M113
Systèmes d'Exploitation et Programmation Système

Prof. Fadoua YAKINE

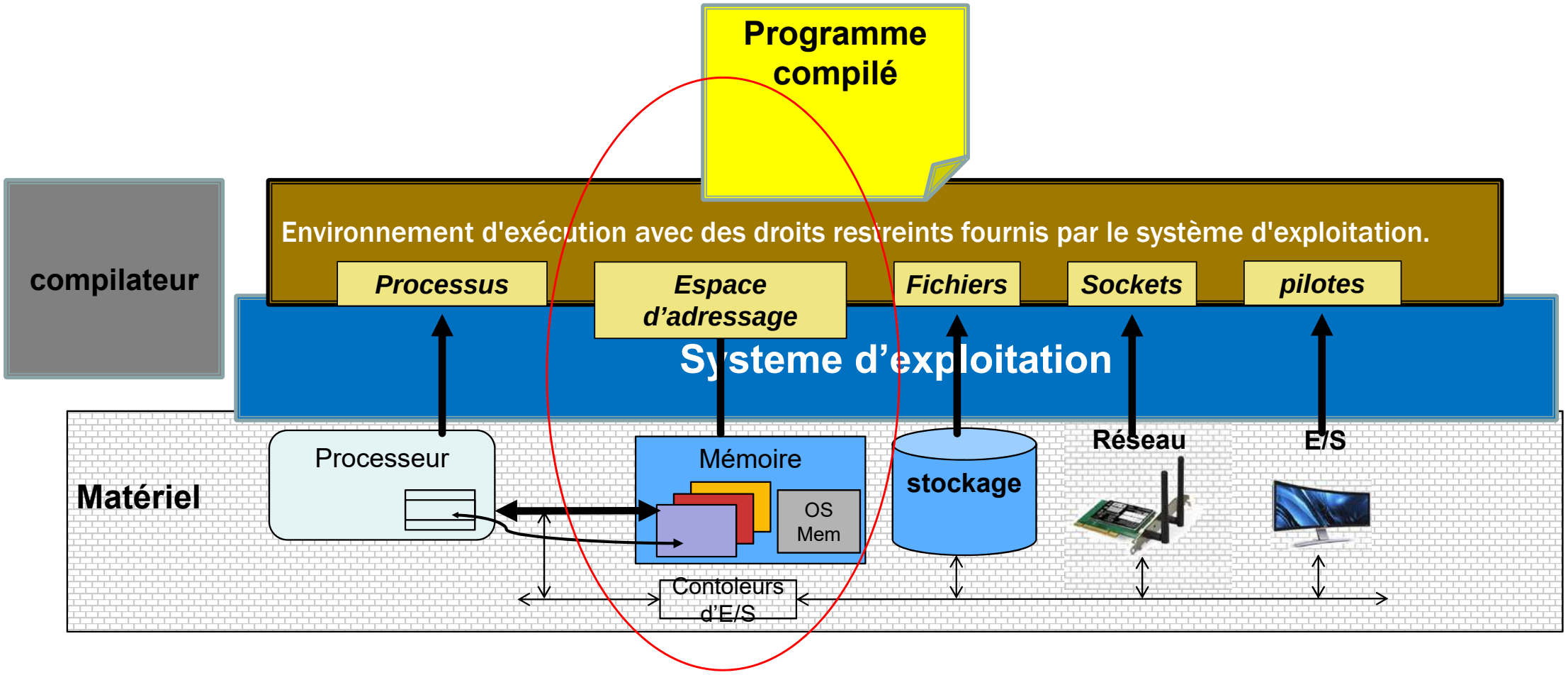
M113
Systèmes d'Exploitation et Programmation Système

Chapitre 4 : Gestion de la mémoire

23 Octobre 2024

Prof. Fadoua YAKINE

Le SE est un ensemble de gestionnaires



Introduction

- Plus que le processeur, la mémoire constitue la ressource la plus critique des systèmes d'exploitation, dont le mésusage peut avoir des effets dramatiques sur les performances globales du système.

- La gestion de la mémoire est du ressort du gestionnaire de la mémoire

Le GM

- Connaître les parties libres de la mémoire physique
- Allouer la mémoire aux processus, en évitant autant que possible le gaspillage
- Récupérer la mémoire libérée par la terminaison d'un processus ;
- Offrir aux processus des services de mémoire virtuelle, de taille supérieure à celle de la mémoire physique disponible, au moyen des techniques de va-et-vient , pagination et segmentation.

Rappels

- Une mémoire est constituée d'un assemblage de cellules mémoires élémentaires.
  - Chaque cellule permet de stocker 1 bit.
  - Les cellules sont parfois organisées en mots de N bits.
  - Chaque mot est accessible à un emplacement physique ou adresse.
  - La capacité d'une mémoire est le nombre de bits qui peuvent y être stockés.

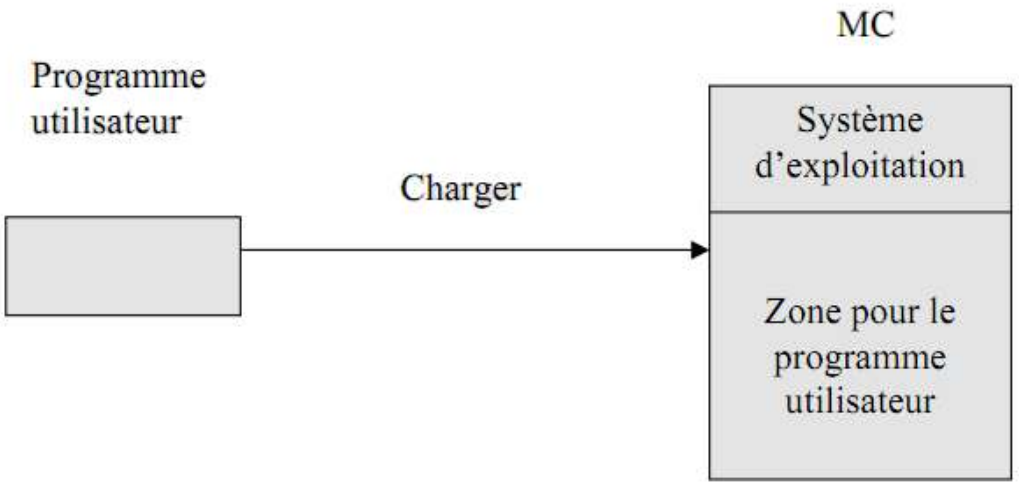
Mémoire sans va et vient ni Pagination

Système mono-programmé

Sans Recouvrement :

- Un seul processus en mémoire à un instant donnée, qui peut utiliser l'ensemble de la mémoire physique disponible.
- Il s'agit de la structuration la plus simple. Elle consiste à diviser la mémoire RAM en deux parties :
  - Partie réservée aux programmes système
  - Partie réservée à l'unique programme utilisateur entrain de s'exécuter
- A la fin de chaque programme, l'interpréteur de commande demande à l'utilisateur le prochain programme à exécuter.
- La taille du programme doit être plus petite que la mémoire disponible

Système mono-programmé



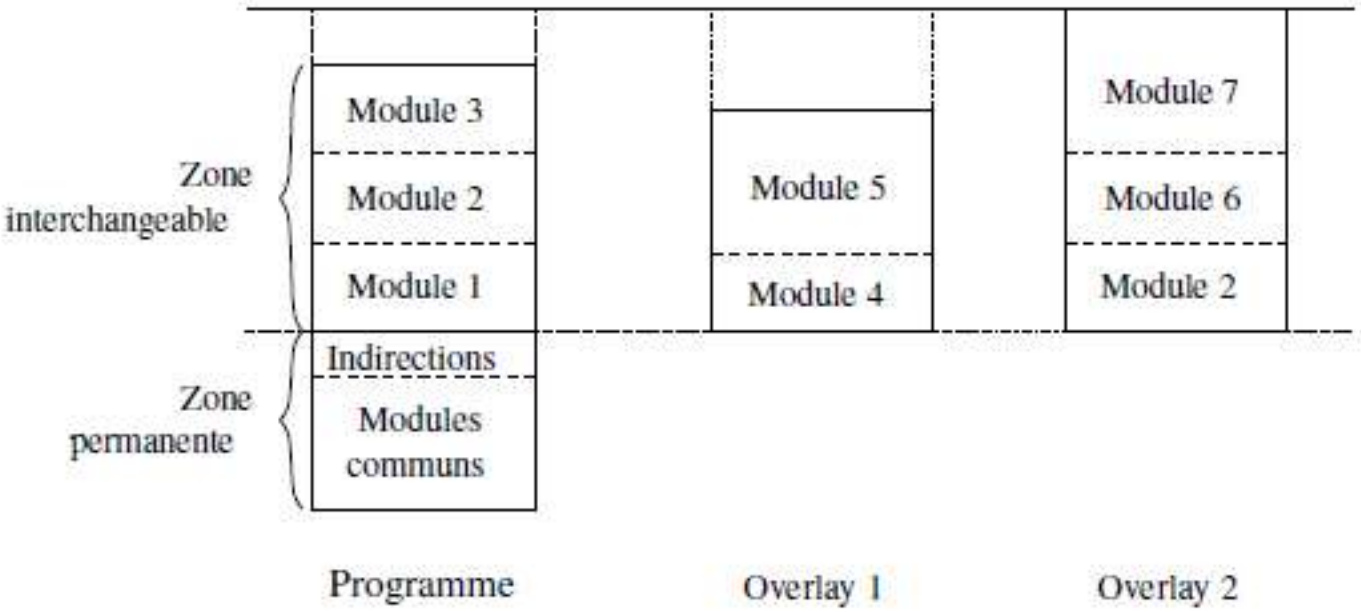
- Dans le cas où la taille du programme utilisateur dépasse la taille de la zone mémoire réservée pour le programme, la demande sera refusée.
- Avantage : simple à utiliser.

