République Algérienne Démocratique et Populaire
Ministère de l'Enseignement Supérieur et de la Recherche Scientifique
L'université Mohamed Boudiaf - M'Sila –

Faculté de mathématiques et d'informatique

2ème Année Licence (2L)

Système d'exploitation 1 (SE 1)

Semestre : 04

2023/2024

Réalise par

Dr. DABBA ALI

➢ Contactez-nous

alidabba@gmail.com

ali.dabba@univ-msila.dz

- En cas de problèmes ou de difficultés, me contacter ou contacter votre enseignant TD / TP
- Nous sommes à votre disposition pour vous aider

CHAPITRE 4

Gestion de la mémoire

Plan

I. Introduction
II. Les mémoires
III. Objectifs du gestionnaire de la mémoire centrale
IV. Espace d'adressage logique et physique
V. Gestion de la MC dans les systèmes mono-programmés
VI. Gestion de la MC dans les systèmes multiprogrammés

I. Introduction

La mémoire physique sur un système se divise en deux catégories :

- La mémoire vive
- La mémoire de masse (secondaire)

I. Introduction

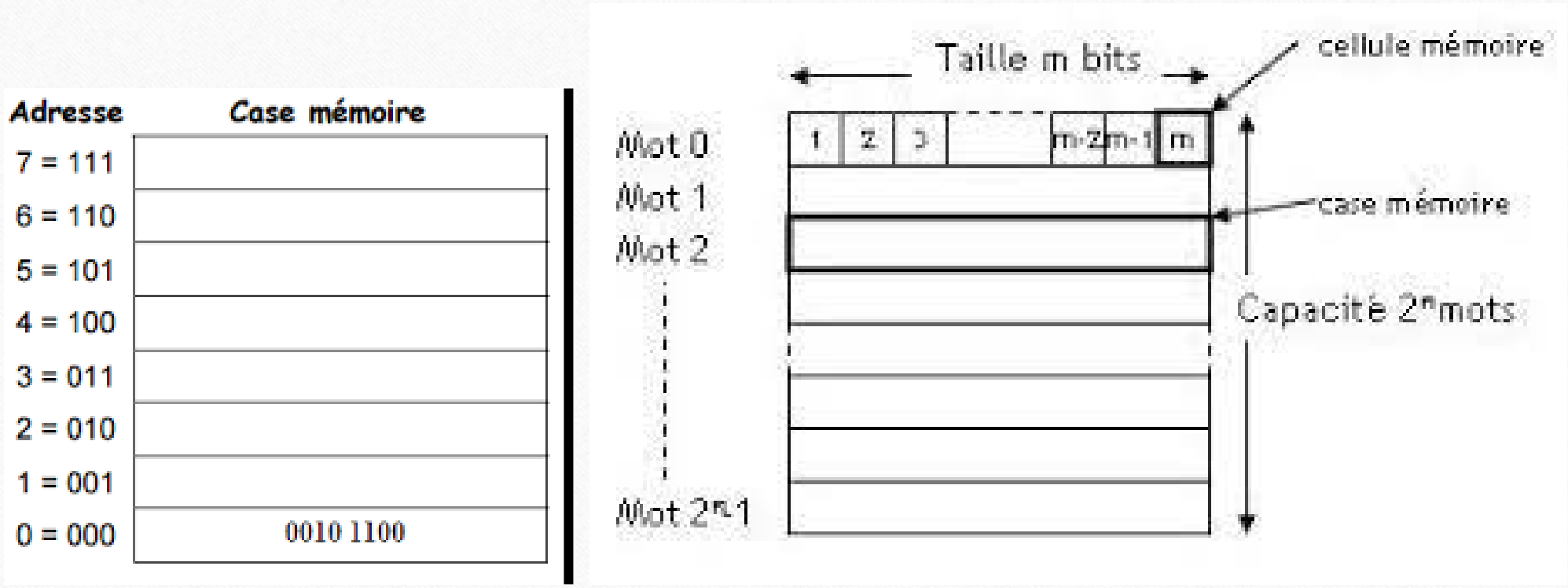
Pour maximiser le rendement de l'UC dans un système multiprogrammé, plusieurs questions se pose concernant :

- Comment partager les données entre les processus ?
- Comment protéger l'espace mémoire de chaque processus ?
- Qui et comment faire l'allocation d'espace mémoire aux process ?
- Comment et quand la Restitution de l'espace libéré par le process se fait ?

Gestionnaire de la MC

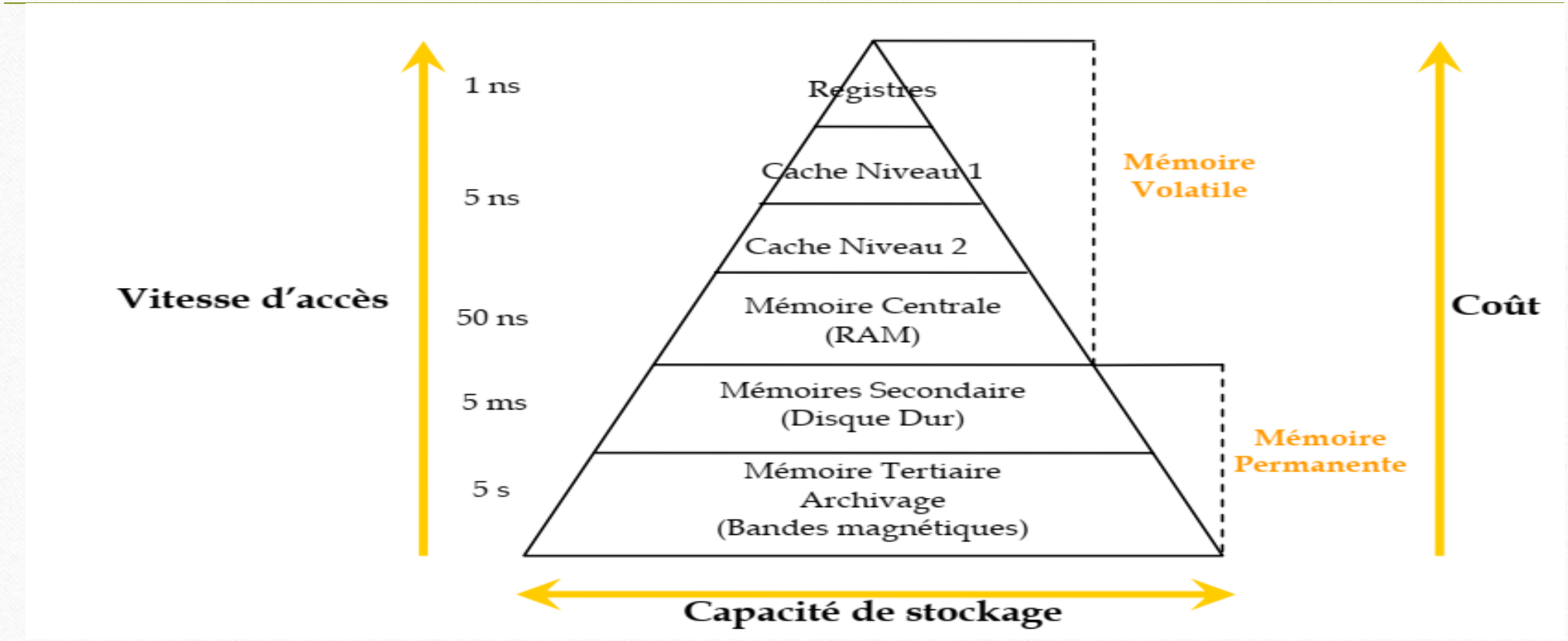
Memory Management Unit (MMU).

II. Les mémoires



Organisation de la mémoire

II. Les mémoires



La hiérarchie mémoire.

II. Les mémoires

La mémoire contient principalement

- Deux types d'informations :
  - **Les instructions** (informations traitantes : commandes)
  - **Les opérandes** (informations traitées)
- Deux types d'opérations peuvent s'effectuer sur les mots mémoires
  - **lecture (Read)**
  - **l'écriture (Write)**

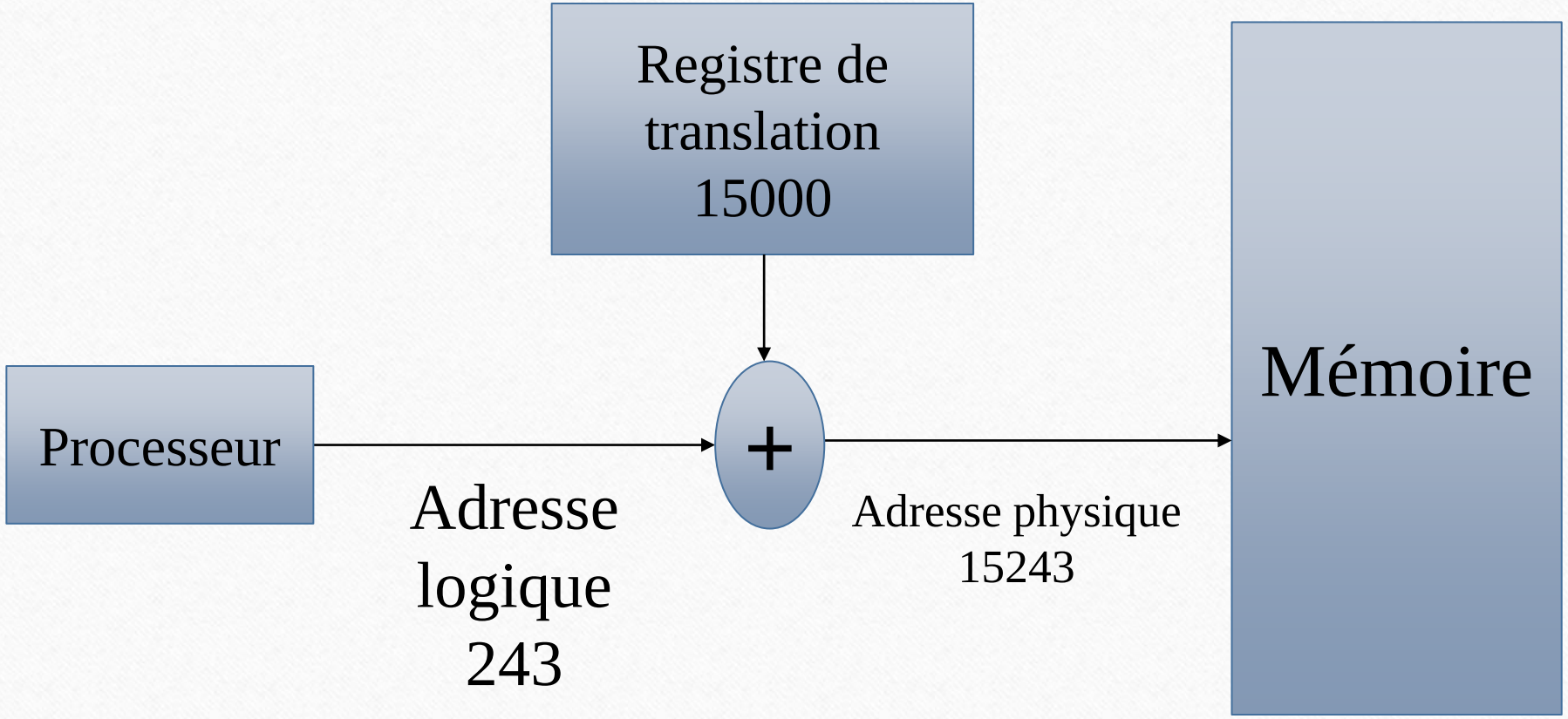
III. Objectifs du gestionnaire de la mémoire centrale

- Organisation logique de la MC (Adressage)
- Allocation et Réallocation
- La libération
- La protection
- Partager des informations
- Surpasser les limites physiques de la capacité de la MC.

IV. Espace d'adressage logique et physique

- **Espace d'adressage logique :** Une adresse générée par la CPU est appelée « adresse logique ».
- **Espace d'Adressage Physique :** Une adresse vue par l'unité mémoire (c'est-à-dire celle chargée dans le registre d'adresses mémoire de la mémoire) est communément appelée « Adresse Physique ».

