MI01 – Automne 2011

Systèmes à microprocesseurs : gestion de la mémoire

Stéphane Bonnet

Poste : 52 56
Courriel : stephane.bonnet@utc.fr

Plan

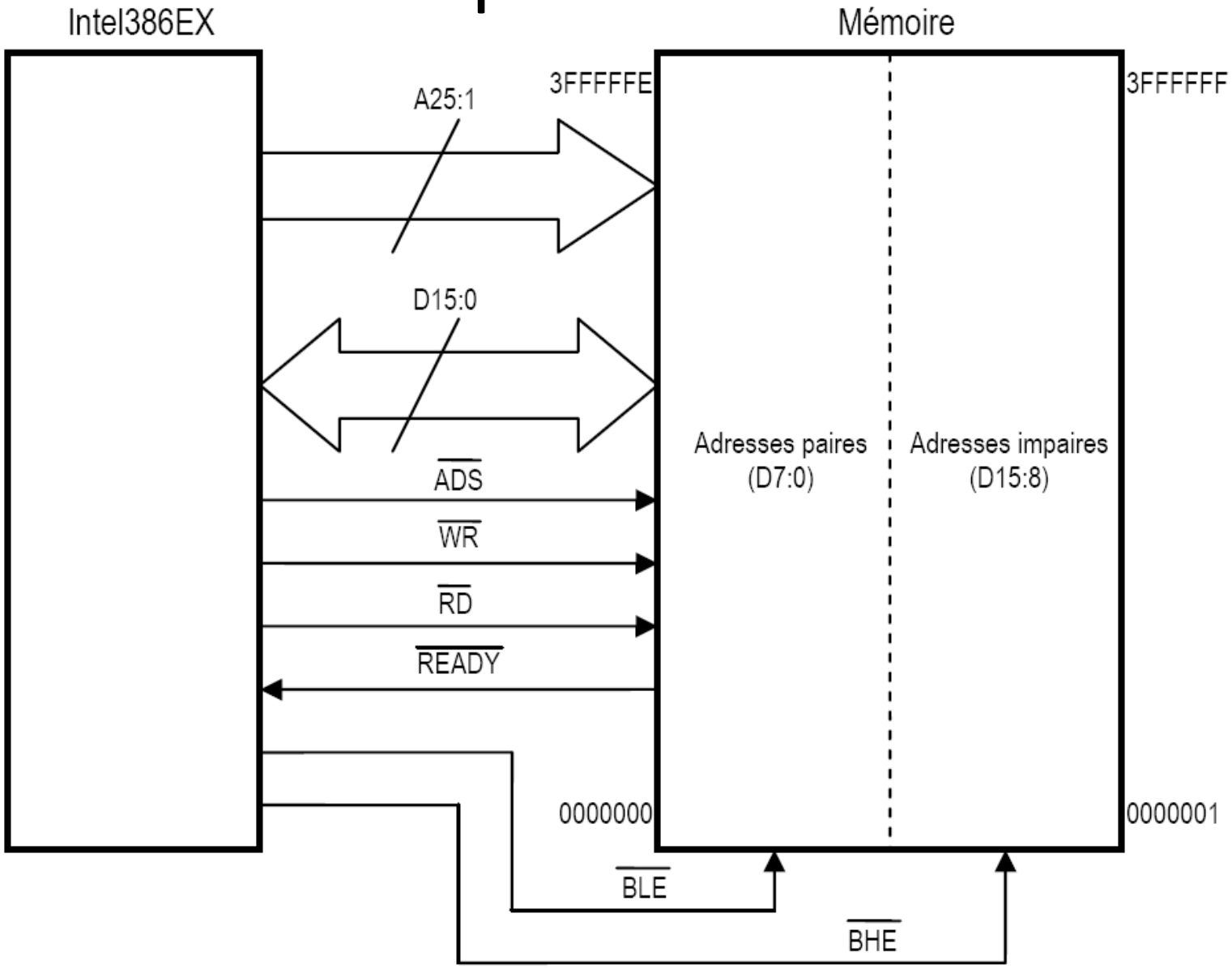
- Mémoire physique
- Gestion de la mémoire
- Mémoire virtuelle
  - Segmentation
  - Pagination

MÉMOIRE PHYSIQUE

Mémoire physique

- Le processeur est connecté à la mémoire au travers de **bus**
- Permettent la communication mémoire ⇔ processeur
  - Bus d'adresses
  - Bus de données
  - Signaux de contrôle et de synchronisation
- Taille du bus d'adresse impose la valeur de la plus grande adresse
- Taille du bus de données impose la quantité de données échangeables en une seule fois

Exemple : i386EX

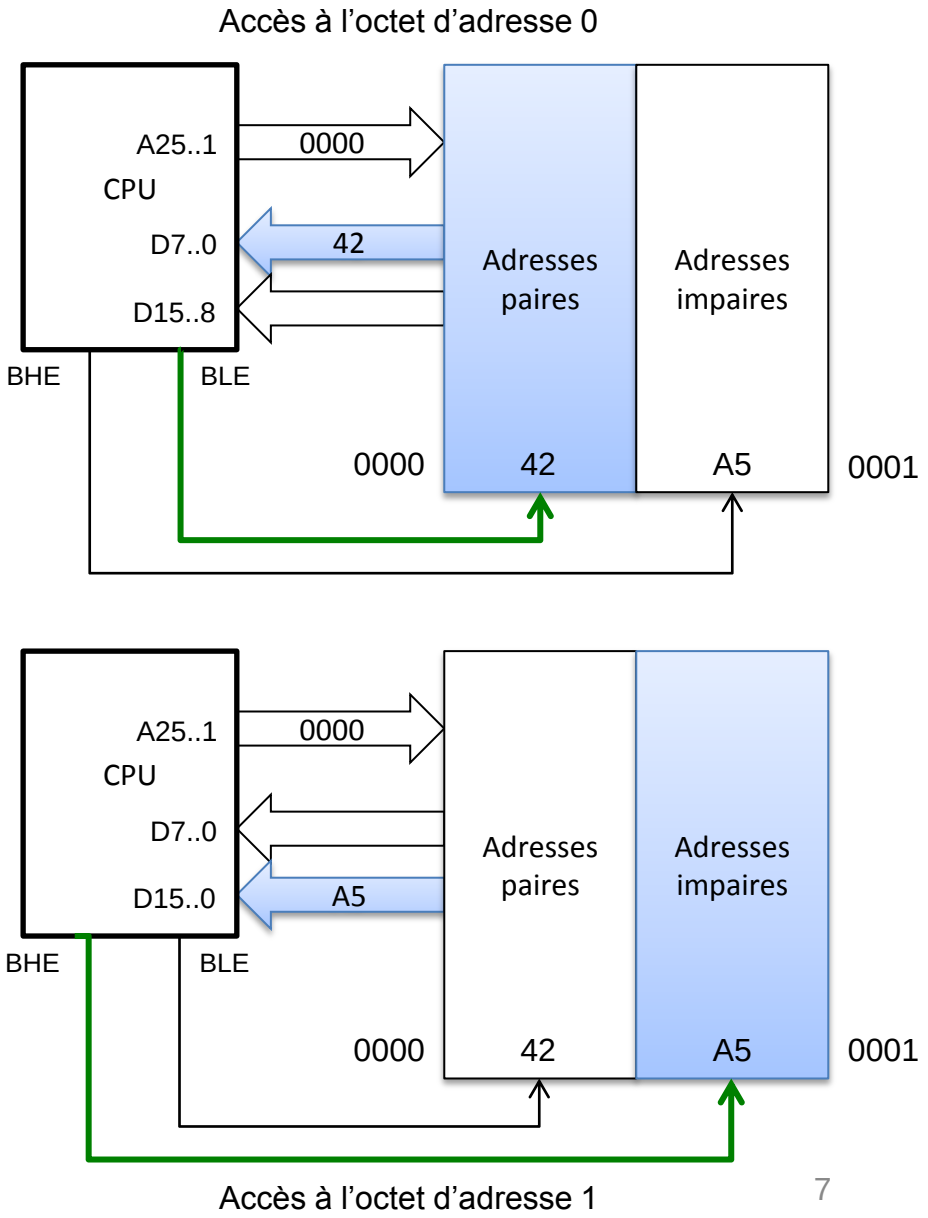


Exemple : i386EX

- Version « embarquée » du 80386
  - Processeur 32 bits
- Bus d’adresse 26 bits
  - 32 bits en interne
- Bus de données 16 bits
  - 32 bits en interne

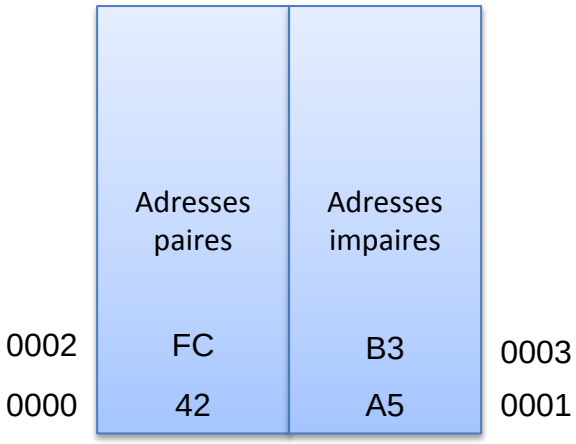
Tranferts : sélection des bus

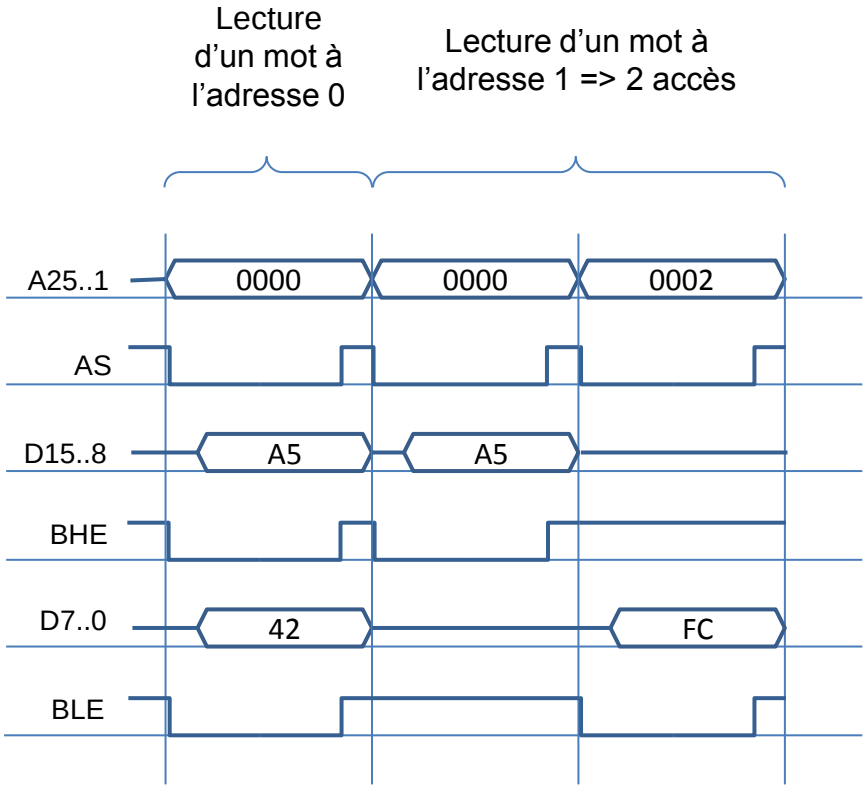
- Un signal est associé à chaque « tranche » de 8 bits du bus de données
  - D7..0 est valide si BLE (Bus Low Enable) est actif
  - D15..8 est valide si BHE (Bus High Enable) est actif
- Le bit d'adresse A0 est représenté par le couple BLE/BHE
- Généralisation à des bus plus larges (ex. 64 bits sur Pentium et plus)



