Systèmes d'exploitations II

Présentation #2 : Concepts fondamentaux des SE

24/10/2021

Ahmed Benmoussa

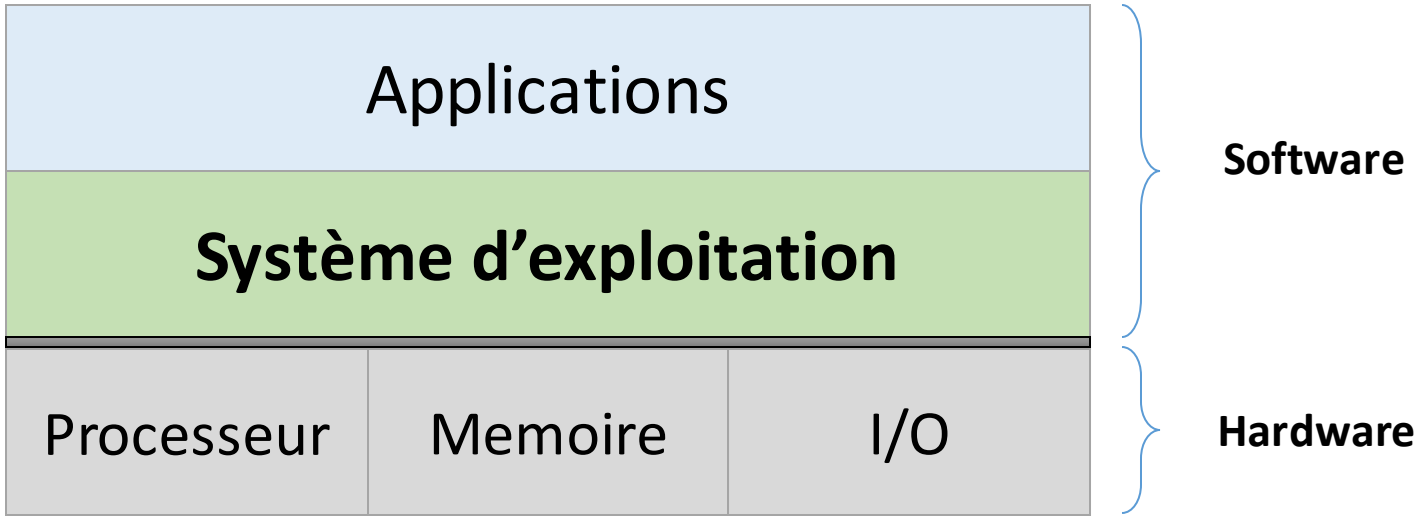
Centre universitaire d'Aflou

Objectif de cette presentation

- Rappel sur la presentation précédente.
- Concepts fondamentaux des SE.

Rappel: Définition d'un système d'exploitation

- Une couche applicative qui joue le role d'intermediaire entre le materiel et les application.



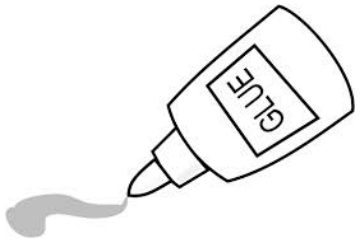
Rappel: Roles d’un système d’exploitation

1. Virtualiser les ressources: faire croire aux precessus qu’ils detiennent toute les resources de la machine.
2. Partage les ressources entre les differents processus.
3. Protection des resources.
4. Assurer la communication entre les differents processus qui communiquent entre eux (Réseau, fichiers).
5. Fournir aux developpeurs un environnement plus agreeable et facile pour developper des applications.
6. Gestion en arrière plan des ressources

Rappel: Roles d'un système d'exploitation

- **Referee (arbitre)**
  - Gere le partage, la protection et l'isolation des resources

- **Illusionist (illusioniste)**
  - Abstraction du materiel facile a utiliser
    - » Memoire infinie, machine dédiée
    - » Objets haut niveau: fichiers, utilisateurs, …
    - » Virtualisation, …



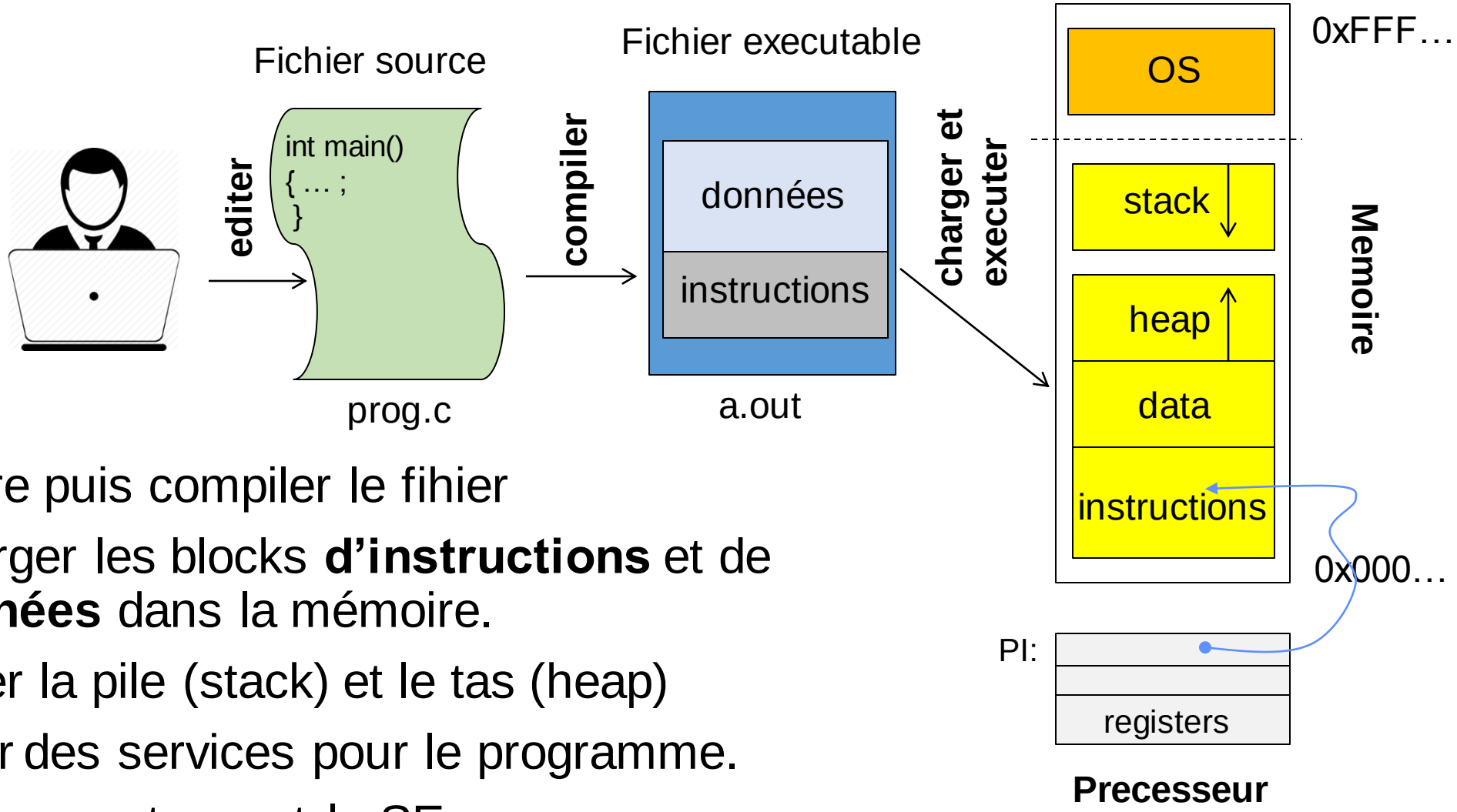
- **Glue (colle)**
  - Services communs
    - » Espace de stockage, Window system, Networking
    - » Partage, Autorisation
    - » Look and feel

Concepts fondamentaux des SE

OS: 04 Concepts fondamentaux

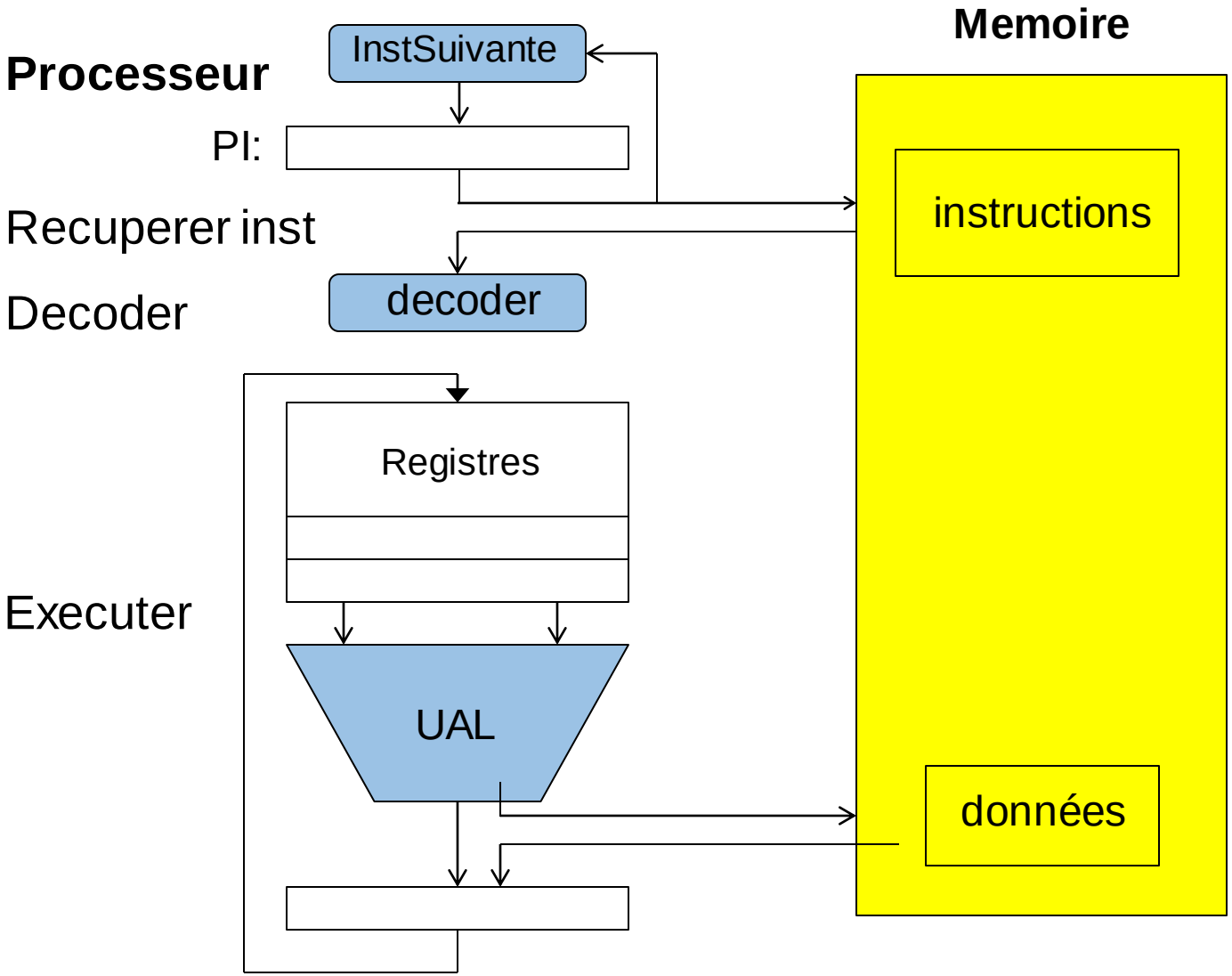
1. **Thread**
   - Décrit un etat du programme.
   - Pointeur d'instruction (*Program Counter*), Registres, Pile, …
2. **Espace d'adresse (*address space*)**
   - Une plage d'adresse accessible pour le programme (lecture/ecriture).
3. **Processus: une instance d'un programme en execution**
   - Espace memoire protégé.
   - Un ou plusieurs Threads.
4. **Dual mode operation / Protection**
   - Seul le SE a accès a certaines ressources.
   - Isoller les programmes les uns des autres, et proteger le SE des programmes.

