Centre Universitaire de Mila

2 ème année licence LMD Informatique

Module : Systèmes d'exploitation 1

Bessouf Hakim

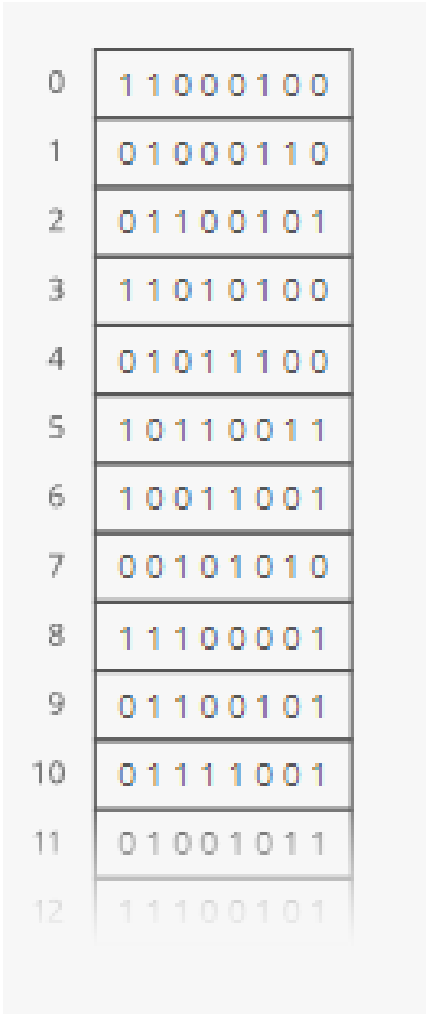
CHAPITRE 4

Gestion de la mémoire centrale

1. Objectifs d'un gestionnaire de la mémoire
2. Fonctions
3. Modes de partage de la mémoire
4. Protection de la mémoire
5. Partage de code

Introduction

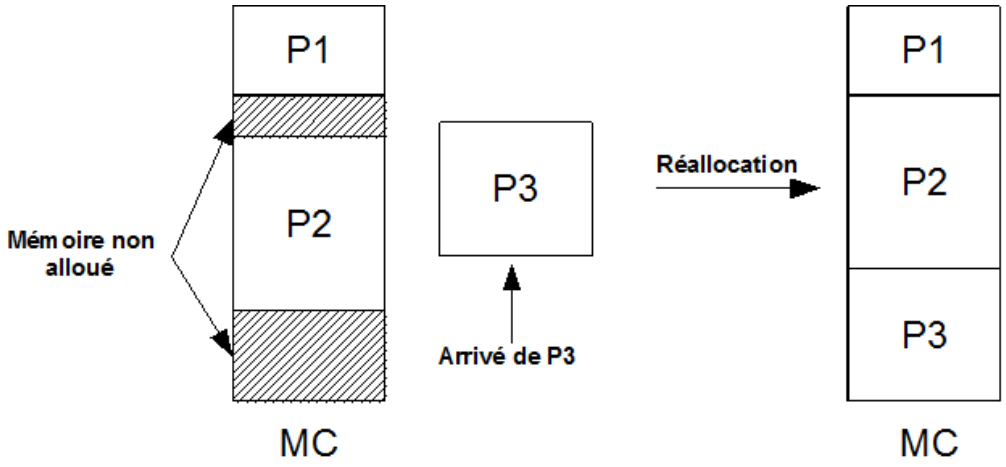
- La mémoire centrale est une partie essentielle de l'ordinateur, elle est utilisé pour charger les programmes (donnés + codes) exécutés par le processeur.
- La mémoire centrale est découpé en plusieurs cases appelés **mots mémoire,** chaque mot mémoire a une adresse unique. Cette adresse est utilisé par le processeur pour lire et écrire les informations.



Le gestionnaire de mémoire

C’est un module du système d'exploitation qui s'occupe de la gestion de la mémoire centrale, ces objectifs sont :

- **Le partage de la mémoire** : Le gestionnaire doit partager la MC entre plusieurs processus,
- **La protection de la mémoire** : Le gestionnaire doit pouvoir empêcher un processus d'accéder à l'espace mémoire d'un autre processus pour éviter les conflits.
- **La réallocation** : Le gestionnaire doit pouvoir réarranger (réallouer) les processus en mémoire centrale pour trouver l'espace nécessaire et charger d'autres processus.



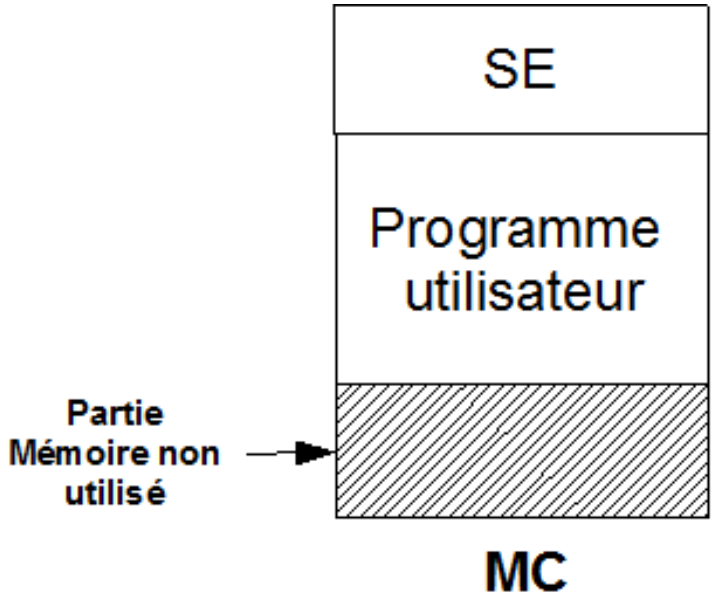
Le gestionnaire de mémoire

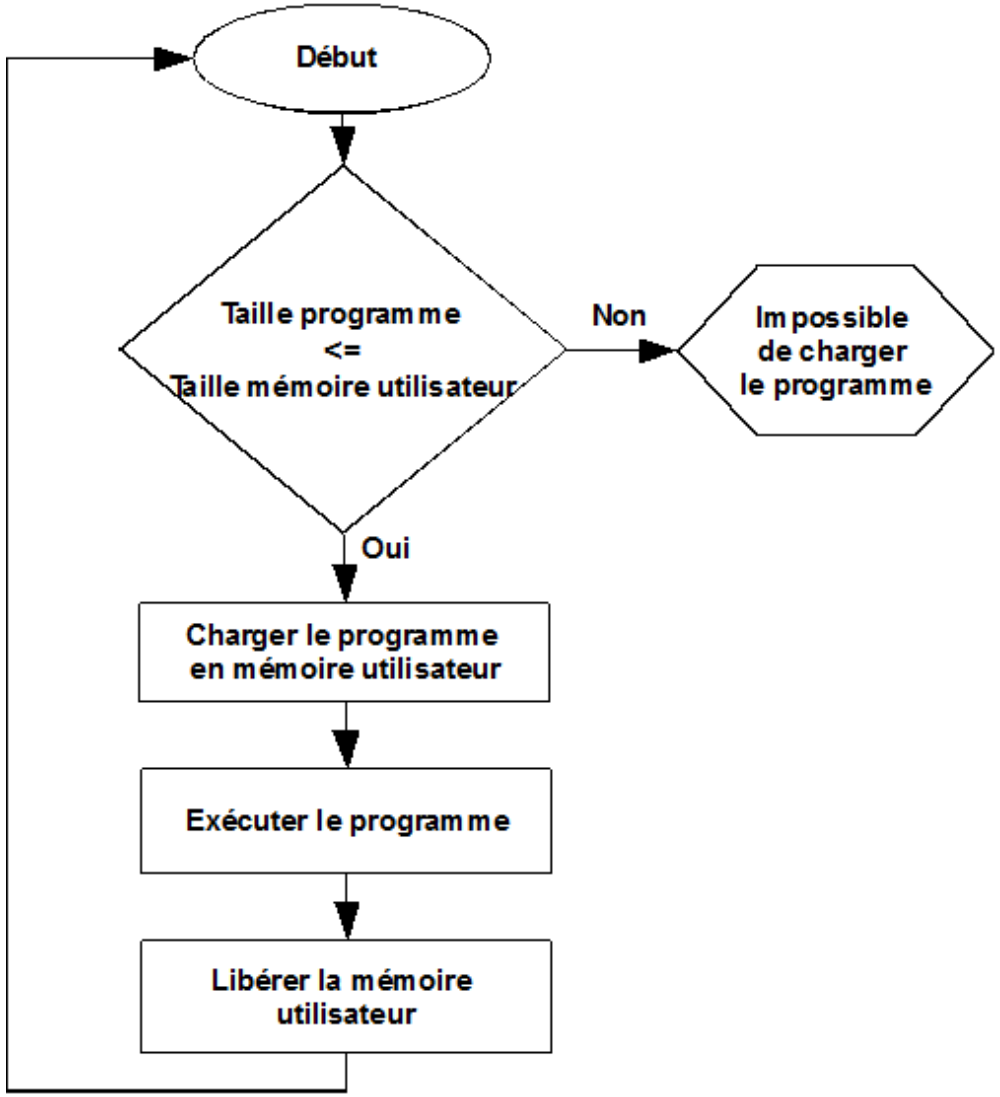
Les fonctions du gestionnaire de mémoire: Un gestionnaire de mémoire doit assurer les fonctions suivantes :