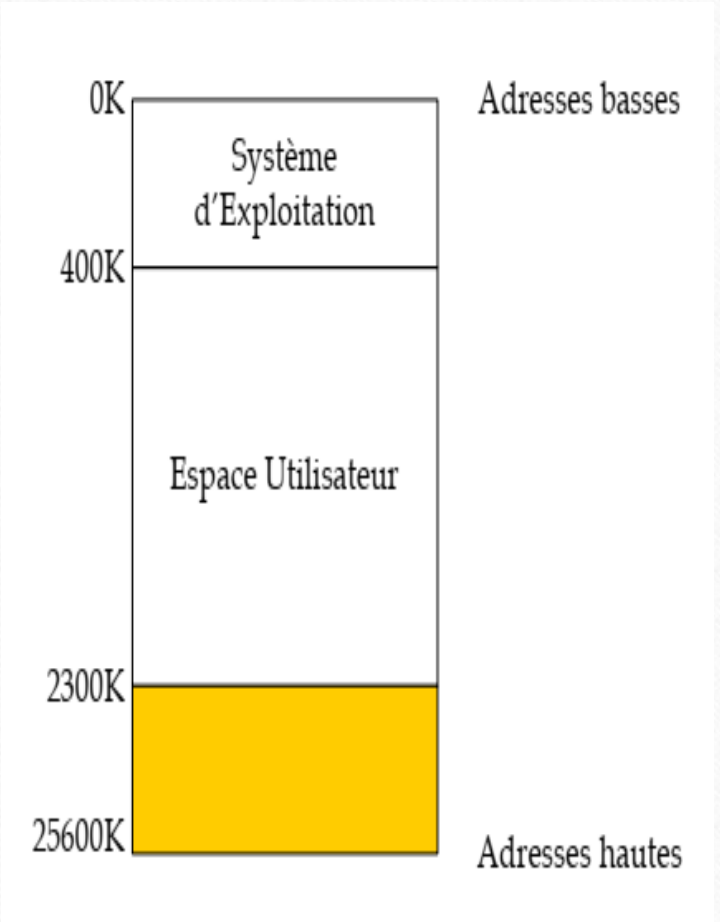
IV. Espace d'adressage logique et physique



Conversion d'adresses logiques en adresses physiques par translation

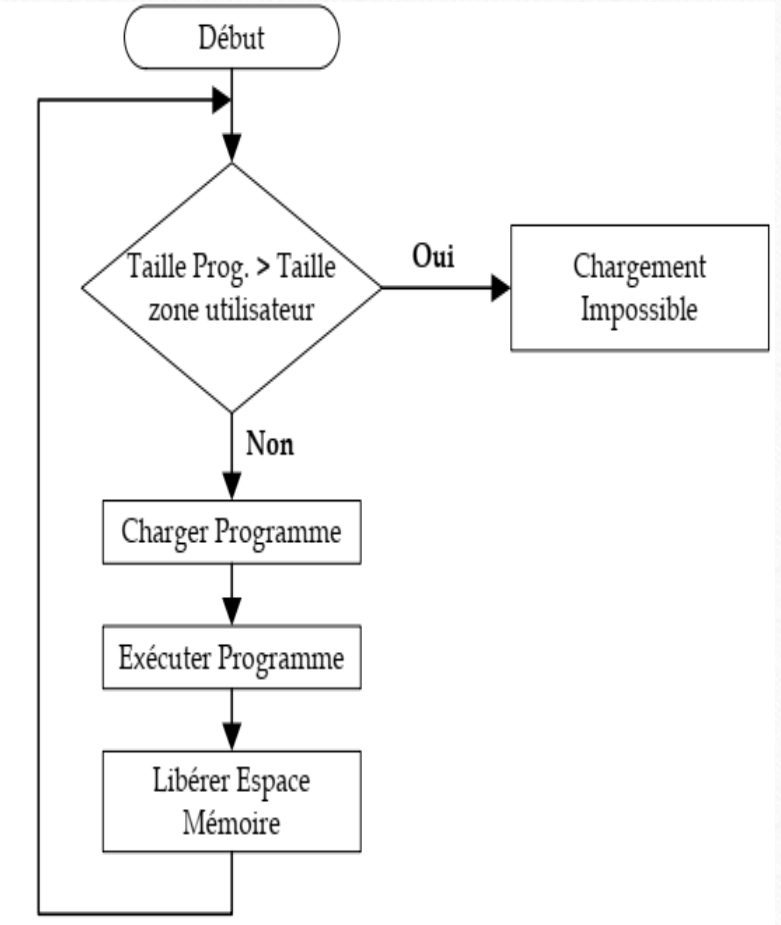
V. Gestion de la MC dans les systèmes mono-programmés

- **Deux partitions contiguës,**
  - une pour le système d'exploitation
  - l'autre pour le processus utilisateur.
- Elle n'autorise qu'un seul processus actif en mémoire à un instant donné dont tout l'espace mémoire usager lui est alloué.



V. Gestion de la MC dans les systèmes mono-programmés

- **Deux partitions contiguës**,
- La gestion de la mémoire
- Simple
- SE doit garder trace de deux zones mémoires.
- L'inconvénient majeur de cette stratégie est la sous-utilisation (i.e. mauvaise utilisation) de la mémoire



V. Gestion de la MC dans les systèmes mono-programmés

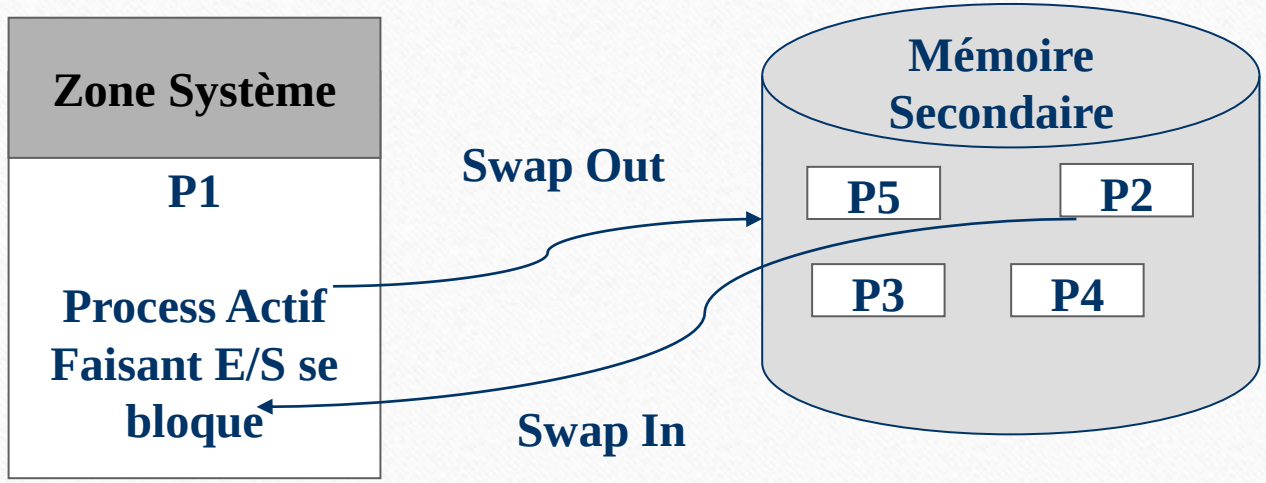
1. Problématique

- En monoprogrammation, si le process actif, le seul en MC, fait une opération d'E/S Bloquante ➔ l'UC est oisive pendant toute la durée de l'E/S même si le système d'E/S est doté de DMA (Direct Memory Access) !
- Si l'on admet l'exécution mono-process, on ne peut pas admettre que l'UC soit oisive pendant une longue durée. Il faut trouver un moyen qui permet de remplir ce temps d'oisiveté.

V. Gestion de la MC dans les systèmes mono-programmés

2. Swapping (ou méthode de va et vient)

Le principe de Swapping se base sur l'état d'un processus, si le processus P1 se bloque suite à une E/S, un autre processus P2 est chargé pour s'exécuter



V. Gestion de la MC dans les systèmes mono-programmés

3. Technique de recouvrement

La taille d'un programme peut dépasser la taille de la MC. Pour surpasser cette limitation dans un système mono-programmé, le programmeur doit diviser son programme au moment de la conception en un ensemble de modules et les charger dynamiquement à l'exécution en MC de telle sorte qu'il garde que les modules dont il a besoin effectivement. Chaque nouveau module chargé prend la place du module qui doit être déchargé.

VI. Gestion de la MC dans les systèmes multiprogrammés

- La multiprogrammation permet l'exécution de plusieurs processus à la fois (la présence de plusieurs processus en mémoire).
- Elle permet d'optimiser le taux d'utilisation du processeur en réduisant notamment les attentes sur des entrées-sorties.

Comment organiser la mémoire de manière à faire cohabiter efficacement plusieurs processus tout en assurant la protection des processus ?

VI. Gestion de la MC dans les systèmes multiprogrammés

Deux cas sont alors à distinguer :

1. Approche d'allocation contiguë

- Un programme→ ensemble de mots mémoires contiguës inséparables (insécables).

On trouve dans cette approche principalement deux types de technique:

- **Technique d'allocation par partitions fixes.**
- **Technique d'allocation par partitions variables.**

VI. Gestion de la MC dans les systèmes multiprogrammés

Deux cas sont alors à distinguer :

2. Approche d'allocation non contiguë

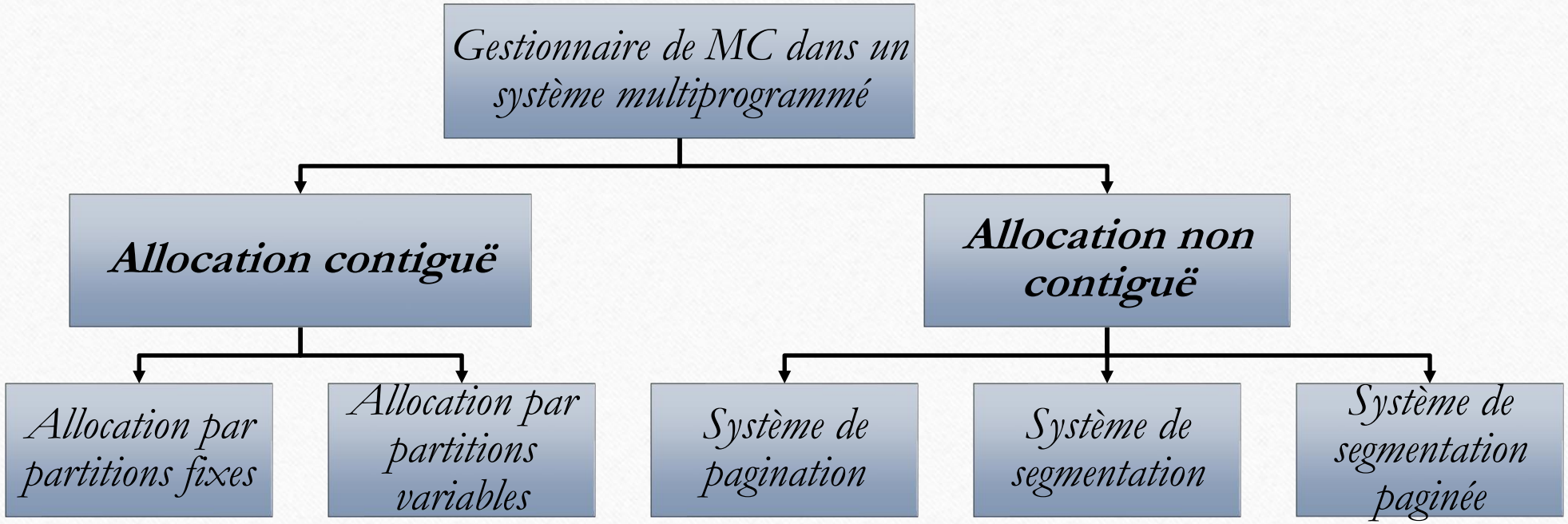
- Un programme → ensemble de mots mémoires non contiguës séparables(sécables).

On trouve dans cette approche principalement les technique de 7

- **pagination,**
- **segmentation**
- **la segmentation paginée.**

VI. Gestion de la MC dans les systèmes multiprogrammés

Deux cas sont alors à distinguer :



VI. Gestion de la MC dans les systèmes multiprogrammés

Allocation contiguë (Contiguous Allocation)

- Cette stratégie constitue une technique simple pour la mise en œuvre de la **multiprogrammation**. La mémoire principale est divisée en régions séparées ou partitions mémoires ; chaque partition dispose de son espace d'adressage. Le partitionnement de la mémoire peut être **statique (fixe)** ou **dynamique (variable).**
- Chaque processus est chargé entièrement en mémoire. L'exécutable contient des adresses relatives et les adresses réelles sont déterminées au moment du chargement.

VI. Gestion de la MC dans les systèmes multiprogrammés

Allocation contiguë par partitions fixes

- Cette solution consiste à diviser la mémoire en partitions fixes, de tailles pas nécessairement égales, à **l'initialisation du système**.
- L'un des problèmes de conception de tel système est la détermination du nombre et des tailles adéquats.

➔ utilisant une table de partions.

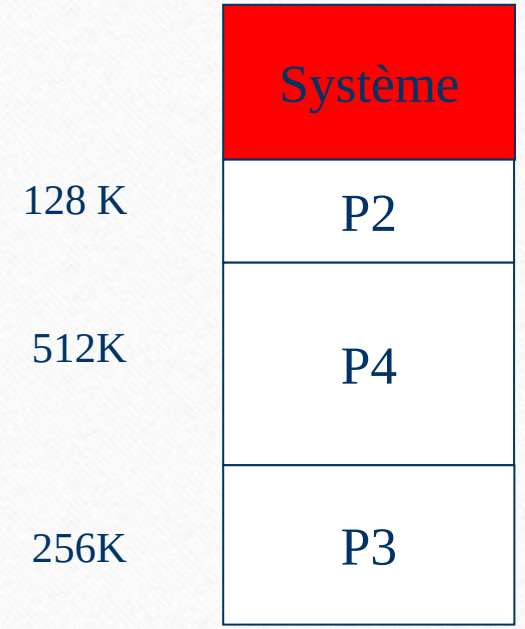
➔ Le système d'exploitation maintient une **table de description des partitions** (PDT, Partition Description Table) indiquant les parties de mémoire disponibles et celles qui sont occupées.

