Agenda :

- ✓ Why sync problem happens ?
- ✓ Mutex — locks
- ✓ synchronized keyword
- ✓ synchronized method
- Producer / consumer.

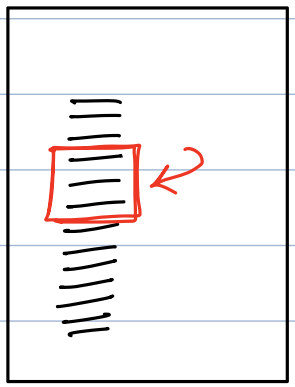
Adder Subtractor
↓
ans = 0

Note: No class on Friday, dashboard already updated

Why ?

① critical section → part of code where you are working on the shared data.

↓ important



T1

```
print("Hello")
this.val += i
print("Hey")
this.val -= i
print("Bye")
```

T2

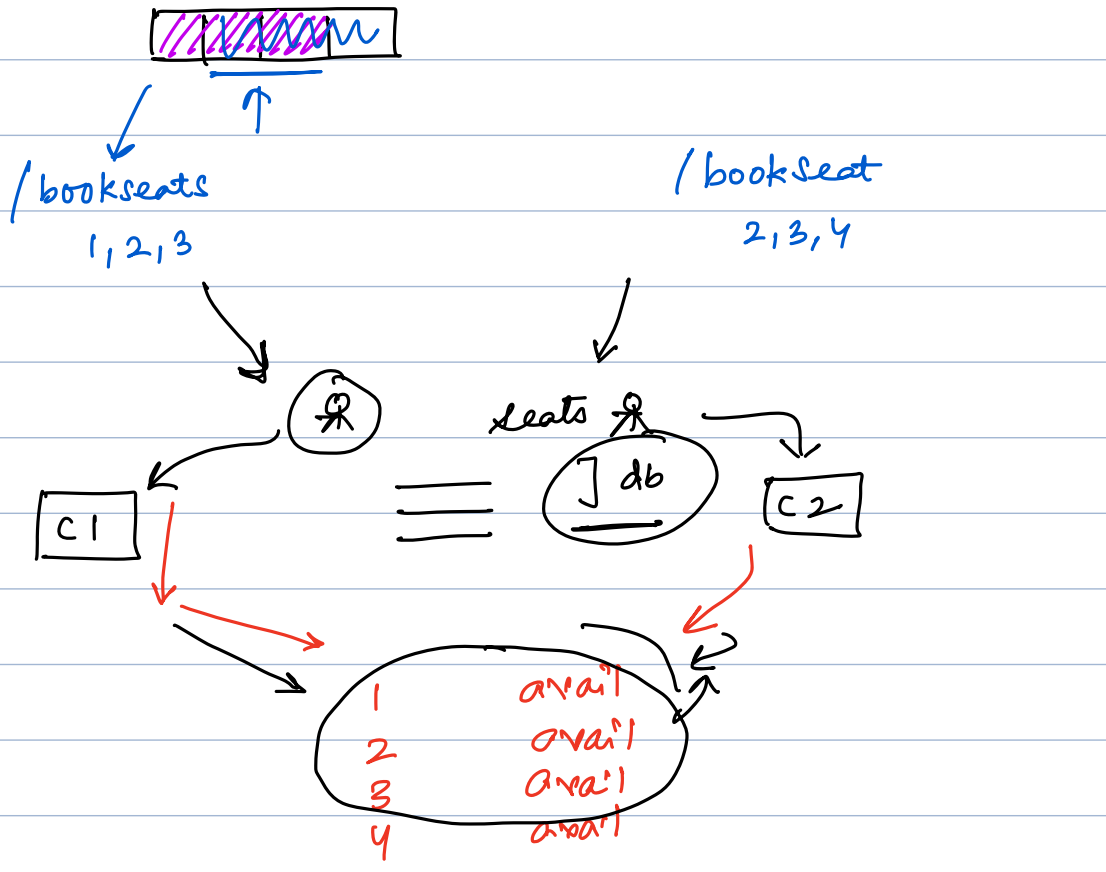
```
print("___")
this.val *= i
print("Bye")
```

