M2-SEM

Concepts avancés d'architecture

Cours 3.2

Gestion Mémoire Virtuelle

Année 2020-2021

Pr. R. BOUDOUR

Mémoire virtuelle

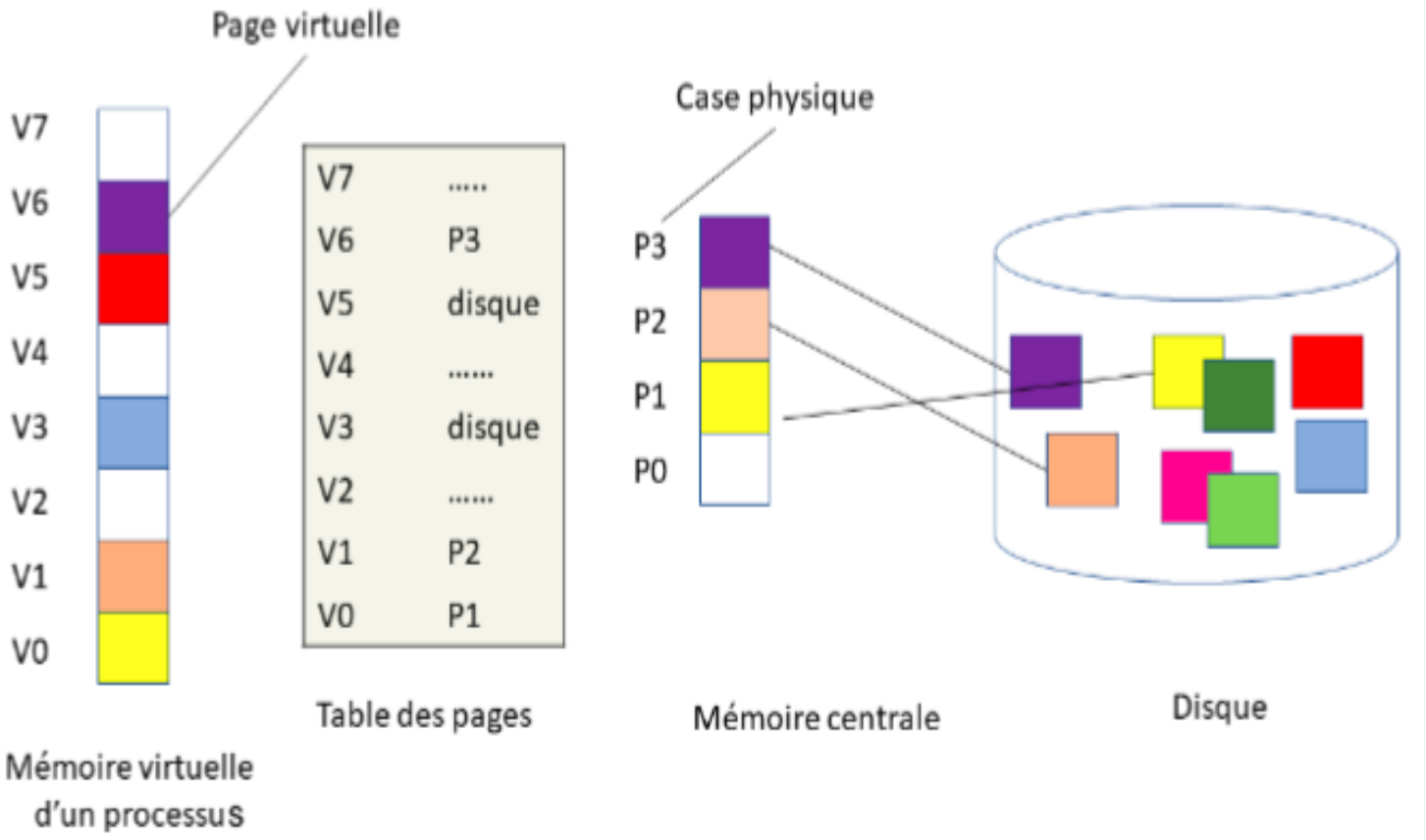
- **Constations :**
  - Mémoire physique coûteuse.
  - Mémoire secondaire (disques, mémoire étendue, ...) peu coûteuse.
  - Programmes gourmands en mémoire et qui ne "tiennent pas" toujours en RAM.

Conséquence : Utiliser la mémoire secondaire "comme" mémoire RAM.

Définitions

- Mémoire physique :
  - Ensemble des emplacements de la mémoire centrale
  - physiquement présents dans l'ordinateur.
- Mémoire virtuelle :
  - support de l'ensemble des informations potentiellement accessibles
  - Ensemble des emplacements dont l'adresse peut être engendrée par le processeur.

Composants de MV



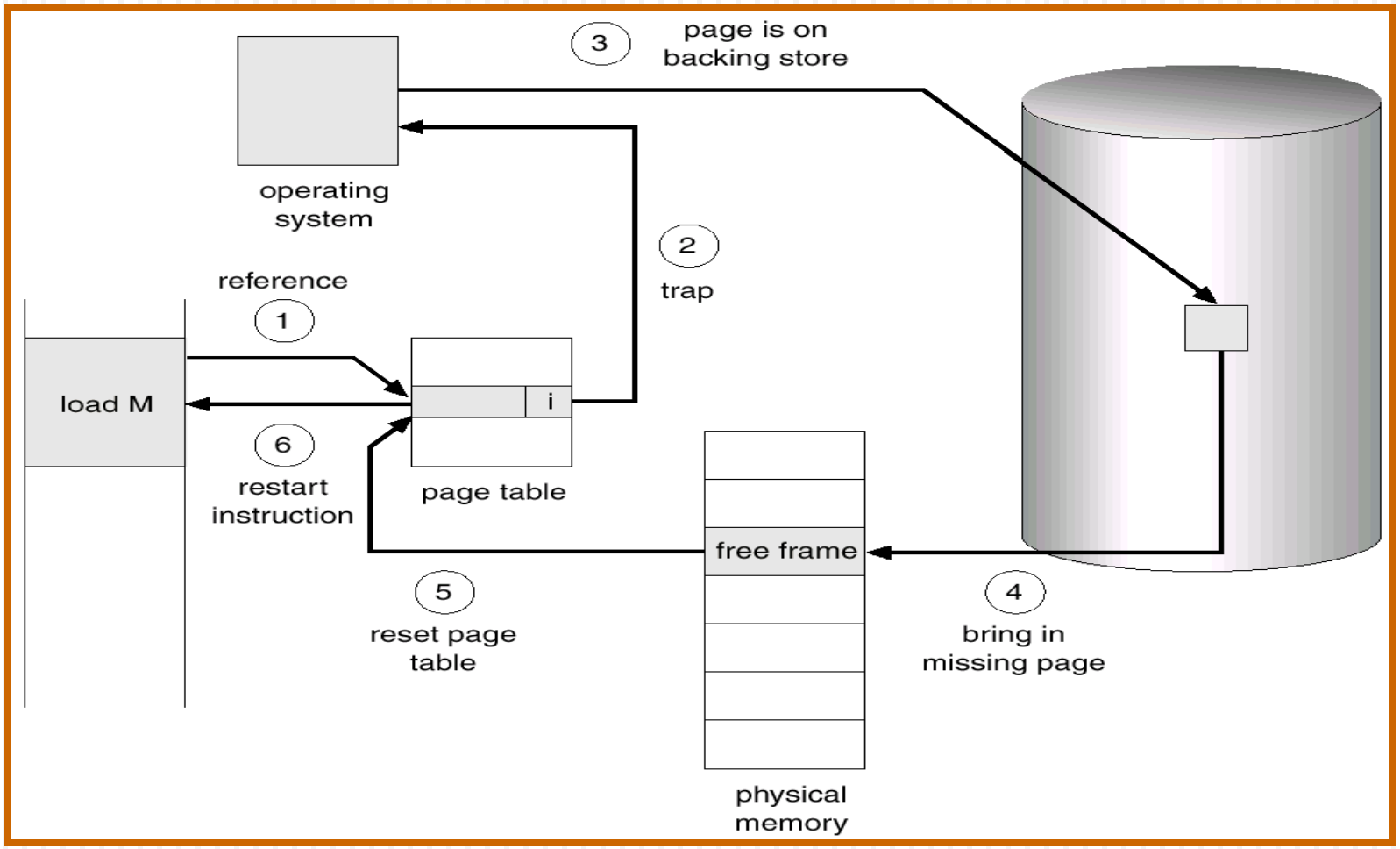
Mécanisme de gestion

- L'exécution des programmes engendre des adresses dites virtuelles.
- Le mécanisme de gestion assure la traduction (dite aussi translation) de ces adresses virtuelles en adresses réelles en mémoire centrale.
- Le mécanisme de gestion des adresses doit être capable de détecter si une information se trouve en mémoire centrale ou non.
- On dispose de structure : table de traduction (translation table)
  - résidente en mémoire centrale,
  - décrit le contenu de la mémoire centrale et celle de la mémoire de masse.

Module Gestion de la mémoire

- Si l'adresse virtuelle correspond à un bloc en mémoire principale :
  - la traduction en adresse physique est immédiate
  - on accède à l'information cherchée.
- Si l'information ne se trouve pas en mémoire principale:
  - Un événement (déroutement) dit de traitement de défaut de page est généré.
  - Cet événement engendre l'interruption (abort) de l'instruction en cours
  - Donne la main au module de transfert pour charger la partie à exécuter (en libérant si nécessaire de la place).
  - Ensuite, le module de gestion des adresses effectue la traduction de l'adresse virtuelle en adresse physique
  - rend tout de suite la main au programme qui a causé le défaut pour terminer normalement l'exécution.

Module Gestion de la mémoire



Remarques

- Le signal de traitement de défaut de page est le même émis lorsqu'on cherche à accéder à une information située à une adresse supérieure à la taille de la mémoire centrale ou à une zone de mémoire protégée (interdite d'accès).
- Le mécanisme de fourniture de l'adresse du programme de traitement du défaut de page est semblable à celui utilisé lors des interruptions :
  - sauvegarde de l'adresse de l'instruction avortée,
  - l'unité centrale récupère l'adresse du programme de traitement dans une table de vecteurs d'interruptions.
- Bien que le traitement de défaut de pages est semblable au traitement d'une interruption externe, le défaut de page s'en distingue fondamentalement par le fait que l'instruction ne doit pas se terminer mais doit au contraire revenir à son état initial. Les registres altérés par l'exécution doivent être restitués de telle sorte, qu'après traitement du déroutement, on puisse relancer l'instruction à son point de départ.

Exemple

- Soit un programme dont le code occupe 1024 octets en mémoire et qui utilise un vecteur de 1000 caractères.
- Ce programme est exécuté dans un système ayant les caractéristiques suivantes :
  - utilise la pagination de la mémoire,
  - la taille de la mémoire réelle est de 1 Mo,
  - la taille d'une page est de 512 octets ,
  - les instructions à référence mémoire ont un champ d'adresse de 24 bits.

Exemple

- Questions :
  - Déterminer :
    - Taille de la mémoire virtuelle
    - *Le nombre de bits du champ déplacement*
    - *Le nombre de bits du numéro d'une page virtuelle*
    - *Le nombre de bits d'une adresse réelle*
    - *Le nombre de bits du numéro d'une page réelle (cadre)*
    - *Le nombre d'entrées de la table des pages virtuelles.*
  - *Le chargement de ce programme en mémoire engendre-t-il une fragmentation interne ? Justifiez votre réponse.*

Propriétés: Algorithme demande pages

- **Statique ou dynamique :**
  - Un algorithme statique alloue un nombre fixe de cadres à chaque processus.
  - Un algorithme dynamique permet de changer le nombre de cadres d'un processus au cours d'une exécution.
- **Stratégie de chargement :**
  - Quand un bloc doit-il être chargé dans un cadre ?
  - On peut choisir la demande pure (un bloc n'est chargé que s'il est demandé) mais on peut aussi décider de prépaginer (c.à.d de charger à l'avance un certain nombre de blocs).
- **Stratégie de remplacement (et de placement) :**
  - S'il n'y a pas de cadre libre, quel cadre occupé doit-on déplacer pour placer le nouveau bloc ?
  - Dans quel cadre un nouveau bloc doit être chargé ?

Stratégies de recherche

- Répondant à la question : quand doit-on ramener la partie suivante d'un programme ou ses données de la mémoire auxiliaire vers la mémoire centrale ?
- Deux catégories de méthodes :
  - Pagination à la demande (Demand Fetch) :
    - Le chargement n'a lieu qu'en cas de défaut de page et on ne charge que la page manquante.
    - Depuis longtemps, la stratégie adoptée consiste à ramener vers la mémoire centrale qu'une partie (programme ou donnée) que lors de sa demande.
  - Pagination anticipée (Anticipatory Fetch) :
    - On charge les pages à l'avance ; mais comment prévoir efficacement ?
    - On croyait en fait que le temps passé pour prévoir l'avenir est un temps perdu.
    - De nouvelles recherches ont montré que la recherche anticipée peut améliorer l'état du système

Pagination à la demande

- **Avantages :**
  - Besoin minimal en E/S (du chargement des blocs)
  - Besoin minimal en MC
  - Réponse plus rapide ?
  - Plus de processus et/ou d'utilisateurs qui partagent la MC (degré de multiprogrammation)

Algorithme de transfert vers MC

Si Défaut de page
Alors
Si existe une zone libre
Alors
Charger la page dans la zone libre
Calcul de l'adresse physique
Sinon
Choisir la page victime
Si La page victime est modifiée
Alors
Ecriture d'une copie sur disque.
FinSi
Chargement de la nouvelle page
Mise à jour de la table de pages
Revenir à l'instruction interrompue
FinSi

