Agenda

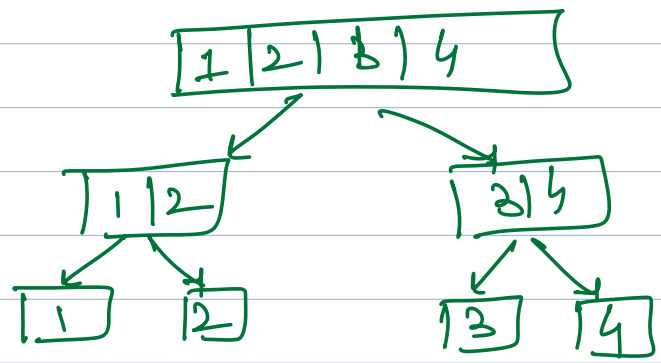
1. Callables → Multithreaded Merge Sort
2. Adder Subtractor
3. Synchronization
4. Sol^n of Sync
   - → Mutex

Next Class : continuation of Synchronization
- → Semaphores
- → Atomic Data Types
- → synchronized keyword

CALLABLES

1. Sorter

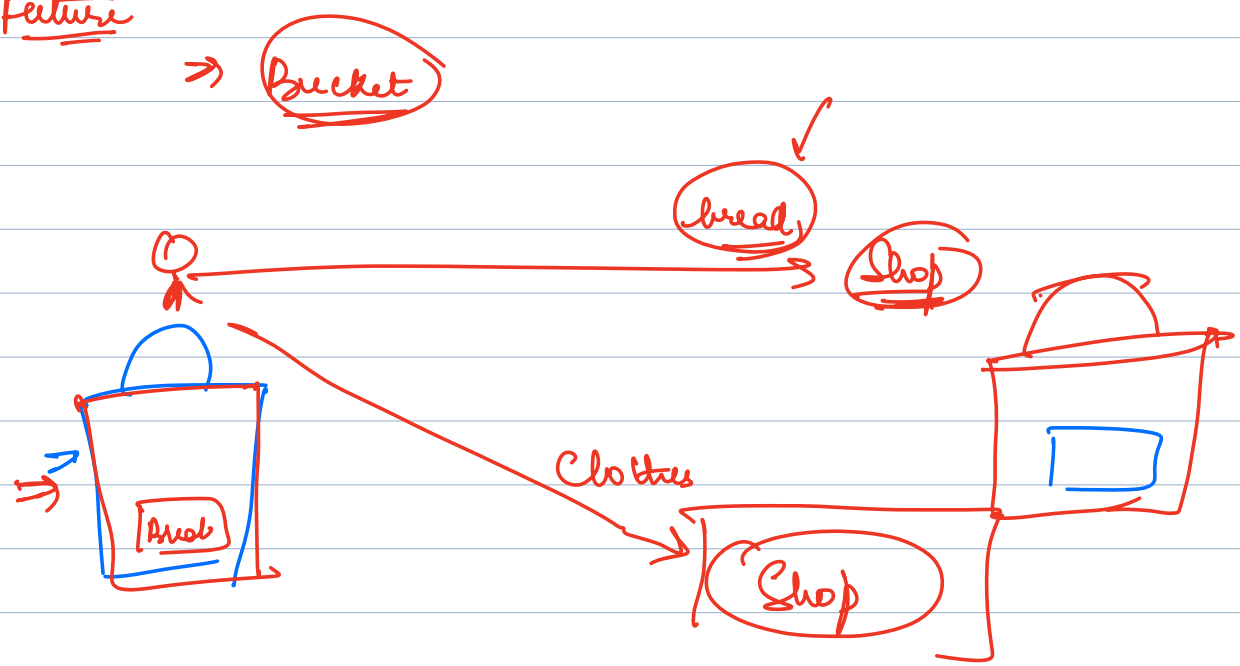
2. Sorter implements Callable<List<Integer>> { (generics)
   - Sorter(List<Integer> arrToSort)
   - List<Integer> call() {
     - — — sort the array
   - }

