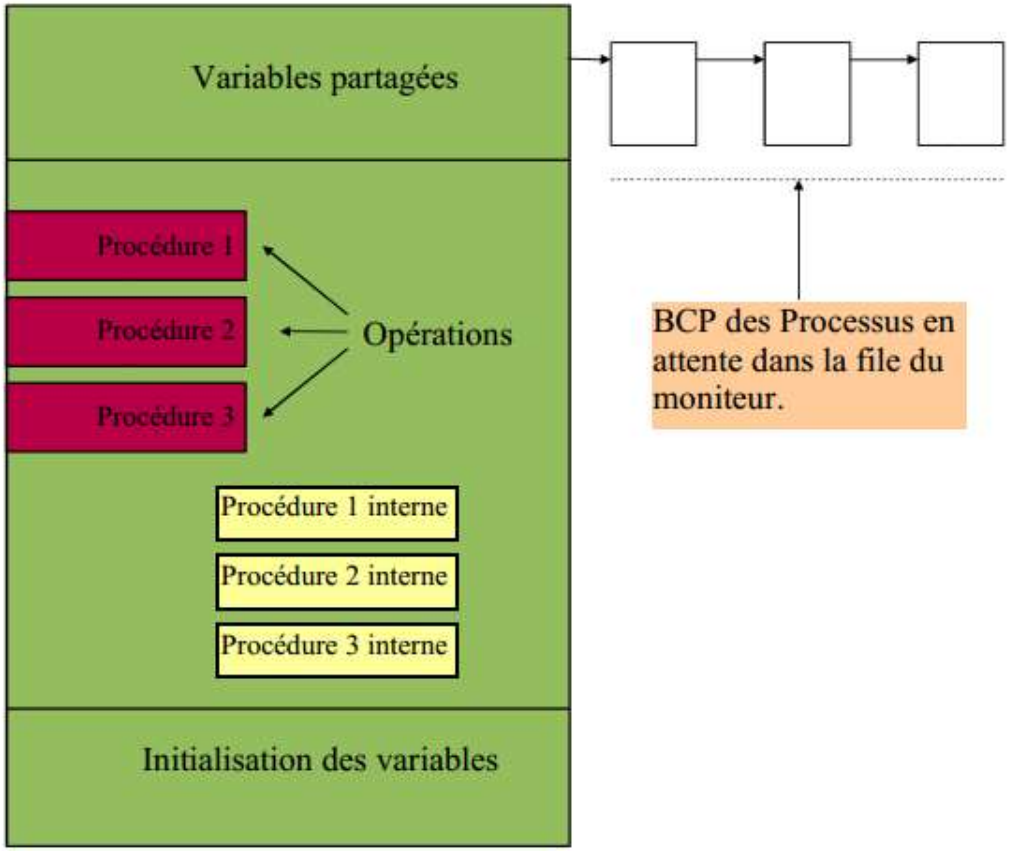
Chapitre 2 : Communication inter-processus par partage d'information : Moniteur

Imene LAHYANI

Les moniteurs (1)

- Ce sont des structures de synchronisation modulaires, qui regroupent des variables partagées par plusieurs processus, ainsi que les instructions qui les manipulent.



Les moniteurs (2)

