Virtualisation (mémoire) : segmentation

Thibaud Martinez
thibaud.martinez@dauphine.psl.eu

Dauphine | PSL
UNIVERSITÉ PARIS

Cette présentation couvre les chapitres 13, 14, 15 et 16 de Operating Systems: Three Easy Pieces.

Les diapositives sont librement adaptées de diapositives de *Youjip Won (Hanyang University)*.

Qu'est-ce que la virtualisation de la mémoire ?

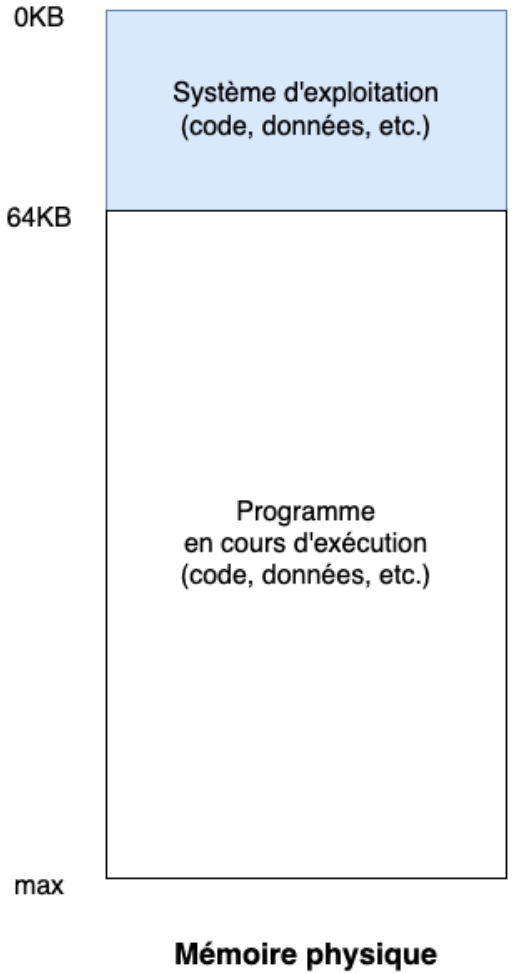
Le système d'exploitation fournit l'illusion au processus qu'il est le seul à utiliser la mémoire et qu'il peut l'utiliser en totalité.

Pourquoi virtualiser la mémoire ?

- Facilité d'utilisation pour la programmation.
- Efficacité de la mémoire en termes de temps et d'espace.
- Garantie d'isolation pour les processus et le système d'exploitation.
  → Protection contre les accès erronés depuis d'autres processus

Les premiers systèmes d'exploitation

Chargement d'un seul processus en mémoire.



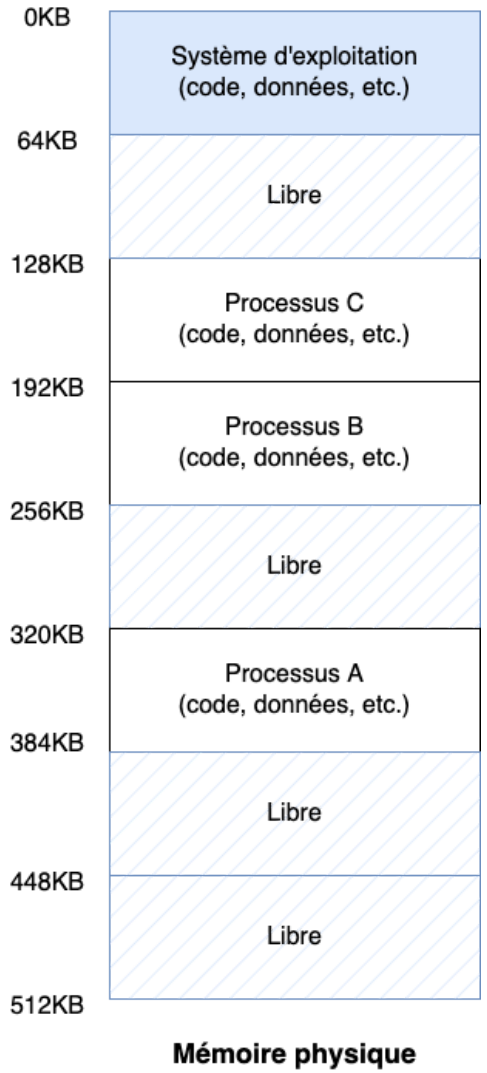
Multiprogrammation et *time sharing*

Chargement de plusieurs processus en mémoire.

- Exécuter un processus pendant une courte période.
- Passer d'un processus à l'autre en mémoire.

