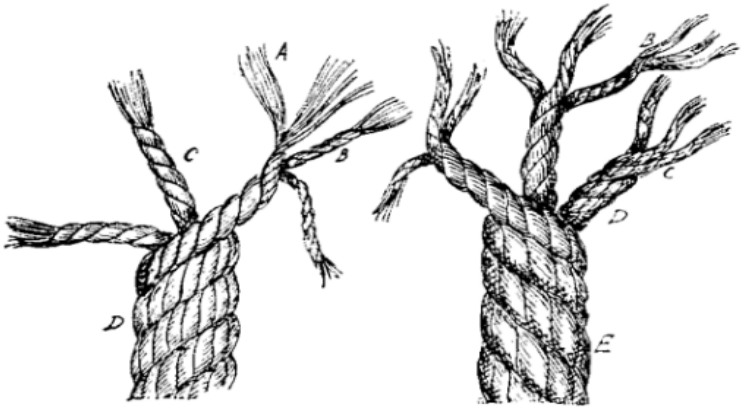
Programmation multi-threads en Java



Module
Systèmes Communicants et Synchronisés

Master Informatique

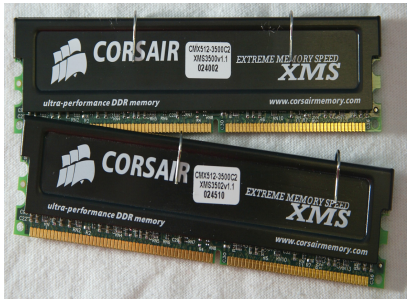
1ère année

V. Felea & L. Philippe

Exploitation des ressources (1)

processeur

+



mémoire

=

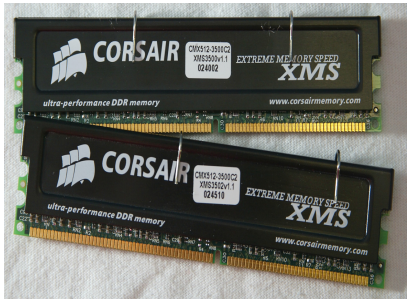
```c
#include <stdio.h>
#include <stdlib.h>

int main(int argc, char *argv[])
{
  int i;
  i=1;
  printf("entrez un entier :");
  scanf("%d",&i);
  printf("i est egal a %d",i);
  printf("\nj'incremente de 1");
  printf("\ni est egal a %d",++i);
  printf("\n");
  system("PAUSE");
  return 0;
}
```

programme

Exploitation des ressources (2)

1 processeur

+



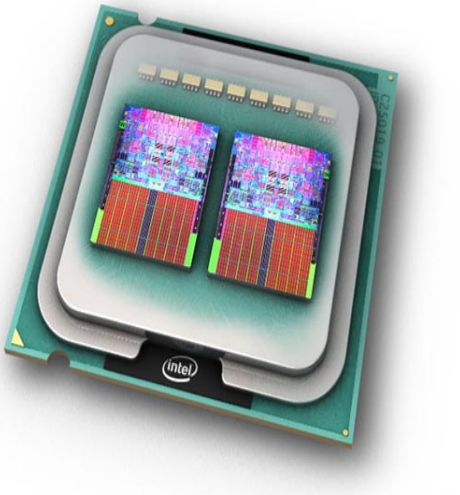
1 mémoire

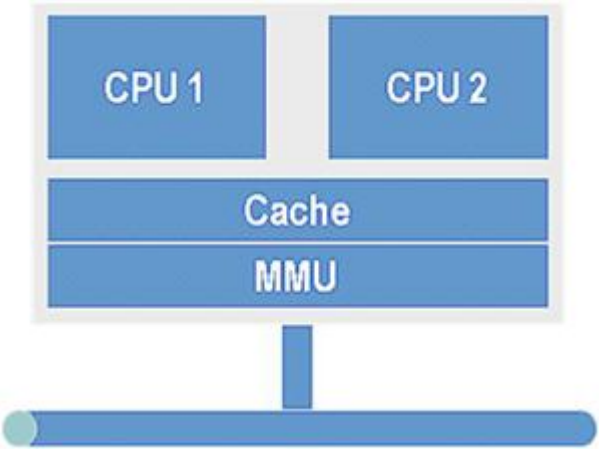
=

premier.c

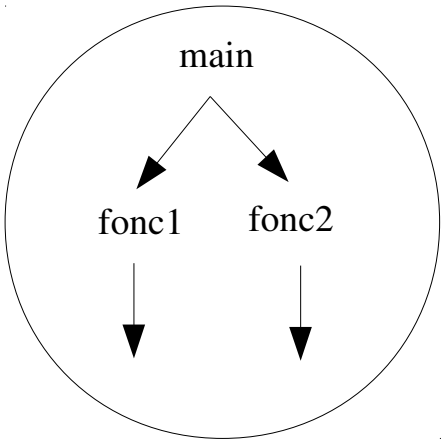
```c
#include <stdio.h>
#include <stdlib.h>

int main(int argc, char *argv[])
{
  int i;
  i=1;
  printf("entrez un entier :");
  scanf("%d",&i);
  printf("i est egal a %d",i);
  printf("\nj'incremente de 1");
  printf("\ni est egal a %d",++i);
  printf("\n");
  system("PAUSE");
  return 0;
}
```

