

Université de Kairouan
Institut Supérieur d'Informatique et de Gestion
Département de l'Informatique

Cours
Systèmes d'Exploitation II
(Gestion de la mémoire)

Filière : 2ème S.I. & T.R.I

Enseigné par : Mlle Imen SFAIHI
Préparé par : Joëlle Delacroix

La gestion de la mémoire

1 – Technologies de mémoires (1/2)

- Il existe plusieurs technologies de mémoire:
  - à semi-conducteur (RAM statique ou dynamique, ROM, PROM, EPROM) ;
  - magnétiques (à bandes, disques) ;
  - optiques (disque optique, CD-ROM).
- Chacune de ces technologies est caractérisée par sa rapidité (temps d'accès), sa taille (capacité en octets) et son coût.
- La mémoire *centrale (MC), réalisée en RAM est d'une importance particulière car c'est* elle qui est la plus utilisée pendant l'exécution des programmes.
- Sa gestion doit être faite avec un maximum de rigueur.

La gestion de la mémoire

1 – Technologies de mémoires (2/2)

- Il y a deux façons de voir la mémoire au cours de l'exécution d'un Programme:
  - comme une *mémoire unique : un seul type de mémoire est* utilisé pour le stockage d'un programme pendant son exécution (c'est la mémoire centrale). C'est le cas des systèmes d'exploitation sur les premières machines et les micro-ordinateurs (MS/DOS en mode réel par exemple) ;
  - une *hiérarchie de mémoire : plusieurs types de mémoire sont* mis à contribution pendant l'exécution d'un programme: mémoire centrale, mémoire secondaire et éventuellement antémémoire (ou mémoire cache).

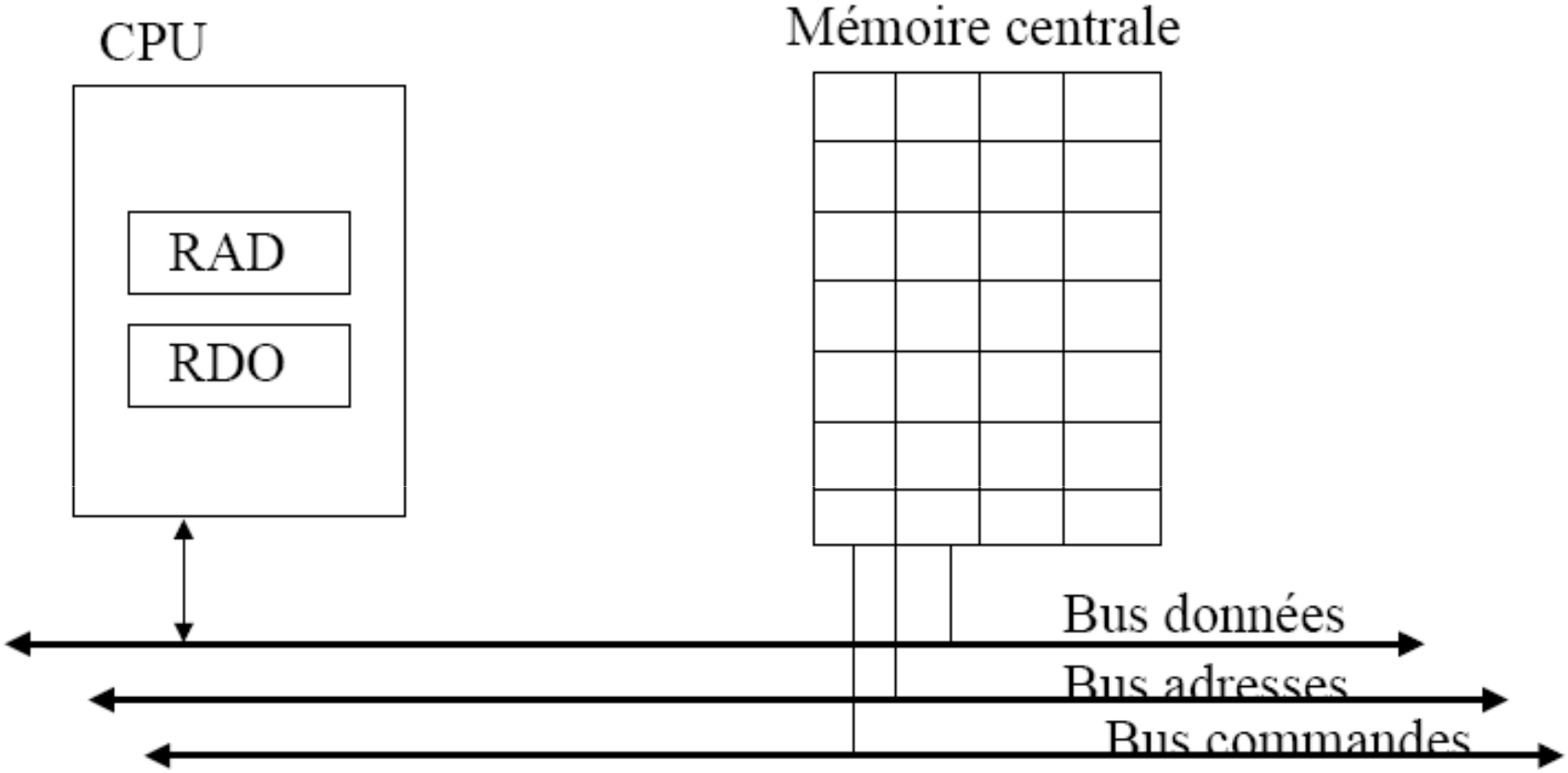
Gestion de mémoire: objectifs

- **Optimisation de l'utilisation de la mémoire principale = RAM**
- **Les plus grand nombre possible de processus actifs doit y être gardé, de façon à optimiser le fonctionnement du système en multiprogrammation**
  - **garder le système le plus occupé possible, surtout l'UCT**
  - **s'adapter aux besoins de mémoire de l'usager**
    - **allocation dynamique au besoin**

Gestion de la mémoire: concepts dans ce chapitre

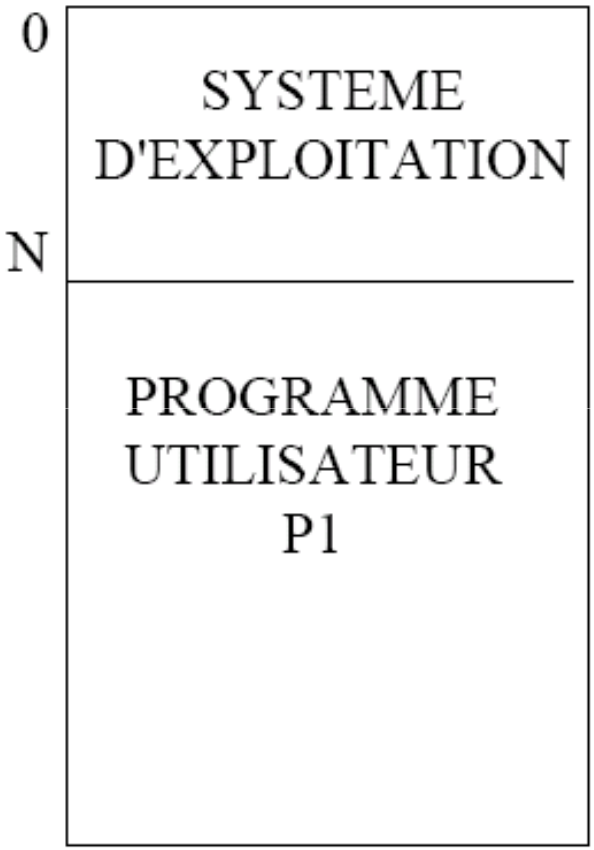
- **Adresse physique et adresse logique**
  - **mémoire physique et mémoire logique**
- **Remplacement**
- **Allocation contiguë**
  - **partitions fixes**
  - **variables**
- **Pagination**
- **Segmentation**
- **Segmentation et pagination combinées**

Fonction de mémorisation

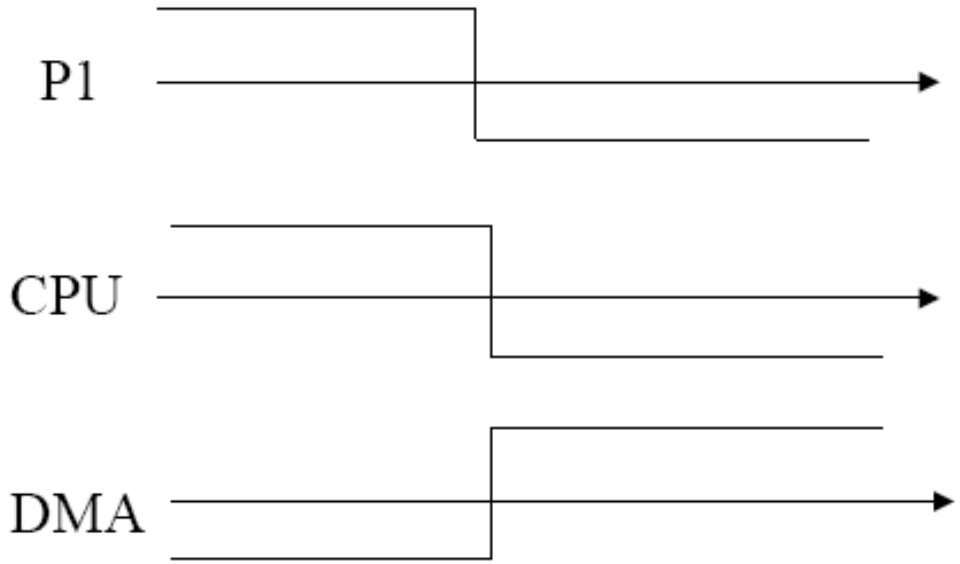


☞ ensemble linéaire de mots d'adresses contigues

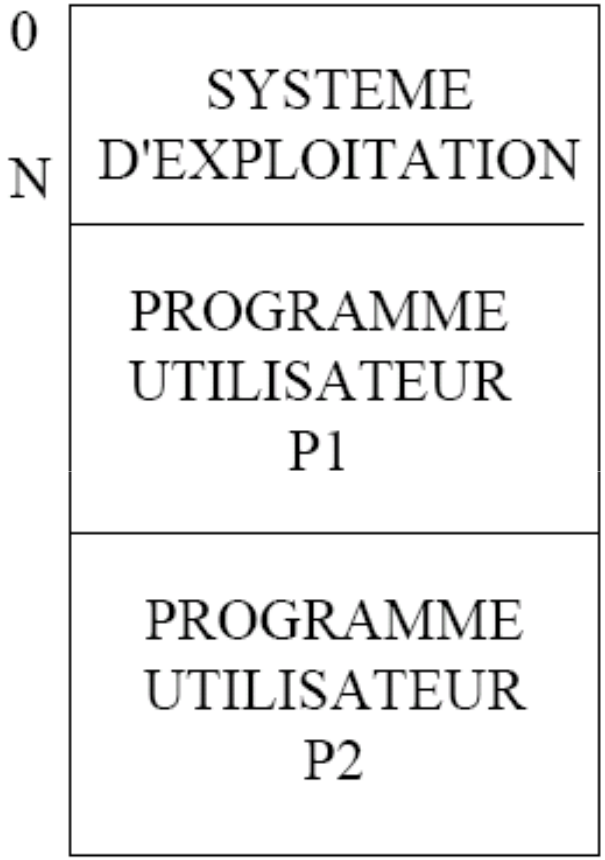
Monoprogrammation



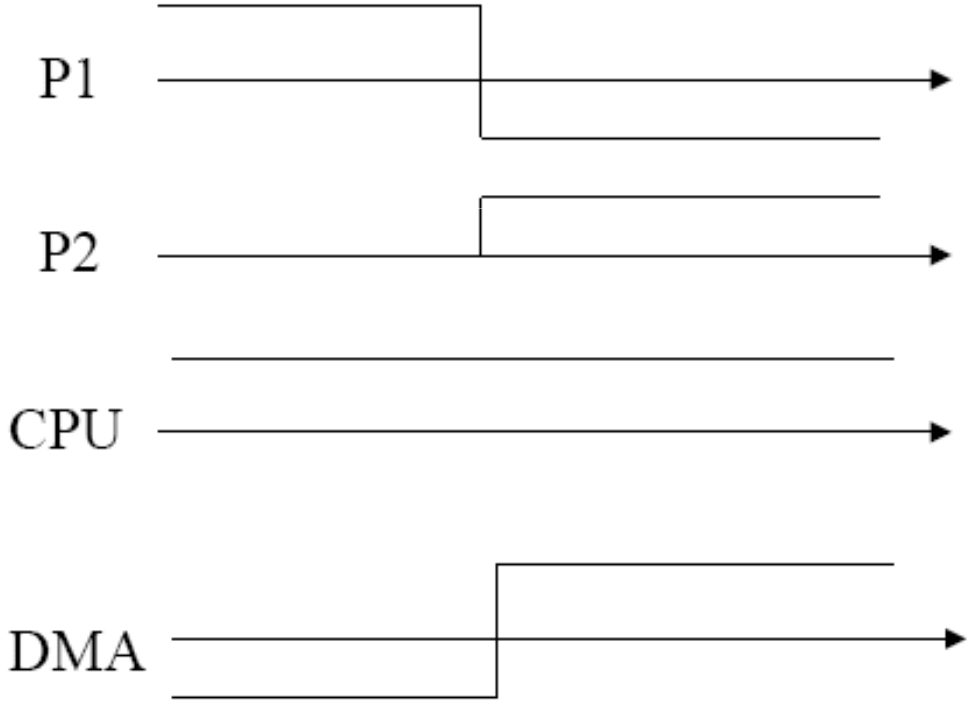
P1 : 100 ms calcul / 100 ms E/S
50 % d'inactivité du processeur



Multiprogrammation : cas idéal



P1 : 100 ms calcul / 100 ms E/S
P2 : 100 ms calcul / 100 ms E/S
0 % d'inactivité du processeur



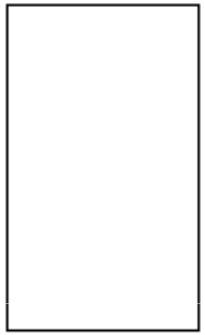
Multiprogrammation et gestion mémoire

- Trois problèmes à résoudre vis-à-vis de la mémoire :
  - Définir un espace d'adressage indépendant pour chaque processus
  - Protéger les espaces d'adressages des processus entre eux
  - Allouer de la mémoire physique à chaque espace d'adressage

Multiprogrammation et gestion mémoire

programme

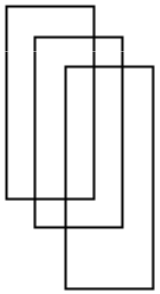
ensemble insécable de mots contigus



espace d'adressage linéaire

allocation en partitions
fixes ou variables

ensemble sécable de mots contigus

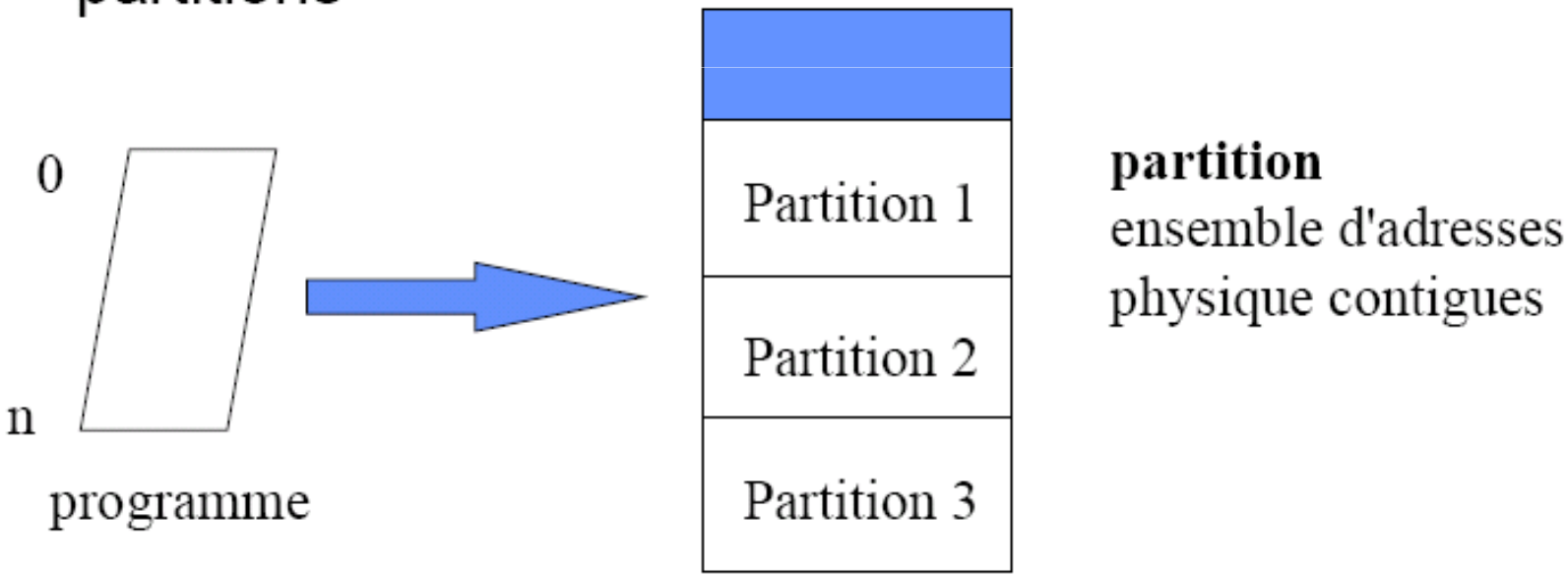


espace d'adressage paginé
espace d'adressage segmenté



Allocation d'un seul tenant

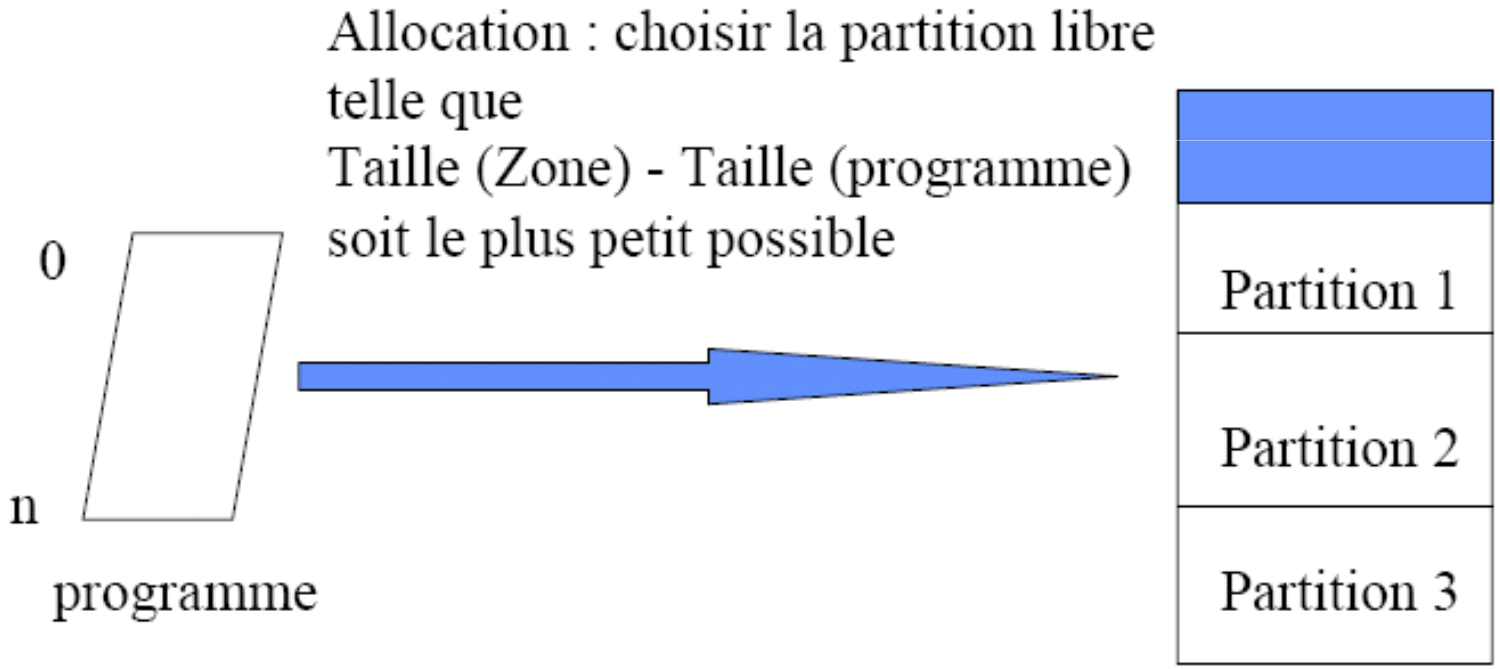
- Le programme est considéré comme un espace d'adresses insécable
- La mémoire physique est découpée en zones disjointes de taille fixe ou variable : les partitions



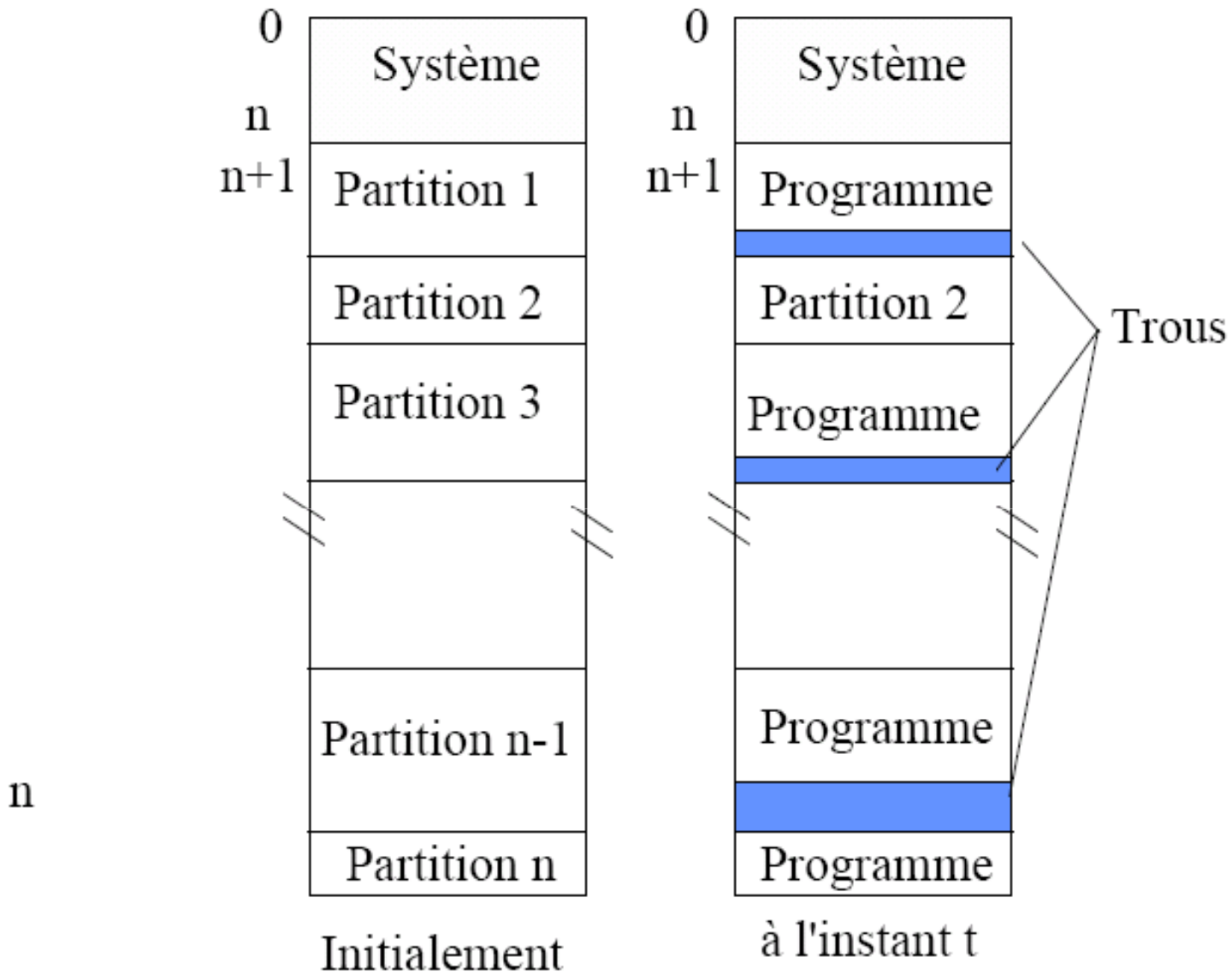


Allocation en partition fixe

- A la génération du système, la mémoire est découpée en partitions de taille fixe, non nécessairement identiques



Allocation en partition fixe



Allocation en partition variable

•La mémoire est découpée en partitions de taille variable, c'est-à-dire adaptables àla taille des programmes.

