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

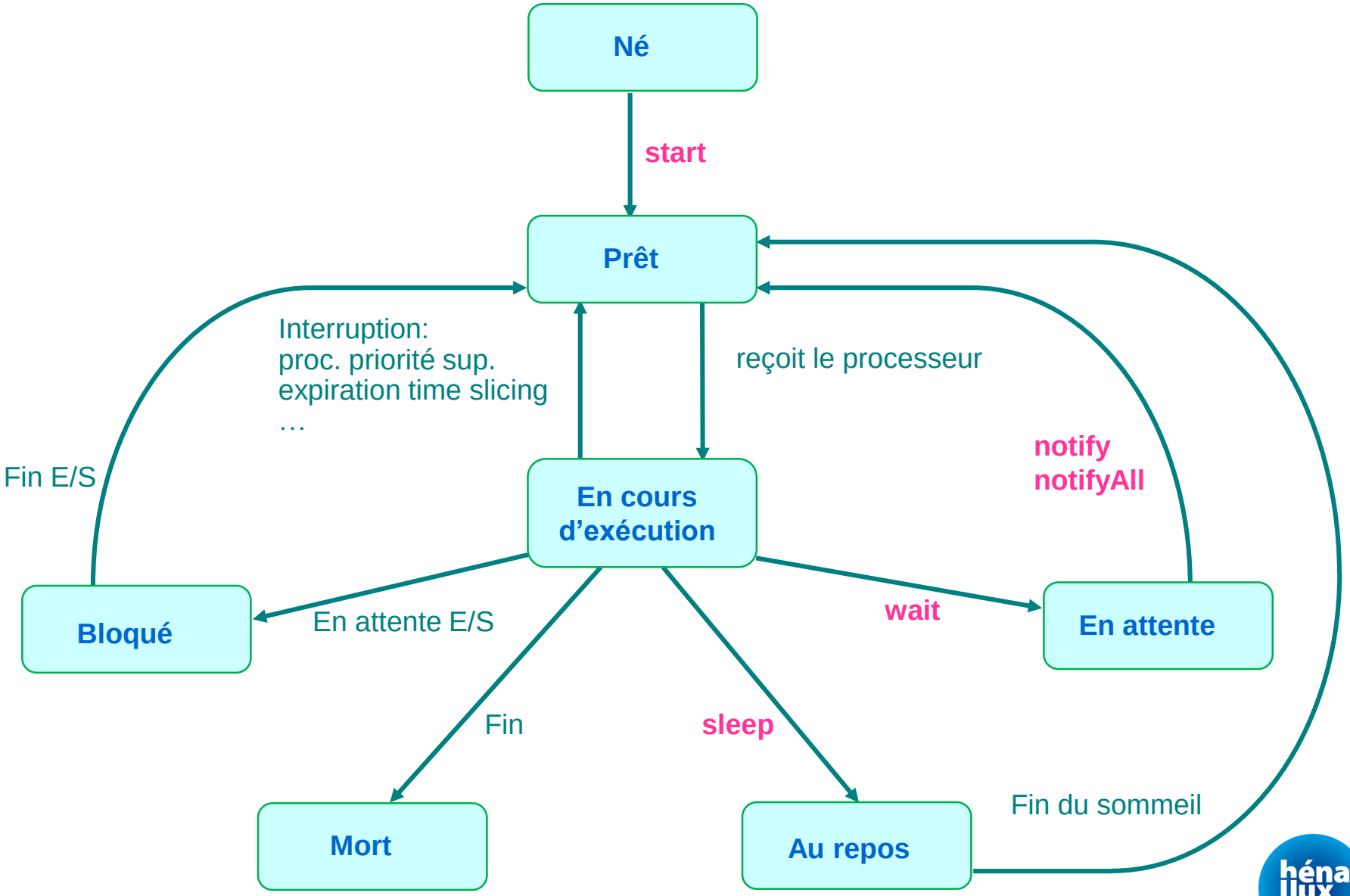
(partage même zone mémoire (espace d'adressage))

