Synchronisation et communication entre processus

Schémas classiques de synchronisation

INTRODUCTION

- **Système multiprocessus**
- **L'ordonnancement "entrelace" les exécutions**

☞ **Processus non indépendants :**
accès concurrents aux ressources

Notion de ressources

- Définitions

  – Une ressource désigne toute entité dont a besoin un processus pour s'exécuter.
    - Ressource matérielle (processeur, périphérique)
    - Ressource logicielle (variable)

  – Une ressource est caractérisé
    - par un état : libre /occupée
    - par son nombre de points d'accès (nombre de processus pouvant l'utiliser en même temps)

Notion de ressources

- Utilisation d'une ressource par un processus

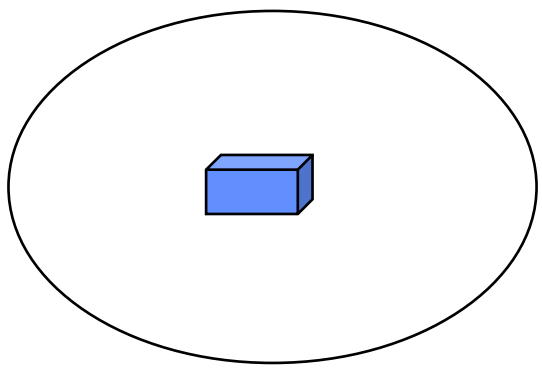
  – Trois étapes : Allocation
  Utilisation
  Restitution

  – Les phases d'allocation et de restitution doivent assurer que le ressource est utilisée conformément à son nombre de points d'accès

    - ressource critique à un seul point d'accès

Un premier problème simple : l'exclusion mutuelle entre processus

Utilisation d’une ressource critique



Introduction : un exemple simple de concurrence

Client 1

Client 2

Réservation :
Si nb_place > 0
alors
Réserver une place
nb_place = nb_place - 1
fsi

nb_place

Introduction : un exemple simple de concurrence

Réservation :
Si nb_place > 0
alors
Réserver une place
nb_place = nb_place - 1
fsi

Client 1

Demande Réservation

Nb_Place > 0 = 1

Ordonnancement/ commutation

Client 2

Demande Réservation

Nb_Place > 0 = 1

Nb_Place = Nb_Place - 1

Nb_Place = 0

Ordonnancement/ commutation

Nb_Place = Nb_Place - 1

Nb_Place = -1 !!!

Section critique et exclusion mutuelle

Processus
Début

Entrée Section Critique

Ressource Critique
Nb_Place

Sortie Section Critique

Fin

- **Ressource utilisable par un seul processus à la fois**

SECTION CRITIQUE (code d'utilisation de la ressource critique)

☞ L'entrée et la sortie de SC doivent assurer qu'à tout moment, un seul processus s'exécute en SC (exclusion mutuelle)

Un exemple simple de concurrence

Client 1

Demande Réservation
Si nb_place > 0
alors
Réserver une place
nb_place = nb_place - 1
fsi

Client 2

Demande Réservation

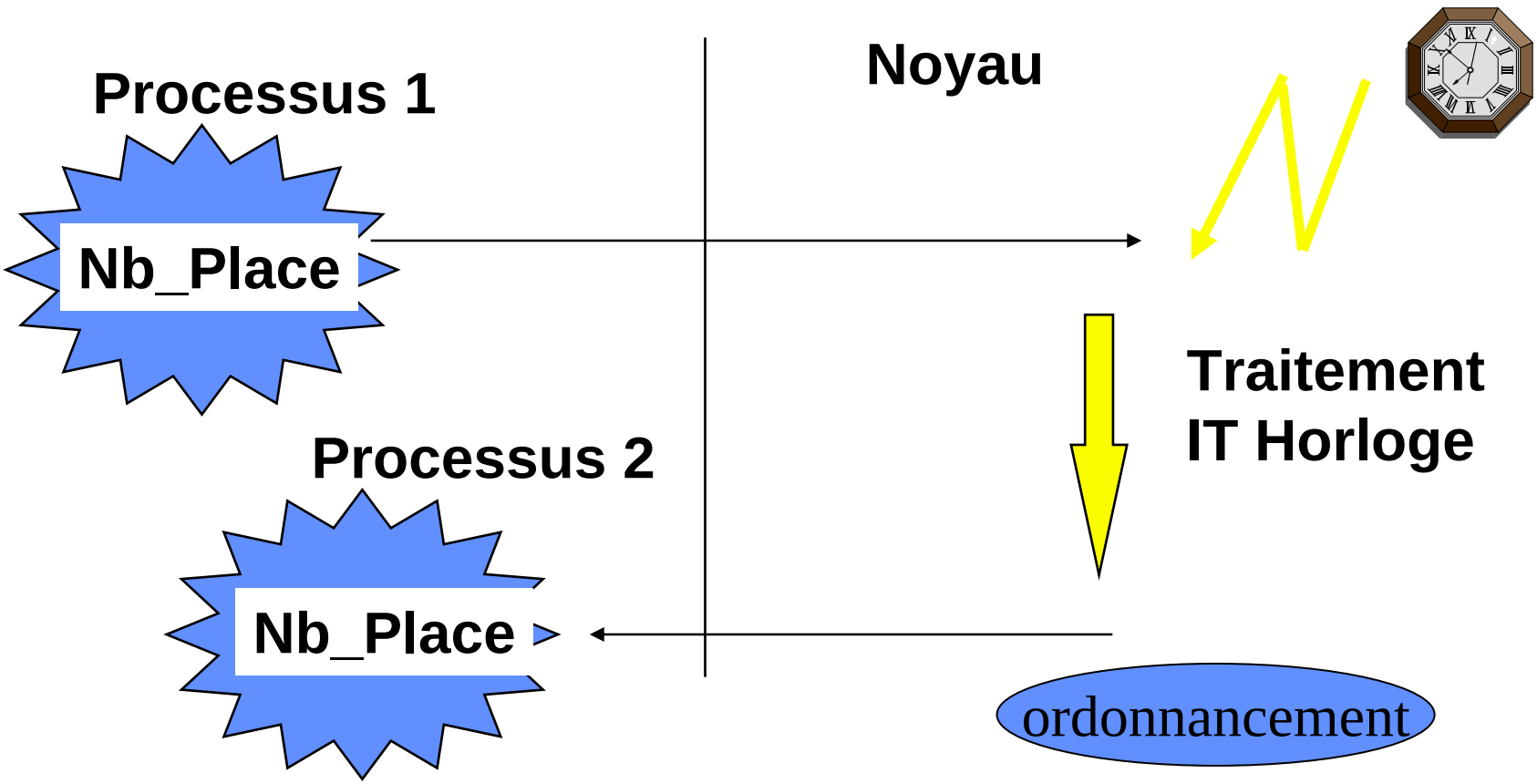
Nb_Place non accessible

Pour le client 2 tant que

client 1 manipule cette variable

Section critique et exclusion mutuelle

- **Masquage des interruptions**

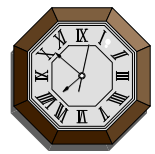


Problème de la section critique
Solution matérielle

- **Masquage des interruptions**

Processus 1

```
Masquer IT
Si nb_place > 0
alors
        Réserver une place
        nb_place = nb_place - 1
fsi
Démasquer IT
```



IT

Non prise en compte

Processus 2

Attente

Problème de la section critique

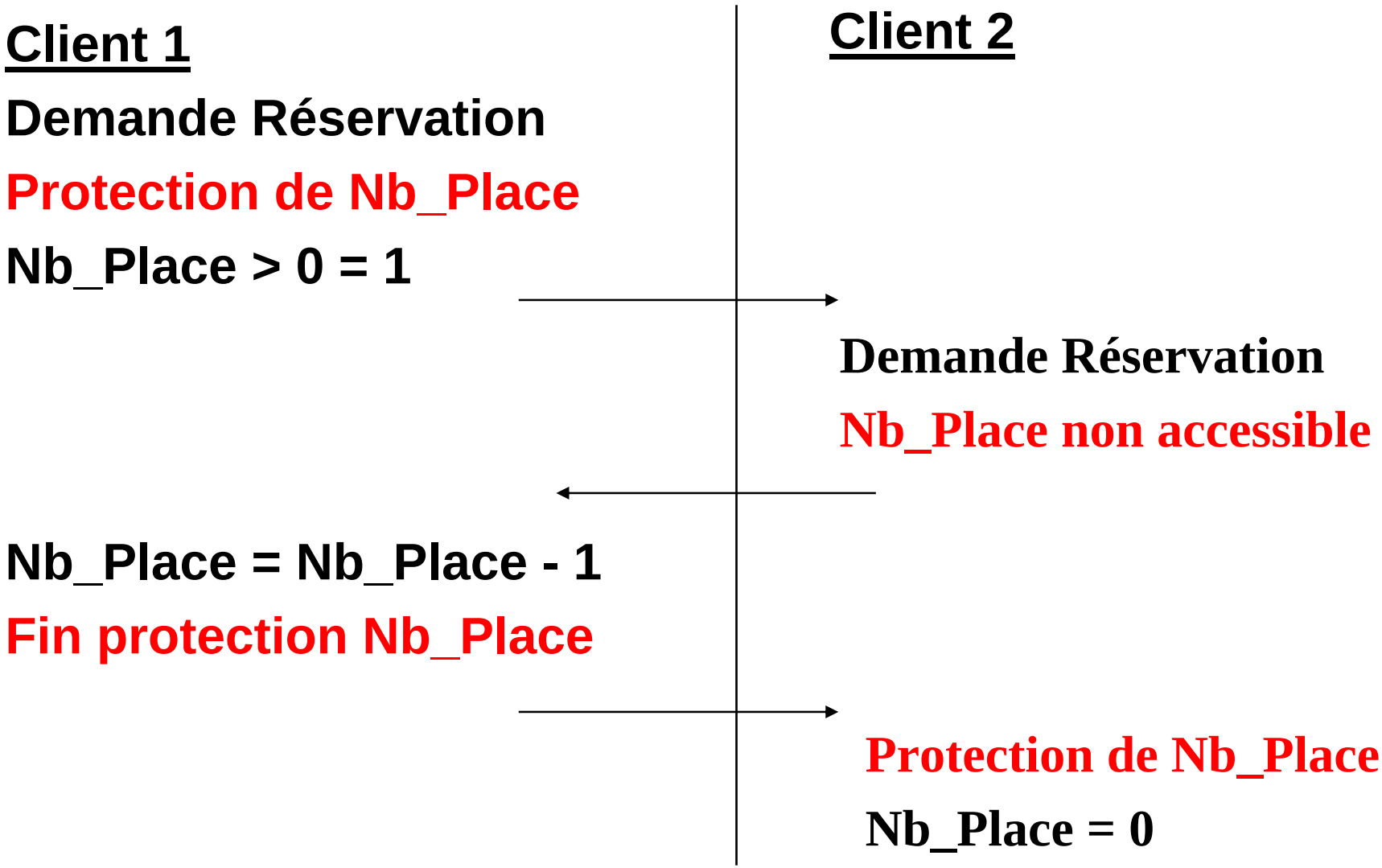
- **Une ressource critique est une ressource accessible par un seul processus à la fois. L'accès se fait en exclusion mutuelle dans la section critique.**

- **Une solution pour réaliser une section critique est d'interdire la prise en compte des interruptions durant l'utilisation de la ressource critique.**

- **Mais**
  - **dangereux**
  - **mode superviseur**
  - **Bloque tous les processus de la machine**

    ☞ **bloquer seulement les processus accédant à nb_place**

Un exemple simple de concurrence



Un exemple simple de concurrence

Client 1

Demande Réservation

Protection de Nb_Place →

Nb_Place > 0 = 1

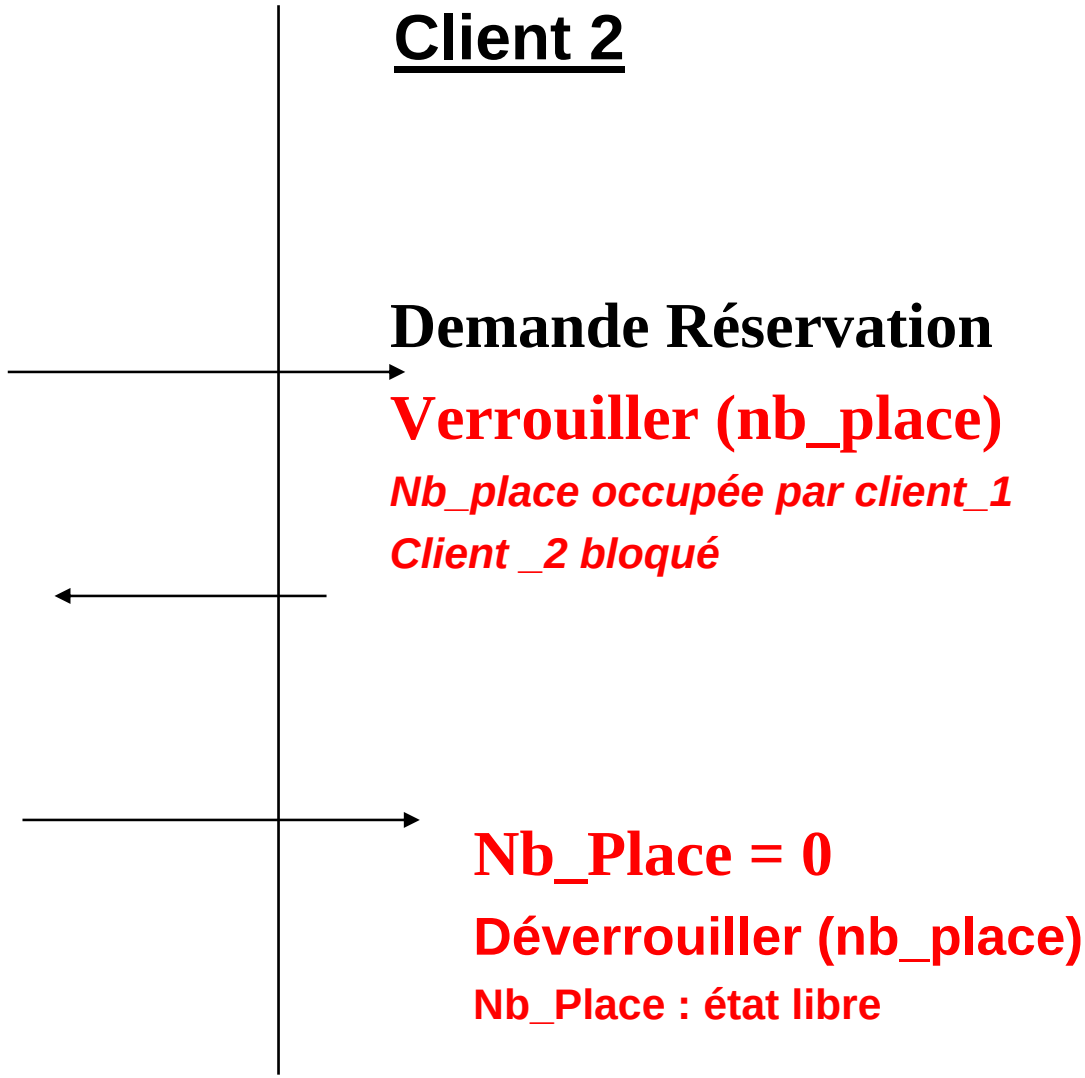
Verrouiller (Nb_Place)
Si Nb_Place libre alors autoriser l'accès et mettre Nb_Place à l'état occupé sinon bloquer le processus

