Mémoire virtuelle

DIRO

Silberschatz, Galvin and Gagne ©2013, Monnier ©2016, Paull ©2018

Menu

- Préliminaires
- Pagination à la demande
- Remplacement de pages
- Allocation de frames
- Thrashing
- Fichiers memory-mapped
- Allocation de mémoire noyau
- Autre considérations

DIRO

Silberschatz, Galvin and Gagne ©2013, Monnier ©2016, Paull ©2018

Menu

- **Préliminaires**
- Pagination à la demande
- Remplacement de pages
- Allocation de frames
- Thrashing
- Fichiers memory-mapped
- Allocation de mémoire noyau
- Autre considérations

DIRO

Silberschatz, Galvin and Gagne ©2013, Monnier ©2016, Paull ©2018

Rappel

- Code doit être en mémoire centrale pour exécuter
  - Mais le programme entier rarement utilisé
    - ✓ Code d'erreur, routines inhabituelles, grandes structures de données
  - Mémoire centrale est trop petit pour accommoder le niveau désiré de multiprogrammation
- Dernier chapitre -> allocation de mémoire non contiguë
  - Segmentation
  - Pages/Frames
- Ce chapitre -> chargement partiel d’un processus en mémoire centrale
  - Comment? Mémoire “virtuelle”

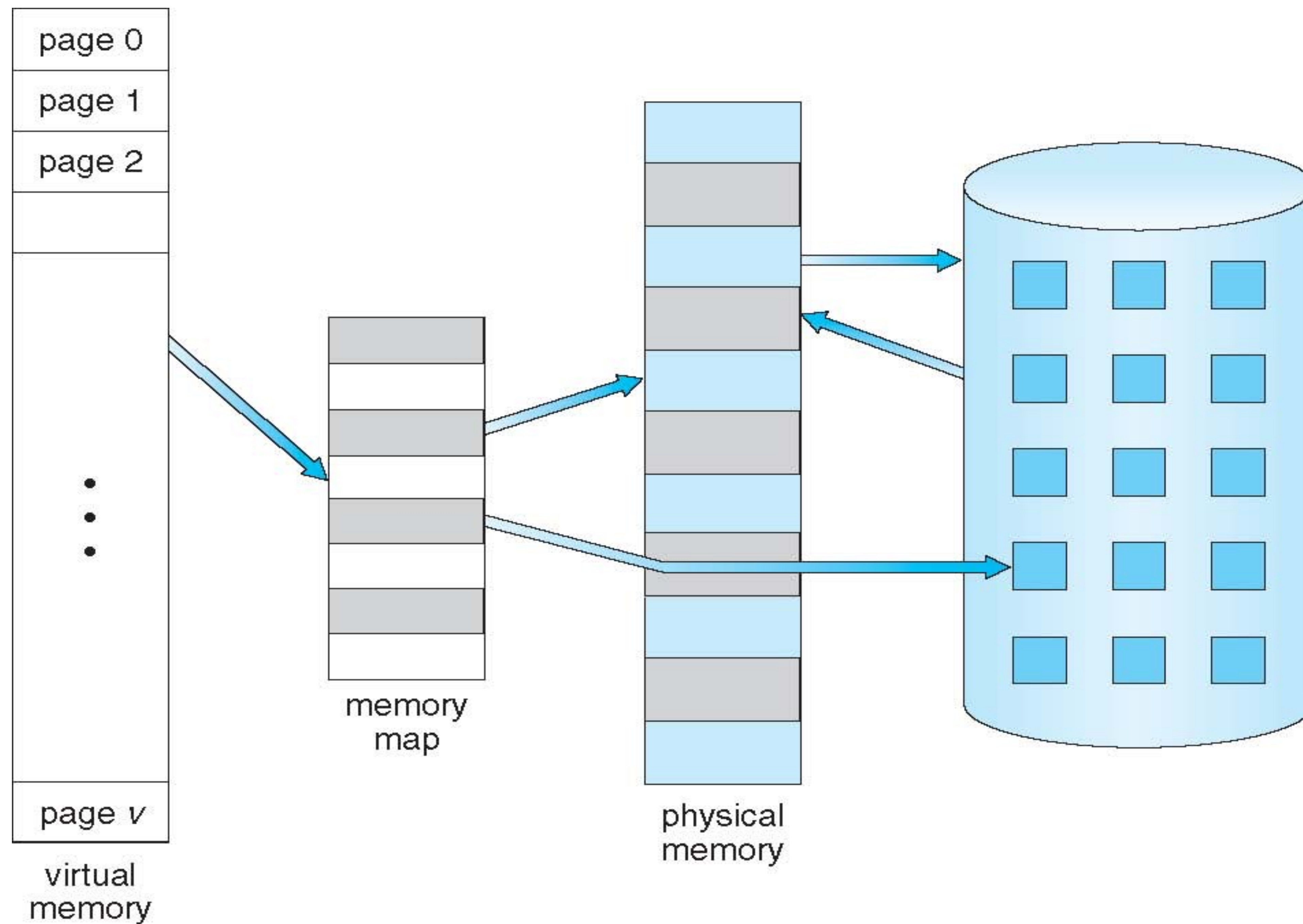
Silberschatz, Galvin and Gagne ©2013, Monnier ©2016, Paull ©2018

Introduction

- Mémoire virtuelle – séparation de la mémoire logique et la mémoire physique
  - Seule une partie du programme doit être en mémoire pour l'exécution
  - L'espace d'adressage logique peut donc être beaucoup plus grand que l'espace d'adressage physique
  - Permet aux espaces d'adresses physique d'être partagés par plusieurs processus
  - Permet une création de processus plus efficace (e.g. avec `fork()`)
  - Plus de programmes fonctionnant simultanément
  - Moins d'E / S nécessaire pour charger ou swapper des processus
- Mémoire virtuelle peut fonctionner avec un système:
  - Pagination à la demande
  - Segmentation à la demande

Silberschatz, Galvin and Gagne ©2013, Monnier ©2016, Paull ©2018

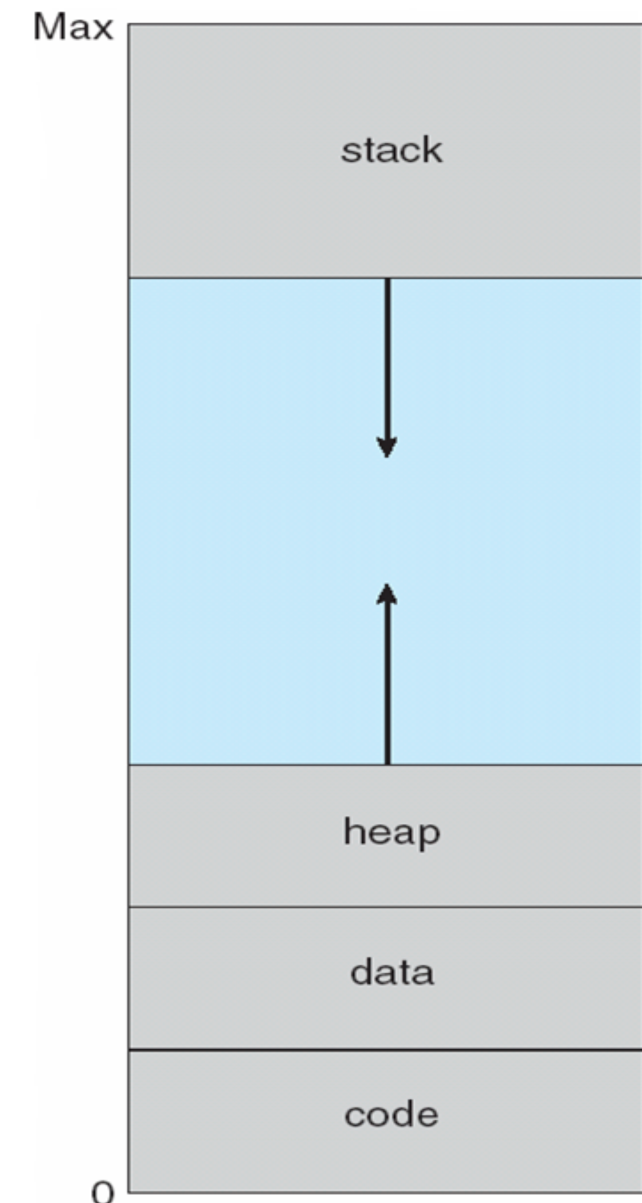
Schéma de mémoire virtuelle



Silberschatz, Galvin and Gagne ©2013, Monnier ©2016, Paull ©2018

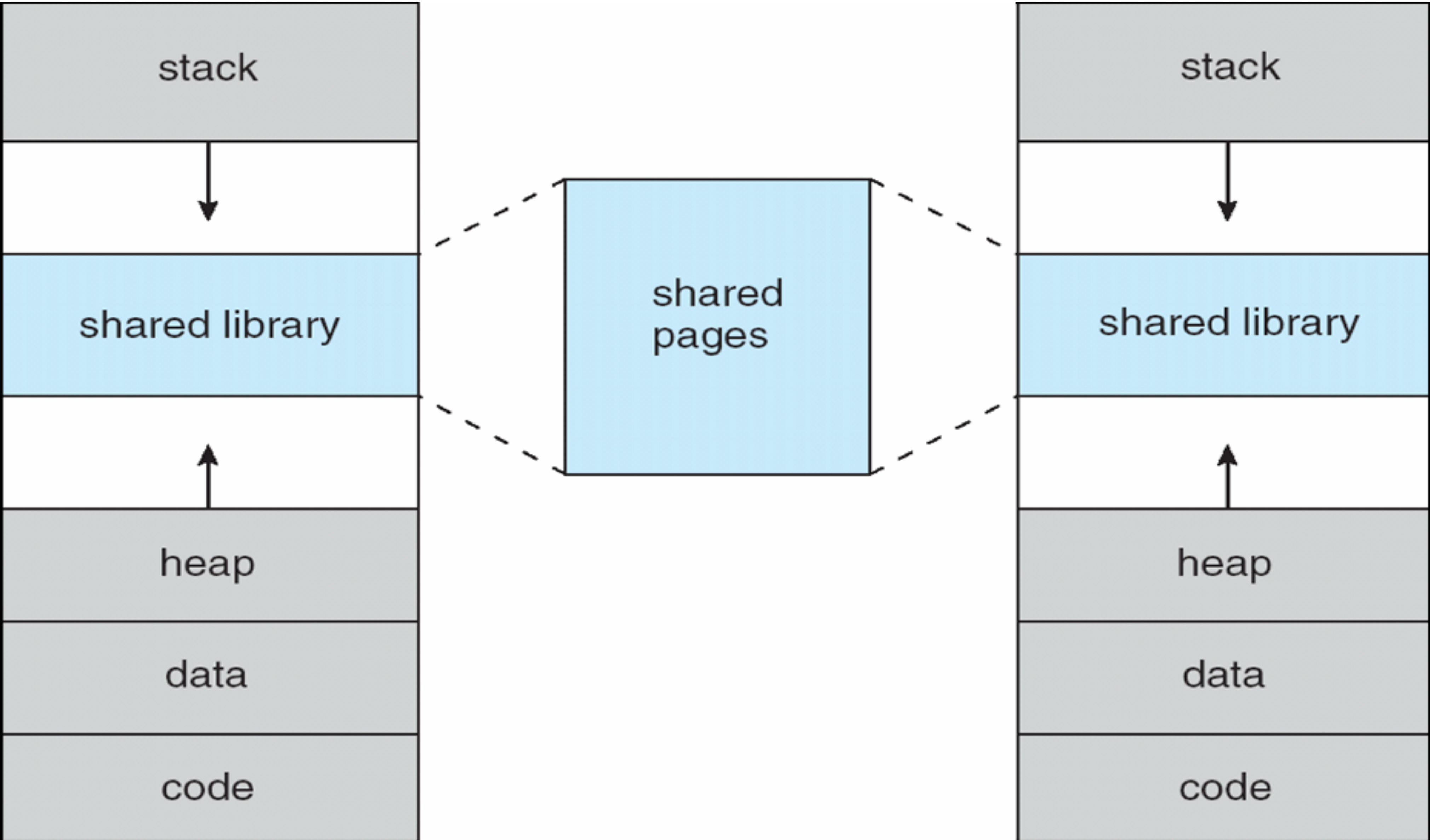
Espace d'adressage virtuel

- Espace d'adressage avec des trous
- Trous pour faciliter allocations futures:
  - Gradir la pile
  - Libraries liées dynamiquement
- Pages partagées avec d'autres processus



Silberschatz, Galvin and Gagne ©2013, Monnier ©2016, Paull ©2018

Librairie partagée



Silberschatz, Galvin and Gagne ©2013, Monnier ©2016, Paull ©2018

Menu

- Préliminaires
- **Pagination à la demande**
- Remplacement de pages
- Allocation de frames
- Thrashing
- Fichiers memory-mapped
- Allocation de mémoire noyau
- Autre considérations

