# 6. Gestion de la mémoire

# Gestion de la mémoire



- Plusieurs dizaines de processus doivent se partager une mémoire commune : la RAM

- La capacité de la mémoire centrale est restreinte (même si elle peut sembler importante aujourd'hui).

- Si les processus sont nombreux, ils vont occuper plus de place que ne peut leur offrir la mémoire centrale.

# Gestion de la mémoire



- **On distingue deux grandes catégories de mémoires :**
  - ✓**mémoire centrale** (appelée également mémoire interne)
  - ✓**mémoire de masse** - correspond aux dispositifs de stockage magnétiques (disque dur), optique (CD-ROM, DVD-ROM), mémoires mortes )
- **Les éléments qui caractérisent une mémoire :**
  - ✓le mode de localisation de l'information (aléatoire ou séquentiel)
  - ✓la capacité ou volume
  - ✓le temps d'accès
  - ✓le temps de cycle
  - ✓Le débit
  - ✓la volatilité

# Gestion de la mémoire



**Mémoire cache :**

- Mémoire intermédiaire entre le processeur et la mémoire centrale
- Elle est intégrée dans le processeur et est cadencée à la même fréquence
- **Débit de la mémoire centrale très lent par rapport au débit requis par le processeur : la mémoire cache accélère la vitesse de lecture des informations par le CPU en les plaçant (en avance) dans le cache**
- Mémoire associative
- Mémoire très rapide
- Taille : de quelques centaines de Ko à quelques Mo

# Gestion de la mémoire

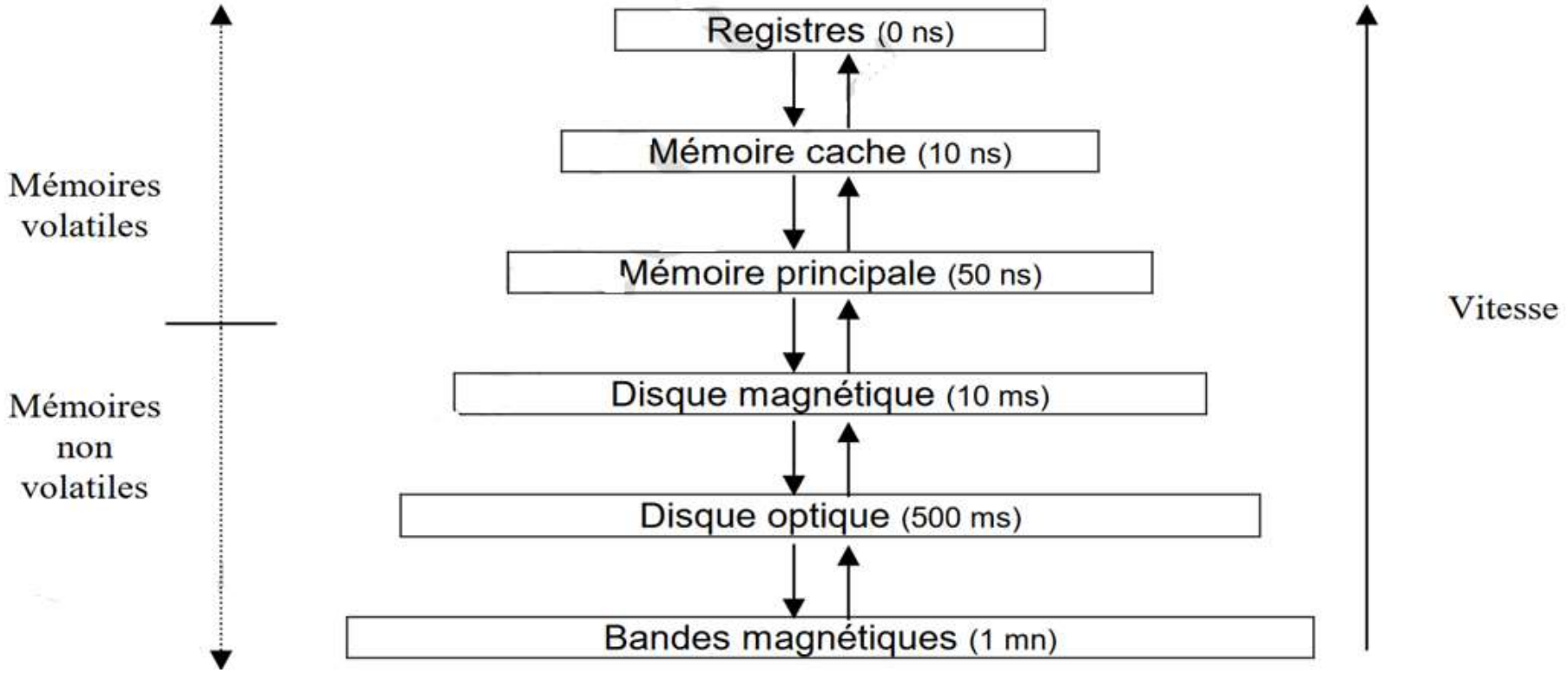


**Registres:**

- Se trouvent intégrés dans le CPU
- Un registre est un mot stockant des informations relatives à une instruction
  - ✓ Opérandes
  - ✓ Paramètres
  - ✓ Résultats
- Peu nombreux dans un CPU
- Très rapides (vitesse du CPU)

# Gestion de la mémoire






**La hiérarchie mémoire**

# Gestion de la mémoire



**Types d'accès à la mémoire:**

- **Accès séquentiel** (e.g. bande magnétique) : le plus lent de tous
- **Accès direct ou aléatoire** (e.g. mémoire centrale, registres) : chaque information a une adresse.
- **Accès semi-séquentiel**
- **Accès par le contenu** (e.g. mémoire cache)

# Gestion de la mémoire



- Le multiprogrammation implique que les programmes d'application sont chargés dans des zones de la mémoire dont la localisation n'est pas connue d'avance.
- Le système prend à sa charge la gestion de la mémoire principale.
- Le système assure sa protection, et il fournit l'information qui permet à chaque programme de s'exécuter sans savoir à l'avance dans quelle zone mémoire il sera au moment de son exécution (cette zone peut d'ailleurs changer entre différentes exécutions et même en cours d'exécution).
- Le but d'une bonne gestion de la mémoire est d'augmenter le rendement global du système.
- La question principale à laquelle le système devra répondre chaque fois qu'un nouveau programme demande à être exécuté sera donc "à quel endroit dans la mémoire ?"

# Espace d’adressage logique ou physique



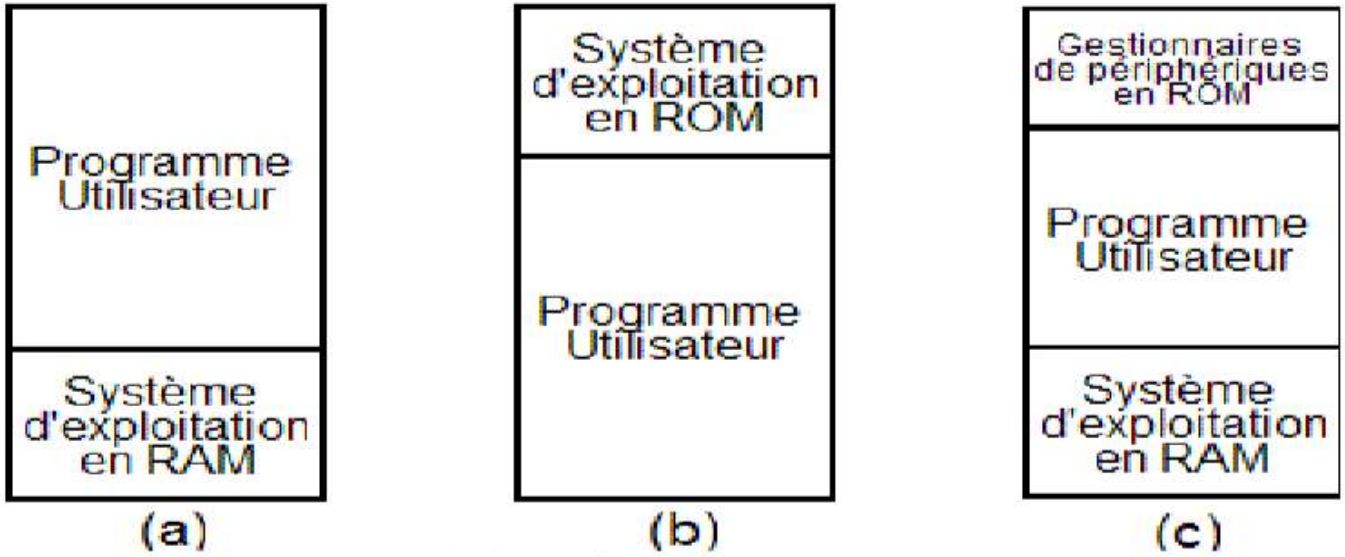
- L’unité centrale manipule des **adresses logiques** (emplacement relatif).
- Les programmes ne connaissent que des adresses logiques, ou virtuelles.
- L’espace d’adressage logique (virtuel) est donc un ensemble d’adresses pouvant être générées par un programme.

- L’unité mémoire manipule des **adresses physiques** (emplacement mémoire).
- Les adresses physiques ne sont jamais vues par les programmes utilisateurs.
- L’espace d’adressage physique est un ensemble d’adresses physiques correspondant à un espace d’adresses logiques.

# Monoprogrammation vs. Multiprogrammation



- **Organisation de la mémoire en monoprogrammation**
  - A la base, le rôle du plus simple gestionnaire de mémoire est d'exécuter un seul programme à la fois, en partageant la mémoire entre le programme et le système d'exploitation.
  - Trois variantes de l'organisation possible sont montrées à la figure suivante :

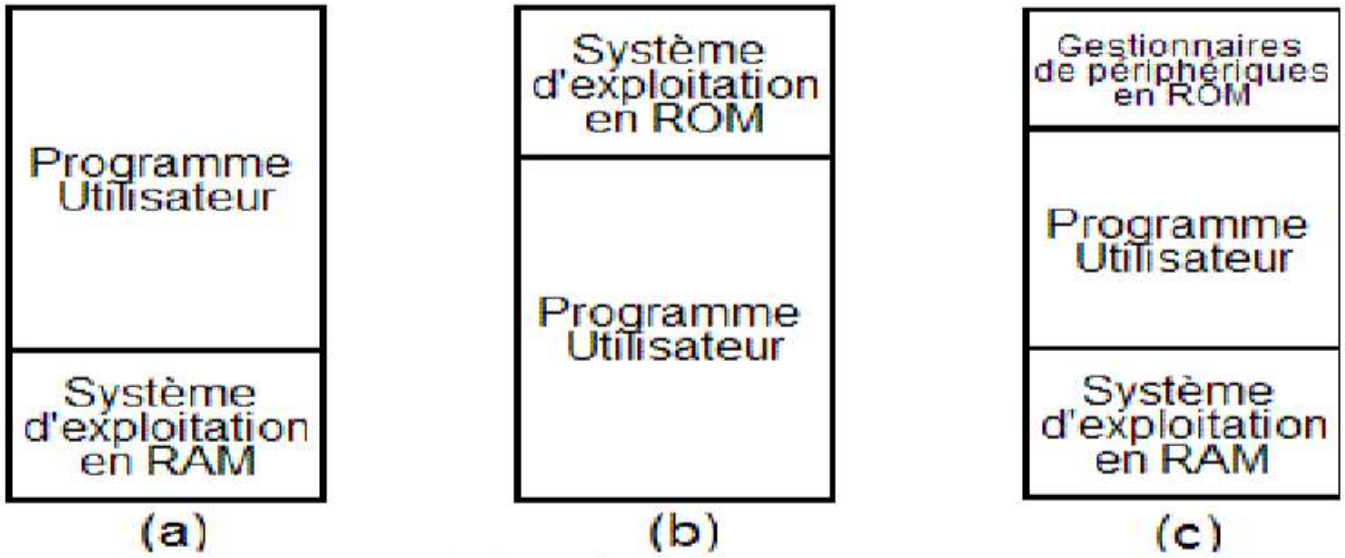



*Organisation de la mémoire en monoprogrammation*

# Monoprogrammation vs. Multiprogrammation



- **Organisation de la mémoire en monoprogrammation**
  - Autrefois utilisé dans les mainframes et les mini-ordinateurs, le premier modèle est aujourd'hui rare.
  - Le deuxième modèle équipe plusieurs ordinateurs de poche ou systèmes embarqués




*Organisation de la mémoire en monoprogrammation*

# Monoprogrammation vs. Multiprogrammation



- **Organisation de la mémoire en monoprogrammation**

- Le troisième modèle était exploité sur les premiers ordinateurs personnels (ceux que faisait fonctionner MS-DOS), dans lesquels la partie du système d'exploitation en ROM s'appelait le BIOS.

- Quand le système est organisé de cette manière, un seul processus peut s'exécuter à la fois.

- Dès que l'utilisateur tape une commande, le système d'exploitation copie le programme demandé depuis le disque vers la mémoire et l'exécute.

# Monoprogrammation vs. Multiprogrammation



- **Gestion de la mémoire et multiprogrammation**

- Excepté sur des systèmes embarqués simples, on n'a plus tellement recours à la monoprogrammation de nos jours.
- La plupart des systèmes modernes autorisent l'exécution de processus multiples en même temps, ce qui implique le séjour de plusieurs programmes en même temps en mémoire et c'est cette technique qui a donné naissance à la gestion moderne de la mémoire.
- Pour supporter la multiprogrammation et donc l'existence de plusieurs processus en mémoire principale, on peut distinguer deux grandes stratégies d'allocation mémoire: **l'allocation de partitions contiguës** et **l'allocation de partitions non-contiguës**.