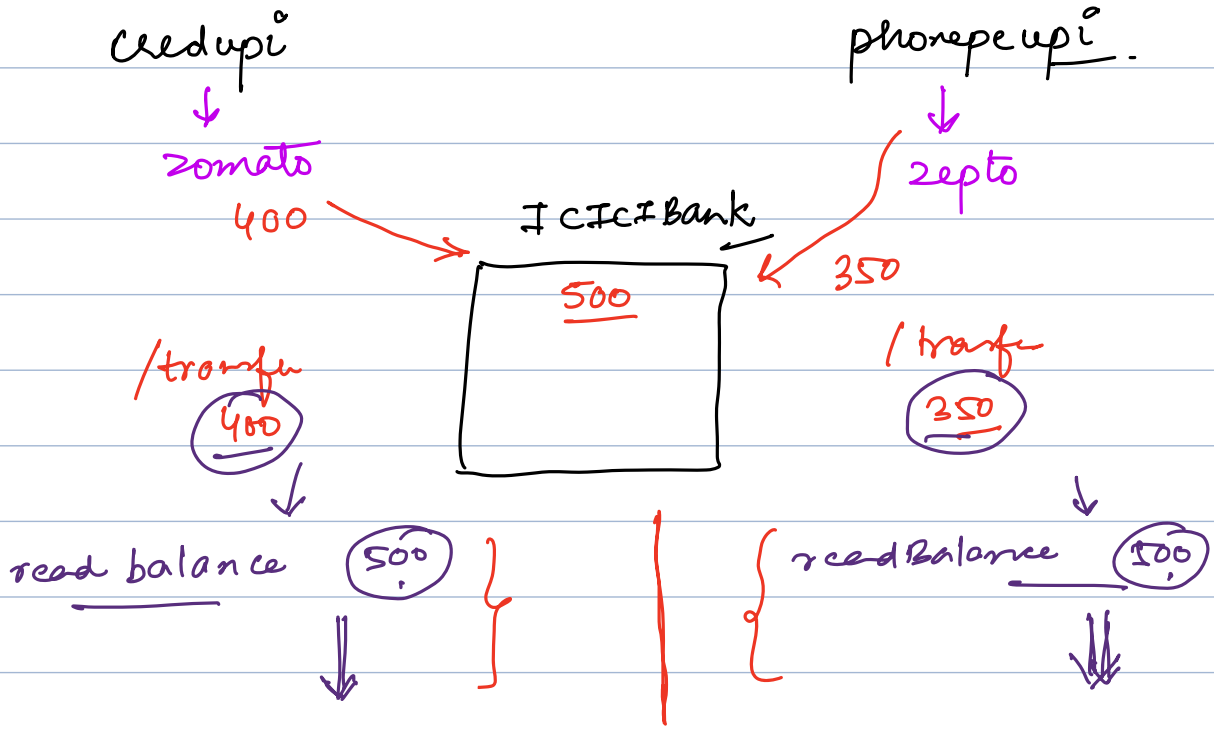
T1

```
add(ques)
submit
display(ques)
```

T2

```
delete(ques)
```


/bookseats
1,2,3
/bookseat
2,3,4
seats
db
C1
C2
1
2
3
4
avail
avail
avail
avail


credupi
zomato
400
phonepeupi
zepto
ICICIBank
500
350
/transfer
400
/transfer
350
read balance
500
readBalance
100

② Race condition :→ " Race of completion of task "

