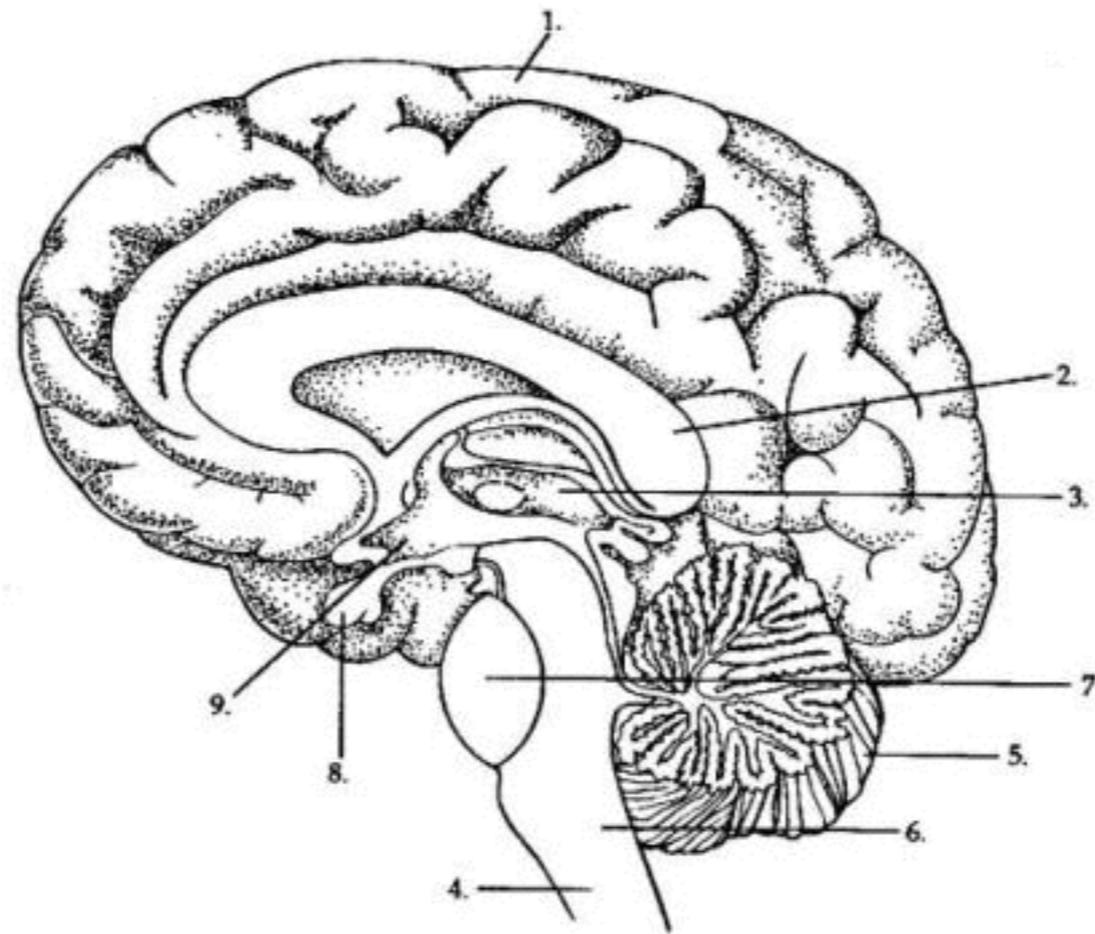


# Gestion de la mémoire



# Rappel: Allocation de la mémoire pour le DOS

- La mémoire du DOS est séparée en plusieurs parties:
  - le système d'exploitation
  - le programme de l'utilisateur
  - les instructions de démarrage
  - la table des vecteurs d'interruption...
- Nous avons vu que le DOS allouait de la mémoire pour un seul programme à la fois.

# Cette semaine

- Mémoire:
  - contigüe
  - paginée
- Adresses physiques et virtuelles

# Buts de la gestion mémoire

- Deux buts principaux:
  - Utilisation simple pour un programme
  - Maximiser l'utilisation de la mémoire disponible

# Allocation contiguë, partitions de taille variable ou fixe

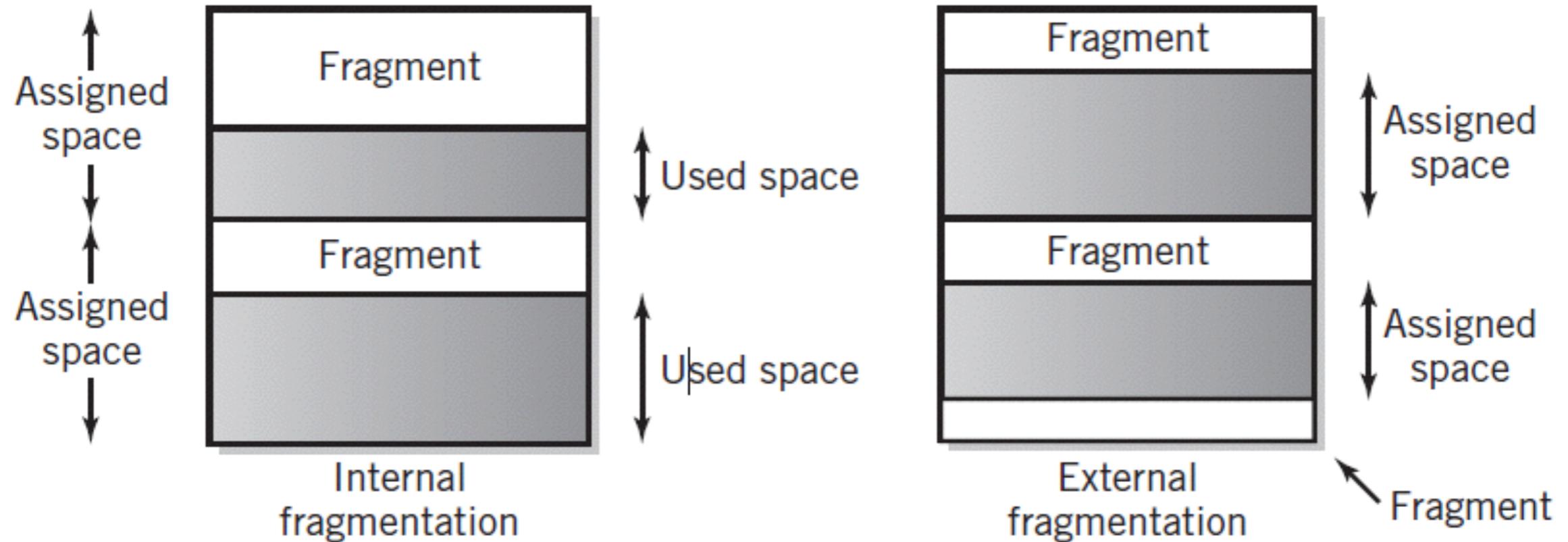
- L'espace mémoire pour les programmes peut être alloué dans des partitions de taille:
  - **fixe**: L'emplacement des partitions est alors prédéterminé.
    - méthode d'allocation de mémoire la plus simple, mais beaucoup d'espace peut être perdu si les programmes chargés en mémoire sont plus petits que les partitions.
  - **variables**: Dans ce cas, les partitions sont créées de la même taille que les programmes qu'elles contiennent. Cela implique que l'emplacement de chaque partition est variable.
    - la mémoire est mieux utilisée, mais le mécanisme est plus difficile à gérer. Par exemple, il faut maintenir une liste des espaces de mémoire disponibles.

# Fragmentation de la mémoire

- On définit un bloc de mémoire comme étant un espace contigu de mémoire.
- La fragmentation de mémoire est une mesure du nombre de blocs de mémoire qui sont libres (qui ne contiennent pas le OS ou un processus). Une mémoire fragmentée est une mémoire dans laquelle plusieurs blocs de mémoire non contigus sont libres.
- L'allocation contiguë de partitions à taille fixe crée de la fragmentation interne. Entre chaque partition de taille fixe, un peu de mémoire est perdue parce que le programme contenu dans la partition n'a pas nécessairement la même taille que la partition.
- L'allocation contiguë de partitions à taille variable crée de la fragmentation externe. Lorsqu'un programme est retiré de la mémoire, il laisse un bloc de mémoire libre. Il est possible, par la suite, que ce bloc soit rempli partiellement par un processus de taille moindre (dans une nouvelle partition). Il reste alors de la mémoire libre à l'extérieur des partitions.
- Une mémoire très fragmentée est une mémoire lente et une mémoire dans laquelle des blocs (programmes) de grandes dimensions ne peuvent plus être alloués, car l'espace libre est répartie partout dans le mémoire. Il existe des méthodes de compaction (pour la mémoire) et défragmentation (pour un disque dur) afin de réduire la fragmentation.

