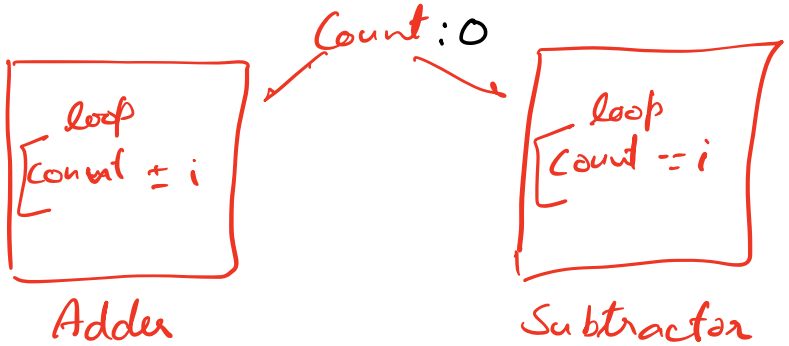
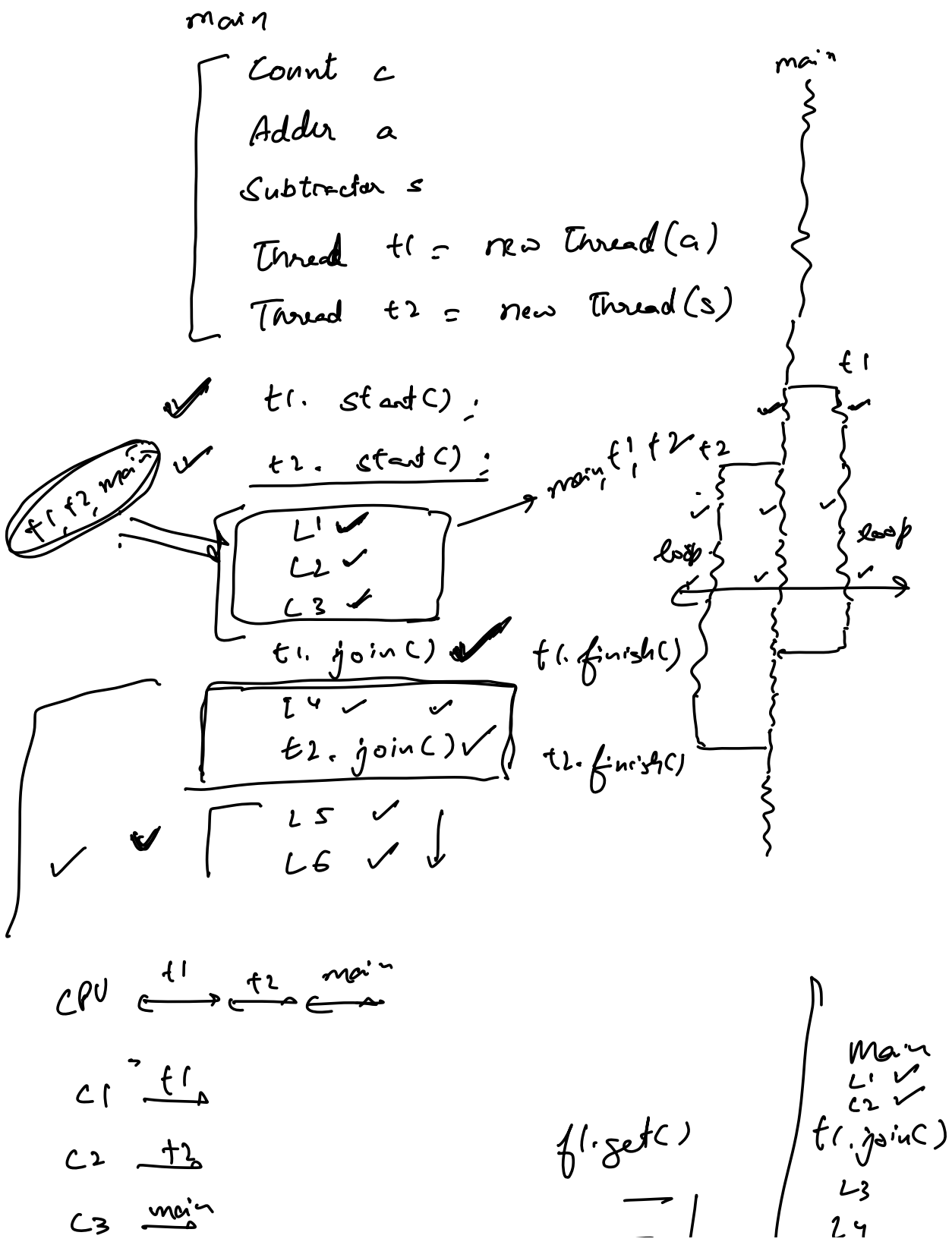
1. Good Evening
2. We will begin at 9:07 pm
3. Topic – Synchronization

