Module 8 – La mémoire virtuelle

Chapitre 9 (Silberchatz)

Mémoire Virtuelle

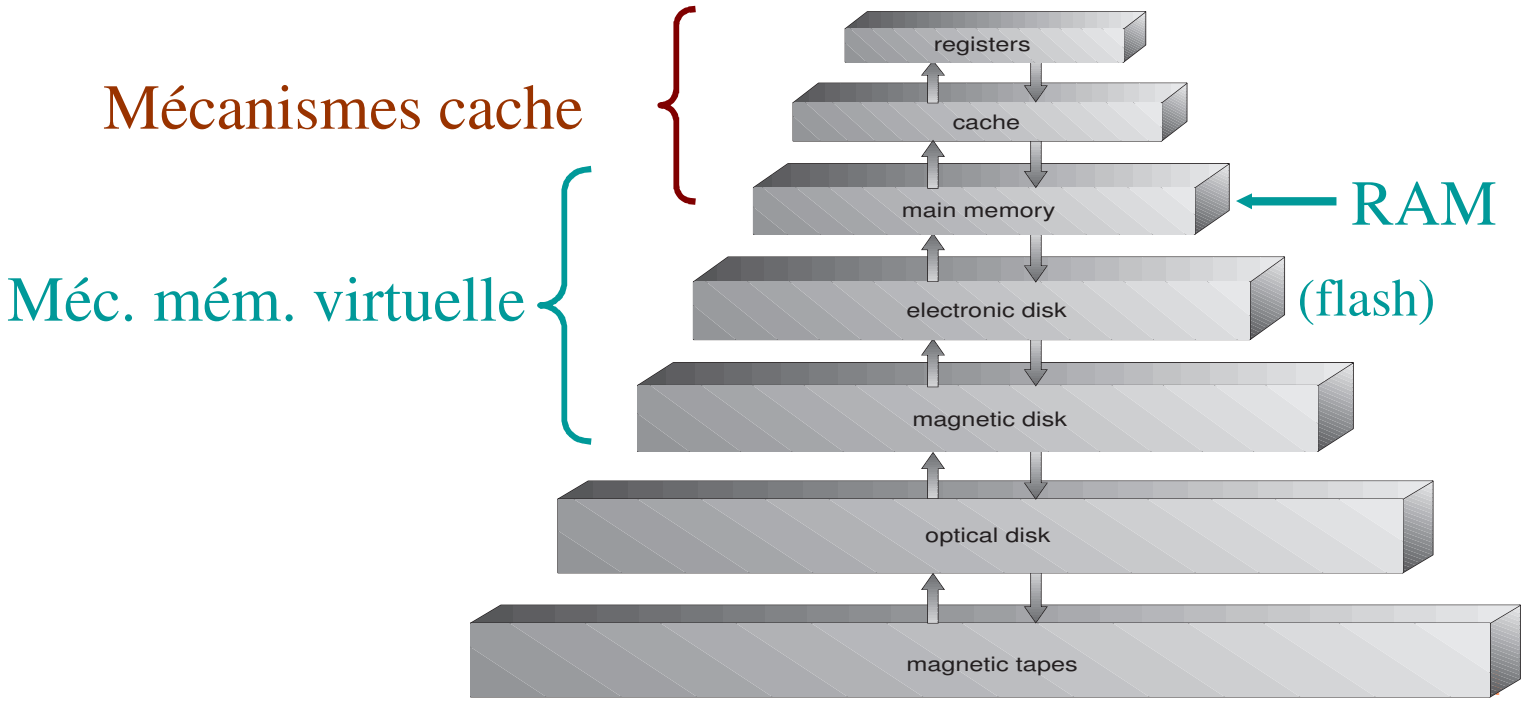
- Pagination sur demande
- Problèmes de performance
- Remplacement de pages: algorithmes
- Allocation de cadres de mémoire
- Ensemble de travail

Concepts importants du Module 8

- **Localité des références**
- **Mémoire virtuelle implémentée par va-et-vient des pages, mécanismes, défauts de pages**
- **Adresses physiques et adresses logiques**
- **Temps moyen d'accès à la mémoire**
  - Récriture ou non de pages sur mém secondaire
- **Algorithmes de remplacement pages:**
  - OPT, LRU, FIFO, Horloge
  - Fonctionnement, comparaison
- **Écroulement, causes**
- **Ensemble de travail (working set)**
- **Relation entre la mémoire allouée à un proc et le nombre d'interruptions**
- **Relation entre la dimension de pages et le nombre d'interruptions**
- **Prépagination, post-nettoyage**
- **Effets de l'organisation d'un programme sur l'efficacité de la pagination**

La mémoire virtuelle est une application du concept de hiérarchie de mémoire

- **C'est intéressant de savoir que des concepts très semblables s'appliquent aux mécanismes de la mémoire cache**
  - Cependant dans ce cas les mécanismes sont surtout de matériel



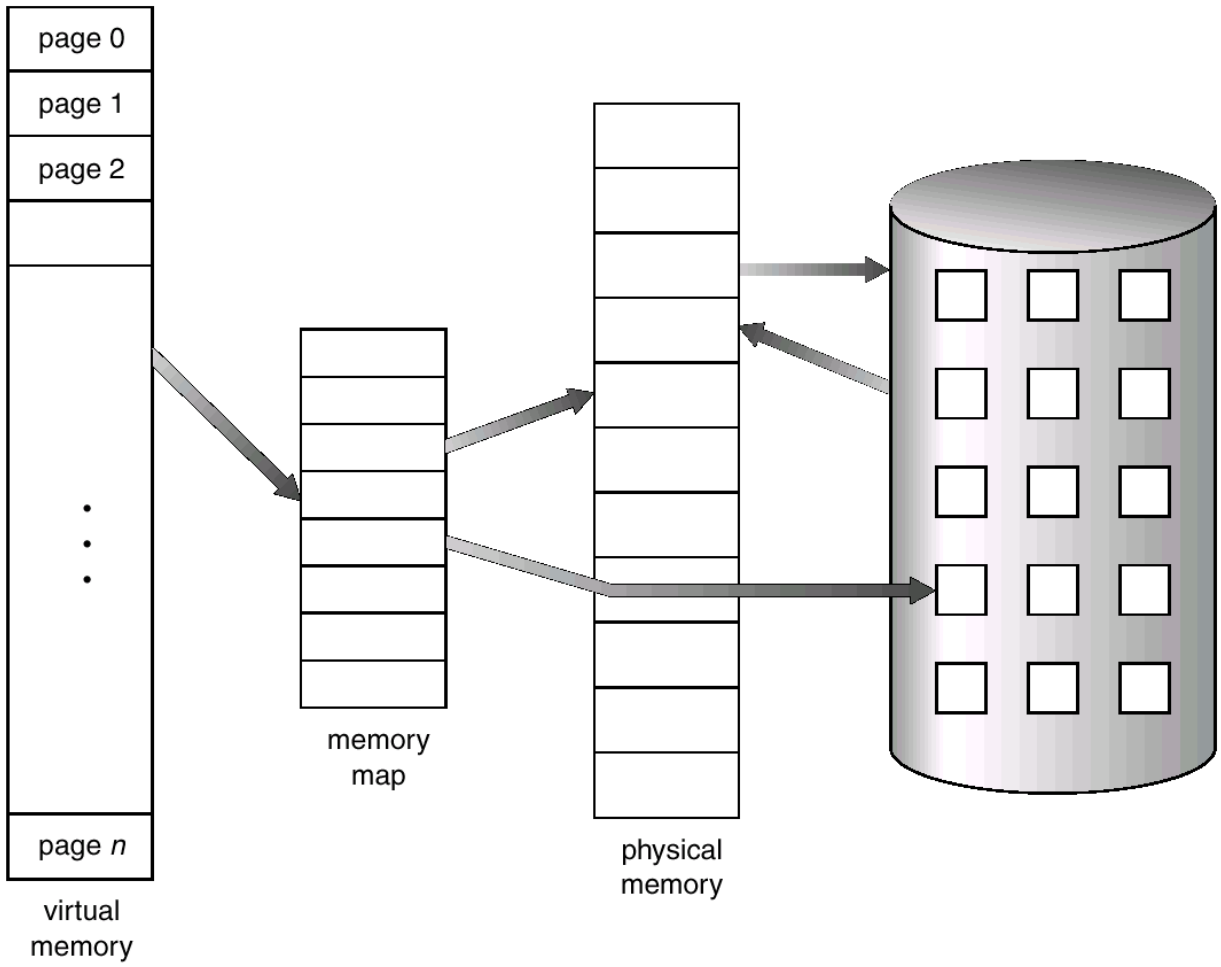
La mémoire virtuelle

- **Lorsqu'un programme est en exécution, il n'est pas nécessaire qu'il soit entièrement dans la mémoire principale!**
- **Seules les parties qui sont en exécution ont besoin d'être dans la mémoire centrale**
- **Les autres parties peuvent être dans la mémoire secondaire (ex. disque), prêtes à être amenées en mémoire centrale sur demande**
  - Mécanisme de va-et-vient ou swapping
- **Ceci rend possible l'exécution de programmes beaucoup plus grands que la mémoire physique**
  - Réalisant ainsi une mémoire virtuelle qui est plus grande que la mémoire physique

De la pagination et segmentation à la mémoire virtuelle

- **Un processus est constitué de morceaux (pages ou segments) ne nécessitant pas d'occuper une région contiguë de la mémoire principale**
- **Références à la mémoire sont converties en adresses physiques au moment de l'exécution**
  - Un processus peut être déplacé à différentes régions de la mémoire, incluant la mémoire secondaire!
- **Donc: tous les morceaux 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 un morceau se trouvant en mémoire principale
- **La somme des mémoires logiques des processus en exécution peut donc excéder la mémoire physique disponible**
  - Le concept de base de la mémoire virtuelle
- **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

