IUT DE PARIS
UNIVERSITÉ PARIS DESCARTES

Gestion de la Mémoire

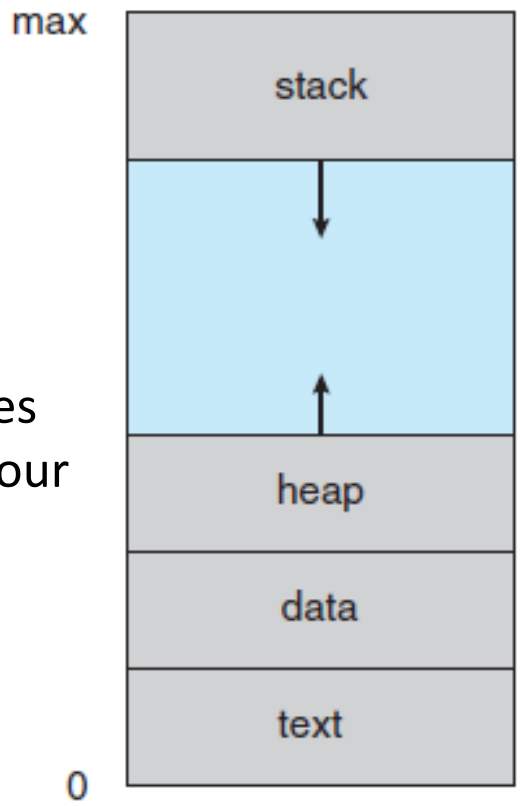
Professeur

Jocelyne Elias

IUT DE PARIS
UNIVERSITÉ PARIS DESCARTES

Processus en mémoire

- Un processus est un programme en exécution.
- Un processus bien évidemment est plus qu'un code de programme. Ce code est appelé section texte ("**text section**").
- En plus, un processus comprend les éléments suivants :
- *l'activité actuelle*, comme représentée par la valeur du **compteur** et le ***contexte*** (contenu des registres du processus).
- Une pile ("**stack**") : elle contient des données temporaires comme les paramètres des fonctions, les adresses de retour et variables locales
- Une section de données (**data**) qui contient les variables globales.
- Un tas ''**heap**'' : une mémoire allouée dynamiquement durant le temps d'exécution du proc.
- Donc, la structure générale d'un proc. est illustrée dans cette figure.



La mémoire principale

- La mémoire principale **RAM** (*Random Access Module*)/**ROM** (*Read Only Memory*) est une ressource importante
- La taille des programmes s'accroît aussi vite que celle des mémoires
- Mémoire infiniment grande, rapide et non volatile ?

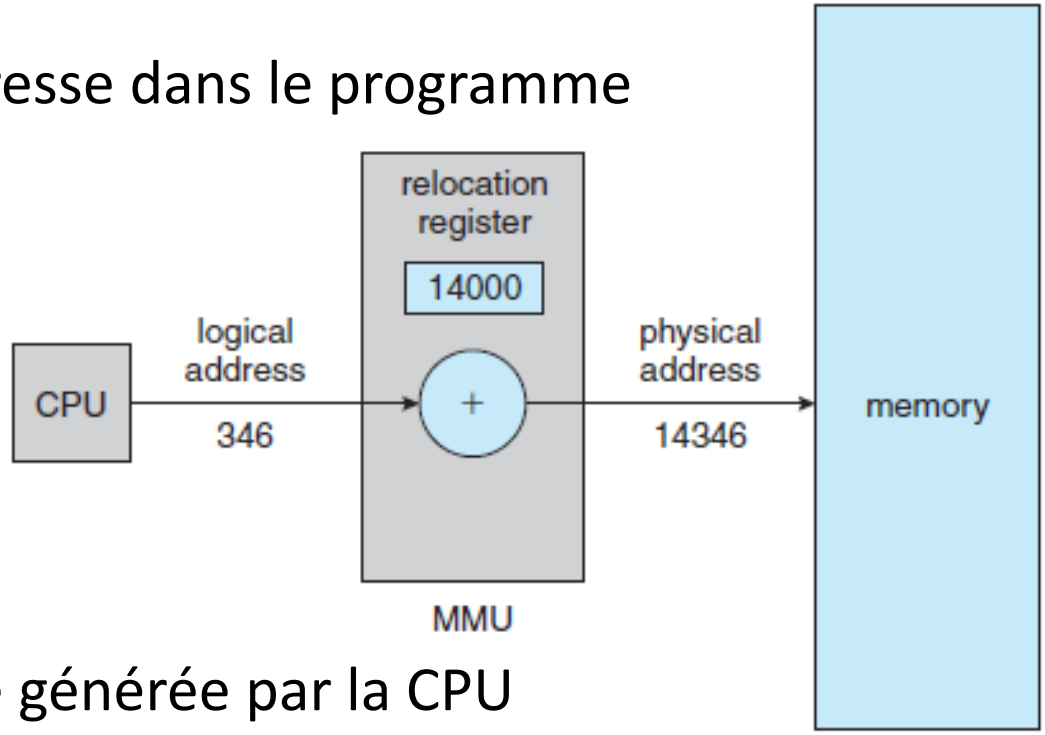
Hiérarchie de la mémoire

- Quelques mégaoctets de mémoire cache volatile, rapide et chère → **cache**
- Quelques giga-octets de mémoire volatile de rapidité d'accès et de prix moyens → **RAM**
- Quelques téraoctets de mémoire plus lente et non volatile (mémoire de masse) → **Disque magnétique**

- Le Système d'Exploitation (SE) a pour rôle de coordonner la manière dont toutes ces différentes mémoires sont utilisées.
  - Gestionnaire de mémoire

Adresse logique/physique

- **Adresse symbolique** : adresse dans le programme



- **Adresse logique** : adresse générée par la CPU
- **Adresse physique** : adresse vue par la mémoire et registres d'adresses de la mémoire
- La **MMU** (**Memory Management Unit**) fait le mappage entre l'adresse logique et l'adresse physique

IUT DE PARIS
UNIVERSITÉ PARIS DESCARTES

Abstraction de la mémoire : Espaces d'adressage

- Pour avoir plusieurs applications (ou programmes) en mémoire simultanément →
  - Savoir résoudre les problèmes de protection et de réallocation
  - créer une abstraction de la mémoire qui soit efficace
  - Utiliser la notion de l'espace d'adressage!
- **Espace d'adressage** : ensemble des adresses qu'un processus peut utiliser pour adresser la mémoire.

Espace d'adressage

- Chaque processus doit avoir son propre espace d'adressage pour éviter des conflits avec les autres processus.
- Déterminer l'ensemble des adresses légales
- La **base** : la plus petite adresse physique légale
- La **limite** (étendue) : la **taille totale** de l'espace auquel le processus peut/a le droit d'accéder.

