Systèmes d'Exploitation & Programmation Concurrente

Chap. IV. La Gestion de la Mémoire Centrale

Faïza NAJJAR

ENSI – SRSI (Bureau 109)

Email : Faiza.Najjar@ensi-uma.tn

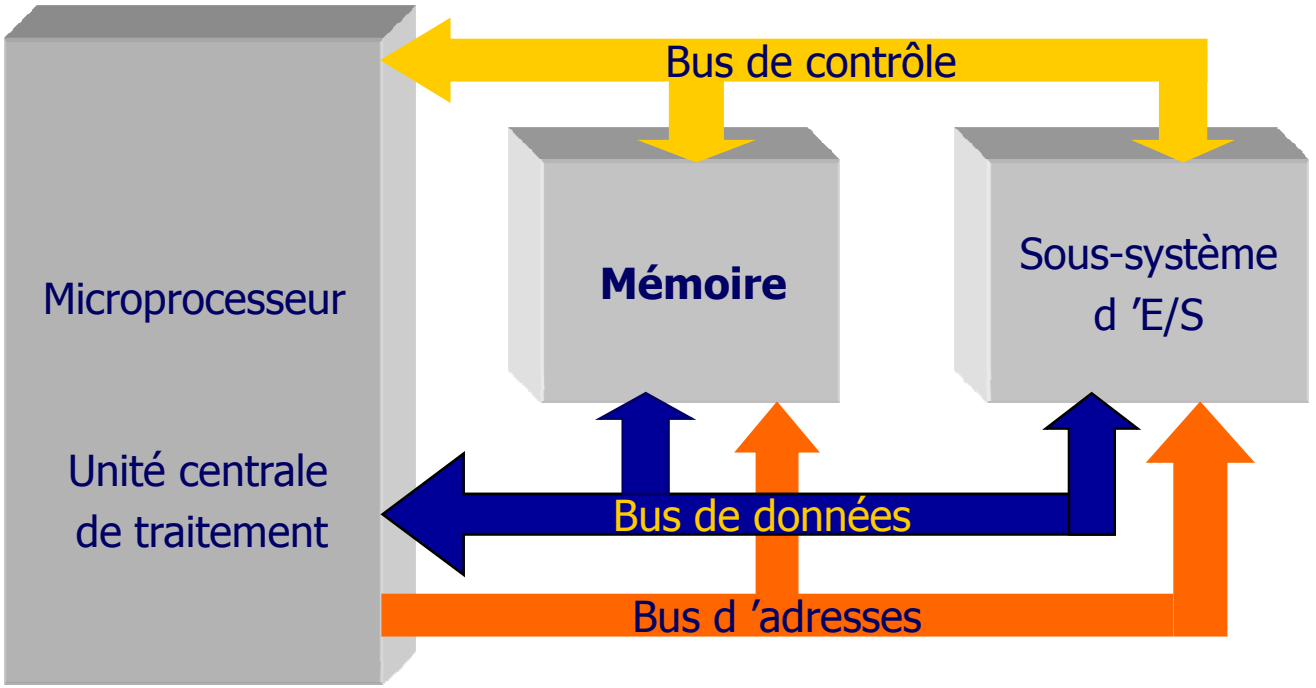
Partie I. La Mémoire Physique

Contenu

1. Introduction
   - ⇨ Notion de multiprogrammation
   - ⇨ Protection des processus
2. Concepts fondamentaux
   - ⇨ Production d'un programme exécutable
   - ⇨ Principe de gestion
3. Allocation contiguë
   - ⇨ Statique/dynamique
   - ⇨ Politiques d'allocation
     - ⇨ Algorithmes de placement 1 : First Fit, Best Fit et Worst Fit
     - ⇨ Algorithmes de placement 2 : Buddy System
4. La segmentation
5. La pagination
6. Le va-et-vient

Introduction

✓ *Structure simplifiée d'un micro-ordinateur*

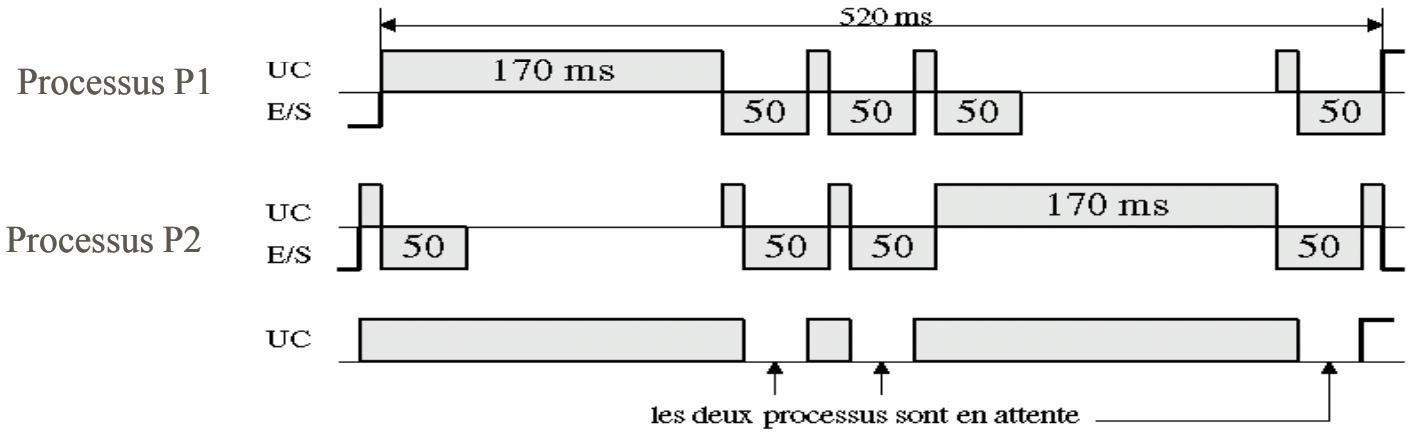


Introduction (suite)

- ✓ **Mémoire centrale :**
  - ⇨ ***Rôle :*** toutes les informations transitent par la MC
    - ↳ Pour qu'un programme puisse être exécuté, il doit être place en MC (Code, données..)
  - ⇨ ***Caractéristiques de la MC*** :
    - ↳ Type de mémoire : RAM/ROM (volatile/non volatile)
    - ↳ Méthode d'accès : direct (temps d'accès constant)
    - ↳ Mémoire à contenu adressable
    - ↳ Format des mots et des adresses
      - → Le processeur accède aux de la MC par le biais de **2 Registres : Adresse, Données** (lecture/Ecriture)
    - ↳ Capacité : nombre total d'octets
- ✓ **Nécessite d'une gestion optimale de la MC: fondamentale**
  - ⇨ En dépit de sa grande disponibilité, elle n 'est, en général, jamais suffisante. Ceci en raison de la taille continuellement grandissante des programmes

Notion de multiprogrammation (Rappel)

- ✓ Multiprogrammation: technique permettant d'optimiser le taux d'utilisation du processeur en réduisant notamment les attentes sur les E/S.
  - ⇨ Activité du processeur : 400 ms sur 520 ms, soit 77%



- ☞ Conséquences de multiprogrammation et gestion mémoire :
  - ↳ ***Définir un espace indépendant pour chaque processus***
  - ↳ ***Protéger les espaces d'adressages des processus entre eux***
  - ↳ ***Allouer de la mémoire physique à chaque espace d'adressage***

Protection des Processus

- ✓ La gestion de la mémoire est presque impossible sans l'aide du matériel → assurer la **protection**
- ✓ Dans la MC, on a 2 types de processus (système, utilisateur)
  - ⇨ Différentes fonctionnalités/droits
  - ⇨ **Problème:** coexistence de processus de nature différentes
  - ⇨ **Solution:** interdire à un utilisateur d'accéder n'importe comment au noyau du système, ou aux autres programmes des utilisateurs

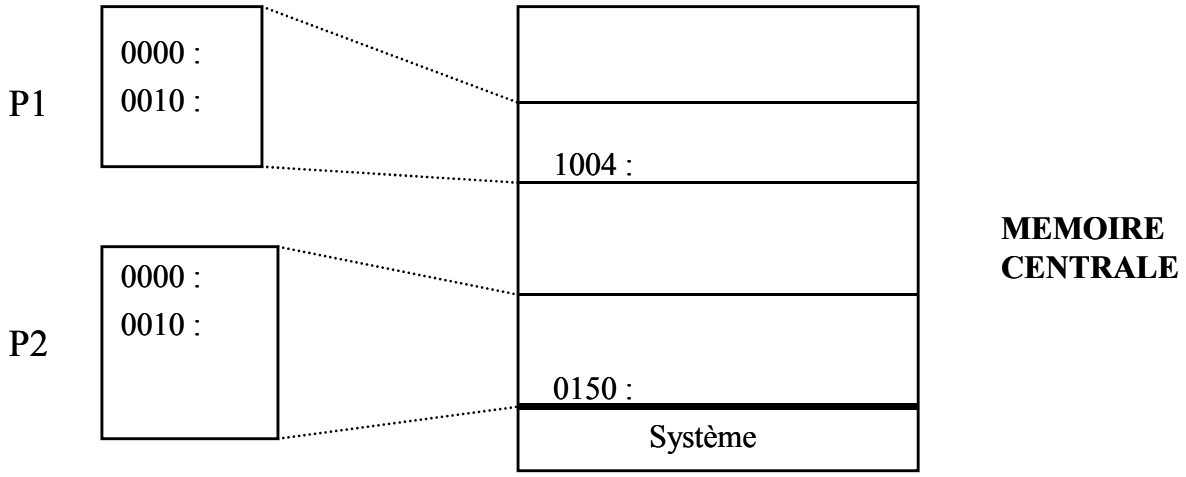
Protection des Processus (suite)

- ✓ **Protection fondamentale**: général. **2 registres** pour chaque processeur
  1. **BASE:** contient
     - la 1ère @ mémoire physique disponible au système
     - l'adresse début de l'espace adressage utilisateur
     - la base (c-à-d base 2 binaire) pour déterminer la taille de page
     - taille min. de mémoire physique à affecter à chaque processus
  2. **LIMITE:** contient l'adresse fin de la zone utilisateur
- ✓ Comparer les adresses émises par le processus à ces 2 registres
  - ⇨ **Si** LIMITE < l'adresse référencée < BASE

    Alors génération d'une It (violation de protection mémoire).

Concepts Fondamentaux (Rappel)
Production d'un programme exécutable

- ✓ Avant d'être exécuté un programme doit passer par plusieurs étapes :
  - ⇨ *Edition du programme* dans un langage source (par exemple C)
  - ⇨ *Compilation* : transformation en un module objet (langage binaire)
    - Le code produit est, en général, relogeable, commençant à l 'adresse 0000 et pouvant se translater à n'importe quel endroit de la mémoire
    - Référence initiale = le registre de base; les adresses représentent alors le décalage par rapport à ce registre.
  - ⇨ *Edition de liens* : rassembler les modules objets
  - ⇨ *Chargement (statique/dynamique)*: modifier le code exécutable pour effectuer les liaisons nécessaires (appels système avec le noyau) et puis chargement en mémoire



Principe de Gestion

- ✓ Deux grandes familles de méthodes d'allocation mémoire :
  - ⇨ **Allocation contiguë:**
    - ↳ Programme = ensemble de mots contigus insécables --**espace d'adressage linéaire**
    - ↳ Allocation en partitions fixes ou variables
  - ⇨ **Allocation non contiguë:**
    - ↳ Programme = ensemble de mots sécables, c- à-d le programme peut être divisé en plus petits morceaux, chaque morceau étant lui même un ensemble de mots contigus
    - ↳ Chaque morceau peut alors être alloué de manière indépendante → Mécanismes de **Segmentation** et de **Pagination**

Principe de Gestion (suite)