# Multiprogrammation et partitions multiples contigües



## 1. Partitions contiguës fixes

***Table de description des partitions***

- Dans cette stratégie la mémoire est divisée en *n* partitions de tailles fixes (si possible inégales). Ce partitionnement se fait au démarrage du système. Le système d'exploitation maintient une table de description des partitions.
- ***Exemple : Table de description des partitions***
- Le tableau suivant est un exemple de table de description de partitions :

| Adresse | Taille | Etat |
|---|---|---|
| 312k | 8K | allouée |
| 320k | 32K | allouée |
| 352k | 32K | libre |
| 384k | 120K | libre |
| 504k | 520K | allouée |

Tableau 6 : Table de description de partitions

# Multiprogrammation et partitions multiples contigües



## 1. Partitions contiguës fixes

- La protection consiste à interdire au processus de construire une adresse en dehors de sa partition
- Le processeur peut être doté par exemple de deux registres accessibles uniquement en mode noyau :

  1) un registre de base contenant la limite inférieure de la partition,

  2) un registre de limite contenant la limite supérieure de la partition,

- Le processeur se déroutera vers le système si un processus génère une adresse en dehors de sa partition.
- L'allocation de la mémoire physique est simple, puisqu'un processus reçoit une portion de mémoire physique de la même taille que son espace d'adressage

# Multiprogrammation et partitions multiples contigües



## 1. Partitions contiguës fixes

- La place occupée par un programme étant inférieure ou égale à la taille de la partition, il reste souvent des espaces libres inoccupés .
- Il existe plusieurs algorithmes d'attribution des partitions aux processus et le système peut gérer plusieurs queues de processus en attente, en triant les processus selon leur taille, ou une file d'attente unique .
- Lorsqu'une partition est libre, le système recherche un processus de la file qui peut se satisfaire de la taille de cette partition, et le charge en mémoire dans cette partition.
- La recherche peut s'arrêter sur le premier trouvé, ou rechercher celui qui a le plus gros besoin d'espace, tout en étant inférieur à la taille de la partition ; notons que ceci peut entraîner la famine d'un processus qui ne demanderait qu'un tout petit espace

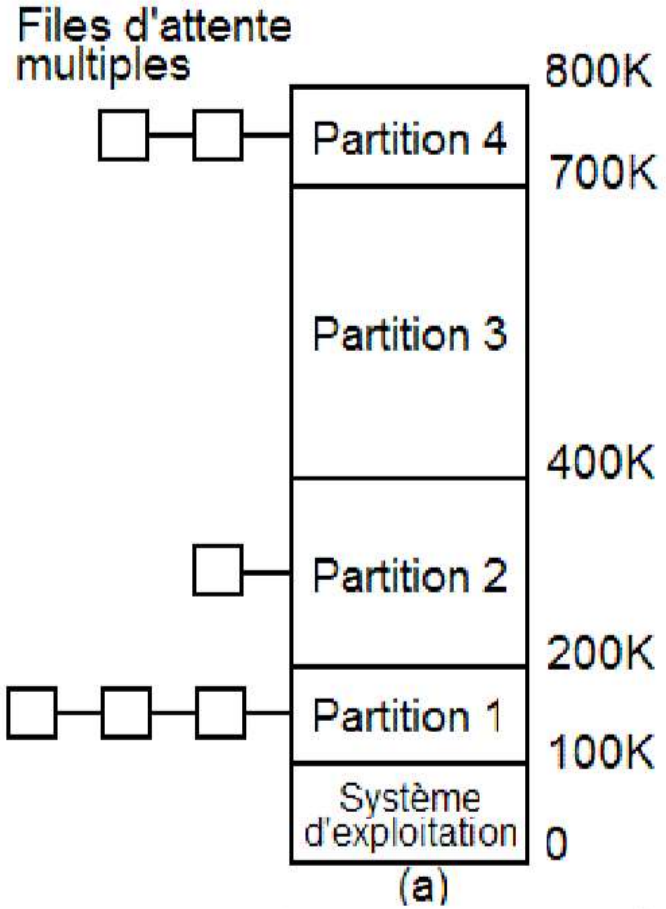
# Multiprogrammation et partitions multiples contigües



## 1. Partitions contiguës fixes

### Allocation de partitions fixes : files d'attente séparée

- Une file d'attente est associée à chaque partition.
- 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. La figure illustre ce type de systèmes à partitions fixes.
- Le tri des processus qui arrivent en différentes files d'attente présente des inconvénients lorsque la file d'attente pour une grande partition est vides tandis que celle d'une petite partition est pleine, comme l'illustrent les partitions 1 et 3 de la figure




*Partitions contigües fixes : files d'attente séparées*

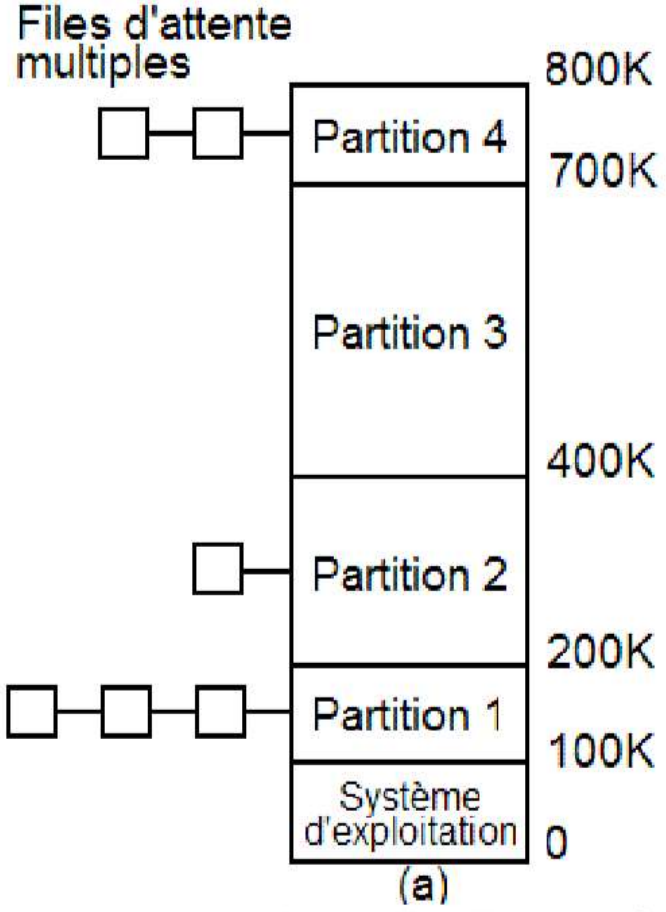
# Multiprogrammation et partitions multiples contigües



## 1. Partitions contiguës fixes

**Allocation de partitions fixes : files d'attente séparée**

- Le fait d'éviter d'allouer une partition trop grande à un petit processus conduit parfois à des aberrations.
- Il arrive que des partitions plus grandes restent inutilisées alors que se forment ailleurs des files interminables de petits processus. La mémoire est donc mal utilisée.
- On perd en général de la place au sein de chaque partition (**fragmentation interne**)
- Il peut y avoir des partitions inutilisées (leur file d'attente est vide)
- Une autre solution est de créer une file unique.




*Partitions contigües fixes : files d'attente séparées*

# Multiprogrammation et partitions multiples contigües

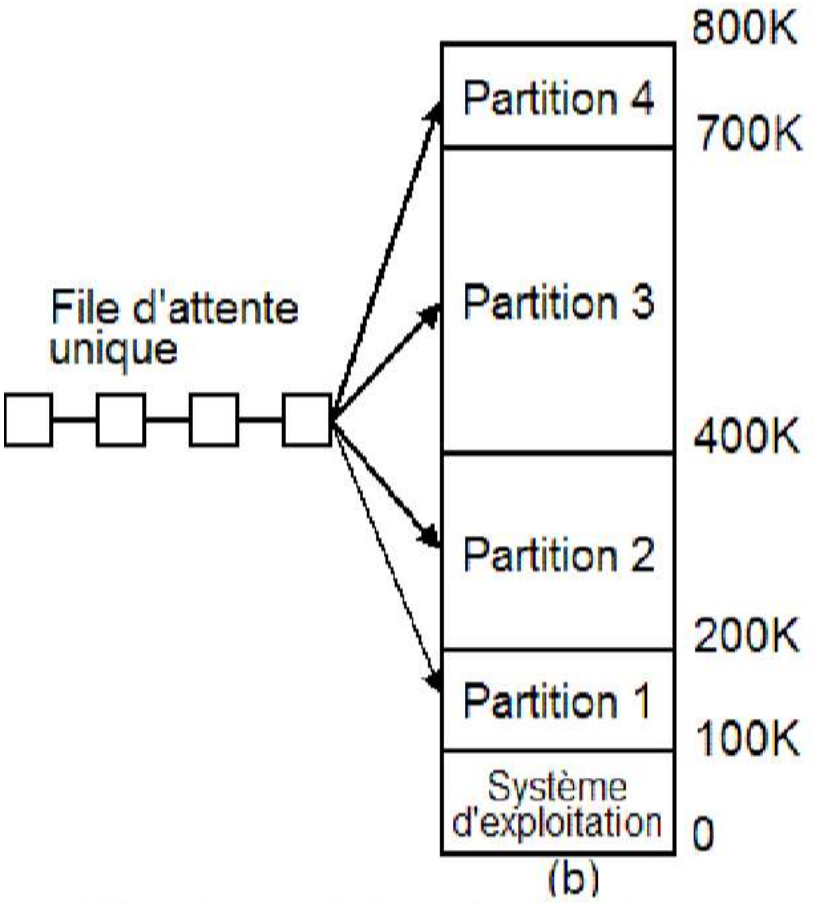


## 1. Partitions contiguës fixes

### Allocation de partitions fixes : file d'attente commune

- Une autre organisation possible consiste à gérer une seule file d'attente, comme montré dans la figure

- Dès qu'une partition devient libre, toute tâche placée en tête de file d'attente et dont la taille convient peut être chargée dans cette partition vide et exécutée.




*Partitions fixes contigües : file d'attente commune*

# Multiprogrammation et partitions multiples contigües



## 2. Partitions contiguës dynamiques

***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.

- Chaque processus est alloué exactement la taille de mémoire requise.

- Probablement des trous inutilisables se formeront dans la mémoire: c'est la **fragmentation externe**

# Multiprogrammation et partitions multiples contigües



## 2. Partitions contiguës dynamiques

***Stratégies d'allocation:*** Le lancement des processus dans les partitions se fait selon différentes stratégie

1. 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.

2. Stratégie du meilleur qui convient (Best Fist):

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.

3. Stratégie du pire qui convient (Worst Fit):

On alloue au processus la partition de plus grande taille

# Multiprogrammation et partitions multiples contigües



## 2. Partitions contiguës dynamiques

**Allocation de partitions dynamiques contigües :**

- Soit une MC dont la table des partitions libres est la suivante :

| Adresse | Taille |
|---|---|
| 352K | 32K |
| 504K | 520K |

*Tableau 7 : Partitions libres*

- Soit les demandes suivantes P4(24K), P5(128K), P6(256K).

Evolution de la table des partitions libres:

**First Fit:**

| init | P4 | P5 | P6 |
|---|---|---|---|
| 32K | 8K | 8K | 8K |
| 520K | 520K | 392K | 136K |

**Best Fit:**

| init | P4 | P5 | P6 |
|---|---|---|---|
| 32K | 8K | 8K | 8K |
| 520K | 520K | 392K | 136K |

**Worst Fit:**

| init | P4 | P5 | P6 |
|---|---|---|---|
| 32K | 32K | 32K | 32K |

*Tableau 8 : Allocation de partitions dynamiques contiguës selon stratégies différentes*

# Multiprogrammation et partitions multiples contigües



## 2. Partitions contiguës dynamiques

- Etant donné cinq partitions de mémoire de **100 Ko, 500 Ko, 200 Ko, 300 Ko et 600 Ko** (dans l'ordre).

- Comment chacun des algorithmes du first-fit, best-Fit et worst-Fit placent les processus de **212 Ko, 417 Ko, 112 Ko et 426 Ko** (dans l'ordre) ?

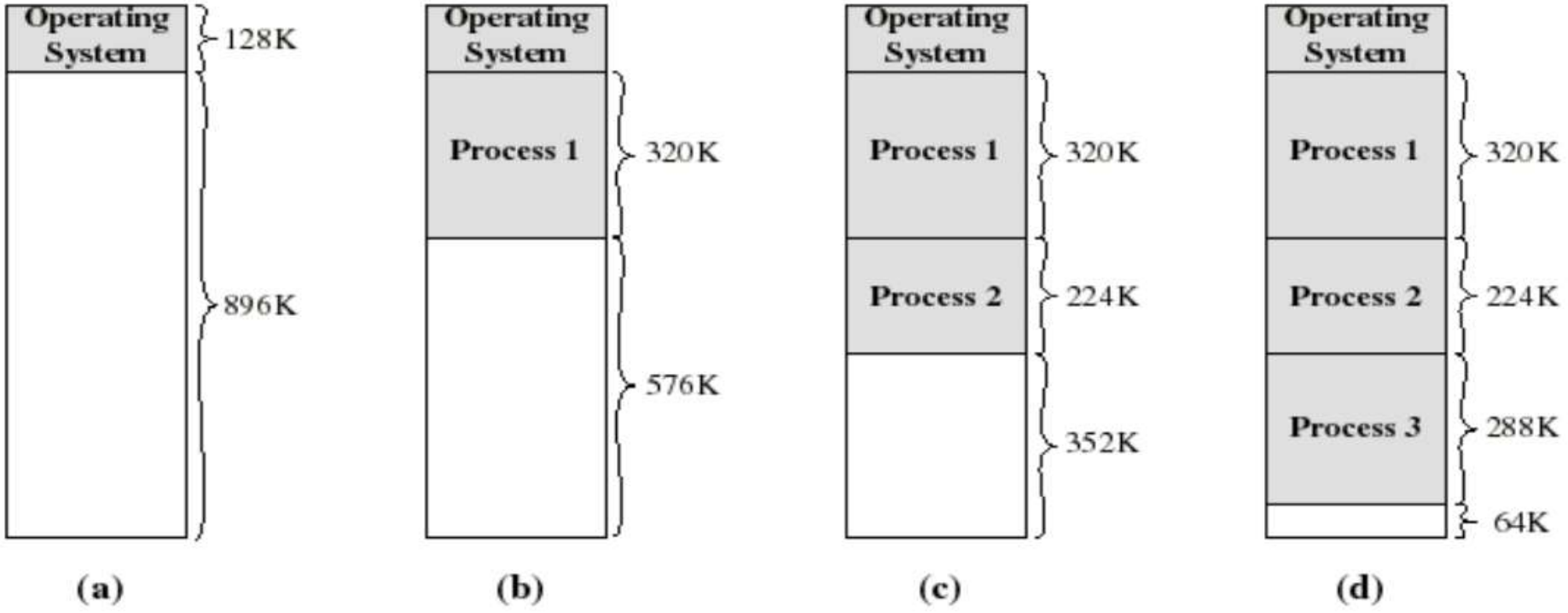
- Quel algorithme optimise le mieux l'utilisation de la mémoire ?