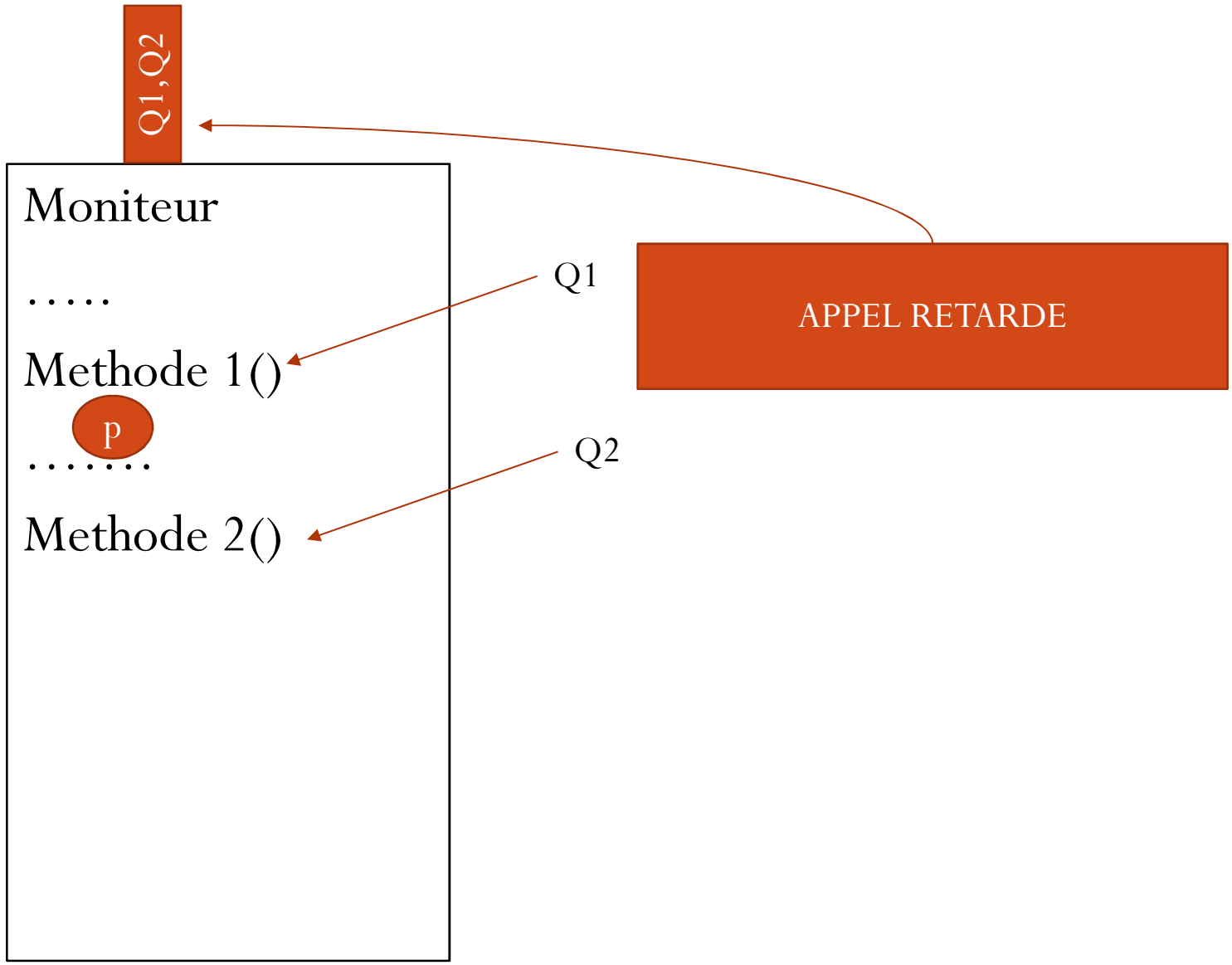
- Une variable partagée devient un objet abstrait qualifié par les procédures (méthodes) permettant de l'accéder.
- Le contrôle d'accès aux procédures ne se fait pas par le processus exécutant des sections critiques mais plutôt par le moniteur lui-même
- Il est donc nécessaire d'imposer des règles d'accès aux méthodes d'un moniteur
  - **Règle 1**: Exclusion mutuelle
  - **Règle 2**: Synchronisation entre processus
  - **Règle 3**: Priorité pour l'accès aux méthodes