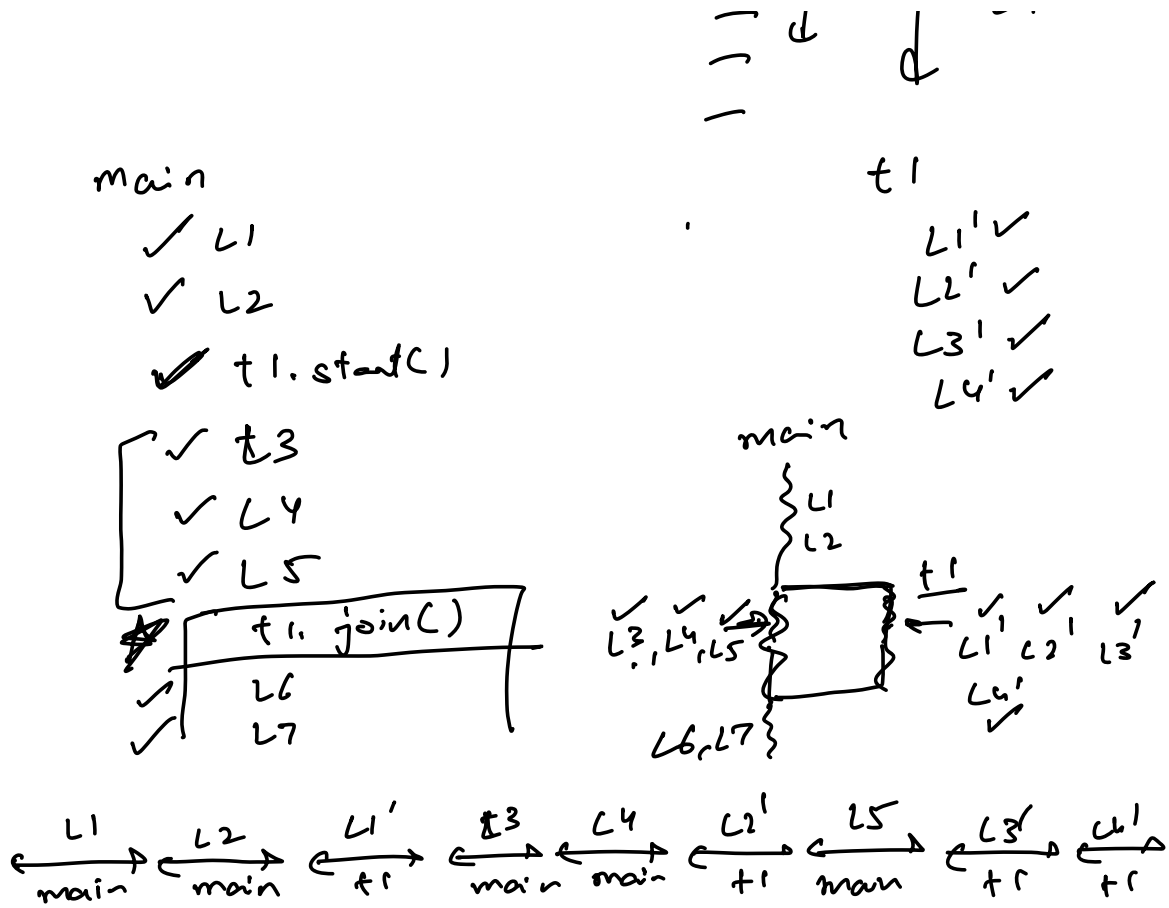
Agenda

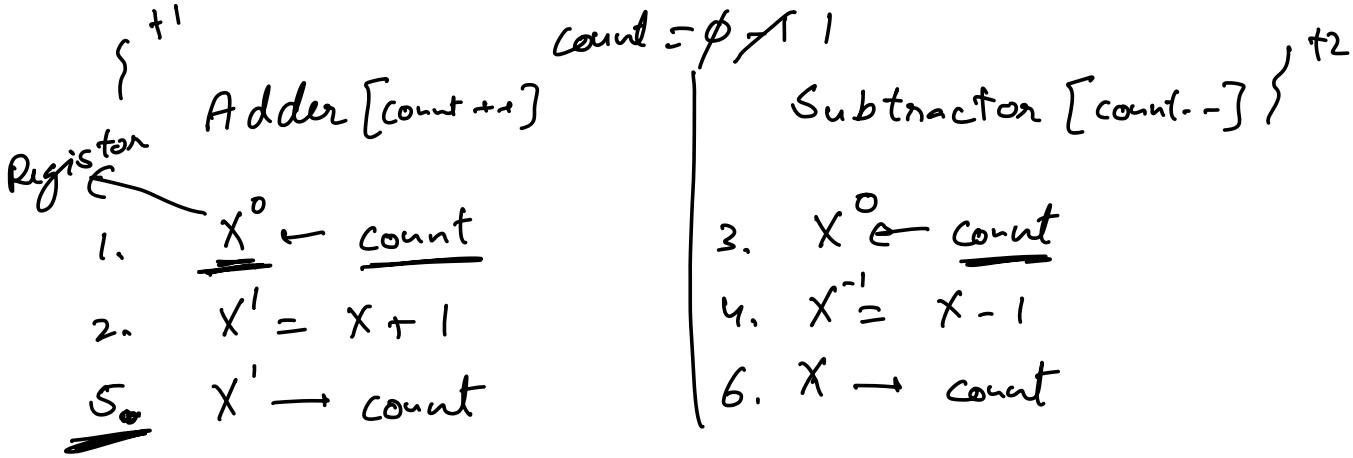
1. Problem of Synchronization
    1. Adder-Subtractor problem
    2. What is problem of synchronization?
    3. Conditions that lead to synchronization problems.
2. Solution to synchronization problems
    1. A theoretical solution : Features.
    2. Practical Solutions
        1. Mutex Locks ✓
        2. Synchronized keyword ✓
        3. Atomic variables ✓
        4. Semaphores : Producer-Consumer Problem

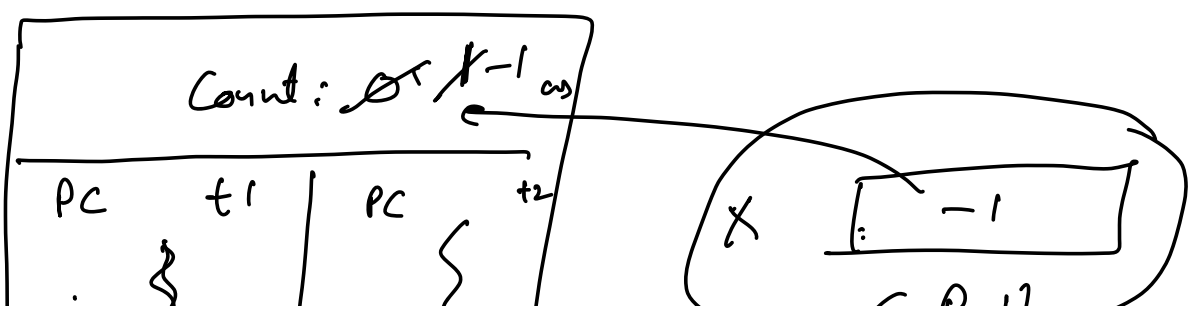
Adder - Subtractor Problem




t1
t2
main
Count c
Adder a
Subtractor s
Thread t1 = new Thread(a)
Thread t2 = new Thread(s)
t1.start();
t2.start();
t1, t2, main
L1
L2
L3
t1.join()
L4
t2.join()
L5
L6
main
t1
t2
loop
loop
t1.finish()
t2.finish()
CPU
t1
t2
main
c1
t1
c2
t2
c3
main
t1.get()
main
L1
L2
t1.join()
L3
L4


main
L1
L2
t1.start()
t3
L4
L5
t1.join()
L6
L7
t1
L1'
L2'
L3'
L4'
main
L1
L2
t1
L3, L4, L5
L1' L2' L3'
L4'
L6, L7
L1 main
L2 main
L1' t1
t3 main
L4 main
L2' t1
L5 main
L3' t1
L6' t1


t1
Adder [count++]
Register
1. X^0 ← count
2. X' = X + 1
5. X' → count
count = 0 → 1
t2
Subtractor [count--]
3. X^0 ← count
4. X^-1 = X - 1
6. X → count