VI. Gestion de la MC dans les systèmes multiprogrammés

Allocation contiguë par partitions fixes

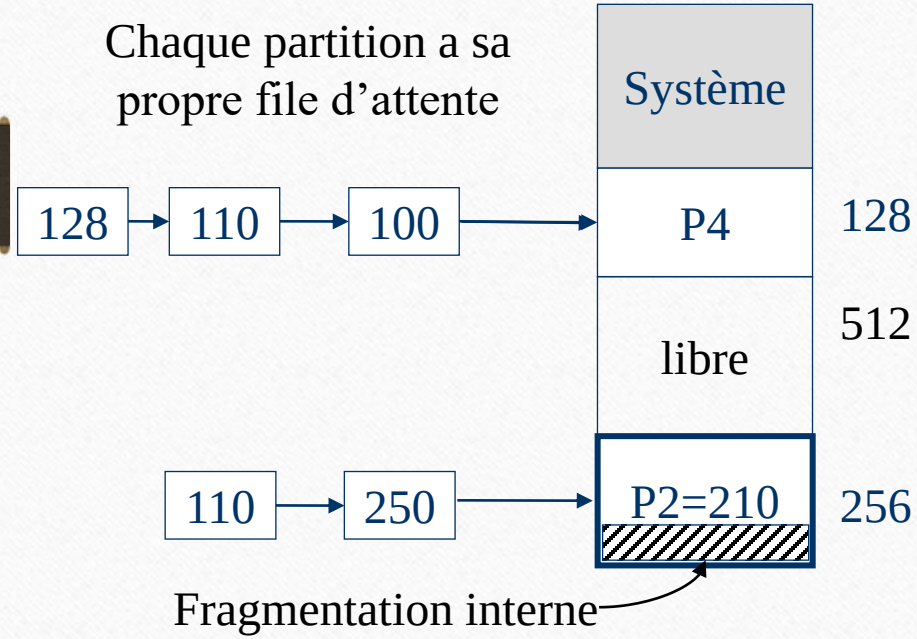
table des partions

	Taille	État d'allocation
1	128	**Libre**
2	512	**Occupée**
3	**256**	**Occupée**

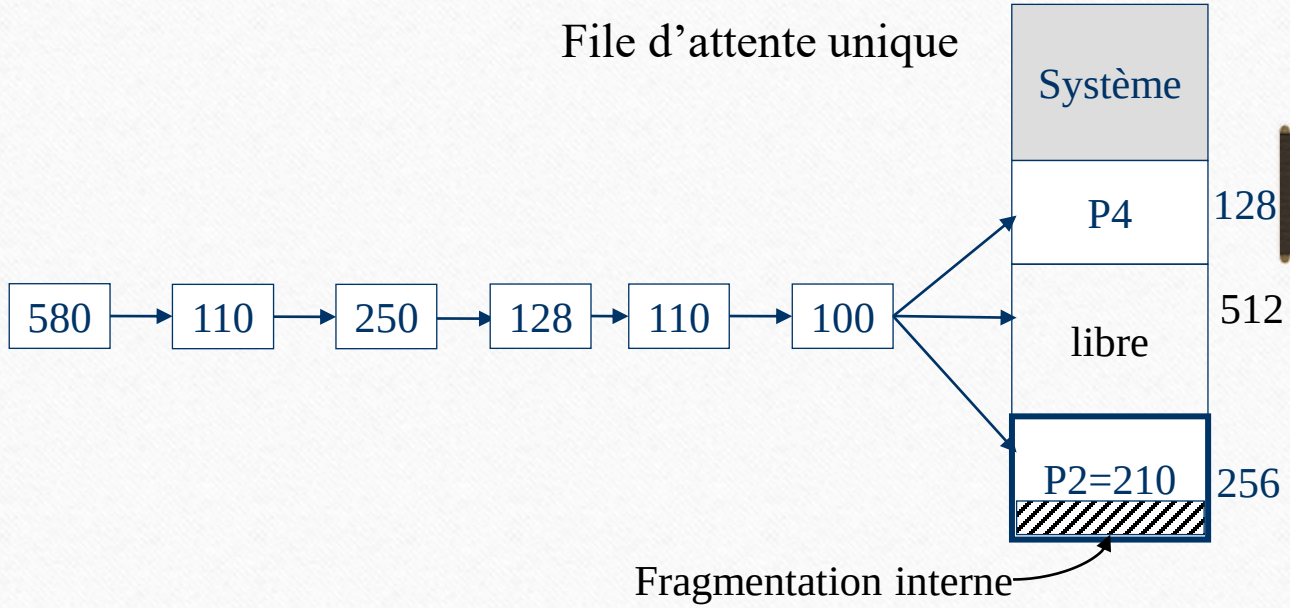


VI. Gestion de la MC dans les systèmes multiprogrammés

Allocation contiguë par partitions fixes



Partitions fixes avec files multiples.



Partitions fixes avec une seule file.

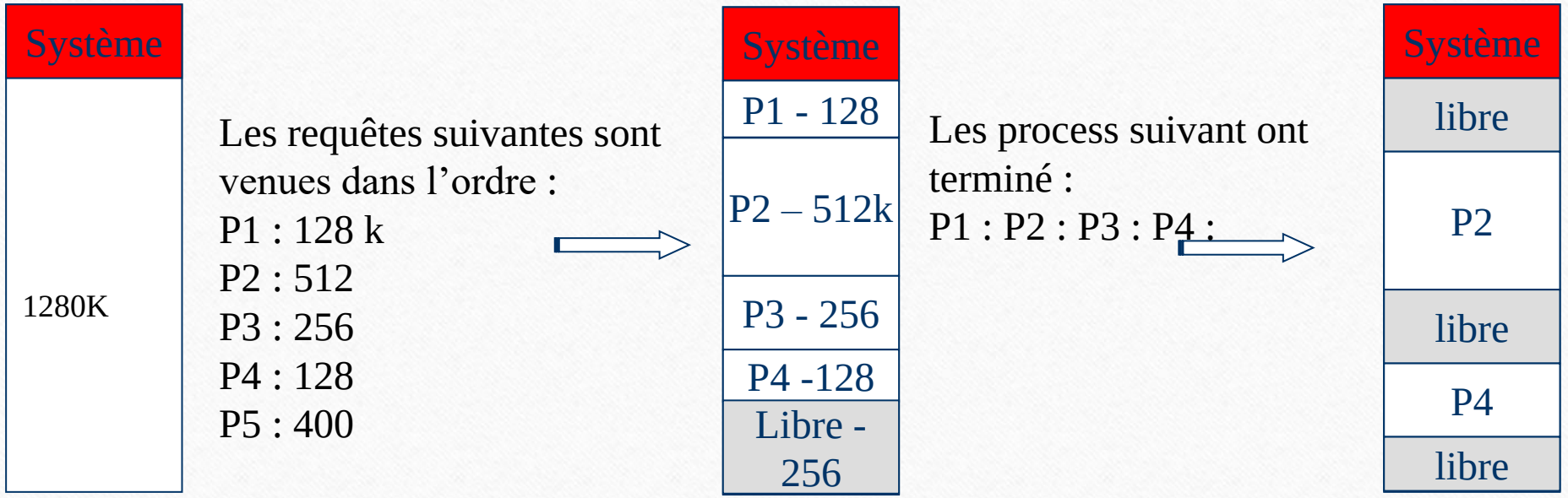
VI. Gestion de la MC dans les systèmes multiprogrammés

Allocation contiguë par partitions variables

- Dans ce cas, la mémoire est découpée dynamiquement **(partitions variables), suivant la demande** des processus et au moment de chargement.
- Ainsi, à chaque programme est allouée une partition exactement égale à sa taille.
- Quand un programme termine son exécution, sa partition est récupérée par le système pour être allouée à un autre programme complètement ou partiellement selon la demande.

VI. Gestion de la MC dans les systèmes multiprogrammés

Allocation contiguë par partitions variables



Au début l'espace user est vide

P5 en attente

P5 en attente

Inconvénients : Fragmentation Externe

Exemple d'allocation des partitions variables et fragmentation externe.

VI. Gestion de la MC dans les systèmes multiprogrammés

Gestion de l'espace par table de bits (bitmaps)

On divise la MC en un ensemble de blocs d'allocation de quelques octets à quelques Ko. Pour gérer ces blocs, on prévoit pour chacun un bit indiquant sa disponibilité :

- La valeur 1 si le bloc d'allocation (l'unité mémoire) est occupé ;
- La valeur 0 si le bloc d'allocation (l'unité mémoire) est libre.

L'ensemble de ces bits constitue ce qu'on appelle table de bits ou mapping table. Cette table est stockée en MC dans la zone système (protégée). En augmentant la taille de l'unité d'allocation, on réduit la taille de la table de bits mais on perd beaucoup de place mémoire (fragmentation interne).

VI. Gestion de la MC dans les systèmes multiprogrammés

Gestion de l'espace par table de bits (bitmaps)

Inconvénient

Lorsqu'on doit ramener un processus de **k** unités, le gestionnaire de la mémoire doit alors parcourir la table de bits à la recherche de **k** zéros consécutifs. Cette technique est rarement utilisée car la méthode de recherche est lente

VI. Gestion de la MC dans les systèmes multiprogrammés

Gestion de l'espace par Liste Linéaire Chaînée

L'espace mémoire est géré par une liste linéaire chaînée des segments libres et occupés. Un segment est un ensemble d'unités d'allocations consécutives. Un élément de la liste est composé de quatre champs qui indiquent :

- L'état libre ou occupé du segment (processus).
- L'adresse de début du segment.
- La longueur du segment.
- Le pointeur sur l'élément suivant de la liste.

VI. Gestion de la MC dans les systèmes multiprogrammés

Gestion de l'espace par Liste Linéaire Chaînée

La figure 4.12 montre l'exemple d'un tel chaînage, les segments occupés par un processus sont marqués (P), les libres sont marqués (H).

En résumé, les listes chaînées sont une solution plus rapide que la précédente pour l'allocation, mais plus lente pour la libération.

VI. Gestion de la MC dans les systèmes multiprogrammés

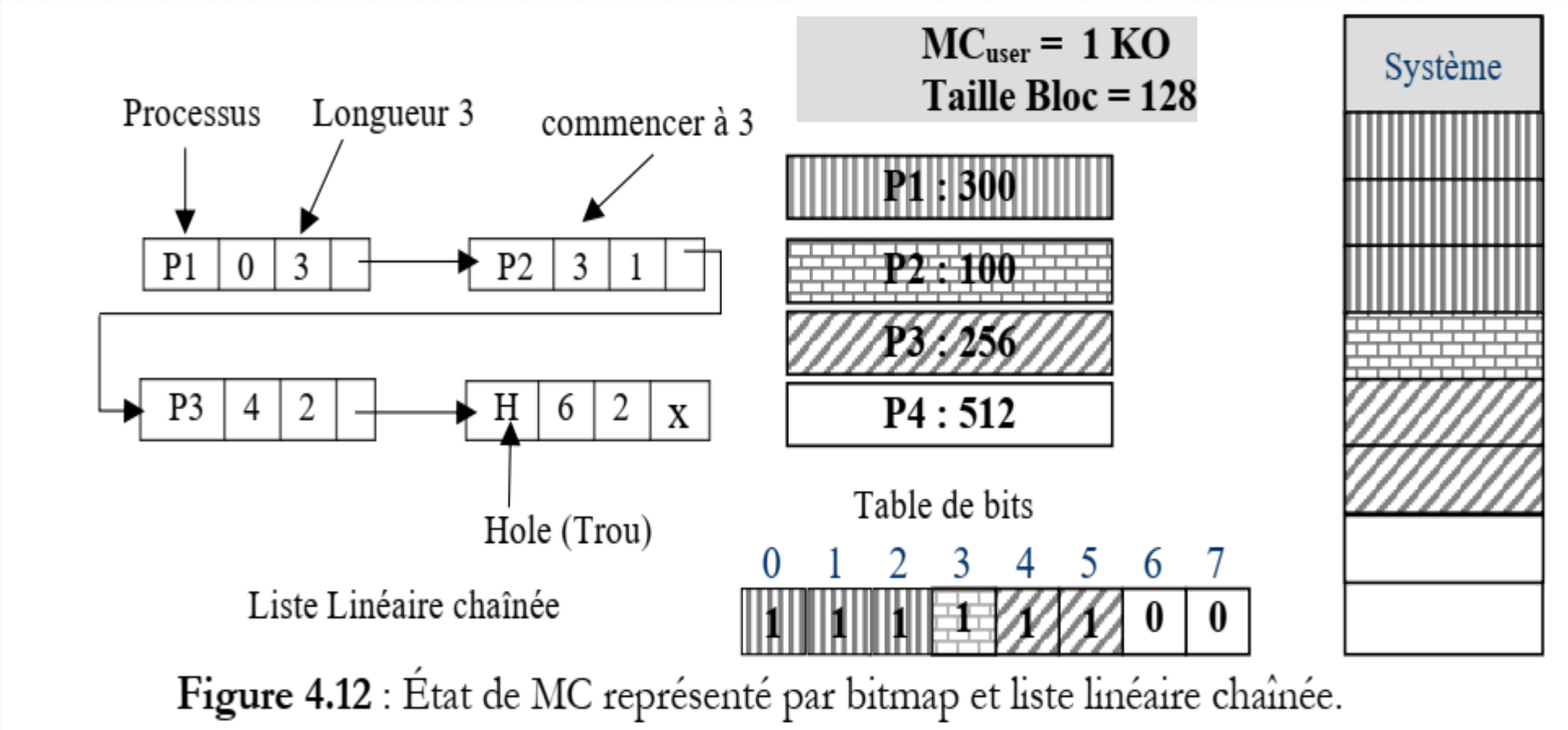


Figure 4.12 : État de MC représenté par bitmap et liste linéaire chaînée.

VI. Gestion de la MC dans les systèmes multiprogrammés

Stratégies d'allocation

1. **First Fit :** Parcourir la liste des zones libres jusqu'à la rencontre de la première zone supérieur ou égale à la demande.

➢ Recherche rapide ⇔ Allocation rapide

➢ La Zone choisie n'est pas toujours la bonne ➔ Possibilité de bloquer d'autres process dont la Zone choisie pouvait les satisfaire. Exemple : si P5 vient demandant 130 ??

Cette stratégie tend à ordonner la liste selon les adresses croissantes des zones libres.

VI. Gestion de la MC dans les systèmes multiprogrammés

Stratégies d'allocation

2. **Best Fit** : Choisir une zone dont la taille est la plus proche de la demande, ce qui implique un parcours complet de la liste.

