Agenda :-

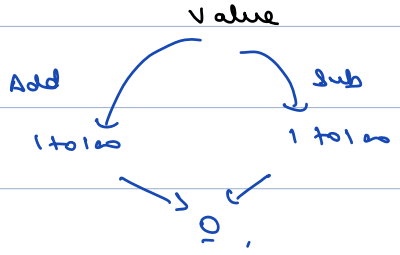
Adder Subtracter

Mutex

Synchronized keyword

Synchronized Method

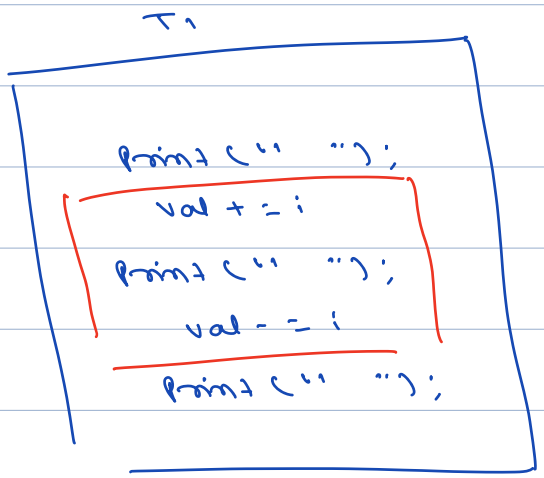
Producer consumer problem.



Synchronization issue. -> It's a problem in concurrent system that arise when multiple threads try to access our shared resource.

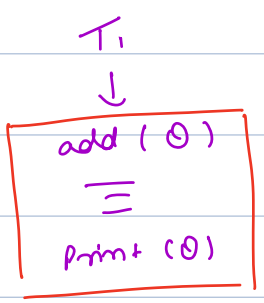
Why ?

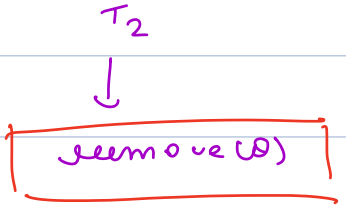
① Critical Section :- Part of code where we work on shared data.



T2 .

```
print ("  ");
val + = i
print ("  ");
```





(2) Race Condn → "Race of completing the task"

↓

Two or more threads accessing the shared resource at same time.

↳ may be.

(3) Pre-emptiveness → when you move from one task to another without completing it.

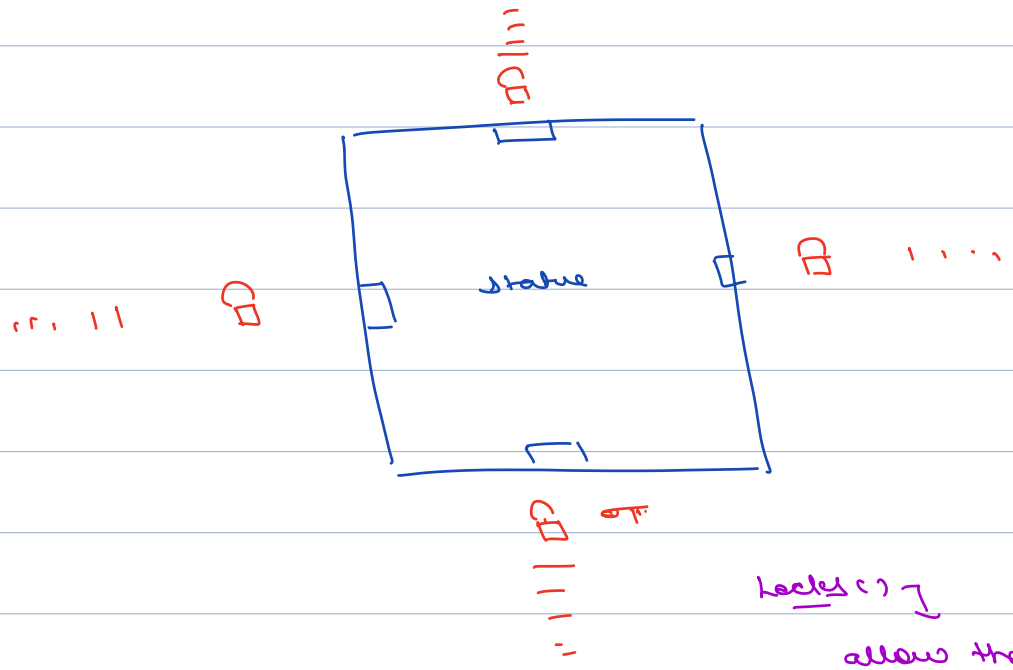
↓

context switching.

