3. Les threads

- 3.1. Processus et threads

Processus : +/- "programme en cours d'exécution"

Ressources allouées à un processus:
mémoire, processeur, I/O (fichiers, …), …

Thread : "processus léger"
+/- processus à l'intérieur d'un processus

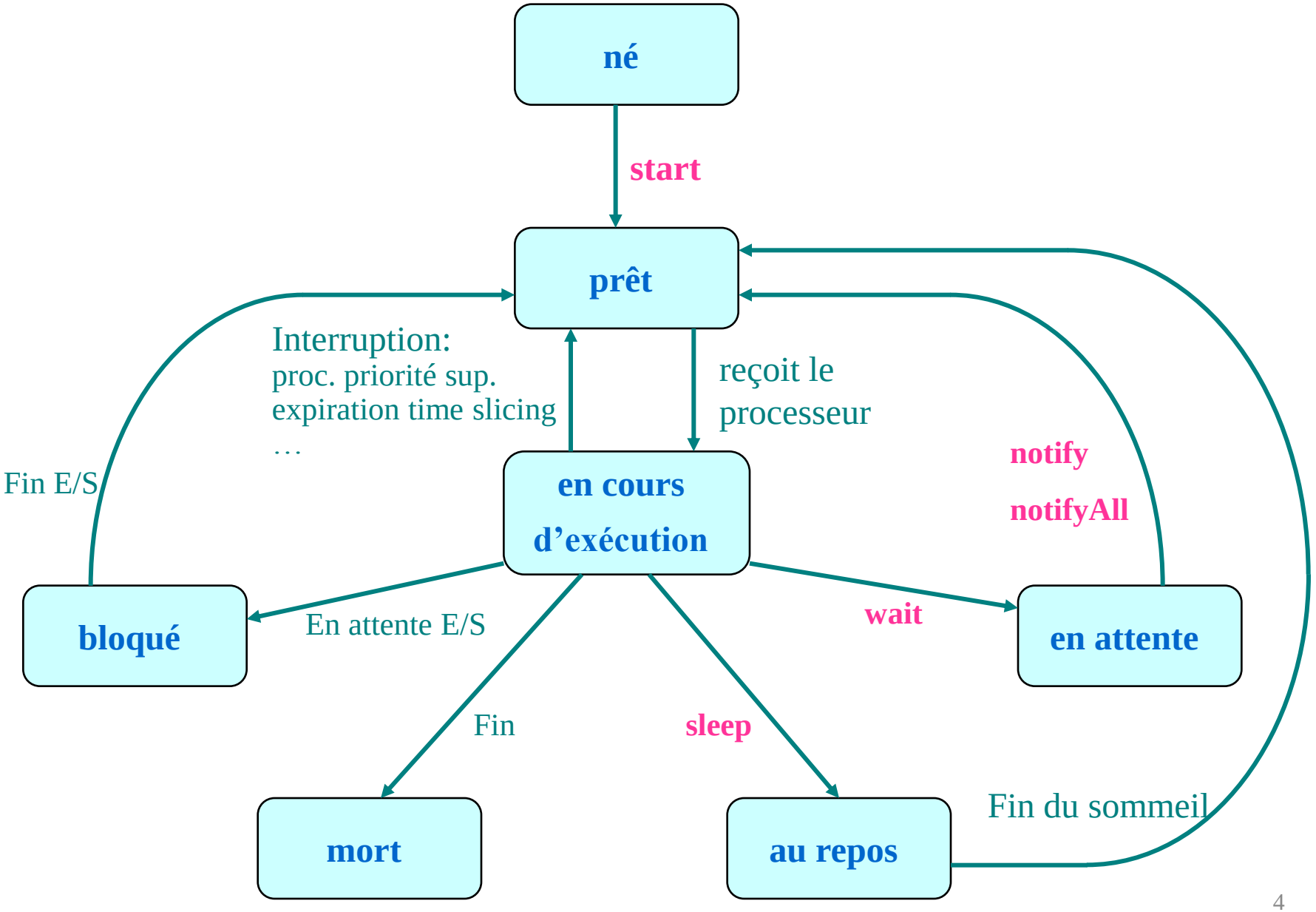
Si plusieurs threads dans le même processus:
les threads travaillent en parallèle

Partage des ressources du processus entre les threads du processus
(partage même zone mémoire (espace d'addressage))

Un processus contient au moins 1 thread (cfr méthode main(…))
Ex: Garbage collector de java

3. Les threads

- 3.1. Processus et threads
- 3.2. Cycle de vie d’un thread


né
start
prêt
Interruption:
proc. priorité sup.
expiration time slicing
…
reçoit le
processeur
Fin E/S
en cours
d'exécution
notify
notifyAll
bloqué
En attente E/S
wait
en attente
Fin
sleep
Fin du sommeil
mort
au repos

3. Les threads

- 3.1. Processus et threads
- 3.2. Cycle de vie d’un thread
- 3.3. Choix 1: implements Runnable

class **MyThread** implements Runnable

{ …

public void run( )

