Programmation Concurrente

Notion de thread

Polytech/INFO 4, 2022-2023

Fabienne Boyer
UFR IM2AG, LIG, Université Grenoble Alpes
Fabienne.Boyer@imag.fr

Notion de thread

- **Flôt d'exécution qui s'exécute au sein d'un processus**
  - Contexte "léger"
    - Une partie propre : pile, registres
    - Une partie *mutualisée* : mémoire adressable, fichiers ouverts, …

- **Commutations rapides**

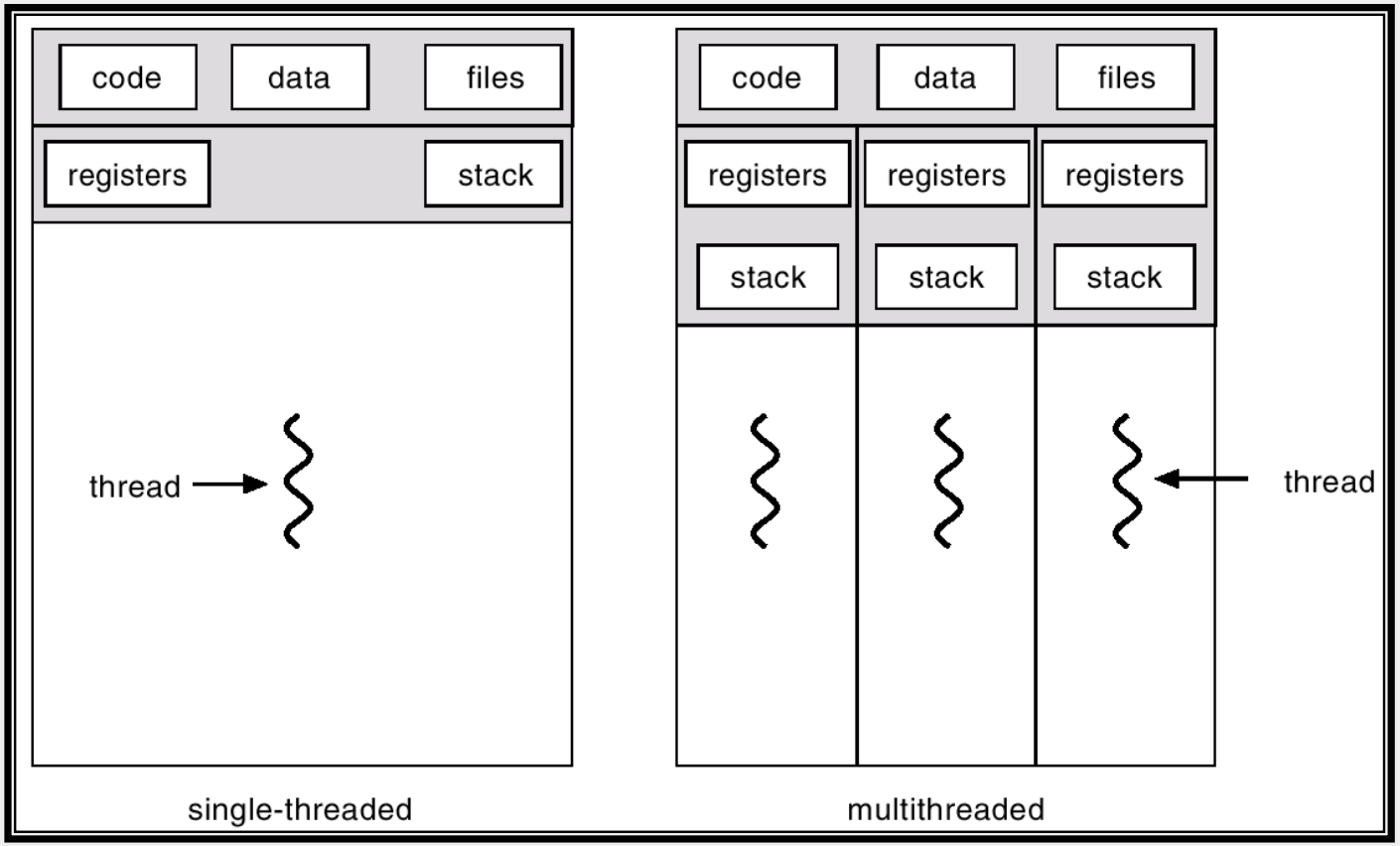
→ **Efficacité d'exécution**

- **Partage de mémoire**

→ **Facilité de programmation des applications concurrentes**

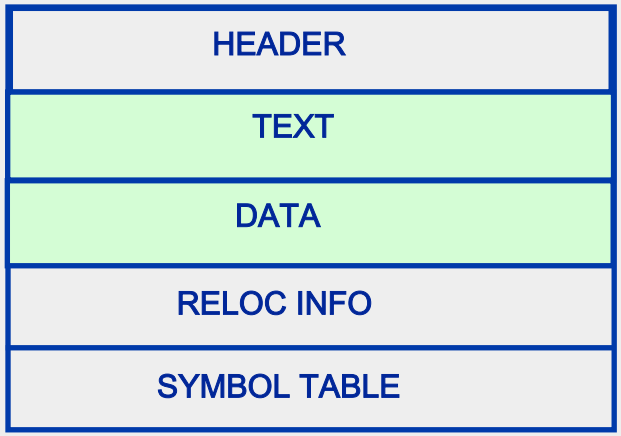
Processus single-threaded et multi-threaded

