Systèmes d'exploitation - Gestion de la mémoire

Pierre Gançarski

DUT Informatique - S31

ATTENTION
Ces transparents ne sont qu'un guide du cours : de nombreuses explications et illustrations manquent.

De nombreuses précisions seront données au tableau et à l'oral pendant le cours.

Université de Strasbourg

IUT Robert Schuman
UNIVERSITÉ DE STRASBOURG

Plan

Mémoire

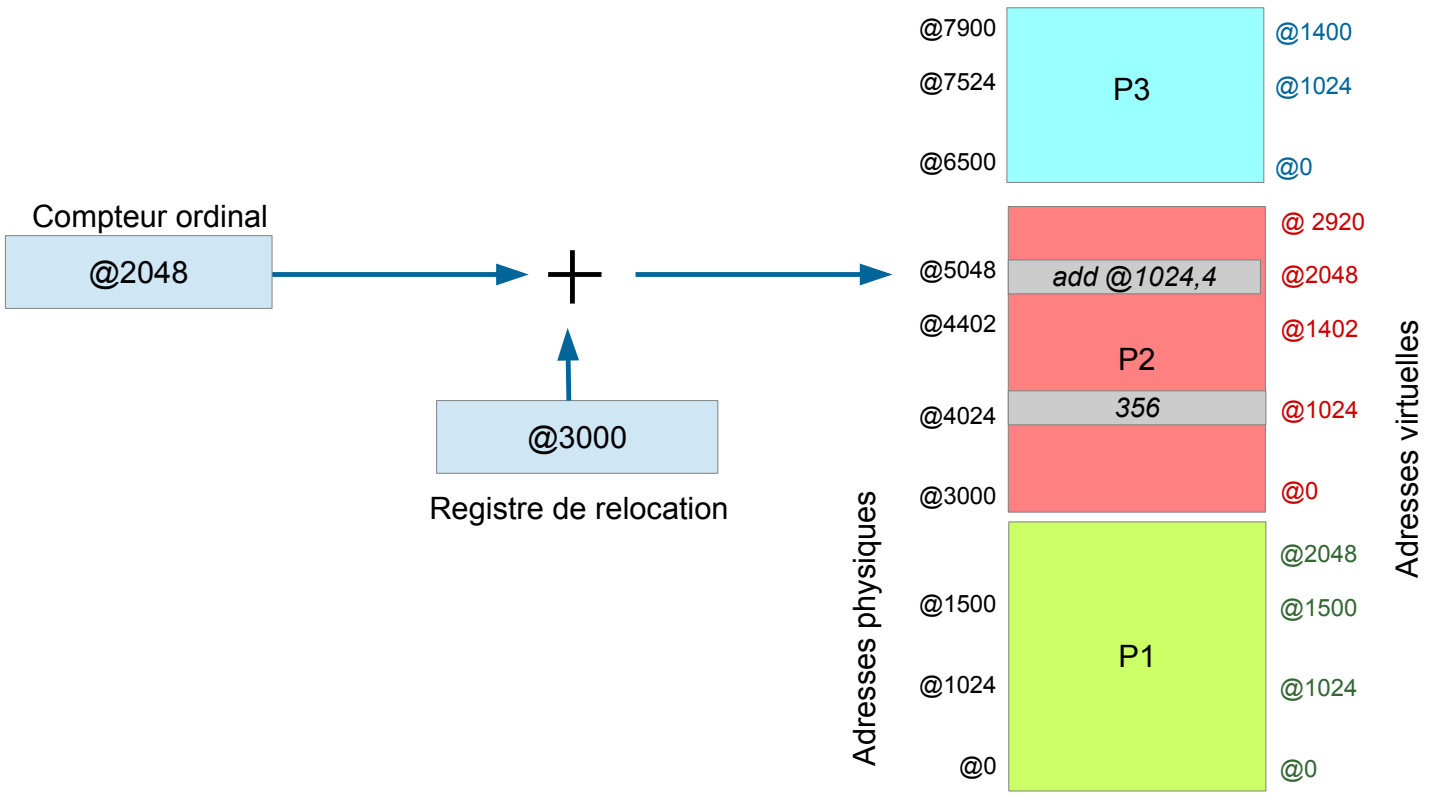
Problématique

- Les premiers ordinateurs (avant 1960, mini-ordinateurs avant 1970, micro-ordinateurs avant 1980,...) ne disposaient pas de mécanisme d'abstraction de la mémoire.
- Chaque programme accédait directement à la mémoire physique
- Comment exécuter simultanément plusieurs programmes ?
  - Va-et-vient ou swapping : le SE recopie sur disque l'intégralité du contenu de la mémoire avant de charger et d'exécuter le programme suivant.
  - Nécessité d'un matériel spécialisé (IBM 360).
  - Mais restait très coûteux et peu pratique.