Système mono-programmé

Avec Recouvrement :

- taille(programme) > taille(mémoire)
  - Utilisation de segments de recouvrements (overlays)
- Le programme est alors découpé :
  - En zone permanente servant à la fois aux changement d'overlays et à contenir les routines les plus fréquemment.
  - Un ensemble d'overlays interchangeables correspondant le plus souvent aux grandes fonctionnalités du programme

Système mono-programmé

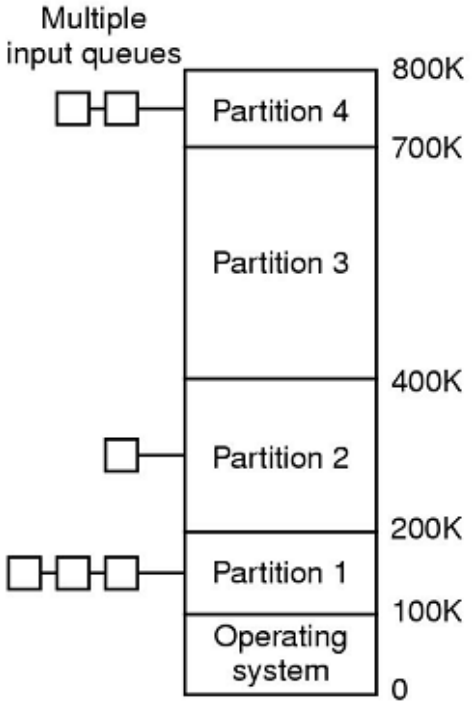


Système multiprogrammé

Partitions fixes

- Division de la mémoire en partitions (tailles inégales)
- Réalisé au démarrage de la machine
- Chaque nouvelle tâche placée dans la file d'attente de la plus petite taille pouvant la contenir
- Inconvénient :

Les files d'attente des grandes partitions sont vides



MÉMOIRE AVEC VA ET VIENT

Le va et vient

- Dans les systèmes temps partagés, la mémoire physique ne peut pas contenir tous les processus des utilisateurs.

- Il est nécessaire de placer sur le disque quelques un de ces processus et qu'il faudra les ramener en mémoire centrale pour les exécuter.
- Le mouvement des processus entre la mémoire centrale et le disque est appelé va et vient (swapping)

Multiprogrammation

Partitions variables

- Le nombre, la position et la taille varient dynamiquement au fur et à mesure que les processus entrent et sortent de la mémoire.

- Pas de limitation par un nombre fixe de partition, ou qu'elles soient trop grandes ou trop petites.

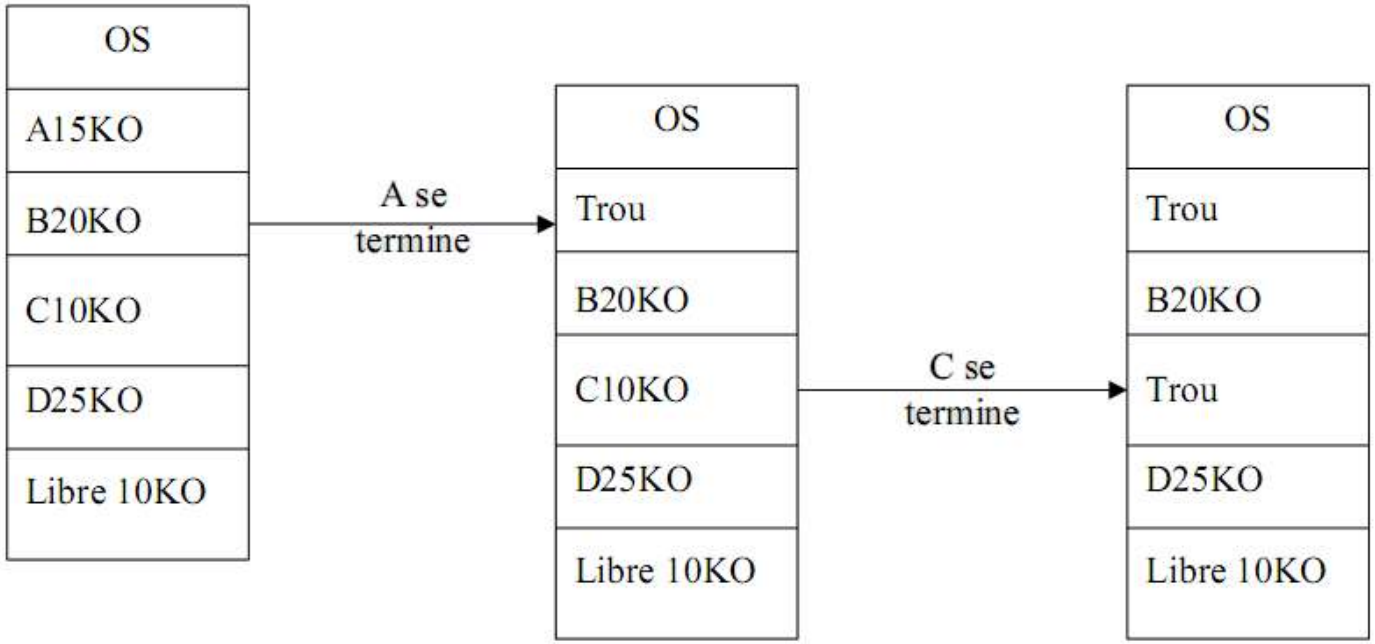
- Amélioration de l'usage de la mémoire mais cela complique son allocation et sa libération.

Exemple:

La taille d'une mémoire centrale = 100 Ko dont 20 réservés pour le SE.

G 11 KO
F 32 KO
E 25 KO
D 25 KO
C 10 KO
B 20 KO
A 15 KO

Multiprogrammation



Multiprogrammation

Trou : est la conséquence de la terminaison des processus et la libération de l'espace qu'ils ont pu occuper lors de leur exécution.

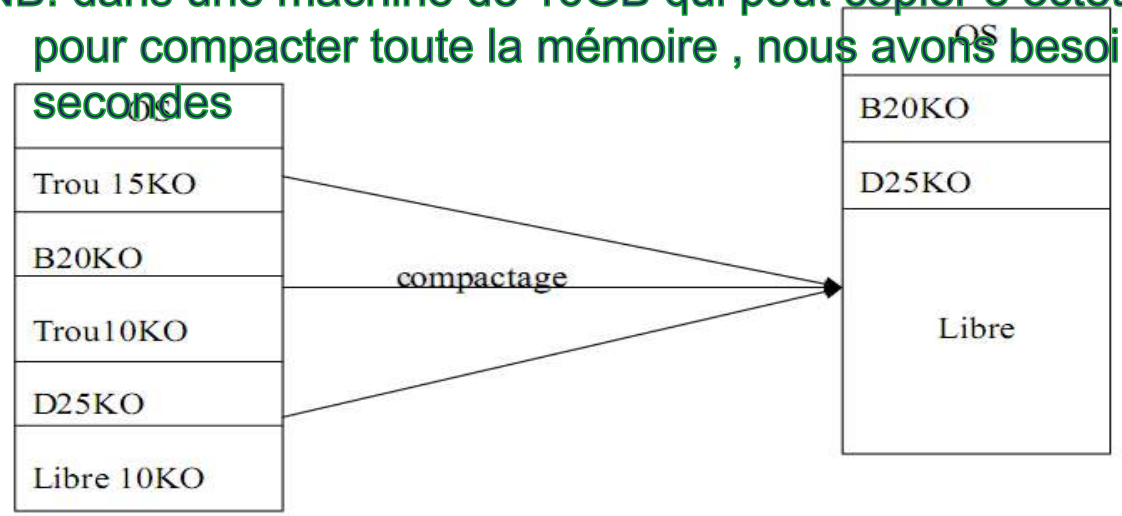
Au bout d'un certain temps de fonctionnement, la mémoire prend l'aspect d'une alternance de trous (zones libres) et de partitions (zones occupées) de taille quelconque →

