Agenda :

1) Semaphores for Synchronisation [PRODUCER CONSUMER PROBLEM]

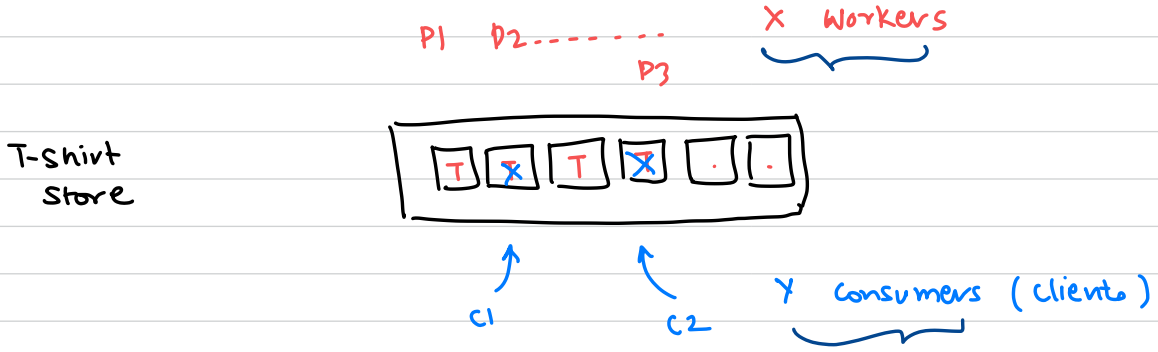
2) Concurrent Data Types

3) Deadlocks (Next Class)

Synchronisation

1) Mutex / Lock
2) Synchronized
3) Semaphores

PROBLEM:



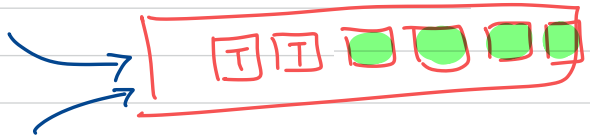
Constraints:

⇒ I want to allow a consumer to buy if a T-shirt is avaible for them

⇒ " " " " " producer if there is space available for them to put the Tshirt.

X Many Producers Threads

Max No of Producers : at a given time = No of empty display slots.



Shared Object (Fixed Size Queue)

Y Many consumer Threads.

max No of consumers at give of time = No of filled slots

Concurrent

Queue < Shirt > q ;

MAX-SIZE = 6

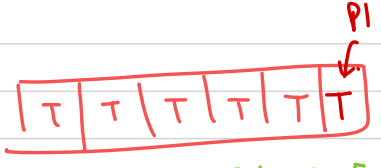
PRODUCER

```
if ( q.size() < MAX-SIZE ){
    q.add(new Tshirt());
}
```

Consumer

```
if ( q.size() > 0 ){
    q.remove()
}
```

Many Producer & Consumer Threads



P1	P2
5	5
6	7

P1

1 ⇒ if (q.size() < MAX-SIZE){

3 q.add(new Tshirt());

}

P2

2 ⇒ if (q.size() < MAX-SIZE){

4 q.add(new Tshirt());

}

P3 . . .

C1

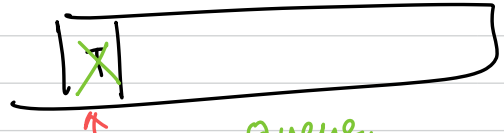
```
if ( q.size() > 0 )
    ↳ q.remove()
```

error

C2

```
if ( q.size() > 0 )
    ↳ q.remove()
```

: Mutual Exclusion: only one P/C can access the queue at a time

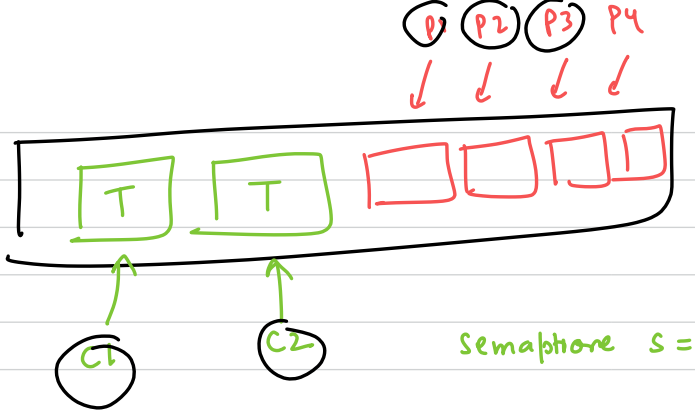


Queue.

