1. Good Evening
2. We will begin at 9:07 pm
3. Topic → Coding with threads

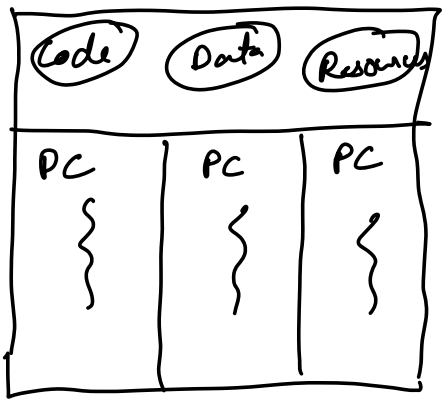
Agenda ✓

0. Recall
1. Introduction to Threads Coding ✓
   - A1 : Print the Thread's name ✓
   - A2 : Hello World via separate thread ✓
   - A3 : Print 1 to 100 via different ✓ threads.
2. Coding with threads in production ✓
   - Executors & Thread Pools, WOs
3. Returning values from threads ✓
   - Callable* ← | Runnable
   - Merge Sort
4. Sharing data ✓
   - Problem of synchronization ✓ ☆
   - AdderSubtractor problem

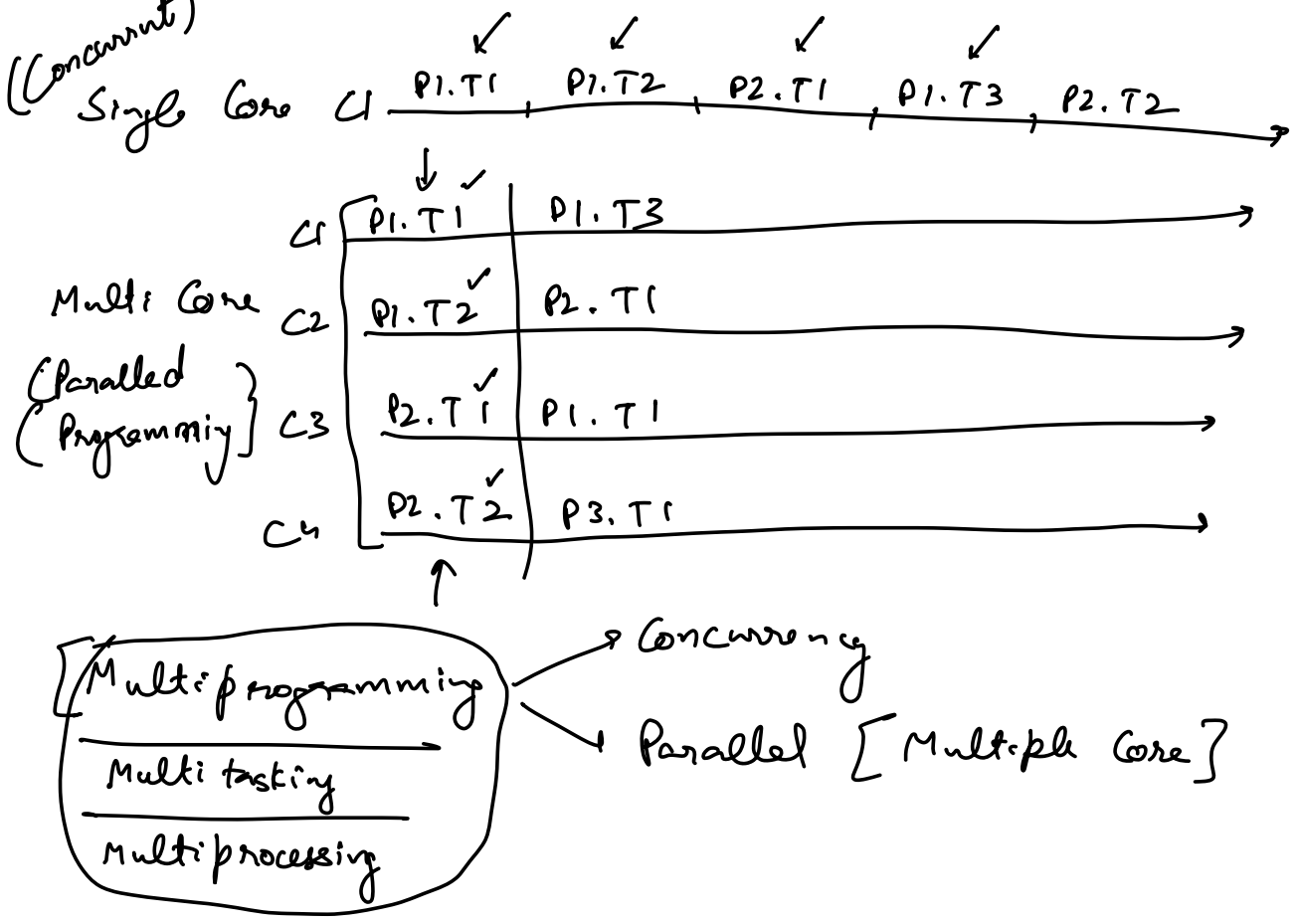
Recall → CPU runs processes

Q. _____

→ Core runs threads ✓



Process



1. Introduction to Coding with threads.

Ai. → Print a thread's name

A2. → Print something via a separate thread

Wisdom: Think in terms of tasks that can be independently done in parallel.

1. For a task, create a class

```
class HelloWorldPrinter {

}
```

2. Make the class implement Runnable interface. ✓

3. Implement run() ✓ method in the class

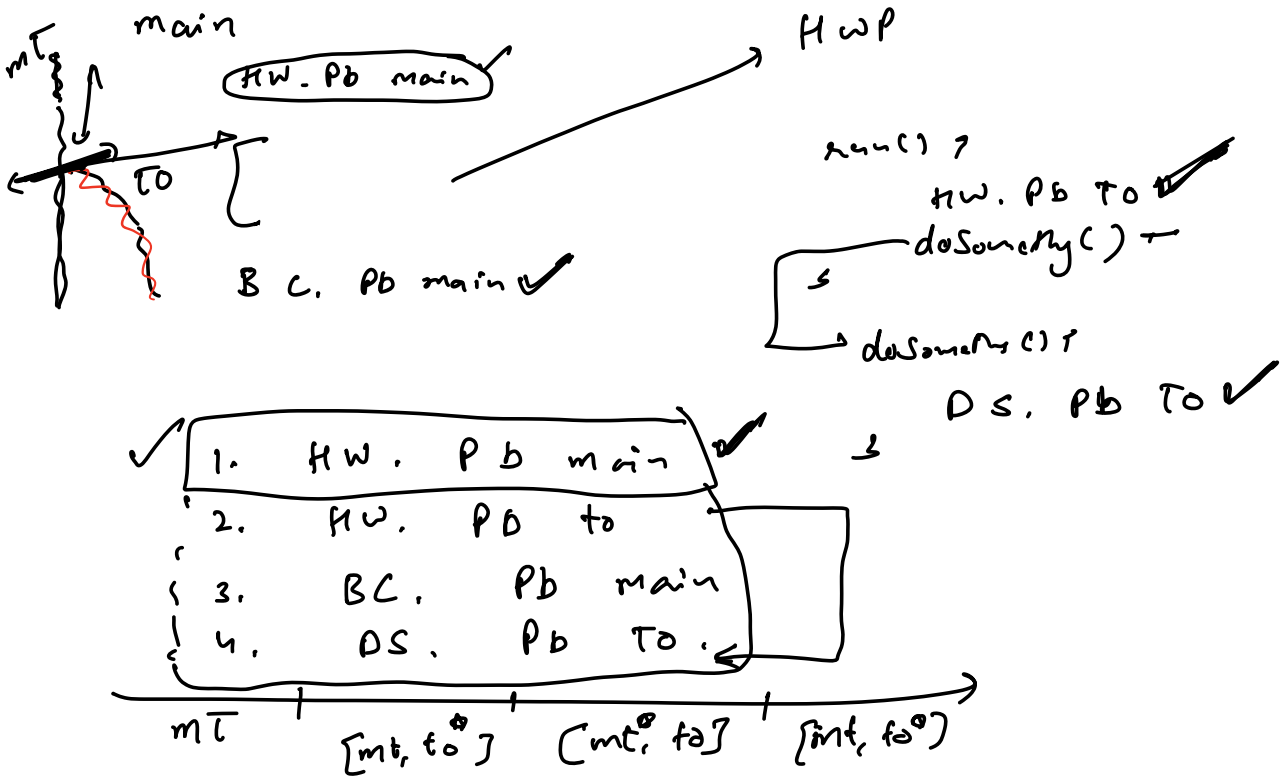
4. [Write code to do the task inside run method. ✓]

