V. GESTION DE LA MÉMOIRE

GESTION DE LA MÉMOIRE

- La mémoire principale est le lieu où se trouvent les programmes et les données quand le processeur les exécute.
- On l'oppose au concept de mémoire secondaire, représentée par les disques, de plus grande capacité, où les processus peuvent séjourner avant d'être exécutés.
- La nécessité de gérer la mémoire de manière optimale est toujours fondamentale, car en dépit de sa grande disponibilité, elle n'est, en général, jamais suffisante. Ceci en raison de la taille continuellement grandissante des programmes.

GESTION DE LA MÉMOIRE

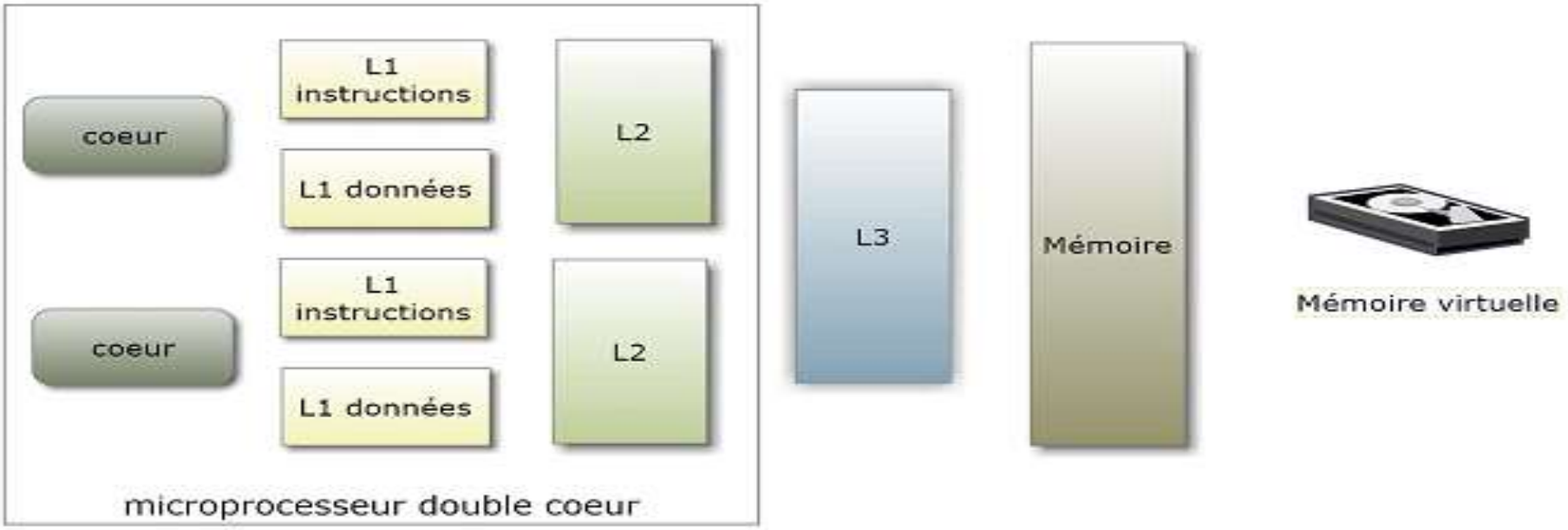
La gestion de la mémoire a deux objectifs principaux :

- d'abord le partage de mémoire physique entre les programmes et les données des processus prêts,
- et ensuite la mise en place des paramètres de calcul d'adresses, permettant de transformer une adresse virtuelle en adresse physique.
- Pour ce faire le gestionnaire de la mémoire doit remplir plusieurs tâches :
  - Connaître l'état de la mémoire (les parties libres et occupées de la mémoire).
  - Allouer de la mémoire à un processus avant son exécution.

GESTION DE LA MÉMOIRE

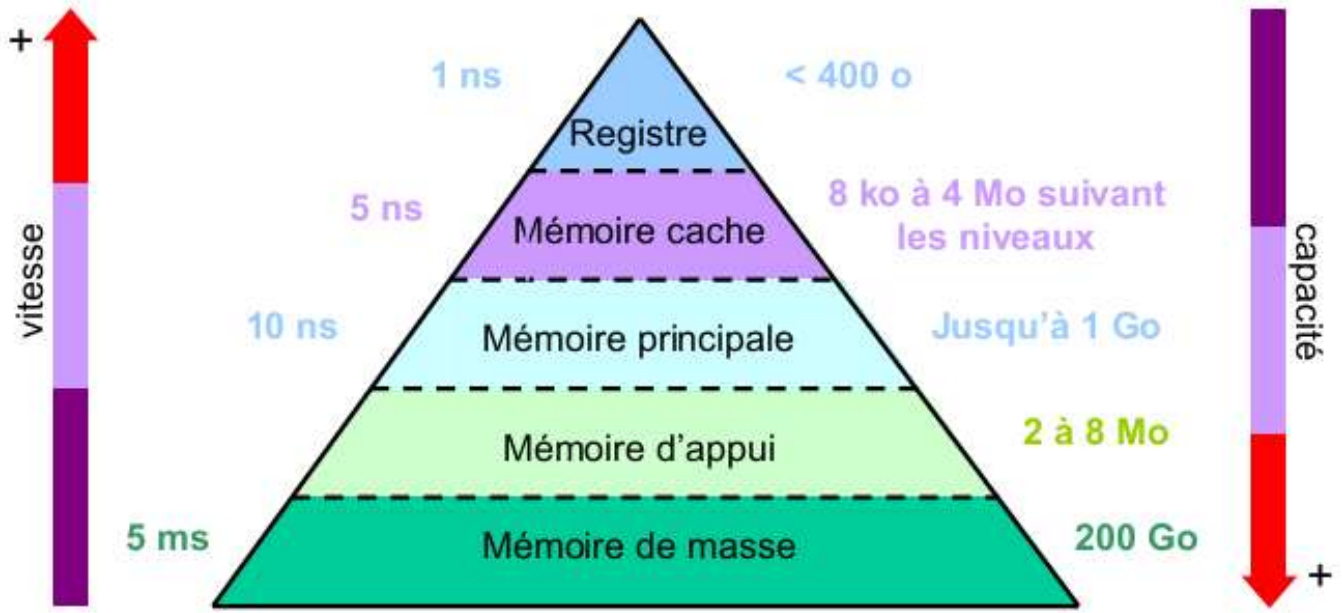
- Récupérer l'espace alloué à un processus lorsque celui-ci se termine
- Traiter le va-et-vient (swapping) entre le disque et la mémoire principale lorsque cette dernière ne peut pas contenir tous les processus.
- Calculer les adresses physiques (absolues) car, en général, les adresses générées par les compilateurs et les éditeurs de liens sont des adresses relatives ou virtuelles.

HIÉRARCHIE DE MÉMOIRE



- Chaque cœur :
  - Dispose de ses caches de niveau L1 (2 x 32Ko) et L2 (256Ko)
  - Partage le cache de niveau L3 (8Mo) et la mémoire (quelques Go)
- La mémoire virtuelle est sur le disque (quelques dizaines de Go)

HIERARCHIE DES MÉMOIRES



HIÉRARCHIE DES MÉMOIRES

- Les **registres** sont les éléments de mémoire les plus rapides. Ils sont situés au niveau du processeur et servent au stockage des opérandes et des résultats intermédiaires.
- La **mémoire cache** est une mémoire rapide de faible capacité destinée à accélérer l'accès à la mémoire centrale en stockant les données les plus utilisées.
- La **mémoire principale** est l'organe principal de rangement des informations. Elle héberge les programmes (instructions et données) et est plus lente que les deux mémoires précédentes.

HIÉRARCHIE DES MÉMOIRES

- La mémoire d'appui sert de mémoire intermédiaire entre la mémoire centrale et les mémoires de masse. Elle joue le même rôle que la mémoire cache.
- La mémoire de masse est une mémoire périphérique de grande capacité utilisée pour le stockage permanent ou la sauvegarde des informations. Elle utilise pour cela des supports magnétiques (disque dur, ZIP) ou optiques (CDROM, DVDROM).

CACHES ET MÉMOIRE VIRTUELLE

- **Deux propriétés importantes :**
  - **Localité spatiale** : l'accès à une donnée située à une adresse a de fortes chances d'être suivi d'un accès à une donnée situé à une adresse très proche de A.
  - **Localité temporelle** : l'accès à une zone mémoire à un instant donné a de fortes chances de se reproduire rapidement
- Les caches sont gérés par du matériel
  - Le SE peut configurer ce matériel (zones à ne pas mettre en cache)
- La mémoire virtuelle est gérée par le SE
  - Détection d'absence
  - Choix des zones de mémoire à remplacer
  - Copies disque-mémoire

GESTION DE LA MÉMOIRE

- Le SE doit gérer la distribution de la mémoire entre les différents processus ainsi que la mémoire virtuelle.

NB : Pour que ça marche il faut qu'un processus puisse s'exécuter n'importe où en mémoire.

- Normalement, quand on compile un programme, les noms de variables et d'étiquettes sont remplacés par des adresses fixes donc il faudrait recompiler le programme selon l'adresse où on le met et celle où on met ses variables

GESTION DE LA MÉMOIRE

- Les microprocesseurs modernes permettent d'éviter ça en utilisant des registres de segment ou de base. Toute adresse est définie relativement au contenu de ces registres
- Il suffit alors au SE d'initialiser ces registres selon l'endroit de la mémoire où il met le code du processus et ses variables.

GESTION DE L'ESPACE MÉMOIRE

- On divise la mémoire en blocs de tailles différentes (unités d'allocation).
- Quand un processus démarre on lui alloue un bloc qui lui suffit.