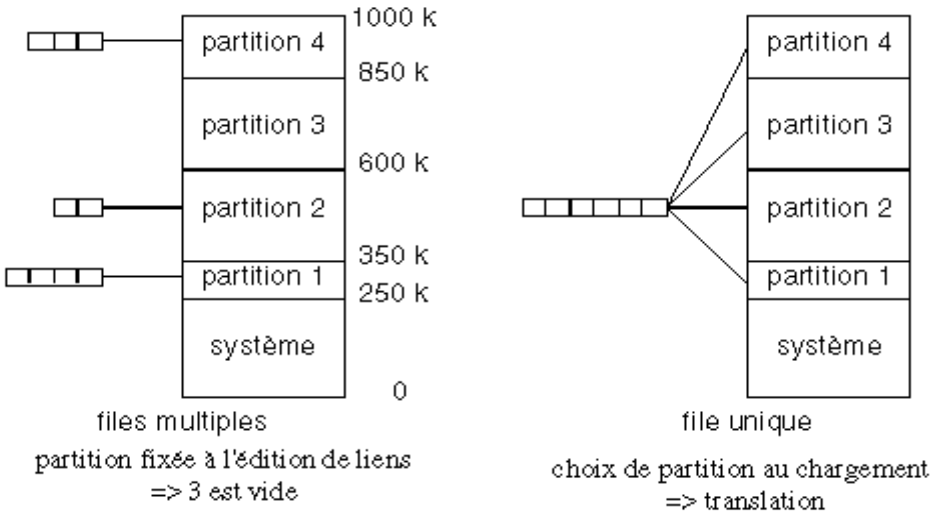
- ✓ Dans beaucoup de cas, il n 'est pas possible de faire tenir tous les processus ensemble en mémoire. Parfois même, la taille d'un programme est trop importante.
  - ⇨ Mettre en œuvre une stratégie de recouvrement (overlay)
    - ↳ Restructurer le programme sous forme arborescente → très fastidieux
  - ⇨ Le SE s'en occupe et décharge le programmeur de cette tache en mettant en œuvre:
    - ↳ le va-et-vient (manque de mémoire pour héberger plusieurs programmes)
    - ↳ la mémoire virtuelle (taille d'un prog. dépasse la taille de la mémoire)

Allocation Contiguë

- ✓ La mémoire physique est découpée en zones disjointes ➪ Nécessité de connaître l'état de la mémoire (zones libres/occupées)
- ✓ Disposer d'une stratégie d'allocation et enfin de procédures de libération
- ✓ 2 types d'allocation :
  - ➪ *Statique* : la MC est découpée en zones fixes -- *partitions* (nombre fixe)
  - ➪ *Dynamique* : la MC est constituée de zones fixes/variables -- *régions* (nombre variable).

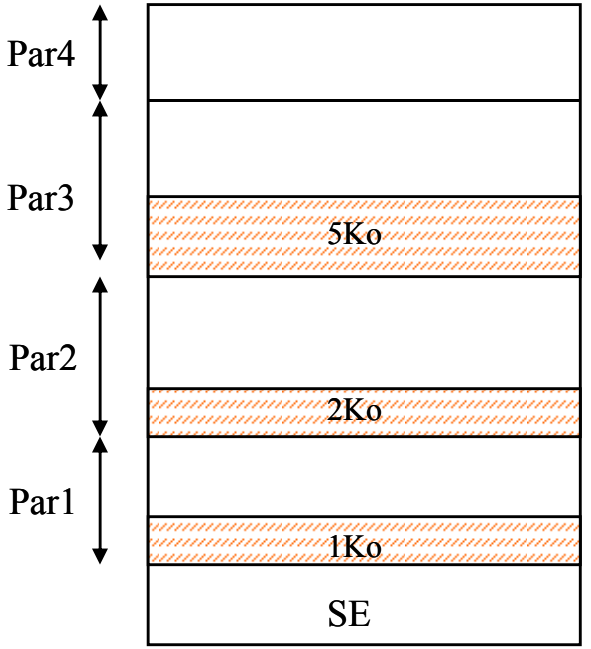
Allocation Statique

- ✓ Un programme est considéré comme un espace d'adresses insécable; Les partitions sont de taille fixe, non nécessairement identiques;
  - ➪ Allocation d'un seul tenant ➪ une partition ne peut contenir qu'un seul processus.



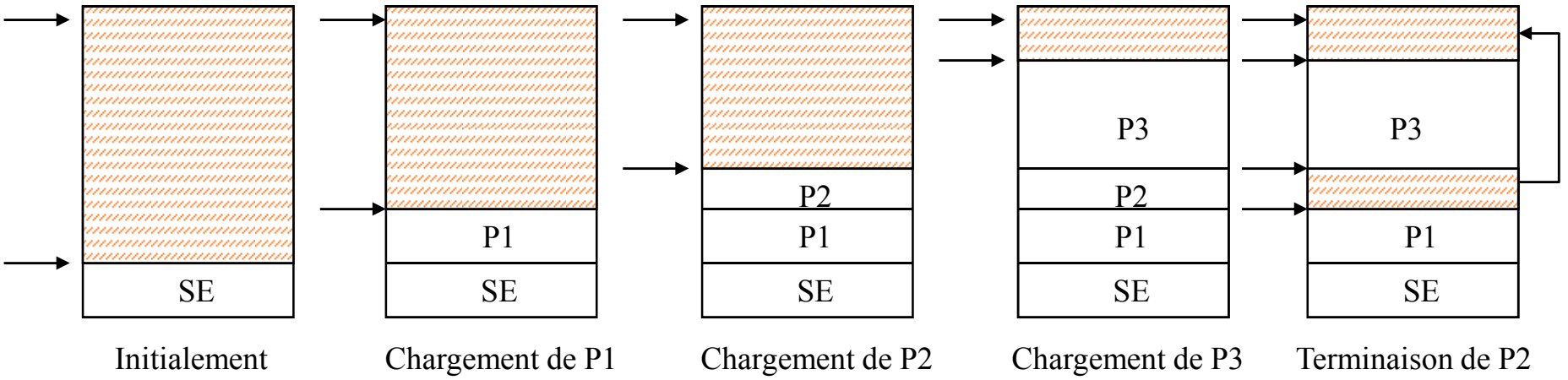
Allocation Contiguë --Allocation Statique

- Etant données la taille(partition) et la taille(processus), 3 cas se présentent:
  - **Cas 1:** taille(partition) = taille(processus) Aucune perte de mémoire
  - **Cas 2:** taille(partition) > taille(processus) → *fragmentation interne* -- perte de mémoire
  - **Cas 3 :** taille(partition) < taille(processus) → *fragmentation externe*
- **Inconvénient :** *Fragmentation -- phénomène de gruyère*
  - Charger un processus de taille 6 Ko?
  - Récupérer les fragments et faire un *tassement* (bas/haut)
    - *Garbage Collector*
  - Création d'une partition poubelle → Coûteuse
    - Translation des adresses dynamiques
  - Les partition restent fixes jusqu'au prochain tassement
- **Avantages :**
  - Protection assez simple à réaliser
  - Adaptée aux systèmes temps réel
    - partitionnement fixe car les programmes sont toujours les mêmes



Allocation Contiguë --Allocation dynamique de la Mémoire

- ✓ Allocation en fonction de la taille du programme à exécuter
  - ⇨ le nombre et la taille des zones varient dynamiquement
  - ⇨ Il faut prévoir des protections entre processus car ils fonctionnent dans le même espace, et tout débordement de l'un risquerait de perturber le fonctionnement des autres



- ✓ Représenter la mémoire par des listes chaînées (libres/occupées)
- ✓ Libération de zones libres si zones adjacentes alors *fusionner* sinon chaîner
- ☹ Risque de fragmentation externe compactage/tassement (coûteux).
  - ⇖ la récupération est une condition nécessaire et non suffisante

Politiques d'Allocation de Mémoire

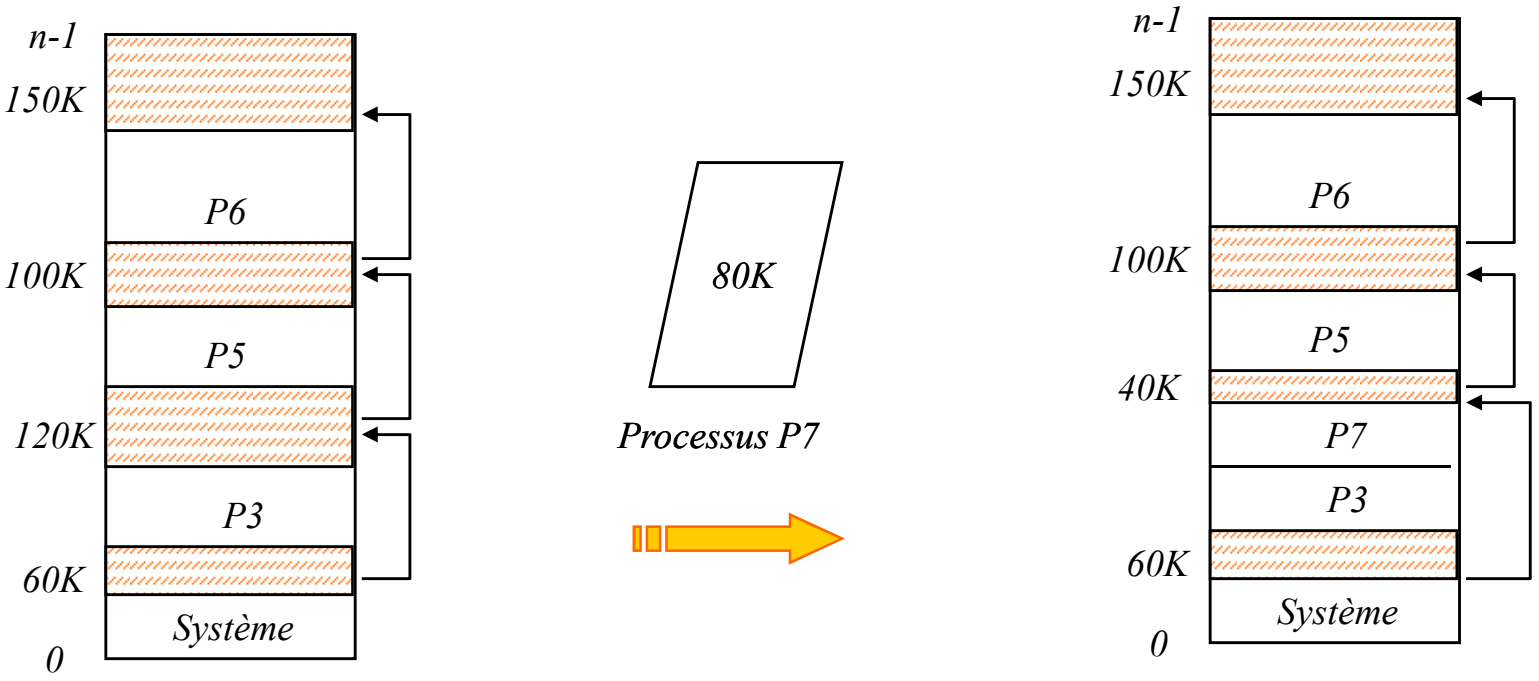
- **Questions générales :**
  - **Q1 : Quand charger ?**
    - Chargement à demande quand on a besoin
    - Pré-chargement avant d'en avoir besoin
  - **Q2 : Où charger en mémoire centrale ?**
    - S'il y a de place, quel emplacement doit-on choisir ? → ***Problème de Placement***
    - S 'il n 'y a pas de place quel est le processus victime à décharger pour le charger à sa place → ***Problème de Remplacement*** (swapping -- va-et-vient)

Algorithmes de Placements 1

- La mémoire est formée d'un ensemble de zones libres et de zones occupées (allouées)
- Allouer un programme P de taille Taille(P) :
  - Trouver une zone libre telle que Taille(zone libre) >= Taille(P)
  - 3 stratégies principales :
    - La première zone qui convient (le premier ajustement) -- First Fit
    - Le meilleur ajustement -- Best Fit
    - Le pire ajustement -- Worst Fit