# Multiprogrammation et partitions multiples contigües



## 2. Partitions contiguës dynamiques

**Allocation de partitions dynamiques contigües : Fragmentation externe**

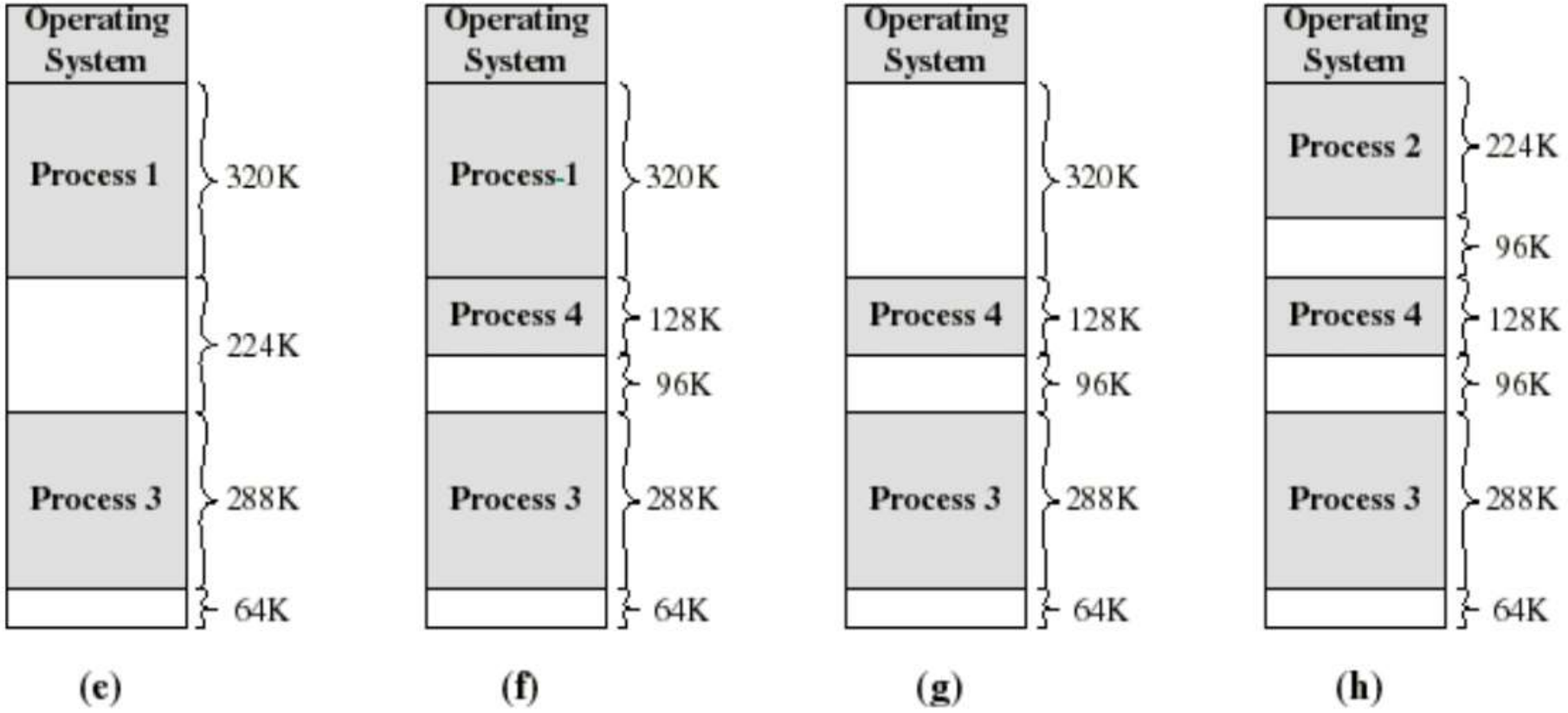


# Multiprogrammation et partitions multiples contigües



## 2. Partitions contiguës dynamiques

**Allocation de partitions dynamiques contigües : Fragmentation externe**



# Multiprogrammation et partitions multiples contigües



## 3. Partitions contiguës Siamoises (Buddy system)

- C'est un compromis entre partitions de tailles fixes et partitions de tailles variables.
- La mémoire est allouée en unités qui sont des puissances de 2.
- Initialement, il existe une seule unité comprenant toute la mémoire.
- Lorsque de la mémoire doit être attribuée à un processus, ce dernier reçoit une unité de mémoire dont la taille est la plus petite puissance de 2 supérieure à la taille du processus.
- S'il n'existe aucune unité de cette taille, la plus petite unité disponible supérieure au processus est divisée en deux unités "siamoises" de la moitié de la taille de l'original.
- La division se poursuit jusqu'à l'obtention de la taille appropriée. De même deux unités siamoises libres sont combinées pour obtenir une unité plus grande.

# Multiprogrammation et partitions multiples contigües



## 3. Partitions contiguës Siamoises (Buddy system)

### Exemple 1

| Action | Mémoire | | | | |
|---|---|---|---|---|---|
| Au départ | 1Mo | | | | |
| A, 150 ko | A | 256ko | | | 512ko |
| B, 100 ko | A | B | 128ko | | 512ko |
| C, 50 ko | A | B | C | 64ko | 512ko |
| B se termine | A | 128ko | C | 64ko | 512ko |
| C se termine | A | 256ko | | | 512ko |
| A se termine | 1Mo | | | | |

Tableau 9 : Buddy system

# Multiprogrammation et partitions multiples contigües



### 3. Partitions contiguës Siamoises (Buddy system) : Exemple 2

| Action | Mémoire | | | | | |
|---|---|---|---|---|---|---|
| Au départ | 1M | | | | | |
| A=70K | A | 128 | | 256 | | 512 K |
| B=35K | A | B | 64 | 256 | | 512 |
| C=80K | A | B | 64 | C | 128 | 512 |
| A se termine | 128 | B | 64 | C | 128 | 512 |
| D=60K | 128 | B | D | C | 128 | 512 |
| B se termine | 128 | 64 | D | C | 128 | 512 |
| D se termine | 256 | | | C | 128 | 512 |
| C se termine | 512 | | | | | 512 |
| Fin | 1024K | | | | | |

# Multiprogrammation et partitions multiples contigües



## 4. Ré-allocation et protection

- La multiprogrammation introduit deux problèmes essentiels à résoudre: la réallocation et la protection.

- Avec la multiprogrammation plusieurs processus s'exécutent à différentes adresses.

- Quand le programme est lié (programme principal, sous-programmes utilisateur, et sous programmes des bibliothèques ne forment qu'un seul espace d'adressage), l'éditeur de lien doit savoir à quelle adresse le programme démarrera en mémoire.

# Multiprogrammation et partitions multiples contigües



## 4. Ré-allocation et protection

**Exemple :**

- Supposons que la première instruction corresponde à l'appel d'une procédure à l'adresse absolue 100 dans le fichier binaire produit par l'éditeur de lien.
- Si le programme est chargé dans la partition 1 (à l'adresse 100K), cette instruction sautera à l'adresse absolue 100, laquelle se trouve dans la zone du système d'exploitation. Ce qui est nécessaire, c'est l'adresse 100K+100.
- Si le programme est chargé dans la partition 2, il aura besoin d'une adresse à 200K+100, et ainsi de suite. Ce problème est appelé problème de ré-allocation ou de translation d'adresse

# Multiprogrammation et partitions multiples contigües



## 4. Ré-allocation et protection

### Protection

- Dans les systèmes multi-utilisateurs, il est déconseillé d'autoriser des processus à lire ou à écrire dans la mémoire appartenant à d'autres utilisateurs (et a fortiori appartenant au système d'exploitation). Ce problème est appelé problème de protection.

### Registres de base et de limite

- Une solution aux deux problèmes de la ré-allocation et de la protection consiste à équiper l'ordinateur avec deux registres matériels particuliers, appelés registres de base et de limite.
- Quand un processus est activé, le registre de base est chargé avec l'adresse de départ de la partition, et le registre de limite est chargé avec la longueur de la partition

# Multiprogrammation et partitions multiples contigües



## 4. Ré-allocation et protection

### Registres de base et de limite

- Les adresses sont comparées avec la valeur du registre de limite, afin d'assurer qu'elles ne référenceront pas une adresse hors de la partition courante.

- La valeur du registre base est ajoutée à toutes les adresses mémoire générées automatiquement avant qu'elles ne soient envoyées en mémoire.

- Ainsi, si le registre de base contient la valeur 100K, une instruction comme CALL 100 se transforme effectivement en CALL 100K+100.

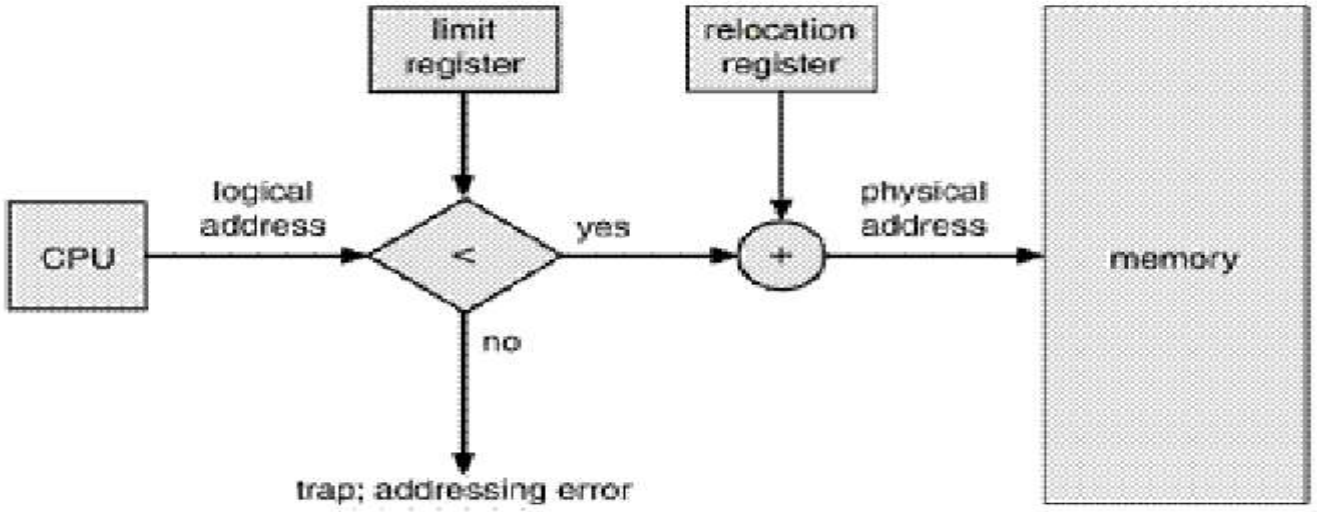
# Multiprogrammation et partitions multiples contigües



## 4. Ré-allocation et protection

### Registres de base et de limite

- Le matériel protège les registres de base et de limite afin d'empêcher les programmes utilisateur de les modifier. La figure illustre ce mécanisme de ré-allocation et de protection.




*Registres de base et limite*

# Multiprogrammation et partitions multiples contigües



## 5. Va et vient (Swap)

### Multi-programmation et limite de la mémoire principale

- Dans un système à temps partagé, parfois la mémoire principale est insuffisante pour maintenir tous les processus courants actifs : il faut alors conserver les processus supplémentaires sur un disque et les charger pour qu'ils s'exécutent dynamiquement.

### Swap

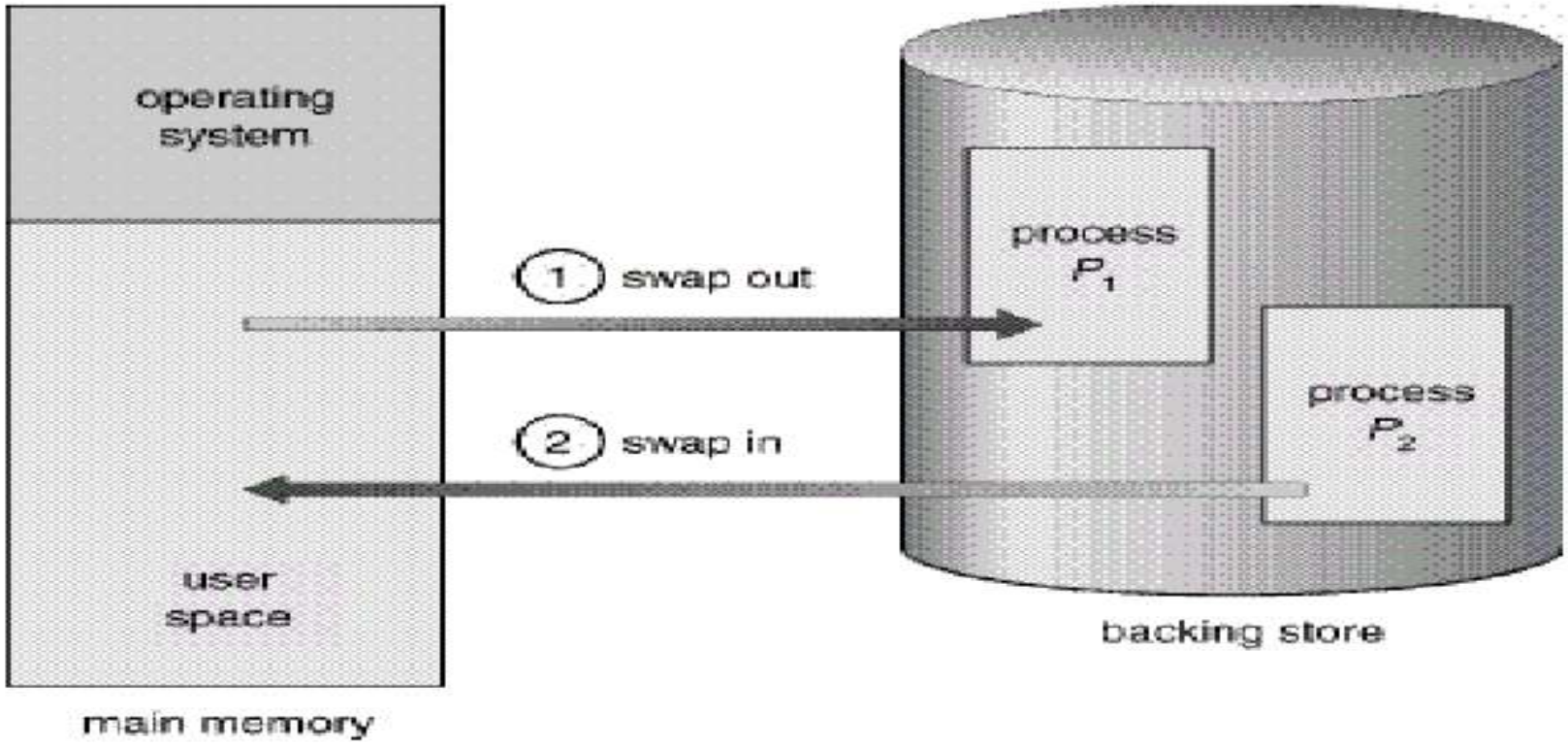
- La stratégie la plus simple, appelée va-et-vient (swap) consiste à considérer chaque processus dans son intégralité: en cas de réquisition du processeur, le programme en cours doit être sauvegardé sur disque avant le chargement en mémoire principale de son successeur, dans sa totalité, pour exécution.

# Multiprogrammation et partitions multiples contigües



## 5. Va et vient (Swap)

**Swap**




*Swap*

# Multiprogrammation et partitions multiples contigües



## 6. Fragmentation et Compactage

**Définition : Fragmentation interne**

- Soient un programme de taille *M* et une partition de taille *N* (partition fixe). Si la partition est allouée au programme avec *N>M* alors la partie non occupée par le programme est appelée fragmentation interne.