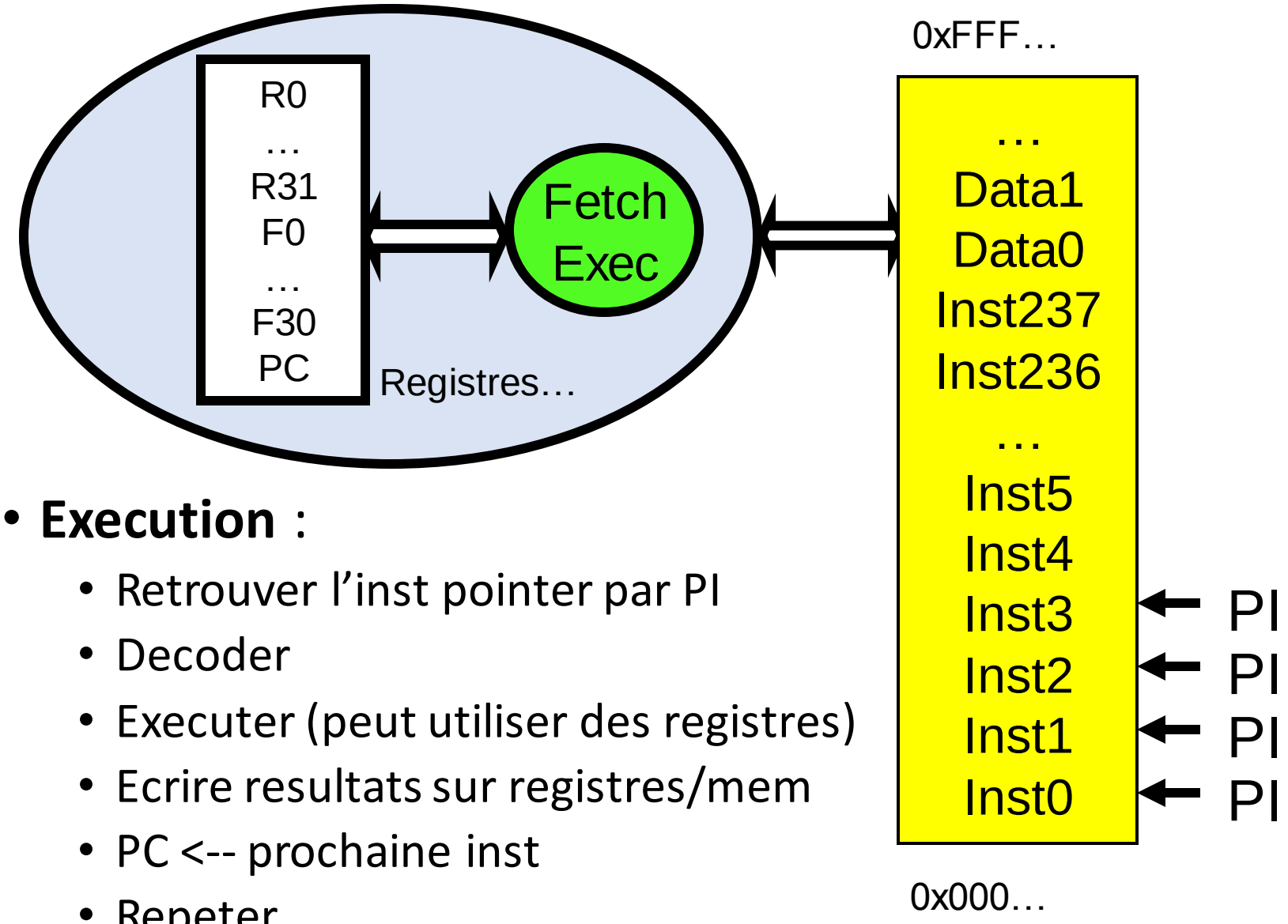
Execution d'un programme



- Ecrire puis compiler le fihier
- Charger les blocks **d'instructions** et de **données** dans la mémoire.
- Créer la pile (stack) et le tas (heap)
- Offrir des services pour le programme.
- Tout en protegeant le SE.

Cycle des instructions





- **Execution** :
  - Retrouver l'inst pointer par PI
  - Decoder
  - Executer (peut utiliser des registres)
  - Ecrire resultats sur registres/mem
  - PC <-- prochaine inst
  - Repeter

Concept 1: Thread

- **Thread: Une tache ou fil d'execution**

  Pointeur d'instruction, Registres, Execution Flags, Pile, Etat de la memoire

- Un thread est en **execution** sur un processeur (Coeur) lorsque il **reside au niveau des registres.**

1. Pointeur d'instruction et l'instruction en cours d'execution.
   - » PC pointe sur la prochaine instruction en memory
   - » Les instructions sont sauvegarder sur la memoire.
2. Les valeurs actuels, comme: integers, pointeurs vers autres valeur en memoire
3. Pointeur de pile. Sauvegarde l'adresse du haut de la pile
4. Le reste est en memoire

Threads: suite

- Un thread est **suspendu** (*suspended*) quand il n'est pas chargé au niveau des registres.
  - Le processeur execute un autres Thread.
  - Pointeur d'instruction ne pointe pas sur une instruction de ce Thread.
  - Une copie des valeurs de chaque register est sauvegarder en memoire.

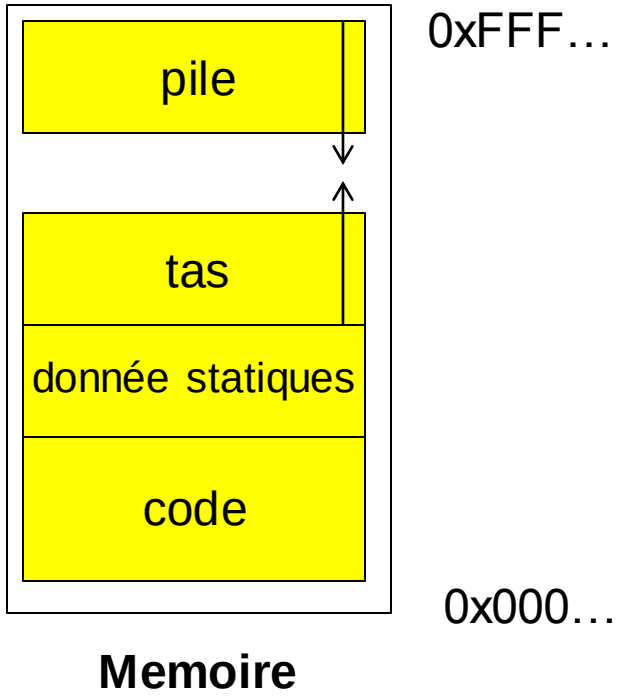
Concept 2: Espace d'adresses

- **Espace d'adresses ⇒ Un ensemble d'adresses accessibles + leurs etats:**
  - Pour un processeur 32-bit, le nombre d'addresses est de
    2^{32} = 4 billion addresses
  - Processeur 64-bit: 2^{64} = **18 quintillion (10^{18}) addresses**

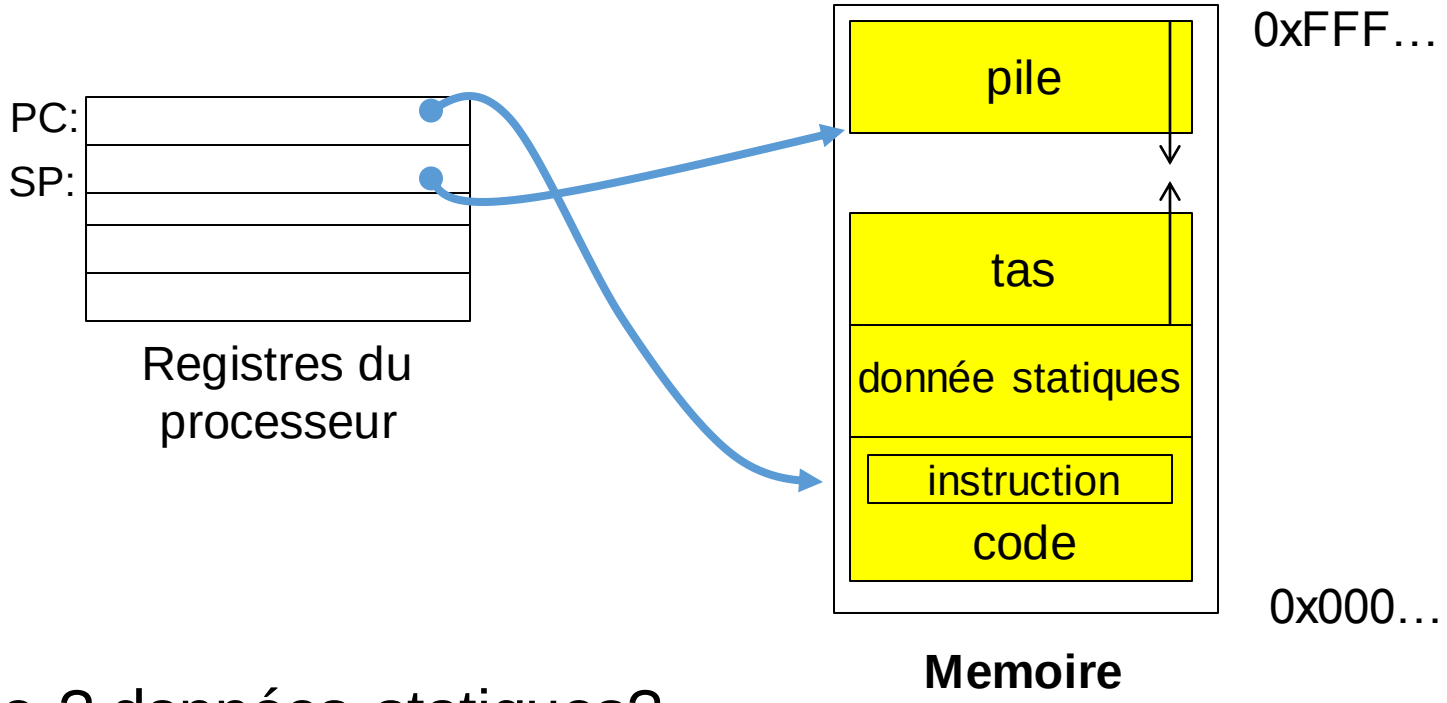
Que peut il se passer quand on lit/ecrit sur une adresse memoire

- Peut agir normalement (memoire ordinaire)
- Peur ignorer l'ecriture memoire
- Peut engender une operation I/O
- Peut causer (exception fault)
- Communiquer avec un autre programme.

….

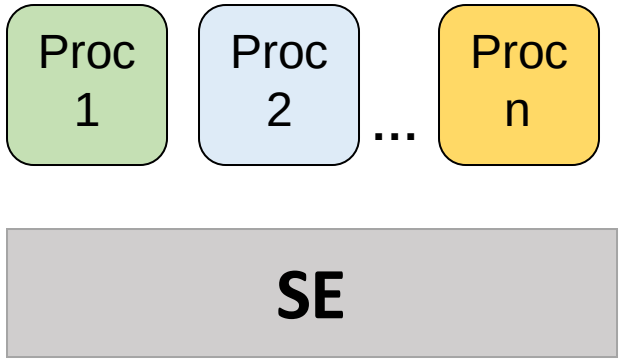


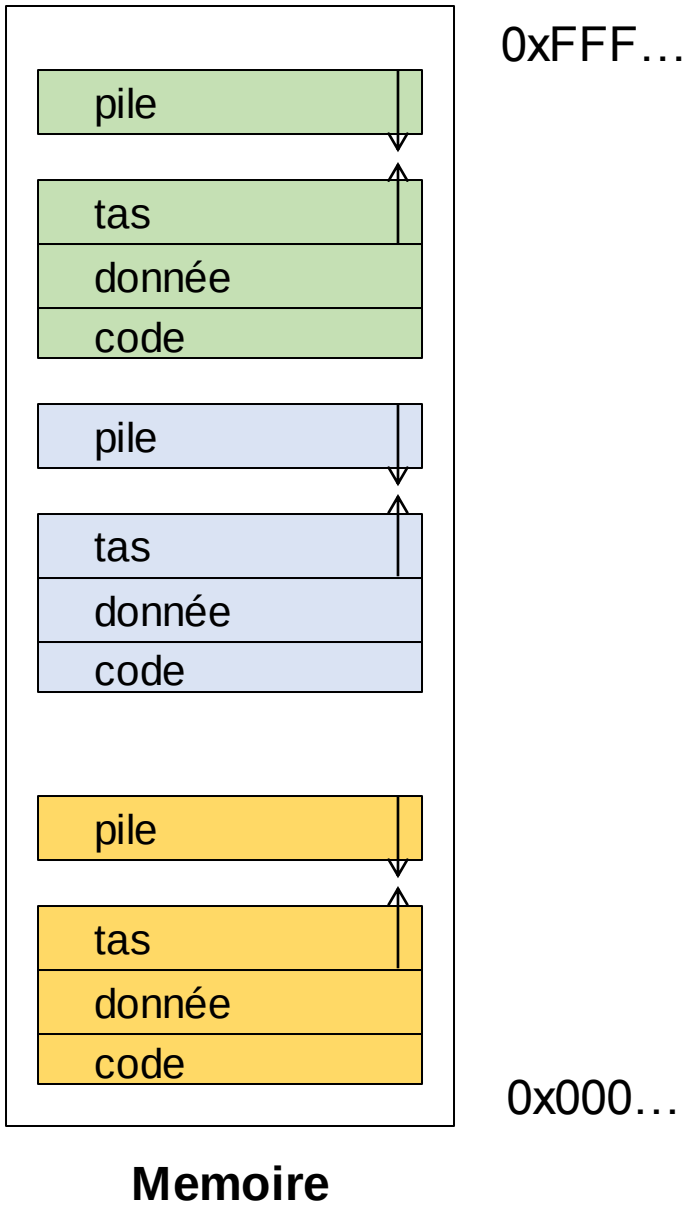
Concept 2: Espace d'adresses

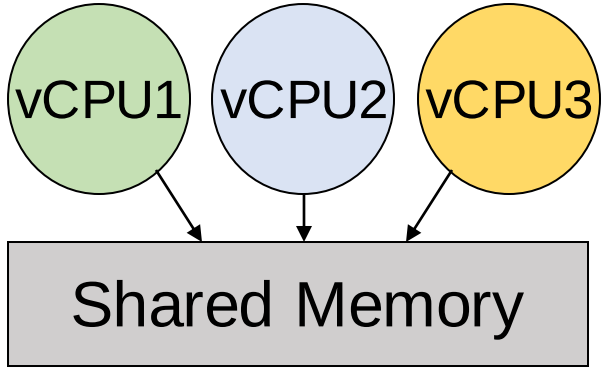


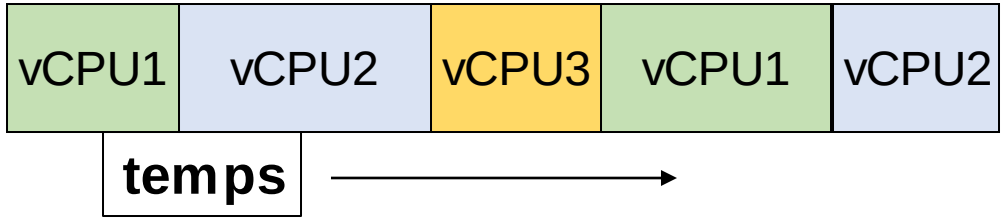
- Segment de code ? données statiques?
- Que contien la Pile et Le Tas?
  - Comment sont il alloués? Leur tailles?

