

# II

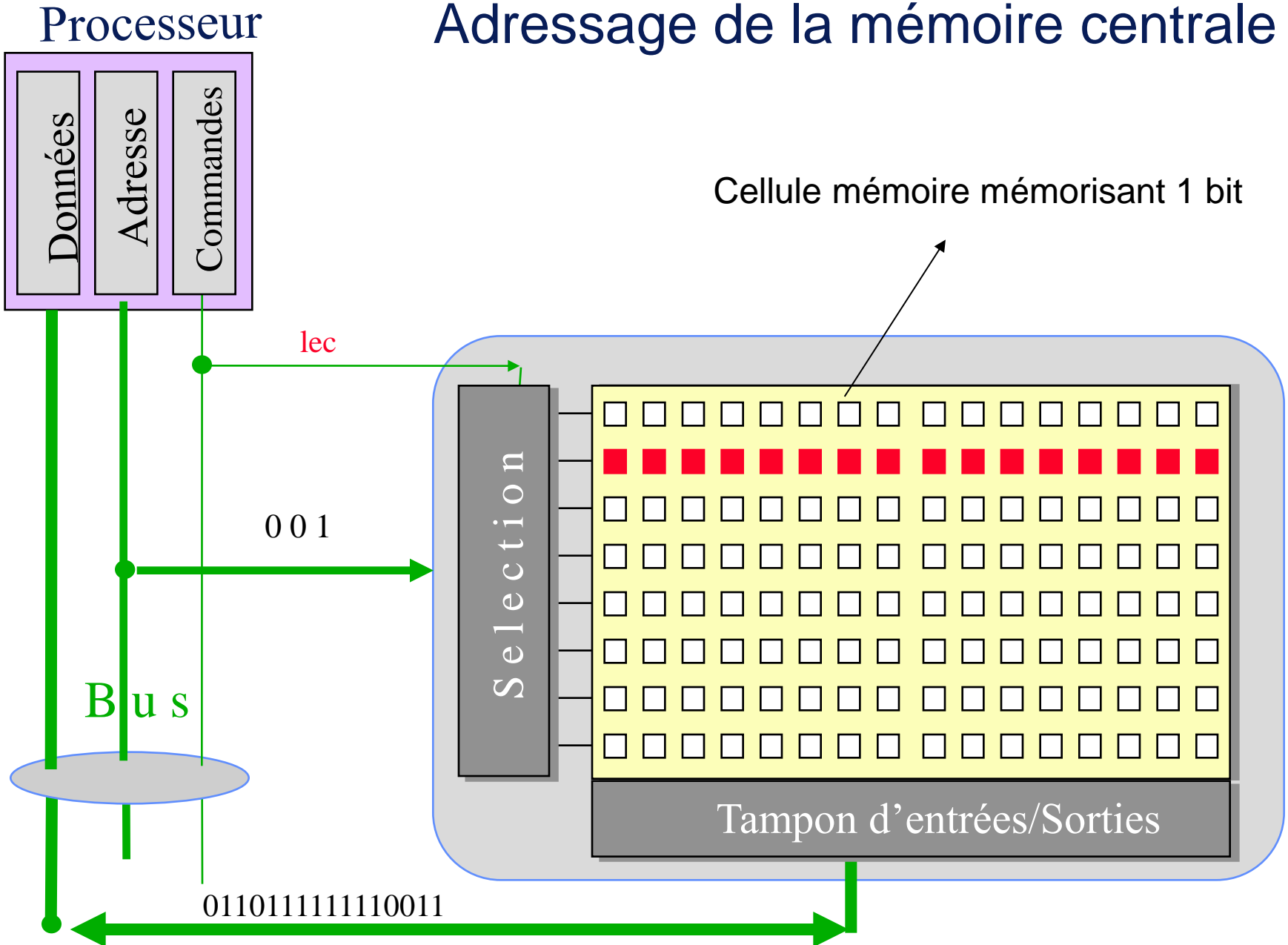
## Gestion de la mémoire centrale

### Allocation de la mémoire physique

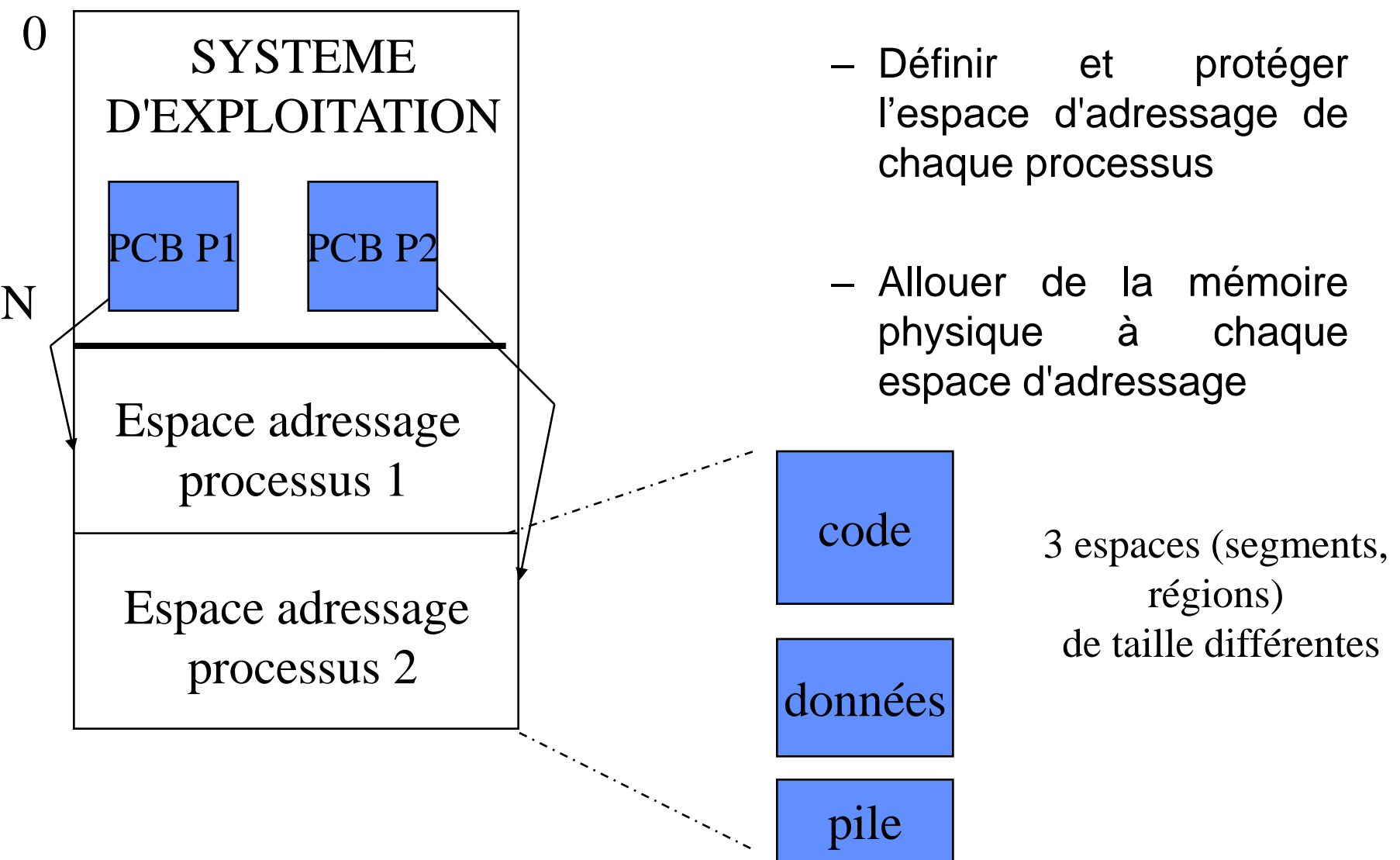
### Illustration sous Linux



# Adressage de la mémoire centrale



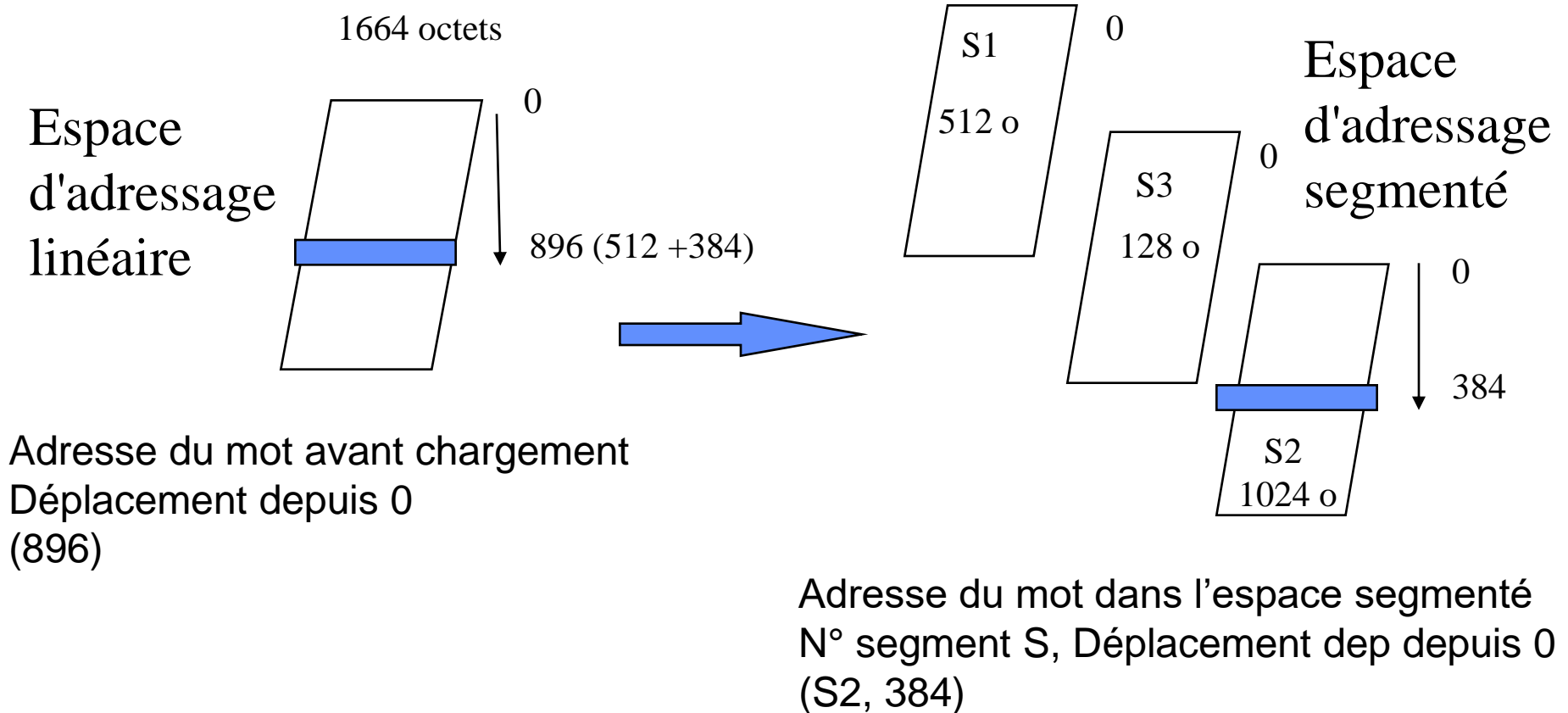
# Multiprogrammation et gestion mémoire



- Définir et protéger l'espace d'adressage de chaque processus
- Allouer de la mémoire physique à chaque espace d'adressage

3 espaces (segments, régions) de taille différentes

# Espace d'adressage segmenté adresse logique segmentée ( n°segment S, déplacement dep dans le segment)



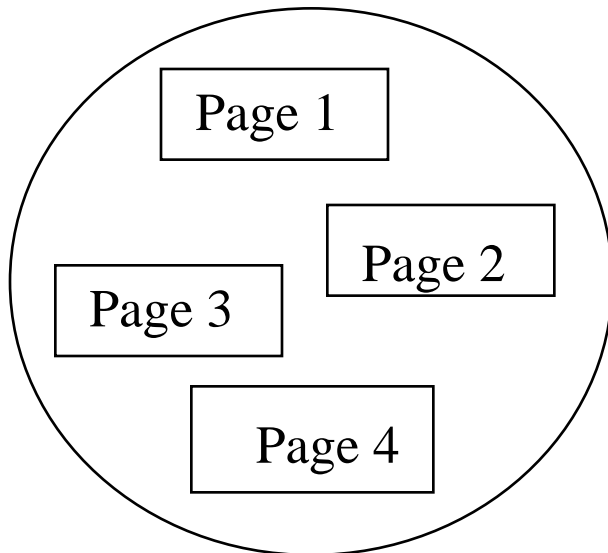
# La pagination

L'espace d'adressage du programme est découpé en morceaux linéaires de même taille : **la page**.

