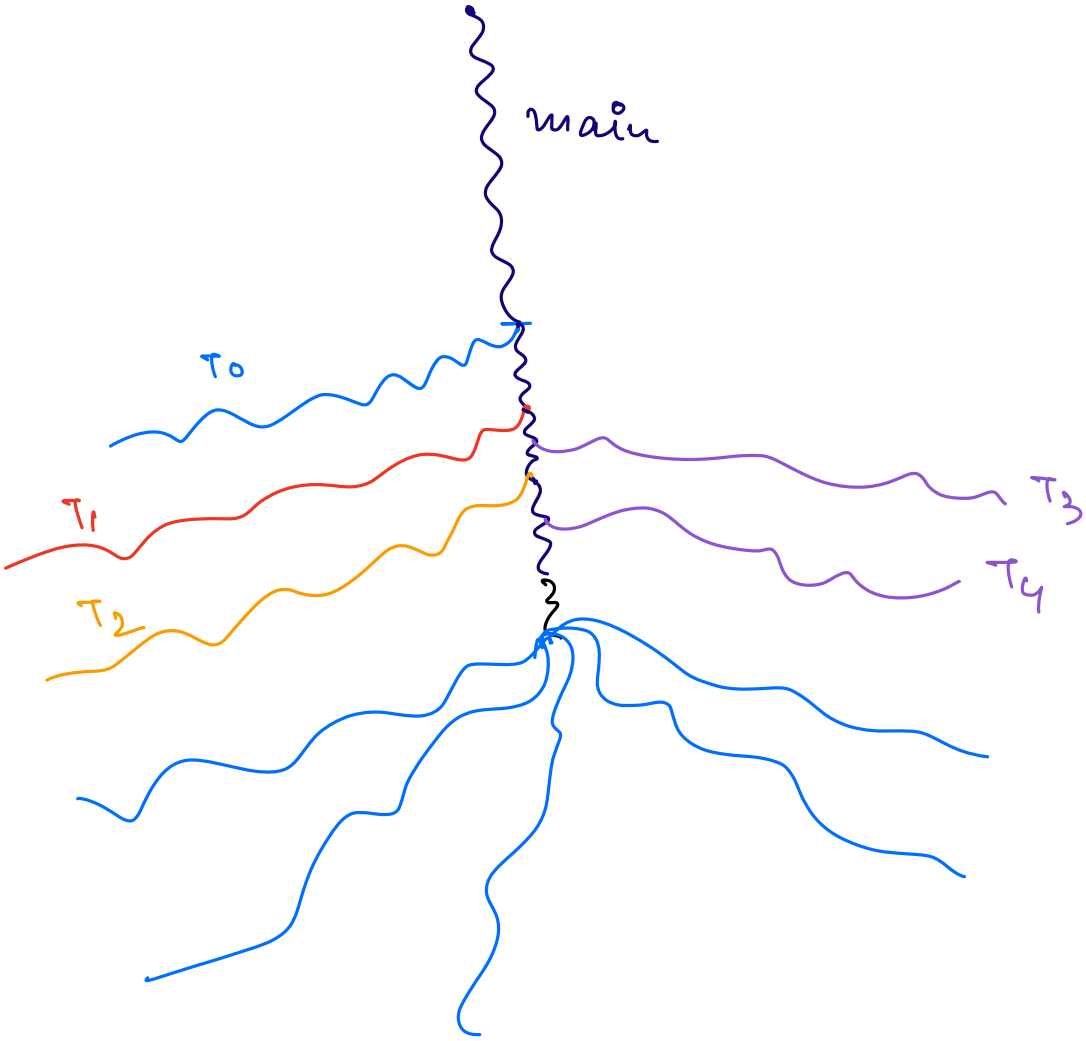
Code.