**Définition : Fragmentation externe**

- Soit un programme de taille *M*. Si toute partition libre est de taille *Pi* telle que *Pi<M* alors le programme ne peut être chargé dans aucune partition libre alors que :

$$\sum_i P_i \geq M$$

*Formule 1 : Fragmentation externe*

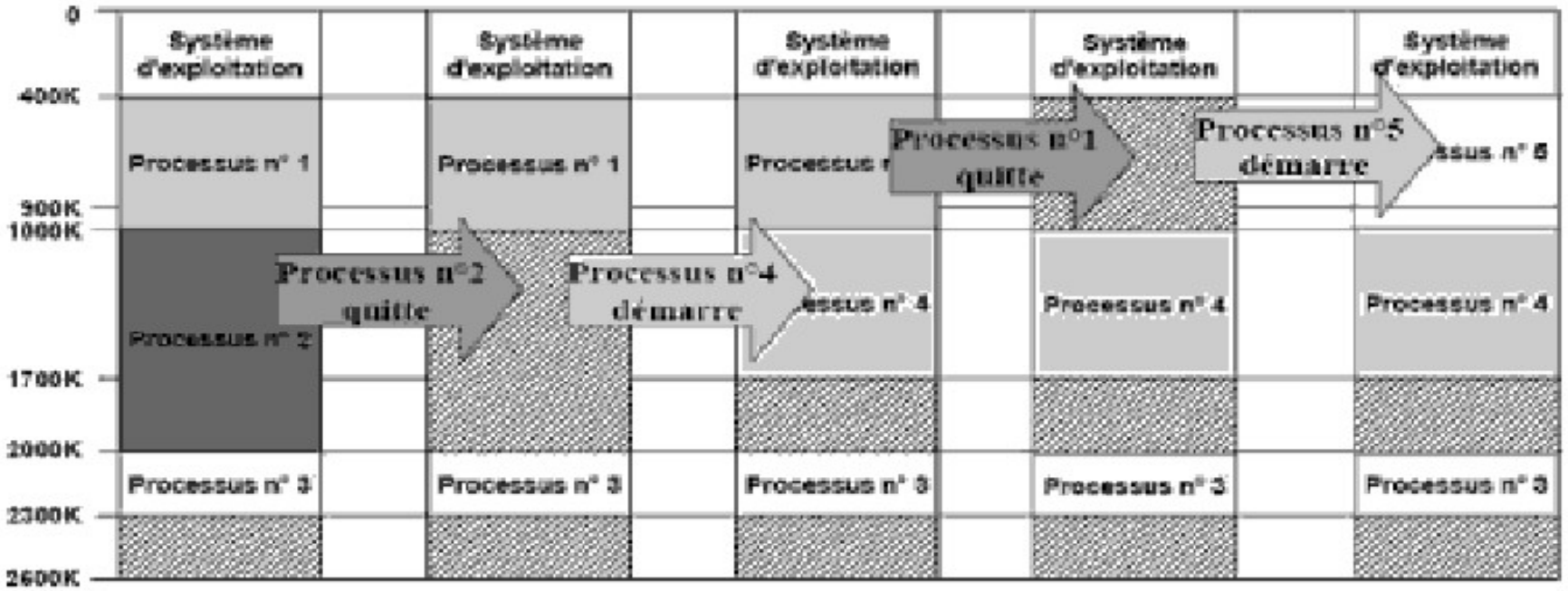
- On dit qu'il y a fragmentation externe. La mémoire est fragmentée en un grand nombre de petits trous.

# Multiprogrammation et partitions multiples contigües



## 6. Fragmentation et Compactage

La figure illustre le phénomène de fragmentation de la mémoire.




*Fragmentation mémoire*

# Multiprogrammation et partitions multiples contigües



## 6. Fragmentation et Compactage

**Méthode : Compactage de la mémoire**

- Les zones libres sont regroupées. C'est une technique très coûteuse, et en plus elle change les adresses de chargement des processus.

**Remarque : Compactage et réallocation**

- Le compactage de la mémoire n'est possible que s'il y a un mécanisme de réallocation dynamique des processus (basé sur l'utilisation des registres de translation d'adresse (registre base))

# Multiprogrammation et partitions multiples non contigües



- La traduction dynamique d'adresse réalisée par l'emploi d'un registre de base impose que les programmes soient chargés en mémoire principale dans des cellules contiguës.

- Les techniques d'allocation associées conduisent à la fragmentation de la mémoire. Pour l'éviter, il faut pouvoir implanter un programme dans plusieurs zones non contiguës.

- Plusieurs techniques proposent d'allouer des espaces mémoire non-contiguës pour les processus : **Pagination et Segmentation**.

# Multiprogrammation et partitions multiples non contigües



## 1. Pagination

- La solution apportée par les mécanismes de pagination consiste à découper l'espace adressable, ou espace virtuel, en zones de taille fixe appelée pages.
- La mémoire réelle est également découpée en cases (frame en anglais) ayant la taille d'une page de sorte que chaque page peut être implantée dans n'importe quelle case de la mémoire réelle
- Chaque page peut alors être placée dans une case quelconque.
- Une table de correspondance (une par processus), appelée table des pages (page table) est gérée par le système d'exploitation.

# Multiprogrammation et partitions multiples non contigües



## 1. Pagination

- En général, l'adresse de cette table est une adresse physique. Chaque entrée de cette table contient les informations suivantes :

✓L'indicateur de *présence* indique s'il y a une case allouée à cette page.

✓Le *numéro de case* alloué à cette page.

✓Les indicateurs de *protection* de la page indique les opérations autorisées sur cette page par le processus.

✓L'indicateur de *page accédée* est positionné lors d'un accès quelconque.

✓L'indicateur de *page modifiée* est positionné lors d'un accès en écriture.

# Multiprogrammation et partitions multiples non contigües



## 1. Pagination

### a) Pagination à un niveau

- Une adresse est divisée en deux parties : un numéro de page *p* et un déplacement à l'intérieur de la page *d*

- La taille de la page (et donc de la case) est une puissance de 2 variant généralement entre 512 et 8192 octets selon les architectures

- Le mécanisme de traduction des adresses virtuelles en adresses réelles doit associer à chaque numéro de page virtuelle le numéro de case réelle qui contient cette page, si elle existe.

# Multiprogrammation et partitions multiples non contigües

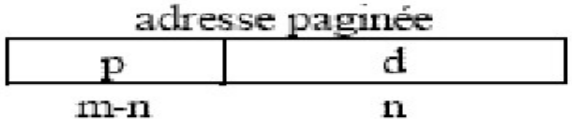


## 1. Pagination

### a) Pagination à un niveau

- Le choix de tailles de puissance 2 permet de simplifier la division nécessaire pour calculer le numéro de page et le déplacement à partir d'une adresse logique, ainsi : Si la taille de l'espace d'adressage logique est $2^m$ et la taille d'une page est $2^n$ (octets ou mots)

  - Les *m-n* bits de poids fort d'une adresse paginée désignent le numéro de page ; et
  - Les *n* bits de poids faible désignent le déplacement dans le page.



Structure d'une adresse virtuelle

- Si *T* est la taille d'une page et *A* une adresse logique, l'adresse paginée (*p, d*) est déduite à partir des formules : $p = A \ div \ T$, $d = A \ modulo \ T$ et l'adresse logique est égale à : $A = p*T + d$

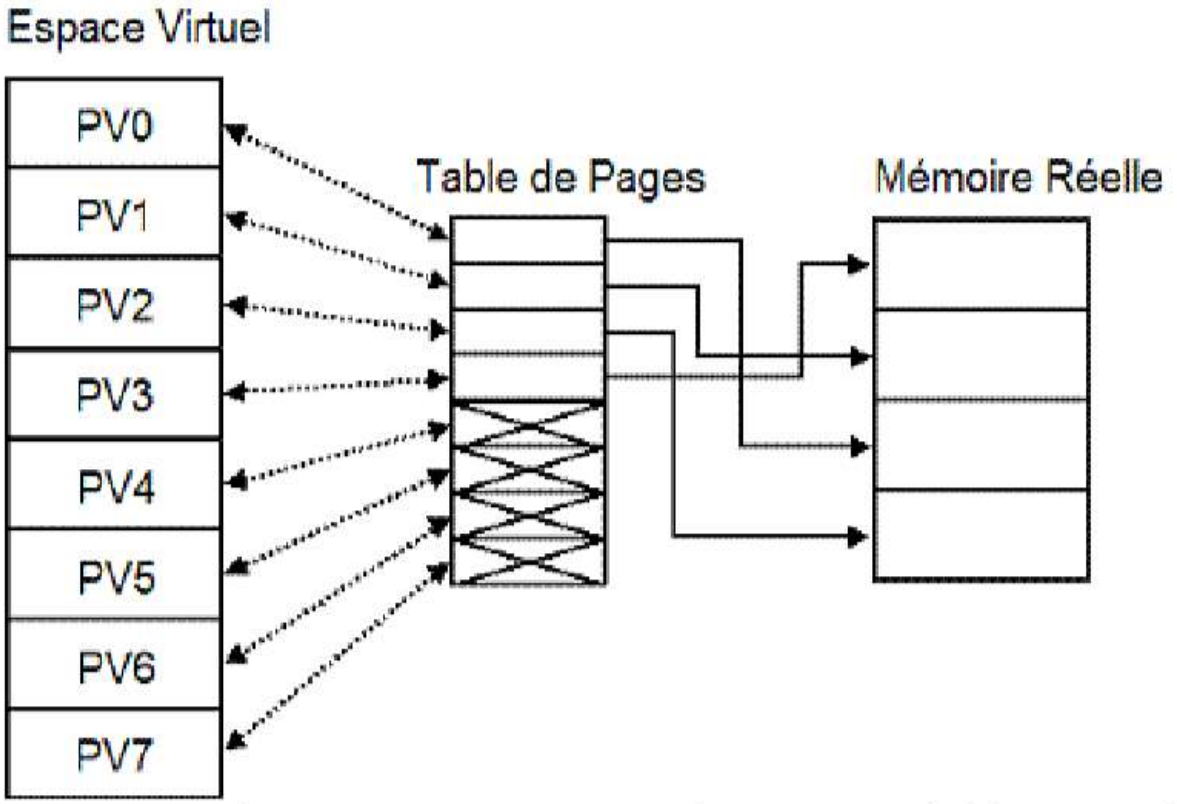
# Multiprogrammation et partitions multiples non contigües



## 1. Pagination

### Table des pages

- La traduction des adresses utilise une table des pages qui est située en mémoire centrale ou dans des registres, et dans laquelle les entrées successives correspondent aux pages virtuelle consécutives.
- La *p-ième* entrée de la table des pages contient le numéro *r* de la case où est implantée la page *p*, et éventuellement des indicateurs supplémentaires.




*Correspondance entre espace virtuel et espace réel [crocus75]*

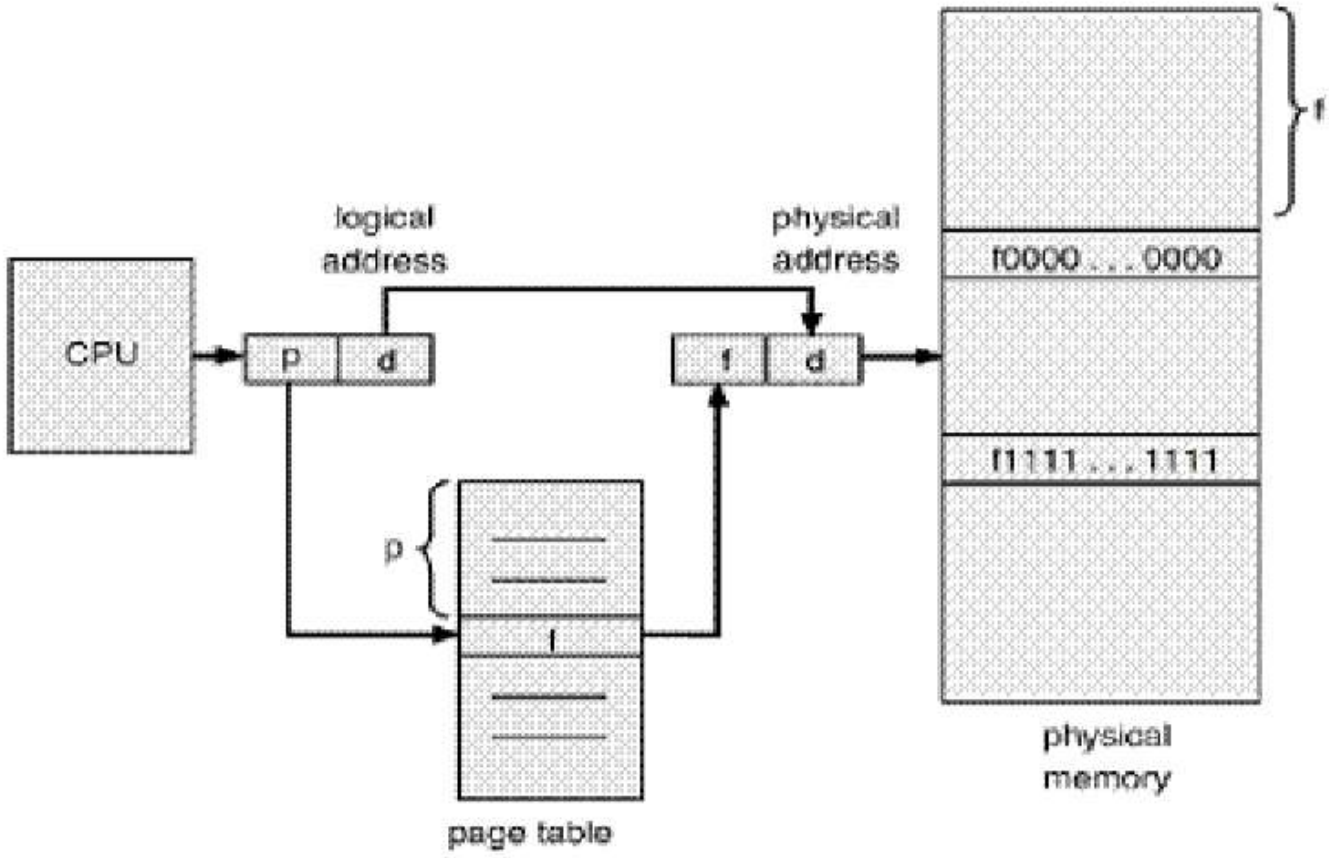
# Multiprogrammation et partitions multiples non contigües



## 1. Pagination

### Traduction d'adresse

- L'adresse réelle (*f,d*) d'un mot d'adresse virtuelle (*p,d*) est obtenue en remplaçant le numéro de page *p* par le numéro de case *f* trouvé dans la *p-ième* entrée (voir figure (cf. 'Traduction d'une adresse virtuelle en une adresse réelle).