Multiprogramming





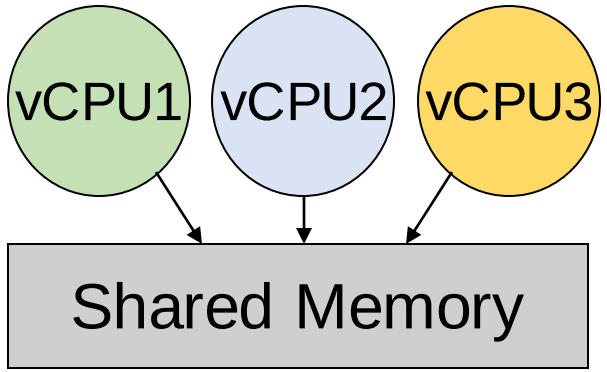


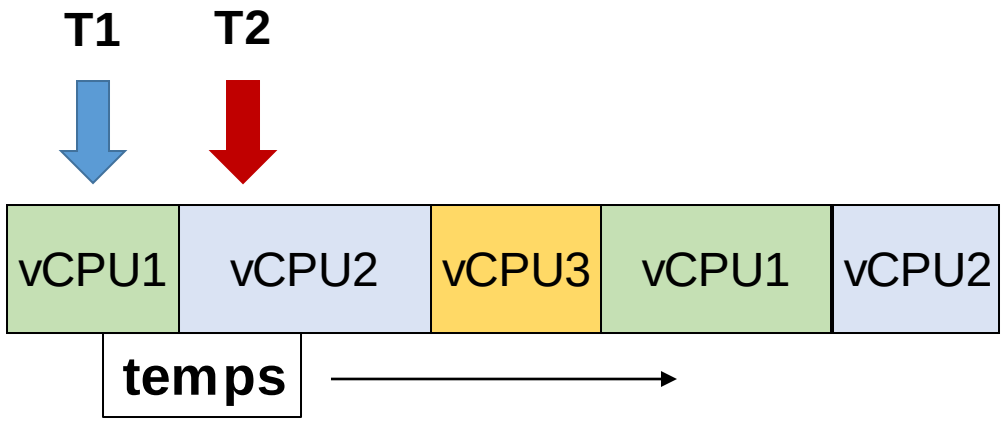


- Suupons un seul processeur, comment peut avoir l'impression d'avoir plusieurs processeurs?

Multiplexage dans le temps

- Chaque CPU virtuel a besoin d'une structure pour
  1. Pointeur d'instruction (PI), pointeur de pile (PP)
  2. Registres (Integer, Floating point, …)





- **Comment switcher d'un vCPU a un autre?**
  1. Sauvegarder PI, PP, et l'etat des registres dans un block d'etat (***Thread Control Block***).
  2. Charger PI, PP et les registres d'un autre TCB.

- **Qu'est ce qui declanche ce changement?**
  – Timer, volontaire, I/O, autres.

Propriétés de ce simple multiplexage

- Tous le vCPU partage les ressources materiels autre que le CPU
  - Memoire, peripheriques E/S

Consequences

- Chaque Thread peut lire/ecrire d'autres espaces memoire
  - Acceder aux données et instrcutions des autres Threads.
  - Corrompre le SE?

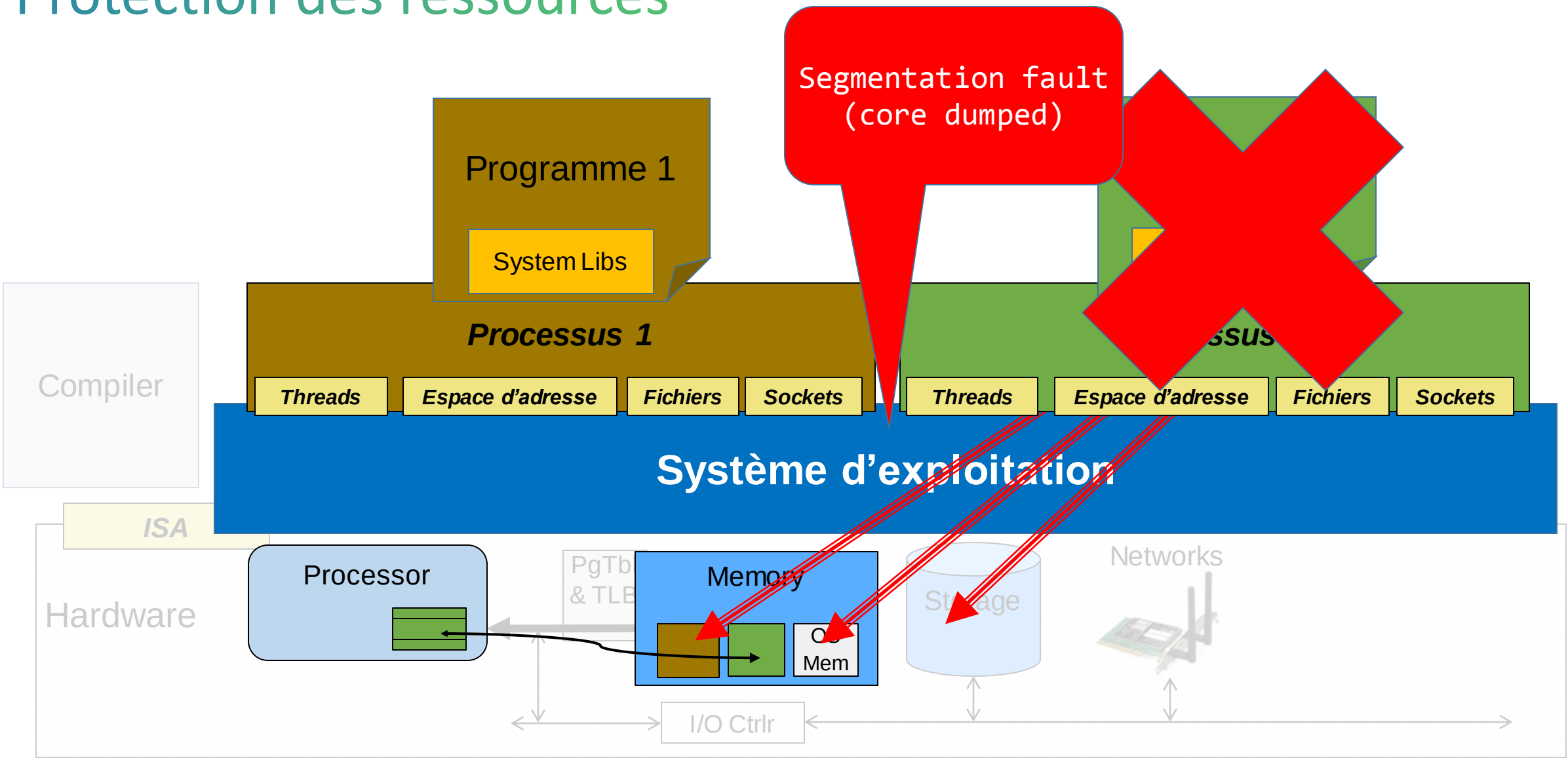
- Ce model (non progtégé) est utilisé:
  - Application embarqués
  - Windows 3.1, Premiere generation des Macintosh (switch se fait volontairement).
  - Windows 95-ME (switch se fait volontairement ou avec un timer).

Simple multiplexage: Pas de protection

- Le système d'exploitation doit se proteger des autres programmes
  - **Fiabilité**: Peut faire planter le SE
  - **Securité**: Limiter le champ d'operation des Threads
  - **Vie privée**: interdire l'accès non autorisé au données des Threads.
  - **Equitabilité**: Limiter le partages des ressources (temps CPU, memoire, E/S, …) entre les Threads.

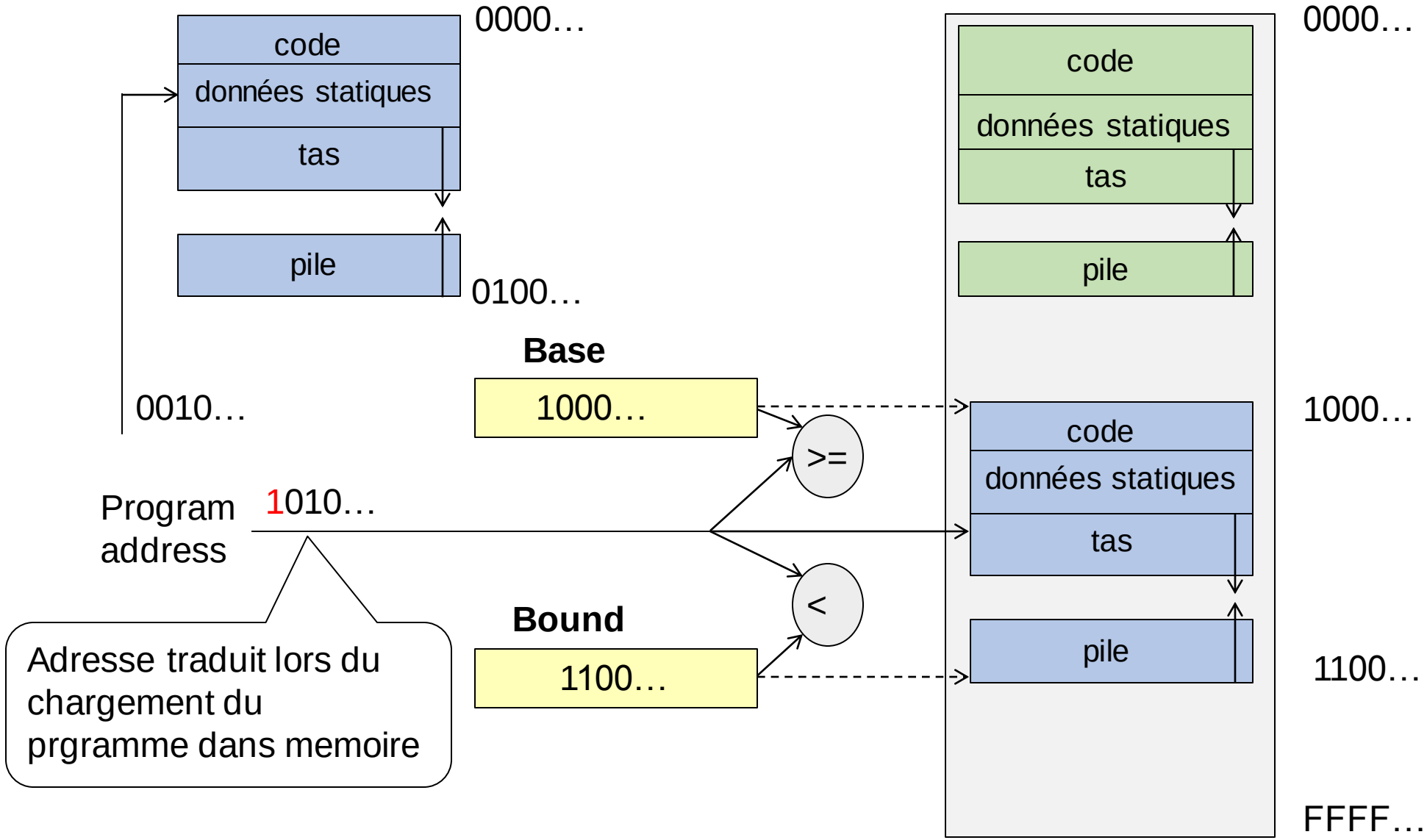
- Le SE doit protéger les utilisateurs des programme les uns des autres.
  - Prevent threads owned by one user from impacting threads owned by another user
  - Prevenir qu'un thread d'un utilisateur n'affecte pas les Threads d'autres utilisteurs.
  - Exemple: prevenir un utilisateur de voler des information d'un autre utlisateur.

