recap
- → imp of multi-threaded code
- → executor service
- → callable & Future
- → Synchronisation problem

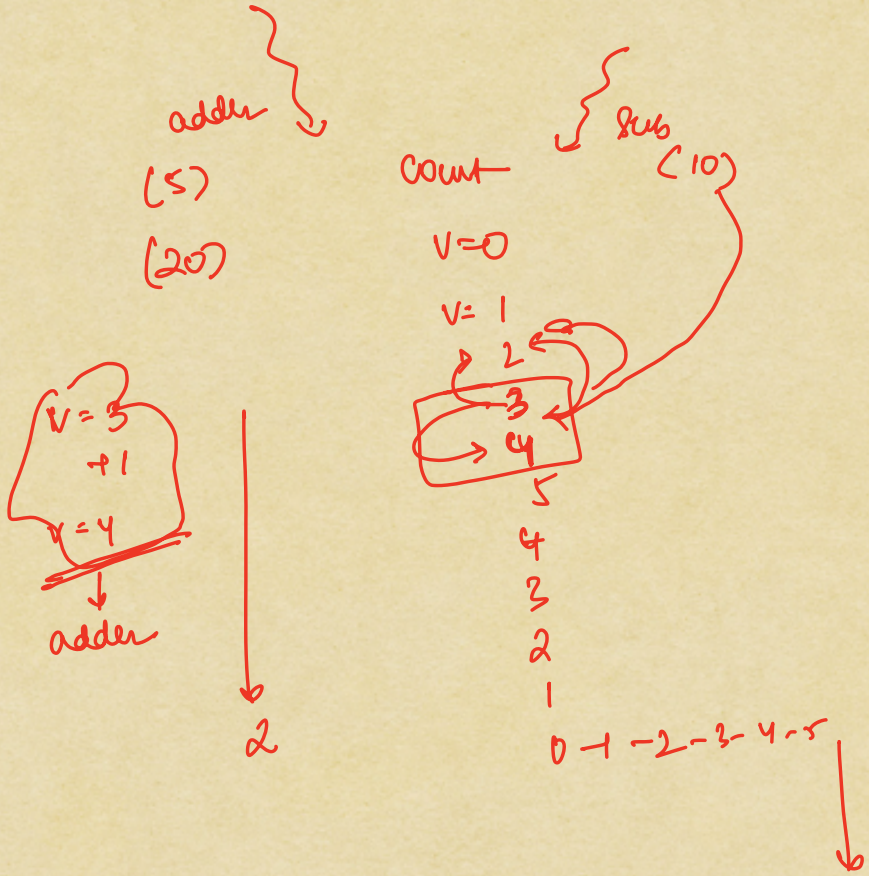
⇒ Synchronization
- what
- why
- How we can solve
- mutex
- Synchronised keyword
- Semaphore

```
public static void main(String[] args) throws Execution
    /*...*/
    Count c = new Count( value: 0);

    Adder adder = new Adder(c);
    Subtractor subtractor = new Subtractor(c);

    Thread adderThread = new Thread(adder);
    Thread subtractorThread = new Thread(subtractor);

    adderThread.start();
    subtractorThread.start();

    adderThread.join();
    subtractorThread.join();

    System.out.println(c.getValue());
```

main waits

makes your current thread wait until the thread at which join is called is/ut completed & killed

prints the value inside c.(count)



⇒ Properties of a good soln of synchronisation prob.

Critical section:- part of the code that updates the shared data

critical section

```
public void run() {
    → Sout("Hello")
        for(i → 1 to 1000) {
            → Sout("inside count")
                c.set(c.get+1);
            → Sout("done")
        }
    } → Sout("bye")
```

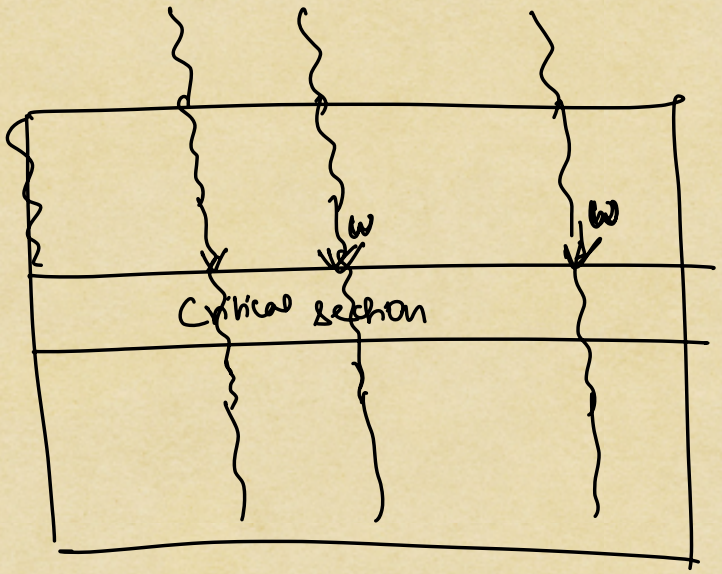
Count

↓

[setValue()]

1) **Mutually exclusive:-** Only 1 thread can execute the critical section at a time.

