Système d'Exploitation

La Mémoire

Juan Angel Lorenzo del Castillo
juan-angel.lorenzo-del-castillo@cyu.fr

Contributions de :
Stefan Bornhofen
Taisa Guidini Goncalves
Mariem ALLOUCH MAHDI

Plan

Mémoire centrale ou vive

(Mémoire RAM - *Random Access Memory*)

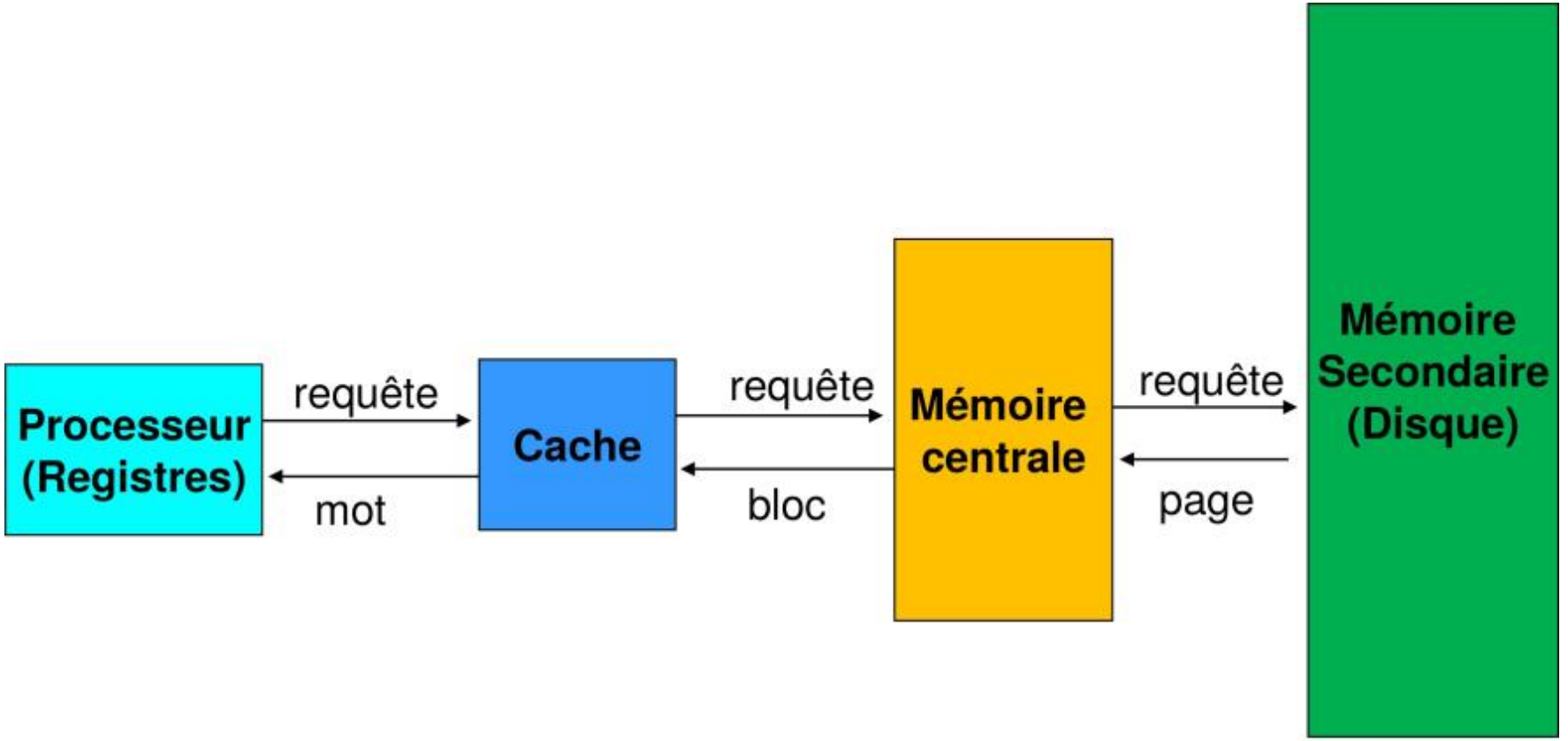
La mémoire vive (ou RAM) est la mémoire dans laquelle les données et processus sont placées lors de leur traitement

Quelques caractéristiques de la RAM :

- mémoire à accès rapide
- mémoire volatile i.e., besoin de courant électrique
- "cases" numérotées à partir de 0
  - le numéro correspond à l'adresse
  - chaque case contient un octet (8 bits; 1 bit : 1 ou 0)
    1 octet = 8 bits $\rightarrow 2^8 = 256$ valeurs possibles
- Deux types d'opérations possibles : lecture / écriture

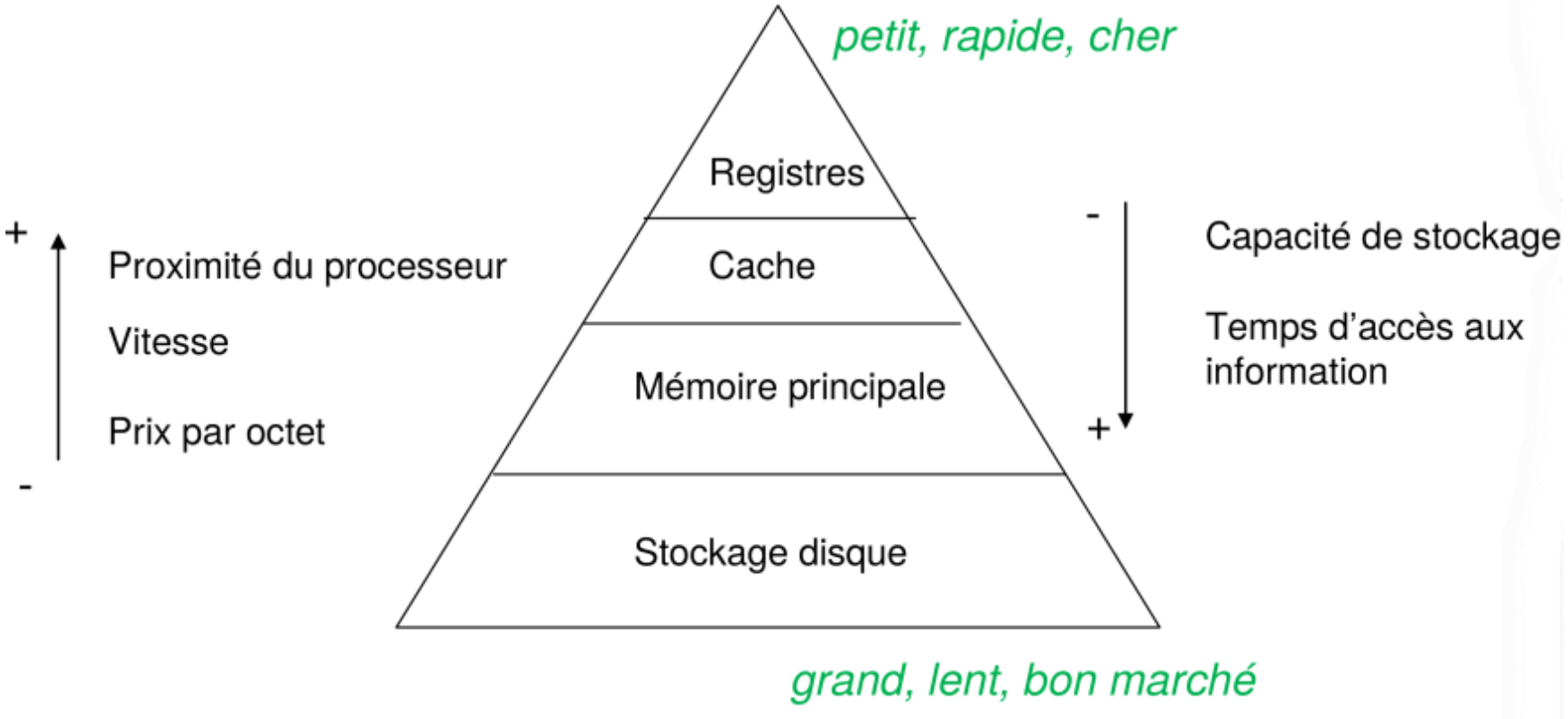
Plan

Hiérarchie de la mémoire



	Registre	Cache	Mémoire centrale	Disque
Capacité	<Ko	Mo	Go	>To
Temps d'accès (ns)	1-5	3-10	10-400	5 000 000
Coût relatif	50	10	1	0,001

Hiérarchie de la mémoire



Hiérarchie de la mémoire

Différents types de mémoire (on souhaite grande, rapide, pas cher)

- avec différentes caractéristiques :

  volatile ou pas, temps d'accès, débit, taille, prix/taille, ...