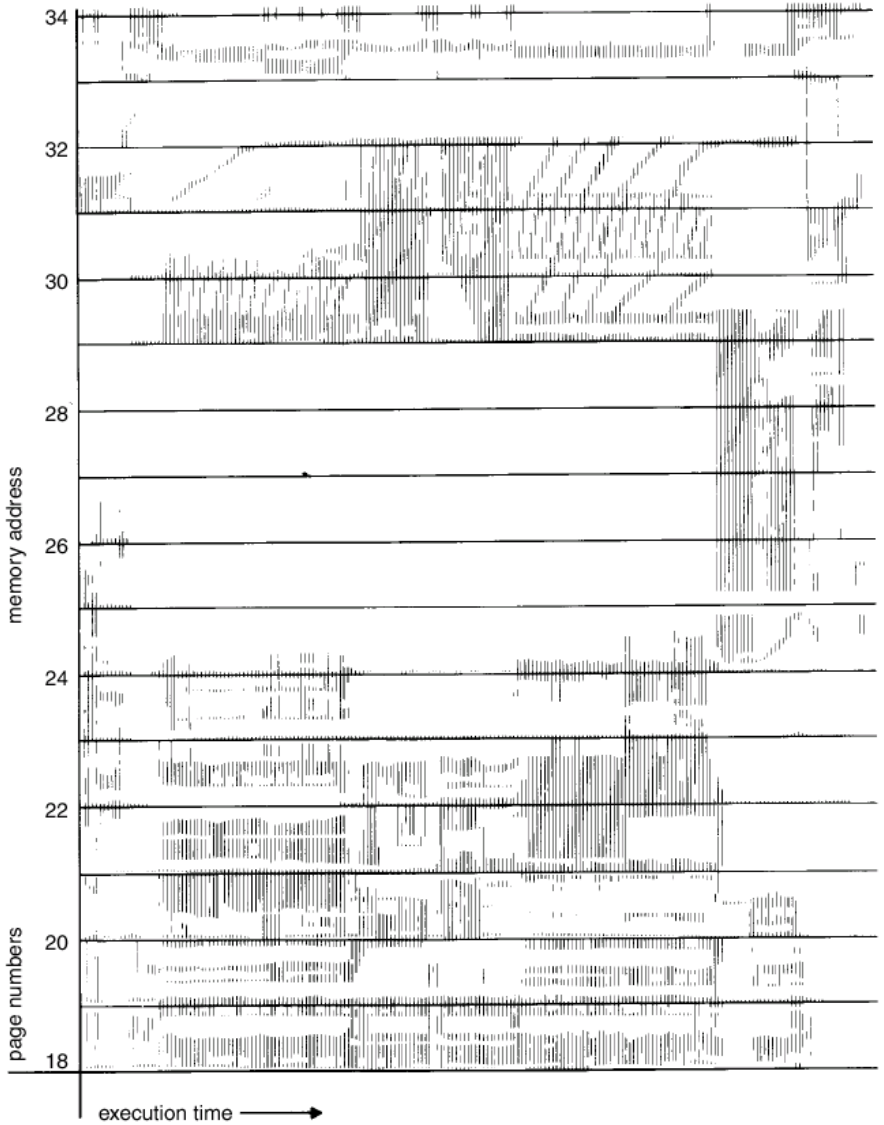
Mémoire virtuelle:
résultat d'un mécanisme qui combine la mémoire principale et les mémoires secondaires



Localité et mémoire virtuelle

- Principe de **localité des références**: les références à la mémoire dans un processus tendent à se regrouper
- Donc: seule quelques pièces (pages ou segments) d'un processus seront utilisées durant une petite période de temps
- Il y a une bonne chance de "deviner" quelles seront les pièces demandées dans un avenir rapproché

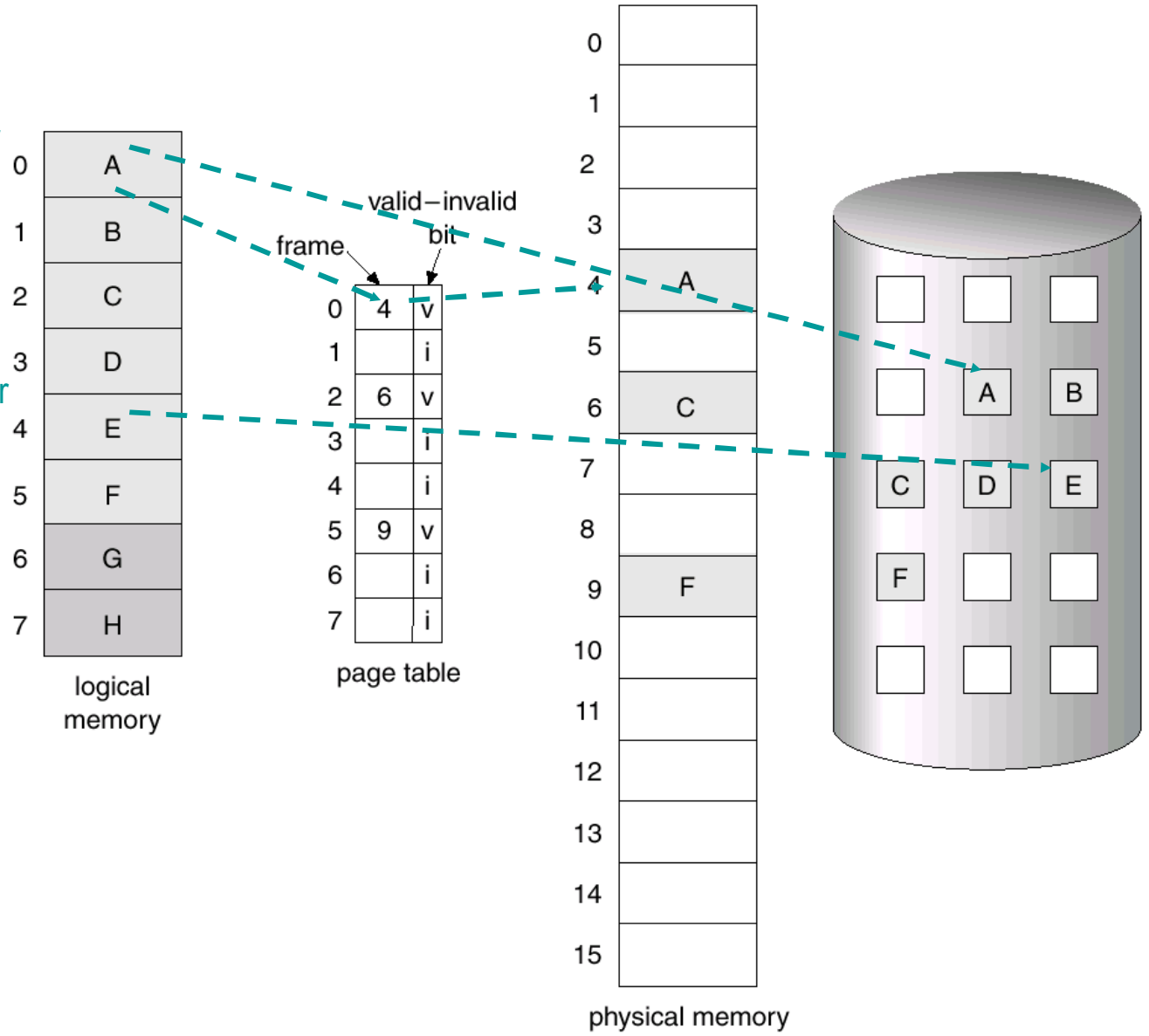
Visualisation de la localité des références



Pages en RAM ou sur disque

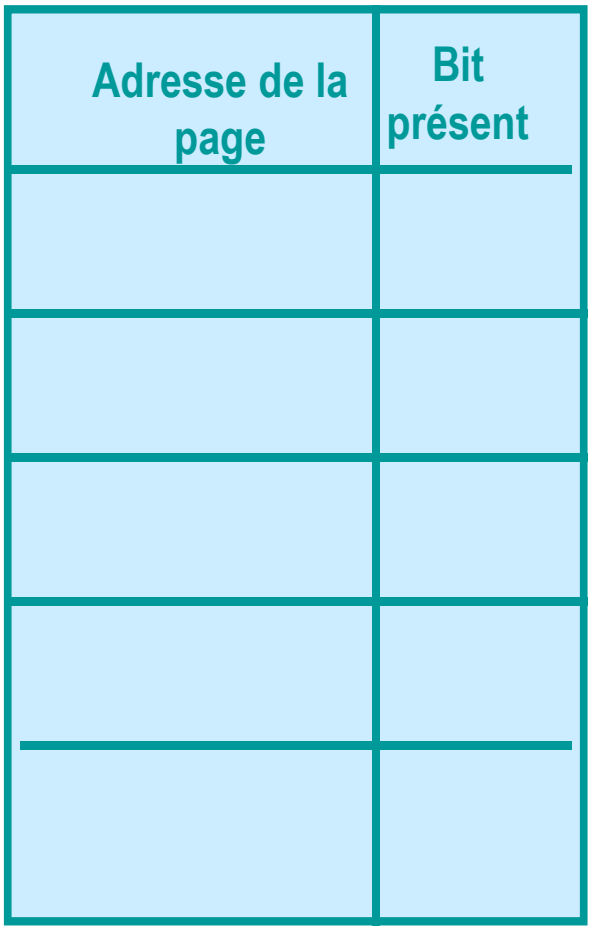
Page A en RAM **et** sur disque

Page E **seulement** sur disque



Nouveau format du tableau des pages (la même idée peut être appliquée aux tableaux de segments)

Si la page est en mém. princ., c'est une adresse de mémoire principale sinon c'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

Avantages du chargement partiel

- **Plus de processus peuvent être maintenus en exécution en mémoire**
  - Puisque seules quelques pièces sont chargées pour 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, meilleure 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 ensemble de processus lorsque leur 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 **> >** espace d 'adressage physique