Transferts : alignement

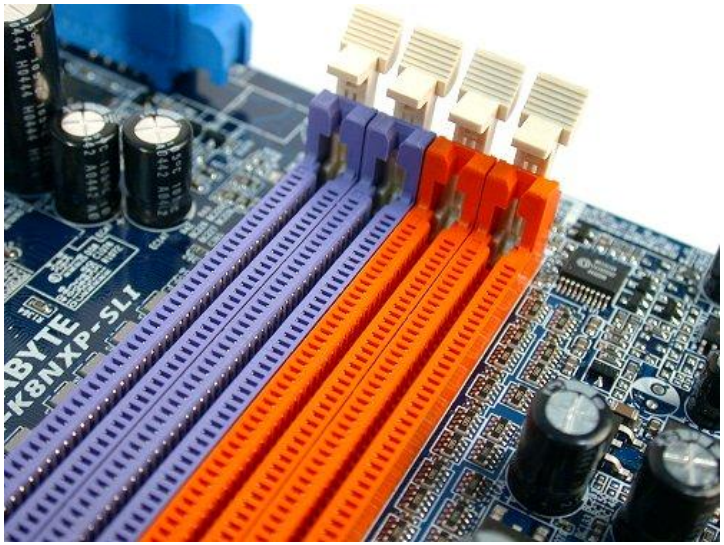
- Le processeur peut nécessiter plusieurs cycles pour lire une donnée unique
  - Dans ce cas, on dit que la donnée est non alignée
- **Une donnée est alignée si son adresse est un multiple de sa taille**

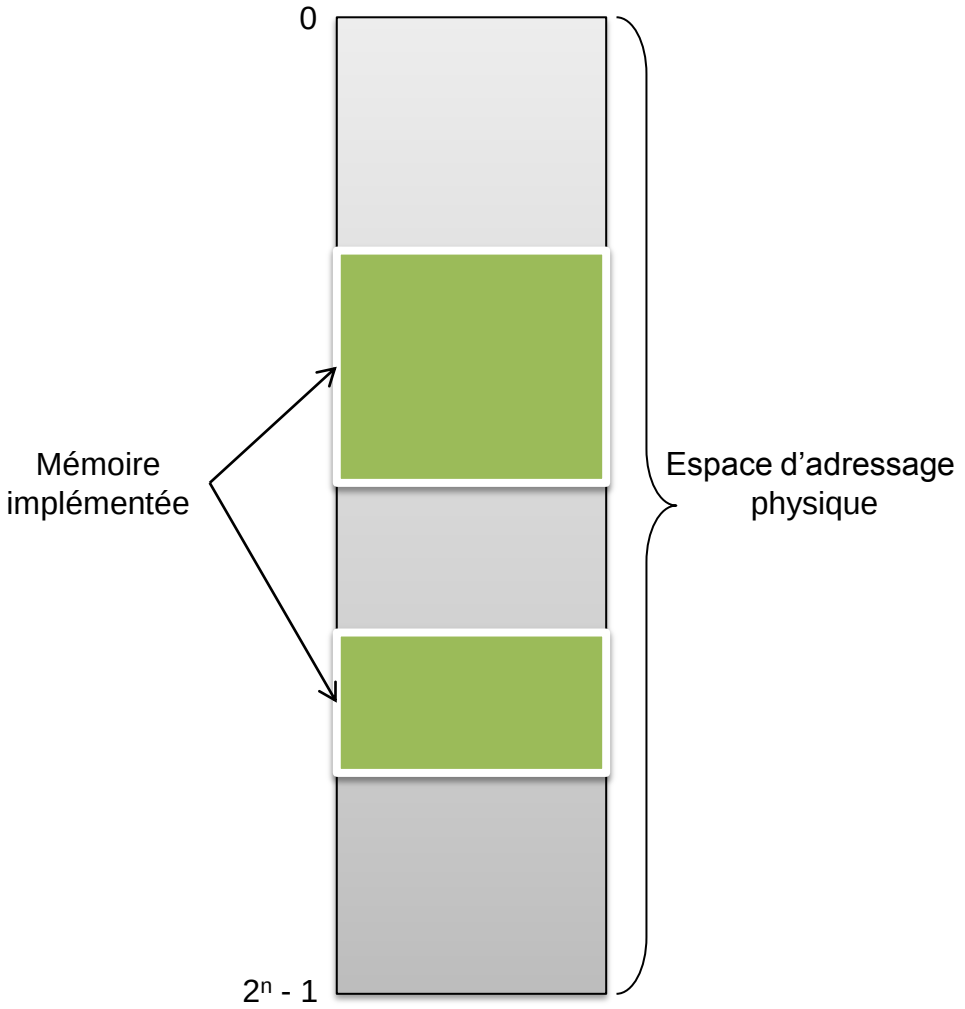




Espace d’adressage physique

- La mémoire implémentée n’occupe pas nécessairement tout l’espace d’adressage physique





GESTION DE LA MÉMOIRE

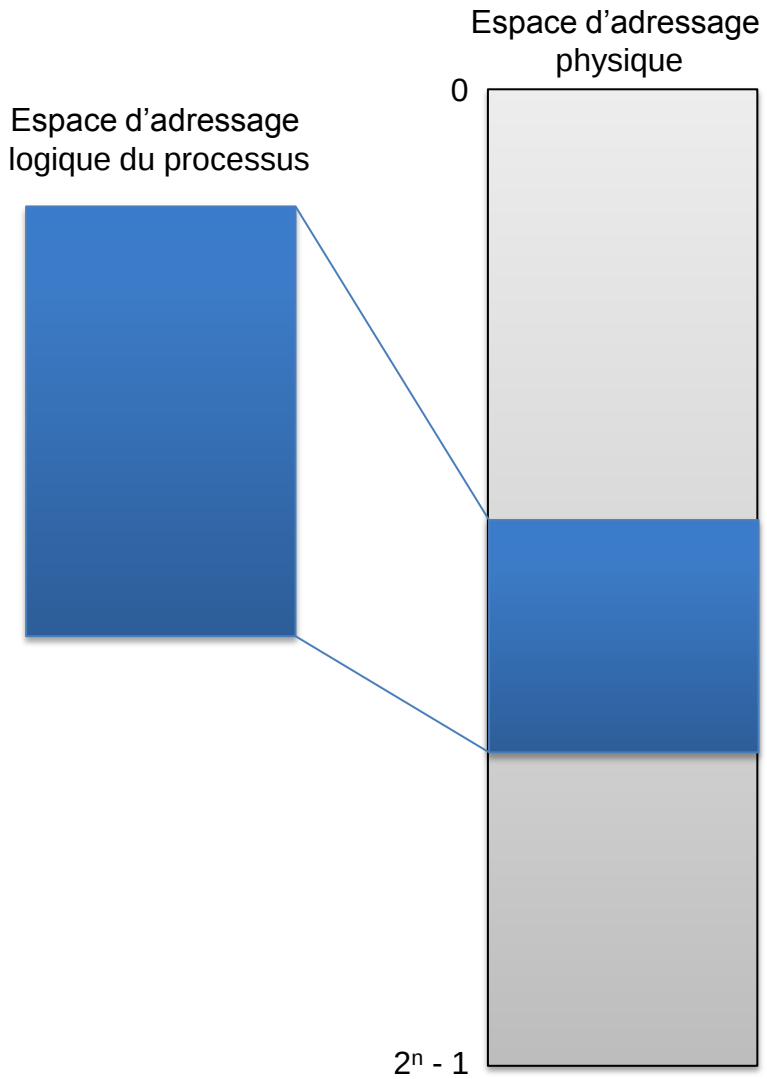
Quelques définitions utiles

- Un **programme** est une image exécutable stockée dans une mémoire secondaire

- Un **processus** est la matérialisation d'un programme et de ses données quand ils sont chargés en mémoire principale pendant l'exécution du programme

Programme et mémoire

- Un **processus** est un programme et ses données en cours d'exécution
- Quand un bloc de mémoire principale est alloué à un processus, **l'espace d'adressage logique** est associé à l'ensemble des adresses physiques correspondantes

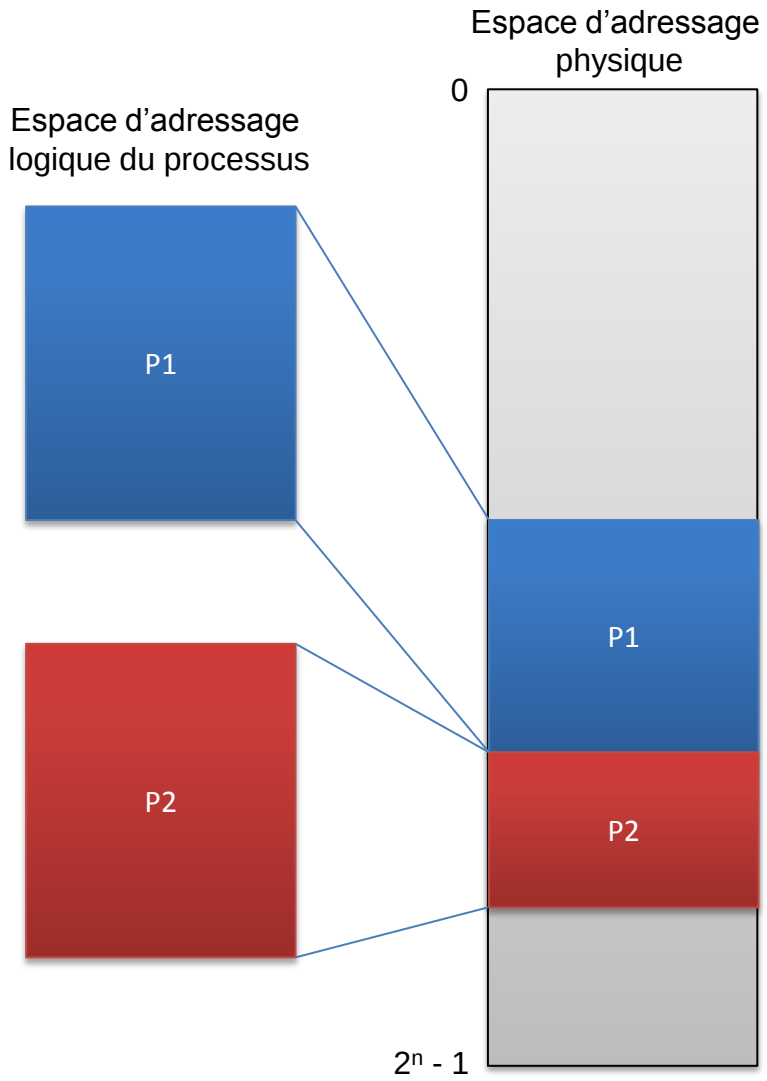


Où est le processus en mémoire ?

