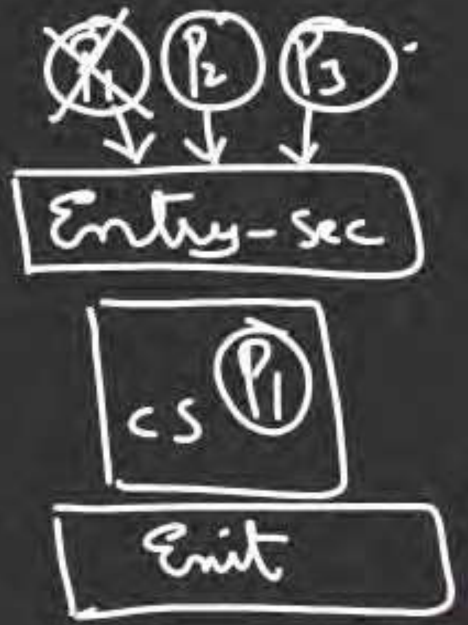
Session-II: Requirements of CS Problem/Synch. Mechanism

1) **Mutual Exclusion**: No TWO processes may be Simultaneously presen in their CS;

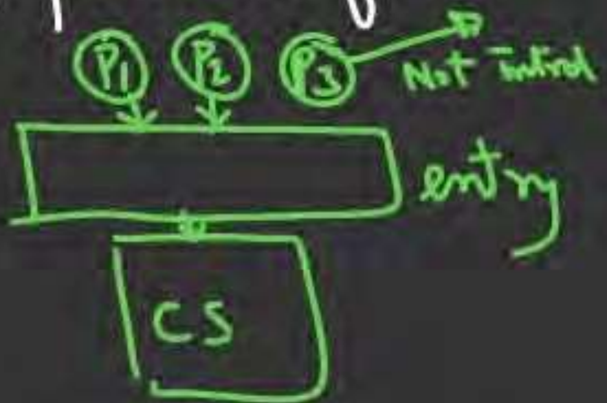
Primary-Requirement

Inconsisten

data loss



2) **Progress**: No process running out side (Non cs) the CS, Should block|Prevent|Influence the other Interested processes from entering CS;