Mutex → Exclusion.

↓

Mutual



lock()

allow that only on thread has access to it.

lock

T_1

Adder

① lock();

② val++;

unlock();

T_2

Subtracter

lock();

val--;

unlock();

T_1 T_2 T_3 T_4

lock

T_1

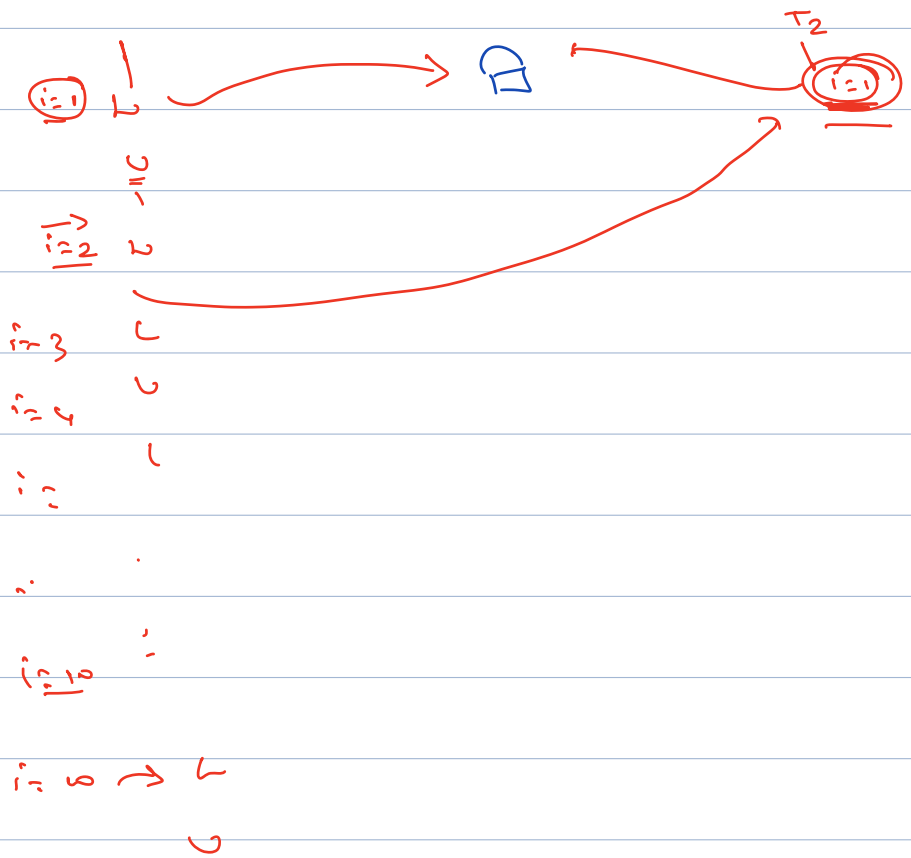
Adder

T_2

Subtracter

```
@Override  new *
public Void call() throws Exception {
    for(int i=1;i<=10000;i++){
        lock.lock();
        System.out.println("Adder" + i);
        this.v.value+=i;
        lock.unlock();
    }
    return null;
}
```

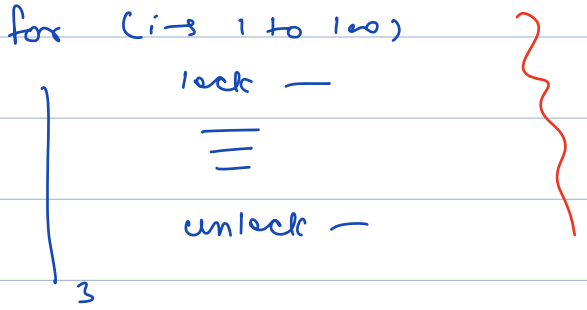
```
@Override  new *
public Void call() throws Exception {
    for(int i=1;i<=10000;i++){
        lock.lock();
        System.out.println("Subtracter" + i);
        this.v.value-=i;
        lock.unlock();
    }
    return null;
}
```