Multiprogrammation

Compactage:

- Cette technique consiste à **rassembler** les trous (espaces libérés et libres) en une seule zone.
- En Déplaçant tous les processus vers le bas de la mémoire

NB: dans une machine de 16GB qui peut copier 8 octets en 8 ns → pour compacter toute la mémoire , nous avons besoin de 17.18 secondes



ramasse miettes (Garbadge collector)

Multiprogrammation avec partitions variables

- Avantages :
  - Une solution bien plus flexible
  - Une meilleur utilisation de la mémoire
- Désavantages :
  - Plus compliquée à implémenter
  - Il peut y avoir des “trous” laissés dans la mémoire, qui peuvent être compactés pour corriger le problème

Gestion de la mémoire

Pour gérer l'allocation et la libération de l'espace mémoire, le gestionnaire doit :

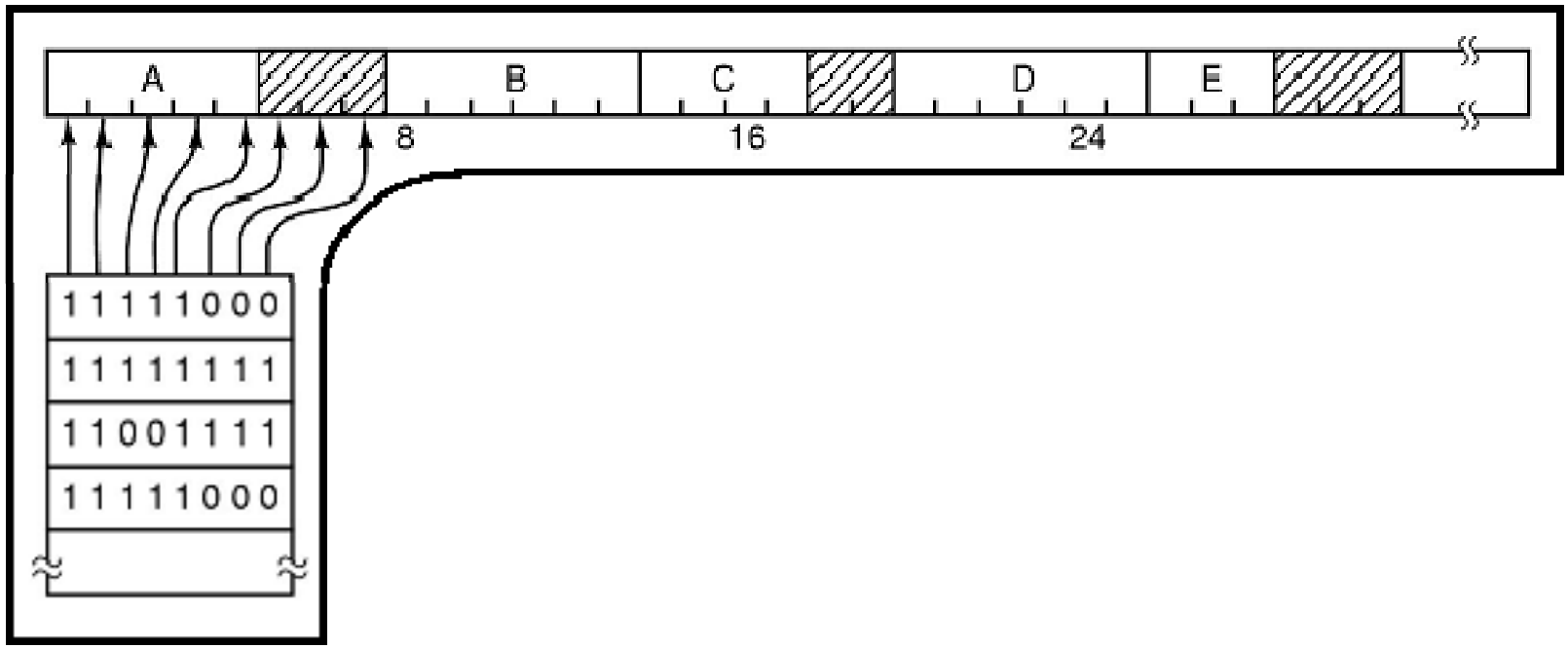
- Connaître l'état de la mémoire :
  - Tables de bits,
  - Listes chaînées,
- Avoir une politique de placement et de récupération d'espace.
  - First fit
  - Next fit
  - Best fit
  - Worst fit
  - Subdivision.

Gestion de la mémoire

- Gestion de la mémoire avec tableaux de bits:
  - Divise la mémoire en unités d'allocation dont la taille peut varier de quelques mots à plusieurs kilo-octets
  - Utilise un tableau de bits avec des 1 pour désigner les unités alloués et des 0 pour désigner les unités libres

Gestion de la mémoire: tables de bits

- Solution



Gestion de la mémoire: tables de bits

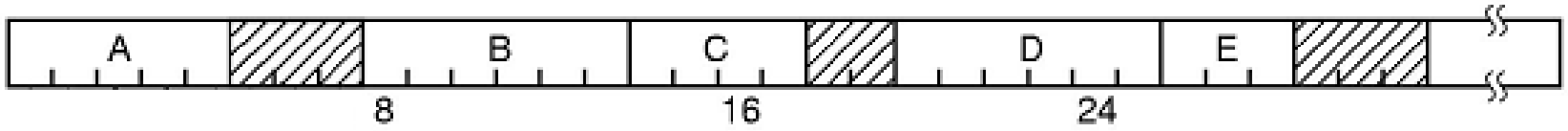
- Gestion de la mémoire avec tableaux de bits:
  - plus l'unité d'allocation est petite, plus le tableau de bits correspondant sera grand.
  - Avantages:
    - Facile à implémenter
    - Le tableau de bits est de grandeur fixe peu importe combien de programmes sont en mémoire
  - Désavantages:
    - Peut prendre du temps pour chercher dans le tableau une série de 0 consécutifs pour placer un programme
    - Les unités qui sont larges nous font perdre la fin de la dernière unité, ex: pour un unité de 64KO, si nous avons un programme qui a 65KO, alors nous perdons 63 kilooctets!

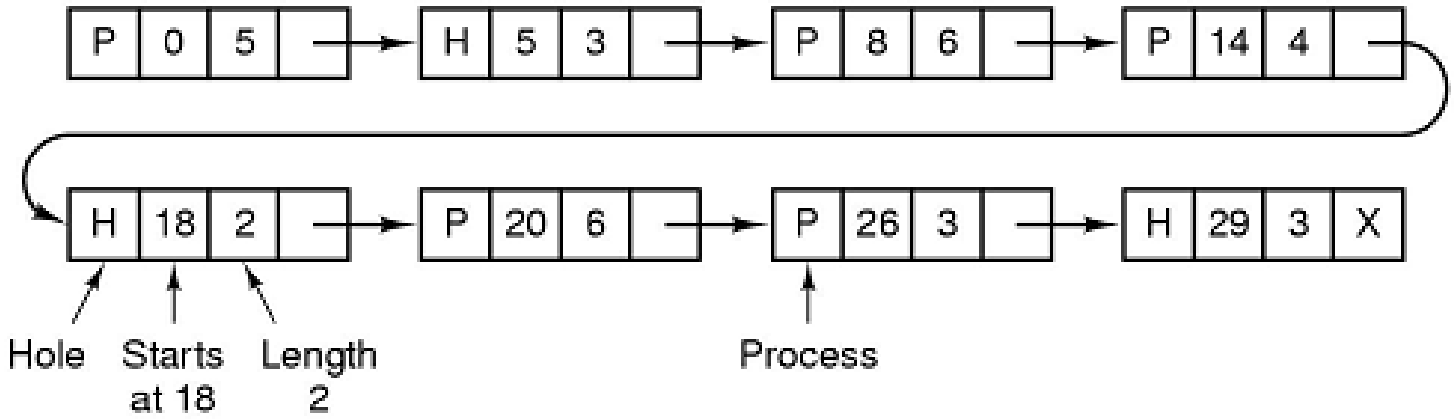
Gestion de la mémoire: listes chainées

- Gestion de la mémoire avec des listes chaînées:
  - Utilise une liste chaînée pour tracer les “segments”
    - Segments sont ou bien un processus ou un trou entre les processus.
  - moins de recherche à faire ce qui accélère l’allocation mais la libération des segments est lente.

