OS: Synchronization

Will start at 9:10PM

1. Adder Subtractor Problem
2. Intro to Synchronization
3. Condⁿ for Sync Problem
4. Solⁿ to Synchronization
   - ↳ Ideal Solⁿ Characteristics
   - ↳ Mutex
   - ↳ Semaphores
5. Producer Consumer Problem

Synchronized ()

Next Class

CN classes / LLD

MWF

Tanmay

TTS

Naveen

Support @ Scaler.com

Name of the Lang Module recordings not there

PROBLEM STATEMENT

variable

0

add 1 to 100 to this shared variable

subtract 1 to 100 to this shared variable

+1 +2 +3

+4

+100

−1 −2

−3 −4

−5 −

−100

Adder

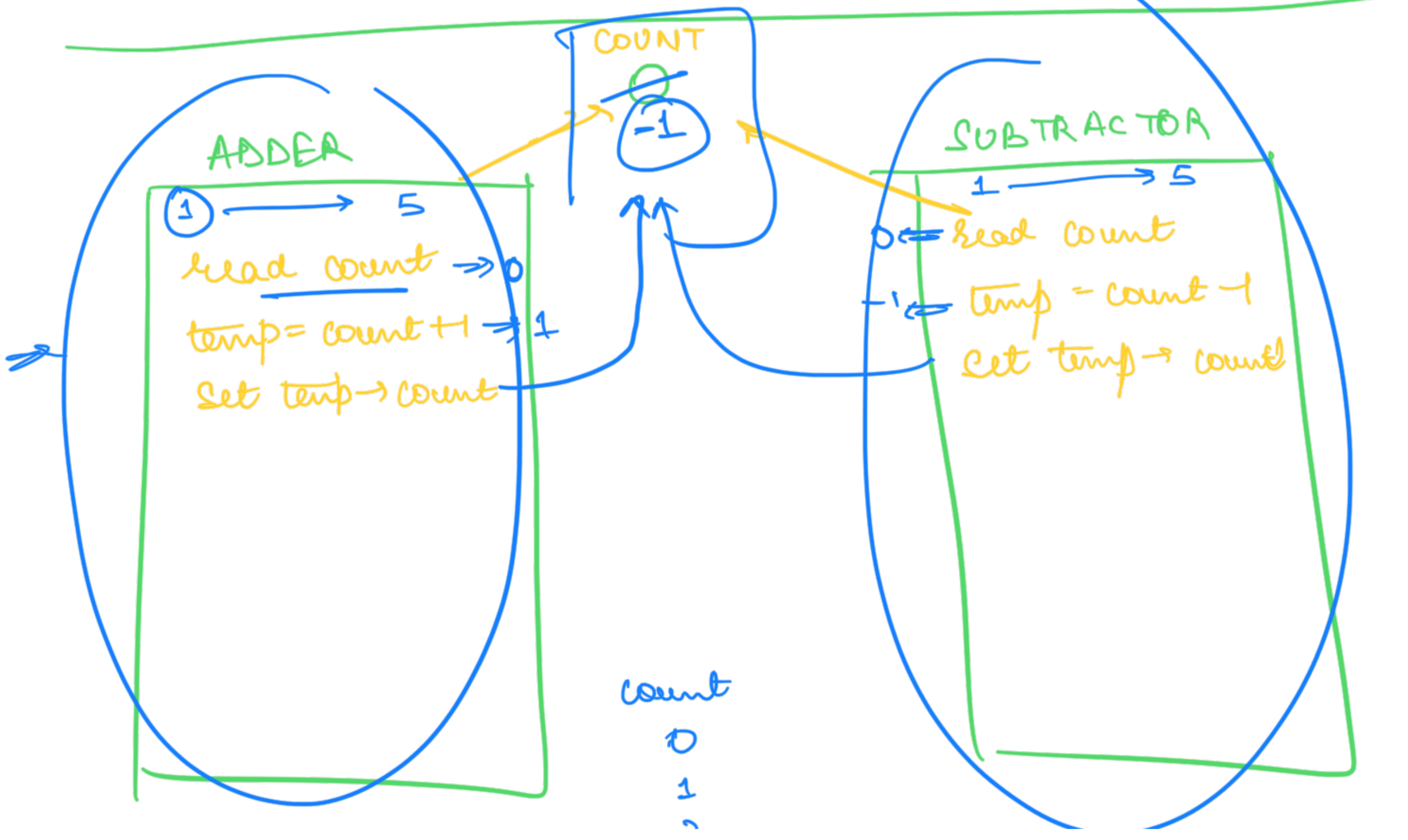
→ adds 1 to 100

Subtractor

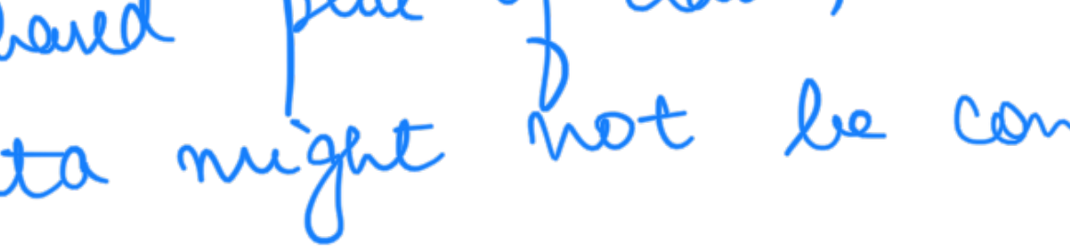
→ Subtracts 1 to 100

→ Pass the shared variable via conn

→ Pass the shared variable via conn



3
6
10



Synchronization problem

When more than one thread is manipulating a shared piece of data, the final value of data might not be consistent

---

When does a sync problem happen

① critical section :