5. On the Client [or the place from where you want to invoke the task], create an object of the class.

```
HelloWorldPrinter hwp = new HelloWorldPrinter();
```

6. Create a thread & start it.

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



Run vs t.start()

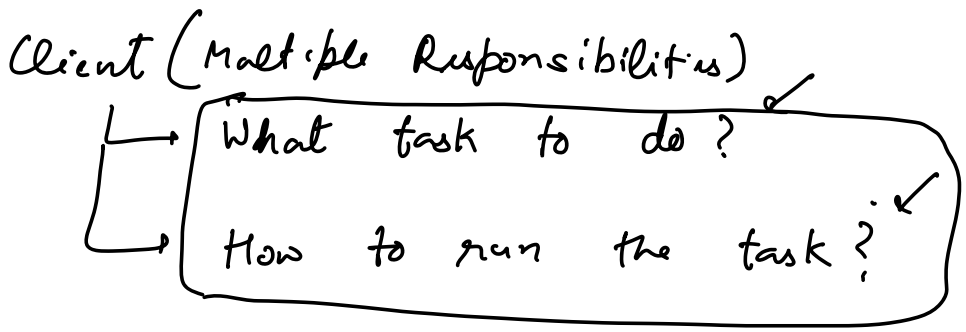
Will call on main thread only

- Creates a new OS thread
- Calls the run method on new thread.

Asgn3 : Print 1 to 100 via different threads

1. class NumberPrinter implements Runnable ?
   constructor
   void run() ?
   {
   }