- **Allouer l'espace mémoire aux processus** : Le gestionnaire de mémoire doit déterminer une technique d'allocation de la mémoire.
- **Libérer l'espace mémoire** : Le gestionnaire de mémoire doit déterminer une technique pour libérer l'espace mémoire quand un processus termine sont exécution.
- **Déterminer les parties libres** : Le gestionnaire de mémoire doit garder une trace de chaque partie de la mémoire centrale pour pouvoir la gérer correctement.

Modes de partage de la mémoire (stratégies d'allocation de la mémoire)

- **Une seule zone contiguë (single continuous allocation)**
  Dans cette stratégie un seul programme est exécuté a la fois (**monoprogrammation**). La mémoire centrale est divisé en deux partie, la première partie contient le système d'exploitation et la deuxième partie contient le programme utilisateur.





Organigramme de l'allocation de la mémoire avec une seule zone contiguë

RQ: Cette stratégie d'allocation est inefficace car une partie de la mémoire peu être non utilisé.

Modes de partage de la mémoire (stratégies d'allocation de la mémoire)

- **Partitions multiples**
  Dans cette stratégie la MC est divisé en plusieurs partitions, chaque partition peut contenir un processus. On peut donc avoir plusieurs processus en mémoire centrale en même temps (**multiprogrammation**).

- Partitions multiples statiques

- Partitions multiples dynamiques

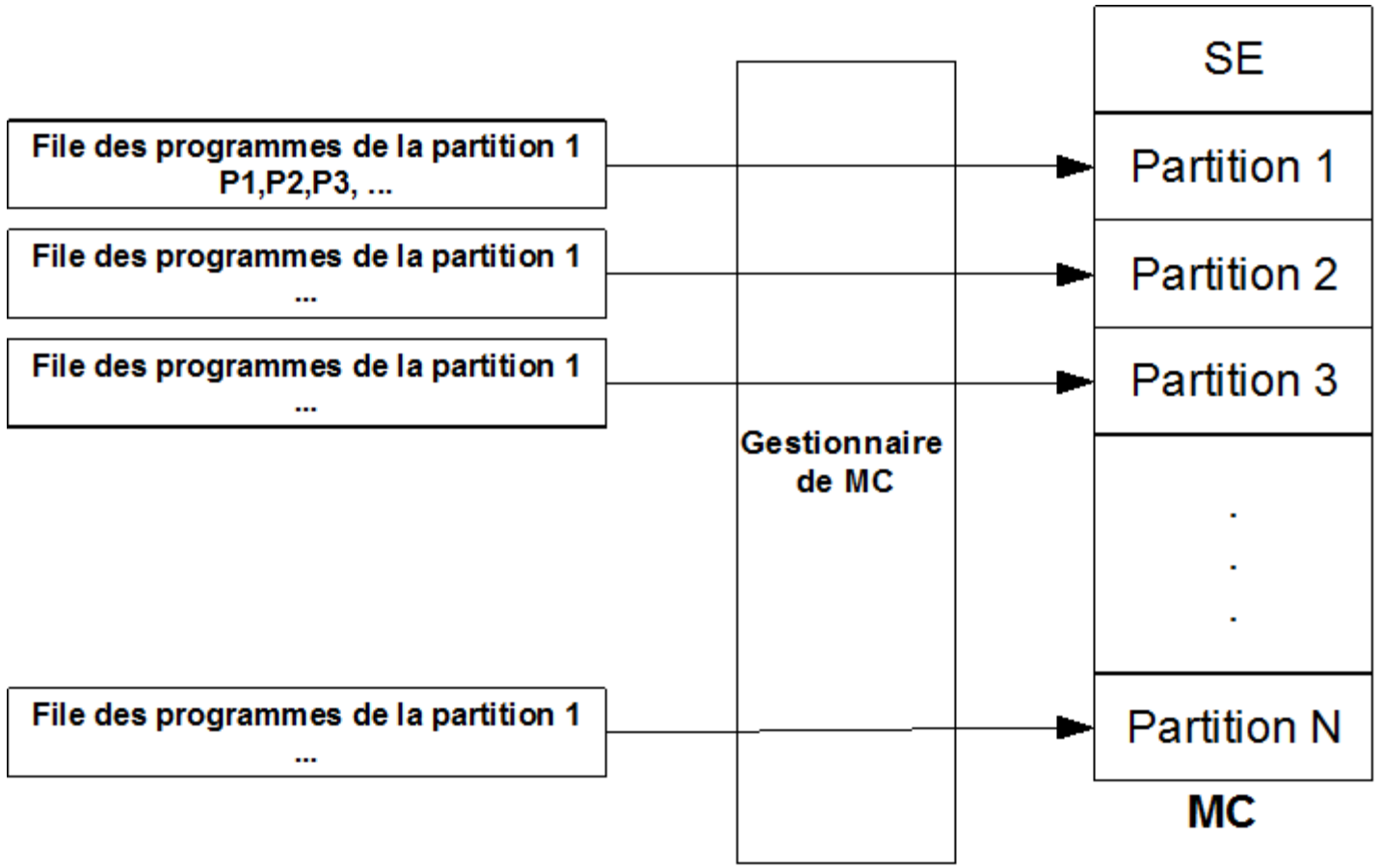
1. Partitions multiples statiques

1. Partitions multiples statiques :

- Dans ce schéma la taille et le nombre de partitions est fixé a l'avance lors du démarrage du système d'exploitation. Pour exécuter un programme il y a deux cas passibles :

1.1 Le programme à exécuter est en code absolus :
Dans ce cas les adresses du programme sont des adresses physiques et le programme est donc lié à une partition précise et ne peut pas s'exécuter dans une autre partition.

- Pour pouvoir exécuter les programmes le gestionnaire de mémoire centrale associe à chaque partition une file des processus en attente d'exécution comme décrit dans la figure suivante :