# Fragmentation interne vs. externe

Internal and External Fragmentation



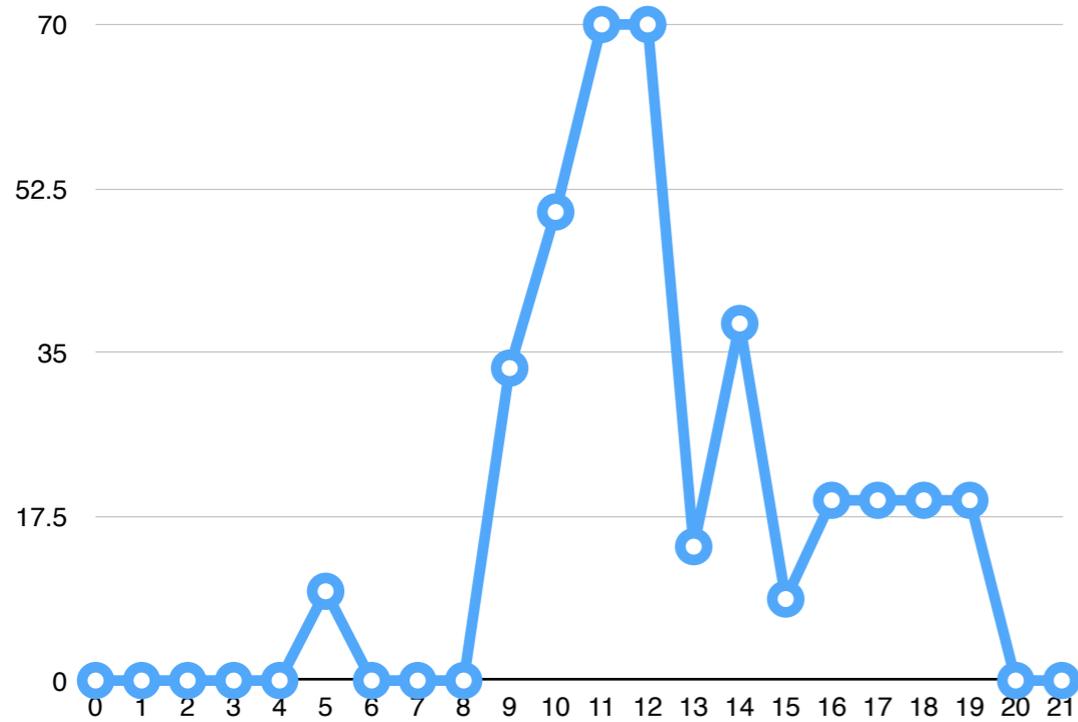
# Algorithmes d'allocation de mémoire

- Il existe plusieurs algorithmes afin de déterminer l'emplacement d'un programme en mémoire (allocation contiguë). Le but de tous ces algorithmes est de maximiser l'espace mémoire occupé.
  - **First Fit:** Le programme est mis dans le premier bloc de mémoire suffisamment grand à partir du début de la mémoire.
  - **Next Fit:** Le programme est mis dans le premier bloc de mémoire suffisamment grand à partir du dernier bloc alloué.
  - **Best Fit:** Le programme est mis dans le bloc de mémoire le plus petit dont la taille est suffisamment grande pour l'espace requis.
  - **Worse Fit:** Le programme est mis dans le bloc de mémoire le plus grand.
- Même si "Best Fit" semble le meilleur, ce n'est pas toujours l'algorithme retenu en pratique car il demande beaucoup de temps de calcul. Par ailleurs, Next Fit crée un peu plus de fragmentation que First Fit. Aussi First Fit, malgré sa simplicité apparente, est souvent utilisé.

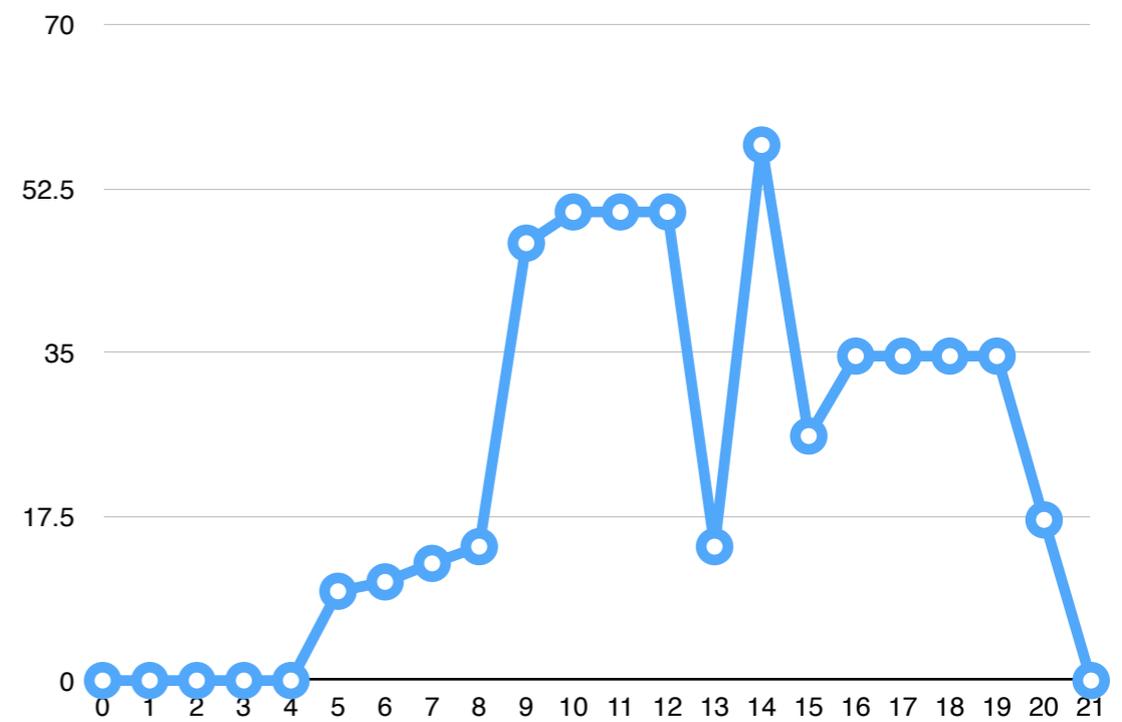
Démonstration

# Démonstration — fragmentation

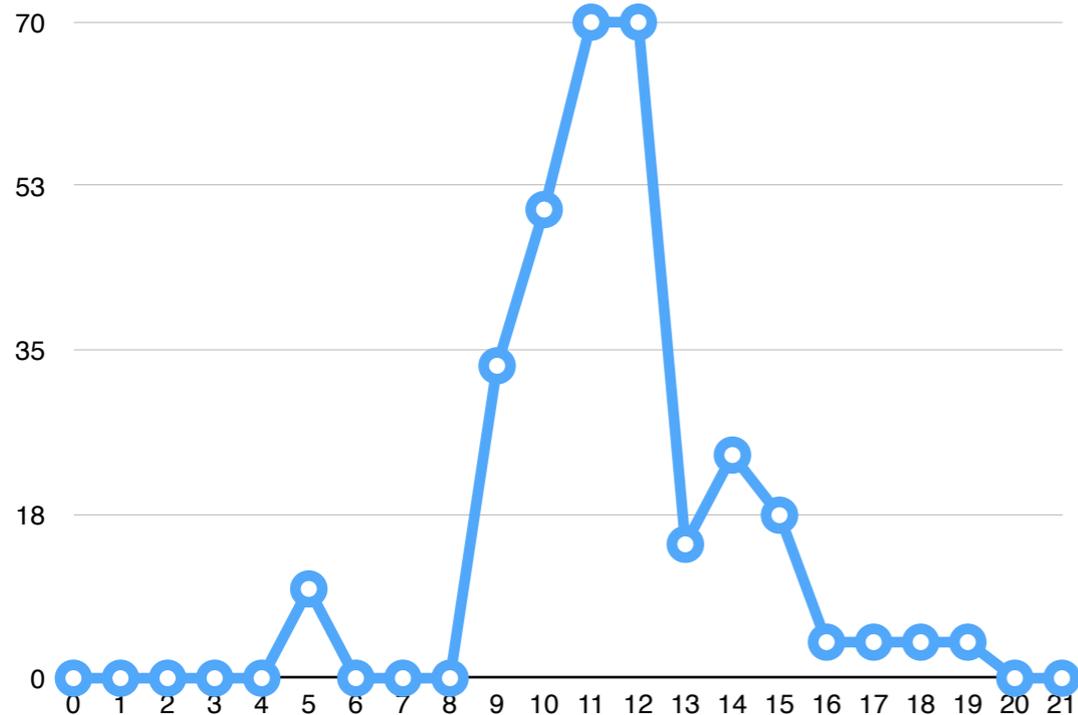
First fit (moyenne = 15.83%)