Nb_Place = Nb_Place - 1

Fin protection Nb_Place →

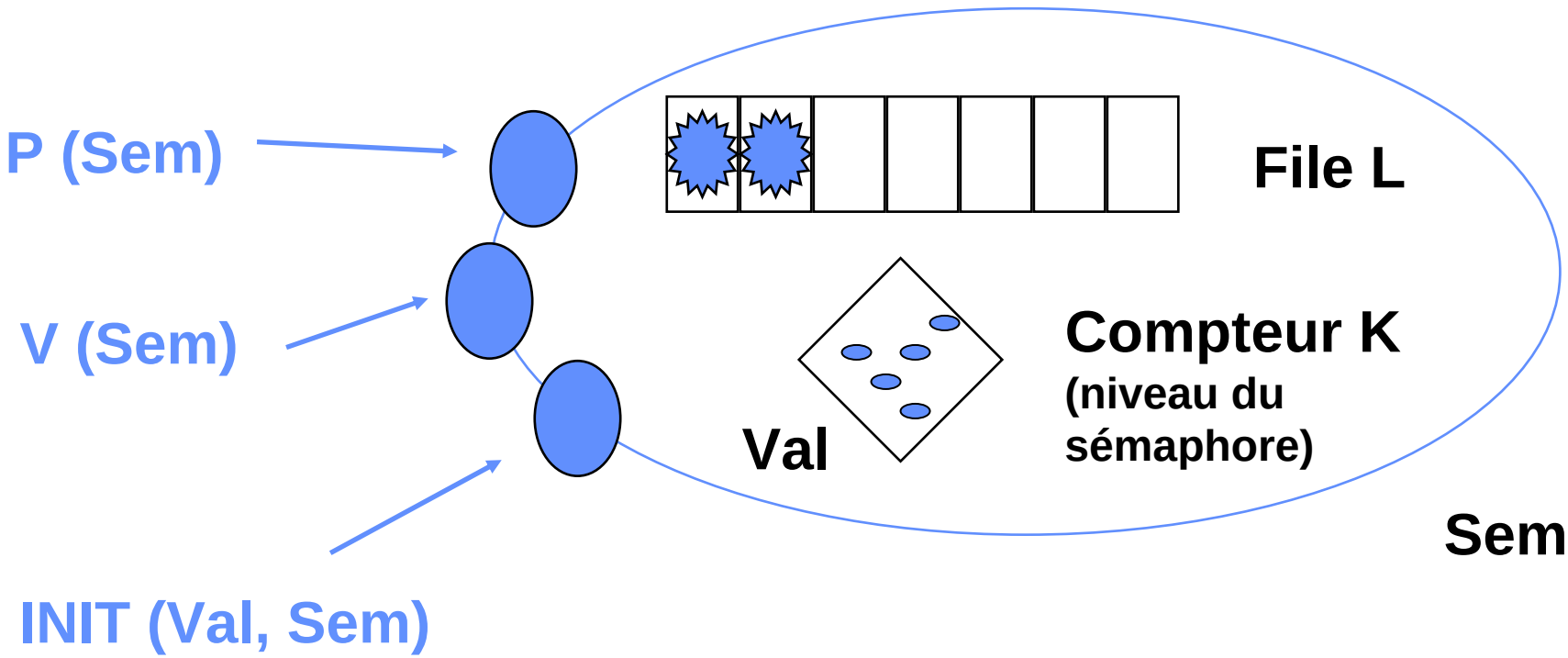
Déverrouiller (Nb_Place)
Si un processus bloqué en attente pour accéder à Nb_Place, le débloquer, Sinon Nb_Place libre

Un exemple simple de concurrence



Les sémaphores

- **Structure**



Opérations indivisibles

Les sémaphores

- **Un sémaphore Sem peut être vu comme un distributeur de jetons; le jeton étant un droit à poursuivre son exécution**
  - **l'opération INIT (Sem, Val) fixe le nombre de jetons initial**
  - **l'opération P(Sem) attribue un jeton au processus appelant si il en reste sinon bloque le processus dans Sem.L**
  - **l'opération V(Sem) restitue un jeton et débloque un processus de Sem.L si il en existe un.**

Les sémaphores

- **Opération Init (Val, Sem)**

Init (Val, Sem)

début

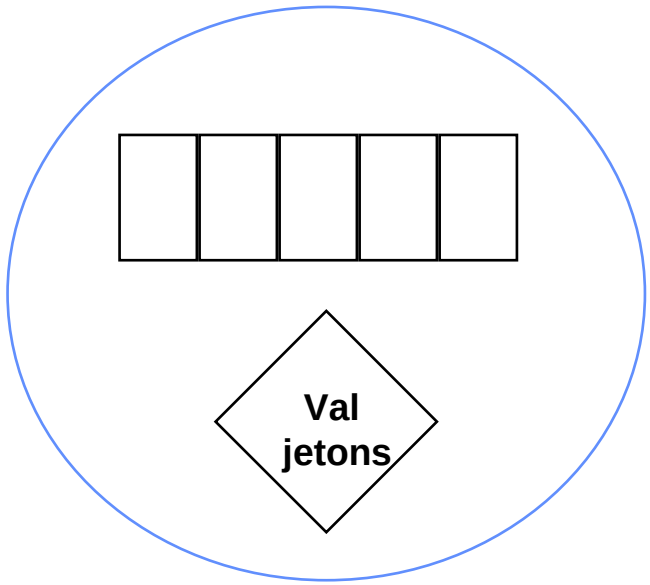
masquer_it

Sem. K := Val jetons;

Sem. L := ∅

demasquer_it

fin



Les sémaphores

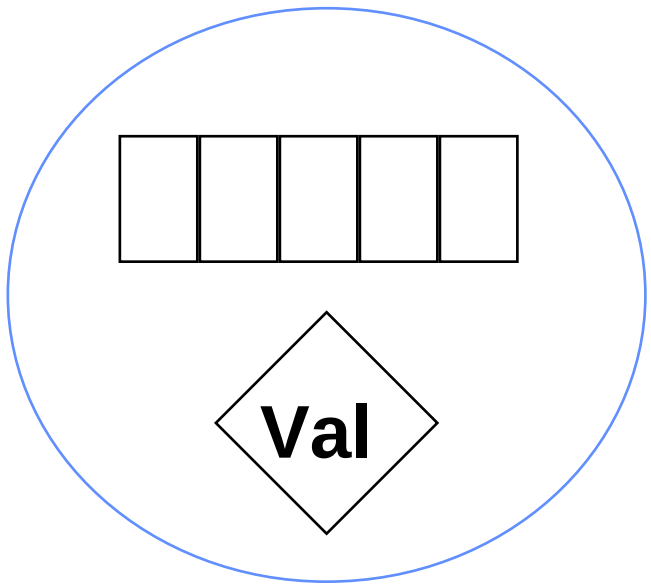
- **Opération Init (Val, Sem)**

Init (Val, Sem)

début

Sem. K := Val;
Sem. L := ∅

fin

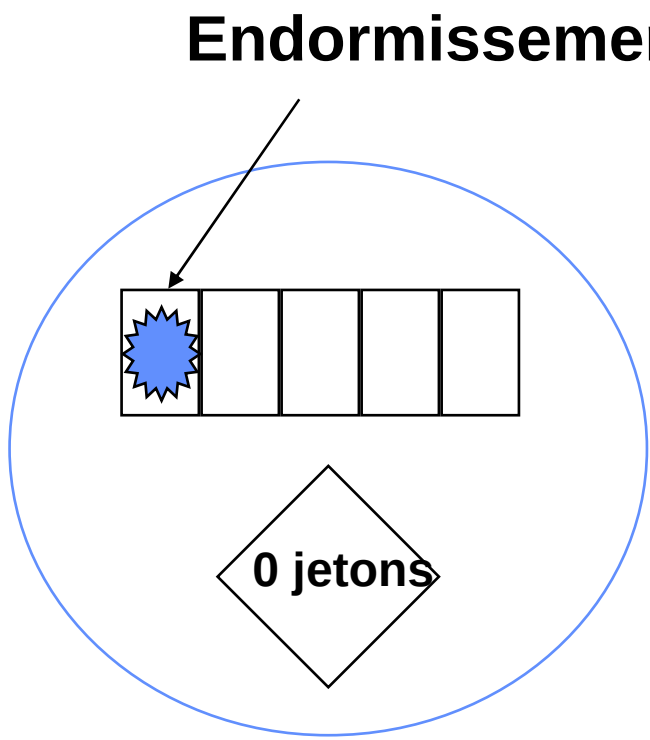


Les sémaphores

- **Opération P (Sem)**

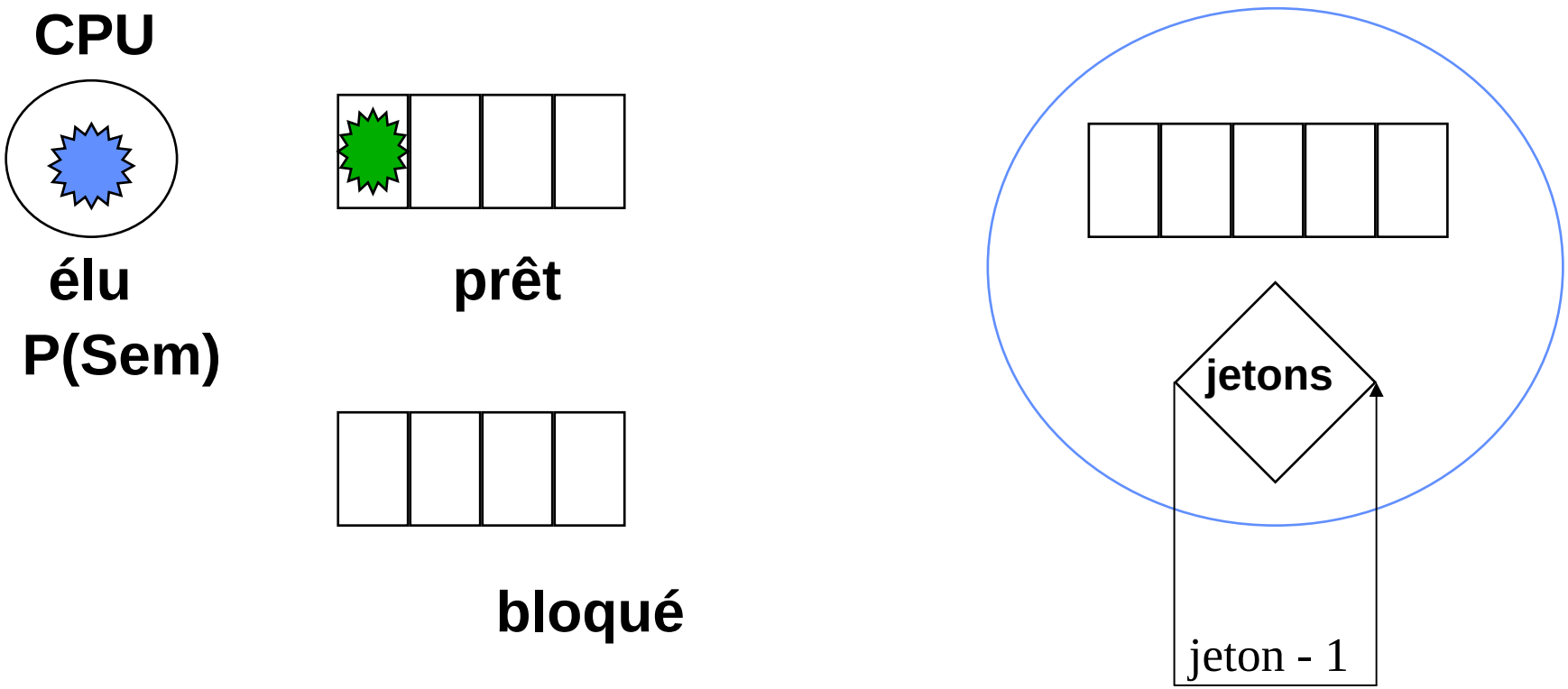
P (Sem)

début
masquer_it
Si il reste un jeton
alors
le donner à ce processus
sinon
ajouter ce processus à Sem.L
bloquer ce processus
fsi
demasquer_it
fin