↓

Two or more threads enters into critical section

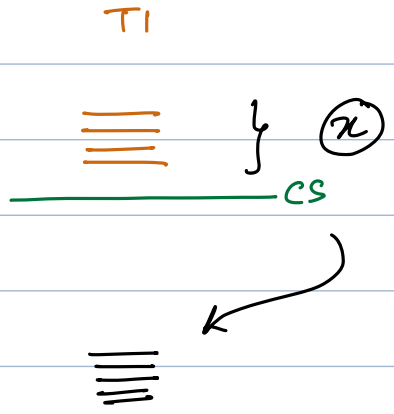
③ Pre emptiveness

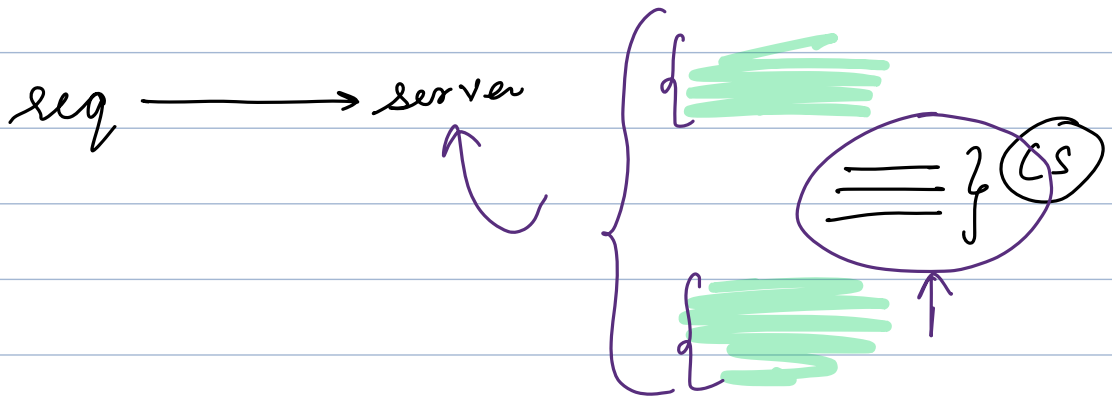
↓

context swit .

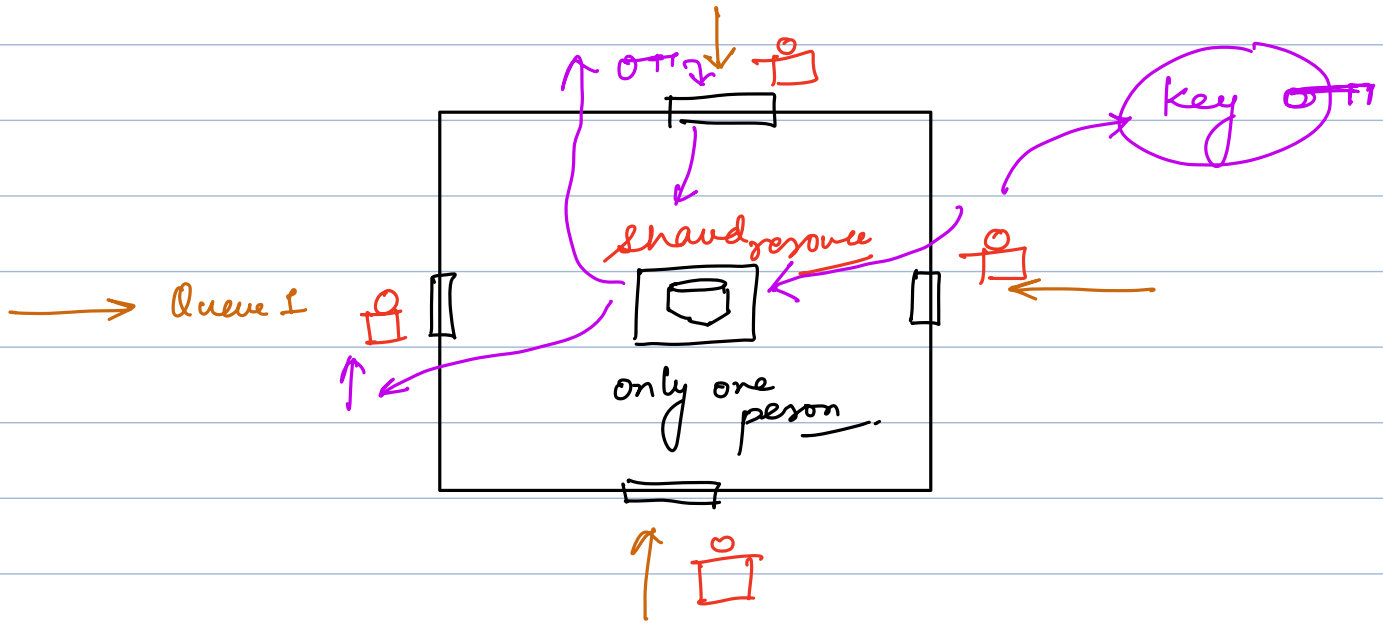
↓

we move from one task to another task before completing it .

