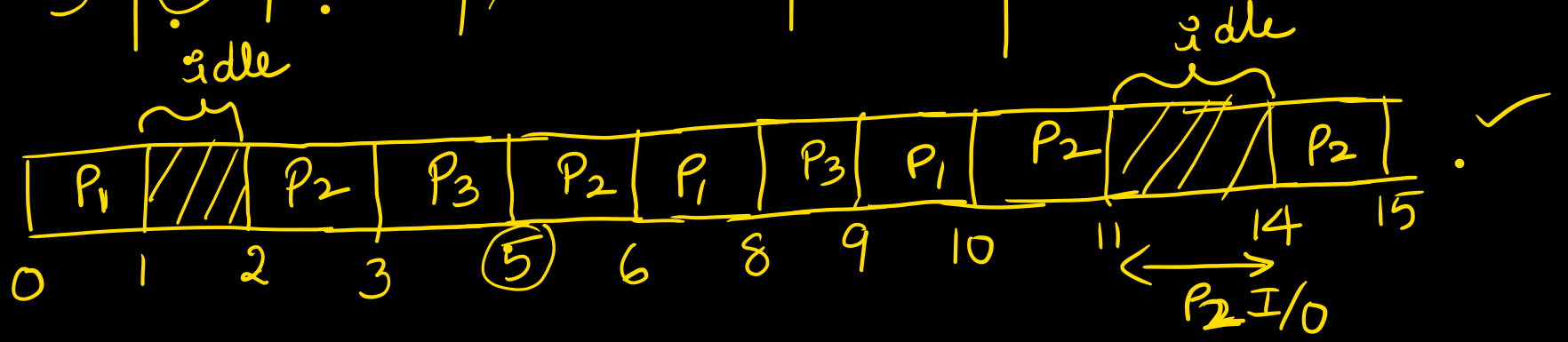
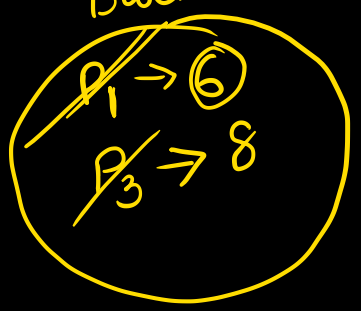
Preemptive priority: ✓

completion time of P_2 ? = 15

PNO	AT	PRI	BT	I/O	BT	TBT	CT	TAT	WT	RT
1 ✓	0	2	10	5	2	4 ✓	10	10	6	0
2	2 ✓	3 (L)	3	3	1	4	15	13	9	0
3	3	1 (H)	2	3	0	3	9	6	3	0



