CS & IT ENGINEERING

Operating System

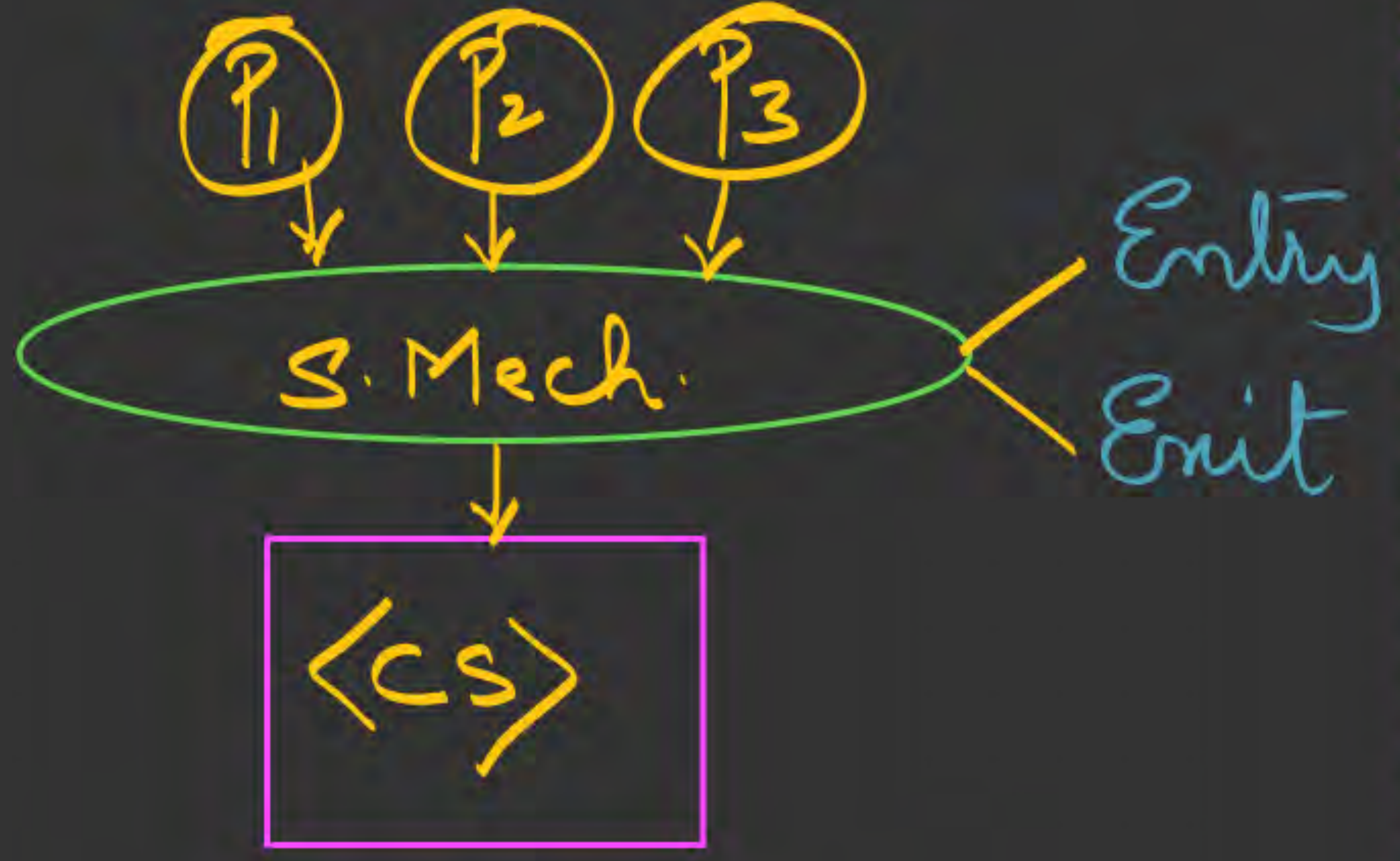
Process Synchronization & Coordination

Lecture No. 2

By- Dr. Khaleel Khan sir

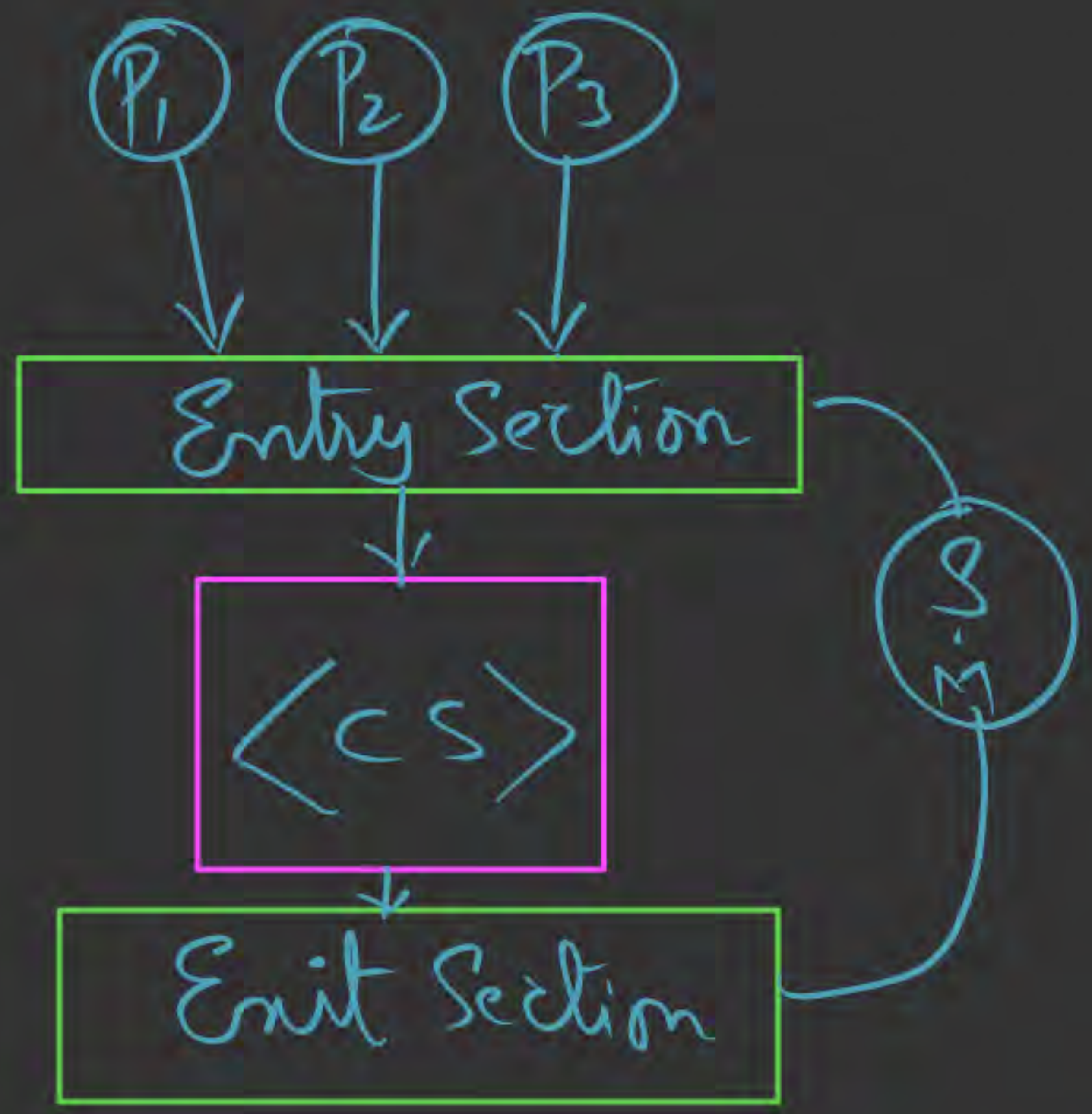
TOPICS TO BE COVERED

- Critical Section Problem
- Requirements of CS Problem
- Lock Variable



Model / Arch. of IPC Environment

<CS - Problem>



Arch. | Model of CS Problem

Requirements of Critical Section Problem:

(row + column + diagonal)

(n-Queens)