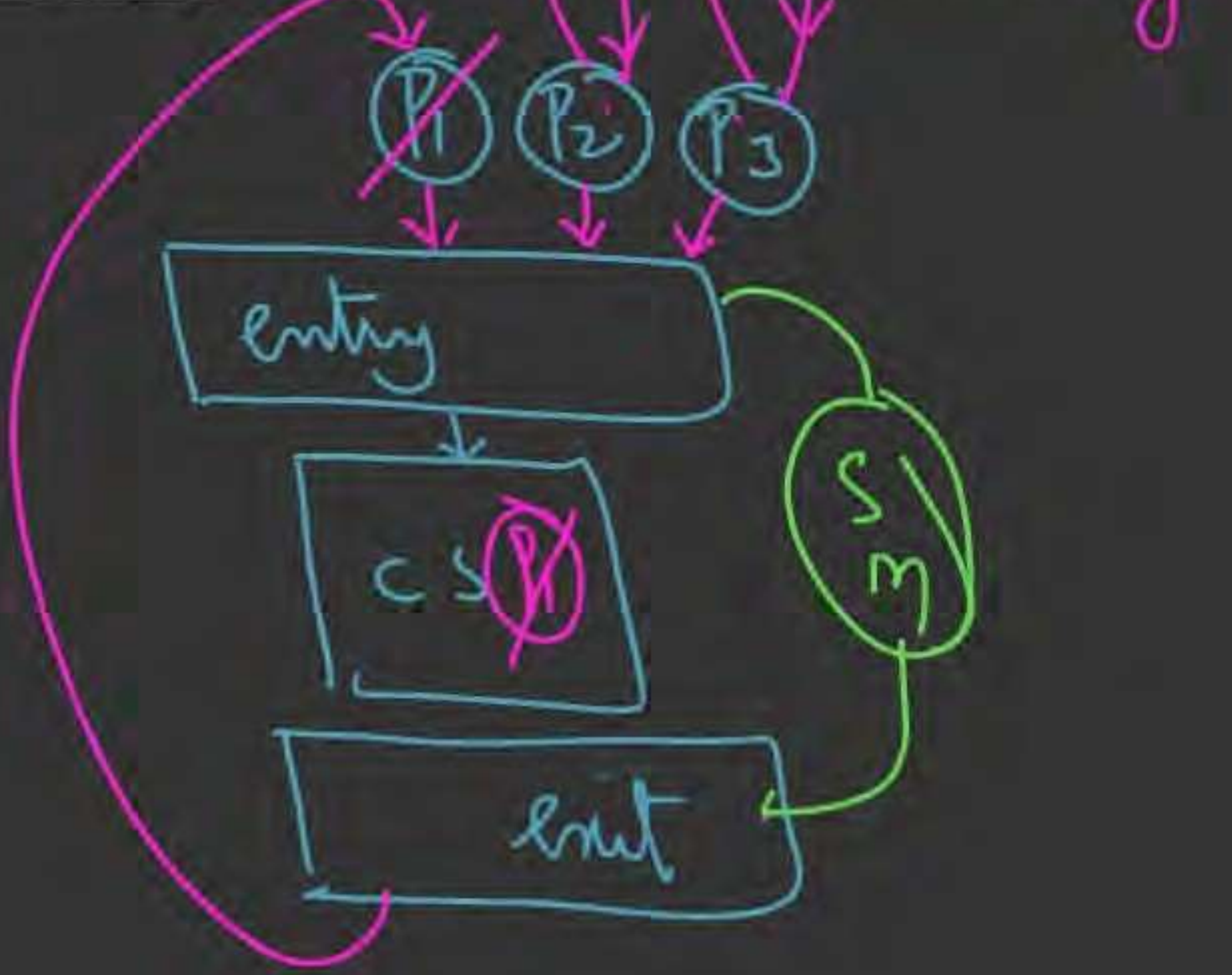


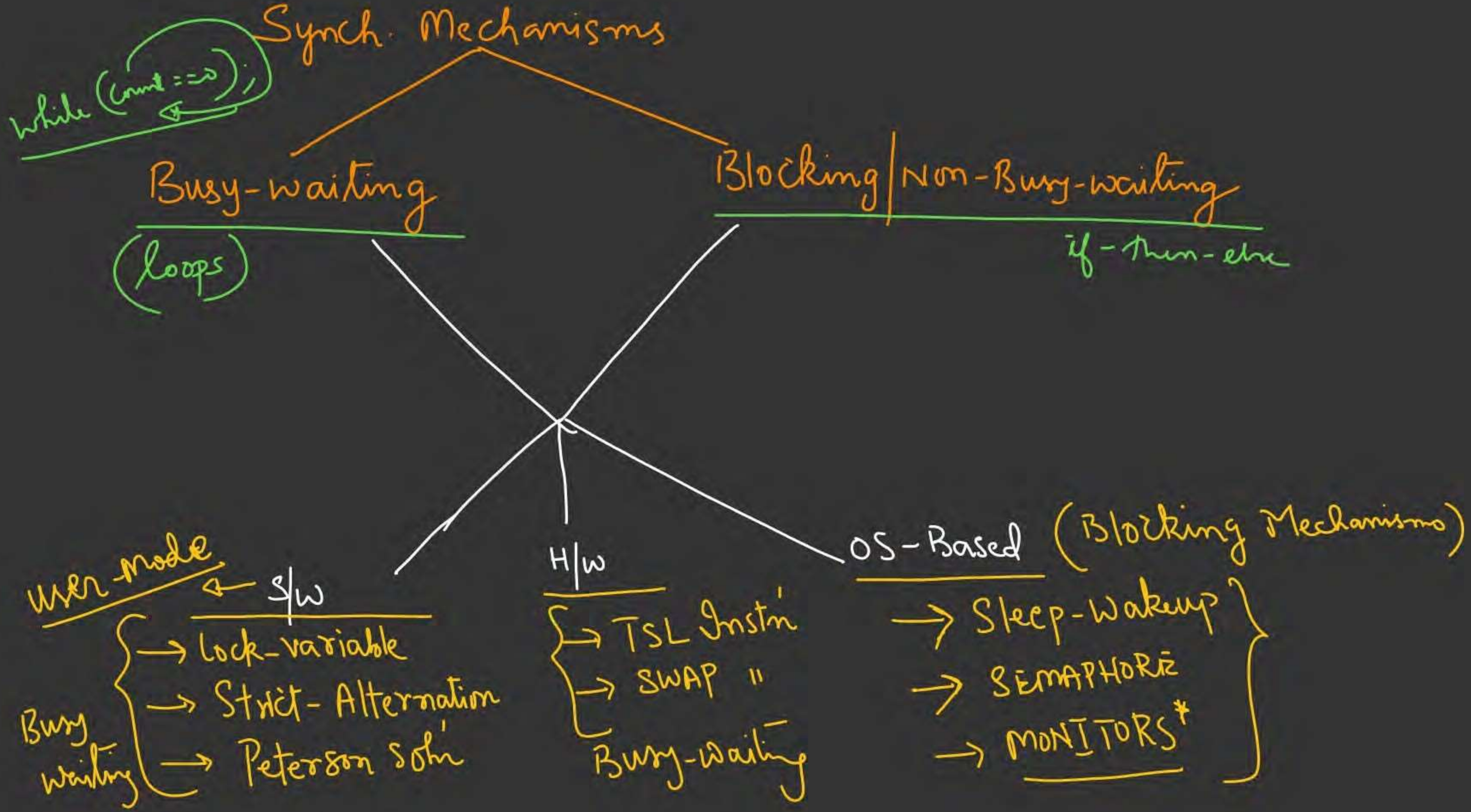
3) **Bounded waiting**: No process has to wait for ever to access "CS";