{ // code de la tâche que le thread doit effectuer

}

}

Méthode à redéfinir

Création du thread:

MyThread **myT** = new MyThread( );

Thread **t1** = new Thread(**myT**);

t1.**start( )**;

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

3. Les threads

- 3.1. Processus et threads
- 3.2. Cycle de vie d’un thread
- 3.3. Choix 1: implements Runnable
- 3.4. Choix 2: extends Thread

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

Méthode à redéfinir

Création du thread:

```
MyThread   myT  = new MyThread( );
myT.start( );
```

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

3. Les threads

- 3.1. Processus et threads
- 3.2. Cycle de vie d’un thread
- 3.3. Choix 1: implements Runnable
- 3.4. Choix 2: extends Thread
- 3.5. Méthode sleep

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

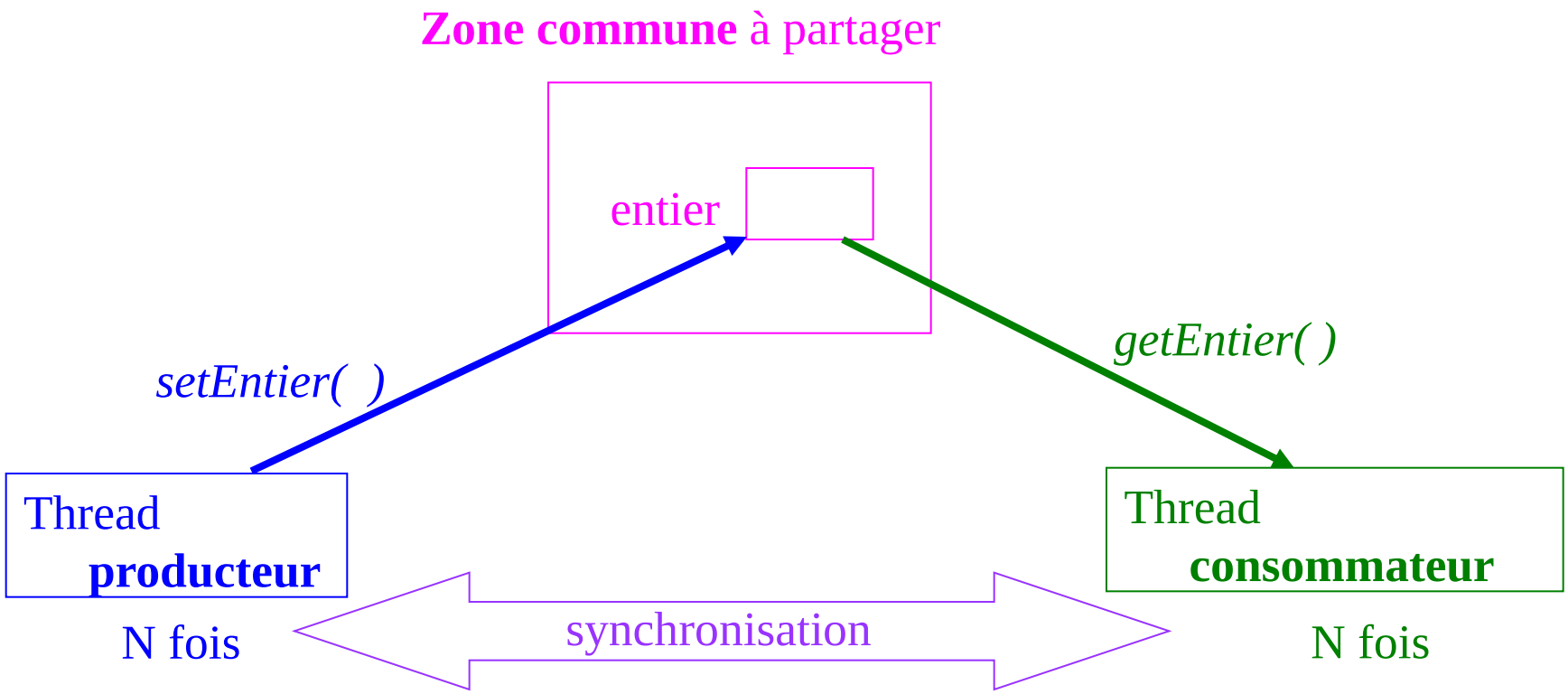
Attention: boucle infinie (se termine quand l'application se termine)

En millisecondes

3. Les threads

- 3.1. Processus et threads
- 3.2. Cycle de vie d'un thread
- 3.3. Choix 1: implements Runnable
- 3.4. Choix 2: extends Thread
- 3.5. Méthode sleep
- 3.6. Synchronisation de threads

