Today's Agenda :-

1) Synchronisation Problem. (Recap)

2) Ideal Solutions ✓

- Mutex / Locks
- Synchronized Keyword
- Sync Methods

3) Producer Consumer Problem.

17th April — 19th

Wed's ⟶ Fri

Consider Wed as a break.

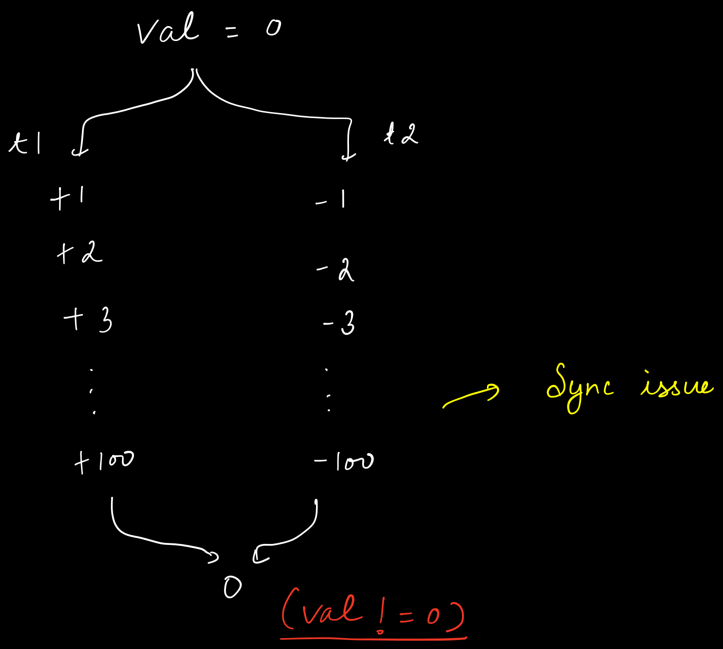
15th - 19th

16, 17, 18, 19
✓ ✓ ✓ ✓

↓

{ { Backlogs till Concurrency -3.

{ % change in psp.

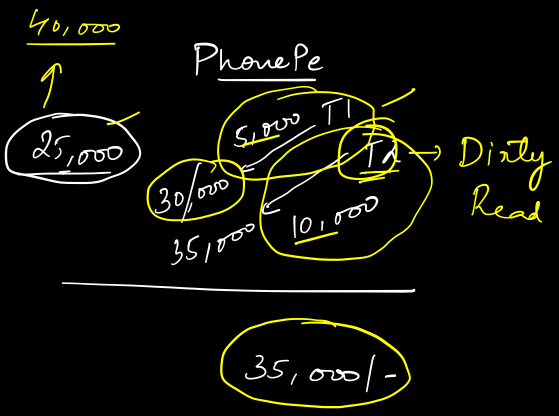


Why ?

1) **Critical Section** :- Part of the code where we are working on shared data.

```
print("Hello")
val += i
print("Bye")
val -= i
print("Done")
```

→ critical section



Adder

```
for (            ) {
    count += i
}
```

Subtractor

```
for (            ) {
    count -= i
}
```

Shared data

2) Race Condition :- 2 threads kind of race to complete a task.

when more than 1 thread tries to enter the critical section at the same time.

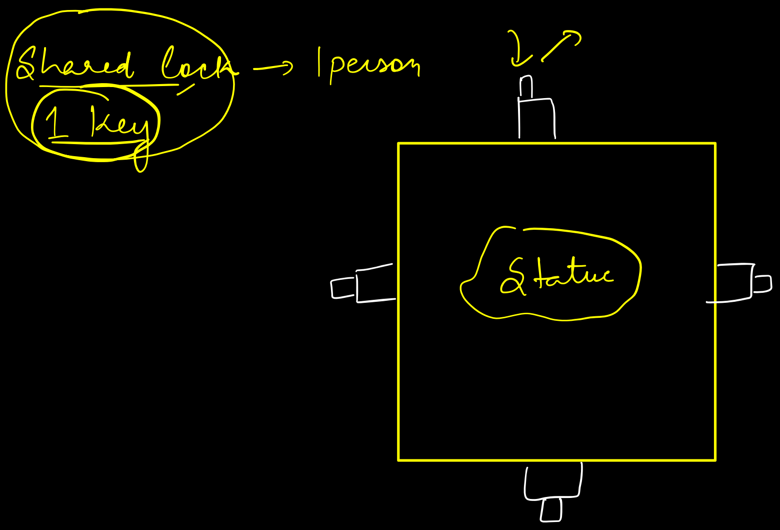
3) Preemptiveness :- We move from one task to another in partial state.