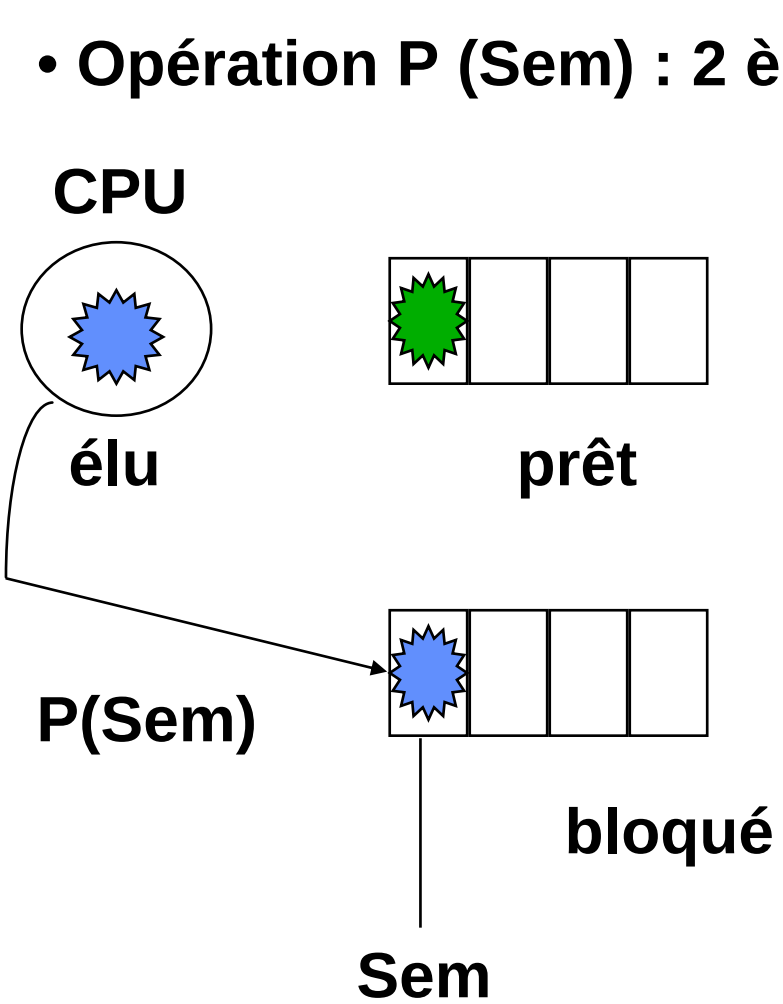
Les sémaphores

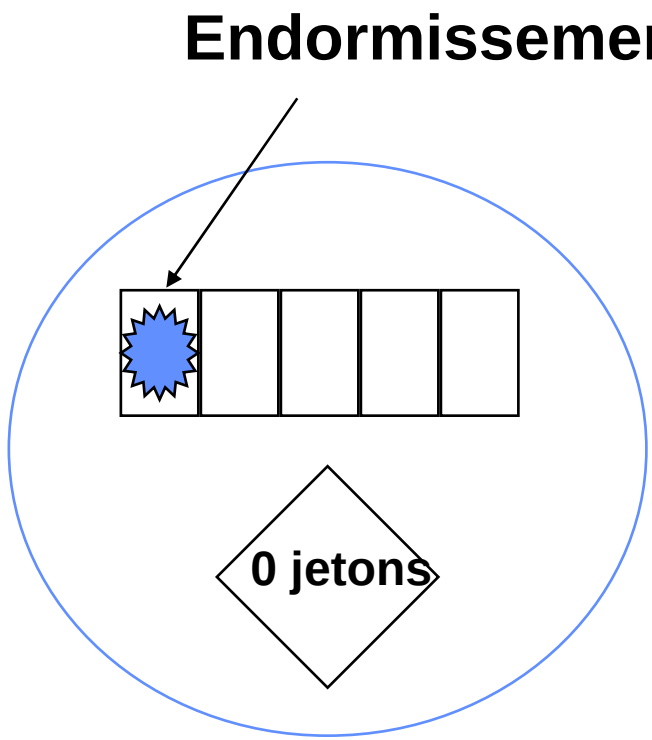
- **Opération P (Sem) : 1er cas**



Les sémaphores

- **Opération P (Sem) : 2 ème cas**



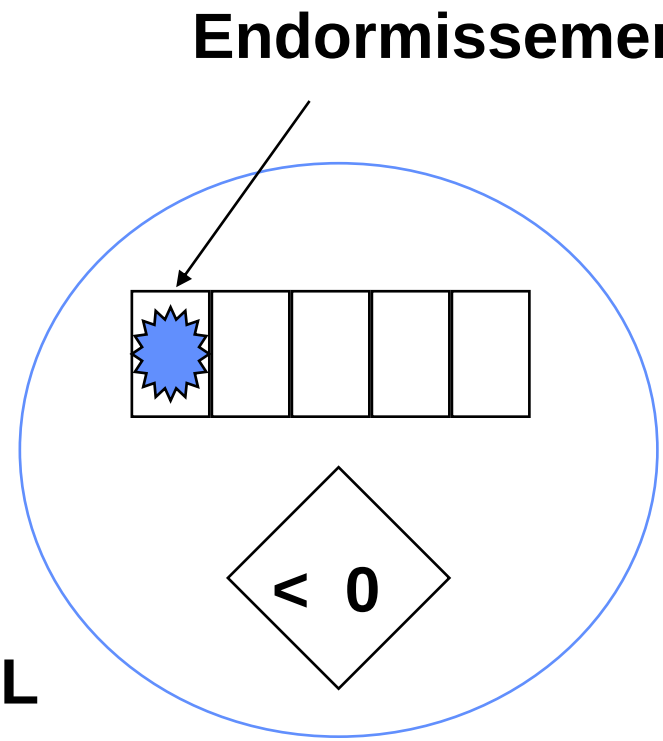


Les sémaphores

- **Opération P (Sem)**

P (Sem)

début
Sem.K := Sem.K - 1;
Si Sem.K < 0
alors
ajouter ce processus à Sem.L
endormir ce processus
fsi
fin

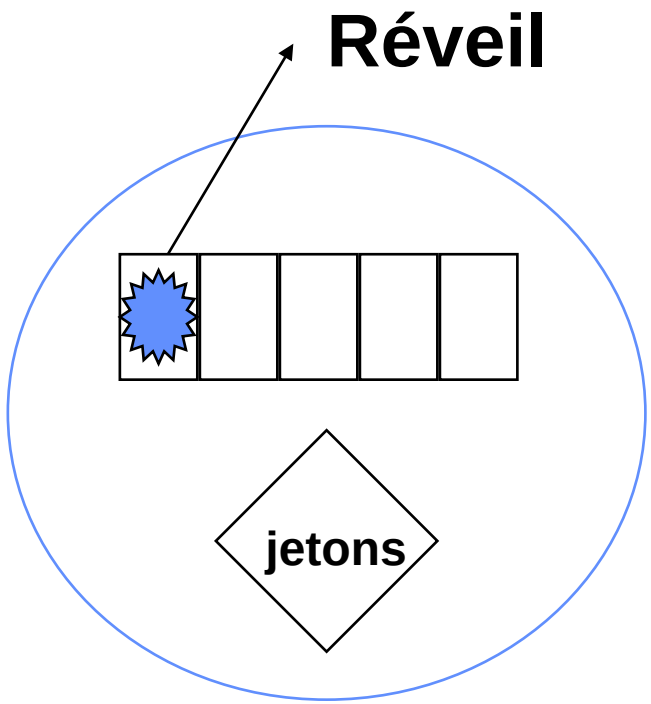


Les sémaphores

- **Opération V (Sem)**

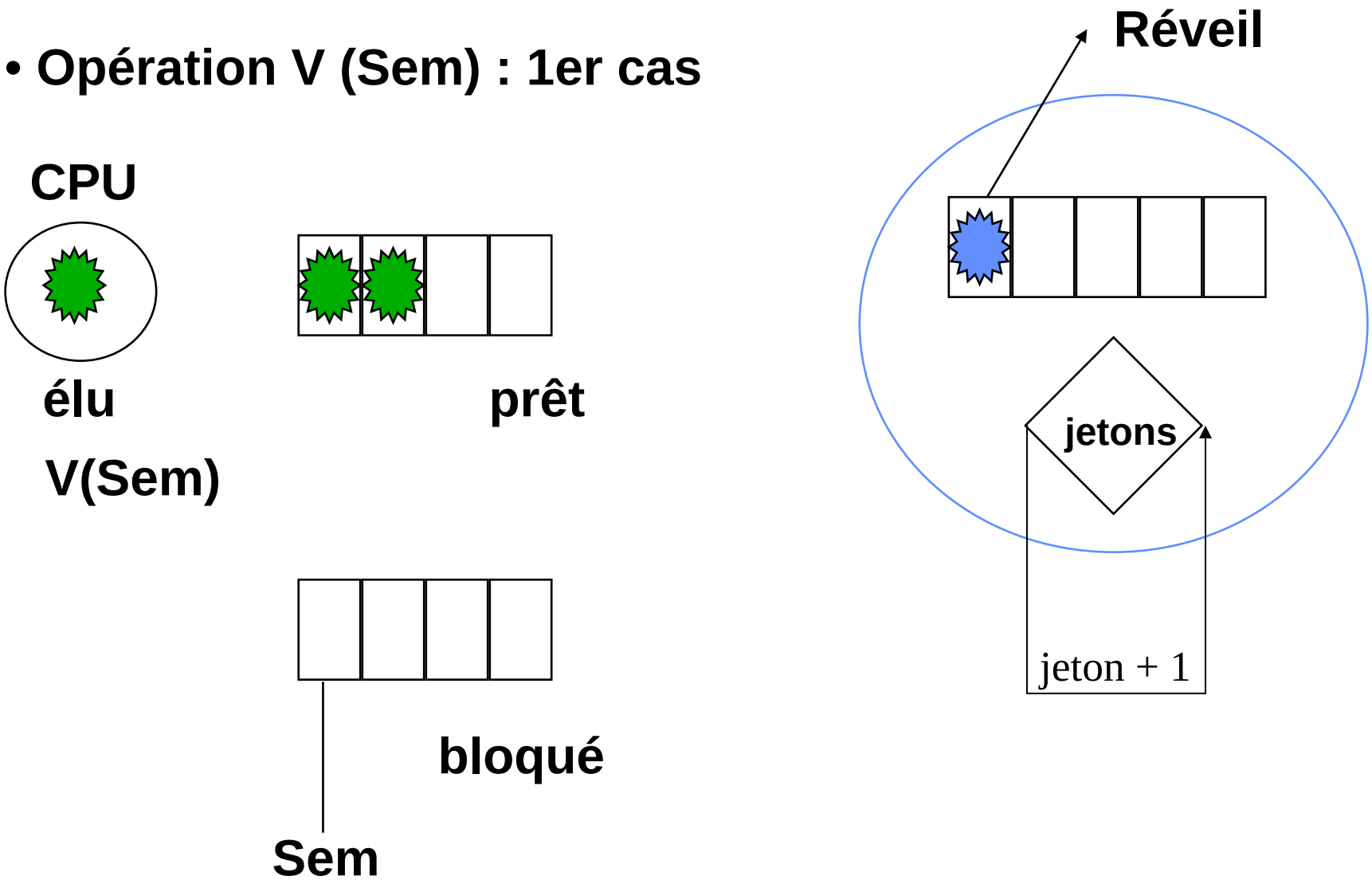
V (Sem)

début
masquer_it
Ajouter un jeton à Sem.K
Si il y a un processus en attente
de jeton dans Sem.L
alors
sortir un processus de Sem.L
lui donner un jeton
réveiller ce processus
fsi
demasquer_it
fin



