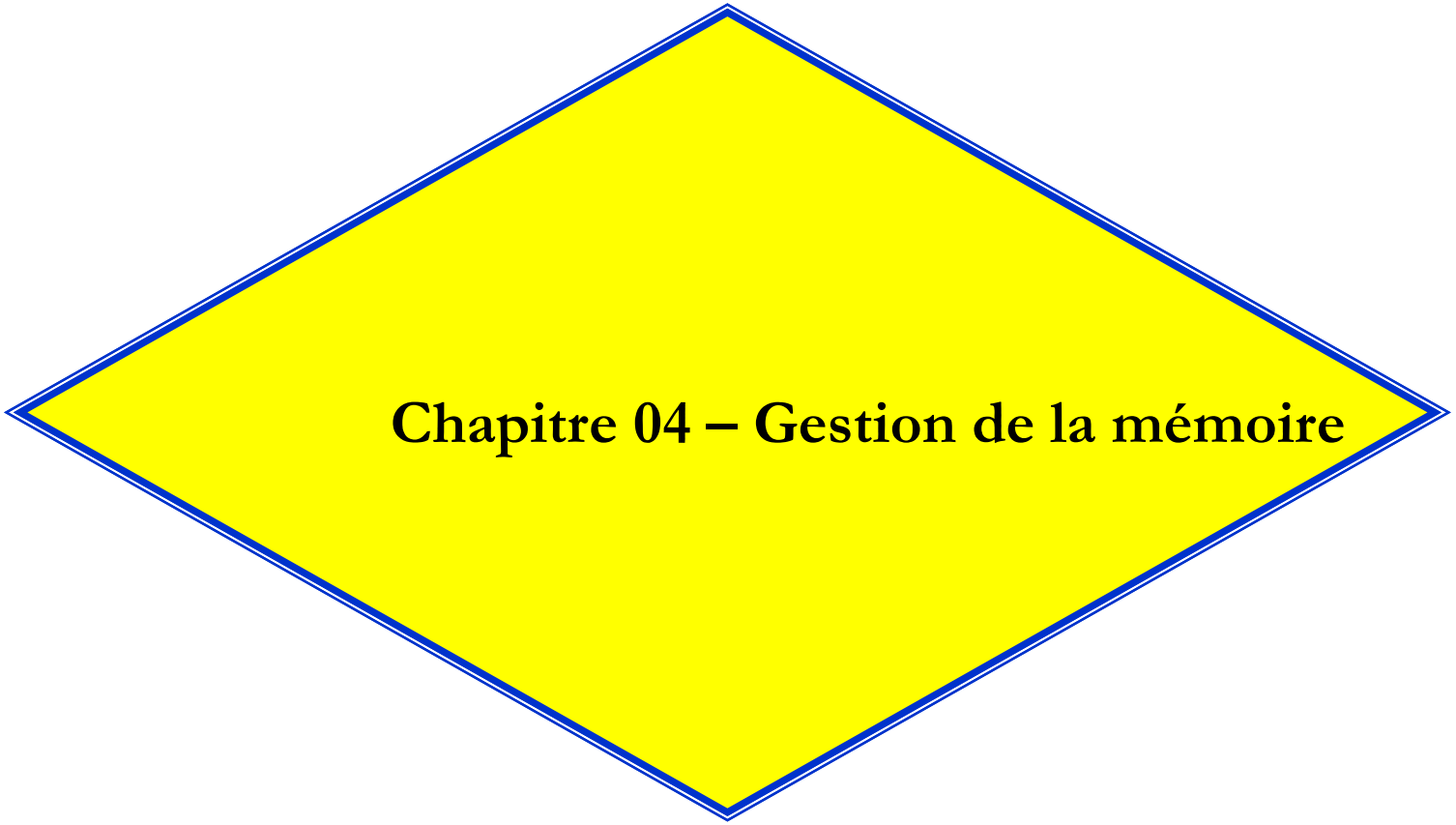
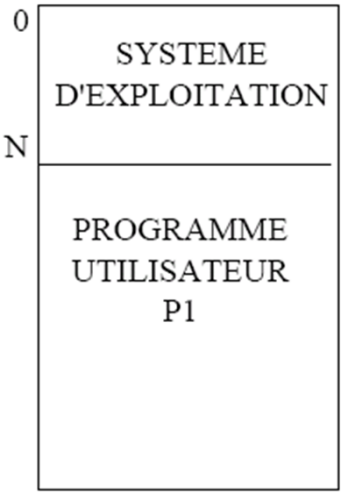


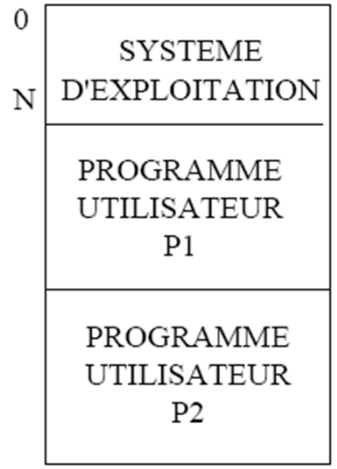
Chapitre 04 – Gestion de la mémoire

Gestion de la mémoire centrale : Introduction

- Introduction :
  - Dans un système *mono-programmé*,
    - la mémoire centrale est découpée en deux parties: une partie est réservée au système et le reste est attribué au processus.
    - Lorsque le processus est en attente de ressource, le processeur n'a rien à faire.



  - Dans un systèmes *multiprogrammé* :
    - la mémoire centrale est découpée en **plusieurs parties**, de façon à mettre plusieurs processus en mémoire au même moment, de telle sorte à avoir plusieurs processus candidats au processeur.



- On peut distinguer trois problèmes que doit résoudre le système.

1. Définir un espace d'adresses par processus.

   → Chaque processus doit conserver son indépendance.

   → Il doit pouvoir créer des objets dans son espace d'adresses et les détruire, sans qu'il y ait interférence avec les autres processus.

   → Les caractéristiques du matériel sont importantes, car elles offrent plus ou moins de souplesse pour cette définition.

2. Protéger les processus entre eux.

→ Il est nécessaire que "chacun reste chez soi", du moins pour les objets qui leur sont propres.

→ Il faut donc pouvoir limiter les actions des processus sur certains objets, en particulier interdire les accès à ceux qui *ne leur appartiennent pas*.

3. Attribuer un espace de mémoire physique.

→ Comme un objet n'est accessible au processeur que s'il est en mémoire physique, il faut "allouer" de la mémoire physique aux processus, et assurer la traduction d'une adresse d'un objet dans son espace propre en l'adresse en mémoire physique où il est situé.

Rôles du gestionnaire de la mémoire
« MMU : Memory Management Unit »

Rôles du gestionnaire de la mémoire

- Toute instruction ou donnée de l'espace d'adressage doit être chargée en mémoire centrale (principale, physique) avant d'être traitée par un processeur.

- Le gestionnaire de la mémoire est le composant du système d'exploitation qui se charge de gérer l'allocation d'espace mémoire aux processus à exécuter :

  → **Comment organiser la mémoire** ? (une ou plusieurs partitions, le nombre et la taille des partitions fixes ou variables au cours du temps).

  → **Faut-il allouer une zone contiguë** à chaque processus à charger en mémoire ? Faut-il allouer tout l'espace nécessaire à l'exécution du processus entier ? (⇔ politique d'allocation).

  → **Comment mémoriser l'état de la mémoire**? Parmi les parties libres en mémoire, lesquelles allouées au processus? (⇔ politique de placement)

  → **S'il n'y a pas assez d'espace en mémoire, doit-on libérer de l'espace en retirant des parties ou des processus entiers**? Si oui lesquels ? (⇔ politique de remplacement)

  → **Les adresses figurant dans les instructions sont-elles relatives?** Si oui, comment les convertir en adresses physiques.

  → **Si plusieurs processus peuvent être résidents en mémoire**, comment assurer la protection des processus (⇔ éviter qu'un processus soit altéré par un autre?)

Exigences

1. *Efficacité : la mémoire doit être allouée équitablement et à moindre coût tout en assurant une meilleure utilisation des ressources (mémoire, processeurs et disque).*
2. *Protection : Les processus ne peuvent pas se corrompre.*
3. *Transparence :* Chacun processus doit ignorer l'existence des autres en mémoire.
4. *Relocation : la possibilité de déplacer un processus en mémoire (lui changer de place)*

Espace d’adressage d’un processus

Espace d'adressage d'un processus

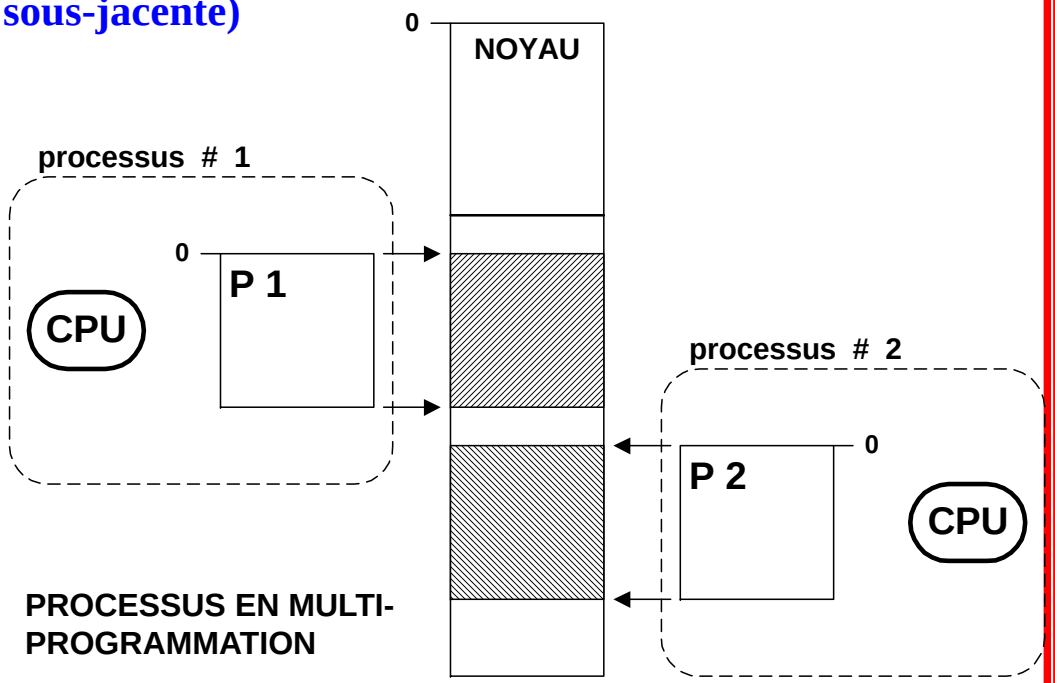
- **L'espace d'adressage d'un processus est constitué de l'ensemble des adresses auxquelles le processus a accès au cours de son exécution.**
- **Il est constitué « au moins » de trois parties :**
  1. **Le code du processus**
  2. **Les données du processus**
  3. **Sa pile d'exécution (**pour variables locales aux fonctions) (Le « tas » pour allocations dynamiques)
- **L'espace d'adressage associé à un processus s'appelle espace d'adresses logiques ou virtuelles (indépendante de la mémoire physique sous-jacente)**

Translation d'adresse : logique → physique (quand ?)

- Pendant la compilation
- Pendant le chargement
- Pendant l'exécution
- Dynamique (matériel spécial)

Problèmes:

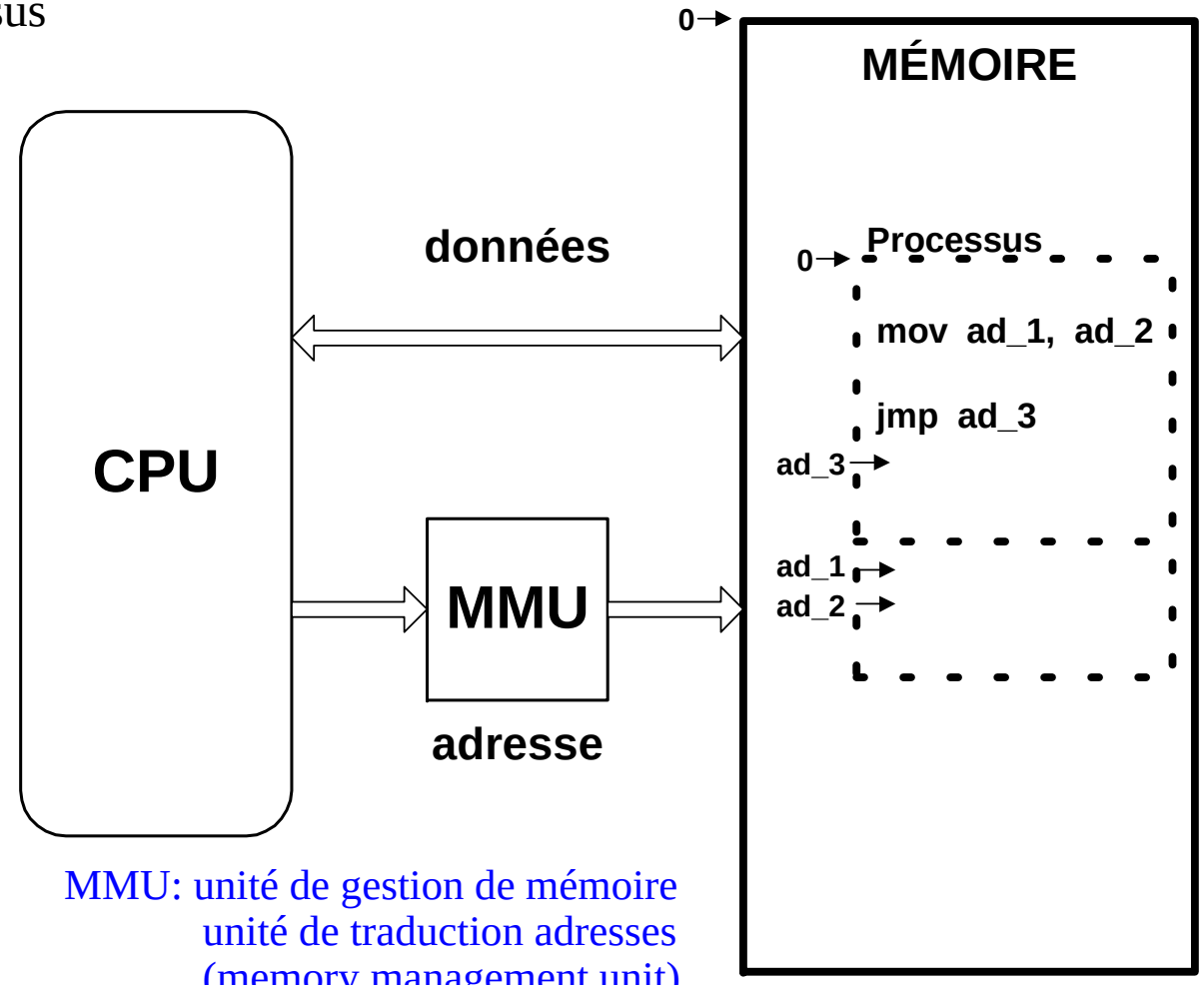
- Protection
- performance



ADRESSE LOGIQUE VS ADRESSE PHYSIQUE

- À quoi correspondent les adresses dans le code en mémoire ?
  - Le programme source contient des adresses symboliques
  - Relatives au processus

- Doivent être traduites (translation d'adresse) : C'est l'Unité de Gestion de Mémoire qui est responsable de la traduction entre les adresse logiques et physiques.



MMU: unité de gestion de mémoire
unité de traduction adresses
(memory management unit)

Liaison d’adresses

Liaison d'adresses-1-

- *La liaison entre les adresses logiques et physiques peut se faire à des moments variés du traitement*

1. Pendant la compilation :
   - Si on connaît au moment de la compilation l'endroit où le processus résidera, on peut générer un code **absolu.**
   - **Exemple :** si on sait a *priori* qu'un processus utilisateur résidera à partir de l'emplacement R, le code généré par le compilateur commencera à s'étendra de ct endroit.
   - Si plus tard, l'endroit de début change, il est nécessaire de recompiler ce code
   - **Exemple :** programmes au format **.COM MS/DOS**

Liaison d'adresses-2-

2. Pendant le chargement :

- Si au moment de la compilation, on connaît pas l'endroit où le processus résidera en mémoire, le compilateur doit générer un code **translatable.**
- La liaison finale est alors reportée au moment de chargement
- Si **l'adresse de début change**, alors seul un nouveau chargement du code utilisateur est requis pour intégrer cette nouvelle valeur
- *Le chargeur peut générer :*
  - Adresses physiques (code exéc. **non relogeable)**
  - Adresses logiques (code exéc. **Relogeable ⇔ programme pas linké avec des adresses absolues)**
    - ⇨ A l'exécution, La liaison logique – physique utilise un matériel spécial (MMU)

3. Liaison pendant l'exécution :

- Si le processus peut-être déplacé d'un segment de la mémoire ***à un autre pendant son exécution,*** la liaison peut être reportée au moment de l'exécution

- Pour que cette méthode puisse s'appliquer, il est nécessaire de disposer d'un matériel spécial pour réaliser (la translation dynamique entre adresses logiques et physique, chargement dynamique, recouvrement, permutation)

Politiques d'Allocation

Politiques d'Allocation mémoire -1-

- Fonction attendue :
  - Allouer-zone (t : taille) → adresse
  - Libérer-zone (a : adresse, t : taille)
- Objectifs :
  - → Optimiser l'utilisation de la mémoire (limiter la fragmentation)
  - → Optimiser les algorithmes d'allocation / libération

Politiques d'Allocation mémoire -2-

1. Implémentation statique : correspondance entre l'adresse virtuelle « logique » et l'adresse physique une fois pour tout :
   - Un programme en mémoire à la fois.
   - Mais nécessité de partage de la mémoire entre plusieurs programmes
   - Techniques :
     - ⇨ Zones contiguës de taille fixées
     - ⇨ Zones contiguës de taille variables

   ☞ **Allocations contiguë ?**
   - les allocations se font par **zones** indivisibles de P blocs de N mots ayant des adresses consécutives

Politiques d'Allocation mémoire -3-

2. Réimplantation dynamique : Correspondance entre l'adresse logique « virtuelle » et l'adresse physique variable dans le temps
   - Par zones d'adresse (⇔ segments)
   - Par pages d'adresses
   - Techniques :
     - ⇨ Zones ***« non contiguës »*** de taille fixe (systèmes paginés)
     - ⇨ Zones ***« non contiguës »*** de taille variable (systèmes segmentés)

Politiques d'Allocation non contiguë de la mémoire centrale

Introduction

- La taille d'un processus doit pouvoir dépasser la taille de la mémoire physique disponible, même si l'on enlève tous les autres processus.

