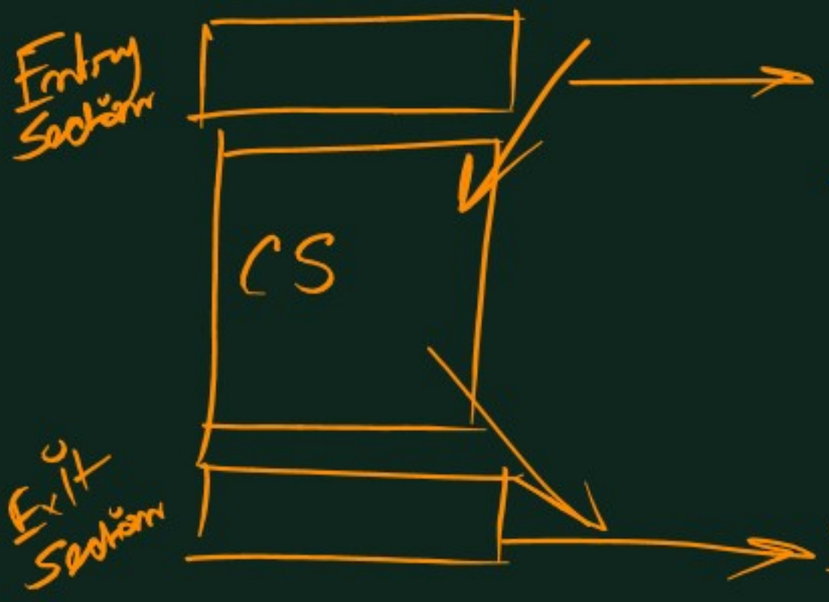
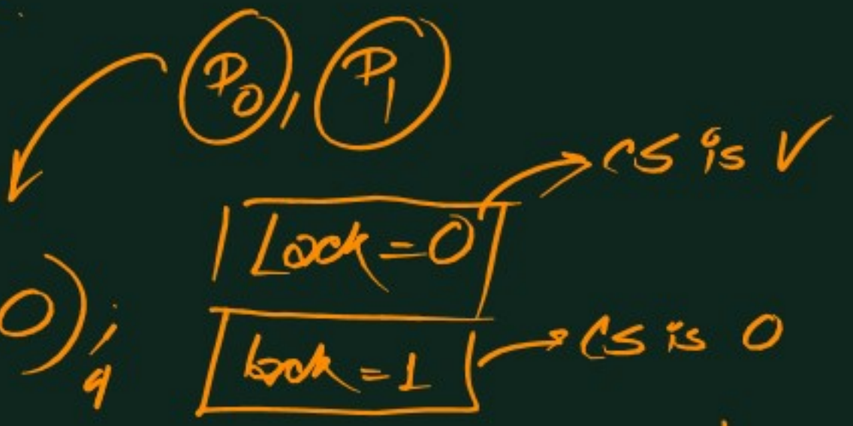
Process Synchronization



Lock

```
1. while (Lock != 0);
2. Lock = 1;
   CS
3. Lock = 0;
```



P_0 : 1, 2, CS | P_1 : 1 ↻ | P_0 : 3

Assembly code

```
1. MOV R0, LOCK
2. CMP R0, #0
3. JNZ Step 1
4. STORE LOCK, #1
```