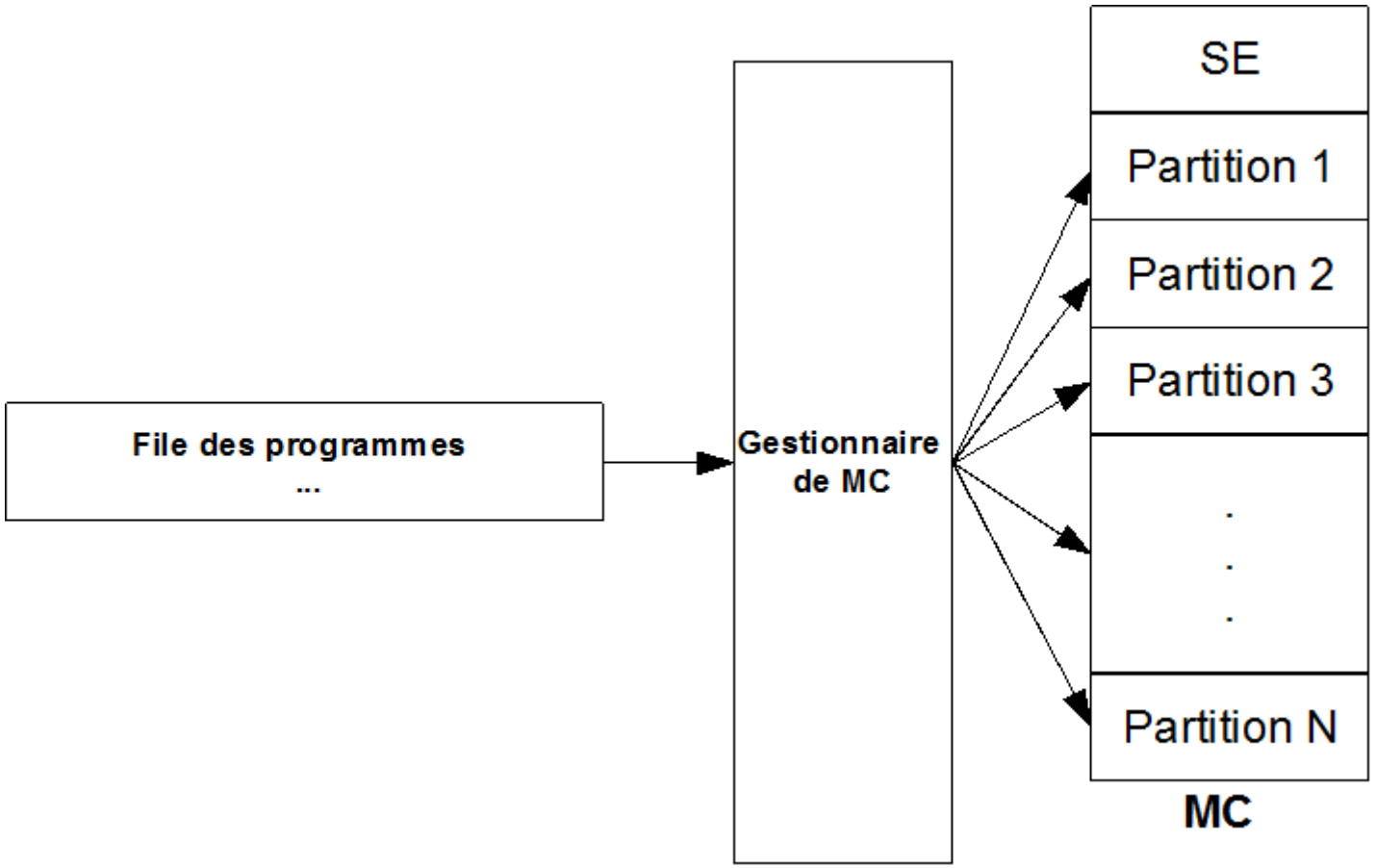
RQ: L'inconvénient du chargement absolus est que des partitions peuvent être libres alors que les programmes sont dans une file d'une autre partition.

Une files de programmes associé a chaque partition statiques.

1. Partitions multiples statiques

1.2 Le programme à exécuter est en code relogeable:
Dans ce cas les adresses du programme sont des adresses logiques (non absolus) et le programme peut être chargé et exécuté dans n'importe quelle partition qui a une taille suffisante.
Le gestionnaire de mémoire associe a tous les processus une seule file d'attente.

- Pour exécuter un programme le gestionnaire de MC doit trouver une partition de taille suffisante qui est libre et charger le programme dans cette partition
- le gestionnaire doit aussi convertir les adresses logique en adresses physique suivant la partition utilisé.
- Pour convertir les adresses logiques en adresses physique on ajoute l'adresse de début de la partition a toutes les adresses logiques du programme.

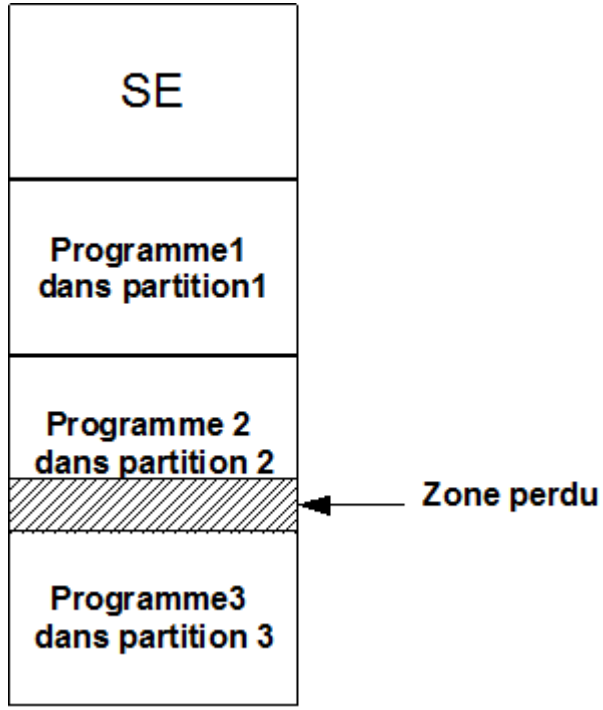


Une seule file de programmes est associé a toutes les partitions statiques.

1. Partitions multiples statiques

- **Problème de fragmentation**
  La fragmentation interne :

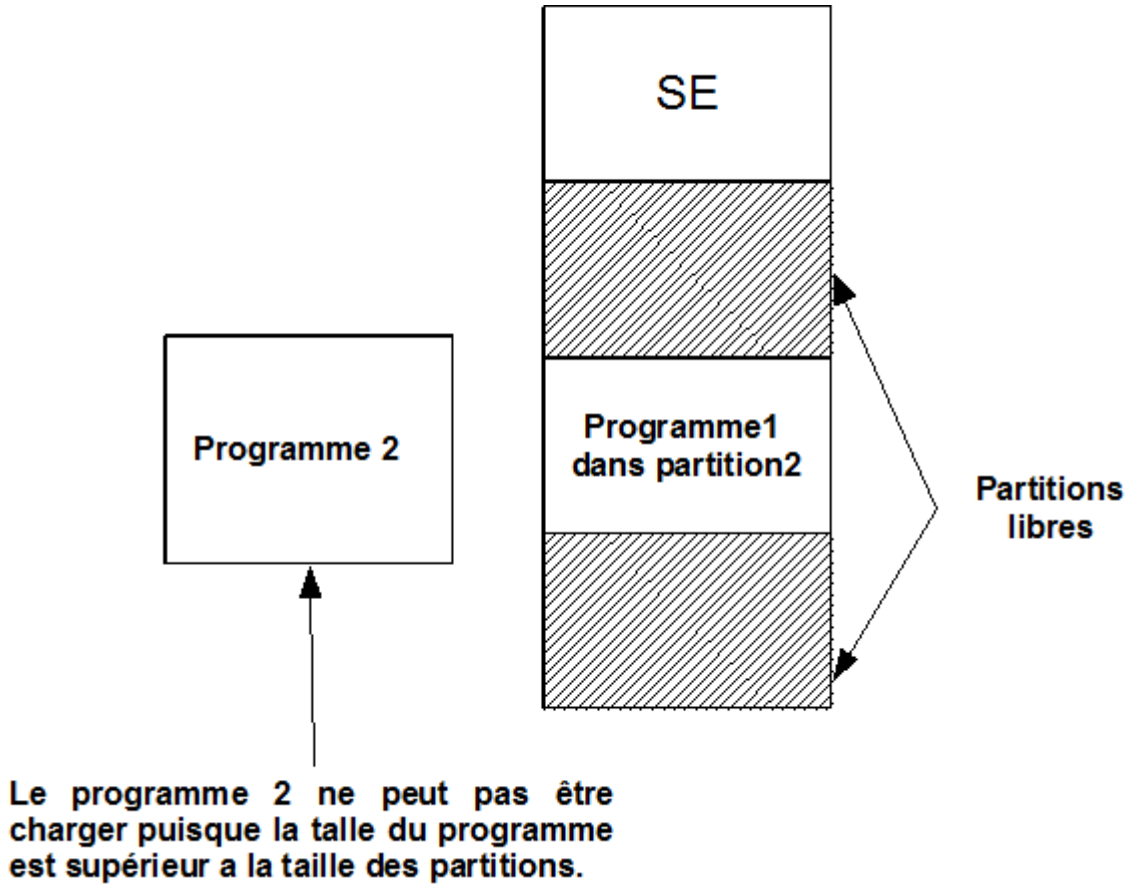
quand un programme est chargé dans une partition et la taille du programme est inférieure a la taille de la partition on aura de la mémoire non utilisé, on dit qu'il y a une fragmentation interne.



