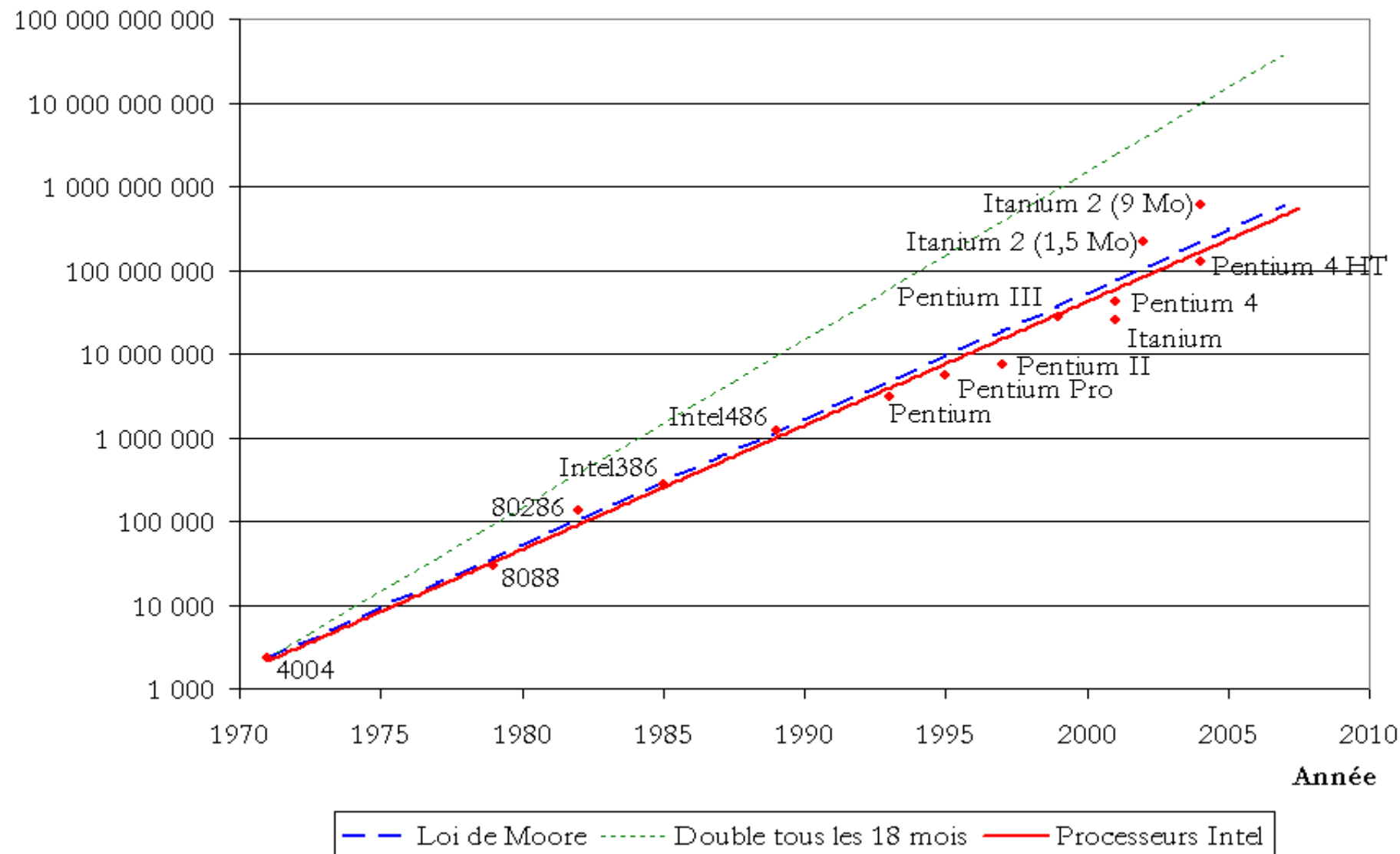


Gestion de la Mémoire

- **Mémoire** : organe d'un ordinateur permettant d'enregistrer, de stocker et de restituer des données.
- La mémoire est une ressource importante qui doit être gérée avec attention.
- La quantité de mémoire d'un ordinateur ne cesse d'augmenter. Mais la taille des programmes s'accroît aussi.
 - ❑ **Loi de Moore**: *les capacités de stockage doublent en taille tous les 18 mois.*
 - ❑ **Loi de Parkinson**: *les programmes grossissent en taille aussi vite que la mémoire.*

Gestion de la Mémoire

Nombre de
transistors



Gestion de la Mémoire

- La nécessité de gérer la mémoire de manière **optimale est toujours d'actualité.**
- Malgré des capacités de plus en plus grandes et des coûts moindres. Elle n'est en général, jamais suffisante, ceci en raison de la **taille continuellement grandissante des programmes.**
 - On aura toujours besoin de gestionnaires de mémoires performants

Gestion de la Mémoire

- Situation idéale (souhaits des programmeurs et utilisateurs)
donner à chaque processus une mémoire :
 - ✓ infiniment grande,
 - ✓ infiniment rapide,
 - ✓ non volatile,
 - ✓ ayant une consommation énergétique réduite,
 - ✓ Moins chère.

Gestion de la Mémoire

- Malheureusement, ces critères sont mutuellement incompatibles. Il n'existe pas encore de mémoire possédant simultanément toutes ces caractéristiques. Les mémoires de grande capacité sont lentes, celles qui sont rapides sont chères.
- Frein technologique → hiérarchie de la mémoire

Gestion de la Mémoire

Il existe trois types de mémoires :

- **Mémoire morte (*mémoire non volatile*)**
 - ✓ mémoire ROM (Read-Only Memory)
 - ✓ mémoire ne s'effaçant pas en absence de courant électrique
 - ✓ mémoire conservant les données nécessaires au démarrage de l'ordinateur
 - ✓ temps d'accès de l'ordre de 150 ns

Gestion de la Mémoire

- **Mémoire vive (*mémoire volatile*)**
 - ✓ mémoire RAM (Random Access Module)
 - ✓ données ne perdurant pas en l'absence de courant électrique
 - ✓ deux types de mémoire RAM : DRAM et SRAM
 - temps d'accès pour la DRAM de l'ordre de 50 ns
 - temps d'accès pour la SRAM de l'ordre de 10 ns

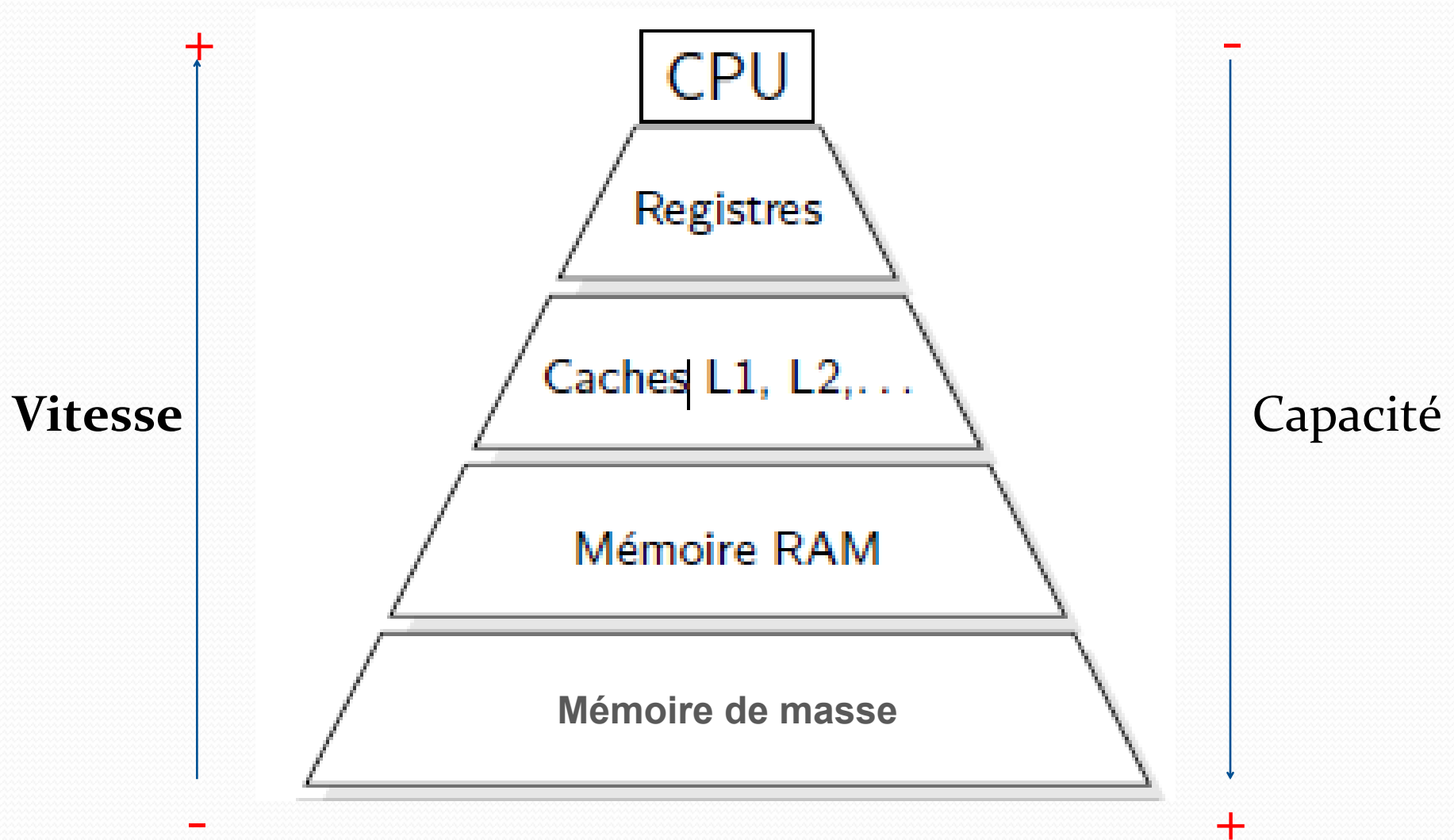
Gestion de la Mémoire

- **Mémoire flash**
 - ✓ compromis entre la mémoire RAM et la mémoire ROM
 - ✓ non volatilité de la mémoire morte
 - ✓ accès en lecture/écriture de la mémoire vive

Gestion de la Mémoire

- Deux niveaux de gestion de la mémoire :
 - **Niveau matériel**
 - ☞ les registres du processeur
 - ☞ la mémoire cache
 - **Niveau système d'exploitation**
 - ☞ la mémoire principale (appelée également mémoire centrale ou interne)
 - ☞ la mémoire secondaire (appelée également mémoire de masse ou physique)
- **Rôle du gestionnaire de mémoire du SE**

Hiérarchie de la Mémoire



Gestion de la Mémoire

- **Les registres** sont les composants de mémoire très performants et les plus rapides.
- D'autre part, les registres servent d'emplacement de mémoire interne à un processeur et se situent en tête de la hiérarchie mémoire : il s'agit de la mémoire avec une performance du temps d'accès, avec un coût de fabrication plus élevé car son emplacement dans un microprocesseur est limitée et son nombre dépasse donc rarement quelques dizaines d'octets.

Gestion de la Mémoire

- **la mémoire cache** est une mémoire au temps d'accès rapide avec une faible capacité destinée à augmenter l'accès à la mémoire centrale pour ensuite stocker les éléments utilisés fréquemment par le microprocesseur.
- La mémoire cache est un type de mémoire à laquelle le microprocesseur peut accéder plus rapidement qu'à la mémoire RAM habituelle. Généralement, elle est directement intégrée dans la puce de l'unité centrale (UC) ou placée sur une puce distincte dotée d'une interconnexion par bus à l'UC.