  S'il n'y en a pas il attend.
- Le SE gère l'utilisation de la mémoire de 3 façons de faire :
  - **Par table de bits** : chaque unité d'allocation a un bit dans cette table qui indique si elle est ou non allouée
  - **Par liste chaînée** : chaque élément de liste correspond à une unité d'allocation il indique si elle est libre ou pas et pointe sur la suivante.
  - **Par les subdivisions (Buddy system):** gestion d'une liste des blocs libres dont les tailles sont de 1, 2, 4, …, 2^n octets jusqu'à la taille maximale de la mémoire

CHOIX DE LA ZONE POUR UN NOUVEAU PROCESSUS

On a plusieurs algorithmes :

- **Algo de la 1ère zone libre (first fit)**: on recherche la 1re unité libre de taille suffisante. On peut améliorer cet algorithme en reprenant la recherche à partir du point où on s'était arrêté la fois précédente et non du début.
- **Algo de la zone libre suivante** Il est identique au précédent mais il mémorise en plus la position de l'espace libre trouvé. La recherche suivante commencera à partir de cette position et non à partir du début.

CHOIX DE LA ZONE POUR UN NOUVEAU PROCESSUS

- **Algo de meilleur ajustement (best fit)**: : on cherche l'unité libre dont la taille est la plus proche de celle demandée.

Cela défavorise les petits processus mais c'est compensé par le fait qu'ils peuvent être choisis pour des unités petites alors que les autres ne peuvent pas.

Certains SE donnent du bonus aux processus qui n'ont pas démarré pour cette raison => quand ils ont accumulé assez de bonus ils démarrent même s'ils utilisent une unité peu adéquate.

CHOIX DE LA ZONE POUR UN NOUVEAU PROCESSUS

- **Algo du Plus grand résidu ou pire ajustement (worst fit)** : Il consiste à prendre toujours la plus grande zone libre pour que la zone libre restante soit la plus grande possible. Les simulations ont montré que cette stratégie ne donne pas de bons résultats.
- **Algo rapide** : on gère deux listes d'unités libres selon leur taille => on parcourt directement la bonne liste et on prend la 1re unité même si sa taille n'est pas optimale. La recherche est plus rapide, mais lors de la libération il faut examiner les voisins afin de voir si une fusion est possible, sinon on aura une fragmentation en petites zones inutilisables.

ESPACE D'ADRESSAGE LOGIQUE OU PHYSIQUE

- L'unité centrale manipule des adresses logiques (emplacement relatif).
- Les programmes ne connaissent que des adresses logiques, ou virtuelles.
- L'espace d'adressage logique (virtuel) est donc un ensemble d'adresses pouvant être générées par un programme.
- L'unité mémoire manipule des adresses physiques (emplacement mémoire). Elles ne sont jamais vues par les programmes utilisateurs.
- L'espace d'adressage physique est un ensemble d'adresses physiques correspondant à un espace d'adresses logiques.

LA MÉMOIRE VIRTUELLE

- Les adresses manipulées par les instructions étant des adresses virtuelles, la MMU (Memory Management Unit) se charge de les transformer en adresses physiques.
- On dispose de beaucoup plus d'adresses virtuelles qu'il n'y a de mémoire physique.
- Lors de cette correspondance on peut détecter qu'une adresse virtuelle ne correspond à aucune adresse physique, dans ce cas la MMU génère une interruption qui sera prise par le SE pour gérer cette absence. Le SE utilise le disque pour compléter la mémoire (swap area).

LA MÉMOIRE VIRTUELLE

- Le SE va :

1. choisir une page de mémoire à libérer
2. transférer le contenu de cette page sur le disque (swap area)
3. transférer dans la page libérée la zone disque correspondant à l'adresse virtuelle demandée.

- La mémoire virtuelle est donc une technique qui permet d'exécuter des programmes dont la taille excède la taille de la mémoire réelle.

LA MÉMOIRE VIRTUELLE

- Il serait trop coûteux d'attribuer à tout processus un espace d'adressage complet, surtout parce que beaucoup n'utilisent qu'une petite partie de son espace d'adressage.
- En général, la mémoire virtuelle et la mémoire physique sont structurées en unités d'allocations (pages pour la mémoire virtuelle et cadres pour la mémoire physique). La taille d'une page est égale à celle d'un cadre.
- Lorsqu'un processus est en cours d'exécution, seule une partie de son espace d'adressage est en mémoire.

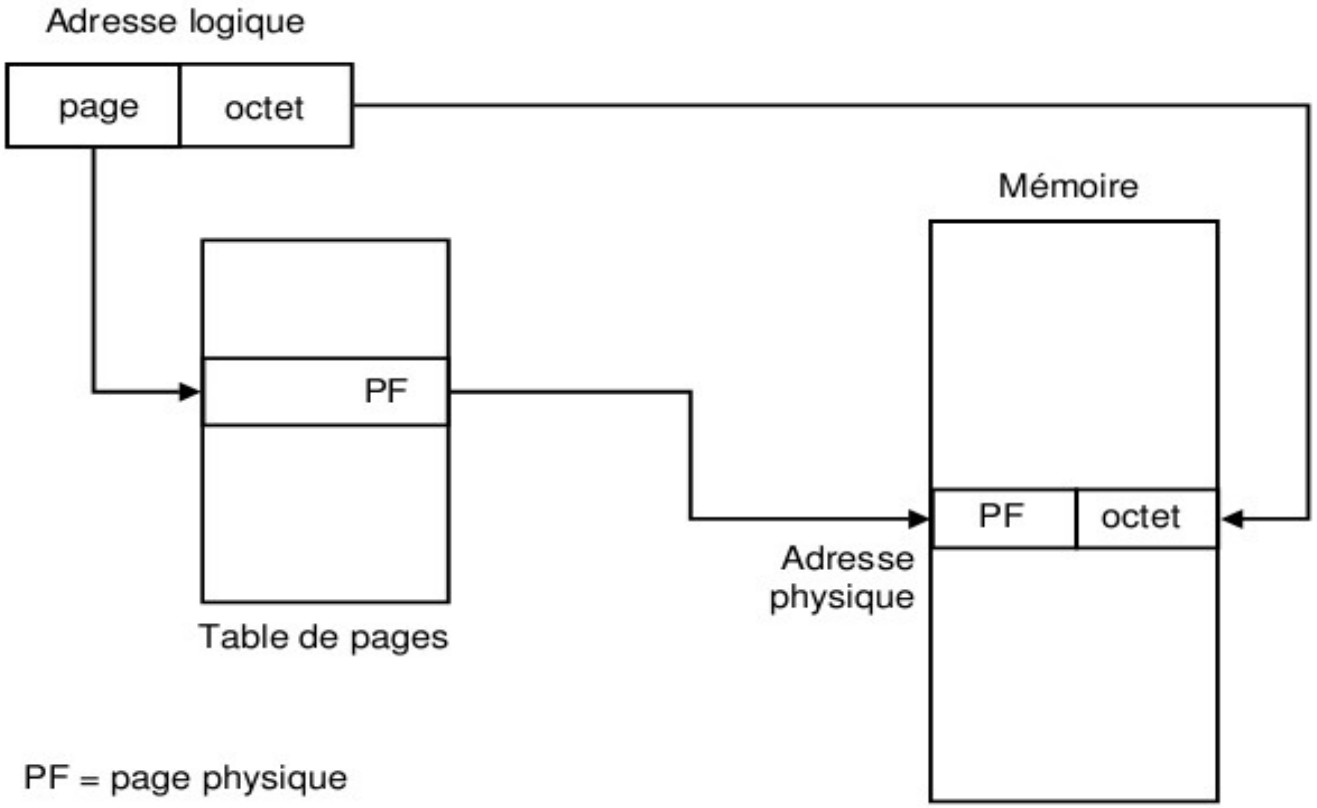
LA MÉMOIRE VIRTUELLE

- Cette conversion des adresses virtuelles en adresses physiques est effectuée par le MMU un composant matériel.
- Si cette adresse correspond à une adresse en mémoire physique, on transmet sur le bus l'adresse réelle, sinon il se produit un défaut de page.
- Par exemple, si la mémoire physique est de 32 Ko et l'adressage est codé sur 16 bits, l'espace d'adressage logique peut atteindre la taille 2^{16} soit 64 Ko.
- L'espace d'adressage est structuré en un ensemble d'unités appelées pages ou segments, qui peuvent être chargées séparément en mémoire.

PAGINATION

- La mémoire virtuelle et la mémoire physique sont structurées en unités d'allocations appelés pages pour la mémoire virtuelle et cases ou cadres pour la mémoire physique.
- La taille d'une page est fixe et égale à celle d'un cadre. Elle varie entre 2 Ko et 16 Ko.
- Un schéma de la traduction d'adresses lors de la pagination pure est montré sur la figure suivante