Gestion de la mémoire: listes chaînées

- Solution





Algorithmes d'allocation

- Comment est-ce que les nouveaux processus sont placés en mémoire?

- **First fit (premier qui correspond)** :Trouve le premier segment vide qui est assez large pour tenir le processus et le segment est divisé en un segment de processus et un segment de trou.

- **Next fit (Prochain segment)** : Une amélioration simple sur le first fit. Elle dénote la place où le dernier programme a été inséré et commence à chercher depuis cette place.
- **Worst fit (Plus grand résidu) :** On prend le plus grand trou disponible, afin de laisser un trou maximum pour le autres processus.
- **Best fit (qui correspond le mieux)** : Parcourt toute la liste et recherche le plus petit trou pouvant contenir le processus.

Algorithmes d'allocation

Exercice :

La mémoire d'un système va et vient contient des zones libres (ordonnées en fonction des adresses mémoires) de 10Ko, 4Ko, 20Ko, 18Ko ,7Ko ,9Ko ,12Ko et 15Ko.

Quelle zone l'algorithme de la première zone libre (**first fit**) sélectionne-t-il pour des demandes d'allocation de segments de 12Ko, 10Ko, 8Ko ?

Même question pour les algorithmes du meilleur ajustement (**best fit**) et de la zone libre suivante (**next fit**) et du **worst fit**

Subdivision

- Cet algorithme proposé par Donald KNUTH en 1973 utilise l'existence d'adresses binaires pour accélérer la fusion des zones libres adjacentes lors de la libération d'unités.
- Le gestionnaire mémorise une liste de blocs libres dont la taille est une puissance de 2 (1, 2, 4, 8 octets, ..., jusqu'à la taille maximale de la mémoire).

Subdivision

Exercice:

Avec une mémoire de 1 Mo,

Un processus A demande 70 Ko,

Un processus B demande 35 Ko ,

Un processus C demande 200 Ko,

A s'achève,

Un processus D demande 60 Ko,

B s'achève,

D s'achève,

C s’ achève.

Subdivision

État initial	1024				
Alloue A=70	A	128		256	512
Alloue B=35	A	B	64	256	512
Alloue C=200	A	B	64	C	512
Libère A	128	B	64	C	512
Alloue D=60	128	B	D	C	512
Libère B	128	64	D	C	512
Libère D	256			C	512
Libère C	1024				

LA MÉMOIRE VIRTUELLE

Pourquoi une mémoire virtuelle ?

- Mémoire **physique** coûteuse.
- Mémoire **secondaire** (disques, mémoire étendue, ...) moins coûteuse.
- Programmes gourmands en mémoire et qui ne "tiennent pas" toujours en RAM.

➔ Utiliser la mémoire secondaire "comme" mémoire RAM.

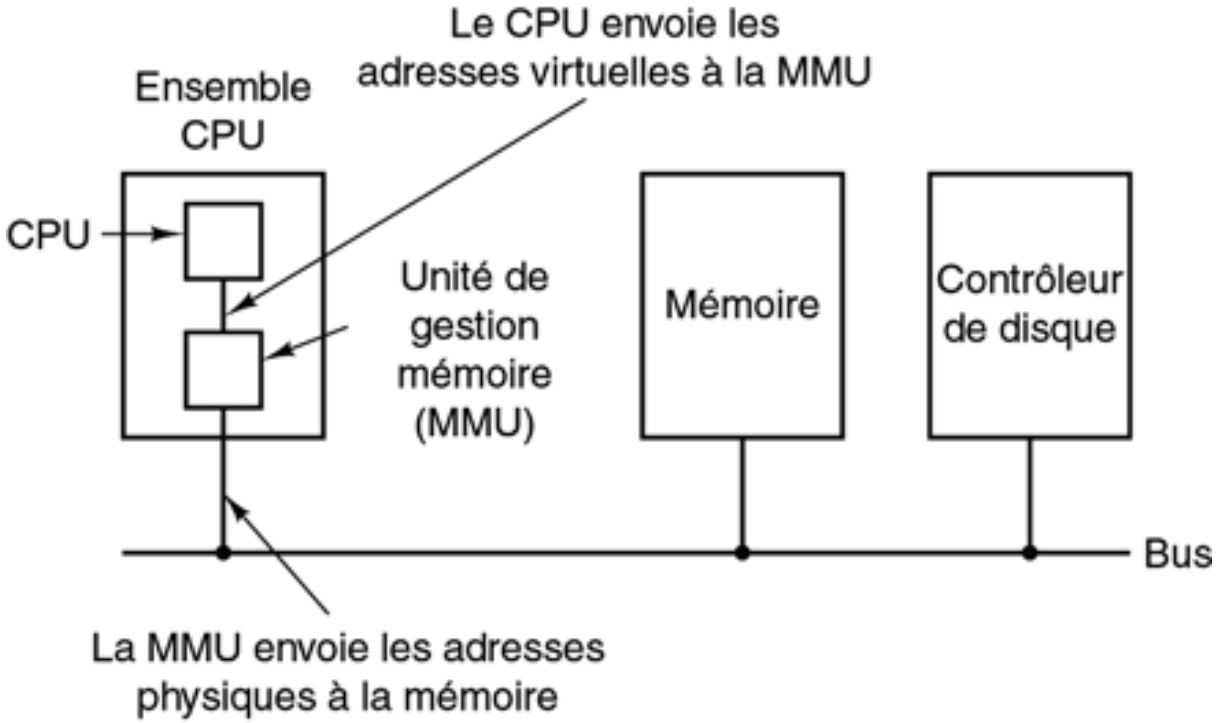
Principe de la MV

- L a mémoire virtuelle est une technique qui permet d'exécuter des programmes dont la taille excède la taille de la mémoire physique.
- L'espace d'adressage d'un processus, généré par les compilateurs, constitue la mémoire virtuelle du processus ou espace d'adressage virtuel
- Avec des techniques de pagination et de segmentation, sa taille peut être très supérieure à celle de la mémoire physique.

la traduction d'adresses

- En général on compile tous les programmes avec des adresses logiques.
- Les adresses physiques sont calculées durant l'exécution du programme.
- La conversion des adresses est réalisée par une pièce de matériel que l'on appelle Unité de gestion de la mémoire (Memory Management Unit) (MMU).
  - Le MMU convertit les adresses logiques du processus en adresses physiques dans la RAM

la traduction d'adresses



Localisation et fonction du MMU

LA MÉMOIRE VIRTUELLE

- Pagination
- segmentation

Principe de pagination

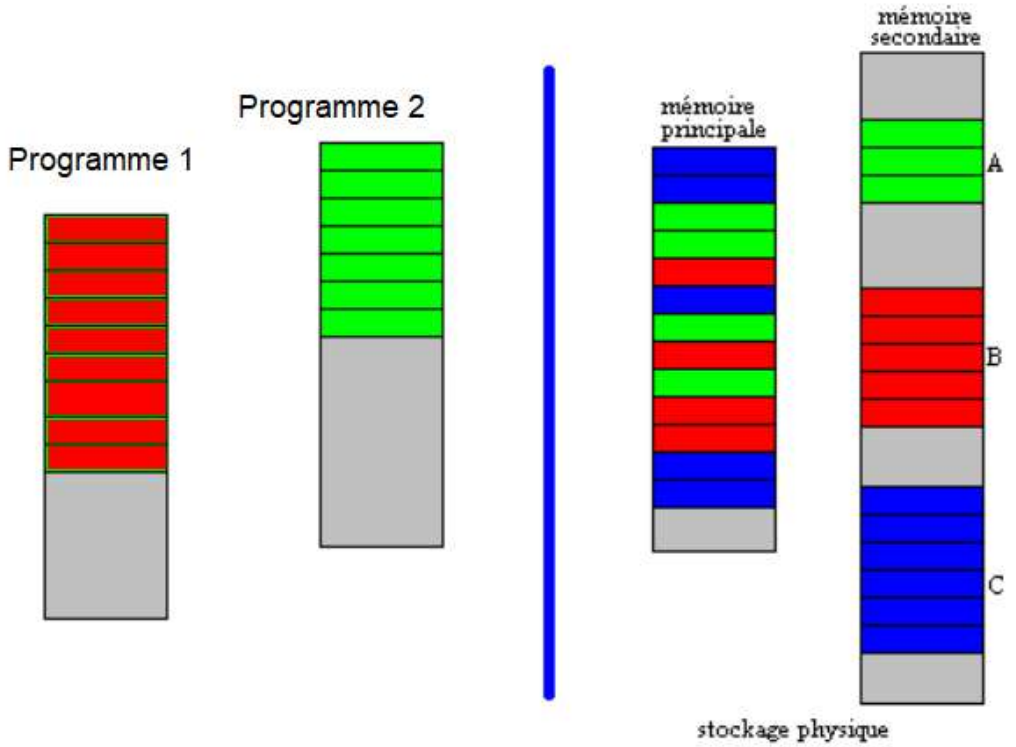
- Chaque programme a son propre espace d'adressage, qui est divisé en morceaux appelés pages.
  - Chaque page est une plage d'adresses contiguës.
- Ces pages sont mappées (traduites) sur la mémoire physique,
- Il n'est pas obligatoire d'avoir toutes les pages en mémoire physique pour exécuter le programme.
  - Quand la page demandée se trouve en RAM, le MMU effectue le mapping nécessaire à la volée.
  - Quand elle ne se trouve pas dans la mémoire physique, le SE prend la main pour aller chercher sur disque ce qui manque, le range en mémoire et reprend la où on s'est arrêté

