Système d'exploitation
Gestion Mémoire

Licence Informatique

Jean-Louis Lanet / Guillaume Bouffard / David Pequegnot
Jean-louis.lanet@unilim.fr

Université de Limoges
FACULTÉ DES SCIENCES ET TECHNIQUES

Plan

- Besoin de gestion mémoire
- Pagination
- Segmentation
- Problèmes de sécurité

Techniques de gestion mémoire

- Partitionnement à taille fixe
  - Divise la mémoire en partitions lors du boot, les tailles peuvent être identiques ou pas mais fixes,
  - Mécanisme simple soufrant de la fragmentation interne
- Partitionnement à taille variable
  - Les partitions sont créées lors du chargement des programmes
  - Ne crée pas de fragmentation interne mais externe,
- Pagination simple
  - Divise la mémoire en pages de taille fixes et charge les programmes dans les pages disponibles
  - Pas de fragmentation externe, mais légère fragmentation interne

Techniques de gestion mémoire

- Segmentation simple
  - Divise les programmes en segments
  - Pas de fragmentation interne, faible fragmentation externe
- Mémoire virtuelle paginée
  - Basée sur un mécanisme de pages, mais pas toutes en mémoire centrale,
  - Autorise un vaste espace de mémoire virtuelle
  - Surcout d'exécution
- Mémoire virtuelle segmentée
  - Basée sur un mécanisme de segments, mais pas toutes en mémoire centrale,
  - Facilité pour partager des modules.

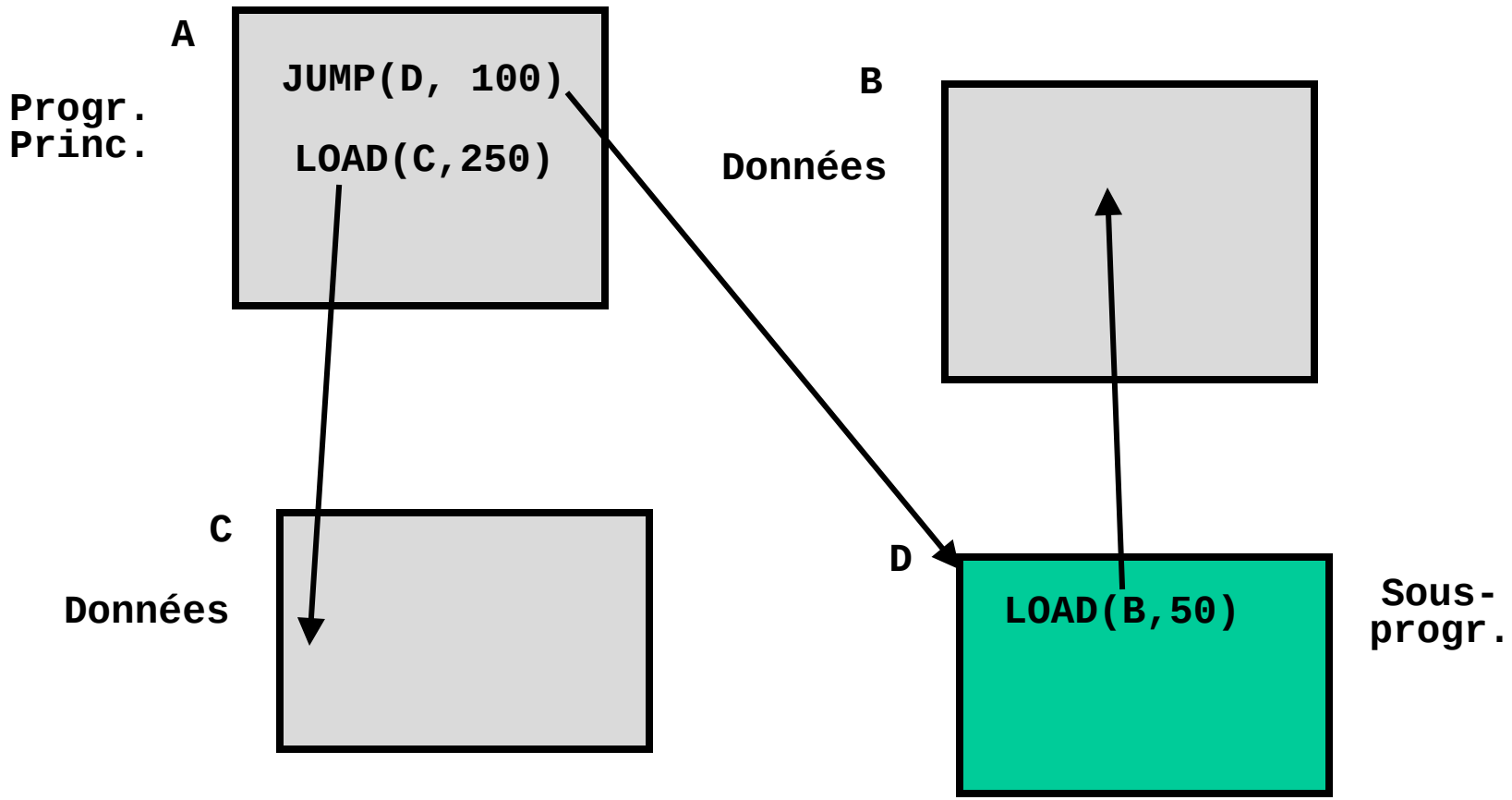
Segmentation

- Un segment est un ensemble d'information considéré comme une unité logique
- La mémoire est coupée en régions appelées segments,
- Tous les segments n'ont pas la même taille,
- Comme pour la pagination, la segmentation utilise un numéro de segment,
- A l'intérieur d'un segment, les informations sont désignées par un déplacement qui est une adresse relative.

Segmentation

- Les segments sont utilisés :
  - Comme unité de découpage logique d'un programme
  - Comme unité de partage entre plusieurs utilisateurs,
  - Comme unité de protection, le segment est l'entité à laquelle sont attachés des droits d'accès,
- Un descripteur est attaché à chaque segment,
  - Adresse d'implantation du segment,
  - Les droits d'accès,
  - Sa taille

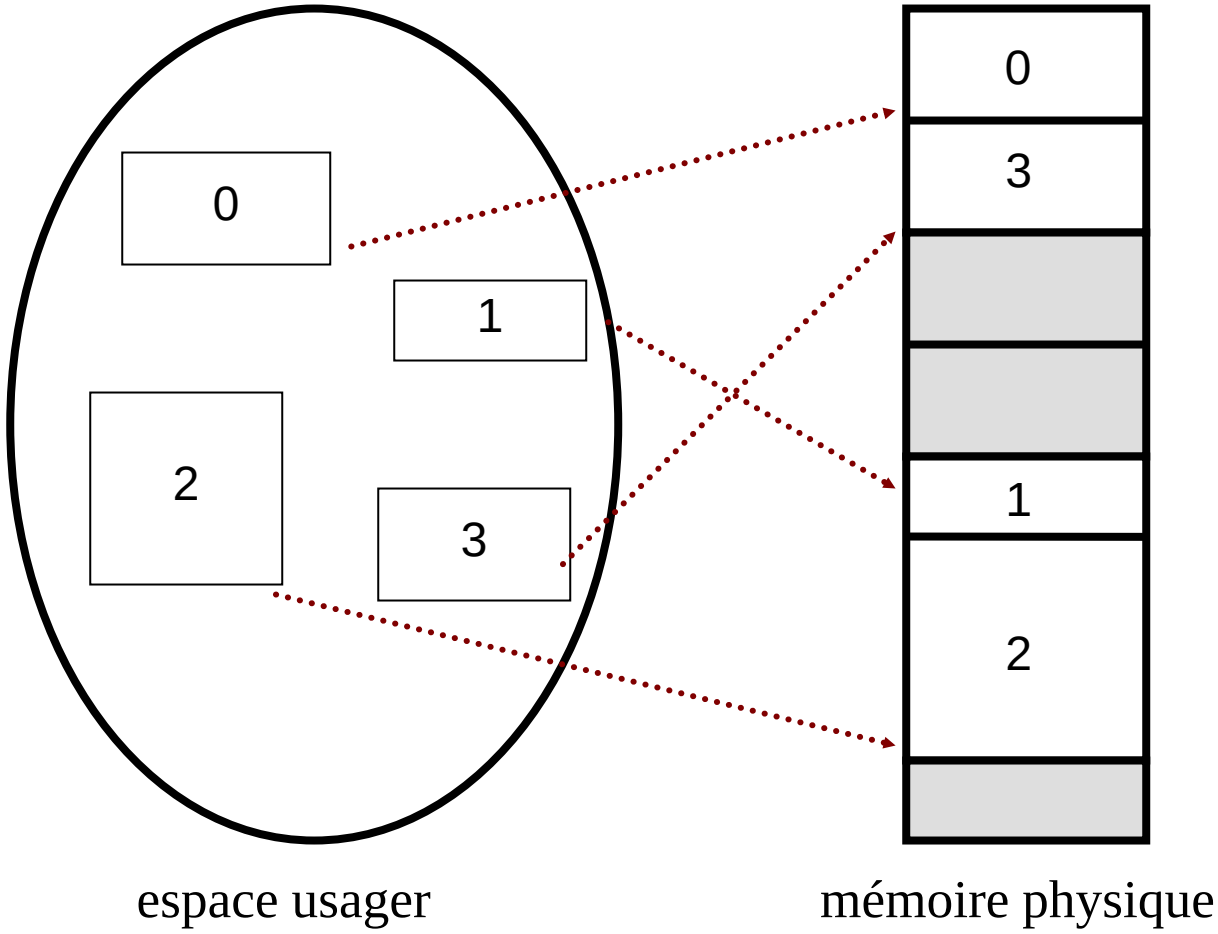
Les segments sont des parties logiques du programme



4 segments: A, B, C, D

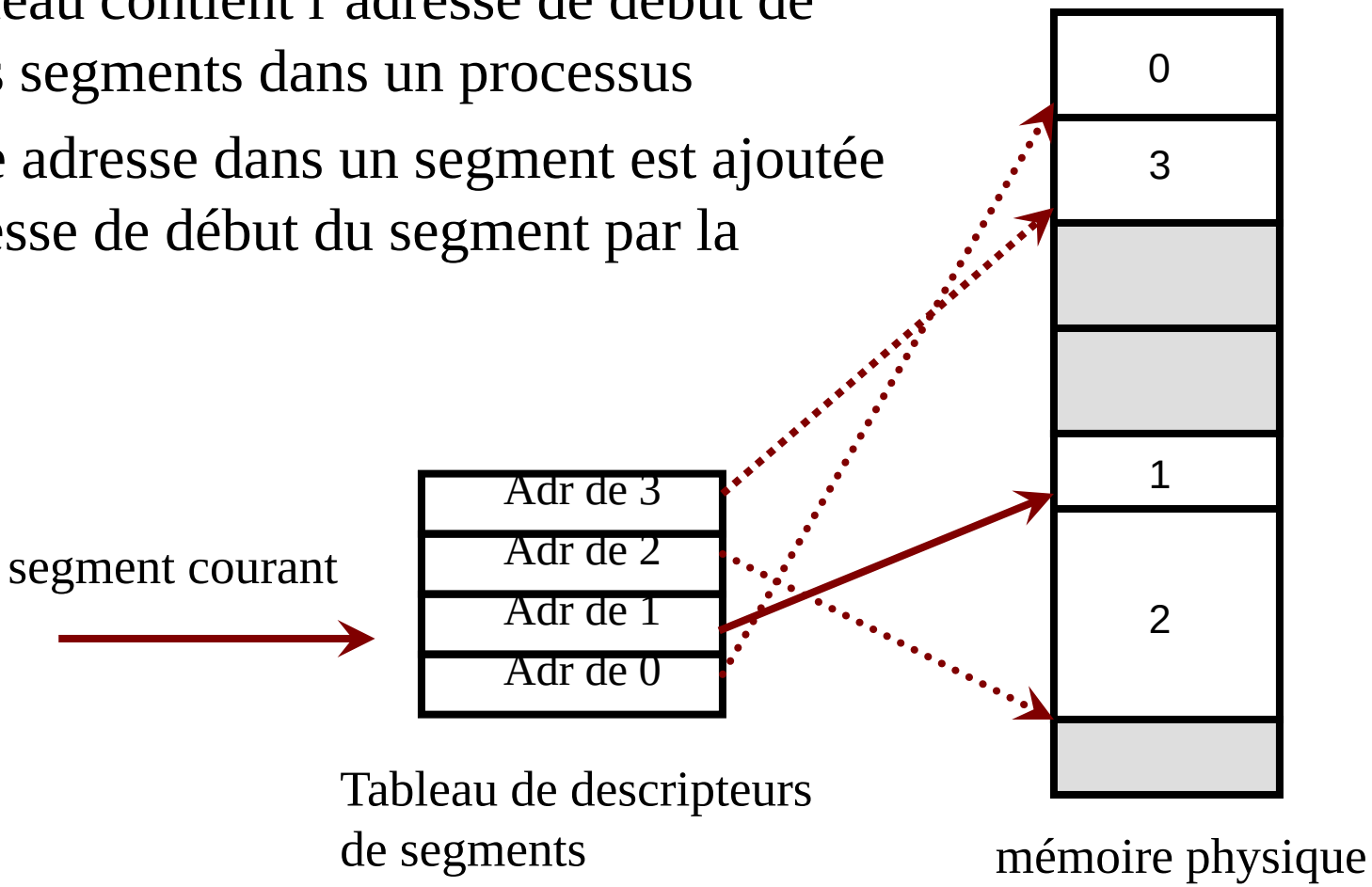
Les segments comme unités

Étant donné que les segments sont plus petits que les programmes entiers, cette technique implique moins de fragmentation (qui est externe dans ce cas)



Mécanisme pour la segmentation

- Un tableau contient l'adresse de début de tous les segments dans un processus
- Chaque adresse dans un segment est ajoutée à l'adresse de début du segment par la MMU

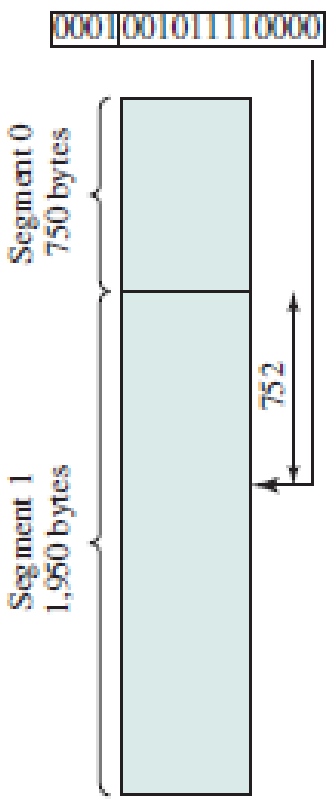


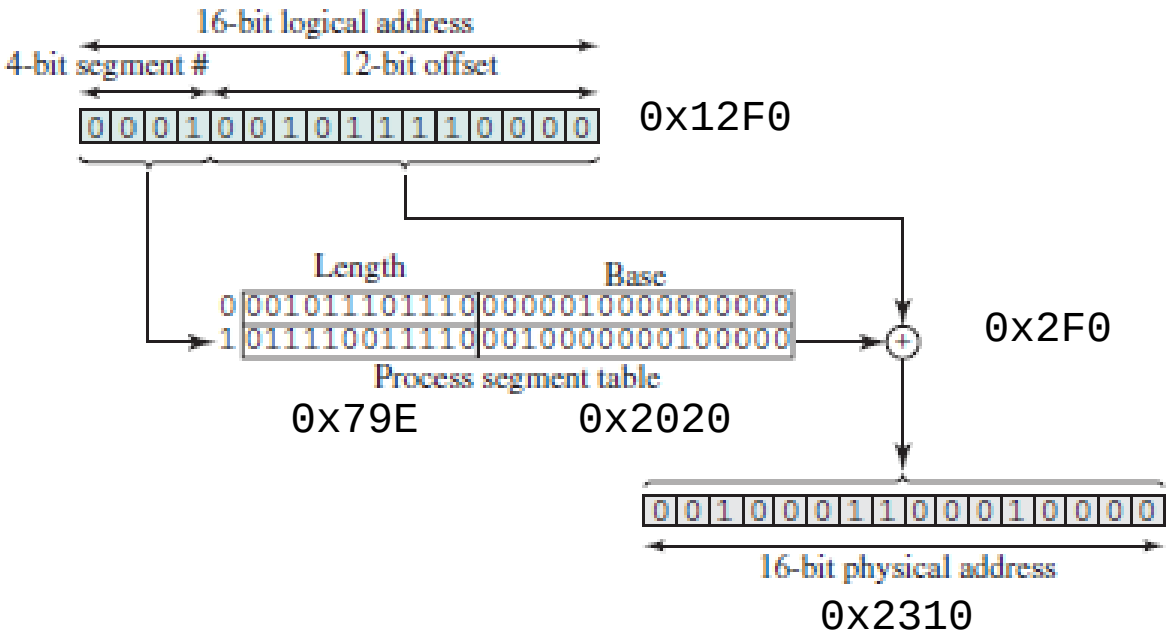
Détails

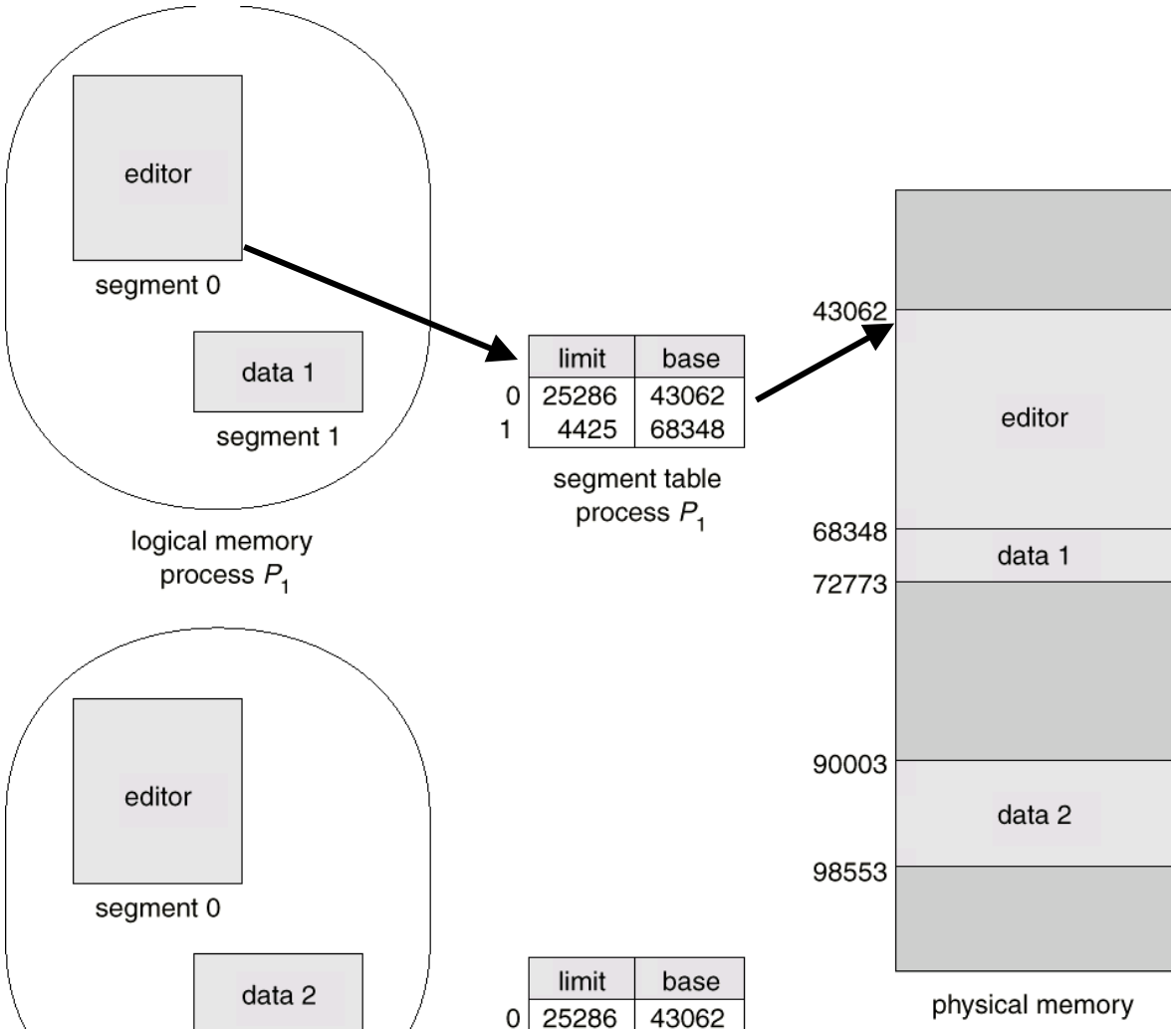
- L’adresse logique consiste d’une paire: <No de segm, décalage>
- Le tableau des segments contient des descripteurs de segments
  - adresse de base
  - longueur du segment
  - Infos de protection…
- Pour le processus il y aura un pointeur à l’adresse en mémoire du tableau des segments
- Il y aura aussi le nombre de segments dans le processus
- Au moment de la commutation de contexte, ces infos seront chargées dans les registres appropriés de la MMU

Détails

taille(n)= 4,
taille(m)=12

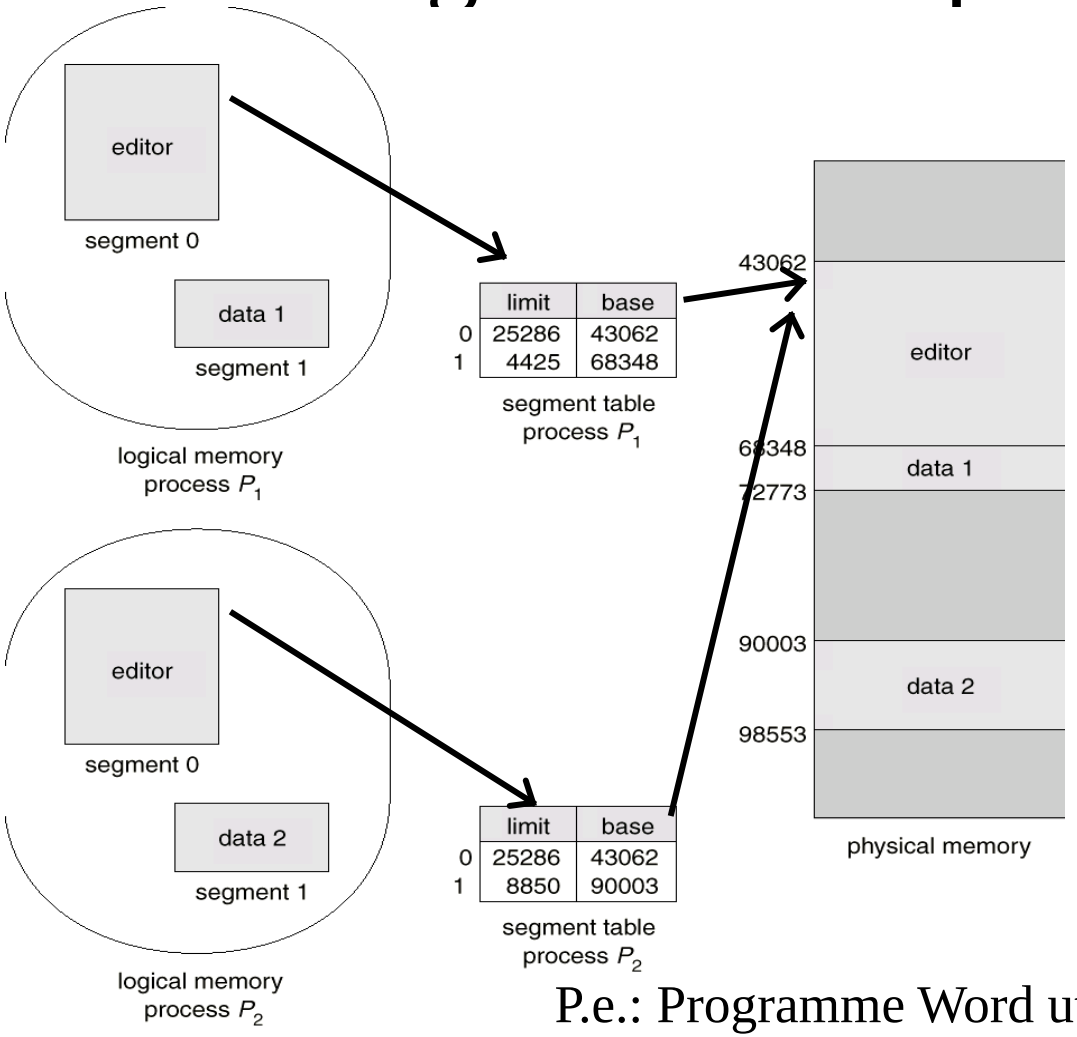






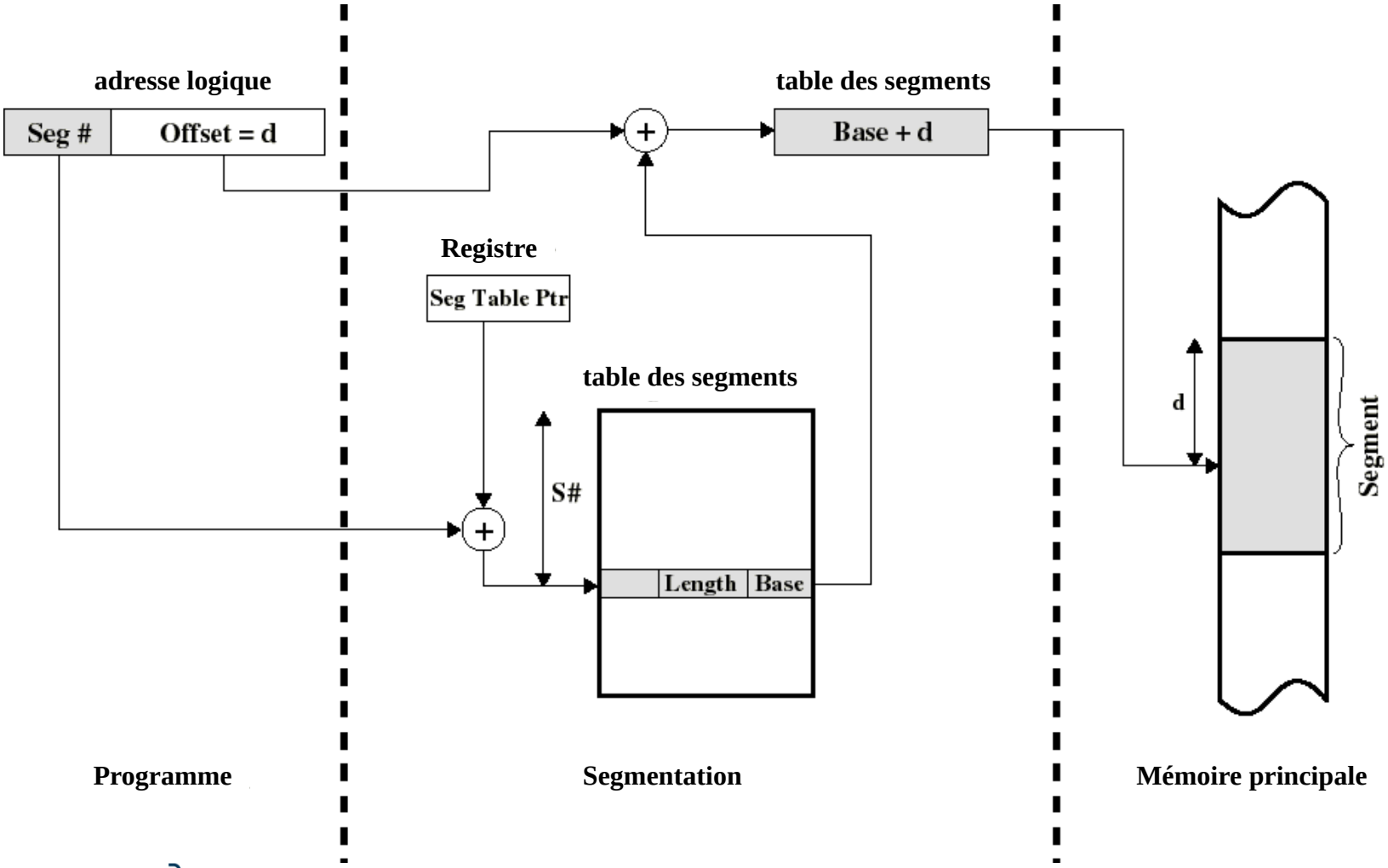
1 seule copie en mémoire du segment partagé

Partage de segments: le segment 0 est partagé



P.e.: Programme Word utilisé pour éditer différents documents , DLL utilisé par plus usagers

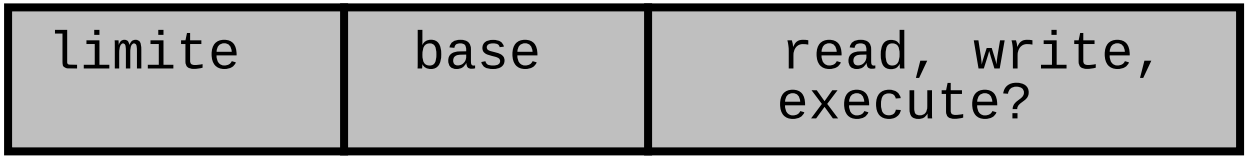
Traduction d'adresses par segmentation



Aussi, si d > longueur: segmentation fault

Segmentation et protection

- Chaque *descripteur de segment* peut contenir des infos de protection:
  - longueur du segment
  - privilèges de l'usager sur le segment: lecture, écriture, exécution
    - Si au moment du calcul de l'adresse on trouve que l'usager n'a pas droit d'accès→interruption
    - ces infos peuvent donc varier d'usager à usager, par rapport au même segment!



Évaluation de la segmentation simple

- Avantages: l'unité d'allocation de mémoire est
  - Plus petite que le programme entier
  - Une entité logique connue par le programmeur
  - Les segments peuvent changer de place en mémoire
  - La protection et le partage de segments sont aisés (en principe)
- Désavantage: le problème des partitions dynamiques:
  - La fragmentation externe n'est pas éliminée:
  - Trous en mémoire, compression?
- Une autre solution est d'essayer de simplifier le mécanisme en utilisant unités d'allocation mémoire de tailles égales

Désavantage

- La zone mémoire d'un processus est contiguë (au moins à l'intérieur d'un segment s'il y a des segments).
- Un processus doit être entièrement en mémoire, ou bien, le compilateur doit le segmenter manuellement. *i.e.* prévoir manuellement le chargement/déchargement des segments.
- Il y a possibilité de fragmentation de la mémoire : impossibilité d'allouer une grosse zone alors qu'il y a plein d'espaces libres mais de petites tailles.
- Remarques
  - Les segments partitionnent la mémoire en grosses zones.
  - Les adresses peuvent être référencés par rapport au début du segment (le logiciel ajoute un décalage)

Traduction d'adresses: segmentation et pagination

- Tant dans le cas de la segmentation, que dans le cas de la pagination, nous *ajoutons* le décalage à l'adresse du segment ou page.
- Cependant, dans la pagination, l'addition peut être faite par simple concaténation:

```
11010000+1010
     =
1101 1010
```

Segmentation simple vs Pagination simple

- La segmentation est visible au programmeur mais la pagination ne l'est pas.
- Le segment est une unité logique de protection et partage, tandis que la page ne l'est pas.
- La segmentation requiert un matériel plus complexe pour la traduction d'adresses (addition au lieu d'enchaînement)
- La segmentation souffre de fragmentation *externe* (partitions dynamiques)
- La pagination produit de fragmentation *interne*, mais pas beaucoup (1/2 cadre par programme)

Adresse logique (pagination)

- Les pages sont invisibles au programmeur, compilateur ou assembleur (seules les adresses relatives sont employées)
- Un programme peut être exécuté sur différents matériels employant des dimensions de pages différentes
  - Ce qui change est la manière dont l'adresse est découpée

Techniques de gestion mémoire

- Segmentation simple
  - Divise les programmes en segments
  - Pas de fragmentation interne, faible fragmentation externe
- Mémoire virtuelle paginée
  - Basée sur un mécanisme de pages, mais pas toutes en mémoire centrale,
  - Autorise un vaste espace de mémoire virtuelle
  - Surcout d'exécution
- Mémoire virtuelle segmentée
  - Basée sur un mécanisme de segments, mais pas toutes en mémoire centrale,
  - Facilité pour partager des modules.

Virtualisation

- Pour qu'un processus s'exécute il faut:
  - Son code est dans la mémoire principale,
  - Il a au moins toute la mémoire dont il a besoin,
  - Ses périphériques sont prêts.
- Toutes les instructions du programme sont chargées,
    - Même si certaines parties ne seront jamais exécutées,
    - L'espace d'adressage physique n'est pas forcément contigüe il peut être étalé sur plusieurs pages.

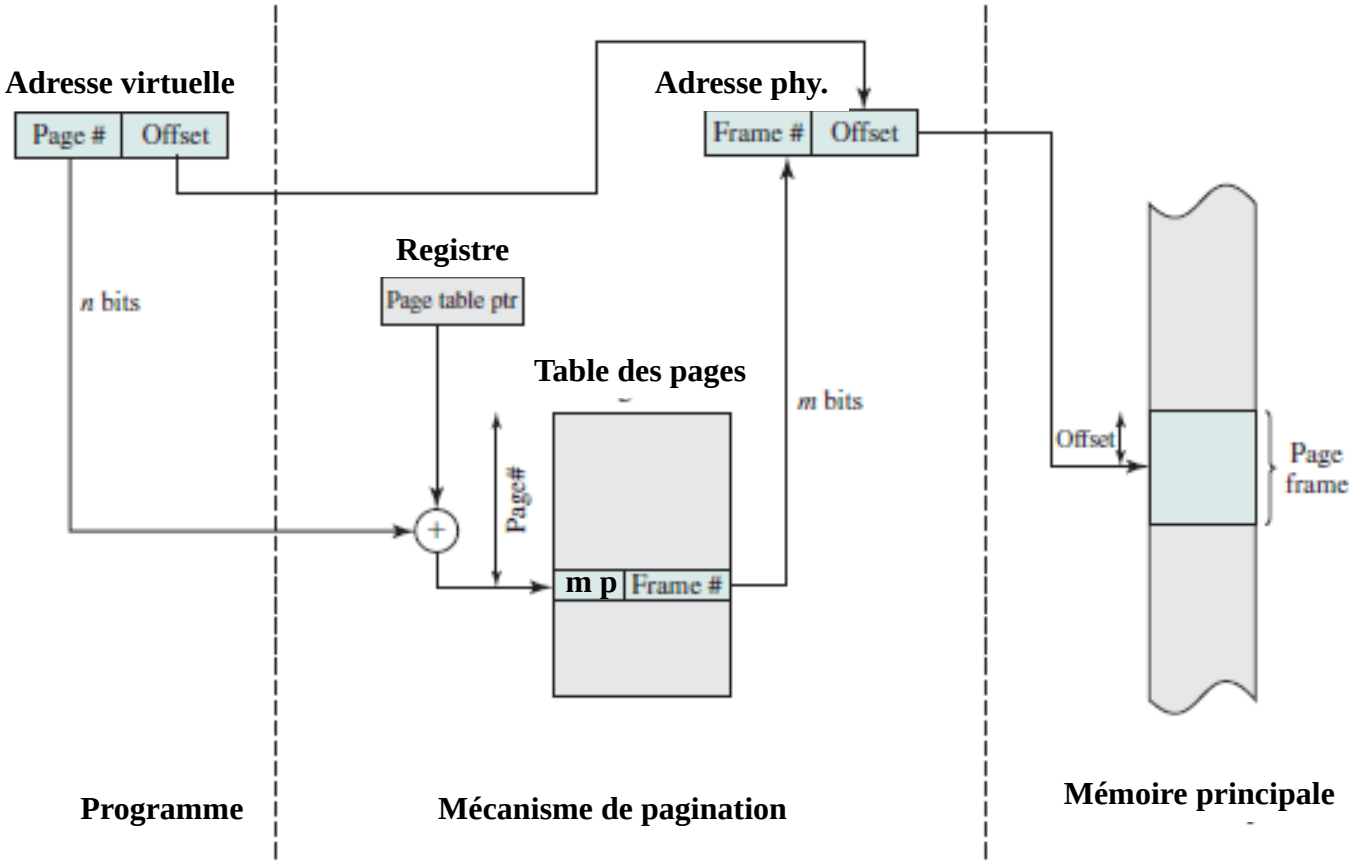
Virtualisation

- Caractéristique de la pagination et segmentation
  - Toutes les références au programme (saut, méthodes) et au données sont relatives,
  - Un processus peut être séparé en plusieurs éléments (pages, segments), pas forcément contigus.
- S'il est possible d'exécuter un programme n'ayant pas toutes ses instructions en mémoire :
  - Potentiellement un espace d'adressage plus vaste
  - Plus de programmes en mémoire centrale
  - Moins d'attente des entrées sorties
  - Programme non utilisé = programme non chargé

Virtualisation

- La mémoire secondaire peut donc être considéré comme une mémoire exécutable virtuelle dont la taille est celle de la mémoire secondaire
- Différence entre mémoire physique et mémoire logique
  - Le matériel est en charge de la translation d'adresse
  - Seule une partie du programme est chargé en mémoire
- Le système doit gérer les fragment absents (page miss) par mise en attente du processus et transfert de la mémoire.

Virtualisation



Problèmes d'efficacité

- La traduction d'adresses, y compris la recherche des adresses des pages et de segments, est exécutée par des mécanismes matériels
- Cependant, si la table des pages est en mémoire principale, chaque adresse logique occasionne au moins 2 références à la mémoire
  - Une pour lire l'entrée de la table de pages
  - L'autre pour lire le mot référencé
- Le temps d'accès mémoire est doublé...

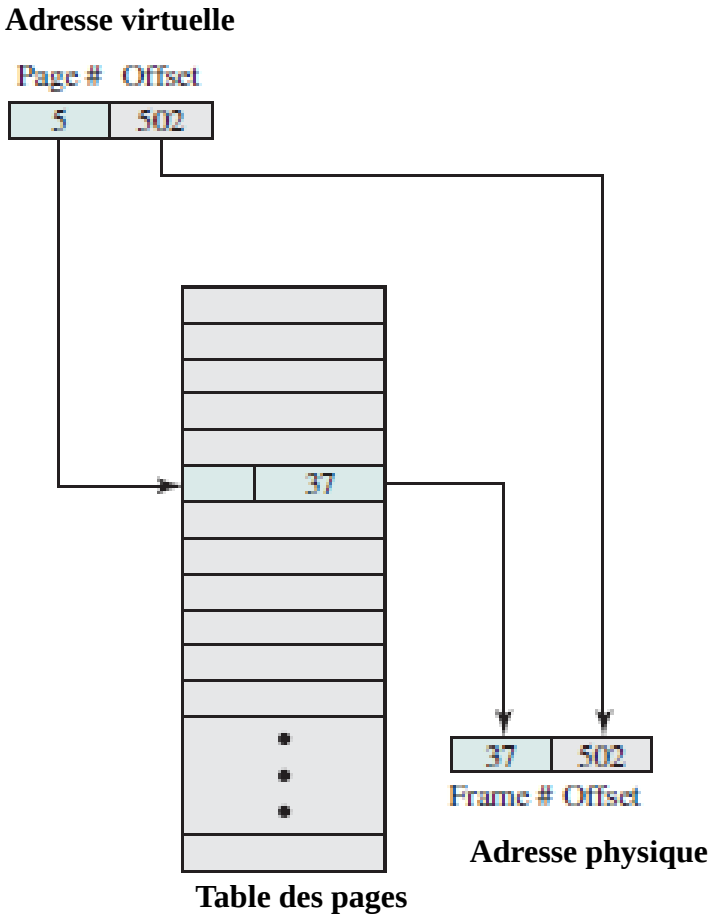
Pour améliorer l’efficacité

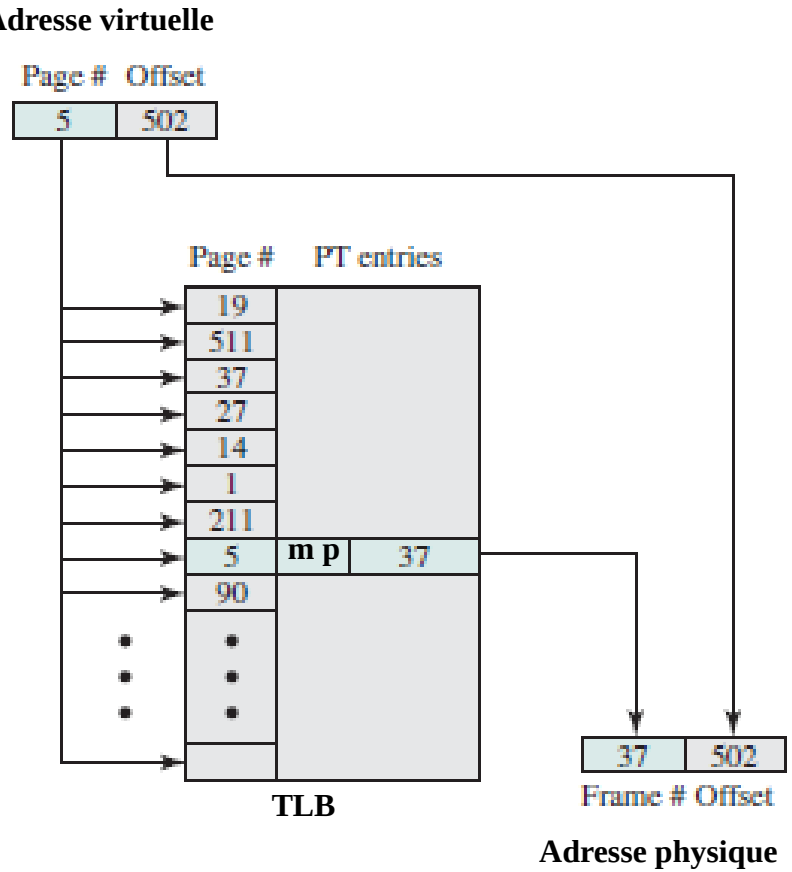
- Où mettre les tables des pages (les mêmes idées s ’appliquent aussi aux tables de segment)
- Solution 1: dans des registres du processeur.
  - avantage: vitesse
  - désavantage: nombre limité de pages par proc., la taille de la mémoire logique est limitée
- Solution 2: en mémoire principale
  - avantage: taille de la mémoire logique illimitée
  - désavantage: mentionné
- Solution 3 (mixte): les tableaux de pages sont en mémoire principale, mais les adresses les plus utilisées sont aussi dans des registres du processeur.

Registres associatifs TLB
Translation Lookaside Buffers, ou *caches* d'adressage

- Recherche parallèle d'une adresse:
  - L'adresse recherchée est cherchée dans la partie gauche de la table en parallèle (matériel spécial)
- Traduction page → cadre
  - Si la page recherchée a été utilisée **récemment** elle se trouvera dans les registres associatifs

TLB versus Accès direct



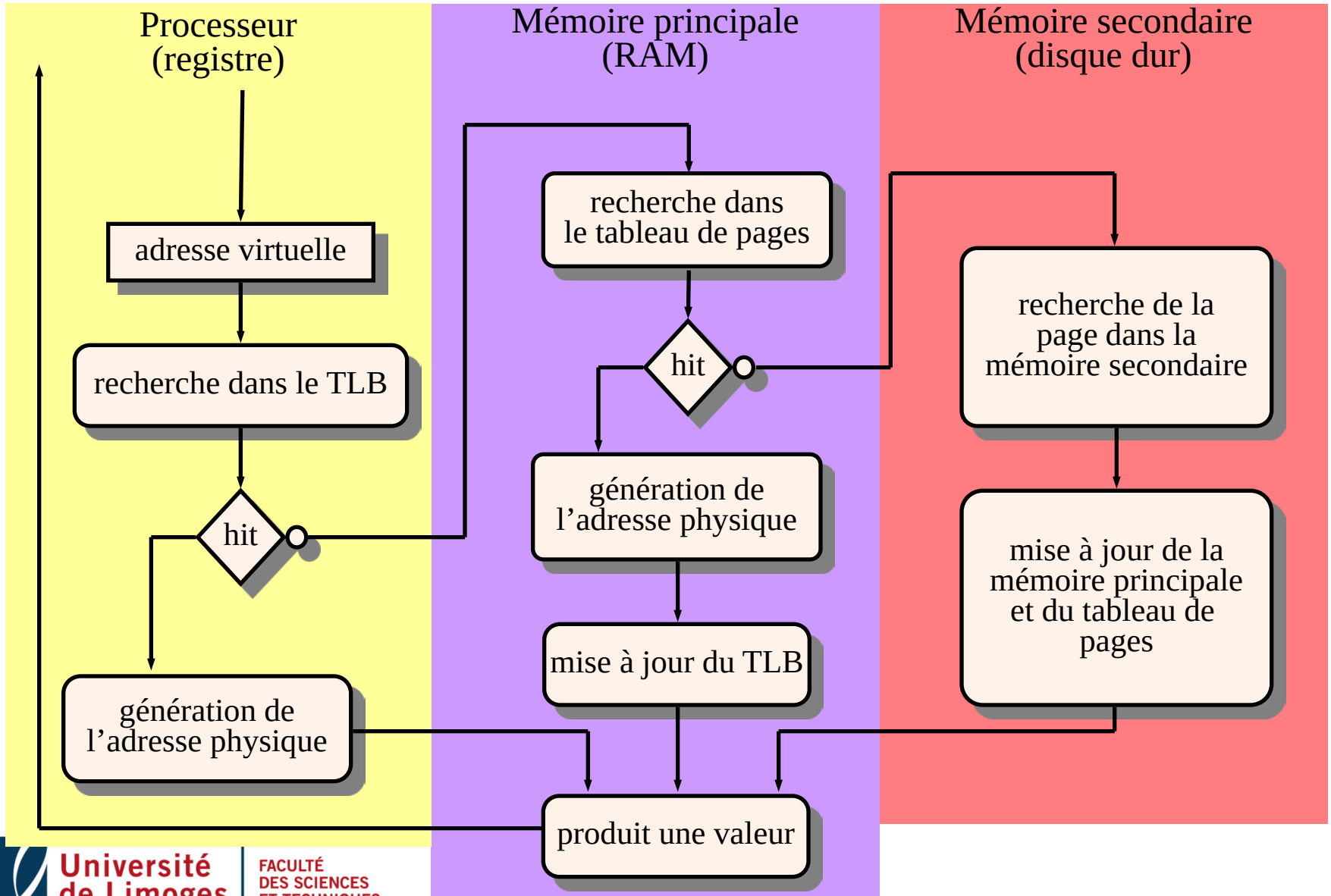


Translation Lookaside Buffer

- Sur réception d'une adresse logique, le processeur examine le cache TLB
- Si cette entrée de page y est, le numéro de cadre en est extrait
- Sinon, le numéro de page indexe la table de page du processus (en mémoire)
  - Cette nouvelle entrée de page est mise dans le TLB
  - Elle remplace une autre pas récemment utilisée
- Le TLB est vidé quand l'UCT change de proc

Les trois premières opérations sont faites par matériel

Pagination - Algorithme



Algorithmes de remplacement de pages

- Une faute de page force le système à choisir:
  - Quelle page doit être enlevée pour faire de la place pour la nouvelle page
- Les pages modifiées doivent d'abord être sauvegardées
  - Sinon on peut simplement les écraser
- Il est préférable de ne pas choisir une page souvent utilisée
  - Sinon il faudra la recharger bientôt

L'algorithme optimal

- Remplacer la page qui sera utilisée après toutes les autres
  - Optimal mais irréalisable
- On exécute deux fois un programme
  - La seconde fois on a l'information nécessaire pour implémenter l'algorithme de remplacement de page optimal
  - Permet de comparer les performances d'algorithmes réalisables avec l'algorithme optimal.

Algorithme de remplacement de la page non récemment utilisée (NRU)

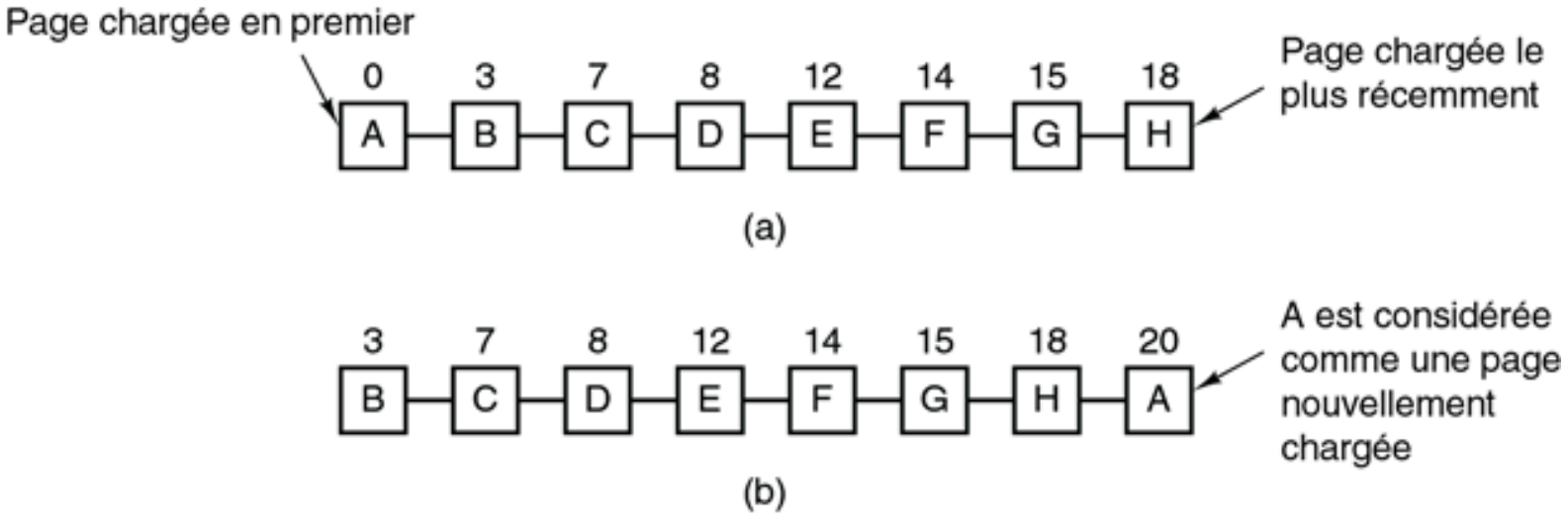
- Chaque page possède deux bits: R et M
  - Ces bits sont mis à 1 lorsque la page est référencée ou modifiée (le bit R est remis à 0 périodiquement).
- 4 classes de pages
  1. non référencée, non modifiée
  2. non référencée, modifiée
  3. référencée, non modifiée
  4. référencée, modifiée
- NRU enlève une page au hasard
  - En partant du plus petit numéro de classe

Algorithme premier arrivé, premier sorti

- On maintient un liste des pages
  - dans l'ordre de leur chargement en mémoire
- La page au début de la liste (la plus ancienne) est remplacée
- Désavantage
  - les vielles pages peuvent aussi être celles qui sont le plus utilisées.

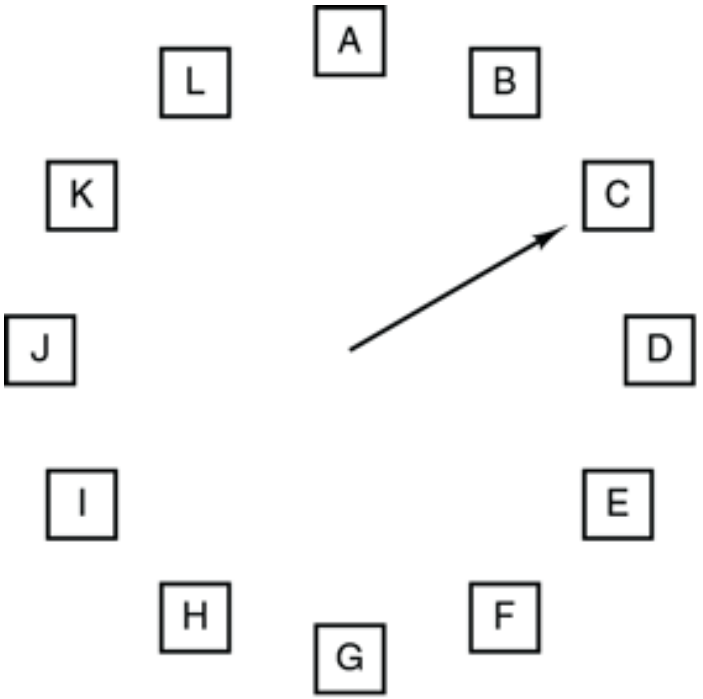
Algorithme de la seconde chance

- Opération *seconde chance:*
  - Pages triées selon l'ordre d'arrivé
  - Si le bit R de la page la plus ancienne est 1 alors on la met à la fin de la liste comme une nouvelle page (son bit R à 0)



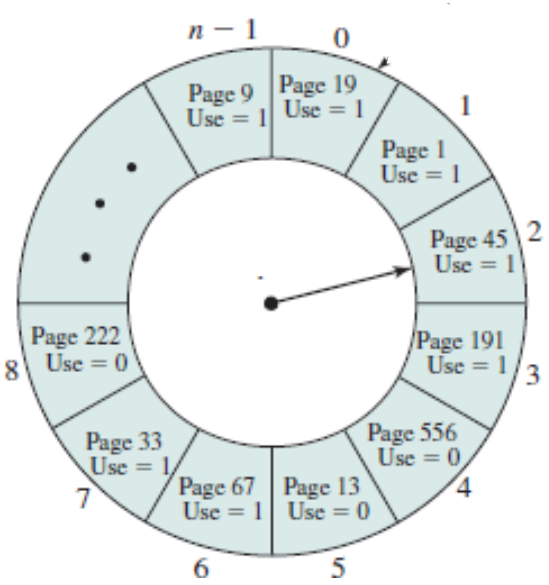
Algorithme de l'horloge

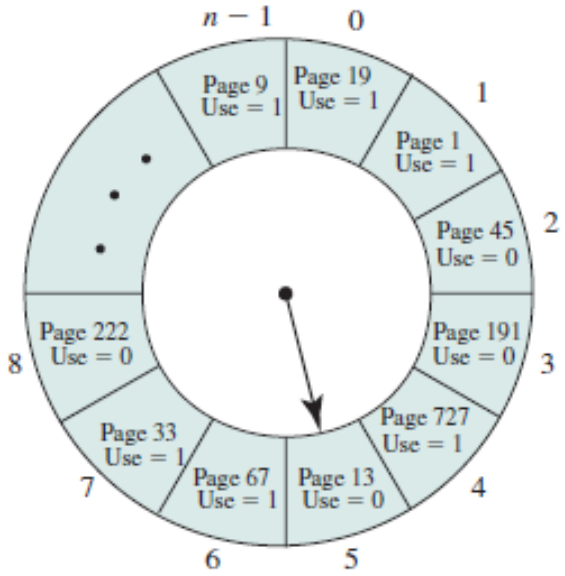
Comme l'algorithme précédent mais l'utilisation d'une liste circulaire évite de déplacer les pages.



Quand un défaut de page se produit, la page pointée est testée. L'action entreprise dépend de la valeur du bit R :
R = 0 : retirer la page
R = 1 : mettre R à 0 et avancer le pointeur

Algorithme de l'horloge





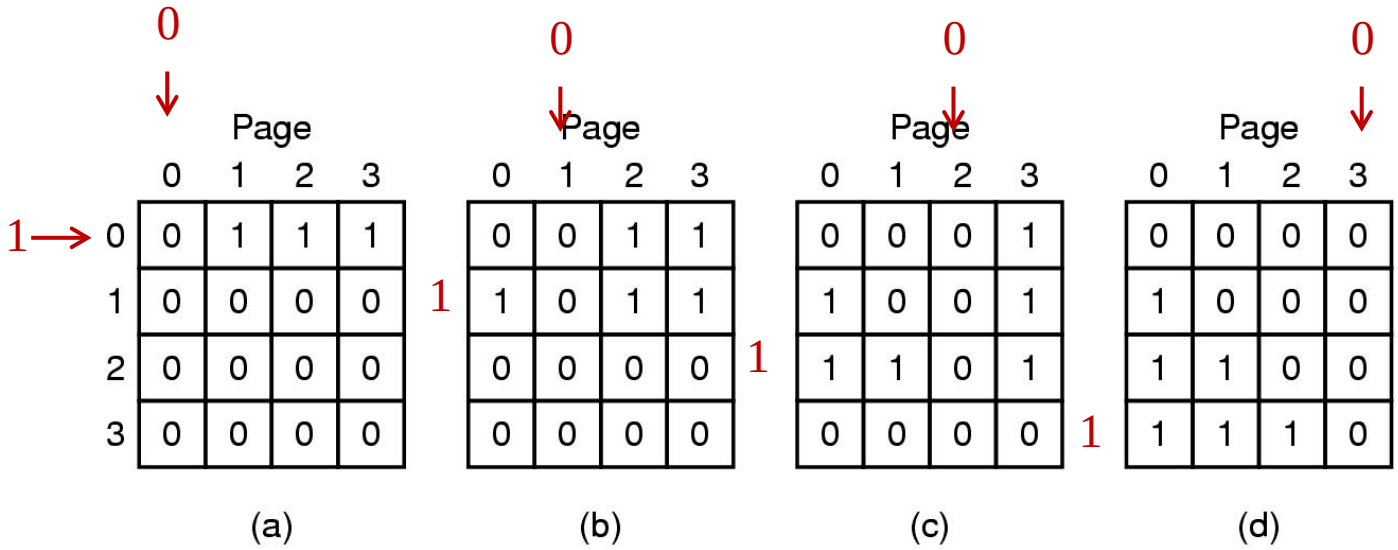
Algorithme de la page la moins récemment utilisée (LRU)

- On suppose que les pages qui ont été utilisées récemment le seront à nouveau prochainement
  - On enlève les pages qui n'ont pas été utilisées depuis longtemps
- Tenir à jour une liste chaînée de pages
  - Page la plus récente en premier
  - Mettre à jour cette liste à chaque référence mémoire !!
- On utilise un compteur C incrémenté après chaque instruction
  - On copie C dans la page des tables pour l'entrée de la page référencées
  - Choisir la page avec la plus petite valeur de compteur
  - On remet le compteur à 0 périodiquement

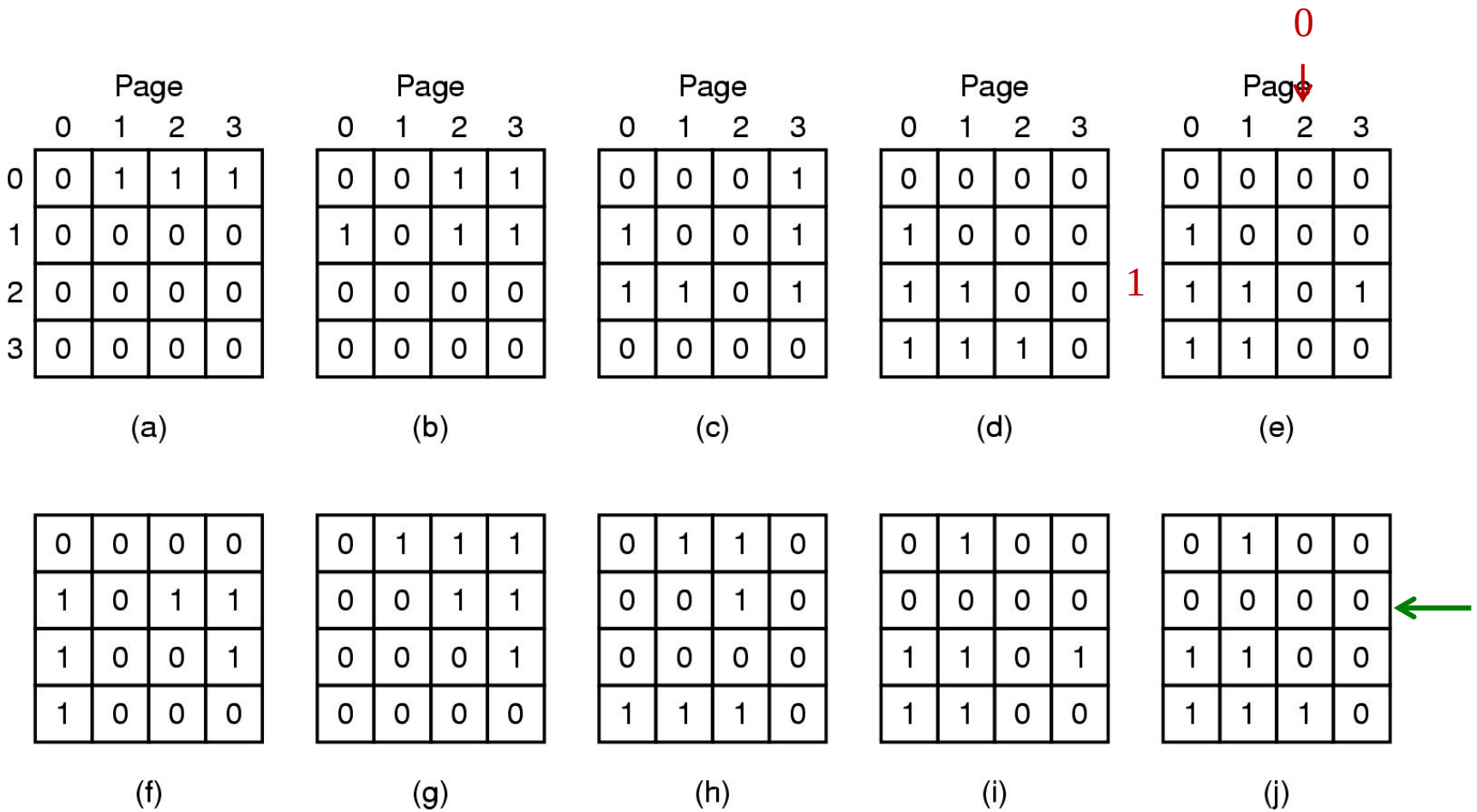
Algorithme de la page la moins récemment utilisée (LRU)

- Autre solution matérielle
  - Pour une machine à n cadre
  - Matrice n×n
  - Quand une page k est référencée
    - mettre à 1 tous les bits de la rangée k
    - mettre à 0 tous les bits de la colonne k

Exemple – référence des pages: 0,1,2,3…2



Exemple – référence des pages:
0,1,2,3,2,1,0,3,2,3



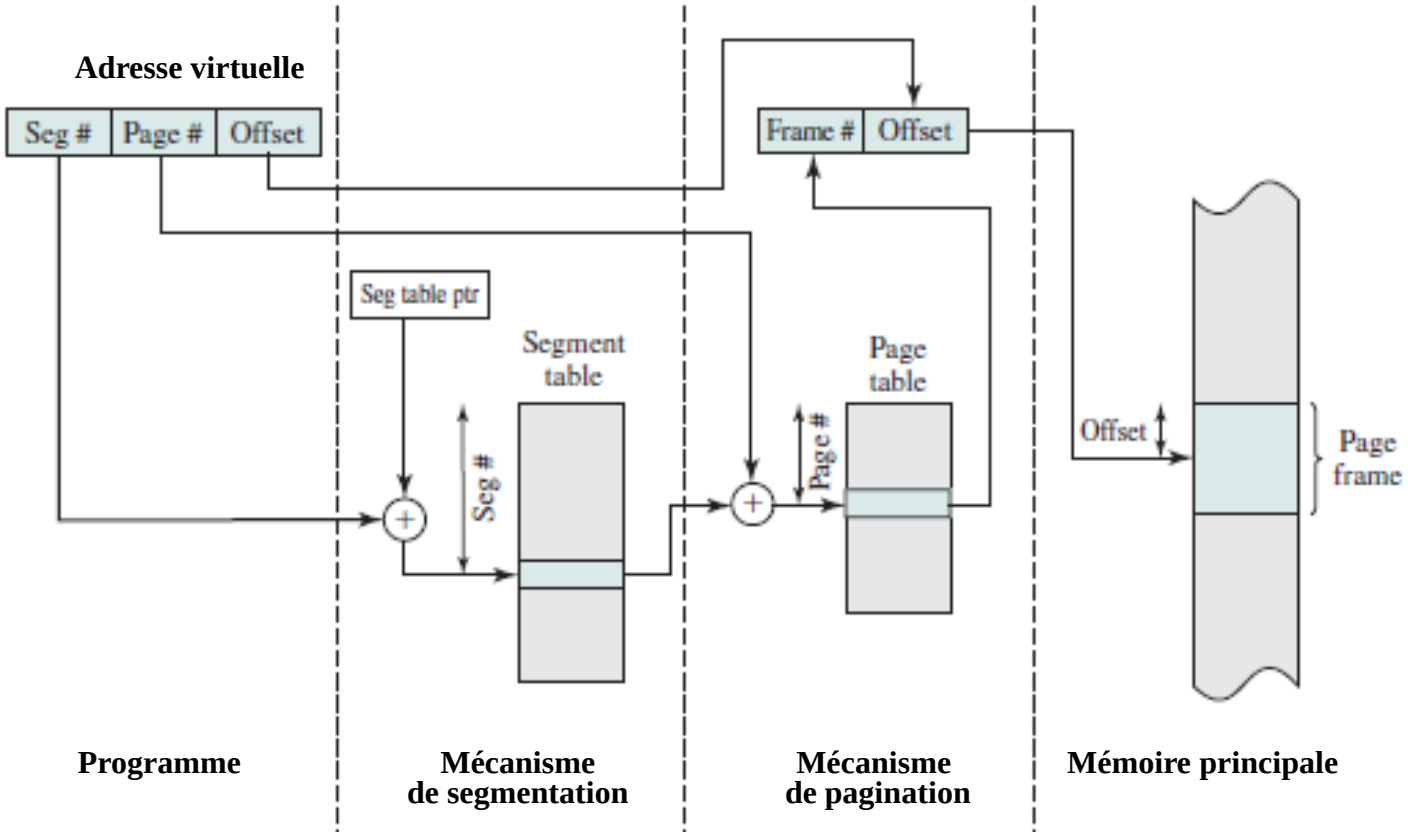
Résumé des algorithmes de remplacement de pages

Algorithme	Commentaires
Optimal	*Utilisé comme banc d'essai*
NRU	*Très grossier*
FIFO	*Peut écarter des pages importantes*
Seconde chance	*Grande amélioration de FIFO*
Horloge	*Réaliste*
LRU	*Excellent mais difficile à implanter*
NFU	*Approximation grossière de LRU*
Vieillissement	*Approximation efficace de LRU*
Ensemble de travail	*Coûteux*
WSClock	*Bonne efficacité*

Pagination et segmentation combinées

- Les programmes sont divisés en segments et les segments sont paginés
- Donc chaque adresse de segment n’est pas une adresse de mémoire, mais une adresse au tableau de pages du segment
- Les tableaux de segments et de pages peuvent être eux-mêmes paginés

Pagination et segmentation



Utilisation de Translation Lookaside Buffer

- Dans le cas de systèmes de pagination à plusieurs niveaux, l'utilisation de TLB devient encore plus importante pour éviter les multiples accès en mémoire afin de calculer une adresse physique
- Les adresses les plus récemment utilisées sont trouvées directement dans la TLB.

Conclusions sur la Gestion Mémoire

- Problèmes de:
  - fragmentation (interne et externe)
  - complexité et efficacité des algorithmes
- Méthodes
  - Allocation contiguë
    - Partitions fixes
    - Partitions variables
    - Groupes de paires
  - Pagination / Segmentation
- Problèmes en pagination et segmentation:
  - taille des tableaux de segments et pages
  - pagination de ces tableaux
  - efficacité fournie par Translation Lookaside Buffer

Récapitulation sur la fragmentation

- Partition fixes: fragmentation interne car les partitions ne peuvent pas être complètement utilisées + fragmentation externe s'il y a des partitions non utilisées.
- Partitions dynamiques: fragmentation externe qui conduit au besoin de compression.
- Segmentation sans pagination: pas de fragmentation interne, mais fragmentation externe à cause de segments de longueur différentes, stockés de façon contiguë (comme dans les partitions dynamiques)

Récapitulation sur la fragmentation

- Pagination:
  - en moyenne, 1/2 cadre de fragmentation interne par processus
  - dans le cas de mémoire virtuelle, aucune fragmentation externe
- Donc la pagination avec mémoire virtuelle offre la meilleure solution au problème de la fragmentation

Comparaison entre la segmentation et la pagination

Considérations	Pagination	Segmentation
Le programmeur doit-il connaître la technique utilisée?	Non	Oui
Combien y a-t-il d'espaces d'adressage linéaires?	1	Plusieurs
L'espace total d'adressage peut-il dépasser la taille de la mémoire physique?	Oui	Oui
Les procédures et les données peuvent-elles être séparées et protégées séparément?	Non	Oui
Peut-on traiter facilement des tables dont les tailles varient?	Non	Oui
Le partage de procédures entre utilisateurs est-il simplifié?	Non	Oui

Any question ?