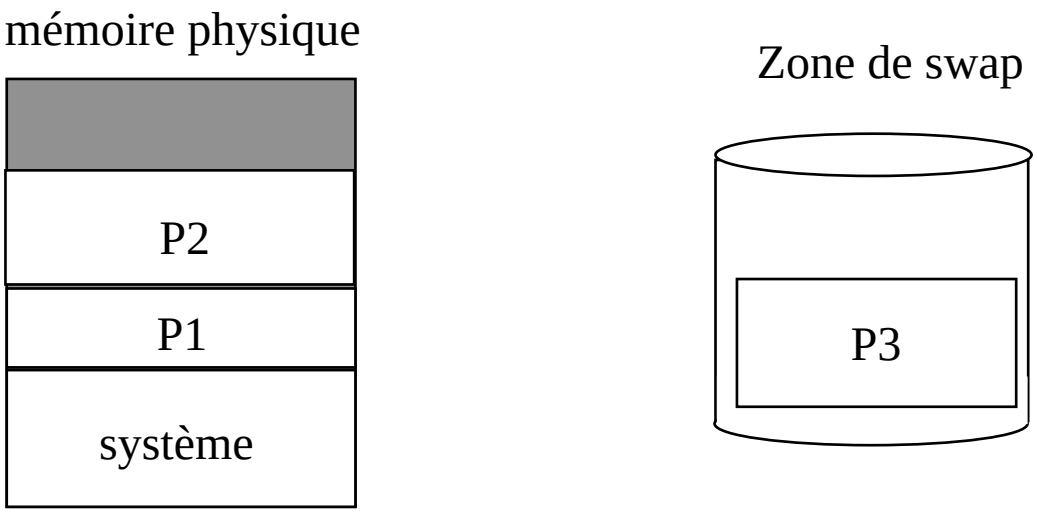
Mémoire virtuelle

1. Notions de base
2. Historique
3. Support Matériel
4. Etude de cas : 4.3BSD
   Pagination, Gestion du swap
5. Les nouveaux système de pagination : 4.4BSD - SVR4

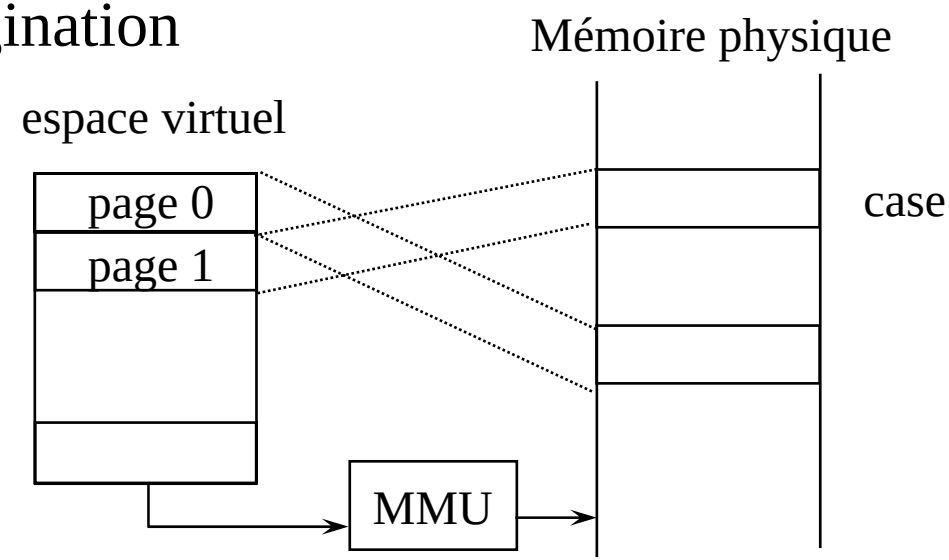
Notions de base

- Le swapping
  - Processus alloués de manière contiguë en mémoire physique
  - chargés /déchargés en entier
  - séparation du code pour optimiser la mémoire (segmentation)