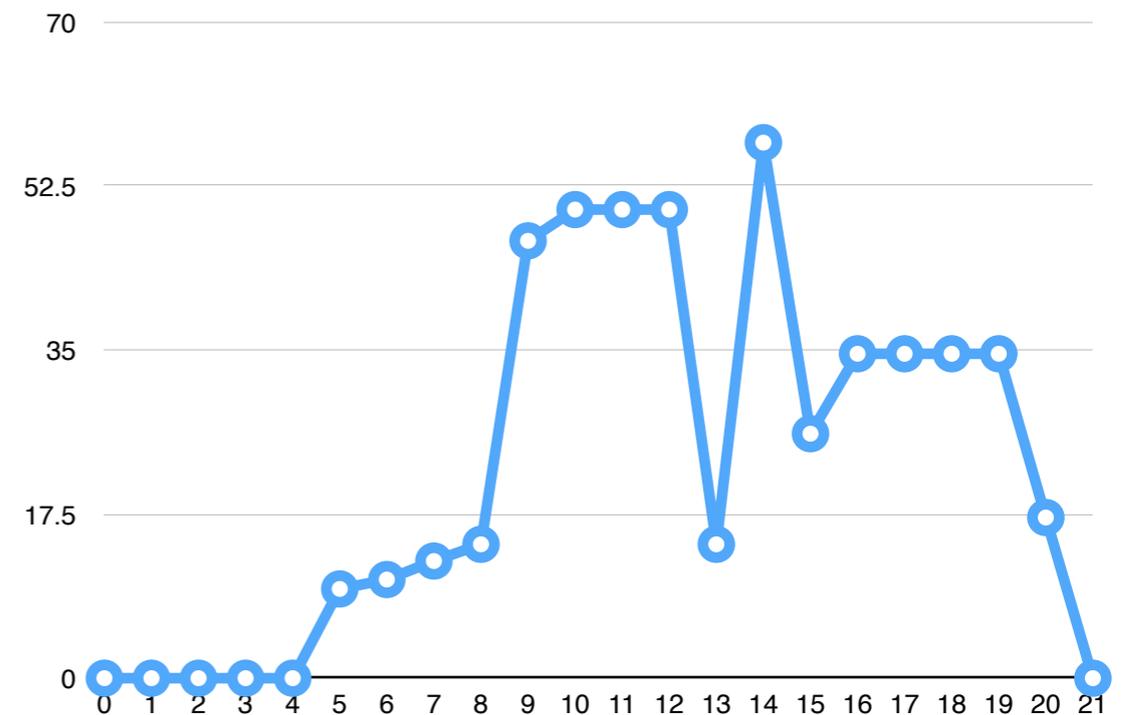
Next fit (moyenne = 16.25%)



Best fit (moyenne = 15%)

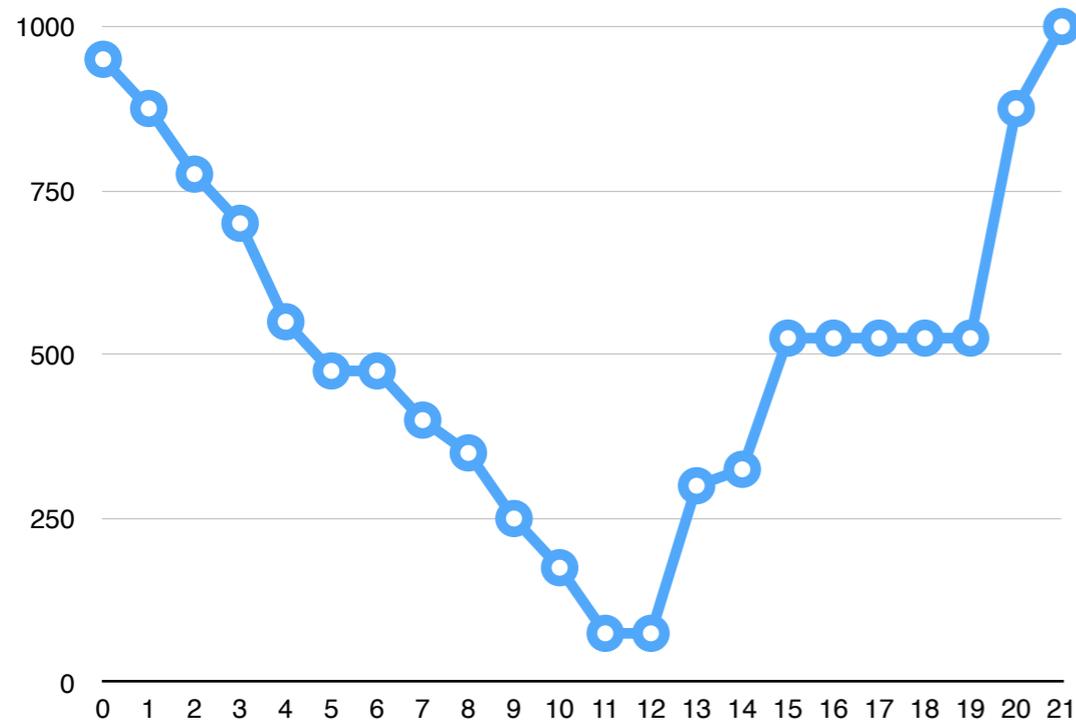


Worst fit (moyenne = 16.25%)