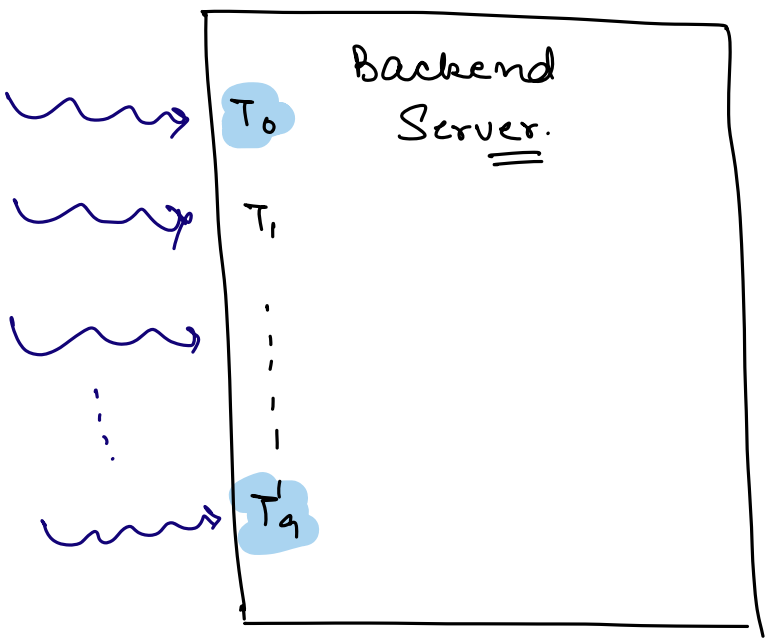


Q. Print no's from 1 to 100, each from a separate thread.

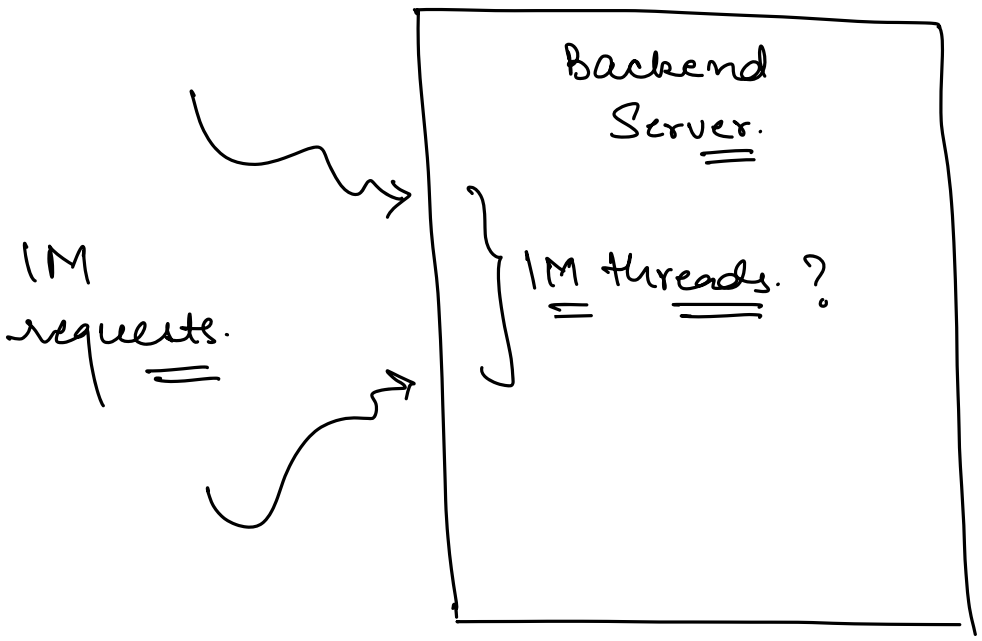


```
Class NumberPrinter implements Runnable {
    private int x;
    NumberPrinter(int n) { x = n; }

    void run() {
        print(x);
    }
}

for(i = 1; i <= 100; i++) {
    NumberPrinter np = new NumberPrinter(i)
    Thread t = new Thred(np)
    t.start()
}
```





⇒ Good user experience



Thread t = new Thread(-);

Object.

time

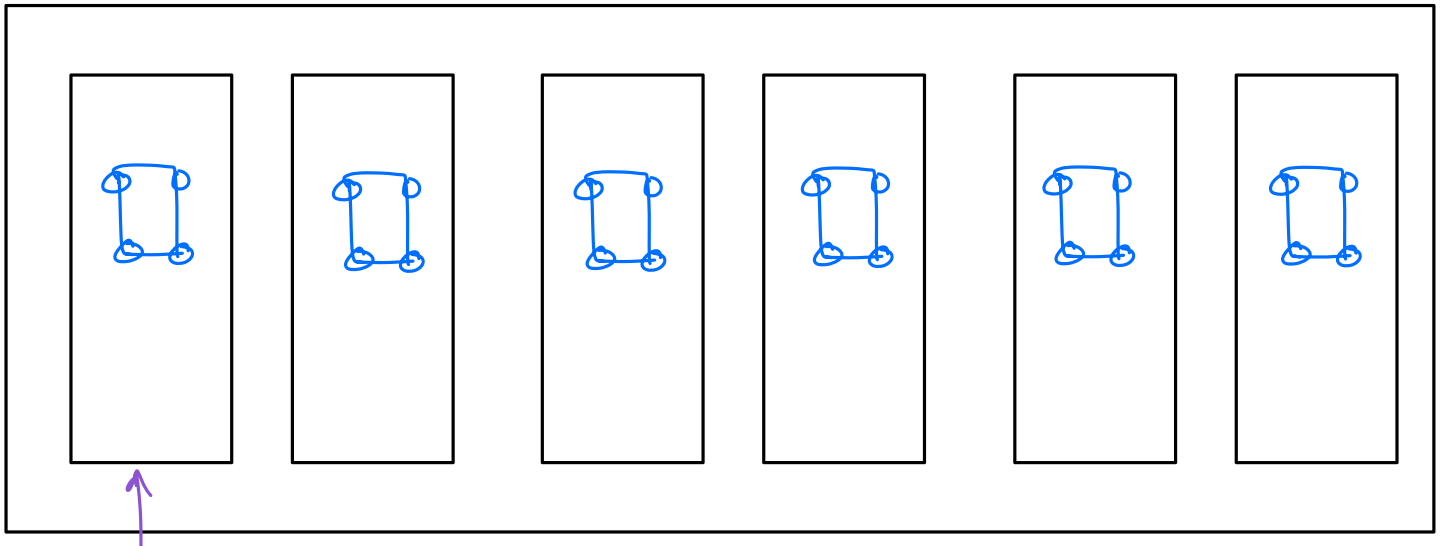
Space

Heap.