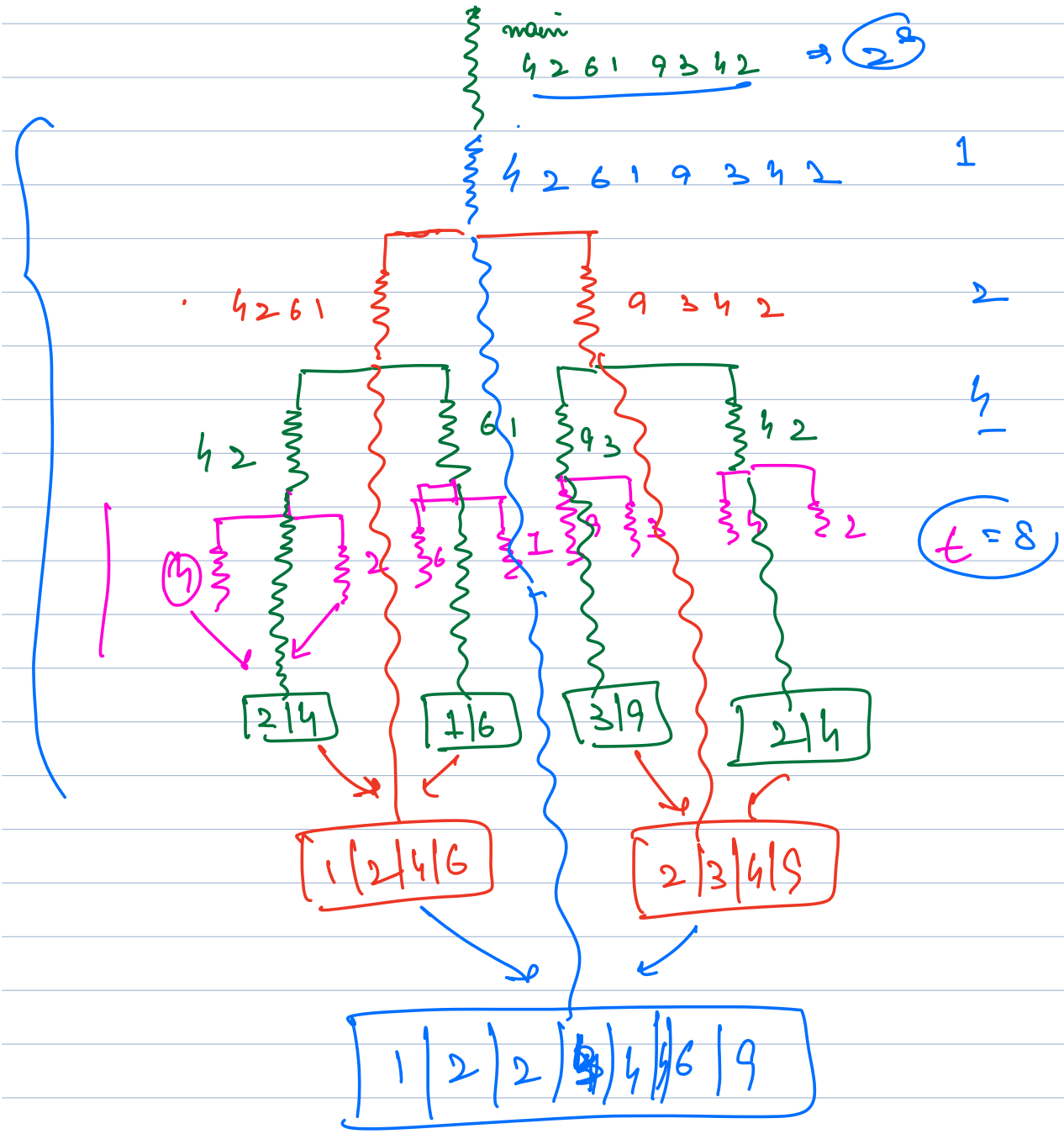


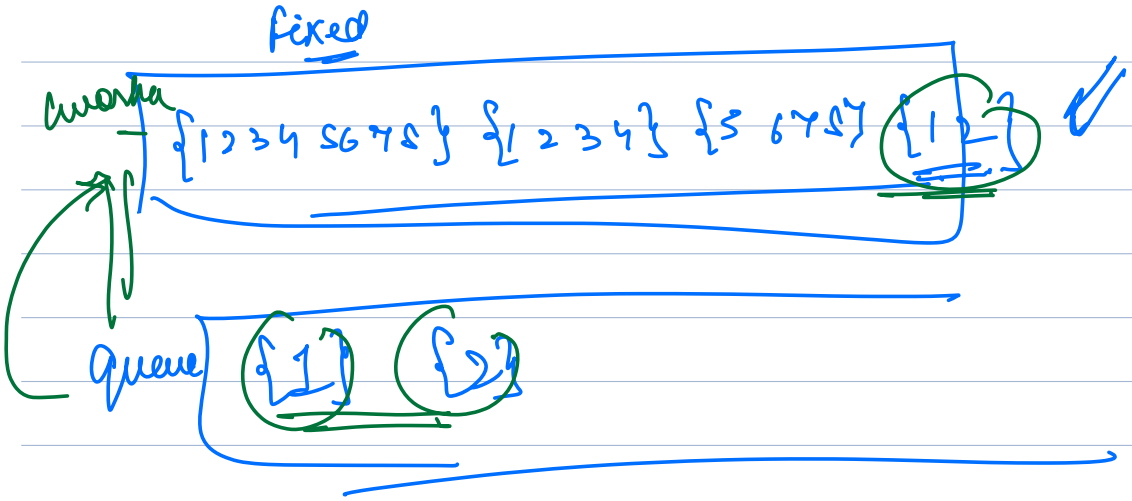
List<Int> sortedLeft = executor.submit(new Sorter(leftArray))

executor.submit(new Sorter(rightArray))

Feature

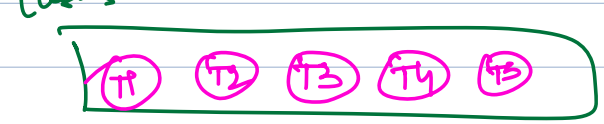


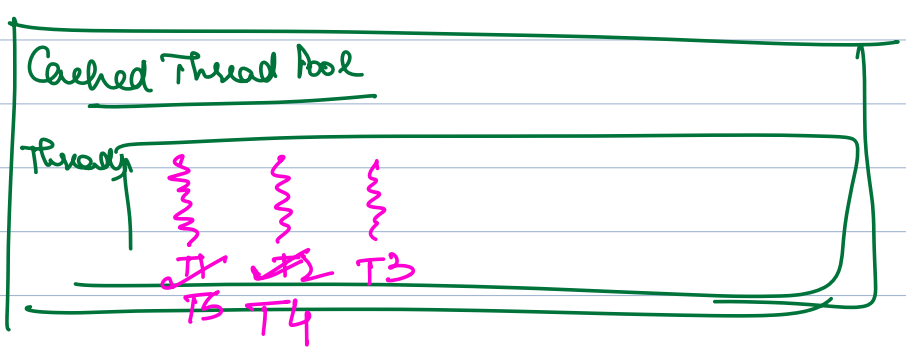

main
4261 9342
2^3
4261 9342
1
4261
9342
2
4
4 2
6 1
9 3
4 2
4
2
6
1
4 3
2
t = 8
2|4
1|6
3|9
2|4
1|2|4|6
2|3|4|9
1|2|2|3|4|6|9



Cached Thread Pool

whenever I get a new task:
    if I have a thread that is free:
        assign task to that thread
    else:
        create a new thread