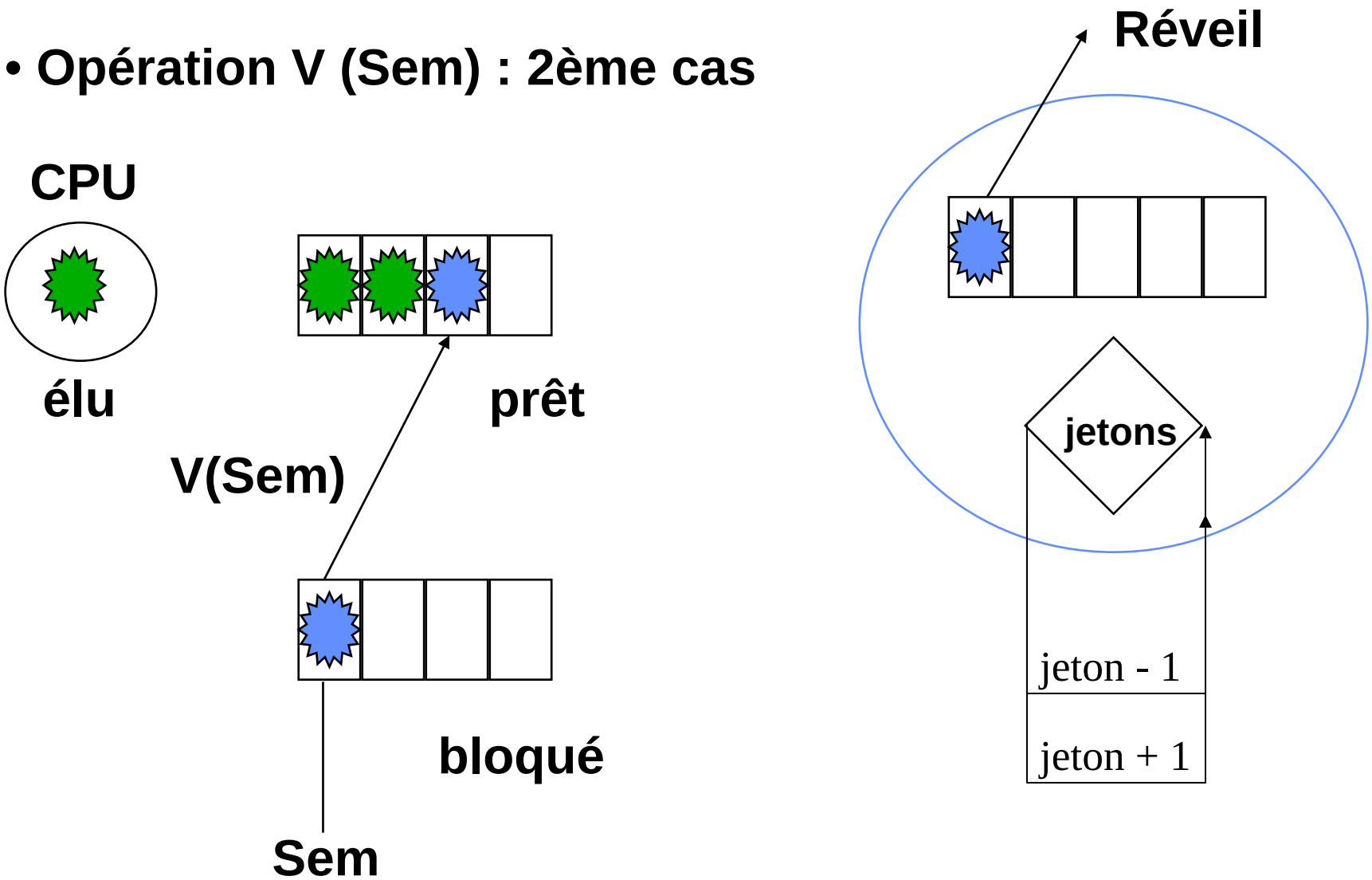
Les sémaphores

- **Opération V (Sem) : 1er cas**



Les sémaphores

- **Opération V (Sem) : 2ème cas**

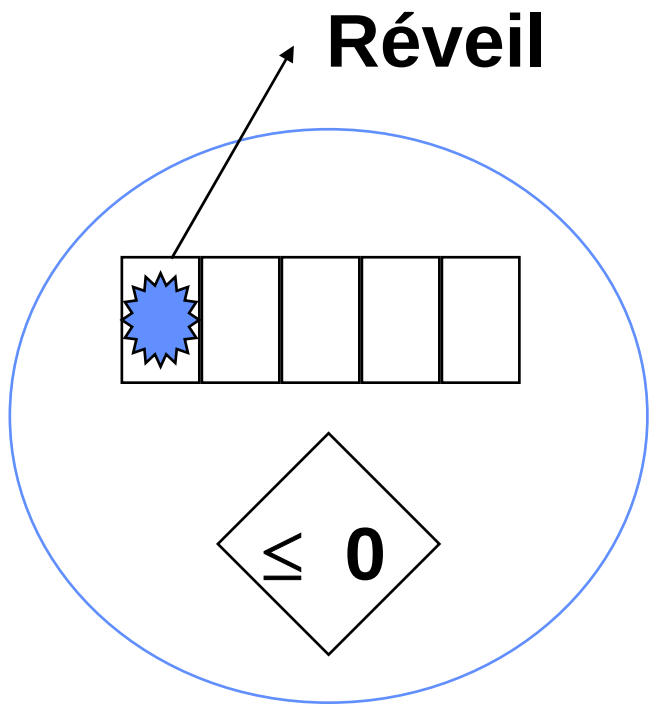


Les sémaphores

- **Opération V (Sem)**

V (Sem)

début
Sem.K := Sem.K + 1;
Si Sem.K ≤ 0
alors
sortir un processus de Sem.L
réveiller ce processus
fsi
fin



Les sémaphores

- **Signification du compteur K**

  ☞ **Si Sem.K > 0, Sem.K est le nombre d'opérations P(Sem) passantes**

P1

P(Sem)

K = 1

P2

P(Sem)

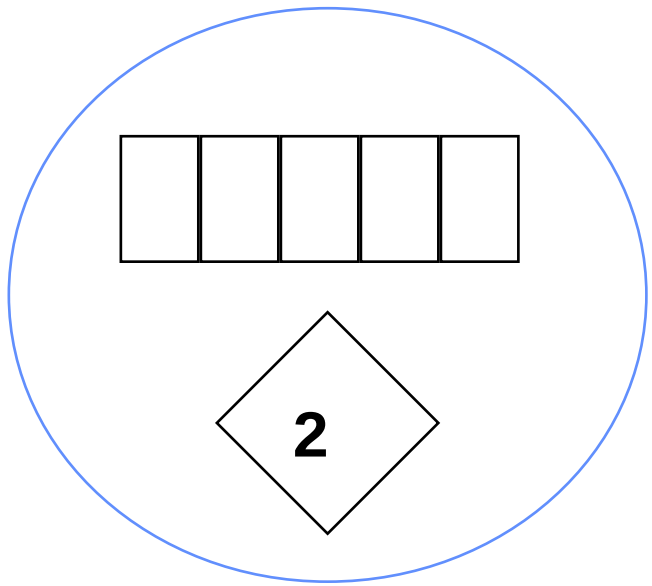
K = 0

P3

P(Sem)

K < 0

Bloqué



Les sémaphores

- **Signification du compteur K**

☞ **Si Sem.K ≤ 0, valeur_absolue(Sem.K) est le nombre de processusbloqués dans Sem.L**

P3 **P4** **P1**

P(Sem)

K = -1

Bloqué

P(Sem)

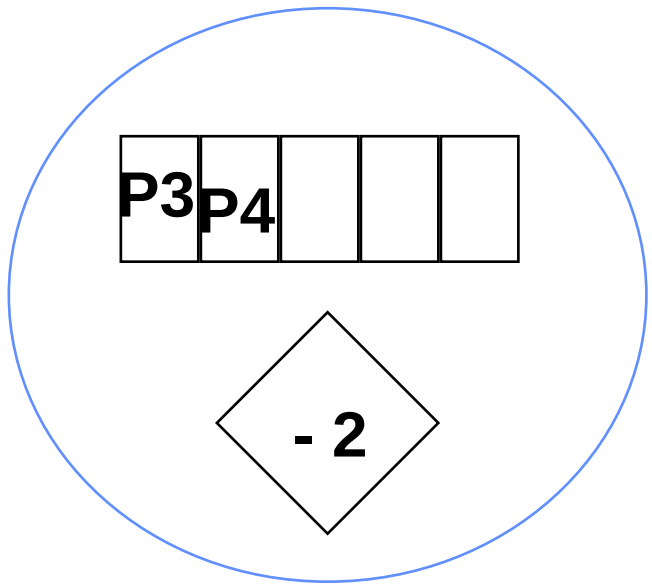
K = - 2

Bloqué

V(Sem)

K = -1

Réveil de P3



Section critique avec sémaphore

1 seul processus en section critique
=> 1 seul jeton
Sémaphore Mutex initialisé à 1

P (Mutex) — **Entrée section_critique**

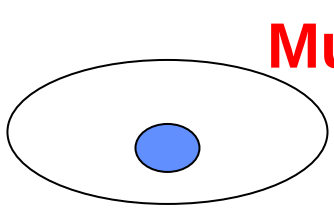
Section Critique

V (Mutex) — **Sortie section_critique**

Section critique avec sémaphore

MUTEX : sémaphore
INIT (Mutex, 1)

Réservation :
P(Mutex)
Si nb_place > 0
alors
    Réserver une place
    nb_place = nb_place - 1
fsi
V(Mutex)



Client 1

Demande Réservation

P(Mutex)

Nb_Place > 0 = 1

Nb_Place = Nb_Place - 1

V(Mutex)

Client 2

Réservation :
P(Mutex)
Si nb_place > 0
alors
Réserver une place
nb_place = nb_place - 1
fsi
V(Mutex)

Demande Réservation

P(Mutex)

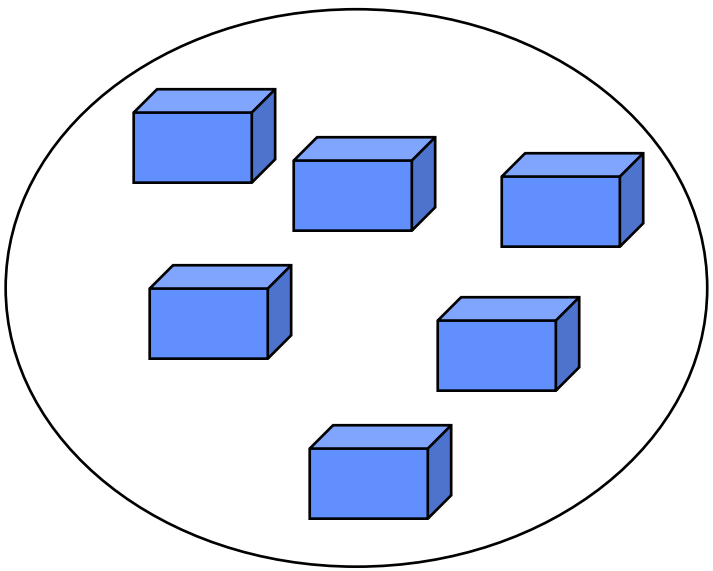
Nb_Place non accessible

Nb_Place = 0

Les sémaphores
Allocations de ressources

N ressources exclusives de même type

Sémaphore Res initialisé à N

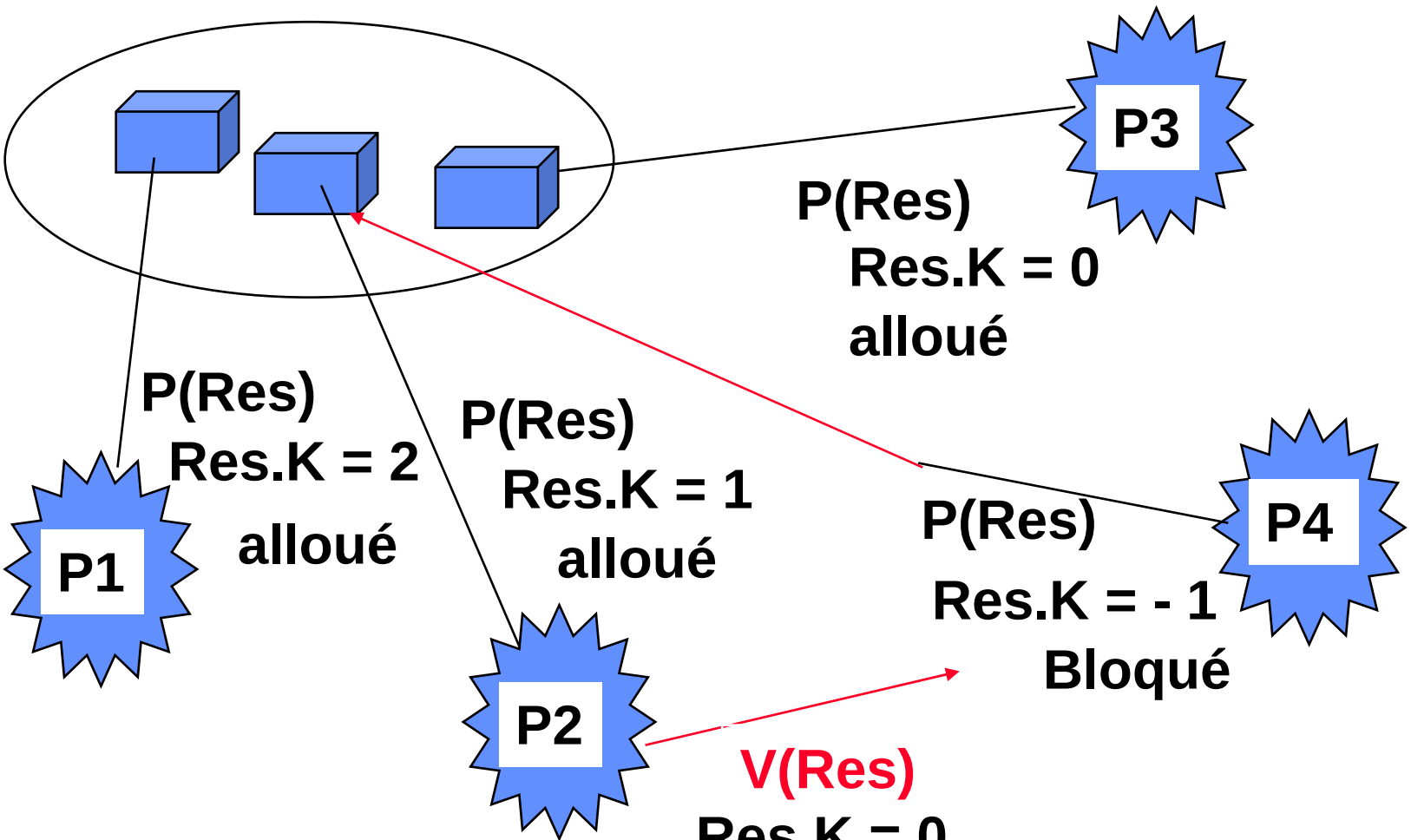


Allocation : P(Res)