- Situation la plus simple : les instructions référencent les données (et les autres instructions…) au moyen de leur adresse physique.
- On parle alors d'adressage **réel**

Cas de plusieurs processus…

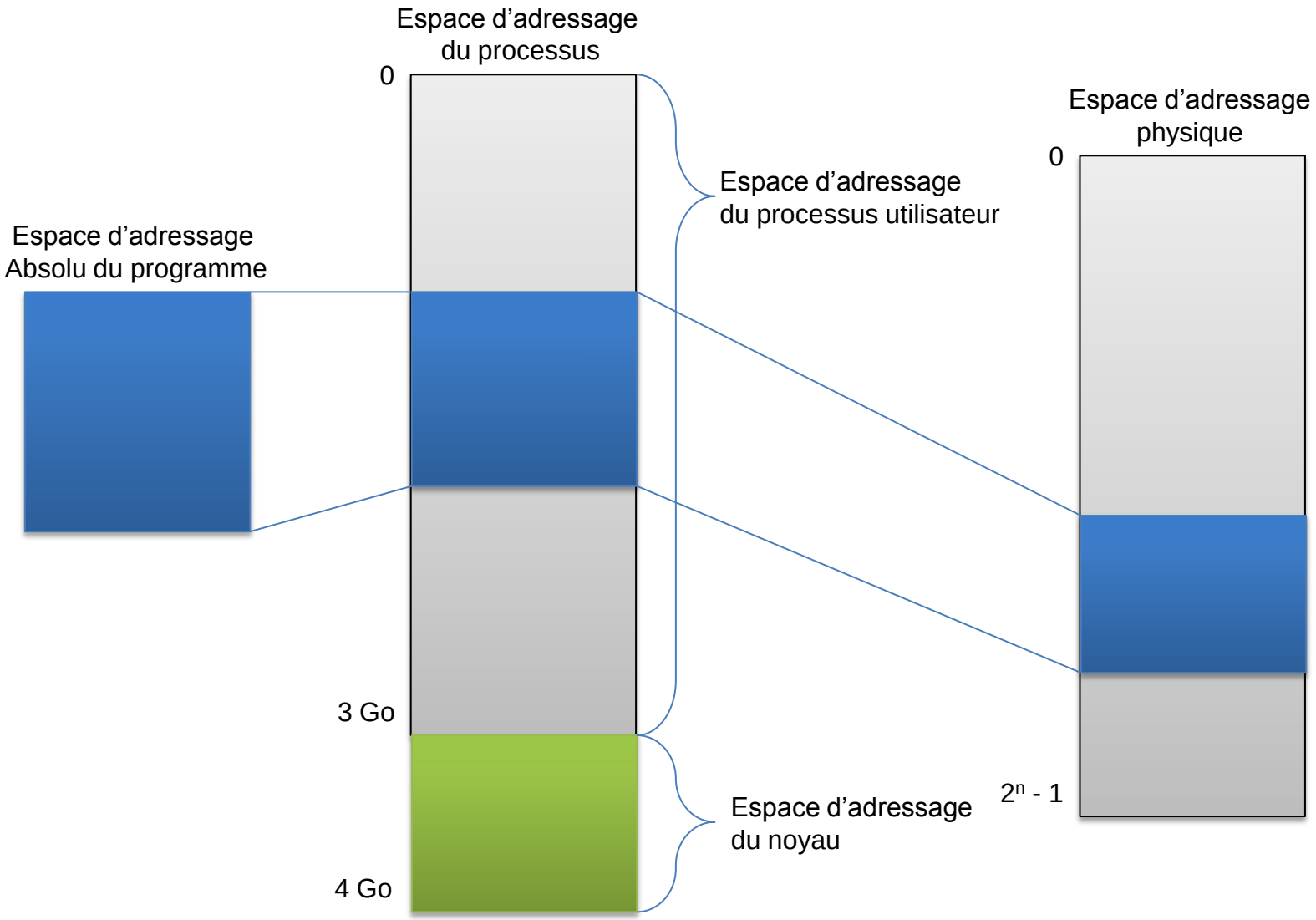
- En adressage réel, il faut que les adresses logiques des processus soient différentes
- Chaque programme doit connaître à l'avance sa place définitive en mémoire
- **Allocation statique** de la mémoire, implémenté par le programmeur et le compilateur => complexe



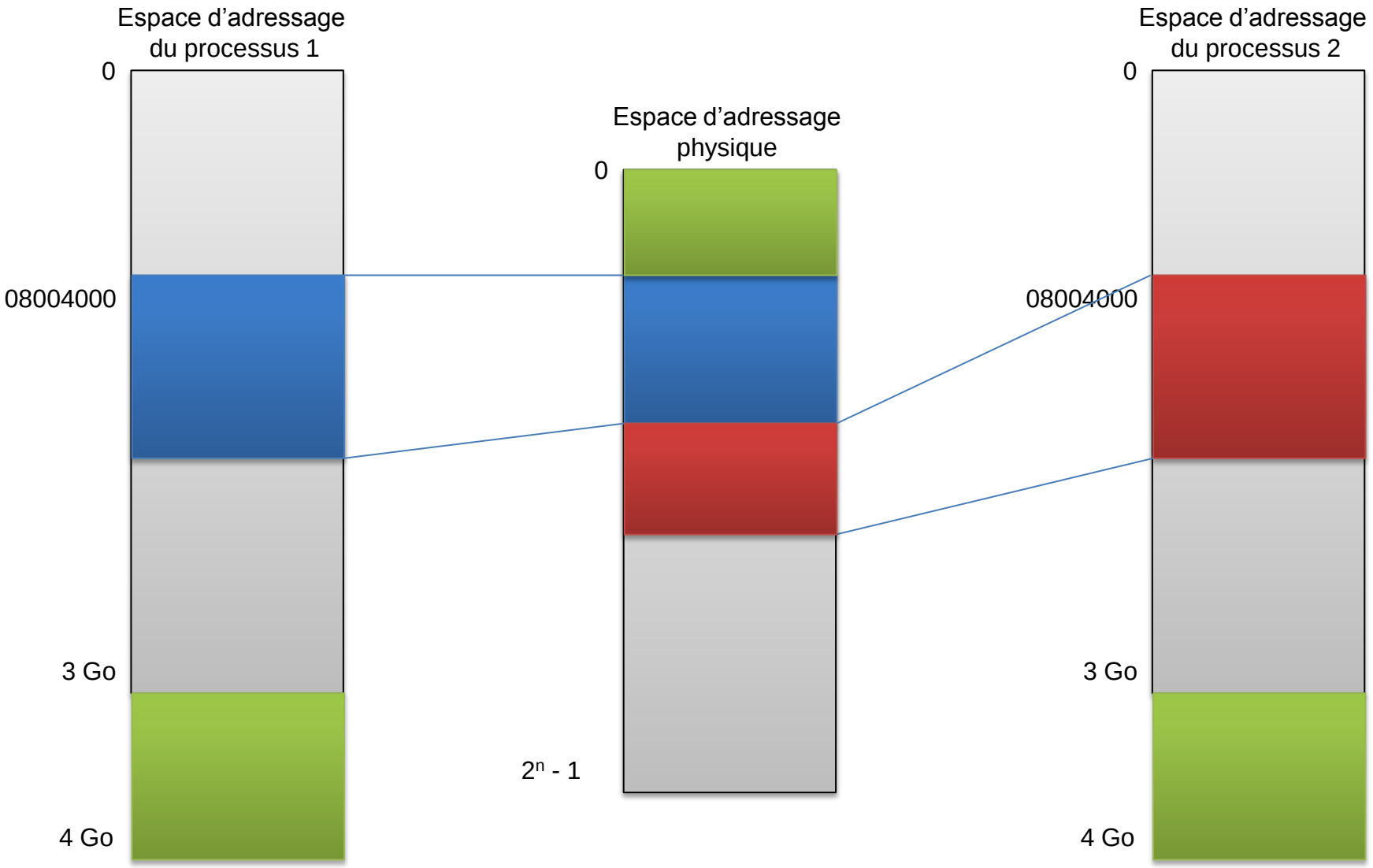
Allocation moderne

- En général chaque processus est créé avec un espace d'adressage vide très étendu (par exemple 4 Go sur IA32).
- Dans les OS comme Linux ou Windows, cet espace d'adressage est divisé en deux parties, typiquement :
  - 3 Go réservés au processus utilisateur
  - 1 Go réservés au système d'exploitation (noyau de l'OS).

Allocation moderne



Cas de plusieurs processus…



Cas de plusieurs processus

- Il n'est pas question de modifier les adresses utilisées par l'éditeur de lien à chaque programme pour que l'espace d'adressage absolu soit différent => la structure logique de l'espace d'adressage est identique pour tous les processus
- Il faut trouver un moyen de faire en sorte que des **adresses logiques identiques** dans chaque processus soient associées à des **adresses physiques différentes**

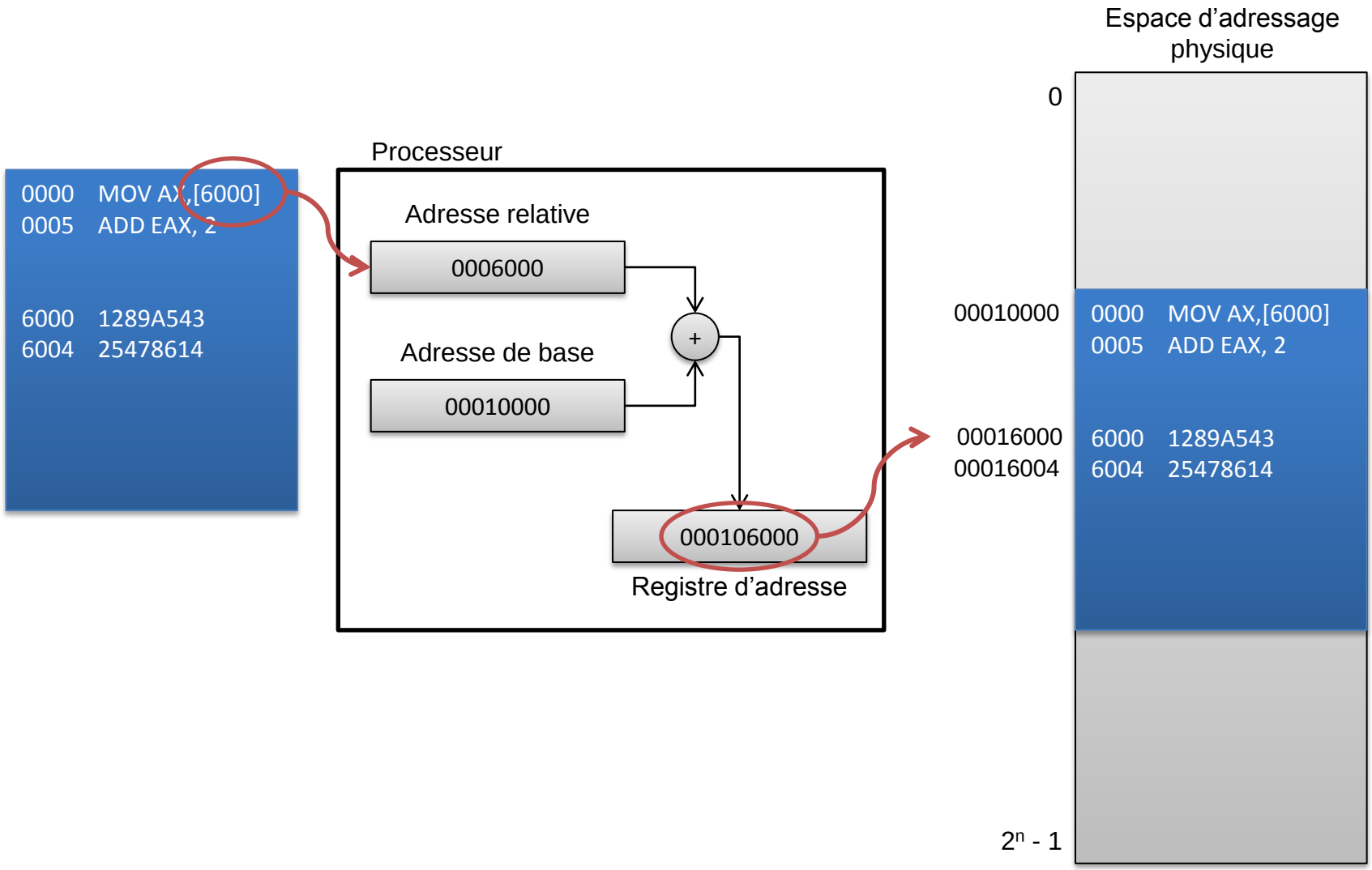
Gestion de la mémoire

- Réalisée par le système d'exploitation (gestionnaire de mémoire)
- Son rôle est de garder une trace des blocs de mémoire principale utilisés et libres
- Il assure en général :
  - L'abstraction de la mémoire principale pour que chaque processus ait l'illusion de disposer d'un grand espace d'adressage contigu.
  - L'allocation de blocs de la mémoire principale à à chaque processus
  - L'isolation des processus entre eux (chaque processus est assuré d'avoir l'usage exclusif de la mémoire qui lui est allouée
- Il est supporté dans ces tâches par les mécanismes de gestion de la mémoire du processeur