"There must be a bound on the no. of times that a process is allowed to enter CS, before other process request is satisfied"

dissatisf

STARVATION





Synch. Mechanisms

while (count == 0);

Busy-waiting (loops)

Blocking | Non-Busy-waiting

if-then-else

user-mode ← S/W
- → Lock-variable
- → Strict-Alternation
- → Peterson soln

Busy waiting

H/W
- → TSL Instn
- → SWAP "

Busy-waiting

OS-Based (Blocking Mechanisms)
- → Sleep-wakeup
- → SEMAPHORE
- → MONITORS*

Assumptions for S.Ms.

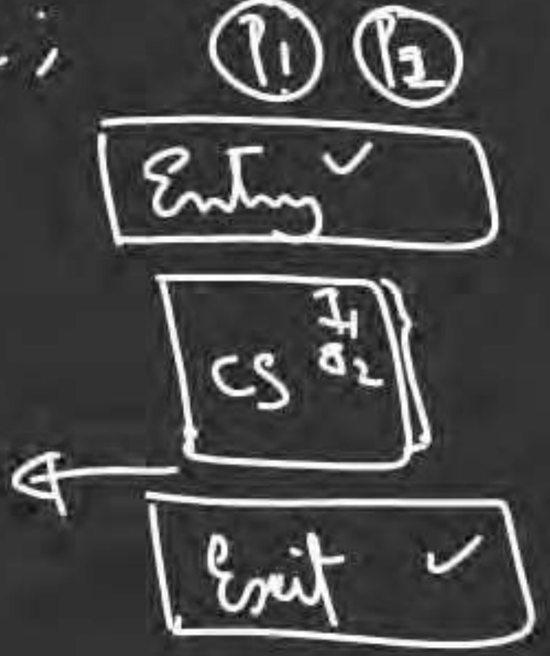
1) PreEmption of Process Can happen when it is executing in Entry, Exit or CS;

2) we assume CS is totally error-free;

3) Every Process enters CS & Comes out of it, in Finite Time;