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
Fin E/S
notify
notifyAll
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) {
                 …
          }
      }
   }
}
```

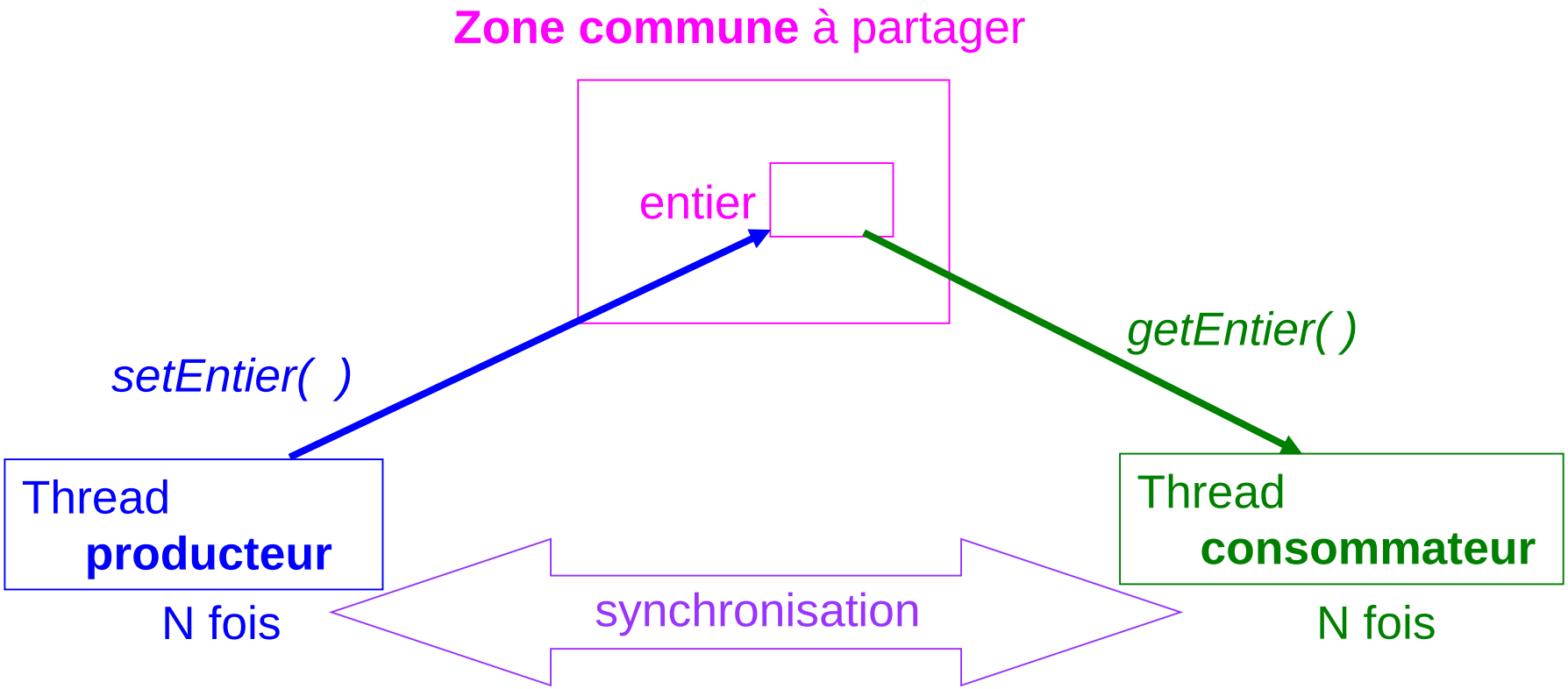
Attention: boucle infinie
(se termine quand l'application se termine)

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

C'est la **zone commune qui place en attente si nécessaire les threads** qui y accèdent :

- **Accès en lecture** (via getter) : placer en **attente** le thread qui veut accéder en lecture ***si la zone commune est "à écraser"*** (càd si elle n'a pas encore été réécrite/mise à jour)
- **Accès en écriture** (via setter) : placer en **attente** le thread qui veut accéder en écriture ***si la zone commune n'est pas "à écraser"*** (càd si elle n'a pas encore été consommée/lue)
- Déclarer tout **getter/setter synchronized** pour empêcher 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

```
public class ZoneCommune {
    private int entier;
    private boolean aEcraser = true;

  public synchronized int getEntier( )  {
    if (aEcraser)  {
        try {
          wait( ) ;
        }
        catch (InterruptedException exception)  {
          exception.printStackTrace( );
        }
    }
    System.out.println ( Thread.currentThread( ).getName( ) +
                         " lit la valeur " + entier);
    aEcraser = true;
    notify( );
    return entier;
  }
```

Deux threads différents ne peuvent exécuter la même méthode en même temps sur le même objet

Thread mis en attente

Réveille un thread en attente

Synchronisation de threads

```
public synchronized void setEntier (int entier) {
    if (! aEcraser) {
        try {
            wait( );
        }
        catch (InterruptedException exception)   {
            exception.printStackTrace();
        }
    }
    this.entier = entier;
    System.out.println ( Thread.currentThread( ).getName( )+
                         " écrit la valeur " + this.entier);
    aEcraser = false;
    notify( );
  }
}
```

Synchronisation de threads

- Le ***producteur*** se contente de tenter d'accéder en ***écriture*** à la zone commune
  - Il se pourrait qu'il soit placé en attente par la zone commune

Synchronisation de threads