Relogement dynamique

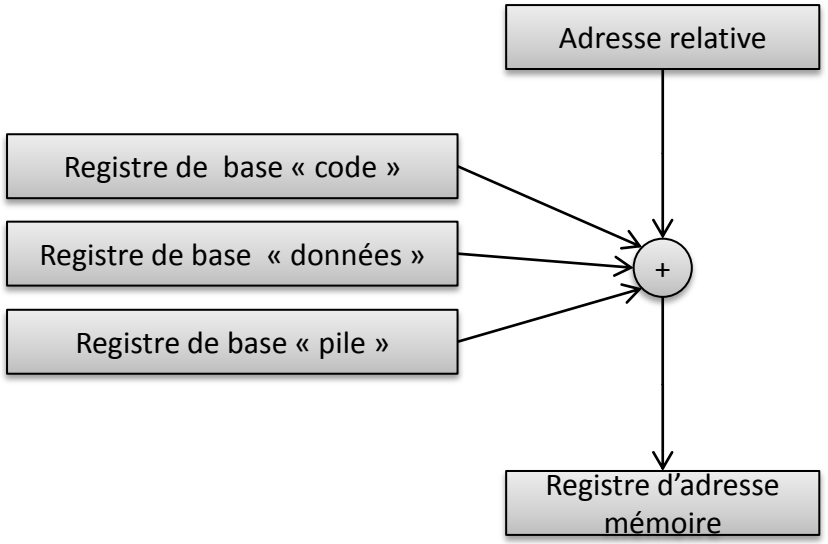
- Permettre au gestionnaire de mémoire de charger un processus n'importe où en mémoire, sans avoir à modifier le programme
- Les adresses absolues de tous les programmes sont les mêmes
- Les adresses des instructions, des données, de la pile sont relatives au début de l'espace d'adressage du processus
- Elles sont transformées en adresses physiques par le processeur « à la volée », au cours de l'exécution du processus => relogement dynamique

Relogement dynamique



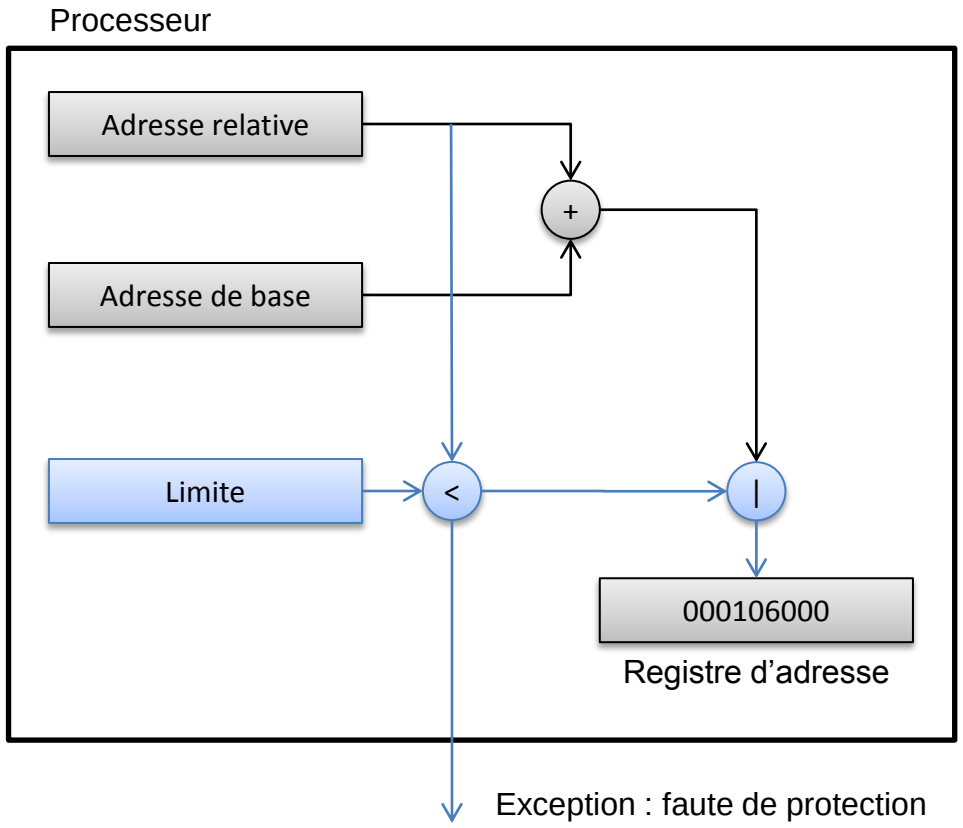
Registres de relogement multiples

- Une adresse de base différente pour chaque partie du processus :
  - Code
  - Données
  - Pile



Vérification des accès (protection)

- Ajout d'une vérification des bornes de l'adresse relative par rapport à une limite maximale
- Protection efficace de la mémoire (isolation des processus)



MÉMOIRE VIRTUELLE

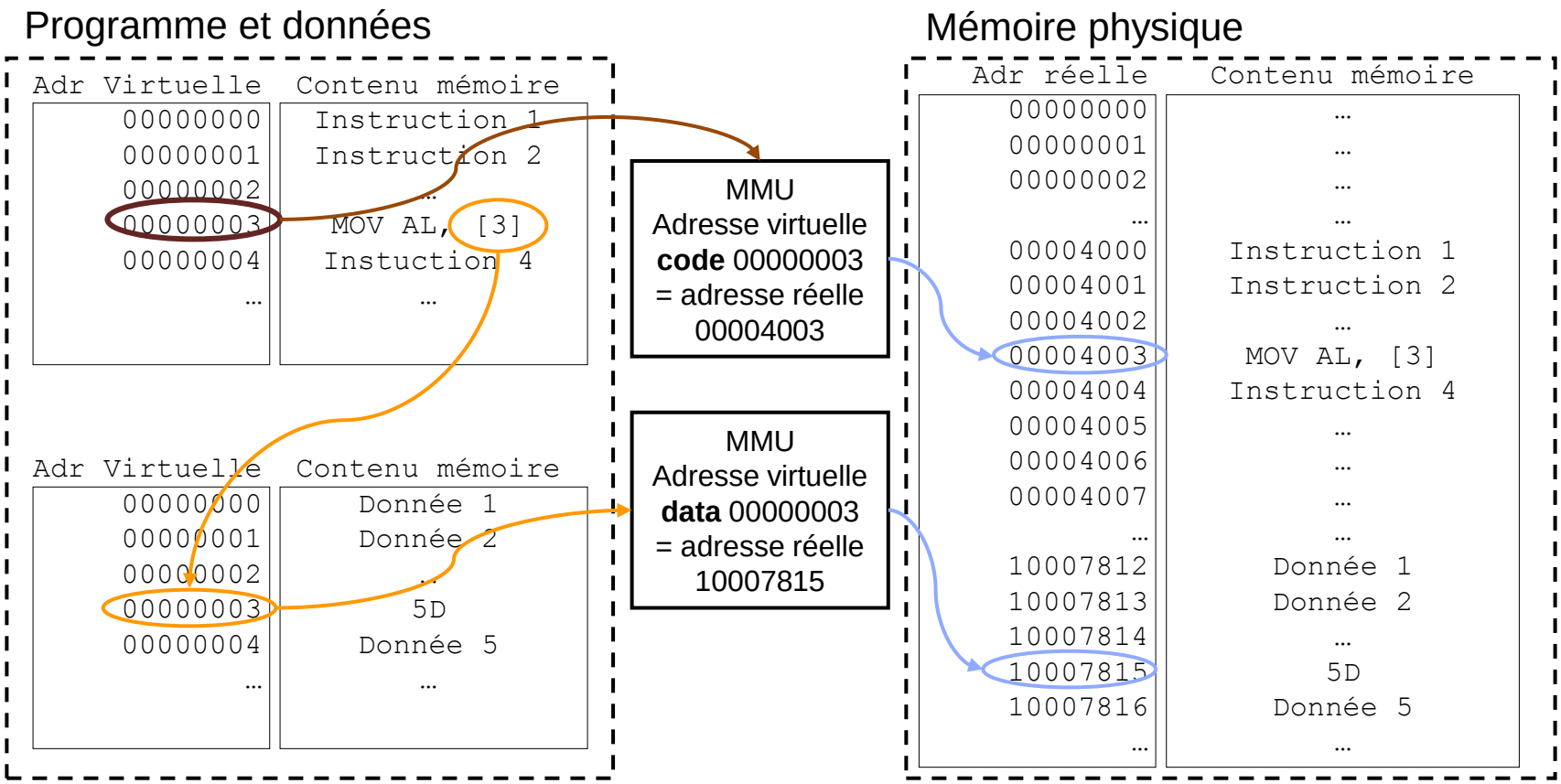
Mémoire virtuelle

- Mémoire virtuelle (VM)
  - Concept abstrait ≠ de la mémoire physique => c'est une abstraction de la mémoire
  - Formée des espaces d'adressage logiques (ou **virtuels**) de tous les processus
- Espace d'adressage virtuel d'un processus
  - Ensemble des adresses virtuelles visibles de ce processus
  - Peut être identique pour tous les processus
- Gérée par le gestionnaire de mémoire avec le soutien du matériel

Mémoire virtuelle

- Adresse virtuelle (ou adresse logique)
  - adresse du point de vue **du processus ≠** adresse physique
- Translation d'adresse
  - conversion adresses virtuelles en adresses physiques. Rôle d'une unité spécifique du processeur : l'unité de gestion mémoire (***MMU, Memory Management Unit***)

