CS & IT ENGINEERING

Operating System

Process Synchronization

Lecture No. 4

By- Dr. Khaleel Khan Sir

Requirements of C.S Problem

TOPICS TO BE COVERED

- Lock Variable
- Strict Alternation
- Peterson Algorithm

Code

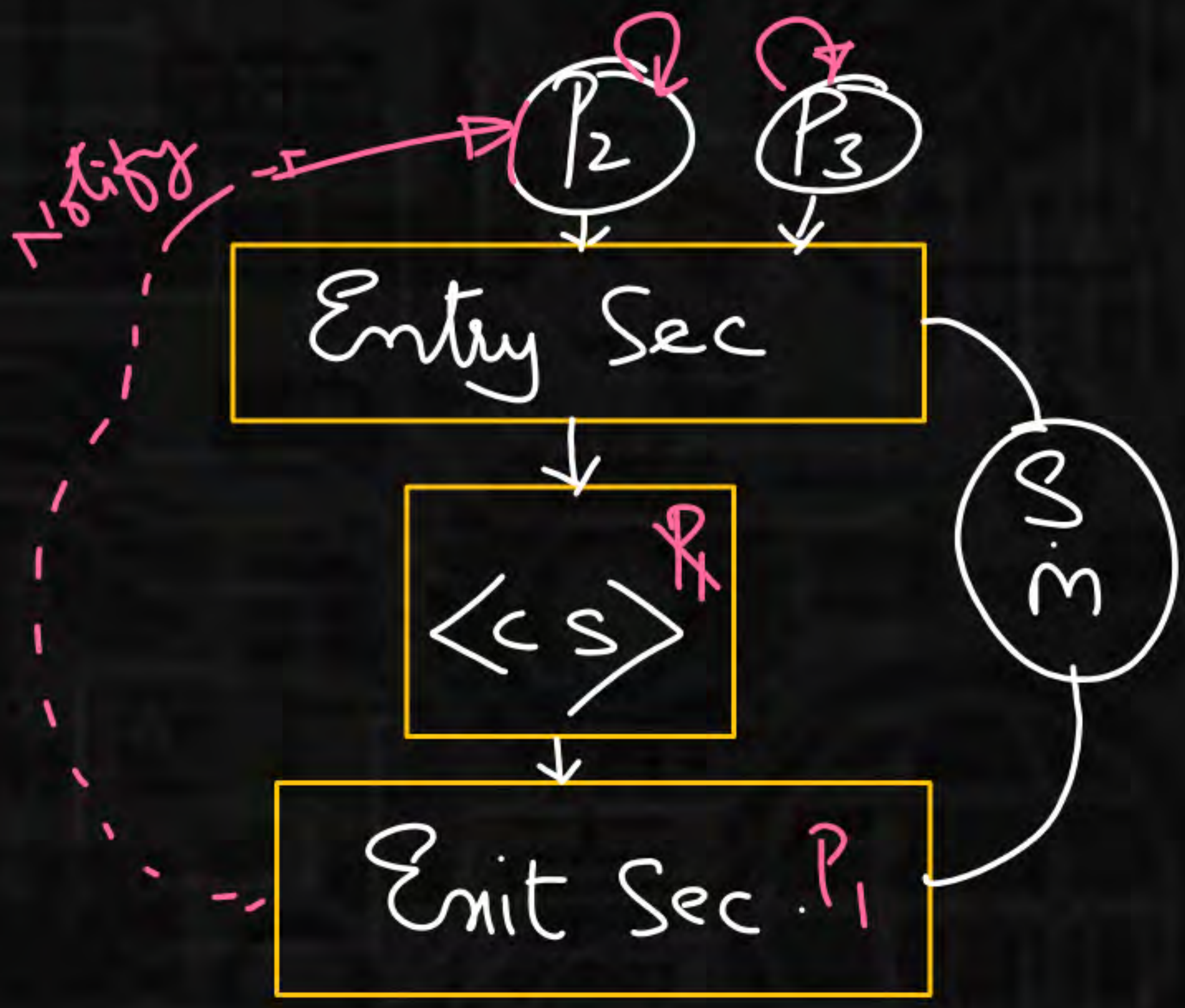
```
Do {
    Entry section

    <Critical Section>

    Exit section

    <Remainder section>

} while (true);
```



General structure

General Structure of a Typical Process with Synch. Mechanism

A solution to the Critical-Section Problem must satisfy the following three requirements:

1. **Mutual Exclusion**: If process P_i is executing in its critical section, then no other processes can be executing in their critical sections.

