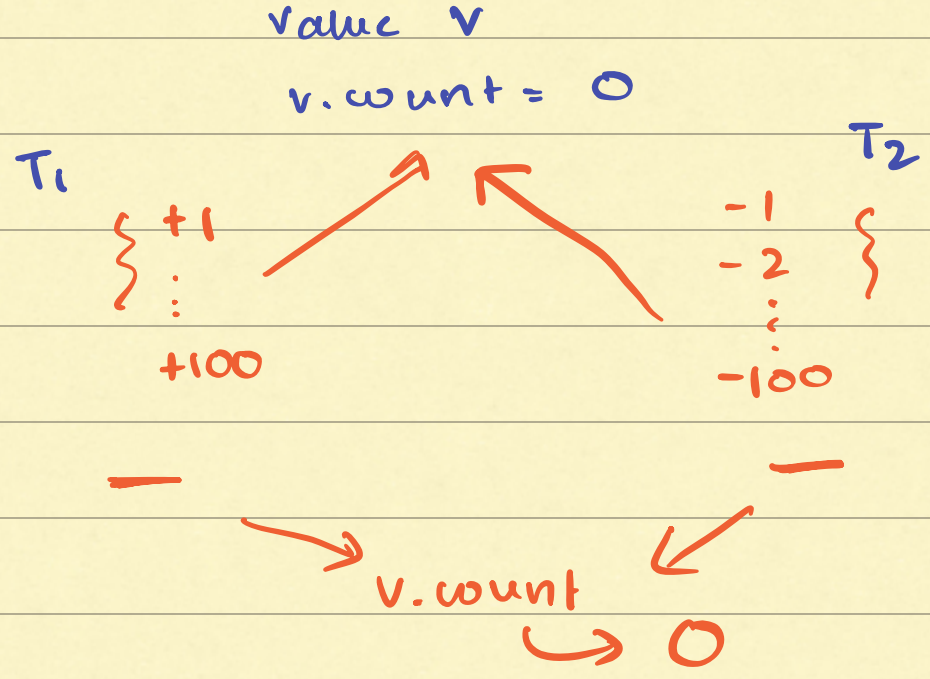
Agenda

- Why synchronisation error happens!
- Mutex
- synchronised keyword
- synchronised method
- Producer/Consumer



1. Critical Section → any part of code dealing with shared memory or data is called CS.

① sout("Hello");
② v.count += 1;
③ sout("v = " + v);