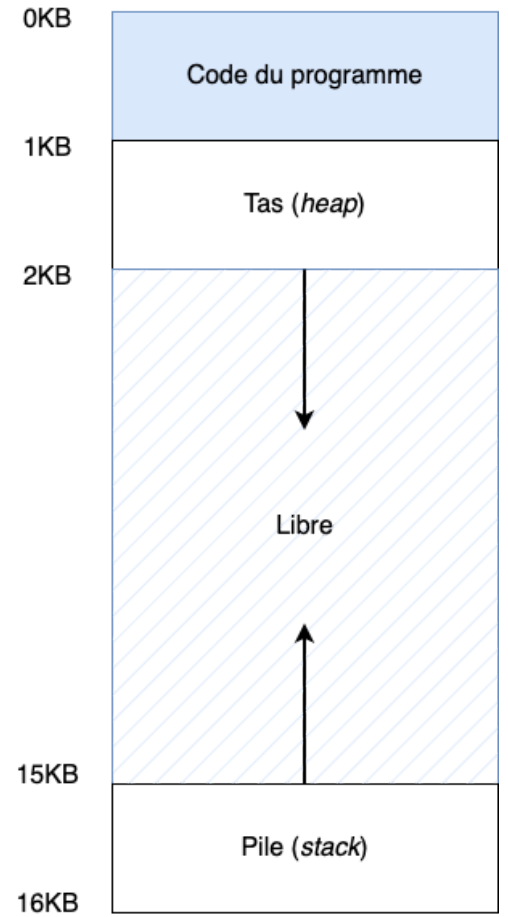
→ Augmenter l'utilisation et l'efficacité.

Problème de protection : **accès erronés à la mémoire par d'autres processus**.



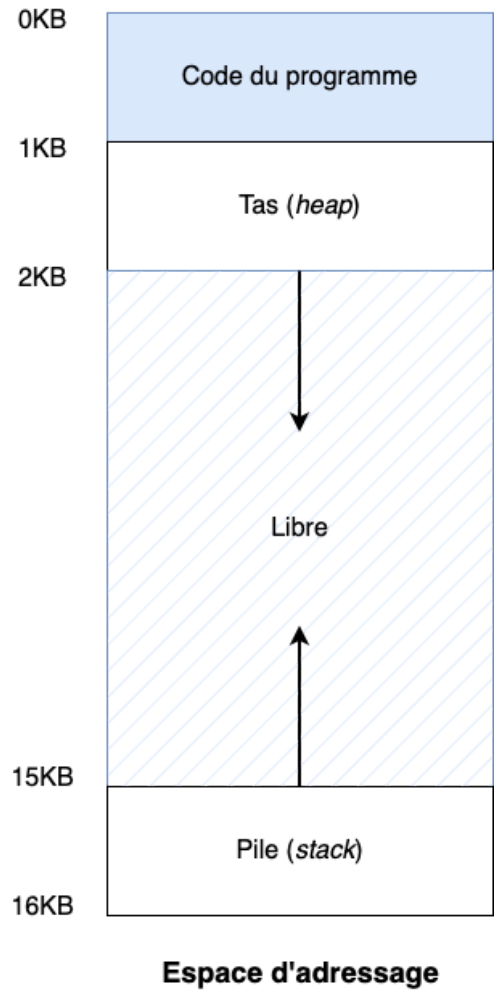
Espace d'adressage

- Le système d'exploitation crée une abstraction de la mémoire physique.
- Il s'agit d'une "vue" de la mémoire physique pour un processus.
- L'espace d'adressage contient tout ce qui concerne un processus en cours d'exécution : code du programme, du tas, de la pile, etc.



Espace d'adressage

- **Code**: où se trouvent les instructions.
- **Tas (*heap*)** : pour allouer dynamiquement de la mémoire.
  - `malloc` en langage C
- **Pile (*stack*)**
  - Stocke les adresses de retour ou les valeurs.
  - Contient les variables locales, les arguments des fonctions.



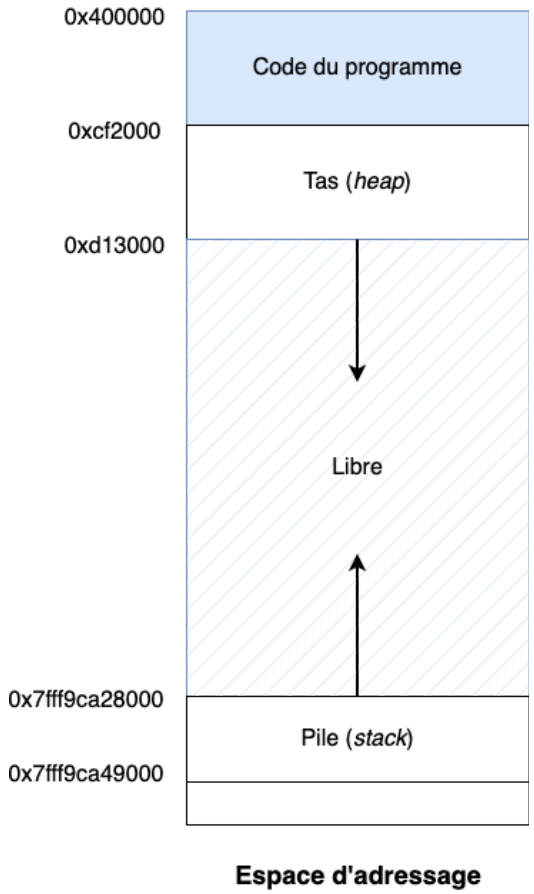
Organisation de l'espace d'adressage : exemple

```c
#include <stdio.h>
#include <stdlib.h>

int main(int argc, char *argv[]) {
    printf("location of code : %p\n", main);
    printf("location of heap : %p\n", malloc(100e6));
    int x = 3;
    printf("location of stack: %p\n", &x);
    return 0;
}
```