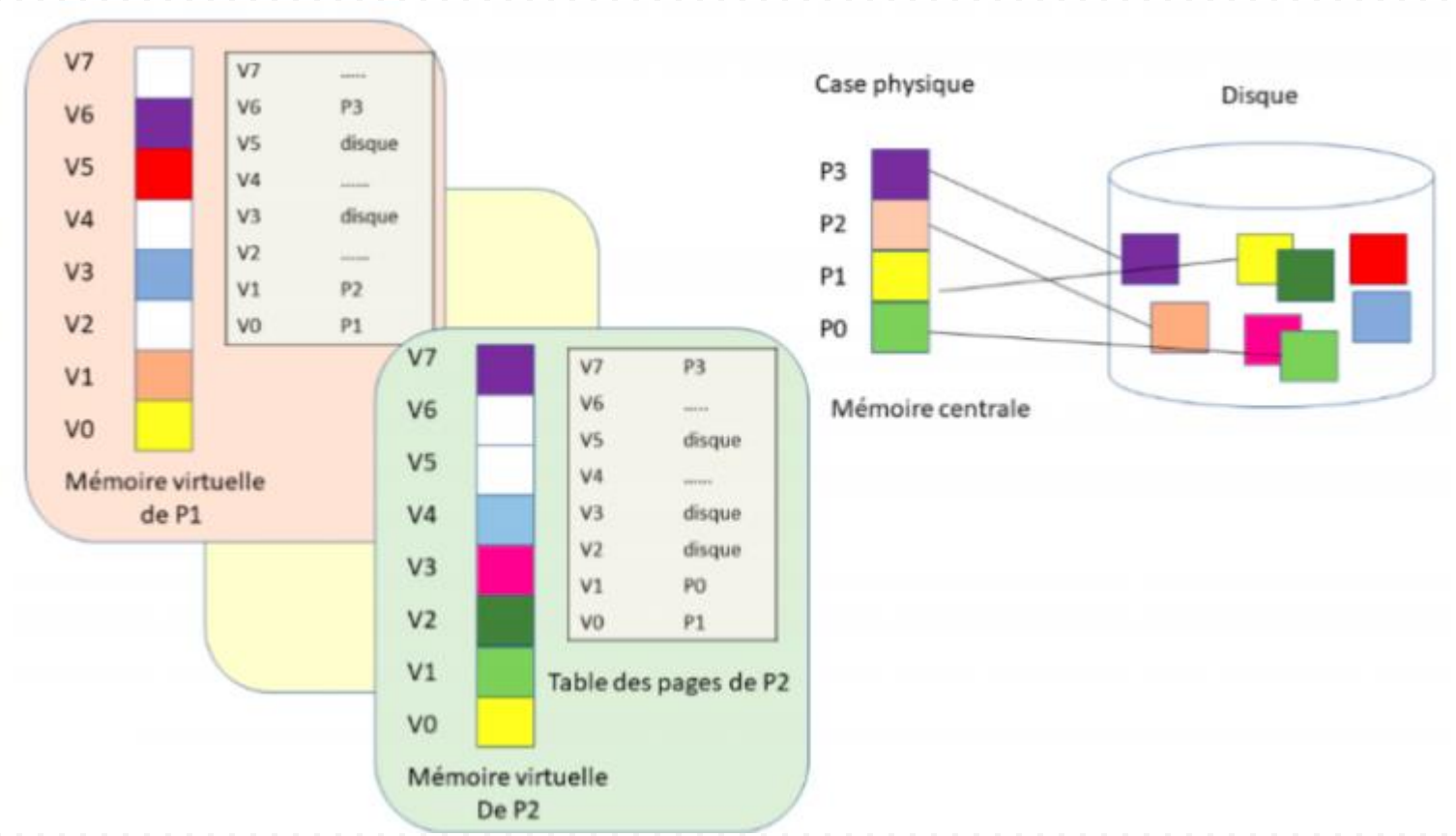
Pagination anticipée : Working set

- **Cas d'un système Multitâches**
- **Idée de base** (principe de localité) :
  - la plupart des programmes référencent leur espace de travail uniformément (peu de changement d'une étape à une autre),
  - ces références tendent à se regrouper autour d'un petit nombre de pages.
- Cet ensemble est appelé : Working Set (Espace de travail)
- Charger l'espace de travail pendent qu'un autre utilisateur est en train d'être servi par l'unité centrale.
- L'inconvénient de cette approche se présente lorsqu'un programme est en train de changer son espace de travail auquel cas on charge inutilement plusieurs pages.

Working set : Caractéristiques

- **Working Set (Espace de travail)**
  - représenté par une fonction W(k, t) où :
    - k est le nombre de références à la mémoire à l'instant t.
    - W(k,t) est une fonction monotone croissante car les k+1 références les plus récentes utilisent les k références précédentes
- La variation de W(k,t) dans le temps est logarithmique car le Working Set change lentement à la fin et très rapide pour k petit.
  - Ceci sous-entend, qu'il existe un intervalle large de k pour lequel l'espace de travail est constant.
  - Ceci permet de parier, avec des probabilités raisonnables de succès, sur les numéros des pages qui seront nécessaires au moment du redémarrage du programme en partant de l'espace de travail utilisé au moment où il a été suspendu.

Partage entre processus



Partage entre processus

- Partage entre deux processus. La figure montre deux processus qui sont en cours d'exécution et qui partagent le code d'une même application.
- Chaque processus possède indépendamment de l'autre processus sa propre mémoire virtuelle avec sa propre table des pages. Les deux instances utilisent les mêmes adresses virtuelles (0, 1, 2, 3, 4, 5, 6, 7).
- À l'instant observé, chaque processus fait trois utilisations distinctes des pages dans sa mémoire virtuelle, soit une page de code partagé, une page de données propres et une page de communication avec l'autre processus. La page virtuelle V_0 de chaque processus donne accès au code partagé et est associée à la même case de mémoire physique P_1 ; la page virtuelle V_1 de chaque processus permet d'adresser des données propres à chaque processus et pour cela chacune des pages V_1 est associée à une case différente, P_2 pour le processus 1, P_0 pour le processus 2 ; la page virtuelle pour l'envoi et la réception de messages entre les processus n'est pas la même dans chaque programme : c'est V_6 pour le processus 1 et V_7 pour le processus 2, mais ces deux pages ont une entrée vers la même case, ce qui leur permet de partager les données communiquées.

Remplacement des pages

- Ceci revient à répondre à la question : Comment répartir les pages sur les différents processus et le système ?
- Deux stratégies sont possibles :
  - Remplacement local : chaque processus choisit parmi les cadres de page qui lui sont déjà alloués
  - Remplacement global : un processus peut récupérer un cadre de page d'un autre processus
- Exemples :
  - Windows NT applique le mode de remplacement local
  - Unix applique un remplacement global.

RL : Principe de base

- A chaque processus, sont affectés un certain nombre de cadres :
  - à utiliser de façon autonome,
  - son temps d'exécution ne dépend que de son propre comportement.
- Ceci peut se faire grâce à un **pointeur** sur la liste de cadres qu'utilise un processus.
- Ainsi, on peut appliquer à chaque processus un algorithme de remplacement différent.

RL : Problématique

- Le remplacement local demande que l'on réalise un partage entre les différents processus :
  - Si on alloue peu de cadres à un processus, son nombre de défauts de pages va augmenter et on ralentit ainsi son exécution.
  - Dans le cas contraire, le processus va pénaliser un second programme qui restera en attente de cadres libres alors qu'un autre a plus que nécessaire.
- **Deux stratégies :**
  - Le partage équitable
  - Le partage proportionnel

Partage équitable

- Si on dispose de m pages de mémoire physique et de n processus,
  - alors on alloue m/n pages par processus !
- On retrouve ici un problème proche de la fragmentation interne :
  - un grand nombre de pages est donné à un processus qui en utilise effectivement peu.

Partage proportionnel

- On peut améliorer cette approche en utilisant $S = \Sigma S_i$ où S_i est le nombre de pages du Processus i.
- Chaque processus se voit attribué m (= S_i/S) pages.
- On peut encore améliorer en faisant varier ce rapport en fonction de **la priorité** de chaque processus.
- Ceci peut engendrer le **problème d'écroulement** :
  - Si le nombre de cadres alloués à un processus non prioritaire tombe en dessous de son minimum vital, ce processus est constamment en défaut de page : il passe tout son temps à réaliser des demandes de pages.
  - Ce processus doit être alors éjecté entièrement en zone de swap et reviendra plus prioritaire quand il y aura de la place

Exemple

- Nombre total de cadres : m = 64
- 2 processus P_1 et P_2,
- S_i : Taille nécesaire pour chaque processus
  - $S_1 = 10$ (nbre pages virtuelles)
  - $S_2 = 127$
- Allocation égale : 32 cadres par processus
- Allocation proportionelle :

$$S = \sum s_i$$

$$a_i = \text{allocation for } p_i = \frac{s_i}{S} \times m$$

$$a_1 = \frac{10}{137} \times 64 \approx 5$$

5 cadres → P_1

$$a_2 = \frac{127}{137} \times 64 \approx 59$$

59 cadres → P_2

RG : Principe de base

- Décharger éventuellement les pages d'un autre programme pour exécuter le processus qui a provoqué le défaut de page.
- Le comportement d'allocation de pages aux processus dépend :
  - de la charge du système
  - du comportement des différents processus.
- En général, le remplacement en mode global est plus utilisé et donne de meilleurs résultats mais il est plus difficile à mettre en œuvre.

Exemple

- Les compilateurs C allouent les structures répétitives sur des plages d'adresses croissantes contiguës :
  - int A[m][n] est une matrice de m x n entiers.
  - En mémoire (ligne par ligne) :
    - A[0][0] suivi de A[0][1] suivi de … A[0][n-1]
    - A[1][0] … A[1][n-1]
    - …
    - A[m-1][0] … A[m-1][n-1]
- En fortran, l'allocation des tableaux se fait par colonne (dans l'autre sens) …

Exemple en C

- On suppose que :
  - la taille d'une page est 4 Ko
  - int est codé sur 2 octets.
  - 1 seul cadre est réservé pour les données du programme
- Le programme :

```
int A[2048][2048]
main
{ int i, j ;
for (i=0 ; i<2048 ; i++)
   for (j=0 ; j<2048; j++)
         A[i][j] = 1 ;
}
```

- **Ce processus accède à une nouvelle page toutes les 2048 affectations.**
- **Nombre de défauts de pages est 2048**

Exemple en C

- On suppose que :
  - la taille d'une page est 4 Ko
  - int est codé sur 2 octets.
  - 1 seul cadre est réservé pour les données du programme
- Le programme :

```
int A[2048][2048]
main
{ int i, j ;
for (i=0 ; i<2048 ; i++)
   for (j=0 ; j<2048; j++)
        A[j][i] = 1 ;
}
```

Ce processus accède à une nouvelle page à chaque affectation.
Nombre de défauts de pages est 2048x2048

Stratégie de remplacement

Problématique :

- Le principal problème : pas d'espace libre où loger le nouveau bloc sollicité par le processeur
- On cherche souvent le bloc "le moins utile" dans la suite de l'exécution, c'est à dire, dont l'absence se fera le moins sentir.
- Il faut :
  - prédire les références futures des informations présentes en MC
  - enlever le bloc dont la prochaine référence est espérée la plus lointaine.
- Ce problème dépend de la dynamique de l'exécution, difficile à prédire.
- Les approches utilisées se basent sur le bon sens (des heuristiques) dont on espère la réussite dans la majorité des cas.

Algorithmes de remplacement de pages (ARP)

- Lorsqu'un défaut de page se produit, il faut amener la page manquante en mémoire centrale.
  - S'il y a de la place pour elle, il suffit de la charger.
  - S'il n'y a pas de place, il faut en libérer une, et le choix de la victime est réalisé par l'un des algorithmes de remplacement.

Algorithmes de remplacement de pages (ARP)

- Choisir la victime de façon à minimiser le taux de défauts de page
  - **Pas évident !**
    - Page dont nous n'aurons pas besoin dans le futur ?
      - Impossible à savoir !!
    - ou page pas souvent utilisée ?
    - ou page qui a été longtemps en mémoire ?
    - etc.

Algorithmes de remplacement de pages (ARP)

On appelle **Oracle** un algorithme (hypothétique) qui serait capable de prendre les meilleures décisions en tenant compte du futur.

Performance d'un ARP

- p : Probabilité d'un défaut de page
  - $0 \leq p \leq 1.0$
  - Si p = 0, pas de défauts de pages
  - Si p = 1, chaque référence est un défaut de page
- Temps d'Accès Effectif (TAE) :

  TAE = (1 – p) x accès MC + p x Temps défaut de page
- Temps de défaut de page = Somme des 3 temps :
  - Swap éventuel de la page victime sur disque (échange)
  - Swap de la page sollicitée en mémoire centrale
  - Relancement de l'instruction

Exemple

- Données réelles :
  - Temps d'accès mémoire ~ 1 µs
  - Temps de Swap disque ~ 10 ms = 10000 µs
  - 50% du temps, la page remplacée a été modifiée

    => elle sera swappée sur disque
- Temps d'Accès Effectif
  - TEA = (1 – p) x 1 µs + p x 10000 µs

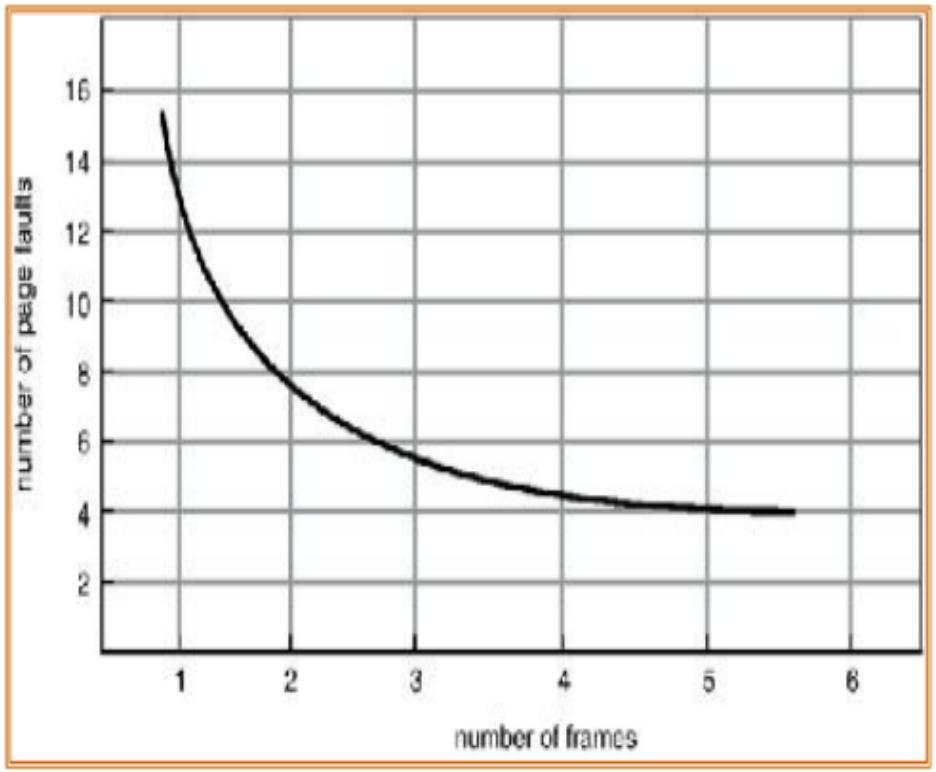
    = 0.5 + 5000 µs
  - TAE est de l'ordre de la ms

Algorithme pagination stable

- On appelle suite de références $w = p_1\ p_2 \ldots p_n$ une suite de numéros de pages correspondant à des accès à ces pages.
- Un algorithme A de pagination sur m cadres est stable si ∀ m , ∀ w , on a :

  Coût (A , m , w) ≤ Coût (A , m+1 , w)
- Cette notion est importante pour éviter l'écroulement du SE (thrashing) par une génération excessive de défauts de pages.