$\downarrow$

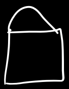
Context Switch

1) Mutual exclusion (Mutex) | Lock

↳ Allows only 1 thread to enter the critical section at a time.



I want only 1 person to enter at a given point of time.

 for critical section of the code

lock ( )
} critical
unlock ( )

2) Synchronized.

↳ implict lock for objects that exists in Java.

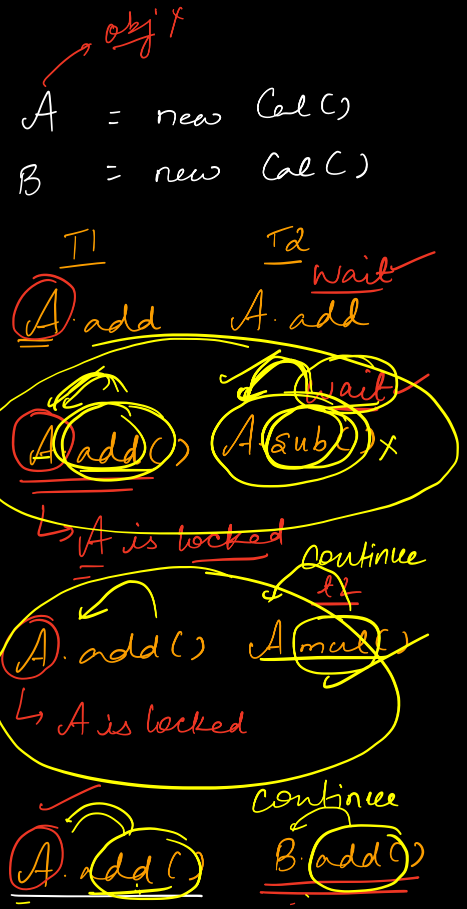
```
synchronised (cnt) {
    cnt.val += i
}
```

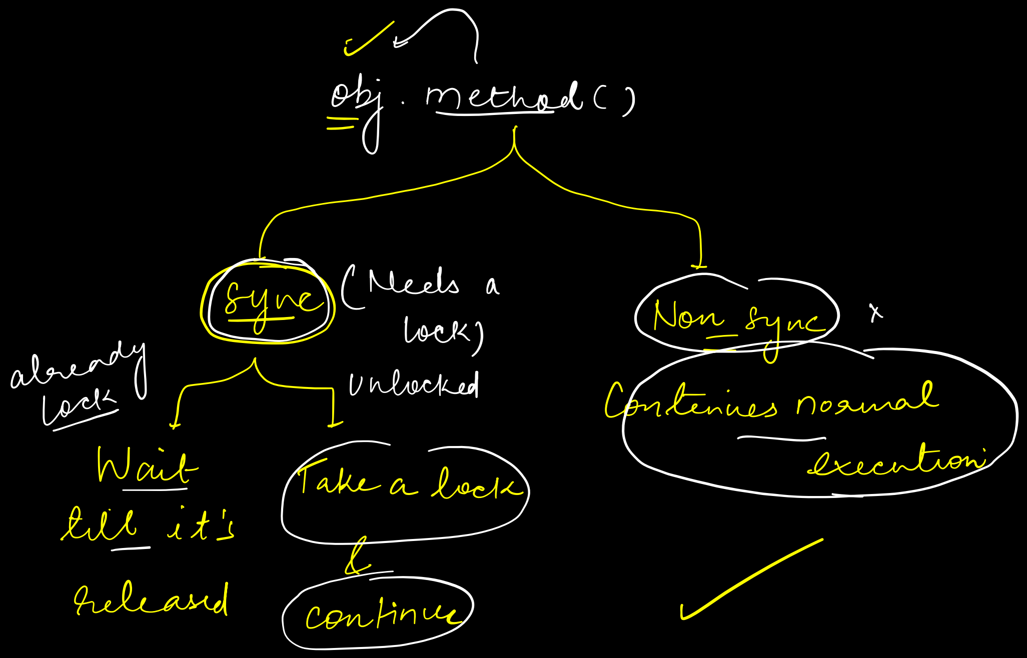
3) Synchronized methods.

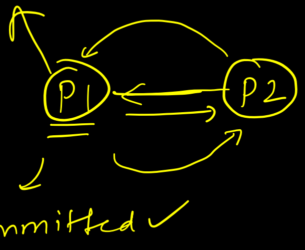
lock is going to be taken on the object which calls the sync method.