PAGINATION PURE

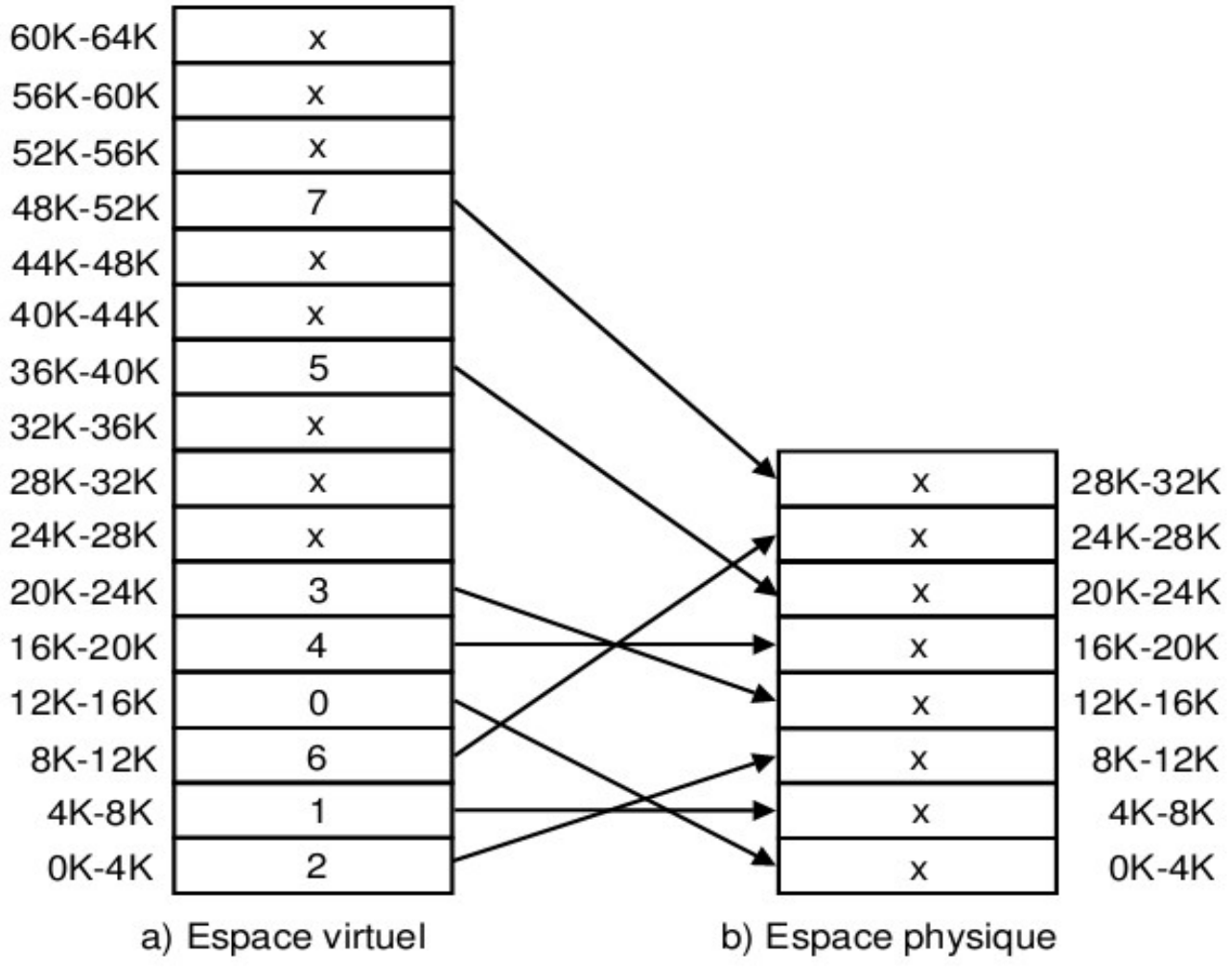


PAGINATION PURE

- Dans la pagination il n'y a pas de fragmentation externe car toutes les pages sont de même taille. Par contre, il peut y avoir une fragmentation interne si la dernière page de l'espace d'adressage logique n'est pas pleine.

EXEMPLE 1

- Soit un programme de 64 Ko sur une machine 32 Ko de mémoire.
- Son espace d'adressage logique est sur le disque. Il est composé de 16 pages de 4 Ko.
- La mémoire physique est structurée en 8 cadres de 4Ko



EXEMPLE 1

- Quand le programme tente d'accéder à l'adresse 0 (par exemple avec l'instruction **mov AX, $0**),
  - l'adresse virtuelle 0 est envoyée au MMU (Memory Management Unit).
  - Le MMU constate que cette adresse se trouve dans la page 0 (0K - 4K), et la transformation donne la case 2 (8K - 12K) dans la mémoire physique.
  - Cette valeur (8192) est mise dans le bus du système.
  - Ainsi les adresses virtuelles entre 0 et 4095 sont transformées en adresses physiques comprises entre 8192 à 12287.

UNITÉ DE GESTION DE LA MÉMOIRE (MMU)

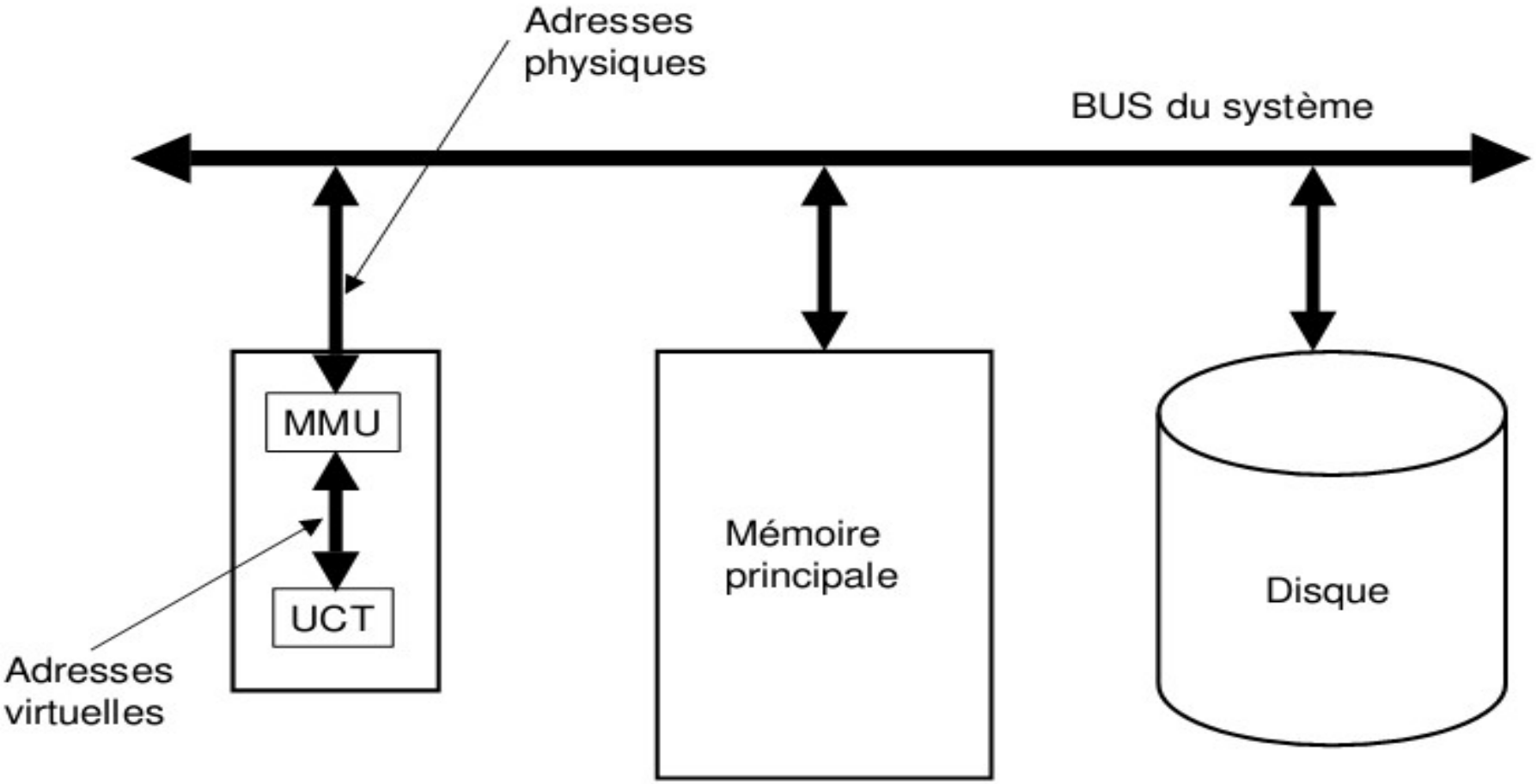
- Les adresses virtuelles, générées par les compilateurs et les éditeurs de liens, sont des couples composés d'un numéro de page et d'un déplacement relatif au début de la page.
- Les adresses virtuelles référencées par l'instruction en cours d'exécution doivent être converties en adresses physiques.
- Cette conversion d'adresse est effectuée par le MMU, qui sont des circuits matériels de gestion.
- La correspondance entre les pages et les cases est mémorisée dans une table appelée Table de pages (TP).

TABLE DES PAGES

Dans chaque entrée d'une table de pages de 2ème niveau on trouve :