L'espace de la mémoire physique est lui-même découpé en morceaux linéaires de même taille : la **case ou cadre de pages**

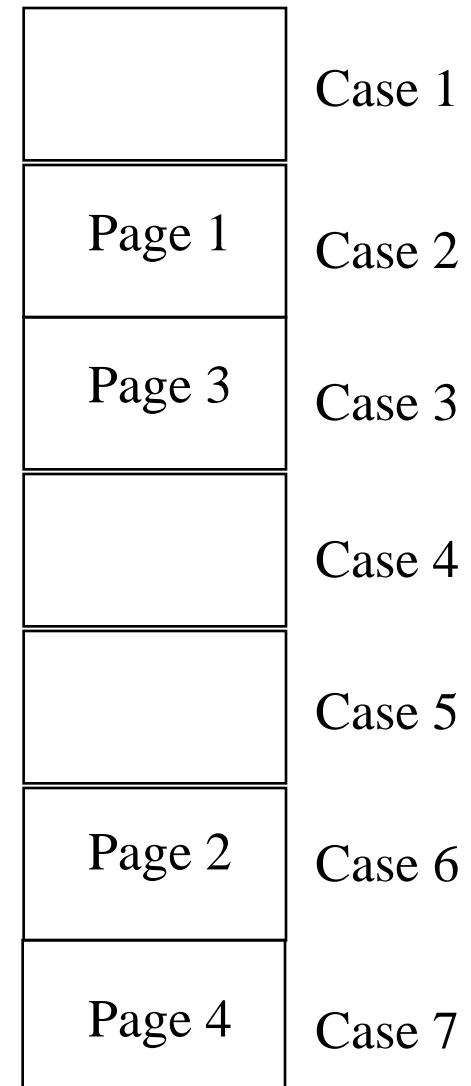
La taille d'une case est égale à la taille d'une page

→ **Les segments de l'espace d'adressage sont eux-mêmes paginés**



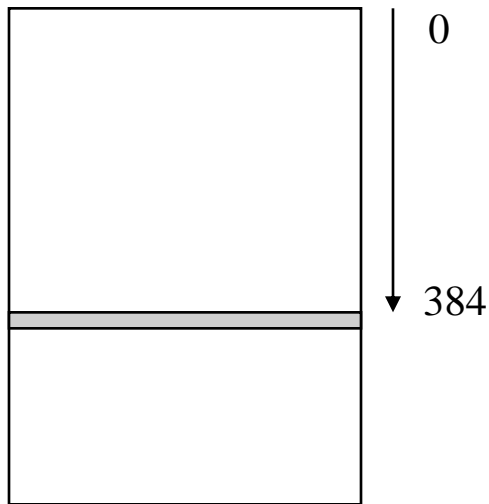
Espace d'adressage  
du programme

## Mémoire

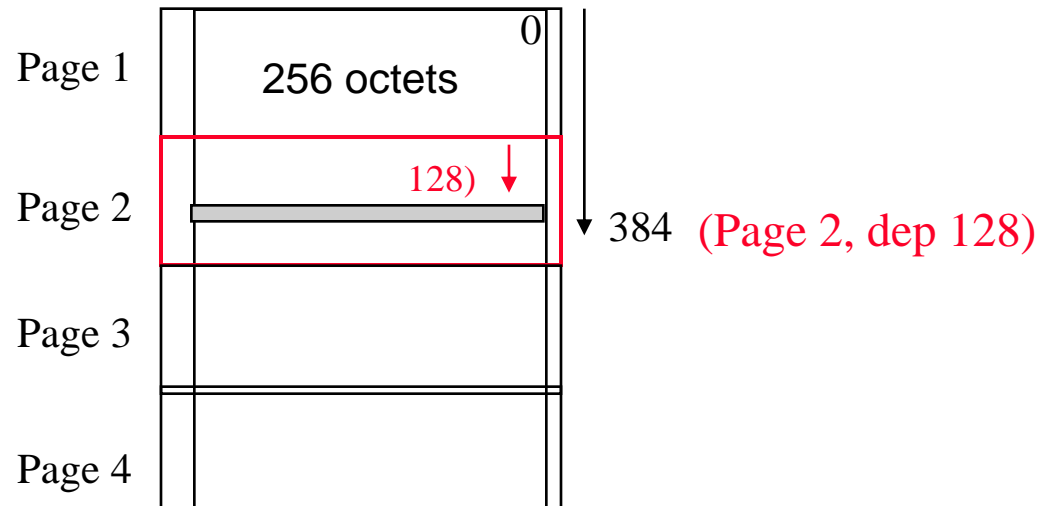


# PAGINATION d'un segment

- L'espace d'adressage linéaire du segment est coupé en portions de taille fixe et égale à l'unité d'allocation de la mémoire centrale : **les pages**. Chaque adresse au sein du segment devient une **adresse paginée** formée d'un couple (numéro de page, déplacement dans la page)



Espace d'adressage  
du segment S2 (1024)  
**LINEAIRE**  
Adresse linéaire  
(déplacement depuis 0)



Espace d'adressage  
du segment S2  
**PAGINE**  
Adresse paginée (logique)  
(n° de page, déplacement dans la page depuis 0)

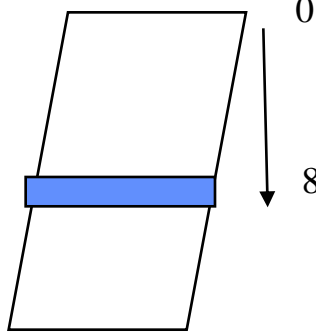
# Espace d'adressage segmenté et paginé

## adresse logique segmentée paginée

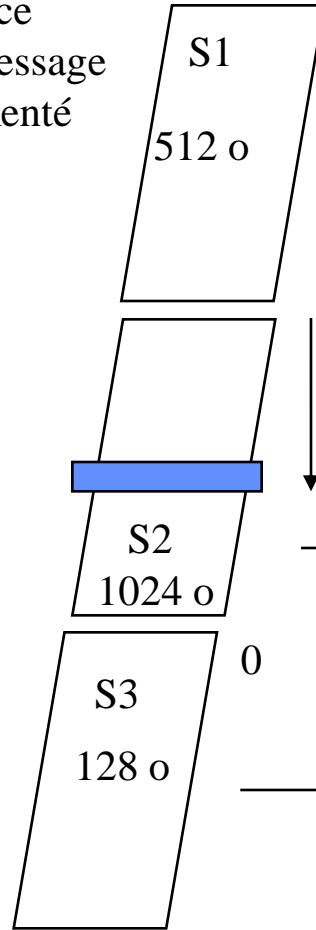
( n°segment S, n°page du segment, déplacement dep dans la page)

Espace  
d'adressage  
linéaire

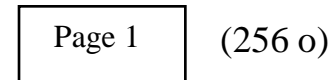
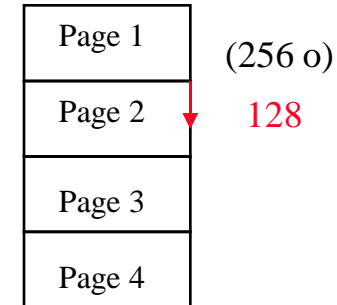
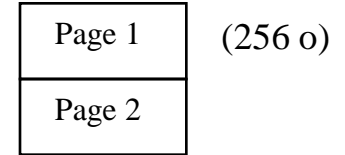
1664 octets



Espace  
d'adressage  
segmenté



Espace d'adressage segmenté,  
paginé



Adresse du mot  
Déplacement depuis 0  
(896)

Adresse du mot dans l'espace segmenté  
N° segment S, Déplacement dep depuis 0  
(S2, 384)

Adresse du mot dans l'espace segmenté, paginé  
N° segment S, page p dans segment, déplacement  
dep' dans la page depuis 0  
(S2, P2, 128)

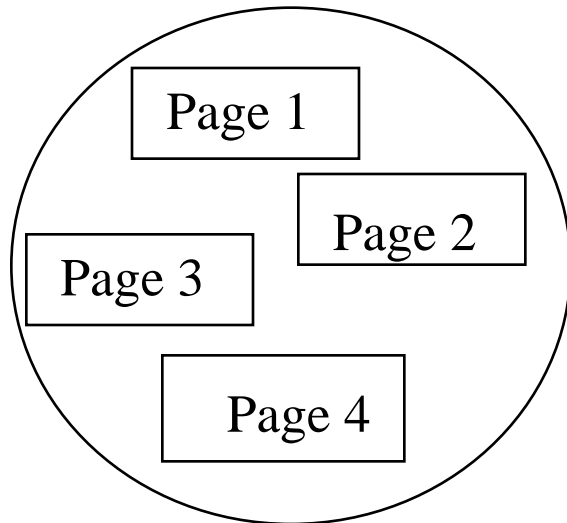
# Allocation des segments en mémoire centrale

L'espace d'adressage du segment est découpé en morceaux linéaires de même taille : **la page**.

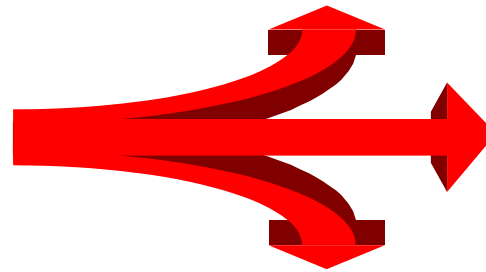
L'espace de la mémoire physique est lui-même découpé en morceaux linéaires de même taille : la **case ou cadre de pages**

La taille d'une case est égale à la taille d'une page

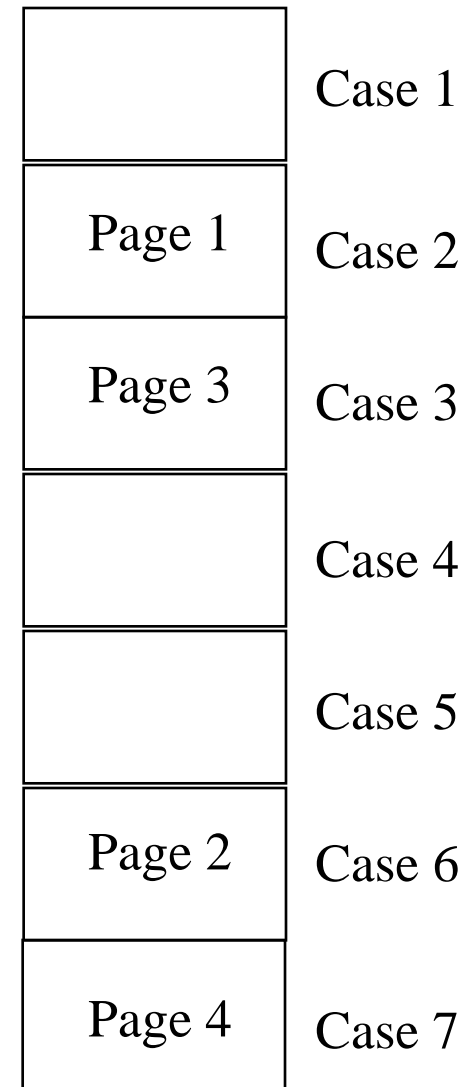
→ Les pages d'un segment sont placées dans n'importe quelle case libre de la mémoire centrale



