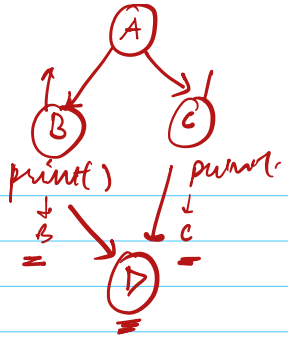
(Starting by 9:05)



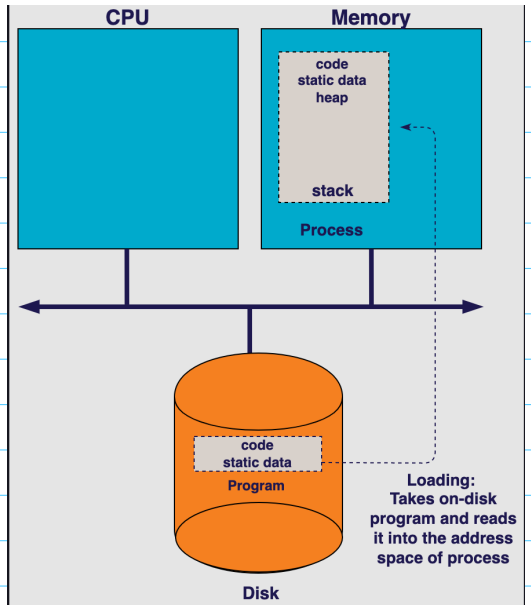
Agenda

1. Recap
2. Thread coding
   - → Thread's name
   - → Print hello world in separate thread
   - → Print 1-100 via different threads
3. Threads in production
   - → Executors and Thread pools
4. Returning values from threads
   - → Callables
   - → Merge sort

PROCESS (a running program)

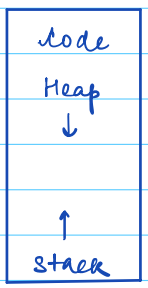
Illusion of many CPUs → Time sharing

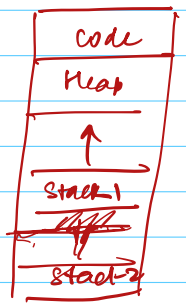
PC → Program Counter or IP → Instruction Pointer
(which instruction executes next)



Address space : memory that process can access

Memory Layout for a process :



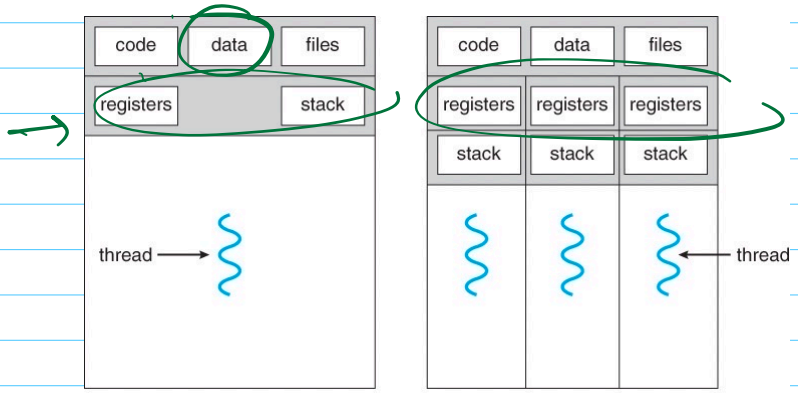


Threads → basic unit of execution.

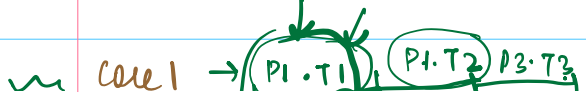
Providing multiple points of execution in a program.