Protection des ressources

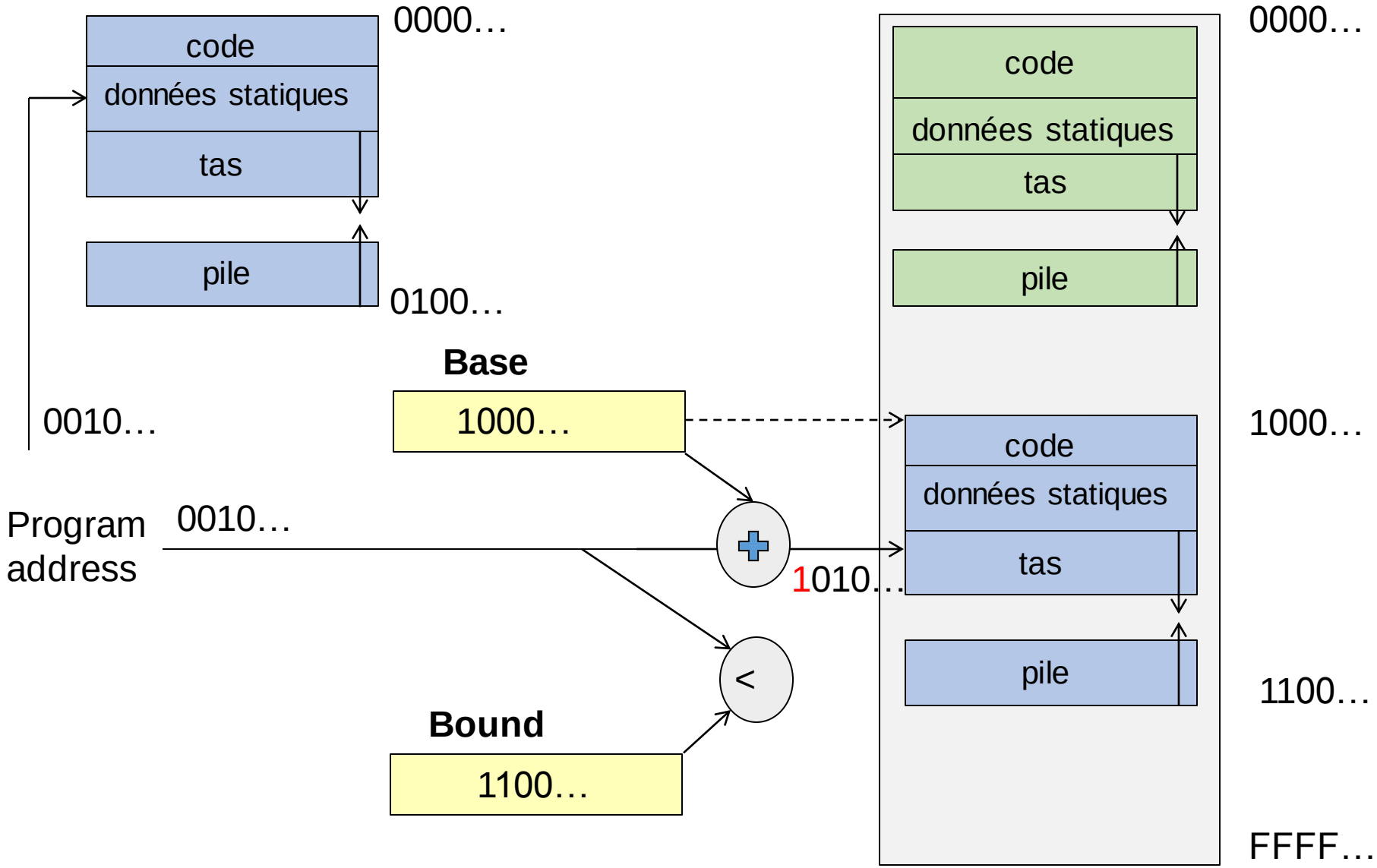


Que peut on faire pour protéger les ressources du système d'exploitation?

Simple solution: Base and Bound (B&B)

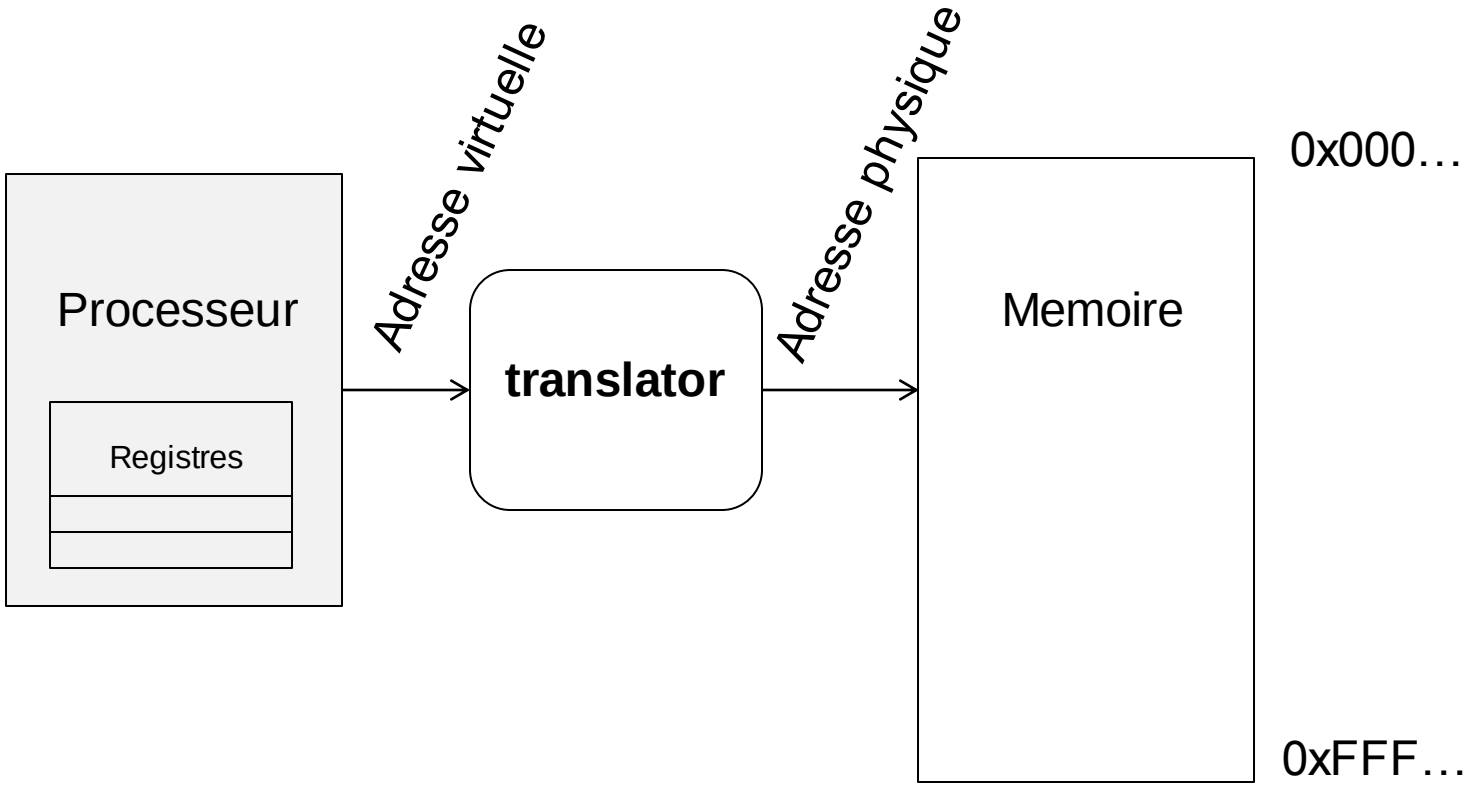


Simple solution: Base and Bound (B&B)

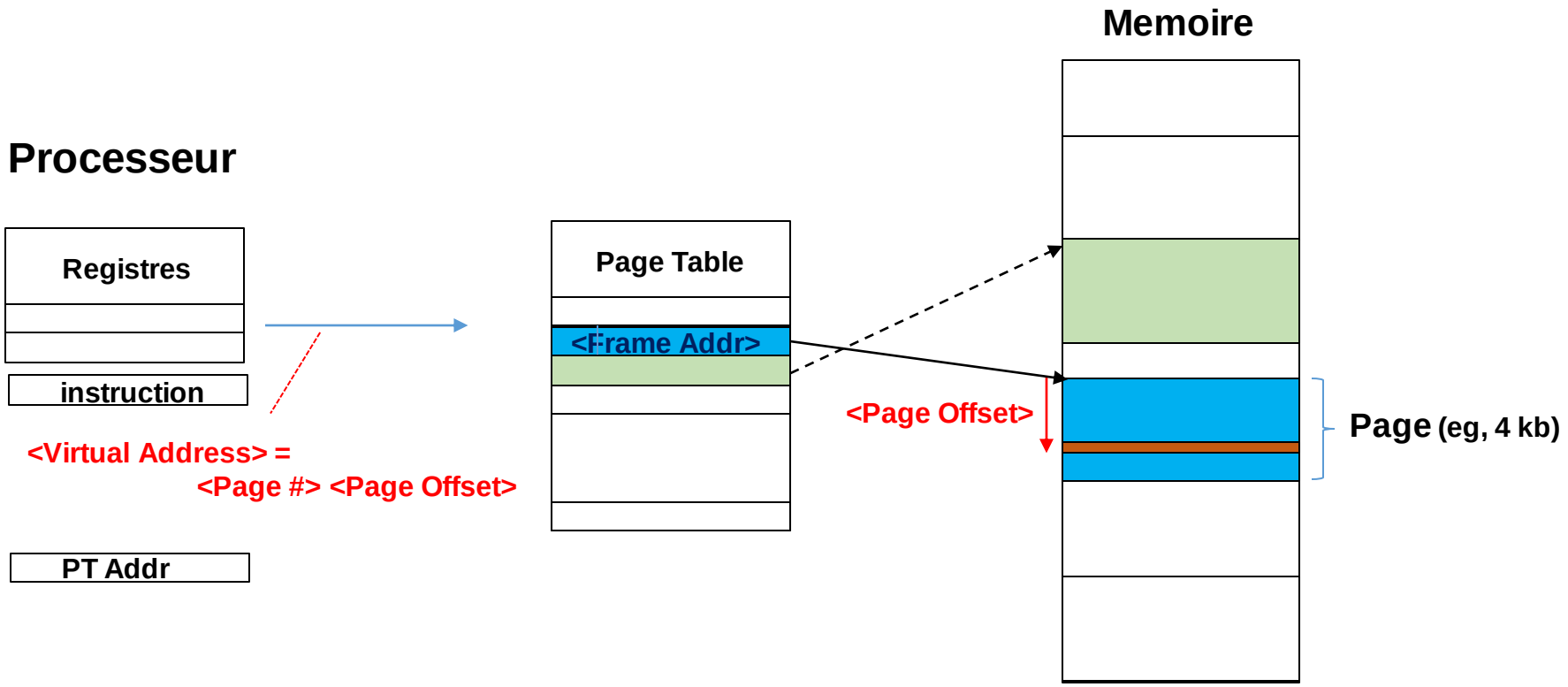


Autre solution: Address Space Translation

- Program operates in an address space that is distinct from the physical memory space of the machine