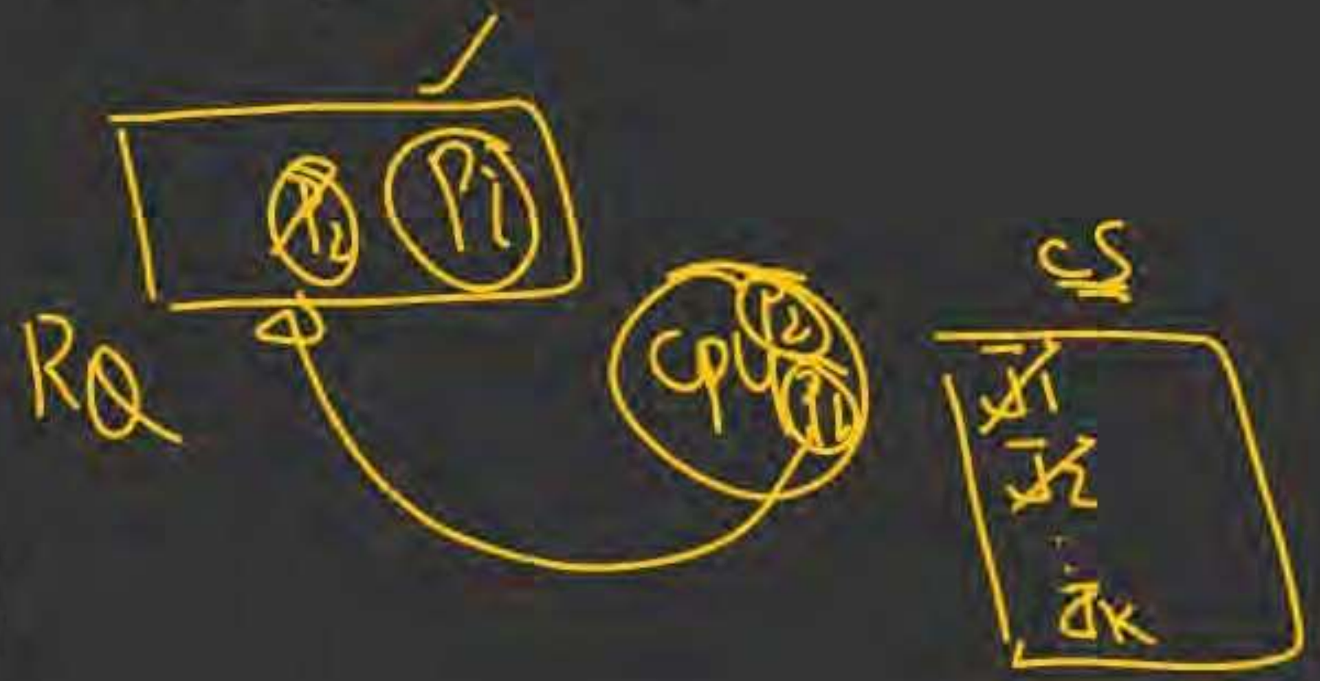
4) Process Can enter CS only after Completing entry Section

5) Process is said to have left "CS" only if it Completes exit Section;



✱

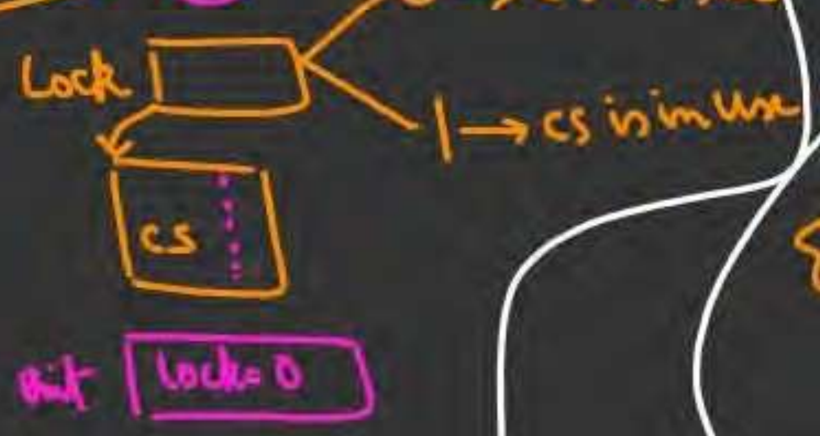
6) If a process gets PreEmpted from CPU while executing 'cs' Code, then still the process is said to hold "cs";



