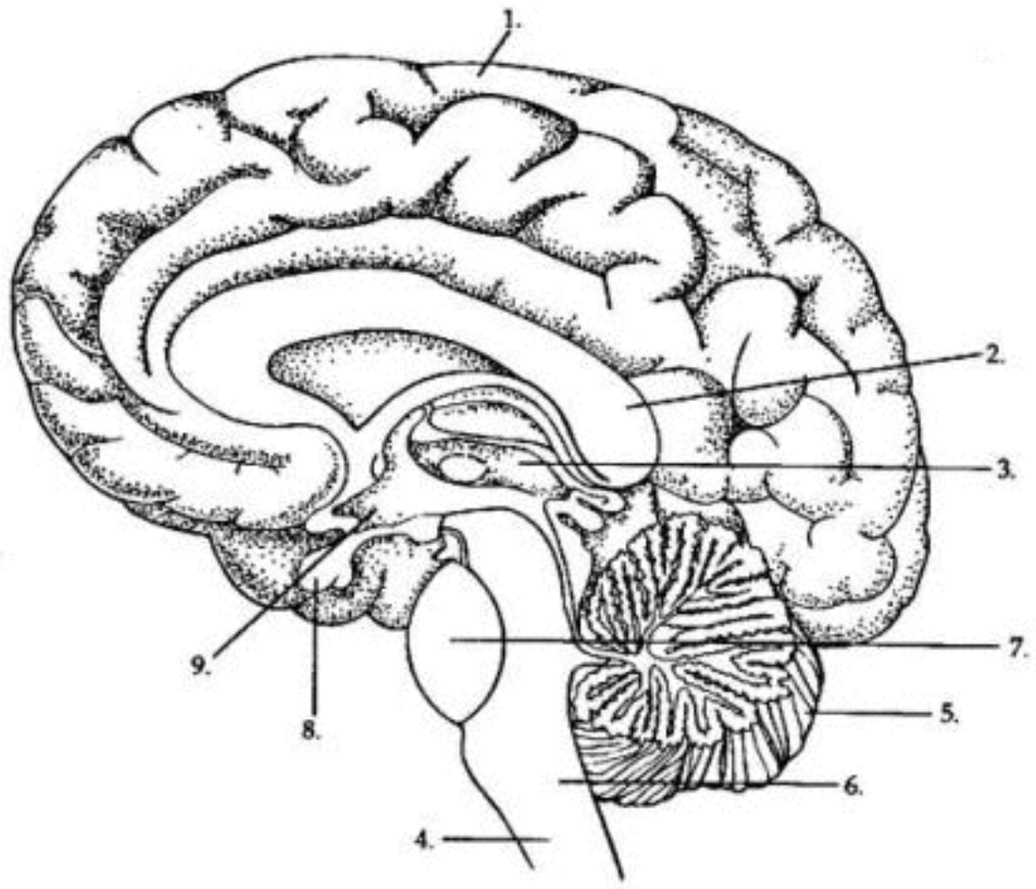
Gestion de la mémoire



GIF-1001 Ordinateurs: Structure et Applications, Hiver 2016

Jean-François Lalonde

Planification

- Vendredi: Vendredi Saint!
- TP5: 22 mars — 4 avril
  - 2 semaines, 1 atelier (pas d'atelier ce vendredi!)
- TP6: 5 avril — 18 avril
  - 2 semaines, 2 ateliers
- Examen final: 26 avril
  - 1 semaine complète après la remise du TP6

Modification?

- TP5: 22 mars — 11 avril
  - 3 semaines, 2 ateliers
- TP6: 5 avril — 22 avril (vendredi)
  - 2.5 semaines, 3 ateliers
  - 1 semaine de chevauchement avec le TP5
- Examen final: 26 avril
  - si remis à temps: 4 jours après la remise du TP6

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

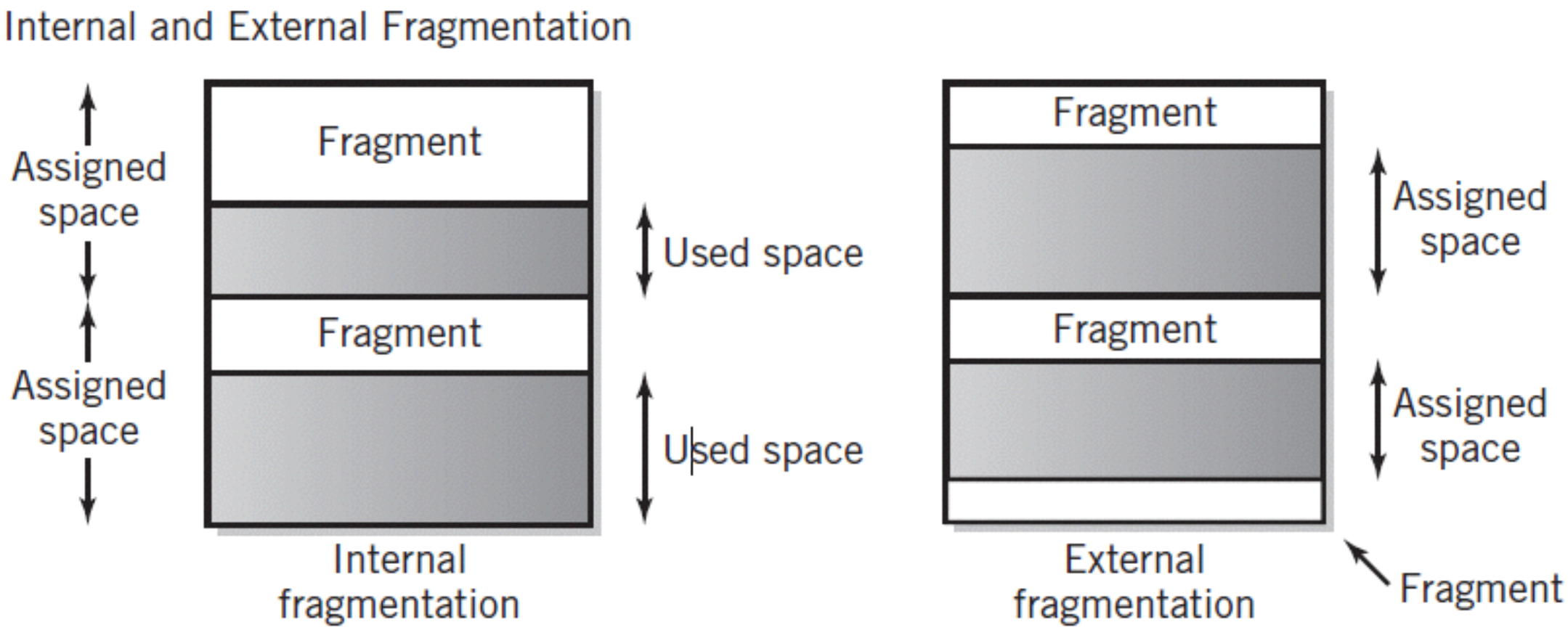
Allocation contiguë, partitions de taille variable ou fixe