Mémoire

Solution possible

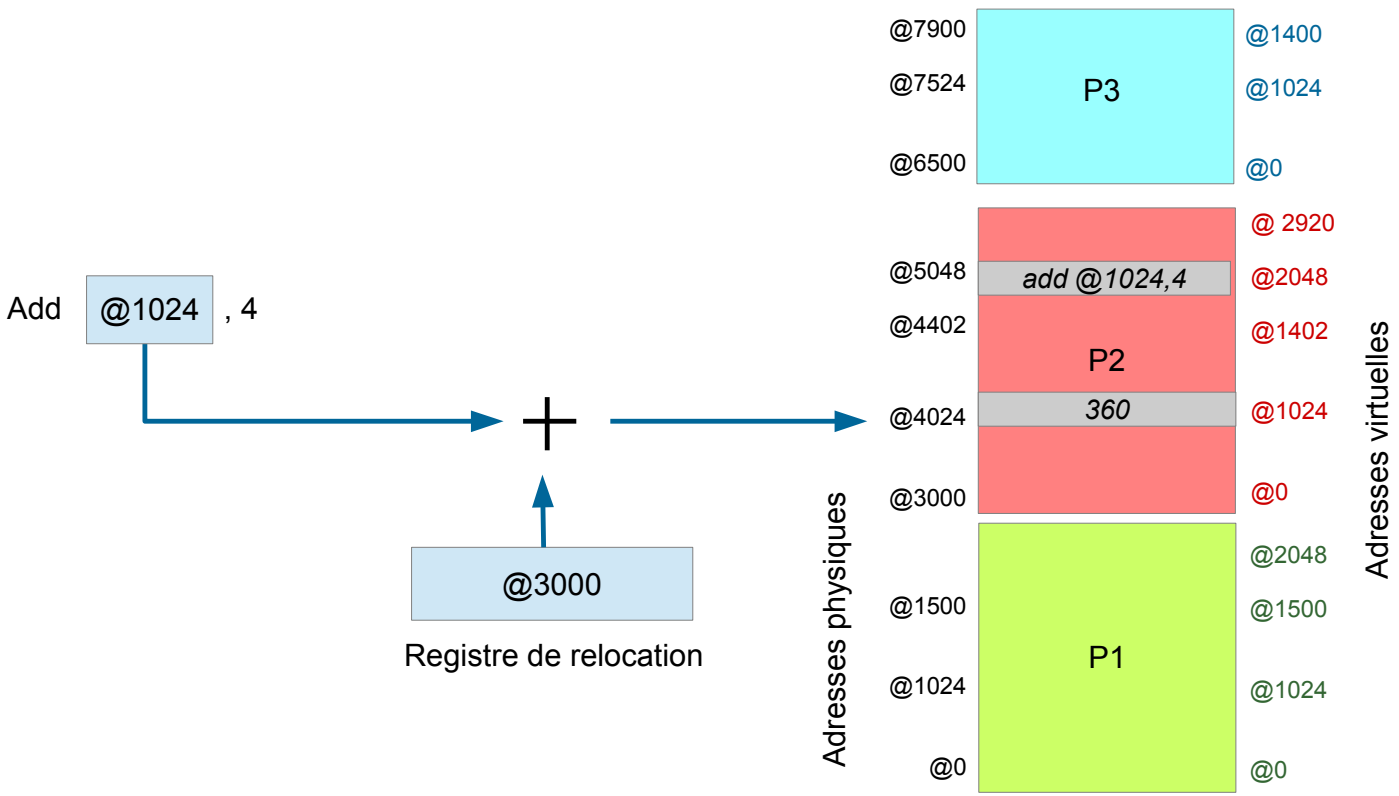
- Les processus sont chargés en mémoire → Allocation statique : on ajoute à chaque adresse l'**adresse de relocation**
- Exemple : on suppose que le processus P2 effectue l'instruction à l'adresse @2048 *dans son espace d'adressage*