Count: 0 → X-1 ans
PC t1
PC t2
X
-1

Problem of synchronization : If multiple threads will work on shared data, at the same time, we might get inconsistent / wrong results because of CPU scheduling.

When will synchronization problems happen? Or the conditions that lead to synchronization problems?

0. Multiple threads & shared data.

1. Critical Section : Part of code working on shared piece of data.

count

Adder t1

1. print hi
2. count += 1
3. print bye

Subtractor t2

1. print hello
2. count -= 1
3. print see you.

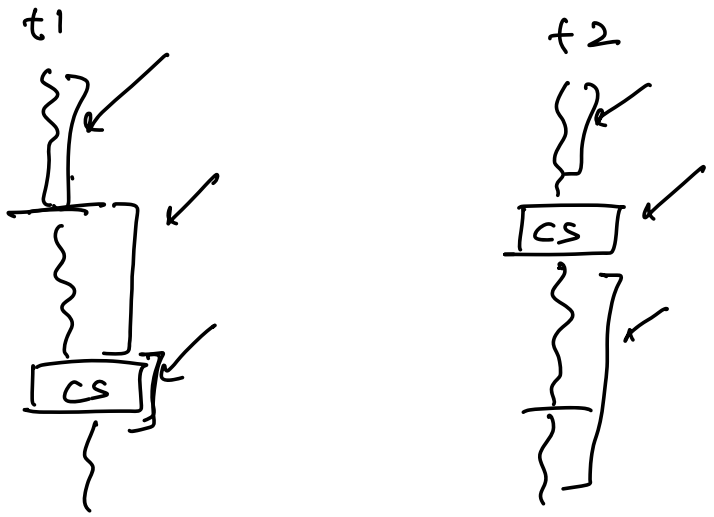
Squarer

print hi there

```
count = getCount()
count = count * count
count += offset
```

print bye there

2. Race Condition : More than 1 thread trying to enter critical section at the same time.



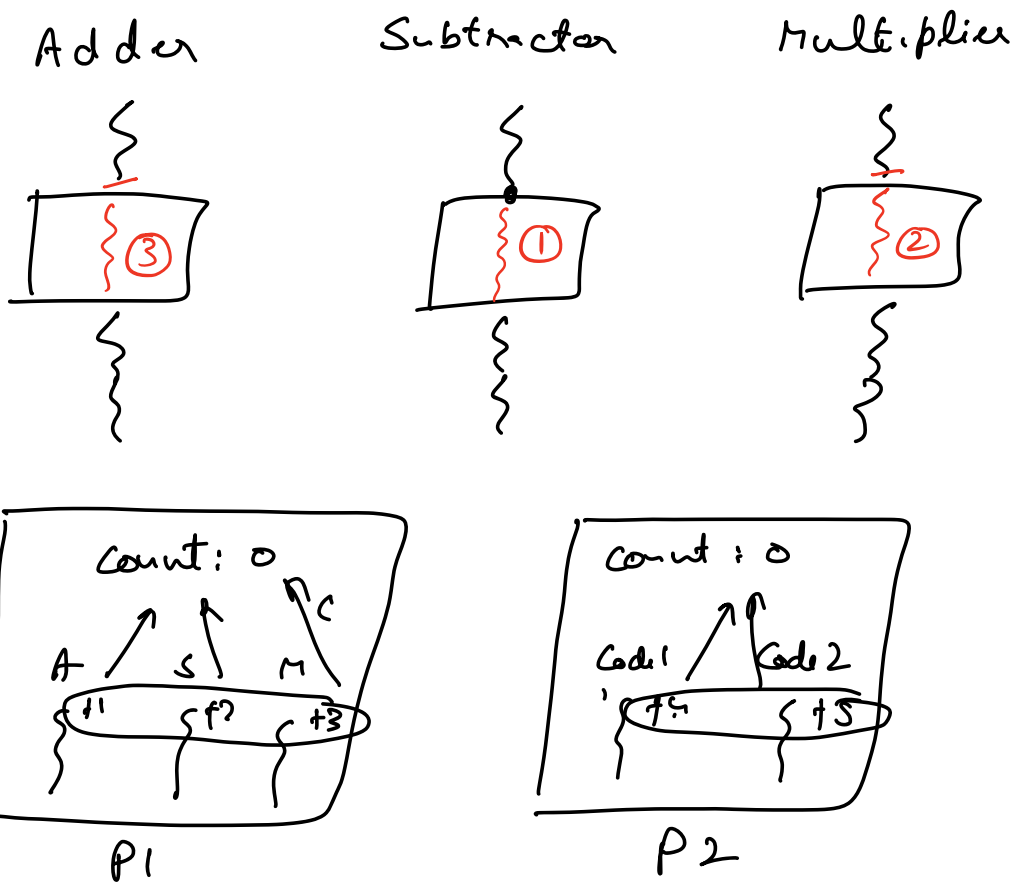
3. Pre-emption : Of a thread executing it's critical section gets "context-switched" out we may have synchronization problems.