Schéma typique d'application concurrente

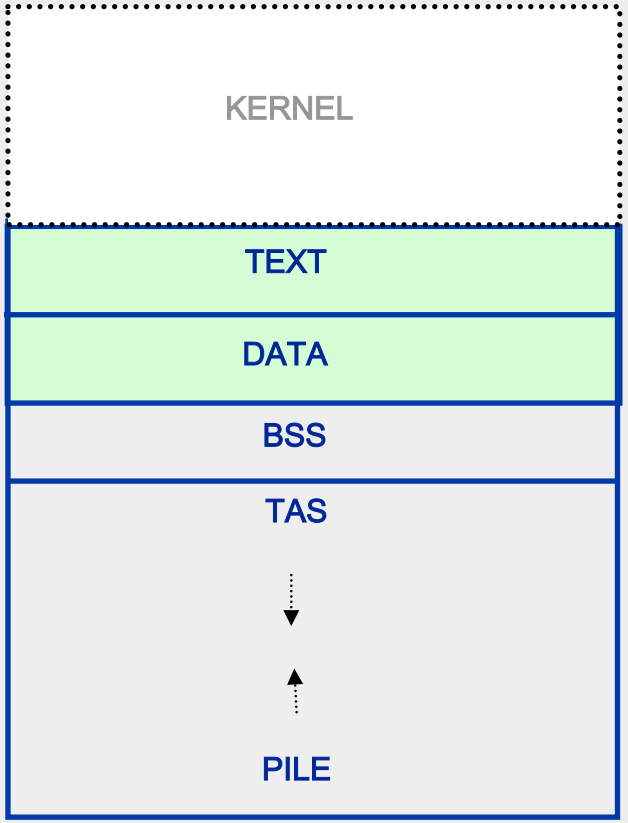


A.Sylberschatz

(rappel) Espace d'adressage d'un processus

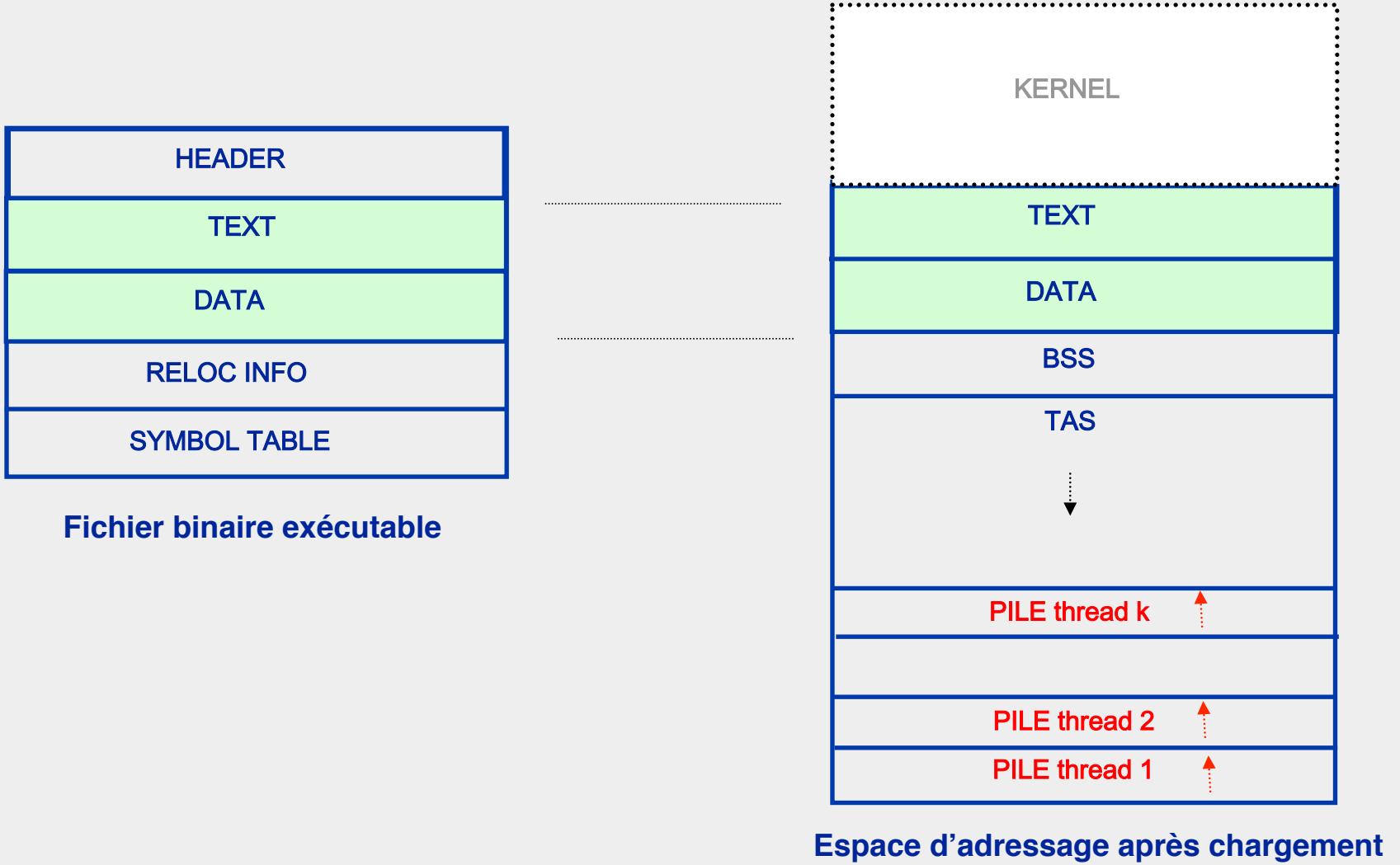


Fichier binaire exécutable



Espace d'adressage après chargement

Espace d'adressage d'un processus *multi-threadé*



Application concurrente

- **Multi-processus ou multi-threads?**
  - Besoin d'isolation mémoire et/ou fautes

