OS-3:

Agenda

- → Threads Practical
  - → Hello World
  - → Print 1 to 100
- → Executors and Thread Pools
  - ↓ Print 1 to 100
- → Callables and Futures
  - Multithreaded Merge Sort

HOW TO CREATE THREADS

① Don't Think in terms of what thread I want to create.
Think in terms of WHAT TASK do I want to do separately

For every task, create a class for that.

↓ Noun

→ Class Hello World Printer {

}

② Make that class implement "Runnable" interface.

Class HelloWorldPrinter implements Runnable {

}

③ Implement run() method

```
class HelloWorldPrinter implements Runnable {

      void run() {
            print (" Hello World ")
      }
}
```

write code to do what you want to do sep

main()

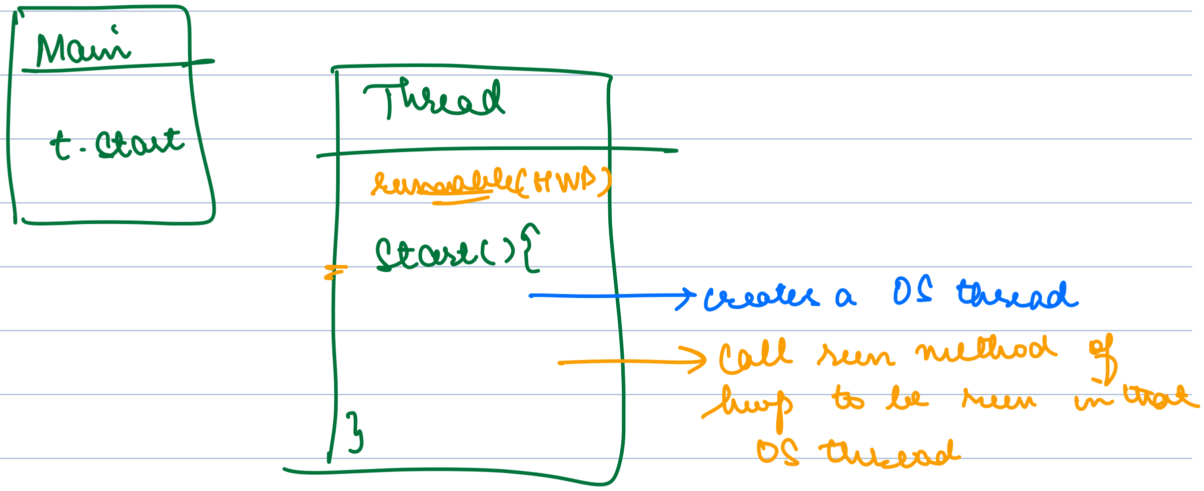
④ From the place where you want to start the task in a separate thread

(i) create an instance of that class

(ii) create an instance of Thread by passing instance create in (i) to it

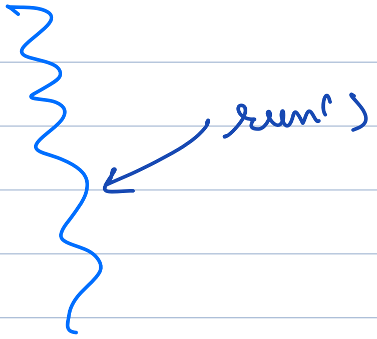
(iii) t.start()

