Les processus et les threads
Processes and Threads

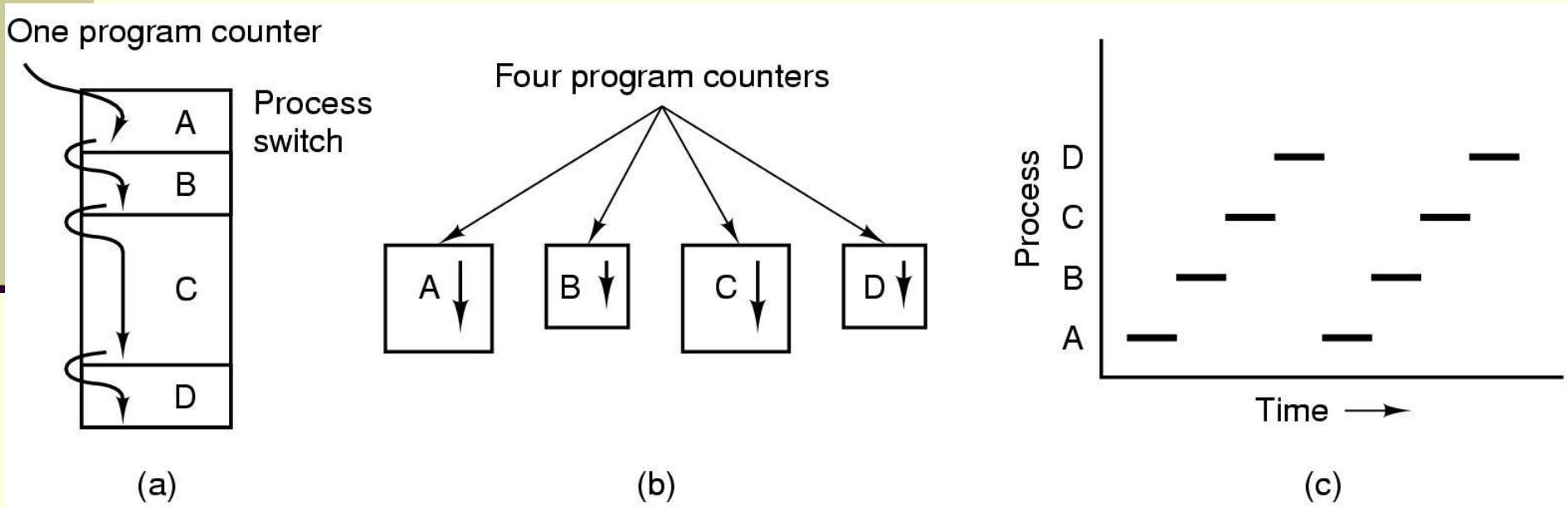
Grégory Seront
Institut Paul Lambin
E-mail: gregory.seront@ipl.be

Les Processus

- Un OS exécute plusieurs programmes “en même temps”.
- Vraiment « en même temps? »

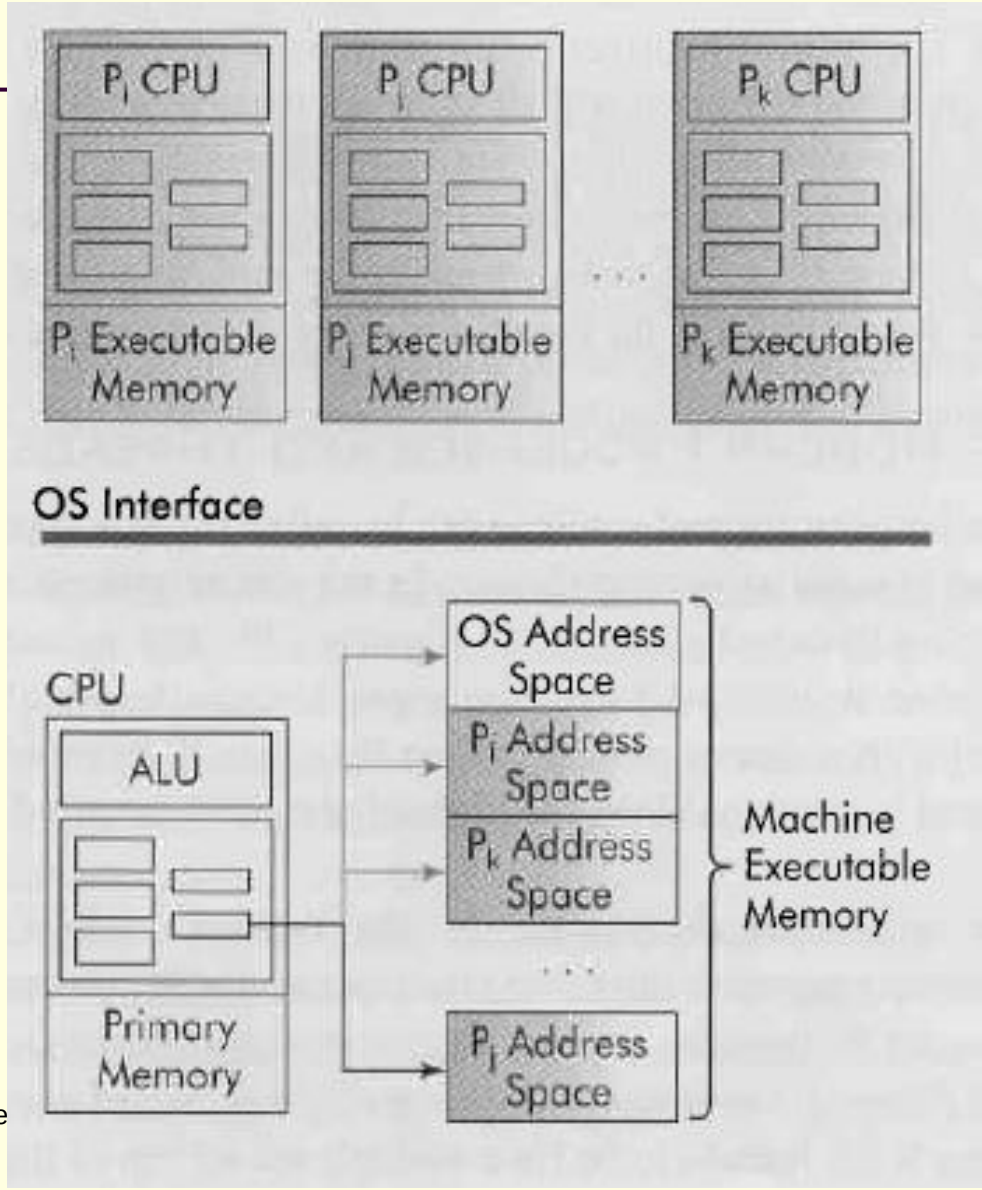
Pseudo Parallélisme

- Pas « en même temps »
- En passe de l'un à l'autre (context switching)



Abstraction de la machine hardware

- Chaque programme pense qu'il est seul sur la machine
  - Seul sur le CPU
  - Seul dans la mémoire
  - Seul avec les périphériques



Définition processus

- Processus = Abstraction de la machine hardware pour permettre à programme de « croire qu'il est seul ».
- (vue utilisateur)
- Et pour l'OS?

Définition processus

- Pour l'OS:
- Processus = Code à exécuter + ensemble des ressources utilisées + état de la mémoire et du processeur (+droits+propriétaire+...)

Descripteur de processus

- On a donc dans l'OS une table de processus contenant ces informations
- Cette table contient une entrée par processus
- Entrée = process control block (PCB)

Attributs Processus?

Attributs Processus?

Process management	Memory management	File management
Registers	Pointer to text segment	Root directory
Program counter	Pointer to data segment	Working directory
Program status word	Pointer to stack segment	File descriptors
Stack pointer		User ID
Process state		Group ID
Priority		
Scheduling parameters		
Process ID		
Parent process		
Process group		
Signals		
Time when process started		
CPU time used		
Children's CPU time		
Time of next alarm		

Attributs Processus?

Gestion du processus	Gestion de la mémoire	Gestion de fichier
Registres	Pointeur vers un segment de texte	Répertoire racine
Compteur ordinal	Pointeur vers un segment de données	Répertoire de travail
Mot d'état du programme	Pointeur vers un segment de la pile	Descripteurs de fichiers
Pointeur de la pile		ID utilisateur
État du processus		ID du groupe
Priorité		
Paramètres d'ordonnancement		
ID du processus		
Processus parent		
Groupe du processus		
Signaux		
Heure de début du processus		
Temps de traitement utilisé		
Temps de traitement du fils		
Heure de la prochaine alerte		

Création des processus

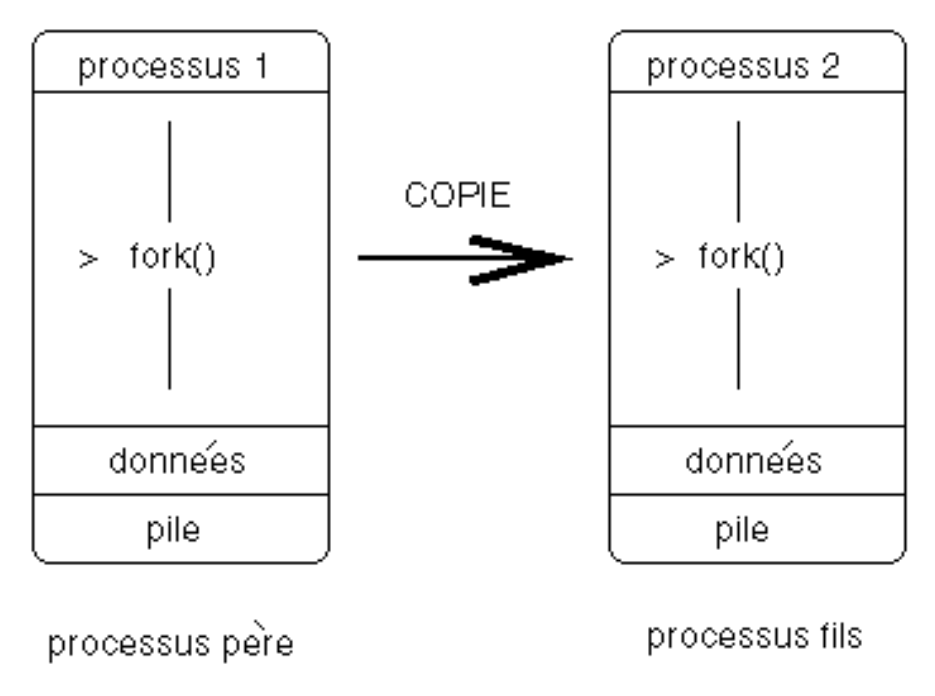
- Quand?