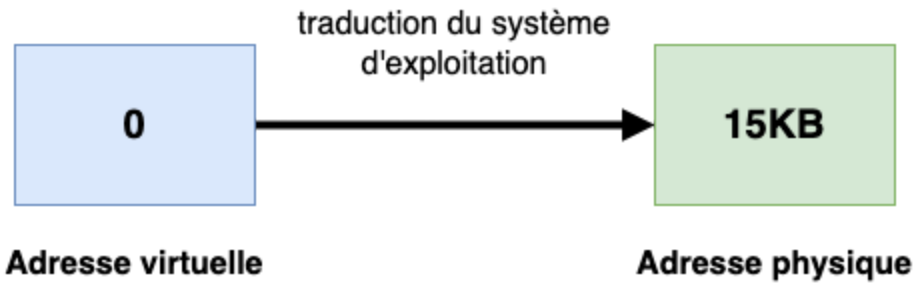
Le résultat sur une machine Linux 64 bits.

```
location of code : 0x40057d
location of heap : 0xcf2010
location of stack : 0x7fff9ca45fcc
```



Adresse virtuelle

- Chaque adresse d'un programme en cours d'exécution est **virtuelle**.
- Le processus a l'impression que son espace d'adressage débute à l'adresse mémoire 0.
- Le système d'exploitation traduit l'**adresse virtuelle** en **adresse physique**.



Note : toutes les adresses mémoires affichées par un processus en mode utilisateur sont des adresses virtuelles.

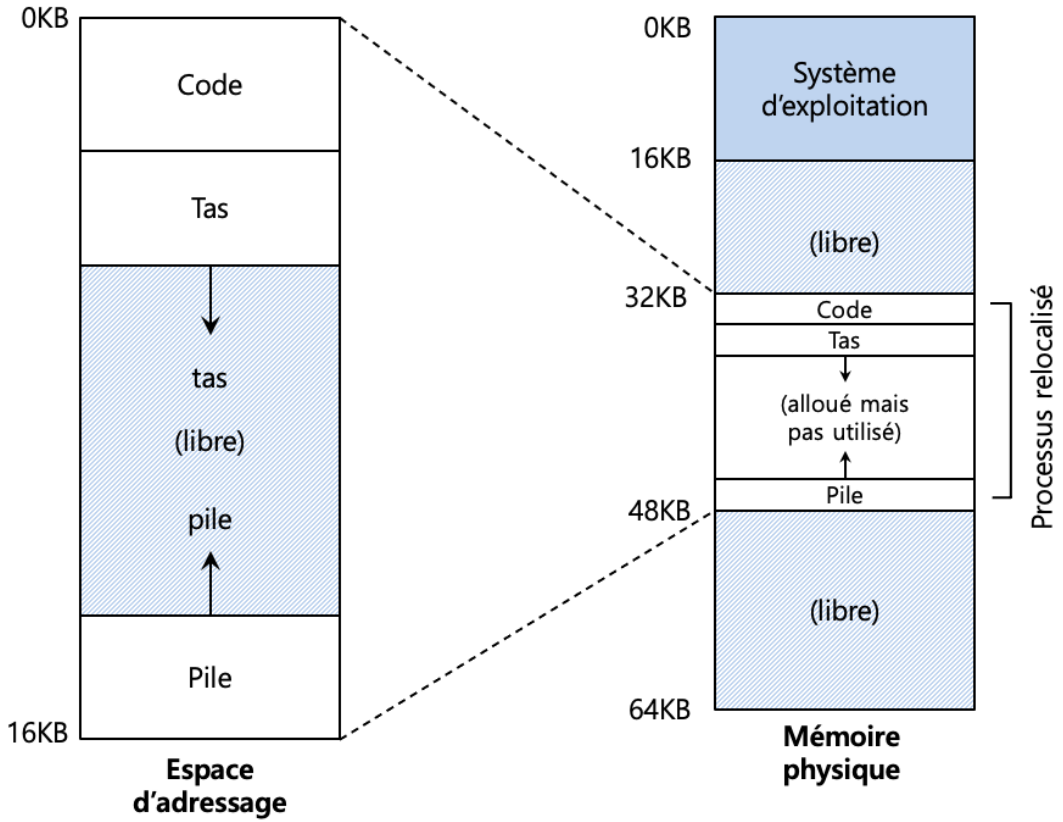
Comment virtualiser la mémoire avec efficacité et contrôle ?

- Mécanisme de bas niveau assuré par le matériel.
  Traduction d'adresse : le matériel transforme une adresse virtuelle en adresse physique. Les données sont en réalité stockées à une adresse physique.

- Intervention du système d'exploitation pour gérer la mémoire :
  - Quels espaces sont libres ?
  - Quels accès mémoires sont autorisés ?

Relocalisation de l'espace d'adressage

- Pour le processus l'espace d'adressage débute à l'adresse mémoire 0.
- Le système d'exploitation veut placer le processus à un autre endroit de la mémoire physique, et non à l'adresse 0.