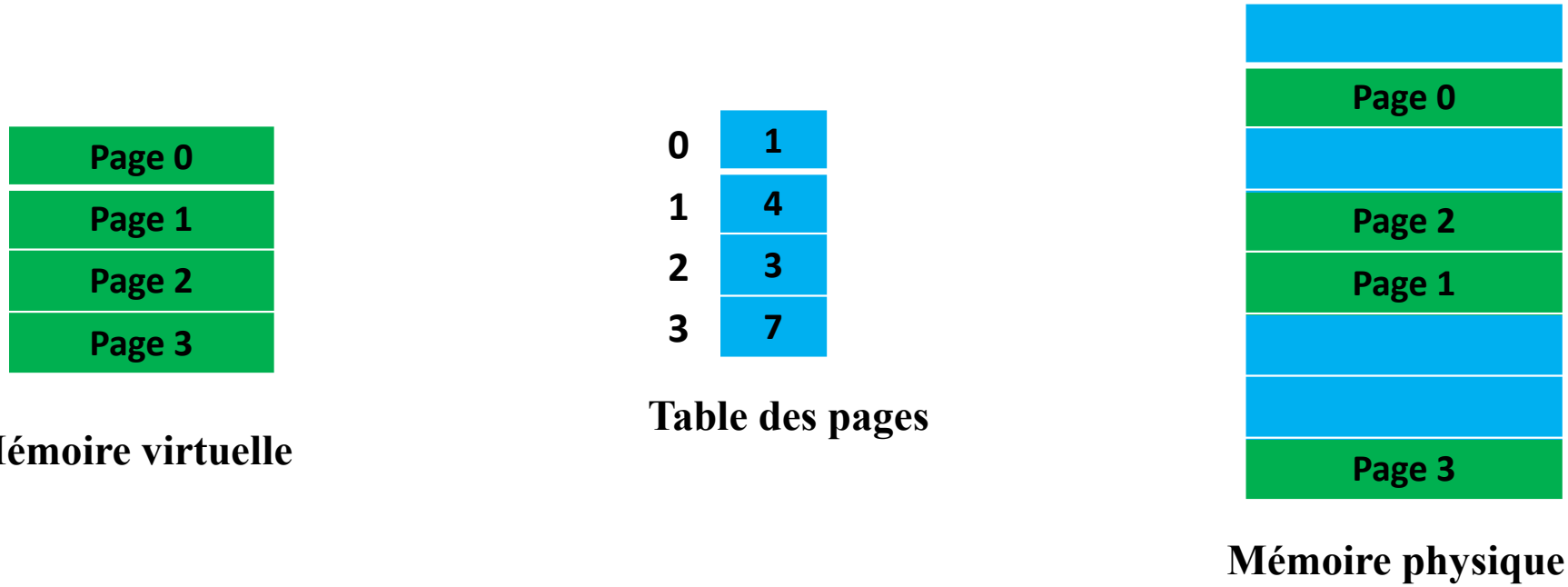

*Traduction d'une adresse virtuelle en une adresse réelle*

# Multiprogrammation et partitions multiples non contigües



## 1. Pagination

### Traduction d'adresse : exemple



# Multiprogrammation et partitions multiples non contigües



## 1. Pagination

### Exemple : Mémoire topographique du CII 10070

Dans la machine CII 10070, une adresse virtuelle occupe 17 bits, ce qui permet d'adresser 128K mots. La taille maximale de la mémoire réelle est également de 128K mots. L'espace virtuel est découpé en 256 pages de 512 mots. Les 8 bits de gauche de l'adresse virtuelle fournissent le numéro de page et les 9 bits de droite le déplacement dans la page. La traduction dynamique des adresses est réalisée par une "mémoire topographique" constituée de 256 registres (de 8 bits) pouvant contenir chacun un numéro de case. Un verrou d'accès virtuel de 2 bits est associé à chaque entrée de la mémoire topographique; sa signification est la suivante:

- 00 : tout accès autorisé,
- 01 : écriture interdite,
- 10 : lecture interdite,
- 11 : aucun accès permis

# Multiprogrammation et partitions multiples non contigües



## 1. Pagination

- La plupart des systèmes informatiques modernes utilisent des espaces d'adresses logiques de 2^32 à 2^64
- Avec 2^32 addresses et des tailless de pages de 4Ko ( 2^12 ), Cela crée 2^20 entrées dans le table des pages.
- À 4 octets par entrée, cela équivaut à une table de pages de 4 Mo, qui est trop grande pour la garder dans la mémoire contiguë.
- Notez qu'avec des pages de 4Ko, cela prendrait 1024 pages uniquement pour contenir la table des pages

# Multiprogrammation et partitions multiples non contigües



## 1. Pagination

### b) Pagination à deux niveaux

- La pagination à un niveau d'un espace virtuel de grande taille peut conduire à des tailles importantes de la table des pages
- La pagination à deux niveaux a pour but de réduire la représentation de la table des pages d'un processus, sans avoir des contraintes aussi fortes
- Un adresse virtuelle est composée de : numéro de page maître, numéro de page secondaire, décalage (*d*)
- La table de page maîtres répertorie les adresses virtuelles vers la table de la page secondaire

# Multiprogrammation et partitions multiples non contigües



## 1. Pagination

### b) Pagination à deux niveaux

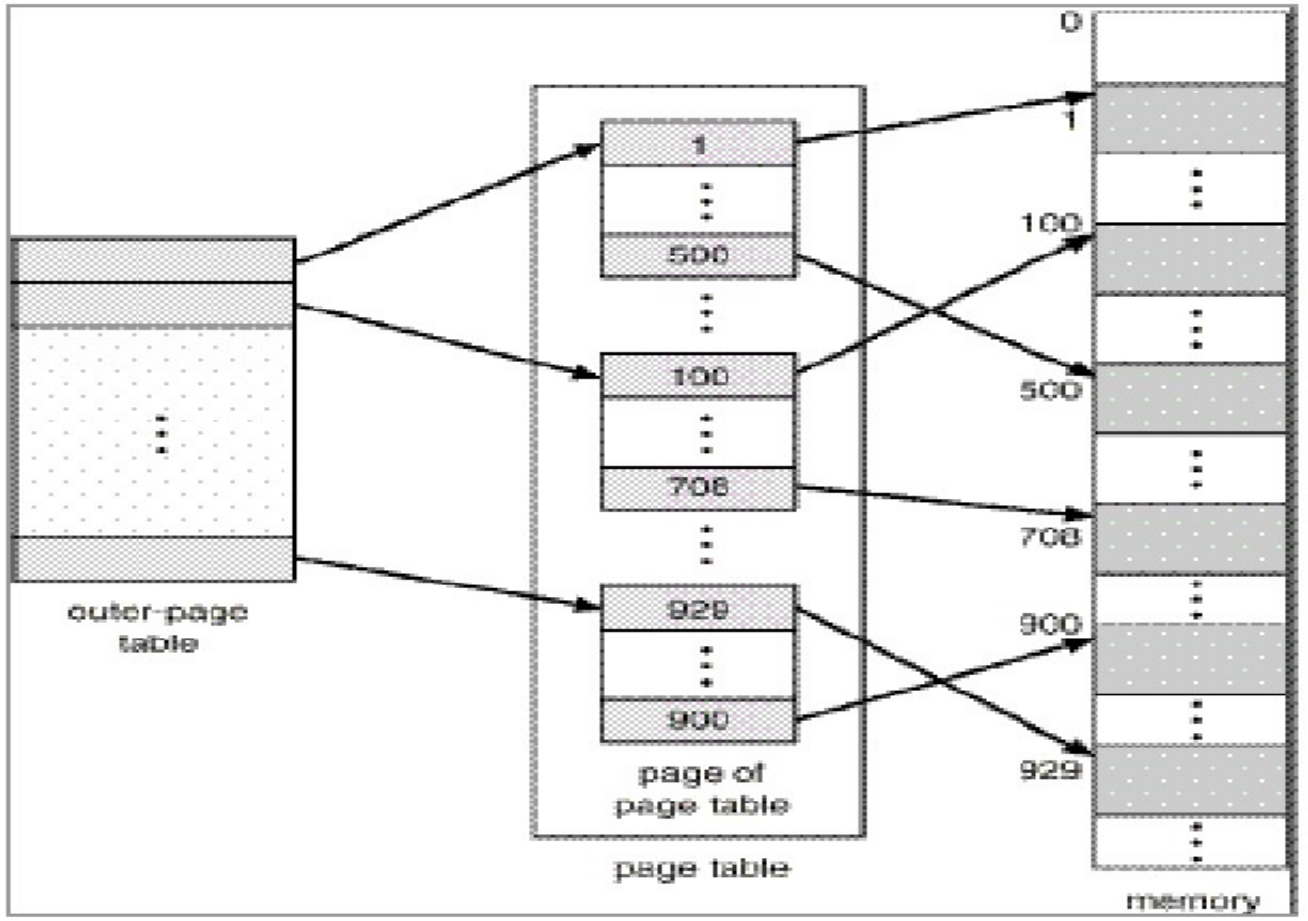



*Table de pages hiérarchique*

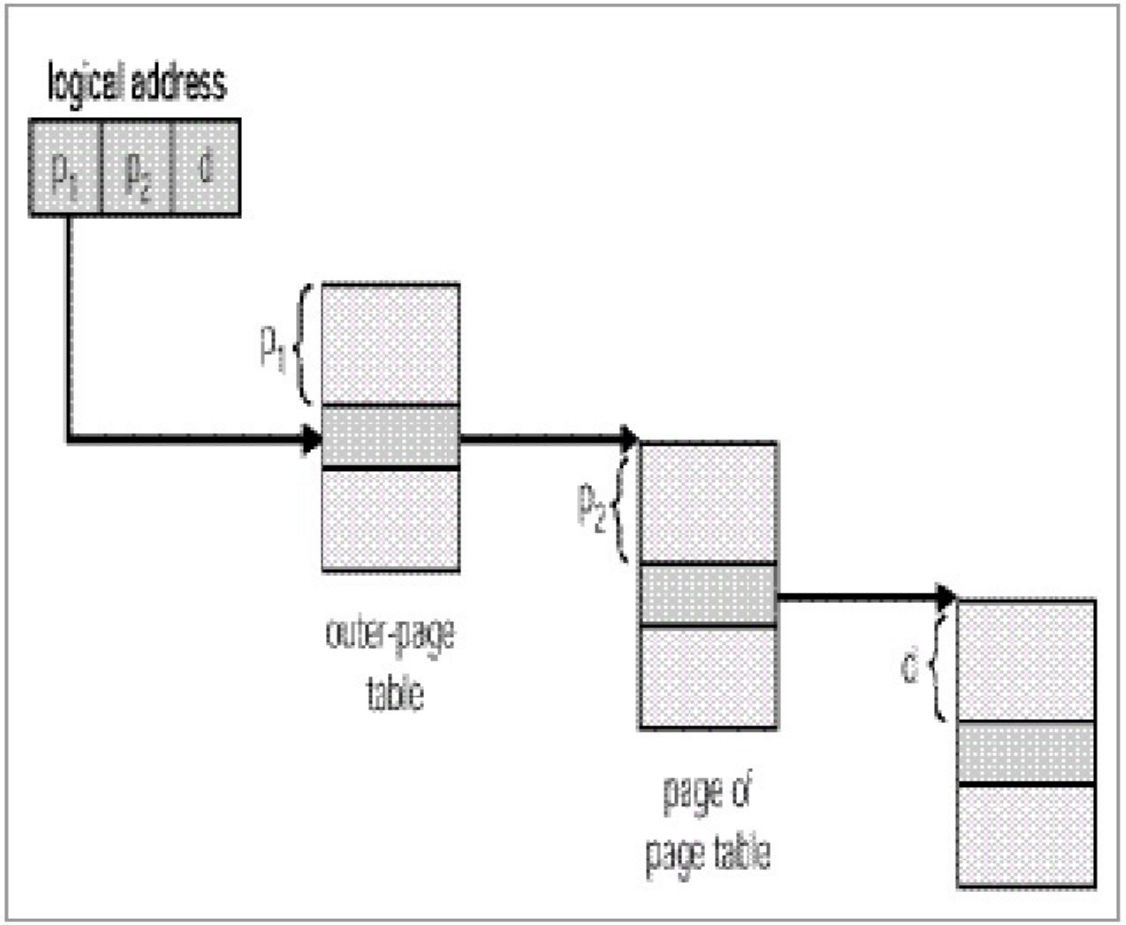
# Multiprogrammation et partitions multiples non contigües



## 1. Pagination

**Pagination hiérarchiques** :

- Optimisation de la taille des tables de page
- Pour éviter d'avoir des tables de pages trop longues, des tables de plusieurs niveaux (hiérarchiques) peuvent être utilisés.



# Multiprogrammation et partitions multiples non contigües



## 1. Pagination

### Remarque : Pagination et fragmentation

- Avec la pagination, il n y a plus de fragmentation externes (s'il y a suffisamment de mémoire (de pages) non allouée à un programme il les aura), par contre la fragmentation interne persiste: si un programme nécessite 2 pages et 1 octet, il aura 3 pages.

# Multiprogrammation et partitions multiples non contigües



## 2. Segmentation

- La segmentation est analogue à la pagination sauf que la taille d'un segment est variable.
- L'espace d'adressage logique est divisé en un ensemble de segments.
- L'avantage de la segmentation par rapport à la pagination, est que les segments peuvent refléter une vision logique du programme.
- Par exemple chaque segment peut représenter un module de programme généré au moment de la compilation.

**Exemple :**

- Segment de code / Segment de données / Segment de pile

# Multiprogrammation et partitions multiples non contigües



## 2. Segmentation

- La segmentation doit être vue comme une structuration de l'espace des adresses d'un processus
- Alors que la pagination doit être vue comme un moyen d'adaptation de la mémoire virtuelle à la mémoire réelle.
- Avec la segmentation, le processus dispose d'un espace des adresses à deux dimensions que le processeur transforme en une adresse dans une mémoire virtuelle.
- Sans la segmentation, le processus dispose directement de cette mémoire virtuelle.

# Multiprogrammation et partitions multiples non contigües



## 2. Segmentation

### Table de segments

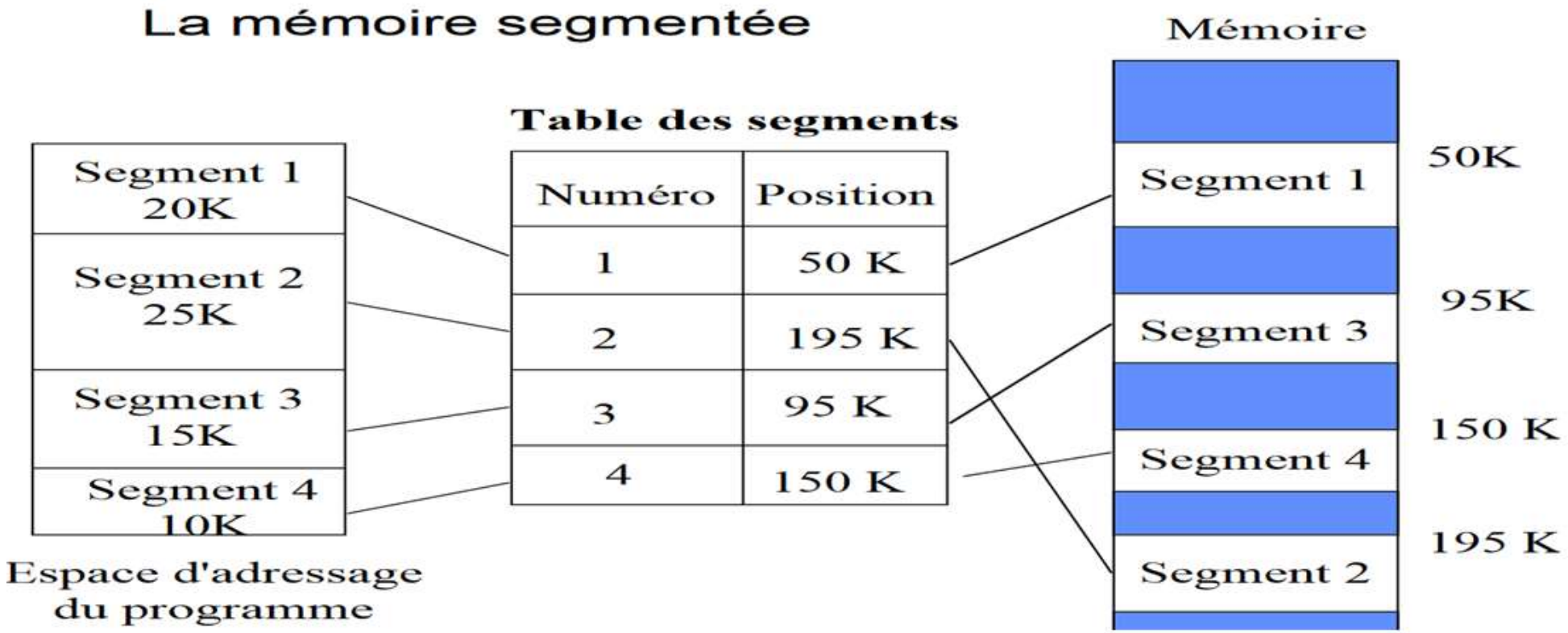
- L'espace d'adressage physique correspondant peut ne pas être contigu.
- Une table de segment spécifie pour chaque segment deux valeurs :
  - base : adresse début du segment en mémoire centrale
  - limite : taille du segment
- Une adresse logique est dite segmentée. Elle comprend un numéro de segment *(s)* et un déplacement dans le segment *(d)*. *(s)* est utilisé comme index dans la table de segments.
- La table de segments est généralement implantée par des registres rapides. Si *d* > *limite* : alors erreur de débordement (fameux Segmentation fault)