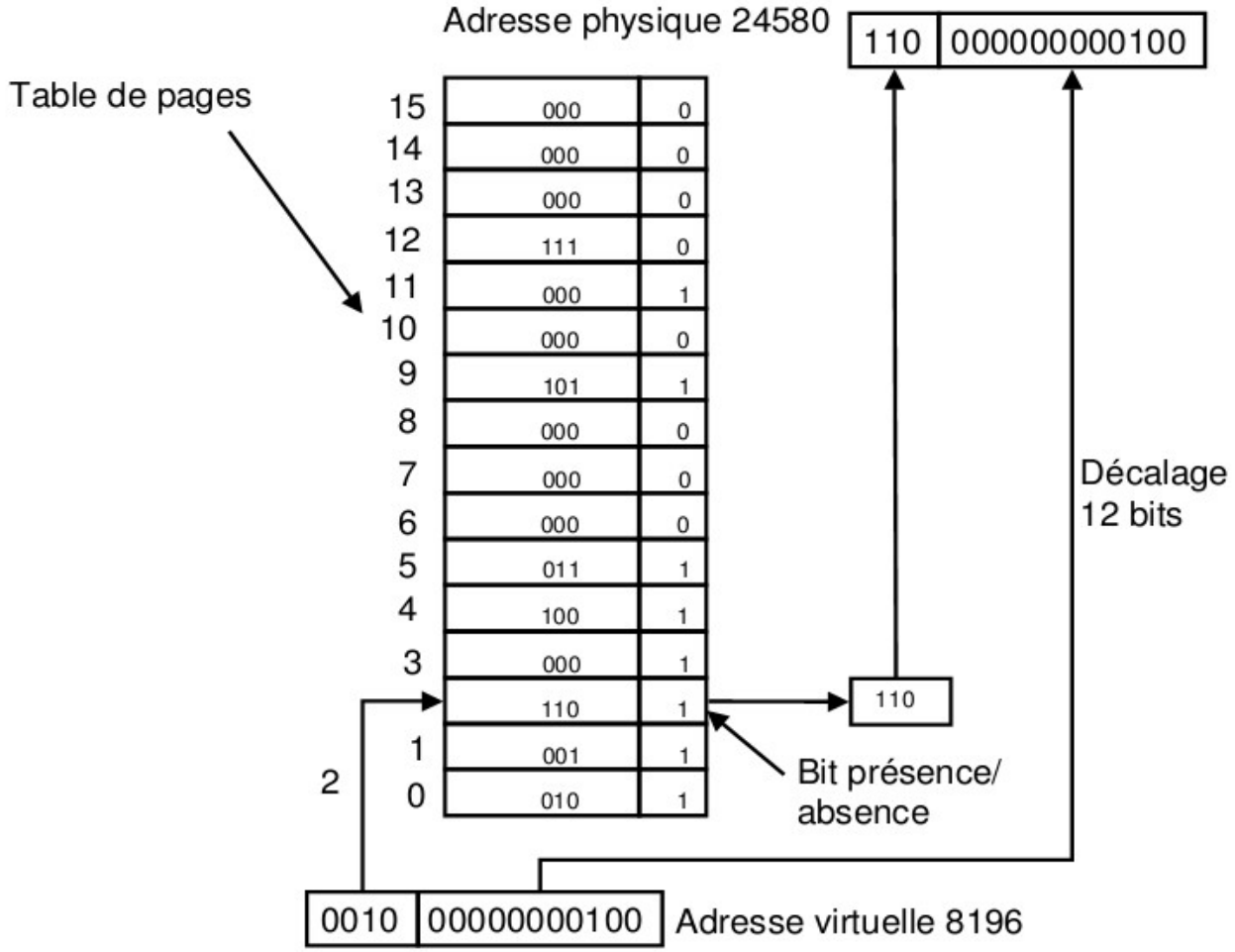
1. L'adresse de la page en mémoire physique
2. Un bit de présence (la page est ou pas présente en mémoire)
3. Des bits de protection (du genre rwx)
4. Des bits indiquant l'utilisation de la page (modifiée, référencée). Ils serviront lors des changements de page (modifiée => à écrire sur disque avant libération, référencée => à ne pas libérer si possible).
5. Un bit pour inhiber le cache, permet de ne pas utiliser la mémoire cache pour cette page.

UNITÉ DE GESTION DE LA MÉMOIRE (MMU)



FONCTIONNEMENT D'UN MMU

- Le MMU reçoit, en entrée une adresse virtuelle et envoie en sortie l'adresse physique ou provoque un déroutement.
- Dans l'exemple de la figure ci-contre, l'adresse virtuelle est codée sur 16 bits.



FONCTIONNEMENT D'UN MMU

- Les 4 bits de poids fort indiquent le numéro de page, comprise entre 0 et 15.
- Les autres bits donnent le déplacement dans la page, entre 0 et 4095.

- Le MMU examine l'entrée dans la Table de pages correspondant au numéro de page, dans ce cas 2.
- Si le bit de présence est à 0, la page n'est pas en mémoire alors le MMU provoque un déroutement. Sinon, il détermine l'adresse physique en recopiant dans les 3 bits de poids le plus fort, le numéro de case (110) correspondant au numéro de page (0010), et dans les 12 bits de poids le plus faible de l'adresse virtuelle.
- L'adresse virtuelle 8196 (0010 0000 0000 0100) est convertie alors en quoi?

  en adresse physique 24580 (1100 0000 000 0100) comme sur la figure.

Le SE maintient une copie de la table de pages pour processus, qui permet d'effectuer la translation des adresses.

TABLE DES PAGES A PLUSIEURS NIVEAUX

- La table des pages peut devenir très grande => difficile à garder en mémoire.
- On peut résoudre ce problème en utilisant une table des pages à 2 niveaux ou plus.
- L'intérêt est qu'on ne garde en mémoire que la table de 1er niveau et les quelques tables de 2ème niveau utilisées.
- Par exemple, une table de pages à deux niveaux, pour un adressage sur 32 bits et des pages de 4 Ko, est composée de 1024 tables de 1 Ko.

TABLE DES PAGES A PLUSIEURS NIVEAUX

- Il est ainsi possible de charger uniquement les tables de 1 Ko nécessaires. Dans ce cas, une adresse virtuelle de 32 bits est composée de trois champs : un pointeur sur la table du 1er niveau, un pointeur sur une table du 2ème niveau et un déplacement dans la page, de 12 bits.
- **Exemple**
  - Si l'on reprend l'exemple ci-haut l'UC avec des adresses logiques à 32 bits, des pages à 4 Ko avec des entrées à 4 octets et si le schéma proposé est de deux niveaux avec 10 bits d'adresses dans chaque niveau.
  - La table de pages de premier niveau aurait une taille de 4 Ko (2^{10}entrées de 4 octets) qui pointeraient vers 1024 tables de pages de deuxième niveau. A leur tour, chaque table de pages de deuxième niveau va utiliser 4 Ko (2^{10} entrées de 4 octets) qui pointeraient vers 1024 cadres. Chaque table de deuxième niveau aura un espace d'adressage de 4 Mo (1024 cadres * 4 Ko).

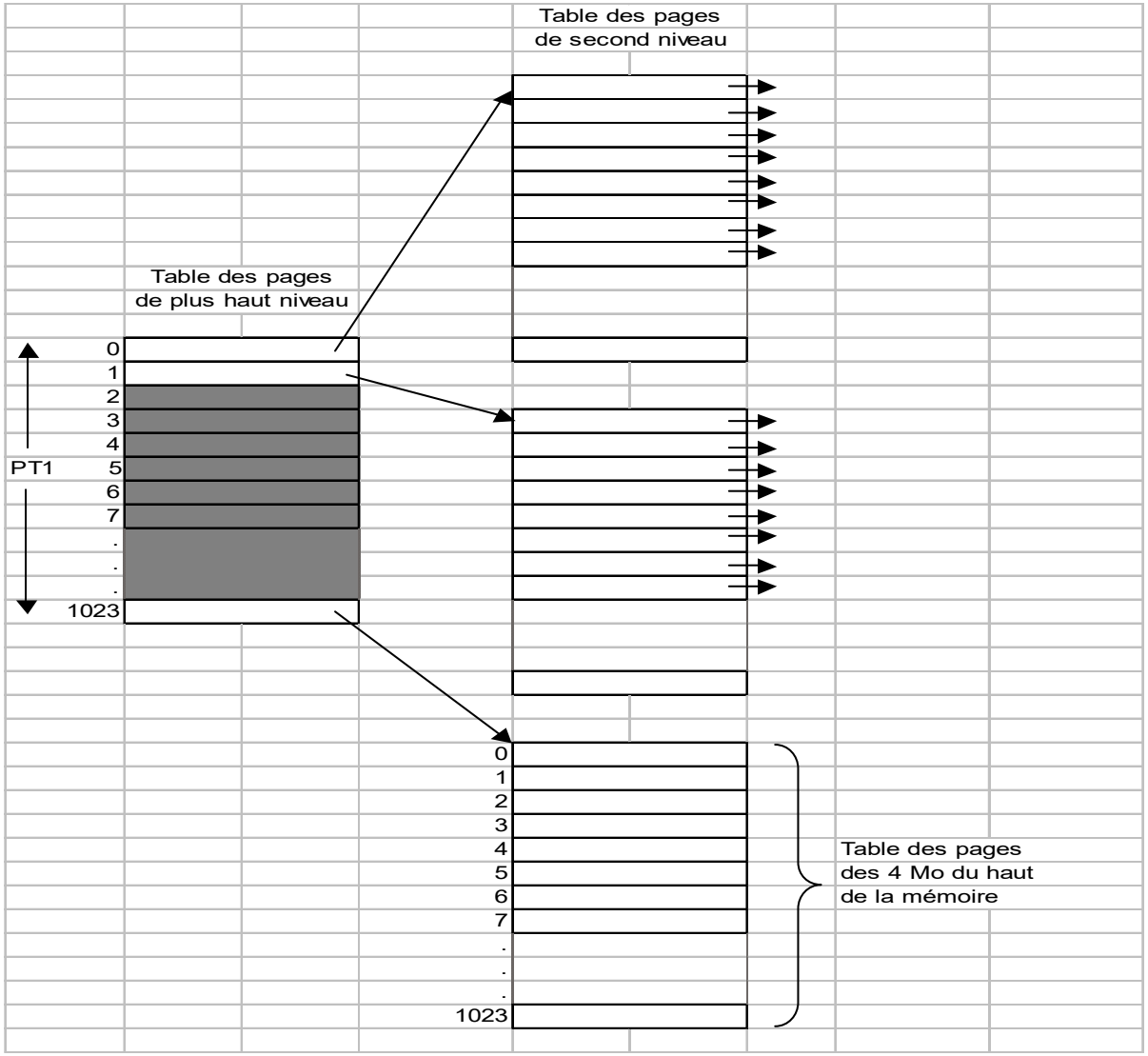
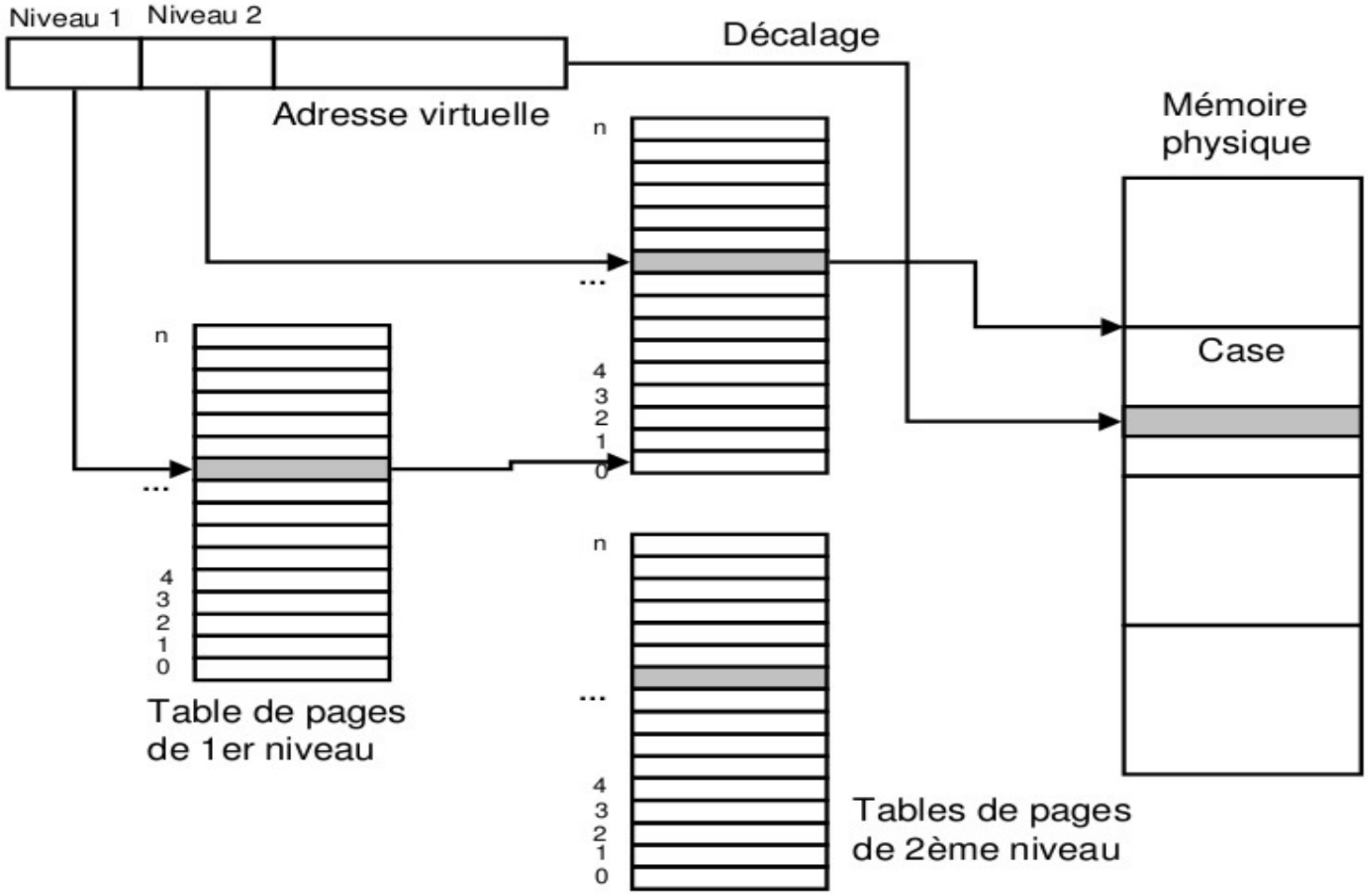