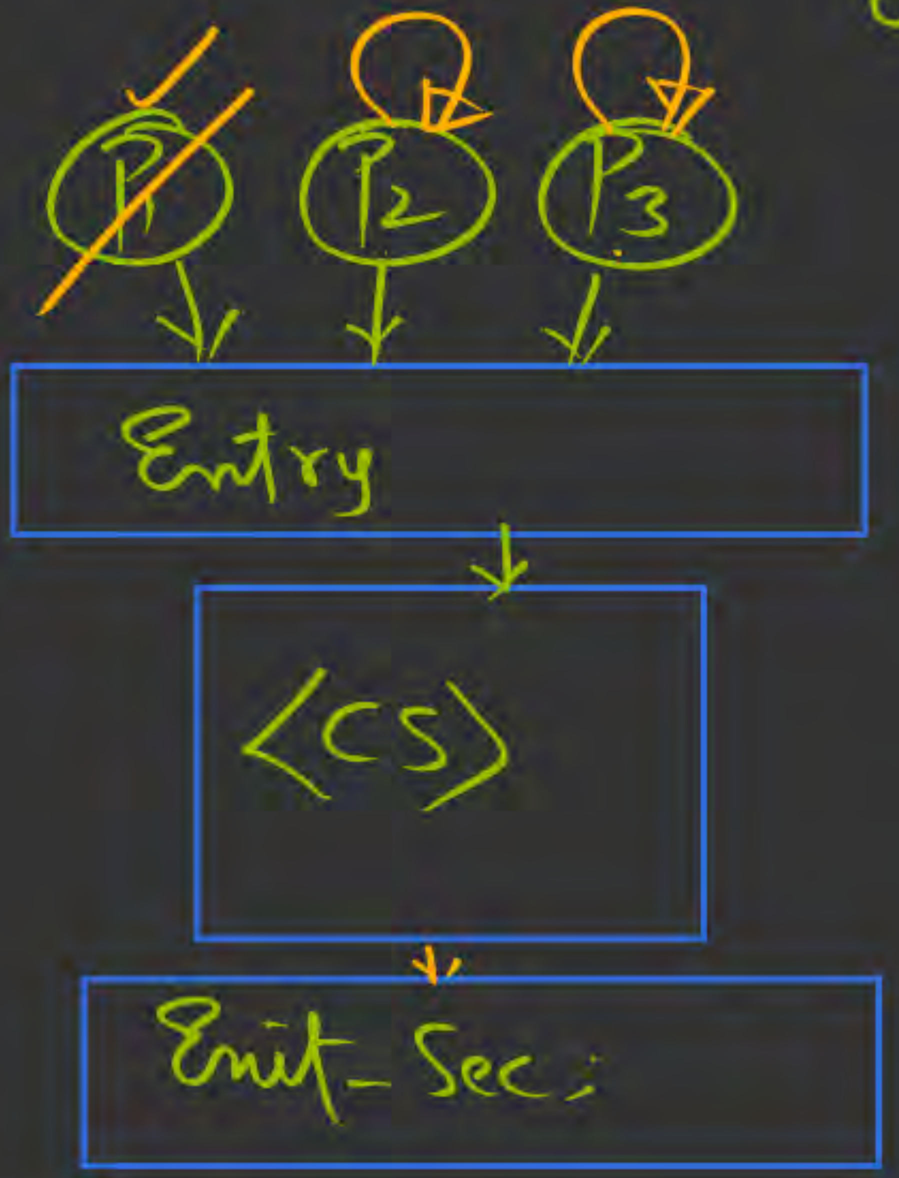
*

1) Mutual Exclusion (ME)
2) Progress
3) Bounded waiting

$n = 4$

1) <u>Mutual Exclusion</u> : No 2 - Processes may be Simultaneously Present in the CS,

2) <u>Progress</u> : No Process running outside the C.S, Should block the other Process from entering CS;