✓

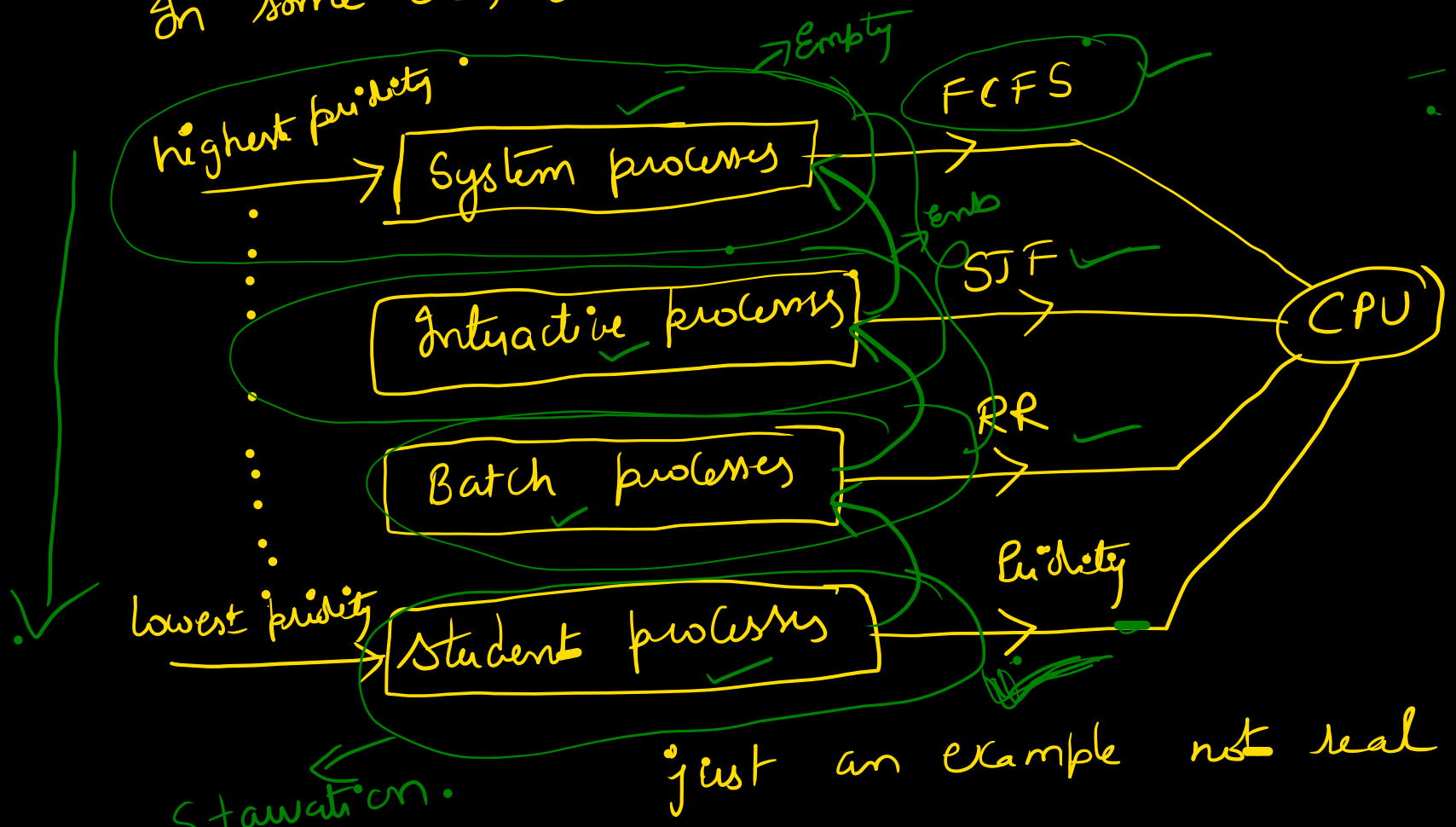


$$\text{CPU idle time} = \frac{4}{15} \times 100\%$$

$$\eta = \frac{11}{15} \times 100\ \%$$

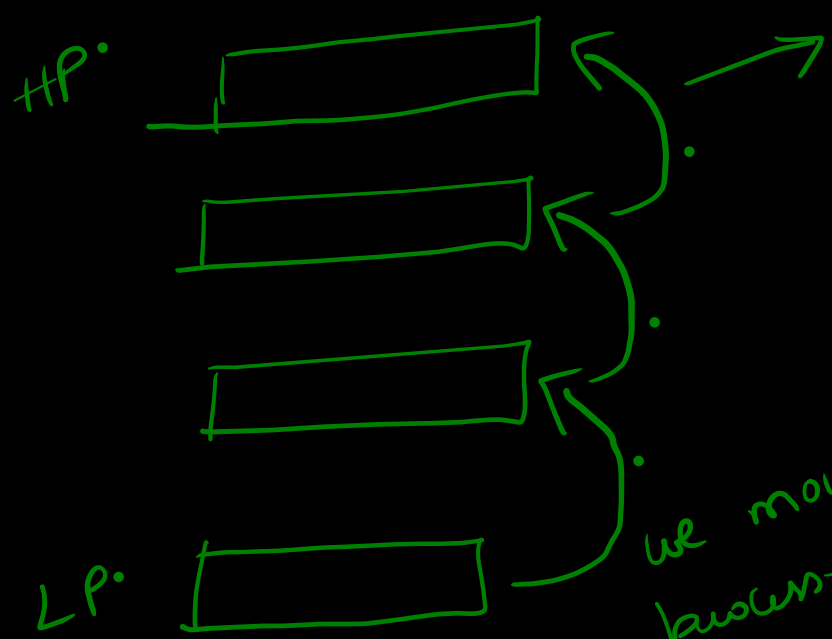
multi level queue scheduling:

In some OS, a MLQ is followed. Lets see one example



Example → not Real.

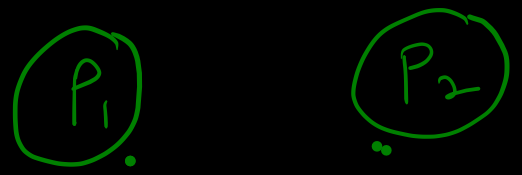
just an example not real



→ multilevel Queue scheduling with Feedback.

we move the processes from lower priority to higher priority Qs regularly.

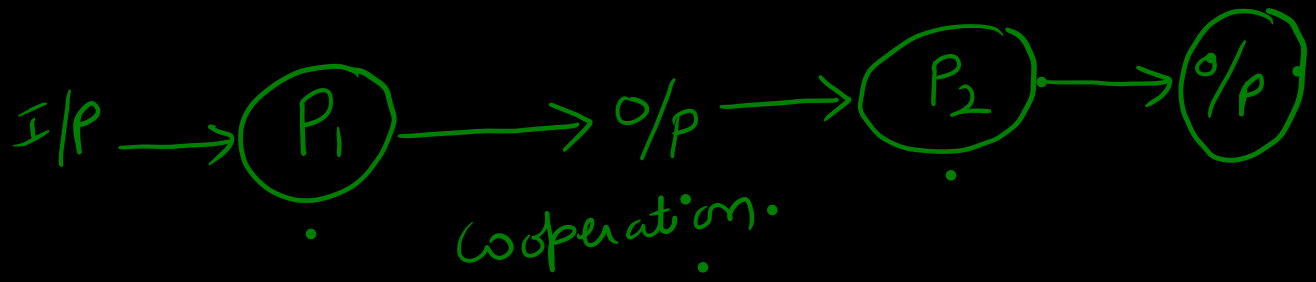
Process Synchronization:



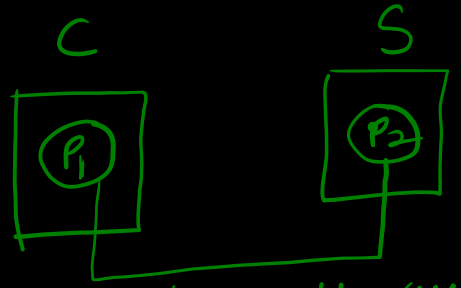
1) Cooperate and Complete a task

or

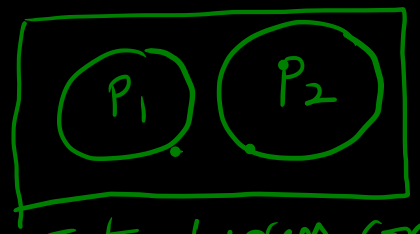
2) Compete with resources.



Cooperation.