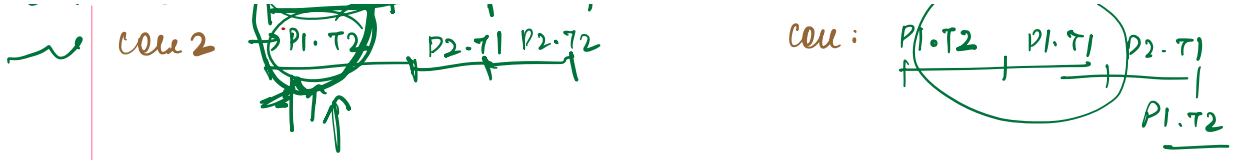
multiple PCs
multiple stacks.

Threads are very much like separate processes EXCEPT 1 difference : → share the same address space and hence the data.



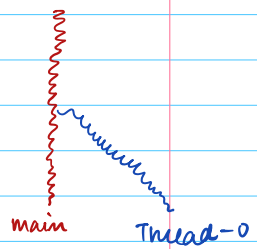
Parallel Programming	Concurrent Programming
REAL	ILLUSION
Multiple cores / PCs	Single core / PC
No need for time sharing	Time sharing
Multiple processes / threads making progress at same time.	More than 1 process / thread in execution but only 1 making progress





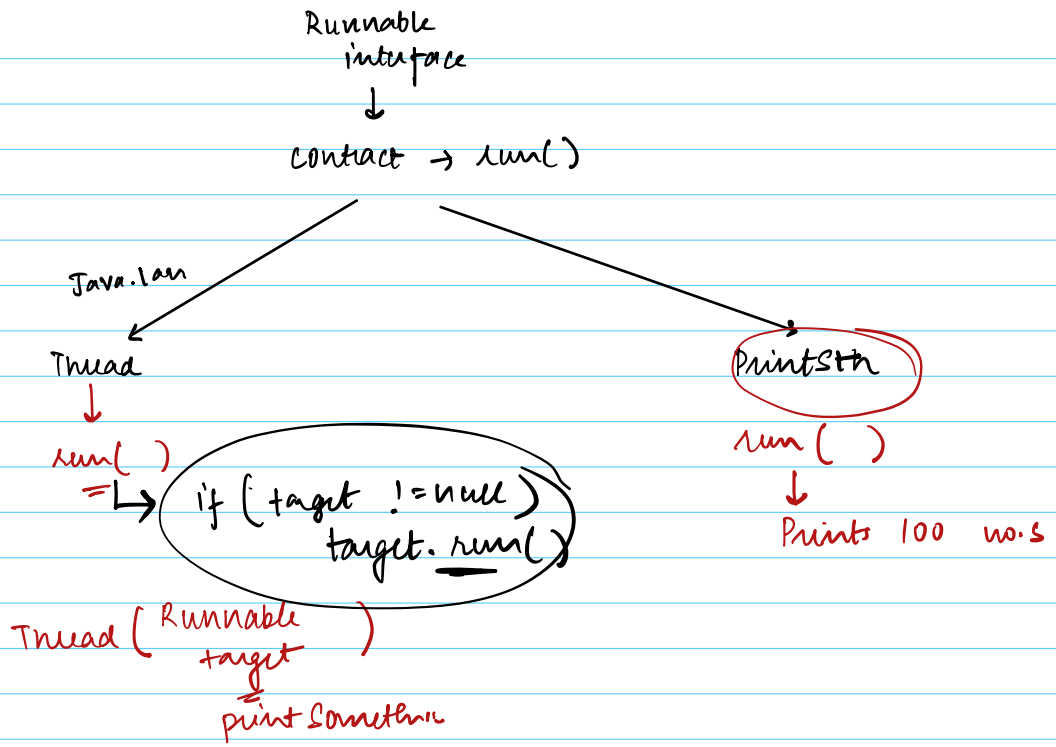

```
class NumberPrinter implements Runnable {

    @Override
    public void run() {

        for ( i = 0 ; i < 100 ; i++ )
            print(i)

    }

}

class Client {

    psvm( ){

        NumberPrinter np = new NumberPrinter( );
        Thread t = new Thread (np);
        t.start( )

    }

}
```

not t.run( ) ?