- 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 alloué, 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



Englander, ch. 18

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.
  - **Première allocation (« First Fit »)**: Le programme est mis dans le premier bloc de mémoire suffisamment grand à partir du début de la mémoire.
    - Souvent utilisé malgré sa simplicité apparente!
  - **Prochaine allocation (« Next Fit »)**: Le programme est mis dans le premier bloc de mémoire suffisamment grand à partir du dernier bloc alloué.
    - Crée souvent un peu plus de fragmentation que « First Fit »
  - **Meilleure allocation (« Best Fit »)**: Le programme est mis dans le bloc de mémoire le plus petit dont la taille est suffisamment grande pour l'espace requis.
    - Semble meilleur, mais demande beaucoup de temps de calcul!
  - **Pire allocation (« Worse Fit »)**: Le programme est mis dans le bloc de mémoire le plus grand.

Démonstration

Exercice mémoire contigüe #1

- Effectuez les étapes suivantes pour un système ayant une mémoire de 16Mo:
  - Les processus suivants sont alloués en mémoire (dans l'ordre)
    - P1, 3Mo
    - P2, 5Mo
    - P3, 2Mo
    - P4, 2Mo
  - P1 et P3 se terminent
  - À quel endroit en mémoire le processus (P5, 2Mo) sera-t-il alloué si l'on emploie chacun des algorithmes d'allocation mémoire

Exercice mémoire contigüe #1

- Le processus P5 sera placé à des endroits différents, en fonction de l'algorithme utilisé:
  - Première allocation: 0—2
  - Meilleure allocation: 8—10
  - Prochaine (ou pire) allocation: 12—14

Exercice mémoire contigüe #2

- Un système possédant une mémoire de 16Mo doit allouer les processus suivants:

Processus	Taille	Temps de création	Durée
P1	3Mo	0	3
P2	2Mo	1	3
P3	4Mo	2	3
P4	2Mo	3	3
P5	3Mo	4	3

- Indiquez le contenu de la mémoire après l'allocation du processus P5

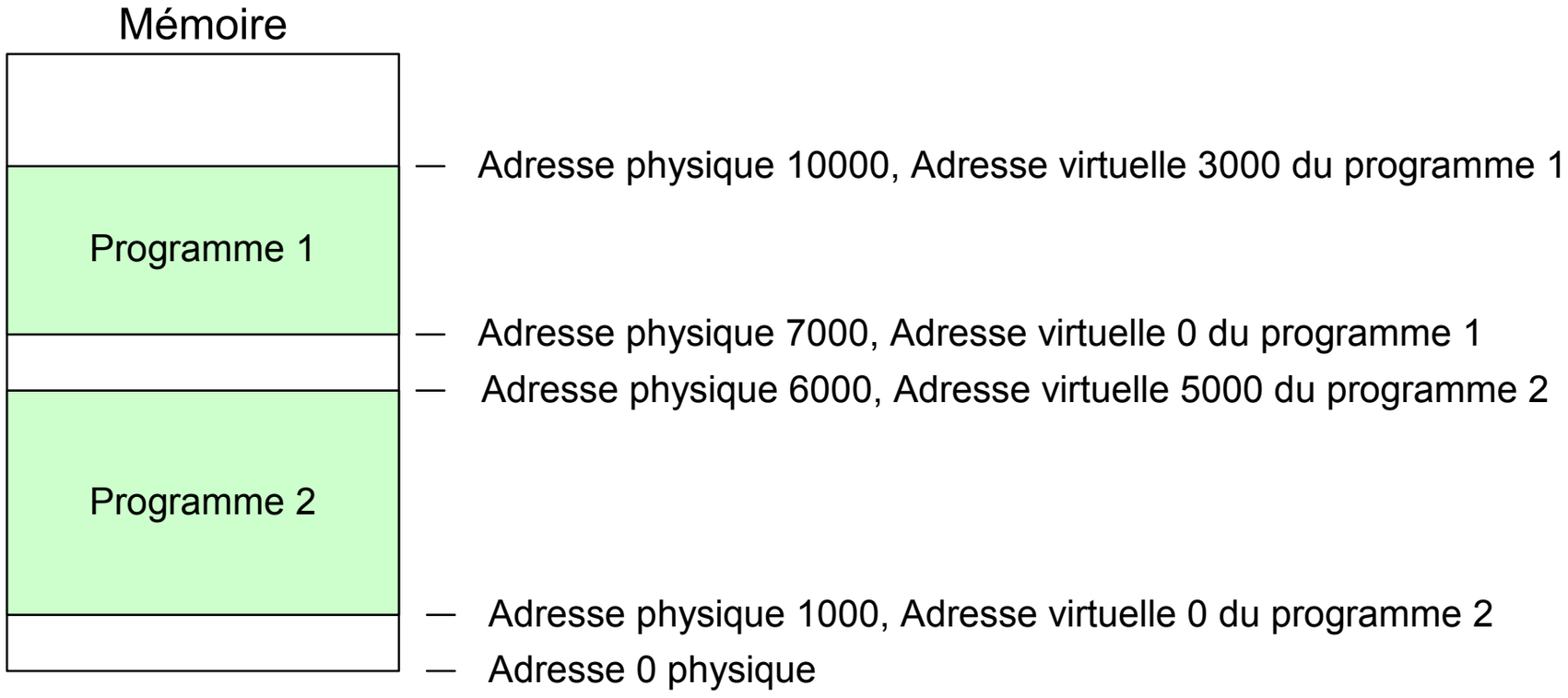
Exercice mémoire contigüe #2

Processus	Taille	Temps de création	Durée
P1	3Mo	0	3
P2	2Mo	1	3
P3	4Mo	2	3
P4	2Mo	3	3
P5	3Mo	4	3

Utilisez le simulateur avec le fichier « allocSimulation3.xml »
pour valider votre réponse!

