

IMT Mines Alès
École Mines-Télécom

Informatique concurrente et répartie

Les processus et les threads

Emmanuel Romagnoli

- ♦ Les processus lourds
- ♦ Les processus légers
- ♦ Synthèse

La notion de processus

Un programme s'exécute au sein d'une coquille logicielle appelée « **processus** ».

Le système d'exploitation a plusieurs tâches dont celle de **gérer les processus** qui sont liés entre eux par des liens hiérarchiques de type « père – fils ».

On obtient donc un arbre des processus qui peut être plus ou moins complexe selon le système d'exploitation.

Le cas des SE monotâches

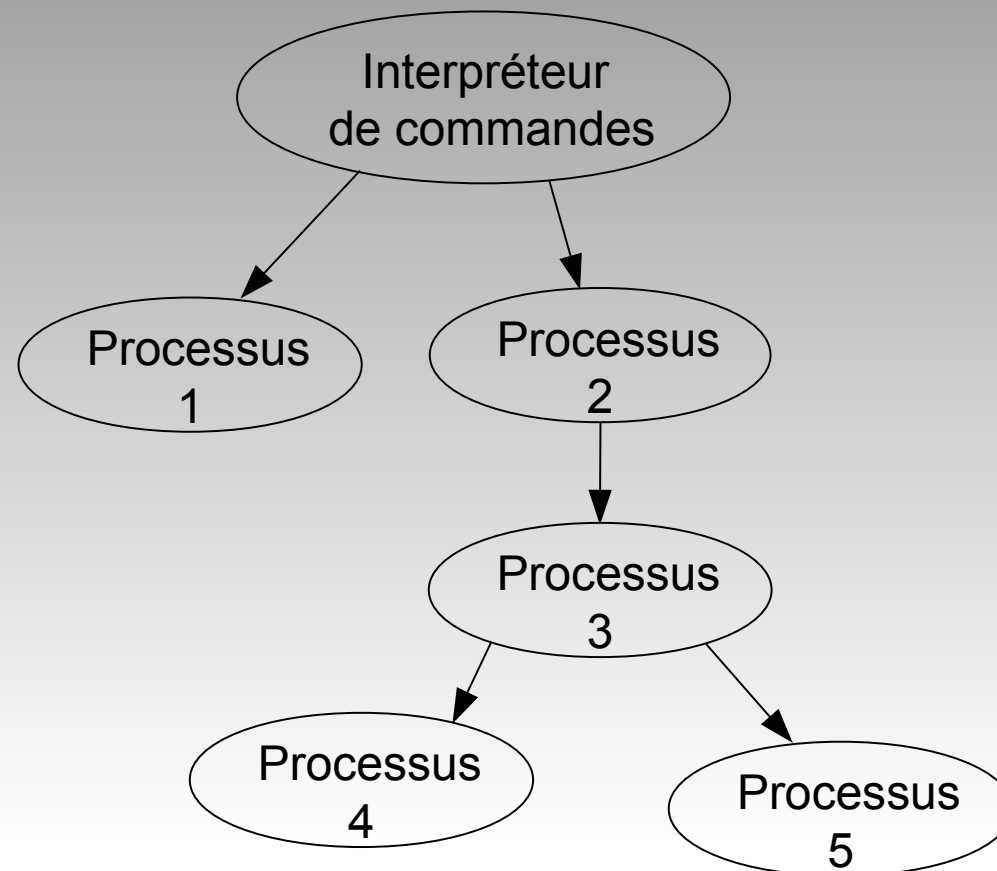
Dans le cas d'un système d'exploitation **monotâche** comme le **MS-DOS**, l'arbre est réduit à une branche.

L'interpréteur de commandes (command.com) correspond au processus père qui crée un processus fils pour lancer l'exécution d'un programme.

Le processus père est alors bloqué jusqu'à la complétion du processus fils car le MS-DOS n'est pas un système d'exploitation **préemptif** (il ne peut pas interrompre le fonctionnement d'un processus pour le relancer ultérieurement).

Le cas des SE multitâches

Dans le cas d'un système d'exploitation **multitâches** comme **Linux**, l'arbre comporte plusieurs branches qui ne sont pas nécessairement de longueurs égales.



Le cycle de vie des processus

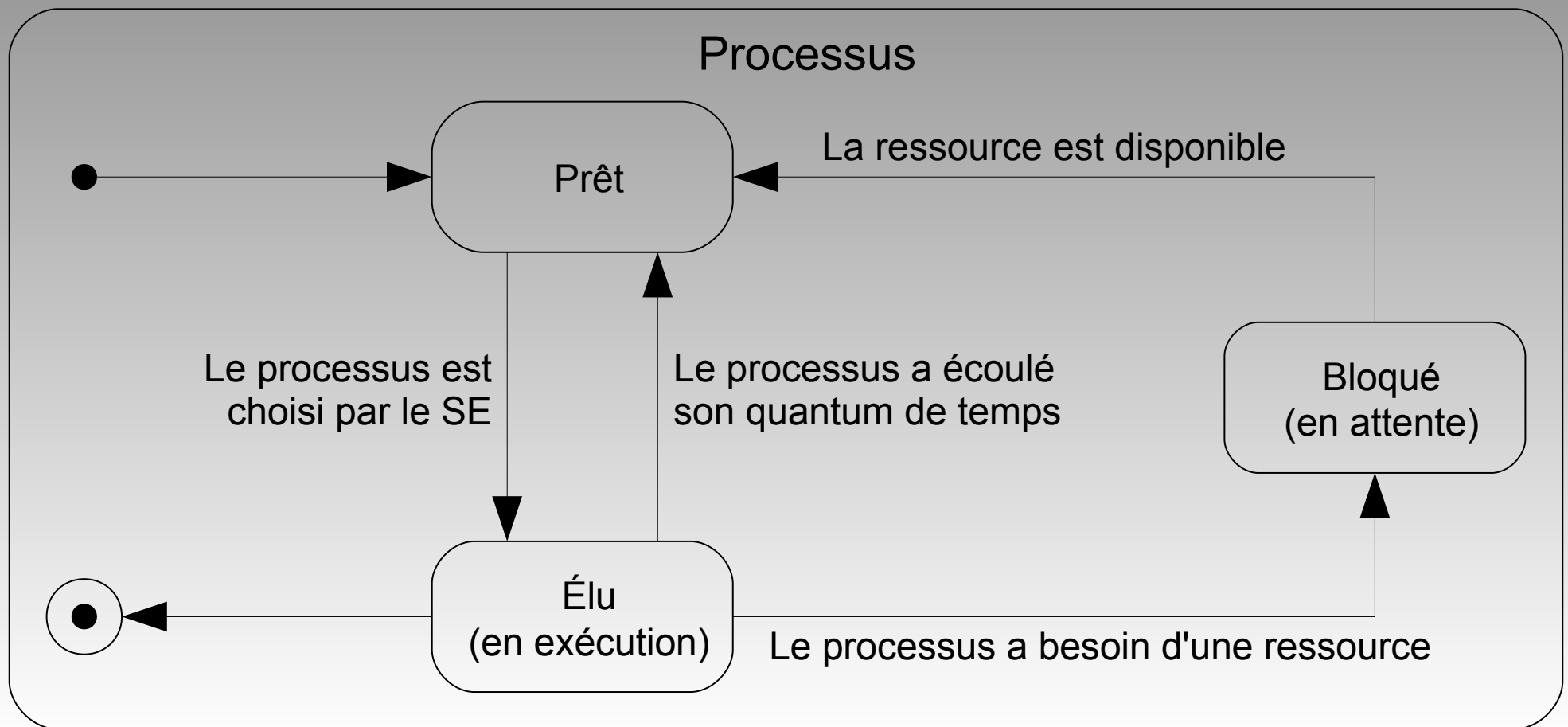
Si on considère un ordinateur muni d'un seul processus mono-coeur, ces processus s'exécutent de façon **pseudo-simultanée**.