Client :

1. Creates a new thread and starts it (by calling .start( ))
2. Provides with the method to be executed in separate thread.



t.start() → Create a thread
↳ started
↳ t.run()

run() v/s start()

→ method of interface Runnable

→ code that was supposed to be run in seperate thread

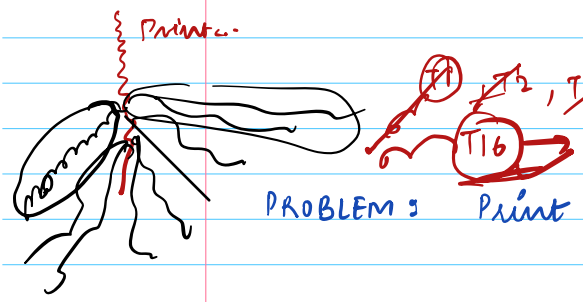
→ method of Thread library

→ Takes care of creating, initializing the thread

→ Finally calls run method.

What if you call t.run() ?

↳ Code runs on main thread.



PROBLEM : Print 1-100 each in seperate thread.

Client

→ i = 1 to 100

→ create a new thread

→ request the thread to print i

Number Printer (Runnable)

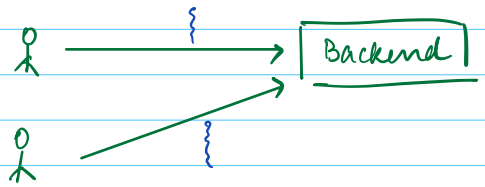
public void run() {

sout (i)

}

How to pass i to run method ?

Production scenario :



clients sending multiple requests.

Obviously backend can't process each request sequentially

What if ? 1 Thread for each request ?

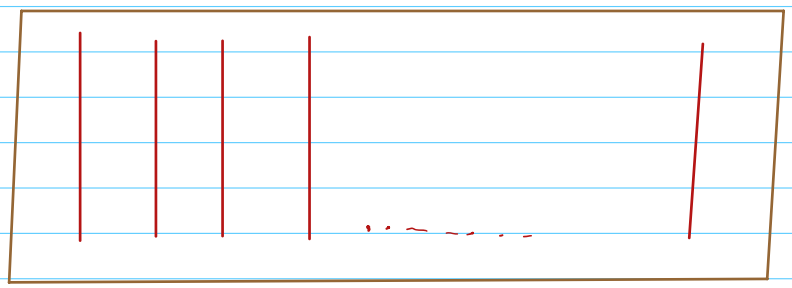
1M Req → 1M Threads

↓

(1) More memory requirement

(2) More context switching

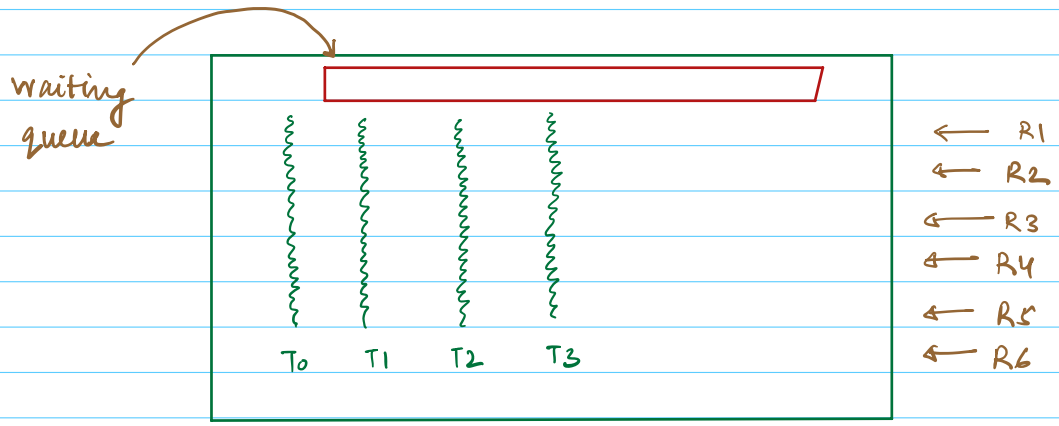
Example : Car Manufacturing Factory



→ 1000 production lines to manufacture 1000 cars ?

