Gestion Mémoire

Menu

- Introduction
- Swapping
- Gestion mémoire contiguë
- Segmentation
- Pagination

Menu

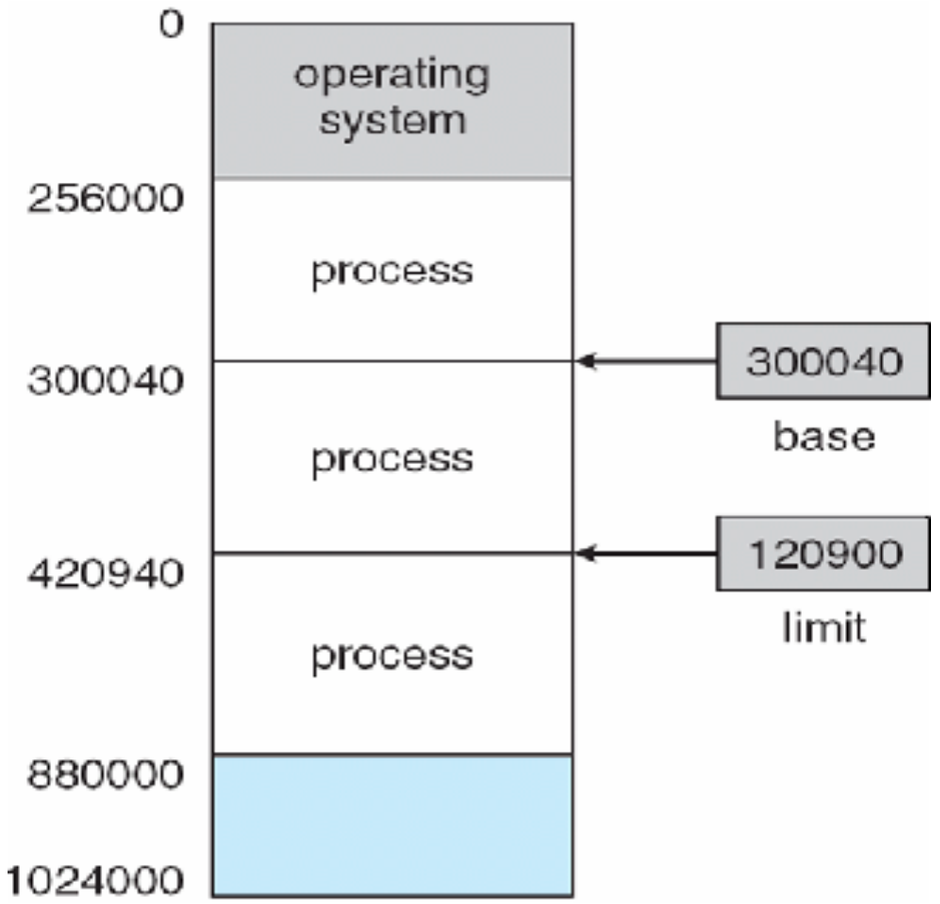
- **Introduction**
- Swapping
- Gestion mémoire contiguë
- Segmentation
- Pagination

Introduction

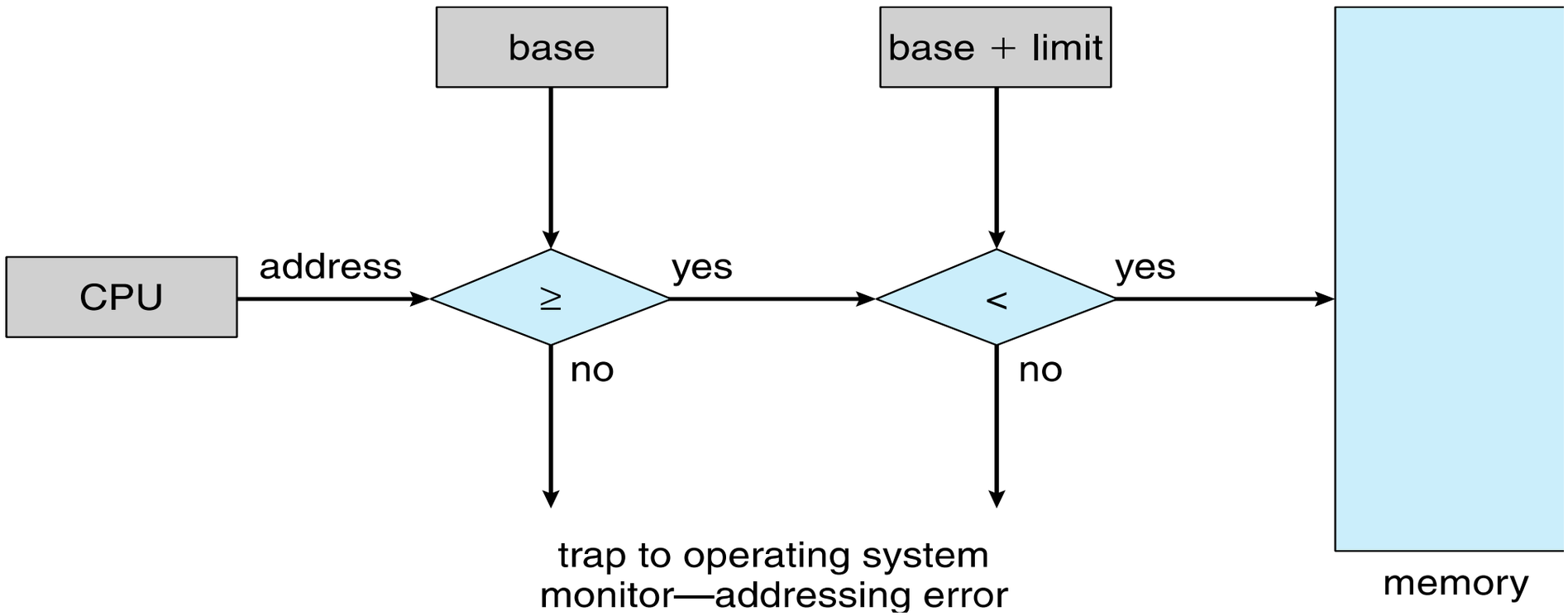
- Le programme doit être amené en mémoire et placé dans un processus pour être exécuté
- La mémoire centrale et les registres sont la seule stockage que le CPU peut accéder directement
- **Memory unit (MMU)** ne voit qu'un flux d'*adresses + demandes de lecture*, ou *adresse + données et demandes d'écriture*
- Le registre peut être consulté dans **une** CPU
- L'accès à la mémoire centrale peut prendre plusieurs cycles, provoquant un **stall**
- La mémoire cache se trouve entre la mémoire centrale et les registres du processeur
- Protection de la mémoire requise pour assurer le bon fonctionnement
  - entre les processus du SE et les utilisateurs, et entre les processus des utilisateurs
  - fourni par le support matériel, sinon trop coûteux pour impliquer le SE

Registres `Base` et `Limit`

- Chaque processus reçoit un morceau contigu de mémoire
  - Le noyau contrôle les registres `base` et `limit`
- CPU vérifie que chaque accès mémoire est entre `base` et `limit`
- SE peut accéder à n'importe quelle mémoire pour accomplir ses tâches



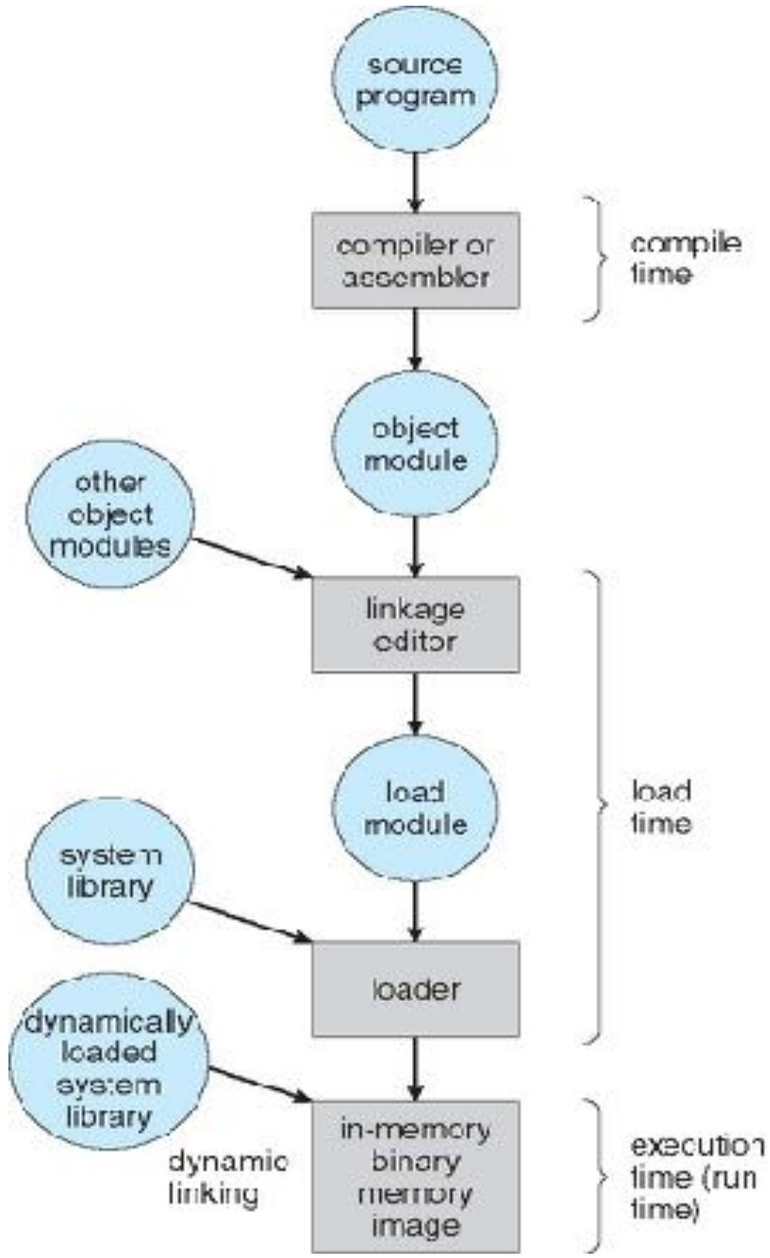
Protection d'adresse matérielle



Adresses

- Programmes stockés sur disque, prêt à être mis en mémoire pour s'exécuter
- Addresses à l'execution dépendent de `base`
- Différentes addresses à différentes étapes d'un programme:
  - Code source adresse généralement symbolique (identificateur)
  - Code compilé: adresses **relocatable**
    - ✓ i.e. "14 bytes après le début de ce module"
  - Le **linker** combine des modules en un programme
    - ✓ i.e. "74014 bytes après le début du programme"
  - Le **loader** converti en adresse d'exécution
    - ✓ i.e. "adresse 75014" (si `base` vaut 1000)

Vie d'un programme



“Binding” des instructions et les données à la mémoire

- La “binding” d'adresse d'instructions et de données à des adresses de mémoire peut se produire à trois différentes étapes:
  - *Temps de compilation*: Si l’adresse est connu *a priori*, le code **absolu** peut être généré et *binding* est faite lors de la compilation (doit recompiler le code si l’adresse de départ change)
  - *Temps de chargement (load)*: le compilateur doit générer du code **relocatable** si l’adresse n'est pas connu au moment de la compilation, et la liaison est effectuée lors du chargement (doit recharger si l’adresse de départ change)
  - *Temps d'exécution*: “Binding” retardée jusqu'à l'exécution si le processus peut être déplacé pendant son exécution d'un segment de mémoire à un autre
    - Besoin d'un support matériel pour les cartes d'adresses (par exemple, registres de base et de limite)