1) LOCK VARIABLE:

- → User Mode s/w Soln;
- → Busy-waiting; (loop)
- → Multi-process Soln;

Ideology: P_i



I. H.L.I: ✓

```
int Lock=0;
Void Process(int i)
{
    while(1)
    {
        a) Non_CS();                 P1
        b) while(lock!=0);   Entry
        c) lock=1
        d) <cs>
        e) lock=0;           Exit
    }
}
```

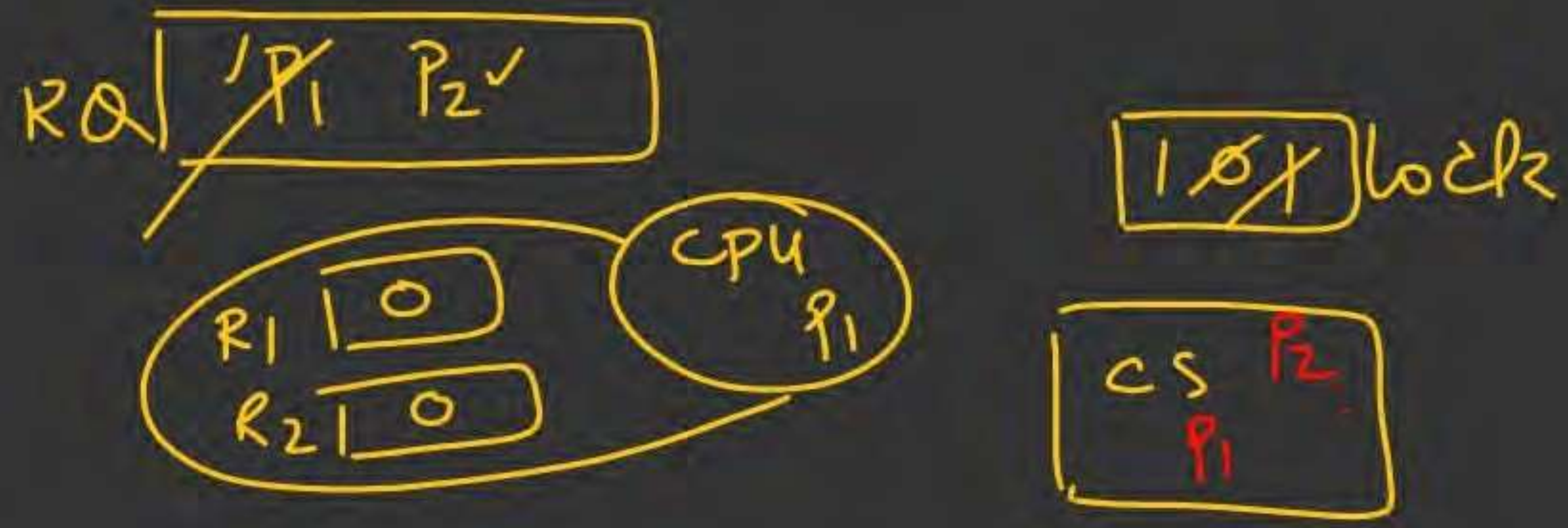
II. L.L.I: Process: ✓

JNZ: Jump If NOT 0

```
I.   Non_cs;
II.  Load Ri, Lock;      Entry
III. Cmp Ri, #0;
IV.  JNZ step II
V.   Store lock, #1
VI.  <cs>
VII. store lock, #0;
```

Bounded Waiting

Analysis: ME; Prog; B.W



t_1: P_1: I; II; III; IV; Pre

t_2: P_2: I; II; III; IV; V; VI; Pr

t_3: P_1: V; VI

"Violation of Mutual Exclusion"