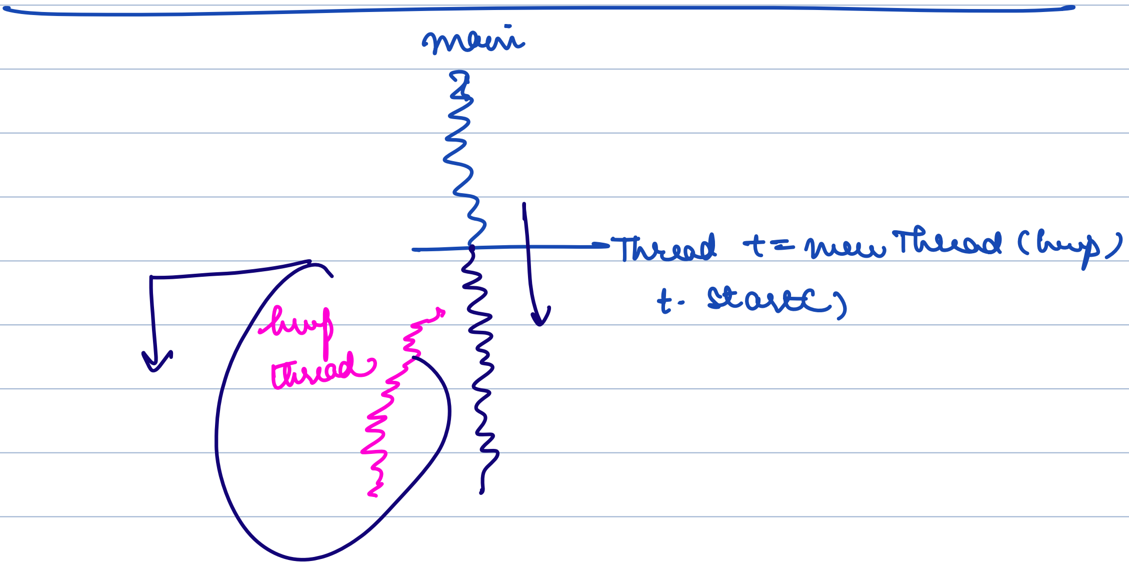
```
class Main {

      main() {
            HelloWorldPrinter hwp = new HelloWorldPrinter()
            Thread t = new Thread (hwp);
            t. start();

      }
}
```



creates a OS thread

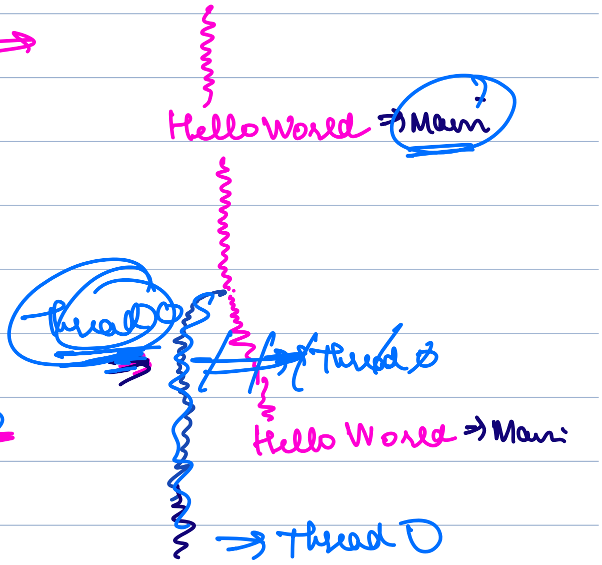
call run method of hwp to be run in that OS thread

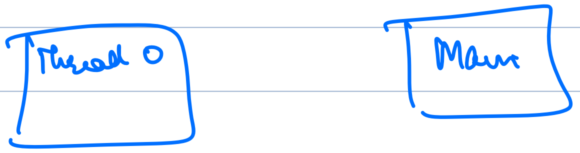




Thread t = new Thread(hwp)
t.start()

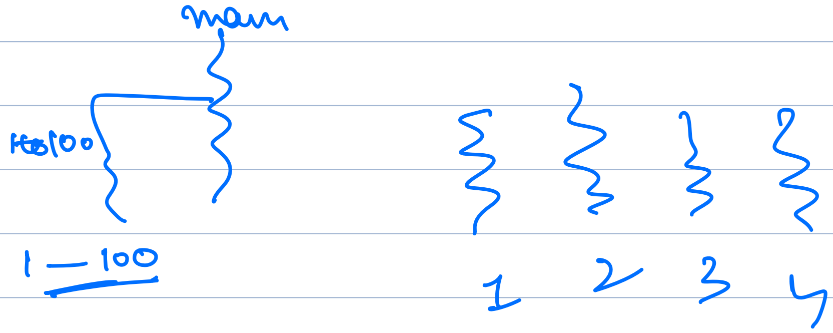
```
main() {
    print (Hello World)
    Thread ——
    t. start ()
    print (Hello World)
}
```