1 exécution



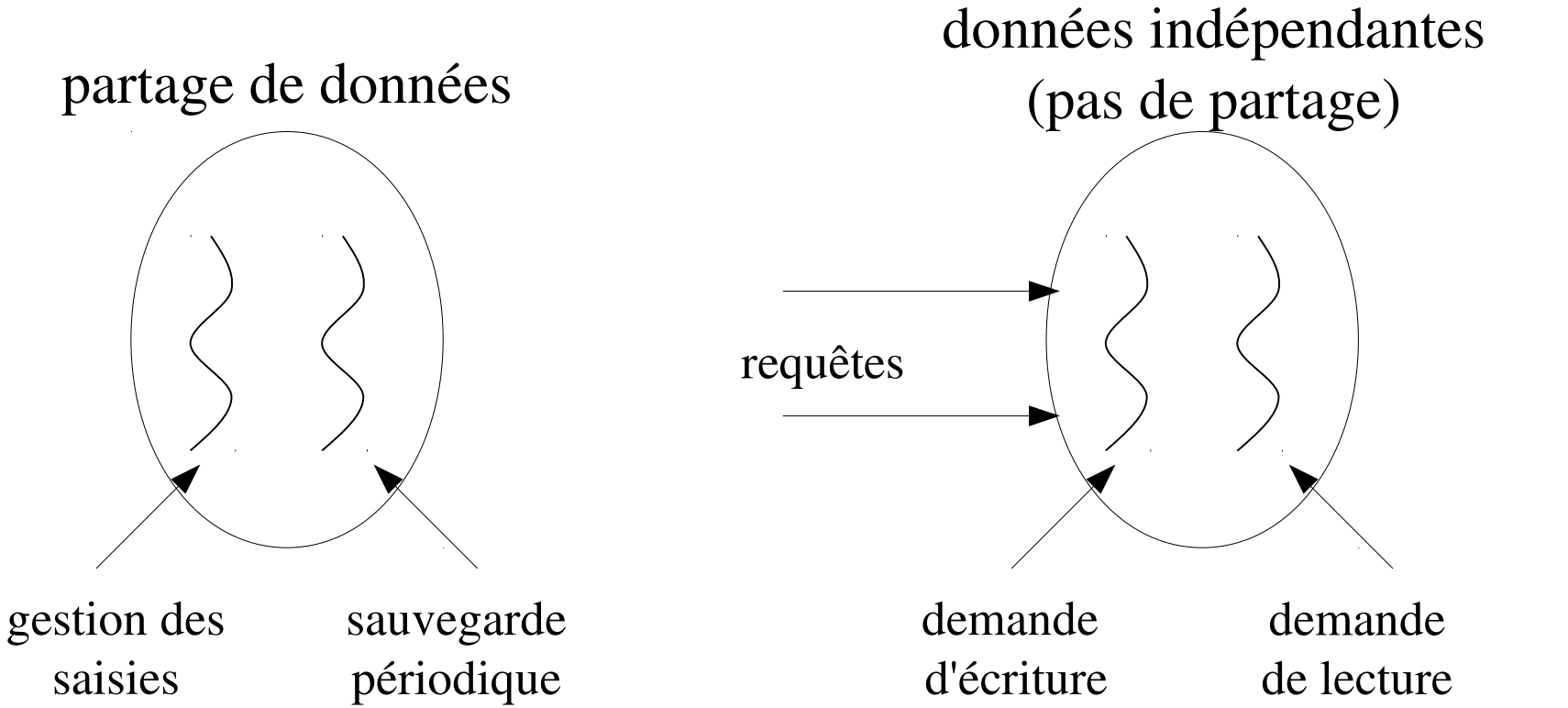


=



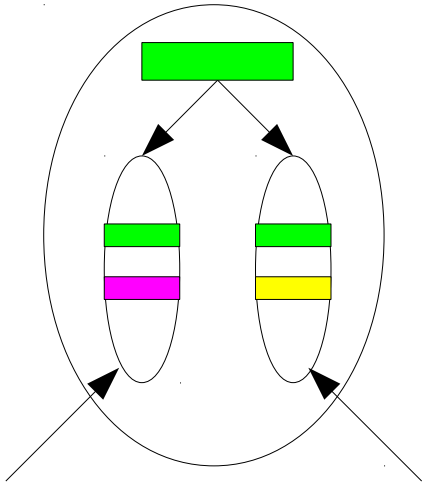
performances !!!

Traitements non bloquants



Traitements non bloquants – partage de données

application
traitement de texte



gestion des saisies

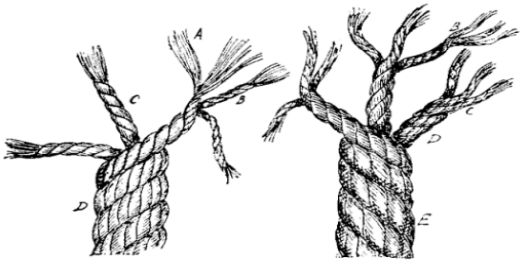
sauvegarde périodique

création de processus
- un par activité

partage de données ??
- données du fichier courant

solutions : communication inter-processus ou mémoire partagée

Threads



- flot/fil d'exécution (d'une suite d'instructions) qui s'exécute au sein d'un processus
- appellations
  - processus légers
  - tâche / task
- principe
  - plusieurs exécutions simultanées dans un même processus
  - partage de la mémoire
  - partage des ressources : fichiers, périphériques, etc.
  - registres et pile privés aux threads