1. Partitions multiples statiques

- **Problème de fragmentation**
  La fragmentation externe :

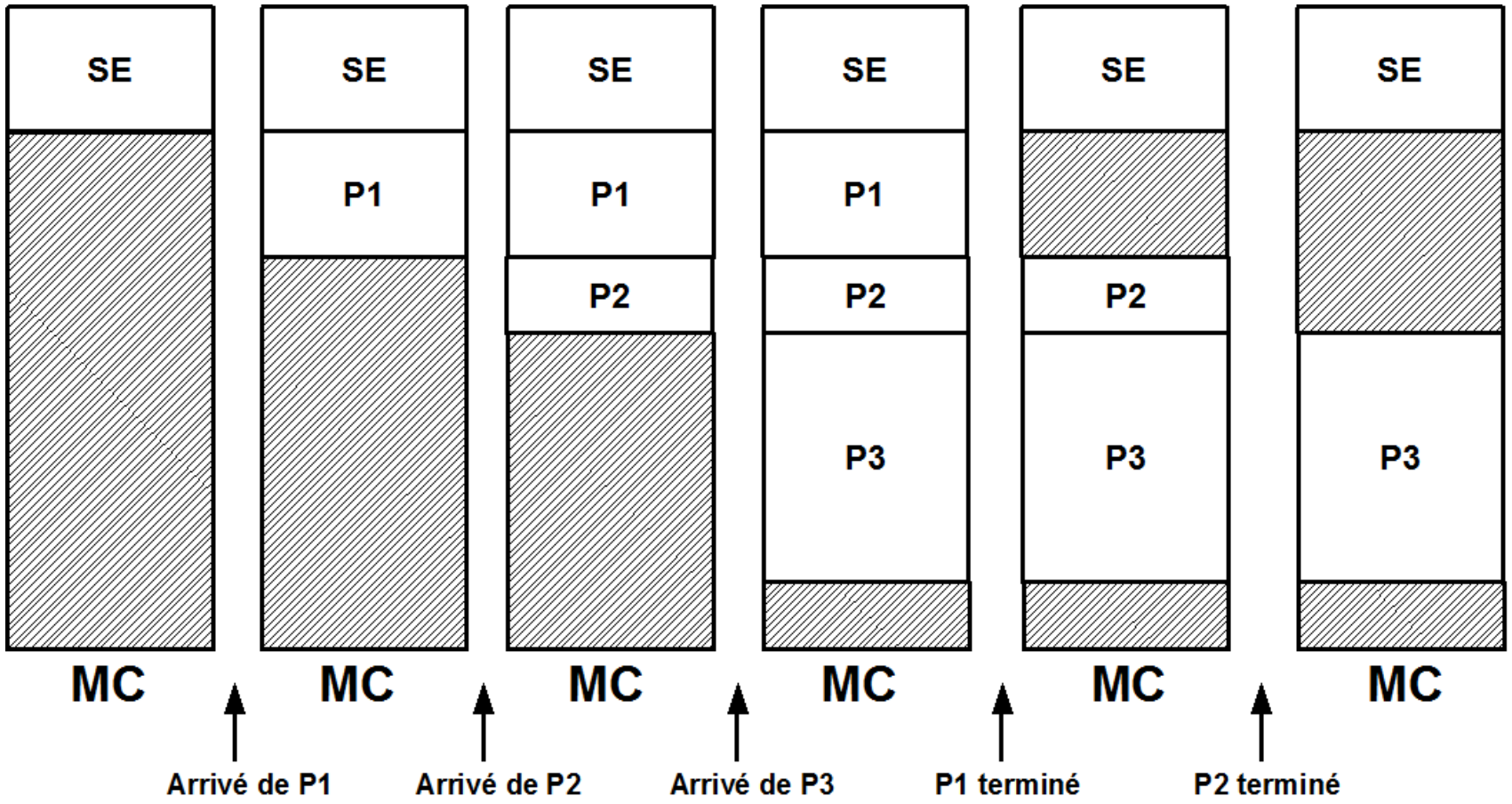
Si la taille d'un programme est supérieur a la taille de toutes les partitions libres et inferieur à la somme des partitions libres, donc le programme ne peut pas être chargé. Dans ce cas on dit qu‘on a une fragmentation externe.



2. Partitions multiples dynamiques

- Le gaspillage de la mémoire centrale dû a la fragmentation dans le partitionnement multiple statique a conduit au schéma de partitionnement multiple dynamique.
- Dans ce schéma suivant on partitionne la mémoire centrale dynamiquement selon la demande, et on alloue aux programmes des partitions exactement égale a leurs tailles. Lorsque un programme termine son exécution sa partition est récupéré pour être alloué à un autre programme. Si une partition libre est adjacente à une autre partition libre, les deux partitions sont **compactés** en une seule partition plus grande.

2. Partitions multiples dynamiques

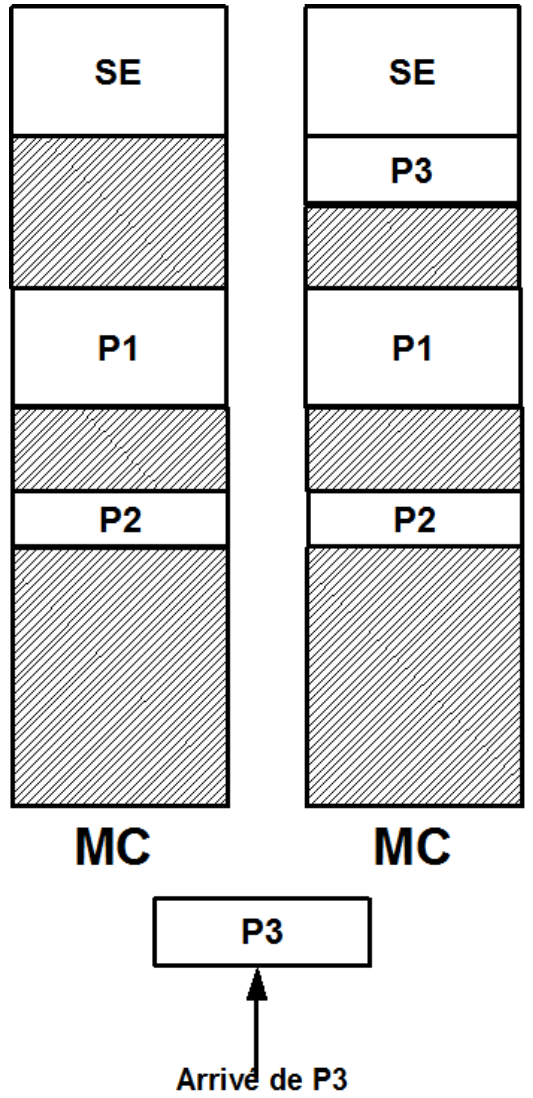


2. Partitions multiples dynamiques

- Le placement des programmes dans les partitions se fait suivant différentes stratégies:

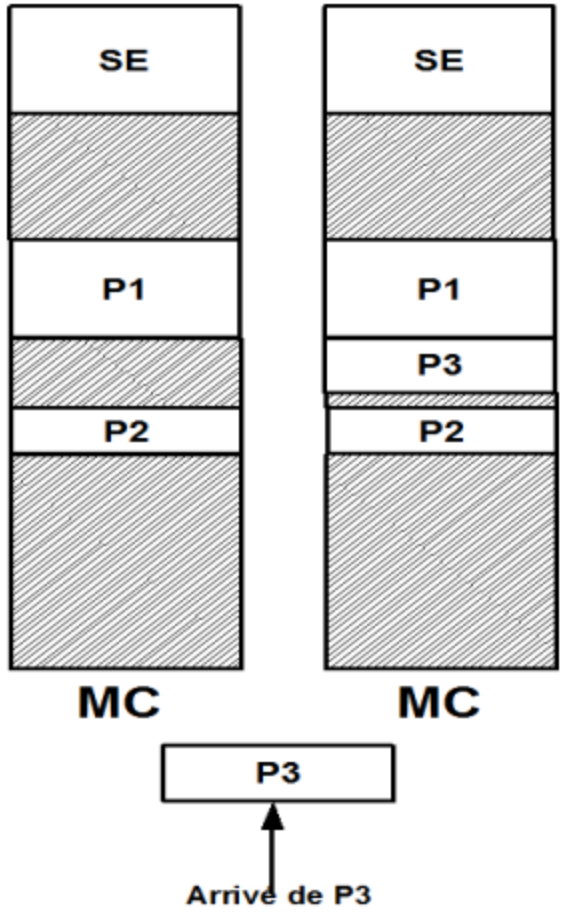
Stratégie du premier qui convient (first fit):

Dans cette stratégie le gestionnaire de mémoire place le programme dans la première partition libre qui a une taille suffisante pour l'exécution du programme.



