Today

- → Threads
- → HW Problem : Calc 'Factorial' on a separate thread.
  - ↓ Threead 'JOIN'

→ Extra bit + Videos.

- → [Executors and Thread Pools]
- → Callables
- → Synchronization (Intro) [Try if time is left]

Thread lifecycle

1) New : A NEW Thread (or a Born Thread) is a thread that's been created but not yet started. It remains in this state until we start it using the start() method.

2) Runnable

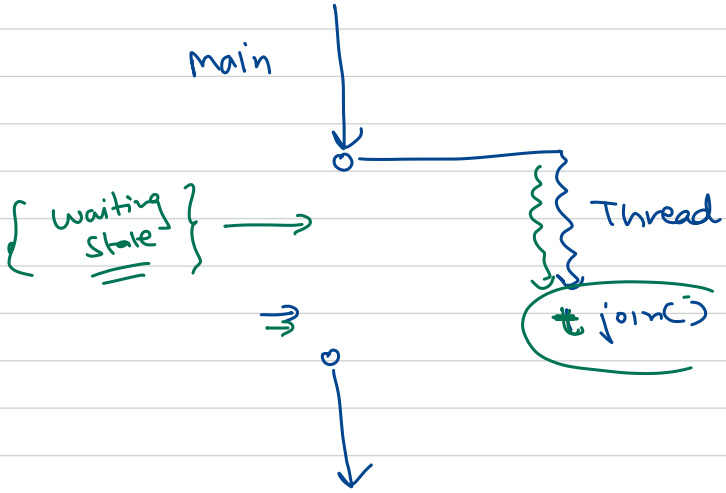
When we've created a new thread and called the start() method on that, it's moved from NEW to RUNNABLE state. Threads in this state are either running or ready to run, but they're waiting for resource allocation from the system.

3) Blocked

4) Waiting

5) Timed waiting

6) Terminated.



Extra Videos

Problem

↓ Multiple Task

↳ Create a Task

↳ Create Threads, manage Threads & decide when a thread will run. } Executor Framework

Number Printer.

T1 — thread(T1)

T2 — thread(T2)

T3 — thread(T3)

⋮

T100 — thread(T100)

4 Cores — 4 threads at max.

{T7, T11, T20, T4}

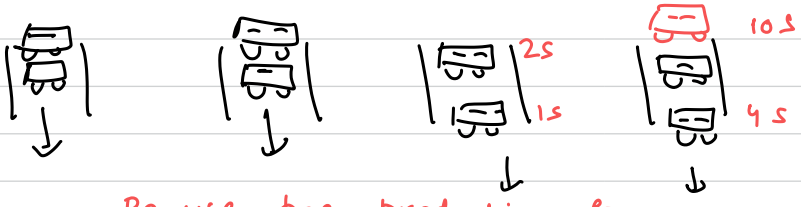
(Create Thread → execute it → terminated)

Re-use the same threads?

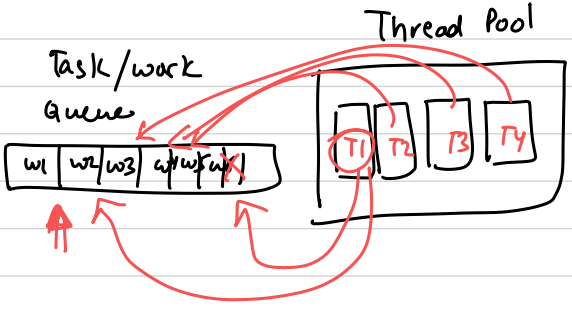
Car Factory

100 Production to 100 cars.

Idea-2



Re-use the production for more tasks of same nature.



W6 ✓ W11 W12 W5

2s 4s 3.15s 10s

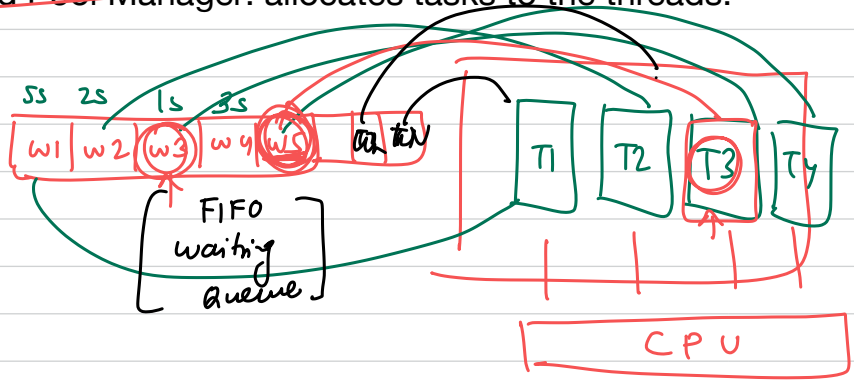
- efficiency
- Re-using, ~~no~~ less overhead of thread creation.