Chaque processus dispose, **à tour de rôle**, du processeur pendant un **quantum de temps** afin d'exécuter un morceau de leur programme.

Lorsque le processus est bloqué en attente d'une ressource ou lorsqu'il a écoulé son quantum de temps, celui-ci est alors **interrompu** par le système d'exploitation.

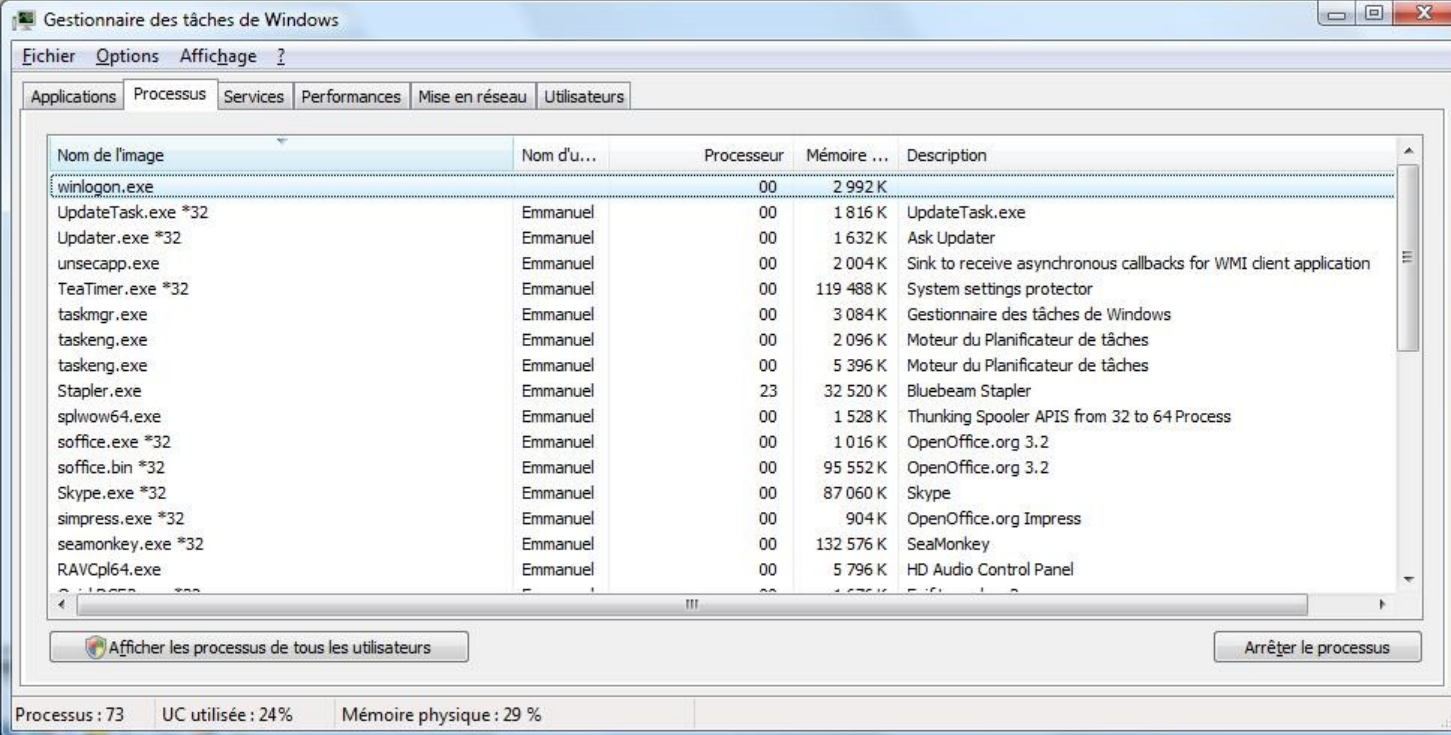
Le cycle de vie des processus

Le processus peut être considéré comme une machine à états.



La table des processus

Pour gérer l'état des différents processus, le système d'exploitation doit maintenir une **table des processus** comportant autant de lignes qu'il y a de processus en mémoire.



The screenshot shows the 'Gestionnaire des tâches de Windows' (Windows Task Manager) window with the 'Processus' (Processes) tab selected. The window displays a list of running processes with columns for 'Nom de l'image' (Image name), 'Nom d'utilisateur' (User name), 'Processeur' (Processor), 'Mémoire' (Memory), and 'Description' (Description). The processes listed include winlogon.exe, UpdateTask.exe, Updater.exe, unsecapp.exe, TeaTimer.exe, taskmgr.exe, taskeng.exe, Stapler.exe, splwow64.exe, soffice.exe, soffice.bin, Skype.exe, simpres.exe, seamonkey.exe, and RAVCpl64.exe. The status bar at the bottom indicates 'Processus : 73', 'UC utilisée : 24%', and 'Mémoire physique : 29%'.

Nom de l'image	Nom d'utilisateur	Processeur	Mémoire	Description
winlogon.exe		00	2 992 K	
UpdateTask.exe *32	Emmanuel	00	1 816 K	UpdateTask.exe
Updater.exe *32	Emmanuel	00	1 632 K	Ask Updater
unsecapp.exe	Emmanuel	00	2 004 K	Sink to receive asynchronous callbacks for WMI client application
TeaTimer.exe *32	Emmanuel	00	119 488 K	System settings protector
taskmgr.exe	Emmanuel	00	3 084 K	Gestionnaire des tâches de Windows
taskeng.exe	Emmanuel	00	2 096 K	Moteur du Planificateur de tâches
taskeng.exe	Emmanuel	00	5 396 K	Moteur du Planificateur de tâches
Stapler.exe	Emmanuel	23	32 520 K	Bluebeam Stapler
splwow64.exe	Emmanuel	00	1 528 K	Thinking Spooler APIS from 32 to 64 Process
soffice.exe *32	Emmanuel	00	1 016 K	OpenOffice.org 3.2
soffice.bin *32	Emmanuel	00	95 552 K	OpenOffice.org 3.2
Skype.exe *32	Emmanuel	00	87 060 K	Skype
simpres.exe *32	Emmanuel	00	904 K	OpenOffice.org Impress
seamonkey.exe *32	Emmanuel	00	132 576 K	SeaMonkey
RAVCpl64.exe	Emmanuel	00	5 796 K	HD Audio Control Panel

La table des processus

Chaque ligne comporte des informations concernant l'état du processus, la mémoire qu'il utilise, les fichiers... Dans son ouvrage « Systèmes d'exploitation », Tanenbaum en a répertorié quelques-unes comme :