Espace d'adressage  
du segment S2

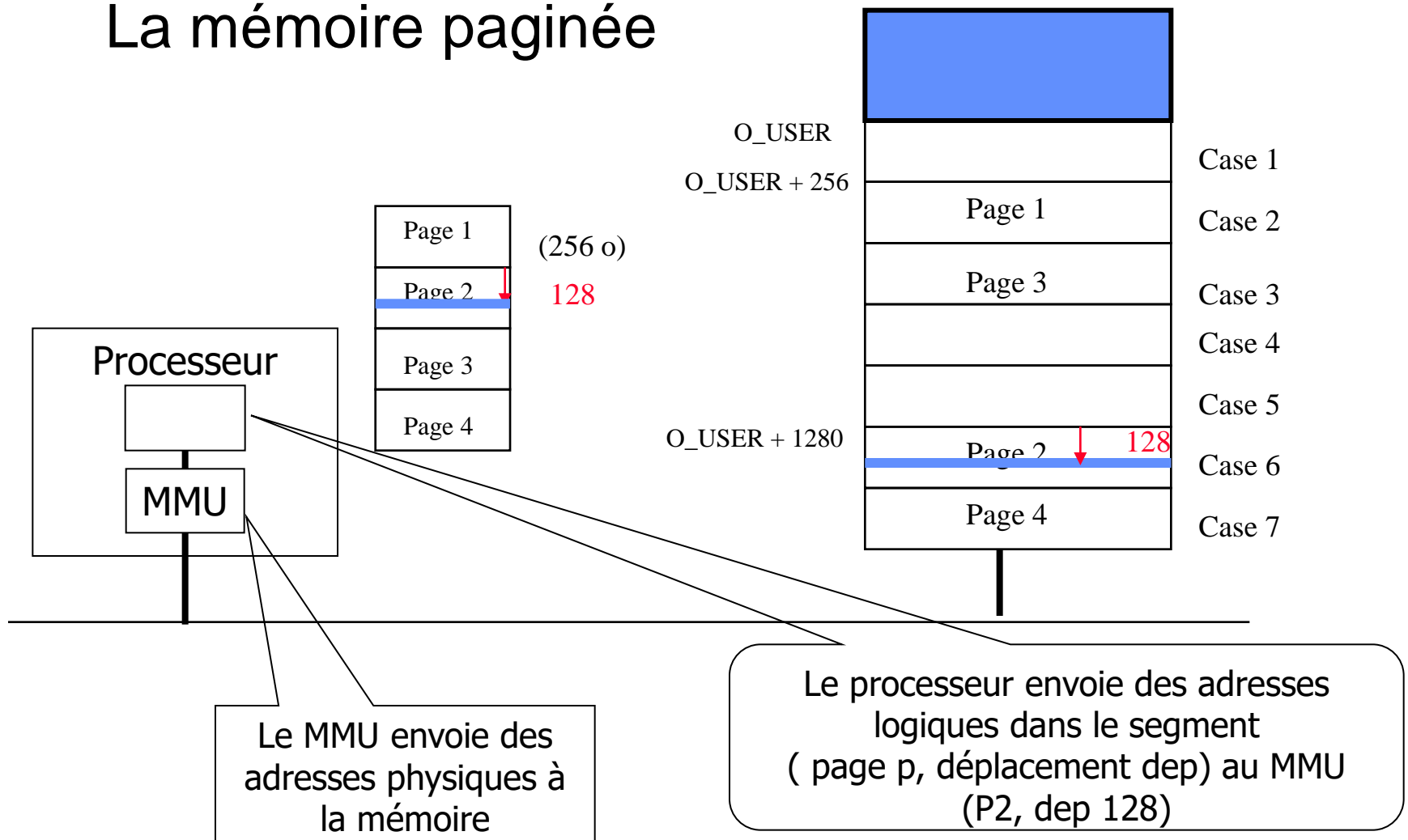


## Mémoire





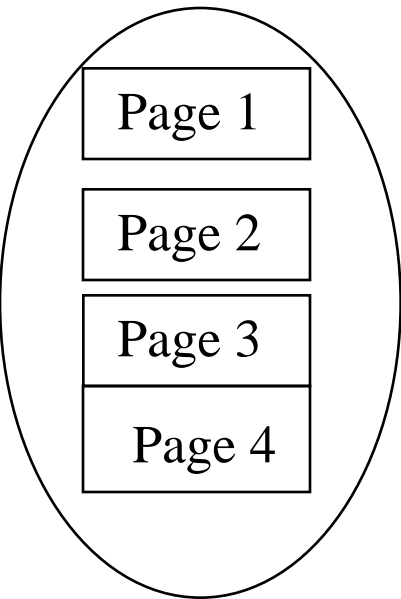
# La mémoire paginée



☞ Il faut convertir l'adresse paginée en son équivalent adresse physique  
Adresse physique = **adresse implantation case contenant la page adr ( $O\_User + 1280$ ) + déplacement dep (128)**

➤ Table des pages

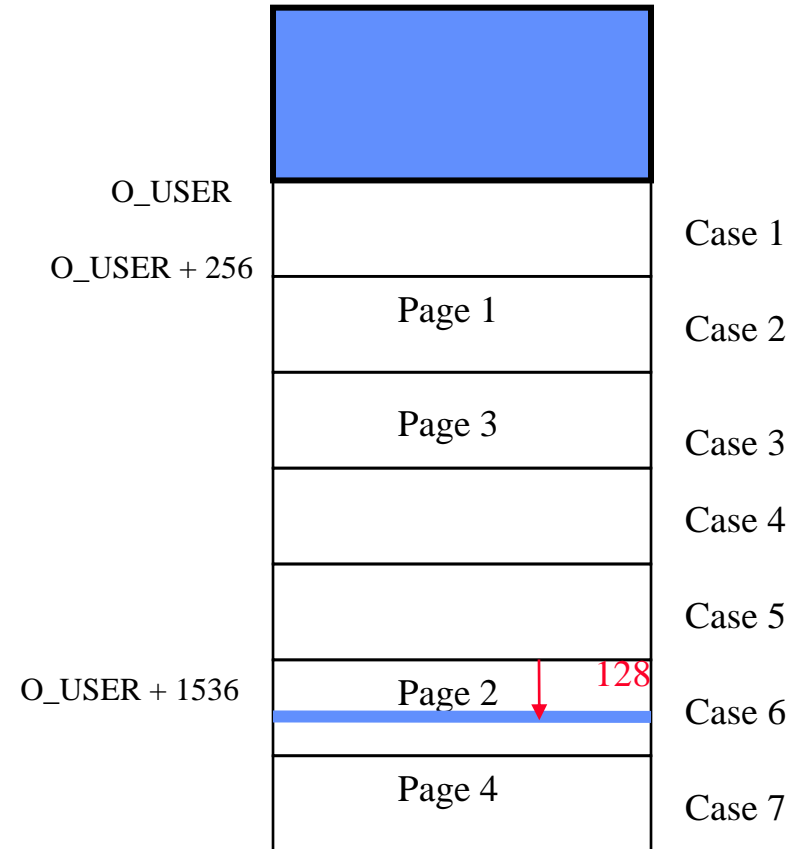
# La mémoire paginée



Espace d'adressage  
du segment S2

**Table des pages  
du segment**

Numéro page	adresse case
1	Adr C2 (O_User + 256)
2	Adr C6 (O_User + 1280)
3	Adr C3 (O_User + 512)
4	Adr C7 (O_User + 1536)



## Table des pages du segment

Adresse logique (P2, 128)

P2

Numéro page	adresse case
1	Adr C2 (O_User + 256)
2	Adr C6 (O_User + 1280)
3	Adr C3 (O_User + 512)
4	Adr C7 (O_User + 1536)

Processeur

MMU

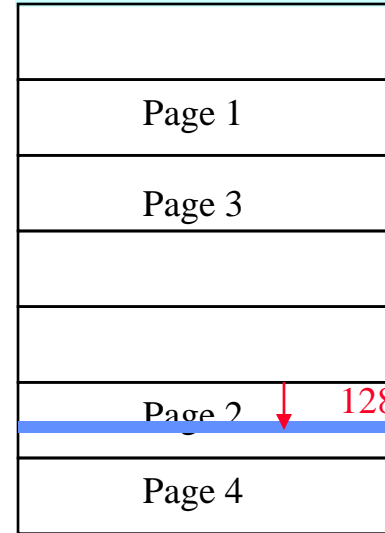
O\_USER + 1280

O\_USER + 1280 + 128

Le MMU envoie des adresses physiques à la mémoire

## Table des pages

O\_USER  
O\_USER + 256



Case 1  
Case 2  
Case 3  
Case 4  
Case 5  
Case 6  
Case 7

☞ Il faut convertir l'adresse paginée en son équivalent adresse physique  
 Adresse physique = **adresse implantation case contenant la page adr (O\_User + 1280) + déplacement dep (128)**

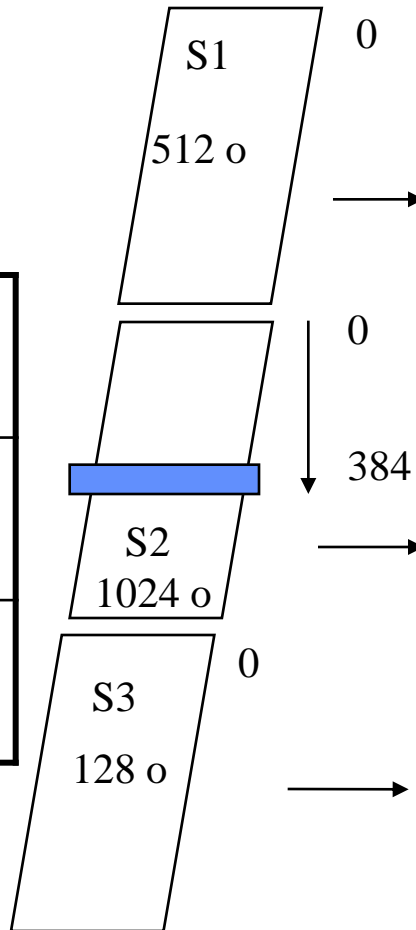
# Espace d'adressage segmenté et paginé

## Structures de données

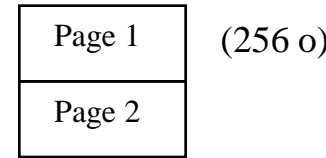
Espace d'adressage de processus segmenté

Table des segments du processus

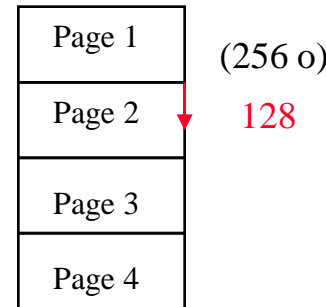
S1	table des pages S1	Taille S1 Droits accès
S2	table des pages S2	Taille S2 Droits accès
S3	table des pages S3	Taille S3 Droits accès



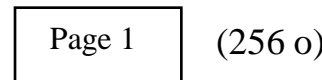
Espace d'adressage segmenté, 1 table des pages paginé pour chaque segment



Page 1	Adr case
Page 2	Adr case



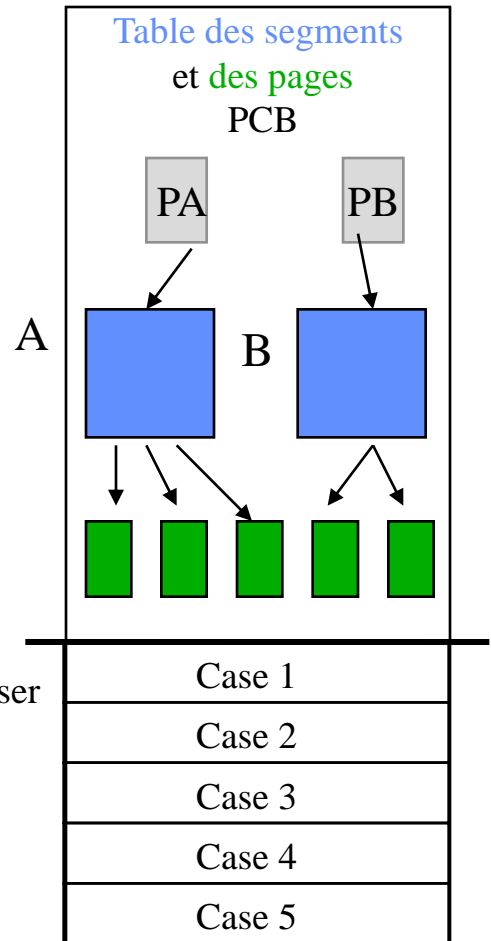
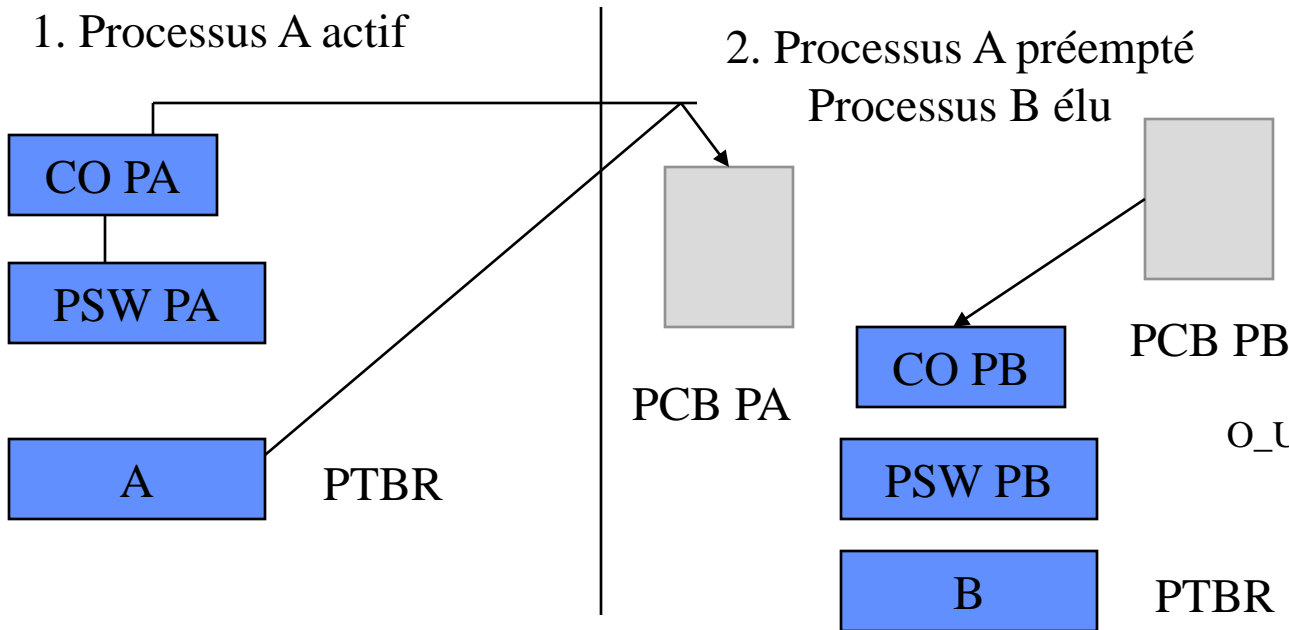
Page 1	Adr Case 2
Page 2	Adr Case 6
Page 3	Adr Case 3
Page 4	Adr Case 7