Autre solution: Address Space Translation



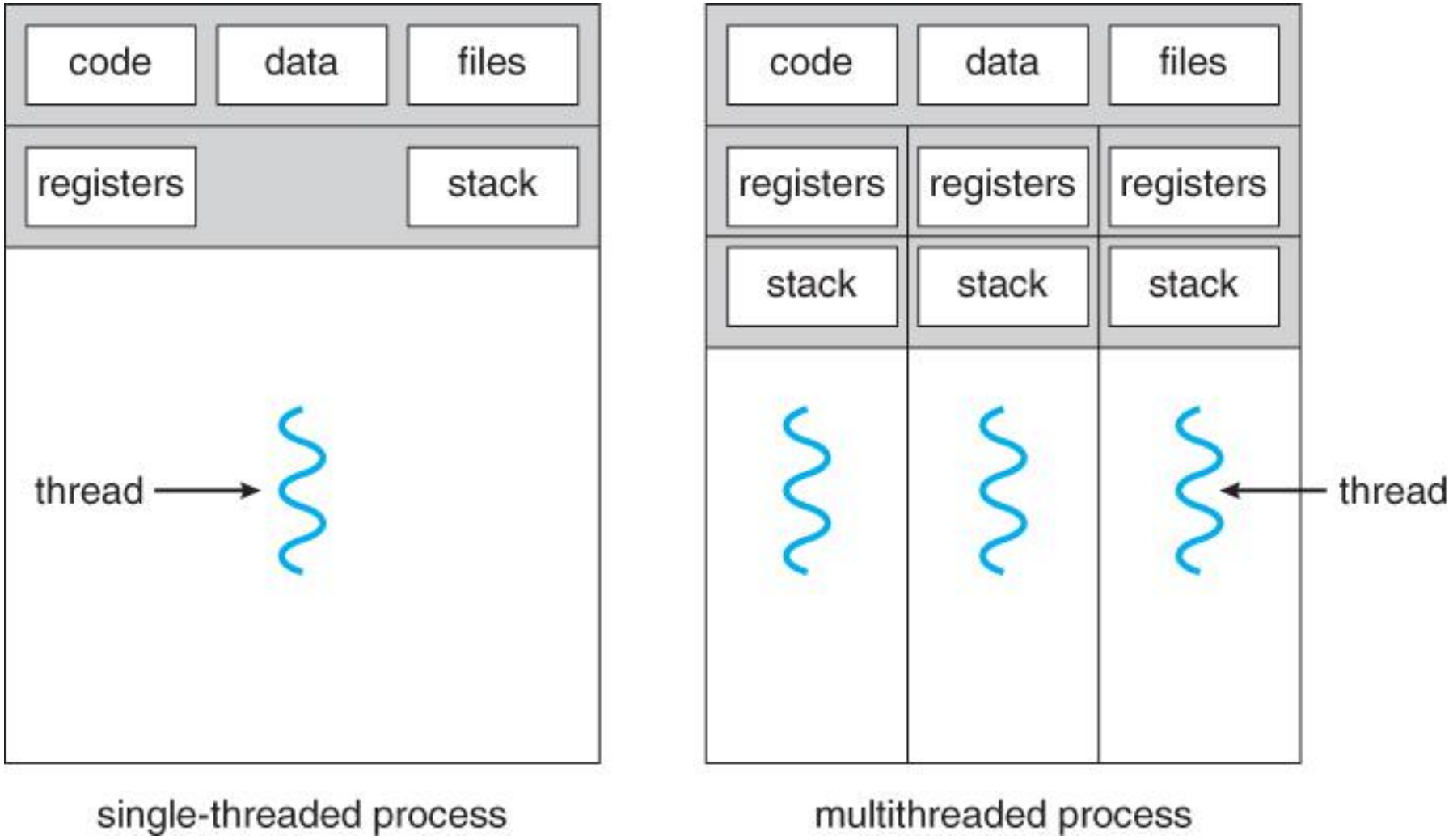
Concept 3: Processus

- **Processus: un environement d'execution avec des droits réduits**
  - Un espace d'adresse avec un ou plusieurs Threads
  - Possede un espace memoire
  - Possede descripteurs de fichiers
  - Encapsule un ou plusieurs Thread.
- Une application consiste d'un ou plusieurs processus.

- **Pourquoi processus?**
  - Se proteger les uns des autres
  - Et proteger le SE
  - Apporter une protection pour la memoire.

- Compromis entre protection et efficacité
  - Communication est facile a l'interieur d'un processus
  - Plus difficile entre les processus

Processus mono-thread vs processus multi-thread



Programme vs Processus

- Un programme se constitute de **code** et **données**.
- Specifié dans un language de programmation.
- Sauvegarder sur fichier qui lui est stocké sur le disque.

- “**Executer un programme**” = **creation d’un processus**.

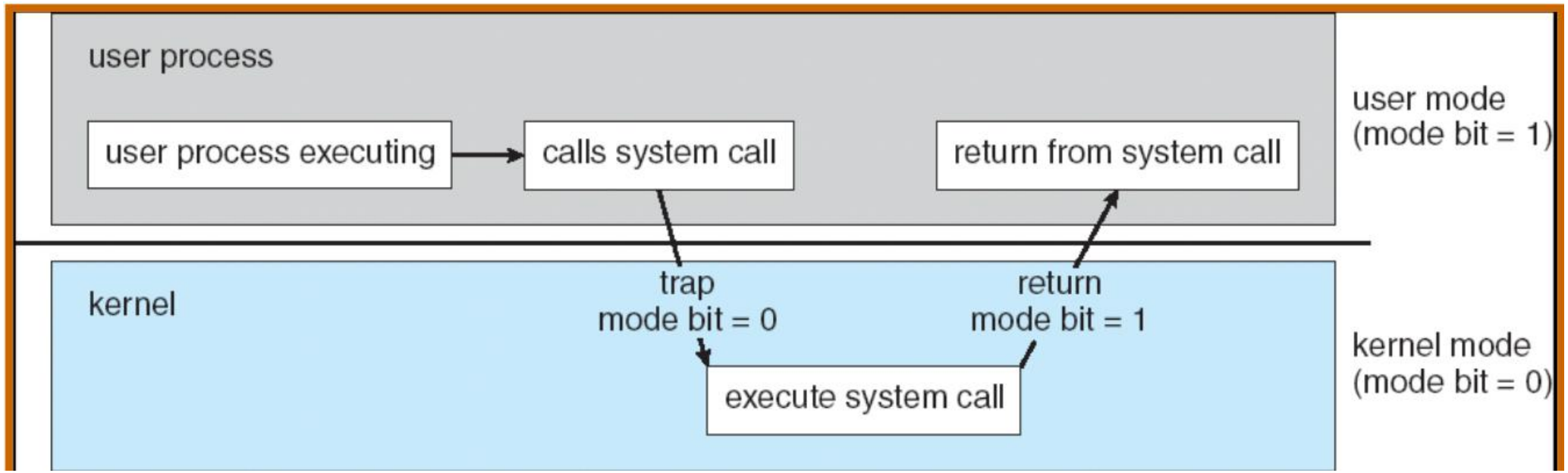
Concept 4 : Concept d'operation en dual-mode

- Il existe deux modes:
  1. **Mode noyau (*Kernel Mode*)**. aussi "supervisor" mode.
  2. **Mode utilisateur (*User Mode*)**

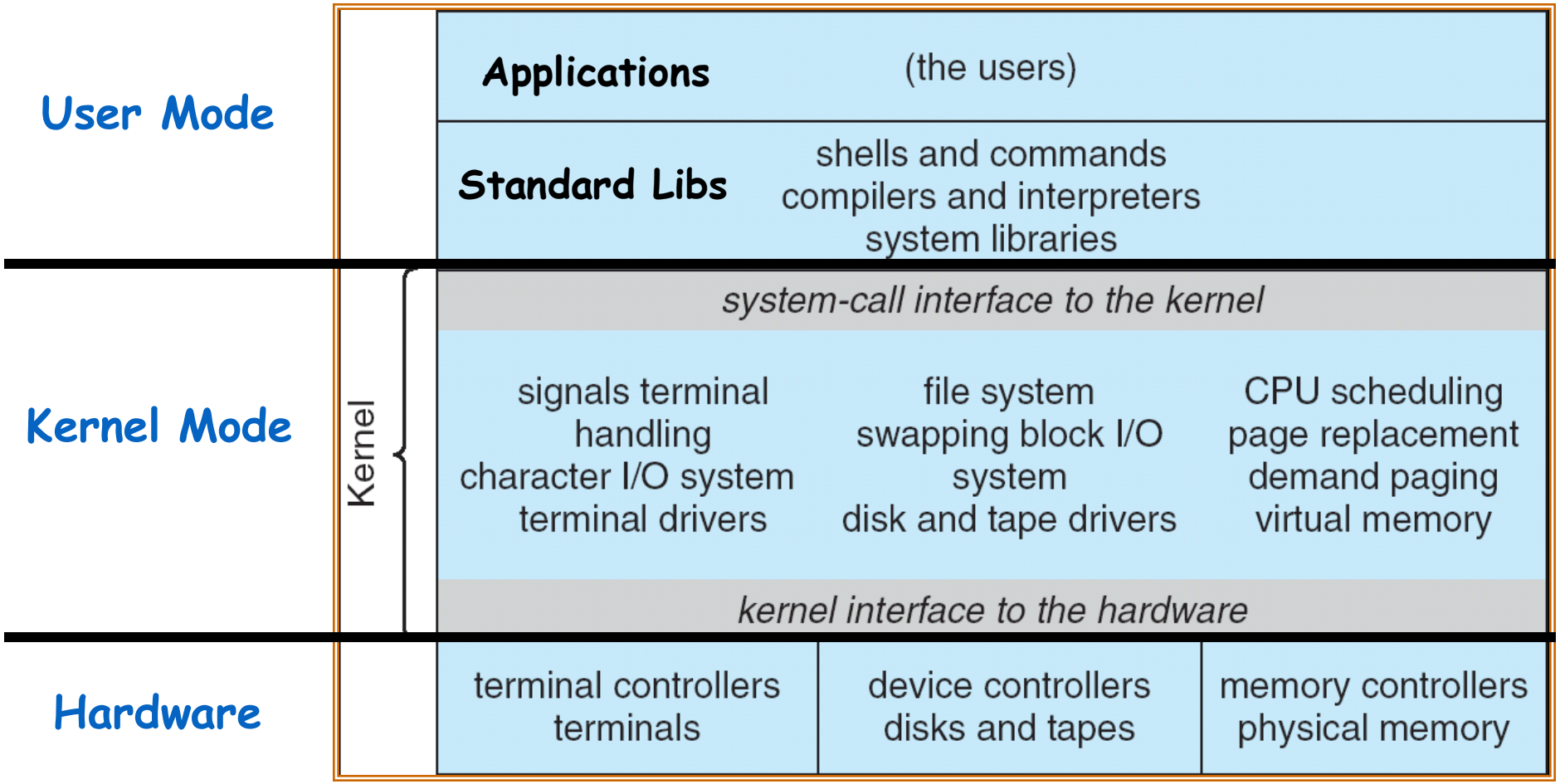
- Certaines operations sont interdites en mode utilisateur.
  - Changement de table de pagination, desactivation des interuptions, interaction directe avec le materiel ou modification de memoire du noyau.

- Controler la transition entre les deux modes
  - Appels système (*system calls*), interuptions, exceptions.

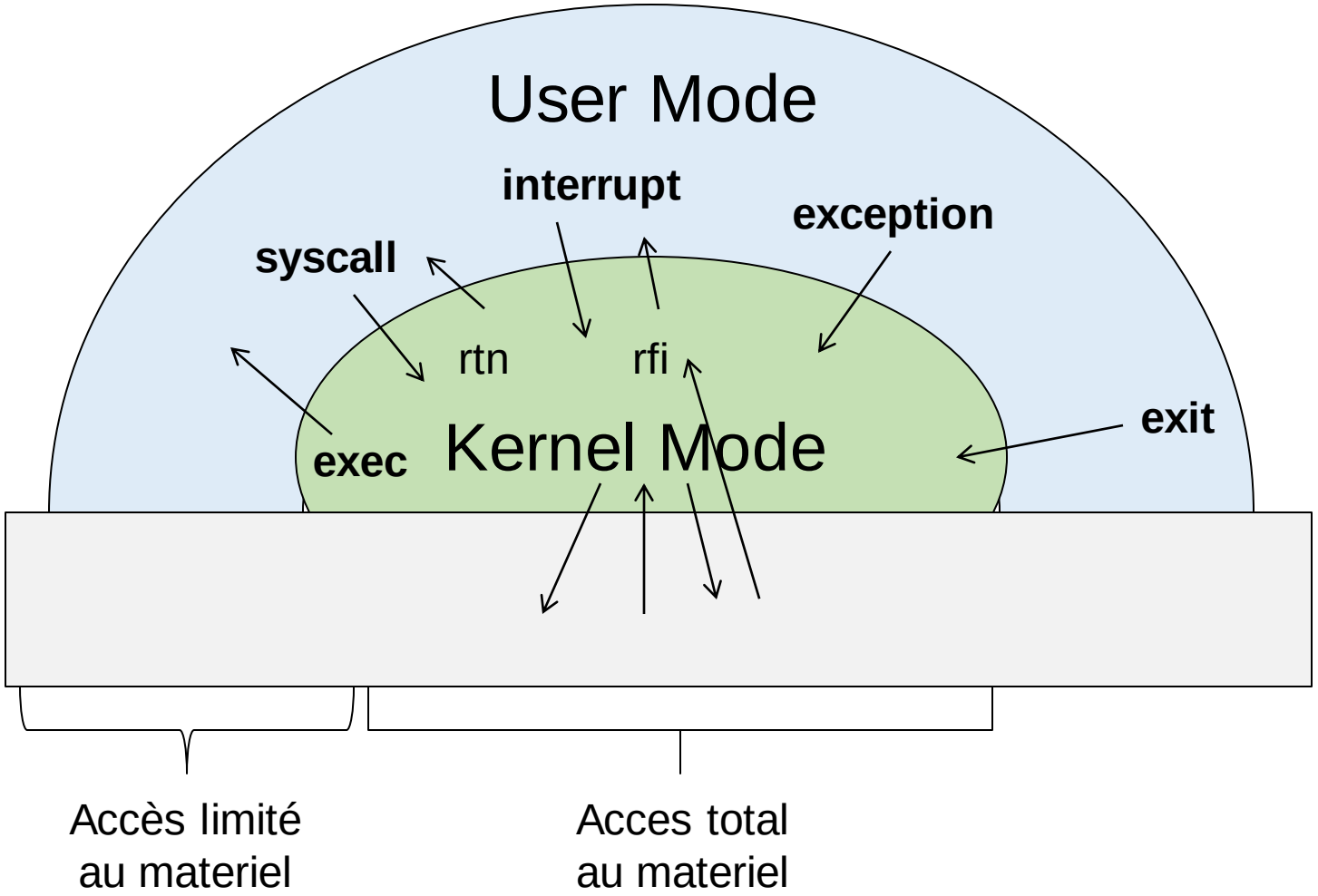
Dual-mode


user process
user process executing
calls system call
return from system call
user mode
(mode bit = 1)
kernel
trap
mode bit = 0
return
mode bit = 1
execute system call
kernel mode
(mode bit = 0)

