Chapitre 3
Threads

Gestion de processus parallèles

Threads

1. Processus et threads

Processus et threads

Ressources allouées à un processus :

mémoire, processeur, I/O (fichiers …) …

Thread : "processus léger"

+/- processus à l'intérieur d'un processus

Si plusieurs threads dans le même processus :

les threads travaillent en parallèle

Partage des ressources du processus entre les threads du processus

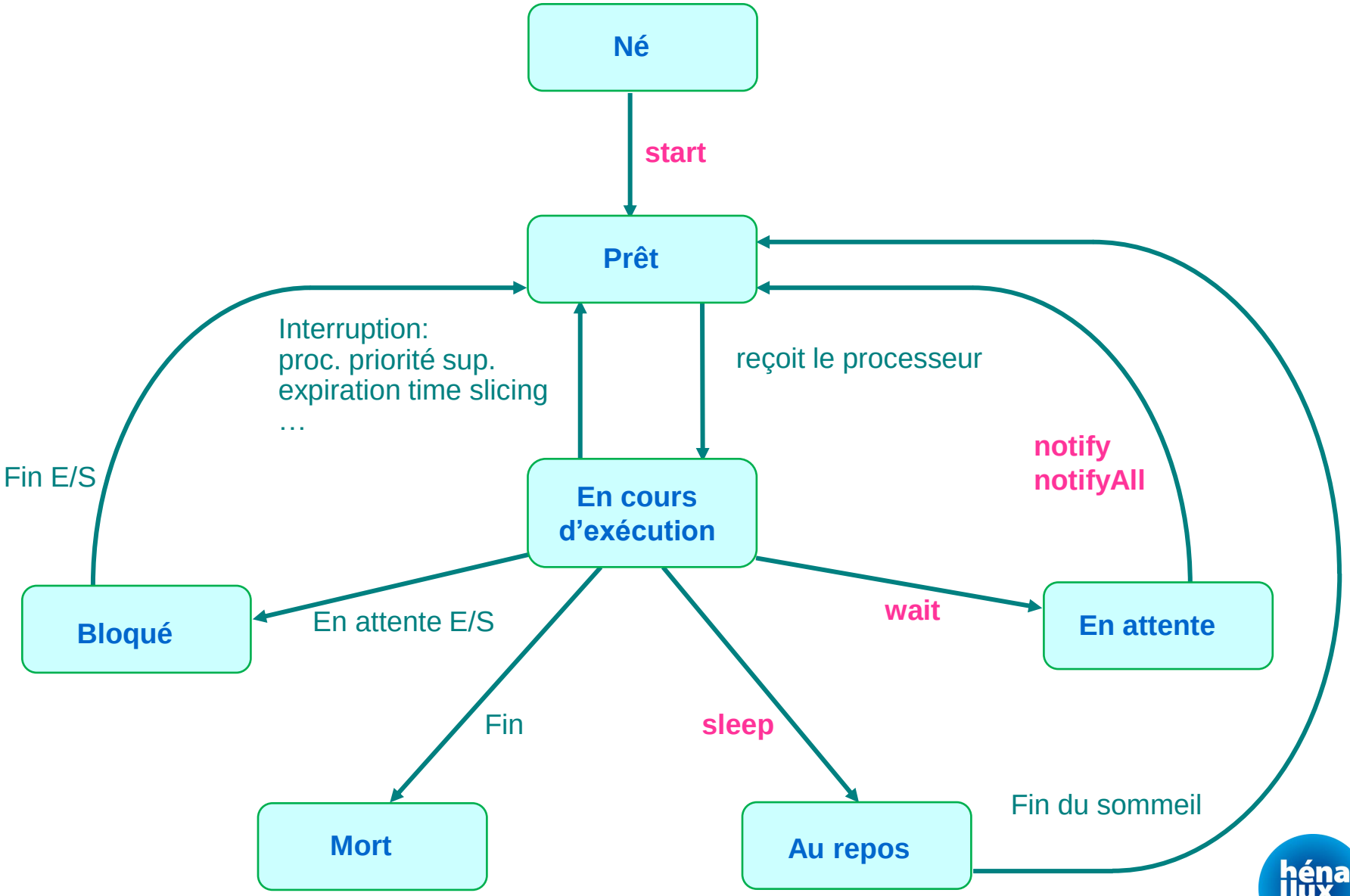
(partage même zone mémoire (espace d'adressage))

Un processus contient au moins 1 thread (cfr méthode main(…))