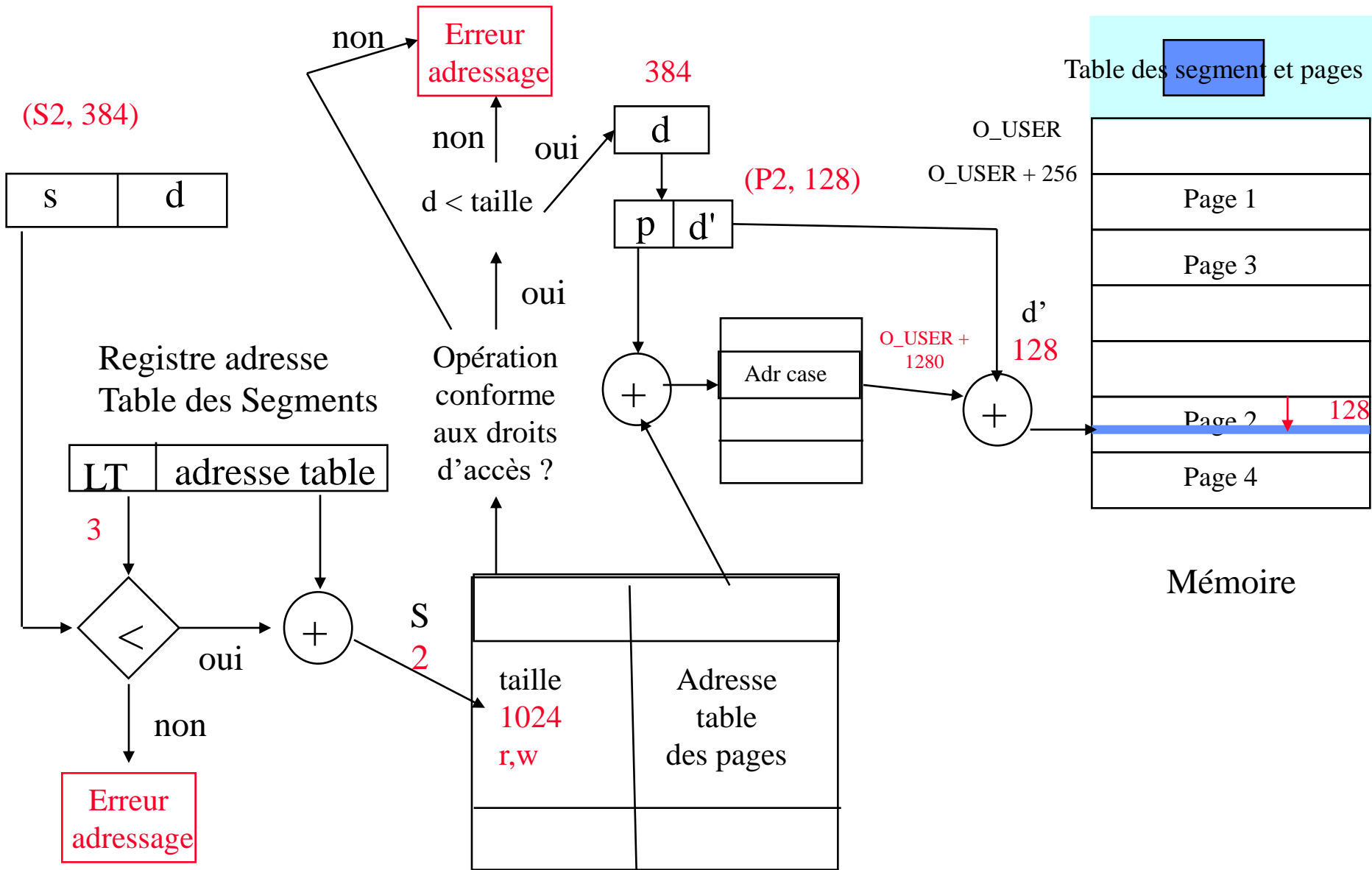
Page 1	Adr case
--------	----------

# Implémentation des structures de données (table des pages et table des segments)

- Ces structures sont des tables logicielles pointées depuis le PCB de chaque processus.
  - commuter de processus = charger le registre PTBR avec l'adresse de la table des segments du processus

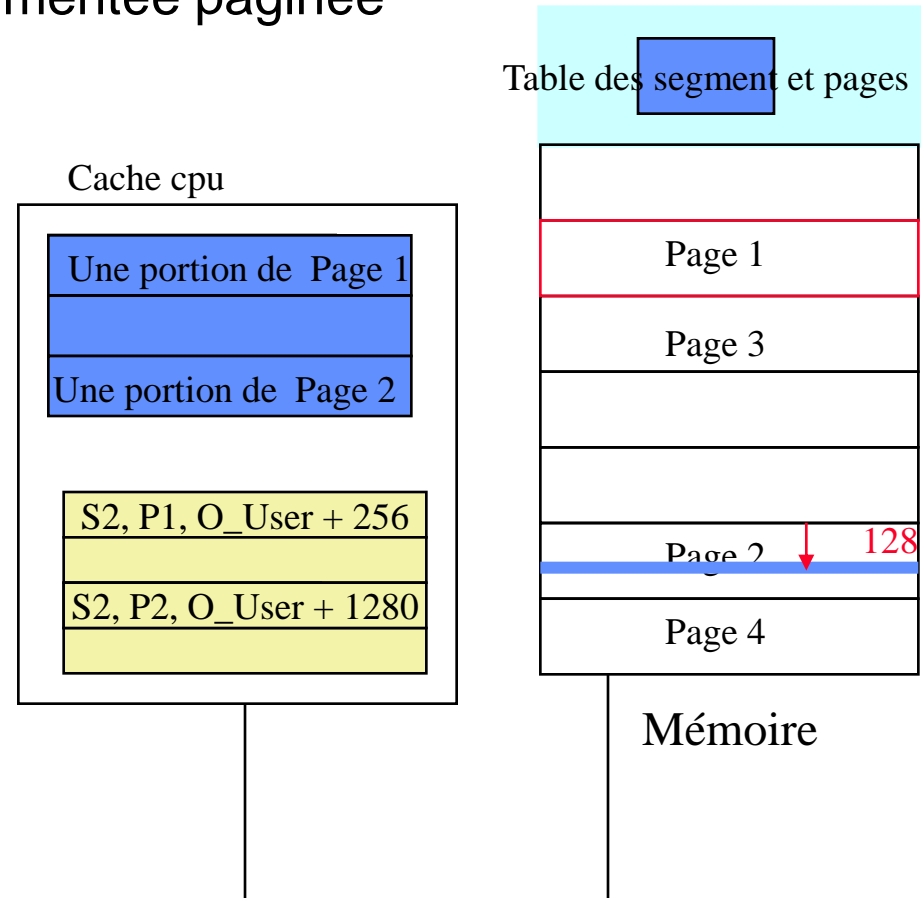


# La mémoire segmentée paginée



# La mémoire segmentée paginée

- Conversion d'une adresse logique en adresse physique : 3 accès mémoire
- **cache du processeur** : il contient les instructions et données les plus récemment accédées en mémoire centrale
- **Cache de la MMU** : contient les associations (n°segment, page du segment, case de la mémoire physique) les plus récemment formés



# Caches du processeur et de la MMU

- L'utilisation de caches associatifs s'appuie sur les principes de localité d'exécution d'un programme
  - **Localité temporelle** : si le processeur accède à l'instant  $t$  à l'adresse  $(s, p, d)$ , la probabilité qu'il demande de nouveau accès à cette adresse  $(s, p, d)$  à l'instant  $t'$  très proche de  $t$  est grand
    - ⇒ on enregistre le mot d'adresse  $(s, p, d)$  dans le cache du processeur
    - ⇒ On mémorise l'association  $(s, p, c)$  dans le cache de la MMU avec  $c$  la case contenant la page  $p$  du segment  $s$
  - **Localité spatiale** : si le processeur accède à l'instant  $t$  à l'adresse  $(s, p, d)$ , la probabilité qu'il demande accès à une adresse voisine dans cette même page  $(s, p, d')$  à l'instant  $t'$  très proche de  $t$  est grand
    - ⇒ on enregistre le mot d'adresse  $(s, p, d)$  et des voisins dans le cache du processeur
    - ⇒ On mémorise l'association  $(s, p, c)$  dans le cache de la MMU avec  $c$  la case contenant la page  $p$  du segment  $s$



# Implémentation de la table des pages

```
Pour i = 1 à 10
Faire
A(i) = 2;
B(i) = 5;
i = i + 1;
Fait
```

```
0 R1 ← 1
1 R2 ← 10
2 comp R1, R2
3 si R1 > R2 fin
4 A(R1) ← 2
5 B(R1) ← 5
6 R1 ← R1 + 1
7 aller à 2
```

Page 1 code

Page de 10 mots

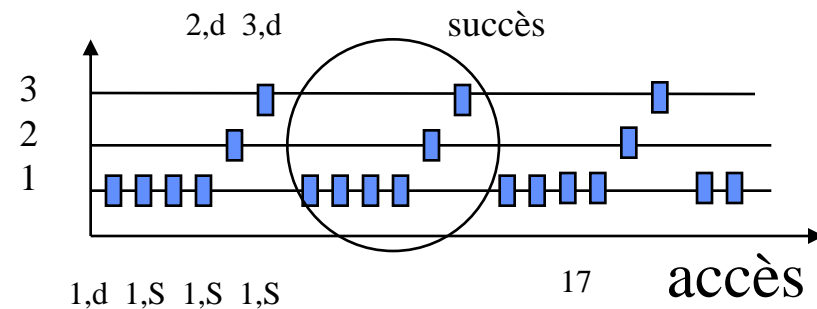


Page 2 vecteur A

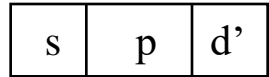


Page 3 vecteur B

Références aux pages : 11(112311)<sup>10</sup>



# La mémoire segmentée paginée



Le mot d'adresse (s, p, d')  
est-il dans le cache CPU ?

Oui (succès) 0

Non (défaut)

L'association (s, p, c)  
est elle dans le cache MMU ?