Création des processus

- Au démarrage de la machine
  - Task manager

Création des processus

- Par un autre processus via un appel système
  - Unix: fork() (clone le processus courant)
  - Windows: createProcess()



Création des processus

- Par l'utilisateur via le shell ou la GUI
- Le shell est en fait un processus. Il va faire un appel système pour créer le process.

Fin d'un processus

- Quand?
- Arrêt normal (fin programme)
- Arrêt sur erreur (plantage => interruption)
- Arrêt par un autre processus
  - Kill (unix)
  - TerminateProcess (windows)
- Arrêt via shell ou task manager

Hiérarchie des processus

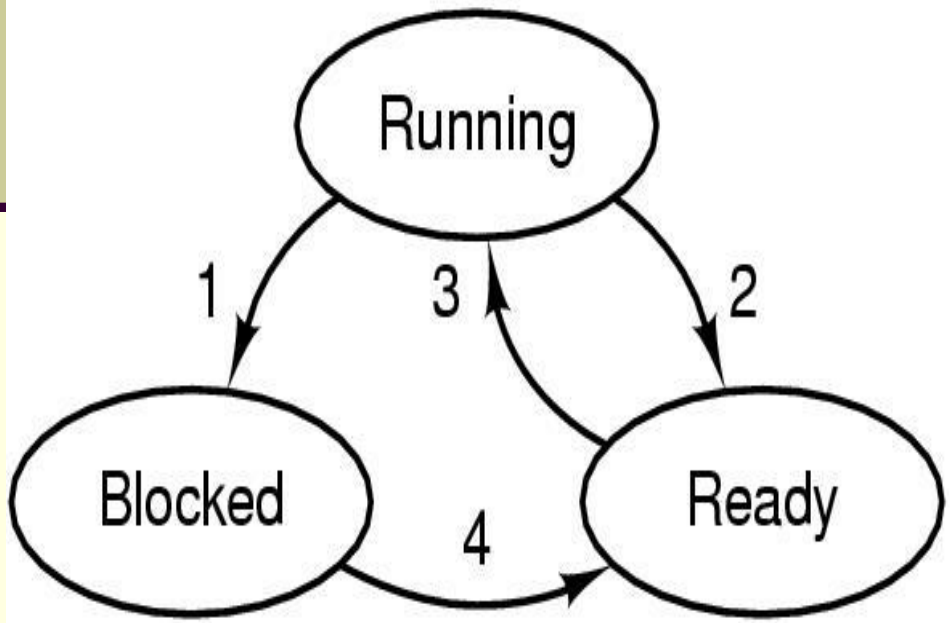
- En Unix on retient le lien Processus Parent – Enfant (pas en windows)
- Parent meurt, tous les enfants meurent

Etats d'un processus

- Processus pas tout le temps actif.
- Quels sont les états?

Etats d'un processus

- Blocage:
  - attente ressources
  - Mis en attente par l'OS
- Déblocage:
  - Ressource dispo
  - OS le choisi



1. Process blocks for input
2. Scheduler picks another process
3. Scheduler picks this process
4. Input becomes available

Alerte ou signal

- Un processus passe d'un état à l'autre (bloqué, débloqué, ou mort) lorsqu'il reçoit un « signal » ou une « alerte »
- Un signal indique
  - Qu'une ressource devient disponible
  - Qu'un timer a expiré
  - Qu'un autre processus veut vous mettre en sommeil ou vous réveiller (ou vous tuer)
  - …

Que se passe-t-il aux changement d'état?

Que se passe-t-il aux changement d'état?

- Context Switching! (encore une fois)

1	Le matériel place dans la pile le compteur ordinal, etc.
2	Le matériel charge un nouveau compteur ordinal à partir du vecteur d'interruption
3	La procédure en langage assembleur sauvegarde les registres
4	La procédure en langage assembleur définit une nouvelle pile
5	Le service d'interruption en C s'exécute (généralement pour lire des entrées et les placer dans le tampon)
6	L'ordonnanceur décide du prochain processus à exécuter
7	La procédure C retourne au code assembleur
8	La procédure en langage assembleur démarre le nouveau processus actif

Ordonnancement

- Quand il y a un changement de process il faut choisir “l'heureux élu” qui le replacera.
- C'est l'Ordonnancement ou Scheduling
- La partie de l'OS qui gère ça est l'ordonnanceur ou scheduler.

Quand est-ce qu'on ordonnance?

1. Création process (père ou fils?)
2. Fin de processus
3. Blocage E/S
4. Interruption (HW) E/S (qui est débloqué?)
5. Interruption horloge (50hz?)
    1. Préemptif
    2. Non préemptif

Objectifs de l'ordonnacement?

- Dépend du but du système
- Objectifs communs
- Traitement par lots (batch)
- Interactif
- Temps réel

Objectifs de l'ordonnancement

All systems

- Fairness - giving each process a fair share of the CPU
- Policy enforcement - seeing that stated policy is carried out
- Balance - keeping all parts of the system busy

Batch systems

- Throughput - maximize jobs per hour
- Turnaround time - minimize time between submission and termination
- CPU utilization - keep the CPU busy all the time

Interactive systems

- Response time - respond to requests quickly
- Proportionality - meet users' expectations

Real-time systems

- Meeting deadlines - avoid losing data
- Predictability - avoid quality degradation in multimedia systems

Objectifs de l'ordonnancement

Tous les systèmes

Équité: attribuer à chaque processus un temps processeur équitable

Application de la politique : faire en sorte que la politique définie soit bien appliquée

Équilibre : faire en sorte que toutes les parties du système soient occupées

Systèmes de traitement par lots

Capacité de traitement : optimiser le nombre de jobs à l'heure

Délai de rotation : réduire le délai entre la soumission et l'achèvement

Utilisation du processeur : faire en sorte que le processeur soit occupé en permanence

Systèmes interactifs

Temps de réponse : répondre rapidement aux requêtes

Proportionnalité : répondre aux attentes des utilisateurs

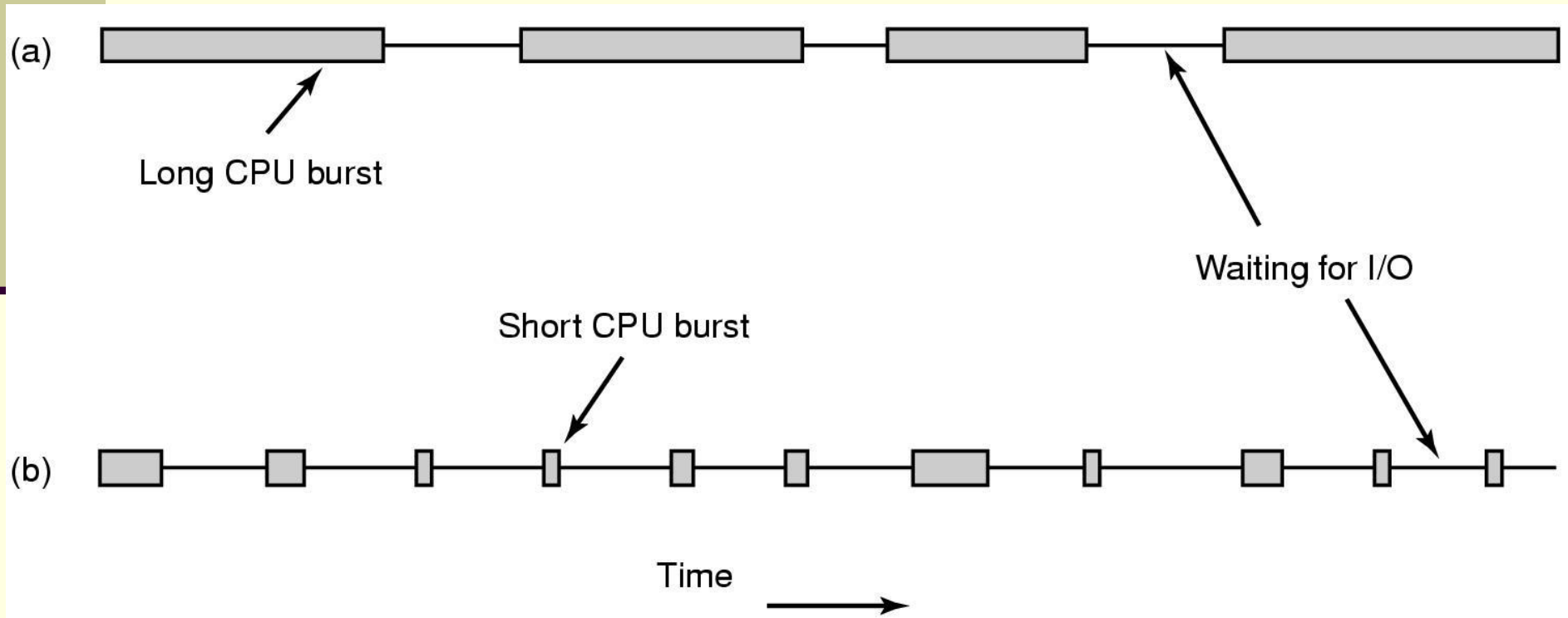
Systèmes temps réel

Respecter les délais : éviter de perdre des données

Prévisibilité : éviter la dégradation de la qualité dans les systèmes multimédias