Mémoire virtuelle

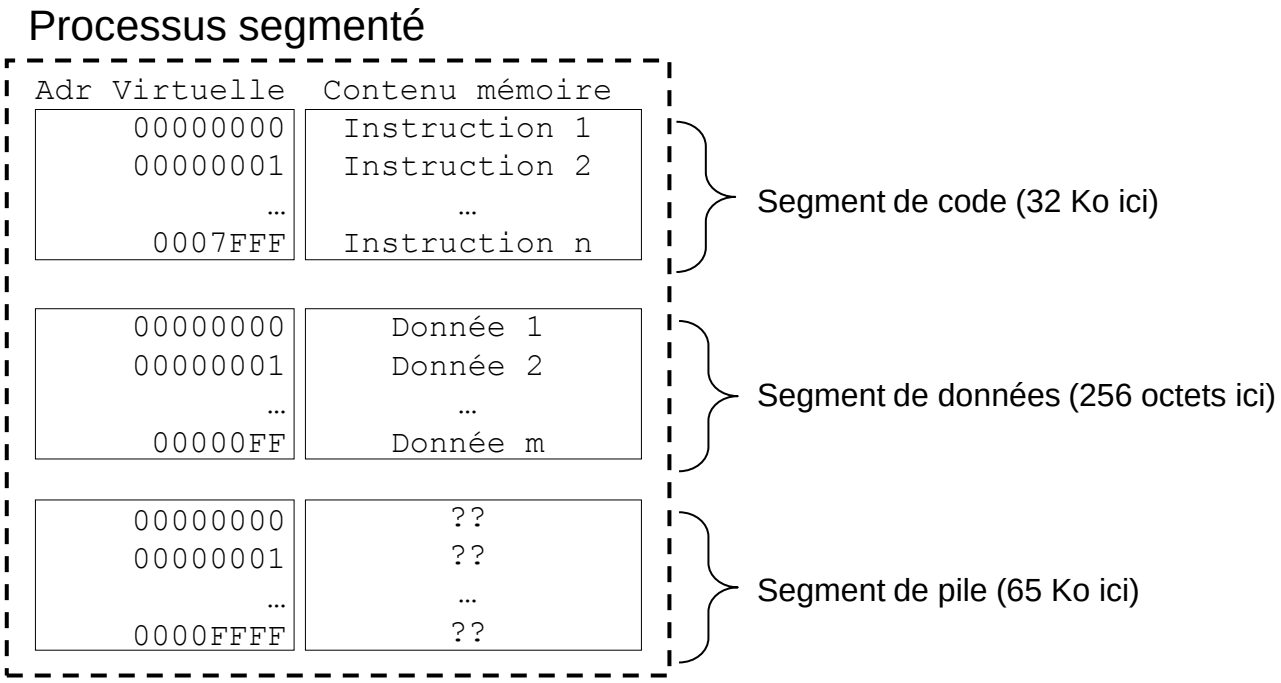


Mémoire virtuelle

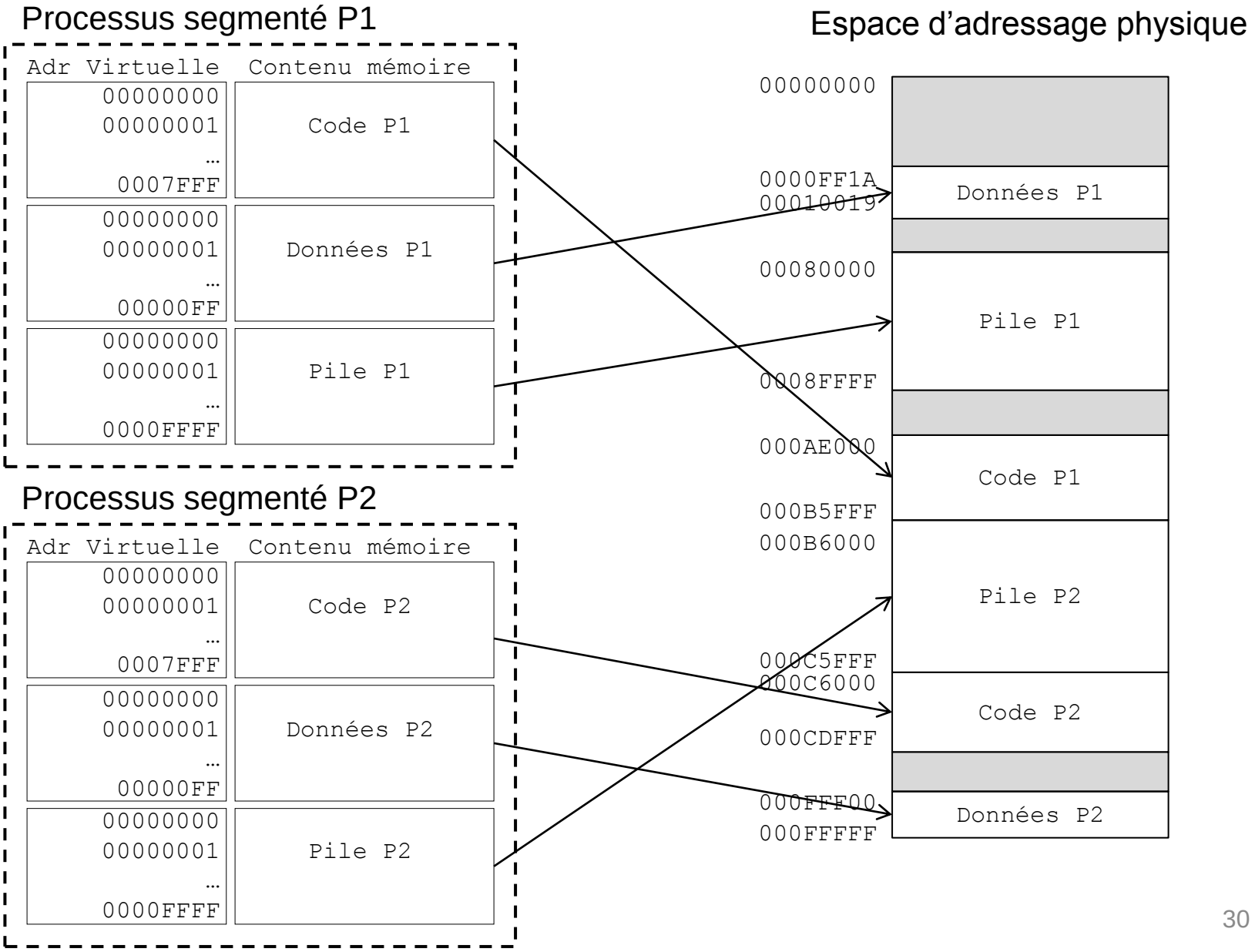
- Peut être implémentée par deux mécanismes
  - Segmentation
  - Pagination

Segmentation mémoire

- Chaque processus est divisé en plusieurs segments :
  - Segment de code qui contient les instructions
  - Segment de données
  - Segment de pile



Segmentation mémoire



Segmentation mémoire

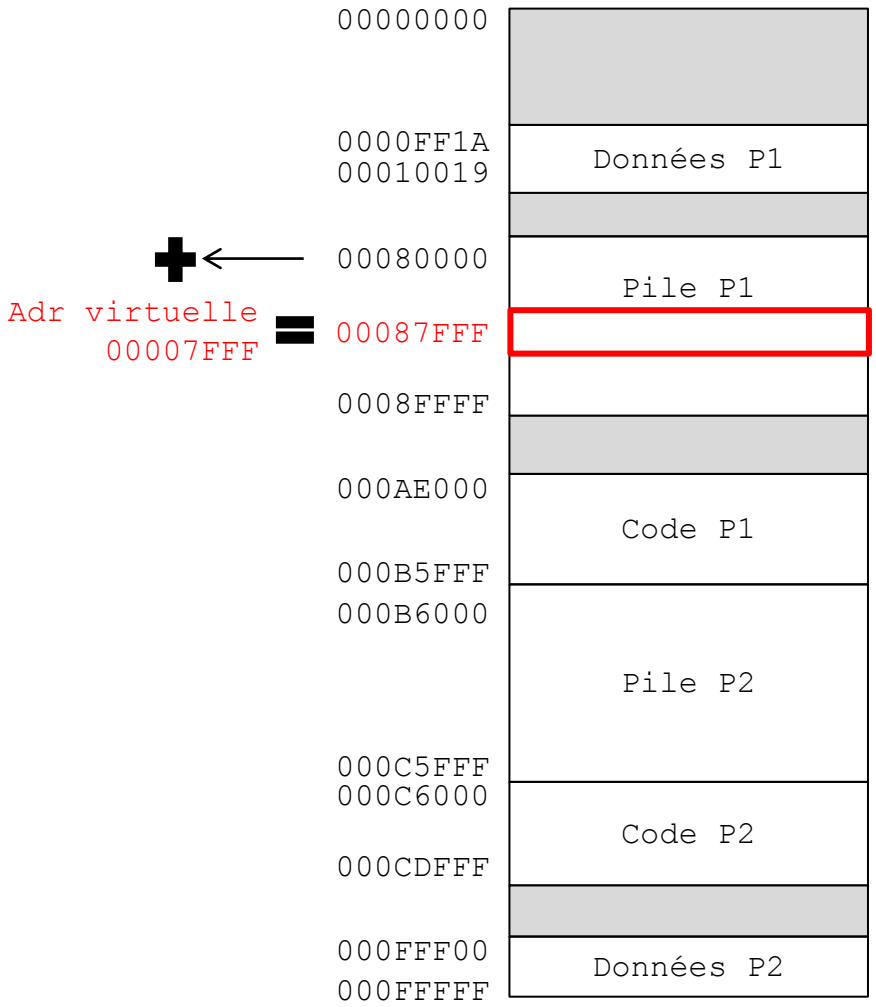
- Translation d'adresse

Adresse linéaire =

Adresse virtuelle

+ adresse linéaire du début du segment

⇔ Relogement dynamique



Segmentation mémoire

- L'adresse physique du début de chaque segment est stockée dans une **table accessible seulement par le système d'exploitation**
- On y trouve aussi la longueur du segment et d'autres informations
- Stockée en **mémoire physique** à une **adresse physique** connue du processeur
- Chaque entrée de la table représente un segment différent : **descripteur de segment**
- Son index dans la table permet de retrouver un descripteur : **sélecteur de segment**

Segmentation mémoire

Processus segmenté P1

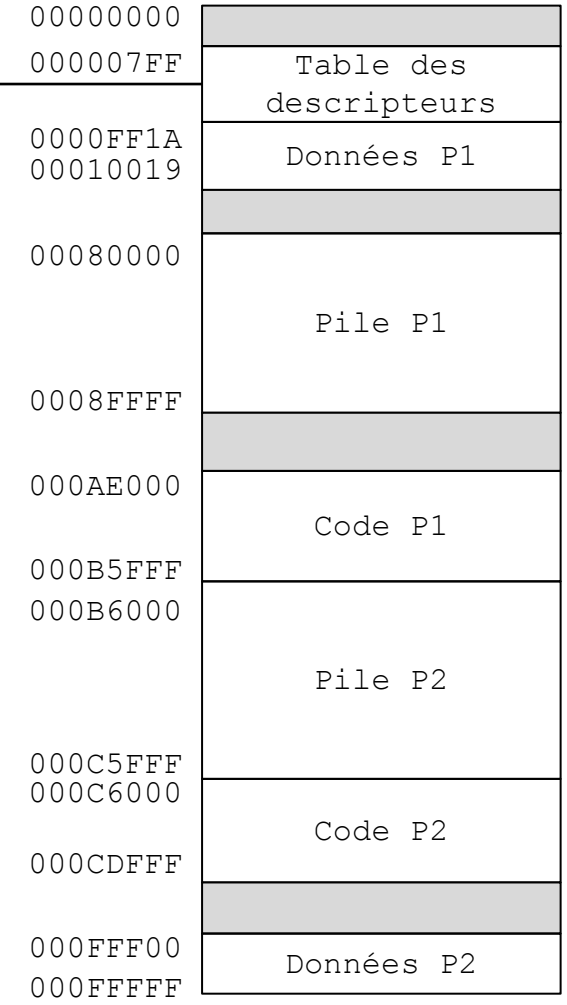
Adr Virtuelle	Contenu mémoire
00000000 00007FFF	Code P1 Sélecteur 0
00000000 000000FF	Données P1 Sélecteur 1
00000000 0000FFFF	Pile P1 Sélecteur 2

Processus segmenté P2

Adr Virtuelle	Contenu mémoire
00000000 00007FFF	Code P2 Sélecteur 3
00000000 000000FF	Données P2 Sélecteur 4
00000000 0000FFFF	Pile P2 Sélecteur 5

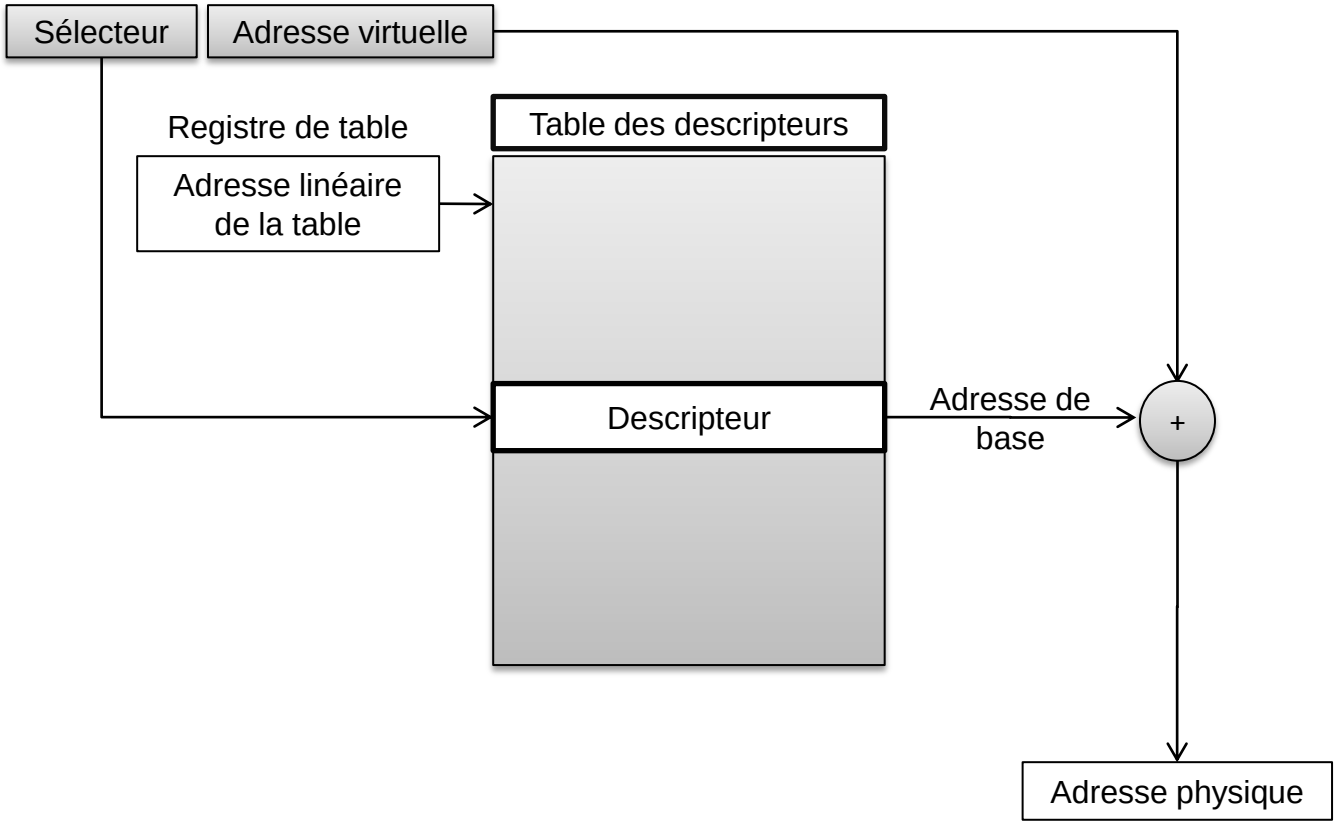
Sélecteur	Base	Limite
0000	000AE000	00008000
0001	0000FF1A	00000100
0002	00080000	00010000
0003	000C6000	00008000
0004	000FFF00	00000100
0005	000B6000	00010000
0006	…	…