Utilisation Ressource

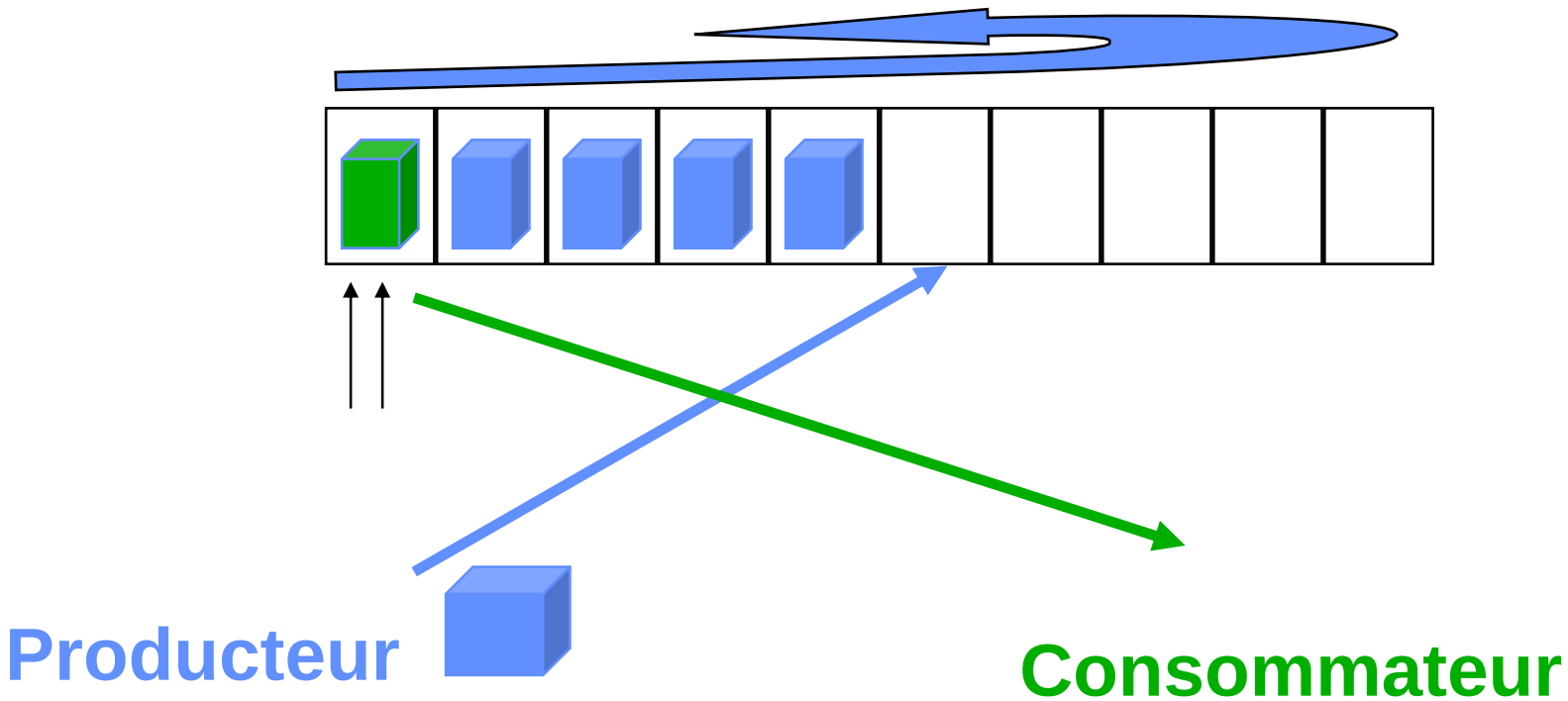
Restitution V(Res)

Les sémaphores
Allocations de ressources



Notion de synchronisation
Producteur - Consommateur

Tampon de messages géré circulairement



Notion de synchronisation Producteur-Consommateur

- **Producteur**

Si ***il y a au moins une case libre***

alors

déposer le message

prévenir le consommateur

sinon

attendre

fsi

- **Consommateur**

Si ***il y a au moins une case pleine***

alors

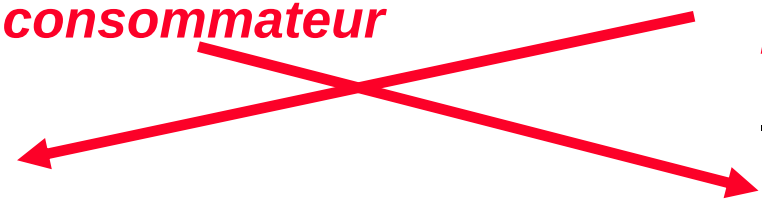
prendre le message

prévenir le producteur

sinon

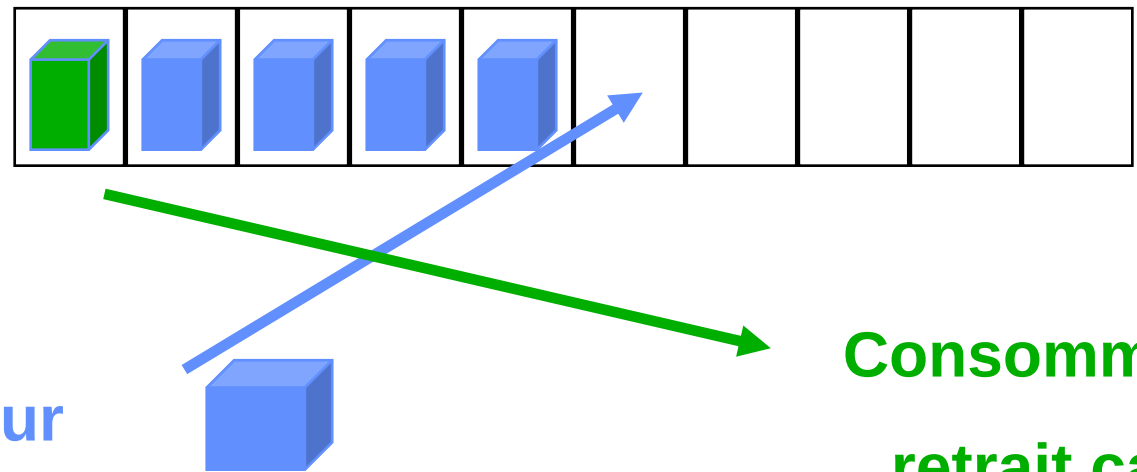
attendre

fsi



Les sémaphores
Producteur-Consommateur

Tampon de N messages



Producteur

dépot case i

Ressources cases vides
N

Sémaphore Vide

Consommateur

retrait case j

Ressources cases pleines
0

Sémaphore Plein

Les sémaphores
Producteur-Consommateur

• **Producteur**

Si ***il y a au moins***
une case libre
alors
déposer le message
prévenir le consommateur
sinon
attendre
fsi

allocation de ressources cases vides
P (Sémaphore Vide)

une ressource case pleine disponible
V (Sémaphore Plein)

Les sémaphores
Producteur-Consommateur

allocation de ressources cases pleines
P (Sémaphore Plein)

une ressource case vide disponible
V (SémaphoreVide)

- **Consommateur**

Si ***il y a au moins une case pleine***

alors

prendre le message prévenir le producteur

sinon

attendre

fsi

Sémaphore Vide initialisé à N : Init (Vide, N)

Sémaphore Plein initialisé à 0 : Init (Plein, 0)

- **Producteur**

index i de dépot : = 0

P(Vide)

déposer le message

dans T(i);

i : = i + 1 mod N;

V(Plein)

- **Consommateur**

index j de retrait := 0

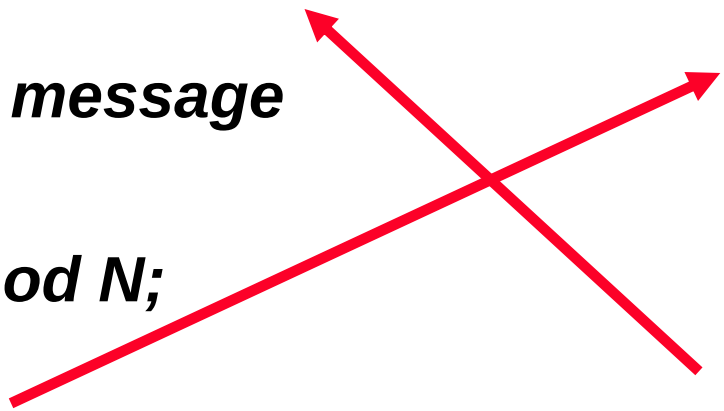
P(Plein)

retirer le message

de T(j);

j : = j + 1 mod N;

V(Vide)



Les sémaphores
Producteur-consommateur

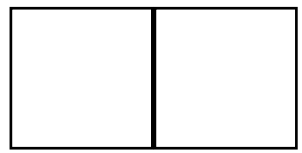
Producteur

Consommateur

P(Plein)

Plein.K = -1

Bloqué

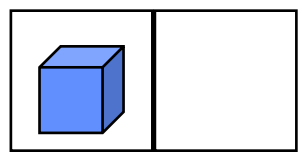


P(Vide)

Vide.K := 1

dépot

V(Plein)



Débloqué
1 case pleine

Les sémaphores
Producteur-consommateur

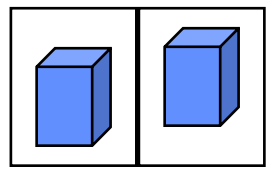
Producteur

P(Vide)
Vide.K := 0
dépot
V(Plein)

P(Vide)
Vide.K := - 1
Bloqué

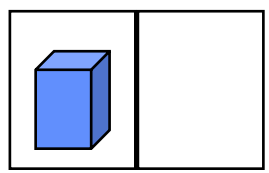
Débloqué

Consommateur

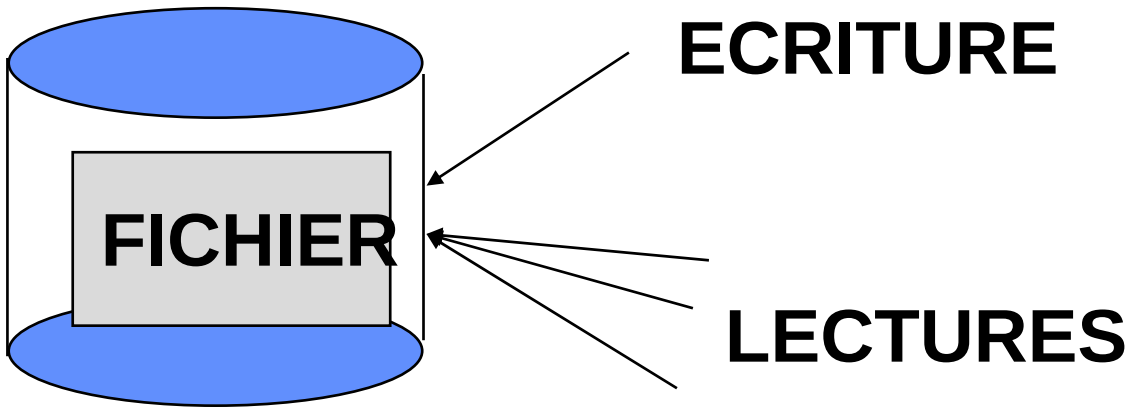


2 cases pleines

Retrait
V(Vide)



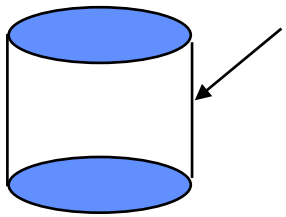
Les sémaphores
Lecteurs / Rédacteurs



- **Ecriture seule - Lectures simultanées**

☞ **Un écrivain exclut** **- les écrivains**
- les lecteurs

☞ **Un lecteur exclut** **- les écrivains**



Les sémaphores Lecteurs / Rédacteurs

- **Un écrivain exclut les écrivains et les lecteurs**

☞ **Un écrivain effectue des accès en exclusion mutuelle des autres écrivains et des lecteurs**

➔ **Sémaphore d'exclusion mutuelle <u>Accès</u> initialisé à 1**

Les sémaphores
Lecteurs / Rédacteurs

Ecrivain

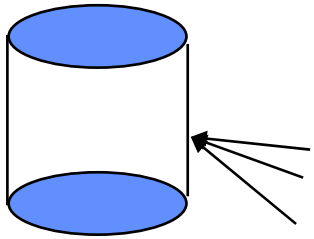
M'assurer que l'accès au fichier est libre

P(Accès)

entrer en écriture

Libérer l'accès au fichier

V(Accès)



LECTURES

Les sémaphores Lecteurs / Rédacteurs

- **Un lecteur exclut les écrivains**

  ☞ **Un premier lecteur doit s'assurer qu'il n'y a pas d'accès en écriture en cours**

  ☞ **Le dernier lecteur doit réveiller un éventuel écrivain**

➔ **NL, nombre de lecteurs courants, initialisé à 0**

Les sémaphores
Lecteurs / Rédacteurs

Lecteur

Compter un lecteur de plus
Si je suis le premier lecteur
alors
Y-a-t-il un écrivain ?
si oui, attendre
fsi

entrer en lecture

Compter un lecteur de moins
Si je suis le dernier, réveiller
un écrivain

P(Mutex)
NL : = NL + 1
NL = 1
P(Accès)
V(Mutex)

P(Mutex)
NL : = NL - 1
NL = 0
V(Mutex)
V(Accès)