Critical section ×

Critical section ×

Critical section ✓

2. **Progress:** If no process is executing in its critical section and some processes wish to enter their critical sections, then only those

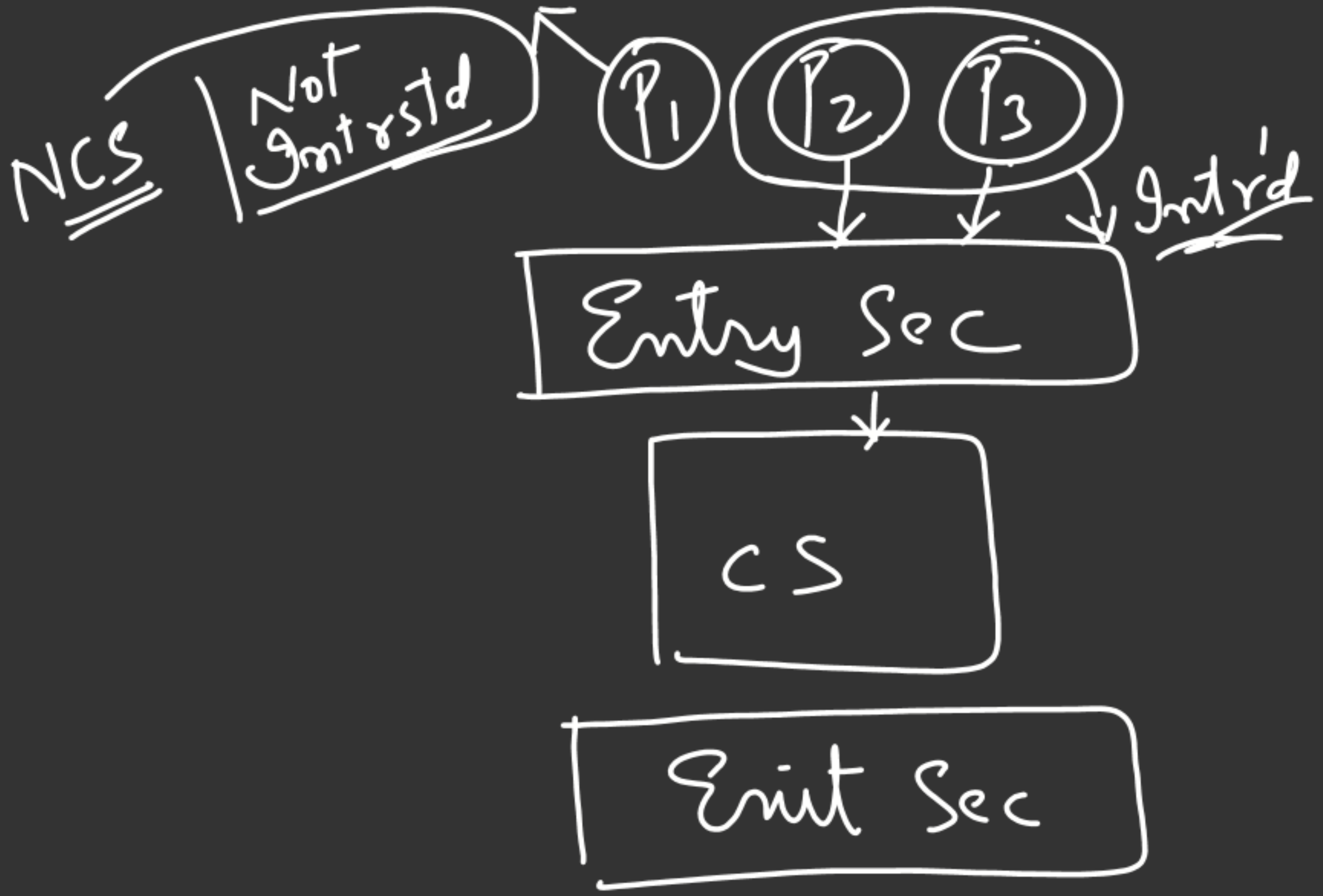
$(P_2 \; P_3)$

processes that are not executing in their remainder sections can participate in deciding which will enter its critical section next, and this selection cannot be postponed indefinitely.