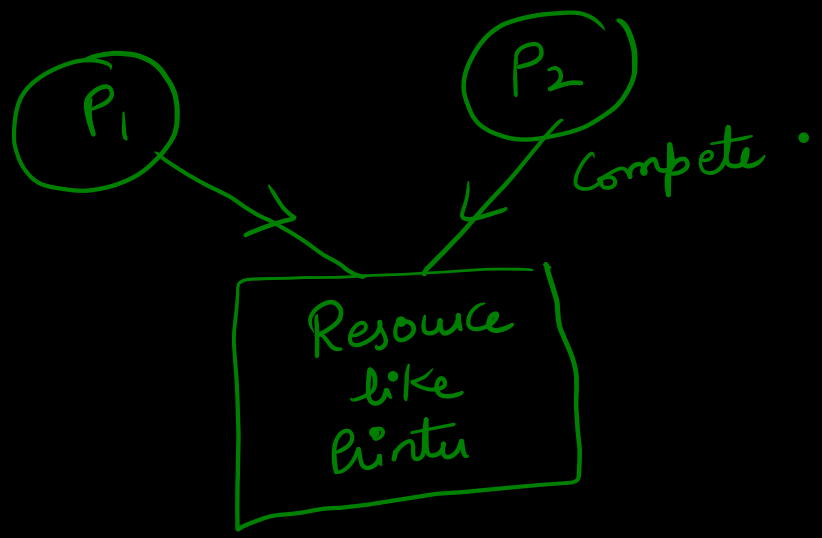


N/w software will help in C-S communication

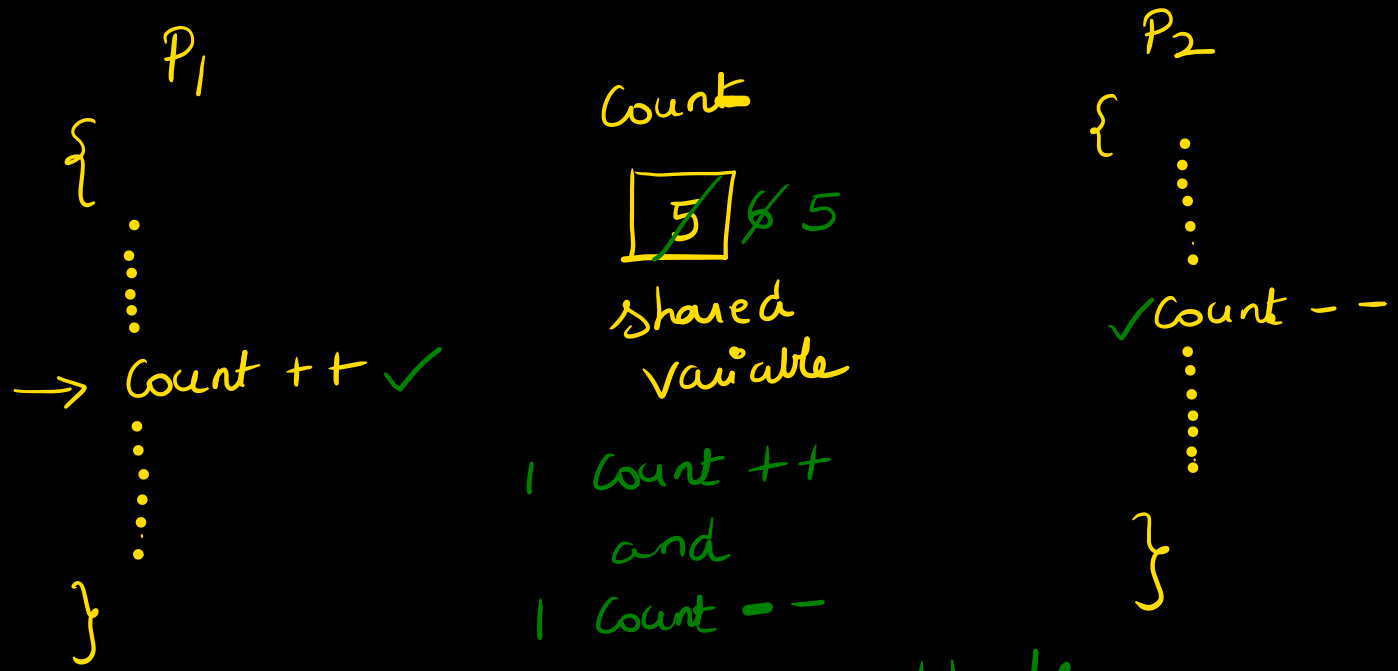


Inter process communication - OS ✓

Compete.



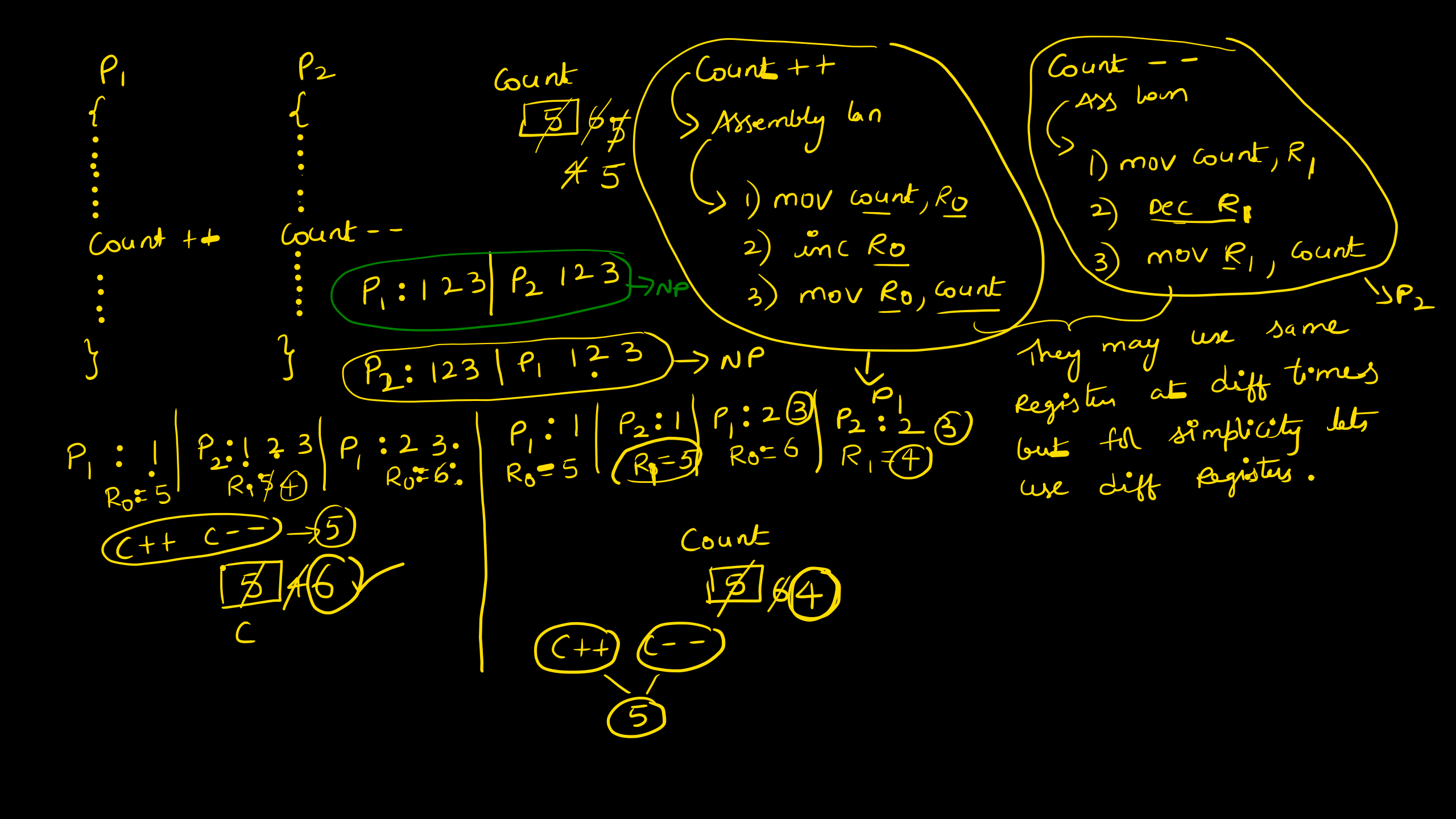
Both the cases we may get data inconsistencies.