    → multi-processus
  - Besoin de partage mémoire et/ou d'efficacité

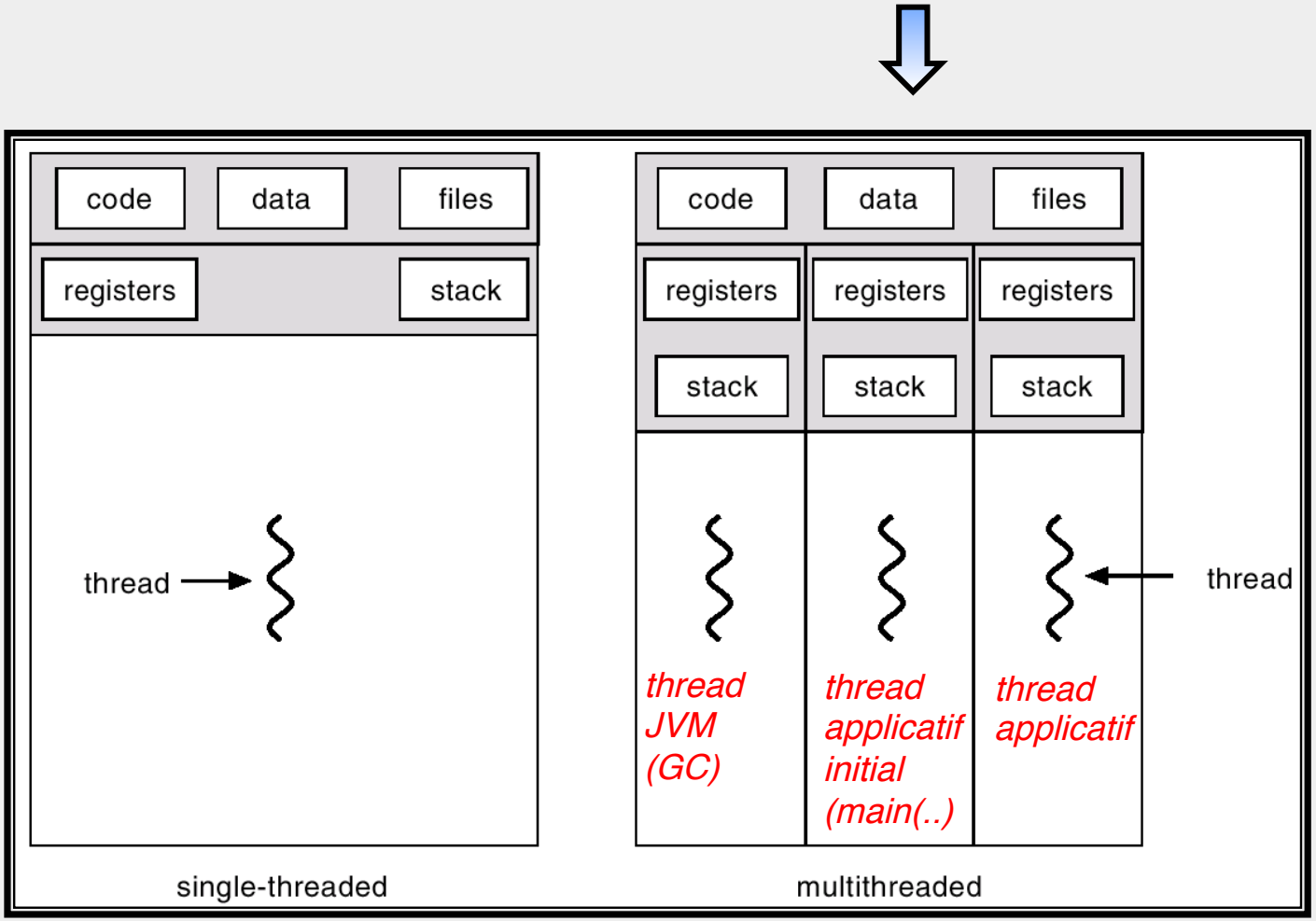
    → multi-threads

Exemple / Machine Virtuelle Java

> java MyClass ..

- Le shell crée un processus (*fork*)
- Ce processus appelle *execv (/usr/bin/java, “MyClass”)*
- Le thread initial exécute le point d’entrée (méthode *main*) du binaire *java*
- Ce point d'entrée crée des threads pour gérer des tâches de fond de la JVM (ex: garbage collector)
- Puis il charge la classe passée en argument
- Il invoque la méthode statique *main(..)* sur la classe chargée
- Ce thread peut à son tour créer d'autres threads applicatifs

Machine Virtuelle Java



A.Sylberschatz

Exemple basique avec les pthreads (threads POSIX)

```
#include <stdio.h>
#include <pthread.h>

static void *task_a (void *p_data) {
   puts ("A  says: Hello world ");
}

static void *task_b (void *p_data) {
   puts (”B says: Hello universe");
}
```

```
int main (void) {
   pthread_t ta;
   pthread_t tb;

   puts ("main init");
   pthread_create (&ta, NULL, task_a, NULL);
   pthread_create (&tb, NULL, task_b, NULL);

   pthread_join (ta, NULL);
   pthread_join (tb, NULL);

   puts (“that’s all folk guys");
   return NULL;
}
```