Gestion de la Mémoire

- La fonction de base de la mémoire cache est de stocker les instructions de programme qui sont fréquemment référencées par les logiciels en cours d'exécution. L'accès rapide à ces instructions accroît la vitesse globale des logiciels.
- La mémoire cache est rapide et coûteuse. Traditionnellement, elle est classée « par niveaux » en fonction de sa proximité avec le microprocesseur et de son degré d'accessibilité à ce dernier :
- **La mémoire cache de niveau 1 (L1)** est extrêmement rapide mais relativement petite, et généralement incorporée dans le processeur.
- **La mémoire cache de niveau 2 (L2)** a souvent une plus grande capacité que la L1 ; elle peut être placée sur l'UC, sur une puce distincte ou sur un coprocesseur équipé d'un autre bus système à grande vitesse assurant son interconnexion avec l'UC afin qu'elle ne soit pas ralentie par le trafic du bus

Gestion de la Mémoire

Objectifs du gestionnaire de mémoire du système d'exploitation :

- Partager la mémoire
- Allouer des blocs de mémoire aux différents processus
- Protéger les espaces mémoire utilisés
- Optimiser la quantité de mémoire disponible



Mécanisme de mémoire virtuelle

Mécanismes de découpage de la mémoire

Gestion de la Mémoire

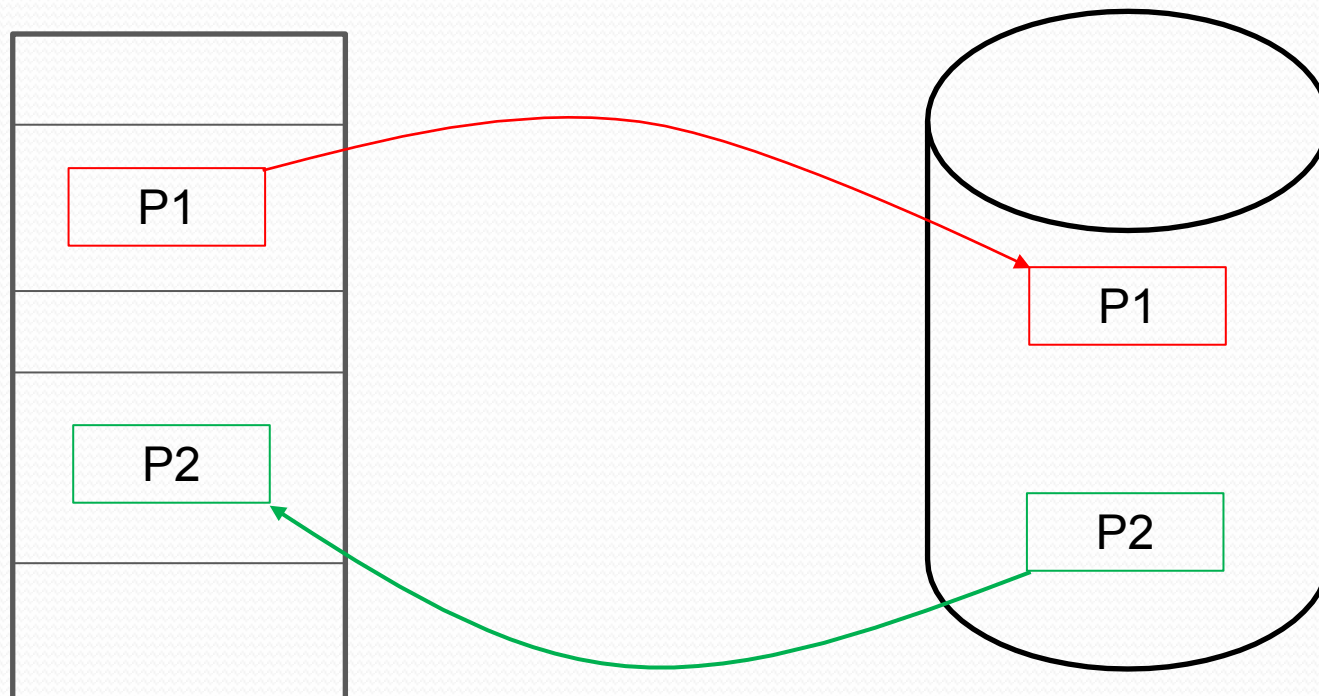
- **Mémoire virtuelle**
- La mémoire virtuelle est une *fonctionnalité* d'un système d'exploitation qui permet à un ordinateur de *compenser* le manque de mémoire physique en transférant temporairement des pages de données de la mémoire vive (RAM, Random Access Memory) vers un stockage sur disque.
- Pour un processus nécessitant une taille de 1 Go, si la mémoire centrale physique possède une taille de 512 Mo.

Gestion de la Mémoire

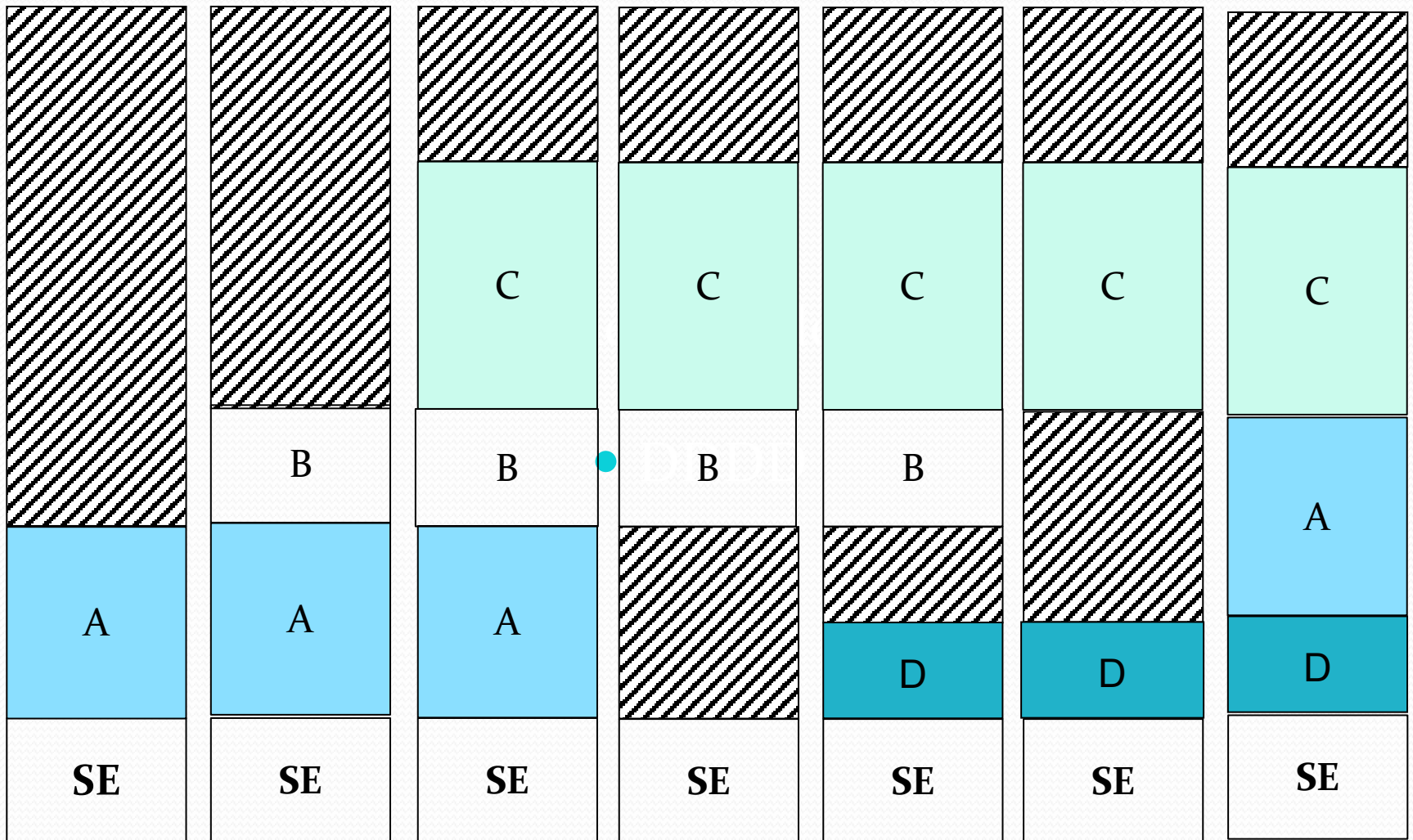
- le mécanisme de **mémoire virtuelle** permet de ne mettre à un instant donné dans les 512 Mo de la MC, que les éléments strictement nécessaires à l'exécution du processus, les autres éléments restant stockés sur le disque dur, prêts à être ramenés en MC à la demande.
- Une partie de l'espace d'adressage d'un processus peut être enlevé temporairement de la mémoire centrale au profit d'un autre (**swapping**)

Le swap (ou va-et-vient)

- Le swap est mis en œuvre lorsque tous les processus ne peuvent pas tenir simultanément en mémoire.
- On doit alors en déplacer temporairement certains sur disque dans une zone de swap.



Va-et-vient



Gestion de la mémoire dans le SE

Le mécanisme de gestion de la mémoire par le SE a pour objectifs :

- l'organisation de la mémoire en zones ,
- l'allocation, libération des zones
- la protection des processus
- la simulation d'une mémoire de grande taille (mémoire virtuelle sur disque)
- la transparence des accès (quelque soit le type de mémoire physique ou virtuelle)

Organisation de la mémoire

On peut voir la mémoire comme un tableau à une dimension.

- Elle est divisée en zones de taille fixe ou variable
- Une zone de taille N de la mémoire est un sous-ensemble des N mots (ayant des adresses consécutives)
- Le SE doit garantir l'intégrité de ces zones en interdisant les accès non autorisés
- Le contenu de certaines zones doit pouvoir être partagé

Organisation de la mémoire - Multiprogrammation

La plupart des systèmes modernes autorisent l'exécution de processus multiples en même temps.

- ☞ séjour de plusieurs programmes en même temps en mémoire
- ☞ naissance à la gestion moderne de la mémoire.

Pour supporter la multiprogrammation (l'existence de plusieurs processus) en mémoire principale, on peut distinguer deux grandes stratégies d'allocation mémoire:

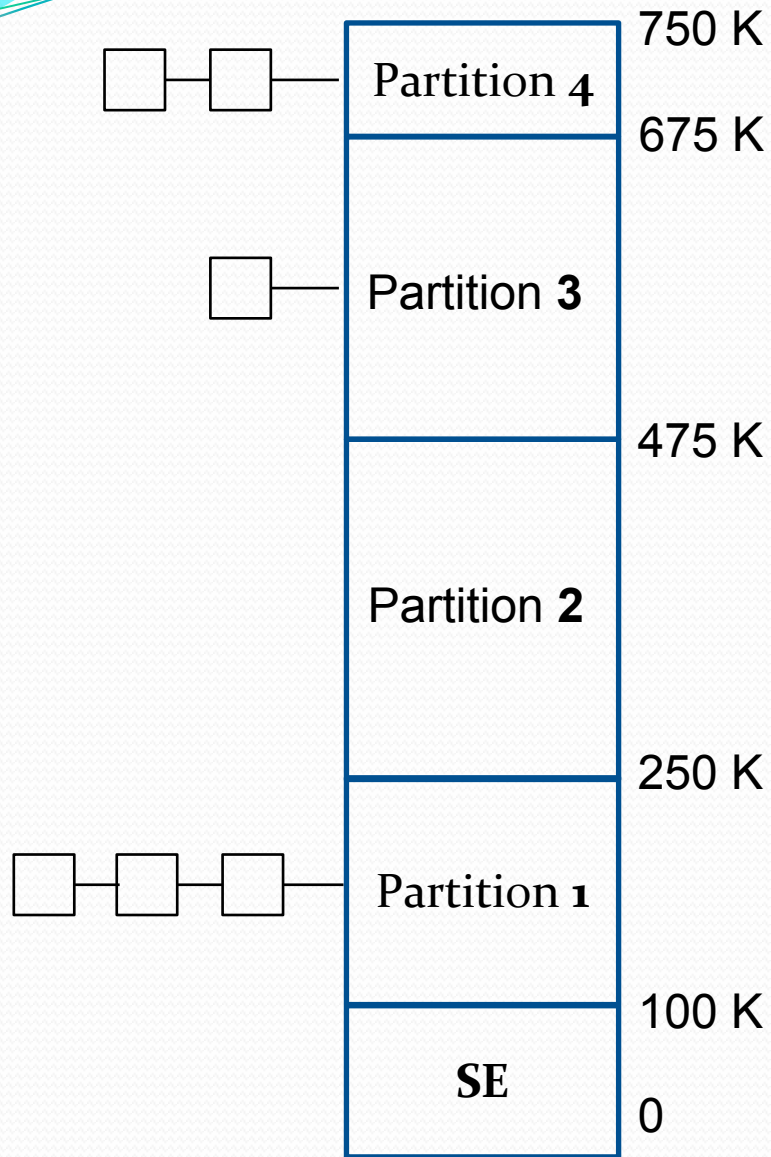
- allocation de partitions contiguës
- allocation de partitions non-contiguës.

