

# Gestion de la Mémoire

- Bases
- Swapping
- Allocation Contigue
- Pagination
- Ségmentation
- Ségmentation avec Pagination

# Bases

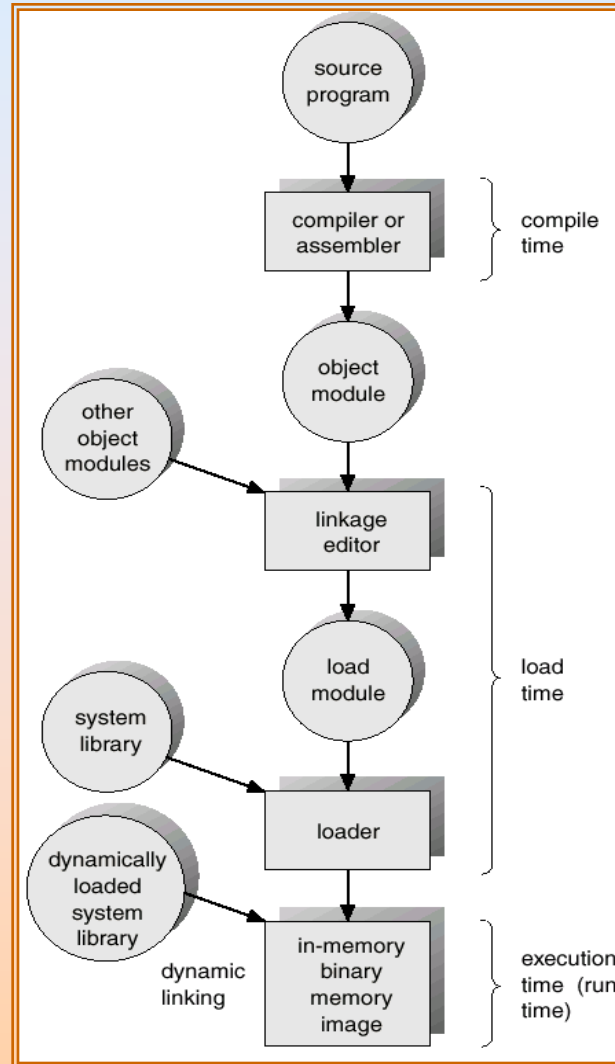
- Le programme doit être mis en mémoire et intégré à un processus pour l'exécution
- **File d'entrée** – collection de processus sur le disque en attente d'être exécutés en mémoire
- Programmes utilisateurs passent par plusieurs étapes avant d'être exécutés

# Lier Instructions et Données à Mémoire

Lier les adresses d'instructions et des données aux adresses mémoire peut se faire à trois étapes différentes

- **A la compilation:** Si le lieu mémoire est connu à priori, du *code absolu* peut être généré; doit recompiler le code si l'emplacement d'exécution change
- **Au chargement:** Doit générer du code *relogeable* si le lieu mémoire n'est pas connu à la compilation
- **A l'exécution:** Lien différés jusqu'à l'exécution si le processus peut être déplacé durant son exécution d'un segment mémoire à un autre. Besoin d'un support matériel pour le mapping d'adresses (e.g., registres *base* et *limite*).

# Etapes de Passage d'un Programme Utilisateur



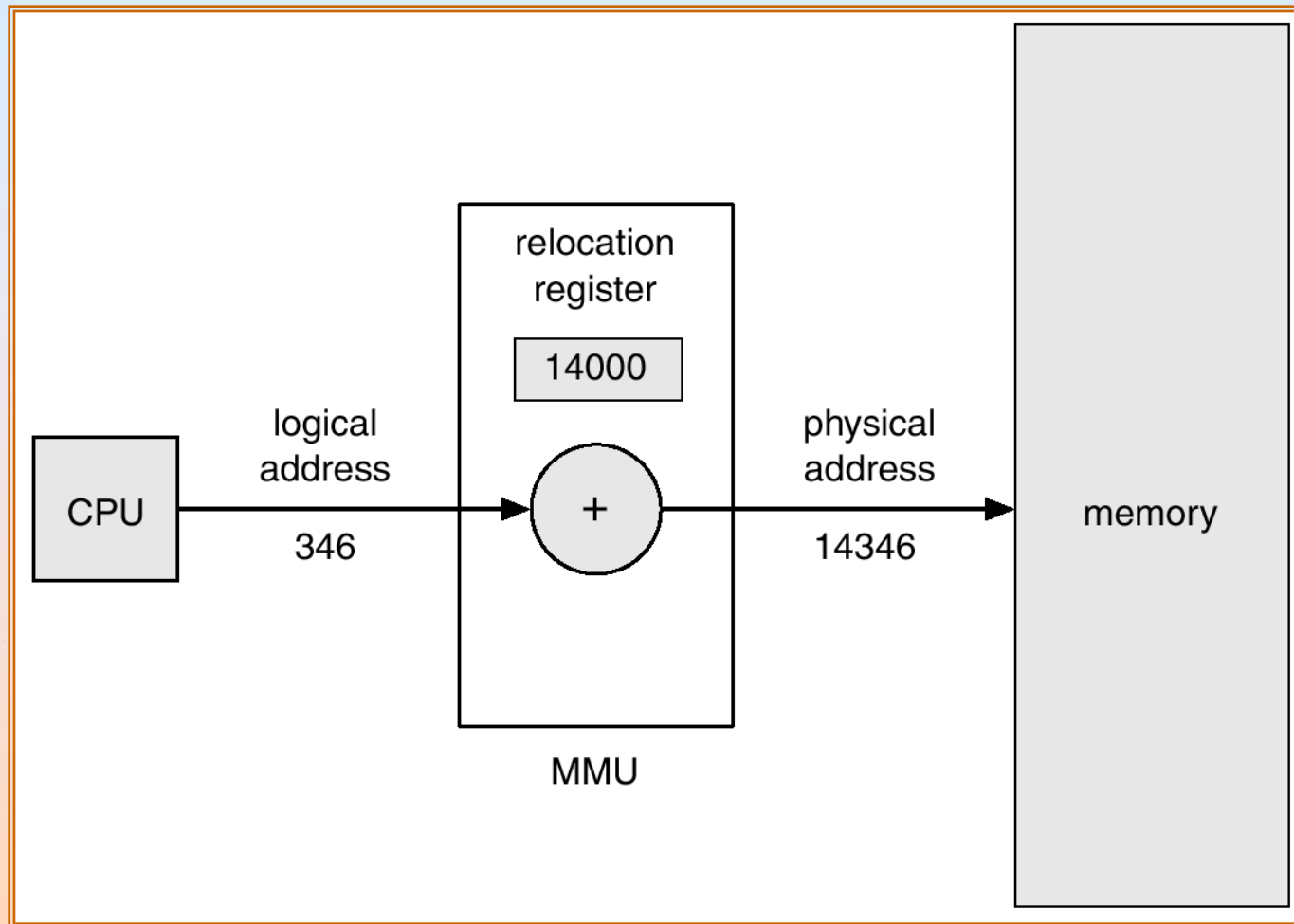
# Espace d'Adressage Logique vs. Physique

- Le concept d'*espace d'adressage logique* lié à un *espace d'adressage physique* différent est central à une gestion propre de la gestion mémoire
  - **Adresse logique** – généré par la CPU; appelée aussi *adresse virtuelle*
  - **Adresse physique** – adresse reçue par l'unité mémoire
- Adresses logiques et physiques sont les mêmes pour les schémas de lien à la compilation et au chargement, mais ils sont différents dans un schéma de lien à l'exécution

# Unité de Gestion Mémoire (MMU)

- En anglais : Memory Management Unit
- Matériel de mapping des adresses virtuelles en adresses physiques
- La valeur du registre d'emplacement est ajoutée à chaque adresse générée par un processus au moment de l'envoi à la mémoire
- Le processus traite des adresses *logiques*; il ne voit jamais les adresses physiques

# Relocation Dynamique



# Chargement Dynamique

- Routine non chargée avant son invocation
- Meilleure utilisation mémoire; routine non utilisée jamais chargée
- Utile quand du code assez important en taille n'est utilisé que pour gérer des cas très peu fréquents
- Rien à changer dans la conception du programme pour supporter le chargement dynamique (Support OS transparent)



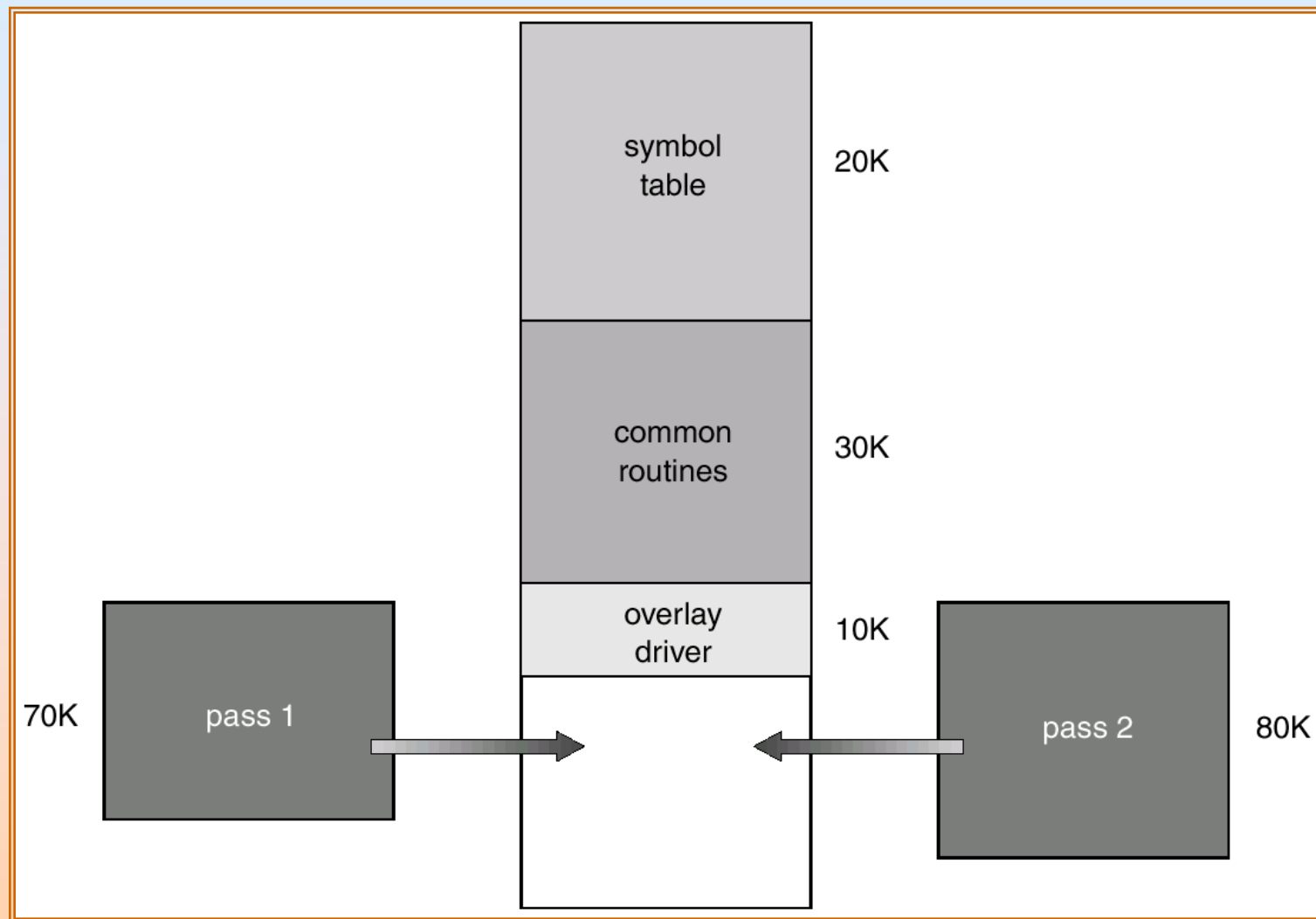
# Lien Dynamique

- Lien retardé jusqu'à l'exécution
- Un petit bout de code, *stub*, utilisé pour localiser la bonne routine
- Stub remplace son adresse dans le code par l'adresse de la routine, puis exécute la routine
- OS vérifie que la routine est dans l'espace d'adressage du processus
- Liens dynamiques très utiles pour les bibliothèques