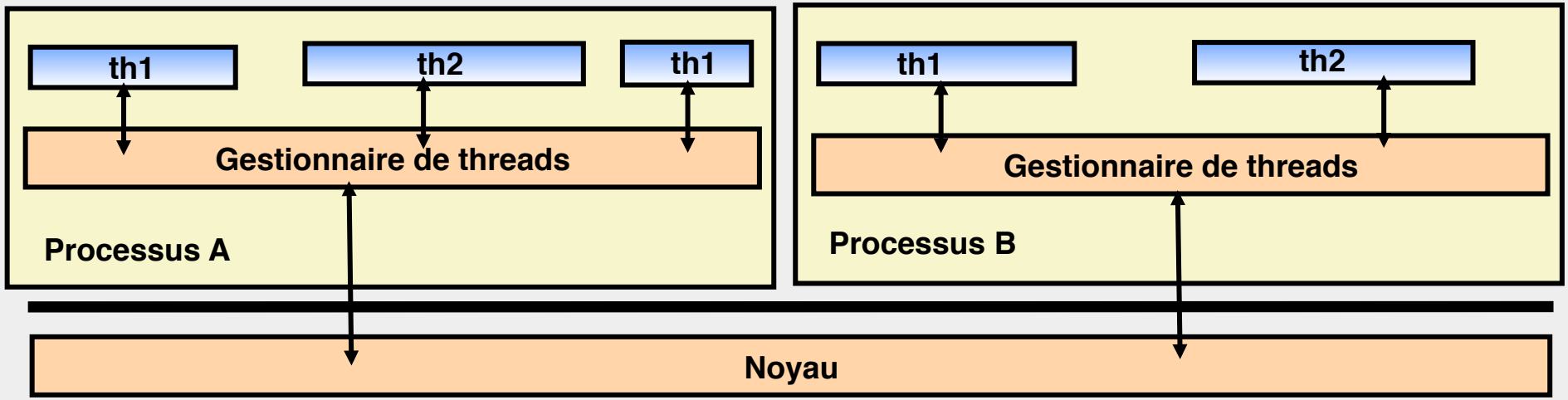
Implantation des threads

- **3 types d'implantation des threads**
  - Threads User-level
  - Threads Kernel-Level
  - Solutions mixtes

Threads *user-level*

- Code de gestion des threads dans une librairie
- Pas de modification du noyau
- Le gestionnaire de threads ainsi que les threads s'exécutent au sein d'un processus utilisateur

Ex: Mach C-threads, Solaris threads, Threads Java (initialement)



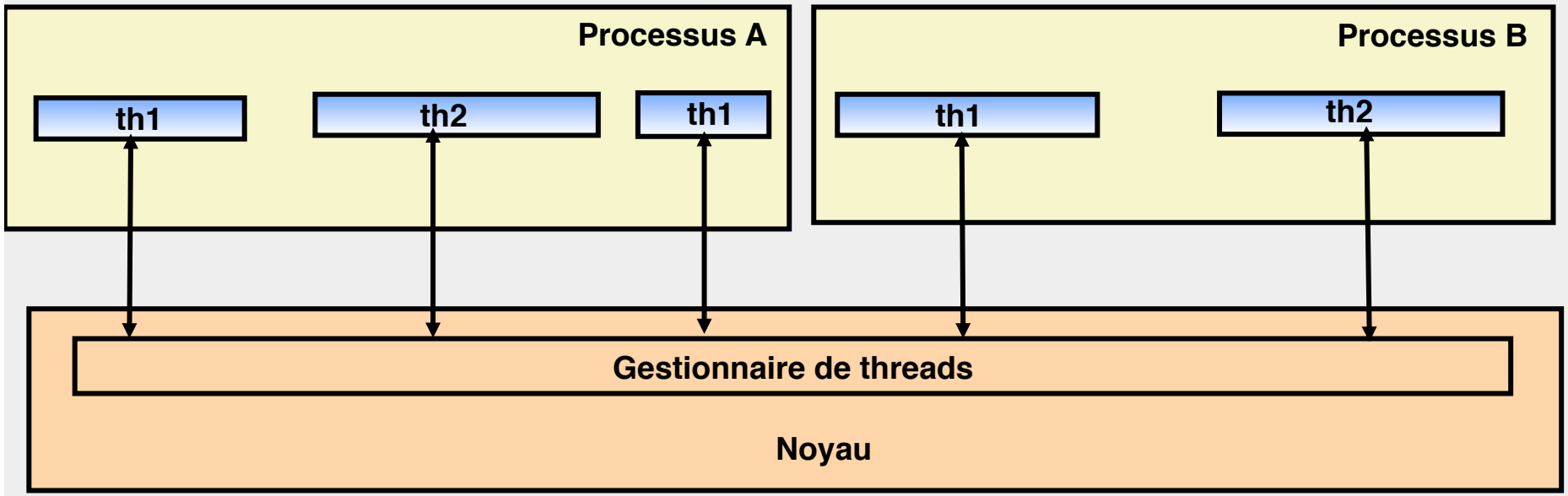
Avantages et inconvénients des threads
User-level