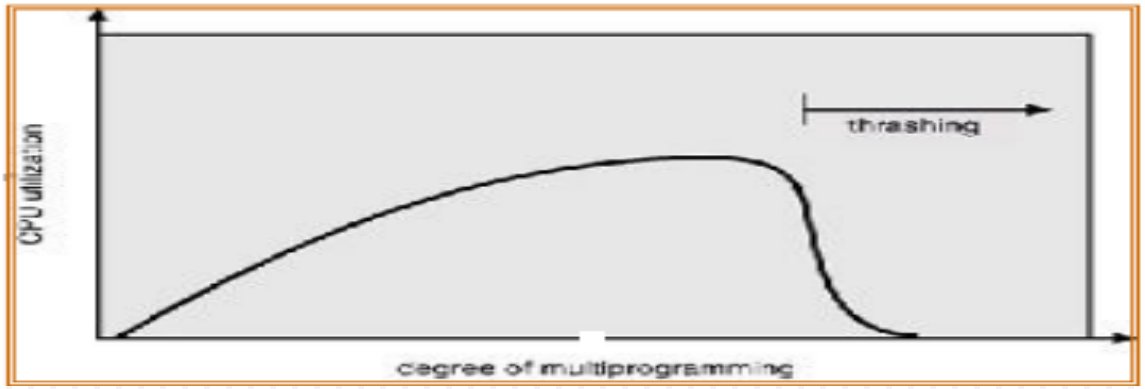
Défauts de page vs cadres



Plus le nombre de cadres augmente, plus le nombre de défauts pages a tendance à diminuer

Ecroulement (trashing)

- Pourquoi la pagination fonctionne ?
- Modèle de localité
  - Les processus migrent d'une localité à une autre
  - Les localités peuvent s'entrelacer
- Pourquoi l'écroulement a lieu ?
  - ∑ taille des localités > taille totale de la mémoire



Mécanismes MV basée pagination

- Deux mécanismes sous-jacents à la pagination et à la mémoire virtuelle :
  - **Faute de page** (défaut de page, *page fault*)
    - La page n'est pas présente dans la RAM
    - Interruption matérielle (MMU ⟶ CPU)
  - **Remplacement de page**
    - Chargement de page avec une mémoire physique (RAM) pleine
    - Algorithme de remplacement

Amélioration de performances

- Si à un instant t le processus utilise l'adresse a, il y a une forte probabilité qu'à l'instant t + Δt , le processus référence l'adresse a + Δa (Δt et Δa étant petits).
- Plusieurs algorithmes sont basés sur ce principe ; leurs mises en œuve nécessitent souvent des dispositifs matériels particuliers
- Il y a deux critères pour améliorer la performance de la mémoire virtuelle :
  - Réduire le temps d'accès aux disques : critère purement matériel
  - Réduire la valeur de p par le choix d'un bon algorithme de demande de page : critère purement logiciel
- Pour illustrer les algorithmes de remplacement, on suppose qu'on dispose de :
  - m cadres en MC
  - On utilise la suite de références W

ARP optimal : Principe

- Le règle de remplacement est « remplacer le *cadre* qui ne sera pas utilisé pendant la durée la plus longue
- Pour ce faire, le SE indexe chaque page par le nombre d'instructions qui seront exécutées avant qu'elle ne soit référencée.
- En cas de nécessité, le SE retire la page d'indice le plus élevé, c'est à dire la page qui sera référencée dans le futur le plus lointain.

ARP optimal : Principe

- L'algorithme optimal (OPT) choisit pour page à remplacer celle qui sera référencée le plus tard possible.

  En d'autres termes :

  « Remplacer la page qui ne sera pas utilisée pendant la durée la plus longue »

  - Il produit le plus petit nombre de défauts de page
  - Il est impossible à réaliser (car il nécessite la connaissance du futur) mais sert à la comparaison pour les autres algorithmes :
    - ordre chronologique d'utilisation (LRU)
    - ordre chronologique de chargement (FIFO)
    - deuxième chance ou horloge (clock)

ARP optimal : Principe

- Cet algorithme est pratiquement impossible à appliquer :
  - Comment calculer les indices des pages (instructions) ?
- Cet algorithme est stable.
- Utilisé pour mesurer les performances des autres algorithmes
  - avec un simulateur, on peut évaluer les performances de cet algorithme et s'en servir comme référence pour les suivants.

ARP Optimal : Exemple

- Etant donné 4 frames et la séquence de références suivante :

Pages virtuelles 1, 2, 3, 4, 1, 2, 5, 1, 2, 3, 4, 5

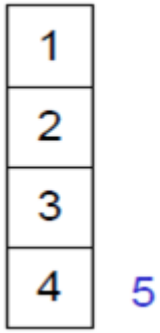
Cadres

ARP Optimal : Exemple

- 4 frames (cadres) et une séquence de références :

$$W = 1, 2, 3, 4, 1, 2, 5, 1, 2, 3, 4, 5$$

4 cadres



4

5

6 défauts de page

- Lorsque arrive la référence 5, on voit qu'il faut éliminer la page 4.

ARP Optimal : autres exemples

- L'algorithme Optimal nécessite de prévoir l'avenir.
  - Il retire la page non réutilisée
  - ou qui sera réutilisée le plus loin dans le futur

Exemple ($M = 3$: 7 défauts de pages)

défaut	↓	↓	↓	↓			↓			↓	↓	
ω	0	1	2	3	0	1	4	0	1	2	3	4
	0	0	0	0	0	0	0	0	0	2	2	2
		1	1	1	1	1	1	1	1	1	3	3
			2	3	3	3	4	4	4	4	4	4
Y	-	-	-	2	-	-	3	-	-	0	1	-

Exemple ($M = 4$: 6 défauts de pages)

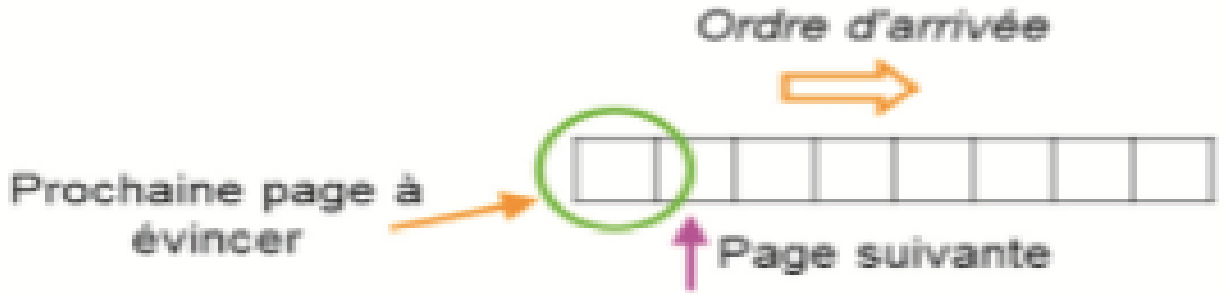
défaut	↓	↓	↓	↓			↓				↓	
ω	0	1	2	3	0	1	4	0	1	2	3	4
Y	-	-	-	-	-	-	3	-	-	-	0	-

ARP FIFO

- On éjecte la page la plus ancienne en mémoire sans tenir compte des références qui ont pu être faites.
- Le bon sens de cette approche : le bloc le plus anciennement chargé en mémoire centrale est probablement (du moins on espère) celui qui a le moins de chance d'être utilisé lors des prochains accès

ARP FIFO

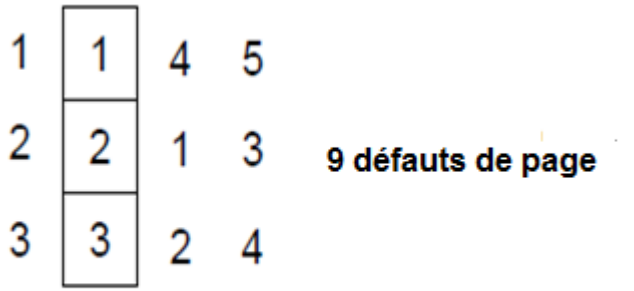
- L'algorithme **FIFO** consiste à remplacer les pages **selon leur ordre d'arrivée.**
  - **Facile** à implémenter et rapide
  - Ne tient pas compte de **l'utilisation récente et future** des pages.



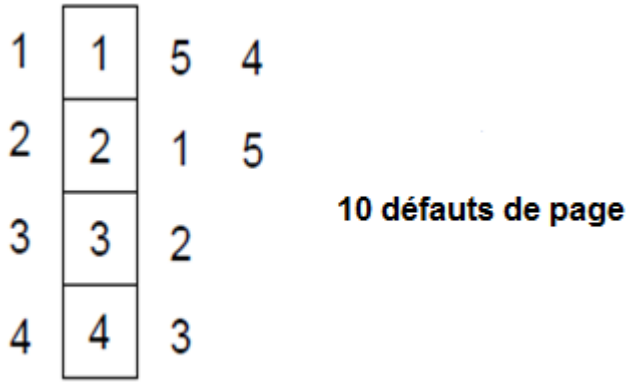
ARP FIFO

- **Soit la séquence de références :** 1, 2, 3, 4, 1, 2, 5, 1, 2, 3, 4, 5

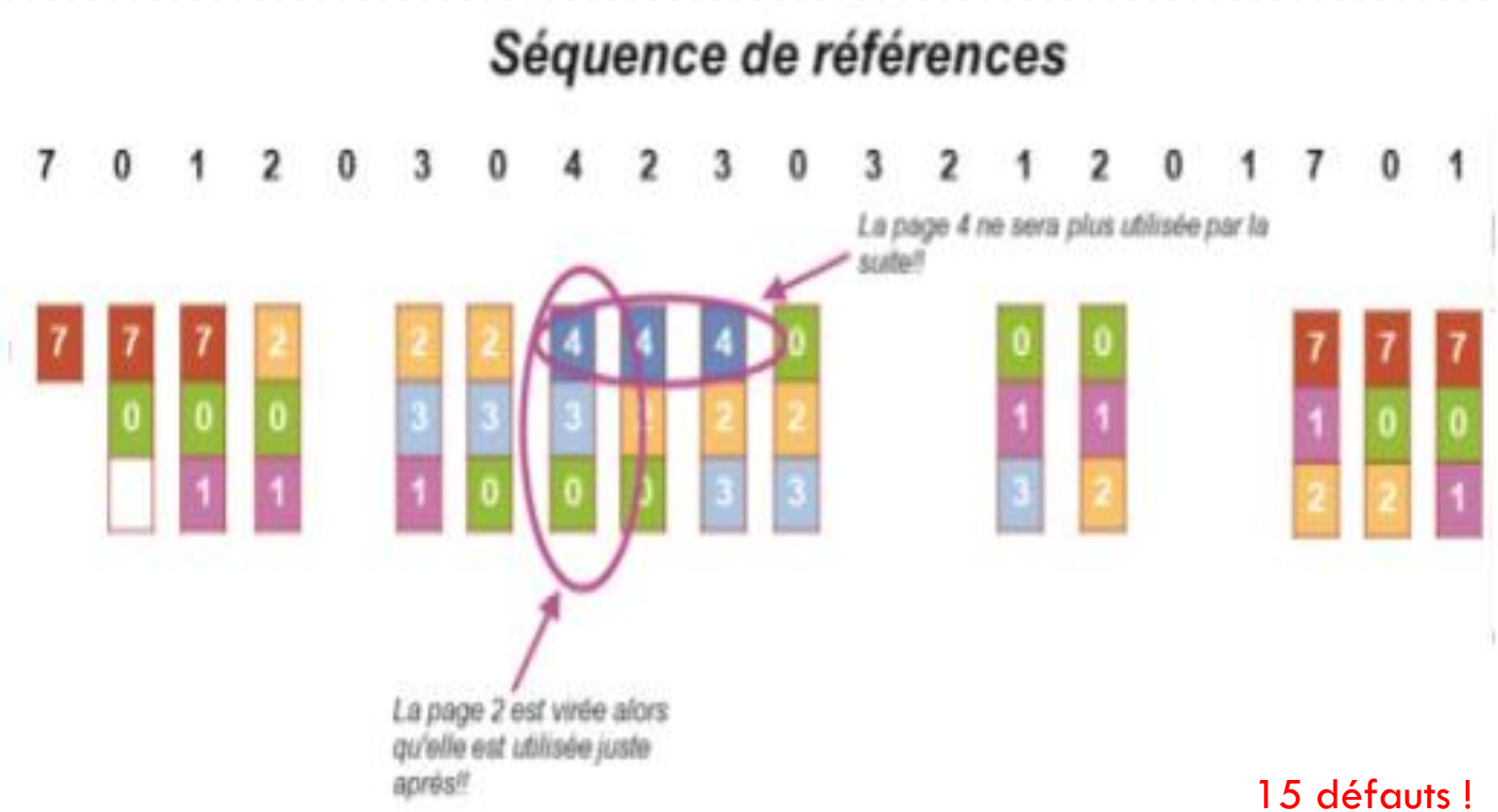
  - Avec 3 cadres et 5 pages



  - Avec 4 cadres et 5 pages



ARP FIFO



15 défauts !

Implémentation et inconvénients

- On peut implémenter cet algorithme grâce à une liste chaînée où :
  - La page en tête est la première chargée en mémoire et celle en queue est la dernière.
  - Le remplacement se fait sur la page en tête de liste.
- Avantages :
  - Simple à implémenter
- Inconvénients : manque d'efficacité.
  - Il existe des suites de pages pour lesquelles cet algorithme fait plus de défauts de page avec quatre cadres mémoire qu'avec trois :

    Par exemple : 1 2 3 4 1 2 5 1 2 3 4 5.
  - Les vieilles pages peuvent aussi être celles qui sont le plus utilisées.