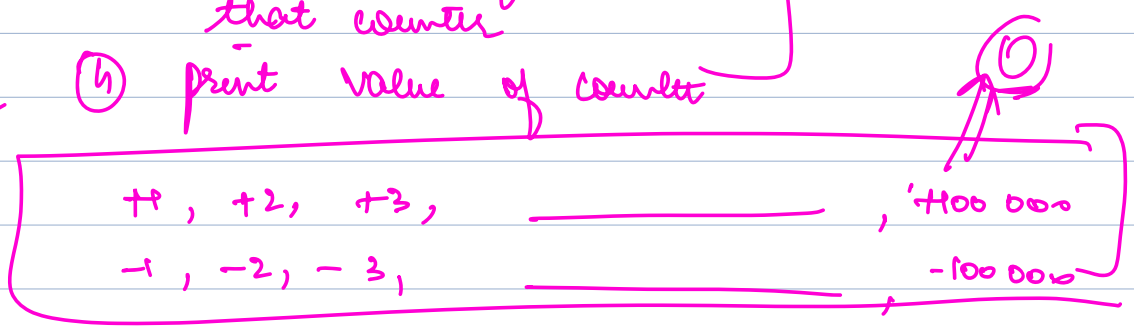




Assgn

1. Shared Counter
2. 1 thread that adds 1 to 100,000 to that counter
3. 1 thread that subtracts 1 to 1000000 from that counter
4. print value of counter