- En 1961, J. FOTHERINGHAM proposa le principe de la mémoire virtuelle :
  - ⇨ le SE conserve en mémoire centrale les parties utilisées des processus et stocke, si nécessaire, le reste sur disque.

- **Mémoire virtuelle et multiprogrammation se complètent bien** : un processus en attente d'une ressource n'est plus conservé en MC, si cela s'avère nécessaire.

- La mémoire virtuelle fait appel à deux mécanismes :
  1. Segmentation
  2. Pagination.
     - ⇔ La mémoire est divisée en segments ou pages.

- Sans recours à la mémoire virtuelle, un processus est entièrement chargé à des adresses contiguës ; avec le recours à la mémoire virtuelle, un processus peut être chargé dans des pages ou des segments non contigus.

Plan du chapitre

Généralités :
- **Principe**
- **Conversion**
- **Mémoire virtuelle**

Systèmes paginés
- **Algorithmes de remplacement de page**
  - **NRU**
  - **FIFO**
  - **LRU**

Systèmes segmentés

Systèmes combinés

Généralités -1-

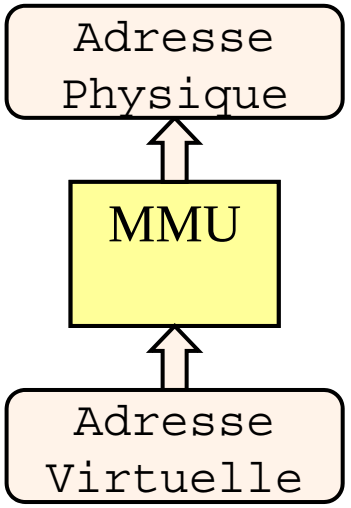
A. Principe :

- Dans les méthodes décrites jusqu'à présent, le seul procédé de conversion utilisé est la translation.
  - ➢ Ce procédé ne s'applique plus lorsqu'on désire une allocation dynamique non contiguë.

- Il faut cette fois :
  - ☞ *Dissocier complètement* l'ensemble des adresses référencées par un processus en cours d'exécution de l'ensemble des adresses physiques disponibles en mémoire centrale (plus de translation),
  - ☞ Créer, entre l'espace logique du processus et l'espace physique, une correspondance qui sera réalisée dynamiquement pendant l'exécution de ce processus (⇔ Table des pages)

Généralités -2-

- Remarque :
  - Sur les ordinateurs **sans mémoire virtuelle**, Les adresses logiques sont directement placées sur le bus de la mémoire et provoquent la lecture ou l'écriture du mot à l'adresse spécifiée.
  - Lorsque la **mémoire virtuelle est utilisée**, les adresses virtuelles ne sont pas directement placées sur le bus de la mémoire.
    - Elles sont envoyées à *l'Unité de gestion de la mémoire* ou *MMU (memory management 'Unit)*, composant qui traduit les adresses virtuelles en adresses physiques comme le montre la figure



Généralités -3-

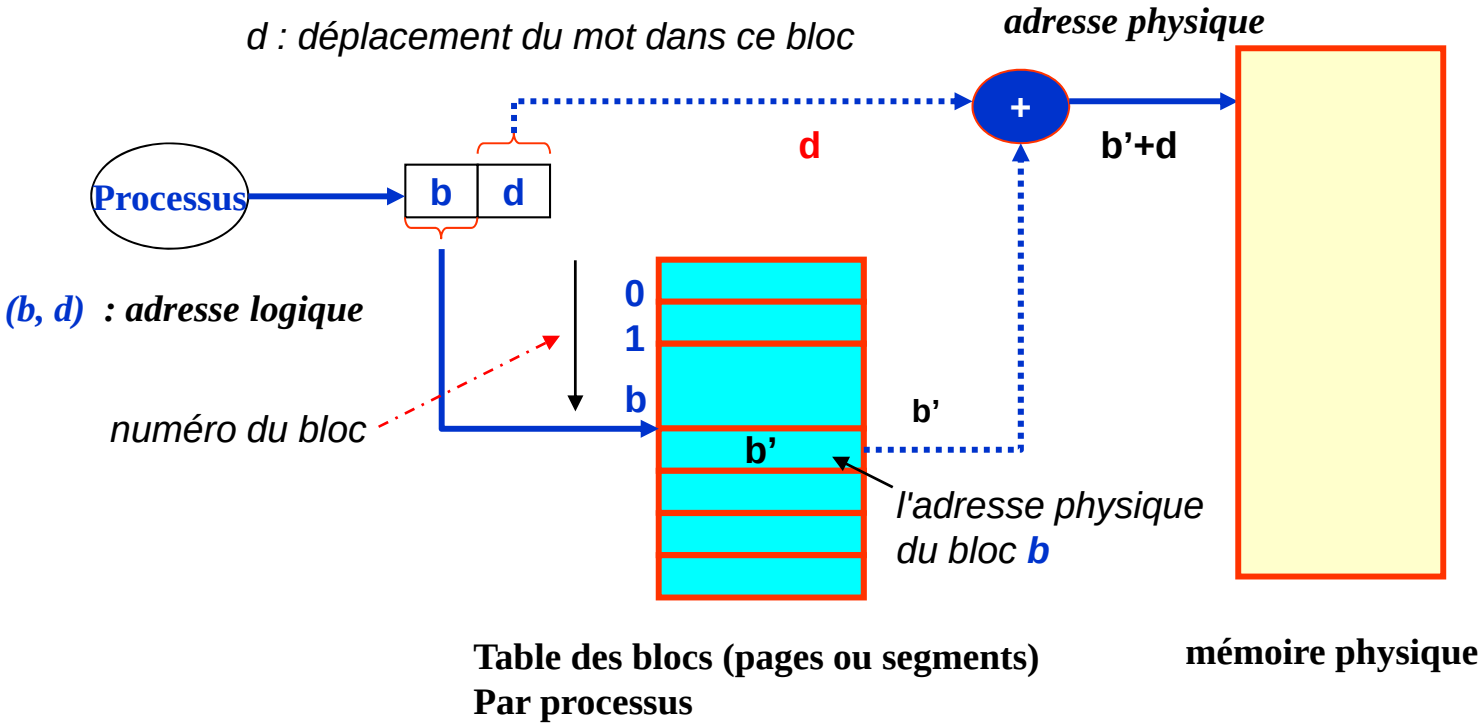
- Deux choix fondamentaux concernent le caractère fixe ou variable de la taille des zones.

  1. pour les systèmes paginés, les zones sont toutes de **même taille** et sont appelées **pages,**
  2. pour les systèmes segmentés, les zones sont de **taille variable** et sont appelées **segments.**

- **N.B :**
  - Certains systèmes d'exploitation ***combinent les deux approches***, par exemple en utilisant des segments composés eux-mêmes de pages (**Segmentation paginée)**.

B. Principe de la correspondance : conversion.

- N.B : le terme de bloc est ici utilisé aussi bien pour désigner une page dans un système paginé qu'un segment dans un système segmenté.



- Remarque :
  - Les procédés de conversion utilisés par les divers systèmes d'exploitation suivent globalement ce schéma de base.
  - Signalons que la conversion est réalisée de façon matérielle

- Principe de la correspondance : conversion – suite -
  - Le couple **(b, d)** représente l'adresse logique d'un mot dans un bloc
    - **b** : est le numéro du bloc
    - **d** : le déplacement du mot dans ce bloc, c'est-à-dire sa position à partir du début.
  - Le calcul de l'adresse physique se fait de la façon suivante:
    - Chaque processus dispose **d'une table de blocs** dans laquelle les numéros de blocs sont **rangés par ordre croissant**.
    - L'entrée de la table correspondant au bloc b contient l'adresse physique b' de ce bloc.
    - L'adresse physique r du mot est finalement obtenue en ajoutant le déplacement dans le bloc : ***r = b' + d.***

Généralités -6-

C. **Mémoire virtuelle :** l'organisation non contiguë de la mémoire centrale se distingue de celles que nous avons envisagées jusqu'à présent par trois aspects :

1. la dissociation des adresses logiques et physiques (**point 1)**,,
2. le fractionnement de l'espace logique d'un processus en blocs (**point 2)**
3. la non-contiguïté de l'allocation.(**point 3)**

Généralités -6- (point 1)

- **point 1 :**
  - Signifie que les espaces logiques et physiques peuvent être de **tailles différentes**.
    - l'espace logique du processus peut être soit plus petit que l'espace physique, mais également lorsqu'il est plus grand.
  - L'ensemble des adresses logiques d'un processus dépasse alors la taille de la mémoire physique.
    - Nous retrouvons ***le principe de mémoire virtuelle***, (⇔ systèmes avec recouvrement ou chargement dynamique)

Généralités -6- (Point 2)

- **Point 2 :**
  - Lorsque la taille d'un processus n'est plus limitée par la taille de la mémoire centrale, il faut évidemment prévoir comment réaliser ***l'exécution de ce processus***.
    - ☞ La question est résolue très simplement **grâce au point 2** : le processus sera exécuté en plusieurs étapes, chacune ne chargeant que les brocs utiles.

Généralités -6- (Point 3)

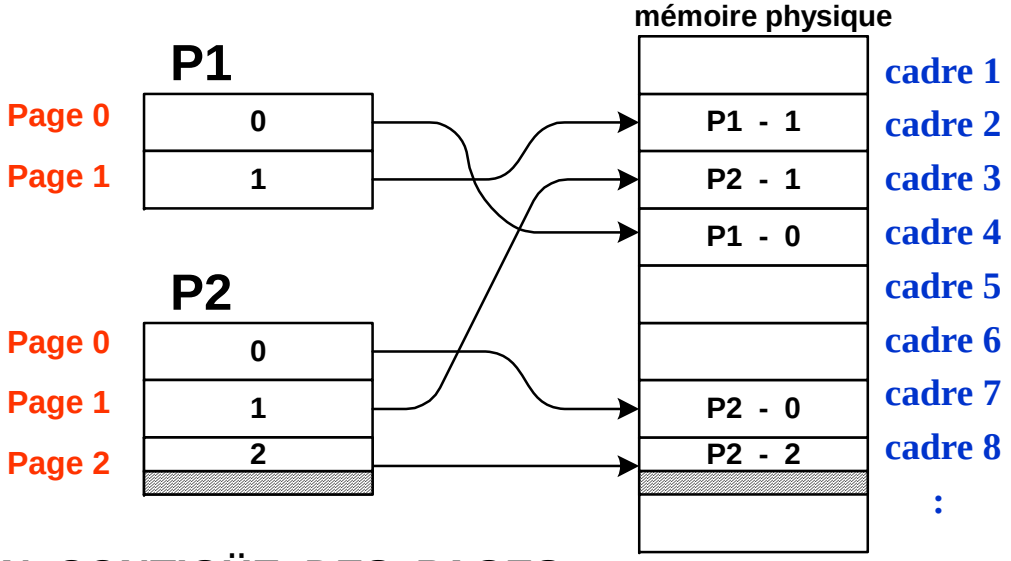
- **Point 3**
  - Le point 3 permet alors *d'insérer ces blocs* à tout endroit disponible de la mémoire centrale.
    - ☞ les blocs d'espace virtuel qui ne sont pas situés en mémoire centrale doivent avoir une représentation physique.
    - ☞ Là encore, c'est une mémoire auxiliaire (disque) qui leur servira de support.

Systèmes paginés

- Généralités :
  - Principe
  - Conversion
  - Mémoire virtuelle
- Systèmes paginés
  - Algorithmes de remplacement de page
    - NRU
    - FIFO
    - LRU
- Systèmes segmentés
- Systèmes combinés

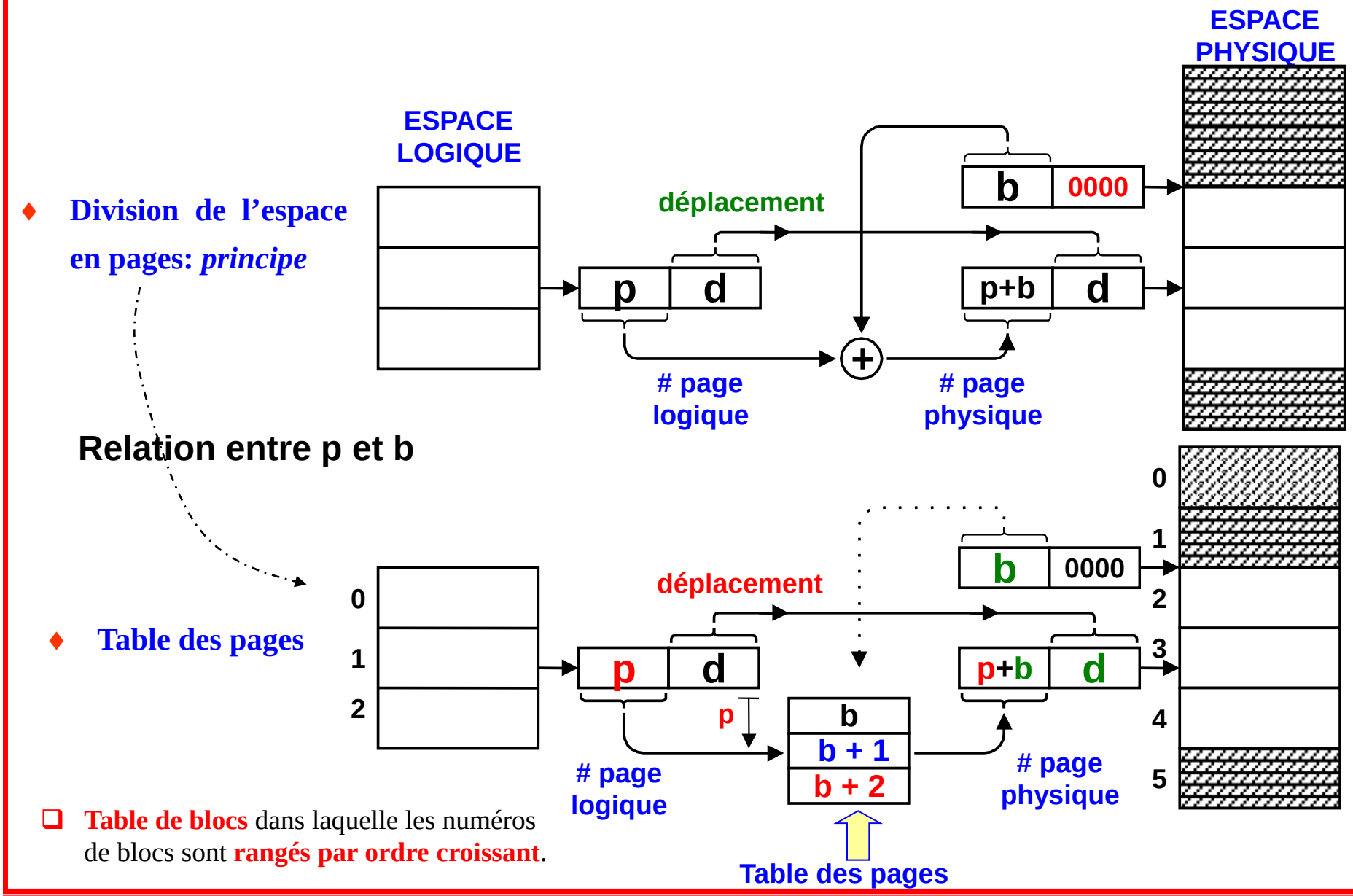
A. Conversion adresse logique – adresses physique :

1. On va examiner d'abord le cas où les blocs sont de taille fixe: **ce sont des pages.**
2. La mémoire physique est elle-même divisée en **blocs de même taille que les pages**, appelés **cadres de pages (cases)**
   → La taille d'un cadre (une case) est égale à la taille d'une page
3. Dans ce contexte, charger un programme en mémoire centrale consiste à placer les pages dans n'importe quelle case disponible.



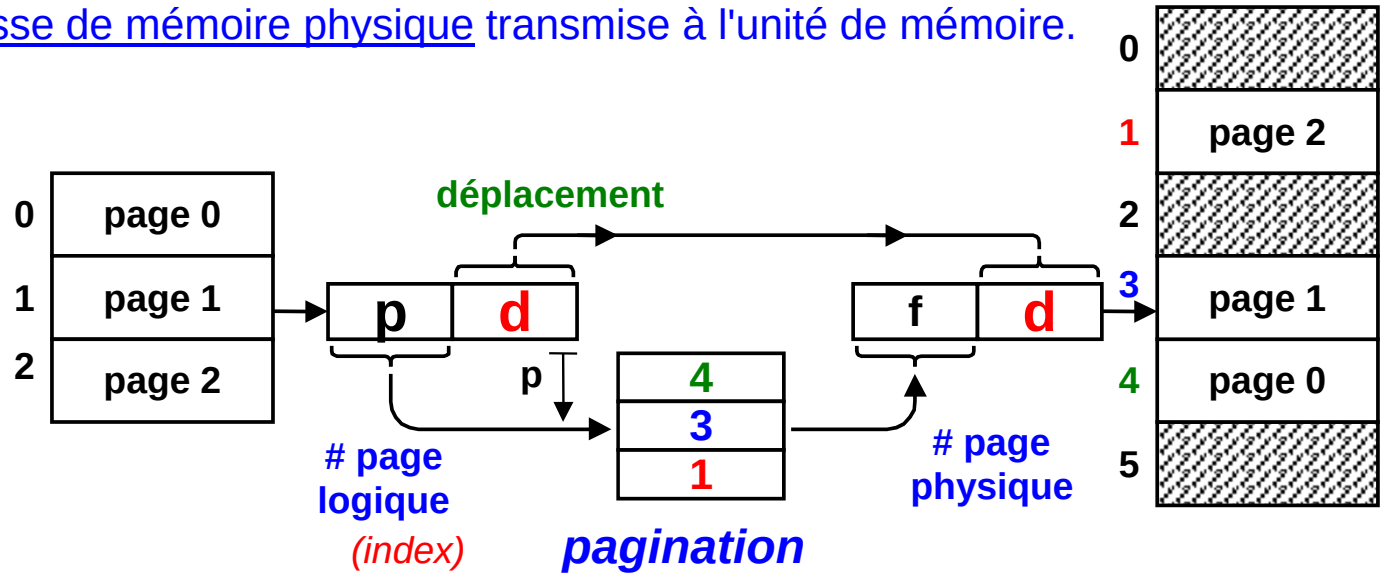
ALLOCATION NON CONTIGÜE DES PAGES

Systèmes paginés : Division de l'espace en pages



- **Table de blocs** dans laquelle les numéros de blocs sont **rangés par ordre croissant**.