Print 1 to 100, each should be printed from a sep thread

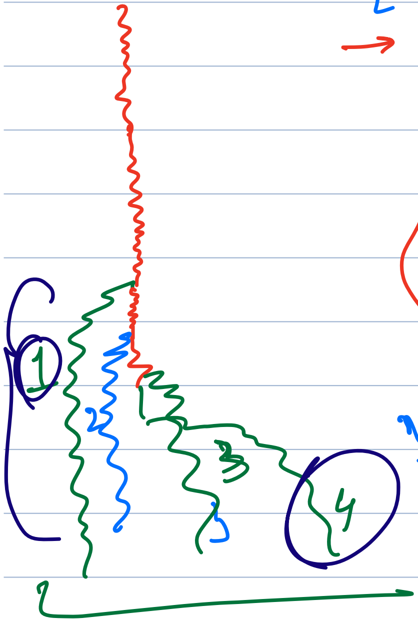
N!



1 Printer 2 Printer 3 Printer

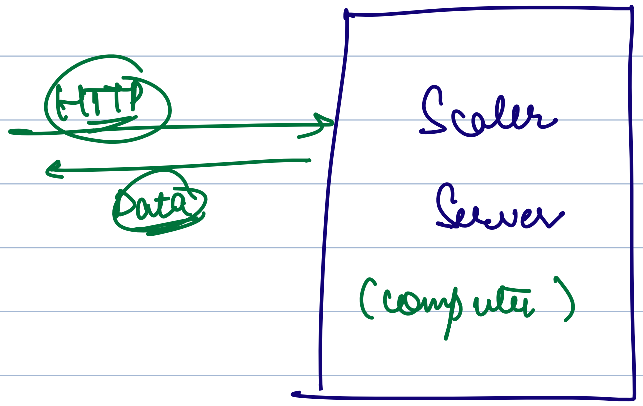
```
Class NumberPrinter implements Runnable {
    int numToPrint ;
    NumberPrinter (int numToPrint) {
            this. numToPrint = numToPrint
    }

    void run() {
            print (numToPrint) ;

}
    }
```

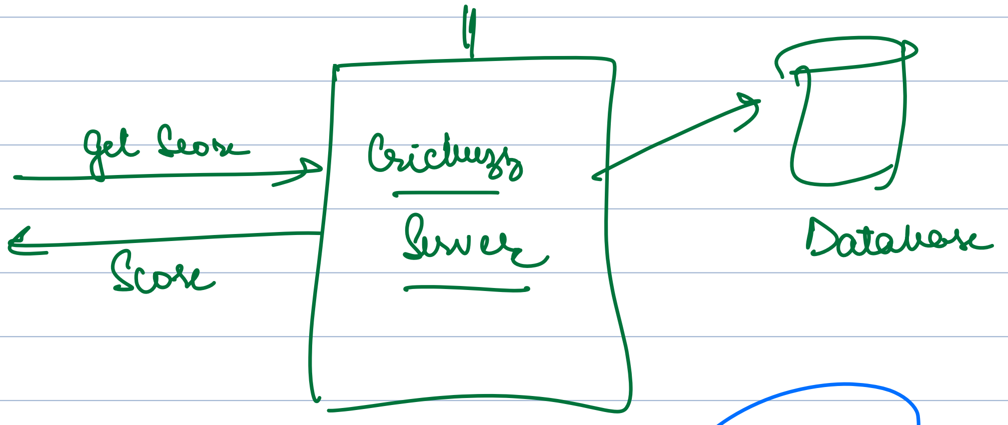
```
Main {
    main () {
        for (int i=1 ; i<=100; i++ ) {
            NumberPrinter np = new NumP( i );
            Hello World
            Thread t = new Thread (np);
            t. start ()
        }
}
```