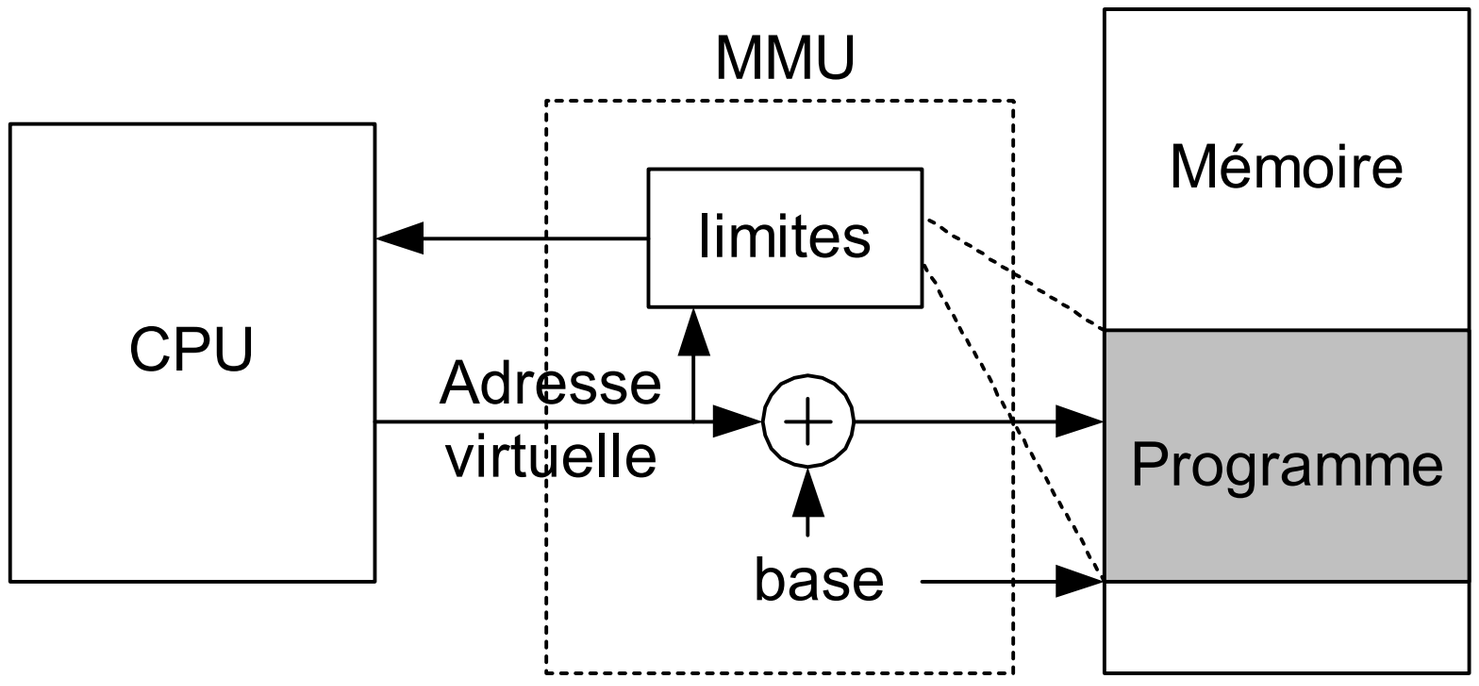
Adresse virtuelle et adresse physique

- 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'additionner l'adresse de base du programme (sa première adresse dans la mémoire physique) à son adresse dans le programme (adresse virtuelle).



Exercice mémoire virtuelle #1

- Dans un système en allocation contigüe avec partitions à taille variable, un processus P1 occupe les adresses mémoire 0x2000 à 0x4FFF.
    - Aux « yeux » de P1, quelle est sa plage d'adresses (virtuelles) disponible?
    - Quelle est la taille totale de mémoire disponible?
    - Quelle est l'adresse physique correspondant aux adresses virtuelles:
        - 0x0010?
        - 0x1234?

Exercice mémoire virtuelle #1

- Dans un système en allocation contigüe avec partitions à taille variable, un processus P1 occupe les adresses mémoire 0x2000 à 0x4FFF.
  - Aux « yeux » de P1, quelle est sa plage d'adresses (virtuelles) disponible?
    - 0x0000 à 0x2FFF
  - Quelle est la taille totale de mémoire disponible?
    - 12Ko
  - Quelle est l'adresse physique correspondant aux adresses virtuelles:
    - 0x0010?
      - 0x2010
    - 0x1234?
      - 0x3234

Récapitulation

1. Un nouveau programme est copié dans un emplacement disponible en mémoire, de façon contigüe.

2. On peut créer des partitions de taille:

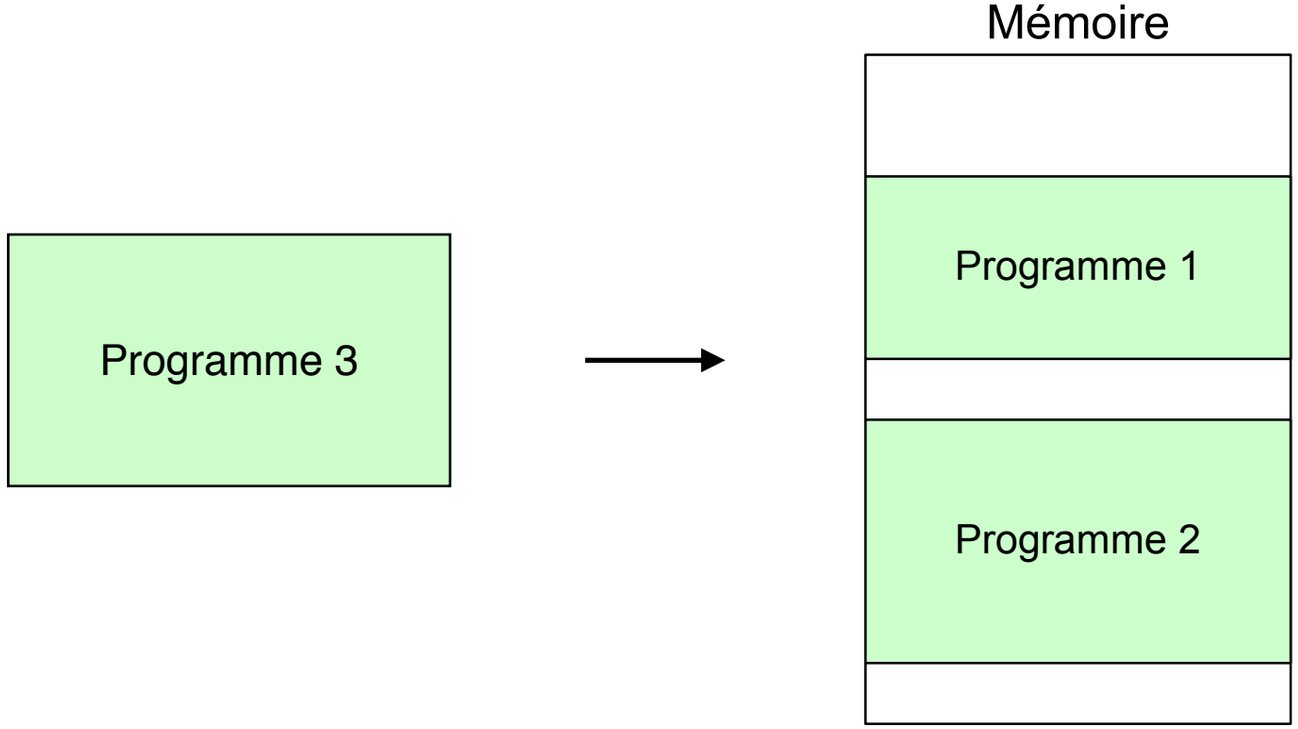
    - **fixe**: la première partition disponible est choisie quand un nouveau processus doit être alloué
    - **variable**: on doit déterminer où créer la partition, nécessite le choix d'un algorithme d'allocation mémoire plus compliqué

