Allocation de la mémoire centrale

- Mémoire centrale (RAM)
- Organisation logique / physique
- Organisation en partitions
- Pagination et segmentation

▶▶ Mémoire centrale (RAM) ◀◀

La **mémoire centrale** (**RAM** pour **Random Access Memory**) est une zone de stockage composée **d'octets** (8 bits).

Chaque octet est repéré par un **adresse physique** (sur 32 ou 64 bits).

```
             +---+
0x00000000   |   |
             +---+      2^32 adresses soit
             |   |      2^32 octets soit
             | . |      2^22 kilo-octets soit
   ...       | . |      2^12 méga-octets soit
             | . |      2^2  giga-octets soit
             |   |      4    giga-octets
             +---+
0xFFFFFFFF   |   |
             +---+
```

La mémoire physique est **contigüe** (les adresses varient de N à M) et **uniforme** (tous les processeurs ont accès à tous les octets).

Il existe trois opérations : **lecture**, **écriture**, **exécution** (lecture pour exécution) donc **trois permissions**.

▶ Utilisation de la mémoire ◀

Un exemple :

```
int x;

void main(void) {
    sleep(1);
    printf("%x␣", &x);
    sleep(1);
}
```

Résultats de l'exécution

```
600800 600800 600800 600800 600800 600800 600800 600800 600800
```

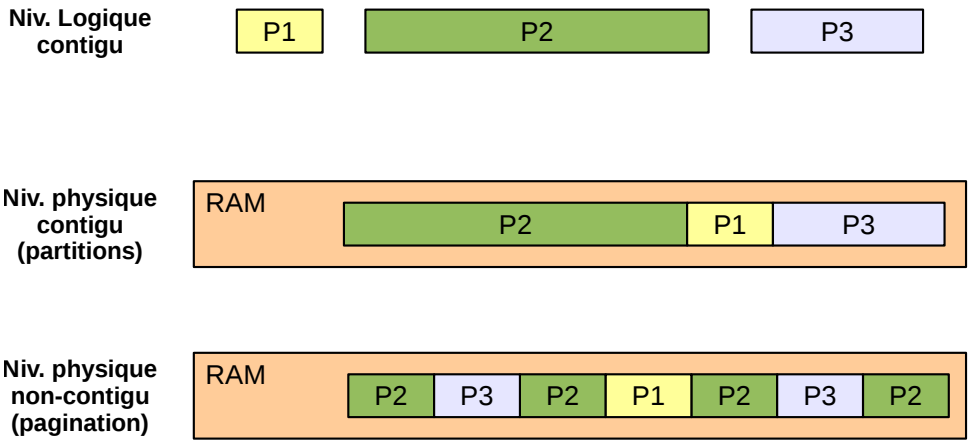
- Chaque processus travail dans une **mémoire logique** qui est une partie de la mémoire centrale.
- Les octets de la mémoire logique sont repérés par des **adresses logiques**.
- Á l'exécution, les processus génèrent des adresses logiques qui varient de 0 à $N-1$ (N étant la **taille de la mémoire logique du processus**).

▶▶ Allocation de mémoire ◀◀

- **Correspondance** entre adresses logiques et adresses physiques,
  - ▷ **fixe** : établie à la compilation
  - ▷ **statique** : établie au chargement
  - ▷ **dynamique** : variable dans le temps
- **Gestion** de la mémoire physique.
- **Partage** de données entre processus.
- **Protection** de chaque processus.

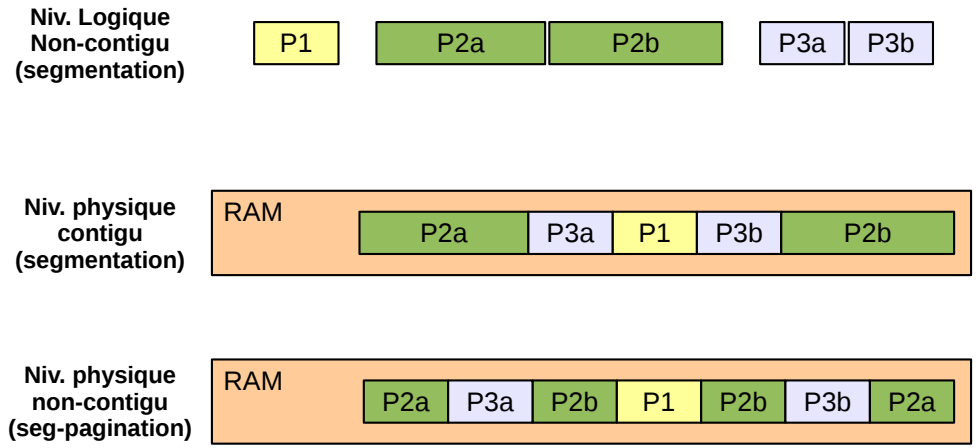
► Mémoire logique contiguë ◄

La mémoire logique des processus est constitué **d'un seul morceau** (une partition) :