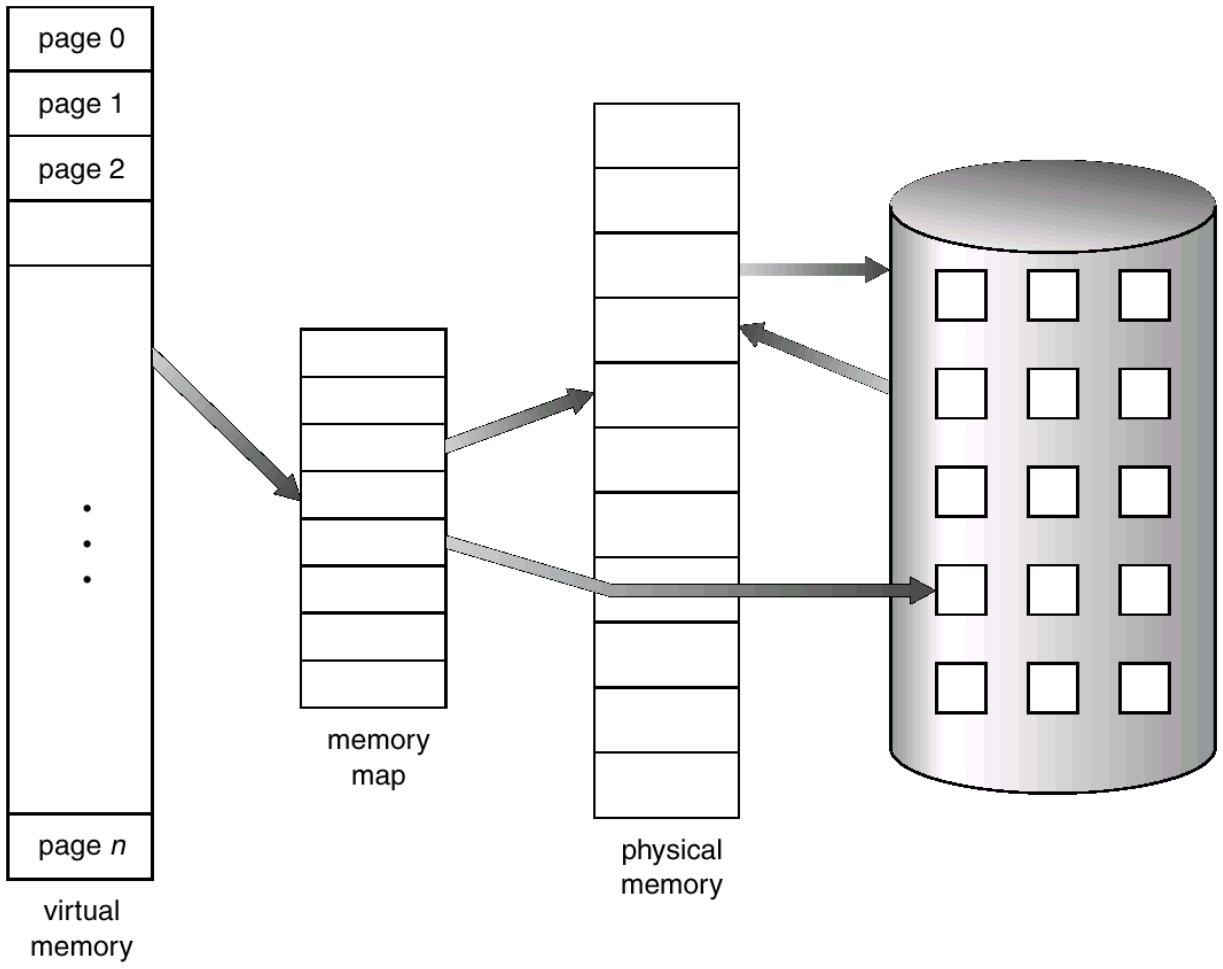
Mémoire Virtuelle: Pourrait Être Énorme!

- **Ex: 16 bits sont nécessaires pour adresser une mémoire physique de 64KB**
- **En utilisant des pages de 1KB, 10 bits sont requis pour le décalage**
- **Pour le *numéro de page* de l'adresse logique nous pouvons utiliser un nombre de bits qui excède 6, car toutes les pages ne doivent pas être en mémoire simultanément**
- **Donc la limite de la mémoire virtuelle est le nombre de bits qui peuvent être réservés pour l'adresse**
  - Dans quelques architectures, ces bits peuvent être inclus dans des registres
- **La mémoire logique est donc appelée *mémoire virtuelle***
  - Est maintenue en mémoire secondaire
  - Les pièces sont amenées en mémoire principale seulement quand c'est nécessaire, sur demande

Mémoire Virtuelle

- **Pour une meilleure performance, la mémoire virtuelle se trouve souvent dans une région du disque qui est n'est pas gérée par le système de fichiers**
  - Mémoire va-et-vient, swap memory
- **La mémoire physique est celle qui est référencée par une adresse physique**
  - Se trouve dans le RAM et cache
- **La conversion de l'adresse logique en adresse physique est effectuée en utilisant les mécanismes étudiés dans le chapitre précédent.**

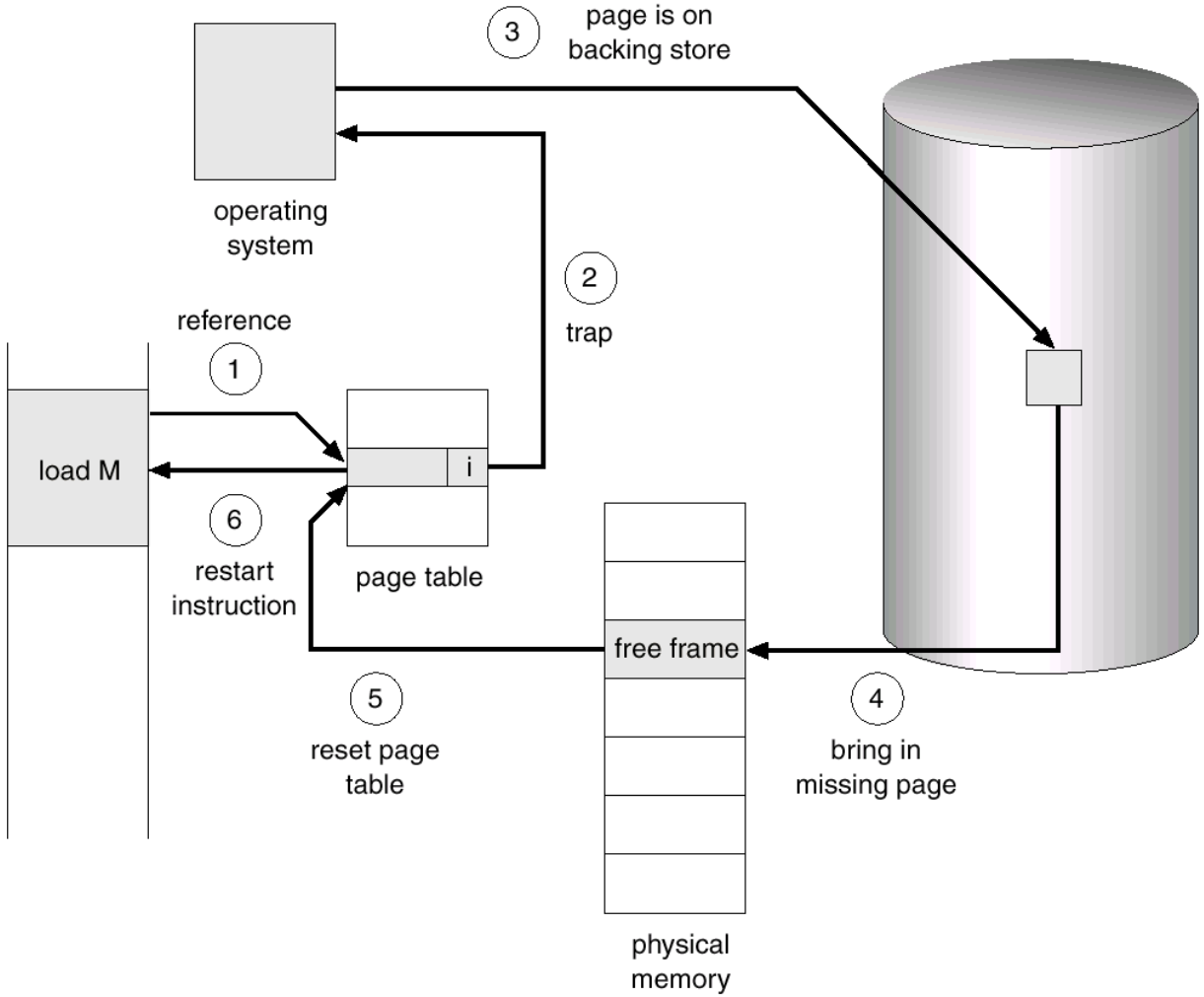
Mémoire virtuelle: le mécanisme de va-et-vient



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 indique si la page ou segment se trouve 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

Exécution d'une défaut de page: va-et-vient plus en détail



Séquence d'événements pour défaut de page

- **Trappe au SE: page demandée pas en RAM**
- **Sauvegarder registres et état du proc dans PCB**
- **Un autre proc peut maintenant gagner l 'UCT**
- **SE détermine si la page demandée est légale**
  - sinon: terminaison du processus
- **et trouve la position de la page sur disque**
  - dans le descripteur de la page
- **lire la page de disque dans un cadre de mémoire libre (supposons qu`il y en a!)**
  - exécuter les ops disque nécessaires pour lire la page

Séquence d'événements pour défaut de page (ctn.)

- **L 'unité disque a complété le transfert et interrompt l'UCT**
  - sauvegarder les registres etc. du proc exécutant
- **SE met à jour le contenu du tableau des pages du processus qui a causé le défaut de page**
- **Ce processus devient prêt=ready**
- **À un certain point, il retournera à exécuter**
  - la page désirée étant en mémoire, il pourra maintenant continuer

Temps moyen d'accès à la mémoire

Supposons que:

- L'accès en mémoire: 100 nanosecs
- temps de traitement de défaut de pagination: 25 millisecs = $25x10^6$ ns
- p: probabilité de ne pas trouver une page en mémoire (défaut)