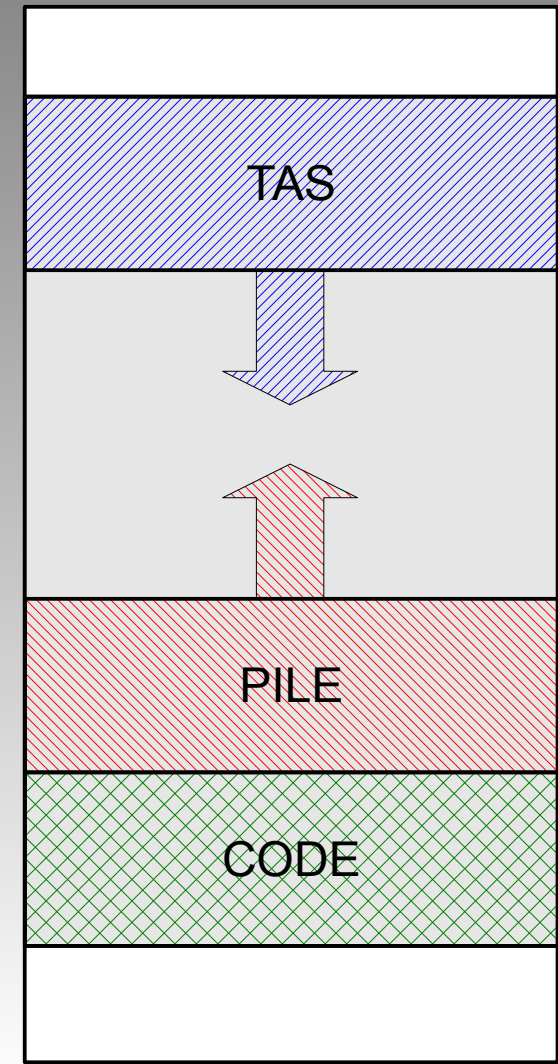
- ♦ la copie des registres du processeur ;
- ♦ l'état du processus ;
- ♦ le registre de base ;
- ♦ le registre limite ;
- ♦ l'identifiant du processus (PID) ;
- ♦ le répertoire de travail ;
- ♦ les descripteurs de fichiers ...

Voir <http://fr.nicebooks.com/ISBN/9782744072994>

La zone mémoire utilisée par le processus

Un processus se voit attribuer une zone mémoire qui est divisée en 3 parties :

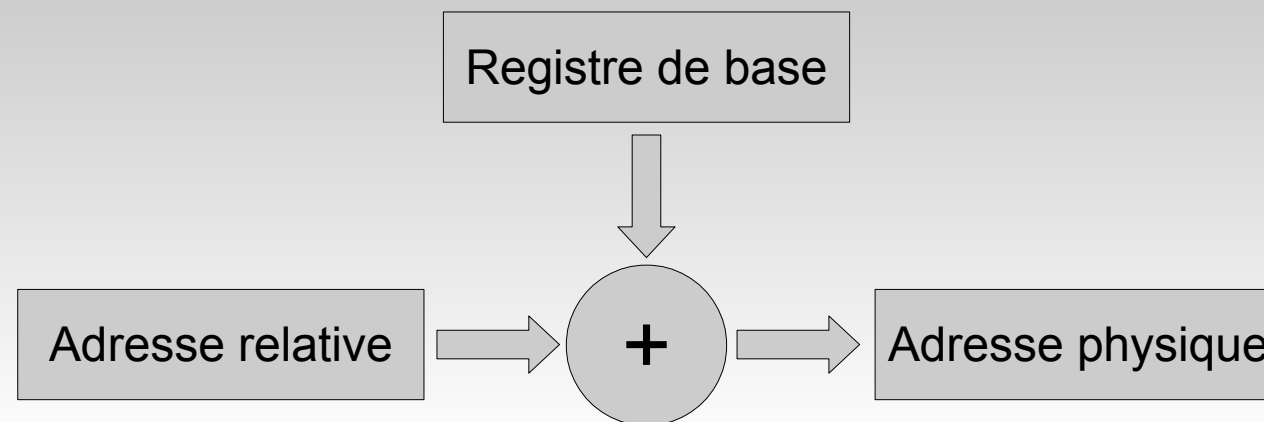
- ♦ le tas où est allouée la mémoire, grâce à malloc ;
- ♦ la pile sont stockés les paramètres des fonctions, l'adresse de retour et les variables locales ;
- ♦ le code où sont placés les opcodes (codes machines) du programme à exécuter.



La translation d'adresses

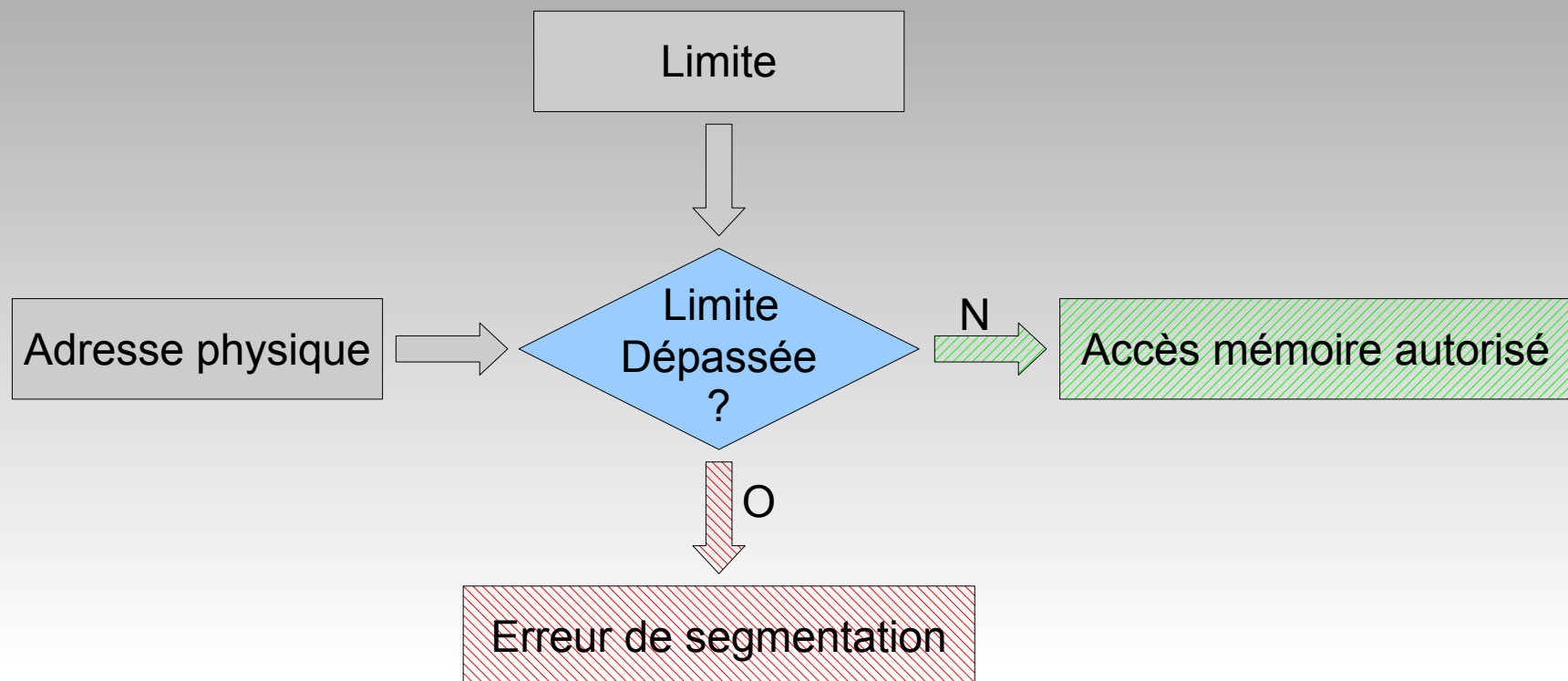
Les adresses contenues dans le code (JMP, JNZ, LOOP...) sont des adresses relatives à cette zone mémoire.