Solutions

↳ A theoretical solution

What features should a synchronization solution have?

1. Mutual Exclusion : Only 1 thread should execute it's critical section at a time. → prevent race conditions.

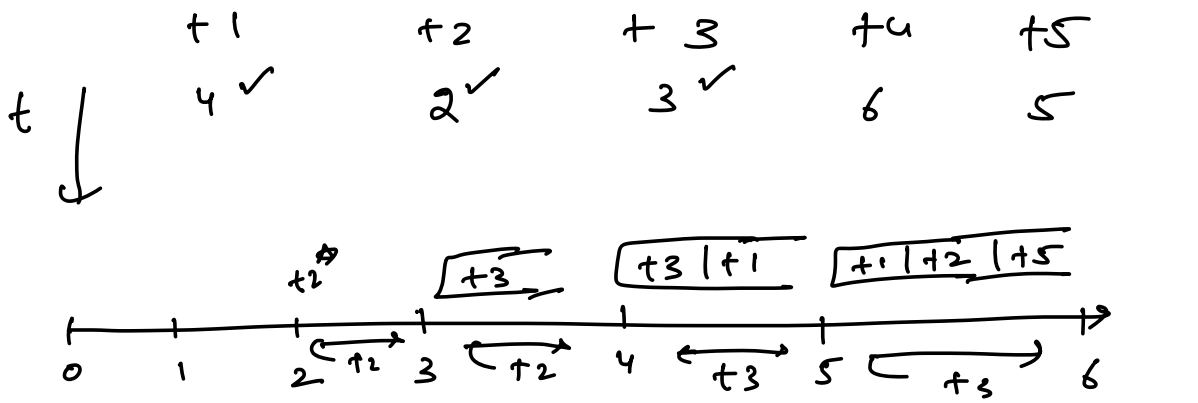


2. Progress : Entire application should not stop. At least 1 thread should

be making progress.

3. Bounded Waiting → No thread should wait indefinitely to enter it's critical section.

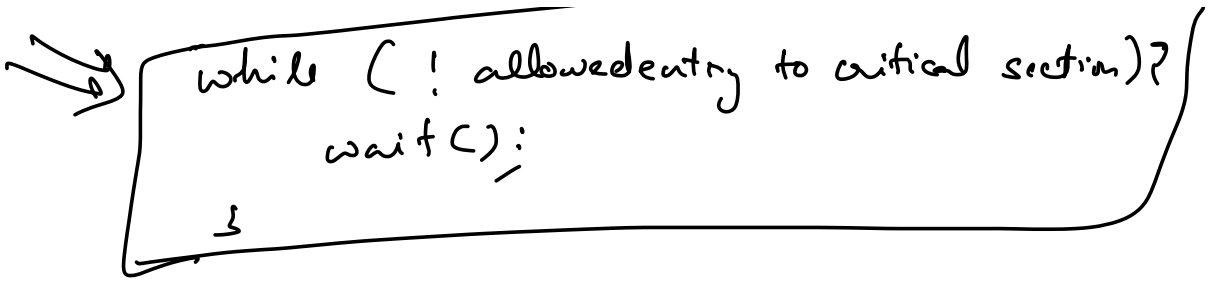
→ Threads should be given access to critical section in FCFS manner of request.