Counter {
value

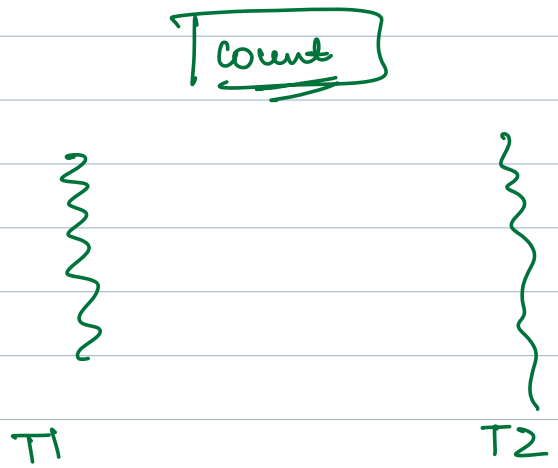


Task 1 Adding 1 to 100000 → Adder

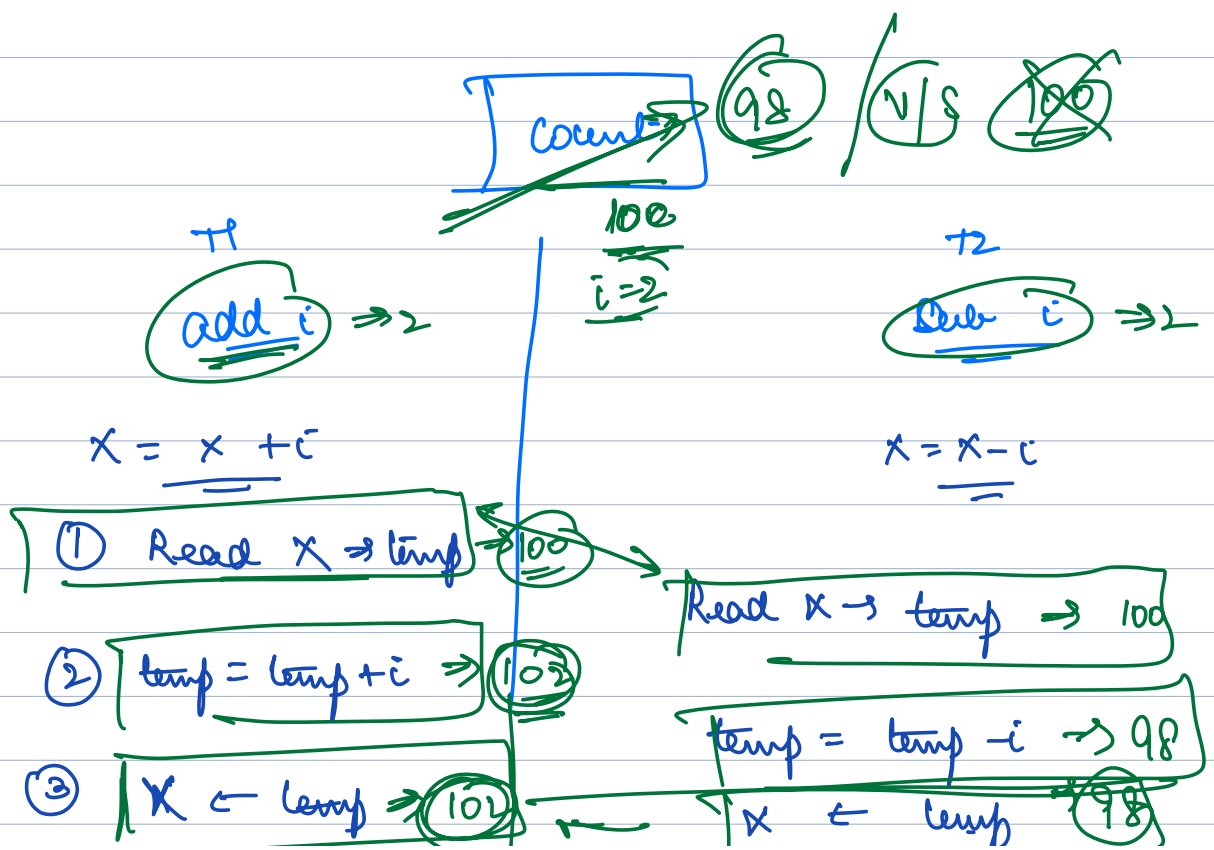
Task 2 Subtracting 1 to 100000 → Subtracter

Constructor

Count



Synchronization Problem: When more than one thread is working on same data at same time and may cause wrong values because of that



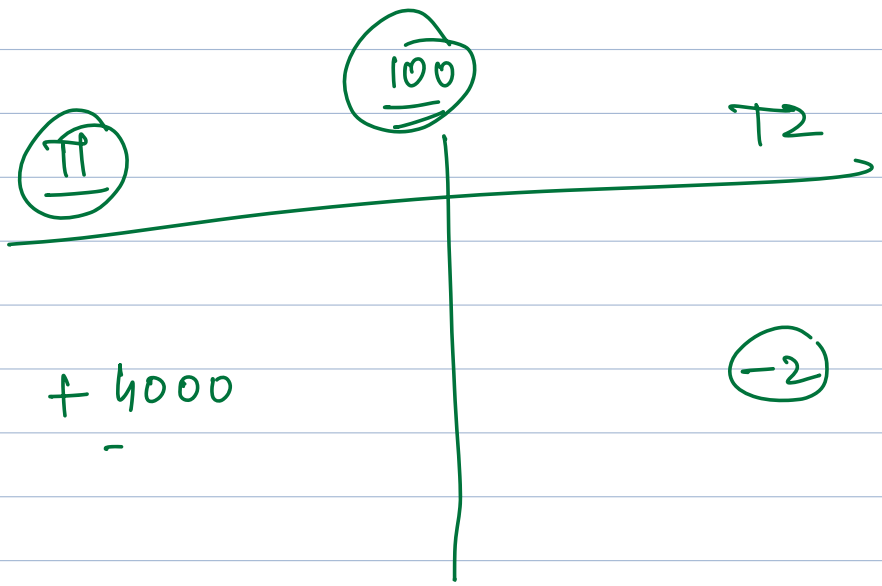
⇒ because 2 threads were sharing same data and modifying at same time