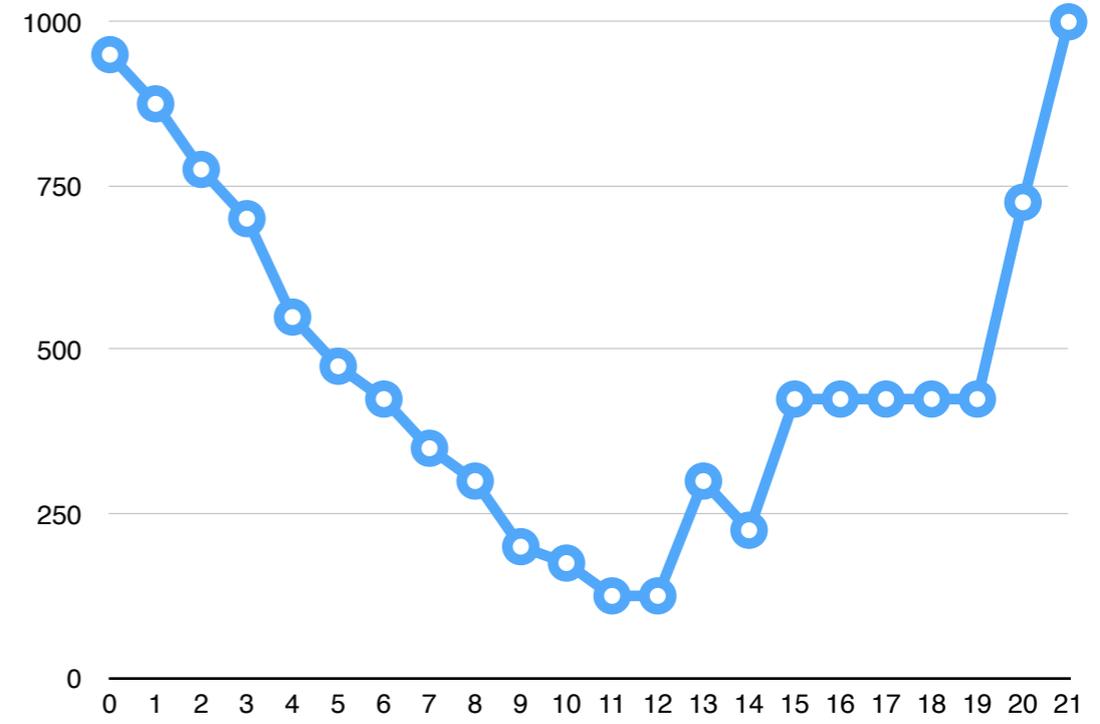


# Démonstration — taille du plus grand bloc libre

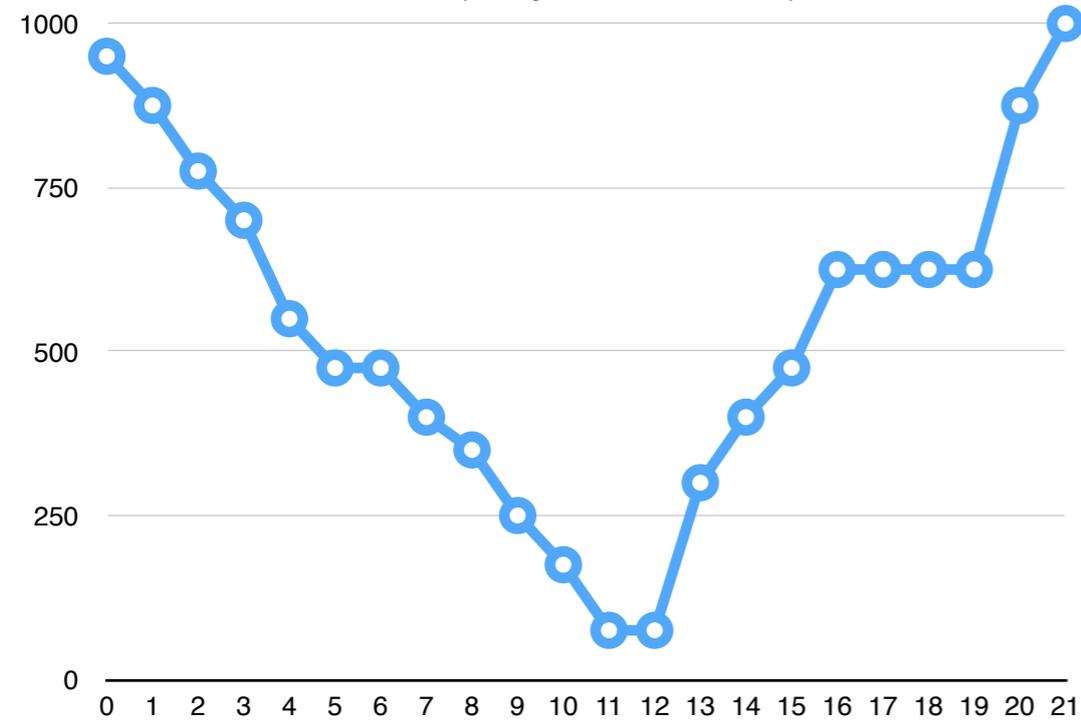
First fit (moyenne = 511)



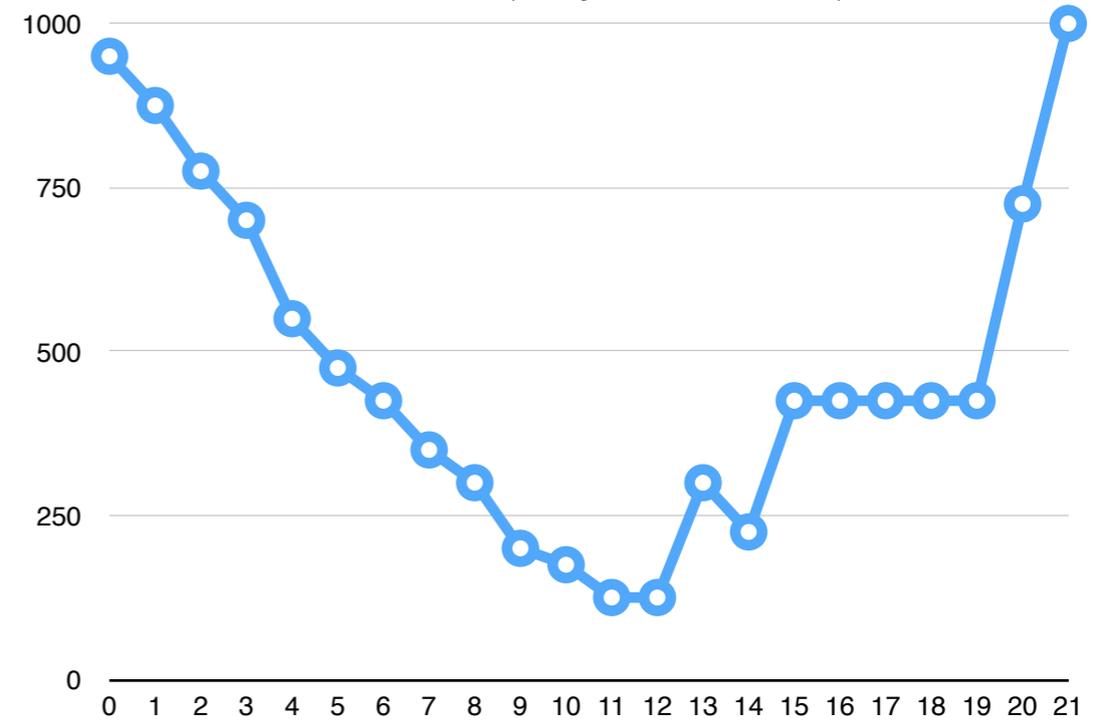
Next fit (moyenne = 472)



Best fit (moyenne = 530)

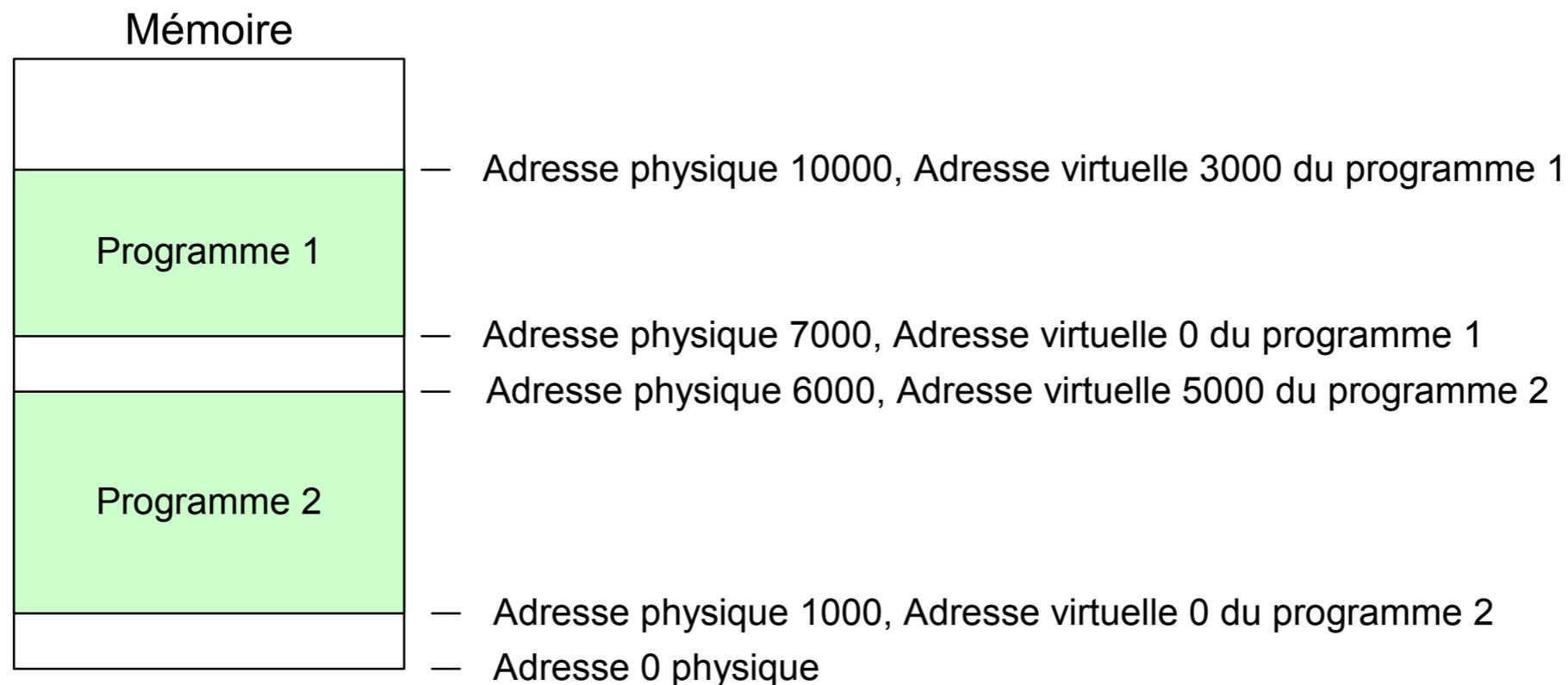


Worst fit (moyenne = 472)



