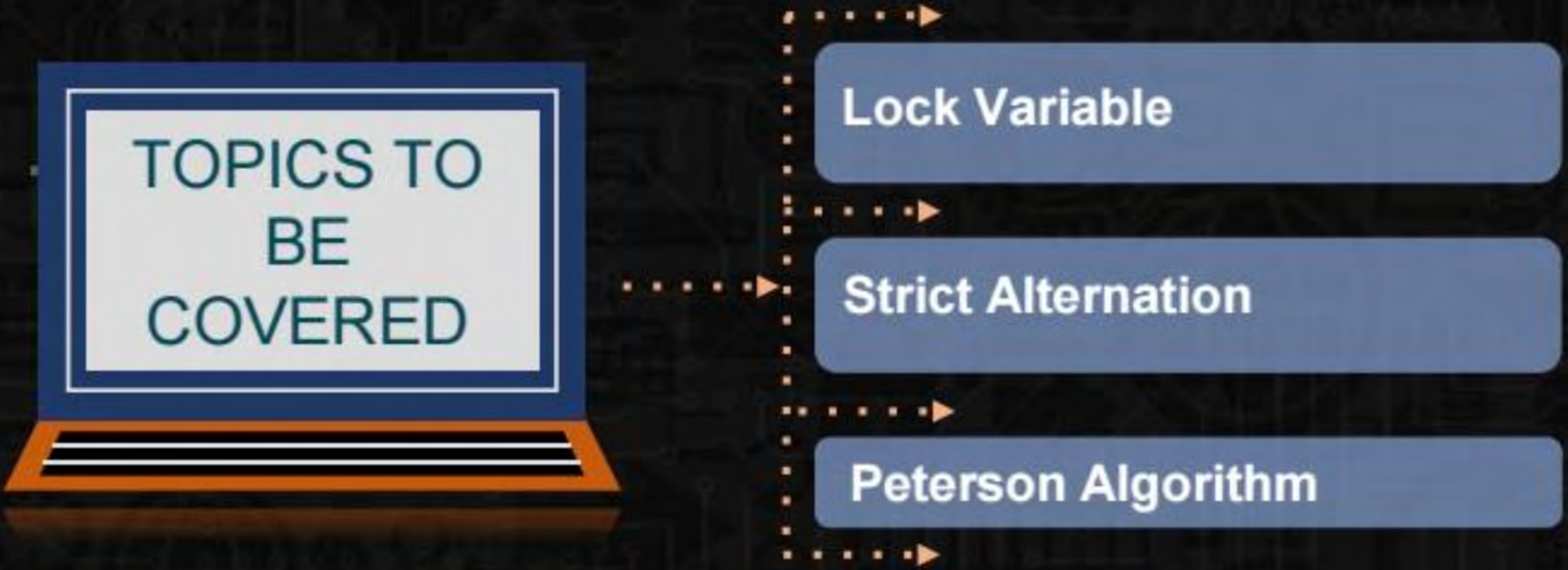
CS & IT ENGINEERING

Operating System

Process Synchronization

Lecture No. 5

By- Dr. Khaleel Khan Sir



TOPICS TO BE COVERED

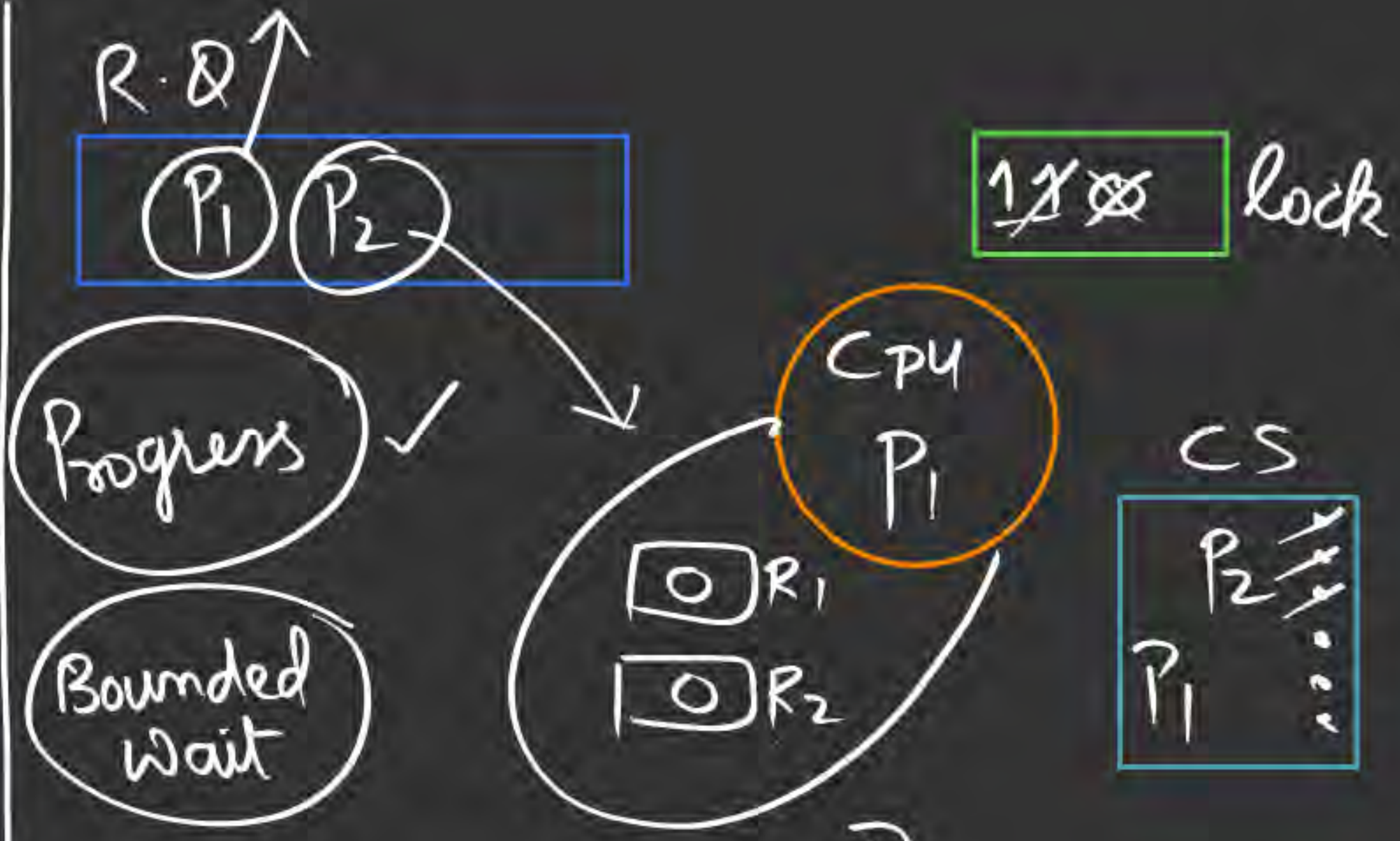
- Lock Variable
- Strict Alternation
- Peterson Algorithm

Lock variable

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

✓

```
integer lock=0
Process(i):
a) Non_CS:
b) load Ri, #lock
c) Cmp Ri, #0
d) JNZ step b
e) Store lock, #1
f) <CS>
g) Store lock, #0
```