piece of code that is working on a shared piece of data

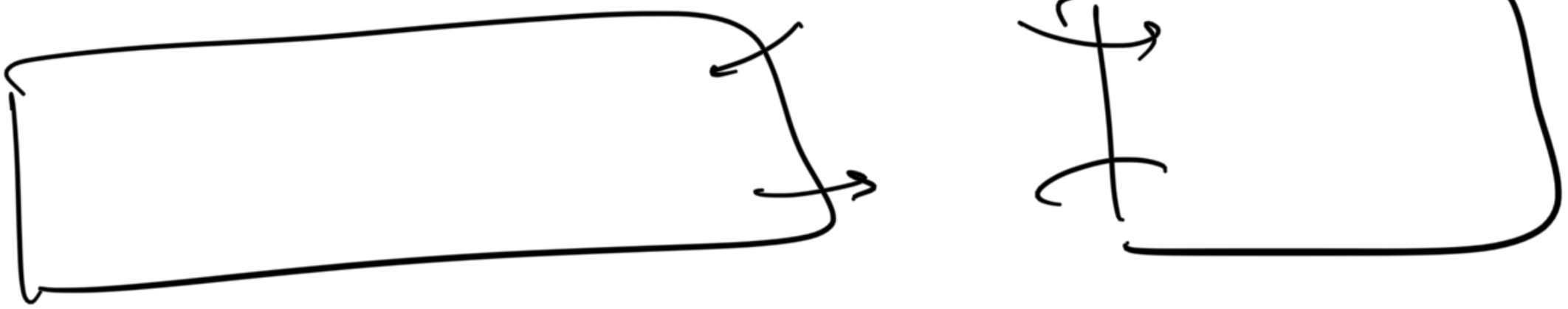
Sync issue can happe

② Race Condition

More than one thread trying to enter the critical section at the same time

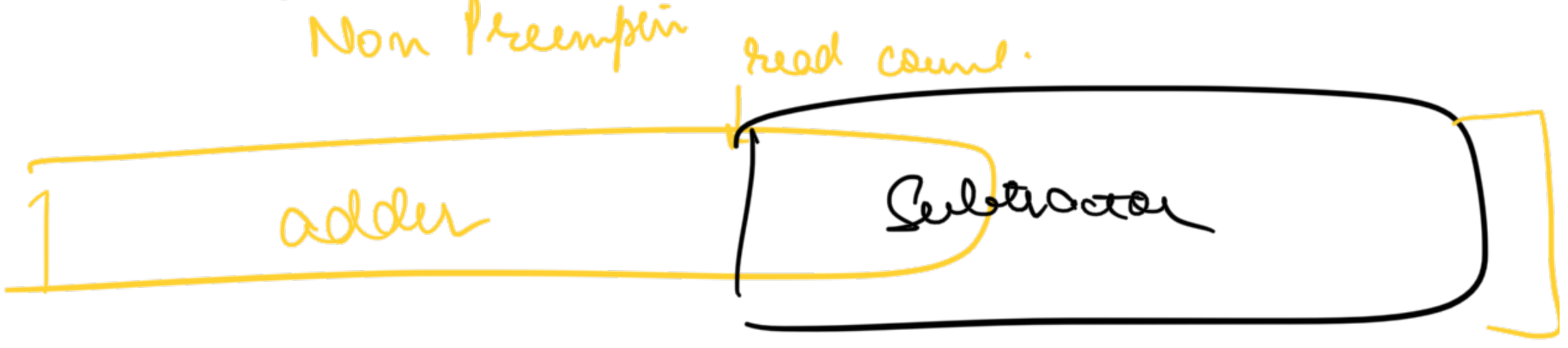
③ Preemption

Can pause a process/thread even before it completes