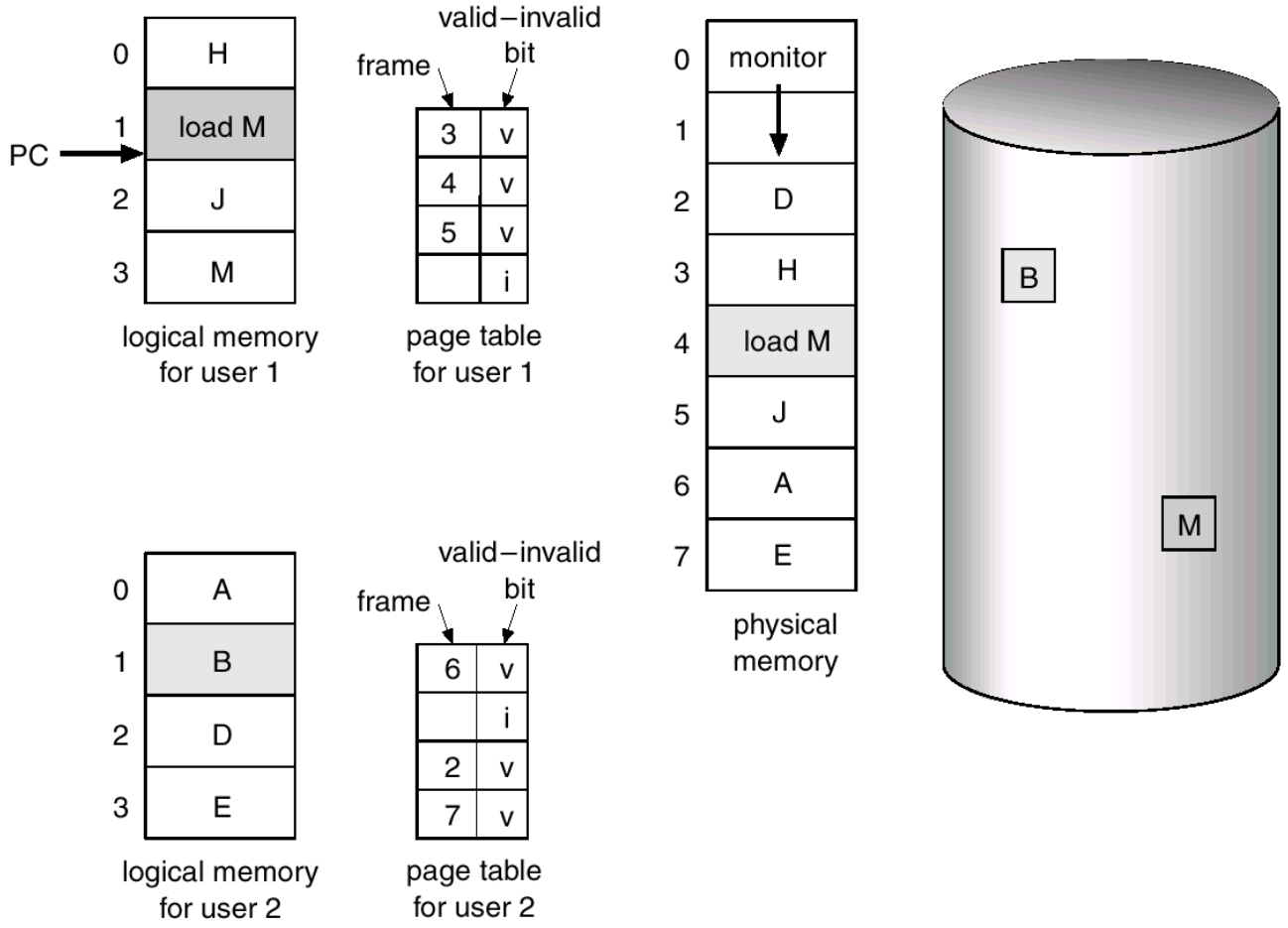
Temps moyen d'accès mémoire:

$(1- p) \times 100 + p \times 25x10^6$ (pas de défaut + défaut)

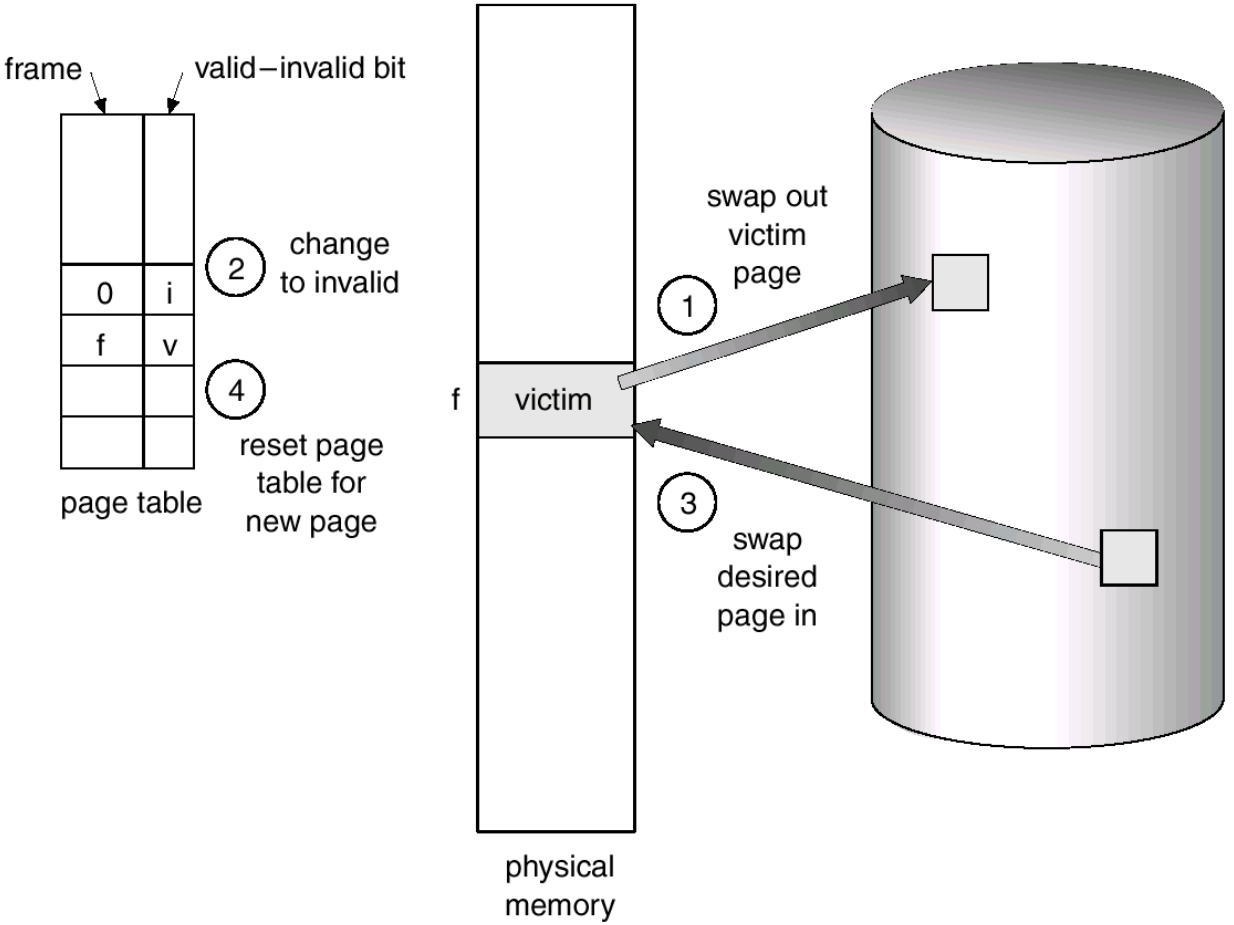
En utilisant la même formule, nous pouvons déterminer quel est le nombre de défauts que nous pouvons tolérer, si un certain niveau de performance est désiré (v. manuel).

Ex. avec ces paramètres, si le ralentissement à cause de pagination ne peut pas excéder 10%, 1 seul défaut de pagination peut être toléré pour chaque 2,500,000 accès de mémoire.

Quand la RAM est pleine mais nous avons besoin d`une page pas en RAM



La page victime...



Remplacement de pages

- **Quoi faire si un processus demande une nouvelle page et qu'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`:**
  - ex. frames contenant le noyau du SE, tampons d 'E/S...

Bit de modification , *dirty bit*

- **La ‘victime’ doit-elle être réécrite en mémoire secondaire?**
- **Seulement si elle a été changée depuis qu`elle a été amenée en mémoire principale**
  - sinon, sa copie sur disque est encore fidèle
- **Bit de modification de chaque descripteur de page indique si la page a été changée**
- **Donc pour calculer le coût en temps d’une référence à la mémoire il faut aussi considérer la probabilité qu’une page soit ‘sale’ et le temps de réécriture dans ce cas**

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, ex. mise à jour de tableaux en mémoire pour chaque accès de mémoire
- Ni l`utilisation de matériel dispendieux

Explication et évaluation des algorithmes

- **Nous allons expliquer et évaluer les algorithmes en utilisant la chaîne de référence pages suivante (prise du livre de Stallings):**

  2, 3, 2, 1, 5, 2, 4, 5, 3, 2, 5, 2

- **Attention: les séquences d'utilisation pages ne sont pas aléatoires...**
  - Localité de référence
- **Ces références proviendront de plusieurs processus**
- **L'évaluation sera faite sur la base de cet exemple qui évidemment n'est pas suffisant pour en tirer des conclusions générales**

Algorithmes pour la politique de remplacement

- **L'algorithme optimal (OPT) choisit pour page à remplacer celle qui a été référencée le plus récemment**
  - 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)

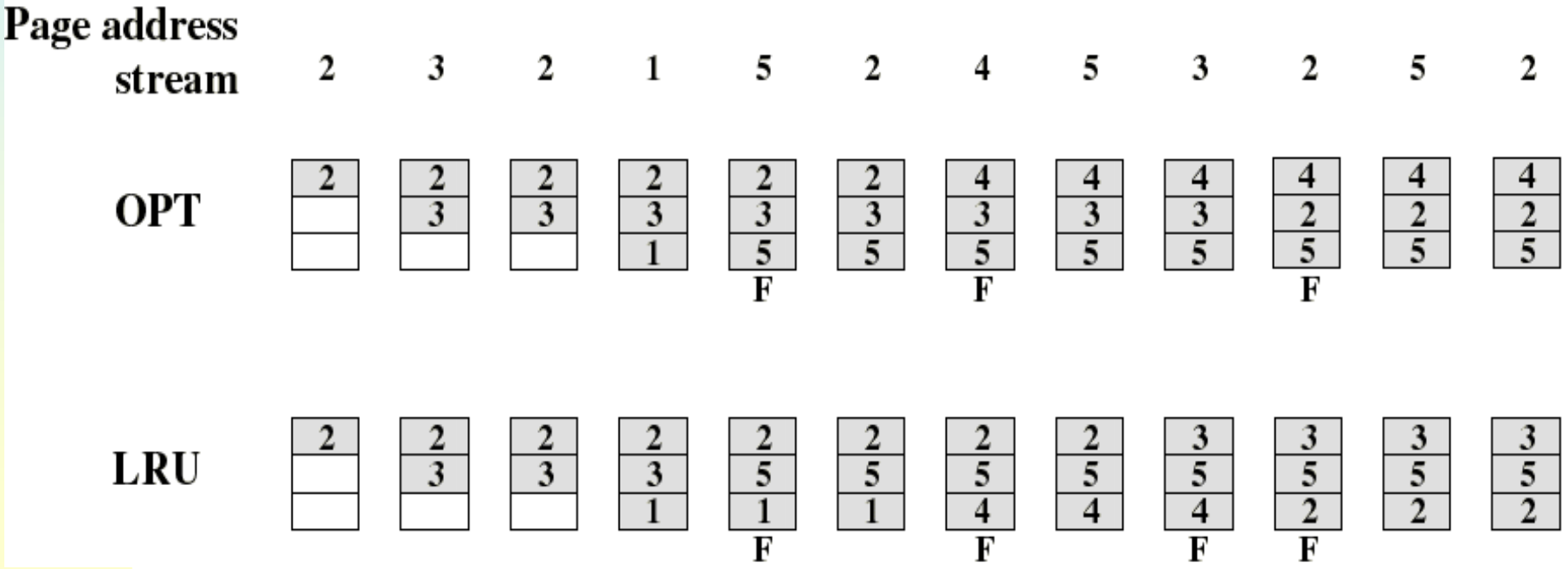
Algorithmes pour la politique de remplacement