Notions de base (2)

- La pagination



Le processus est partiellement en mémoire

  - les pages sont chargées **à la demande**

- Le remplacement de page
  - Evincer une page lorsqu'il n'y a plus assez de cases libres
- Notion d'espace de travail
  - ensemble des pages les plus utilisées par un processus (localité)

Historique

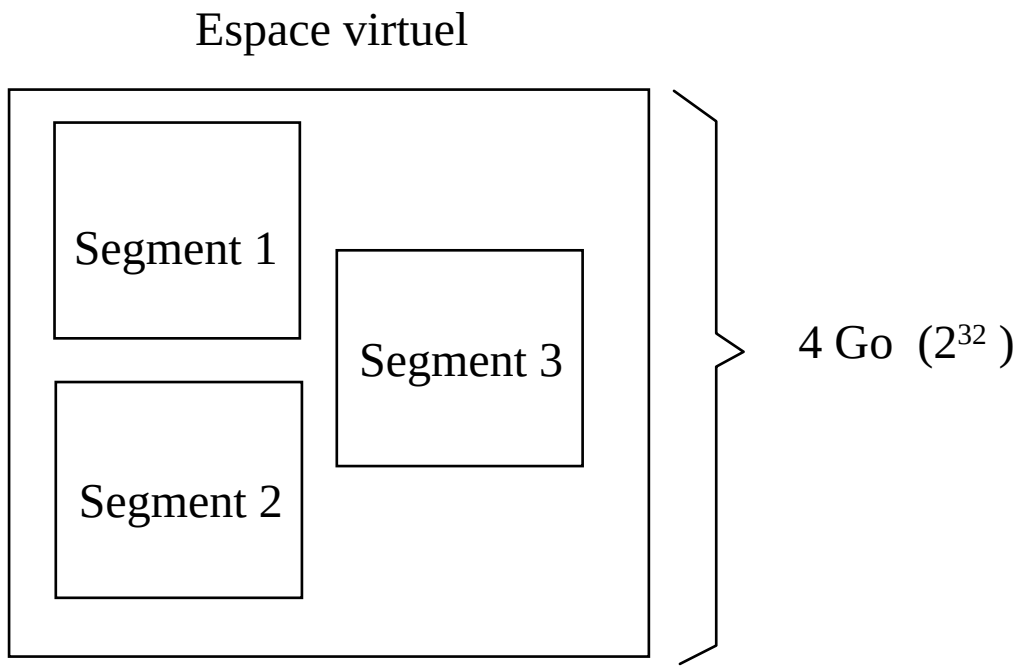
- Apparition tardive de la mémoire paginée dans Unix
- Jusqu'en 1978 utilisation exclusive du swap de processus

  PDP-11 16 bits
- 1979 introduction de la pagination
  3BSD sur vax-11/780 - 32 bits
  => 4 Go d'espace d'adressage
- Milieu de années 80 toutes les versions d'Unix incluent la mémoire virtuelle