Espace d'adressage linéaire / réel



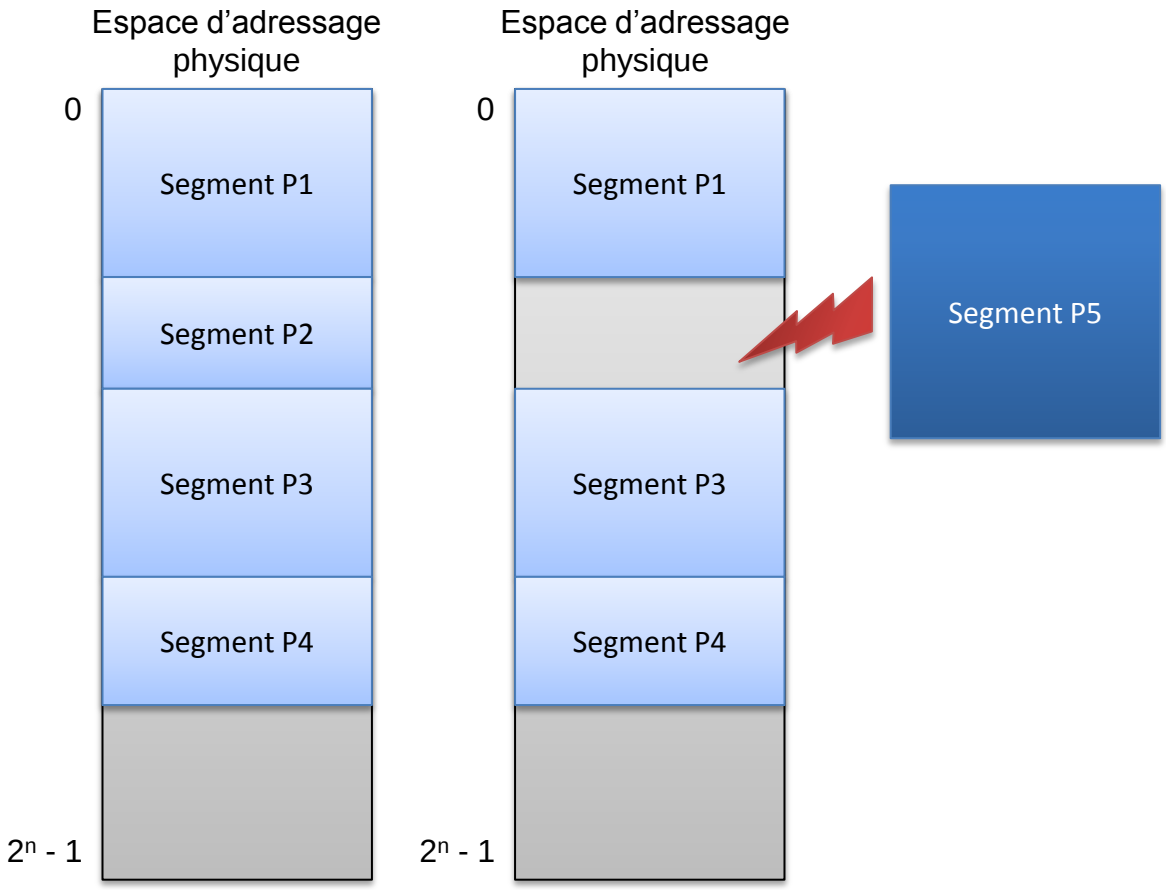
Segmentation mémoire

- Récapitulons… conversion d'adresse virtuelle en adresse physique



Fragmentation externe

- P2 est supprimé et le gestionnaire de mémoire veut allouer un nouveau segment pour P5…



Segmentation mémoire

- Avantages
  - Chaque segment occupe exactement la quantité de mémoire physique nécessaire
  - Tables de taille réduite (une entrée par segment)
- Inconvénients
  - Fragmentation externe
  - Mécanisme d'allocation mémoire inefficace (recherche linéaire, compactage)

D’un point de vue pratique…

- Il serait malcommode de devoir à chaque opération mémoire répéter le sélecteur du segment auquel l’opération fait référence
  - **Les sélecteurs changent rarement (voire jamais) au cours de la vie d’un processus.**

- Le processeur dispose de 6 registres de segment, contenant chacun un sélecteur définissant les segments de travail:
  - CS : segment de code contenant les instructions
  - DS : segment de données contenant les données
  - ES, FS, GS : segments supplémentaires libres
  - SS : sélecteur du segment contenant la pile

D’un point de vue pratique (bis)

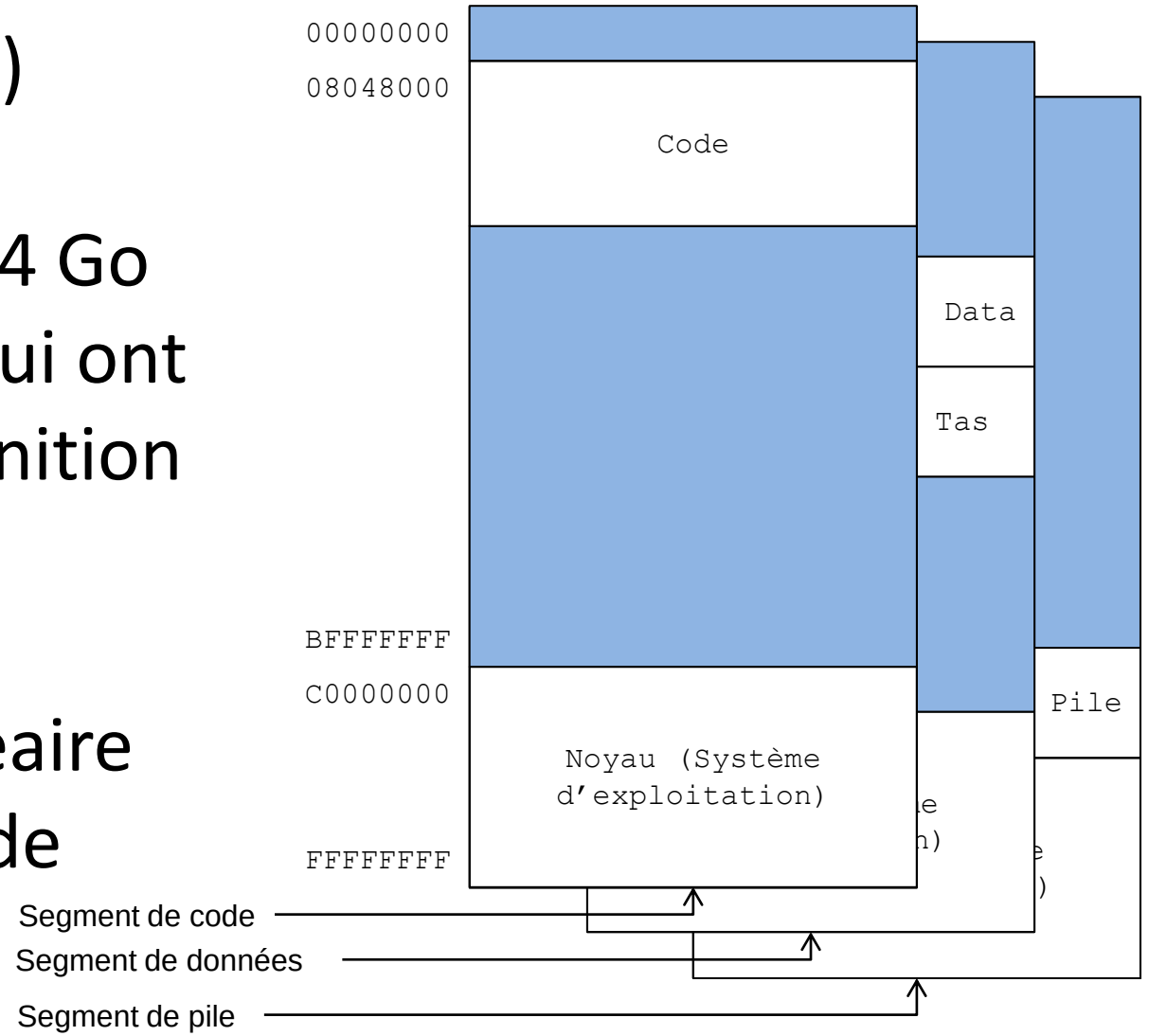
- Les modes d’adressages considèrent par défaut que les adresses de données sont relatives au segment de sélecteur DS.
- Cependant :
  - Si un registre de base est **EBP** ou **ESP**, le segment par défaut est défini par le sélecteur **SS**
- `MOV EAX, [EBP]` => Adresse SS:EBP
  - On peut spécifier explicitement le segment :
    - `MOV CS:[6000h], AX`
    - `MOV SS:[EBX + ESI * 2 + 4], EAX`
  - On parle de **substitution de segment**

Pagination mémoire

- La segmentation fait correspondre l'organisation de la mémoire centrale à l'image que le programmeur a de son programme.
- Toutefois :
  - Elle est complexe puisque chaque programme peut avoir une segmentation différente
  - Chaque segment doit pouvoir être alloué intégralement en mémoire centrale (que se passe-t-il alors dans le cas d'un processus Windows, Linux dont l'espace d'adressage virtuel est de 4 Go ?)
- Solution : utiliser la pagination mémoire.