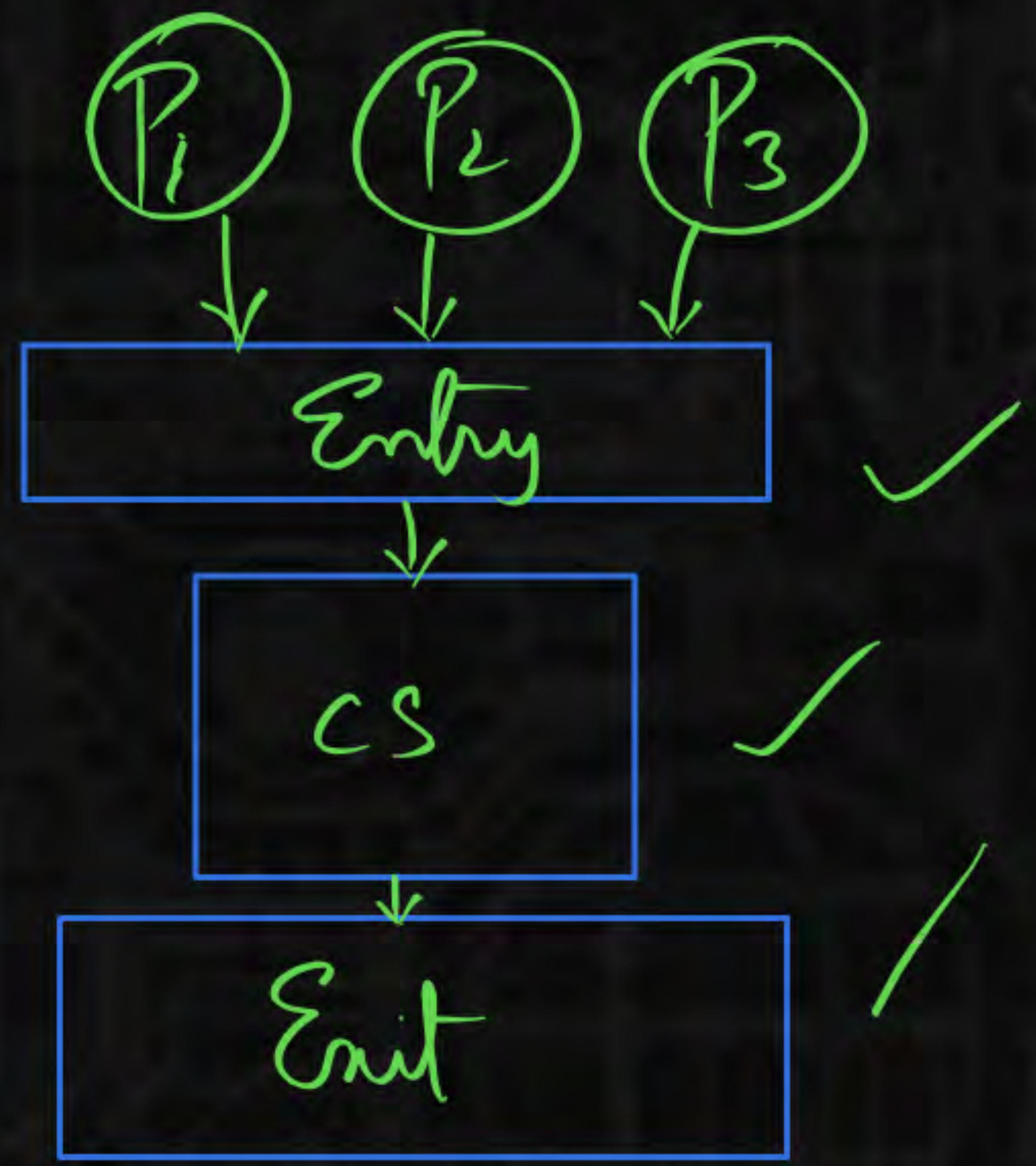
3) <u>Bounded waiting</u> : No process has to wait for ever to access its C.S;

(Starvation)

i.e there Should be a <u>bound</u> on the no. of Times a Process is allowed to enter CS, before other process request is granted;

limit

```
Do {

    Entry section

        Critical section

    Exit section

        Remainder section

} while (true);
```



General Structure of a Typical Process with Synch. Mechanism

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

Classification of Synchronization Mechanisms

(Spin-lock)
Busy-waiting
<loop>

<if-then-else>
Non-Busy-waiting
<Blocking>

(@ User-Mode)
Software
Lock ; Strict alternation ;
Peterson ; Dekker's Algo

(Spl. H/w Instrns)
Hardware
(TSL ;
SWAP)