5 mins ⤷ 10:20pm

Idea: Create a fixed set of threads and run jobs from a queue on those threads.



Thread pool (fixed set of threads)

```
Executor e = Executors.newFixedThreadPool(10)
```

↳ Creates and manages 10 threads.

Running a Runnable:

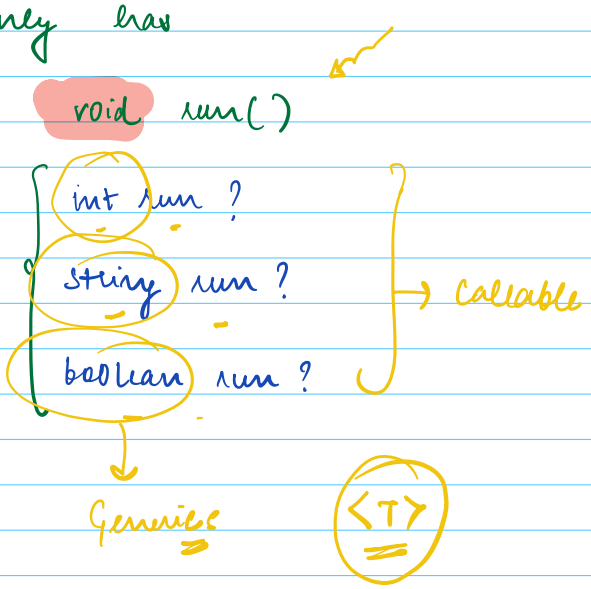
```
e.execute(numberPrinter);
```

Demo!

Returning values from a Thread

Threads should return some value to main thread

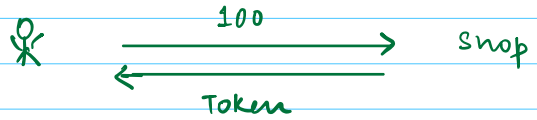
Callable only has



Steps :

1) Identify task
2) Create a class implements Callable<V>
3) Implement call method
4) Create object of class and pass it to executor
5) Future<V> is returned.
6) Get the value

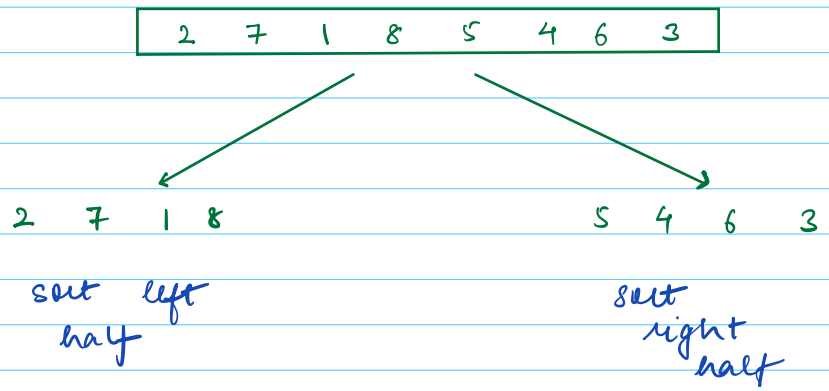
What is Future ?



↓

A promise to deliver product in future.