Assume a single core CPU

Non Preemptive



Non Preemptive ⟺ No Race Condⁿ

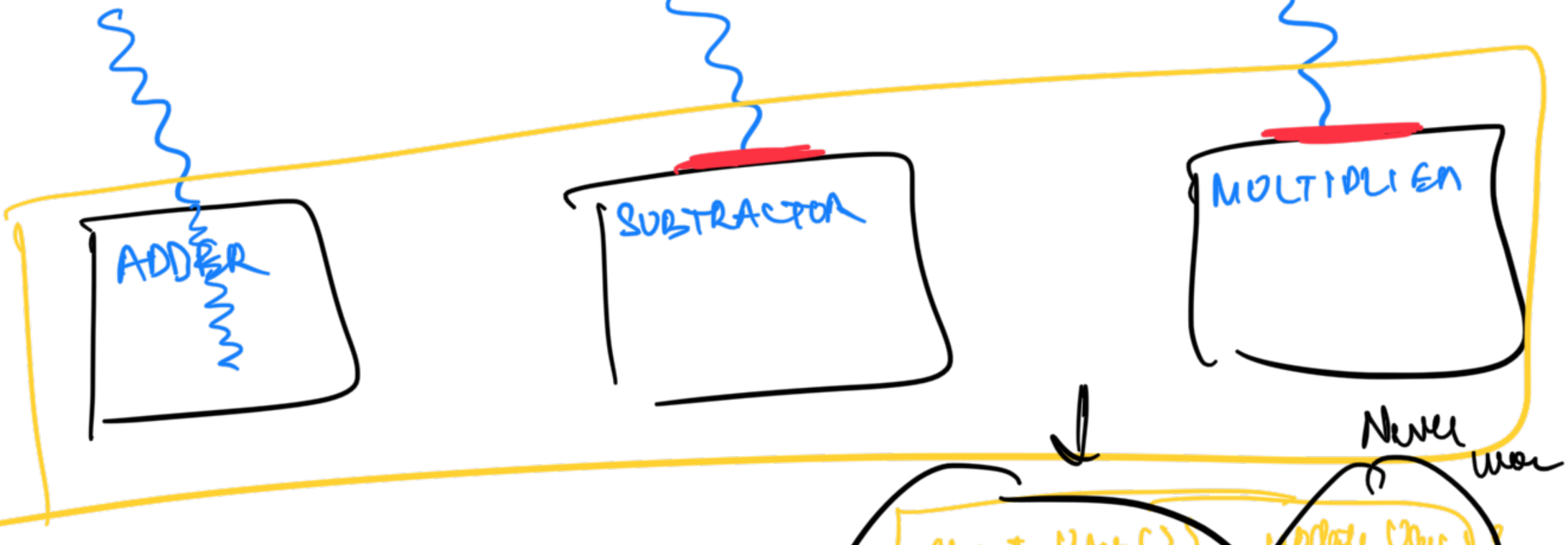
Pre

How to Solve Sync Problem

Properties of a good sync problem solution

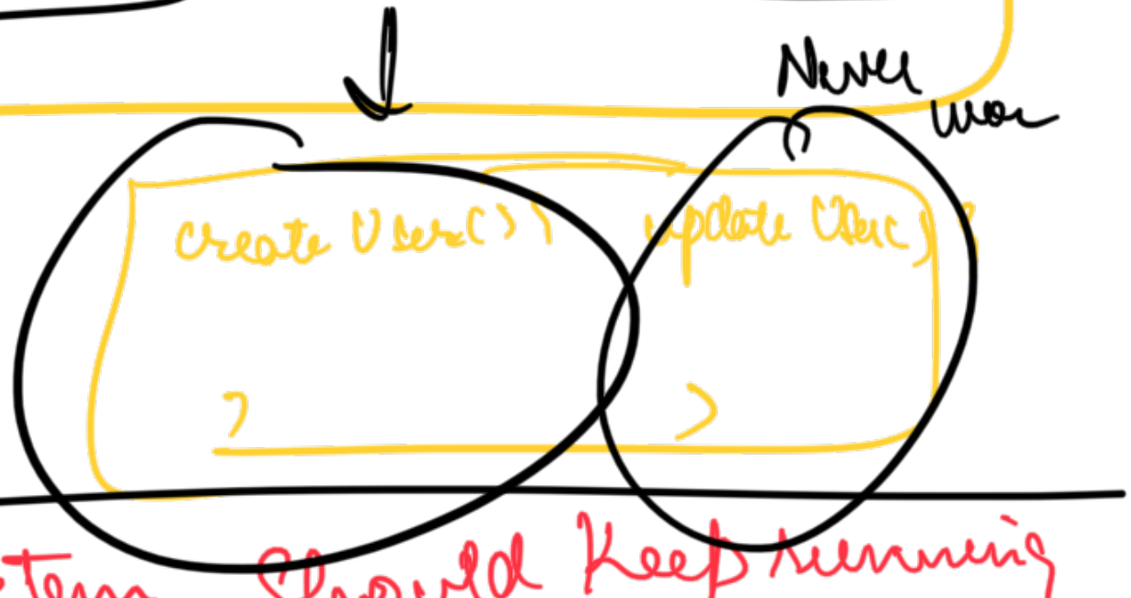
(1) Mutual Exclusion

Only one thread can be enter a CS at one time



(2) Progress