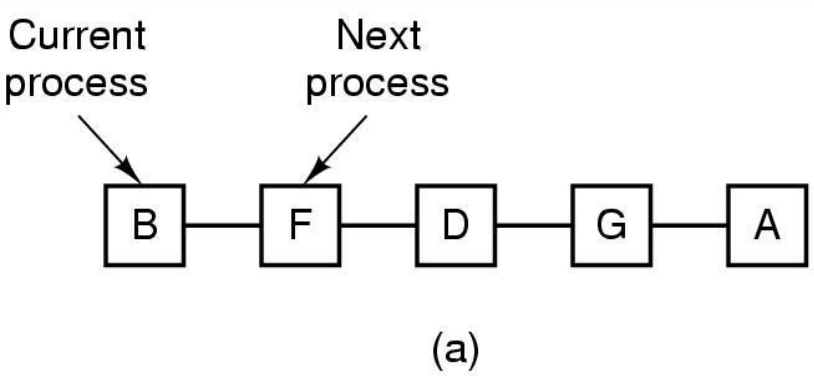
CPU bound vs I/O bound processes

- Processus de traitement vs processus E/S
- Aujourd'hui de + en + vers E/S

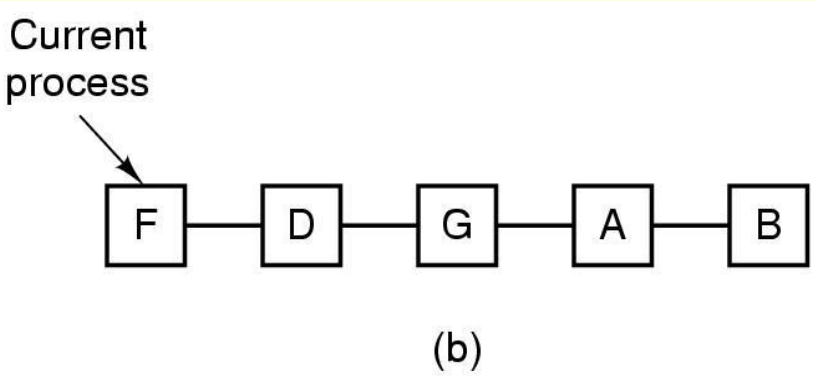


Ordonnancement systèmes interactifs

- Ordonnancement à deux niveaux (mémoire et processeur).
- Temps de réponse!
- Round Robin (tourniquet)
  - On défini un quantum de temps max
  - Passe au suivant si temps épuisé ou process bloqué ou fini



(a)



(b)

Round Robin

- Quelle durée pour le Quantum?
- Dépend de quoi?

Round Robin

- Quelle durée pour le Quantum?
- Dépend de quoi?
- Temps de changement de contexte
  - Ex: si context switching time = 1ms et quantum = 4ms on fait un changement toutes les 5 ms => on perd 20% du temps en switching
- Nombre d'utilisateurs/processus
- Niveau d'interactivité requis
- 20-50 ms bon compromis
- Windows 10: 20ms (2 clock intervals)
- Windows Server: 120 ms [Russ]

Ordonnancement par priorité

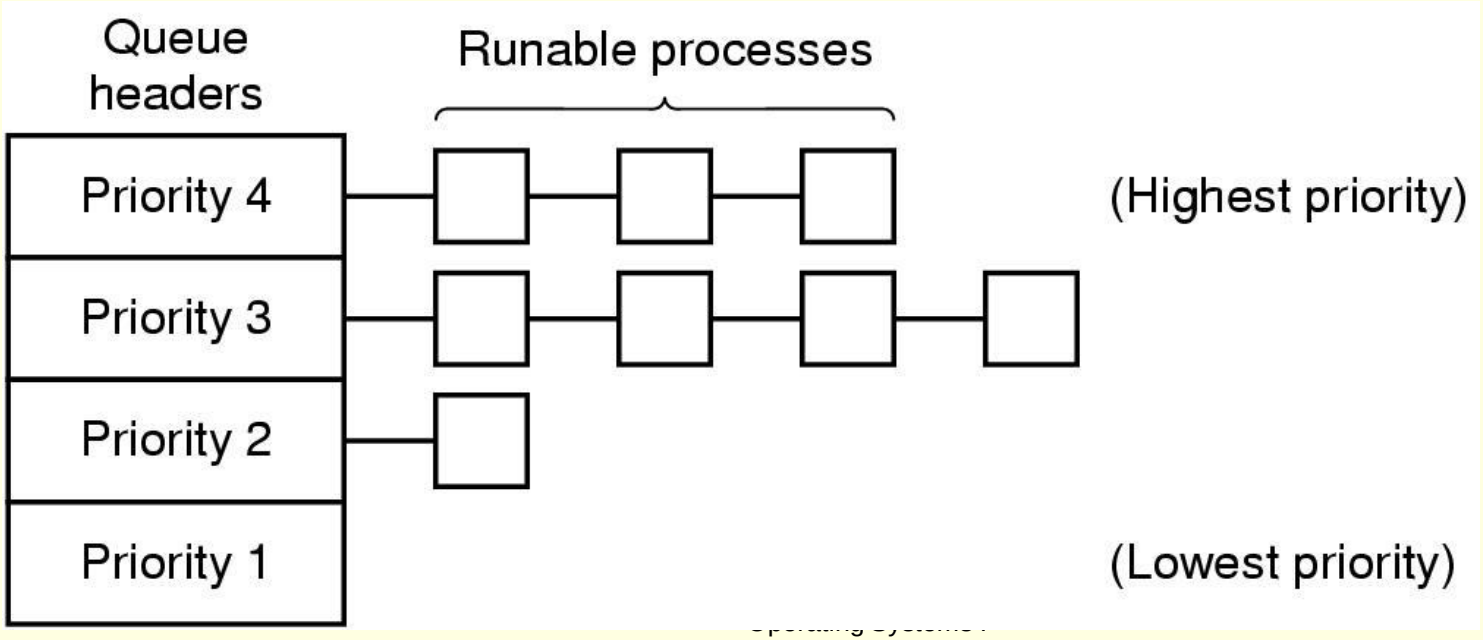
- Tous les processus ne sont pas égaux
- Ex: Envoi mail moins prioritaire que rafraîchissement écran
- Choisi le plus prioritaire lorsqu'on change de processus
- Risque?

Ordonnancement par priorité

- Choisi le plus prioritaire lorsqu'on change de processus
- Risque?
- Certain processus de basse priorité ne seront jamais sélectionnés. Il y a “famine” (starvation).
- Solution?
- On diminue la priorité des processus qui s'exécutent trop (aging)
- On booste la priorié de ceux qui ne s'éxécutent pas assez (Windows)

Ordonnancement par priorité

- Round Robin multiple
- On change les processus de file pour éviter les famines



Les threads

- Problème si il y a beaucoup de changement de processus?

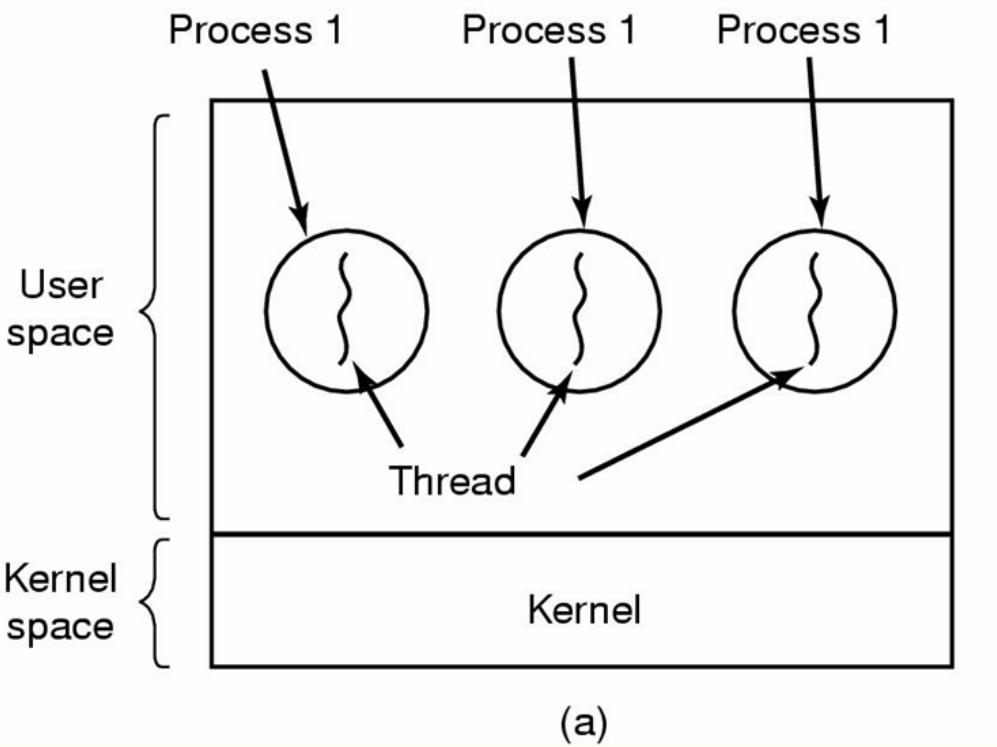
1	Le matériel place dans la pile le compteur ordinal, etc.
2	Le matériel charge un nouveau compteur ordinal à partir du vecteur d'interruption
3	La procédure en langage assembleur sauvegarde les registres
4	La procédure en langage assembleur définit une nouvelle pile
5	Le service d'interruption en C s'exécute (généralement pour lire des entrées et les placer dans le tampon)
6	L'ordonnanceur décide du prochain processus à exécuter
7	La procédure C retourne au code assembleur
8	La procédure en langage assembleur démarre le nouveau processus actif