Ex: Garbage collector de java

Threads

1. **Processus et threads**
2. **Cycle de vie d'un thread**


Né
start
Prêt
Interruption:
proc. priorité sup.
expiration time slicing
…
reçoit le processeur
notify
notifyAll
Fin E/S
En cours
d'exécution
Bloqué
En attente E/S
wait
En attente
Fin
sleep
Mort
Au repos
Fin du sommeil

Threads

1. **Processus et threads**
2. **Cycle de vie d'un thread**
3. **Choix 1 : implements Runnable**

implements Runnable

```
class ThreadX  implements Runnable  {
  …
        public void run( )  {
        // code de la tâche que le thread doit effectuer
        }
}
```

Méthode à redéfinir

Création du thread

```
ThreadX   threadX  = new ThreadX( ) ;

Thread  thread = new Thread(threadX) ;

thread.start( ) ;
```

Appelle la méthode **run( ) de ThreadX**

Threads

1. **Processus et threads**
2. **Cycle de vie d'un thread**
3. **Choix 1 : implements Runnable**
4. **Choix 2 : extends Thread**

extends Thread

```
class ThreadX   extends Thread  {
 …
          public void run( )  {
            // code de la tâche que le thread doit effectuer
          }
}
```

Méthode à redéfinir

Création du thread

```
ThreadX   threadX  = new ThreadX( );
```

```
threadX.start( );
```

Appelle la méthode **run( ) de ThreadX**

Threads

1. **Processus et threads**
2. **Cycle de vie d'un thread**
3. **Choix 1 : implements Runnable**
4. **Choix 2 : extends Thread**
5. **Méthode sleep**

Méthode sleep

```
public class ThreadX  extends Thread  {
   …
    public void run ( ) {
       …
       while (true)  {
          try  {Thread.sleep(1000) ;
                …        // code de la tâche à exécuter par le thread
          }
          catch (Exception e) {
                …
          }
      }
   }
}
```

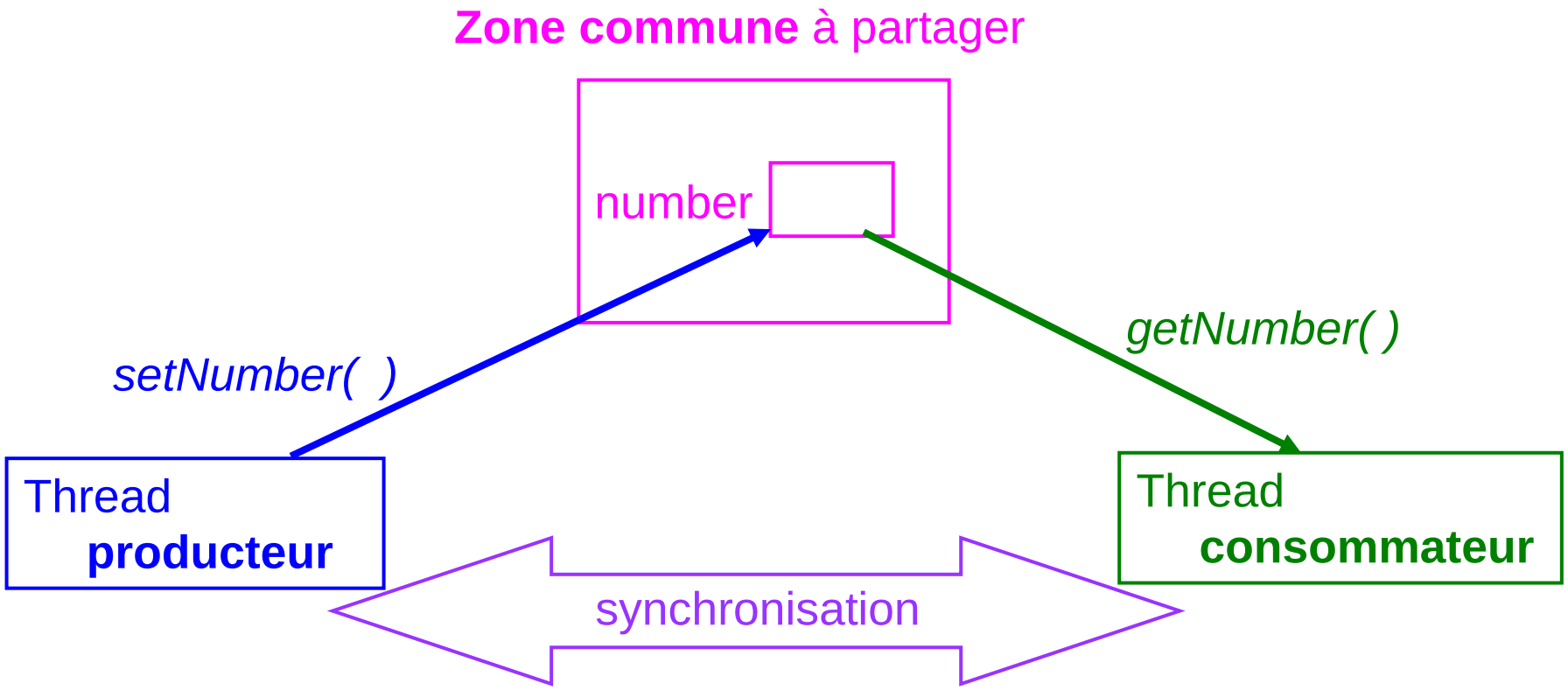
Attention: boucle infinie
(se termine quand l'application se termine)