Algorithmes de Placements 1 --Allocation First Fit

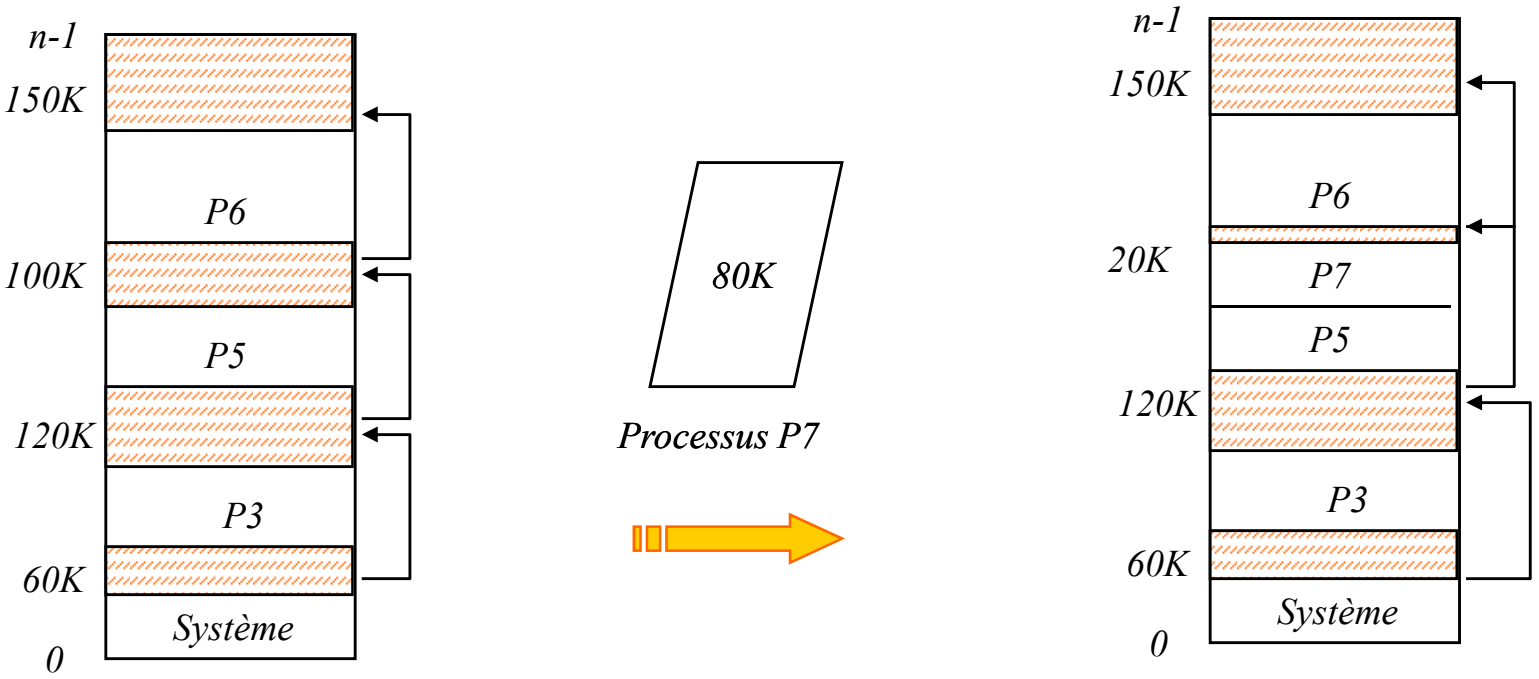
- ✓ Trouver la première zone libre suffisamment grande pour pouvoir y placer le programme



- ☺ Solution simple, peu coûteuse et la recherche est accélérée
- ☹ Concentration des résidus en tête de la liste chaînée des zones libres

Algorithmes de Placements 1 -- Allocation Best Fit

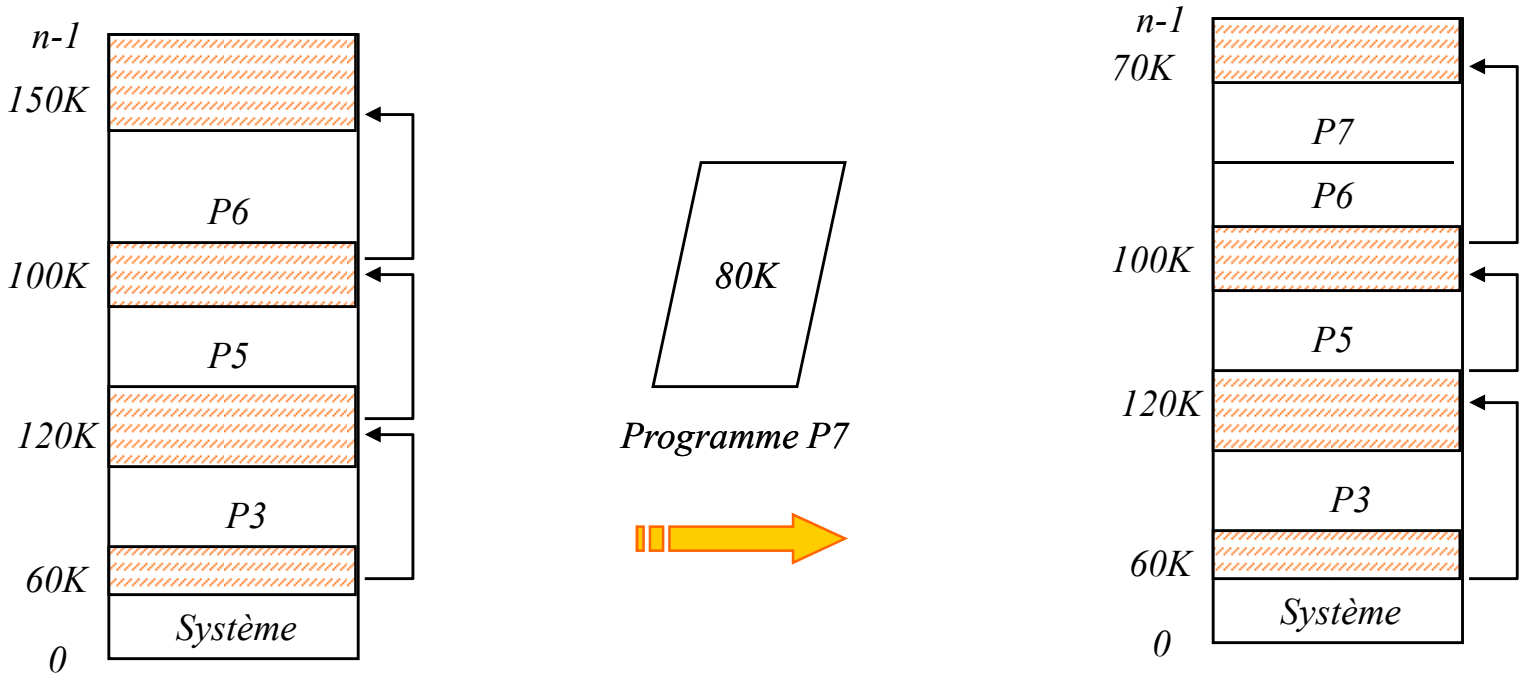
- ✓ *Chercher* dans toute la liste des zones libre, et *choisir la plus petite zone libre* qui puisse contenir le programme à allouer
  - ⇨ Produire le plus petit trou résiduel



- ☹ Parcourir toute la liste chaînée de zones libre pas assez efficace
  - 😐 Trier (ordre croissant) la liste selon les tailles des zones libres
- ☹ Génération de trous inutilisables

Algorithmes de Placements 1 -- Allocation Worst Fit

- ✓ *Politique du plus grand résidu*
- ✓ *Chercher* à placer un programme dans la zone libre la plus grande
  - ⇨ Possibilité d'utiliser le fragment pour le chargement du prochain programme



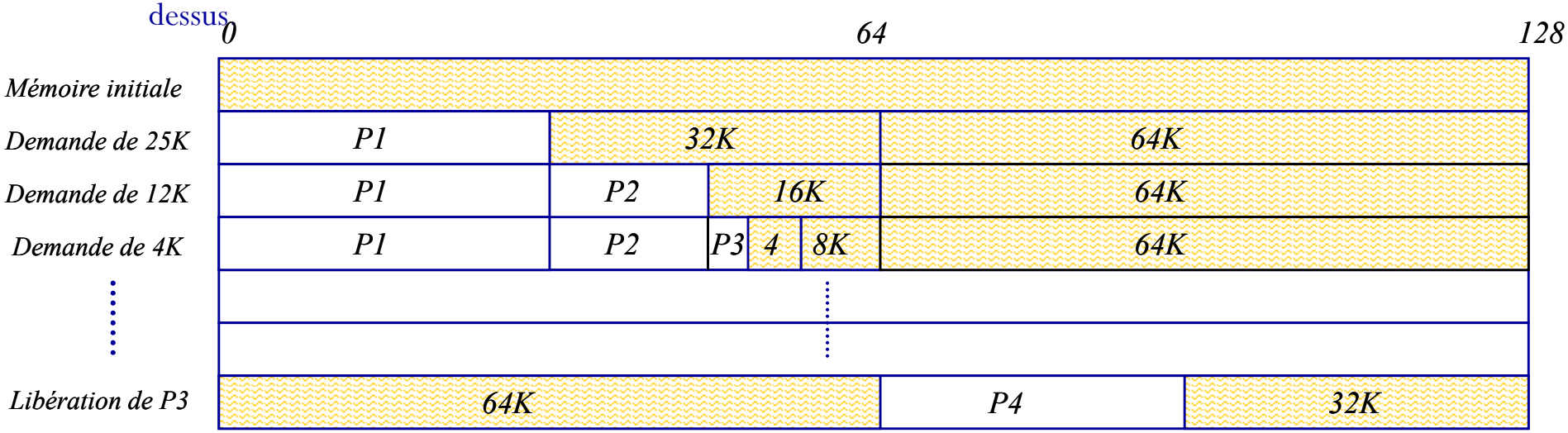
- ☺ Combattre l'émiettement en réutilisant les résidus
- ☺ *Amélioration de la méthode* : fixer une limite inférieure à la taille des résidus
  - 😐 Si Taille(résidu) ≥ $Taille_{min}$ Alors création d'une zone libre
    Sinon le résidu ne fait pas parti des zones libres
    - ☹ Fragmentation interne

Algorithmes de Placement 2 -- Buddy System

- ***Système Compagnon***
  - Subdiviser la mémoire en zones dont les tailles (nombre de mots) forment une suite croissante
  - Les tailles des zones sont quantifiées
    - *Quantification = multiple d'une certaine unité*
  - L'allocation se fait par une relation de récurrence
    - Stratégie Best Fit
- Essentiellement deux algorithmes :
  - ***Système binaire (1, 2, 4, 8, 16, 32, 64, ...)***
    - Un bloc (zone mémoire) $S_{i+1} = 2*S_i$
  - ***Suite de Fibonacci (1, 2, 3, 5, 8, 13, 21, ...)***
    - Un bloc $S_{i+1} = S_i + S_{i-1}$
- **Allocation :** soit une demande de taille S_i
  - Cas 1 : bloc libre existe et taille satisfaisante Allocation du bloc S_i
  - Cas 2 : le bloc Si n 'existe pas créer le bloc par la relation de récurrence
    - Création par subdivision
- **Libération :**
  - MAJ des blocs libres
  - Création éventuelle 'un nouveau bloc en consultant le ''compagnon ''

B. Buddy System -- Exercice d'application

- ✓ Soit un ordinateur utilisant la technique Buddy System pour la gestion de sa mémoire centrale. Initialement, la mémoire a un seul bloc de 128Ko. On dispose de 4 processus P1, P2, P3 et P4 de tailles respectives 25, 12, 4 et 28K. On désire exécuter effectuer les demandes des processus suivantes :
  1. Chargement de P1
  2. Chargement de P2
  3. Chargement de P3
  4. Terminaison de P2
  5. Chargement de P4
  6. Terminaison de P1
  7. Terminaison de P3
- ✓ En utilisant le système binaire, donnez à chaque fois l'état de la mémoire après exécution des demandes ci-dessus