- **Efficacité (+)**
  - La commutation de contexte est rapide
- **Pas de parallélisme (-)**
  - Pas de parallélisme réel entre les threads d'un EV
- **Appels systèmes bloquants (-)**
  - Le processus est bloqué au niveau du noyau
  - Tous les threads sont bloqués tant que l'appel système (ex: I/O) n'est pas terminé
  - Il n'y a donc pas de multiprogrammation dans l'application concurrente

Threads *kernel-level*

- Notion de thread gérée par le noyau
- Lorsqu'un thread se bloque, le noyau alloue le processeur à un autre thread

Examples: Windows 95/98/NT/2000, Solaris, Tru64 UNIX, Linux, Threads Java (HotSpot)



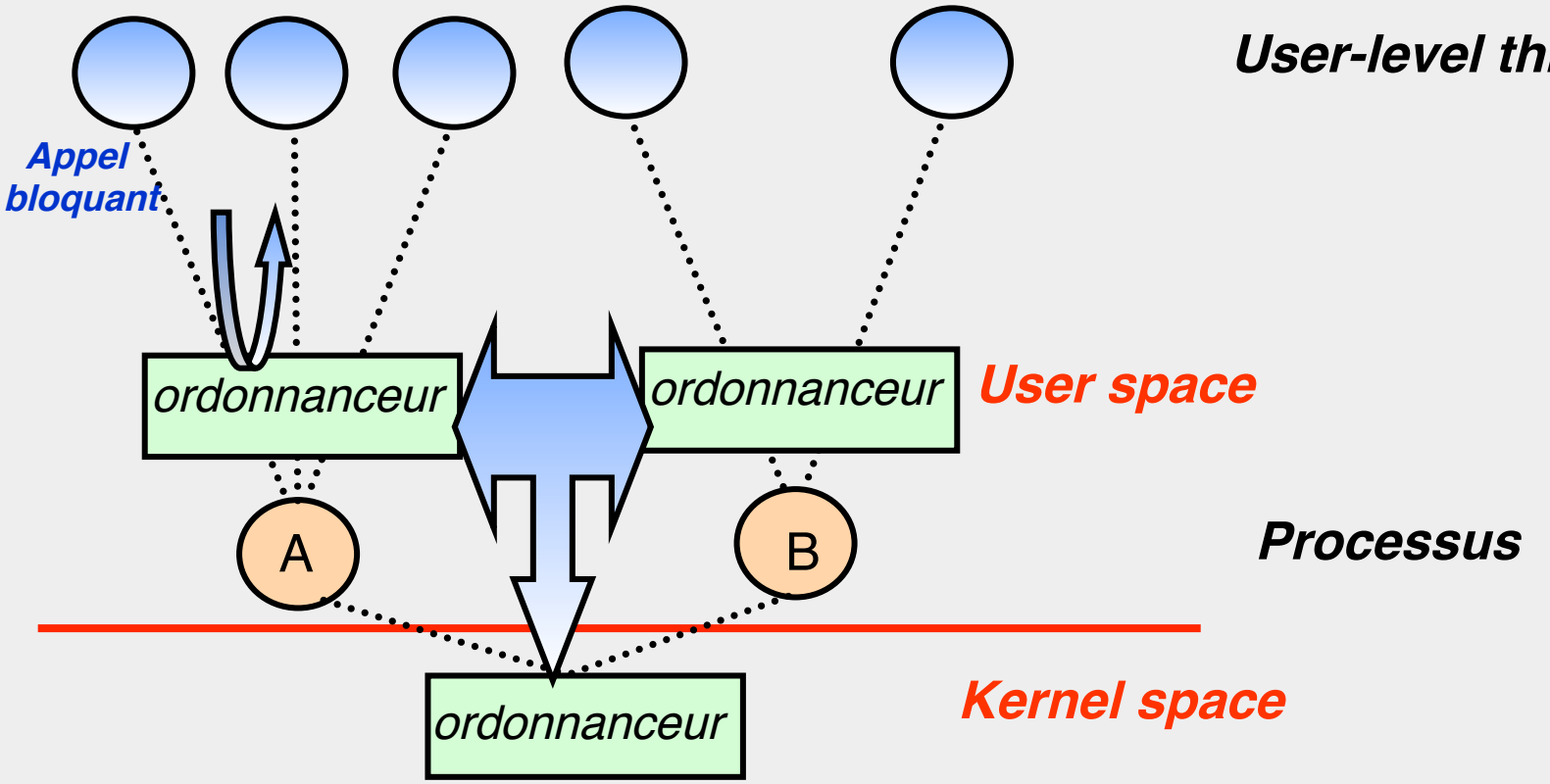
Avantages et inconvénients des threads
Kernel-level

- **Gestion des appels systèmes bloquants (+)**
  - Pas de blocage des threads d'une application concurrente lors d'un appel système
- **Parallélisme réel (+)**
  - N threads d'une application concurrente peuvent s'exécuter sur K processeurs
- **Moins efficace (-)**
  - Commutation plus chère que pour les threads User-level, car demande un passage en mode noyau.