+1
+2
+3

−1
−2
−3
−4

+10000

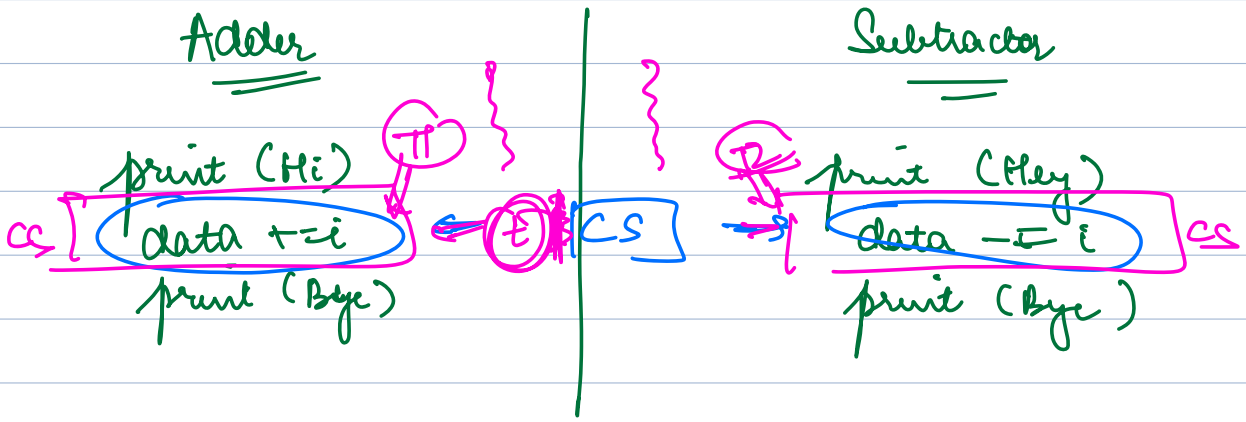
−10000



When do data sync problems happen

① Critical Section — in both threading

Part of code where data is being shared



Square

```
// take lock
data += 2
i = getIC()
dat ^= i
// release lock
```

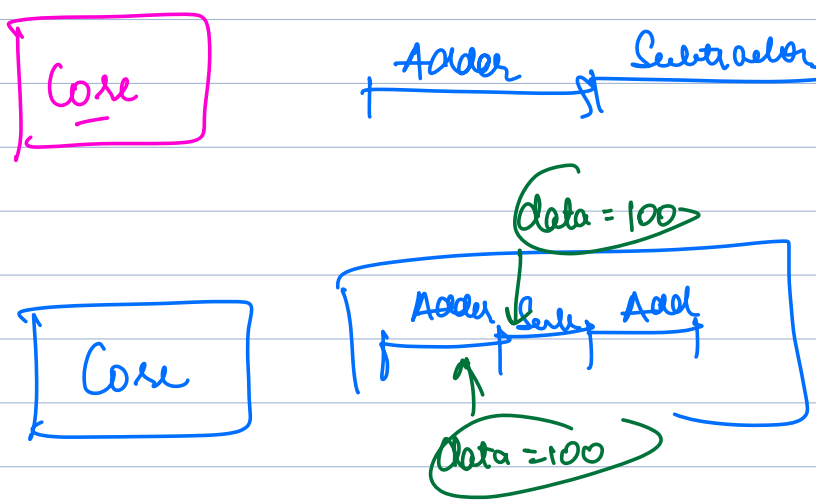
② Race Condition

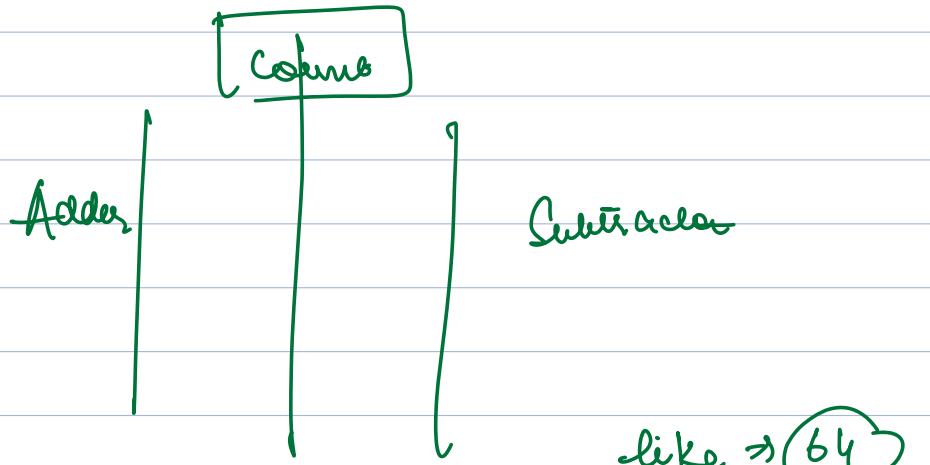
→ More than 1 thread is competing with each other.