Lorsque le processeur doit accéder à une case mémoire, il doit calculer l'adresse physique en additionnant l'adresse relative (offset ou déplacement) et le registre de base : on parle alors de translation d'adresses.



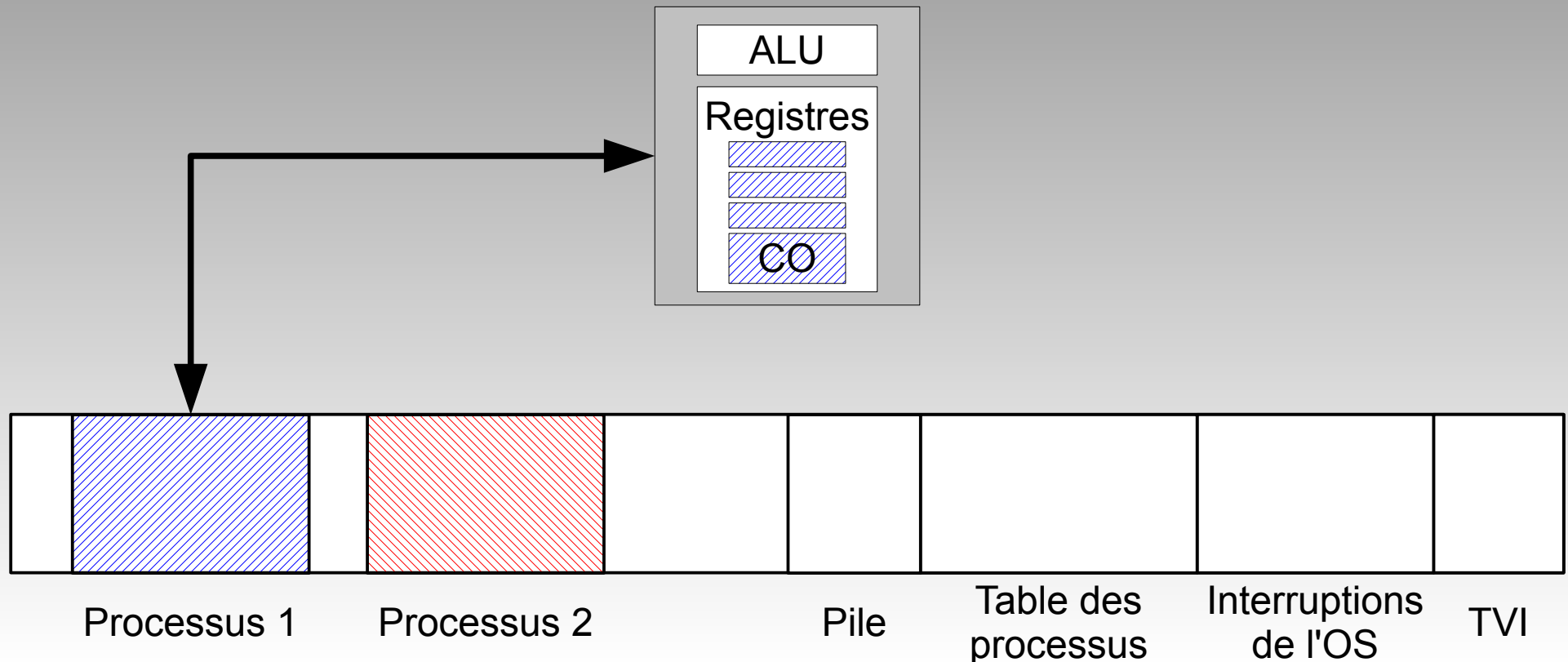
Le mécanisme de protection

On vérifie que cette adresse physique ne se situe pas en dehors de la zone autorisée. Si c'est le cas, on obtient une **erreur de segmentation**.



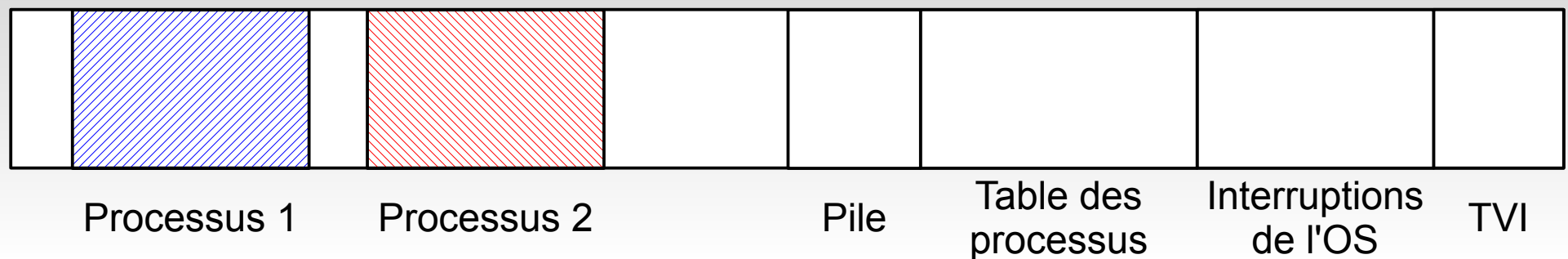
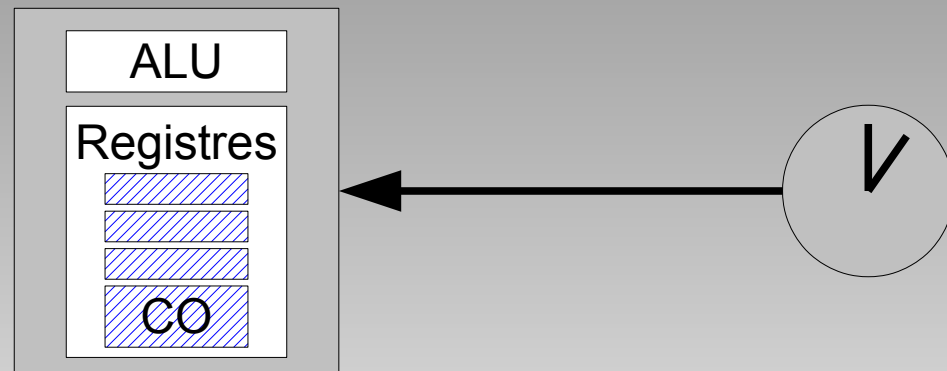
Le changement de contexte

On imagine que 2 processus (P1 et P2) ont été lancés et que P1 est élu. Le compteur ordinal (CO) du processeur désigne des opcodes du segment de code de P1.