So far



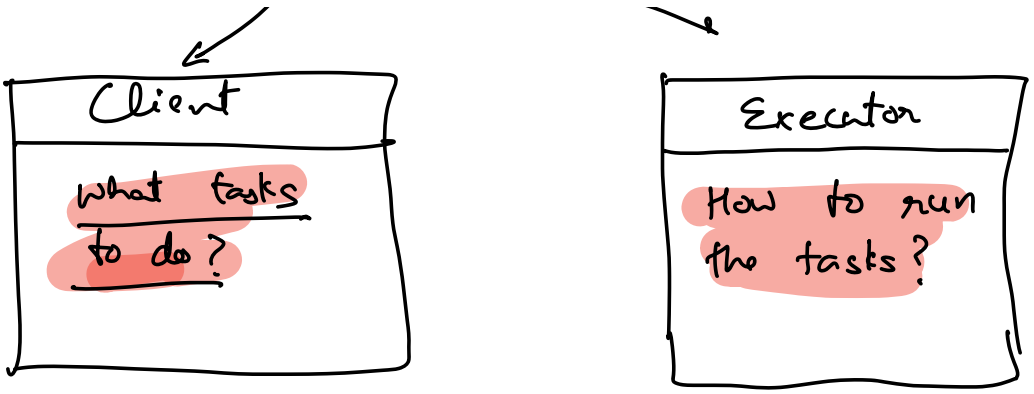
LLD → SOLID Principles

↓

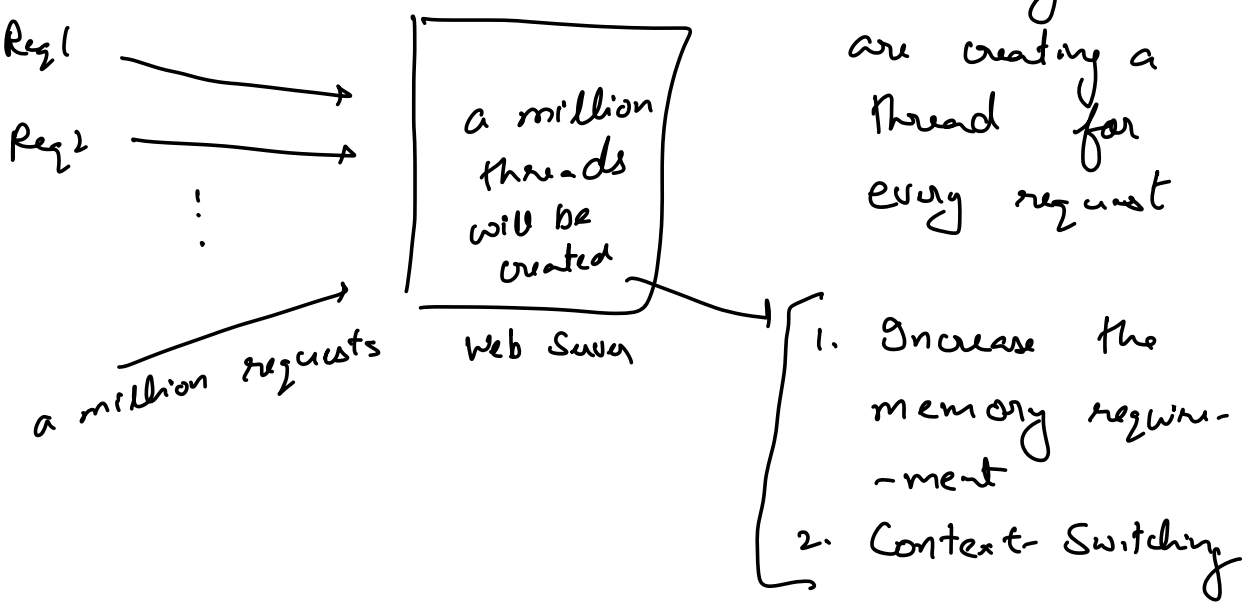
SRP → Single Responsibility Principle

Production Level Code

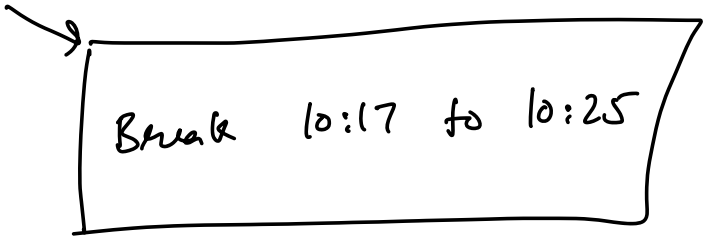
1. Division of Responsibility



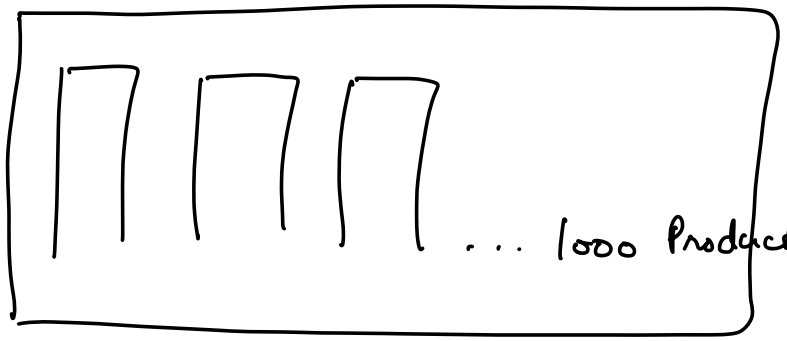
e.g. Saver



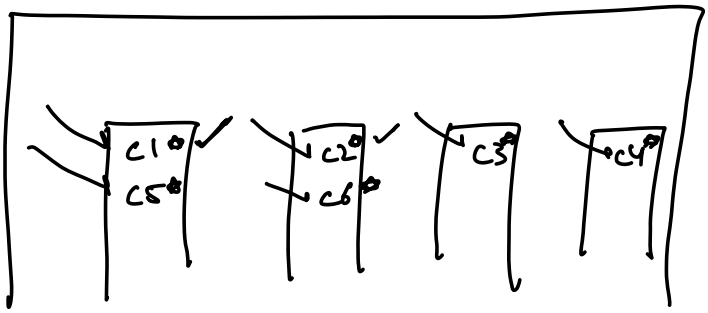
Executor works on the concept of thread pools & waiting queues.