Multiprogrammation et partitions multiples contiguës

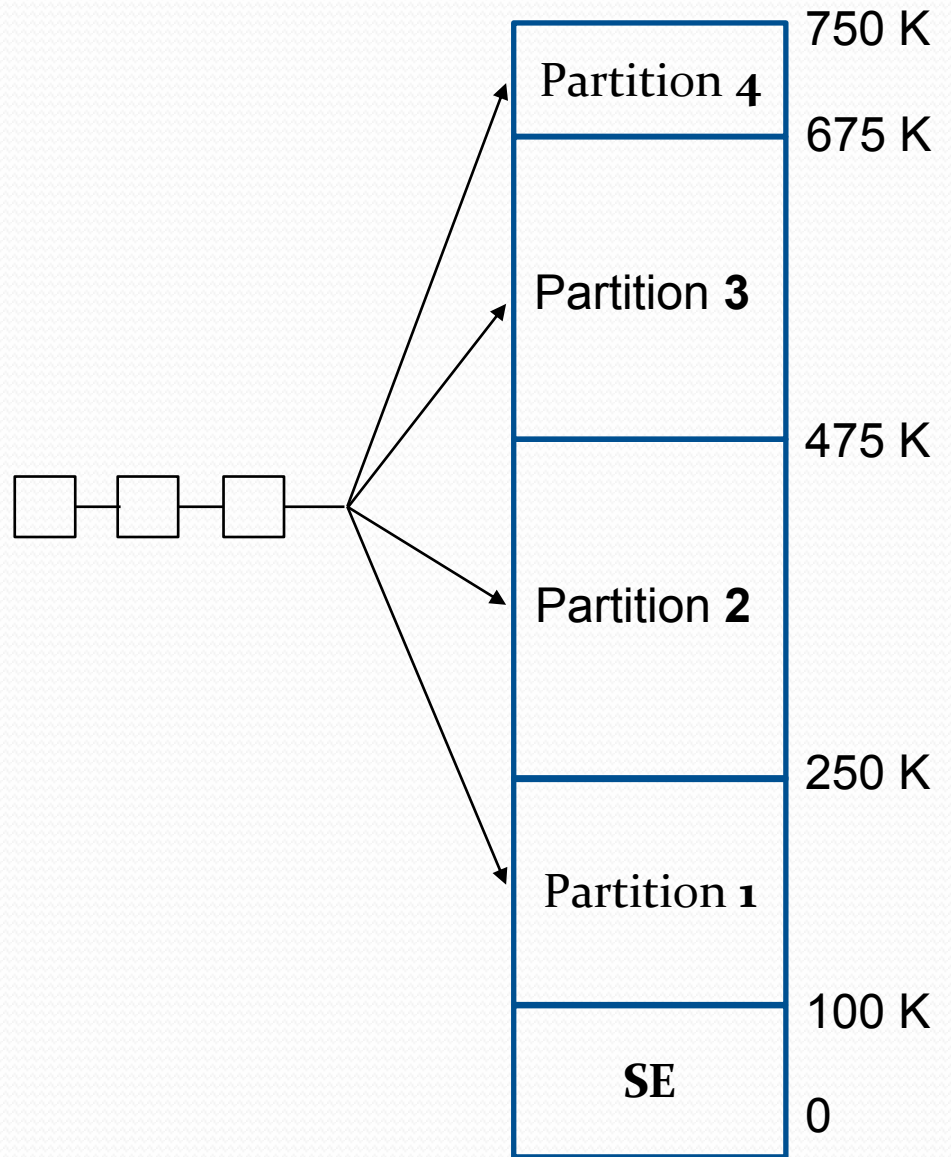
L'espace mémoire est divisé en partitions. Chaque partition peut être allouée à un programme. La taille des partitions et donc leur nombre peuvent être fixe ou variable.

- **Partitions contiguës fixes** : dans cette stratégie la mémoire est divisée en n partitions de tailles fixes. Ce partitionnement se fait au démarrage du système. Le système d'exploitation maintient une table de description des partitions.
 - files d'attente séparées : quand un processus arrive, il peut être placé dans la file d'attente des entrées de la plus petite des partitions assez larges pour le contenir.

Partitions contigües fixes



(a) files d'attente multiples



(b) file d'attente commune

Multiprogrammation et partitions multiples contigües

Inconvénients :

Possibilités d'avoir une file d'attente vide pour une grande partition
longue file d'attente pour une petite partition

Partitions contigües fixes : file d'attente commune

➤ files d'attente commune (b):

Une autre solution est de créer une seule file d'attente.

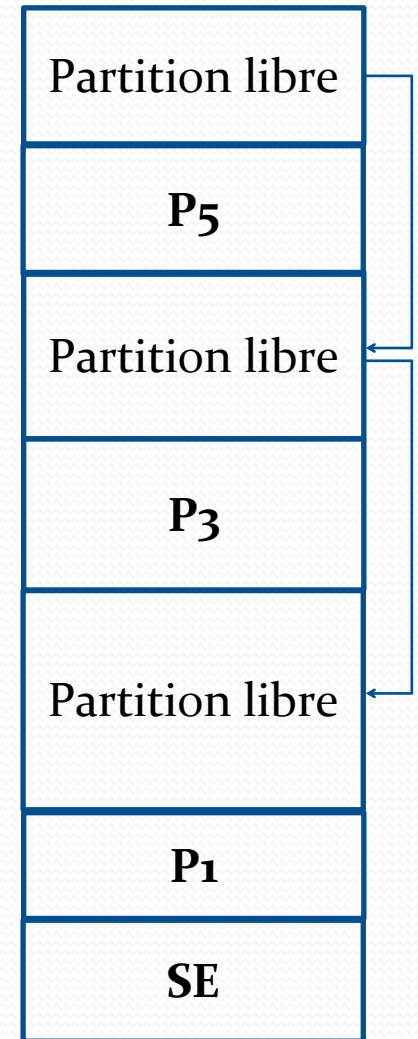
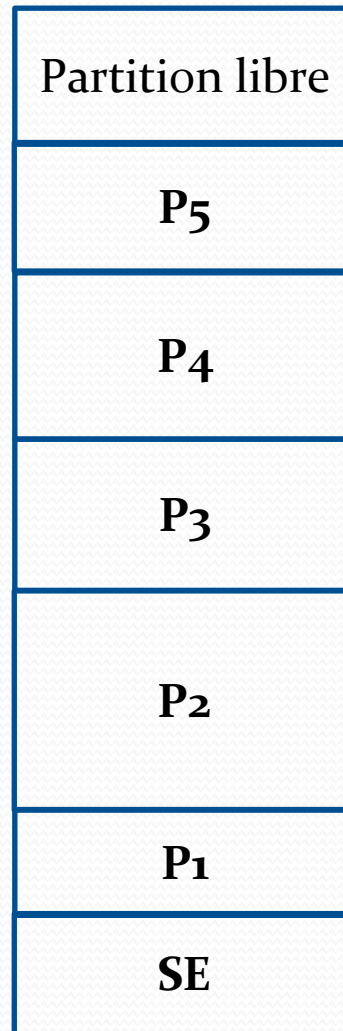
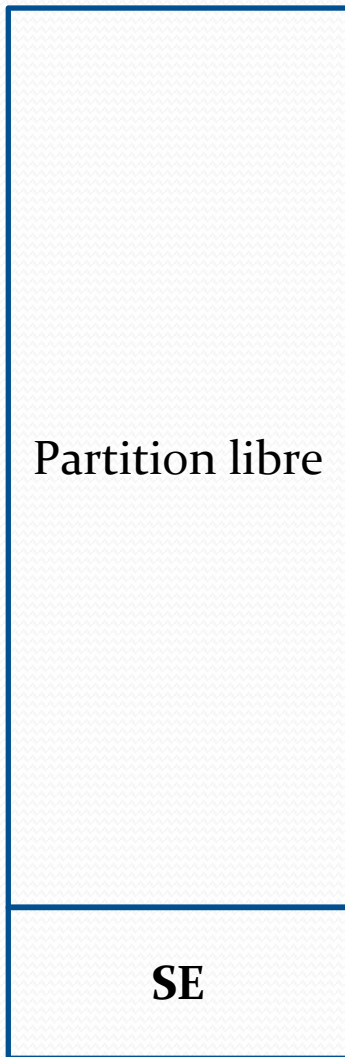
Lorsqu'une partition se libère, on consulte la file pour trouver la tâche qui l'occuperait de manière optimale.

Partitions dynamiques

Dans cette stratégie la mémoire est partitionnée dynamiquement selon la demande. Lorsqu'un processus se termine sa partition est récupérée pour être réutilisée (complètement ou partiellement) par d'autres processus.

- Au fur et à mesure des chargements de programmes, la zone libre va se réduire et à l'instant t , elle n'occupe plus qu'une fraction de la mémoire centrale.
- Lorsque l'exécution des programmes se termine, la mémoire est libérée : il se crée alors pour chaque zone libérée, une nouvelle zone libre. Finalement, la mémoire centrale se retrouve constituée d'un ensemble de zones allouées et de zones libres réparties dans toute la mémoire. Les zones libres sont organisées en une liste chaînée de zones libres repérée par une tête de liste.

Partitions dynamiques



Stratégies d'allocation

Le chargement d'un nouveau processus dans les partitions consiste à trouver une zone libre suffisamment grande pour pouvoir y placer le programme et il se fait selon différentes stratégies.

Le gestionnaire de la mémoire doit avoir une liste des partitions occupées et des partitions libres.

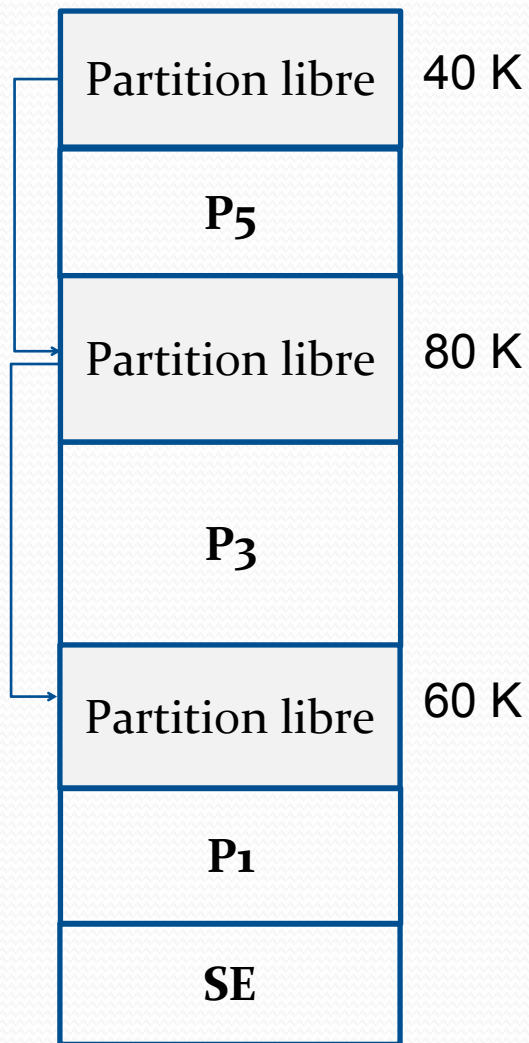
On distingue les stratégies de placement suivantes:

- Stratégie du premier qui convient (First Fit): La liste des partitions libres est triée par ordre des adresses croissantes. La recherche commence par la partition libre de plus basse adresse et continue jusqu'à la rencontre de la première partition dont la taille est au moins égale à celle du processus en attente.