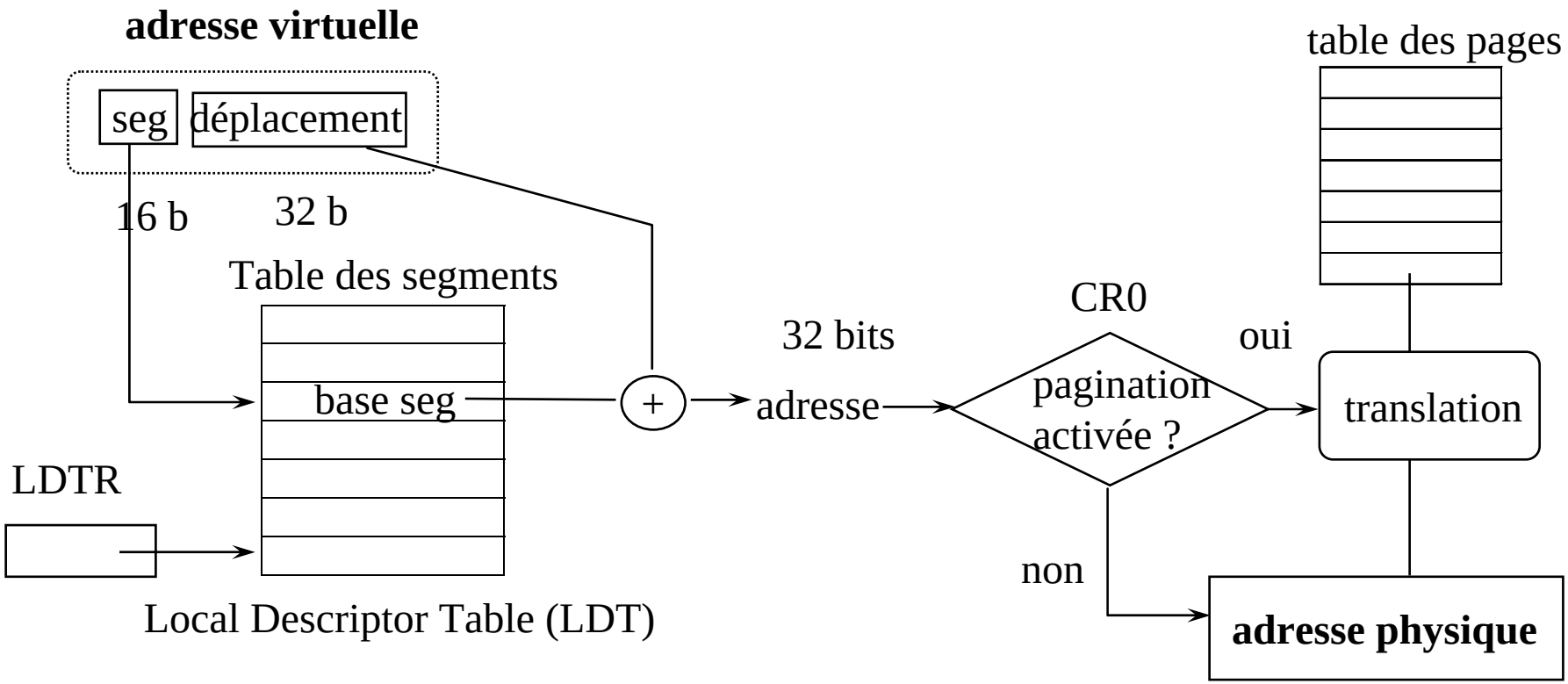
- Dans Unix, segmentation cachée à l'utilisateur, utilisée uniquement pour le partage et la protection

Support matériel :
Exemple Pentium

- A partir de Intel 80386 adresses sur 32 bits
  => 4 Go d'espace d'adressage
- Mémoire segmentée paginée