Les sémaphores
Lecteurs / Rédacteurs

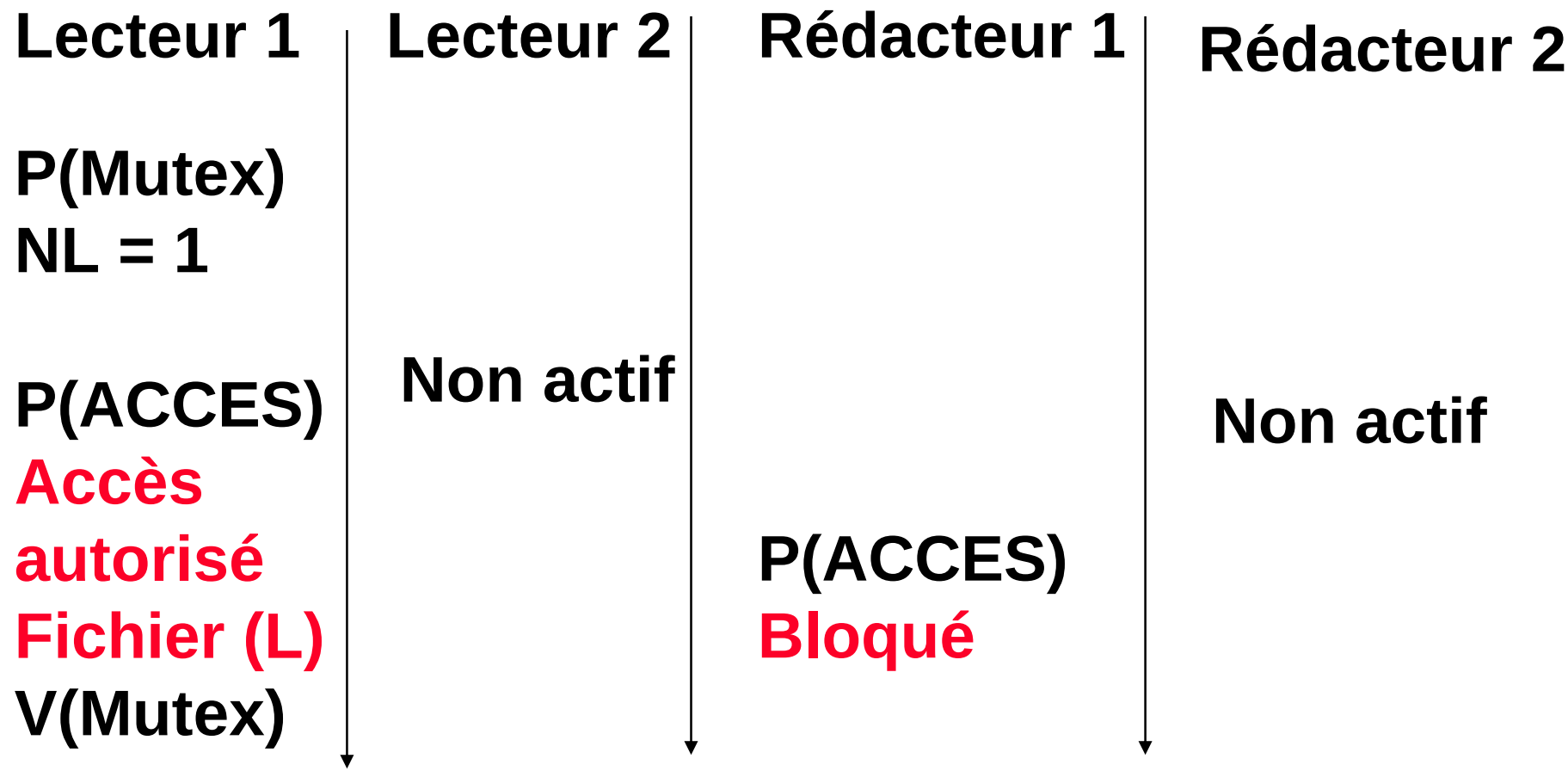
Lecteur

```
P(Mutex)
NL : = NL + 1
Si (NL = 1)
alors
        P(Accès)
fsi
V(Mutex)
```

Accès lecture

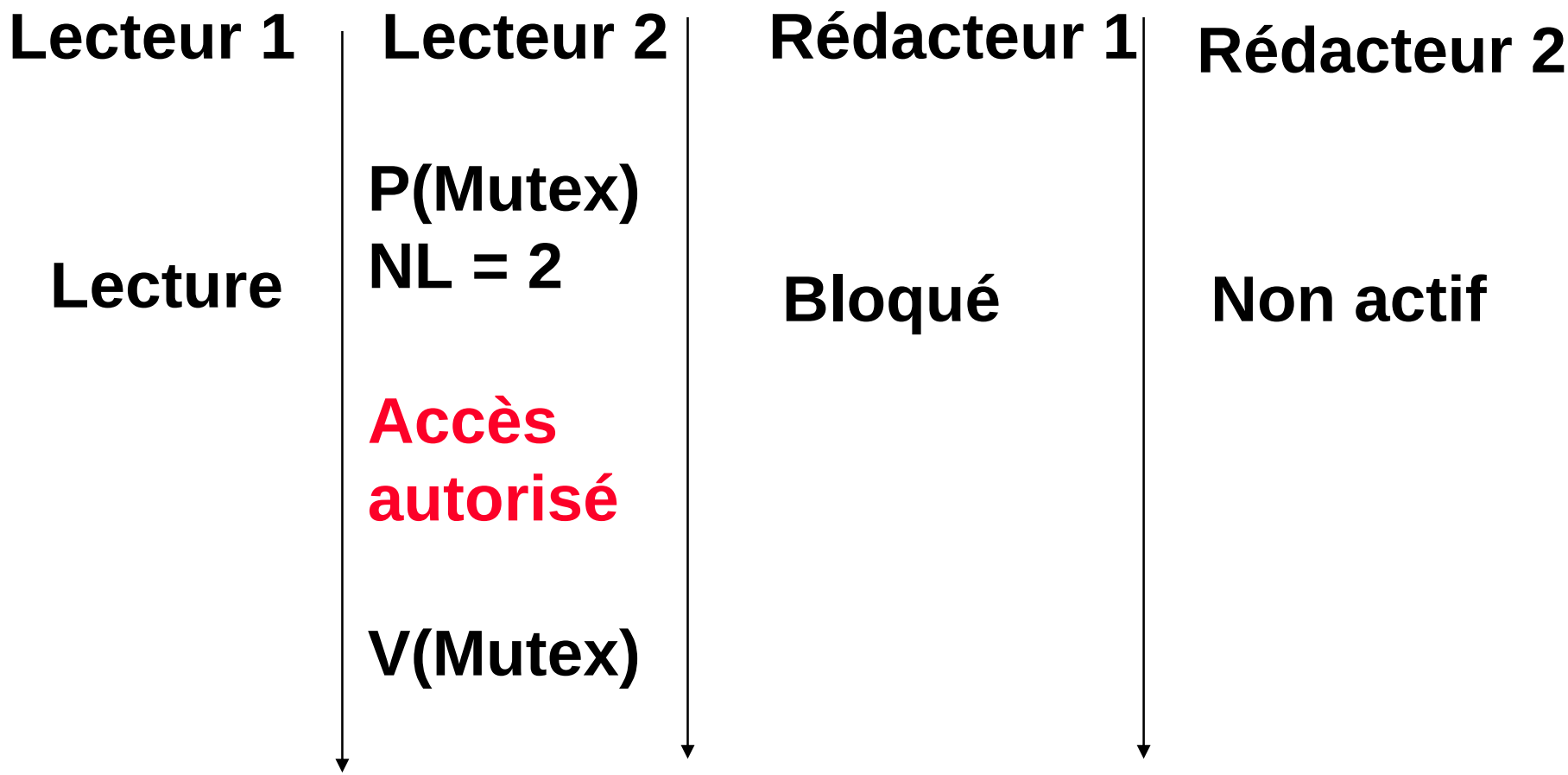
```
P(Mutex)
NL := NL - 1;
Si (NL = 0)
alors
        V(Accès)
fsi
V(Mutex)
```