Principe de pagination

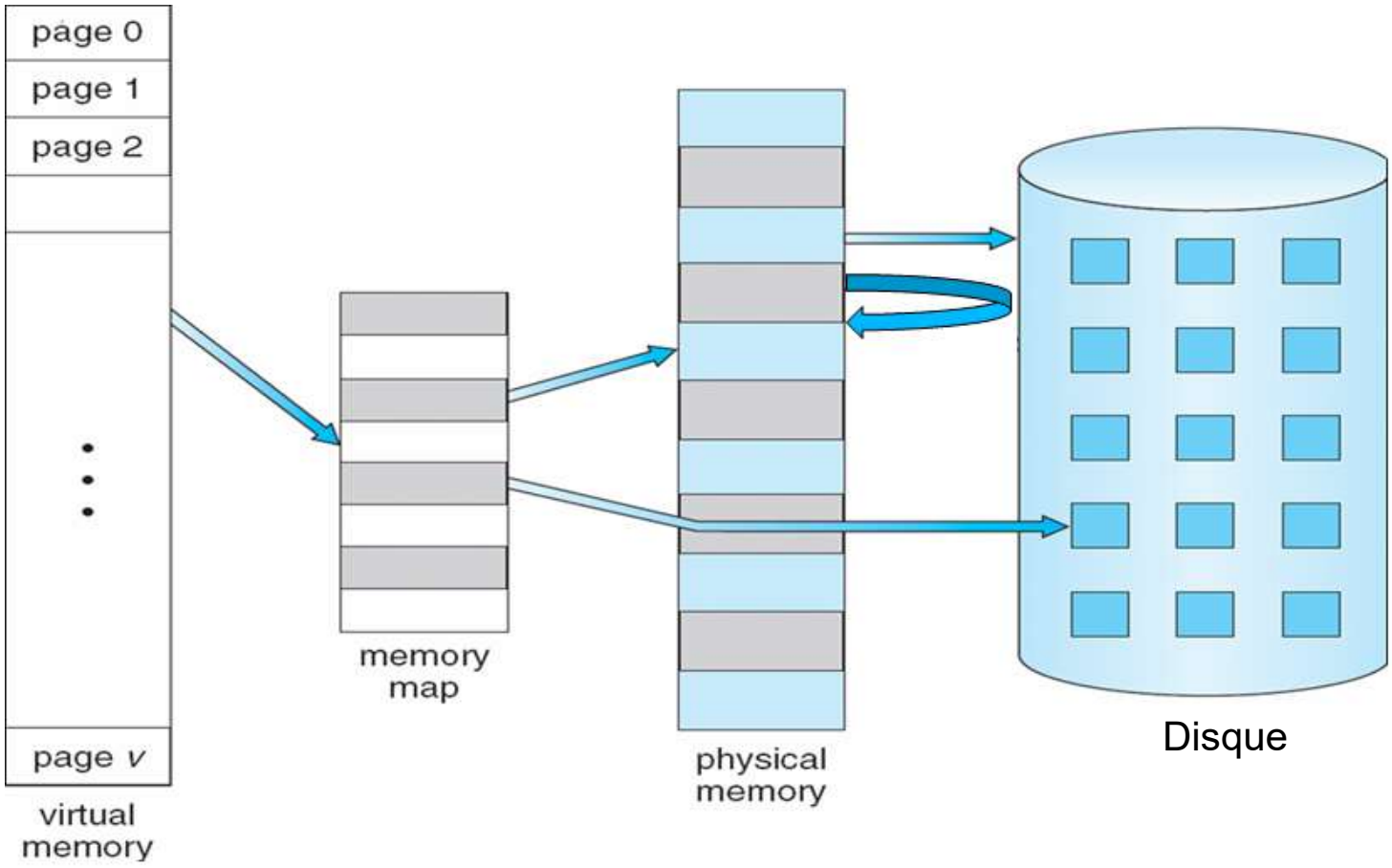
- L'espace d'adressage virtuel est divisé en pages dont la taille est une puissance de 2 (comprise en 512 octets et 64 Ko)
- Les unités correspondantes en mémoire physique sont des cadres de pages (page frames)
- Les pages et les cadres de page sont généralement de même taille.
- Le MMU dispose d'une table de traduction pour convertir les adresses virtuelles en adresses physiques
  - La table de pages donne pour chaque page virtuelle le numéro de cadre où elle est placée (mapping)

pagination

A un instant donné, la situation est la suivante, pour plusieurs programmes en exécution :



Pagination


page 0
page 1
page 2
page v
virtual memory
memory map
physical memory
Disque

Exemple :
Un PC peut générer des adresses de 16 bits de 0 à 64K-1.
C'est l'adressage virtuel.
Mais ce PC n'a que 32Ko de mémoire physique.

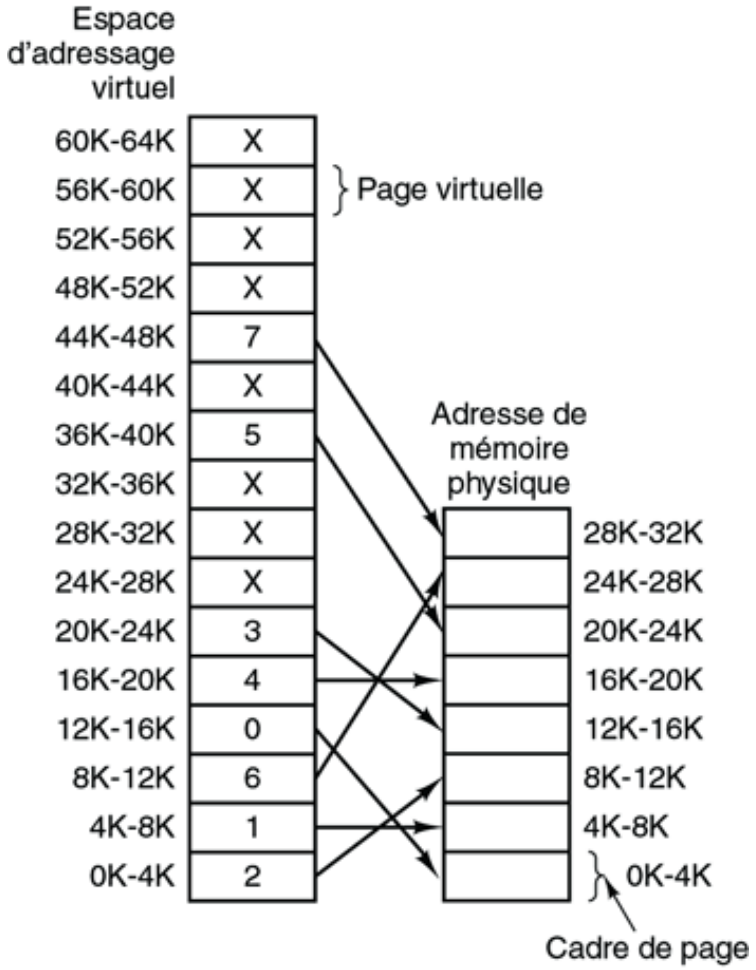
Conséquence : il est possible d'écrire un programme de 64Ko mais pas de le charger entièrement en mémoire.

