Deuxième partie

Gestion de la mémoire

- Introduction et concepts
- Mémoire uniforme
  - Partition unique / Partitions multiples
  - Pagination
  - Segmentation
  - Segmentation paginée
- Mémoire hiérarchisée (virtuelle)
  - Swapping
  - Pagination à la demande
  - Algorithmes de remplacement de pages
- Etude de cas
  - Le Pentium
  - Linux

Introduction

- Problème d'allocation de ressources : la mémoire est une ressource importante finie d'un système informatique, en particulier dans un système multiprogrammé

Introduction

- Problème d'allocation de ressources : la mémoire est une ressource importante finie d'un système informatique, en particulier dans un système multiprogrammé

Fonctions d'un gestionnaire de la mémoire

- garder trace de l'état de chaque portion de la mémoire (libre ou allouée)
- mettre en place d'une politique d'allocation/libération
  - choisir entre plusieurs processus
  - espace mémoire à allouer à un processus donné
- assurer la protection inter et intra-processus

Introduction

- Problème d'allocation de ressources : la mémoire est une ressource importante finie d'un système informatique, en particulier dans un système multiprogrammé

Fonctions d'un gestionnaire de la mémoire

- garder trace de l'état de chaque portion de la mémoire (libre ou allouée)
- mettre en place d'une politique d'allocation/libération
  - choisir entre plusieurs processus
  - espace mémoire à allouer à un processus donné
- assurer la protection inter et intra-processus

Objectifs d'un gestionnaire de la mémoire

- optimiser l'utilisation de la mémoire principale
- augmenter le rendement global du système
  - par exemple, avoir suffisamment de processus en mémoire pour assurer une bonne répartition entre les cycles E/S et CPU

Rappel. Hiérarchie mémoire

- l'UC a un accès direct à ses registres et à la mémoire principale :

Rappel. Hiérarchie mémoire

- l'UC a un accès direct à ses registres et à la mémoire principale :
  - un cycle d'horloge (ou moins) pour accéder à un registre

Rappel. Hiérarchie mémoire

- l'UC a un accès direct à ses registres et à la mémoire principale :
  - un cycle d'horloge (ou moins) pour accéder à un registre
  - souvent, plusieurs cycles d'horloge sont nécessaires (suivant le type d'adressage) pour accéder à la mémoire principale

Rappel. Hiérarchie mémoire

- l'UC a un accès direct à ses registres et à la mémoire principale :
  - un cycle d'horloge (ou moins) pour accéder à un registre
  - souvent, plusieurs cycles d'horloge sont nécessaires (suivant le type d'adressage) pour accéder à la mémoire principale
- mémoire plus rapide = mémoire plus petite parce que plus coûteuse

Rappel. Hiérarchie mémoire

- l'UC a un accès direct à ses registres et à la mémoire principale :
  - un cycle d'horloge (ou moins) pour accéder à un registre
  - souvent, plusieurs cycles d'horloge sont nécessaires (suivant le type d'adressage) pour accéder à la mémoire principale
- mémoire plus rapide = mémoire plus petite parce que plus coûteuse
- le cache se situe entre les registres et la mémoire principale

Rappel. Hiérarchie mémoire

- l'UC a un accès direct à ses registres et à la mémoire principale :
  - un cycle d'horloge (ou moins) pour accéder à un registre
  - souvent, plusieurs cycles d'horloge sont nécessaires (suivant le type d'adressage) pour accéder à la mémoire principale
- mémoire plus rapide = mémoire plus petite parce que plus coûteuse
- le cache se situe entre les registres et la mémoire principale
- l'accès au disque dur (mémoire secondaire) est plus lent par rapport à la mémoire centrale

Rappel. Hiérarchie mémoire

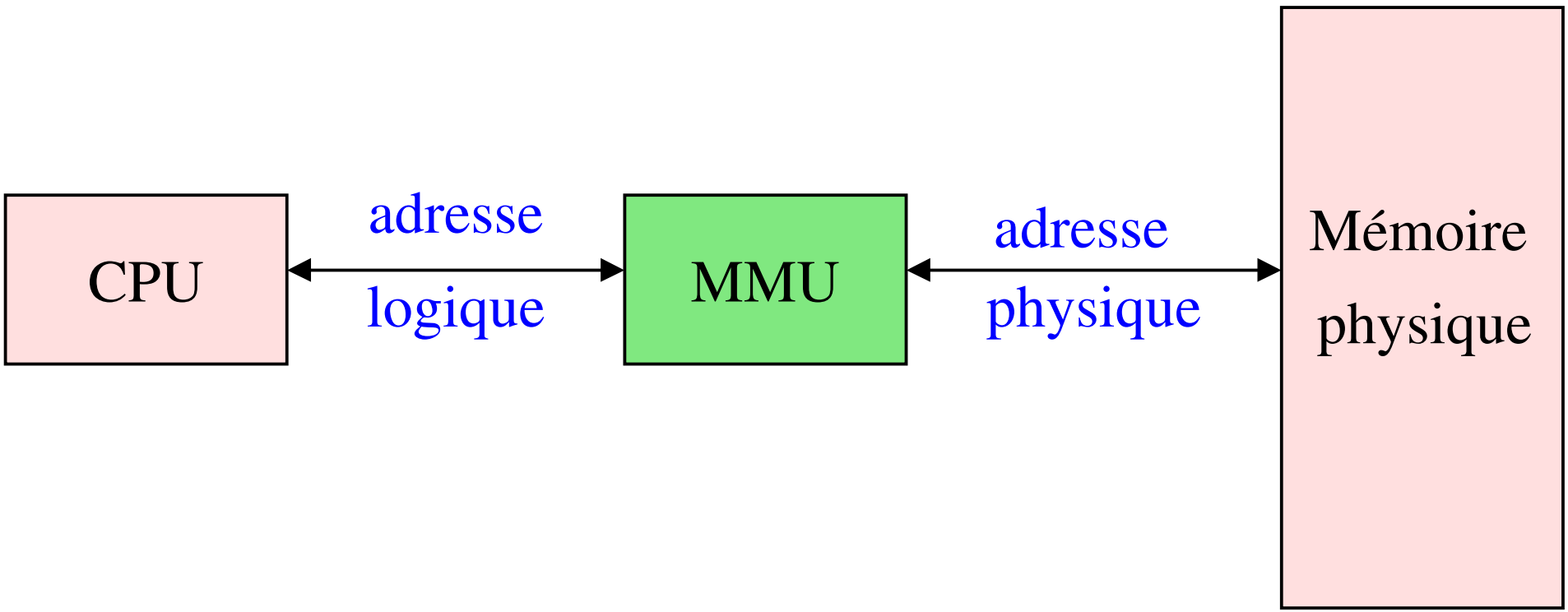
- l'UC a un accès direct à ses registres et à la mémoire principale :
  - un cycle d'horloge (ou moins) pour accéder à un registre
  - souvent, plusieurs cycles d'horloge sont nécessaires (suivant le type d'adressage) pour accéder à la mémoire principale
- mémoire plus rapide = mémoire plus petite parce que plus coûteuse
- le cache se situe entre les registres et la mémoire principale
- l'accès au disque dur (mémoire secondaire) est plus lent par rapport à la mémoire centrale
- le matériel et le système d'exploitation sont responsables du déplacement des objets le long de cette hiérarchie :

Rappel. Hiérarchie mémoire

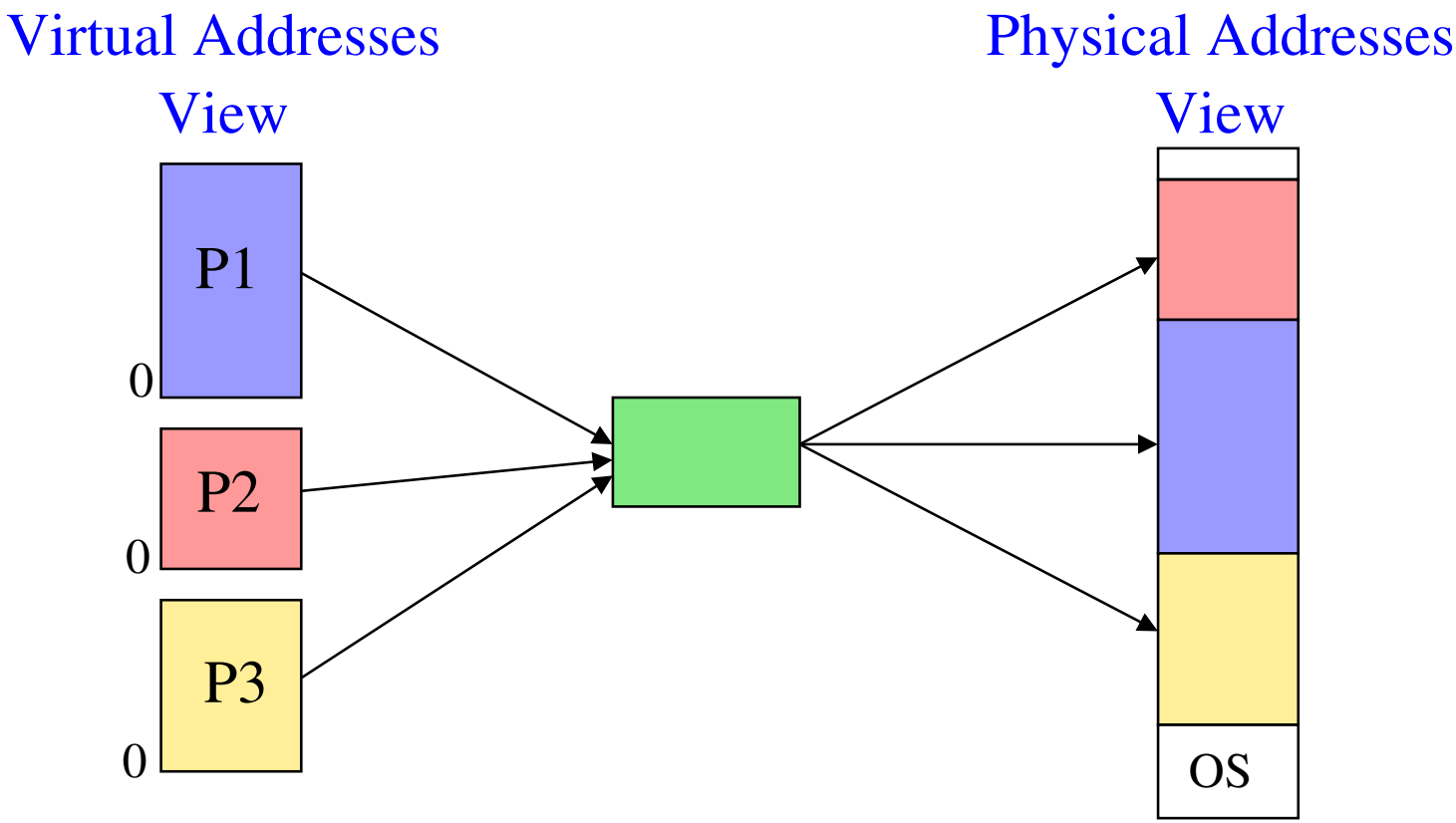
- l'UC a un accès direct à ses registres et à la mémoire principale :
  - un cycle d'horloge (ou moins) pour accéder à un registre
  - souvent, plusieurs cycles d'horloge sont nécessaires (suivant le type d'adressage) pour accéder à la mémoire principale
- mémoire plus rapide = mémoire plus petite parce que plus coûteuse
- le cache se situe entre les registres et la mémoire principale
- l'accès au disque dur (mémoire secondaire) est plus lent par rapport à la mémoire centrale
- le matériel et le système d'exploitation sont responsables du déplacement des objets le long de cette hiérarchie :
  - exemple : de la mémoire centrale au disque et inversement.

Adresses physiques vs. logiques

- adresse physique (réelle) correpond à la vision de la mémoire centrale → espace d'adressage physique ou réel
- adresse logique (virtuelle) est l'adresse générée par l'unité centrale → espace d'adressage logique ou virtuel
- c'est lors de la liaison d'adresses (compilation, chargement ou exécution) que les adresses logiques sont traduites en adresses physiques

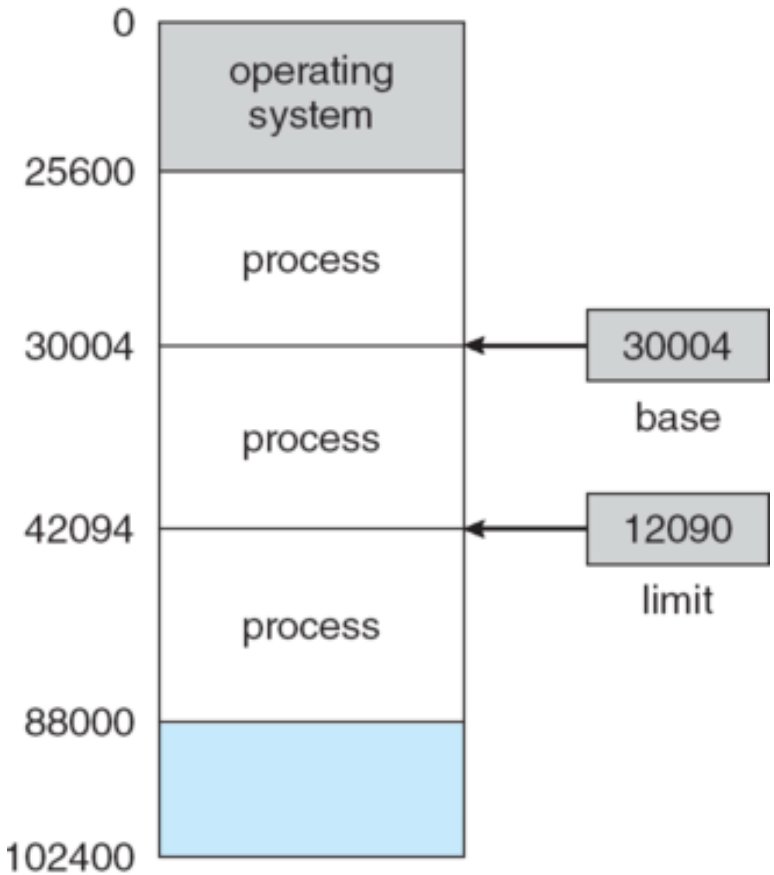


Adresses physiques vs. logiques (cont.)



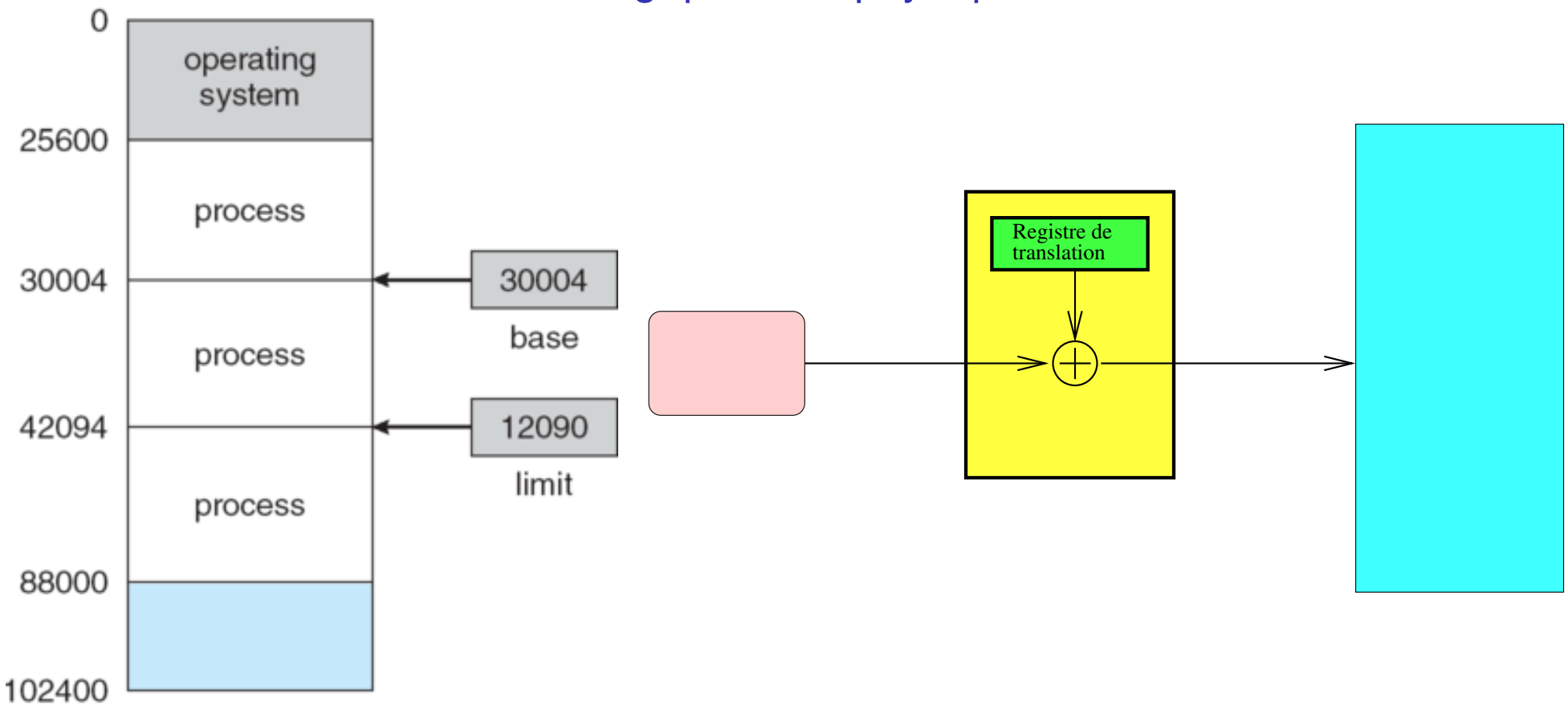
Memory Management Unit (MMU)

La MMU est un dispositif matériel qui permet la conversion :
@logique $\rightarrow$ @physique



Memory Management Unit (MMU)

La MMU est un dispositif matériel qui permet la conversion :
@logique → @physique



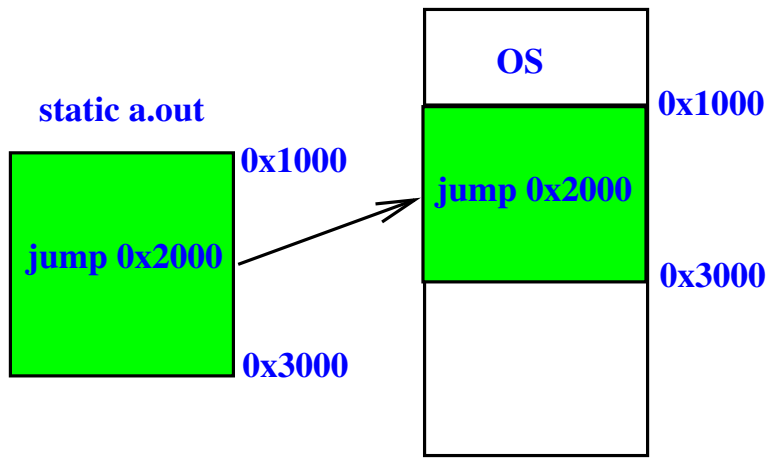
Moment de la traduction d'adresses

UC $\rightarrow$ @logique $\rightarrow$ @physique $\rightarrow$ MC

Liaison statique

Traduction lors de la compilation ou l'édition des liens

- Résultat : programme absolu



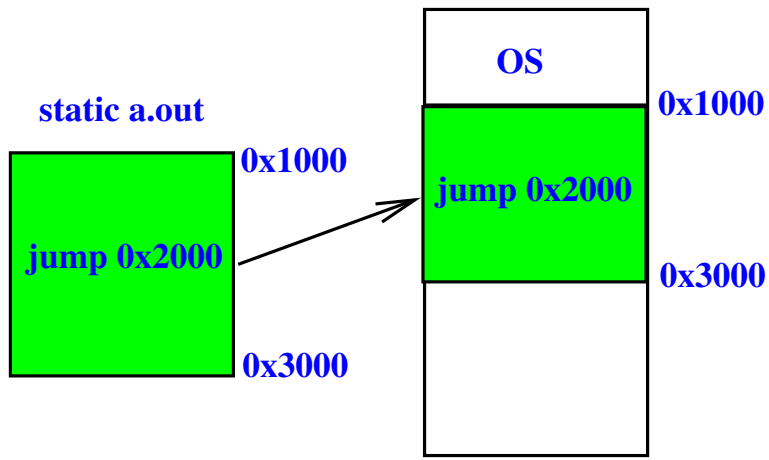
Moment de la traduction d'adresses

UC → @logique → @physique → MC

Liaison statique

Traduction lors de la compilation ou l'édition des liens

- Résultat : programme absolu
- Le chargement d'un tel programme entraine une demande formulée en termes de taille et d'adresse ce qui laisse peu d'initiatives à l'allocateur



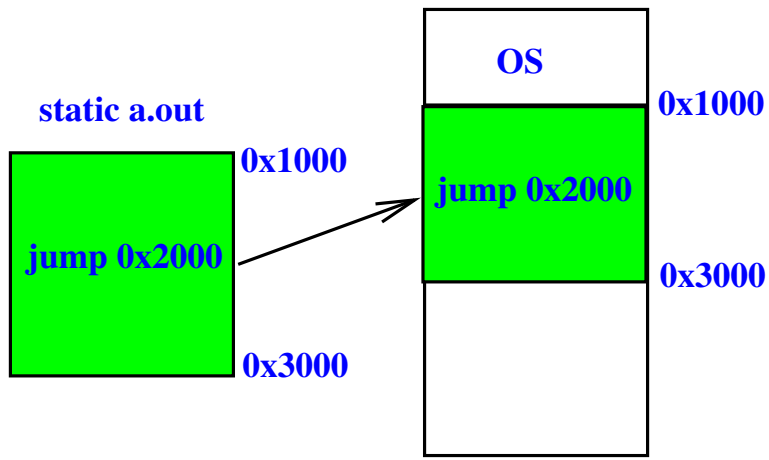
Moment de la traduction d'adresses

UC → @logique → @physique → MC

Liaison statique

Traduction lors de la compilation ou l'édition des liens

- Résultat : programme absolu
- Le chargement d'un tel programme entraine une demande formulée en termes de taille et d'adresse ce qui laisse peu d'initiatives à l'allocateur
- Si l'adresse de chargement change, il est nécessaire de recompiler



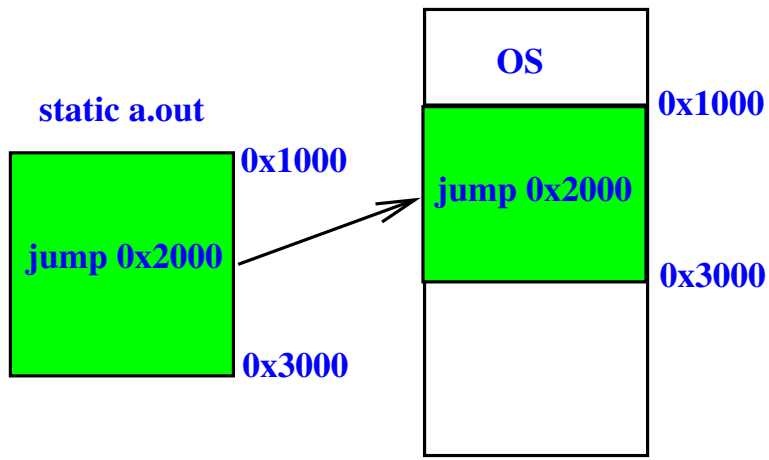
Moment de la traduction d'adresses

UC → @logique → @physique → MC

Liaison statique

Traduction lors de la compilation ou l'édition des liens

- Résultat : programme absolu
- Le chargement d'un tel programme entraine une demande formulée en termes de taille et d'adresse ce qui laisse peu d'initiatives à l'allocateur
- Si l'adresse de chargement change, il est nécessaire de recompiler
- Cas des programmes .COM de MS-DOS

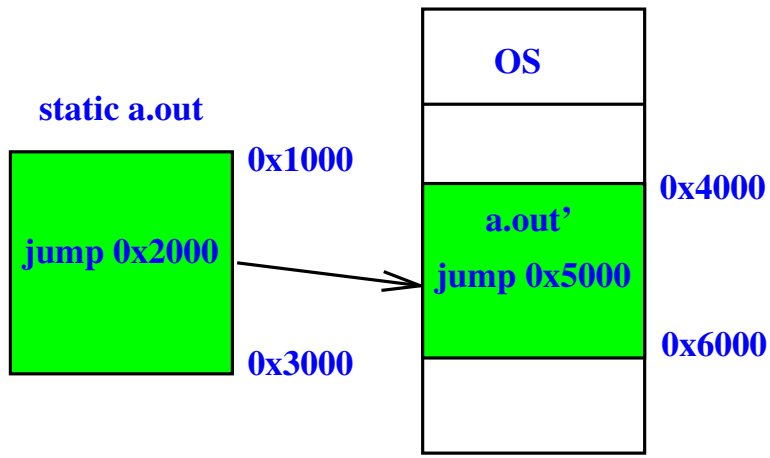


Moment de la traduction d'adresses

UC → @logique → @physique → MC

Liaison au chargement

- résultat : programme translatable

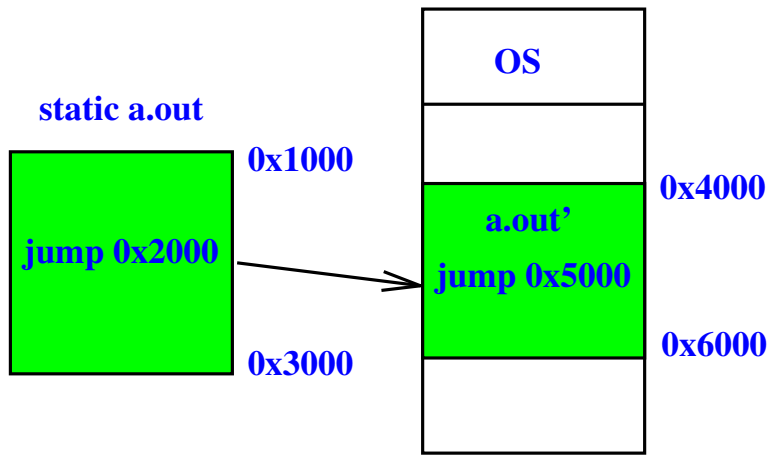


Moment de la traduction d'adresses

UC → @logique → @physique → MC

Liaison au chargement

- résultat : programme translatable
- un chargeur est capable de l'implanter n'importe où en mémoire (réimplantation statique)

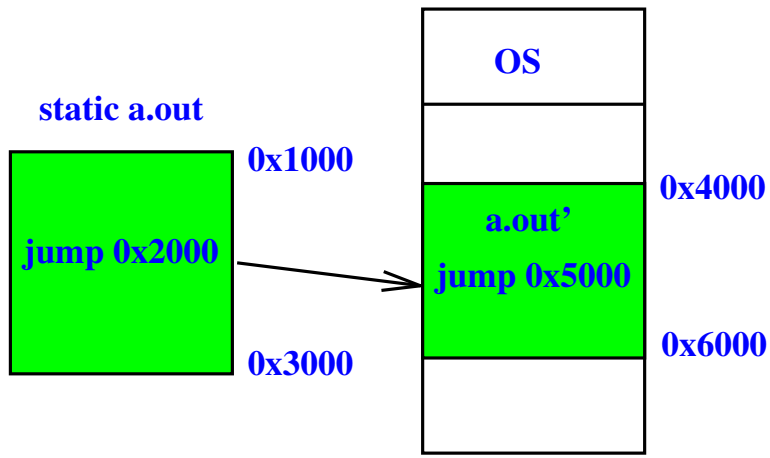


Moment de la traduction d'adresses

UC → @logique → @physique → MC

Liaison au chargement

- résultat : programme translatable
- un chargeur est capable de l'implanter n'importe où en mémoire (réimplantation statique)
- l'adresse du chargement est laissée à l'initiative de l'allocateur

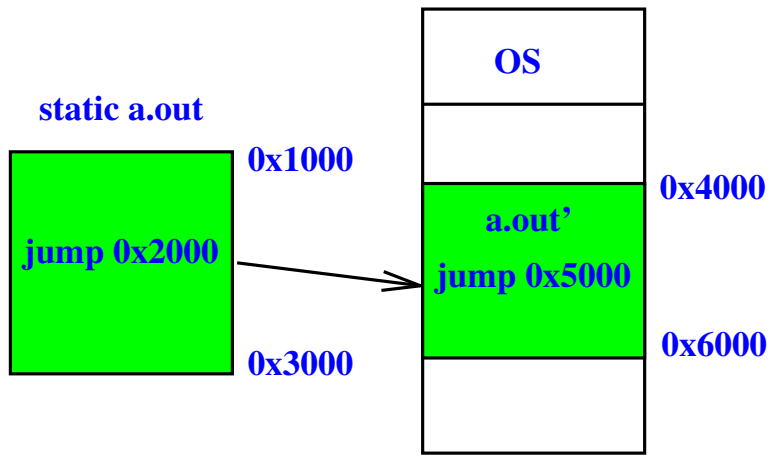


Moment de la traduction d'adresses

UC → @logique → @physique → MC

Liaison au chargement

- résultat : programme translatable
- un chargeur est capable de l'implanter n'importe où en mémoire (réimplantation statique)
- l'adresse du chargement est laissée à l'initiative de l'allocateur
- gestion plus efficace

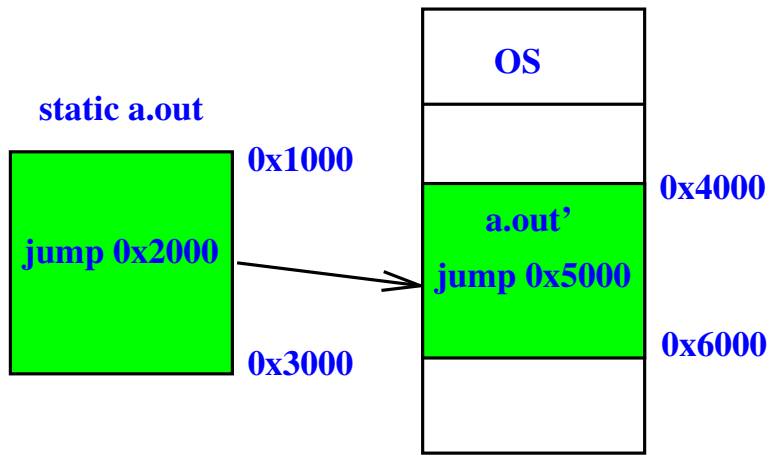


Moment de la traduction d'adresses

UC → @logique → @physique → MC

Liaison au chargement

- résultat : programme translatable
- un chargeur est capable de l'implanter n'importe où en mémoire (réimplantation statique)
- l'adresse du chargement est laissée à l'initiative de l'allocateur
- gestion plus efficace



- limitation : un programme ayant commencé son exécution ne peut plus être déplacé. S'il est swappé sur le disque, il faut qu'il retrouve son emplacement initial (il n'est pas forcément relogeable)

Moment de la traduction d'adresses

UC $\rightarrow$ @logique $\rightarrow$ @physique $\rightarrow$ MC

Liaison au moment l'exécution

- traduction dynamique des adresses
- permet la réimplantation dynamique d'un programme au cours de son exécution
- le module résultant est translatable et relogeable
- Implantations possibles (support matériel) :
  - utilisation d'un registre de base (maintien de la contiguité physique du programme)
  - la pagination (sera vue plus tard)

Mémoire uniforme vs. mémoire hiérarchisée

Mémoire uniforme

- la mémoire centrale (MC), seule mémoire disponible est organisée linéairement et gérée comme un vecteur d'emplacements contigus

Limitations :

- cas d'un programme dont la taille est supérieure à celle de la mémoire
- existence de processus inactifs occupant inutilement une partie de la mémoire centrale

Mémoire uniforme vs. mémoire hiérarchisée

Mémoire uniforme

- la mémoire centrale (MC), seule mémoire disponible est organisée linéairement et gérée comme un vecteur d'emplacements contigus
- à chaque processus, est affectée une zone mémoire dont la taille est le maximum estimé de ses besoins

Limitations :

- cas d'un programme dont la taille est supérieure à celle de la mémoire
- existence de processus inactifs occupant inutilement une partie de la mémoire centrale

Mémoire uniforme vs. mémoire hiérarchisée

Mémoire uniforme

- la mémoire centrale (MC), seule mémoire disponible est organisée linéairement et gérée comme un vecteur d'emplacements contigus
- à chaque processus, est affectée une zone mémoire dont la taille est le maximum estimé de ses besoins
- tout le processus est en mémoire

Limitations :

- cas d'un programme dont la taille est supérieure à celle de la mémoire
- existence de processus inactifs occupant inutilement une partie de la mémoire centrale

Mémoire uniforme vs. mémoire hiérarchisée

Mémoire uniforme

- la mémoire centrale (MC), seule mémoire disponible est organisée linéairement et gérée comme un vecteur d'emplacements contigus
- à chaque processus, est affectée une zone mémoire dont la taille est le maximum estimé de ses besoins
- tout le processus est en mémoire

Limitations :

- cas d'un programme dont la taille est supérieure à celle de la mémoire
- existence de processus inactifs occupant inutilement une partie de la mémoire centrale

Mémoire uniforme vs. mémoire hiérarchisée

Mémoire uniforme

- la mémoire centrale (MC), seule mémoire disponible est organisée linéairement et gérée comme un vecteur d'emplacements contigus
- à chaque processus, est affectée une zone mémoire dont la taille est le maximum estimé de ses besoins
- tout le processus est en mémoire

Limitations :

- cas d'un programme dont la taille est supérieure à celle de la mémoire
- existence de processus inactifs occupant inutilement une partie de la mémoire centrale

Mémoire uniforme vs. mémoire hiérarchisée

Mémoire uniforme

- la mémoire centrale (MC), seule mémoire disponible est organisée linéairement et gérée comme un vecteur d'emplacements contigus
- à chaque processus, est affectée une zone mémoire dont la taille est le maximum estimé de ses besoins
- tout le processus est en mémoire

Limitations :

- cas d'un programme dont la taille est supérieure à celle de la mémoire
- existence de processus inactifs occupant inutilement une partie de la mémoire centrale

Mémoire uniforme vs. mémoire hiérarchisée

Mémoire uniforme

- la mémoire centrale (MC), seule mémoire disponible est organisée linéairement et gérée comme un vecteur d'emplacements contigus
- à chaque processus, est affectée une zone mémoire dont la taille est le maximum estimé de ses besoins
- tout le processus est en mémoire

Limitations :

- cas d'un programme dont la taille est supérieure à celle de la mémoire
- existence de processus inactifs occupant inutilement une partie de la mémoire centrale

Mémoire hiérarchisée (virtuelle)

- la mémoire est constituée d'un ensemble structuré de mémoires de taille et de temps d'accès différents, de telle sorte que les références se fassent toujours à la mémoire la plus rapide
- le cas le plus simple correspond à 2 niveaux : la mémoire centrale et une mémoire secondaire (généralement le disque)

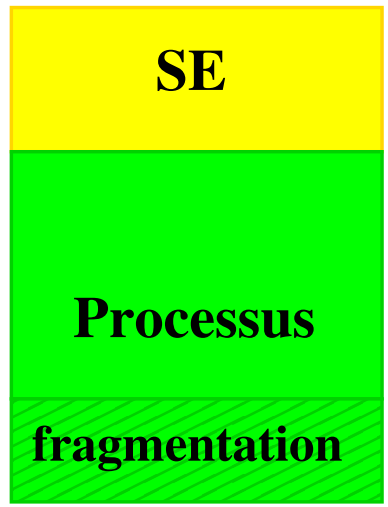
Deuxième partie

Gestion de la mémoire

- Introduction et concepts
- Mémoire uniforme
  - Partition unique / Partitions multiples
  - Pagination
  - Segmentation
  - Segmentation paginée
- Mémoire hiérarchisée (virtuelle)
  - Swapping
  - Pagination à la demande
  - Algorithmes de remplacement de pages
- Etude de cas
  - Le Pentium
  - Linux

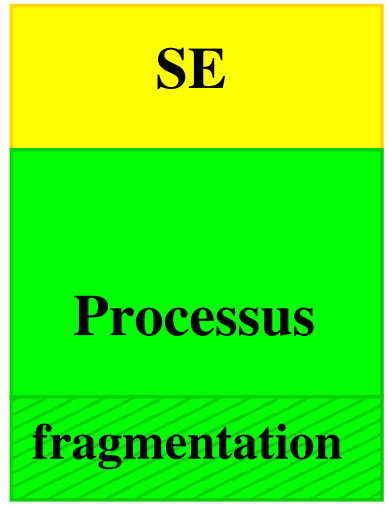
Mémoire uniforme - Partition unique

- Un seul processus à la fois en mémoire



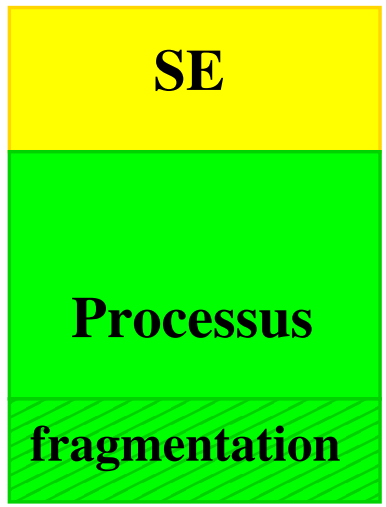
Mémoire uniforme - Partition unique

- Un seul processus à la fois en mémoire
- Fragmentation interne : espace allouée à un processus mais pas utilisé



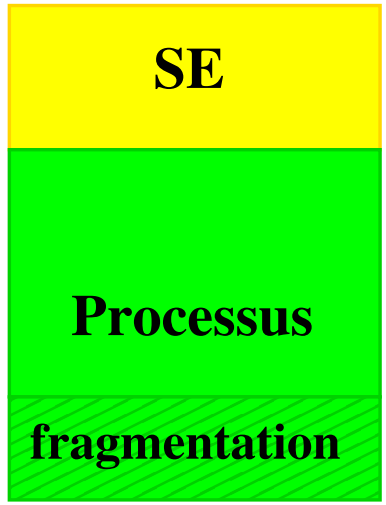
Mémoire uniforme - Partition unique

- Un seul processus à la fois en mémoire
- Fragmentation interne : espace allouée à un processus mais pas utilisé
- Pour des raisons de protection (SE, processus), on associe au processus un registre de base (de translation) et un registre limite



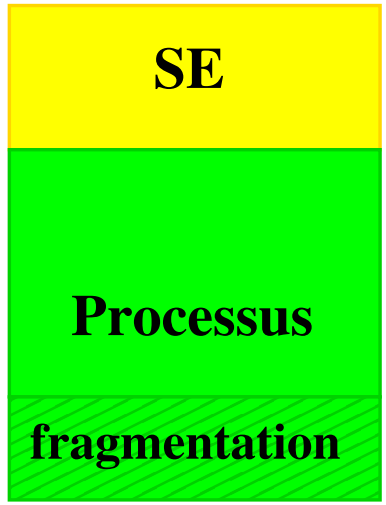
Mémoire uniforme - Partition unique

- Un seul processus à la fois en mémoire
- Fragmentation interne : espace allouée à un processus mais pas utilisé
- Pour des raisons de protection (SE, processus), on associe au processus un registre de base (de translation) et un registre limite
- C'est le cas de MS-DOS et les systèmes embarqués pour PDA par exemple



Mémoire uniforme - Partition unique

- Un seul processus à la fois en mémoire
- Fragmentation interne : espace allouée à un processus mais pas utilisé
- Pour des raisons de protection (SE, processus), on associe au processus un registre de base (de translation) et un registre limite
- C'est le cas de MS-DOS et les systèmes embarqués pour PDA par exemple

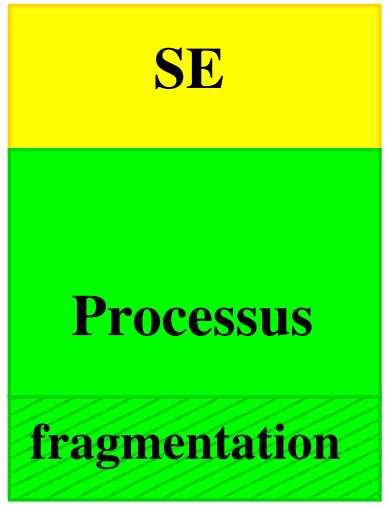


Avantages et inconvénients

- tout le processus est en mémoire

Mémoire uniforme - Partition unique

- Un seul processus à la fois en mémoire
- Fragmentation interne : espace allouée à un processus mais pas utilisé
- Pour des raisons de protection (SE, processus), on associe au processus un registre de base (de translation) et un registre limite
- C'est le cas de MS-DOS et les systèmes embarqués pour PDA par exemple

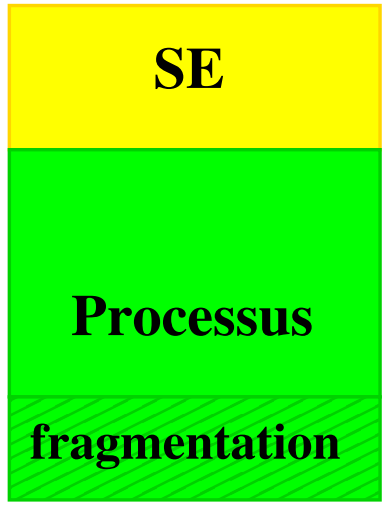


Avantages et inconvénients

- tout le processus est en mémoire
- facile à implanter

Mémoire uniforme - Partition unique

- Un seul processus à la fois en mémoire
- Fragmentation interne : espace allouée à un processus mais pas utilisé
- Pour des raisons de protection (SE, processus), on associe au processus un registre de base (de translation) et un registre limite
- C'est le cas de MS-DOS et les systèmes embarqués pour PDA par exemple



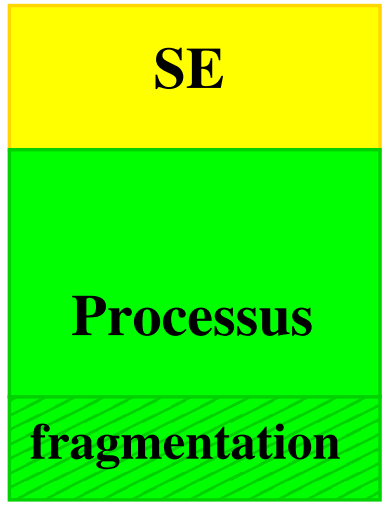
Avantages et inconvénients

- tout le processus est en mémoire
- facile à implanter

- fragmentation interne

Mémoire uniforme - Partition unique

- Un seul processus à la fois en mémoire
- Fragmentation interne : espace allouée à un processus mais pas utilisé
- Pour des raisons de protection (SE, processus), on associe au processus un registre de base (de translation) et un registre limite
- C'est le cas de MS-DOS et les systèmes embarqués pour PDA par exemple



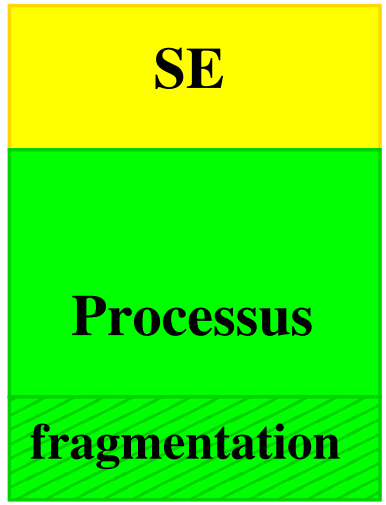
Avantages et inconvénients

- tout le processus est en mémoire
- facile à implanter

- fragmentation interne
- limite le degré de multiprogrammation

Mémoire uniforme - Partition unique

- Un seul processus à la fois en mémoire
- Fragmentation interne : espace allouée à un processus mais pas utilisé
- Pour des raisons de protection (SE, processus), on associe au processus un registre de base (de translation) et un registre limite
- C'est le cas de MS-DOS et les systèmes embarqués pour PDA par exemple



Avantages et inconvénients

- tout le processus est en mémoire
- facile à implanter

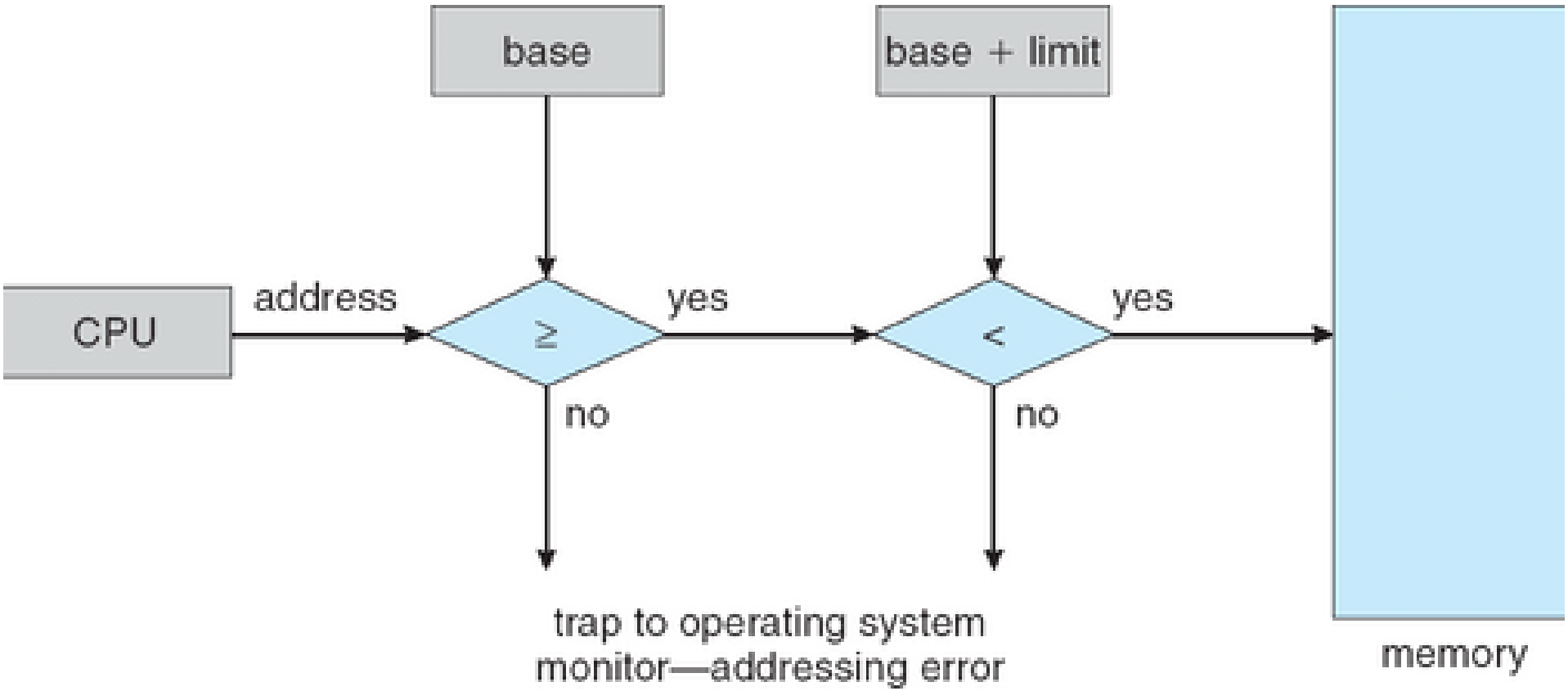
- fragmentation interne
- limite le degré de multiprogrammation
- mauvaise exploitation de l'UC et la MC

Mémoire uniforme - Plusieurs partitions

- une partition peut contenir un seul processus

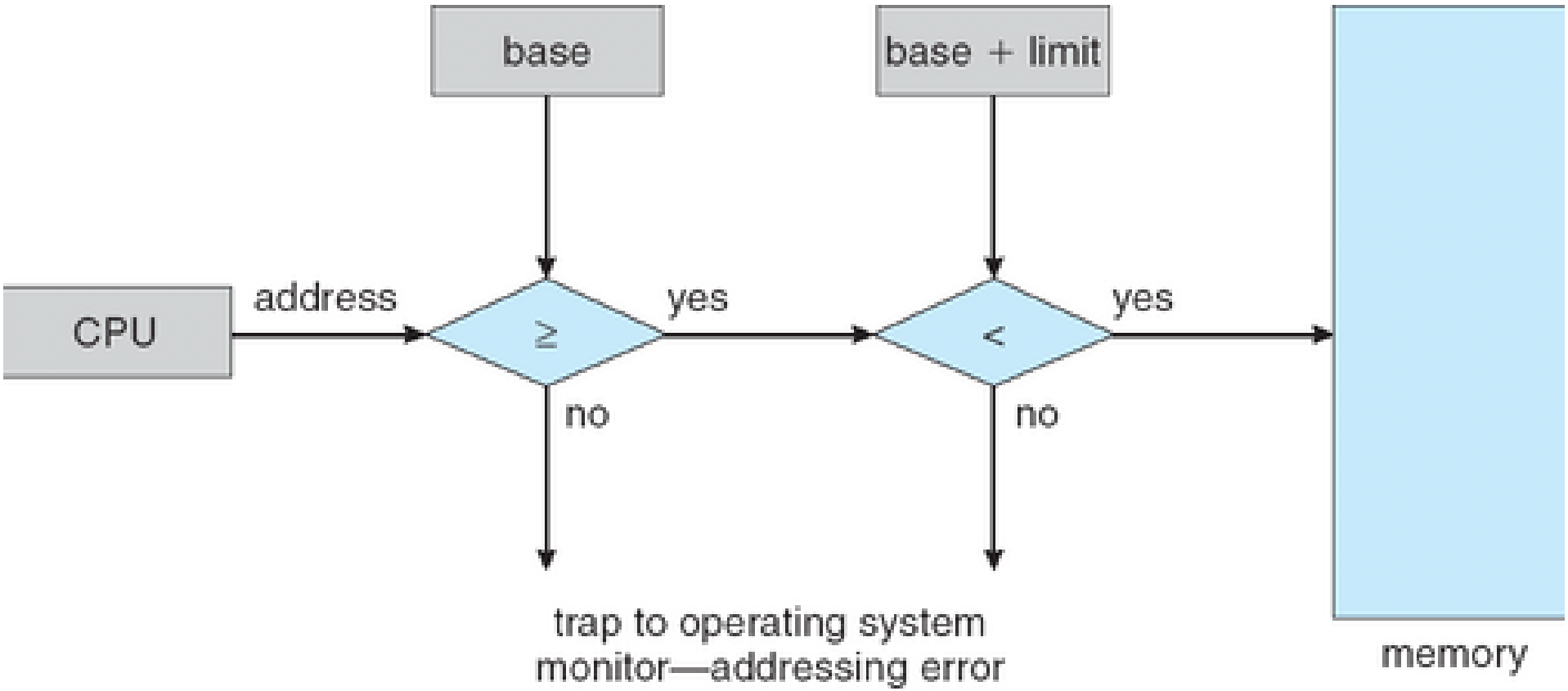
Mémoire uniforme - Plusieurs partitions

- une partition peut contenir un seul processus
- allocation contiguë avec un mécanisme matériel de protection et de conversion d'adresses



Mémoire uniforme - Plusieurs partitions

- une partition peut contenir un seul processus
- allocation contiguë avec un mécanisme matériel de protection et de conversion d'adresses
- les partitions peuvent être fixes ou dynamiques



Mémoire uniforme - Partitions fixes

La taille des partitions est indépendantes des processus chargés

Implantation

Charger le processus p dans une partition P_i libre si $T(p) \leq T(P_i)$
T : taille

Mémoire uniforme - Partitions fixes

La taille des partitions est indépendantes des processus chargés

Implantation

`Charger le processus` p `dans une partition` P_i `libre si` $T(p) \leq T(P_i)$
T `: taille`

- Avantage : facile à implanter

Mémoire uniforme - Partitions fixes

La taille des partitions est indépendantes des processus chargés

Implantation

`Charger le processus` p `dans une partition` P_i `libre si` $T(p) \leq T(P_i)$
T `: taille`

- Avantage : facile à implanter
- Inconvénients :
  - degré de multiprogrammation limité par le nombre des partitions
  - cas des processus dont la taille est supérieure ou inférieure à la taille des partitions ?
  - fragmentation interne

Mémoire uniforme - Partitions fixes

La taille des partitions est indépendantes des processus chargés

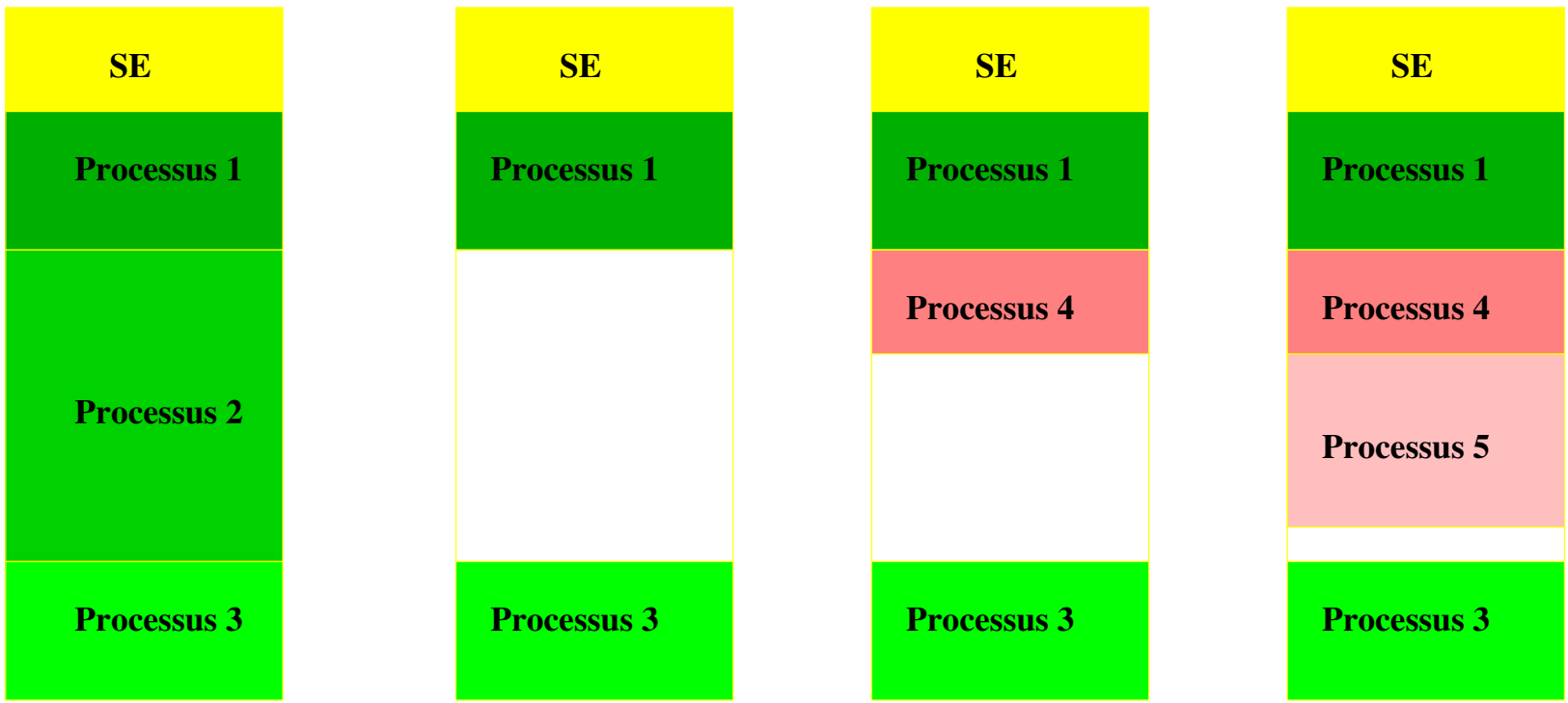
Implantation

`Charger le processus` p `dans une partition` P_i `libre si` $T(p) \leq T(P_i)$
T `: taille`

- Avantage : facile à implanter
- Inconvénients :
  - degré de multiprogrammation limité par le nombre des partitions
  - cas des processus dont la taille est supérieure ou inférieure à la taille des partitions ?
  - fragmentation interne
- Solution : partitions dynamiques

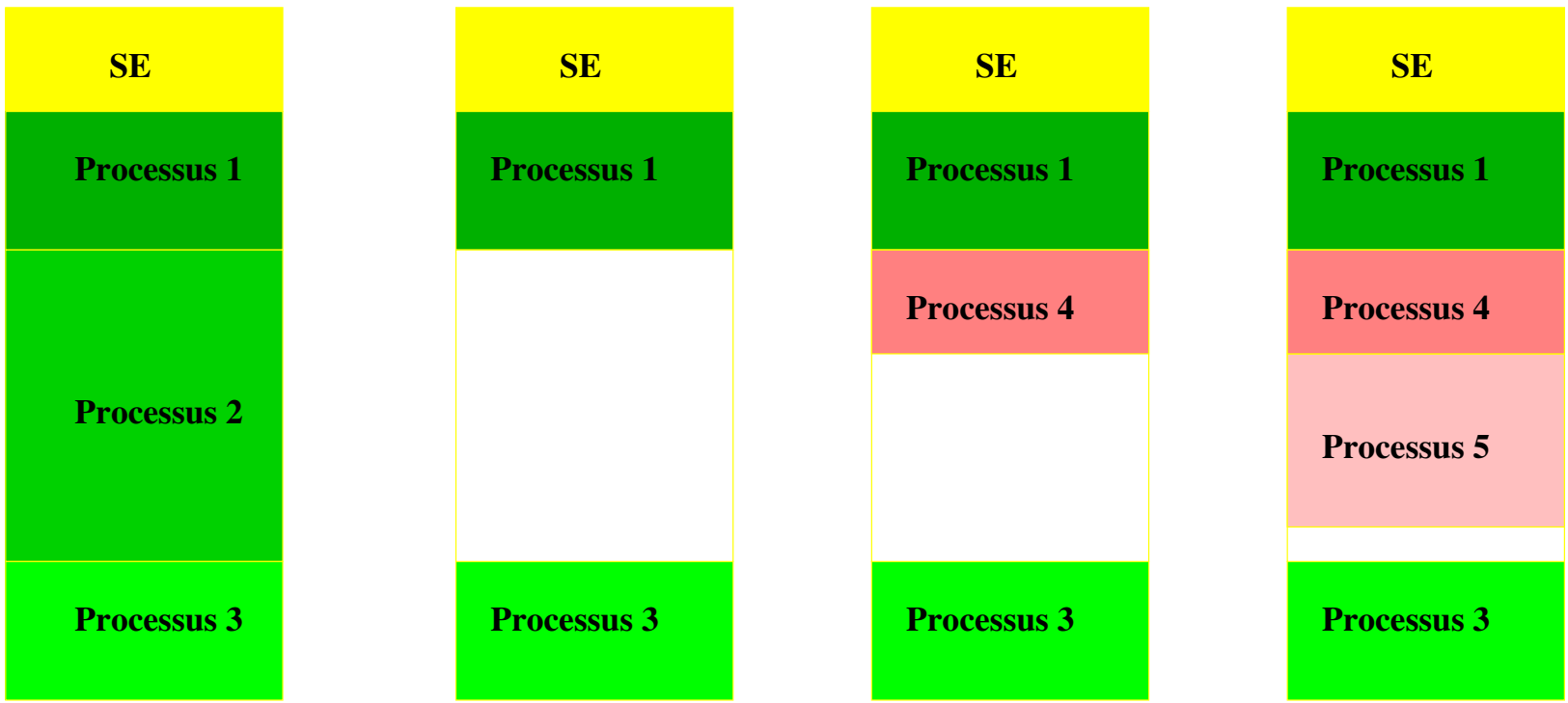
Mémoire uniforme - Partitions dynamiques

- La taille d'une partition $T(P_i)$ est déterminée en chargeant le programme p_i telle que $T(P_i) = T(p_i)$



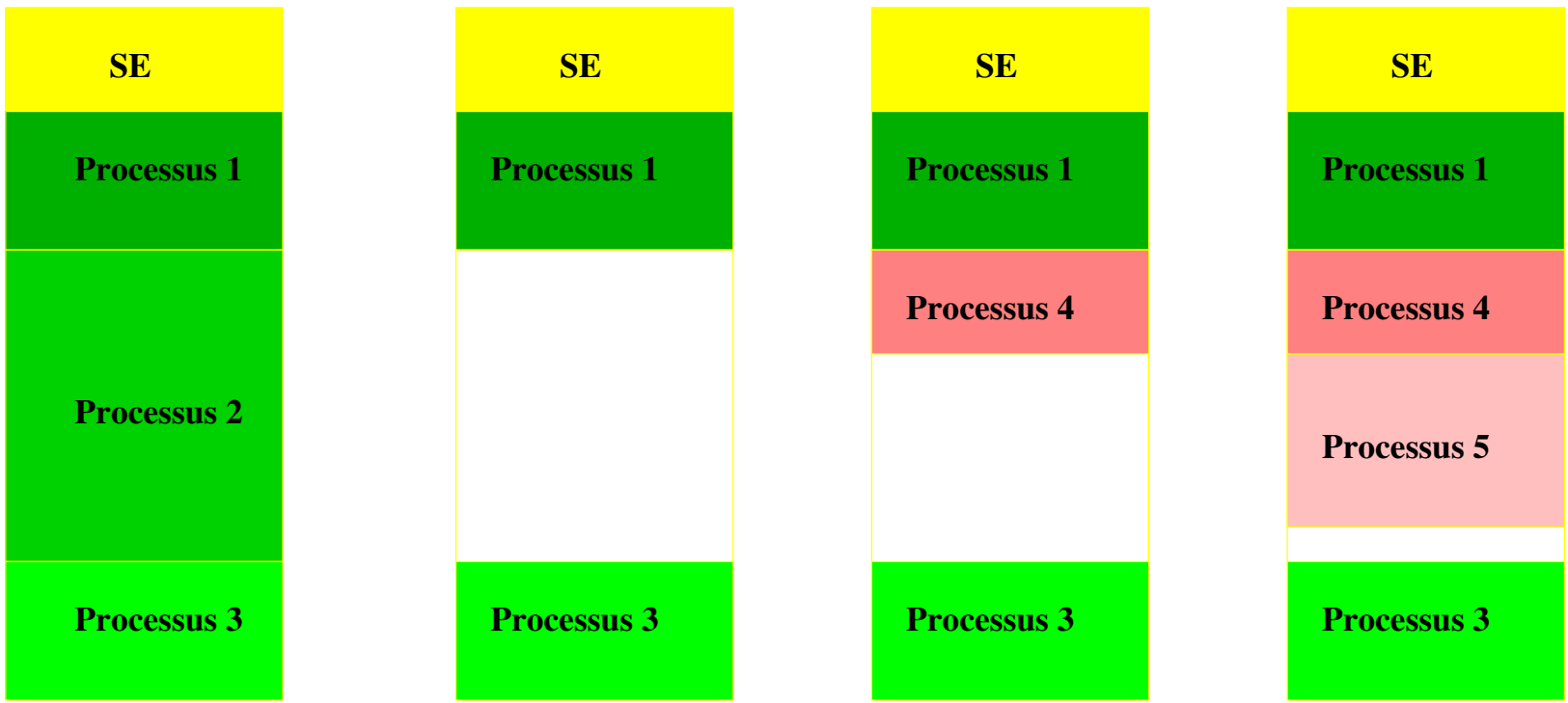
Mémoire uniforme - Partitions dynamiques

- La taille d'une partition $T(P_i)$ est déterminée en chargeant le programme p_i telle que $T(P_i) = T(p_i)$
- Avantages :
  - augmente le degré de multiprogrammation
  - élimine le problème de la fragmentation interne



Mémoire uniforme - Partitions dynamiques

- La taille d'une partition $T(P_i)$ est déterminée en chargeant le programme p_i telle que $T(P_i) = T(p_i)$
- Avantages :
  - augmente le degré de multiprogrammation
  - élimine le problème de la fragmentation interne
- Inconvénient : la fragmentation externe



Mémoire uniforme - Fragmentation externe

Le compactage est une solution possible si réimplantation dynamique (programmes translatables au moment de l'exécution) :

Mémoire uniforme - Fragmentation externe

Le compactage est une solution possible si réimplantation dynamique (programmes translatables au moment de l'exécution) :

- compacter tous les processus à chaque terminaison d'un processus :
  - solution chère

Mémoire uniforme - Fragmentation externe

Le compactage est une solution possible si réimplantation dynamique (programmes translatables au moment de l'exécution) :

- compacter tous les processus à chaque terminaison d'un processus :
  - solution chère
- compacter en cas de non possibilité d'allocation pour un processus :
  - compacter tous les processus
  - compacter un processus à la fois jusqu'à satisfaction de la requête

Mémoire uniforme - Fragmentation externe

Le compactage est une solution possible si réimplantation dynamique (programmes translatables au moment de l'exécution) :

- compacter tous les processus à chaque terminaison d'un processus :
  - solution chère
- compacter en cas de non possibilité d'allocation pour un processus :
  - compacter tous les processus
  - compacter un processus à la fois jusqu'à satisfaction de la requête
- Inconvénient : le compactage introduit de l'overhead

Mémoire uniforme - Fragmentation externe

Le compactage est une solution possible si réimplantation dynamique (programmes translatables au moment de l'exécution) :

- compacter tous les processus à chaque terminaison d'un processus :
  - solution chère
- compacter en cas de non possibilité d'allocation pour un processus :
  - compacter tous les processus
  - compacter un processus à la fois jusqu'à satisfaction de la requête
- Inconvénient : le compactage introduit de l'overhead
- une autre solution : la pagination

Partitions multiples : algorithmes d'allocation

First-fit

- allouer la première partition P_i libre telle que $T(P_i) \geq T(p)$
- Inconvénients :
  - tendance à produire des petits trous au début de la mémoire
  - vitesse de la recherche de plus en plus longue

Partitions multiples : algorithmes d'allocation

First-fit

- allouer la première partition P_i libre telle que $T(P_i) \geq T(p)$
- Inconvénients :
  - tendence à produire des petits trous au début de la mémoire
  - vitesse de la recherche de plus en plus longue

Best-fit

- allouer la plus petite partition libre telle que $T(P_i) \geq T(p)$
- Inconvénients :
  - recherche coûteuse sur toute la liste des partitions libres
  - fragmentation externe plus importante que les autres algorithmes

Partitions multiples : algorithmes d'allocation

First-fit

- allouer la première partition P_i libre telle que $T(P_i) \geq T(p)$
- Inconvénients :
  - tendence à produire des petits trous au début de la mémoire
  - vitesse de la recherche de plus en plus longue

Best-fit

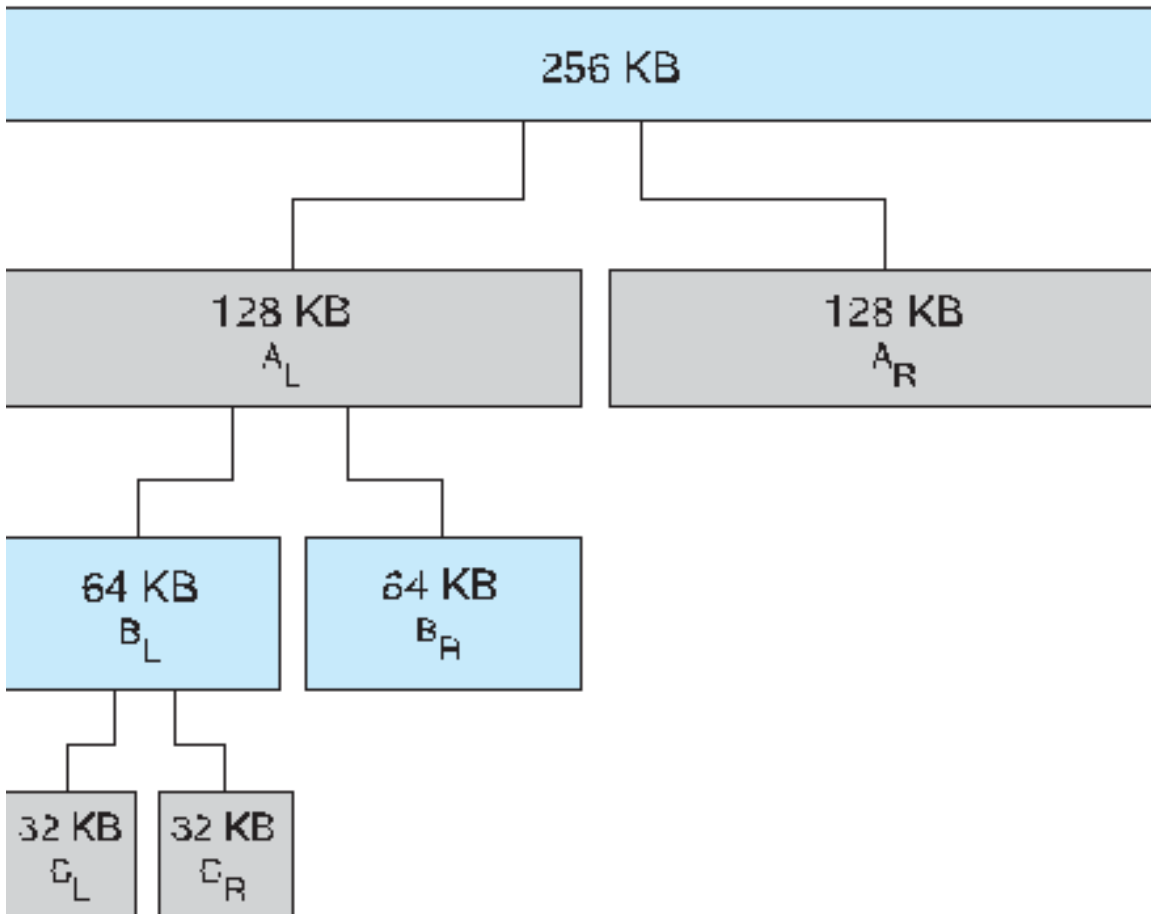
- allouer la plus petite partition libre telle que $T(P_i) \geq T(p)$
- Inconvénients :
  - recherche coûteuse sur toute la liste des partitions libres
  - fragmentation externe plus importante que les autres algorithmes

Worst-fit

- allouer la plus grande partition libre telle que $T(P_i) \geq T(p)$
- produit des trous les plus grands possibles

Algorithme des frères siamois (buddy system)

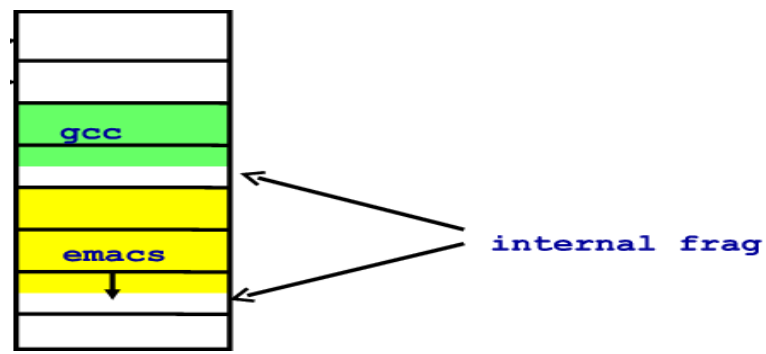
- Donald KNUTH [1973]
- Liste de blocs libres dont la taille est une puissance de 2 (1, 2, 4, 8 octets, ..., jusqu'à la taille maximale de la mémoire)



Pagination

- la mémoire physique est divisée en blocs de taille fixe appelés cadres de pages ou cases

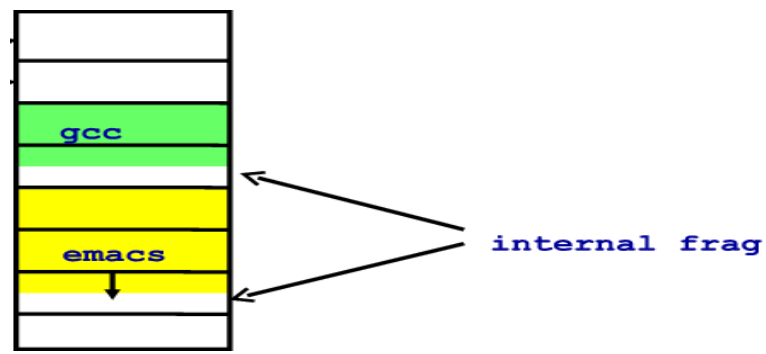
Remarques



Pagination

- la mémoire physique est divisée en blocs de taille fixe appelés cadres de pages ou cases
- en termes de mémoire logique, on parle de pages

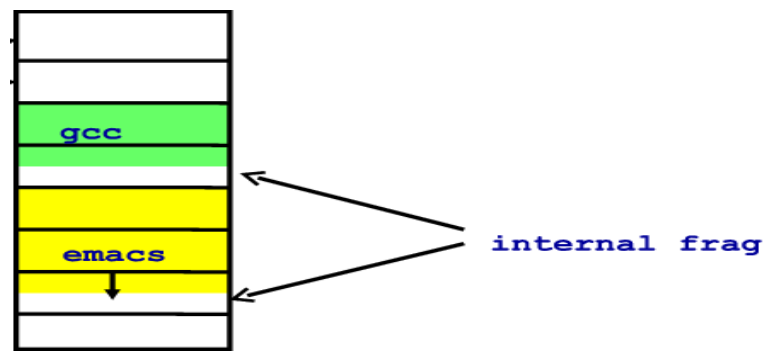
Remarques



Pagination

- la mémoire physique est divisée en blocs de taille fixe appelés cadres de pages ou cases
- en termes de mémoire logique, on parle de pages
- la taille des cases est une puissance de 2, généralement entre 512 et 8 192 octets

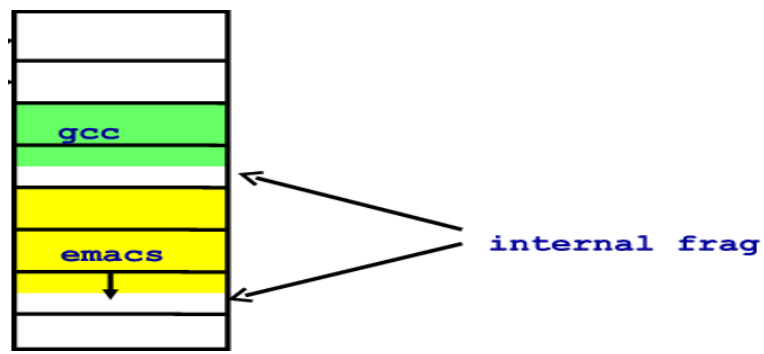
Remarques



Pagination

- la mémoire physique est divisée en blocs de taille fixe appelés cadres de pages ou cases
- en termes de mémoire logique, on parle de pages
- la taille des cases est une puissance de 2, généralement entre 512 et 8 192 octets
- la taille d'un cadre est définie par le matériel

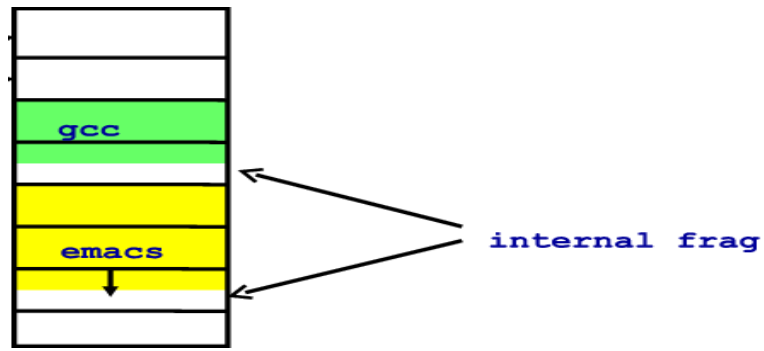
Remarques



Pagination

- la mémoire physique est divisée en blocs de taille fixe appelés cadres de pages ou cases
- en termes de mémoire logique, on parle de pages
- la taille des cases est une puissance de 2, généralement entre 512 et 8 192 octets
- la taille d'un cadre est définie par le matériel
- lorsque l'on veut exécuter un programme, on charge ses pages dans les cases disponibles à partir de la mémoire auxiliaire

Remarques

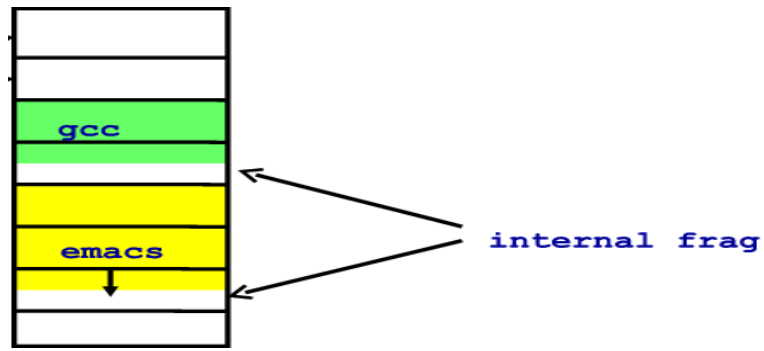


Pagination

- la mémoire physique est divisée en blocs de taille fixe appelés cadres de pages ou cases
- en termes de mémoire logique, on parle de pages
- la taille des cases est une puissance de 2, généralement entre 512 et 8 192 octets
- la taille d'un cadre est définie par le matériel
- lorsque l'on veut exécuter un programme, on charge ses pages dans les cases disponibles à partir de la mémoire auxiliaire

Remarques

- devoir garder trace de toutes les cases libres

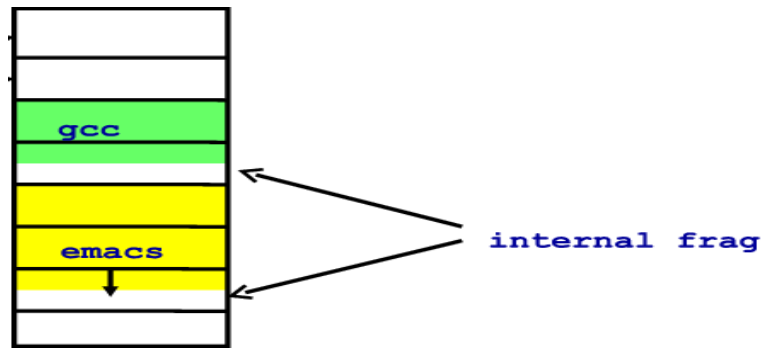


Pagination

- la mémoire physique est divisée en blocs de taille fixe appelés cadres de pages ou cases
- en termes de mémoire logique, on parle de pages
- la taille des cases est une puissance de 2, généralement entre 512 et 8 192 octets
- la taille d'un cadre est définie par le matériel
- lorsque l'on veut exécuter un programme, on charge ses pages dans les cases disponibles à partir de la mémoire auxiliaire

Remarques

- devoir garder trace de toutes les cases libres
- allocation non contigüe

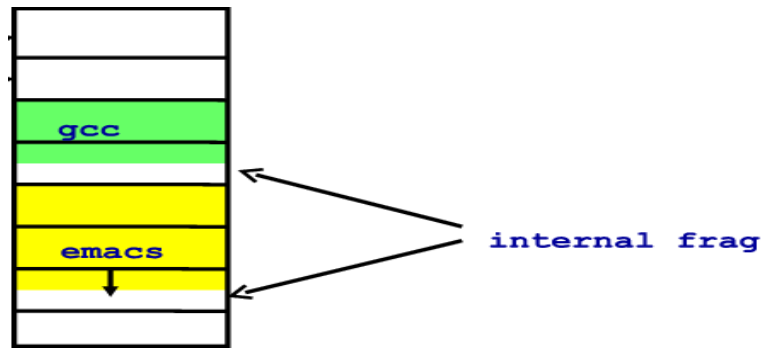


Pagination

- la mémoire physique est divisée en blocs de taille fixe appelés cadres de pages ou cases
- en termes de mémoire logique, on parle de pages
- la taille des cases est une puissance de 2, généralement entre 512 et 8 192 octets
- la taille d'un cadre est définie par le matériel
- lorsque l'on veut exécuter un programme, on charge ses pages dans les cases disponibles à partir de la mémoire auxiliaire

Remarques

- devoir garder trace de toutes les cases libres
- allocation non contigüe
- une solution à la fragmentation extérieure

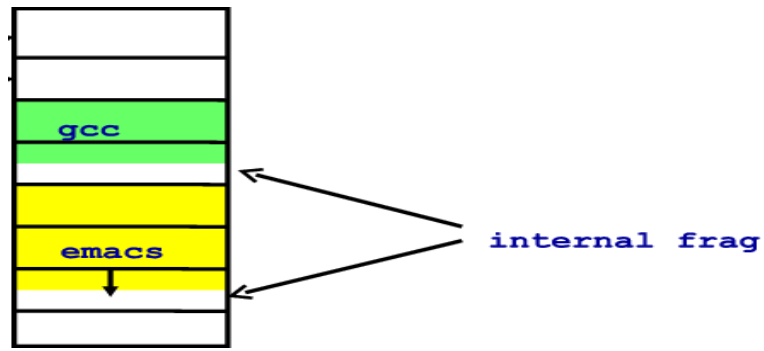


Pagination

- la mémoire physique est divisée en blocs de taille fixe appelés cadres de pages ou cases
- en termes de mémoire logique, on parle de pages
- la taille des cases est une puissance de 2, généralement entre 512 et 8 192 octets
- la taille d'un cadre est définie par le matériel
- lorsque l'on veut exécuter un programme, on charge ses pages dans les cases disponibles à partir de la mémoire auxiliaire

Remarques

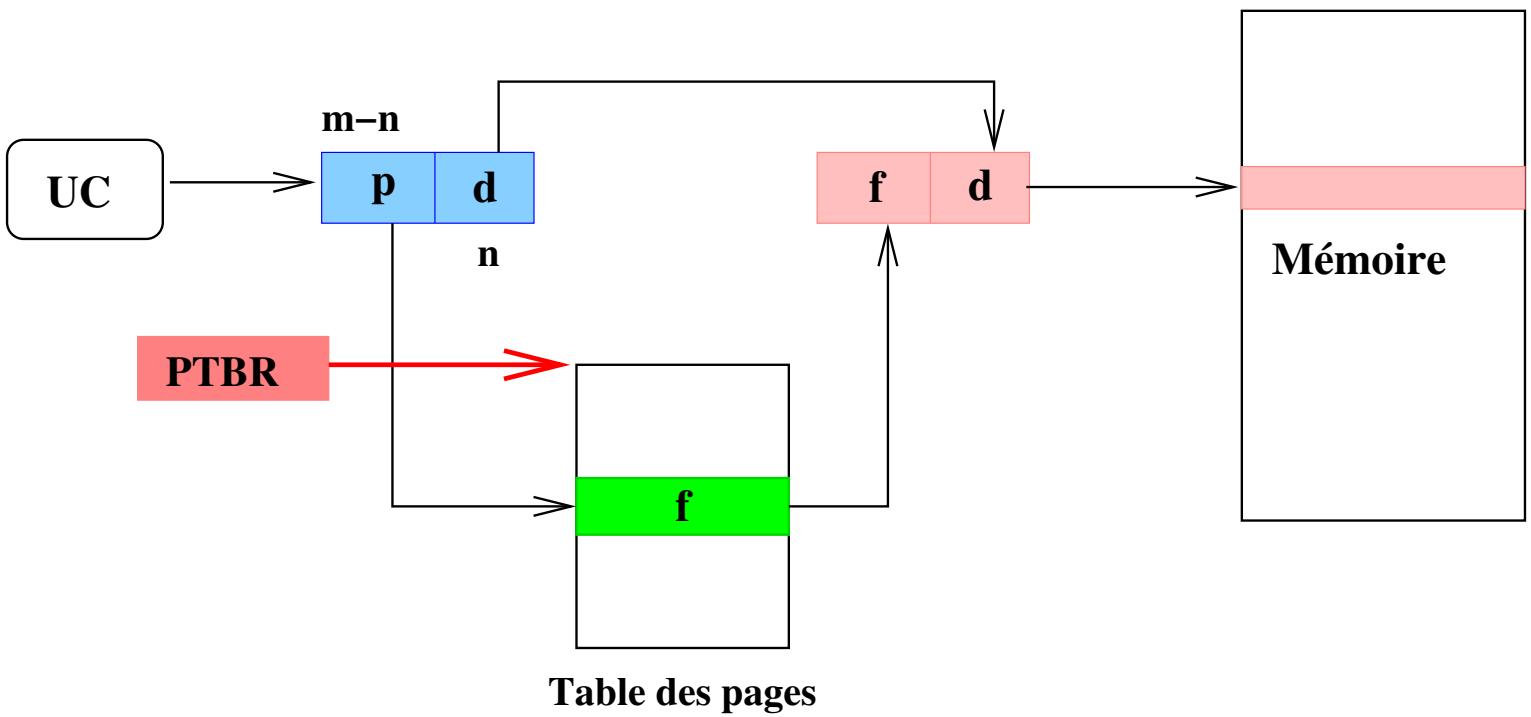
- devoir garder trace de toutes les cases libres
- allocation non contigüe
- une solution à la fragmentation extérieure
- limite le problème de la fragmentation interne



Pagination - Conversion @logique $\rightarrow$ @physique

- la table des pages (TP) est maintenue en mémoire

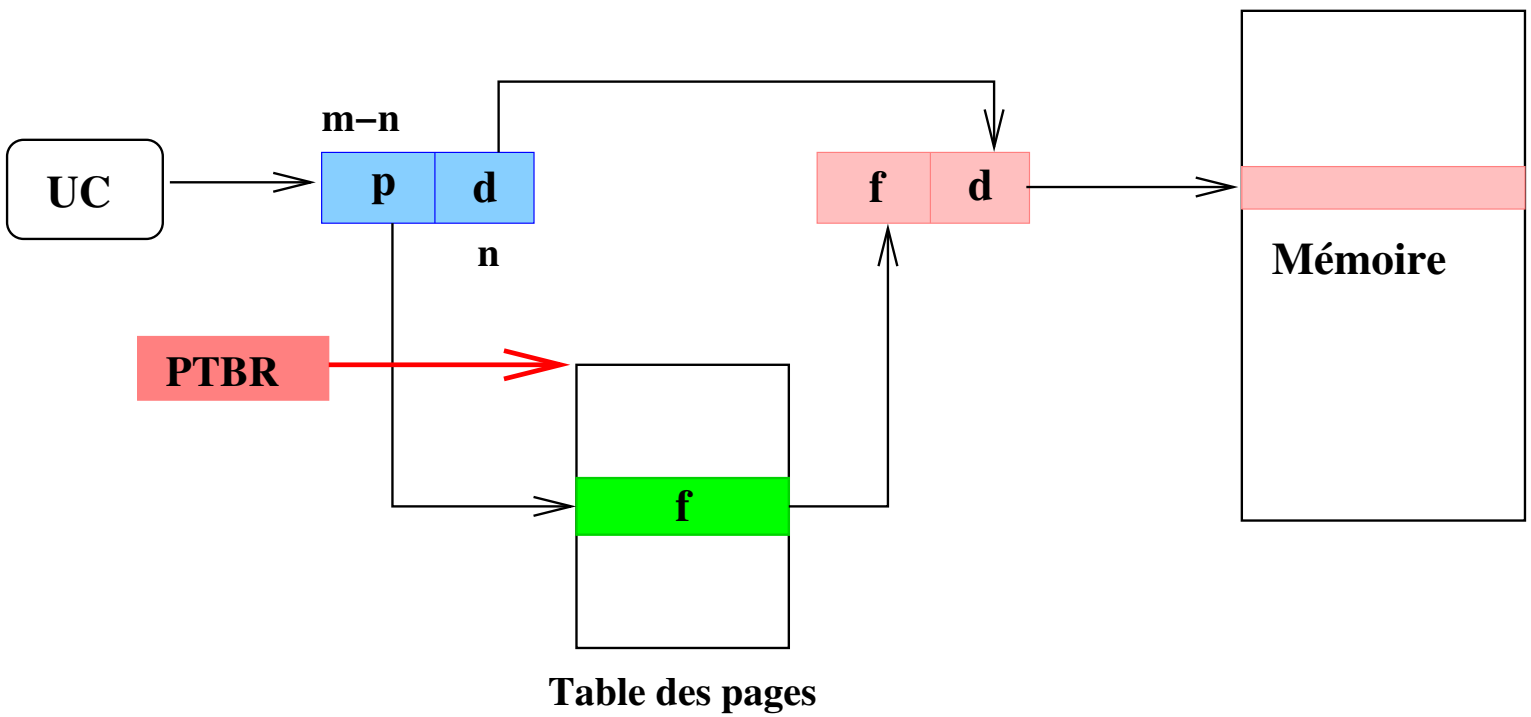
$$(p, d) \rightarrow TP[p] \times T(page) + d$$



Pagination - Conversion @logique $\rightarrow$ @physique

- la table des pages (*TP*) est maintenue en mémoire
- Page-table base register (PTBR) pointe vers la table des pages

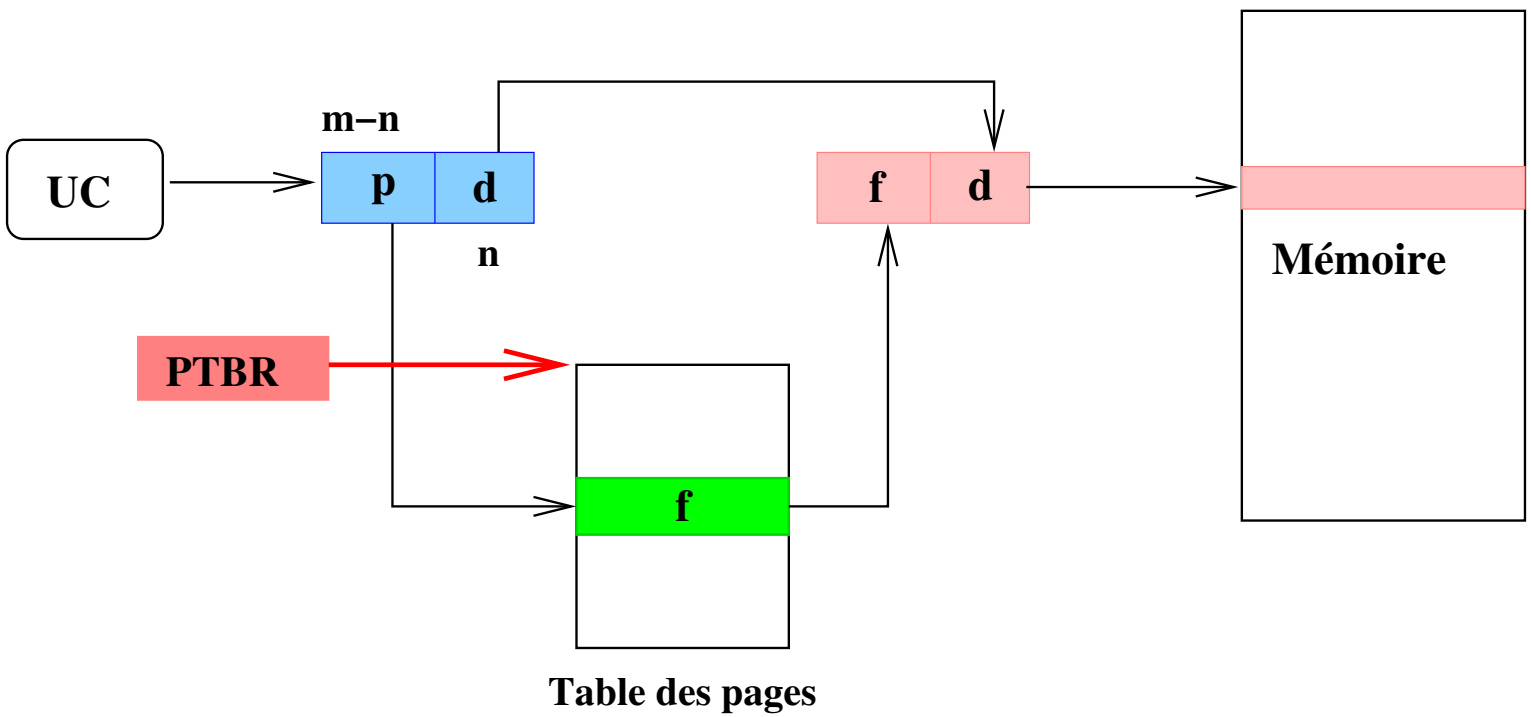
$$(p, d) \rightarrow TP[p] \times T(page) + d$$



Pagination - Conversion @logique $\rightarrow$ @physique

- la table des pages (*TP*) est maintenue en mémoire
- Page-table base register (PTBR) pointe vers la table des pages
- Page-table length register (PRLR) indique la taille de la table des pages

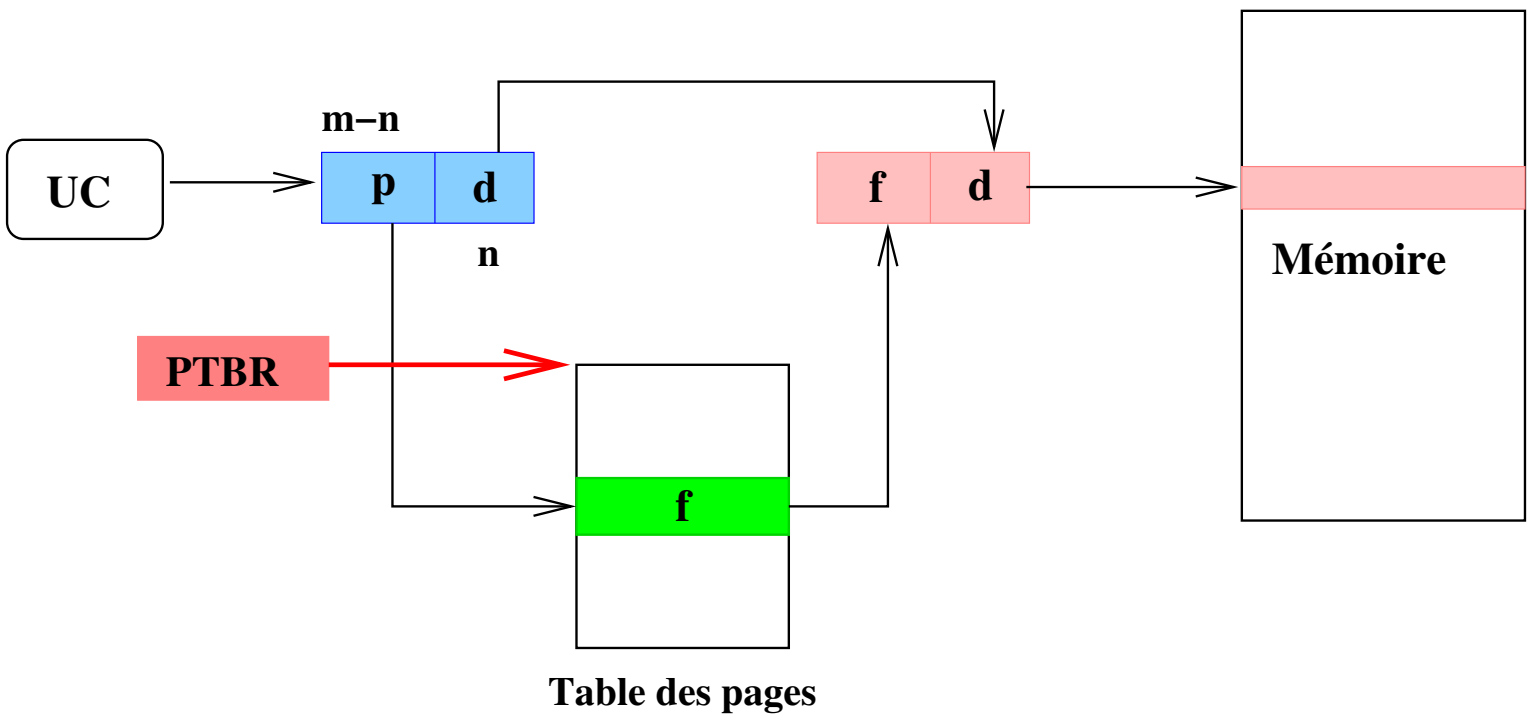
$$(p, d) \rightarrow TP[p] \times T(page) + d$$



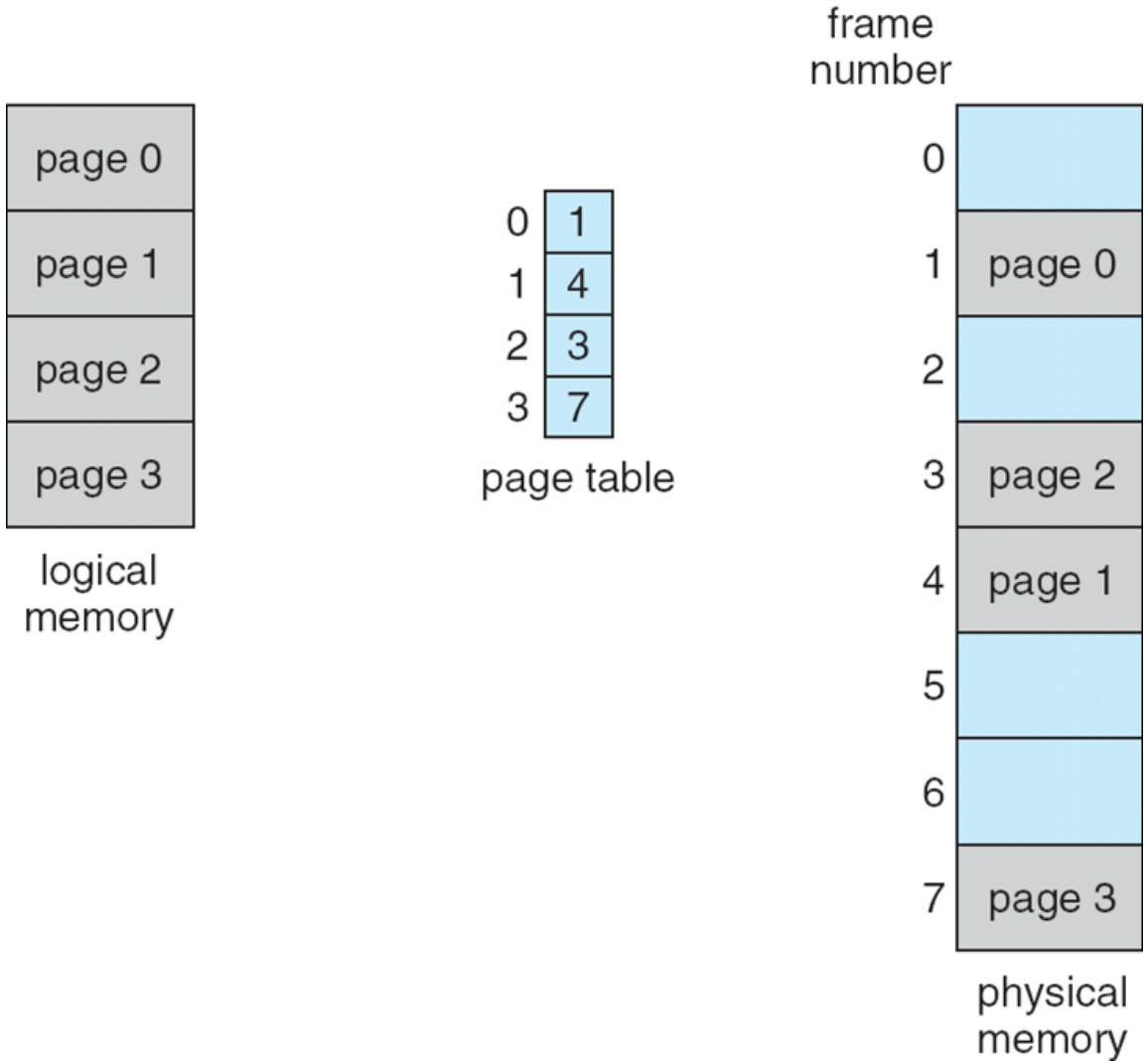
Pagination - Conversion @logique $\rightarrow$ @physique

- la table des pages (TP) est maintenue en mémoire
- Page-table base register (PTBR) pointe vers la table des pages
- Page-table length register (PRLR) indique la taille de la table des pages
- Machine m bits avec $T(page) = 2^n$

$$(p, d) \rightarrow TP[p] \times T(page) + d$$

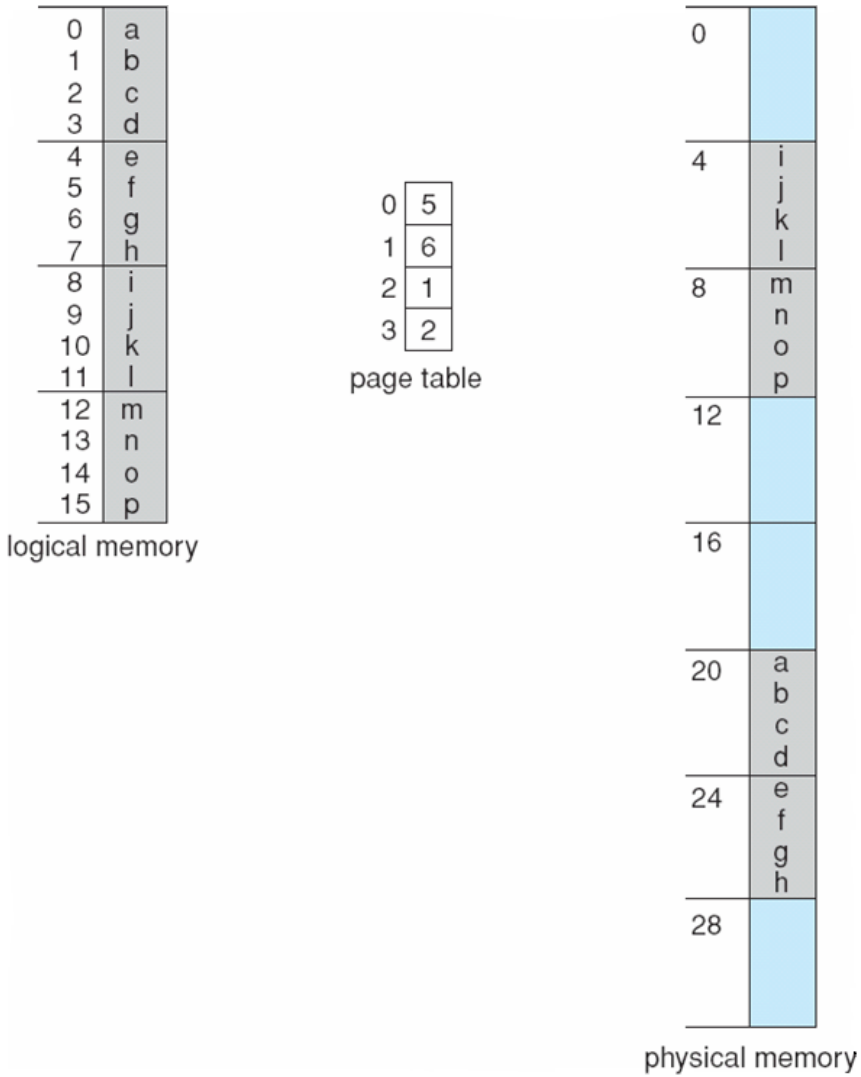


Pagination - Exemple (1/3)



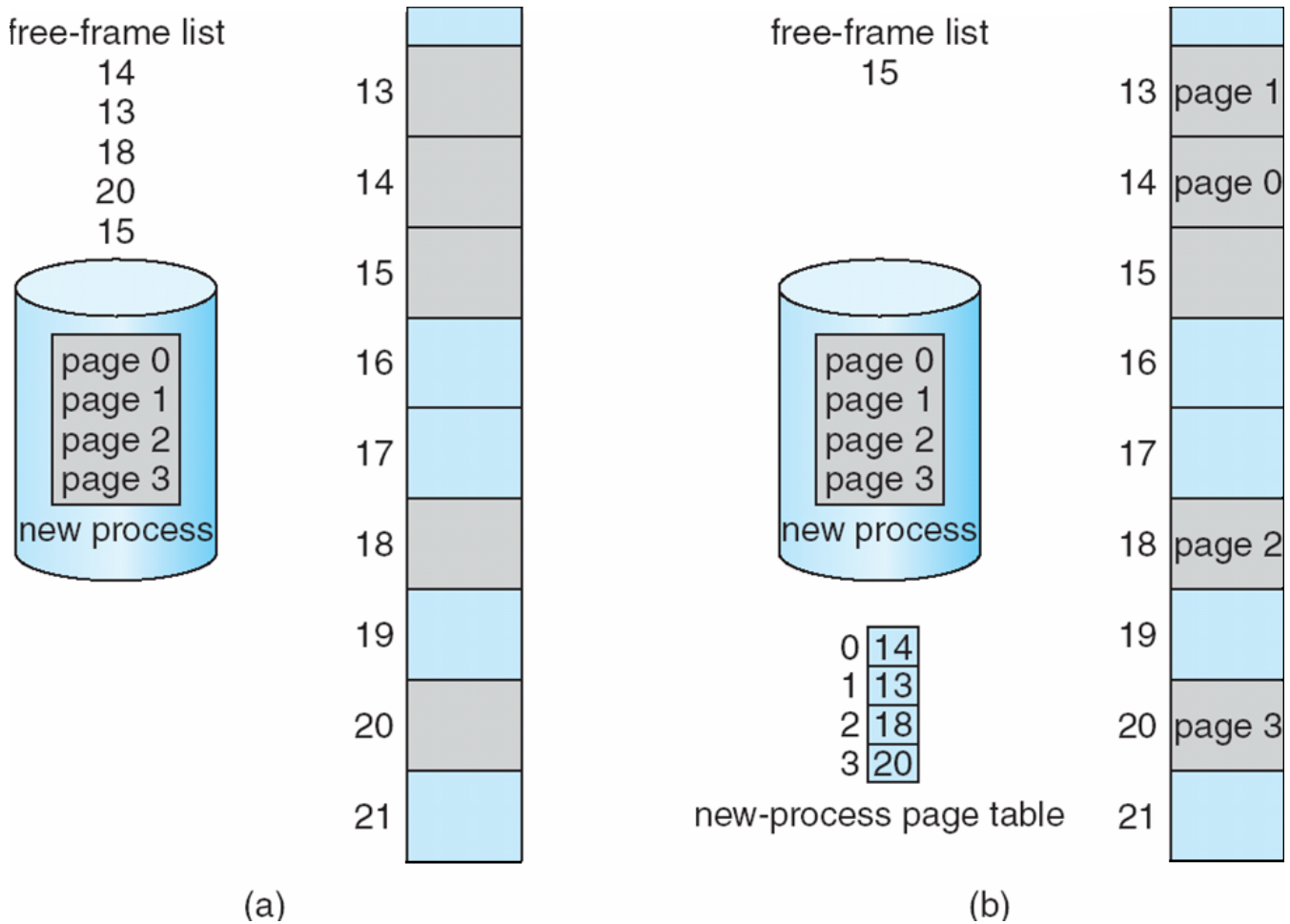
Pagination - Exemple (2/3)

- $n = 2$, $m = 4$
- 32-byte memory
- 4-byte pages



Pagination - Exemple (3/3)

Liste des cadres libres



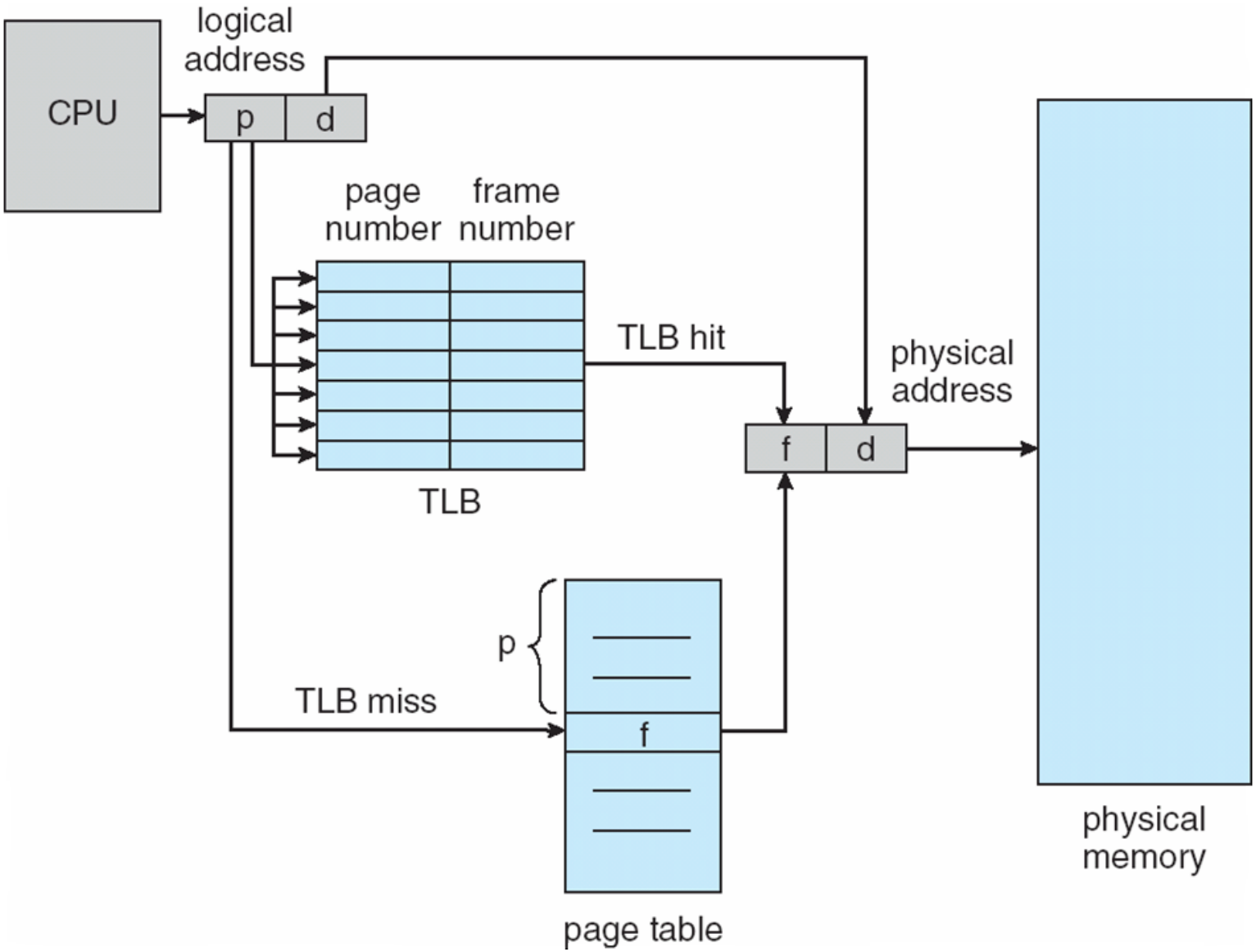
Before allocation

After allocation

Translation look-aside buffers

- 2 accès mémoire sont nécessaires pour accéder à une donnée/instruction.
- Solution : utilisation d'une mémoire cache assez rapide appelée mémoire associative ou translation look-aside buffers (TLBs)
- Certaines TLBs assure la protection inter-processus en mémorisant un identificateur pour chaque processus : address-space identifiers (ASIDs)
- Translation $(p, d) \rightarrow (c, d)$:
  si p est disponible dans la TLB, alors récupérer c, sinon aller à la table des pages en mémoire.

Pagination - TLB (illustration)



Pagination multiniveaux (1/3)

- la table des pages doit être entièrement chargée en mémoire physique.

Pagination multiniveaux (1/3)

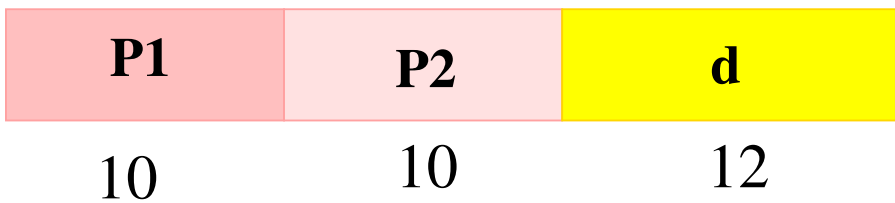
- la table des pages doit être entièrement chargée en mémoire physique.
- Exemple. une machine 32 bits avec des pages de 4K.
  Une adresse logique : numéro de page (20 bits), déplacement (12 bits)
  $\Rightarrow$ taille de la TP est $2^{20} \times 4$ octets

Pagination multiniveaux (1/3)

- la table des pages doit être entièrement chargée en mémoire physique.
- Exemple. une machine 32 bits avec des pages de 4K.
  Une adresse logique : numéro de page (20 bits), déplacement (12 bits)
  $\Rightarrow$ taille de la TP est $2^{20} \times 4$ octets
- la pagination multi-niveaux permet de limiter la taille de la table des pages qui doit être résidente en mémoire.

Pagination multiniveaux (1/3)

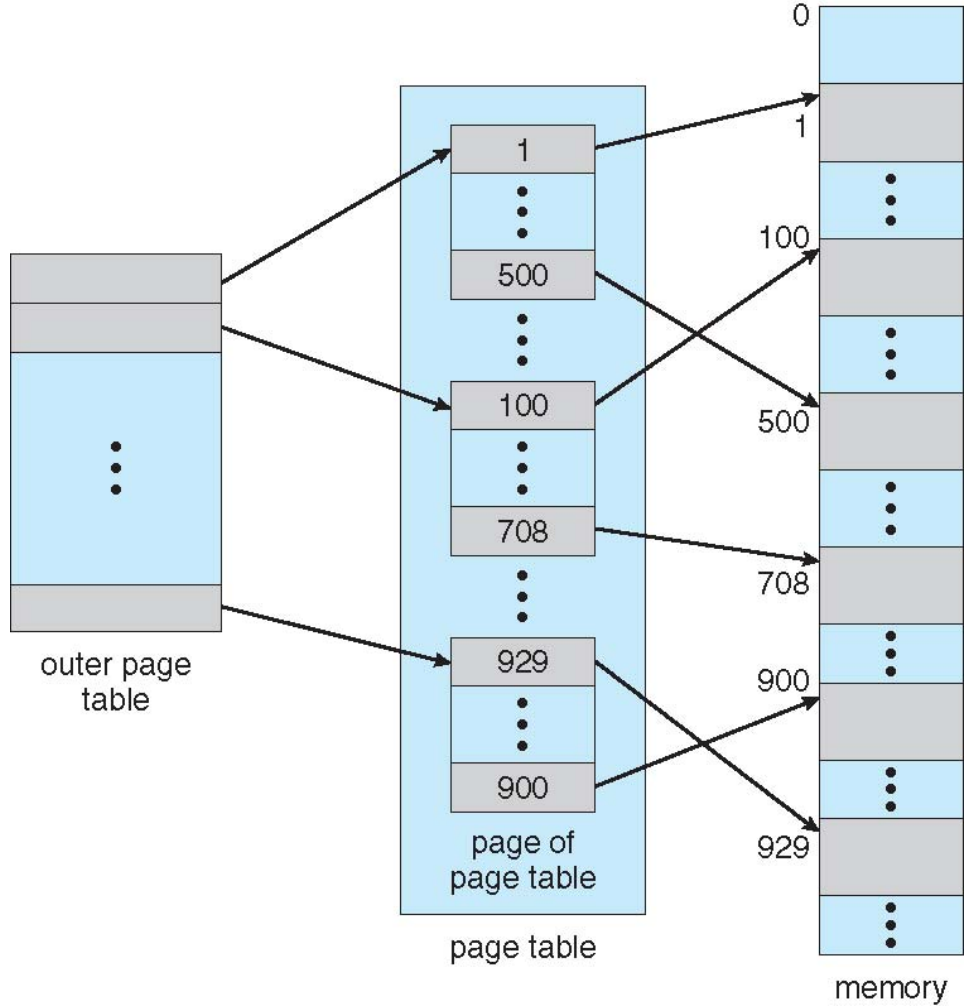
- la table des pages doit être entièrement chargée en mémoire physique.
- Exemple. une machine 32 bits avec des pages de 4K.
  Une adresse logique : numéro de page (20 bits), déplacement (12 bits)
  $\Rightarrow$ taille de la TP est $2^{20} \times 4$ octets
- la pagination multi-niveaux permet de limiter la taille de la table des pages qui doit être résidente en mémoire.
- cas de 2 niveaux, une adresse logique prend la forme :



Pagination multiniveaux (2/3)

P1	P2	d
10	10	12

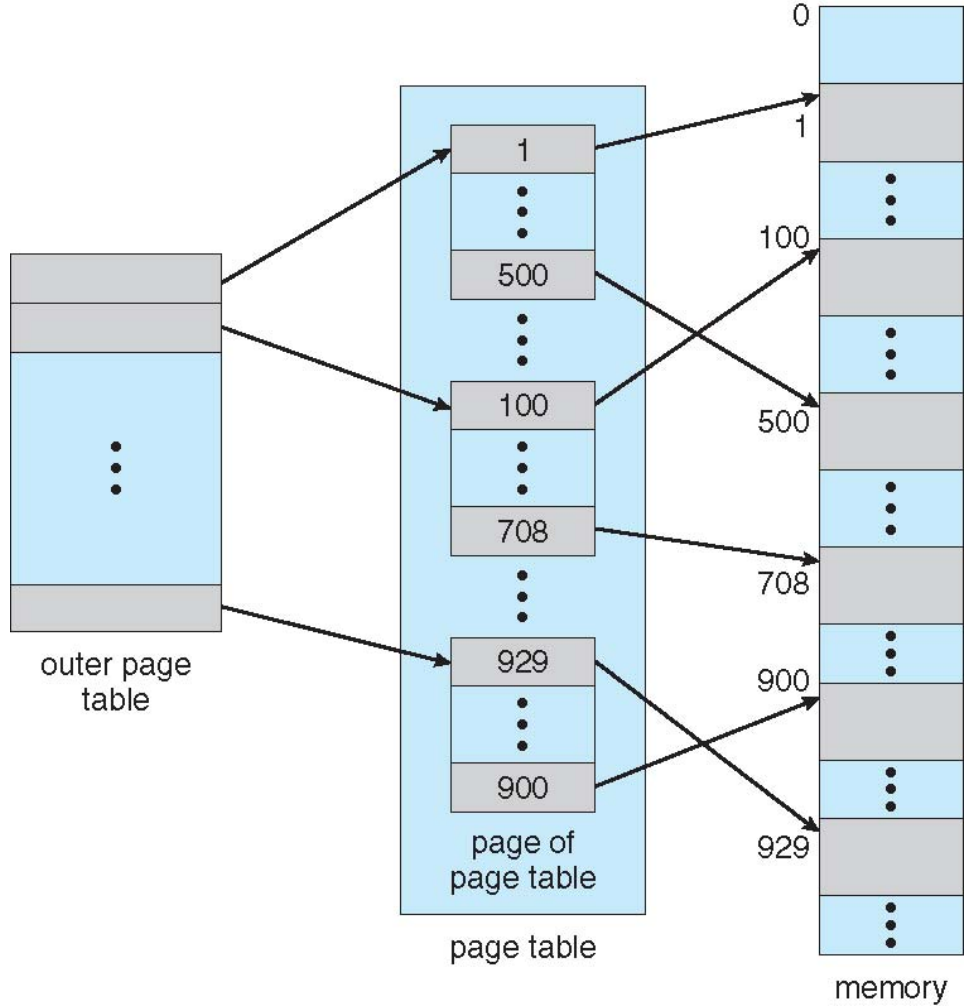
- la mémoire nécessaire pour les différentes TP est de $(2^{10} + 2^{10} \times 2^{10}) \times 4$ octets.



Pagination multiniveaux (2/3)

P1	P2	d
10	10	12

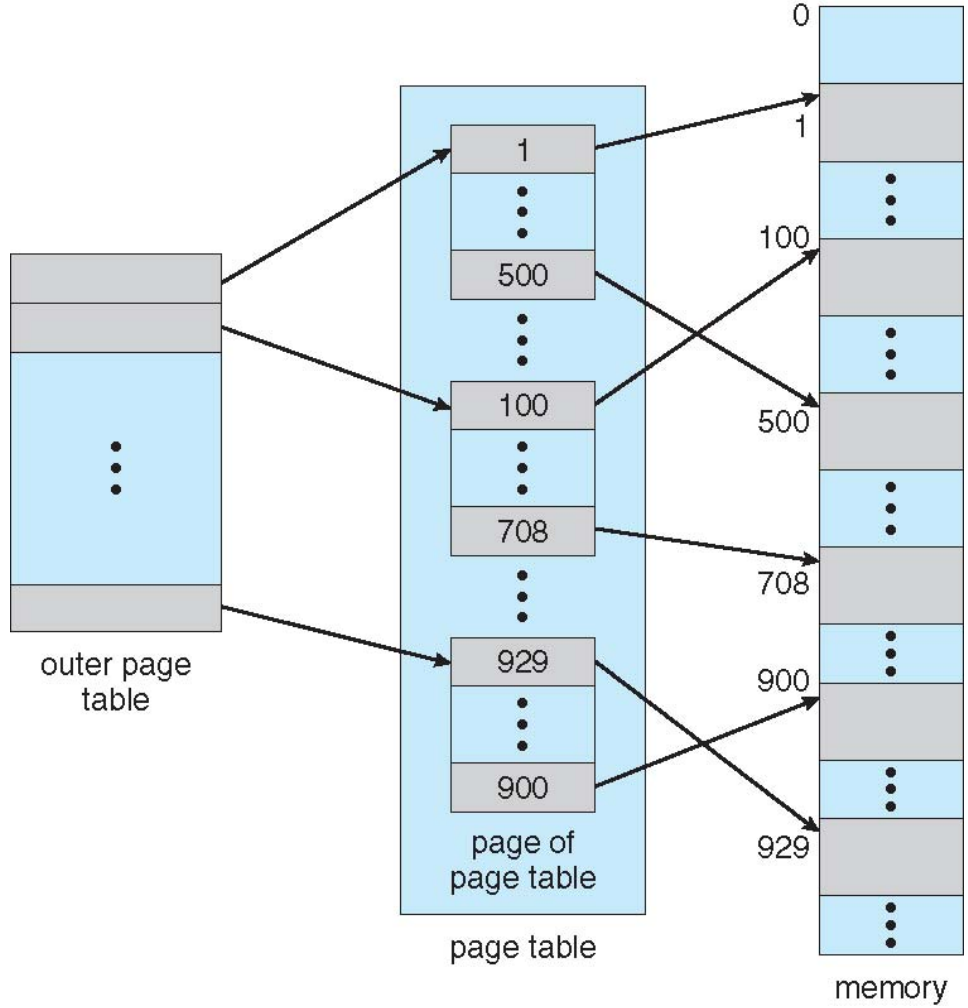
- la mémoire nécessaire pour les différentes TP est de $(2^{10} + 2^{10} \times 2^{10}) \times 4$ octets.
- mais, il n' y a que la TP de niveau 1 qui doit résider en mémoire.



Pagination multiniveaux (2/3)

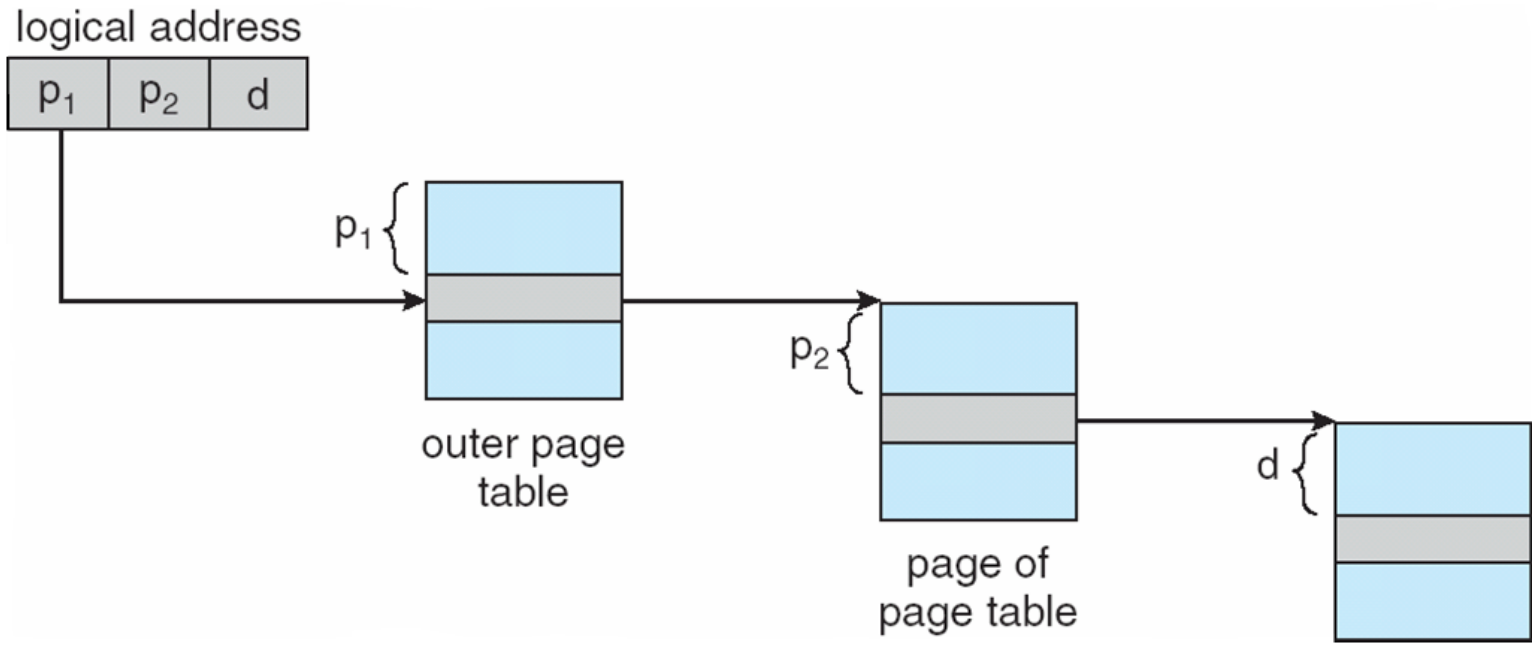
P1	P2	d
10	10	12

- la mémoire nécessaire pour les différentes TP est de $(2^{10} + 2^{10} \times 2^{10}) \times 4$ octets.
- mais, il n' y a que la TP de niveau 1 qui doit résider en mémoire.
- cas d'un programme de 2^{10} pages, besoin de seulement $2^{10} + 2^{10} = 2^{11}$ entrées résidentes en mémoire



Pagination multiniveaux (3/3)

Traduction d'adresses



Pagination - Table des pages inversée

Le problème :

- une machine 32 bits avec une taille de pages de 1K
- une table de pages (associée à chaque processus) a 2^{22} entrées
- comment peut-on réduire cela ?

Pagination - Table des pages inversée

Le problème :

- une machine 32 bits avec une taille de pages de 1K
- une table de pages (associée à chaque processus) a 2^{22} entrées
- comment peut-on réduire cela ?

Solution : table des pages inversée

- une seule table pour tout le système
- une entrée par case (cadre de page) mémoire

Pagination - Table des pages inversée

Le problème :

- une machine 32 bits avec une taille de pages de 1K
- une table de pages (associée à chaque processus) a 2^{22} entrées
- comment peut-on réduire cela ?

Solution : table des pages inversée

- une seule table pour tout le système
- une entrée par case (cadre de page) mémoire
- Schéma utilisé par certaines stations de travail IBM et HP

Pagination - Table des pages inversée

Le problème :

- une machine 32 bits avec une taille de pages de 1K
- une table de pages (associée à chaque processus) a 2^{22} entrées
- comment peut-on réduire cela ?

Solution : table des pages inversée

- une seule table pour tout le système
- une entrée par case (cadre de page) mémoire
- Schéma utilisé par certaines stations de travail IBM et HP
- Inconvénient : parcourir la table pour trouver une correspondance

Pagination - Table des pages inversée

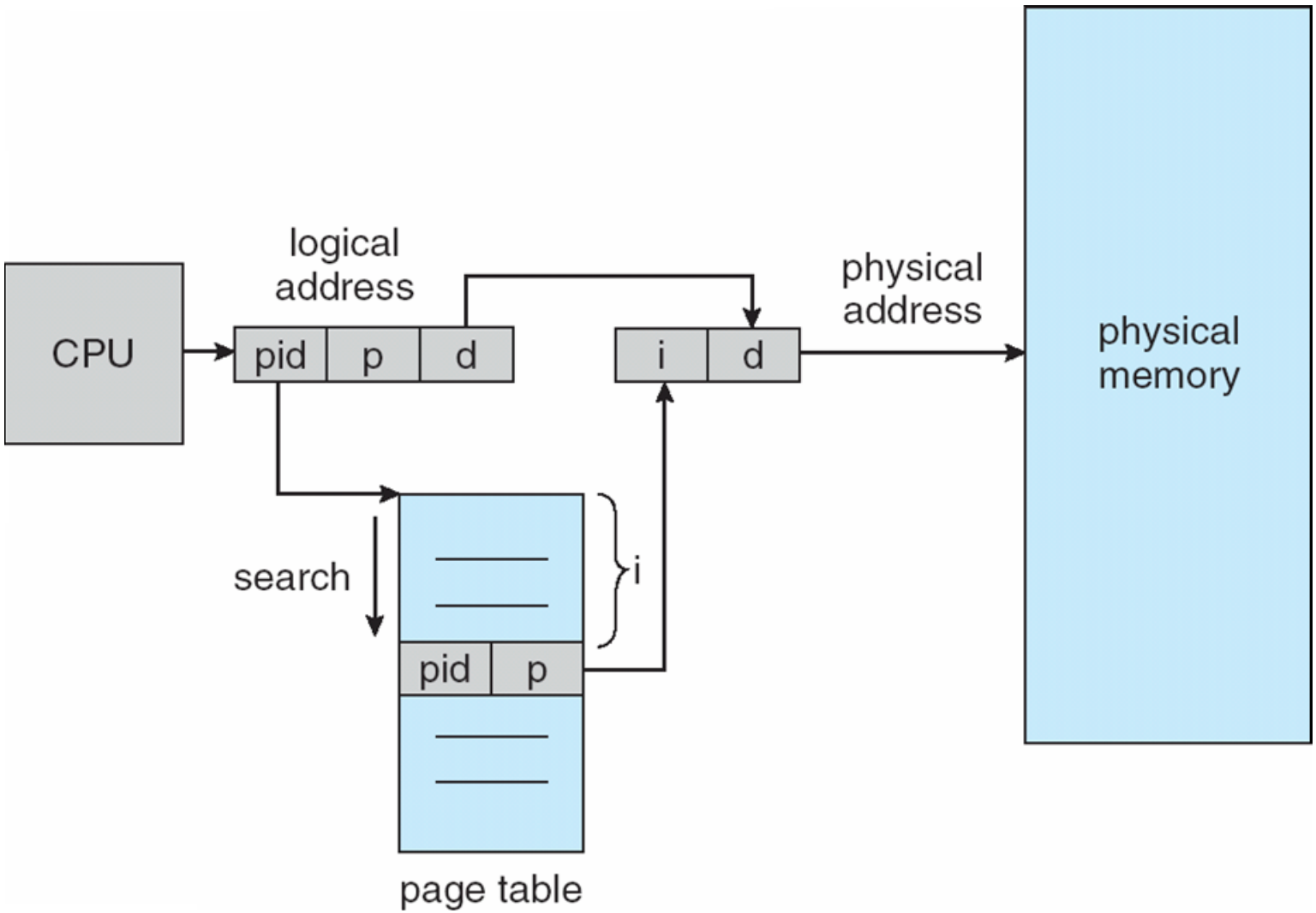
Le problème :

- une machine 32 bits avec une taille de pages de 1K
- une table de pages (associée à chaque processus) a 2^{22} entrées
- comment peut-on réduire cela ?

Solution : table des pages inversée

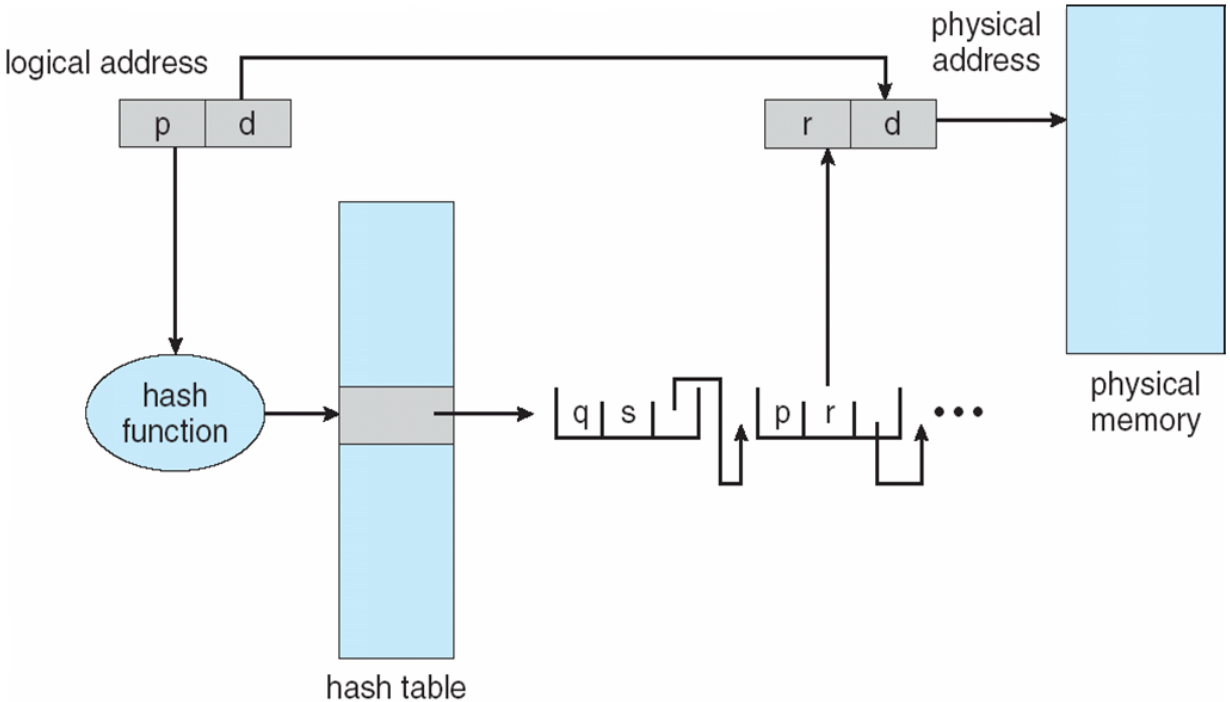
- une seule table pour tout le système
- une entrée par case (cadre de page) mémoire
- Schéma utilisé par certaines stations de travail IBM et HP
- Inconvénient : parcourir la table pour trouver une correspondance
- Solutions : TLB, hashage.

Pagination - Table des pages inversée illustrée



Pagination - Table des pages Hashée

Motivation : réduire la complexité dans les systèmes $>$ 32 bits

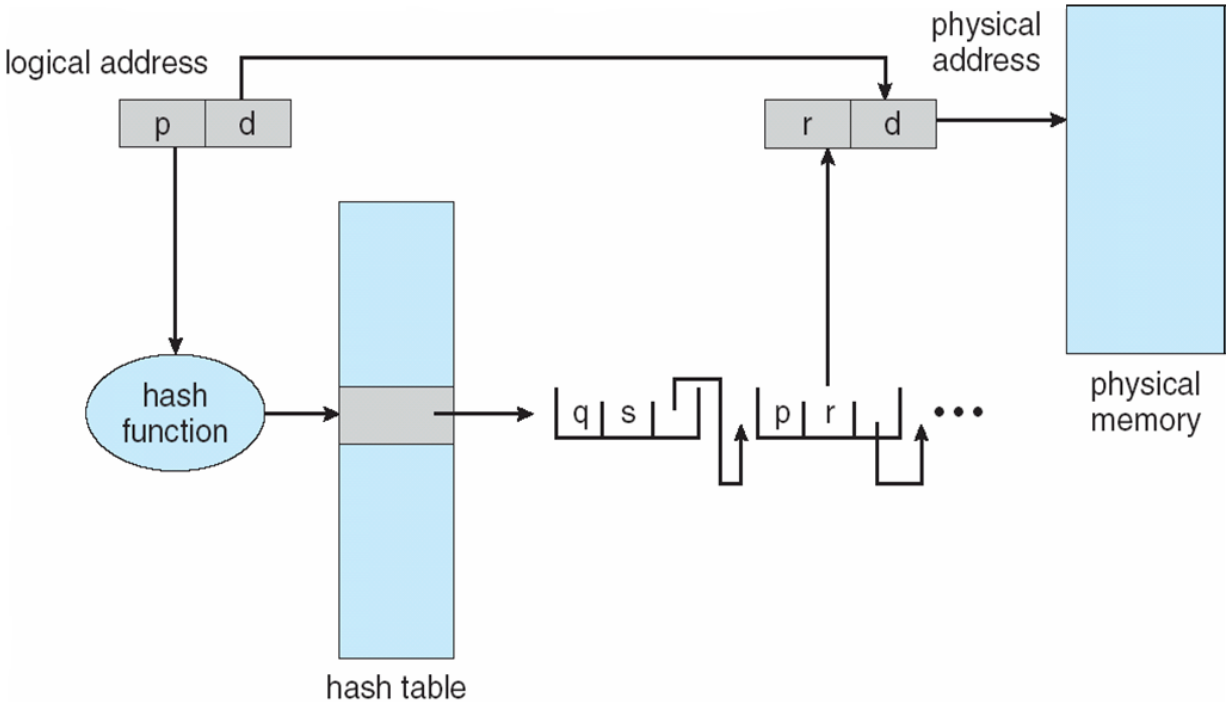


Principe de hashage

- une fonction surjective qui associe à chaque clé, un ou plusieurs éléments

Pagination - Table des pages Hashée

Motivation : réduire la complexité dans les systèmes $>$ 32 bits

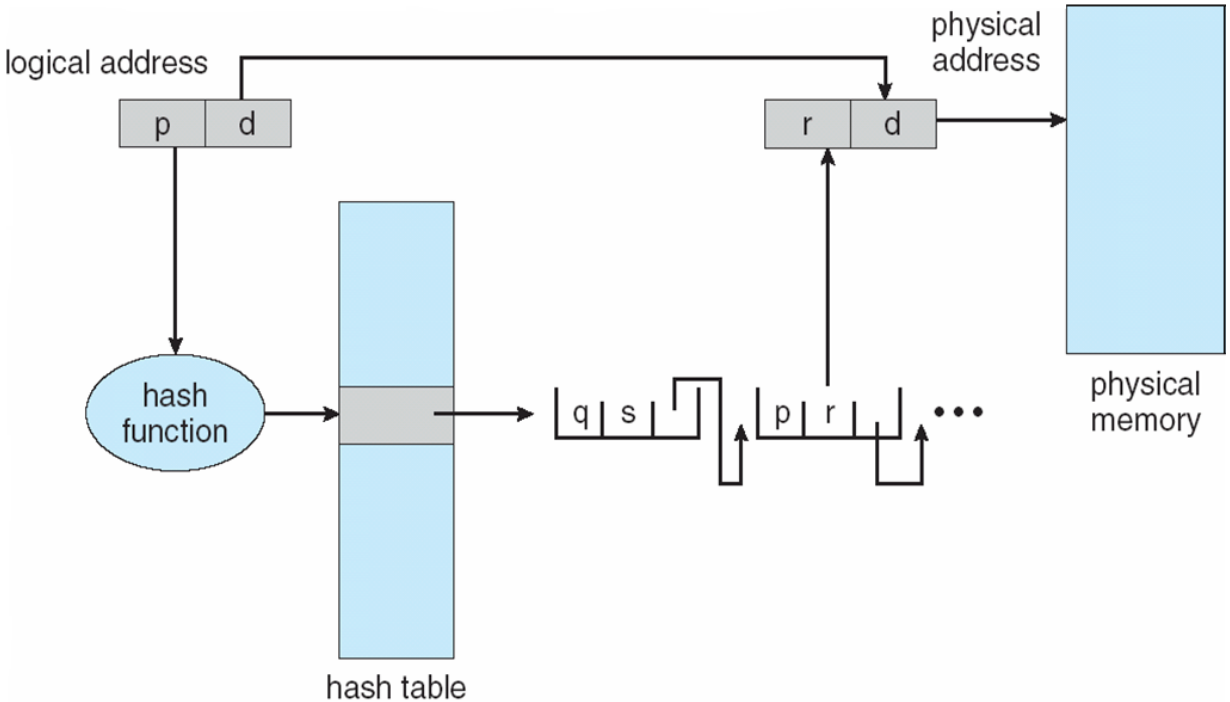


Principe de hashage

- une fonction surjective qui associe à chaque clé, un ou plusieurs éléments
- au lieu de stocker tous les éléments dans un tableau :

Pagination - Table des pages Hashée

Motivation : réduire la complexité dans les systèmes $>$ 32 bits

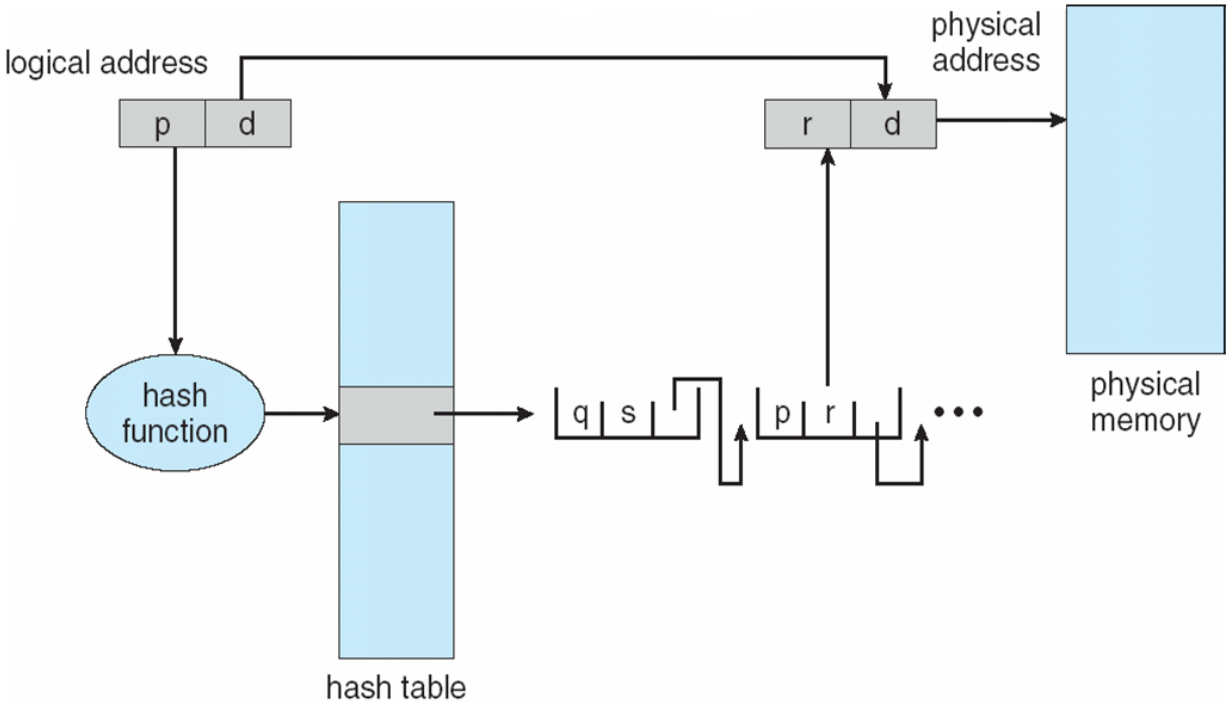


Principe de hashage

- une fonction surjective qui associe à chaque clé, un ou plusieurs éléments
- au lieu de stocker tous les éléments dans un tableau :
  - une entrée par clé

Pagination - Table des pages Hashée

Motivation : réduire la complexité dans les systèmes $>$ 32 bits

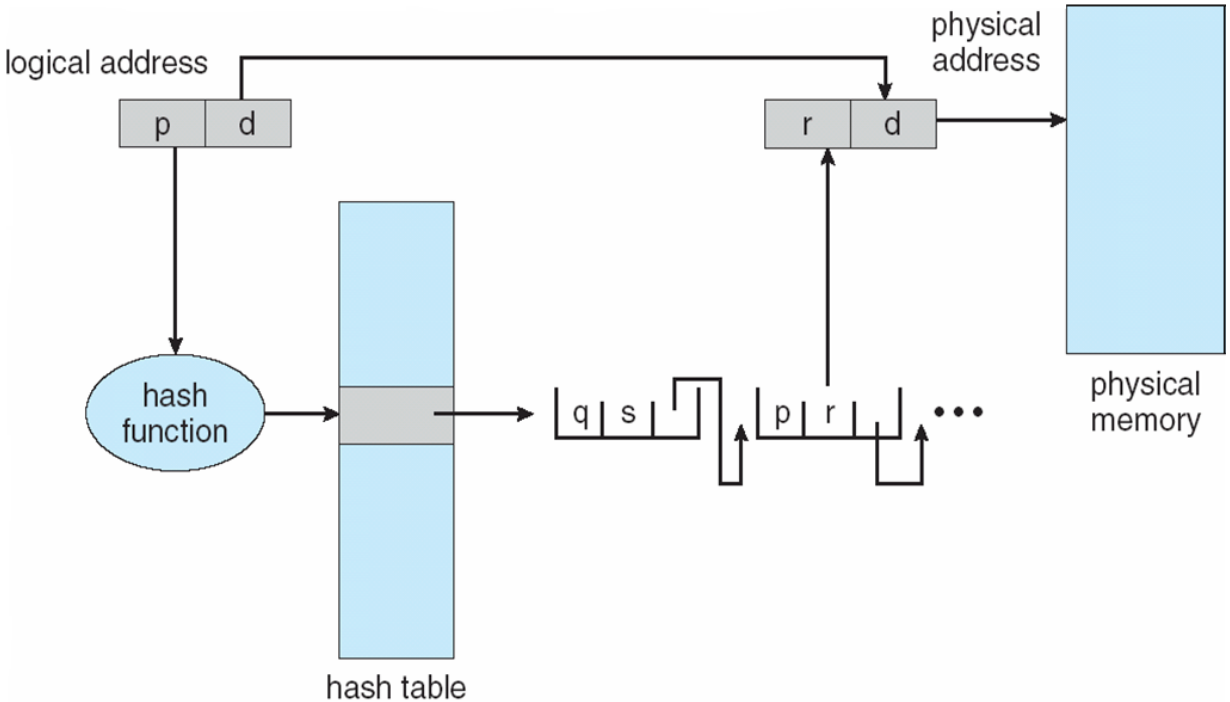


Principe de hashage

- une fonction surjective qui associe à chaque clé, un ou plusieurs éléments
- au lieu de stocker tous les éléments dans un tableau :
  - une entrée par clé
  - une entrée est une liste de tous les éléments ayant la méme clé

Pagination - Table des pages Hashée

Motivation : réduire la complexité dans les systèmes $>$ 32 bits



Principe de hashage

- une fonction surjective qui associe à chaque clé, un ou plusieurs éléments
- au lieu de stocker tous les éléments dans un tableau :
  - une entrée par clé
  - une entrée est une liste de tous les éléments ayant la méme clé
- intérêt : recherche plus rapide en utilisant la clé

Pagination : la protection (1/2)

- Un bit de protection est associé à chaque page mémoire (lecture seule ou lecture/ecriture)
- Un bit (valid/invalid) est associé à chaque page pour indiquer si elle est dans l'espace d'adresse logique du processus

Exemple

- espace d'adresse à 14 bits (0-16383)

Pagination : la protection (1/2)

- Un bit de protection est associé à chaque page mémoire (lecture seule ou lecture/ecriture)
- Un bit (valid/invalid) est associé à chaque page pour indiquer si elle est dans l'espace d'adresse logique du processus

Exemple

- espace d'adresse à 14 bits (0-16383)
- taille d'une page est 2K = 2^{11}

Pagination : la protection (1/2)

- Un bit de protection est associé à chaque page mémoire (lecture seule ou lecture/ecriture)
- Un bit (valid/invalid) est associé à chaque page pour indiquer si elle est dans l'espace d'adresse logique du processus

Exemple

- espace d'adresse à 14 bits (0-16383)
- taille d'une page est 2K = 2^{11}
- espace d'adresse = $2^{14}/2^{11} = 8$ pages

Pagination : la protection (1/2)

- Un bit de protection est associé à chaque page mémoire (lecture seule ou lecture/ecriture)
- Un bit (valid/invalid) est associé à chaque page pour indiquer si elle est dans l'espace d'adresse logique du processus

Exemple

- espace d'adresse à 14 bits (0-16383)
- taille d'une page est 2K = 2^{11}
- espace d'adresse = $2^{14}/2^{11} = 8$ pages
- un programme qui n'utilise que les adresses (0-10468)

Pagination : la protection (1/2)

- Un bit de protection est associé à chaque page mémoire (lecture seule ou lecture/ecriture)
- Un bit (valid/invalid) est associé à chaque page pour indiquer si elle est dans l'espace d'adresse logique du processus

Exemple

- espace d'adresse à 14 bits (0-16383)
- taille d'une page est 2K = 2^{11}
- espace d'adresse = $2^{14}/2^{11} = 8$ pages
- un programme qui n'utilise que les adresses (0-10468)
  - besoin de 6 pages

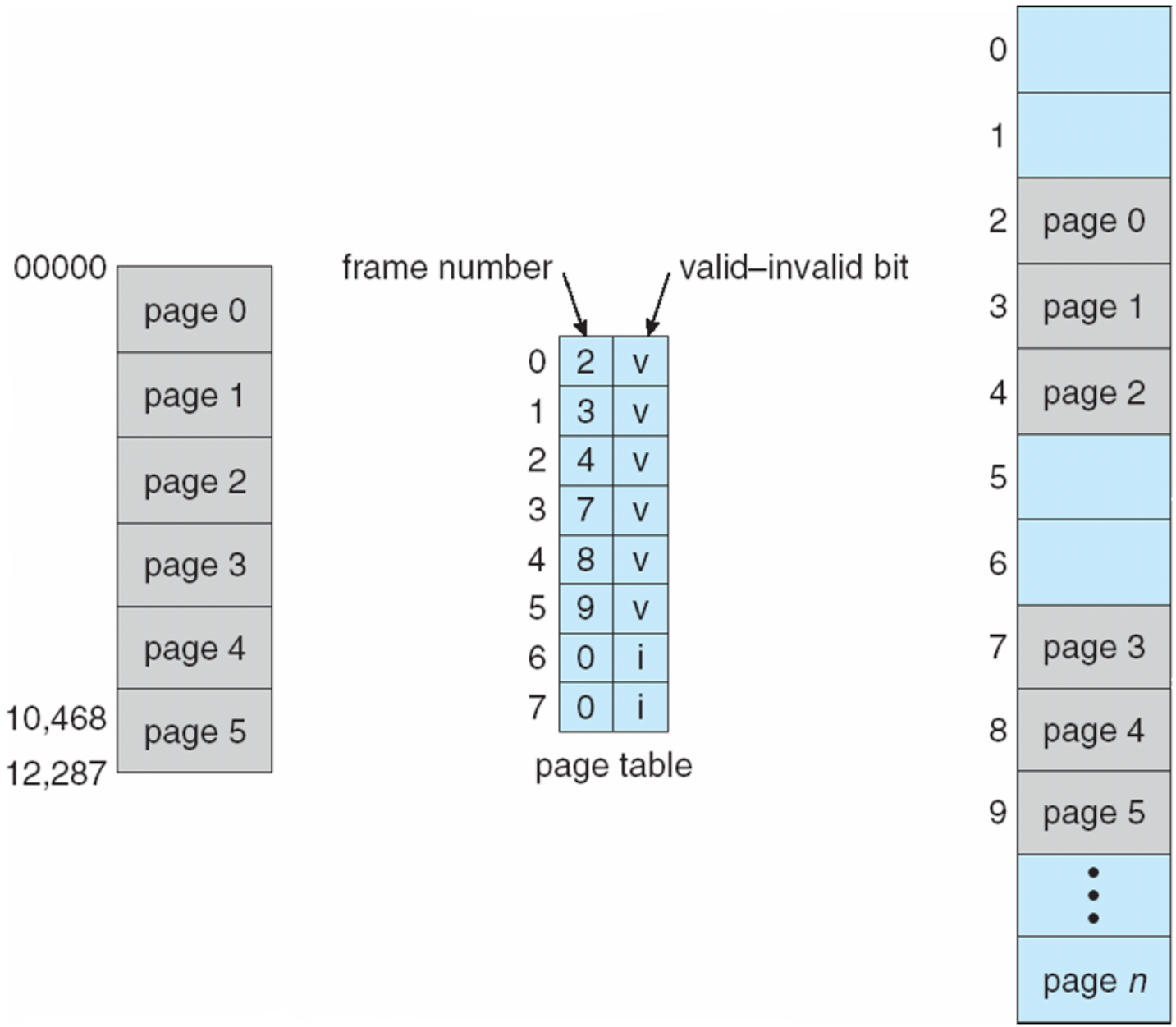
Pagination : la protection (1/2)

- Un bit de protection est associé à chaque page mémoire (lecture seule ou lecture/ecriture)
- Un bit (valid/invalid) est associé à chaque page pour indiquer si elle est dans l'espace d'adresse logique du processus

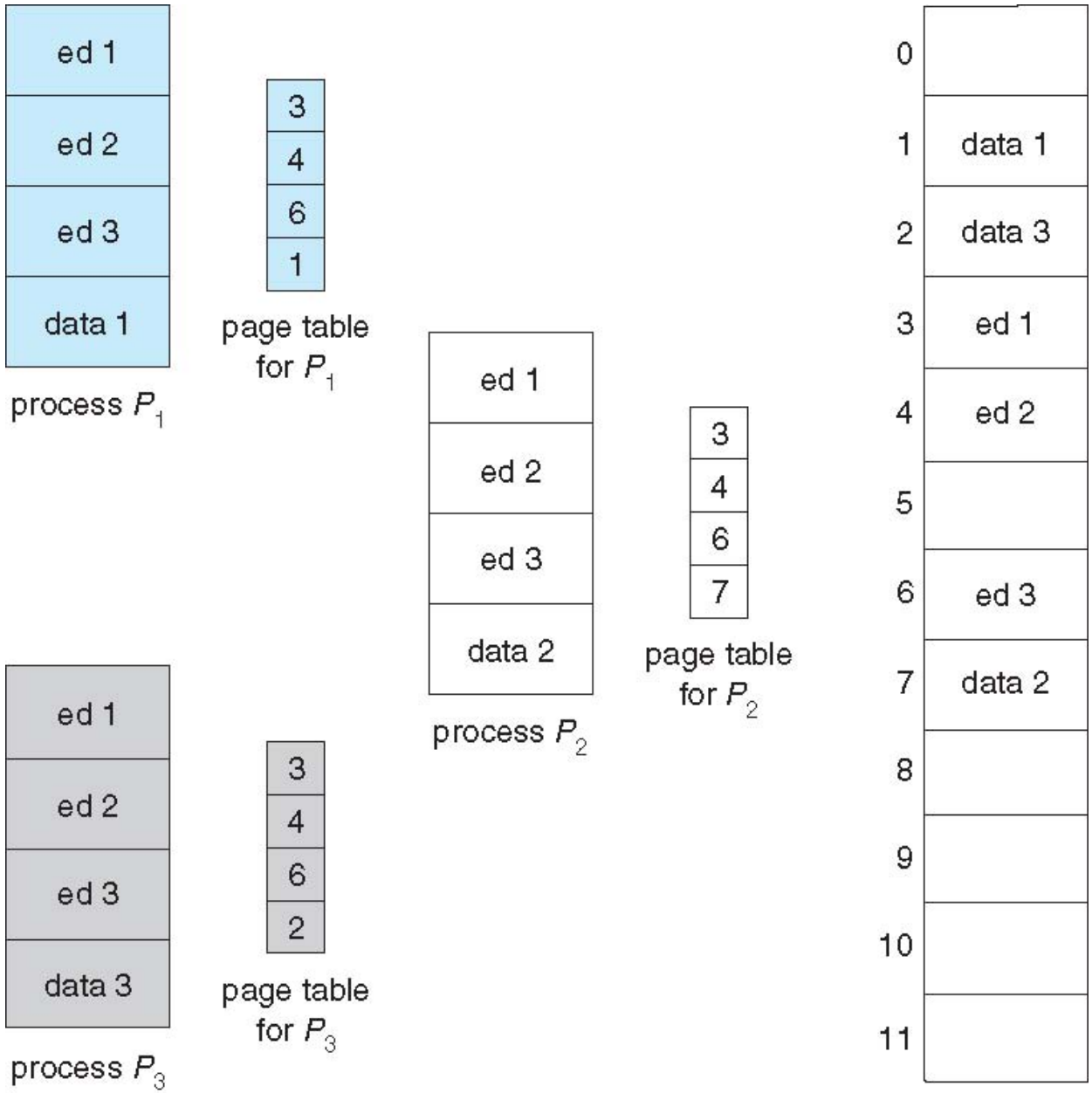
Exemple

- espace d'adresse à 14 bits (0-16383)
- taille d'une page est 2K = 2^{11}
- espace d'adresse = $2^{14}/2^{11} = 8$ pages
- un programme qui n'utilise que les adresses (0-10468)
  - besoin de 6 pages
- les pages 6 et 7 doivent être marquées comme étant invalides

Pagination : la protection (2/2)



Pagination : le partage



Pagination - avantages et inconvénients

Avantages

- pas de fragmentation externe

Inconvénients

Pagination - avantages et inconvénients

Avantages

- pas de fragmentation externe
- séparation (vue utilisateur, mémoire physique)

Inconvénients

Pagination - avantages et inconvénients

Avantages

- pas de fragmentation externe
- séparation (vue utilisateur, mémoire physique)
- protection entre processus implicite

Inconvénients

Pagination - avantages et inconvénients

Avantages

- pas de fragmentation externe
- séparation (vue utilisateur, mémoire physique)
- protection entre processus implicite
- partage des pages entre plusieurs utilisateurs

Inconvénients

Pagination - avantages et inconvénients

Avantages

- pas de fragmentation externe
- séparation (vue utilisateur, mémoire physique)
- protection entre processus implicite
- partage des pages entre plusieurs utilisateurs

Inconvénients

- augmente le temps de commutation de contexte

Pagination - avantages et inconvénients

Avantages

- pas de fragmentation externe
- séparation (vue utilisateur, mémoire physique)
- protection entre processus implicite
- partage des pages entre plusieurs utilisateurs

Inconvénients

- augmente le temps de commutation de contexte
- fragmentation interne (limitée)

Segmentation

- Allocation non contigüe, partitions de tailles différentes où chaque partition est spécialisée, ce qui permet une vue utilisateur de la mémoire

Segmentation

- Allocation non contigüe, partitions de tailles différentes où chaque partition est spécialisée, ce qui permet une vue utilisateur de la mémoire
- Un segment est associé à une portion du programme sémantiquement définie. Exemple :

Segmentation

- Allocation non contigüe, partitions de tailles différentes où chaque partition est spécialisée, ce qui permet une vue utilisateur de la mémoire
- Un segment est associé à une portion du programme sémantiquement définie. Exemple :
  - segment code

Segmentation

- Allocation non contigüe, partitions de tailles différentes où chaque partition est spécialisée, ce qui permet une vue utilisateur de la mémoire
- Un segment est associé à une portion du programme sémantiquement définie. Exemple :
  - segment code
  - segment données

Segmentation

- Allocation non contigüe, partitions de tailles différentes où chaque partition est spécialisée, ce qui permet une vue utilisateur de la mémoire
- Un segment est associé à une portion du programme sémantiquement définie. Exemple :
  - segment code
  - segment données
  - segment pile . . .

Segmentation

- Allocation non contigüe, partitions de tailles différentes où chaque partition est spécialisée, ce qui permet une vue utilisateur de la mémoire
- Un segment est associé à une portion du programme sémantiquement définie. Exemple :
  - segment code
  - segment données
  - segment pile . . .
- Protection :

Segmentation

- Allocation non contigüe, partitions de tailles différentes où chaque partition est spécialisée, ce qui permet une vue utilisateur de la mémoire
- Un segment est associé à une portion du programme sémantiquement définie. Exemple :
  - segment code
  - segment données
  - segment pile . . .
- Protection :
  - entre processus

Segmentation

- Allocation non contigüe, partitions de tailles différentes où chaque partition est spécialisée, ce qui permet une vue utilisateur de la mémoire
- Un segment est associé à une portion du programme sémantiquement définie. Exemple :
  - segment code
  - segment données
  - segment pile ...
- Protection :
  - entre processus
  - lecture, écriture, exécution

Segmentation

- Allocation non contigüe, partitions de tailles différentes où chaque partition est spécialisée, ce qui permet une vue utilisateur de la mémoire
- Un segment est associé à une portion du programme sémantiquement définie. Exemple :
  - segment code
  - segment données
  - segment pile ...
- Protection :
  - entre processus
  - lecture, écriture, exécution
- Partage : possibilité d'utilisation du même segment de code par plusieurs processus différents

Segmentation

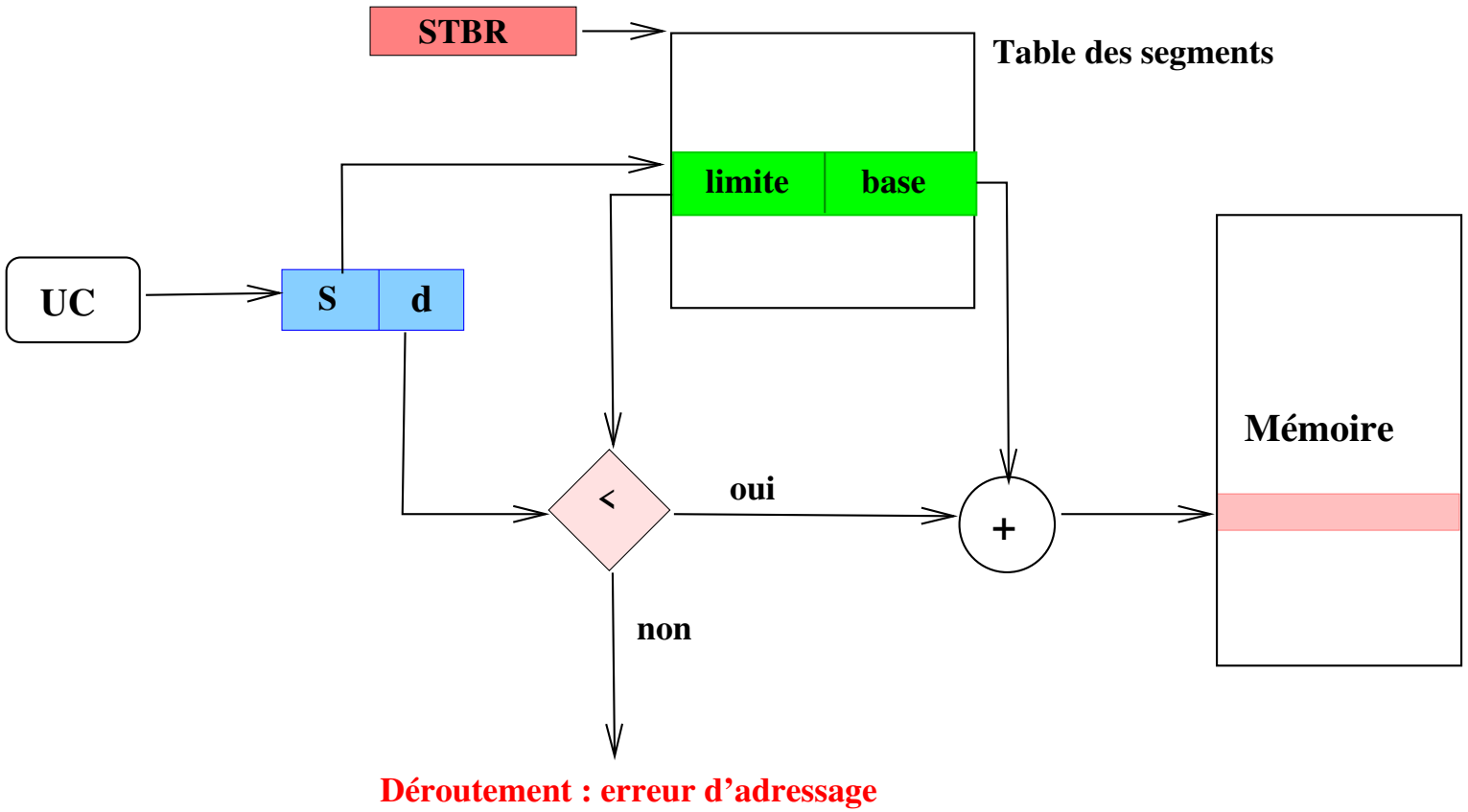
- Allocation non contigüe, partitions de tailles différentes où chaque partition est spécialisée, ce qui permet une vue utilisateur de la mémoire
- Un segment est associé à une portion du programme sémantiquement définie. Exemple :
  - segment code
  - segment données
  - segment pile ...
- Protection :
  - entre processus
  - lecture, écriture, exécution
- Partage : possibilité d'utilisation du même segment de code par plusieurs processus différents
- Problème : fragmentation externe quand les trous mémoire sont trop petits pour ranger un segment

Segmentation

- Allocation non contigüe, partitions de tailles différentes où chaque partition est spécialisée, ce qui permet une vue utilisateur de la mémoire
- Un segment est associé à une portion du programme sémantiquement définie. Exemple :
  - segment code
  - segment données
  - segment pile . . .
- Protection :
  - entre processus
  - lecture, écriture, exécution
- Partage : possibilité d'utilisation du même segment de code par plusieurs processus différents
- Problème : fragmentation externe quand les trous mémoire sont trop petits pour ranger un segment
- Solution : la segmentation paginée

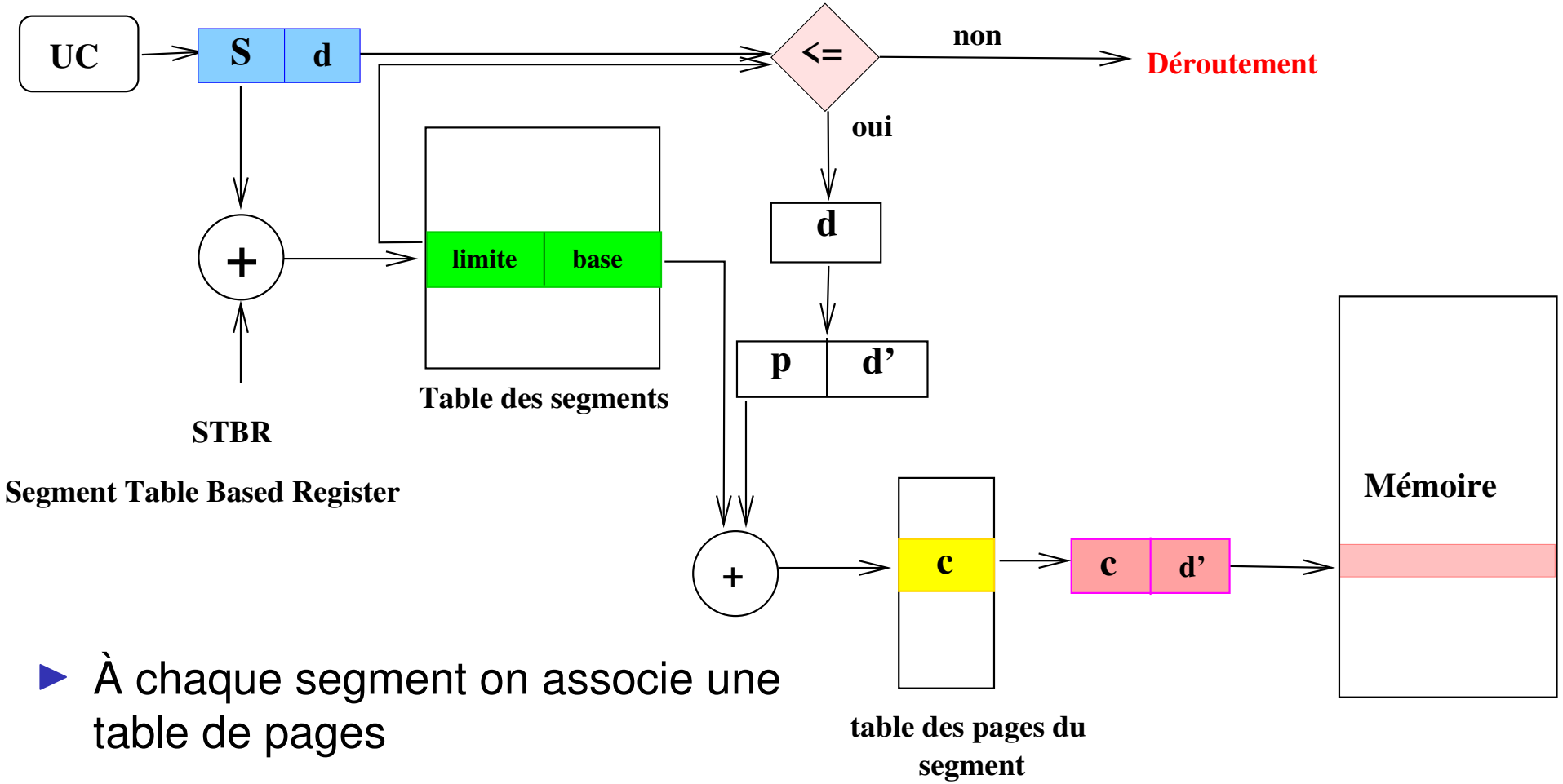
Segmentation - Conversion d'adresses

- Une adresse virtuelle est de la forme <num_segment, déplacement>
- Une table des segments est nécessaire. Elle peut être stockée en mémoire ou dans des registres
- STBR (segment-table base register) : adresse de début de la table des segments



Segmentation paginée

- un segment est composé de plusieurs pages
- permet de réduire la fragmentation externe



- À chaque segment on associe une table de pages

Deuxième partie

Gestion de la mémoire

- Introduction et concepts
- Mémoire uniforme
  - Partition unique / Partitions multiples
  - Pagination
  - Segmentation
  - Segmentation paginée
- Mémoire hiérarchisée (virtuelle)
  - Swapping
  - Pagination à la demande
  - Algorithmes de remplacement de pages
- Etude de cas
  - Le Pentium
  - Linux

Mémoire virtuelle (hiérarchisée)

- une technique qui permet l'exécution de programmes pouvant ne pas être complètement en mémoire

Différentes implémentations

Mémoire virtuelle (hiérarchisée)

- une technique qui permet l'exécution de programmes pouvant ne pas être complètement en mémoire
- la mémoire est vue comme un grand tableau de stockage uniforme : séparation de la mémoire logique (vision utilisateur) de la mémoire physique

Différentes implémentations

Mémoire virtuelle (hiérarchisée)

- une technique qui permet l'exécution de programmes pouvant ne pas être complètement en mémoire
- la mémoire est vue comme un grand tableau de stockage uniforme : séparation de la mémoire logique (vision utilisateur) de la mémoire physique
- permet une meilleure utilisation de la mémoire : une routine n'est chargée en mémoire qu'au besoin.

Différentes implémentations

Mémoire virtuelle (hiérarchisée)

- une technique qui permet l'exécution de programmes pouvant ne pas être complètement en mémoire
- la mémoire est vue comme un grand tableau de stockage uniforme : séparation de la mémoire logique (vision utilisateur) de la mémoire physique
- permet une meilleure utilisation de la mémoire : une routine n'est chargée en mémoire qu'au besoin.

Différentes implémentations

- le swapping

Mémoire virtuelle (hiérarchisée)

- une technique qui permet l'exécution de programmes pouvant ne pas être complètement en mémoire
- la mémoire est vue comme un grand tableau de stockage uniforme : séparation de la mémoire logique (vision utilisateur) de la mémoire physique
- permet une meilleure utilisation de la mémoire : une routine n'est chargée en mémoire qu'au besoin.

Différentes implémentations

- le swapping
- le swapping à la demande :

Mémoire virtuelle (hiérarchisée)

- une technique qui permet l'exécution de programmes pouvant ne pas être complètement en mémoire
- la mémoire est vue comme un grand tableau de stockage uniforme : séparation de la mémoire logique (vision utilisateur) de la mémoire physique
- permet une meilleure utilisation de la mémoire : une routine n'est chargée en mémoire qu'au besoin.

Différentes implémentations

- le swapping
- le swapping à la demande :
  - la pagination à la demande (la plus commune)

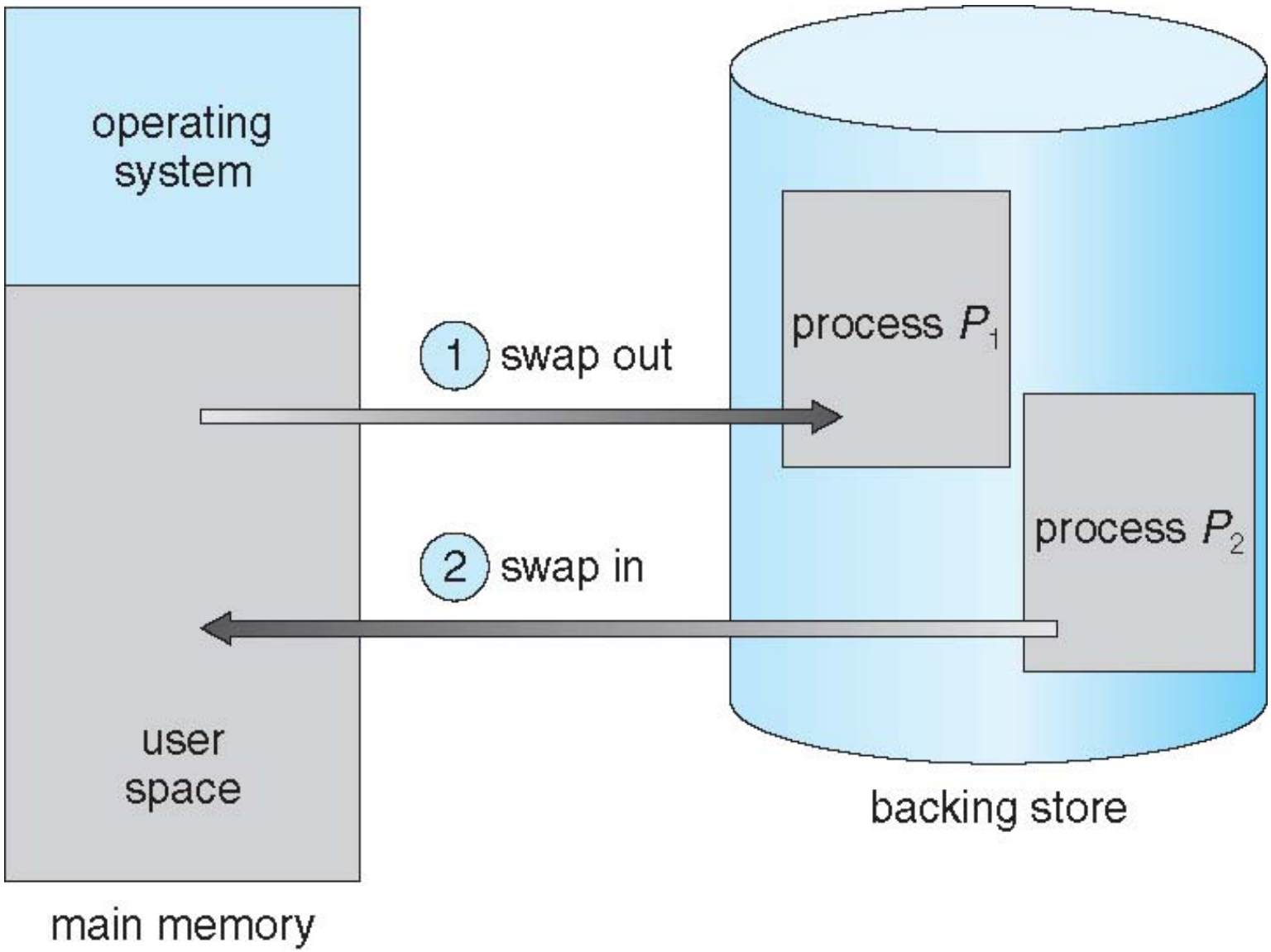
Mémoire virtuelle (hiérarchisée)

- une technique qui permet l'exécution de programmes pouvant ne pas être complètement en mémoire
- la mémoire est vue comme un grand tableau de stockage uniforme : séparation de la mémoire logique (vision utilisateur) de la mémoire physique
- permet une meilleure utilisation de la mémoire : une routine n'est chargée en mémoire qu'au besoin.

Différentes implémentations

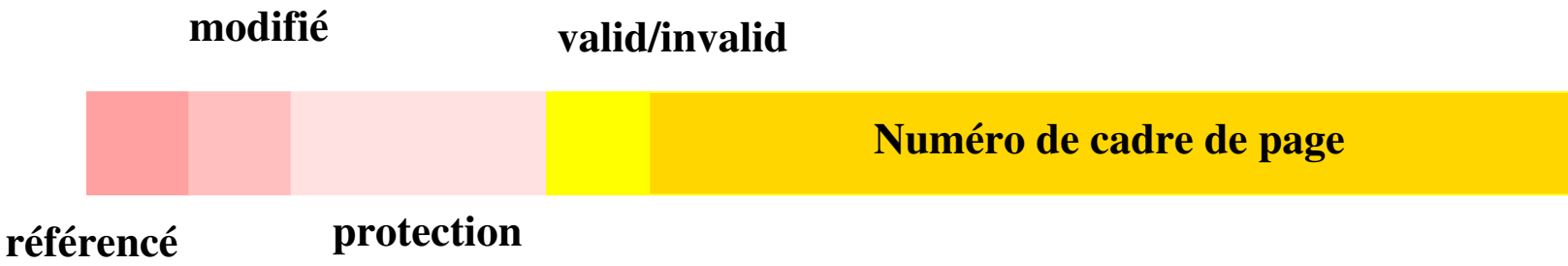
- le swapping
- le swapping à la demande :
  - la pagination à la demande (la plus commune)
  - la segmentation à la demande (difficile à cause de la taille variable des segments).

Swapping


operating system
user space
main memory
1 swap out
2 swap in
process P_1
process P_2
backing store

Pagination à la demande

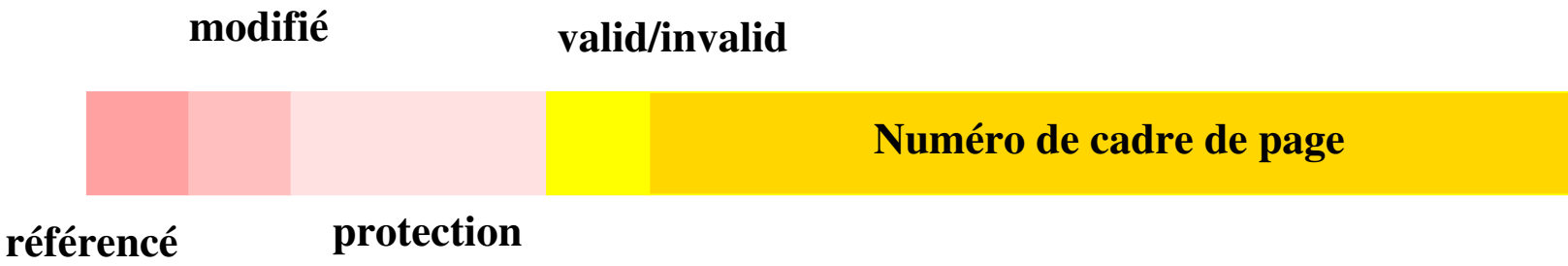
Structure possible d'une entrée de la table des page



- numéro de cadre de page

Pagination à la demande

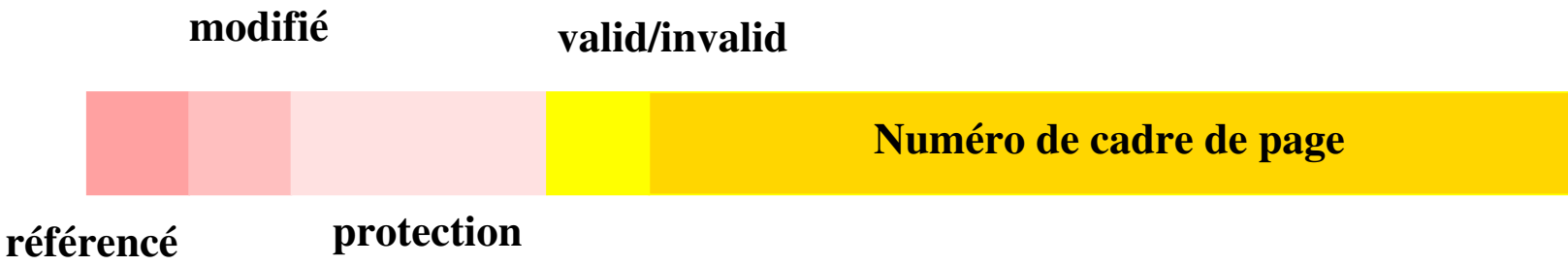
Structure possible d'une entrée de la table des page



- numéro de cadre de page
- bit de présence (valid/invalid)

Pagination à la demande

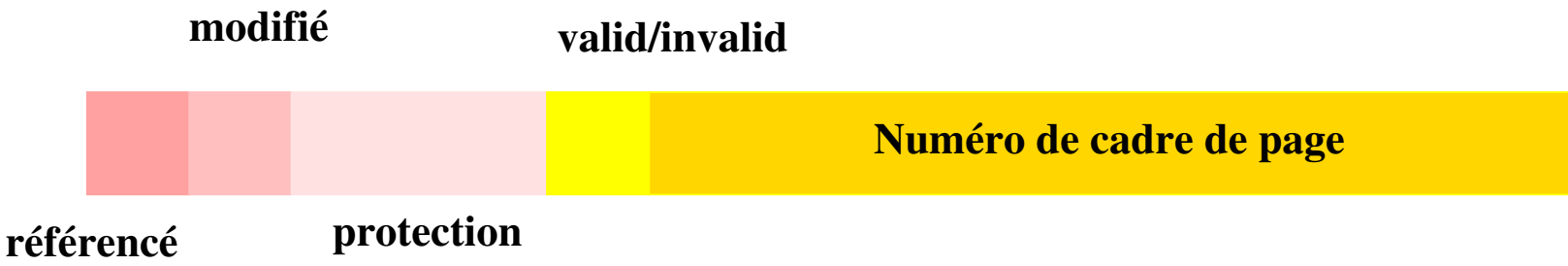
Structure possible d'une entrée de la table des page



- numéro de cadre de page
- bit de présence (valid/invalid)
- bit (3 bits) de protection

Pagination à la demande

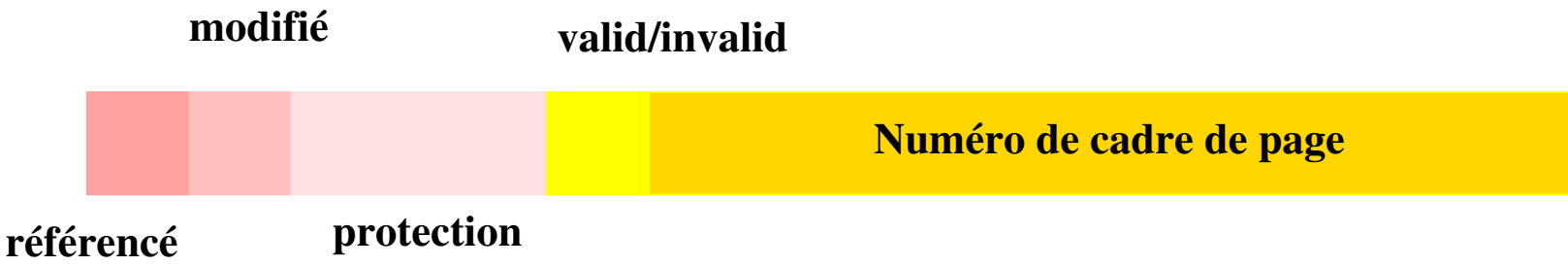
Structure possible d'une entrée de la table des page



- numéro de cadre de page
- bit de présence (valid/invalid)
- bit (3 bits) de protection
- bit modifié (dirty bit) : si positionné à 1, la page doit être écrite sur le disque (accédée en écriture par un processus)

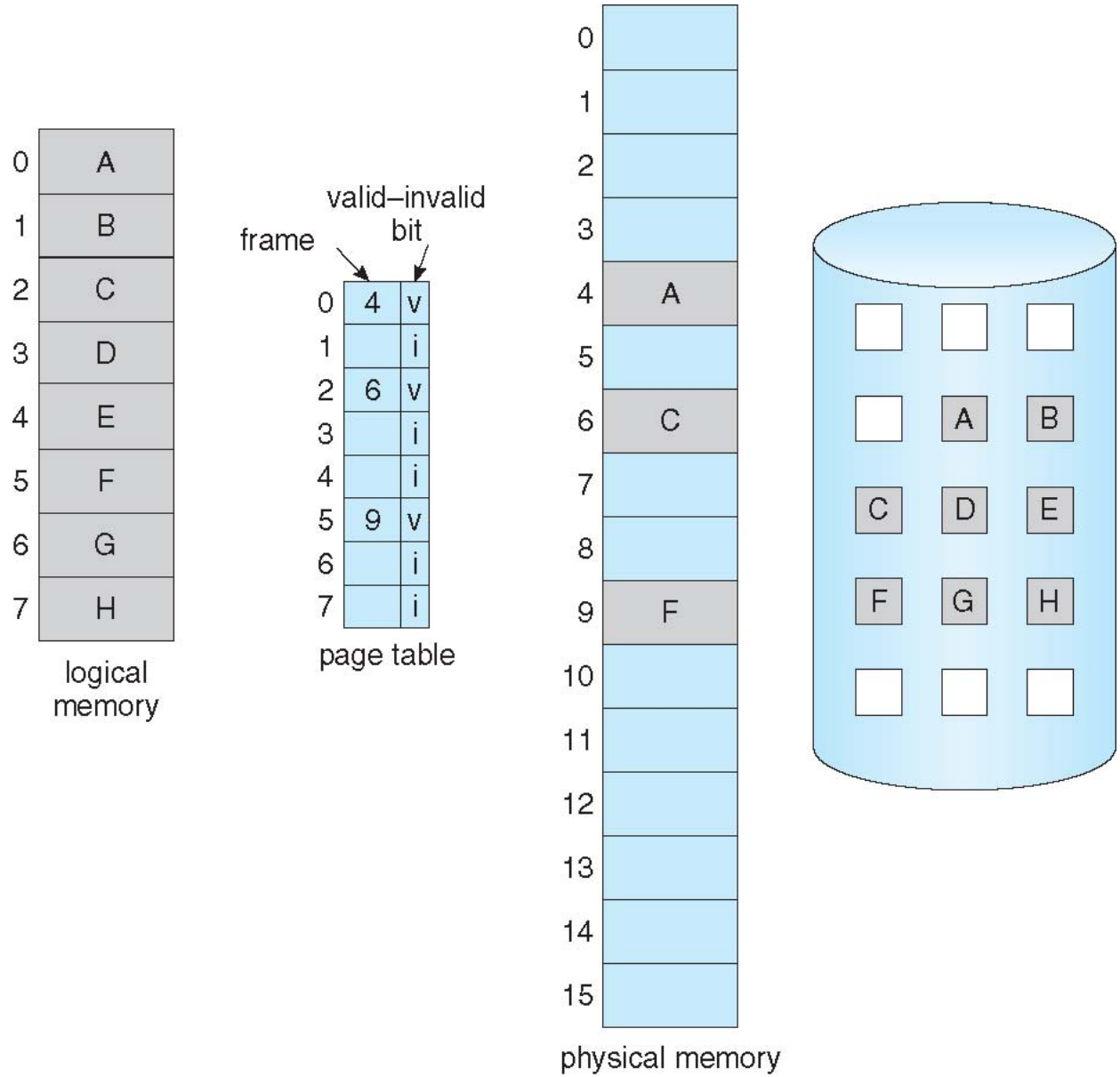
Pagination à la demande

Structure possible d'une entrée de la table des page



- numéro de cadre de page
- bit de présence (valid/invalid)
- bit (3 bits) de protection
- bit modifié (dirty bit) : si positionné à 1, la page doit être écrite sur le disque (accédée en écriture par un processus)
- bit référencé : mis à 1 à chaque fois que la page est accédée en lecture ou en écriture $\Rightarrow$ ça sert surtout pour les algorithmes de remplacement de pages

Pagination à la demande



Pagination à la demande

- Une page est transférée de la mémoire secondaire à la mémoire centrale (MC) lorsqu'on a besoin de cette page.

```
loop
   ins(@,opérandes)
   if ins en mémoire then
      exécuter (ins)
      if ins modifie page then
         TP(page).bit_modif ← 1
      end if
   else
      déroutement(défaut de
      page)
   end if
end loop
```

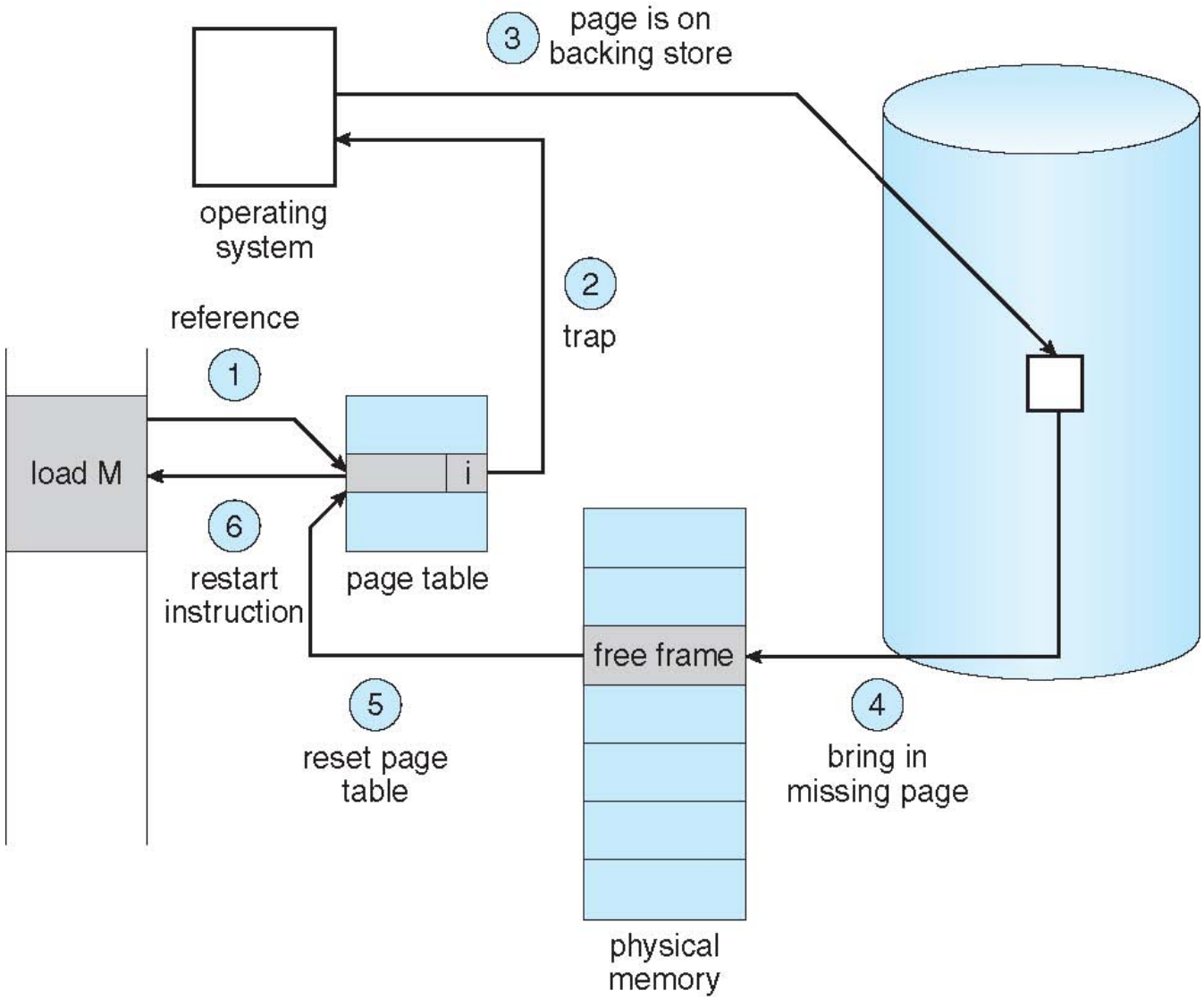
défaut_de_page(*page*)

```
if case libre then
   case ← page
else
   choix (case)
   libérer (case) et y charger (page)
end if
TP(page).case ← case
TP(page).bit_presence ← 1
réexécuter (ins)
```

libérer(*case*) - *case* contient $page_c$

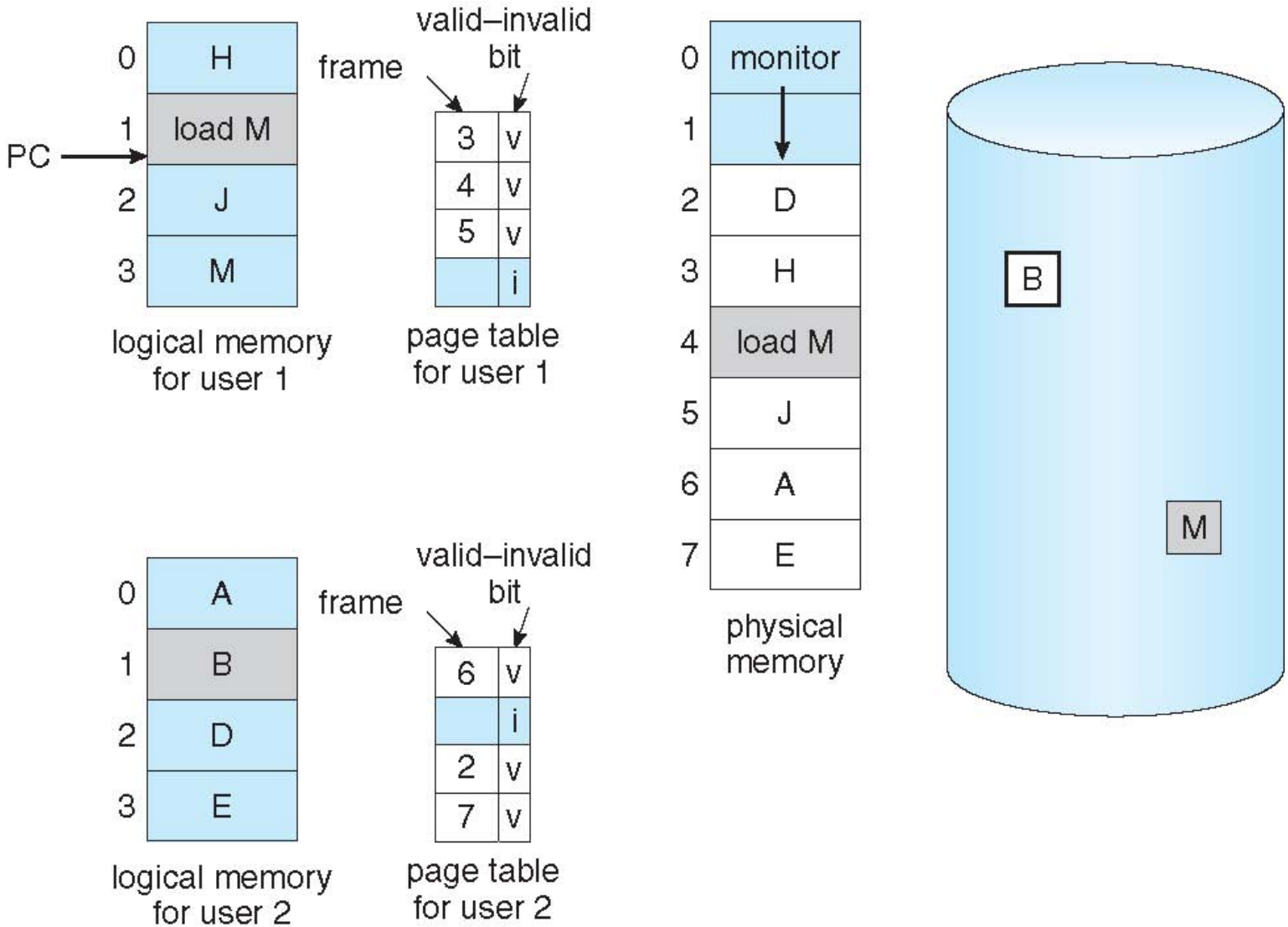
```
TP(page_c).bit_presence ← 0
if TP(page_c).bit_modif = 1 then
   transférer page_c vers la mémoire
   auxiliaire
end if
```

Pagination à la demande - Défaut de page illustré



Algorithmes de remplacement

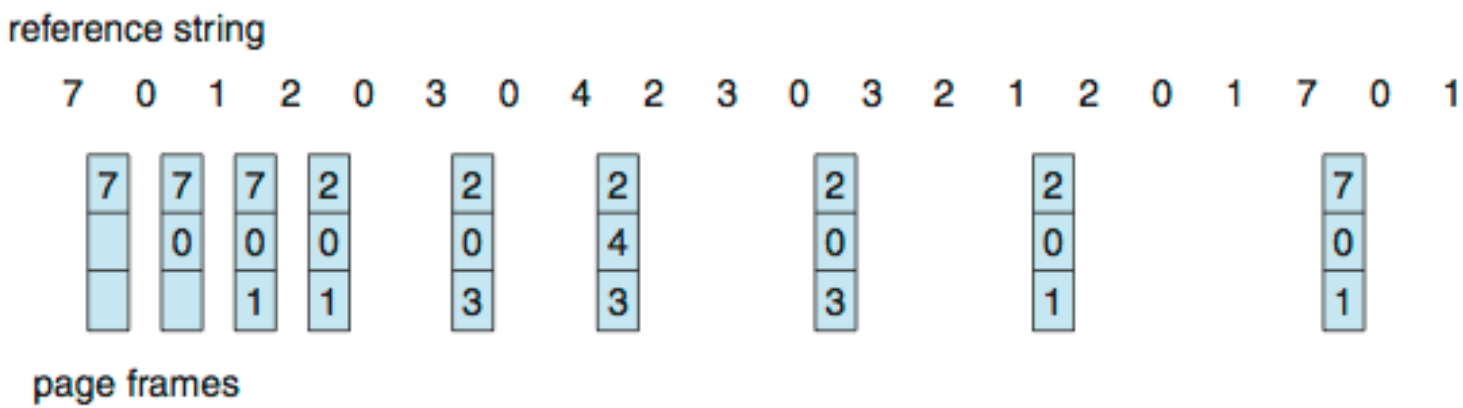
Besoin de remplacement de pages



Algorithmes de remplacement

L'algorithme optimal

- La page victime est celle qui mettra le plus de temps à être référencée.

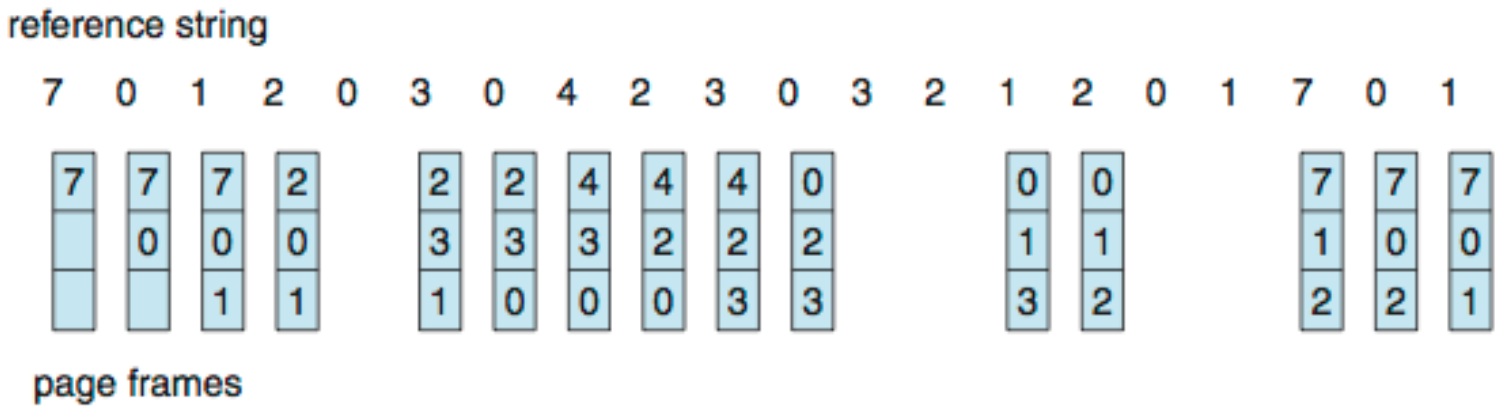


- Algorithme pratiquement impossible.

Algorithmes de remplacement

Premier arrivé, premier servi : FIFO

- la page victime est celle qui a été chargée la première



Implémentation

- file d'attente FIFO des pages (la victime en tête de file)

Algorithmes de remplacement

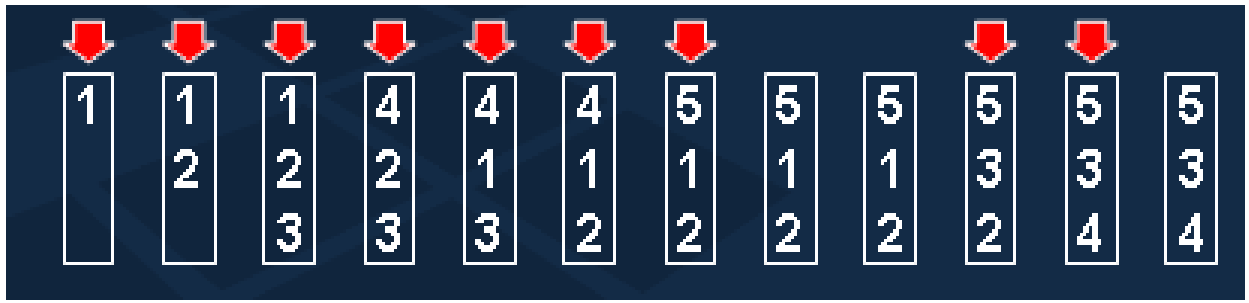
Anomalie de Belady

Soit la chaîne de référence : 1, 2, 3, 4, 1, 2, 5, 1, 2, 3, 4, 5.

Algorithmes de remplacement

Anomalie de Belady

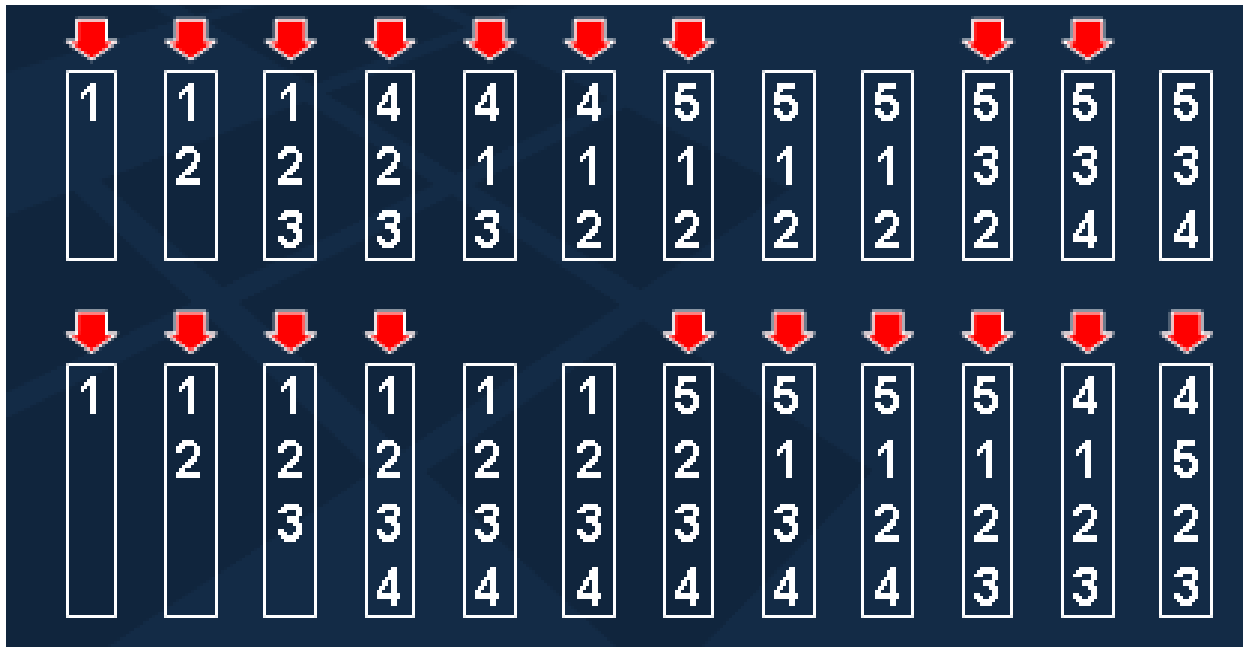
Soit la chaîne de référence : 1, 2, 3, 4, 1, 2, 5, 1, 2, 3, 4, 5.



Algorithmes de remplacement

Anomalie de Belady

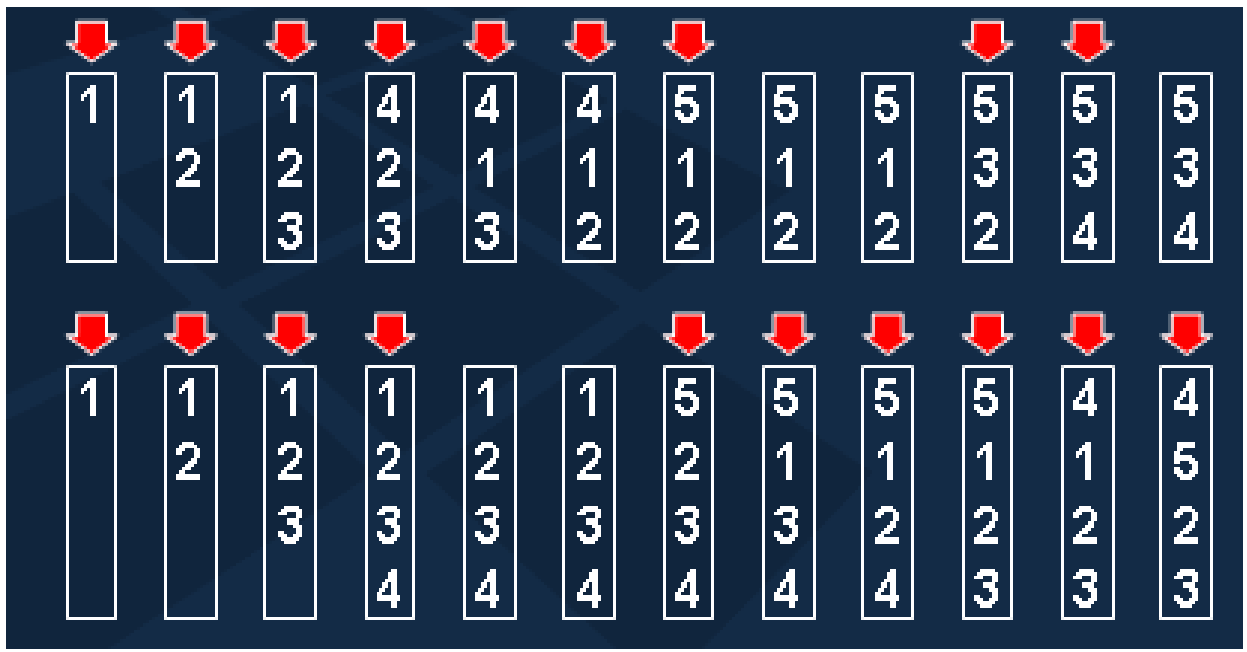
Soit la chaîne de référence : 1, 2, 3, 4, 1, 2, 5, 1, 2, 3, 4, 5.



Algorithmes de remplacement

Anomalie de Belady

Soit la chaîne de référence : 1, 2, 3, 4, 1, 2, 5, 1, 2, 3, 4, 5.

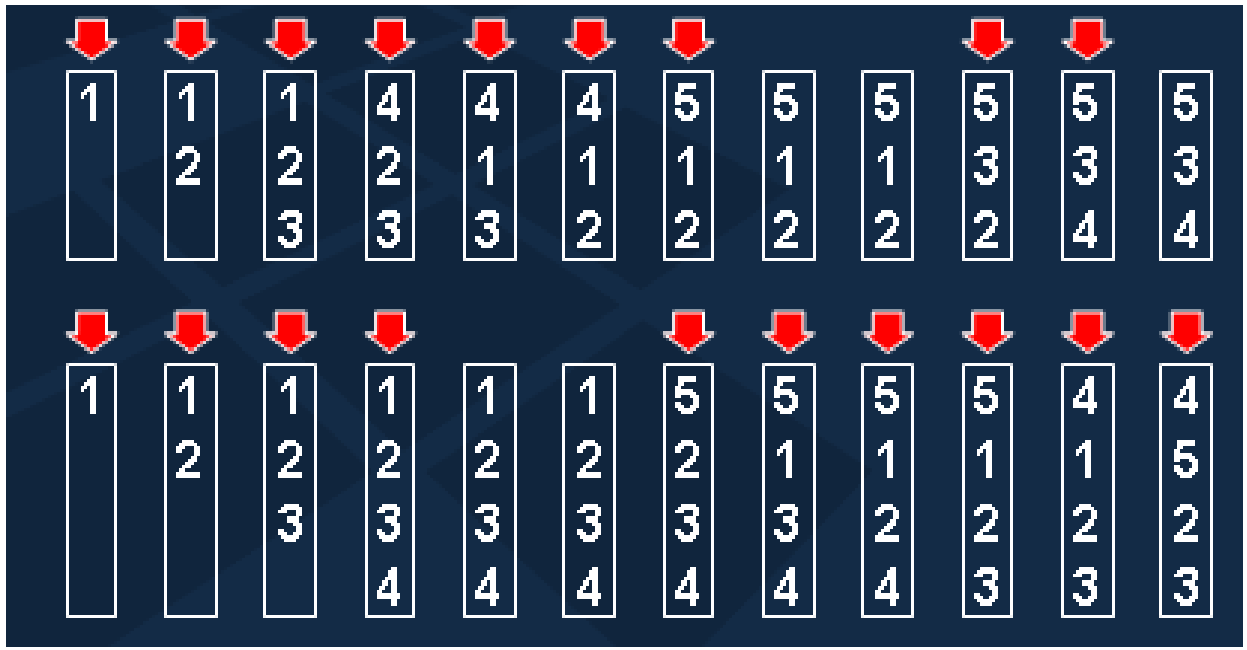


- 3 cadres de pages : 9 défauts de pages
- 4 cadres de pages : 10 défauts de pages

Algorithmes de remplacement

Anomalie de Belady

Soit la chaîne de référence : 1, 2, 3, 4, 1, 2, 5, 1, 2, 3, 4, 5.



- 3 cadres de pages : 9 défauts de pages
- 4 cadres de pages : 10 défauts de pages
- Avec plus de cadres, on peut avoir plus de défauts de pages !

Algorithmes de remplacement

Not Recently Used : NRU

- associer à chaque pages, 2 bits : R et M initialisés à 0

Algorithmes de remplacement

Not Recently Used : NRU

- associer à chaque pages, 2 bits : R et M initialisés à 0
- à chaque accès en lecture : $R \leftarrow 1$

Algorithmes de remplacement

Not Recently Used : NRU

- associer à chaque pages, 2 bits : R et M initialisés à 0
- à chaque accès en lecture : $R \leftarrow 1$
- à chaque accès en écriture : $M \leftarrow 1$

Algorithmes de remplacement

Not Recently Used : NRU

- associer à chaque pages, 2 bits : R et M initialisés à 0
- à chaque accès en lecture : $R \leftarrow 1$
- à chaque accès en écriture : $M \leftarrow 1$
- à chaque interruption d'horloge, le SE remet R à 0

Algorithmes de remplacement

Not Recently Used : NRU

- associer à chaque pages, 2 bits : R et M initialisés à 0
- à chaque accès en lecture : $R \leftarrow 1$
- à chaque accès en écriture : $M \leftarrow 1$
- à chaque interruption d'horloge, le SE remet R à 0
- Valeurs possibles (MR) :

Algorithmes de remplacement

Not Recently Used : NRU

- associer à chaque pages, 2 bits : R et M initialisés à 0
- à chaque accès en lecture : $R \leftarrow 1$
- à chaque accès en écriture : $M \leftarrow 1$
- à chaque interruption d'horloge, le SE remet R à 0
- Valeurs possibles (MR) :
  - 00 : non modifiée, non référencée

Algorithmes de remplacement

Not Recently Used : NRU

- associer à chaque pages, 2 bits : R et M initialisés à 0
- à chaque accès en lecture : $R \leftarrow 1$
- à chaque accès en écriture : $M \leftarrow 1$
- à chaque interruption d'horloge, le SE remet R à 0
- Valeurs possibles (MR) :
  - 00 : non modifiée, non référencée
  - 01 : non modifiée, référencée

Algorithmes de remplacement

Not Recently Used : NRU

- associer à chaque pages, 2 bits : R et M initialisés à 0
- à chaque accès en lecture : $R \leftarrow 1$
- à chaque accès en écriture : $M \leftarrow 1$
- à chaque interruption d'horloge, le SE remet R à 0
- Valeurs possibles (MR) :
  - 00 : non modifiée, non référencée
  - 01 : non modifiée, référencée
  - 10 : modifiée, non référencée

Algorithmes de remplacement

Not Recently Used : NRU

- associer à chaque pages, 2 bits : R et M initialisés à 0
- à chaque accès en lecture : $R \leftarrow 1$
- à chaque accès en écriture : $M \leftarrow 1$
- à chaque interruption d'horloge, le SE remet R à 0
- Valeurs possibles (MR) :
  - 00 : non modifiée, non référencée
  - 01 : non modifiée, référencée
  - 10 : modifiée, non référencée
  - 11 : modifiée, référencée

Algorithmes de remplacement

Not Recently Used : NRU

- associer à chaque pages, 2 bits : R et M initialisés à 0
- à chaque accès en lecture : $R \leftarrow 1$
- à chaque accès en écriture : $M \leftarrow 1$
- à chaque interruption d'horloge, le SE remet R à 0
- Valeurs possibles (MR) :
  - 00 : non modifiée, non référencée
  - 01 : non modifiée, référencée
  - 10 : modifiée, non référencée
  - 11 : modifiée, référencée
- La page victime est choisie au hasard parmi celles ayant le plus petit indice (MR)

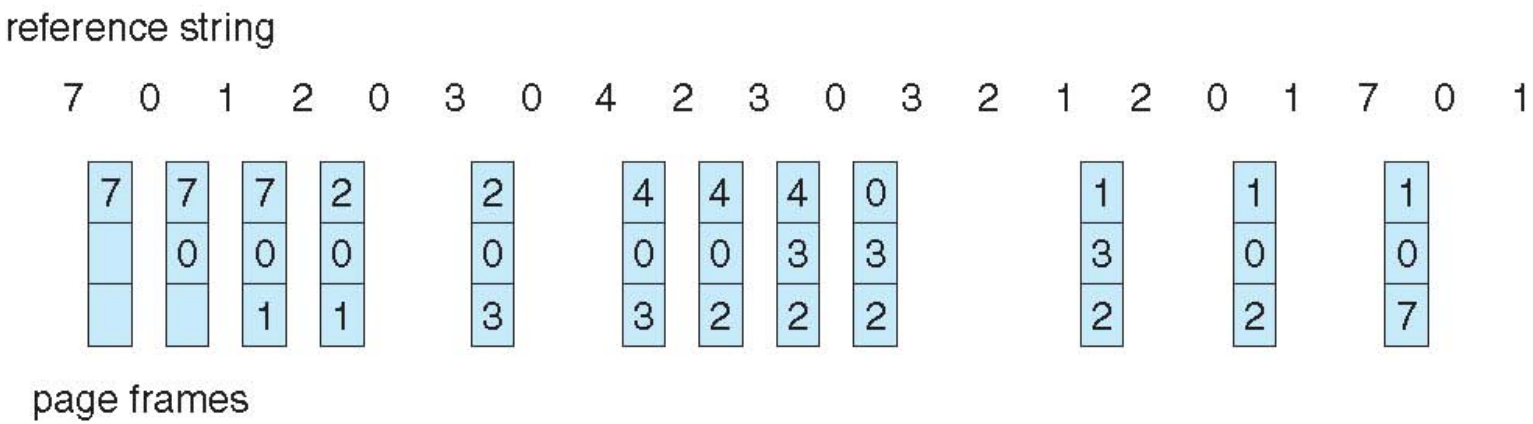
Algorithmes de remplacement

LRU : Least Recently Used

- la page victime est la moins récemment référencée
- **Exemples** :

1, 2, 3, 4, 1, 2, 5, 1, 2, 3, 4, 5

1	1	1	1	5
2	2	2	2	2
3	5	5	4	4
4	4	3	3	3



LRU : Implémentations

1. Associer à chaque page le moment de sa dernière utilisation.

Algorithmes de remplacement

LRU : Implémentations

1. Associer à chaque page le moment de sa dernière utilisation.
2. Utiliser une liste chaînée des descripteurs de pages :

LRU : Implémentations

1. Associer à chaque page le moment de sa dernière utilisation.
2. Utiliser une liste chaînée des descripteurs de pages :
   - à chaque référence à une page, la supprimer et la mettre en tête de liste

LRU : Implémentations

1. Associer à chaque page le moment de sa dernière utilisation.
2. Utiliser une liste chaînée des descripteurs de pages :
   - à chaque référence à une page, la supprimer et la mettre en tête de liste
   - la page victime est celle à la fin de la liste

Algorithmes de remplacement

LRU : Implémentations

1. Associer à chaque page le moment de sa dernière utilisation.
2. Utiliser une liste chaînée des descripteurs de pages :
   - à chaque référence à une page, la supprimer et la mettre en tête de liste
   - la page victime est celle à la fin de la liste

$\Rightarrow$ algorithme stable mais nécessite une gestion coûteuse de la liste modifiée à chaque accès à une page

Approximations de LRU

Bit de référence (R)

- R est initialement 0 et mis à 1 à chaque référence
- FIFO est appliqué d'abord sur les pages ayant le bit de référence à 0.

Algorithmes de remplacement

Approximations de LRU

Bit de référence (R)

- R est initialement 0 et mis à 1 à chaque référence
- FIFO est appliqué d'abord sur les pages ayant le bit de référence à 0.

La seconde chance

- Le bit de référence est positionné à 1 à chaque référence
- Liste circulaire avec un pointeur sur la prochaine victime

Algorithmes de remplacement

Approximations de LRU

Bit de référence (R)

- R est initialement 0 et mis à 1 à chaque référence
- FIFO est appliqué d'abord sur les pages ayant le bit de référence à 0.

La seconde chance

- Le bit de référence est positionné à 1 à chaque référence
- Liste circulaire avec un pointeur sur la prochaine victime
- Si la page à remplacer est telle que R = 0, on la remplace sinon on lui donne une 2ème chance :

Algorithmes de remplacement

Approximations de LRU

Bit de référence (R)

- R est initialement 0 et mis à 1 à chaque référence
- FIFO est appliqué d'abord sur les pages ayant le bit de référence à 0.

La seconde chance

- Le bit de référence est positionné à 1 à chaque référence
- Liste circulaire avec un pointeur sur la prochaine victime
- Si la page à remplacer est telle que R = 0, on la remplace sinon on lui donne une 2ème chance :
  - $R \leftarrow 0$

Algorithmes de remplacement

Approximations de LRU

Bit de référence (R)

- R est initialement 0 et mis à 1 à chaque référence
- FIFO est appliqué d'abord sur les pages ayant le bit de référence à 0.

La seconde chance

- Le bit de référence est positionné à 1 à chaque référence
- Liste circulaire avec un pointeur sur la prochaine victime
- Si la page à remplacer est telle que R = 0, on la remplace sinon on lui donne une 2ème chance :
  - $R \leftarrow 0$
  - laisser la page en mémoire

Algorithmes de remplacement

Approximations de LRU

Bit de référence (R)

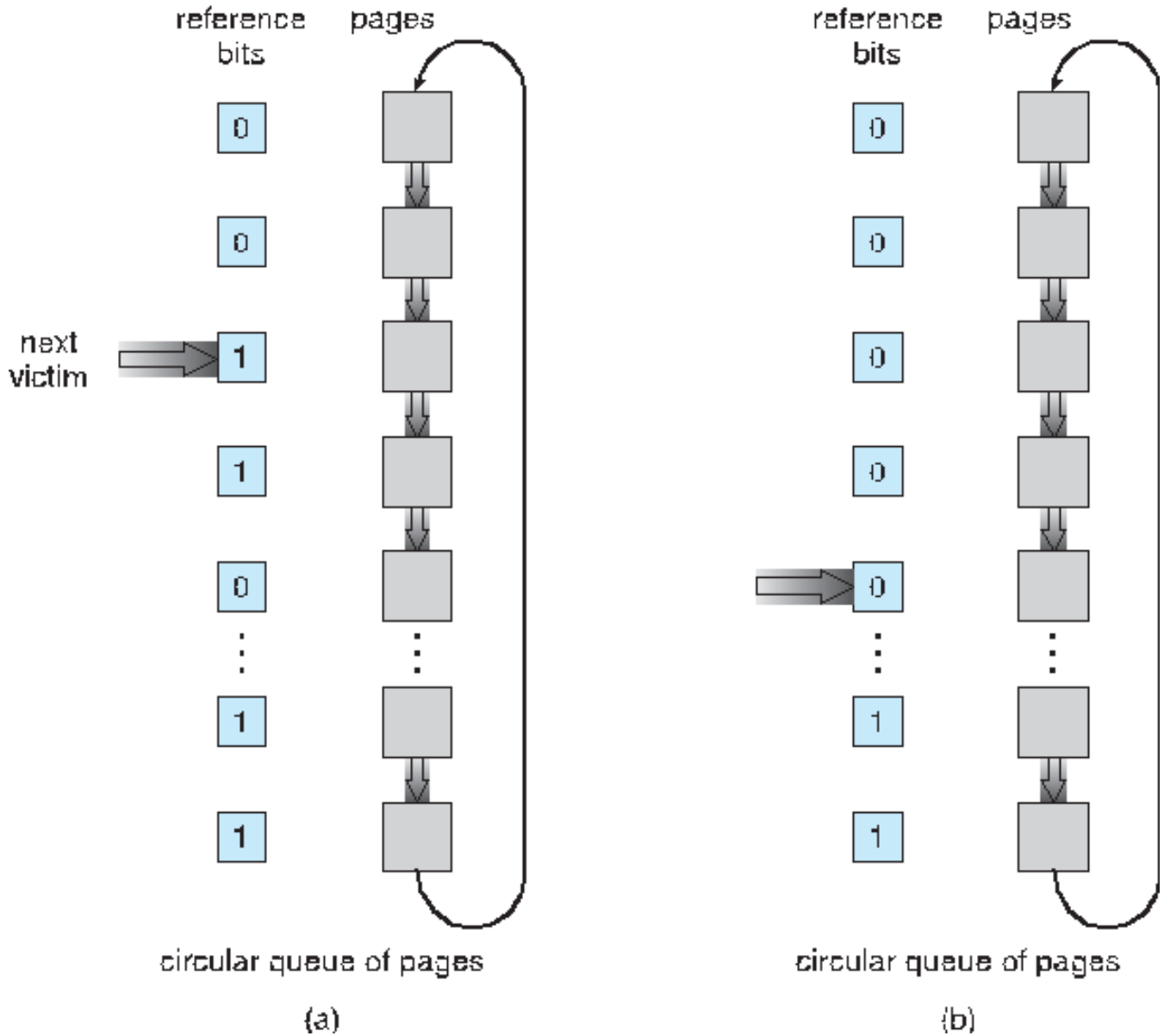
- R est initialement 0 et mis à 1 à chaque référence
- FIFO est appliqué d'abord sur les pages ayant le bit de référence à 0.

La seconde chance

- Le bit de référence est positionné à 1 à chaque référence
- Liste circulaire avec un pointeur sur la prochaine victime
- Si la page à remplacer est telle que R = 0, on la remplace sinon on lui donne une 2ème chance :
  - $R \leftarrow 0$
  - laisser la page en mémoire
  - passer à la suivante dans la liste en appliquant les mêmes règles.

Algorithmes de remplacement

La seconde chance



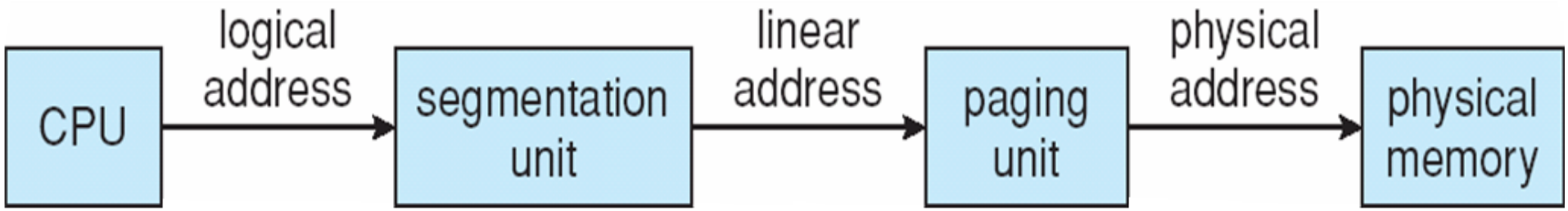
Etude de cas

Exemples d'architectures

- SPARC (32 bits) pagination à 3 niveaux
- Motorola 68030 (32bits) à 4 niveaux
- VAX segmentation paginée (2 niveaux) machine 32 bits
- Intel Pentium

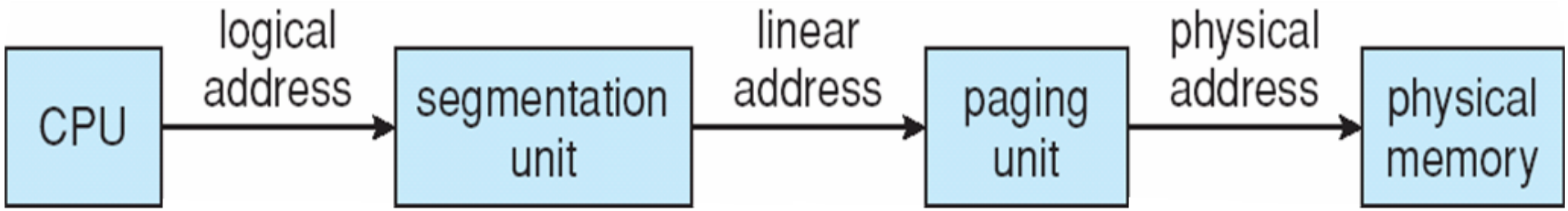
Etude de cas - Intel Pentium (IA-32)

- supporte à la fois la segmentation et la segmentation paginée



Etude de cas - Intel Pentium (IA-32)

- supporte à la fois la segmentation et la segmentation paginée

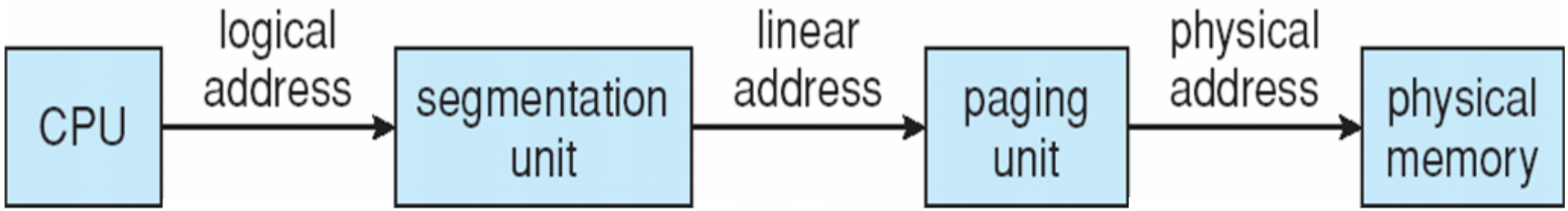


- Adresse logique : utilisée par le programme pour spécifier l'adresse d'un opérande ou d'une instruction.

  adresse logique = segment , offset

Etude de cas - Intel Pentium (IA-32)

- supporte à la fois la segmentation et la segmentation paginée



- Adresse logique : utilisée par le programme pour spécifier l'adresse d'un opérande ou d'une instruction.

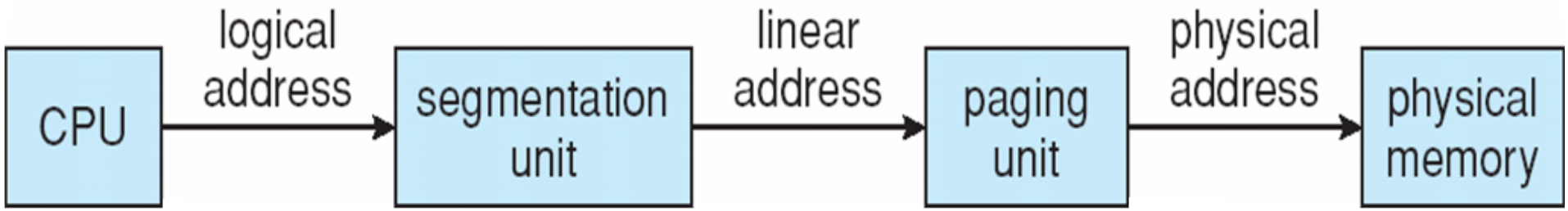
  adresse logique = segment , offset

- Adresse linéaire : un entier non signé sur 32 bits permettant l'adressage de 4GB.

  de 0x0000 0000 à 0xffff ffff

Etude de cas - Intel Pentium (IA-32)

- supporte à la fois la segmentation et la segmentation paginée



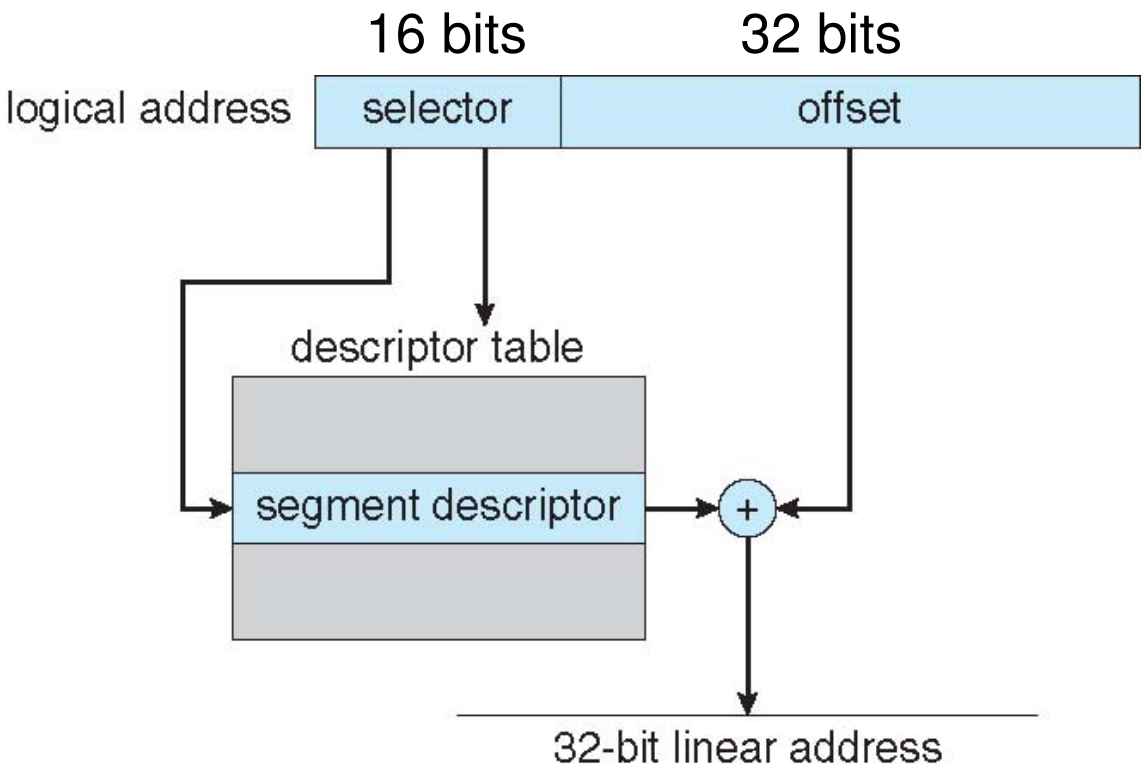
- Adresse logique : utilisée par le programme pour spécifier l'adresse d'un opérande ou d'une instruction.

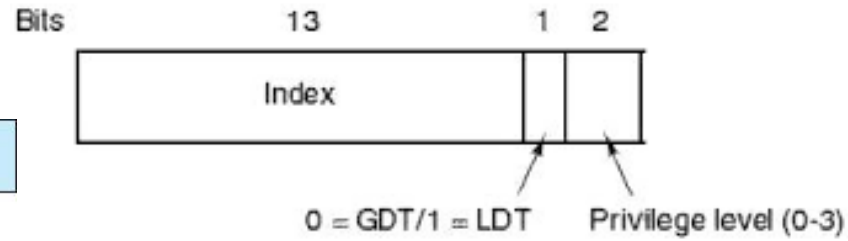
  adresse logique = segment , offset
- Adresse linéaire : un entier non signé sur 32 bits permettant l'adressage de 4GB.

  de 0x0000 0000 à 0xffff ffff
- Adresse physique : l'ensemble des signaux envoyés par le processeur à la mémoire sur le bus d'adresse permettant l'accès à une cellule mémoire.

Etude de cas - Intel Pentium (IA-32)

Unité de segmentation : adresse logique → adresse linéaire





- selector :
  - index : indice dans la table des segments
  - GDT (Global Descriptor Table) : segments système
  - LDT (Local Descriptor Table) : un par processus
  - RPL : niveau de privilège

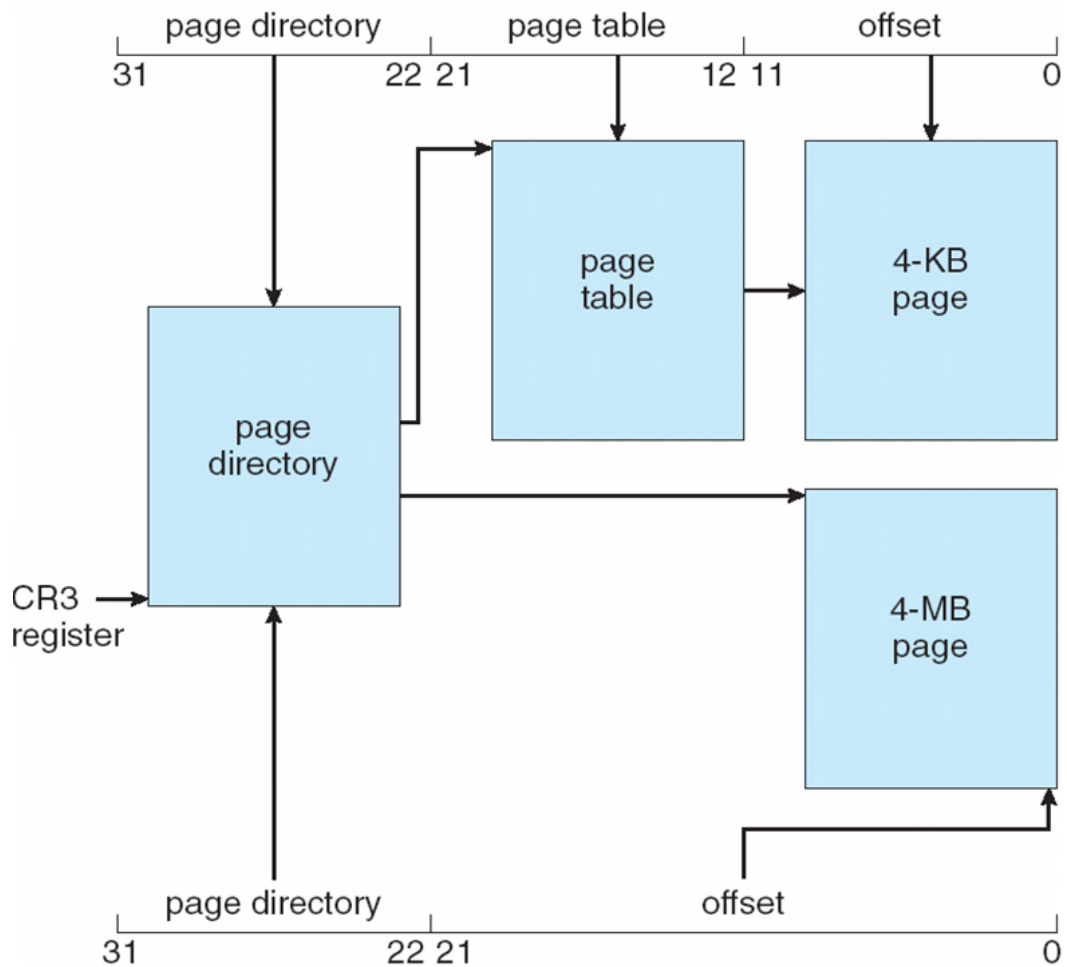
- Descripteur de segment (8 octets) :
  - base (32 bits) : @linéaire du 1er octet du segment
  - limit (20 bits) : longueur du segment en octets (G = 0) ou en pages (G = 1)

- Registres spécialisés :
  - GDTR : pointe sur la GDT
  - LDTR : pointe sur la LDT en cours

Etude de cas - Intel Pentium (IA-32)

Unité de pagination

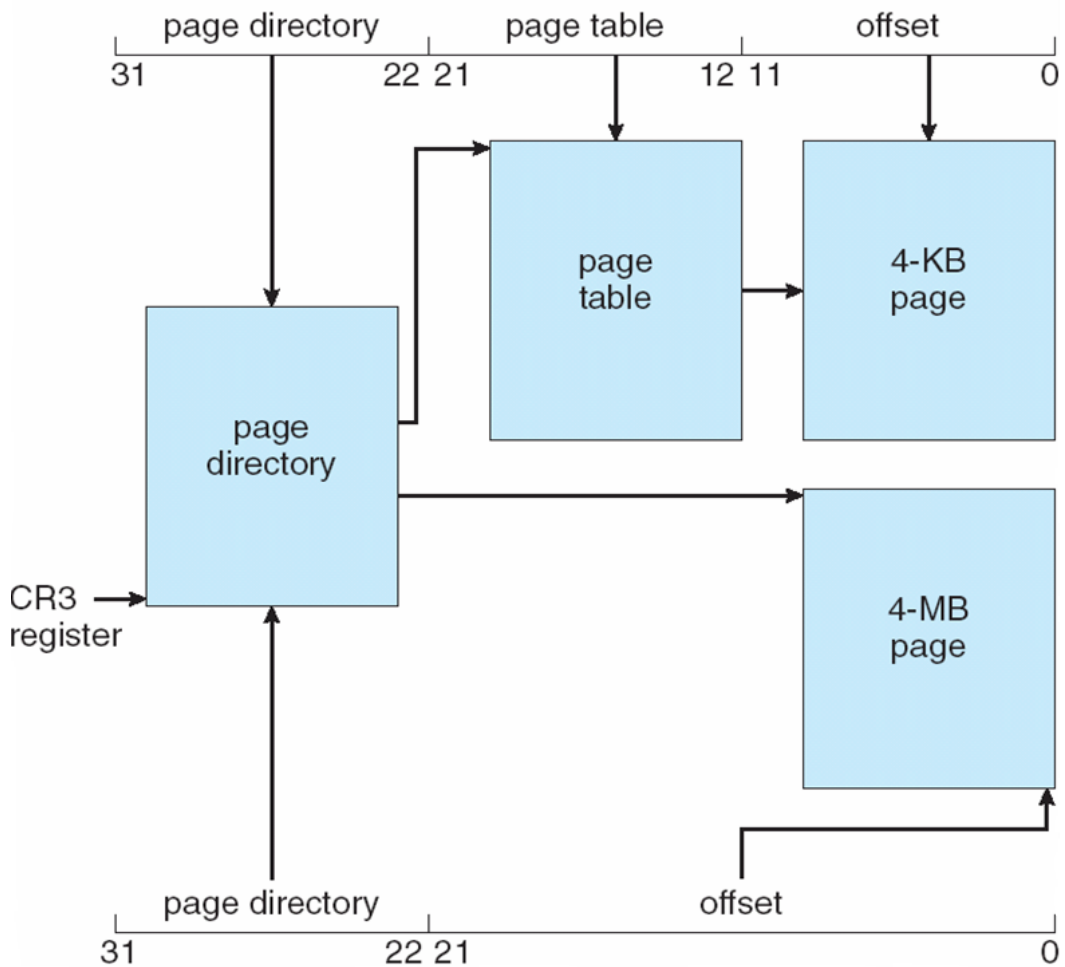
- la pagination est activée :
  flag PG du registre cr0



Etude de cas - Intel Pentium (IA-32)

Unité de pagination

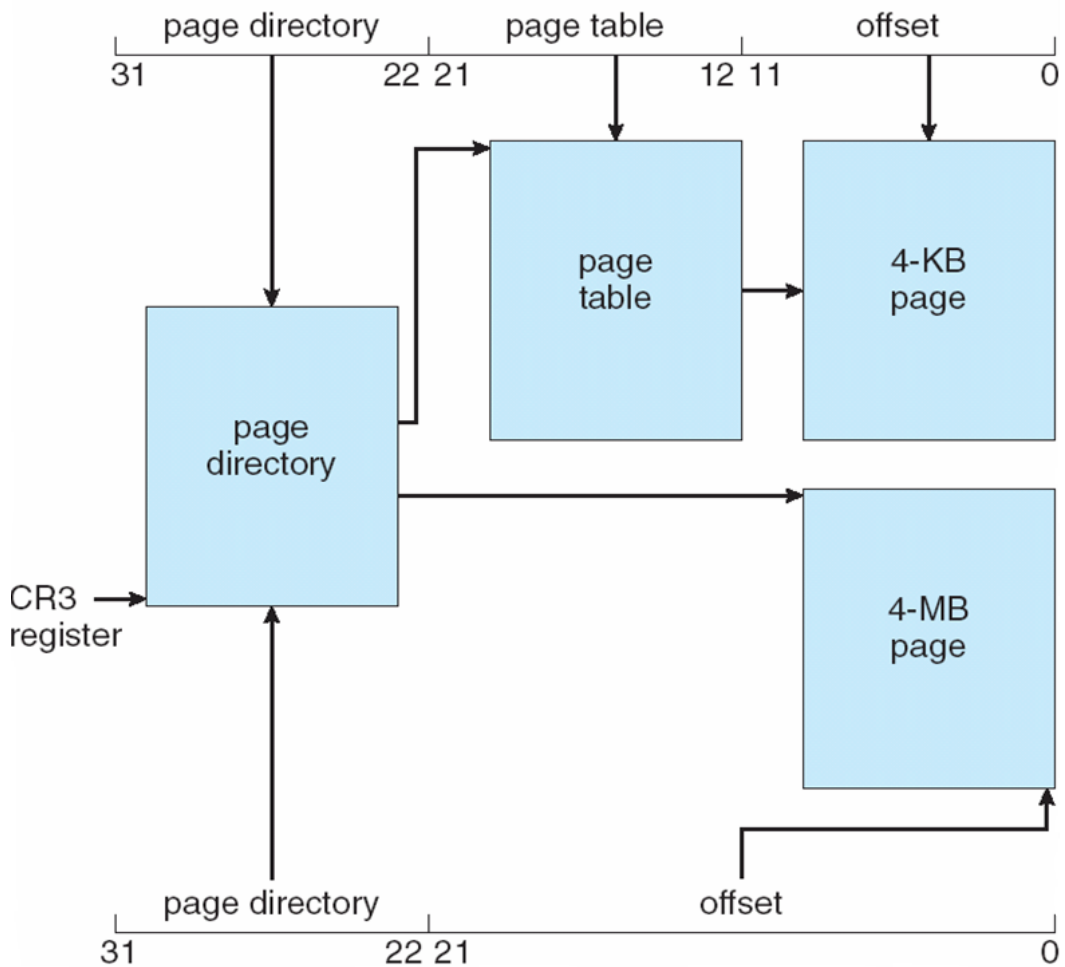
- la pagination est activée : flag PG du registre cr0
- taille de page : 4K



Etude de cas - Intel Pentium (IA-32)

Unité de pagination

- la pagination est activée : flag PG du registre cr0
- taille de page : 4K
- cr3 contient l'@physique de la "page directory"

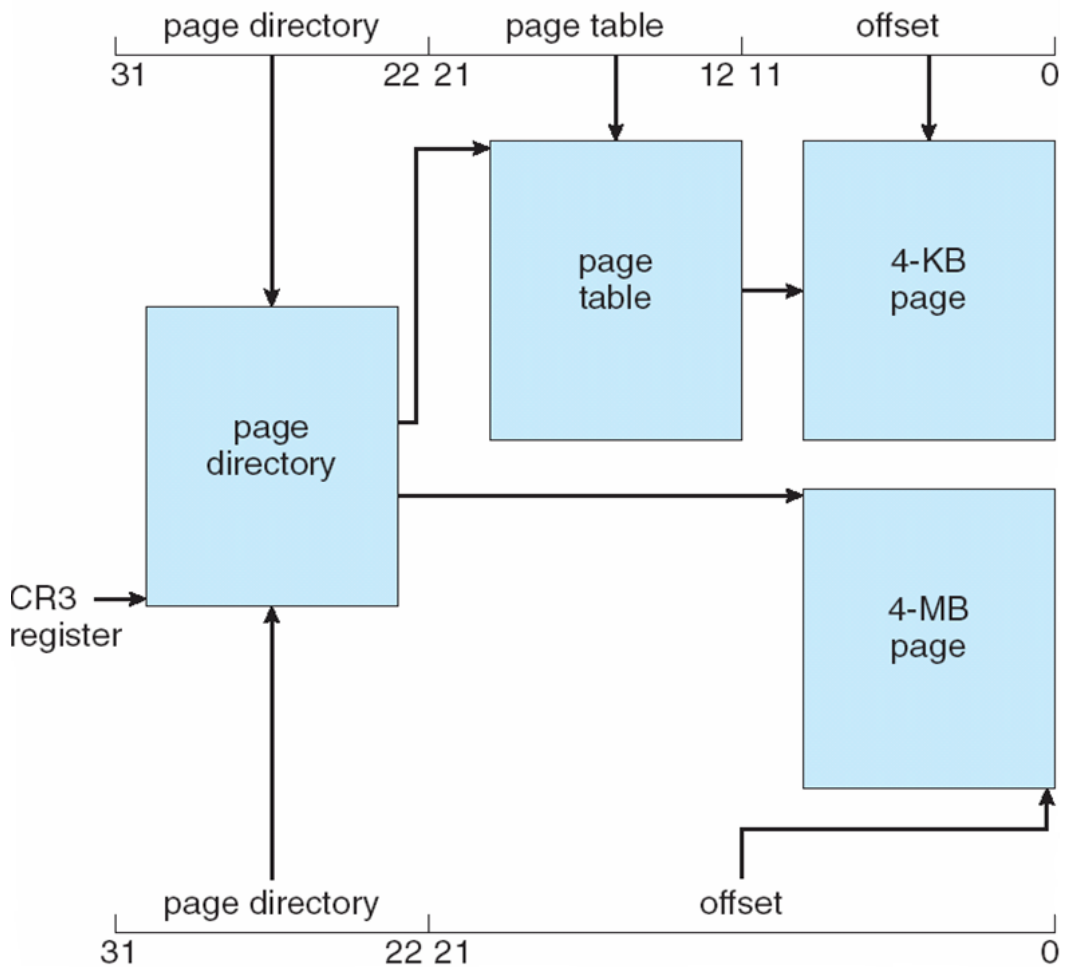


Etude de cas - Intel Pentium (IA-32)

Unité de pagination

- la pagination est activée : flag PG du registre cr0
- taille de page : 4K
- cr3 contient l'@physique de la "page directory"

- Pagination étendue : pages de taille 4M au lieu de 4K

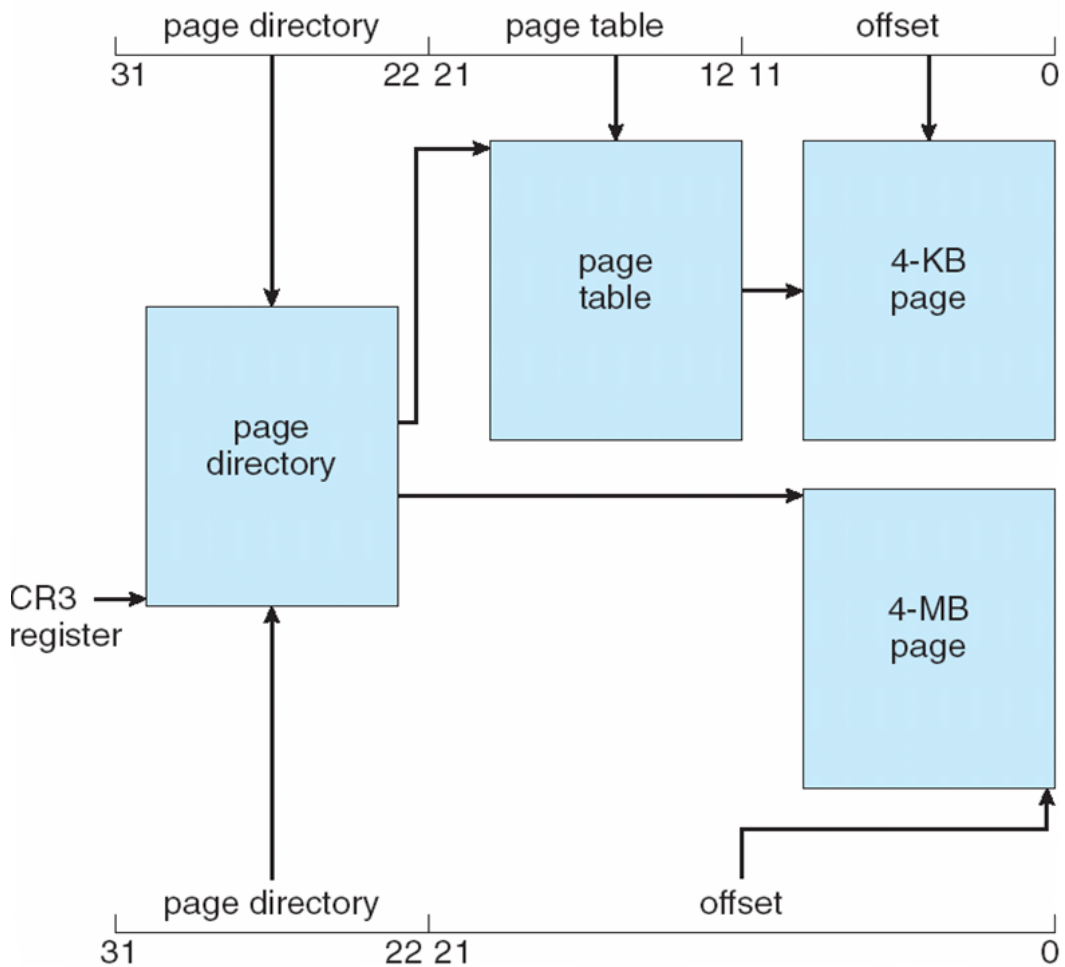


Etude de cas - Intel Pentium (IA-32)

Unité de pagination

- la pagination est activée : flag PG du registre cr0
- taille de page : 4K
- cr3 contient l'@physique de la "page directory"

- Pagination étendue : pages de taille 4M au lieu de 4K
  - activée : flag "Page Size" de l'entrée dans "page directory"

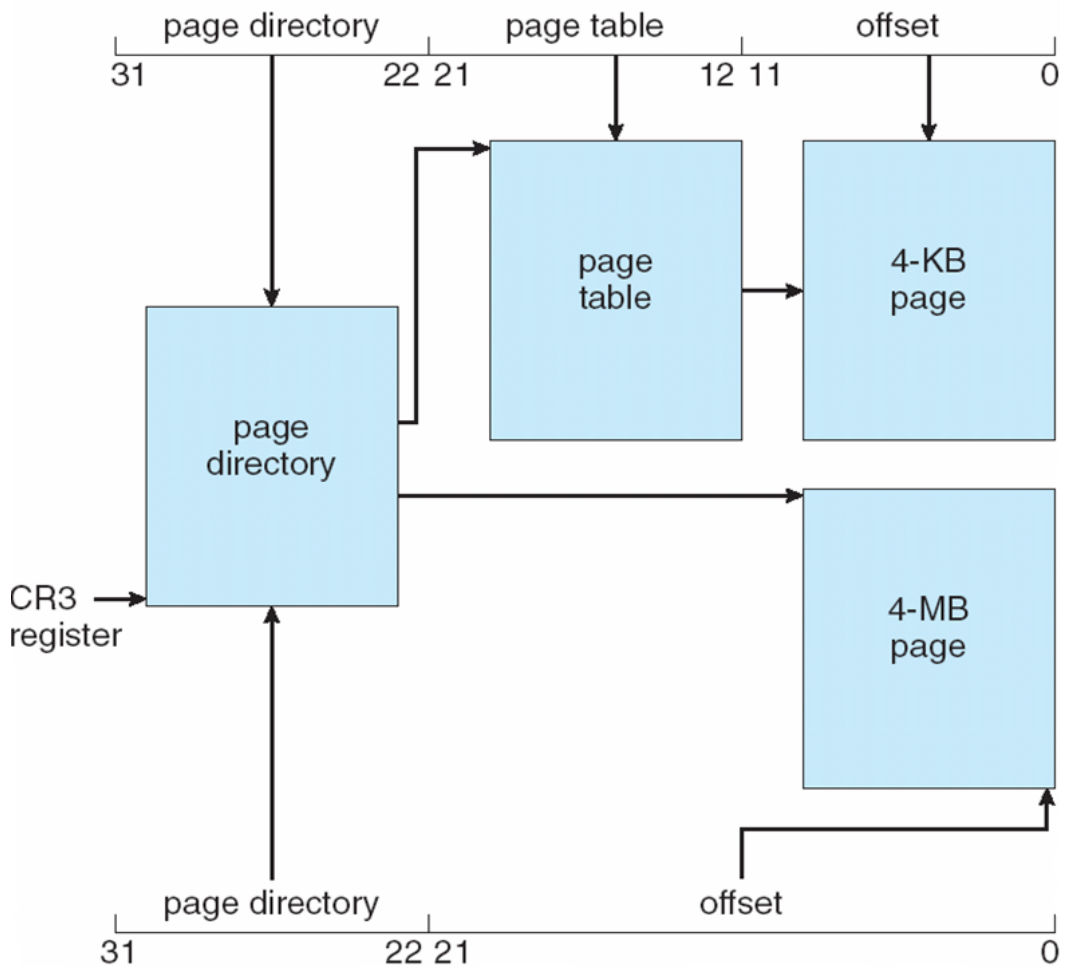


Etude de cas - Intel Pentium (IA-32)

Unité de pagination

- la pagination est activée : flag PG du registre cr0
- taille de page : 4K
- cr3 contient l'@physique de la "page directory"

- Pagination étendue : pages de taille 4M au lieu de 4K
  - activée : flag "Page Size" de l'entrée dans "page directory"
  - @linéaire = page (10bits) + offset (22 bits)



Etude de cas : Linux

Organisaion physique de la mémoire

- Machine 32 bits (Pentium)
- l'espace d'adressage d'un processus = 2^{32} = 4Go :
  - 3 Go pour le processus
  - 1 Go réservé et utilisé par le processus en mode noyau : table des pages et autres données du noyau
- l'espace d'adressage est divisé en zones homogènes et contiguës

zone	physical memory
ZONE_DMA	< 16 MB
ZONE_NORMAL	16 .. 896 MB
ZONE_HIGHMEM	> 896 MB

Etude de cas : Linux

Espace d'adressage d'un processus : régions

- une région contient un ensemble de pages consécutives possédant les mêmes propriétés de protection : région de code, région de fichiers mappés . . .
- à un processus, est associée une liste de descripteurs de régions (vm_area_struct)
  - **vm_mm** : la mm_struct à laquelle appartient cette région.

Etude de cas : Linux

Espace d'adressage d'un processus : régions

- une région contient un ensemble de pages consécutives possédant les mêmes propriétés de protection : région de code, région de fichiers mappés . . .
- à un processus, est associée une liste de descripteurs de régions (vm_area_struct)
  - **vm_mm** : la mm_struct à laquelle appartient cette région.
  - **vm_start, vm_end** : l'adresse de début et fin de la région.

Etude de cas : Linux

Espace d'adressage d'un processus : régions

- une région contient un ensemble de pages consécutives possédant les mêmes propriétés de protection : région de code, région de fichiers mappés . . .
- à un processus, est associée une liste de descripteurs de régions (vm_area_struct)
  - **vm_mm** : la mm_struct à laquelle appartient cette région.
  - **vm_start, vm_end** : l'adresse de début et fin de la région.
  - **vm_next** : toutes les régions d'un processus sont reliées entre elles dans l'ordre de leurs adresses par **vm_next**.

Etude de cas : Linux

Espace d'adressage d'un processus : régions

- une région contient un ensemble de pages consécutives possédant les mêmes propriétés de protection : région de code, région de fichiers mappés . . .
- à un processus, est associée une liste de descripteurs de régions (vm_area_struct)
  - **vm_mm** : la mm_struct à laquelle appartient cette région.
  - **vm_start, vm_end** : l'adresse de début et fin de la région.
  - **vm_next** : toutes les régions d'un processus sont reliées entre elles dans l'ordre de leurs adresses par **vm_next**.
  - **vm_page_prot** : les flags de protection positionnés dans chacune des entrées de la table des pages de cette région.

Etude de cas : Linux

Espace d'adressage d'un processus : régions

- une région contient un ensemble de pages consécutives possédant les mêmes propriétés de protection : région de code, région de fichiers mappés . . .
- à un processus, est associée une liste de descripteurs de régions (vm_area_struct)
  - **vm_mm** : la mm_struct à laquelle appartient cette région.
  - **vm_start, vm_end** : l'adresse de début et fin de la région.
  - **vm_next** : toutes les régions d'un processus sont reliées entre elles dans l'ordre de leurs adresses par **vm_next**.
  - **vm_page_prot** : les flags de protection positionnés dans chacune des entrées de la table des pages de cette région.
  - **vm_flags** : un ensemble de flags décrivant les propriétés et la protection de la région :

Etude de cas : Linux

Espace d'adressage d'un processus : régions

- une région contient un ensemble de pages consécutives possédant les mêmes propriétés de protection : région de code, région de fichiers mappés . . .
- à un processus, est associée une liste de descripteurs de régions (vm_area_struct)
  - **vm_mm** : la mm_struct à laquelle appartient cette région.
  - **vm_start, vm_end** : l'adresse de début et fin de la région.
  - **vm_next** : toutes les régions d'un processus sont reliées entre elles dans l'ordre de leurs adresses par **vm_next**.
  - **vm_page_prot** : les flags de protection positionnés dans chacune des entrées de la table des pages de cette région.
  - **vm_flags** : un ensemble de flags décrivant les propriétés et la protection de la région :
    - VM_READ

Etude de cas : Linux

Espace d'adressage d'un processus : régions

- une région contient un ensemble de pages consécutives possédant les mêmes propriétés de protection : région de code, région de fichiers mappés . . .
- à un processus, est associée une liste de descripteurs de régions (vm_area_struct)
  - **vm_mm** : la mm_struct à laquelle appartient cette région.
  - **vm_start, vm_end** : l'adresse de début et fin de la région.
  - **vm_next** : toutes les régions d'un processus sont reliées entre elles dans l'ordre de leurs adresses par **vm_next**.
  - **vm_page_prot** : les flags de protection positionnés dans chacune des entrées de la table des pages de cette région.
  - **vm_flags** : un ensemble de flags décrivant les propriétés et la protection de la région :
    - VM_READ
    - VM_WRITE

Etude de cas : Linux

Espace d'adressage d'un processus : régions

- une région contient un ensemble de pages consécutives possédant les mêmes propriétés de protection : région de code, région de fichiers mappés . . .
- à un processus, est associée une liste de descripteurs de régions (vm_area_struct)
  - **vm_mm** : la mm_struct à laquelle appartient cette région.
  - **vm_start, vm_end** : l'adresse de début et fin de la région.
  - **vm_next** : toutes les régions d'un processus sont reliées entre elles dans l'ordre de leurs adresses par **vm_next**.
  - **vm_page_prot** : les flags de protection positionnés dans chacune des entrées de la table des pages de cette région.
  - **vm_flags** : un ensemble de flags décrivant les propriétés et la protection de la région :
    - VM_READ
    - VM_WRITE
    - VM_EXEC

Etude de cas : Linux

Espace d'adressage d'un processus : régions

- une région contient un ensemble de pages consécutives possédant les mêmes propriétés de protection : région de code, région de fichiers mappés . . .
- à un processus, est associée une liste de descripteurs de régions (vm_area_struct)
  - **vm_mm** : la mm_struct à laquelle appartient cette région.
  - **vm_start, vm_end** : l'adresse de début et fin de la région.
  - **vm_next** : toutes les régions d'un processus sont reliées entre elles dans l'ordre de leurs adresses par **vm_next**.
  - **vm_page_prot** : les flags de protection positionnés dans chacune des entrées de la table des pages de cette région.
  - **vm_flags** : un ensemble de flags décrivant les propriétés et la protection de la région :
    - VM_READ
    - VM_WRITE
    - VM_EXEC
    - VM_SHARED

Etude de cas : Linux

Espace d'adressage d'un processus : régions

- une région contient un ensemble de pages consécutives possédant les mêmes propriétés de protection : région de code, région de fichiers mappés . . .
- à un processus, est associée une liste de descripteurs de régions (vm_area_struct)
  - **vm_mm** : la mm_struct à laquelle appartient cette région.
  - **vm_start, vm_end** : l'adresse de début et fin de la région.
  - **vm_next** : toutes les régions d'un processus sont reliées entre elles dans l'ordre de leurs adresses par **vm_next**.
  - **vm_page_prot** : les flags de protection positionnés dans chacune des entrées de la table des pages de cette région.
  - **vm_flags** : un ensemble de flags décrivant les propriétés et la protection de la région :
    - VM_READ
    - VM_WRITE
    - VM_EXEC
    - VM_SHARED
    - . . .

Espace d'adressage d'un processus : régions

cat /proc/self/maps

```
08048000-0804c000 r-xp 00000000 03:07 293186 /bin/cat
0804c000-0804d000 rw-p 00003000 03:07 293186 /bin/cat
0804d000-0806e000 rwxp 00000000 00:00 0
40000000-40016000 r-xp 00000000 03:07 390929 /lib/ld-2.3.2.so
40016000-40017000 rw-p 00015000 03:07 390929 /lib/ld-2.3.2.so
40017000-40018000 rw-p 00000000 00:00 0
40027000-4014f000 r-xp 00000000 03:07 390935 /lib/libc-2.3.2.so
4014f000-40157000 rw-p 00127000 03:07 390935 /lib/libc-2.3.2.so
40157000-4015a000 rw-p 00000000 00:00 0

bfffe000-c0000000 rwxp fffff000 00:00 0
```

Espace d'adressage d'un processus : régions

cat /proc/self/maps

```
08048000-0804c000 r-xp 00000000 03:07 293186 /bin/cat
0804c000-0804d000 rw-p 00003000 03:07 293186 /bin/cat
0804d000-0806e000 rwxp 00000000 00:00 0
40000000-40016000 r-xp 00000000 03:07 390929 /lib/ld-2.3.2.so
40016000-40017000 rw-p 00015000 03:07 390929 /lib/ld-2.3.2.so
40017000-40018000 rw-p 00000000 00:00 0
40027000-4014f000 r-xp 00000000 03:07 390935 /lib/libc-2.3.2.so
4014f000-40157000 rw-p 00127000 03:07 390935 /lib/libc-2.3.2.so
40157000-4015a000 rw-p 00000000 00:00 0

bfffe000-c0000000 rwxp fffff000 00:00 0
```

Espace d'adressage d'un processus : régions

cat /proc/self/maps

```
08048000-0804c000 r-xp 00000000 03:07 293186 /bin/cat
0804c000-0804d000 rw-p 00003000 03:07 293186 /bin/cat
0804d000-0806e000 rwxp 00000000 00:00 0
40000000-40016000 r-xp 00000000 03:07 390929 /lib/ld-2.3.2.so
40016000-40017000 rw-p 00015000 03:07 390929 /lib/ld-2.3.2.so
40017000-40018000 rw-p 00000000 00:00 0
40027000-4014f000 r-xp 00000000 03:07 390935 /lib/libc-2.3.2.so
4014f000-40157000 rw-p 00127000 03:07 390935 /lib/libc-2.3.2.so
40157000-4015a000 rw-p 00000000 00:00 0

bfffe000-c0000000 rwxp fffff000 00:00 0
```

Espace d'adressage d'un processus : régions

cat /proc/self/maps

```
08048000-0804c000 r-xp 00000000 03:07 293186 /bin/cat
0804c000-0804d000 rw-p 00003000 03:07 293186 /bin/cat
0804d000-0806e000 rwxp 00000000 00:00 0
40000000-40016000 r-xp 00000000 03:07 390929 /lib/ld-2.3.2.so
40016000-40017000 rw-p 00015000 03:07 390929 /lib/ld-2.3.2.so
40017000-40018000 rw-p 00000000 00:00 0
40027000-4014f000 r-xp 00000000 03:07 390935 /lib/libc-2.3.2.so
4014f000-40157000 rw-p 00127000 03:07 390935 /lib/libc-2.3.2.so
40157000-4015a000 rw-p 00000000 00:00 0

bfffe000-c0000000 rwxp fffff000 00:00 0
```

Espace d'adressage d'un processus : régions

cat /proc/self/maps

```
08048000-0804c000 r-xp 00000000 03:07 293186 /bin/cat
0804c000-0804d000 rw-p 00003000 03:07 293186 /bin/cat
0804d000-0806e000 rwxp 00000000 00:00 0
40000000-40016000 r-xp 00000000 03:07 390929 /lib/ld-2.3.2.so
40016000-40017000 rw-p 00015000 03:07 390929 /lib/ld-2.3.2.so
40017000-40018000 rw-p 00000000 00:00 0
40027000-4014f000 r-xp 00000000 03:07 390935 /lib/libc-2.3.2.so
4014f000-40157000 rw-p 00127000 03:07 390935 /lib/libc-2.3.2.so
40157000-4015a000 rw-p 00000000 00:00 0

bfffe000-c0000000 rwxp fffff000 00:00 0
```

Espace d'adressage d'un processus : régions

cat /proc/self/maps

```
08048000-0804c000 r-xp 00000000 03:07 293186 /bin/cat
0804c000-0804d000 rw-p 00003000 03:07 293186 /bin/cat
0804d000-0806e000 rwxp 00000000 00:00 0
40000000-40016000 r-xp 00000000 03:07 390929 /lib/ld-2.3.2.so
40016000-40017000 rw-p 00015000 03:07 390929 /lib/ld-2.3.2.so
40017000-40018000 rw-p 00000000 00:00 0
40027000-4014f000 r-xp 00000000 03:07 390935 /lib/libc-2.3.2.so
4014f000-40157000 rw-p 00127000 03:07 390935 /lib/libc-2.3.2.so
40157000-4015a000 rw-p 00000000 00:00 0

bfffe000-c0000000 rwxp fffff000 00:00 0
```

Espace d'adressage d'un processus : régions

cat /proc/self/maps

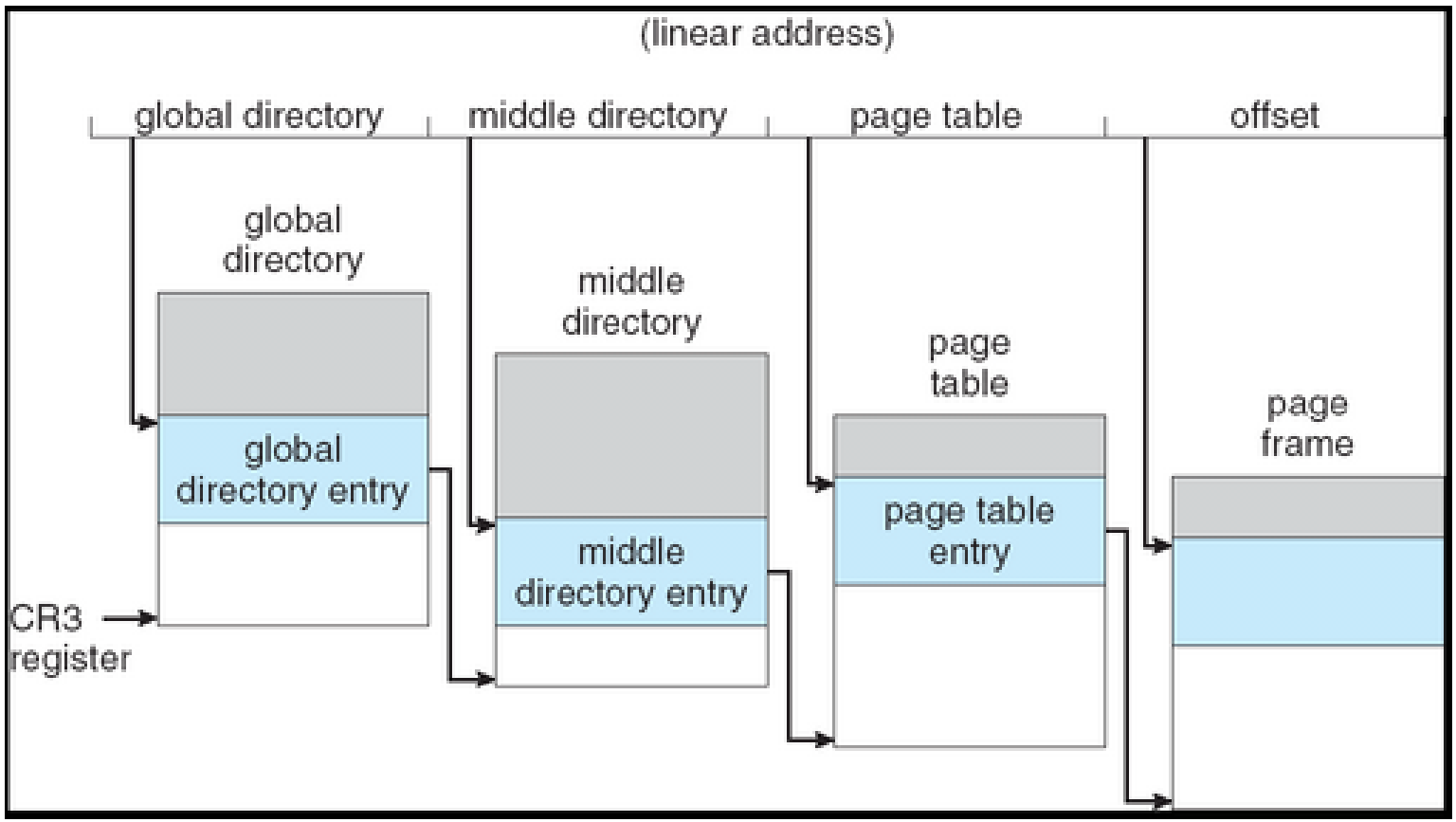
```
08048000-0804c000 r-xp 00000000 03:07 293186 /bin/cat
0804c000-0804d000 rw-p 00003000 03:07 293186 /bin/cat
0804d000-0806e000 rwxp 00000000 00:00 0
40000000-40016000 r-xp 00000000 03:07 390929 /lib/ld-2.3.2.so
40016000-40017000 rw-p 00015000 03:07 390929 /lib/ld-2.3.2.so
40017000-40018000 rw-p 00000000 00:00 0
40027000-4014f000 r-xp 00000000 03:07 390935 /lib/libc-2.3.2.so
4014f000-40157000 rw-p 00127000 03:07 390935 /lib/libc-2.3.2.so
40157000-4015a000 rw-p 00000000 00:00 0

bfffe000-c0000000 rwxp fffff000 00:00 0
```

Etude de cas : Linux

Pagination à 3 niveaux

- Taille d'une page mémoire est 4Ko
- Pagination à 3 niveaux généralisée à toutes les architectures



Etude de cas : Linux

Remplacement de pages

- Tenter de garder un minimum de pages libres

Etude de cas : Linux

Remplacement de pages

- Tenter de garder un minimum de pages libres
- Au départ, init lance un démon de pagination kswapd qui s'exécute toutes les secondes

Etude de cas : Linux

Remplacement de pages

- Tenter de garder un minimum de pages libres
- Au départ, init lance un démon de pagination kswapd qui s'exécute toutes les secondes
- kswapd vérifie le nombre de pages libres, si bon, il se rendort

Etude de cas : Linux

Remplacement de pages

- Tenter de garder un minimum de pages libres
- Au départ, init lance un démon de pagination kswapd qui s'exécute toutes les secondes
- kswapd vérifie le nombre de pages libres, si bon, il se rendort
- sinon il lance 3 procédures de récupération de pages :

Etude de cas : Linux

Remplacement de pages

- Tenter de garder un minimum de pages libres
- Au départ, init lance un démon de pagination kswapd qui s'exécute toutes les secondes
- kswapd vérifie le nombre de pages libres, si bon, il se rendort
- sinon il lance 3 procédures de récupération de pages :
  - cache de pagination ou le tampon du cache

Etude de cas : Linux

Remplacement de pages

- Tenter de garder un minimum de pages libres
- Au départ, init lance un démon de pagination kswapd qui s'exécute toutes les secondes
- kswapd vérifie le nombre de pages libres, si bon, il se rendort
- sinon il lance 3 procédures de récupération de pages :
  - cache de pagination ou le tampon du cache
  - partagées qu'aucun processus ne semble utiliser abondamment

Etude de cas : Linux

Remplacement de pages

- Tenter de garder un minimum de pages libres
- Au départ, init lance un démon de pagination kswapd qui s'exécute toutes les secondes
- kswapd vérifie le nombre de pages libres, si bon, il se rendort
- sinon il lance 3 procédures de récupération de pages :
  - cache de pagination ou le tampon du cache
  - partagées qu'aucun processus ne semble utiliser abondamment
  - utilisateur ordinaires

Etude de cas : Linux

Remplacement de pages

- Tenter de garder un minimum de pages libres
- Au départ, init lance un démon de pagination kswapd qui s'exécute toutes les secondes
- kswapd vérifie le nombre de pages libres, si bon, il se rendort
- sinon il lance 3 procédures de récupération de pages :
  - cache de pagination ou le tampon du cache
  - partagées qu'aucun processus ne semble utiliser abondamment
  - utilisateur ordinaires
- bdflush se réveille périodiquement. Il vérifie si une trop forte proportion de pages est modifiée : écriture sur disque

Etude de cas : Linux

Remplacement de pages : kswapd