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 petite parties de taille fixes: des pages. Chaque partie se retrouve à un endroit différent de la mémoire.

- Il faut maintenir un s des programmes et les pages de la

    - La table de corre able”).

    - Il peut y avoir une s programmes.

- Le principe de loca r d’un programme sont probablemen lle 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.

RAJOUTER:

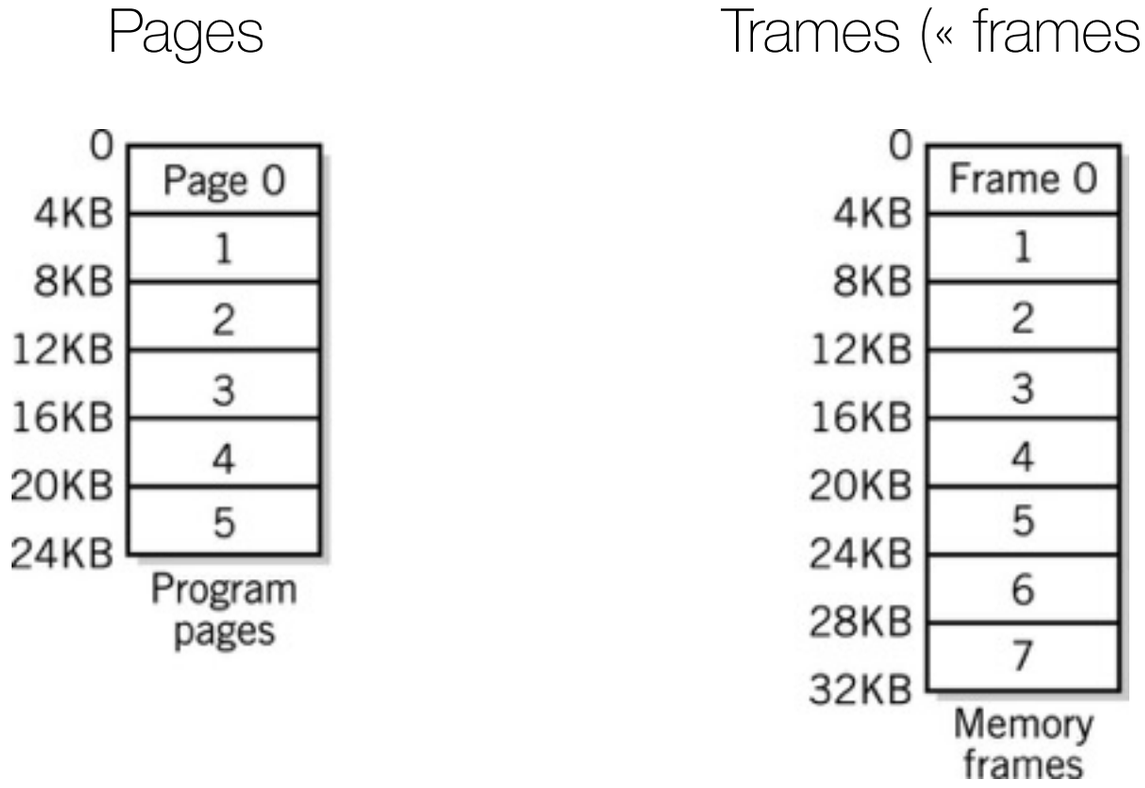
https://en.wikipedia.org/wiki/Memory_management_unit#/media/File:MMU_principle_updated.png

Mémoire paginée, étape #1

- Diviser la mémoire **virtuelle** en « pages » de taille fixe
  - par exemple: 4Ko
- Diviser la mémoire **physique** en « trames » de *la même taille* que les pages
  - dans notre exemple: 4Ko!

Pages et trames

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



Truc pour s'en rappeler : **P**ages = **P**rogramme

Englander, ch. 18

Mémoire paginée, étape #2

- On divise une adresse en deux:
  - le numéro de page (MSB)
  - la position dans la page (LSB)
- Dans notre exemple, les pages ont 4Ko. Combien de bits avons-nous besoin pour encoder la position de chaque octet dans la page?
  - il y a 4K octets dans une page, donc $4K = 4 \times 2^{10}$
  - $\log_2(4 \times 2^{10}) = \log_2(2^2 \times 2^{10}) = \log_2(2^{12}) = 12$ bits