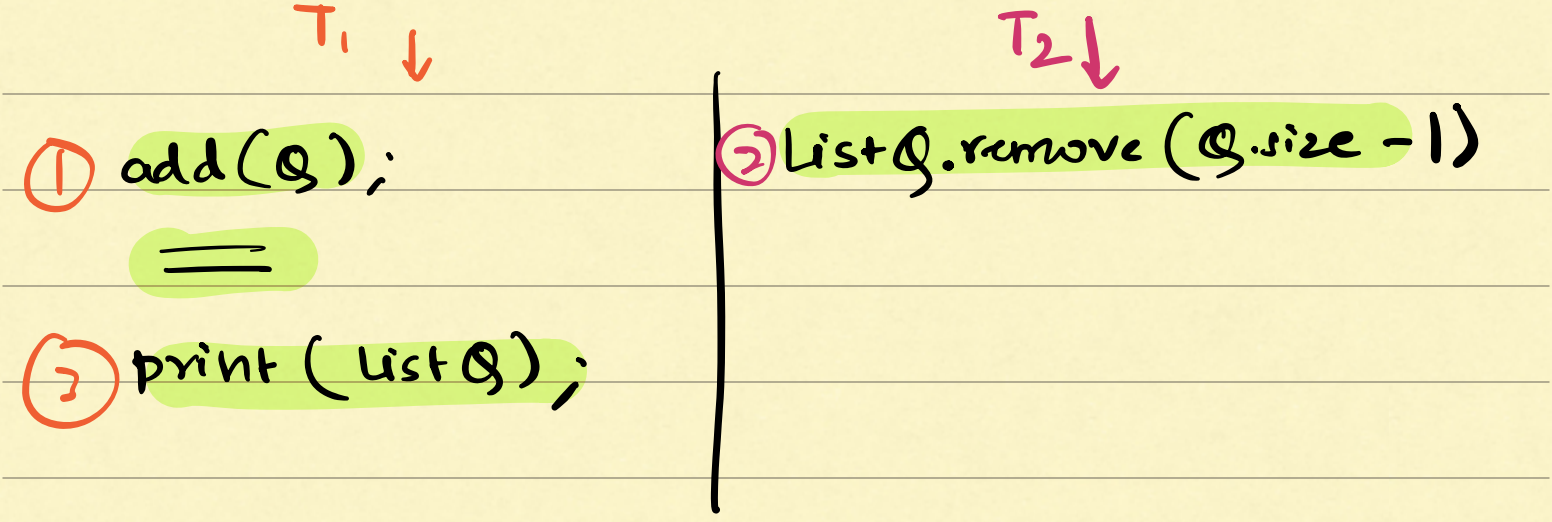
} Critical Section

① sout("Hello");
② v.count -= 1;
③ sout("Hello2");
④ v.count *= 1;