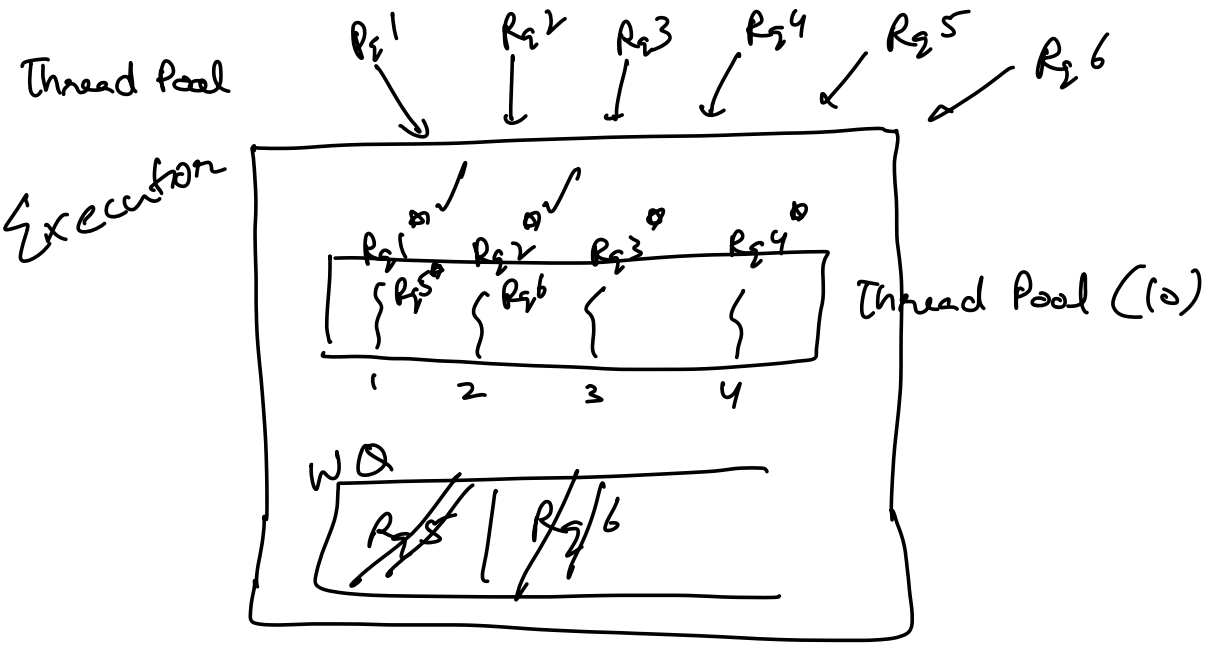


example → Factory creating cars (1000)



→ No reuse of resources.





Executor [Thread Pools & Waiting Queues]

Returning values via threads

↳ Callable.