ARP LRU

- LRU (Least Recently Used : moins récemment utilisé) se base sur l'heuristique suivante :
  - Si un bloc en mémoire centrale est longuement non sollicité, il y a de fortes chances que les prochains accès ne seront pas à ce bloc.
  - Inversement, on fait le pari que les pages qui ont été récemment utilisées le seront dans un proche avenir.
- Bien qu'on puisse trouver des cas où cette approche échoue, elle a, en pratique, montré ces performances.

ARP LRU

- **Ordre chronologique d'utilisation (LRU)**
  - Il remplace la page dont la dernière référence remonte au temps le plus lointain (le passé est 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
    - La page sortante est celle qui n'a plus été utilisée depuis le plus longtemps
- Performance presque aussi bonne que celle de l'algorithme optimal

ARP LRU

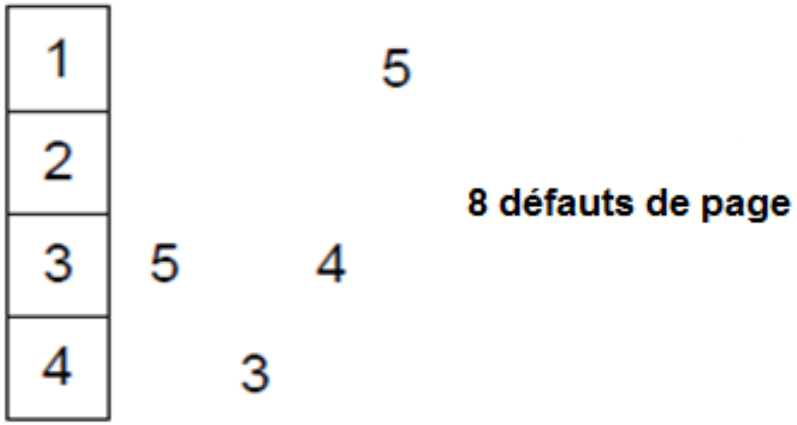
- Etant donné les 4 frames (cadres) et une même séquence de références :

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

ARP LRU

- Etant donné les 4 frames (cadres) et une même séquence de références :

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



ARP LRU

Exemple ($M = 3$: 10 défauts de page)

défaut	↓	↓	↓	↓	↓	↓	↓			↓	↓	↓
ω	0	1	2	3	0	1	4	0	1	2	3	4
	0	0	0	3	3	3	4	4	4	2	2	2
		1	1	1	0	0	0	0	0	0	3	3
			2	2	2	1	1	1	1	1	1	4
Y	-	-	-	0	1	2	3	-	-	4	0	1

Exemple ($M = 4$: 8 défauts de page)

défaut	↓	↓	↓	↓			↓			↓	↓	
ω	0	1	2	3	0	1	4	0	1	2	3	4
Y	-	-	-	-	-	-	2	-	-	3	4	0

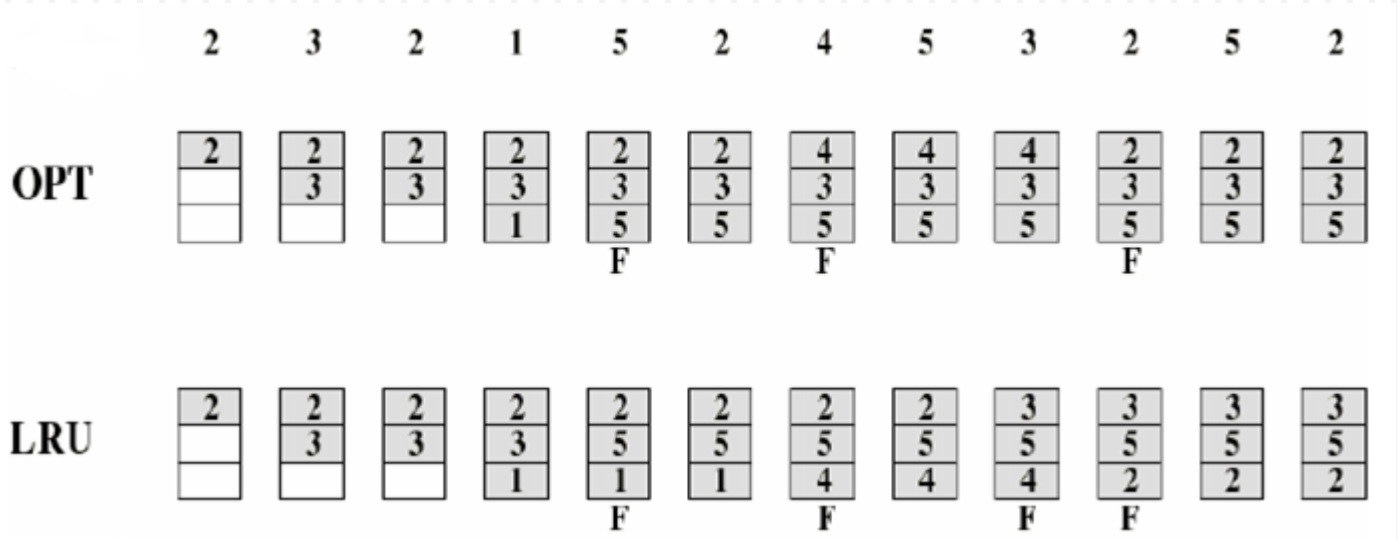
Comparaison OPT-LRU

- Soit à comparer les algorithmes OPT et LRU sur un processus de 5 pages et 3 pages physiques disponibles seulement

	2	3	2	1	5	2	4	5	3	2	5	2
OPT	2	2	2	2	2	2	4	4	4	2	2	2
		3	3	3	3	3	3	3	3	3	3	3
				1	5	5	5	5	5	5	5	5
LRU	2	2	2	2	2	2	2	2	3	3	3	3
		3	3	3	5	5	5	5	5	5	5	5
				1	1	1	4	4	4	2	2	2

Comparaison OPT-LRU

- Soit à comparer les algorithmes OPT et LRU sur un processus de 5 pages et 3 pages physiques disponibles seulement :
  - OPT occasionne 3 + 3 défauts de page
  - LRU occasionne 3 + 4 défauts de page



Difficultés d'impémentation

- La mise en œuvre du LRU est difficile car :
  - Maintenir l'ordre des pages oblige une action complexe à chaque accès mémoire ⇒ impraticable.
  - Stocker la date du dernier accès, cela oblige un tri pour le choix ⇒ possible mais coûteux.
  - Compteur bidimensionnel : dispositif matériel coûteux
- La méthode a un résultat proche de l'algorithme optimal.
- Mais il faut faire quelque chose à chaque accès mémoire et obtenir un maximum à chaque remplacement de page.
- Souvent la MMU ne fournit pas les outils indispensables.
- D'où l'utilisation d'approximations logicielles de LRU.
- Pour ces approximations on utilise souvent un système présent dans la plupart des MMU : le bit d'accès A (ou R) et le bit de modification M (ou D le dirty bit).

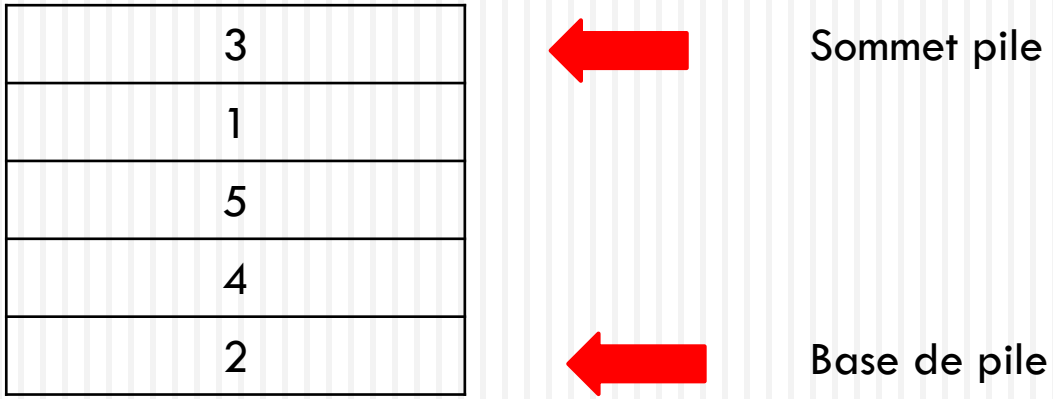
Implémentation avec compteur

- A chaque entrée de la table des pages, on ajoute un compteur de temps qui est mis à jour à chaque accès.
  - A chaque fois qu'une page est référencée, on copie une horloge logique dans ce compteur
  - Quand une page est référencée, incrémenter l'horloge logique et la copier dans le compteur de la page référencée

    $\Rightarrow$ On a toujours le temps de la dernière utilisation de cette page
  - Rechercher sur l'ensemble de la table la victime : la page avec la plus petite valeur de compteur

Implémentation avec pile

- A chaque fois que l'on accède à une page, celle-ci est placée en sommet de la pile.
  - Le dessus est toujours la page la plus récemment utilisée
  - Le fond de la pile la moins récemment utilisée.



Implémentation avec masques

- On utilise un octet associé à chaque page.
- Le système positionne à 1 le bit de poids fort à chaque accès à la page.
- Toutes les N (nanosecondes) clics d'horloge, le système applique un décalage à droite de l'octet associé à chaque page.
- On obtient ainsi un historique de l'utilisation de la page.
  - L'octet à 00000000 indique que la page n'a pas été utilisée depuis 8 cycles
  - L'octet 11111111 indique que la page a été utilisée pendant les 8 derniers cycles.
  - La page de masque 11000100 a été utilisée plus récemment que 01110111.

Implémentation avec masques

- Si l'on interprète ces octets comme des entiers non signés :
  - c'est la page ayant le plus petit octet qui a été utilisée le moins récemment
  - l'unicité des numéros n'étant pas assurée la sélection entre numéros identiques se fait avec l'ordre FIFO.
- Cet algorithme est en théorie :
  - le meilleur pour implémenter le remplacement des pages puisqu'il retire la page la moins pénalisante et celle qui risque fort d'être utilisée à nouveau.
  - le plus coûteux étant donné que les mises à jour sont faites à chaque accès à une page.

Implémentation avec matrices

- Les opérations sur les listes chaînées étant très coûteuses, les concepteurs ont eu recours aux matrices carrées de taille n, n est le nombre de cadres.
- La matrice est initialisée à 0.
- Lorsque la page logée dans le cadre n° i est référencée :
  - la ième ligne est mise à 1
  - ensuite la ième colonne est mise à 0.

  } les 2 actions ensemble
- La page la moins récemment référencée est celle dans le cadre ayant la ligne contenant la plus petite valeur binaire.

Exemple

Suite de références :
0 2 1 3
1 2 3

	0	**1**	**2**	**3**
0	0	0	0	0
1	0	0	0	0
2	0	0	0	0
3	0	0	0	0

Initialisation

	0	**1**	2	3
0	0	**0**	0	1
1	**1**	**0**	**1**	**1**
2	1	**0**	0	1
3	0	**0**	0	0

Page 1

	0	1	**2**	3
0	0	0	**0**	0
1	1	0	**0**	1
2	**1**	**1**	**0**	**1**
3	1	0	**0**	0

	0	1	2	3
0	**0**	**1**	**1**	**1**
1	**0**	0	0	0
2	**0**	0	0	0
3	**0**	0	0	0

Page 0

	0	1	2	**3**
0	0	0	0	**0**
1	1	0	1	**0**
2	1	0	0	**0**
3	**1**	**1**	**1**	**0**

Page 3

	0	1	2	**3**
0	0	0	0	**0**
1	1	0	0	**0**
2	1	1	0	**0**
3	**1**	**1**	**1**	**0**

	0	1	**2**	3
0	0	1	**0**	1
1	0	0	**0**	0
2	**1**	**1**	**0**	**1**
3	0	0	**0**	0

Page 2

	0	**1**	2	3
0	0	**0**	0	0
1	**1**	**0**	**1**	**1**
2	1	**0**	0	0
3	1	**0**	1	0

Page 1

	0	1	2	3
0				
1				
2				
3				

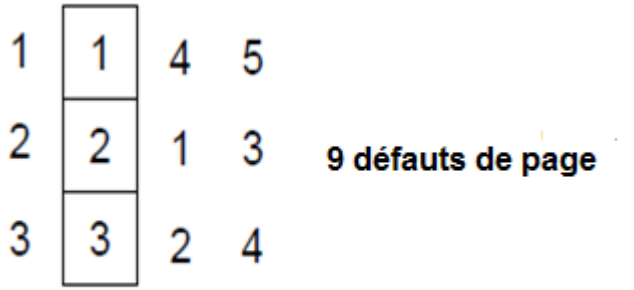
ARP FIFO (Retour)

- **Logique :** une page qui a été longtemps en mémoire a eu sa chance de s'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 vieille, elle sera remplacée par FIFO !