2) **Progress :-**



Overall system should keep progressing only critical section area threads should wait.

3) **Bounded Waiting:-**

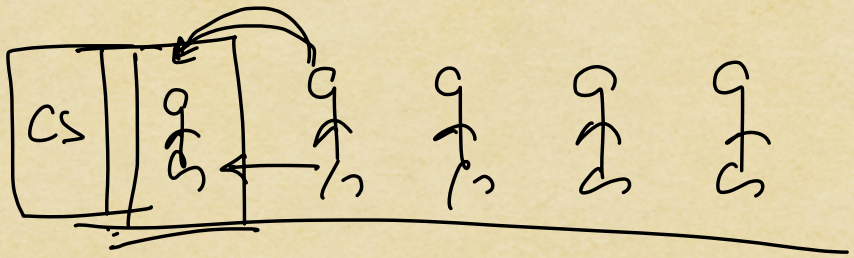
→ 1 thread enters CS and 2 threads are waiting

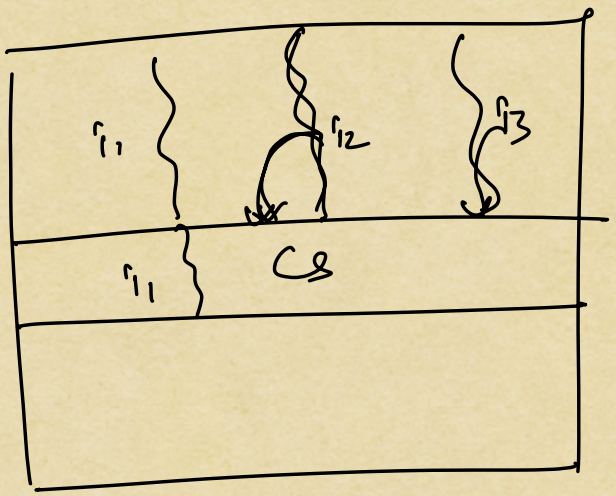
→ for some reason, thread in CS gets stuck

→ so, we shouldn't have unlimited waiting there

should be bounded waiting

4) NO BUSY WAITING





when a thread is waiting to enter the CS, it should not continously keep checking if CS is available to enter.

↓

ideal soln is that the thread that exits the CS should notify other waiting threads. or something else