► Mémoire logique non contiguë ◄

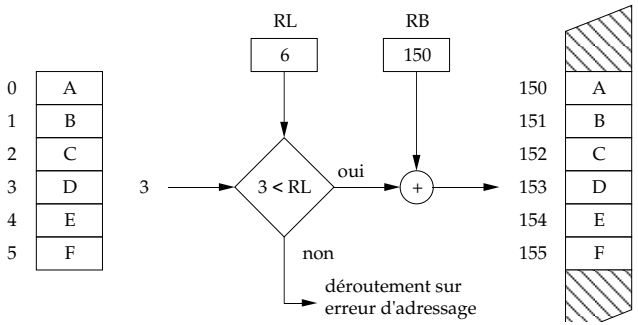
La mémoire logique des processus est constitué de **plusieurs morceaux** (segments) :



► Système à partitions variables ◄

Les partitions sont **allouées et libérées à la demande** (création ou fin d'un processus).

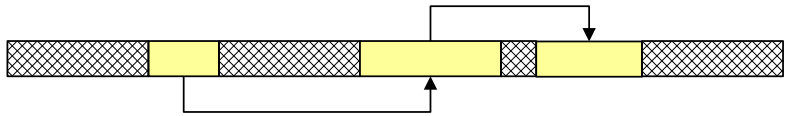
Le **registre de base** pointe sur la partition et le **registre limite** en indique la taille.



`RB` et `RL` sont utilisés **par la CPU** pour traduire **à chaque accès mémoire** les adresses logiques en adresses physiques.

► Fragmentation externe/interne ◄

Algorithme d'allocation par chaînage des zones libres (mais le parcours de la liste est coûteux) :



Algorithme d'allocation par subdivision :

8					
A	2		4		Alloc. de A, longueur = 2
A	B	1	4		Alloc. de B, longueur = 1
A	B	1	C	2	Alloc. de C, longueur = 2
2	B	1	C	2	Libération de A
2	B	1	4		Libération de C
8					Libération de B

Conséquences :

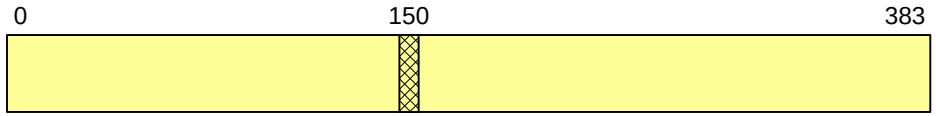
- La **fragmentation externe** est due à l'émiettement de la mémoire lors des allocations/libérations.
- La **fragmentation interne** c'est l'unité de mémoire minimum que le S.E. est capable de gérer (généralement plusieurs Kilo-octets).

▶▶ Mémoire paginée ◀◀

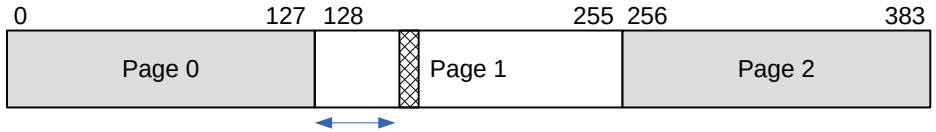
- Principe
- Correspondance des adresses
- Mémoires associatives
- Partage de pages

▶ Mémoire paginée : principe ◀

Mémoire linéaire de 384 octets (3 x 128)



Mémoire paginée de 3 pages (3 x 128 octets)