- du plus cher et plus rapide, au moins cher et plus lent :

	tps d'accès (cycles)	débit	taille typique
registres de processeurs	$\simeq$ 1	-	20 × 4 octets
cache niveau 1	$\simeq$ 4	qq Go/s	128 Ko
cache niveau 2	$\simeq$ 10	idem	1 Mo
cache niveau 3	$\simeq$ 20	idem	6 Mo
RAM	$\simeq$ 100	idem	8 Go
Disque dur SSD	$\simeq$ 1ms	500 Mo/s	250 Go
Disque dur magnétique	$\simeq$ 10 ms	100 Mo/s	1 To

Les prix du Gigaoctet

En 2015, RAM : 10 euros/Go ; SSD : 0, 27 euros/Go ; DD : 0, 05 euros/Go

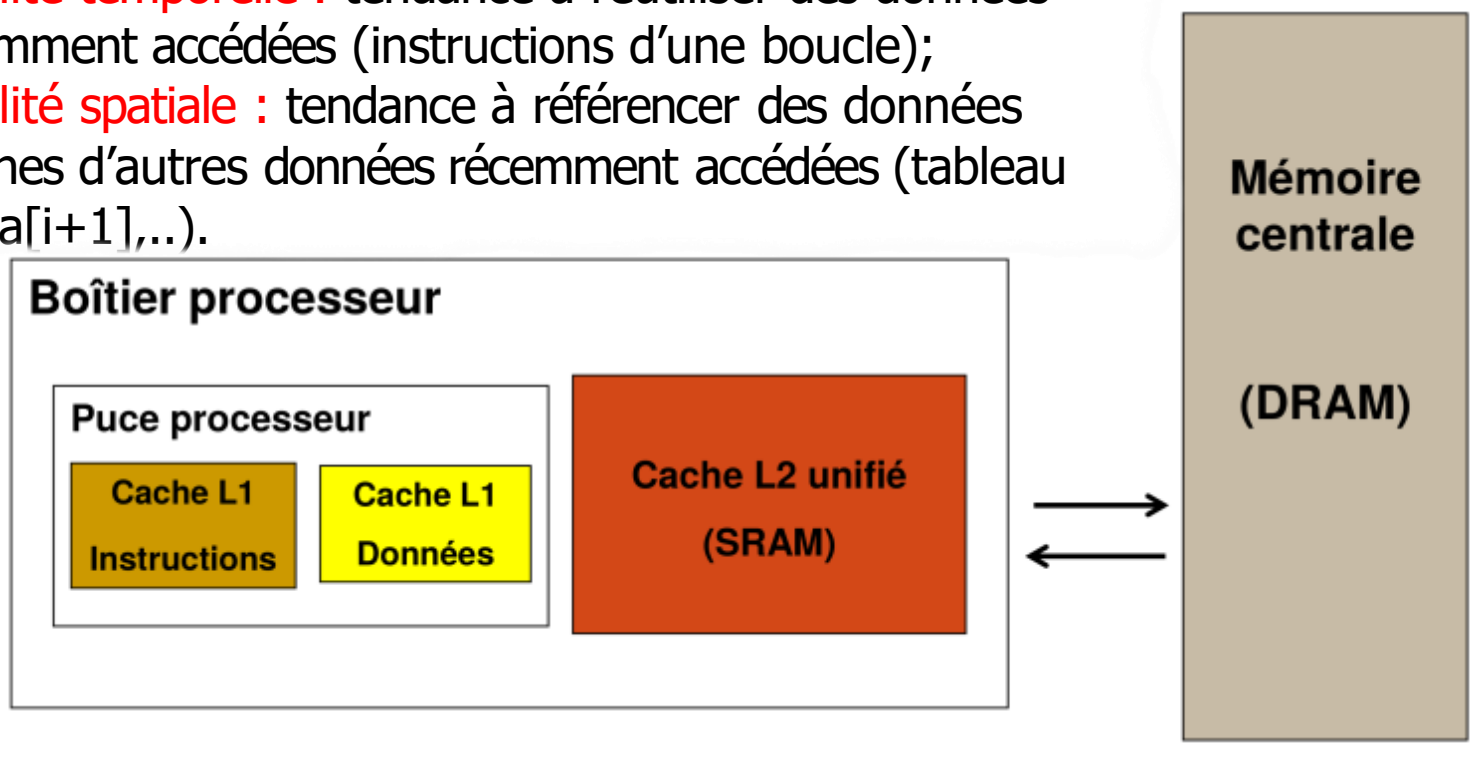
Évolution des prix du Go (DD)

- 1956 : 1 Go = 26 000 000 euros/Go !!!
- 1973 : 1 Go = 3 000 000 euros/Go
- 1980 : 1 Go = 100 000 euros/Go
- 1987 : 1 Go = 40 000 euros/Go
- 1995 : 1 Go = 800 euros/Go
- 2002 : 1 Go = 2 euros/Go
- 2007 : 1 Go = 0,28 euros/Go
- 2015 : 1 Go = 0,05 euros/Go

Mémoire Cache

Éviter de rechercher en mémoire centrale des données déjà cherchées précédemment en les conservant près du processeur dans une petite mémoire à accès rapide.

- Géré par le matériel (hors SE)
- Principe de localité :
  - Localité temporelle : tendance à réutiliser des données récemment accédées (instructions d'une boucle);
  - Localité spatiale : tendance à référencer des données voisines d'autres données récemment accédées (tableau a[i], a[i+1],..).



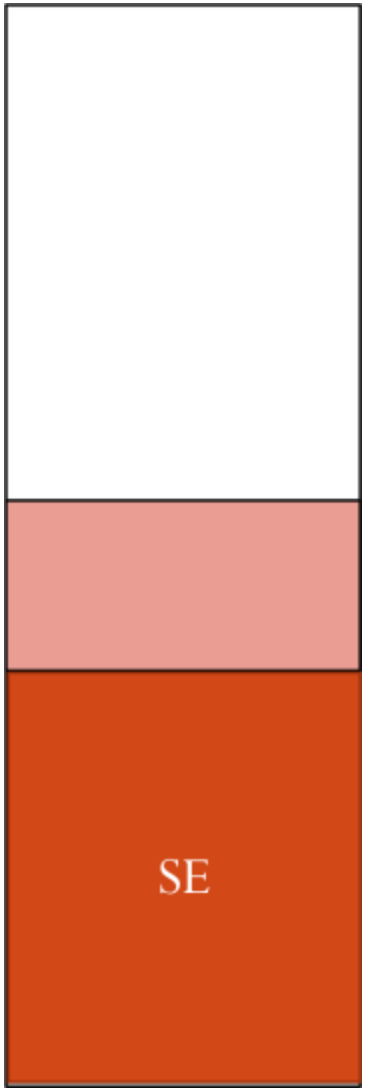
Plan

Utilisation de la mémoire dans des systèmes monotâche

Seules les instructions stockées en mémoire centrale peuvent être exécutées par la CPU.