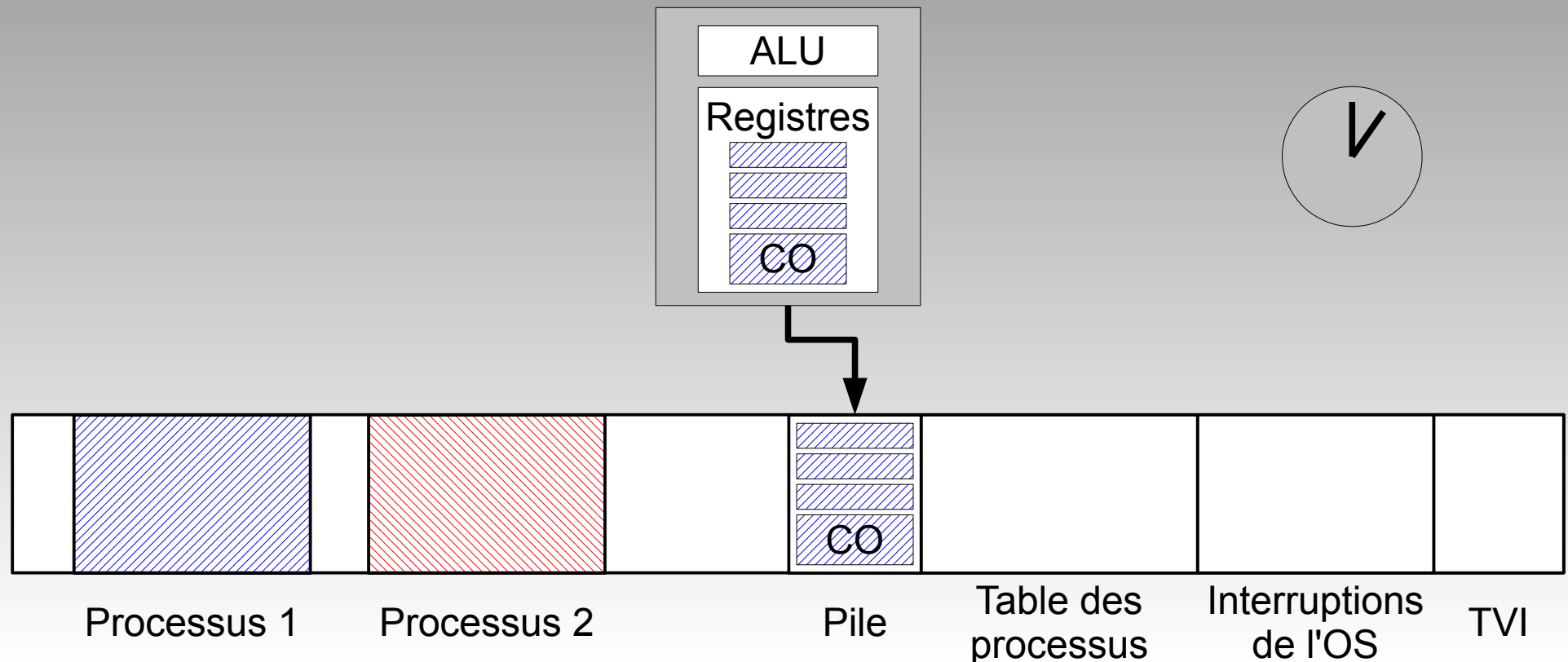
Le changement de contexte

Périodiquement, le processeur reçoit un signal de demande d'interruption (Interrupt request ou IRQ) de l'horloge. Il interrompt le traitement en cours.



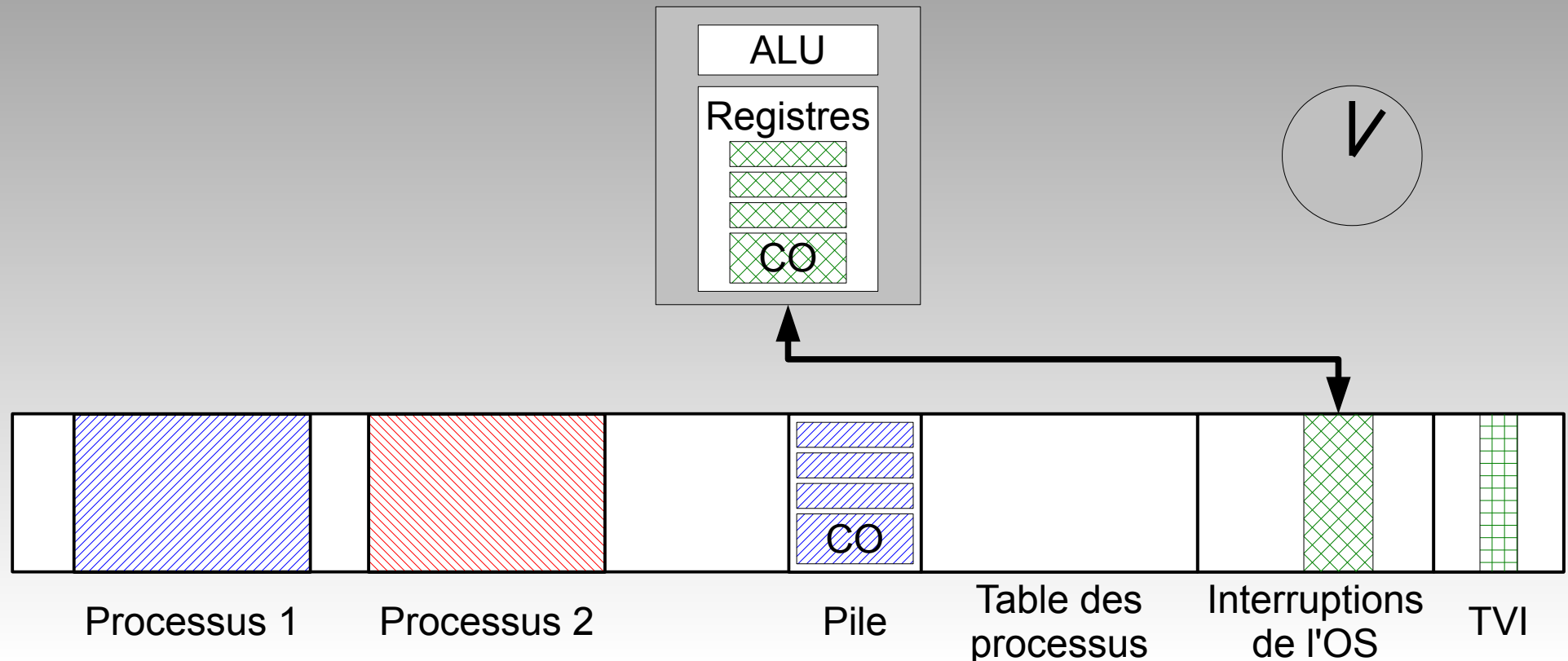
Le changement de contexte

Le processeur sauvegarde le contenu de ses registres dans la pile gérée par le système d'exploitation.