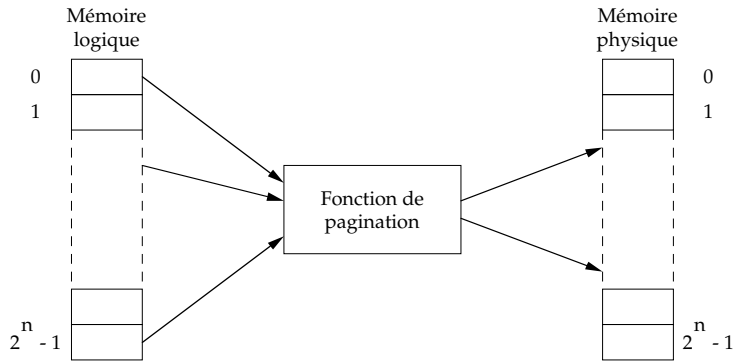


Adresse linéaire 150 = **adresse paginée < page 1, déplacement 22 >** avec 22 = 150 % 128

Découpage d'une adresse linéaire : avec une page de 128 octets (2^7) :

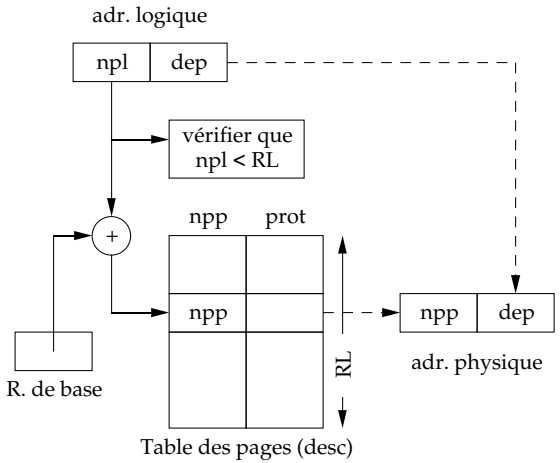
```
150  =   00000000000000000000001   0010110
         <------- 25 bits ------->  <- 7 ->
                     1                22     (16+4+2)
150  =            (1 * 128)       +   22
```


La **fonction de pagination** assure la correspondance entre numéro de page logique et numéro de page physique.



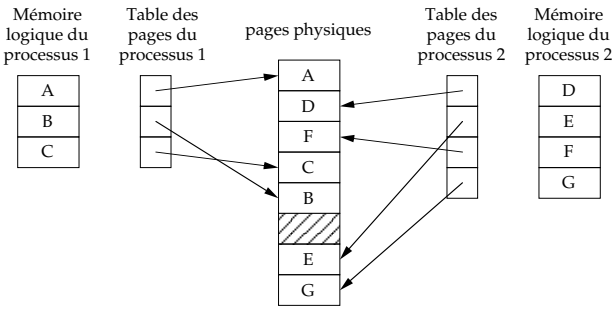
▶ Pages logiques versus pages physiques ◀

Pour chaque processus, le système prépare une **table de pages logiques** (notée `desc` ci-dessous).



▶ Exemple et discussion ◀

Un exemple avec deux processus :



Avantages :

- Gestion mémoire **plus simple** (liste des pages libres)
- Compactage **inutile**
- Protections différentes pour chaque page

Inconvénients :

- temps d'accès doublé
- nécessite une **PMMU** (**Page Memory Management Unit**)

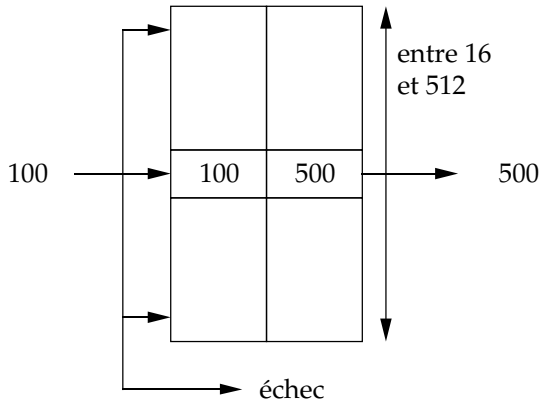
▶ Comportement des processus ◀

Comportement **en moyenne** des processus :

- **Non uniformité** : 20% des pages regroupent 90% des accès
- **Principe de localité** :
  - ▷ stabilité des accès sur une **courte période**
  - ▷ l'activité actuelle est une **bonne estimation** de l'activité future

