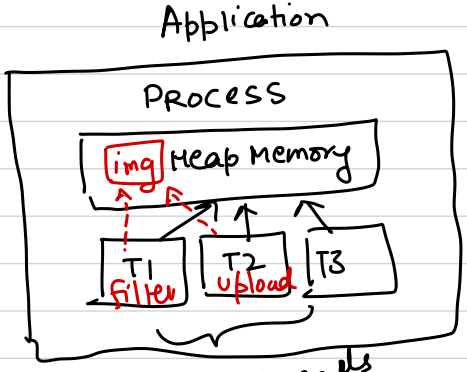
# Synchronisation

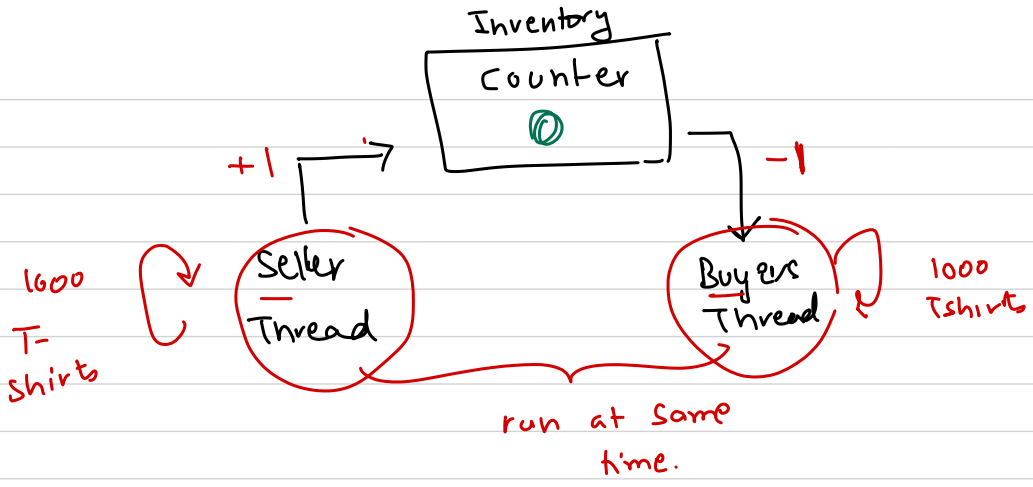
- → Problem
- → Solutions




3 Threads share all objects created within the app.

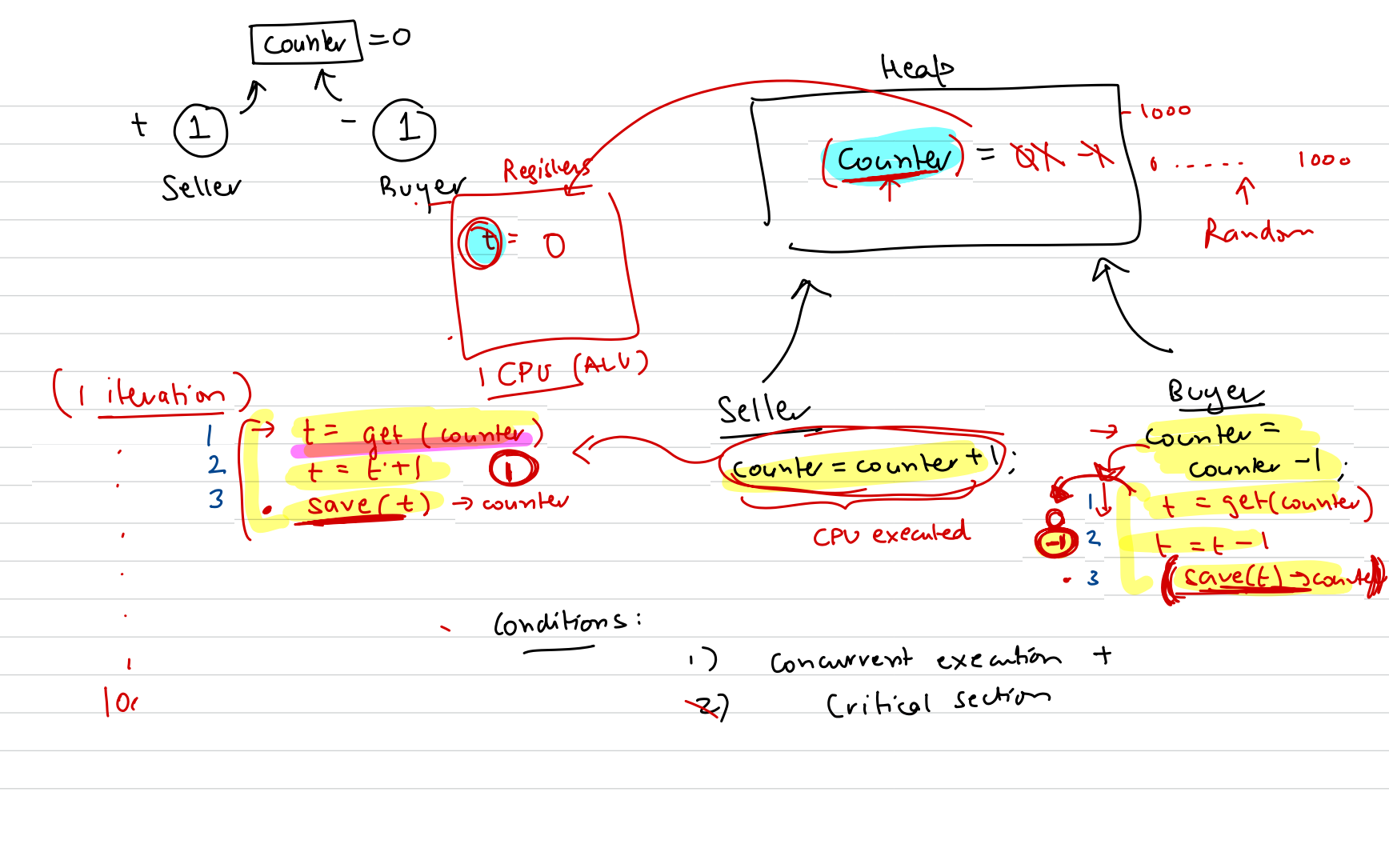
Within a process Heap memory is shared across all threads.

→ we don't need to create separate copies of object,




Seller stored 1000 Tshirts.
Buyer bought 1000 Tshirts

Expected : 0

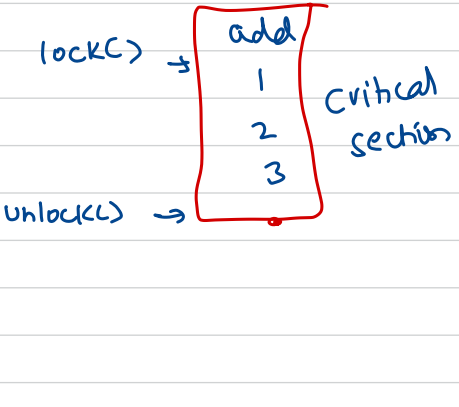

Counter = 0
+ 1
- 1
Seller
Buyer
Heap
Counter = 0
1000
0 ..... 1000
Random
Registers
t = 0
1 CPU (ALU)
(1 iteration)
1
2
3
t = get (counter)
t = t+1
save(t) → counter
Seller
counter = counter + 1;
CPU executed
Buyer
counter = counter - 1;
t = get(counter)
t = t - 1
save(t) → counter
Conditions:
1) Concurrent execution +
2) Critical section

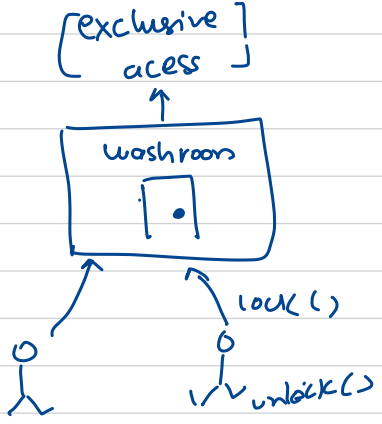
Solutions :

Mutual Exclusion.

① Locks / Mutex locks

locks ensure only one thread can acess the critical section a time.



(2) Synchronized Keyword

↳ Synchronized Methods
↳ Synchronized Block : provides a more fine-grained locking.

---

Atomic Datatypes

↓

Java has some data types designed for concurrent / multi-threaded env.

Atomic Integer

[ Concurrent Data Structures ]

↑

designed to support concurrency in Java Collections.