3. **Bounded waiting:** There exists a bound, or limit, on the number of

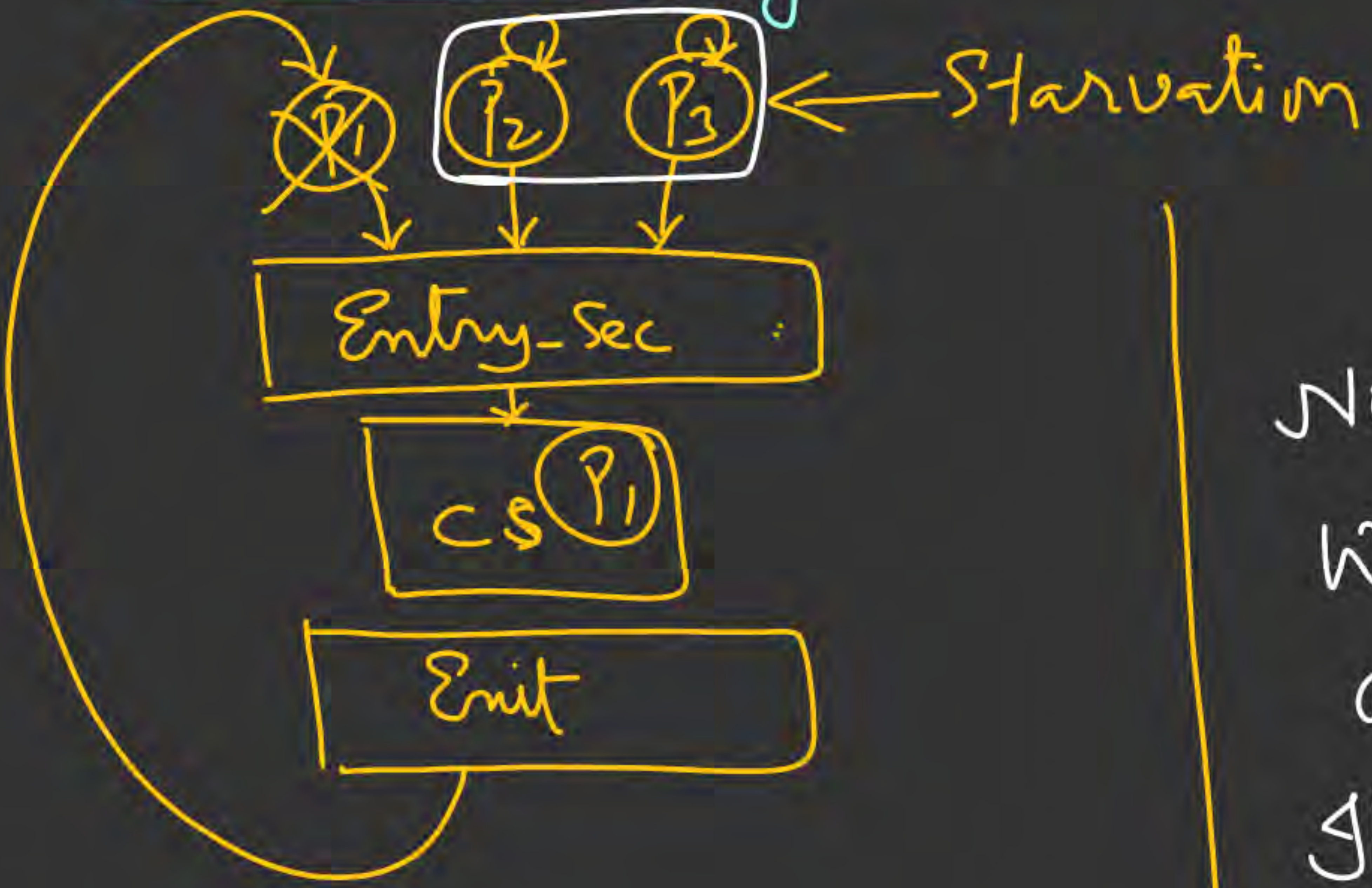
P_1

times that other processes are allowed to enter their critical sections after a process has made a request to enter its critical section and before that request is granted. P_2, P_3



No process (Not Intrstd / NCS) has got right to block other Processes (Intrst'd) from entering C.S.