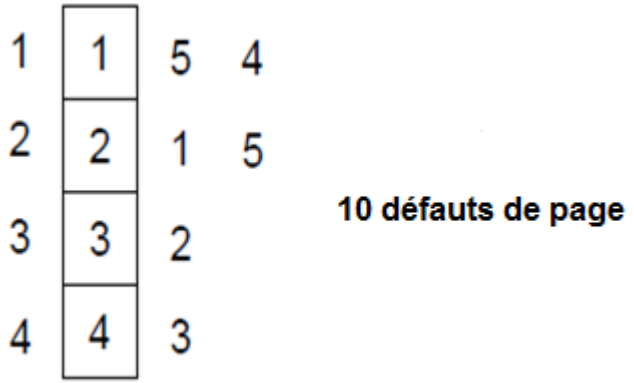
ARP FIFO (retour)

- **Soit la séquence de références :** 1, 2, 3, 4, 1, 2, 5, 1, 2, 3, 4, 5

  - Avec 3 cadres et 5 pages



  - Avec 4 cadres et 5 pages



ARP FIFO (retour)

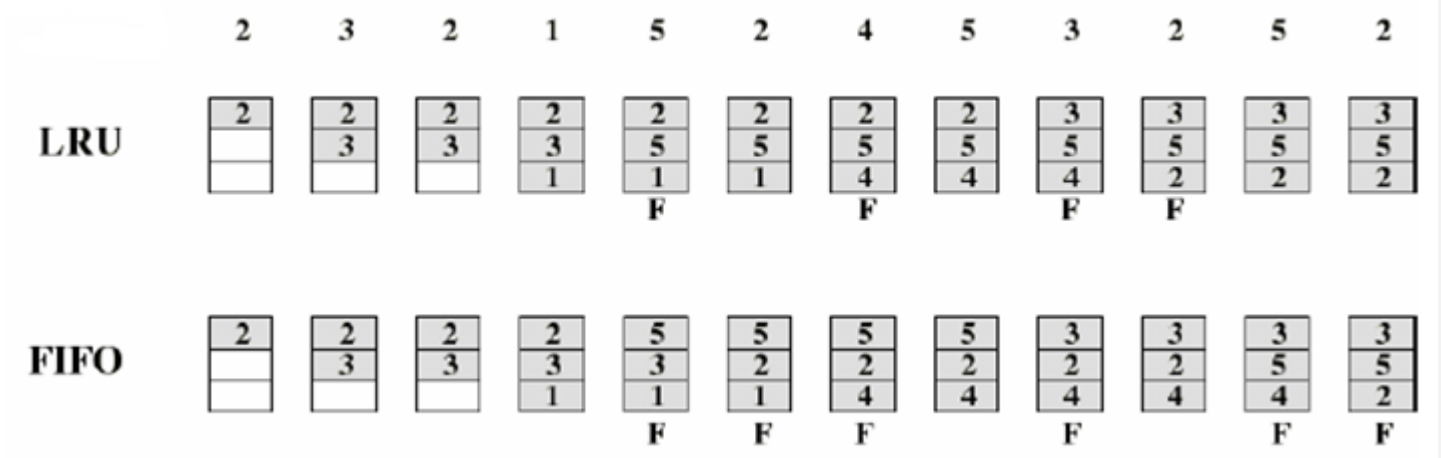
Exemple ($M = 3$: 9 défauts de pages)

défaut	↓	↓	↓	↓	↓	↓	↓			↓	↓	
ω	0	1	2	3	0	1	4	0	1	2	3	4
	0	0	0	3	3	3	4	4	4	4	4	4
		1	1	1	0	0	0	0	0	2	2	2
			2	2	2	1	1	1	1	1	3	3
Y	-	-	-	0	1	2	3	-	-	0	1	-

Exemple ($M = 4$: 10 défauts de pages !)

défaut	↓	↓	↓	↓			↓	↓	↓	↓	↓	↓
ω	0	1	2	3	0	1	4	0	1	2	3	4
	0	0	0	0	0	0	4	4	4	4	3	3
		1	1	1	1	1	1	0	0	0	0	4
			2	2	2	2	2	2	1	1	1	1
				3	3	3	3	3	3	2	2	2
Y	-	-	-	-	-	-	0	1	2	3	4	0

Comparaison FIFO avec LRU



- Contrairement à FIFO, LRU reconnait que les pages 2 et 5 sont utilisées fréquemment
- La performance de FIFO est moins bonne :
  - Pour cet exemple, 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 un problème avec notre façon de comparer les méthodes sur la base d'une séquence aléatoire :
  - les références aux pages dans un programme ne sont pas vraiment aléatoires

ARP Seconde chance : Principe

- Amélioration de l'algorithme FIFO :
  - Conserve l'avantage de simplicité
  - Pallie les inconvénients en donnant une seconde chance à la page en tête.
- Comment faire ?
  - Si la page en tête de liste est réutilisée, elle sera enlevée au prochain remplacement

ARP Seconde chance : Mise en oeuvre

- Un bit associé à chaque page est positionné à 1 à chaque fois qu'une page est utilisée par un processus.
- Avant de retirer une page de la mémoire, on va essayer de lui donner une deuxième chance.
  - Si le bit d'utilisation est à 0 la page est swappée hors mémoire (elle n'a pas été utilisée depuis la dernière demande de page).
  - Si le bit est à 1, il est positionné à zéro et la page est mise en queue de liste et on cherche une autre victime.
- Ainsi une page ne sera swappée hors mémoire que si :
  - toutes les autres pages ont été utilisées,
  - ont utilisé aussi leur deuxième chance.

ARP Seconde chance : Mise en oeuvre

- **Basé sur l'algorithme FIFO**
- **Utilise le bit R de la PTE**

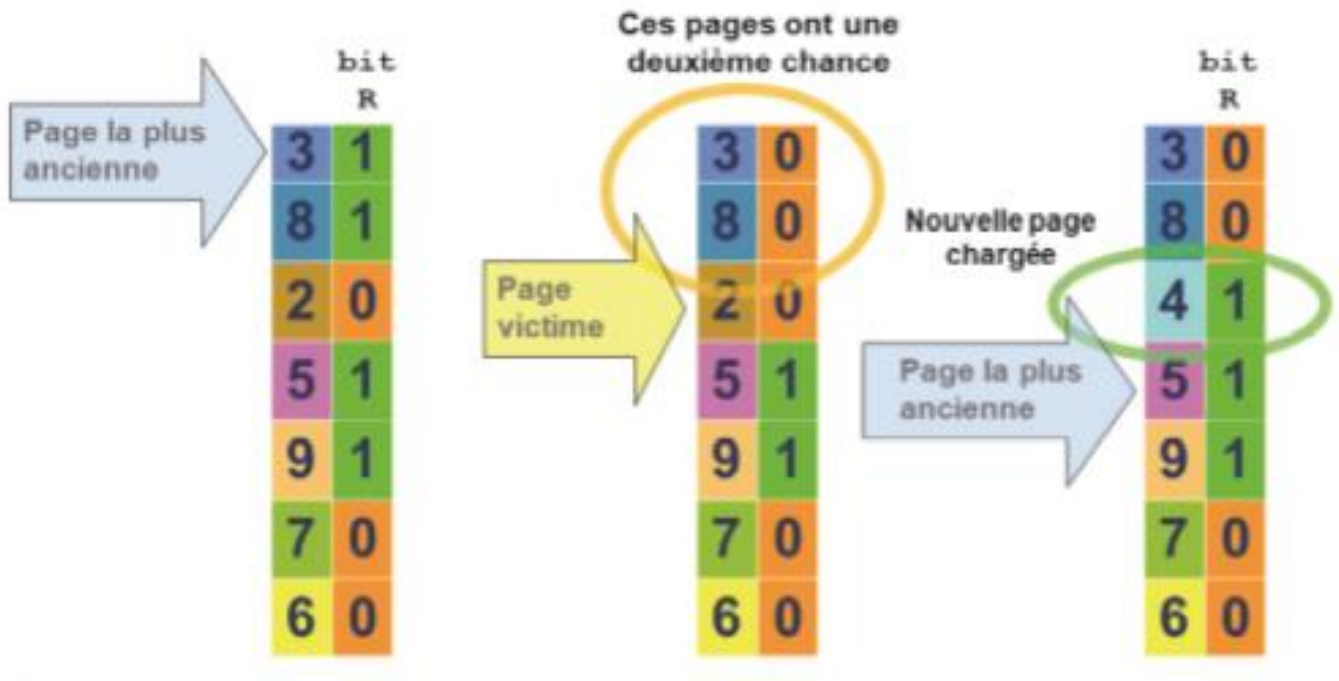
- Choisir la page la plus ancienne (queue FIFO)
- Si le bit R est à 1
  - La page est considérée comme nouvellement arrivée dans le FIFO.
  - Le bit R est remis à zéro (page non référencée).
- Sinon (bit R à 0), c'est la page à évincer.

ARP Seconde chance : Mise en oeuvre

- Utilisation d'un algorithme FIFO et du bit d'utilisation R
- Inspection du bit R de la plus vieille page
- Si R==0 (page non accédée récemment), on peut la remplacer
- Si R==1 la page est mise en fin de liste avec R=0
  - La page a droit à une deuxième chance
  - L'algorithme continue sur la prochaine page de la liste
- Si toutes les pages ont R à 1, alors retombe sur un FIFO

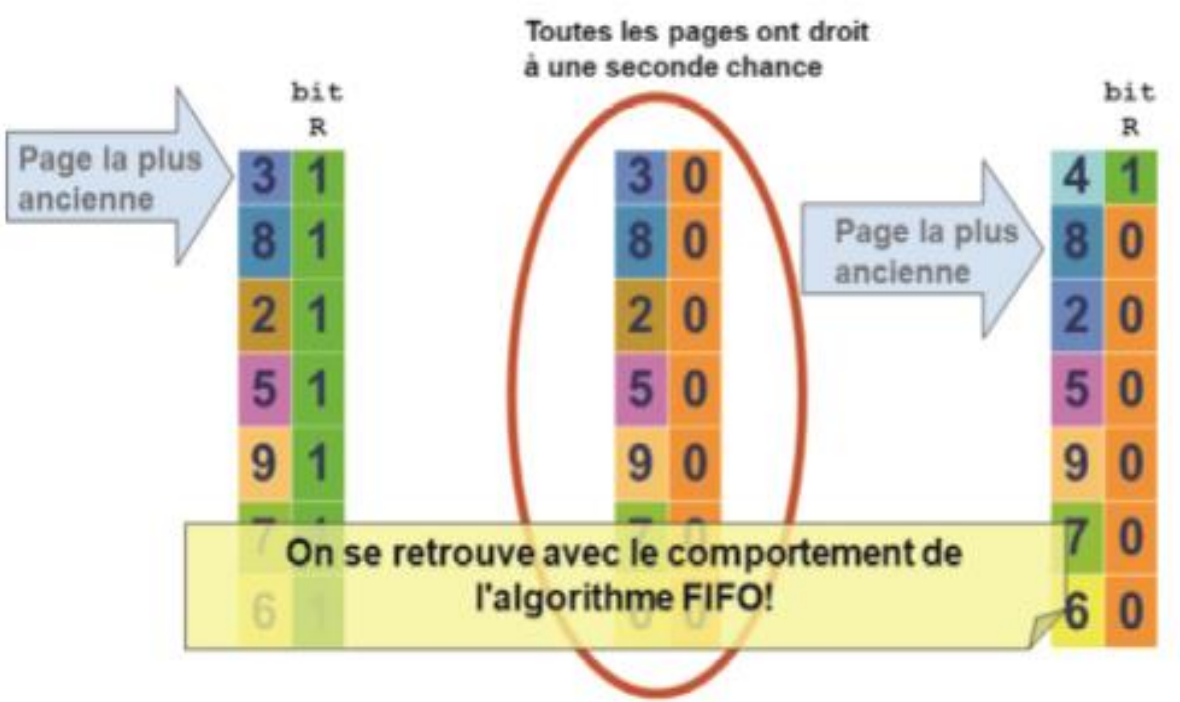
ARP Seconde chance : Mise en oeuvre

- **Cas de la page 4 référencée**



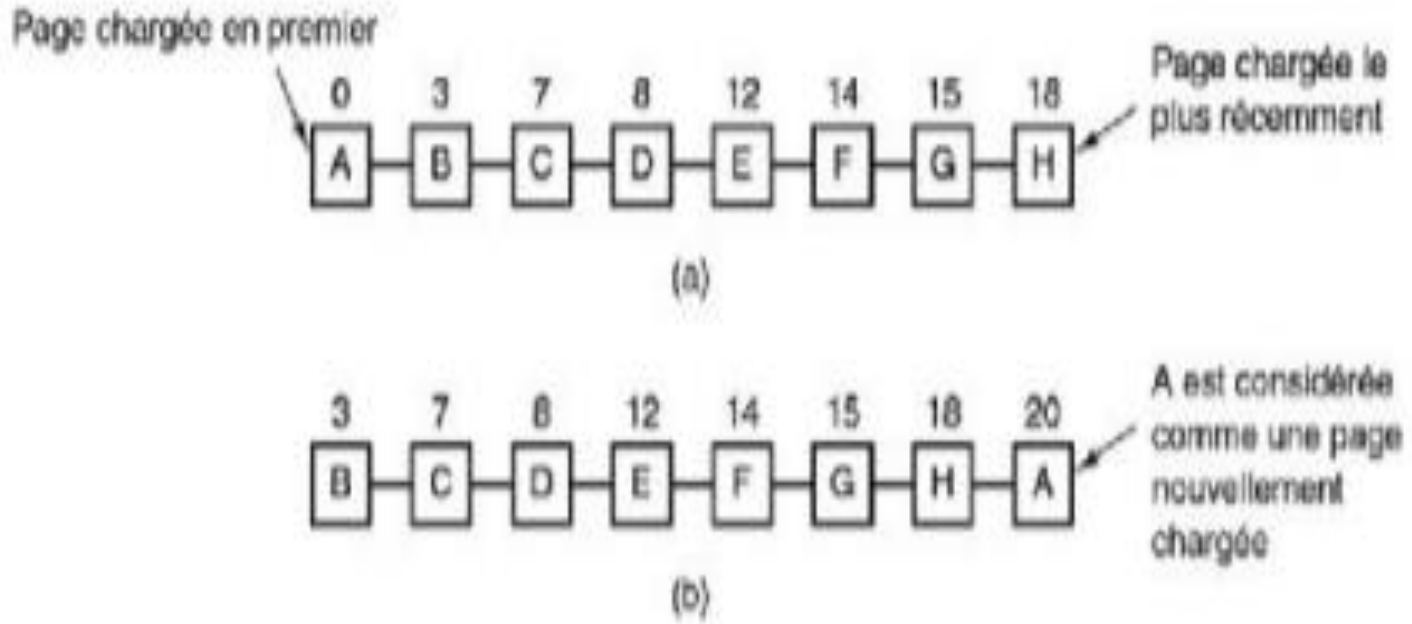
ARP Seconde chance : Mise en oeuvre

- **Autre cas de la page 4 référencée**



ARP Seconde chance

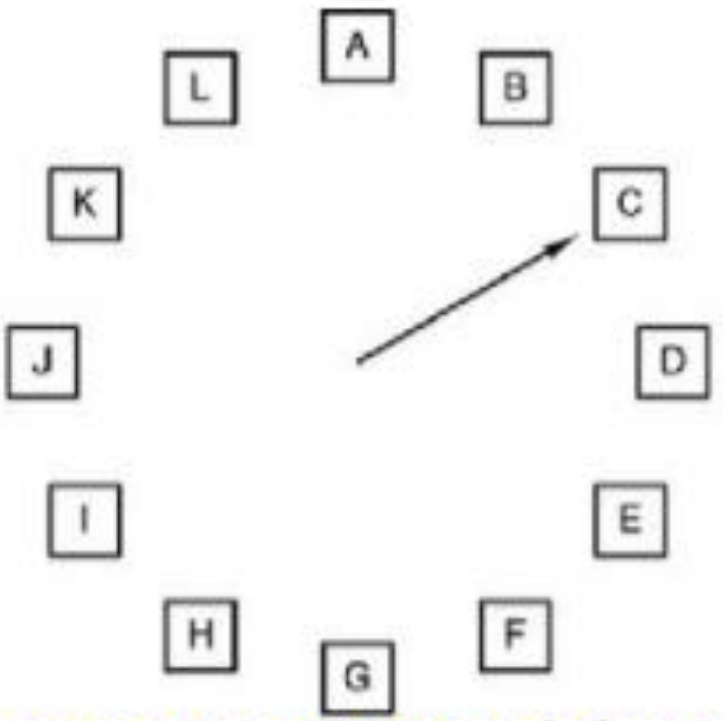
- **Liste chainée**



ARP Horloge : liste circulaire

- Peut être vue comme une queue circulaire où on avance sur les pages qui ont le bit à 1 (en le positionnant à zéro) jusqu'à ce que l'on trouve une page avec le bit d'utilisation à 0.
- Ceci est connu sous le nom de l'algorithme de l'horloge.