i) MUTEX ii) Synchronised keyword

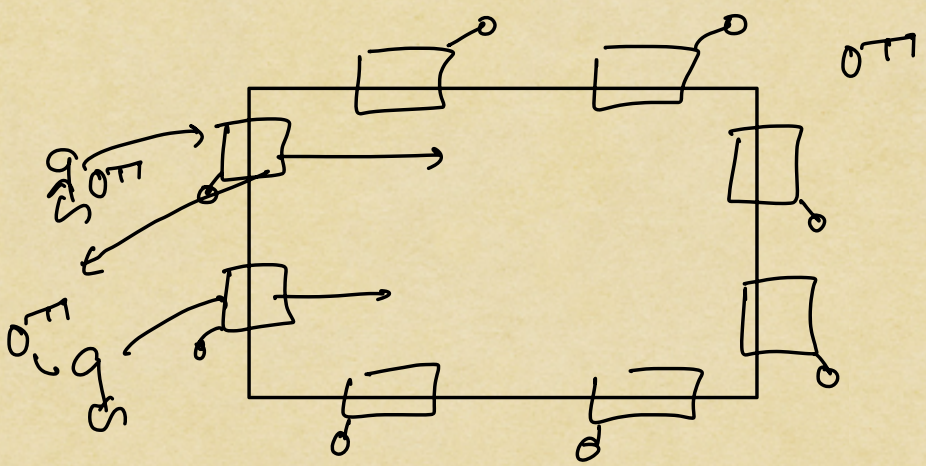
=> Mutex

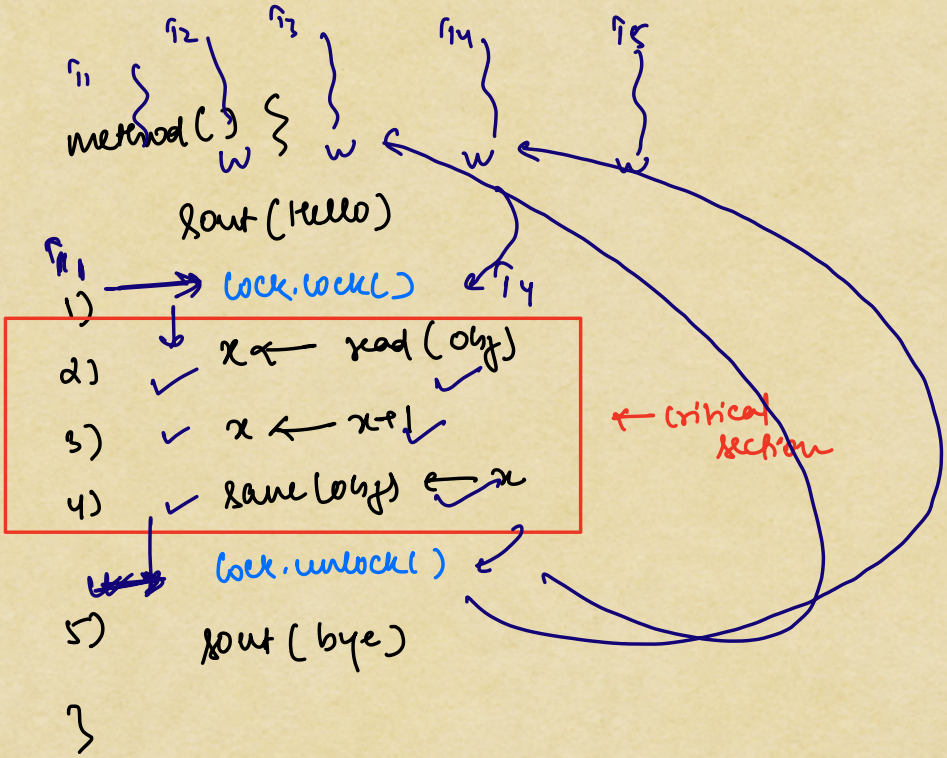
=> Mutual Exclusion lock | Mutex lock

```
method() {
    sout(Hello)
1)  lock.lock()
2)  x ← read(obj)
3)  x ← x+1
4)  save(obj) ← x
    lock.unlock()
5)  sout(bye)
}
```