  Producteur - Consommateur


Zone commune à partager
entier
setEntier( )
getEntier( )
Thread
producteur
N fois
synchronisation
Thread
consommateur
N fois

```
public class ZoneCommune {

private int entier = -1;
private boolean aEcraser = true;

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

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

Thread en attente

Réveille un thread en attente

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

}
```

```
public class ProducteurSynchro extends Thread {

   private ZoneCommune commun;

   public ProducteurSynchro (ZoneCommune z)
   {super("producteur");
    commun = z;
   }

   public void run( )
   {for (int i = 1; i<= 10; i++)
    {try { Thread.sleep ((int) (Math.random( ) * 3000)) ;}
     catch (InterruptedException e)
                          {e.printStackTrace( );}
     commun.setEntier((i));
     }
   }
}
```

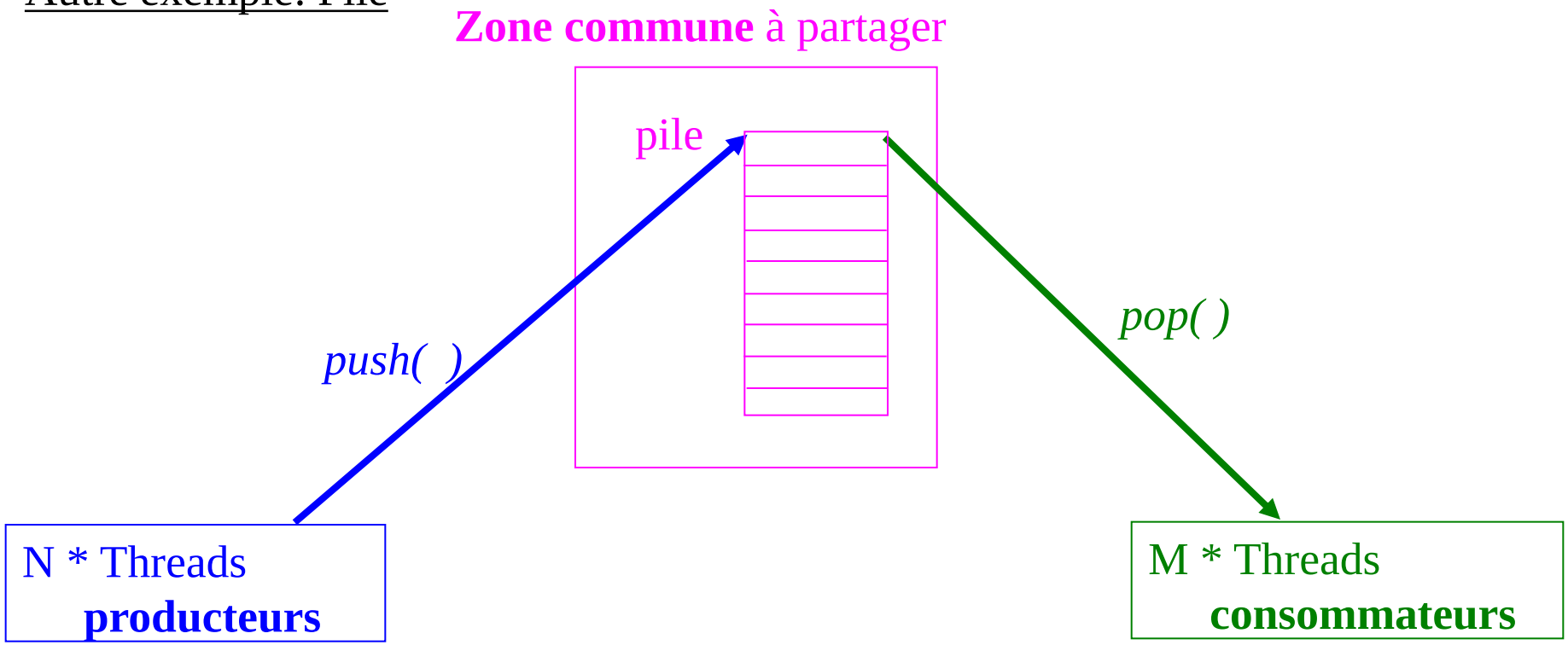
```
public class ConsommateurSynchro extends Thread{

   private ZoneCommune commun;

   public ConsommateurSynchro (ZoneCommune z)
   {super("consommateur");
    commun = z;}

   public void run( )
   { int en, somme = 0;
     for (int i = 1; i<= 10; i++)
    {try {Thread.sleep ((int) (Math.random( ) * 3000)) ;}
     catch (InterruptedException e)
              {e.printStackTrace( );}
     en = commun.getEntier( );
     somme += en;}
     System.out.println("La somme des nombres lus par le consommateur est : " +
      somme);
   }    }
```

```
public class Main {

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

        ProducteurSynchro p = new ProducteurSynchro(entier);
        ConsommateurSynchro c = new ConsommateurSynchro(entier);

        p.start( ) ;
        c.start( ) ;
    }

}
```

notify( ) réveille un thread en attente

notifyAll( ) réveille tous les threads en attente

Autre exemple: Pile



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

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