Mémoire

Solution possible

- Les processus sont chargés en mémoire → Allocation statique : on ajoute à chaque adresse l'**adresse de relocation**
- P2 doit alors accéder à l'adresse @1024 pour incrémenter la case mémoire



Mémoire

Le va-et-vient (swap)

- Que faire si la taille totale des processus est supérieure à la totalité de la mémoire physique ?
- Impossibilité de maintenir tous les processus en mémoire.
- Stratégie la plus simple : va-et-vient (swap). Certains processus sont transférés intégralement sur le disque

Segmentation

- Comment réduire la taille des processus → partager du code : noyau, librairies ... (qui représentent très souvent bcp plus de code que le programme lui-même)

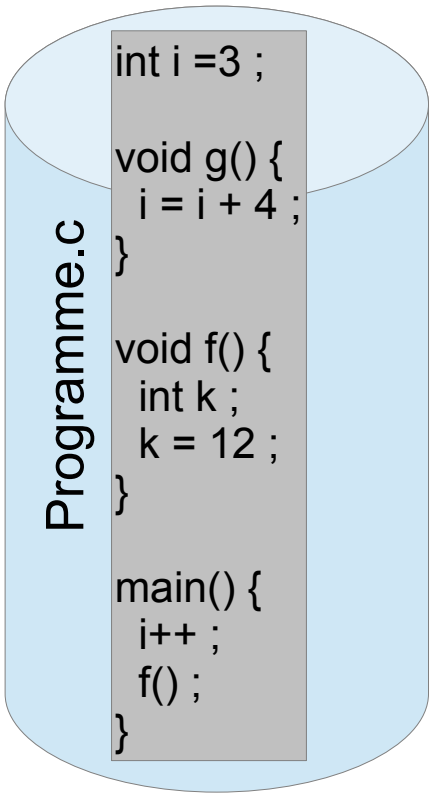
→ Segmentation

Plan

La segmentation

Principe

- Séparer les différentes parties d'un processus : **segments**.
  - Chaque segment est une suite d'adresses contiguës
  - Le processus manipule explicitement les segments → les adresses sont de la forme `< Numéro segment, Dépl. dans le segment >`



Compilation

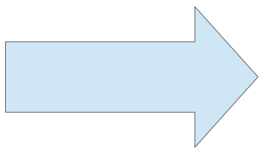
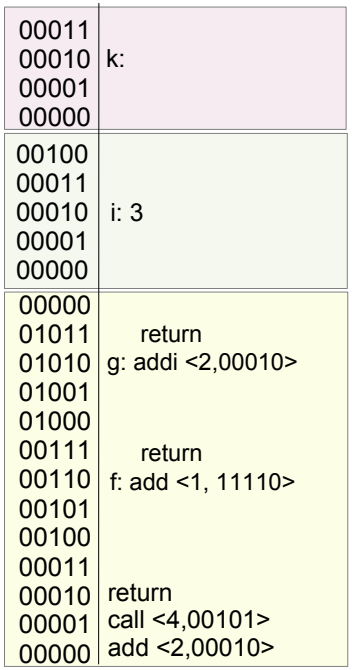