Les threads

- Problème si il y a beaucoup de changement de processus?
- C’est très lourd! On ralenti la machine

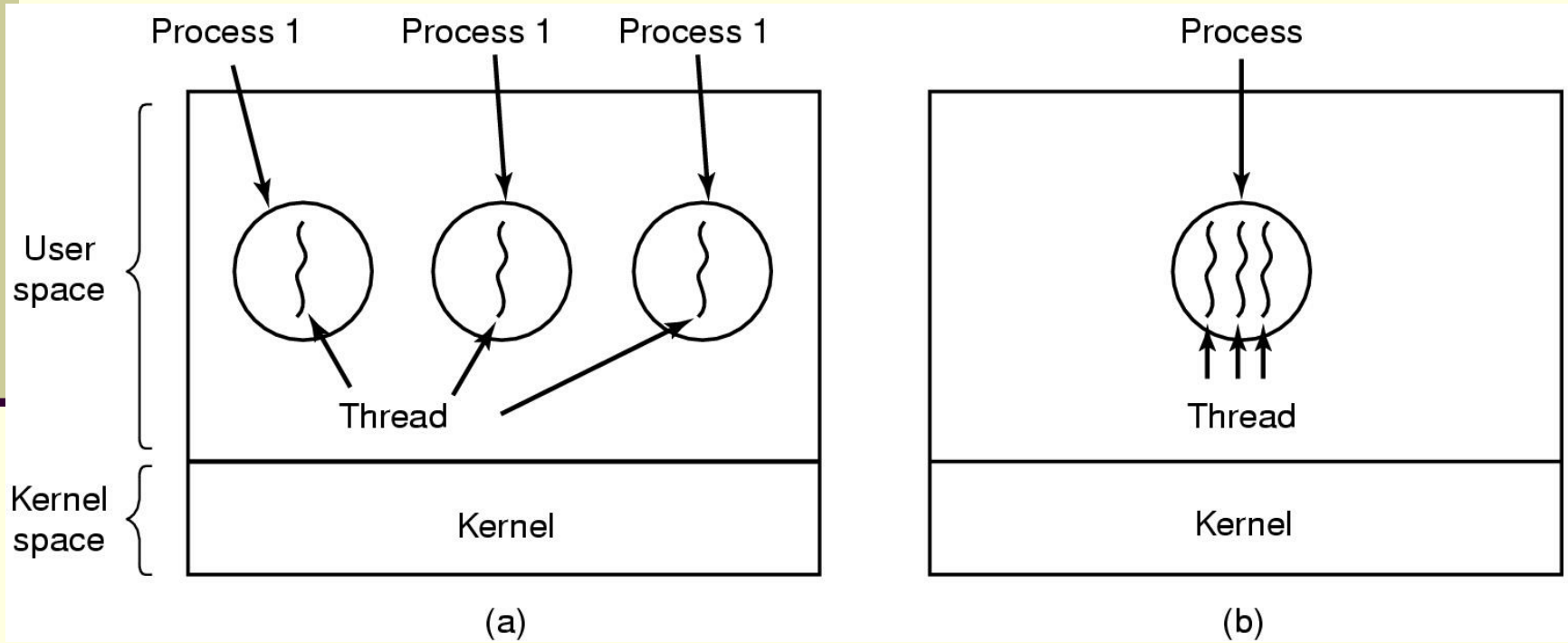
Processus Légers (threads)

- Processus regroupe 2 choses:
  - Les ressources (fichiers, mémoire, droits, …)
  - Le fil d'exécution (fil = thread)



Processus Légers (threads)

- En multi-threading, plusieurs fils d'exécution partagent les même ressources

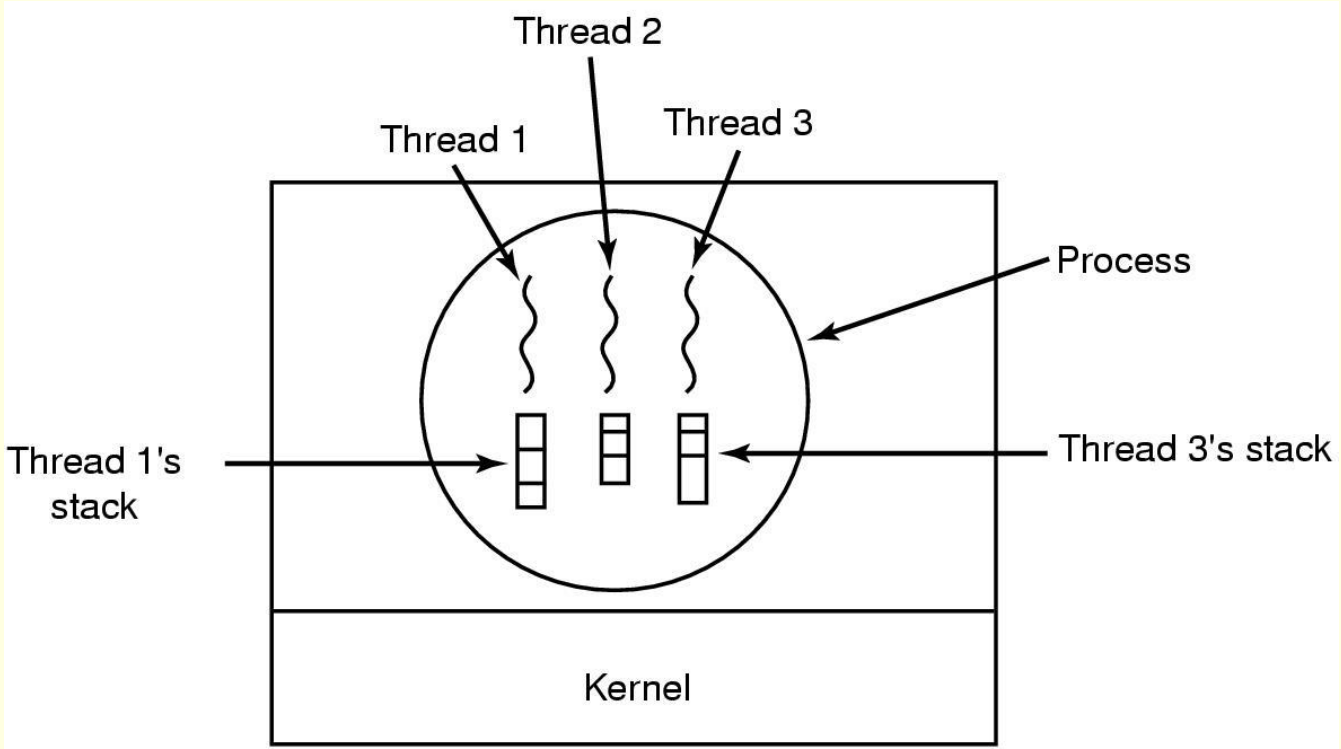


Processus Légers (threads)

- Les threads partagent donc plein de choses:
  - Espace d'adressage mémoire
  - Variable globales
  - Fichiers ouverts
  - Alertes en attente
- Ce qui est privé
  - Compteur ordinal (Program Counter)
  - Registres
  - Pile
  - Etat (bloqué, prêt, en cours d'exécution, terminé)

Un stack (pile) par thread

- Les thread ont des historiques d'exécution différentes
- Ils ont donc besoin de leur propre pile!