•La mémoire est formée d'un ensemble de zones libres et de zones occupées (partitions allouées). Une zone libre est caractérisée par son adresse d'implantation (adr) et sa taille (taille).

•Allouer un programme P de taille Taille(P) :

 trouver une zone libre telle que

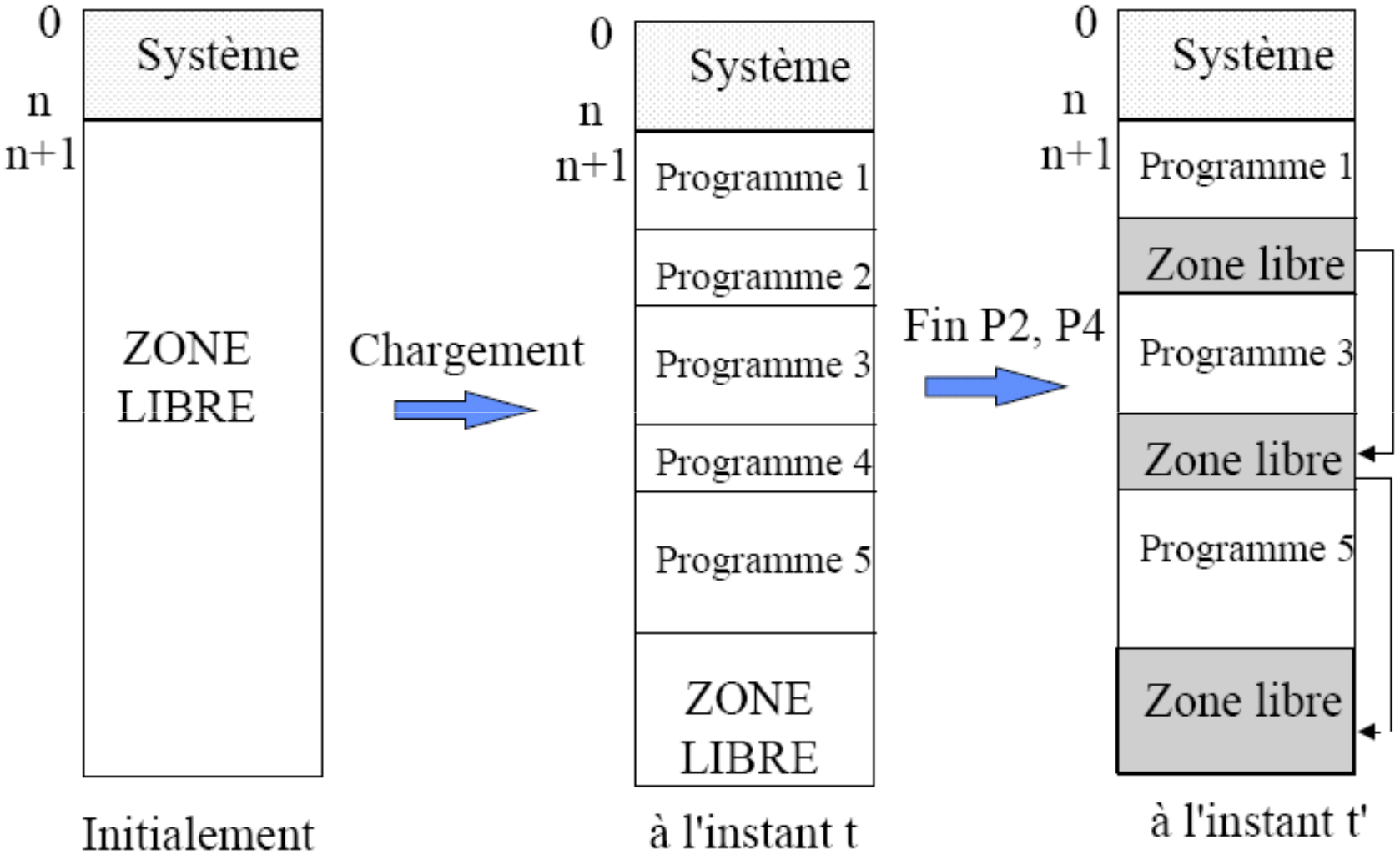
Taille (Zone Libre) >= Taille (P)

Le système maintient une liste des zones libres en MC

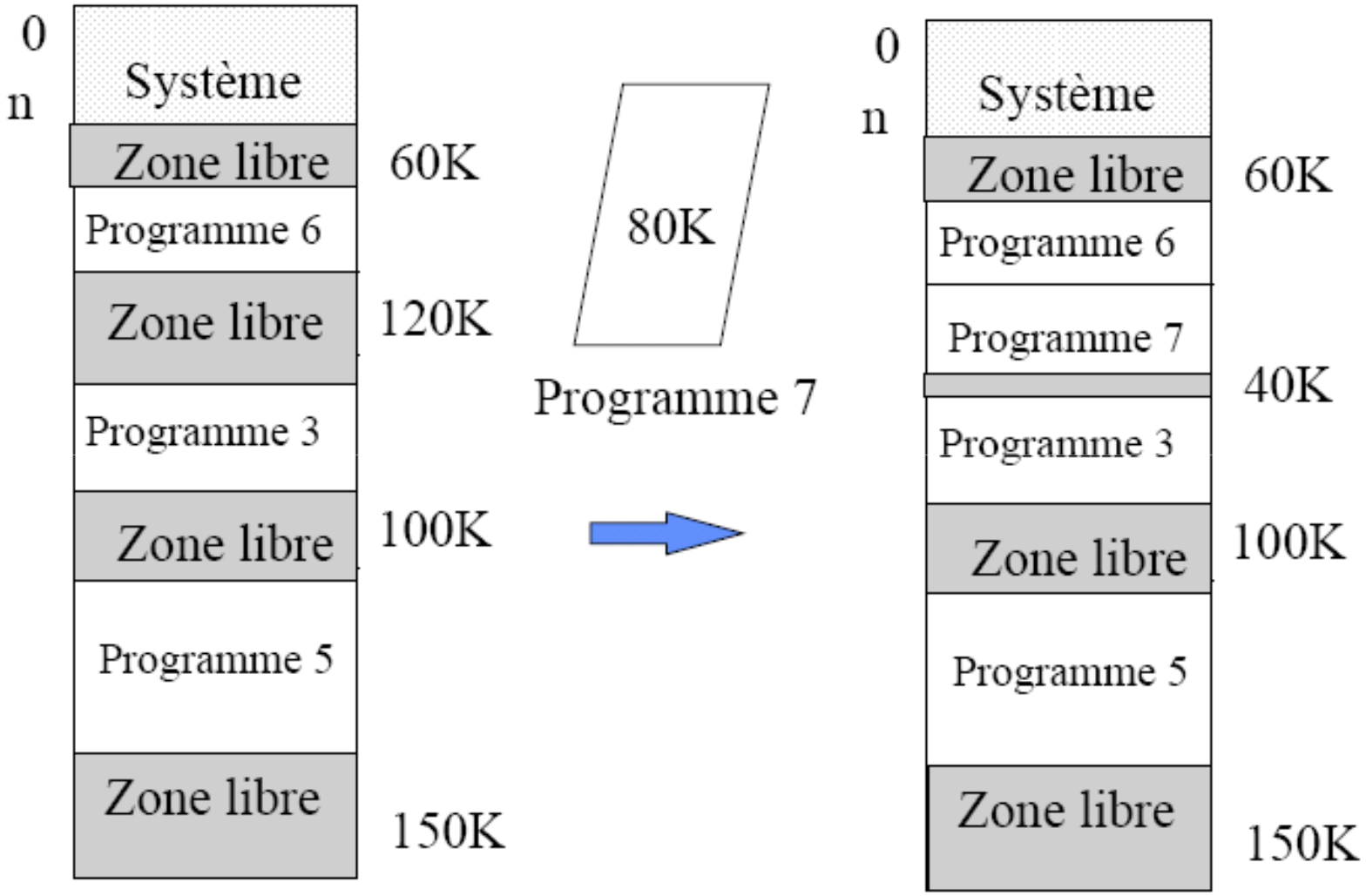
 First Fit : la première zone qui convient

 Best Fit : celle générant le moins de perte

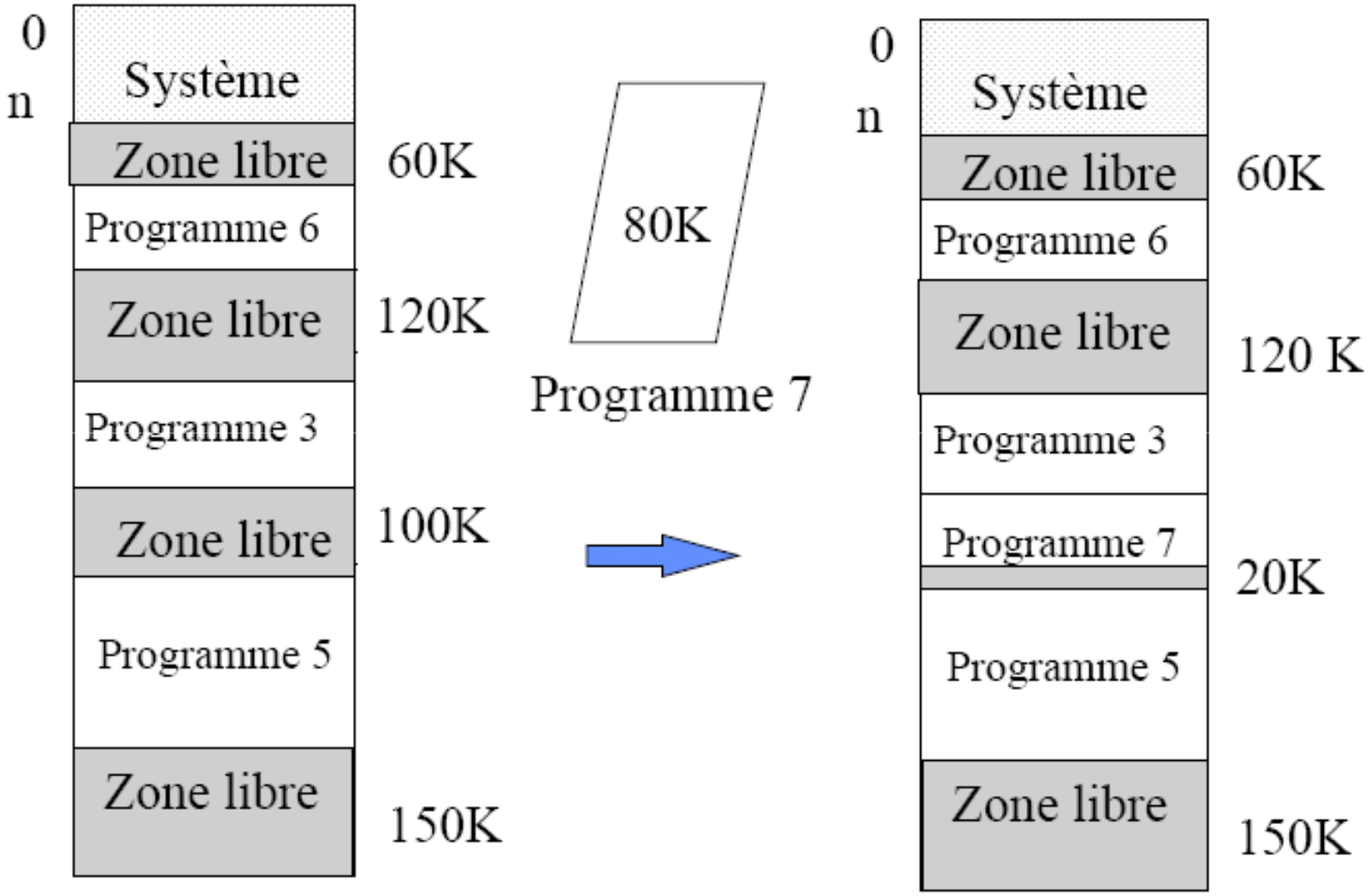
La mémoire est formée d'un ensemble de zones libres
et de zones occupées (partitions allouées)



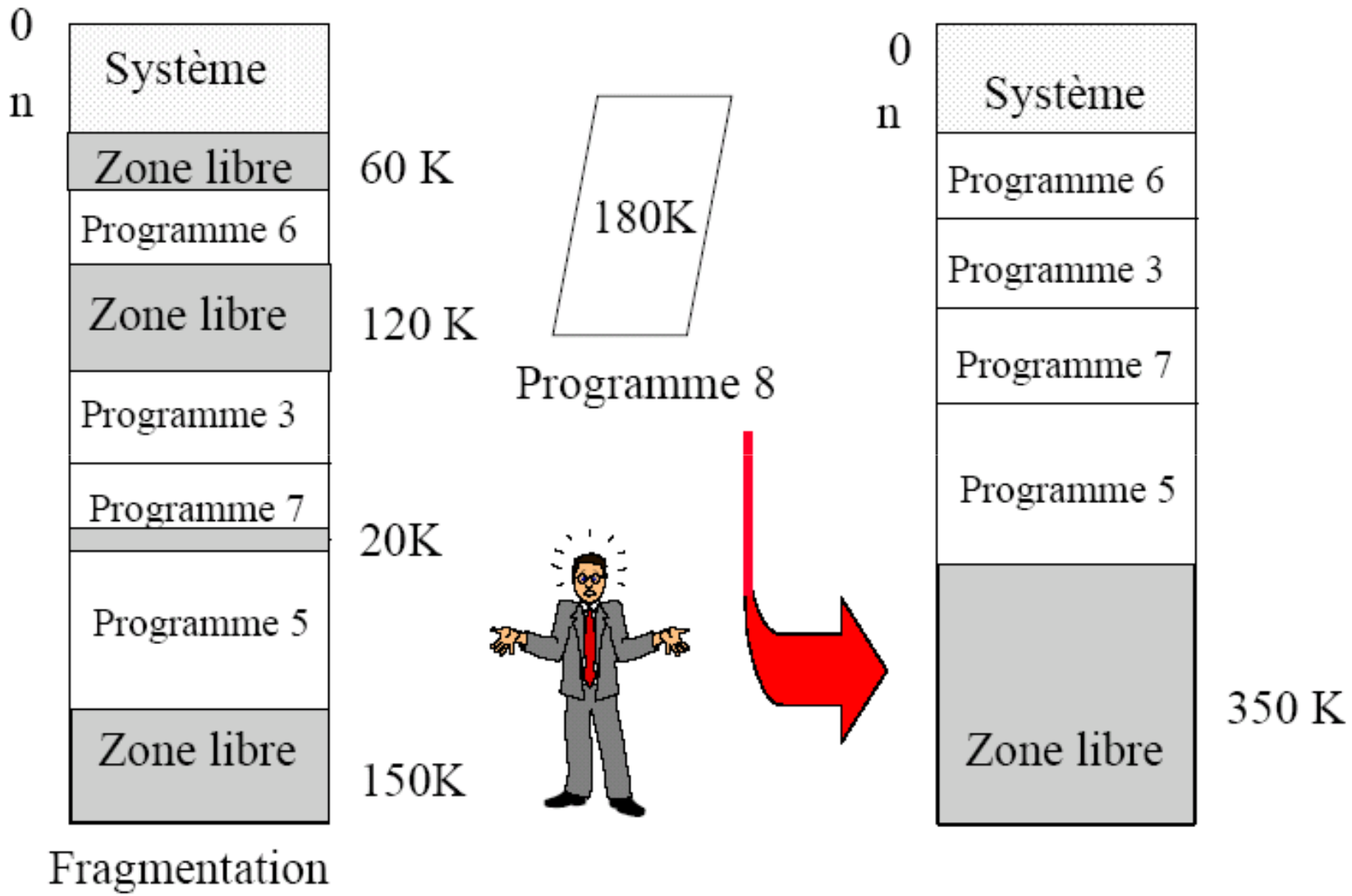
Allocation First Fit



Allocation Best Fit



Compactage de la mémoire





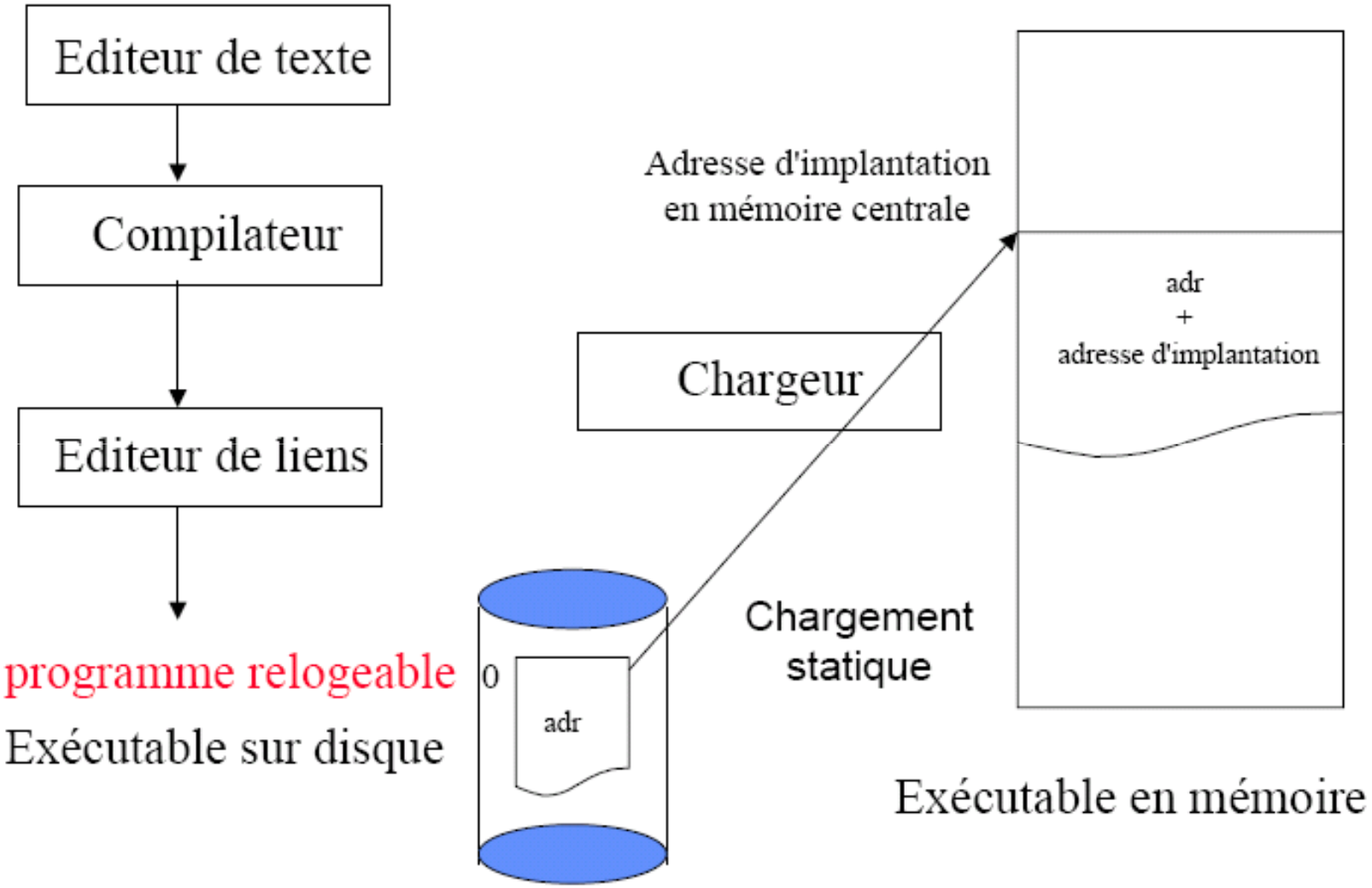
Compactage de la mémoire

- Allocations et désallocations successives des programmes créent des trous (zones libres de taille insuffisante) en mémoire centrale : fragmentation