Silberschatz, Galvin and Gagne ©2013, Monnier ©2016, Paull ©2018

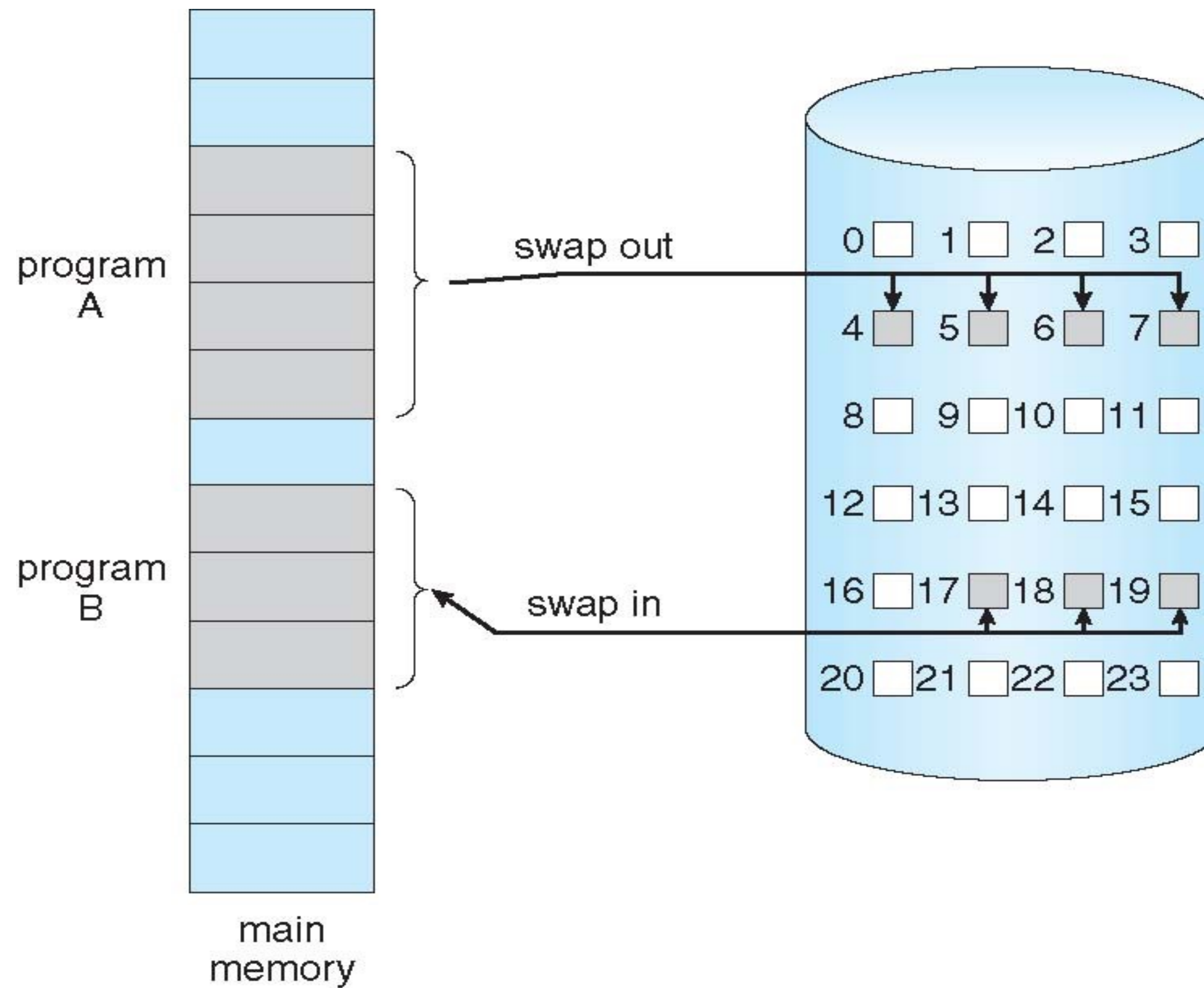
Pagination à la demande

- Option 1: Mettre tout le processus en mémoire au moment du chargement
- Option 2: N'apportez une page en mémoire que lorsque cela est nécessaire
  - Moins d'E/S
  - Moins de mémoire utilisée
  - Réponse plus rapide -> plus de processus en mémoire
- Accès mémoire => besoin d'un page? fait le référence:
  - En mémoire => le CPU se charge de tout
  - Sinon, trap vers le SE
  - Si le référence est invalide => abort!
  - Autrement, charger la page du disque en mémoire et ressayer
- "Lazy swapper" – n'échange jamais une page en mémoire à moins que la page ne soit nécessaire
  - Swapper => processus, Pager => pages

- Option 3: Quelque part au milieu

Silberschatz, Galvin and Gagne ©2013, Monnier ©2016, Paull ©2018

Transfert du/vers le disque



Silberschatz, Galvin and Gagne ©2013, Monnier ©2016, Paull ©2018

Le bit valide / invalide

- Avec chaque entrée de table de page un bit valide-invalide est associé:
  - v => dans mémoire centrale (memory resident)
  - i => pas de la mémoire centrale
    1. valide mais pas de la mémoire
    2. invalide (pas dans l'espace logiciel du processus

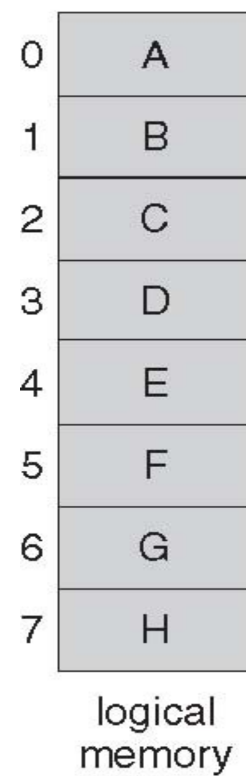
- Pendant la traduction d'adresse, si le bit valide-invalide dans l'entrée de la table de page est i ⇒ erreur de page (**page fault**)

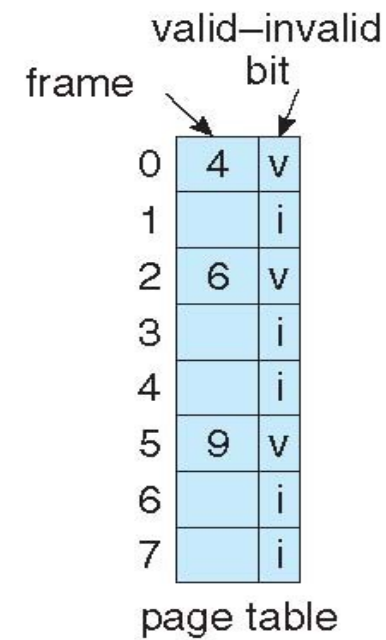
Frame #	valid-invalid bit
	v
	v
	v
	v
	i
	i
	i

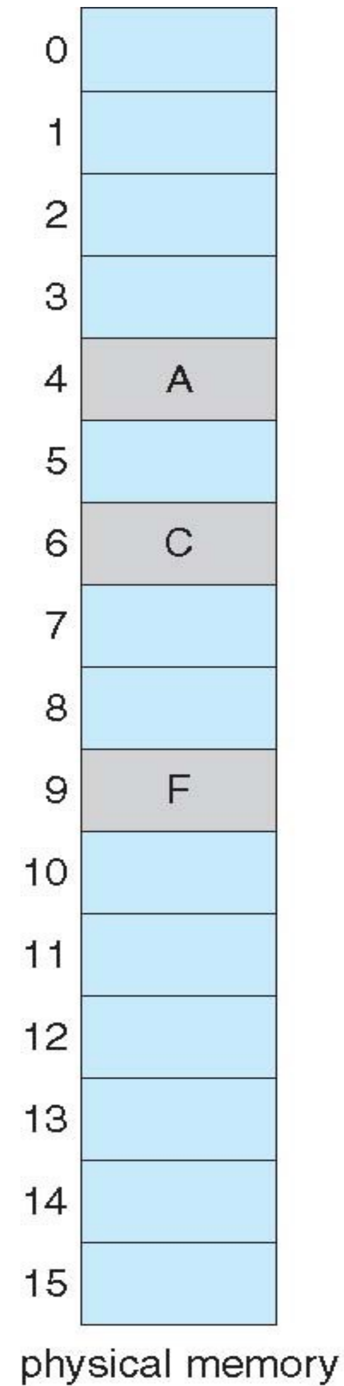
table de page

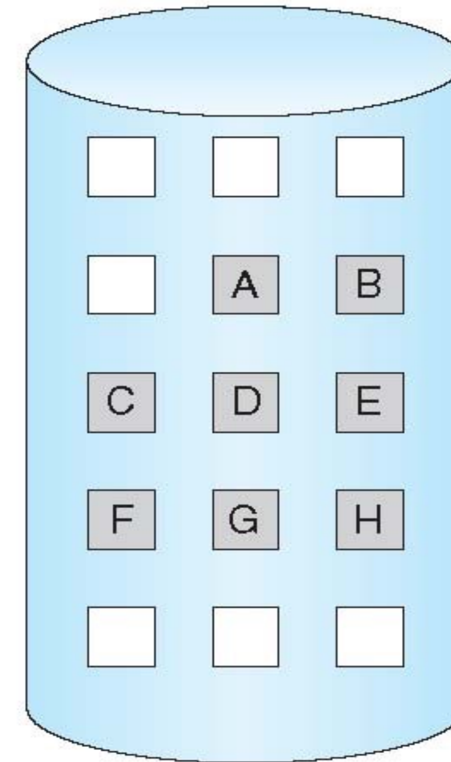
Silberschatz, Galvin and Gagne ©2013, Monnier ©2016, Paull ©2018

Tables de pages avec page manquantes









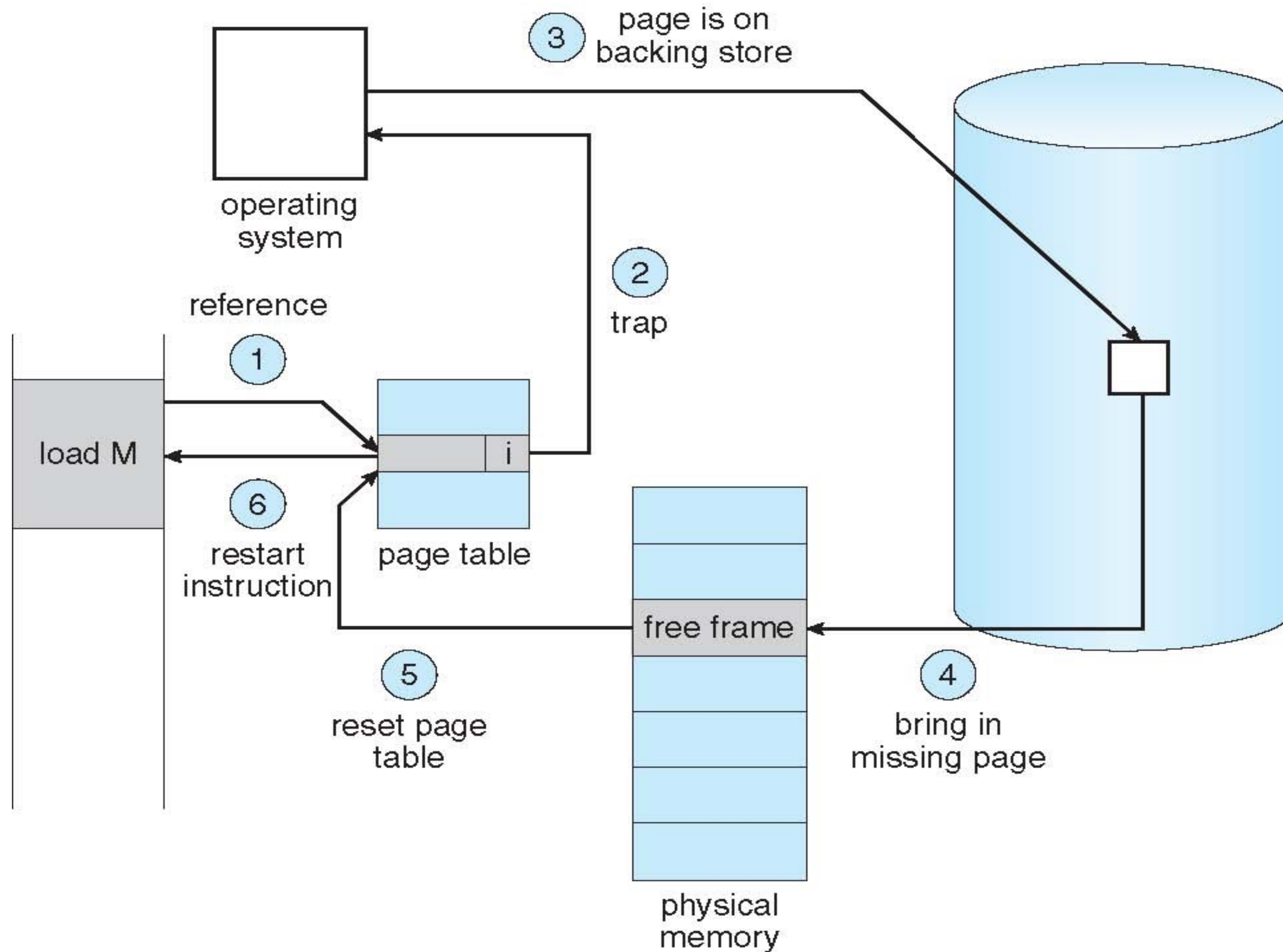
Silberschatz, Galvin and Gagne ©2013, Monnier ©2016, Paull ©2018

Page fault

1. Le CPU utilise table des pages pour trouver l'address physique
2. Si la table des pages indique une entrée invalide => page fault! CPU passe en mode noyau et appelle le SE
3. Le SE cherche dans sa propre table, si l'adresse logique est vraiment invalide, termination du processus, sinon, trouver un frame libre
4. Transférer la page depuis le disque vers cette frame
5. Mettre à jour la table des pages
6. Réexécuter l'instruction du programme

Silberschatz, Galvin and Gagne ©2013, Monnier ©2016, Paull ©2018

Schéma d’une page fault



Silberschatz, Galvin and Gagne ©2013, Monnier ©2016, Paull ©2018

Pagination sur demande

- Cas extrême - aucun pages en mémoire au départ
  - SE met le PC à la première instruction du processus -> page fault   (pour chaque processus)
- Difficulté: une instruction peut accéder à plusieurs pages
  - rare en raison de la localité de référence
- Materiel requise pour la pagination sur demande
  - MMU avec des bits valide/invalide par page
  - Une mémoire secondaire
  - La capacité de réexécuter une instruction
    - ✓ Instructions atomiques
    - ✓ Garder trace de la partie déjà exécutée

Silberschatz, Galvin and Gagne ©2013, Monnier ©2016, Paull ©2018

Coût d’un page fault

1. Trap au système d'exploitation
2. Enregistrer le PCB
3. Déterminer que l'interruption était une page fault
4. Vérifiez que la référence de la page était légale et déterminez l'emplacement de la page sur le disque
5. Émettre une lecture à partir du disque à un cadre libre:
   A. Attendre dans une file d'attente pour ce périphérique jusqu'à ce que la demande de lecture soit traitée
   B. Attendre la recherche de l'appareil et / ou le temps de latence
   C. Commencer le transfert de la page vers un cadre libre
6. En attendant, allouez le CPU à un autre utilisateur
7. Recevoir une interruption du système d'E/S de disque (E/S terminée)
8. Enregistrer le PCB pour l'autre utilisateur
9. Déterminer que l'interruption provient du disque
10. Corriger la table de la page et les autres tables pour afficher la page est maintenant en mémoire
11. Attendez que le processeur soit à nouveau alloué à ce processus
12. Restaurez le PCB et la nouvelle table de pages, puis reprenez l'instruction interrompue

Silberschatz, Galvin and Gagne ©2013, Monnier ©2016, Paull ©2018

Temps d'accès effectif

- Page Fault Rate $0 \le p \le 1$
  - p = 0, pas de page faults
  - p = 1, tous les référence sont des page faults
- Effective Access Time (EAT)

  EAT = (1 – p) x memory access time

  + p x (page fault overhead + swap page in + restart overhead)

Silberschatz, Galvin and Gagne ©2013, Monnier ©2016, Paull ©2018

Exemple

- Temps d’accès à la mémoire centrale = 200 nanoseconds
- Temps de service d’un page fault = 8 milliseconds (8,000,000 nanoseconds)

EAT = (1 – p) x 200 + p x (8 milliseconds)
= (1 – p) x 200 + p x 8,000,000
= 200 + p x 7,999,800

- 1 “miss” par 1,000 accès:

  EAT = 8.2 microseconds = 8199.8 nanoseconds (= 400 * Temps d’accès mémoire!!!)