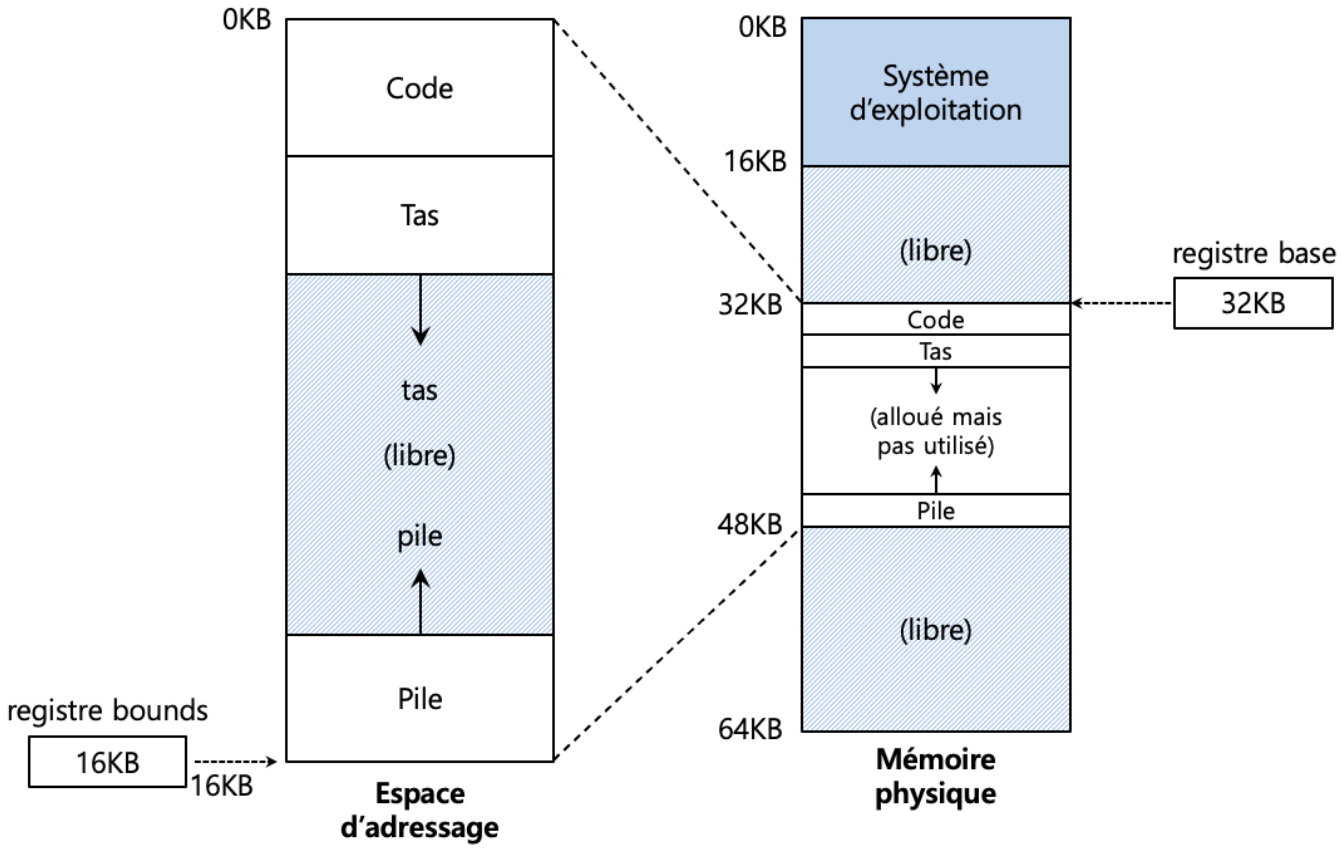
Relocalisation dynamique, registres *base* et *bounds*

Lorsqu'un programme commence à s'exécuter, le système d'exploitation décide de l'endroit de la mémoire physique où le processus doit être chargé.

- Les registres ***base*** et ***bounds*** se voient attribué une valeur.
- `adresse physique = adresse virtuelle + base`
- Les adresses virtuelles ne doivent pas dépasser la valeur `base + bounds`, sinon le processeur lève une exception.

Les registres *base* et *bounds* se trouvent dans une partie du processeur appelée la **memory management unit (MMU)**.

Registres *base* et *bounds*



Exemple de traduction d'adresses

- `base = 16KB`
- `bounds = taille de l'espace d'adressage = 4KB`

Adresse virtuelle		Adresse physique
0	→	16KB
1KB	→	17KB
3000B	→	19384B
4400B	→	Erreur (hors limites)

Limites de l'approche *base* et *bounds*

- Un gros morceau d'espace "libre" occupe la mémoire physique.
- Difficile de fonctionner lorsqu'un espace d'adressage ne tient pas dans la mémoire physique.