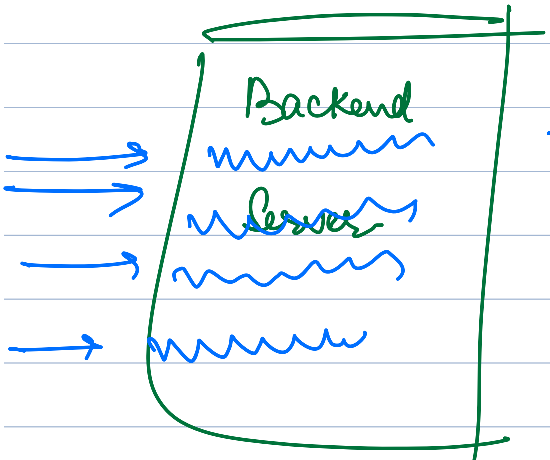
4 → 3
1

Threads ⇒ Handle web requests in a Server



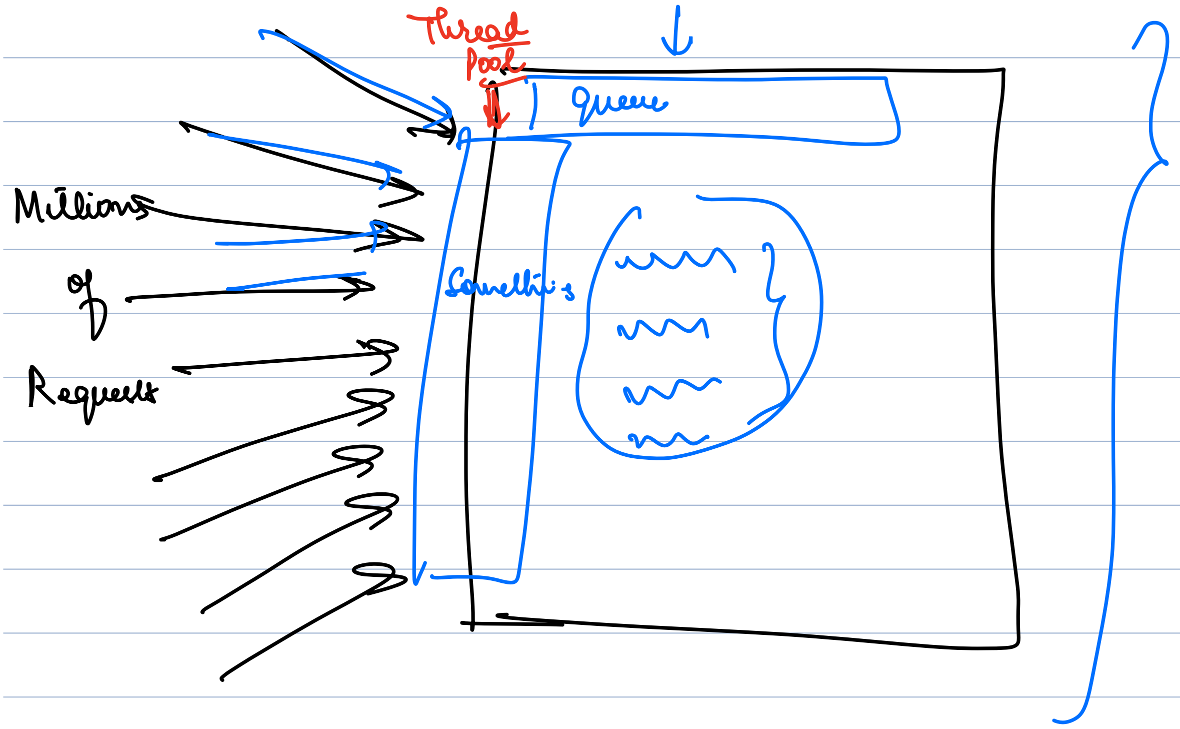


1m rps



→ each req gets served via a sep thread

maxThread Counts = 20 → 4
max Req Count = 20



Executors and Thread Pools

→ Appⁿ developer

Complex Thing

① Define the task