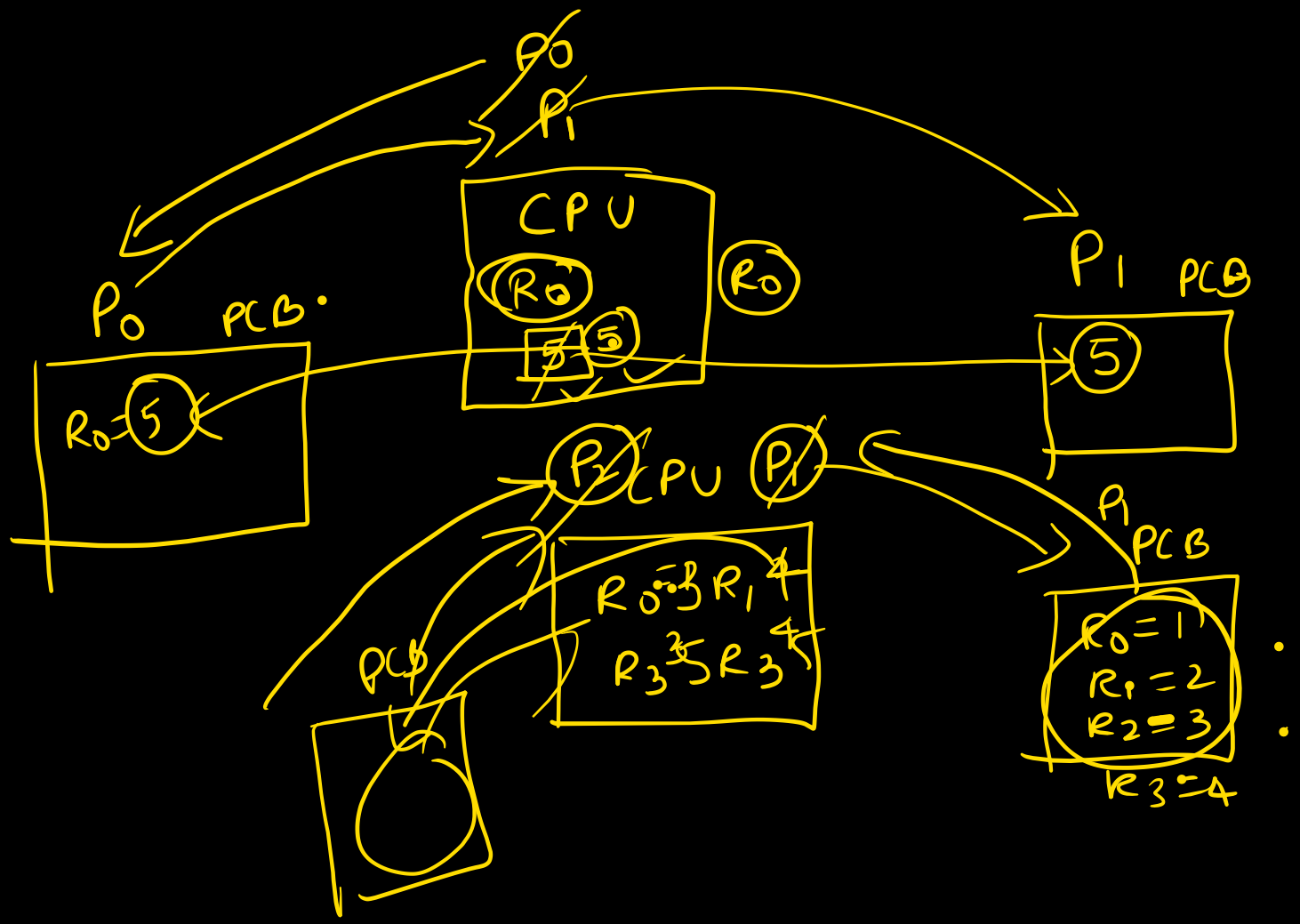


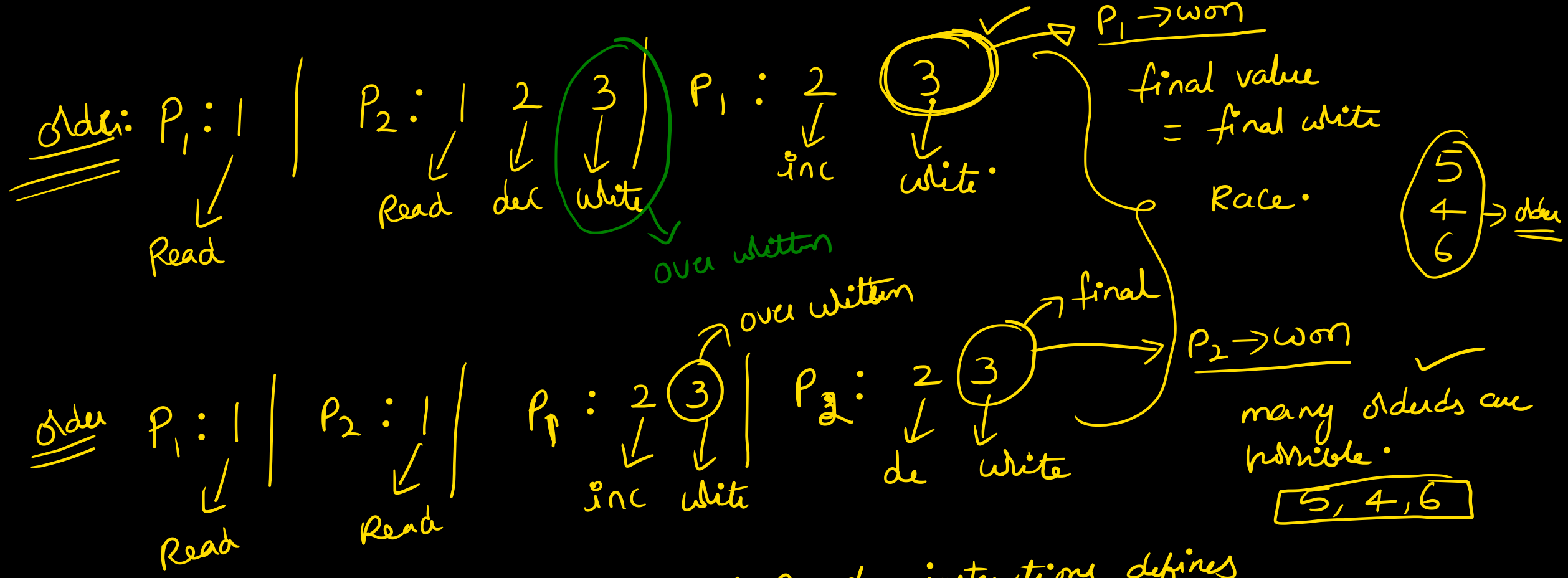
1 Count ++
and
1 Count --

Value should be same ✓

But be cause of Preemption, it may not be same.