Solutions hybrides permettant une commutation légère

- **Principes des threads User-level (commutations gérées au niveau utilisateur)**
- **Modification du noyau pour gérer les appels systèmes de manière à ne pas bloquer les threads**
  - Lorsqu'un thread fait un appel système bloquant, le noyau ne préempte pas le processeur
  - Un mécanisme de signaux permet de gérer la fin de l'appel bloquant
  - L'ordonnanceur est départagé entre l'espace système et l'espace utilisateur (ordonnanceurs coopérants)

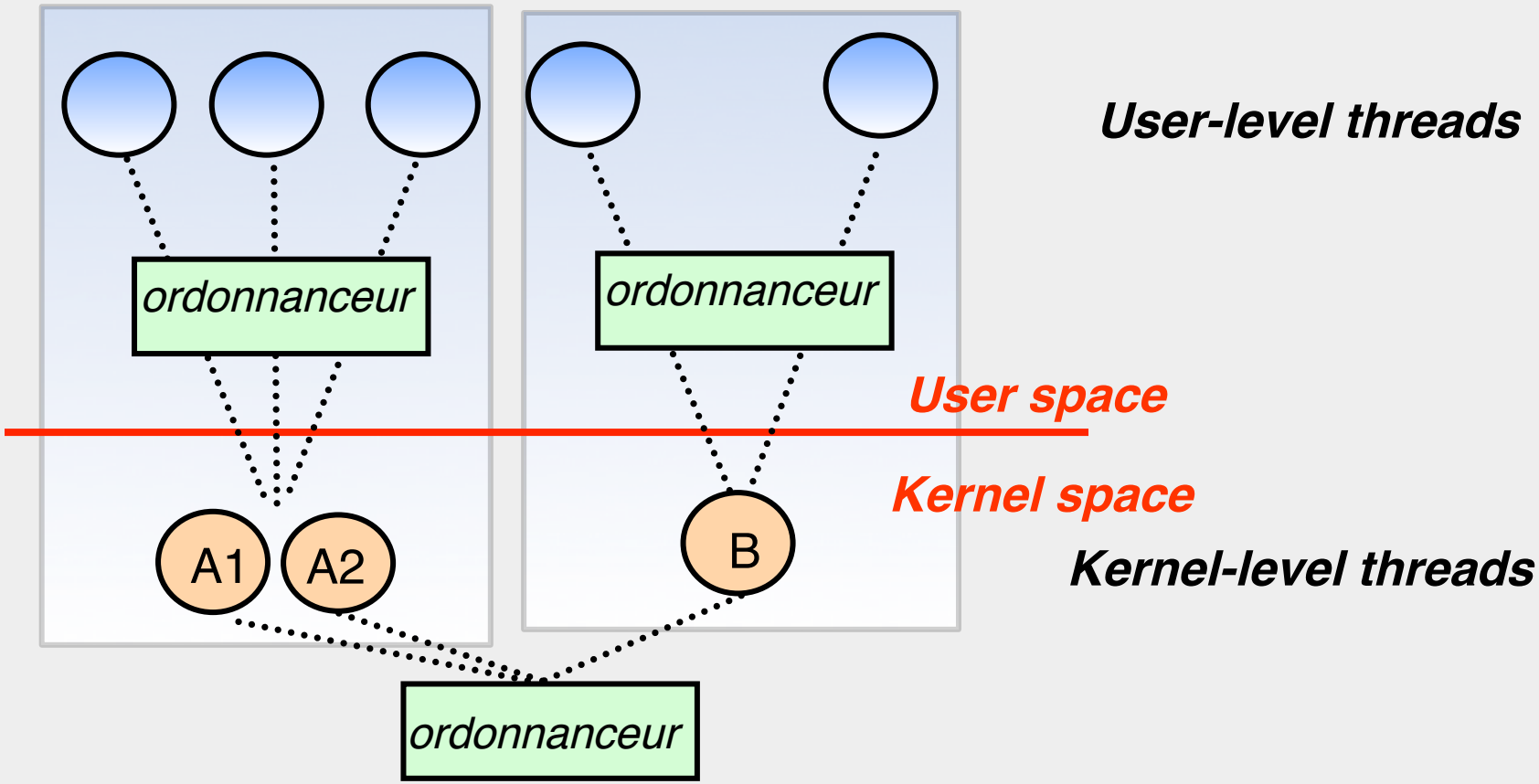
Solutions hybrides permettant une commutation légère



Autres solutions hybrides permettant un parallélisme réel

- **On utilise les deux types de threads**
- **Les threads Kernel-level fournissent la capacité de parallélisme**
- **Les threads User-level fournissent la capacité d'avoir des commutations légères**
  - On peut associer plusieurs threads User-level à un thread kernel-level