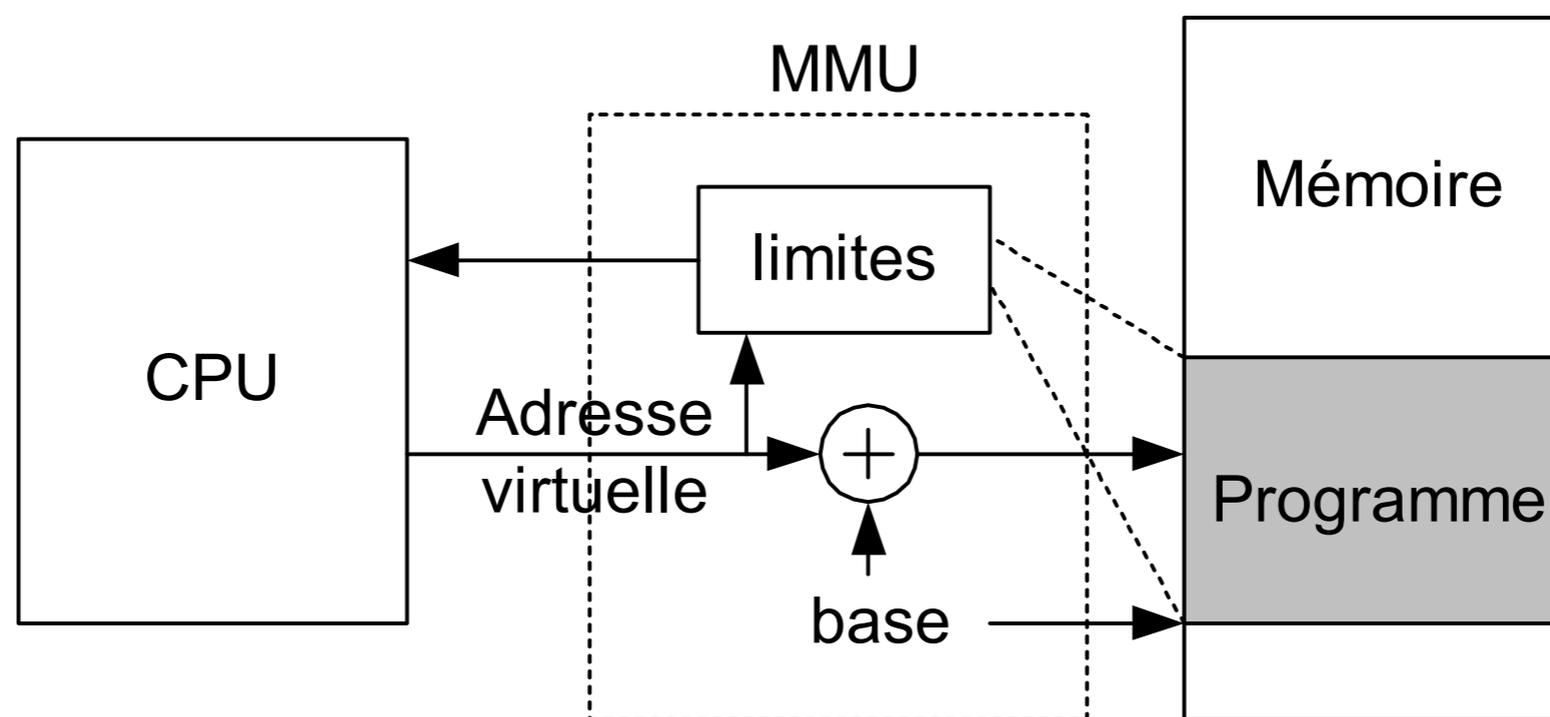
# Adresse virtuelle et adresse physique

- Chaque programme a ses adresses internes (adresse logique ou virtuelle) et une adresse réelle (adresse physique) en mémoire.
- Le système d'exploitation place le programme à un endroit donné de la mémoire.
- Le microprocesseur, et souvent les premières caches de l'ordinateur, utilise des adresses virtuelles (du programme) pour exécuter les programmes.
- La translation d'adresse consiste à traduire une adresse virtuelle en adresse physique. Elle se fait avec du matériel spécialisé: le Memory Management Unit (MMU).



# Allocation contiguë et translation d'adresse

- L'allocation contiguë de mémoire consiste à placer les programmes entiers dans une zone unique de la mémoire.
- Plusieurs programmes peuvent être mis en mémoire, chaque programme occupant un bloc unique de mémoire
- Pour faire la transition entre l'adresse réelle et l'adresse logique, le Memory Management Unit peut être conçu de façon très simple: il suffit de d'ajouter l'adresse de base du programme (sa première adresse dans la mémoire physique) à son adresse dans le programme (adresse virtuelle).

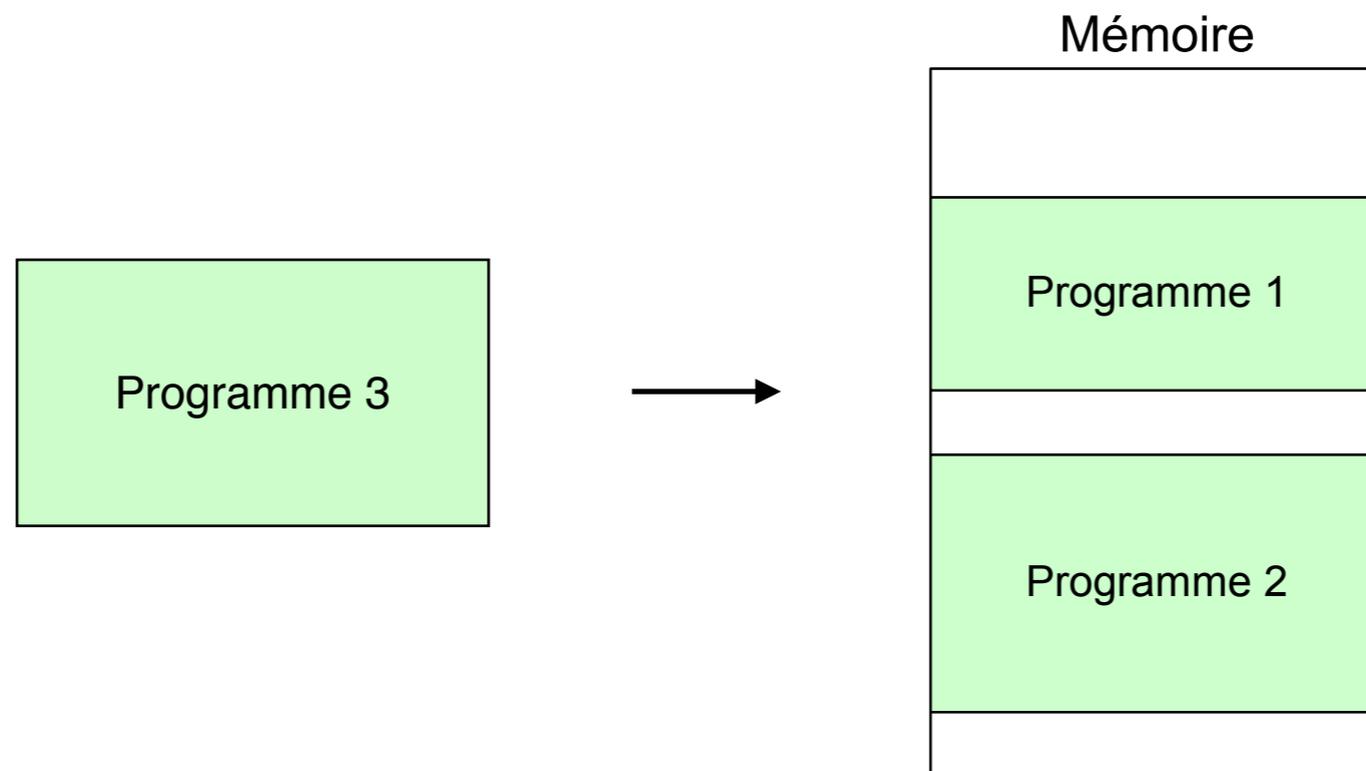


# Récapitulation

1. Un nouveau programme est copié dans un emplacement disponible en mémoire, de façon contigüe.
2. On peut créer des partitions de taille:
  - **fixe**: la première partition disponible est choisie quand un nouveau processus doit être alloué
  - **variable**: on doit déterminer où créer la partition, nécessite le choix d'un algorithme d'allocation mémoire plus compliqué
3. Le programme utilise des adresses "virtuelles"
4. Le MMU traduit les adresse virtuelles en adresses physiques

# Question

- Que faire avec “Programme 3”?