Oui (succès)

Lire le mot (c,d') en MC

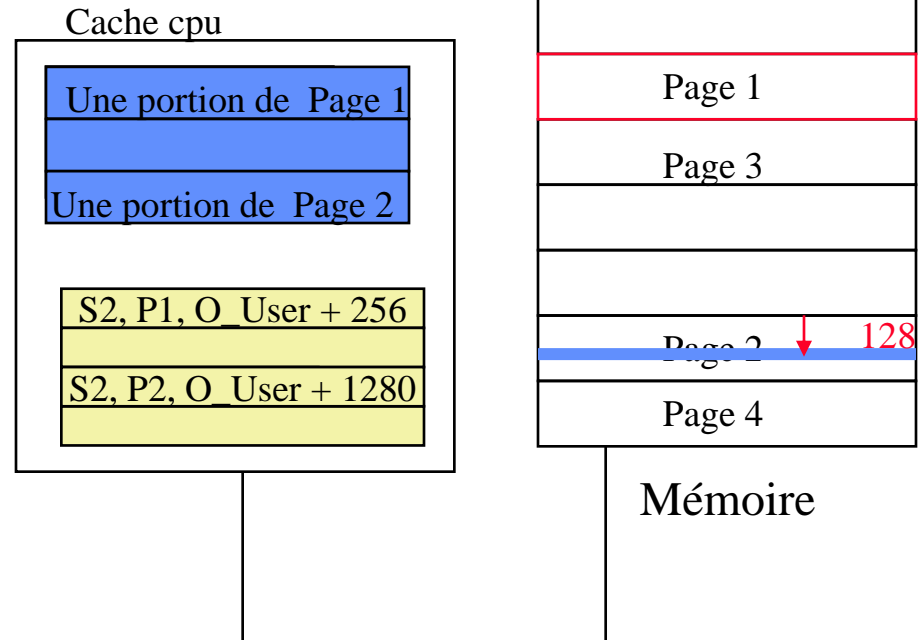
Le copier dans le cache CPU avec ses voisins

1

3

Non (défaut)

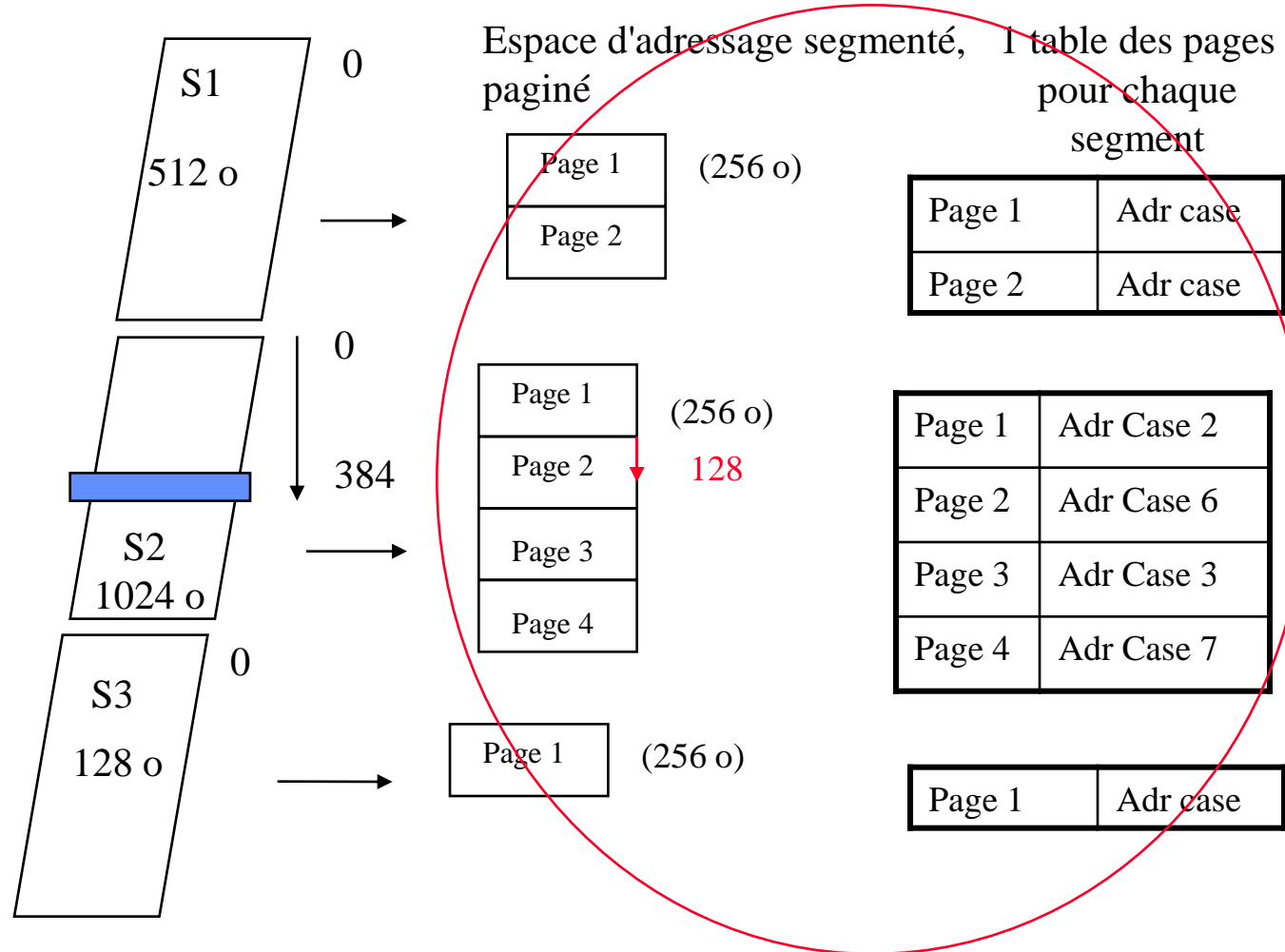
Accéder à la table des segments et à la table des pages du segment  
Enregistrer l'association formée (s, p, c) dans le cache de la MMU



# Gestion de la mémoire centrale

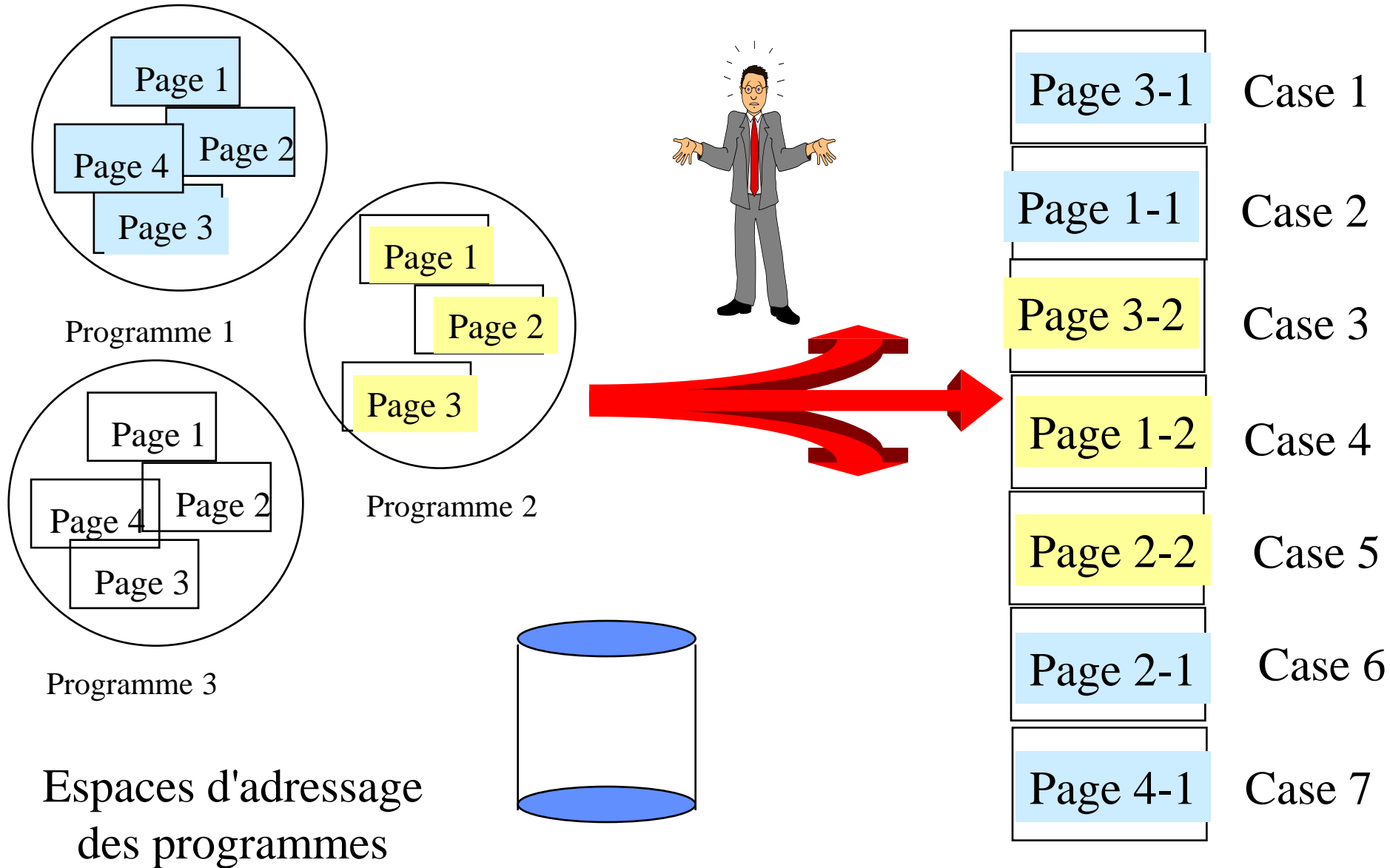
➤ **Mémoire virtuelle**

- On suppose des espaces d'adressages segmentés paginés que l'on représente comme un ensemble de pages



# Mémoire virtuelle

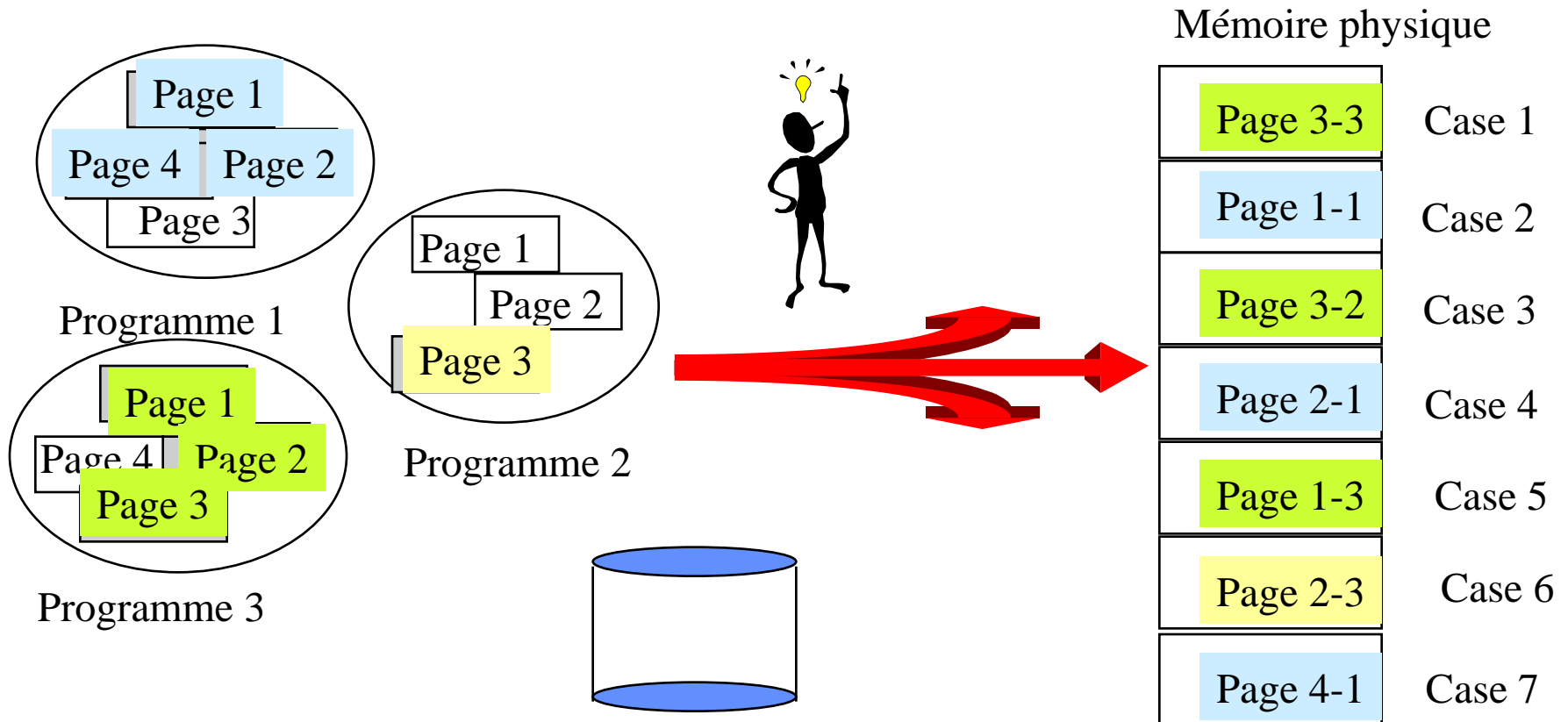
## Mémoire physique



# Mémoire virtuelle

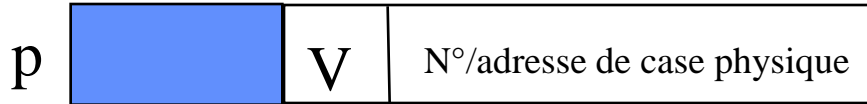
- La capacité de la mémoire centrale est trop petite pour charger l'ensemble des pages des programmes utilisateurs.