▶ Mémoires associatives ◀

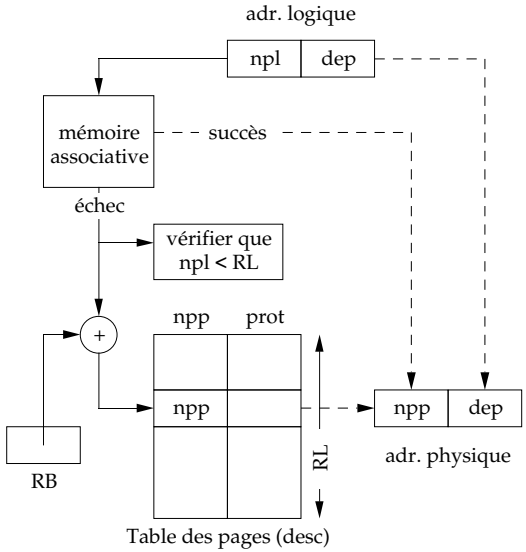
Principe des **mémoires associatives** :



- **Rapidité** : les tests sont faits **en parallèle** (quelques nanosecondes)
- Ces circuits sont **très couteux**

▶ Mémoires associatives et pagination ◀

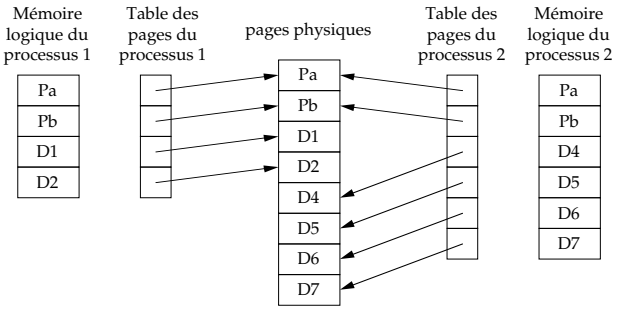
Principe : retenir les derniers couples (**page logique**, **page physique**), pour éviter l'accès mémoire à la table des pages.



Conséquences :

- Il faut mettre à jour la mémoire associative après les échecs
- Il faut vider la mémoire associative lors des changement de processus
- Le taux de réussite est lié à la taille de la M.A. (entre 80% et 95%).
  - ▷ $0,80 \times (100 + 20) + 0,20 \times (100 + 100 + 20) = 140$ ns
  - ▷ $0,95 \times (100 + 20) + 0,05 \times (100 + 100 + 20) = 125$ ns

▶ Partage de pages entre processus ◀



Les pages contenant le programme (Pa et Pb) sont partagées, mais les pages de données (D1, …, D7) ne le sont pas.

- Les pages contenant le programme Pa et Pb sont **partagées**, tandis que les pages Dx ne le sont pas.
- Pour une même page physique, il est possible d'avoir des **protections différentes** suivant le processus qui l'utilise.

Gestion d'une mémoire virtuelle paginée

▶▶ Mémoire virtuelle paginée ◀◀

Principe : les programmes utilisent 20% de leur page, donc il est inutile de toutes les conserver en mémoire.

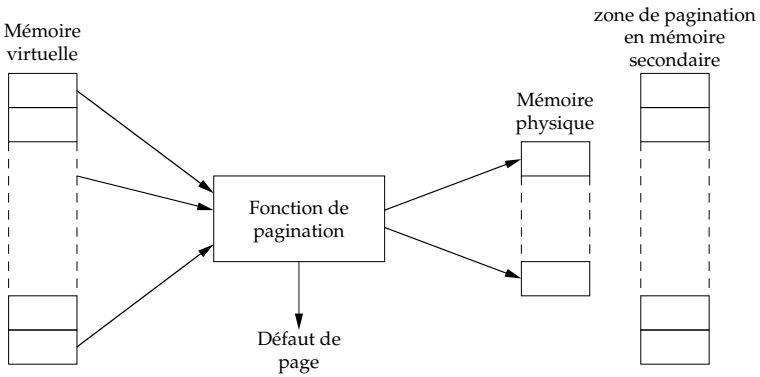
Exemple : 1000 pp = 10 processus de 100 pages logiques ou 50 processus de (100 x 0,2) pages utiles.