Applications

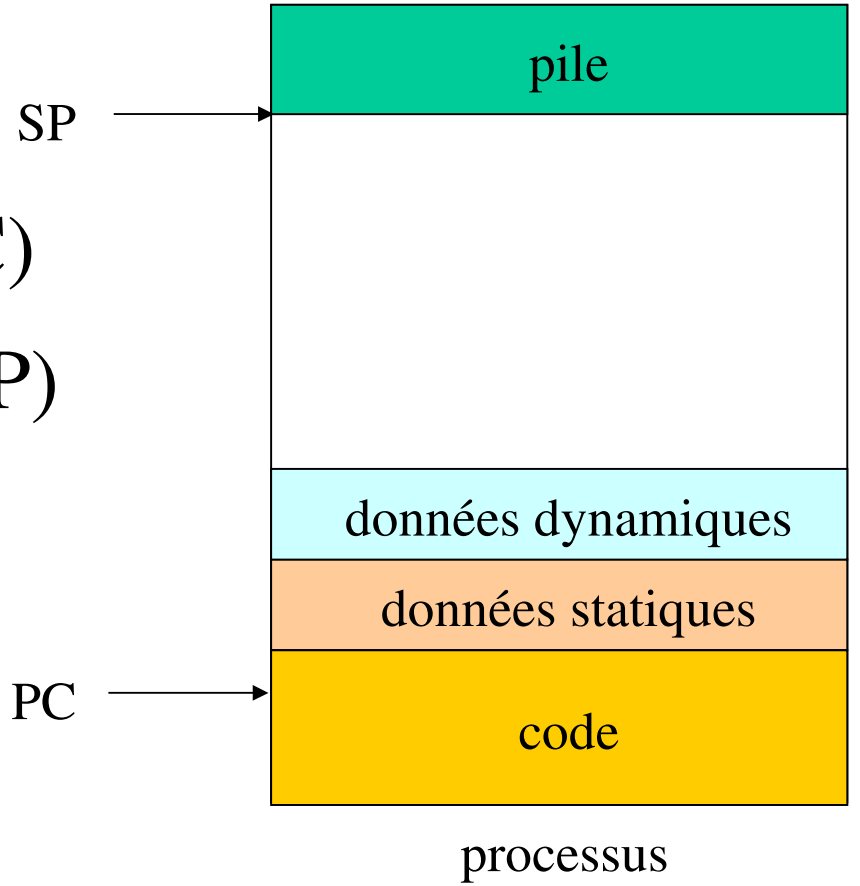
- multi-processeurs
- multi-coeurs
- programmation événementielle
- programmation parallèle
- programmation interactive

Programmation des serveurs

- ensemble de threads prêts à recevoir
- un thread par requête
- plusieurs requêtes sur la même mémoire
  - clients à état
  - historique
- pas de blocage du processus exécutant des fonctions de communication réseau bloquantes : pas besoin de select

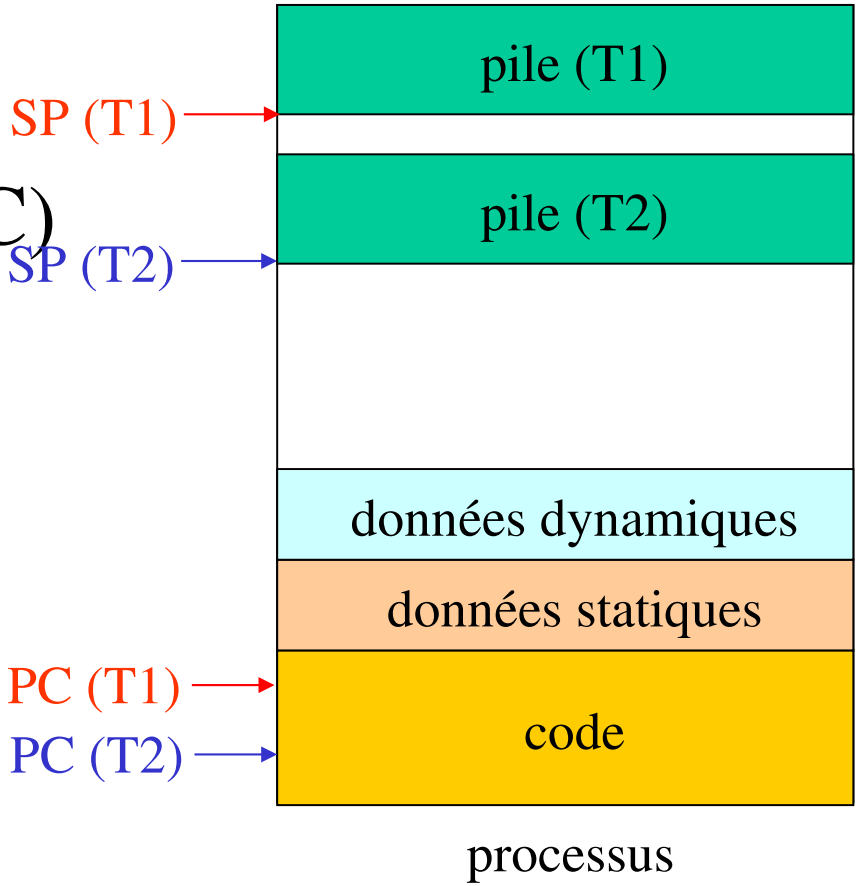
Processus

- contexte d'exécution
  - compteur ordinal (PC)
  - pointeur de la pile (SP)
  - registres de données
- code
- données
- pile



Threads

- contexte d'exécution
  - compteur ordinal (PC)
  - pointeur de la pile (SP)
  - registres de données

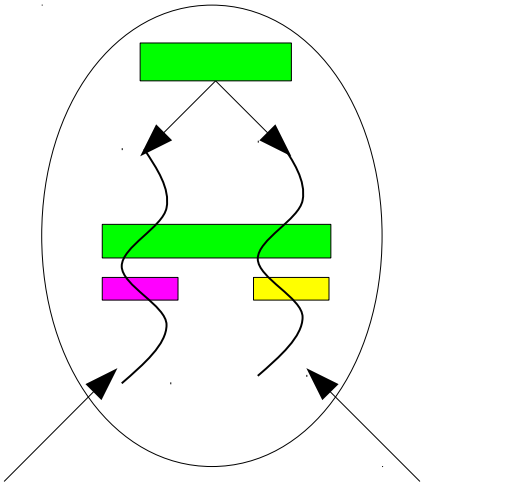


Processus - Threads

- chaque processus détient un ou plusieurs threads
- chaque thread appartient à un seul processus
- processus
  - changement de contexte et communication interprocessus couteux
  - sécurité : un processus ne peut pas corrompre les données d'un autre processus
- threads
  - communication inter-threads moins coûteuse
  - sécurité : un thread peut écrire dans la mémoire d'un autre thread