☞ Ne charger que les pages utiles à un instant (principes de localité).



# Bit de validation

- Ne charger que les pages utiles à un instant
  - il faut pouvoir tester la présence d'une page en mémoire centrale
- Bit validation à vrai si la page p est présente en mémoire centrale



V	2
V	4
I	-
V	7

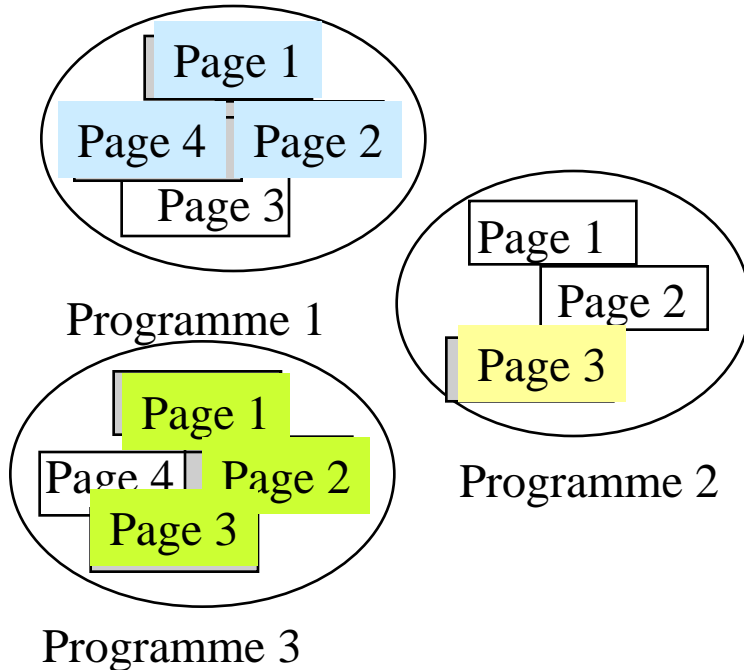
Processus 1

I	-
I	-
V	3

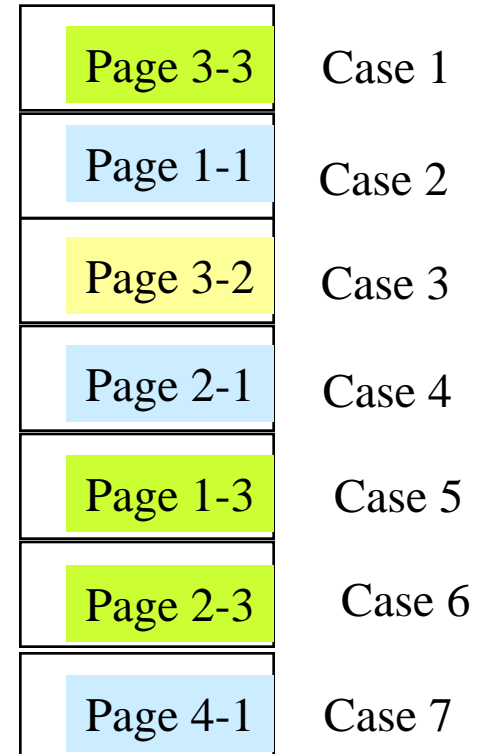
Processus 2

V	5
V	6
V	1
I	-

Processus 3



## Mémoire physique



# Bit de validation et défaut de page

V	2
V	4
I	-
V	7

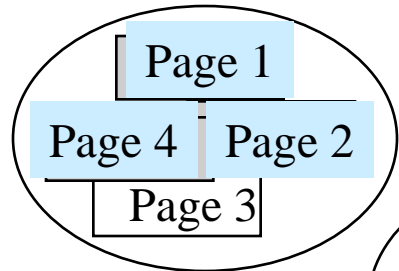
Processus 1

I	-
I	-
V	3

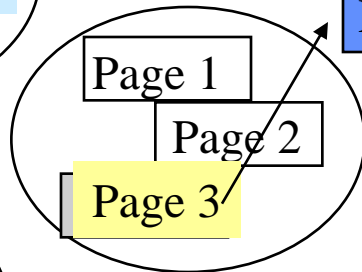
Processus 2

V	5
V	6
V	1
I	-

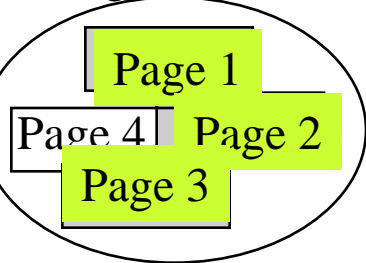
Processus 3



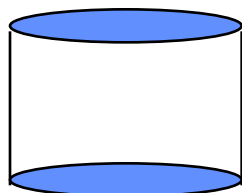
Programme 1



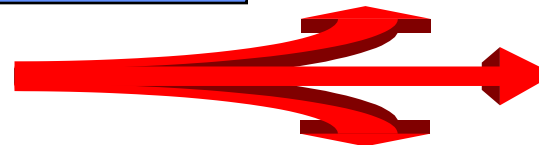
Programme 2



Programme 3



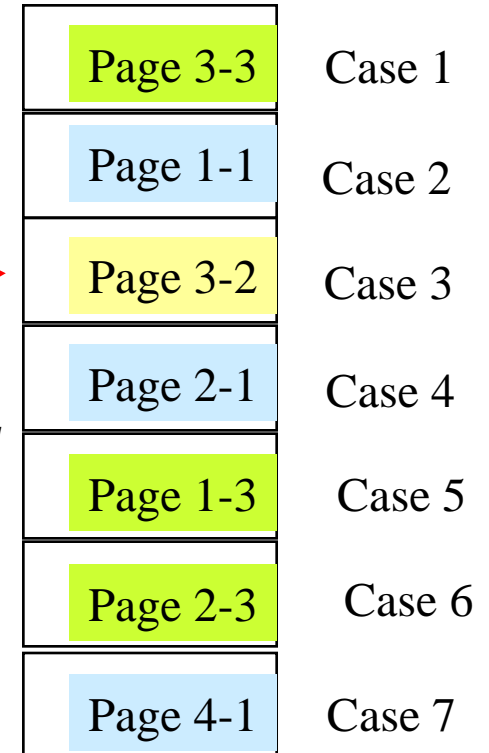
Load D R (P2, d)



Processus 2 : accès à la page 2

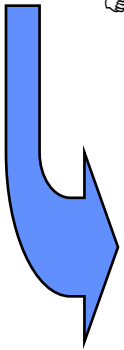
**DEFAUT DE PAGE**

Mémoire physique



# Bit de validation et défaut de page

- Ne charger que les pages utiles à un instant
  - ☞ il faut pouvoir tester la présence d'une page en mémoire centrale :
    - ☞ rôle du bit de validation
  - ☞ si un processus cherche à accéder à une page non présente en mémoire centrale, il se produit un **déroutement de défaut de page**



le système d'exploitation lance une entrée/sortie disque pour charger la page en mémoire dans une case libre.

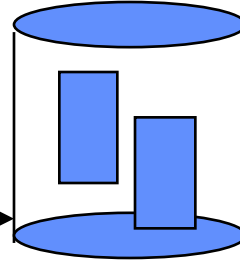
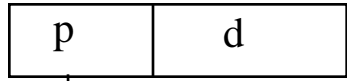
L'adresse de la page sur disque est stockée dans la table des pages.

Le chargement des pages **s'effectue à la demande**.



# Défaut de page

Adresse logique



1. Déroulement  
E/S disque

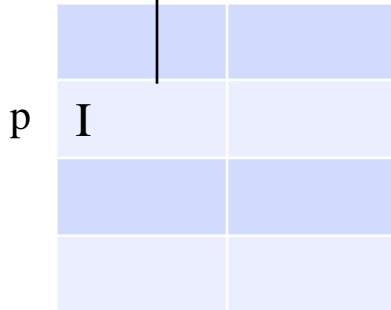
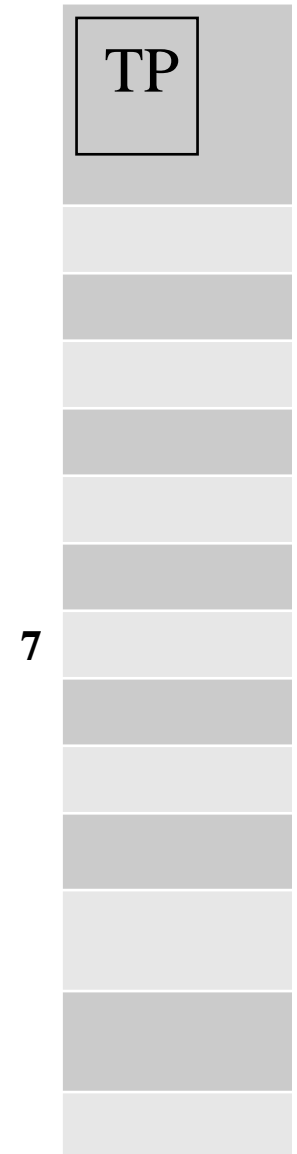


Table des pages

occupée



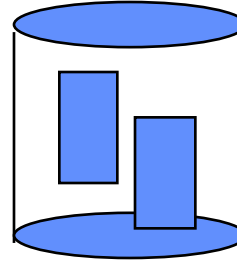
Mémoire centrale

# Défaut de page

Adresse logique



4. Reprise instruction



2 Chargement de la page

7

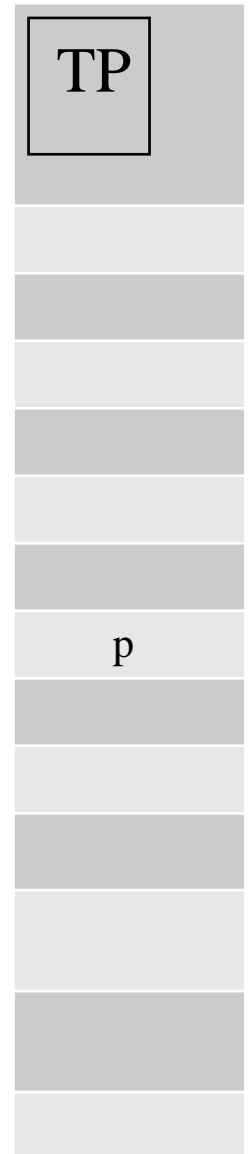
p

V	6	

Table des pages

3. Mise à jour table des pages

occupée

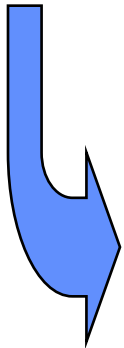


Mémoire centrale

# Chargement de page

- Lors d'un défaut de page, la page manquante est chargée dans une case libre

☞ la totalité des cases de la mémoire centrale peuvent être occupées



- le système d'exploitation utilise un algorithme pour choisir une case à libérer

L'optimal est de retirer une page devenue inutile

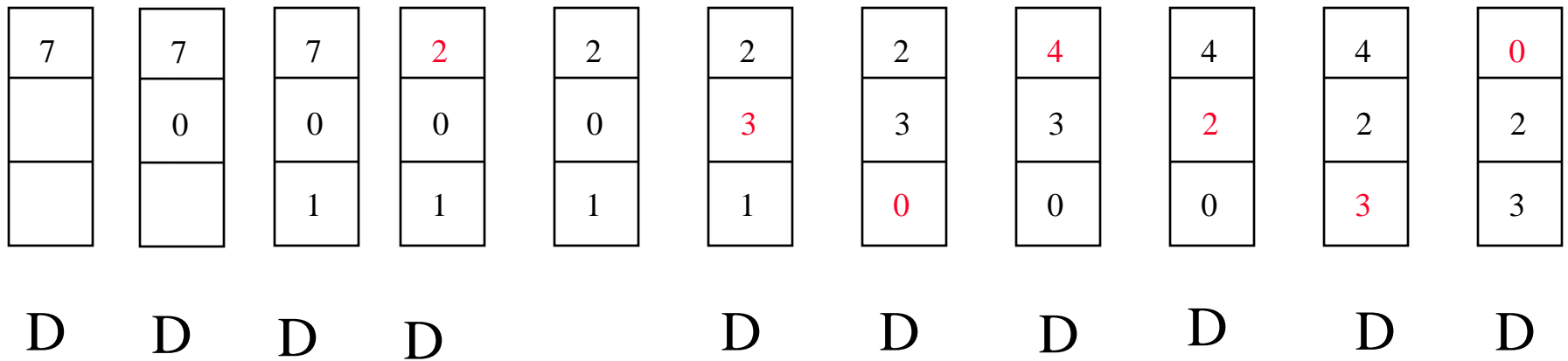
- Au hasard
- FIFO (First In, First out)
- LRU (Least Recently Used) : moins récemment utilisée