# Multiprogrammation et partitions multiples non contigües



## 2. Segmentation

### Table de segments



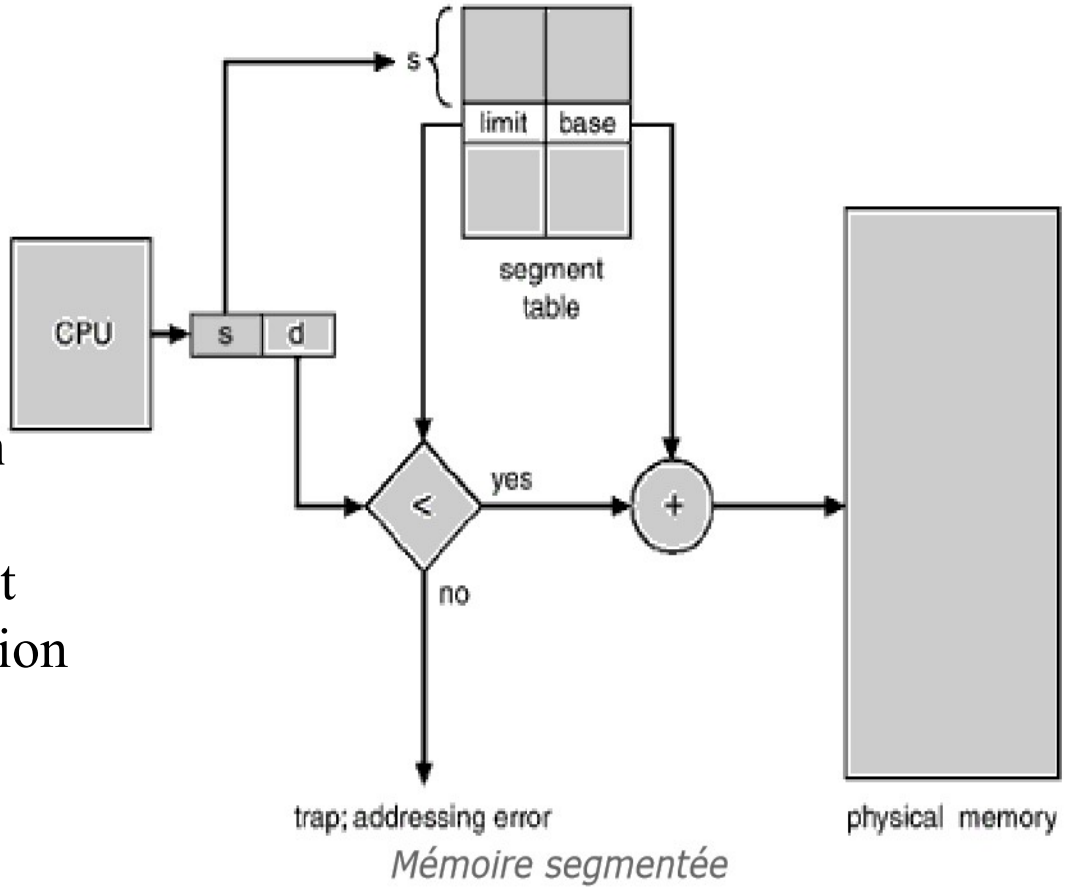
# Multiprogrammation et partitions multiples non contigües



## 2. Segmentation

**Méthode : Traduction d'adresse**
La figure le mécanisme de traduction d'adresses segmentées et le mécanisme de protection.

**Remarque : Segmentation et fragmentation externe**
Avec la segmentation la taille des segments est variable ce qui peut engendrer une fragmentation externe. Ceci peut être résolu en combinant la segmentation et la pagination.



Mémoire segmentée

# Multiprogrammation et partitions multiples non contigües



## 3. Segmentation paginée

- Dans la cas où la pagination et la segmentation sont simultanément employées, une table des segments est définie pour chaque segment de l'espace d'adressage du processus.

- Chaque segment est à son tour paginé, il existe donc une table des pages pour chaque segment

- Ainsi une entrée de la table des segment ne contient plus l'adresse du segment correspondant en mémoire physique mais contient l'adresse de la table des pages en mémoire physique pour ce segment

# Multiprogrammation et partitions multiples non contigües



## 3. Segmentation paginée

### Adressage en segmentation paginée

- L'adressage est composé de deux niveaux :
  - Le niveau segment et
  - Le niveau page
- Chaque segment est un espace linéaire d'adressage logiques. Il est découpé en pages. Une adresse est formée de trois parties : (*s, p, d'*)
  - s : numéro du segment
  - p : numéro de page
  - d' : déplacement dans la page

# Multiprogrammation et partitions multiples non contigües

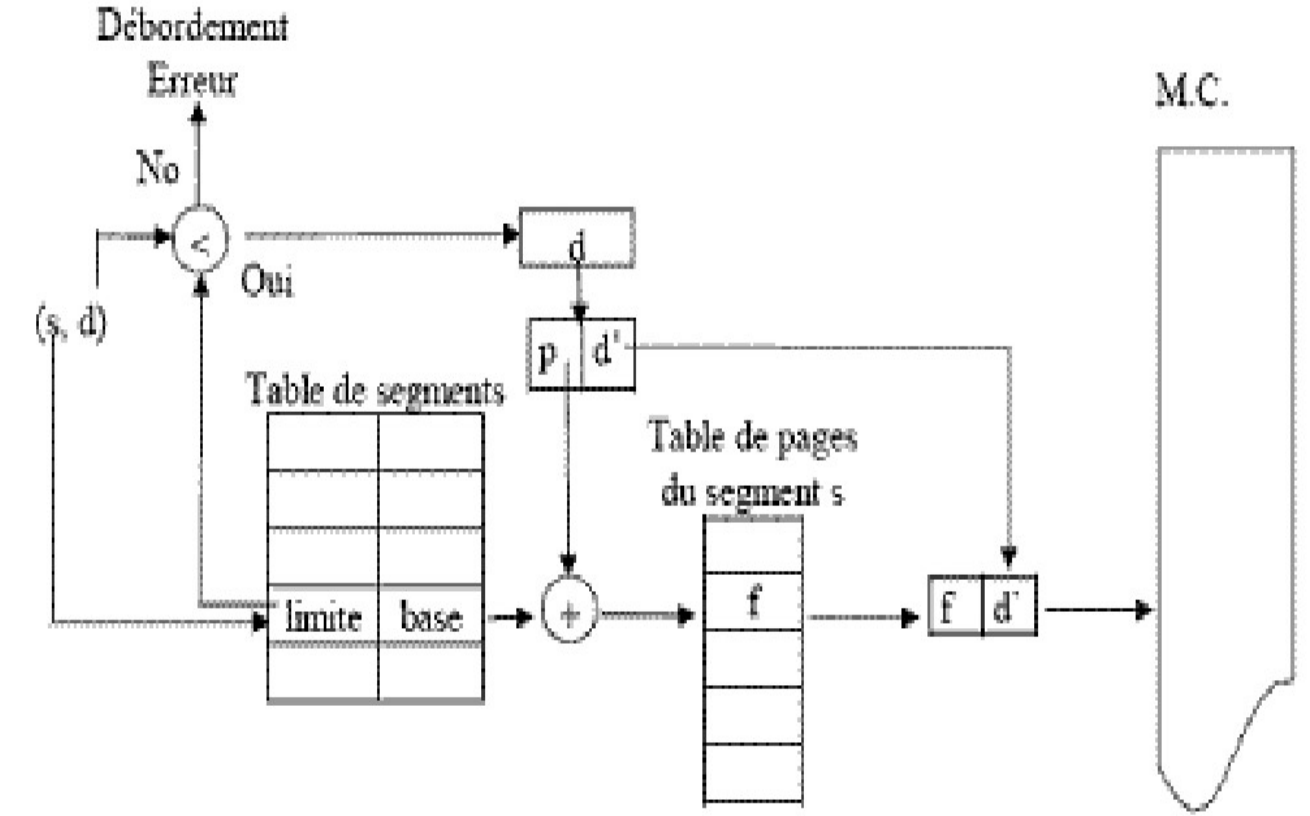


## 3. Segmentation paginée

**Méthode : Traduction d'adresse**

- La table de page est divisée en portions. Chaque portion concerne un segment bien précis.

- La base permet de retrouver la portion concernant le segment donné, et p donne le numéro de page dans cette portion.




Segmentation paginée

# Mémoire virtuelle



- La mémoire virtuelle est une technique permettant d'exécuter un programme partiellement chargé en mémoire centrale ; la taille du programme étant nettement supérieure à celle de la mémoire centrale.
- Pour un processus, la mémoire virtuelle est le support de l'ensemble des informations potentiellement accessibles.
- C'est donc plus précisément, l'ensemble des emplacements dont l'adresse peut être engendrée par le processeur.
- L'allocation de mémoire consiste à concrétiser cette mémoire virtuelle par des supports physiques d'information tels que mémoire principale (RAM), disques magnétiques, etc
- En dernier ressort, l'accès d'un processus à une information (de son espace virtuel) est concrétisé par l'accès d'un processeur physique à un emplacement de mémoire principale (RAM) adressable par ce processeur

# Mémoire virtuelle



- La mémoire virtuelle repose sur le principe suivant : ***la taille de l'ensemble formé par le programme, les données et la pile peut dépasser la capacité disponible de la mémoire physique.***
- Le système d'exploitation conserve les parties de programme en cours d'utilisation dans la mémoire principale, et le reste sur le disque.

**Exemple**

- Un programme de *16 Mo* peut s'exécuter sur une machine de *4 Mo* de mémoire si les *4 Mo* à garder en mémoire à chaque instant sont choisis avec attention, et que des parties du programme passent du disque à la mémoire, à la demande.

**Mémoire virtuelle et E/S**

- Quand un programme attend le chargement d'une partie de lui-même, il est en attente d'E/S et ne peut s'exécuter; le CPU peut par conséquent être donné à un autre processus, de la même façon que pour tout autre système multiprogrammé

# Mémoire virtuelle



**Implémentation de la mémoire virtuelle**

- Il existe deux techniques permettant d'implanter une mémoire virtuelle :

  Les **"overlays"** et la **"pagination à la demande"**

- Dans les deux cas le manque en mémoire centrale est compensé par la mémoire secondaire.

- Le principe des deux méthodes consiste à ne maintenir en mémoire centrale que les instructions et les données nécessaires à l'exécution d'un programme à un instant *t*.

# Mémoire virtuelle



## Overlays (segments de recouvrement)

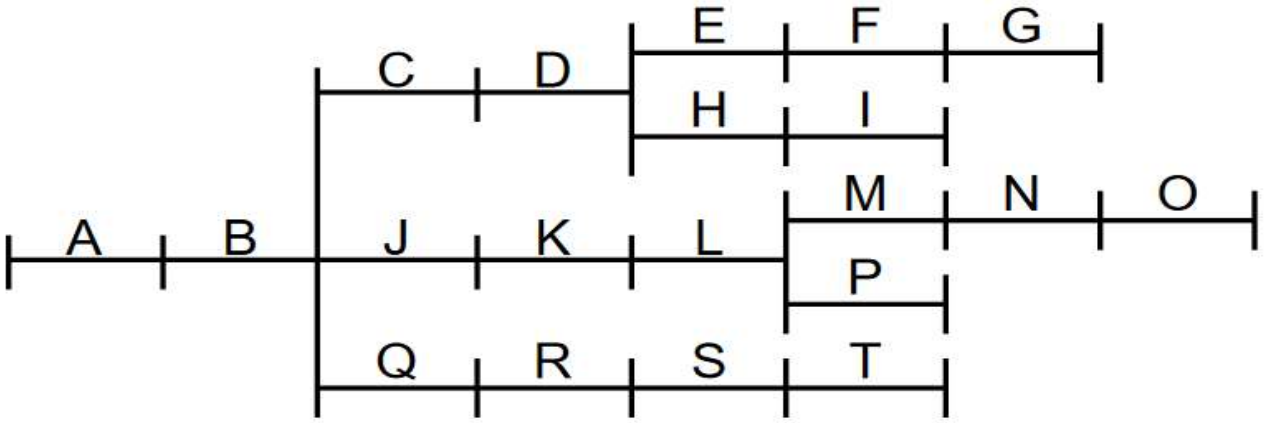
- Au début de l'informatique, il arrivait que les programmes soient trop importants pour être supportés par la mémoire disponible.
- La solution la plus courante consistait à diviser le programme en parties, appelées *segments de recouvrement* ("overlays").
- Le segment 0 était exécuté en premier. Quand cela était réalisé, il pouvait appeler un autre segment. Les segments de recouvrement étaient conservés sur le disque et chargés en mémoire par le système d'exploitation, dynamiquement à la demande
- Les différentes passes d'un compilateur sont souvent réalisées en utilisant un overlay (préprocesseurs, pass1, pass2, pour les compilateurs C).

# Mémoire virtuelle



## Overlays (segments de recouvrement)

- En général, le découpage et le recouvrement sont indiqués sous la forme d'un arbre, dont la figure suivante est un exemple. L'éditeur de liens déduira de cet arbre que, entre autre, les modules E et H peuvent être implantés au même endroit, de même que les modules C , J et Q .



# Mémoire virtuelle



## Overlays (segments de recouvrement)

- Cette technique n'a plus cours maintenant.
- Elle était utilisée à l'époque du DOS pour des applications volumineuses.
- Le programmeur devait prévoir le découpage de son application en imaginant comment ces overlays serait chargés en mémoire les uns à la suite des autres pour qu'au cours de son exécution l'application puisse atteindre toutes les fonctions nécessaires.
- C'était en quelque sorte comme cela qu'on concevait la mémoire virtuelle à l'époque des systèmes d'exploitation mono-tâches.