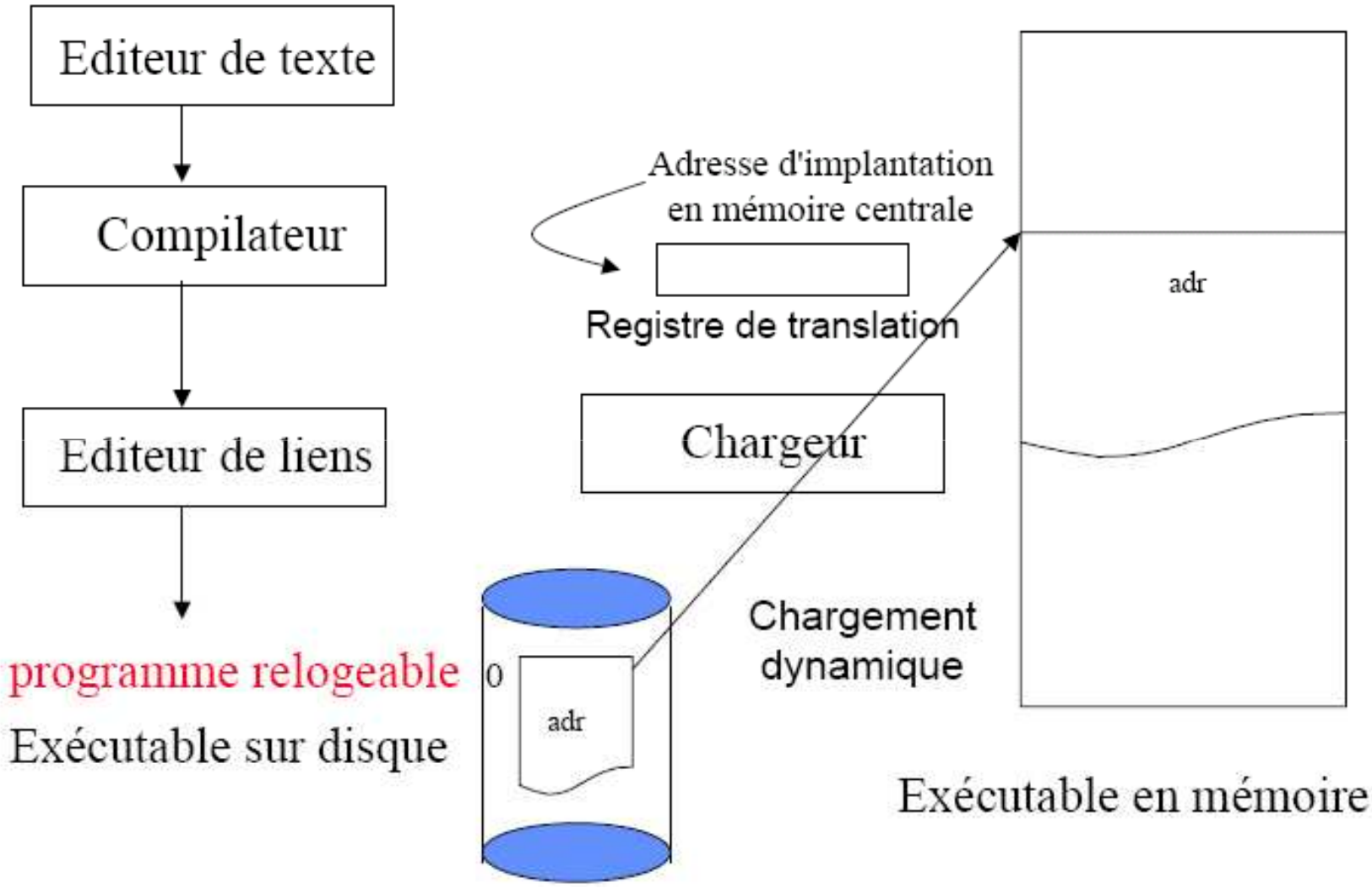
- Le compactage consiste à déplacer les programmes en mémoire centrale de manière à ne créer qu'une seule et unique zone libre.

- Le compactage est une opération très coûteuse. Elle suppose une **translation des adresses dynamique**.

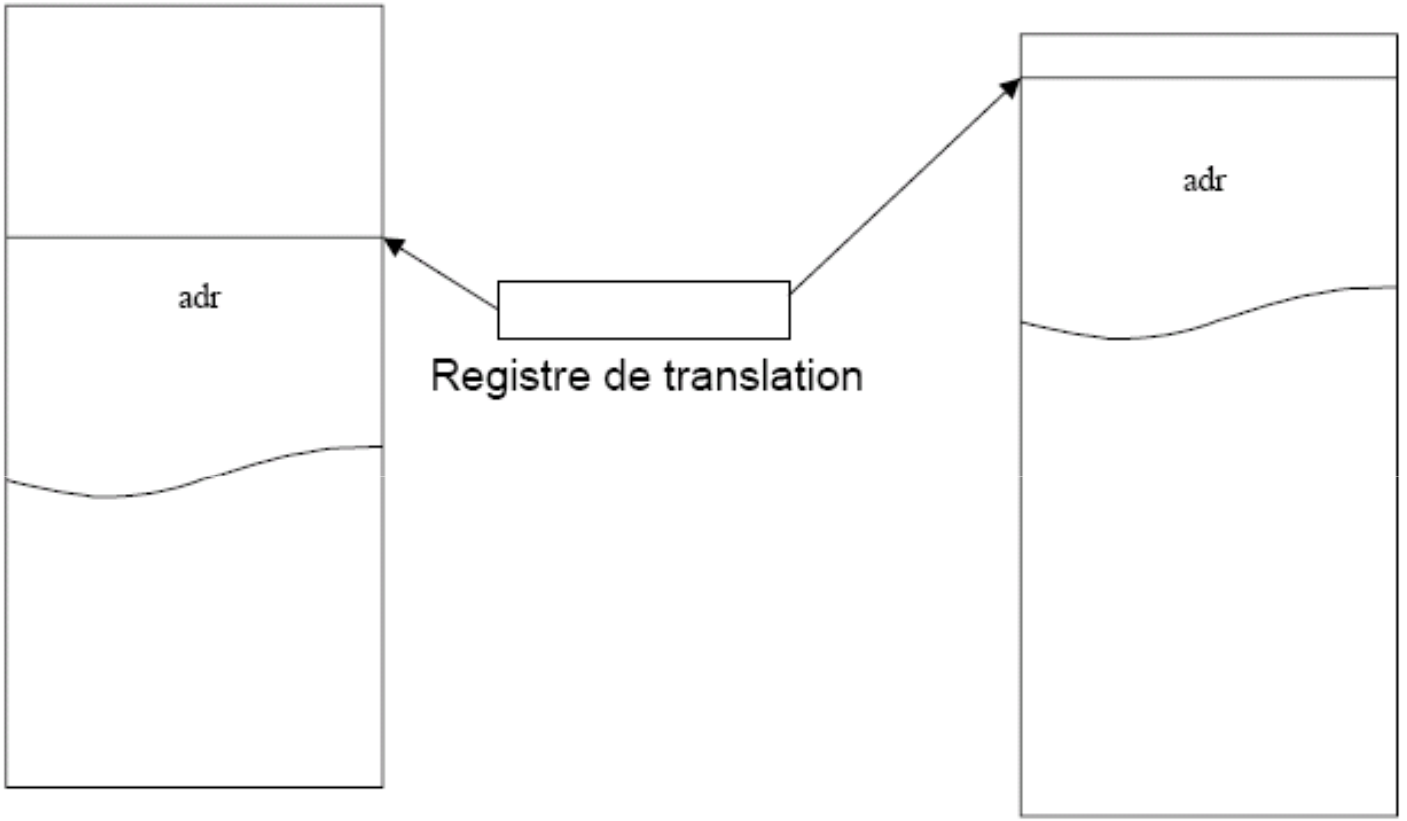
LE CHARGEMENT statique : les adresses du programme sont translatées de la valeur de l'adresse d'implantation en MC au chargement



LE CHARGEMENT dynamique: les adresses du programme sont translatées de la valeur de l'adresse d'implantation en MC à l'exécution



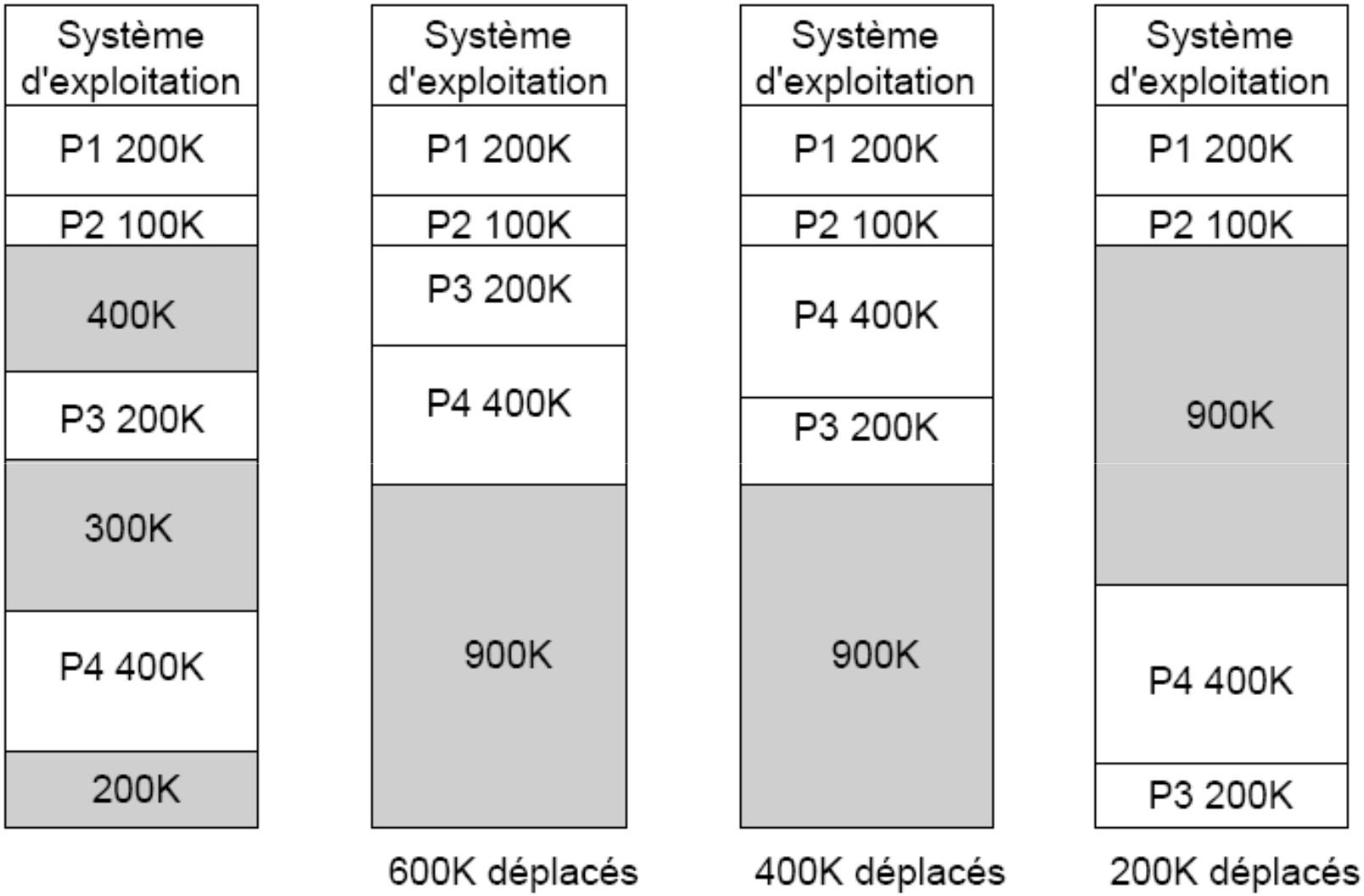
Compactage



- Déplacer un programme : modifier la valeur du registre de translation

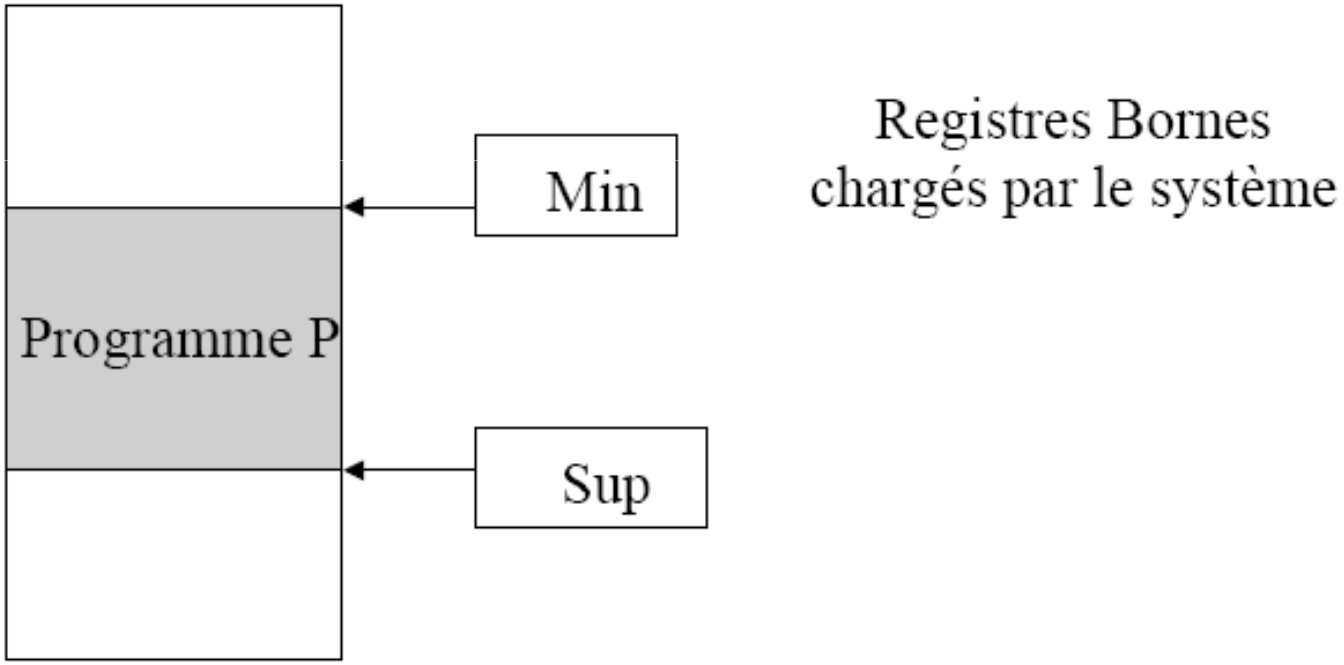


Compactage de la mémoire centrale



Protection en allocation d'un seul tenant

- Toute adresse générée par l'exécutuion du programme doit appartenir à l'intervalle des adresses qui lui sont allouées





Faiblesses de l'allocation d'un seul tenant

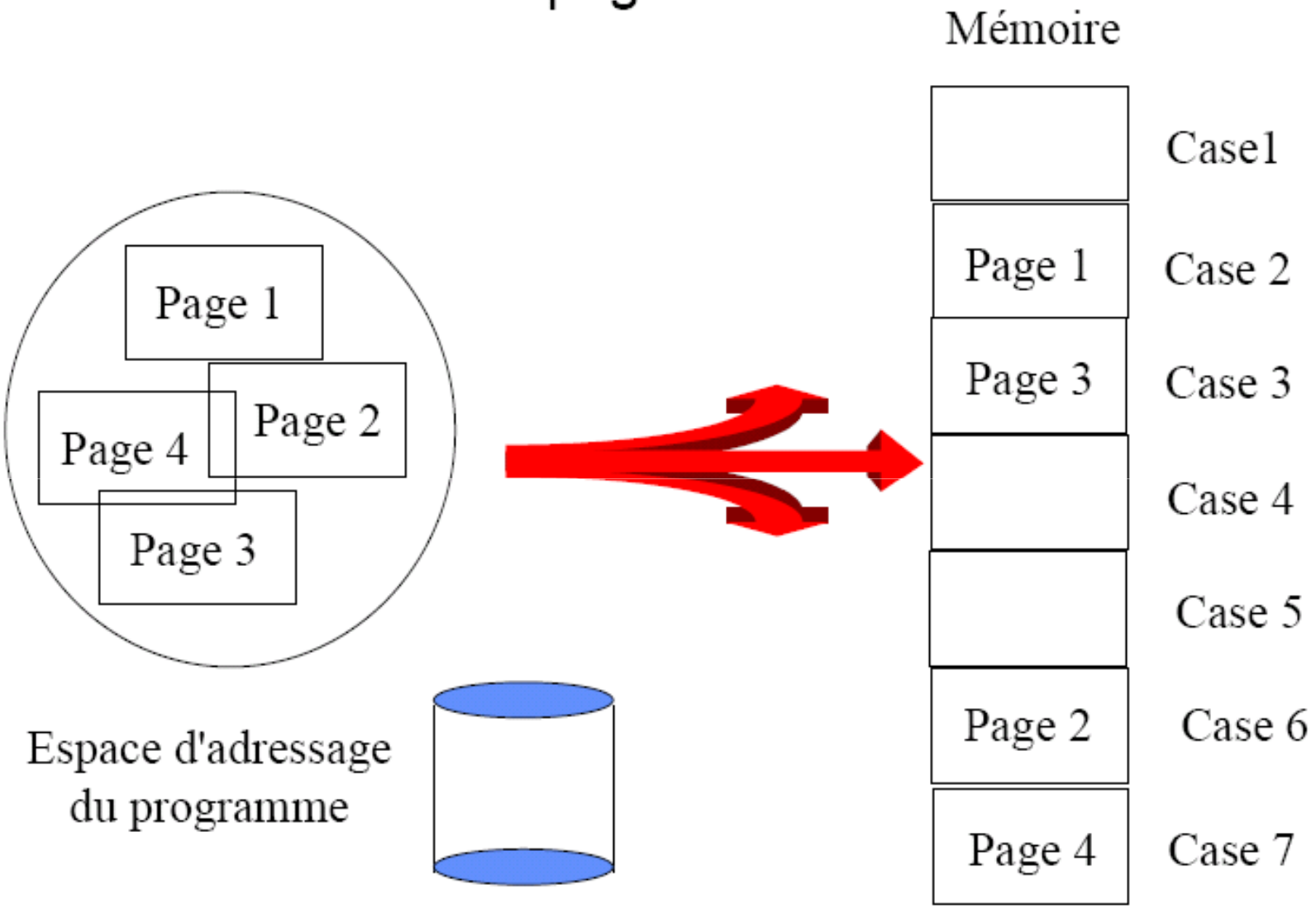
- Nécessité d'une opération de compactage de la mémoire

- Exigence d'allouer le programme en une zone d'un seul tenant



Diviser le programme en portions de taille fixe et égale à l'unité d'allocation de la mémoire centrale : les **pages**

La pagination



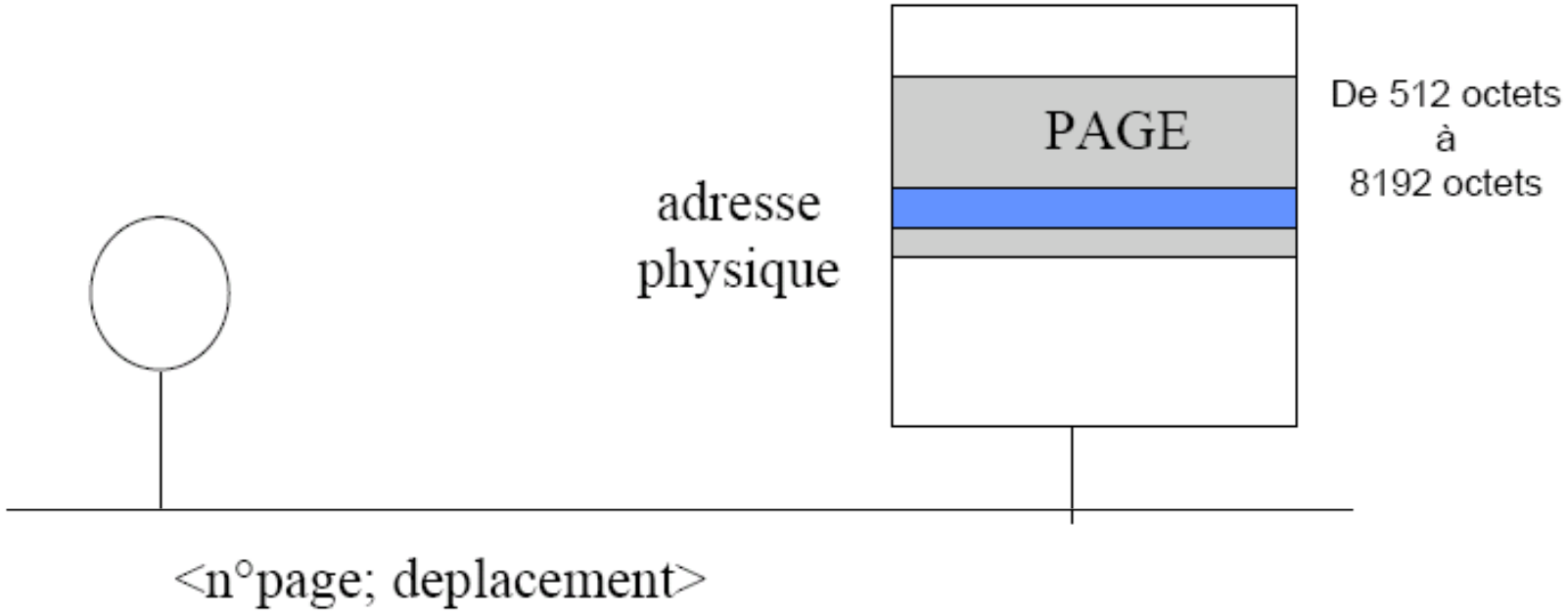


La pagination

- L'espace d'adressage du programme est découpé en morceaux linéaires de même taille : la page.
- L'espace de la mémoire physique est lui-même découpé en morceaux linéaires de même taille : la case
- La taille d'une case est égale à la taille d'une page
- Charger un programme en mémoire centrale consiste à placer les pages dans n'importe quelle case disponible.