# Recouvrements

- Laisser en mémoire seulement les instructions et les données qui sont utiles à un instant donné
- Utile quand le processus est plus grand que l'espace mémoire qui lui est alloué
- Implementé par l'utilisateur, pas de support spécial de l'OS => Conception des programmes devient plus complexe

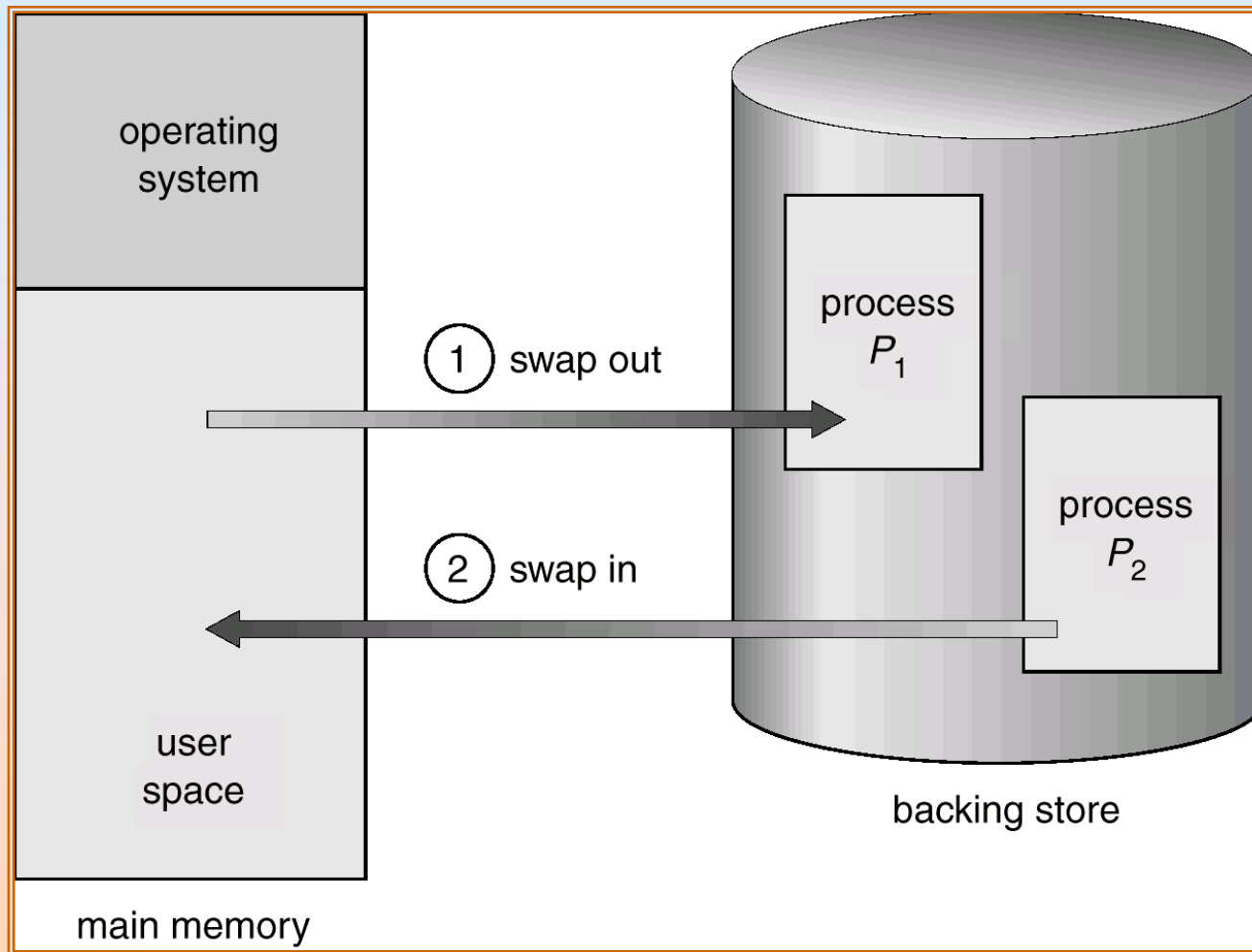
# Recouvrements pour Assembleur à Deux Passes



# Swapping

- Un processus peut être *swappé* temporairement en dehors de la mémoire vers un stockage secondaire, et puis remis en mémoire pour continuer son exécution
- **Stockage secondaire** – Disque rapide d'accès/transfert et assez large pour accommoder les copies de toutes les images mémoires pour tous les utilisateurs
- **Roll out, roll in** – variante du *swapping* utilisée dans les ordonnancements à base de priorité; un processus à plus basse priorité est *swappé* en dehors de façon à laisser sa place pour un de plus haute priorité
- Majeure partie du temps de *swap* est le transfert; temps de transfert total est directement proportionnel à la taille mémoire *swappée*
- Versions modifiées du *swapping* se trouvent sur les différents systèmes (i.e., UNIX, Linux, and Windows)

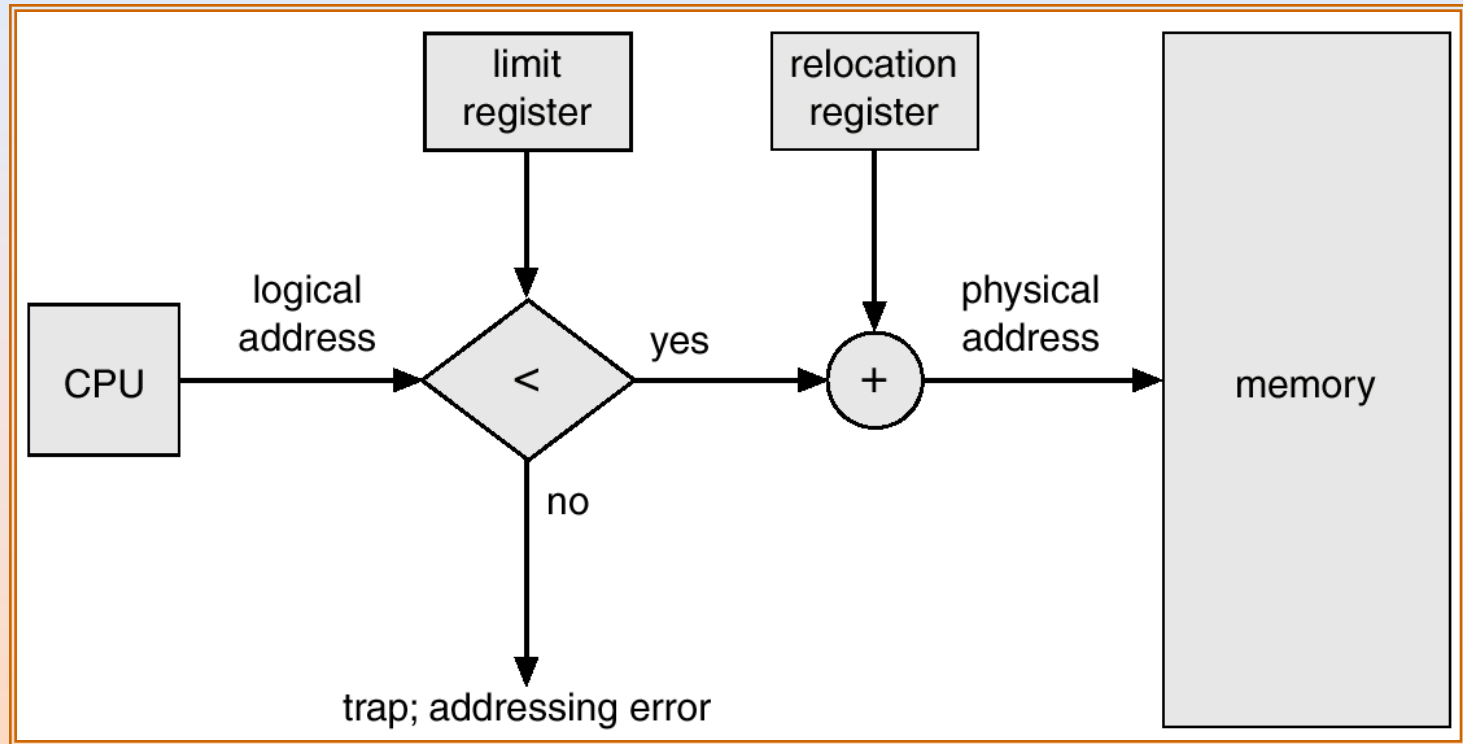
# Vue Schématique du Swapping



# Allocation Contigue

- En général, mémoire principale coupée en deux:
  - OS résident, en général dans la partie inférieure de la mémoire avec le vecteur d'interruptions
  - Processus utilisateurs dans la partie supérieure de la mémoire
  
- Allocation à 1 partition
  - Registre de relocation utilisé pour protéger les processus utilisateurs entre eux, et le code et les données OS
  - Registre de relogement contient la valeur de l'adresse physique la plus petite; le registre de limite contient l'étendue des adresses logiques possibles – chaque adresse logique doit être plus petite que la valeur du registre limite

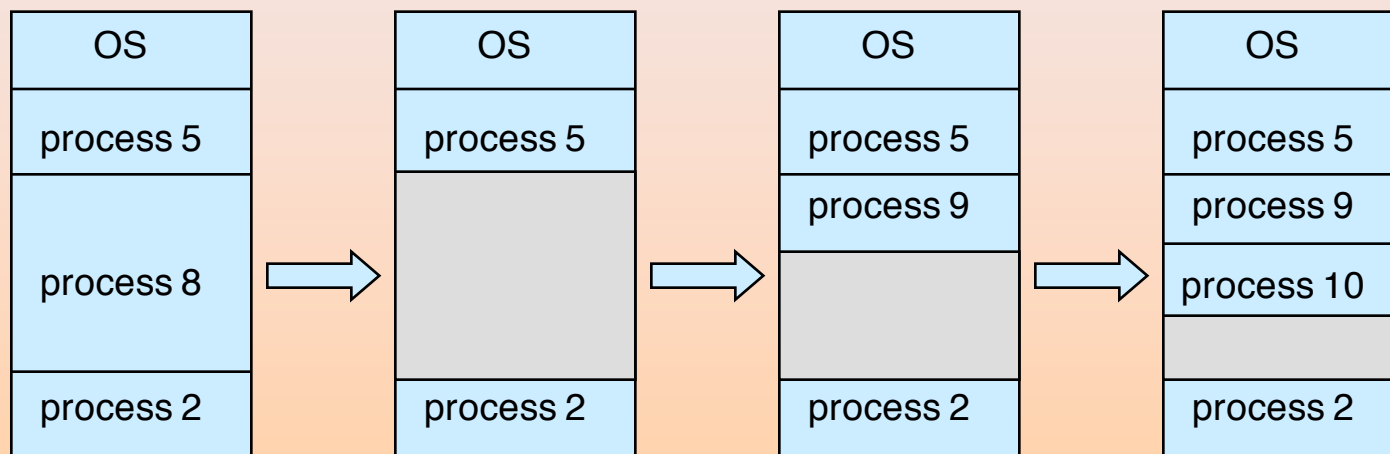
# Support Matériel : Registres de Relogement et Limite



# Allocation Contigue (Cont.)

## ■ Allocation à plusieurs partitions

- *Trous* – bloc de mémoire disponible; trous de taille variable dispersés dans la mémoire physique
- Quand un processus arrive, on lui alloue de la mémoire dans un trou assez large pour accomoder ses besoins
- OS retient de l'information sur:  
a) blocs alloués    b) blocs libres (trous)





# Problème de l'Allocation Dynamique

Comment satisfaire une requête de taille  $n$  à partir d'une liste de blocs

- **First-fit**: Allouer le premier bloc assez grand
- **Best-fit**: Allouer le plus petit bloc assez grand; doit parcourir la liste entière de blocs, sauf si ordonnée par taille. Produit des restes de blocs les plus petits.
- **Worst-fit**: Allouer le plus grand bloc; doit aussi rechercher dans toute la liste, sauf si ordonnée. Produit des restes de blocs les plus larges.