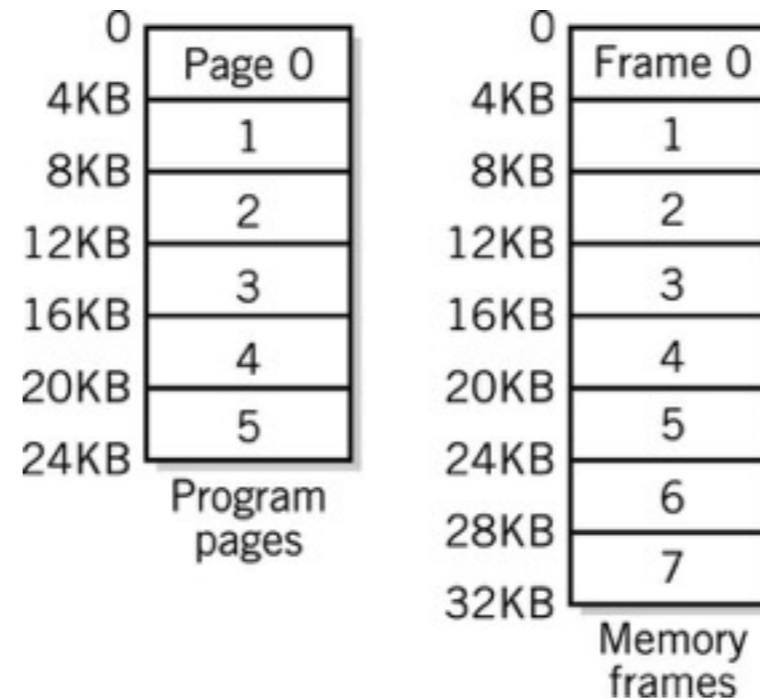


# Mémoire paginée

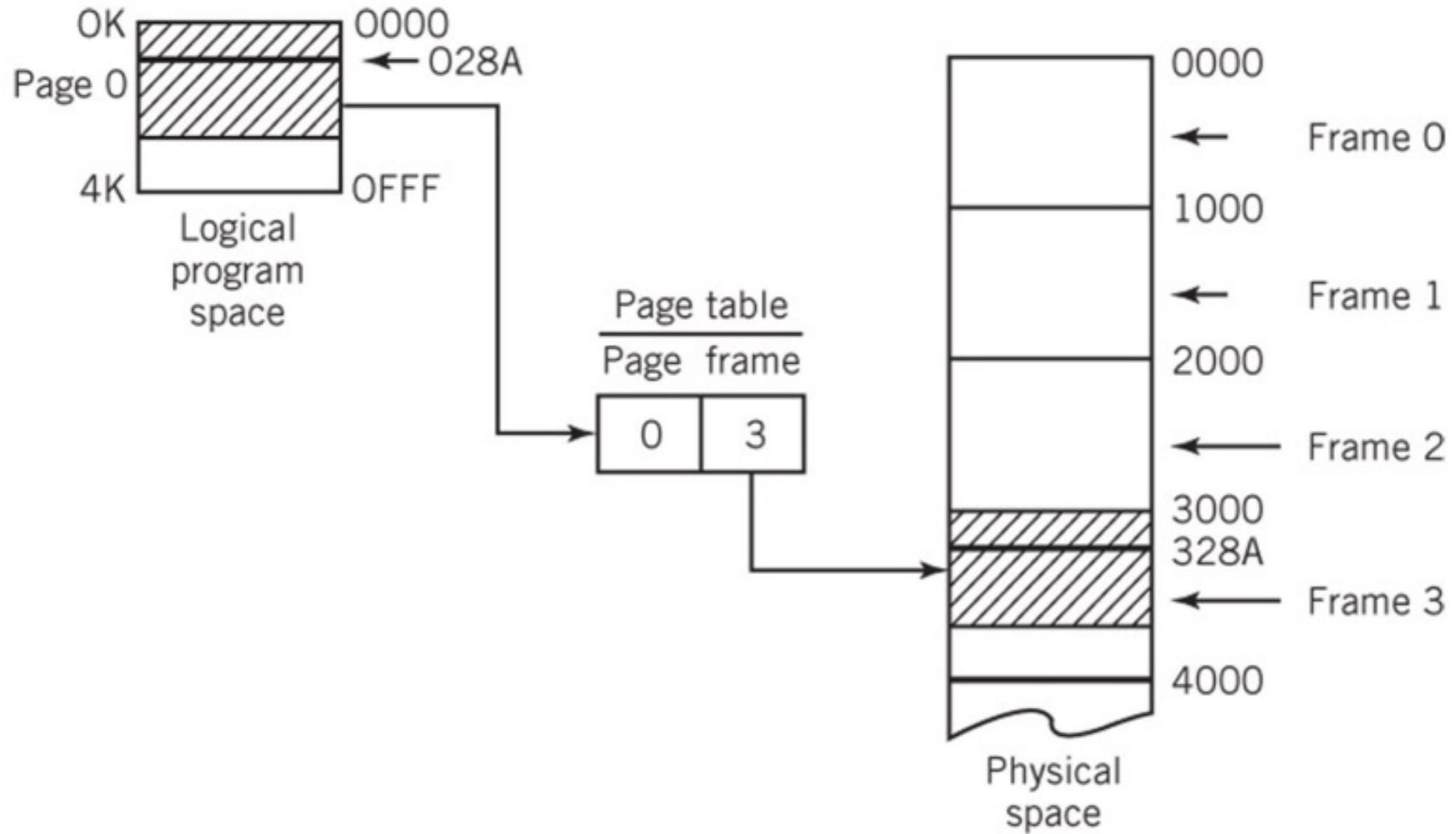
- Un gros programme peut être difficile à placer de manière contiguë en mémoire. La solution à ce problème consiste à séparer le programme en petites parties de taille fixe: des pages. Chaque partie se retrouve à un endroit différent de la mémoire.
- Il faut maintenir une table de correspondance entre les pages des programmes et les pages de la mémoire (appelées "frames").
  - La table de correspondance s'appelle Table des pages ("page table").
  - Il peut y avoir une table par programme ou une table pour tous les programmes.
- Le principe de localité (les prochaines instructions à exécuter d'un programme sont probablement adjacentes à l'instruction en cours ou celle en cours de nouveau) permet d'avoir quelques pages d'un programme seulement dans la mémoire physique. Mettre une partie seulement du programme dans la mémoire permet d'économiser de l'espace mémoire précieux.
- Ajouter de l'information dans la table des pages permet de dire si les pages d'un programme sont dans la mémoire ou encore sur le disque.

# Pages et “frames”

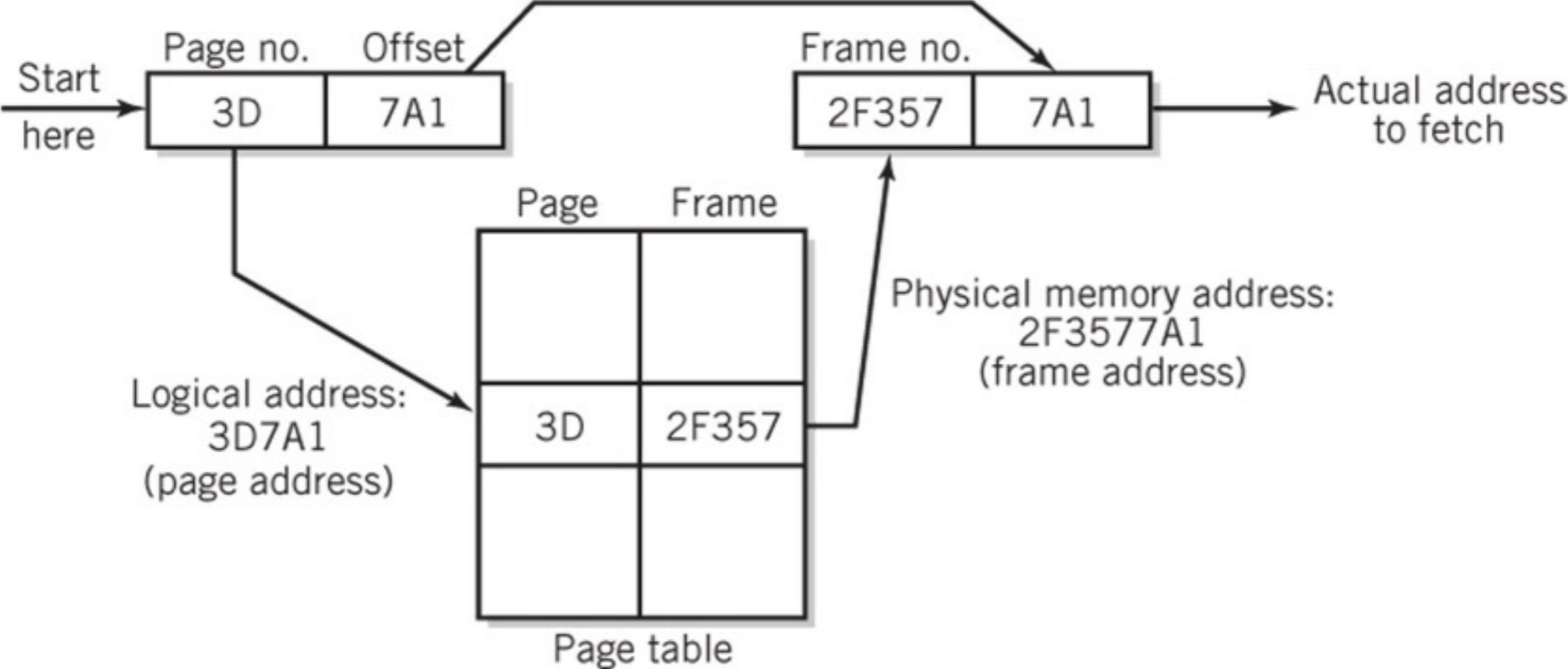
- Chaque programme possède sa propre collection de pages
- Est-ce que le nombre de pages peut être plus grand que le nombre de frames?



