Critical regions :

Critical region is a high level linguistic structure consisting of a sequence of instructions forming a critical section that manipulates state variables shared by multiple processes.

Implementation :

Inconditionnal Critical Regions :

All shared variabales must explicitely be declared as shared.
The shared variabales can only be accessed inside the critical Region

```
Var V: shared type
Region V do
Begin
    SC      // critical section where the variable V is manupulated
end
```

The CR insures mutual exclusion among processes, only one process at any given moment can access a CR.

Region V do SC1 → process P_1

Region V do SC2 → process P_2

in this example only P_1 or P_2 can enter the CR while the other waits.

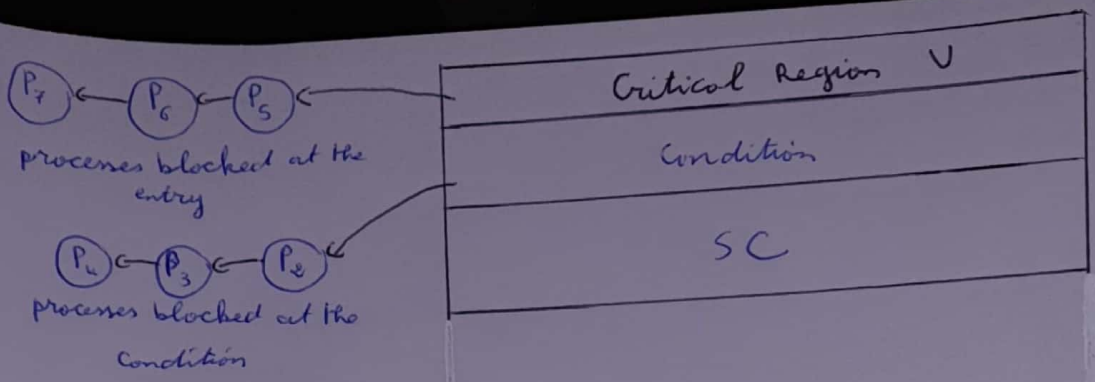
Conditionnal Critical Regions :

It is a critical region where the acces is tide to a condition. There are two formes of CCR.

1) Region V when condiction do SC

Acces Protocole :

- First P needs to have acces to the CR, if not, P is blocked at the entry
- If P has access to the CR, P evaluate the condition.
  - * Condition == false : P frees the CR and is blocked at the condition
  - * Condition == true : P executes SC and leaves when it is done
- When leaving CR, P wakes up a process blocked at the condition if there is one or it opens the CR.

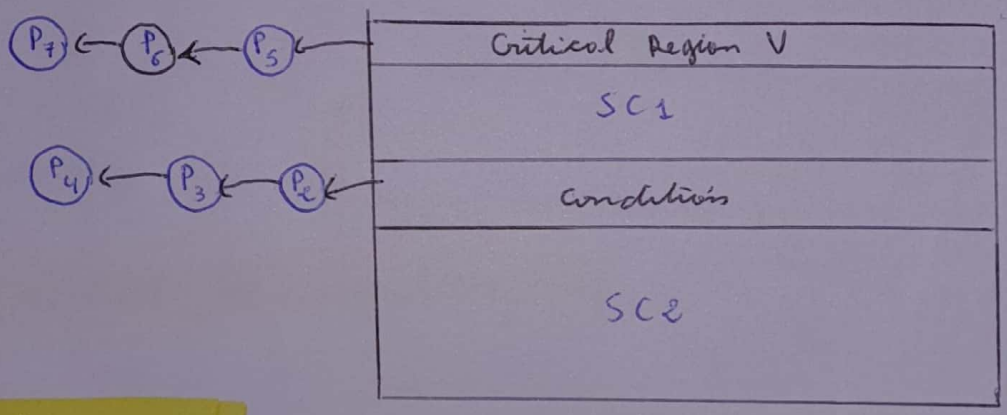


2) Region V do SC1 await(condition) SC2

Access Protocole :

- First P needs to have acces to the CR, if not, P is blocked at the entry
- If P has access to the CR, it executes SC1 then it evaluates the condition :
  - * condition == false : P frees the CR and it is blocked at the condition
  - * Condition == true : P executes SC2 and leaves when it's done
- when leaving CR, P wakes up a process blocked at the condition if there is on

or it opens the CR



Rendez - vous de N process :

Les variables :

Var : cpt: shared integer init 0;

L'implementation :

```
Pi :
  Region cpt do
  Begin
      cpt ++;
      await(cpt == N);
  End;
```

Producteur / Consommateur :

Variables :

```
Var  Buffer : shared record
                Tampon : Array [1 .. N] of element
                nPlein : integer
                in , out : integer
              end ;

init : Buffer.nPlein = Buffer.in = Buffer.Out = 0 ;
```

Implementation :

Producteur :

```
Var resultat : element
Produire (resultat)
Region Buffer when nPlein < N do
Begin
    nPlein ++ ;
    Tampon [In] = resultat ;
    in = (in + 1) mod N ;
End ;
```

Consommateur :

```
Var donnee : element
Region Buffer when nPlein > 0 do
Begin
    nPlein -- ;
    donnee = Tampon [out] ;
    out = (out + 1) mod N ;
End ;

Consommer (donnee) ;
```

Lecteurs / Redacteurs : priorité absolue aux Lecteurs :

Variables :

```
Var V : shared record
            nl : integer
            E : boolean
            nlatt : integer
        End ;

init  nl = 0 ;
      E = fals ;
      nlatt = 0 ;
```

Implementation:

DL :

```
Region V do
Begin
    nlatt ++;
    await (E == false);
    nlatt --;
    nl ++;
End;
```

FL :

```
Region V do
Begin
    nl --;
End;
```

DE :

```
Region V when ( E == false && nlatt == 0)
do
Begin
    E = true;
End;
```

FE :

```
Region V do
Begin
    E = false;
End;
```

Simulation d'un sémaphore général :

```
Var  S: shared integer init N

P(S)
    Region S when S > 0 do S--;

V(S)
    Region S do S++;
```