Faiblesses de l'Allocation Contiguë

☺ Exigence d'allouer le programme en une zone d'un seul tenant

☺ ***Fragmentation***

⇨ Nécessité d'une opération de compactage de la mémoire

✓ ***Diviser le programme :***

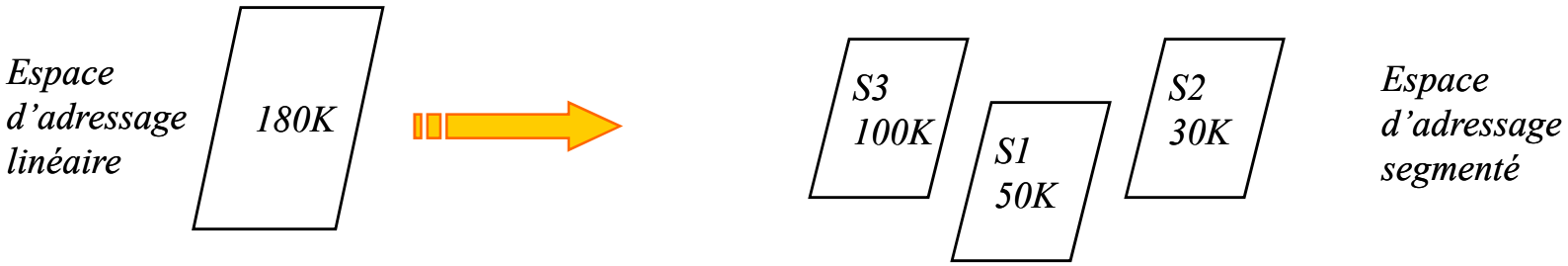
⇨ en ***segments*** (code, données, pile), qui peuvent être de taille différente

↳ Correspond à l'image que le programmeur a de son programme (données manipulées par le programme, programme principal, procédures séparées, …)

⇨ en portions de taille fixe et égale à l'unité de la mémoire centrale -- les ***pages***

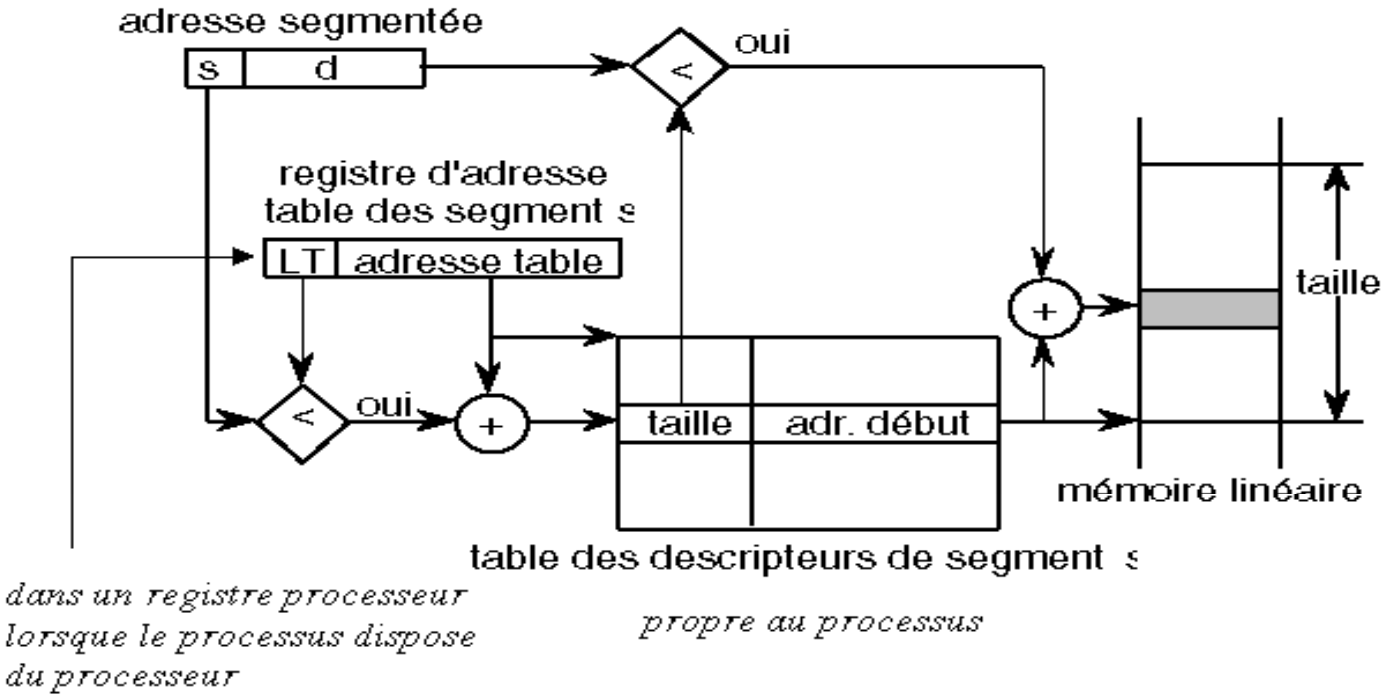
Allocation non contiguë --Segmentation

- ✓ Le programme est divisé en *segments,* un segment étant un espace d'adressage linéaire formé d'adresses contiguës.
- ✓ ***Segment*** = une partie logique du programme : segment de code, segment de données, et segment de pile.
- ✓ ***Intérêt :*** faciliter la gestion par les processus de leur espace propre (exemple le partage)
  - ⇨ espace à deux dimensions ***<N° Segment, Déplacement>***
  - ⇨ *simulée par éditeur de liens et chargeur*
  - ⇨ pris en compte par le matériel sur Multics, iAPX286, iAPX386, …



- ✓ Il faut convertir l'adresse segmentée, générée au niveau du processeur, en une adresse physique équivalente.
  - ⇖ ***Adresse physique = adresse implantation segment + déplacement***

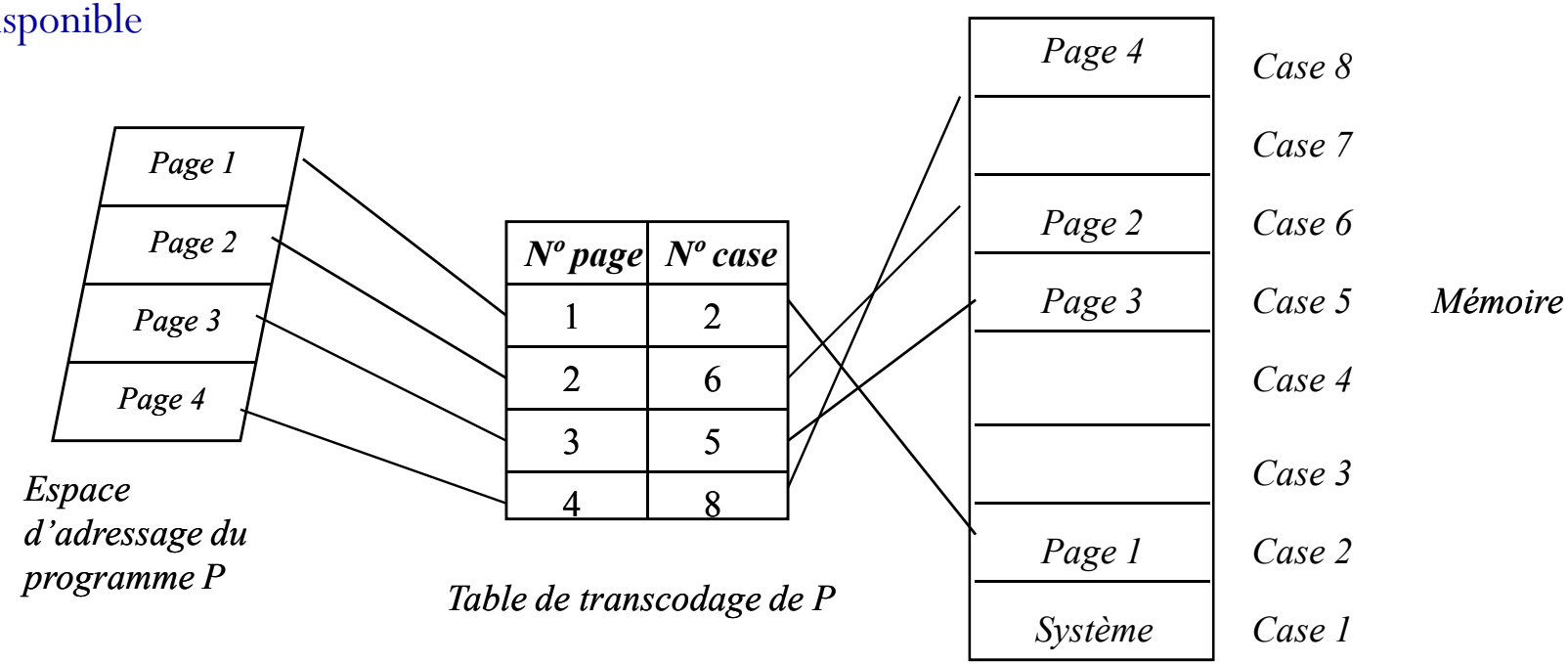
Segmentation --Mémoire Segmentée



- ✓ Allouer un segment S de taille Taille(S) :
  - ⇨ Trouver une zone libre telle que Taille(Zone libre) ≥ Taille(S)
  - ⇨ ***Allocations et libérations successives des segments peuvent créer également un problème de fragmentation***

Allocation non contiguë --Pagination

- ✓ La pagination permet d'avoir en mémoire un processus dont les adresses sont non contiguës
  - ⇨ L'espace d 'adressage du programme est découpé en morceaux linéaires de même taille -- la ***page*** (qcq Ko).
  - ⇨ L'espace de la mémoire physique est lui-même découpé en morceaux linéaires de même taille -- la ***case***
- ✓ Charger un programme en mémoire centrale consiste à placer les pages dans n 'importe quelle case disponible