② Creating a thread to run that task

③ Start the thread

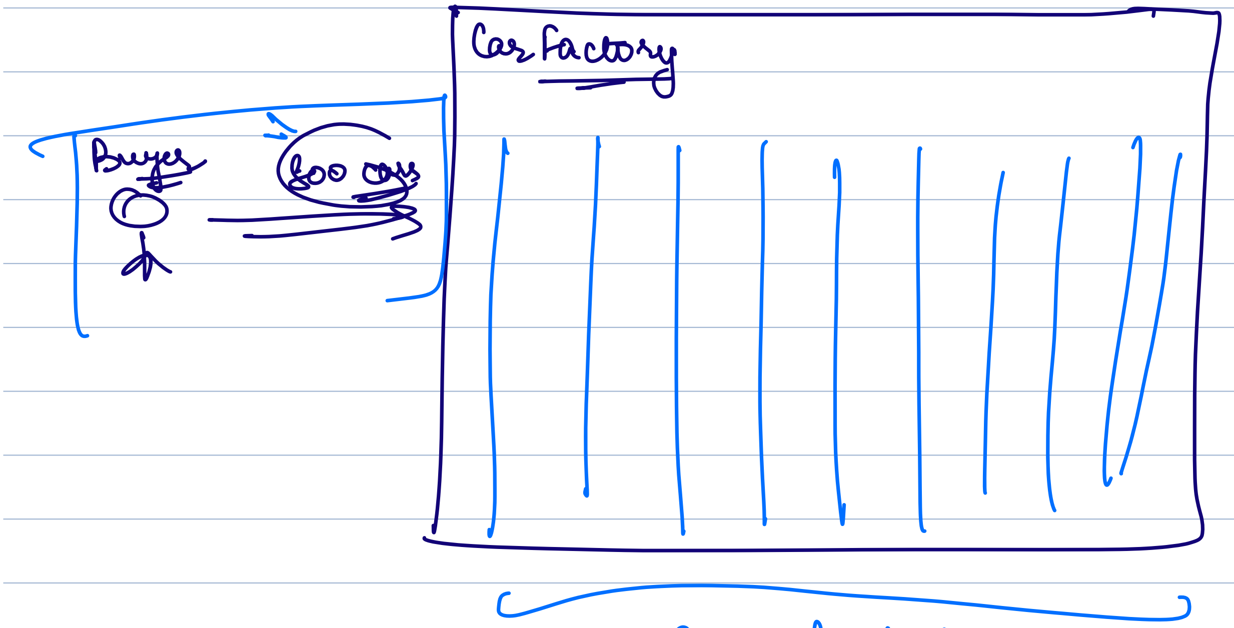
leave it to Lang developers

Executors

→ (1) Division of responsibility

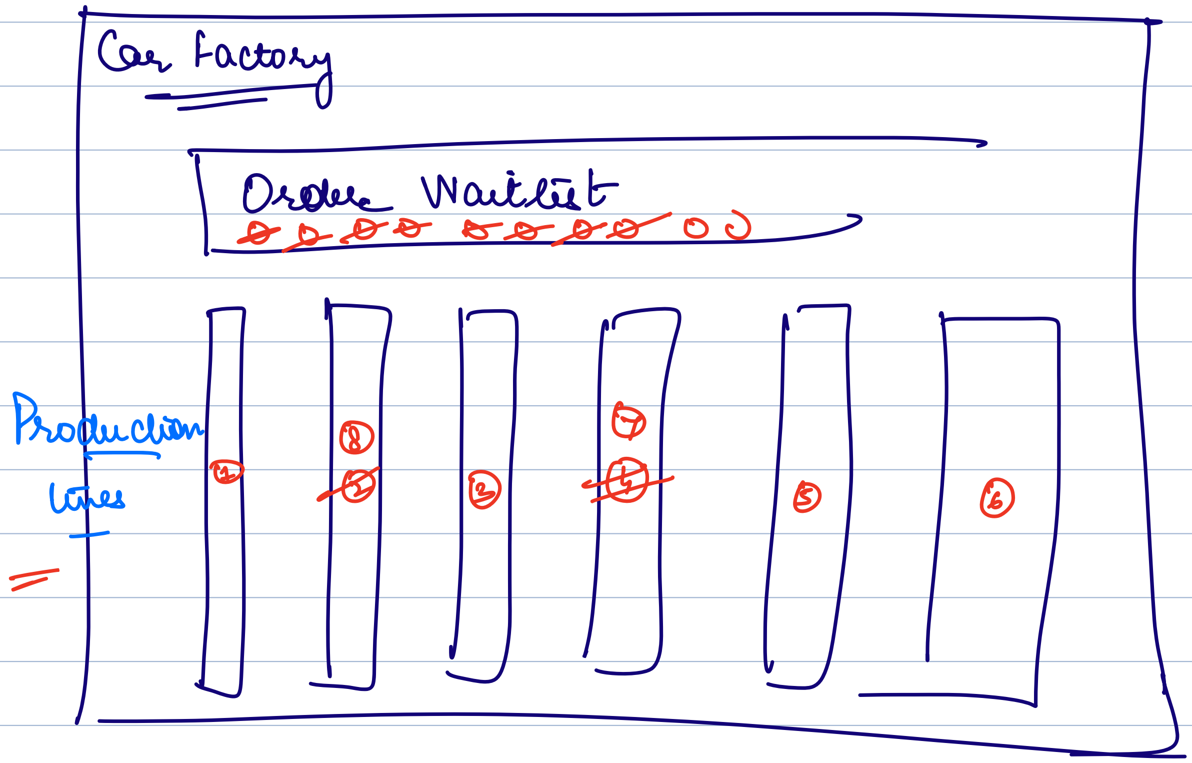
① We define the task
② We give task to executor
③ Executor when it feels most appropriate runs that task