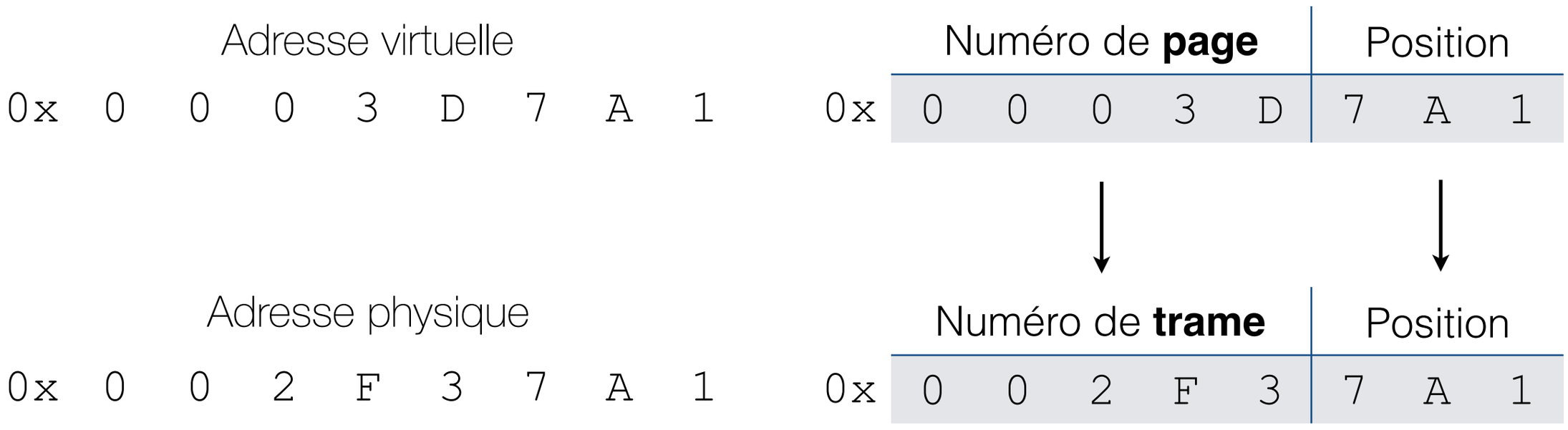
Quelle position dans quelle page réfère cette adresse?

0x 0 0 0 3 D 7 A 1

	Numéro de page					Position		
0x	0	0	0	3	D	7	A	1

Mémoire paginée, étape #3

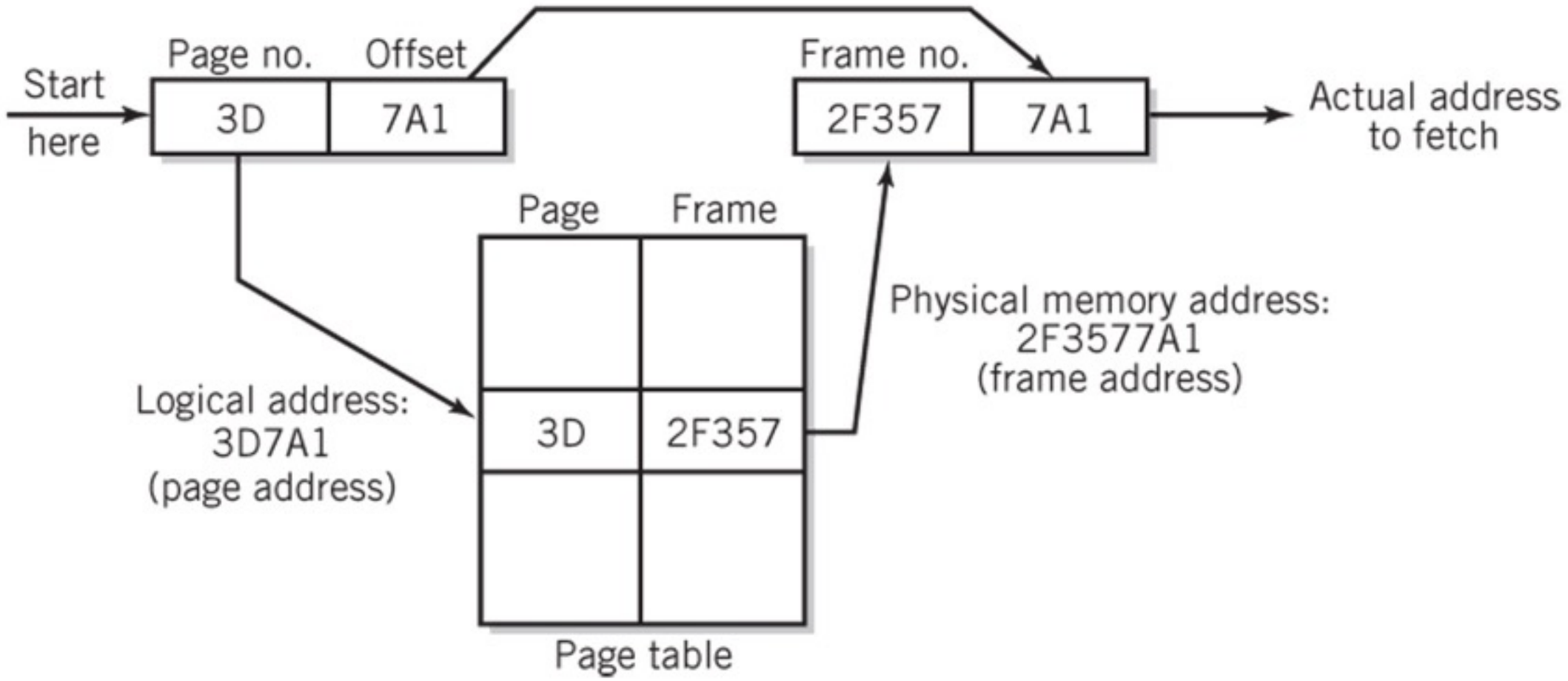
- Si l'on vous donne l'endroit où la page est stockée en mémoire.
  - Par exemple: la page 3D est stockée dans la trame 2F3
- Comment faire pour convertir une adresse virtuelle en une adresse physique?



Mémoire paginée, étape #4

- Comment fait-on pour savoir
  - l'endroit où une page est stockée?
  - la correspondance entre une page et une trame?
- On utilise la table des pages!

Table des pages



Englander, ch. 18

Taille de la table des pages

- La table des pages est définie par:
  - le nombre de lignes: il y a une ligne par page, donc c'est équivalent au **nombre de pages**
  - l'information stockée dans chaque ligne: le numéro de trame correspondant à chaque page.
    - notez qu'il n'est pas nécessaire de stocker le numéro de page, nous n'avons qu'à lire la ligne correspondant à la page.
      - par exemple, la page 0x2D (45) est à la ligne 45 dans la table
    - combien de bits sont nécessaires pour représenter le numéro de trame?
      - cela dépend du nombre total de trames dans le système!