The overall system should keep running



related

③ Bounded Waiting

Not keep waiting infinitely

④ No Busy Waiting

CPU should not spend Time doing useless work

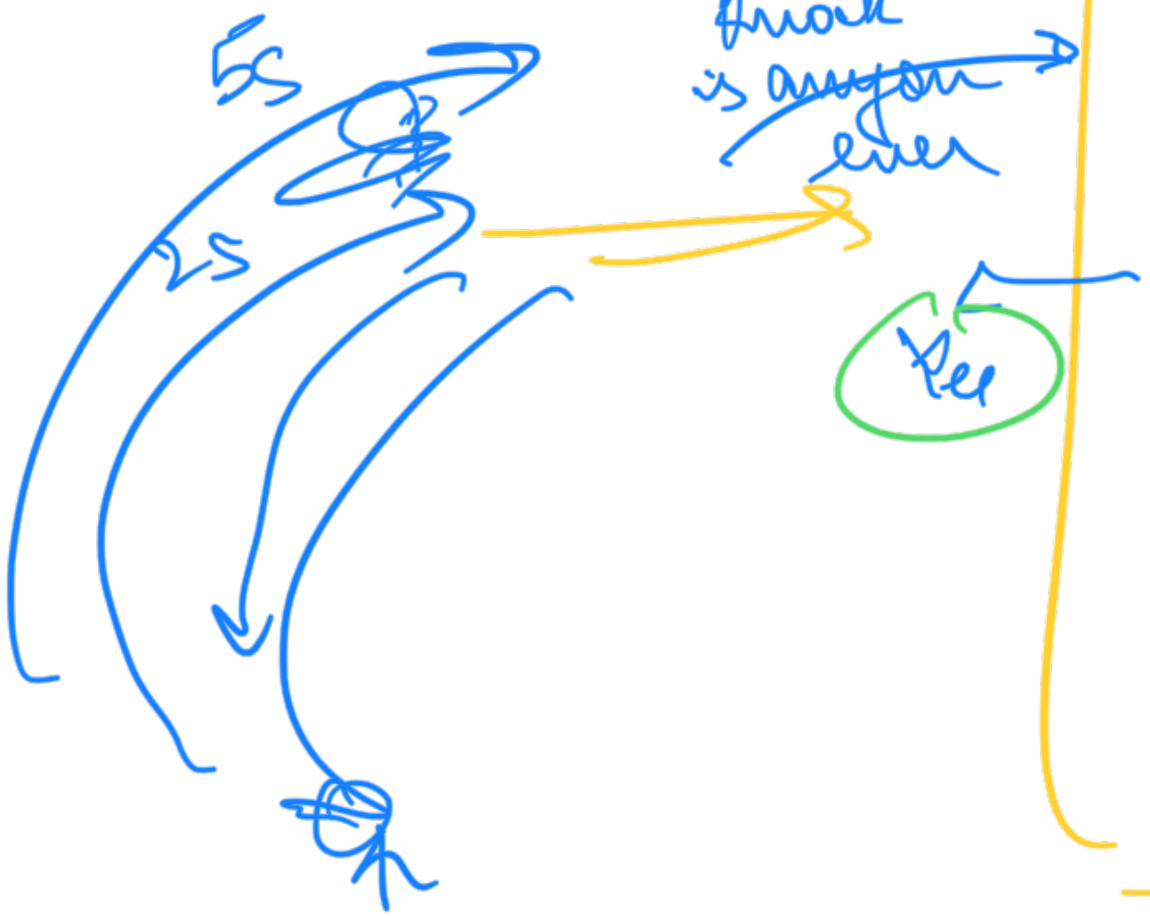
→ (checking if I have access to CS now

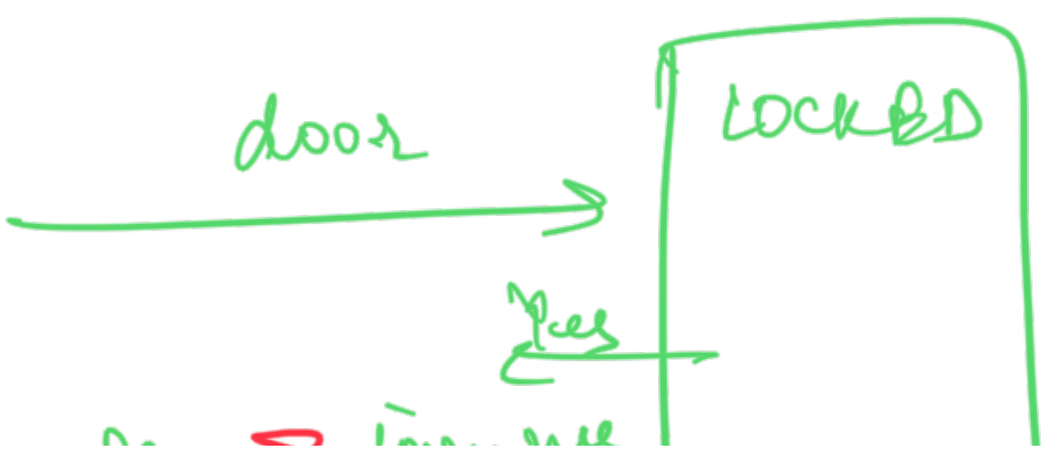
```
while (! lock.available()) {

    wait()

}
```