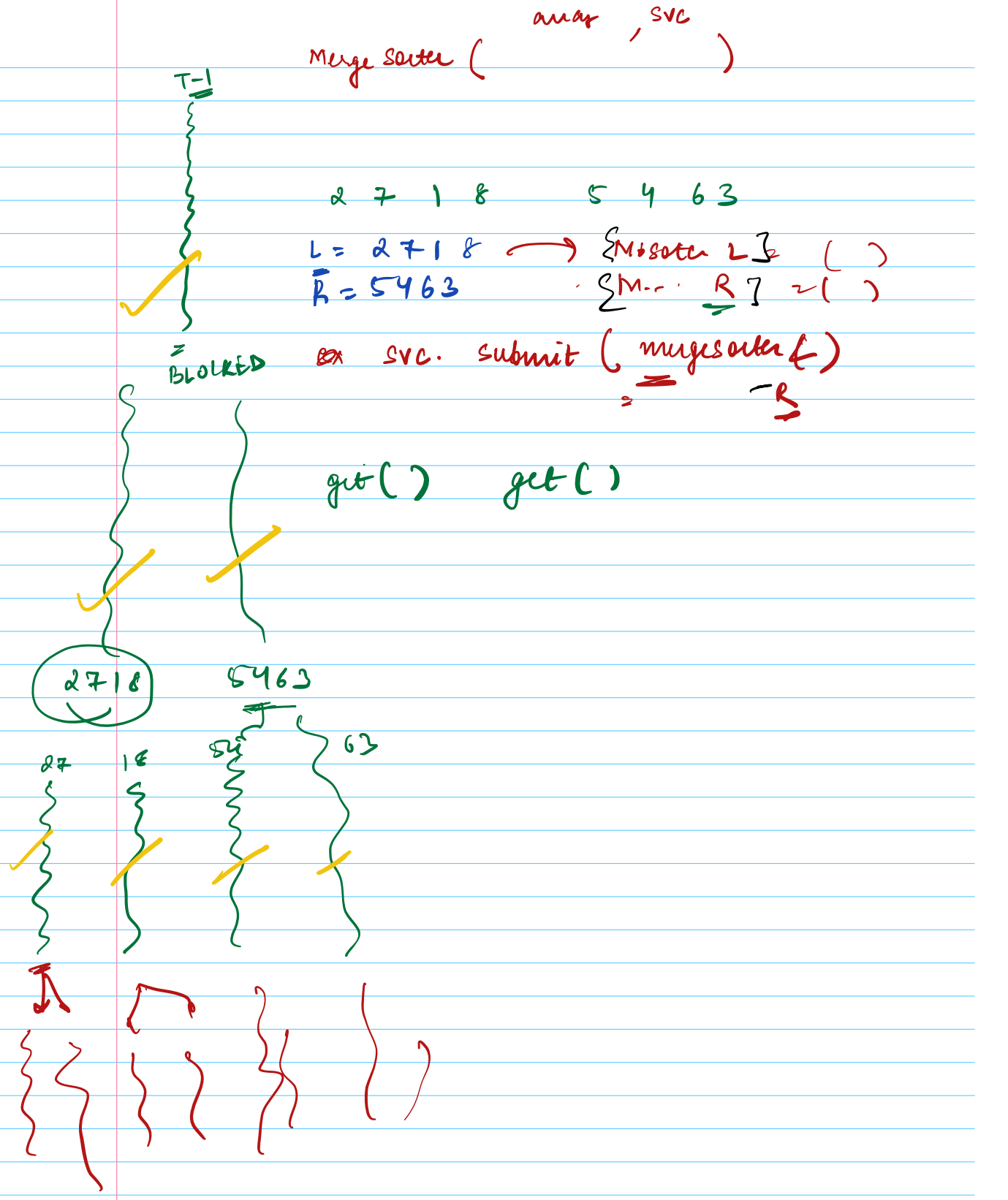
Merge Sort via Callables



can work in parallel

↓

merge finally.


array , svc
Merge sorter ( )
T-1
2 7 1 8 5 4 6 3
L = 2 7 1 8
R = 5 4 6 3
BLOCKED
svc. submit (mergesorter)
get ( ) get ( )
2718
5463
27
18
54
63