First-fit et best-fit sont meilleurs que worst-fit en termes de rapidité et utilisation de la mémoire

# Fragmentation

- **Fragmentation Externe** – somme des blocs peut satisfaire une requête, mais non contigus
- **Fragmentation Interne** – mémoire allouée peut-être légèrement plus large que la mémoire demandée; cette différence de taille est une mémoire interne à un bloc alloué, mais non utilisée
- Réduire la fragmentation externe par du **compactage**
  - Réorganiser la mémoire pour mettre les blocs ensemble pour en faire 1 grand bloc
  - Compactage possible *seulement* si relocation est dynamique, et se fait à l'exécution
  - Problème d'E/S
    - ▶ Processus en mémoire exécutant un appel système d'E/S
    - ▶ Faire les E/S seulement vers des tampons OS

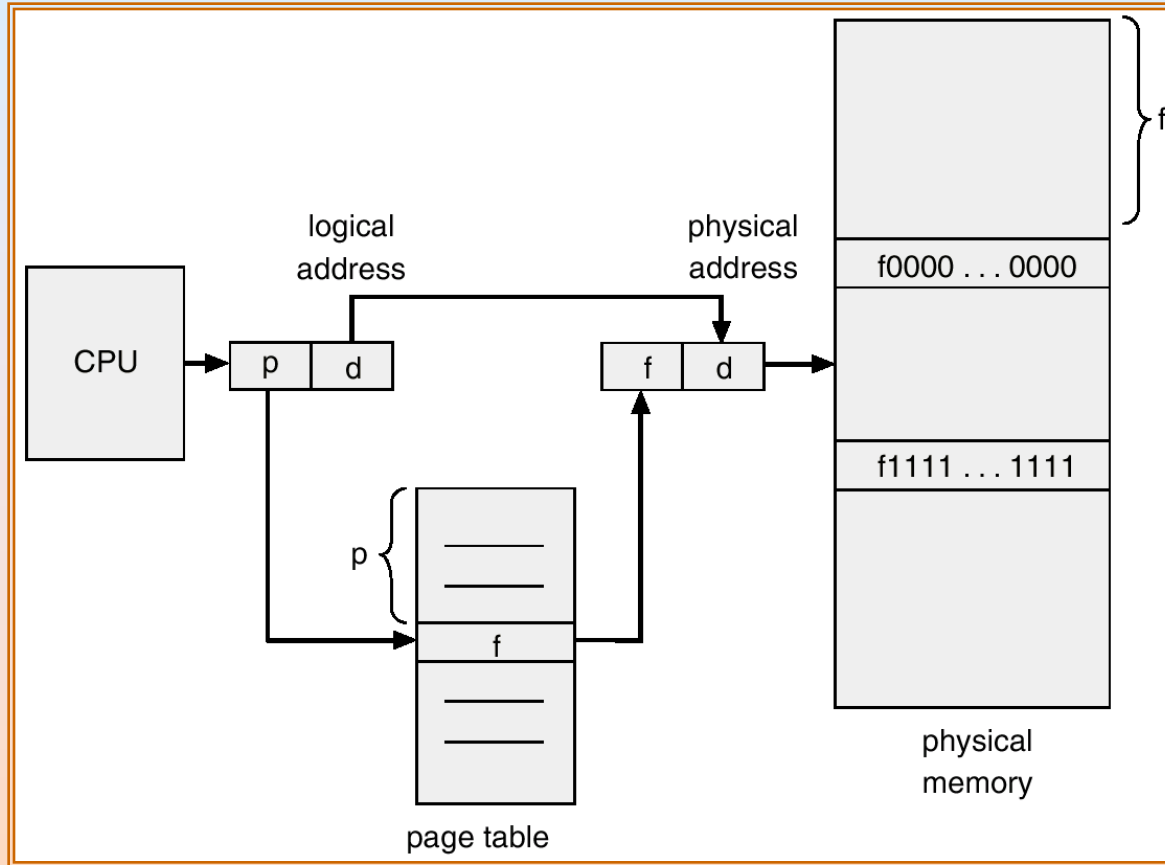
# Pagination

- Espace d'adressage logique peut être non contigu; on alloue de la mémoire physique à un processus quand la mémoire devient disponible
- Diviser la mémoire physique en blocs de tailles fixes appelés **cadres de page** (taille est puissance de 2, entre 512 octets et 16 Mo)
- Diviser la mémoire logique en blocs de même taille appelés **pages**.
- Gérer la liste des **cadres de page** libres
- Pour exécuter un programme de  $n$  **pages**, l'OS a besoin de trouver  $n$  **cadres de page** libres et charger le programme
- Gérer une **table des pages** pour traduire les adresses logiques en adresses physiques
- Fragmentation interne

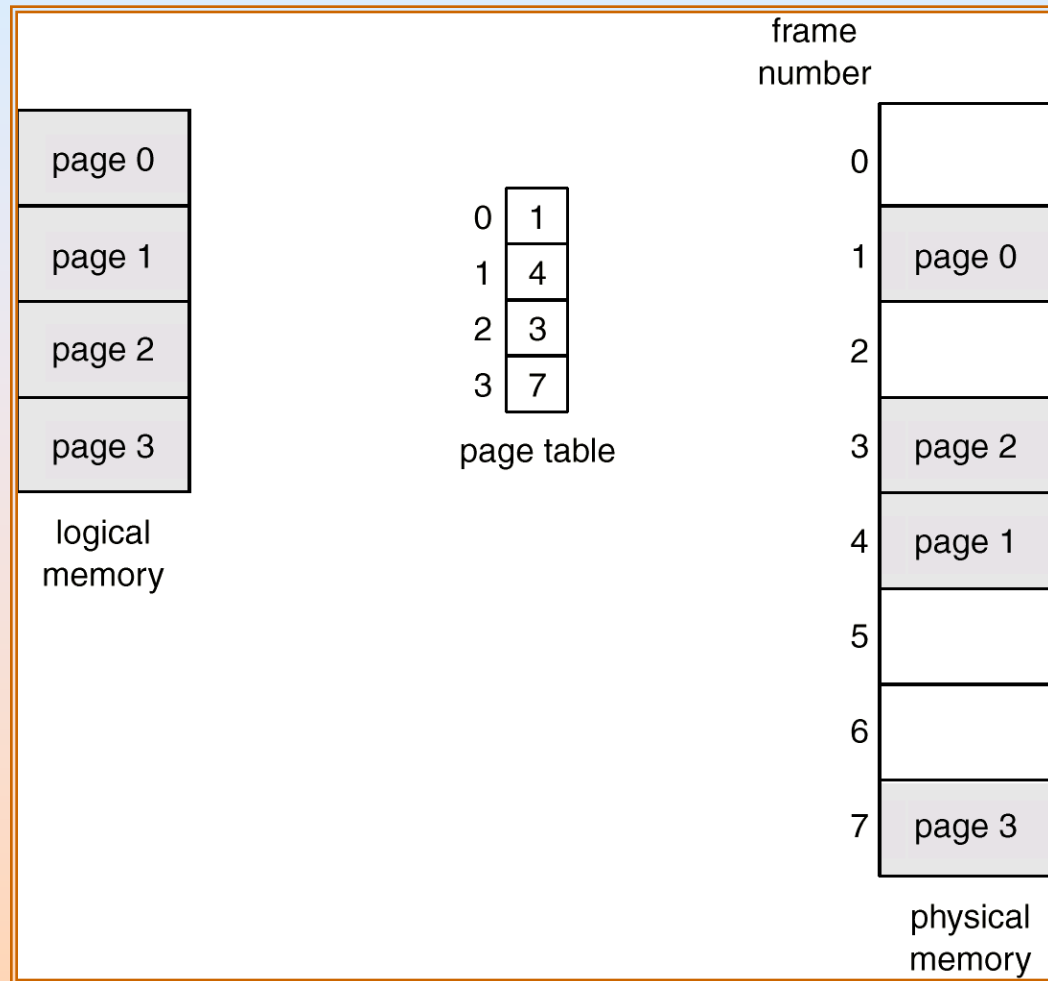
# Translation d'Adresses

- Adresse générée par la CPU est divisée en :
  - *Numéro de Page (p)* – utilisé comme indice dans la table des pages qui contient dans chaque entrée l'adresse physique en mémoire de base de la page correspondante
  - *Déplacement dans la Page (d)* – combiné avec l'adresse de base pour définir l'adresse physique complète envoyée à la mémoire

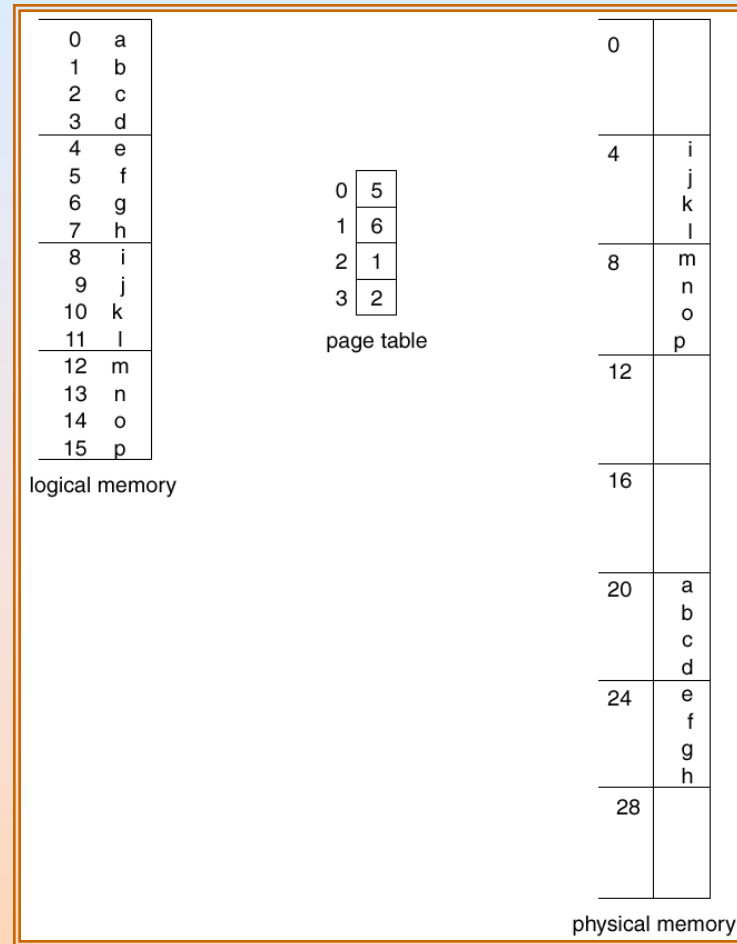
# Translation d'Adresses (cont.)