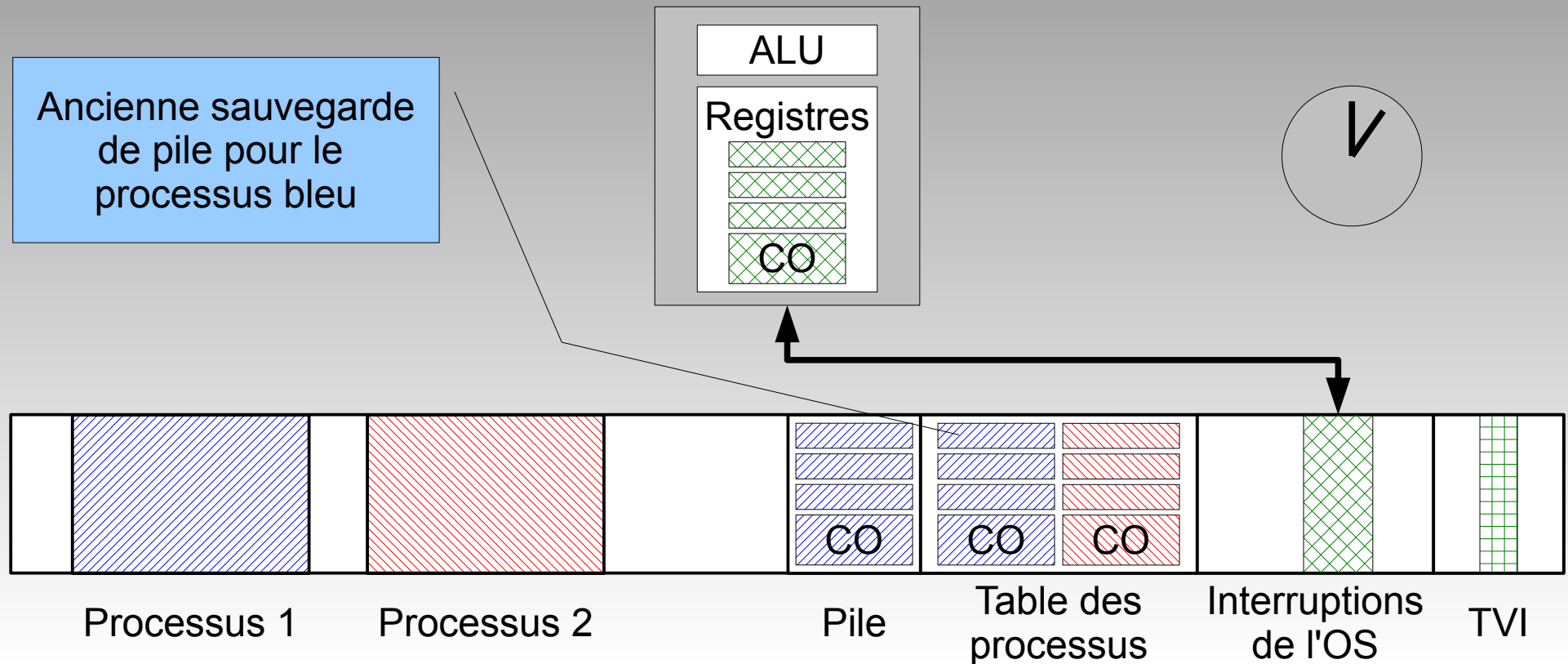
Le changement de contexte

Le code exécuté est celui de l'ordonnanceur du système d'exploitation qui va désigner (par exemple) P2 comme nouveau processus élu.



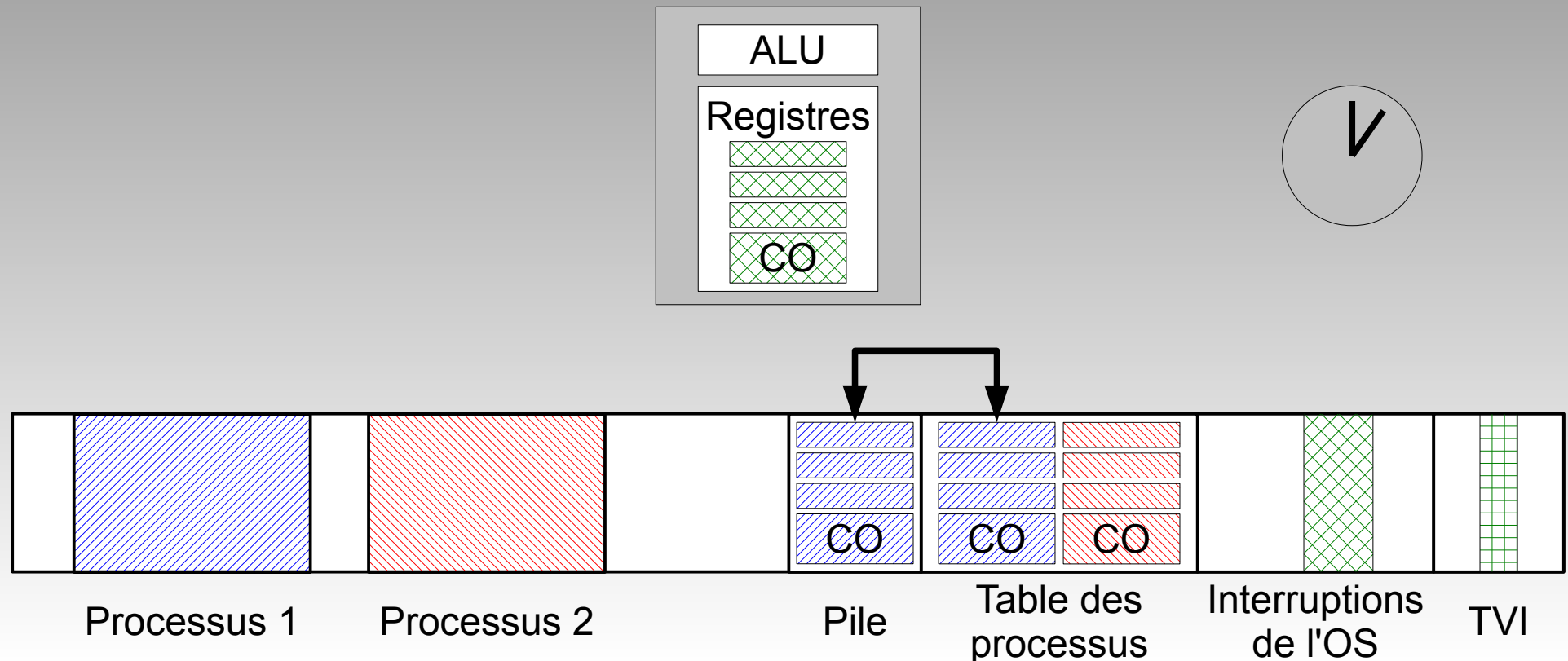
Le changement de contexte

Ce code copie le contenu de la pile dans la ligne associée à P1, dans la table des processus.