← critical section

=> multiple locks but single key



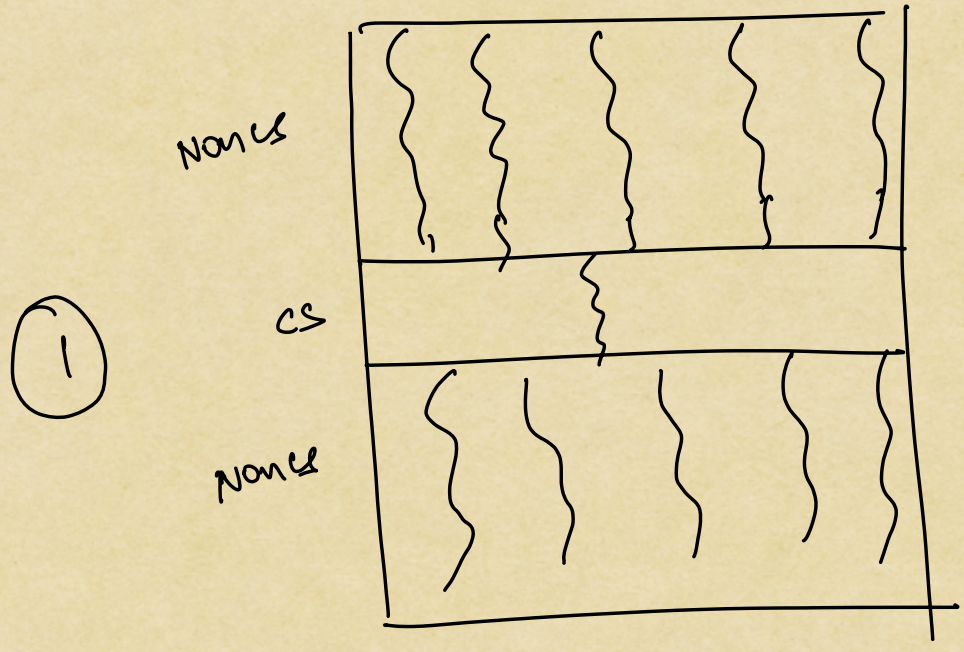


→ Mutual exclusion ✓

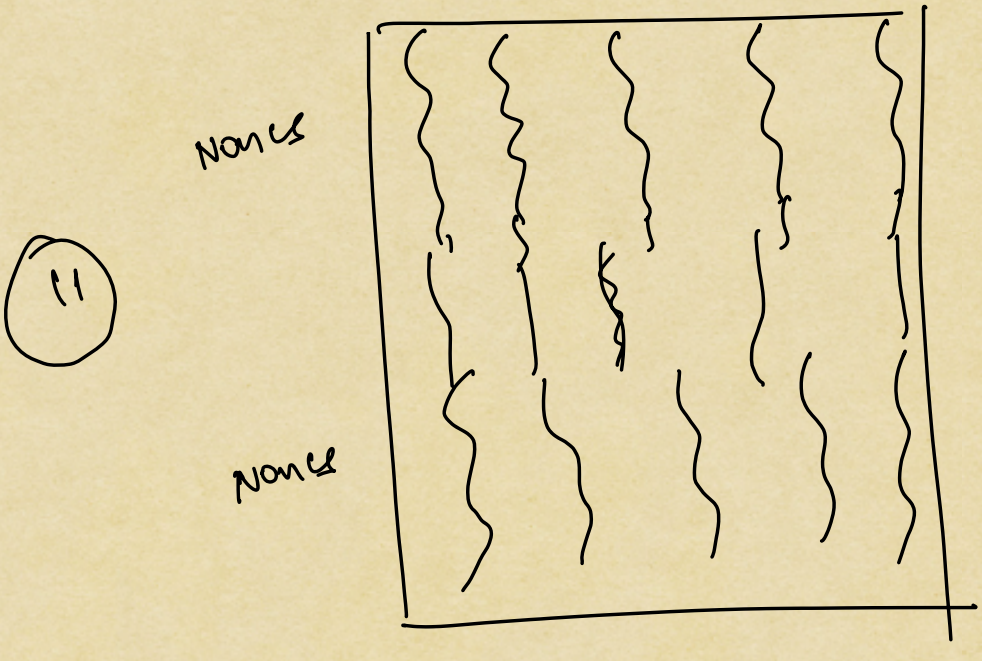
→ progress ✓

→ bounded waiting → default wait time ✓

→ no busy waiting ✓ ⇒ unlock() notifies the threads and allows any one of them to enter.

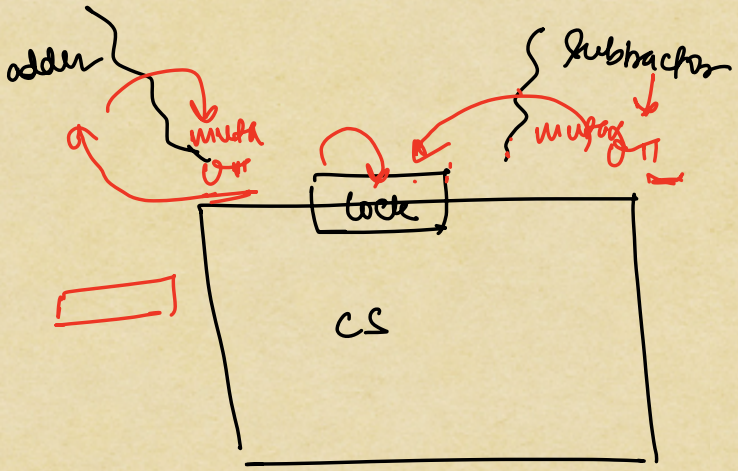


(I)