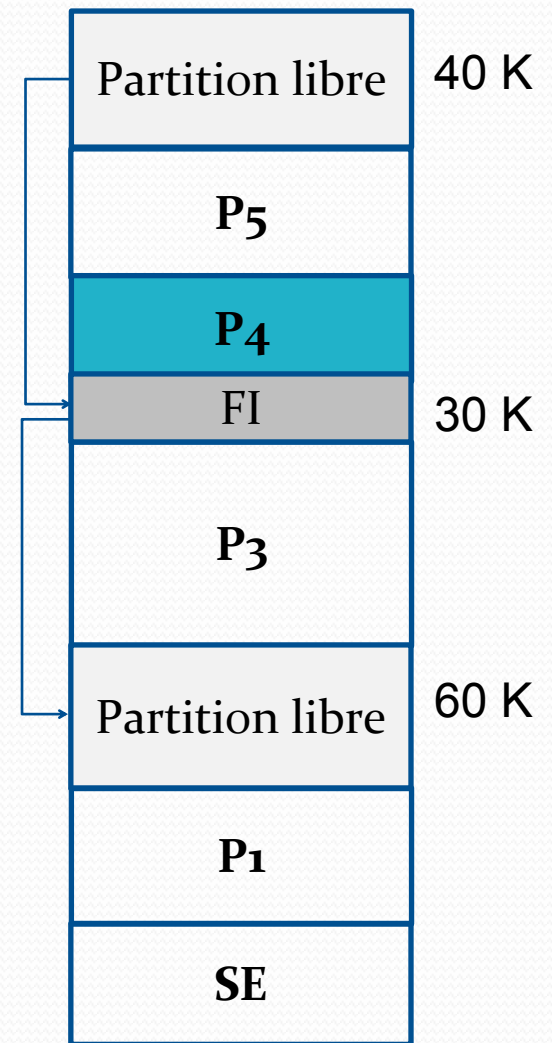
Stratégies d'allocation

- Stratégie du meilleur qui convient (Best Fit): On alloue la plus petite partition dont la taille est au moins égale à celle du processus en attente. La table des partitions libres est de préférence triée par tailles croissantes.
- Stratégie du pire qui convient (Worst Fit): On alloue au processus la partition de plus grande taille.

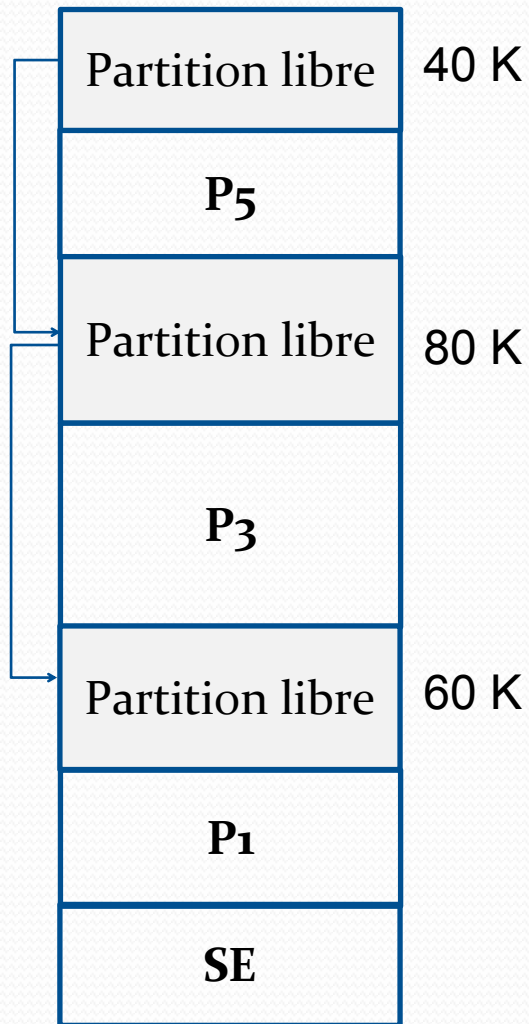
Stratégies d'allocation – First Fit



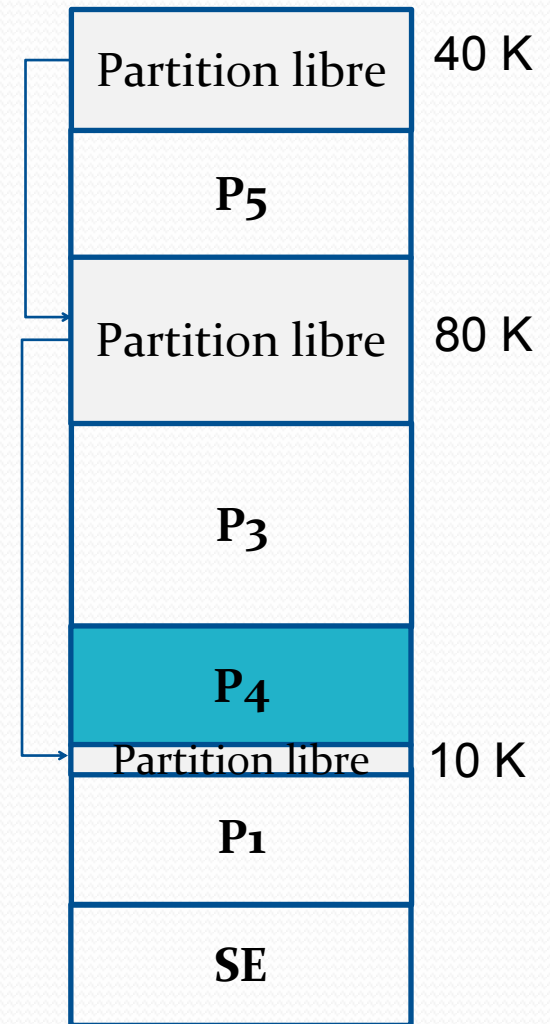
P₄ (50 K) ?



Stratégies d'allocation – Best Fit



P₄ (50 K) ?



Comparaison des algorithmes

Même si Best Fit semble le meilleur, ce n'est pas toujours l'algorithme retenu en pratique car il demande beaucoup de temps de calcul.

First Fit, malgré sa simplicité apparente, est souvent utilisé.

Dans le cas d'une liste de partitions libres triée par taille, le Best-fit et le First-fit sont aussi rapides l'un que l'autre.

Comparaison des algorithmes

Au fur et à mesure des opérations d'allocation et de libération, la mémoire centrale devient composée d'un ensemble de zones occupées et de zones libres éparpillées dans toute l'étendue de la mémoire centrale.

Ces zones libres peuvent devenir trop petites pour permettre l'allocation de nouveaux processus (problème de fragmentation de la mémoire). Pour permettre l'allocation d'un processus, il faut réunir l'ensemble des zones libres pour ne former plus qu'une zone libre suffisante : c'est l'opération de compactage de la mémoire centrale

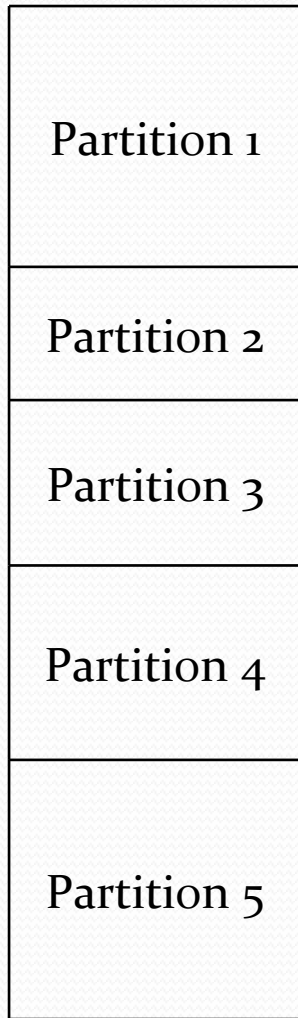
Fragmentation et Compactage

- Une mémoire fragmentée est une mémoire dans laquelle plusieurs blocs de mémoire non contigus sont libres.
- L'allocation contiguë de partitions à taille fixe crée de la *fragmentation interne*. Entre chaque partition de taille fixe, un peu de mémoire est perdue parce que le programme contenu dans la partition n'a pas nécessairement la même taille que la partition.
- L'allocation contiguë de partitions à taille variable crée de la *fragmentation externe*. Lorsqu'un programme est retiré de la mémoire, il laisse un bloc de mémoire libre.

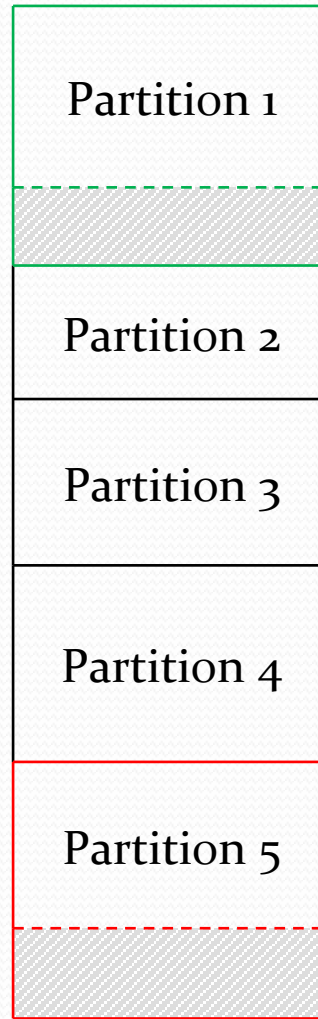
Fragmentation et Compactage

- Il est possible, par la suite, que ce bloc soit rempli partiellement par un processus de taille moindre (dans une nouvelle partition). Il reste alors de la mémoire libre à l'extérieur des partitions.
- Une mémoire très fragmentée est une mémoire lente et une mémoire dans laquelle des blocs (programmes) de grandes dimensions ne peuvent plus être alloués, car l'espace libre est répartie partout dans la mémoire.
- Il existe des méthodes de compaction (pour la mémoire) et défragmentation (pour un disque dur) afin de réduire la fragmentation.