Synchronised → Problem ? Yes

is it fast enough ?

Semaphores



Semaphore S = new Semaphore (2),

No of threads that can acq. lock at same time.

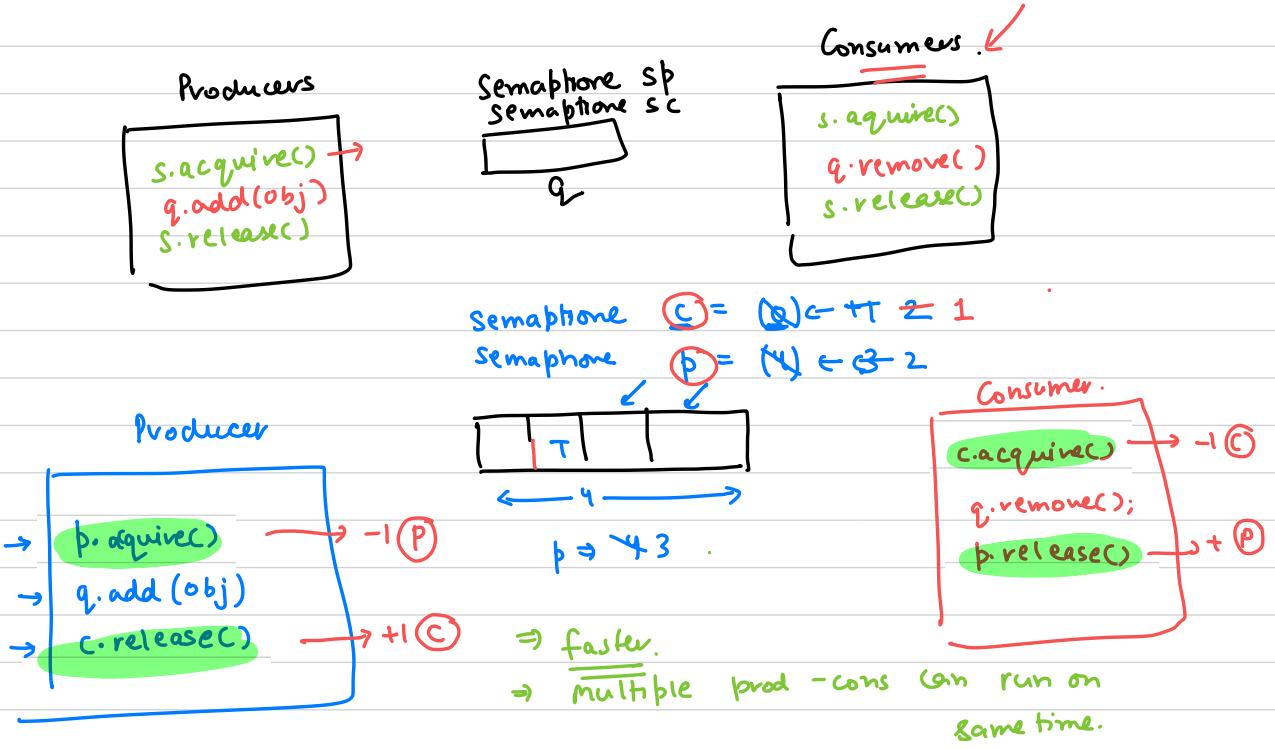
↳ nothing kind of locks, more than one thread can acquire at same time.

[Mutex/lock : Semaphore with value 1]

- lock()
- unlock()

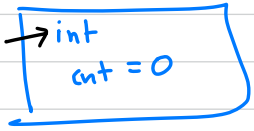
- acquire()
- release()

- How to use this lock?
- What will ~~be the size of lock~~? ?
- How many producers/consumers will are allowed to enter the store at same time?