P1
{
Count ++
}
P2
{
Count --
}
Count
5
6
4
5
Count ++
Assembly lan
1) mov count, R0
2) inc R0
3) mov R0, count
Count --
Ass lan
1) mov count, R1
2) Dec R1
3) mov R1, count
P2
P1 : 1 2 3 | P2 1 2 3 → NP
P2 : 1 2 3 | P1 1 2 3 → NP
P1 : 1 | P2 : 1 2 3 | P1 : 2 3 :
R0 = 5 | R1 = 5 | R0 = 6
C++ C-- → 5
5 6
C
P1 : 1 | P2 : 1 | P1 : 2 3 | P2 : 2 3
R0 = 5 | R1 = 5 | R0 = 6 | R1 = 4
Count
5 4
C++ C--
5
They may use same Register at diff times but for simplicity lets use diff Registers.


P0
P1
CPU
R0
P0 PCB
R0=5
5
5
P1 PCB
5
P2
CPU
P1
R0=3 R1
R3=5 R3
PCB
P1
PCB
R0=1
R1=2
R2=3
R3=4



✓ Race condition: The order of execution of instructions defines the result produced

Race condition causes data inconsistency.

conditions → Race Condition

1) Shared resources (like variables).

2) Preemption.

$P_1 | P_2$ & $P_2 | P_1$ → No problem.

$P_1 | P_2 | P_1 | P_2 \cdots$ → may be Race conditions

```
P1
{
1
2
3  Count++
4
5  Count --
6
7
8  Cout = 10
9
11 Count = 20
```

```
P2
1
2
3  Count--
4
5  Count++
6
7
8  Count = 40
9
10 Count = 50
```

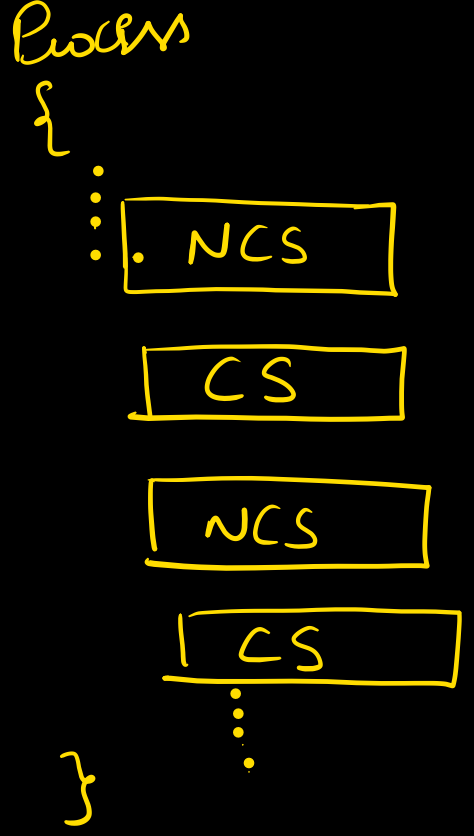
shared variable
→ Count

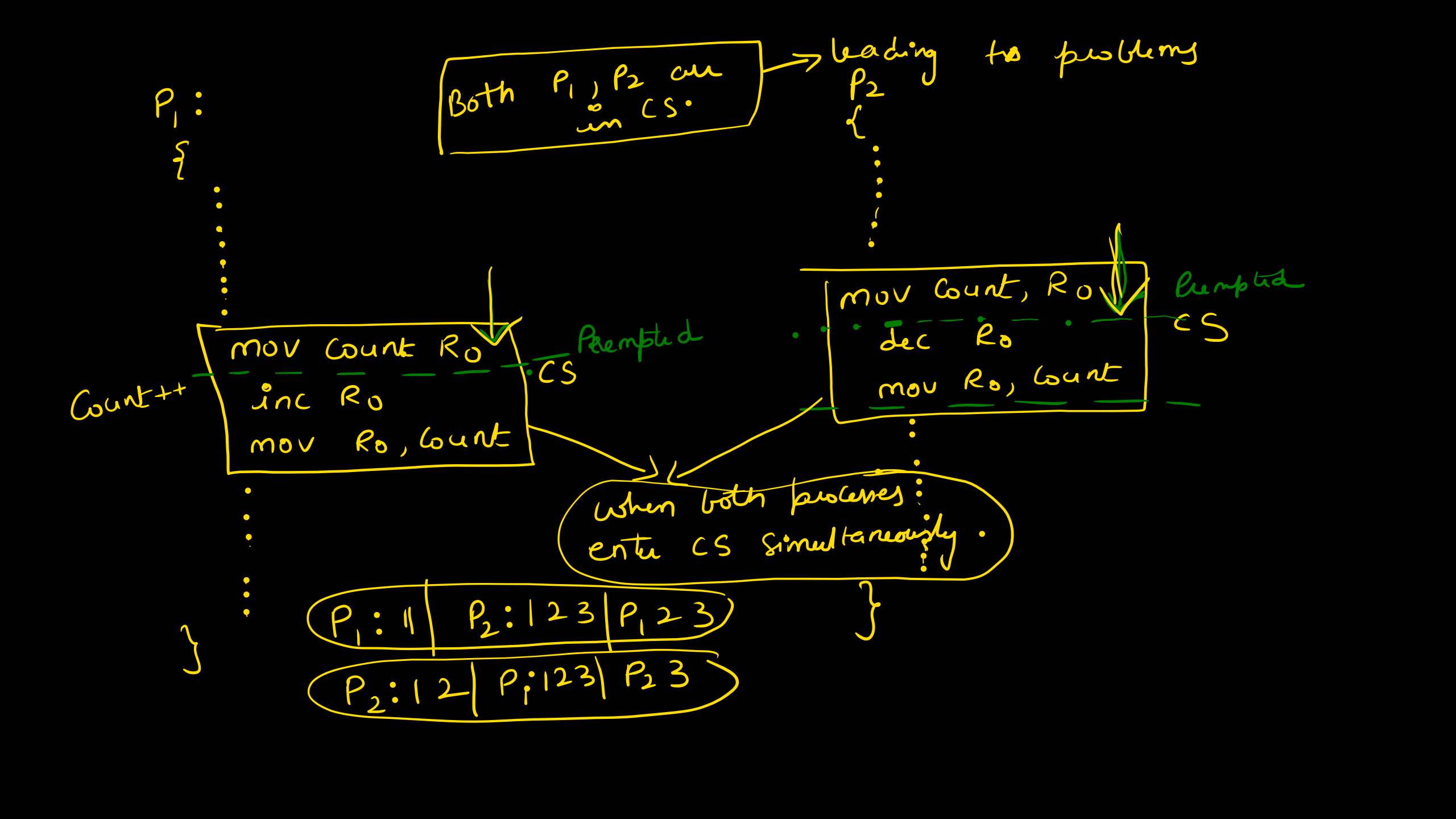
P_1: 3, 5, 8, 11
Critical sections
→ 4 CS

P_2: 3, 5, 8, 10
Critical section
→ 4 CS.