La Mémoire Segmentée Paginée

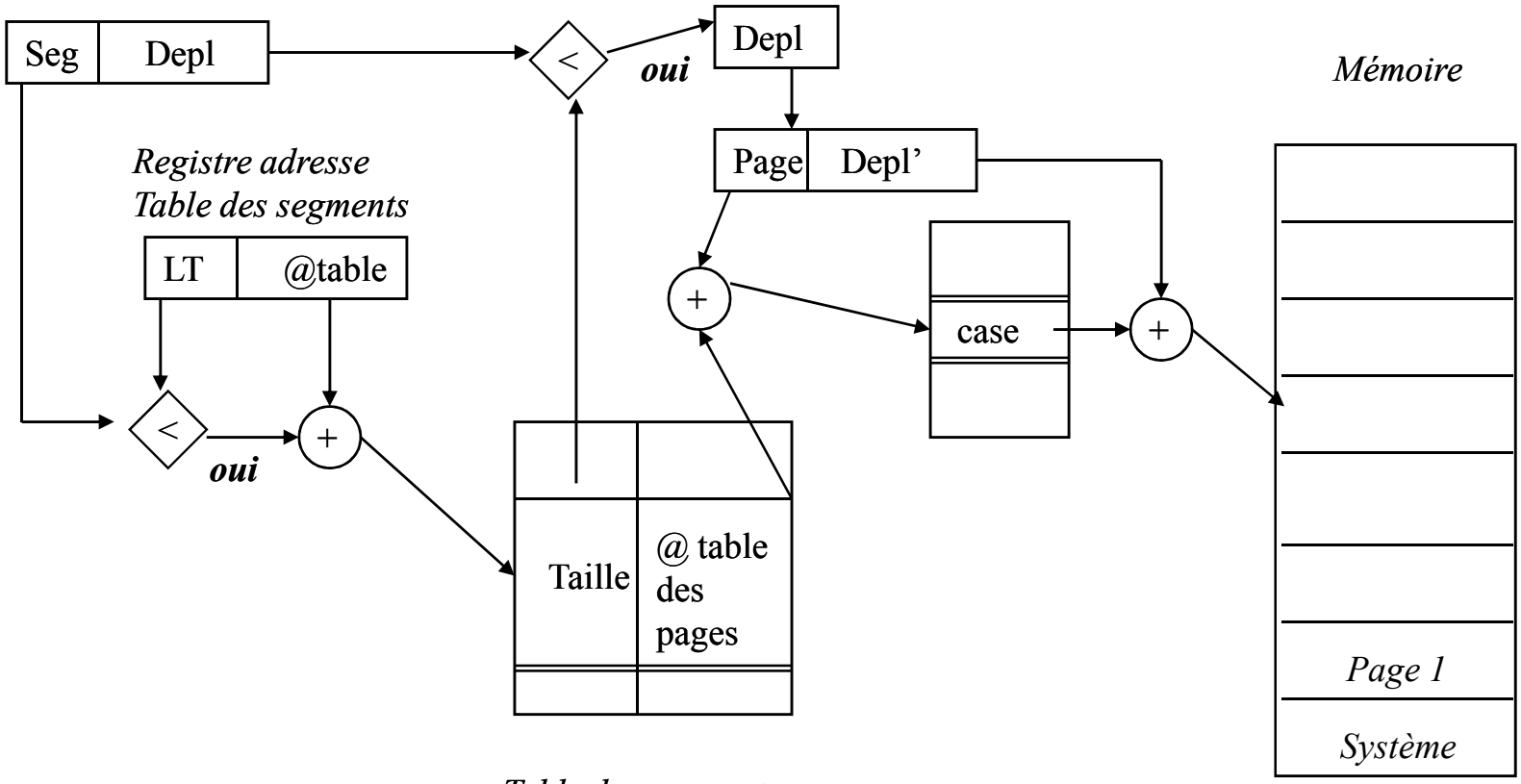
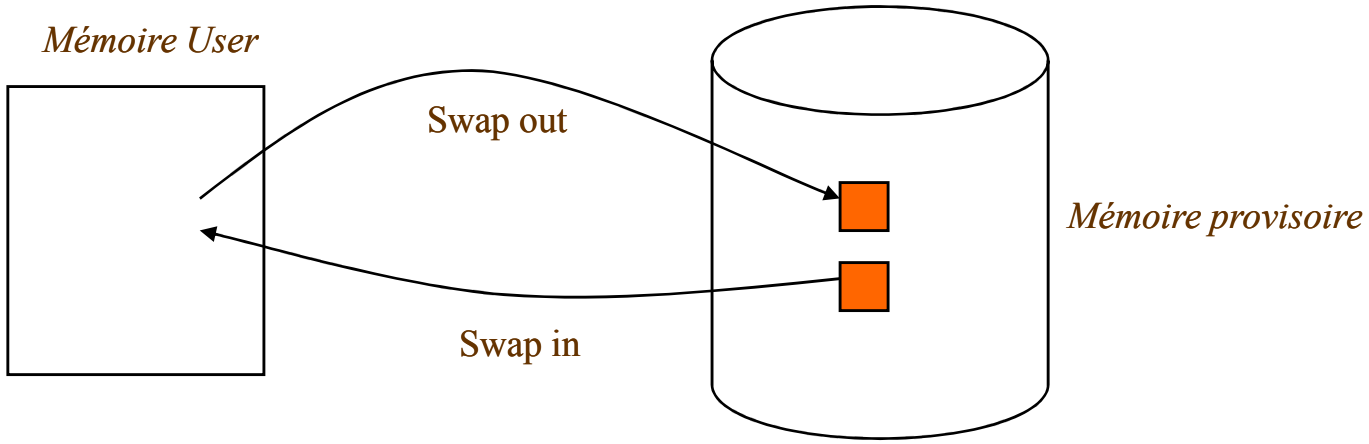


Table des segments

- ✓ Une table de segment est définie pour chaque segment de l'espace d'adressage du processus
  - ⇨ Chaque segment est à son tour paginé :
    - ↳ Il existe une table des pages pour chaque segment
    - ↳ Déplacement dans un segment = <N°page, Déplacement dans la page>

Le Va-Et-Vient (SWAPPING)

- ✓ Lorsque tous les processus ne peuvent pas tenir simultanément en mémoire
  - ⇨ Taille des processus en cours d'exécution > la taille de la mémoire physique.
  - ⇨ Déplacer temporairement certains sur une mémoire provisoire (partie réservée du disque)



- ✓ Allocation à la demande de la zone de va-et-vient sur disque:
  - ⇨ Quand un processus est déchargé de la MC, on lui cherche une place (même gestion que celle de la MC)
  - ⇨ Lors d'un déchargement, le processus est sure d'avoir une zone d'attente libre sur le disque
  - ⇨ La zone de va-et-vient doit être suffisamment grande pour contenir à la fois les images des processus actifs que celles de ceux qui ont été déchargés car l 'espace qu'ils occupaient a été réquisitionné.
- ☹ Le swapping s'il permet de pallier le manque de mémoire nécessaire à plusieurs processus users, *n'autorise cependant pas l'exécution de programme de taille supérieure à celle de la MC.*

Récapitulatif -- Gestion de la Mémoire Physique

- ✓ L'allocation en ***partitions*** ou en ***régions*** considère le programme comme un ensemble d'adresses *insécables*.
  - ⇨ Problème de *fragmentation* nécessite d 'une opération de compactage de la mémoire centrale
- ✓ La ***segmentation*** découpe l'espace d 'adressage du programme en segments correspondant à des morceaux logiques du programme.
  - ⇨ Une adresse générée par le processeur est de la forme <*N° segment, déplacement*>
  - ⇨ La table des segments du processus permet de transformer l'adresse segmentée en adresse physique
- ✓ La ***pagination*** découpe l 'espace d 'adressage du programme en pages et la mémoire physique en cases de même taille.
  - ⇨ Une adresse générée par le processeur est de la forme <*N ° page, déplacement*>
  - ⇨ La table des pages du processus permet de traduire l'adresse paginée en adresse physique
- ✓ *Segmentation et pagination sont très souvent associées*

Partie II. La Mémoire Virtuelle

Contenu du cours

1. Présentation
2. La pagination -- transformation des adresses
3. Les algorithmes de Remplacement
   - ⇨ FIFO/OPTIMAL/LRU/NRU
4. Autres Considérations
   - ⇨ Politiques d'allocation locale/globale
   - ⇨ Ecroulement
   - ⇨ L'ensemble de travail
   - ⇨ Prépagination

Présentation

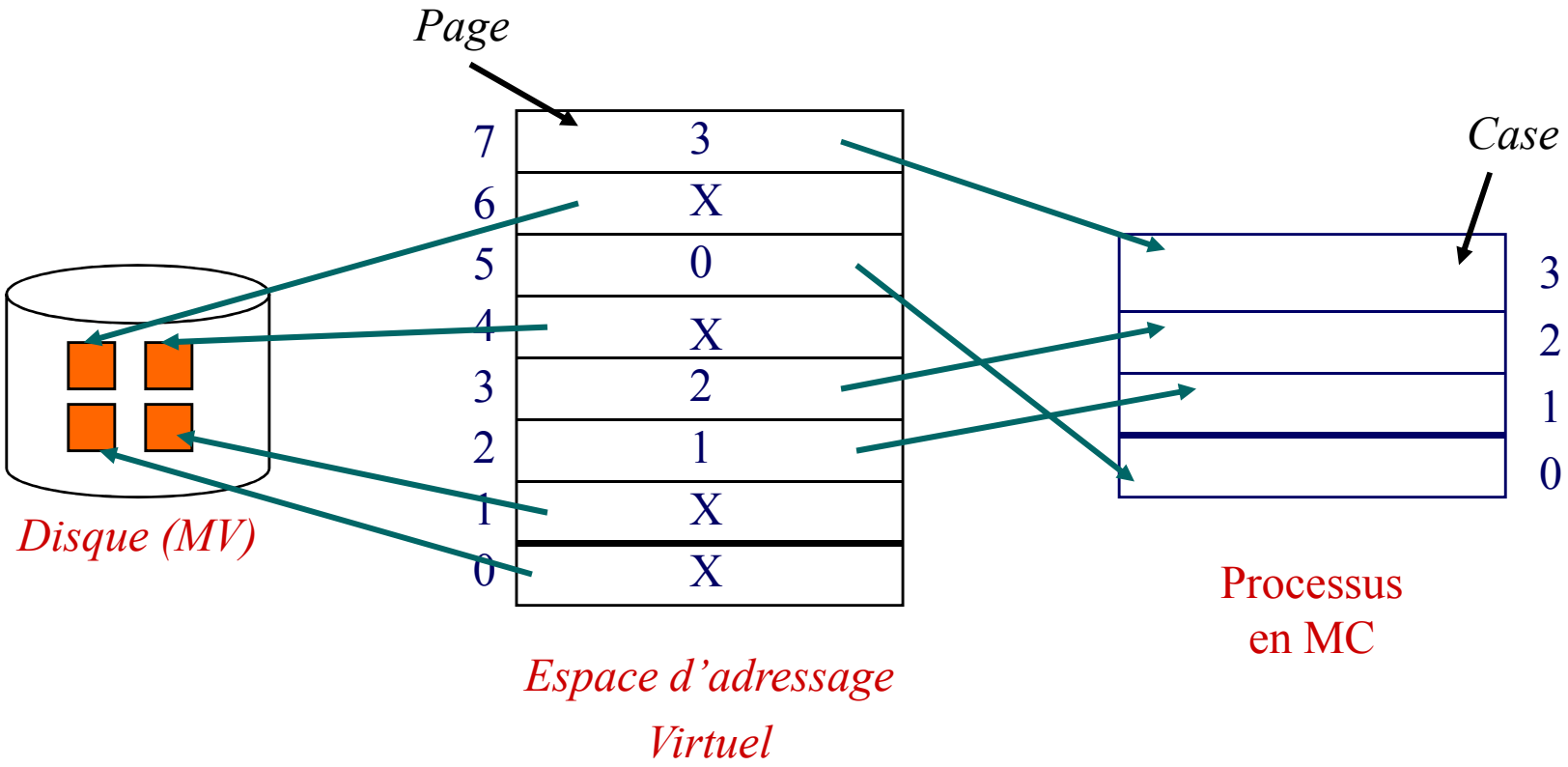
- ✓ **Objectif:** fournir un espace d'adressage indépendant de celui de la mémoire physique
  - ⇨ La mémoire virtuelle permet d'exécuter des programmes dont la taille excède celle de la mémoire physique
  - ⇨ L'espace d'adressage virtuel >> l'espace physique
  - ⇨ Allocation non contiguë
  - ⇨ Facilité de mise en œuvre de la multiprogrammation

Présentation (suite)

- ✓ **Réalisation de la mémoire virtuelle (MV)**
  - ⇨ Représentation physique : MC + MS (disque)
  - ⇨ Gestion basée sur les techniques de pagination
- ✓ **Pagination --Principe:**
  - ⇨ l'espace d'adressage virtuel est divisé en petites unités --**PAGES**
  - ⇨ l'espace d'adressage physique est aussi divisé en petites unités -- **CASES** (frames)
  - ⇨ Les pages et les cases sont de même tailles

Pagination -- Principe

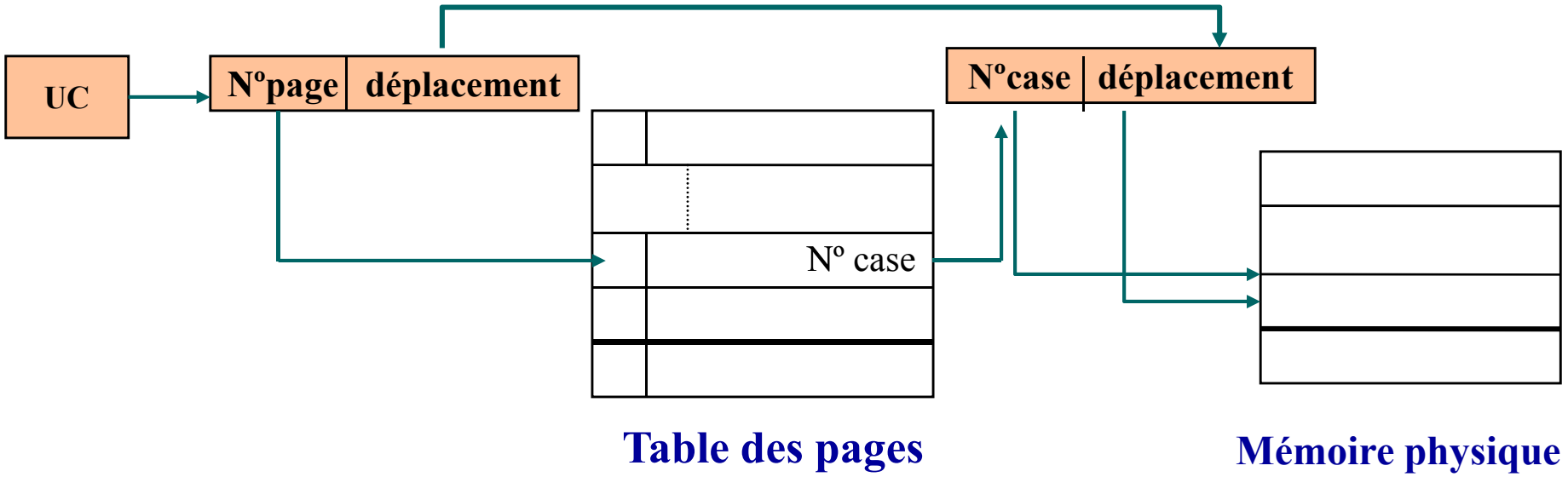
- ✓ On ne charge qu'un ensemble de pages en mémoire :
  - ⇨ ce sous-ensemble est appelé l'espace physique ou réel