Autres solutions hybrides permettant un parallélisme réel

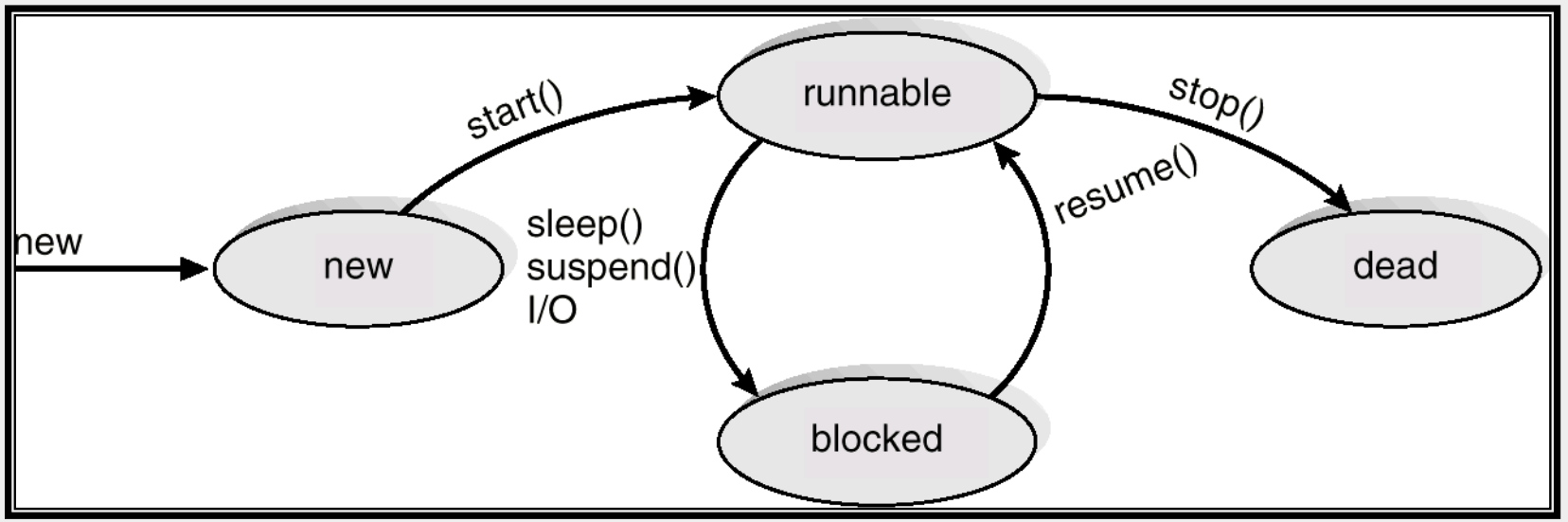


Principales familles de threads

- **Systèmes Unix**
  - POSIX Threads (IEEE POSIX 1003.1c-1995 standart) (appelés Pthreads, threads kernel)
  - DCE Threads
  - Solaris Threads
- **Microsoft-style threads (PCs)**
  - Win32 (Microsoft Windows 95 and Windows NT)
  - OS/2 threads (IBM)

Threads Java

- User-level / Kernel-level, selon l'implémentation de la JVM (généralement kernel-level)
- Fixed Priority Scheduling (round-robin basé sur priorités)



Threads Java

- **Deux méthodes de création**
  - Par héritage (extension de la classe Thread)
  - Par délégation (implémentation de l'interface Runnable)

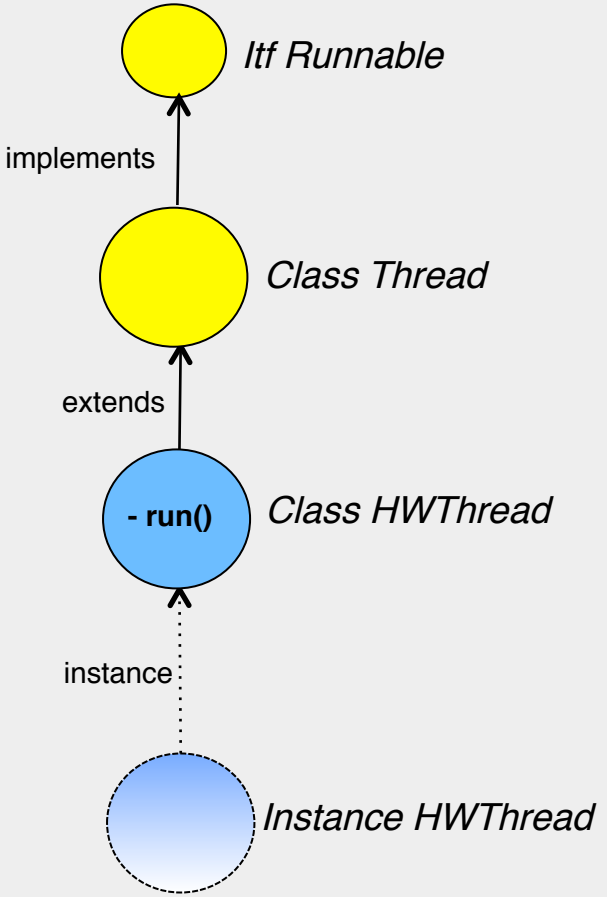
    interface Runnable { public abstract void run(); }

- **Dans les deux cas**
  - Programmer la méthode *run()* qui définit les instructions exécutées par le thread

Définition et création de threads Java par héritage

```
// Defining a thread class
public class HWThread extends Thread {
        public void run() {
            System.out.println("Hello World");
            …
        }
}
```

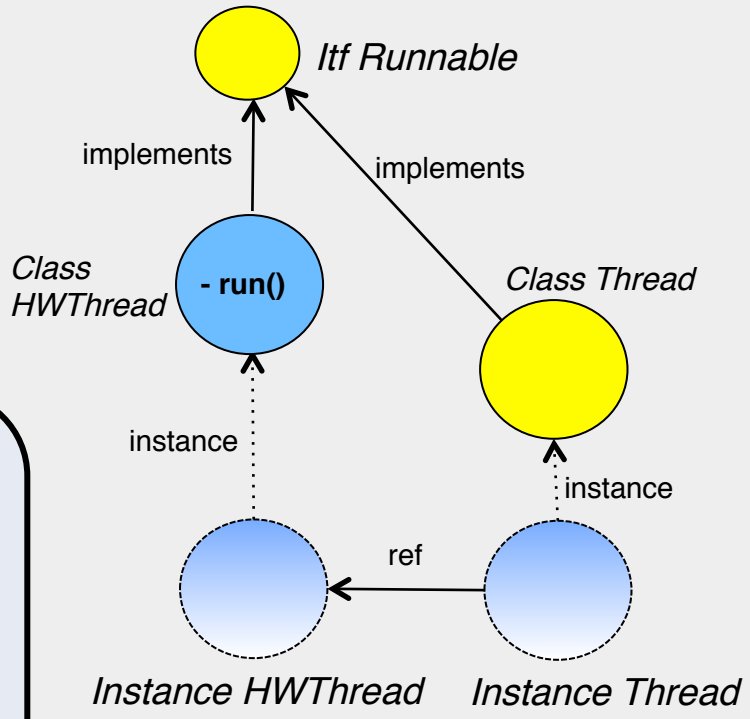
```
// Creating and starting a thread of class HWThread
HWThread hwt = new HWThread ();
hwt .start();
// Waiting for the completion of the created thread
hwt .join();
…
```