- En monoprogrammation
  - Mémoire réservée au SE
  - Mémoire réservée au (seul) programme à exécuter
  - Adressage direct possible

  ...et en multiprogrammation (systèmes multitâche) ?



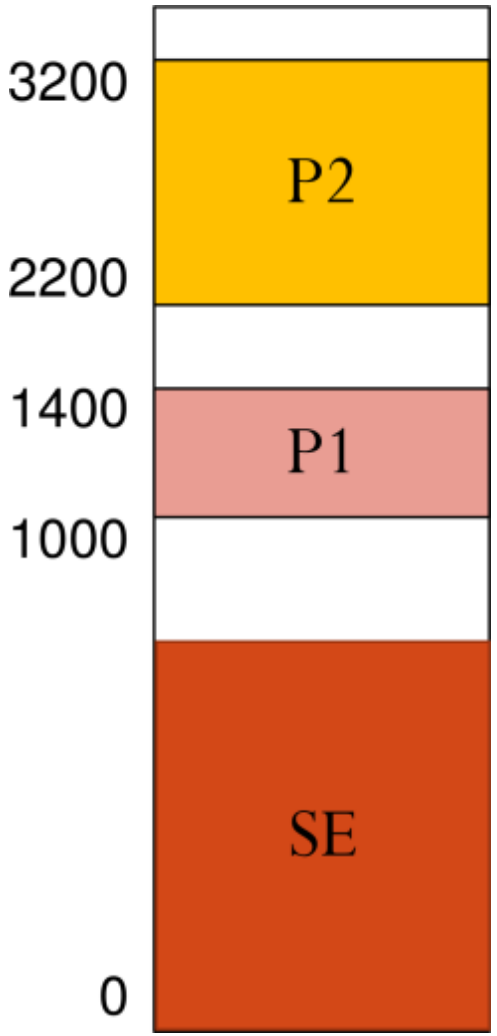
Utilisation de la mémoire dans des systèmes multitâche

Système multitâche → Plusieurs processus en RAM

Espace d'adressage

- Base / limite
- Adresse physique = base + adresse
- Adresse maximale = base + limite

Processus	Base	Limite
P1	1000	400
P2	2200	1000



Utilisation de la mémoire dans systèmes multitâche

Système multitâche → Plusieurs processus en RAM

Trois types de problèmes :

- Isolation de processus
- Placement des processus
- La taille de la RAM est limitée

Aucun problème s'il n'y a qu'un seul processus en RAM

Utilisation de la mémoire dans systèmes multitâche

Problème d'isolation de processus :

> Un processus ne doit pas pouvoir accéder à la RAM d'un autre processus ou du système d'exploitation

Comment ?

- Le SE alloue une zone de RAM à un seul processus
- Déclenchement d'une erreur si un processus accède à la mémoire non allouée

Pourquoi ?

- Pour garantir un fonctionnement correct des processus ou du SE
- Pour garantir la confidentialité des données

Utilisation de la mémoire dans systèmes multitâche

Problème de placement des processus en RAM :

> Comment placer les processus dans la mémoire RAM ?

Solution la plus simple : Chaque processus utilise une zone contiguë de RAM.

Cependant...

- Comment le SE peut savoir combien de RAM un processus va avoir besoin ?
- Comment gérer la variation de besoins ? (ex : ouverture d'un petit fichier, puis d'un gros)
- Que se passe-t-il s'il n'y a pas de place assez grande (mais plein de petites places) ?
- Que se passe-t-il s'il y a plusieurs places assez grandes ?

Remarque : il est en général impossible de déplacer un processus en RAM

Utilisation de la mémoire dans systèmes multitâche

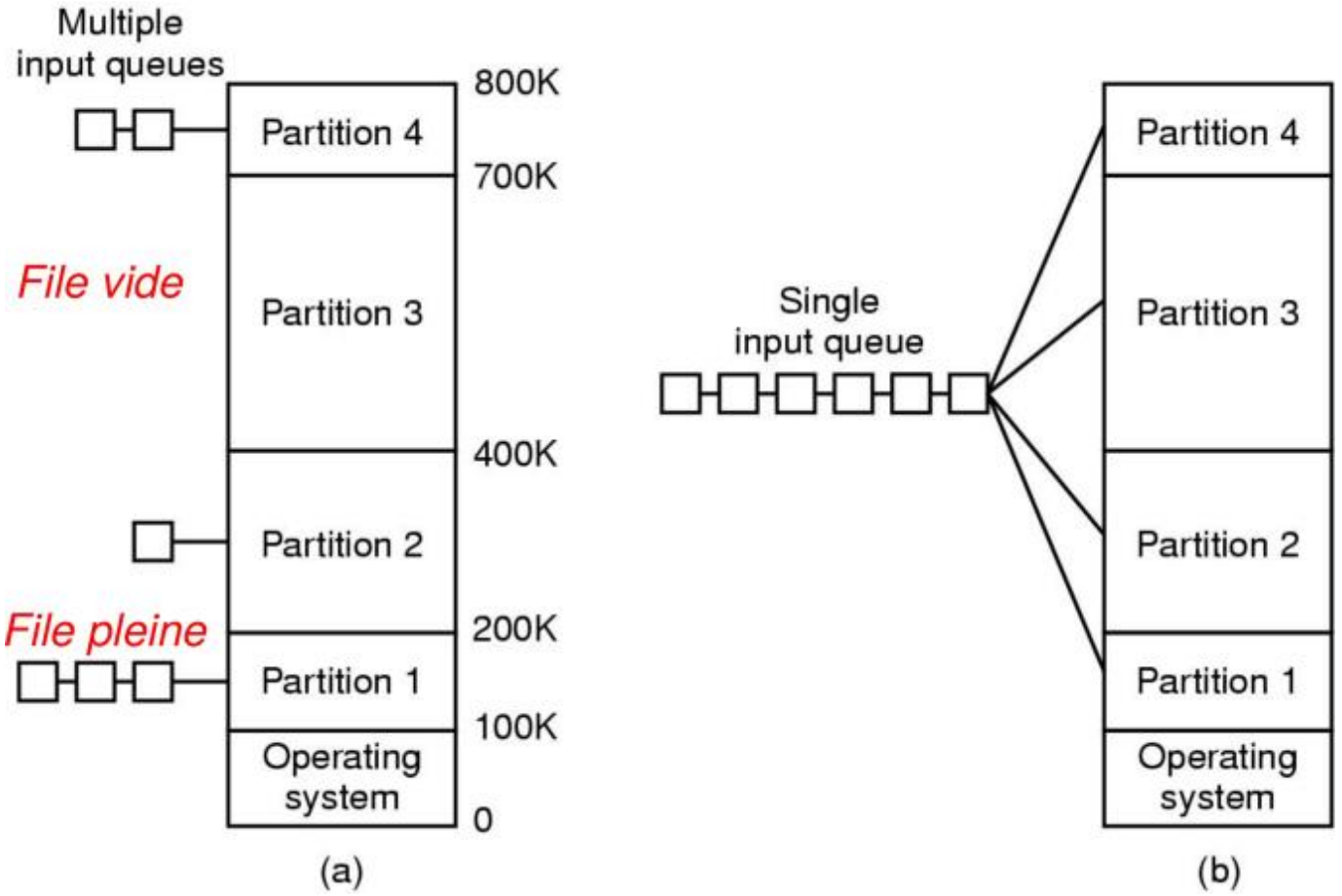
Partitions de taille fixe :