- ✓ ***Ne charger que les pages utiles à un instant donné***

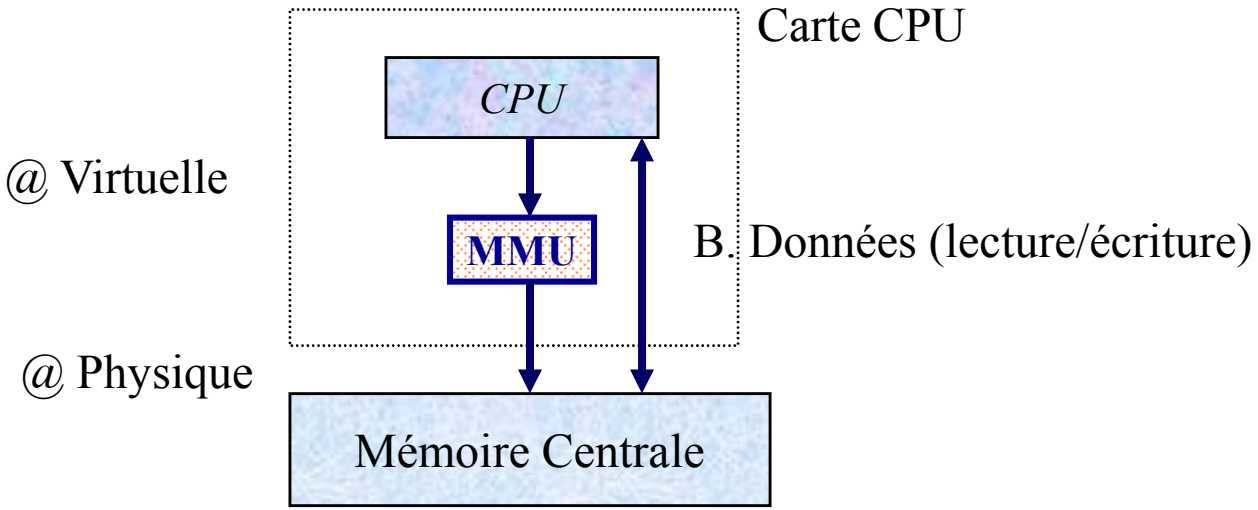
Transformation des adresses virtuelles (1)

- ✓ Les adresses manipulées par le programmes sont des adresses virtuelles
- ✓ Lorsqu'une adresse est générée, elle est transcodée, grâce à une table, pour lui faire correspondre son équivalent en mémoire physique



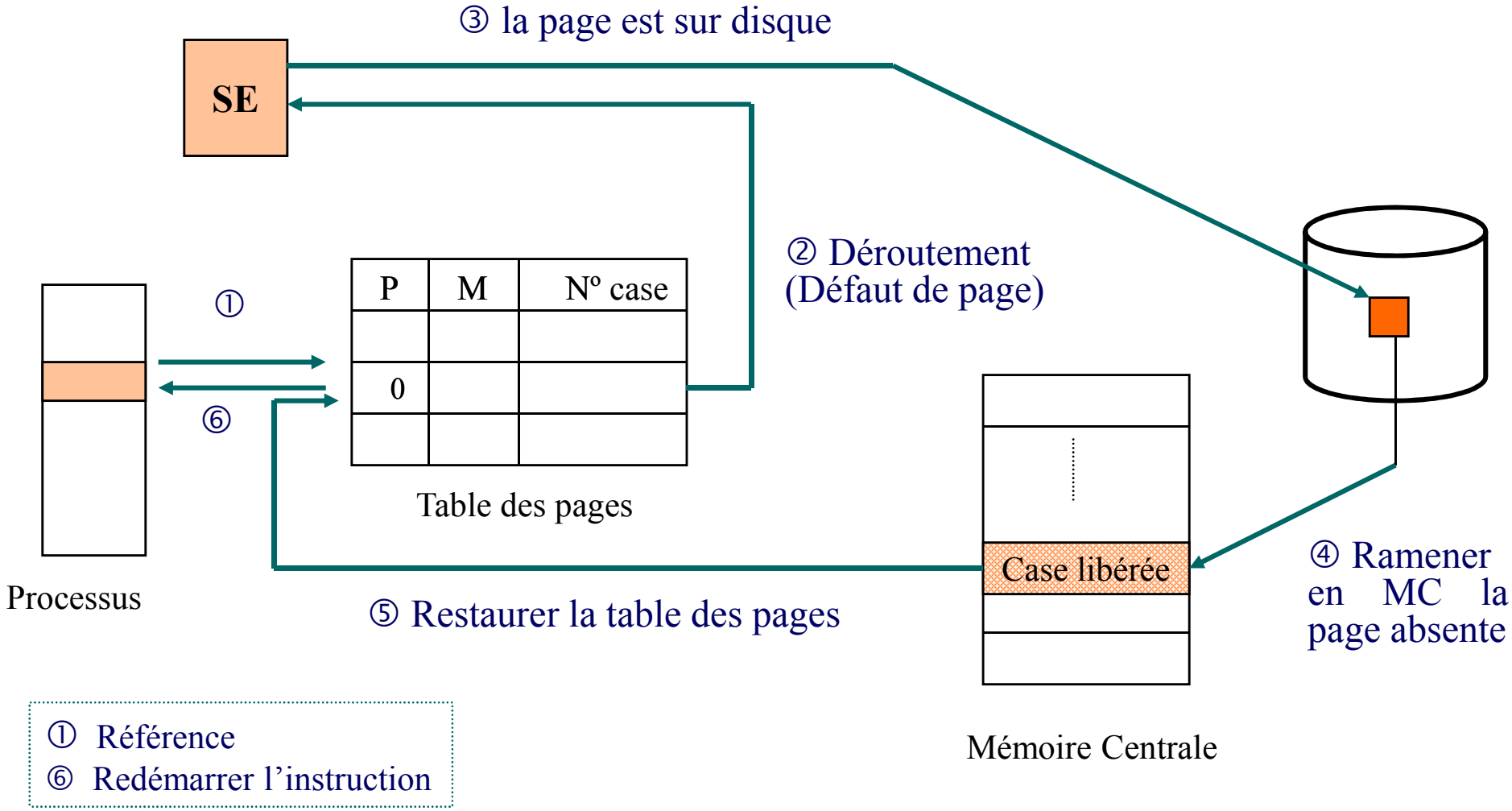
Transformation des adresses virtuelles (2)

- ✓ Ce transcodage est effectué par des circuits matériels de gestion : unité de gestion de mémoire -- **MMU (Memory Management Unit)**



- ✓ Si l'adresse générée correspond à une adresse mémoire physique, le MMU transmet sur le bus l'adresse réelle, sinon il se produit un ***DEFAUT DE PAGE*** ***(Page Fault --PF)***
- ✓ Chaque table des pages contient les champs nécessaires au transcodage, avec notamment :
  - ⇨ 1 bit de présence (**P** 1/0) pour marquer la présence de la page en mémoire physique
  - ⇨ 1 bit de modification (**M** 0/1) pour signaler si on écrit dans la page

Bit de Présence et Défaut de Page



Exemple

- ***Codage des @ virtuelle ou réelle***

  ⇨ on réserve les bits de poids nécessaires pour coder les numéros des pages ou des cases

  ⇨ les bits de poids faibles codent le déplacement

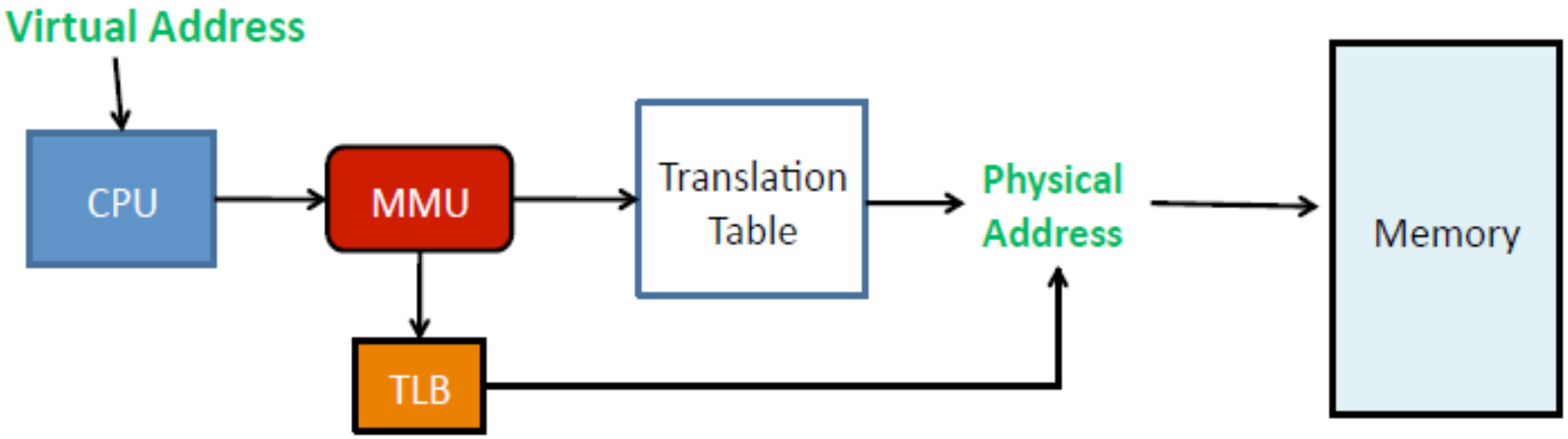
- Taille(page) = 4Ko; Taille(MC) = 32 cases; Taille(processus) = 128 pages

  ⇨ Combien de bits a-t-on besoin pour représenter les @ virtuelles et les @ réelles?