Table des segments

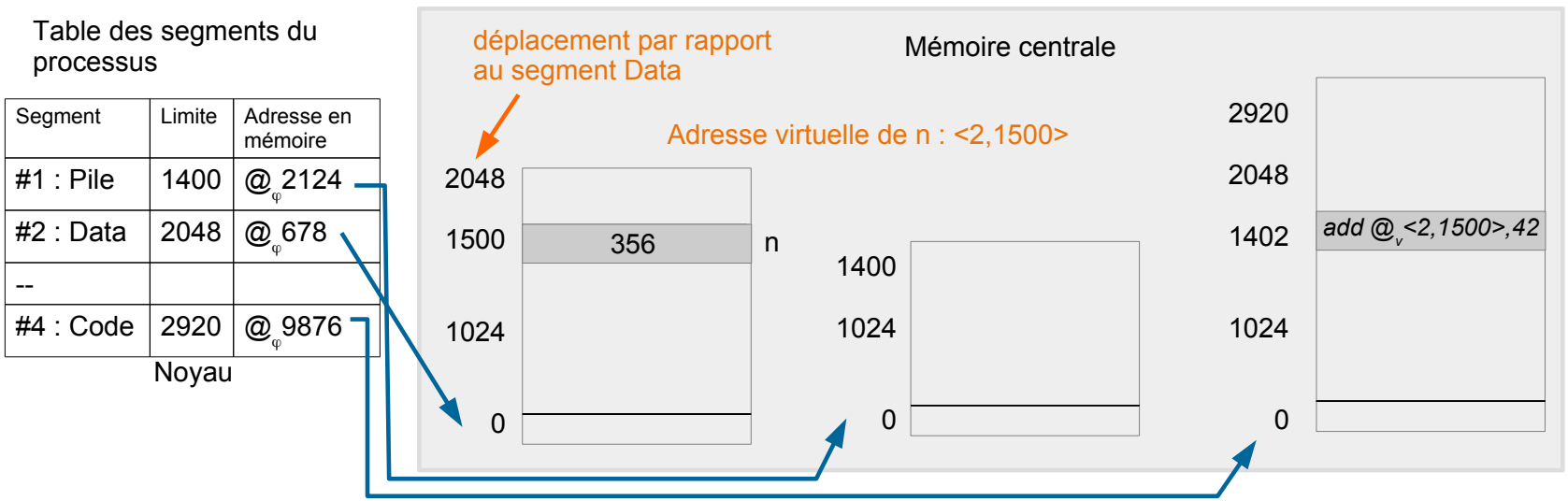
Segment	Limite
#1 : Pile	4
#2 : Data	5
--	
#4 : Code	12



La segmentation

Principe

- À l'exécution les segments sont chargés dans des zones mémoire non nécessairement contigues.
  - La table des segments donne les adresses de base et la taille des segments
  - Le CO du processus est donné par rapport au segment de code

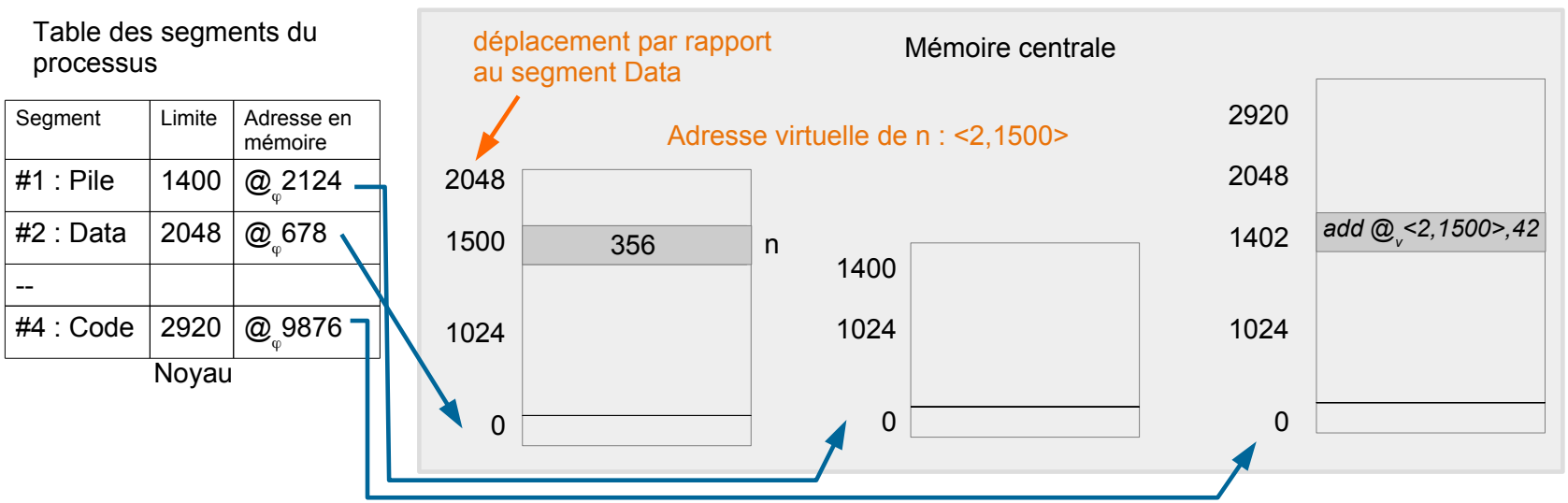


Quelle est l'adresse physique de n d'adresse virtuelle $@_v < 2, 1500 >$?