Le système doit détecter (avec l'aide du matériel) :

- les pages **inutilisées** (**réquisition**)
- les pages **utiles** et **présentes** en mémoire physique
- les pages **utiles** et **absentes** de la mémoire physique (**défaut de page**)

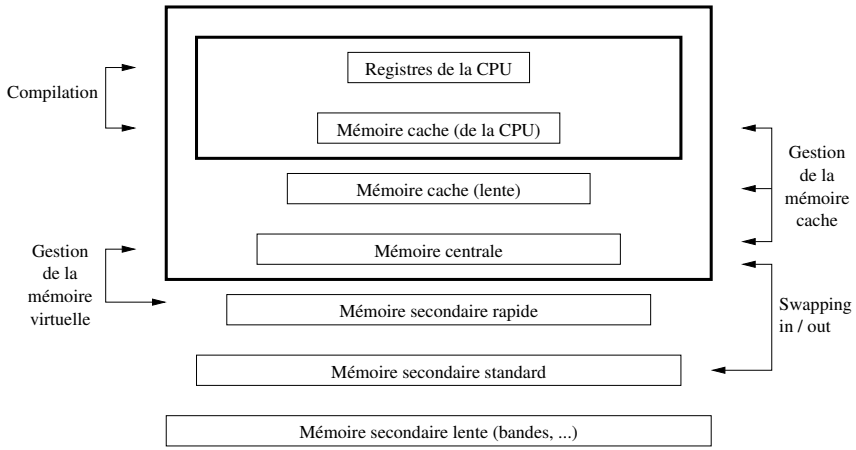
▶ Fonction de pagination ◀

Fonction de pagination virtuelle :



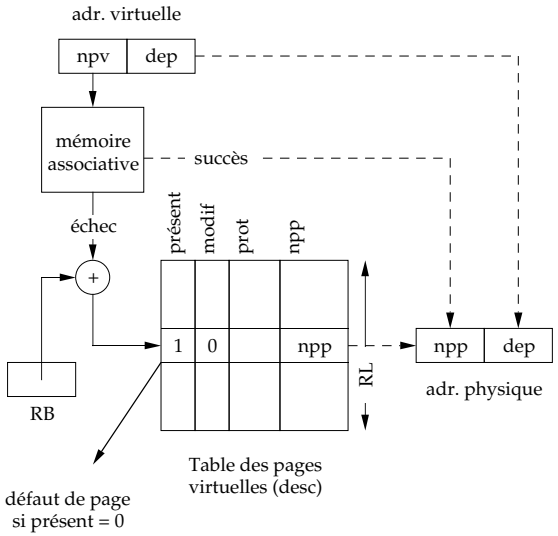
▶ Hiérarchie de mémoire ◀

La mémoire virtuelle implante la **gestion d'un cache** :

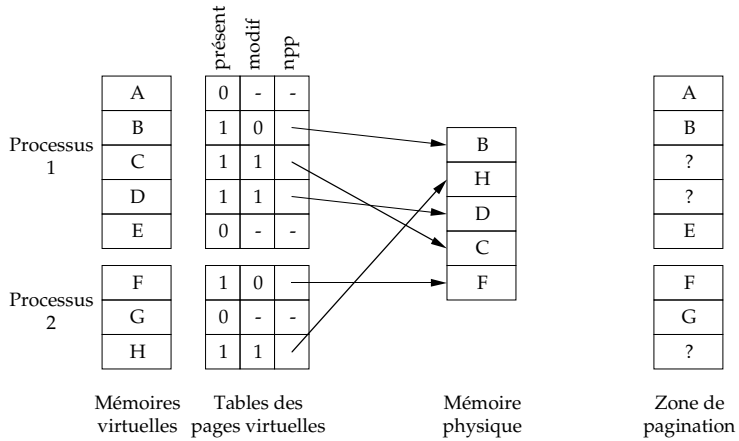


▶ Adresses virtuelles versus adresses physiques ◀

Pour chaque processus, le système prépare une **table des pages virtuelles** (pointée par le registre de base) :



▶ Un exemple sur deux processus ◀



▶ Pagination a plusieurs niveaux ◀

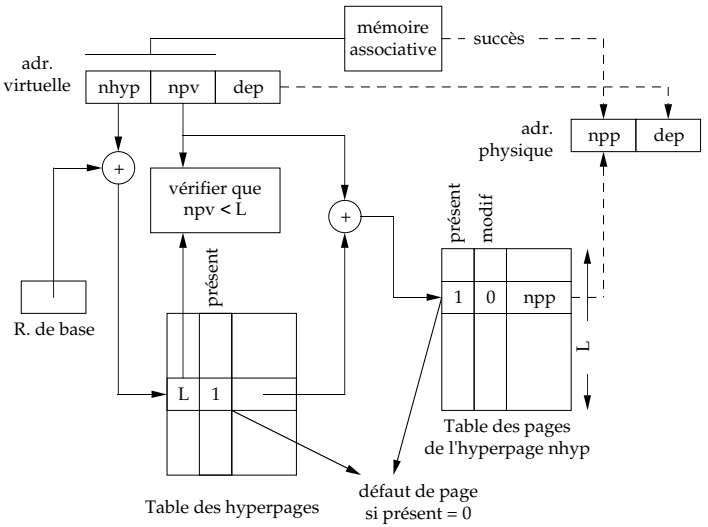
Principe : Si la mémoire est importante le nombre de pages augmente et la table des pages devient **imposante**.