Représentation d'une Table des Pages

- ✓ **Chaque processus a sa propre table des pages**
- ✓ **Différentes organisations des tables des pages :**
  - ⇨ **Table des pages dans une mémoire cache –**TLB (Translation Lookaside Buffer)
    - ↳ très rapide/Coûteuse
  - ⇨ **Table des pages dans la MC:**
    - ↳ l'adresse à la table est maintenue dans un registre
    - ↳ chaque référence implique 2 accès à la MC (obtenir le numéro de page dans la table et le deuxième concerne l '@ physique

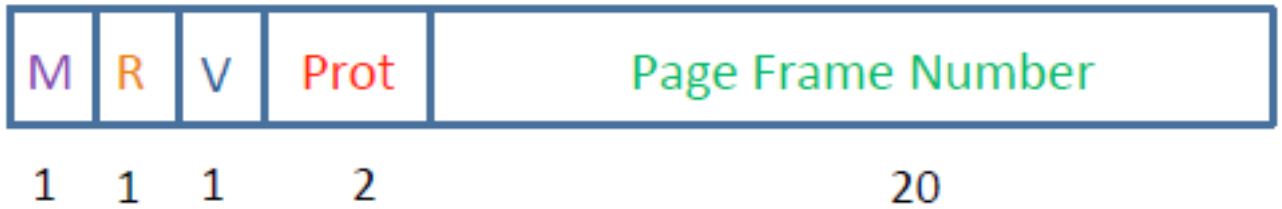
Représentation d'une Table des Pages (suite)



Source: *Frank Uyeda, Memory Management, CSE 120: Principles of Operatng Systems, UC San Diego, 2009*

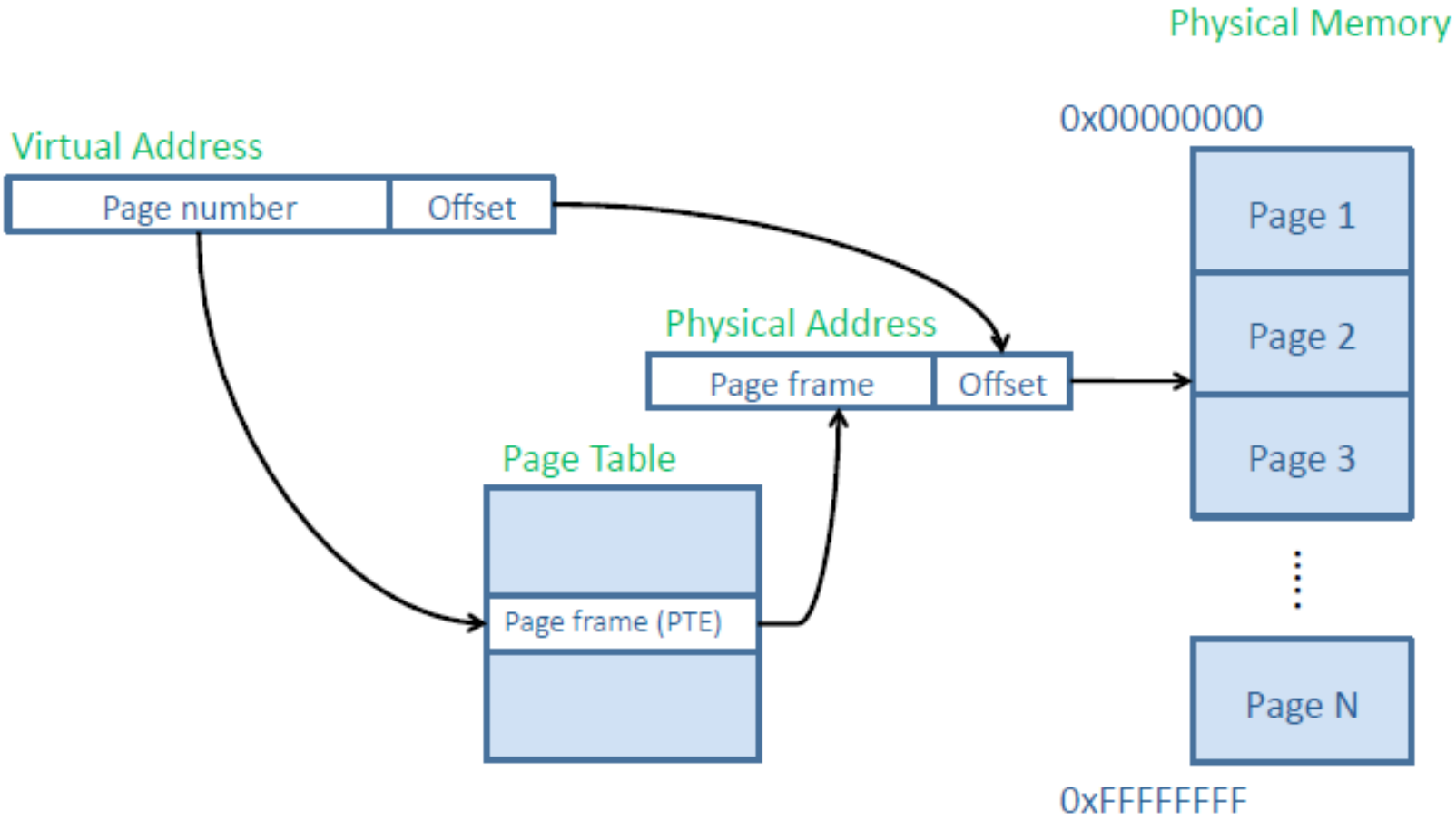
Les entrées d'une able des Pages (PTE) - Récap.

- ✓ **PTE --**Page Table Entries



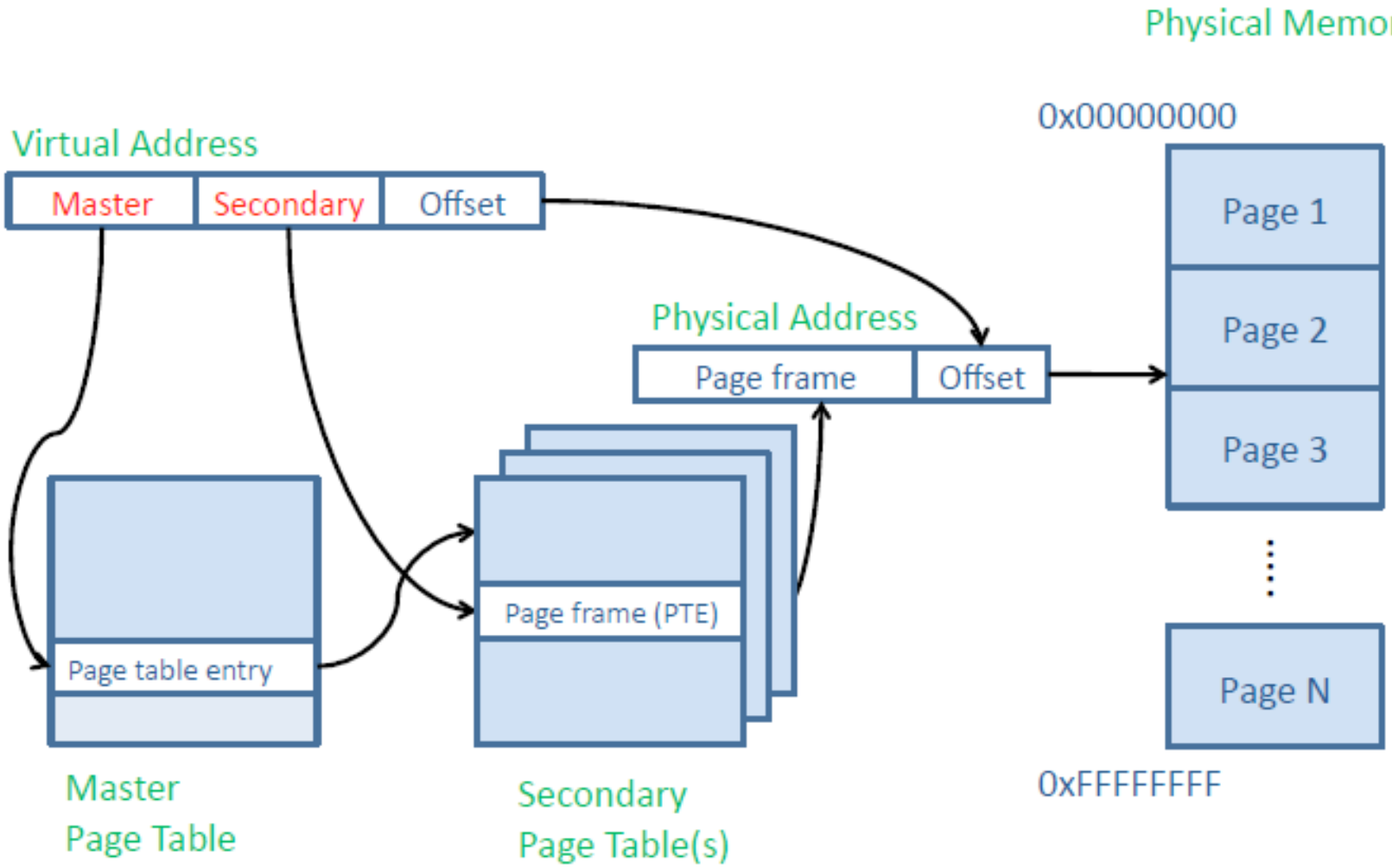
- ✓ Page table entries control mapping
  - ⇨ The **Modify** bit says whether or not the page has been written
    - ↳ It is set when a write to the page occurs
  - ⇨ The Reference bit says whether the page has been accessed
    - ↳ It is set when a read or write to the page occurs
  - ⇨ The **Valid** bit says whether or not the PTE can be used
    - ↳ It is checked each >me the virtual address is used
  - ⇨ The **Protect** >on bits say what opera>ons are allowed on page
    - ↳ Read, write, execute
  - ⇨ The **page frame number (PFN)** determines physical page
- ✓ **Note**: when you do exercises in exams or hw, we'll open tell you to ignore counting M,R,V,Prot bits when calculating size of structures

Table des Pages à un niveau –Recap.



Source: *Frank Uyeda, Memory Management, CSE 120: Principles of Opera>ng Systems, UC San Diego, 2009*

Table des Pages à deux niveaux



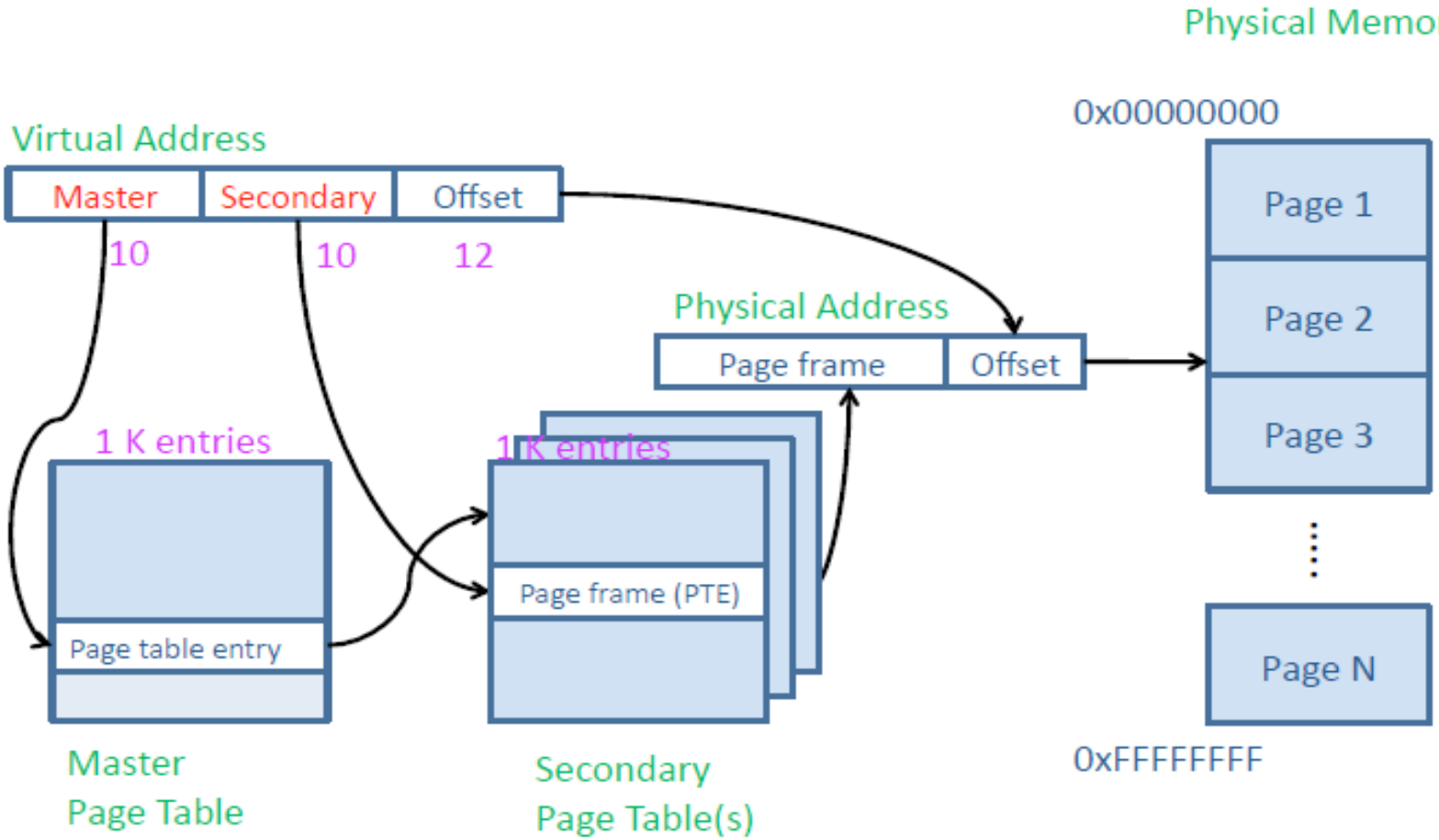
Source: *Frank Uyeda, Memory Management, CSE 120: Principles of Opera>ng Systems, UC San Diego, 2009*