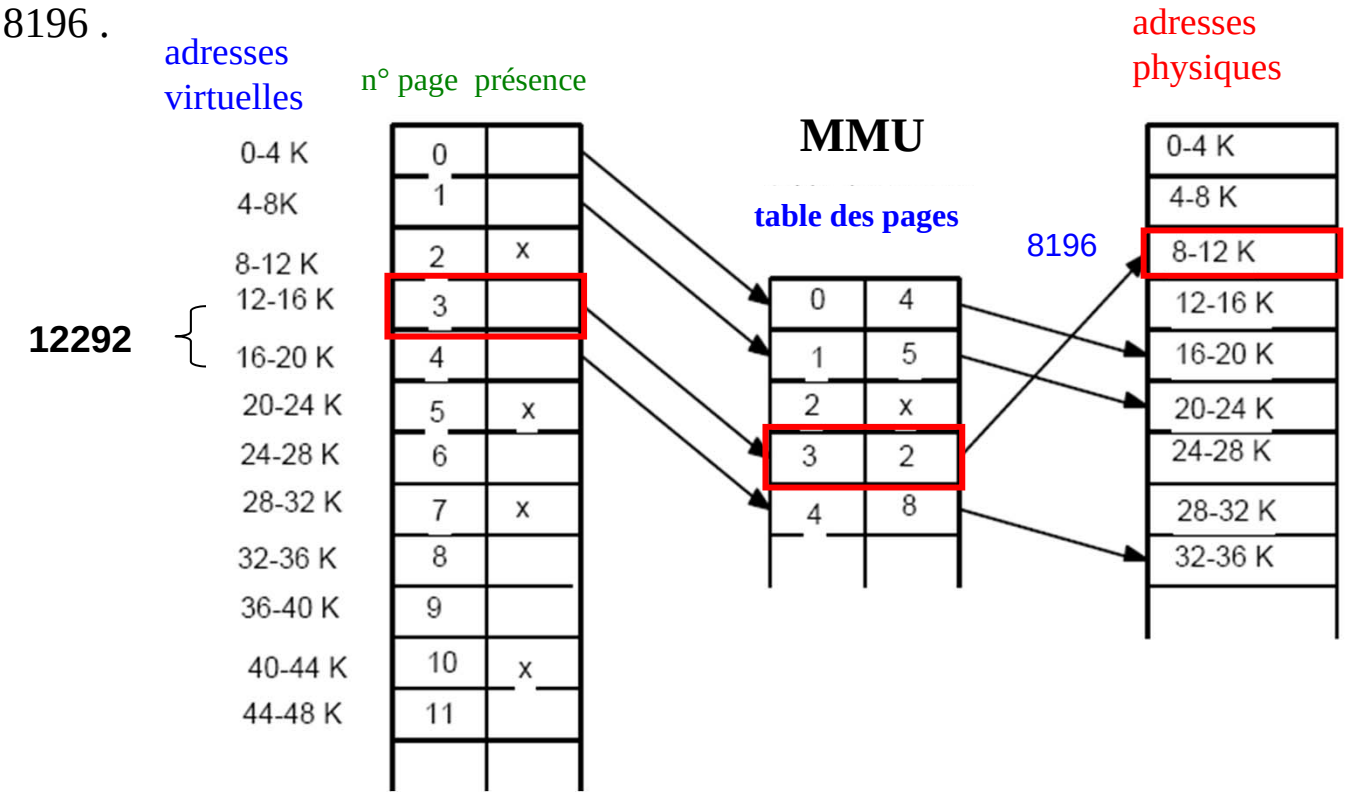
- **Support matériel de la pagination :**
  - → Chaque adresse logique générée par l'unité centrale est divisée en deux éléments:
    - ✓ un numéro de page *(p)*
    - ✓ et un déplacement dans la page *(d)*.
  - → Le numéro de page « **sert d'index »** dans une table de pages.
  - → La table de pages contient *l'adresse de base* de chaque page dans **la mémoire physique.**
  - → Cette adresse de base est combinée au déplacement dans la page pour définir l'adresse de mémoire physique transmise à l'unité de mémoire.



Conversion adresses logiques – adresses physiques – rôle de la MMU

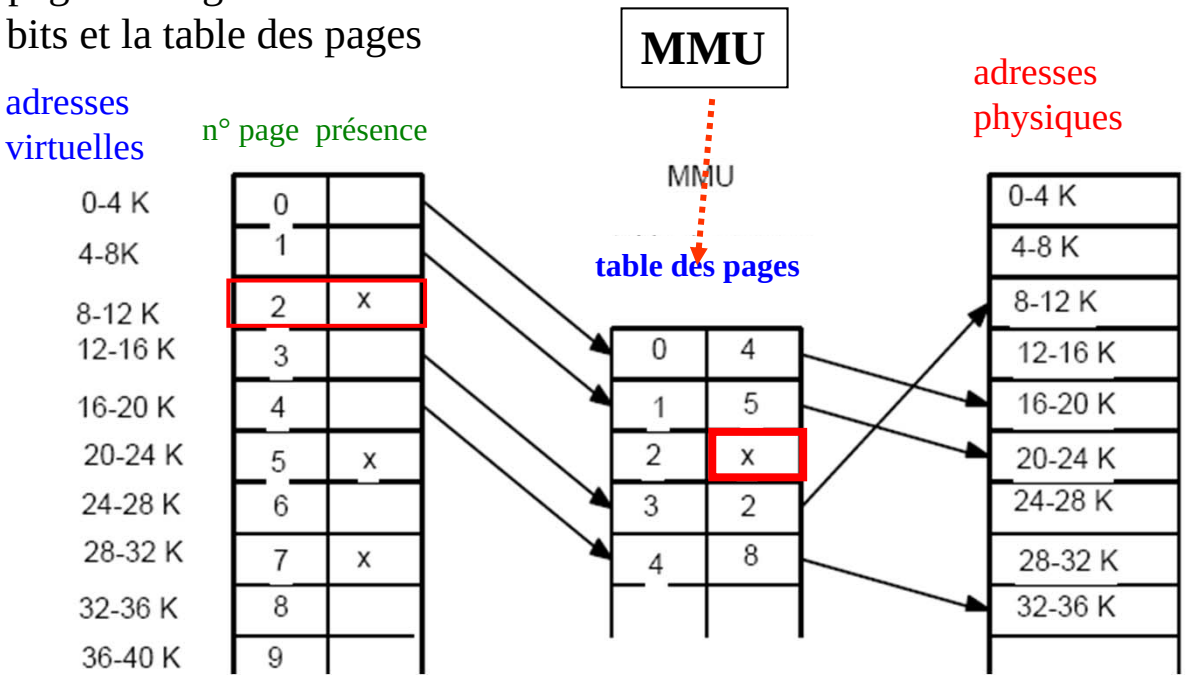
- la MMU (Memory Management Unit ou unité de gestion de la mémoire) traduit les adresses virtuelles en adresses physiques.
- La MMU *mémorise* :
  - ☞ les cadres physiques alloués à des processus (*sous forme d'une table de bits de présence*)
  - ☞ les cadres mémoire alloués à chaque processus (*sous forme d'une table des pages*)
- On dira qu'une page est **mappée** ou **chargée** si elle est physiquement présente en mémoire.

- **Exemple :**
  - Dans cet exemple, les pages ont une taille de 4 Ko.
  - L'adresse virtuelle 12292 correspond à un déplacement de **4 octets** dans la page virtuelle **3** (car **12292** = 12288 + 4 et 12288 = 12*1024).
  - La page virtuelle **3** correspond à la page physique **2.**
  - L'adresse physique correspond donc à un déplacement de 4 octets dans la page physique 2, soit : (8*1024) + 4 = 8196 .



Défaut de page ?

- Remarques :
  - la page **virtuelle 2** n'est **pas mappée**. Une adresse virtuelle comprise entre **8192** et **12287** donnera lieu à un *défaut de page*.
  - Il y a défaut de page quand il y a un accès à une adresse virtuelle correspondant à une page non mappée. (*En cas de défaut de page, un déroutement se produit (trap) et le processeur est rendu au SE).*
  - *Le système doit alors effectuer les opérations suivantes (voir partie mémoire virtuelle):*
    1. Déterminer la page à charger
    2. Déterminer la page à décharger sur le disque pour libérer un cadre
    3. Lire sur le disque la page à charger
    4. Modifier la table de bits et la table des pages



Modèle de pagination des mémoire logique et physique

Modèle de pagination des mémoire logique et physique

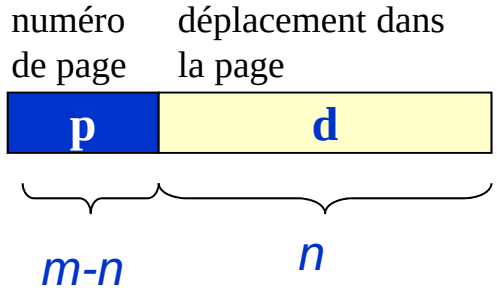
- **N.B :**
  - La taille des pages comme la taille du cadre de page est définie par le matériel.
  - En principe est une puissance de 2 (512 octets à 16 M Octets). **L'adresse logique est facilement traduite en adresse physique lorsque la taille des pages est une puissance de 2**

→ Soit : 2^m la taille de l'espace logique

→ La taille d'une page est 2^n unités d'adressage (octets ou mots),

→ Alors :

- ☞ les ***m-n*** bits les plus significatifs d'une adresse logique désignent le numéro de page (⇔ indice dans la table des pages),
- ☞ et les ***n*** bits les moins significatifs désignent les déplacements dans la page.
- ☞ En conséquence, l'adresse logique est la suivante:



N.B : Le choix d'une puissance de 2 comme taille de page facilite la traduction d'une adresse logique en un numéro de page et un déplacement dans la page

Recherche d'une adresse

- **Remarque :**

→Supposons qu'au cours de cette exécution, le processus fasse référence à une adresse logique **L.**

→Cette adresse logique est transformée en un couple ***(p, d),***

- **p** : est le numéro de page et
- **d :** le déplacement dans la page.

→Soit : **c** la taille d'une page,

♦alors :

→ p = **L** div **c** (quotient **)**

→ d = **L** mod c (le reste **)**

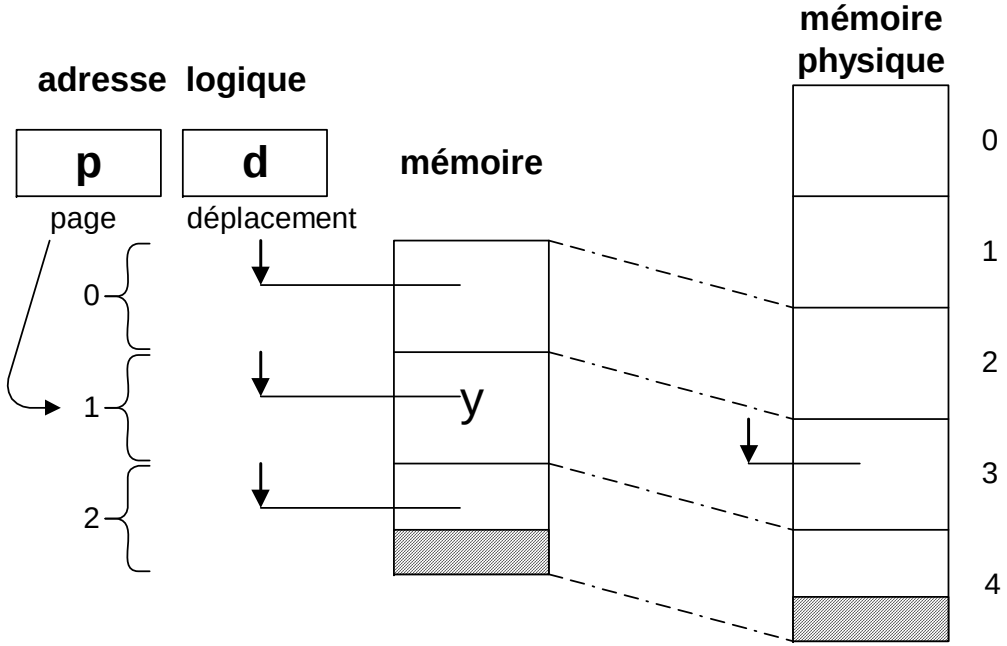
- L'adresse logique est donc:

→ **L** = p.c + d.

- Pour calculer l'adresse physique

→ **P (associé à L)** = y.c + d

Page p est dans le cadre y

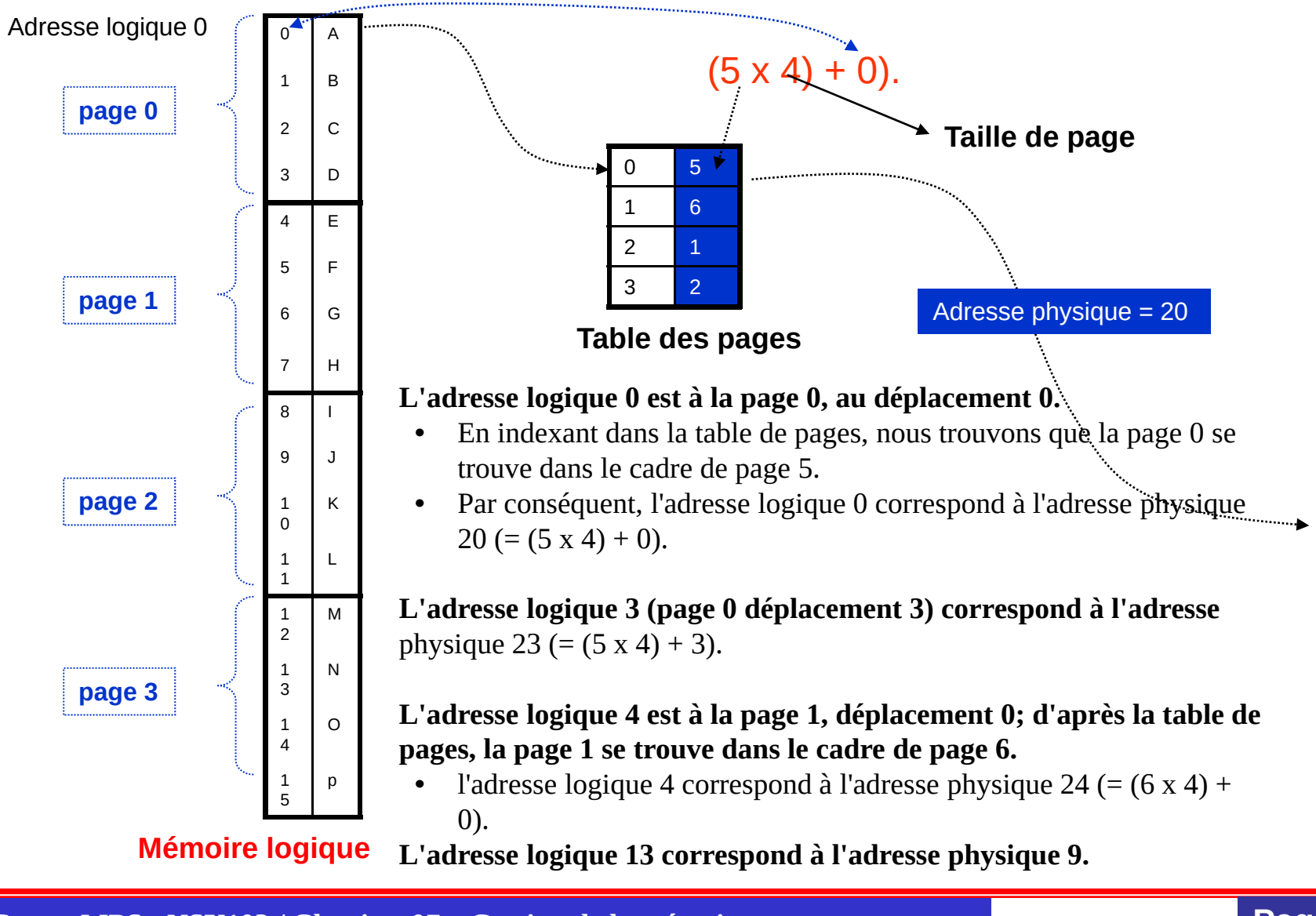


d : **déplacement = adresse modulo taille des pages**

p : **no de page = adresse / taille des pages**

Recherche d'une adresse

- **Exemple :**Taille de page est de 4 octets et une mémoire physique de 32 octets (8 pages)



L'adresse logique 0 est à la page 0, au déplacement 0.

- En indexant dans la table de pages, nous trouvons que la page 0 se trouve dans le cadre de page 5.
- Par conséquent, l'adresse logique 0 correspond à l'adresse physique 20 (= (5 x 4) + 0).

L'adresse logique 3 (page 0 déplacement 3) correspond à l'adresse physique 23 (= (5 x 4) + 3).

L'adresse logique 4 est à la page 1, déplacement 0; d'après la table de pages, la page 1 se trouve dans le cadre de page 6.

- l'adresse logique 4 correspond à l'adresse physique 24 (= (6 x 4) + 0).

L'adresse logique 13 correspond à l'adresse physique 9.

B. Remarques :

- On constate que la pagination est ne forme de translation dynamique.

- Pas de fragmentation externe: *chaque* cadre de page libre peut être alloué à un processus qui en a besoin.

- Fragmentation Interne : Possible. En effet, les cadres de page sont alloués sous forme d'unités. En fonction des besoins en mémoire d'un processus, il se peut que le dernier cadre de page alloué, ne soit pas complètement rempli, ce qui entraine une fragmentation interne pour le dernier cadre alloué

- Exemple :
  - les pages font 2048 octets, le processus fait 72 766 octets ce qui nécessite 35 pages plus 1086 Octets (⇔ 36 cadres attribués) et une fragmentation interne de 2048 – 1086 = 962 octets

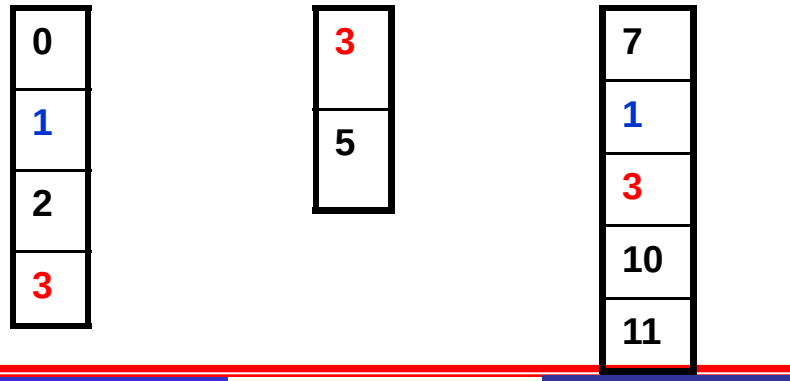
Partage et protection des pages

Partage et protection des pages

♦ **Exemple :**

- ♦ Supposons qu'un éditeur de texte comprenne 30K de code et 5K de données. Si cet éditeur sert à 40 utilisateurs et qu'il n'y a pas de partage, il faut utiliser 1400K (35*40 = 1400) de mémoire. En ne gardant qu'un seul exemplaire du code de l'éditeur et 40 copies de l'espace de données, il suffit de 230K.
- ♦ La solution qui consiste à partager certaines pages permet donc de réduire la quantité de mémoire nécessaire et, par conséquent, d'augmenter le nombre d'utilisateurs.
- ♦ Pour réaliser le partage d'une page chargée dans le cadre « y », il suffira que ce numéro apparaisse dans la table de plusieurs processus

- La page chargée dans le cadre 3 est partagée parles trois processus
- La page chargée dans le cadre 1 est partagée par les processus 1 et 3



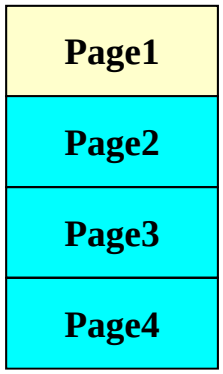
Partage et protection des pages

♦ *Des précautions s'imposent.*

- Par exemple, il faut empêcher qu'un processus ne modifie des données lues par un autre processus.
- On distingue alors deux catégories d'objets :
  - les procédures ou données *susceptibles d'être modifiées* et pour lesquelles *le partage n'est pas autorisé,*
  - les procédures ou données *non modifiables* qui pourront être partagées, *avec un accès en lecture seulement (de tels éléments sont dits réentrants).*
- Le caractère modifiable ou non doit être répercuté sur les pages pour permettre au système d'exploitation d'en assurer la protection.
  - Chaque entrée de la table des pages contient un ou plusieurs *bits de protection*,
  - Chaque bit étant associé à *un type d'accès*. On peut utiliser par exemple un bit correspondant à l'accès en lecture/ écriture ou en lecture seulement (ou bien aussi en exécution).
  - Les pages partageables sont en accès lecture seulement et une tentative d'écriture est décelée au niveau du matériel et provoque une interruption.