Table des pages
de second niveau
Table des pages
de plus haut niveau
PT1
0
1
2
3
4
5
6
7
.
.
.
1023
0
1
2
3
4
5
6
7
.
.
.
1023
Table des pages
des 4 Mo du haut
de la mémoire

TABLE DES PAGES



PROBLÈME DE TAILLE DE LA TABLE DES PAGES

- Lorsque la taille des adresses virtuelles augmente la taille et le nombre de tables de pages augmente aussi.
- Exemple : adresse virtuelle sur 64 bits si on fait des pages de 4Ko on en a 252 possibles même en admettant que chaque entrée de table de page soit sur 8 bits les tables occuperaient 4500 tera octets !
- Certes toutes les adresses virtuelles possibles ne sont pas utilisées et donc toutes les tables de 2ème niveau ne sont pas présentes en mémoire vive mais le problème de la place occupée reste important.

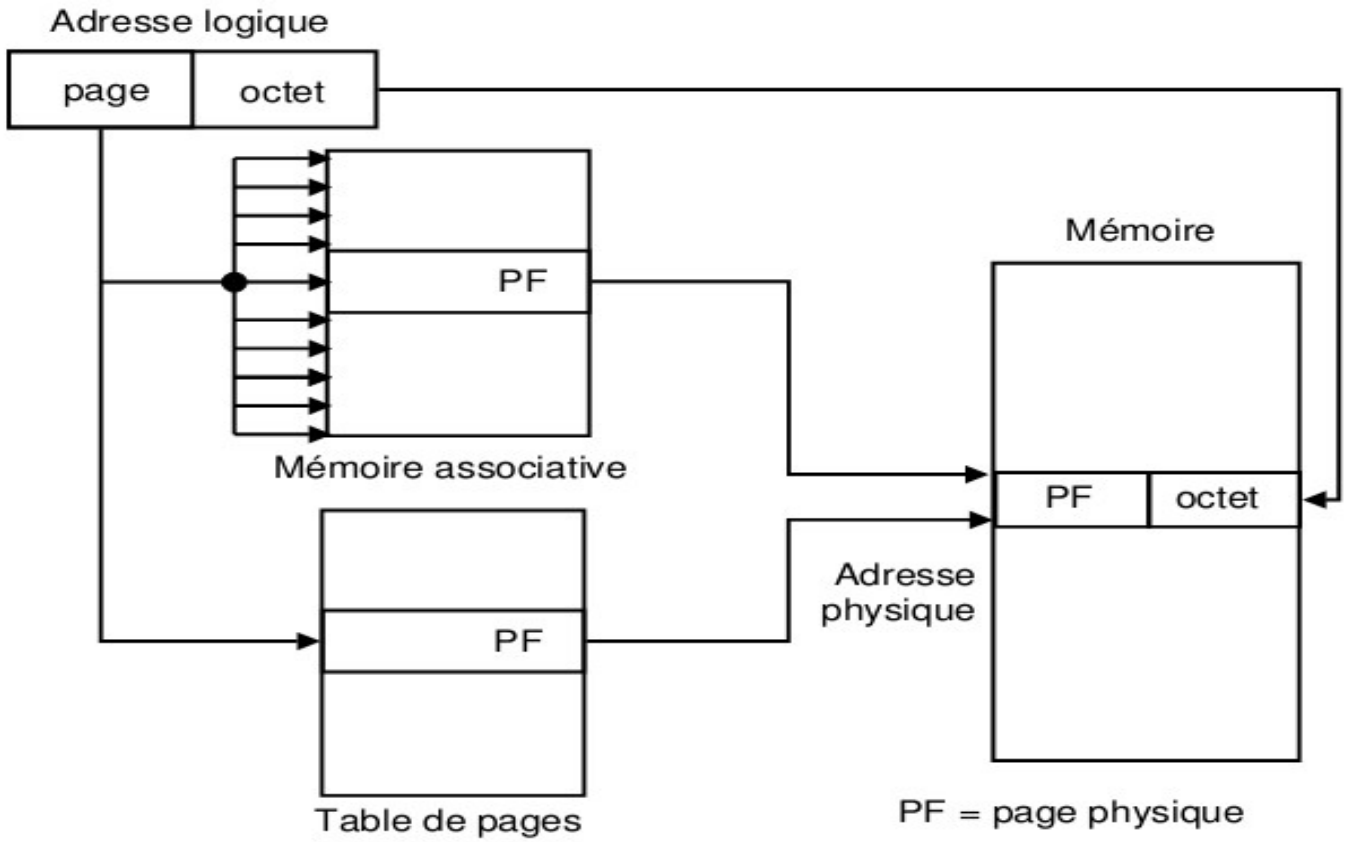
PROBLÈME DE TAILLE DE LA TABLE DES PAGES

- Solution : L'une des approches modernes de ce problème est de faire des tables inversées càd qu'il existe une entrée de table par page de mémoire physique. La mémoire physique étant plus petite, la table reste de taille raisonnable
- Problème : la transformation @virtuelle→@physique suppose de parcourir la table des pages jusqu'à trouver la bonne entrée ou constater qu'elle n'y est pas => c'est moins rapide (d'où l'utilisation de la mémoire associative).

MÉMOIRE ASSOCIATIVE

- Pour accélérer le processus de passage adresse virtuelle → adresse physique, le matériel propose en général une petite **mémoire associative** appelée TLB (Translation Lookaside Buffer) qui conserve les entrées les plus utilisées des tables de pages.
- Ceci évite d'accéder à la mémoire pour aller les chercher.

MÉMOIRE ASSOCIATIVE



REMPLACEMENT DE PAGE

Le SE a en charge de remplacer les pages de la mémoire physique. On trouve plusieurs algorithmes de remplacement des pages :

- Remplacement de page optimal (Algorithme de Belady)
- NRU (Non Récemment Utilisé)
- FIFO (First In First Out)
- Algorithme de la seconde chance
- LRU (Least Recently Used)

REMPLACEMENT DE PAGE OPTIMAL

- Algorithme de Remplacement de page optimal ou Algorithme de Belady
- L'algorithme optimal consiste à retirer la page qui sera référencée le plus tard possible dans le futur.
- Cette stratégie est impossible à mettre en oeuvre car il est difficile de prévoir les références futures d'un programme.
- Le remplacement de page optimal a été cependant utilisé comme base de référence pour les autres stratégies, car il minimise le nombre de défauts de page.
- Exemple 3. Soit un système avec cases de mémoire et une suite de références ={7,0,1,2,0,3,0,4,2,3,0,3,2,1,2,0,1,7,0,1}.