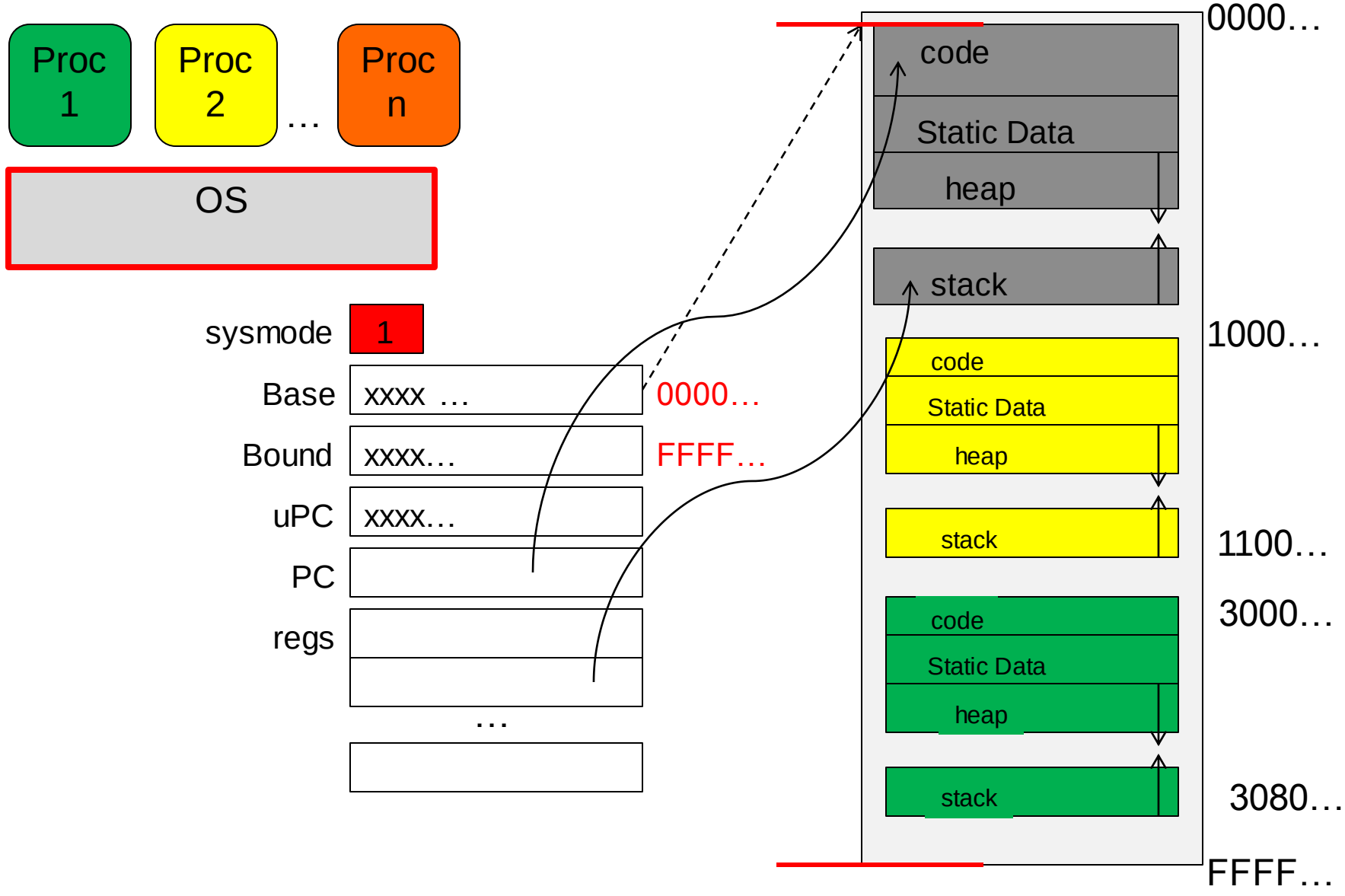
Exemple: Structure du système UNIX



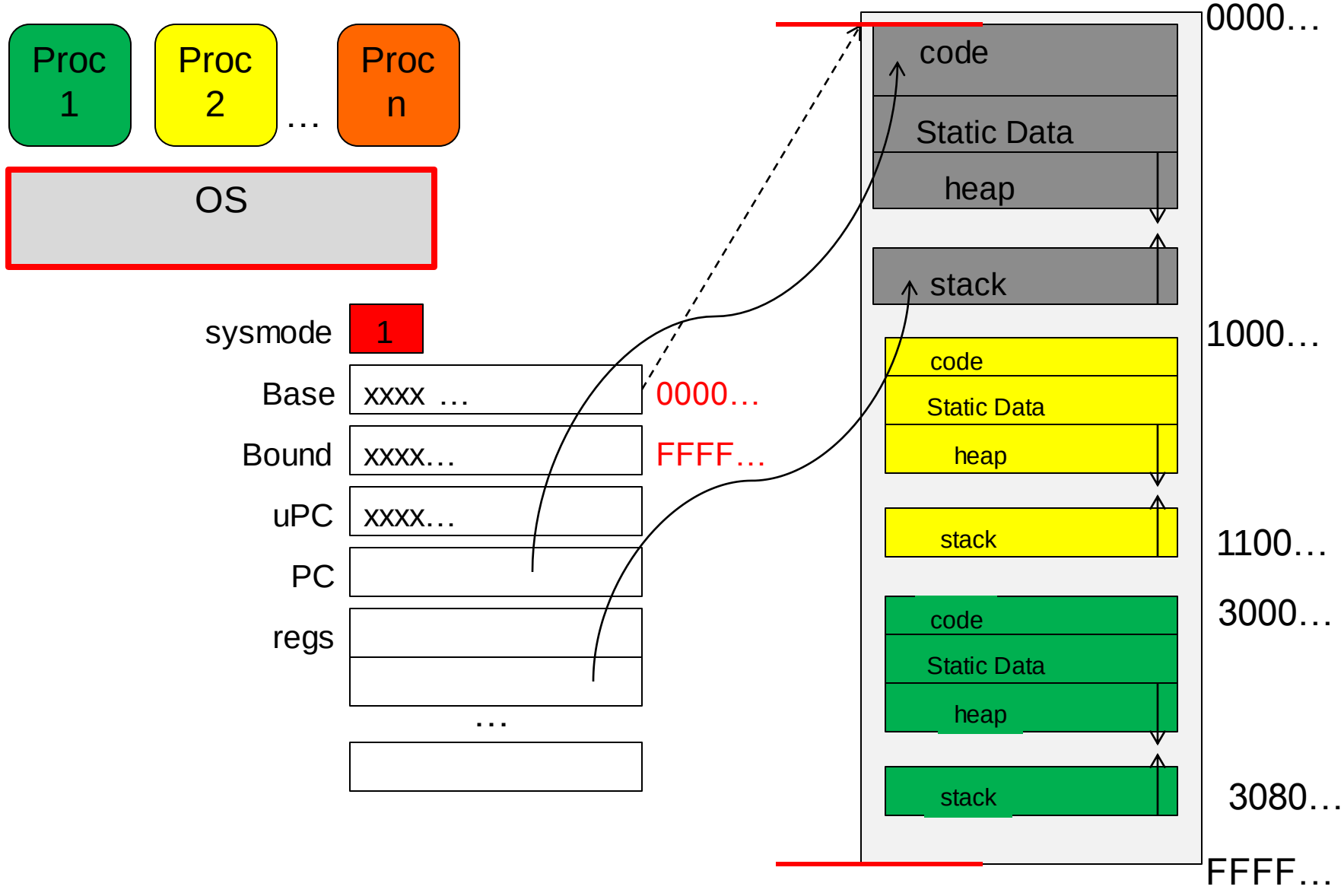
User/Kernel mode



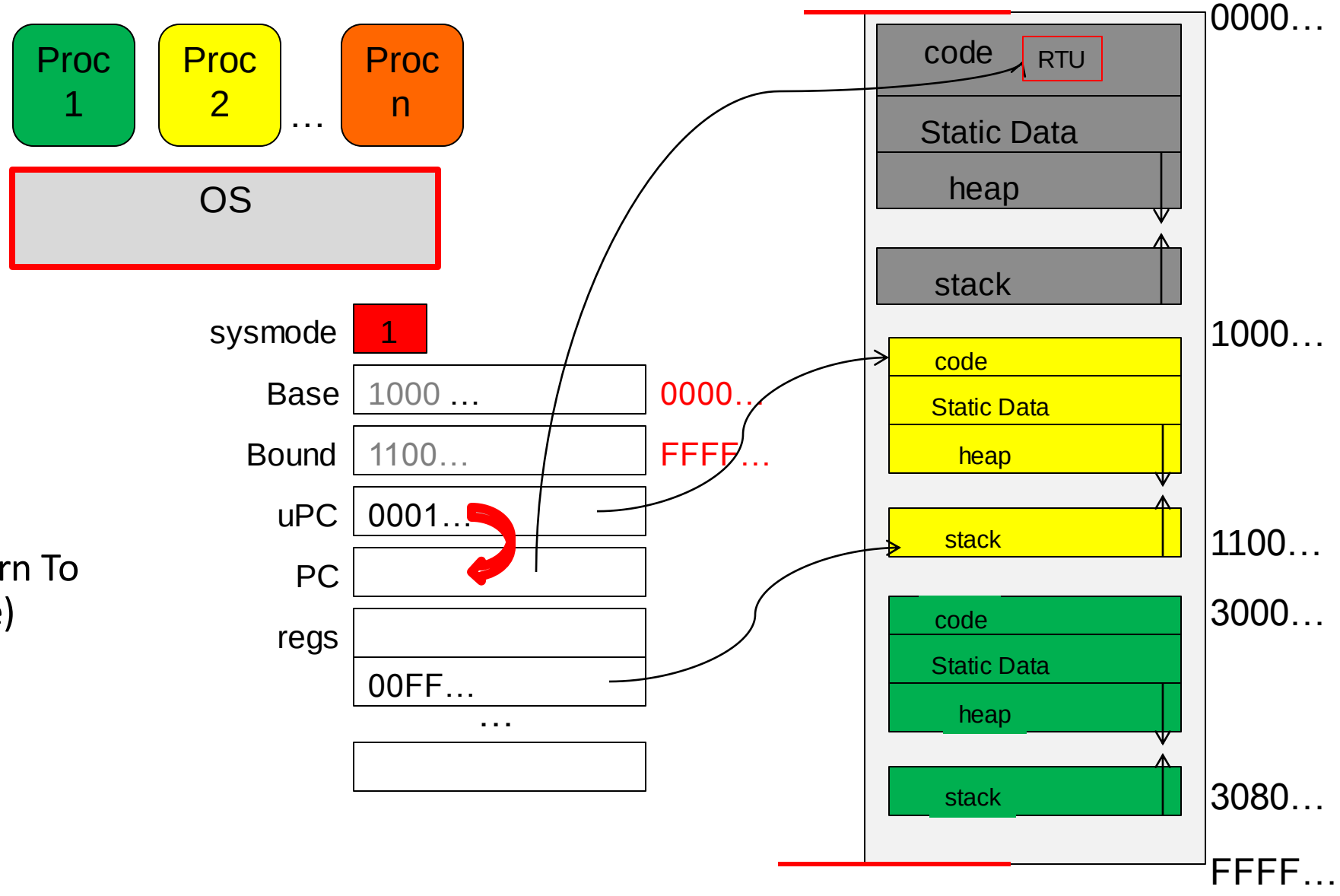
Tying all together



SE charge le processus

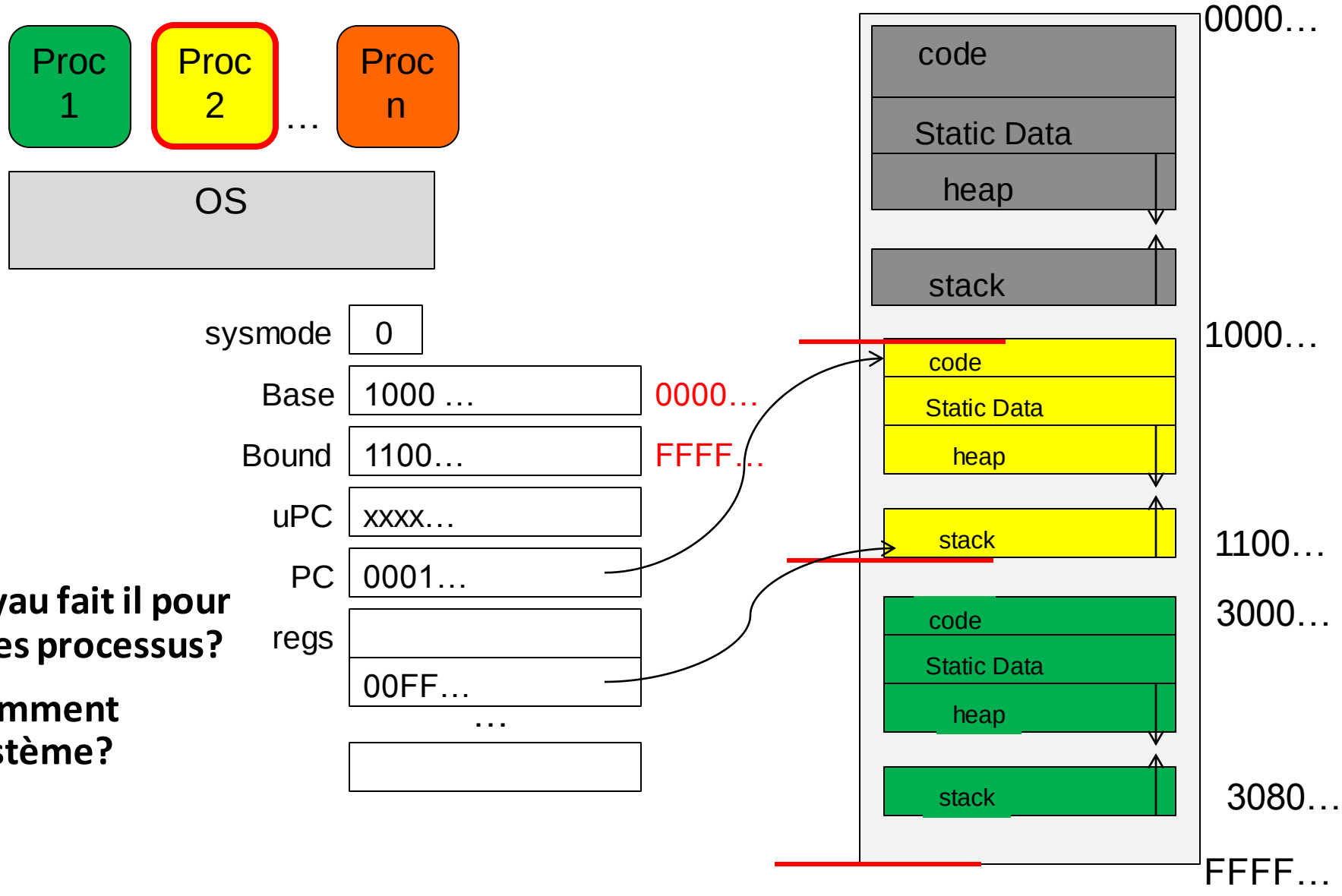


SE se prepare pour executerer le processus



- RTU (Return To Usermode)

Code en execution



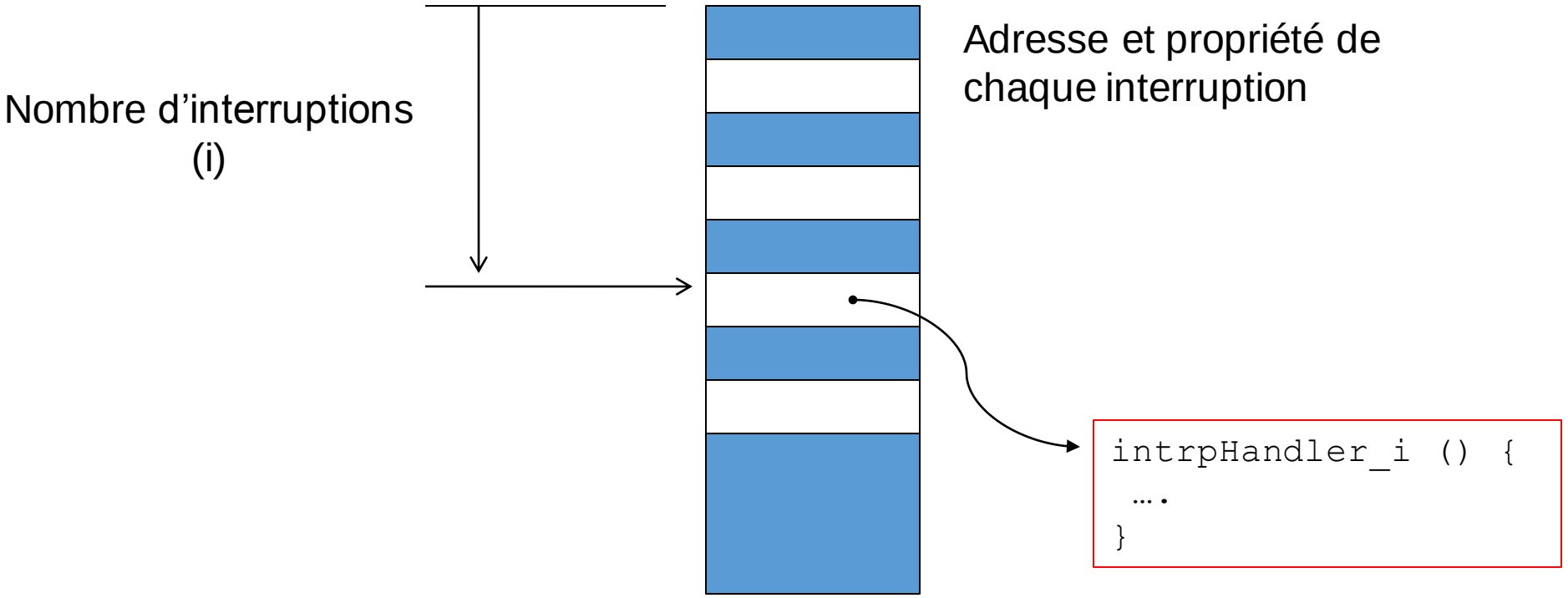
- **Comment le noyau fait il pour switcher entre les processus?**