Pagination mémoire

- Linux (par ex.) définit trois segments de 4 Go superposés qui ont la même définition pour chaque processus, d'adresse linéaire de base 0 et de limite 4 Go



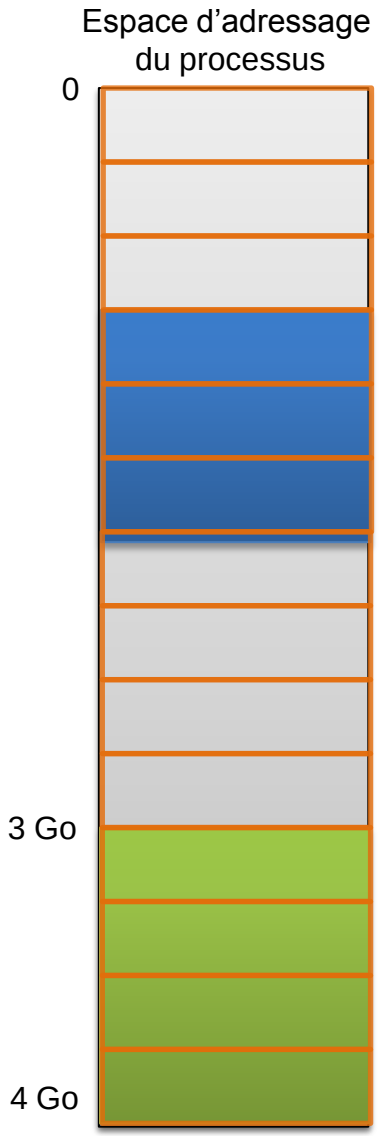
Pagination mémoire

- Principe général :
  - Construire une table de correspondance adresse virtuelle ⇔ adresse physique

- Avantage : chaque adresse virtuelle peut être à un emplacement différent en mémoire centrale. Une adresse virtuelle non utilisée peut être simplement ignorée (pas de correspondance)
- Inconvénient : taille de la table de correspondance…

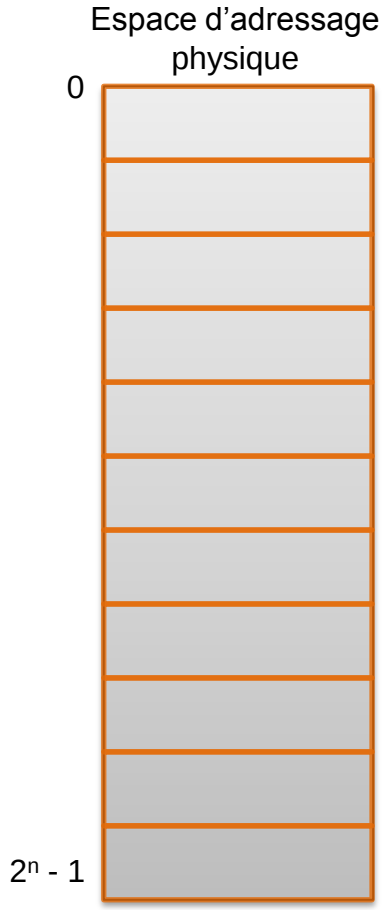
Pagination mémoire

- On divise l'espace d'adressage virtuel des processus en zones contigües de taille fixe (4 ko sur IA32) : **les pages**

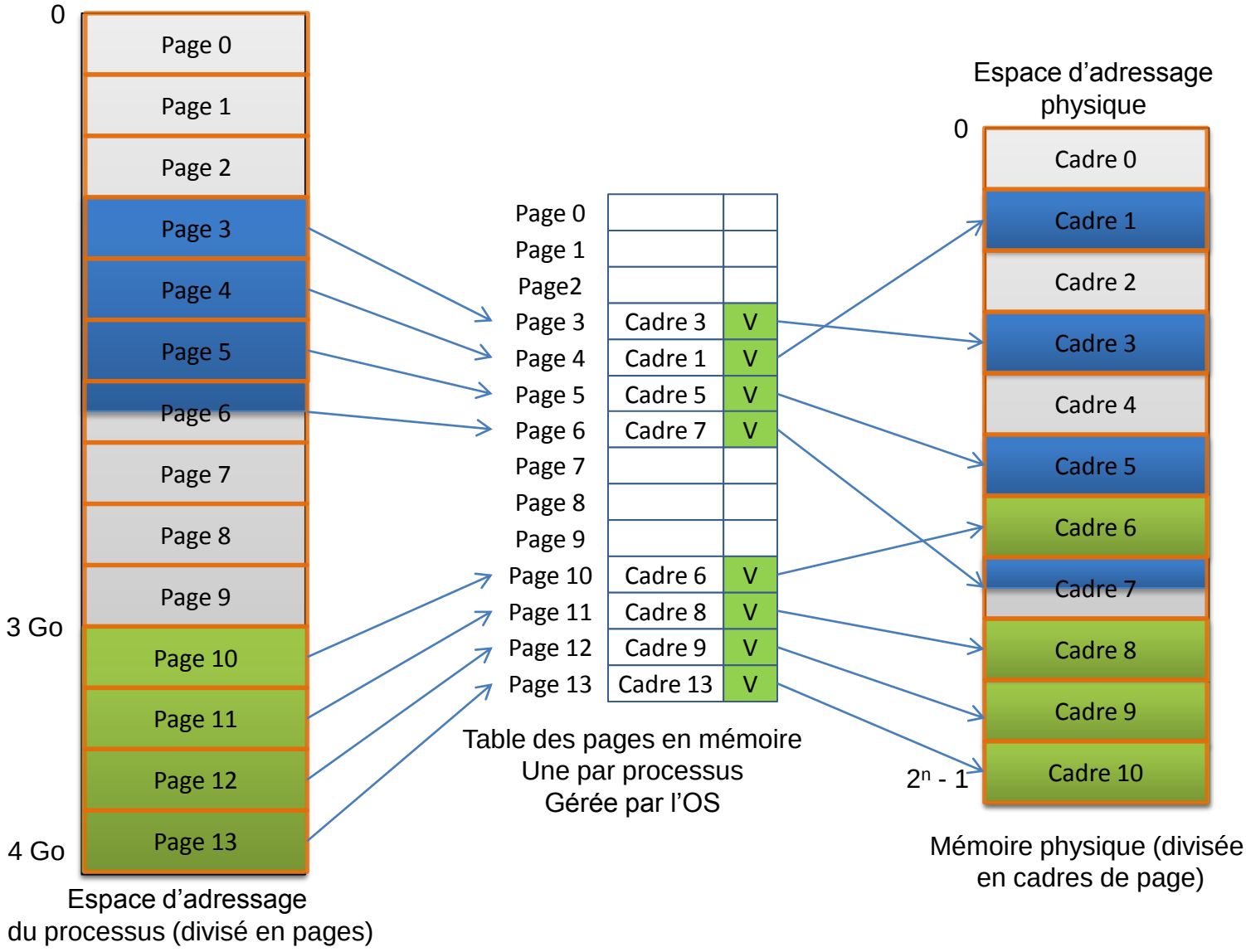


Pagination mémoire

- On divise la mémoire physique en zones contigües de taille fixe (4096 octets sur IA32) : **les cadres de pages**

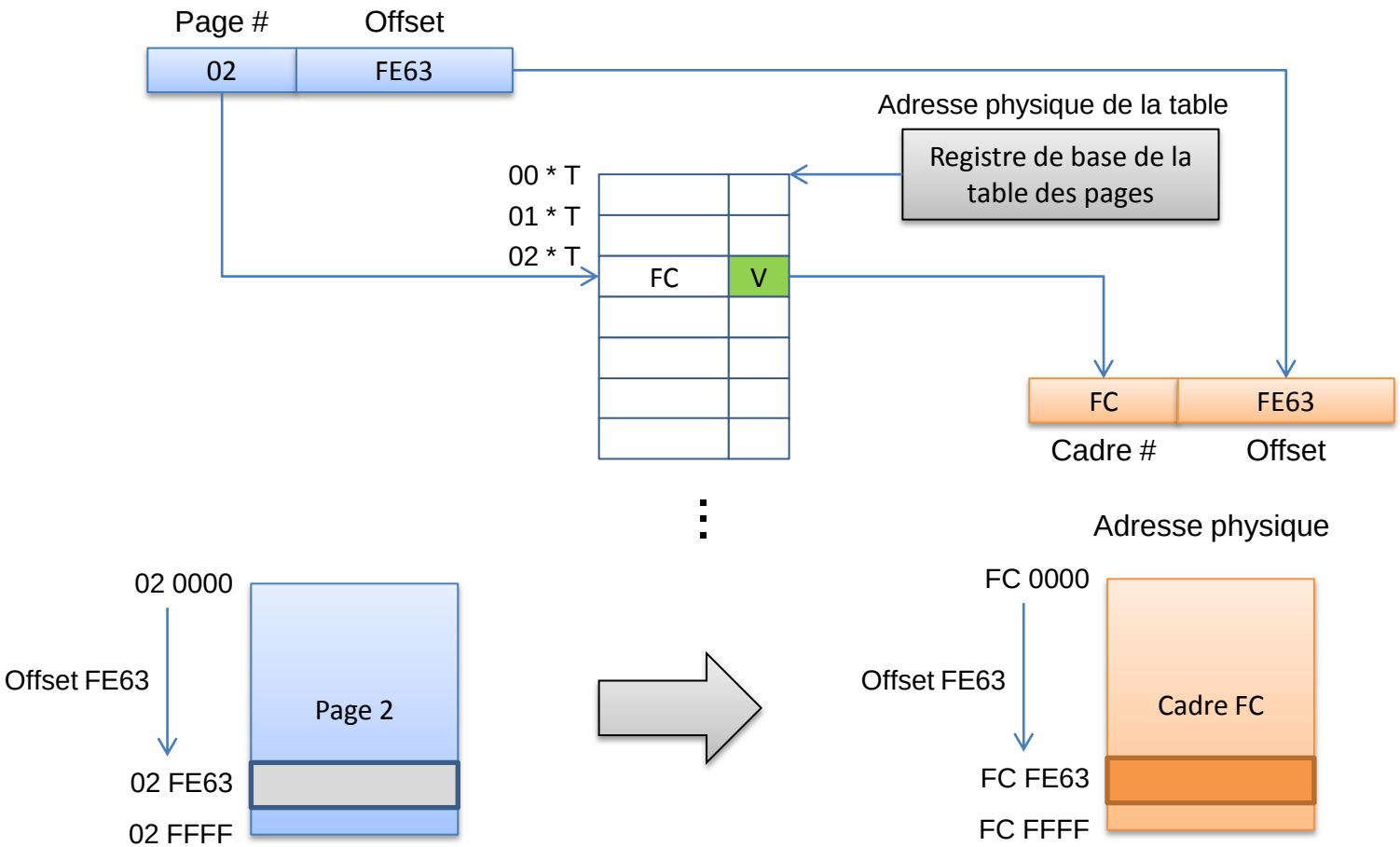


Pagination mémoire



Pagination mémoire

- Translation d'adresse avec un seul niveau de table



Pagination mémoire

- Problème : limiter la taille des tables
  - La table contient une entrée pour chaque page de l'espace d'adressage virtuel
  - Ex : sur IA32, pages et cadres de 4 Ko
    - 4 Go / 4 Ko = 1 048 576 entrées par processus (4 Mo, chaque entrée fait 32 bits)
- Solution : diviser l'espace d'adressage virtuel en « groupes de pages », les groupes en groupes et ainsi de suite : plusieurs niveaux de table
- Sur IA32 : groupes de 1024 tables, seules les tables utilisées effectivement sont allouées en mémoire principale. Une table principale, le répertoire des pages,

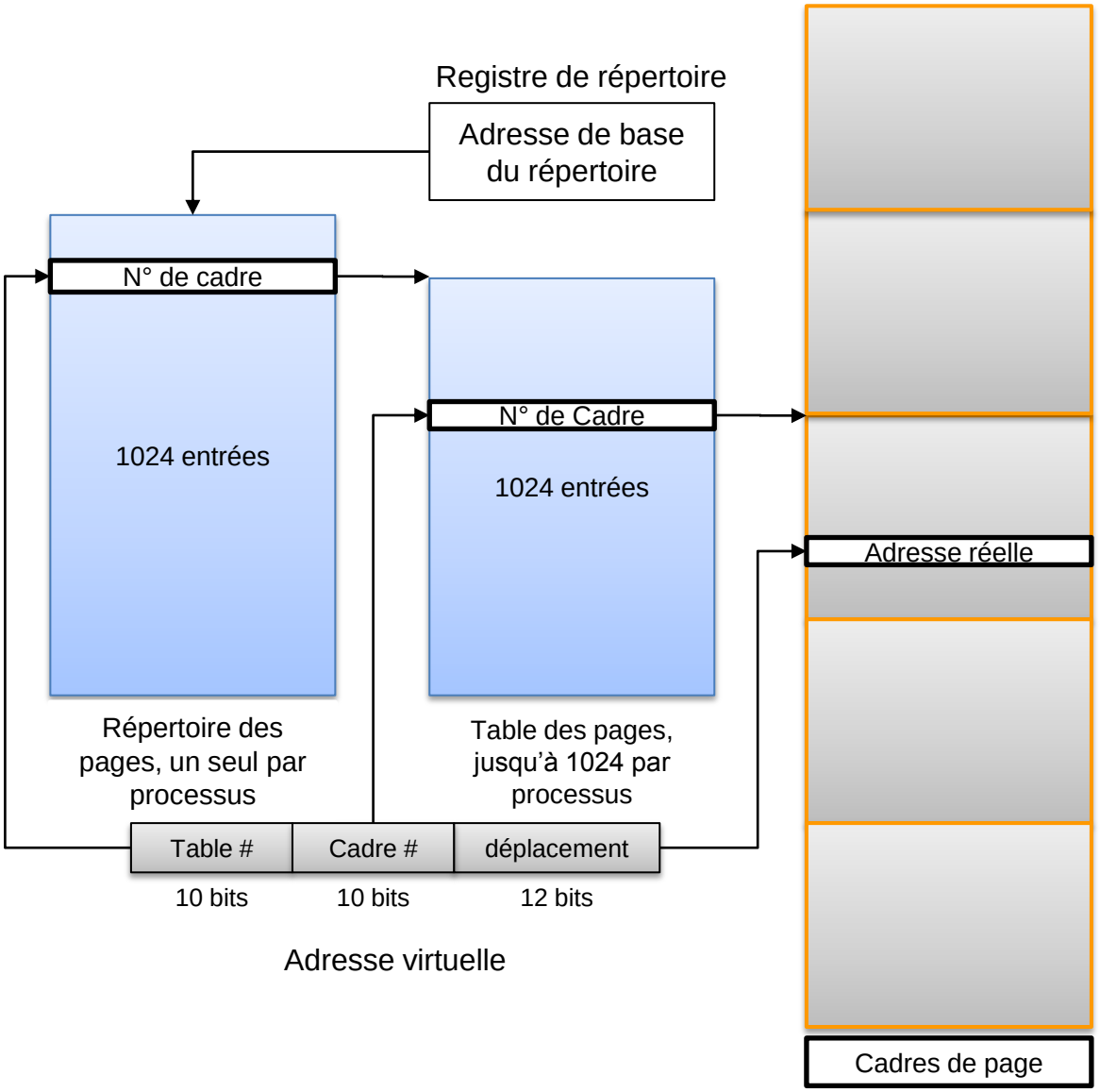
Pagination mémoire -

Translation d'adresse avec 2 niveaux de table (IA32)