Adresses logiques vs physiques

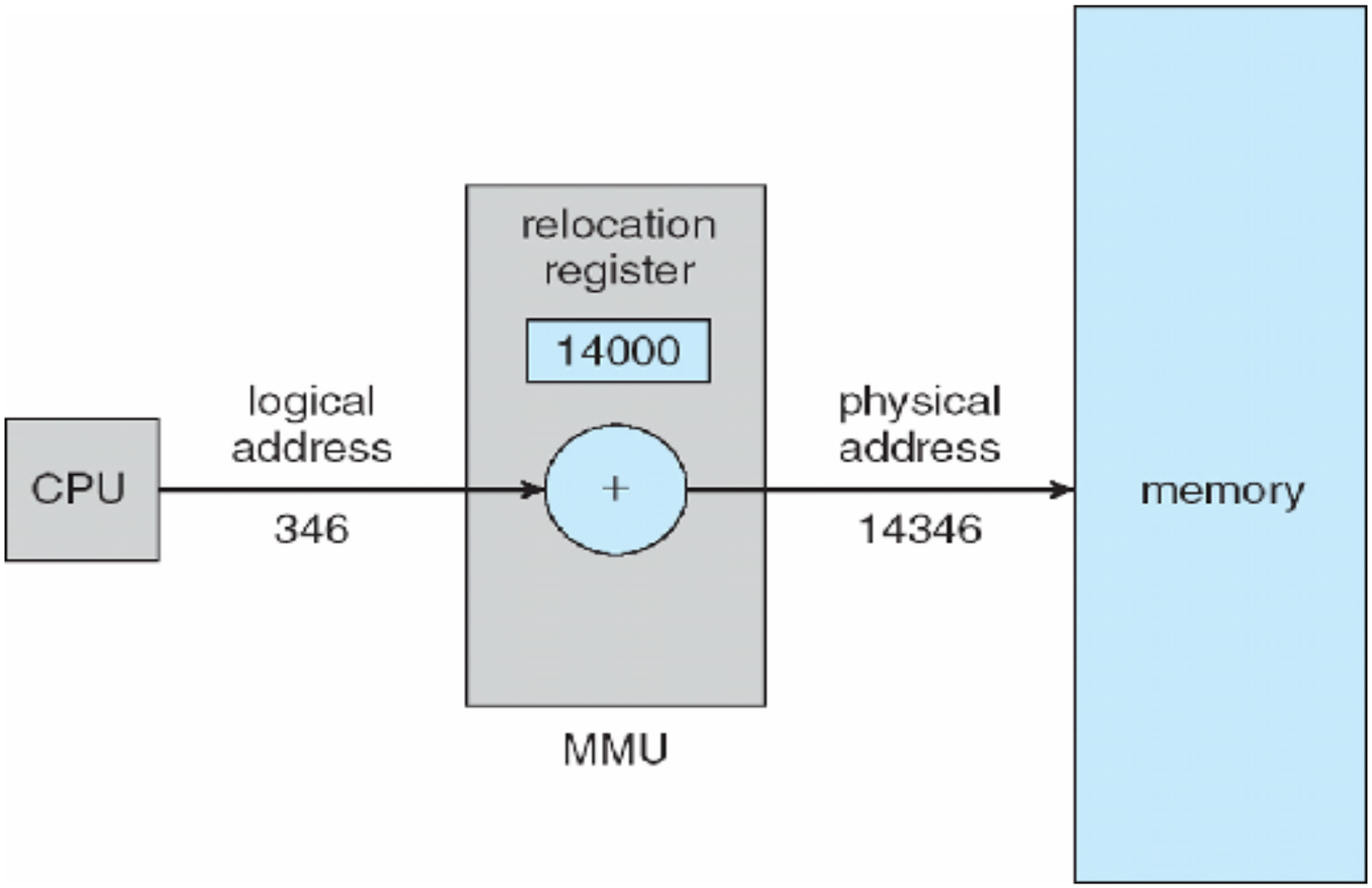
- Adresses *logiques (virtuelles)* – les adresses utiliser par les processus. Les processus n'ont pas besoin de savoir où ils sont placés
- Adresses *physiques* – le vrai adresse en mémoire

  - Logical address space -> toutes les adresses logiques
  - Physical address space -> ensemble des address physiques

Memory-Management Unit (MMU)

- Traduire les address *logiques* en adresses *physiques*
- Différentes techniques de traductions
- Compromis entre flexibilité et exécution “instantanée”
- Inclut généralement vérification de droits d’accès

Ex: registre de relocation



Loading dynamique

- La routine n'est pas chargée tant qu'elle n'est pas appelée
- Meilleure utilisation de l'espace mémoire routine inutilisée n'est jamais chargée
- Toutes les routines conservées sur le disque dans un format de chargement relocatable
- Lorsqu'elle est appelée, la routine est vérifiée si elle n'est pas en mémoire, et si c'est le cas, elle est chargée. Les tables d'adresses de programme sont mises à jour et le contrôle est transmis à la routine nouvellement chargée
- Utile lorsque de grandes quantités de code sont nécessaires pour traiter des cas peu fréquents
- Aucune assistance spéciale du système d'exploitation n'est requise
  - Mis en œuvre par la conception du programme
  - OS peut aider en fournissant des bibliothèques pour implémenter le chargement dynamique

Linkage dynamique

- **Static linking** - bibliothèques système et code de programme combinés par le chargeur dans l'image de programme binaire
  - certains systèmes d'exploitation ne prennent en charge que les liens statiques
- Liaison dynamique - liaison reportée jusqu'à l'exécution
- Petit morceau de code, **stub**, utilisé pour localiser la routine de bibliothèque résidant en mémoire appropriée
- Stub se remplace par l'adresse de la routine, et exécute la routine
- Le système d'exploitation vérifie si la routine est dans l'adresse mémoire des processus
  - Si ce n'est pas dans l'espace d'adressage, ajoutez l'espace d'adressage
- La linkage dynamique est particulièrement utile pour les bibliothèques
- Système également connu sous le nom de bibliothèques partagées
- Envisager l'applicabilité aux bibliothèques de correctifs
  - Le versionnement peut être nécessaire
- OS doit prendre en charge si les processus peuvent accéder à la bibliothèque partagée dans le même espace mémoire (plus tard ...)

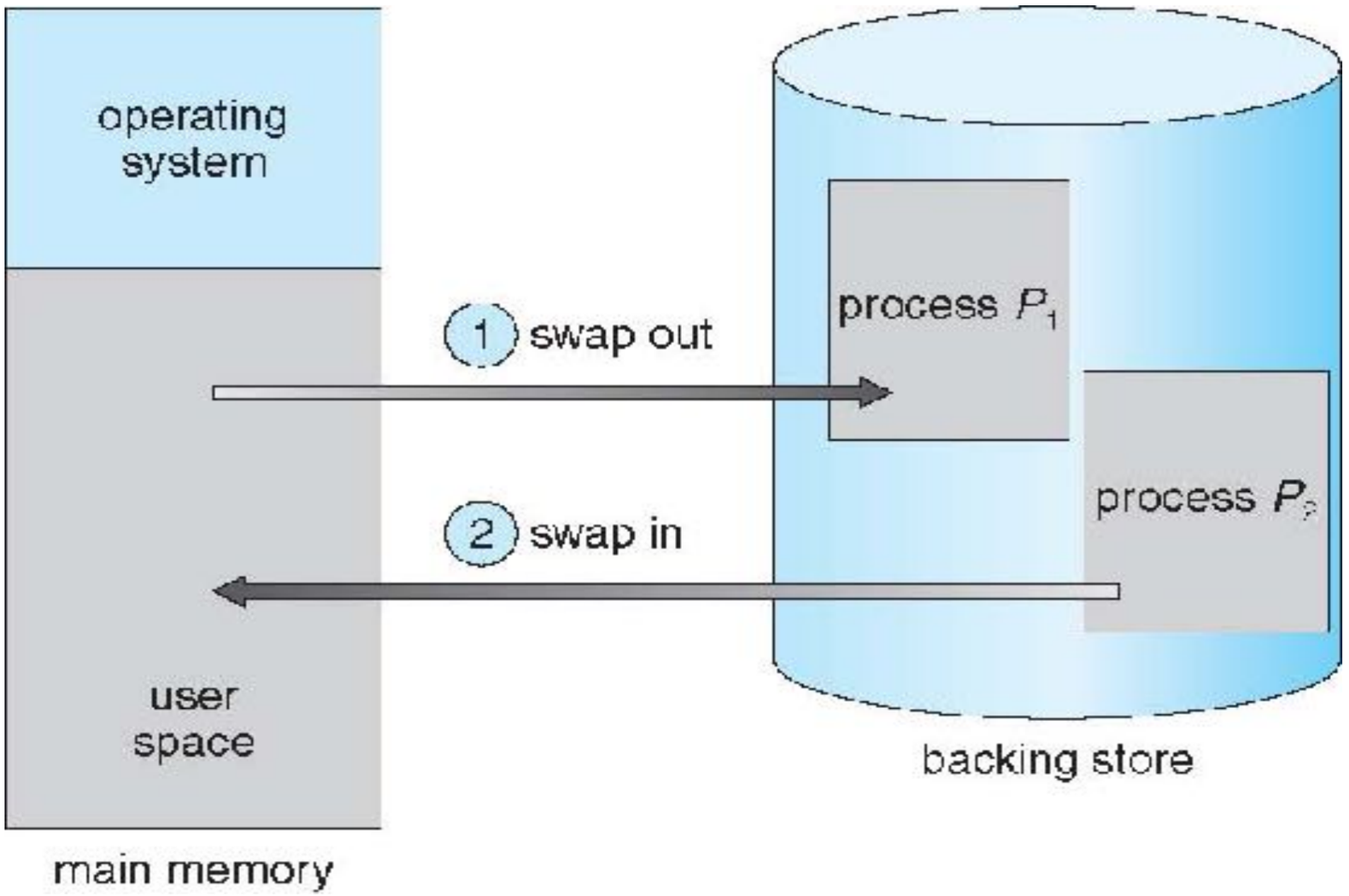
Menu

- Introduction
- **Swapping**
- Gestion mémoire contiguë
- Segmentation
- Pagination

Swapping

- Un processus peut être temporairement ***swapped*** hors de la mémoire vers un magasin de sauvegarde, puis remis en mémoire pour une exécution continue
  - L'espace mémoire physique total des processus peut dépasser la mémoire physique
- **Backing store** - disque rapide assez grand pour accueillir des copies de toutes les images de la mémoire pour tous les utilisateurs; doit fournir un accès direct à ces images de la mémoire
- **Roll out, roll in** - variante de remplacement utilisée pour les algorithmes de planification basée sur la priorité
  - processus de priorité inférieure est permuté de sorte qu'un processus de priorité plus élevée peut être chargé et exécuté
- La majeure partie du temps d'échange est le temps de transfert; le temps de transfert total est directement proportionnel à la quantité de mémoire échangée
- Système maintient une **ready queue** de processus prêts à exécuter qui ont des images de mémoire sur le disque