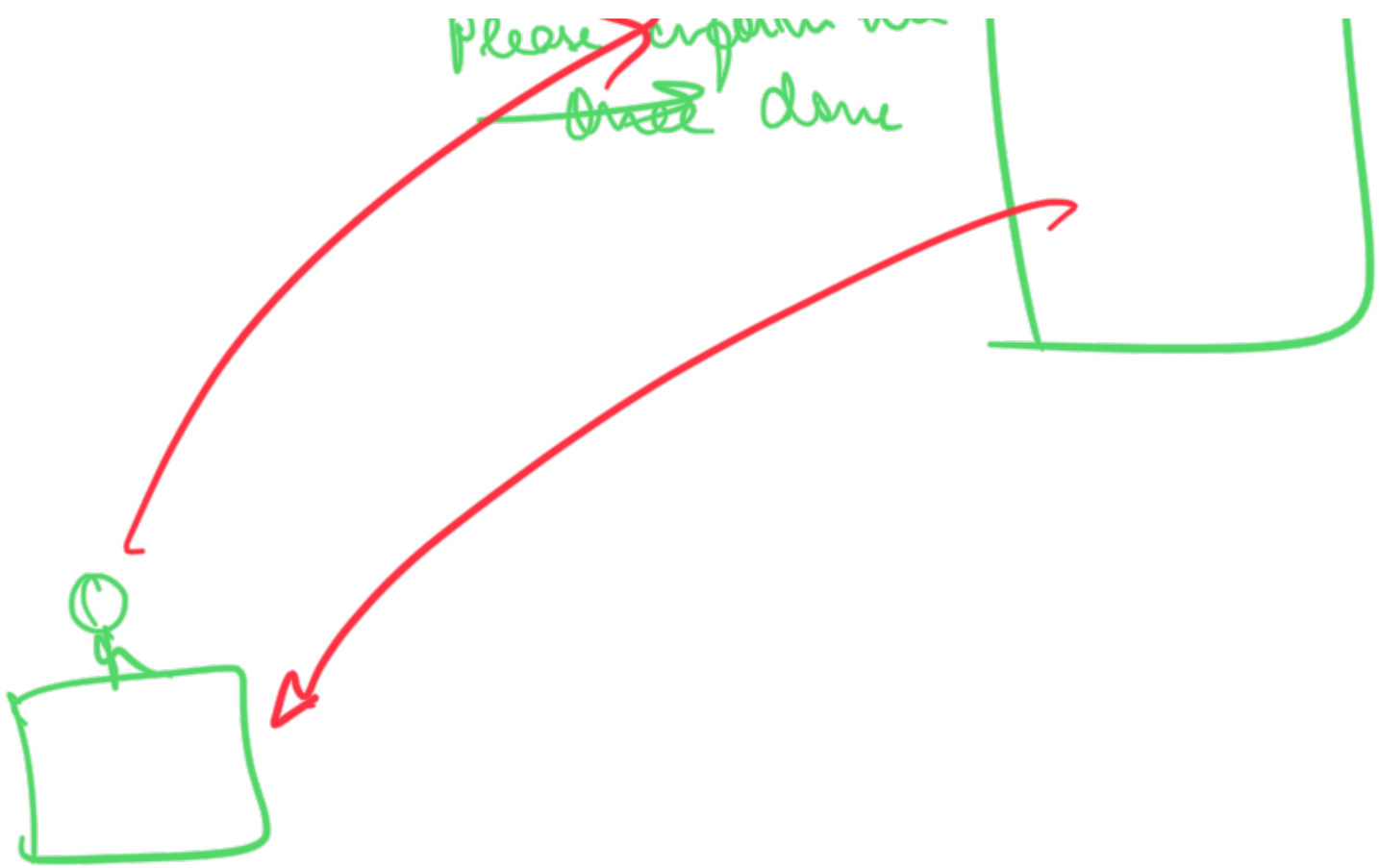
→ while(! open()) {

Thread.sleep(2s)

}

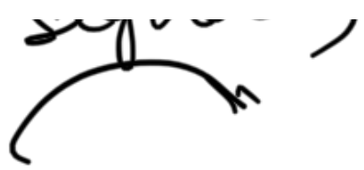