> Partitionner la mémoire pour contenir un nombre maximum de programmes

- Partitions de même grandeur? De différentes grandeurs?
- Un gros programme ne rentre que dans une grande partition
- Un petit programme rentabilise mal une grande partition
- Gaspillage de mémoire.
- Les files multiples présentent un inconvénient lorsque la file des grandes partitions est vide et celles des petites est pleine
- une alternative consiste à utiliser une seule file

Utilisation de la mémoire dans systèmes multitâche

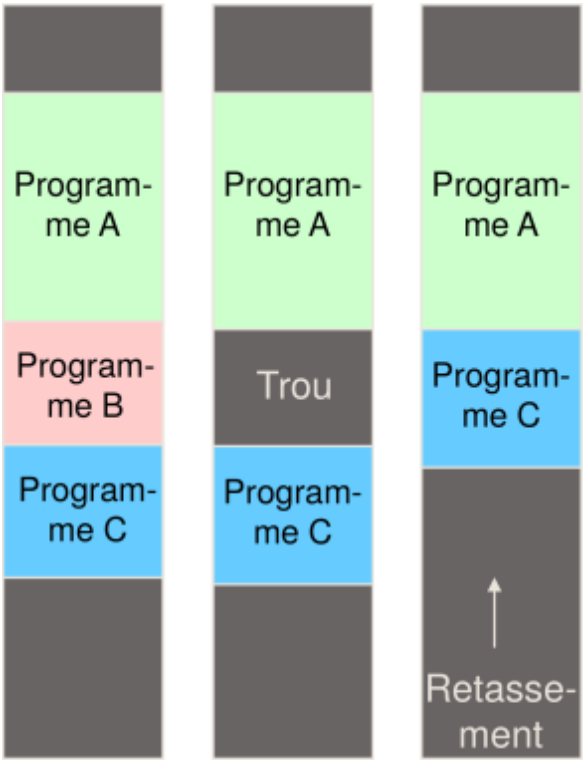
Partitions de taille fixe :



Utilisation de la mémoire dans systèmes multitâche

Partitions de taille variable :

- Le nombre, la position et la taille des partitions varient dynamiquement
- Améliore l'usage de la mémoire
- mais complique sa gestion
- Lorsqu'un programme termine son exécution, il laisse un trou en mémoire
- Compactage de mémoire = relocaliser des programmes en cours d'exécution (« retassement »).



Utilisation de la mémoire dans systèmes multitâche

Problème de taille de RAM limitée :

> Comment gérer le petit espace mémoire de la RAM ?

Pourquoi : car les processus en utilisent de plus en plus

Loi de Parkinson

"*Work expands so as to fill the time available for its completion.*"

Les programmes s'acroissent pour remplir la mémoire disponible qui leur est réservée.

Utilisation de la mémoire dans systèmes multitâche

Missions du SE

- Connaître les parties libres et occupées de la mémoire
- Allouer de la mémoire aux processus qui en ont besoin
- Libérer la mémoire d'un processus lorsque celui-ci se termine
- Gérer le cas où la mémoire ne peut pas contenir tous les processus actifs

Plan

Abstraction de la mémoire

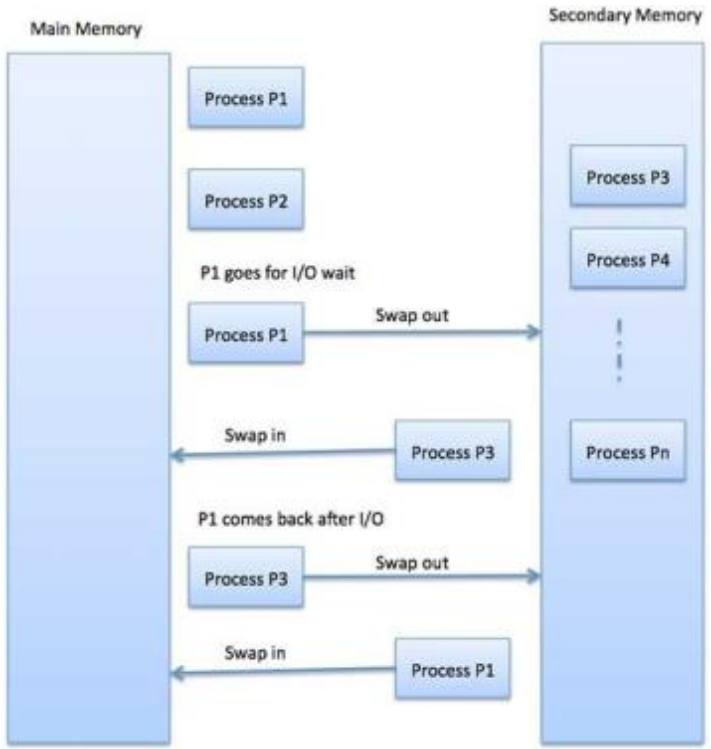
- Dans un environnement de multiprogrammation, la mémoire est normalement insuffisante pour maintenir tous les processus courants actifs. Il faudra :
  - sauvegarder les processus sur le disque
  - ...et les recharger dynamiquement.
- Deux approches pour gérer la surcharge de mémoire :
  1. swapping (va-et-vient) : sauvegarde du processus dans son intégralité
  2. La mémoire virtuelle : exécution du processus partiellement en mémoire

Ces approches permettent de lever la contrainte de dimension de la mémoire

Swapping (va-et-vient)

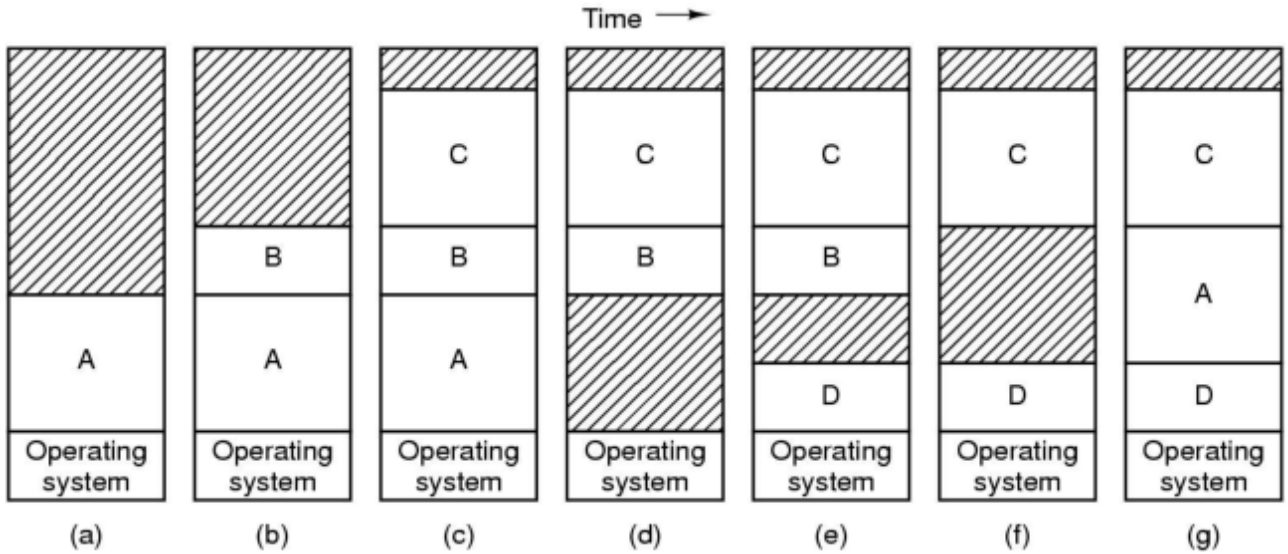
Mécanisme dans lequel un processus peut être temporairement supprimé de la mémoire principale (ou déplacé) vers le stockage secondaire (disque) afin de rendre cette mémoire disponible pour d'autres processus. Plus tard, le système remplace le processus du stockage secondaire vers la mémoire principale.