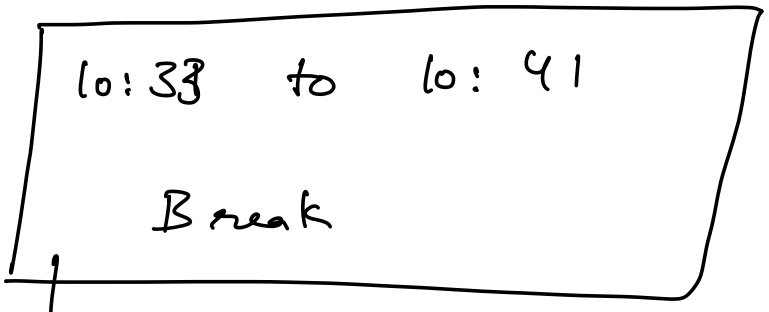
t1 | t2 | t5 | t6

t6 should not access critical section before t1, t2 & t5

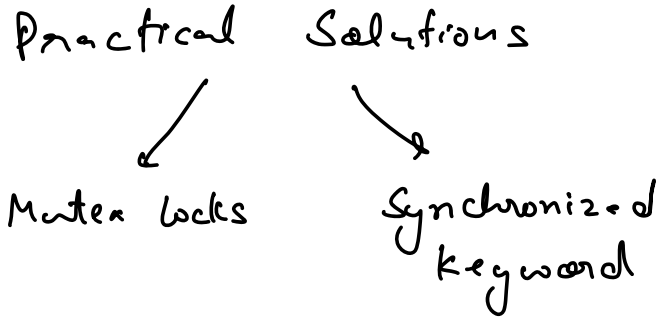
4. No busy waiting : Busy waiting code should not be there.



☆ Notification mechanism instead of busy waiting.