Taille de la table des pages

- Un système possède les caractéristiques suivantes:
  - une mémoire *physique* de 32Mo
  - une mémoire *virtuelle* de 64Mo
  - la taille d'une page est de 4Ko
- Quelle est la taille de la table des pages?

Taille de la table des pages

- Un système possède les caractéristiques suivantes:
  - une mémoire *physique* de 32Mo
  - une mémoire *virtuelle* de 64Mo
  - la taille d'une page est de 4Ko
- Quelle est la taille de la table des pages?

Rappel
pages = mémoire *virtuelle*
trames = mémoire *physique*

  - Déterminer le nombre de *pages*

$$64\text{Mo} / 4\text{Ko} = 64 \times 2^{20} / 4 \times 2^{10} = 2^{26} / 2^{12} = 2^{14} \text{ pages}$$

  - Déterminer le nombre de *trames*

$$32\text{Mo} / 4\text{Ko} = 32 \times 2^{20} / 4 \times 2^{10} = 2^{25} / 2^{12} = 2^{13} \text{ trames}$$

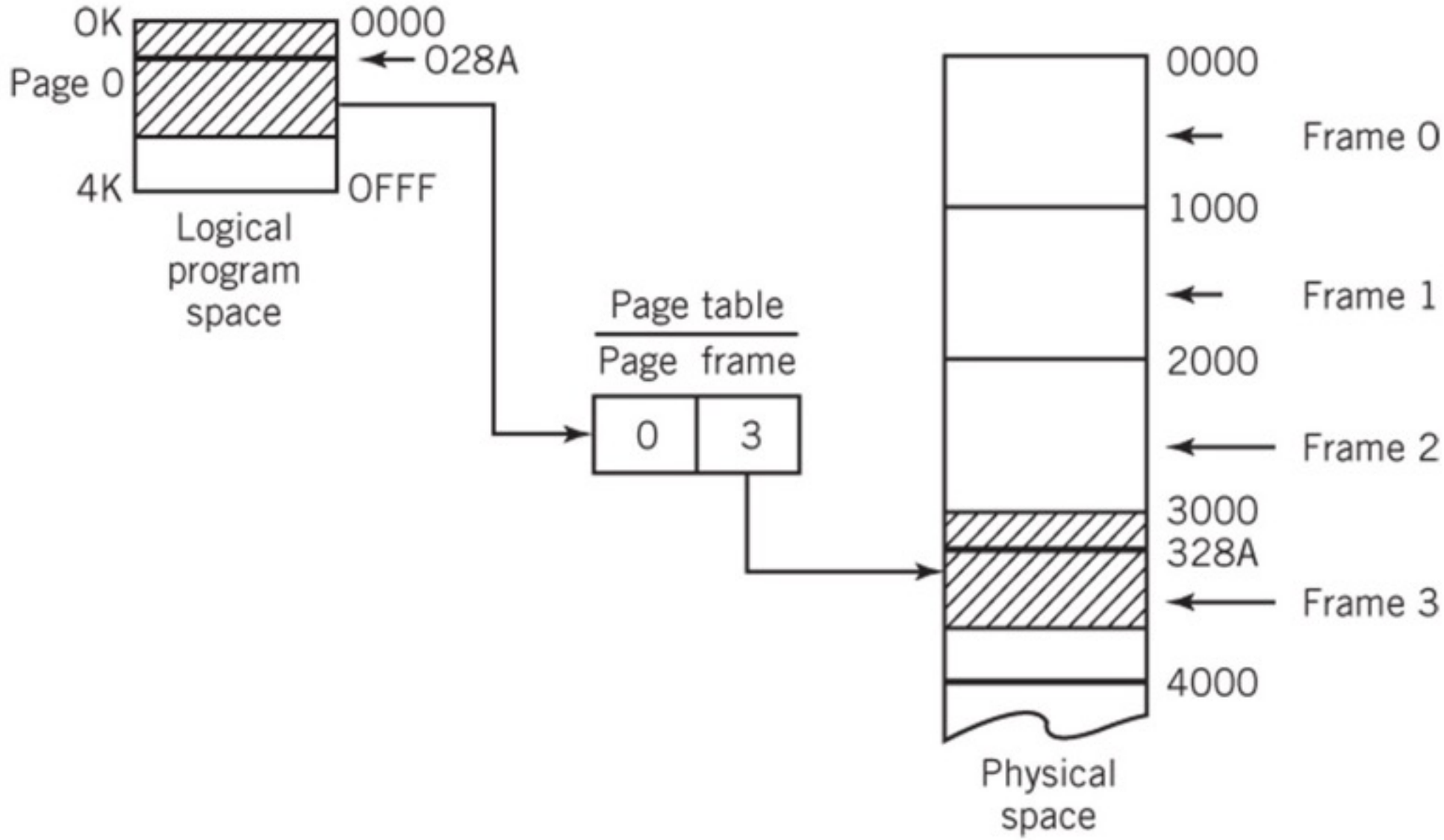
  - Déterminer le nombre de bits nécessaires pour stocker le # de trame

$$\log_2(2^{13}) = 13 \text{ bits}$$

  - Calculer la taille totale (#pages x #bits)

$$2^{14} \times 13 = 2^4 \times 13 \times 2^{10} = 208 \times 2^{10} = 208\text{Kbits} = 26\text{Ko}$$