- La Zone choisie est la meilleure
- Fragmentation Externe de petites tailles qu'elles peuvent être au point où ces petits espaces seront rarement réutilisables.
- Recherche lente ⇔ Allocation lente

On a intérêt à Ordonner la liste par ordre croissant des tailles pour éviter la parcours complet de la liste et accélérer ainsi l'allocation. Mais, la libération(l'insertion) alors prendra beaucoup plus de temps.

VI. Gestion de la MC dans les systèmes multiprogrammés

Stratégies d'allocation

3. **Worst Fit :** Choisir une zone dont la taille est la plus grande parmi toutes les tailles possibles. ce qui implique un parcours complet de la liste.

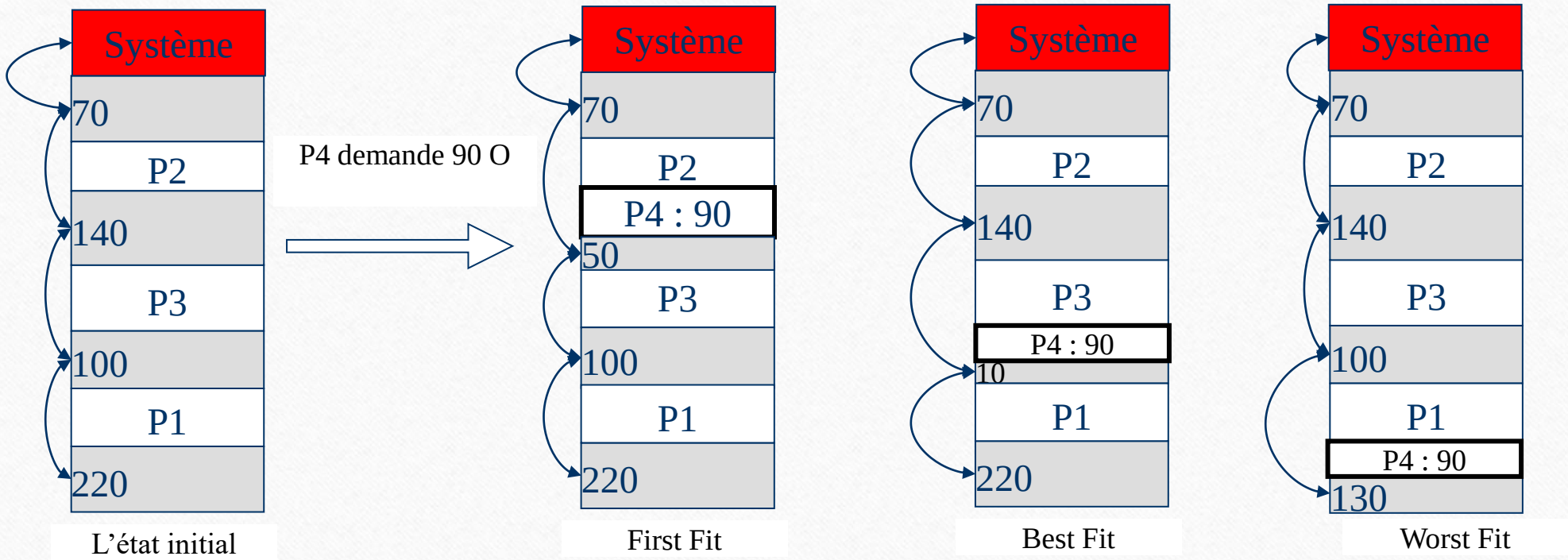
➢ Fragmentation Externe la plus grande possible ➔ possibilité de réutilisation par un autre process.

➢ Recherche lente ⇔ Allocation lente

On a intérêt à Ordonner la liste par ordre décroissant des tailles pour éviter la parcours complet de la liste et accélérer ainsi l'allocation. Mais, la libération (l'insertion) alors prendra beaucoup plus de temps.

VI. Gestion de la MC dans les systèmes multiprogrammés

Stratégies d'allocation



VI. Gestion de la MC dans les systèmes multiprogrammés

Gestion de la mémoire par subdivision (Buddy system)

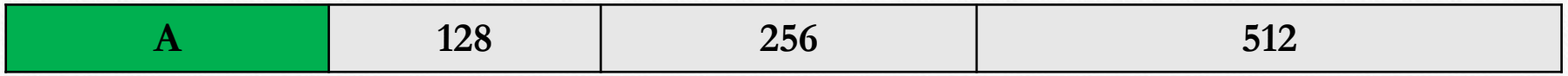
- C'est un compromis entre partitions de tailles fixes et partitions de tailles variables. La mémoire est allouée en unités qui sont des puissances de 2.
- Initialement, il existe une seule unité comprenant toute la MC.
- Lorsque de la mémoire doit être attribuée à un processus, ce dernier reçoit une 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.

VI. Gestion de la MC dans les systèmes multiprogrammés

Gestion de la mémoire par subdivision (Buddy system)

Exemple : Avec une mémoire de 1 Mo, on a ainsi 251 listes. Initialement, la mémoire est vide. toutes les listes sont vides, sauf la liste 1 Mo qui pointe sur la zone libre de 1 Mo :

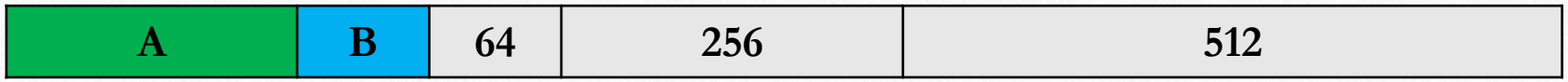
Un processus A demande 70 Ko : la mémoire est fragmentée en deux segments de 512 Ko; l'un d'eux est fragmenté en deux blocs de 256 Ko; l'un de ces deux derniers est fragmenté en deux blocs de 128 Ko et on loge A dans l'un d'eux, puisque $64 < 70 < 128$:



VI. Gestion de la MC dans les systèmes multiprogrammés

Gestion de la mémoire par subdivision (Buddy system)

Un processus B demande 35 Ko : l'un des deux blocs de 128 Ko est fragmenté en deux de 64 Ko et on loge B dans l'un d'eux puisque $32 < 35 < 64$:



Un processus C demande 80 Ko : le bloc de 256 Ko est fragmenté en deux de 128 Ko et on loge C dans l'un d'eux puisque $64 < 80 < 128$:

VI. Gestion de la MC dans les systèmes multiprogrammés

Gestion de la mémoire par subdivision (Buddy system)

A s'achève et libère son bloc de 128 Ko. Puis un processus D demande 60 Ko : le bloc libéré par A est fragmenté en deux de 64 Ko, dont l'un logera D :

B s'achève, permettant la reconstitution d'un bloc de 128 Ko :

D s'achève, permettant la reconstitution d'un bloc de 256 Ko , etc...

VI. Gestion de la MC dans les systèmes multiprogrammés

Gestion de la mémoire par subdivision (Buddy system)

- Allocation et libération très simple
- Fragmentation Interne. Un process demandant $\mathbf{2^n + 1}$ (Ex. 257) aura un bloc de $\mathbf{2^{n+1}}$ (ex. 512)

VI. Gestion de la MC dans les systèmes multiprogrammés

Fragmentation mémoire

- **Fragmentation interne (Internal Fragmentation) :** La mémoire allouée peut être légèrement plus grande que la mémoire requise. Cette différence est appelée fragmentation interne – de la mémoire qui est interne à une partition mais n'est pas utilisée.
- **Fragmentation externe (External Fragmentation) :** La fragmentation externe se présente quand il existe un espace mémoire total suffisant pour satisfaire une requête, mais il n'est pas contigu ; la mémoire est fragmentée en un grand nombre de petits trous (i.e. blocs libres) où un programme ne peut être chargé dans aucun de ces trous.

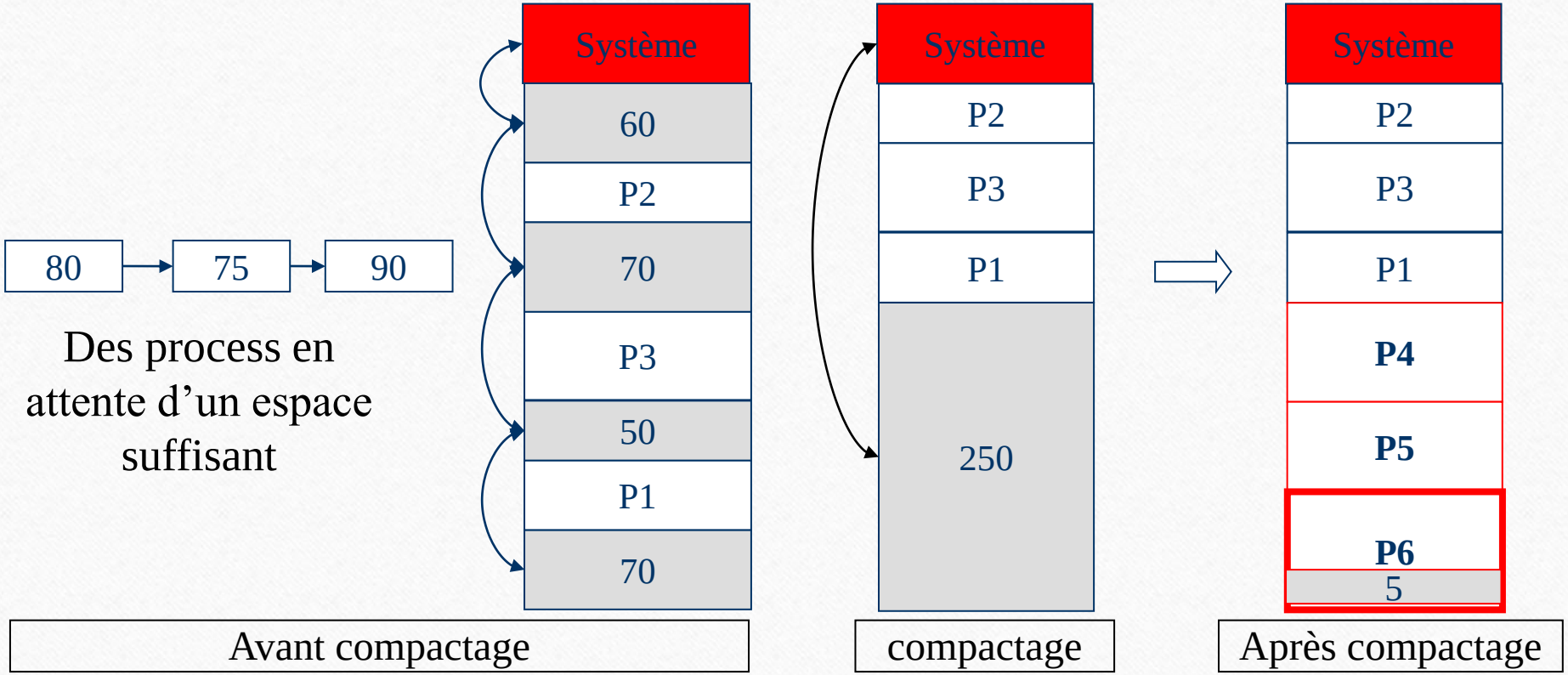
VI. Gestion de la MC dans les systèmes multiprogrammés

Compactage (Compaction)

- Au fur et à mesure de l'allocation et la libération, le nombre de zones libres constituant la fragmentation externes augmente. Il est logique que plus que ce nombre est grand, moins que l'allocation ait une chance de trouver une zone adéquate. ➔ **Les performances diminuent**!
- Cette opération est appelé le compactage de la mémoire centrale (Ramasse miettes ou Garbage Collector).
- Le compactage est une solution pour la fragmentation externe qui permet de regrouper les espaces inutilisés (i.e. les blocs libres) dans une partie de la mémoire (Ex. début, milieu). Cette opération est **très coûteuse en temps CPU**.

VI. Gestion de la MC dans les systèmes multiprogrammés

Compactage (Compaction)



VI. Gestion de la MC dans les systèmes multiprogrammés

Allocation non contiguë (Non Contiguous Allocation)

Les techniques d'allocation précédentes, de l'approche d'allocation contiguë, présentent plusieurs limitations :

1. Au moment de l'allocation, le gestionnaire doit trouver un espace libre contigu suffisant pour loger le programme. Même s'il y a des fragments libres dont la somme est suffisante, l'allocation contiguë refuse de le loger. Plus que la taille d'un programme augmente, moins que les chances de trouver un tel espace
2. La fragmentation externe fréquente due aux allocations et désallocations de l'espace. Le compactage en était la solution, mais ce n'est plus la meilleure !
3. Les programmes dont la taille dépasse la mémoire centrale sont impossible à être exécutés

VI. Gestion de la MC dans les systèmes multiprogrammés

Allocation non contiguë (Non Contiguous Allocation)

- Si les morceaux du programme sont de tailles fixes et égales, on parle de **système de pagination**.
- Si les morceaux du programme sont de tailles variables, on parle alors de **système de segmentation**.
- Ces deux techniques peuvent être combinées, on parle dans ce cas de la technique de **segmentation paginée**.

Système de Pagination de la MC

Principe de la pagination

- Dans le mécanisme de pagination, l'espace d'adressage des programmes est divisé en un ensemble de morceaux de même taille appelés **PAGES**.
- Cet espace est appelé l'espace logique du programme.
- A son tour, l'espace de la mémoire physique (MC) est lui-même découpé en un ensemble de morceaux de taille fixe appelés **CASES** ou cadres de page (Frame Page) (voir figure).