- Veut une dégradation des performances < 10% ? (EAT < 1.1 * Temps d’accès mémoire)

  220 > 200 + 7,999,800 x p
  20 > 7,999,800 x p

  p < .0000025

  => 1 page fault par 400,000

Silberschatz, Galvin and Gagne ©2013, Monnier ©2016, Paull ©2018

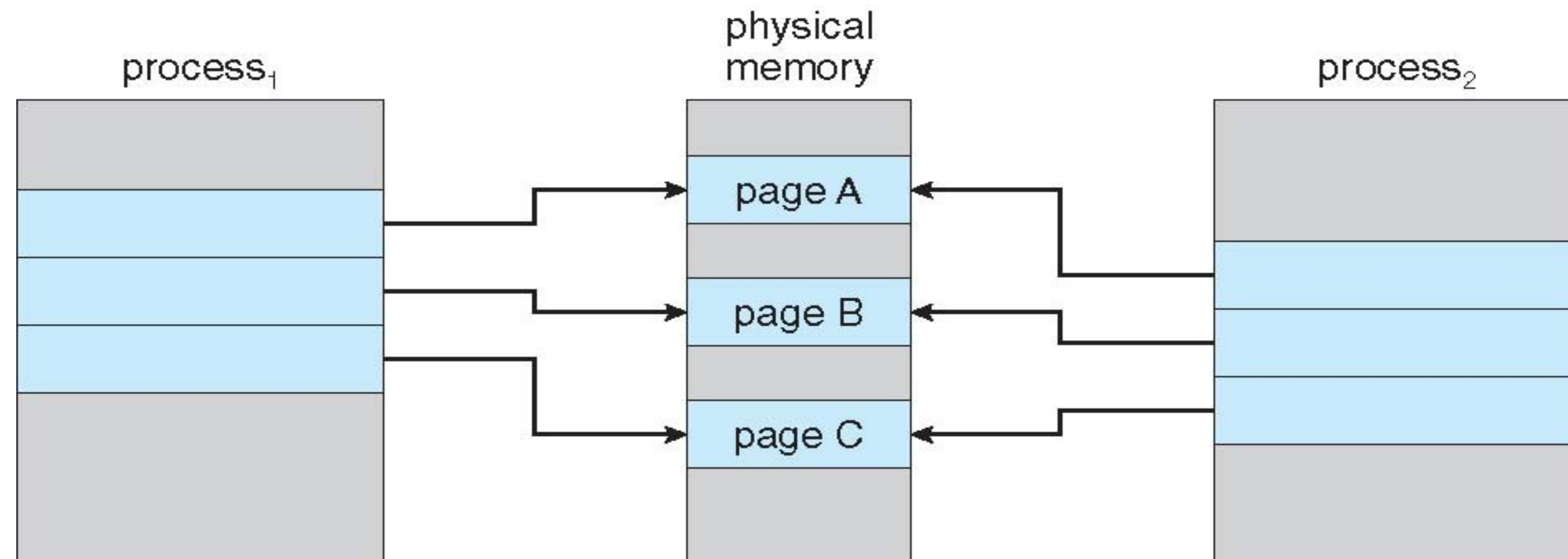
Copy-on-Write

- Copie paresseuse via la table des pages
- La copie partage la même mémoire physique que l'original
  - Pages marquées read-only
  - En cas de store, la page affectée est copiée
  - La table des pages est ajustée et marquée read-write
- Indispensable pour `fork`
  - nous évite de copier un processus entier seulement pour que ce processus soit complètement remplacé immédiatement par un `exec`

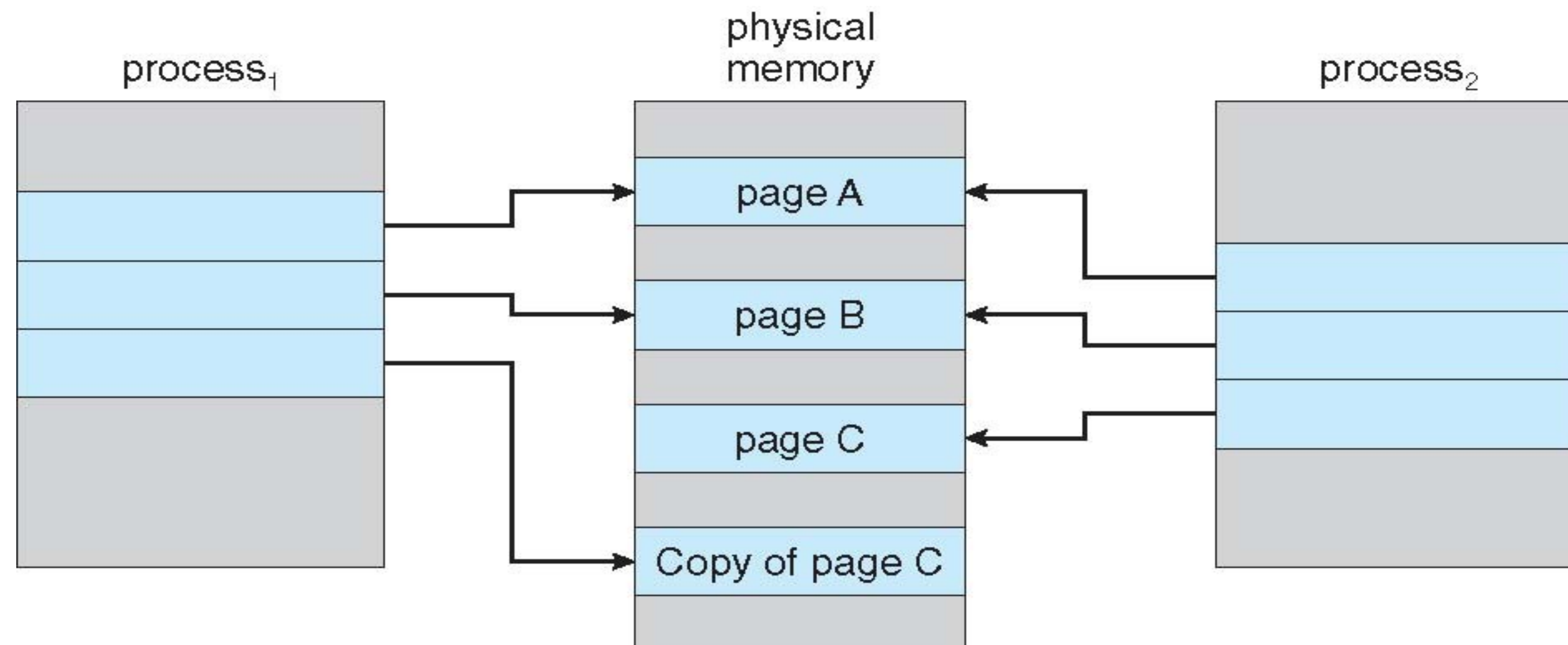
Silberschatz, Galvin and Gagne ©2013, Monnier ©2016, Paull ©2018

Copy-on-write

avant mod
le page C
par processus 1



après mod



Silberschatz, Galvin and Gagne ©2013, Monnier ©2016, Paull ©2018

Menu

- Préliminaires
- Pagination à la demande
- **Remplacement de pages**
- Allocation de frames
- Thrashing
- Fichiers memory-mapped
- Allocation de mémoire noyau
- Autre considérations

DIRO

Silberschatz, Galvin and Gagne ©2013, Monnier ©2016, Paull ©2018

Plus de *frame* libre

- Que faire s'il n'y a plus de *frame* libre:
  - Tuer un ou des processus
  - Swap un processus complètement
  - Evincer une page seulement (page replacement)
- Pour évincer une page, il est très important de trouver un bon candidat

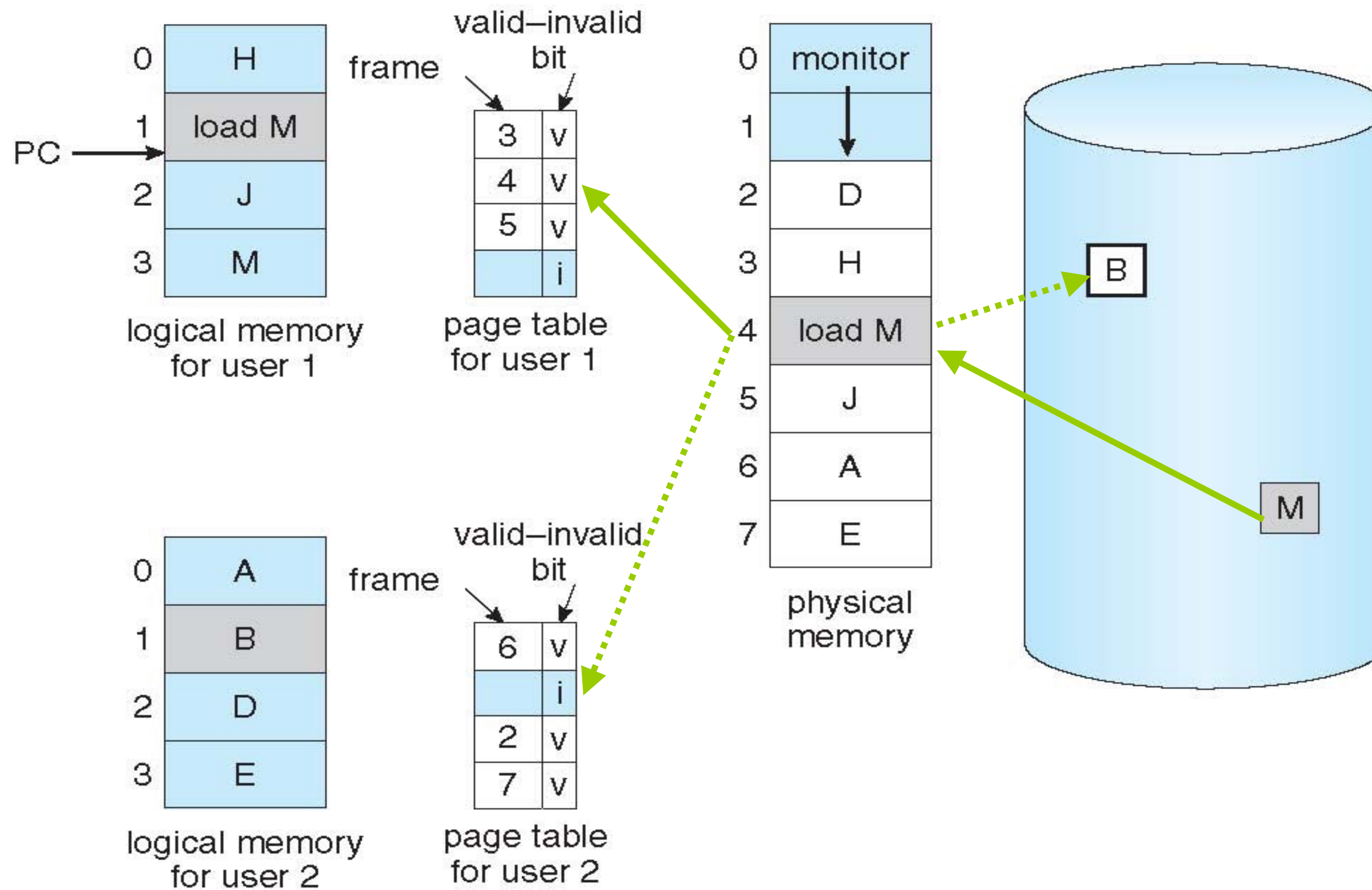
Silberschatz, Galvin and Gagne ©2013, Monnier ©2016, Paull ©2018

Remplacement de page

- Le bit modify (dirty) indique si une page / frame a été modifiée en lui écrivant
  - utilisé pour réduire le temps de transfert des pages - seules les pages modifiées sont réécrites sur le disque (stratégie similaire pour les pages en read-only)
- Le remplacement de page termine la séparation entre la mémoire logique et la mémoire physique - une mémoire virtuelle importante peut être fournie sur une mémoire physique plus petite