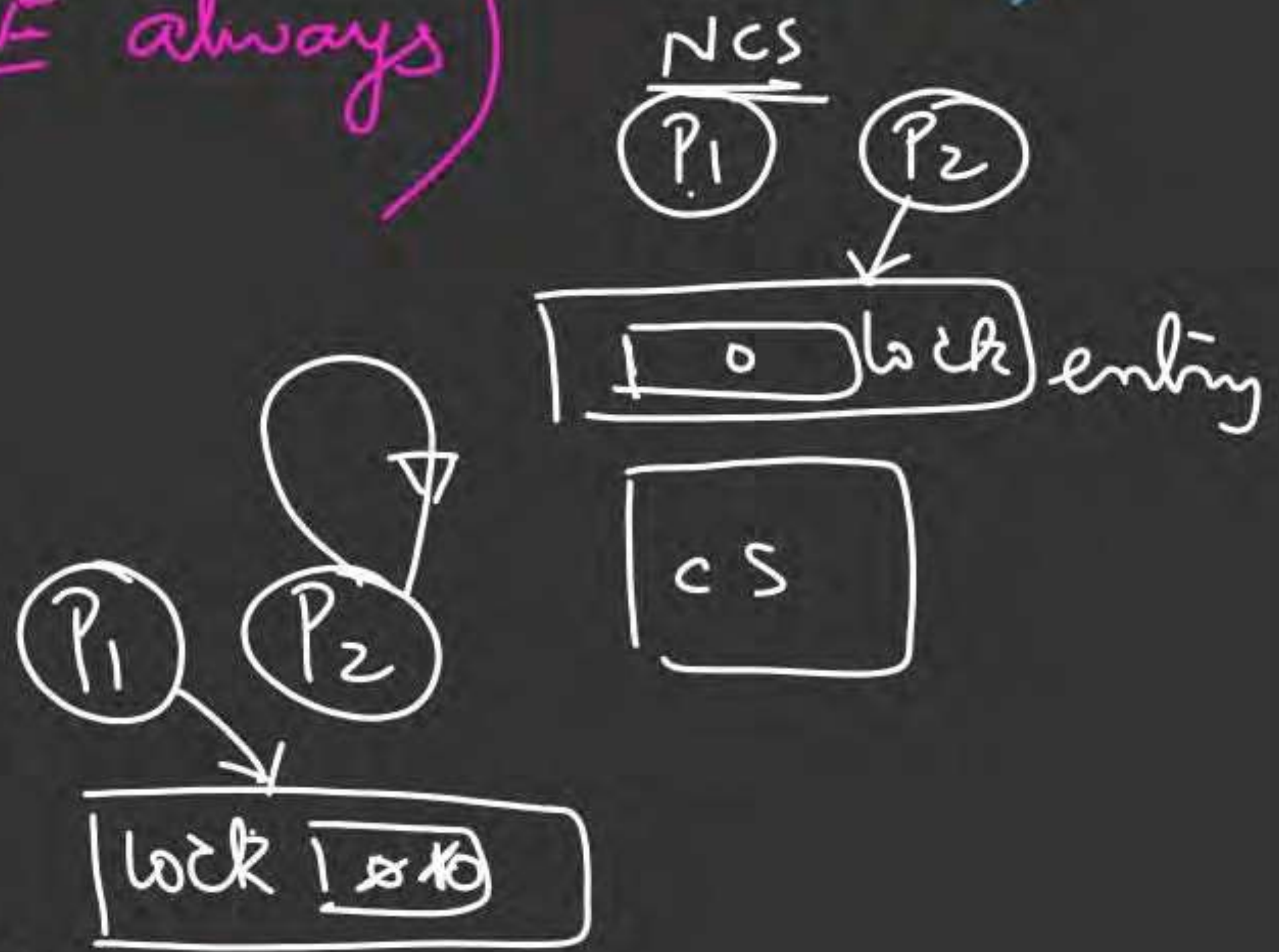
Progress

→ Lock variable Fails to guarantee Mutual Exclusion,

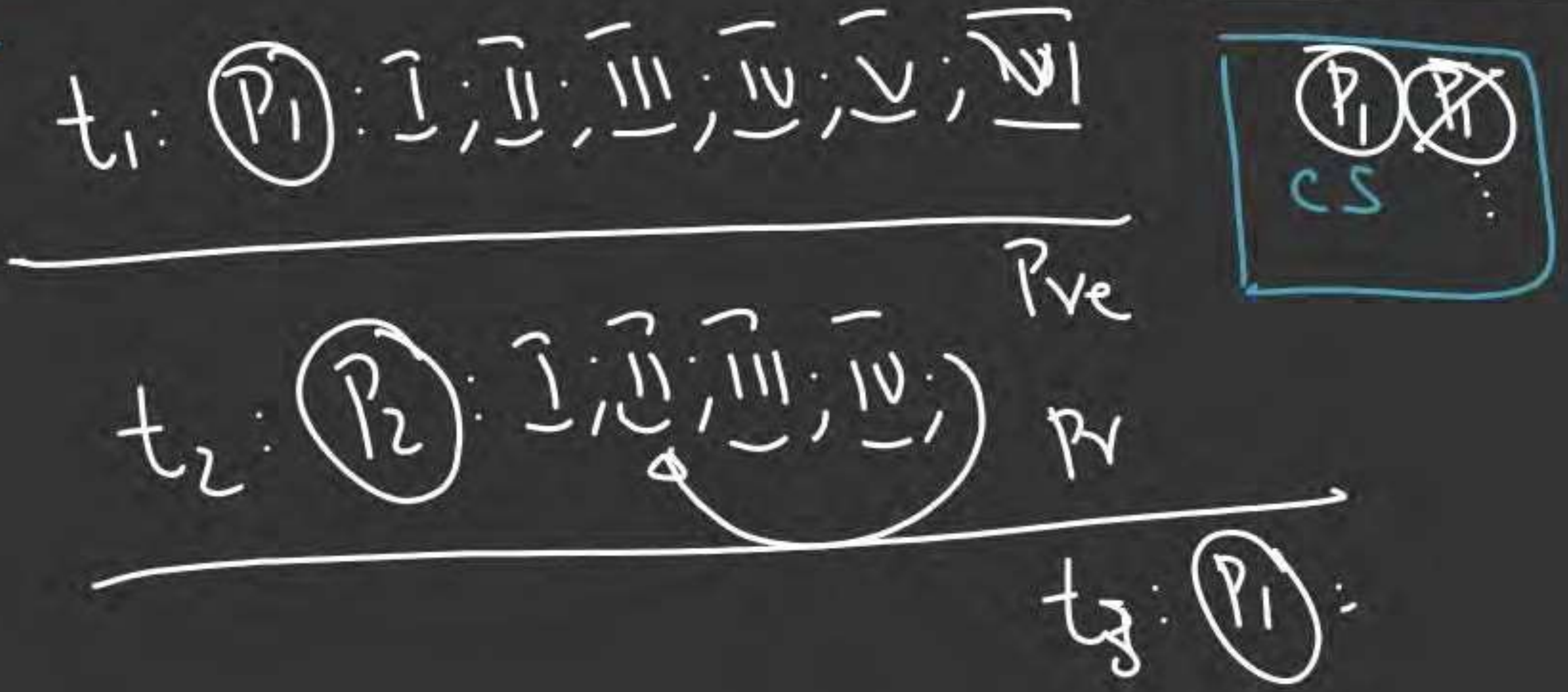
(Does not guarantee M/E always)

→ Since no process executing in NCS, will block other process from entering CS

∴ Progress is always guaranteed.