La mémoire paginée

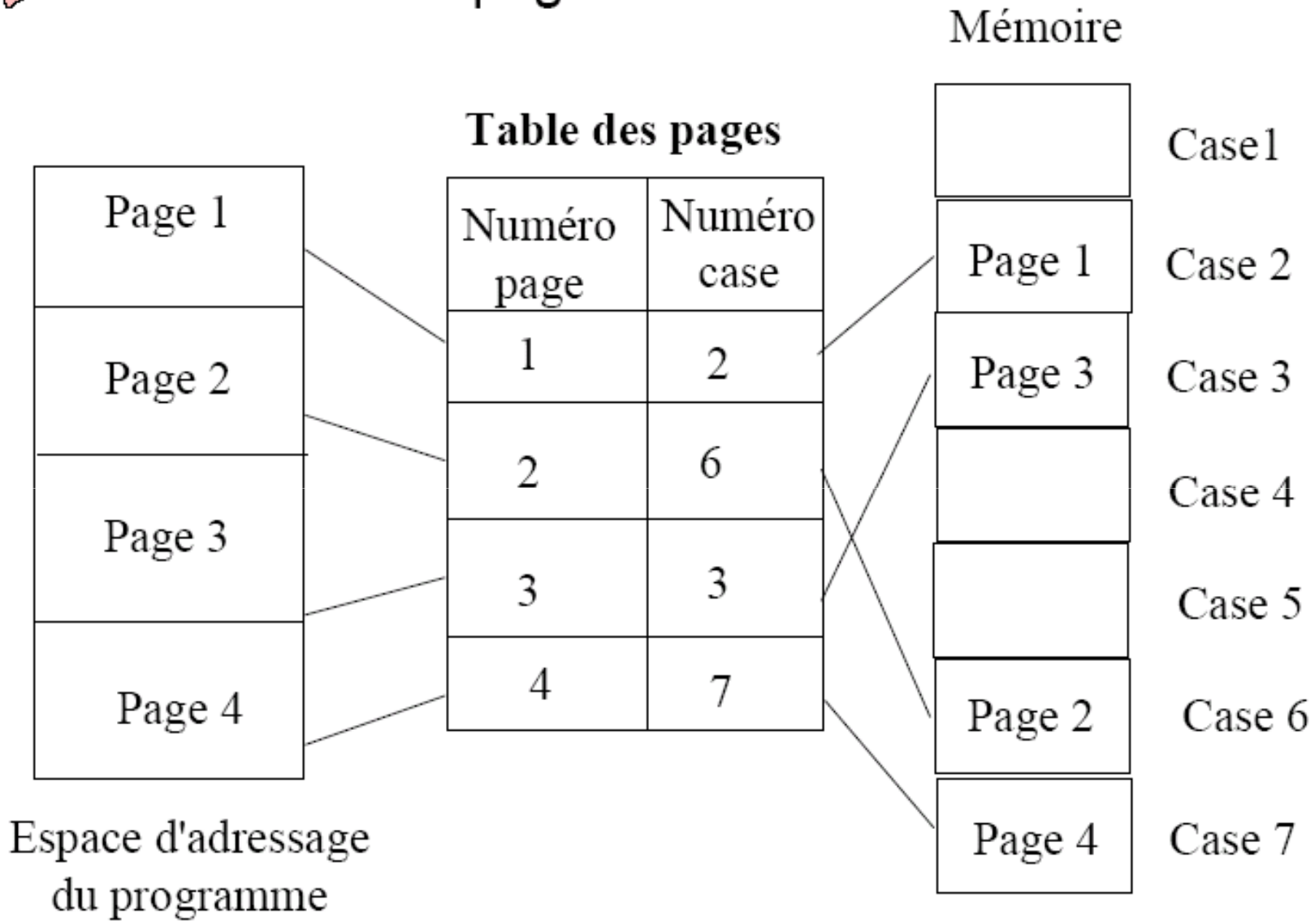


☞ Il faut convertir l'adresse paginée en son équivalent adresse physique

Adresse physique = **adresse implantation case contenant la page**
+ deplacement

↖ Table des pages

La mémoire paginée



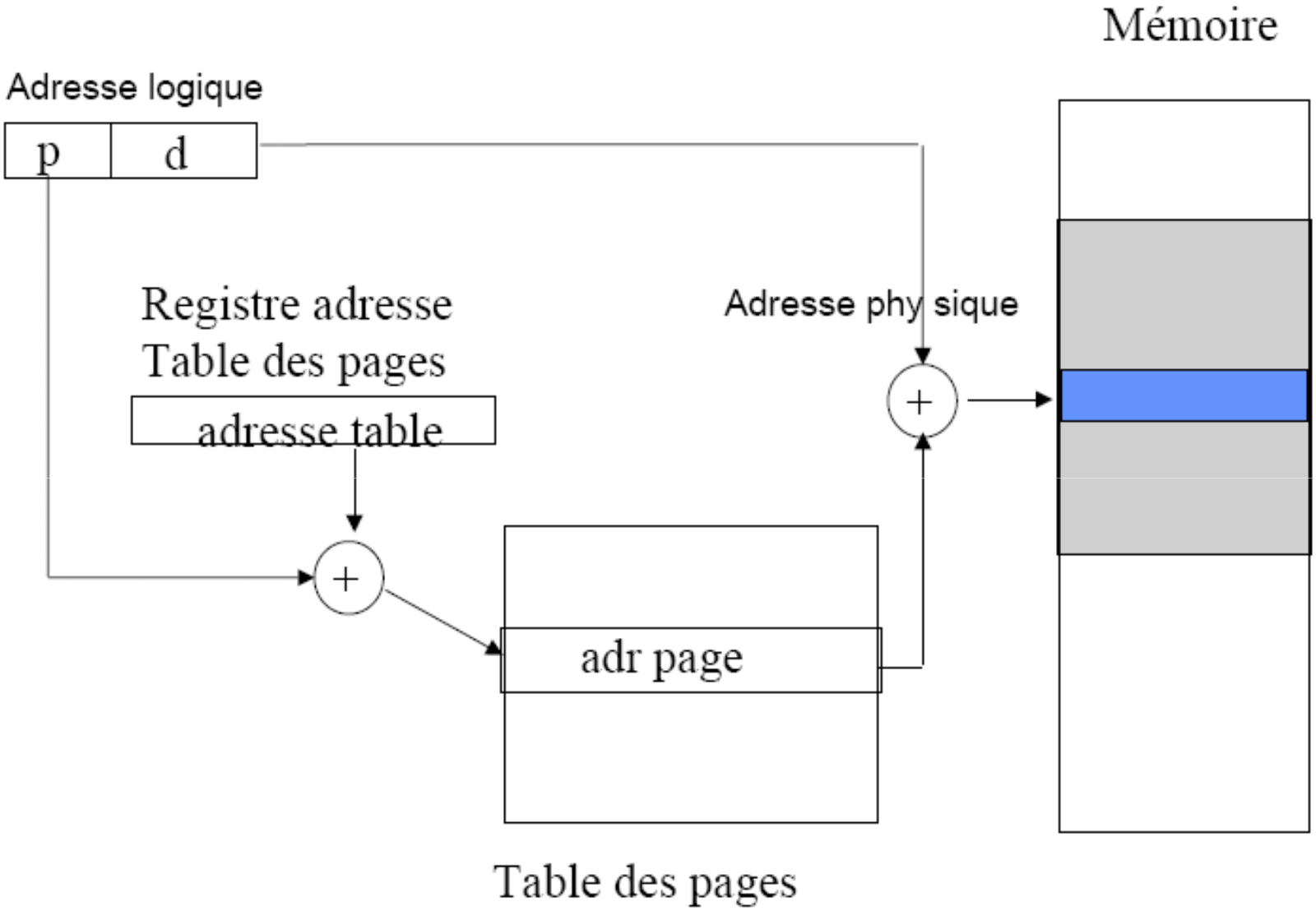
Implémentation de la table des pages

- Un ensemble de registres matériels
  - commuter de processus = charger tous les registres avec les adresses des pages du processus
  - la table des pages ne peut pas être très grande (256 entrées)

- En mémoire centrale, repérée par un registre PTBR (*page table base register*)
  - commuter de processus = charger le registre PTBR avec l'adresse de la table des pages du processus
  - pas de limite de taille à la table des pages

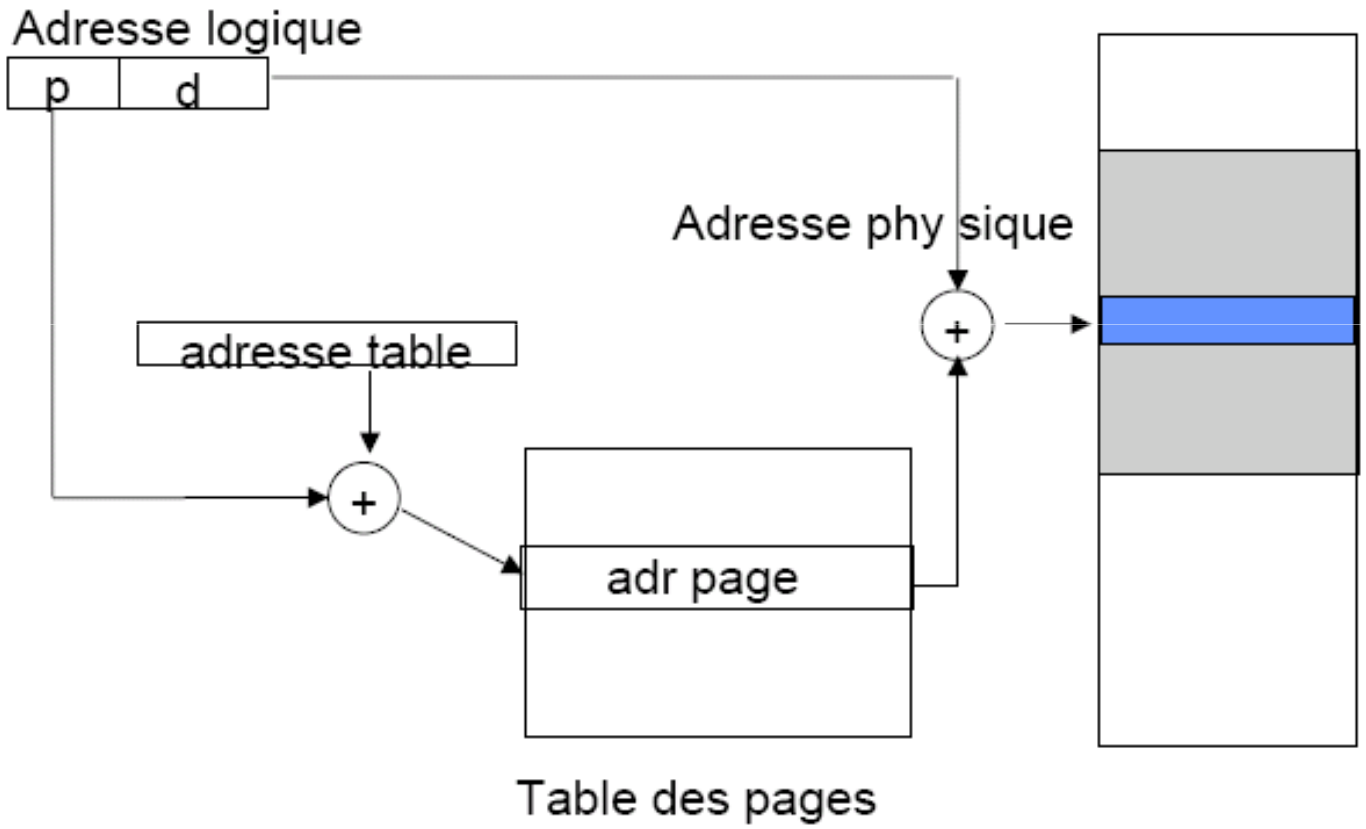
☞ problème : le temps d'accès à un emplacement mémoire d'un programme

La mémoire paginée



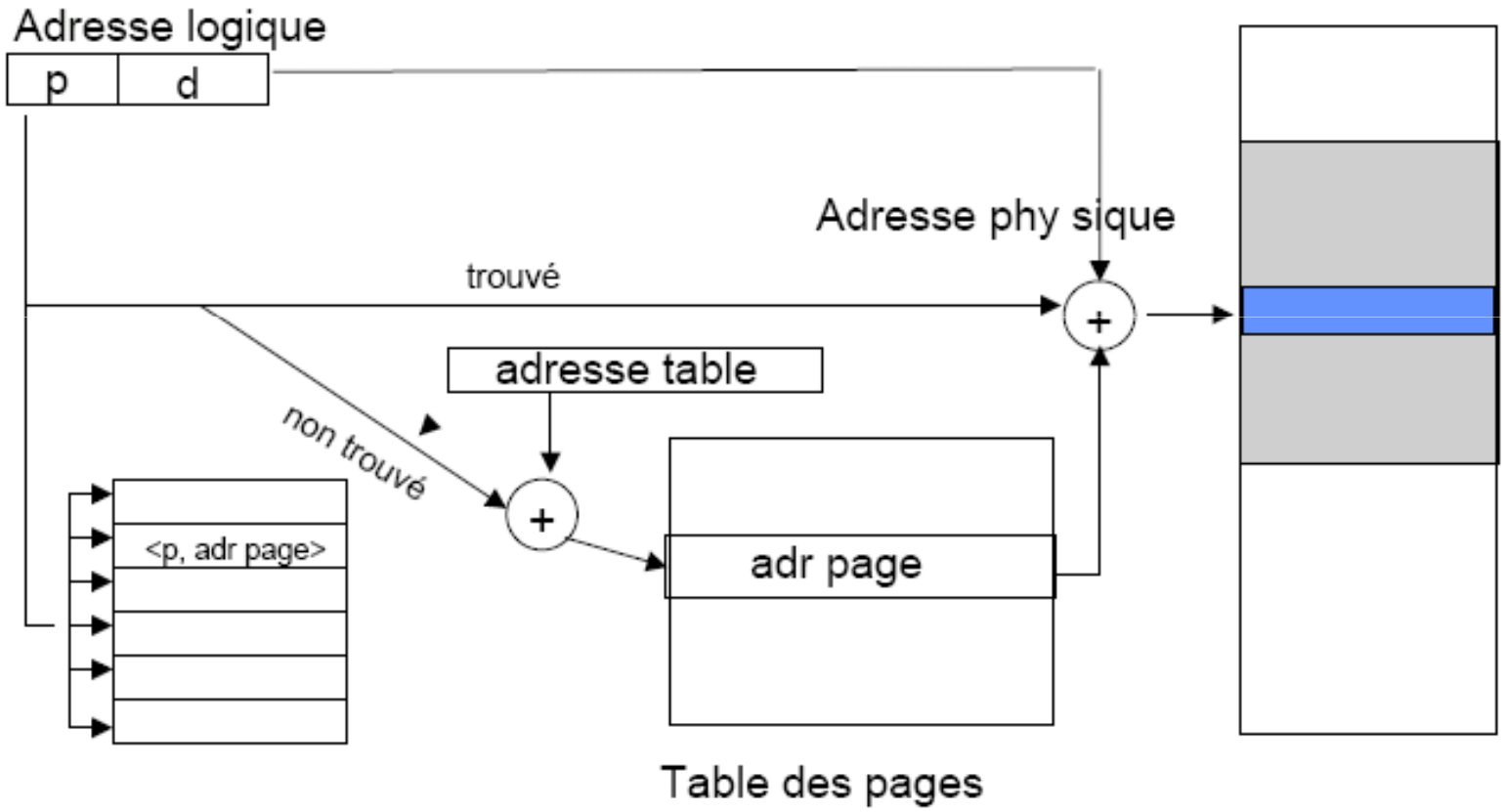
Implémentation de la table des pages

- temps d'accès à un emplacement mémoire d'un programme : deux accès mémoire



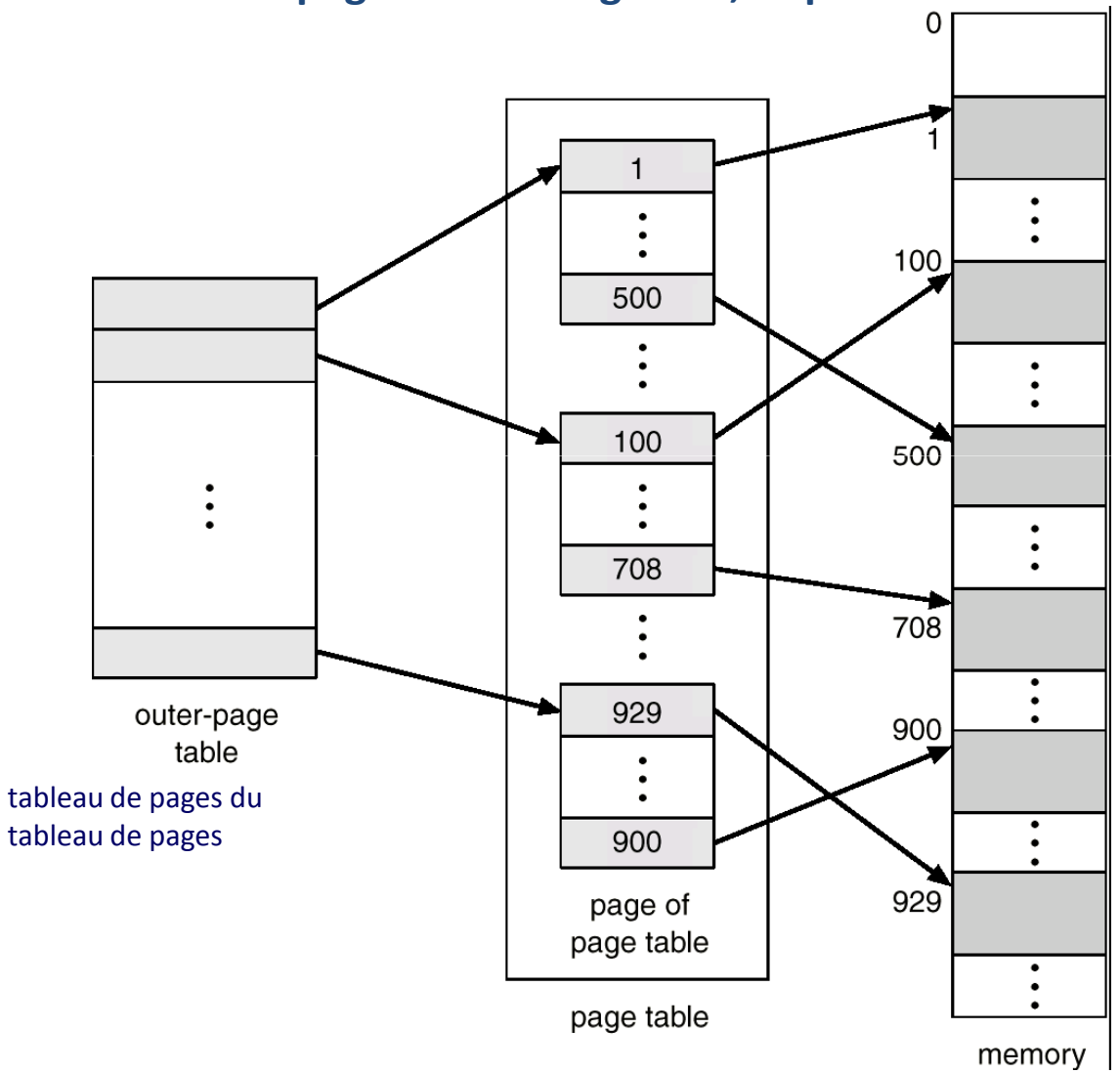
Implémentation de la table des pages

- Un cache associatif contenant les derniers couples <page, case> accédés est placé en amont de la table des pages