Les sémaphores
Lecteurs / Rédacteurs

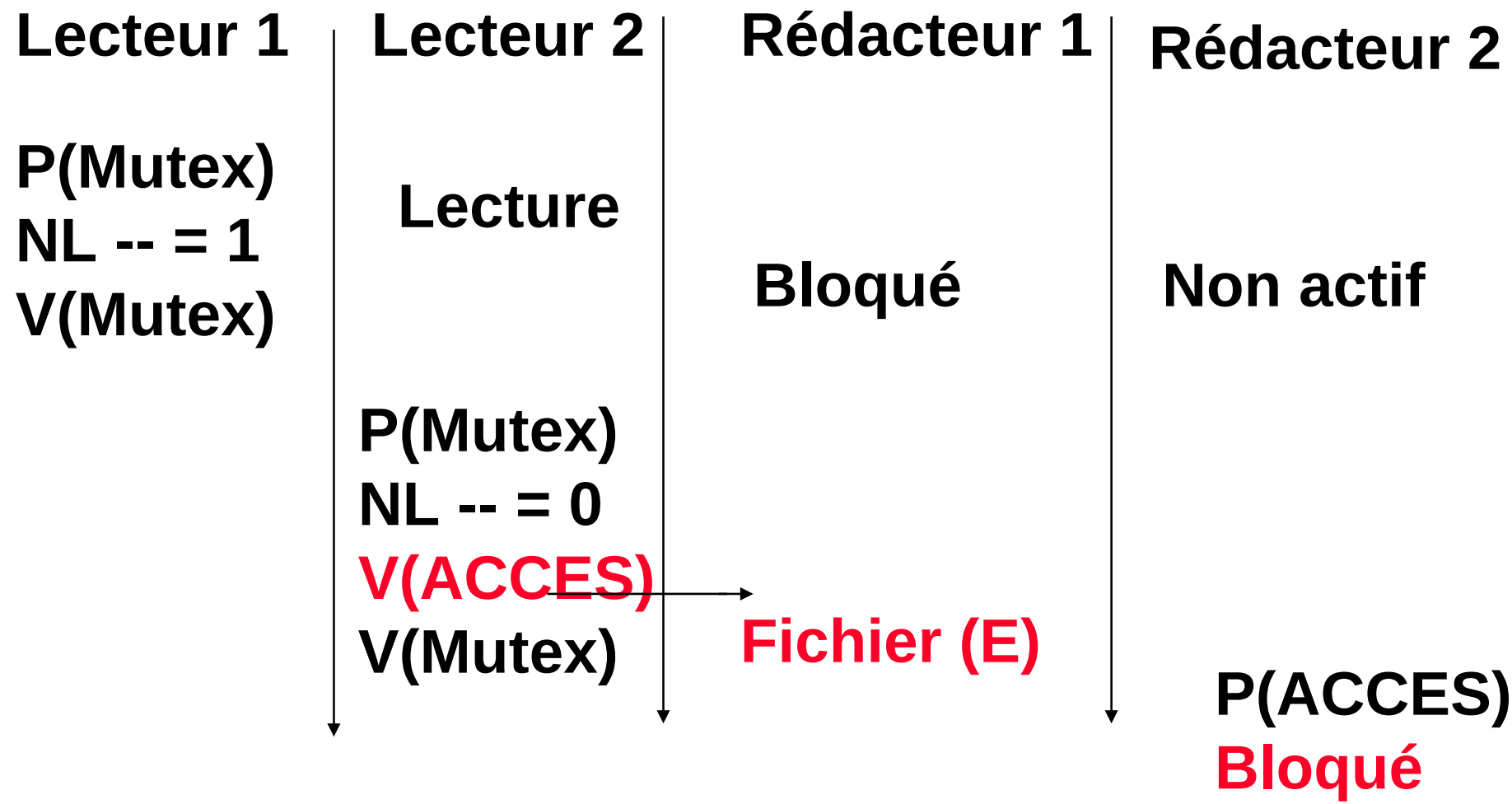


Les sémaphores

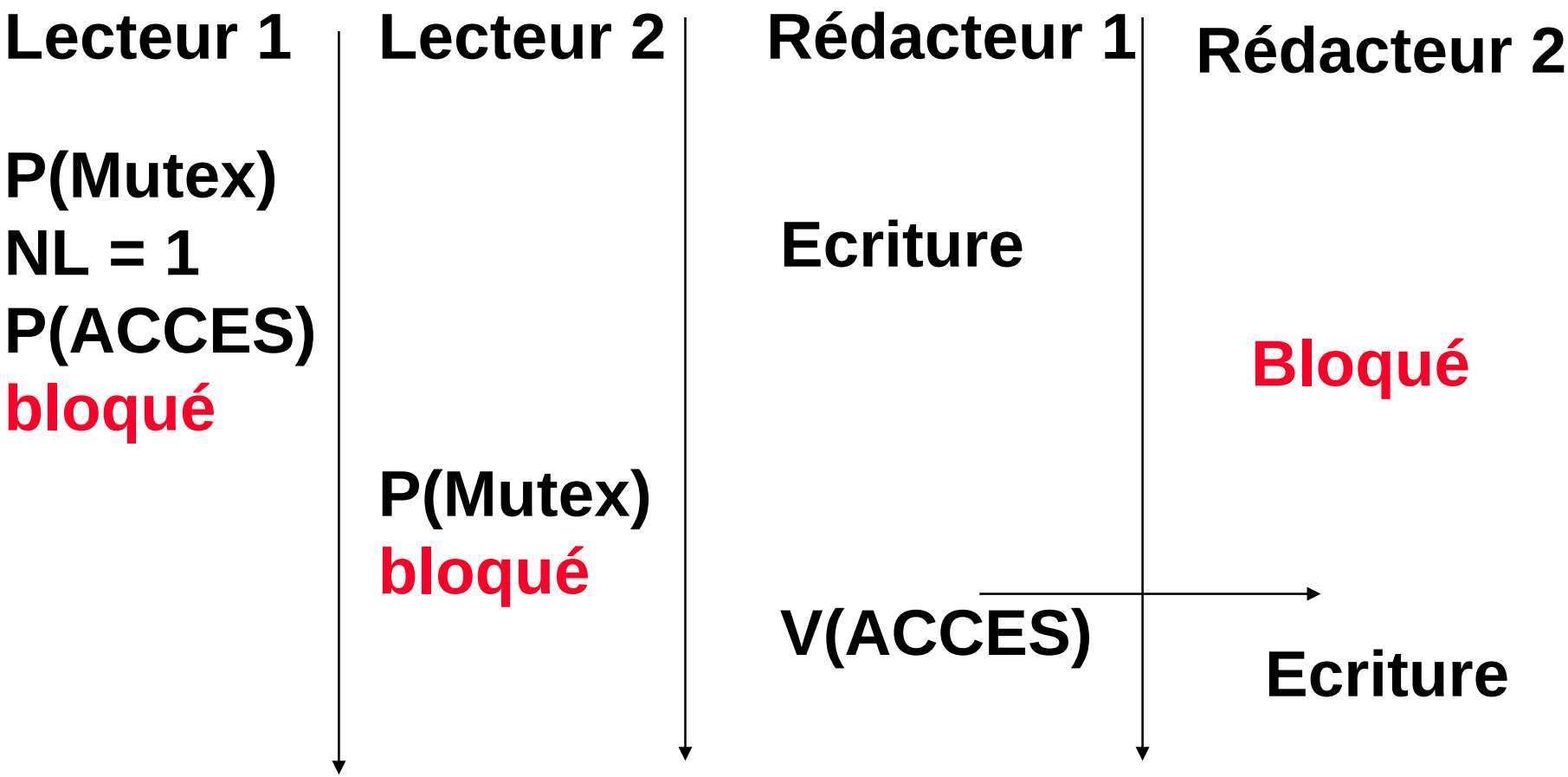
Lecteurs / Rédacteurs



Les sémaphores
Lecteurs / Rédacteurs

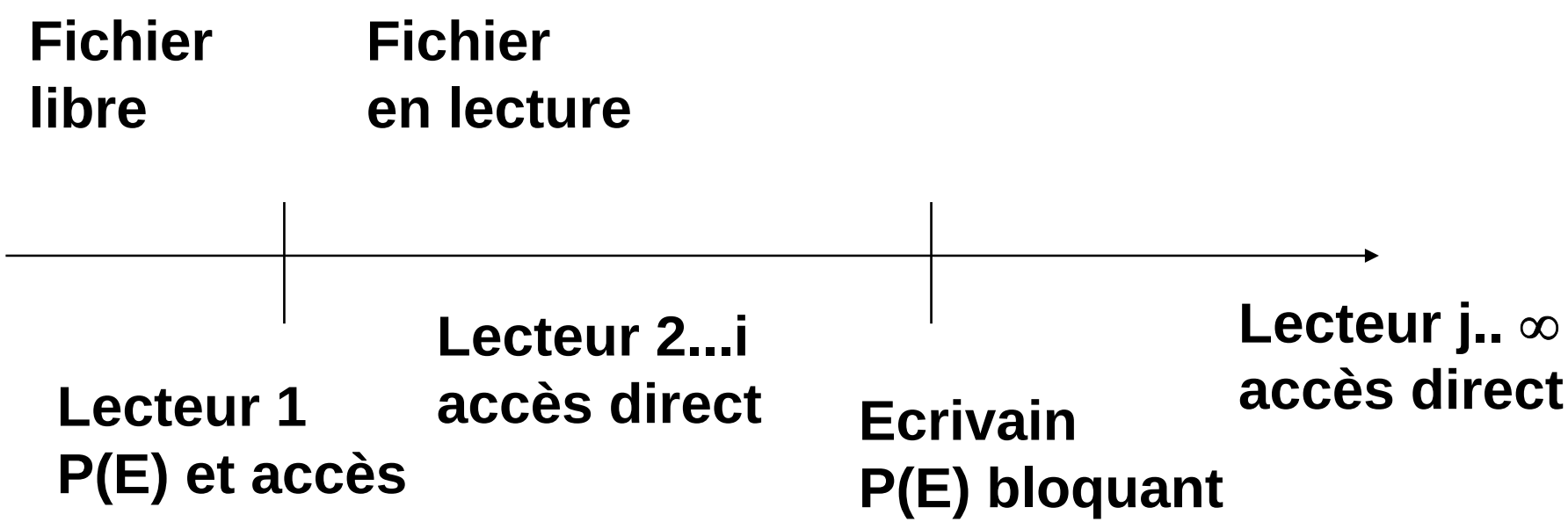


Les sémaphores
Lecteurs / Rédacteurs



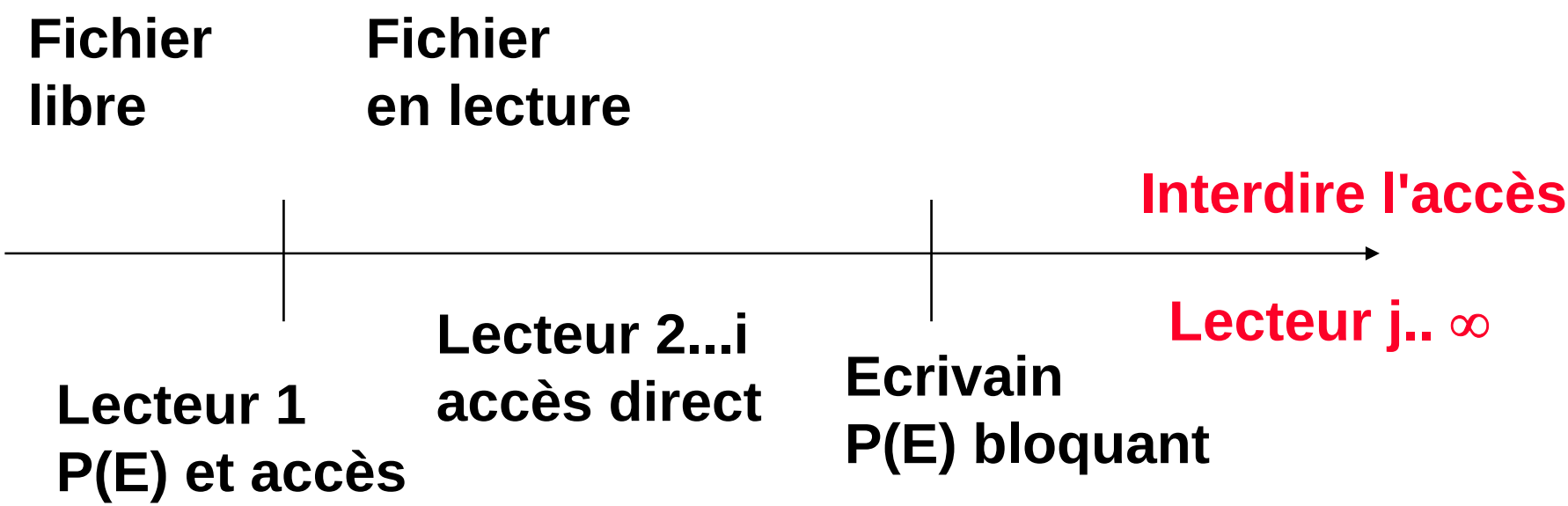
Les sémaphores
Lecteurs / Rédacteurs

- **Coalition des lecteurs contre les écrivains**



Les sémaphores
Lecteurs / Rédacteurs

- **Solution à la coalition**



Synchronisation et communication entre processus

Sémaphores Linux

Sémaphores Linux

- Famille des IPCs, un ensemble de sémaphores est identifié par une clef.

- Les opérations P, V et ATT (attente qu'une valeur de sémaphore soit nulle) s'effectuent sur un tableau de sémaphores, atomiquement

  ↳ L'ensemble des opérations est réalisée avant que le processus puisse poursuivre son exécution..

Sémaphores Linux

- Famille des IPCs, un ensemble de sémaphores est identifié par une clef.

- Les opérations P, V et ATT (attente qu'une valeur de sémaphore soit nulle) s'effectuent sur un tableau de sémaphores, atomiquement

  ↳ L'ensemble des opérations est réalisée avant que le processus puisse poursuivre son exécution..

Sémaphores Linux

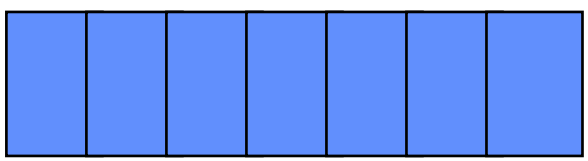
- Création et recherche d'un ensemble de sémaphore

int semget (key_t cle, int nsems, int semflg);

Création ou recherche d'un ensemble de n sémaphores identifiés par clé.

Retorune un identifiant interne de type entier

Semflg = constantes IPC_EXCL, IPC_CREAT, 0



nsems

```
Struct semaphore {
atomic_t count ; /* compteur */
int sleepers; /* nombre de processus endormis */
Wait_queue_head_t wait; /* file d'attente */ }
```

Sémaphores Linux

- Opérations sur les sémaphores

int semop (int semid, struct sembuf *ops, unsigned nsops);

Réalisation d'un ensemble d'opérations (nsops) décrite chacune dans une structure sembuf sur l'ensemble de sémaphore semid.

```
struct sembuf {
    unsigned short sem_num;/* numéro du sémaphore dans le tableau */
    short sem_op; /* opération à réaliser sur le sémaphore */
    short sem_flg; /* options */
     };
```

– si sem_op est négatif, l'opération à réaliser est une opération P;
– si sem_op est positif, l'opération à réaliser est une opération V;
– si sem_op est nul, l'opération à réaliser est une opération ATT.

Sémaphores Linux

- Initialiser un sémaphore

int semctl (int semid, int semnum, int cmd, union semun arg);

semctl (semid, 0, SETVAL, 3) initialisation à la valeur 3 du sémaphore 0 dans l'ensemble désigné par l'identifiant semid.

- Détruire un ensemble de sémaphores

int semctl (int semid, 0, IPC_RMID, 0);

```
#include <stdio.h>
#include <pthread.h>
#include <sys/ipc.h>
#include <sys/sem.h>

int i, nb_place;
int semid;
struct sembuf operation;

void reservation()
{
/* opération P */
operation.sem_num = 0;
operation.sem_op = -1;
operation.sem_flg = 0;
semop (semid, &operation, 1);
nb_place = nb_place - 1;
/* opération V */
operation.sem_num = 0;
operation.sem_op =  1;
operation.sem_flg = 0;
semop (semid, &operation, 1);
}
```

```
main()
{ pthread_t num_thread[3];


/* création d'un sémaphore initialisé à
la valeur 1 */
semid = semget (12, 1,
                IPC_CREAT|IPC_EXCL|0600);
semctl (semid, 0, SETVAL, 1);


for(i=0; i<3; i++) {
pthread_create(&num_thread[i], NULL,
(void *(*)())reservation, NULL);
pthread_join(num_thread, NULL);
semctl (semid, 0, IPC_RMID, 0)
}
```

Synchronisation et communication entre processus

Interblocage et coalition

Notion de ressources

- Définitions
  - Une ressource désigne toute entité dont a besoin un processus pour s'exécuter.
    - Ressource matérielle (processeur, périphérique)
    - Ressource logicielle (variable)
  - Une ressource est caractérisée
    - par un état : libre /occupée
    - par son nombre de points d'accès (nombre de processus pouvant l'utiliser en même temps)

Notion de ressources

- Utilisation d'une ressource par un processus

  – Trois étapes : Allocation
    Utilisation
    Restitution

  – Les phases d'allocation et de restitution doivent assurer que le ressource est utilisée conformément à son nombre de points d'accès

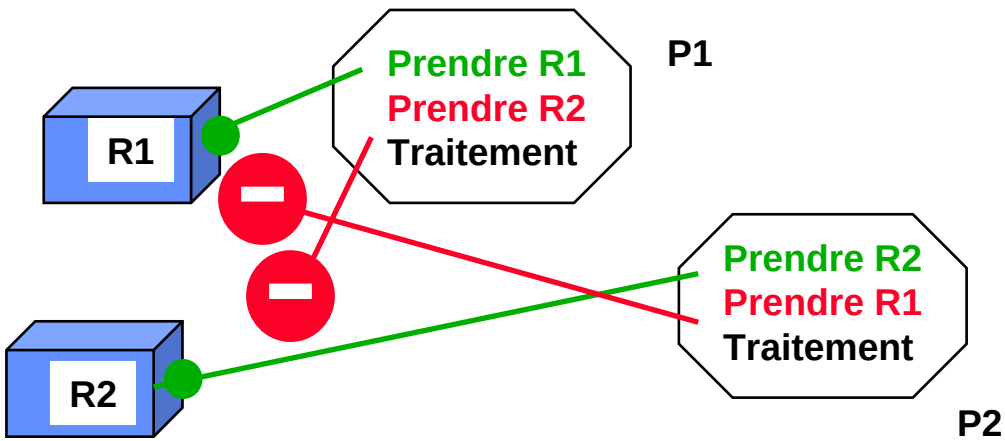
    - ressource critique à un seul point d'accès

Interblocage, Famine et Coalition

- **Interblocage**

Ensemble de n processus attendant chacun une ressource déjà possédée que par un autre processus de l'ensemble

R1 et R2 à un seul point d'accès



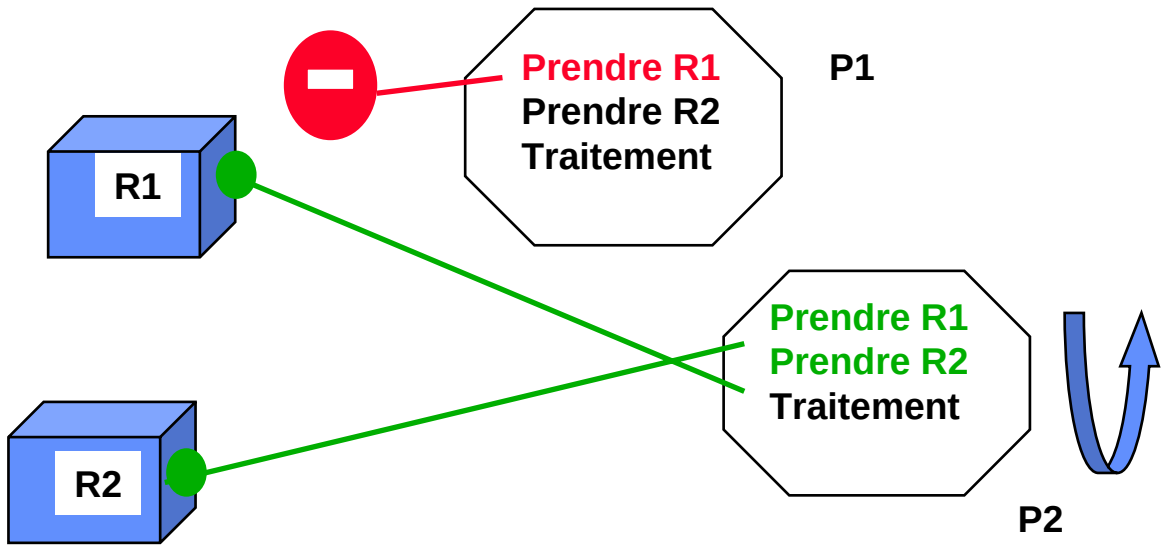
☞ **Aucun processus ne peut poursuivre son exécution**

Attente Infinie

Interblocage, Famine et Coalition

- **Coalition**

Ensemble de n processus monopolisant les ressources au détriment de p autres processus



☞ **Famine**

☞ **Attente finie mais indéfinie**

Un exemple d'interblocage

- CLIENT

/* ouverture du tube tube1 en écriture */

tub1 = open ("tube1", O_WRONLY); -- en attente de l'ouverture en lecture de tube1

/* ouverture du tube tube2 en lecture */

tub2 = open ("tube2", O_RDONLY);