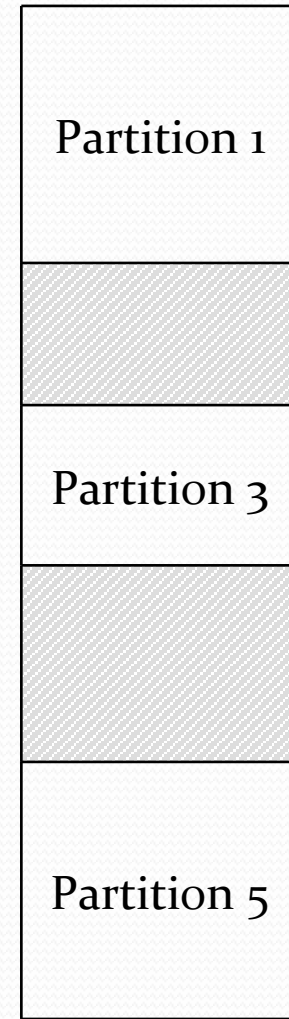
Fragmentation externe Vs Fragmentation interne



Initial



Fragmentation interne



Fragmentation externe

Fragmentation et Compactage

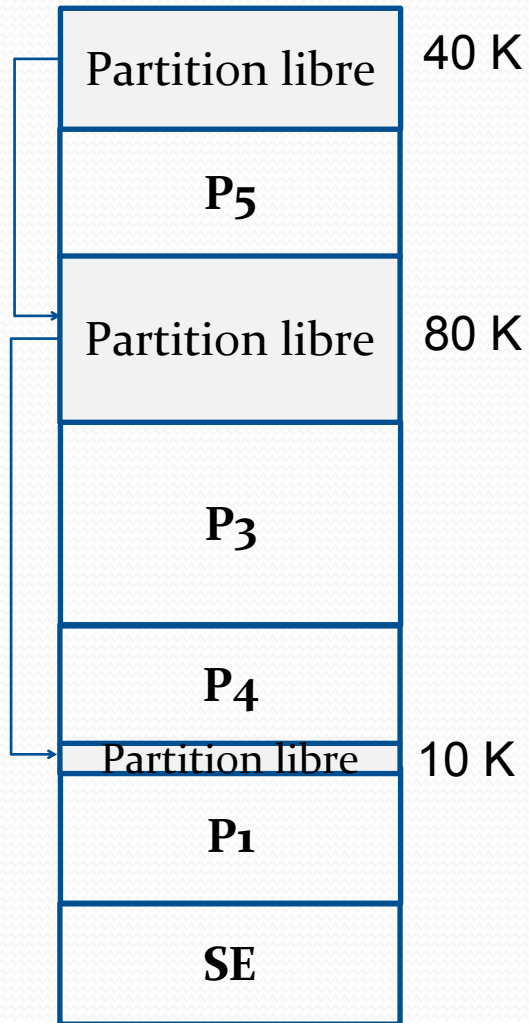
Compactage de la mémoire centrale

- Le compactage de la mémoire centrale consiste à déplacer les programmes en mémoire centrale de manière à ne créer qu'une seule et unique zone libre.
- Le compactage de la mémoire centrale est une opération coûteuse.
- Il nécessite une translation des adresses dynamiques. Dans le mécanisme de chargement dynamique, les adresses du programme chargé en mémoire centrale ne sont pas traduites de la valeur de l'adresse d'implantation au moment du chargement, mais seulement au moment de l'exécution.

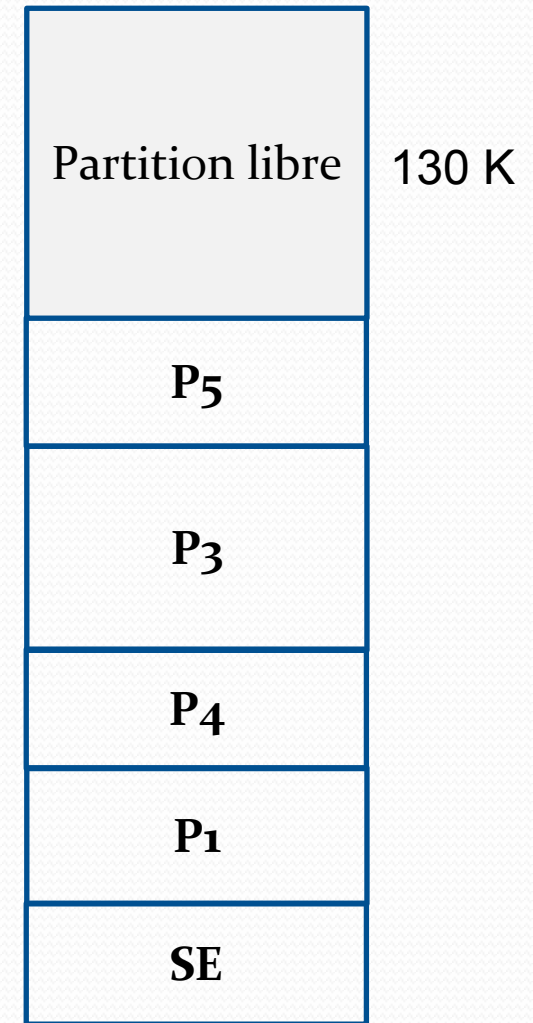
Fragmentation et Compactage

- L'adresse d'implantation du programme est conservée dans un registre processeur - le registre de translation –
- Ainsi lors de l'opération de compactage, déplacer un programme consiste à changer la valeur d'adresse d'implantation à charger dans le registre de translation.

Compactage de la mémoire



P₂ (100 K) ?



Conclusion

- Le partitionnement de la mémoire que ce soit avec des partitions de tailles fixes ou de tailles variables, ne permet pas d'utiliser la mémoire au mieux.
- L'allocation contigu de la mémoire centrale souffre donc de deux défauts principaux :
- Elle nécessite une opération de compactage de la mémoire qui est une opération très coûteuse
- Elle exige d'allouer le programme en une zone d'un seul tenant.
- Une solution est de diviser le programme en portions de taille fixe et égales à l'unité d'allocation de la mémoire centrale. On dit alors que le programme est découpé en pages. Le mécanisme d'allocation associé s'appelle la ***pagination***.

Allocation non contigüe

Il y a deux techniques de base pour faire ceci: la ***pagination*** et la ***segmentation***.

- La **segmentation** utilise des parties de programme qui ont une valeur logique (des modules)
- La **pagination** utilise des parties de programme arbitraires (morcellement du programmes en pages de longueur fixe).
- La combinaison des deux (**segmentation paginée**)

Pagination

La technique de pagination ou paging consiste à organiser la mémoire en zones de taille fixes ou identiques appelées *cadres* de page ou en anglais «page *frame*».

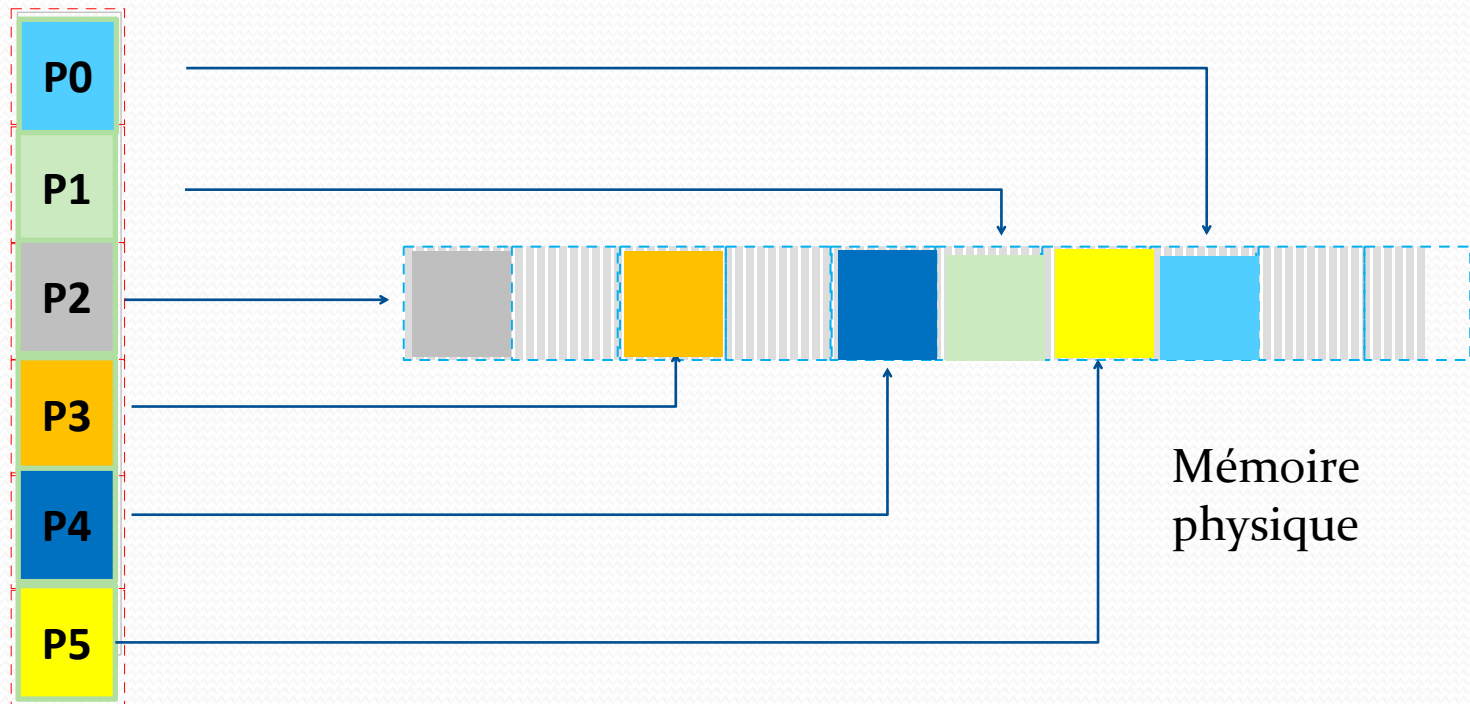
Principe :

- L'espace d'adressage virtuel est divisé en petites unités de taille appelées *pages*
- Les unités correspondantes dans la mémoire physique sont appelées *cadres* de pages (pages frames)
- Les *pages* et les *cadres* de pages ont toujours la **même taille**

Pagination

- Chaque page est indivisible: Elle est entièrement en mémoire physique ou elle n'y est pas du tout.
- Il peut y avoir plus de pages que de cadres (c'est la tout l'intérêt)
- Il n'est plus nécessaire de placer le processus dans une zone contiguë de mémoire.
- Il devient possible d'allouer de la mémoire à un processus sans forcément avoir à réaliser de compactage.

Pagination



Allocation processus

Pagination

- Les pages logiques d'un processus peuvent donc être assignés aux cadres disponibles n'importe où en mémoire principale
- Conséquences:
 - ✓ un processus peut être éparpillé n'importe où dans la mémoire physique.
 - ✓ la fragmentation externe est éliminée
 - ✓ Seule la dernière page d'un processus peut souffrir de fragmentation interne (qui peut être plus petite qu'un cadre)

Allocation de mémoire - pagination

0		0	P1_0	0	P1_0	0		0	P4_0
1		1	P1_1	1	P1_1	1		1	P4_1
2		2	P1_2	2	P1_2	2		2	P4_2
3		3	P1_3	3	P1_3	3		3	P4_3
4		4		4	P2_0	4	P2_0	4	P2_0
5		5		5	P2_1	5	P2_1	5	P2_1
6		6		6	P2_2	6	P2_2	6	P2_2
7		7		7	P2_3	7	P2_3	7	P2_3
8		8		8	P2_4	8	P2_4	8	P2_4
9		9		9	P3_0	9	P3_0	9	P3_0
10		10		10	P3_1	10	P3_1	10	P3_1
11		11		11		11		11	P4_4
12		12		12		12		12	P4_5
13		13		13		13		13	
14		14		14		14		14	
Mémoire principal		P1 : IN		P1,P2,P3 : IN		P1 OUT - P2,P3 : IN		P1 ,P2,P3 ,P4 : IN	

Pagination

Même s'il ne reste pas d'espace continu suffisant, un processus peut être chargé sans défragmentation de la mémoire.

- Il n'est plus nécessaire de placer le processus dans une zone contiguë de mémoire.
- Il devient possible d'allouer de la mémoire à un processus sans forcément avoir à réaliser de compactage.

Problème

- Taille d'un processus $>$ taille mémoire principale
- Taille de l'ensemble des processus $>$ taille mémoire principale

Pagination

Même s'il ne reste pas d'espace continu suffisant, un processus peut être chargé sans défragmentation de la mémoire.