REMPLACEMENT DE PAGE OPTIMAL

- À la figure suivante on montre une ligne pour chaque case et une colonne pour chaque référence.
- Chaque élément de ligne i et colonne j montre la page chargée dans la case i après avoir été référencée.
- Les entrées en noir sont des pages chargées après un défaut de page.
- L'algorithme optimale fait donc 9 défauts de page.

m \ ω	7	0	1	2	0	3	0	4	2	3	0	3	2	1	2	0	1	7	0	1
0	**7**	7	7	**2**	2	2	2	2	2	2	2	2	2	2	2	2	2	**7**	7	7
1		**0**	0	0	0	0	0	**4**	4	4	**0**	0	0	0	0	0	0	0	0	0
2			**1**	1	1	**3**	3	3	3	3	3	3	3	**1**	1	1	1	1	1	1

NRU (NON RÉCEMMENT UTILISÉ)

- Le NRU utilisé des statistiques sur les pages
- Pour effectuer des statistiques sur les pages utilisées, on associe à chaque page 2 bits d'information :
  - Le bit R est positionné chaque fois qu'une page est référencée
  - Le bit M est positionné quand on écrit dans une page
- Ces bits sont modifiés lors des accès à la page : (référencée ←1 et modifiée ←1 si écriture). Le bit référencé est remis à 0 périodiquement par le SE.
- NRU recherche une page NON référencé ET NON modifiée (00) et l'enlève (une au hasard s'il y en a plusieurs). S'il n'en trouve pas il cherche NON référencé ET modifiée (01) puis référencée ET NON modifiée (10) puis référencée ET modifiée (11).

FIFO (FIRST IN FIRST OUT)

- Le SE mémorise une liste de toutes les pages en mémoire, la première page de cette liste étant la plus ancienne et la dernière la plus récente ;
- lorsqu'il se produit un défaut de page, on retire la première page de la liste et on place la nouvelle page à la fin de la liste
- Cet algorithme fournit de piètre performance

FIFO (FIRST IN FIRST OUT)

- **Exemple** . La suite de références ={7,0,1,2,0,3,0,4,2,3,0,3,2,1,2,0,1,7,0,1} avec m=3 cases.
- Donner les différentes étapes de remplacement des pages et le nombre de défauts de page avec l'algorithme FIFO.

m \ ω	7	0	1	2	0	3	0	4	2	3	0	3	2	1	2	0	1	7	0	1
0	**7**	7	7	**2**	2	2	2	**4**	4	4	**0**	0	0	0	0	0	0	**7**	7	7
1		**0**	0	0	0	**3**	3	3	**2**	2	2	2	2	**1**	1	1	1	1	**0**	0
2			**1**	1	1	1	**0**	0	0	**3**	3	3	3	3	**2**	2	2	2	2	**1**

- Cet algorithme FIFO fait15 défauts de page (ce qui est en gras sur la figure).

ALGORITHME DE LA SECONDE CHANCE

- Une variante de FIFO
- On classe les pages par ordre d'allocation et on cherche la page la plus ancienne ayant le bit référencée à 0. Pendant cette recherche toutes les pages qui la précèdent (donc plus anciennes mais référencées) sont mises en fin de liste (récentes) mais marquées comme non référencées.
- L'algorithme de la seconde chance cherche donc une ancienne page qui n'a pas été référencée au cours du dernier top d'horloge
- Si toutes les pages ont été référencées, l'algo de la seconde chance est équivalent au FIFO, puisque les bits de chaque page seront mis à 0 et la plus ancienne page sera remplacée

LRU (LEAST RECENTLY USED)

- Le LRU se fonde sur l'observation que les pages les plus référencées au cours des dernières instructions seront probablement très utilisées au cours des prochaines
- A l'inverse, les pages qui n'ont pas été référencées pendant un long moment ne seront pas probablement demandées pendant un certain temps
- LRU remplace la page qui n'a pas été référencée depuis le plus longtemps.

LRU (LEAST RECENTLY USED)

- L'algorithme LRU mémorise dans une liste chaînée toutes les pages en mémoire. La page la plus utilisée est en tête de liste et la moins utilisée est en queue.
- Lorsqu'un défaut de page se produit, la page la moins utilisée est retirée. Pour minimiser la recherche et la modification de la liste chaînée, ces opérations peuvent être réalisées par le matériel. Cet algorithme est coûteux.
- **Exemple** La suite de références w={7,0,1,2,0,3,0,4,2,3,0,3,2,1,2,0,1,7,0,1} avec m=3 cases fait 12 défauts de page comme sur la figure suivante

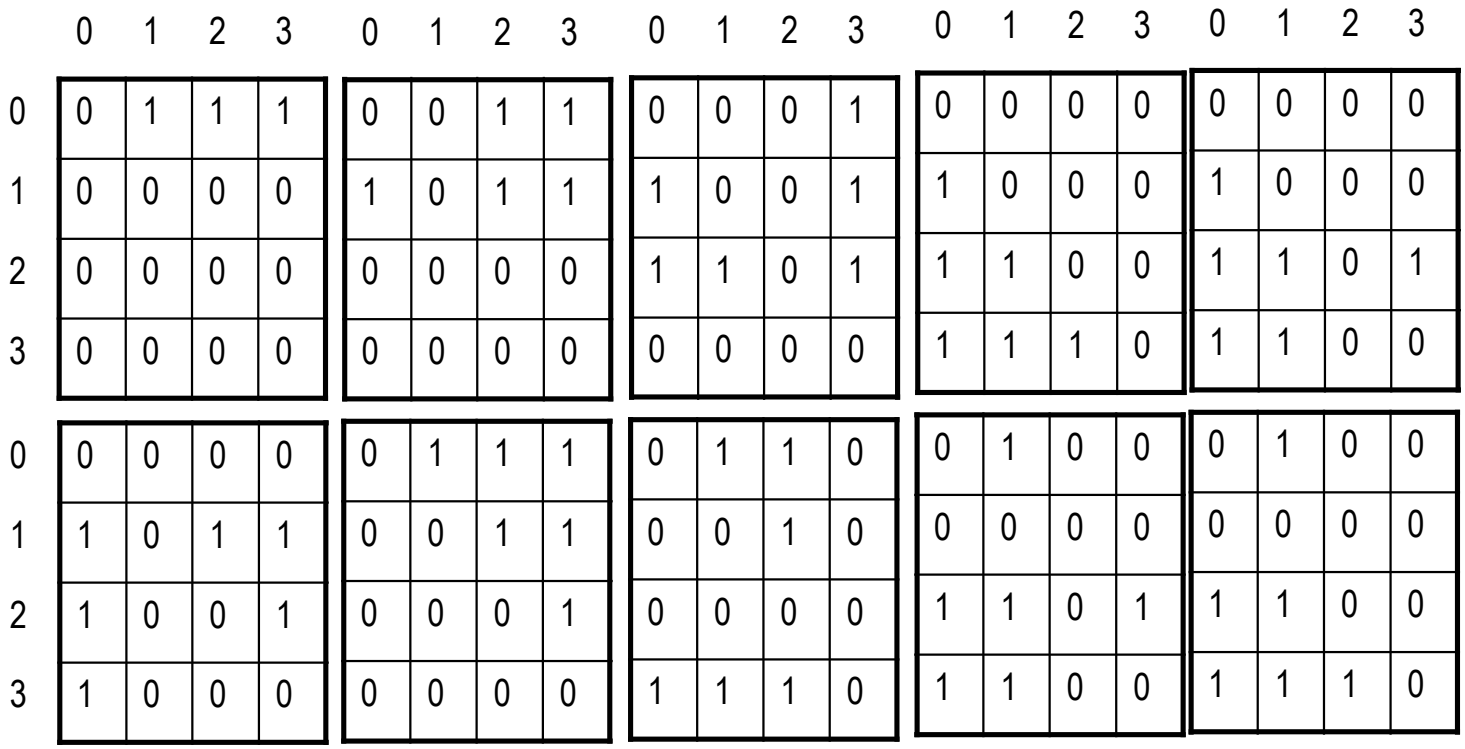
m \ ω	7	0	1	2	0	3	0	4	2	3	0	3	2	1	2	0	1	7	0	1
0	**7**	7	7	**2**	2	2	2	**4**	4	4	**0**	0	0	1	1	1	1	**1**	1	1
1		**0**	0	0	0	0	0	**0**	0	**3**	3	3	3	3	3	**0**	0	0	0	0
2			**1**	1	1	**3**	3	3	**2**	2	2	2	2	2	2	2	2	**7**	7	7

LRU (LEAST RECENTLY USED)

- Une solution matérielle de LRU consiste à disposer d'un matériel mémorisant une matrice de $n \times n$ bits initialement nulle
- Quand une page *k* est référencée, le matériel positionne tous les bits de la rangée *k* à 1 puis tous les bits de la colonne *k* à 0
- A tout instant la ligne contenant le moins de 1 indique la page la moins récemment utilisée

LRU (LEAST RECENTLY USED)

Exemple : 4 cases et les références aux pages suivantes : 0 1 2 3 2 1 0 3 2 3



OPTIMISATION DE LA GESTION DE LA MÉMOIRE

- Quand un processus démarre il ne dispose pas en mémoire physique de ses variables, de sa pile etc. => il va créer de nombreux défauts de pages.
- Au bout d'un moment il aura tout ce qu'il faut en mémoire et fonctionnera sans problèmes.
- Ceci est normal au démarrage d'un processus mais quand un processus est suspendu (bloqué) il va petit à petit être viré de la mémoire lui et les pages qu'il utilisait pour laisser de la place aux autres.
- Quand il va redémarrer (débloqué) il va à nouveau créer des défauts de pages.
- Les SE essayent d'éviter cela en chargeant d'avance, quand le processus redémarre, l'ensemble de son espace de travail tel qu'il était quand le processus a été suspendu.