La segmentation

Principe

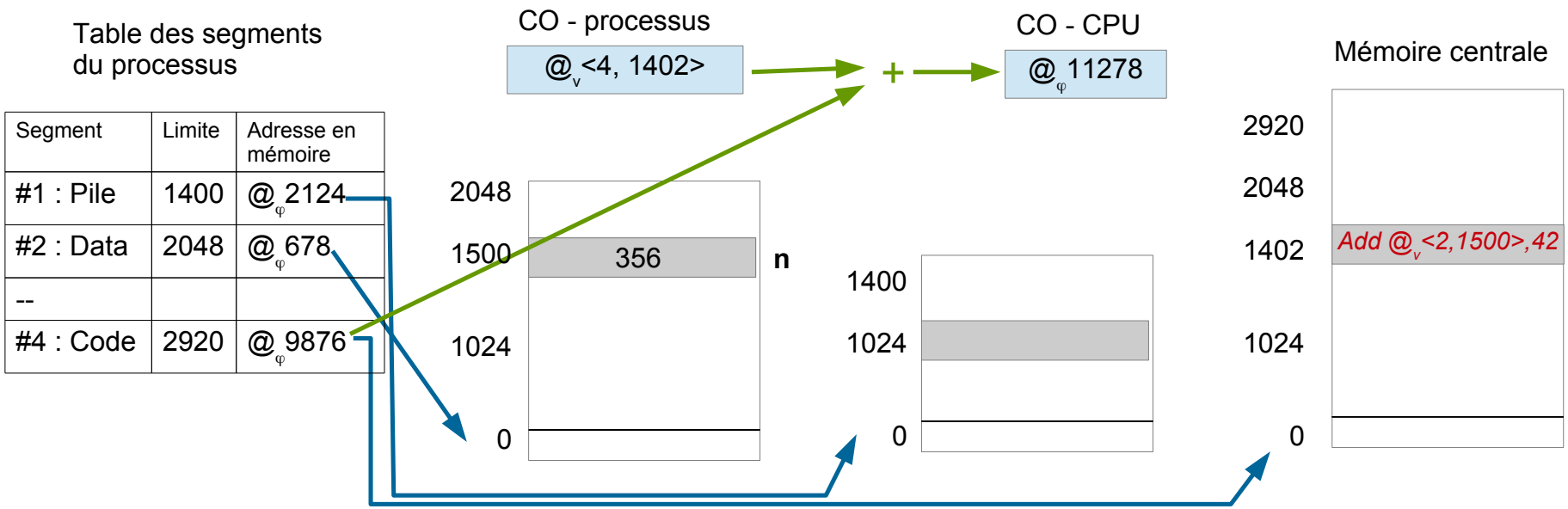
- La traduction d'adresse se fait simplement par addition de la base du segment et du déplacement
  - $@_v < 2, 1500 > = @_\phi 678 + 1500 = @_\phi 2178$