Swapping



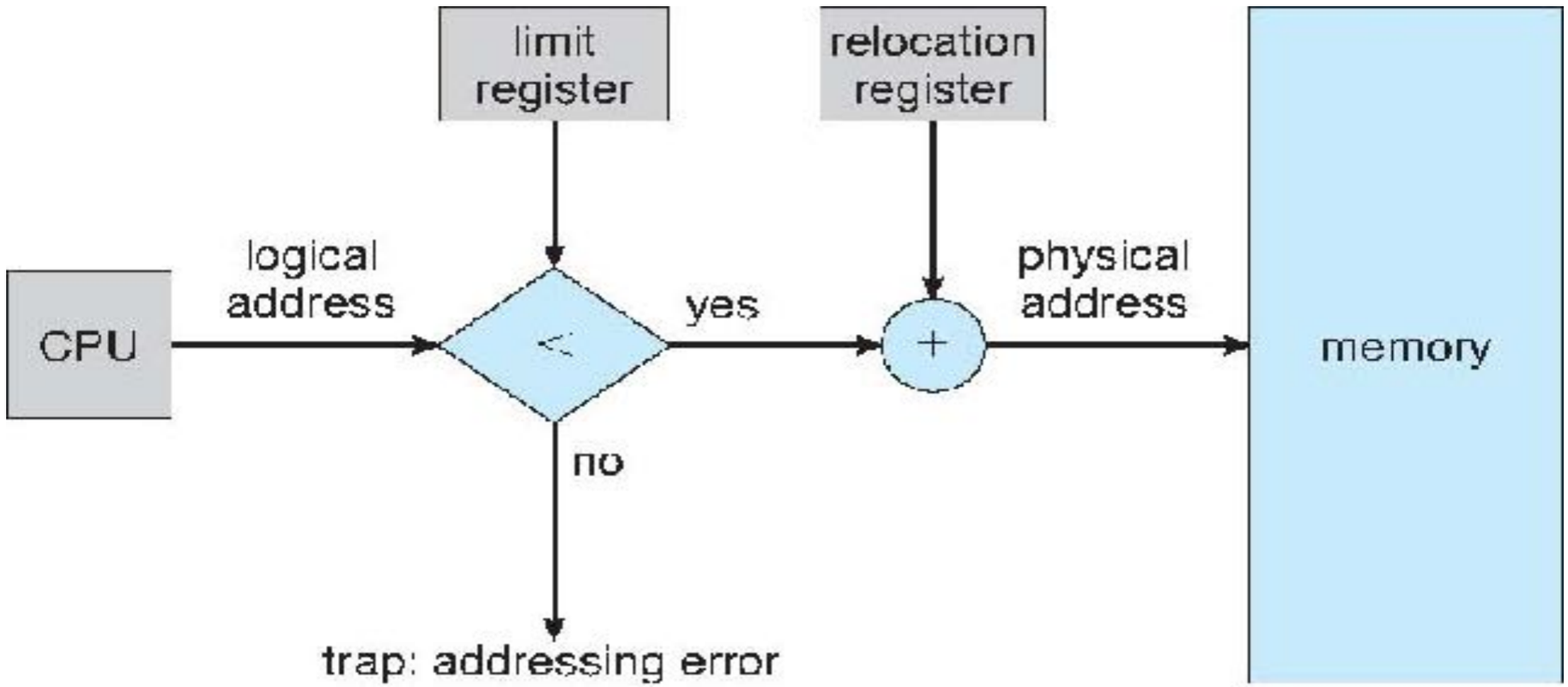
Menu

- Introduction
- Swapping
- **Gestion mémoire contiguë**
- Segmentation
- Pagination

Allocation contiguë

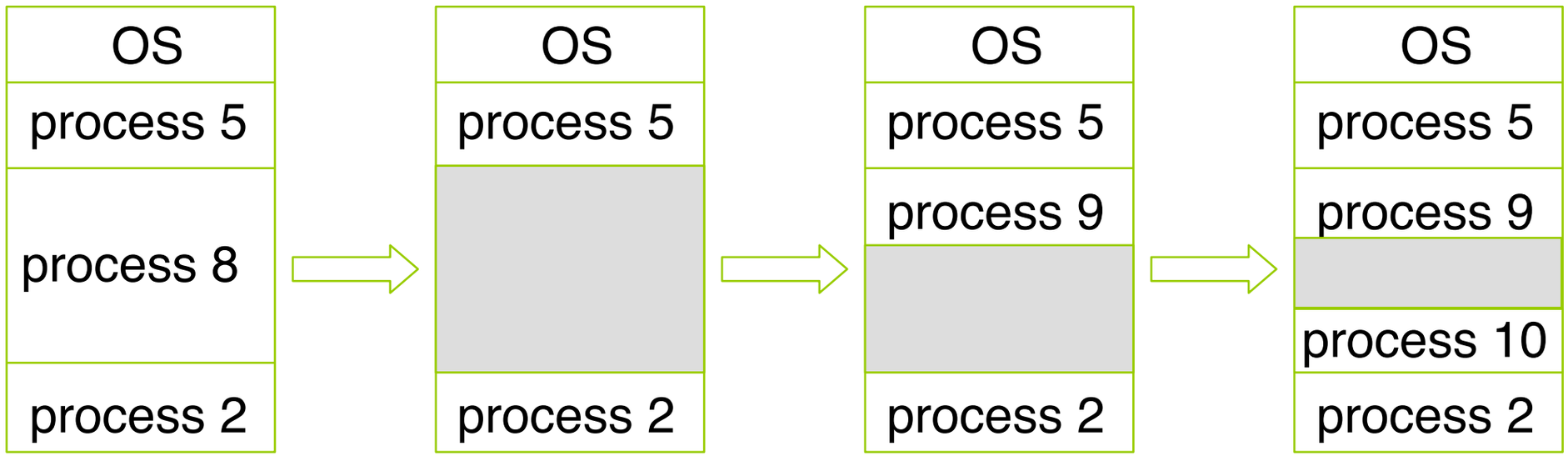
- Mémoire principale généralement en deux partitions:
  - Système d'exploitation résident, généralement tenu en mémoire faible avec un vecteur d'interruption
  - Les processus utilisateur sont ensuite conservés en mémoire haute
  - Chaque processus contenu dans une seule section contiguë de mémoire
- Registres de relocalisation utilisés pour protéger les processus utilisateur les uns des autres et pour modifier le code et les données du système d'exploitation
  - Le registre de base contient la valeur de la plus petite adresse physique
  - Le registre de limite contient une plage d'adresses logiques - chaque adresse logique doit être inférieure au registre de limite
  - MMU mappe l'adresse logique dynamiquement
  - Peut alors autoriser des actions telles que le code noyau transitoire et la taille du noyau changeante

MMU avec `base` + `limit` (relocation)



Gestion mémoire contiguë

- Allocation à plusieurs partitions
  - Avec des partitions de taille **fixe**, le degré de multiprogrammation est limité par le nombre de partitions
  - Tailles de partitions **variables** pour l'efficacité (dimensionnées pour les besoins d'un processus donné)
  - **Trou** - bloc de mémoire disponible; des trous de différentes tailles sont dispersés dans la mémoire
  - Quand un processus arrive, on lui attribue de la mémoire à partir d'un trou assez grand pour l'accueillir
  - Le processus sortant libère sa partition, les partitions libres adjacentes sont combinées
  - Le système d'exploitation conserve des informations sur:
    a) les partitions allouées
    b) les partitions libres (trou)



Problème de l'allocation mémoire dynamique

Comment satisfaire une requête de ***N*** bytes?

- First-fit: utilise le premier trou assez grand
- Best-fit: utilise le plus petit trou assez grand
- Worst-fit: utilise le plus gros trou

Fragmentation

- **Fragmentation externe**: mémoire inutilisable parce que trop petit
  - Ex: il y a assez de mémoire libre, mais fragmentée en plusieurs trous tous trop petits

- **Fragmentation interne**: mémoire gaspillée par le système.
  - Ex: le processus a besoin do 600KB mais le SE alloue par morceaux de 1MB, laissant 400KB inutilisées

- Fragmentation totale peut être de l'ordre de 33%

- Traduction d'adresses permet heureusement de *compacter*

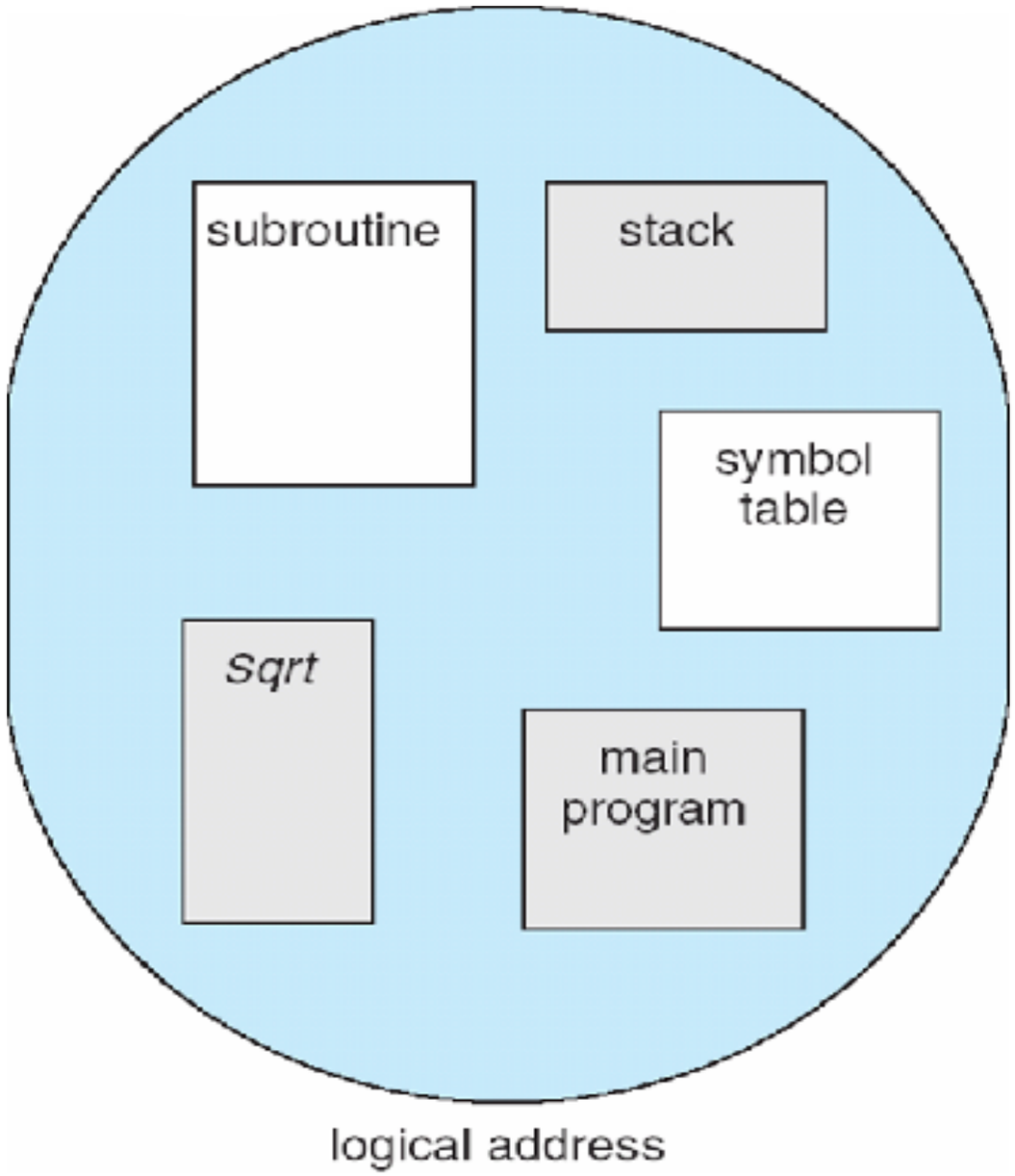
Menu

- Introduction
- Swapping
- Gestion mémoire contiguë
- **Segmentation**
- Pagination

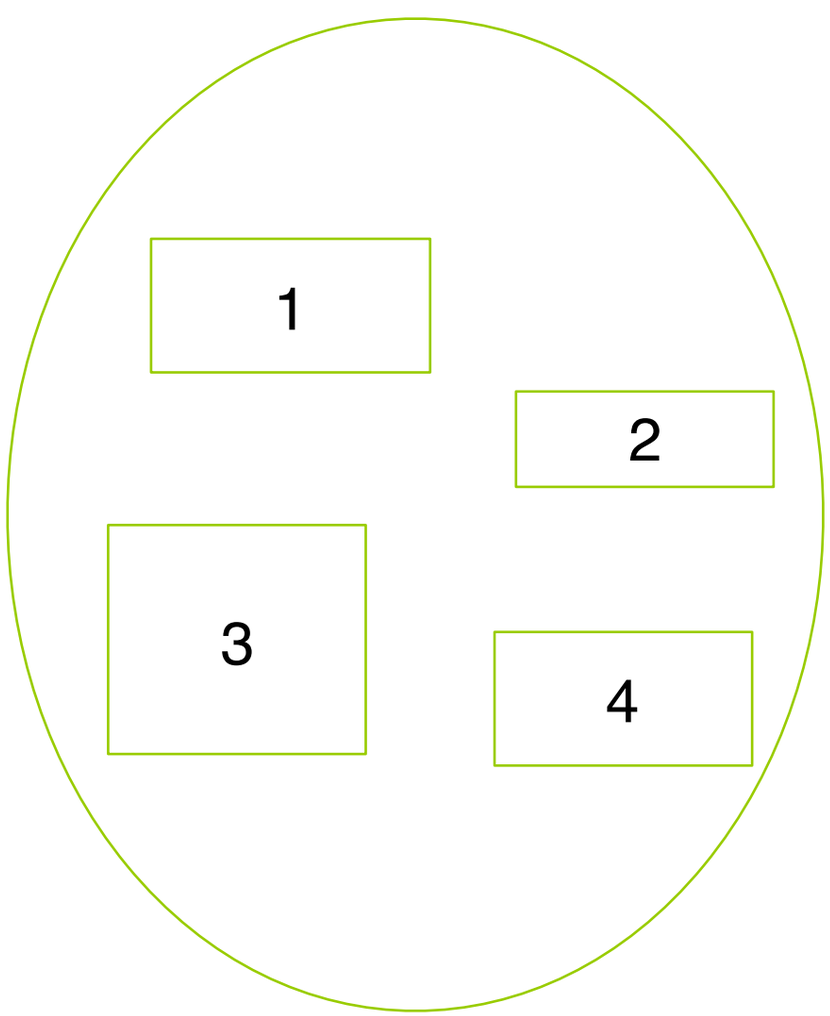
Segmentation

- Généralisation be `base+limit`
- Diviser l'espace logique d'un programme en segments:
  - main program
  - procedure
  - function
  - method
  - object
  - local variables, global variables
  - common block
  - stack
  - symbol table
  - arrays
- Chaque segment a sa propre `base` et `limit`