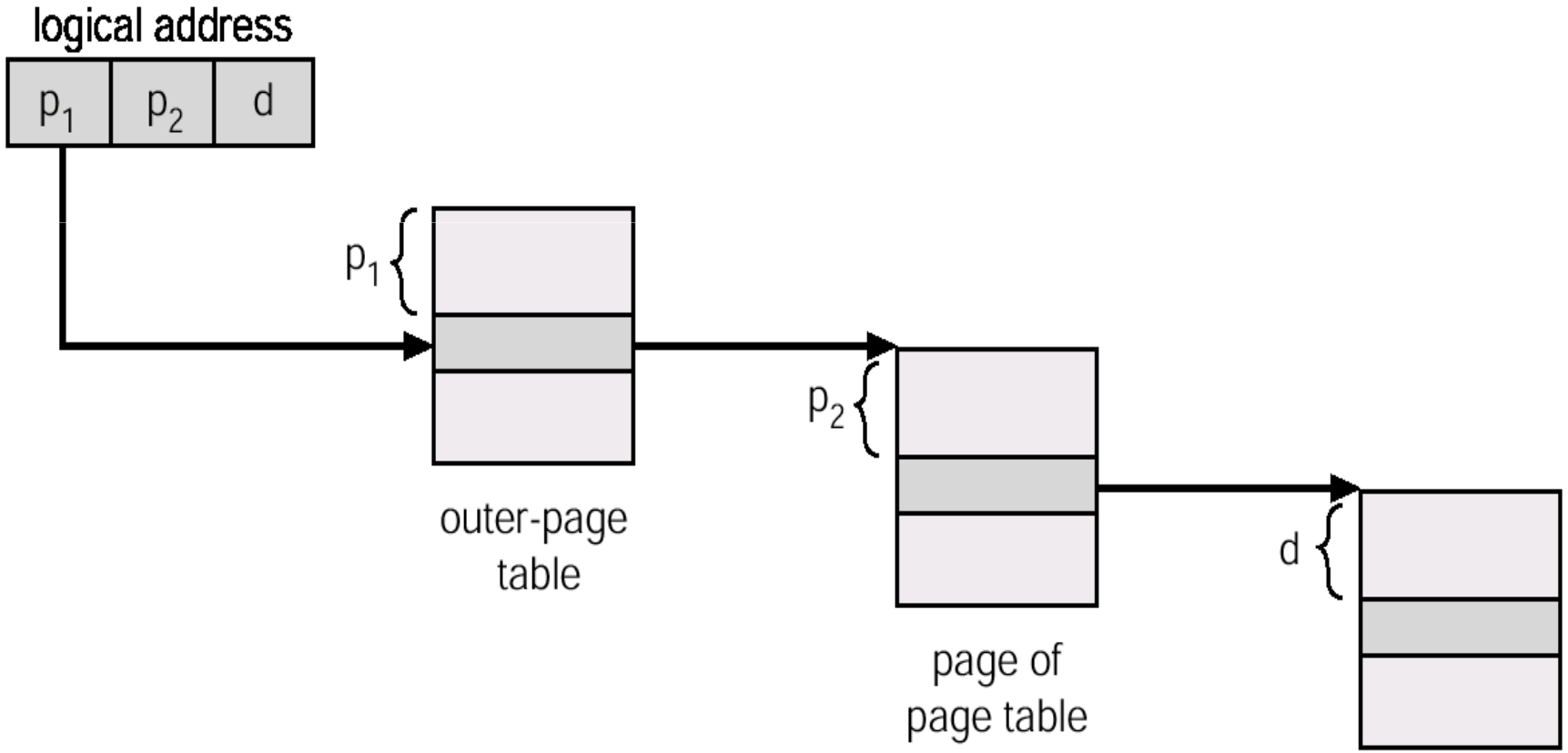
Tableaux de pages à deux niveaux

(quand les tableaux de pages sont très grands, ils peuvent être eux mêmes paginés)

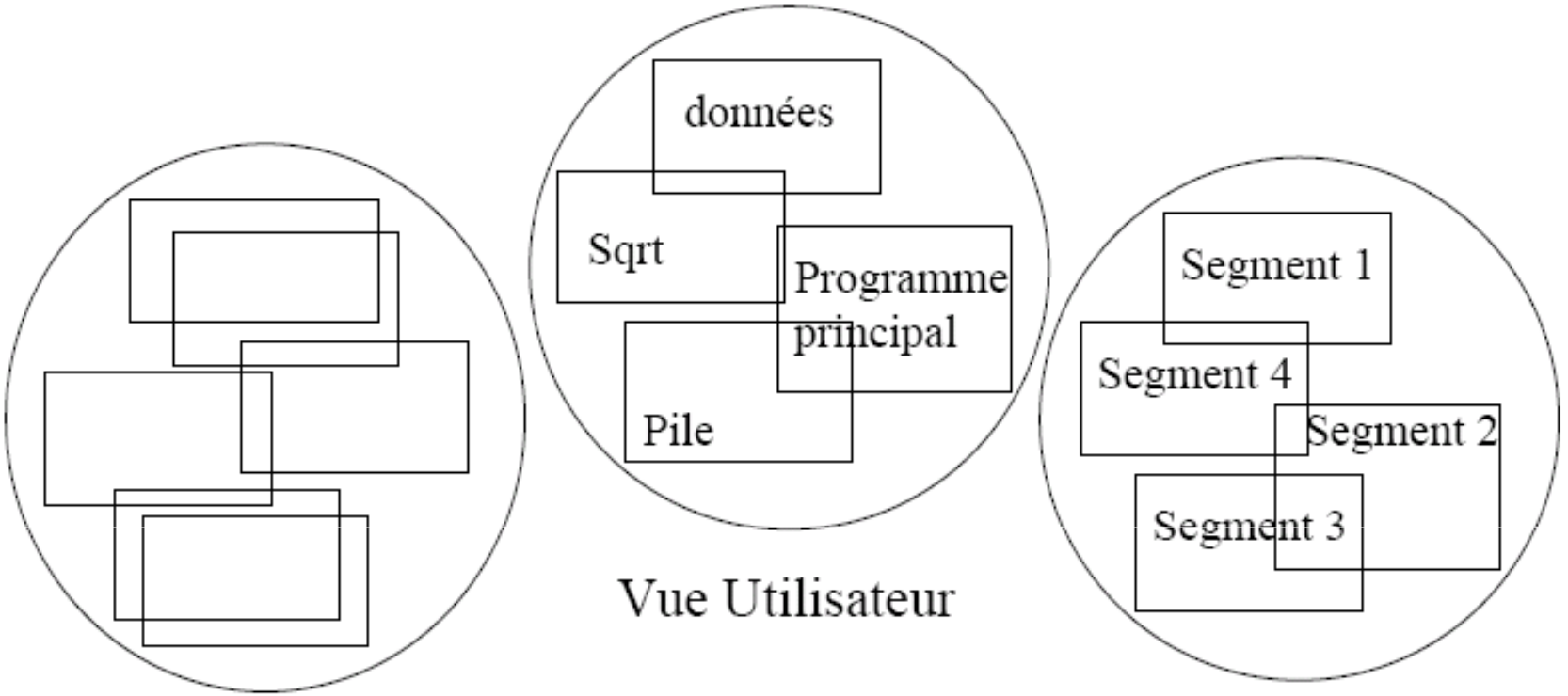


Tableaux de pages à deux niveaux

- La partie de l ’adresse qui appartient au numéro de page est elle-même divisée en 2



Pagination - Segmentation



Vue Utilisateur

Espace d'adressage
du programme
Ensemble de pages

Espace d'adressage
du programme
Ensemble de segments

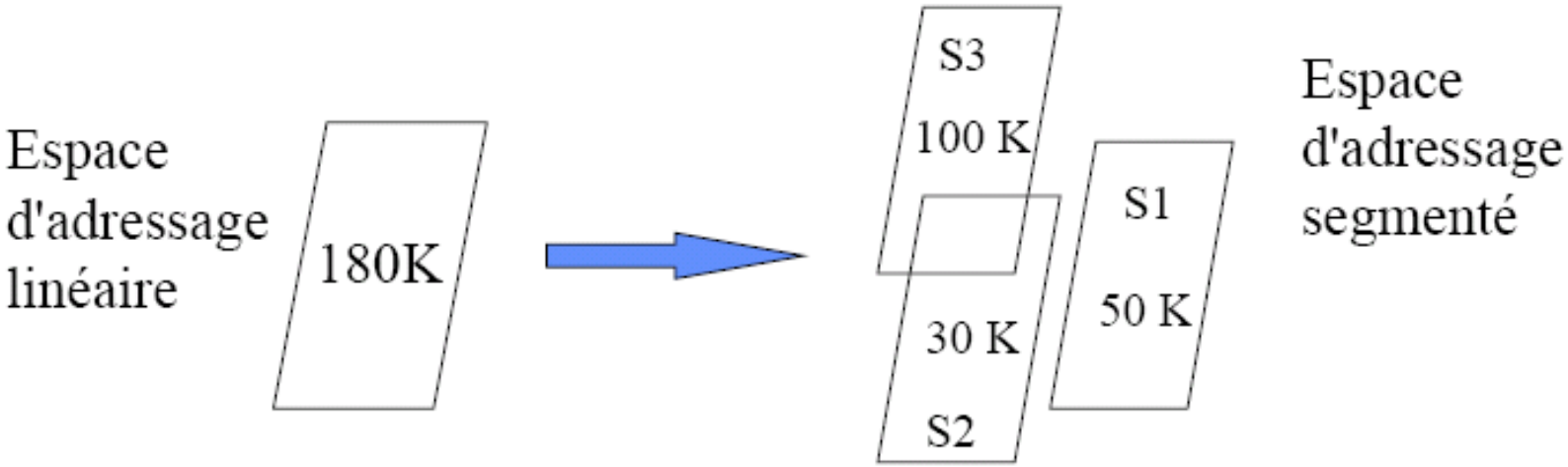
- A fin d ’éliminer le besoin de compactage, le prochain pas est d`utiliser l’allocation non contiguë
  - diviser un programme en morceaux et permettre l`allocation séparée de chaque morceau
- Il y a deux techniques de base pour faire ceci: la pagination et la segmentation
  - la segmentation utilise des parties de programme qui ont une valeur logique (des modules)
  - la pagination utilise des parties de programme arbitraires (morcellement du programmes en pages de longueur fixe).
  - elles peuvent être combinées
- Je trouve que la segmentation est plus naturelle !!!



La mémoire segmentée

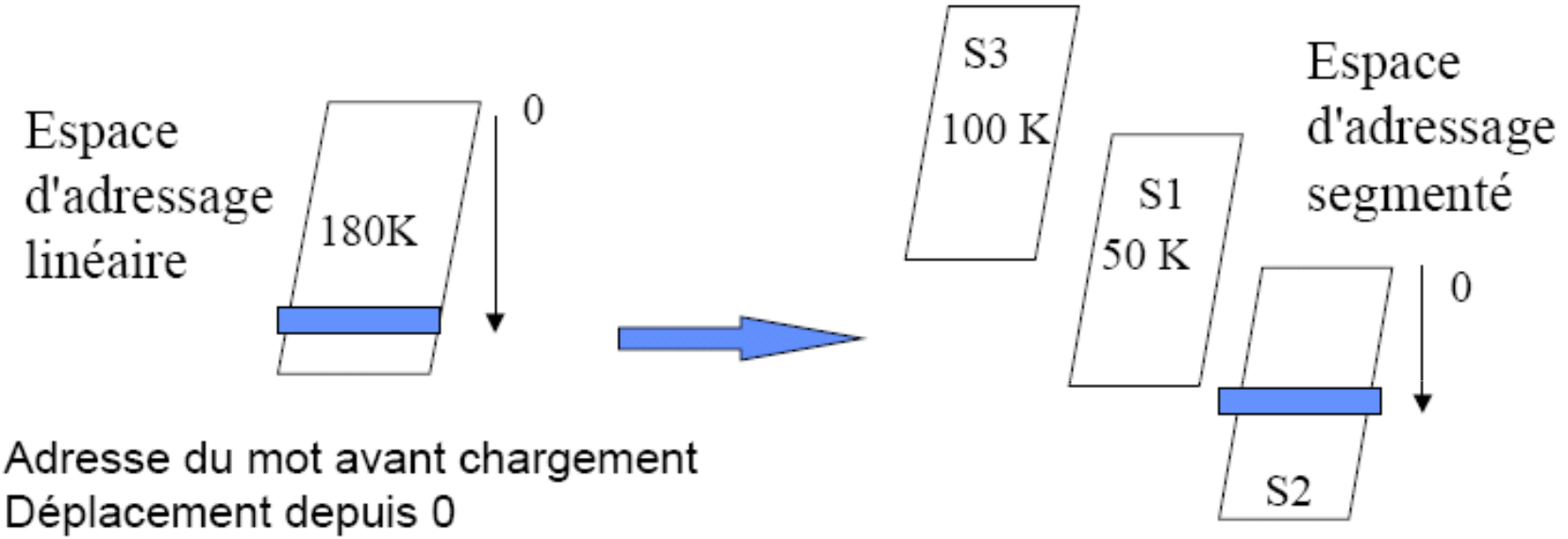
- Le programme est divisé en segments par le compilateur, un segment étant un espace d'adressage linéaire formé d'adresses contigues.

- Un segment correspond à une partie logique du programme : segment de code, segment de données, segment de pile





La mémoire segmentée



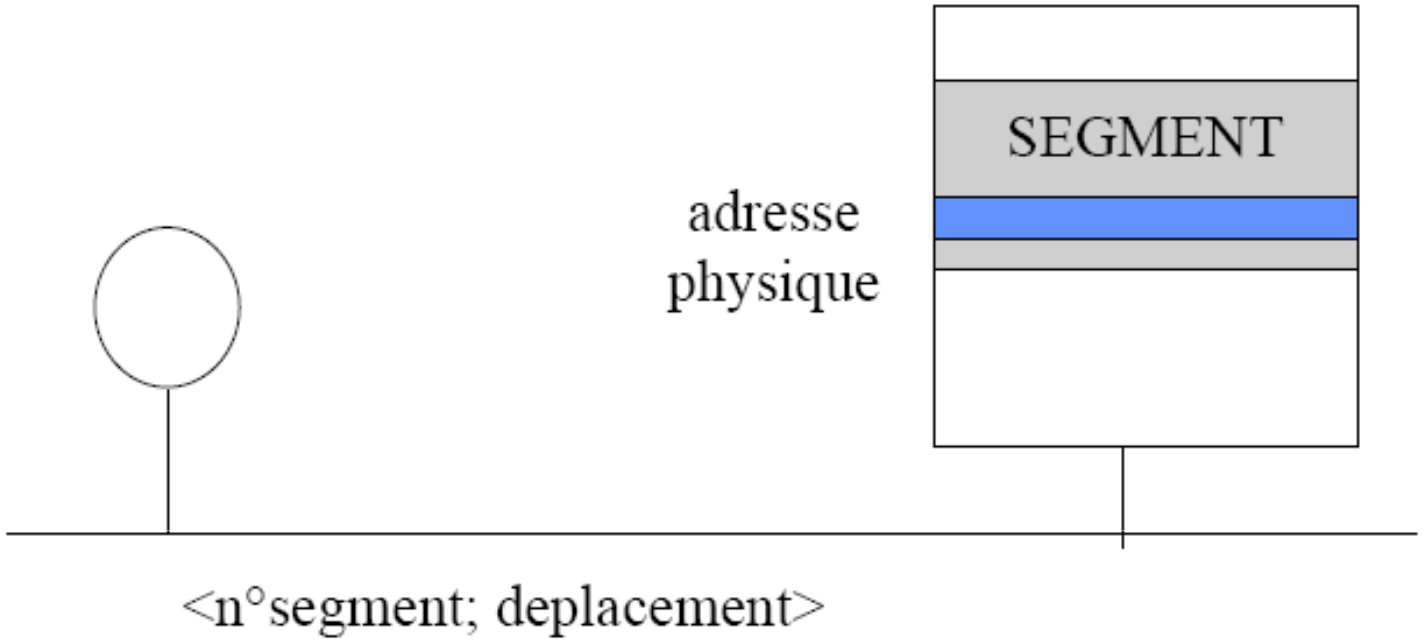
Adresse du mot avant chargement
Déplacement depuis 0

Adresse du mot après chargement
Adresse d'implantation + Déplacement

Adresse du mot avant chargement
N° segment, Déplacement depuis 0

Adresse du mot après chargement
Adresse d'implantation du segment
+ Déplacement

La mémoire segmentée



☞ Il faut convertir l'adresse segmentée en son équivalent adresse physique

Adresse physique = adresse implantation segment + deplacement

⇖ Table des segments

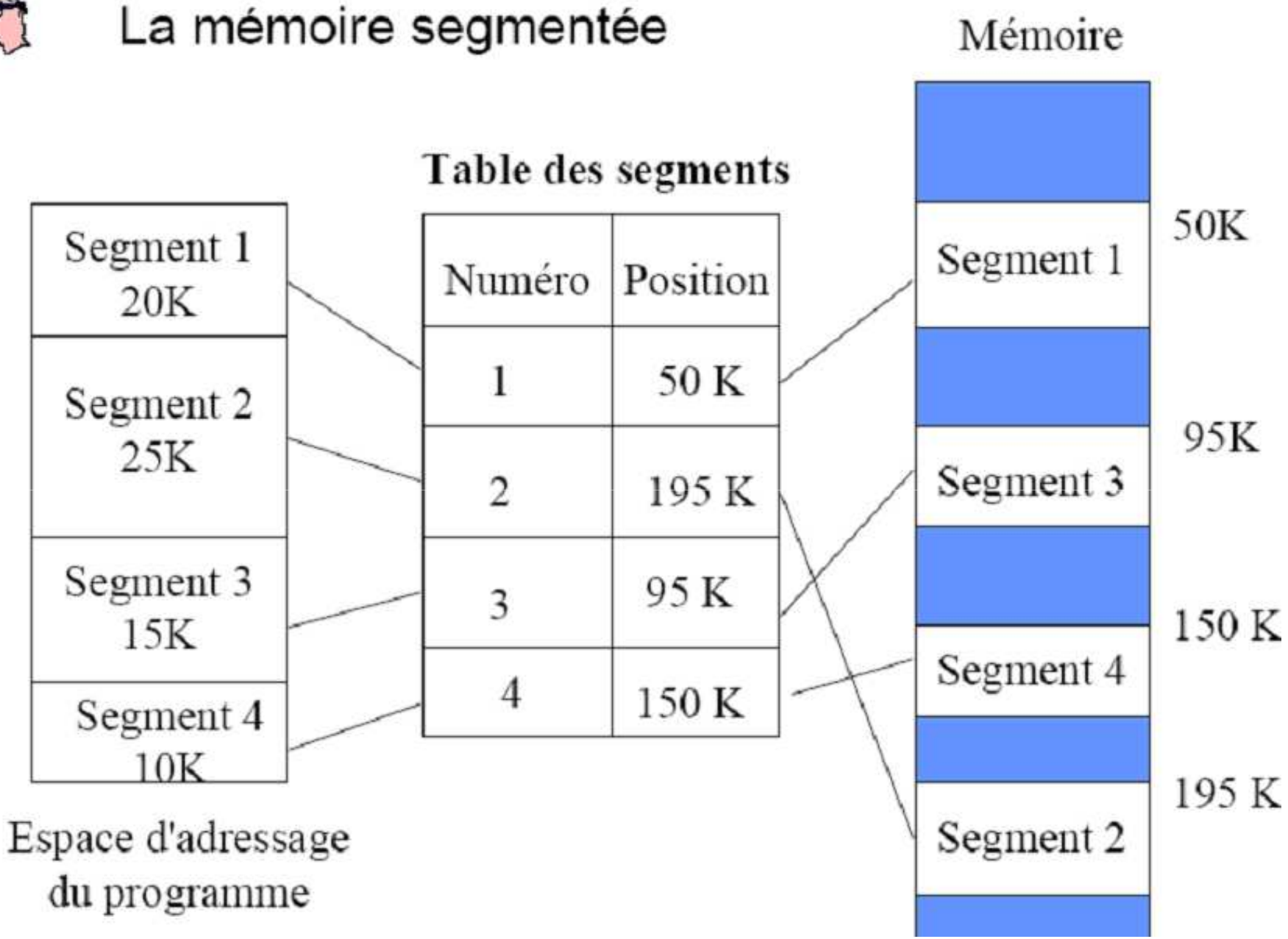
Détails

- **L'adresse logique consiste d'une paire:**

  <No de segm, déplacement>

- **le tableau des segments contient: descripteurs de segments**
  - **adresse de base**
  - **longueur du segment**
  - **Infos de protection, on verra…**
- **Dans le PBC du processus il y aura un pointeur à l 'adresse en mémoire du tableau des segments**
- **Il y aura aussi là dedans le nombre de segments dans le processus**
- **Au moment de la commutation de contexte, ces infos seront chargées dans les registres appropriés d'UCT**

