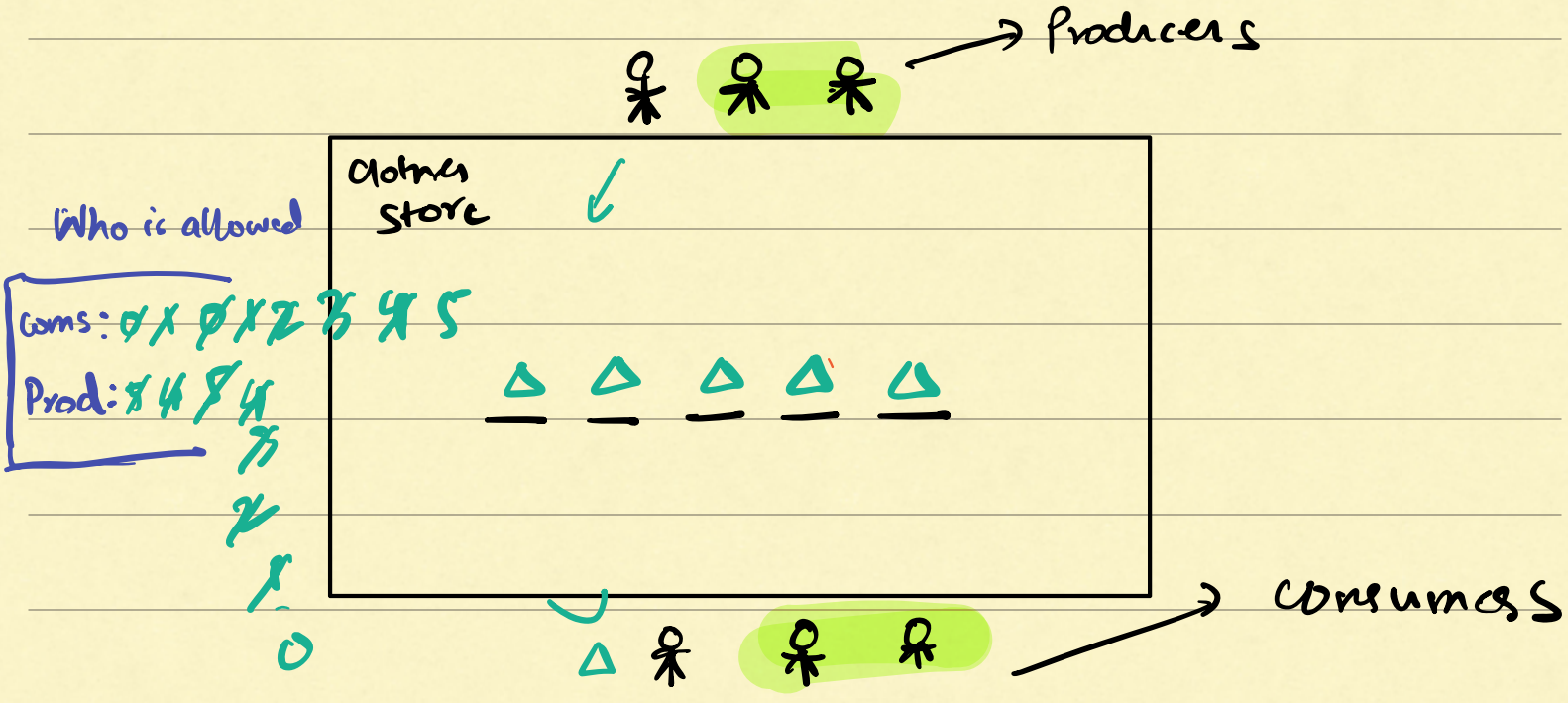
Agenda :

Producer Consumer

- Synchronised with locks
- Semaphores

Atomic Integer

Producer Consumer :



V1 : Synchronisation Issues

V2 : Synchronised but allowed only 1 thread to work on store.

is having #1 thread in CS → Critical section always

beneficial ? → No

> # Producer + # Consumer <= size of store.

- Some sort of signalling mechanism to signal consumers/producers.

producer = 5 ✓

consumer = 0 ✓

producer :	consumer :
producer--;	consumer--;
produce the item.	consume the item;
consumer++;	producer++;

Semaphores : • Mutex + count

• Signaling

• It allows multiple threads in one CS.