Atomic Data Structures

10 min break

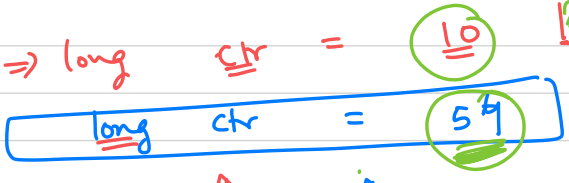


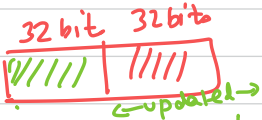
cnt ++ ;

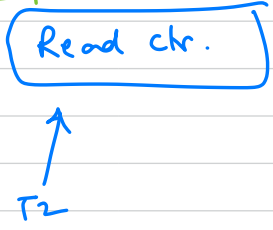
single unit of execution

cnt -- ;

→ Not atomic but we wanted to make it atomic.

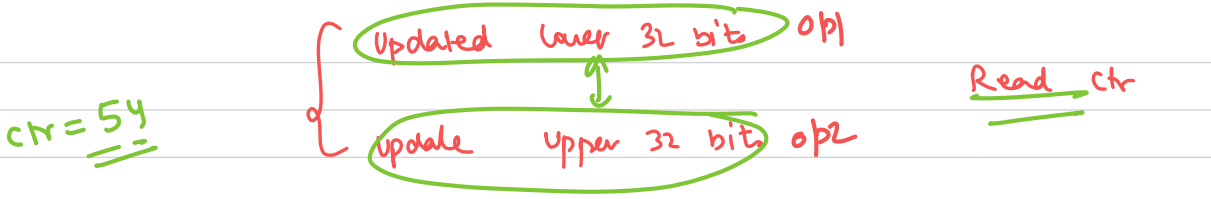




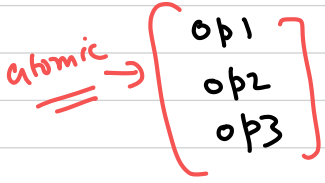


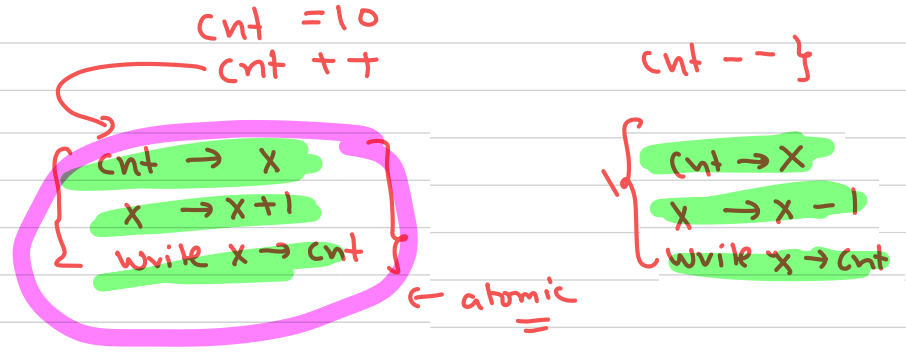
general

- Reading & writing primitive data-types are atomic except long and double
- Assignment is also atomic.



Atomic Data Types :



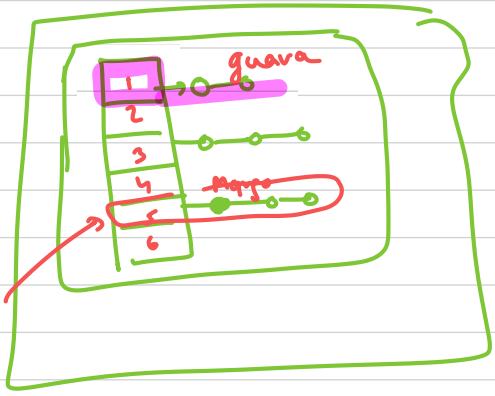


Concurrent Data Structures

↳ Queue

↳ Concurrent Hashmap (★)

↳ fine grained locking



- insert ("Mango")

T1

- Read ("guava")

T2

Additional Problems:

https://leetcode.com/problems/print-in-order/
https://leetcode.com/problems/the-dining-philosophers/ https://leetcode.com/problems/fizz-buzz-multithreaded/ https://leetcode.com/problems/building-h2o/

Additional Course(Optional)

https://www.udemy.com/course/multithreading-and-parallel-computing-in-java/