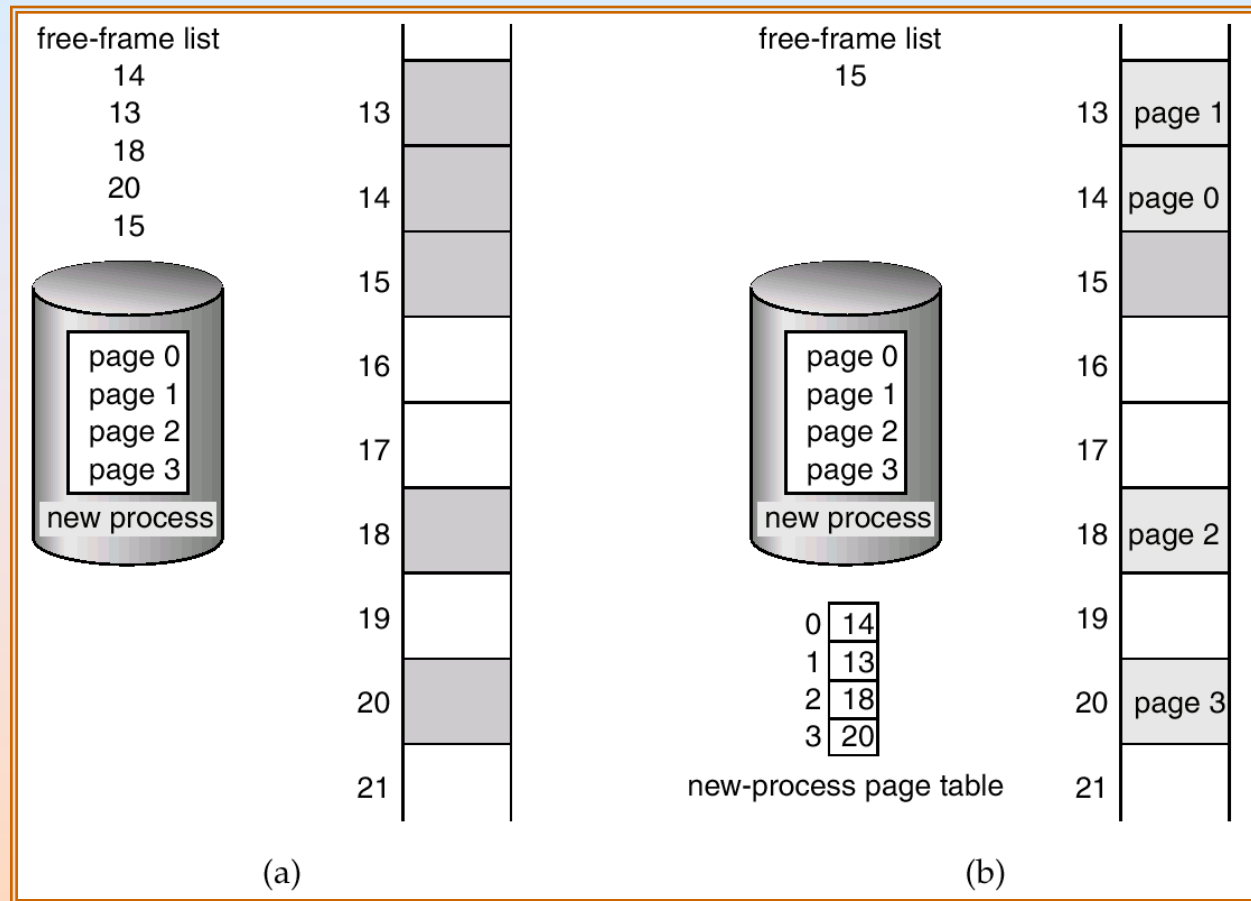
# Exemple de Pagination



# Exemple de Pagination



# Cadres de page Libres



Before allocation

After allocation



# Implémentation des Tables des Pages

- **Table de pages** en mémoire centrale
- *Registre de base de la table des pages (PTBR)* pointe sur la table de pages
- *Registre de longueur de la table des pages (PRLR)* indique la taille de la table des pages
- Dans ce schéma, chaque accès donnée/instruction requiert 2 accès mémoire; (1) accès à la table des pages, et (2) accès en mémoire à la donnée/instruction.
- Le problème de l'accès en plus peut être résolu par l'utilisation d'un cache de translation d'adresses appelé **translation look-aside buffer (TLB)**

# TLB

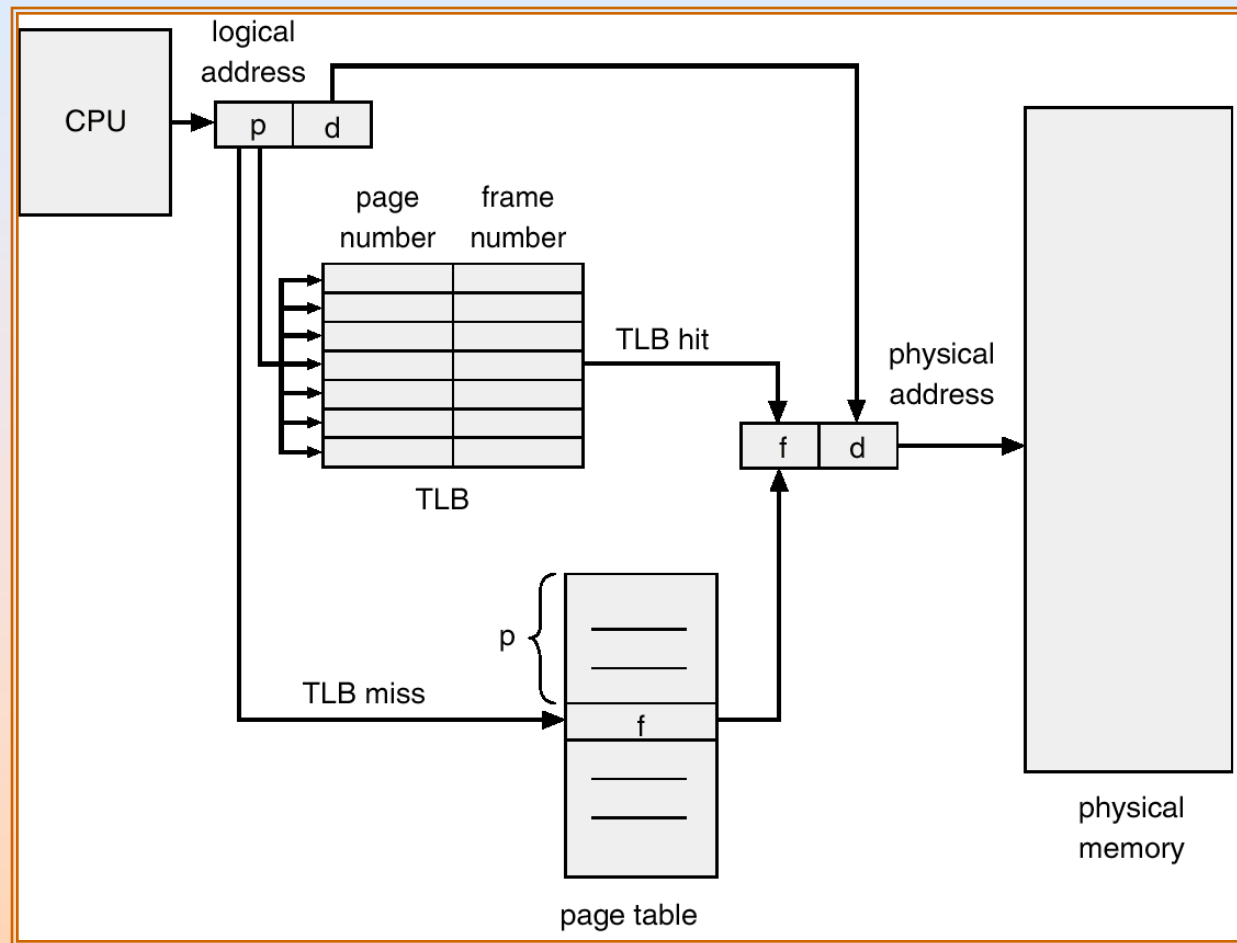
## ■ Mémoire associative – recherche en parallèle

Page #	Cadre de page #

Translation d'adresse ( $A'$ ,  $A''$ )

- Si  $A'$  est dans la TLB, lire l'entrée associée ( $A''$ ) correspondant au # de **cadre de page** en mémoire centrale
- Sinon, chercher le # de **cadre de page** dans la **table des pages**

# Matériel pour Pagination avec TLB



# Temps d'Accès Effectif

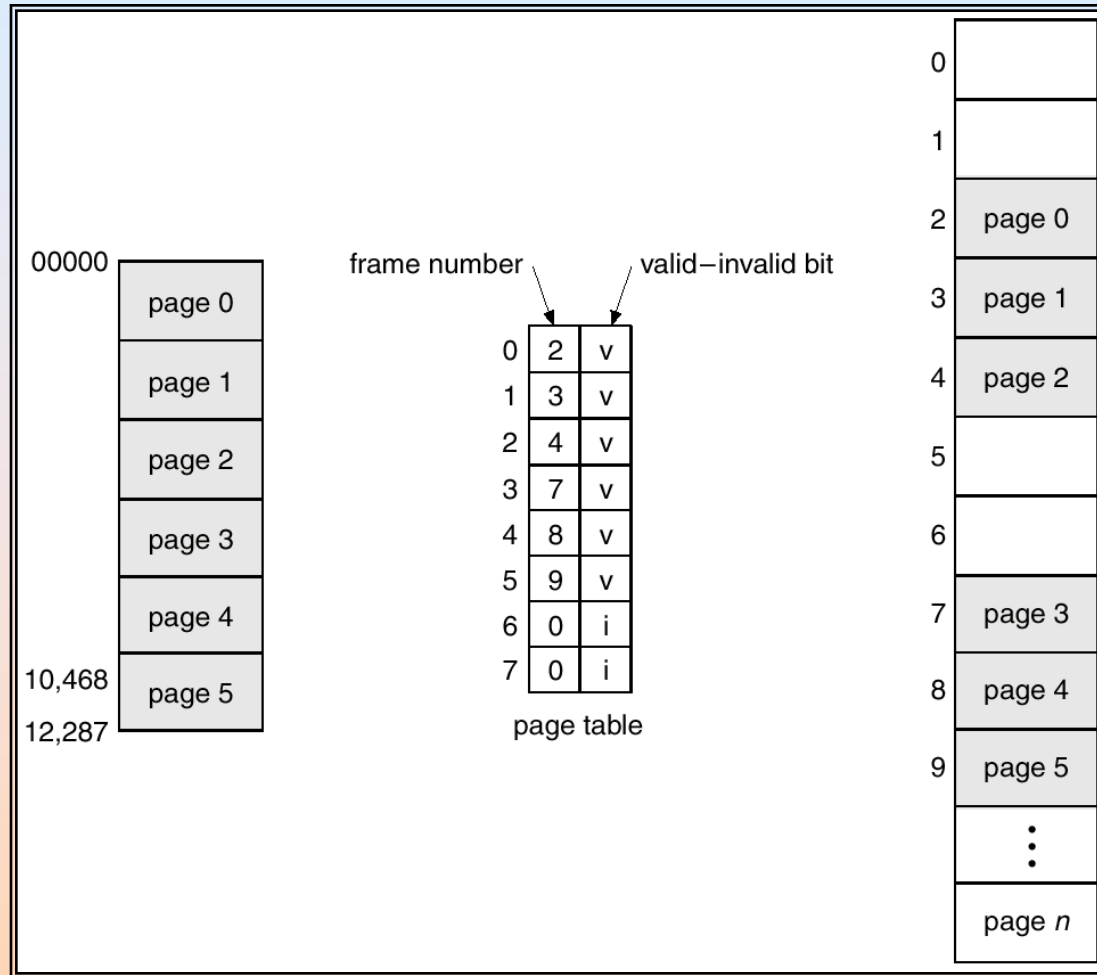
- Recherche associative =  $\frac{1}{H}$  unités de temps
- Supposons que le temps de cycle mémoire est de 1 microsecond
- Ratio Hit – pourcentage de fois qu'un numéro de page est trouvé dans la TLB; en rapport avec le nombre d'entrées de la TLB
- Ratio Hit =  $H$
- **Temps d'Accès Effectif (TAE)**