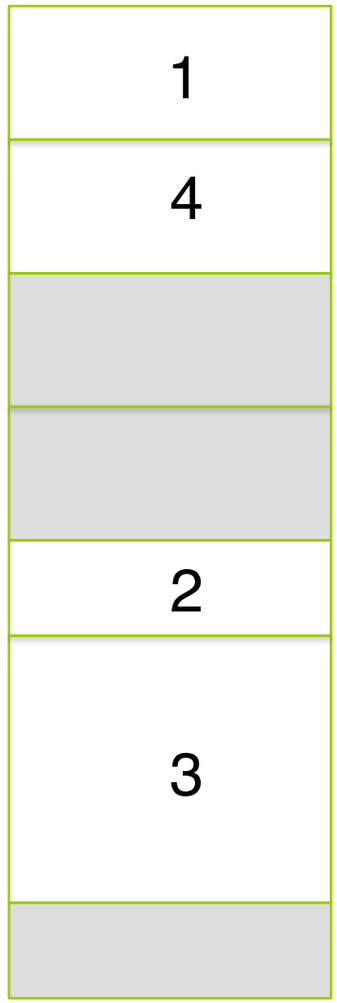
Un programme



Vue logique de la segmentation



espace utilisateur



espace mémoire physique

Architecture de segmentation

- Adresses logique:

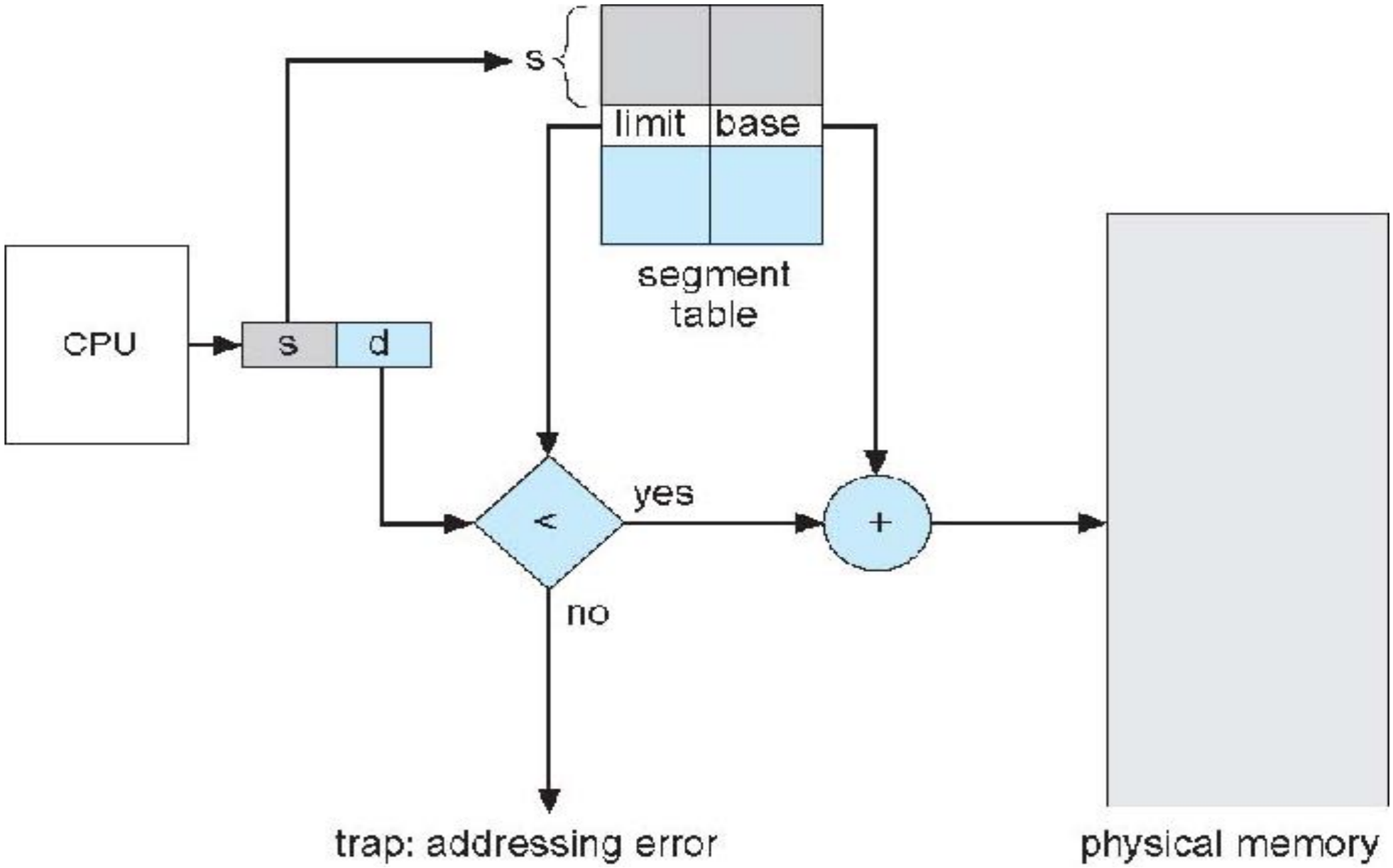
  < SegID, offset >

- Table de segment – chaque entrée de table a:
  - base – contient l'adresse physique de départ où les segments résident dans la mémoire
  - limit – specifies the length of the segment

- Address physique = `base[SegID] + Offset`
- Vérification: `Offset < limit[SegID]`

- Segment-table base register (STBR) pointe vers location de la table de segment en mémoire
- Segment-table length register (STLR) indique le nombre de segments utilisés par un programme;

Segmentation



Les problèmes avec segmentation

- Jamais assez de segments: on aimerait un segment par *objet*
- Trop de segments: table devient très large
- Addresses trop grandes
- Fragmentation externe

Menu

- Introduction
- Swapping
- Gestion mémoire contiguë
- Segmentation
- **Pagination**

Pagination

- Diviser l'espace *logique* en ***pages***
- Diviser l'espace *physique* en ***frames***
- Chaque page et frame a la même taille (e.g. 4KB)

- **Table de pages** (page table) effectue la traduction d'adresses logiques en adresses physiques

- Pas de fragmentation externe
- Fragmentation interne à cause de l'allocation par page

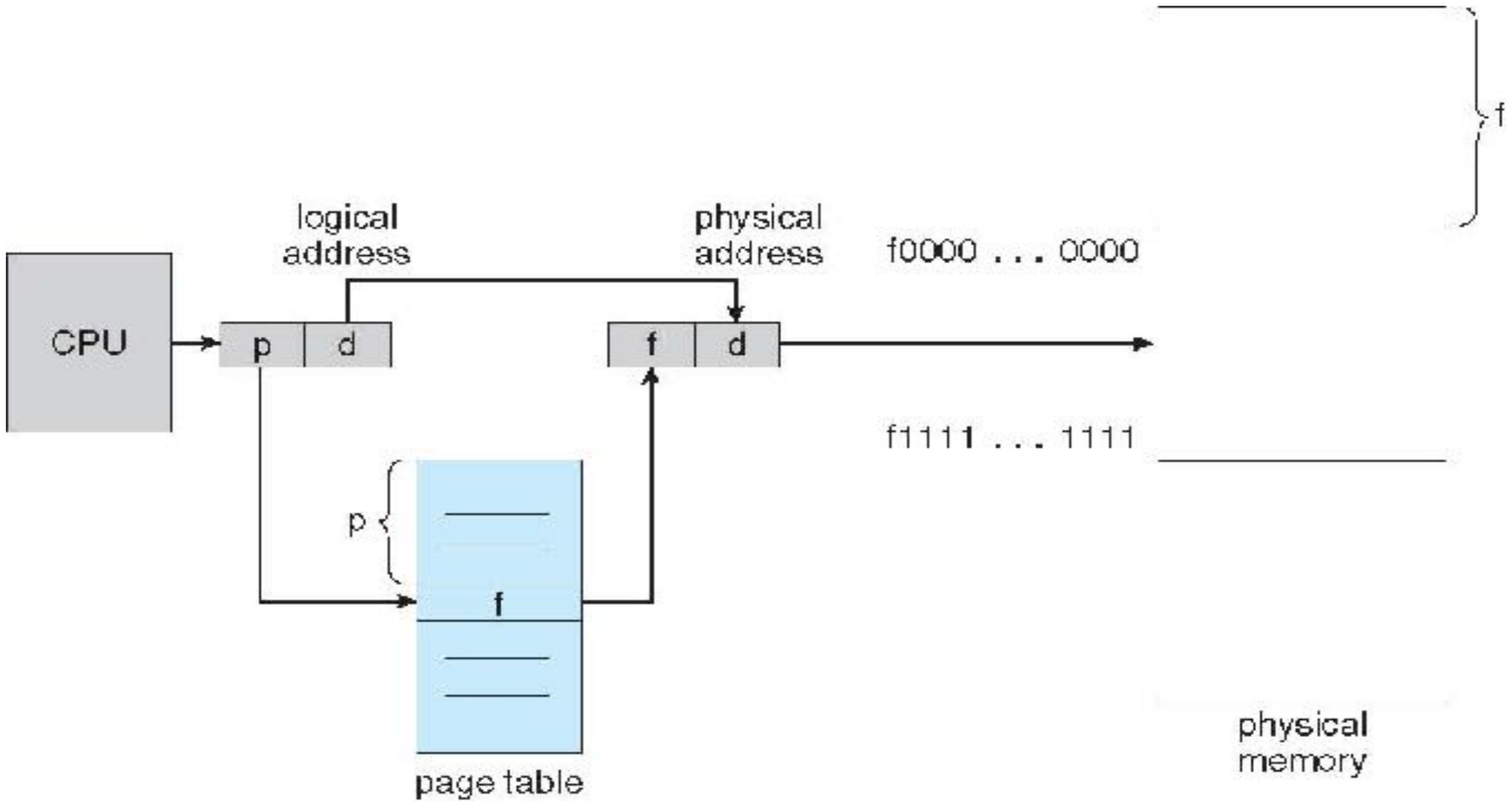
Système de traduction d'adresses

- L'adresse générée par le CPU est divisée en:
  - **Page number** (***p***) – utilisé comme un index dans une table de page qui contient l'adresse de base de chaque page dans la mémoire physique
  - **Page offset** (***d***) – combiné avec l'adresse de base pour définir l'adresse de mémoire physique qui est envoyée à l'unité de mémoire