Silberschatz, Galvin and Gagne ©2013, Monnier ©2016, Paull ©2018

Remplacement de page



Silberschatz, Galvin and Gagne ©2013, Monnier ©2016, Paull ©2018

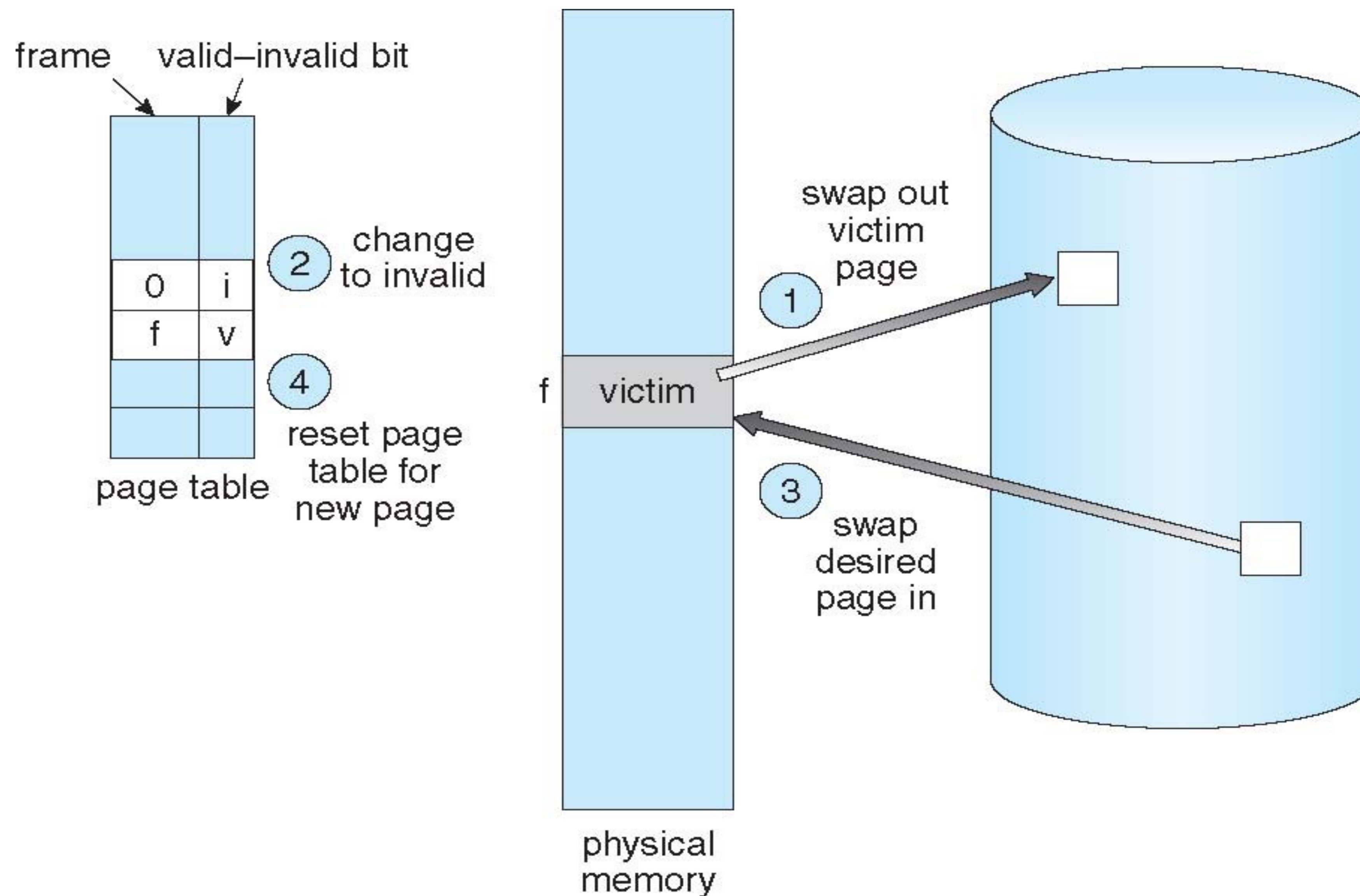
Remplacement de page

1. Trouver l'emplacement de la page souhaitée sur le disque
2. Trouver un cadre gratuit:
   1. S'il y a un cadre libre, utilisez-le
   2. S'il n'y a pas de trame libre, utilisez un algorithme de remplacement de page pour sélectionner une **victime frame**,
   3. écrire la *victim frame* sur le disque si elle est marquée comme “dirty”, et mettre à jour les tableaux de pages et de frames
3. Apportez la page désirée dans le cadre (nouvellement) libéré; mettre à jour les tables de pages et de frames
4. Continuez le processus en redémarrant l'instruction qui a provoqué le piège

- Notez maintenant potentiellement les transferts de deux pages pour la faute de page - augmentation EAT

Silberschatz, Galvin and Gagne ©2013, Monnier ©2016, Paull ©2018

Remplacement de page



On peut utilisé un “modify bit” (dirty bit) pour savoir si une page a été modifiée et peut-être évite le swap #1

Silberschatz, Galvin and Gagne ©2013, Monnier ©2016, Paull ©2018

Comment choisir la page à échanger?

- Une algorithmes de remplacement de page veut minimiser le nombre de *page faults* futures
  - Soit globalement
  - Soit par processus ou utilisateur ou groupe de processus

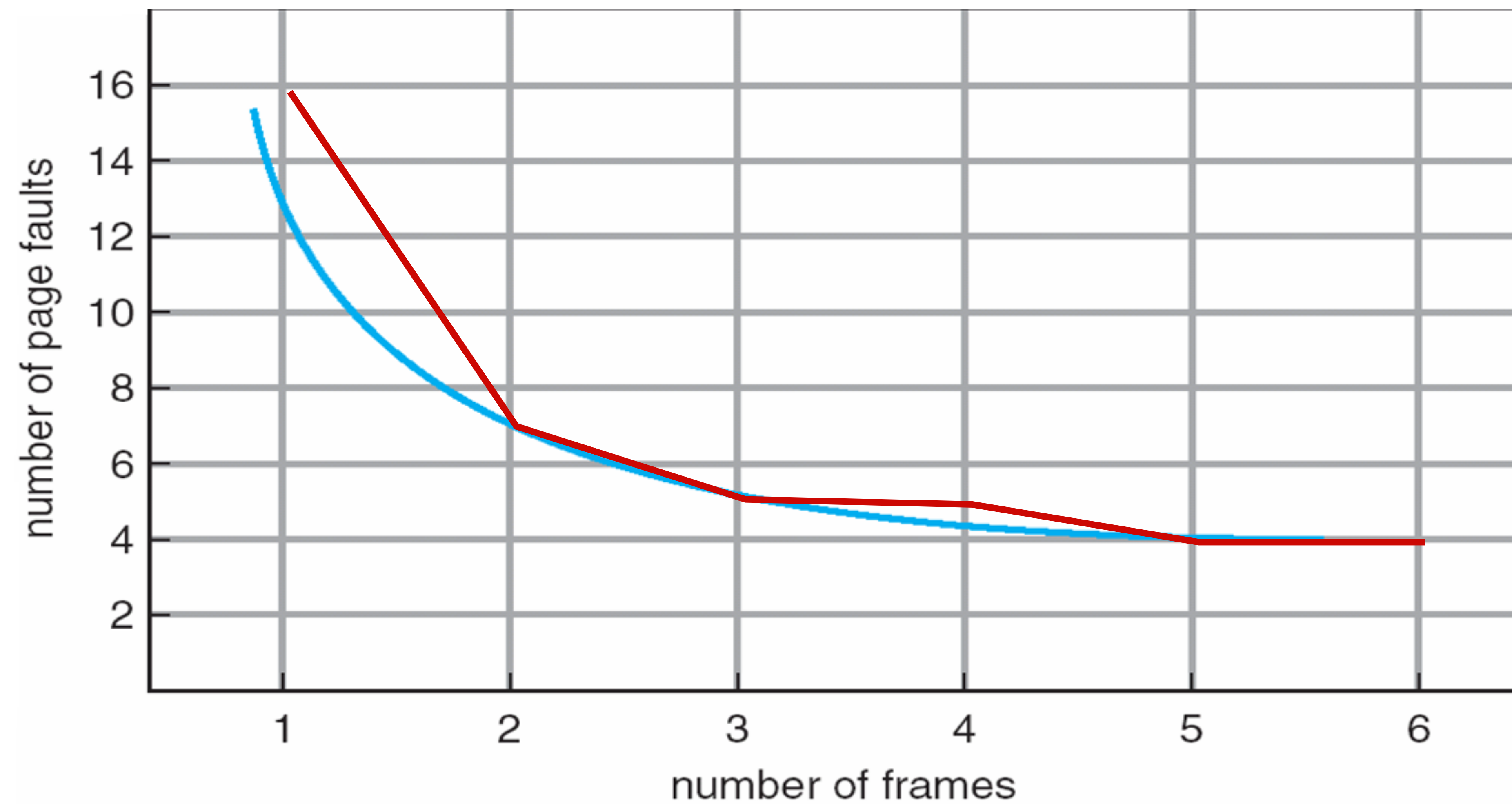
- Nous allons évalué chaque algorithms sur une séquence de pages "symbolique":

  7,0,1,2,0,3,0,4,2,3,0,3,0,3,2,1,2,0,1,7,0,1

Silberschatz, Galvin and Gagne ©2013, Monnier ©2016, Paull ©2018

Pages faults vs. nombre de frames

En générale, nous prévoyons:



Silberschatz, Galvin and Gagne ©2013, Monnier ©2016, Paull ©2018

Remplacement par random

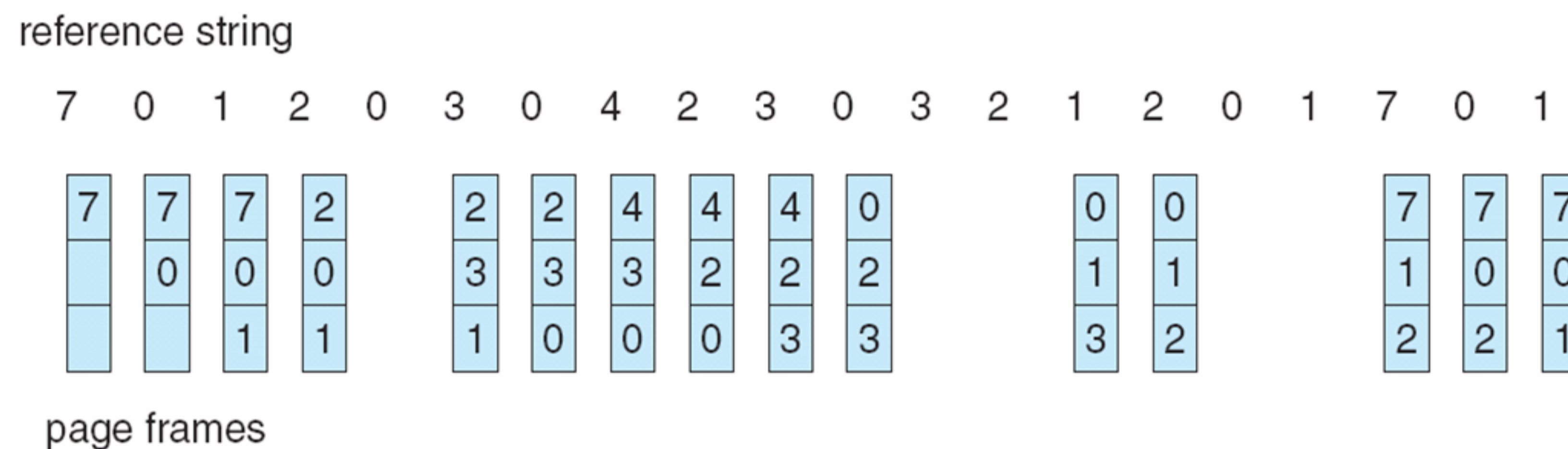
- On choisit une frame au hasard
- Facile à implémenter
- Pas très bon en générale
- Mais, dans le pire des cas n'est si pire que les autres

DIRO

Silberschatz, Galvin and Gagne ©2013, Monnier ©2016, Paull ©2018

Remplacement par First-In-First-Out (FIFO)

- Ex avec 3 frames:



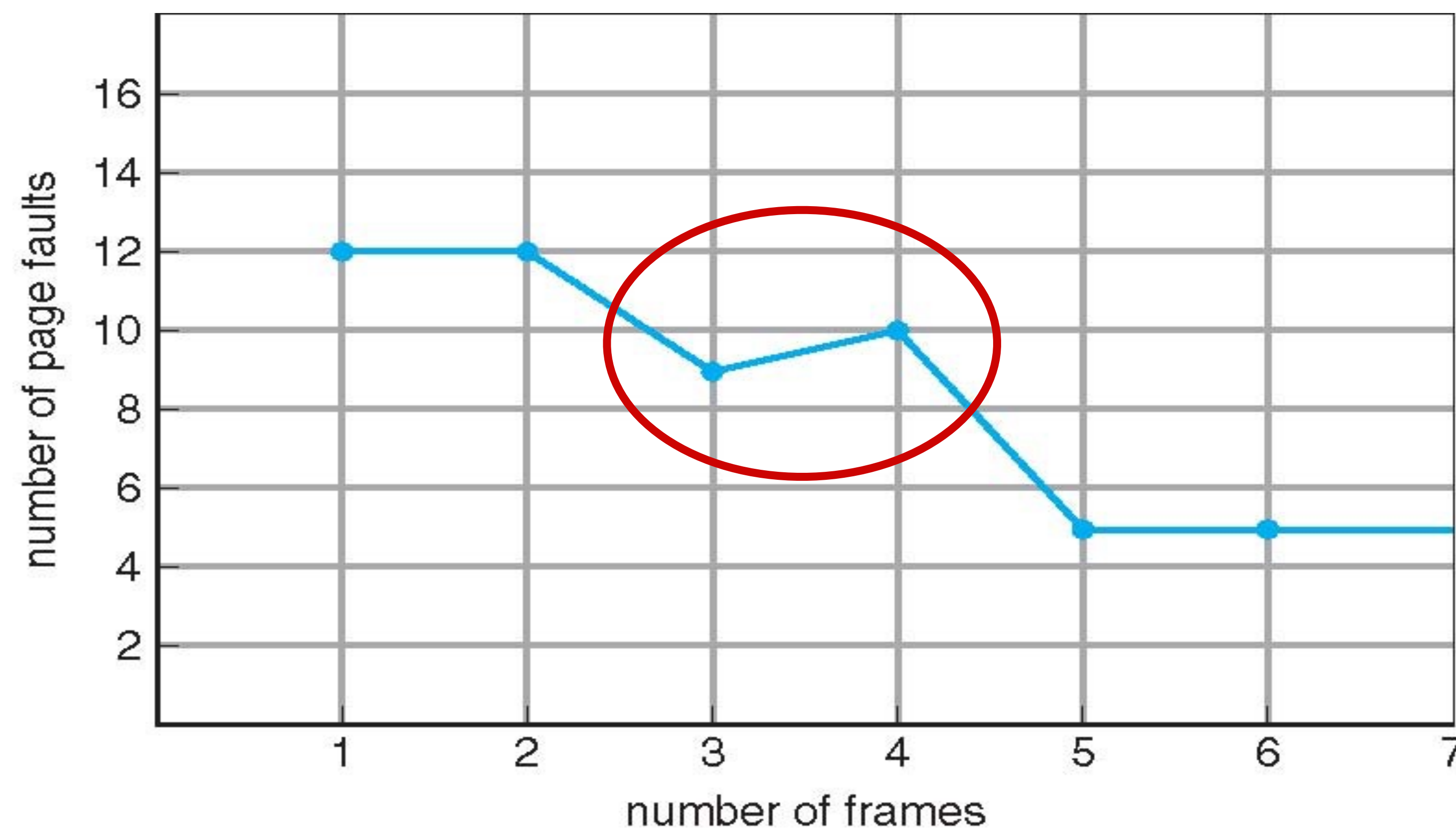
- 15 page faults

- Facile à implémenter mais pas toujours mais ce n'est pas toujours une bonne idée puisque les pages fréquemment utilisées peuvent être remplacées plusieurs fois

Silberschatz, Galvin and Gagne ©2013, Monnier ©2016, Paull ©2018

Anomalie de Belady

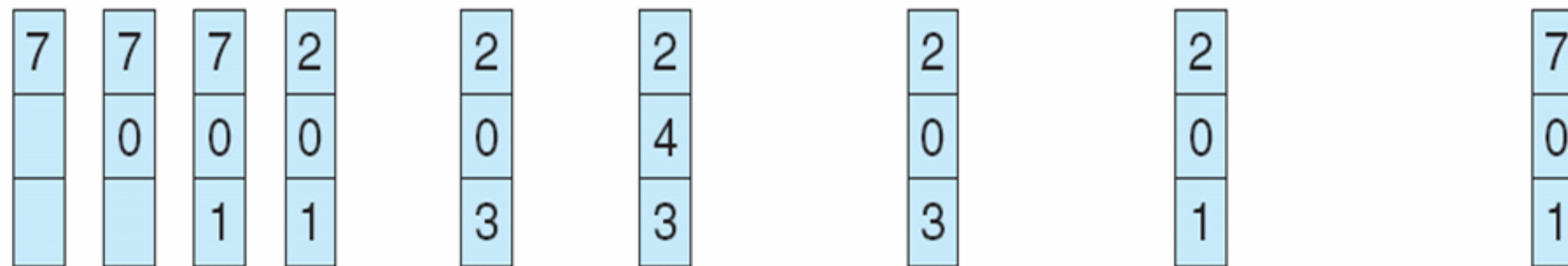
- La performance dépend de la chaîne de référence, considère:
  1,2,3,4,1,2,5,1,2,3,4,5
  - plusieurs frames entraîne une augmentation page fault



Silberschatz, Galvin and Gagne ©2013, Monnier ©2016, Paull ©2018

Remplacement optimal

- Remplacer la page qui restera inutilisée plus longtemps

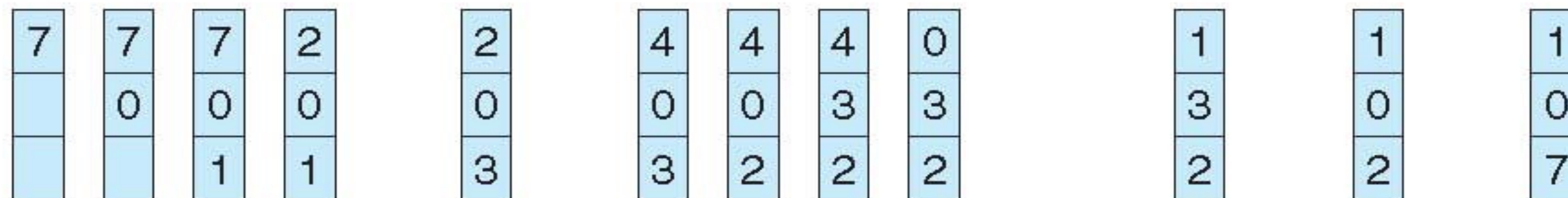


- Seulement 9 page faults
- Malheureusement, il faut connaître le futur
- Bonne référence pour comparer les autres algorithmes

Silberschatz, Galvin and Gagne ©2013, Monnier ©2016, Paull ©2018

Remplacement par Least Recently Used (LRU)

- Remplace la frame inutilisée depuis le plus de temps



- 12 page faults dans notre exemple
- Généralement un des meilleurs algorithms
- LRU et OPT sont des algorithmes de piles - ne souffre pas de l'anomalie de Baladie
- LRU a besoin de plus de support matériel - besoin d'implémenter un compteur ou une pile

Silberschatz, Galvin and Gagne ©2013, Monnier ©2016, Paull ©2018

Talon d'Achille de LRU

- Longs accès séquentiels
- Sans répétition: le futur ne ressemble pas du tout au passé
- Avec répétition: accès à N+1 pages
  - Lorsqu'on arrive à N+1, on évince 0
  - Lorsqu'on arrive à 0, on évince 1
  - …
- Random fonctionnerait beaucoup mieux (~1 page fault par boucle au lieu de N)

Silberschatz, Galvin and Gagne ©2013, Monnier ©2016, Paull ©2018

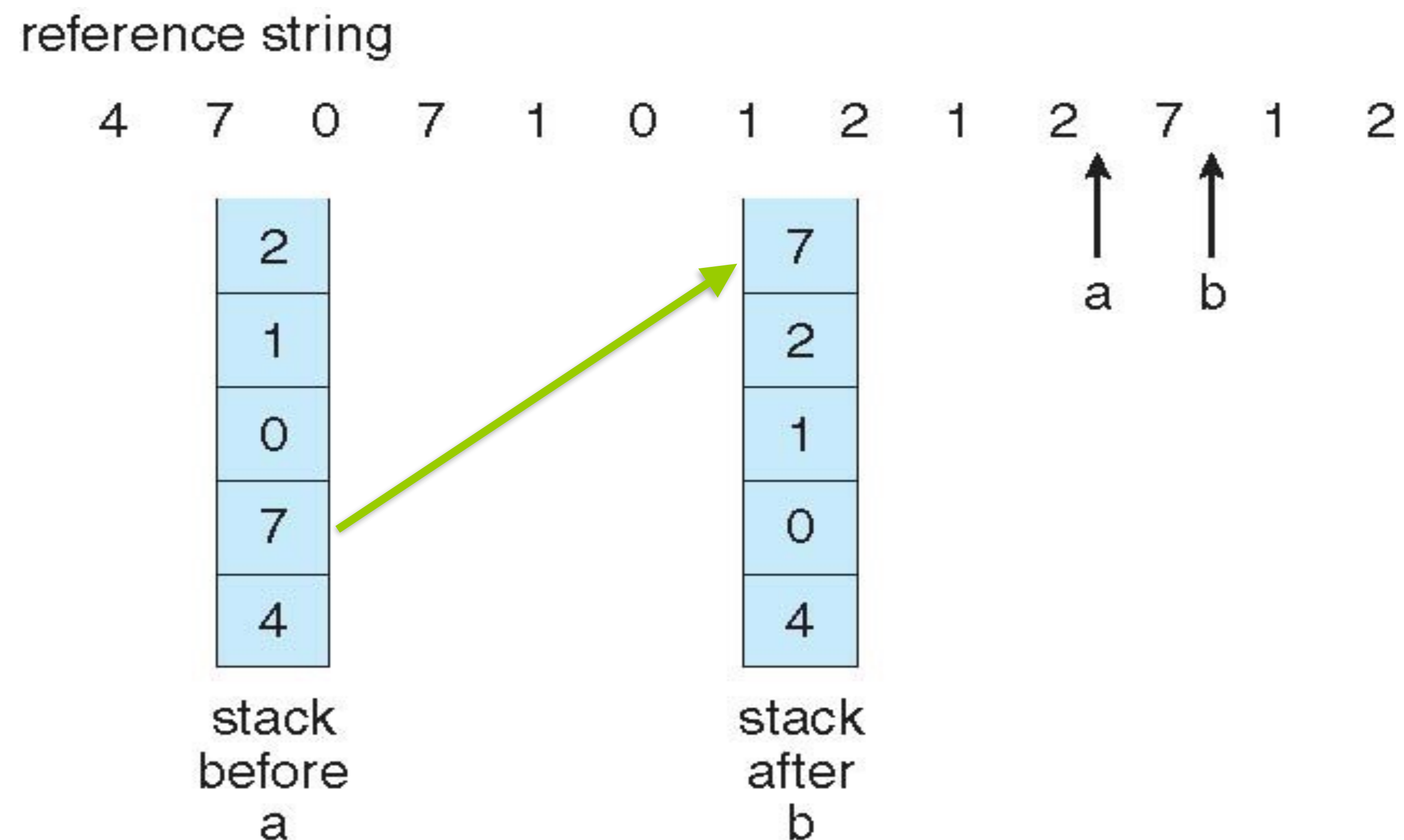
Implementation de LRU avec des compteurs

- Chaque frame garde l'heure du dernier accès
- Le compteur est incrémentée à chaque accès
- Lors du remplacement, cherche la frame avec compteurs plus hauts
- Coût:
  - recherche dans la table nécessaire
  - doit écrire contre la mémoire pour chaque accès
  - être conscient du débordement d'horloge

Silberschatz, Galvin and Gagne ©2013, Monnier ©2016, Paull ©2018

LRU Algorithm - Pile

- Garde une list ordonnée de toutes les frames
- À chaque accès, déplace la frame en première position
- Lors du remplacement, prend la dernière frame de la liste
- Pas de recherche au moment du remplacement mais prend plus de temps pour mettre à jour la pile



Silberschatz, Galvin and Gagne ©2013, Monnier ©2016, Paull ©2018

Approximation de LRU: reference bit

- LRU requiert trop de travail à chaque accès
- Certains CPU donnent une approximation
- Un “reference bit” dans la table de page
  - Lors de l’accès à une page, le CPU met ce bit à 1
  - Périodiquement, le SE collect tous ces bits et les remet à 0
  - Le SE ne peut plus distinguer l’ordre d’accès dans une même période
    - ✓ Parmis les frames d’une même période on peut utiliser le FIFO
- Plus de “reference bits”
  - Set le bit le plus à gauche sur un accès, puis effectuez périodiquement un right-shift des bits
  - Le bit 1 le plus à gauche donne une estimation de la dernière fois utilisée

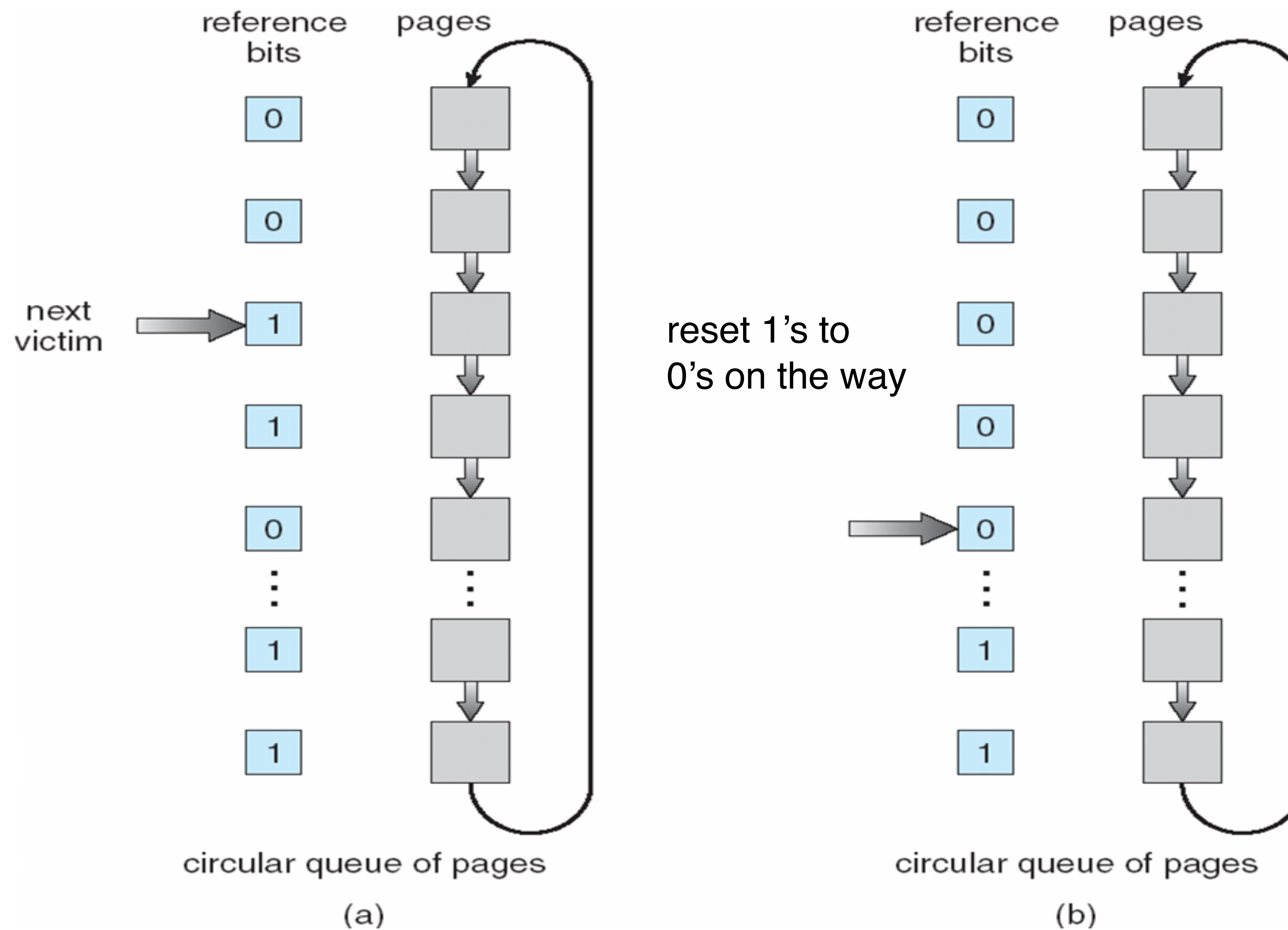
Silberschatz, Galvin and Gagne ©2013, Monnier ©2016, Paull ©2018