Système de Pagination de la MC

Principe de la pagination

Process Pr1

Pg0
Pg1
Pg2
Pg3
Pg4

Espace logique d'un programme

MC

Case 0
Case1
Case2
Case3
Case4
Case5
Case6
Case7

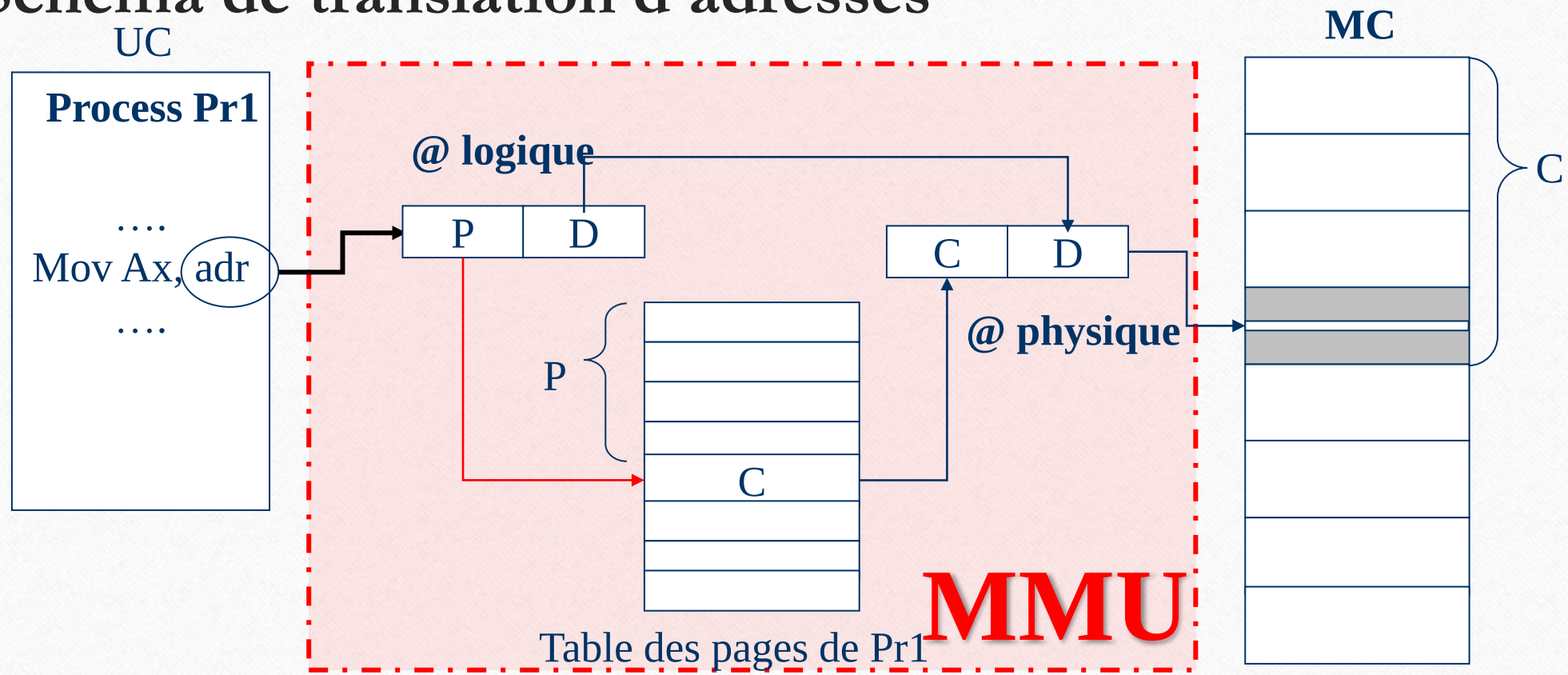
Espace logique vs Espace physique

Système de Pagination de la MC

- La taille d'une case est égale à la taille d'une page.
- Généralement, c'est une puissance de 2, selon l'architecture de l'ordinateur.
- Ainsi, un programme de taille **L** sera divisé en un nombre de '**p**' pages tel que '**p**' est le premier entier vérifiant la relation : **L >= n * taille_page.**
- Au moment du chargement, chaque page d'un programme sera chargée dans une case libre quelconque de la mémoire centrale.
- Dans un système de pagination pure, un programme ne peut être chargé que s'il y a un nombre de cases libres égal au nombre de pages du programme.
- Si la taille L d'un programme n'est pas un multiple de la taille d'une page, alors la dernière page dans l'espace logique du programme ne sera pas entièrement remplie. Un espace vide irrécupérable apparaît constituant une fragmentation interne.

Système de Pagination de la MC

Schéma de translation d'adresses



Système de Pagination de la MC

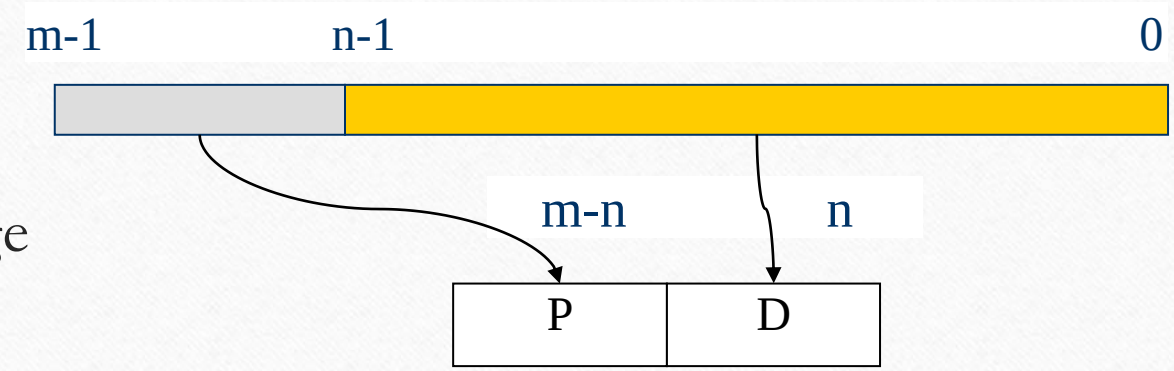
Correspondance Adresses logiques - Adresses physiques

On divise chaque adresse (logique) générée par l'UC en deux parties :

- **Un numéro de page (P, Page Number)** qui est utilisé comme un index dans la table des pages. La table des pages contient l'adresse de base de chaque page dans la mémoire physique.
- **Un déplacement dans la page (D, Page Offset)** qui est combinée à l'adresse de base de la page pour définir l'adresse mémoire physique qui sera envoyée à l'unité mémoire.
- Pour un espace d'adressage logique de 2^m **octets**, en considérant des pages de 2^n **octets**, les **m - n** premiers bits d'une adresse logique correspondent au numéro de page **P** et les **n** bits restants au déplacement **D** dans la page

Système de Pagination de la MC

Correspondance Adresses logiques - Adresses physiques



- Donc, si **T** est la taille d'une page
- et **U** une adresse logique,
- alors l'adresse paginée **(P, D)** est déduite à partir des formules suivantes :
- **P = U Div T** (où, **Div** est la division entière)
- **D = U Mod T** (où, **Mod** est le reste de la division)

Système de Pagination de la MC

Obtenir l'adresse Physique?

- L'adresse physique correspondante à une adresse logique **adr = <P, D>** est obtenue en remplaçant le numéro de page **P** par le numéro de la case **C**, ou en pratique son adresse d'implantation, contenant cette page.
- Le déplacement **D** dans la page étant le même dans la case puisque les deux ont la même taille. Le numéro de la case ou son adresse étant obtenu en indexant la table des page du process actif par la valeur **P**. la case correspondante nous renvoie **C**, ou l'adresse équivalente.
- Au moment de l'exécution, et pour un processeur donné, une seule table des pages qui est active ; c'est celle correspondant au process en cours d'exécution. Chaque opération de commutation de contextes implique le changement de la table des pages active au niveau processeur.

Système de Pagination de la MC

Obtenir l'adresse Physique? Exemple

- Une adresse **AB = 1010 1011** sur **8** bits, **m=8** bits. Nous avons aussi des pages de taille 2^6 mots. Donc le nombre de pages est de $2^{m-n}=2^{8-6}=4$.
- Le déplacement à l'intérieur d'une page est de **n = 6 bits**.
- Alors **l'adresse logique** devienne adresse paginée **<P, D>** comme suit :

- **P= 171 Div 64 = 2, P=2.** → Le numéro de la page est 2
- **D =171 Mod 64 = 43,** → **D=43** est le déplacement (offset) dans la page numéro 2.

1010 1011 → <2 , 43>

Système de Pagination de la MC

Table des pages du Process P1

	Num case
pg0	C5
Pg1	C2
Pg2	C0
Pg3	C3

Table des pages Process P2

	Num case
pg0	C1
Pg1	C6

	MC
	CSys0
	CSys1
C0	Pg2 – P1
C1	Pg0 – P2
C2	Pg1 – P1
C3	Pg3 – P1
C4	
C5	Pg0 – P1
C6	Pg1 – P2
C7	

Modèle de pagination de la mémoire logique et physique

Système de Segmentation de la MC

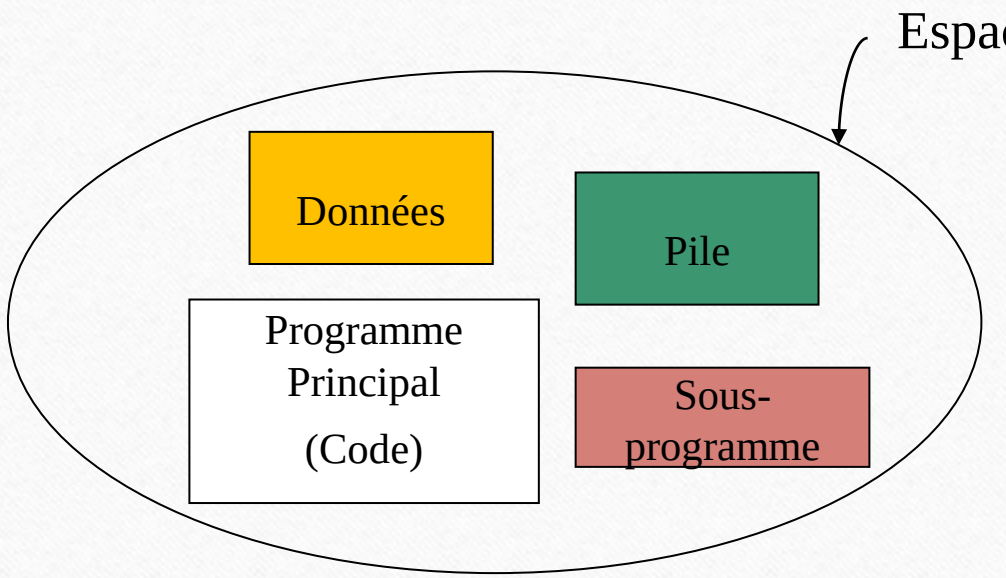
Principe de la segmentation

Le découpage tel qu'il a été présenté dans un système de pagination ne correspond pas à la vision et à la manière de pensée de l'utilisateur. Ce dernier voit un programme comme étant un ensemble d'unités logiques : code, données, pile, programme principal, procédures, une dll, … appelées généralement Segments.

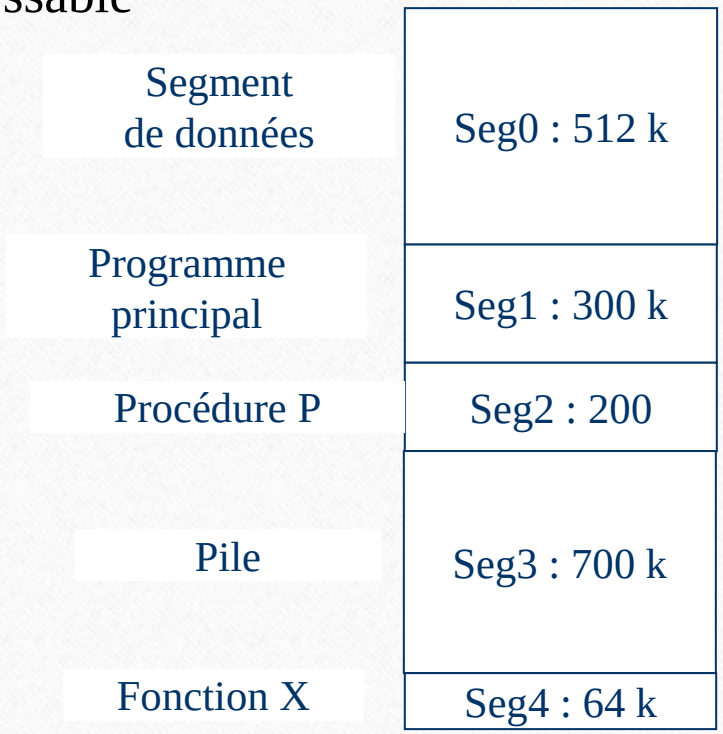
- **La pagination ne projette pas cet aspect de raisonnement.**
- **La MC n'est pas découpé en bloc comme dans le cas de pagination.**

Système de Segmentation de la MC

Principe de la segmentation



Vue de l'utilisateur à son programme



Système de Segmentation de la MC

Principe de la segmentation

- Dans une mémoire segmentée, chaque unité logique d'un programme usager est stockée dans un bloc mémoire, appelé « segment » à l'intérieur duquel les adresses sont relatives au début du segment. Ces segments sont de **tailles différentes**. Un programme sera donc constitué d'un ensemble de segments de code et de données, pouvant être dispersés en MC.
- La segmentation facilite l'édition de liens, ainsi que le partage entre processus de segments de données ou de codes.
- Contrairement au schéma de pagination, la segmentation permet d'éliminer la fragmentation interne car l'espace mémoire alloué à un segment est de taille égale exactement à la taille du segment. Cependant, une fragmentation externe peut se produire due au fait qu'on fait une allocation dynamique de l'espace mémoire.