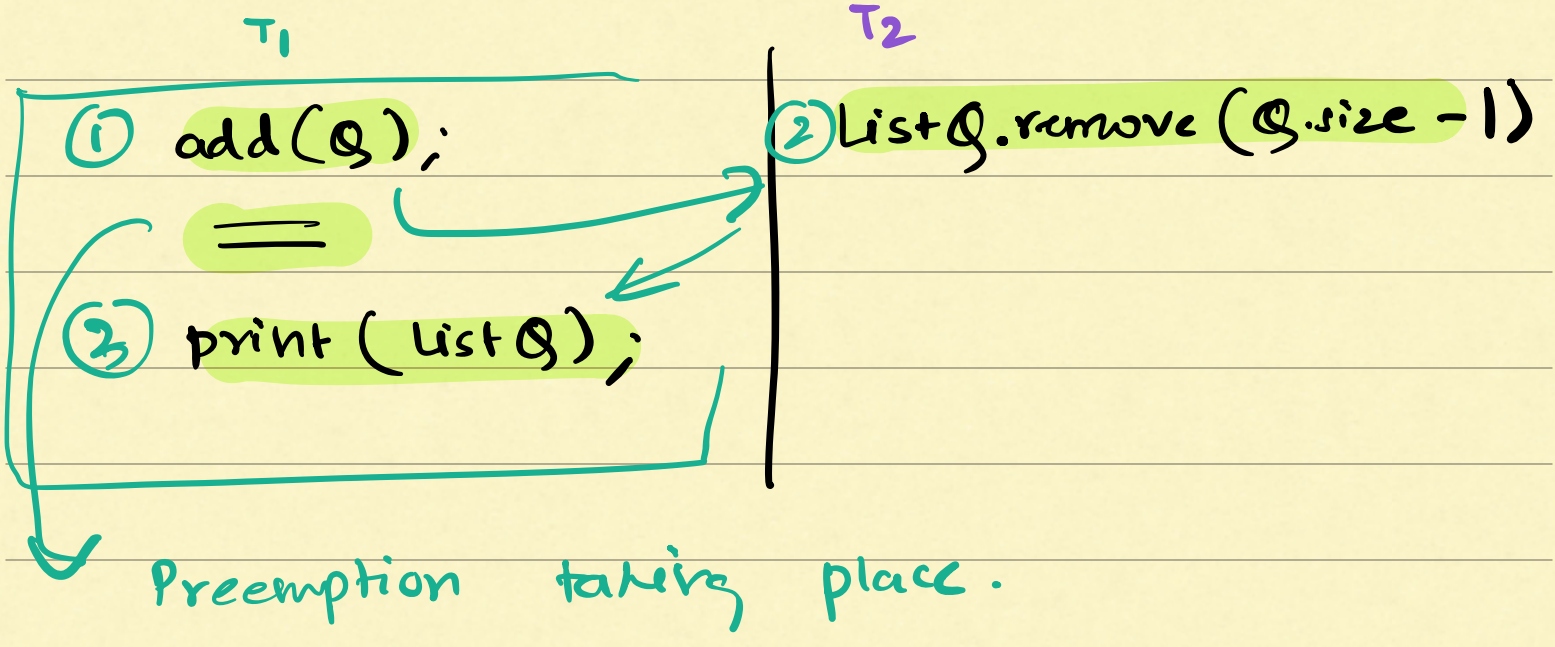
} Critical Section

⑤ sout("Hello 3");