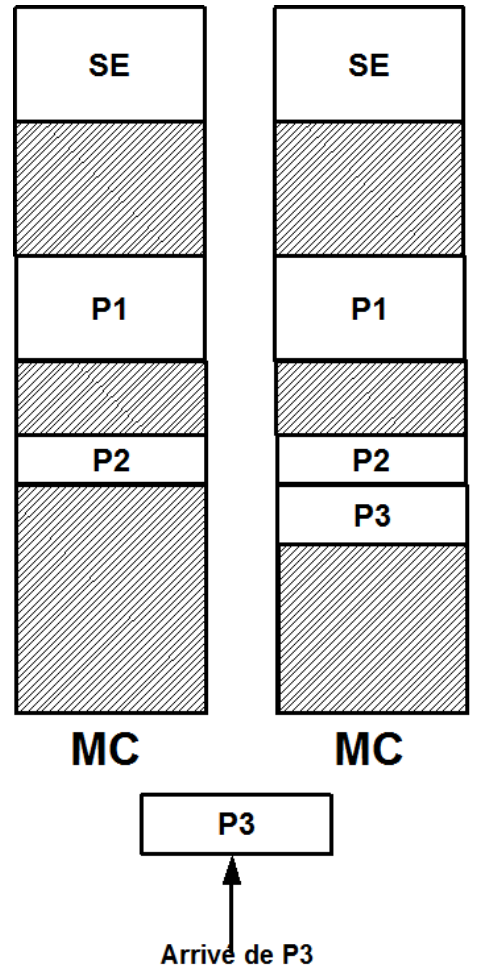
2. Partitions multiples dynamiques

- **Stratégie du meilleur qui convient (best fit):** Dans cette stratégie le gestionnaire de mémoire place le programme dans la partition libre la plus petite qui a une taille suffisante pour l'exécution du programme.



2. Partitions multiples dynamiques

- **Stratégie du pire qui convient (Worst fit):**
  Dans cette stratégie le gestionnaire de mémoire place le programme dans la partition libre la plus grande qui a une taille suffisante pour l'exécution du programme.



2. Partitions multiples dynamiques

La stratégie du **premier qui convient** est **rapide** mais conduit à des **pertes de mémoire** importantes du à la fragmentation.

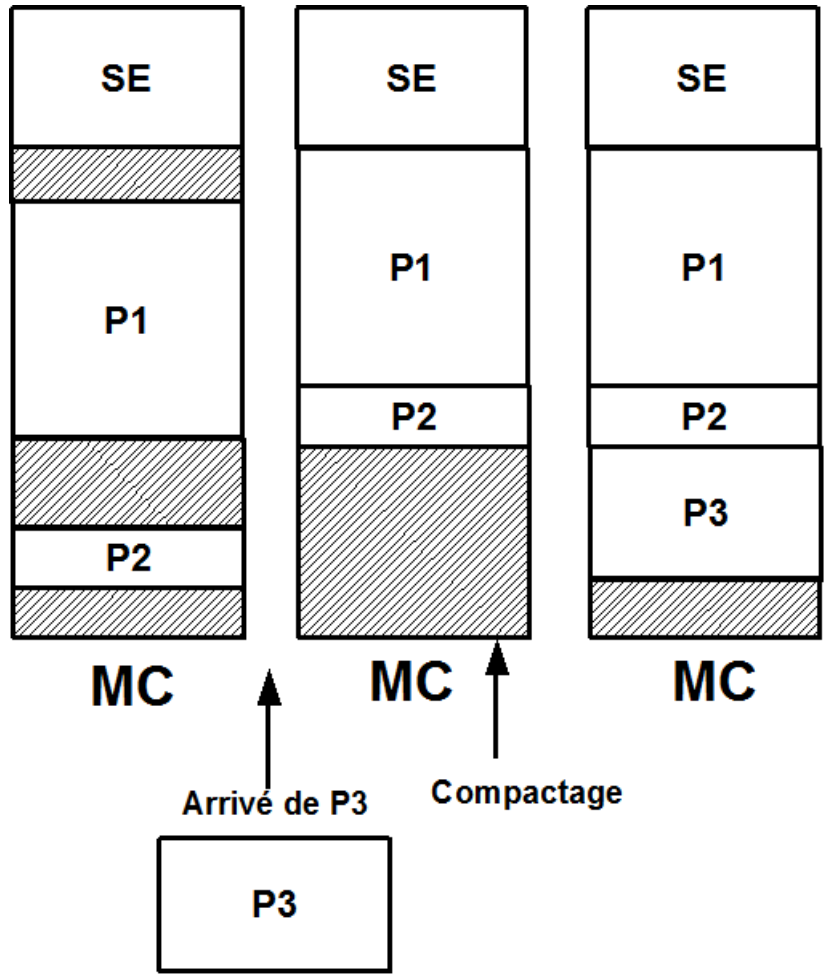
Les stratégies du **meilleur qui convient** et du **pire qui convient** sont **moins rapides** puisque il nécessites la recherche dans toutes les partitions libres.

RQ: Des simulation ont montrés que la stratégie du premier qui convient est meilleur que la stratégie du meilleur qui convient et les deux stratégies sont meilleurs que la stratégie du pire qui convient.

2. Partitions multiples dynamiques

- **Le compactage de la mémoire:**
- Quelque soit la stratégie d'allocation utilisé il y a toujours une fragmentation externe.
- Le compactage consiste a regrouper toutes les partitions mémoires non utilisés en une seule partition plus grande dans la quelle on peut charger les programmes.

- **RQ:** *l'opération de compactage est très coûteuse en temps, si par exemple on a une machine avec 1 Go de RAM et qui peut copier 4 octets en 20 ns, on aura besoin de 5 secondes pour compacter toute la mémoire*

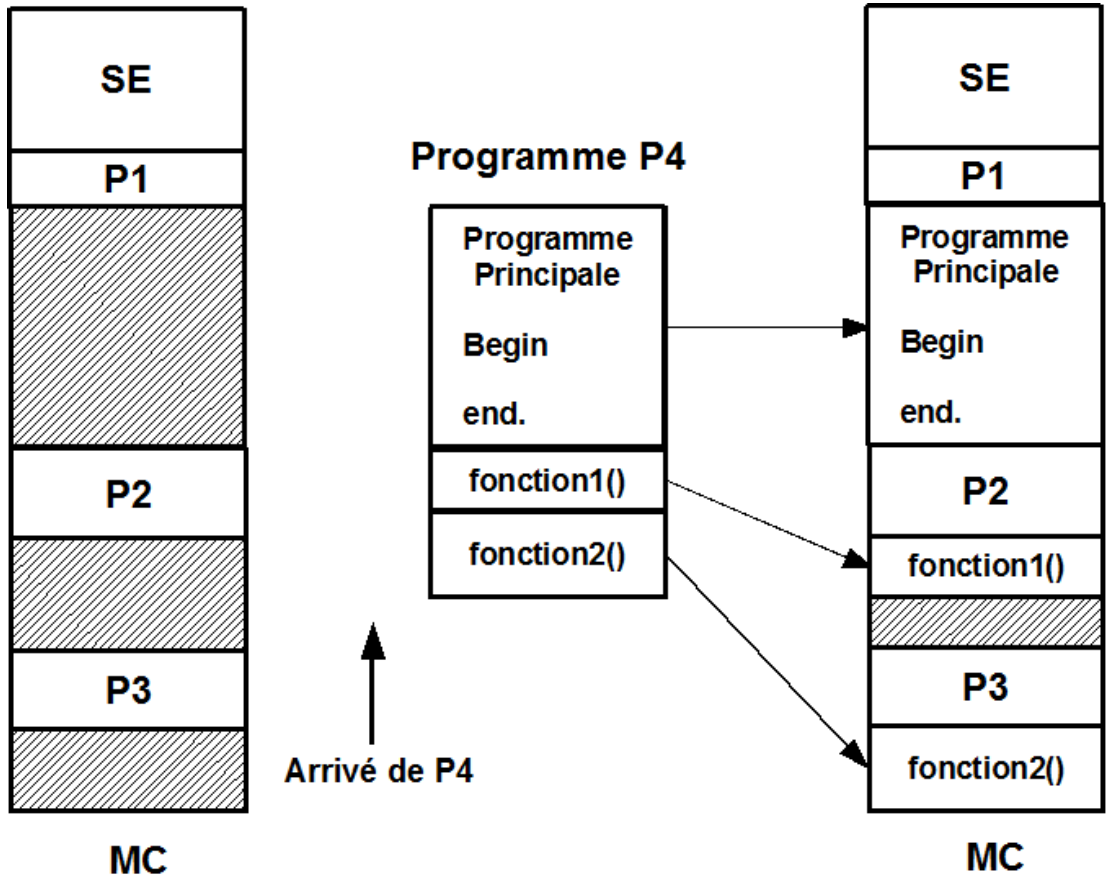


Modes de partage de la mémoire (stratégies d'allocation de la mémoire)