-

- SERVEUR

/*ouverture du tube 2 en écriture */

tub2 = open ("tube2", O_WRONLY); -- en attente de l'ouverture en lecture de tube2

/* ouverture du tube 1 en lecture */

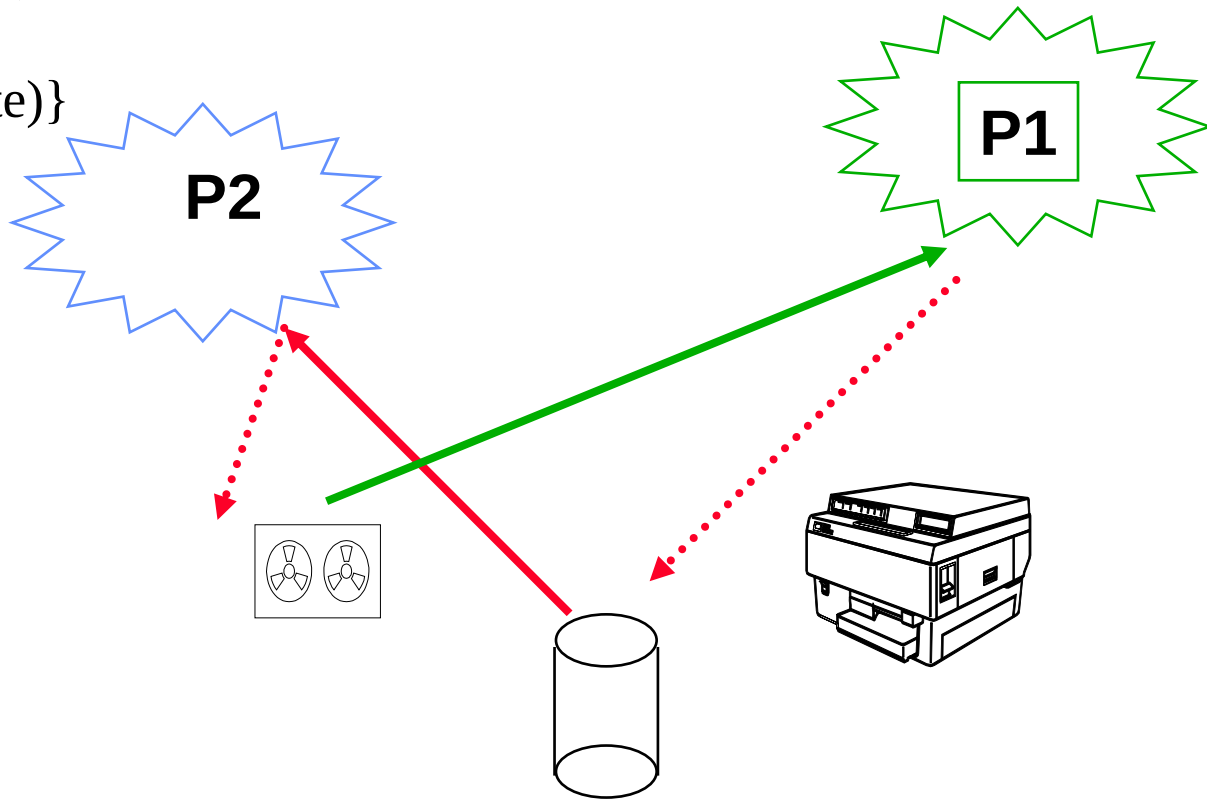
tub1 = open ("tube1", O_RDONLY);

Conditions nécessaires à l'obtention d'un interblocage

- **Exclusion mutuelle**
  - **Une ressource au moins doit se trouver dans un mode non partageable**
- **Occupation et attente**
  - **Un processus au moins occupant une ressource attend d'acquérir des ressources supplémentaires détenues par d'autres processus**
- **Pas de réquisition**
  - **Les ressources sont libérées sur seule volonté des processus les détenant**
- **Attente circulaire**

Attente circulaire

```
P1 {                    P2 {
P(bande)                P(disque)
P(disque)               P(bande)
 …..                    …..
V(bande)                V(bande)
V(disque)               V(disque)
P(imprimante)           P(imprimante)
….
V(imprimante)}          V(imprimante)}
```



P1 : bande
P2 : disque
P2 : attente bande
P1 : attente disque

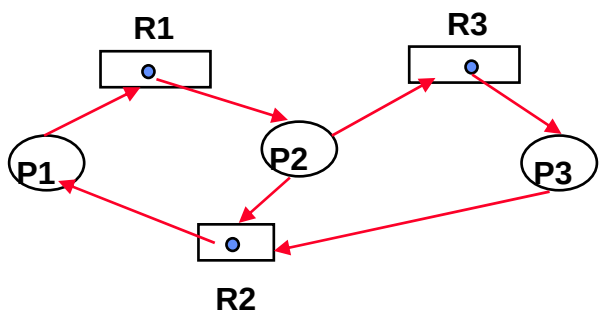
Méthodes de traitement des interblocages

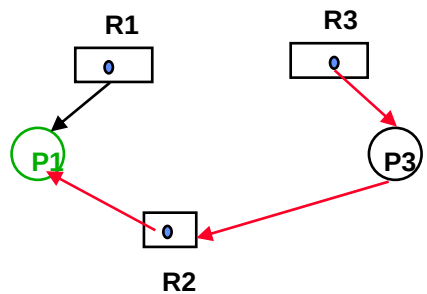
- **Permettre l'interblocage et le corriger**
  - ☞ **Politique de guérison**
- **Ne pas permettre l'interblocage**
  - ☞ **Politique de prévention ou d'évitement**
- **Ignorer le problème**
  - ☞ **Politique de l'Autruche (cf Unix)**

Politiques de guérison

- **Le système maintient un graphe représentant l'allocation des ressources et les attentes des processus**
- **Régulièrement, le système parcourt le graphe à la recherche de cycles**
- **Si un cycle est découvert, celui-ci est cassé en avortant les processus en interblocage appartenant au cycle**

☞ **coûteux**





Politiques de prévention

- **Assurer qu'au moins une des conditions nécessaires ne peut avoir lieu**

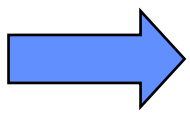
  - ☞ **Exclusion mutuelle : difficile**
  - ☞ **Occupation et attente : demander les ressources en une seule fois**
  - ☞ **Pas de réquisition : difficile**
  - ☞ **Attente circulaire : ordre total sur l'ordre de demandes de ressources**

Politiques de prévention

- **<u>Occupation et attente</u> : demander les ressources en une seule fois**

```
P1 {
P(bande)
P(disque)
 …..
V(bande)
V(disque)
P(imprimante)
….
V(imprimante)}
```

```
P2 {
P(disque)
P(bande)
…..
V(bande)
V(disque)
P(imprimante)
….
V(imprimante)}
```



```
P2 {
P(disque, bande, imprimante)
…..
V(bande)
V(disque)
V(imprimante)}
```

```
P1 {
P(bande, imprimante, disque)
….
V(bande)
V(disque)
V(imprimante)}
```

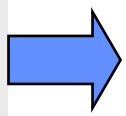
↳ Mauvaise utilisation des ressources

Politiques de prévention

- **<u>Attente circulaire</u> : ordre total sur l'ordre de demandes de ressources**
- **Unité de bandes avant le disque et avant l'imprimante**

```
P1 {
P(bande)
P(disque)
 …..
V(bande)
V(disque)
P(imprimante)
….
V(imprimante)}
```

```
P2 {
P(disque)
P(bande)
…..
V(bande)
V(disque)
P(imprimante)
….
V(imprimante)}
```



```
P1 {
P(bande)
P(disque)
 …..
V(disque)
V(bande)
P(imprimante)
….
V(imprimante)}
```

```
P2 {
P(bande)
P(disque)
…..
V(disque)
V(bande)
P(imprimante)
….
V(imprimante)}
```

Politiques d'évitement

- **Examen dynamique de l'état d'allocation des ressources afin d'éviter l'attente circulaire**

  ☞ **A chaque demande d'allocation de ressource, le système détermine si accepter cette allocation peut on non mener le système à l'interblocage, ie l'état du système reste-t-il *sain*.**

  ☞**si oui, l'allocation est refusée.**

  ☞ **Vision pessimiste**

Politiques d'évitement

	Besoins maximaux	Ressources allouées	Demandes restantes
P1	10	5	5
P2	4	2	2
P3	9	2	7

12 exemplaires de ressources au total
Le nombre de ressources disponibles est égal à 3.
La séquence d'exécution < P2, P1, P3 > est saine:
– satisfaction de P2, ressources disponibles = 1;
– restitution des ressources par P2, ressources disponibles = 5;
– satisfaction de P1, ressources disponibles = 0;
– restitution des ressources par P1, ressources disponibles = 10;
– satisfaction de P3, ressources disponibles = 3;
– restitution des ressources par P3, ressources disponibles = 12.

Politiques d'évitement

	Besoins maximaux	Ressources allouées	Demandes restantes
P1	10	5	5
P2	4	2	2
P3	9	**3**	6

12 exemplaires de ressources au total
Le nombre de ressources disponibles est égal à 2
L'état devient malsain et aucune séquence d'exécution incluant les trois processus ne peut être construite. Ici, seul P2 peut être satisfait:
– satisfaction de P2, ressources disponibles = 0;
– restitution des ressources par P2, ressources disponibles = 4.
Maintenant, ni P3, ni P2 ne peuvent être satisfaits.

Politique de l'Autruche

- **Prétendre que les interblocages ne se produisent jamais et ne rien prévoir**

  ☞ **Un interblocage peut se produire et n'est pas détecté**

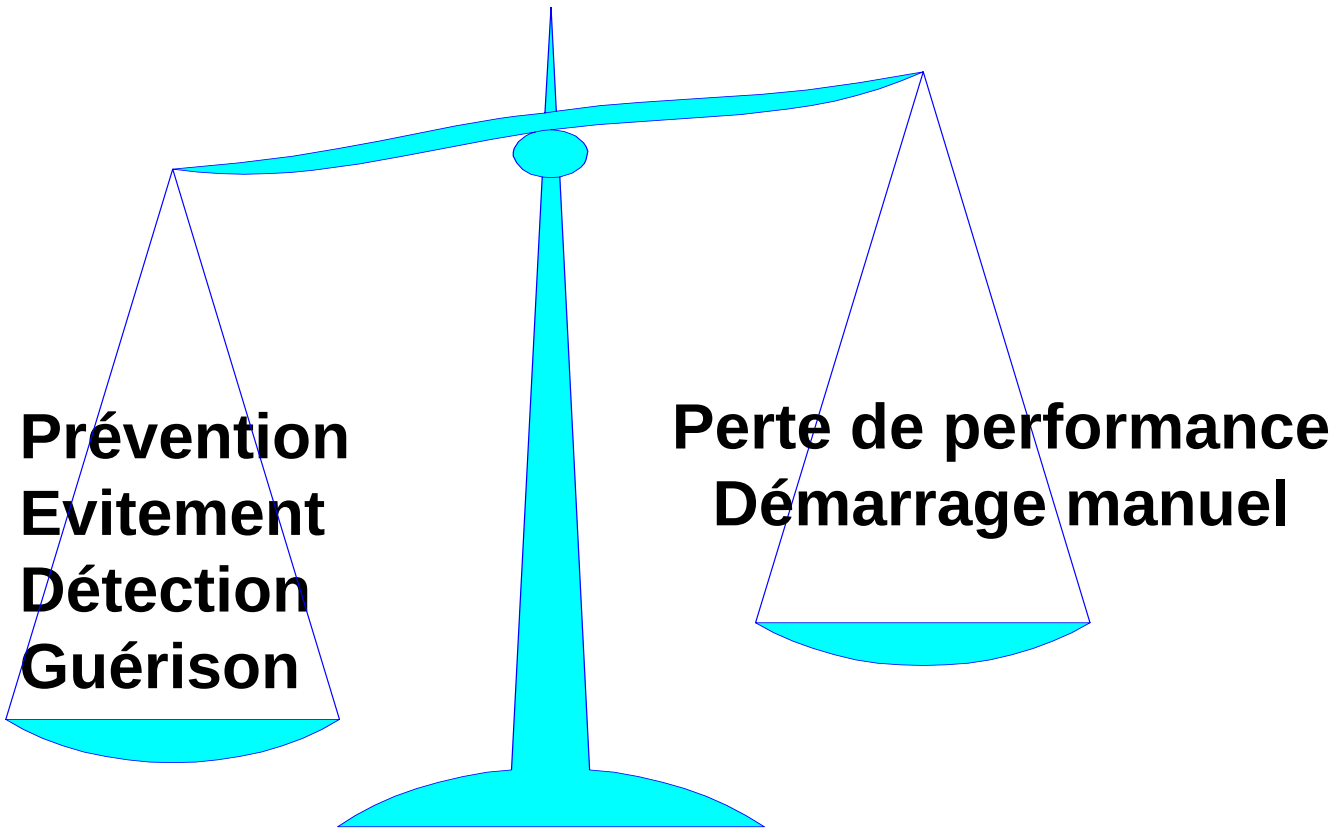
  ☞ **Détérioration des performances jusqu'à arrêt complet du système**

  ☞ **Rédémarrage manuel du système**

Politique de l'Autruche

- **Justification de ce choix**

Fréquence de l'interblocage



Interblocage, coalition et famine

- **L'interblocage est une situation pour laquelle n processus sont en attente de ressources allouées à p autres processus eux-mêmes en attente de ressources allouées aux n processus : il se produit une attente circulaire entre les processus et aucun d'entre eux ne peut poursuivre son exécution : l'attente est infinie.**

- **Il existe quatre grandes familles de solutions vis-à-vis de l'interblocage : la politique de l'Autruche, le politique de prévention, la politique d'évitement, la politique de guérison**

- **La coalition est une situation pour laquelle un ensemble de n processus monopolisent les ressources au détriment de p autres processus qui se trouvent en situation de famine. Les p autres processus ne peuvent pas s'exécuter : leur attente est finie mais de durée indéterminée.**