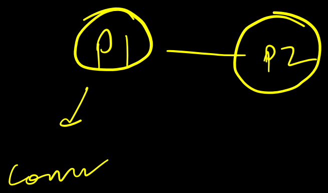
```
class   Calculator {

    synchronized void add() {
    |        ✓
    }
    void mul() {                        2
    |   { Not a critical section } x
    }
    synchronised void sub() {        3
    |        ✓
    }
}
```









3

Let's have a quick break :

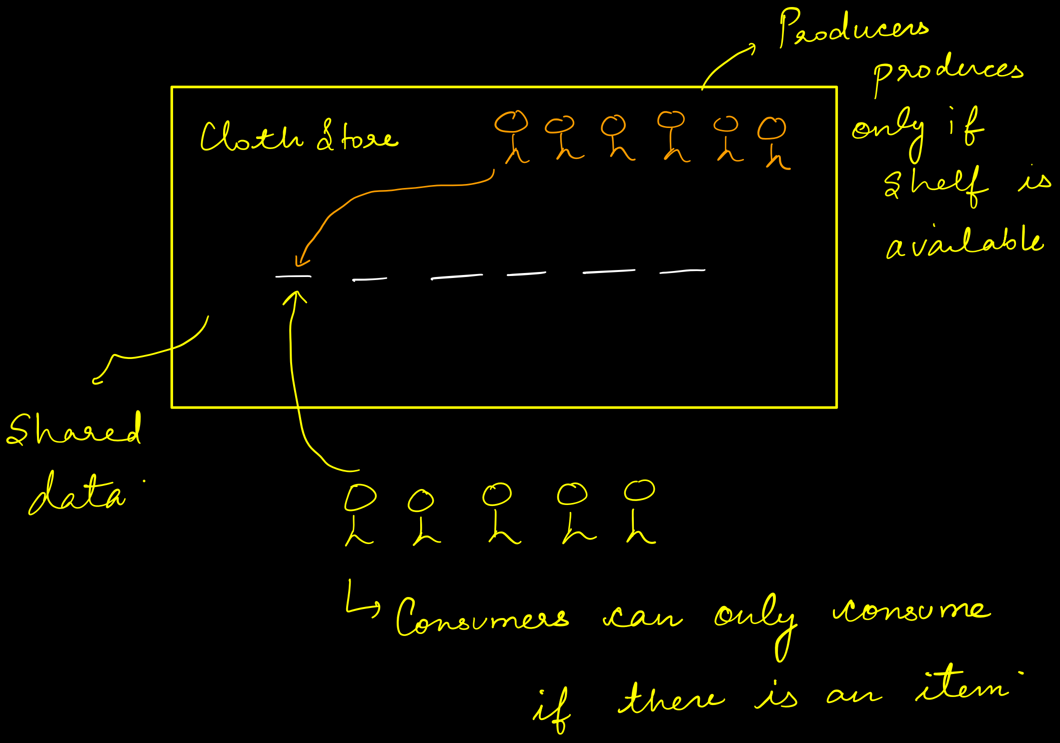
1) Mutex

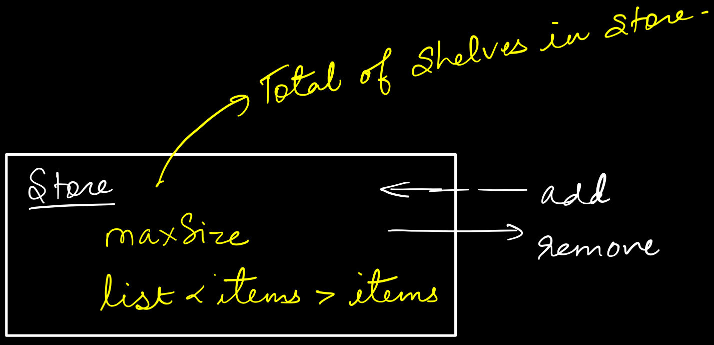
2) Synchronized

3) Synchronized methods

4) Semaphores
↳ topic for next class

Producer | Consumer Problem


Producers produces only if shelf is available
Cloth Store
Shared data.
Consumers can only consume if there is an item.



Producers

```
while (true) {
  if (Store.items.size() < maxSize) {
              3        <     8    ✓
     Store.add(items)   ✓
  }
}
        8 ✓
```

→ 9 items ??

Critical section

Consumers

```
while (true) {
  if (Store.items.size() > 0)
                  3
     Store.remove(items)
  }
}
```

→ only 3 will actually be successful
last one will fail

Semaphores

Scaler Thread ^ Thread

Hashmap