Indefinite Waiting

≠



Solutions

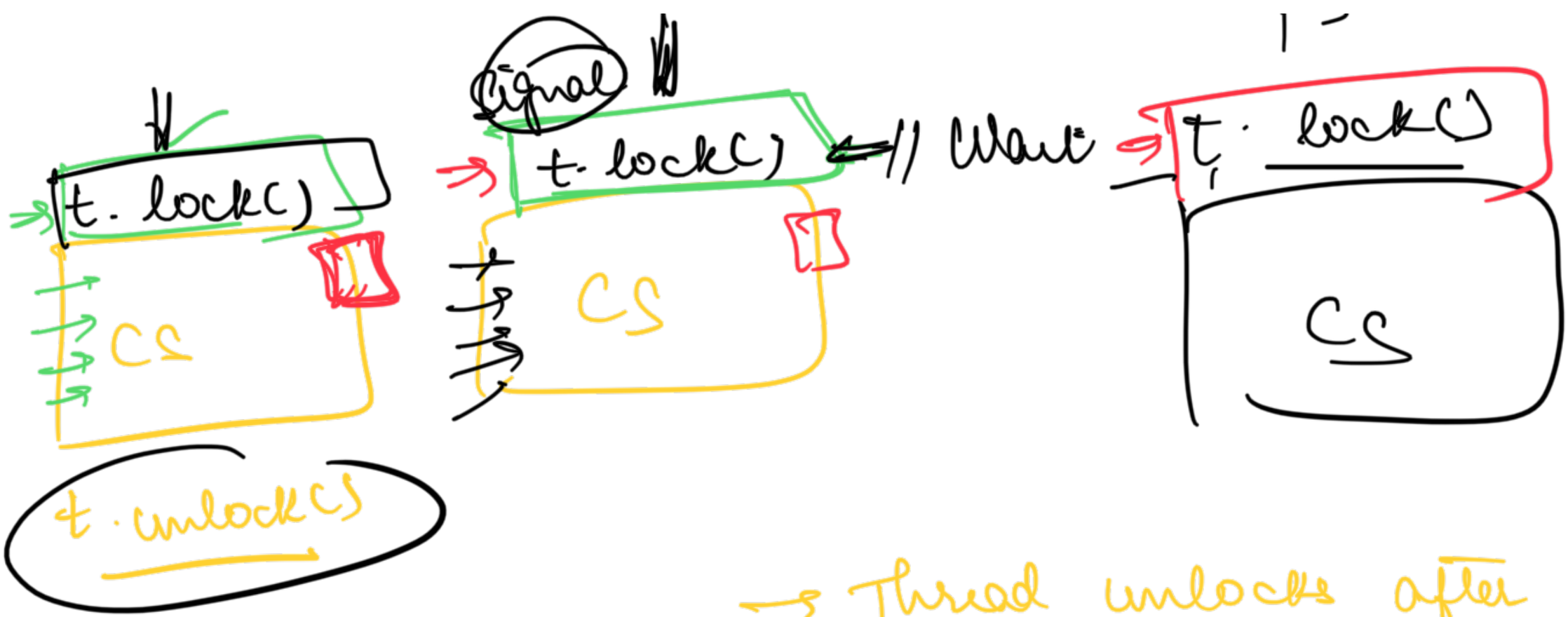
① Mutex / Lock (Mutex lock) (Lock)