→ More than 1 thread in CS at the same time.

③ Preemption (Single Core)

When CPU can stop a process in b/w to run another process





like ⇒ 64
63

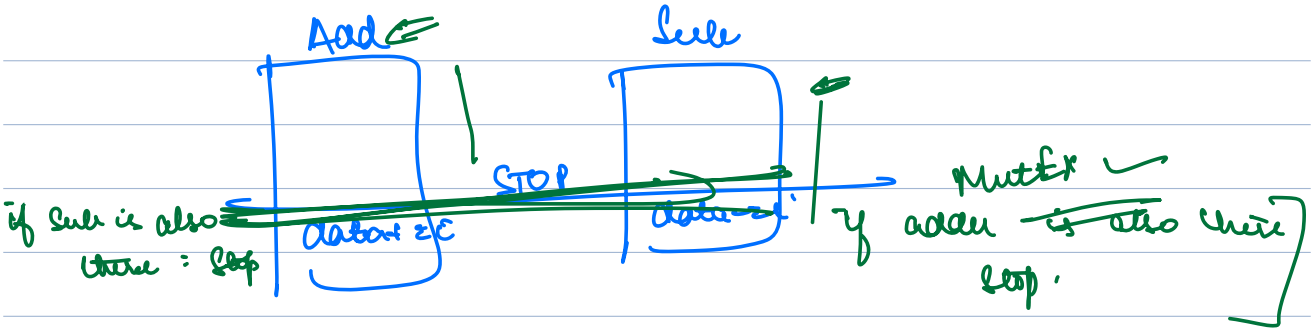
What a good soln of sync should be

① Mutual Exclusion (MutEx)