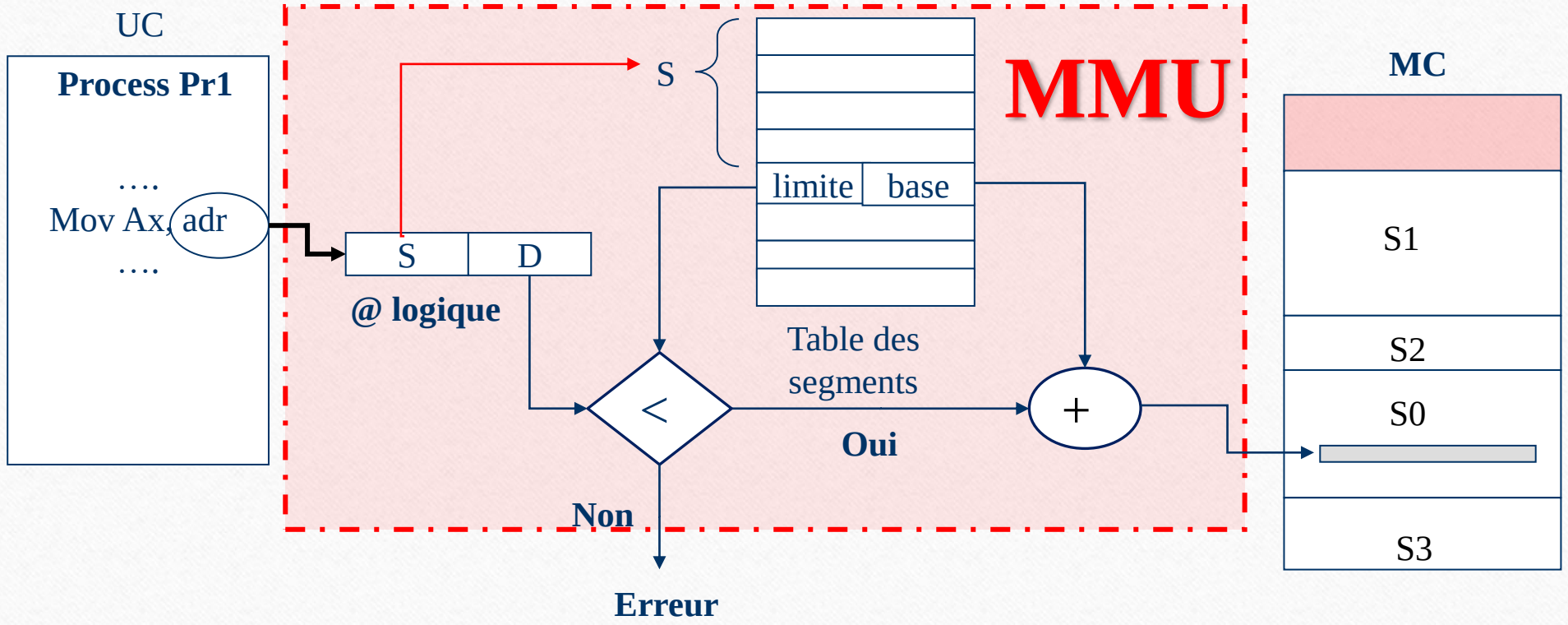
Système de Segmentation de la MC

Schéma de translation d'adresses

- Chaque segment est repéré par son numéro **S** et sa longueur variable **L**. Un segment est un ensemble d'adresses logiques contiguës. Une adresse logique est donnée par un couple **(S, D)**, où **S** est le numéro du segment et **D** le déplacement dans le segment.
- L'association d'une adresse logique à une adresse physique est décrite dans une table appelée **table de segments (Segment Table)**.
- Chaque entrée de la table de segments possède un **segment de base** et un **segment limite**. Le segment de base contient l'adresse physique de début où le segment réside en mémoire, tandis que le registre limite spécifie la longueur du segment.

Système de Segmentation de la MC

Schéma de translation d'adresses



Système de Segmentation de la MC

Correspondance Adresses logiques - Adresses physiques

Dans un système segmenté, une adresse logique référencée par un instruction est de la forme d' un couple **<S, D>** tel que **S** : numéro de segment, et **D** déplacement dans le segment. Le gestionnaire de mémoire doit garder la trace de l'emplacement mémoire dans lequel est hébergé le segment **S**. Pour ce faire il construit pour chaque process une structure, généralement logicielle, appelée table des segments dans laquelle il fait la correspondance entre chaque segment chargée dans la MC et son adresse d'implantation(dans la MC) ainsi que sa taille.

Système de Segmentation de la MC

Correspondance Adresses logiques - Adresses physiques **Exemple :**

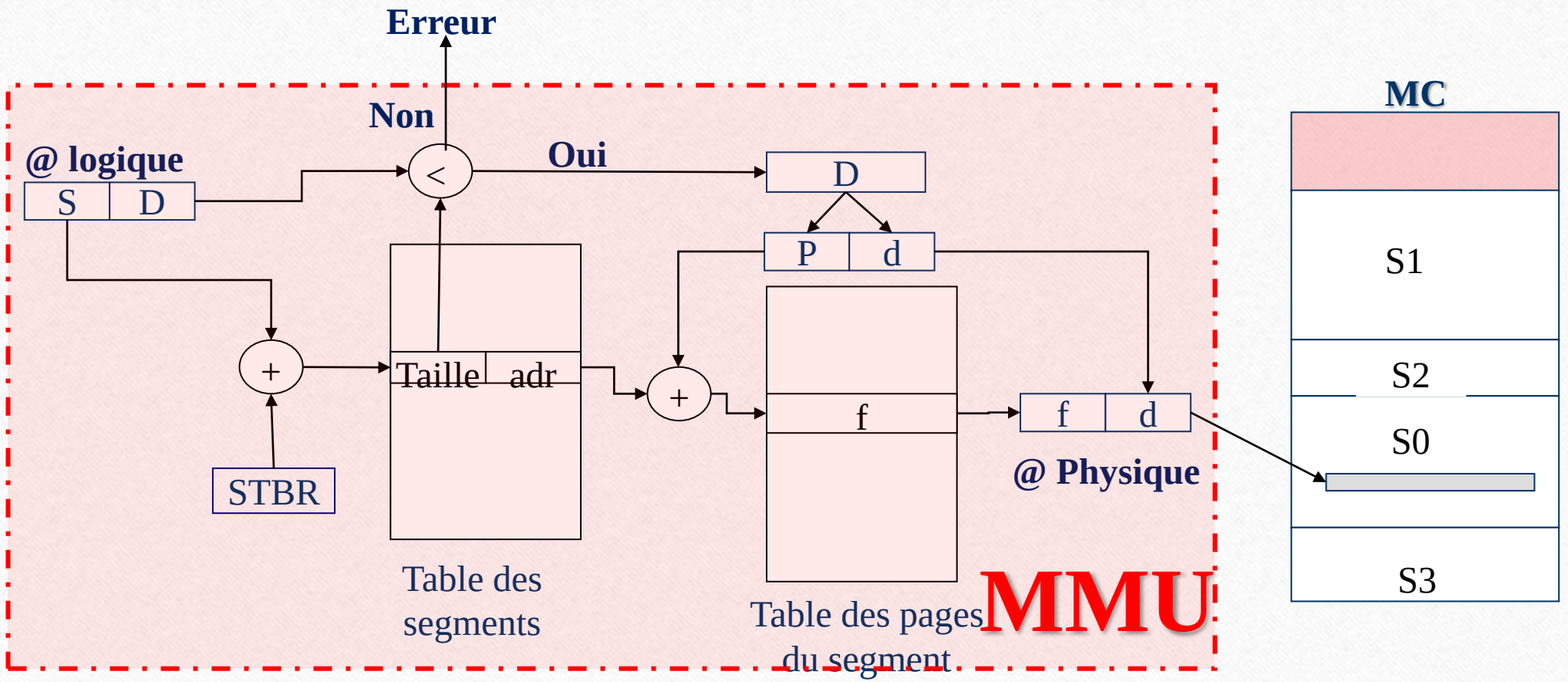
Sur un système utilisant la segmentation simple, calculez l'adresse physique de chacune des adresses logiques, à partir de la table des segments ci-après. On suppose que les adresses soient décimales au lieu de binaires.

Segment	Base	Limite		Adresse logique	Adresse physique
0	1100	500		<0, 300>	300<500, donc @=1100+300=1400
1	2500	1000		<2, 800>	800>600, donc Erreur d'adressage
2	200	600		<1, 600>	600<1000, donc @=2500+600=3100
3	4000	1200		<3, 1100>	1100<1200, donc @=4000+1100=5100
				<1, 1111>	1111>1000, donc Erreur d'adressage

Système de Segmentation paginée

- La taille d'un segment peut être importante, d'où un temps de chargement long qui peut en résulter. La segmentation paginée combinant entre la segmentation et la pagination, ce qui peut être une solution.
- Dans cette technique, les programmes sont alors divisés en segments et chaque segment en pages. Cette technique a été inventée pour le système Multicast.
- A chaque processus, on associe une table de segments et une table de pages. Donc chaque adresse de segment n`est pas une adresse de mémoire, mais une adresse au tableau de pages du segment
- Une adresse logique **(S, D)**, avec **S** numéro de segment et **D** déplacement dans le segment, est transformée en **(S, P, d)**, où **P** est un numéro de page et **d** un déplacement dans la page **P**. chaque adresse devient alors un triplet **(S, P, d)**.

Système de Segmentation paginée



Mémoire virtuelle

Les systèmes de pagination et de segmentation tels qu'on a introduit présentent plusieurs limitations :

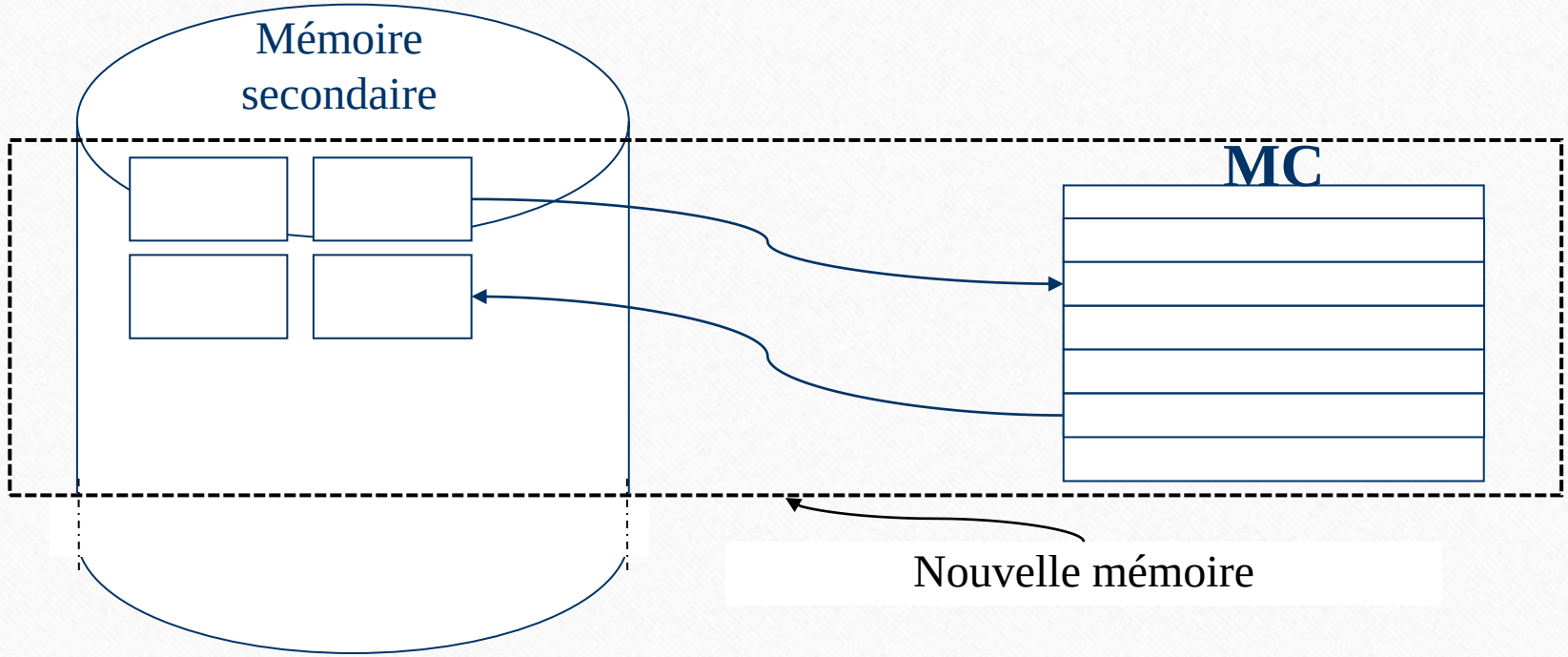
- Un programme ne peut être chargé que s'il y a des cases libres (zones libres) égales au nombre de pages du programme sinon il sera mis en attente. Une conséquence directe de cette limitation est qu'un programme dont la taille est supérieure à la taille de la MC ne pourra jamais s'exécuter !!!!
- Pour un process actif, une seule page (ou segments) est active à un moment donné ; c'est celle depuis laquelle il lit les instructions ou à laquelle il accède pour besoin de données. Mais ce process a préalablement chargé toutes ses pages (ou segments) dans la MC alors qu'il y a des programmes qui attendent d'être chargées.

Mémoire virtuelle

- La mémoire virtuelle est une technique autorisant l'exécution de processus pouvant ne pas être complètement en mémoire. ➔ technique de virtualisation de la MC
- Cela est possible en utilisant une mémoire auxiliaire comme espace de travail pour charger et décharger les différentes pages par le SE.
- Pour ce faire, les programmes au niveau de la Mémoire secondaire (MS) sont divisés en un certain nombre de morceaux dont chacun est de taille inférieure à la taille physique de la MC.
- Ainsi, le SE considère la MS comme étant une extension logique de la MC. Dans la pratique, une partie de la MS uniquement, appelée zone de swap ou MV, est étendue à la MC.