Partage et protection des pages –exemple

Espace d'adressage de P1 (16 KO)



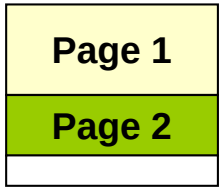
4 KO

Table des pages de P1

Page 1	**r--**	**Cadre 2**
Page 2	rw-	Cadre 6
Page 3	r-x	Cadre 3
Page 4	r-x	Cadre 7

Table des pages de P2

Page 1	**r--**	**Cadre 2**
Page 2	rw-	Cadre 8

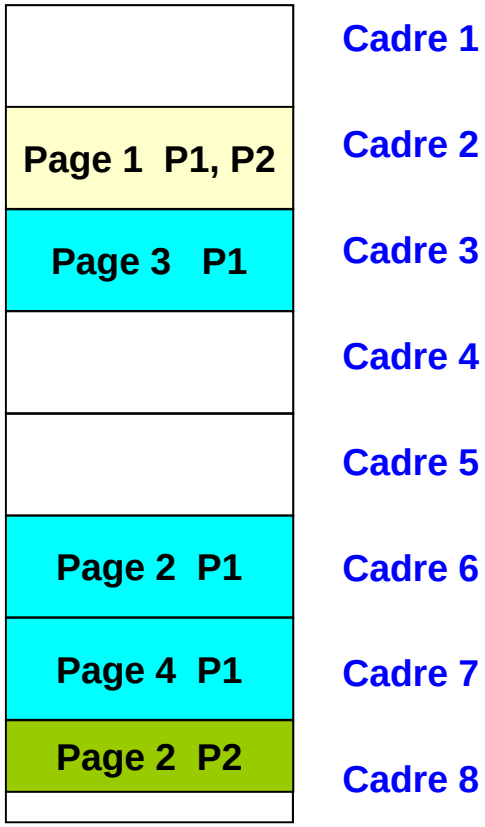


Espace d'adressage de P1 (7 KO)

Table des cadres

	Page	Processus
Cadre 1		
Cadre 2	**1**	**P1, P2**
Cadre 3	3	P1
Cadre 4		
Cadre 5		
Cadre 6	2	P1
Cadre 7	4	P1
Cadre 8	2	P2

Mémoire Physique

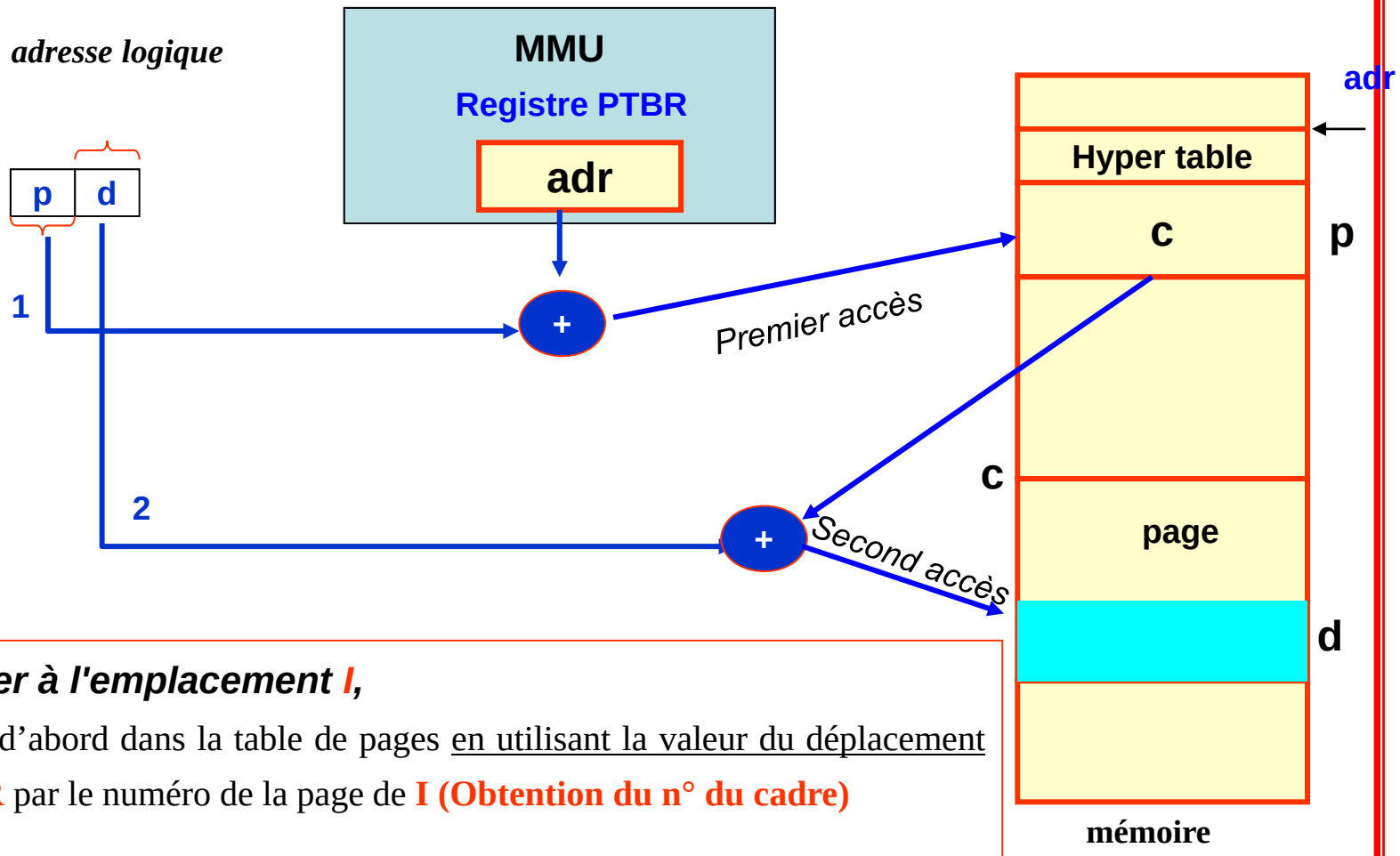


Gestion des tables des pages

Gestion des tables des pages

- **Introduction :** L'implémentation matérielle de la table de pages peut être effectuée de différentes façons.
  - Cas simple : implémentation sous forme de jeu de registres dédié.
  - Cas plus complexe : implémentation avec des registres associatifs ou (TLB, *Translation Look-aside Buffer)*

1. Implémentation avec le registre PTBR (*Page-Table Base Register*)



- ***Pour accéder à l'emplacement I,***
  1. Indexer, d'abord dans la table de pages en utilisant la valeur du déplacement du PTBR par le numéro de la page de **I (Obtention du n° du cadre)**
  2. Le N° du cadre est combiné au déplacement de la page pour donner l'adresse réelle. Nous pouvons alors accéder à l'emplacement désiré en mémoire.

1. Implémentation avec le registre PTBR (*Page-Table Base Register*)

- La table de pages est *conservée dans la mémoire principale*,
- un registre de base de tables de pages (**PTBR**, *Page-Table Base Register)* pointe sur la table de pages (en plus d'un autre registre ⇔ taille de la table).
- Une modification des tables de pages ne requiert qu'une modification de ce registre, ce qui réduit considérablement le temps de commutation de contexte
-
- Problème :
  - Deux accès mémoire sont nécessaires pour accéder à l'adresse physique

3. Implémentation avec TLB (Translation Look-Aside Buffer)

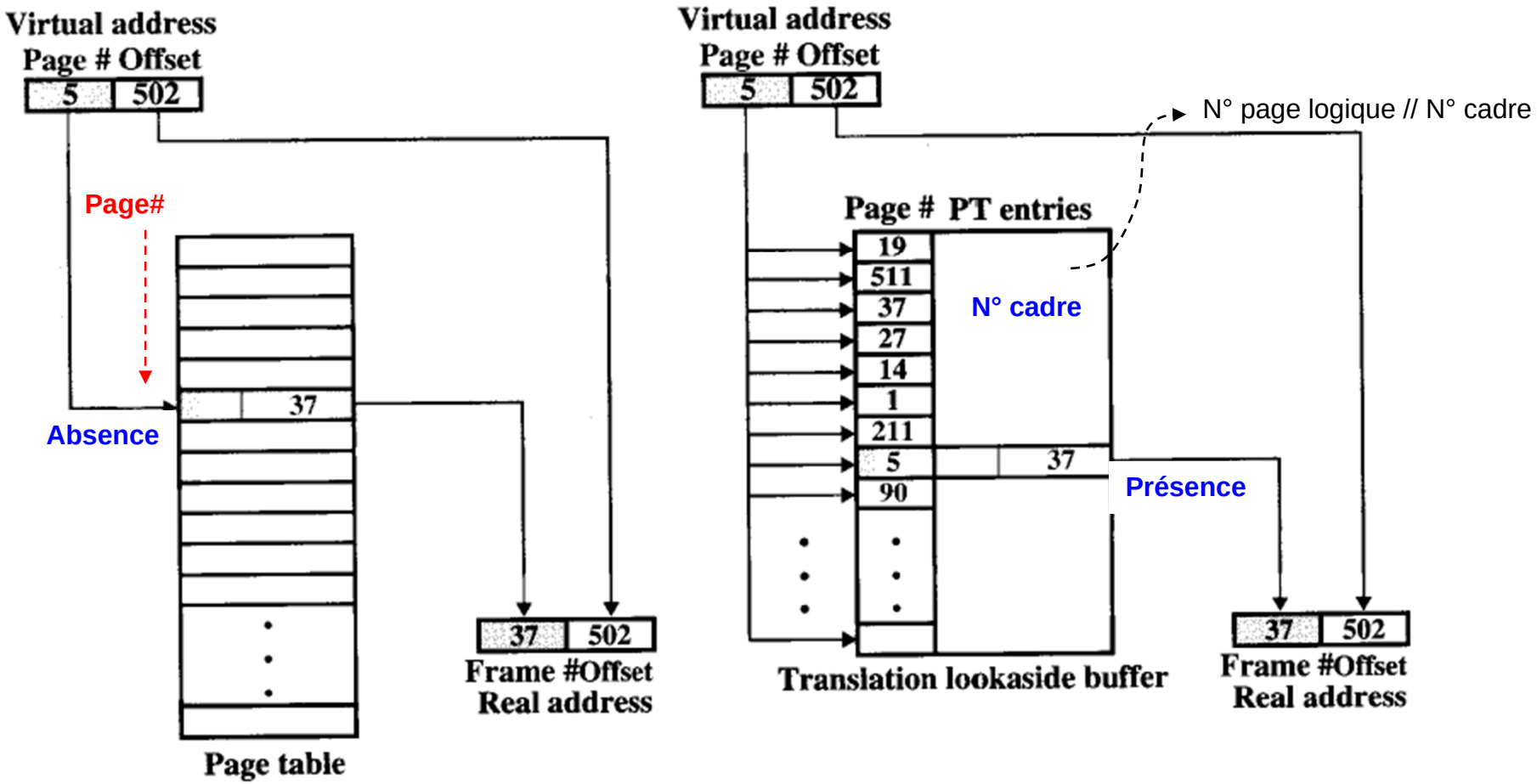
- La conversion peut être accélérée en plaçant la table dans une mémoire cache.
- Lorsque cette dernière est constituée de registres associatifs, toutes les entrées sont examinées simultanément lors d'une référence, ce qui rend le premier accès beaucoup plus rapide.

- Recherche parallèle d ’une adresse:
  - l ’adresse recherchée est cherchée dans la partie gauche de la table en parallèle (matériel spécial)

- Traduction page → cadre
  - Si la page recherchée a été utilisée récemment elle se trouvera dans les registres associatifs
    - recherche rapide

No Page	No Cadre
3	15
7	19
0	17
2	23

- Implémentation avec TLB (Translation Look-aside Buffer)



Source : stallings)

Recherche *associative* dans TLB

- Le TLB est un petit tableau de registres de matériel où chaque ligne contient une paire:
  - Numéro de page logique // Numéro de cadre
- Le TLB utilise du matériel de *mémoire associative*:
  - ⇨ interrogation simultanée de tous les numéros logiques pour trouver le numéro physique recherché
- Chaque paire dans le TLB est fournie d'un indice de référence pour savoir si cette paire a été utilisée récemment.
  - Sinon, elle est remplacée par la dernière *paire dont on a besoin*

principe de localité

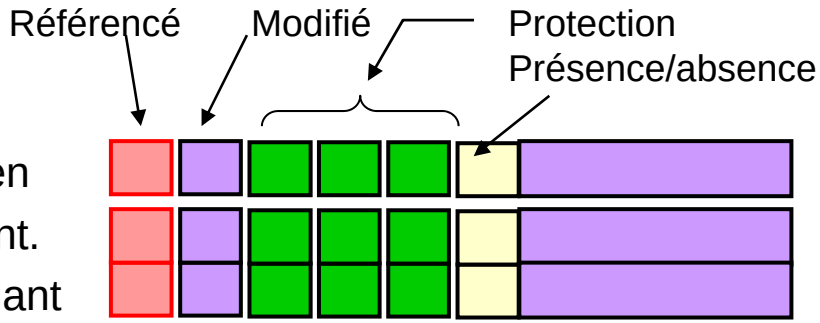
- **Principe de localité :** les adresses référencées par un processus ne sont pas réparties uniformément mais tendent à se regrouper à la fois *dans le temps* et *dans l'espace.*

1. **La localité dans le temps** :
   - si un ensemble d'adresses a été référencé dans un passé immédiat, ces mêmes adresses seront référencées dans un futur immédiat, avec une forte probabilité.
   - **Exemples** :
     - *Cas **d'une boucle** dans un programme* est un facteur qui favorise ce type de localité: la suite des références situées à l'intérieur de la boucle **est reproduite plusieurs fois.**
     - *Cas d'un sous-programmes* : pendant toute la durée d'exécution d'une procédure, l'ensemble des adresses référencées est pratiquement invariant.
2. **La localité dans l'espace** signifie que, sur un intervalle de temps moyen, les références portent en majorité sur **des adresses voisines**.
   - **Exemple :**
     - le parcours d'un tableau illustre bien ce type de localité: pendant la durée du parcours, des adresses consécutives sont référencées.

Conséquences de ce principe

- Toutes les pages d'un processus n'ont pas besoin d'être (chargées) en mémoire lors de l'exécution
- Nécessite d'ajouter un bit d'invalidité à chaque entrée de la table des pages (1 = page en mémoire; 0 = page non chargée)
- Traduction d'adresse :
  - → Bit (valide) = 1 : ⇔ l'entrée est valide et peut être utilisée
  - → Bit (invalide) = 0 : la page à laquelle appartient cette entrée n'est pas en mémoire (⇔ exception levée (faute de page), traitée par le système d'exploitation)
- Permet d'utiliser moins de mémoire pour un processus, et donc d'avoir plus de processus en mémoire
- Permet d'exécuter des processus dont l'espace virtuel est plus grand que la mémoire disponible

Structure d'une entrée typique d'une Table de pages



Présence/absence (valide) :

- Si le bit de présence est à 0, la page n'est pas en mémoire alors le MMU provoque un déroutement.
- Sinon, il détermine l'adresse physique en recopiant dans les numéro de case correspondant au numéro de page

Protection

Indique le type d'accès autorisé (lecture/Ecriture/Exécution)

Les bits Modifié / Référence mémorisent l'utilisation de la page

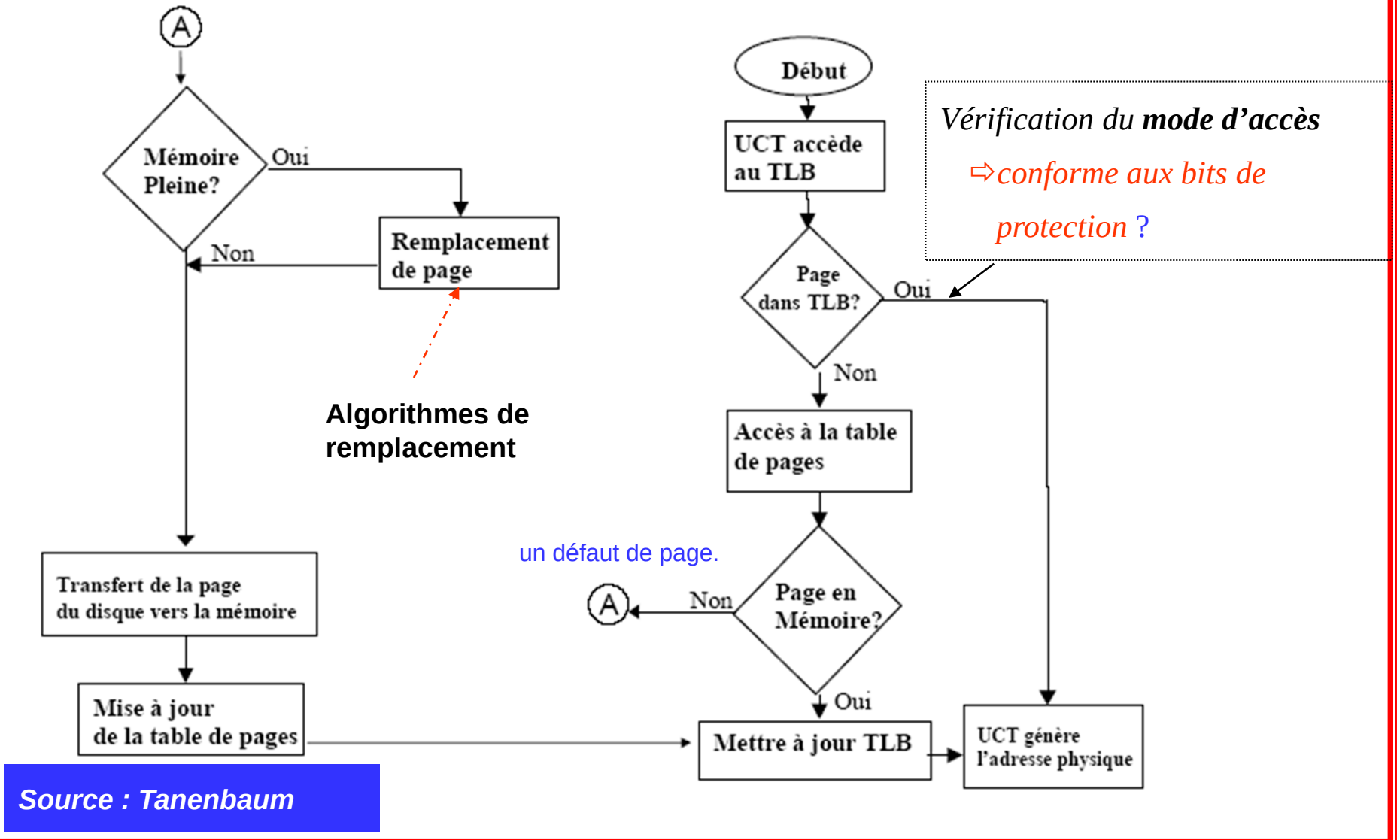
Bit de modification (M)

Le bit Modifié est positionnée par le matériel lorsque la page est écrite.
Important lorsque le SE décide de retirer une page de la mémoire centrale
Si Modifié = 1, alors la page doit être re-écrite sur le disque
Si Modifié = 0, la page peut être simplement retiré de la mémoire

Référencé (R) :

est utilisé chaque fois une page est accédé en lecture ou en écriture
Permet de choisir la page à retirer en cas de défaut de page

Conversion d'adresses virtuelles MMU avec mémoire associative



Source : Tanenbaum

Conversion d'adresses virtuelles MMU avec mémoire associative

1. Lorsqu'une adresse virtuelle est présentée au MMU, il contrôle d'abord si le numéro de la page virtuelle est présent dans la mémoire associative (**en le comparant simultanément à toutes les entrées**).