Approximation de LRU: Deuxième Chance

- Comme FIFO, prend la frame suivante, mais vérifie son “reference bit”
  - Si reference bit est 0, alors remplace cette page
  - Si reference bit est 1, alors remet reference à 0 et passe à la frame suivante
- Dégénère à FIFO si les reference bits sont tous 0 (ou tous 1)
- Peut également coupler les bits de modification et de référence
  - (0,0) non référencé ni modifié: meilleure victime
  - (0,1) non référencé mais modifié: pas génial car il faudra l'écrire sur le disque
  - (1,0) référencé mais non modifié: pourrait être réutilisé
  - (1,1) référencé et modifié: la pire victime

Silberschatz, Galvin and Gagne ©2013, Monnier ©2016, Paull ©2018

Algorithme deuxième chance



Silberschatz, Galvin and Gagne ©2013, Monnier ©2016, Paull ©2018

Algorithmes de comptage

- Gardez un compteur du nombre de références qui ont été faites à chaque page
- **Algorithme LFU** (least frequently used): remplace la page par le plus petit nombre
  - mais si une page est référencée beaucoup au début, elle restera toujours là
  - pourrait utiliser une stratégie de décalage vers la droite pour diminuer le compteur
- **Algorithme MFU** (most frequently used): remplace la page par le plus grand nombre
  - sur la base de l'argument que la page avec le plus petit nombre a probablement été introduit et n'a pas encore été utilisé
- Ni l'un ni l'autre de ces algorithmes ne se comparent bien à l'OPT, ils sont coûteux et peu utilisés

Silberschatz, Galvin and Gagne ©2013, Monnier ©2016, Paull ©2018

Algorithmes de page-buffering

- Découpler les opérations pour répondre plus vite à un page fault
- Garder les frames libres
  - Choisir à l'avance les victimes futures
  - Si elle est *dirty* (modifié), écrire le contenu dans le disque
  - Marque invalide. Mais si accédé, on peut la ressusciter à bon prix
- Ne pas laisser les pages *dirty* trop longtemps
  - Quand le disque est inactif, écrire le contenu des pages *dirty*
  - Ainsi la page peut être évincée plus rapidement

Silberschatz, Galvin and Gagne ©2013, Monnier ©2016, Paull ©2018

Applications et remplacement de page

- Tous ces algorithmes ont des devinements d'SE concernant l'accès futur à la page
- Certaines applications ont une meilleure connaissance - par exemple, des bases de données
- Les applications peuvent provoquer une double mise en mémoire tampon
  - SE conserve la copie de la page en mémoire en tant que tampon d'E / S
  - L'application garde la page en mémoire pour son propre travail
- Le système d'exploitation peut donner un accès direct au disque, en s'écartant des applications
  - Mode **raw disk**
- Bypass tamponnage, verrouillage, etc.

Silberschatz, Galvin and Gagne ©2013, Monnier ©2016, Paull ©2018

Menu

- Préliminaires
- Pagination à la demande
- Remplacement de pages
- **Allocation de frames**
- Thrashing
- Fichiers memory-mapped
- Allocation de mémoire noyau
- Autre considérations

Silberschatz, Galvin and Gagne ©2013, Monnier ©2016, Paull ©2018

Allocation de frames

- Q1: Laissons-nous un processus changer la page d'un autre?

Silberschatz, Galvin and Gagne ©2013, Monnier ©2016, Paull ©2018

Allocation de frames

- **Remplacement global** - le processus sélectionne une frame de remplacement parmi l'ensemble de toutes les frames; un processus peut prendre une frame d'un autre
  - Le temps d'exécution du processus peut varier considérablement d'une exécution à l'autre
  - Mais un plus grand débit, donc plus commun
- **Remplacement local** - chaque processus sélectionne uniquement son propre ensemble de frames attribuées
  - Performances par processus plus cohérentes
  - Mais peut-être que la mémoire sous-utilisée
- Nous pouvons également autoriser un processus à prendre une frame d'un autre processus s'il a une priorité inférieure

Silberschatz, Galvin and Gagne ©2013, Monnier ©2016, Paull ©2018

Allocation de frames

- Q1: Laissons-nous un processus changer la page d'un autre?
- Q2: Combien de frames allouer à chaque processus?

Silberschatz, Galvin and Gagne ©2013, Monnier ©2016, Paull ©2018

Allocation de frames

- Allocation égale - Par exemple, s'il y a 100 images (après avoir alloué des images pour le système d'exploitation) et 5 processus, donner à chaque processus 20 images
  - Gardez-en en tant que pool de mémoire tampon de frame libre
  - Mais un petit processus n'aura pas besoin de frames alors qu'un grand ne fonctionnera pas bien

- Allocation proportionnelle - Allouer en fonction de la taille du processus

s_i: taille du process i
S: taille total
m: nombre de frames
a_i: allocation de frames a processus i

$$m = 64$$
$$s_1 = 10$$
$$s_2 = 127$$
$$a_1 = \frac{10}{137} \times 64 \approx 5$$
$$a_2 = \frac{127}{137} \times 64 \approx 59$$

- Dynamique en degré de multiprogrammation, les tailles de processus changent
- Peut aussi utiliser la proportion selon différentes priorités

Silberschatz, Galvin and Gagne ©2013, Monnier ©2016, Paull ©2018

Non-Uniform Memory Access (NUMA)

- Jusqu'à présent, nous considérons que tous les accès mémoire prennent un temps égal
- De nombreux systèmes sont NUMA - la vitesse d'accès à la mémoire varie
  - E.g., plusieurs noeuds (CPU + mémoire) interconnectés
- Meilleure performance avec allocation *proche*
  - Et en modifiant le planificateur pour planifier le thread sur la même carte système lorsque cela est possible
- Parmi tous les frames qui sont libres, choisissez celui qui est le plus proche

Silberschatz, Galvin and Gagne ©2013, Monnier ©2016, Paull ©2018

Menu

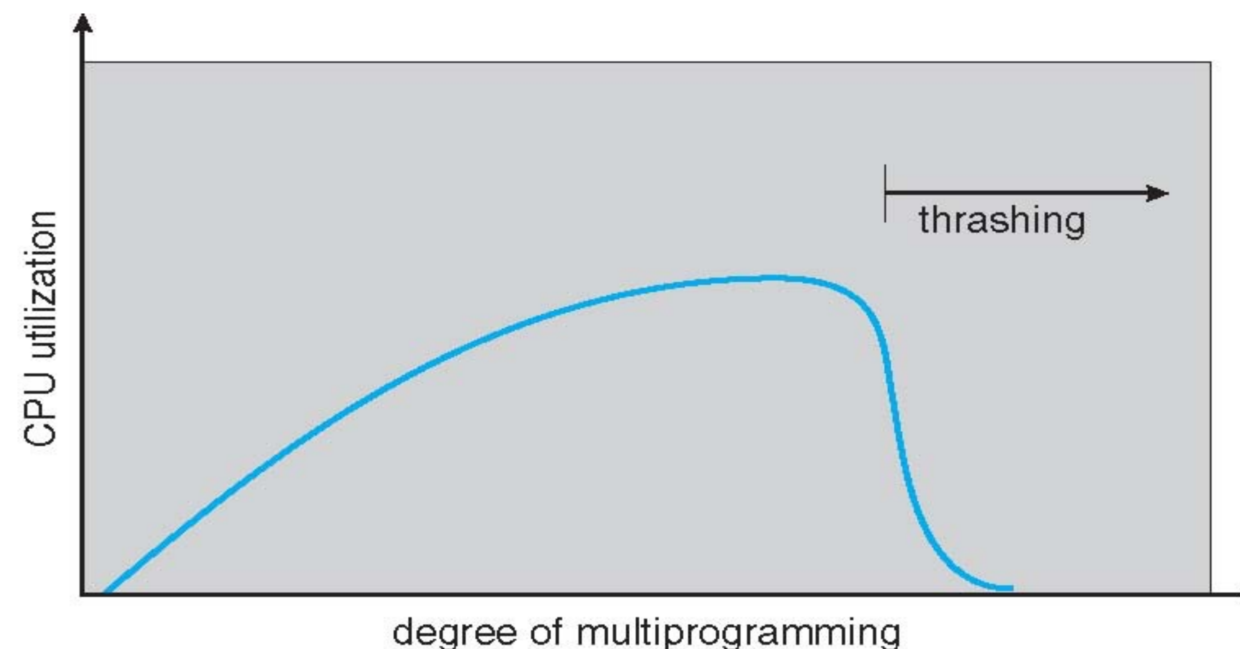
- Préliminaires
- Pagination à la demande
- Remplacement de pages
- Allocation de frames
- **Thrashing**
- Fichiers memory-mapped
- Allocation de mémoire noyau
- Autre considérations

DIRO

Silberschatz, Galvin and Gagne ©2013, Monnier ©2016, Paull ©2018

Thrashing

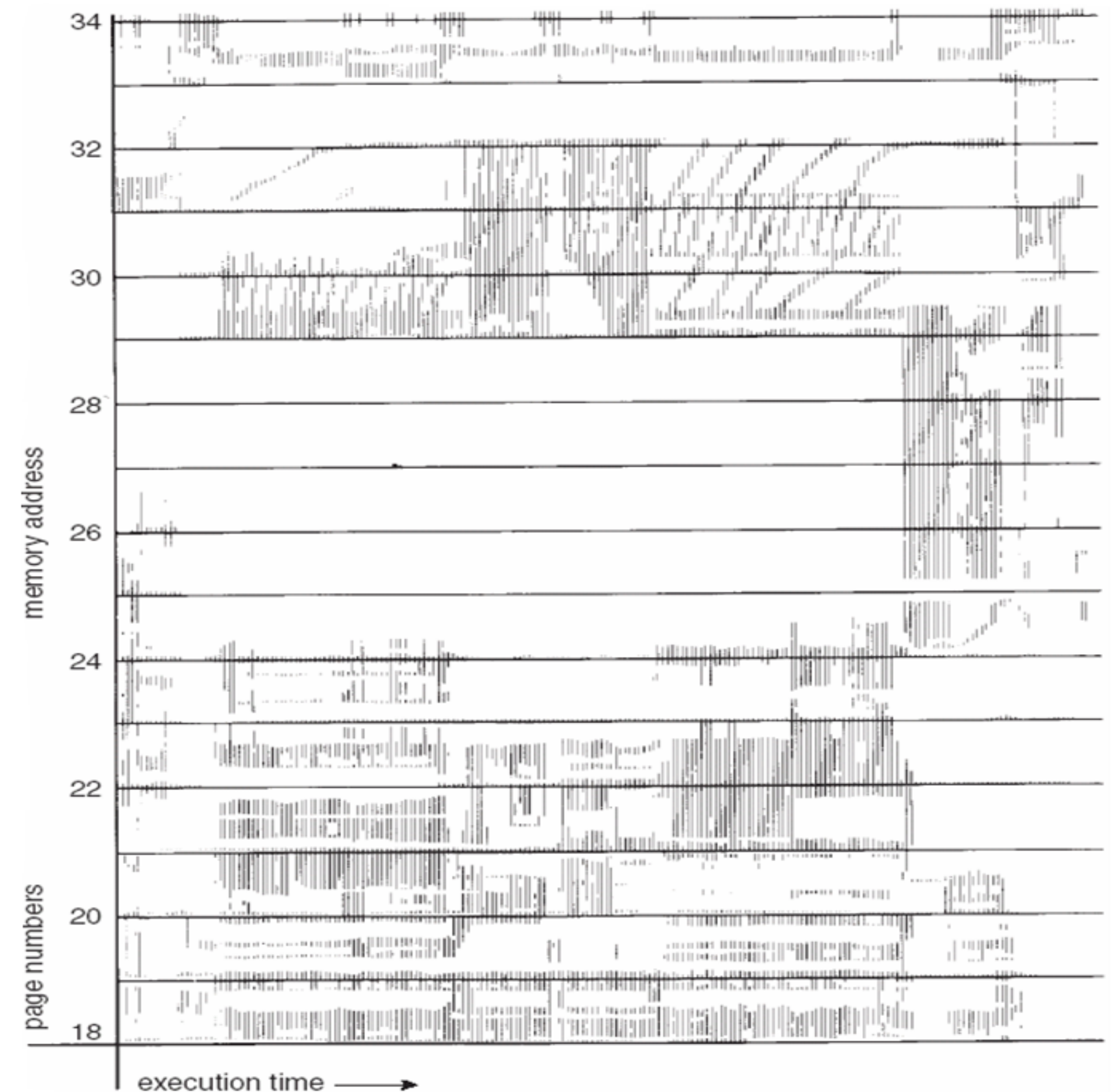
- Si un processus n'a pas de pages "suffisantes", le taux de défaillance de la page peut devenir très élevé
  - page fault pour obtenir la page
  - Remplacer le cadre existant
  - Mais a besoin rapidement du cadre remplacé
  - Cela mène à:
    - Faible utilisation du processeur
    - OS pense qu'il devrait augmenter le degré de multiprogrammation
    - Un autre processus ajouté au système
- Thrashing ≡ un processus est occupé à permuter des pages (c'est-à-dire, passe plus de temps à pagination qu'à s'exécuter)