$$\begin{aligned} \text{TAE} &= (1 + \frac{1}{H}) H + (2 + \frac{1}{H})(1 - H) \\ &= 2 + \frac{1}{H} - H \end{aligned}$$

# Protection Mémoire

- Protection mémoire implémentée en associant un bit de protection à chaque **cadre de page**
- **Valide-Invalide** : bit attaché à chaque entrée dans la **table des pages**:
  - “valide” indique que la page indiquée existe bel et bien dans l’espace d’adressage logique du processus
  - “invalide” indique que la page n’est pas dans l’espace d’adressage logique du processus

# Bit Valide (v) ou Invalide (i) dans la Table des Pages



# Structure de la Table des Pages

- Pagination Hiérarchique
- Table des Pages “Hashée”
- Table des Pages inversées

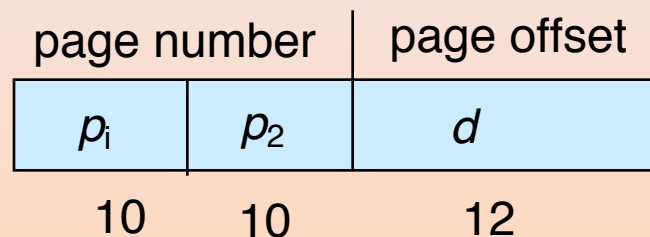
# Table des Pages Hiérarchique

- Décomposer l'espace d'adressage logique en plusieurs tables des pages
- Une simple décomposition : tables des pages à 2 niveaux



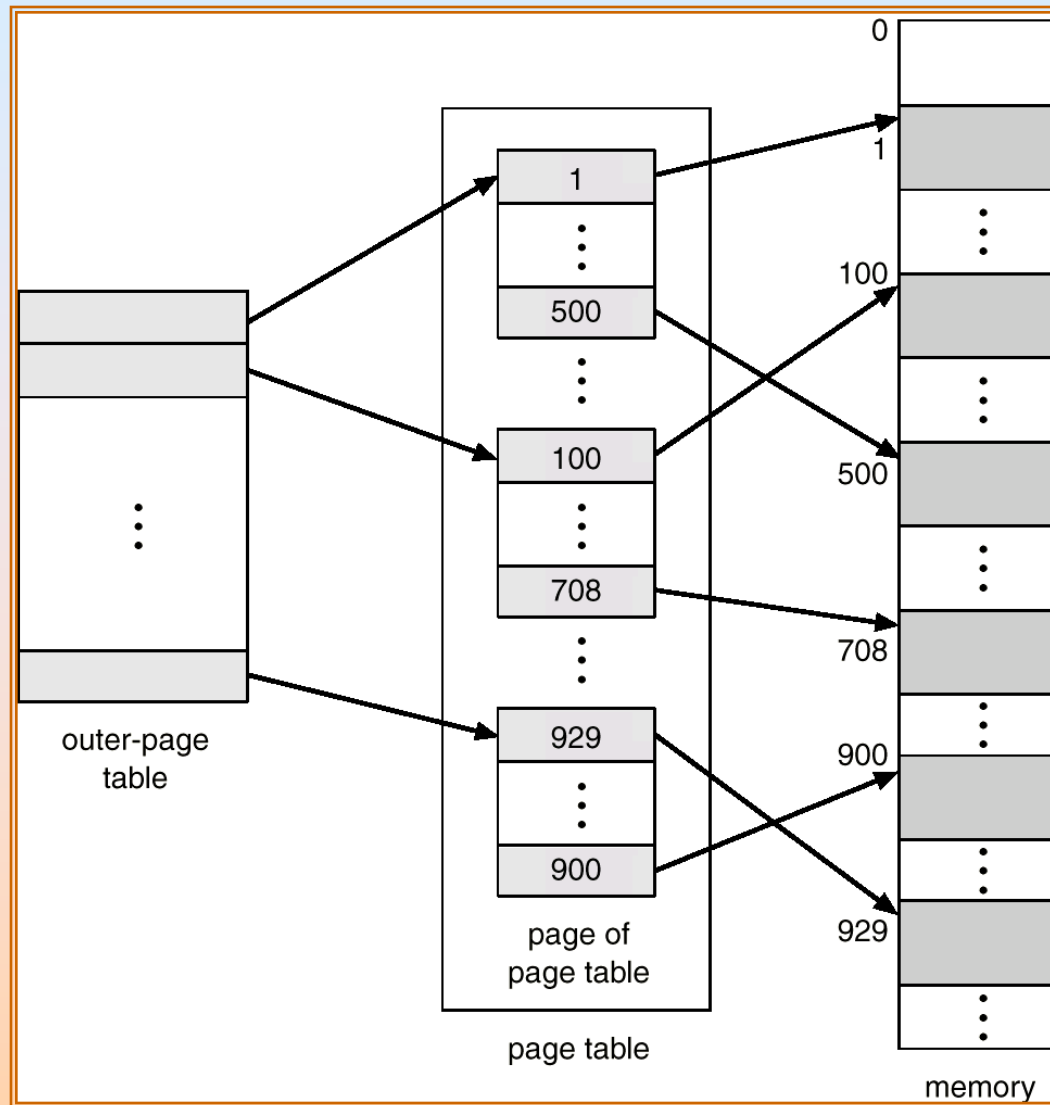
# Exemple: Pagination à 2 Niveaux

- Une adresse logique (sur 32-bit machine avec des pages de taille 4K) est décomposée en:
  - Un numéro de page de 20 bits
  - Un déplacement (offset) de 12 bits
- Comme la table des pages est paginée, le numéro de page est aussi décomposé en:
  - Un numéro de page de 10 bits
  - Un déplacement (offset) de 10 bits
- Ainsi, une adresse logique est de la sorte:



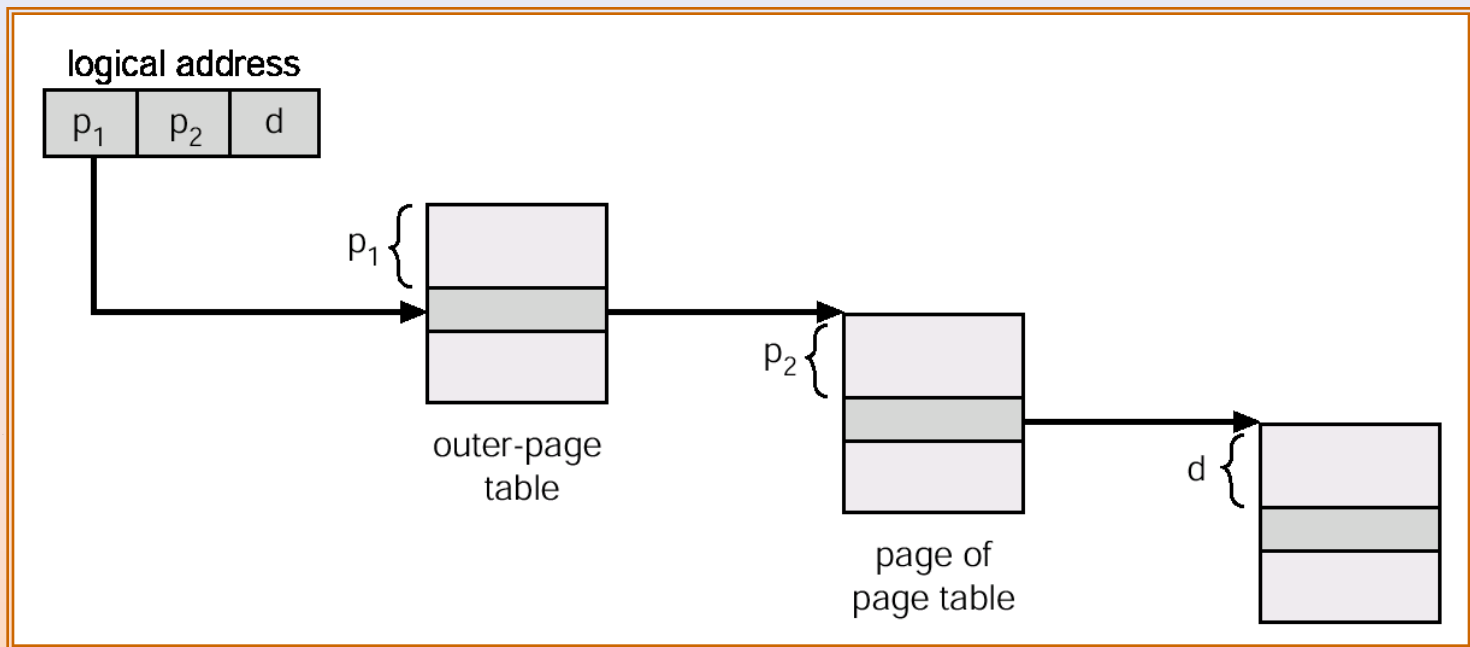
où  $p_1$  est un indice de la 1<sup>ère</sup> table des pages pointant sur une page contenant la 2<sup>ème</sup> table des pages, et  $p_2$  un indice de cette dernière