Partage de données

- possibilité de créer d'autres flots qui se partagent la zone de mémoire allouée au processus
- ces flots peuvent alors communiquer entre eux en utilisant des variables stockées dans le tas



gestion des saisies

sauvegarde régulière

Implémentation des threads

- niveau utilisateur (*green*)
- niveau noyau (*native*)
- approche hybride

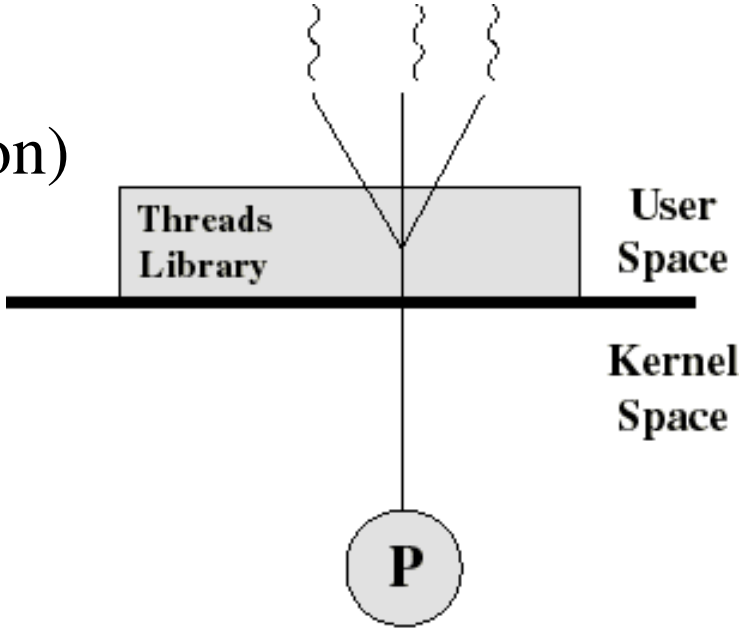
Threads utilisateur (1)

- implémentés dans une bibliothèque (niveau utilisateur) qui fournit un support pour leur gestion
  - complètement transparents pour le noyau (le noyau : table de processus)
  - le temps d'allocation du processeur est réparti entre les threads du processus (non gérée par le noyau)
  - si un thread bloque, tous les threads du même processus bloquent
- exemples : threads Java, versions anciennes d'UNIX

Threads utilisateur (2)

- à tout instant, au plus un thread par processus est en cours d'exécution

- gestion rapide des threads (création)
- portabilité

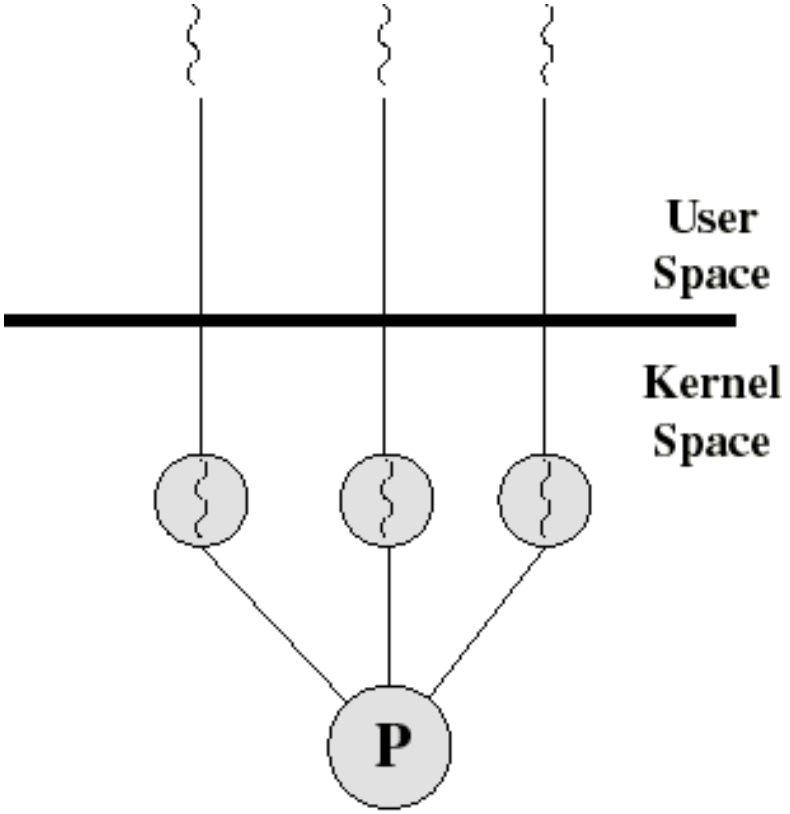


Threads noyau (1)

- les threads sont ordonnancés par le noyau
  - temps alloué à chaque thread
  - un thread bloqué n'influe pas sur les autres threads d'un même processus
  - modèle qui exploite les systèmes multi-processeurs
  - nécessite de changement de contexte, mais moins coûteux que le changement de contexte entre processus
- exemples :Windows NT, Windows 2000, Linux