⇒ We can't create any # of Objects blindly.

TATA.

↳ Car manufacturing factory.

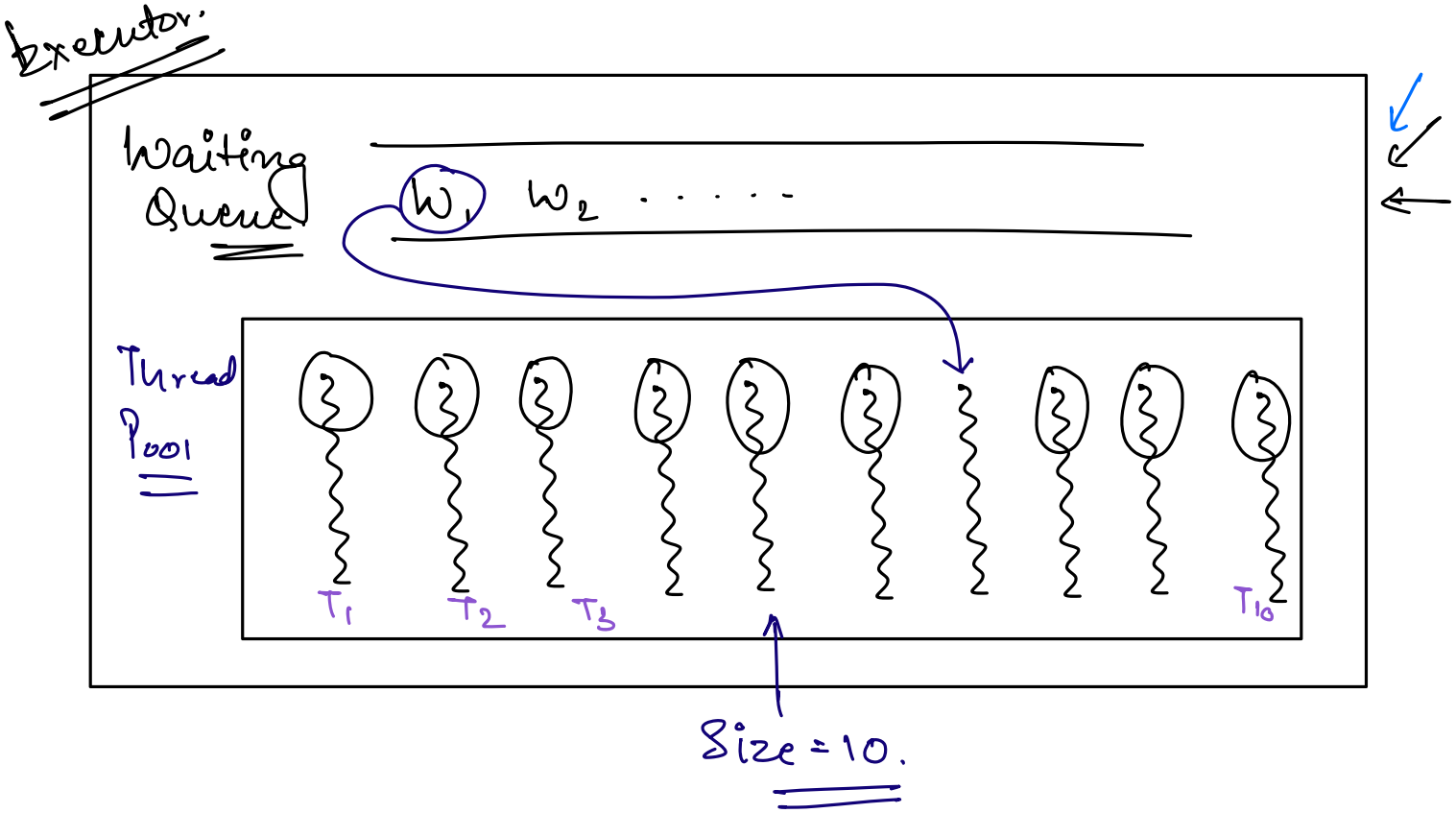


Prod" line.

⇒ 10,000 Cars.

⇒ Efficient usage of Prod" line.

Executor framework & Thread Pool.



Q. Print no's from 1 to 100.

Performance ↑ Cost ($) ↑

⇒ Right balance b/w perf & cost.