# Tables des Pages à 2 Niveaux



# Translation d'Adresses

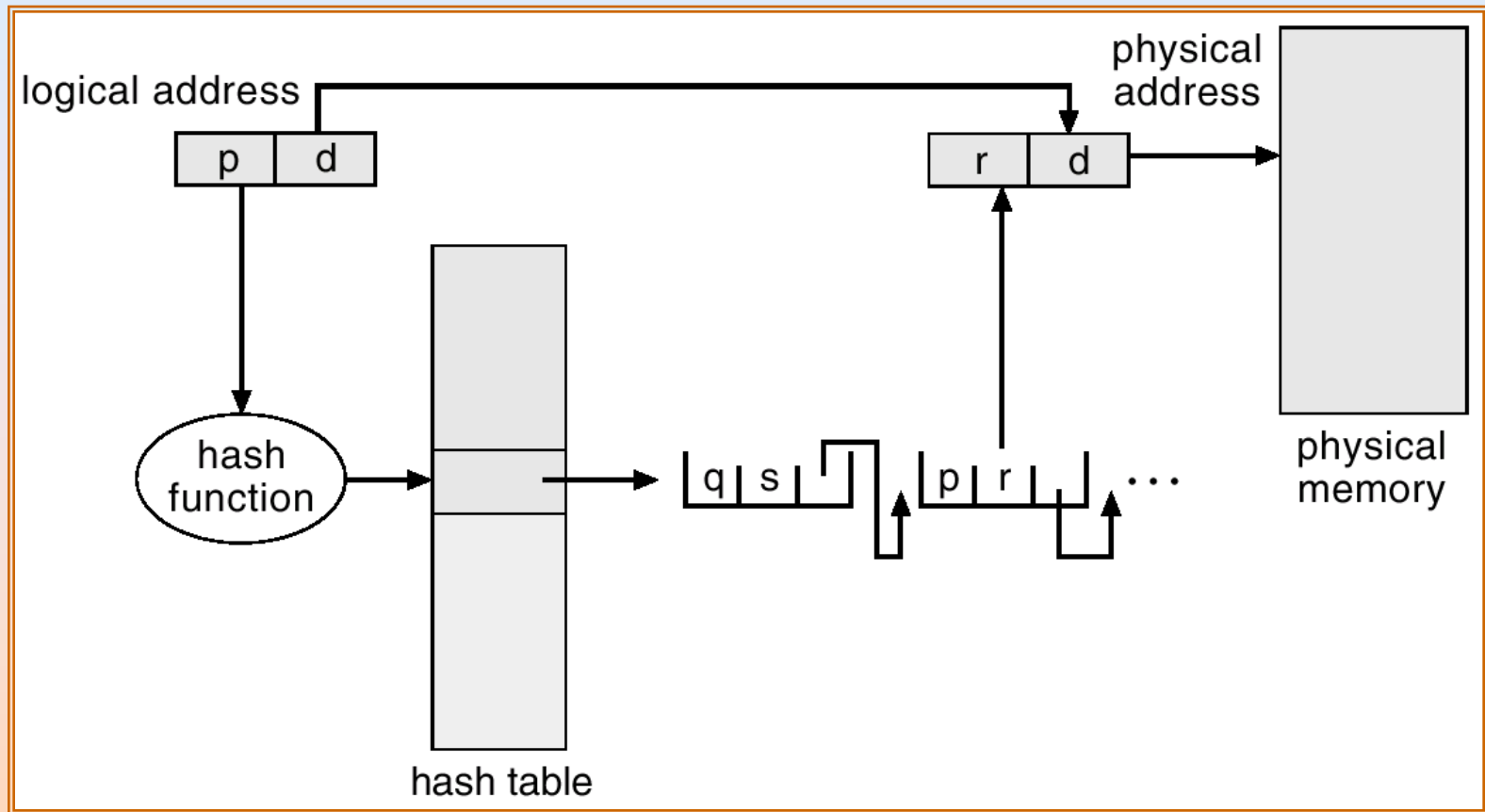
- Translation d'adresses pour une architecture à 32 bits avec pagination à 2 niveaux



# Table des Pages “Hashées”

- Utilisée dans des espaces d’adressage  $> 32$  bits
- Le numéro de page virtuelle est hashé et utilisé comme indice dans la table des pages; chaque entrée de la table des pages contient une séquence d’éléments
- Les numéros de pages virtuelles sont comparés dans cette séquence recherchant une équivalence avec le numéro de page virtuelle. Si trouvée, le numéro de cadre de page physique est extrait.

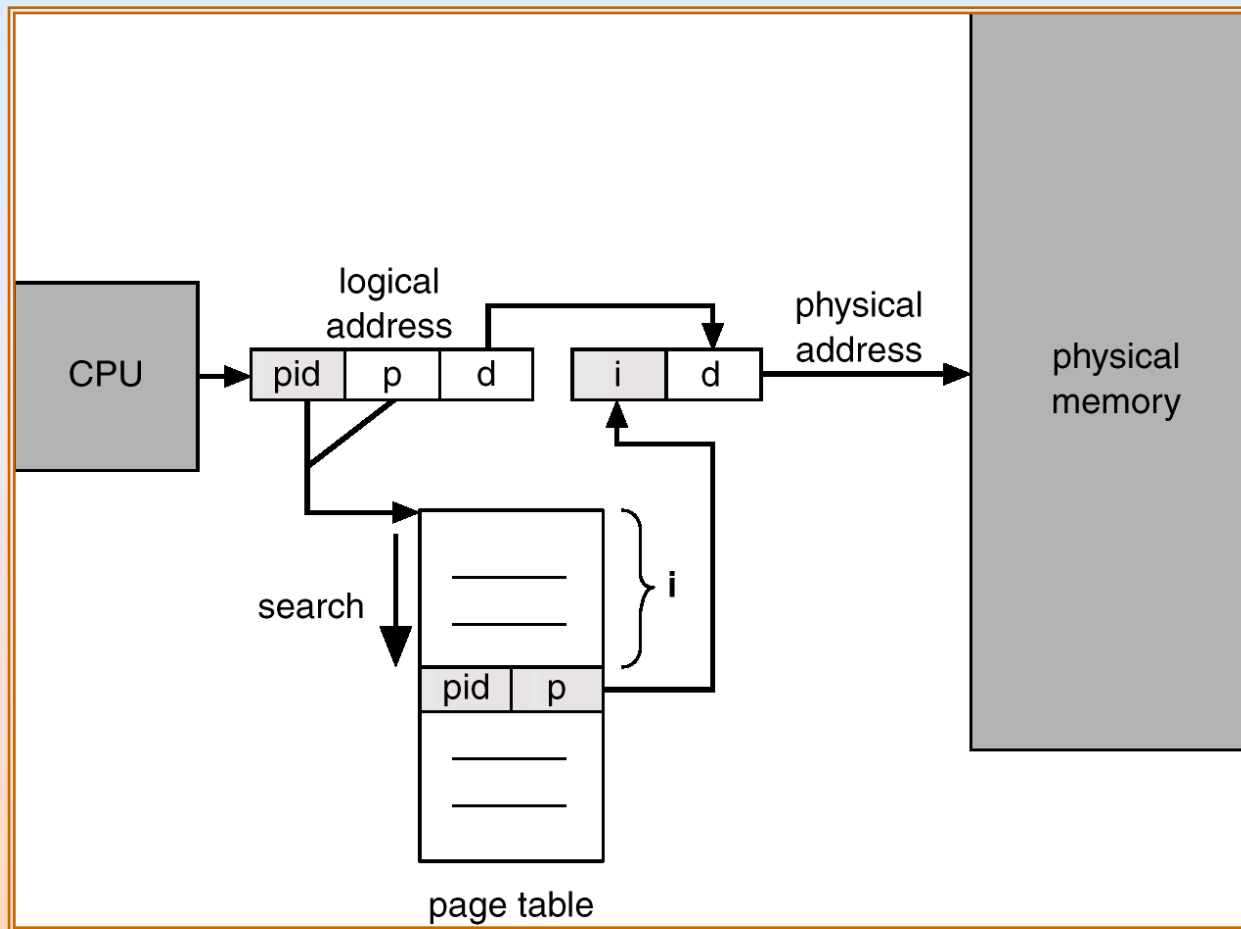
# Table des Pages Hashée



# Table des Pages Inversée

- Une entrée pour chaque page mémoire
- Une entrée contient l'adresse virtuelle correspondante à cette page associée au processus qui la détient
- Décroît l'espace mémoire nécessaire pour stocker chaque table des pages de tous les processus, mais accroît le temps nécessaire pour la translation d'adresses
- Utiliser une table de hashage pour limiter le temps de recherche (retrouver en 1 ou quelques essais)

# Table des Pages Inversée



# Pages Partagées

## ■ Code partagé

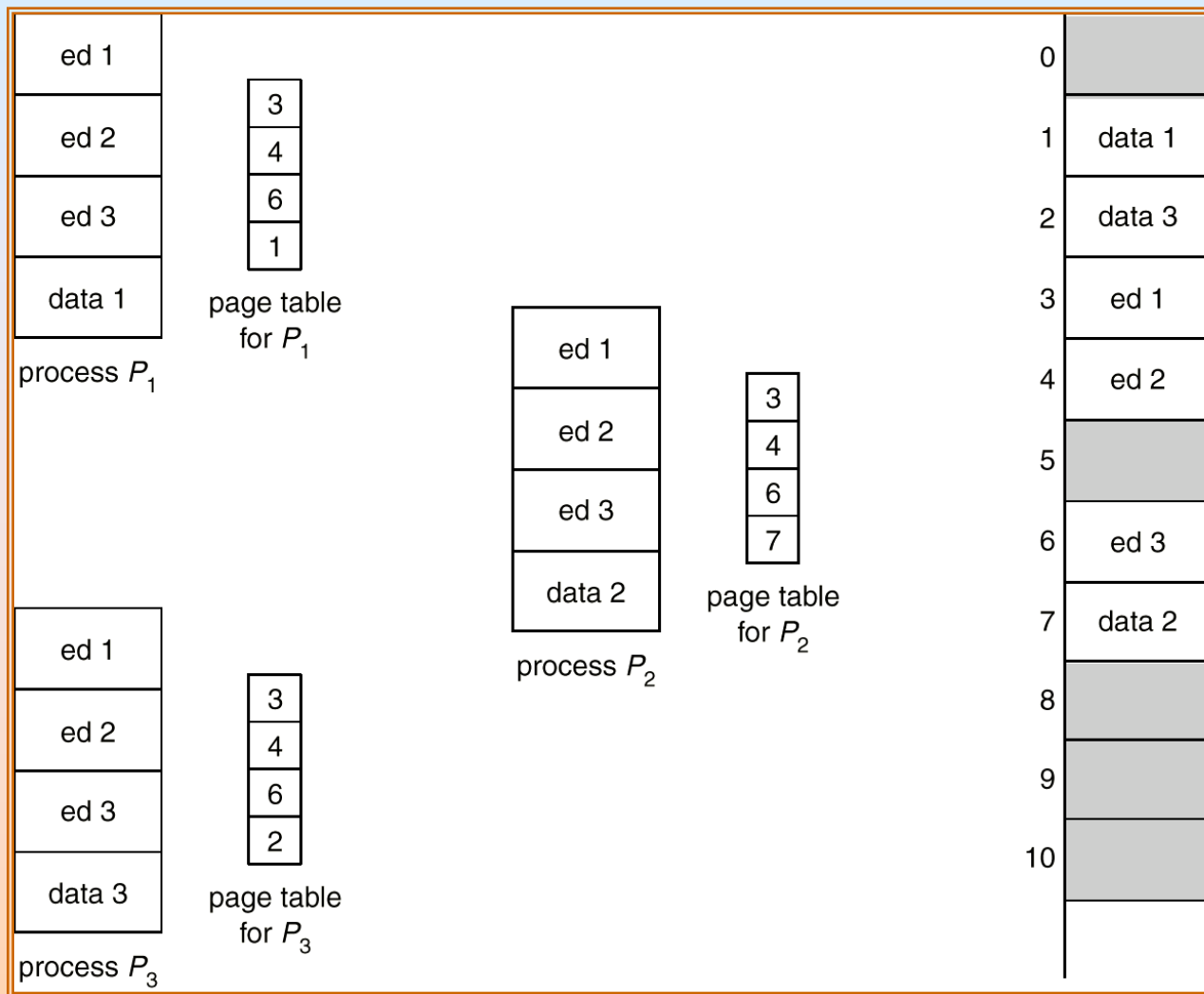
- Une copie en lecture seulement (réentrant), code partagé entre processus (i.e., text editors, compilers, window systems).
- Un code partagé doit apparaître au même endroit dans l'espace d'adressage logique de tous les processus