# Mémoire virtuelle



## Pagination à la demande

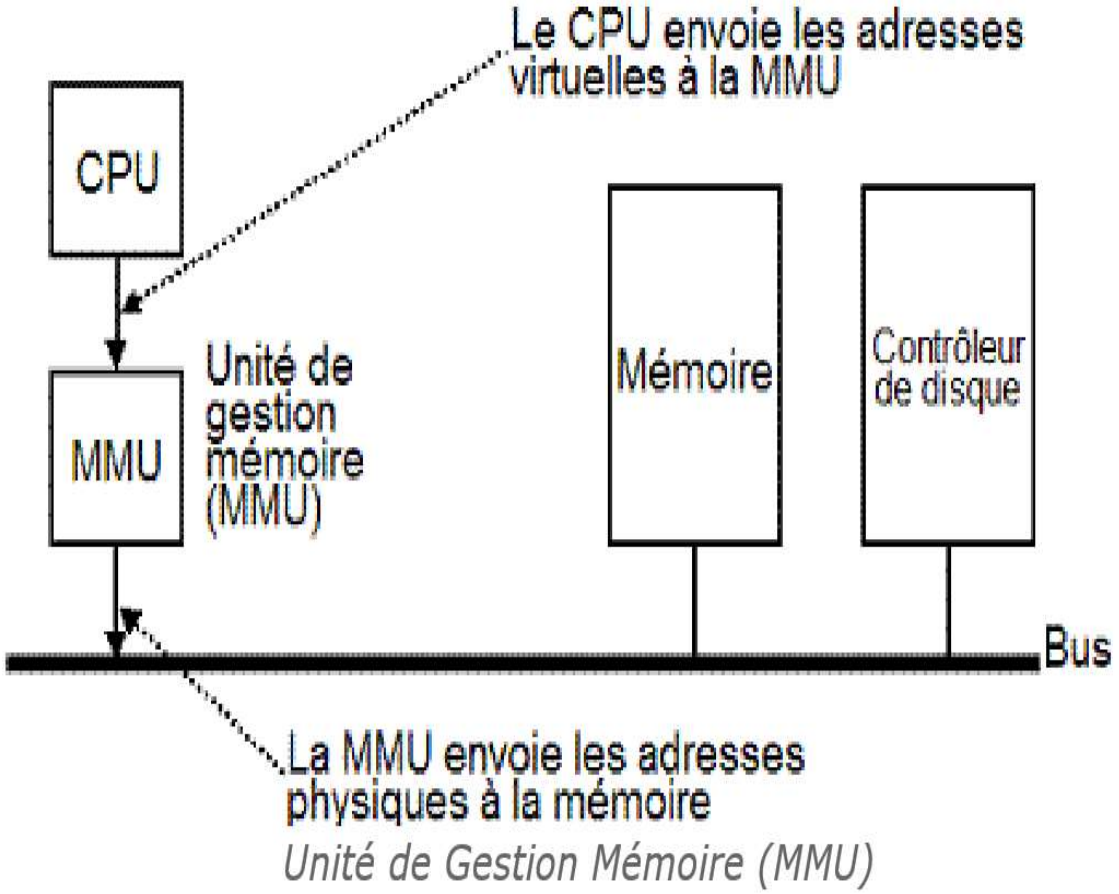
- La mémoire virtuelle peut être de taille beaucoup plus grande que la mémoire physique.
- Il faut alors mettre en mémoire physique uniquement les pages de mémoire virtuelle dont les processus ont besoin pour leur exécution courante, les autres étant conservées sur mémoire secondaire (disque).
- Lorsqu'un processus accède à une page qui n'est pas en mémoire physique, l'instruction est interrompue, et un déroutement au système est exécutée. On dit qu'il y a **défaut de page**.
- Le système doit alors charger cette page, cette méthode qui consiste à charger une page uniquement lorsqu'elle est demandée s'appelle pagination à la demande (demand paging).

# Mémoire virtuelle



## Pagination à la demande

- Dans un système paginé, les adresses générées par un programme sont appelées des adresses virtuelles et elles forment l'espace d'adressage virtuel.
- Ces adresses virtuelles ne vont pas directement sur le bus mémoire mais dans une unité de gestion mémoire (***MMU, Memory Management Unit***) qui fait correspondre les adresses virtuelles à des adresses physiques, en se basant sur une table de page.




*Unité de Gestion Mémoire (MMU)*

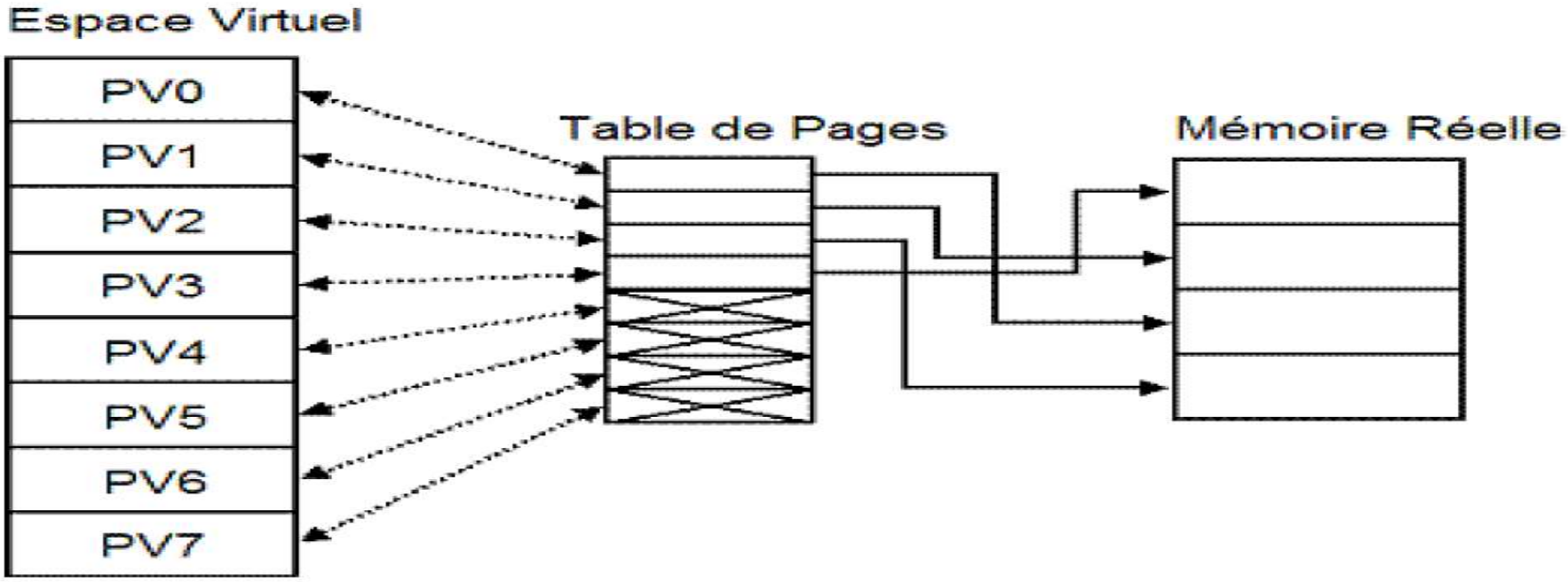
# Mémoire virtuelle



## Pagination à la demande

### Traduction d'adresse et bit de présence

- La figure suivante donne un exemple très simple de la manière dont se fait le travail de correspondance dans une MMU à travers une table de pages.



# Mémoire virtuelle



## Pagination à la demande

### Traduction d'adresse et bit de présence

- Cependant, cette capacité de mettre en correspondance les 8 pages virtuelles sur les 4 cases physiques en initialisant correctement la MMU ne résout pas le problème posé par un espace d'adressage virtuel plus grand que la mémoire physique.
- Puisqu'il n'y que 4 cases physiques, seuls les 4 des pages virtuelles de la figure sont mises en correspondance avec la mémoire physique.
- Les autres, illustrées par des croix dans la figure, ne sont pas mises en correspondance.
- En terme de matériel, **un bit de présence/absence** conserve la trace des pages qui se trouvent physiquement en mémoire.
- **bit de présence/absence :** 1: page en mémoire (RAM), 0: sur disque

# Mémoire virtuelle



## Pagination à la demande

### Défaut de page

- Que se passe-t-il si le programme essaye par exemple de faire appel à une page non présente ? La MMU remarque que la page est absente (ce qui est indiqué par le bit de présence) et fait procéder le CPU à un déroutement, c'est-à-dire que le processeur est restitué au système d'exploitation. Ce déroutement, appelé **défaut de page.**

- **Le défaut de page est un déroutement qui oblige le processeur à suspendre l'exécution du programme en cours pour lancer une entrée/sortie qui charge la page manquante en mémoire centrale dans une case libre.**

# Mémoire virtuelle



## Pagination à la demande

### Défaut de page

- Le déroutement d'un défaut de page est réalisé de la manière suivante :

  - le système d'exploitation sélectionne une case peu utilisée et écrit son contenu sur le disque;
  - il transfère ensuite la page qui vient d'être référencée dans la case libérée,
  - modifie la correspondance et
  - recommence l'instruction déroutée

# Mémoire virtuelle



## Pagination à la demande

Défaut de page
Le programme gère une adresse logique paginée (*p,d*)
SI *p* est valide
ALORS générer adresse physique
SINON /* *Défaut de page* */
Lire *p* de la mémoire secondaire /* swap in */
SI il y une case libre en mémoire centrale
ALORS
allouer cette case à *p*
mettre à jour la table de page
générer l'adresse physique
SINON /* algorithme de remplacement */
choisir une page présente en mémoire centrale
l'écrire en mémoire secondaire /* swap out */
mettre à jour le bit de validité
affecter la case libérée à la nouvelle page
et mettre à jour la table de page
générer l'adresse physique
FSI
FSI

# Mémoire virtuelle



## Algorithmes de remplacement de page

### Remplacement de page (local / global)

- 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.

### Définition : Chaîne de références

- On appelle chaîne de références la séquence des numéros de pages référencées par un programme durant une exécution.
- 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.

# Mémoire virtuelle



## Algorithmes de remplacement de page

### Méthode : Algorithme optimal

- Pour une chaîne de référence donnée, on peut montrer que l'algorithme suivant minimise le nombre total de défauts de pages :

***lors d'un défaut de page, choisir comme victime une page qui ne fera l'objet d'aucune référence ultérieure, ou, à défaut, la page qui fera l'objet de la référence la plus tardive.***

✓*Remplacer la page qui ne sera pas utilisée pendant la durée la plus longue*

✓Cet algorithme suppose une connaissance de l'ensemble de la chaîne de référence; il est donc irréalisable en temps réel. Il permet d'évaluer, par comparaison, les autres algorithmes

# Mémoire virtuelle



## Algorithmes de remplacement de page

### Tirage Aléatoire

- La victime est choisie au hasard (loi uniforme) parmi l'ensemble des pages présentes en mémoire.
- Cet algorithme n'a aucune vertu particulière, car il ne tient aucun compte du comportement observé ou prévisible du programme.
- il sert surtout de point de comparaison

# Mémoire virtuelle



## Algorithmes de remplacement de page

### Ordre Chronologique de Chargement (FIFO)

- Cet algorithme choisit comme victime la page la plus anciennement chargée.
- Son principal intérêt est sa simplicité de réalisation : il suffit d'entretenir dans une file FIFO les numéros des cases où sont chargées les pages successives
- Simple à comprendre et à implémenter
- Performance de FIFO n'est pas toujours bonne

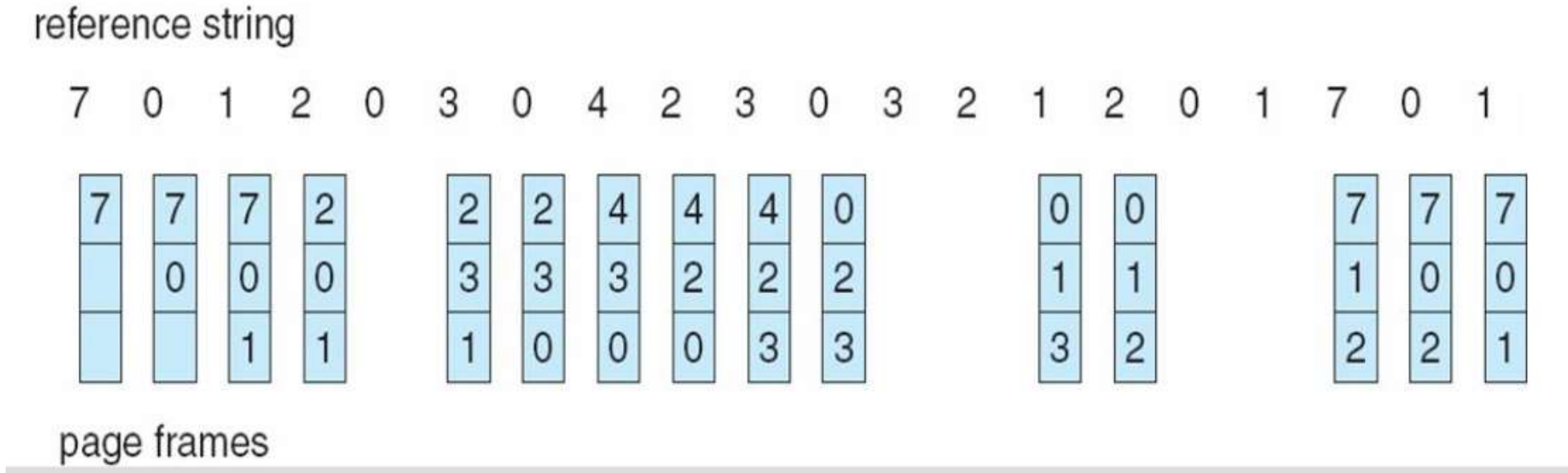
# Mémoire virtuelle



## Algorithmes de remplacement de page

### Ordre Chronologique de Chargement (FIFO)

- **Exemple:** On considère notre séquence de référence : **7,0,1,2,0,3,0,4,2,3,0,3,2,1,2,0,1,7,0,1**
  - On exécute l'algorithme de remplacement FIFO avec 3 cadres de pages
  - L'exécution de FIFO donne 15 défauts de pages pour cette séquence



# Mémoire virtuelle



## Algorithmes de remplacement de page

### Ordre Chronologique de Chargement (FIFO)

- **Anomalie de Belady :**

*Logiquement s'il on a plus de cases, on aura moins de défauts de pages. Belady a montré que ce n'est pas le cas.*

**Exemple : Anomalie de Belady**

- On considère un programme qui utilise 6 pages et la chaîne de référence suivante: 1, 2, 3, 4, 1, 2, 5, 1, 2, 3, 4, 5.
- Puis nous déroulons l'algorithme de remplacement de pages FIFO en utilisant une mémoire à 4 cases puis à 3 cases.
- On remarque qu'avec 3 cases on obtient moins de défauts de pages qu'avec 4 cases.
- La figure illustre le déroulement de l'algorithme dans ces deux situations:

# Mémoire virtuelle

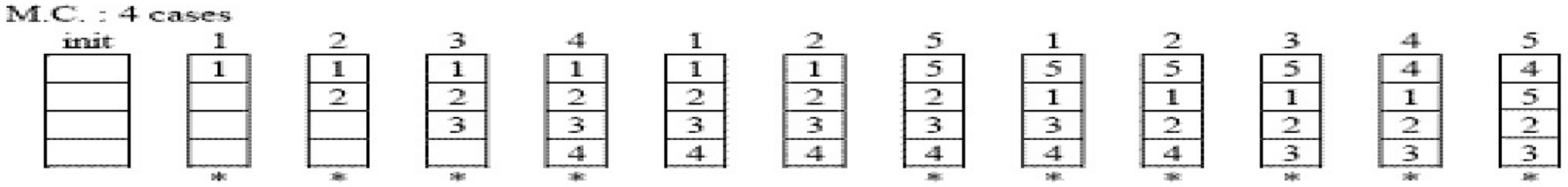


## Algorithmes de remplacement de page

### Ordre Chronologique de Chargement (FIFO)

**Exemple : Anomalie de Belady**

La figure illustre le déroulement de l'algorithme dans ces deux situations :




*Anomalie de Belady*

## Algorithmes de remplacement de page

### Ordre Chronologique d'utilisation (LRU: Least Recently Used)

- Cet algorithme tente d'approcher l'algorithme optimal, en utilisant la propriété de localité.
- Son principe s'explique comme suit: *puisque les pages récemment utilisées ont une probabilité plus élevée que d'autres d'être utilisées dans un futur proche, une page non utilisée depuis un temps élevé a une probabilité faible d'être utilisée prochainement.*
- L'algorithme choisit donc comme victime la page ayant fait l'objet de la référence la plus ancienne.
- La réalisation de l'algorithme impose d'ordonnancer les cases selon la date de dernière référence de la page qu'elles contiennent.