La segmentation

Principe

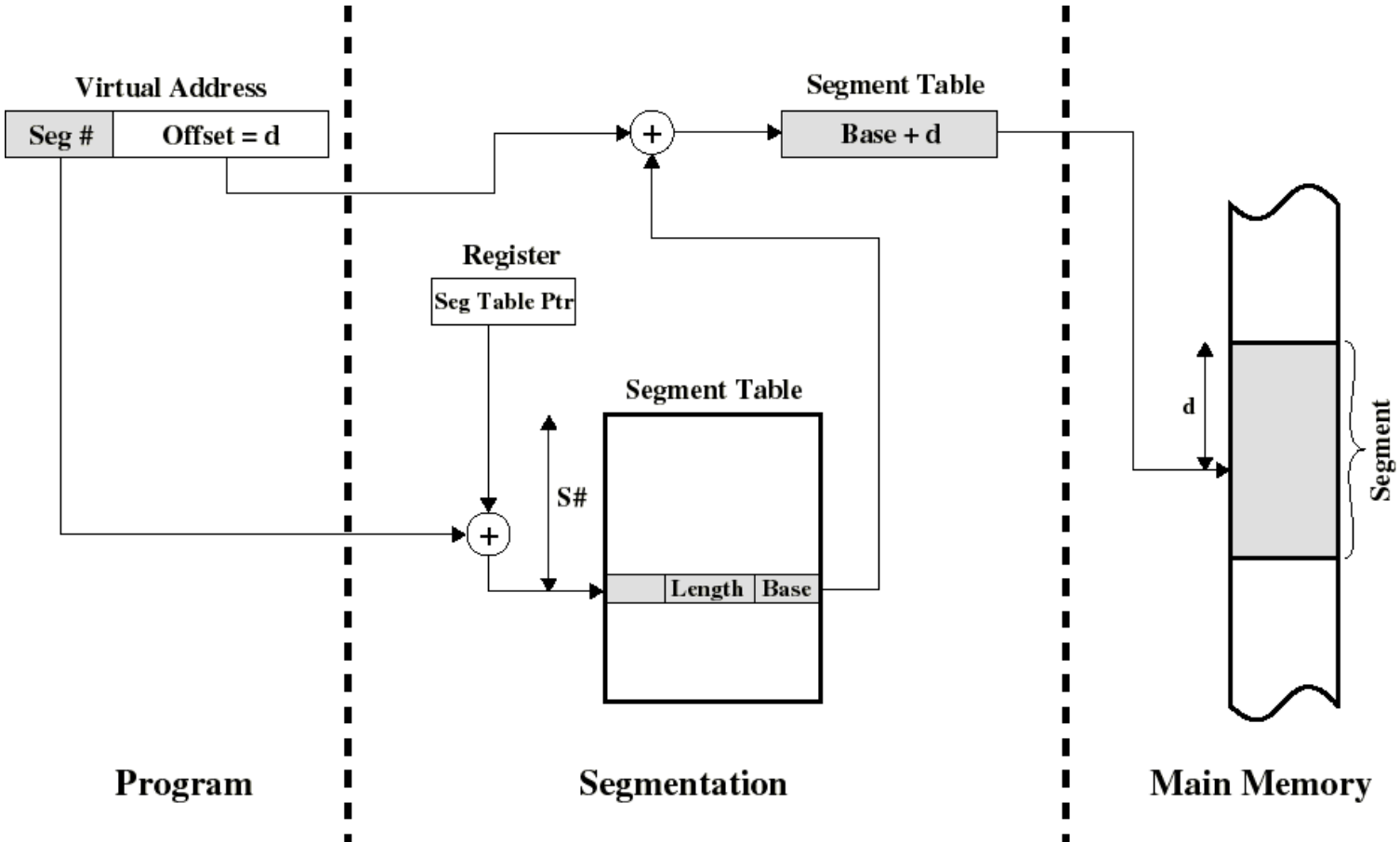
- La traduction d'adresse se fait simplement par addition de la base du segment et du déplacement
  - Idem pour le compteur ordinal