LE RÔLE DU SE DANS LA GESTION DES PAGES

- Il y a 4 moments clés :

1. Création d'un processus
   - Déterminer la taille du programme et de ses données
   - Créer une table de pages de niveau 2 pour lui
   - Allouer de la mémoire physique
   - Prévoir de la place sur le disque pour ses pages
2. Exécution d'un processus
   - Mettre à jour la TLB pour ce processus
   - Charger en mémoire tout ou partie de l'espace de travail de ce processus

LE RÔLE DU SE DANS LA GESTION DES PAGES

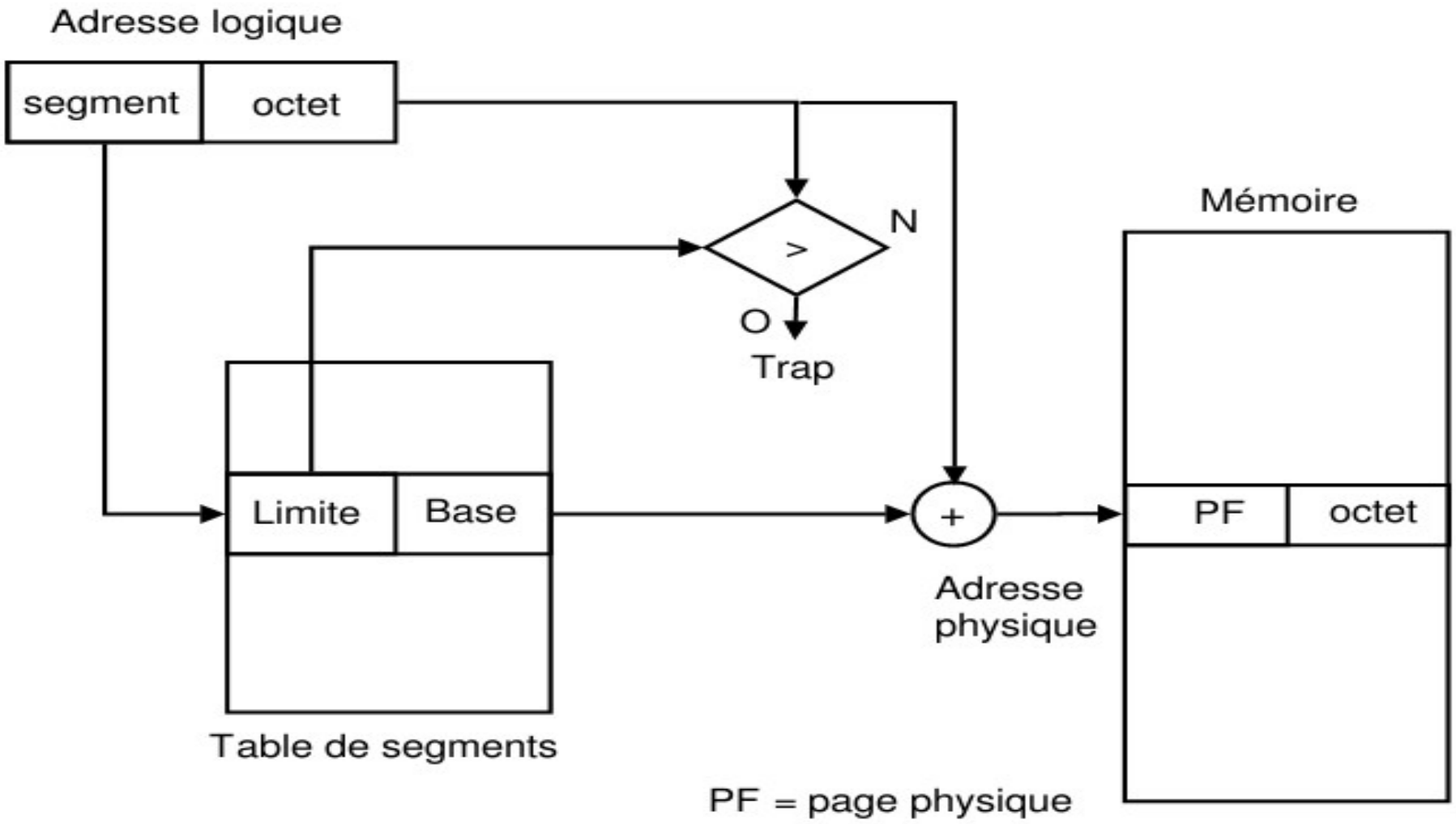
3. Fin d'un processus
   - Libérer sa table des pages
   - Libérer les pages en mémoire physique
   - Libérer la place sur le disque prise par ses pages
4. Défaut de page
   - Localiser sur disque la page correspondant à l'@ virtuelle ayant causé le défaut
   - Choisir une page en mémoire physique libre ou en libérer une
   - Copier sur disque la page libérée si elle a été modifiée
   - Charger la page du disque dans cette page de mémoire libre
   - Reculer le CO du processus pour qu'il ré exécute l'instruction ayant provoqué le défaut de page.