Création/Fin des thread

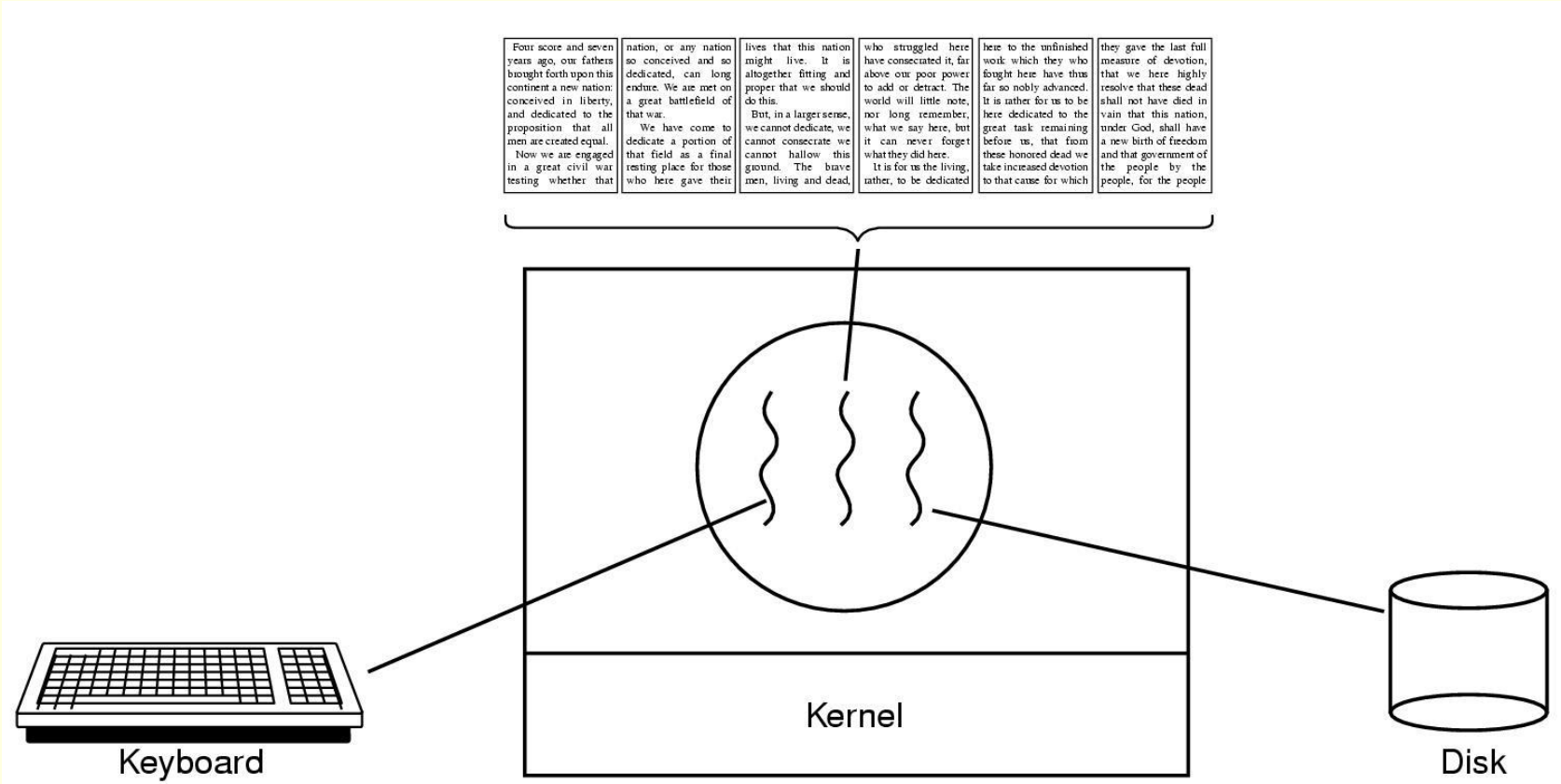
- Au démarrage d'un processus
  - 1 seul thread
- Appels fonctions (Unix)
  - `thread_create(fct_a_executer)`
  - `thread_exit()`
- Dans certains cas les thread sont non préemptibles au sein du processus.
- => ils doivent rendre la main eux-mêmes
  - `thread_yield()`

Problèmes introduits par les threads?

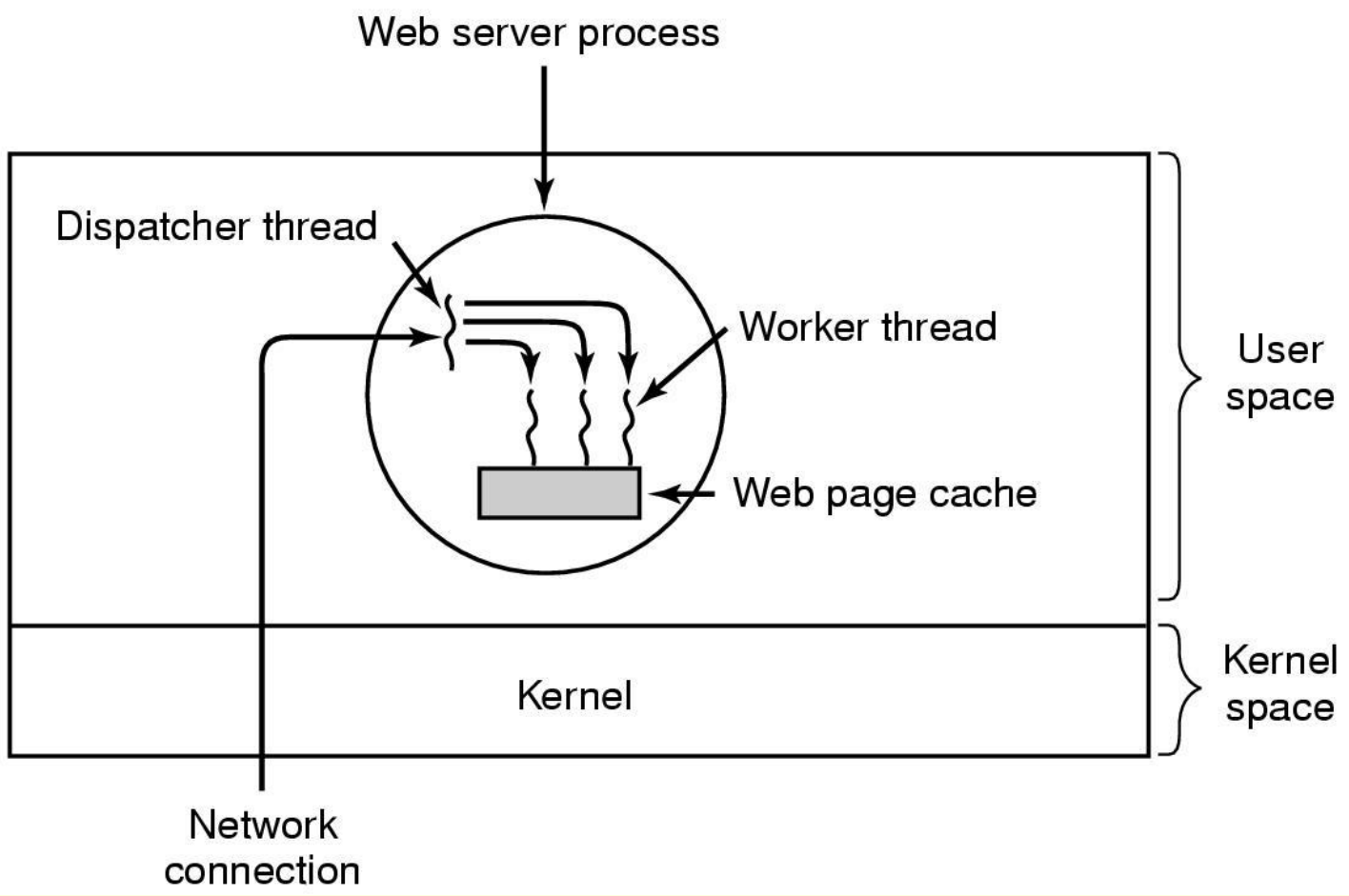
Utilité des threads?

- Création thread 100x plus rapide que processus!
- Systèmes mutli-processeurs
- Architecture logicielle plus simple
  - On pense le traitement en unités indépendantes
  - Pour les applications interactives, on doit moins se soucier des temps de réponse pendant qu'on fait autre chose.

Architecture logicielle multi-threads (traitement de texte)



Architecture logicielle multi-threads (server web)



Architecture logicielle multi-threads (server web)

```
while (TRUE) {
  get_next_request(&buf);
  handoff_work(&buf);
}
```

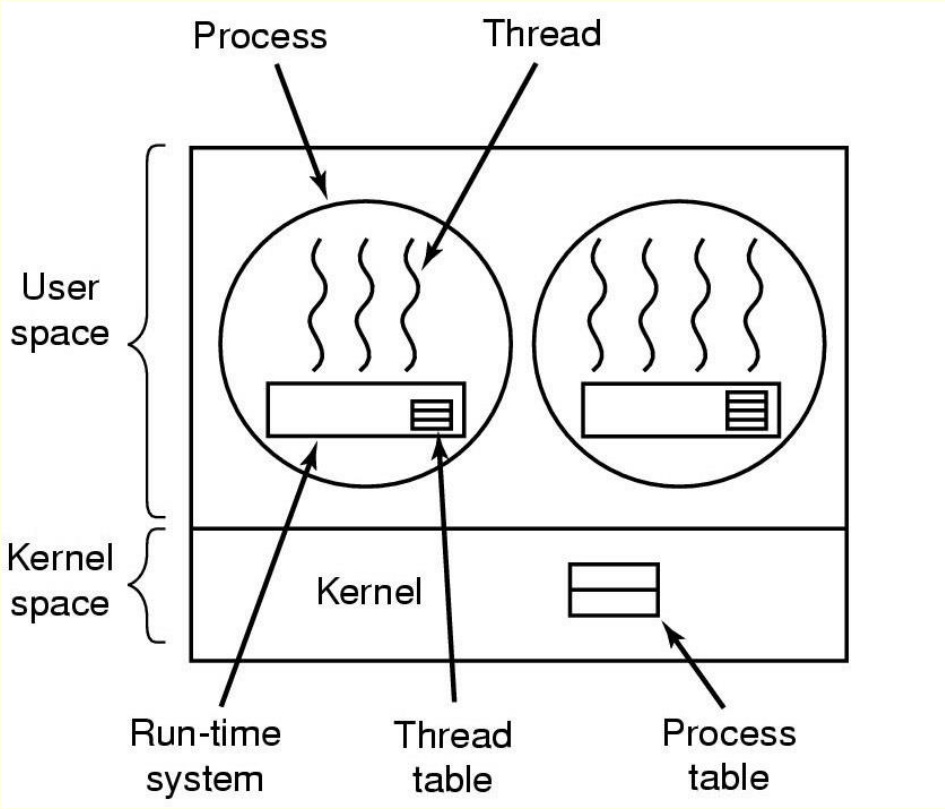
(a)

```
while (TRUE) {
  wait_for_work(&buf)
  look_for_page_in_cache(&buf, &page);
  if (page_not_in_cache(&page)
      read_page_from_disk(&buf, &page);
  return_page(&page);
}
```

(b)