   ⇨ S'il le trouve et ***le mode d'accès est conforme aux bits de protection***, le numéro de case est pris directement de la mémoire associative (sans passer par la table des pages).

   ⇨ Si le numéro de page est présent dans la mémoire associative **mais le mode d'accès est non conforme**, il se produit un ***défaut de protection*****.**

2. Si le numéro de page n'est pas dans la mémoire associative, le MMU accède à la table des pages à l'entrée correspondant au numéro de page.

3. *Si le bit de présence de l'entrée trouvée est à 1*, le MMU remplace une des entrées de la mémoire associative par l'entrée trouvée.

4. *Si le bit de présence de l'entrée trouvée est à 0*, il provoque un défaut de page.

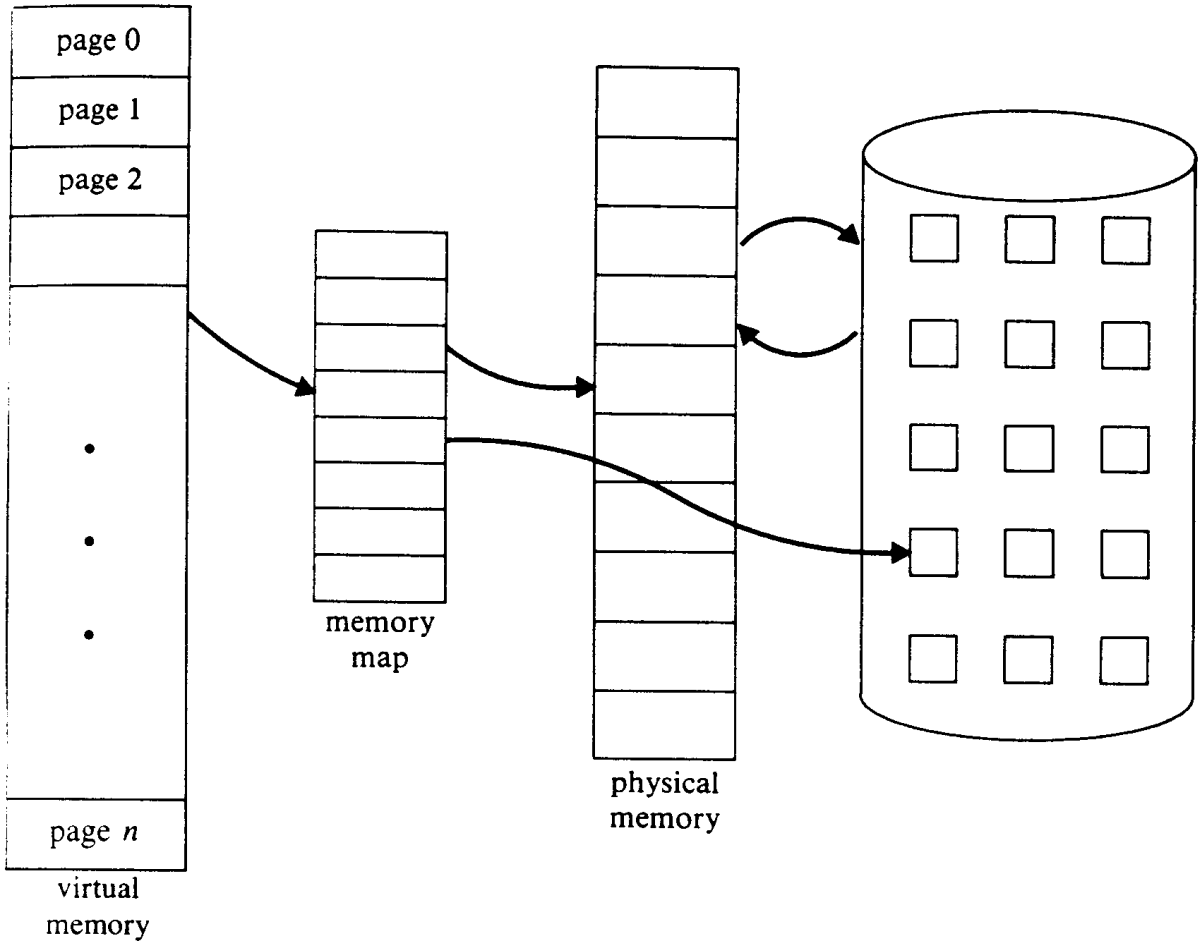
☞ VERS LA MÉMOIRE VIRTUELLE

De la pagination à la Mémoire Virtuelle

1. Références à la mémoire sont traduites en adresses physiques au moment d'exécution
   - Un processus peut être déplacé à différentes régions de la mémoire
2. Un processus est constitué de pièces (pages ou segments) ne nécessitant pas d'occuper une région contiguë de la mémoire principale
3. Donc: toutes les pièces 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 une pièce se trouvant en mémoire principale
4. La somme des mémoires logiques des procs en exécution peut donc excéder la mémoire physique disponible
   - ⇔ Le concept de base de la mémoire virtuelle
5. 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

Diagramme montrant une mémoire virtuelle plus grande que la mémoire physique

- séparation entre l'espace logique d'un programme et la mémoire physique
- l'espace logique peut être plus grand que la mémoire RAM
- seulement une partie du programme est en mémoire



Localité et mémoire virtuelle

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

Mémoire virtuelle - conclusion

- Le principe de la localité suggère que le concept de mémoire virtuelle fonctionne.
- Pour que ceci fonctionne dans la pratique, deux schémas doivent être utilisés :
  1. Le support matériel du concept de pagination (segmentation)
  2. L'OS doit inclure des algorithmes de gestion de mouvement des pages (des segments) entre la mémoire principale et secondaire

DEMANDE DE PAGES « Pages en RAM ou sur disque »

- Pagination à la demande :

1. Au départ, les processus résident en mémoire auxiliaire (généralement un disque) (⇔ bit de présence dans les pages = 0)
2. Un processus est vu comme une séquence de pages (la mémoire aussi)
3. Lorsque on veut exécuter un processus, on le transfert en mémoire.
   - Pas de transfert du ***processus entier***
   - Transfert des pages leur de ***leurs utilisation*** (⇔ on ne transfert jamais une page en mémoire tant que celle-ci n'est pas indispensable.

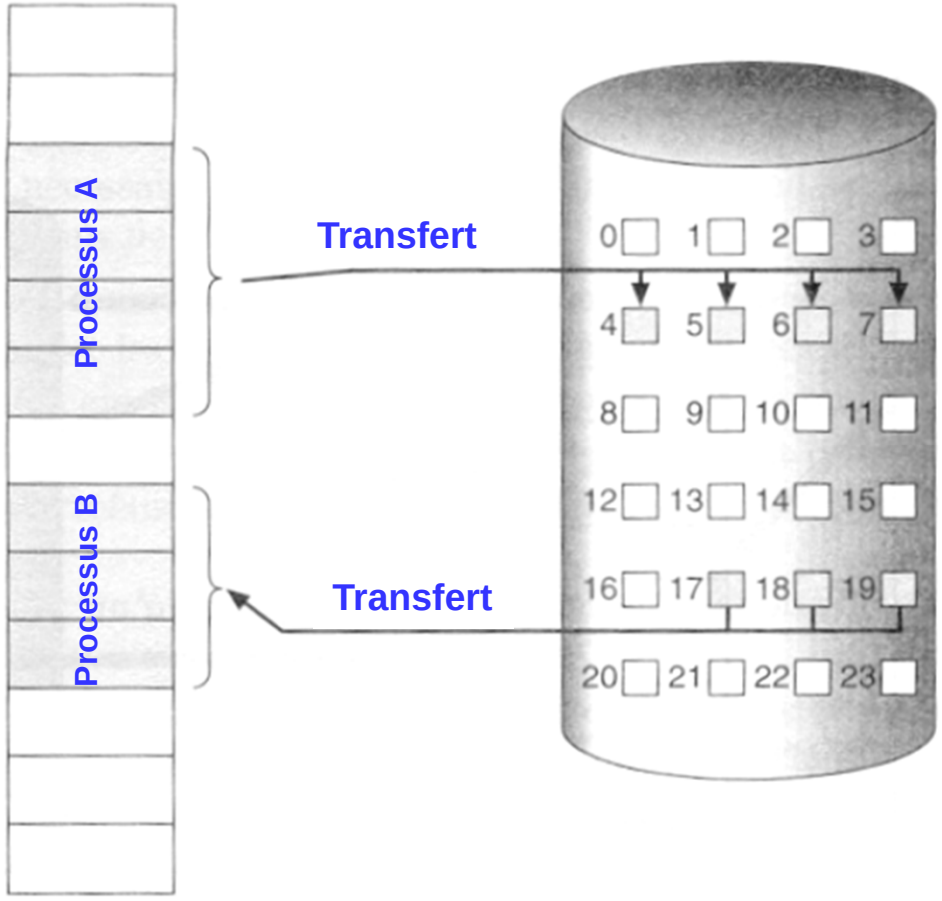


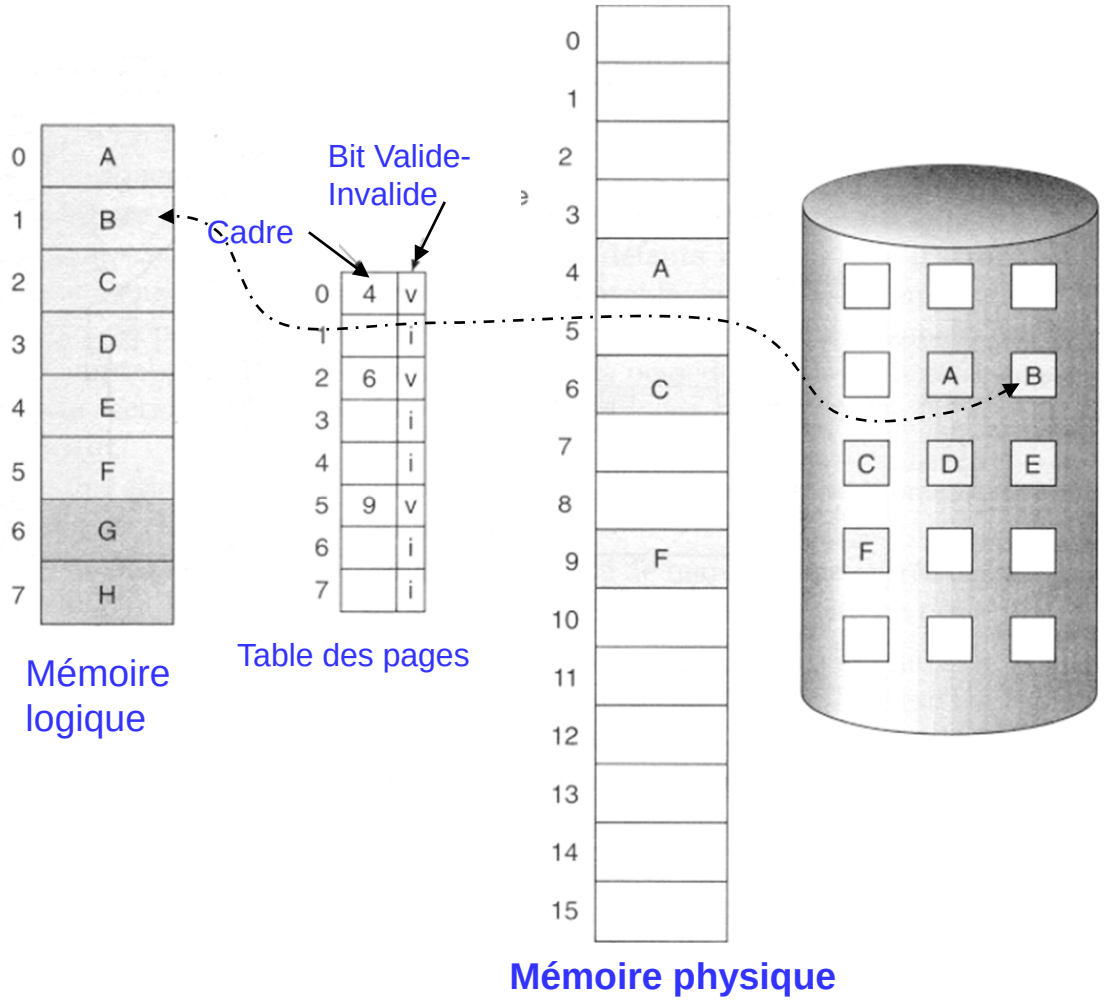
Table des pages lorsque quelques pages ne sont pas dans la mémoire principale

DEMANDE DE PAGES « Pages en RAM ou sur disque »

- support matériel pour faire le distinguo entre les pages en mémoire et les pages sur disque ⇔ bit *valide-invalide*

- **Bit = « valide »** ⇔ la page concernée est à la fois légale et en mémoire.

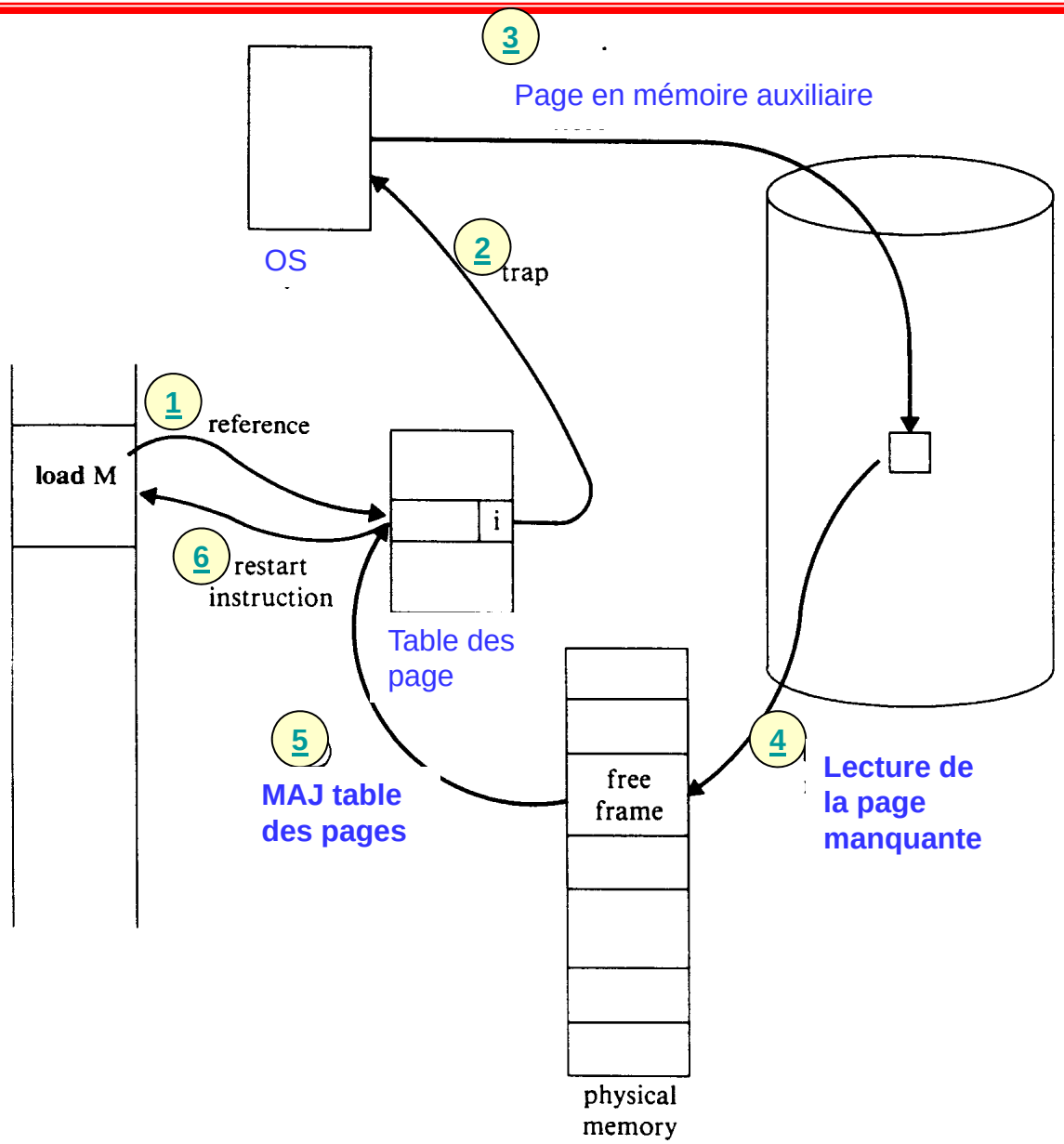
- **Bit = « invalide »** ⇔ il indique que la page est
  - ⇨ soit invalide (elle ne se trouve pas dans l'espace d'adresses logique du processus),
  - ⇨ soit valide mais se trouvant pour l'heure sur le disque.