Silberschatz, Galvin and Gagne ©2013, Monnier ©2016, Paull ©2018

Pagination et thrashing

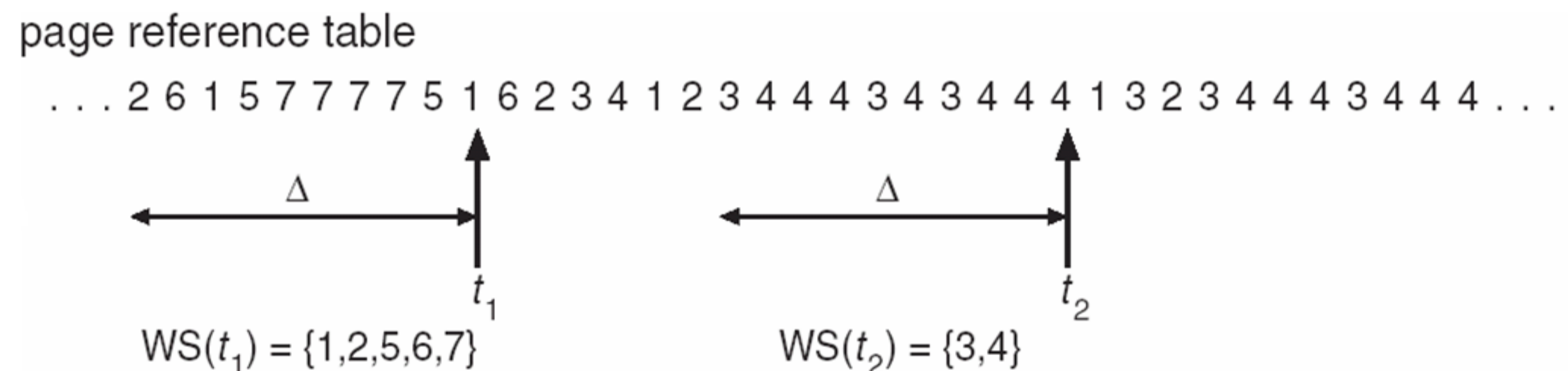
- Pagination marche parce que
  - Exécution migre de localité à localité
  - Localité peuvent se chevaucher
  - On peut revenir à une localité
- Thrashing
  - Taille de localité > mémoire
  - Peut se limiter aux processus coupables



Silberschatz, Galvin and Gagne ©2013, Monnier ©2016, Paull ©2018

Modèle du working-set

- Δ ≡ fenêtre du *working-set:* un certain intervalle de temps (e.g. 10,000 instructions)
  - Un Δ trop petit ne couvre pas toute la localité
  - Un Δ trop grand couvre plusieurs localités (trop pessimiste)
  - Δ = ∞, program en complet
  - Δ dépend typiquement du temps d'accès au disque
- $\boldsymbol{WSS_i}$ (le working set du process $\boldsymbol{P_i}$) = Nombres de pages différents référencées par $\boldsymbol{P_i}$ pendant **Δ**



- $\boldsymbol{D} = \boldsymbol{\Sigma}\ \boldsymbol{WSS_i}$: Taille totale des localités courantes
  - Approximation of locality
- D > Mémoire => thrashing: il faut suspendre des processus
- D - Mémoire > N => peut charger un nouveau processus en mémoire

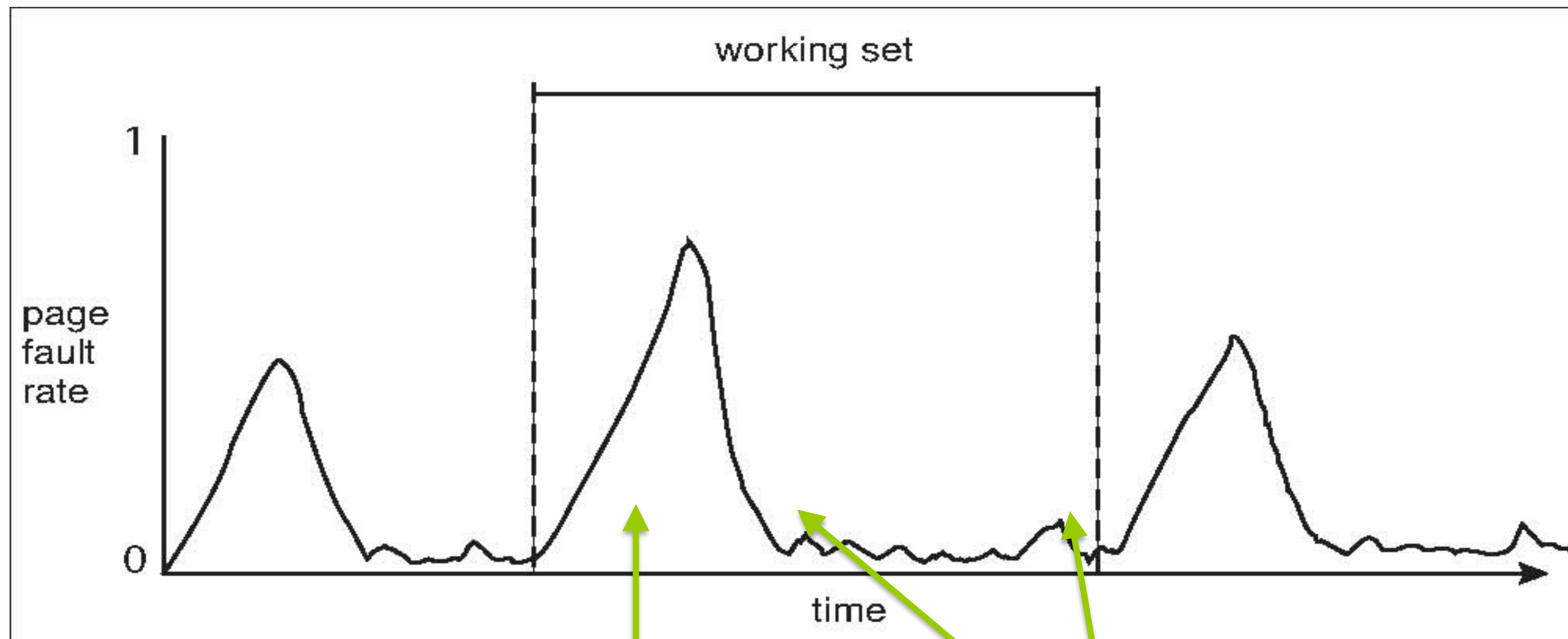
Silberschatz, Galvin and Gagne ©2013, Monnier ©2016, Paull ©2018

Garder la trace du working-set

- Approximatif avec compteur + un bit de référence
- Exemple: $\Delta = 10{,}000$
  - Gardez en mémoire 2 bits pour chaque page
  - Le compteur interrompt toutes les 5000 unités de temps
  - Chaque fois que le compteur s'interrompe, copiez tous les bits de référence en mémoire et remettez leur à 0

Silberschatz, Galvin and Gagne ©2013, Monnier ©2016, Paull ©2018

Working-set et fréquence de page-faults



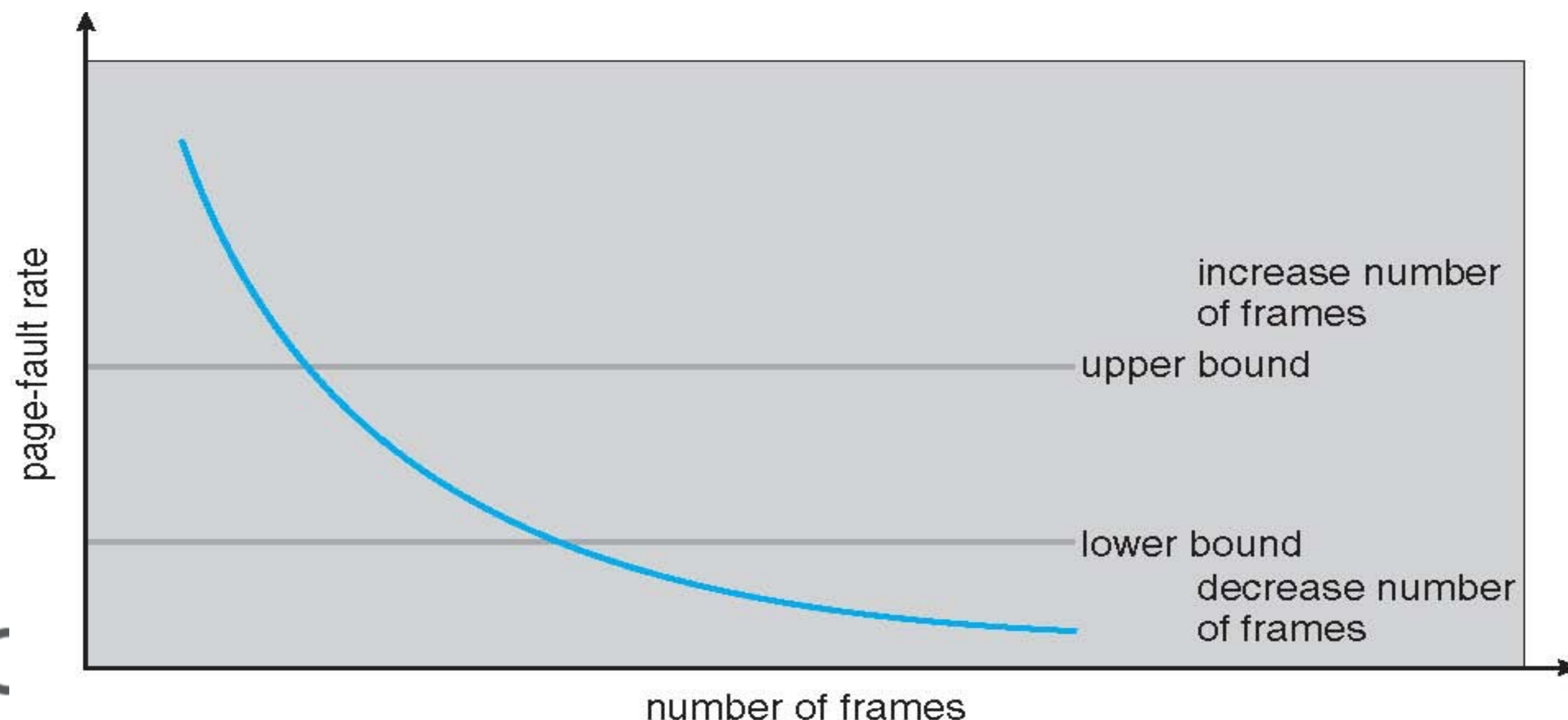
processus entre dans une nouvelle localité: taux de page fault augmente

processus fonctionne dans sa localité: taux de page fault inférieure

Silberschatz, Galvin and Gagne ©2013, Monnier ©2016, Paull ©2018

Fréquence de page-fault

- Autre approche, plus directe que working-set
- Décide d'une fréquence acceptable de page-faults
- Measure fréquence de page faults pour chaque processus
  - Fréquence trop basse: diminuer la mémoire allouée au processus
  - Fréquence trop élevée: augmenter la mémoire allouée
  - Fréquence trop élevée, mais plus de mémoire disponible: suspendre



Silberschatz, Galvin and Gagne ©2013, Monnier ©2016, Paull ©2018

Menu

- Préliminaires
- Pagination à la demande
- Remplacement de pages
- Allocation de frames
- Thrashing
- **Fichiers memory-mapped**
- Allocation de mémoire noyau
- Autre considérations