Pagination

Exemple :

- *Dans l'exemple, un ordinateur peut produire des adresses sur 16bits (64 Ko) mais il n'y a que 32 Ko de mémoire physique.*

- *La mémoire virtuelle est divisée en pages de 4Ko*
- *La mémoire physique est divisée en cadres de 4Ko*



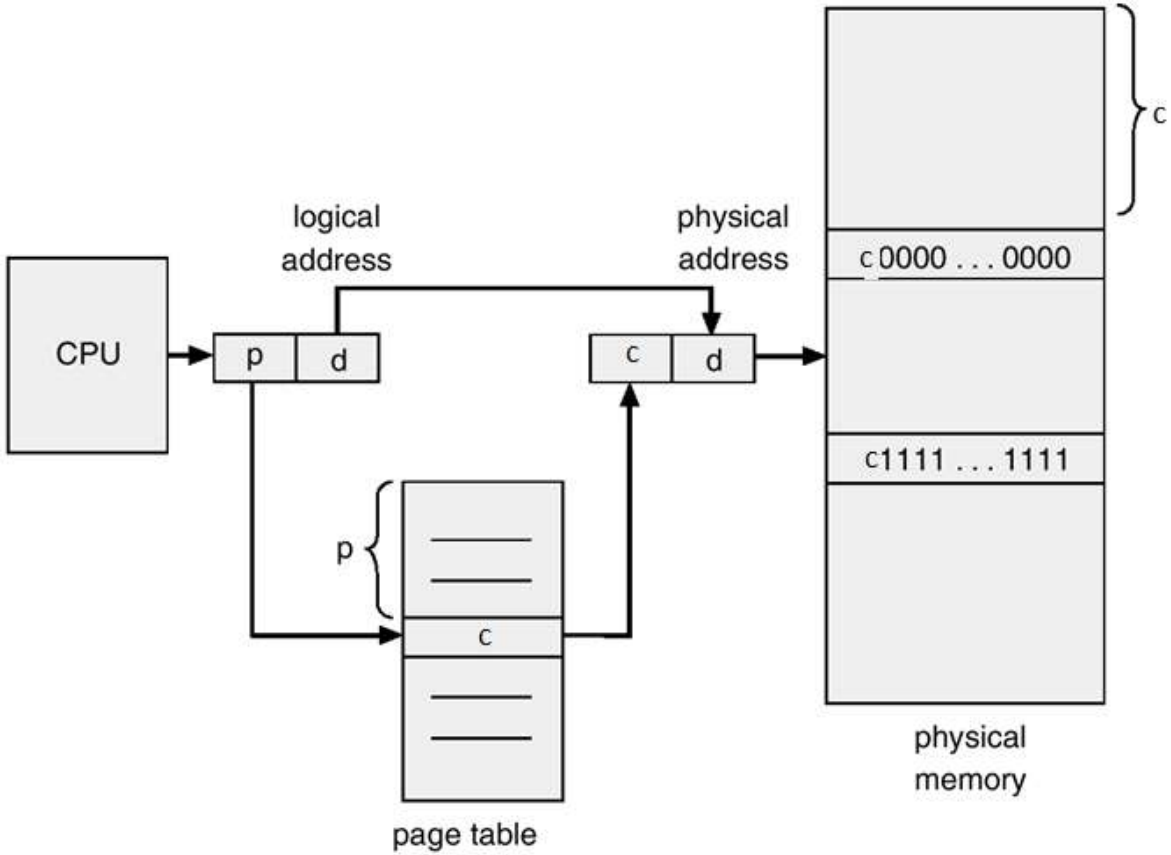
Traduction d'adresse

Translation d'adresses

- Adresse générée par la CPU (@ virtuelle ou logique) <p,d>
  - *Numéro de Page (p)*
  - *Déplacement dans la Page (d)*

- Adresse physique <c,d>
  - *Numéro de cadre (c)*
  - *Déplacement dans le cadre (d)*

- La table de page est une fonction de traduction <p,d>→<c,d>
  - La table de page est maintenue par le MMU


CPU
logical address
p
d
physical address
c
d
c 0000 . . . 0000
c1111 . . . 1111
c
p
c
page table
physical memory

Pagination

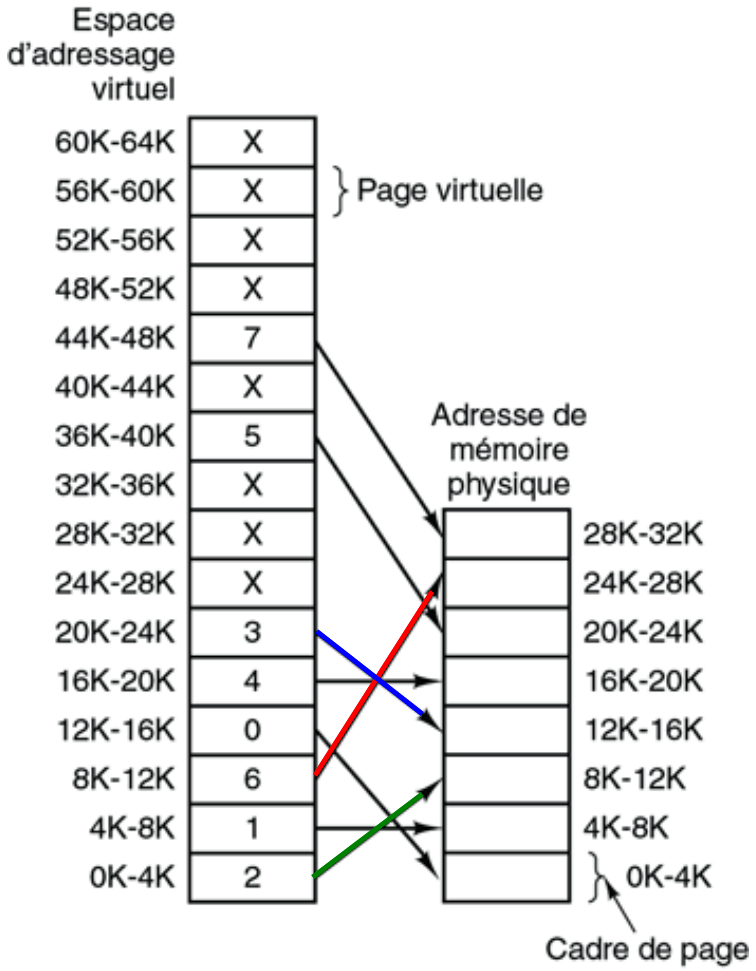
Exemple :

pour chacune des adresses virtuelles décimales suivantes cherchez les adresses physiques.
0,100,8192,20500,62000

Le programme va exécuter l'instruction suivante :

MOV REG, 0 --> Copier le contenu de l'adresse virtuelle 0 dans le registre REG.

Cette @virtuelle sera envoyée au MMU pour la traduire en @physique

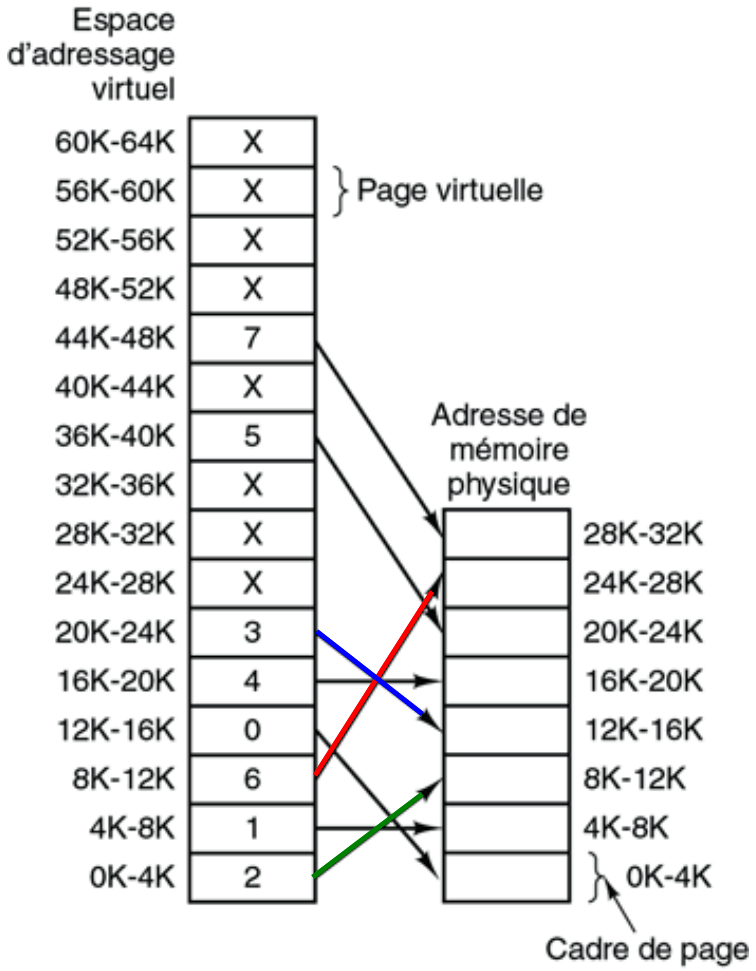


Pagination

Exemple :

pour chacune des adresses virtuelles décimales suivantes cherchez les adresses physiques.
0,100,8192,20500,62000