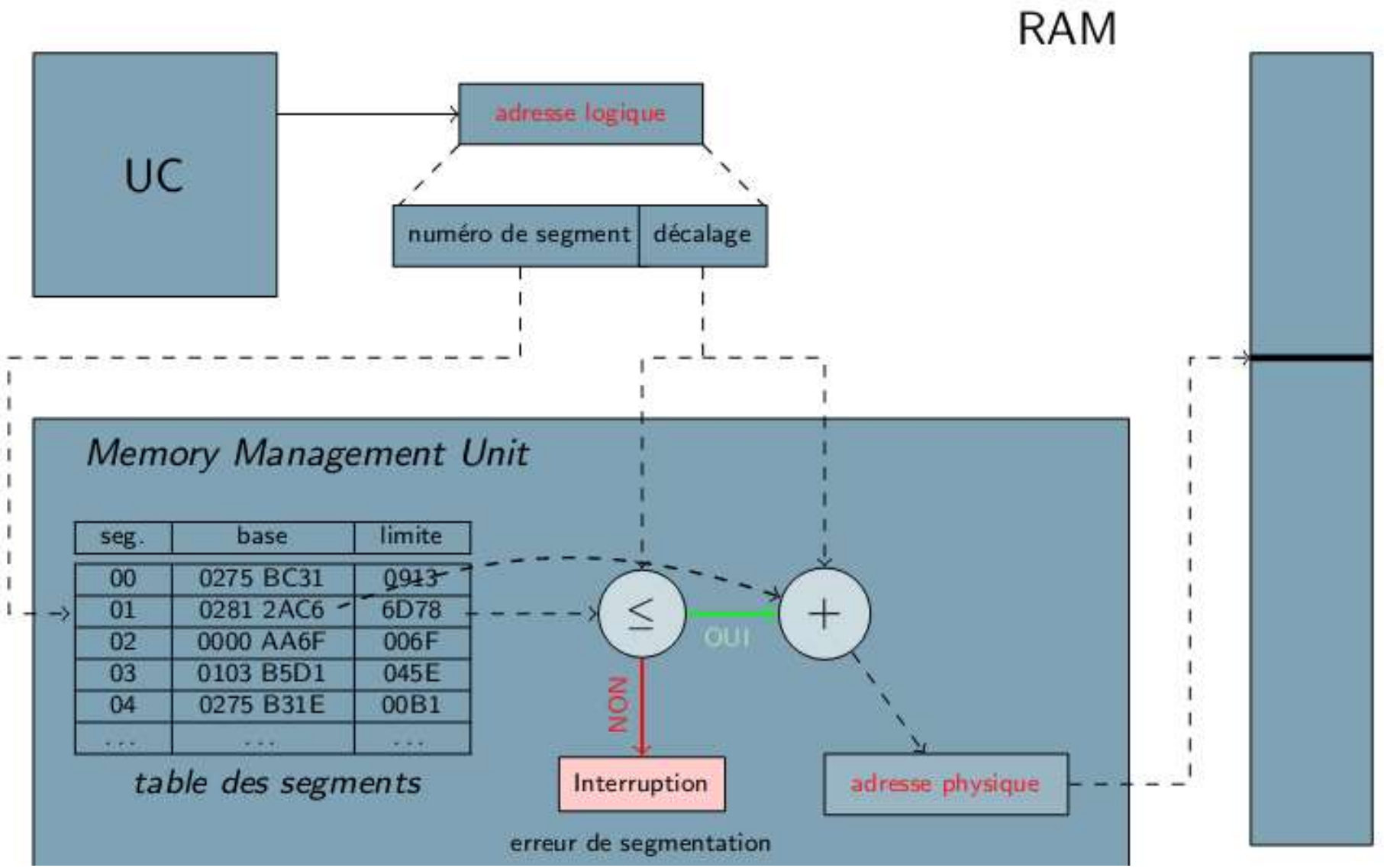
SEGMENTATION

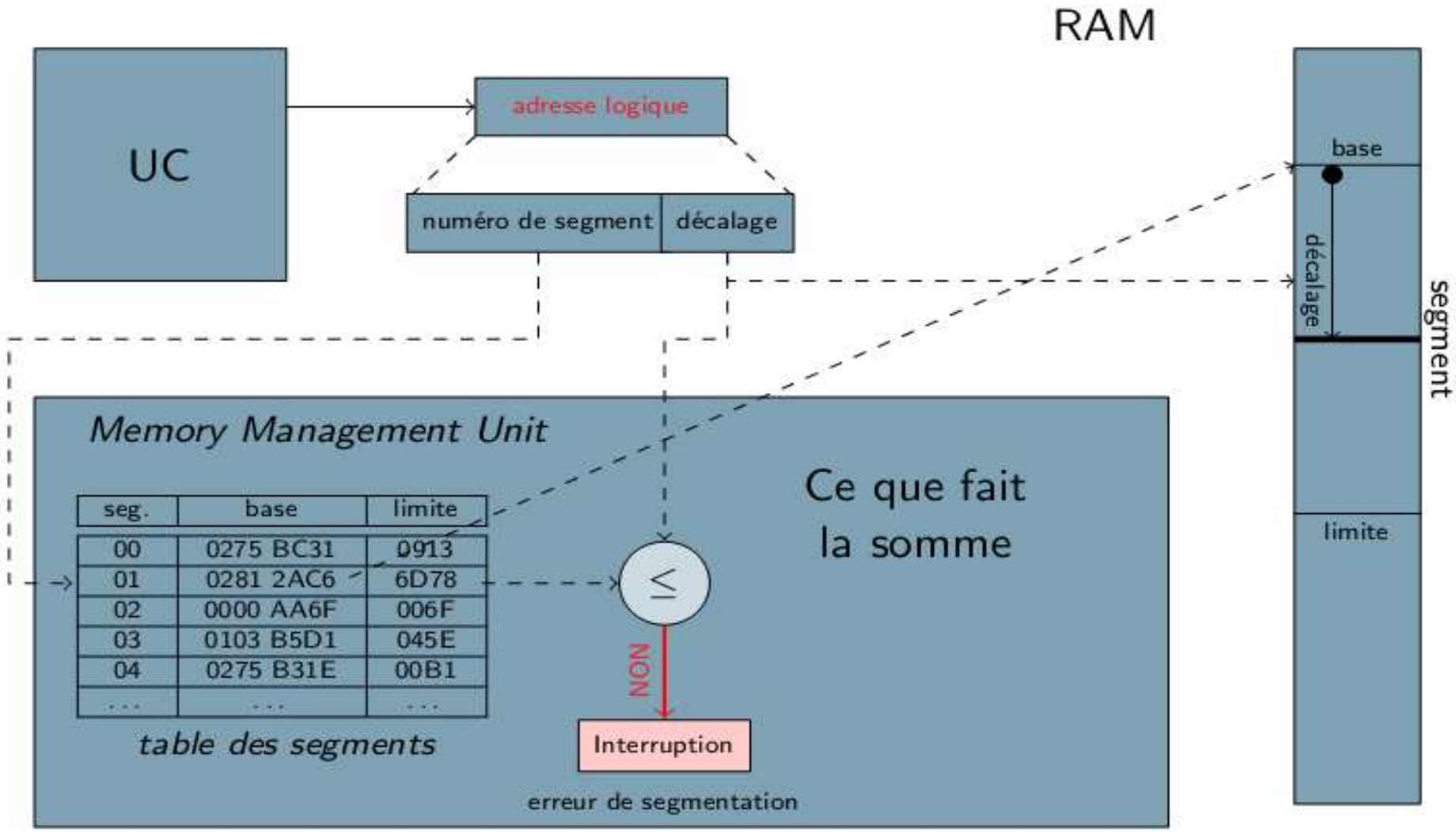
- L'idée de la technique de segmentation est d'avoir un espace d'adressage à deux dimensions.
- On peut associer à chaque unité logique un espace d'adressage appelé segment. L'espace d'adressage d'un processus est composé d'un ensemble de segments.
- Ces segments sont de tailles différentes (fragmentation externe).
- Gérée au niveau de la MMU

SEGMENTATION : PRINCIPE DE RÉSOLUTION

- Numéro de segment (selecteur) permet de déterminer l'adresse de base
- Base + décalage → adresse physique
- Table des segments aussi appelée table des descripteurs
- Pour chaque segment :
  - Base = adresse physique de départ du segment
  - Limite = taille du segment
- Un schéma de traduction d'adresses par segmentation est montré sur la figure suivante.


Adresse logique
segment
octet
>
N
O
Trap
Mémoire
Limite
Base
+
PF
octet
Adresse
physique
Table de segments
PF = page physique


RAM
UC
adresse logique
numéro de segment
décalage
Memory Management Unit
seg.
base
limite
00
0275 BC31
0913
01
0281 2AC6
6D78
02
0000 AA6F
006F
03
0103 B5D1
045E
04
0275 B31E
00B1
table des segments
≤
OUI
+
NON
Interruption
adresse physique
erreur de segmentation


RAM
UC
adresse logique
numéro de segment
décalage
base
décalage
segment
limite
Memory Management Unit
seg.
base
limite
00
0275 BC31
0913
01
0281 2AC6
6D78
02
0000 AA6F
006F
03
0103 B5D1
045E
04
0275 B31E
00B1
table des segments
≤
NON
Interruption
erreur de segmentation
Ce que fait
la somme

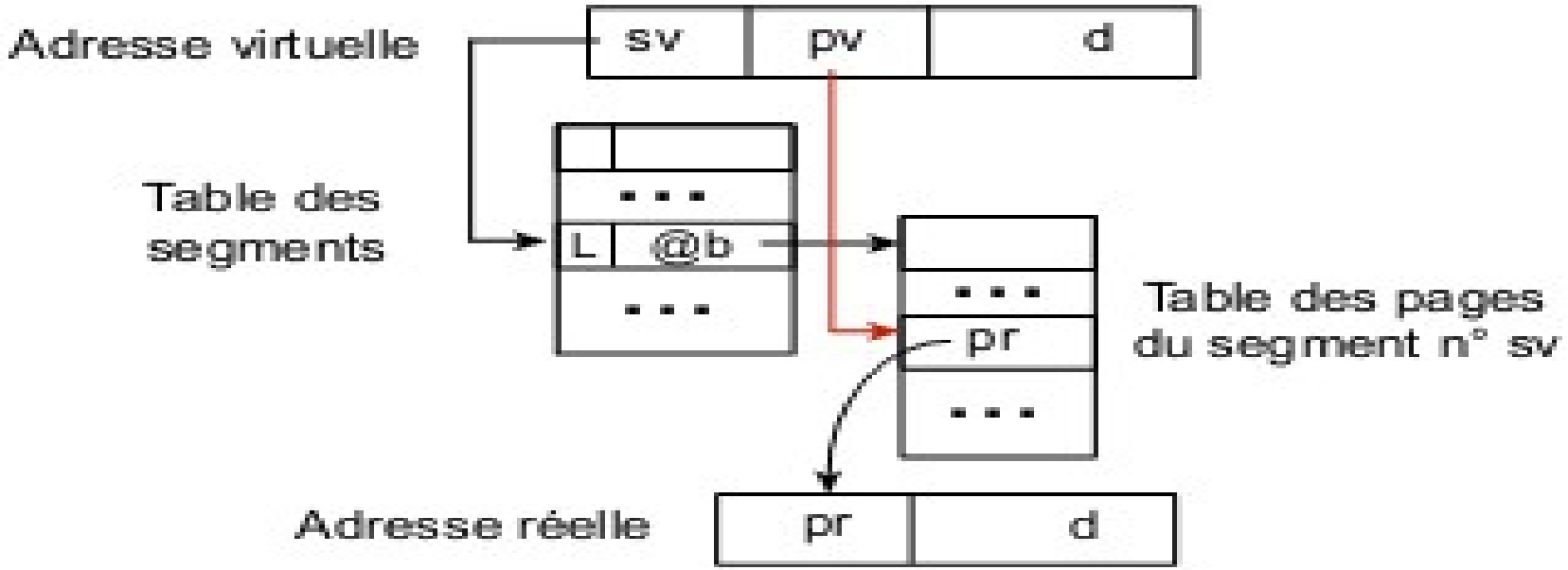
SEGMENTATION PAGINEE

- La segmentation peut être combinée avec la pagination. On parle de la segmentation paginée.
- La segmentation repose sur les règles suivantes :
  - L'espace d'adressage virtuel est découpé en blocs de taille fixe appelés segments;
  - Les segments sont découpés en pages ;
  - L'espace d'adressage réel est découpé en pages ;
- La mémoire réelle retrouve une structure de pages qui facilitera l'allocation
- Chaque segment est composé d'un ensemble de pages. Les adresses générées par les compilateurs et les éditeurs de liens, dans ce cas, sont alors des triplets :

<numéro du segment, numéro de page, déplacement dans la page>

SEGMENTATION PAGINEE

La conversion d'une telle adresse en adresse réelle (physique) repose sur les table des segments qui mémorise pour tout numéro de segment virtuel valide, l'adresse de début et la longueur effective du segment alloué en mémoire réelle.



ALLOCATION DE MÉMOIRE PAR UN PROCESSUS

- On a vu que quand un processus démarre il se voit allouer de la mémoire pour :
  - son code
  - sa pile
  - ses données
- Mais souvent un programme a besoin de plus de mémoire dont la taille est dynamiquement définie => il doit pouvoir demander au SE de la mémoire supplémentaire.
- La demande de mémoire se traduit dans les langages (C++) par **new** et sa libération par **free**

EXEMPLE D'UNIX

UNIX offre les primitives pour demander de la mémoire:

- **malloc** accepte un paramètre qui est la taille demandée et retourne un pointeur sur la zone allouée. Ce pointeur peut être NULL si l'allocation a échouée.
- **calloc** fait pareil mais initialise la zone allouée à 0
- **realloc** accepte un paramètre supplémentaire qui est un pointeur sur une zone déjà allouée. Son rôle est de modifier la taille de cette zone sans en perdre le contenu
- **free** accepte en paramètre un pointeur sur une zone déjà allouée et la libère.

EXEMPLE D'UNIX

- Problème de la taille de mémoire à demander : En général un programme alloue de la mémoire pour y mettre une variable et il ne connaît pas la taille en octets de cette variable mais il sait de quel type elle est.
- C propose une fonction **sizeof**(type) qui donne la taille en octets d'un type.
- Commande **vmstat** fourni des renseignements et des statistiques sur l'usage de la mémoire virtuelle du système.

FIN