DIRO

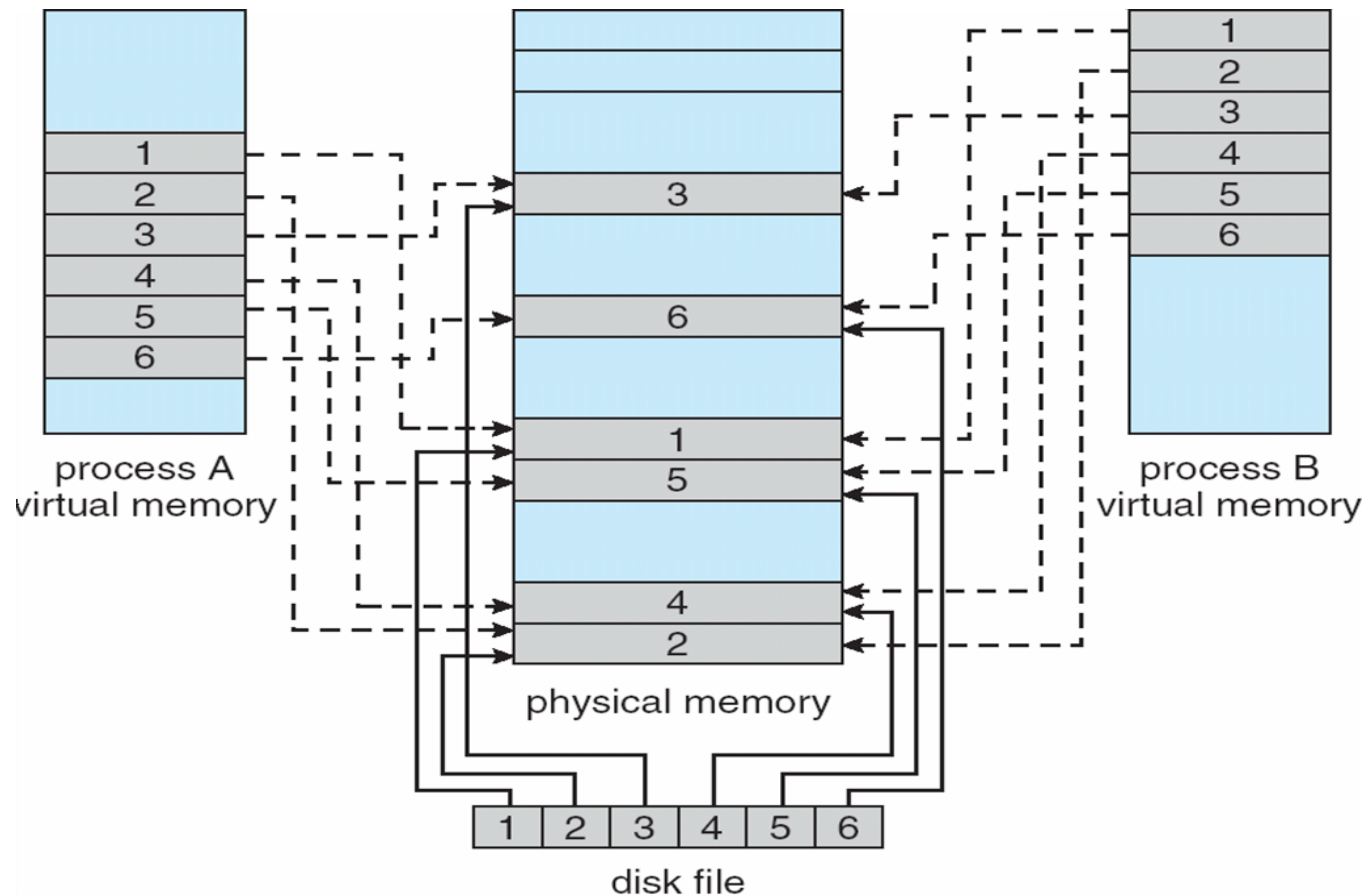
Silberschatz, Galvin and Gagne ©2013, Monnier ©2016, Paull ©2018

Fichiers memory-mapped

- Rendre un fichier accessible comme une partie de la mémoire
- Un fichier est initialement lu en utilisant la pagination sur demande
  - Une partie du fichier de la taille d'une page est lue du système de fichiers dans une page physique (frame)
  - Les lectures / écritures suivantes vers / depuis le fichier sont traitées comme des accès mémoire ordinaires
- Simplifie et accélère l'accès aux fichiers en générant des E/S de fichiers via la mémoire au lieu des appels système `read()` et `write()`
- Permet également à plusieurs processus de mapper le même fichier permettant de partager les pages en mémoire
- Les données écrites le rendent sur le disque:
  - Périodiquement et / ou à la `close()` (COW)
  - Par exemple, lorsque le pager analyse les pages sales

Silberschatz, Galvin and Gagne ©2013, Monnier ©2016, Paull ©2018

Exemple de memory mapping



Silberschatz, Galvin and Gagne ©2013, Monnier ©2016, Paull ©2018

Menu

- Préliminaires
- Pagination à la demande
- Remplacement de pages
- Allocation de frames
- Thrashing
- Fichiers memory-mapped
- **Allocation de mémoire noyau**
- Autre considérations

DIRO

Silberschatz, Galvin and Gagne ©2013, Monnier ©2016, Paull ©2018

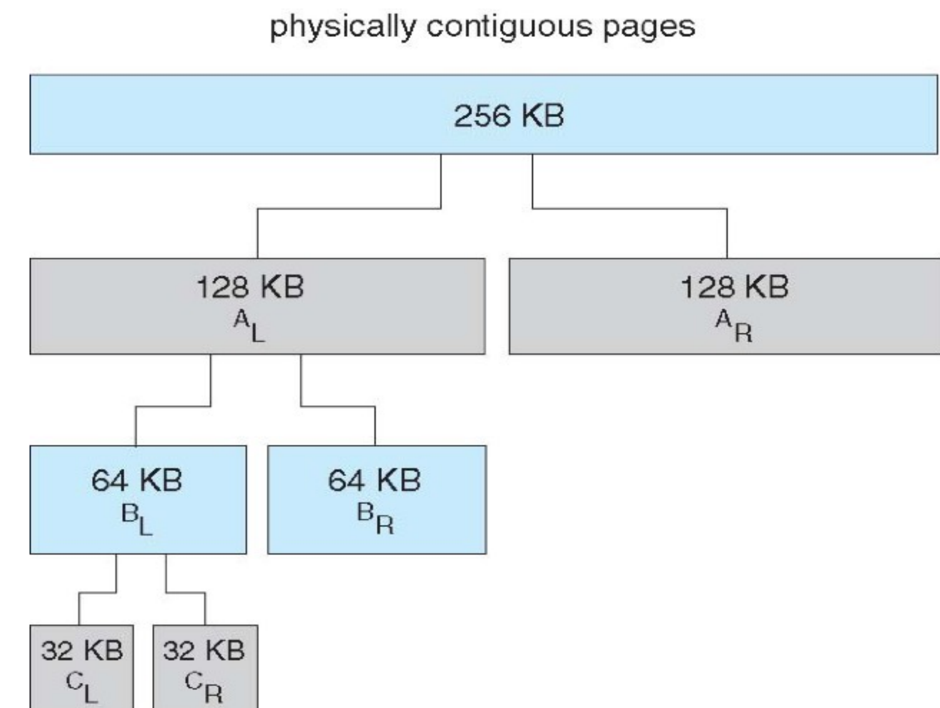
Allocation de mémoire noyau

- La mémoire du noyau est traitée différemment
- Mémoire du noyau gérée par objet, comme avec `malloc`
- Certaines parties doivent être contiguës

Silberschatz, Galvin and Gagne ©2013, Monnier ©2016, Paull ©2018

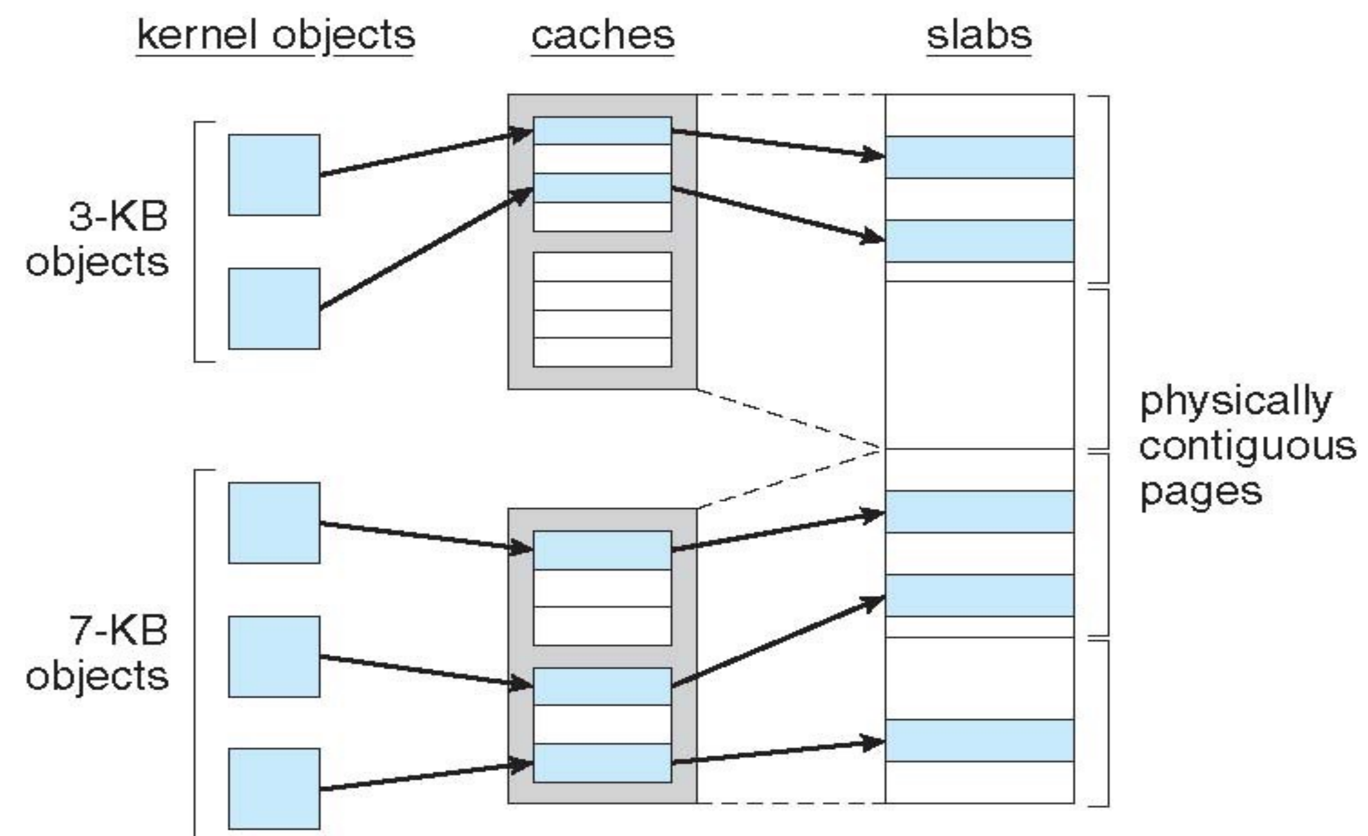
Allocation style *buddy*

- Utilise une zone mémoire contiguë de taille fixe
- Cette zone est divisée successivement par 2
- object < 1/2 taille de mémoire => divise par 2
- Par exemple, supposons qu'un bloc de 256 KB est disponible, le noyau demande 21 KB
  - Divise en A_L and A_R 128KB chaque
  - Choisi une, divise encore en B_L et B_R de 64KB
  - Encore en C_L and C_R de 32KB
    - ✓ utilise C_L ou C_R
- Avantage - fusionne rapide des morceaux inutilisés en plus gros morceaux
- Désavantage - fragmentation



Allocation style *Slab*

- Slab: une ou plusieurs pages, physiquement contiguës
- Cache: ensembles de slabs
- Chaque cache dédié à un type (ou une taille) d'objets
- Cache grandi par slab, divisée en objets (initialement libres)
- Si la slab est plein d'objets usagés, l'objet suivant est attribué à partir d'un slab vide
  - Si aucune slab vide, une nouvelle slab est allouée
- Peu de fragmentation, en pratique
- Efficace



Silberschatz, Galvin and Gagne ©2013, Monnier ©2016, Paull ©2018

Menu

- Préliminaires
- Pagination à la demande
- Remplacement de pages
- Allocation de frames
- Thrashing
- Fichiers memory-mapped
- Allocation de mémoire noyau
- **Autre considérations**

DIRO

Silberschatz, Galvin and Gagne ©2013, Monnier ©2016, Paull ©2018

Prepaging

- Pour réduire le grand nombre de défauts de page qui se produisent lors du (re) démarrage du processus
- “Prepage” tout ou partie des pages dont un processus aura besoin avant de les référencer
- Mais si les pages “Prepaged” ne sont pas utilisées, les E/S et la mémoire ont été gaspillées
- Supposons que les pages ***s*** sont prepaged et ***α*** des pages sont utilisées
  - Le coût de ***s x α*** enregistre-t-il des page faults plus ou moins importantes que le coût de la prepaging de ***s x (1- α)*** pages inutiles?
  - ***α*** proche de zéro ⇒ perte de prepaging
  - ***α*** près d'un ⇒ prépayer pourrait gagner

Silberschatz, Galvin and Gagne ©2013, Monnier ©2016, Paull ©2018

Taille de page

- La taille des pages est parfois imposée, mais pas touhours
- Facteurs de choix:
  - Fragmentation: mieux vaut des petites pages
  - Taille de la table: mieux vaut des grosses pages
  - Efficacité du TLB: mieux vaut des grosses pages
  - Localité: mieux vaut des petits pages
  - Résolution: mieux vaut des petites pages
  - Coût d'E/S et page faults: mieux vaut de grosses pages
- En générale, les tailles devraient augmenter peu à peu

Silberschatz, Galvin and Gagne ©2013, Monnier ©2016, Paull ©2018

TLB Reach

- TLB reach - quantité de mémoire accessible depuis le TLB
- TLB reach = (taille TLB) x (taille de page)
- Idéalement, le working set de chaque processus est stocké dans le TLB

Silberschatz, Galvin and Gagne ©2013, Monnier ©2016, Paull ©2018

Géré la localité

```
int data[128][128];
```

Quel accès choisir:

```
for (i = 0; i < 128; i++)
    for (j = 0; j < 128; j++)
        data[i, j] = 0;
```

ou:

```
for (j = 0; j < 128; j++)
    for (i = 0; i < 128; i++)
        data[i, j] = 0;
```

Beaucoup de manières d'influencer la performance d'un programme

Silberschatz, Galvin and Gagne ©2013, Monnier ©2016, Paull ©2018

Sommaire

- La mémoire virtuelle nous permet de mapper un grand espace d'adressage logique sur une plus petite mémoire physique
- Dans la pagination au demande pure, une page n'est pas mise en mémoire jusqu'à ce qu'elle soit nécessaire
- Augmente le degré de multiprogrammation car plus de processus peuvent être partiellement chargés simultanément dans la mémoire
- Si nous manquons de frames libres, nous devons utiliser un algorithme pour choisir quelle image remplacer
- Il y a aussi la question du nombre de frames qui devraient être allouées à chaque processus
- Si un processus passe plus de temps à réparer les fautes de page que de l'exécuter c'est le thrashing
- La mémoire du noyau est allouée différemment de la mémoire utilisateur

Silberschatz, Galvin and Gagne ©2013, Monnier ©2016, Paull ©2018