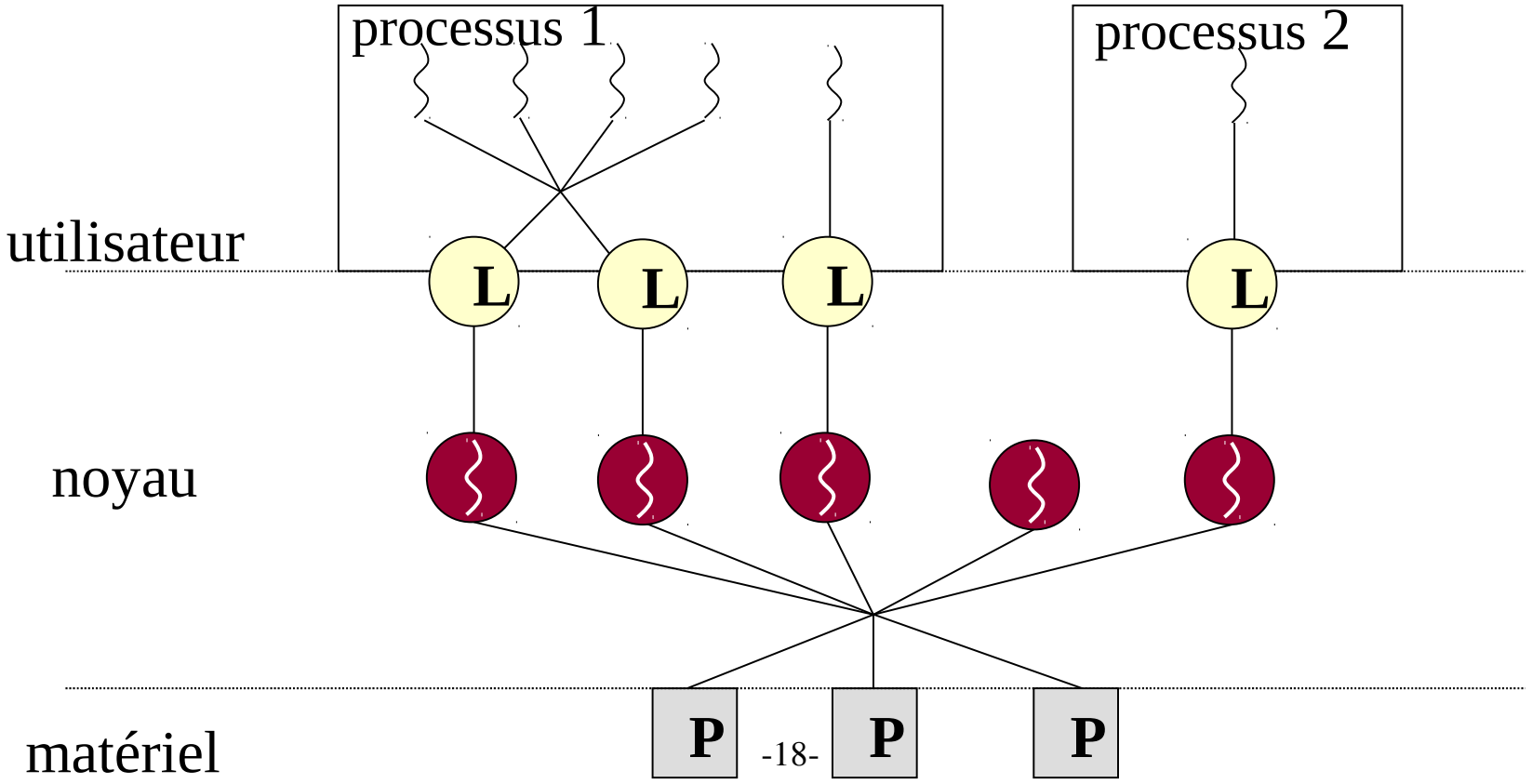
Threads noyau (2)

- gestion plus coûteuse des threads
- portabilité réduite



Approche hybride

- threads intermédiaires entre les threads utilisateur et les threads noyau : LightWeight Process (LWP) transparents pour l'application
  - les threads utilisateurs peuvent être mappés sur plusieurs LWPs
- exemple : Solaris



Programmation des threads Java

- classe de base : *java.lang.Thread* (JDK depuis 1.1)
- sur une base objet : hérite de la classe *Object*
- deux approches
  - par héritage : de la classe *Thread*
  - par spécification d'interface : implémentation de *Runnable*
- point d'entrée : méthode d'instance *void run()* – à redéfinir
- point de départ : appel à la méthode *void start()* sur l'objet

Création des threads : *Object* et *Runnable*

```
// classe Thread
class MonObjetThread extends Thread {
  public void run() {
   // code du thread
    System.out.println("Debut thread");
  }
}

// lancement Thread
public class Principal {
  public static void main(String args[]) {

    MonObjetThread mot = new MonObjetThread();
    System.out.println("lance premier");
   // démarrage du Thread
    mot.start();
    System.out.println("Thread lance !");
  }
}
```

2|3

1

2|3

```
// interface Runnable
class MonObjetRun implements Runnable {
  public void run() {
   // code du thread
    System.out.println("Debut thread");
  }
}

// lancement d'un thread
public class Principal {
  public static void main(String args[]) {

    MonObjetRun mor = new MonObjetRun();

    Thread th = new Thread(mor);
    System.out.println("lance premier");
    // démarrage du Thread
    th.start();
    System.out.println("Thread lance !");
  }
}
```

2|3

1

2|3

Exécution:

```
$ java Principal
lance premier
Thread lance !
Debut thread
```

```
$ java Principal
lance premier
Debut thread
Thread lance !
```

test.java

La classe *Thread* : constructeurs

- *Thread()*
- *Thread(Runnable target)*
- *Thread(String name)*
- *Thread(Runnable target, String name)*
- *Thread(ThreadGroup tg, Runnable target, String s, long stackSize)*
- + combinaisons

La classe *Thread* : méthodes de gestion

- création + démarrage : *new* + *start*
- terminaison
  - fin de fonction
  - *Runtime.exit(int status)*
- attente de terminaison : *void join([long mili [,int nanos]])*
- laisse la main : *static void yield()*
- endormissement : *static void sleep([long mili [,int nanos]])*
- interruption : *void interrupt() / static boolean interrupted() / boolean isInterrupted()*