Table des Pages à deux niveaux (suite)

Source: *Frank Uyeda, Memory Management, CSE 120: Principles of Opera>ng Systems, UC San Diego, 2009*

- ✓ Originally, virtual addresses (VAs) had two parts
  - ⇨ Page number (which mapped to frame) and an offset
- ✓ Now VAs have three parts:
  - ⇨ Master page number, secondary page number, and offset
- ✓ **Master page table** maps VAs to secondary page table
  - ⇨ We'd like a manageable master page size
- ✓ **Secondary table** maps page number to physical page
  - ⇨ Determines which physical frame the address resides in
- ✓ Offset indicates which byte in physical page
  - ⇨ Final system page/frame size is s>ll the same, so offset length stays the same

Table des Pages à deux niveaux --Exemple



Source: *Frank Uyeda, Memory Management, CSE 120: Principles of Opera>ng Systems, UC San Diego, 2009*

Algorithmes de Remplacement de Pages

- A la suite d'un défaut de page, le SE doit retirer une page de la MC pour libérer de la place manquante
  - ***Problème :*** quelle page choisir à décharger afin de récupérer l'espace et minimiser le nombre de défauts de pages?
- Plusieurs considérations doivent être tenues en compte :
  - il est - coûteux de remplacer une page qui n 'a pas été modifiée en MC (inutile de la recopier sur disque car elle existe déjà!).
    - Une page d'un segment de code est préférable par rapport à une page d'un segment de données, qui aura certainement été modifiée depuis son chargement.
    - Partage de pages entre plusieurs processus
    - Date d'utilisation
- ***Plusieurs algorithmes de remplacements :***
  - Aléatoire
  - Première entrée, première sortie -- FIFO
  - Optimal
  - Remplacement de la page la moins récemment utilisée -- LRU (Least Recently Used)
  - Remplacement d 'une page non récemment utilisée -- NRU (Not Recently Used)

Algorithmes de Remplacement (2)

- **Algorithme aléatoire** -- Random:
  - ⇨ La victime est choisie au hasard
- **FIFO :**
  - ⇨ Lors d'un défaut de page, la page la plus anciennement chargée est la page retirée pour être remplacée
  - ☺ Facile à implanter
  - ☹ Remplacement d'une page très référencée → trop de défaut de pages
- **Exemple 1 :** Supposons avoir 3 cases et 4 pages avec la chaîne de référence suivante :

A B C A B D A D B C B

cases\ref.	*A*	*B*	*C*	*A*	*B*	*D*	*A*	*D*	*B*	*C*	*B*
1	A					D				C	
2		B					A				
3			C						B		
	DP	*DP*	*DP*			*DP*	*DP*		*DP*	*DP*	

7 défauts de pages
(dont 3 placements 4 remplacements)

Algorithmes de Remplacement (3)

- **Algorithme Optimal :**
  - ***Principe :***
    - choisir comme victime la page qui sera référencée le plus tard possible
    - Nécessite la connaissance, pour chacune des pages, le nombre d'instructions qui seront exécutées avant que la page soit référencée
  - Algorithme irréalisable dans un contexte ''offline''
    - Connaissance des références qui seront faites
  - ***Intérêt :*** permet de comparer les performances des autres algorithmes
  - ***Exemple :*** reprendre exemple 1 en appliquant optimal

cases\ref.	*A*	*B*	*C*	*A*	*B*	*D*	*A*	*D*	*B*	*C*	*B*
1	A									C	
2		B									
3			C			D					
	DP	*DP*	*DP*			*DP*				*DP*	

5 défauts de pages
(dont 2 remplacements)

Algorithmes de Remplacement (4)

- **Algorithme LRU (Least Recently Used) :**
  - Principe : remplacer la page la moins récemment utilisée (accédée)
    - Remplacer la page qui est restée inutilisée le plus de temps
  - Une bonne approximation de l'algorithme optimal
  - Théoriquement réalisable mais très coûteux
    - Nécessite des dispositifs matériels particuliers (compteur pour chaque référence)
  - ***Exemple 2 :*** énoncé exemple 1 + la chaîne de référence A B C D A B C D A B C D

cases\ réf.	*A*	*B*	*C*	*D*	*A*	*B*	*C*	*D*	*A*	*B*	*C*	*D*
1	A			D			C			B		
2		B			A			D			C	
3			C			B			A			D

LRU :
12 défauts de pages
Optimal : ?

Algorithmes de Remplacement (5)

- **Algorithme NRU (Not Recently Used) :**
  - ***Idée :*** marquer les pages référencées
  - ***Principe :***
    - A chaque page sont associées deux bits R et M :
      - R=1 chaque fois que la page est référencée (lecture/écriture), R=0 sinon
      - M=1 lorsque la page a été modifiée dans la mémoire centrale
    - Au lancement d'un processus, le SE met à zéro R et M de toutes les pages
    - Périodiquement, le bit R est remis à 0 pour différencier les pages qui n'ont pas été récemment référencées des autres
    - Lors d'un défaut de page, le SE retire une page au hasard dont la valeur MR est la plus petite :
      - MR = 00 : non référencée, non modifiée
      - MR = 01 : non modifiée, référencée
      - MR = 10 : modifiée, non référencée,
      - MR = 11 : référencée, modifiée
    - ☹ Algorithme basé sur une solution matérielle

Algorithmes de Remplacement -- Conclusion

- ***Critère de performance:***

- La taille de la MC influe beaucoup sur les performances que l 'algorithme :
  - ⇨ N'essayer pas de raffiner un algorithme, mais plutôt augmenter, si nécessaire, la taille de la mémoire, et ce afin de minimiser le remplacement.
- ***Question :*** est-ce que le rajout de mémoire réduit toujours le nombre de défauts de pages?
  - ⇨ Oui pour seulement optimal et LRU
  - ⇨ Non pour FIFO!!
- ***Exemple 3 :*** on dispose de 3 cases mémoires. Appliquer FIFO à la chaîne de références suivante :

A B C D A B E A B C D E

Rajouter 1 case et reprendre l 'algorithme de remplacement FIFO. Conclure

Conversion d'une Adresse Virtuelle

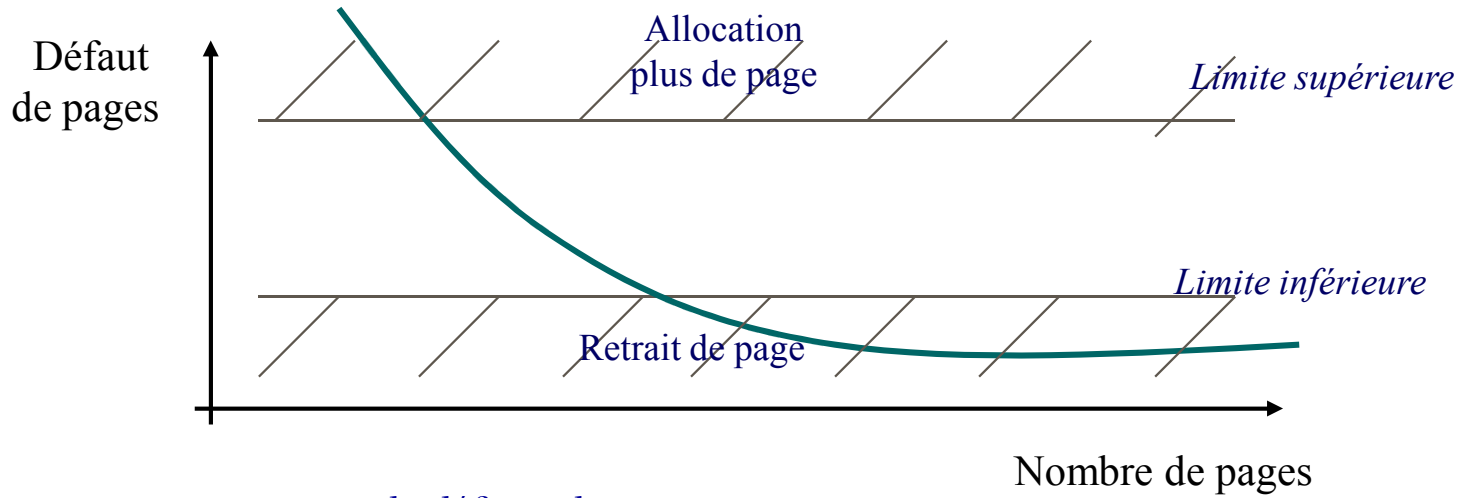
```
Procédure Conversion (Entrée : advirt; Sortie : adphysique)
Début
        index = advirt.page + adresse_table(processus)
        Si (non index.P)                    /* page absente  */
        Alors /*  Défaut de page*/
                charger_page(advirt.page, adresse_case
                 index.P = 1
                index.case = adresse_case
        finsi
         adphysique = adresse_case + advirt.deplacement
Fin
Procédure charger_page (E : page; S : case)
Début
      Si (Non trouver_case_libre())
      Alors choisir_case_à_libérer(case_à_libérer, page_victime)
                Si (page_victime.M) Alors ecrire_disque(page_victime) finsi
                lire_disque(case_voila_liberer, page)
      finsi
Fin
```

Autres Considérations (1)

- ***Politique d'allocation locale / globale :***
  - Remplacement de la page la plus ancienne :
    - Globale -- la plus ancienne du système
    - Locale -- la plus ancienne du processus
    - En général, l 'allocation globale produit de meilleurs résultats
- ***Redémarrage d'une instruction après le défaut de page***
- ***La taille d'une page?*** Influe sur les tables de pages utilisées par la MMU
- ***Mémoire de SWAP***
  - où stocker les pages délogées de la MC?
  - sur un ou plusieurs disques locaux
    - partition de swap : + rapide, - de place pour le SGF
    - Fichier de swap : - rapide, + de place pour les autres fichiers
    - En général, le SE utilise les deux simultanément
    - Plusieurs disques = swap en parallèle
  - sur un serveur (de disques) distant :
    - Net PC, TX, STB, …

Autres Considérations (2)

- ***Ecroulement -- thrashing***
  - Si le nombre de processus est très grand, l'espace propre à chacun est insuffisant et ils passeront leur temps à gérer des défauts de pages
  - Ecroulement du système : une haute activité de pagination
    - un processus s'écroule lorsqu'il passe plus de temps à paginer qu'à s'exécuter
- ***Limiter le risque d'écroulement :***



- Si un processus provoque trop de défauts de pages :
  - au dessus d 'une limite supérieure : on lui allouera plus de pages
  - en dessous d'une limite inférieure : on lui en retirera
- S'il y a plus de pages disponibles et trop de défauts de pages, on devra suspendre un des processus

Autres Considérations (3)

- ✓ ***L'espace de travail -- Working set (W)***
  - ⇨ W = les pages d'un processus référencées sur un court instant de temps
- ✓ Une allocation optimale : allouer à un processus actif autant de pages que nécessite W
  - ⇨ les défauts de pages seront provoqué lors des changements d'espace de travail
    - ⇘ Ce modèle n'est utilisé que pour la pré-pagination
- ✓ ***Pré-pagination***
  - ⇨ Lors du lancement d'un processus ou lors de sa reprise après suspension, on provoque obligatoirement un certain nombre de défauts de page
    - ⇘ Essayer de les limiter -- enregistrer W avant suspension
    - ⇘ Au lancement d 'un programme les 1eres pages de code seront vraisemblablement exécutées

Conclusions

- ✓ Découpage en pages et cases de même taille
- ✓ Les pages d'un processus ne sont chargées en mémoire physique que lorsque le processus y accède
  - ⇨ les pages peuvent être mises dans n'importe quelle case
- ✓ Lorsqu'un processus accède à une page non présente en mémoire physique, il se produit un DEFAUT DE PAGE :
  - ⇨ la page manquante est alors chargée dans une case libre
  - ⇨ pas de case libre, le système utilise un algorithme de remplacement de page pour choisir une case à libérer