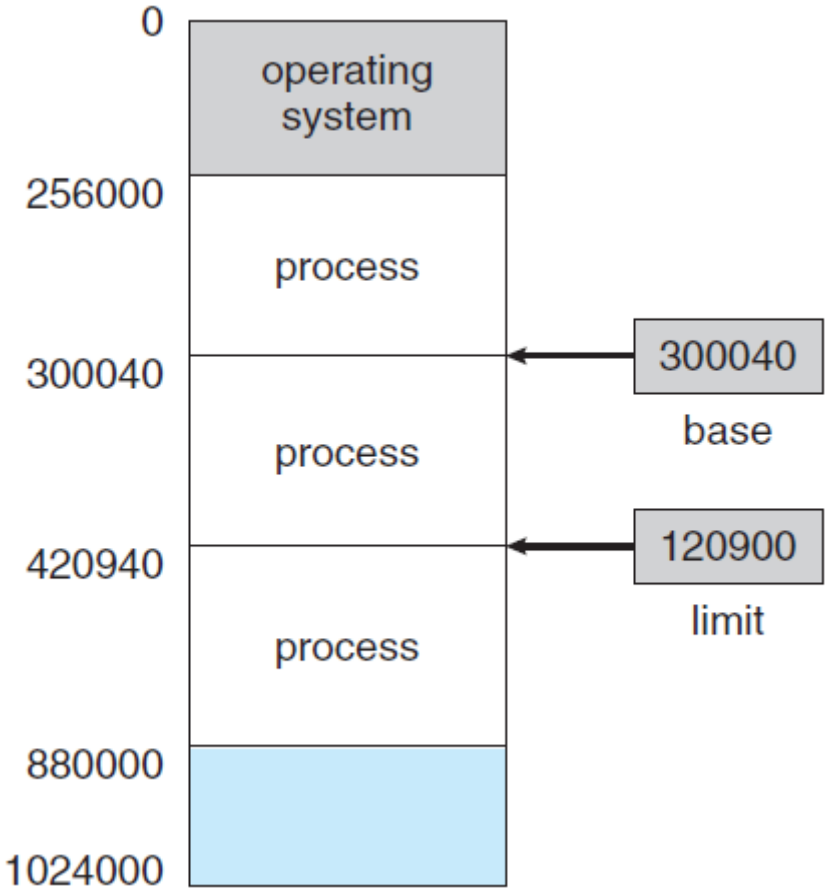
Les registres de base et de limite

- Façon simple d'attribuer à chaque processus son espace d'adressage.
- Equiper l'Unité Centrale de deux registres matériels : les registres de ***base*** et de ***limite***
- Les programmes sont chargés en mémoire sans réallocation
  - Sont rangés dans les mots mémoires consécutifs.
- A l'exécution du programme, le SE met dans les registres de base et de limite les adresses physiques du début et de fin du programme.

Exemple
registres de base/limite

- Registre de base : 300040
- Registre de limite: 120900
- Donc le processus peut accéder d'une manière légale à toutes les adresses allant de 300040 à 420939 (incluse)

- Registre de base : 420940
- Registre de limite: 459060
- …



Les registres de base et de limite

- *Cette méthode convient si la mémoire physique de l'ordinateur est suffisamment grande pour contenir tous les processus*
- *Mais ce n'est pas le cas !*

- *Besoin d'une autre solution …*
  - Segmentation ou/et pagination
  - Mémoire virtuelle
    - Chaque programme a son propre espace d'adressage découpé en petites entités appelées pages
  - Va-et-vient (swapping)

Allocation de la mémoire

- En général, la mémoire est divisée en 2 partitions : une pour le OS et l'autre pour les processus des utilisateurs.
- L'objectif est de décider comment allouer la mémoire disponible aux processus qui se trouvent dans la file d'entrée en attente pour être exécuté en mémoire.

- **Allocation contiguë** : chaque processus est placé dans une seule section, qui est à côté d'une autre section contenant le processus suivant, et ainsi de suite …
- **Protection de la mémoire** → registre de base/relocation et registre de limite

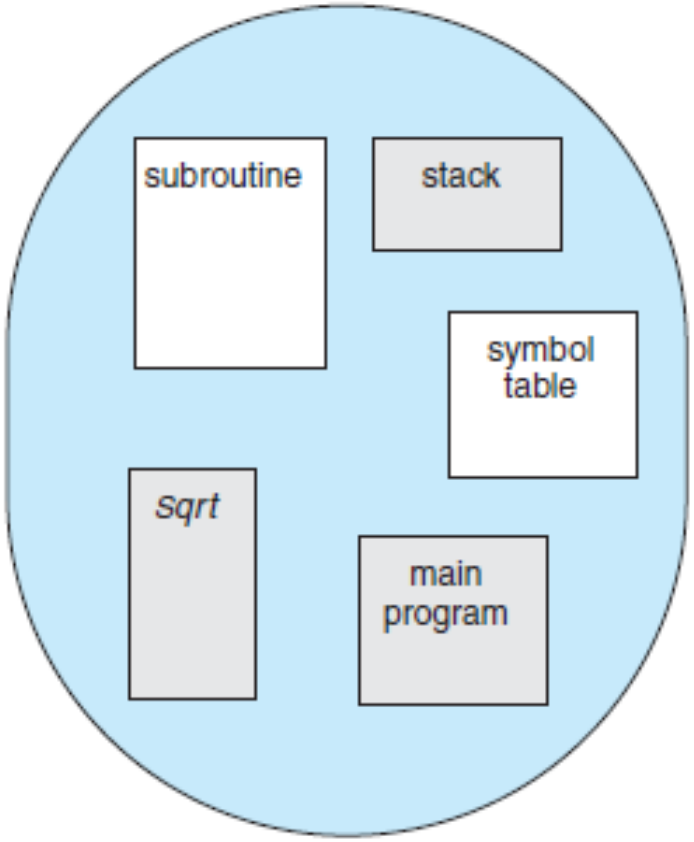
Allocation de la mémoire

- **La Stratégie First Fit** : choisir le **1er bloc libre** de taille suffisante. Nous arrêtons la recherche à peine nous avons trouvé ce qu'il nous faut.
- **Stratégie Best Fit** : choisir le **plus petit bloc libre** de taille suffisante. Nous sommes obligés dans ce cas de parcourir toute la liste, sauf si la liste est ordonnée
- **Stratégie Worst Fit** : choisir le **plus grand bloc libre**.
- **Inconvénients** : **fragmentation** de la mémoire en un grand nombre de petits trous (non contigus) qui sont inutilisables pour répondre à une requête même si l'espace total le permettrait.
- Pour *First Fit*, par exemple, si nous avons N blocs de mémoire alloués, nous aurions en même temps 0.5*N blocs inutilisables dus à la fragmentation (50-percent rule!)

Segmentation

- Segmentation : c'est un mécanisme qui permet de faire un mappage entre l'environnement du programmeur (adresses définies par le prog.) et la mémoire physique réelle.

- Un espace d'adressage logique peut être représenté comme un ensemble de segments de tailles différentes.



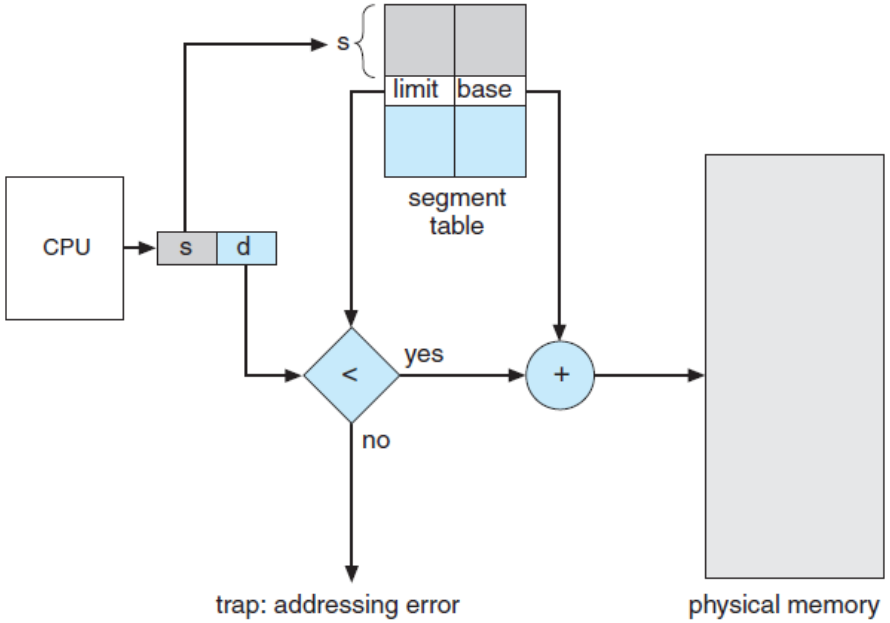
Segmentation

- Une adresse logique consiste de : <**num. du segment**, **offset**>
- La **table de segments** permet de mapper une adresse logique (adresse de dim. 2, définie par le prog.) en une adresse physique

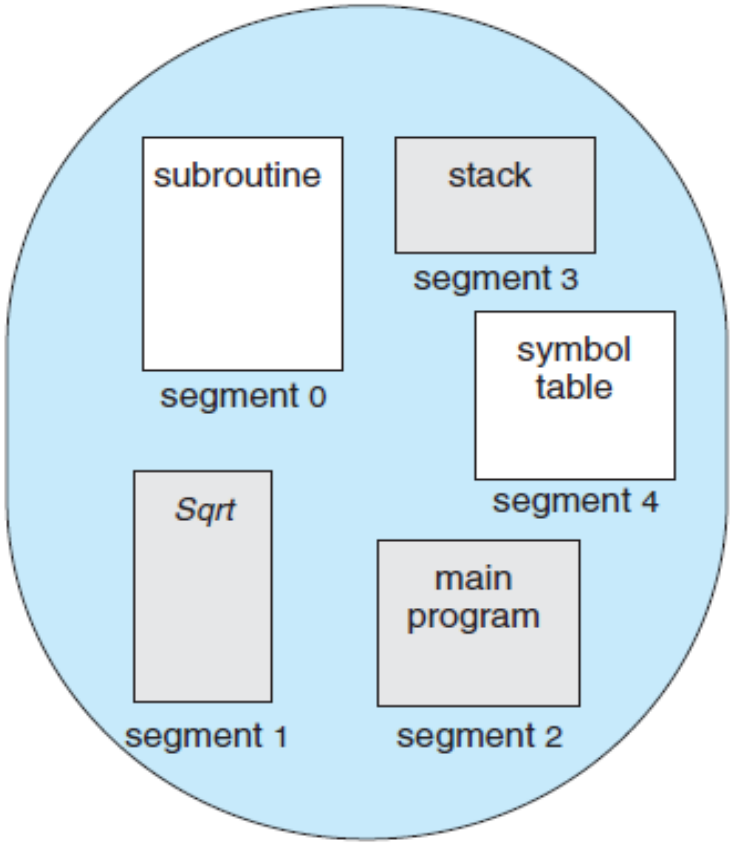
- Adresse logique <***s,d***> :
- ***s*** joue le rôle d'un index et pointe sur une entrée dans la table de segments. On obtient la base (le début du segment) et la limite.
- **d** (**déplacement/offset**) doit être entre **0** et la valeur de **limite** du segment (***d < limite***).
- On envoie à la mémoire la **somme** de la valeur de ***d*** et de la **base** (**base+d**).

L'**adresse de base** ou la **base** : La première adresse physique ou le début du segment dans la mémoire

Limite/étendue : la longueur du segment

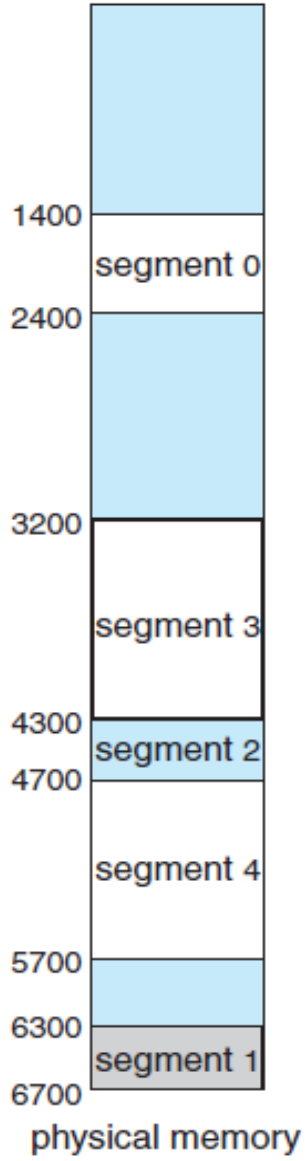


Exemple de segmentation



	limit	base
0	1000	1400
1	400	6300
2	400	4300
3	1100	3200
4	1000	4700

segment table



Segmentation – inconvénients ?

- **Problème de fragmentation externe** : pas suffisamment de place pour placer un nouveau processus même si la taille totale disponible permettrait de le faire. Les blocs d'octets libres ne sont pas contigus.
  - Besoin de faire défragmentation, **mais …**
- Problème de déplacer des segments de différentes tailles de la mémoire sur le disque dur, qui est structuré en des unités de stockage de taille fixe/égale.
  - Temps d'accès lent et donc défragmentation impossible du disque dur.

Pagination

- La méthode de base consiste a découper la mémoire physique en des blocs de taille fixe, appelés **frames** (ou **cadres de pages**), et découper la mémoire logique en des blocs de la même taille, appelés **pages**.

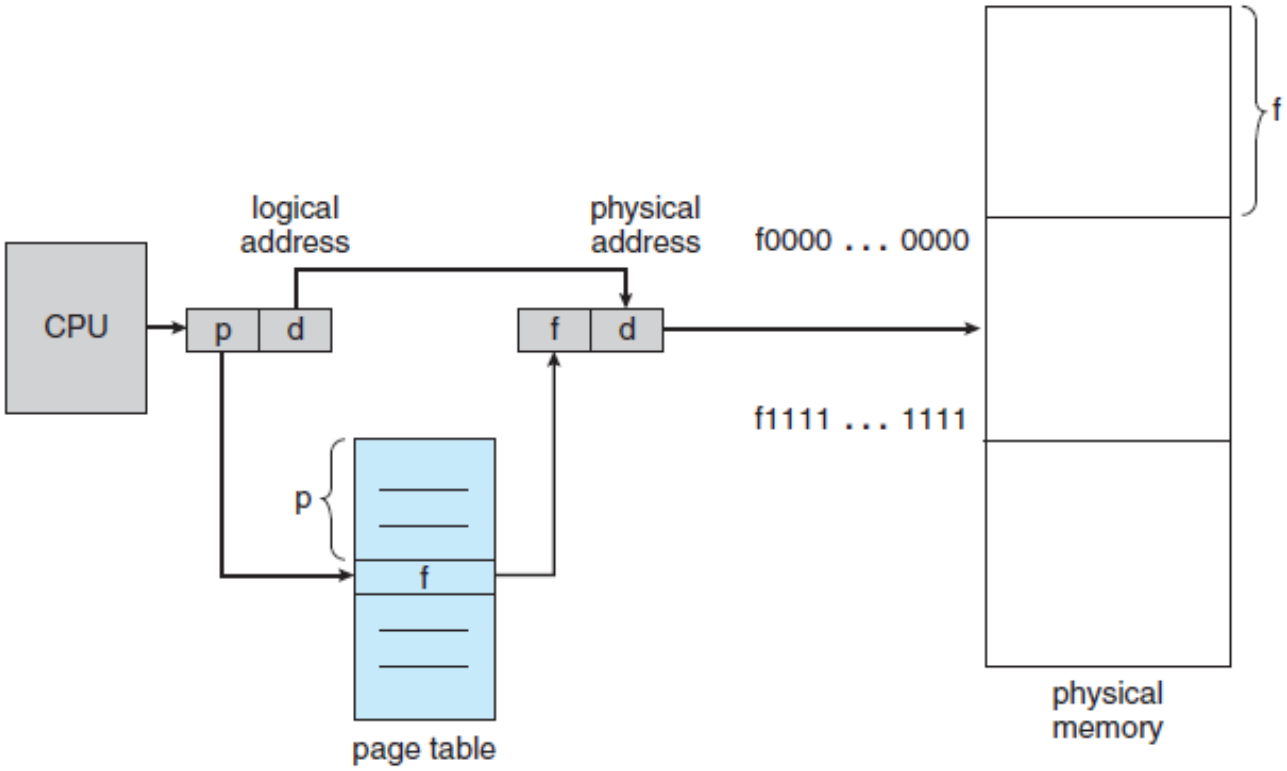
- Quand un processus devrait être exécuté, ses pages seront chargées dans les frames de mémoire physique disponibles.

- **Avantage** : l'espace d'adressage physique est complètement séparé de l'espace d'adressage logique. Donc, un processus peut avoir un espace d'adressage logique de *64 bits* même si la taille de la mémoire physique est inferieure a *2^64 bytes*.

IUT DE PARIS
UNIVERSITÉ PARIS DESCARTES

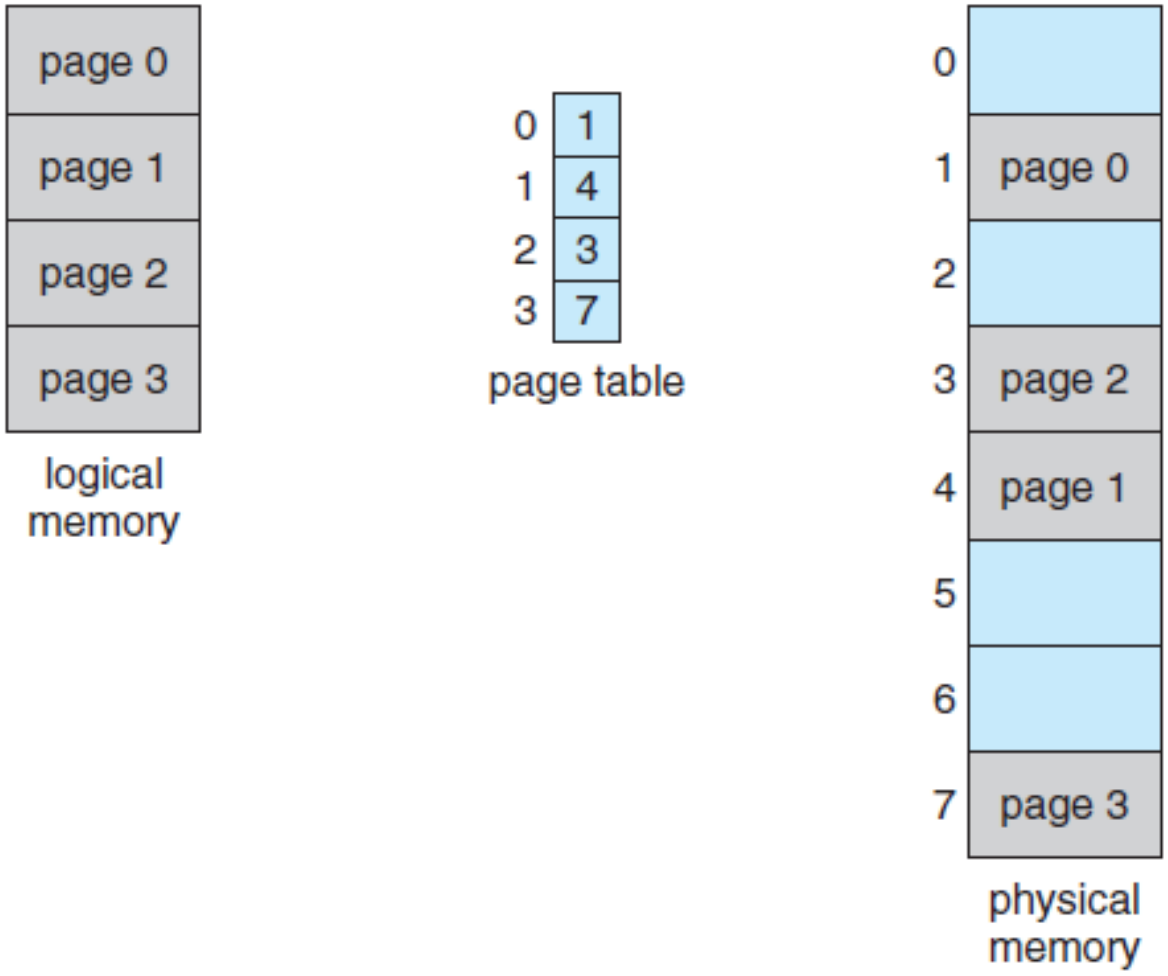
Pagination – en pratique

- L'adresse logique est composée de 2 champs : **p** et **d**
- **p** : le numéro de la page
- **d** : l'offset
- La taille de la page (comme la taille du frame) est définie/fixée par le hardware.
- La taille d'une page est définie en tant que puissance de 2
- Elle varie entre 512 octets et 1 GO, et dépend de l'architecture de l'ordinateur.



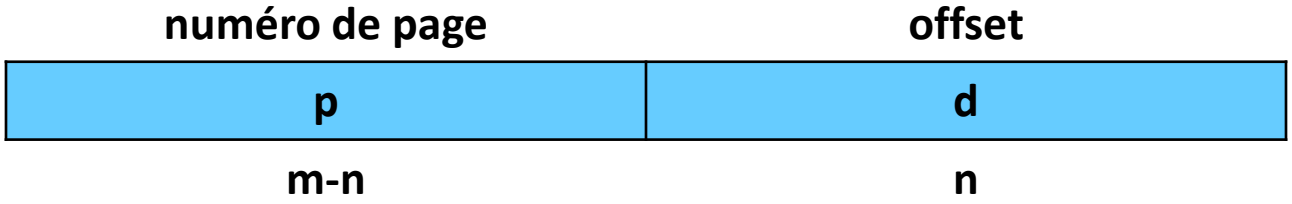
Un exemple de pagination

- page 0 est mappée vers frame 1
- Page 1 → frame 4
- Page 2 → frame 3
- Page 3 → frame 7



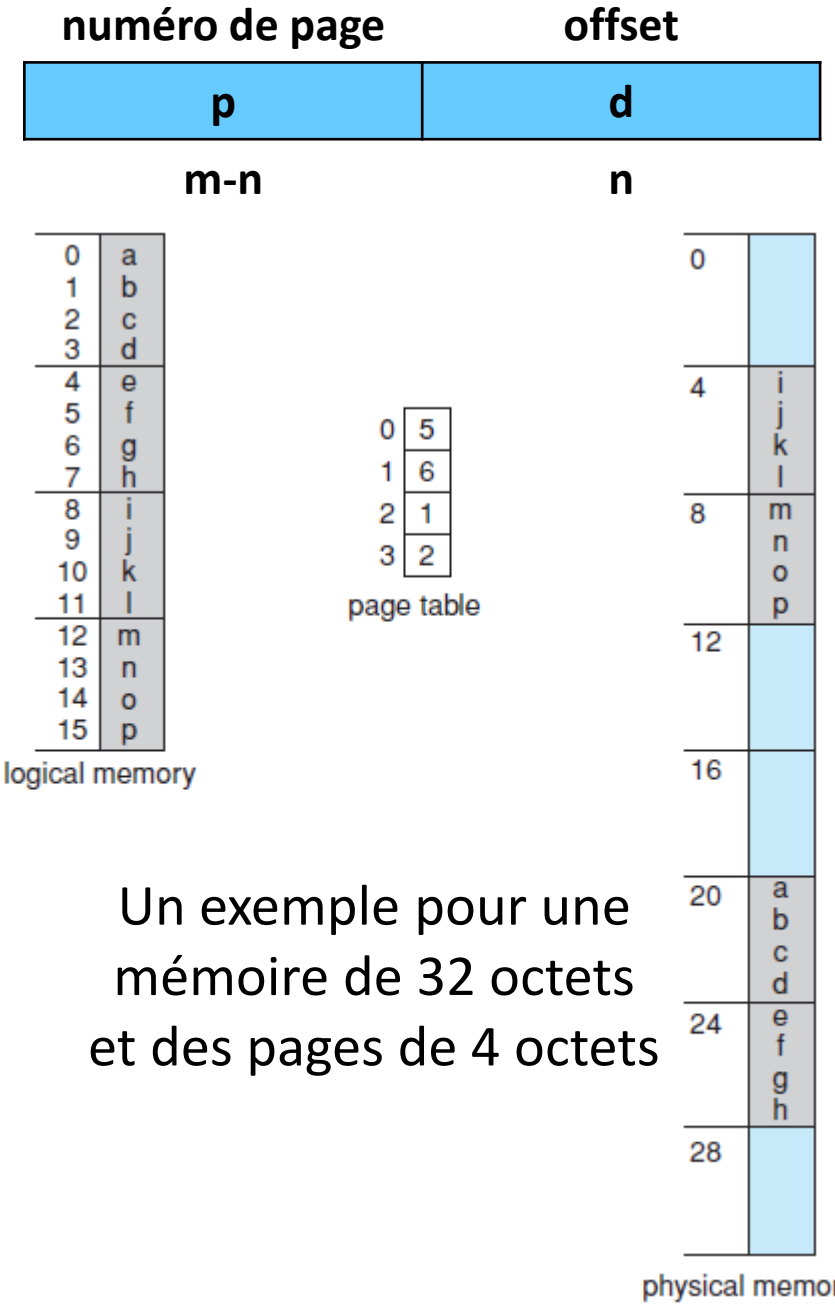
Pagination – en pratique

- Si la taille de l'espace d'adressage logique est **2^m** et la taille de la page est **2^n** on a :
- **m-n** bits les plus significatifs de l'**adresse logique** désigne le numéro de la page
- **n** bits les moins significatifs de cette adresse désigne le déplacement/l'offset dans la page.



Un autre exemple de pagination

- Cas ou *n=2*, *m=4* et taille de la mémoire physique est 32 octets (donc en total 8 pages/frames) :
- L'adresse logique 0 → page 0, offset 0.
- page 0 est mappée vers frame 5 (voir table des pages)
- L'adresse logique 0 est mappée vers l'adresse physique 20 [= (5*4)+0]
- L'adresse logique 13 → page 3, offset 1. page 3 → frame 2. Donc l'adr. log. 13 est mappée vers l'adr. phys. 9 [=(2*4)+1]



Un exemple pour une mémoire de 32 octets et des pages de 4 octets

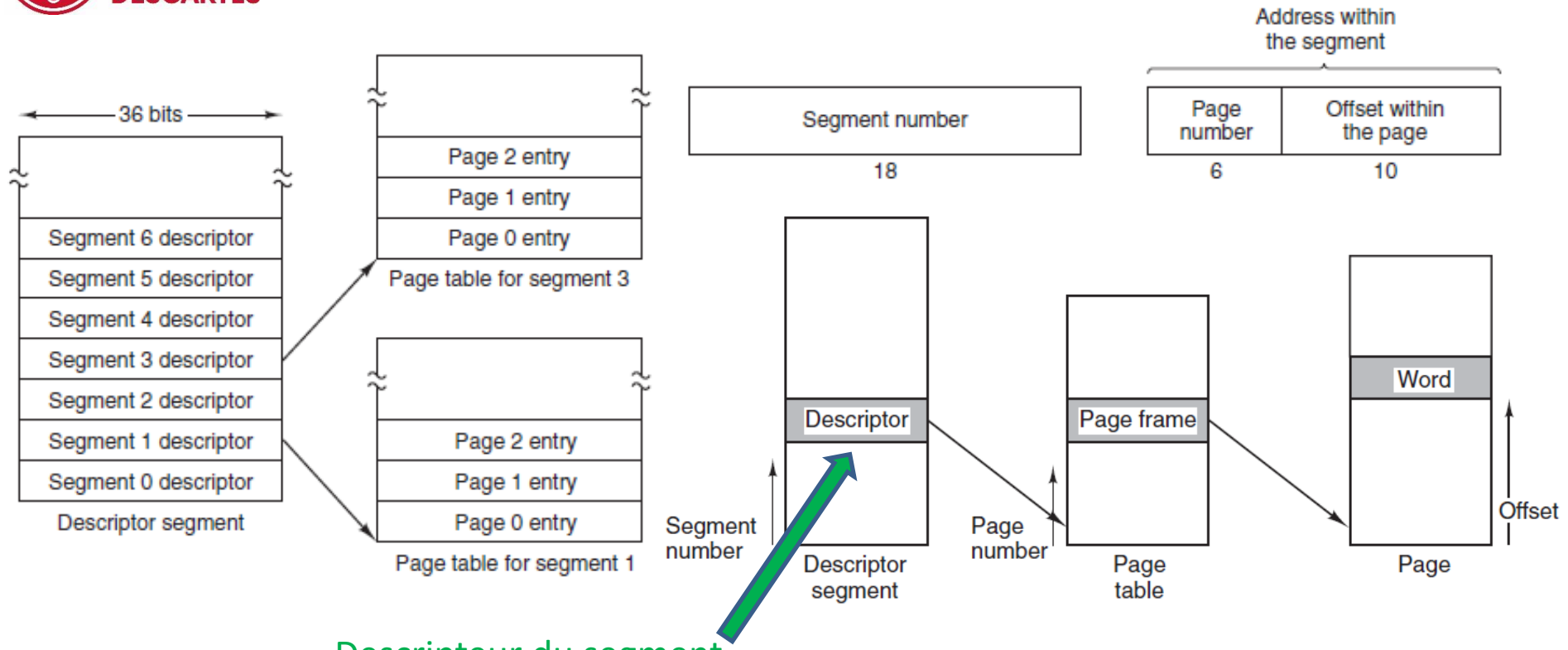
Segmentation et/ou pagination ?

- Le MULTICS (1964-2000) : parmi les premiers ou même le premier système d'exploitation qui a utilisé une **architecture de segments paginés** (ou segmentation + pagination).

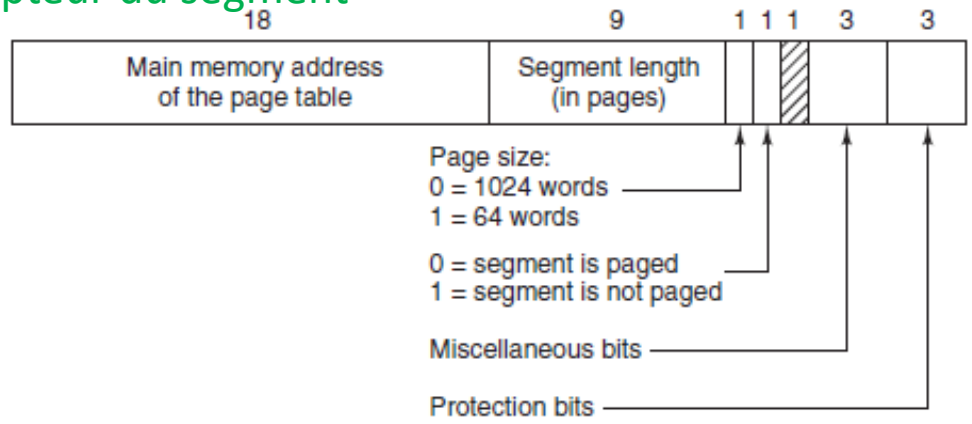
- Les microprocesseurs Intel 8086 (16 bits) et Intel 8088 (16 bits) sont basés sur une **architecture segmentée**.

- Les microprocesseurs 32 bits (IA-32) et 64 bits (x86-64) utilisent la **segmentation avec la pagination**.

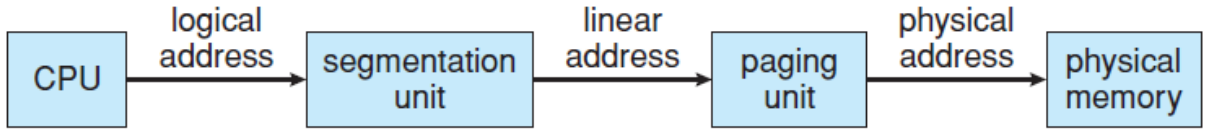
Exemple (1) de Pagination + Segmentation : **MULTICS**



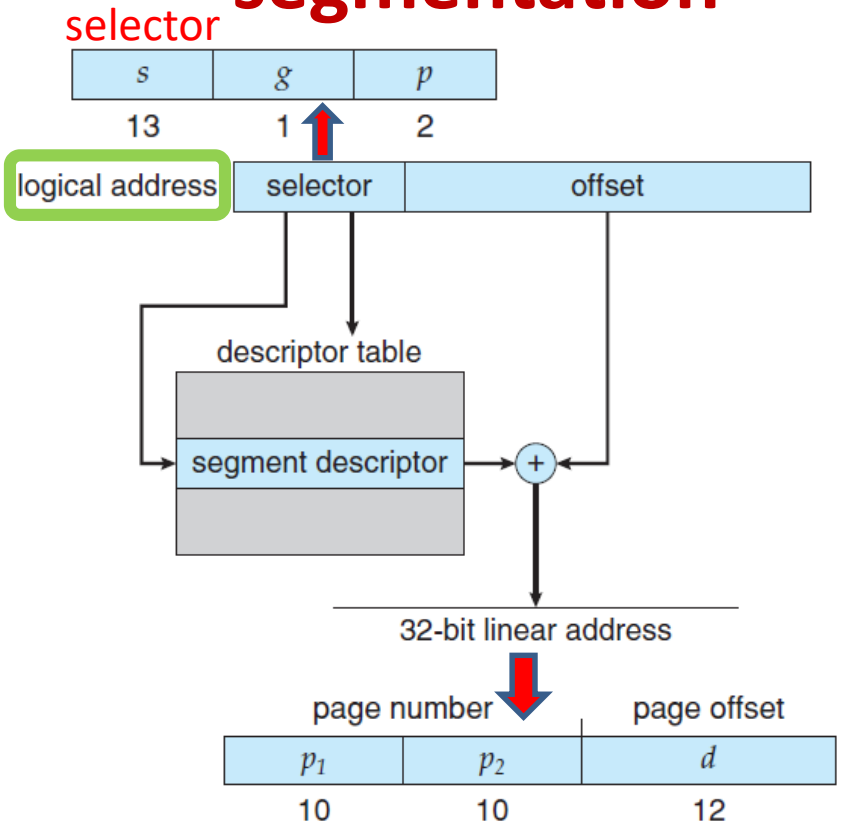
Descripteur du segment



Exemple (2) de Pagination + Segmentation : IA-32



segmentation



pagination

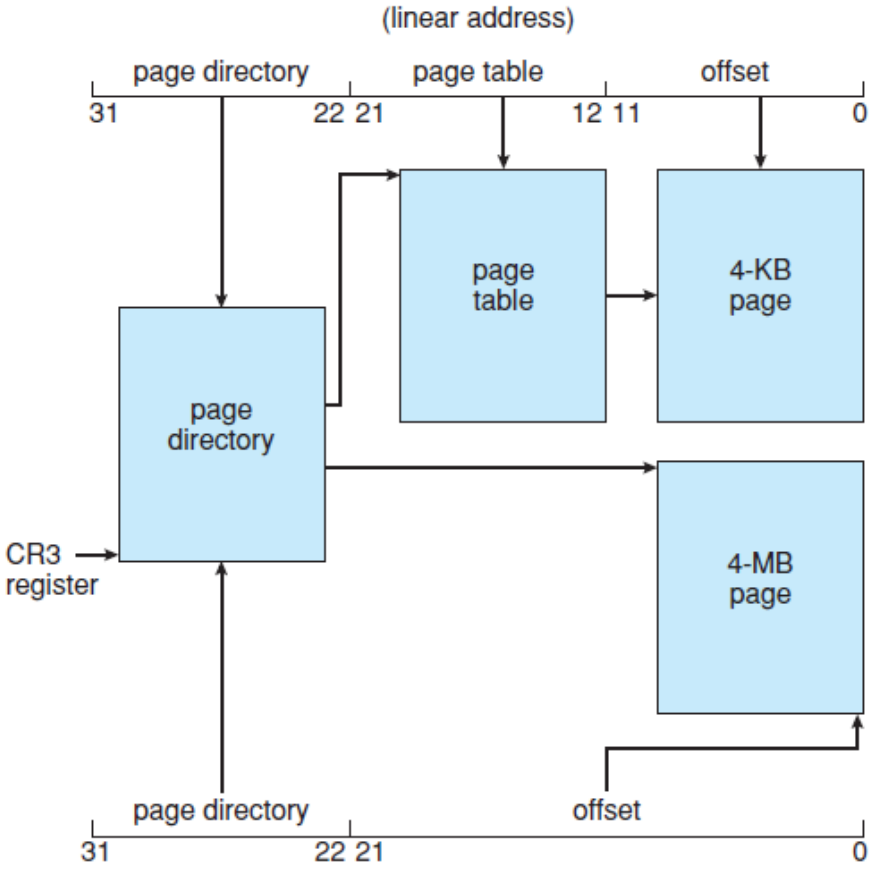


Table des pages - implémentation

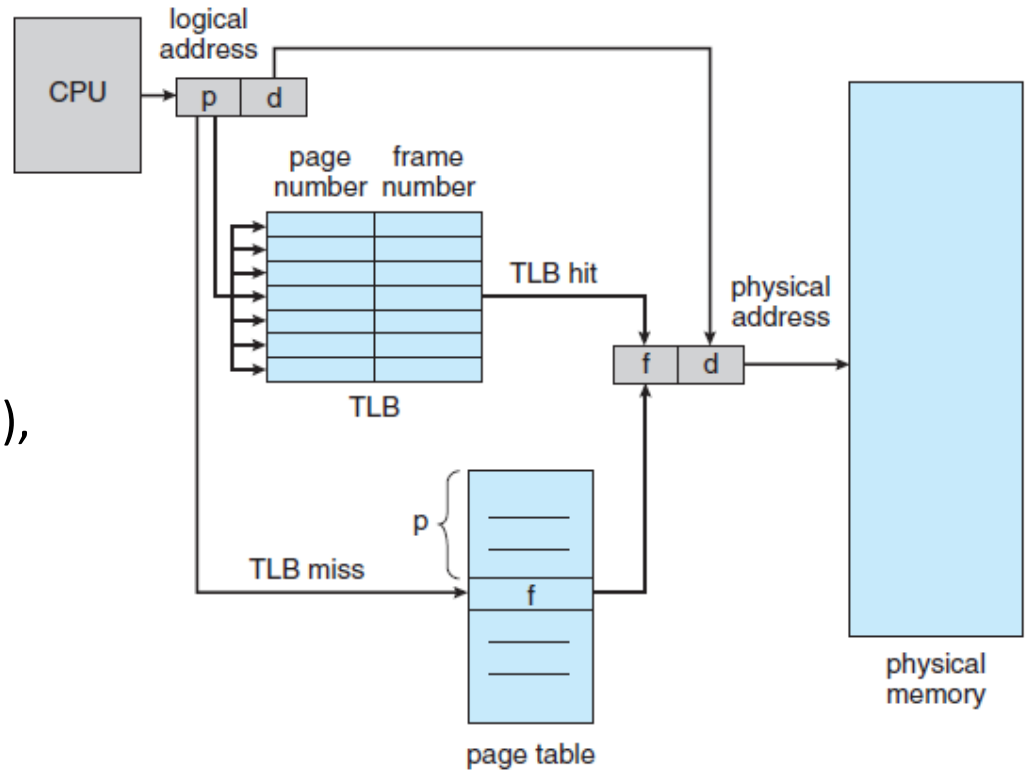
- Chaque OS a sa propre méthode pour stocker les tables des pages; un pointeur vers la table des pages sauvegardée dans le bloc de control du proc. ou des tables fournies par le OS
- L'implémentation de point de vue hardware :

1. un ensemble de **registres dédiés**. Ces registres doivent être construits très rapidement pour permettre un mappage rapide et efficace d'adresses.
   - **Inconvénient** : Difficile de gérer des tables de pages de grande taille
2. Si la table des pages est grande, celle-ci sera sauvegardée en mémoire et un **page-table base-register (PTBR)** pointe vers cette table.
   - **Inconvénient** : nous avons besoin de 2 accès mémoire pour obtenir l'octet désiré et donc l'accès en mémoire est ralenti d'un facteur de 2.

Table des pages

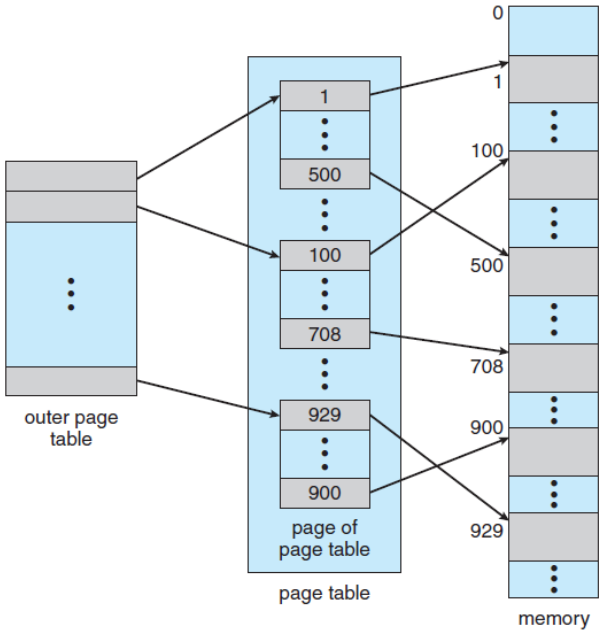
3. Un **translation look-aside buffer (TLB)** : c'est une sorte d'une cache spéciale, petite et très rapide.

   - La TLB est de petite taille. Elle contient entre 32 et 1024 entrées
   - Chaque entrée comporte une paire *<clef,valeur>*
   - Si le numéro de la page se trouve dans la TLB (**TLB hit**), le numéro du frame est renvoyé
   - Sinon (**TLB miss**), on va chercher le numéro de la page dans la table des pages



Structure de la table de pages

- Pagination hiérarchique



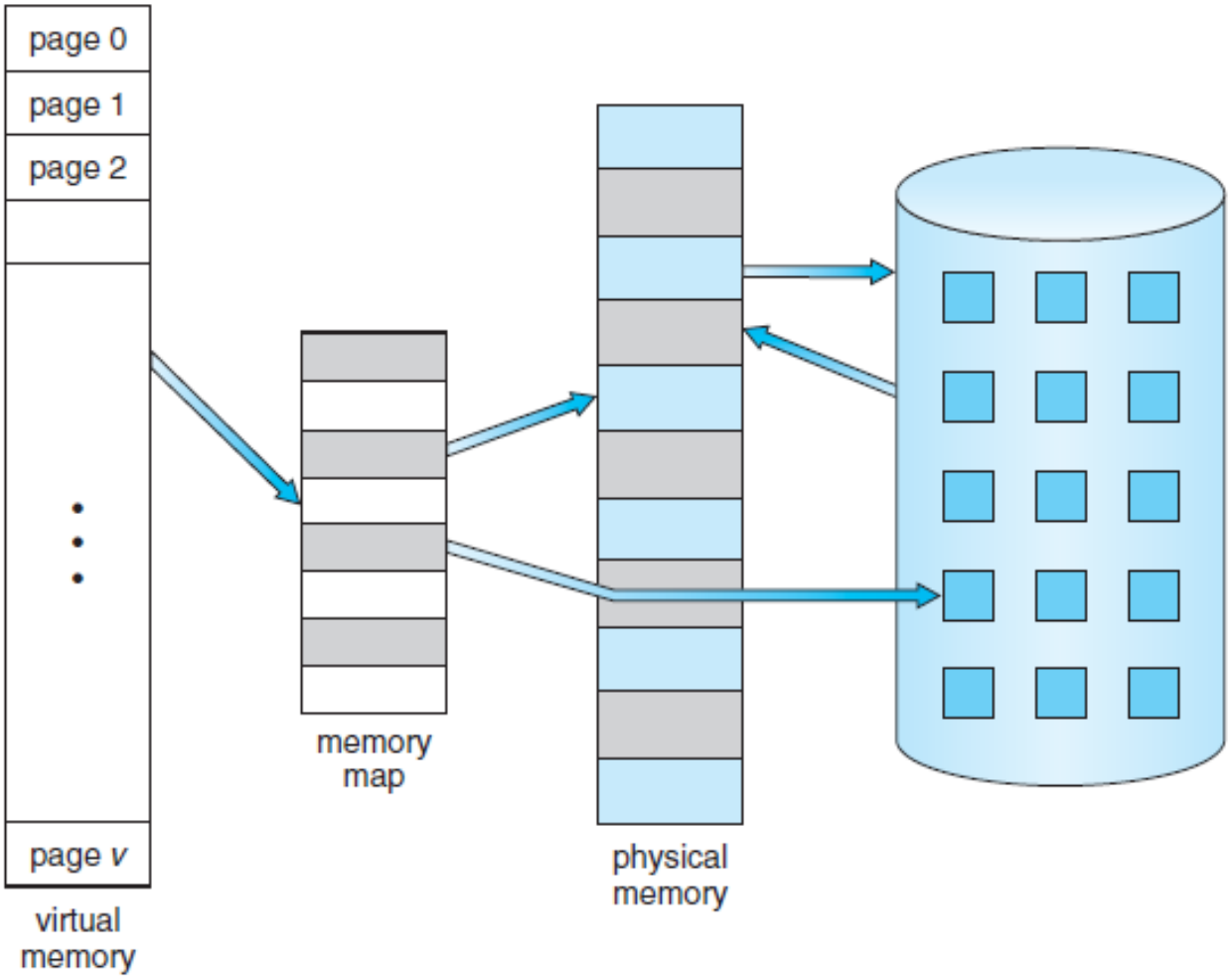
- Pagination Hashed page tables
- Table de pages inversée …

Mémoire virtuelle

Idée de base

- La mémoire virtuelle est une technique qui permet d'exécuter des processus qui ne sont pas entièrement dans la mémoire.
- Un des avantages majeurs de cette technique est que les programmes peuvent être plus grands que la taille de la mémoire physique.
- Elle abstrait la mémoire en un grand nombre d'unités de stockage, séparant la mémoire logique (celle vue de l'utilisateur/du programmeur) de la mémoire réelle physique.

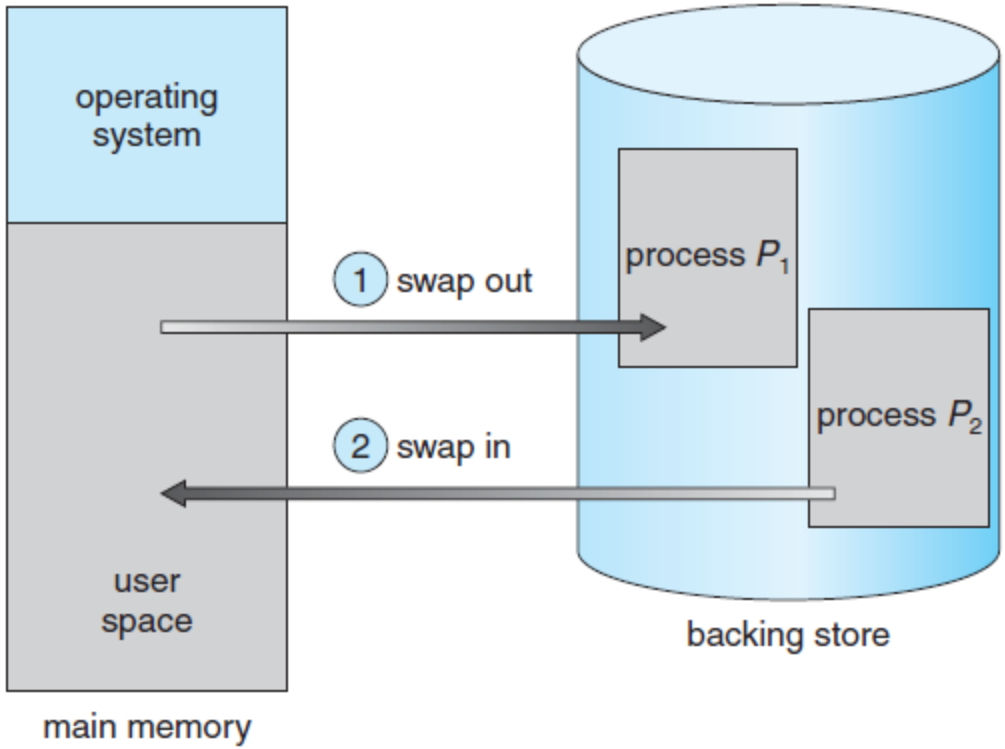
Mem. Virt. > Mem. Phys.



- La mémoire virtuelle est plus grande de la mémoire physique

Swapping

- Un processus doit être chargé en mémoire pour être exécuté.
- Toutefois, un processus peut être transféré temporairement hors de la mémoire vers une unité de stockage (le disque dur) et rechargé dans la mémoire plus tard pour exécution.
- Le **swapping** permet d'avoir un espace total d'adressage de tous les processus qui est supérieur à la taille réelle de la mémoire physique
- Augmentation du degré de multiprogrammation dans le système



Swapping --- inconvénients ?

- **Le temps de Swapping n'est pas négligeable**
  - Exemple : pour un processus de 100 MO et un taux de transfert du disque de 50 MO/seconde, il nous faut 2 secondes pour le swapping …
- Le processus doit être complètement idle

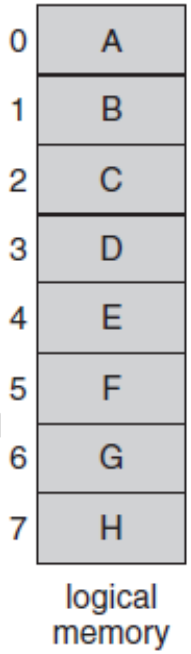
- Le swapping standard n'est pas actuellement utilisé dans les OS modernes
- Les OS pour dispositifs mobiles ne possèdent pas du swapping en aucune forme. En général, les appareils mobiles utilisent la mémoire *flash* plutôt que des disques durs pour leur stockage persistant.

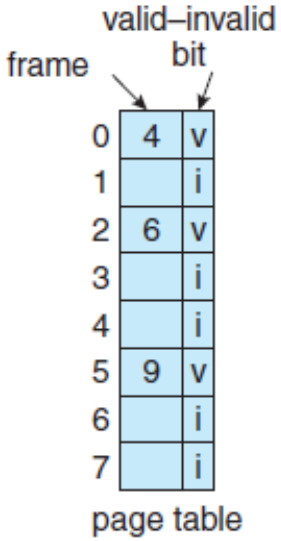
Pagination à la demande

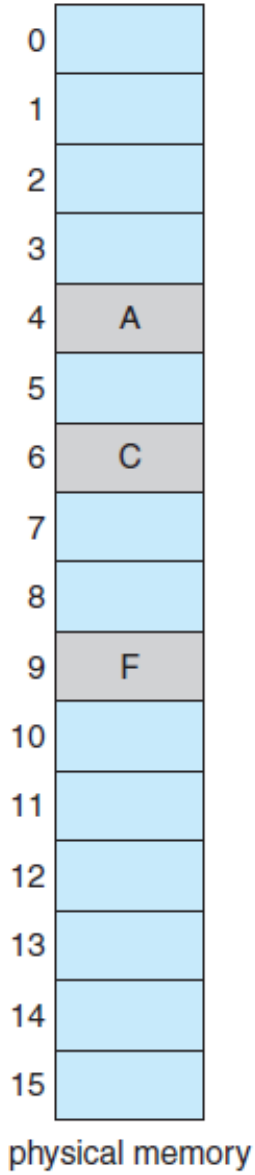
- Considérons maintenant comment un exécutable peut être chargé du disque en mémoire.
- Charger l'intégralité du programme ? Est-ce que nous avons vraiment besoin de le charger tout entier ?
- **Idée de base** de la ***pagination à la demande*** est de charger la page en mémoire quand on en a besoin. Cette technique est communément utilisée dans les systèmes des mémoires virtuelles.
- **Swapper** → processus tout entier ; **Pager** (ou **lazy swapper**) → une page correspondant au processus : **similitude**

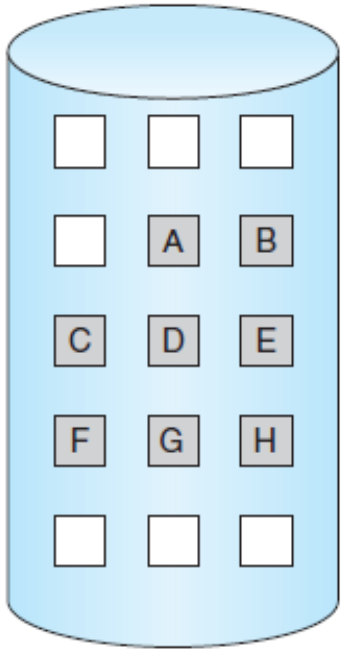
Pagination à la demande – concept de base

- **v** : **valid** → la page est légale et en mémoire
- **i** : **invalid** → la page est illégale (n'appartient pas à l'espace d'adr du processus) ou elle se trouve actuellement sur le disque

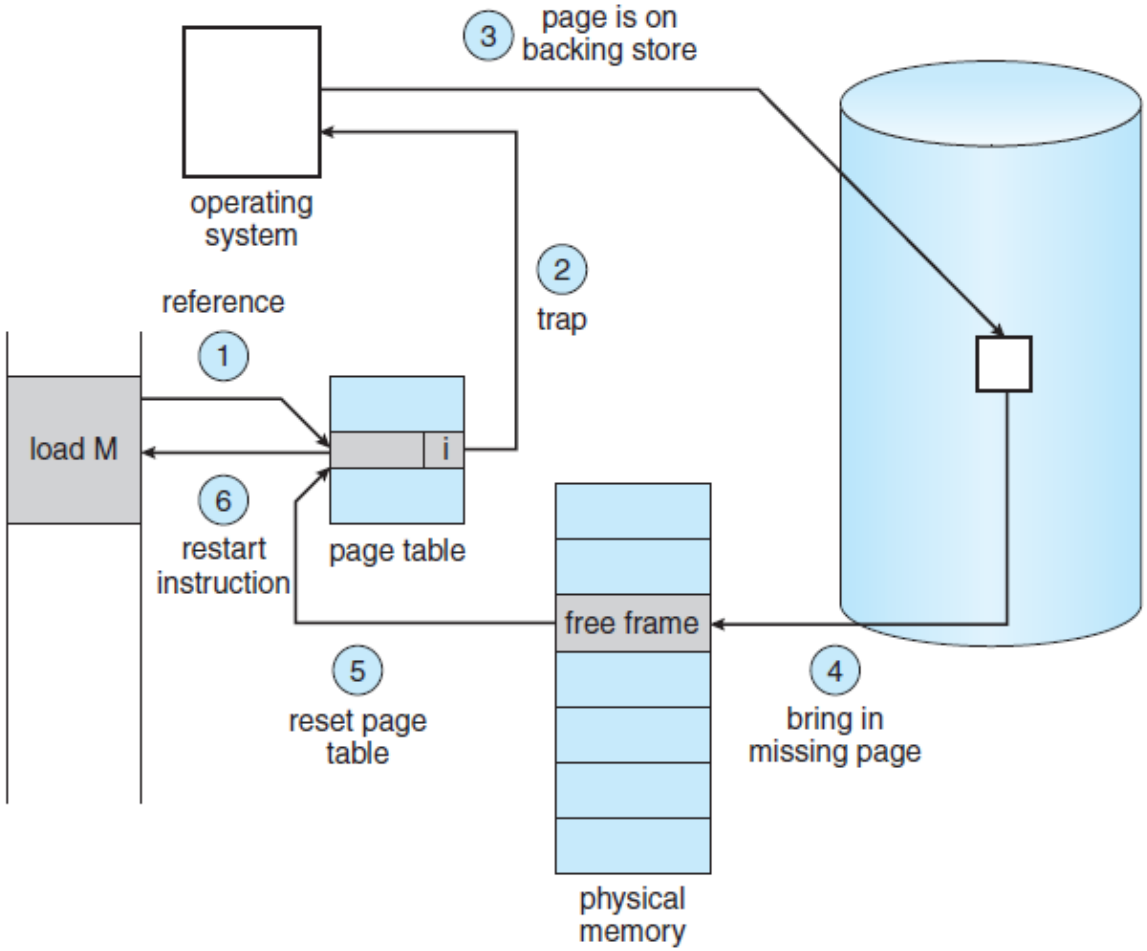








Gestion de l'erreur de page (page fault)



- Les différentes étapes de gestion de l'erreur de page
- Un exemple est donné dans cette figure

Algorithmes de remplacement de page

- FIFO → La plus ancienne
  - Garder trace de l'heure de chargement de la page en mémoire
- OPT → choisir d'une manière optimale la page à remplacer (benchmark)
- LRU (Least Recently Used) → La page la moins récemment utilisée
  - Garder trace de l'heure de la dernière utilisation de la page
- Différentes approximations du LRU
- Least frequently used (LFU)
- Most frequently used (MFU) …