2. Race condition → when two threads enter the same critical section at the same time.

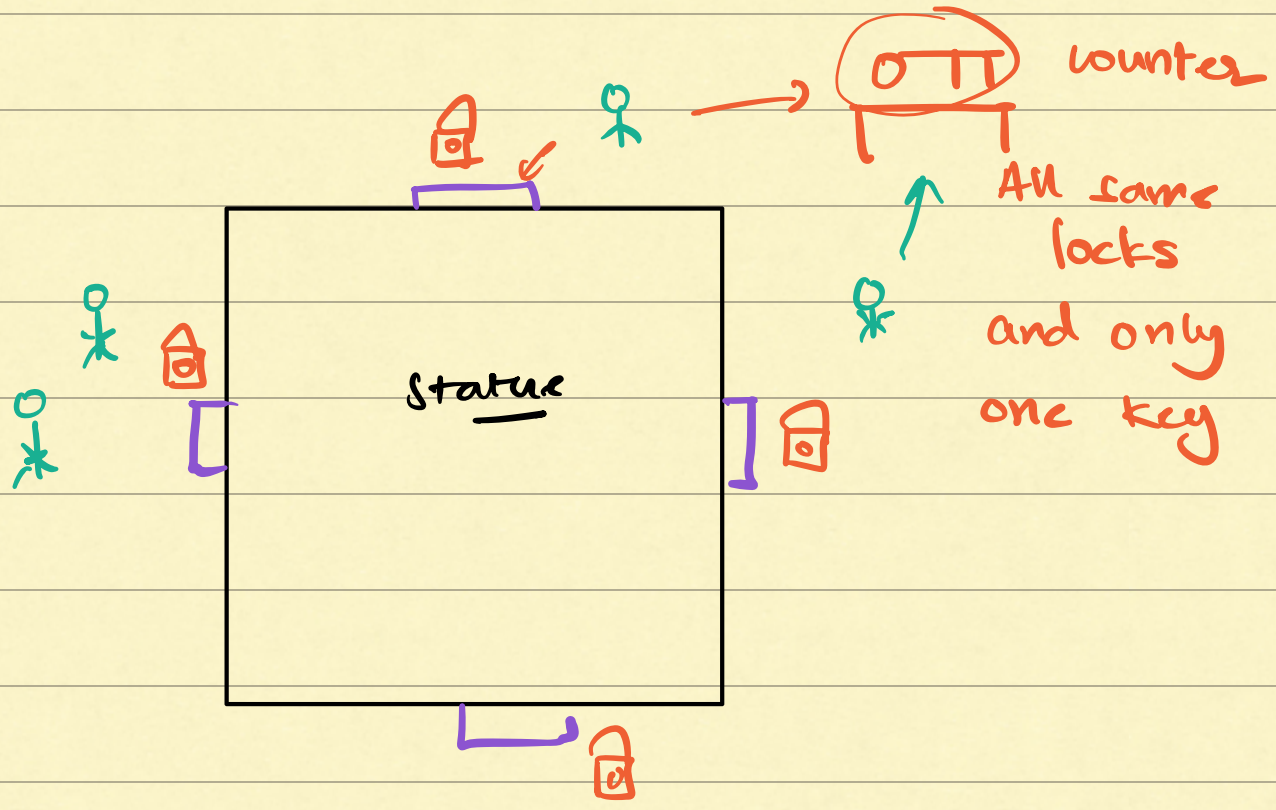


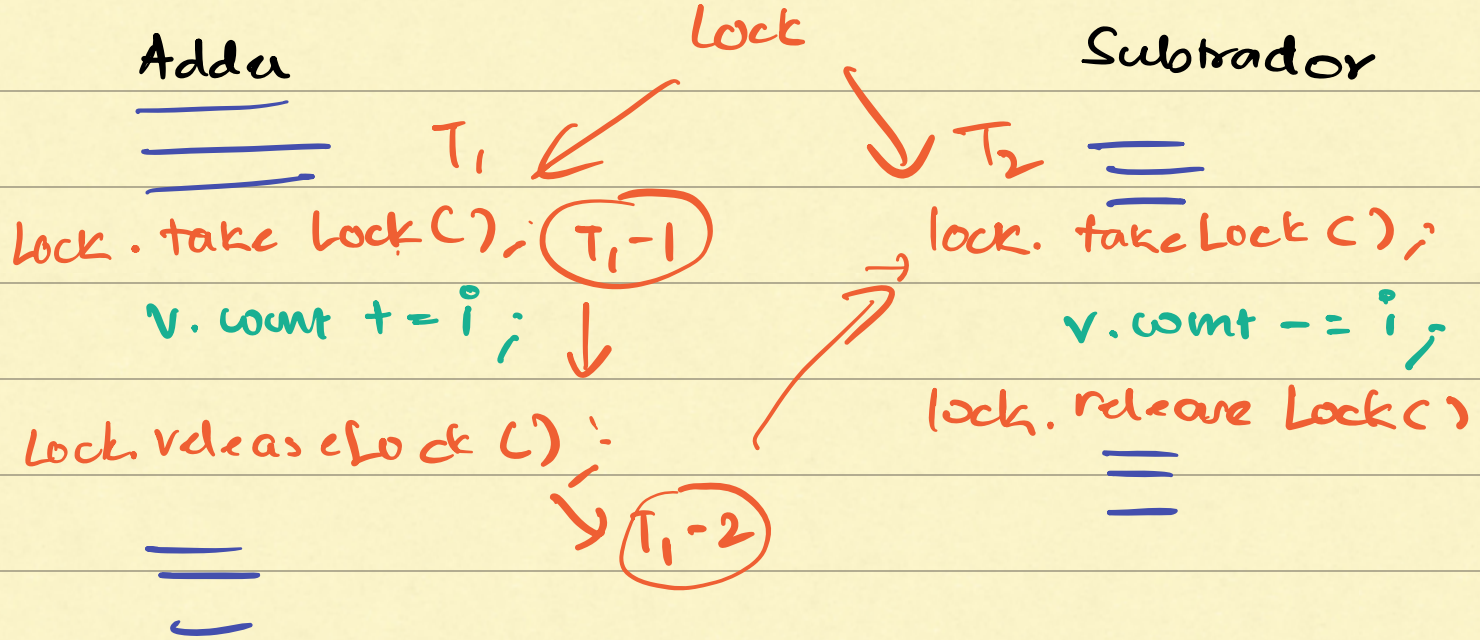
3. Preemptiveness: Ending a task before completion



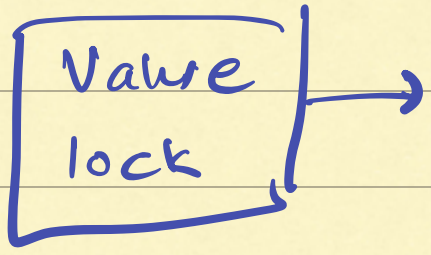
CONTEXT SWITCHING

Mutex → Mutual Exclusion





Synchronised keyword



→ Synchronised block.