- **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'algorithme OPT

Stallings

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

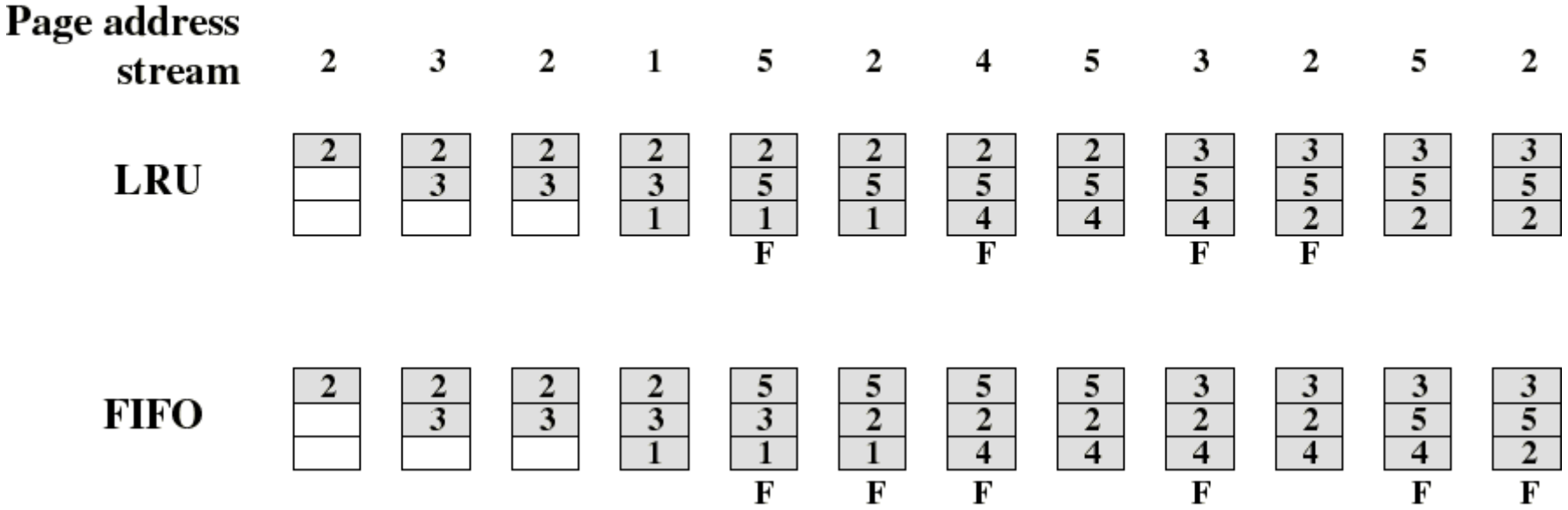
Implémentation problématique de LRU

- **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)**
- **On pourrait penser à 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 avec FIFO!**

Comparaison de FIFO avec LRU (Stallings)



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

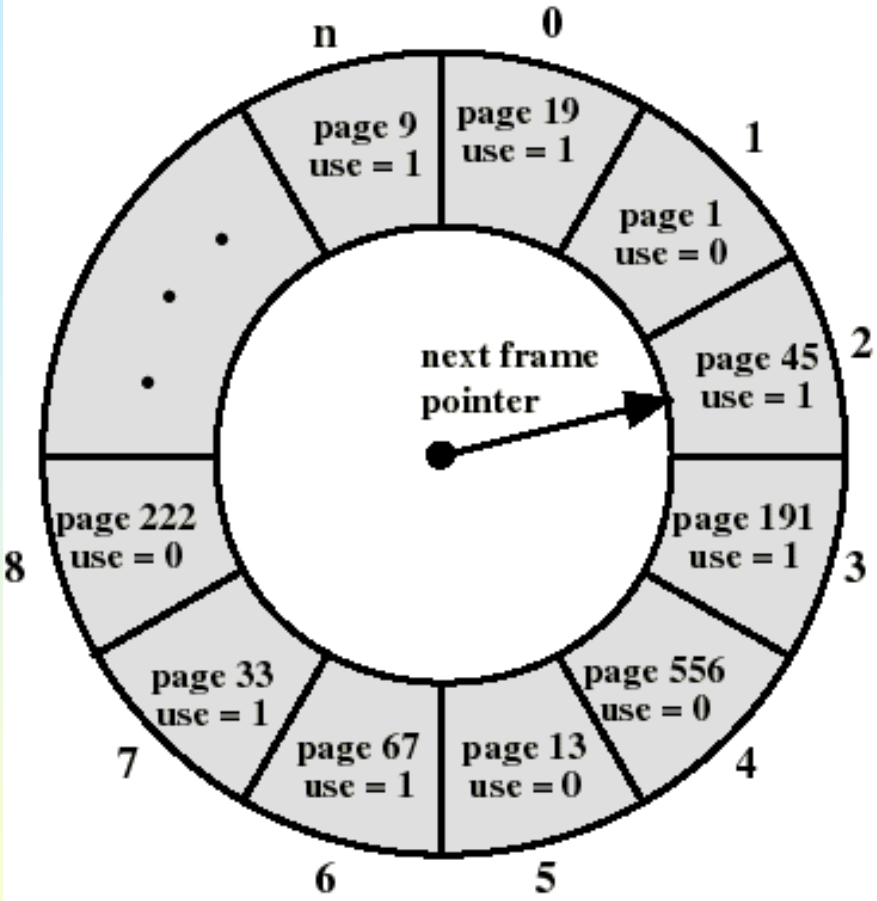
Problème conceptuel avec FIFO

- **Les premières pages amenées en mémoire sont souvent utiles pendant toute l'exécution d'un processus!**
  - variables globales, programme principal, etc.
- **Ce qui montre qu'i y a un problème avec notre façon de comparer les méthodes sur la base d'une séquence aléatoire:**
  - les références aux pages dans un programme réel ne seront pas vraiment aléatoires

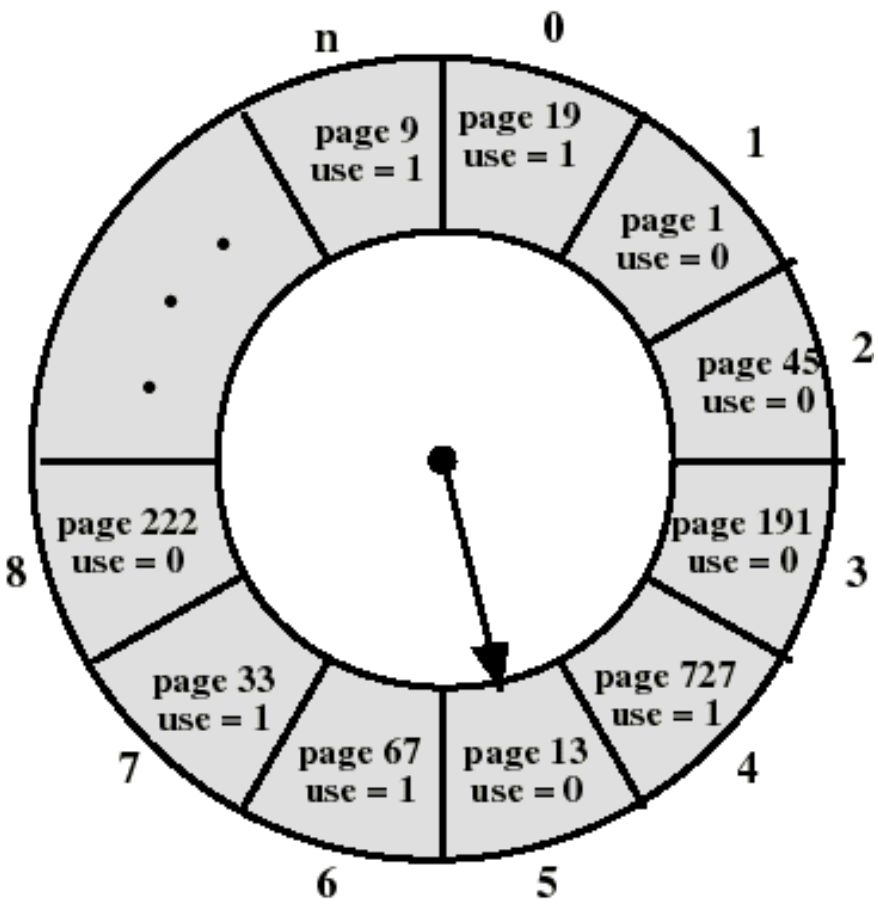
L'algorithme de l'horloge (deuxième chance)

- **Semblable à FIFO, mais les cadres qui viennent d'être utilisés (bit=1) ne sont pas remplacées (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 (Stallings).



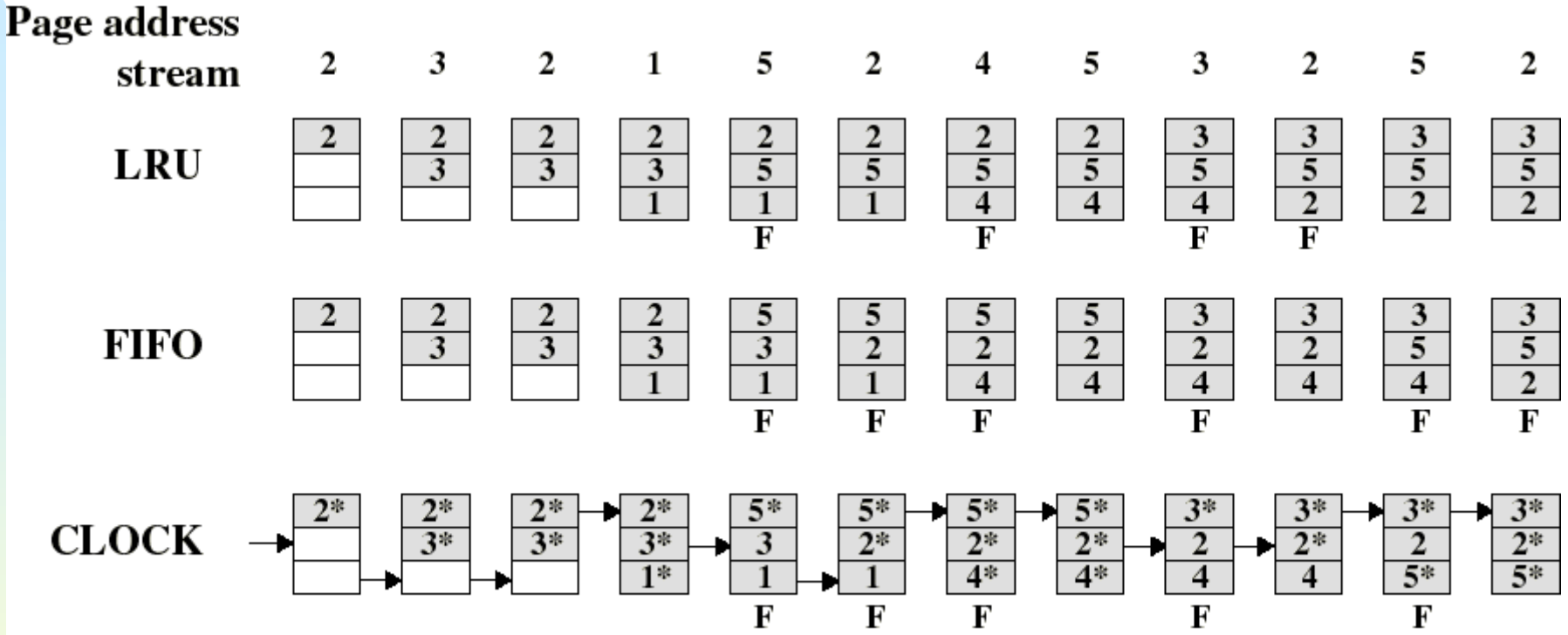
(a) State of buffer just prior to a page replacement