- *L'adresse virtuelle 0 est transformée en adresse physique 8192=2*4096*
- L'adresse virtuelle 8192=2*4096 est transformée en adresse physique 24576=6*4096
- L'adresse virtuelle 20500=5*4096+20 est transformée en adresse physique 12308=3*4096+20



Défaut de page

- Une adresse virtuelle peut référencer :
  - *Soit une page en mémoire physique (page hit)*
  - *Soit une page sur le disque dur (page miss)*
- Si la page cherchée se trouve sur le DD , le MMU engendre une interruption (page fault) et une procédure se charge de transférer les données en mémoire physique.
  - Le SE sélectionne un cadre peu utilisé et écrit son contenu sur le DD,
  - Il transfère ensuite la page qui vient d'être référencée dans le cadre qui vient d'être libéré.
  - Mets à jour le mapping et recommence l'instruction

LA MÉMOIRE VIRTUELLE

- Pagination
- Segmentation

Segmentation

- Gestion de la mémoire qui supporte une vue utilisateur de la mémoire
- Un programme est une collection de segments. Un segment est une unité logique telle que:

  programme principal,

  procédure,

  fonction,

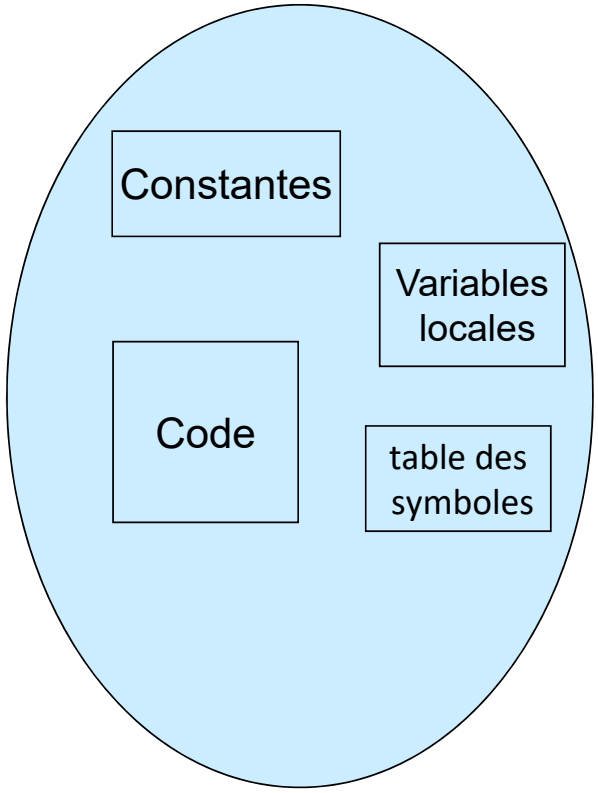
  méthode,

  objet,

  variables locales, variables globales,

  bloc commun ….

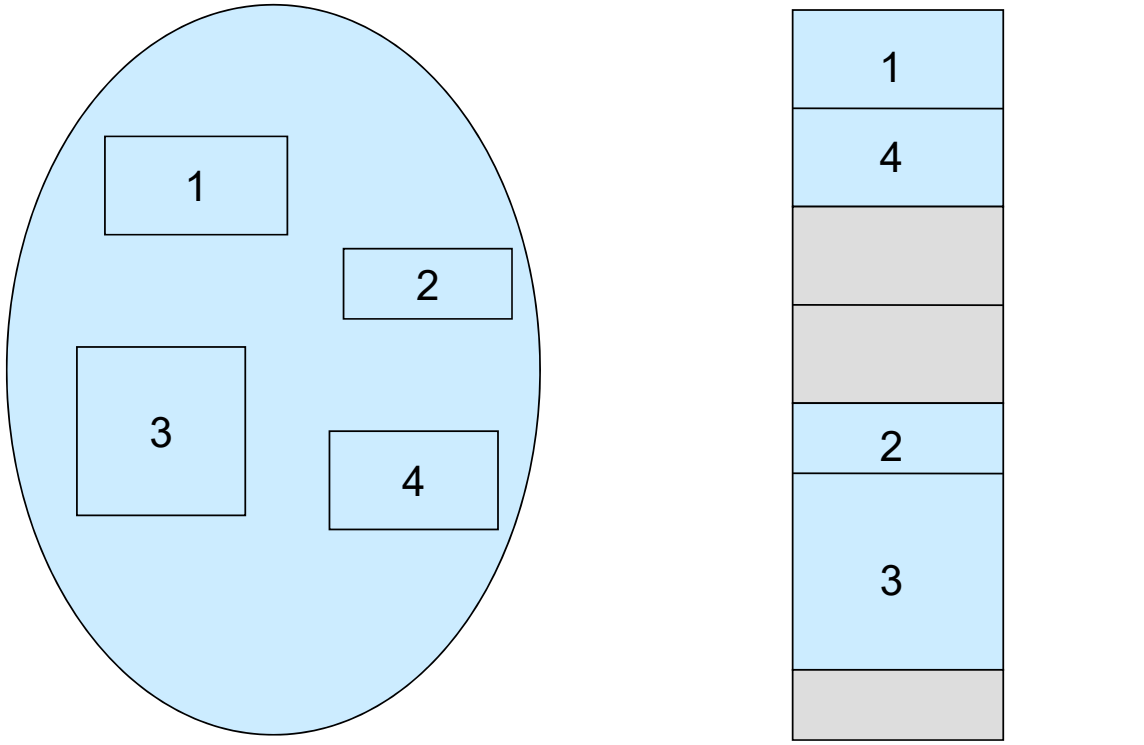
Vue Utilisateur d'un Programme



Segmentation

- La segmentation est une approche assez semblable à la pagination. La différence principale est que les segments sont des blocs de mémoire de taille variable.
- La segmentation offre plusieurs espaces d'adresses indépendants appelés segments.
  - Chaque segment est une suite d'adresses continues de 0 à une adresse maximale autorisée.
- Les segments ont des tailles différentes qui varient en cours d'exécution, ils représentent des espaces d'adressage séparés.
- La segmentation simplifie le partage des procédures et des données entre plusieurs procédures.