Gestion de la mémoire par le système d'exploitation

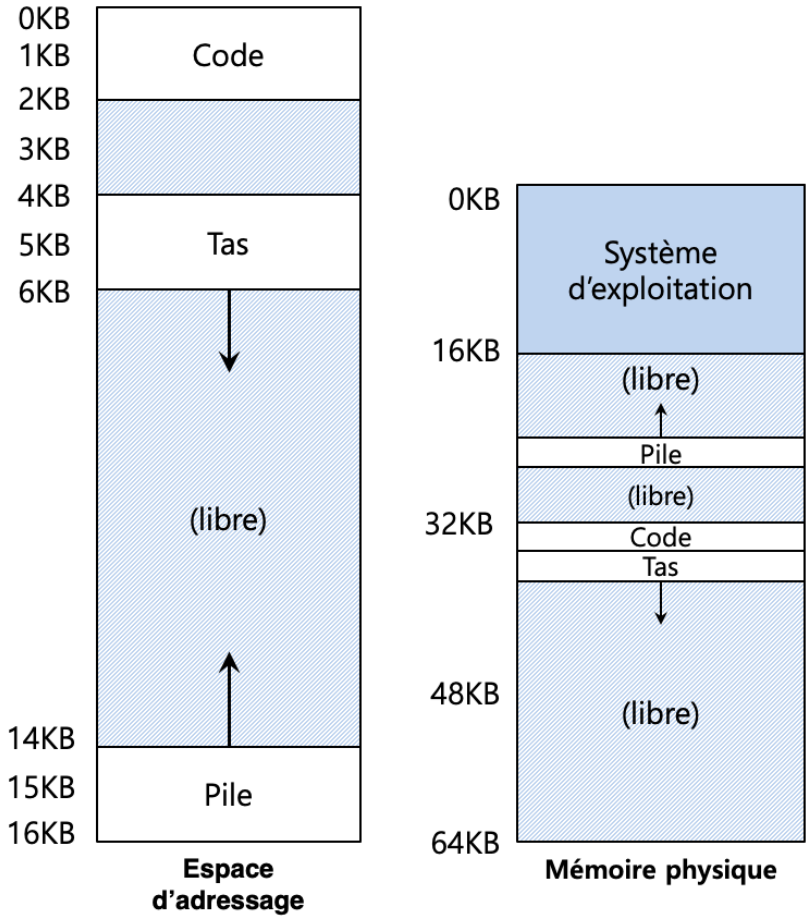
1. Lorsqu'un processus est créé le système d'exploitation doit **trouver un emplacement en mémoire pour son espace d'adressage**. Le système d'exploitation maintient une structure de données qui référence les emplacements disponibles (*free list*).
2. Lorsqu'un processus se termine, le système met à jour les emplacements disponibles.
3. Lorsqu'une **commutation de contexte** se produit, le système d'exploitation sauvegarde et restaure l'état des registres *base* and *bounds*.

Segmentation

La segmentation permet de dépasser les limites de l'approche *base* and *bounds*.

- Un segment est une portion **contiguë** de l'espace d'adressage d'une longueur donnée.
- Pour un processus donné, on a 3 segments différents : code, pile, tas.
- Chaque segment peut être placé dans une partie différente de la mémoire physique.
- Les registres *base* et *bounds* existent pour chaque segment. La MMU contient les registres des segments.

Placer un segment dans la mémoire physique



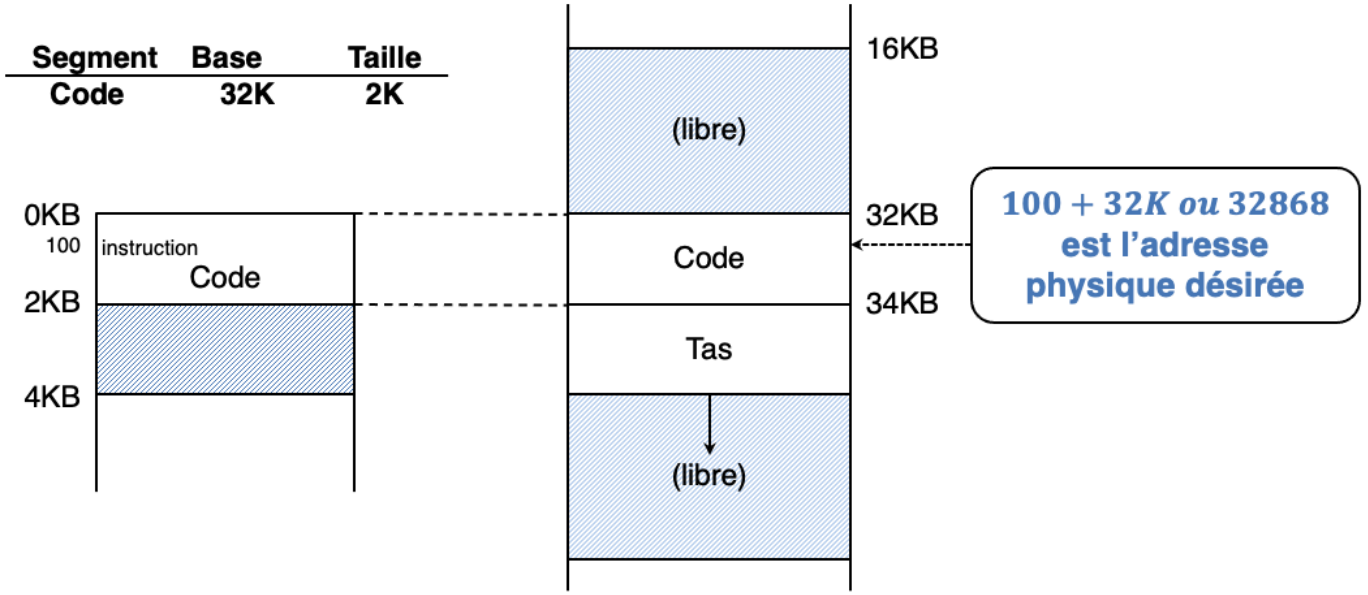
Segment	Base	Taille
Code	32K	2K
Tas	34K	2K
Pile	28K	2K

Traduction d'adresse pour le segment code

```
adresse physique = base + offset
```

L'*offset* est la position de l'adresse par rapport au début du segment.