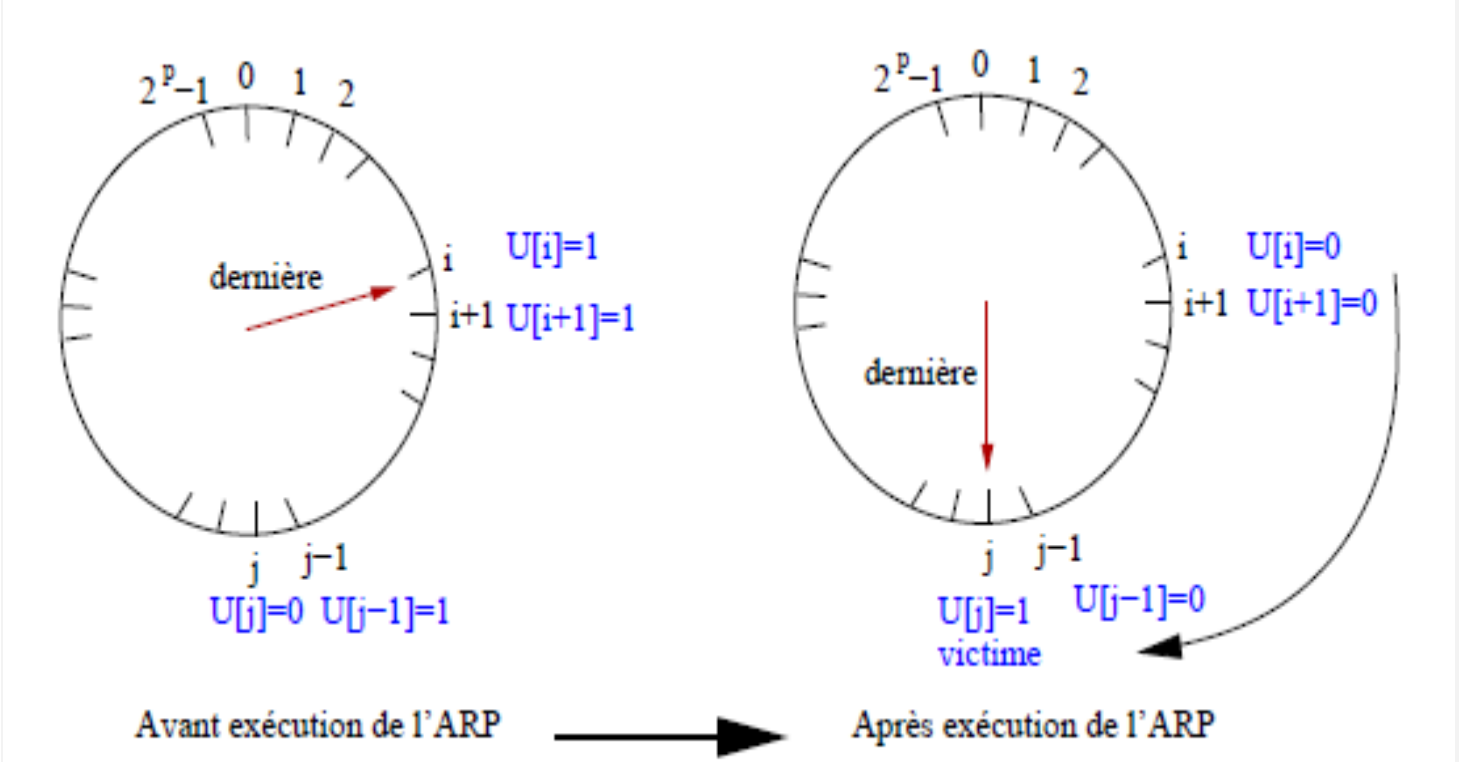
ARP Horloge : liste circulaire



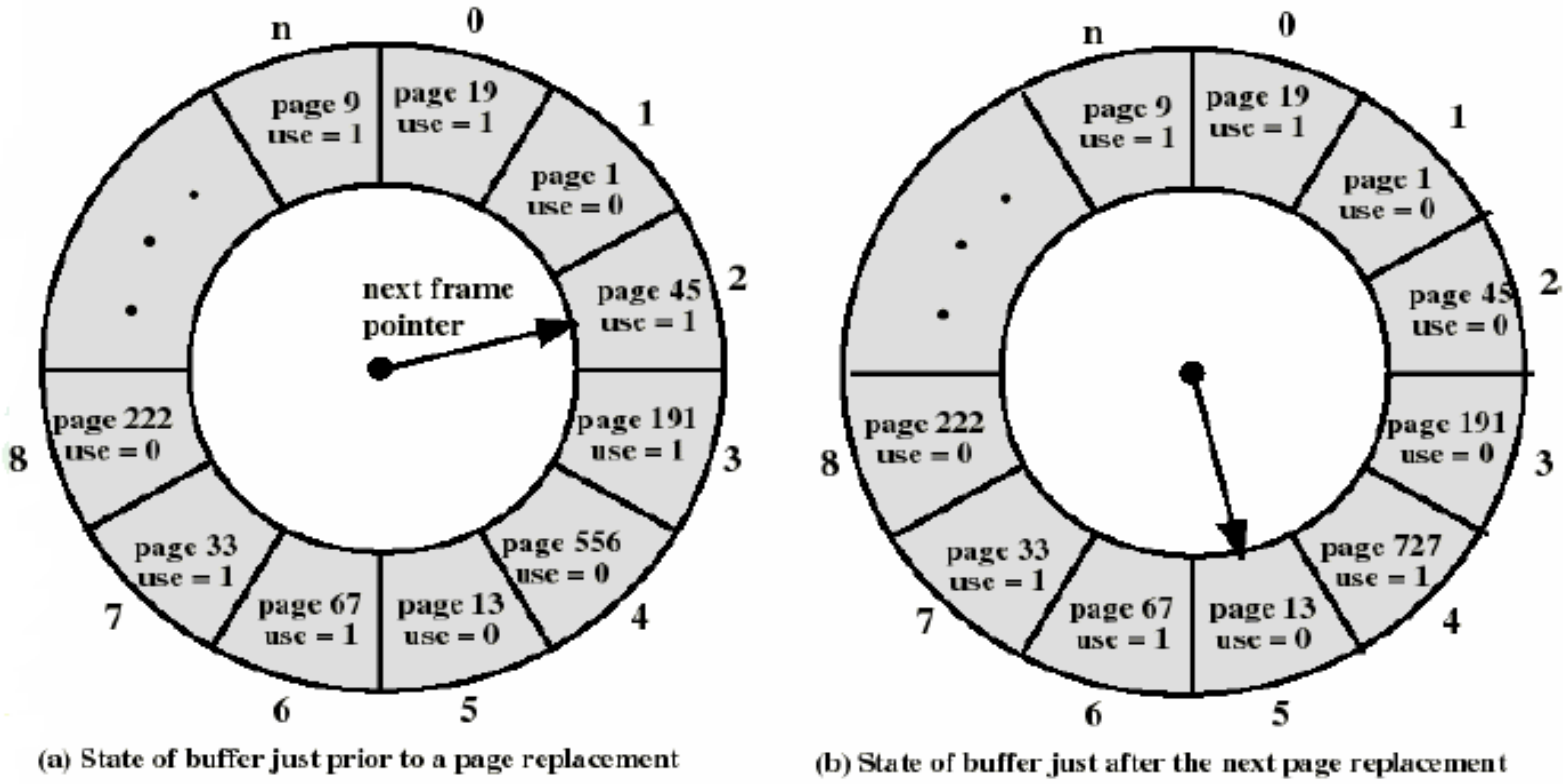
Quand un défaut de page se produit, la page pointée est testée. L'action entreprise dépend de la valeur du bit R :
R = 0 : retirer la page
R = 1 : mettre R à 0 et avancer le pointeur

Comme l'algorithme précédent mais l'utilisation d'une liste circulaire évite de déplacer les pages.

ARP Horloge : Illustration



ARP horloge : Illustration



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

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

La page 727 est chargée dans le cadre 4.

La proch. victime est 5, puis 8.

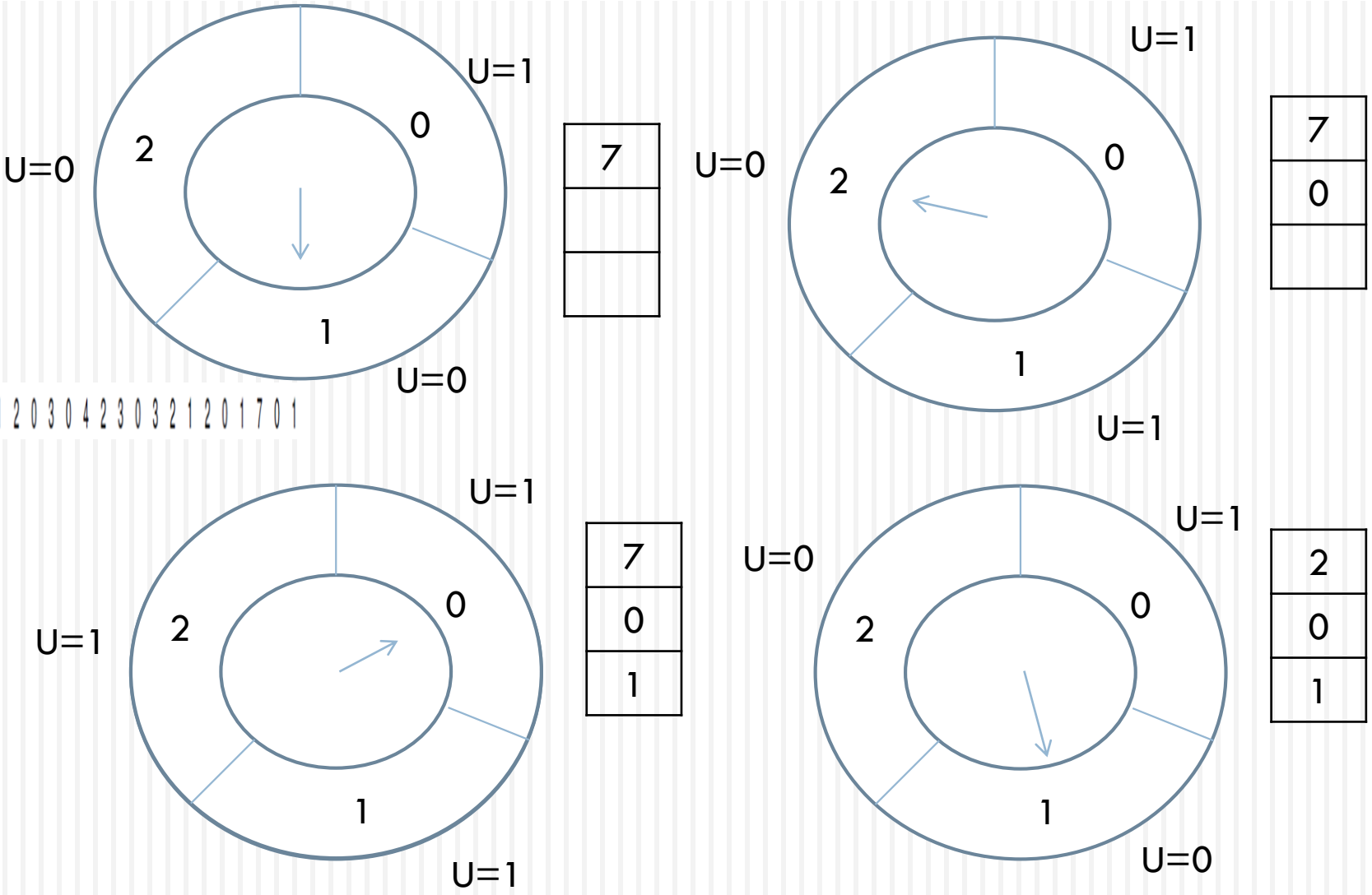
Algorihme de l'horloge

- L'algorithme de la deuxième chance fait beaucoup de manipulations de liste
  - Pas forcément efficace
- Utilisation d'une liste circulaire
  - Comme une horloge
  - Une « aiguille » pointe vers la page la plus ancienne
- Si *page fault*
  - La page pointée est inspectée
  - Si R==0
    - Page supprimée
    - Nouvelle page insérée à sa place
    - Aiguille avancée de 1
  - Si R==1
    - R=0
    - Aiguille avancée de 1
- Juste une implémentation différente de la deuxième chance