Définition et création de threads Java par délégation

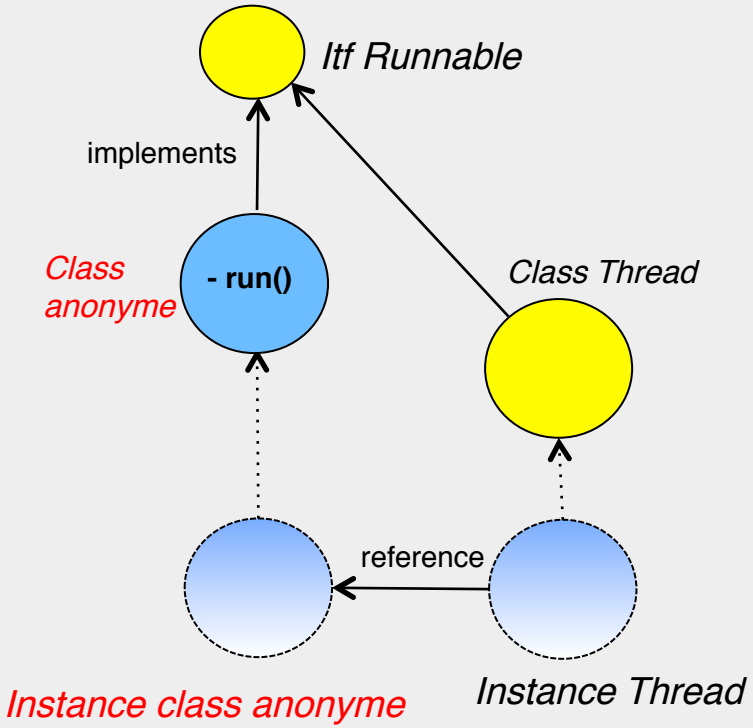
```
// Defining a thread class
public class HWThread implements Runnable {
        public void run() {
            System.out.println("Hello World");
            …
        }
}
```

```
// Creating and starting a thread of class HWThread
….
Thread hwt= new Thread(new HWThread());
hwt.start();
….
hwt.join();
…
```



Définition et création par délégation via une classe anonyme

```
...
new Thread(
  new Runnable() {
      public void run() {
              System.out.println("Hello World"); .. }
  }).start();
...
```



Exemple

```
Collection c = ..;
Thread[] threads = new Thread[c.size()];
…
for (final Object o : c) {
    threads[i] = new Thread(new Runnable(){
                public void processObj(Object o) {  …  }
                public void run(){ processObj(o); }
      };
    threads[i].start();
    i++;
}
for (int j=0; j<I; j++)
    threads[j].join();
System.out.println("done");
…
```

Manipulation de threads Java

- **Un thread peut manipuler un autre thread s'il possède sa référence**
- **Des méthodes permettent de gérer le cycle de vie des threads**
- **Méthodes d'instances et méthodes statiques**

```
public class Thread implements Runnable {
 …
public Thread();
public Thread(String name);

public static Thread currentThread();
public static void sleep(long ms);
…
public void start();
public void join ();
…
```

Que fait un thread tout au long de sa vie ?

- **Il invoque des méthodes sur des objets (instances ou classes)**
- **Un thread manipule les objets qu'il a crée ou dont il a obtenu la référence**
  - Il suffit de partager un objet pour pouvoir échanger des données (références ou valeurs primaires)
  - Deux threads partagent un objet O s'ils possèdent tous les deux une référence vers O
  - Lors de leur création, les threads d'une même application peuvent recevoir en argument la référence d'un objet partagé initial

Terminaison d'un thread Java

- **Naturelle**
  - Le thread a fini d'exécuter sa dernière instruction
- **Par interruption**
  - Invocation de la méthode *interrupt()* sur le thread
  - Si celui-ci est dans une méthode bloquante, il sort de la méthode avec une *InterruptedException*
  - Sinon, interrupt() ne fait que positionner un drapeau (flag) dans le thread et le thread concerné est censé tester régulièrement ce drapeau :

    if (Thread.interrupted()) …

Terminaison d'un programme Java

- **La machine virtuelle Java s'arrête quand tous les threads ont terminé leur exécution**
  - A l'exception des threads de type démon (méthode setDaemon() fournie par la classe Thread)
  - Concept fourni pour faciliter la gestion des threads qui exécutent une boucle infinie

```
// Example of a thread that prints the CPU usage every 10 ms
public void run(){
    while(true){
       printCPUUsage();
       sleep(10);
}
…
```

Point crucial à retenir

- **Une application concurrente (ex Java)**
  - Est composée d'un ensemble de threads qui partagent des objets
  - Ces threads sont gérés en multi-programmation et temps-partagé, ils peuvent donc être commutés n'importe quand
- **Il faut donc faire attention**
  - Un objet peut devenir incohérent si deux threads le modifient en parallèle