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

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

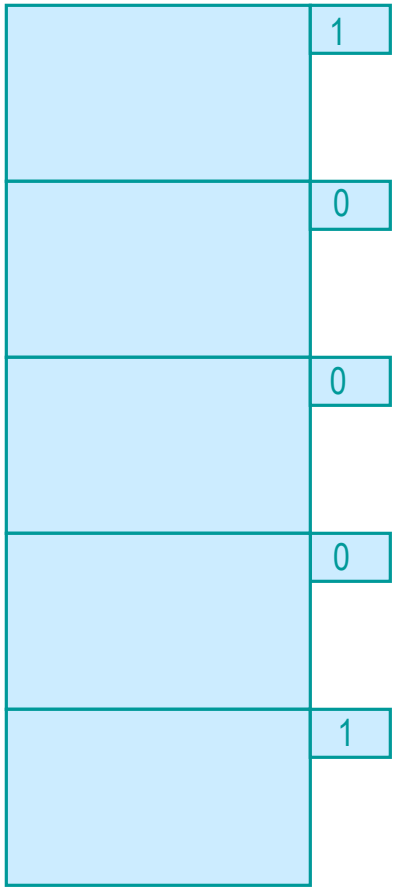
Comparaison: Horloge, FIFO et LRU (Stallings)



- Astérisque indique que le bit utilisé est 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

Matériel additionnel pour l'algorithme CLOCK

- **Chaque bloc de mémoire a un bit 'touché' (use)**
- **Quand le contenu du bloc est utilisé, le bit est mis à 1 par le matériel**
- **Le SE regarde le bit**
  - S'il est 0, la page peut être remplacée
  - S'il est 1, il le met à 0



Mémoire

Comparaison: Horloge, FIFO et LRU

- **Les simulations montrent que l'horloge est presque aussi performant que LRU**
  - Des variances 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 alloués, le nombre de défaut de pages produit par FIFO est presque le double de celui produit par LRU, alors que celui de CLOCK est entre les deux
  - Ce facteur s'approche de 1 lorsque plusieurs (plus de 12) cadres sont alloués.
- **Cependant dans le cas réel avec des milliers ou millions de pages et cadres, la différence n`est pas trop importante...**
  - On peut tranquillement utiliser FIFO

Algorithmes *compteurs*

- Garder un compteur pour les références de chaque page
- LFU (Least Frequently Used): remplace les pages avec le plus petit compteur
- MFU (Most Frequently Used) remplace les pages bien usées pour donner une chance aux nouvelles
- Ces algorithmes ont des implémentations couteuses et ne sont pas beaucoup utilisés

Utilisation d'une pile (stack)

- **Quand une page est utilisée, elle 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 le matériel, car nous ne pouvons pas tolérer l'exécution d'un programme à chaque fois qu'une page est utilisée**
- **Donc pas pratique**

Anomalie de Belady

- **Pour quelques algorithmes, dans quelques cas il pourrait avoir plus de défauts avec plus de mémoire!**
  - Ex. FIFO, mais pas LRU, OPT, CLOCK

Situation considérée normale

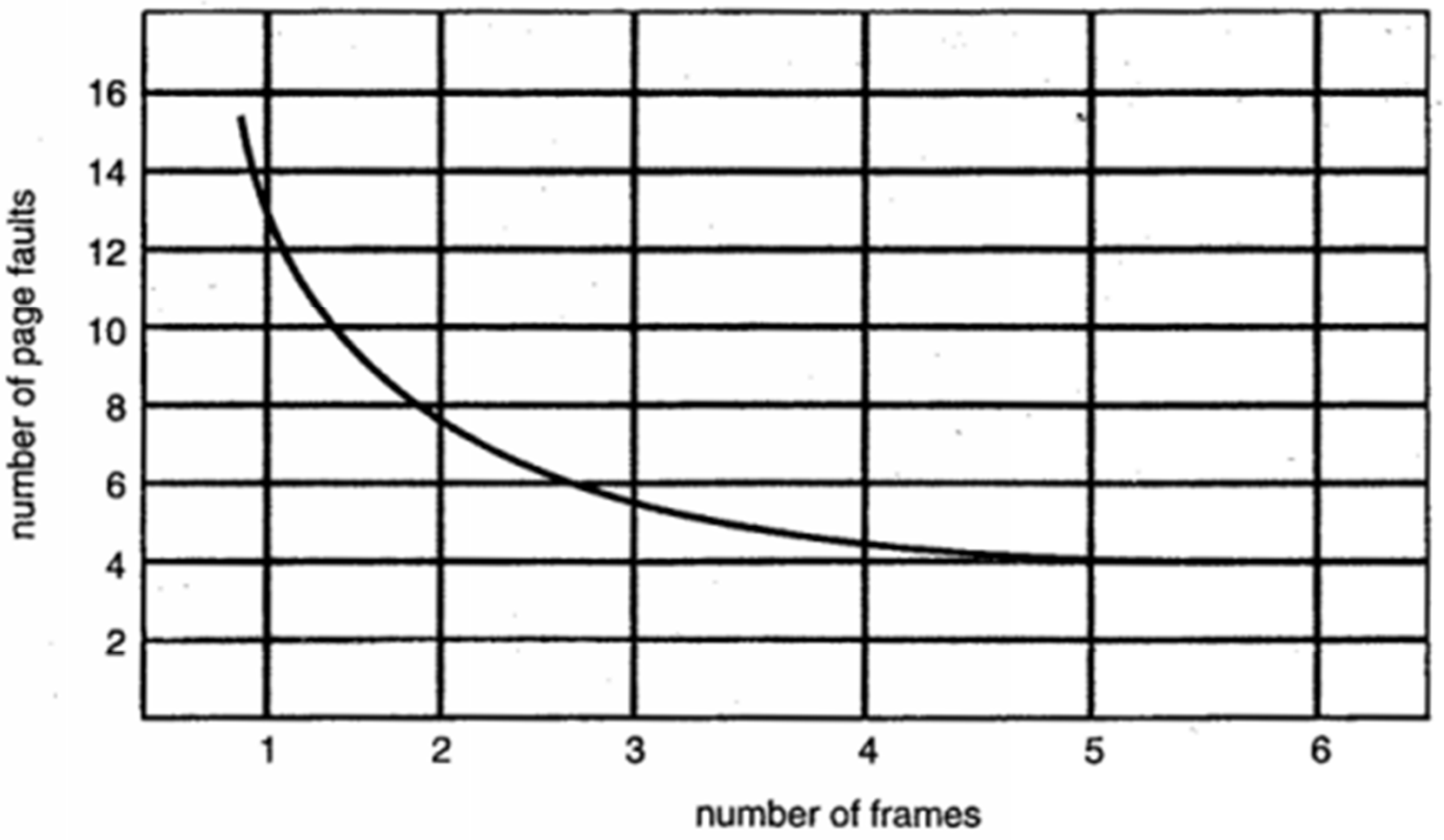
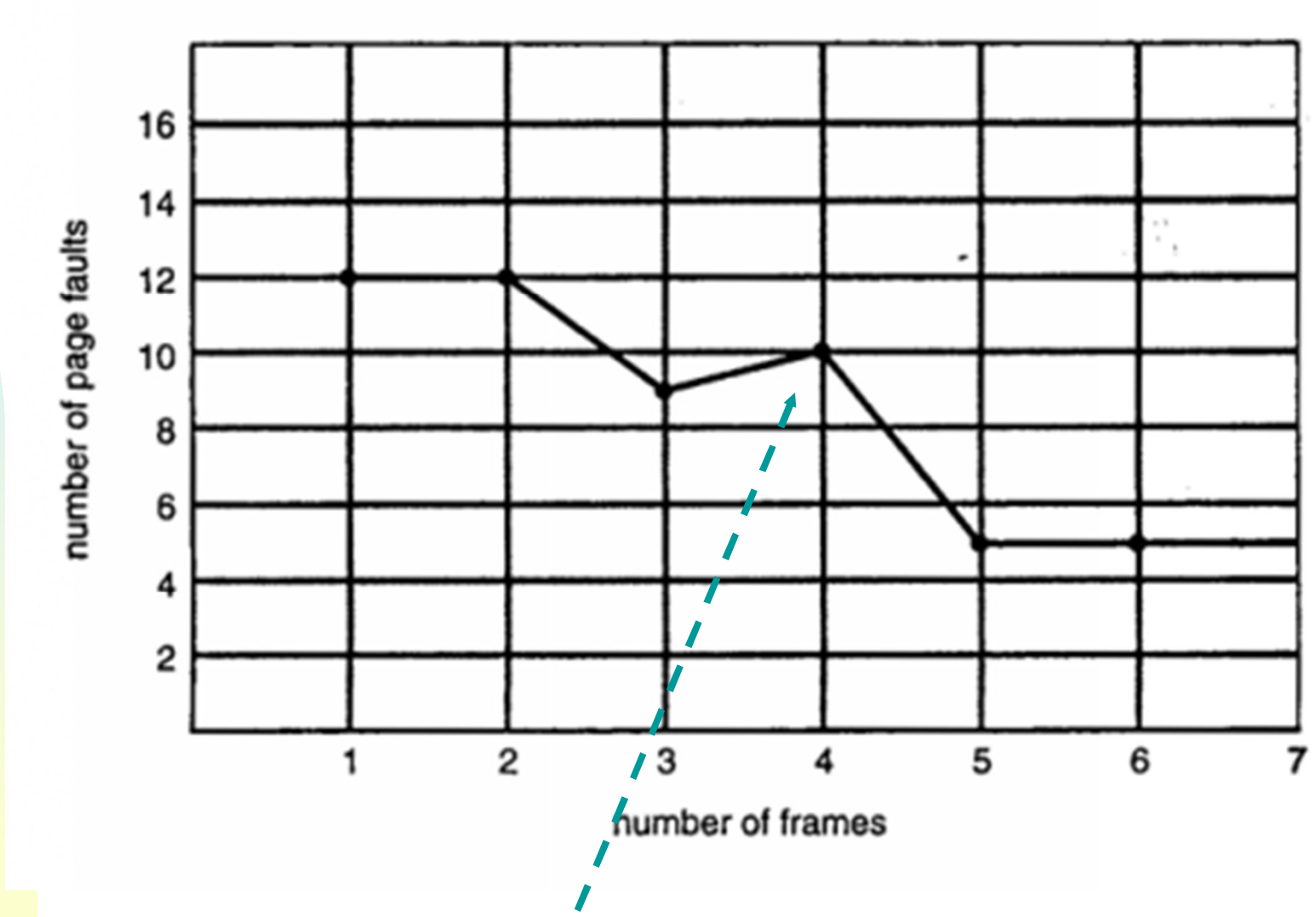


Figure 10.7 Graph of page faults versus the number of frames.