Mémoire virtuelle

- On insiste sur le fait que la MC pour les process utilisateur est celle formée de la MC réelle + la zone swapp. Les échanges entre les deux sont transparents par rapport à l'utilisateur



Mémoire virtuelle

Recouvrement (Overlay)

- Le recouvrement (Overlay) est une technique qui permet de remplacer une partie de la MC par une autre.
- À un instant donné, un programme n'utilise qu'une petite partie du code et des données qui le constituent. Les parties qui ne sont pas utiles en même temps peuvent donc se "**recouvrir**" ; c'est-à-dire, occuper le **même emplacement** en mémoire physique. Cette technique consiste à permettre à l'utilisateur de diviser son programme en segments, appelés **segments de recouvrement (Overlays)** et à charger un segment en mémoire. Le segment 0 s'exécute en premier. Lorsqu'il se termine, il appelle un autre segment de recouvrement qui sera chargé dans le même emplacement mémoire.

Mémoire virtuelle

Pagination à la demande

- Le principe de la mémoire virtuelle est couramment implémenté avec **la pagination à la demande (On-Demand Paging)** ; c'est-à-dire que les pages des processus ne sont chargées en MC que lorsque le processeur demande à y accéder. La pagination à la demande est semblable à un système de pagination avec va-et-vient (swapping).

Comment le processeur puisse distinguer entre les pages qui sont en mémoire, et celles qui sont sur disque afin de détecter leur éventuelle absence ?

Mémoire virtuelle

Pagination à la demande

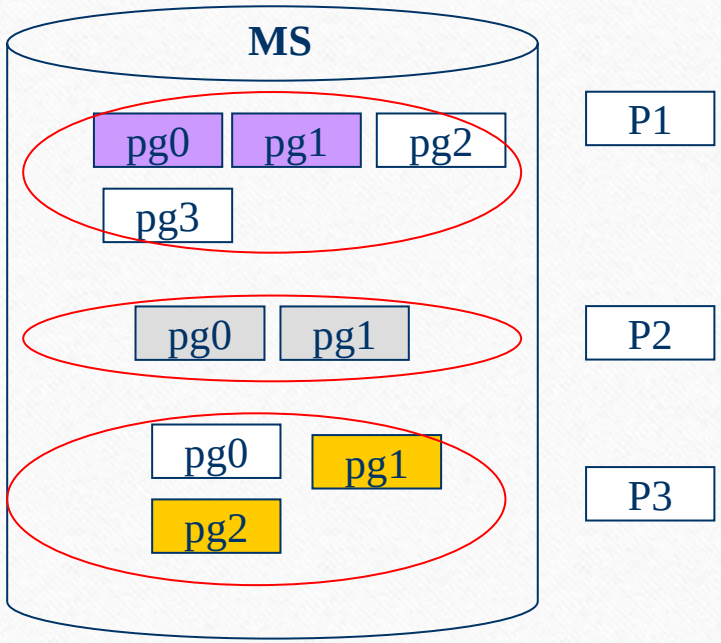
- ***Comment le processeur puisse distinguer entre les pages qui sont en mémoire, et celles qui sont sur disque afin de détecter leur éventuelle absence ?***

- Avec cette technique, le SE dispose de moyens pour distinguer les pages qui sont en mémoire, et celles qui sont sur disque. Il utilise dans la table des pages un bit supplémentaire **V** appelé **bit de validation. (valide/invalide)** pour décrire si la page est chargée en mémoire ou non.

- V = 1 ➔ La page est chargée dans la MC, l'accès à elle est valide
- V = 0 ➔ La page n'est pas chargée dans la MC, l'accès à elle n'est pas valide

Mémoire virtuelle

Pagination à la demande



P1

P2

P3

table des Pages de P1

Pge0	5	1
Pge1	2	1
Pge2	--	0
Pge 3	--	0

table des Pages de P2

Pge0	1	1
Pge1	6	1

table des Pages P3

Pge0	--	0
Pge1	0	1
Pge2	3	1

V

MC

	CSys0
	CSys1
C0	Pg1 – P3
C1	Pg0 – P2
C2	Pg1 – P1
C3	Pg2 – P3
C4	
C5	Pg0 – P1
C6	Pg1 – P2
C7	

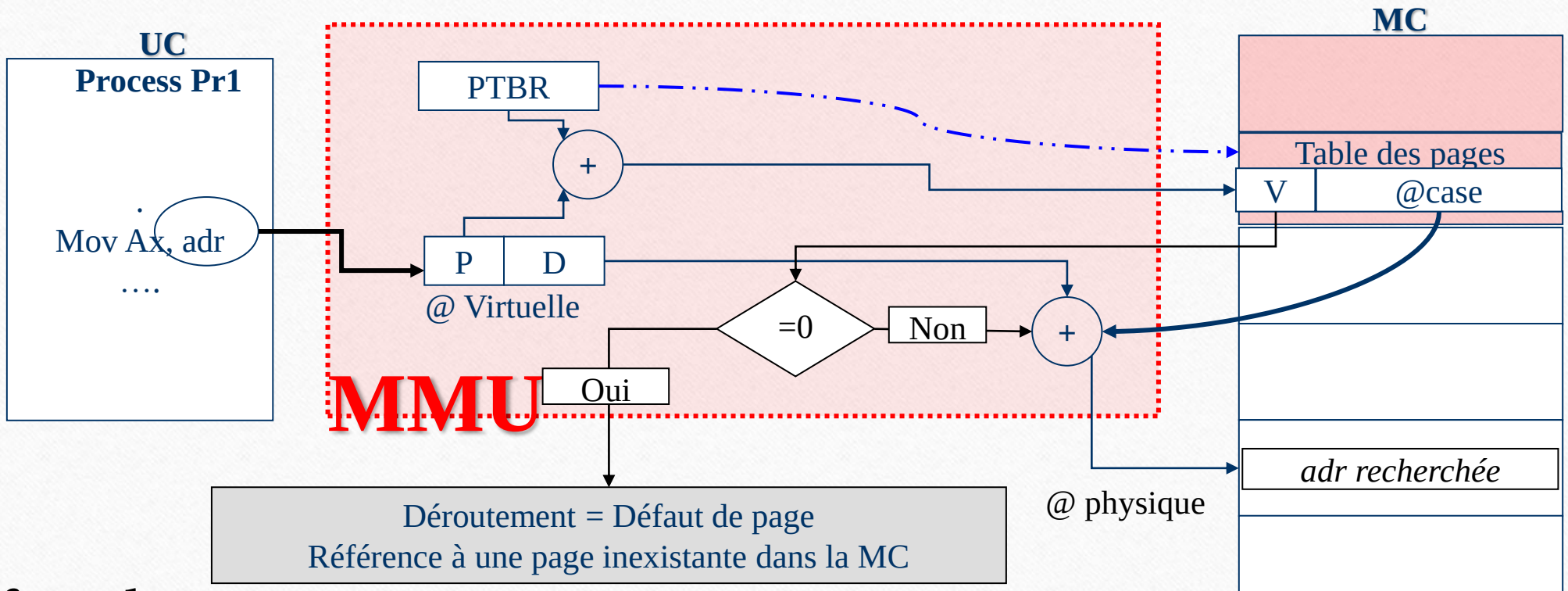
Mémoire virtuelle

Pagination à la demande

- Suite à une référence à une adresse '**adr**' quelconque, cette dernière est convertie par la **MMU** en un couple **<P, D>**.
- La valeur **P** permet d'indexer la table de pages, avant de convertir on doit consulter d'abords le bit **V** associé à elle.
- Si ce bit est à **1** ceci signifie que la page est chargée en MC alors on peut continuer la conversion.
- Sinon, un déroutement est généré par la MMU signalant que le process veut accéder à une page qui n'existe pas dans la MC. Ce déroutement est appelé **défaut de pages** (voir figure).

Mémoire virtuelle

Pagination à la demande



Défaut de page

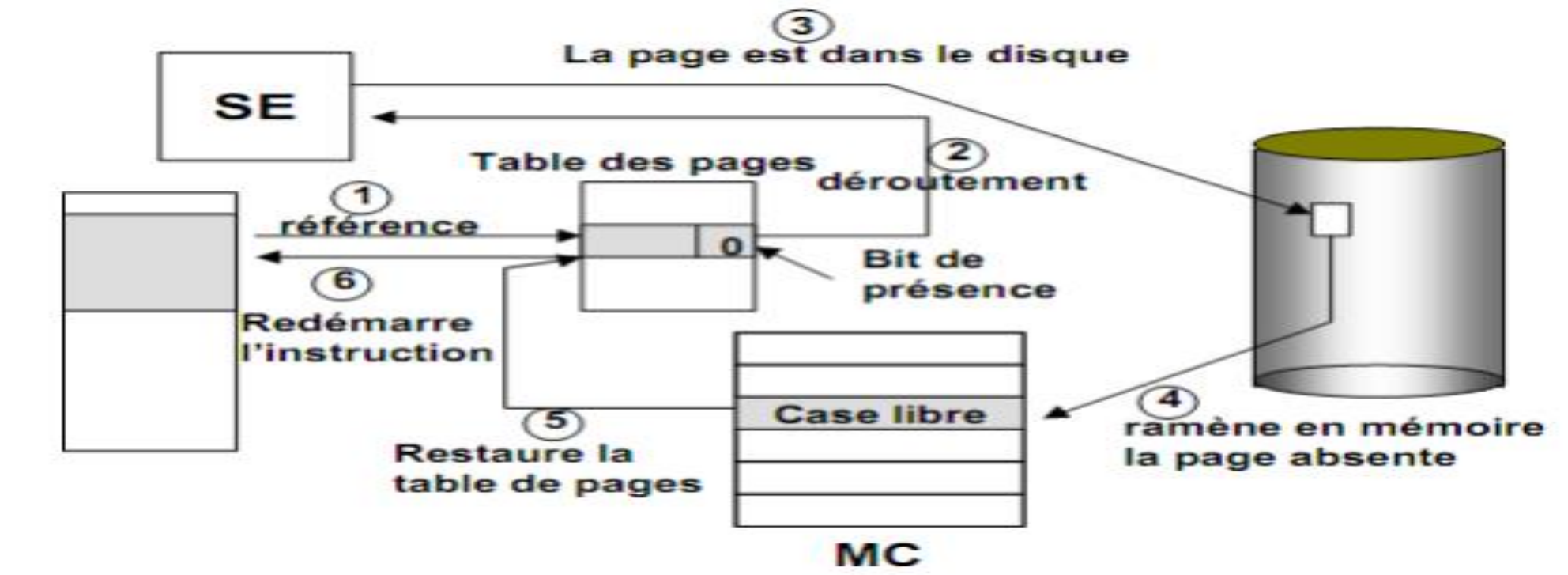
Mémoire virtuelle

Pagination à la demande

- ***Que se passe-t-il si un processus essaie d'utiliser une page qui n'est pas en mémoire ?***
- L'accès à une page marquée invalidé provoque **un défaut de page (Fault Page)** vers le système d'exploitation.. En essayant d'accéder à cette page, il y a un déroutement vers le SE. La procédure permettant de traiter ce défaut de page est la suivante :
  - ❑S'assurer que la référence de la page est correcte.
  - ❑S'assurer que la page désirée est bien en mémoire auxiliaire.
  - ❑Trouver un cadre de page libre et charger la page.

Mémoire virtuelle

Stratégies de remplacement de pages



Procédure de traitement d'un défaut de pages

Mémoire virtuelle

Quelques algorithmes de remplacement de pages

- Il existe plusieurs algorithmes différents de remplacement de pages. En général, on souhaite celui qui provoquera le taux de défauts de pages le plus bas. On évalue un algorithme en l'exécutant sur une séquence particulière de références mémoires et en calculant le nombre de défauts de page.
- A un instant donné, la séquence des numéro de pages référencées par un process depuis son exécution est appelée **chaîne de Références** du process. Le comportement futur d'un programme peut être contrôlé par la connaissance de cette chaîne de référence.
- Afin de déterminer le nombre de défauts de pages pour une chaîne de références et un algorithme de remplacement particulier, on doit également connaître le nombre de cadres de pages disponibles. Evidemment, au fur et à mesure que le nombre de cadres de pages augmente, le nombre de défauts de pages doit diminuer.

Mémoire virtuelle

Exemple : Soit un processus divisé en **5 pages**, la taille d'une page = **100**.

Un Programme **P** fait successivement référence aux adresses suivantes : **100, 210, 355, 120,420, 110, 200, 550, 139, 201, 395, 404, 505.**

La chaîne de Références associée est : **1,2,3,1,4,1,2,5,1,2,3,4,5**.

Quel est le nombre de défauts de pages avec différents algorithmes de remplacement ?

- **FIFO** (First In First Out)
- Optimal
- **LRU** (Least Recently Used)
- **NFU** (Not Frequantly Used)
- **MFU** (Most frequently used)
- Algorithme de la seconde chance

Mémoire virtuelle

Stratégie de remplacement FIFO

Principe : Remplacer l'ancienne Page chargée

Raison : Il est fort probable que l'ancienne page n'est pas fréquemment référencée comme une page récente

Implémentation : Prévoir pour chaque page dans la table des pages un champs (date+temps) initialisé à l'heure de chargement.