La segmentation

Traduction d'adresses

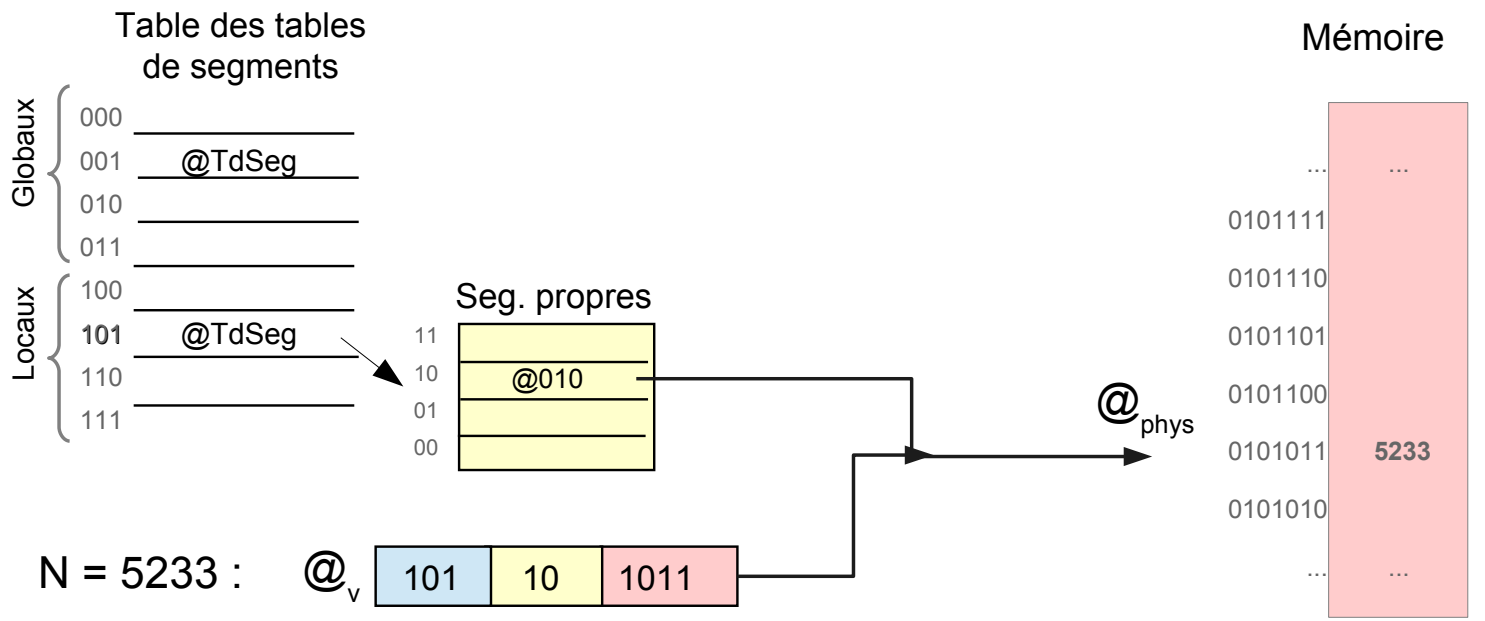
- Ces adresses `< Numero , Dépl.>` sont traduites par la MMU
- MMU (Memory magement unit) : composante électronique intégrée directement dans le processeur



La segmentation

Cas d'Unix

- Les tables de segments sont référencées dans une table "système" : chaque processus dispose d'entrées dans cette table
- Les tables donnent les adresses de relocation
- Tab. de seg. globaux –> partage de segments : bibliothèques, mémoires partagées
- Tab. de seg. locaux –> espace d'adressage propre aux processus



La segmentation

Avantages

- À l'exécution n'est chargé que ce qui est utilisé :
  - Les trous entre les segments ne sont pas "alloués"
  - Les segments peuvent grandir (on peut allouer au départ des zones mémoire minimales)
  - La pile d'une fonction enclenchée est créée dynamiquement
  - L'appel système `malloc` peut faire grossir le segment "Tas"

La segmentation

Fork et Execv

- À l'exécution d'un `fork` :
  1. Le père passe en mode noyau et lance la fonction noyau `_fork`
  2. La fonction `_fork`
     - 2.1 cherche une entrée libre dans la table de processus pour le contexte du fils
     - 2.2 recopie le contexte du père dans ce contexte (y compris le CO)
     - 2.3 le modifie en associant un nouvelle table des segments à ce processus
     - 2.4 recopie les "base - limites" de tous les segments SAUF "Pile" et "Tas"
     - 2.5 crée les segments Pile et Tas et recopie leurs contenus depuis le père
     - 2.6 met le pid du fils crée dans la pile du père et 0 dans celle du fils
     - 2.7 le processus fils est mis en mode "activable" (runnable) et donc apparait dans le tourniquet
  3. Le père repasse en mode utilisateur