Traduction d'adresses



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

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

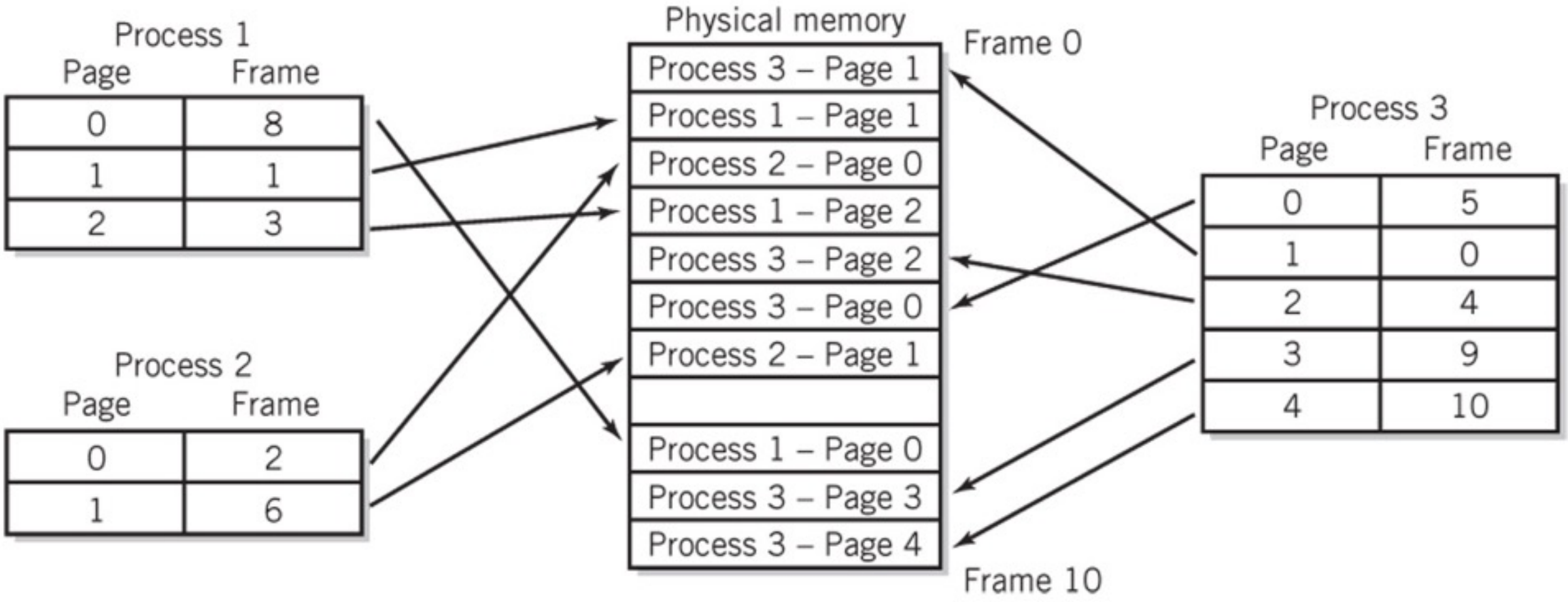


Table de pages inversées

Englander, ch. 18

Table des pages et table des pages inversée

- Caractéristiques du système:
  - Pages de 4Ko
  - Mémoire physique (RAM) de 16Ko
  - Mémoire virtuelle de 32Ko
- Effectuez la séquence d'opérations suivantes en mettant à jour les tables à droite, et indiquez les adresses physiques correspondantes
  - Charger valeur (e.g. LDR) à l'adresse 0x0A38
  - Écrire une valeur (e.g. STR) à l'adresse 0x3B97
  - Charger la valeur (e.g. LDR) à l'adresse 0x2928
  - Brancher (e.g. B) à l'adresse 0x7BF0

Table des pages

# page	# trame
0	1
1	2
2	
3	0
4	
5	
6	
7	

Table des pages inversée

# trame	# page
0	page 3
1	page 0
2	page 1
3	

Table des pages et table des pages inversée

- Caractéristiques du système:
  - Pages de 4Ko
  - Mémoire physique (RAM) de 16Ko
  - Mémoire virtuelle de 32Ko
- Effectuez la séquence d'opérations suivantes en mettant à jour les tables à droite, et indiquez les adresses physiques correspondantes
  - Charger valeur (e.g. LDR) à l'adresse 0x0A38
    - l'adresse physique est 0x1A38
  - Écrire une valeur (e.g. STR) à l'adresse 0x3B97
  - Charger la valeur (e.g. LDR) à l'adresse 0x2928
  - Brancher (e.g. B) à l'adresse 0x7BF0

Table des pages

# page	# trame
0	**1**
1	2
2	
3	0
4	
5	
6	
7	

Table des pages inversée

# trame	# page
0	page 3
1	page 0
2	page 1
3	

Table des pages et table des pages inversée

- Caractéristiques du système:
  - Pages de 4Ko
  - Mémoire physique (RAM) de 16Ko
  - Mémoire virtuelle de 32Ko
- Effectuez la séquence d'opérations suivantes en mettant à jour les tables à droite, et indiquez les adresses physiques correspondantes
  - Charger valeur (e.g. LDR) à l'adresse 0x0A38
  - Écrire une valeur (e.g. STR) à l'adresse 0x3B97
    - l'adresse physique est 0x0B97
  - Charger la valeur (e.g. LDR) à l'adresse 0x2928
  - Brancher (e.g. B) à l'adresse 0x7BF0

Table des pages

# page	# trame
0	1
1	2
2	
3	**0**
4	
5	
6	
7	

Table des pages inversée

# trame	# page
0	page 3
1	page 0
2	page 1
3	

Table des pages et table des pages inversée

- Caractéristiques du système:
  - Pages de 4Ko
  - Mémoire physique (RAM) de 16Ko
  - Mémoire virtuelle de 32Ko
- Effectuez la séquence d'opérations suivantes en mettant à jour les tables à droite, et indiquez les adresses physiques correspondantes
  - Charger valeur (e.g. LDR) à l'adresse 0x0A38
  - Écrire une valeur (e.g. STR) à l'adresse 0x3B97
  - Charger la valeur (e.g. LDR) à l'adresse 0x2928
    - Faute de page!
  - Brancher (e.g. B) à l'adresse 0x7BF0

Table des pages

# page	# trame
0	1
1	2
2	**?**
3	0
4	
5	
6	
7	

Table des pages inversée

# trame	# page
0	page 3
1	page 0
2	page 1
3	

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.