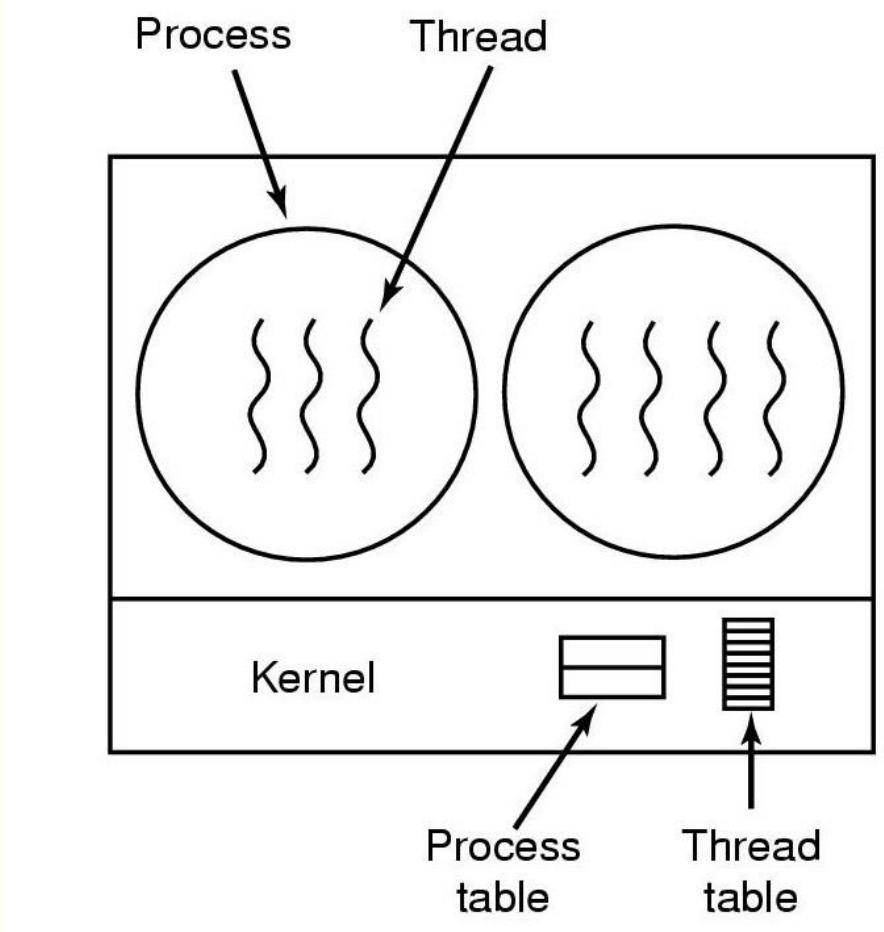
Implémentation des thread dans l'espace utilisateur

- On ne passe pas par l'OS pour gérer les threads.
- On a une librairie run-time qui le fait (ex: Java Virtual Machine)
- Une table des thread par processus
- Appel bloquant => on passe la main à un autre processus



Implémentation des thread dans l'OS (kernel space)

- C'est l'OS qui gère les threads
- On a une table des threads dans l'OS
- Lorsqu'un thread est bloqué, on peut rendre la main à un thread du même process!



Kernel space vs. User space threads

- User space threads :-)
  - Appel plus léger (pas d'appel système => context switching + léger)
  - Contrôle de la politique d'ordonnancement
  - N'utilise pas de ressource OS (nombre de threads potentiellement limité dans l'OS)

Kernel space vs. User space threads

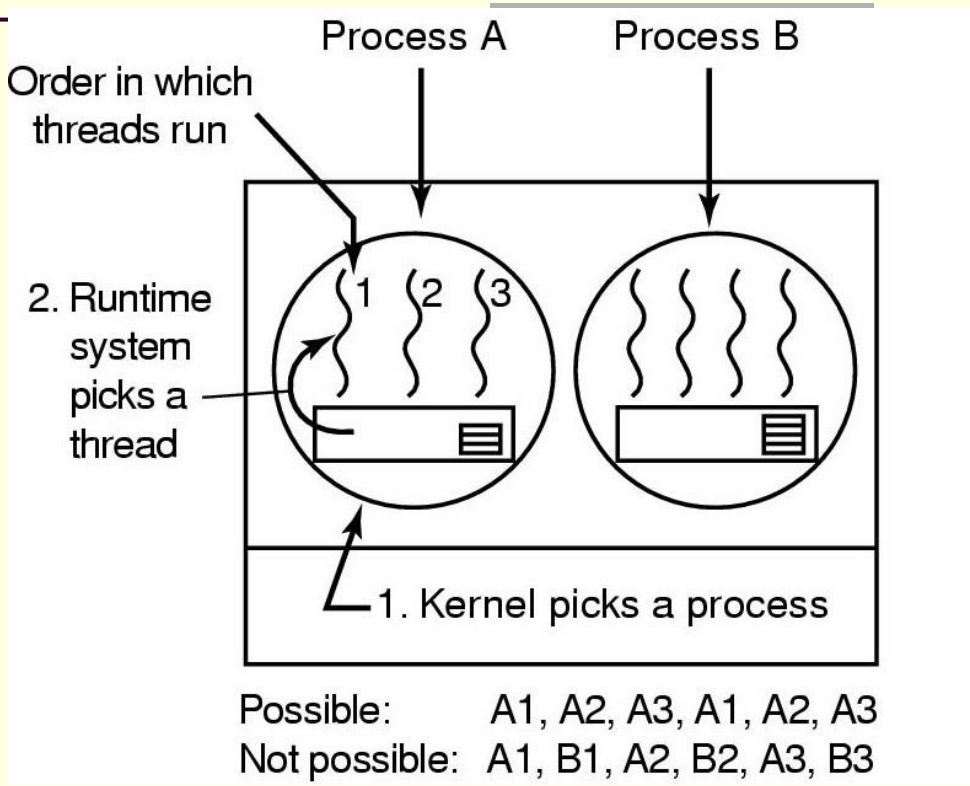
- User space threads :-(
  - Appels système bloquants => on bloque tout le processus
  - Thread mal programmé qui ne rend jamais la main => tout le processus est bloqué
  - La plupart des applications utilisant des threads font justement souvent des appels systèmes bloquant (cf. Web server)
  - => ordonnancement d'un thread n'alourdit pas tellement plus l'appel

Odonnancement des threads

- On a deux niveaux d'ordonnancement
  - Processus
  - Threads
- Ordonnancement sera différent selon qu'on implémente les threads dans le noyau ou dans l'espace utilisateur (kernel space vs. user space threads)

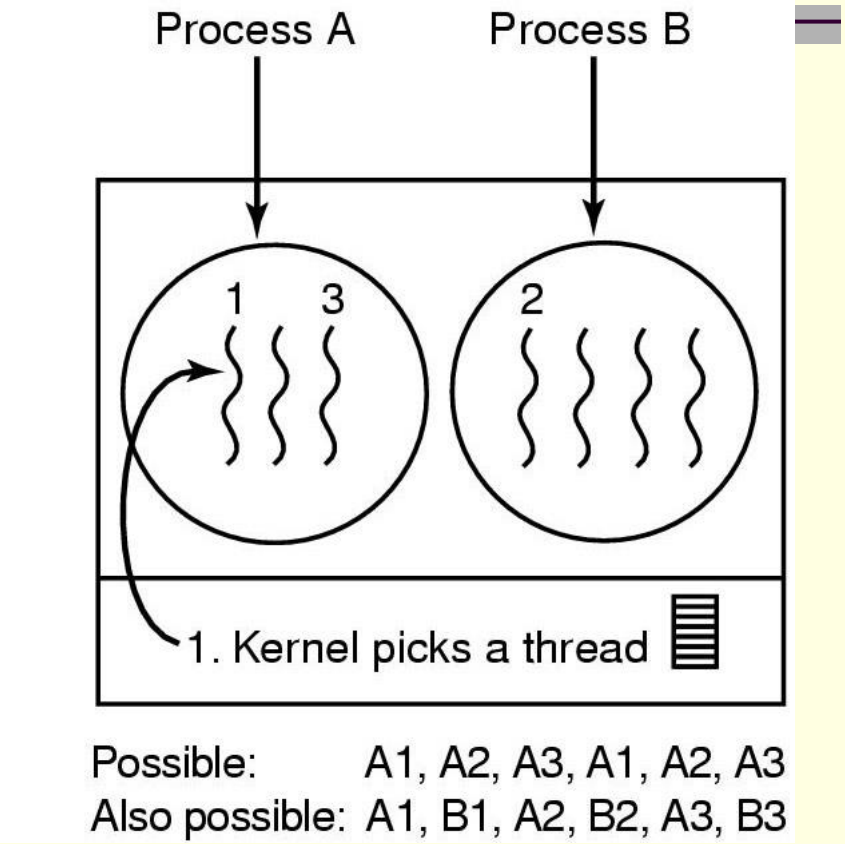
Ordonnancement de user space threads

- L'OS alloue un quanta au processus
- Le run-time du processus réparti entre les threads
- Ex: Quanta process 50 ms et threads dépensent 5 ms chacun
- Algorithmes ordonnancement au choix de processus