Exécution d'une défaut de page

DEMANDE DE PAGES

Exécution d'une défaut de page



Remplacement de pages

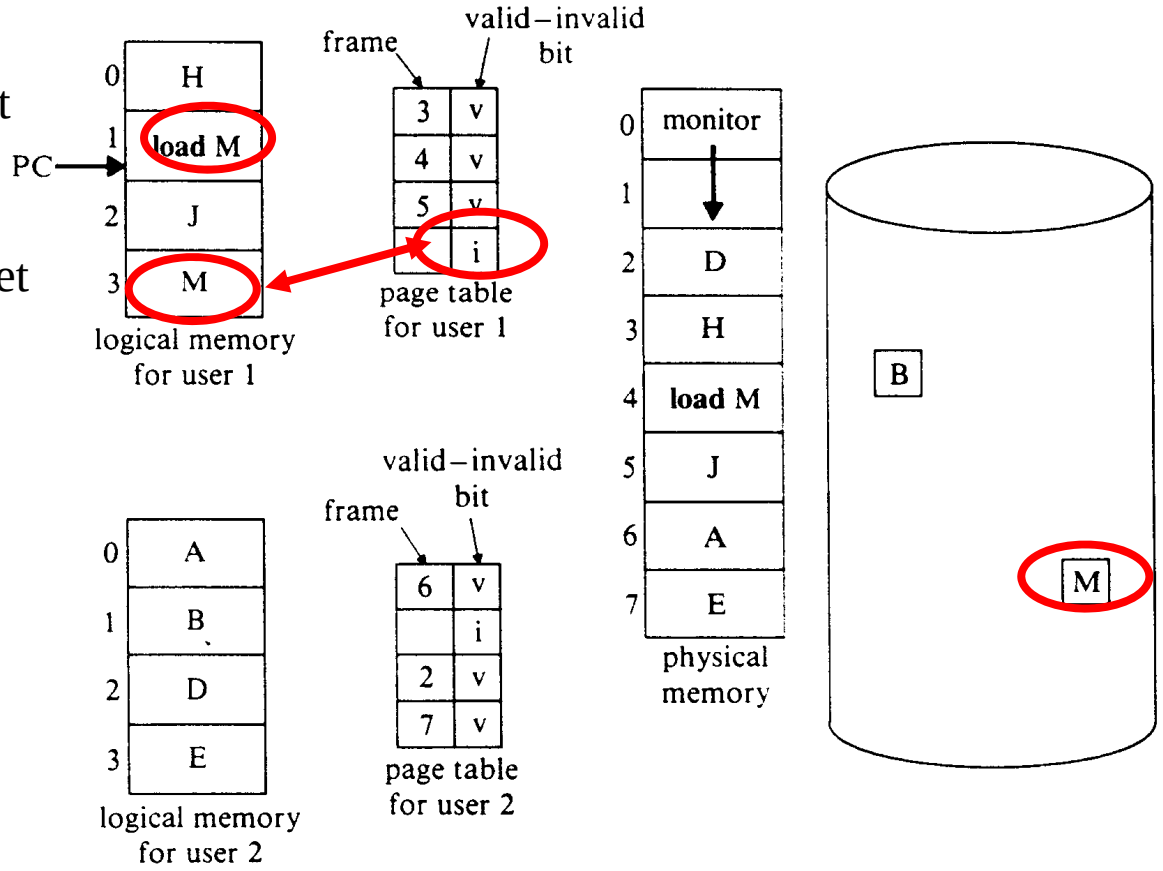
Quand le RAM est plein mais nous avons besoin d`une page qui n'est pas en RAM

- Toutes les pages physiques sont prises :

  →choisir une page peu utilisée et

  →la sauver dans le stockage de réserve

⇨ bit de modification de page (dirty bit)



Remplacement des pages

1. Trouver l'emplacement sur disque de la page souhaitée.

2. Trouver un cadre de page libre
   a) S'il en existe un, l'utiliser ;

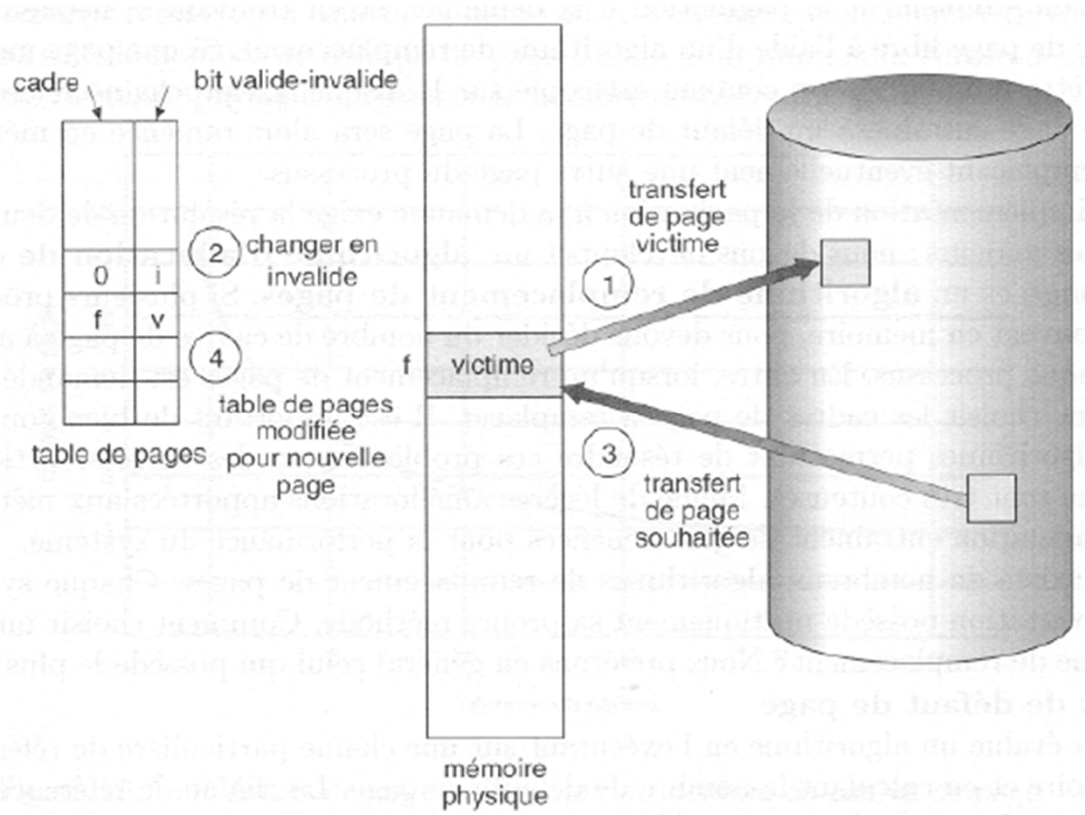
   b) S'il n'en existe pas, utiliser un algorithme de remplacement de pages pour sélectionner un cadre de page victime ;

   c) Ecrire la page victime sur le disque; modifier les tables de pages et de cadres de page en conséquence.

3. Lire la page souhaitée dans le cadre de page (nouvellement) libéré; modifier les tables de pages et de cadres de page.

4. Relancer le processus utilisateur .

La page victime...



REMPLACEMENT DE PAGE

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

REMPLACEMENT DE PAGE : Bit de modification , *dirty bit*

- **Remarque :** si aucun cadre de page n'est libre, *deux* transferts de pages (un depuis la mémoire, l'autre vers la mémoire) sont nécessaires. Cette situation double le temps de traitement du défaut de page et augmente le temps d'accès effectif en conséquence.

- Solution : Utilisation du bit de modification **« Dirty Bit »**
  - La 'victime' doit-elle être récrite 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 modif**. sur 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 récriture dans ce* cas
    - Dirty Bit = 1, alors page a été modifié depuis sa lecture du disque => il faut réécrire la page sur le disque
    - Dirty Bit = 0 ; page n'a pas été modifié depuis sa lecture => pas de réécriture

ALGORITHMES DE REMPLACEMENT DE PAGE

ALGORITHMES DE REMPLACEMENT DE PAGE

- 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. ...

Algorithmes pour la politique de remplacement

- Un algorithme de remplacement de page doit minimiser le nombre de fautes de pages :
  - Algorithme qui minimise au mieux la probabilité d'occurrence d'une erreur de page
  - Un algorithme est évalué en prenant une chaîne de pages et en comptant le nombre de fautes de page qui ont lieu durant cette suite d'accès et cela en fonction du nombre de pages de mémoire centrale dont il dispose
  - Pour illustrer les algorithmes de remplacement, on utilisera dans la suite, la suite des pages suivantes :
    - 7, 0, 1, 2, 0, 3, 0, 4, 2, 3, 0, 3, 2, 1, 2, 0, 1, 7, 0, 1
    - et 3 pages (cadres) en mémoires centrales

Remplacement de pages FIFO : Premier Arrivé – Premeir Sortie

- Principe premier entré, premier sorti (FIFO, *First in, First out).*
  - Il associe à chaque page le moment où celle-ci a été portée en mémoire.
  - Au moment de remplacer une page, *la plus ancienne* est choisie.
  - On peut créer une file *FIFO* contenant toutes les pages en mémoire.
    - On remplace la page en tête de file.
    - Lorsqu'une page est chargée en mémoire, on l'insère en fin de file.

Remplacement des pages FIFO -1-

références de pages logiques

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

pages physiques --> pages logiques

7

file : # page logique

7

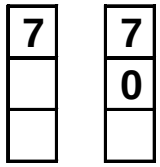
FIFO : First-In-First-Out
: Enlever la plus vielle page

Remplacement des pages FIFO -2-

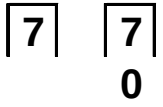
références de pages logiques

7 0
F F

pages physiques --> pages logiques



file : # page logique



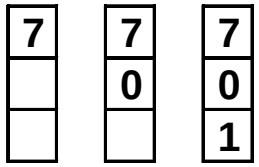
FIFO : First-In-First-Out
: Enlever la plus vielle page

Remplacement des pages FIFO -3-

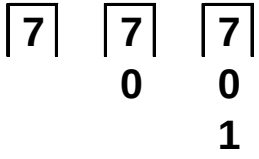
références de pages logiques

7	0	1
F	F	F

pages physiques --> pages logiques



file : # page logique



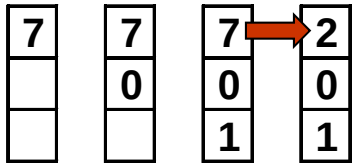
FIFO : First-In-First-Out
: Enlever la plus vielle page

Remplacement des pages FIFO -4-

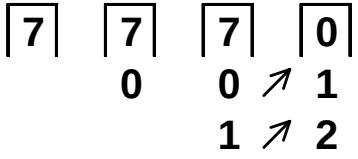
références de pages logiques

7	0	1	2
F	F	F	F

pages physiques --> pages logiques



file : # page logique



FIFO : First-In-First-Out
: Enlever la plus vielle page

Remplacement des pages FIFO -5-

références de pages logiques

7	0	1	2	0
F	F	F	F	

pages physiques --> pages logiques

7	7	7	2
	0	0	0
		1	1

file : # page logique

7	7	7	0
	0	0 ↗	1
		1 ↗	2

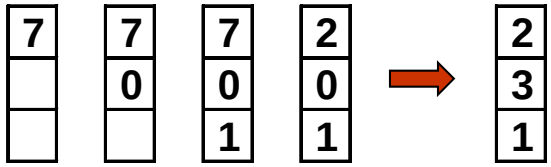
FIFO : First-In-First-Out
: Enlever la plus vielle page

Remplacement des pages FIFO -6-

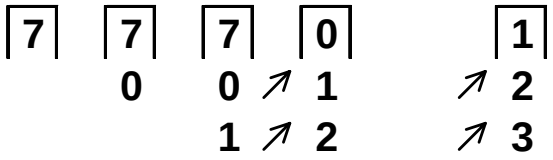
références de pages logiques

7	0	1	2	0	3
F	F	F	F		F

pages physiques --> pages logiques



file : # page logique



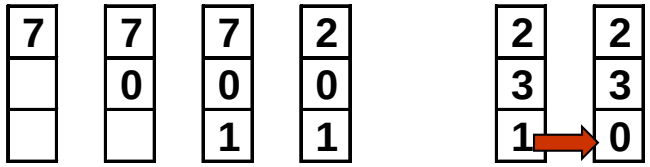
FIFO : First-In-First-Out
: Enlever la plus vielle page

Remplacement des pages FIFO -7-

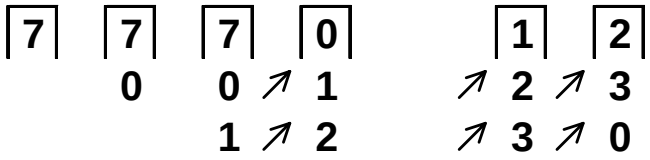
références de pages logiques

7	0	1	2	0	3	0
F	F	F	F		F	F

pages physiques --> pages logiques



file : # page logique



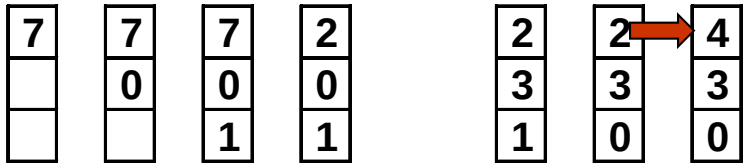
FIFO : First-In-First-Out
: Enlever la plus vielle page

Remplacement des pages FIFO -8-

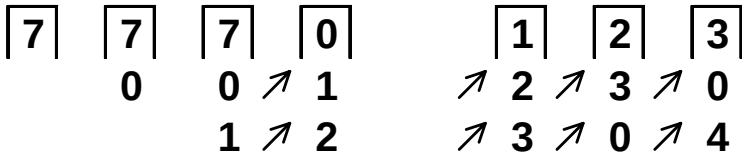
références de pages logiques

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

pages physiques --> pages logiques



file : # page logique



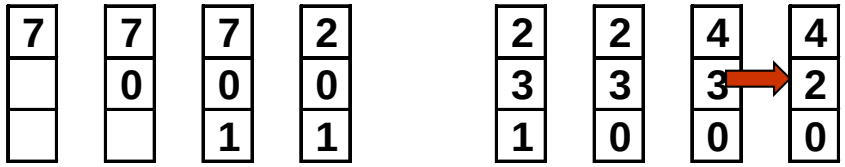
FIFO : First-In-First-Out
: Enlever la plus vielle page

Remplacement des pages FIFO -9-

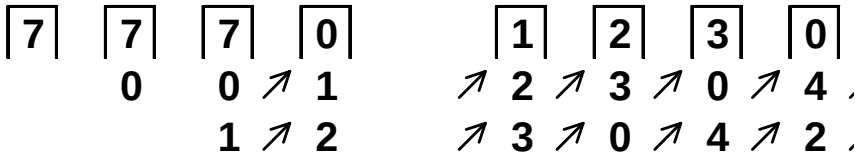
références de pages logiques

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

pages physiques --> pages logiques



file : # page logique



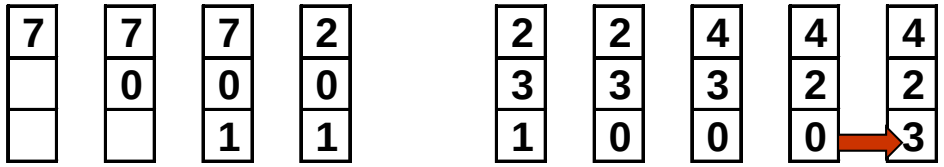
FIFO : First-In-First-Out
: Enlever la plus vielle page

Remplacement des pages FIFO -10-

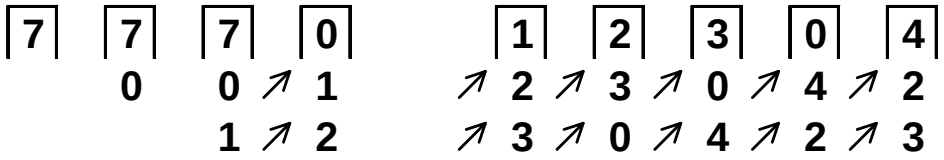
références de pages logiques

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

pages physiques --> pages logiques



file : # page logique



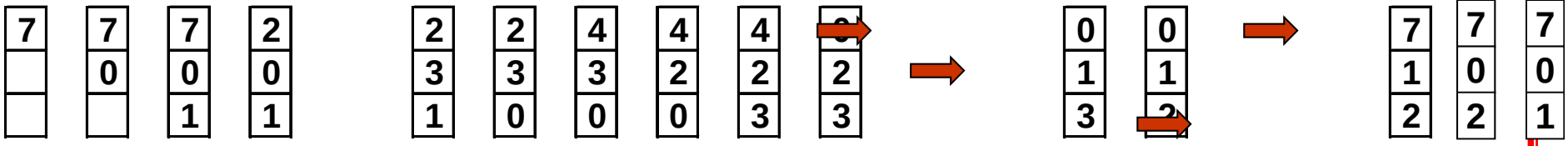
FIFO : First-In-First-Out
: Enlever la plus vielle page

Remplacement des pages FIFO -11-

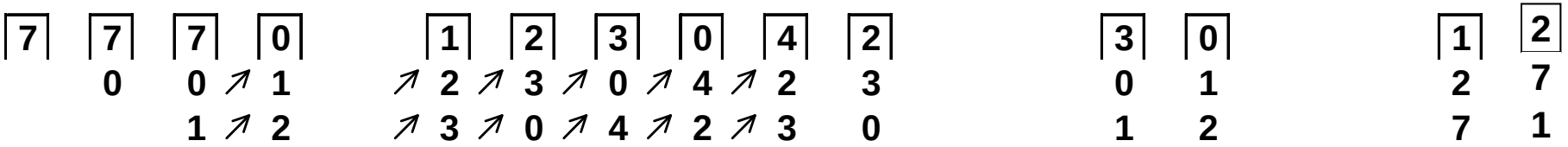
références de pages logiques

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

pages physiques --> pages logiques



file : # page logique



15 FAUTES DE PAGES

Remplacement des pages FIFO -12-

- L'algorithme de remplacement de pages FIFO est simple à comprendre et à programmer, mais sa performance *n'est pas toujours excellente*.
  - La page remplacée peut être un *module d'initialisation* utilisé depuis un long moment et dont on n'a plus besoin.
  - Elle peut également contenir *une variable* souvent utilisée, initialisée depuis longtemps et *constamment sollicitée*.