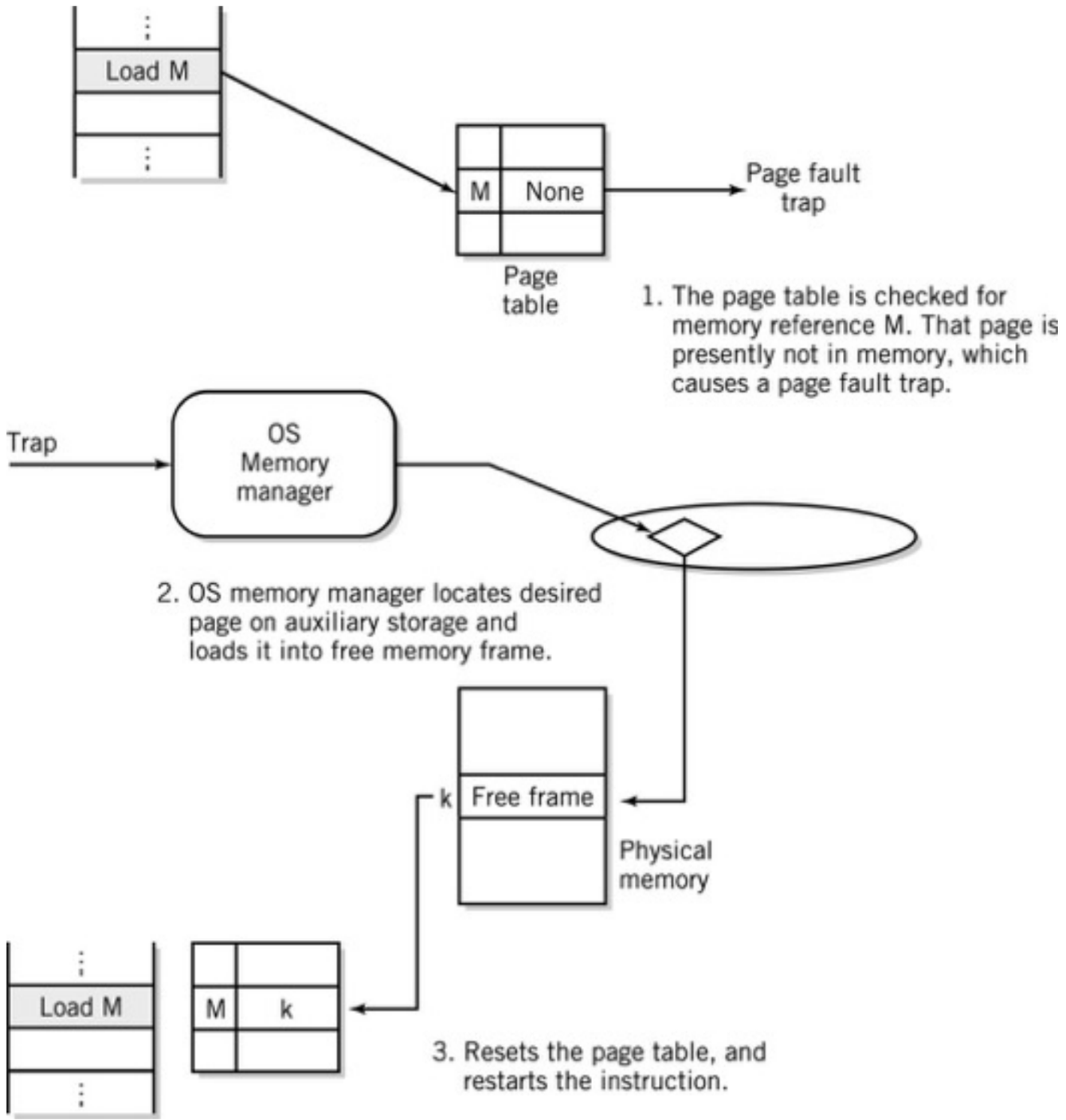
- Désallouer des pages est simple: il suffit de mettre une page dans les pages libres.

- 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



Englander, ch. 18

Table des pages et table des pages inversée

- Caractéristiques du système:
  - Pages de 4Ko
  - Mémoire physique (RAM) de 16Ko
  - Mémoire virtuelle de 32Ko
- Effectuez la séquence d'opérations suivantes en mettant à jour les tables à droite, et indiquez les adresses physiques correspondantes
  - Charger valeur (e.g. LDR) à l'adresse 0x0A38
  - Écrire une valeur (e.g. STR) à l'adresse 0x3B97
  - Charger la valeur (e.g. LDR) à l'adresse 0x2928
    - L'adresse physique est 0x3928
  - Brancher (e.g. B) à l'adresse 0x7BF0

Table des pages

# page	# trame
0	1
1	2
2	**3**
3	0
4	
5	
6	
7	

Table des pages inversée

# trame	# page
0	page 3
1	page 0
2	page 1
3	**page 2**

Table des pages et table des pages inversée

- Caractéristiques du système:
  - Pages de 4Ko
  - Mémoire physique (RAM) de 16Ko
  - Mémoire virtuelle de 32Ko
- Effectuez la séquence d'opérations suivantes en mettant à jour les tables à droite, et indiquez les adresses physiques correspondantes
  - Charger valeur (e.g. LDR) à l'adresse 0x0A38
  - Écrire une valeur (e.g. STR) à l'adresse 0x3B97
  - Charger la valeur (e.g. LDR) à l'adresse 0x2928
  - Brancher (e.g. B) à l'adresse 0x7BF0
    - Faute de page! Quelle trame utiliser? Que faire avec le contenu de cette trame?

Table des pages

# page	# trame
0	1
1	2
2	3
3	0
4	
5	
6	
7	**?**

Table des pages inversée

# trame	# page
0	page 3
1	page 0
2	page 1
3	page 2

Table des pages et table des pages inversée

- Caractéristiques du système:
  - Pages de 4Ko
  - Mémoire physique (RAM) de 16Ko
  - Mémoire virtuelle de 32Ko
- Effectuez la séquence d'opérations suivantes en mettant à jour les tables à droite, et indiquez les adresses physiques correspondantes
  - Charger valeur (e.g. LDR) à l'adresse 0x0A38
  - Écrire une valeur (e.g. STR) à l'adresse 0x3B97
  - Charger la valeur (e.g. LDR) à l'adresse 0x2928
  - Brancher (e.g. B) à l'adresse 0x7BF0
    - L'adresse physique est 0x2BF0

Table des pages

# page	# trame
0	1
1	~~**2**~~
2	3
3	0
4	
5	
6	
7	**2**

Table des pages inversée

# trame	# page
0	page 3
1	page 0
2	**page 7**
3	page 2

Table des pages et table des pages inversée

- Tables finales:

Table des pages

# page	# trame
0	1
1	
2	3
3	0
4	
5	
6	
7	2

Table des pages inversée

# trame	# page
0	page 3
1	page 0
2	page 7
3	page 2

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.

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
…	…	…
0x14	B	En mémoire, modifiée
0x13	4	En mémoire
0x12	C	En mémoire, modifiée
0x11	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 log2(nombre de page de mémoire). Donc:
  - Taille d'une entrée = log2(16Mo/4ko) bits + 4 bits = 16 bits
    - R1: La taille de la table = 16k* 2bytes = 32k
- 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)
  - 0x143AB/0x01000 (4k = 0x01000) = 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