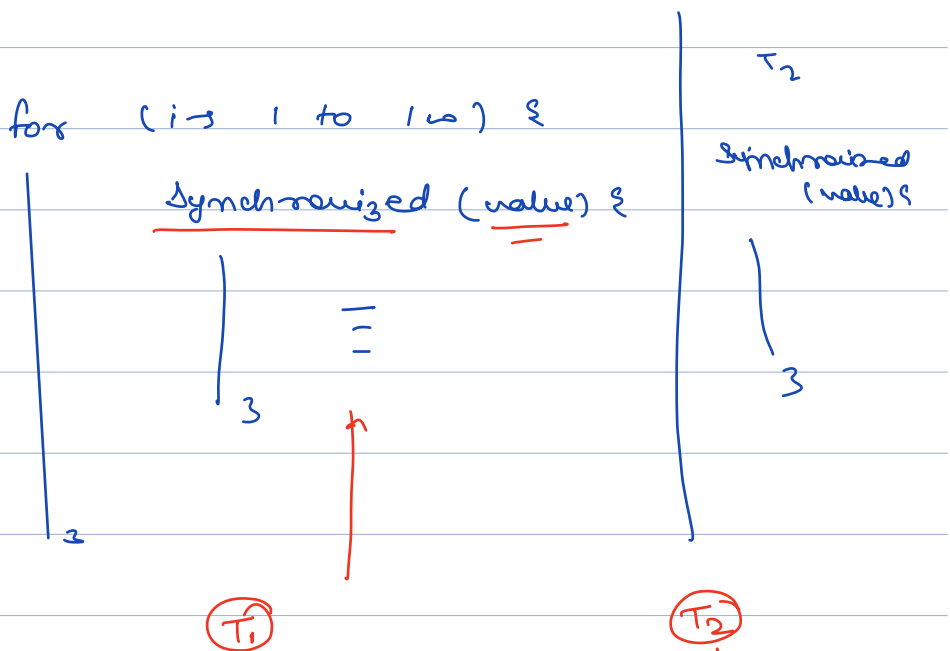


Java

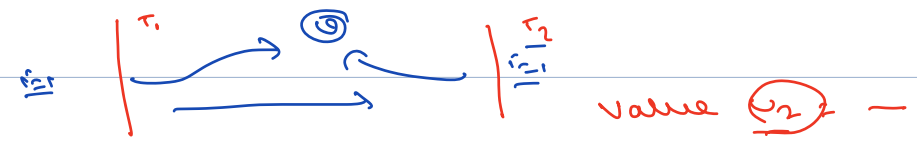
↳ implicit Lock

Synchronized keyword.



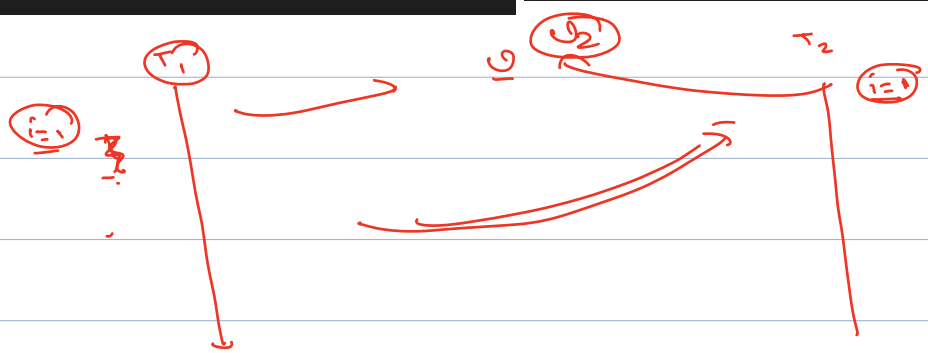


Object has intrinsic lock.



```java
@Override  new *
public Void call() throws Exception {
    for(int i=1;i<=10000;i++){
        synchronized (v) {
            System.out.println("Adder" + i);
            this.v.value += i;
        }
    }
    return null;
}
```

```java
@Override  new *
public Void call() throws Exception {
    for(int i=1;i<=10000;i++){
        synchronized (v) {
            System.out.println("Subtracter" + i);
            this.v.value -= i;
        }
    }
    return null;
}
```