La segmentation

Fork et Execv

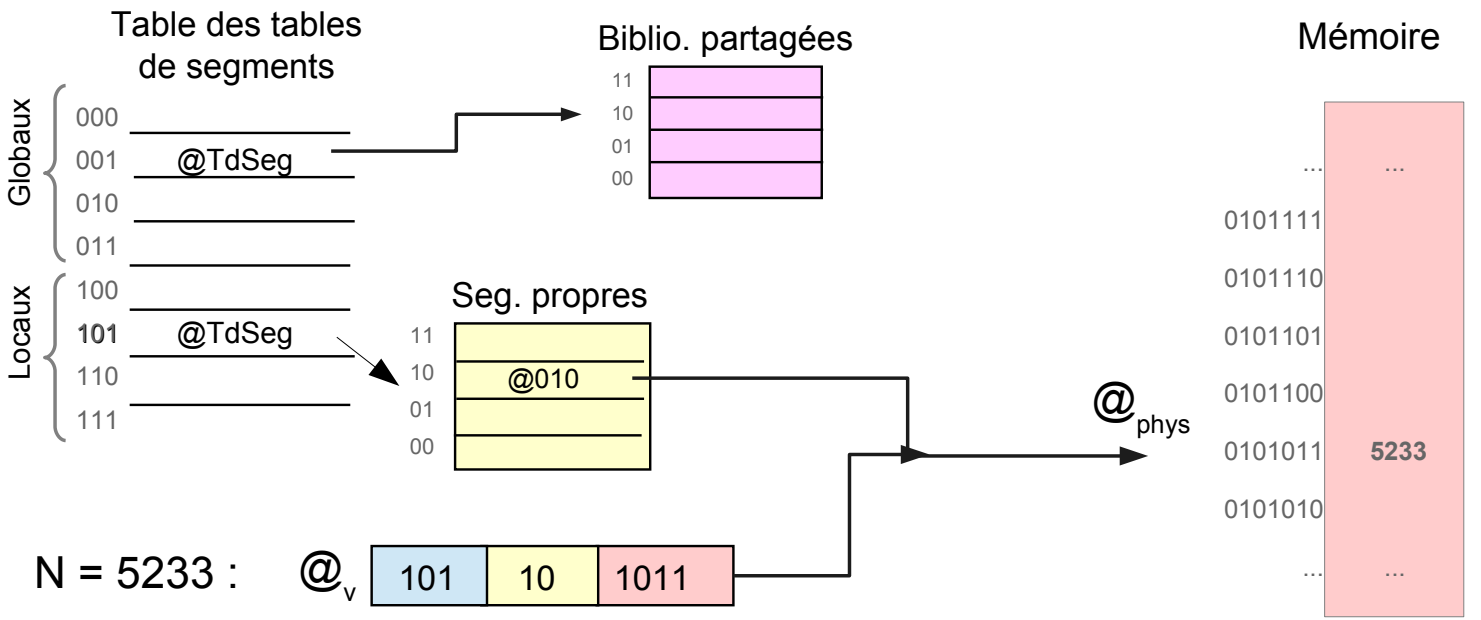
- À l'exécution d'un `execv` :
  1. Le processus passe en mode noyau et lance la fonction noyau `_execv`
  2. La fonction `_execv`
     - 2.1 cherche une entrée dans la table de processus correspondant au contexte de ce processus
     - 2.2 le modifie en supprimant la table des segments associée puis en associant un nouvelle table des segments à ce processus
     - 2.3 crée les segments Code, Pile et Tas
     - 2.4 recopie le contenu du fichiers executable dans ces trois segments
     - 2.5 remet le CO à 0
     - 2.6 le processus est mis en mode "activable" (runnable) et donc apparait dans le tourniquet
  3. Le père repasse en mode utilisateur

Plan

La segmentation

Tables des segments en Unix

- Les tables de segments sont référencées dans une table "système" : chaque processus dispose d'entrées dans cette table
- Les tables donnent les adresses de relocation
- Tab. de seg. globaux –> partage de segments : bibliothèques, mémoires partagées
- Tab. de seg. locaux –> espace d'adressage propre aux processus



La segmentation

Partage de mémoires utilisateur

- Le partage de mémoire entre plusieurs processus est le moyen le plus rapide d'échange de données.
- Pour pouvoir partager de la mémoire, quatre appels systèmes (sera vu en TD) :
  - `int shmget(key_t key, int size, int shmflg)` : construit le segment de mémoire partagée → rend l'identifiant du segment ayant la clé key.
  - `int shmctl()` : contrôle du segment ;
  - `shmat(nt shmid, char *shmadrr, int shmflg)` : attachement du segment `shmid` avec les droits spécifiés
  - `shmdt()` : dé-attachement d'un segment.

La segmentation

Cas d'Unix

- Les tables de segments sont référencées dans une table "système" : chaque processus dispose d'entrées dans cette table
- Les tables donnent les adresses de relocation
- Tab. de seg. globaux –> partage de segments : bibliothèques, mémoires partagées
- Tab. de seg. locaux –> espace d'adressage propre aux processus