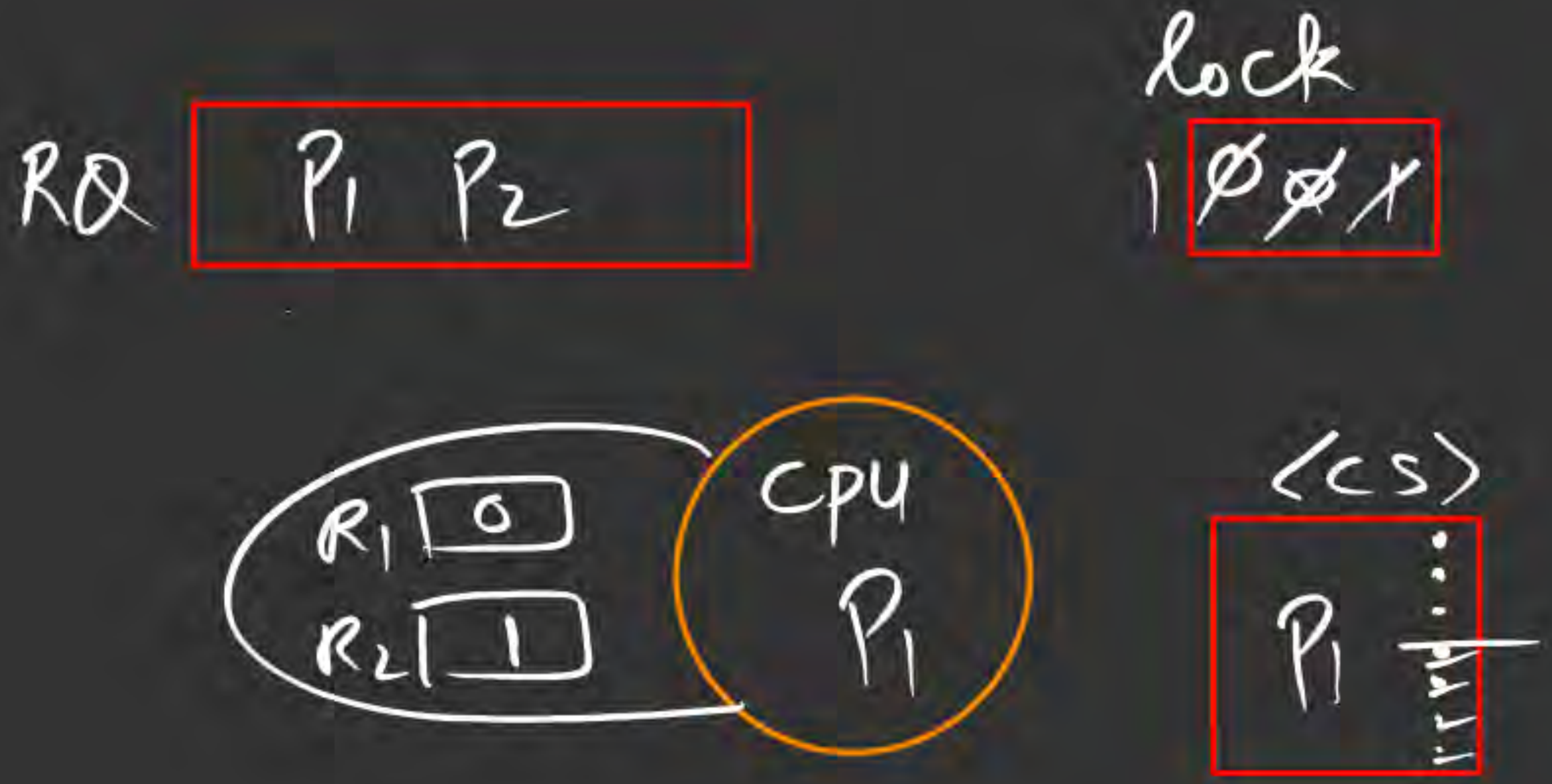


t1: P1: a; b; c; d; Pre

t2: P2: a; b; c; d; e; f <CS> Pre

t3: P1: e; f; <CS>

<Mutual Exclusion is violated>



t_1 : P_1 : a; b; c; d; e; f; <CS>; Pre

t_2 : P_2 ; a; b; c; d; Busywait ; Pre

t_3 : P_1 : <CS>; g; a; b; c; d; e; f; <CS>

Bounded wait is Not Guaranteed

1. Lock variable does not guarantees M/E always;

2. Lock " guarantees Progress always;

3. Lock " Fails to Satisfy Bounded wait

< Because a Process can enter CS multiple times Successively while other Processes are waitaing for their turn to enter 'CS' >

4. Lock variable is a Busy waiting solution leading to wastage of cpu-time/cycles (in the loop)

Q. 1

Several concurrent processes are attempting to share an I / O device. In an attempt to achieve Mutual Exclusion each process is given the following structure. (Busy is a shared Boolean Variable)

Lock-variable

```
<code unrelated to device use>   N/mcs :
        Repeat
                      NULL
Entry   until busy = false;   while (busy==TRUE);
        Busy = true;
<code to access shared >   <CS>
        Busy = false;
<code unrelated to device use>   Non CS
```

busy

IOD

Which of the following is (are) true of this approach?