Thread Pool has 3 components:

1. Worker Threads: are available in the pool to execute tasks. They are kept alive thought the lifetime of application.

2. Submitted Tasks : are placed in FIFO queue. Threads pop out tasks from the queue and execute them in the order they are submitted.

3. Thread Pool Manager: allocates tasks to the threads.

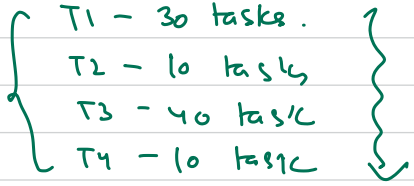


⇒ Fixed Thread Pool

↳ thread pool with fixed no threads.

↳ Re-use threads for submitted tasks

100 Tasks, 4 Threads.



↳ if all threads busy, task will wait inside the queue until the threads becomes available.

- Cached Thread Pool

  ↳ dynamically adjust the no of threads based upon demand

  ↳ Create new threads as needed, but will re-use threads if available.

  ↳ Suitable for handling a large no of short-lived tasks.

Thread becomes available very fast ↓ Re-used.

Callables

→ Runnable Tasks don't have return val. (Runnable I)

```
void run() {
    ≡
}
```

↓

void run()

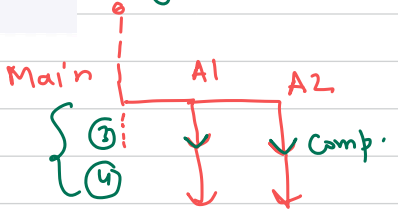
→ Tasks with a Return value. (callable interface)

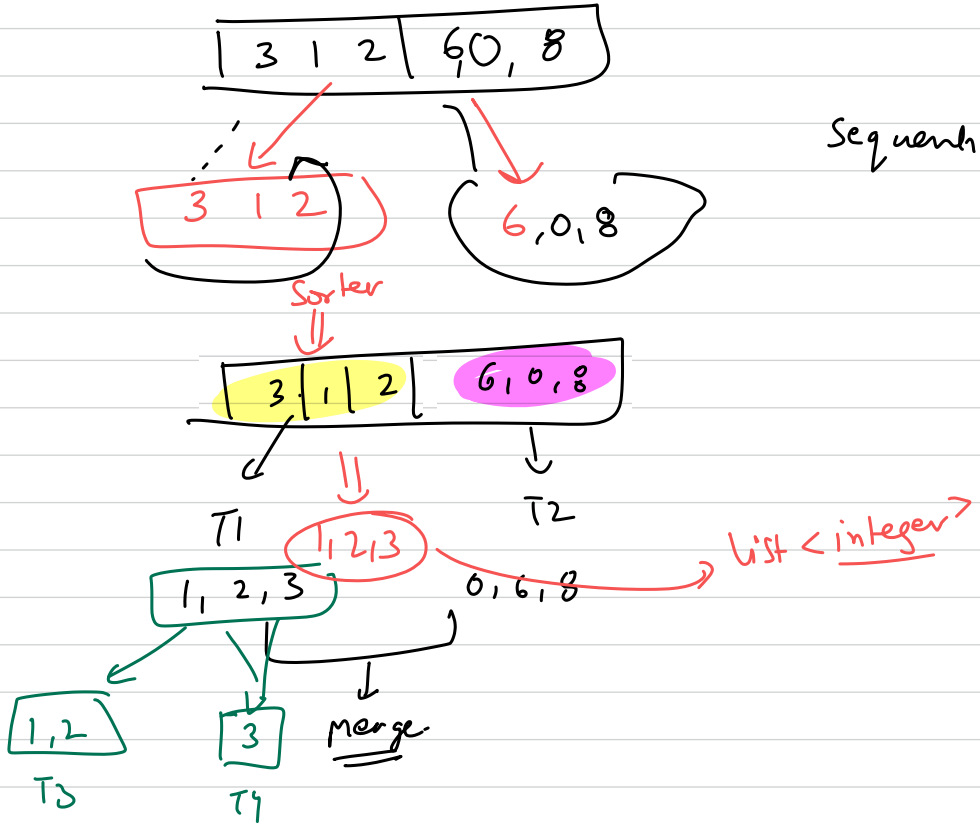
↓

Integer call()

List<Integer> call()

```
Future<Integer> result1 = executor.submit(a1);
Future<Integer> result2 = executor.submit(a2);
// do other things
System.out.println(result1);   ← result1.get(); // Blocking
System.out.println(result2);   ←
```





Sequential.

→ Fixed

→ Cached Thread Pool.