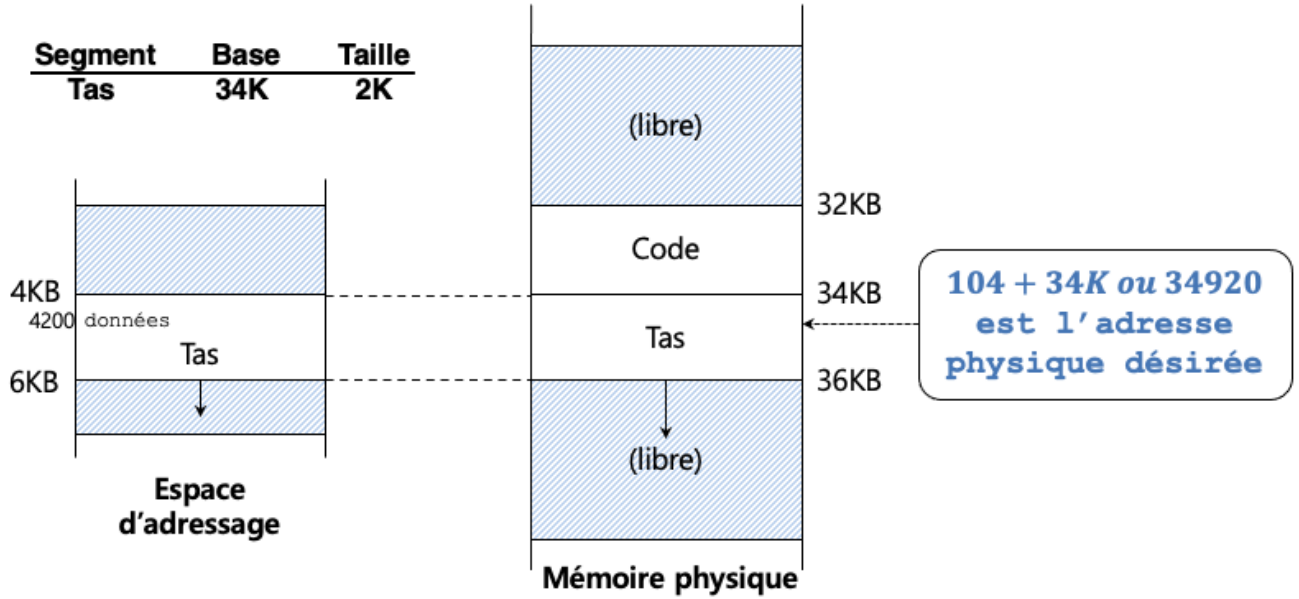
Pour le segment code, l'*offset* de l'adresse virtuelle 100 est 100 car le segment code **débute à l'adresse virtuelle 0 dans l'espace d'adressage.**



Traduction d'adresse pour le tas

⚠️ `adresse virtuelle + base` ne correspond pas à l'adresse physique correcte.

En effet, le segment tas commence à l'adresse virtuelle 4096 (4KB) dans l'espace d'adressage. Ainsi, l'*offset* de l'adresse virtuelle 4200 est `4200 - 4096 = 104`.



Erreur de segmentation (*segmentation fault*)

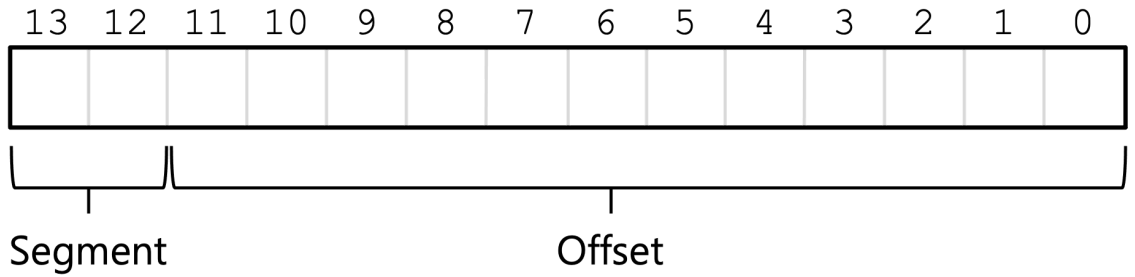
Si une adresse illégale, telle que 7 KB, qui se trouve au-delà de la fin du tas, est référencée, le système d'exploitation lève **une erreur de segmentation (*segmentation fault*)**.

Le matériel détecte que l'adresse est hors limites.