Anomalie de Belady (FIFO)



Cas d'intérêt théorique: + de mémoire, + de fautes (v. livre)

Tamponnage de pages (Page Buffering)

- **Les pages choisies pour être remplacées demeurent temporairement en mémoire principale pour compenser la faible performance d'algorithmes de remplacement comme le FIFO**
- **Deux listes de pointeurs sont maintenues: chaque entrée pointe sur une page désignée pour remplacement**
  - une liste de cadres libres pour les pages n'ayant pas été modifiées depuis leur chargement (disque inutile)
  - une liste de pages modifiées depuis qu'elles furent chargées (doivent être écrites sur disque)
- **Un pointeur est ajouté à la queue d'une des listes lorsqu'une page est désignée pour remplacement et le bit présent est mis à 0 dans l'entrée de la table de pages**
  - mais la page demeure en mémoire principale

Tamponnage de pages (Page Buffering)

- **À chaque défaut de page, ces listes sont examinées pour savoir si la page désirée se trouve encore en mémoire principale**
  - Si oui, le bit présent est remis à 1 (et on enlève l'entrée de la liste): cette page appartient de nouveau au processus.
  - Si non, la page désirée est chargée à l'endroit pointé par la tête de la liste de pages libres, (écrasant la page qui s'y trouvait)
    - la tête de la liste de pages libres est désormais la 2ième entrée
  - (Le numéro de la page physique peut être utilisé pour la recherche sur les listes, ou bien chaque entrée de liste peut contenir le PID et le numéro de page virtuelle)
- **La liste de pages modifiées sert aussi à écrire en lots (et non individuellement) ces pages au disque**

Allocation de cadres

- **Pour exécuter, un processus a besoin d'un nombre minimal de cadres de mémoire**
  - par exemple, quelques instructions pourraient avoir besoin de plusieurs pages simultanément pour exécuter!
- **Il est aussi facile de voir qu'un processus qui reçoit très peu de mémoire subira un nombre excessif de défauts de pagination, donc il sera excessivement ralenti**
- **Comment s'assurer qu'un processus est alloué son minimum**
  - allocation égale: chaque processus a droit a une portion égale de la mémoire physique
  - allocation proportionnelle: chaque processus a droit à une portion proportionnelle à sa taille
    - le critère devrait plutôt être le besoin de pages: voir working set

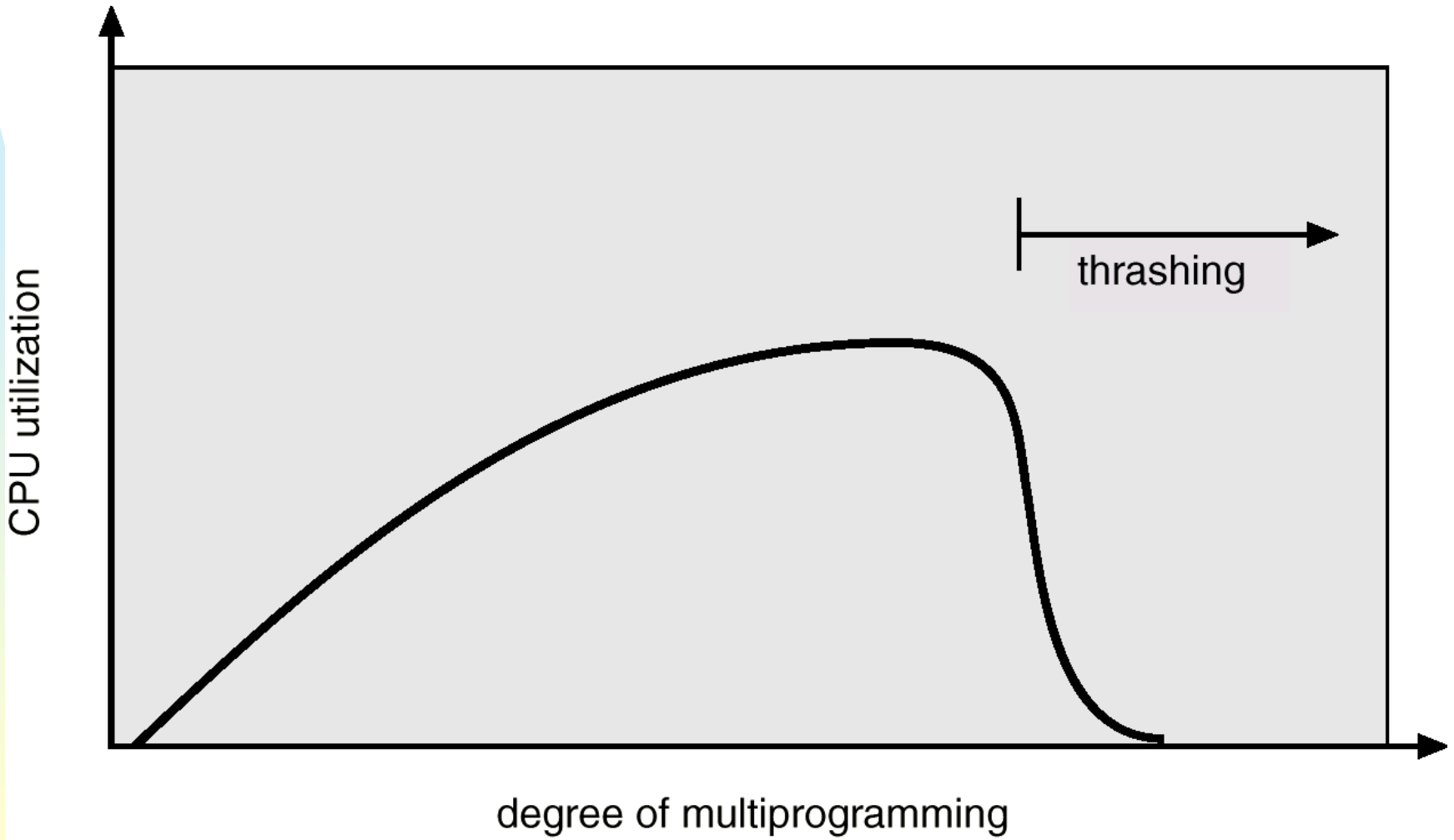
Allocation globale ou locale

- globale: la ‘victime’ est prise de n’importe quel processus
- locale: la ’victime’ est prise du processus qui a besoin de la page

Écroulement ou thrashing (liter.: défaite)

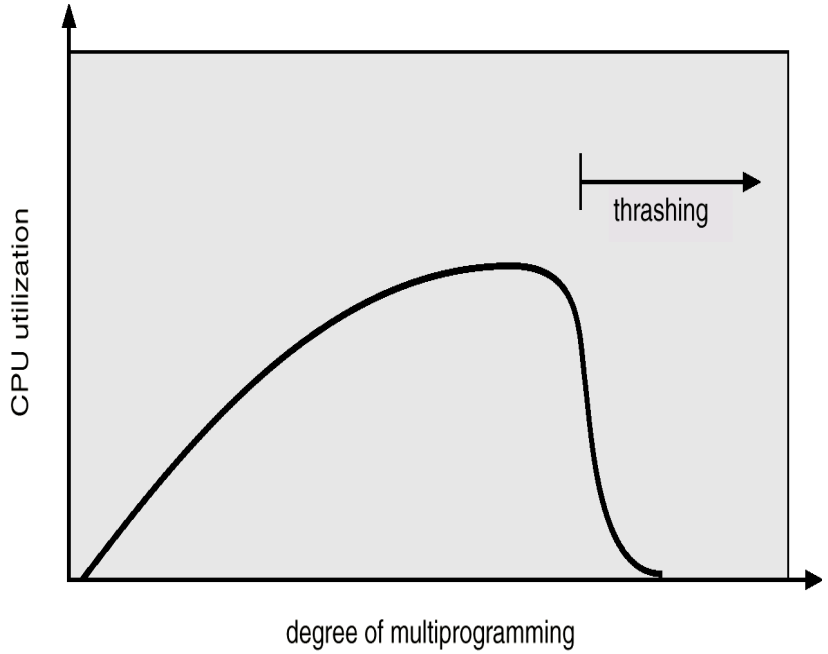
- **S'il n'y a pas assez de mémoire pour exécuter un processus sans trop de défauts de pagination, le processus finira par passer trop de temps dans les files d`attente**
- **Si cette situation se généralise à plusieurs processus, l 'UCT se trouvera à être sous-utilisée**
- **Le SE pourra chercher à remédier à cette situation en augmentant le niveau de multiprogrammation**
  - plus de processus en mémoire!
  - moins de mémoire par processus!
  - 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 processus à un moment d'exécution donné est l'ensemble des pages dont le processus a besoin pour exécuter sans trop de défauts de pagination**
  - Malheureusement, un concept flou

Chercher à prévoir les demandes de pages sur la base des demandes passées

- **Fixer un intervalle Δ**
- **Les pages intéressées par les dernières Δ operations de mémoire sont dans l 'ensemble de travail déterminé par Δ**
- **Comment choisir un Δ approprié?**