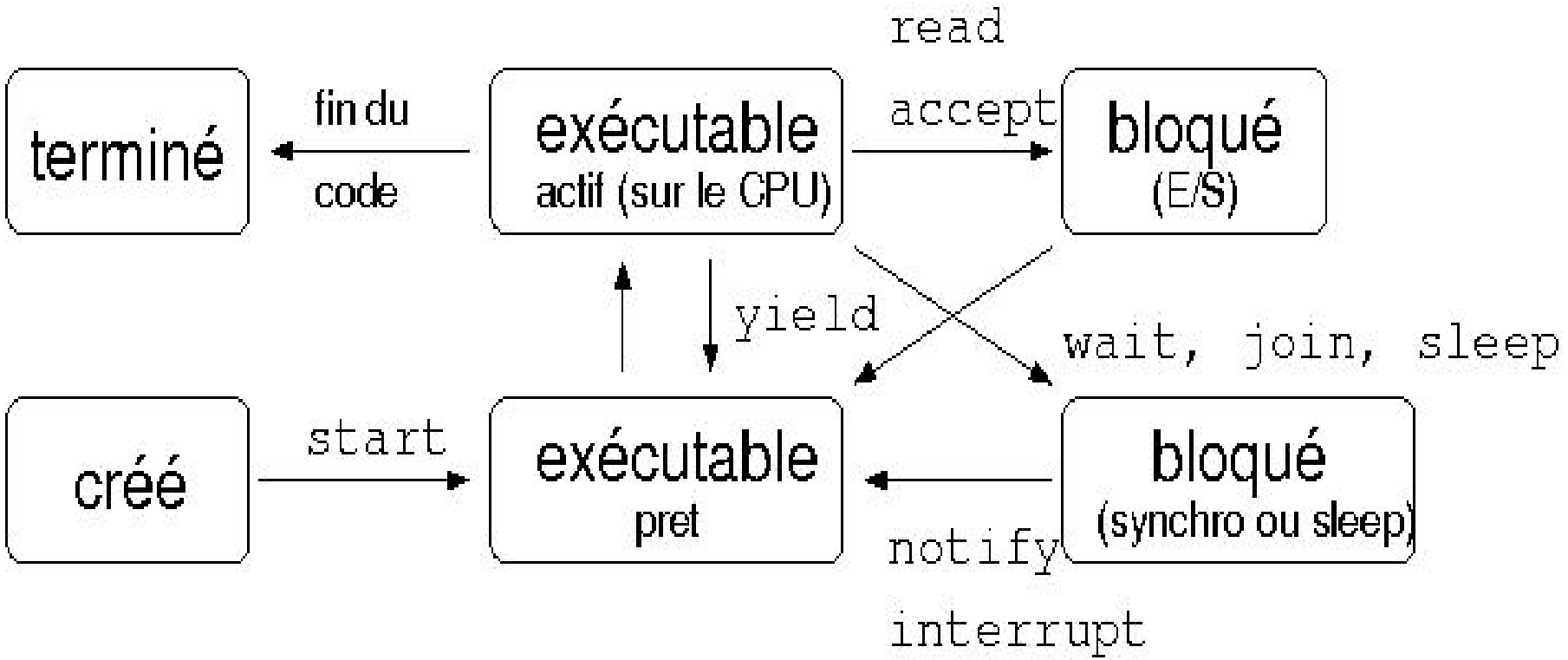
Exemple de gestion des threads

EssaiYield.java

```
class EssaiThread extends Thread {
   private long vart;
   public EssaiThread(long t) { vart = t;  }
   public void run(){ for (int i = 0;  i < 5; i++){
                          System.out.println( getName() + " "+ vart+" i = "+i);
                          this.yield();    // passe la main
} }  }
public class EssaiYield {
    public static void main(String args[]) {
        EssaiThread thread1 = new EssaiThread(1);
        EssaiThread thread2 = new EssaiThread(2);
        thread1.start();
        thread2.start();
        for (int i = 0; i < 6; i++){
            System.out.println("je suis la tache principale !");
            Thread.yield();
         }
         try {   thread1.join();
                 System.out.println("Fin thread 0!");
                 thread2.join();
                 System.out.println("Fin thread 1!");
          } catch(Exception e) {    System.out.println(e) ; }
}   }
```

```
$ java EssaiYield
Thread-0 1 i = 0
je suis la tache principale !
Thread-1 2 i = 0
je suis la tache principale !
Thread-0 1 i = 1
Thread-1 2 i = 1
je suis la tache principale !
Thread-0 1 i = 2
Thread-1 2 i = 2
je suis la tache principale !
Thread-0 1 i = 3
Thread-1 2 i = 3
je suis la tache principale !
Thread-0 1 i = 4
Thread-1 2 i = 4
je suis la tache principale !
Fin thread 0!
Fin thread 1!
```

Etat des threads



Priorités des threads

- une priorité est associée à chaque thread
- gestion de la priorité
  - priorité initiale d'un thread = priorité du thread qui l'a créé
  - *int getPriority()*
  - *void setPriority(int newPri)*
- valeurs prédéfinies (constantes entières de 1 à 10)
  - *Thread.MIN_PRIORITY*
  - *Thread.NORM_PRIORITY*
  - *Thread.MAX_PRIORITY*
- ordonnancement dépend du système

Priorité des threads - Linux

```
class EssaiThread extends Thread {
   private long vart;
   public EssaiThread(long t) { vart = t;}
   public void run(){
        for (int i = 0; i < 5; i++){
                System.out.println(getName() +" "+ vart+" i = "+i);
      }   }   }
public class EssaiPriorite {
   public static void main(String args[]) {
        EssaiThread thread1 = new EssaiThread(1);
        EssaiThread thread2 = new EssaiThread(2);
        thread1.setPriority(Thread.MIN_PRIORITY);
        thread2.setPriority(Thread.MAX_PRIORITY);
        thread1.start();
        System.out.println("Debut thread 1");
        thread2.start();
        System.out.println("Debut thread 2");
        for (int i = 0;  i< 6; i++)
          System.out.println("je suis la tache principale !");

        try {    thread1.join();
                 System.out.println("Fin thread 0!");
                 thread2.join();
                 System.out.println("Fin thread 1!");
        } catch(Exception e) {    System.out.println(e);     }
   }   }
```