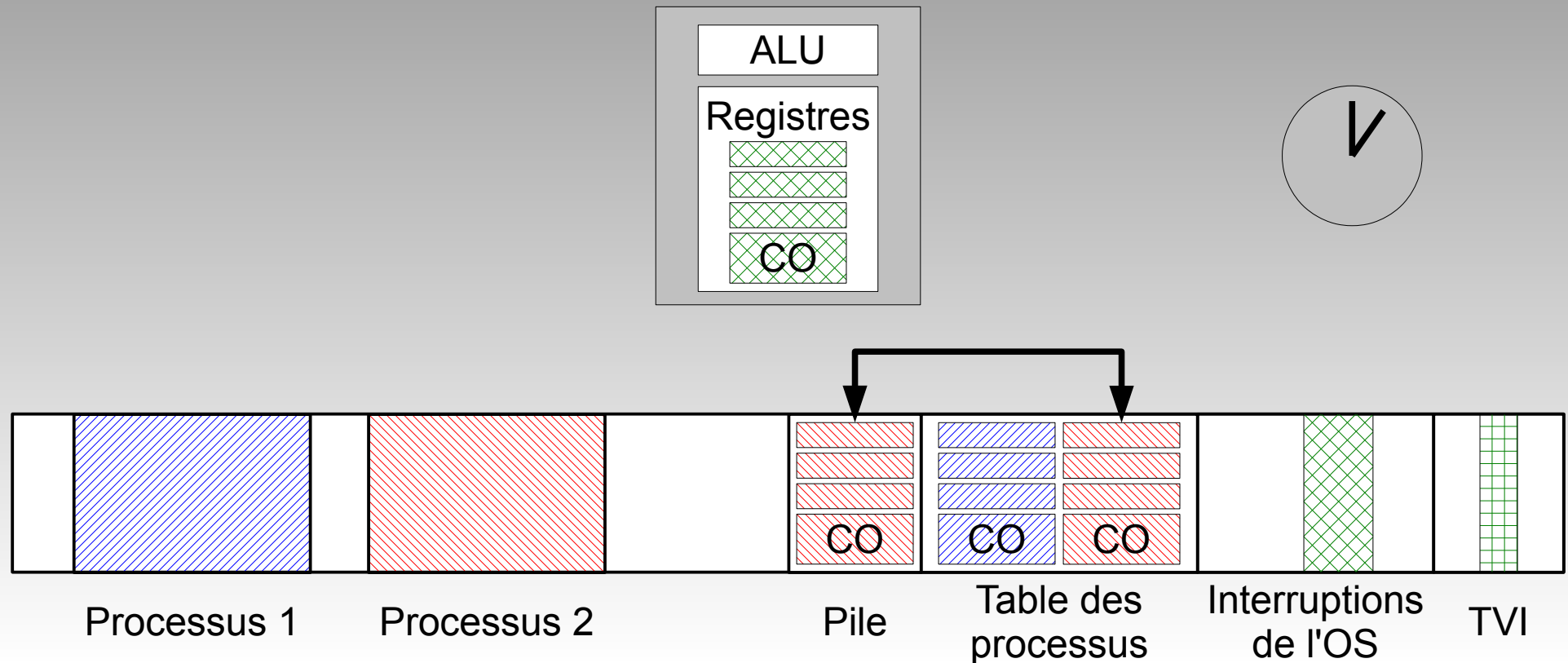
Le changement de contexte

Ce code copie le contenu de la pile dans la ligne associée à P1, dans la table des processus.



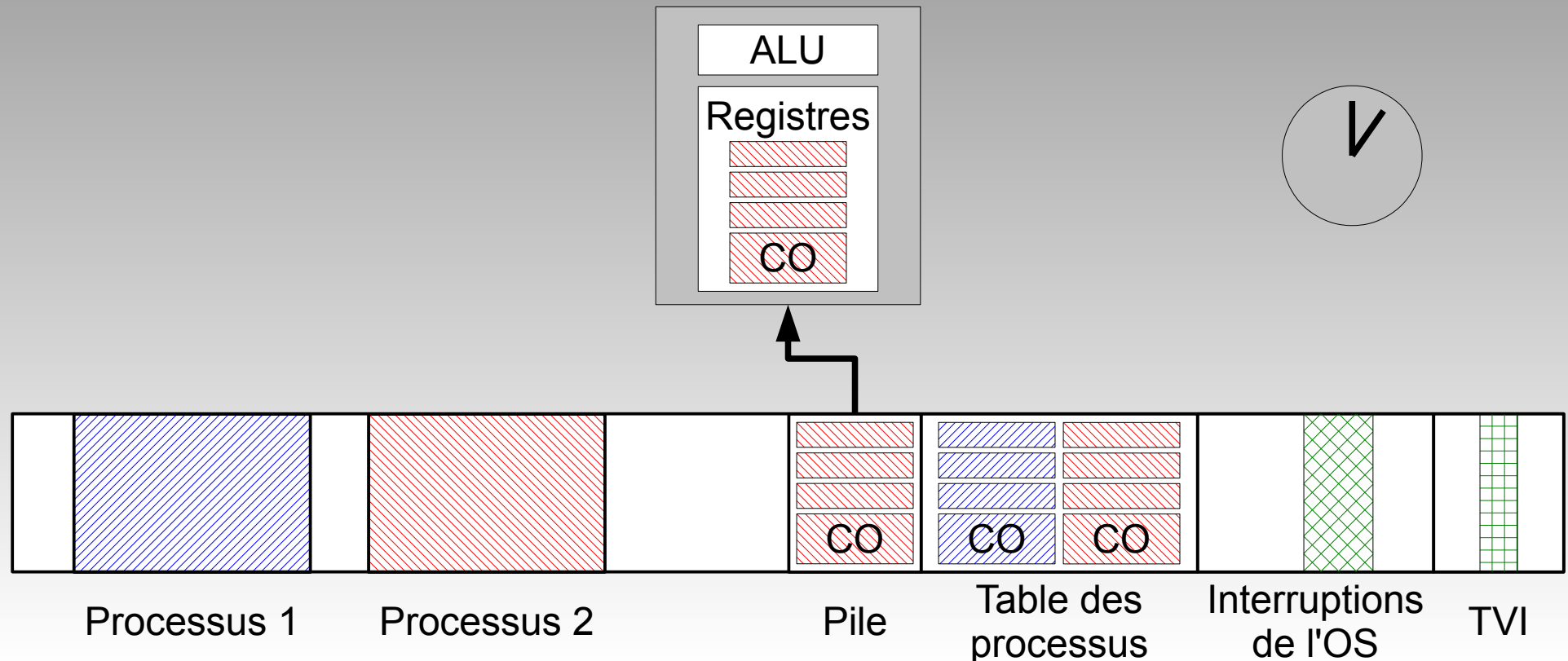
Le changement de contexte

Puis le contenu de la ligne associée à P2, dans la table des processus, écrase celui de la pile.