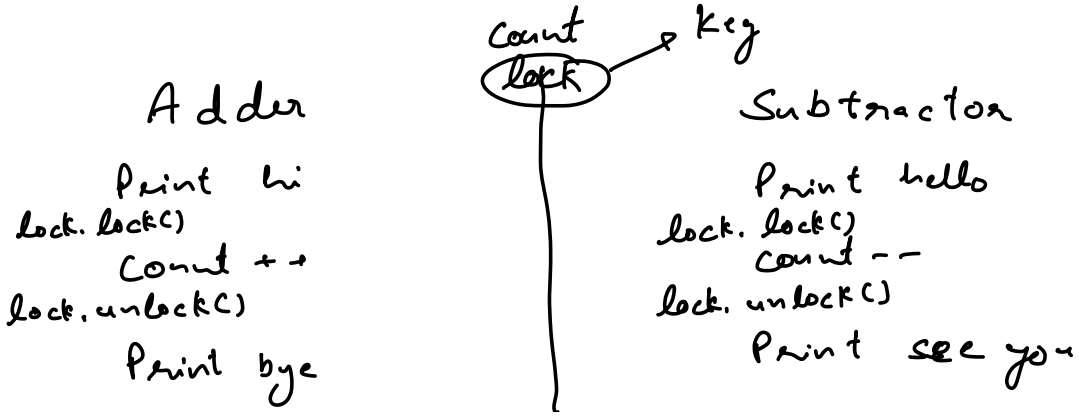


Practical Solutions



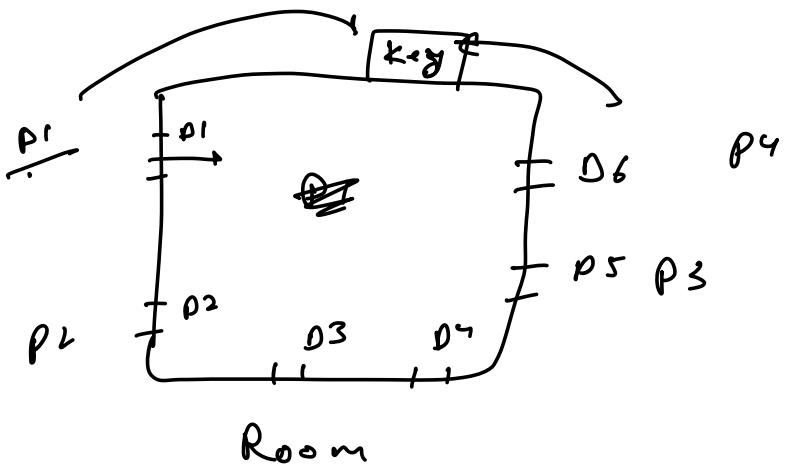
Practical Solutions

1. Mutex Lock :

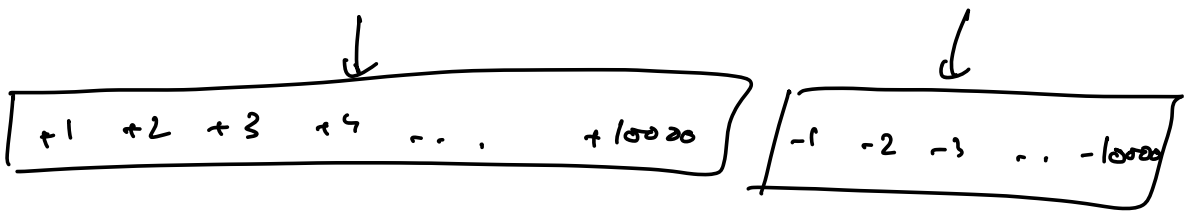


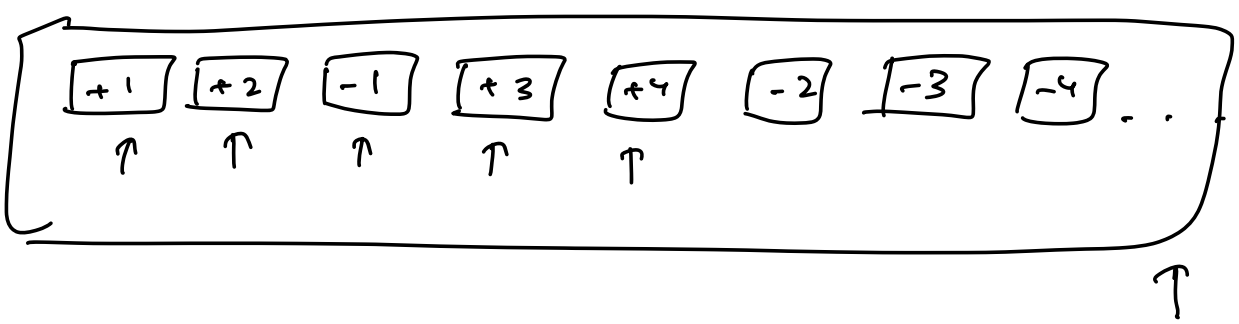
→ Before entering the critical section the thread must take a lock