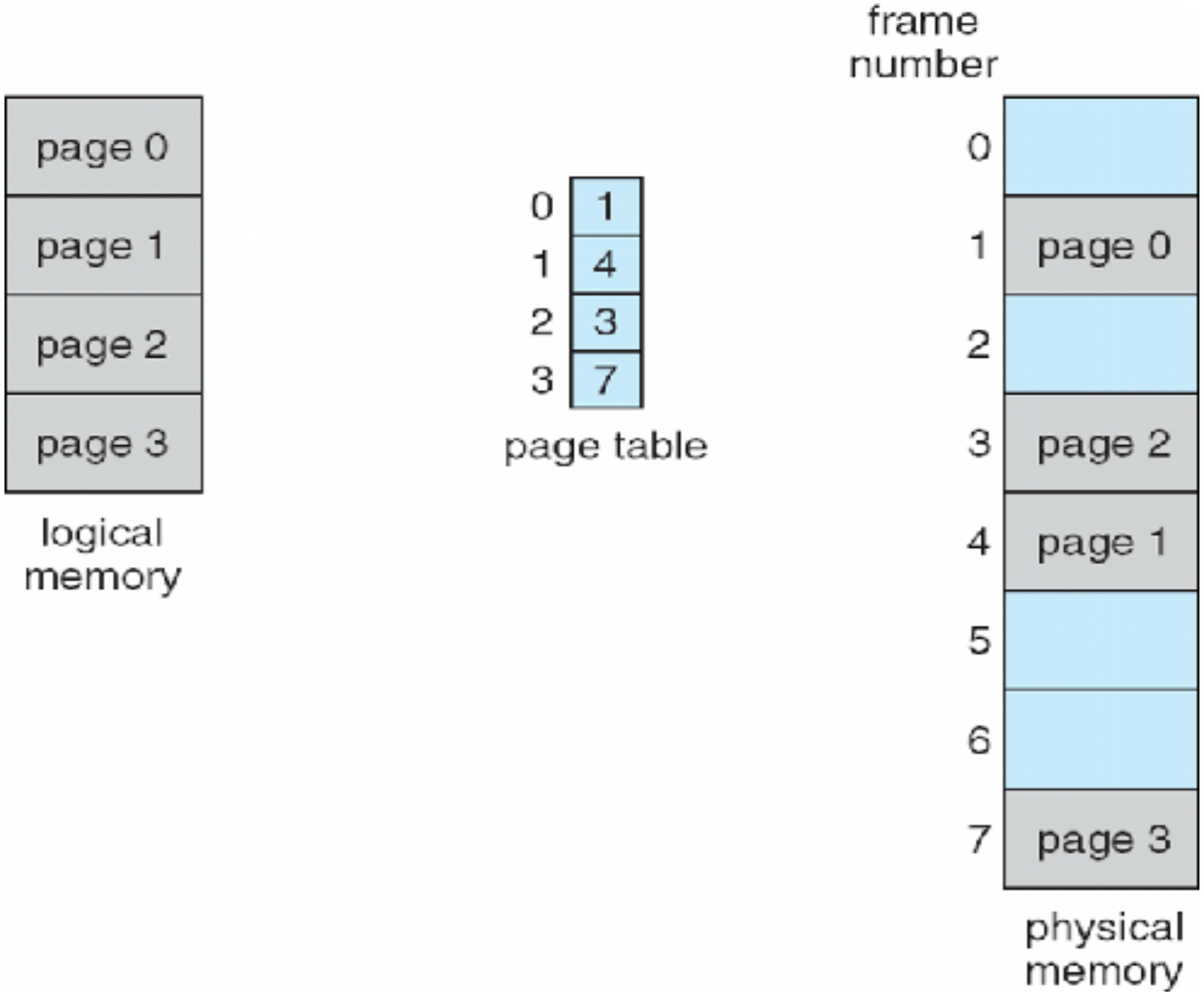
page number	page offset
p	d
(m – n) bits	n bits

- Pour un espace logique grandeur 2^m et les page de grandeur 2^n

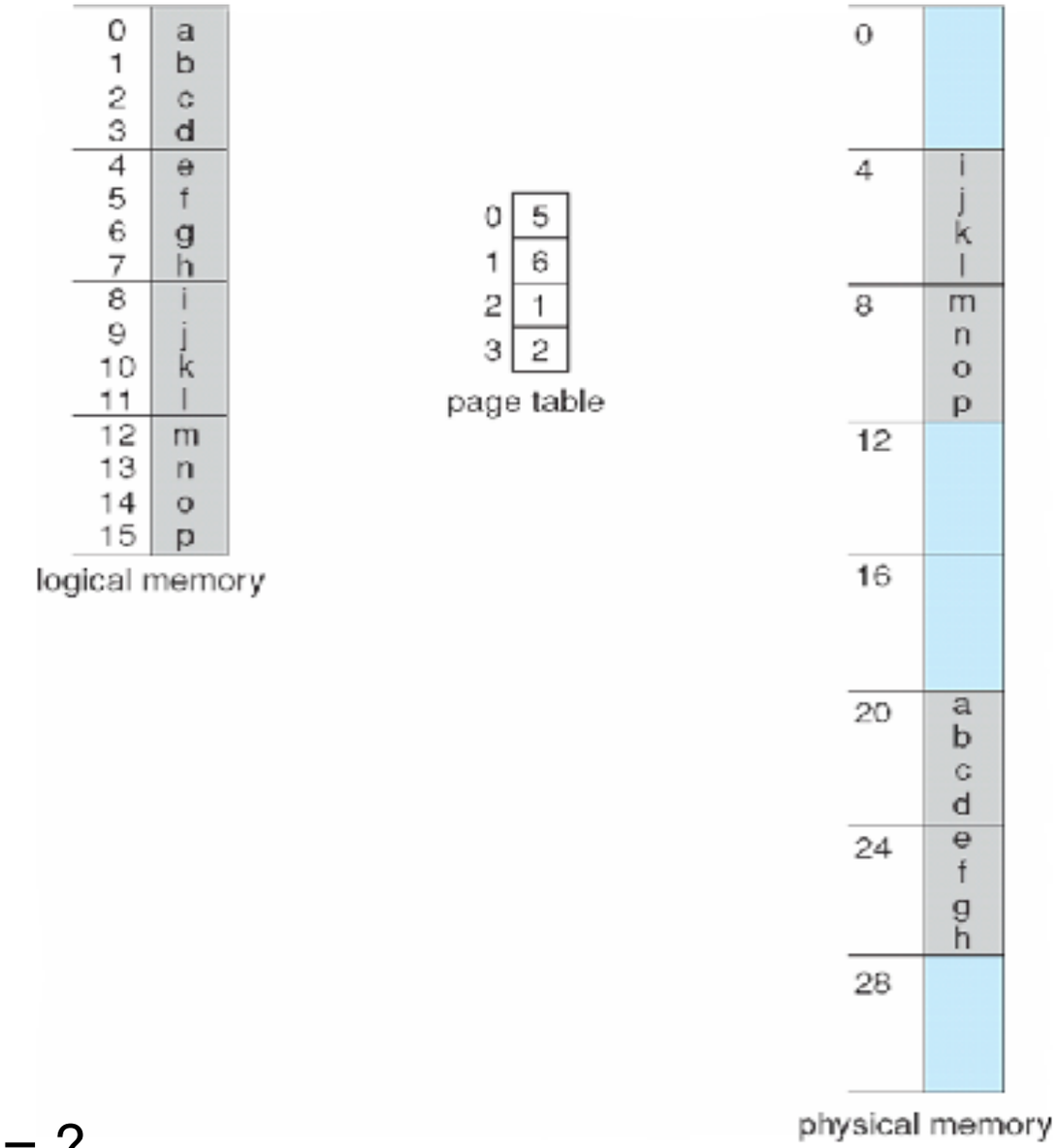
Pagination dans le MMU



Modèle de pagination



Exemple de pagination



n = ? et m = ?

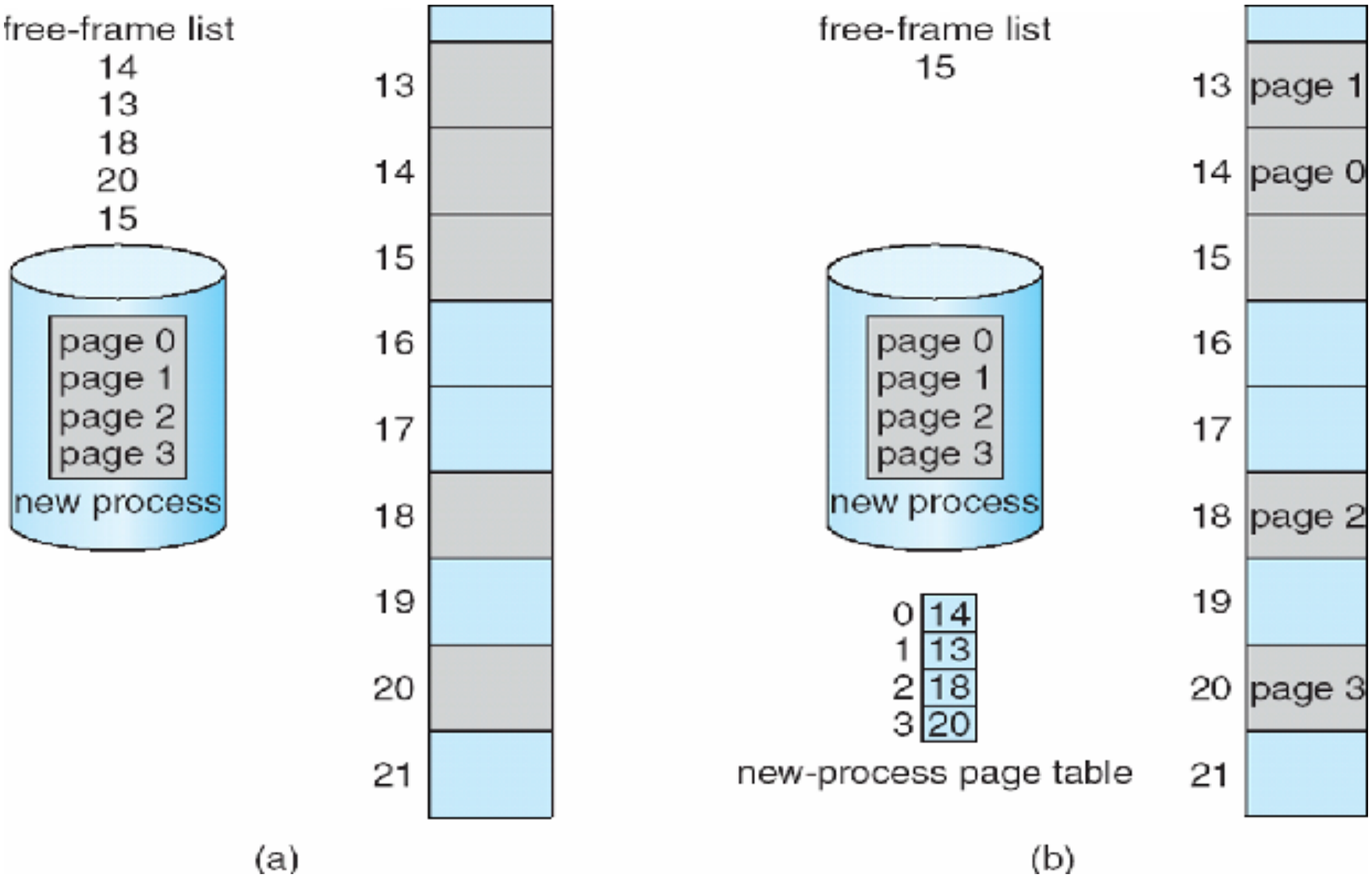
n = 2, m = 4 (32-byte mémoire physique et 4-byte grandeur de page)

Coût de la pagination

- Fragmentation interne
  - Page size = 2,048 bytes
  - Process size = 72,766 bytes
  - 35 pages + 1,086 bytes
  - Fragmentation interne: 2,048 - 1,086 = 962 bytes
  - Pire fragmentation interne: = 1 frame + 1 byte
  - moyen = 1 / 2 frame size

  - Pages (frames) plus petites -> plus de pages
    - ✓ e.g. Adresses de 64bit, pages de 4KB => m = 64, n= 12 => 2^52 pages