Synchronization Method

a.add(b), b.multi();

class calculator {

synchronized void add() {

=

}

synchronized void multi() {

=

}

synchronized void subtract() {

=

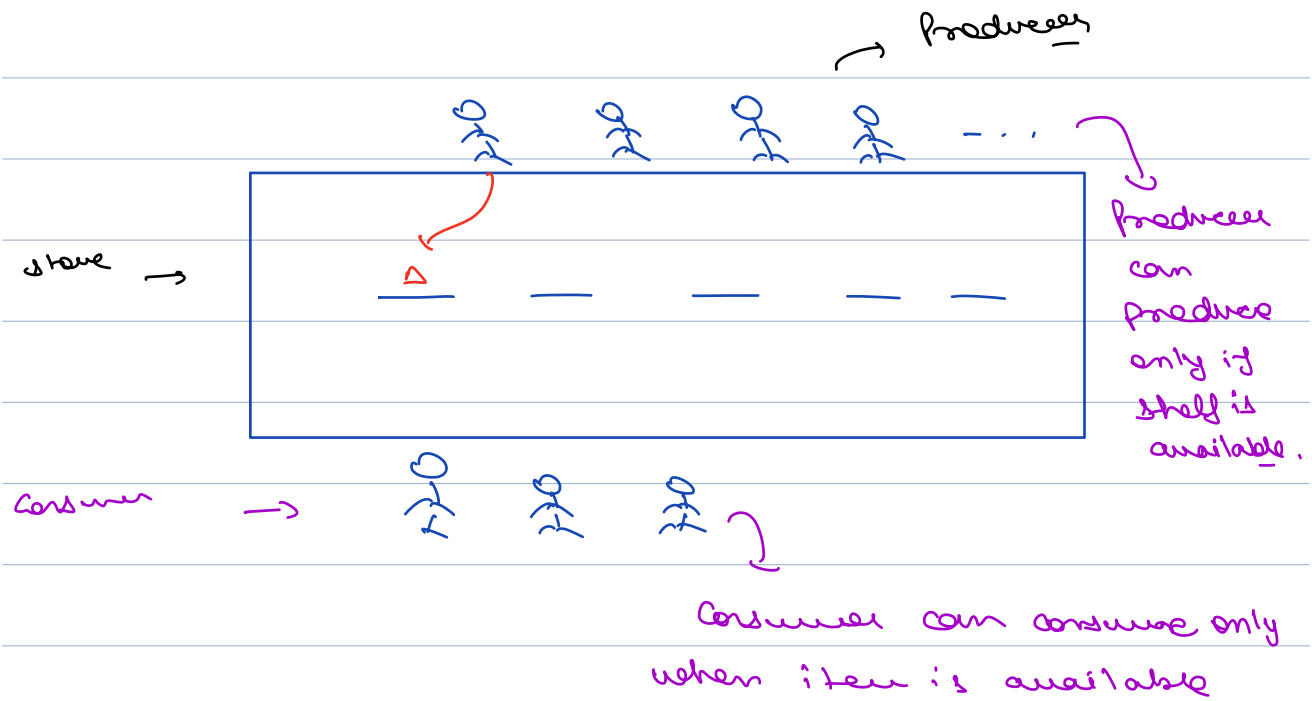
}

try to take a lock on the object on which it is called,

a = new calc();