En millisecondes

Threads

1. **Processus et threads**
2. **Cycle de vie d'un thread**
3. **Choix 1 : implements Runnable**
4. **Choix 2 : extends Thread**
5. **Méthode sleep**
6. **Synchronisation de threads**
   - Producteur - consommateur

Synchronisation de threads



Synchronisation de threads

Classe **CommonZone**

Peut être vu comme un flux de taille maximale 1

⇨ on peut le remplir si number == null

Contient 2 variables d'instance :

Integer number :

zone commune

Au départ, le flux est vide ⇨ number = null

boolean completed :

booléen pour signaler la fin du flux

Au départ ⇨ completed= false;

Synchronisation de threads

```
public class CommonZone {

    private Integer number;          // null au départ
    private boolean completed;       // false au départ


    public Integer getNumber() { return number; }

    public void setNumber(Integer number) { this.number = number; }

    public boolean isCompleted() { return this.completed; }

    public void complete() {
        this.completed = true;
    }
}
```

Synchronisation de threads

Mot-clé *synchronized* :

Empêche les accès concurrents de plusieurs threads

Soit à un bloc de code (ou méthode)

Soit à un objet

Synchronisation de threads

Déclarer une méthode synchronized

Exemple : public synchronized void methodX()

⇨ empêche deux threads d'exécuter en même temps cette méthode **sur le même objet**

Le premier thread qui tente d'exécuter la méthode à la main ;

tout autre futur thread qui tente d'exécuter la méthode sur le même objet

est placé en attente (bloqué)

Synchronisation de threads

Dans les classes **Producer** et **Consumer :**

Variable d'instance de type CommonZone

Utilisation :

```
synchronized (commonZone) {
…
}
```

⇨ commonZone joue à la fois le rôle de flux et à la fois le rôle de lock pour la synchronisation : plusieurs threads ne peuvent pas accéder au même objet en même temps

Synchronisation de threads

Producteur : le thread qui veut accéder en ***écriture*** à la zone commune doit être placé en **attente** si l'élément partagé (number) n'a pas encore été lu.

Synchronisation de threads

```
public class Producer extends Thread {

    private CommonZone commonZone;

    public Producer(CommonZone commonZone) {
        super("producteur");
        this.commonZone = commonZone;
    }
```

```
public void run() {
        synchronized (commonZone) {
            for (int i = 1; i <= 10; i++) {
                try {
                    Thread.sleep ((int) (Math.random( ) * 3000)) ;
                    commonZone.setNumber(i);
                    System.out.println("Producer écrit l'entier : " + i);
                    this.commonZone.notify();
                    this.commonZone.wait();
                }
                catch (InterruptedException exception) { … }
            }
            this.commonZone.complete();
            this.commonZone.notify();
        }
    }
}
```

Pour simuler un accès aléatoire dans le temps à la zone commune

⇨ *remplir le flux avec le nouvel élément*

⇨ *prévenir le consommateur que le flux est rempli*

⇨ *attendre que le flux soit vidé*

⇨ *signaler la fin du remplissage du flux*

⇨ *prévenir le consommateur*

Synchronisation de threads

Consommateur : le thread qui veut accéder en ***lecture*** à la zone commune doit être placé en **attente** si l'élément partagé (number) n'a pas encore été mis à jour (réécrit).