# Algorithmes de remplacement de page

- FIFO : la page la plus anciennement chargée est la page remplacée

Chaine de référence

7      0      1      2      0      3      0      4      2      3      0



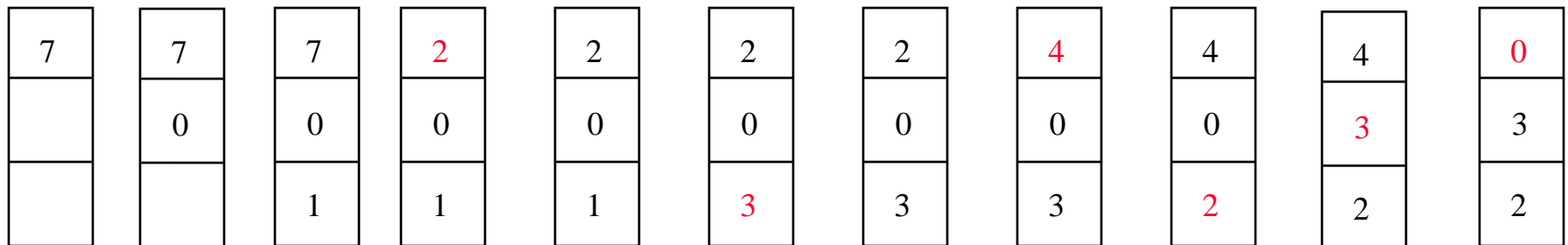
➤ Facile mais peu pertinent

# Algorithmes de remplacement de page

- LRU : la page la moins récemment accédée est la page remplacée

Chaine de référence

7      0      1      2      0      3      0      4      2      3      0



D      D      D      D                      D                      D      D      D      D



7      0      1

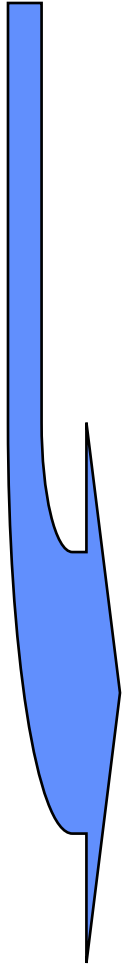


7      0      1      2      0

➤ Pertinent, mais couteux

# Algorithme de remplacement de page

	A	M	V	N° de case physique
--	---	---	---	---------------------

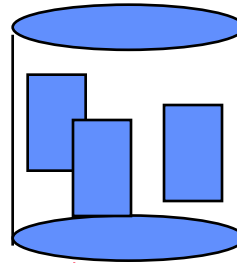


- Bit modification à vrai si la page a été modifiée en mémoire centrale (page à sauvegarder si modifiée en MC)

- Champ Accès :  
FIFO : date de chargement  
LRU : date de dernier accès

# Libération de page

Adresse logique

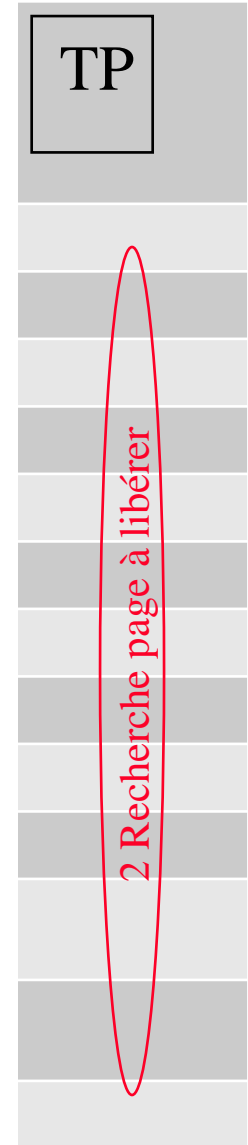


3 Ecriture de la page si  $q.M = \text{vrai}$

p	I		
q	V	M	10

Table des pages

Mémoire



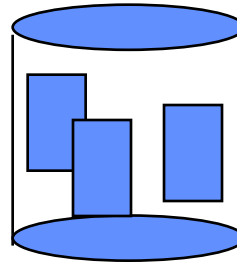
Mémoire centrale

# Libération de page

Adresse logique

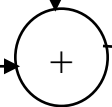


6. Reprise instruction



Registre adresse  
Table des pages

adresse table



p	V	10
q	I	

Table des pages

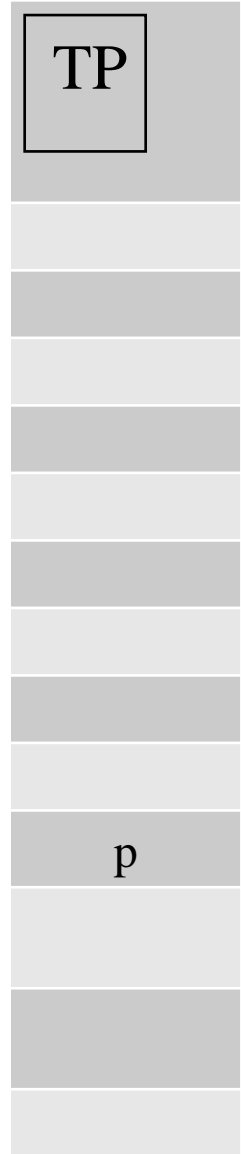
5 Mise à jour table des pages

4 Chargement de la page p

6

10

Mémoire



Mémoire centrale



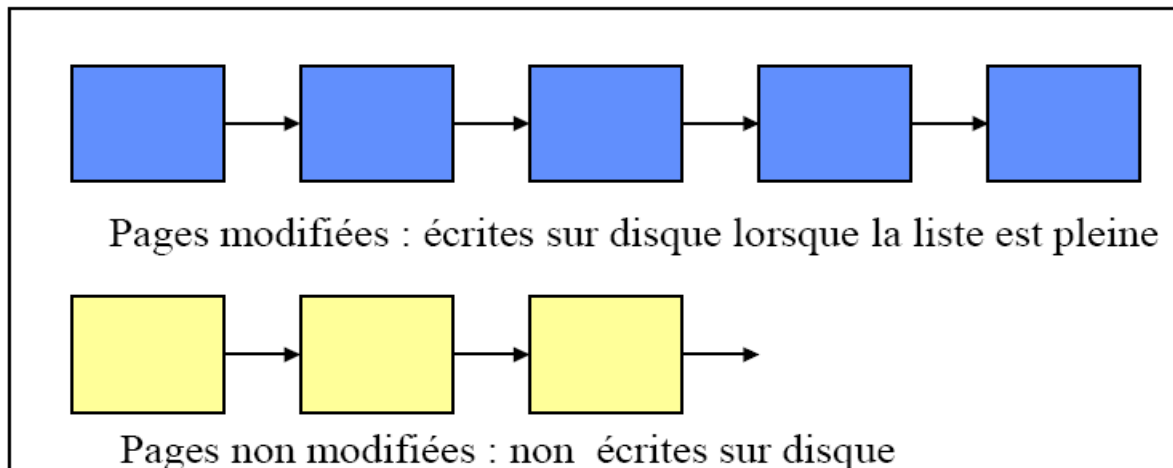
# Performances

- $p$ ,  $0 \leq p \leq 1$ , probabilité de défaut de pages
- $T_a$ , temps d'accès à un mot sans défaut de pages (100 ns)
- $T_f$ , temps d'accès à un mot avec défaut de pages (25 ms)
- Temps d'accès effectif =  $(1 - p) * T_a + p * T_f$   
$$\frac{(1 - p) * 100 + p * 25\,000\,000}{100 + 24\,999\,900 p}$$

Dégradation des performances < 10 %  $\rightarrow p < 0,000\,000\,4$   
1 / 2 500 000 accès provoque un défaut

# Performances

- Un disque de pagination avec un taux d'accès rapide
- Un dispositif de pré-pagination : lors d'un défaut de pages, chargement de la page manquante et de pages voisines
- Une cache des pages



Page libérée : insérée dans une des listes

Défaut de pages : la page est recherchée dans le cache avant le lancement de l'entrée-sortie

# Gestion de la mémoire centrale

➤ Exemple de Linux

# Le répertoire /proc

- **/proc** est un système de fichier utilisé par le noyau pour envoyer des informations aux différents processus.
- On y trouve :
  - des fichiers contenant des informations générales sur le système. Comme :
    - `uptime` donnant le temps de fonctionnement du système.
    - `stat` donnant diverses statistiques sur l'utilisation des ressources du système (CPU, mémoire... ).
    - `meminfo` donnant un récapitulatif de l'utilisation de la mémoire.
    - `cpuinfo` donnant une description des CPU du système.
  - un répertoire par processus actif. Chacun porte le numéro du processus (PID). Il est constitué des fichiers suivants :
    - `cmdline` contient la ligne de commande qui a créé le processus.
    - `status` contient des informations sur l'état du processus (en attente, en exécution, propriétaire... ).
    - `exe` est un lien vers le fichier exécutable utilisé par le processus.
    - `maps` contient les informations relatives aux régions du processus

# Le répertoire /proc

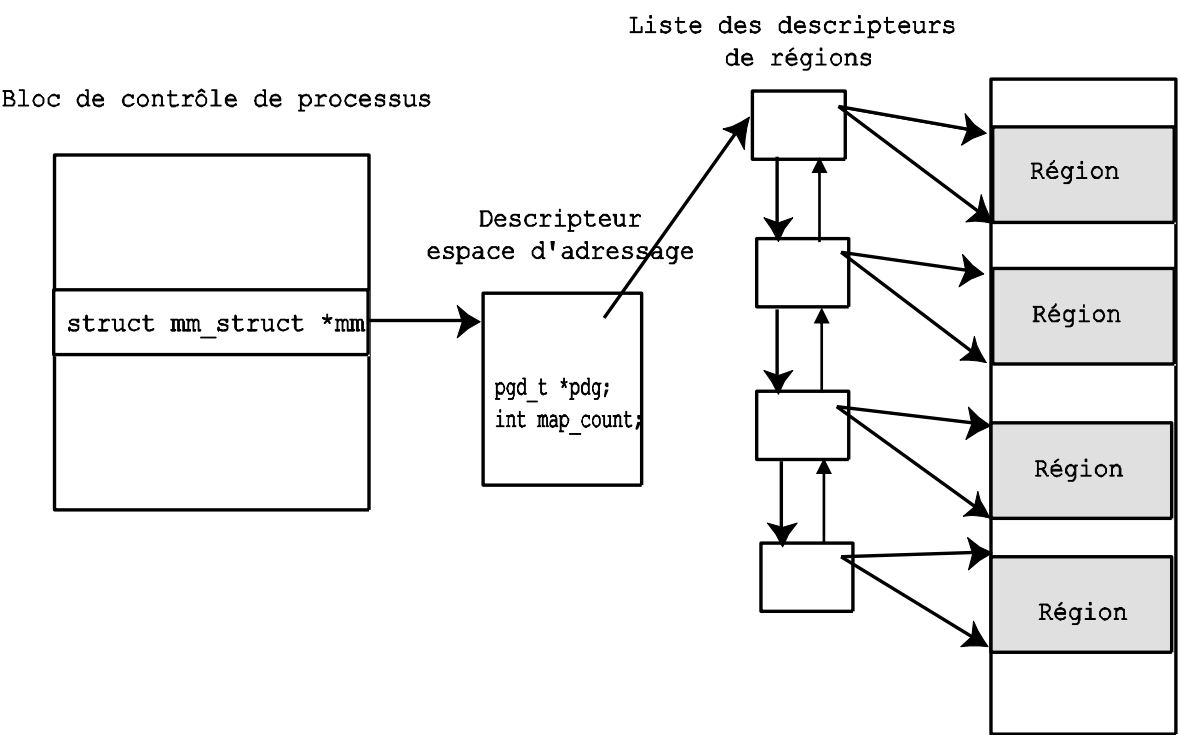
```
linux-9bxb:/proc/3431 # cd /proc
linux-9bxb:/proc # ls
1      14      19      22      26      297     3040    3203    3260    3338    337     3470    38      402     49      534     58      66      8      cmdline      dri
10     15      1943    23      27      3       31      3204    3261    3341    3379    3488    3873    41      5       54      59      67      9      config.gz    driver
11     16      2       24      28      3017    3133    3242    3262    3356    34      3495    3881    43      50      540     6       681     966    consoles     execdomains
110    17      20      243     282     3028    3152    3243    3263    3357    3428    35      39      44      51      541     60      69      ACPI         cpuinfo      fb
12     18      21      244     283     3029    3162    3244    33     336     343     3508    4       45      52      542     61      7       asound       crypto       filesystems
1289   186     212     25      2865    3033    3176    3246    3315    3360    3431    37      40      46      53      56      634     71      buddyinfo   devices      fs
13     1860    213     252     288     3037    32     3249    3327    3362    3439    3704    400     47      530     57      652     72      bus          diskstats    interrupts
136    1861    214     253     29      3038    3202    3255    3334    3365    3446    3706    401     48      533     572     655     73      cgroups     dma           iomem

linux-9bxb:/proc # cd 3431
linux-9bxb:/proc/3431 # more cmdline
/bin/bash
linux-9bxb:/proc/3431 # more maps
00400000-0049c000 r-xp 00000000 08:01 262151      /bin/bash
0069b000-0069c000 r--p 0009b000 08:01 262151      /bin/bash
0069c000-006a0000 rw-p 0009c000 08:01 262151      /bin/bash
006a0000-006ab000 rw-p 00000000 00:00 0
01643000-0178c000 rw-p 00000000 00:00 0
7fe8fad9e000-7fe8faf43000 r-xp 00000000 08:01 1441977    /lib64/libc-2.18.so
7fe8faf43000-7fe8fb143000 ---p 001a5000 08:01 1441977    /lib64/libc-2.18.so
7fe8fb143000-7fe8fb147000 r--p 001a5000 08:01 1441977    /lib64/libc-2.18.so
7fe8fb147000-7fe8fb149000 rw-p 001a9000 08:01 1441977    /lib64/libc-2.18.so
7fe8fb149000-7fe8fb14d000 rw-p 00000000 00:00 0
7fe8fb14d000-7fe8fb150000 r-xp 00000000 08:01 1441972    /lib64/libdl-2.18.so
7fe8fb150000-7fe8fb34f000 ---p 00003000 08:01 1441972    /lib64/libdl-2.18.so
7fe8fb34f000-7fe8fb350000 r--p 00002000 08:01 1441972    /lib64/libdl-2.18.so
7fe8fb350000-7fe8fb351000 rw-p 00003000 08:01 1441972    /lib64/libdl-2.18.so
7fe8fb351000-7fe8fb37c000 r-xp 00000000 08:01 1441795    /lib64/libtinfo.so.5.9
7fe8fb37c000-7fe8fb57b000 ---p 0002b000 08:01 1441795    /lib64/libtinfo.so.5.9
7fe8fb57b000-7fe8fb57f000 r--p 0002a000 08:01 1441795    /lib64/libtinfo.so.5.9
7fe8fb57f000-7fe8fb584000 rw-p 0002e000 08:01 1441795    /lib64/libtinfo.so.5.9
7fe8fb584000-7fe8fb585000 rw-p 00000000 00:00 0
```