Critical section:

The part of the code ~~which~~ where shared resources are accessed is called Critical section.



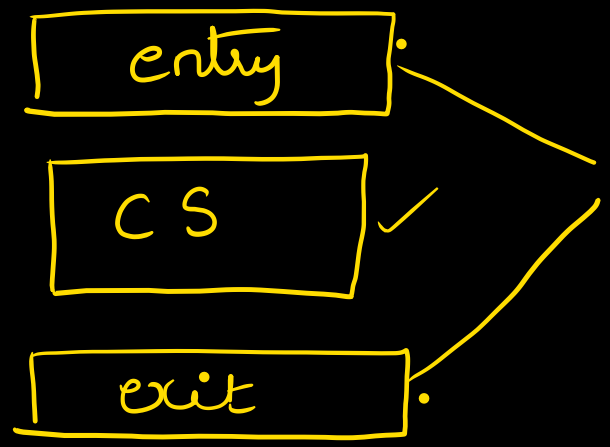

Both P1, P2 are in CS.
leading to problems
P1 :
{
P2
{
mov Count, R0
dec R0
mov R0, Count
Preempted
CS
Count++
mov Count R0
inc R0
mov R0, Count
Preempted
CS
when both processes enter CS simultaneously.
}
}
P1 : 1 1 | P2 : 1 2 3 | P1 2 3
P2 : 1 2 | P1 : 1 2 3 | P2 3

Race condition → when two processes enter CS at the same time.

- CS
- Preemption

Sol: we have to protect CS such that only one process enters it.

Process
{



we need to add code, such that only one process enters CS. This is called synchroniz
mechanism.

}

Requirements of synchronization mechanisms:

Primary:

1 → mutual exclusion ✓ → Compulsory.

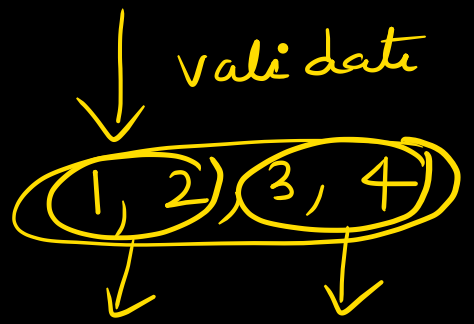
2 → Progress ✓

Secondary: → optional.

3 → Bounded waiting

4 → Portability & architectural neutrality.

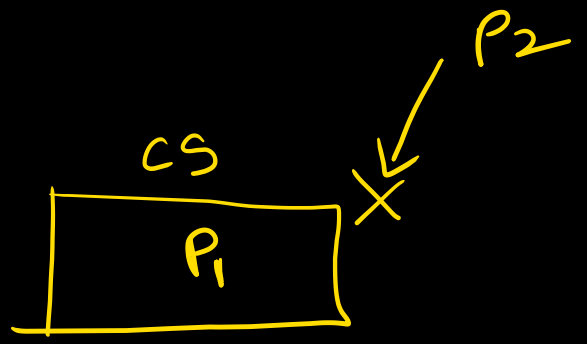
Shubam → new syn mech → Sub mech.

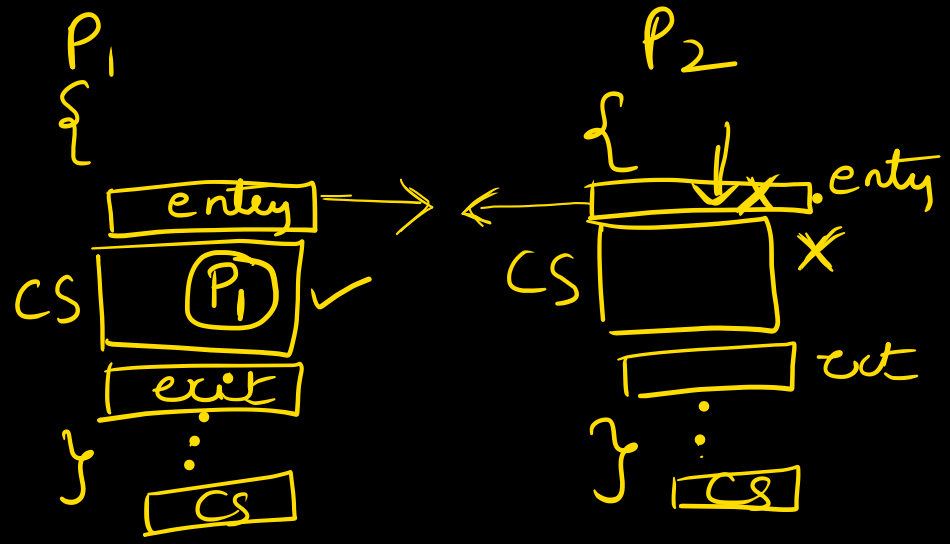


There are 100's of mechanism

But we will study most popular.

mutual exclusion:

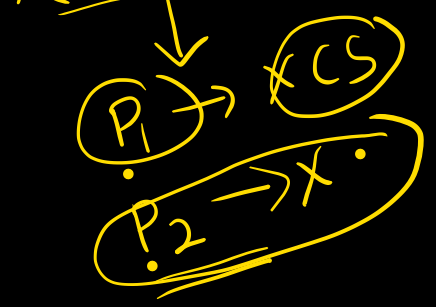


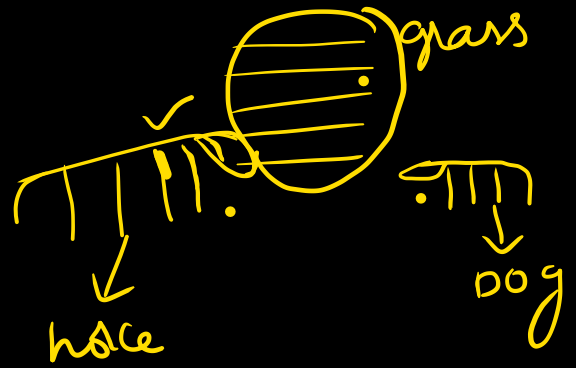


If one process is in CS, then other process should not enter into CS

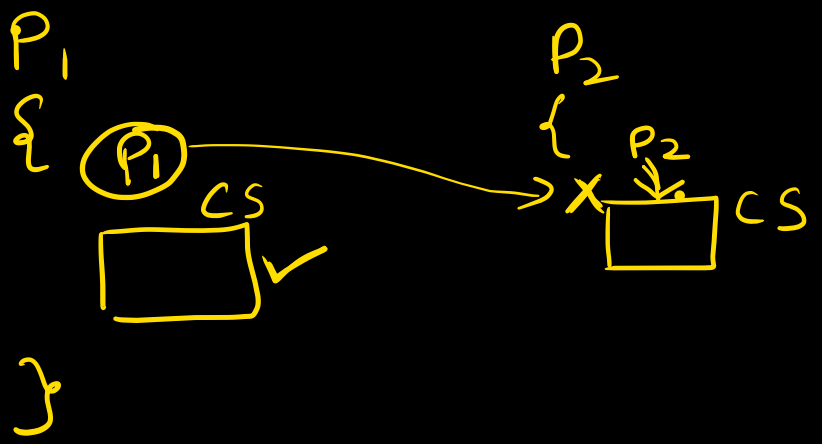
Progress: If a process does not want to get into critical section, then the process should not stop other process from getting into critical section.

Deep mech:





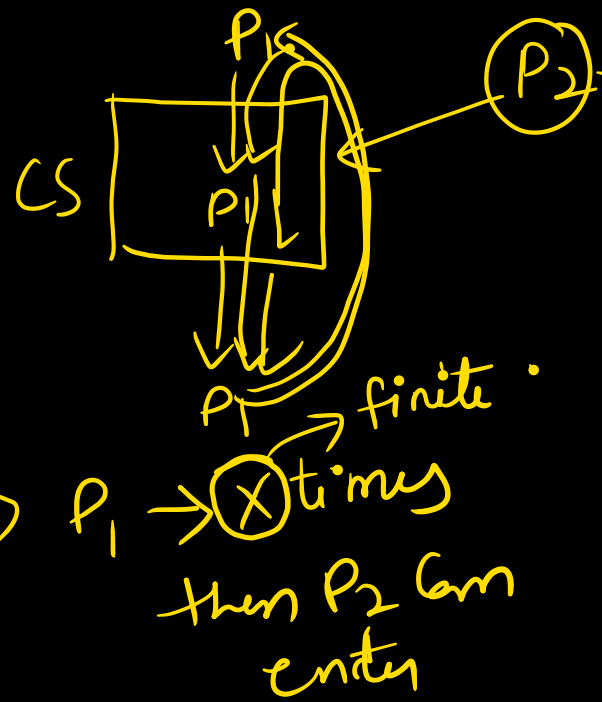
dog doesn't want grass. It should not stop horse from eating



Bounded waiting:

If a process wants CS, it should not wait forever.

There should be a finite bound on waiting time.

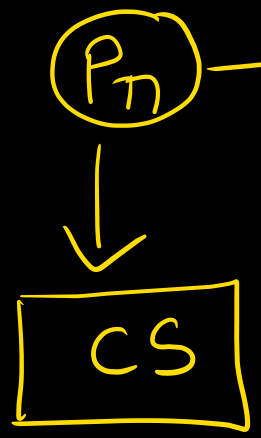


→ forever then there is not bounded wait.

$P_1\ P_2\ P_1\ P_2\ P_1 \ldots$

$P_1\ P_1\ P_2\ P_1\ P_1\ P_2\ P_1\ P_1\ P_2 \ldots$

$P_1\ P_1\ P_1\ P_2\ P_1\ P_1\ P_1\ P_2 \ldots$



→ should not wait ∞ time.

we should say after how long P_n is get a chance.

P_1 P_2 P_1 P_2 ....

P_1 P_1 P_2 P_1 P_1 P_2 ...