I. It provides a reasonable solution to the problem of guaranteeing mutual exclusion. ×

II. It may consume substantial CPU time accessing the Busy variable. ✓

III. It will fail to guarantee mutual exclusion. ✓

A. I only

B. II only

C. III only

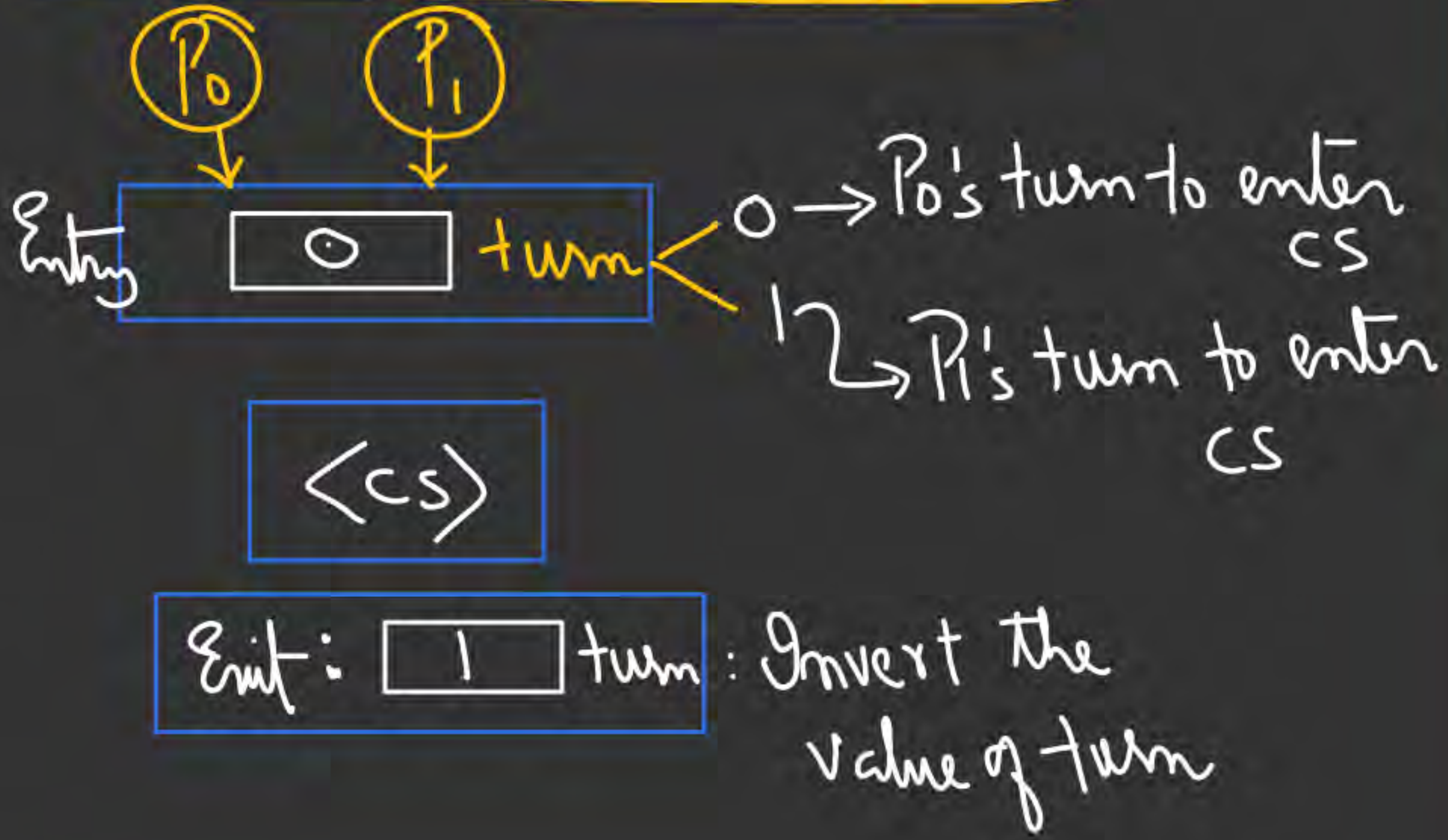
D. I & II

[✓ E. II & III] ✓

2) STRICT ALTERNATION

"Strictly on alternate basis, Processes takes turn to enter CS"