La mémoire segmentée



La mémoire segmentée

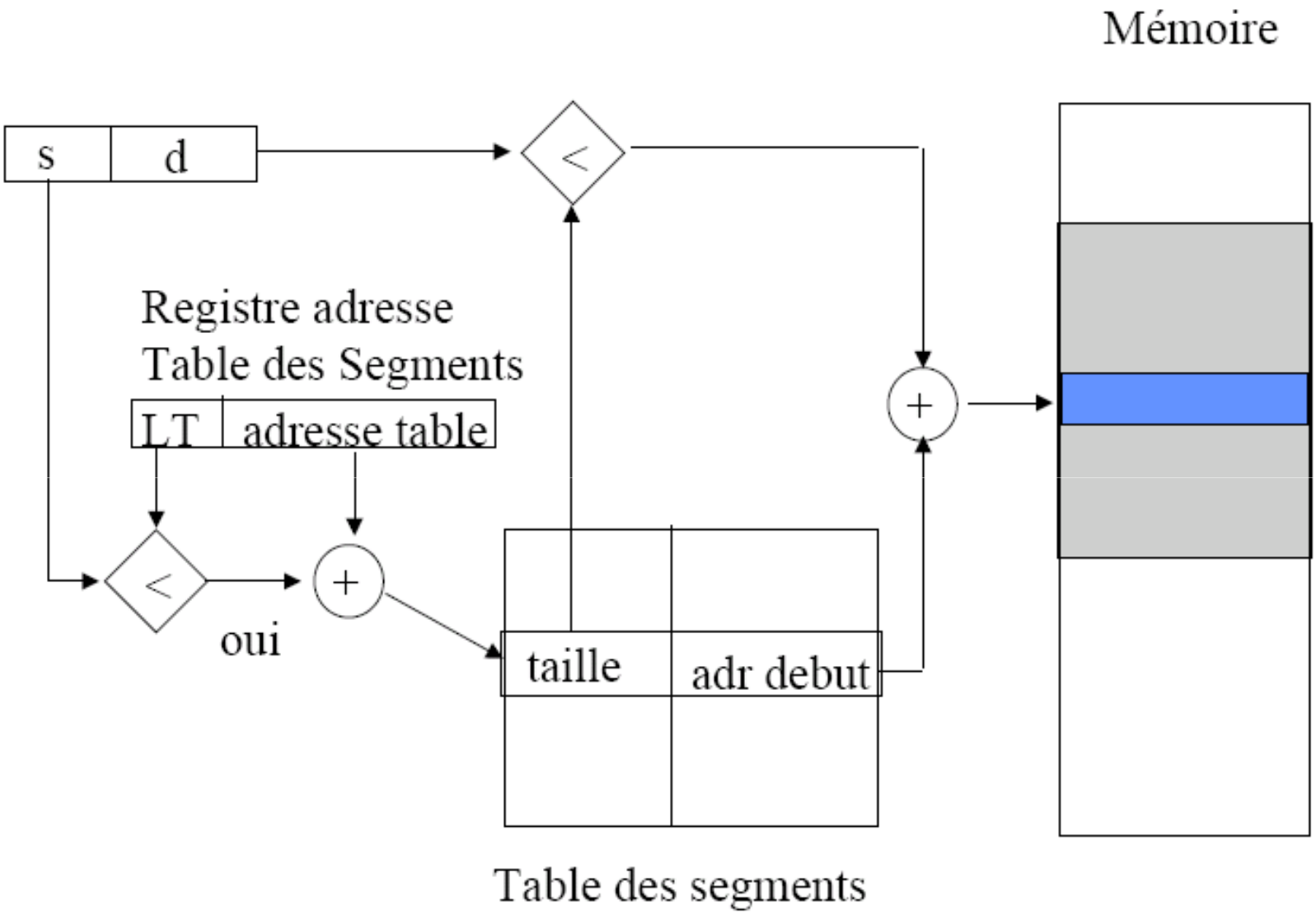


Table des segments



La segmentation

- Allouer un segment S de taille Taille(S) :
  - ⇖ trouver une zone libre telle que
    Taille (Zone Libre) > Taille (S)

- Allocations et désallocations successives des segments créent également un problème de fragmentation

☞ combiner segmentation et pagination : paginer les segments

La mémoire segmentée paginée

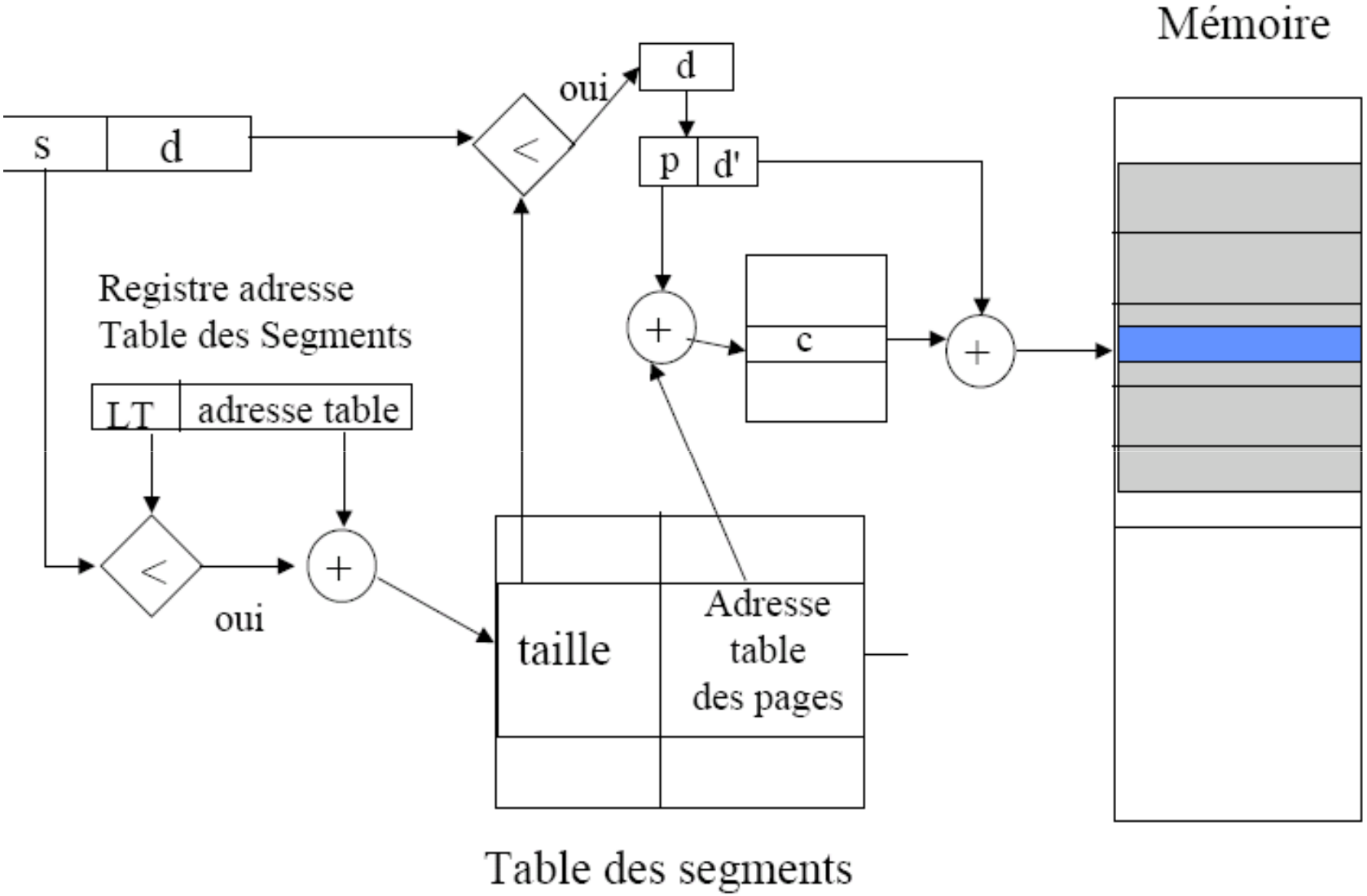


Table des segments



Gestion mémoire

- L'allocation en partitions fixes ou variables considère le programme comme un ensemble d'adresses insécables. Ce type d'allocation pose un problème de fragmentation et nécessite des opérations de compactage de la mémoire centrale

- La pagination découpe l'espace d'adressage du programme en pages et la mémoire physique en cases de même taille. Une adresse générée par le processeur est de la forme <n°page, déplacement dans la page>. La table des pages du processus permet de traduire l'adresse paginée en adresse physique

- La segmentation découpe l'espace d'adressage du programme en segments correspondant à des morceaux logiques du programme. Une adresse générée par le processeur est de la forme <n°segment, déplacement dans le segment>. La table des segments du processus permet de traduire l'adresse segmentée en adresse physique

- Segmentation et pagination sont très souvent associées.

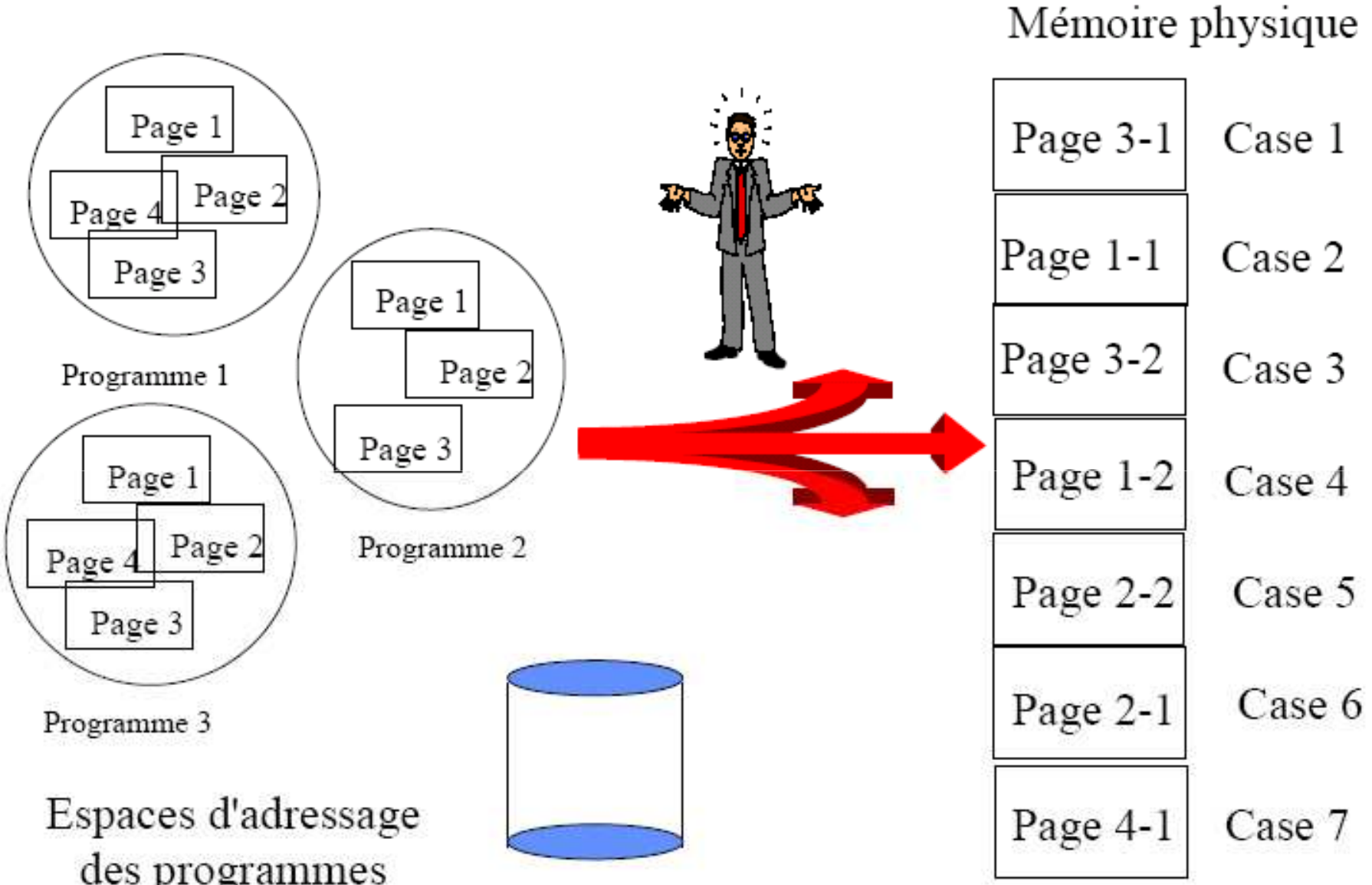
Gestion de la mémoire centrale

Mémoire virtuelle

De la pagination et segmentation à la mémoire virtuelle

- **Un processus est constitué de pièces (pages ou segments) ne nécessitant pas d'occuper une région contiguë de la mémoire principale**
- **Références à la mémoire sont traduites en adresses physiques au moment d'exécution**
  - **Un processus peut être déplacé à différentes régions de la mémoire, aussi mémoire disque!**
- **Donc: toutes les pièces d'un processus ne nécessitent pas d'être en mémoire principale durant l'exécution**
  - **L'exécution peut continuer à condition que la prochaine instruction (ou donnée) est dans une pièce se trouvant en mémoire principale**
- **La somme des mémoires logiques des procs en exécution peut donc excéder la mémoire physique disponible**
- **Une image de tout l'espace d'adressage du processus est gardée en mémoire secondaire (normal. disque) d'où les pages manquantes pourront être prises au besoin**
  - **Mécanisme de va-et-vient ou swapping**

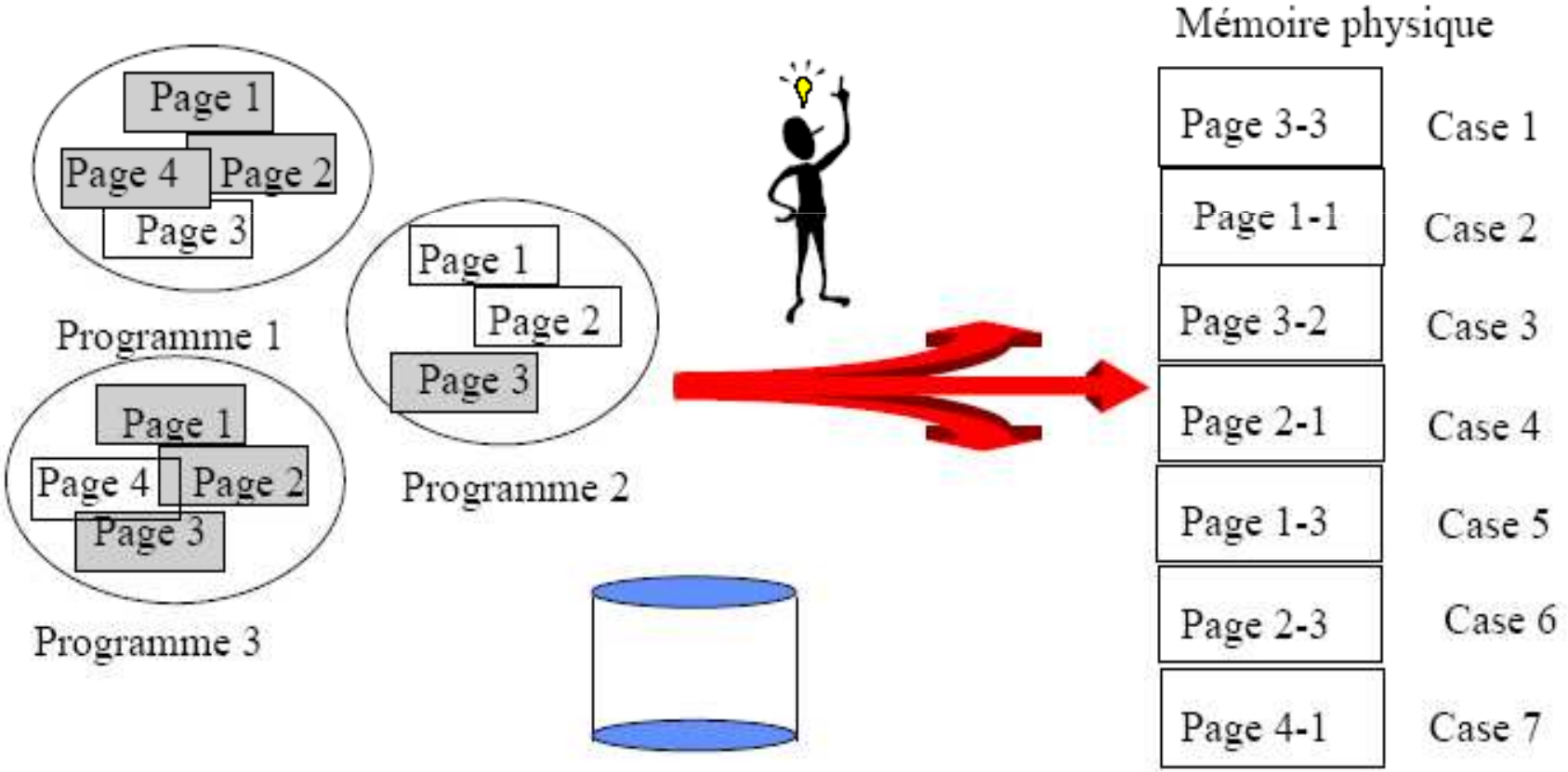
Mémoire virtuelle



Mémoire virtuelle

- La capacité de la mémoire centrale est trop petite pour charger l'ensemble des pages des programmes utilisateurs.

  ☞ Ne charger que les pages utiles à un instant.



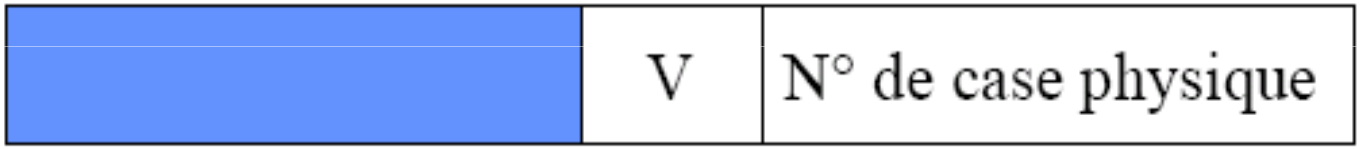
Avantages du chargement partiel

- Plus de processus peuvent être maintenus en exécution en mémoire
  - Car seules quelques pièces sont chargées par chaque processus
  - L'usager est content, car il peut exécuter plusieurs processus et faire référence à des gros données sans avoir peur de remplir la mémoire centrale
  - Avec plus de processus en mémoire principale, il est plus probable d'avoir un processus dans l'état prêt, bonne utilisation d'UCT
- Plusieurs pages ou segments rarement utilisés n'auront peut être pas besoin d`être chargés du tout
- Il est maintenant possible d'exécuter un processus lorsque sa taille excède celle de la mémoire principale
  - Il est possible d'utiliser plus de bits pour l'adresse logique que le nombre de bits requis pour adresser la mémoire principale
  - Espace d 'adressage logique **> >** esp. d 'adressage physique

Bit de validation

- Ne charger que les pages utiles à un instant

  ☞ il faut pouvoir tester la présence d'une page en mémoire centrale



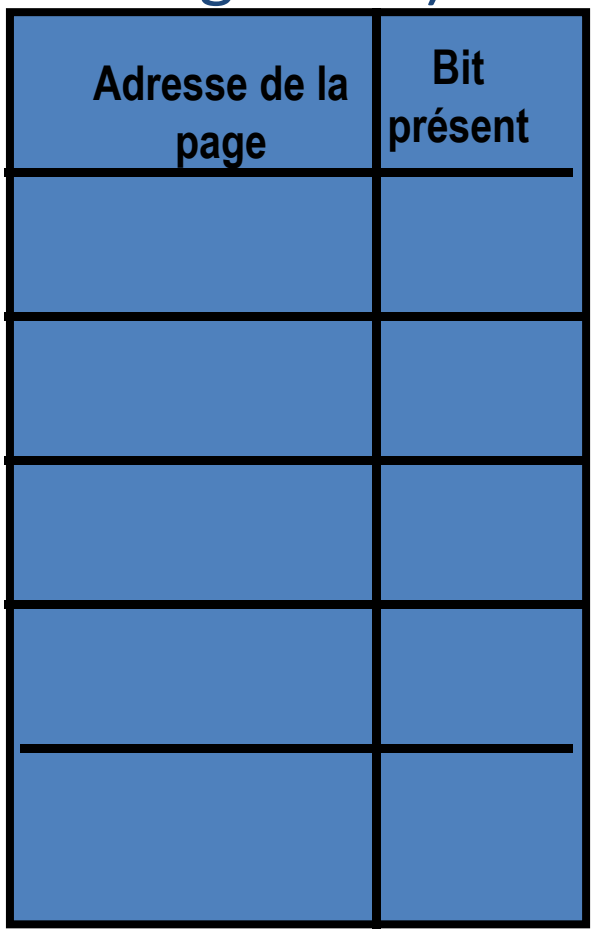


• Bit validation à vrai si la page est présente en mémoire centrale

Nouveau format du tableau des pages

(la même idée peut être appliquée aux tableaux de segments)

Si la page est en mém. princ., ceci est une adr. de mém. principale sinon elle est une adresse de mémoire secondaire

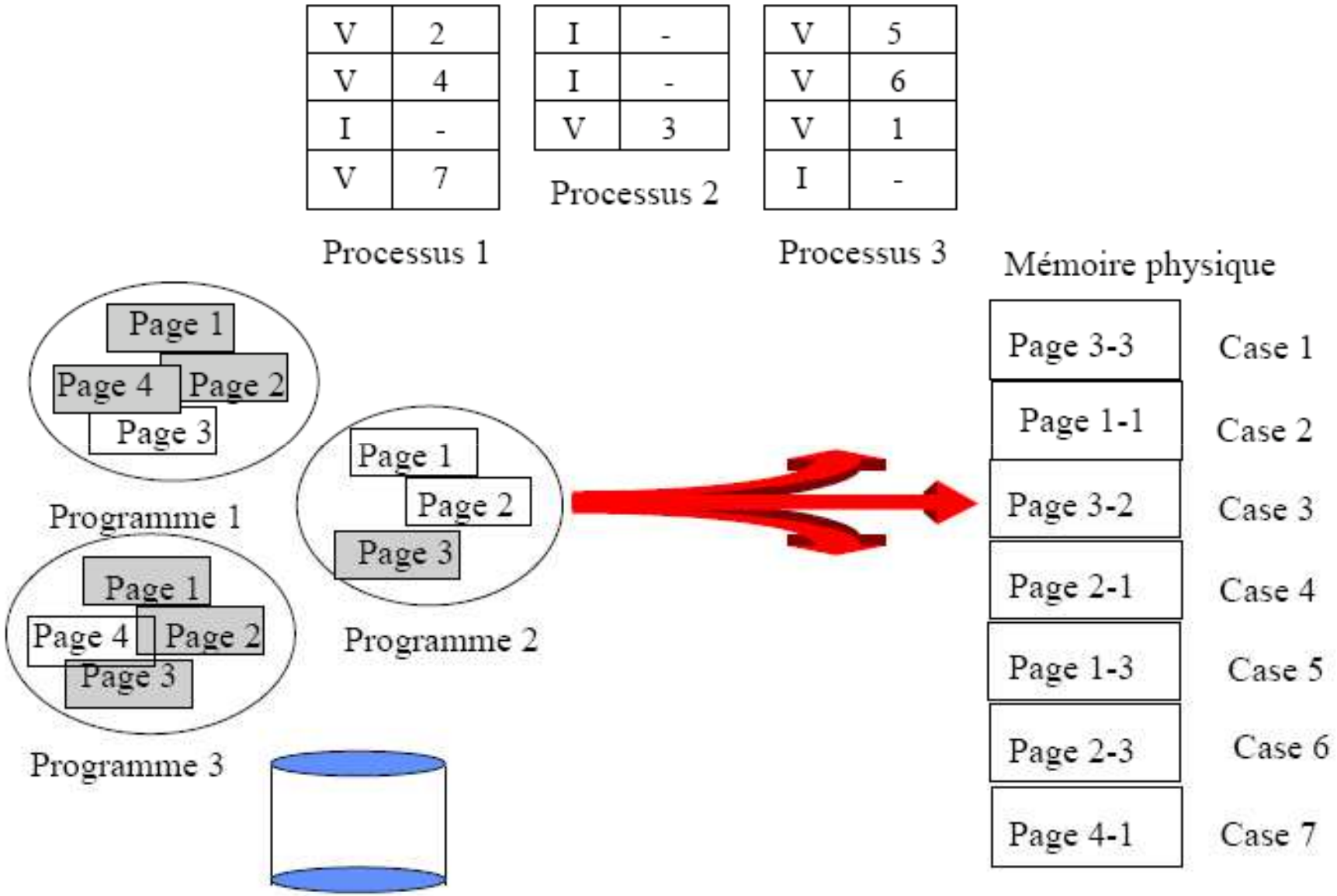


bit *présent*
1 si en mém. princ.,
0 si en mém second.