Exemple : mémoire de **3 cases**

Demandes	1	2	3	1	4	1	2	5	1	2	3	4	5
Case 1	**1**	1	1	1	**4**	4	4	**5**	5	5	5	5	5
Case 2		**2**	2	2	2	**1**	1	1	1	1	**3**	3	3
Case 3			**3**	3	3	3	**2**	2	2	2	2	**4**	4
Défaut de page	**D**	**D**	**D**		**D**	**D**	**D**	**D**			**D**	**D**	

Mémoire virtuelle

Stratégie de remplacement FIFO

On peut se dire que si l'on augmente le nombre de cases le nombre de défaut de pages diminuera !

- Nous ajoutons une case. Et on va voir :

Demandes	1	2	3	1	4	1	2	5	1	2	3	4	5
Case 1	**1**	1	1	1	1	1	1	**5**	5	5	5	**4**	4
Case 2		**2**	2	2	2	2	2	2	**1**	1	1	1	**5**
Case 3			**3**	3	3	3	3	3	3	**2**	2	2	2
Case 4					**4**	4	4	4	4	4	**3**	3	3
Défaut de page	**D**	**D**	**D**		**D**			**D**	**D**	**D**	**D**	**D**	**D**

- Le résultat est l'augmentation du nombre de défauts de pages à 10.
- Ce phénomène est appelé **l'anomalie de Belady.**

Mémoire virtuelle

Stratégie de remplacement optimal

Principe : la page victime sera la page qui ne sera pas référencée dans le futur immédiat.

Raison : c'est la case la plus adéquate, car toutes les autres vont être référencée avant elle.

Implémentation : Comment peut-on savoir qu'une page sera référencée dans le future?

- Algorithme non implémentable, il est utilisé pour référence de comparaison des autres algorithmes.

Mémoire virtuelle

Stratégie de remplacement optimal

Exemple : mémoire de 4 cases

Demandes	1	2	3	1	4	1	2	5	1	2	3	4	5
Case 1	**1**	1	1	1	1	1	1	1	1	1	1	**4**	4
Case 2		**2**	2	2	2	2	2	2	2	2	2	2	2
Case 3			**3**	3	3	3	3	3	3	3	3	3	3
Case 4					**4**	4	4	**5**	5	5	5	5	5
Défaut de page	**D**	**D**	**D**		**D**			**D**				**D**	

- **6 défauts de pages pour une mémoire de 4 cases !!**

Mémoire virtuelle

Stratégie de La page la moins récemment utilisée (LRU : Least Recently Used)

Principe : la page victime sera la page la moins référencée dans un passé proche.

Raison : Principe de localité spatiale et localité temporelle favorise qu'une page récemment utilisée a une grande chance d'être référencée dans un future très proche. Donc la page victime devrait être une page anciennement référencée.

Implémentation : un champ pour chaque entrée dans la table des pages indiquant le temps de la dernière référence à la page. A Chaque référence (en écriture ou e lecture) à la page, il sera mis à jour.

- La page victime est celle ayant le plus ancien temps de référence.
- C'est un algorithme un petit peu lourd à implémenter. Utilisé pratiquement mais avec des versions un petit peu légère.

Mémoire virtuelle

Stratégie de La page la moins récemment utilisée (LRU : Least Recently Used)

Exemple : mémoire de 4 cases

Demandes	1	2	3	1	4	1	2	5	1	2	3	4	5
Case 1	**1**	1	1	1	1	1	1	1	1	1	1	1	**5**
Case 2		**2**	2	2	2	2	2	2	2	2	2	2	2
Case 3			**3**	3	3	3	3	**5**	5	5	5	**4**	4
Case 4					**4**	4	4	4	4	4	**3**	3	3
Défaut de page	**D**	**D**	**D**		**D**			**D**			**D**	**D**	**D**

➢ **8 défauts de pages pour une mémoire de 4 cases !!**

Mémoire virtuelle

Stratégie de NFU : not frequently used (Least Frequently Used)

Principe : la page victime est la page qui n'est pas fréquemment utilisée

Raison : c'est la page la moins utilisée dans le passé proche parmi les autres, donc on peut se dire qu'elle a une faible probabilité qu'elle soit référencée dans le future proche.

Implémentation : un champ pour chaque entrée dans la table des pages indiquant le nombre de fois que cette page a été référencée.

- La page victime est celle ayant la plus petite valeur.

Mémoire virtuelle

Stratégie de NFU : not frequently used (Least Frequently Used)

Exemple : mémoire de 4 cases

Demandes	1	2	3	1	4	1	2	5	1	2	3	4	5
Case 1	**1**	1	1	1	1	1	1	1	1	1	1	1	1
Case 2		**2**	2	2	2	2	2	2	2	2	2	2	2
Case 3			**3**	3	3	3	3	**5**	5	5	5	**4**	4
Case 4					**4**	4	4	4	4	4	**3**	3	**5**
Défaut de page	**D**	**D**	**D**		**D**			**D**			**D**	**D**	**D**

➢ 8 défauts de pages pour une mémoire de 4 cases !!

Mémoire virtuelle

Stratégie de MFU : Most frequently used

Principe : la page victime est la page qui est fréquemment utilisée

Raison : une page qui n'est pas fréquemment utilisée, possible qu'elle le soit dans le proche future.

Implémentation : un champ pour chaque entrée dans la table des pages indiquant le nombre de fois que cette page a été référencée.

- La page victime est celle ayant la plus grande valeur.

Mémoire virtuelle

Stratégie de MFU : Most frequently used

Exemple : mémoire de 4 cases :

Demandes	1	2	3	1	4	1	2	5	1	2	3	4	5
Case 1	**1**	1	1	1	1	1	1	**5**	5	5	5	**4**	4
Case 2		**2**	2	2	2	2	2	2	**1**	1	1	1	**5**
Case 3			**3**	3	3	3	3	3	3	**2**	2	2	2
Case 4					**4**	4	4	4	4	4	**3**	3	3
Défaut de page	**D**	**D**	**D**		**D**			**D**	**D**	**D**	**D**	**D**	**D**

- 10 défauts de pages pour une mémoire de 4 cases !!

Mémoire virtuelle

Stratégie de La Seconde chance

Principe : c'est l'algorithme FIFO avec les modifications suivantes : une page victime choisie, si elle est référencée on lui donne une seconde chance et on cherche une autre.

Raison : améliorer FIFO

Implémentation : de plus celle de FiFO on ajoute un bit Référence qui sera mis à 1 à chaque référence à la page. Une fois la page a été sélectionnée comme victime : si ce bits est à zéro, la page sera la victime sinon, le bit est remis à 0 et on cherche une autre page plus ancienne.

- C'est une approximation du LRU

Mémoire virtuelle

Stratégie de La Seconde chance

Exemple : mémoire de 4 cases :

Demandes	1	2	3	1	4	1	2	5	1	2	3	4	5
Case 1	$\underline{1}_1^+$	1_1^+	1_1^+	1_1^+	1_1^+	1_1^+	1_1^+	$\underline{5}_1$	5_1	5_1	5_1^+	$\underline{4}_1$	4_1
Case 2		$\underline{2}_1$	2_1	2_1	2_1	2_1	2_1	2_0^+	$\underline{1}_1$	1_1	1_1	1_0^+	$\underline{5}_1$
Case 3			$\underline{3}_1$	3_1	3_1	3_1	3_1	3_0	3_0^+	$\underline{2}_1$	2_1	2_0	2_0^+
Case 4					$\underline{4}_1$	4_1	4_1	4_0	4_0	4_0^+	$\underline{3}_1$	3_0	3^0
Défaut de page	**D**	**D**	**D**		**D**			**D**	**D**	**D**	**D**	**D**	**D**

- 10 défauts de pages pour une mémoire de 4 cases !!

Mémoire virtuelle

Stratégie de La Seconde chance

Exemple : mémoire de 4 cases :

Demandes	1	2	3	1	4	5	2	5	1	2	3	4	5
Case 1	$\underline{1}_1^+$	1_1^+	1_1^+	1_1^+	1_1^+	$\underline{5}_1$	5_1	5_1	5_1	5_1	5_1^+	$\underline{4}^1$	4_1
Case 2		$\underline{2}_1$	2_1	2_1	2_1	2_0^+	2_1^+	2_1^+	2_0	2_1	2_1	2_0^+	$\underline{5}_1$
Case 3			$\underline{3}_1$	3_1	3_1	3_0	3_0	3_0	$\underline{1}_1$	1_1	1_1	1_0	1_0^+
Case 4					$\underline{4}_1$	4_0	4_0	4_0	4_0^+	4_0^+	$\underline{3}_1$	3_0	3_0
Défaut de page	**D**	**D**	**D**		**D**	**D**			**D**		**D**	**D**	**D**

- 9 défauts de pages pour une mémoire de 4 cases !!

Mémoire virtuelle

Stratégie de La Seconde chance

Exemple : mémoire de 4 cases :

Demandes	1	2	3	4	1	2	5	2	3	5	2	1	4
Case 1	$\underline{1}_1^+$	1_1^+	1_1^+	1_1^+	1_1^+	1_1^+	$\underline{5}_1$	5_1	5_1	5_1	5_1	5_1^+	5_0
Case 2		$\underline{2}_1$	2_1	2_1	2_1	2_1	2_0^+	2_1^+	2_1^+	2_1^+	2_1^+	2_0	$\underline{4}_1$
Case 3			$\underline{3}_1$	3_1	3_1	3_1	3_0	3_0	3_1	3_1	3_1	3_0	3_0^+
Case 4				$\underline{4}_1$	4_1	4_1	4_0	4_0	4_0	4_0	4_0	$\underline{1}_1$	1_1
Défaut de page	**D**	**D**	**D**	**D**			**D**					**D**	**D**

- 7 défauts de pages pour une mémoire de 4 cases !!

Notion d'Ecroulement

Un process s'écroule lorsqu'il passe plus de temps à paginer qu'à s'exécuter. Ceci est dû au fait que le process n'a pas assez de pages dans la MC et que l'allocation était globale.

Solution de l'Ecroulement : notion de working Set

Un working set**, Δ,** du process **P** est un entier définissant le nombre minimum de pages nécessaire que **P** doit avoir dans la MC pour commencer son exécution. Ce nombre doit être maintenu durant toute l'exécution du process. Autrement dit, si l'une des pages de **P** est choisie comme victime, ce choix sera accepté par **P** si le nombre de ses pages chargées dans la MC est strictement supérieur à Δ sinon il sera refusé.

Solution de l'Ecroulement : notion de working Set

En fait, le **Working set** est un compromis établi entre le nombre de défaut de pages toléré par le système et le nombre de process pouvant être simultanément dans la MC.

Questions ?

Questions ?

Q1) Quels sont les deux dispositifs matériels qui permettent au système d'exploitation la protection des processus en exécution ? les décrire brièvement.

Réponse :

- L'unité de gestion de la mémoire (MMU) empêche les processus d'écrire n'importe où dans la mémoire.
- Les modes kernel/user empêchent les processus d'exécuter certaines instructions

Questions ?

Q2) Dire quelle est la différence principale entre les termes de chacune des paires suivantes :

a) Une segmentation et une pagination

Réponse :

- Partitionnement de l'espace d'adressage logique/physique ;
- La segmentation considère la mémoire comme des espaces, ou des régions, dédiés à une utilisation particulière par exemple : le code d'un programme, les données, la pile, un ensemble de sous-programmes, des modules, un tableau, etc. La segmentation reflète cette organisation.
- La pagination est un partitionnement qui exige les mêmes tailles de pages et de cases.

Questions ?

Q2) Dire quelle est la différence principale entre les termes de chacune des paires suivantes :

b) Une fragmentation interne et une fragmentation externe

Réponse :

Suite aux allocations mémoire des fragments très petits ne seront jamais suffisant pour contenir un processus (fragmentation Interne) alors que la fragmentation externe apparaît suite à une demande d'espace contigüe qui n'est pas disponible bien la somme des fragments séparés dépasse la taille demandée.

Questions ?

Q2) Dire quelle est la différence principale entre les termes de chacune des paires suivantes :

c) Un ordonnancement préemptif et un ordonnancement non préemptif

Réponse :

Si la CPU est allouée à un processus, elle peut lui être réquisitionnée à tout moment avant sa fin alors que l'ordonnancement non préemptif garde le processeur alloué au processus du début à sa fin.

Questions ?

Q3) Dans un système de gestion mémoire virtuelle à pagination

a) Quand est-ce qu'un défaut de page se produit

Réponse :

Si le bit de présence = 0 dans la table des pages ; c.-à-d. que la page demandée est absente en mémoire.

Questions ?

Q3) Dans un système de gestion mémoire virtuelle à pagination

b) Dans quel cas est-il nécessaire de réécrire la page sur l'espace de swap

Réponse :

Si le bit de modification = 1 dans la table des pages ; c.-à-d. que la page demandée a été modifiée depuis son chargement.

Questions ?

Q3) Dans un système de gestion mémoire virtuelle à pagination

c) En déduire son coût en termes entrées/sorties disque

Réponse :

Meilleur cas 1 E/S et au pire cas 2 E/S.

Questions ?

Q4) Quels sont les inconvénients de l'algorithme de remplacement MFU ?

Réponse :

L'algorithme de remplacement MFU exige qu'on remplace la page la plus fréquemment utilisée. Cette méthode peut être inappropriée dans le cas par exemple où une page est très sollicitée par un processus (traitement répétitif dans une boucle sur cette page). Comme elle a un grand compteur d'utilisation, cette page sera choisie à chaque fois comme victime, ce qui provoquera juste après un défaut de page.

Questions ?

Q5) Pourquoi a-t-on intérêt à diminuer le nombre de défauts de pages ?

Réponse :

Les défauts de pages sont « indésirables » car ils provoquent toujours : l'interruption du processus qui les a générés, la prise en charge du déroutement causé par le système d'exploitation, et le chargement à partir du disque (mémoire très lente par rapport à la mémoire centrale) des données demandées.