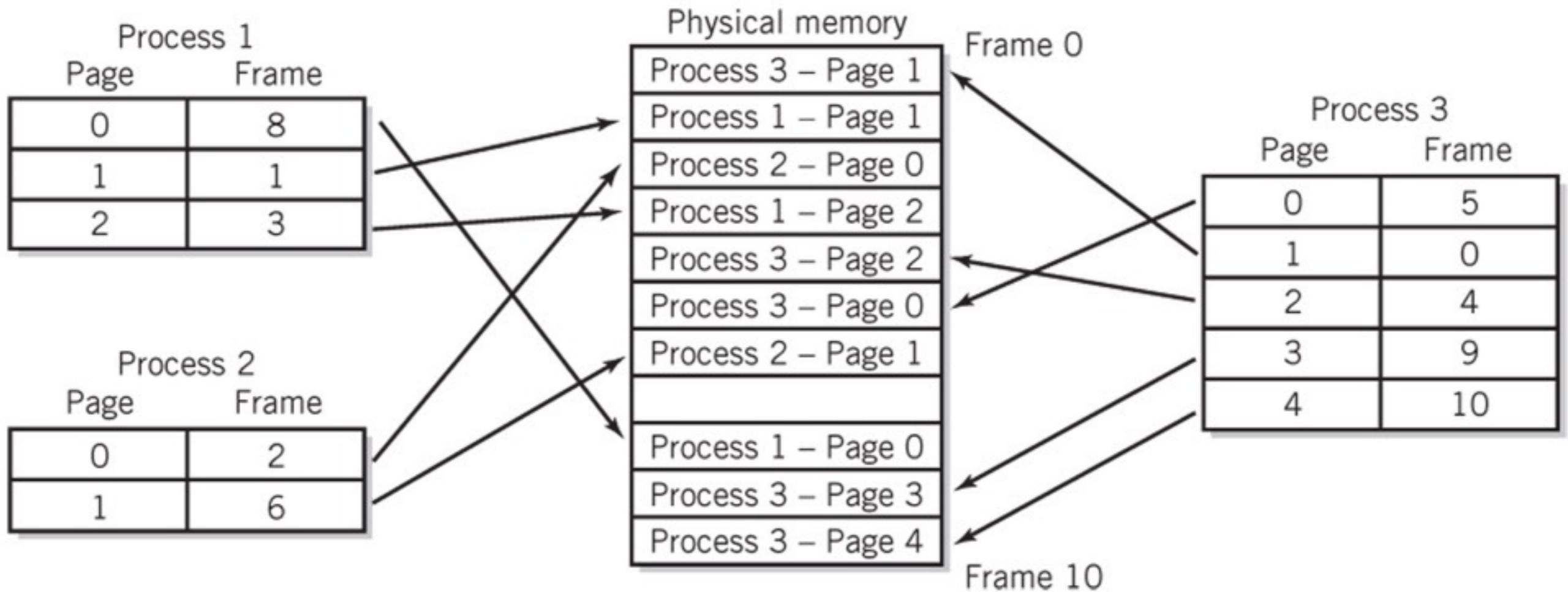
# Traduction d'adresses



# Table des pages



# Illustration de la mémoire virtuelle



# Conditions

- 2 conditions pour qu'un programme puisse s'exécuter:
  - l'instruction (ou la donnée) nécessaire doit être en mémoire RAM
  - la table de pages pour ce programme doit contenir une entrée qui traduit l'adresse (virtuelle) du programme vers l'adresse (physique) en RAM

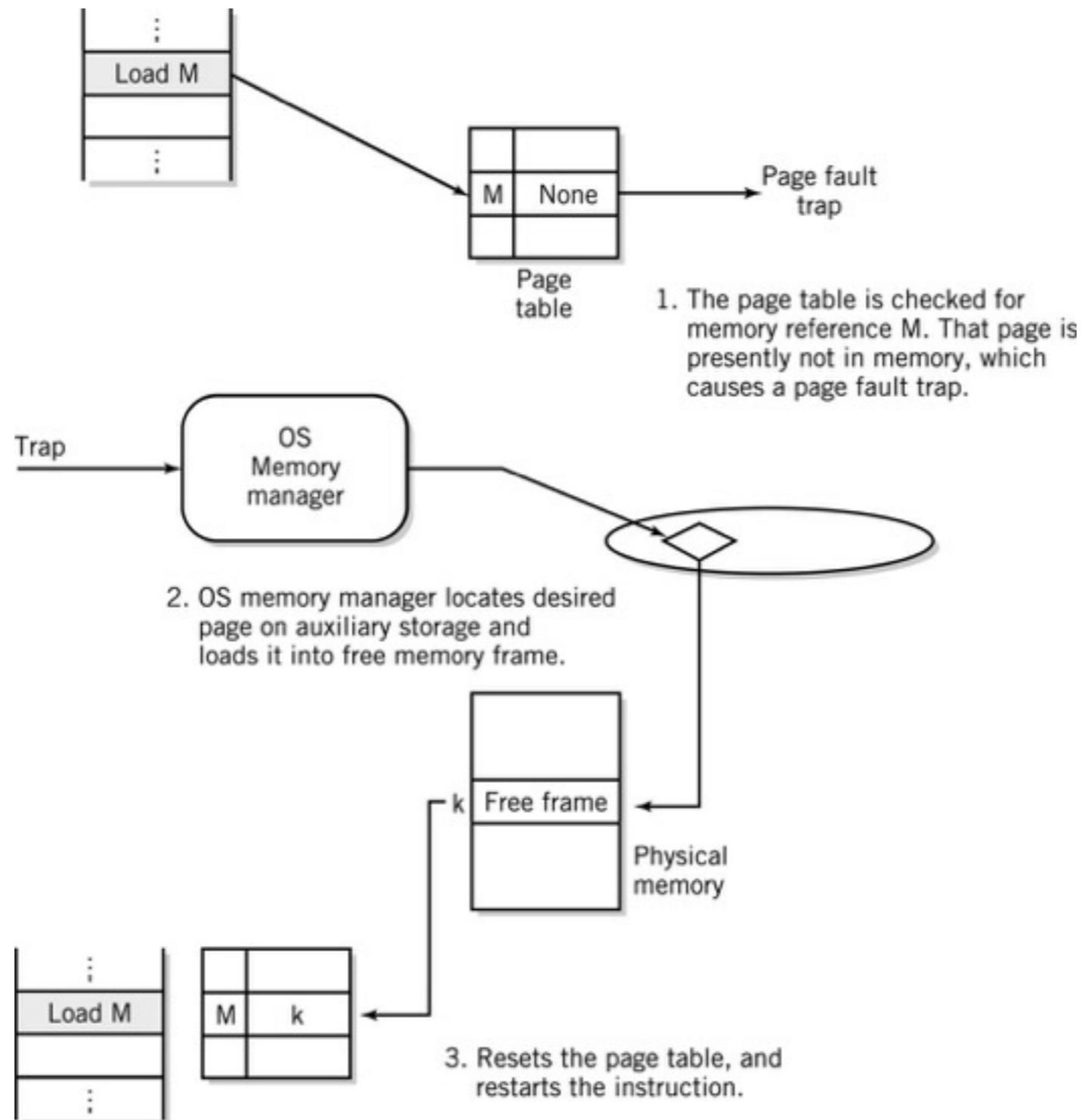
# Allocation et désallocation des pages

- Les pages allouées à un programme peuvent se retrouver séparées dans la mémoire. L'allocation de pages est très simple: il suffit de maintenir une liste des pages libres et de retirer une page libre pour l'allouer.
- Il est possible que plusieurs programmes utilisent une même page s'ils ont du code en commun.
- Désallouer des pages est simple: il suffit de mettre une page dans les pages libres.
- Qu'arrive-t-il si un programme a besoin d'une page qui n'existe pas?

# Fautes de page

- Une faute de page survient lorsque la page d'un programme qui est requise n'est pas en mémoire.
- S'il y a de la place dans la table des pages
  - la chercher sur le disque dur, la copier en mémoire, et mettre à jour la table des pages
- Sinon
  - il faut remplacer une autre page par la page requise. Plusieurs algorithmes permettent de déterminer quelle page sera remplacée. Par exemple, il est possible de remplacer la page qui a été inutilisée depuis le plus longtemps ou il est possible de remplacer la plus vieille page allouée.