Semaphores :

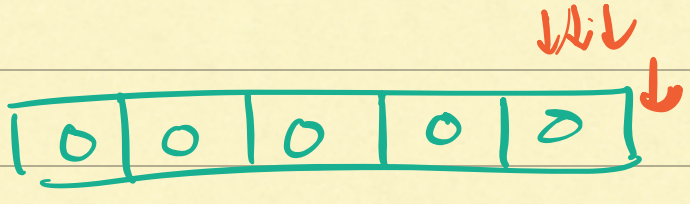
Semaphore x = new Semaphore( ) ← initial count.

x.acquire()	x.release

if (x.curr_val > 0){
x.curr_val--;
proceed to task
} else {
wait();
}
	x.curr_val++;

```
Semaphore prodSema = new Semaphore(5);
Semaphore consSema = new Semaphore(0);
```

prodSema.acquire();	consSem.acquire();
producing the item;	consume item;
consSema.release();	prodSema.release();



→ Atomic Integer

CAS

CAS

↳ Compare & Swap

↳ compare & Swap Algo

value += i ;

↳ t ← value | x ← value

t = t + 1

if (value == x) {

t → value

} else {

retry

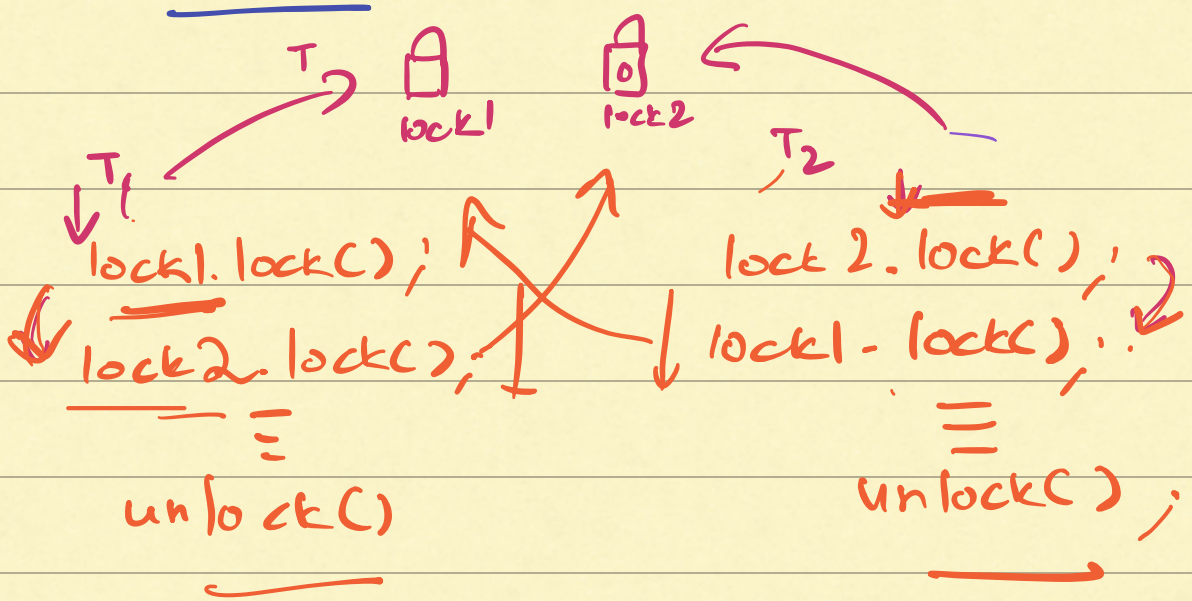
}

↔ → has it changed.

count = 0

T_2	T_1
count -= 1 ;	count += 1 ;
(2) t ← count	(1) t ← count x ← count, x = 0
(3) t = t - 1	(6) t = 0
(4) t → count	(7) t = t + 1, t = 1
(5) count = -1	8 if (count == x) { (changed to -1)
	(9) t → count
	(10) count = 1
	} else { retry the steps
	}

Deadlock :



How to prevent :

1. Dead lock Avoidance → Developer's help maintain ordered locks.

2. Deadlock Ignorance → kill one thread in the deadlock after a timeout.