- **Tout d'abors: comment retourner au système?**

3 Types de changement « User-->Kernel »

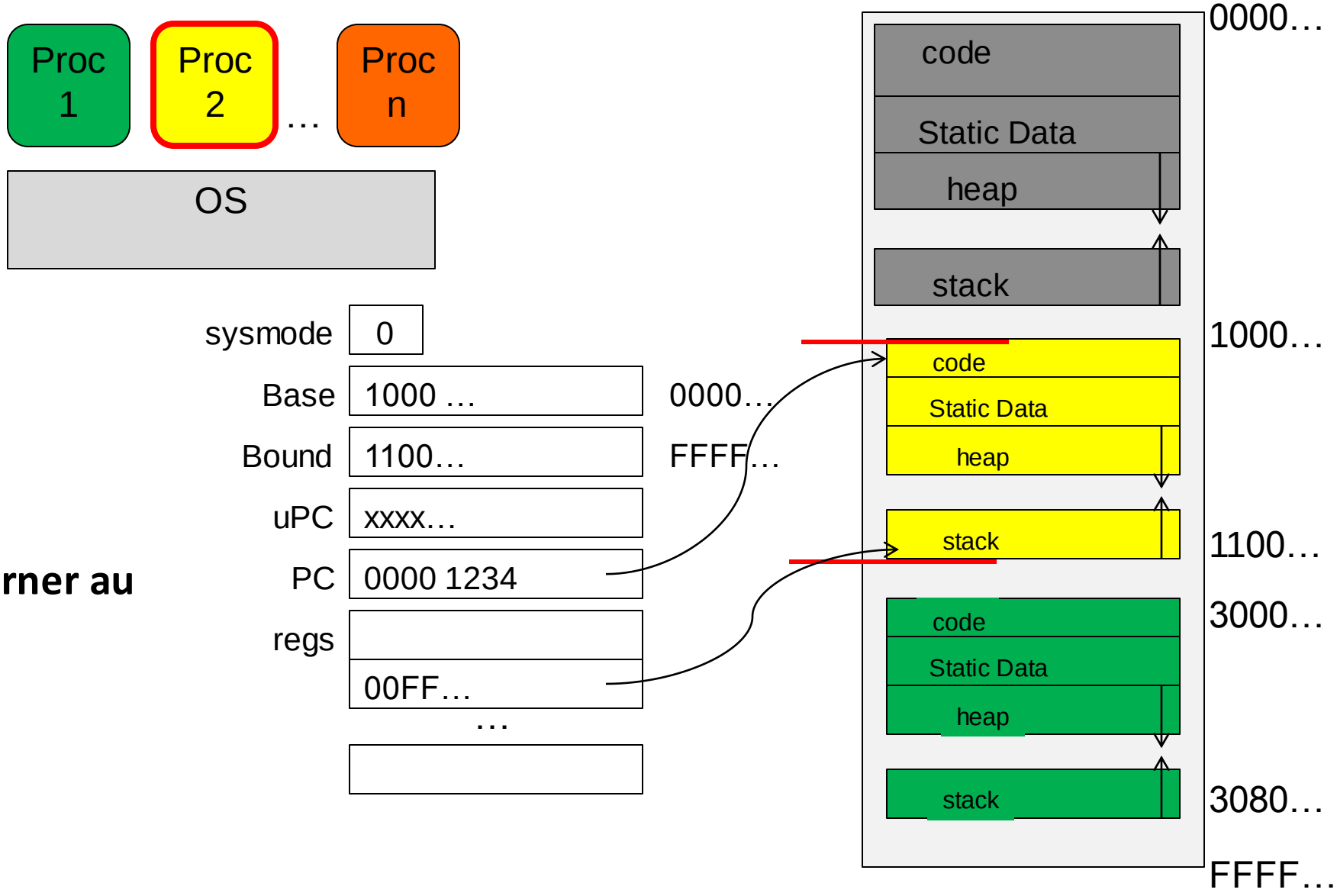
- Appels systèmes (*syscall*)
  - Processus demande un service système, comme “exit”
- Interuptions (*Interrupt*)
  - Timer, I/O
  - Independant du processus utilisateur.
- Exception
  - evenement interne au niveau du processus
  - Violation de protection (*segmentation fault*), division par zero, …

Ou sont sauvegarder ces interruptions?

Vecteur d'interruptions (*Interrupt Vector*)