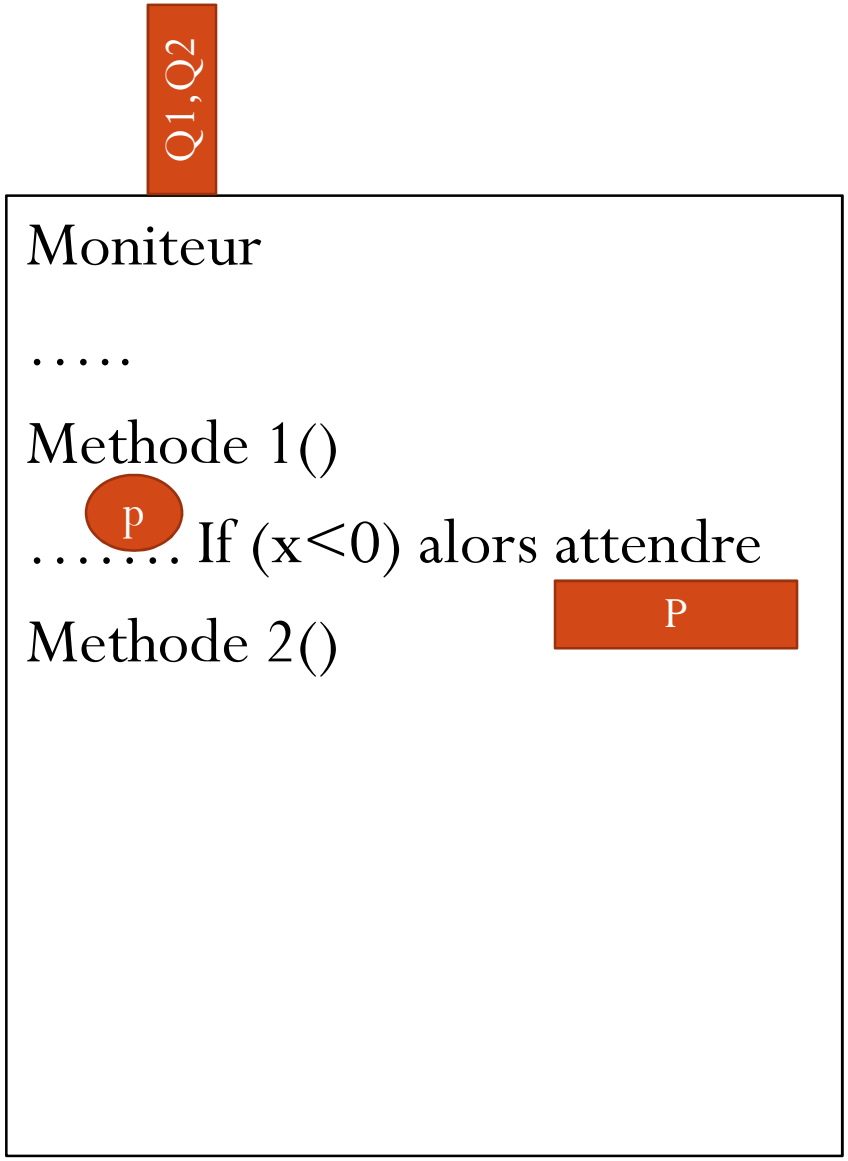
Règle 1 :Exclusion mutuelle

- L'invocation des méthodes d'un moniteur se fait en exclusion mutuelle (les méthodes sont atomiques)
  - A un instant donné, au plus un processus peut être dans le moniteur
- Exemple:
  - Pendant qu'un processus P exécute une méthode d'un moniteur, tout autre appel à cette méthode ou à une autre méthode du moniteur par un autre processus Q est retardé.
  - Le processus Q est mis en attente, il est repris lorsque l'exécution de la méthode est terminée.
- Le moniteur maintient une file d'attente de type FIFO, qui contient les processus bloqués en entrée du moniteur

Règle 2: Synchronisation entre processus

- A fin d'assurer la synchronisation entre les processus, on introduit des variables de type condition qui représentent des files d'attente de type FIFO (First In First Out).
- On distingue trois type d'opérations sur une variable c de type condition :
  - wait(c) : mettre le processus en attente sur la condition. Elle le met à la fin de la pile
  - signal(c) : redémarrer le premier processus en attente sur la condition
  - nonempty(c) : prédicat qui renvoie la valeur vrai si il y a au moins un processus en attente sur la condition.