- Il n'est plus nécessaire de placer le processus dans une zone contiguë de mémoire.
- Il devient possible d'allouer de la mémoire à un processus sans forcément avoir à réaliser de compactage.

Problème

- Taille d'un processus $>$ taille mémoire principale
- Taille de l'ensemble des processus $>$ taille mémoire principale

Mémoire virtuelle

Motivations :

- Attribuer à chaque processus un espace d'adressage complet revient trop cher (mémoire physique coûteuse)
- Mémoire secondaire (disques, mémoire étendue, ...) peu coûteuse

Idée :

- Utiliser la mémoire secondaire "comme" mémoire RAM.
- Il s'agit de conserver en mémoire une « partie » des processus

Mémoire virtuelle

- Si un programme P veut s'exécuter alors qu'il n'y a plus de place en mémoire, un morceau d'un autre programme est déplacé en mémoire secondaire et remplacé par un morceau de P.

Définition :

- La mémoire virtuelle est une fonctionnalité d'un système d'exploitation permettant de compenser le manque de mémoire physique en transférant temporairement des pages de données de la RAM vers le disque.

- La mémoire virtuelle c'est la capacité de donner l'illusion à chaque processus d'avoir accès à toute la mémoire de l'ordinateur pour lui tout seul.

Mémoire virtuelle

- L'astuce consiste en fait à swapper les pages inutiles afin de ne conserver en mémoire vive que celles dont on a forcément besoin.
- Le principe consiste à conserver en mémoire centrale les parties utilisées des processus et stocker, si nécessaire, le reste sur disque.
- La notion de mémoire virtuelle et de multiprogrammation se complètent: un processus en attente d'une ressource n'est plus conservé en mémoire centrale, si cela s'avère nécessaire. La mémoire virtuelle fait appel aux mécanismes de la pagination et segmentation.

Mémoire virtuelle

Le concept de mémoire virtuelle utilise deux types d'adressages de l'espace :

- *Adresse virtuelle (logique)* est une adresse utilisée à l'intérieur d'un programme accessible par l'utilisateur. Elle est générée par le CPU.
- *Adresse physique (réelle)* est une adresse réelle dans la mémoire principale, vue par l'unité de mémoire.

Mémoire virtuelle

Le concept de mémoire virtuelle pose de nouveaux problèmes :

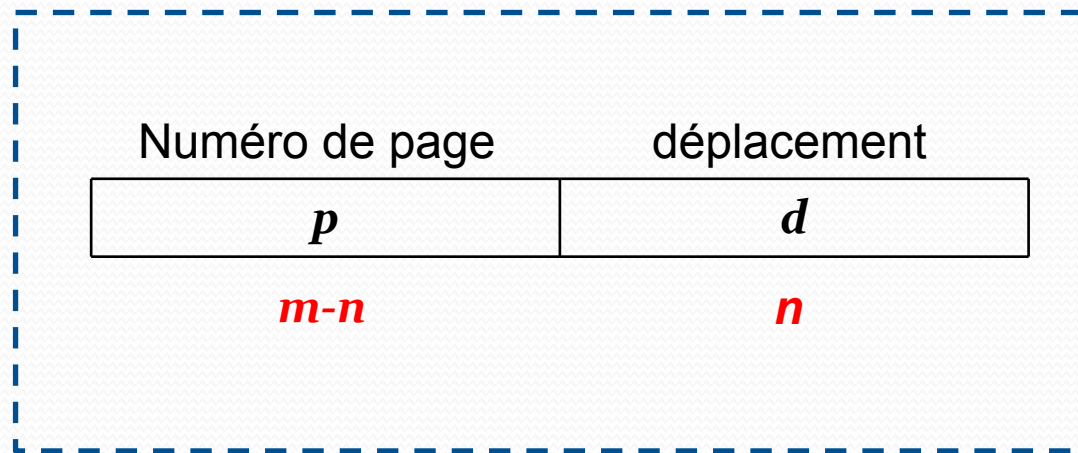
- Comment les adresses virtuelles sont elles converties en adresses physiques ?
- Comment gérer la mémoire physique ?
- Comment assurer la protection mutuelle ?

Mémoire virtuelle

- Système de pagination → à chaque page virtuelle correspond une page physique.
- Chaque adresse générée par le processeur comprend deux parties : numéro de page p et déplacement dans la page d .
- Le système maintient pour chaque programme une table de correspondance entre les pages virtuelles et les pages physiques, appelée **table des pages** (index répertoriant les adresses virtuelles et physiques des pages de données).
- Chaque processus a sa propre table des pages.
- Deux processus en mémoire peuvent utiliser la même *adresse virtuelle*.
Les adresses physiques calculés correspondantes sont différentes.

Mémoire virtuelle

- Si une adresse virtuelle (AV) a m bits (espace d'adressage virtuel 2^m), et si la taille des pages est 2^n , alors:
 - Les n bits les moins significatifs de AV sont utilisés comme déplacement (d),
 - les bits restants de AV sont utilisés comme numéro de page (p).



Application

- Supposons que la taille d'une page est de 1 Ko et un espace d'adressage logique de 15 bits.

Combien de pages y a-t-il dans le système?

Combien faut-il de bits pour adresser chaque octet dans une page?

— 1 Ko = 1024 octets = 2^{10} octets

— Il reste 5 bits pour le numéro de page.

→ 2^5 (soit 32)

- Considérons un espace d'adressage de 15 bits avec 8 pages logiques. Quelle est la taille des pages?

Application

- Avec un espace d'adressage de 15 bits et 8 pages logiques.
- Il faut **3** bits pour référencer 8 pages logiques ($2^3 = 8$). Cela laisse **12** bits pour la taille de la page et donc la taille des pages est 2^{12} (soit 4Ko)

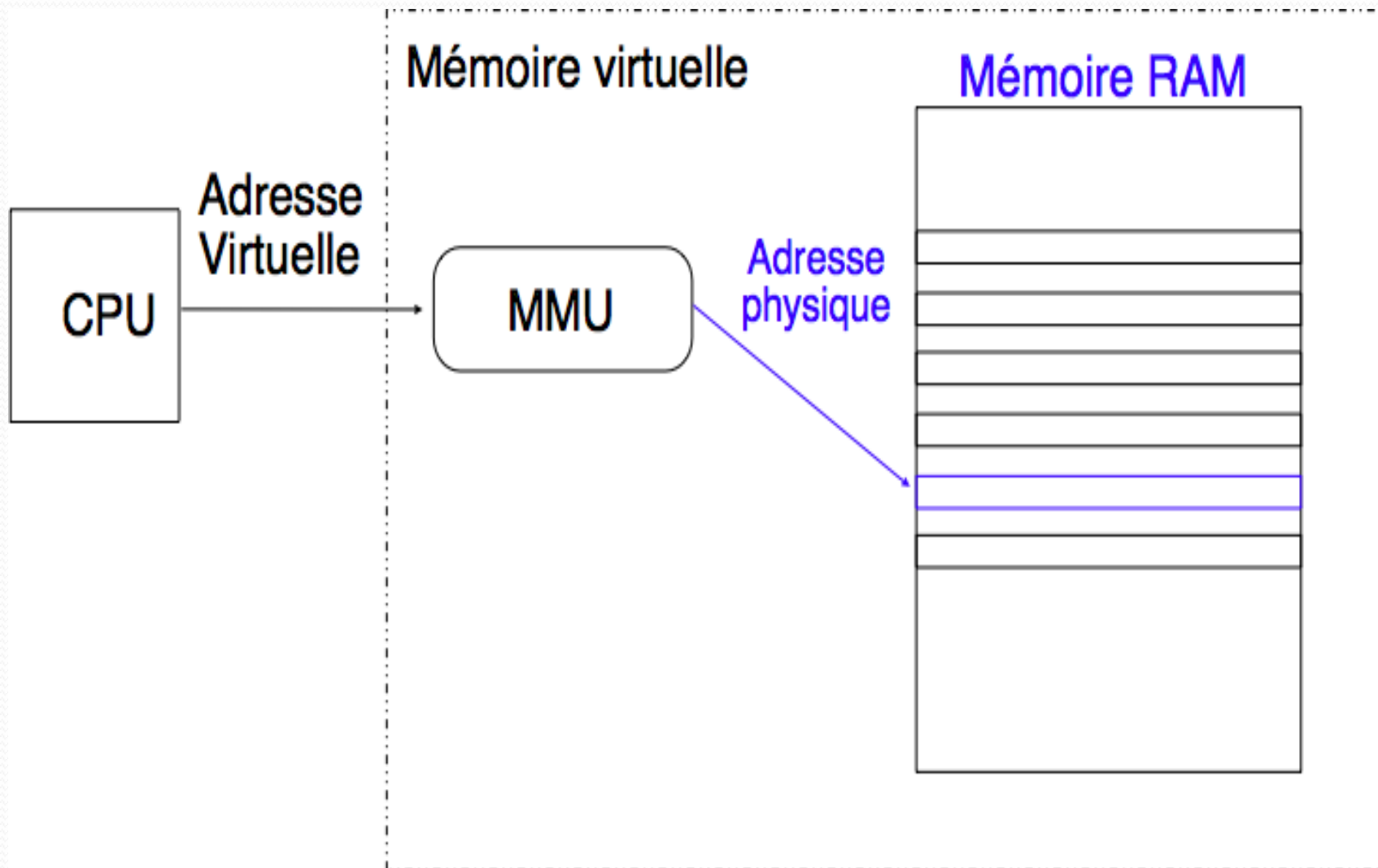
Structure d'une entrée de la table des pages

- Une entrée de la table des pages contient les numéros des pages virtuelles, leur N° de cadre (si elles sont chargées),
- Lors de la traduction d'une adresse, la MMU teste si la page est déjà chargée en mémoire
 - Si oui, l'adresse physique est calculée.
 - Si non, déroutement vers le système d'exploitation : défaut de page.
 - *Un bit* de présence/absence est ajouté dans la table des pages
- Lors du vidage d'un cadre, la MMU doit savoir si la page a été modifiée (recopie sur le disque le cadre associé lorsqu'il sera vidé)
 - *Un bit* de référencement est aussi ajouté (il joue un rôle dans certains algorithmes de remplacement de pages)
- Etc.

Mémoire virtuelle

- Le mécanisme de traduction d'adresses est réalisé en temps réel, par un composant électronique MMU (Memory Management Unit).
- Le MMU est intégré au micro-processeur de la machine, il contient la table des pages.
- Chaque entrée de la table est un couple $\langle \text{N}^\circ \text{ page}, \text{N}^\circ \text{ cadre} \rangle$

Mémoire virtuelle et MMU



Mémoire virtuelle

Page 0
Page 1
Page 2
Page 3
Page 4

Tables des pages	
N° Page	N° Cadre
Page 0	Cadre 3
Page 1	Cadre 2
Page 2	Cadre 5
Page 3	Cadre 7
Page 4	Cadre 0

Page 4	Cadre 0
	Cadre 1
Page 1	Cadre 2
Page 0	Cadre 3
	Cadre 4
Page 2	Cadre 5
	Cadre 6
Page 3	Cadre 7
	Cadre 8

Adressage virtuelle

Adressage physique

Application

- Considérons l'adresse logique 2345 et la table de pages suivante pour certains processus Po. Supposons un espace d'adressage de 15 bits avec une taille de page de 1 Ko.

Quelle est l'adresse physique à laquelle l'adresse logique 2345 sera mappée?