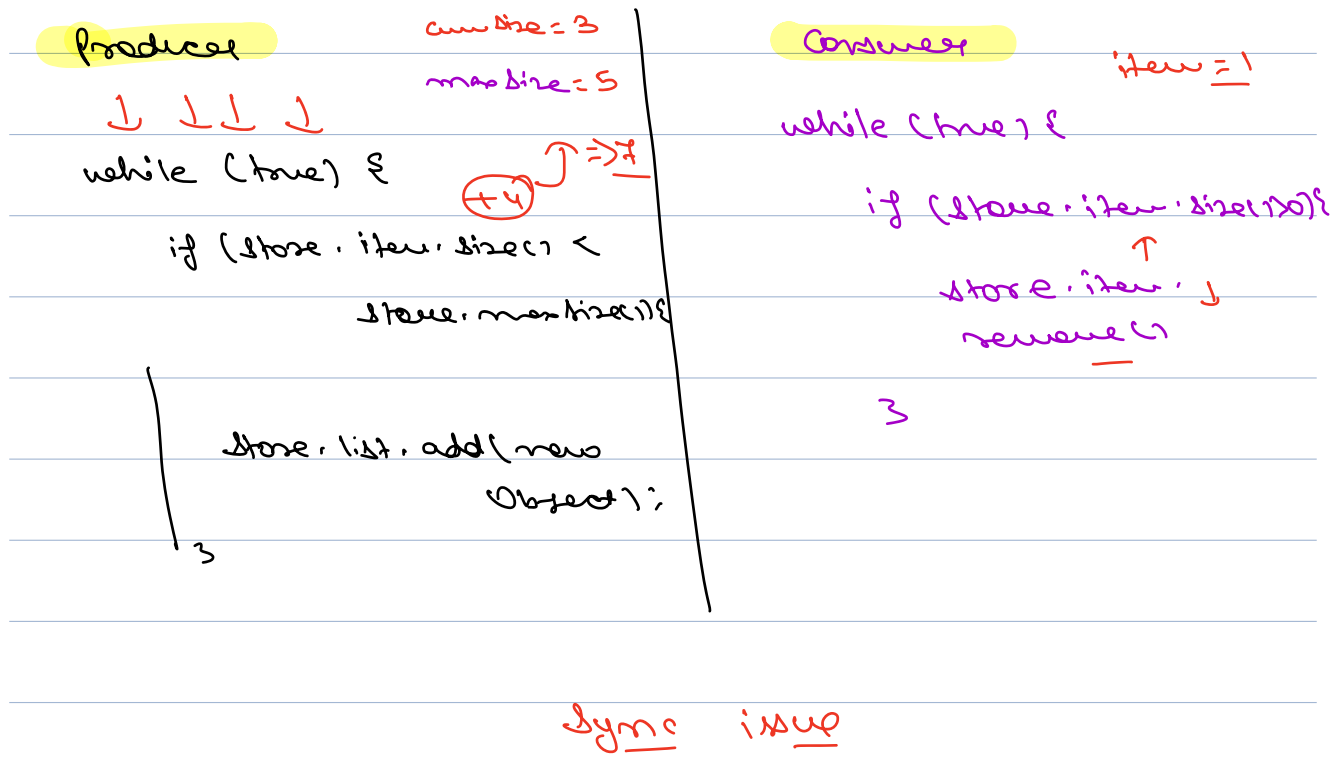
T_1	T_2
a.add(); ✓	a.add(); ✗ wait()
a.add(); ✓	a.multiply(); ✓
a.add(); ✓	a.subtract(); wait(),
a.add() ✓	b.add()

Producer Consumer Problem

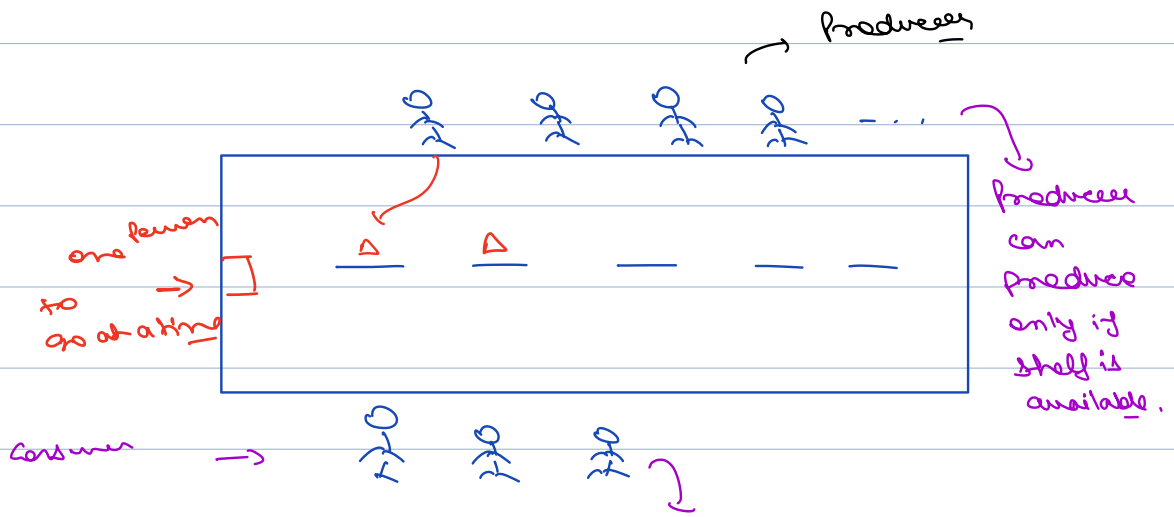


Store

- modize
- List<Object> items ;


Producer
curSize = 3
maxSize = 5
while (true) {
if (store.item.size() <
store.maxSize()){
store.list.add(new
Object());
3
+4
=>7
Consumer
item = 1
while (true) {
if (store.item.size()>0){
store.item.
remove();
3

Sync issue



is restricting # thread in CS, always beneficial ?