- **But :** Utiliser le disque pour simuler une taille de RAM plus grande.
- **Principe :** Une zone du disque (normalement une partition ou fichier) est mise à disposition du SE.
- **Avantage :** libère la RAM
- **Inconvénient :** L'accès au disque est lent.



Swapping (va-et-vient)

- Chaque processus est considéré dans son intégralité.
  - Soit un processus est dans son intégralité en mémoire
  - soit il est supprimé intégralement de la mémoire et stocké sur le disque.
- Le processus peut être rechargé à nouveau sur une zone différente en mémoire : réallocation dynamique.
- Des "trous" peuvent apparaître dans la mémoire : possibilité de compactage (processus lent).



Mémoire virtuelle

Le problème de partage de la mémoire entre processus :

- Des plages d'adressage trop grandes.
- Plusieurs processus en mémoire simultanément : le *swapping* est lent.

Solution : la mémoire virtuelle

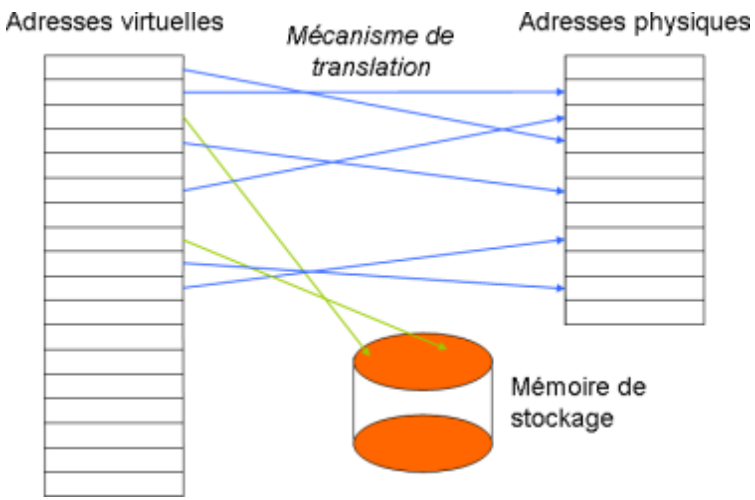
- Pour exécuter un programme (ou un segment de programme), il n'est pas forcément nécessaire de le garder entièrement en mémoire.
- Le SE conserve les parties (« pages ») en cours d'utilisation en mémoire et le reste sur disque (dans la zone de swap).
- Quand un programme attend le chargement d'une partie de lui-même → il est en attente d'E/S.

Pagination

Traiter séparément les adresses virtuelles référencées par le programme, et les adresses réelles de la mémoire physique.

- L'espace des adresses virtuelles est divisé en « pages » de taille fixe.
- La mémoire physique est divisée en « cases / cadres » de même taille.

- Ce n'est pas nécessaire d'avoir toutes les pages en mémoire
  - Les processus démarrent avec aucune page en mémoire.
- Phénomène du défaut de page
  - Le SE cherche la page en mémoire et, s'il n'y la trouve pas, la page sera copiée du disque à la mémoire.
  - Si la mémoire est plaine, il faudra déplacer une « page » de la mémoire au disque pour laisser la place à la « page » à charger en mémoire (**remplacement de page**).



Pagination

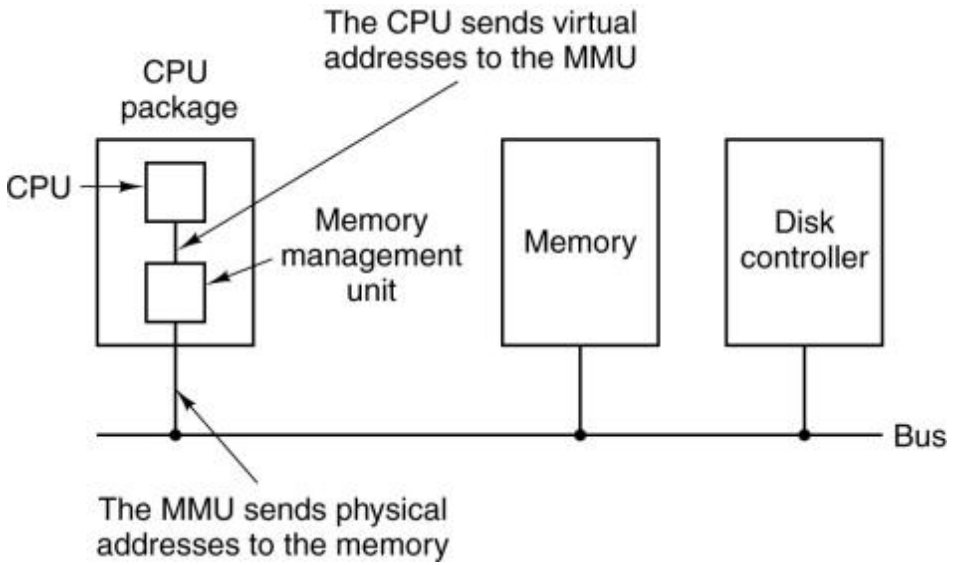
- Une adresse générée par un programme s'appelle une adresse virtuelle

  adresse virtuelle = page virtuelle + offset.

- Le SE maintient pour chaque programme une table de correspondance entre les pages virtuelles et les pages réelles.

  adresse réelle= f(page virtuelle) + offset.

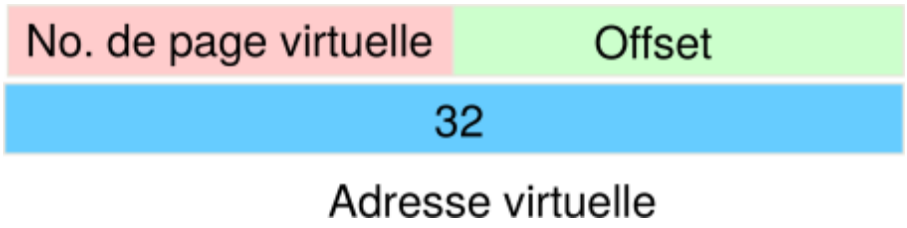
- La **MMU** (***Memory Management Unit***) fait la correspondance rapide entre les adresses virtuelles et les adresses physiques, comme un cache.



Pagination

Adresse virtuelle

- L'adresse virtuelle est scindée en deux champs : Les derniers bits définissent un offset (adresse dans la page), le reste définit le numéro de page virtuelle.

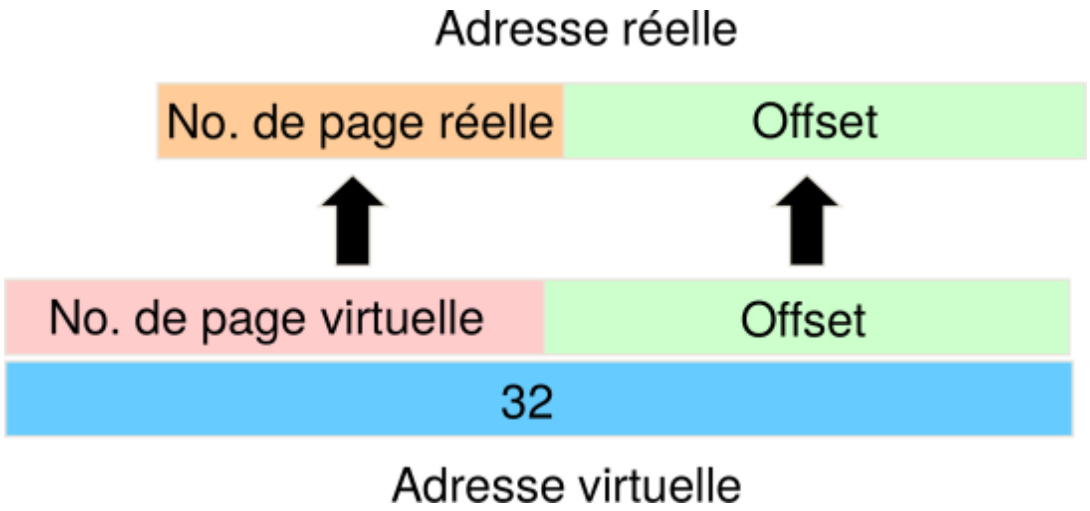


Exemple : Adresses virtuelles de 32 bits et pages de 2KB

- L'offset aura 11 bits (2^{11} = 2 K) / Taille des pages.
- Le numéro de page virtuelle aura 32 - 11 = 21 bits.
- L'espace d'adresses virtuelles est découpé en 2 M pages (2^{21} = 2 . 2^{20} = 2 M pages) de 2 KB