Synchronised methods :

```
Adder {

synchronised add (int i) {        lock.lock()
    ———
    ———
    ———
} lock.unlock()

void subtract( ) {
    ———
    ———
}

synchronisedvoid multiply( ) {
    ———
    ———
    ———
}
```

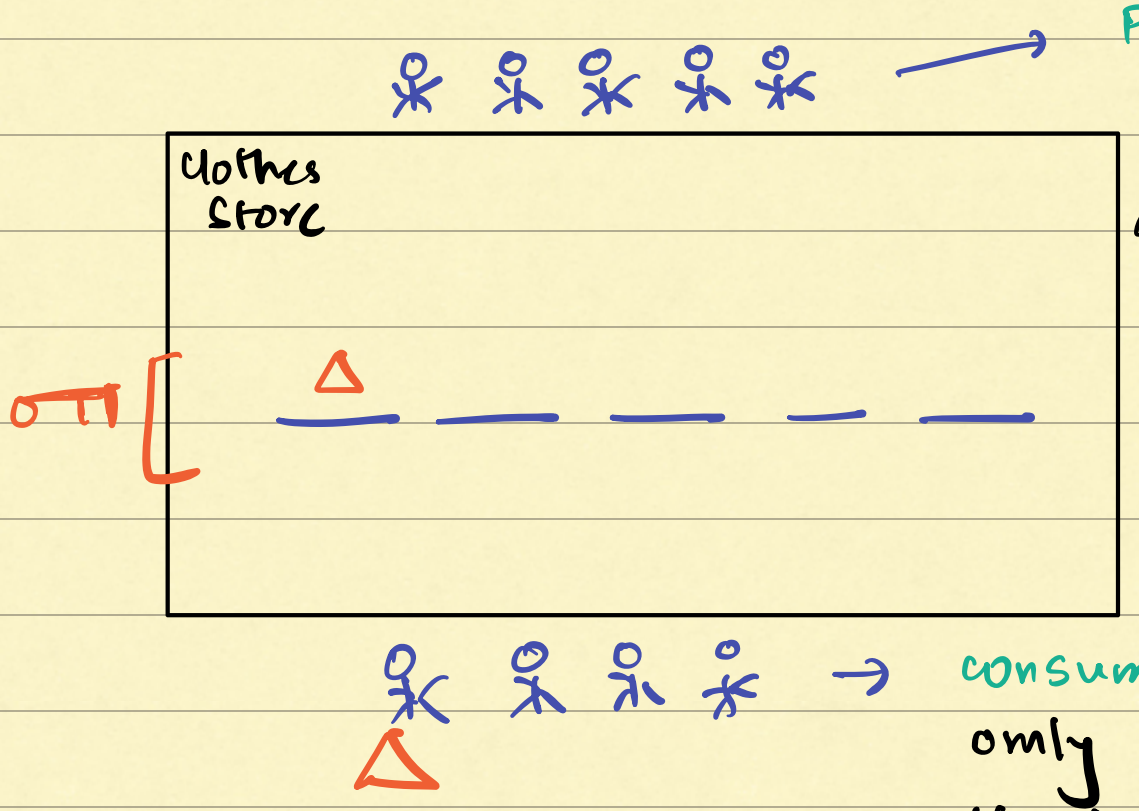
```
Adder a = new
Adder b = new

T1                T2
a.add             a.add
✓                 X wait

T1                T2
a.add             a.multiply
✓                 X wait

T1                T2
a.add             a.subtract
✓                 ✓

T1                T2
```

a.add b.add

✓ ◡

Producer / consumer

↳ is restricting 1 thread to work on a CS always beneficial



Producers can produce item only when a shelf is empty

consumer item only when atleast one shelf has item

Producer

T_1 T_2 T_3 store size=3

```
if ( store.size() < store.maxsize)
{
    store.add(new Object());
}
```

3 < 5 ✓

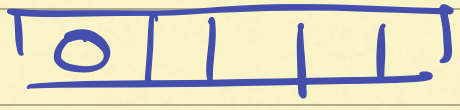
store size → 6

Consumer

T_1 T_2 T_3 store size=4

```
if ( stor.size() > 0){
    Store.remove();
```

T_2, T_3 will throw outofindex error.



T_1 T_2 T_3

1 1 1

1 > 0 ✓

synchronisation issues

↳ Semaphores

T_1, T_2, T_3 → T_1 removes successfully.