# La mémoire vue par Linux

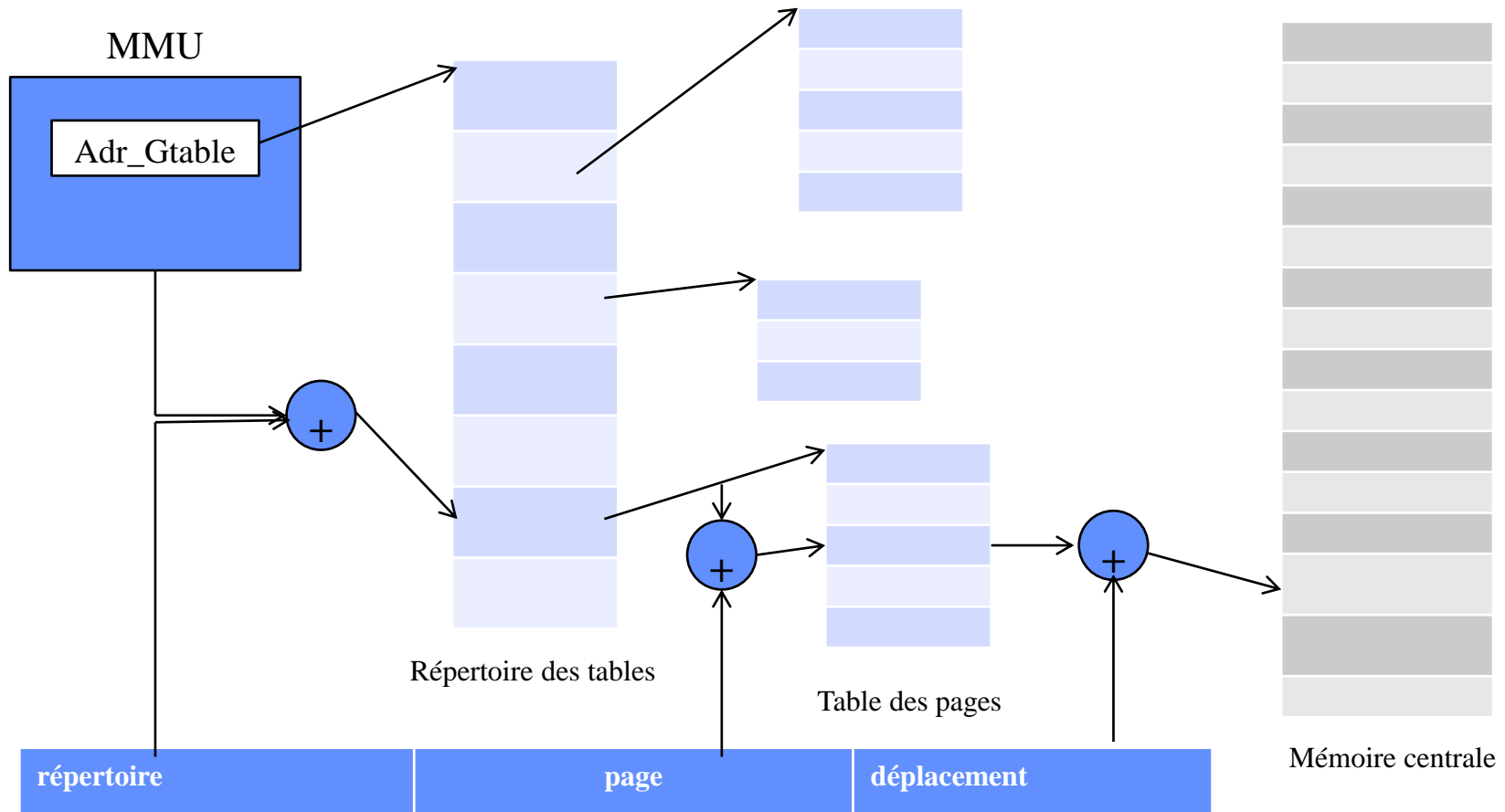
- L'espace d'adressage d'un processus est composé de régions
  - une région de code, une région des variables initialisées, une région des variables non initialisées, une région pour la pile
- Une région est une zone contiguë de l'espace d'adressage traitée comme un objet pouvant être partagé et protégé. Elle est caractérisée par
  - ses adresses de début et de fin
  - les droits d'accès qui lui sont associés
  - l'objet qui lui est associé

```
bbj>cat /proc/self/maps
08048000-0804a000 r-xp 00000000 03:02 7914
0804a000-0804b000 rw-p 00001000 03:02 7914
0804b000-08053000 rwxp 00000000 00:00 0
40000000-40005000 rwxp 00000000 03:02 18336
40005000-40006000 rw-p 00004000 03:02 18336
40006000-40007000 rw-p 00000000 00:00 0
40007000-40009000 r--p 00000000 03:02 18255
40009000-40082000 r-xp 00000000 03:02 18060
40082000-40087000 rw-p 00078000 03:02 18060
40087000-400b9000 rw-p 00000000 00:00 0
bffff000-c0000000 rwxp fffff000 00:00 0
```



# Table des pages et table des cases

- Une région est divisée en pages (PAGE\_SIZE - 4 Ko). La pagination s'effectue à la demande, sur défaut de pages (do\_page\_fault())
- La table des pages d'un processus est sur deux niveaux :



# Table des pages et table des cases

- Une région est divisée en pages (PAGE\_SIZE - 4 Ko). La pagination s'effectue à la demande, sur défaut de pages (do\_page\_fault())
- Une entrée de table des pages contient :

Present	Accessed	Dirty	<i>Read/write</i>		Case occupée ou adresse dans la zone de swap
---------	----------	-------	-------------------	--	---

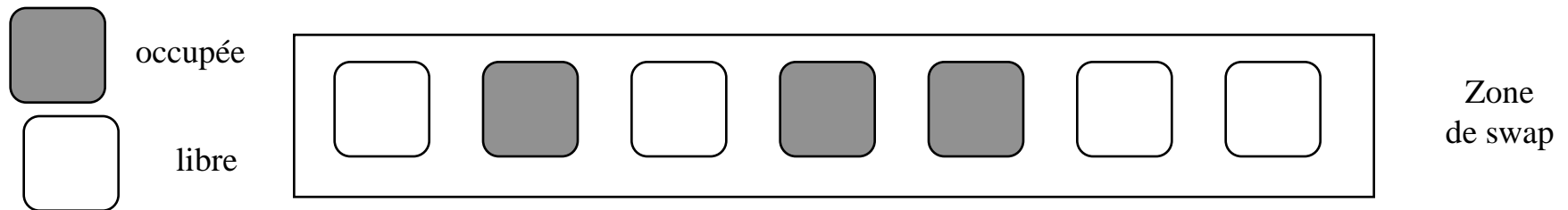
- Chaque case est décrit par un descripteur :

Adresses case libre précédente et suivante
Nombre de processus se partageant la page
État de la page (verrouillée, accédée, ...)
Champ dirty ( page modifiée)
<b>Champ age</b>



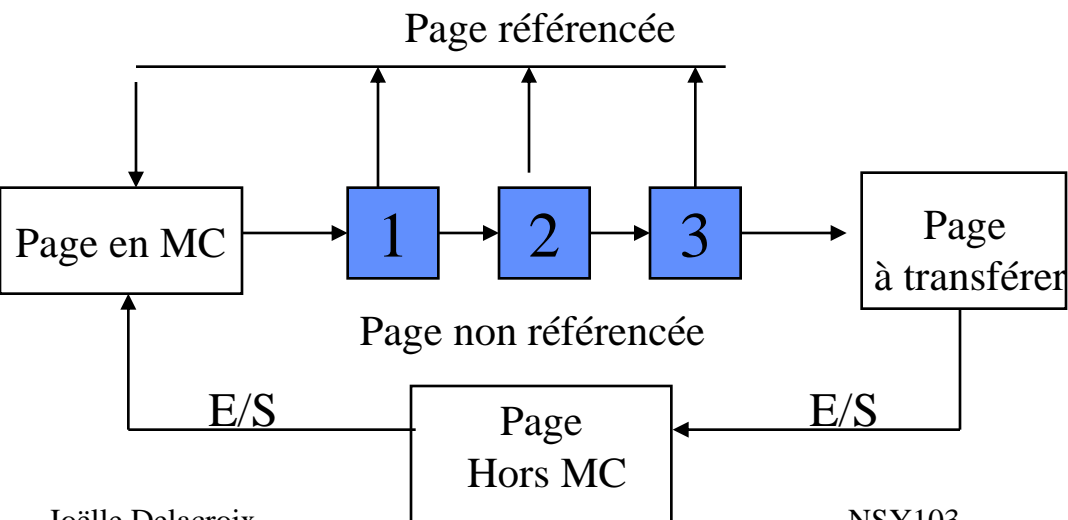
# Swap out des pages

- Les champs Accessed et Age sont utilisés par le thread noyau "Dérobeur de Pages - kswapd" pour choisir des pages victimes : c'est le swap out des pages.
  - le dérobeur de pages est réveillé toutes les 10 secondes
  - le dérobeur de pages libère des pages si le nombre de cases libres est tombé en dessous d'un seuil minimal
  - Le dérobeur de pages sélectionne les processus ayant le plus de pages en mémoire centrale
  - Une page est victime si elle a atteint un age donné (paramètre système) sans être référencée; elle est swappée sur disque
  - Une zone de swap peut être une partition de disque organisée en une séquence d'emplacements de pages
  - Un cache des pages est utilisé



# Swap des pages

- Les champs Accessed et Age sont utilisés par le processus "Dérobeur de Pages" pour choisir des victimes.
- A chaque référence par le processus, l'age de la page devient égal à 0 et le bit Accessed est mis à vrai
- A chacun de ses passages, le dérobeur de pages :
  - met à faux le bit Accessed si il est à vrai
  - incrémente l'age de la page
- Une page est victime si
  - Le bit Accessed est faux
  - l'age limite est atteint (ici par exemple 3)



En mémoire	Accessed	Age
Accès P	v	0
Accès D	F	1
Accès D	F	2
Accès P	v	0
Accès D	F	1
Accès P	v	0
Accès D	F	1
Accès D	F	2
<b>Accès D</b>	F	3
<b>Transférée</b>		