Mutual Exclusion

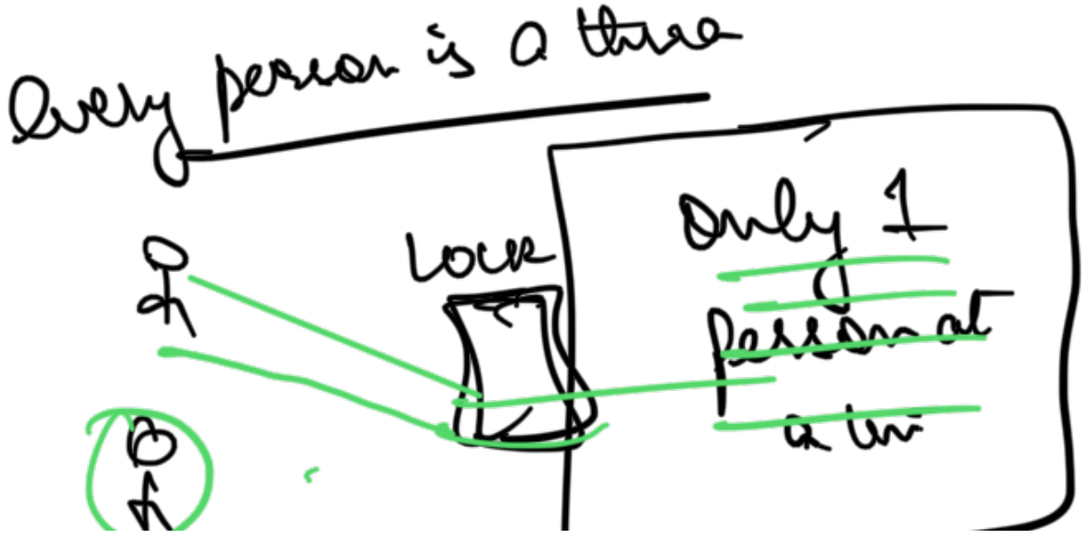
→ A thread before entering a CS takes a lock.

→ Only one thread can take a lock at one time

T1 T2



→ Thread unlocks after critical section



There is only one key

lock for value

lock for count

shared - var

T1

T2

lock. lock()

cs/ count

lock. unlock()

lock. lock()

cs

lock. unlock()

lock for count.lock()

lock for value.lock()

$d = a + b$

Object level lock

Every object has an internal lock

Access that lock by using Synchronized keyword.