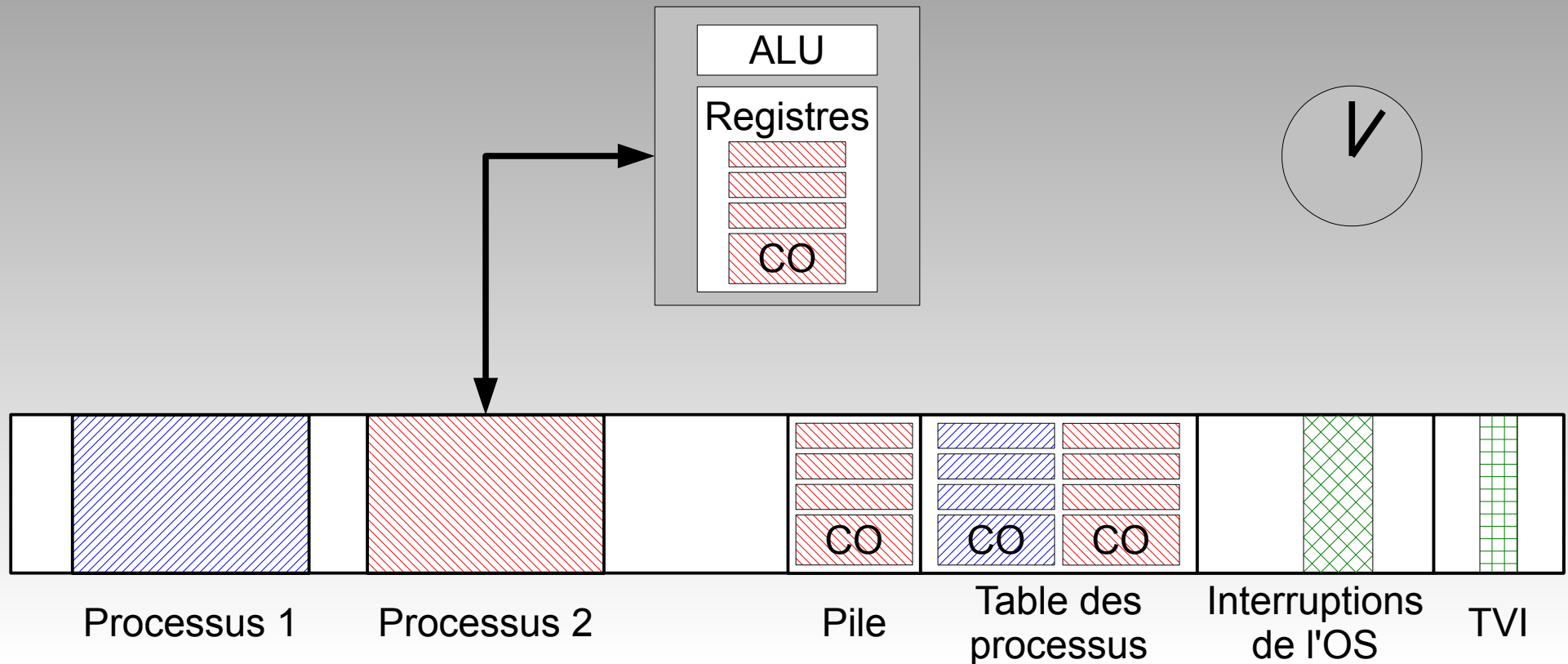
Le changement de contexte

Lorsque le traitement de l'interruption est terminé, on charge le contenu de la pile dans les registres du processeur.



Le changement de contexte

Le processeur reprend son fonctionnement normal, mais en exécutant le processus P2.



La création d'un processus lourd

La création d'un processus dit « lourd » s'effectue par la commande **fork** qu'on étudiera un peu plus loin.

Elle est utilisée pour :

- ♦ créer un **clone** du processus père dans le cadre d'une programmation concurrente ;
- ♦ créer un **clone** d'un shell (le processus père) dont le code sera remplacé par celui du programme qui lancé (grâce à la commande **exec**).

L'inconvénient des processus lourds

Pour créer ce clone, on doit donc dupliquer la zone de mémoire utilisée par le processus père et la ligne qui contient les informations concernant le processus père dans la table des processus.

Dans le cas de la programmation concurrente, ce procédé **prend du temps** et **gaspille de la mémoire** (duplication du code).

En outre, les deux processus utilisant des zones mémoires distinctes, il faut utiliser des procédés « lents » pour la **communication interprocessus** (perte de temps) nécessitant le **recours au système d'exploitation** afin de contourner le mécanisme de protection

Une origine lointaine

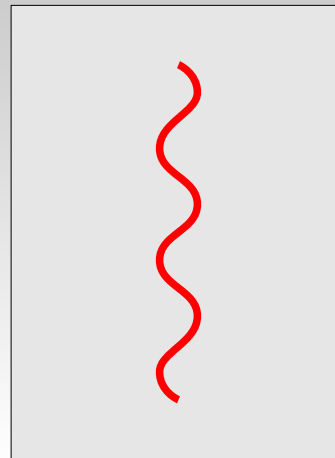
Le mécanisme des processus légers existerait depuis 1965, mais il n'avait pas été utilisé lors de la conception du système Multics car il n'y a pas de protection entre les threads.

Unix, qui dérive de Multics, a donc hérité du système de processus lourds

Voir <http://www.cis.temple.edu/~ingargio/cis307/reference/faq1.txt>
<http://www.faqs.org/faqs/os-research/part1/section-10.html>
<http://perso.numericable.fr/pdeiberale69/patrick.deiber/pub/processus-leger.pdf>

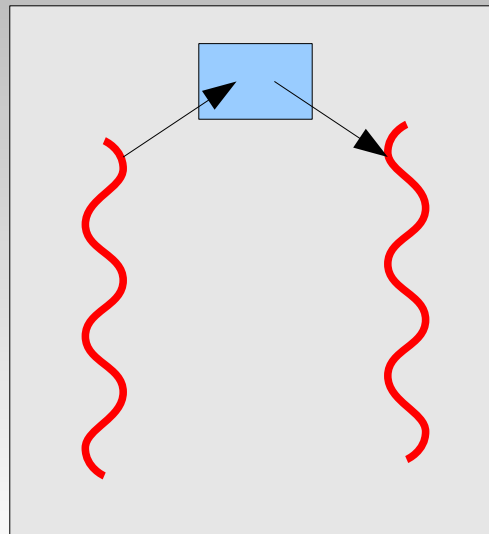
Un processus léger (un thread) est un **flot d'exécution** (d'une suite d'instructions) qui s'exécute **au sein d'un processus lourd**.

Un processus lourd contient **au moins un flot** d'exécution qui est lancé par la fonction principale (le main) du programme.



Une communication facilitée

Il est possible de créer d'autres flots qui **se partagent la zone de mémoire** allouée au processus lourd. Ces flots peuvent alors communiquer entre eux en utilisant des variables stockées dans le tas.

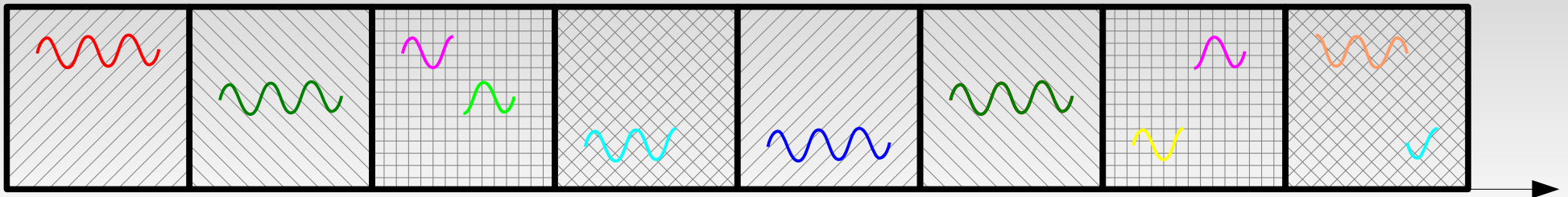


Les threads dans la pratique

Concrètement un thread est constitué par :

- ♦ un morceau du « code » (les instructions à exécuter) ;
- ♦ des informations le concernant dans une « table des threads » située dans le tas.

Ces threads sont exécutés les uns à la suite des autres durant le quantum de temps du processus lourds.



Deux niveaux d'ordonnancement

Un thread spécial, ajouté lors de la compilation à partir d'une bibliothèque est chargé de lancer l'exécution d'un (ou plusieurs threads) lorsque le processus lourd englobant est élu.

On a donc deux niveaux d'ordonnancement :

- ♦ un ordonnancement entre les processus lourds au sein de l'ordinateur ;
- ♦ un ordonnancement entre les processus légers au sein de chaque processus lourd.

	Processus lourd	Processus léger
Temps de création	Lent	Rapide
Mode de création	Duplication de zones mémoire	Création d'un flot d'exécution au sein de la même zone mémoire
Communication	Eléments extérieurs contrôlé par l'OS	Variable dans le tas
Protection	Oui	Non