Au début, bit présent = 0

pour toutes les pages

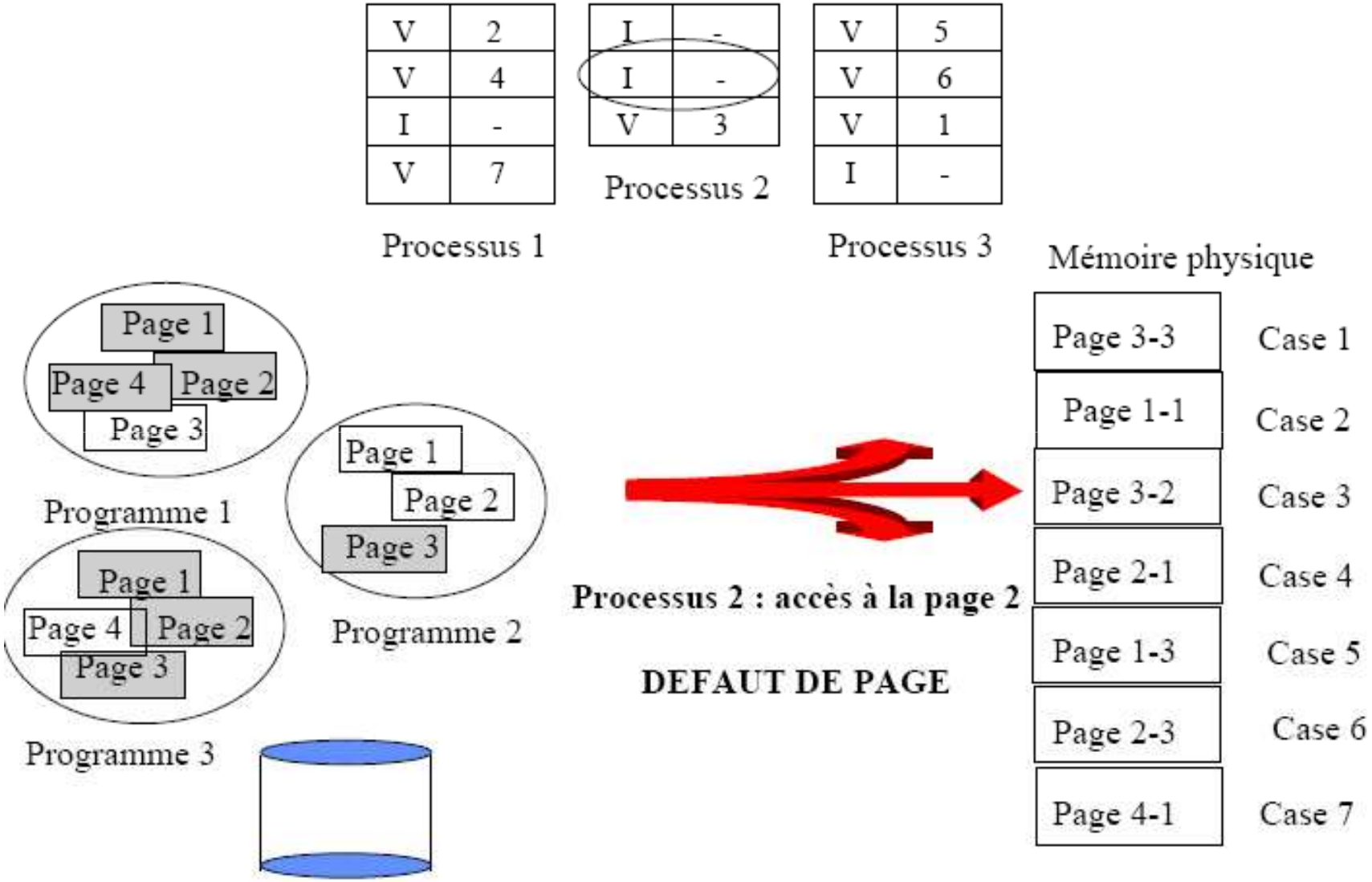
Bit de validation



Exécution d'un Processus

- Le SE charge la mémoire principale de quelques pièces (seulement) du programme (incluant le point de départ)
- Chaque entrée de la table de pages (ou segments) possède un bit présent qui est mis à 1 seulement si le morceau correspondant est en mémoire principale
- L'ensemble résident (résident set) est la portion du processus se trouvant en mémoire principale
- Une interruption est générée lorsque l'adresse logique réfère à une pièce qui n'est pas dans l'ensemble résident
  - défaut de pagination, page fault

Bit de validation et défaut de page



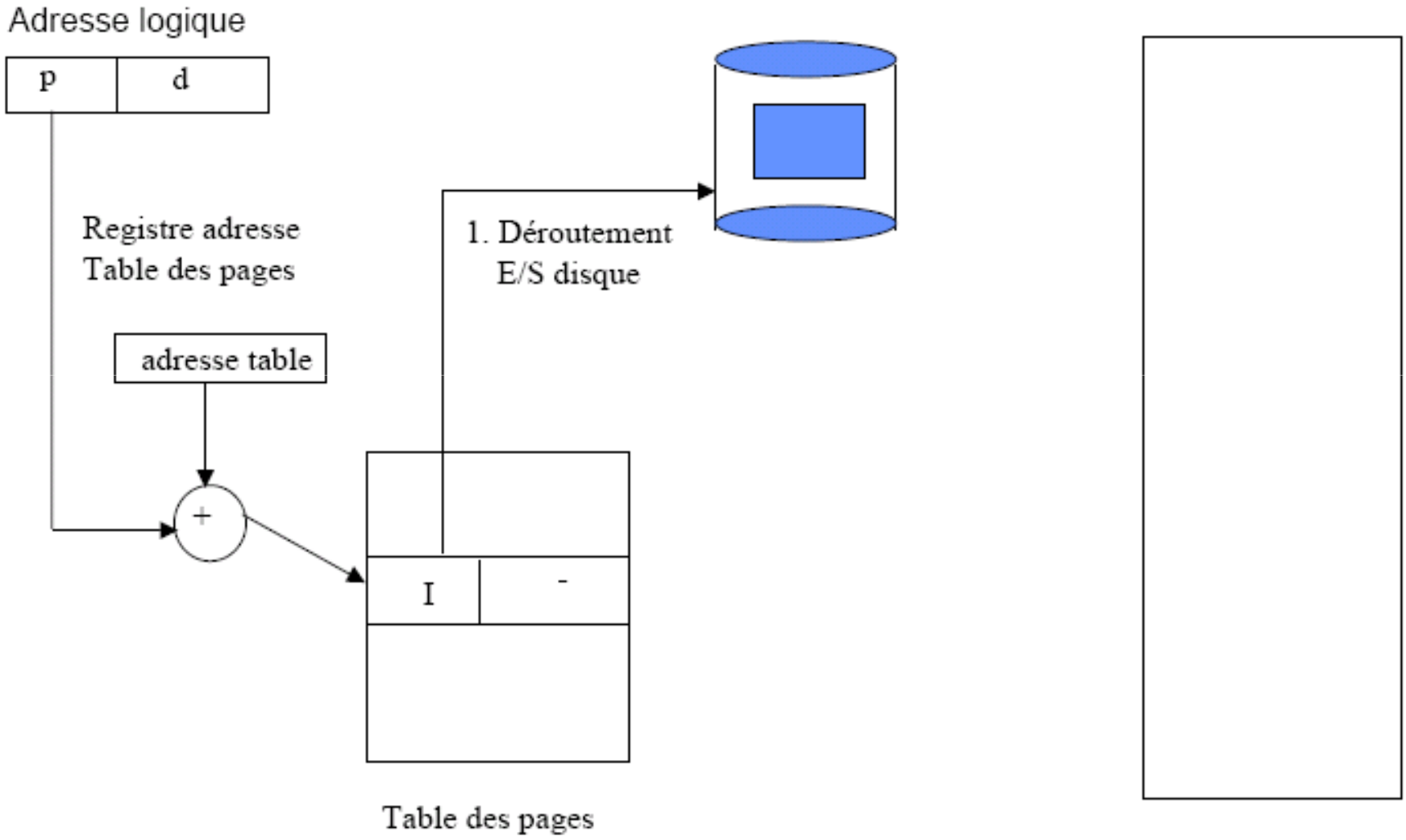
Bit de validation et défaut de page

- Ne charger que les pages utiles à un instant
  - ☞ il faut pouvoir tester la présence d'une page en mémoire centrale :
    - ☞ rôle du bit de validation
  - ☞ si un processus cherche à accéder à une page non présente en mémoire centrale, il se produit un déroutement de défaut de page

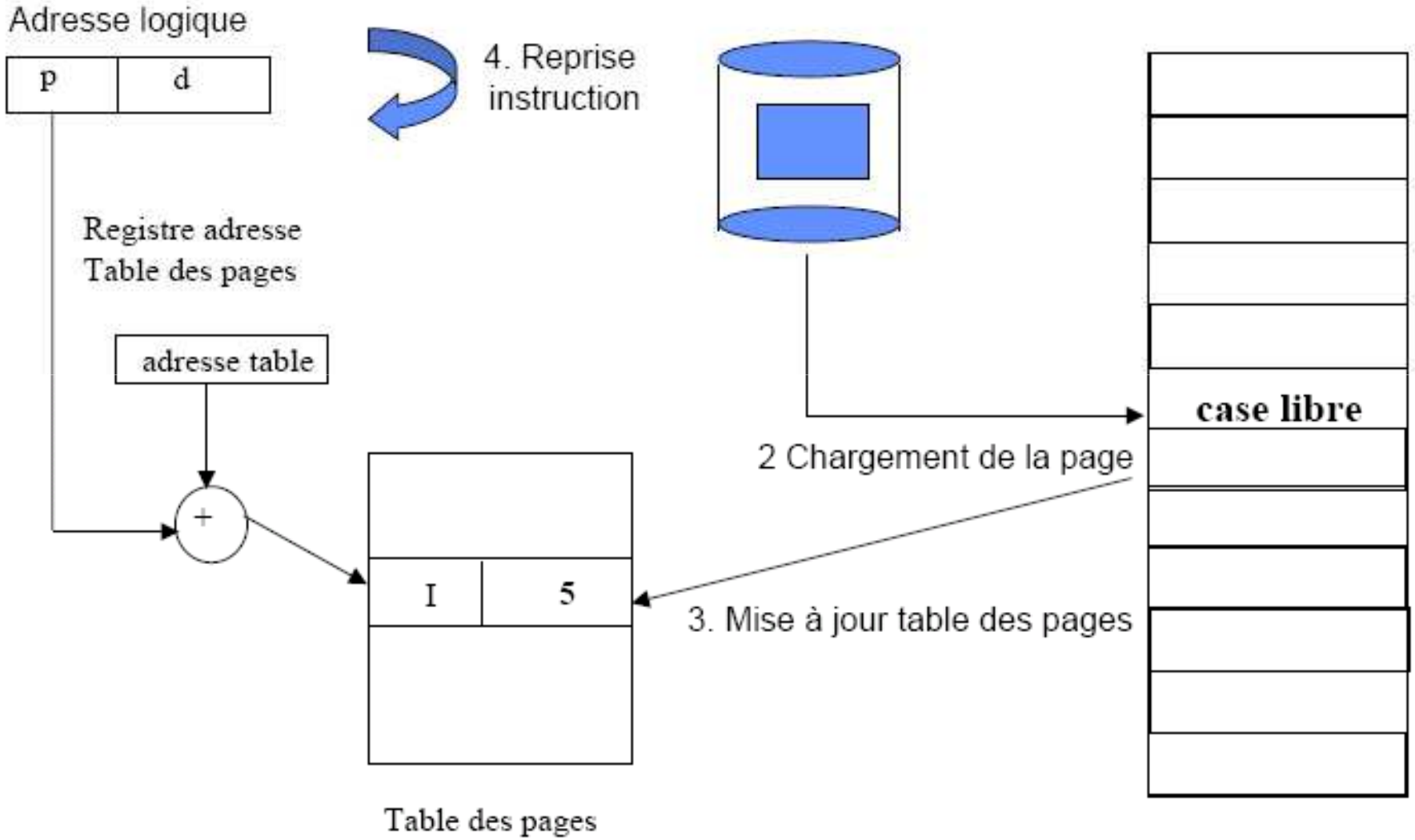


- le système d'exploitation lance une entrée/sortie disque pour charger la page en mémoire dans une case libre

Défaut de page



Défaut de page



Chargement de page

- Lors d'un défaut de page, la page manquante est chargée dans une case libre

  - la totalité des cases de la mémoire centrale peuvent être occupées
    - il faut libérer une case globalement (parmi l'ensemble des cases) ou localement (parmi les cases occupées par les pages du processus en défaut)

Remplacement de pages

- Quoi faire si un processus demande une nouvelle page et il n'y a pas de cadres libres en RAM?
- Il faudra choisir une page déjà en mémoire principale, appartenant au même ou à un autre processus, qu'il est possible d 'enlever de la mémoire principale
  - la victime!
- Un cadre de mémoire sera donc rendu disponible
- Évidemment, plusieurs cadres de mémoire ne peuvent pas être `victimisés`:
  - p.ex. cadres contenant le noyau du SE, tampons d 'E/S...

Algorithmes de remplacement pages

- Choisir la victime de façon à minimiser le taux de défaut de pages
  - pas évident!!!
- Page dont nous n`aurons pas besoin dans le futur? impossible à savoir!
- Page pas souvent utilisée?
- Page qui a été déjà longtemps en mémoire??

- etc. nous verrons...

Critères d'évaluation des algorithmes

- Les algorithmes de choix de pages à remplacer doivent être conçus de façon à minimiser le taux de défaut de pages à long terme
- Mais il ne peuvent pas impliquer des temps de système excessifs, p.ex. mise à jour de tableaux en mémoire pour chaque accès de mémoire
- Ni devraient-ils impliquer l`utilisation de matériel dispendieux

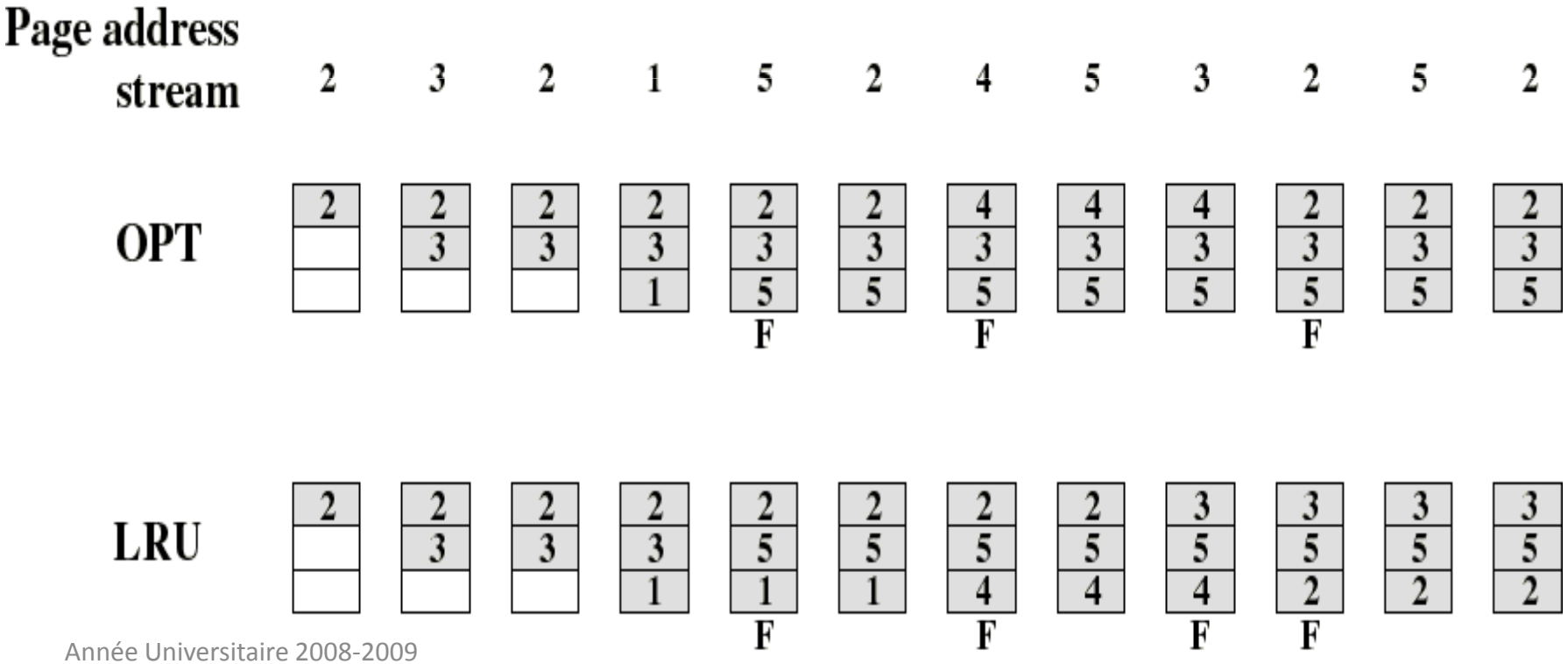
Algorithmes pour la politique de remplacement

- L'algorithme optimal (OPT) choisit pour page à remplacer celle qui sera référencée le plus tardivement
  - produit le + petit nombre de défauts de page
  - impossible à réaliser (car il faut connaître le futur) mais sert de norme de comparaison pour les autres algorithmes:
    - Ordre chronologique d'utilisation (LRU)
    - Ordre chronologique de chargement (FIFO)
    - Deuxième chance ou Horloge (Clock)

- Ordre chronologique d'utilisation (LRU)
- Remplace la page dont la dernière référence remonte au temps le plus lointain (le passé utilisé pour prédire le futur)
  - En raison de la localité des références, il s'agit de la page qui a le moins de chance d'être référencée
  - performance presque aussi bonne que l'algo. OPT

Comparaison OPT-LRU

- Exemple: Un processus de 5 pages s'ìl n`y a que 3 pages physiques disponibles.
- Dans cet exemple, OPT occasionne 3+3 défauts, LRU 3+4.