Synchronisation de threads

```
public class Consumer extends Thread {

    private CommonZone commonZone;

    public Consumer(CommonZone commonZone) {
        super("consommateur");
        this.commonZone = commonZone;
    }
```

```
public void run() {
        Integer number;
        synchronized (commonZone) {
          while (! commonZone.isCompleted()) {
            try {
                  Thread.sleep((int) (Math.random() * 3000));
                  number = commonZone.getNumber();      ⇨ récupérer le contenu du flux
                  if (number == null)
                     commonZone.wait();                 ⇨ attendre s'il n'y a rien à lire
                  else {
                                                  ⇧ utiliser l'objet s'il y en avait un à lire
                     System.out.println("Consumer lit le nombre : " + number);
                     commonZone.setNumber(null);        ⇨ vider le flux
                     commonZone.notify();               ⇨ prévenir le producteur
                  }
            }
            catch (InterruptedException exception) { … }
        }
    }
}
```

Synchronisation de threads

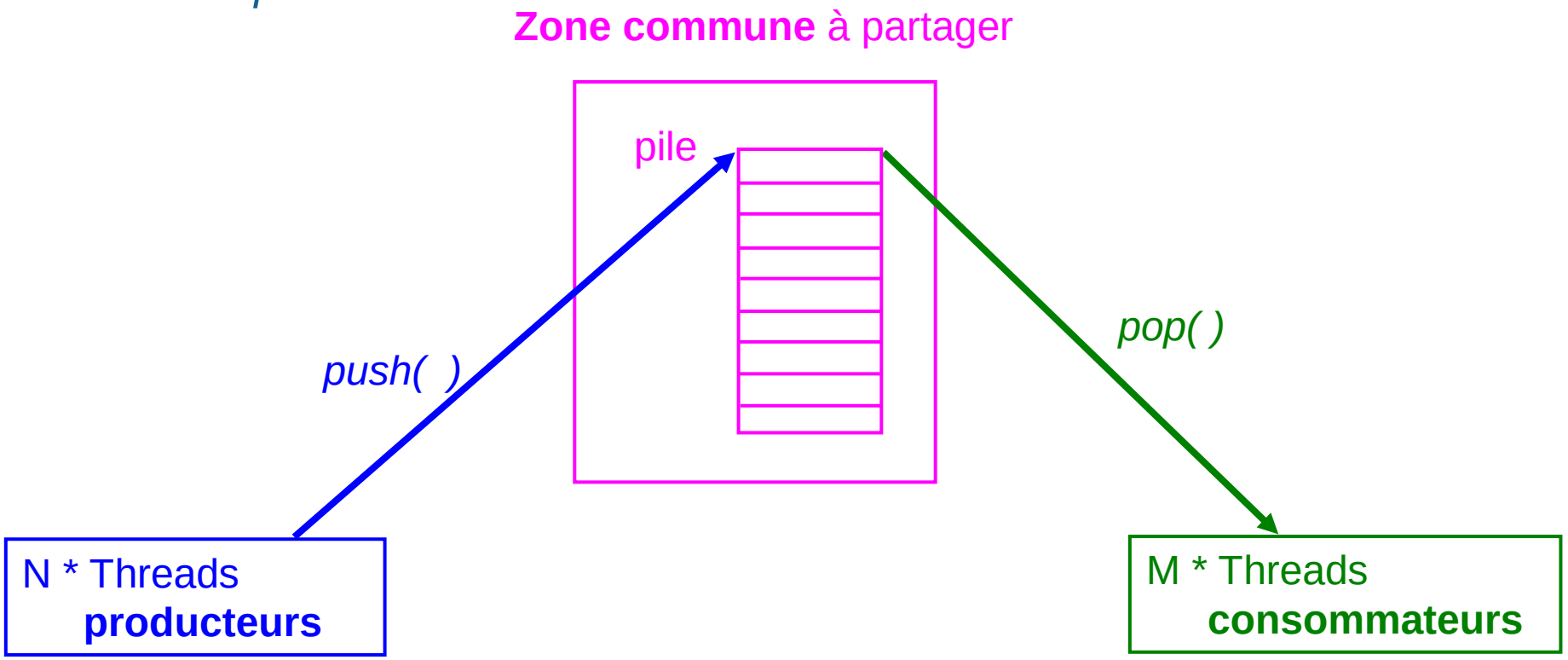
notify( ) réveille un thread en attente

notifyAll( ) réveille tous les threads en attente

C'est l'OS qui décide quel thread réveiller parmi ceux qui sont dans la file d'attente

Synchronisation de threads

Autre exemple : Pile



- wait( ) si pile pleine
- **notifyAll( )** après push( )

- wait( ) si pile vide
- **notifyAll( )** après pop( )