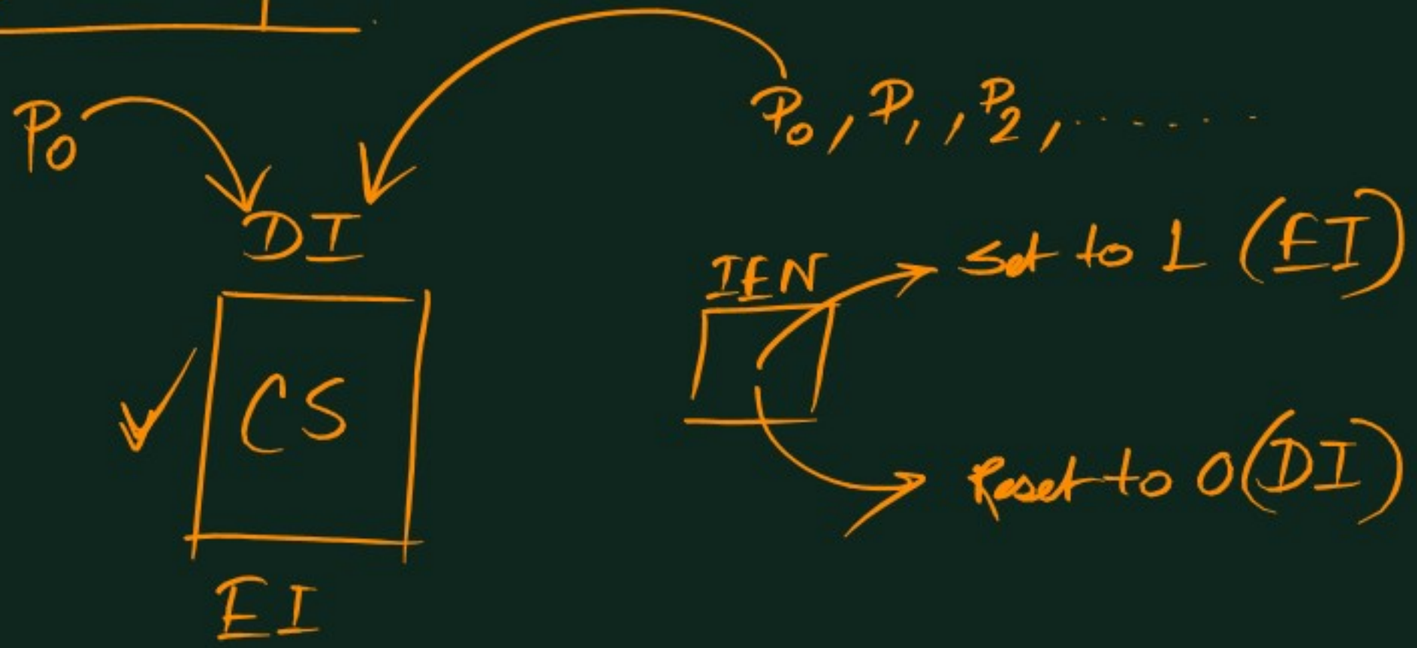
1. Mutual Exclusion } Primary
2. Progress

3. Bounded Waiting } optional
4. AN/PI

X TSL Lock ⇒ TSL
X
CMP R0, #0
JNZ step L

MOV R0, Lock
STORE Lock, #1
CMP R0, #0
JNZ step L

Interrupts



Turn Variable (with strict alternation)

- Busy waiting solution
- 2-process solution (P_0, P_1) (P_i, P_j)
- S/W mechanism implemented at "user mode"

P_0

Non CS

Ent S: while(turn != 0);

CS

Ex S: turn = 1;

Non CS

P_1

Non CS

Ent S: while(turn != 1);

CS

Ex S: turn = 0;

Non CS

L

0 → P_0, P_1, P_2

1 → X

T

0 → P_0

1 → P_1

int turn → 0 ✓, 1

ME ✓

P X

BW ✓

AN ✓

P_1: Ent S | P_0: Ent S, CS, Ex S | P_1: Ent S, CS, Ex S

Interested Variable:

Interested[0] = T → P_0 is interested in executing CS

Interested[1] = T → P_1 " " " " "

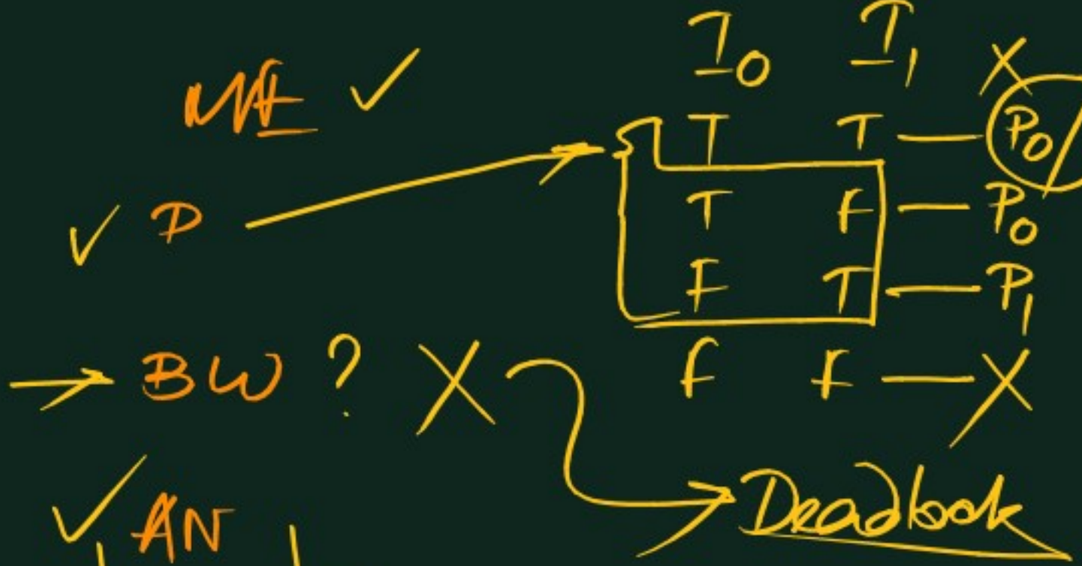
```
P0
while(1)
    Non-CS
    EnS: Intr[0]=T;
         while(Intr[1]==T);
    CS
    ExS: Intr[0]=f;
```

```
P1
while(1)
    Non-CS
    EnS: Intr[1]=T;
         while(Intr[0]==T);
    CS
    ExS: Intr[1]=f;
```