## ■ Code et données privés

- Chaque processus a une copie séparée du code et des données
- Les pages pour le code et les données peuvent apparaître à des endroits différents dans les espaces d'adressage locaux des différents processus



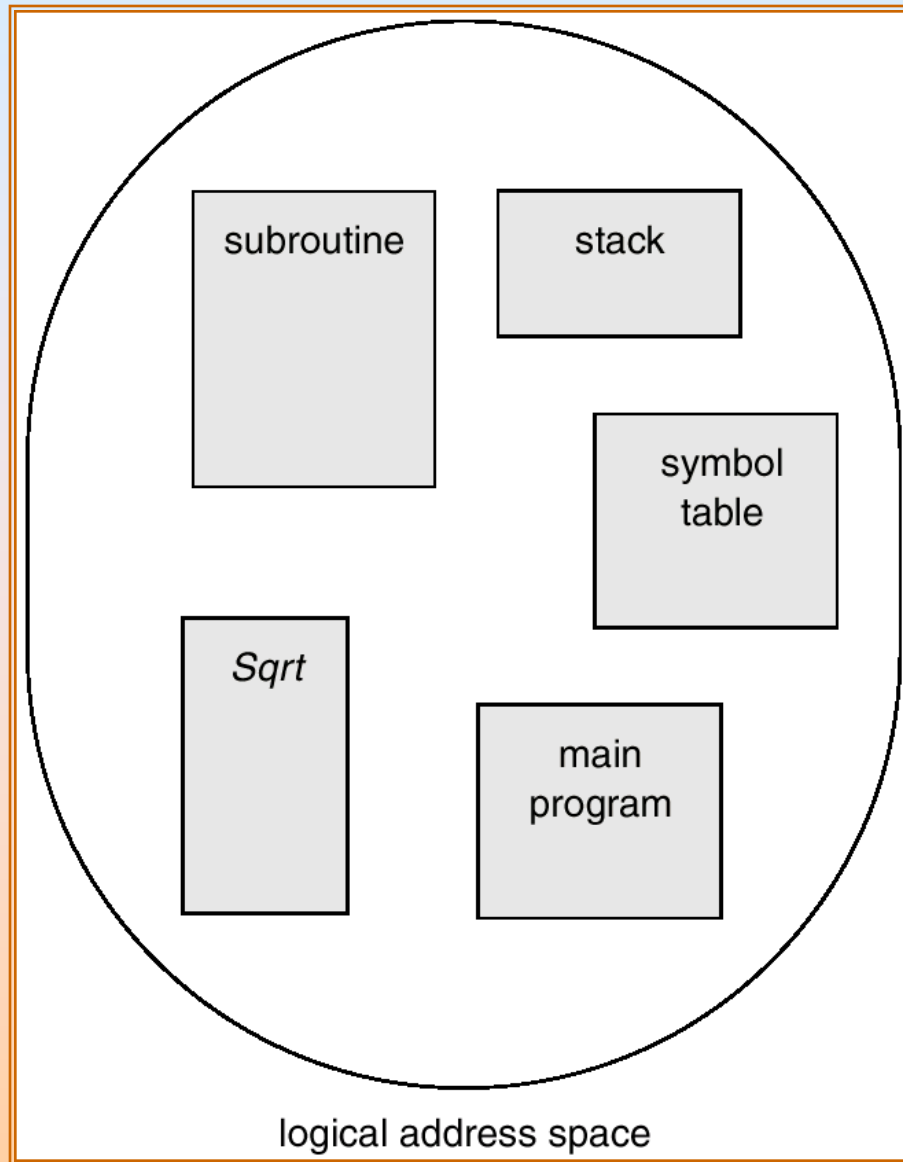
# Exemple de Pages Partagées



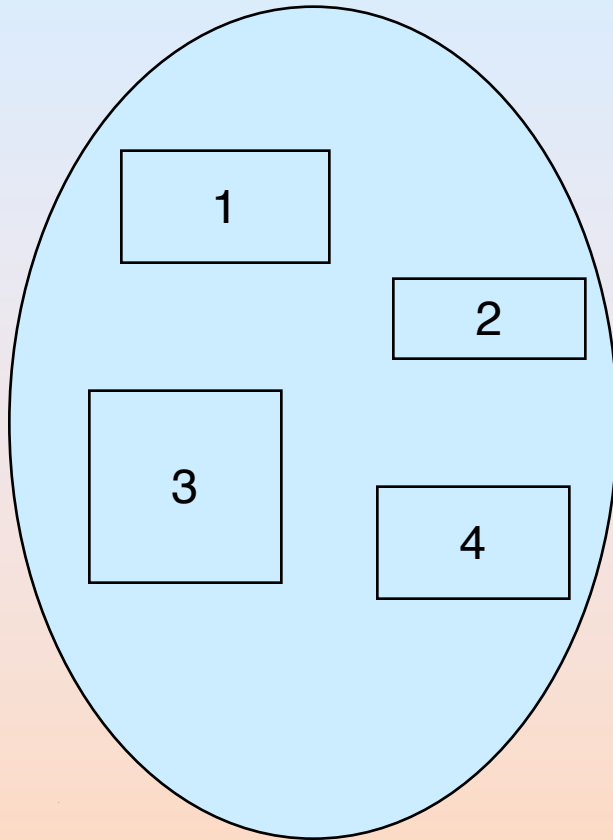
# Ségmentation

- Gestion de la mémoire qui supporte une vue utilisateur de la mémoire
- Un programme est une collection de ségments. Un segment est une unité logique telle que:
  - programme principal,
  - procédure,
  - fonction,
  - méthode,
  - objet,
  - variables locales, variables globales,
  - bloc commun,
  - pile,
  - table des symboles, tableaux

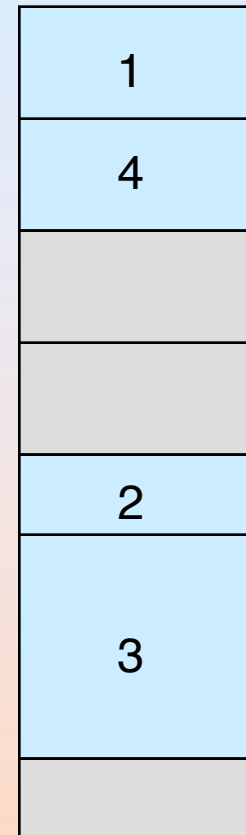
# Vue Utilisateur d'un Programme



# Vue Logique de la Ségmentation



Espace Utilisateur



Espace Mémoire Physique

# Architecture Ségmentation

- Adresse logique contient le couple:  
    <numéro-ségment, déplacement>,
- **Table des Ségments** – translation entre adresse logique et adresse physique; chaque entrée contient:
  - base – contient l'adresse physique de début du ségment en mémoire
  - *limite* – spécifie la longueur du segment
- *Segment-table base register (STBR)* pointe sur la table des ségments en mémoire
- *Segment-table length register (STLR)* indique le nombre de ségments utilisés par un processus;  
    numéro ségment  $s$  est légal si  $s < \text{STLR}$