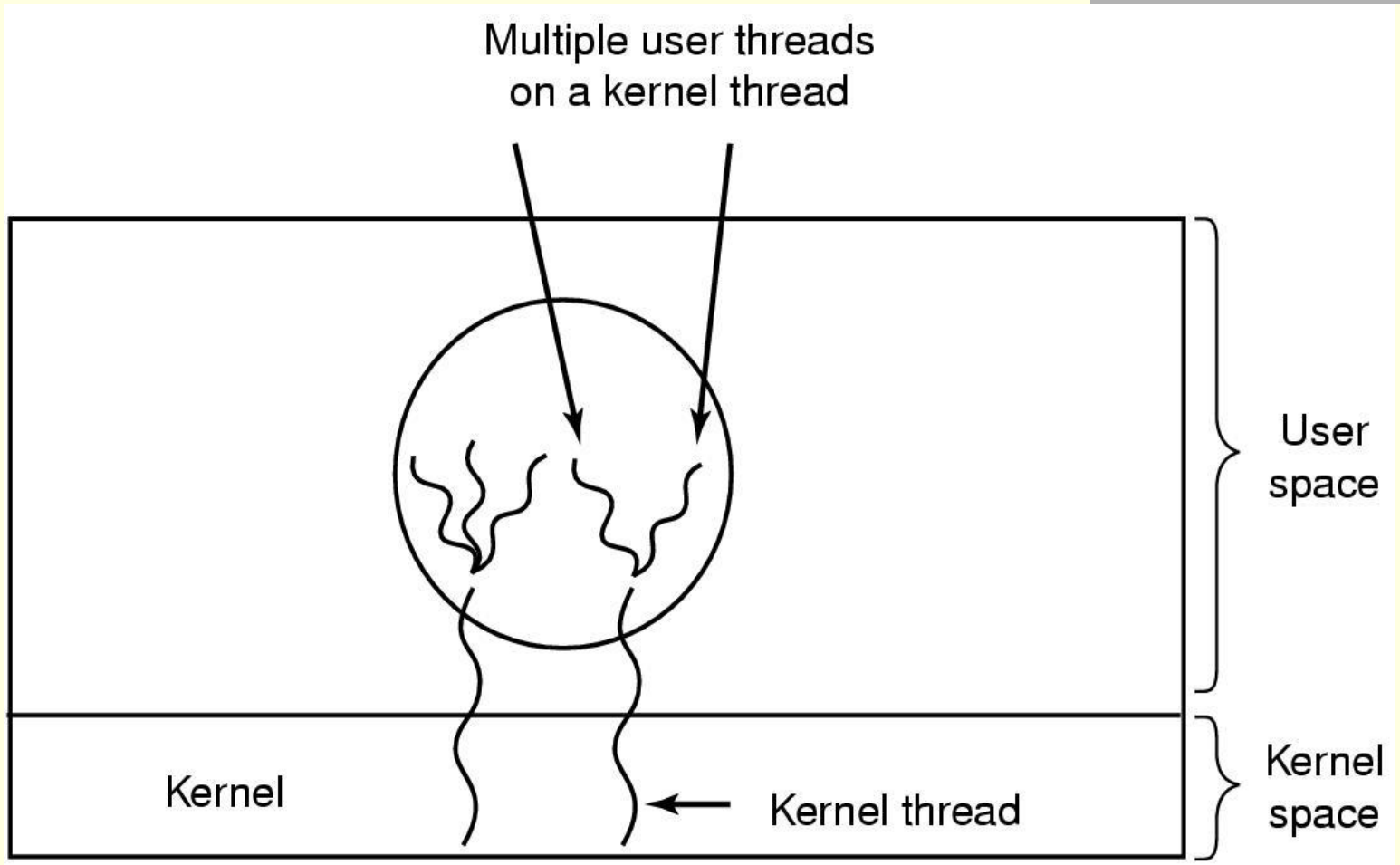


Ordonnancement de kernel space threads

- L'OS fait l'ordonnancement au niveau des threads
- Ex: Quanta process 50 ms et threads dépensent 5 ms chacun