Frames Libres



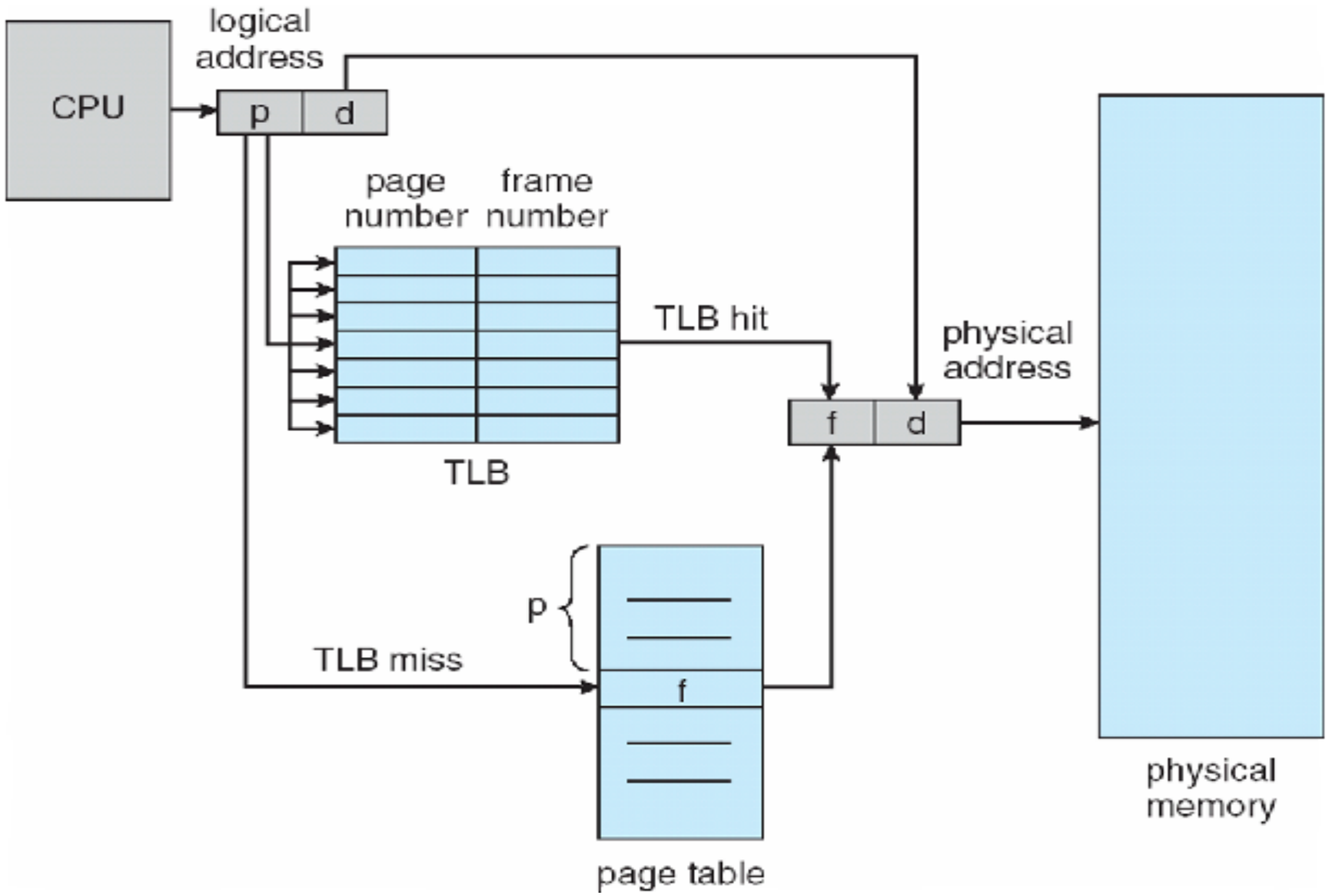
Avant

Après

Implémentation d'une table de pages

- Les tables de pages sont gardées en mémoire
- *Page table base register* donne l'adresse de la table
- *Page table length register* donne la taille de la table
  - Problème: 2 accès mémoire par "accès mémoire"!
- TLB (Translation Look-aside Buffer): cache de la table des pages
- TLB est rapide et danse le CPU, mais petit
- Si un numéro de *page* n'est pas dans le TLB:
  - Chercher le *frame* correspondant dans la table de pages
  - Placer le résultat dans le TLB (comment fair la remplacement? LRU, FIFO?)

Modèle de TLB



Temps d'accès effectif

- ε : Temps de cherche dans le TLB
- α : TLB hit ratio
- m: Temps d'accès à la mémoire

- **Effective Access Time (EAT)**

$$EAT = (\varepsilon + 1m)\ \alpha + (\varepsilon + 2m)(1 - \alpha)$$
$$= 2m + \varepsilon - m\ \alpha$$

- Ex: α = 80%, ε = 20ns, m = 100ns

$$EAT = 0.80 \times (20+100) + 0.20 \times (20+200) = 140ns$$

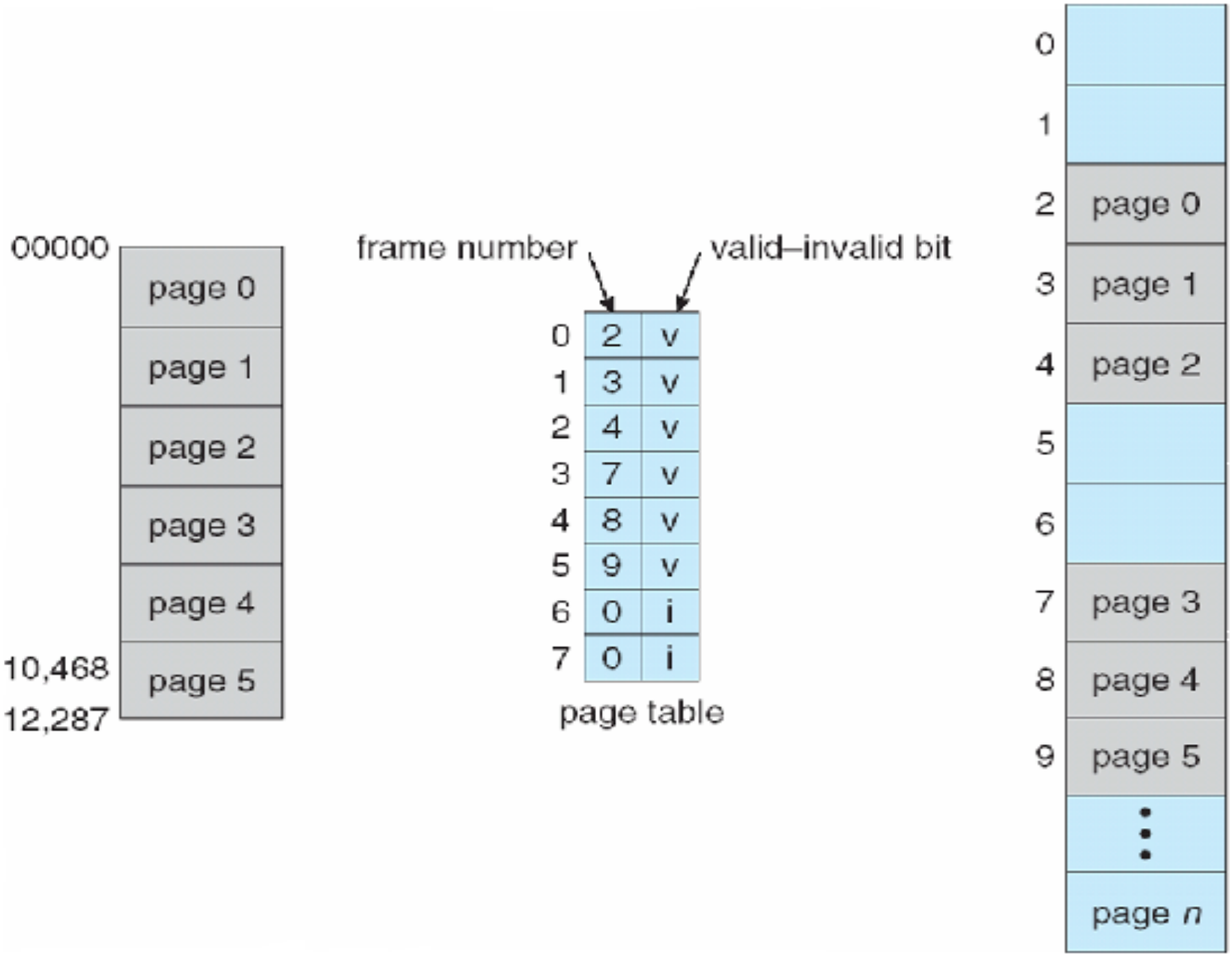
- Ex: α = 99%, ε = 20ns, m=100ns

$$EAT = 0.99 \times (20+100) + 0.01 \times (20+200) = 121ns$$

Protection mémoire

- Protection de la mémoire implémentée en associant un bit de protection à chaque *frame* pour indiquer si l'accès en lecture seule ou en lecture-écriture est autorisé
  - Peut également ajouter plus de bits pour indiquer l'exécution de la page uniquement, etc.
- Deux méthodes pour indiquer les entrées valides dans la table de pages:
  - Bit **valide-invalide** attaché à chaque entrée de la table de pages:
    - "Valide" indique que la page associée est dans l'espace d'adressage logique du processus, et est donc une page légale
    - "Invalide" indique que la page ne se trouve pas dans l'espace adresse logique du processus
  - OU utilisez le **page-table length register** (PTLR)
    - cette méthode gaspille moins de mémoire pour la table de page
- Toute violation entraîne un *trap* au noyau

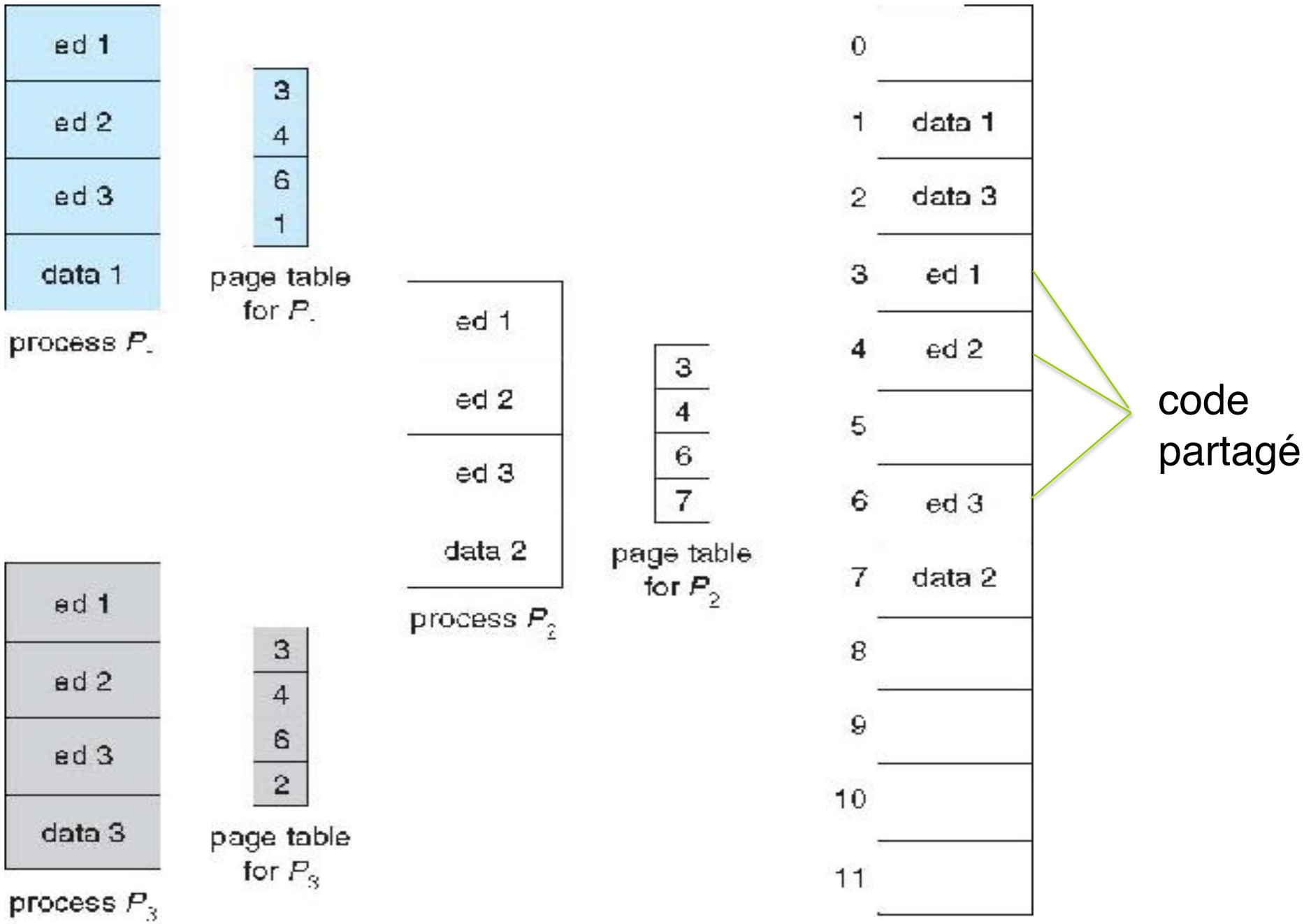
Bit valid-invalide



Page partagées

- Code partagées
  - Une copie du code *read-only* (**réentrant**) partagée entre les processus (e.g., les éditeurs de texte, les compilateurs, les systèmes de fenêtres)
  - Similaire à plusieurs threads partageant le même espace de processus
  - Aussi utile pour la communication interprocessus si le partage de pages en lecture-écriture est autorisé
- Code et données privées
  - Chaque processus conserve une copie distincte du code et des données
  - Les pages pour le code privé et les données peuvent apparaître n'importe où dans l'espace d'adressage logique