Pagination

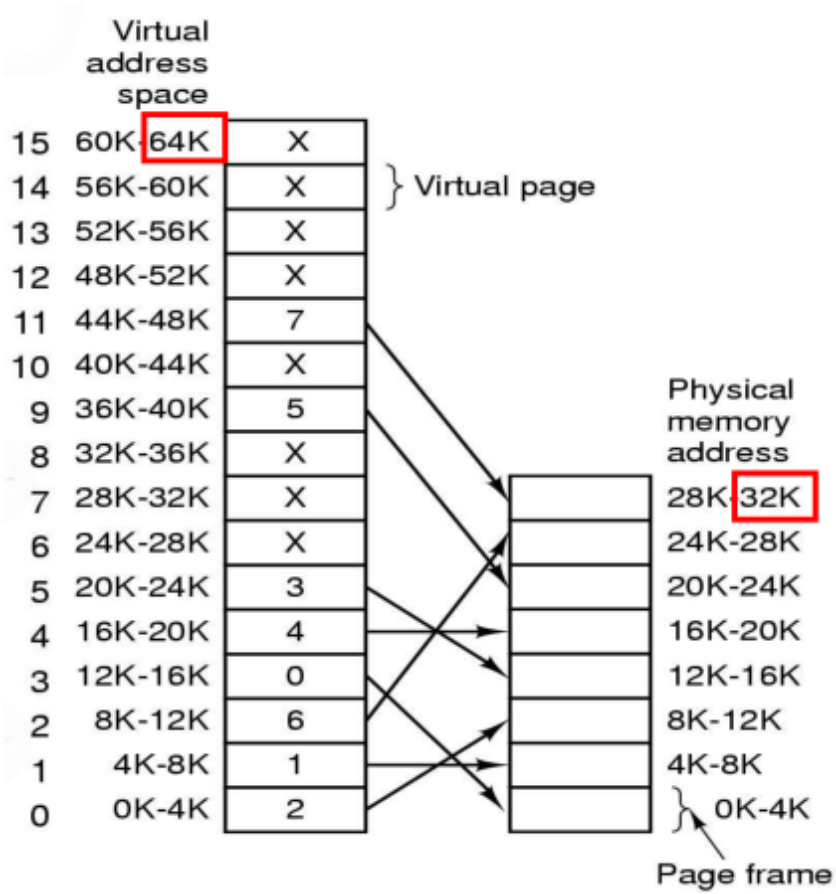
Traduction d'une adresse virtuelle à adresse physique



Pagination

Traduction d'une adresse virtuelle à adresse physique

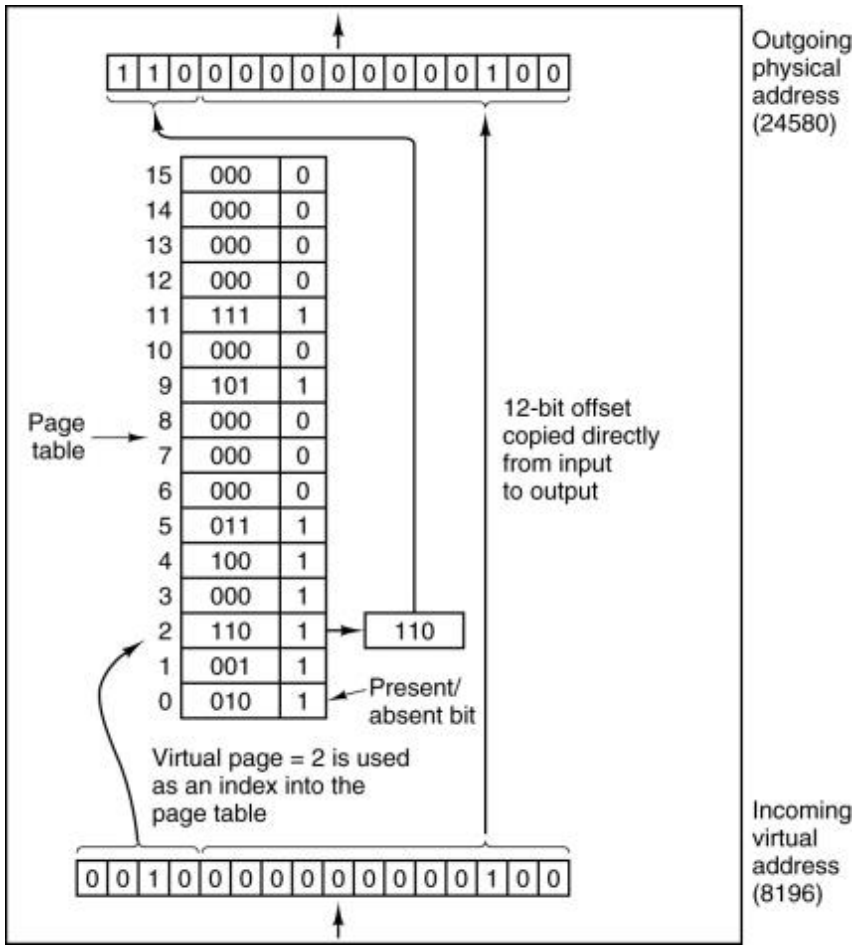
- Exemple : adresses de 16 bits (64KB), mémoire physique de 32KB.
- Espace d'adresses virtuelles en pages.
- Cadres de page (*Page frames*) en mémoire physique (même taille que les pages).
- Transfert entre RAM et disque des pages complètes.
- Pages de 4KB → 16 pages (MV = 64KB), Cadres de page de 4KB → 8 cadres de page (MP = 32KB).
- Si la page n'est pas en mémoire, défaut de page.