- **la segmentation:**

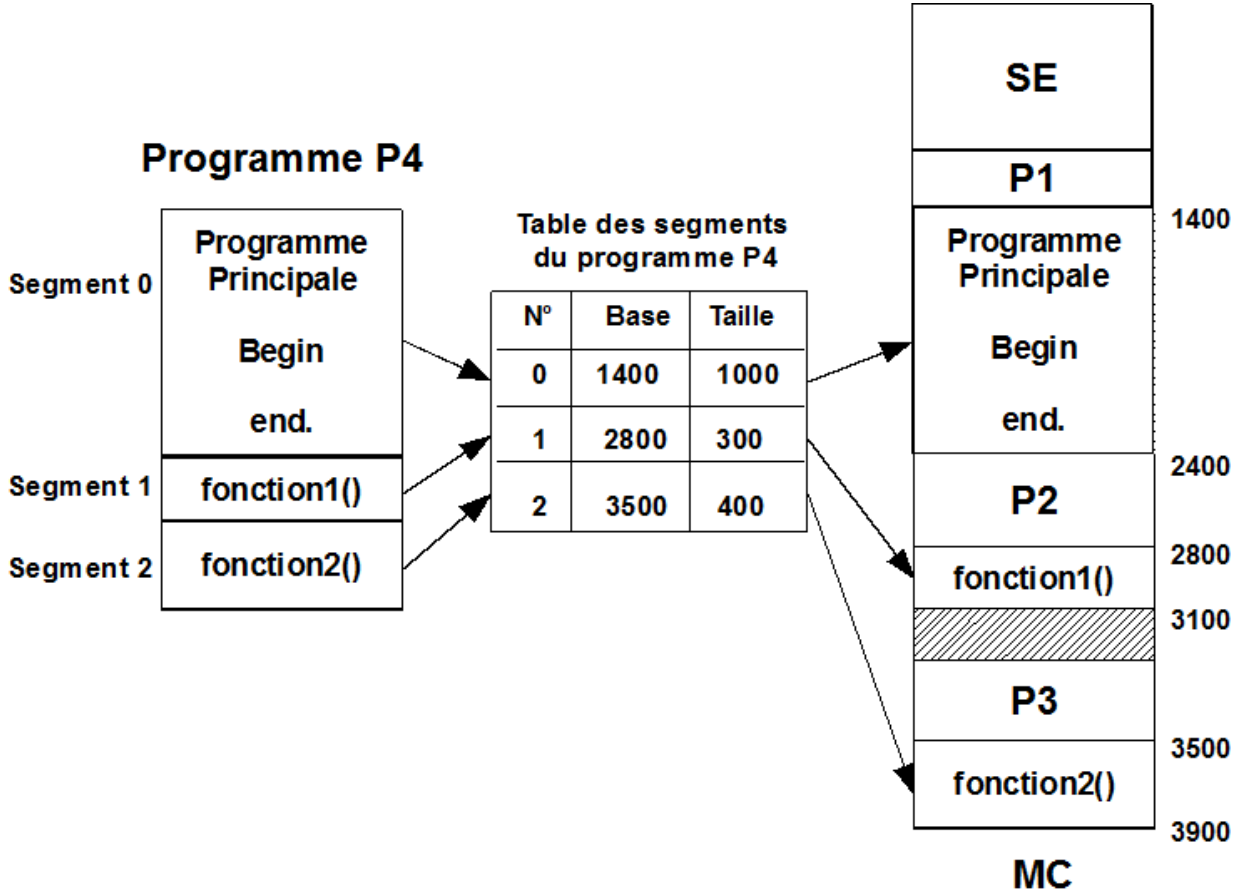
on divise le programme en plusieurs segments, chaque segment correspond a une entité logique (programme principale, procédures et fonctions, structures de données… etc). Un **compilateur pascal** par exemple produit des segments différents pour :

- **Les variables globales,**
- **La pile d'appels de procédures,**
- **Le code de chaque procédure ou fonction,**
- **les variables locales de chaque fonction.**

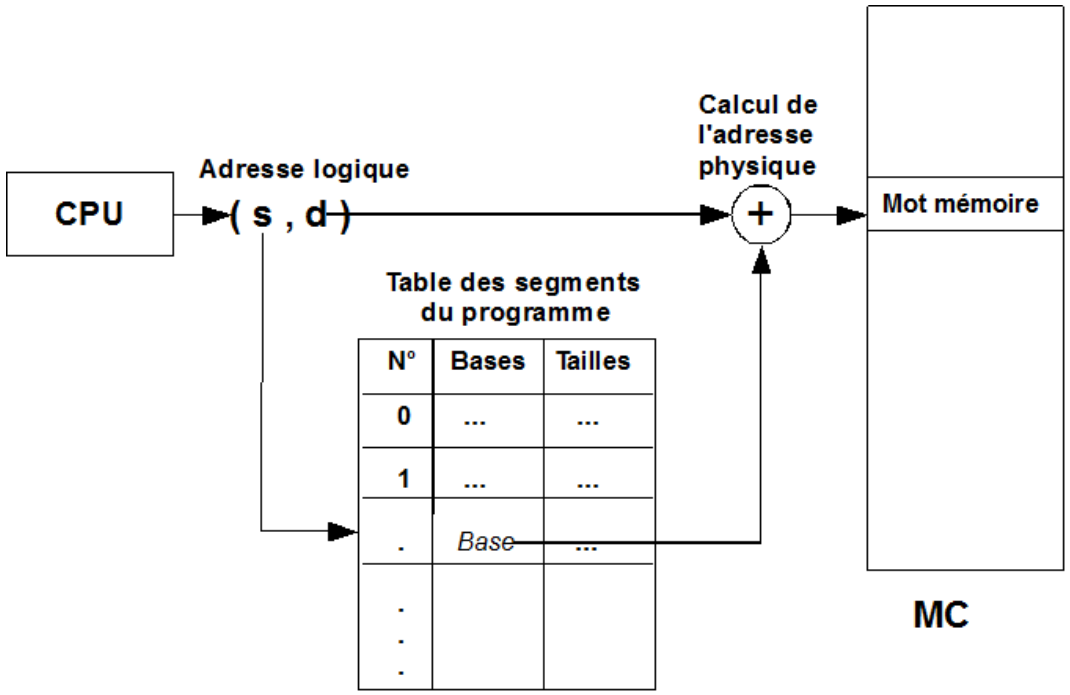


A l'intérieur d'un segment **les adresses sont relatives** au début des segments (**ils commences de 0** jusqu'à la taille du segment).
Les segments d'un même programme peuvent être **chargés dans des espaces mémoire séparés**, la gestion de la mémoire centrale est donc **plus efficace**.

A chaque programme en exécution on associe une **table de segments** qui pour chaque segment du programme donne la **base** (adresse de début) et la **taille** du segment en MC.



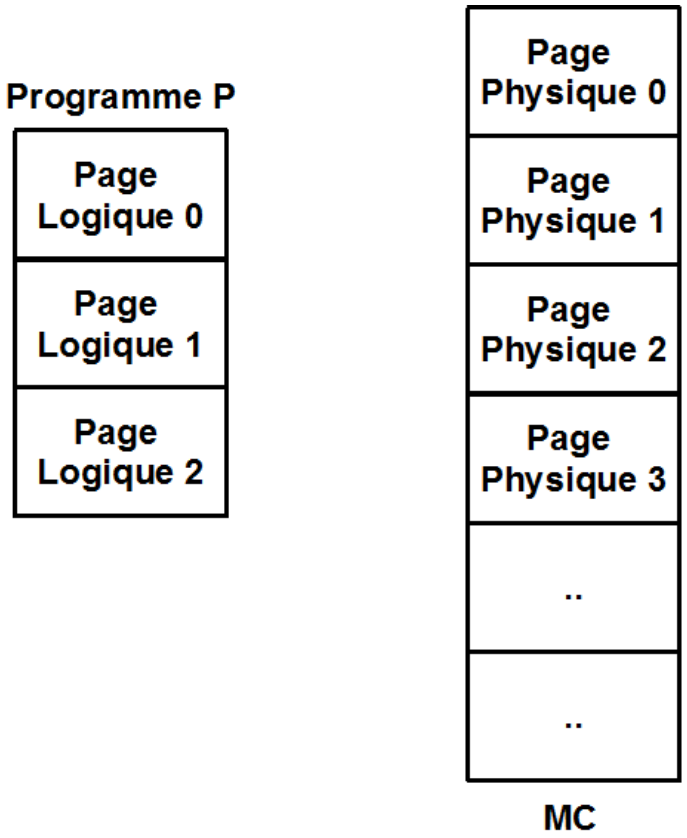
- Les adresses du programme sont des **adresses logiques**, une adresse logique spécifie le segment et le déplacement dans le segment.
- La correspondance entre l'adresse logique du programme et l'adresse physique en mémoire centrale est réalisé en utilisant la table de segments du programme comme suit :
  L'adresse logique est composé du numéro du segments **s**, et du déplacement dans le segments **d**. Adresse logique = <**s,d**>
  Le numéro du segment **s** est utilisé comme indexe dans la table des segments pour trouver l'adresse de base du segment en mémoire centrale, et calculer l'adresse physique.