Executors == Car Factory



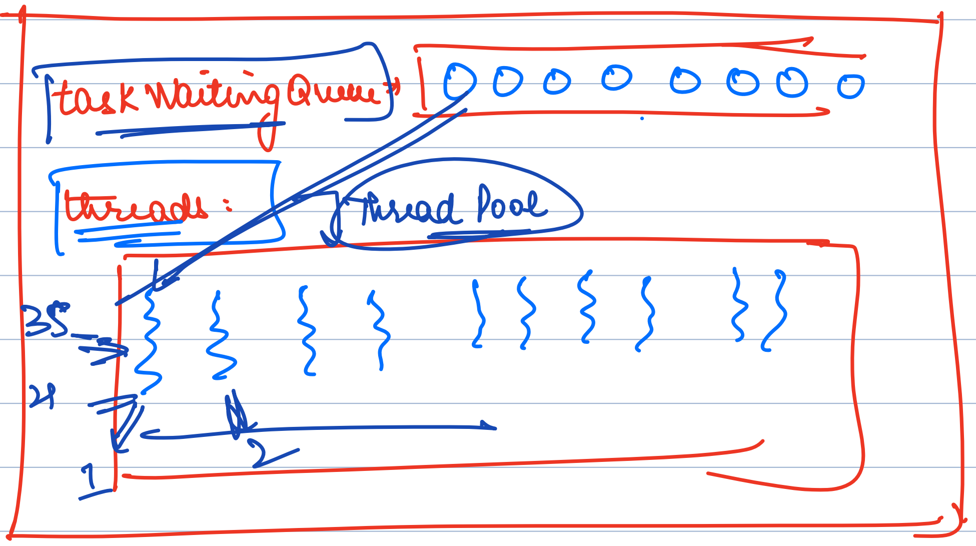
800 production pipeline

→ each 1 car

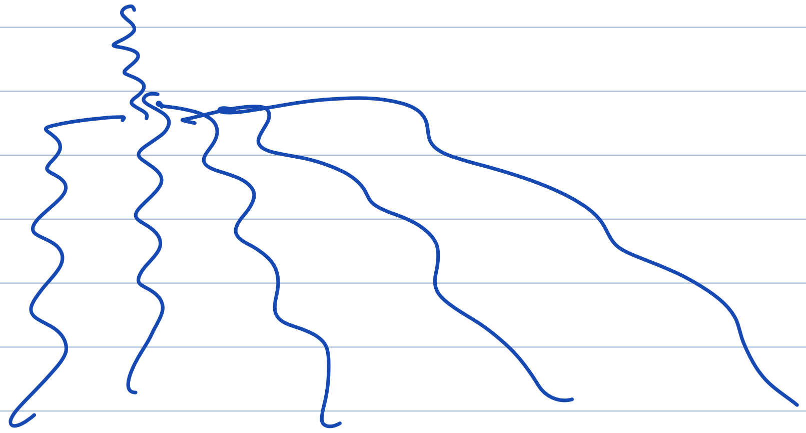


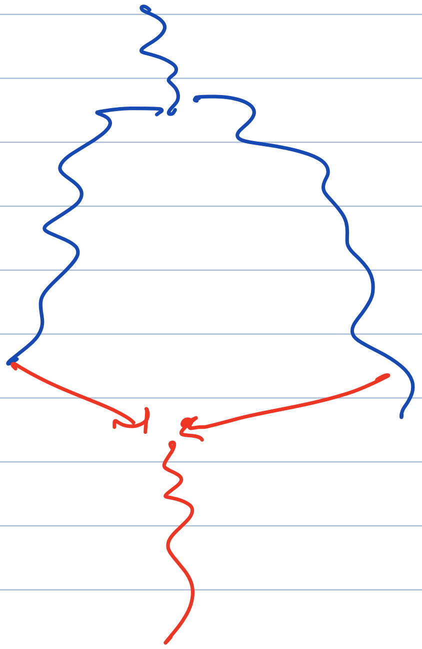
→ whenever I have an available production line, if I have a pending order, I will put that pending order on that line