Intr[0] = ~~f~~ ~~T~~ f

Intr[1] = ~~f~~ T

P_0 : EnS, CS | P_1 : EnS | P_0 : CS, ExS | P_1 : EnS, CS



✓ AN

P_0	P_1	P_0	P_1
I_0=T	I_1=T	I_0=f	W
CS	W	I_0=T	
		W	

Lock < TSL

Interrupt

{ Turn Variable
  Interested Variable

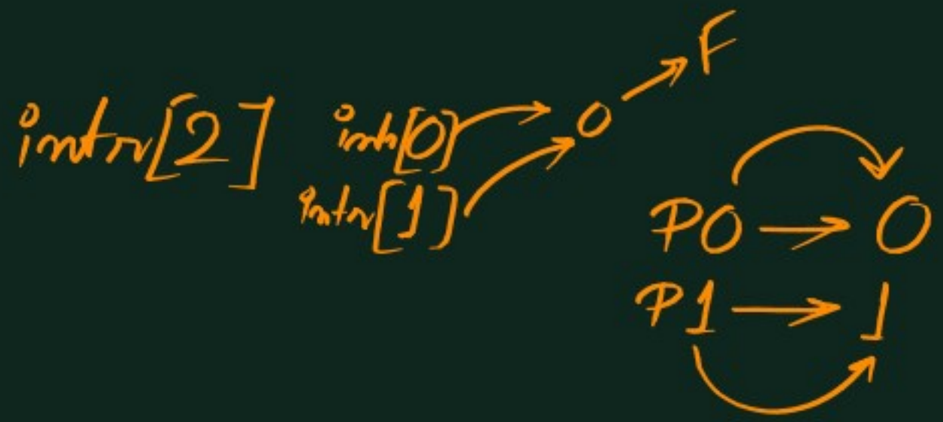
Peterson → 2-process solution

Merge { Turn → Global
        Interested → local

Peterson's Algorithm

```
#define N 2
#define TRUE 1
#define FALSE 0
int interested[N] = FALSE;
int turn;
void Entry_section (int process)
{
    int other;
    other = 1 - process;
    interested[process] = TRUE;
    turn = process;
    while ( interested[other] == TRUE && turn == process);
}
```

```
void Exit_Section (int process)
{
    interested[process] = FALSE;
}
```



```
#define N 2
#define TRUE 1
#define FALSE 0
int interested[N] = FALSE;
int turn;
void Entry_section (int process)
{
1. int other;
2. other = 1 - process;
3. interested[process] = TRUE;
4. turn = process; }
5. while( interested[other] == TRUE && turn == process);
}

void Exit_section (int process)
{
6. interested[process] = FALSE;
}
```

whichever process executes this line first, will get the chance to get into the CS, first

Situation 1:

P_0: 1 2 3 4 5 CS | P_1: 1 2 3 4 5 ↻ | P_0: CS

P_1: 5 CS

interested: F T

turn: 0

Situation 2:

P_0: 1 2 3 | P_1: 1 2 3 4 5 ↻ | P_0: 4 5 ↻ | P_1: 5 CS

interested: T T

turn: 0 ✓

P_0: 1 2 3 | P_1: 1 2 3 | P_0: 4 5 ↻ | P_1: 4 5 ↻ | P_0: 5 CS 6

interested: T F → T T

turn: 0 → 1 0

Case 1: P_1: 5 CS

Case 2: P_0: 1 2 3 4 5 ↻