# Mémoire virtuelle



## Algorithmes de remplacement de page

### Ordre Chronologique d'utilisation (LRU: Least Recently Used)

- Pour cela, une information doit être associée à chaque case et mise à jour à chaque référence.
- Cette référence peut être une date de référence; une solution plus économique consiste à utiliser un compteur incrémenté de 1 à chaque référence; la case dont le compteur a la valeur la plus faible contient la victime.
- En raison de son coût, cette solution n'a été utilisée que sur des installations expérimentales.
- Si la taille du compteur est réduite à un bit, on obtient l'algorithme suivant, dont le coût est acceptable

## Algorithmes de remplacement de page

### L'algorithme NRU (Not recently used)

- L'algorithme NRU consiste à éliminer une page qui n'a pas été utilisée récemment.
- Pour cela, il repose sur le bit R mis à jour par le matériel.
- Régulièrement, le bit est mis a 0 sur toutes les pages actives.
- Lorsqu'une page doit être libérée, une page ayant encore son bit a 0 est choisie (et si aucune n'est disponible, une page ayant son bit a 1 est alors choisie).

# Mémoire virtuelle



## Algorithmes de remplacement de page

### L'algorithme NRU (Not recently used)

- Une amélioration de cet algorithme consiste à définir quatre classes de pages comme suit :
  1. Non référencées, non modifiées.
  2. Non référencées, modifiées.
  3. Référencées, non modifiées.
  4. Référencées, modifiées.

- Les pages de la classe 1 sont choisies en priorité (elles ne nécessitent pas d'écriture sur le disque, et n'ont pas été utilisées), puis celles de la classe 2, et ainsi de suite.

# Mémoire virtuelle



## Algorithmes de remplacement de page

### Algorithme de la seconde chance

- On peut apporter une modification simple à l'algorithme FIFO afin d'éviter la suppression d'une page à laquelle on a couramment recours.
- Il s'agit d'inspecter le bit R de la page la plus ancienne.
- S'il est à 0, la page est à la fois ancienne et inutilisée, elle est donc remplacée immédiatement.
- Si le bit R est à 1, le bit est effacé, la page est placée à la fin de la liste des pages, et son temps de chargement est mis à jour comme si elle venait d'arriver en mémoire. La recherche continue alors.

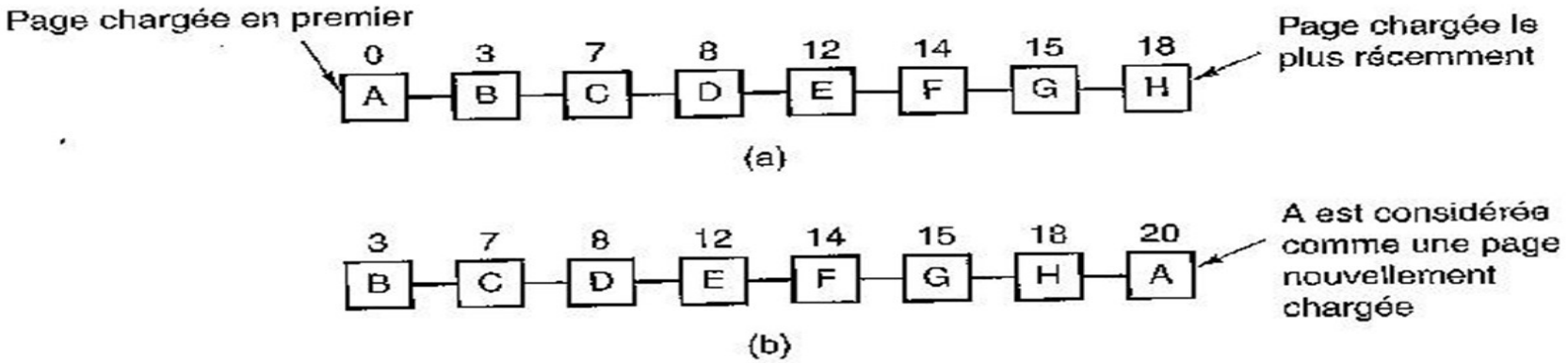
# Mémoire virtuelle



## Algorithmes de remplacement de page

### Algorithme de la seconde chance

- L'opération de cet algorithme, appelée seconde chance, est illustrée ci-dessous nous voyons que les pages de A à H se trouvent dans une liste chaînée, triées par leur temps d'arrivée en mémoire.
- Supposons qu'un défaut de page se produise à l'instant 20.



# Mémoire virtuelle



## Algorithmes de remplacement de page

### Algorithme de la seconde chance

- La page la plus ancienne est A, qui est arrivée à l'instant 0, quand le processus a démarré.
- Si le bit R de la page A est à 0, elle est évincée de la mémoire: elle est soit écrite sur le disque (si elle est modifiée), soit simplement abandonnée (si elle est restée identique à la page sur disque).
- D'un autre côté, si le bit R est à 1, A est placée à la fin de la liste et son temps de chargement est mis à jour avec le temps courant (20).
- Le bit R est aussi mis à 0. La recherche d'une page convenable continue alors avec B.

# Mémoire virtuelle



## Algorithmes de remplacement de page

### L'algorithme de remplacement de page de l'horloge

- Bien que l'algorithme de la seconde chance soit un algorithme satisfaisant, il manque d'efficacité parce qu'il déplace constamment des pages dans sa liste.

- Une meilleure approche consiste à garder tous les cadres de pages dans une liste circulaire formant une horloge, comme l'illustre la figure ci-dessous.

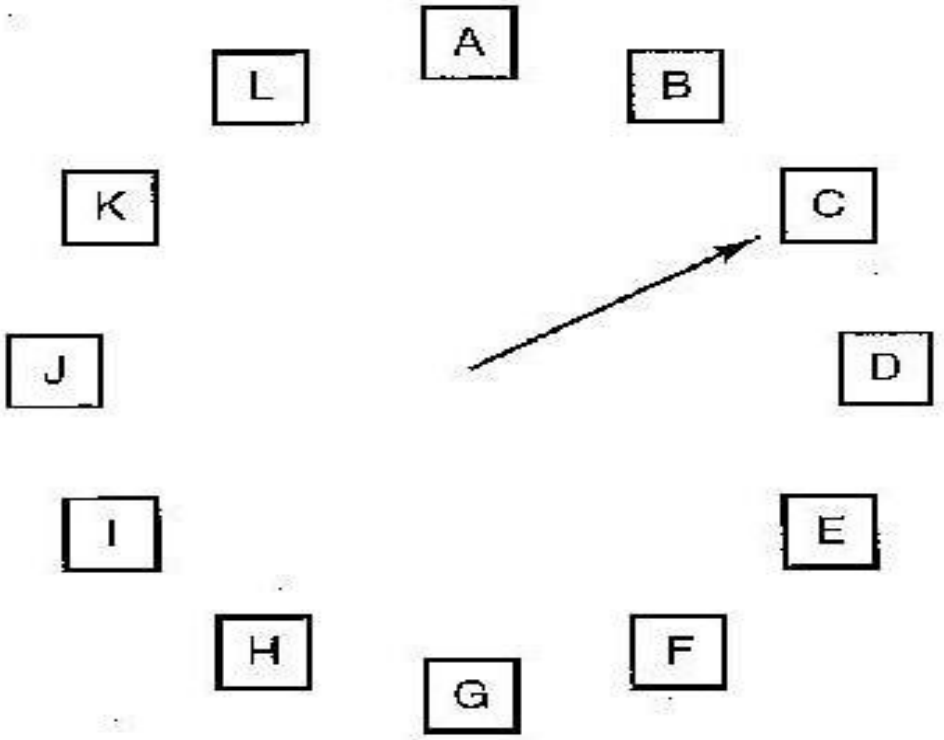
- Un pointeur se trouve sur la page la plus ancienne.

# Mémoire virtuelle



## Algorithmes de remplacement de page

### L'algorithme de remplacement de page de l'horloge



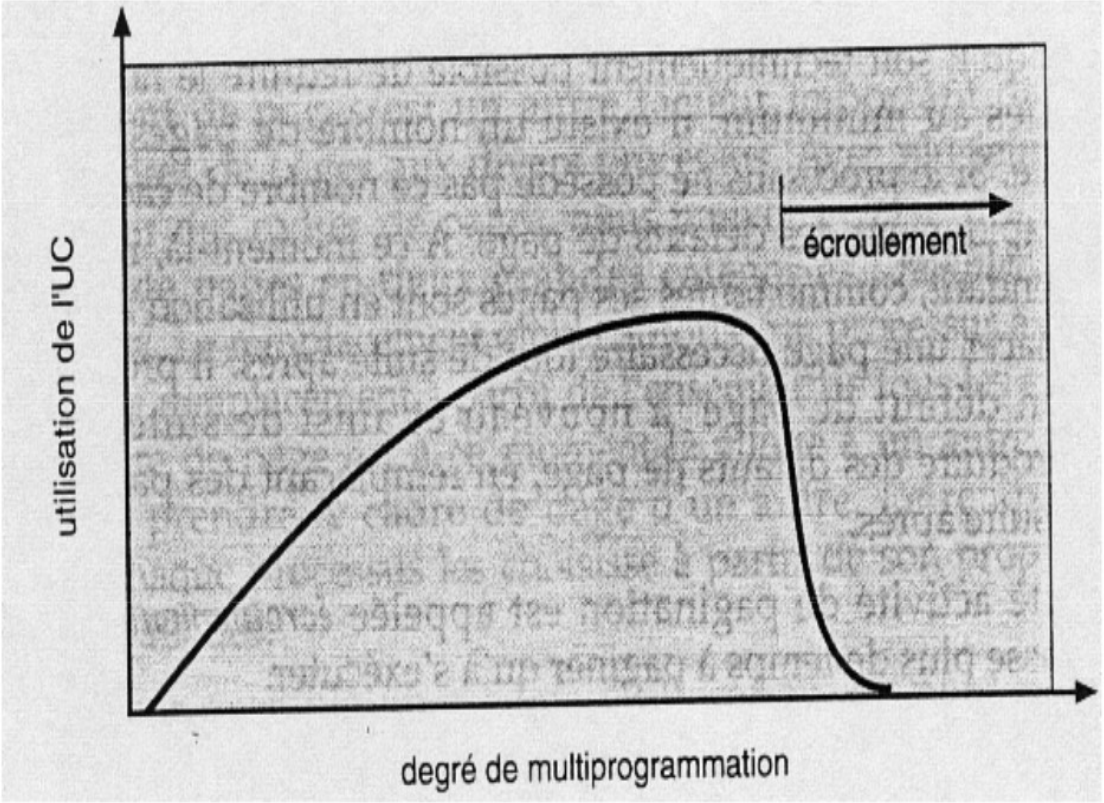

Quand un défaut de page se produit, la page pointée est testée. L'action entreprise dépend de la valeur du bit R :
R = 0 : retirer la page
R = 1 : mettre R à 0 et avancer le pointeur

# Mémoire virtuelle



## Notion d'écroulement

- On appelle Ecroulement, une haute activité de pagination. Un processus s'écroule lorsqu'il passe plus de temps à paginer qu'à s'exécuter
- La figure illustre ce phénomène.
- Elle représente le taux d'utilisation du processeur en fonction du degré de multiprogrammation, c'est-à-dire en fonction du nombre de processus chargés en mémoire centrale.



# Gestion Mémoire dans Linux



**Pagination hiérarchique**

- Dans le système Linux, la gestion mémoire est conçue selon un modèle de pagination à trois niveaux reposant sur les tables suivantes :

  - la table globale (page global directory) : elle contient les numéros des pages contenant des tables intermédiaires;

  - la table intermédiaire (page middle directory) : elle contient les numéros des pages contenant des tables des pages;

  - la table des pages (page table) : elle contient les adresses des pages mémoire contenant le code ou les données utilisés par le processus

# Gestion Mémoire dans Linux



**Pagination sous Linux avec un Pentium**

- Ce modèle est réduit à une pagination à deux niveaux sur les machines basées sur la famille de processus Pentium, en faisant disparaître les tables intermédiaires.
- En effet, ces processeurs, qui possèdent un adressage sur 32 bits, peuvent adresser un espace virtuel de 4 Go.
- Avec des pages mémoires de 4 Ko, la table des pages, dans une pagination à un niveau, contiendrait 220 entrée.

# Mémoire virtuelle



**Pagination sous Linux avec un Pentium**

- Pour éviter une aussi grande table des pages, la mémoire est gérée avec une pagination à deux niveaux. Comme on trouve dans une table des pages 1024 entrées de 4 octets, une table de pages tiendra exactement dans une page mémoire et décrit un espace de 1024 pages de 4 Ko, soit 4 Mo.
- Ainsi un programme de taille inférieure ou égale à Nx4 Mo nécessitera N tables de pages. Mais un programme chargé en mémoire n'aura pas nécessairement ses N tables en mémoire : en effet, il lui suffit pour s'exécuter d'avoir sa table des hyperpages et la table des pages correspondant aux pages en cours d'utilisation. Le bit de présence *P* dans l'entrée *k* de la table des hyperpages indique si la table des pages k est chargée en mémoire ou non.

## Algorithmes de remplacement de page

- Un ordinateur possède une mémoire de 4 pages. Pour chacune des pages, le gestionnaire de mémoire tient à jour les indicateurs suivants : date de chargement, date de dernière référence, rb (bit indiquant si la page a été référencée), mb (bit indiquant si la page a été modifiée). A un instant donné, la situation est la suivante :
- Indiquer la page qu'il faudra remplacer prochainement dans le contexte de chacune des stratégies suivantes : 1)FIFO, 2) LRU, 3) NRU

| Page | Date Chargement | Date Dern. Refer. | rb | mb |
|---|---|---|---|---|
| 0 | 126 | 279 | 0 | 0 |
| 1 | 230 | 260 | 1 | 0 |
| 2 | 120 | 272 | 1 | 1 |
| 3 | 160 | 280 | 1 | 1 |