- → Busy-waiting
- → S/w Soln Impl @ u/m
- → 2-process Soln <Pi|Pj>

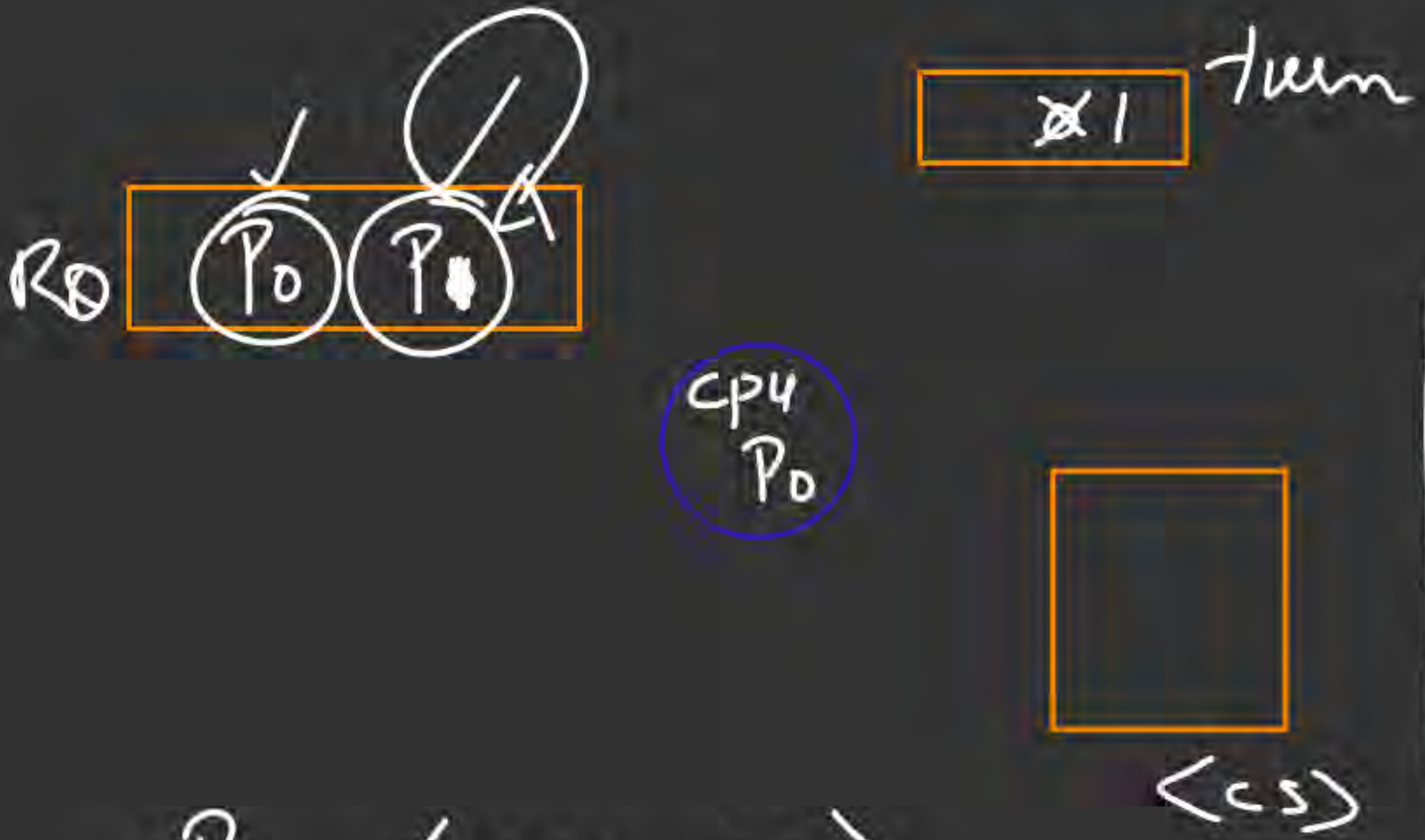


P0
P1
Entry
0
turn
0 → P0's turn to enter CS
1 → P1's turn to enter CS
<CS>
Exit : 1 turn : Invert the value of turn

```
int turn = rand(0,1);

void Process(0)
{
  while(1)
  {
    a) Non-cs();
    b) while(turn!=0);
    c) <CS>
    d) turn=1;
  }
}
```

```
void Process(1)
{
  while(1)
  {
    a) Non-cs();
    b) while(turn!=1);
    c) <cs>
    d) turn=0
  }
}
```



P0: <Non_CS>

P1: a; b;

P0 (who is not Intrst is blocking

P1

a) Always Guarantee M/E

b) Does not guarantee Progress

c) Always Guarantee Bounded wait

d) Busy-waiting <wastage of cpu-time>

Generalized Impl. of STRICT ALTERNATION

```
int turn = rand(i, j);

void Process(int i)
{
    int j = NOT(i);
    while(1)
    {
        a) Non-CS();
        b) while(turn != i);
        c) <CS>
        d) turn = j;
    }
}
```

Q.2 Consider the following two-process synchronization solution.

```
Process 0
        Entry: loop while (turn == 1);
            (critical section)
        Exit: turn = 1;
Process 1
        Entry: loop while (turn == 0);
            (critical section)
        Exit: turn = 0;
```

Strict Alternation

The shared variable turn is initialized to zero.
Which one of the following is TRUE? MCQ

A. This is a correct two-process synchronization solution. ✗

B. This solution violates mutual exclusion requirement. ✗

C. This solution violates progress requirement. ✓