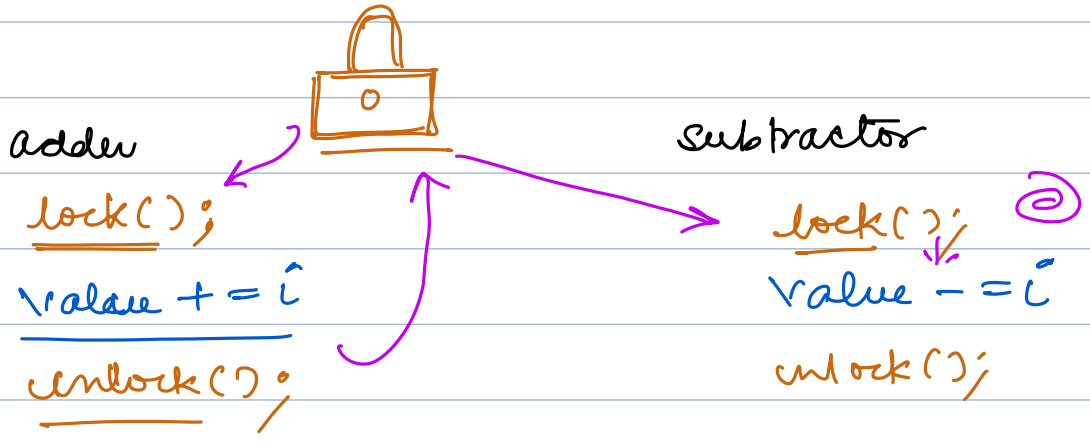


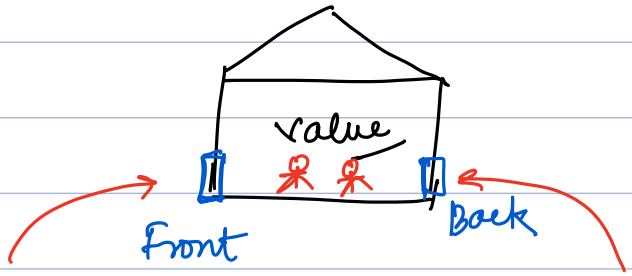


Mutex: Mutual exclusion



Locks → allows one thread to enter in the critical section

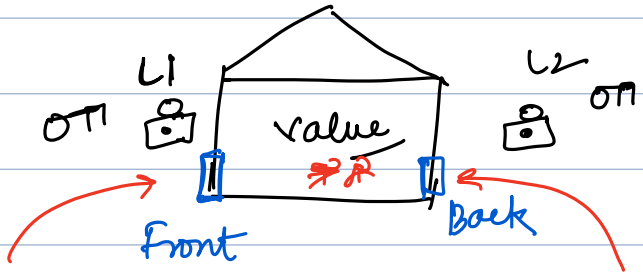


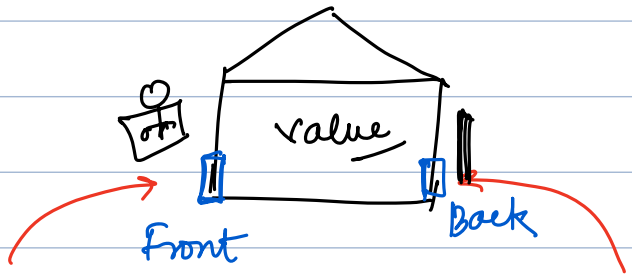


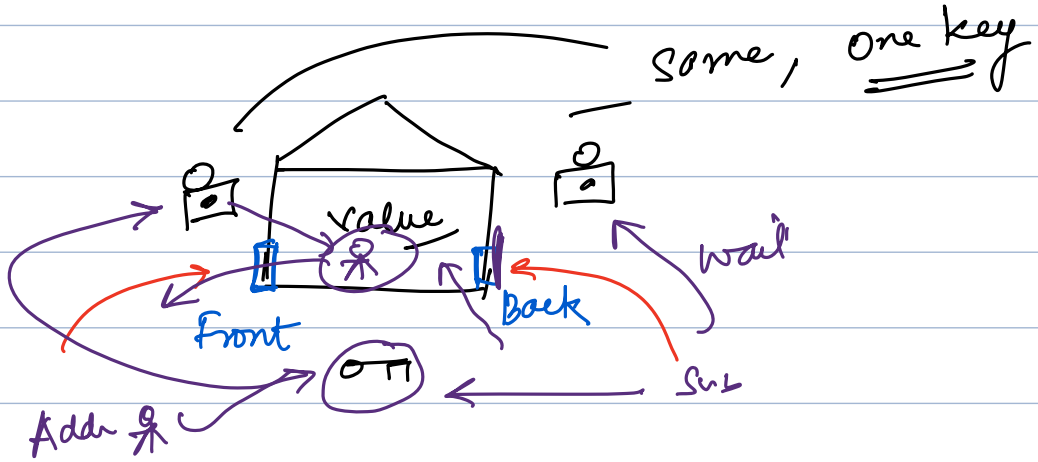
app locks

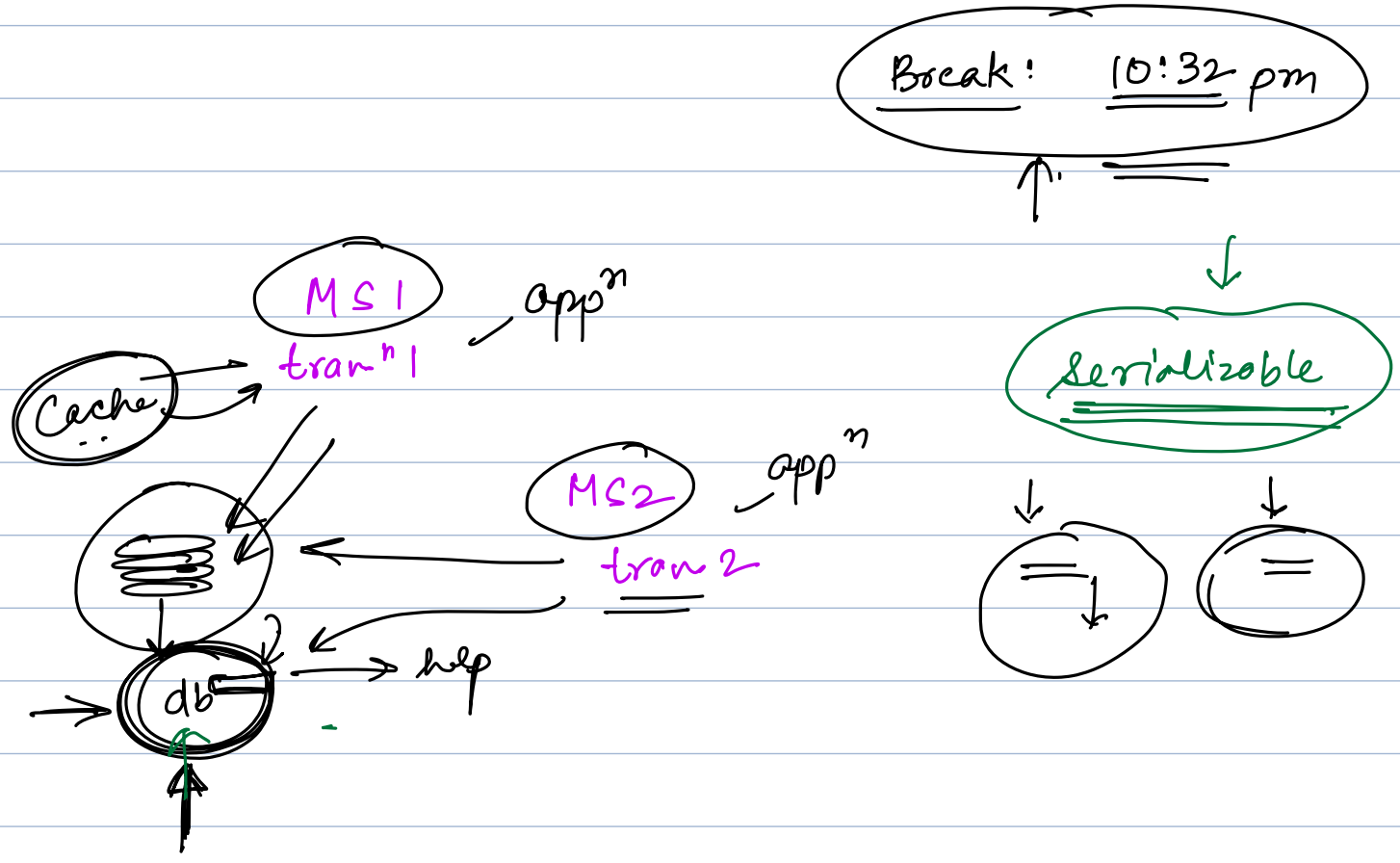
db locks

BMS



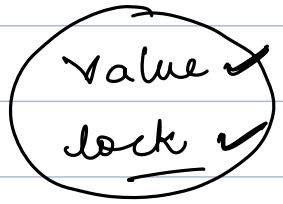






cache → key-value — Hashmap
concurrent Hashmap

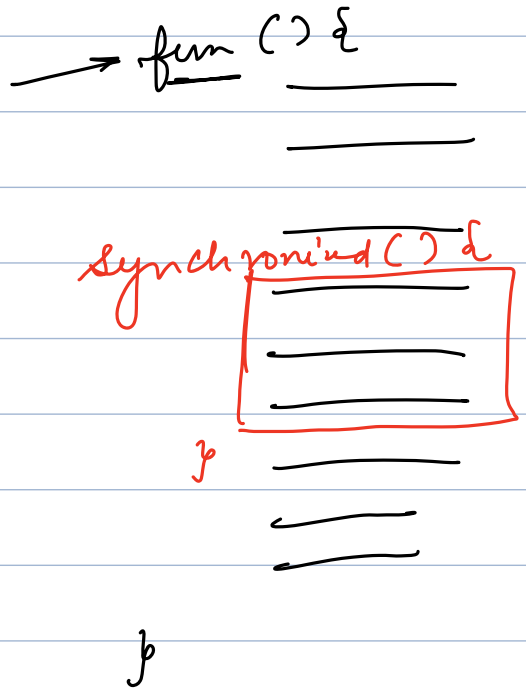