Pagination

La Table de Pages

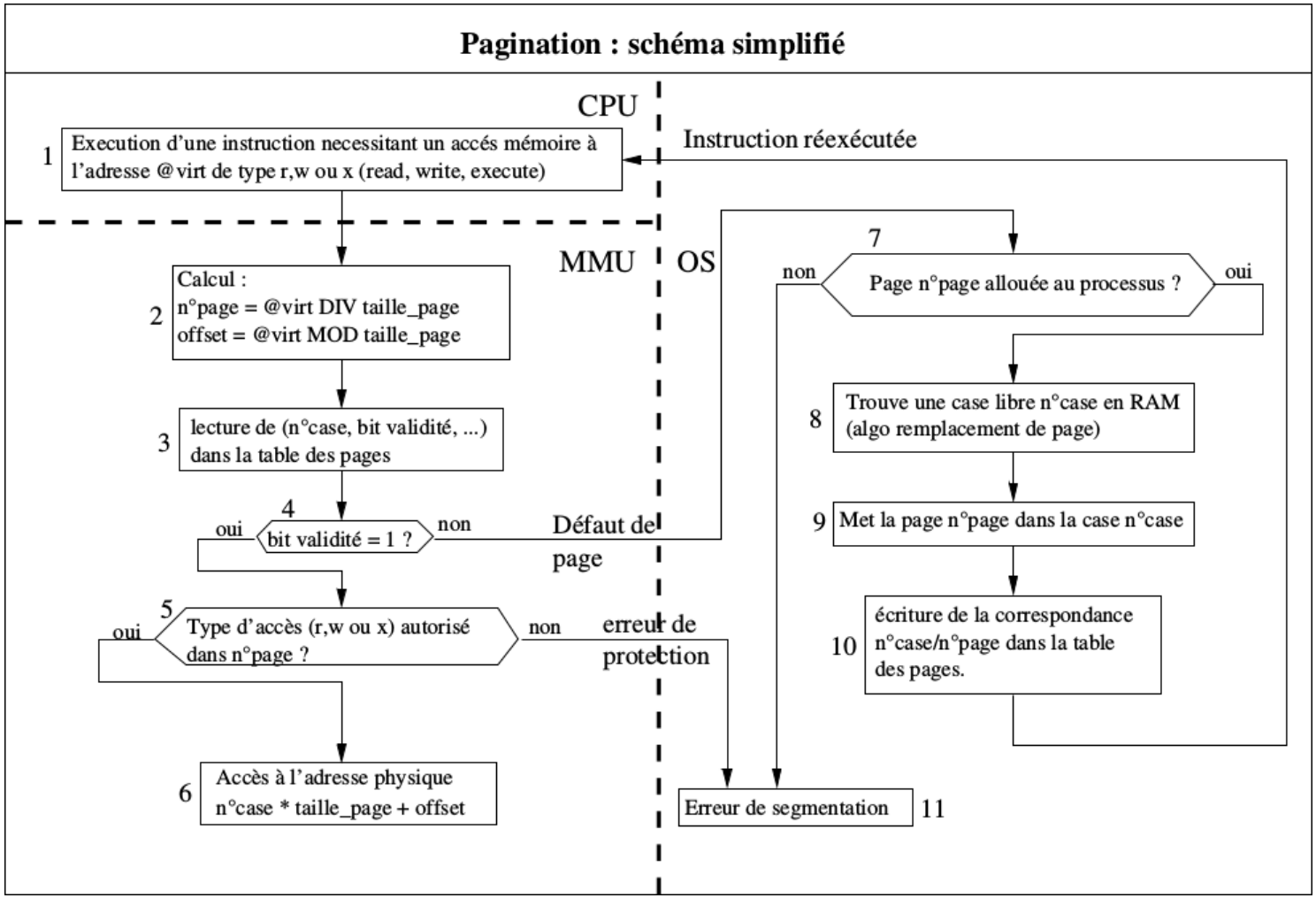
- "Dictionnaire" qui indique la correspondance :
  (processus,page) ⟷ cadre de page
- Maintenue par le SE en mémoire RAM (trop lourd pour être dans la MMU) → lent.
- Traduction de l'adresse virtuelle :
  - Numéro de page
  - Offset
- La Table de Pages retourne le cadre de page
  - Si bit de présence/absence (*Present/absent bit*) = 0 → défaut de page.



Pagination

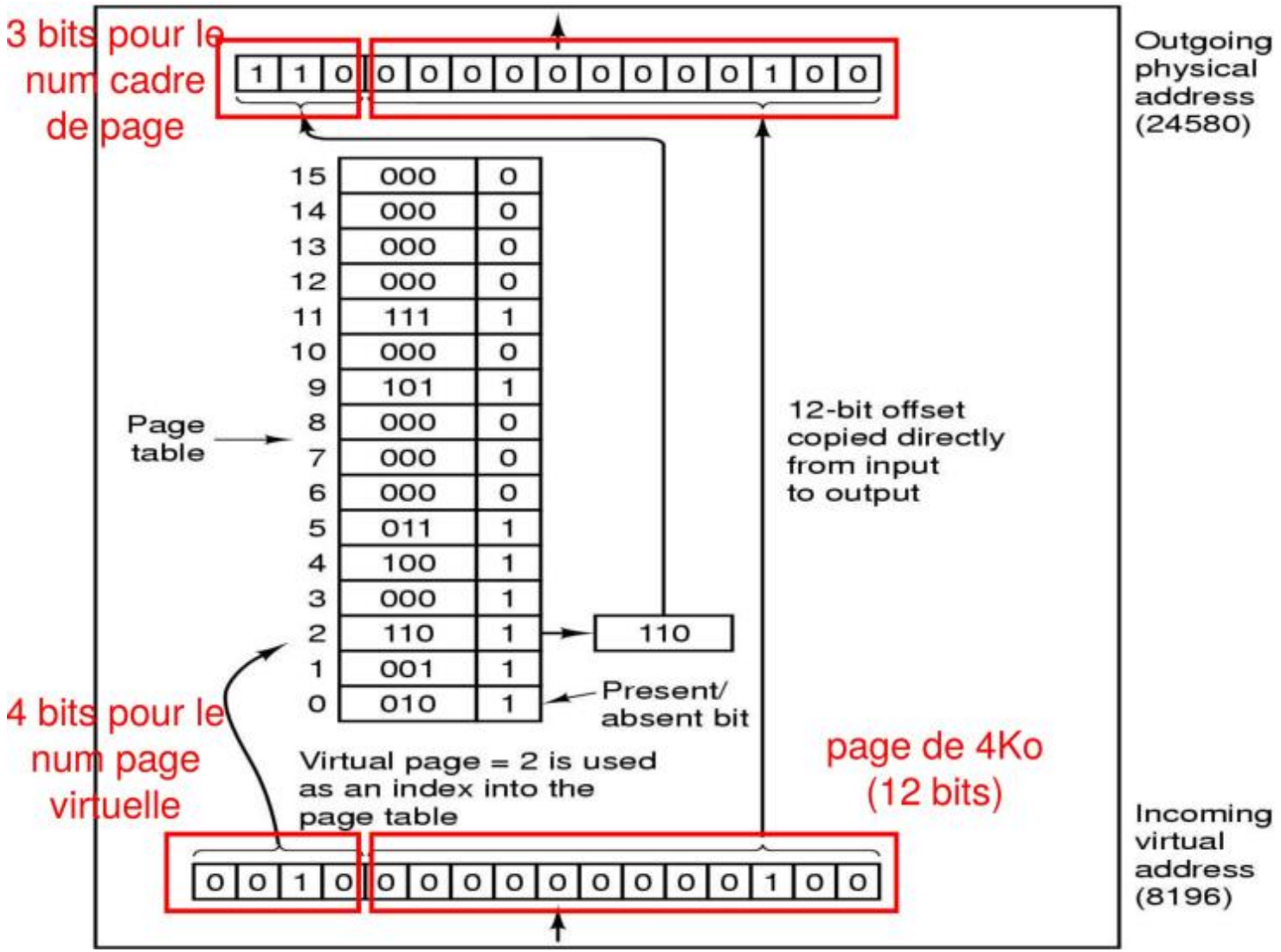
Algorithme de traduction (fait par la MMU)

- Étapes 2-3-6



Pagination

Exemple de traduction



Pagination

Défaut de page

- Étape 4 de l'algorithme de traduction.
- Si le bit de validité (V) est 1, c'est que la page est en mémoire réelle. Il remplace alors le numéro de page virtuelle par le numéro de page réelle qui se trouve dans la rangée en question.
- Si le bit de validité est à 0, il n'y a pas de traduction car soit la page n'est pas allouée au processus, soit elle est sur le disque.
  - Déclenchement d'un "défaut de page".
  - Le processus en cours est interrompu et le SE reprend la main.
  - Le SE lit l'adresse disque de la page en question, charge la page en mémoire et inscrit dans la rangée correspondante de la Table des pages le numéro de page réelle où la page a été placée. Il met ensuite le bit V à 1.