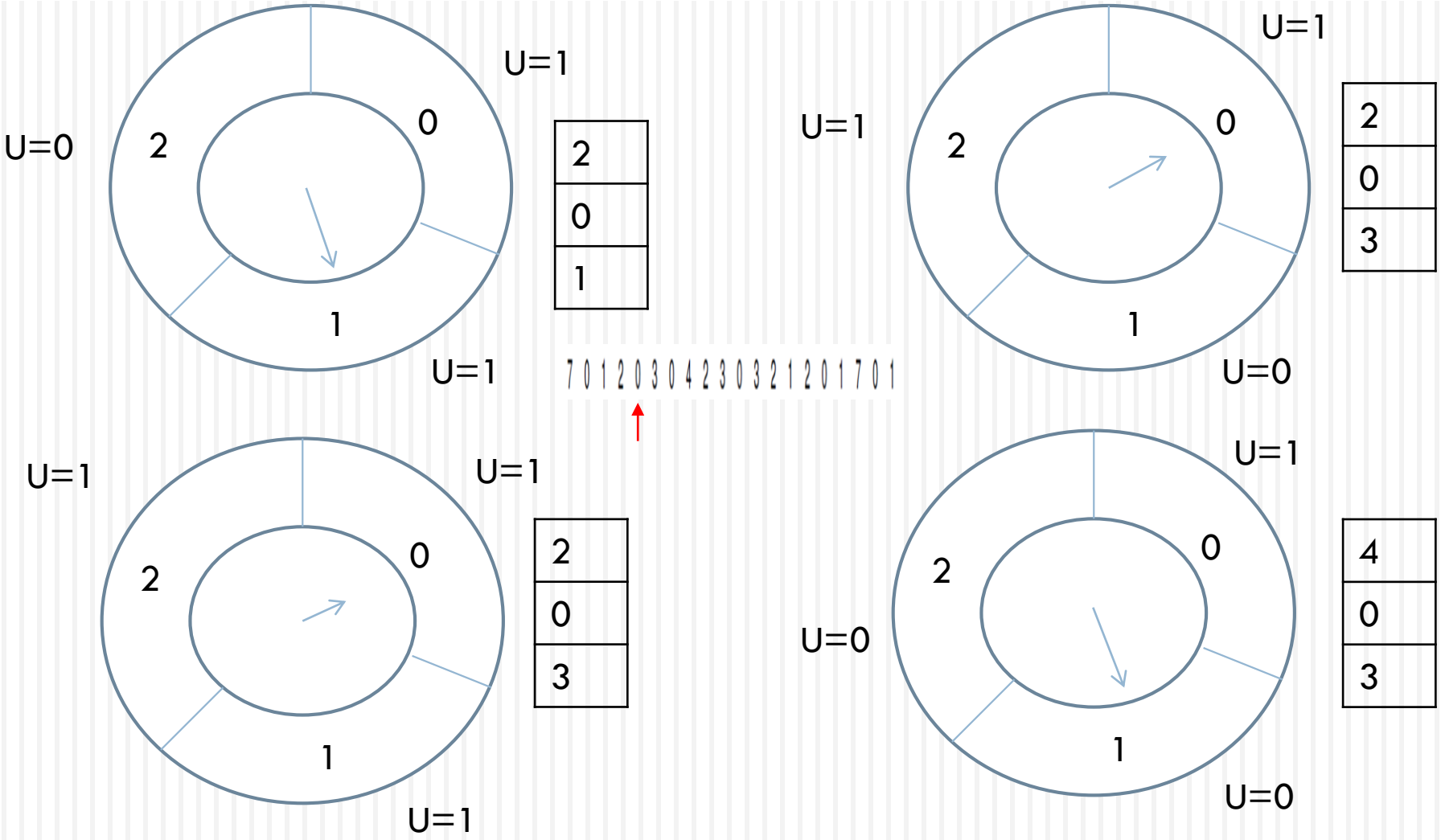
Exemple 1 : ARP Horloge

m \ ω	7	0	1	2	0	3	0	4	2	3	0	3	2	1	2	0	1	7	0	1
0																				
1																				
2																				