↳ Generic / Templates

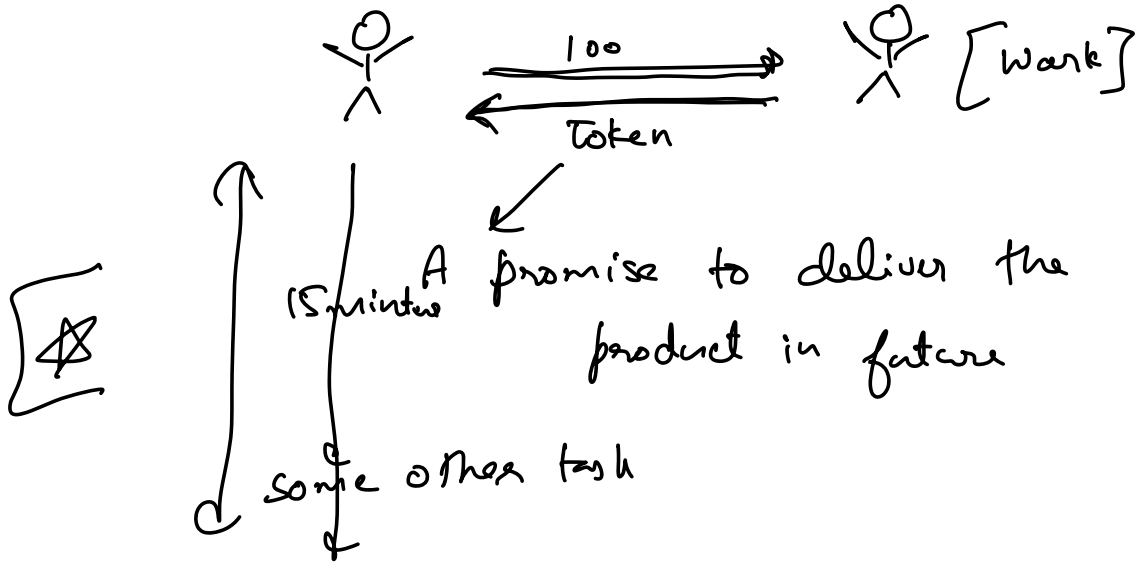
0. Identify the task ✓
1. Create a class ✓
2. Implement Callable<V> interface. Choose datatype according to the value the task must return
3. Implement call method
4. Write the code of task in call method.
5. Create object of class at the point of invocation.
6. Future<String> f = executor.submit(obj) ← Invoke
7. A Future is returned which

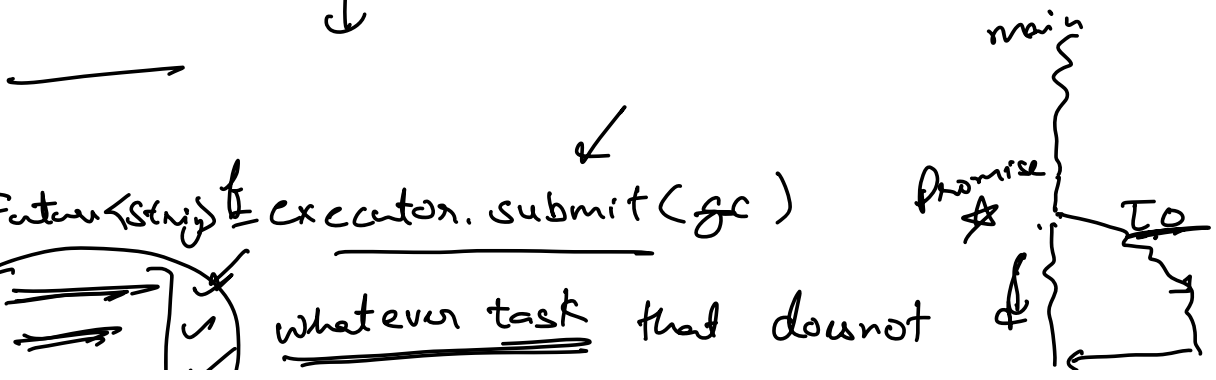
contains the value.

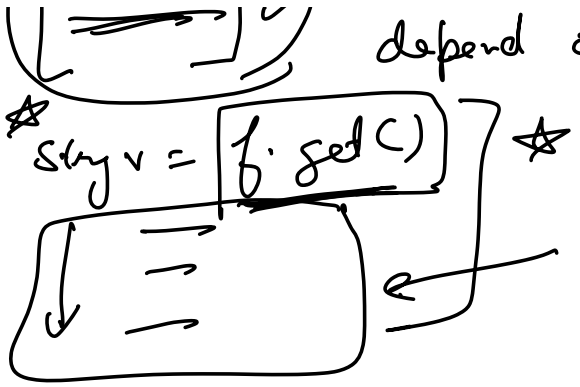
8. Retrieve value from future.