Synchronized (count) {

adder

}

lock for Count - locks()

Synchronized(count) {

Sub

}

lockfor Count - lock()

adder

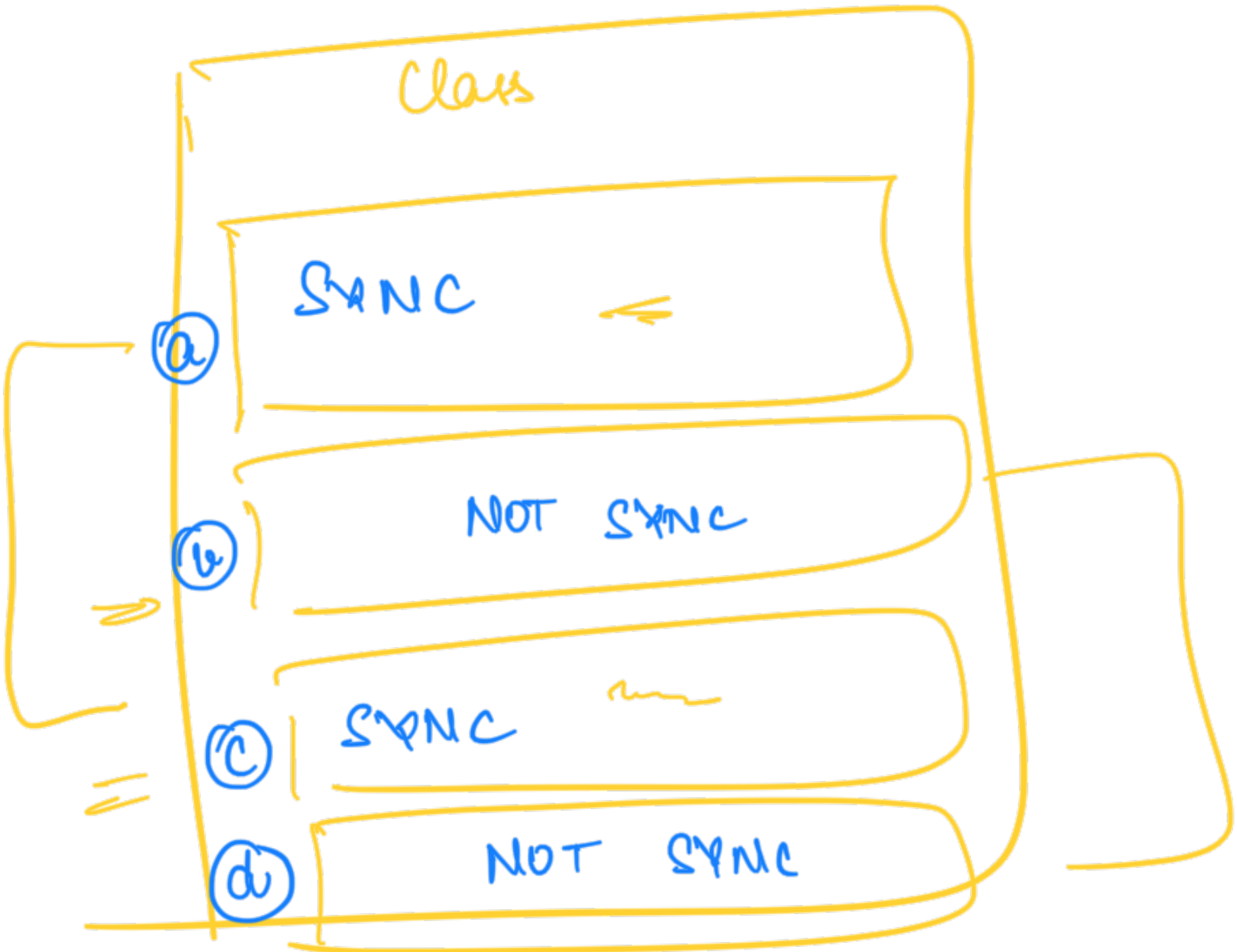
lock for Count · unlock

Cell

lock for Count · unlock

T1	T2

increment (offset) {
read value
temp = value + offset
Store value
	increment (offset) {
	read value
	temp = temp + offset
	Store value
	}



Obj 1.

Obj 2

obj 1. a() obj 2. a()	→ NO LOCKING as 2 diff obj → BOTH RUN
obj 1. a() obj 1. a()	→ Only 1 runs because a is a sync method
obj 1. a() obj 1. b()	→ Both will run because a is syn but b is not syn
obj 1. a() obj 1. c()	→ only 1 will run
obj 1. b() obj 1. d()	

ONLY 1 THREAD can be inside any Sync method of 1 obj at 1 time

Count 1 = new Counter()

Count 2 = new Counter()

value

Count 1 increment (20)

value

Count 2 increment

(10);

Java Conc in Practice