Vue Logique de la Segmentation



Espace Utilisateur

Espace Mémoire Physique

Traduction d'adresses

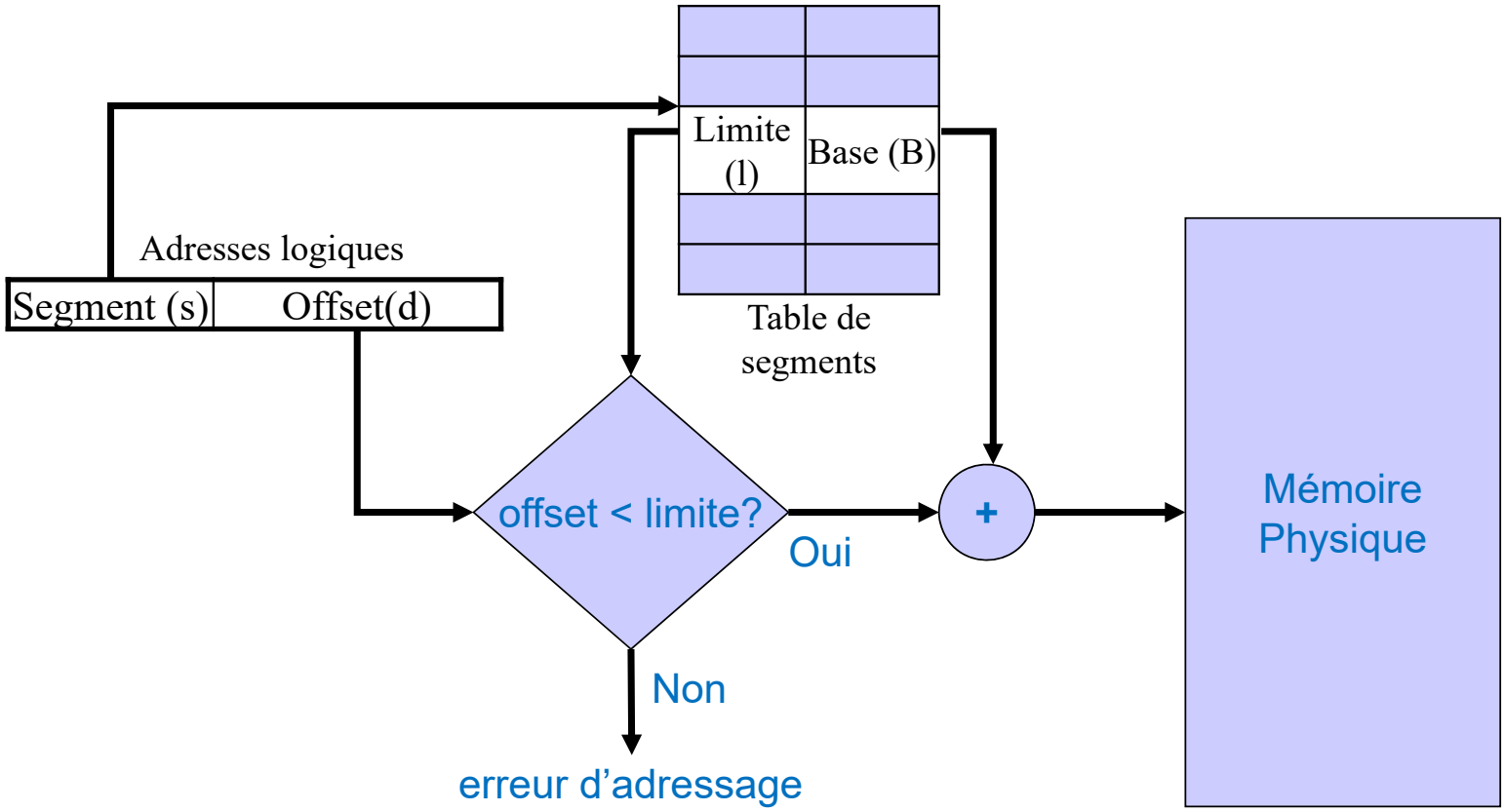
- Adresse logique contient le couple:<s,d>

  <numéro-segment, déplacement>,
- Adresse physique contient le couple: <B,l>
  - Base – contient l'adresse physique de début du segment en mémoire
  - *limite* – spécifie la longueur du segment

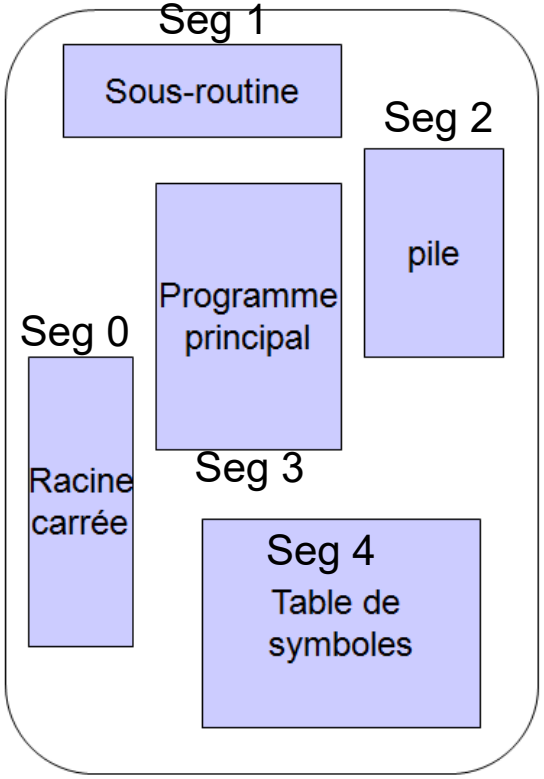
- **Table des Segments** – traduction entre adresse logique et adresse physique <s,d>→<B,l>;
  - La table de segments est maintenue par le MMU

Traduction d'adresses

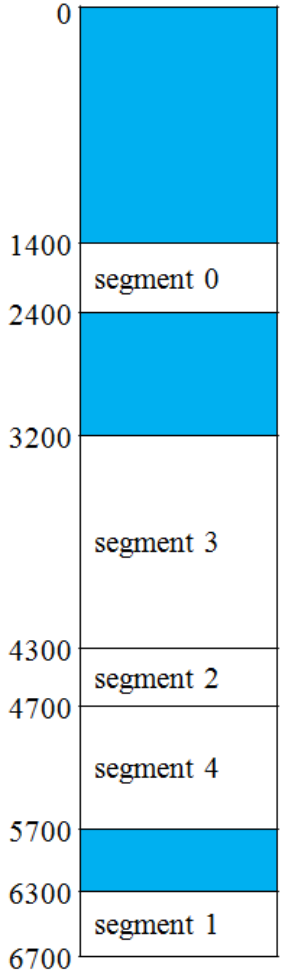
- Comment est-ce que les adresses sont traduites?



Exemple de Segmentation

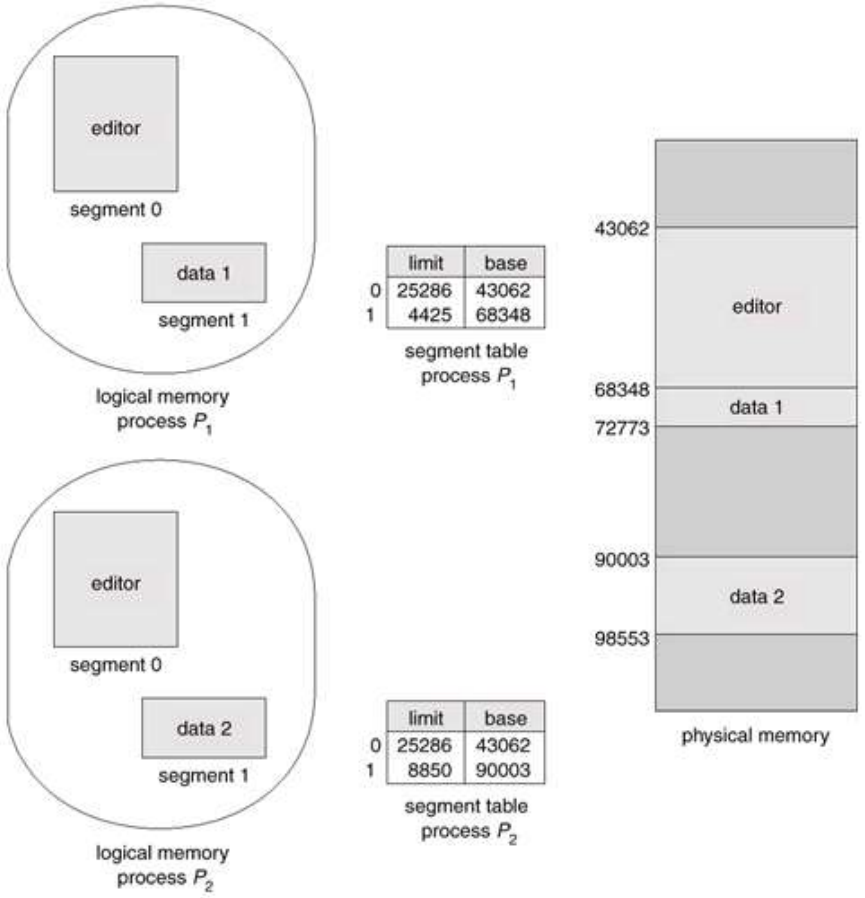


	limite	base
0	1000	1400
1	400	6300
2	400	4300
3	1100	3200
4	1000	4700



Partage de Segments

- deux processus P1 et P2 partagent le même segment de code (programme) mais ont des segments pour les données disjoints et de tailles différentes.



Protection des segments:

- Chaque entrée dans la table des segments peut contenir des infos de protection:
  - longueur du segment
  - privilèges de l`usager sur le segment: lecture, écriture, exécution
    - Si au moment du calcul de l'adresse on trouve que l'usager n'a pas droit d'accès → interruption
    - ces infos peuvent donc varier d'un usager à autre, par rapport au même segment!

Base	Limite	read, write, execute?

Comparaison

Considérations	Pagination	Segmentation
Le programmeur doit-il connaître la technique utilisée ?		
Combien y a-t-il d'espaces d'adressage linéaire ?		
L'espace d'adressage total peut-il dépasser la taille de la mémoire physique ?		
Les procédures et les données peuvent-elles être distinguées et protégées séparément ?		
Le partage de procédure entre utilisateurs est-il simplifié ?		
Pourquoi cette technique a-t-elle été inventée ?		