P_1 P_1 P_1 P_2 P_1 P_1 P_1 P_2

P_2 → after 1 P_1

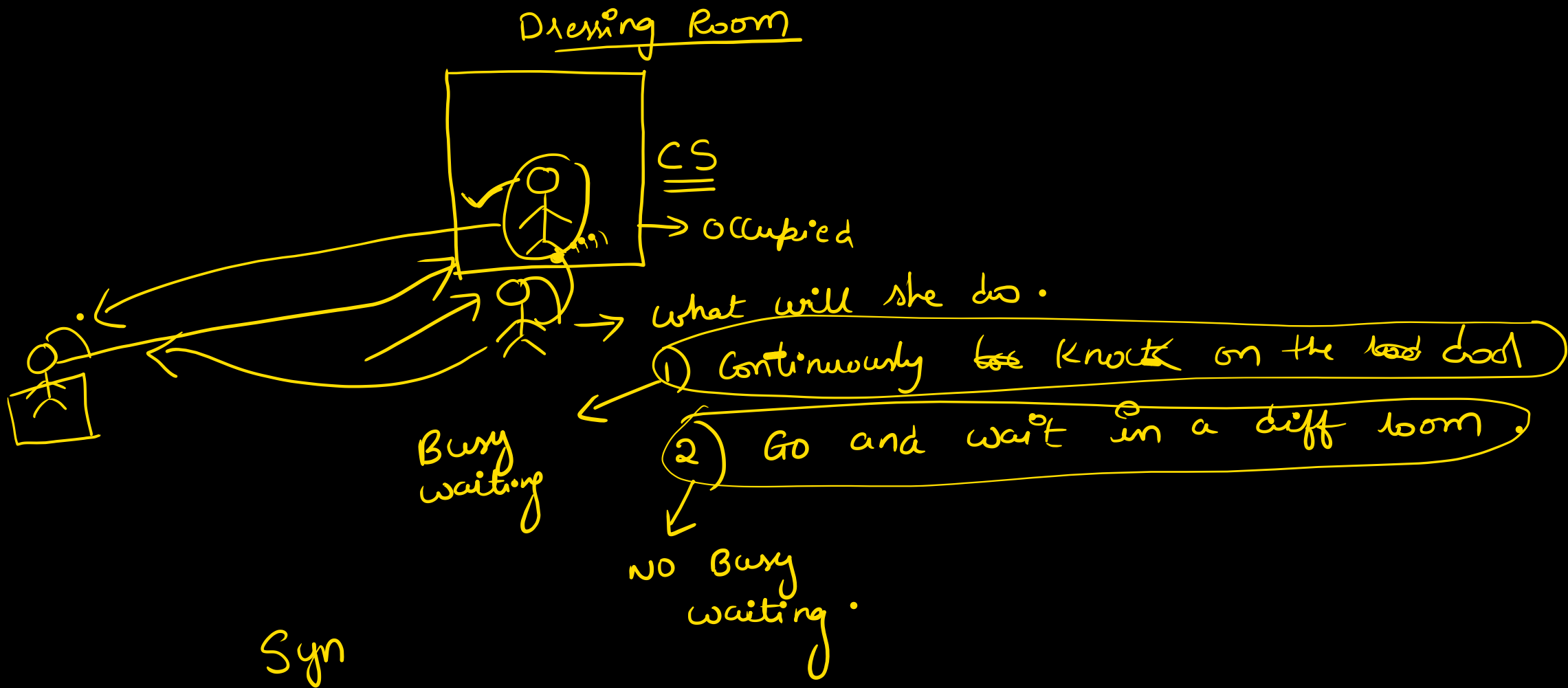
P_2 → after 2 P_1

P_2 → after 3 P_1.

Portability & architectural neutrality:

Synchr mech should work on all H/w and OS.

Ex: we should not say Sync mech works on only window not on linux. ⟶ ~~arch~~ architecture dependent.


Dressing Room
CS
occupied
what will she do.
1 Continuously knock on the door
Busy waiting
2 Go and wait in a diff room.
No Busy waiting.

Syn

Synchronization mech

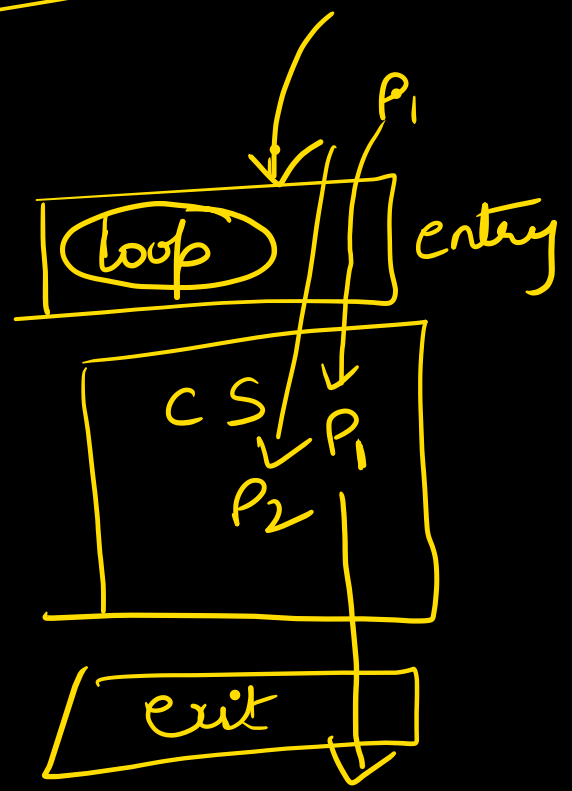
with busy waiting

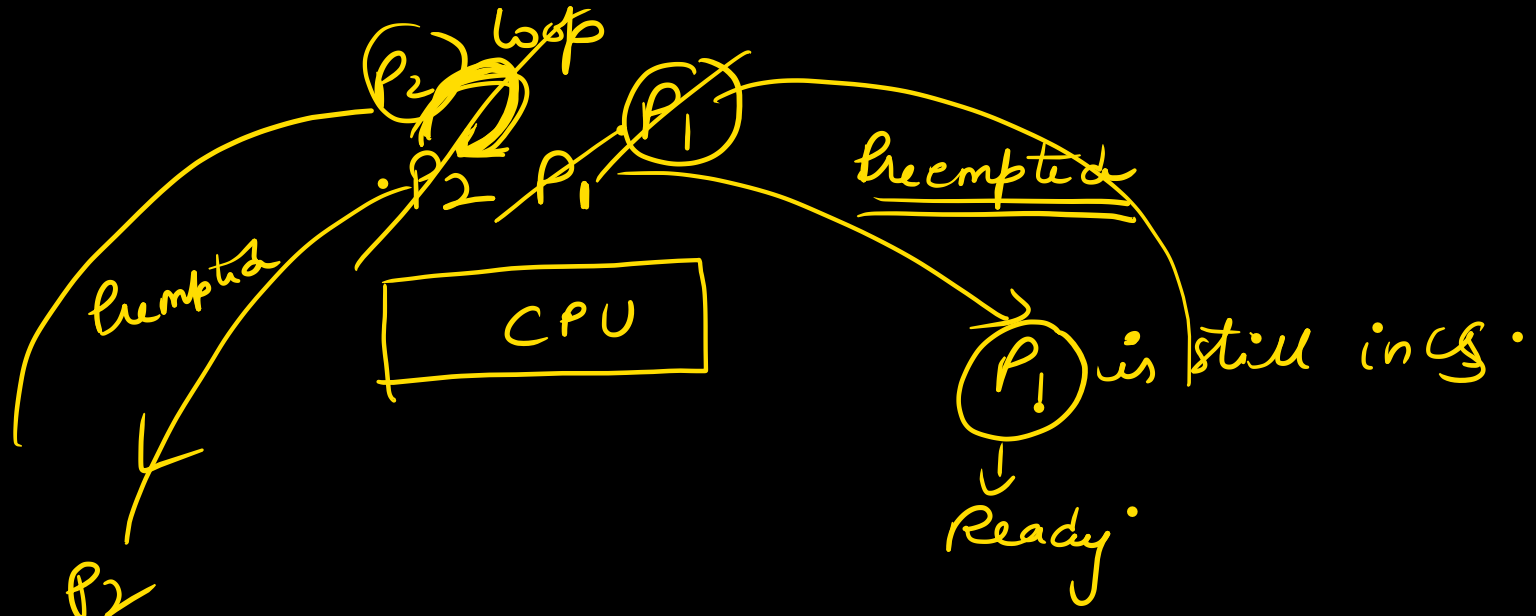
If one process is inside critical section, other process waits in a loop.

without busy waiting

If one process is in critical section, then other process which wants critical section will go to sleep. Later it will be woken up by the process coming out of critical section.

Busy waiting:





without busy waiting: