Les systèmes d'exploitation

Wadoud Bousdira[1]
wadoud.bousdira@univ-orleans.fr

[1]LIFO, University of Orléans
Orléans, France

Orléans, 2023

La mémoire

Introduction

Gérer la mémoire pour :

1. partager la mémoire physique entre les programmes et les données des processus prêts
2. mettre en place des paramètres de calculs d'adresse permettant de transformer
   - une adresse virtuelle (adresse générée par la CPU)
   - en adresse physique (adresse vue par l'unité de mémoire).

Introduction

- Adresse physique : numéro d'un octet de la mémoire physique
- Adresse virtuelle : numéro d'un octet dans un espace sans rapport avec la mémoire physique
  - numéro d'octet dans un espace logique
  - numéro de segment, déplacement dans le segment
  - numéro de page, déplacement dans la page

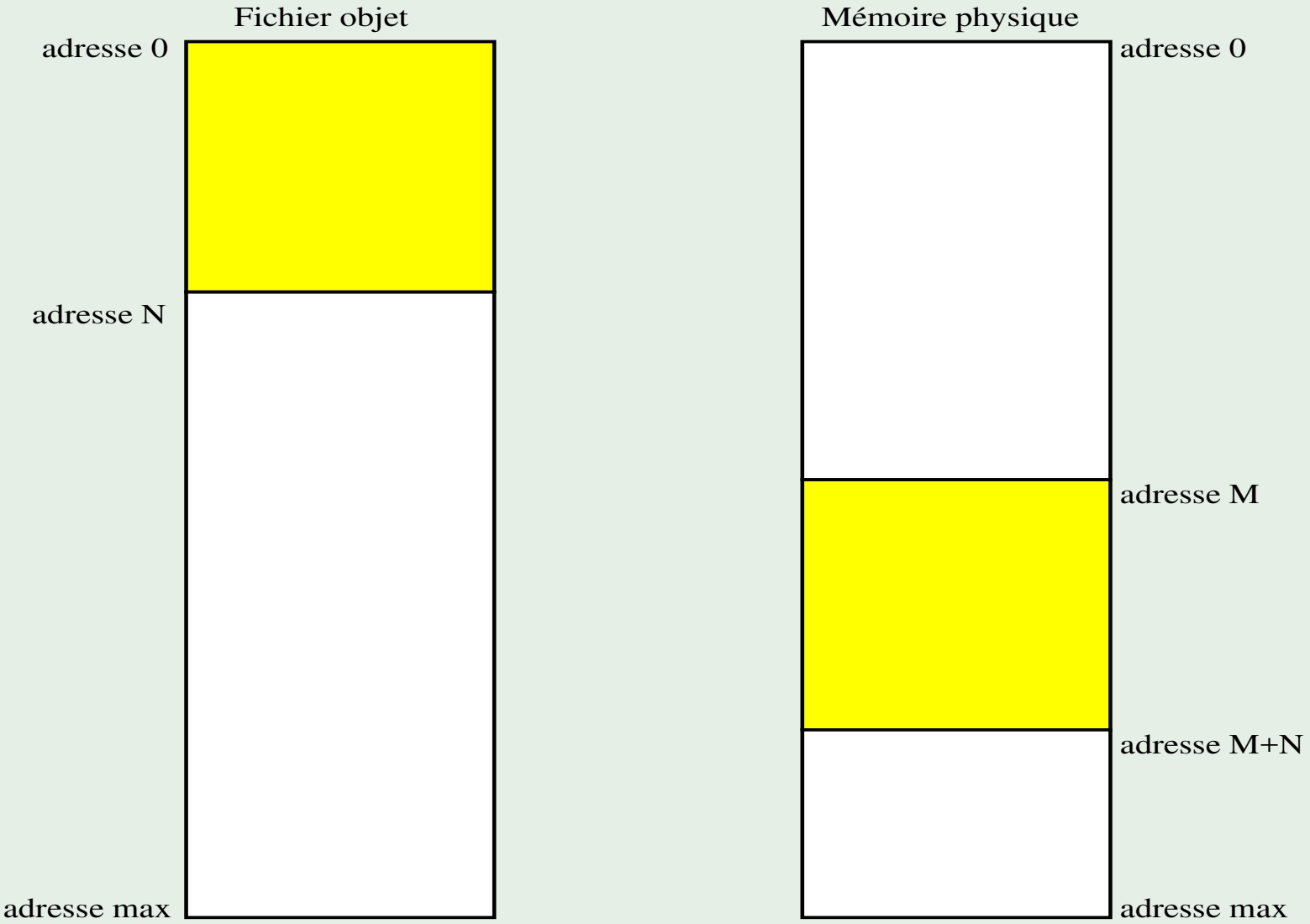
Introduction

Les adresses physiques/virtuelles sont préalablement calculées par les utilitaires de préparation des programmes :

- le compilateur
- l'assembleur
- l'éditeur de liens.

Ne pas ralentir les accès à la mémoire :
la transformation adresse virtuelle $\longleftrightarrow$ adresse physique est faite par le matériel Memory Management Unit lors de l'exécution.

Le chargement



Introduction

Les techniques de gestion de la mémoire sont très conditionnées par les caractéristiques du matériel disponible en dehors des registres usuels

- accumulateurs
- registres d'index
- registres condition.

Introduction

Hiérarchie de mémoires

- Mémoire volatile : ne conserve son contenu que tant qu'elle est alimentée électriquement.
- Mémoire permanente : conserve son contenu, même sans alimentation électrique.

Hiérarchie de mémoires

Mémoires volatiles (dynamiques)

Accessibles en lecture et en écriture.

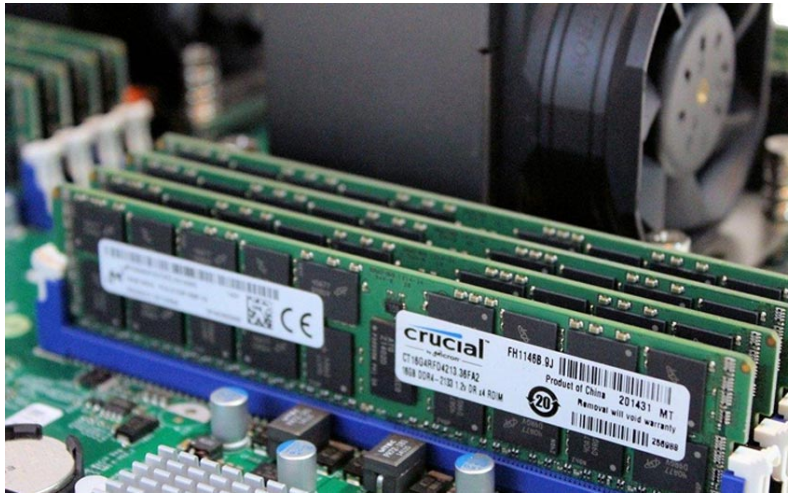
- Registres : au plus près de la CPU.
  32 ou 64 bits, 1 ns
- Mémoire cache :
  Koctets, 5 ns
- Mémoire centrale :
  Goctets, 10 ns.

Hiérarchie de mémoires

La mémoire centrale (DRAM)

Direct Random Access Memory.

- constituée d'un ensemble de mots mémoire, chaque mot représenté par un ensemble d'octets,
- chaque octet est repéré par une adresse,
- sous forme de barrettes, comportant des puces mémoire.



Hiérarchie de mémoires

La mémoire cache

mémoires caches du processeur

- Très rapides,
- gérées de façon à contenir le + souvent possible l'information dont le processeur a besoin à l'instant présent ou dans un futur proche
- très chères,
- induisent des difficultés d'intégration.
- Hiérarchie de mémoires cache mise en place sur 3 ou 4 niveaux :
  1. cache L1, petite, très rapide, placée dans le processeur.
  2. cache L2, capacité + importante, à l'extérieur du processeur, mais sur la même puce.
  3. cache L3, placé sur la carte mère.

La mémoire cache

Fonctionnement

- Le cache contient une copie des données originelles lorsqu'elles sont coûteuses (en terme de temps d'accès) à récupérer ou à calculer par rapport au temps d'accès au cache
- une fois les données stockées dans le cache, l'utilisation future de ces données peut être réalisée en accédant à la copie en cache plutôt qu'en récupérant ou recalculant les données,
  $\Rightarrow$ abaisse le temps d'accès moyen.

La mémoire cache

Le processus fonctionne comme suit :

- l'élément demandeur (microprocesseur) demande une information
- le cache vérifie s'il possède cette information
  - s'il la possède, il la retransmet à l'élément demandeur : succès de cache
  - s'il ne la possède pas, il la demande à l'élément fournisseur (mémoire principale) : défaut de cache
  - l'élément fournisseur traite la demande et renvoie la réponse au cache
  - le cache la stocke pour utilisation ultérieure et la retransmet à l'élément demandeur.

La mémoire cache

La mémoire cache ↗ les performances grâce à 2 principes :

1. localité spatiale : indique que l'accès à une instruction située à une adresse X va probablement être suivi d'un accès à une zone tout proche de X
2. localité temporelle : indique que l'accès à une zone mémoire à un instant donné a de fortes chances de se reproduire dans la suite du programme.

Hiérarchie de mémoires

Mémoires permanentes

- Mémoires magnétiques
- Mémoires de type ROM (Read Only Memory) : seulement accessibles en lecture, mémoires mortes.
  Contenu écrit une seule fois, grâce à un dispositif particulier lors de leur fabrication. Puis leur information ne peut être que lue par le processeur.

Mémoires flash

de type intermédiaire.
Non volatiles mais réinscriptibles. ⇒ caractéristiques d'une mémoire vive (RAM) mais dont les données ne se volatilisent pas lors d'une mise hors tension (ROM).

Gestion de la mémoire (principale)

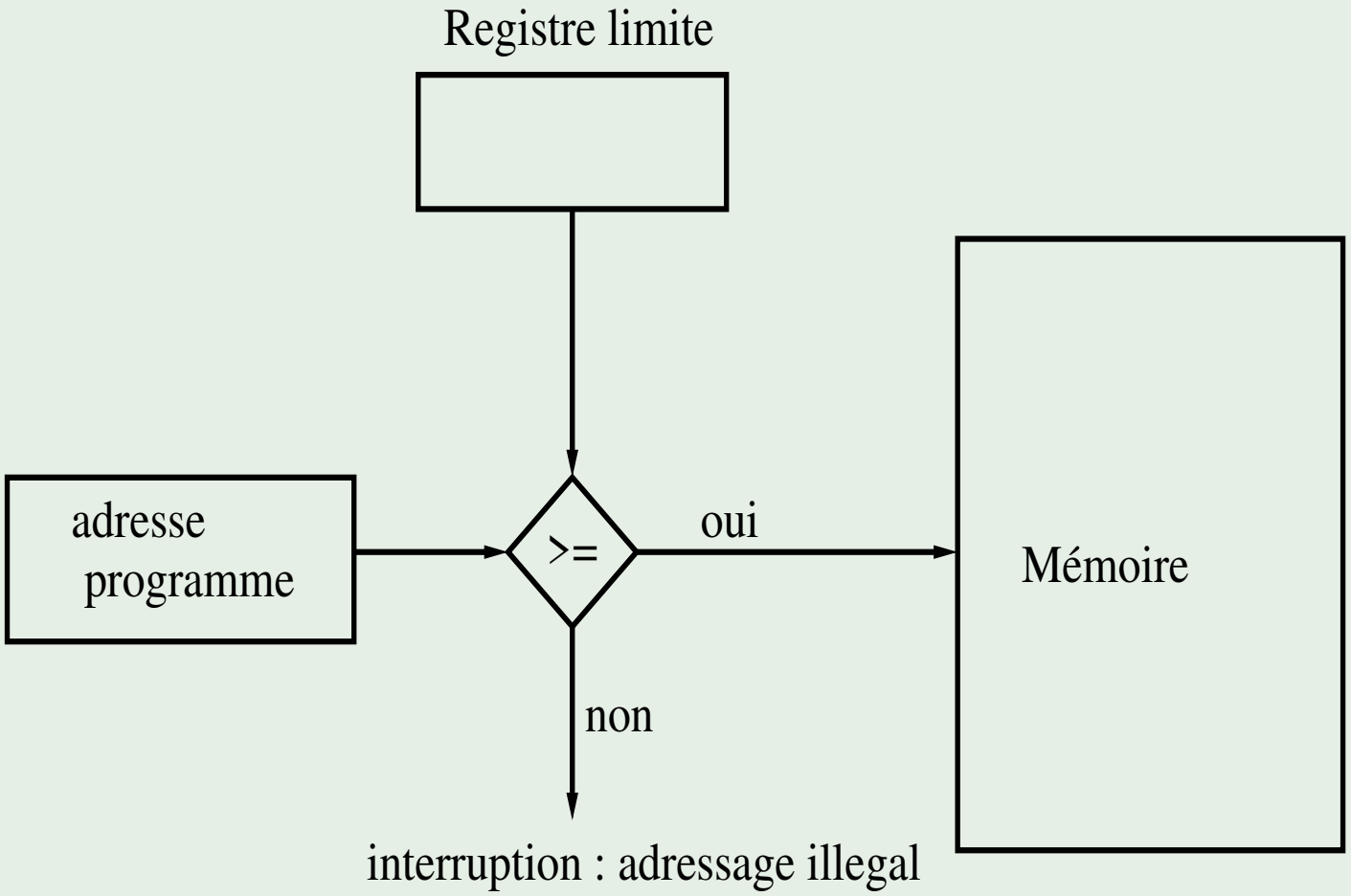
Espace d'adressage

Ensemble des adresses qu'un processus peut utiliser pour adresser la mémoire.

Monoprogrammation

- Un job unique sur le processeur à chaque instant :
- La mémoire est divisée en deux partitions :
  - une pour l'espace d'adressage du processeur (mémoire haute),
  - l'autre pour le SE résident (en général en mémoire basse).
- Utilisé par certains (petits) MS-DOS.
- Mécanisme de protection :
  la zone du SE est protégée par un registre limite R_l qui contient l'adresse de début du processus (code et données).

Monoprogrammation



Plusieurs stratégies pour implanter le registre limite : valeur fixe ou valeur variable.

Multiprogammation avec partitions fixes

On utilise la multiprogrammation pour optimiser l'utilisation du processeur et des E/S.

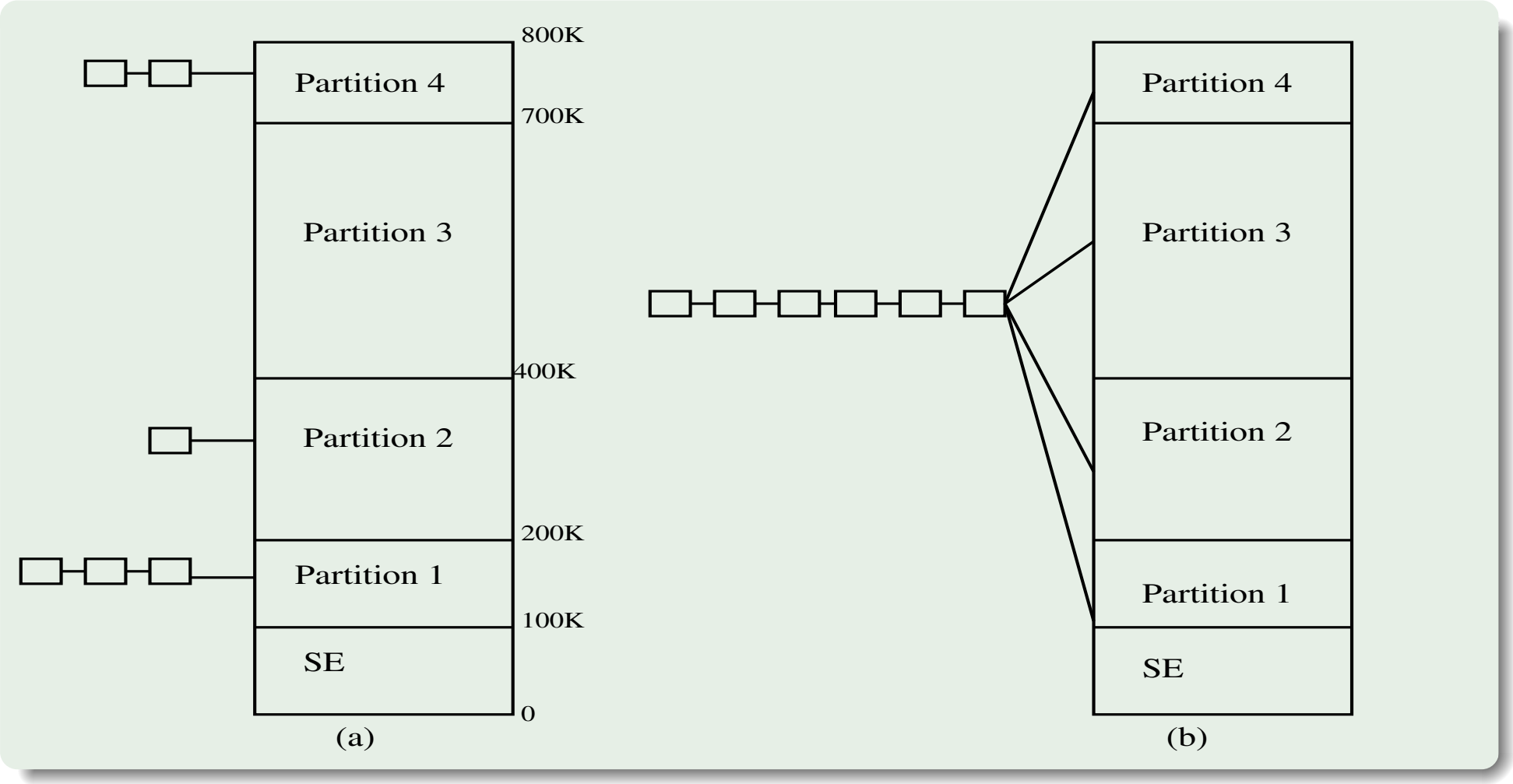
Idée :

- partitionner la mémoire en n partitions de taille fixe,
- chaque partition peut contenir exactement un processus.
- Degré de multiprogammation = nombre de partitions.

Problèmes :

- choisir un bon découpage en partitions de la mémoire
- décider dans quelle partition allouer chaque processus
- minimiser les gaspillages d'espace mémoire
- prendre en charge les translations d'adresses.

Multiprogrammation avec partitions fixes



- (a) Partitions fixes avec des files d'attente différentes
- (b) Partitions fixes avec une seule file d'attente

Gestion de la mémoire

Le partitionnement précédent n'est pas satisfaisant.

1. Que faire s'il y a trop de processus actifs en même temps ?
2. Comment protéger les processus les uns des autres ?
3. Et si un processus ne tient pas lui-même en mémoire ?

- Une réponse logicielle : le va-et-vient (swapping)
  répond à 1, partiellement à 3
- Une réponse matérielle : la mémoire virtuelle
  répond à 1, 2 et 3.

Gestion de la mémoire

- on partitionne dynamiquement la mémoire,
- on utilise une mémoire secondaire de stockage.

Problème : gérer les translations d'adresses.

Solution matérielle : les registres de base.

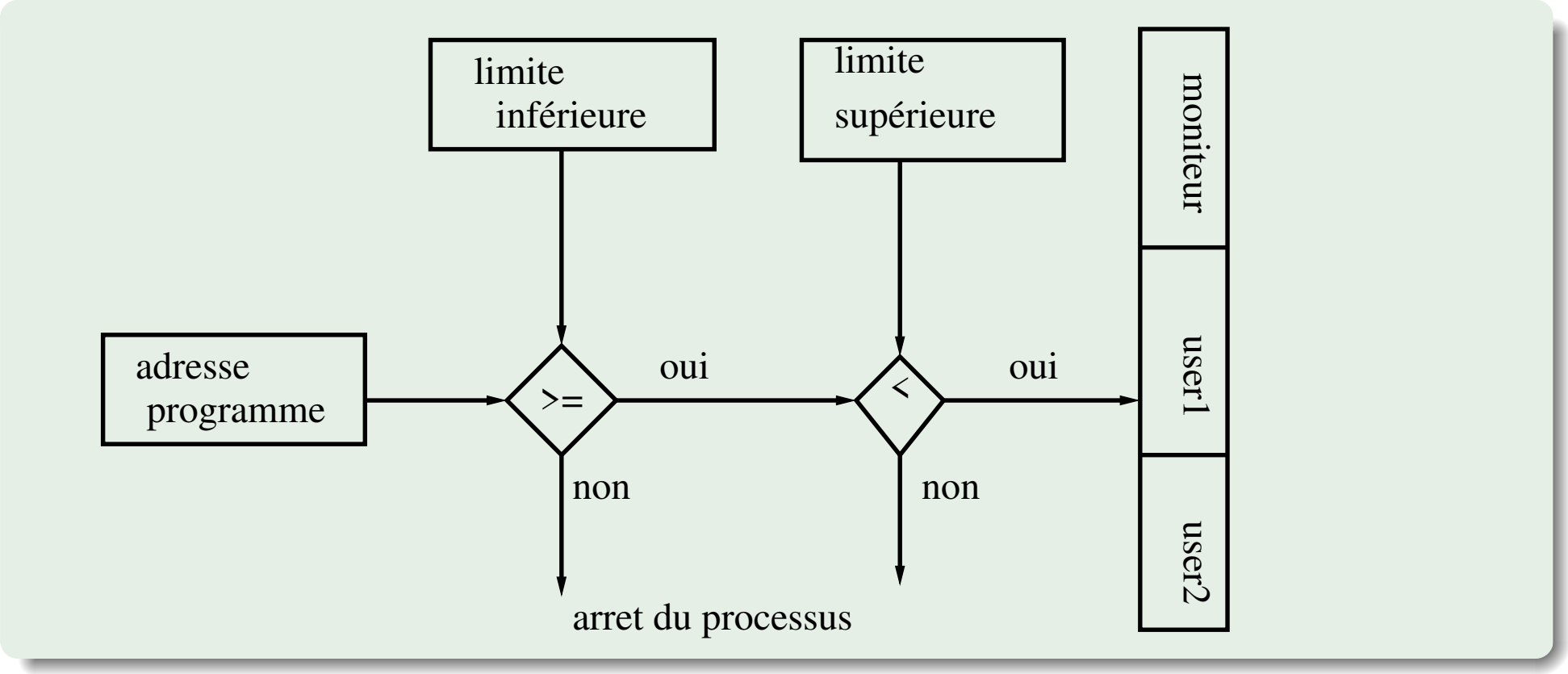
Gestion de la mémoire

Registres de base

L'UC est équipée de 2 registres matériels : le registre de base R_b et le registre de limite R_l.

- Chaque processus a son propre espace d'adressage,
- quand un programme est chargé en mémoire, il est placé là où il y a de la place pour un rangement dans des mots consécutifs,
- à l'exécution du programme, le SE range dans R_b l'adresse physique de début du programme, et dans R_l l'adresse suivant son adresse physique de fin.
- $a \rightsquigarrow a + R_b$ si $a + R_b < R_l$.

Gestion de la mémoire



- Méthode simple d'attribuer à chaque processus son espace d'adressage.
- Exige une addition et une comparaison à chaque référence mémoire !

Gestion de la mémoire

Le va-et-vient (swapping)

Les processus non actifs sont recopiés sur une mémoire de réserve. Ils sont ramenés en mémoire centrale à leur réactivation.

- La mémoire de réserve est un disque rapide, avec un espace nécessaire pour contenir les images mémoire de tous les programmes en cours d'exécution (résidant dans la file de *prêts*).
- Le temps de swapping est $\gg$ au temps de calcul,
  $\Rightarrow$ chaque processus doit s'exécuter pendant un temps $>$ temps de swap.
- Fragmentation de la mémoire, (multiple trous)
  $\rightsquigarrow$ compactage de la mémoire $\Rightarrow$ temps UC considérable.
- ne permet pas aux programmes de dépasser la taille de la mémoire physique.

Gestion de la mémoire

Le va-et-vient

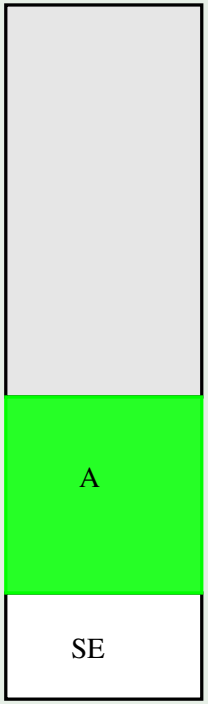
Allocation mémoire :

- si les processus sont créés avec une taille fixe invariable : simple
- sinon,
  (ex. le code contient une allocation dynamique de mémoire à partir du tas $\Rightarrow$ le segment de données $\nearrow$),
  $\rightsquigarrow$ déplacement de processus en mémoire.
- si le processus ne peut pas croître, et si plus de place sur le disque $\Rightarrow$ le processus attend ou il est tué !

Le va-et-vient

Exemple

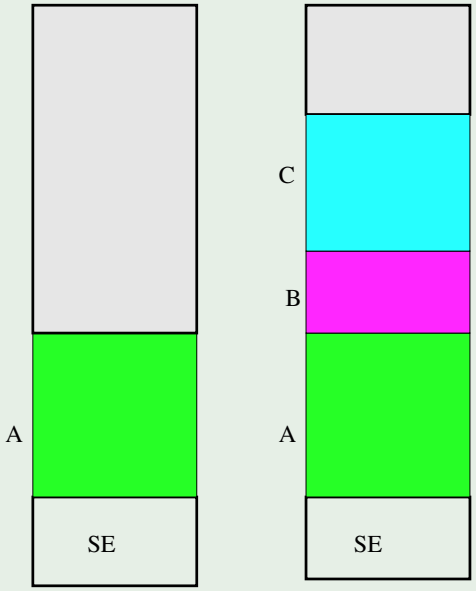
Au départ, seul le processus `A` est en mémoire.



Le va-et-vient

Exemple

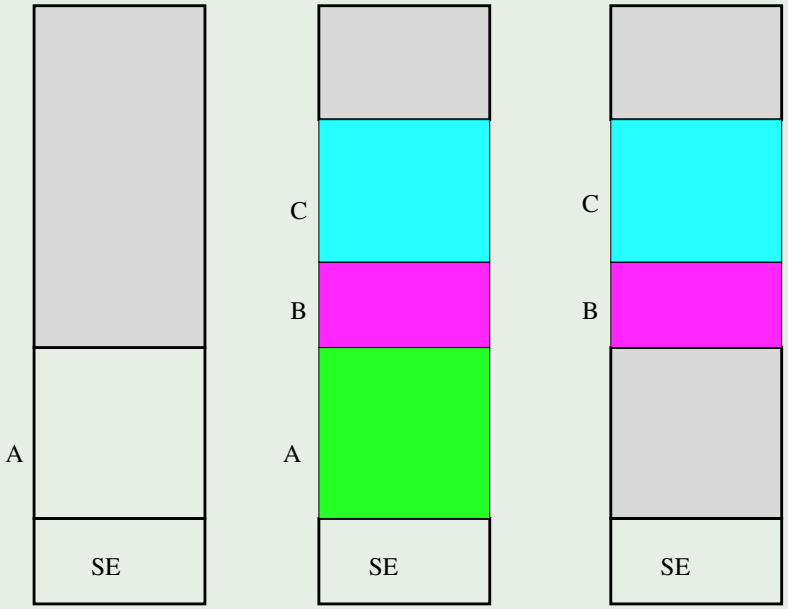
Les processus B puis C sont créés ou chargés depuis le disque.



Le va-et-vient

Exemple

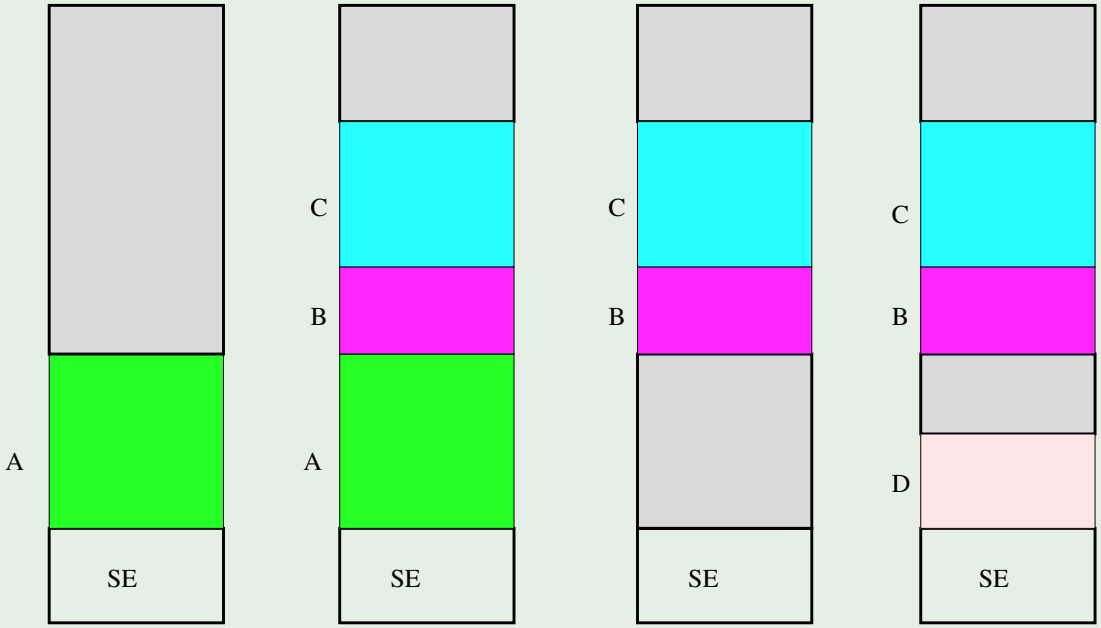
`A` est transféré sur le disque, l'espace mémoire libre se fragmente...



Le va-et-vient

Exemple

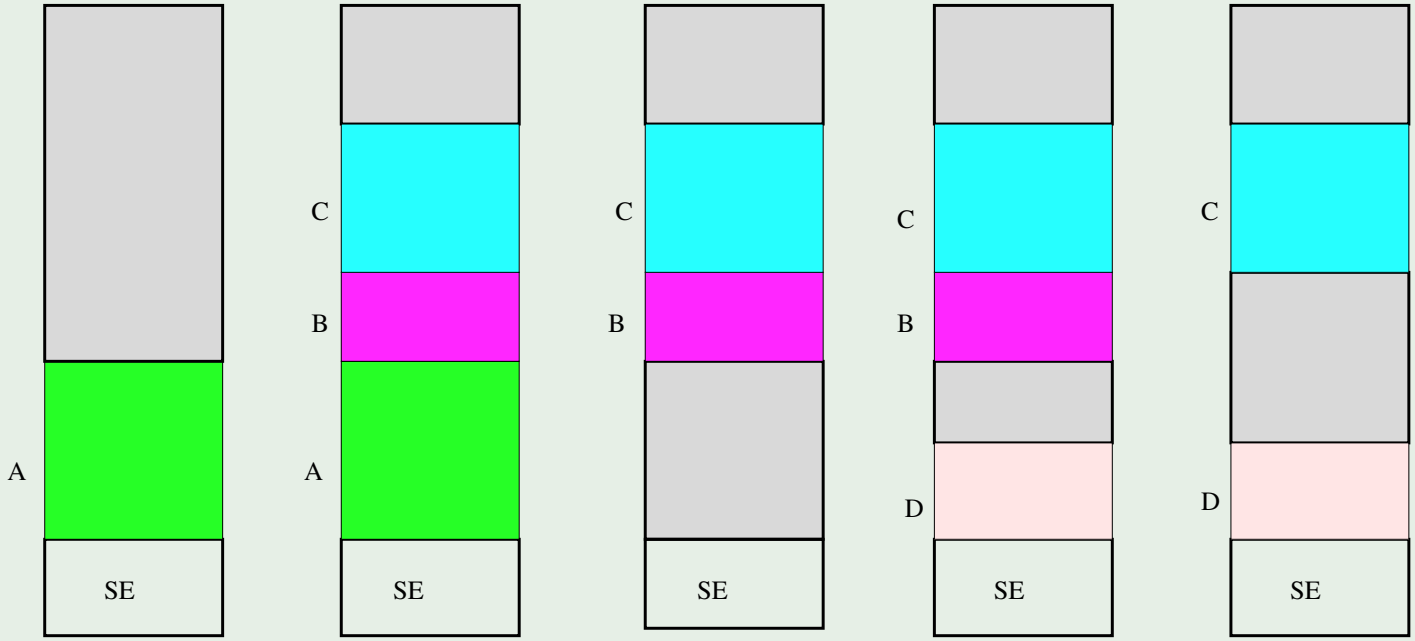
Le processus D est créé ou chargé depuis le disque. Le SE réactualise son information concernant l'espace libre.



Le va-et-vient

Exemple

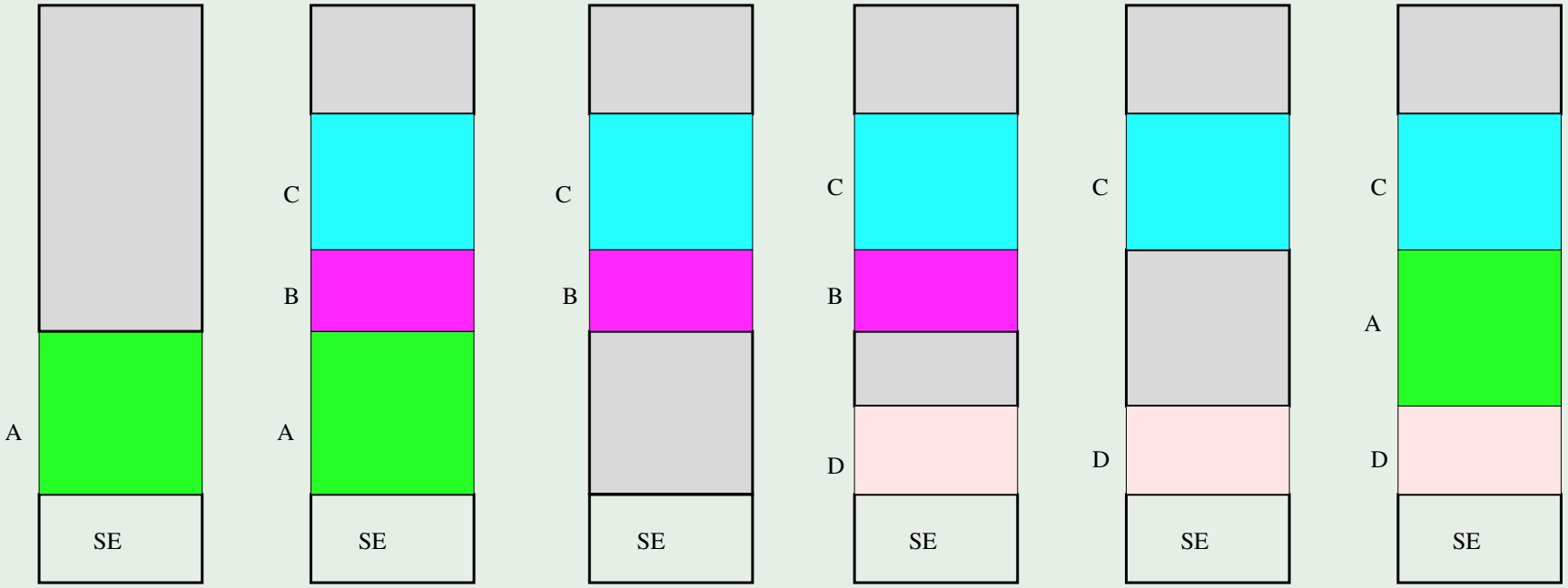
Le processus B est déchargé sur le disque. Mise à jour encore...



Le va-et-vient

Exemple

De nouveau, `A` est rechargé depuis le disque. L'adresse de chargement de `A` a changé et ses adresses en mémoire doivent être recalculées.



Allocation de mémoire pour un processus

En général, on alloue un peu de mémoire supplémentaire chaque fois qu'un processus est chargé ou déplacé,

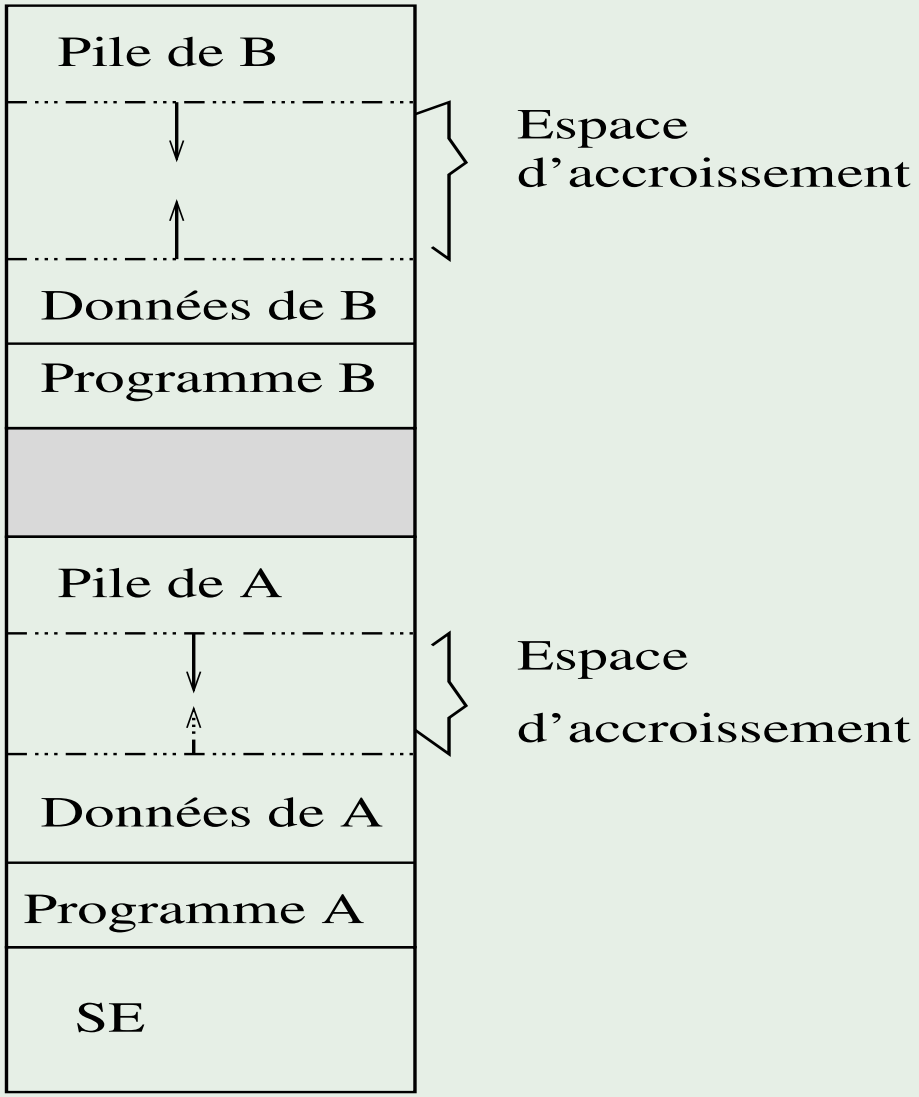
- permet de réduire le temps passé à déplacer les processus dont la taille n'est pas appropriée à la mémoire qui leur est allouée. 
- Lors du transfert vers le disque, seule la mémoire véritablement utilisée est recopiée.

Le va-et-vient

2 espaces d'accroissement :

1. un espace pour le segment de données : sert aux variables allouées et libérées dynamiquement,
2. un espace pour le segment de pile : sert aux variables locales et aux adresses de retour (appels de procédure).

Le va-et-vient



Le va-et-vient

Si la zone mémoire du processus devient insuffisante ?

- le processus est déplacé dans un espace libre suffisamment grand ou
- il est transféré hors de la mémoire jusqu'à la création d'une zone assez grande ou
- il est détruit.

La fragmentation

Un problème générique :

- partitionnement du va-et-vient
- allocation de mémoire,
- système de fichiers

- Méthode naïve : gestion de la mémoire par tables de bits (bit maps).
- Méthode usuelle : par listes chaînées.

La fragmentation

2 types de fragmentation

1. fragmentation interne : due à l'allocation d'un bloc mémoire beaucoup plus large que la place strictement requise par le processus. à charger dans ce bloc.
   $\Rightarrow$ gaspillage de la place mémoire restante.
2. fragmentation externe : due à l'impossibilité de satisfaire une requête bien qu'il y ait suffisamment de mémoire.
   Les blocs libres ne sont pas fusionnables car non contigus.

Le va-et-vient

Le compactage

Une solution au problème de la fragmentation externe :

- objectif : brasser les contenus de la mémoire pour placer toute la mémoire libre ensemble dans un seul bloc.
- Pas toujours possible. Il faut estimer son coût quand il est possible.
- L'algorithme le plus simple : déplacer tous les processus vers une extrémité de la mémoire. Tous les trous se déplacent dans l'autre direction $\Rightarrow$ grand trou de mémoire disponible. Peut être très cher !
- La sélection d'une stratégie de compactage optimale est très difficile.

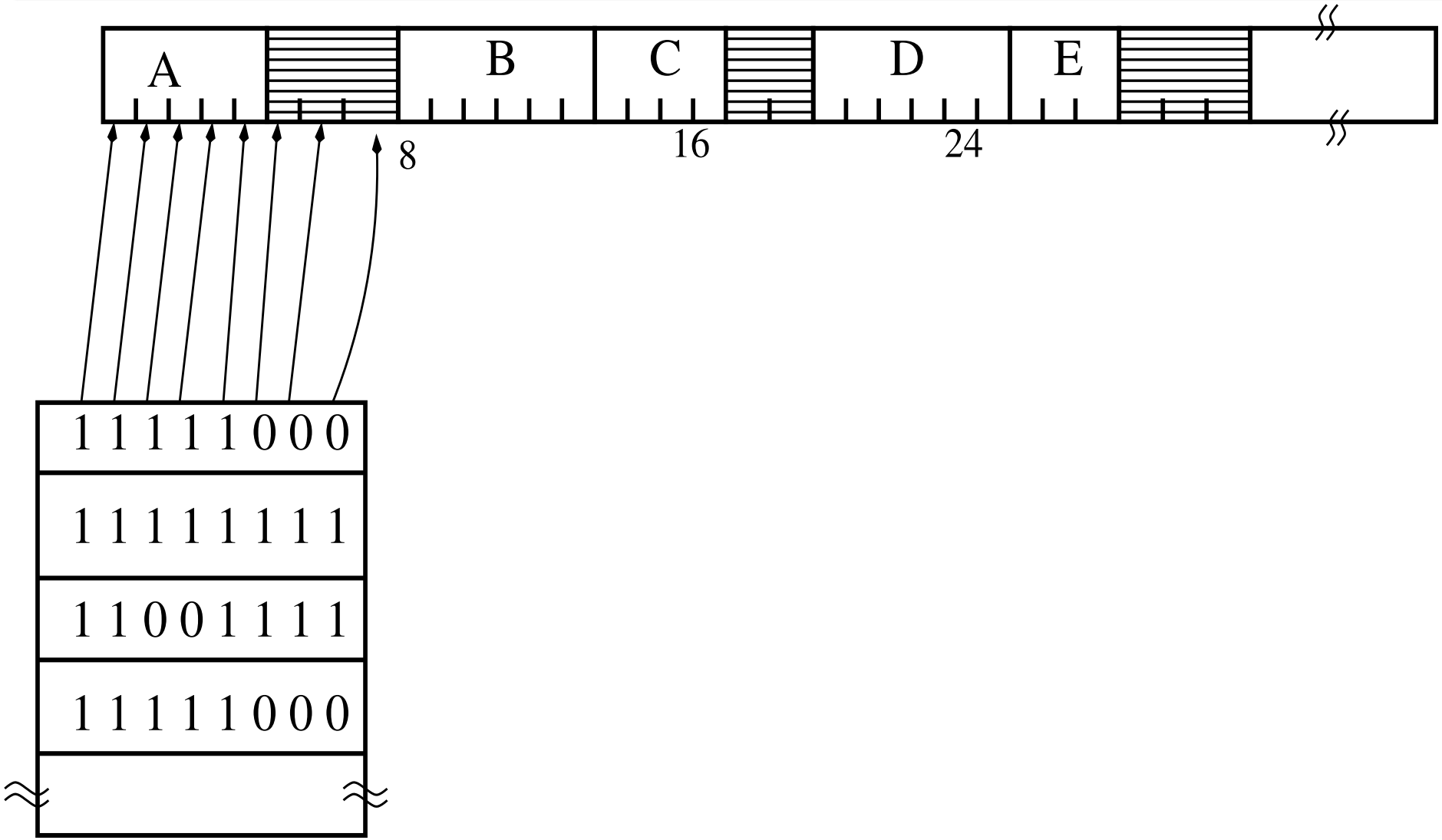
La fragmentation

Gestion par tables de bits

- Mémoire divisée en unités d'allocation dont la taille peut varier de qq mots à plusieurs Kbytes.
- À chaque unité est associé un bit dans un *bitmap*
  - 0 $\longleftrightarrow$ unité libre, 1 $\longleftrightarrow$ unité occupée.

Bit maps

Exemple :



Bit maps

Comment choisir l'unité d'allocation ?

- Plus l'unité est petite, plus le tableau de bits est important,
- Quand l'unité d'allocation est grande, le tableau de bits est petit, mais peut mener à un gaspillage de mémoire dans la dernière unité allouée à un processus si la taille n'est pas un multiple de l'unité d'allocation.

Avantage et inconvénient :

- Moyen simple de garder trace des mots mémoire dans une quantité fixe de mémoire
- Lorsqu'un processus de k unités est chargé en mémoire, la MMU doit parcourir la bit map pour trouver une suite de k bits consécutifs de valeur 0. Peut être lent

La fragmentation

Gestion par listes chaînées

Liste chaînée des segments libres et occupés. Un élément de la liste est composé de :

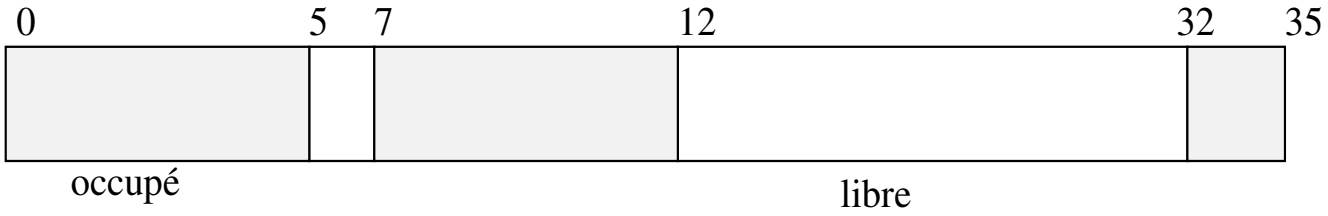
1. l'état libre ou occupé du segment
2. l'adresse de début du segment
3. la longueur du segment
4. le pointeur sur l'élément suivant de la liste.

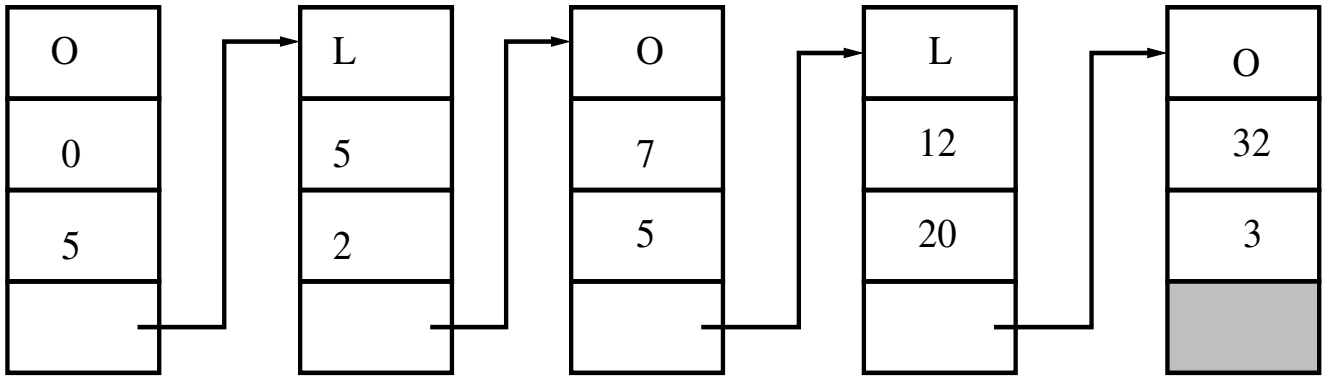
Les listes peuvent être organisées en ordre :

- de libération des zones,
- $\nearrow$ des tailles,
- $\searrow$ des tailles,
- des adresses de zones.

Gestion par listes chaînées

Exemple

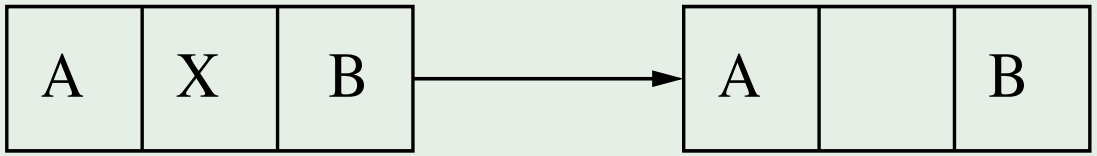




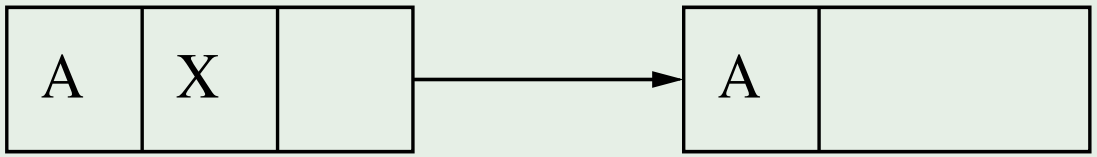
Libération d'espace

4 cas possibles quand on libère l'espace occupé par `X` :

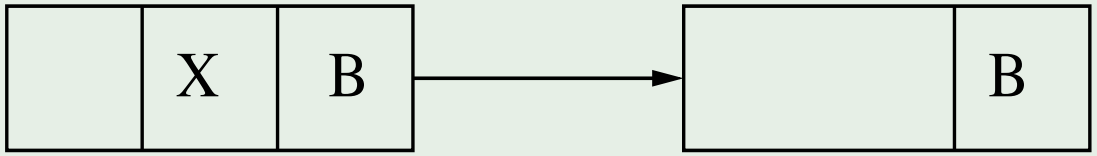
1. modifier l'indicateur de l'état du segment



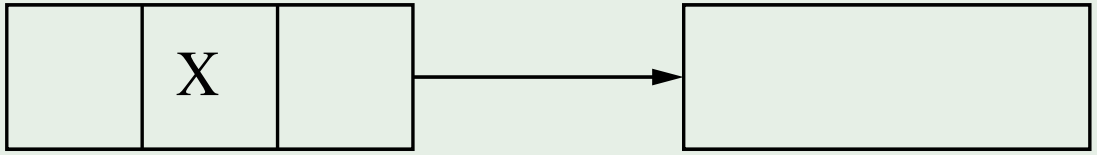
2. fusionner les 2 segments de droite



3. fusionner les 2 segments de gauche



4. fusionner les 3 segments



Allocation d'espace

4 stratégies d'allocation

1. 1ère zone libre (*first fit*)
   On cherche le 1er bloc libre de la liste qui peut contenir le processus à charger.
   La zone est scindée en 2 parties : la 1ère contient le processus et la 2ème est l'espace inutilisé (s'il n'occupe pas toute la zone).
   Fragmentation externe
2. zone libre suivante
   identique au first fit. Mémorise la position de l'espace libre trouvé. La recherche suivante commencera à partir de cette position.

Allocation d'espace

Stratégies d'allocation

3. meilleur ajustement (*best fit*)
   On cherche dans toute la liste la plus petite zone libre qui convient.
   On évite de fractionner une grande zone.
   - Plus lent que (1) 👎
   - Tendance à remplir la mémoire avec de petites zones libres inutilisables 👎
   - Plus rapide si les zones libres sont triées par taille.
4. plus grand résidu ou pire ajustement (*worst fit*)
   prendre toujours la plus grande zone, pour que la restante soit la plus grande possible.
   Simulée, elle ne donne pas de bons résultats. 👎

Allocation d'espace

Une variante : utiliser 2 listes séparées

- liste des blocs libres
- liste des blocs occupés.
  - Complexité plus grande : déplacements d'éléments d'une liste à l'autre.

L'algorithme d'allocation rapide utilise des listes séparées pour les tailles les plus courantes (zones de 4K, 8K, etc...)

- Recherche plus rapide.
- Libération : il faut examiner les voisins pour une fusion possible. Évite une fragmentation en petites zones inutilisables.

Allocation d'espace

Buddy System

ou gestion par subdivision.

On mémorise une liste de blocs libres de taille 1, 2, 4, . . . , 2^n octets jusqu'à la taille maximale de la mémoire.

- Initialement, un seul bloc libre.
- Allocation : si pas de bloc libre de taille S_i, on divise un bloc de taille S_{i+1} en 2 blocs de taille S_i. etc. . .
- Libération : on libère un bloc de taille S_i, si le bloc compagnon est libre, on reconstitue le bloc de taille S_{i+1}.

Relation de récurrence sur les tailles : $S_{i+1} = 2 * S_i$

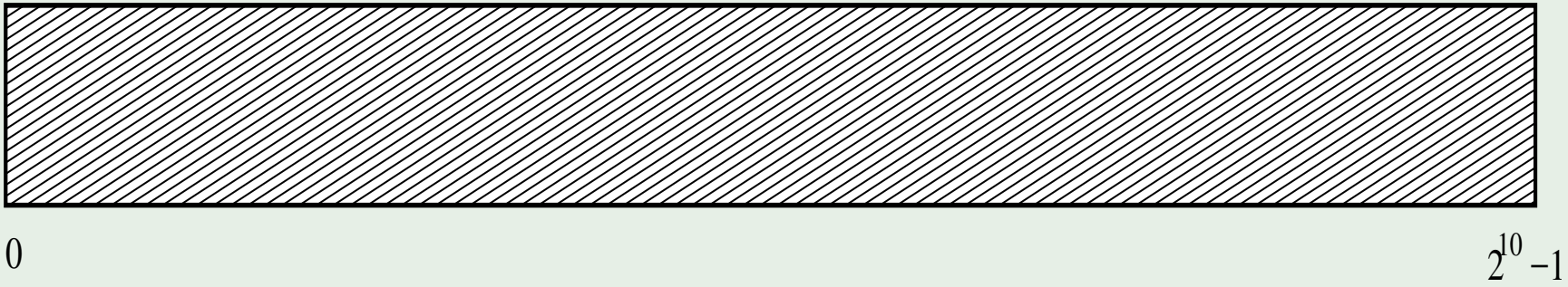
Buddy System

Exemple

Hypothèses :

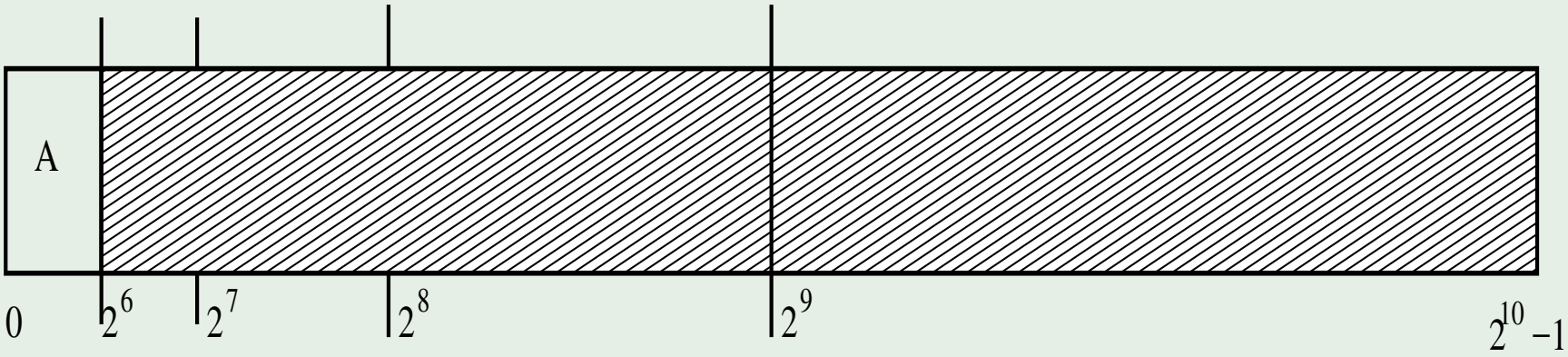
- une mémoire de 1024 Ko,
- la plus petite taille de bloc mémoire qu'on peut allouer est de 64 Ko,

Situation initiale :

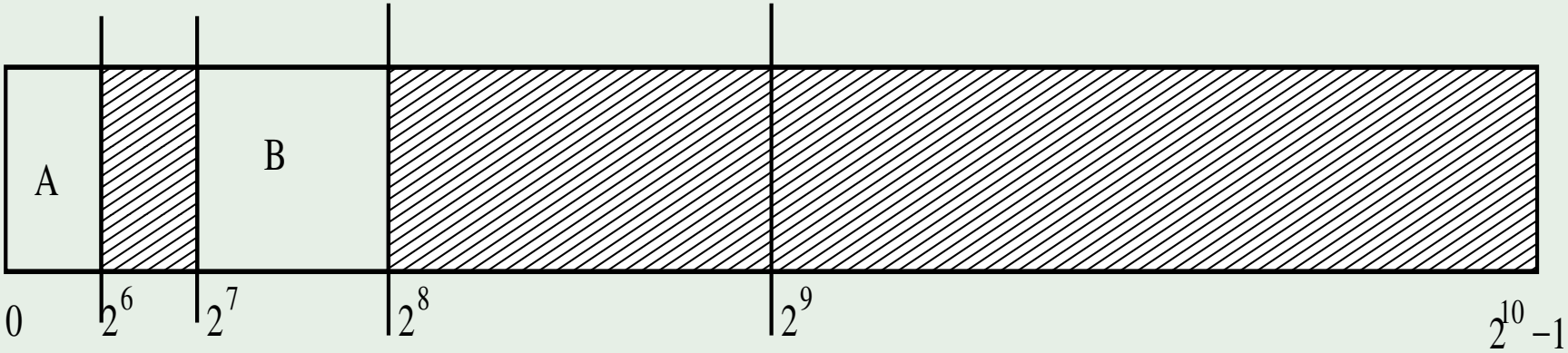


Exemple

Le programme A demande une mémoire de 34 Ko :
$32 = 2^5 < 34 < 2^6 = 64$

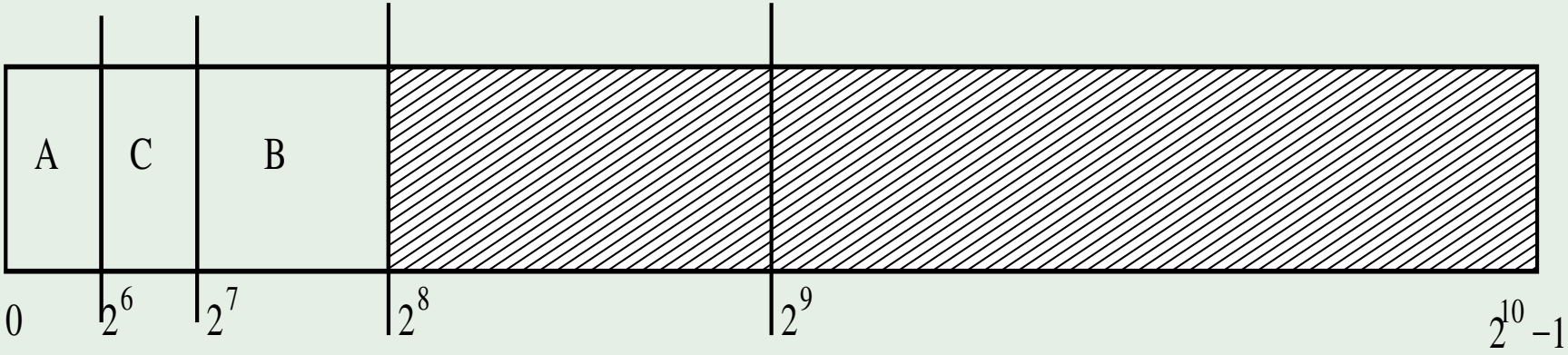


Le programme B demande une mémoire de 66 Ko :
$64 = 2^6 < 66 < 2^7 = 128$

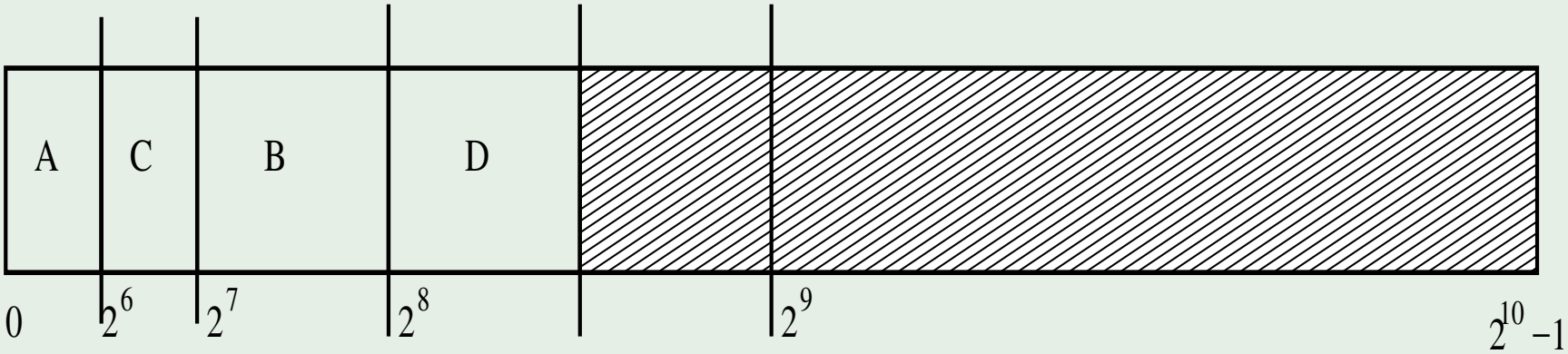


Exemple

Le programme C demande une mémoire de 35 Ko :
$32 = 2^5 < 35 < 2^6 = 64$

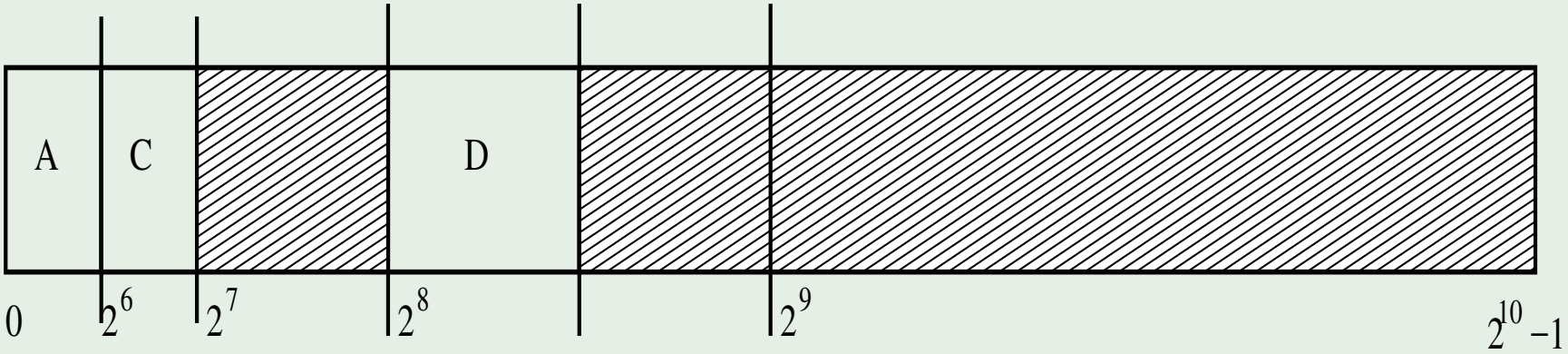


Le programme D demande une mémoire de 67 Ko :
$64 = 2^6 < 67 < 2^7 = 128$

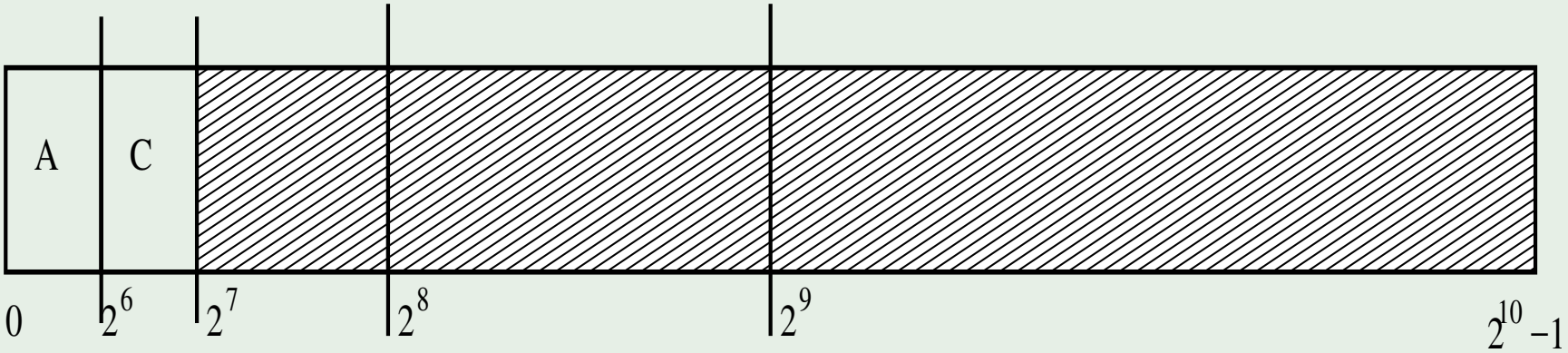


Exemple

Le programme B libère sa mémoire :

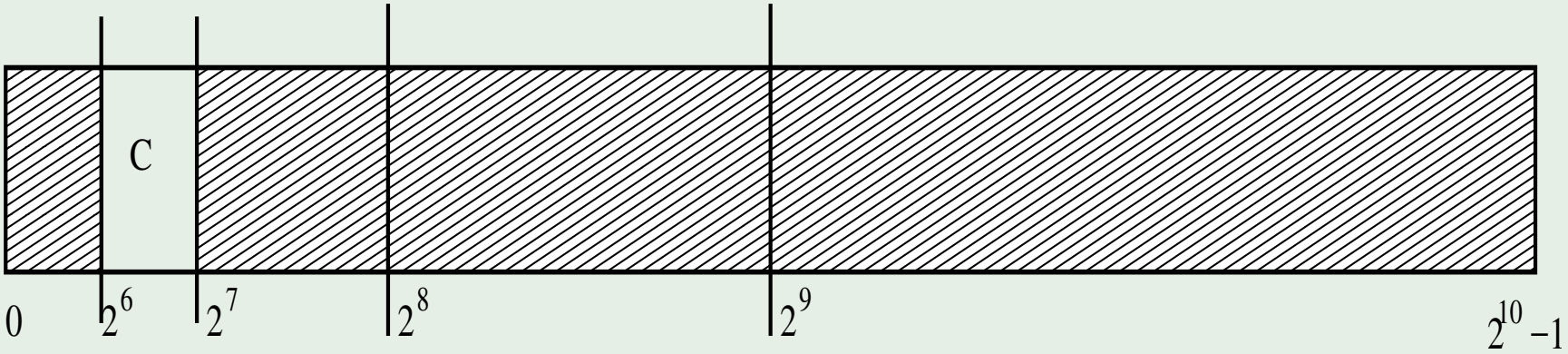


Le programme D libère sa mémoire :

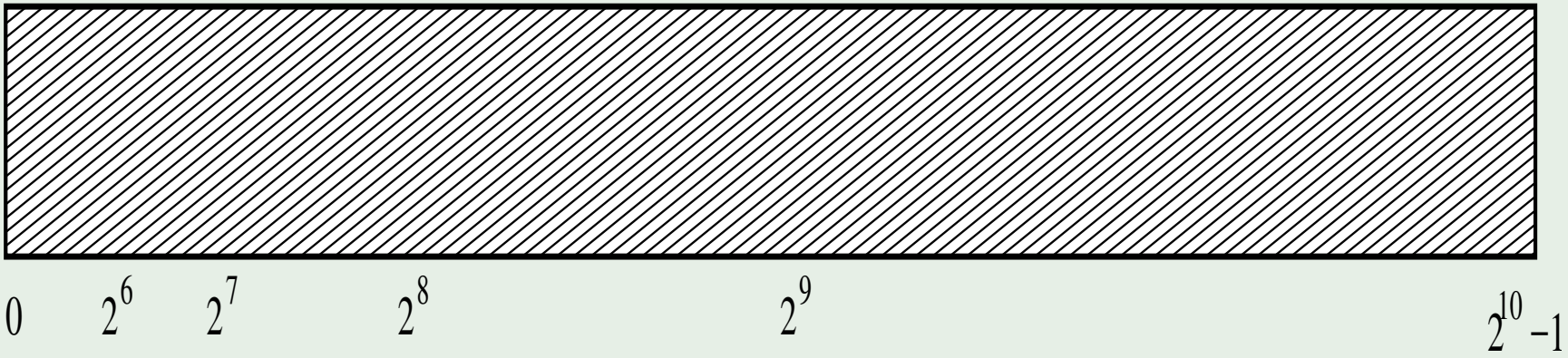


Exemple

Le programme A libère sa mémoire :



Le programme C libère sa mémoire :



Buddy System

- Algorithme rapide
- Inefficace. Arrondir les tailles à une puissance de 2 $\rightsquigarrow$ fragmentation interne.
- utilisé par Linux.

Une variante :

Plutôt que le système binaire, système fondé sur la suite de Fibonacci.

Relation de récurrence sur les tailles : $S_{i+1} = S_i + S_{i-1}$
Taille des zones libres : 1, 2, 3, 5, 8, 13, ...

Gestion de la mémoire

Le va-et-vient

fait l'hypothèse que le programme est intégralement présent dans la mémoire centrale.

Un inconvénient majeur : ne permet pas aux programmes de dépasser la taille de la mémoire physique.

Gestion de la mémoire

La mémoire virtuelle

Technique de gestion de la mémoire qui permet de ne garder en mémoire qu'une partie d'un processus.
En réalité,

- la totalité des instructions de tous les programmes peut ne pas être simultanément en MC ⇒ l'intégralité de tous les programmes n'est pas présente en mémoire en même temps.
- partage du processeur entre plusieurs programmes ⇒ partage de la MC entre des parties de programmes présentes simultanément en MC.

La mémoire virtuelle

répond à 2 problèmes

- la mémoire vue par le programme est physiquement constituée de fragments disjoints ⇒ plus de fragmentation, 
- observe le principe de localité de traitements : à tout instant, le programme a besoin de beaucoup moins de mémoire qu'il ne lui en faut au total ⇒ le contenu de la mémoire inutile à cet instant peut être stocké provisoirement ailleurs !

La mémoire virtuelle

Adresse virtuelle

- ne dépend que de l'espace d'adressage du programme,
- ne tient pas compte de l'adressage physique de la mémoire centrale.
- seulement au moment de l'exécution d'une instruction, et non pas au moment de la compilation ou du chargement, que la correspondance *adresse virtuelle* $\longleftrightarrow$ *adresse physique* est établie.

La mémoire virtuelle

2 techniques

1. par pagination : la plus utilisée.
   processeurs RISQ ex : SUN
2. par segmentation.
   Famille x86.

La mémoire virtuelle

La mémoire virtuelle par pagination

- L'espace adresse logique d'un processus n'est pas contigu.
- La mémoire physique est découpée en blocs de taille fixe : cadres de pages (frames)
  Taille = puissance de 2, entre 512 bytes et 8192 bytes.
- La mémoire logique est subdivisée en blocs de la même taille : pages

$$\text{taille pages} = \text{taille cadres}$$

- Il faut savoir quels cadres sont alloués, quels cadres sont libres.
  Information contenue dans la table de cadres.
- Table de pages : contient l'adresse de la page en mémoire physique.

La mémoire virtuelle

Exemple

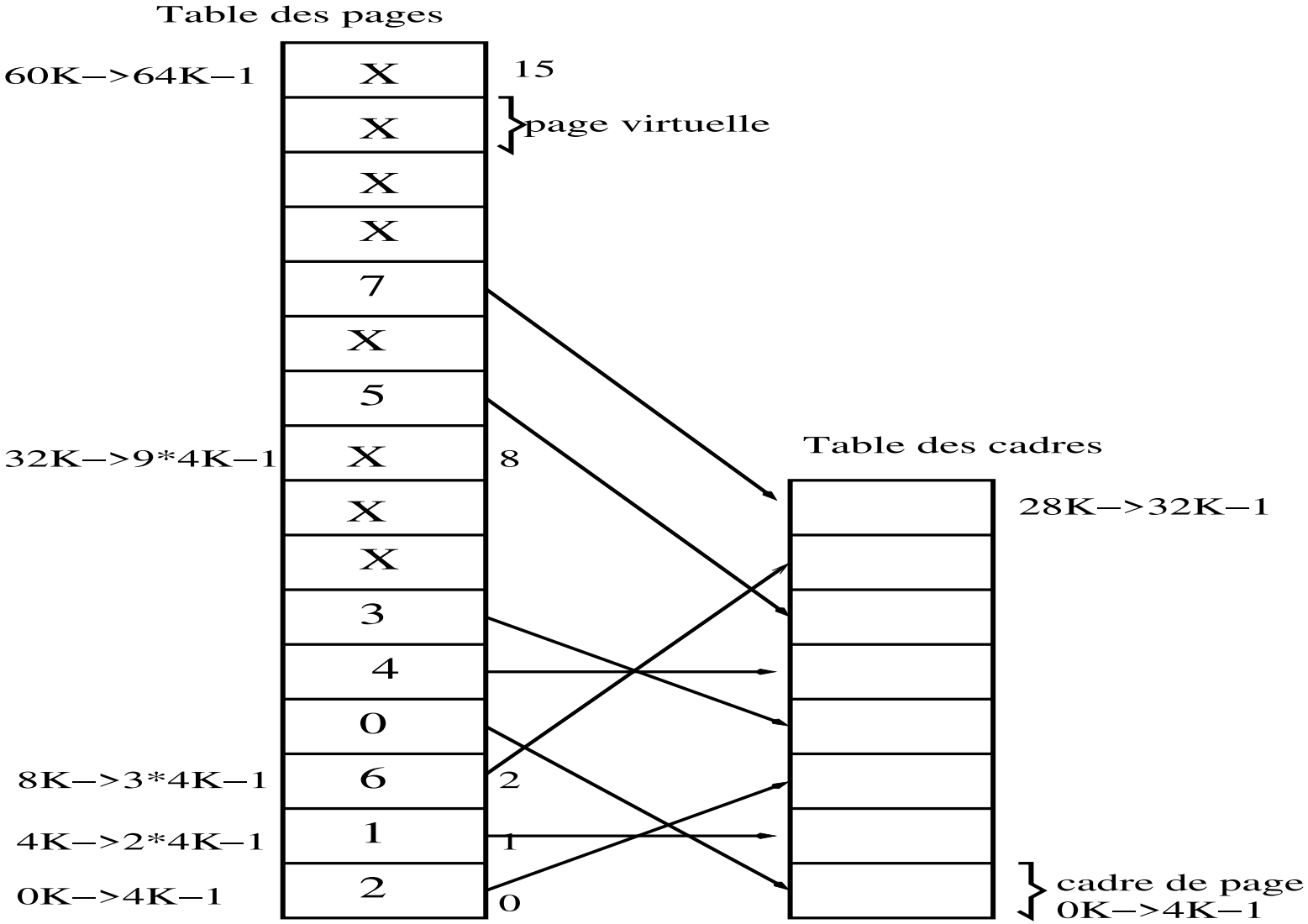
- Adresses virtuelles : un ordinateur peut produire des adresses sur 16 bits avec des valeurs entre 0 et 64 KBytes.
  $\Rightarrow$ l'espace d'adressage virtuel est $[0..2^{16-1}]$.
- L'ordinateur a seulement 32 KBytes de mémoire physique (2^{15} octets).
- La taille des pages et des cadres est de 4 KBytes (2^{12} octets).

Taille de la table des pages $= \frac{2^{16}}{2^{12}} = 2^4 = 16$ entrées.

Combien de cadres de pages ? $\frac{2^{15}}{2^{12}} = 2^3 =$ *8 cadres de pages.*

Pagination

Les adresses multiples de 4096 $(= 2^{12} = 4K)$ sont des frontières de pages :



La mémoire virtuelle par pagination

3 situations possibles pour une page :

1. elle est aussi présente en mémoire physique $\rightsquigarrow$ la table indique son adresse physique. La zone mémoire où est stockée la page est le cadre de page
2. elle correspond à une adresse qui n'a encore jamais été invoquée par le programme $\rightsquigarrow$ elle reste virtuelle
3. cette page a été utilisée à un moment donné, mais l'exécution du programme ne nécessitait pas son usage à cet instant, et si pas de place en MC, elle peut avoir été placée sur disque en mémoire auxiliaire et la table indique son emplacement.

La mémoire virtuelle par pagination

Pagination à la demande

- Transfert d'une page en mémoire seulement quand c'est nécessaire,
- support matériel : possibilité de distinguer entre les pages qui sont en mémoire et celles qui sont sur le disque
  - bit présence/absence (1 : page en mémoire, 0 : sur disque)
- le programme ne connaît que l'adresse virtuelle ⇒ traduction à la volée de l'adresse virtuelle en adresse réelle
  - circuit logique dédié DAT : Dynamic Address Translation.

La mémoire virtuelle par pagination

Défaut de page

Référence à une page marquée absente, $\rightsquigarrow$ déroutement (*page default* SE)

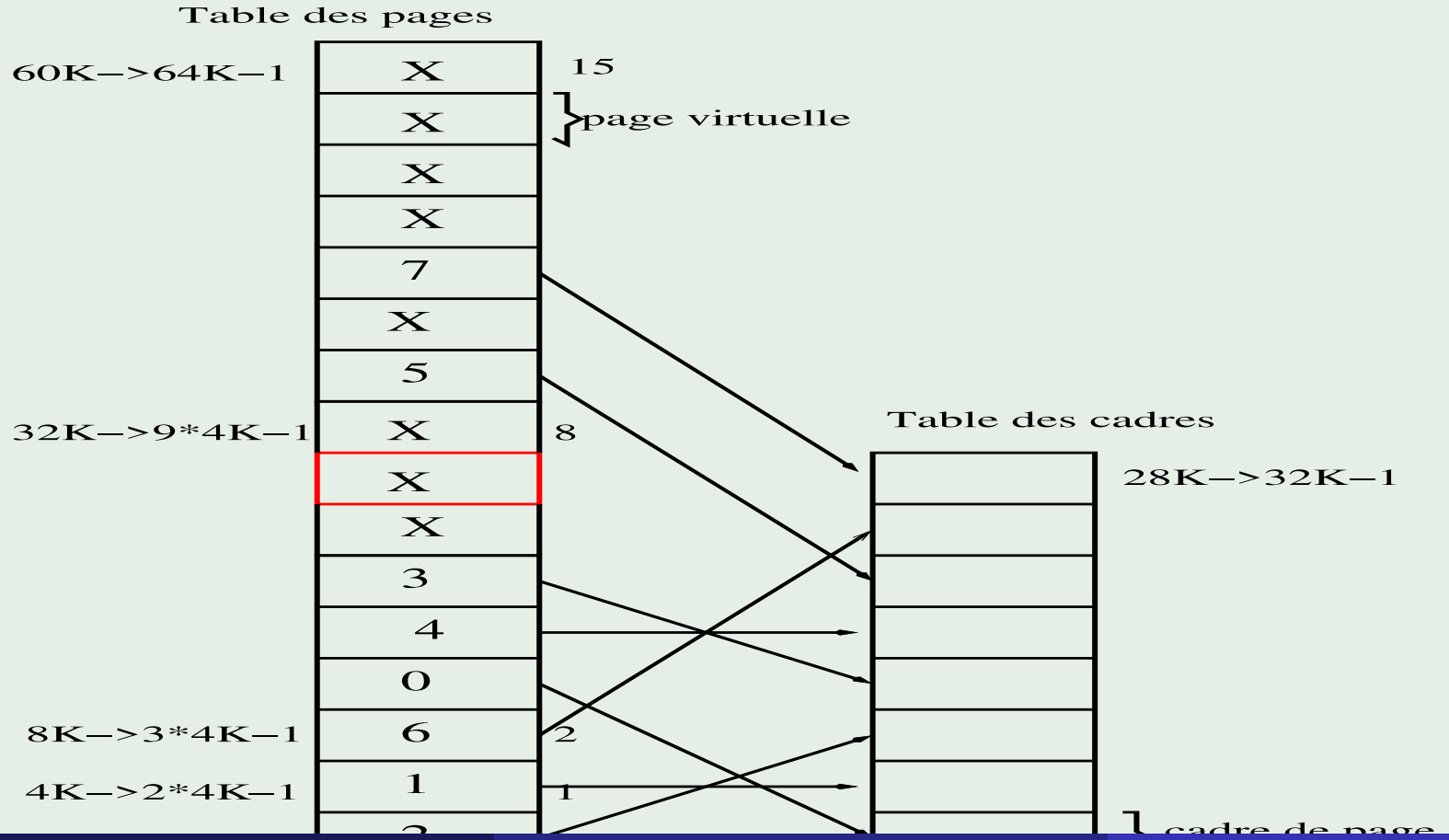
- Trouver un cadre de page libre,
- swap de la page dans le cadre,
- modifier la table des cadres et la table des pages pour indiquer que la page est maintenant en mémoire.
- Redémarrer l'instruction interrompue par le déroutement : le processus peut accéder à la page comme si elle avait toujours été en mémoire.

La mémoire virtuelle par pagination

Exemple

Que se passe-t-il si le programme tente d'accéder à une page non présente, avec l'instruction `MOV REG, 32780`

$32780 = 8 * 4K + 12 \Rightarrow$ correspond à l'octet 12 de la page virtuelle n° 7



Exemple (suite)

Exemple (suite)

- La MMU remarque que la page est absente et provoque un déroutement
- le SE décide de décharger le cadre de page 1, [a] et il charge la page virtuelle n° 7 à l'adresse physique 4K
  - Marquer la page virtuelle 1 absente, (après l'avoir copiée en mémoire auxiliaire de pages)
  - changer le bit de présence/absence de la page n° 7
  - mettre l'adresse virtuelle 32768 en correspondance avec l'adresse 4096
    Conséquence : l'adresse virtuelle 32780 correspondra à l'adresse physique 4108

a. par une stratégie de remplacement de pages

La mémoire virtuelle par pagination

MMU

Module matériel comprenant le DAT qui établit la correspondance adresse virtuelle $\longleftrightarrow$ adresse physique.

Physiquement installé soit sur la puce processeur soit sur le bus d'adresse à la sortie du processeur.

La mémoire virtuelle par pagination

Exemple d'implantation

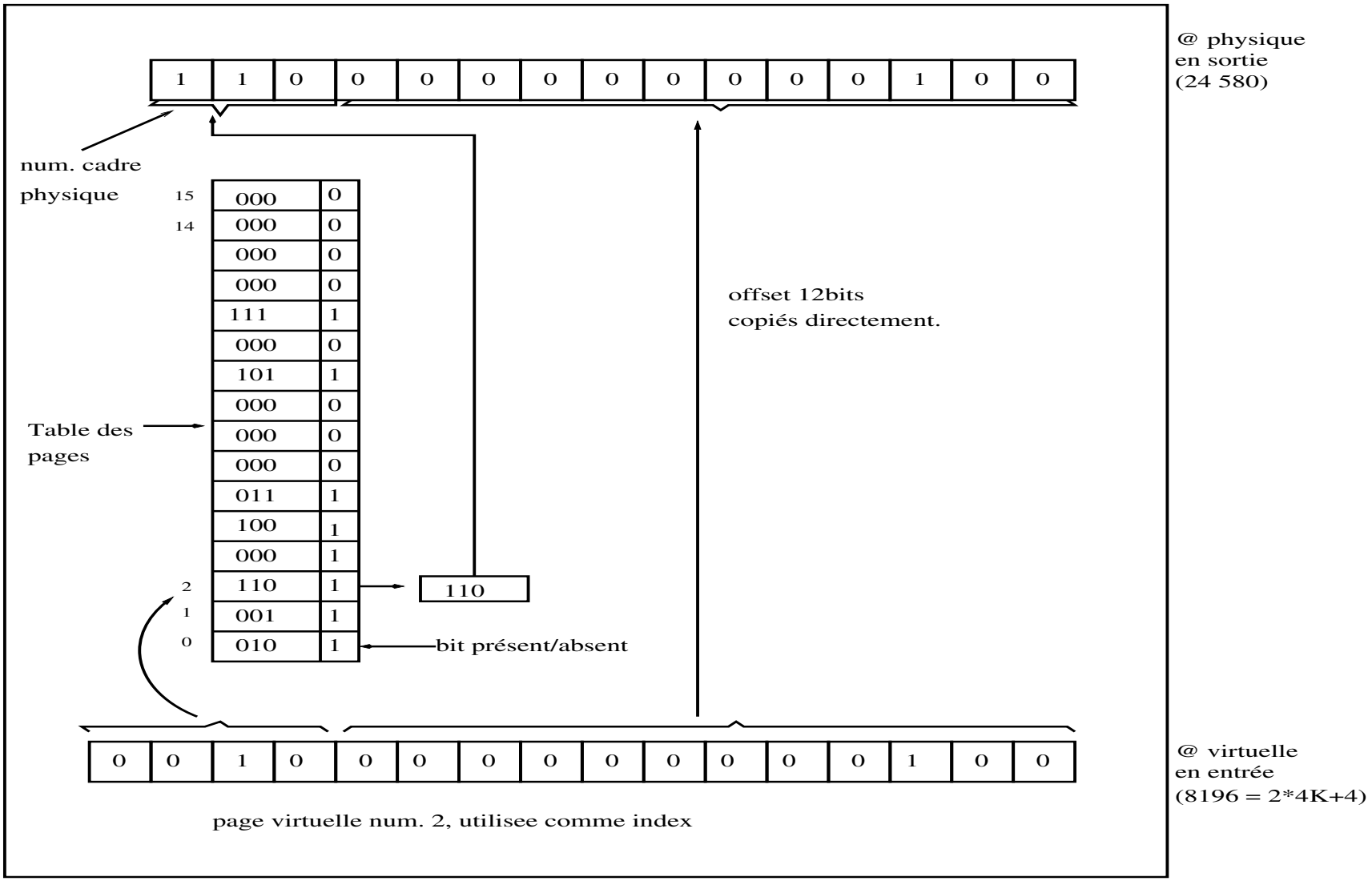
16 pages de 4 Koctets. L'adresse virtuelle de 16 bits est divisée en deux parties :

- un numéro de page sur 4 bits (pour distinguer les 16 pages)
- un décalage (*offset*) sur 12 bits

8 cadres de pages $\Rightarrow$

- 3 bits pour le numéro de cadre
- 12 bits pour le déplacement.

Exemple d'implantation



La mémoire virtuelle par pagination

Adresse virtuelle

- le numéro de page est utilisé comme un index dans la table des pages,
- si le bit de présence/absence est à 0, un déroutement vers le SE est mis en place,
- si le bit de présence/absence est à 1, le numéro de cadre de page trouvé dans la table des pages est copié dans le registre de sortie. On y ajoute les 12 bits de décalage.
- L'adresse physique est formée de ces deux parties.

La table de pages

- L'adresse générée par la CPU est divisée en 2 parties :
  - numéro de page **p**
    indice de la table de pages contenant l'adresse de base de chaque page,
  - déplacement dans la page **d** : combiné avec l'adresse de base, définit l'adresse physique envoyée à l'unuté de mémoire.
- Rôle de la table de pages : faire correspondre des pages virtuelles à des cadres de pages.
- Mathématiquement, une table de pages est une fonction, qui prend en argument **p** et qui rend en résultat le numéro de cadre physique.

La table de pages

Deux contraintes :

- La table des pages doit être extrêmement grande :
  - les ordinateurs modernes utilisent des adresses virtuelles d'au moins 32 bits,
  - la table des pages a environ un million d'entrées !
  - chaque processus a besoin de sa propre table de pages.
- La correspondance doit être rapide.
  - pour chaque instruction, il est nécessaire de faire référence à la table de pages 1 fois, 2 fois, parfois plus...

Implantation de la table de pages

La plus simple :

- Table des pages dans la MC,
- *Page Table Base Register* (PTBR) : indique la table de pages,
- Chaque processus possède sa table de pages. Pour changer de table de pages, il suffit de modifier PTBR.
- Problème : temps requis pour charger la table de pages en totalité à chaque changement de processus.

Implantation de la table de pages

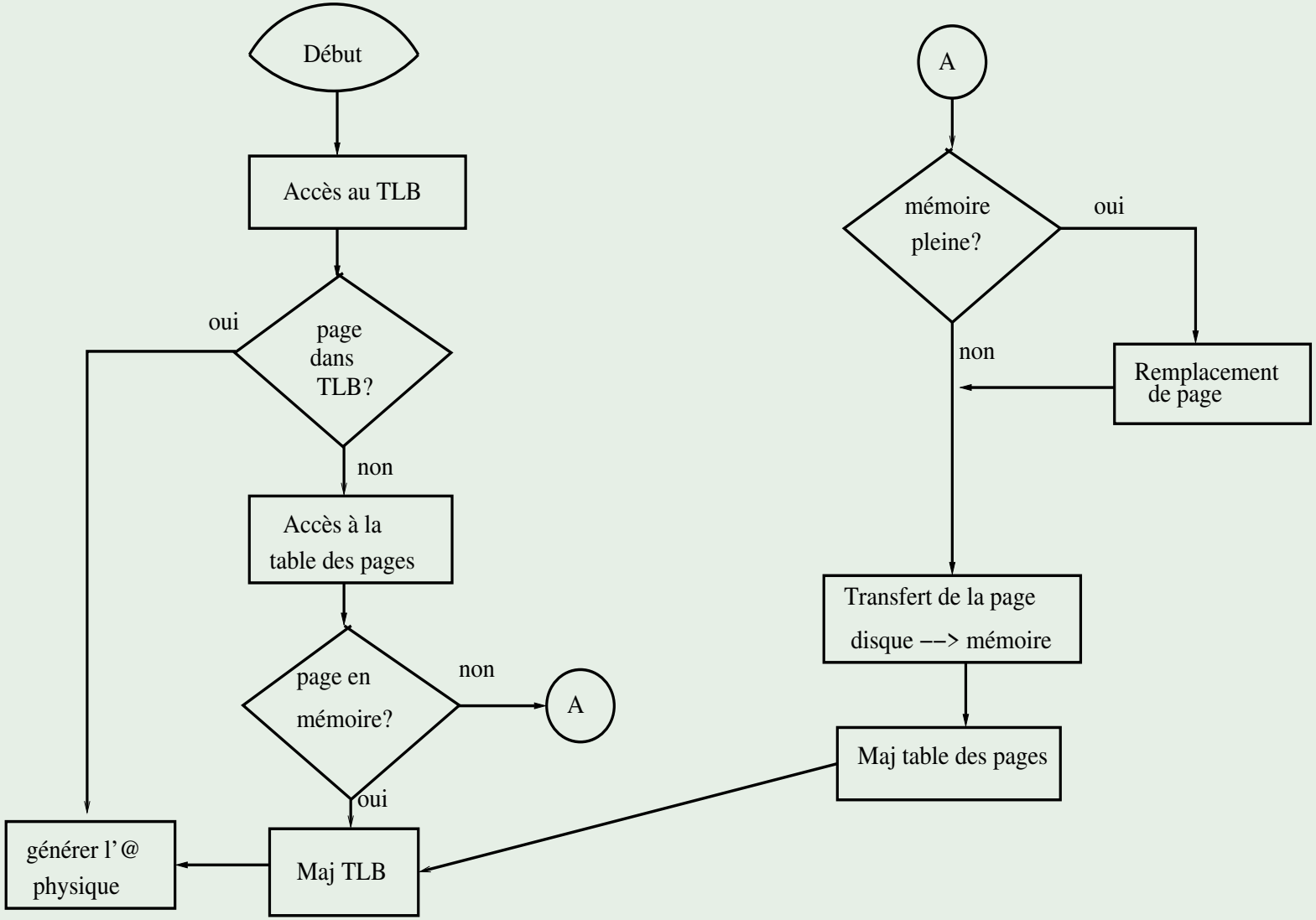
Solution standard :

- Utiliser les *registres associatifs* (ou *Translation Lookaside Buffer*) : petite mémoire cache matérielle spécifique à la consultation rapide
  - contiennent uniquement quelques entrées de la table (le résultat des traductions d'@ les plus récentes),

n° page virtuelle	n° cadre de page

  - recherche en parallèle
    - consulte le TLB pour voir si la page cherchée y figure
    - active le DAT pour parcourir la table des pages.
  - Si le TLB trouve la page, le DAT (+ lent) est interrompu. Sinon, il place son résultat dans le TLB.
  - Si le TLB est plein, le SE sélectionne une entrée devant être remplacée.

La mémoire virtuelle par pagination



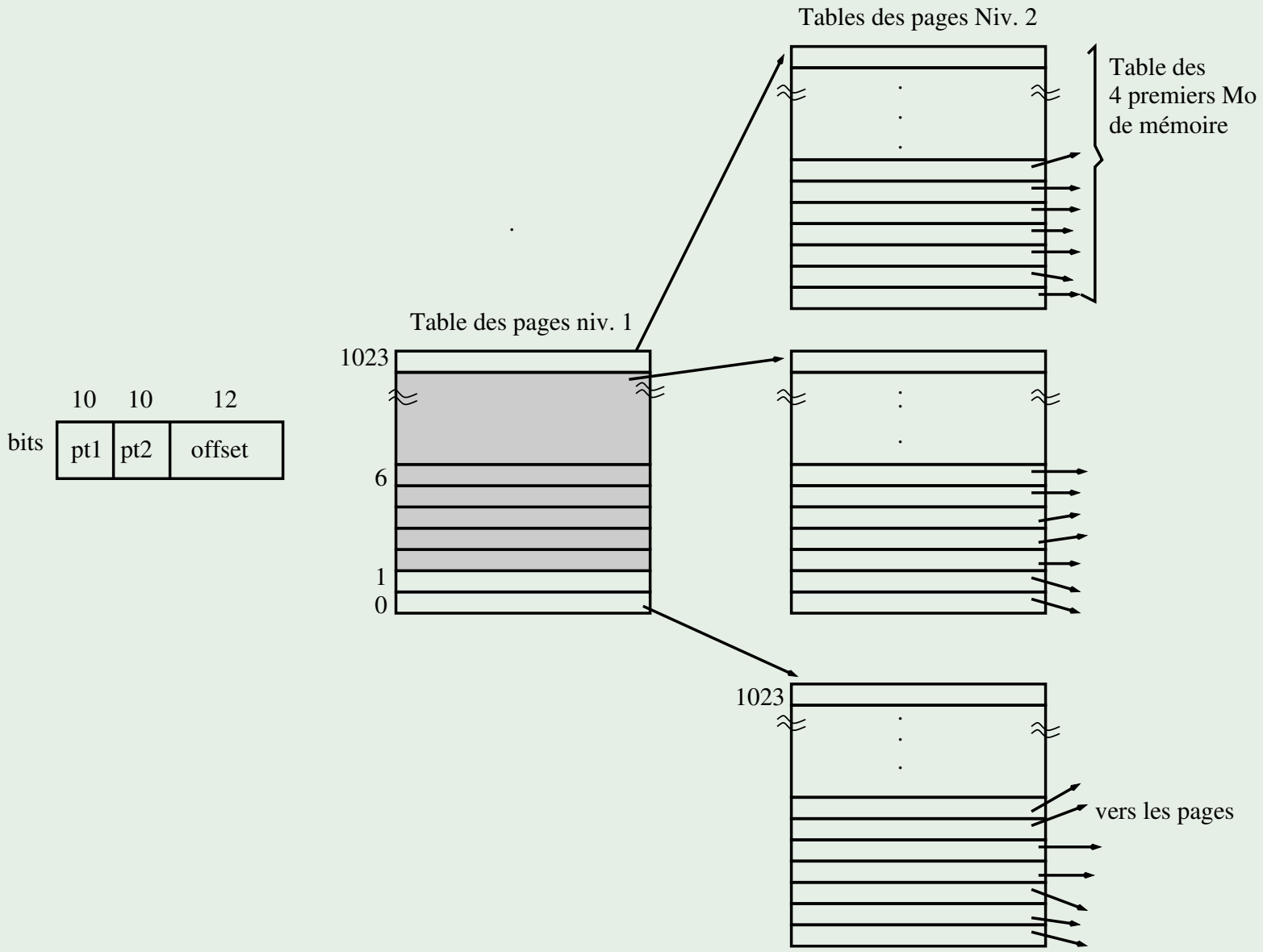
La mémoire virtuelle par pagination

Table des pages à plusieurs niveaux

On hiérarchise pour diminuer l'occupation mémoire.

- Performance : chaque niveau est stocké comme une table séparée en mémoire ⇒ la conversion d'une adresse logique en une adresse physique peut demander 4 accès mémoire.
- Mémoire cache : permet une performance raisonnable.

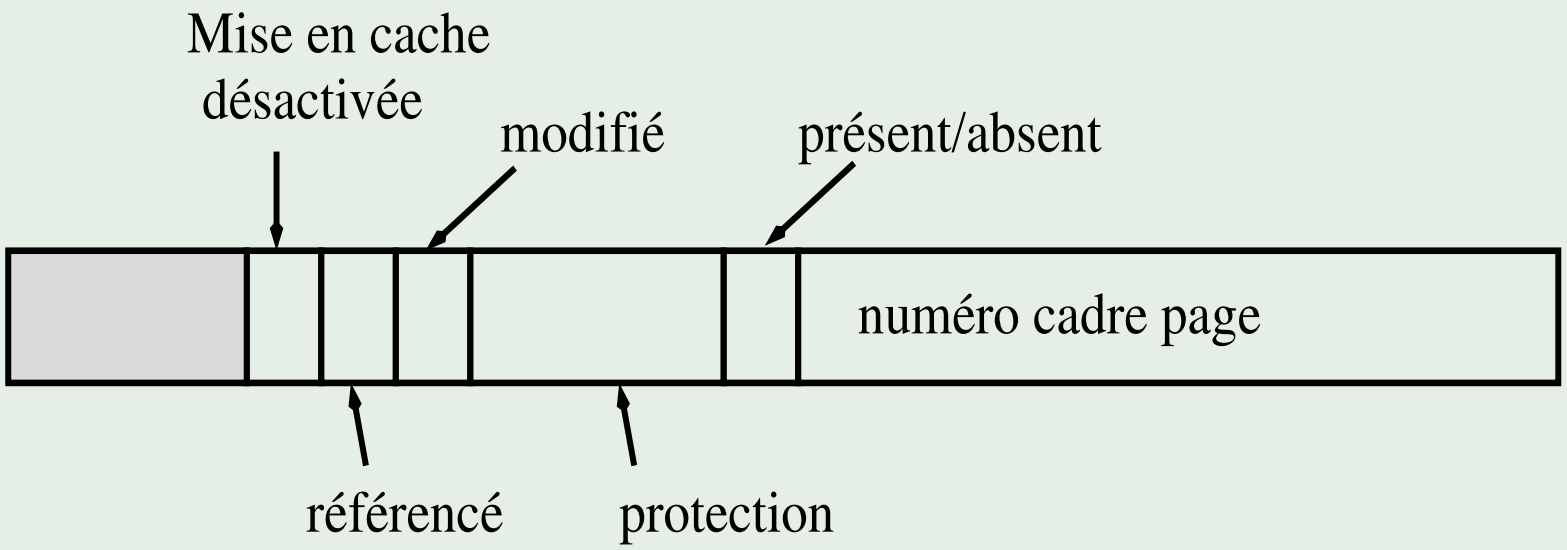
Exemple


Tables des pages Niv. 2
Table des
4 premiers Mo
de mémoire
Table des pages niv. 1
1023
6
1
0
10
10
12
bits
pt1
pt2
offset
1023
vers les pages

Exemple

- Adresse virtuelle sur 32 bits partitionnée en 2 champs de 10 bits (**pt1** et **pt2**) et un champ de 12 bits.
- *offset* de 12 bits $\Rightarrow$ pages de 4K Bytes.
- Total de $(2^{10})^2 = 2^{20}$ pages.
- Table de pages de niveau 1, avec 1024 entrées, correspondant au champ **pt1**.
- L'entrée localisée par l'indexation de la table de pages de niveau 1 produit l'adresse ou le numéro de cadre de page de la table de pages du niveau 2.

Structure d'une entrée de la table des pages



- La taille varie d'une machine à l'autre.
- Champ le plus important : numéro de cadre de page.
- Bit de présence/absence en mémoire.
- Bits de protection : précisent quelle sorte d'accès sont permis.
  Dans sa forme la plus simple, un seul bit : 0 = lecture/écriture, 1 = lecture seulement.
  Schéma plus sophistiqué : 3 bits, lecture, écriture, exécution.

Structure d'une entrée de la table des pages

- Bits modifiés et référencés : conservent une trace de l'utilisation de la page.
- *Dirty bit (modifié)* : quand une page est accédée en lecture, le bit est mis à 1. Lorsque le SE décide de récupérer un cadre de page :
  - si la page a été modifiée, elle doit être écrite sur le disque,
  - si non, on peut "l'écraser".
- Bit référencé : mis à 1 chaque fois qu'une page est consultée (en lecture ou en écriture). Aide le SE à choisir la page à évincer lors d'un défaut de page.
  Les pages qui ne sont pas en cours d'utilisation sont les meilleures candidates. Ce bit jour un rôle important dans plusieurs algorithmes de remplacement de page.
- Bit qui permet d'inhiber le cache pour une page.

La mémoire virtuelle par pagination

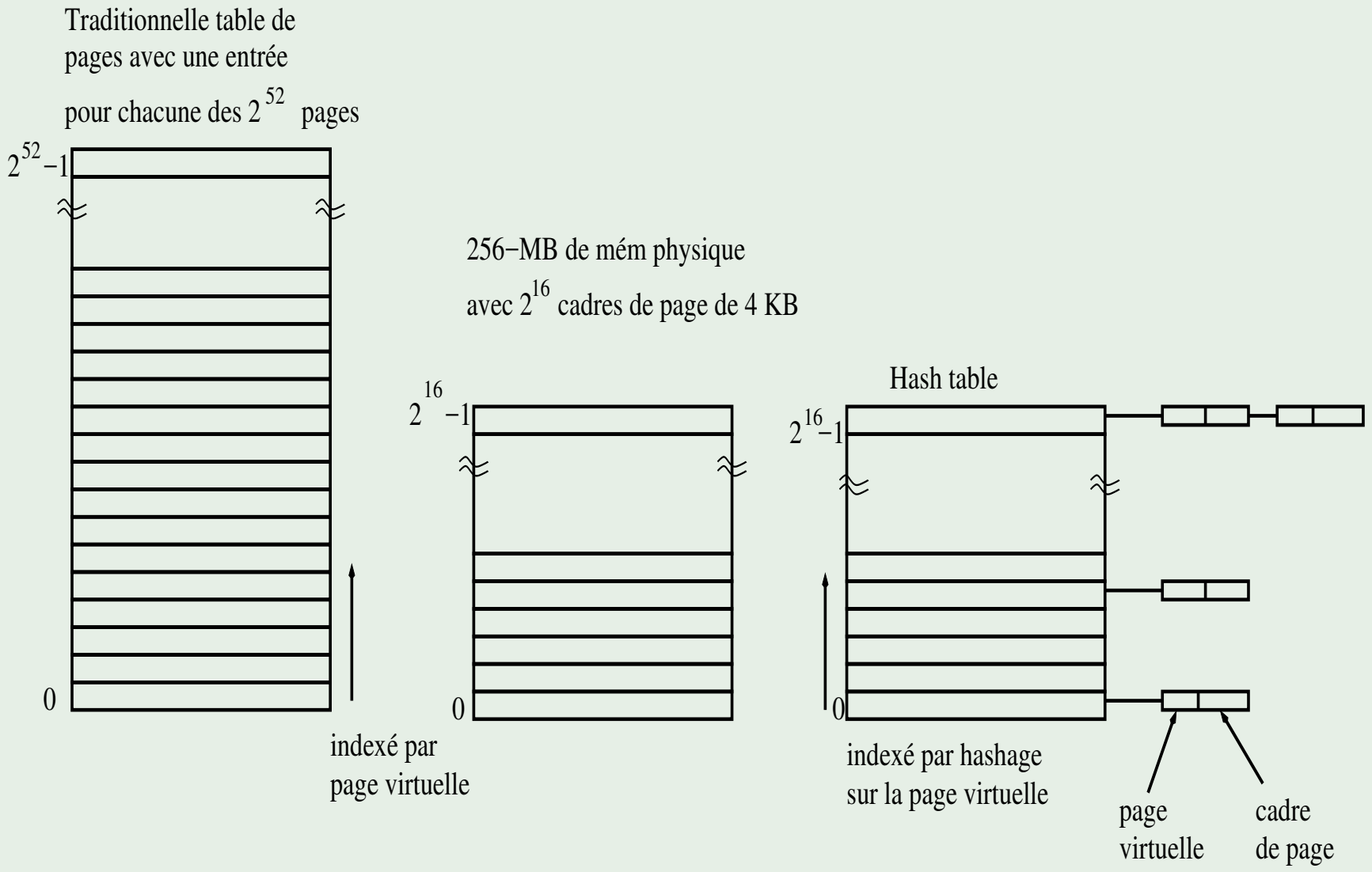
Table de pages inversée

Idée : indexer une table des cadres de pages où chaque cadre contient la référence de la page virtuelle qu'il contient.

- la référence comporte
  - le numéro de la page virtuelle,
  - l'identifiant du programme propriétaire de l'espace d'adresses.

Avantage : la table des cadres de pages est bcp plus petite
Inversion de la traduction d'adresse $\rightsquigarrow$ utiliser le TLB.
Les références non résolues (cas rares) sont traitées par le SE. L'utilisation d'une table de hachage améliore la performance de la recherche pour trouver l'indice de la table possédant le triplet (numéro processus, numéro page virtuelle, numéro page physique).

Table de pages inversée



La mémoire virtuelle par pagination

La pagination avancée

- partage de code entre programmes.
  Une page physique $\longleftrightarrow$ pages virtuelles de plusieurs processus.
  Intéressant pour
  - des données non modifiables : librairies partagées.
  - les techniques de processus légers (threads) : exécution de plusieurs parties de programme en pseudo-simultanéité.
- Pages verrouillées en mémoire.
  Interactions mémoires/périphériques (DMA).
- partage avec copie de la page si écriture.

La mémoire virtuelle par pagination

Pour résumer :

si on suppose qu'une adresse virtuelle (b, d) est sur 32 bits, avec n bits pour d :

- le nombre maximal de pages que peut contenir la mémoire virtuelle est de 2^{32-n}.
- la taille maximale d'une page est de 2^n.

La mémoire virtuelle

Par segmentation

Des segments de tailles variables adaptés à des régions particulières du programme,

- segments de taille variable, sans ordre entre eux.
- Un espace d'adressage logique est un ensemble de segments.
- Chaque segment possède un nom (numéro) et une longueur.
- Adresse logique = $\langle$numéro de segment, déplacement$\rangle$

La segmentation

Schéma d'exécution d'un programme

Donnees

(statiques)

Code

(instructions)

Pile :

contextes des

procedures en

cours

d'execution

Tas :

Autres donnees

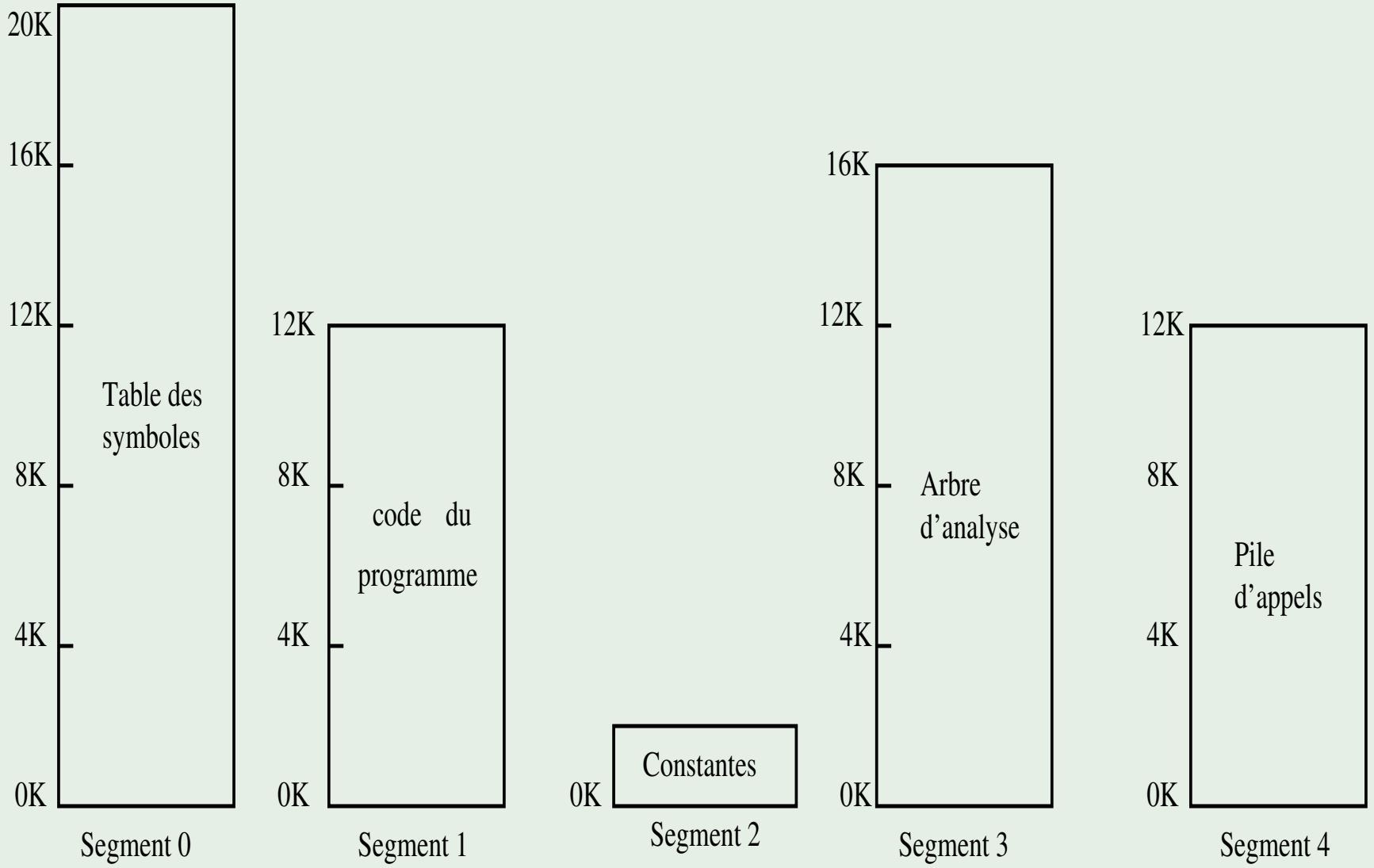
dynamiques

– chaines

– listes

– arbres

La segmentation



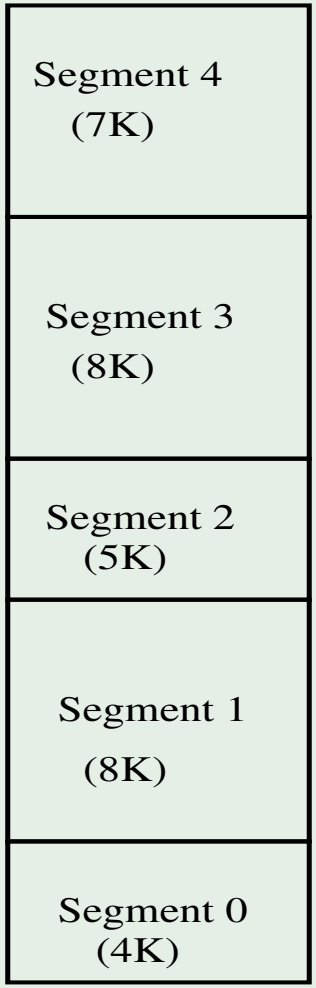
La segmentation

Table des segments

- transforme des adresses à deux dimensions (définies par l'utilisateur) en adresses à une dimension.
- Chaque adresse possède :
  - une **base** : contient l'adresse physique de début où le segment réside en mémoire,
  - une **limite** : spécifie la longueur du segment.

La segmentation

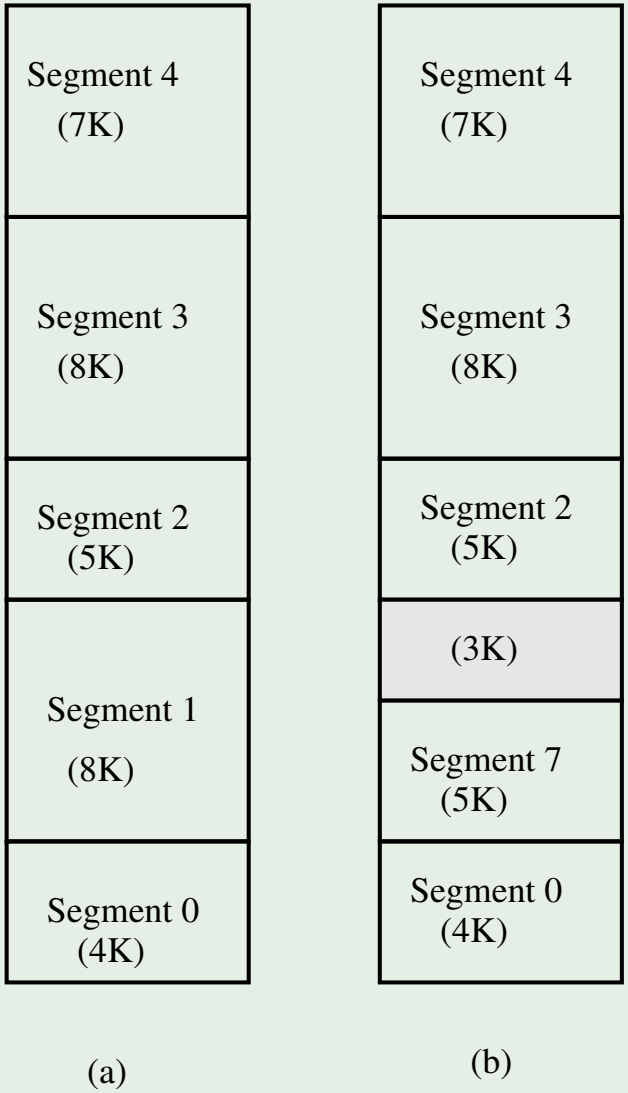
Segmentation pure



(a)

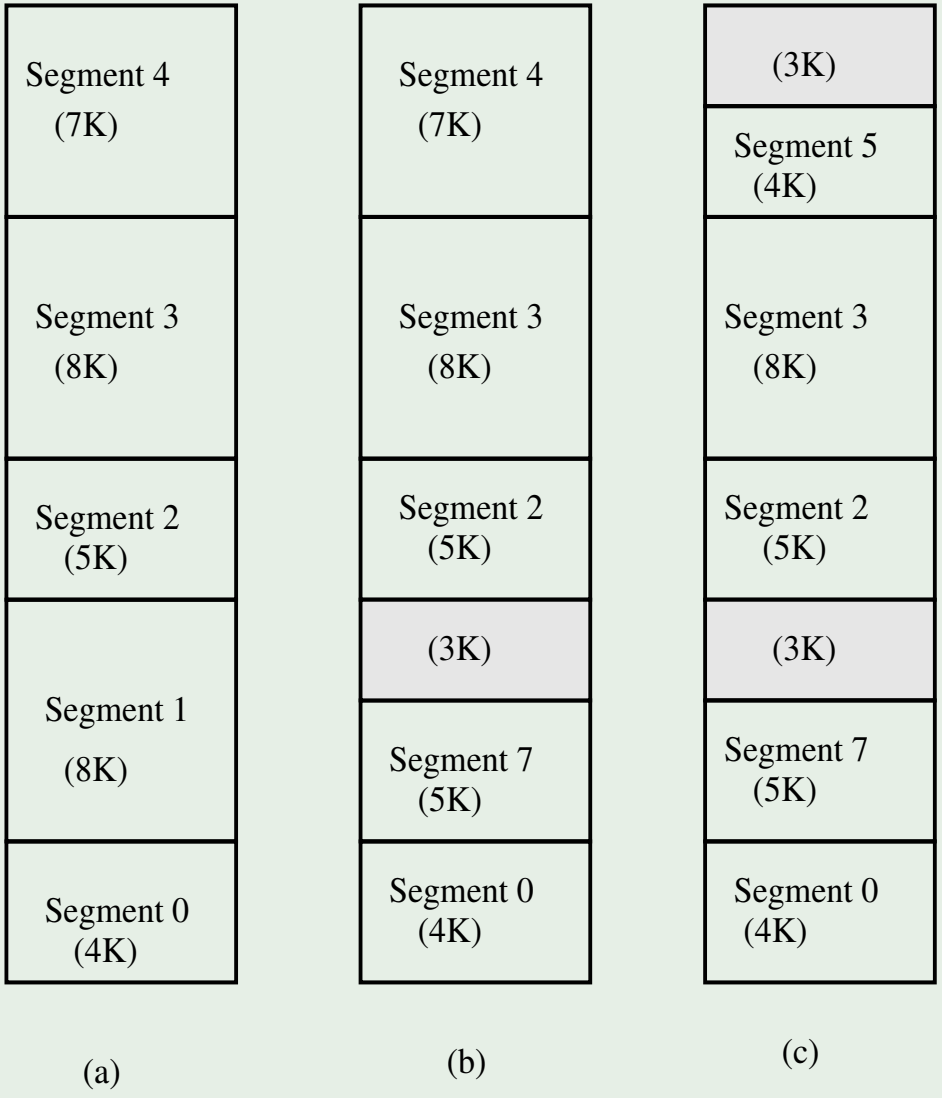
La segmentation

Segmentation pure



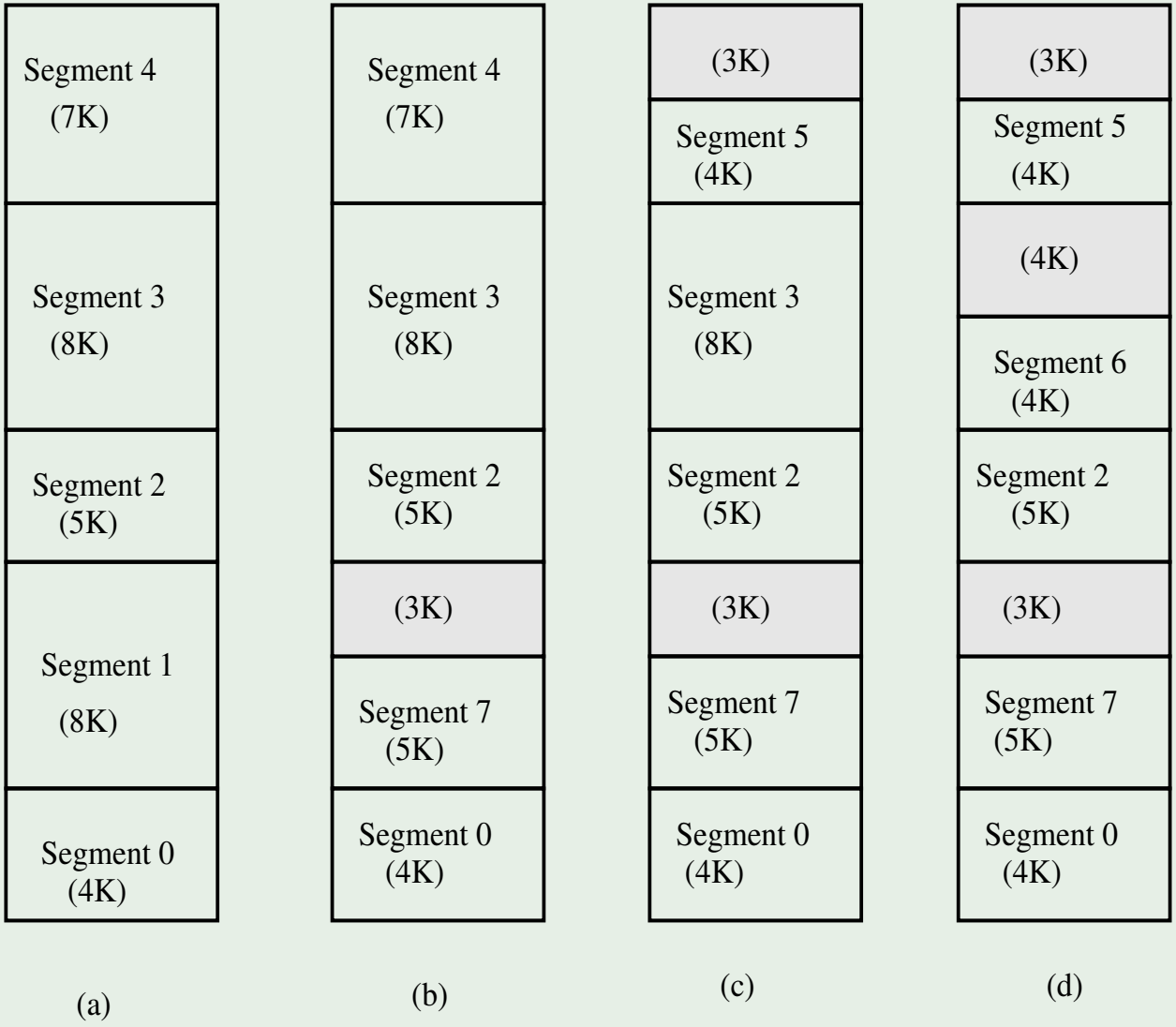
La segmentation

Segmentation pure



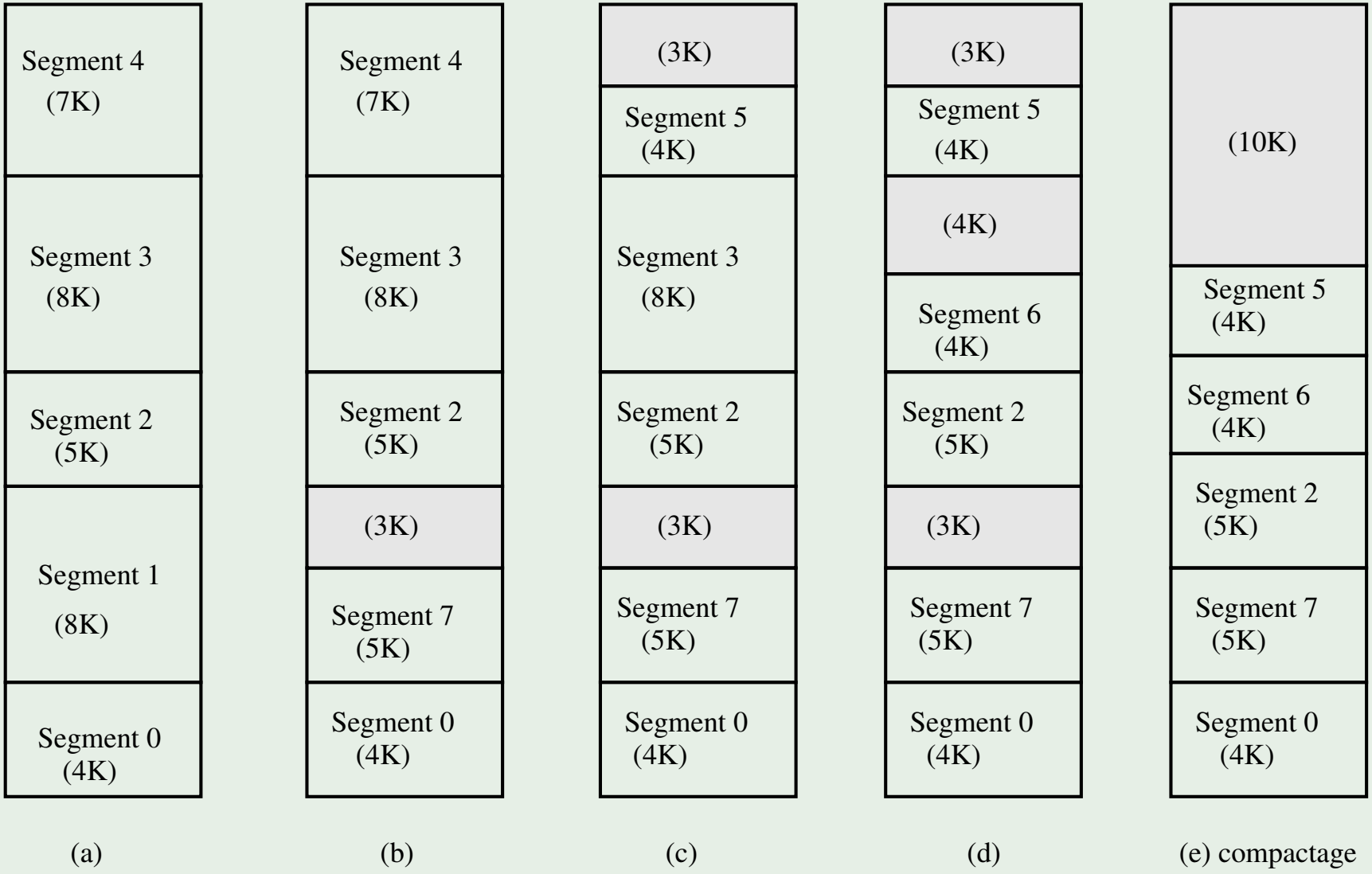
La segmentation

Segmentation pure



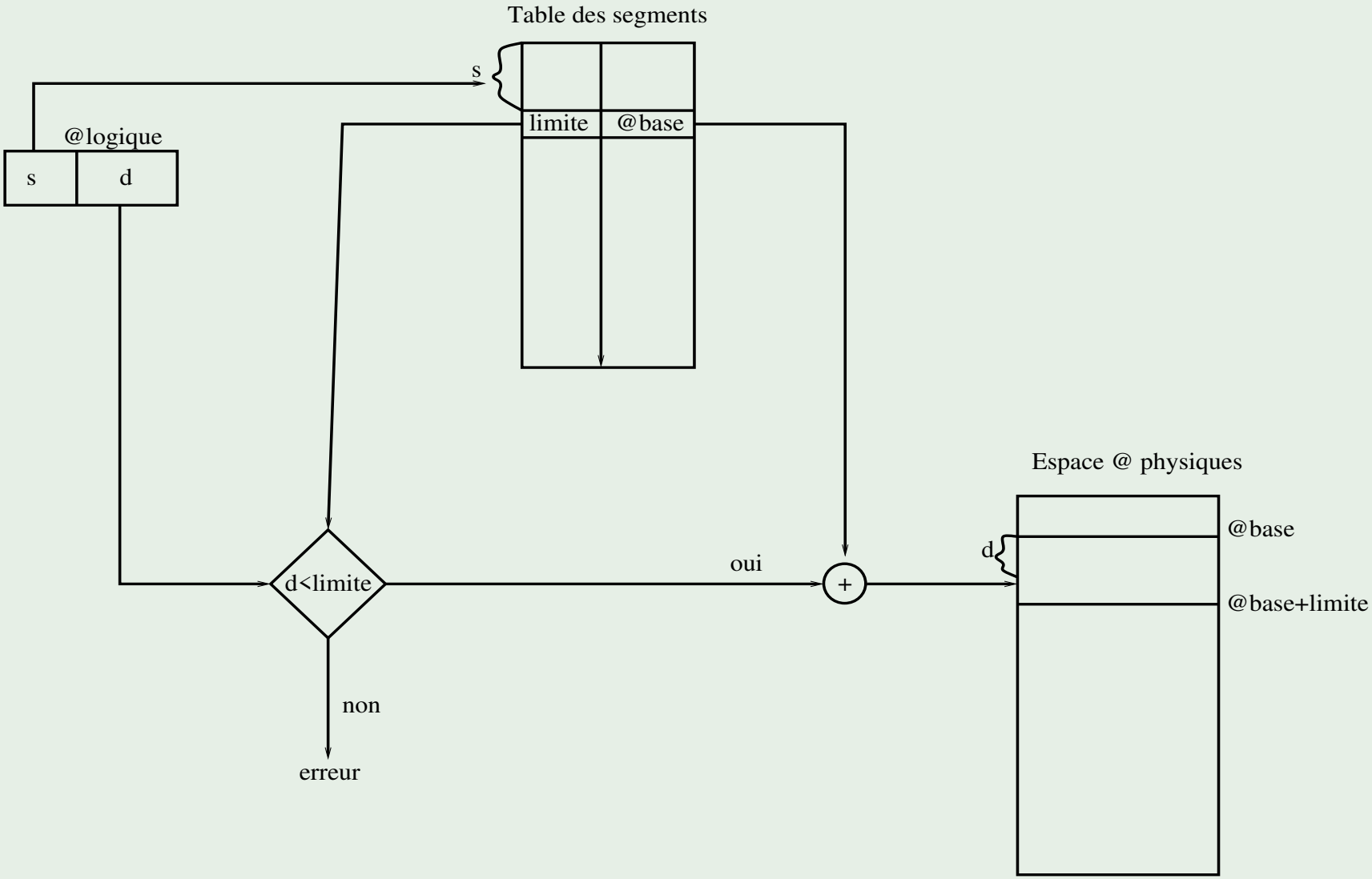
La segmentation

Segmentation pure



La segmentation

Implantation



La segmentation

Implantation

- *Segment table base register* (STBR) : pointe sur la table des segments
- *Segment table length register* (STLR) : indique la longueur de la table des segments
- Adresse logique : (s, d)
  s est légale si $s < \text{STLR}$
  Si s est légale, alors calculer l'adresse physique en utilisant STBR.
- Ralentissement $\Rightarrow$ registres associatifs.

La segmentation

Implantation

- permet de disposer de code "*relogeable*" (déplaçable en mémoire)
- pour déplacer un segment, il suffit de modifier sa base.

La segmentation

Protection

- Association de la protection avec le segment
- Le matériel de conversion de la mémoire vérifie les bits de protection associés à chaque entrée de la table de segments pour empêcher des accès illégaux à la mémoire.

Partage

- Les segments sont partagés quand les entrées dans les tables de segments de 2 processus différents pointent vers le même emplacement.
- Chaque processus possède une table des segments.

Pagination/Segmentation

	pagination	segmentation
Définition	une page a une taille de bloc fixe	un segment est de taille variable
Fragmentation	peut entraîner une fragmentation interne	peut conduire à une fragmentation externe
Adresse	l'adresse spécifiée par l'utilisateur est divisée par le CPU en un numéro de page + un décalage	L'utilisateur spécifie chaque adresse par 2 quantités : un numéro de segment et le décalage (limite de segment)
Taille	la matériel décide la taille de page	la taille du segment est spécifiée par l'utilisateur
Table	une table de pages : contient l'adresse de	une table de segments : contient le numéro de segment et sa longueur

Pagination/Segmentation

	pagination	segmentation
le programmeur doit savoir que cette technique est utilisée	non	oui
combien d'espaces d'adressage linéaires ?	1	plusieurs
l'espace total d'adresses peut-il excéder la taille de la mém. physique ?	oui	oui
les proc. et les données peuvent-elles être distinguées et protégées séparément ?	non	oui
les tables dont la taille varie peuvent-elles être facilement logées	non	oui
le partage entre processus est-il facilité ?	non	oui
pourquoi cette technique ?	un grand espace d'adresses sans devoir acquérir plus de mém. physique	permettre au programme et aux données d'être logiquement séparés en espaces d'ad. indépendants pour facilier le partage et la protection.

La segmentation

Descripteur de segment

- une base : sur 32 bits, indique l'adresse où débute le segment en mémoire
- la limite : sur 16 bits, définit la longueur du segment. Exprimée en octets ou en nombres de pages
- un type : indique si segment de données, de code ou de pile,
- un bit indique le niveau de privilège du segment, 0 correspond au mode super-utilisateur
- un bit de présence du segment en mémoire
- un bit qui précise la taille des données et des instructions manipulées. Mis à 1 pour une taille de 32 bits.

La segmentation

Table de Descripteurs

- une table globale : par ex. contient les segments de code et de données pouvant être partagés par toutes les (ou partie des) tâches (code/données de l'OS, code/données des bibliothèques)
- locale : propre à une tâche.

Segmentation avec pagination

Les 2 méthodes sont combinées :

- Segment = {pages}
- adresse virtuelle = (numéro segment, numéro page, déplacement dans la page).

Traditionnellement, le processus a son espace composé de 4 segments : code, données, pile et tas. Chaque segment est organisé en pages, et a sa propre table des pages.

Algorithmes de remplacement de pages

- À la suite d'un défaut de page, le SE doit ramener en mémoire la page manquante à partir du disque.
- Si pas de cases libres en mémoire, le SE retire une page en mémoire pour la remplacer par la page demandée.
- Si la page à retirer a été modifiée depuis son chargement en mémoire, il faut la réécrire sur le disque.

Quelle page retirer de manière à minimiser le défaut de pages ?

Algorithmes de remplacement de pages

Choix de la page à retirer

- Peut se limiter aux pages du processeur qui a provoqué le défaut de page (*allocation locale*) ou à l'ensemble des pages en mémoire (*allocation globale*).
- En général, l'allocation globale produit de meilleurs résultats que l'allocation locale.

Ces algorithmes mémorisent les références passées aux pages. Le choix de la page à retirer dépend des références passées.

Algorithmes de remplacement de pages

Algorithme aléatoire

Choisir au hasard une page victime, à retirer de la mémoire.

- Facile.
- Versions locale et globale.
- Utilisé pour des comparaisons entre méthodes.

Algorithmes de remplacement de pages

Algorithme optimal (Belady)

Remplacer la page qui sera référencée le plus tard possible dans le futur.

- Irréalisable.
- Versions locale et globale.
- Intérêt : pour faire des études comparatives.

Algorithme implanté avec 2 exécutions :

1. simulation d'exécution. On garde une trace des références de toutes les pages.
2. exécution réelle. On utilise les informations de références de pages.

Algorithme optimal

Exemple

Avec 3 blocs et la série de demandes de pages suivante :

7	0	1	2	0	3	0	4	2	3	0	3	2	1	2	0	1	7	0	1
7	7	7																	
	0	0																	
		1																	

Algorithme optimal

Exemple

Avec 3 blocs et la série de demandes de pages suivante :

7	0	1	2	0	3	0	4	2	3	0	3	2	1	2	0	1	7	0	1
7	7	7	2	2															
	0	0	0	0															
		1	1	1															

Algorithme optimal

Exemple

Avec 3 blocs et la série de demandes de pages suivante :

7	0	1	2	0	3	0	4	2	3	0	3	2	1	2	0	1	7	0	1
7	7	7	2	2	2	2													
	0	0	0	0	0	0													
		1	1	1	3	3													

Algorithme optimal

Exemple

Avec 3 blocs et la série de demandes de pages suivante :

7	0	1	2	0	3	0	4	2	3	0	3	2	1	2	0	1	7	0	1
7	7	7	2	2	2	2	2	2	2										
	0	0	0	0	0	0	4	4	4										
		1	1	1	3	3	3	3	3										

Algorithme optimal

Exemple

Avec 3 blocs et la série de demandes de pages suivante :

7	0	1	2	0	3	0	4	2	3	0	3	2	1	2	0	1	7	0	1
7	7	7	2	2	2	2	2	2	2	2	2	2							
	0	0	0	0	0	0	4	4	4	0	0	0							
		1	1	1	3	3	3	3	3	3	3	3							

Algorithme optimal

Exemple

Avec 3 blocs et la série de demandes de pages suivante :

7	0	1	2	0	3	0	4	2	3	0	3	2	1	2	0	1	7	0	1
7	7	7	2	2	2	2	2	2	2	2	2	2	2	2	2	2			
	0	0	0	0	0	0	4	4	4	0	0	0	0	0	0	0			
		1	1	1	3	3	3	3	3	3	3	3	1	1	1	1			

Algorithme optimal

Exemple

Avec 3 blocs et la série de demandes de pages suivante :

7	0	1	2	0	3	0	4	2	3	0	3	2	1	2	0	1	7	0	1
7	7	7	2	2	2	2	2	2	2	2	2	2	2	2	2	2	7	7	7
	0	0	0	0	0	0	4	4	4	0	0	0	0	0	0	0	0	0	0
		1	1	1	3	3	3	3	3	3	3	3	1	1	1	1	1	1	1

Nombre d'accès : 20

Défaut de pages : 9 (45%)

Algorithme NRU (Not recently used)

Remplacement de la page non récemment utilisée.

- 2 bits d'état associés à chaque page et contenus dans chaque entrée de la table des pages :
  1. `R` mis à 1 chaque fois que la page est référencée ;
  2. `M` mis à 1 chaque fois que la page est modifiée.

  Mises à jour matérielles des bits `R` et `M`.
- Catégories par ordre de remplacement :
  1) non référencée, non modifiée ;
  2) non référencée, modifiée ;$\rightsquigarrow$ survient si une page de catégorie 3 a eu son bit R effacé lors d'une interruption.
  3) référencée, non modifiée ;
  4) référencée, modifiée ;

Algorithme NRU

enlève une page au hasard dans la catégorie la plus basse non vide.

- relativement simple à implanter
- performances suffisantes, sans être optimales
- utilisé dans MacOS (<X)

Algorithmes de remplacement de pages

Algorithme FIFO

La page dont le temps de résidence est le plus long.

- Implantation facile :
  pages résidentes en ordre FIFO $\rightsquigarrow$ on retire la première.
- Ce n'est pas une bonne stratégie :
  son critère n'est pas fondé sur l'utilisation de la page.
- Anomalie :
  On peut rencontrer des exemples où en augmentant le nombre de blocs, on augmente le nombre de défauts de pages au lieu de le diminuer !

Algorithme FIFO

Exemple 1

Avec 3 blocs et la série de demandes de pages suivante :

7	0	1	2	7	0	3	7	0	1	2	3	7	0	1	2	3	2	4	3
7	7	7	2	2	2	3	3	3	3	3	3	7	7	7	2	2	2	2	2
	0	0	0	7	7	7	7	7	1	1	1	1	0	0	0	3	3	3	3
		1	1	1	0	0	0	0	0	2	2	2	2	1	1	1	1	4	4

Nombre d'accès : 20

Défaut de pages : 15 (75%)

Algorithme FIFO

Exemple 3

Avec 4 blocs et la même série de demandes de pages :

7	0	1	2	7	0	3	7	0	1	2	3	7	0	1	2	3	2	4	3
7	7	7	7	7	7	3	3	3	3	2	2	2	2	1	1	1	1	1	1
	0	0	0	0	0	0	7	7	7	7	3	3	3	3	2	2	2	2	2
		1	1	1	1	1	1	0	0	0	0	7	7	7	7	3	3	3	3
			2	2	2	2	2	2	1	1	1	1	0	0	0	0	0	4	4

Nombre d'accès : 20

Défaut de pages : 16 (80%)

Algorithmes de remplacement de pages

Algorithme de seconde chance

Une variante de FIFO.

- FIFO +
  - si bit R = 0, elle est remplacée ;
  - si bit R = 1, R est effacé, la page est enfilée à la fin de la liste des pages et son temps de chargement est mis à jour comme si elle venait d'arriver en mémoire. La recherche continue.
- peut dégénérer en FIFO...

Algorithmes de remplacement de pages

Algorithme LRU (Least Recently Used)

Page résidente la moins récemment utilisée.

- fondé sur le critère de localité :
  une page a tendance à être réutilisée dans un futur proche.
- Difficile à implanter sans un support matériel dédié.

Algorithme LRU

Comment implanter LRU ?

- mémoriser pour chaque page en mémoire la date de la dernière référence
- utilisation d'une liste doublement chaînée :
  - la page la plus récemment utilisée est en tête, la moins utilisée en queue
  - mise à jour de la liste à chaque référence mémoire : coûteux !

Algorithme LRU

Ou

- utilisation d'un compteur c sur n bits par page ($n = 64$) $\rightsquigarrow$ algorithme NFU (Not Frequently Used)
  - chaque entrée de la table des pages doit contenir c
  - à chaque top et pour toutes les pages, décaler le compteur à droite et positionner le bit de poids fort à la valeur du bit R (référence)
  - défaut de pages : on cherche l'entrée avec la plus petite valeur de c : *référencée le plus anciennement*
    La sélection entre numéros identiques se fait avec l'ordre FIFO
  - avec n bits, le SE "oublie" un référencement après n tops.

Algorithmes de remplacement de pages

Algorithme de vieillissement

Approximation efficace de LRU

simule efficacement les pages les moins récemment utilisées

- on utilise un compteur avec un nombre fini de bits (8 suffisent en général) et les tops d'horloge se produisent toutes les N ms (en général 20 ms)
- le SE positionne à 1 le bit de poids fort à chaque accès à la page
- toutes les N ms, le SE fait un décalage à droite de l'octet associé à chaque page $\rightsquigarrow$ on obtient un historique de l'utilisation de la page
  Ex. la page de masque 11000100 a été utilisée plus récemment que celle de masque 01110111
- le bit le plus significatif est mis à 1 à chaque référence
  régulièrement, on décale vers la droite les bits de ce registre, on choisit la page dont la valeur est la plus petite

Algorithme LRU

Exemple 1

Avec 3 blocs et la série de demandes de pages suivante :

7	0	1	2	0	3	0	4	2	3	0	3	2	1	2	0	1	7	0	1
7	7	7	2	2	2	2	4	4	4	0	0	0	1	1	1	1	1	1	1
	0	0	0	0	0	0	0	0	3	3	3	3	3	3	0	0	0	0	0
		1	1	1	3	3	3	2	2	2	2	2	2	2	2	2	7	7	7

Nombre d'accès : 20

Défaut de pages : 12 (60%)

Algorithmes de remplacement de pages

Algorithme de l'horloge

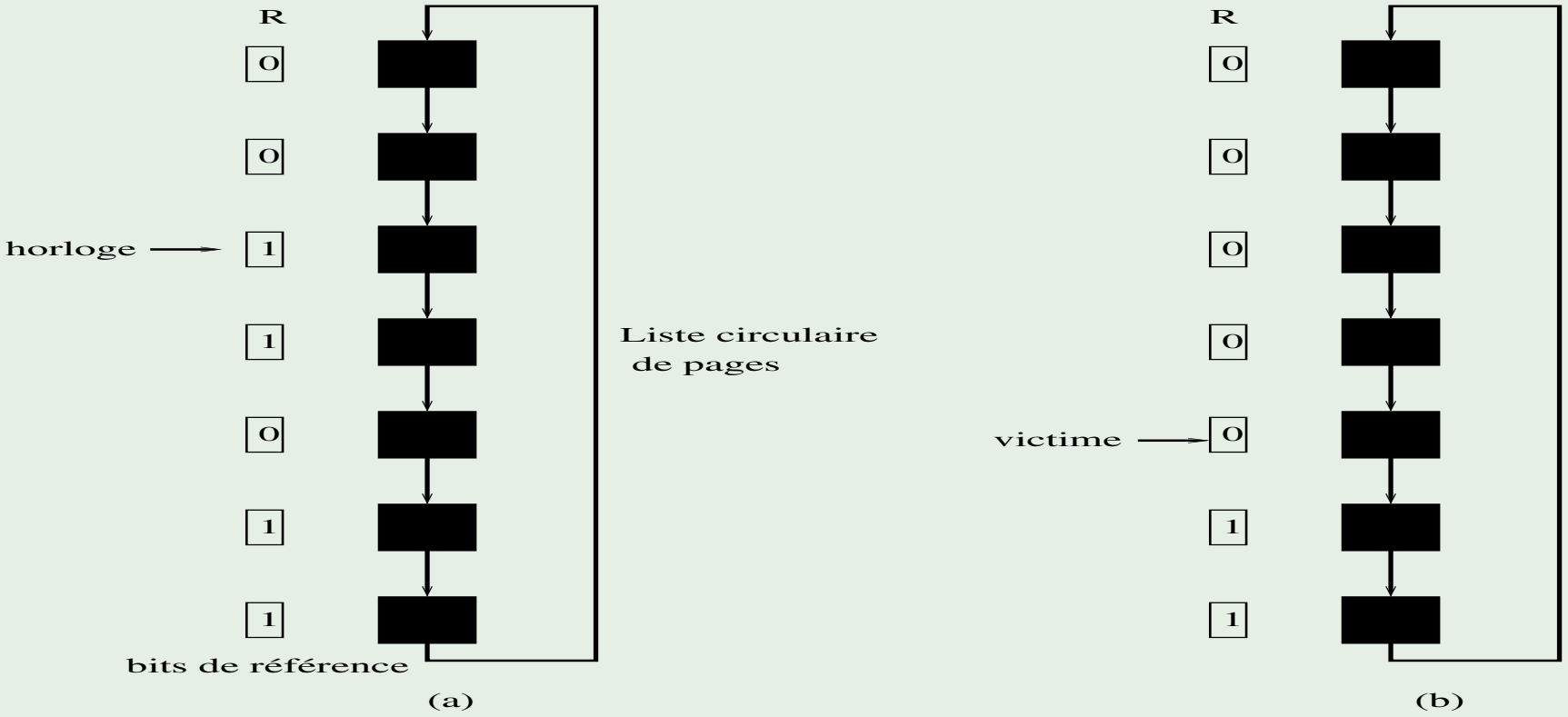
Une variante de LRU.

- Les pages en mémoire sont mémorisées dans une liste circulaire en forme d'horloge ;
- On a un indicateur sur la page la plus ancienne.
- Si défaut de page, les pages sont examinées, une par une, en commençant par celle pointée par l'indicateur ;
- la première page rencontrée avec un bit de référence `R` à 0 est remplacée. Le bit `R` de la page chargée est mis à 1 ;
- si le bit `R` d'une page examinée est différent de 0, il est mis à 0.

Une variante de cet algorithme tient compte aussi du bit de modification `M`.

Algorithme de l'horloge

Exemple :



(a) quand une page est référencée, son bit R est mis à 1.
L'horloge indique la première victime possible. Pas retenue car son bit R est à 1.

(b) on continue jusqu'à une page avec R=0. Au passage, on remet R à 0.

Algorithmes de remplacement de pages

Résumé

- L'algorithme optimal n'est pas praticable. Référence comparative.
- NRU facile à implanter mais grossier. 4 catégories de pages.
- FIFO est mauvais. Efface la page la plus ancienne, même si elle sert encore.
- $2^{\text{ème}}$ chance augmente considérablement les performances de FIFO.
- LRU est excellent mais non réalisable sans matériel dédié.
  NFU est une approximation grossière de LRU, pas très bon.
  L'algorithme de vieillissement et l'algorithme de l'horloge sont plus efficaces.

Politique d'allocation

Si un processus p provoque un défaut de page,

- Allocation locale : l'algorithme de remplacement de pages cherche la page à décharger parmi les pages de p uniquement.
- Allocation globale : l'algorithme de remplacement de pages cherche la page à décharger parmi toutes les pages en MC.

Politique d'allocation

- Les algorithmes locaux allouent à chaque processus une partie fixe de la mémoire.
- Les algorithmes globaux allouent dynamiquement des cases aux processus.
  Conséquence : Le nombre de cases mémoire allouées à chaque processus varie dans le temps.

Politique d'allocation

En général,

les algorithmes globaux sont plus efficaces, surtout lorsque la taille de l'espace de travail du processus peut varier au cours du temps.

- Moins de risque d'écroulement 
- Moins de gaspillage de mémoire 
- Le système doit décider à tout moment du nombre de cases à allouer à chaque processus :
  - déterminer à tout moment le nombre de processus en cours d'exécution. Allouer la même part à chaque processus.
    Équitable mais non justifiée si on alloue le même nombre de cases à 2 processus ayant des tailles d'un ordre de grandeur différent.
  - démarrer chaque processus avec un nombre de cases proportionnel à sa taille. Allocation réajustée dynamiquement.

Politique d'allocation

Algorithme PFF (Page Fault Frequency)

indique quand augmenter ou diminuer l'allocation de page pour un processus

- n'indique rien sur la page à remplacer
- se limite à contrôler la taille de l'ensemble d'allocation

propriété des algorithmes de remplacement de pages

le nombre de défaut de pages $\searrow$ quand le nombre de pages allouées $\nearrow$

- limite supérieure : taux excessivement élevé de défauts de page
- limite inférieure : taux de défauts de page si bas qu'on peut en déduire que le processus a trop de mémoire

PFF tente de maintenir un taux de pagination dans les limites acceptables.

Les pages partagées

Plus efficace de partager des pages que d'avoir 2 copies de la même page en mémoire en même temps.
Partager du code et/ou partager des données ? Le plus simple :

- Seules, les pages en lecture seule sont partagées. Ex. le code d'un programme.
- Structures de données spécifiques pour mémoriser les pages partagées. Si p et q partagent le même code, quand p termine, les pages du code ne doivent pas être déchargées.

Les pages partagées

Partager des données

Ex. dans Unix, après un appel système de `fork`, le parent et le fils partagent le code et les données,

- l'un et l'autre pointent sur le même ensemble de pages ;
- toutes les pages de données des 2 processus sont marquées en lecture seule.

Si mise à jour d'un mot mémoire par l'un des 2 processus,

- déroutement au SE pour cause de violation du mode lecture seule. Le SE fait une copie de la page et chaque processus a son exemplaire.
- les pages non réécrites ne sont pas copiées

Approche *copy on write* : réduit les copies $\rightsquigarrow \nearrow$ les performances.

Gestion de la mémoire

Les bibliothèques partagées

- On n'inclut pas le code de la bibliothèque, trop gourmand !
- on utilise une routine qui permet au processus de se lier à la fonction à l'exécution.
- Les bibliothèques partagées sont chargées
  - au moment du chargement du programme,
  - ou au premier appel de fonction.
- si bibliothèques déjà chargées, on ne les charge pas une $2^{\text{ème}}$ fois.
- les bibliothèques sont paginées, $\Rightarrow$ chargées par pages à la demande.

Économie d'occupation mémoire et fichiers exécutables plus petits

Gestion de la mémoire cache

Souvent plusieurs niveaux de mémoire cache

Souvent de la SRAM (Static Random Access Memory)

- interne au processeur
- intégré sur la carte mère
- possiblement sur le disque dur, dans les serveurs proxy.

La mémoire cache duplique l'information.

Défaut de cache : lorsque le processeur demande une information et qu'elle ne réside pas en mémoire cache.

Gestion de la mémoire cache

Les niveaux de mémoire cache

- L1 : une petite quantité de mémoire très rapide, implantée sur le processeur. Temps d'accès = 1 ou 2 cycles de processeur.
  Intel Xeon E3-1285 v3 4 cœurs : 64 Ko
- L2 : quantité un peu plus grande de mémoire, un peu moins rapide. Connectée au bus interne du processeur, $\approx$ 6-8 cycles du processeur.
  Intel Xeon E3-1285 v3 : 256 Ko/cœur,
- L3 : cache externe, temps d'accès $\approx$ 21 cycles.
  Intel Xeon E3-1285 v3 : 8 Mo.

Il y a une limite au-delà de laquelle l'augmentation de la taille du cache ne sert plus à rien (principe de localité des traitements).

Gestion de la mémoire cache

Lors d'un défaut de cache,

la mémoire cache récupère l'information (de la MC, du disque,...), la stocke et la transmet au demandeur.
2 principes de fonctionnement :

1. localité spatiale : l'accès à une instruction située à l'adresse `X` va probablement être suivi d'un accès à une zone proche de `X` ;
2. localité temporelle : l'accès à une zone mémoire à un instant t a de fortes chances de se reproduire dans la suite du programme.

Gestion de la mémoire cache

Mémoire cache des processeurs

- Plus la taille de la mémoire cache est grande, plus la taille des programmes accélérés est élevée
  - pour profiter du temps d'accès rapide, il faut que les parties de programme tiennent le plus possible dans cette mémoire cache
  - rôle d'optimisation souvent dédié au compilateur
- Élément souvent utilisé par les constructeurs pour augmenter les performances d'un produit sans changer d'autres matériels. Ex.
  - séries bridées (de microprocesseurs avec une taille de mémoire cache volontairement réduite), tels que les Duron chez AMD ou Celeron chez Intel
  - séries haut de gamme avec une grande mémoire cache comme les processeurs Opteron chez AMD, ou Pentium EE ou Core i7 chez Intel

Gestion de la mémoire cache

Ligne

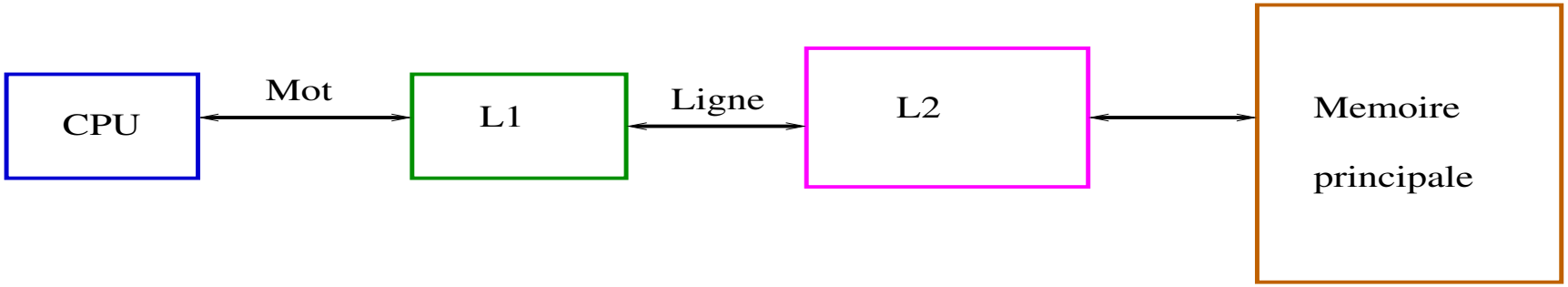
Plus petit élément de données qui peut être transféré entre la mémoire cache et la mémoire de niveau supérieur.

Mot

Plus petit élément de données qui peut être transféré entre le processeur et la mémoire cache.

Gestion de la mémoire cache

- subdivisée en niveaux (jusqu'à 3)
- très rapide ⇒ très chère



Gestion de la mémoire cache

- cache de 1er niveau (L1) dans les processeurs (cache de données souvent séparé du cache d'instructions)
- cache de second niveau (L2) dans certains processeurs (peut se situer hors de la puce)
- cache de 3ème niveau (L3) rarement
  - dans les disques durs
  - dans les serveurs proxy

Gestion de la mémoire cache

Types de défauts de cache

3 types de défauts de page en système monoprocesseur, et 4 en multiprocesseur

1. défaut obligatoire. 1ère demande du processeur pour une donnée ou instruction. Ne peut pas être évité
2. défaut capacitif. L'ensemble des données du processus excède la taille du cache
3. défaut conflictuel. 2 adresses distinctes de la mémoire de niveau supérieur sont enregistrées au même endroit dans le cache.
4. défaut de cohérence : dus à l'invalidation de lignes de la mémoire cache pour maintenir la cohérence entre les différents caches des processeurs d'un système multi-processeurs.

Gestion de la mémoire cache

Le mapping

indique à quelle adresse de la mémoire cache doit être écrite une ligne de la MC.

3 types de mapping :

1. mémoire cache complètement associative
2. mémoire cache directe (à correspondance préétablie)
3. mémoire cache N-associative

Le mapping

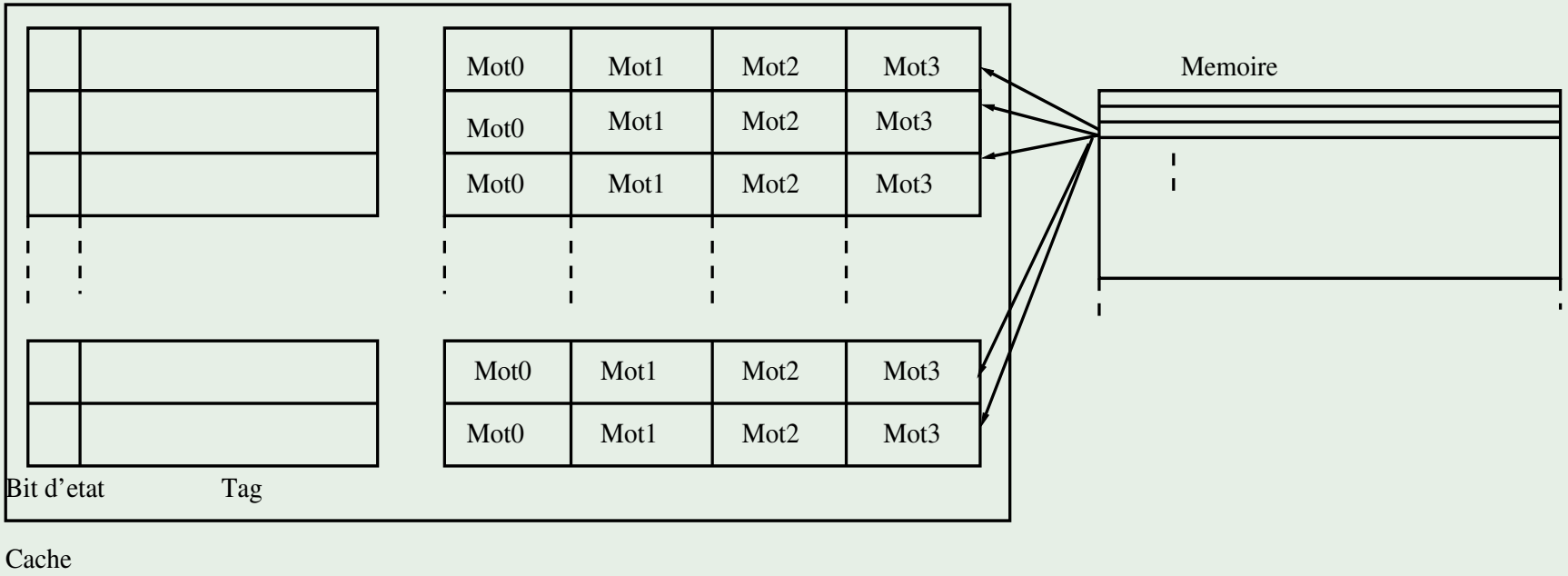
1. Mémoire cache complètement associative :
   - chaque ligne de la mémoire de niveau supérieur peut être écrite à n'importe quelle adresse de la mémoire cache
   - donne accès à bcp de possiblités. Utilisée uniquement dans les mémoires cache de petite taille.

Tag = n° ligne mémoire enregistrée

offset = n° mot dans la ligne.

Le mapping

Mémoire cache complètement associative



Le mapping

Le mapping

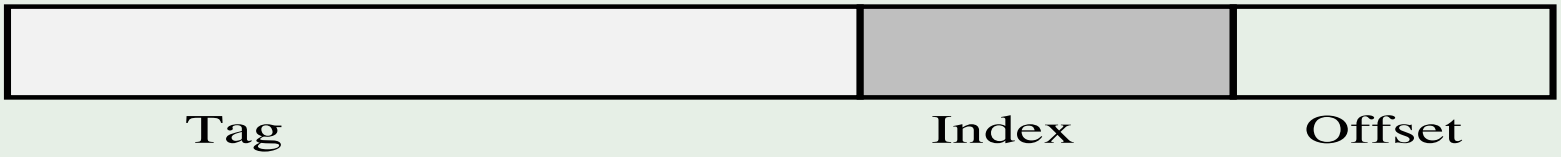
2. Mémoire cache directe :
   - chaque ligne de la MC ne peut être enregistrée qu'à une seule adresse de la mémoire cache (ex. son adresse modulo)
   - si accès à des données mappées sur les mêmes adresses du cache ⇒ nombreux défauts de cache conflictuels
   - sélection de la ligne où la donnée sera enregistrée :
     `ligne = adresse_mémoire modulo nombre_de_lignes`

   une ligne de cache est partagée par de nombreuses adresses de la mémoire de niveau supérieur

simple mais peu efficace : nombreux défauts conflictuels

La mémoire cache directe

- il faut un moyen de savoir quelle donnée est actuellement dans le cache
- information fournie par le *tag*, stocké dans le cache
- l'index correspond à la ligne où est enregistrée la donnée
- le contrôleur de la mémoire cache doit savoir si une ligne contient une donnée ou non. Un bit additionnel (bit de validité) indique si la ligne est libre ou non.



La mémoire cache directe

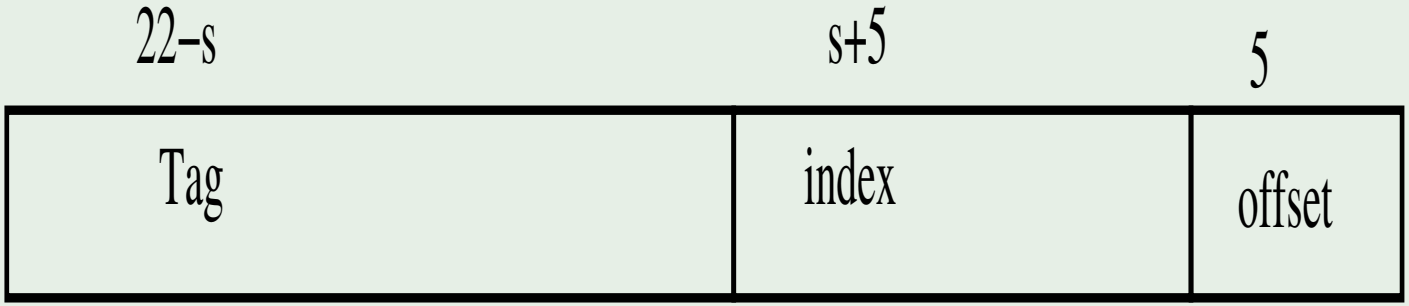
Exemple

Hypothèses :

- mémoire adressable par octet
- taille(ligne) $= 256$ bits $= 2^8$ bits $= 2^5$octets
- adresse sur 32 bits $= 2^5$ bits
- taille(mémoire cache) $= 2^s$ Koctets $= 2^{s+13}$ bits

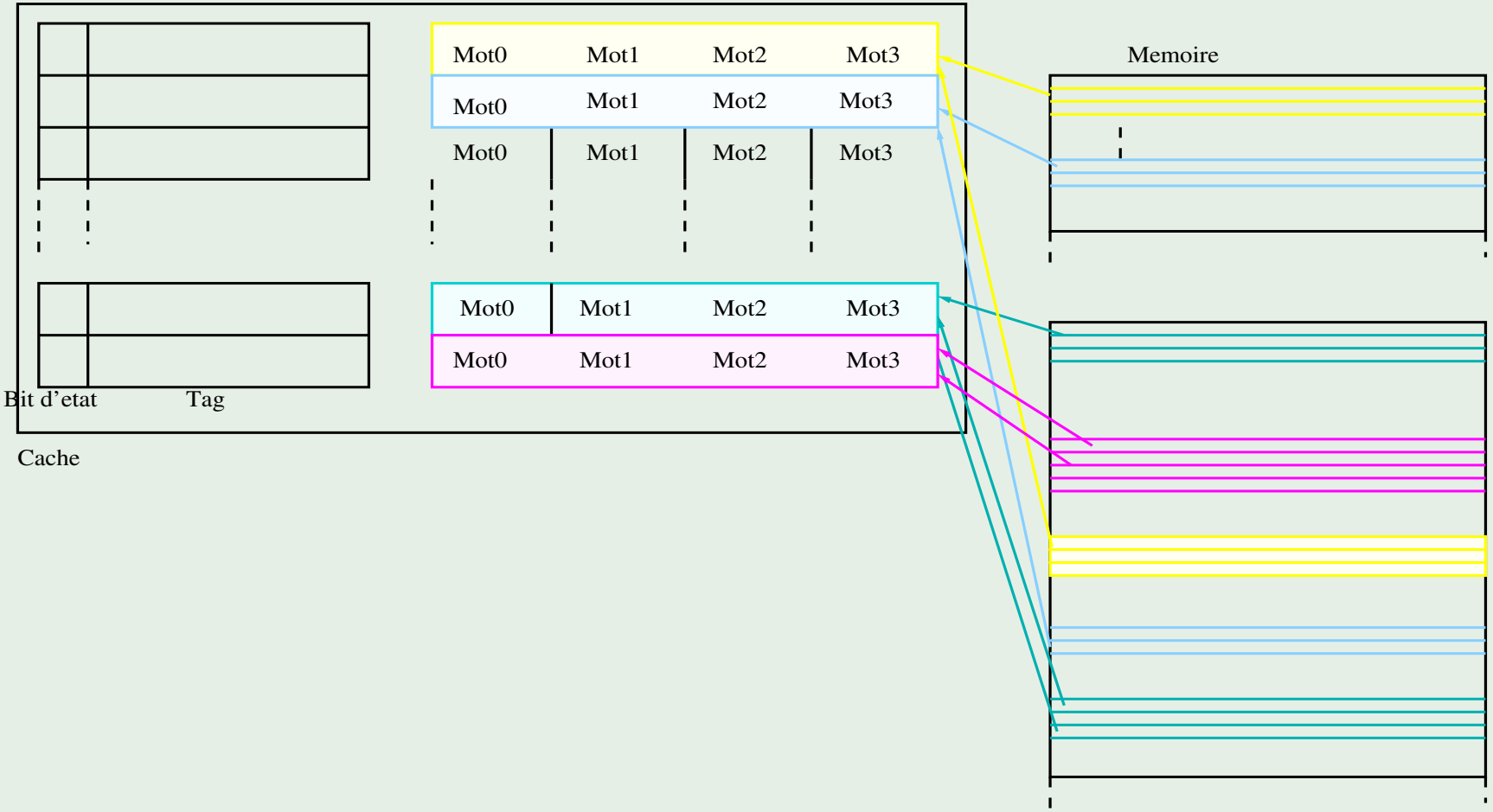
On en déduit :

1. 2^{s+5} lignes de cache $\Rightarrow$ l'index est sur $s+5$ bits.
2. $32 =$ taille(tag) $+(s+5)+5 \Rightarrow$ taille(tag) $= (22-s)$ bits



Le mapping

Mémoire cache directe



Le mapping

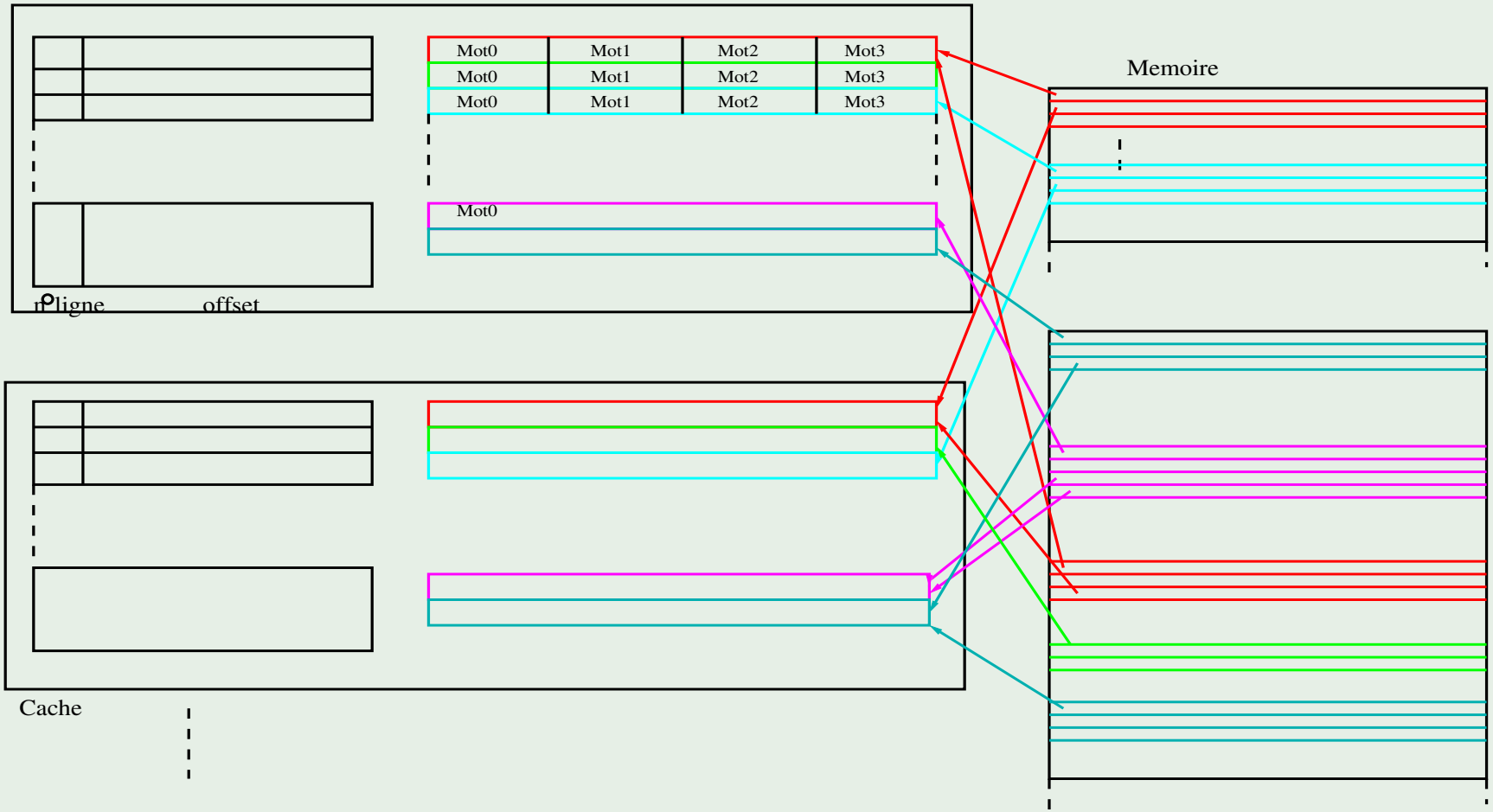
Le mapping

compromis entre les mappings direct et complètement associatif.

3. Mémoire cache N-associative :
   - la mémoire cache est divisée en ensembles de N lignes de cache
   - une ligne de la mémoire de niveau supérieur est affectée à un ensemble. Dans cet ensemble, elle peut être écrite dans n'importe quelle ligne
   - à l'intérieur d'un ensemble, le mapping est direct. Le mapping des N ensembles est complètement associatif
   - sélection de l'ensemble :
     `ensemble = adresse_mémoire modulo (nombre_d'ensembles)`

Le mapping

Mémoire cache N-associative

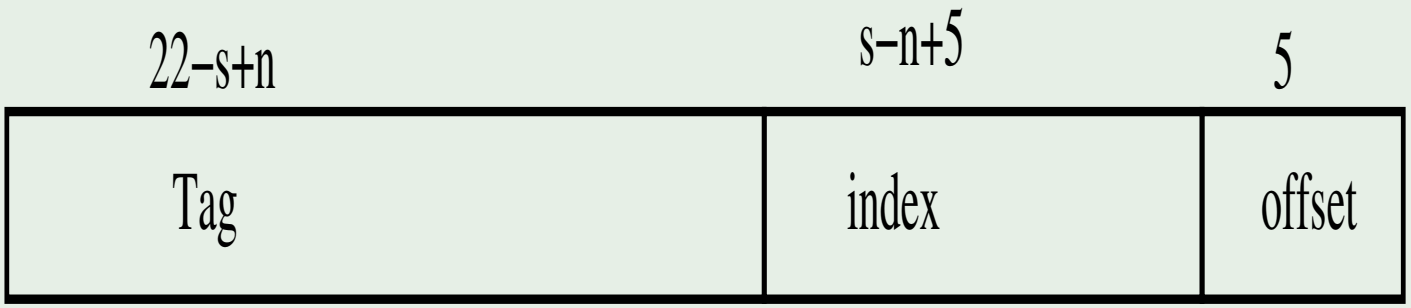


un ensemble est représenté par l'union des rectangles rouges par ex.

Mémoire cache N-associative

Exemple

- mémoire cache de 2^s Koctets $= 2^{s+13}$ bits, constituée de 2^n voies
  $\Rightarrow$ on a 2^{s+13-n} bits / voie
- taille(ligne) $= 2^8$ bits
  $\Rightarrow$ on a 2^{s+5-n} entrées par ensemble $\Rightarrow$ l'index est sur $s+5-n$ bits
- $32 =$ taille(tag) $+(s+5-n)+5 \Rightarrow$ taille(tag) $= 22-s+n$



Gestion de la mémoire cache

Caches unifiés ou caches séparés

Pour fonctionner, un processeur a besoin de données et d'instructions

- cache unifié : données et instructions sont enregistrées dans la mémoire cache,
  + une logique de priorité d'accès.
- caches séparés de données et d'instructions. Le processeur peut accéder simultanément à une donnée et une instruction.

optimisation : tient compte du fait que les instructions sont *très rarement* modifiées par le programme.

Gestion de la mémoire cache

Politique d'écriture dans la mémoire de niveau supérieur

qd une donnée/instruction est dans le cache, le système en possède 2 copies : une dans la mémoire de niveau supérieur et une dans la mémoire cache :

2 politiques :

1. *write through :* la donnée/instruction est écrite à la fois dans le cache et dans la mémoire de niveau supérieur ⇒ cohérence constante
2. *write back :* l'information n'est écrite dans la mémoire de niveau supérieur que lorsque la ligne disparaît du cache (invalidée par d'autres processeurs, évincée pour écrire une autre ligne...).
   - technique la plus répandue : évite de nombreuses écritures mémoires inutiles
   - pour ne pas écrire des informations qui n'ont pas été modifiées, chaque ligne de la mémoire cache dispose d'un bit indiquant si elle a été modifiée

Gestion de la mémoire cache

Algorithmes de remplacement des lignes de cache

On retrouve l'algorithme aléatoire, FIFO, LRU, Horloge...

Les processeurs multicœurs

Composés d'au moins deux cœurs (ou unités de calcul) gravés au sein de la même puce.

Sur ces processeurs, l'organisation des mémoires caches est adaptée à la présence de plusieurs cœurs.

Deux grandes organisations de base :

1. à base de caches dédiés à chaque cœur,
2. à base de caches partagés.

Les processeurs multicœurs

Les caches dédiés

Chaque cœur possède son propre cache, que lui seul peut utiliser ⇒ un cache pour chaque cœur.

- Un programme qui s'exécute sur un cœur ne va pas polluer le cache d'un autre processeur.
- Deux programmes exécutés sur des cœurs différents n'interfèrent pas.

- Le temps d'accès à un cache dédié est souvent plus faible que celui d'un cache partagé : les caches dédiés sont souvent d'une taille assez faible comparé à un gros cache partagé, conçu pour répondre aux besoins de plusieurs cœurs.
- Une donnée utilisée par deux threads lancés sur deux cœurs différents doit être présente dans chacun des caches dédiés.

Les caches partagés

La mémoire cache est partagée entre tous les processeurs, qui peuvent y accéder de façon concurrente.

- Deux programmes peuvent se partager le cache dynamiquement ⇒ ils peuvent se répartir l'occupation du cache d'une manière bien plus souple que ce qui est permis avec un cache dédié ;
- L'implantation de mécanismes de cohérence des caches est facilitée.
- Le cache partagé évite de répliquer des données partagées entre plusieurs threads : la donnée est présente une seule fois dans le cache partagé.

Les caches partagés

- Un cache partagé doit avoir une bande passante suffisante pour alimenter plusieurs cœurs ; se fait souvent au détriment de sa latence.
- Il doit pouvoir permettre à plusieurs cœurs d'accéder à des données différentes dans le cache simultanément, afin d'éviter de mettre un cœur en attente pendant un accès au cache.
  Conséquence : ces caches sont des mémoires multiports, $\rightsquigarrow$ impact sur leur consommation énergétique, leur latence, et leur débit binaire.
- Grande taille $\Rightarrow$ impact négatif sur leur latence et leur consommation énergétique.
- Plusieurs programmes peuvent se marcher sur les pieds dans leur répartition de la mémoire cache. Phénomène rare, mais pas impossible.