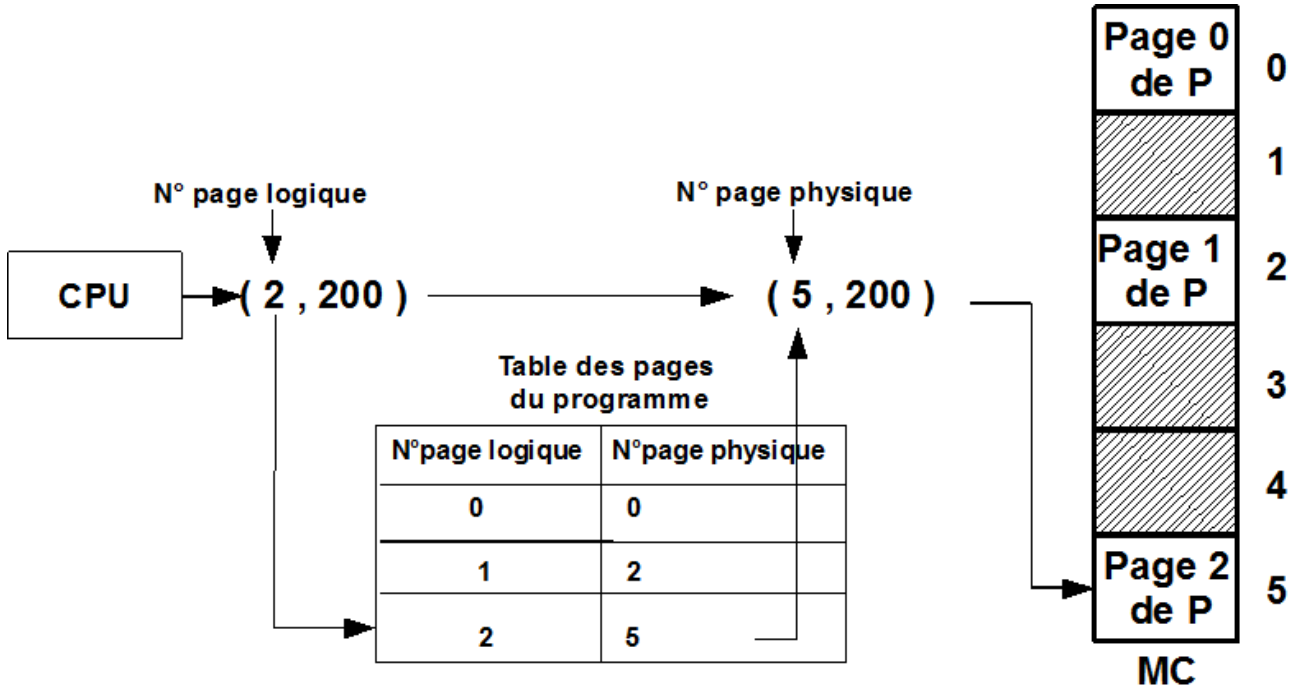
RQ: *Le numéro de segment S doit être inférieur a la longueur de la table des segments, aussi le déplacement dans le segment doit être compris entre 0 et la taille du segment. Dans le cas contraire on aura une erreur d'adressage*

Modes de partage de la mémoire (stratégies d'allocation de la mémoire)

- **La Pagination**
  Dans la pagination la MC est divisé en plusieurs partitions de taille égales et fixes appelées pages physiques. Aussi l'espace d'adressage des programmes est divisé en plusieurs partitions appelés pages logiques. La taille des pages logiques est égale a la taille des pages physiques.
- Un programme peut être chargé dans des pages physiques non contigus, ce qui permet d'éliminer le problème de la fragmentation externe.



- A chaque programme en exécution on associe une **table des pages** qui fait la correspondance entre les pages logiques et les pages physiques en MC .
- Chaque adresse généré par le processeur pendant l'exécution d'un programme est divisé en deux parties , un **numéro de page logique p ,** et **un déplacement d**.



RQ: *La pagination* ***élimine le problème de la fragmentation externe*** *mais le problème de la* ***fragmentation interne*** *reste toujours présent*

Le Swapping

Des fois la taille de la mémoire centrale ne suffit pas pour charger tous les programmes en attente d'exécution. Une solution consiste à sauvegarder temporairement dans une mémoire secondaire (en générale le disque dur) les programme qui sont bloqués (en attente d' E/S ou d'un événement) et charger d'autres programmes dans les partitions libérés.

L'opération est réalisé comme suit :

- Un programme est chargé dans sa totalité en MC (**Swap-in**),
- Le programme restera dans la MC jusqu'à sa terminaison ou son blocage,
- En cas de blocage, le programme peut être transféré en mémoire secondaire (**Swap-out**),
- Un ou plusieurs nouveaux programmes sont chargés dans la partition libéré.

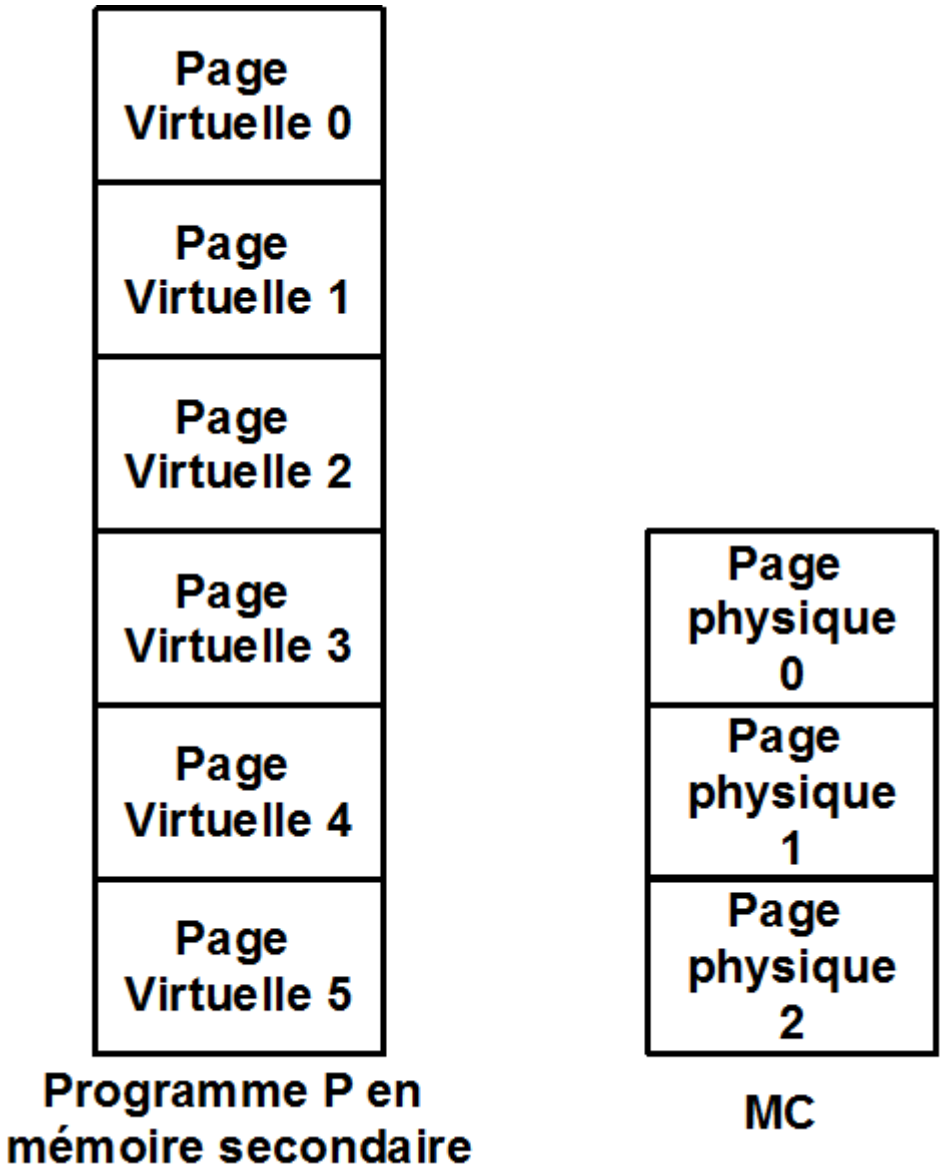
Le swapping est très coûteux en temps à cause du temps de transfère des programmes des mémoires secondaires.