Pagination : Algorithmes de remplacement de pages

En cas de défaut de page, les algorithmes de remplacement permettent de déterminer une case physique victime qui sera sauvée sur le disque (si nécessaire) et remplacée par la page virtuelle à laquelle on souhaite accéder.

- La victime sera récrite sur le disque seulement si elle a été changée depuis qu'elle a été amenée en mémoire principale. Sinon, sa copie sur disque est encore fidèle.
- Un bit de modification (*dirty bit*) sur chaque descripteur de page indique si la page a été changée.
- **Quelques algorithmes :**
  - Optimal ou de Belady
  - FIFO (*First In, First Out*)
  - NRU (*Not Recently Used*)
  - LRU (*Least Recently Used*)

Pagination : Algorithmes de remplacement de pages

Algorithme de Belady

- La victime est la page qui sera référencée le plus tard dans le futur.
- Optimal, mais impossible à mettre en œuvre.
- Il faut connaître à l'avance les accès.
- Utilisé pour comparer avec la performance d'autres algorithmes.

FIFO (*First In First Out*)

- Le SE mémorise dans une file FIFO la liste de pages en mémoire triées par ordre d'arrivée en mémoire.
- La victime est la page la plus anciennement chargée, c'est-a-dire, la page en tête de file.
- Rarement utilisé, car il induit souvent des défauts de pages.

Pagination : Algorithmes de remplacement de pages

NRU (*Not Recently Used*)

- Chaque page possède deux bits d'état (dans la Table de Pages)
  - R = 1 (M = 1) lorsqu'une page est référencée (R) ou modifiée (M).
  - Mise à jour dans chaque référence à mémoire.
  - Ils restent à 1 jusqu'à ce que le SE les efface. Le bit R est effacé périodiquement (à chaque interruption d'horloge).
- S'il existe des pages non référencées (R = 0) et non modifiée (M = 0) alors il sélectionne une page et la retire.
- Sinon, s'il existe des pages telles que R = 0 et M = 1, alors il sélectionne une page et la retire (et sauvegarde la page sur le disque).
- Sinon, s'il existe des pages telles que R = 1 et M = 0, alors il sélectionne une page et la retire.
- Sinon, s'il existe des pages telles que R = 1 et M = 1, alors il sélectionne une page et la retire (et sauvegarde la page sur le disque).
- Implémentation simple.

Pagination : Algorithmes de remplacement de pages

LRU (*Least Recently Used*)

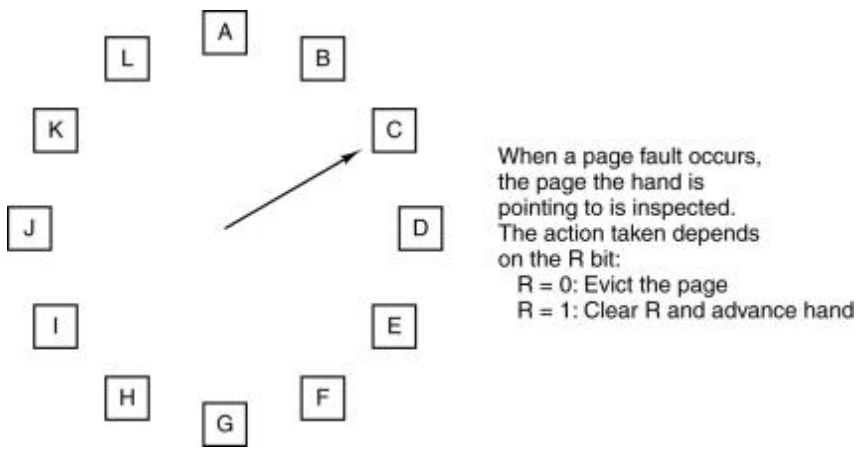
- Localité temporelle
  - Les pages utilisées dans les dernières instructions seront réutilisées dans les instructions suivantes.
  - Les pages qui n'ont pas été utilisées depuis longtemps, continueront sans être utilisées.
- La victime est la page la moins utilisée.
- Implémentation : Utilisation d'une liste chaînée de toutes les pages en mémoire
  - La plus utilisée est en tête de liste et la moins utilisée est en queue.
- Coûteux et difficile à implémenter : il faut garder une liste de pages triée par utilisation.

Pagination : Algorithmes de remplacement de pages

Approximation de LRU : l'algorithme de l'horloge

(ou de la seconde chance)

- Liste circulaire en forme d'horloge avec un pointeur sur la plus vieille page
  - Lorsqu'il y a un défaut de page, la page pointée est examinée
    - Si R = 0 (le bit d'accès de la page est à 0) alors elle est retirée, la nouvelle page est insérée à sa place (et son bit R est mis à 1) et le pointeur avance.
    - Sinon, si R = 1, il est mis à 0 (R = 0) et le pointeur avance pour chercher une autre victime.
- Lorsqu'on accède à page présente dans la liste, son bit de référence R est mis à 1.
- Simple d'implémenter.



Segmentation

- Chaque processus peut avoir plusieurs segments (code, données, pile, etc.).
- Chaque segment commence à une nouvelle adresse de base dans la mémoire physique.

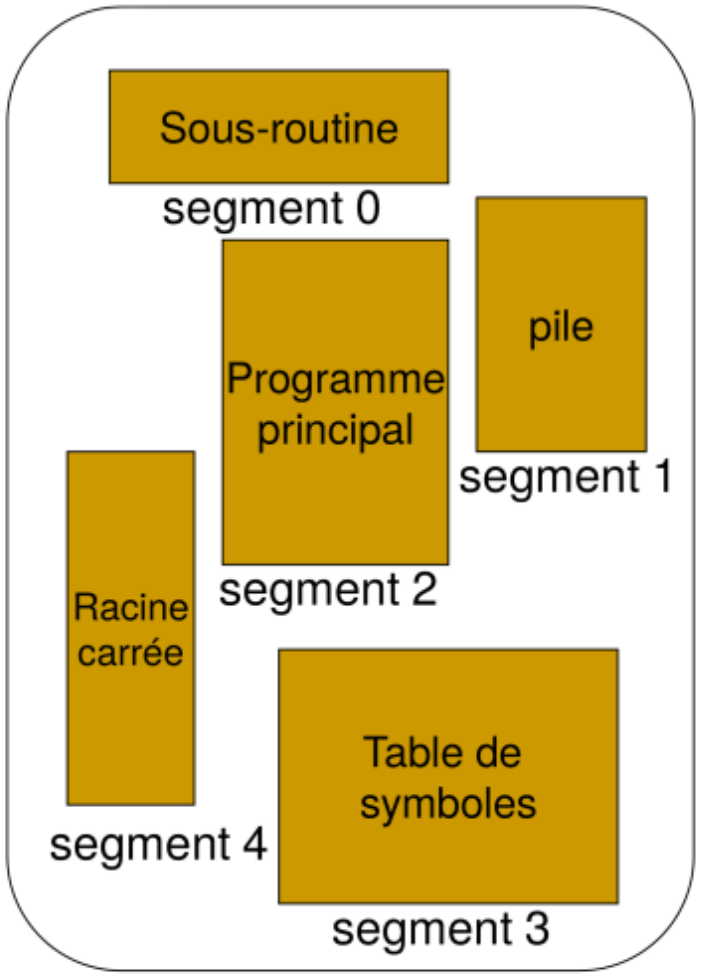
Raison d'être :

- Pour permettre aux programmes et aux données d'être divisés en espaces d'adressage indépendants, et pour faciliter le partage et la protection.

Avantages :

- Les segments peuvent grandir indépendamment.
- Les segments peuvent swapper indépendamment.
- Les segments peuvent être partagés entre processus.

Segmentation



	base	limite
0	1400	1000
1	6300	400
2	4300	400
3	3200	1100
4	4700	1000