Ex: Pages partagées



Implémentations de tables de pages

- Placer une grosse table en mémoire pose problèmes
  - Considérons un espace d'adressage logique 32 bits
  - Grandeur de page of 4 KB (2^{12})
  - La table de page aurait plus de 1 million d'entrées (2^{32} / 2^{12})
  - Si chaque entrée est 4 bytes, cela équivaut à 4 MB d'espace d'adressage physique / mémoire pour la table de pages

- Tables de pages plus complexes:
  - Table de pages hiérarchique
  - Tables de pages inversée
  - Tables de pages “hash”

Tables de pages hiérarchique

- Divisez l'espace d'adressage logique en plusieurs tables de pages
- Une technique simple est une table de page à deux niveaux
- Nous avons ensuite la page de la table de page

Tables de pages hiérarchique

Au lieu de diviser les adresses en | *PageNb* | *Offset* | avec:

$$\mathit{FrameNb} = \texttt{pagetable}[\mathit{PageNb}]$$

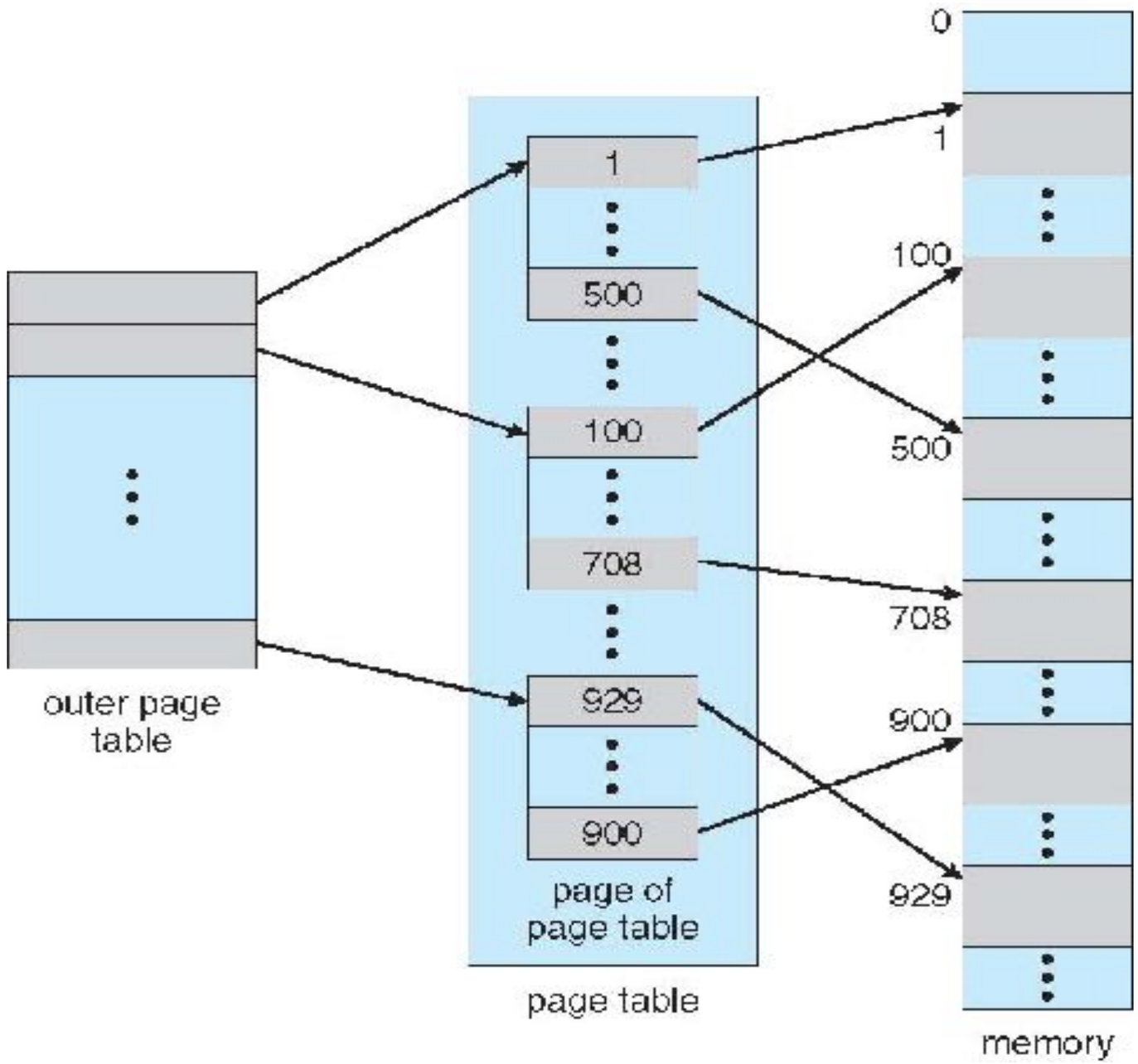
On divise les adresses en | PageNb_1 | PageNb_2 | *Offset* | avec:

$$\mathit{FrameNb} = \texttt{pagetable}[\mathit{PageNb}_1][\mathit{PageNb}_2]$$

Voire | PageNb_1 | $\cdots$ | PageNb_n | *Offset* | avec:

$$\mathit{FrameNb} = \texttt{pagetable}[\mathit{PageNb}_1]\cdots[\mathit{PageNb}_n]$$

Exemple de table de pages à 2 niveaux



Exemple de table de pages à 2 niveaux

- Adresse logiciel (32-bit machine avec 1 KB grandeur de page) est divisé en:
  - numéro de page: 22 bits
  - offset de page: 10 bits

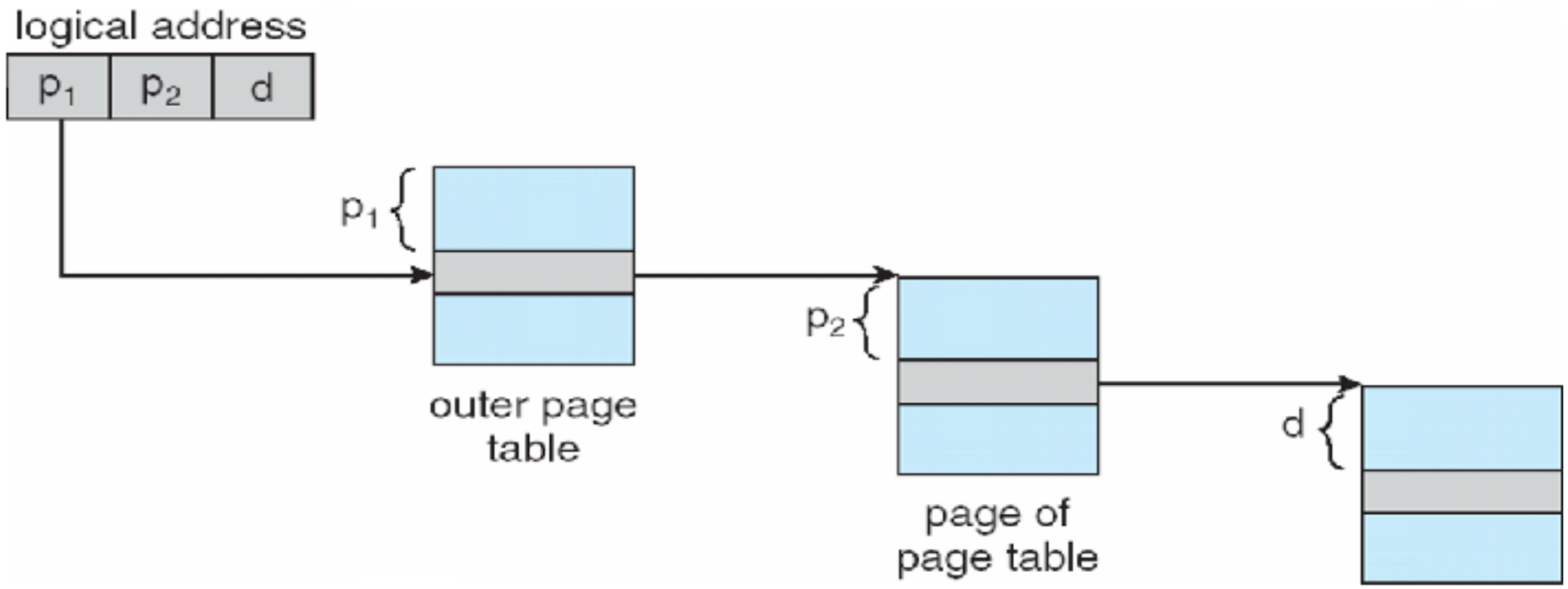
- La numéro de page est encore divisé en::
  - outer page: 12-bit
  - inner page: 10-bit

- Thus, a logical address is as:

page number		page offset
p1	p2	d
12	10	10

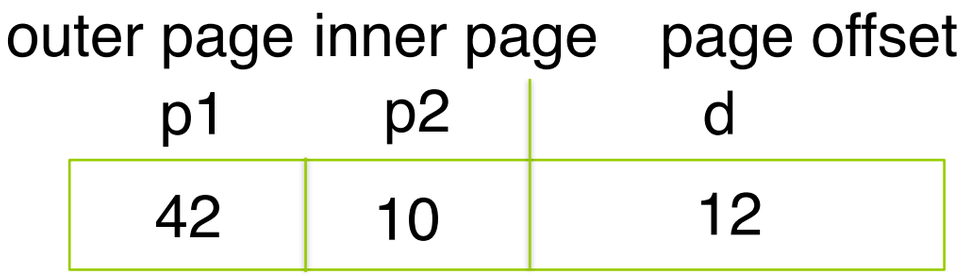
- “**forward-mapped** page table”

Exemple de table de pages à 2 niveaux



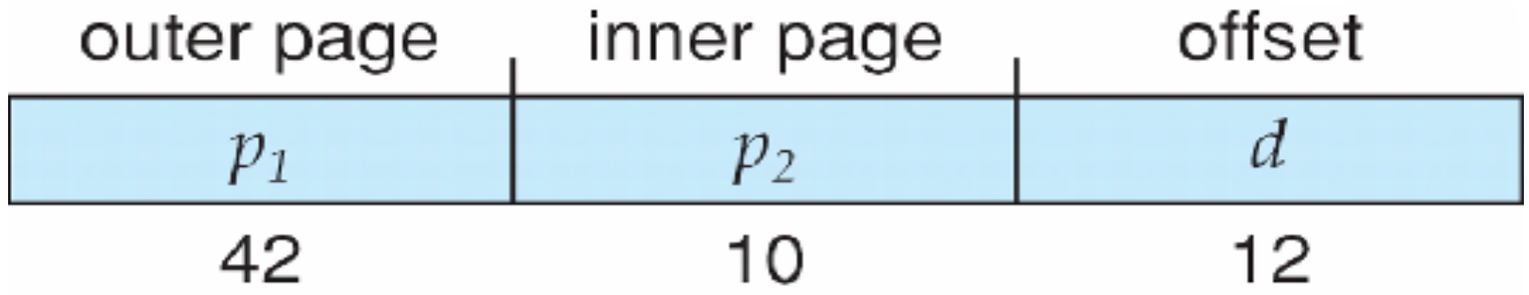
Espace d'adressage logique 64 bits

- 2 niveaux ne suffisent pas pour une machine 64 bits
- Grandeur de page 4 KB (2^{12})
  - Table de page a 2^{52} valeurs
  - Si schéma à deux niveaux, avec une table de page intérieure qui pourrait être 2^{10} x 4-byte valeurs
  - Adresse de forme