synchronized



→ object has intrinsic lock

```
for ( i = 0 ⟶ 100)
{
    lock.lock();
    value += i;
    lock.unlock();
}
```

```
for ( i = 0 ⟶ 100)
{
    synchronized (value) {
        value += i;
    }
}
```



Synchronized methods

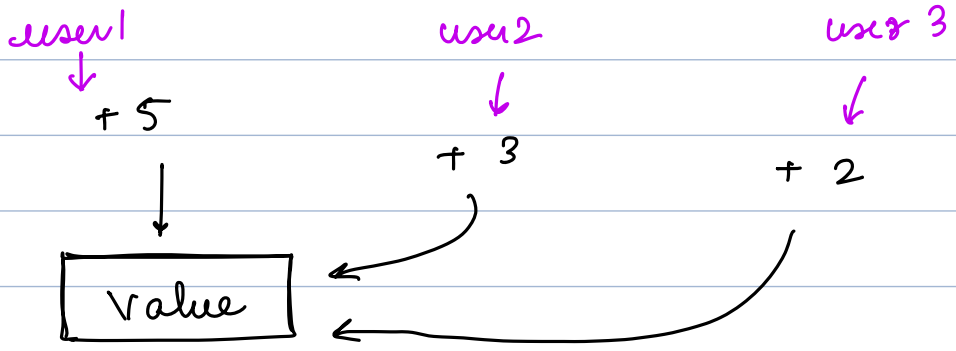
```
class abc {

Synchronized void doSomething ( ) {
    =
    }

}
```

```
Abc x = new Abc ( );
x.doSomething ( );
```

→ It takes a lock on x, if some one tries to take lock, wait.




```
synch- addInValue( int input){
        synchro (value){
            value.x += input;
        }
}
```

value.addInValue

class Calculator {

synchronized void add();

void subtract();

synchronized void multiply();

}

Cal a = new Calculator();
b = new Calculator();

	T1	T2
I	a.add(); ↑ lock	a.add(); ↑ wait
II	a.add(); ↑ lock on a	a.multiply() ↑ wait.
III	a.add(); ↑ lock	a.subtract() ↑ wait.
IV	a.add() ↑ lock on a	b.add() ↑ lock on b.

↓

diff locks unless a & b are reffg to same obj

a.add()
↑ lock on a

a.subtract()
↳ execute.