- Conversion de l'adresse logique en binaire :

2345 \rightarrow 000100100101001

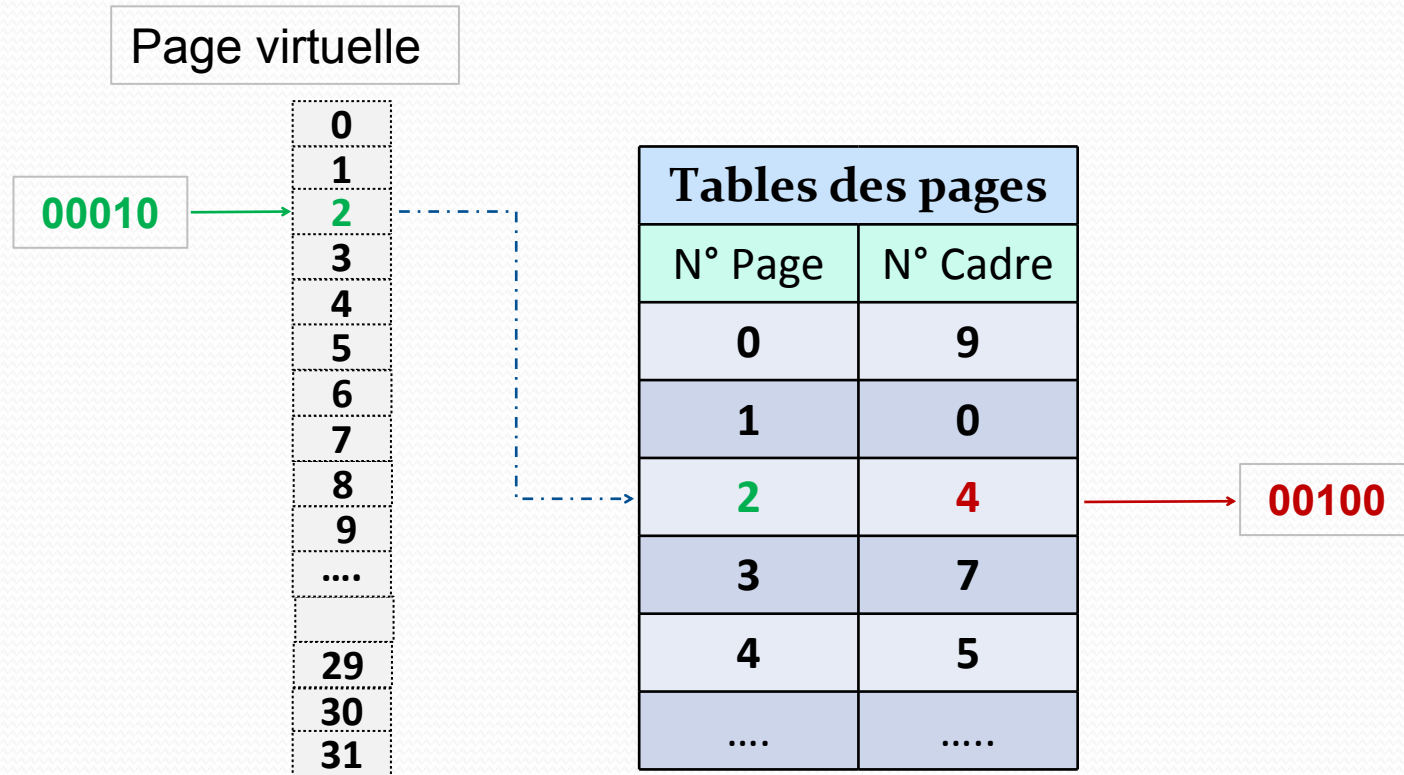
- Déterminez le numéro de page logique:

Comme il y a 5 bits alloués à la page logique, l'adresse est divisée
comme suit: 00010 0100101001

Numéro de page logique

déplacement dans la page

Exemple : Application



Application

- Récupérer le numéro de la page physique à partir de la table des pages :
- Concaténer le déplacement avec le numéro du cadre de la page physique

Adresse virtuelle **000100100101001**

Adresse physique **001000100101001**

Exemple : Application

Page virtuelle

00010

0
1
2
3
4
5
6
7
8
9
...
29
30
31

Tables des pages	
N° Page	N° Cadre
0	9
1	0
2	4
3	7
4	5
...

Mémoire physique

0 -1023
1024-2047
2048-3071
3072-4095
4096-5119
....

Page réelle

00100

0
1
2
3
4
5
6
7
8
9
...
29
30
31

Adresse physique

001000100101001 = 4393

Adresse Physique et logique - formules

- La conversion d'une adresse virtuelle AV sous la forme $AV = (p, d)$

T = taille de page

$p = AV \text{ DIV } T$: numéro de la page

$d = AV \text{ MOD } T$: déplacement dans la page

$$\Rightarrow AV = p \times T + d$$

D'une manière symétrique (avec c : numéro de la frame)

- La conversion d'une adresse physique AP sous la forme $AP = (c, d)$

Limitation des tables pages - Performance

- Lors de chaque accès à la mémoire le MMU doit d'abord consulter la table des pages pour traduire l'adresse virtuelle en une adresse physique correspondante.
- En pratique, l'utilisation d'une table des pages a comme conséquence de doubler le temps d'accès à une donnée en mémoire. Doubler ce temps d'accès aurait un impact négatif sur les performances des processeurs.
- Solution : Équiper les ordinateurs avec un périphérique qui permet de mettre en correspondance les adresses virtuelles avec les adresses physique sans passer par la table des pages

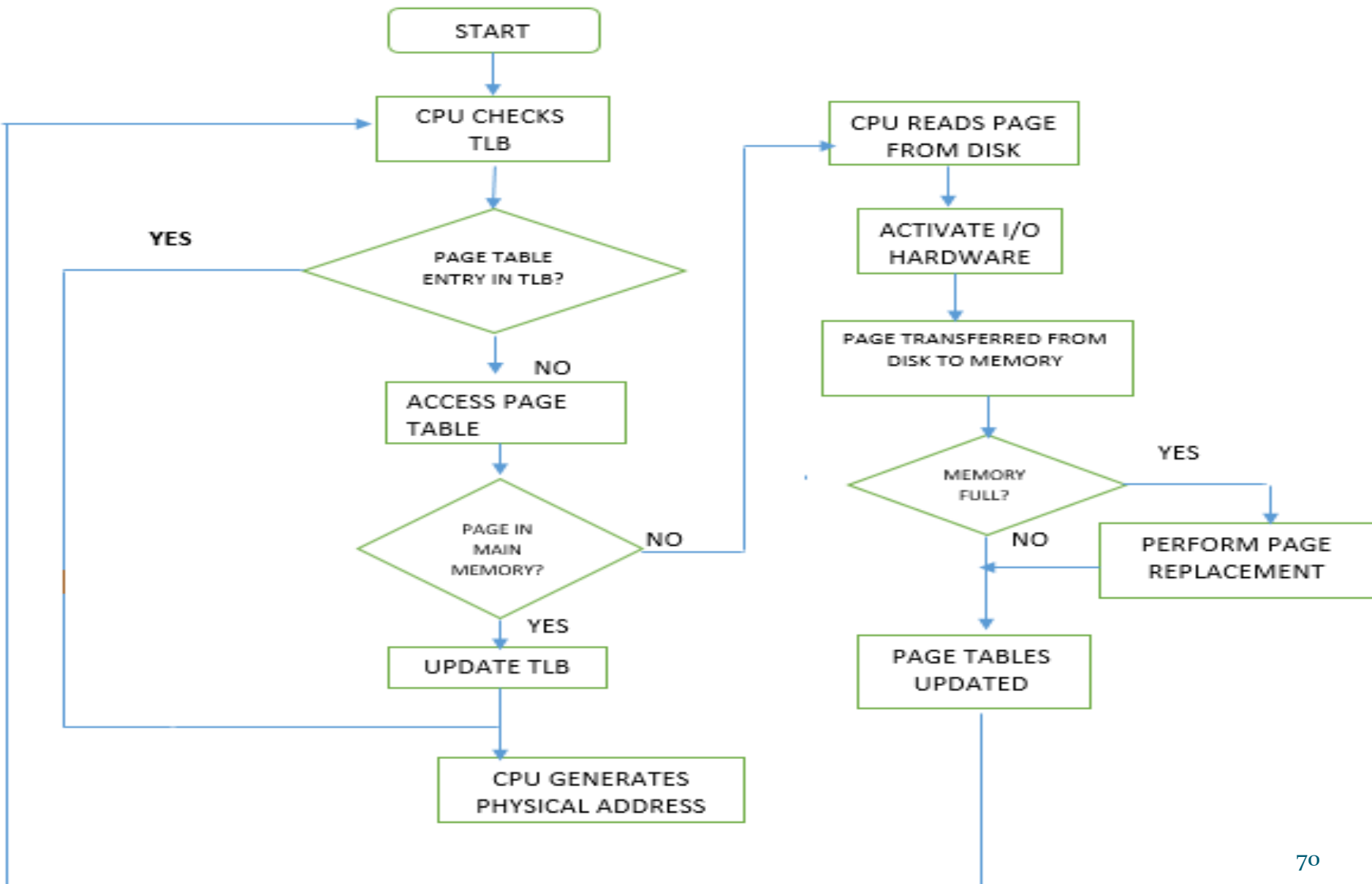
Performance de la mémoire virtuelle

- Ce composant est Translation Lookaside Buffer (TLB), appelé aussi mémoire associative.
- Le TLB, est une table de registres très rapides à l'intérieur du CPU. Il contient quelques entrées de la table de pages, les dernières utilisées. Il s'agit d'une forme de cache. Pour faire de la traduction d'adresse, le MMU regarde d'abord si la page à traduire est dans le TLB.
- Lorsque la page est présente dans le TLB (hit), le temps d'accès à la mémoire n'est pas augmenté par la pagination.

Performance de la mémoire virtuelle

- Lorsque la page n'est pas présente dans le TLB (cache miss), la table des pages est parcourue (deux accès mémoire).
- Grâce à l'utilisation du TLB, la plupart des traductions des adresses virtuelles en adresses physique peuvent être obtenus sans devoir directement consulter la table des pages
- Contient les associations (page, cadre) les plus récemment utilisées
- Accès simultané (parallèle) à toutes les entrées.
- Très efficace en pratique (moins de 5% de défaut de cache).