```
public class ProducteurSynchro extends Thread {
   private ZoneCommune commun;
   public ProducteurSynchro (ZoneCommune commun) {
         super("producteur");
         setCommun(commun);
   }
   public void run( )  {
          for (int i = 1; i<= 10; i++)  {
              try {
                        Thread.sleep ((int) (Math.random( ) * 3000));
              }
             catch (InterruptedException exception) {
                        exception.printStackTrace( );
             }
             commun.setEntier(i);
          }
   }
}
```

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

Synchronisation de threads

- Le ***consommateur*** se contente de tenter d'accéder en ***lecture*** à la zone commune
  - Il se pourrait qu'il soit placé en attente par la zone commune

Synchronisation de threads

```
public class ConsommateurSynchro extends Thread {
   private ZoneCommune commun;
   public ConsommateurSynchro (ZoneCommune commun)  {
         super("consommateur");
         setCommun(commun);
   }
   public void run( )  {
      int entier;
      for (int i = 1; i<= 10; i++) {
            try { Thread.sleep ((int) (Math.random( ) * 3000)) ;
            }
            catch (InterruptedException exception)    {
                  exception.printStackTrace( );
            }
            entier = commun.getEntier( );
            … }
   }
}
```

Synchronisation de threads

```
public class Main {

public static void main(String[ ] args) {
        ZoneCommune zone = new ZoneCommune( );

        ProducteurSynchro producteur = new ProducteurSynchro(zone);
        ConsommateurSynchro consommateur = new ConsommateurSynchro(zone);

       producteur.start( ) ;
       consommateur.start( ) ;
    }
}
```

Synchronisation de threads

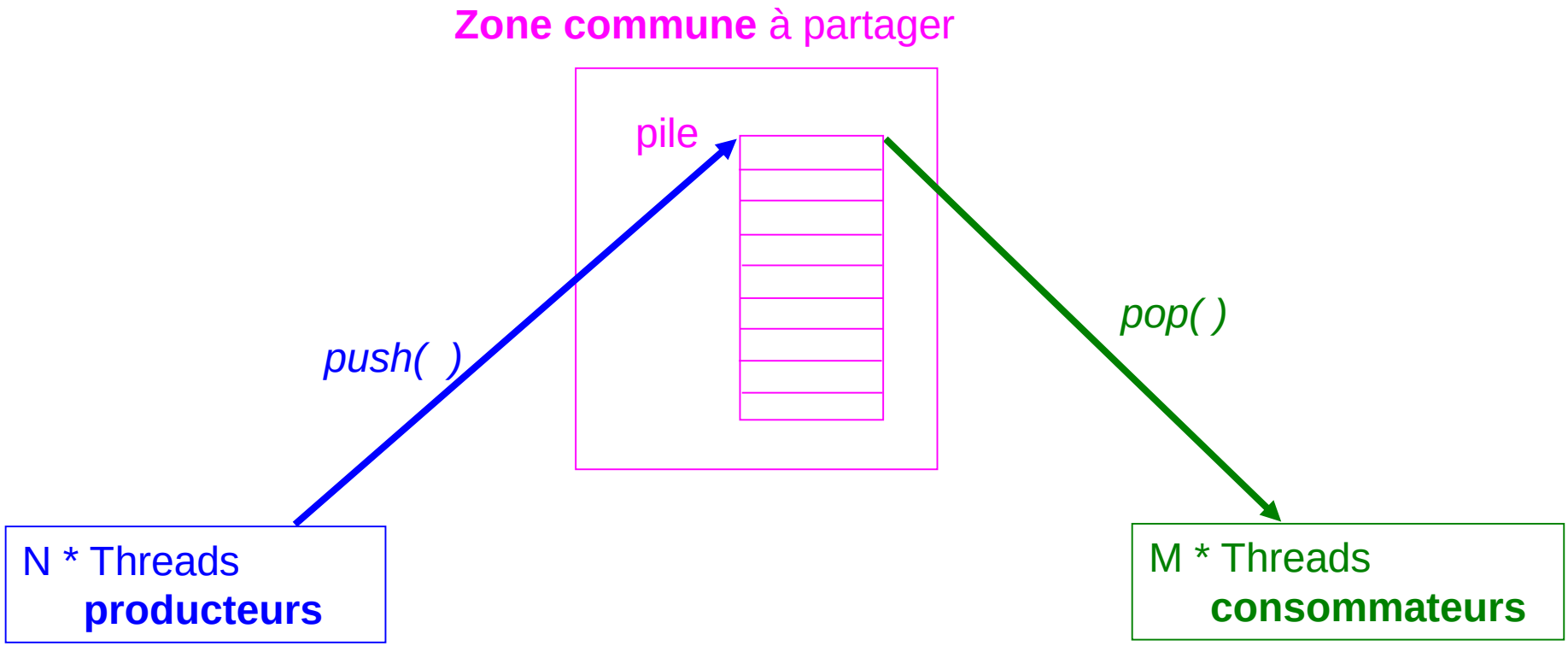
notify( ) réveille un thread en attente

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( )