- Annomalie de Belady
  - Il existe des suites de pages pour lesquelles, l'algorithme fait plus de fautes de pages avec quatre cadre que avec trois
  - Exemple : avec la chaîne de référence suivante : 1, 2, 3, 4, 1, 2, 5, 1, 2, 3, 4, 5

Remplacement des pages LRU –Principe-

- L'algorithme optimal n'est pas réalisable, une approximation est peut-être possible.
- Rappel : Différence essentielle entre les algorithmes FIFO et OPT l'algorithme FIFO
  - utilise le temps pendant lequel une page a été amenée en mémoire
  - l'algorithme OPT utilise le temps pendant lequel une page doit être *employée*.
- Algorithme LRU (*Least Recently Used)*
  - Utilise le passé récent comme approximation du futur proche pour remplacer la page qui *n'aura pas été employée* pendant le plus long moment.
  - Il utilise le principe de la localité temporelle selon lequel les pages récemment utilisées sont celles qui seront utilisées dans le future

Remplacement des pages LRU –Principe-

- Implémentation de cet algorithme :
  - Le remplacement LRU associe à chaque page le temps de sa dernière utilisation. Lorsqu'une page doit être remplacée, LRU choisit celle qui n'a pas été employée pendant le plus longtemps
  - La politique LRU, souvent utilisée comme algorithme de remplacement de pages, est *considérée comme bonne*.
  - Le principal problème concerne la *façon* de l'implémenter.
    - Besoin d'une assistance matérielle assez conséquente pour la délivrance des dates d'accès aux pages

Remplacement des pages LRU –Principe-

- Comment Implémenter cet algorithme : mémorisation pour chaque page en mémoire de la date de la dernière référence
  1. Vieillissement :
     - Un registre de n bits est associé à chaque page
     - Le bit le plus significatif est mis à 1 à chaque référence
     - Régulièrement, on décale vers la droite les bits de ce registre
     - On choisit la page dont la valeur est la plus petite

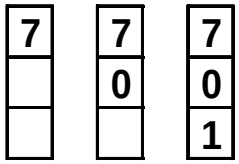
  2. avec une Pile des numéros de pages.
     - Quand une page est utilisée, est mise au sommet de la pile.
       - donc la page la plus récemment utilisée est toujours au sommet,
       - la moins récemment utilisée est toujours au fond

Exemple : Remplacement des pages LRU -1-

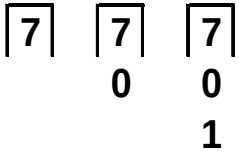
références de pages logiques

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

pages physiques --> pages logiques



file : # page logique

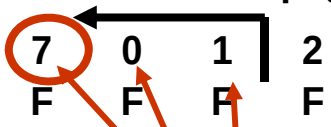


page la plus récente

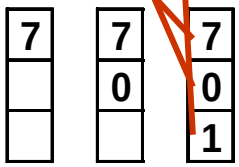
LRU
- Least Recently Used
- Enlever la page la moins récemment utilisée
- On extrapole le passé pour prévoir l'avenir

Remplacement des pages LRU-2-

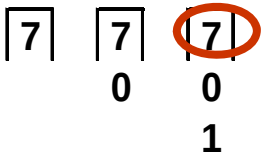
références de pages logiques



pages physiques --> pages logiques



file : # page logique



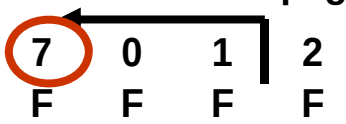
page la plus récente

LRU

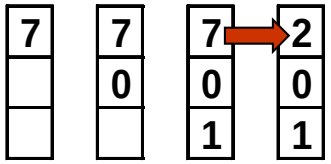
- Least Recently Used
- Enlever la page la moins récemment utilisée
- On extrapole le passé pour prévoir l'avenir

Remplacement des pages LRU-3-

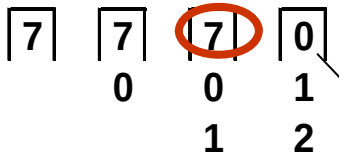
références de pages logiques



pages physiques --> pages logiques



file : # page logique

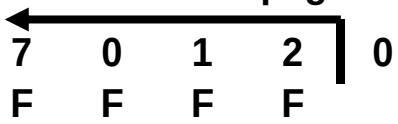


page la plus récente

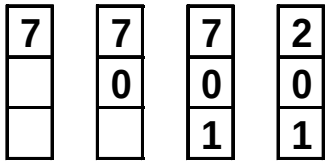
LRU
- Least Recently Used
- Enlever la page la moins récemment utilisée
- On extrapole le passé pour prévoir l'avenir

Remplacement des pages LRU-4-

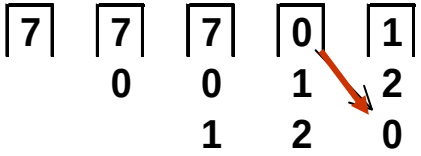
références de pages logiques



pages physiques --> pages logiques



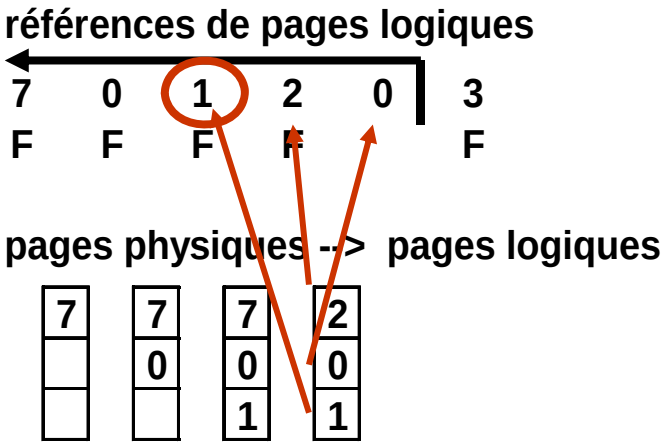
file : # page logique



page la plus récente

LRU
- Least Recently Used
- Enlever la page la moins récemment utilisée
- On extrapole le passé pour prévoir l'avenir

Remplacement des pages LRU-5-

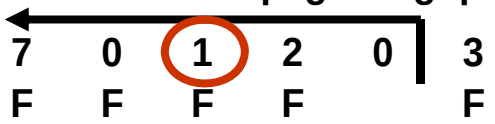


LRU

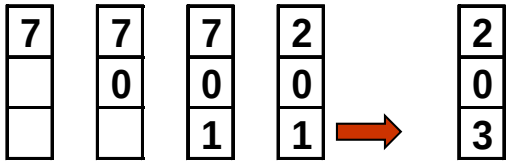
- Least Recently Used
- Enlever la page la moins récemment utilisée
- On extrapole le passé pour prévoir l'avenir

Remplacement des pages LRU-6-

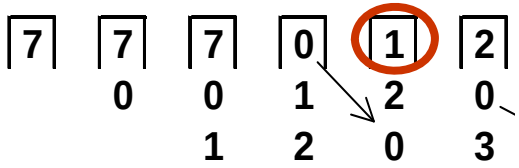
références de pages logiques



pages physiques --> pages logiques



file : # page logique

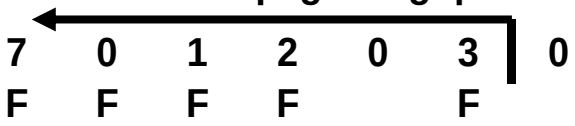


page la plus récente

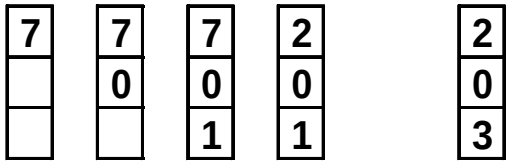
LRU
- Least Recently Used
- Enlever la page la moins récemment utilisée
- On extrapole le passé pour prévoir l'avenir

Remplacement des pages LRU-7-

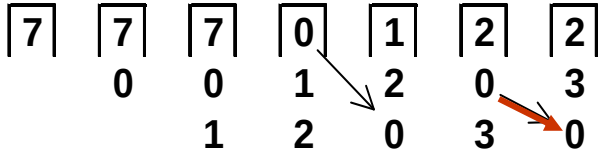
références de pages logiques



pages physiques --> pages logiques



file : # page logique

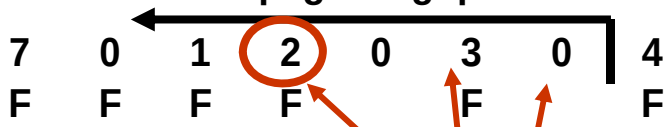


page la plus récente

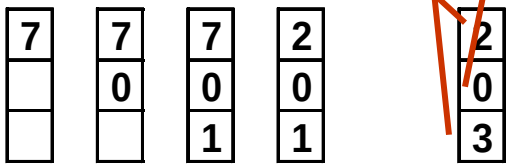
LRU
- Least Recently Used
- Enlever la page la moins récemment utilisée
- On extrapole le passé pour prévoir l'avenir

Remplacement des pages LRU-8-

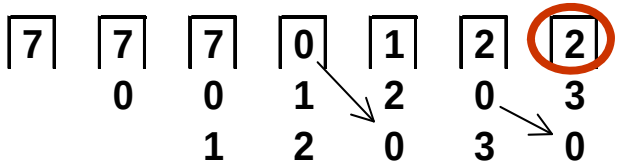
références de pages logiques



pages physiques --> pages logiques



file : # page logique

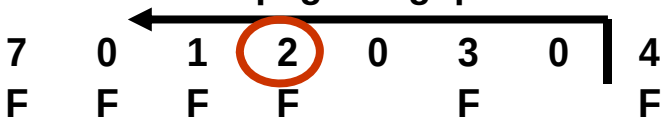


page la plus récente

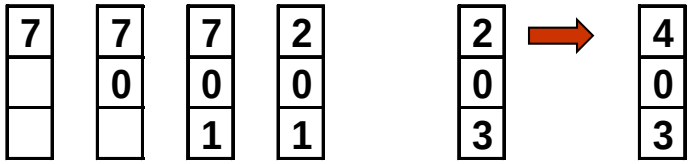
LRU
- Least Recently Used
- Enlever la page la moins récemment utilisée
- On extrapole le passé pour prévoir l'avenir

Remplacement des pages LRU-9-

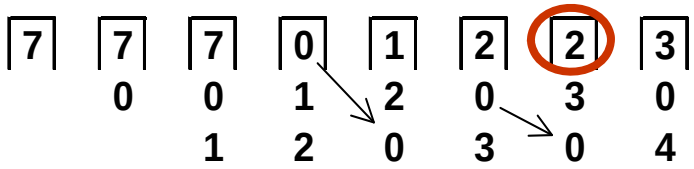
références de pages logiques



pages physiques --> pages logiques



file : # page logique

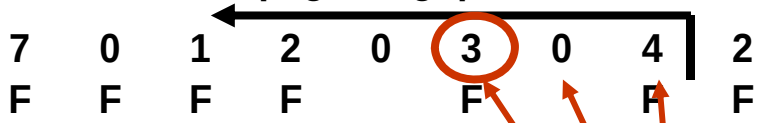


page la plus récente

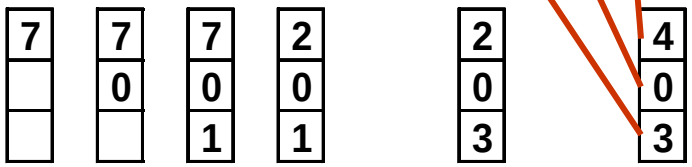
LRU
- Least Recently Used
- Enlever la page la moins récemment utilisée
- On extrapole le passé pour prévoir l'avenir

Remplacement des pages LRU-10-

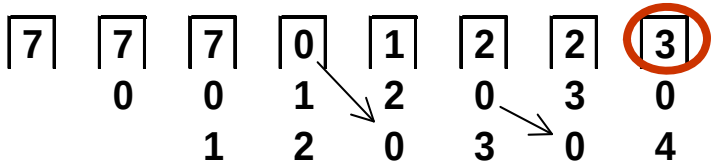
références de pages logiques



pages physiques --> pages logiques



file : # page logique



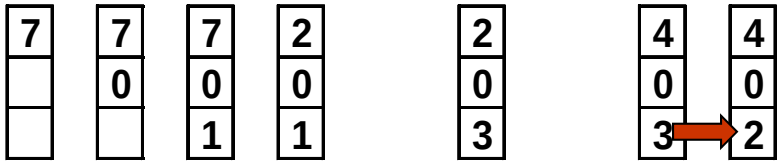
page la plus récente

LRU
- Least Recently Used
- Enlever la page la moins récemment utilisée
- On extrapole le passé pour prévoir l'avenir

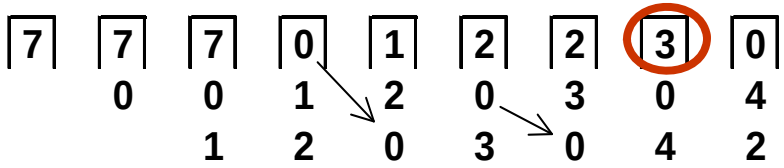
Remplacement des pages LRU-11-

références de pages logiques

pages physiques --> pages logiques



file : # page logique

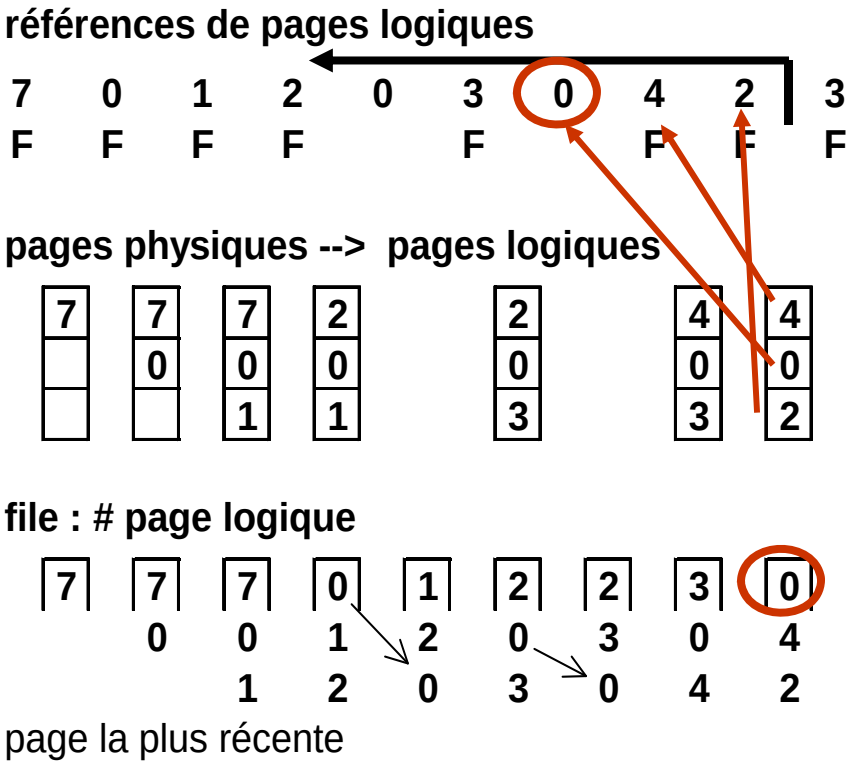


page la plus récente

LRU

- Least Recently Used
- Enlever la page la moins récemment utilisée
- On extrapole le passé pour prévoir l'avenir

Remplacement des pages LRU-12-

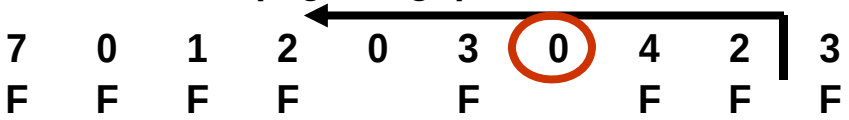


LRU

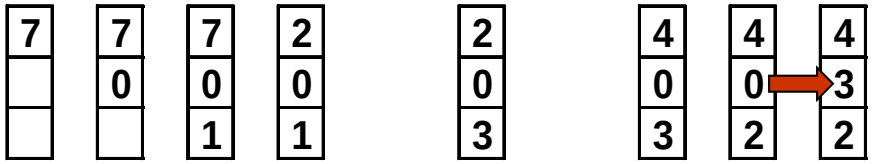
- Least Recently Used
- Enlever la page la moins récemment utilisée
- On extrapole le passé pour prévoir l'avenir

Remplacement des pages LRU-13-

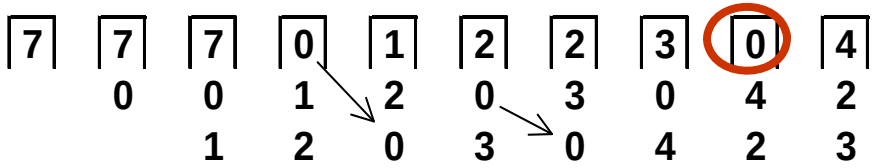
références de pages logiques



pages physiques --> pages logiques



file : # page logique

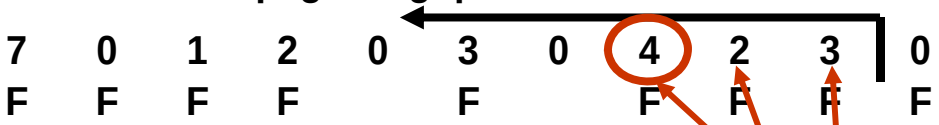


page la plus récente

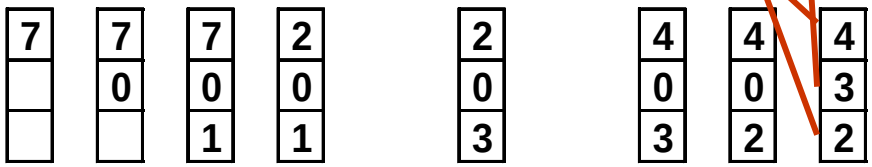
LRU
- Least Recently Used
- Enlever la page la moins récemment utilisée
- On extrapole le passé pour prévoir l'avenir