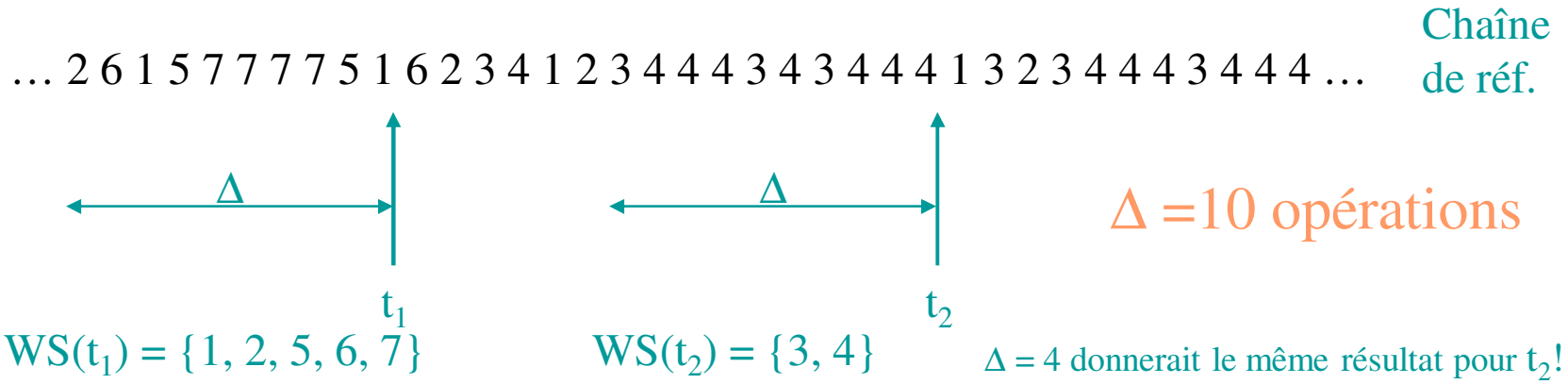


Figure 10.16 Working-set model.

Modèle de l'ensemble de travail

- **Δ = une fenêtre d'ensemble de travail**
  - = un nombre fixe de références de pages
  - ex. 10.000 opérations de mémoire
- **S'il est trop petit, il ne contiendra pas tout l'ensemble de pages couramment utilisé par un processus**
- **S'il est trop grand, il contiendra plusieurs ensembles de pages**
- **WSS_i (ensemble de travail du processus i)**
- **$D = \sum WSS_i$ nombre total de cadres demandés par tous les processus en exécution**
- **Si D > mémoire $\Rightarrow$ Risque d'écroulement**
- **S'assurer que ceci ne se produie pas**
  - si nécessaire, suspendre un des processus
- **Problème: choisir un bon Δ**
  - **peut être fait par le gérant du système**

Implémentation du concept de WS: difficile!

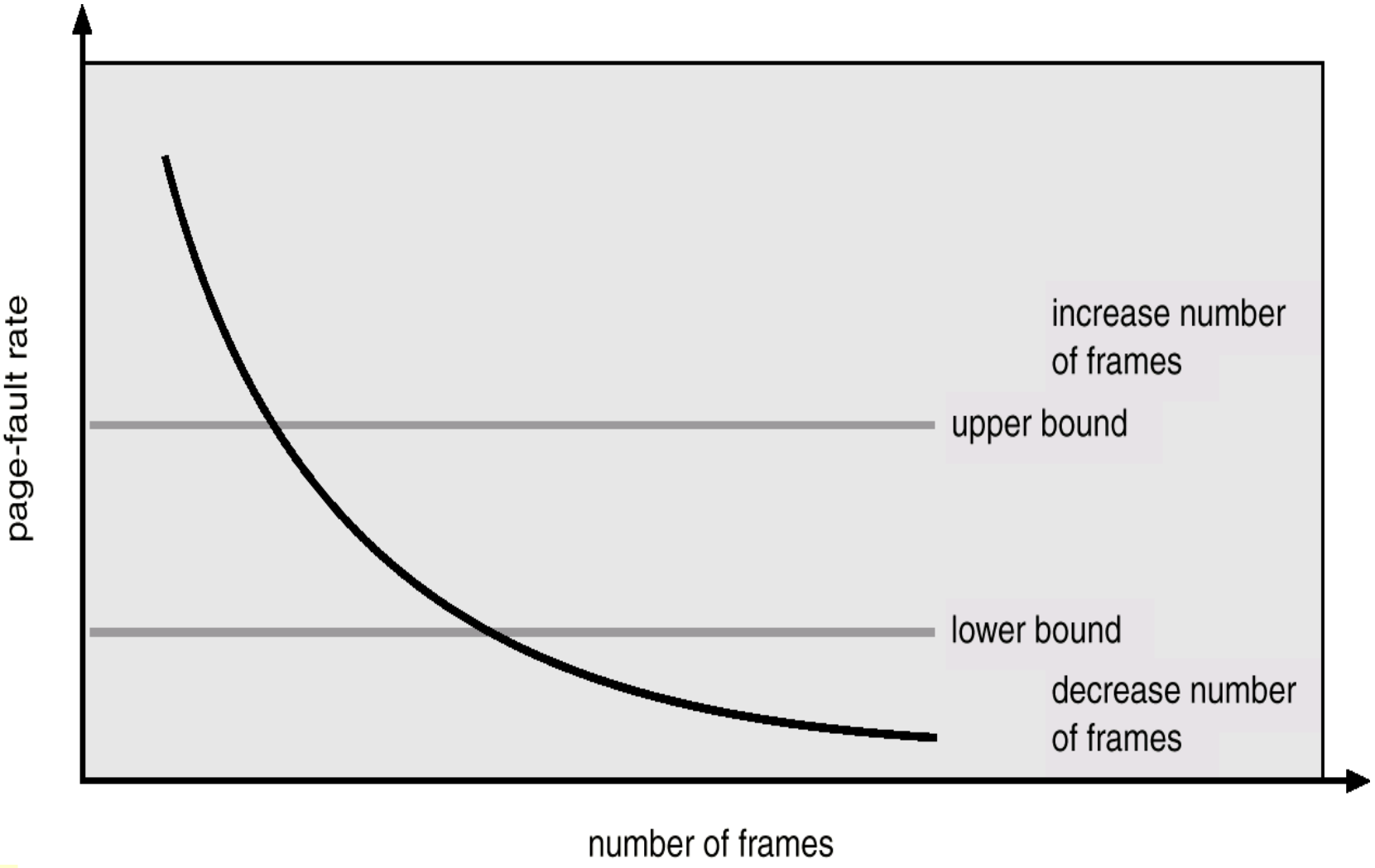
- Minuterie et bits de référence
- Bit de référence qui est mis à 1 chaque fois qu'une page est utilisée
- Minuterie qui s'interrompt régulièrement pour voir les pages qui ont été utilisées dans un intervalle de temps

Le concept de WS en pratique

- **Deux types de difficultés:**
  - fixer le Δ de façon différente pour chaque processus, pour représenter ses besoins
- **Du matériel spécial est nécessaire pour suivre le WS d'un proc à un moment donné**

Pour chaque processus, il existe une dimension de mémoire acceptable

ceci suggère **une approche plus pratique**



Une méthode plus facile à implanter que WS

- Le gérant du système détermine quelles sont les nombres de défauts de pagination maximales et minimales tolérables dans le système, et pour chaque travail, selon ses caractéristiques
- Si un travail en produit plus que sa juste partie, lui fournir plus de mémoire
- Si un travail en produit moins, lui donner moins de mémoire
- Suspendre si possible des travaux qu'on ne peut pas satisfaire
- Ou amorcer d'autres travaux si les ressources sont disponibles

Taille de pages et localité processus

- **Dans le cas de programmes qui exécutent du code qui ’saute’ beaucoup, les petites pages sont préférables (code OO est dans cette catégorie)**

Verrouillage de pages en mémoire

- **Certaines pages doivent être verrouillées en mémoire, ex. celles qui contiennent le noyau du SE**
- **Il est aussi essentiel de verrouiller en mémoire des pages sur lesquelles il y a exécution d'E/S**
- **Ceci peut être obtenu avec un bit `verrou` sur le cadre de mémoire**
  - ce bit veut dire que ce cadre ne peut pas être sélectionné comme `victime`

Systèmes en temps réel

- **Avec la mémoire virtuelle, les temps d'exécution d'un processus deviennent moins prévisibles**
  - retards inattendus à cause de la pagination
- **Donc les systèmes en temps réel `durs` utilisent rarement la mémoire virtuelle**

Combinaison de techniques

- **Les SE réels utilisent les techniques que nous avons étudiées en *combinaison*, e.g.**
  - Linux utilise le buddy system en combinaison avec la pagination (la plus petite portion de mémoire allouable est une page)
  - d`autres systèmes utilisent les partitions fixes avec la pagination, ce qui peut être fait de plusieurs façons:
    - diviser la mémoire *réelle* en partitions fixes, assigner chaque partition à un ou plusieurs processus, puis paginer un processus dans la partitions qui lui a été assignée
    - diviser la mémoire *virtuelle* en partitions, assigner chaque partition à un ou plus. processus, puis utiliser la technique appropriée pour chaque processus dans sa partition
- **Les SE réels sont complexes et variés, mais les principes étudiés dans ce cours en constituent la base.**

Conclusions 1

- **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

Conclusions 2

- **Le problème de choisir la page victime n ’est pas facile.**
  - Les meilleurs algorithmes sont impossibles ou difficiles à implanter
  - Cependant en pratique l’algorithme FIFO est acceptable

Conclusions 3

- **Il faut s'assurer que chaque processus ait assez de pages en mémoire physique pour exécuter efficacement**
  - risque d'écroulement
- **Le modèle de l'ensemble de travail exprime bien les exigences, cependant il est difficile à implanter**
- **Un solution plus pragmatique et de décider de donner + ou - de mémoire aux processus selon leur débit de défauts de pagination**
- **À fin que ces mécanismes de gestion mémoire soient efficaces, plusieurs types de mécanismes sont désirables dans le matériel**

Concepts importants du Chap. 9

- **Localité des références**
- **Mémoire virtuelle implémentée par va-et-vient des pages, mécanismes, défauts de pages**
- **Adresses physiques et adresses logiques**
- **Temps moyen d'accès à la mémoire**
  - Récriture ou non de pages sur mém secondaire
- **Algorithmes de remplacement pages:**
  - OPT, LRU, FIFO, Horloge
  - Fonctionnement, comparaison
- **Écroulement, causes**
- **Ensemble de travail (working set)**
- **Relation entre la mémoire allouée à un proc et le nombre d'interruptions**
- **Relation entre la dimension de pages et le nombre d'interruptions**