EssaiPriorite.java

exécution 1

```
$ java EssaiPriorite
Debut thread 1
Thread-0 1 i = 0
Thread-0 1 i = 1
Thread-0 1 i = 2
Thread-0 1 i = 3
Thread-0 1 i = 4
Debut thread 2
je suis la tache principale !
je suis la tache principale !
je suis la tache principale !
je suis la tache principale !
je suis la tache principale !
je suis la tache principale !
Fin thread 0!
Thread-1 2 i = 0
Thread-1 2 i = 1
Thread-1 2 i = 2
Thread-1 2 i = 3
Thread-1 2 i = 4
Fin thread 1!
```

exécution 2

```
$ java EssaiPriorite
Debut thread 1
Thread-0 1 i = 0
Thread-0 1 i = 1
Debut thread 2
Thread-0 1 i = 2
je suis la tache principale !
je suis la tache principale !
je suis la tache principale !
je suis la tache principale !
je suis la tache principale !
Thread-0 1 i = 3
Thread-1 2 i = 0
Thread-1 2 i = 1
Thread-1 2 i = 2
Thread-1 2 i = 3
Thread-1 2 i = 4
Thread-0 1 i = 4
Fin thread 0!
Fin thread 1!
```

Priorité des threads - Linux

```
class EssaiThread extends Thread {
   private long vart;
   public EssaiThread(long t) { vart = t;}
   public void run(){
        for (int i = 0; i < 5; i++){ System.out.println(getName() +" "+ vart+" i = "+i);
                                     if (i == 1) this.yield();
      }   }   }
public class EssaiPriorite {
   public static void main(String args[]) {
        EssaiThread thread1 = new EssaiThread(1);
        EssaiThread thread2 = new EssaiThread(2);
        thread1.setPriority(Thread.MIN_PRIORITY);
        thread2.setPriority(Thread.MAX_PRIORITY);
        thread1.start();
        System.out.println("Debut thread 1");
        thread2.start();
        System.out.println("Debut thread 2");
        for (int i = 0; i < 6; i++)
          System.out.println("je suis la tache principale !");

        try {    thread1.join();
                 System.out.println("Fin thread 0!");
                 thread2.join();
                 System.out.println("Fin thread 1!");
        } catch(Exception e) {    System.out.println(e);      }
   }   }
```

```
$ java EssaiPriorite
Debut thread 1
Thread-0 1 i = 0
Thread-0 1 i = 1
Debut thread 2
je suis la tache principale !
je suis la tache principale !
je suis la tache principale !
je suis la tache principale !
je suis la tache principale !
je suis la tache principale !
Thread-1 2 i = 0
Thread-1 2 i = 1
Thread-0 1 i = 2
Thread-0 1 i = 3
Thread-0 1 i = 4
Thread-1 2 i = 2
Thread-1 2 i = 3
Thread-1 2 i = 4
Fin thread 0!
Fin thread 1!
```

Ordonnancement des threads Java

- préemptif basé sur la priorité
- (dépendant de l'implémentation) threads avec la même priorité
  - time-slicing (RR)
  - non-préemptive
- « Threads with higher priority are executed **in preference** to threads with lower priority » (http://docs.oracle.com/javase/1.4.2/docs/api/java/lang/Thread.html)
- « At any given time, when multiple threads are ready to be executed, the thread with the highest priority is chosen for execution – However, this is not guaranteed. The thread scheduler may choose to run a lower priority thread to avoid starvation. » ("The Java Tutorial, Third Edition" M.Campione, K. Walrath, A. Huml)

Interruption de thread (1)

- interrupt flag : état
- méthodes levant *InterruptedException* : *sleep*, *join*, *wait*
- *void interrupt()* : interrompre le thread
  - si le thread réalise une opération bloquante (*wait*, *join*, *sleep*)
    - débloque le thread → état prêt
    - reçoit *InterruptedException*, drapeau d'état effacé
  - sinon, le drapeau d'état est positionné
- *static boolean interrupted()* : tester et effacer état
- *boolean isInterrupted()* : tester

Interruption de thread (2)

Thread principal

```
EssaiThread et = new EssaiThread();
et.start() ;
try{
        Thread.sleep(10000) ;
}catch(InterruptedException e){...}
et.interrupt();
...
```

thread interrompu (EssaiThread)

```
for (int i = 0; i < max; i++) {
   / /pause for 4 seconds
   try {
      Thread.sleep(4000);
   } catch (InterruptedException e) {
      System.out.println("Interrupted");
      return;
   }
   // print a message
   System.out.println("bonjour");
}
```

thread interrompu (EssaiThread)

```
for (int i = 0; i < max; i++) {
   travaille(truc);
   if  (Thread.interrupted()) {

      // I've been interrupted
      return;
   }
}
```

La classe *Thread* : divers

- *String getName() / void setName(String name)*
  - par défaut `Thread-0`,`Thread-1`, ...
- *boolean isAlive()*
- *static void sleep(long milis)*
- *static Thread currentThread()*

  => Thread t = Thread.currentThread();

Programmation d'un serveur (1)

- 1 connexion = 1 thread => 1 client = 1 thread
- un seul programme : main + thread
- pas de gestion select ou non-bloquant
- comme processus
  - création d'un thread à chaque requête
  - création d'un pool de threads
- partage mémoire

Programmation d'un serveur (2)

- création d'un thread par connexion

```
class TraitReq extends Thread {
    Socket s;
    public TraitReq (Socket sock) { s = sock; }
    public void run() {
        boolean fini = false;
        try {
            OutputStream os =  s.getOutputStream() ;
            ObjectOutputStream oos = new
                            ObjectOutputStream(os);
           // même pour input stream
            while (!fini) {
                String recu =(String) ois.readObject();
                if (recu.equals("FIN"))  fini = true;
                else  oos.writeObject("Echo " + recu);
            }
            // fermer les flux
            s.close();
        } catch (Exception ex) { ... }
    }}
```

```
public class ServerMessThread {
    public static void main(String [] args) {
        ServerSocket srv;
        int port = 5555 ;
        Socket s = null;
        try {
            srv = new ServerSocket(port);
            while (true) {
                s = srv.accept() ;
                TraitReq tr = new TraitReq(s);
                tr.start();
            }
            // should never been reached
        } catch(Exception e) {
            System.out.println("IO exception" + e);
        }
    }
}
```

- pas de changement du client

Programmation d'un serveur (3)