Chaque table de page contient 1024 entrées => représente un bloc de pages contigu de 4 Mo.

Le répertoire des pages comprend 1024 entrées, 1 par table de page => à chaque entrée correspond un bloc de 4Mo.

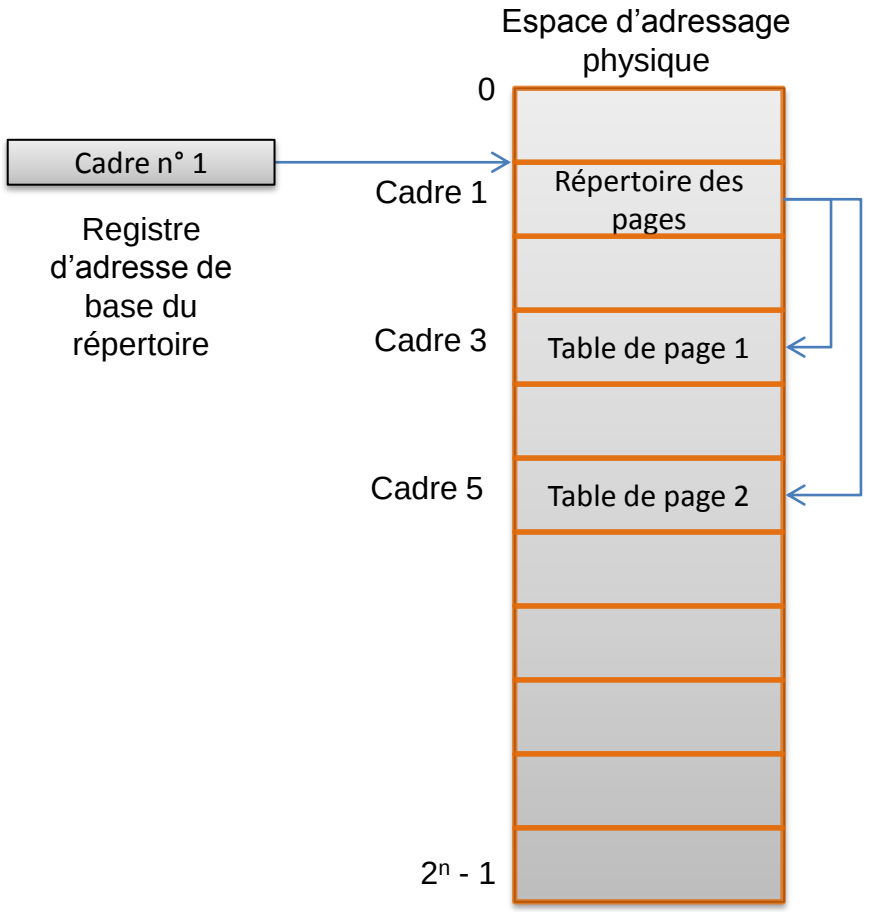
1024 * 4Mo = 4Go.



Pagination mémoire

- Où sont les tables ?
  En mémoire principale !

- Seules les tables de pages qui ont une entrée valide dans le répertoire sont allouées en mémoire



Pagination mémoire

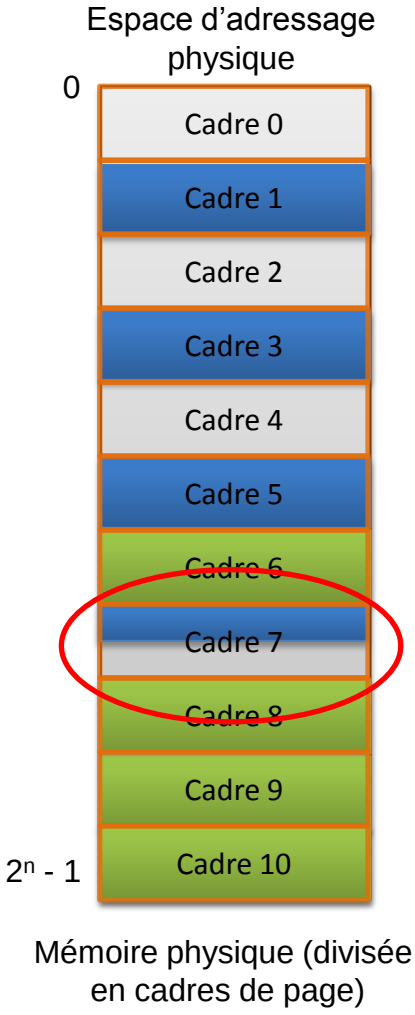
- La translation est une opération coûteuse
- **Principe de localité :**

  « Lors d'un accès mémoire, il y a de fortes chances de l'accès suivant ait lieu à une adresse linéaire proche »

- Le processeur utilise une mémoire associative interne afin d'enregistrer les dernières conversions : il s'agit du **TLB** (*Translation Lookaside Buffer*)

Pagination mémoire

- Fragmentation interne
  - Certains cadres ne sont pas occupés entièrement
  - Il est possible de diminuer la taille des cadres pour atténuer cet inconvénient
  - Compromis avec la taille des tables



Pagination mémoire

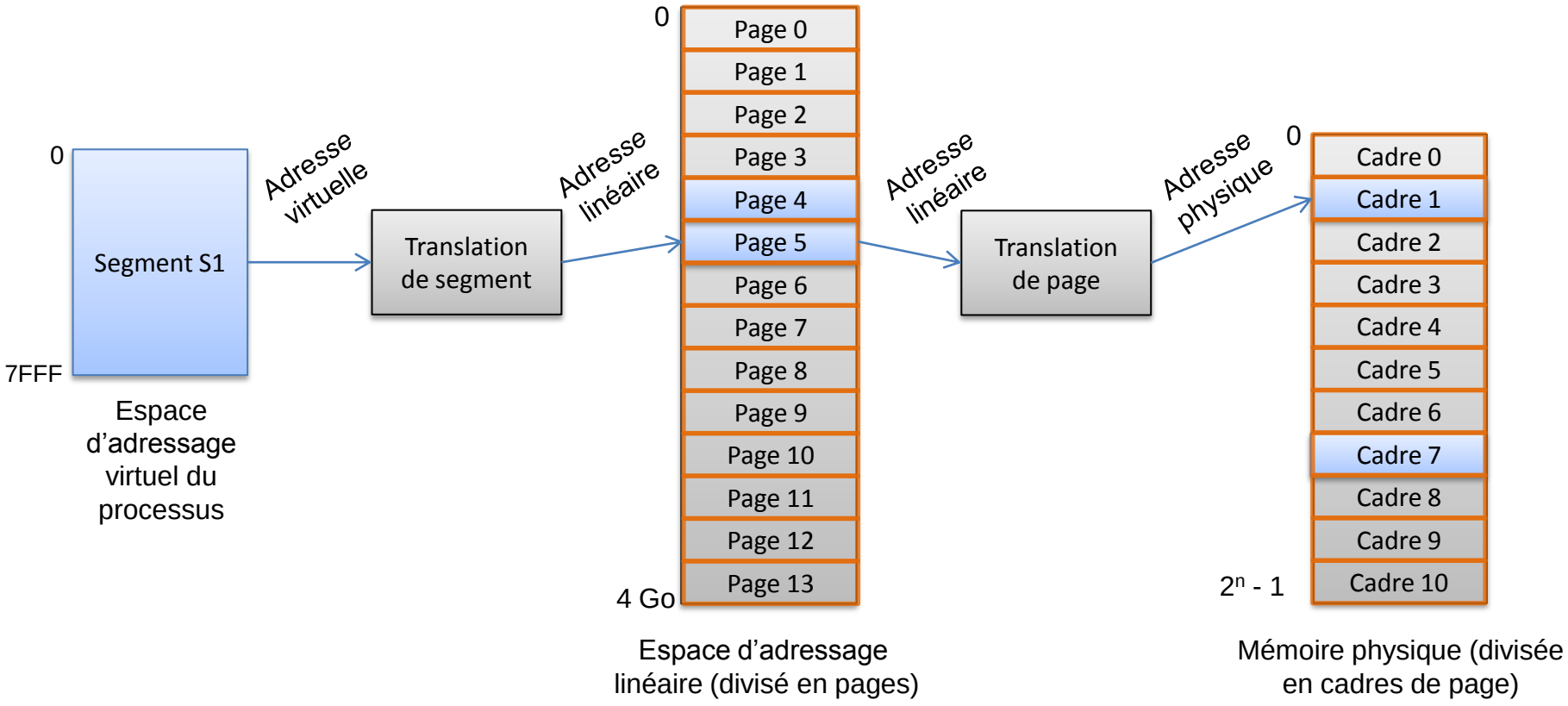
- Quelques fonctionnalités non détaillées ici
  - **Demand paging** : quand il n'y a plus de cadres disponibles, copier le contenu d'un cadre d'un processus sur disque et le remplacer par un autre
  - **Protection** : puisque chaque processus dispose de sa propre table des pages, elles ne contient aucune information relative aux cadres des autres processus. Si il essaie d'accéder à une adresse virtuelle qui ne correspond pas à un des cadres dont il est propriétaire, le processeur (et l'OS) peut interdire l'accès
  - **Partage de mémoire** : deux processus peuvent partager une zone de mémoire en ayant chacun une entrée de leur table de pages correspondant à un cadre de page unique. Dans ce cas, des bits spécifiques dans l'entrée indiquent les droits d'accès du processus au cadre en question
- Plus de détails dans les UV sur les systèmes d'exploitation !

Pagination mémoire

- Avantages
  - Allocation aisée (tête de liste des cadres libres)
  - Pas de fragmentation externe puisqu'on peut découper un bloc de mémoire virtuelle pour le faire correspondre à plusieurs blocs de mémoire physique
- Inconvénients
  - Fragmentation interne
  - Plusieurs accès par accès mémoire (n accès pour les tables, 1 pour l'accès à la donnée)
  - Taille importante des tables (4 Mo si tous les cadres sont occupés, à multiplier par le nombre de processus)

Segmentation et pagination

- On peut mélanger les deux. Sur IA32, la segmentation est toujours active, la pagination peut être désactivée



Adresses : récapitulons

- Les adresses **virtuelles** sont celles manipulées par le **processus** (et le programmeur)

- Les adresses **linéaires** sont les adresses vues du point de vue de **l'espace total d'adressage du processeur – Ce sont aussi les adresses physiques si la pagination n'est pas utilisée**

- Les adresses **réelles** ou **physiques** sont celles vues par le **matériel**