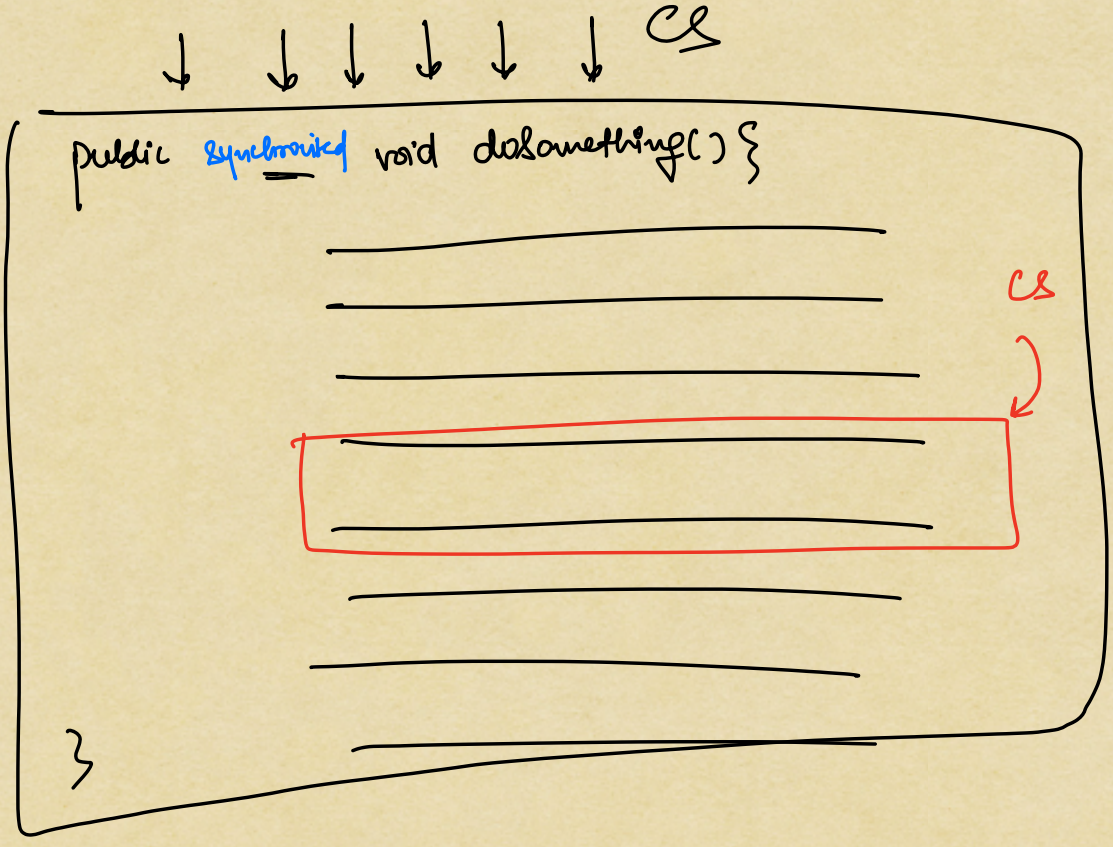
(II)

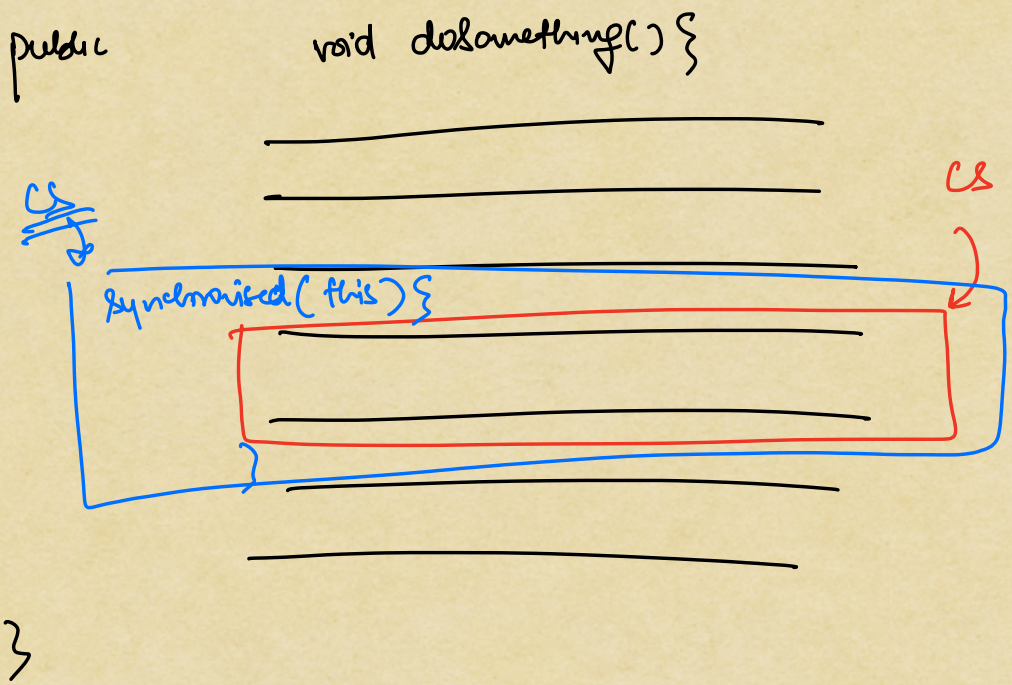
* Solving synchronisation issues reduces performance.



⇒ Synchronised

- Synchronised method
- Synchronised block.




public void doSomething() {
CS
CS
synchronized(this){
}