- création d'un pool de threads

```
class TraitReq extends Thread {
   ServerSocket srv;
   public TraitReq(ServerSocket sock) { srv=sock; }
   public void run() {
        while (true) {
           try {
              boolean fini = false;
              Socket s = srv.accept();
              // créer les flux d'entrée - ois /sortie
              while (!fini) {
                 String recu = (Sring)ois.readObject();
                 if (recu .equals("FIN"))  fini = true;
                 else
                    oos.writeObject("Echo "+recu);
              }
             // fermer les flux
             s.close();
           } catch (Exception ex) { ... }          }} }
```

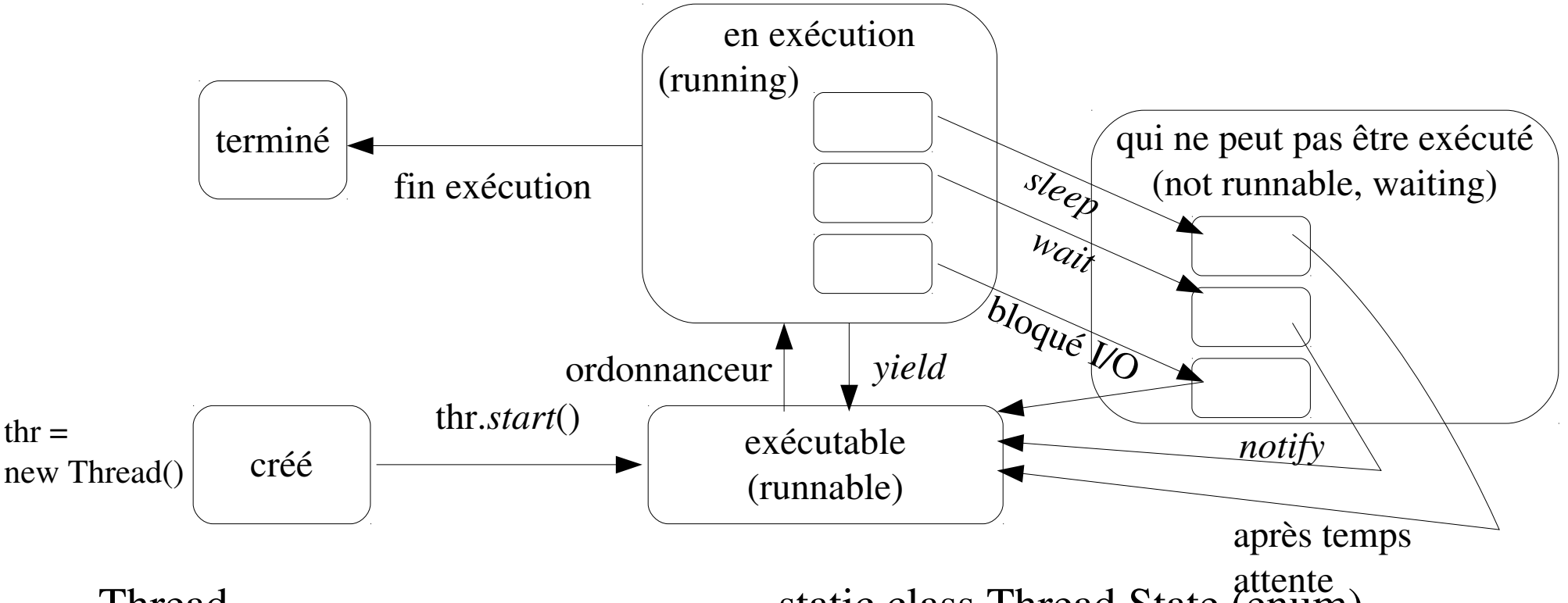
```
public class ServerMessPoolThread {
   private static final int NBTHR = 5;
   public static void main(String [] args) {
        int port = 5555;
        try {
           srv = new ServerSocket(port);
           // Thread pool creation
           TraitReq[] threadTab = new TraitReq[NBTHR];
           for (int i = 0; i < NBTHR; i++) {
                threadTab[i] = new TraitReq(srv);
                threadTab[i].start();
           }
           // do nothing or receive request
           threadTab[0].join();
            // should never been reached
        } catch(Exception e) {
           System.out.println("IO exception" + e);
        }     }}
```

- pas de changement du client

Modèles de programmation

- threads identiques
  - données locales : gestion d'une requête
  - données globales : gestion historique, données stockées
- threads différenciés
  - code différent
  - code identique
    - choix par nom: *getName* / *setName*
  - accède aux mêmes données, sauf pile + données propres aux threads
- gestion de la synchronisation

Etat des threads – wait/notify vus lors de la synchro



Thread

Thread.State getState()

static class Thread.State (enum)

NEW
RUNNABLE
BLOCKED
WAITING
TIMED_WAITING
TERMINATED

Sécurité threads

- méthodes de gestion
  - *void checkAccess()*
    - exception *SecurityException*
- *java.lang.SecurityManager*
    - *void checkAccess(Thread thr)*

Classe *ThreadGroup* (1)

- ensemble de threads
- accès aux informations du groupe
- hiérarchie de groupes
- exemple : un ensemble de threads utilisés pour l'impression, un autre pour afficher, un autre pour sauvegarder de données – si l'impression est abandonnée, il serait utile d'arrêter tous les threads qui sont concernés par l'opération

Classe *ThreadGroup* (2)

- constructeurs
  - *ThreadGroup(String name)*
  - *ThreadGroup(ThreadGroup parent, String name)*
- méthodes
  - *int activeCount()* : estimer le nombre de threads actifs du groupe
  - *void destroy()* : détruire ce groupe et ses sous-groupes
  - *int enumerate(Thread[] list)* : énumère les threads actifs du groupe
  - *void interrupt()* : interrompre tous les threads du groupe
  - *void setMaxPriority(int pri)*

Thread - daemon

- exécution infinie en arrière plan
- *void setDaemon(boolean)*
  - switch user/daemon
- *boolean isDaemon()* - test
- JVM est arrêtée si les seuls threads actifs sont des threads daemon