Exemple : Une mémoire de 256 Mo (soit 2^{28} octets) est divisée en $2^{28}/2^{10} = 2^{18}$ pages. La table des pages a donc 2^{18} entrées soit **1 Mo**.

Conséquence : malgré la pagination, nous devons allouer des ensembles de pages **contigus** pour les tables de pages.

Solution : paginer la table des pages ce qui revient à faire une pagination à **deux niveaux**.

Organisation :



Il peut y avoir jusqu'à **5 niveaux** de pagination. Dans ce cas

$$\text{temps d'accès} = (0,98 \times 120) + (0,02 \times 520) = 128$$

▶ Discussion sur la taille des pages ◀

La taille des pages doit être **grande** pour

- diminuer le **nombre de pages**, donc le nombre de défauts de page et la taille de la table des pages
- optimiser le **temps de transfert** vers ou depuis la zone de pagination
- utiliser des mémoires centrales de plus en plus **grandes**

La taille des pages doit être **petite** pour

- limiter la **fragmentation interne**
- définir avec **plus de précision** les pages utiles

Actuellement la taille des pages varie entre **1 ko** et **32 ko**.

Certains systèmes autorisent **plusieurs tailles** différentes.

▶▶ Algorithmes de remplacement ◀◀

Algorithmes :

- Algorithme **optimale** (base de référence) : choisir la page virtuelle qui est utilisée le **plus tard possible** ou qui n'est plus utilisée.
- Algorithme **aléatoire** (le moins bon).
- Algorithme **FIFO** (il ne tient pas compte de l'utilisation des pages).
- Algorithme **LRU** (*Least Recently Used*) est basé sur le **principe de localité** : choisir la page dont la date du dernier accès est la plus ancienne.

Principe : On choisit en priorité les **pages virtuelles propres** (qui n'ont pas été modifiées).

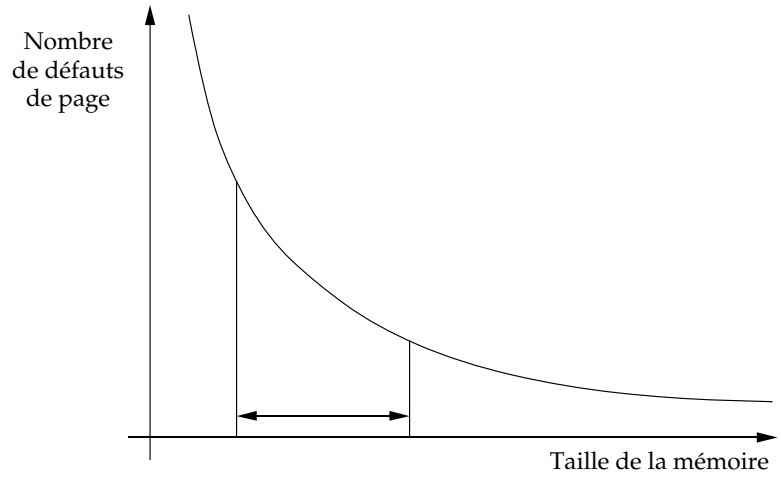
▶ Algorithme FINUFO (First In Not Used First Out) ◀

Principes :

- Un **pointeur global** de page physique P
- Un **bit d'utilisation** par page physique noté `U[k]`
- `U[k]` est forcé à 1 après **chaque accès** à la page physique `k` (**PMMU**)

```
choisir_victime_FINUFO ()
    | tant que (U[P] = 1) faire
    |     | U[P] := 0
    |     | P := (P + 1) mod NB_PAGES_PHYSIQUES
    | fin faire
    | U[P] := 1
    | victime := P
    | P := (P + 1) mod NB_PAGES_PHYSIQUES
    | renvoyer victime
```

Performances :

OPT > LRU > LFU > FINUFO > FIFO > ALEA

▶ Comportement en mémoire virtuelle paginée ◀