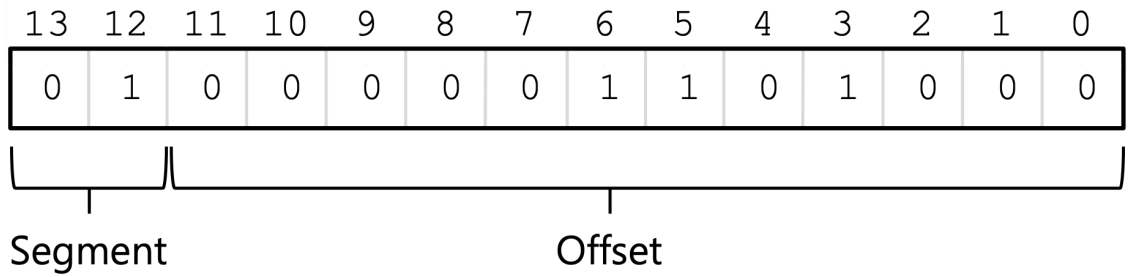
Comment faire référence à un segment ?

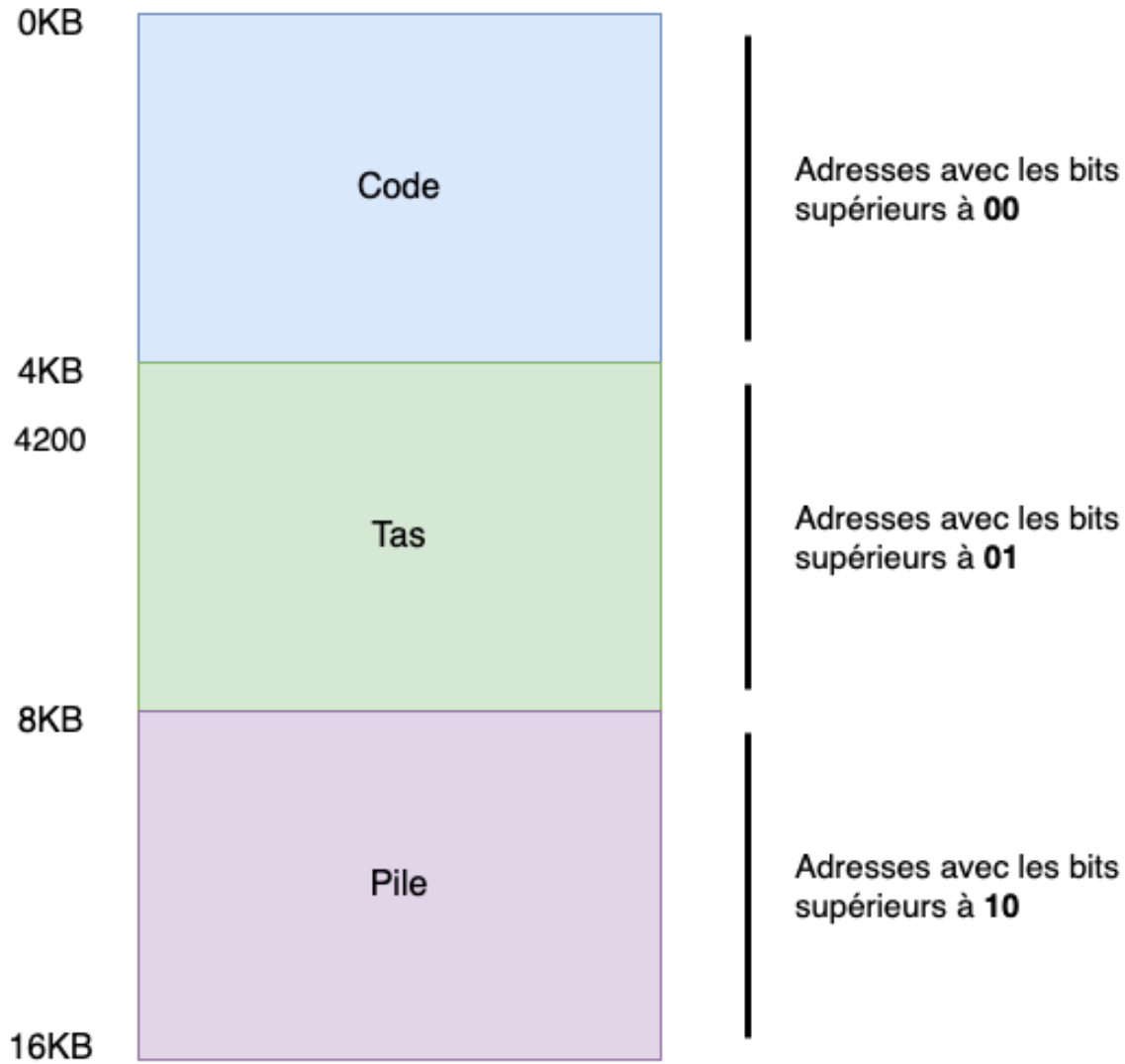
Le matériel utilise des registres de segments pendant la traduction. Comment sait-il quel est l'*offset* dans un segment et à quel segment une adresse se réfère ?

Solution: découper l'espace d'adressage en segments basés sur les deux bits supérieurs de l'adresse virtuelle.