  - Table page “outer” a 2^{42} valeurs ou 2^{44} bytes
  - Nous pouvons continuer à ajouter des niveaux mais chacun nous coûte un autre accès physique à la mémoire

Paging avec 3 niveaux



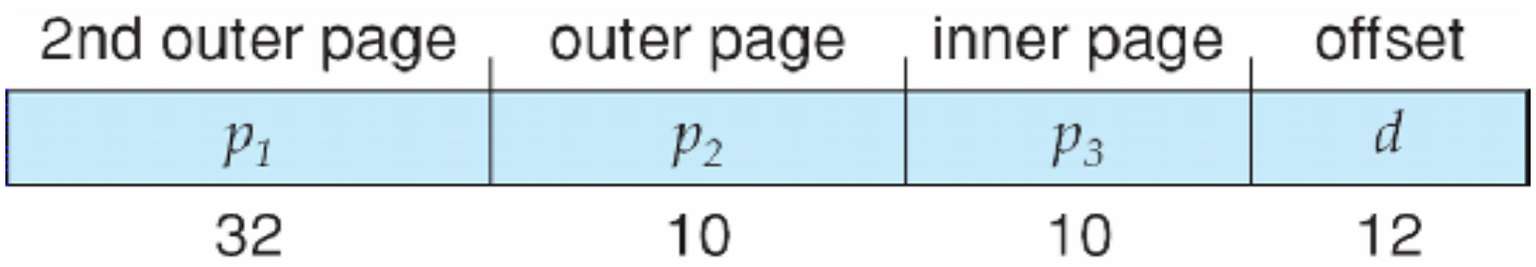
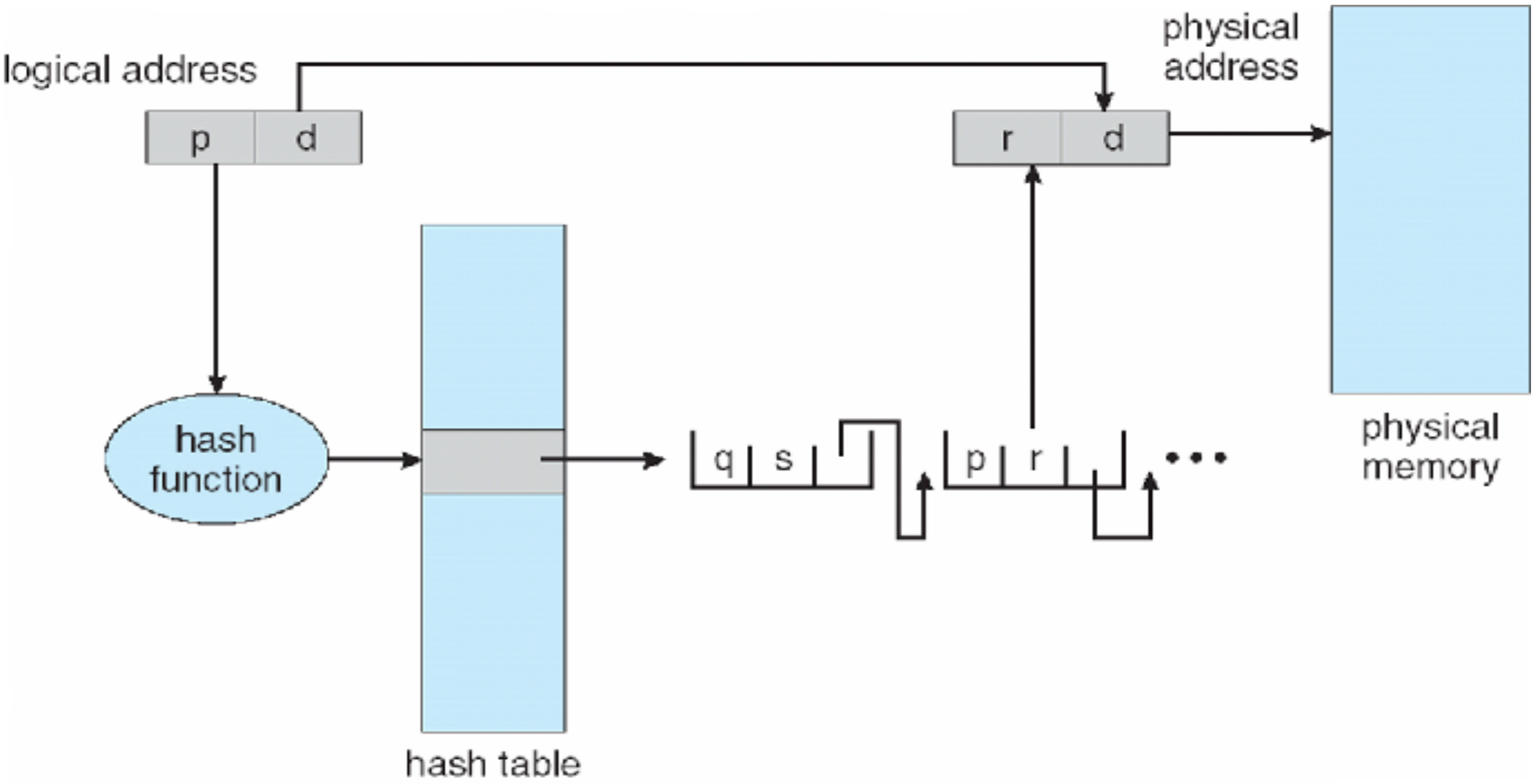


Table de page hachée

- Commun dans les espaces adresse> 32 bits
- Le numéro de page virtuelle est haché dans une table de pages
  - Cette table de pages contient une chaîne d'éléments hachage au même endroit
- Chaque élément contient
  - (1) le numéro de la page virtuelle
  - (2) la valeur du cadre de page mappé
  - (3) un pointeur vers l'élément suivant
- Les numéros de pages virtuelles sont comparés dans cette chaîne, à la recherche d'une correspondance
  - Si une correspondance est trouvée, la *frame* physique correspondante est extraite
- Variation pour les adresses 64 bits: tables de pages groupées
  - Semblable à haché, mais chaque entrée se réfère à plusieurs pages (comme 16) plutôt que 1
  - Particulièrement utile pour les espaces d'adresses clairsemés (où les références de mémoire sont non contiguës et dispersées)

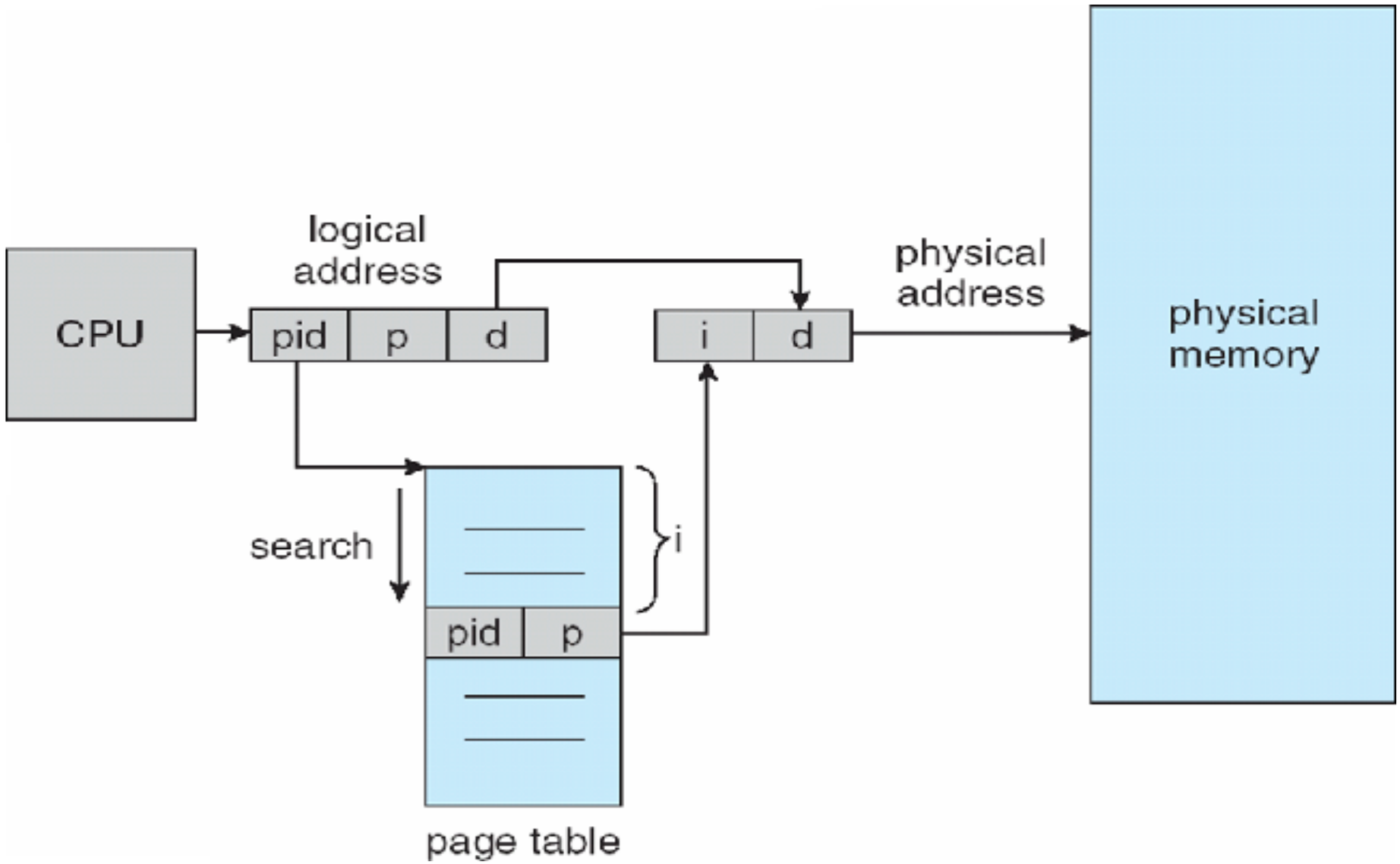
Table de page hachée



Tables de pages inversée

- Plutôt que chaque processus ayant une table de pages et de garder une trace de toutes les pages logiques possibles, suivre toutes les pages physiques
- Une entrée pour chaque vraie page de mémoire
- L'entrée se compose de l'adresse virtuelle de la page stockée dans cet emplacement mémoire réel, avec des informations sur le processus qui possède cette page
- Diminue la mémoire nécessaire pour stocker chaque table de pages, mais augmente le temps nécessaire pour effectuer une recherche dans la table lorsqu'une référence de page se produit
- Mais comment implémenter la mémoire partagée?
  - Un mappage d'une adresse virtuelle à l'adresse physique partagée

Inverted Page Table Architecture



Sommaire

- Les processus ne peuvent être exécutés que lorsqu'ils sont chargés en mémoire
- Nous pouvons distinguer entre les adresses logiques utilisées par les programmeurs et les adresses physiques qui sont les emplacements réels des données et des instructions
- L'unité de gestion de la mémoire est responsable de faire la traduction entre les deux
- Le moyen le plus simple consiste à allouer de la mémoire à un processus de manière contiguë
- Cela peut provoquer une fragmentation
- La segmentation et la pagination sont deux façons d'allouer de la mémoire de manière non contiguë
- Dans le cas de la pagination, nous pouvons éliminer la fragmentation externe, mais au prix de grandes tables de pages et d'allocations de mémoire supplémentaires requises par la MMU
- Nous pouvons atténuer ces effets négatifs avec un TLB (sorte de cache pour la table de pages) ou avec des tables de pages hiérarchiques, hachées ou inversées, mais chacun a ses avantages et ses inconvénients.