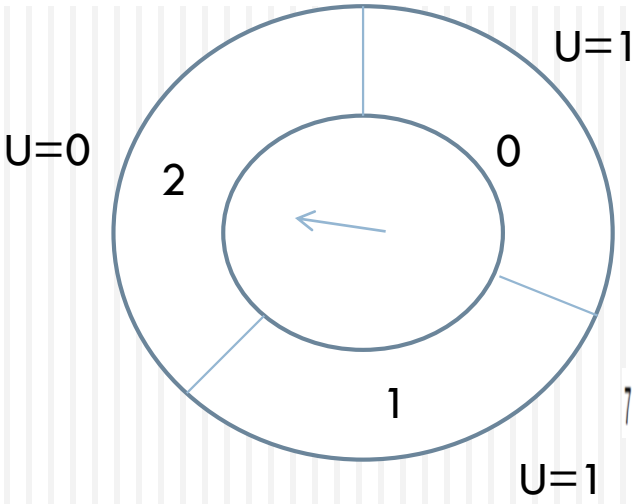
Exemple 1 : ARP Horloge



Exemple 1 : ARP Horloge

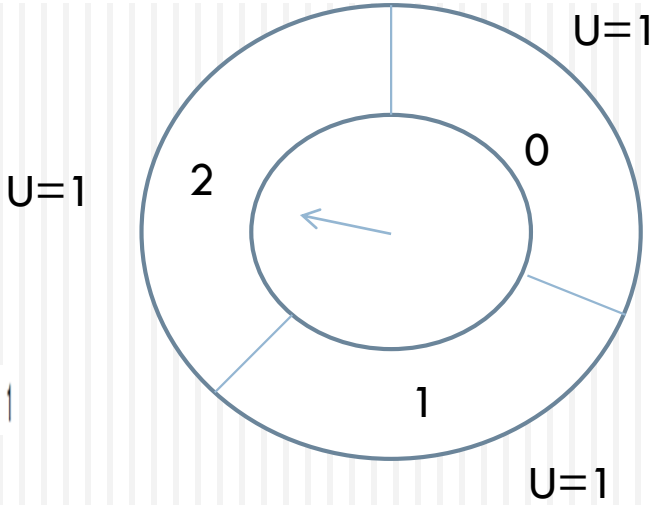


Exemple 1 : ARP Horloge

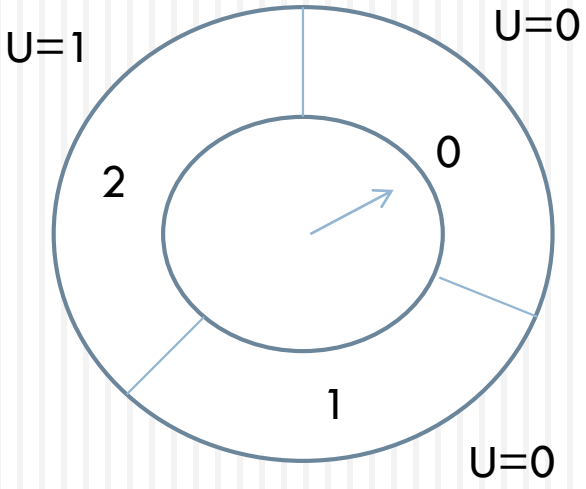


4
2
3

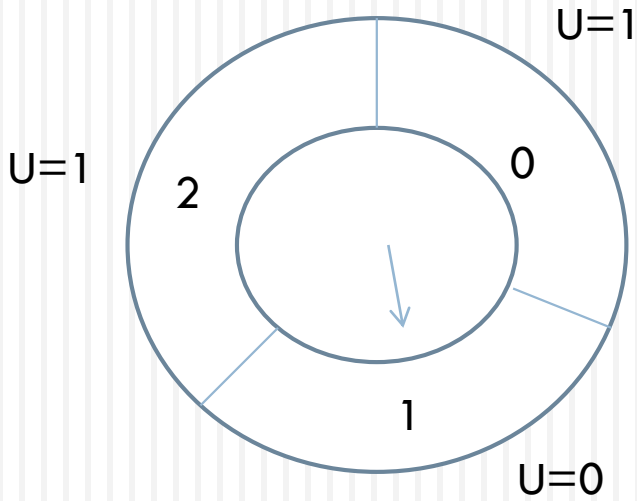
7 0 1 2 0 3 0 4 2 3 0 3 2 1 2 0 1 7 0 1



4
2
3

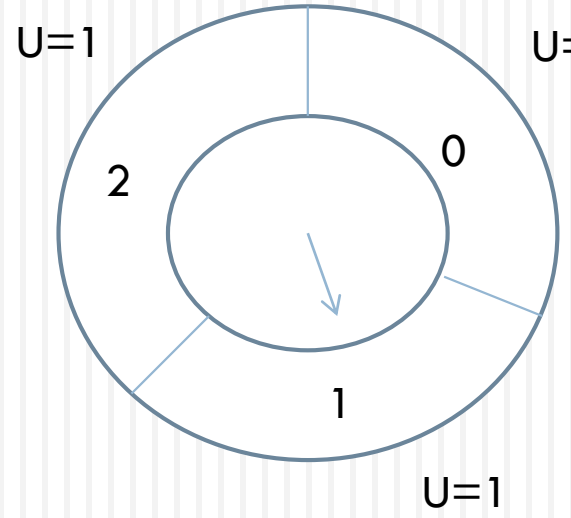


4
2
0

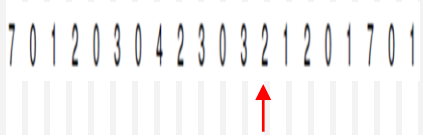


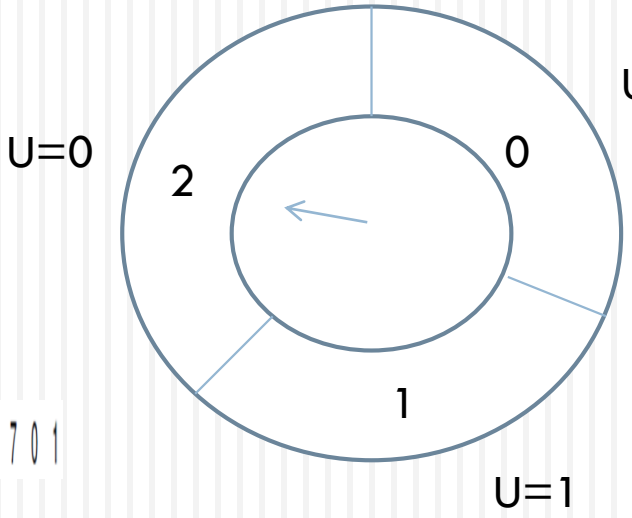
3
2
0

Exemple 1 : ARP Horloge

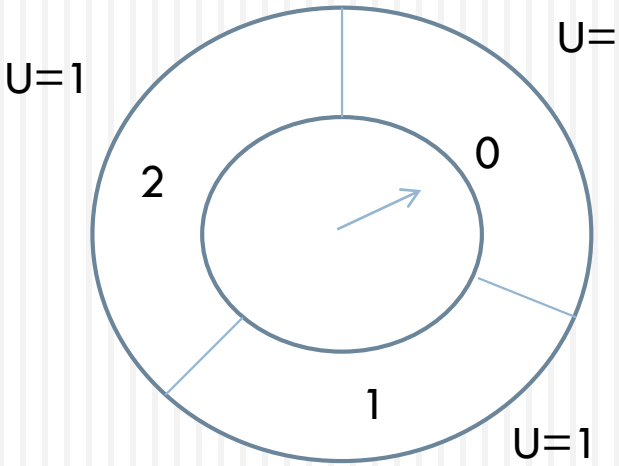


3
2
0

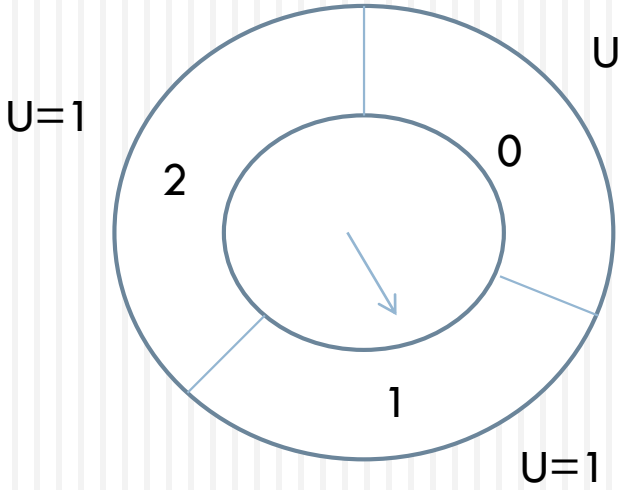




3
1
0

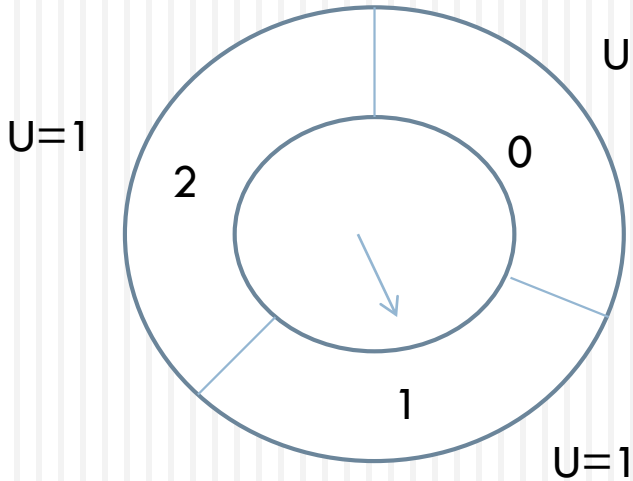


3
1
2

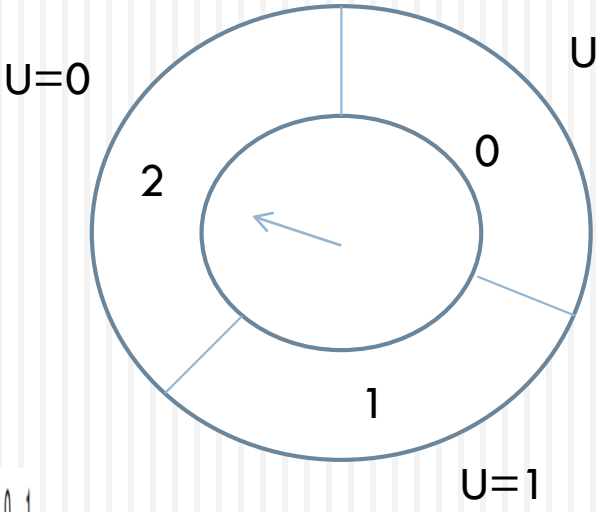


0
1
2

Exemple 1 : ARP Horloge

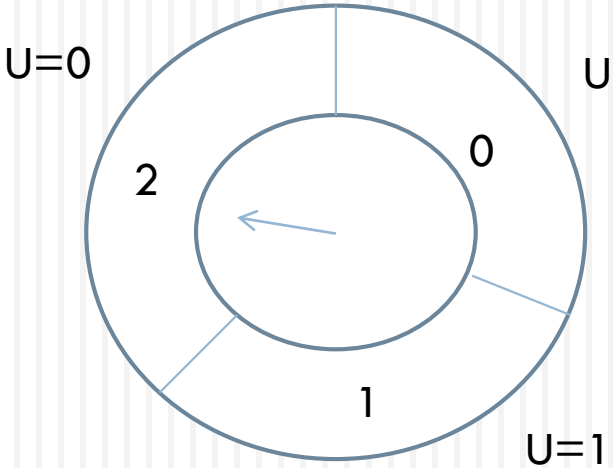


0
1
2

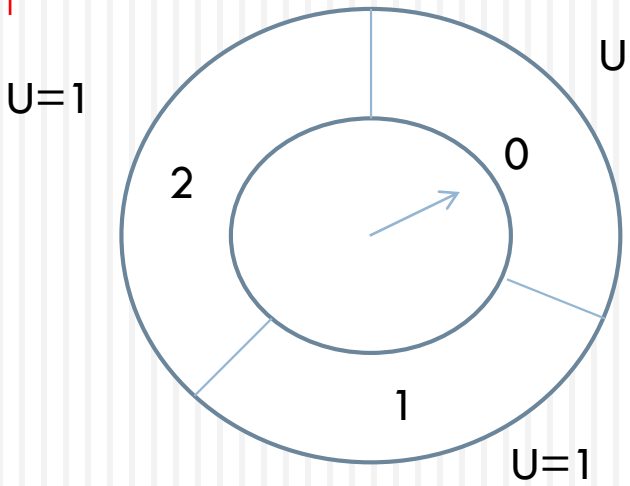


0
7
2

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



0
7
2

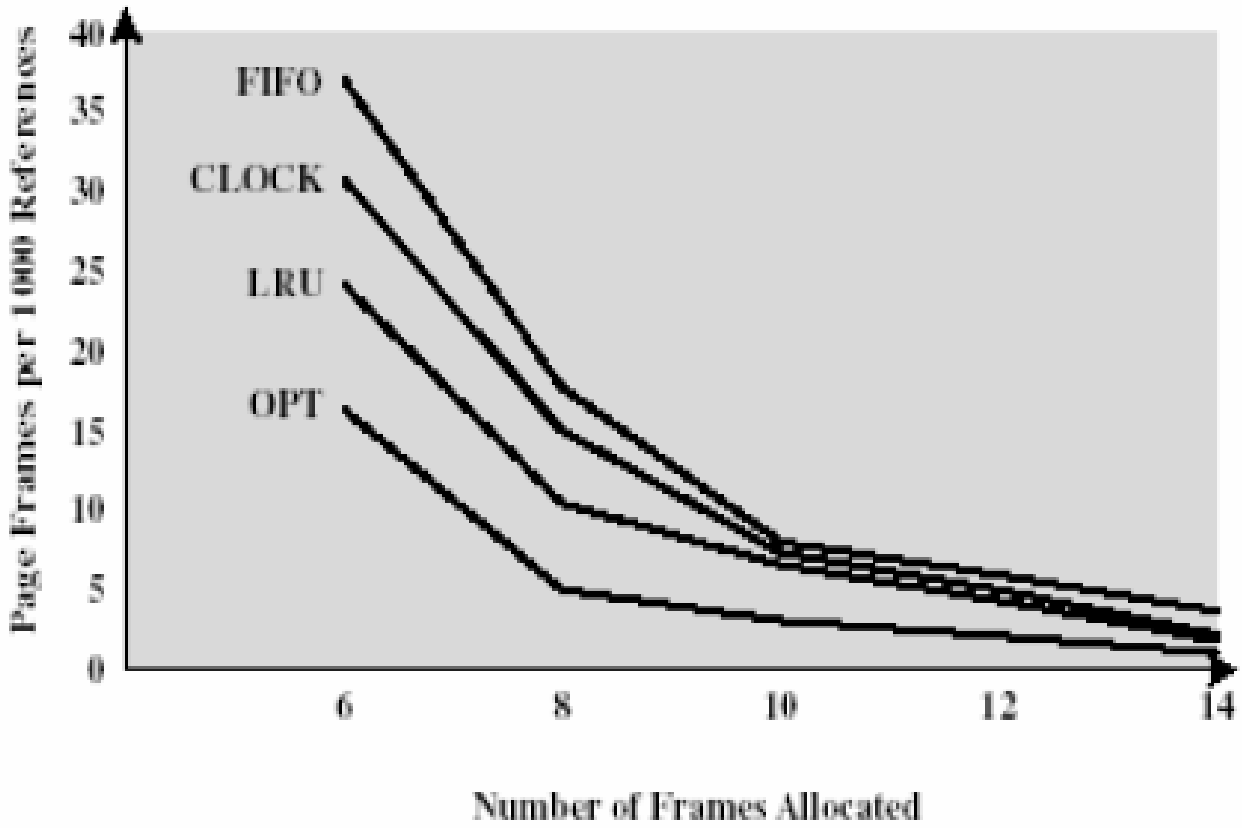


0
7
1

Exemple 1 : ARP Horloge

m \ w	7	0	1	2	0	3	0	4	2	3	0	3	2	1	2	0	1	7	0	1
0	7	7	7	2	2	2	2	4	4	4	4	3	3	3	3	0	0	0	0	0
1		0	0	0	0	0	0	0	2	2	2	2	2	1	1	1	1	7	7	7
2			1	1	1	3	3	3	3	3	0	0	0	0	2	2	2	2	2	1

ARP : Pas de grosse différence en pratique



Résumé des ARP

OPTIMAL	Le meilleur en principe mais pas implantable, utilisé comme référence
LRU	Excellent en principe, mais demande du matériel dispendieux
FIFO	Facile à implanter, mais peut écarter des pages très utilisées
Horloge CLOCK	Modification de FIFO vers LRU: évite d'écarter des pages récemment utilisées

Anomalie de Belady

- Il ne faut pas exagérer la portée de cette curiosité. Elle montre certes que l'algorithme FIFO n'a pas *en général* une propriété à laquelle on se serait attendu (ajouter de la mémoire réduit les défauts de page) mais elle ne montre pas qu'elle ne l'a pas *en moyenne*. Et de toute façon l'algorithme FIFO n'est jamais utilisé pour le remplacement de page.
- Par ailleurs, on peut démontrer que certains algorithmes de remplacement de pages (LRU par exemple) ne sont pas sujets à ce type d'anomalie.

Anomalie de Belady

« Il vaut mieux être riche et en bonne santé que pauvre et malade »

Cette affirmation n'est pas vraie dans tous les cas. Cependant l'existence de contre-exemples n'empêche pas l'affirmation d'être vraie presque tout le temps

- Anomalie de Belady (1969) : Il est raisonnable de penser que plus on dispose de mémoire moins on aura de défauts de page.

- Pour quelques algorithmes, dans quelques cas il pourrait avoir plus de défauts avec plus de mémoire !
  Exemple : FIFO, mais pas : LRU, OPT, HORLOGE

Anomalie de Belady

- L'**anomalie de Belady** est une anomalie de comportement observée pour l'algorithme de remplacement des lignes de cache FIFO. Augmenter le nombre de voies de la mémoire cache peut accroître le taux de défauts de cache. Ce phénomène n'est pas spécifique aux N-way Set mais est général à toutes les applications où l'algorithme FIFO est utilisé.
- Il est également observable en gestion des pages.

FIFO

Une intuition trompeuse :

"Avec davantage de mémoire vive, j'aurais moins de défauts de page".

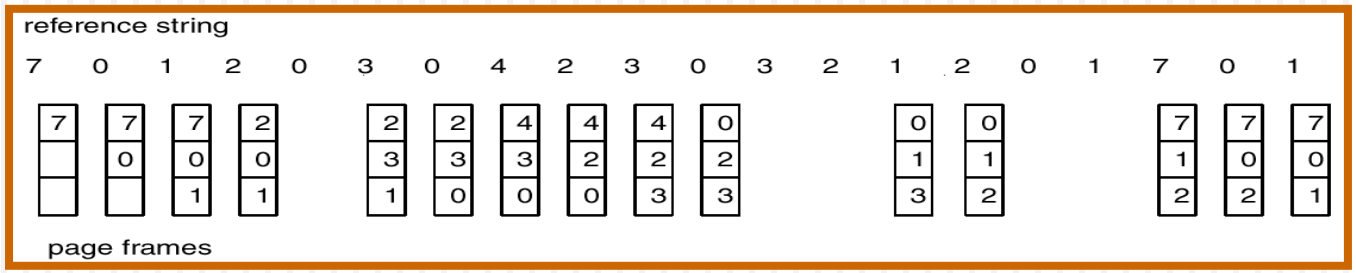
- Références mémoire : 1, 2, 3, 4, 1, 2, 5, 1, 2, 3, 4, 5
- 3 cadre de page (3 pages peuvent être en mémoire à un cer moment par processus)

1	1	4	5	
2	2	1	3	9 page faults
3	3	2	4	

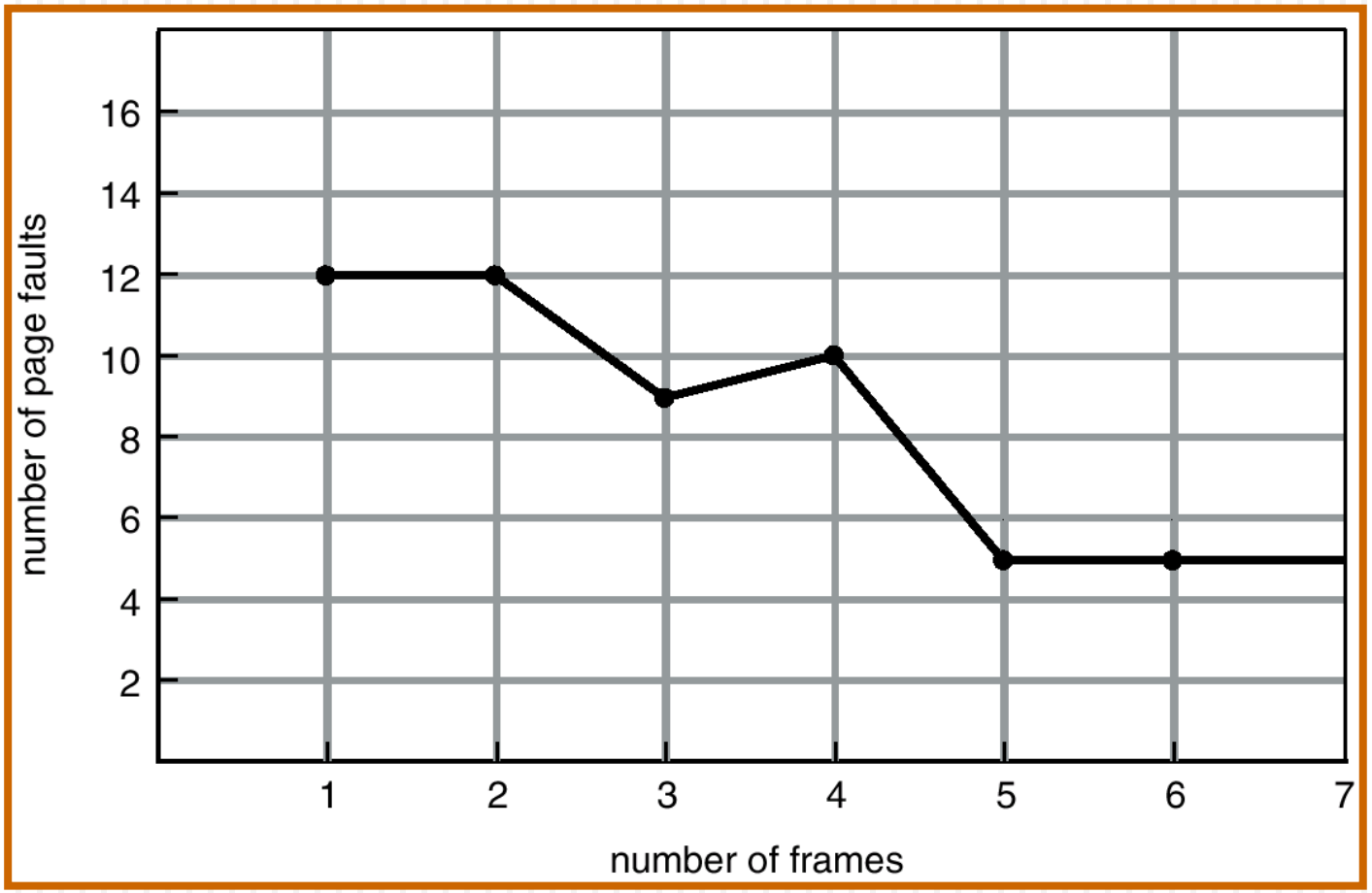
- 4 cadre de page

1	1	5	4	
2	2	1	5	10 page faults
3	3	2		
4	4	3		

- Remplacement FIFO – Anomalie de Belady
  - Plus de cadre de page ⇒ plus de defauts de pages



Anomalie de Belady avec FIFO



Anomalie de Belady

- Prenons l'exemple d'une mémoire cache associative de degré 3 et d'une seconde de degré 4. Comparons leurs résultats pour la séquence : 3 2 1 0 3 2 4 3 2 1 0 4. On suppose que ces lignes sont mappées sur le même ensemble.
- Voies accédées : 321032432104
  - Voie 1 : 330004444 4
  - Voie 2 : 2223333111
  - Voie 3 : 1112222200
- Voies accédées : 321032432104
  - Voie 1 : 333334444 00
  - Voie 2 : 22222233334
  - Voie 3 : 1111112222
  - Voie 4 : 000000111

(en rouge les défauts de cache)

Conclusion

- Le concept de mémoire virtuelle a marqué une étape scientifique importante dans l'évolution des systèmes d'exploitation et a, en grande partie, libéré le programmeur des contraintes de mémoire.
  - Sa mise en œuvre physique repose sur le mécanisme matériel de pagination qui permet de morceler la mémoire centrale et le programme, sur l'utilisation de disques pour stocker les processus à exécuter et sur un va-et-vient rapide entre disque et mémoire centrale pour transférer les parties de processus qui doivent être exécutées par l'unité centrale.
- Deux progrès appréciables en ont résulté.
  La mémoire virtuelle donne au système d'exploitation la faculté de gérer un grand nombre de processus concomitants en leur permettant de partager dynamiquement la mémoire centrale.
  - La concurrence entre processus et leur synchronisation pour l'accès à des informations communes rendent complexe la gestion de ce partage et en font un des problèmes techniques les plus difficiles des systèmes d'exploitation.
  - La mémoire virtuelle permet au programmeur d'écrire de grands programmes ou de manipuler des données de grande taille sans avoir désormais à gérer les entrées-sorties et les va-et-vient avec la mémoire secondaire. Il n'est alors plus contraint par la taille de la mémoire centrale. Il peut avoir l'illusion d'une mémoire quasi illimitée (par contre, si la taille de la mémoire centrale de l'ordinateur est petite devant celle du programme, ce qui peut se produire entre autres pour les objets connectés, cela se traduit pour le système d'exploitation par de nombreux va-et-vient et, pour l'application, par un temps d'exécution important).
- Voilà pourquoi la mémoire virtuelle et la pagination sont aujourd'hui présentes dans la plupart des systèmes d'exploitation des ordinateurs, que ce soit celui d'un téléphone portable, d'une tablette ou d'un objet connecté.

Thank's