Algorithme schématique TLB



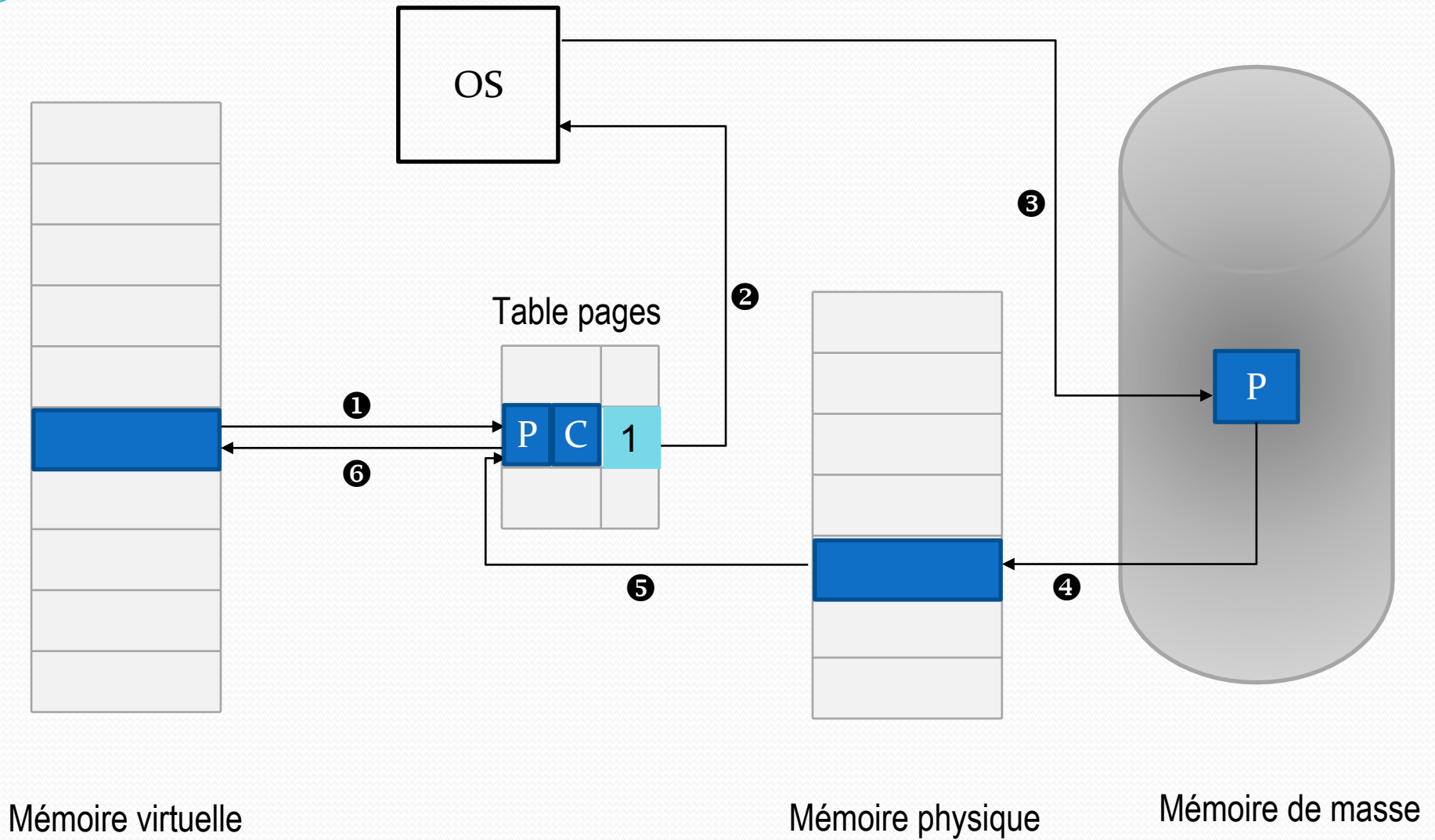
Défaut de page

- Le mécanisme de mémoire virtuelle permet d'adresser plus de pages qu'il n'y a de cadres.
 - Un certain nombre de pages sont donc déplacées sur le disque.
 - Même si le nombre de cadres est suffisant les systèmes tendent à basculer certains pages sur le disque pour augmenter la taille des caches de disque.
- La recherche dans une table de pages peut échouer pour deux raisons.
 - Aucune traduction valide pour l'adresse virtuelle spécifiée (processus tente d'accéder à une zone de mémoire non autorisée) : *erreur de segmentation*.

Défaut de page

- La page demandée n'est pas chargée en mémoire principale (bit de validation à 0) : *défaut de page*.
- La page demandée doit être alors extraite de la mémoire secondaire.
- Lors d'un défaut de page, ou lorsqu'une nouvelle page est allouée et qu'il n'y a plus de cadres disponibles une page est choisie pour être envoyée sur le disque (page victime).
- Plusieurs algorithmes de remplacement de pages.

Défaut de page



Algorithmes de remplacement de pages

- Lors d'un défaut de page, la page manquante est chargée dans une case libre.
- **Problème** : que faire pour satisfaire une demande de page non présente en mémoire principale (défaut de page) alors toute la mémoire physique est épuisé ?
- **Solution** : L'OS va chercher une page à enlever (**victime**) de la mémoire afin de faire la place pour la page en défaut. Il s'agit de *remplacement de page*.
- Si la page qui doit être remplacée a été modifiée quand elle était en mémoire, elle doit être réécrite sur le disque pour mettre à jour la copie disque.

Algorithmes de remplacement de pages

- Le système d'exploitation utilise un algorithme pour choisir une case à libérer.
- Lors d'un défaut de page, s'il n'y a pas de case libre pour charger la page demandée comment choisir la page à remplacer ? Un algorithme de remplacement peut être :
 - ❑ *Local* : si sa victime est choisie parmi les pages allouées au programme
 - ❑ *Global* : si sa victime est choisie parmi l'ensemble de toutes les pages présentes en mémoire centrale.

Algorithmes de remplacement de pages

- Un bon algorithme de remplacement doit diminuer le nombre de défauts de pages engendrés par une chaîne de référence définie par l'exécution d'un processus.
 - Choisir comme victime la page qui ne sera pas utilisé dans le futur? Difficile à deviner !
 - Choisir comme victime la page non souvent utilisée?
 - Choisir comme victime la page chargée en mémoire depuis longtemps?

Algorithme optimal

- L'ensemble des pages en mémoire est trié d'après le nombre d'instructions qui sera exécuté avant que la page correspondante soit référencée
 - La page qui sera référencée le plus tard possible dans le futur est choisie comme victime.
- Irréalisable en pratique car on ne peut pas calculer à l'avance le moment où chacune des pages sera référencée.
- Utilisé à titre de comparaison avec d'autres algorithmes réalisables en pratique.

Algorithme optimal

- Nous allons estimer le nombre de défauts de page en considérons comme exemple la chaîne de référence suivante :

6 0 1 3 0 4 0 5 3 4 0 4 3 1 3 0 1 6 0 1

6	0	1	3	0	4	0	5	3	4	0	4	3	1	3	0	1	6	0	1
6	6	6	3	3	3	3	3	3	3	3	3	3	3	3	3	3	6	6	6
	0	0	0	0	0	0	5	5	5	0	0	0	0	0	0	0	0	0	0
		1	1	1	4	4	4	4	4	4	4	4	1	1	1	1	1	1	1
DP	DP	DP	DP		DP		DP			DP			DP				DP		

Le schéma de remplacement ci-dessus donne lieu à **neuf** défauts de page

Algorithme FIFO

- Premier entré/premier sorti : le système d'exploitation maintient une liste de toutes les pages selon leur ordre de chargement en mémoire.
- La page la plus ancienne est remplacée.
- **Avantage** : Simplicité
- **Inconvénient** : il est probable que la victime soit la plus utilisée.

Algorithme FIFO

- Considérons l'exemple la chaîne de référence suivante :

6 0 1 3 0 4 0 5 3 4 0 4 3 1 3 0 1 6 0 1

6	0	1	3	0	4	0	5	3	4	0	4	3	1	3	0	1	6	0	1
6	6	6	3	3	3	3	5	5	5	0	0	0	0	0	0	0	6	6	6
	0	0	0	0	4	4	4	3	3	3	3	3	1	1	1	1	1	0	0
		1	1	1	1	0	0	0	4	4	4	4	4	3	3	3	3	3	1
DP	DP	DP	DP		DP	DP	DP	DP	DP	DP			DP	DP			DP	DP	DP

Le schéma de remplacement ci-dessus donne lieu à **quinze** défauts de page

Algorithme Seconde chance - Horloge

- Amélioration du FIFO : prise en compte de l'utilisation récente de pages.
- Les cadres qui viennent d'être utilisés (bit $R=1$) ne sont pas remplacés (*seconde chance*).

Principe :

- Si le bit R est à 0, la page est à la fois ancienne et inutilisée, elle est donc remplacée immédiatement
- Liste circulaire en forme d'horloge avec un pointeur sur la plus vieille page.

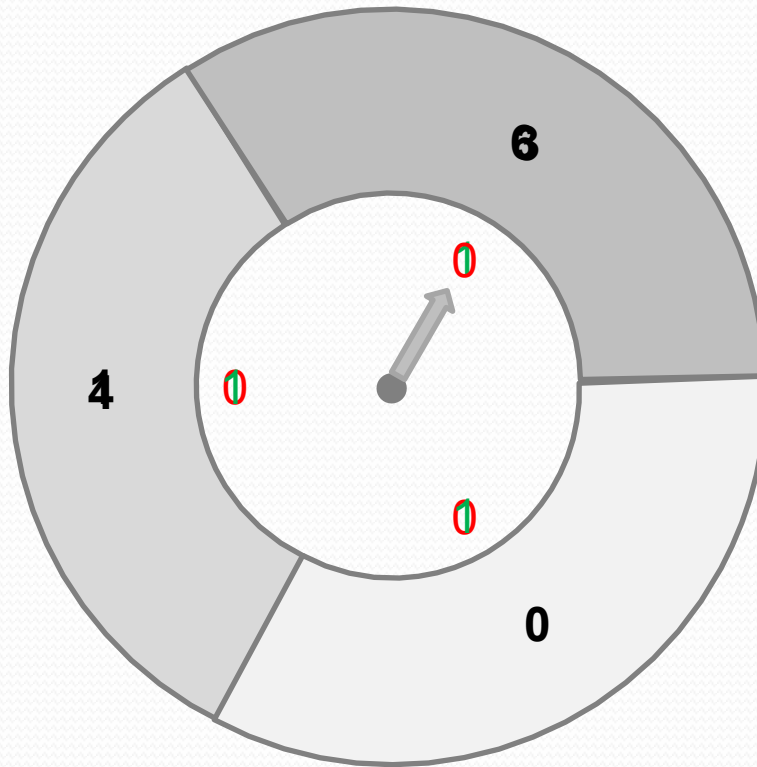
Algorithme Seconde chance - Horloge

- Lorsqu'il y a 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 d'accès est mis à 1) et le pointeur avance
 - Sinon, si $R = 1$ alors $R = 0$ et le pointeur avance
- Lorsqu'on accède à page présente dans la liste, son bit d'accès R est mis à 1
- Simple d'implémentation

Algorithme Seconde chance - Horloge

6 0 1 3 0 4 0 5 3 4 0 4 3 1 3 0 1 6 0 1

6 0 1 3 0 4



Algorithme LRU

- L'algorithme Least Recently Used (LRU) mémorise dans une liste chaînée toutes les pages en mémoire. Il choisit comme victime la page dont la dernière référence remonte au temps le plus lointain (le passé utilisé pour prédire le futur)
- Il s'agit de la page qui a la plus petite chance d'être utilisée
- **Avantage** : Performance pratique presque aussi bonne que l'algorithme Optimal.
- **Inconvénient** : Cet algorithme est coûteux.

Algorithme LRU

- Considérons l'exemple la chaîne de référence suivante :

6,0,1,3,0,4,0,5,3,4,0,4,3,1,3,0,1,6,0,1

6	0	1	3	0	4	0	5	3	4	0	4	3	1	3	0	1	6	0	1
6	6	6	3	3	3	3	5	5	5	0	0	0	1	1	1	1	1	1	1
	0	0	0	0	0	0	0	0	4	4	4	4	4	4	0	0	0	0	0
		1	1	1	4	4	4	3	3	3	3	3	3	3	3	3	6	6	6
DP	DP	DP	DP		DP		DP	DP	DP	DP			DP		DP		DP		

Le schéma de remplacement ci-dessus donne lieu à **douze** défauts de page

Algorithme NRU

L'algorithme Not Recently Used (NRU), tente à remplacer la page non récemment utilisée

Principe :

- Utilisation des bits R (page référencée) et M (page modifiée).
 - Quand une page est chargée en MC, les bits R et M sont à 0.
 - Si la page est modifiée, le bit M est à 1 : il ne change plus
 - Si la page est accédée en lecture/écriture, le bit R est mis à 1.
 - Périodiquement le bit R est remis à 0.

Algorithme NRU

- Quatre classes possibles :
 - Classe 0 : non référencée, non modifiée.
 - Classe 1 : non référencée, modifiée.
 - Classe 2 : Référencée, non modifiée.
 - Classe 3 : Référencée, modifiée.
- Priorite : Classe 3 > Classe 2 > Classe 1 > Classe 0. FIFO en cas d'égalité

Performance :

- ✓ Peu de défaut de page
- ✓ Peu couteux en mémoire
- ✗ Temps de calcul : a chaque reset et a chaque page manquante.