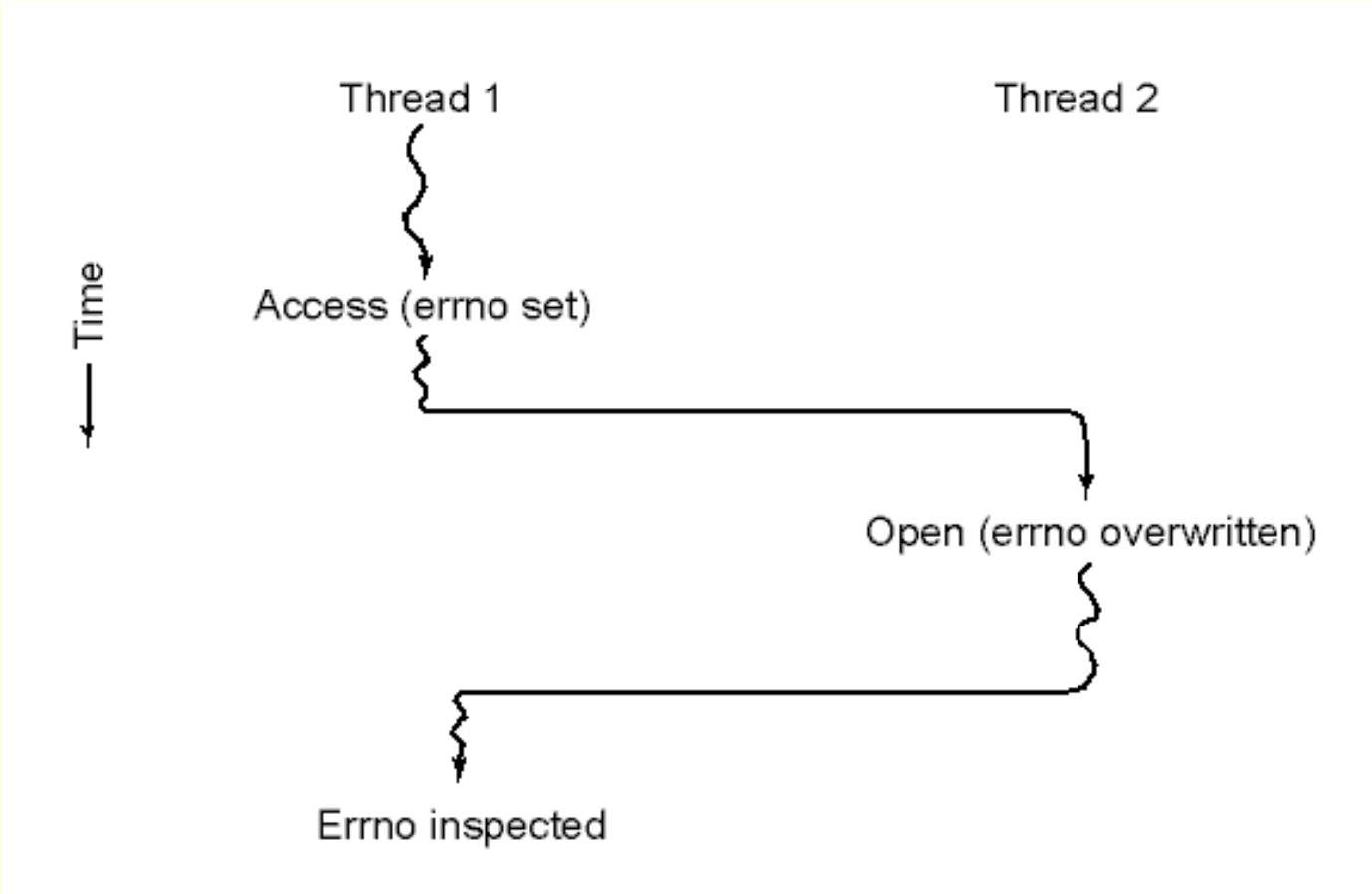


Implantation hybrides

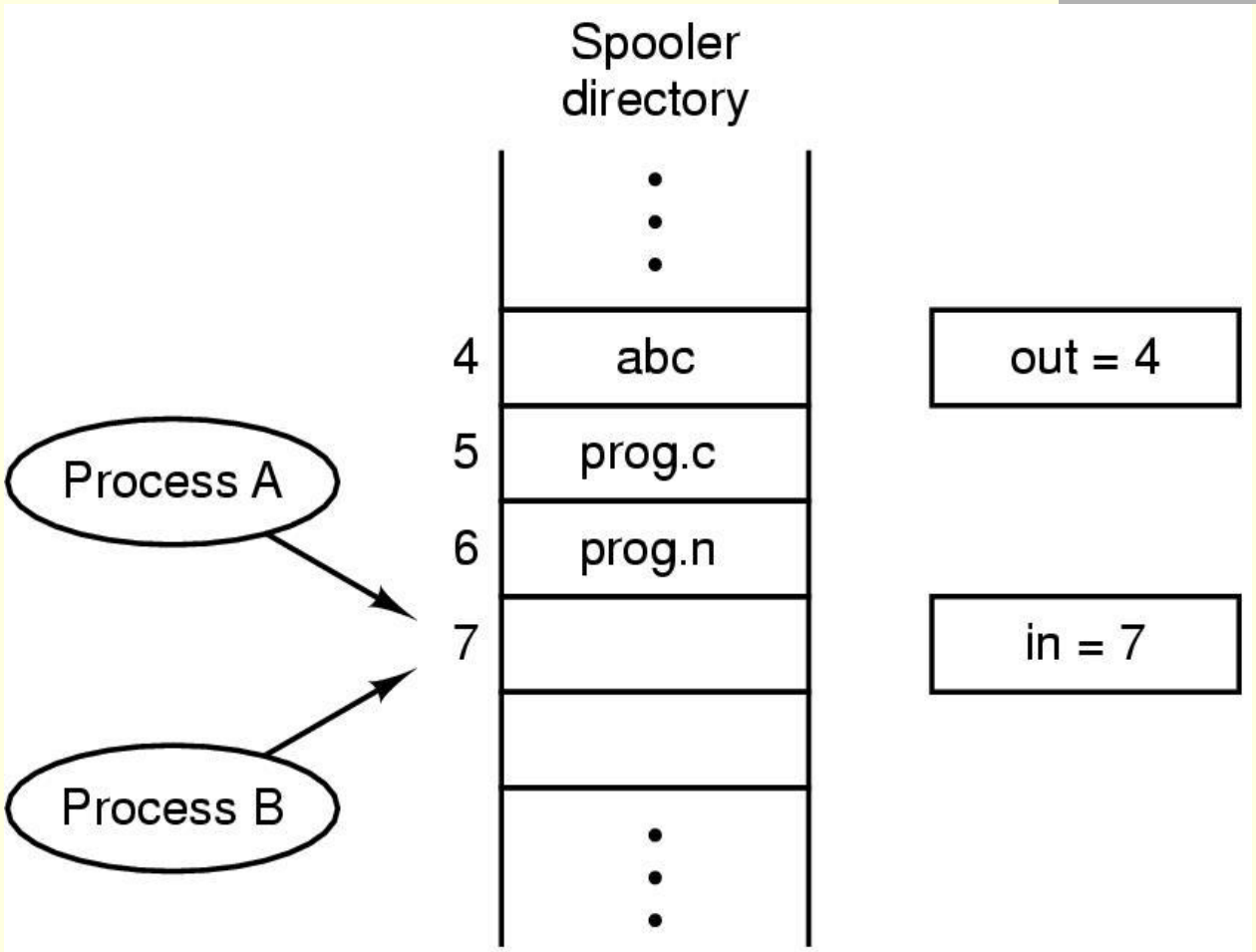


Programmation multi-threads

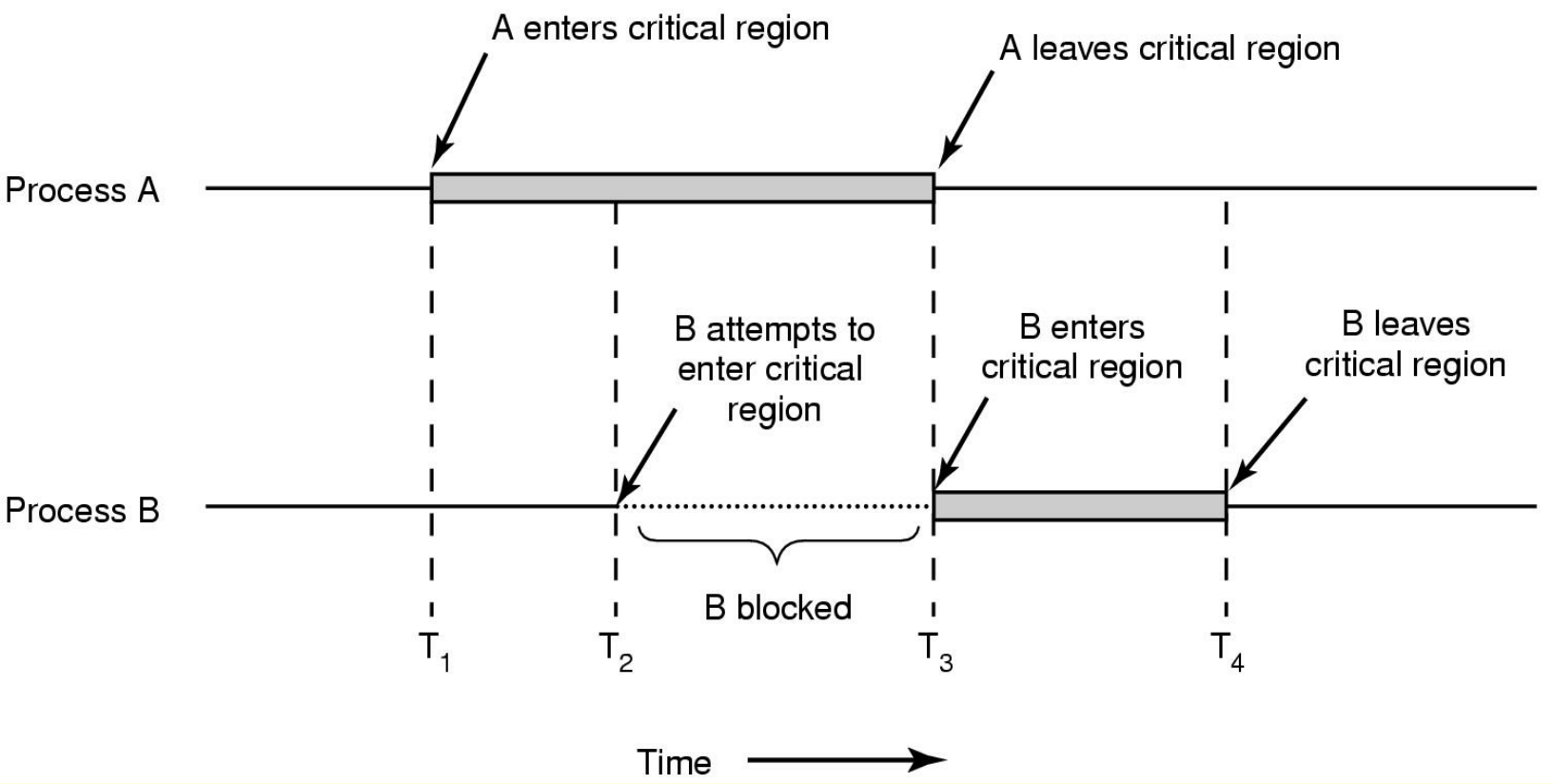
- Attention aux variables globales!
- Variables locales OK



Conditions de "course" (race conditions)



Sections critiques

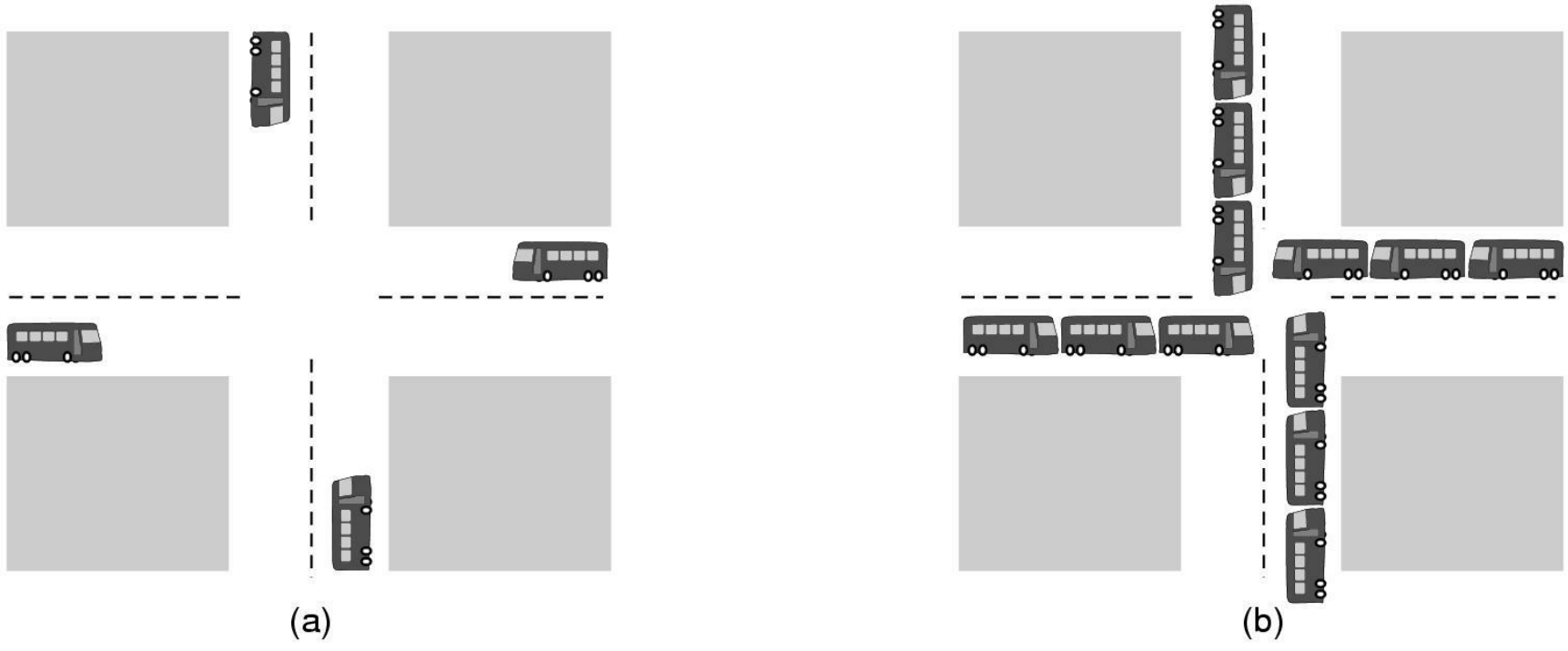


Petit exemple Java

Deadlock

- Les processus ne sont pas tous seul dans l'ordinateur.
- Ils partagent des ressources.
- Source potentielle de blocage
- Ex: Graveur CD et lecteur de cartouche
- Process A et B veulent tous deux graver un CD à partir d'une cartouche (pas la même).

Deadlock



Petit exemple Java de Deadlock

Condition Deadlock

1. Exclusion mutuelle
   - Une ressource ne peut attribuée à un seul process
2. Hold and wait
   - Le processus peut demander d'autres ressources
3. Pas de préemption
   - On ne peut pas reprendre la ressource
4. Attente circulaire
   - Chaque processus attend une ressource détenue par un autre

Prévention des Deadlocks

- Il existe différentes solutions
- La plus simple:
  - **hiérarchiser** les ressources
  - Tout le monde **lock** les ressources dans leur **ordre hiérarchique**
- Example Java

Concepts abordés

- Processus
- Descripteurs de processus
- Etats des processus
- Signal
- Ordonnancement
  - Round robin, priorité, famine
  - À trois niveaux

Concepts abordés

- Threads
- Programmation multi-threads
- Section critique
- Deadlocks