D. This solution violates bounded wait requirement. ✗

```
int turn = 1;

void Process(0)
{
   While (1)
   {
      a) Non_CS();
      b) while (turn==1);
      c)   <CS>
      d) turn=1;
   }
}
```

```
void Process(1)
{
   While (1)
   {
      a) Non-CS();
      b) while (turn==1);
      c) <CS>
      d) turn=1;
   }
}
```

Q

Does this guarantee M/E ?

(Fails)

Can any Process enter CS Now ?

"Deadlock"

```
int turn = rand(i, j)
void Process(int i)
{
    int j = NOT(i);
    while (1)
    {
        a) Non-cs();
        b) while (turn != i);
              turn = j;
        c) <cs>
        d) turn = j;
    }
}
```

H/W

(i) Does this Guarantee M/E ?

(ii) Progress ?

(iii) Bounded wait ?

3) PETERSON's SOLUTION/ALGO.

- → Busy-waiting
- → Software Soln Impl @ u/m
- → 2-process Soln
- → L.V + S.A

```
#define N 2
#define TRUE 1
#define FALSE 0
int flag[N]={FALSE}
int turn;
```

```
void Process(int i)
{
    int j = NOT(i);
    while(1)
    {
        a) Non_cs();
        b) flag[i]= TRUE;
        c) turn = i;
        d) while(flag[j]==TRUE &&
                 turn == i);
        e) <CS>
        f) flag[i]=FALSE;
    }
}
```

F && X

Short Circuit

RQ | P_0 P_1 |

CPU P_1

flag[0] = ~~F~~ ~~T~~ ~~F~~ T

flag[1] = ~~F~~ T

| 0 ~~0~~ ~~1~~ | turn

<CS>

t_1: P_0 $i=0; j=1$: a; b; c; Pre

t_2: P_1 $i=1; j=0$: a; b; c; d : Pre

t_3: P_0 $i=0; j=1$: d; e: <CS>; a; b; c; d

✓ (i) Guarantee M/E always.

✓ (ii) Guarantee Progress

✓ (iii) " Bounded Wait

THANK YOU
GW
SOLDIERS !