Bounded waiting



No Process has to wait for ever to access C.S;

There should be a bound on the No. of Times a Process is allowed to enter C.S, b/f other Process req. is granted;

1. If M/E is Not Guaranteed, then (Inconsistency + Loss of Data) May happen

2. If Bounded waiting is not Guaranteed then Starvation;

3. If Progress is violated then it is unfair Soln (Indefinite Postponement)

```
while (count == N);
```

Synchronization Mechanisms

Busy waiting
Spin-lock

1) LOCK VAR
2) STRICT ALTERNATION
3) PETERSON SOLN
4) DEKKERS ALGO

Non-Busy-waiting
Blocking

- Sleep-wakeup
- SEMAPHORE
- MONITORS

Software
<user-mode>

Hardware
(Spl. Instns)

- TSL Instn
- SWAP "

OS-Based
(Kernel)

Assumptions:

1) Process enters <cs> & Come out of it in Finite amount of Time;

2) When a process is in entry then it means, it is intrstd in <cs>

*3) A process is said to have left <cs> only when it has executed its exit_section;

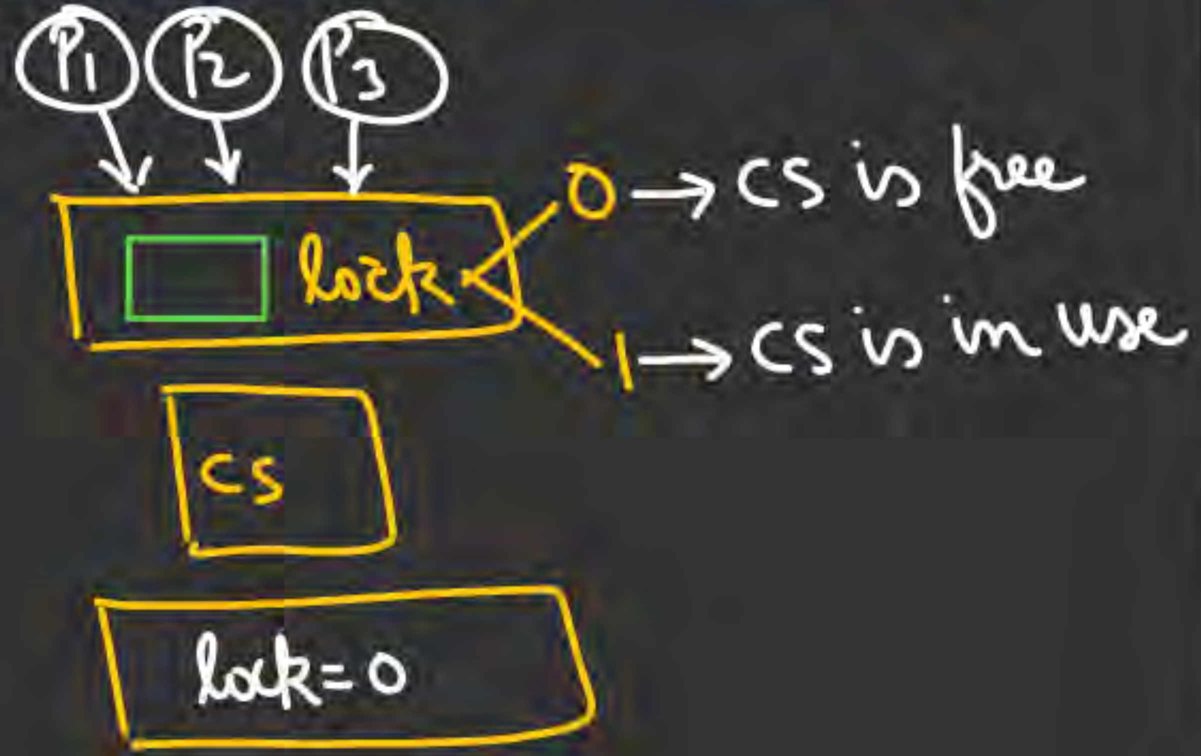
4) A process can get preEmpted from cpu, while executing Entry + <cs> + Exit section;

1) LOCK-VARIABLE:

→ Busy-waiting Soln

→ Software soln Impl. @ u/m

→ Multi-process Soln



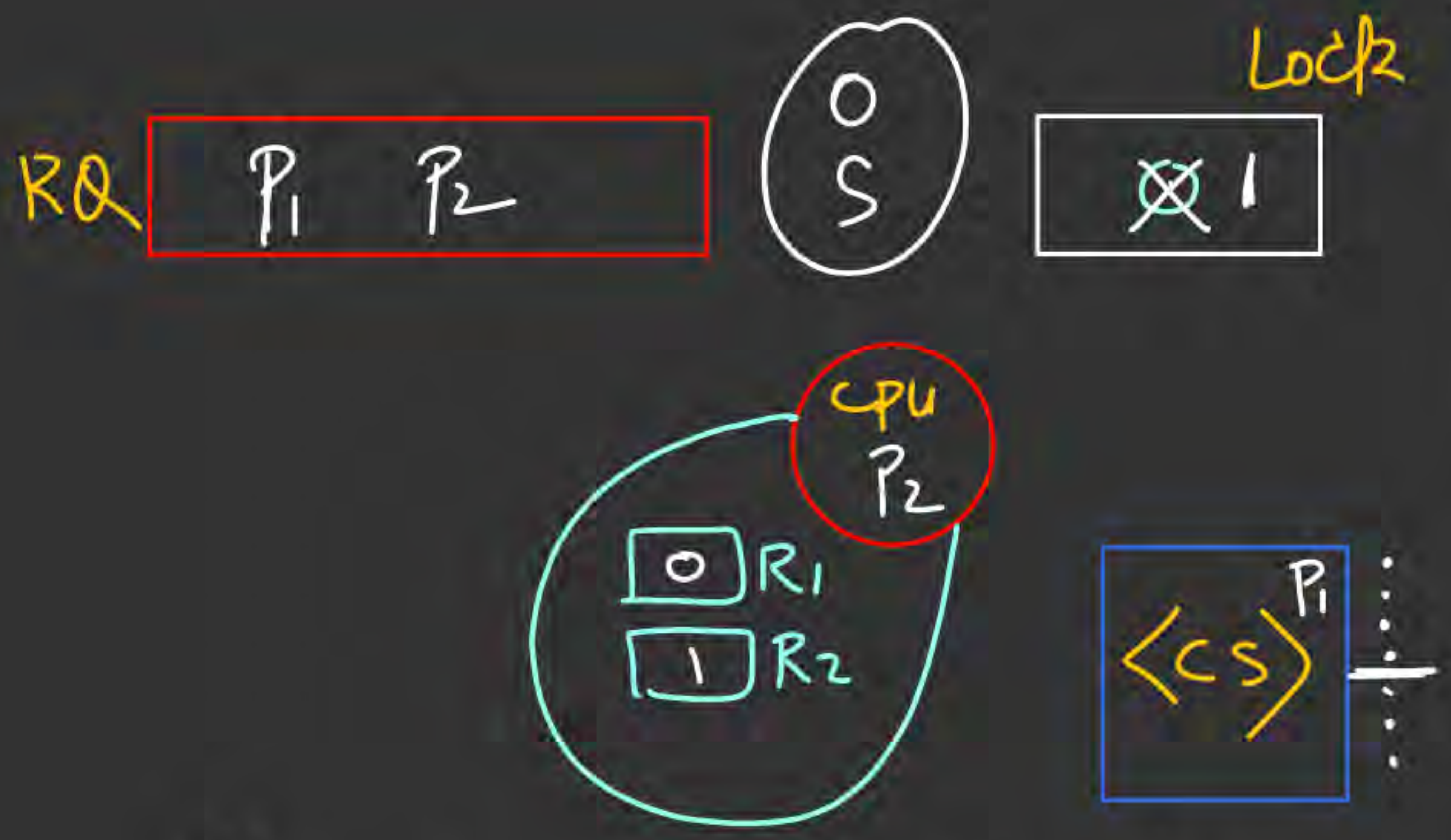
Implementation High level

```
int lock = 0;
void Process(int i)
{
  while(1)
  {
    a) <Non-CS>
    b) while(lock != 0);
       lock = 1;            Entry Section
    c) <CS>
    d) lock = 0;
  }
}
```

Low-Level Impl.

```
integer lock = 0;
Process(int i):
a) Non-CS     Ri: 0    Lock
b) Load Ri, Lock;
c) Cmp Ri, #0;
d) JNZ Step b
e) Store lock, #1;
f) <CS>
g) Store lock, #0
```

<JNZ: Jump if NOT ZERO>



P1 : <NCS> : b; c; d; e; f : <CS>

P2 : <NCS> : b; c; d; b

→ Does lock variable guarantee M/E always?

→ Progress?

→ Bounded Waiting?


THANK YOU
GW
SOLDIERS !