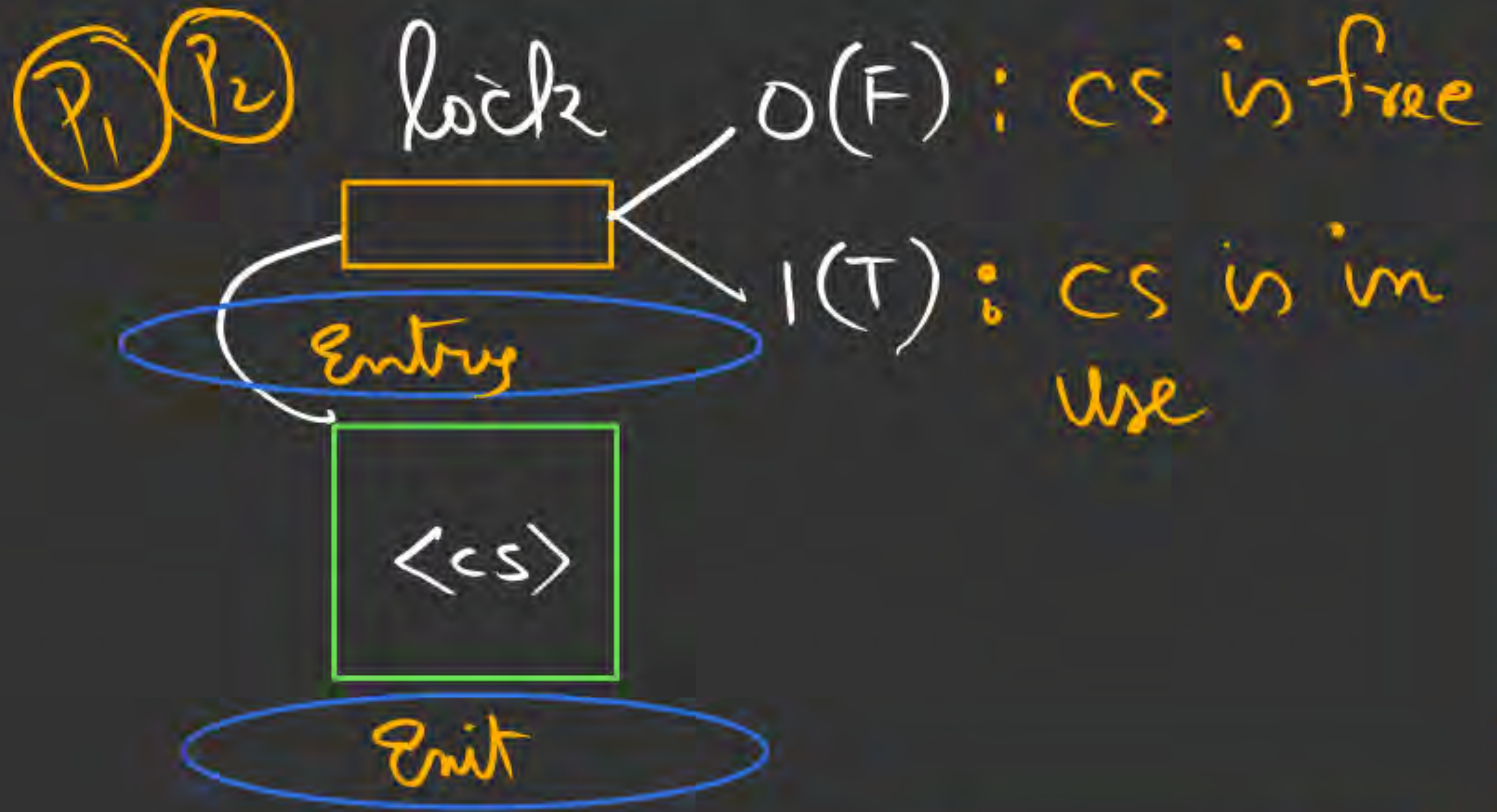
OS-Based
Sleep-wakeup ;
Semaphores ;
Monitors

Imp. points | Assumptions

1. Process enters CS & execute it in finite time & come out;
2. Any process desiring to enter CS, must execute Entry Section & finally after completing CS, must execute Exit Section.
3. The process can get preEmpted from cpu, while executing Entry-Section | CS | Exit-Section
4. Process is said to have left the CS, only when it completes exit-section;

1. Lock-Variable :

- → Busy-waiting
- → S/w Soln
- → Multi-Process Soln



Busy waiting Soln Results in wastage of cpu time

```
int lock = 0;
```

High-level Impl.

```
void Process(int i)     i=1, n
{
  while (1)
  {
    a) Non_cs();
    b) while(lock != 0);     Entry
       lock=1;
    c) <cs>
    d) lock=0;               Exit
  }
}
```

Low-Level Impl of lock variable

Process i:

1. Non-CS:

Entry

2. Load R_i, lock;
3. Cmp R_i, #0;
4. JNZ Step 2
5. Store lock, #1

6. <CS>

7. Store lock, #0;

(i) Mutual Exclusion: ✗

R.Q

~~P_1~~ P_2

lock

~~1~~ ~~0~~ 1

<CS> P_2 P_1

M.E is NOT Satisfied

(ii) Progress ✓

(iii) Bounded waiting

t_1: P_1 : 1; 2; 3; 4; Pre

t_2: P_2 : 1; 2; 3; 4; 5; 6; Pre

t_3: P_1 : 5; 6;

RQ: P_1 P_2 P_3

lock: $\cancel{0}\,1$

P_1

Entry

$\langle CS \rangle$ P_1

Exit: lock = 0

CPU

t_1: P_1 ----- $\langle CS \rangle$ Pre

t_2: P_2 ---- Busy-waiting Pre

t_3: P_3 ------ Busy-waiting Pre

t_4: P_1 --- $\langle CS \rangle$ ....

(Bounded waiting fails)


THANK YOU
GW
SOLDIERS !