Note sur le comptage des défauts de page

- Lorsque la mémoire principale est vide, chaque nouvelle page que nous ajoutons est le résultat d'un défaut de page
- Mais pour mieux comparer les algorithmes, il est utile de garder séparés ces défauts initiaux
  - car leur nombre est le même pour tous les algorithmes

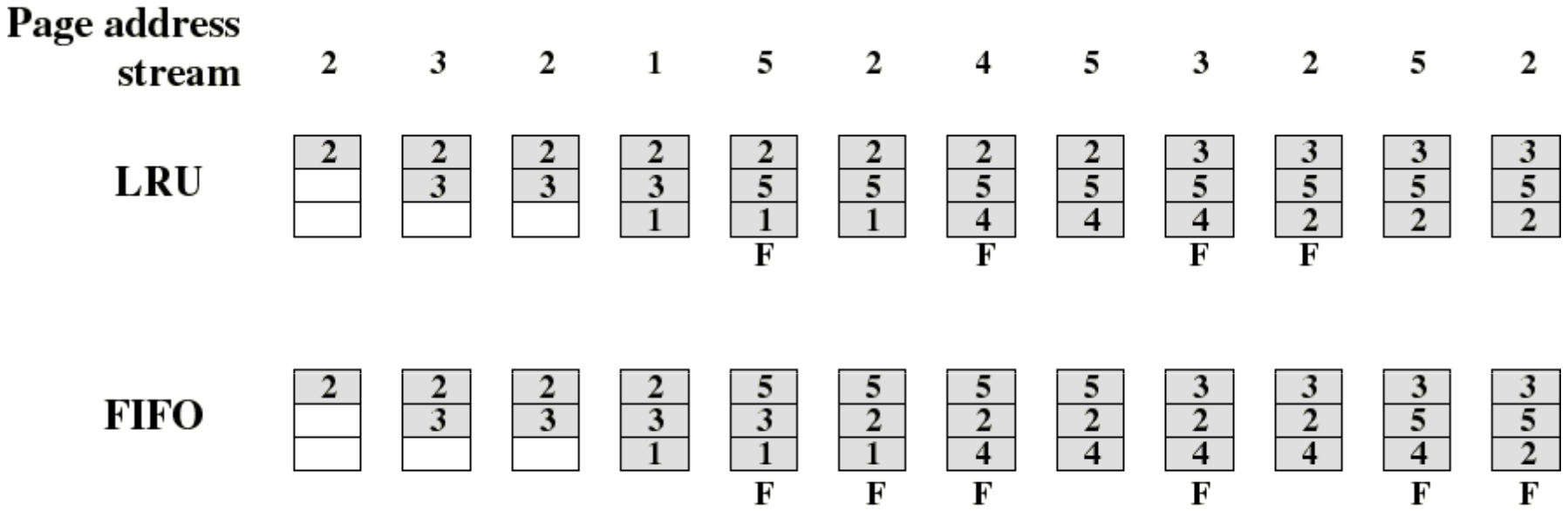
Implémentation problématique de LRU

- Chaque page peut être marquée (dans le descripteur dans la table de pages) du temps de la dernière référence: besoin de matériel supplémentaire.
- La page LRU est celle avec la + petite valeur de temps (nécessité d'une recherche à chaque défaut de page)
- Ou sinon on pourrait utiliser une liste de pages dans l'ordre d 'utilisation: perte de temps à maintenir et consulter cette liste (elle change à chaque référence de mémoire!)
- D'autres algorithmes sont utilisés:
  - LRU *approximations*

Premier arrivé, premier sorti (FIFO)

- Logique: une page qui a été longtemps en mémoire a eu sa chance d ’exécuter
- Les cadres forment conceptuellement un tampon circulaire, débutant à la plus vieille page
  - Lorsque la mémoire est pleine, la plus vieille page est remplacée. Donc: “first-in, first-out”
- Simple à mettre en application
  - tampon consulté et mis à jour seulement aux défauts de pages...
- Mais: Une page fréquemment utilisée est souvent la plus vielle, elle sera remplacée par FIFO!

Comparaison de FIFO avec LRU

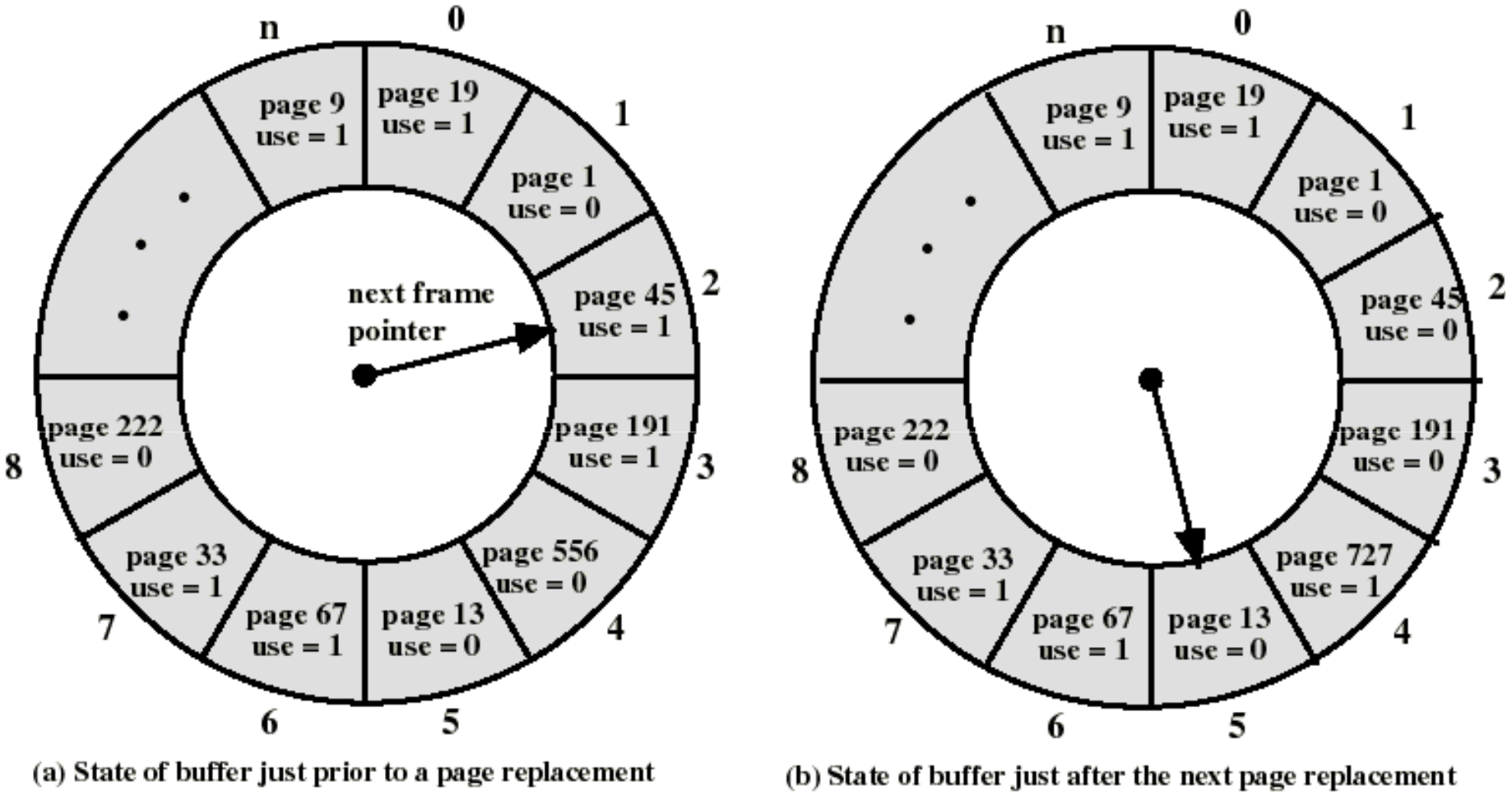


- Contrairement à FIFO, LRU reconnaît que les pages 2 et 5 sont utilisées fréquemment
- La performance de FIFO est moins bonne:
  - dans ce cas, LRU = 3+4, FIFO = 3+6

L'algorithme de l'horloge (deuxième chance)

- Semblable à FIFO, mais les cadres qui viennent d'être utilisés (bit=1) ne sont pas touchés (deuxième chance)
  - Les cadres forment conceptuellement un tampon circulaire
  - Lorsqu'une page est chargée dans un cadre, un pointeur pointe sur le prochain cadre du tampon
- Pour chaque cadre du tampon, un bit "utilisé" est mis à 1 (par le matériel) lorsque:
  - une page y est nouvellement chargée
  - sa page est utilisée
- Le prochain cadre du tampon à être remplacé sera le premier rencontré qui aura son bit "utilisé" = 0.
  - Durant cette recherche, tout bit "utilisé" = 1 rencontré sera mis à 0

Algorithme de l'horloge: un exemple

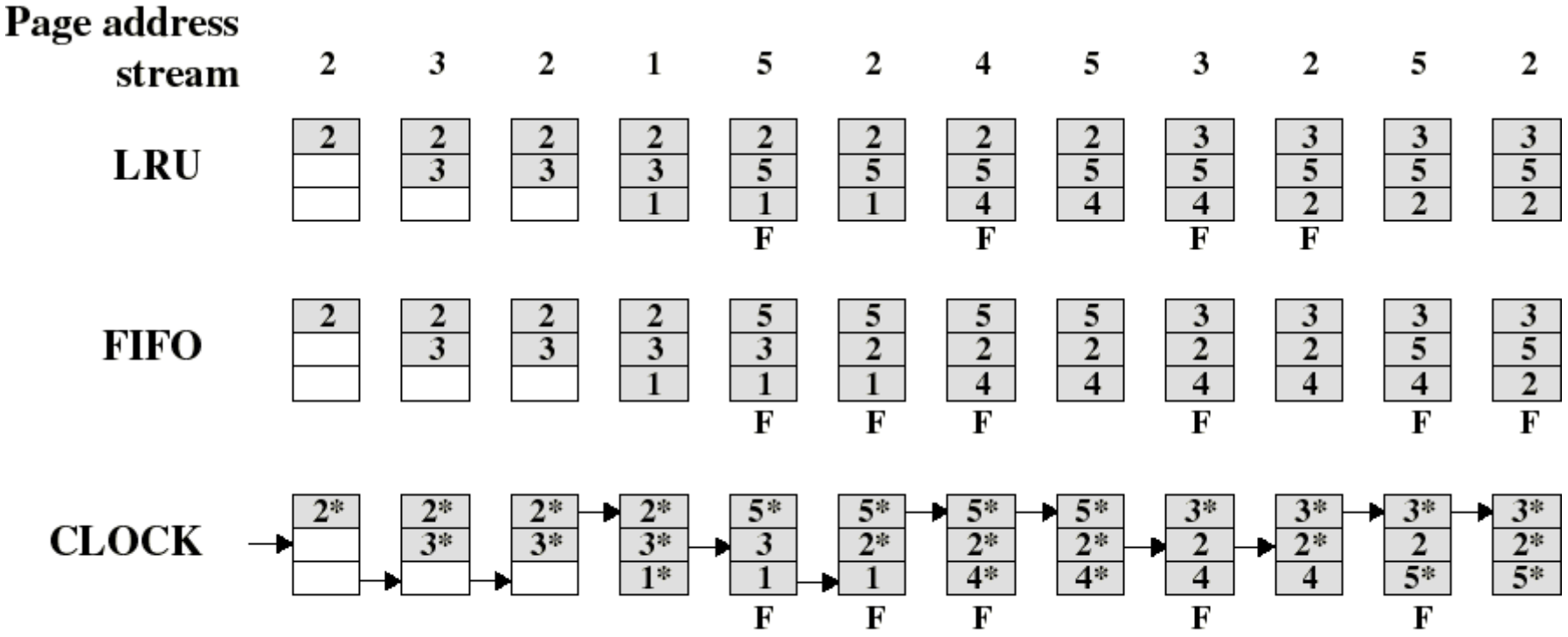


(a) State of buffer just prior to a page replacement

(b) State of buffer just after the next page replacement

La page 727 est chargée dans le cadre 4.
La proch. victime est 5, puis 8.

Comparaison: Horloge, FIFO et LRU



- Astérisque indique que le bit utilisé est mis à 1
- L'horloge protège du remplacement les pages fréquemment utilisées en mettant à 1 le bit "utilisé" à chaque référence
- LRU = 3+**4**, FIFO = 3+**6**, Horloge = 3+**5**

Comparaison: Horloge, FIFO et LRU

- **Les simulations montrent que l'horloge est presque aussi performant que LRU**
  - **variantes de l`horloge ont été implantées dans des systèmes réels**
- **Lorsque les pages candidates au remplacement sont locales au processus souffrant du défaut de page et que le nombre de cadres alloué est fixe, les expériences montrent que:**
  - **Si peu (6 à 8) de cadres sont allouées, le nombre de défaut de pages produit par FIFO est presque double de celui produit par LRU, et celui de CLOCK est entre les deux**
  - **Ce facteur s'approche de 1 lorsque plusieurs (plus de 12) cadres sont alloués.**
- **Cependant le cas réel est de milliers et millions de pages et cadres, donc la différence n`est pas trop importante en pratique...**
  - **On peut tranquillement utiliser LRU**

Algorithmes *compteurs*

- **Garder un compteur pour les références à chaque page**
- **LFU: Least Frequently Used: remplacer la page avec le plus petit compteur**
- **MFU: Most Frequently Used: remplacer les pages bien usées pour donner une chance aux nouvelles**
- **Ces algorithmes sont d'implantation dispendieuse et ne sont pas beaucoup utilisés**

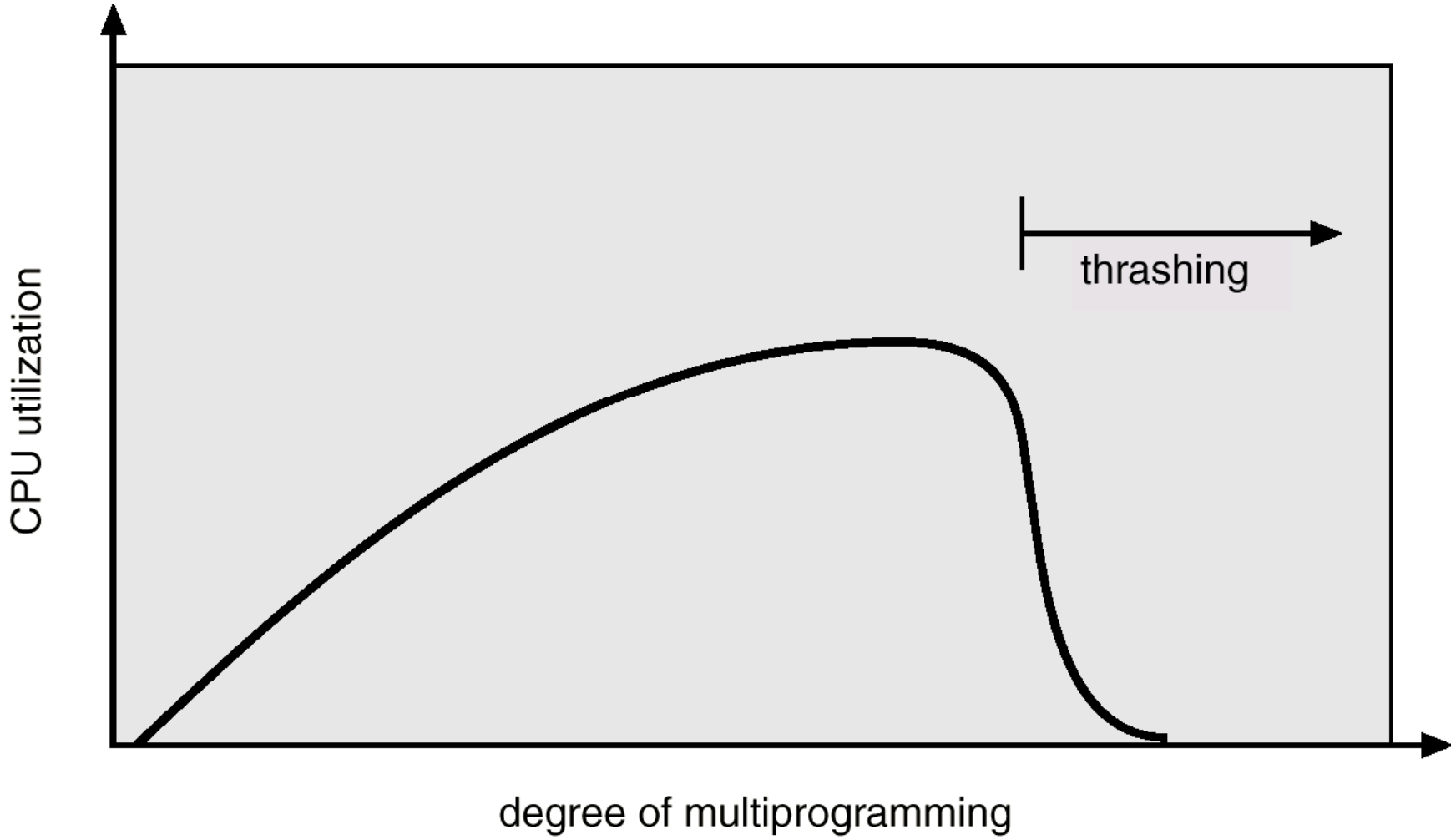
Utilisation d'une pile (stack)

- Quand une page est utilisée, est mise au sommet de la pile.
  - donc la page la plus récemment utilisée est toujours au sommet,
  - la moins récemment utilisée est toujours au fond
- Bonne implémentation du principe de localité, cependant…
- La pile doit être implantée par matériel, car nous ne pouvons pas tolérer l'exécution d'un programme à chaque fois qu'une page est utilisée
- Donc pas pratique

Écroulement ou thrashing (liter.: défaite)

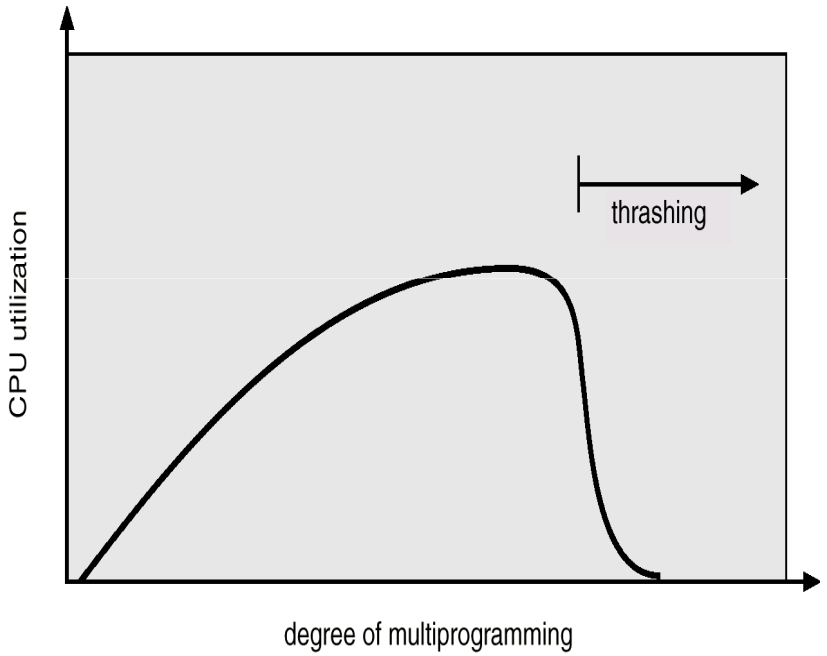
- **S ’il n`y a pas assez de mémoire pour exécuter un proc sans trop de défauts de pagination, le proc finira pour passer trop de temps dans les files d`attente**
- **Si cette situation se généralise à plusieurs procs, l ’UCT se trouvera à être sous-utilisée**
- **Le SE pourra chercher de remédier à cette situation en augmentant le niveau de multiprogrammation**
  - **plus de procs en mémoire!**
  - **moins de mém par proc!**
  - **plus de défauts de pagination!**
- **Désastre: écroulement**
  - **le système devient entièrement occupé à faire des E/S de pages, il ne réussit plus à faire de travail utile**

Écroulement



La raison de l`écroulement

- **Chaque processus a besoin d ’un certain nombre de pages pour exécuter efficacement**
- **Le nombre de pages dont l’ensemble de processus a besoin à l’instant excède le nombre de cadres de mémoire disponible**
  - **défaite du concept de mémoire virtuelle**

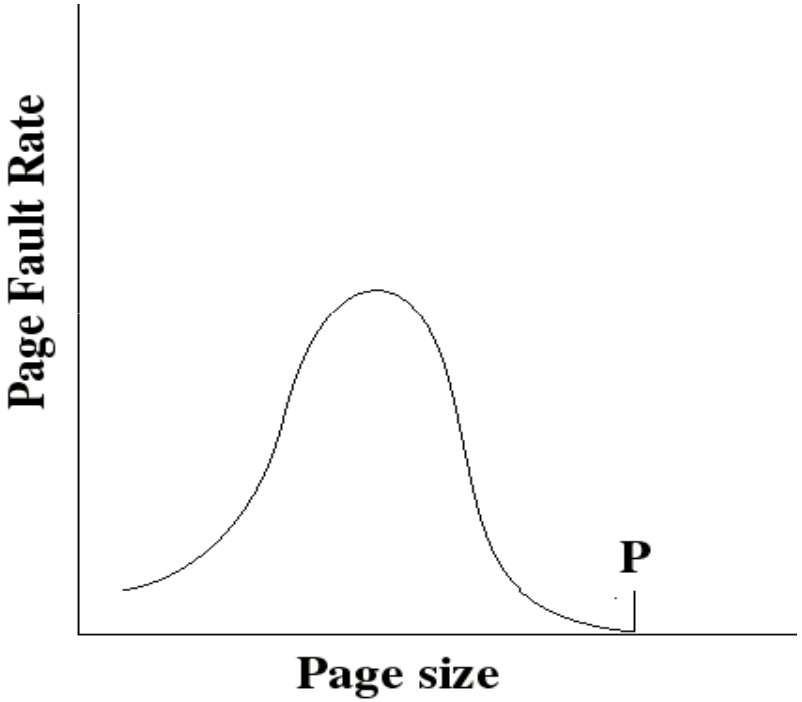


Ensemble de travail (working set)

- L'ensemble de travail d'un proc donné à un moment d'exécution donné est l'ensemble des pages dont le proc a besoin pour exécuter sans trop de défauts de pagination
  - Malheureusement, un concept flou

Pages grandes ou petites

- Avec une petite taille, nous pourrons
  - mettre un grand nombre de pages en mém. centrale.
  - chaque page contient uniquement le code utilisé
  - peu de défauts de page
- En augmentant la taille
  - moins de pages peuvent être gardées dans RAM
  - chaque page contient plus de code qui n'est pas utilisé
  - + de défauts de page
- Mais ils diminuent lorsque nous approchons le point P où la taille d'une page est celle d'un programme entier



Conclusion

- Il est fortement désirable que l`espace d ’adressage de l ’usager puisse être beaucoup plus grand que l ’espace d ’adressage de la mémoire RAM
- Le programmeur sera donc libéré de la préoccupation de gérer son occupation de mémoire
  - cependant, il devra chercher à maximiser la localité de son processus
- La mémoire virtuelle aussi permet à plus de processus d ’être en exécution
  - UCT, E/S plus occupées
- Le problème de décider la page victime n ’est pas facile.
  - Les meilleurs algorithmes sont impossibles ou difficiles à implanter
  - Cependant en pratique l ’algorithme FIFO est acceptable