→ Does NOT guarantee Bounded wait,



II. STRICT ALTERNATION:

→ Busy-waiting

→ S/w Soln @ User Mode

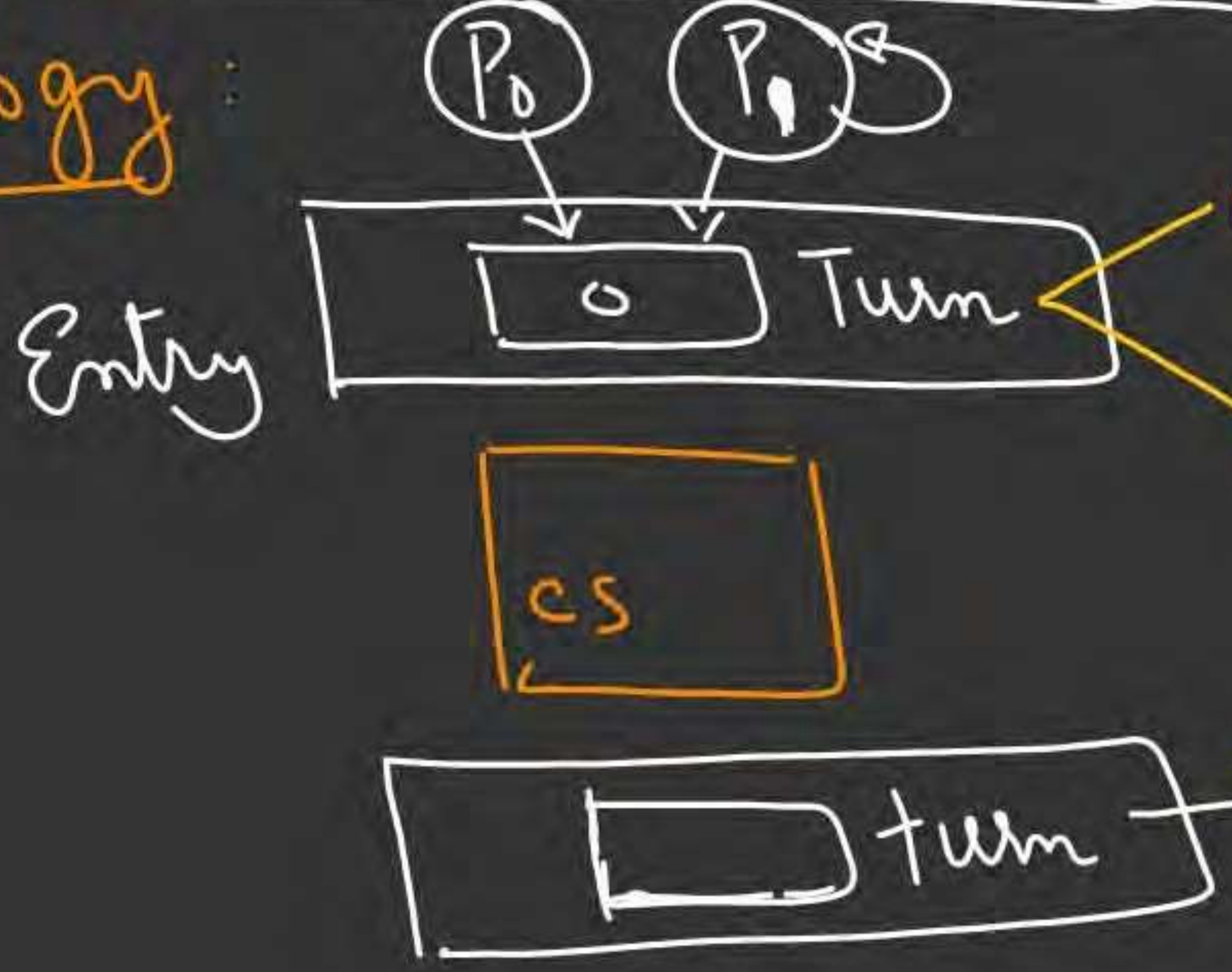
→ 2-process Soln (P_0; P_1)

<P_i | P_j>

→ The value of turn indicates, which process turn to enter CS

Strictly on alternate basis, processes takes turn to enter CS

Idealogy:



0 → P0's turn to enter CS

1 → P1's turn to enter CS

→ Make the value of turn to the id of other Process;