Executors

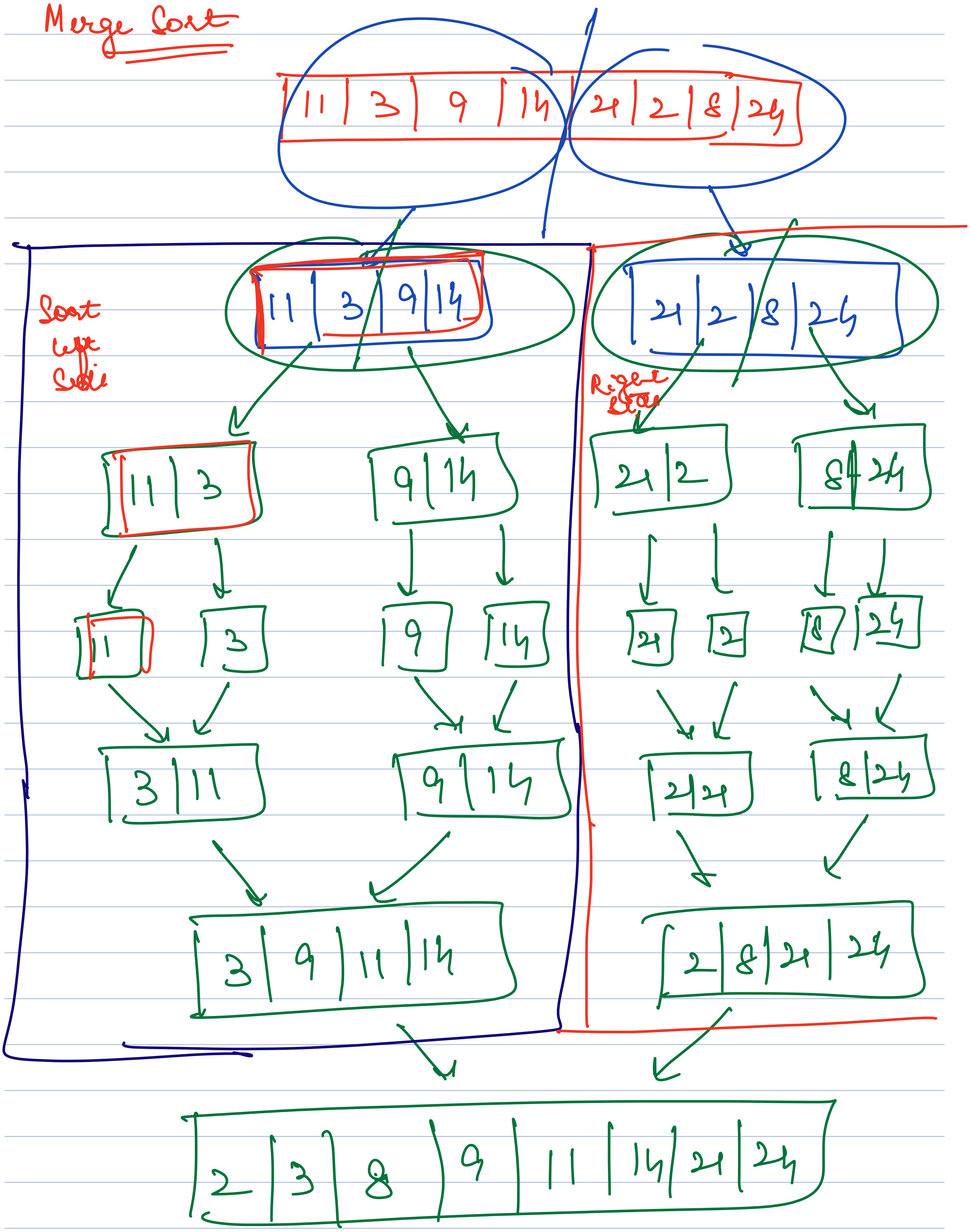


Qn- { If I had to print 1 to 100 via executor with 10 threads }

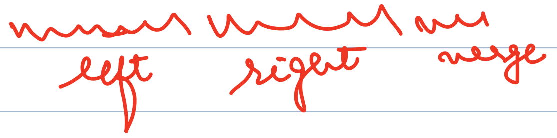


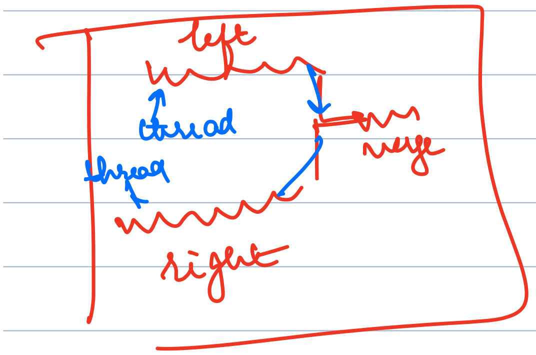


Merge Sort


11 | 3 | 9 | 14 | 21 | 2 | 8 | 24
11 | 3 | 9 | 14
21 | 2 | 8 | 24
Sort left side
Right side
11 | 3
9 | 14
21 | 2
8 | 24
11
3
9
14
21
2
8
24
3 | 11
9 | 14
2 | 21
8 | 24
3 | 9 | 11 | 14
2 | 8 | 21 | 24
2 | 3 | 8 | 9 | 11 | 14 | 21 | 24

Sort left side => 5 se
Sort right side = 5 sec → 10 sec





Callable => interface that allows to create threads that can return data to a parent

Runnable

```
void run() {

}
```

Callable

```
T    call() {

}
```

List < Integer> vector <int>

list < Boolean> vector <bool>

List < Animal>

Generics / Templates

List <Integer>

List <Animal>

Animal

~~Integer~~

class List <~~T~~> {

Animal

~~Integer~~

void add (~~T~~ data) ;

Integer

~~T~~ get (int index);

}

Merge Sort

~~<Boolean>~~

class Sorter implements Callable<List<Integer>> {

List<Integer> call() {

{

}

Code that I want to execute sep

}

Next Class: Complete implementation