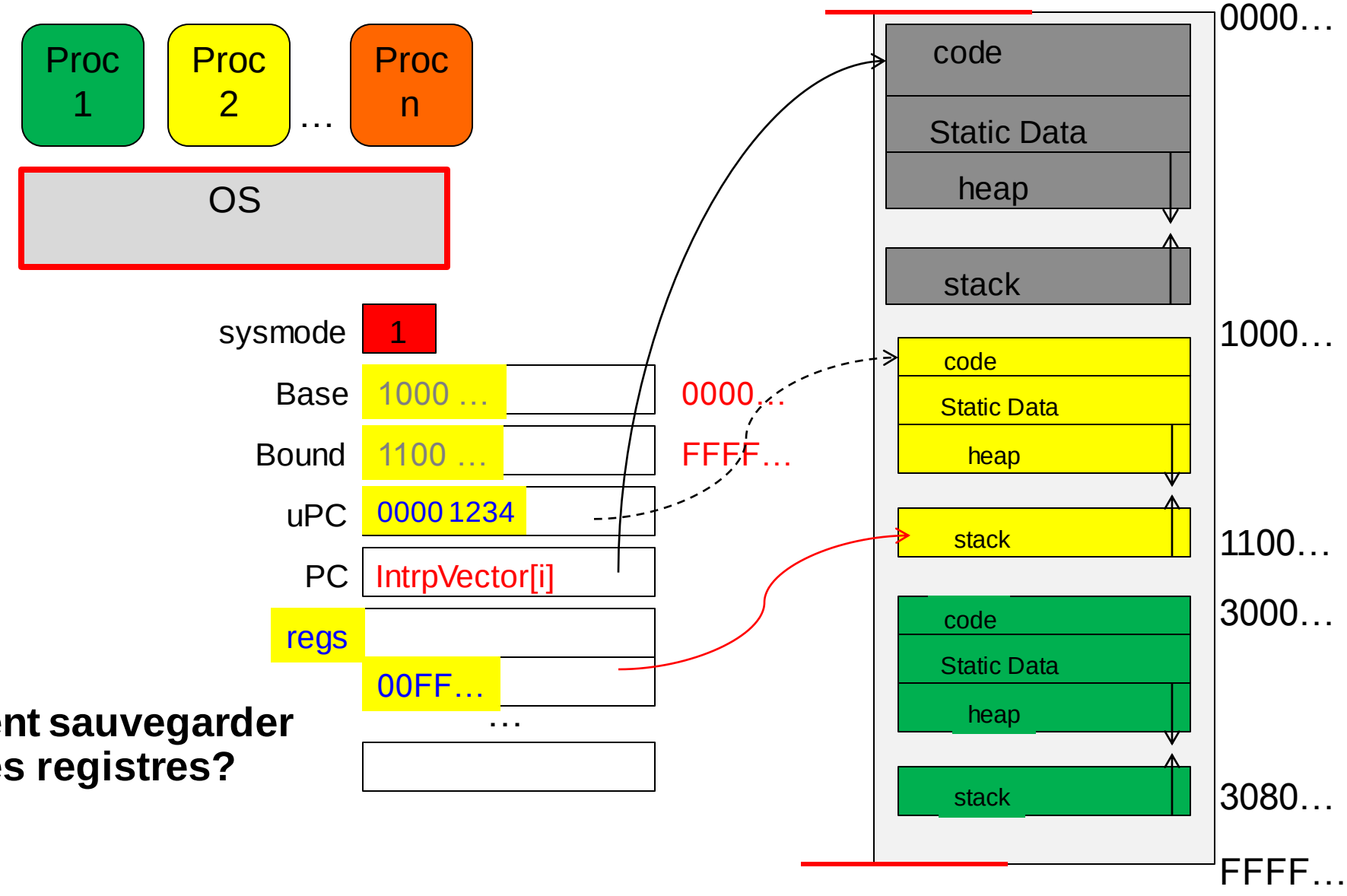


User mode --> Kernel mode



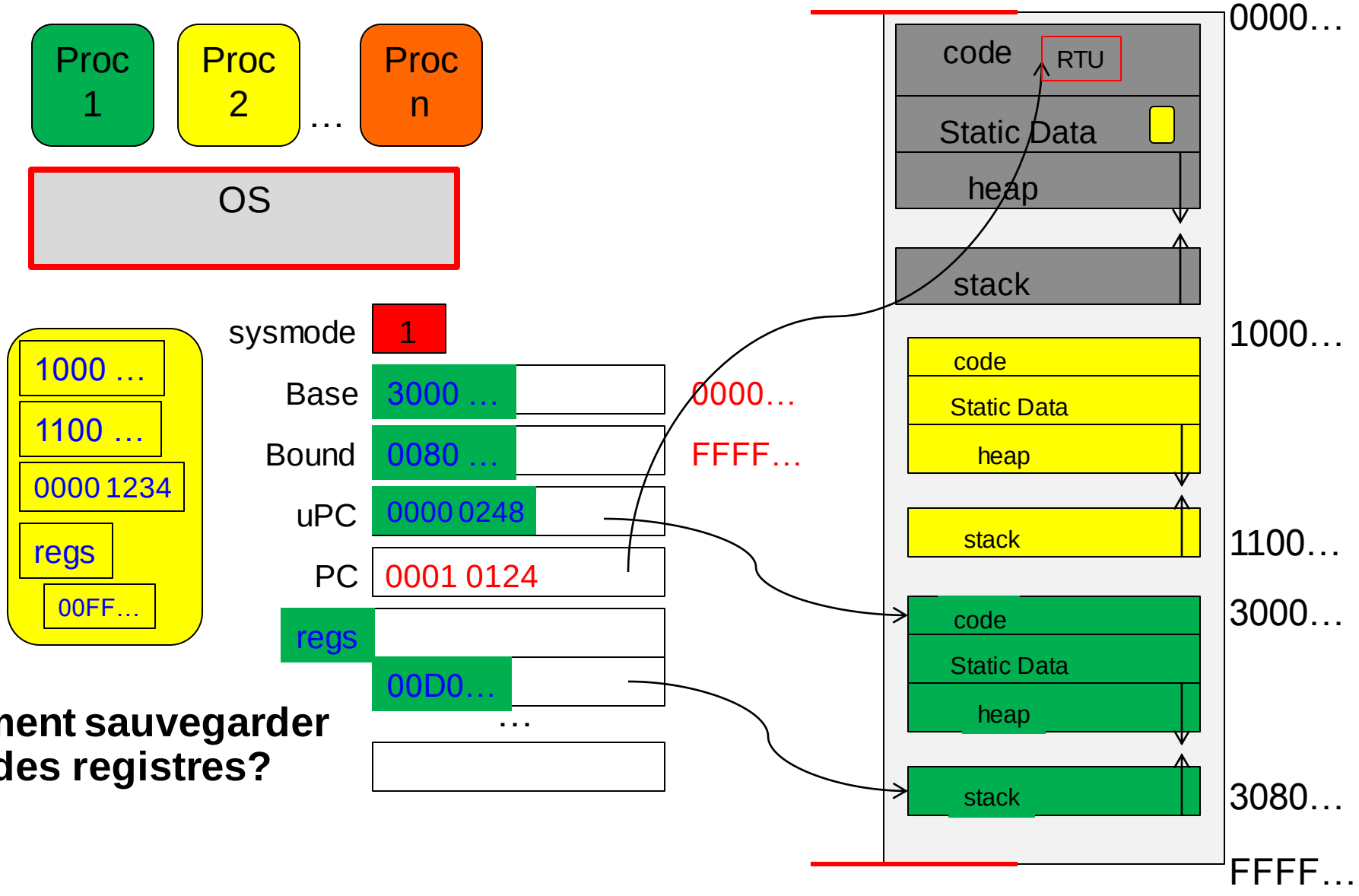
- **Comment returner au système?**
  - Timer
  - Requêtte E/S
  - Autre

Interruption



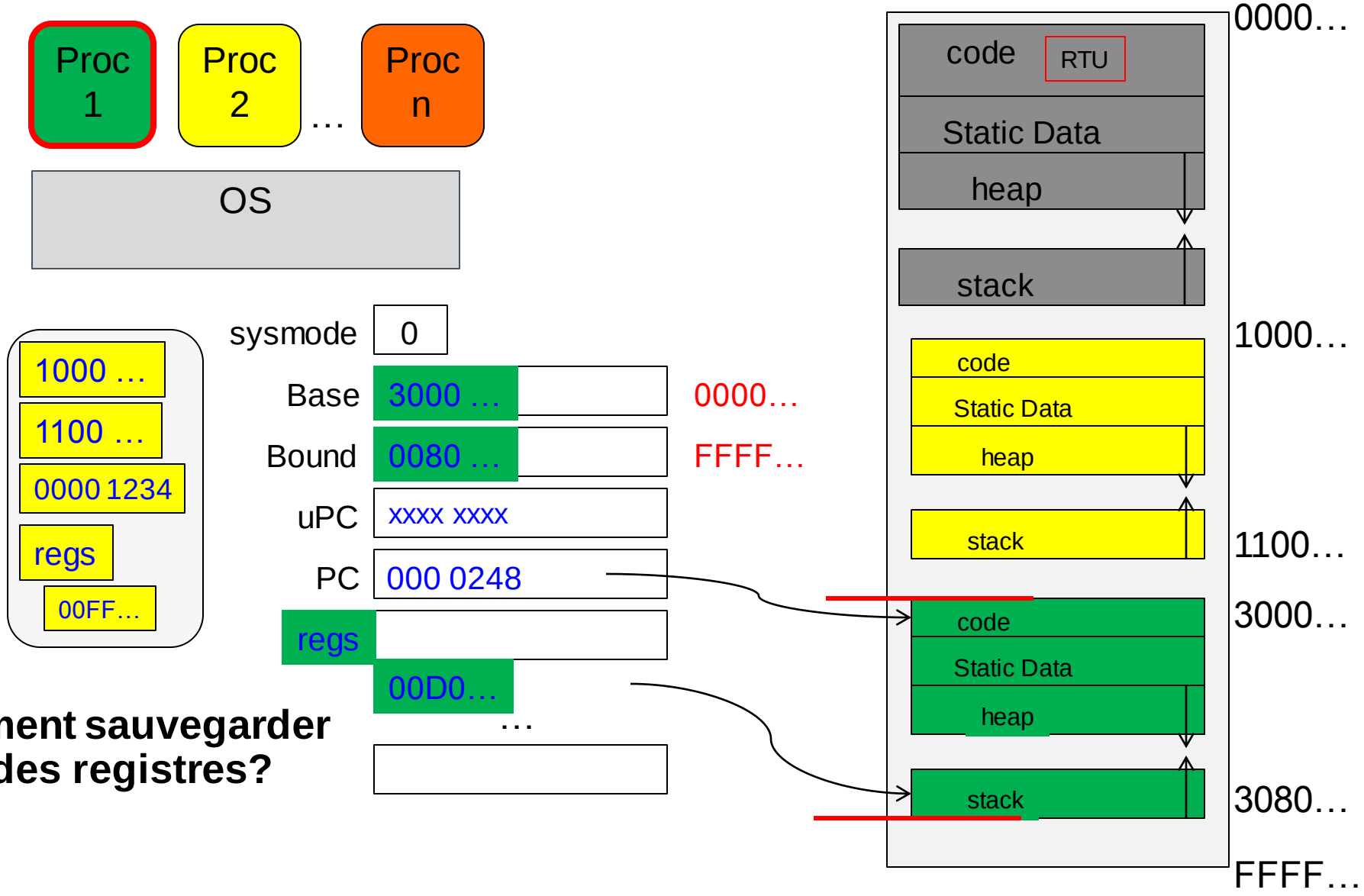
- **Comment sauvegarder l'etat des registres?**

Switcher vers deuxième processus



- **Comment sauvegarder l'etat des registres?**

deuxième processus en cours d'execution

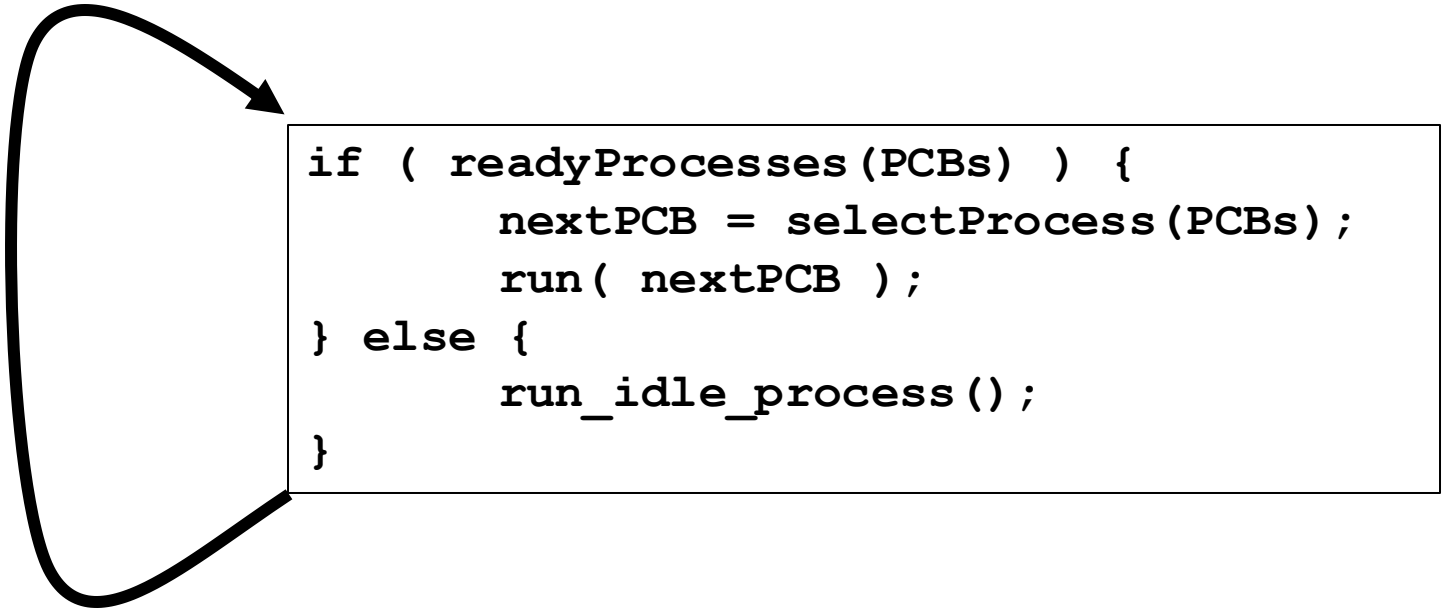


- **Comment sauvegarder l'etat des registres?**

- Nous avons les mechanism basics pour:
  - Switcher entre user et kernel mode
  - Le noyau peut switcher entre les different processus
  - Le SE peut proteger les processus les uns des autres

- **Questions**
- Comment choisir quel processus executer?
- Comment sauvegarder l'etat du processus?
- Comment representer les processus au niveau du SE
- …

Le planificateur (*Scheduler*)



```
if ( readyProcesses(PCBs) ) {
        nextPCB = selectProcess(PCBs);
        run( nextPCB );
} else {
        run_idle_process();
}
```

Process Control Block (PCB)

- Le noyau represente chaque processus comme un “block de control” (process control block)
  - Etat (execution, prêt, bloquer, …)
  - L’etat des registres
  - Process ID (PID), utilisateur, priorité, …
  - Temps d’execution
  - L’espace memoire, …
- Le planificateur du noyau (*Kernel Scheduler*) maintient une structure de données contenant les PCBs.
- L’algorithme de planification choisi le prochain processus.

Conclusion

Conclusion

- **Thread**
  - Décrit un etat du programme.
  - Pointeur d'instruction (*Program Counter*), Registres, Pile, …
- **Espace d'adresse (*address space*)**
  - Une plage d'adresse accessible pour le programme (lecture/ecriture).
- **Processus: une instance d'un programme en execution**
  - Espace memoire protégé.
  - Un ou plusieurs Threads.
- **Dual mode operation / Protection**
  - Seul le SE a accès a certaines ressources.
  - Isoller les programmes les uns des autres, et proteger le SE des programmes.