# Architecture Ségmentation (Cont.)

## ■ Relocation

- dynamique
- Via la table de ségments


## ■ Partage

- Ségments partagés
- Même numéro de ségment

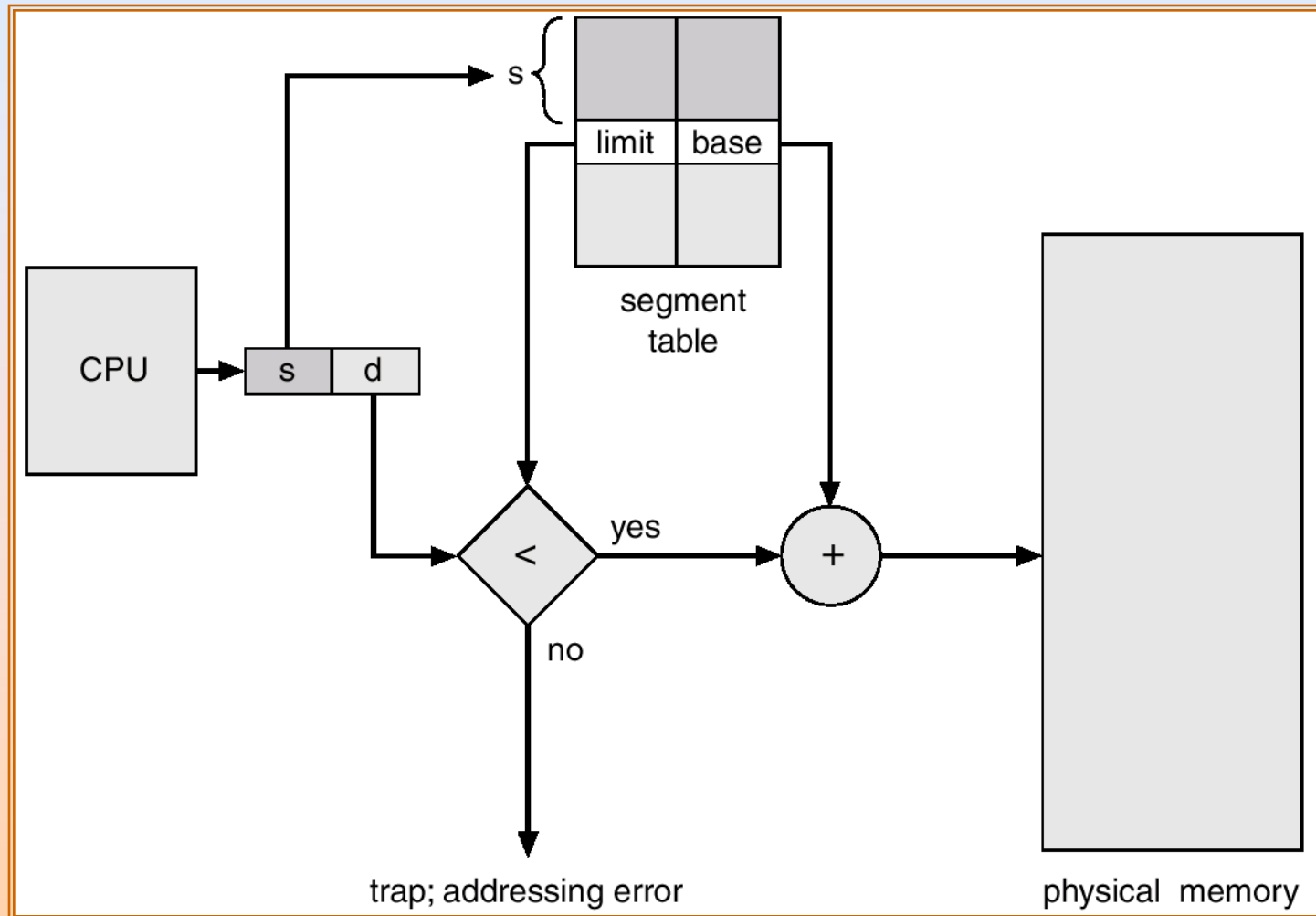
## ■ Allocation

- first fit/best fit
- Fragmentation externe

# Architecture Ségmentation (Cont.)

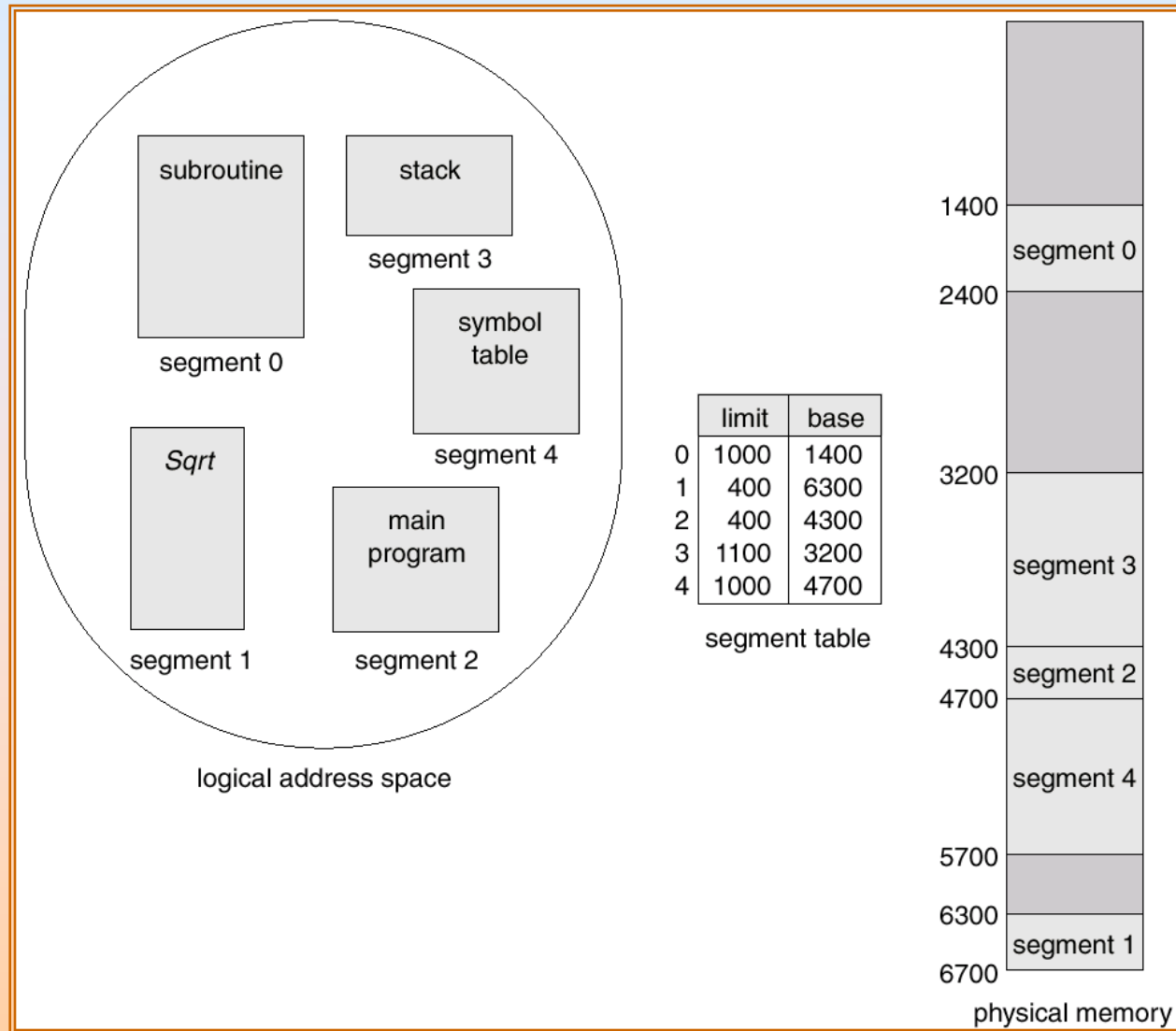
- **Protéction.** Est associé à chaque entrée dans la table des ségments:
  - Bit de validation = 0  illegal segment
  - Privilèges read/write/execute
- Bits de protection associés aux ségments; partage de code se fait au niveau des ségments
- Comme les ségments varient en taille, l'allocation mémoire est un problème d'allocation mémoire dynamique (vue en début de cours)

# Support Matériel pour Ségmentation

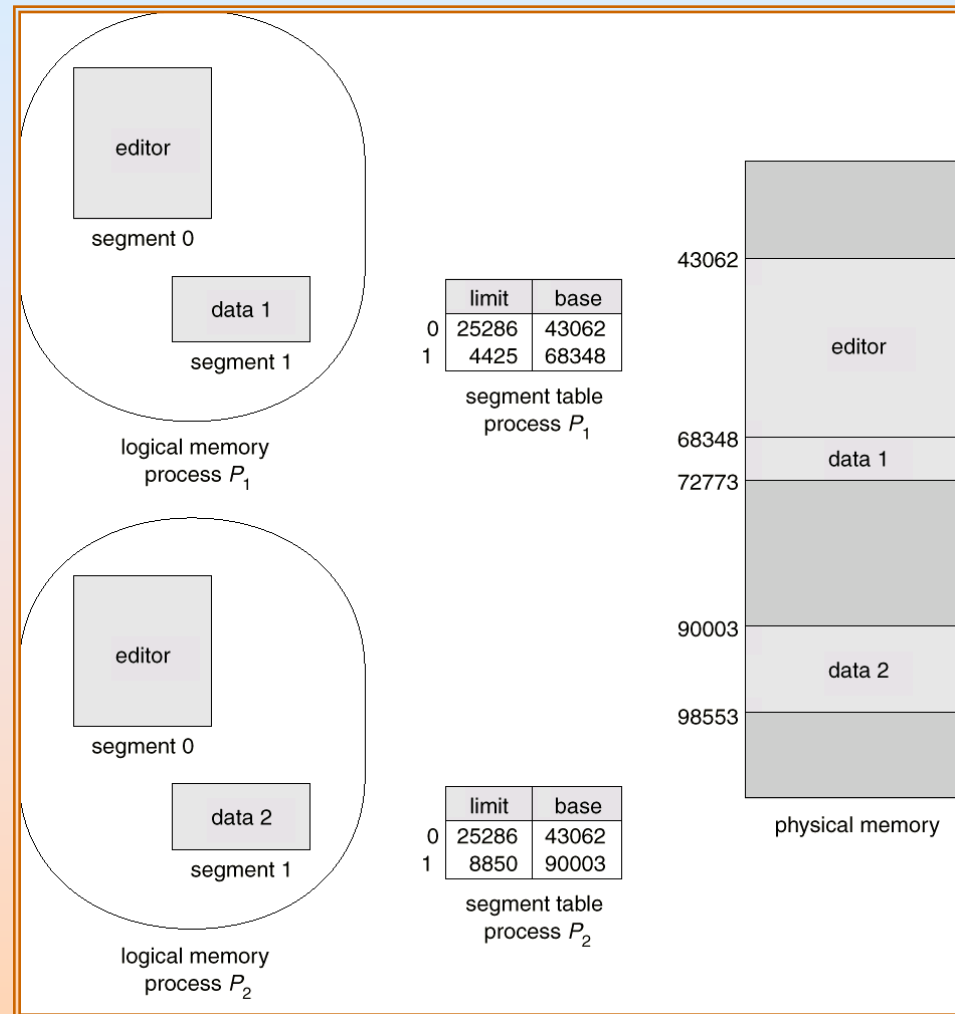




# Exemple de Ségmentation



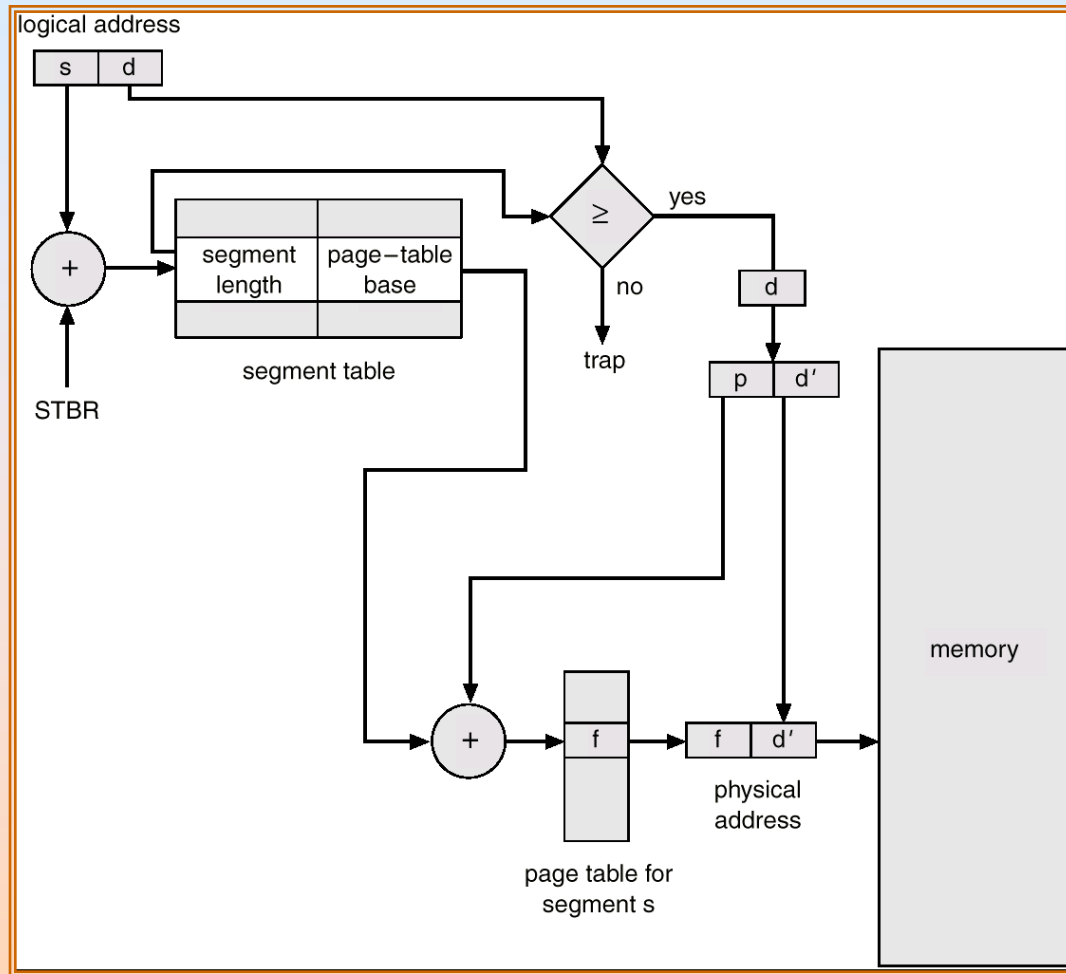
# Partage de Ségments



# Ségmentation avec Pagination – MULTICS

- Le système MULTICS a résolu des problèmes de fragmentation externe en paginant les ségments
- Solution diffère de la pure ségmentation; une entrée de la table des ségments contient l'adresse de base d'une table des pages pour ce ségment

# MULTICS: Translation d'Adresses



# Ségmentation paginée – Intel 386

- Comme montré dans le diagramme suivant, Intel 386 utilise la ségmentation paginée à 2 niveaux pour la gestion de la mémoire

# Intel 30386 : Translation d'Adresses