Remplacement des pages LRU-14-

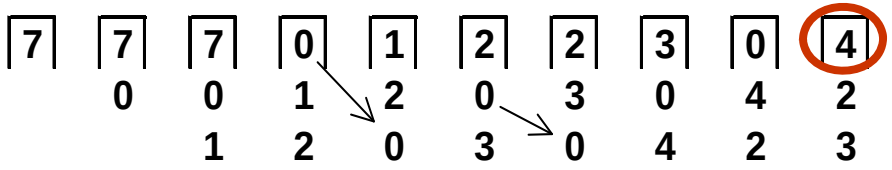
références de pages logiques



pages physiques --> pages logiques



file : # page logique

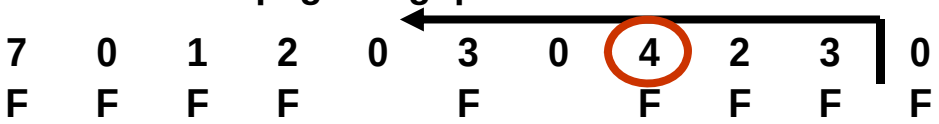


page la plus récente

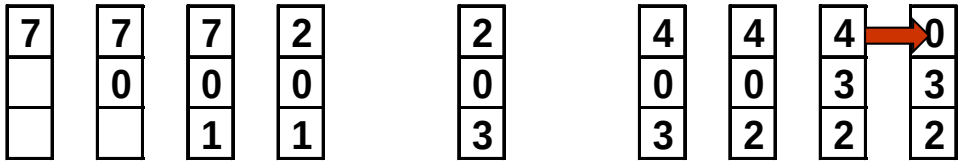
LRU
- Least Recently Used
- Enlever la page la moins récemment utilisée
- On extrapole le passé pour prévoir l'avenir

Remplacement des pages LRU-15-

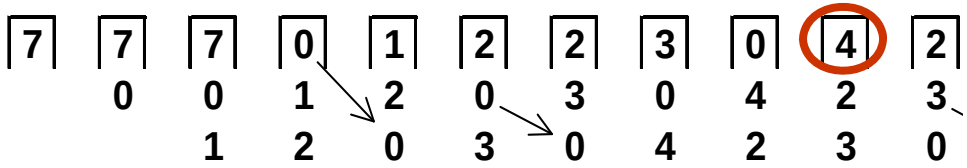
références de pages logiques



pages physiques --> pages logiques



file : # page logique



page la plus récente

LRU
- Least Recently Used
- Enlever la page la moins récemment utilisée
- On extrapole le passé pour prévoir l'avenir

Remplacement des pages LRU-16-

références de pages logiques

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

pages physiques --> pages logiques

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

file : # page logique

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

page la plus récente

LRU

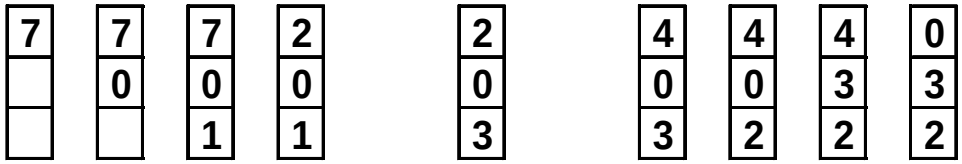
- Least Recently Used
- Enlever la page la moins récemment utilisée
- On extrapole le passé pour prévoir l'avenir

Remplacement des pages LRU-17-

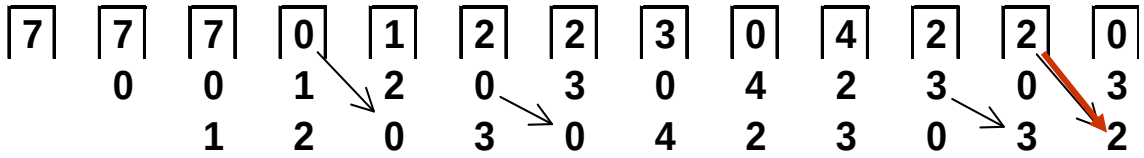
références de pages logiques

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

pages physiques --> pages logiques



file : # page logique



page la plus récente

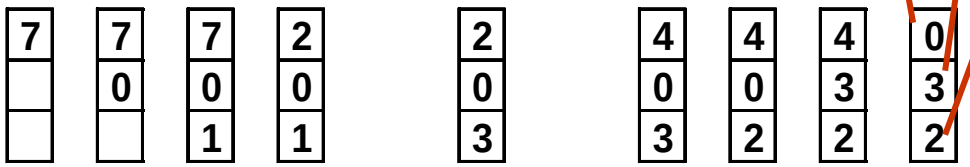
LRU
- Least Recently Used
- Enlever la page la moins récemment utilisée
- On extrapole le passé pour prévoir l'avenir

Remplacement des pages LRU-18-

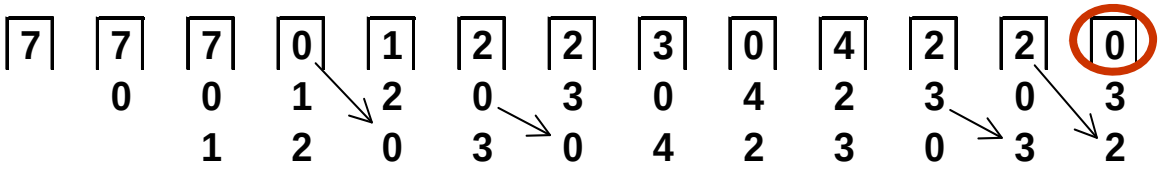
références de pages logiques

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

pages physiques --> pages logiques



file : # page logique

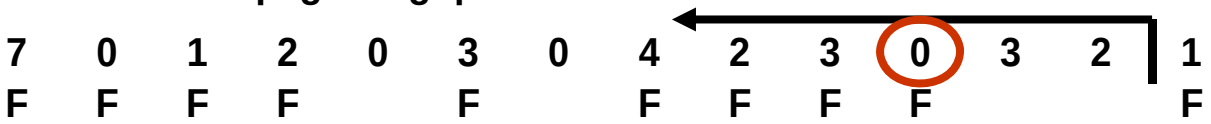


page la plus récente

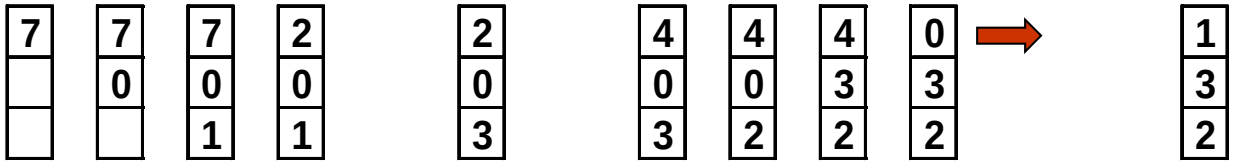
LRU
- Least Recently Used
- Enlever la page la moins récemment utilisée
- On extrapole le passé pour prévoir l'avenir

Remplacement des pages LRU-19-

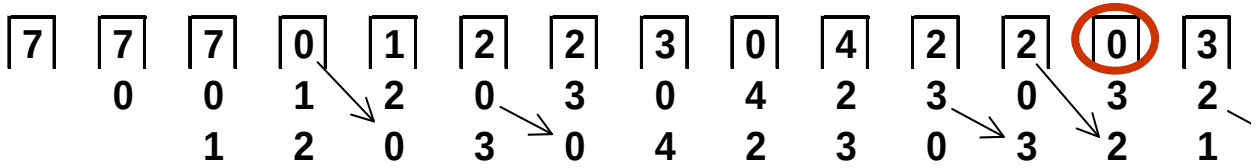
références de pages logiques



pages physiques --> pages logiques



file : # page logique



page la plus récente

LRU
- Least Recently Used
- Enlever la page la moins récemment utilisée
- On extrapole le passé pour prévoir l'avenir

Remplacement des pages LRU-20-

références de pages logiques

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

pages physiques --> pages logiques

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

file : # page logique

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

page la plus récente

LRU

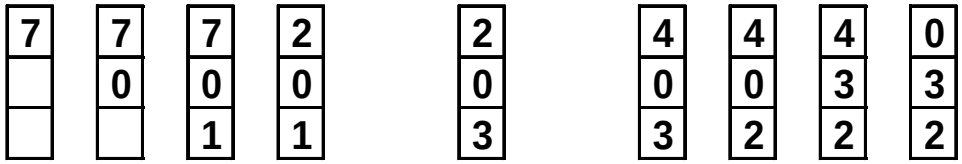
- Least Recently Used
- Enlever la page la moins récemment utilisée
- On extrapole le passé pour prévoir l'avenir

Remplacement des pages LRU-21-

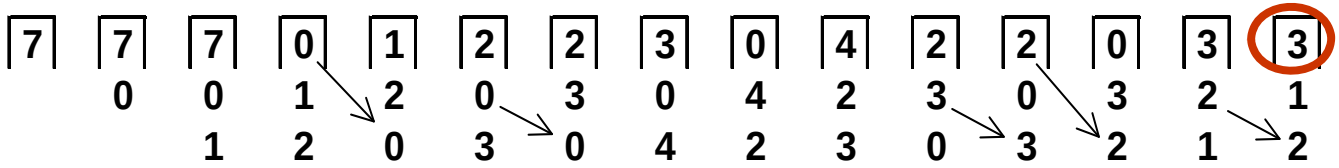
références de pages logiques

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

pages physiques --> pages logiques



file : # page logique



page la plus récente

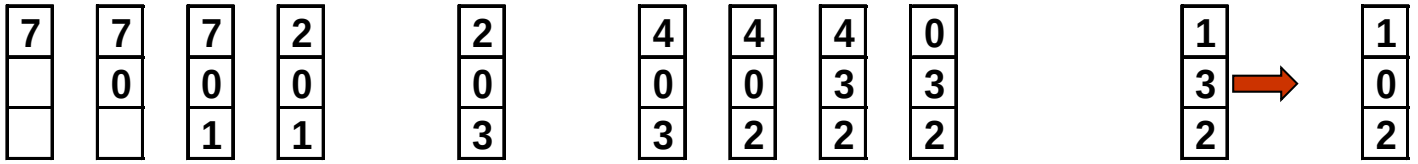
LRU
- Least Recently Used
- Enlever la page la moins récemment utilisée
- On extrapole le passé pour prévoir l'avenir

Remplacement des pages LRU-22-

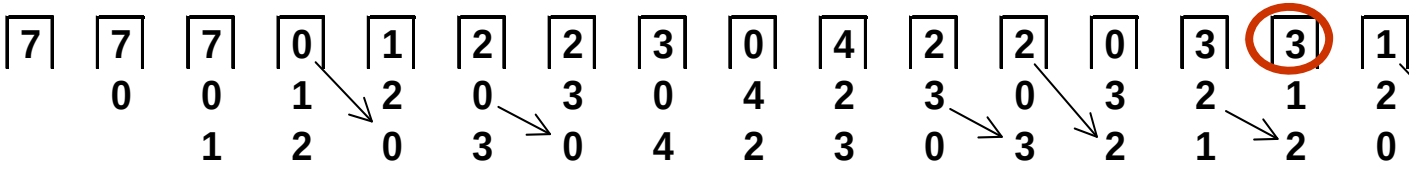
références de pages logiques

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

pages physiques --> pages logiques



file : # page logique



page la plus récente

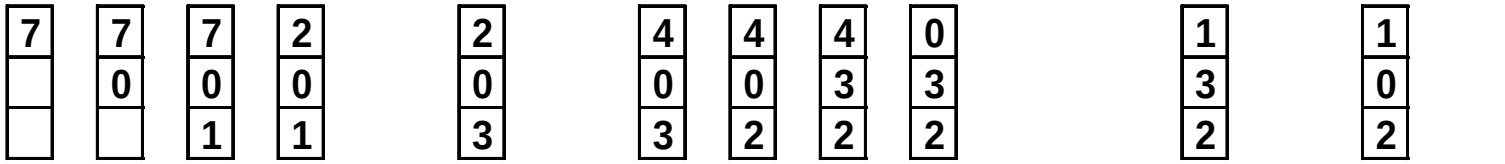
LRU
- Least Recently Used
- Enlever la page la moins récemment utilisée
- On extrapole le passé pour prévoir l'avenir

Remplacement des pages LRU-23-

références de pages logiques

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

pages physiques --> pages logiques



page la plus récente

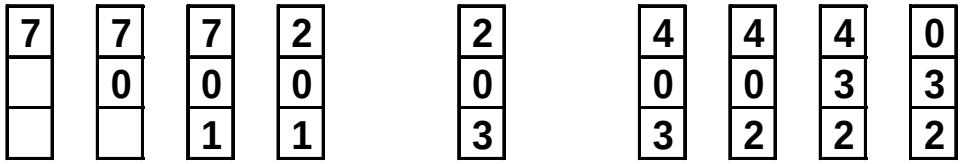
LRU
- Least Recently Used
- Enlever la page la moins récemment utilisée
- On extrapole le passé pour prévoir l'avenir

Remplacement des pages LRU-24-

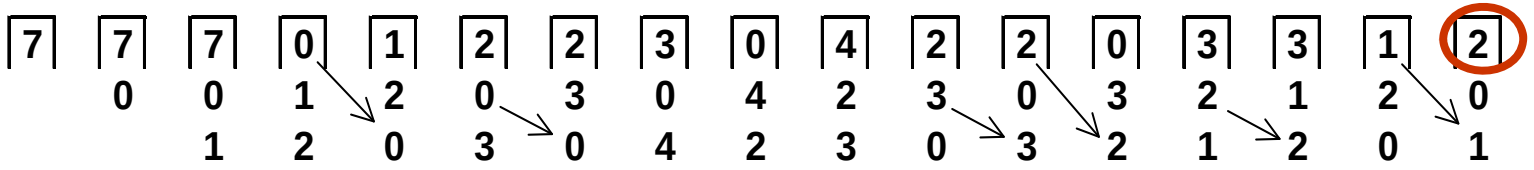
références de pages logiques

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

pages physiques --> pages logiques



file : # page logique



page la plus récente

LRU

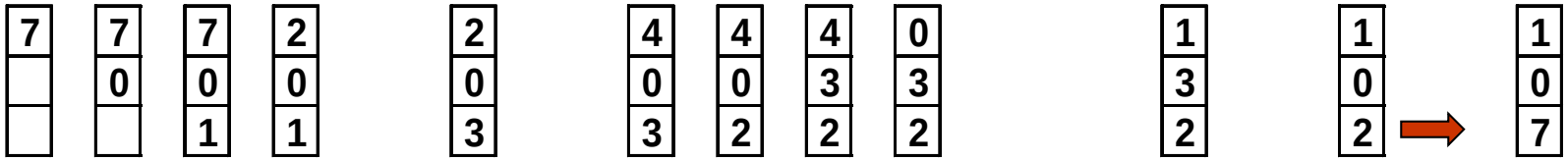
- Least Recently Used
- Enlever la page la moins récemment utilisée
- On extrapole le passé pour prévoir l'avenir

Remplacement des pages LRU-25-

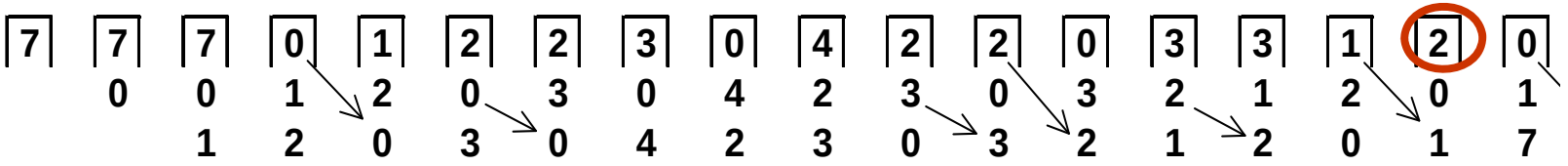
références de pages logiques

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

pages physiques --> pages logiques



file : # page logique



page la plus récente

LRU

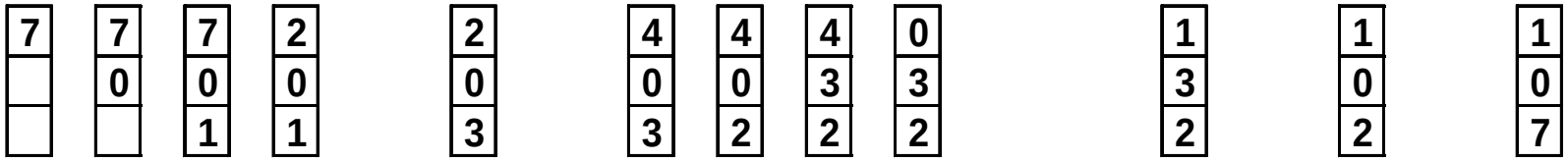
- Least Recently Used
- Enlever la page la moins récemment utilisée
- On extrapole le passé pour prévoir l'avenir

Remplacement des pages LRU-26-

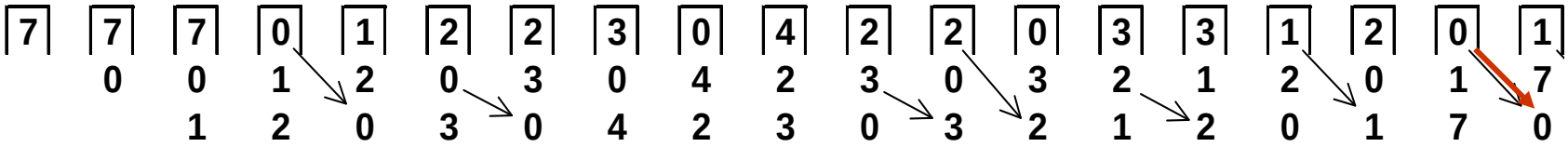
références de pages logiques

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

pages physiques --> pages logiques



file : # page logique



page la plus récente

LRU

- Least Recently Used
- Enlever la page la moins récemment utilisée
- On extrapole le passé pour prévoir l'avenir

Remplacement des pages LRU-27-

références de pages logiques

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

pages physiques --> pages logiques

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

file : # page logique

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

page la plus récente

12 FAUTES DE PAGES

LRU
- Least Recently Used
- Enlever la page la moins récemment utilisée
- On extrapole le passé pour prévoir l'avenir

Algorithme de remplacement de la page non récemment utilisé (NRU : Not Recently Used)

- **Cet algorithme utilise les bits R et M associés à chaque page pour déterminer les pages non récemment utilisés.**
  - Le bit R est positionné à 1 chaque fois une page est référencée
  - Le bit R est remis à 0 à chaque interruption d'horloge.
  - Le bit M est positionné lorsque la page est modifiée (elle n'est plus identique à sa copie sur le disque)

- **Lorsque un défaut de page se produit, l'algorithme NRU sélectionne la page à retirer en procédant comme suite :**
  1. Il vérifie si il existe des pages non référencés, non modifiées (R=0, M=0). Si c'est le cas, il sélectionne une page et la retire
  2. Sinon, il vérifie s'il existe des pages non référencées et modifiées (R=0, M=1). Si c'est le cas, il sélectionne une page et la retire (une sauvegarde de la page retirée sur le disque est nécessaire)
  3. Sinon, il vérifie s'il existe des pages référencées et non modifiées (R=1, M=0). Si c'est le cas, il sélectionne une page et la retire
  4. Sinon, sil sélectionne une page référencée et modifiée et la retire (R=1, M=1). Dans ce cas, une sauvegarde sur le disque de la page retirée est nécessaire

FIN