# Fautes de page



# F.A.Q.

- Q. Où est la table des pages?
  - A. En mémoire
- Q. Combien d'accès mémoire doit-on faire pour accéder à des données?
  - A. 2: un pour accéder à la table des pages, un pour accéder aux données
- Q. Que faire pour éviter les accès mémoires doublés?
  - A. On utilise une "cache" spécialisée: le TLB!

# Traduction d'adresse, MMU et TLB

- Le MMU d'une mémoire paginée est plus complexe que celui d'une mémoire allouée de façon contiguë. En effet, pour tous les accès à la mémoire, il faut accéder à la table de pages afin de décoder l'adresse à lire. Comme la table de page est souvent volumineuse, elle est elle-même en mémoire: il faut faire deux lectures de la mémoire pour aller chercher une instruction, ce qui est très long et quasi-inadmissible.
- Le TLB ("Translation Lookaside Buffer"), est une table de registres très rapides à l'intérieur du CPU. Le TLB contient quelques entrées de la table de pages, les dernières utilisées. Il s'agit d'une forme de cache. Pour faire de la traduction d'adresse, le MMU regarde d'abord si la page à traduire est dans le TLB. Puisque le TLB contient les dernières entrées de la page de table utilisées, alors le MMU y trouve presque toujours la page à lire. Lorsqu'un hit se produit dans le TLB, le temps d'accès à la mémoire n'est pas augmenté par la pagination.
- Un TLB typique contient 8 à 4096 entrées de page de table. Lorsque la page est trouvée dans le TLB,  $\frac{1}{2}$  cycle à 1 cycle d'horloge est requis pour faire la translation d'adresse. Lorsque la page n'est pas dans le TLB, il faut faire deux accès mémoire... Le taux de succès dans le TLB est entre 99% et 99.99%!

# Table des pages inversée

- Pour plusieurs raisons, il peut être utile de savoir quelle page de quel processus utilise quelle page physique. Afin d'avoir cette information, il faut construire une page de table inversée.
- Une page de table inversée est une table pour laquelle l'indice de la table est un numéro de frame (page de mémoire) et le contenu d'un élément de la table est une page de processus.
- La taille d'une page de table inversée est fixe et directement proportionnelle à la taille de la mémoire. Elle est aussi inversement proportionnelle à la taille des pages.

# Illustration de la mémoire virtuelle

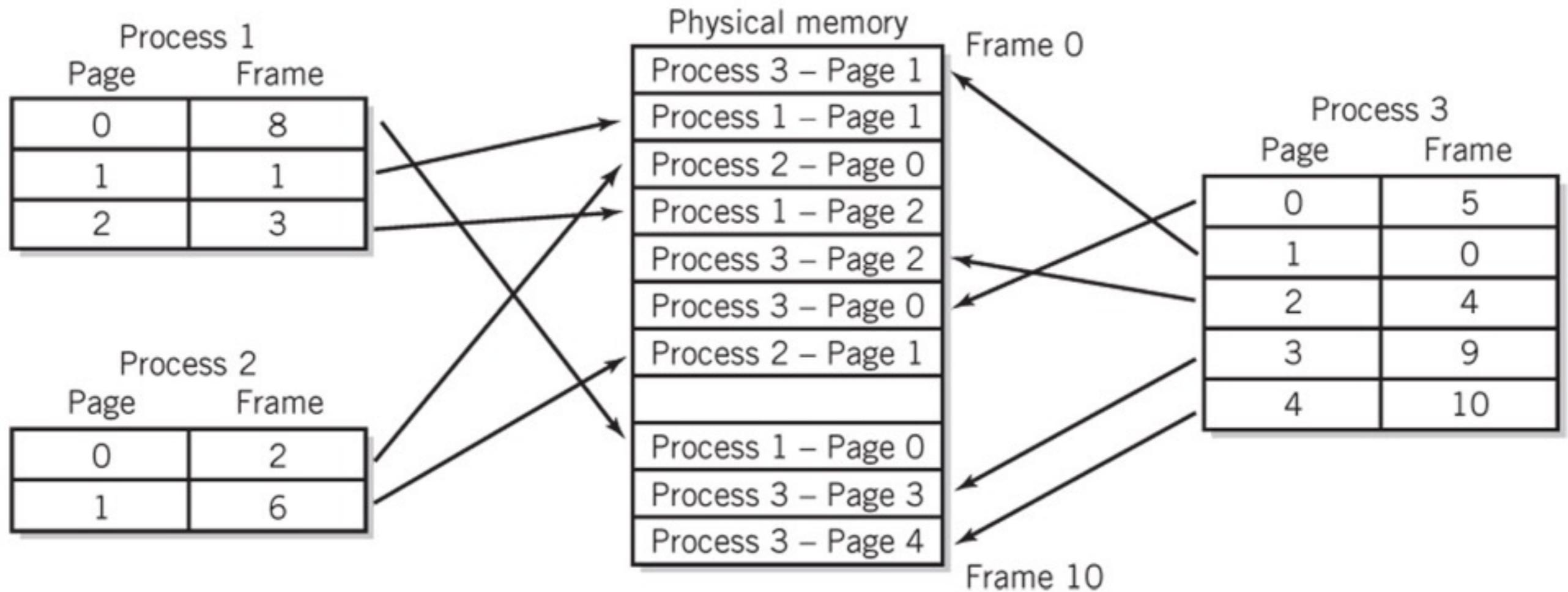


Table de pages inversées

# Mémoire virtuelle vs cache

- Cache
  - accélérer la vitesse d'accès mémoire
- Mémoire virtuelle
  - augmente la "quantité perçue" de mémoire disponible
  - indépendant de l'architecture

# Adresse d'un programme vs adresse de mémoire

- Lorsqu'un programme est écrit, des références à des adresses logiques ou variables/fonctions sont définis.
- Lors de la compilation, un fichier objet est créé. Le fichier objet contient le code et des tables pour les références.
- L'éditeur de lien utilise le code et les tables de plusieurs fichiers objets afin de construire un programme.
- Le programme est sur le disque dur. Lorsqu'il est chargé en mémoire à plusieurs endroits ou même à un seul, les adresses du programme ne correspondent pas nécessairement aux adresses de la mémoire: le MMU fera la translation entre l'adresse contenue dans le programme et l'adresse physique.
- L'adresse utilisée dans le code d'un programme est rarement l'adresse véritable d'une instruction ou d'une variable.

# Références

- Références
  - Irv Englander: chapitre 18

# Exercice 1: Exemple de calculs reliés aux tables de pages

- Supposons une mémoire de 16 Mo et un système d'exploitation supportant des pages de 4Ko pour des programmes ayant 64 Mo maximum. Supposons que 4 bits par page de programme sont utilisés pour donner des informations supplémentaires
- Supposons l'extrait de la table de page pour le programme X suivant:

#page virtuelle	#frame de la mémoire	Bits d'information
...	...	...
14h	B	En mémoire, modifiée
13h	4	En mémoire
12h	C	En mémoire, modifiée
11h	2	En mémoire
...	...	...

Q1: Quelle est la taille de la table de page?

- Q2: À quelle adresse de la mémoire retrouverons-nous l'instruction du programme à l'adresse virtuelle 0x143AB?

# Solution de l'exercice 1

- Q1

- La taille de la table = nombre d'entrée \* taille d'une entrée
- Le nombre d'entrée dans la table de page sera le nombre de page virtuelle: nombre d'entrée = taille de prog./taille des pages = 16k
- La taille d'une entrée est le nombre de bits qu'il faut pour d'écrire le numéro de page de mémoire + les 4 bits d'information. Le nombre de bits requis pour décrire le numéro de page de mémoire est  $\log_2(\text{nombre de page de mémoire})$ . Donc:
- Taille d'une entrée =  $\log_2(16\text{Mo}/4\text{ko})$  bits + 4 bits = 16 bits
- R1: La taille de la table =  $16\text{k} * 2\text{bytes} = 32\text{k}$

- Q2

- Pour trouver l'adresse physique, il faut trouver le numéro de page virtuelle et le remplacer par le numéro de page physique selon la table de page
- Le numéro de page virtuelle est l'adresse virtuelle divisée par la taille d'une page. Le reste de la division est la position dans la page (offset)
- $0\text{x}143\text{AB}/0\text{x}01000$  ( $4\text{k} = 0\text{x}01000$ ) = 14h reste 3ABh
- Selon la table de pages, la page 14h du programme est placé à la page B de la mémoire.
- R2: L'adresse virtuelle 0x143AB se retrouve à l'adresse physique 0xB3AB

## Exercice 2: accès mémoire paginée avec caches

- Décrivez les étapes nécessaires pour que le micro-processeur écrive une donnée en mémoire si:
  - l'adresse mémoire n'est dans aucune cache;
  - il y a deux niveaux de cache (L1 et L2);
  - le système utilise la stratégie "write-through";
  - la mémoire est paginée, mais la table des pages ne contient pas la page nécessaire.