Q1,Q2
Moniteur
.....
Methode 1()
p
.....
Methode 2()
Q1
Q2
APPEL RETARDE

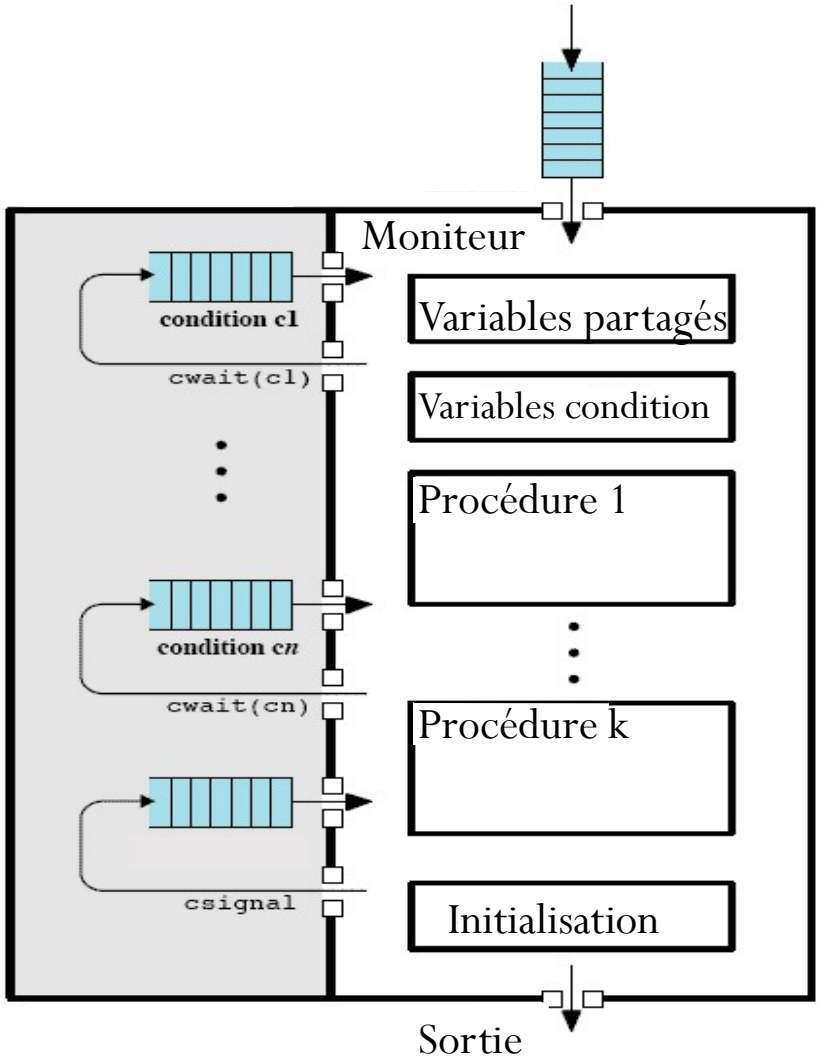


Variable de type condition

Règle 2: Synchronisation entre processus

- **Remarque** : Les opérations wait(c) et signal(c) sont semblables aux opérations utilisées sur les sémaphores, mais elles ont un comportement différent.
- Une condition n'est qu'une file d'attente, pas une file d'attente associés à une variable. Une condition ne peut donc être utilisée pour compter, contrairement à un sémaphore. Par conséquent,
  - wait(c) met toujours le processus qui l'exécute en attente,
  - signal(c) est sans effet si aucun processus n'est en attente.

Règle 2: Synchronisation entre processus



Règle 3: Priorité entre processus

- Un processus utilisant un moniteur peut être en attente soit sur une condition soit pour entrer dans le moniteur. Les règles appliquées sont les suivantes:
- Entre les processus qui souhaitent commencer l'utilisation d'une procédure du moniteur, en applique la discipline FIFO;
- Lorsqu'un processus exécute un wait(c), il libère l'accès au moniteur et permet à d'autres processus d'en exécuter les procédures;

Les moniteurs en Java

- A partir de la version 1.5, la notion de variable condition a été rajoutée ainsi que des méthodes matérialisant les verrous. => Utilisation de l'interface ReentrantLock

Exemple 1: Le corps de la méthode m est protégé par un verrou

```
import java.util.concurrent.locks.*;
class X {
 private final ReentrantLock l = new ReentrantLock(); // …
 public void m()
{ l.lock(); // verouiller
try { // … method body }
finally { l.unlock() ; // déverouiller} } }
```

Les moniteurs en Java

- Comment créer une variable de type condition?

Condition cond1 = l.newCondition();

Cond1.await() // attendre sur la condition

Cond1.signal() // libérer le premier processus dans la file en attente sur la condition

Exemple 2:

```
import java.util.concurrent.locks.*;
class test
{ private final Lock lock = new ReentrantLock();
 private final Condition cond1 = lock.newCondition();
void methode1() throws InterruptedException
{ lock.lock();
try {
…
cond1.await();
…..
} finally  { lock.unlock(); } }
 void methode2() throws InterruptedException
{ lock.lock();
try {
…
cond1.signal();
} finally { lock.unlock(); } } }
```