→ String greeting = f.get();

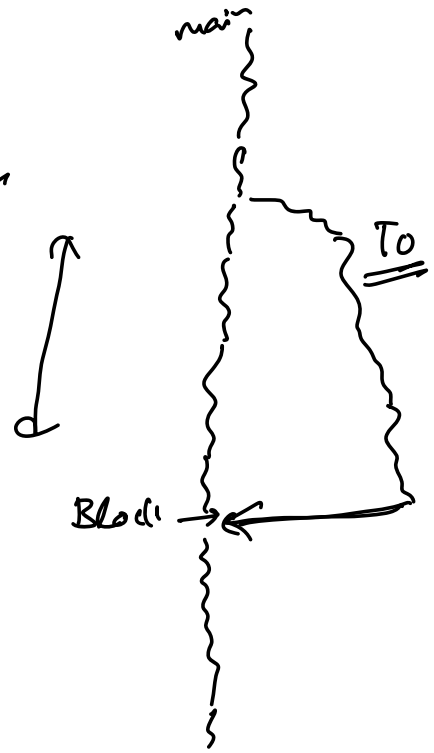
Future



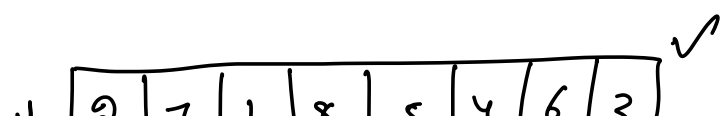


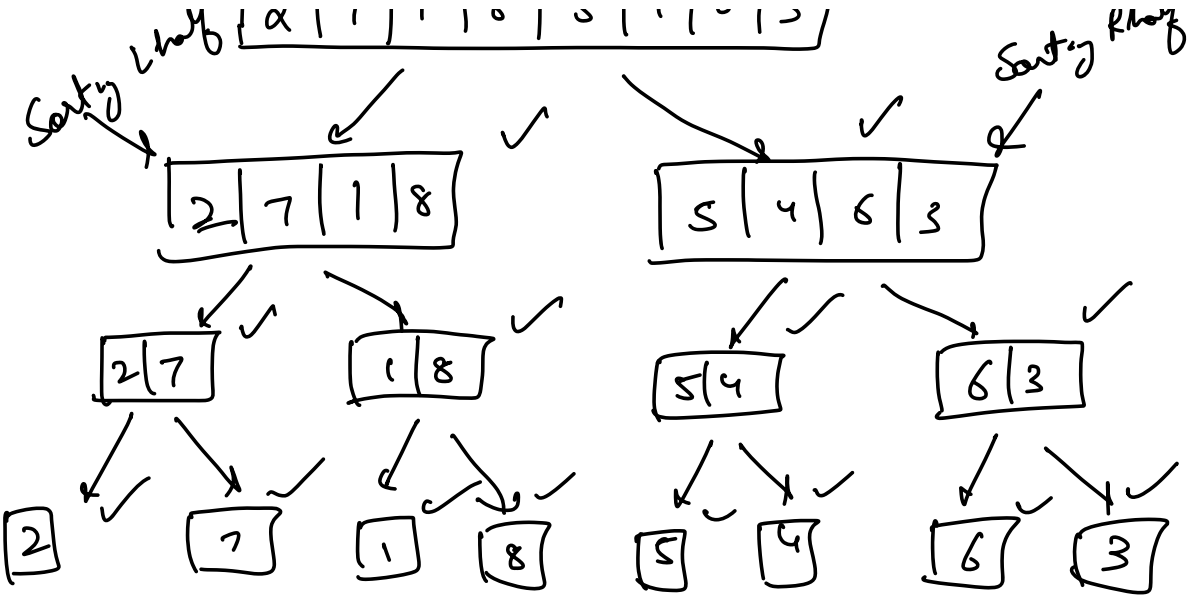


L1
L2
Future f = executor.submit(gc) ✓
L3 ✓
L4 ✓
L5 ✓
String v = f.get()
L6
L7
L8



Merge sort via Callables.





Recall
- Process / Thread
- Single Core vs Multi Core
- Concurrency vs Parallel Programming

→ Intro to Threads → Runnable interface
run
```
Thread t = new Thread()
t.start()
```

→ Executors, Thread Pools, Waiting Queues.
- For division of responsibility
- Reusing the threads
- → Waiting queues → Any task which cannot be assigned within thread pool.

→ Callables <V>

↳ call()

↳ MergeSort

Asgn → Quick Sort