⇒ Only one thread should be there in CS at one time

② Progress

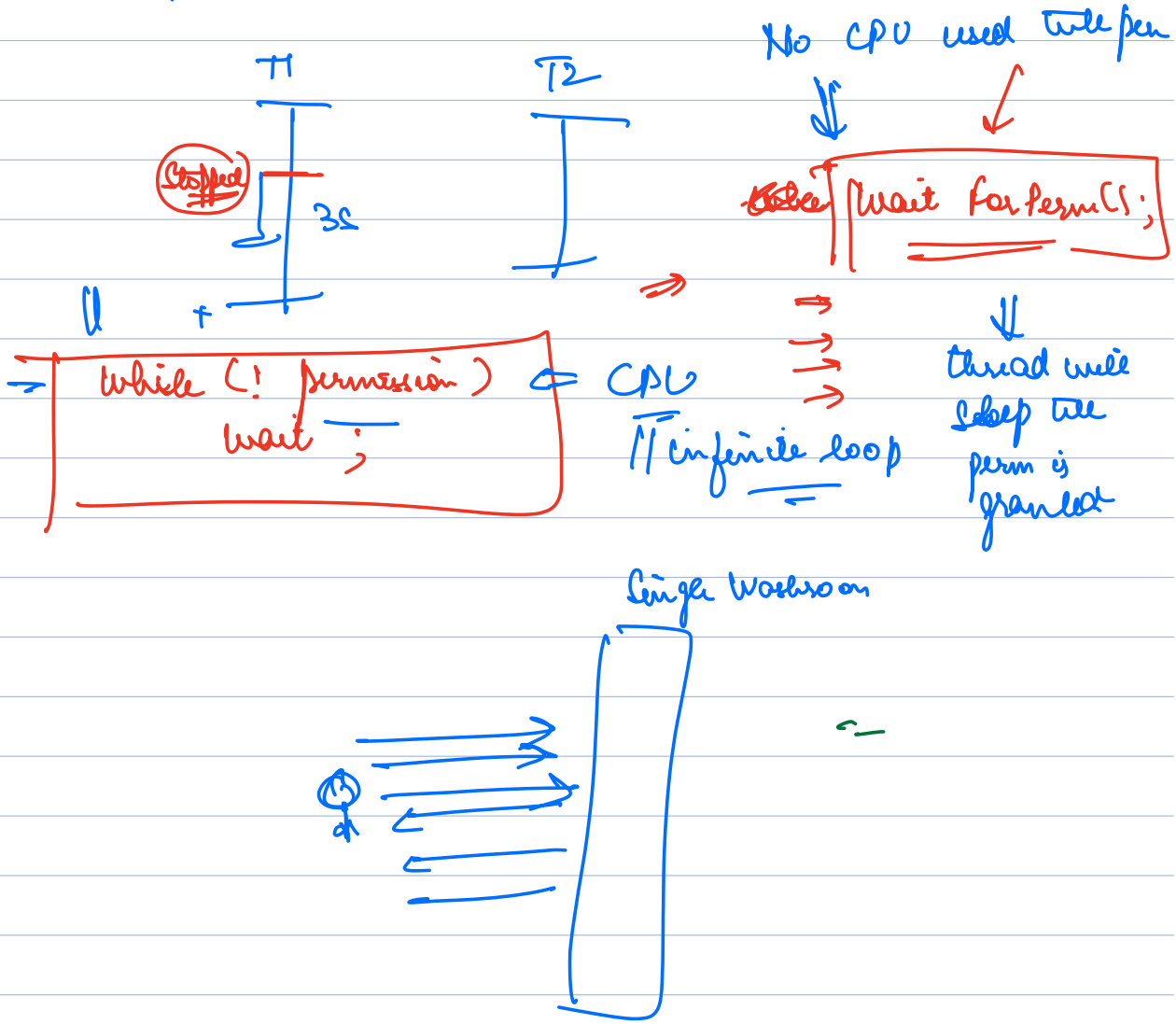
⇒ Overall system should keep on working



(3) Bounded Waiting

→ No thread will have to wait infinitely

(4) No Busy waiting Imp



Mutex | Semaphore

Good Solⁿ

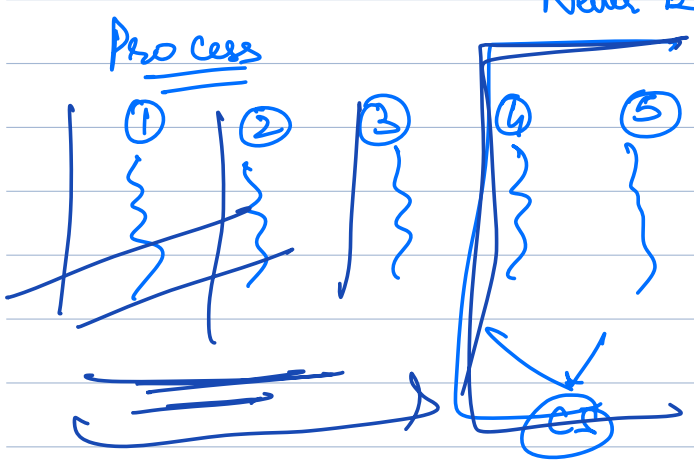
1. MutEx
2. Bounded Waiting
3. Progress
4. No Busy Waiting

Data Sync

→ CS

→ RC

→ Preemption




```
bool anyoneUsing = false;

while(anyoneUsing){
    wait()
}
anyoneUsing = true;
```