La mémoire virtuelle

- Dans les modes vues précédemment si la taille d'un programme est supérieur a la taille de la MC, le programme ne peut pas être exécuté. La mémoire virtuelle consiste à offrir au programme un espace d'adressage virtuelle qui est indépendant des adresses physique.
- La taille de l'espace d'adressage virtuelle peut être beaucoup plus grande que la taille de la mémoire centrale.
- Pour pouvoir exécuter les programme le système utilise une mémoire secondaire (en générale le disque dur) pour stocker une partie du programme en exécution. Le programme est donc chargé par partie et exécuté en mémoire centrale.
- Pour réaliser une mémoire virtuelle on utilise généralement la **pagination** ou la **segmentation**.

Mémoire virtuelle paginé

- Dans la mémoire virtuelle paginé on découpe l'espace d'adressage virtuelle et l'espace d'adressage physique en pages de même taille. Le nombre de pages virtuelles peut être supérieur au nombre de pages physique. Les pages virtuelles sont sauvegardés en mémoire secondaire et charger en mémoire centrale selon les besoins



Mémoire virtuelle paginé

- Pour faire la translation entre adresse virtuelle et adresse physique en MC on utilise **la table des pages**.
- La table des pages contient des informations concernant chaque page virtuelle d'un processus. Parmi ces informations on cite :

- Un bit de présence qui signale la présence ou l'absence de la page en mémoire centrale.
  0 → la page n'est pas en mémoire centrale,
  1 → la page est en mémoire centrale,
- L'emplacement de la page en Mémoire secondaire,
- L'adresse mémoire de la page si elle est chargé en MC,
- Un bite de modification qui indique si la page a était modifié ou non. Si la page a était modifié elle doit être recopié en mémoire secondaire.

Mémoire virtuelle paginé

- La translation entre adresse virtuelle et adresse physique est réalisé par un matérielle spécifique intégré actuellement au processeur appelé unité de gestion de mémoire (MMU : *Memory Management Unit*).
- Cette unité intercepte l'adresse virtuelle généré par le processeur (qui accède à une données ou à une instruction) et la transforme en adresse physique correspondante.

Mémoire virtuelle paginé

- L'exécution d'un processus de translation entre adresse virtuelle et adresse physique se fait comme suit :
- L'adresse virtuelle d'une instruction est donnée par le compteur ordinale,
- L'adresse virtuelle est découpé en numéro de page p , et déplacement d.
- Le numéro de page p est utilisé comme indexe pour accéder à la table des pages.
- Le système vérifie si la page référencé p est présente en MC en utilisant le bit de présence dans la table des page.
- Si la page n'est pas en mémoire centrale , on dit qu'il y a un défaut de page , dans ce cas le système doit charger la page correspondante de la mémoire secondaire en MC.
- Si la page référencé est en MC on calcule l'adresse physique a partir du numéro de page physique et du déplacement d.
- On accède enfin à la MC.

Stratégies de remplacement des pages

- Quand une référence mémoire génère **un défaut de page** (quand la page n'est pas en MC), le système doit libérer de la mémoire pour charger la page référencé.
- La stratégie de remplacement précise quelle page on doit faire sortir de la MC pour libérer de l'espace. La page choisit est appelé **page victime**.

- **Algorithme FIFO**
- **Algorithme LRU (Least Recently used)**
- **Algorithme de remplacement optimal**
- **Algorithme de la seconde chance (Horloge)**

Stratégies de remplacement des pages

- **Algorithme FIFO**
- Cette algorithme choisit la page la plus ancienne en MC.
  Avantage : Il est très facile a réalisé,
  Inconvénient : la page la plus ancienne est remplacé même si elle est fréquemment utilisé.

Stratégies de remplacement des pages

- **Algorithme LRU (Least Recently used)**
- Cette algorithme choisit la page la moins récemment utilisé.
  Avantage : On réduit le nombre de défaut de pages,
  Inconvénient : difficile à réaliser puisque on doit pouvoir ordonner les pages selon leurs utilisation pour savoir la moins récemment utilisé.

Stratégies de remplacement des pages

- **Algorithme de remplacement optimal**
- Dans cette algorithme la page qui ne sera pas référencé durant le plus grand temps à venir sera choisit.
  Avantage : c'est l'algorithme optimal qui réduit le maximum possible les défauts de pages,
  Inconvénient : difficile d'implémenter cet algorithme puisque il est difficile de prévoir les références futures.

Stratégies de remplacement des pages

- **Algorithme de la seconde chance (Horloge)**
- Cet algorithme est basé sur l'algorithme FIFO, il tente d'éviter qu'une page ancienne qui est récemment référencé ne soit choisit. Le principe de l'algorithme est comme suit :
  - On associe à chaque page un **bit de référence** stocké dans la table des pages et qui est mis à 1 par le matériel à chaque fois que la page est utilisé.
  - Quand un défaut de page se produit, la page la plus ancienne selon l'algorithme FIFO est vérifié.

  - Si le bit de référence est égale a 0 elle sera remplacé sinon son bit de référence est remis à 0 et son temps d'arrivé est changé en temps courant (temps actuelle), on vérifie ensuite la page suivante selon l'algorithme FIFO.
  Si une page est utilisé souvent, son bit de référence est maintenu à 1 et elle ne sera pas remplacé.

Protection de la mémoire

- Dans un système informatique on doit pouvoir protéger des bloc de mémoire contres des accès non autorisés suite par exemple à des erreurs de programmation. Il existe plusieurs techniques de protection :
  Cas d'une seul zone contigue
  Dans ce cas on a deux zone mémoire, la zone du système d'exploitation et la zone du programme utilisateur .

  la partition du système d'exploitation doit être protégé contre les accès du programmes utilisateur, cette protection est réalisé par un matériel spécifique qui vérifie chaque adresse référencé avec une adresse limite. L'adresse référencé doit être supérieur ou égale a l'adresse limite de la zone du système d'exploitation.

Protection de la mémoire

- **Cas de partitions multiples fixes**
  Dans ce cas on utilise deux registres qui indiquent la plus petite et la plus grande adresse qui limites le programme. Chaque adresse référencé est vérifié en utilisant ces deux registres.
- **Cas de la segmentation et pagination**
  Dans ce cas des bites sont associés a chaque page ou segment et indiquent si le processus peut ou non accéder à ses pages ou à ces segments.

Partage de code entre les programmes

- Le partage de code entre les programme permet d'économiser l'espace mémoire et le temps de chargement des programmes.
- Soit par exemple un système qui supporte 40 utilisateurs dont chacun exécute un éditeur de texte. Si le code de l'éditeur de texte est 30 Ko et 5 Ko pour les données, on aura besoin de (30+5)x40=1400 Ko de mémoire pour satisfaire tous les utilisateurs. Par contre si on partage le code de l'éditeur entre tous les utilisateurs on aura seulement besoin de (30 + (5 x 40)) = 230 Ko de mémoire pour satisfaire les utilisateurs.
- Pour pouvoir partager une zone mémoire entre plusieurs processus il faut que cette zone ne soit pas modifié pendant l'exécution des processus (accès seulement en lecture).

- Dans le cas de la pagination, affin de partager un ensemble de pages entre plusieurs processus, les pages sont chargés en mémoire une seule fois et une entrée pour chaque page partagé sera ajouté dans la table des pages de chaque processus.
- Dans le cas de la segmentation , affin de partager un ou plusieurs segments entre plusieurs processus, les segments sont chargés en mémoire une seule fois et une entrée pour chaque segment partagé sera ajouté dans la table des segments de chaque processus.