Exemple : adresse virtuelle 4200 (01000001101000)





Espace d'adressage

Traduction d'adresse pour la pile

- La pile croît à l'envers.
- Un support matériel supplémentaire est nécessaire.
- Le matériel vérifie le sens de croissance du segment: 1 : sens positif, 0 : sens négatif

Segment	Base	Taille	Direction
Code	32KB	2KB	1
Tas	34KB	2KB	1
Pile	28KB	2KB	0

Registres de segments

Partage de segments

Un segment peut être partagé entre espaces d'adressage.
→ Le partage de code est encore utilisé actuellement.

Un support matériel supplémentaire est nécessaire sous la forme de **bits de protection** : quelques bits supplémentaires par segment indiquent les autorisations de lecture, d'écriture et d'exécution.

Segment	Base	Taille	Direction	Protection
Code	32KB	2KB	1	Read-Execute
Tas	34KB	2KB	1	Read-Write
Pile	28KB	2KB	0	Read-Write

Registres de segments

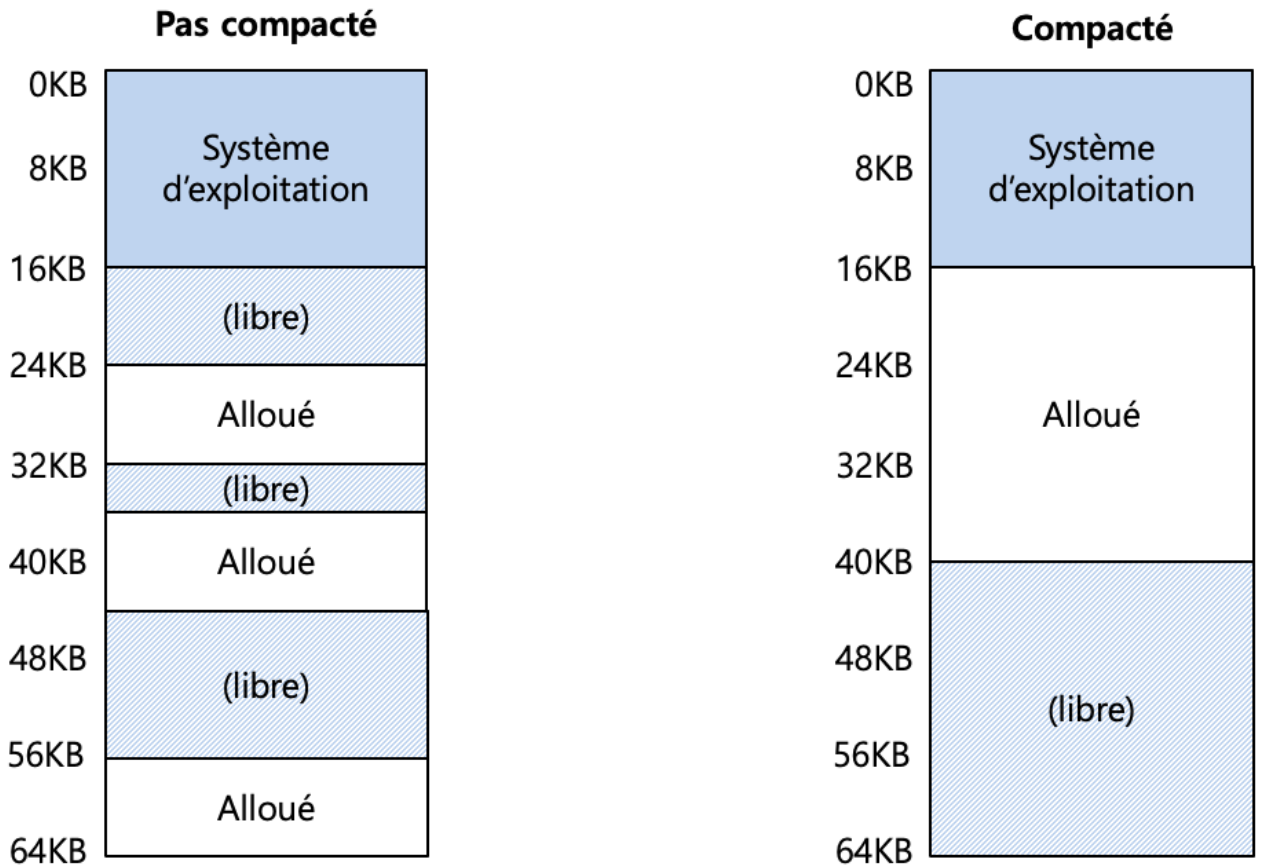
Fragmentation et compactage

Fragmentation externe : trous d'espace libre dans la mémoire physique qui rendent difficile l'allocation de nouveaux segments.

- Exemple : Il y a 24 KB d'espace libre, mais pas dans un segment contigu. Le système d'exploitation ne peut pas satisfaire la demande de 20 KB.

Compactage : réarrangement des segments existants dans la mémoire physique.

- Le compactage est coûteux. Il faut arrêtez le processus en cours, copier les données quelque part, et modifier la valeur du registre de segment.


Pas compacté
0KB
8KB
16KB
24KB
32KB
40KB
48KB
56KB
64KB
Système d'exploitation
(libre)
Alloué
(libre)
Alloué
(libre)
Alloué
Compacté
0KB
8KB
16KB
24KB
32KB
40KB
48KB
56KB
64KB
Système d'exploitation
Alloué
(libre)