→ While leaving the critical section thread should release the lock



Lock on entire for loop



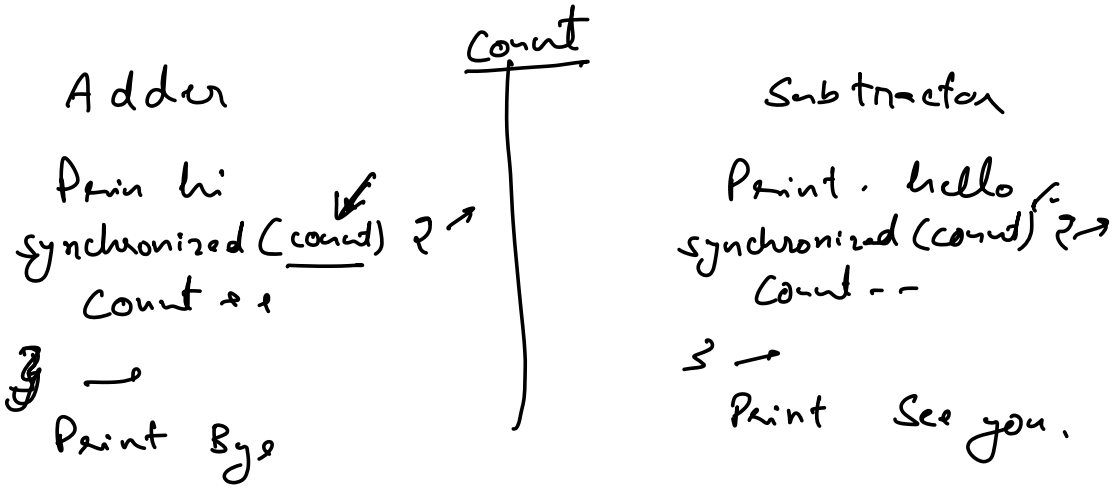


2. Synchronized keyword

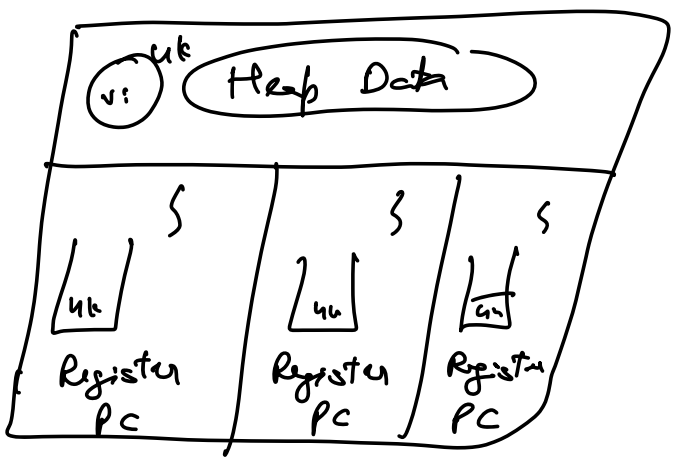
☆ Mutex is an OS concept

↳ ✡ Java specific keyword

In Java locks are available on all objects.



✡ Threads can share data on the heap.



✡ Volatile

Synchronized Methods

↳ 2 Perspectives

- Use an API developed by someone else.
- You have to develop the API used by other people.
  - You want to make sure that your API / DS will work perfectly in multi-threaded applications
    - ↳ methods synchronized.

Count
value X

Count
incrementValue()
getValue()

Synchronized Method

→ If a thread enters a synchronized method on an object, no other thread

can enter any synchronized methods on the same object.

Count

- incValue() ✓
- decValue() ✓
- getValue() ✓

c1 , c2

t1	t2	will they work in parallel?
c1.incValue() ✓	c1.incValue()	✗
||	c1.decValue()	✗
||	c1.getValue()	✓
||	c2.incValue	✓

✓ HashMap → Take locks Onself

Hashtable → ✓✓ [... we have synchronized methods

✓ StringBuilder Vs StringBuffer ✓

↓ not synchronized

↓ Synchronized

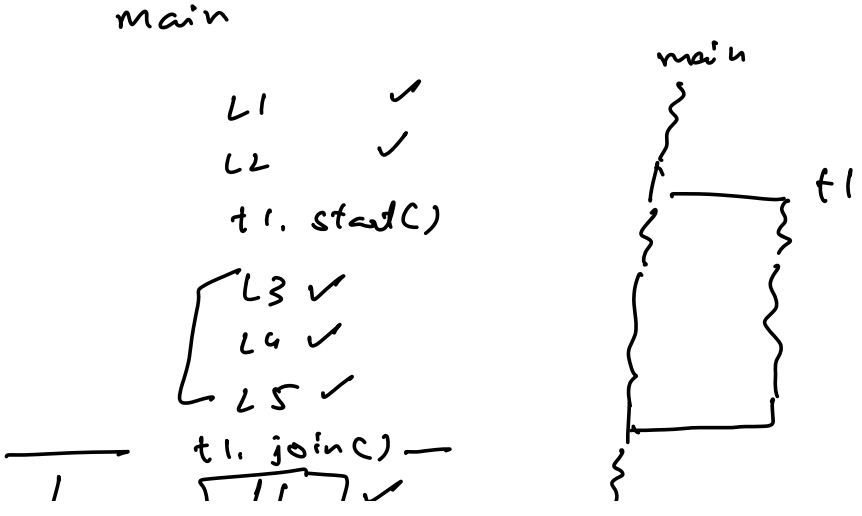
Single-Threaded code → Use Hashmap & StringBuilder

Multi- " " → Use Hashtable & StringBuffer.

Agenda

1. Producer - Consumer problem
2. Semaphores
3. Atomic Variables (& volatile)
4. Deadlocks.
5. Memory Management

Join



ا