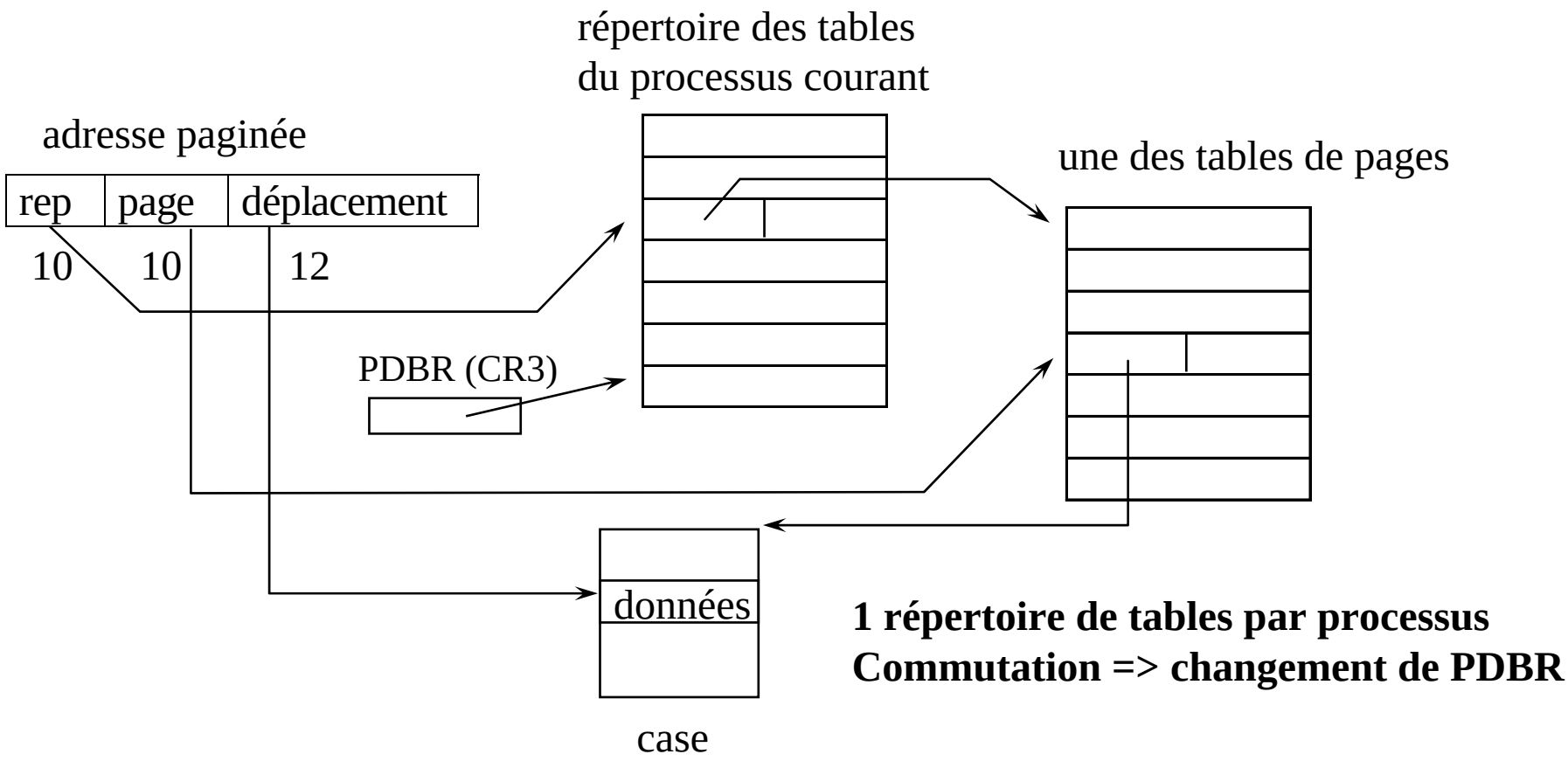
Architecture



- **1 table des segments (LDT) par processus**
- **1 table globale (Global Descripteur Table) = table des segments du système**
- 1 segment particulier : task state segment (TSS) pour sauvegarder les registres lors des commutations

Pagination multi-niveau

- Adressage 32 bits => impossible de maintenir table des pages du processus courant entièrement en mémoire (4 Mo par table !)



Format table des pages

D	Dirty bit (Modification)
A	Accessed bit (Référence)
U	User bit (0: mode utilisateur, 1: mode système)
W	Write bit (0: lecture, 1: écriture)
P	Present bit

Intel prévoit 4 niveaux de protection : Unix en utilise que 2 (util. / syst.)
En mode u les adresses hautes ne sont pas accessibles

Cache d'adresse : la TLB

- Problème de pagination multi-niveaux : accès aux tables
  - => temps d'accès fois 3 (2 niveaux - Intel x86),
    fois 4 (3 niveaux - Sparc),
    fois 5 (4 niveaux - Motorola 680x0)
- Effectuer la traduction uniquement au premier accès

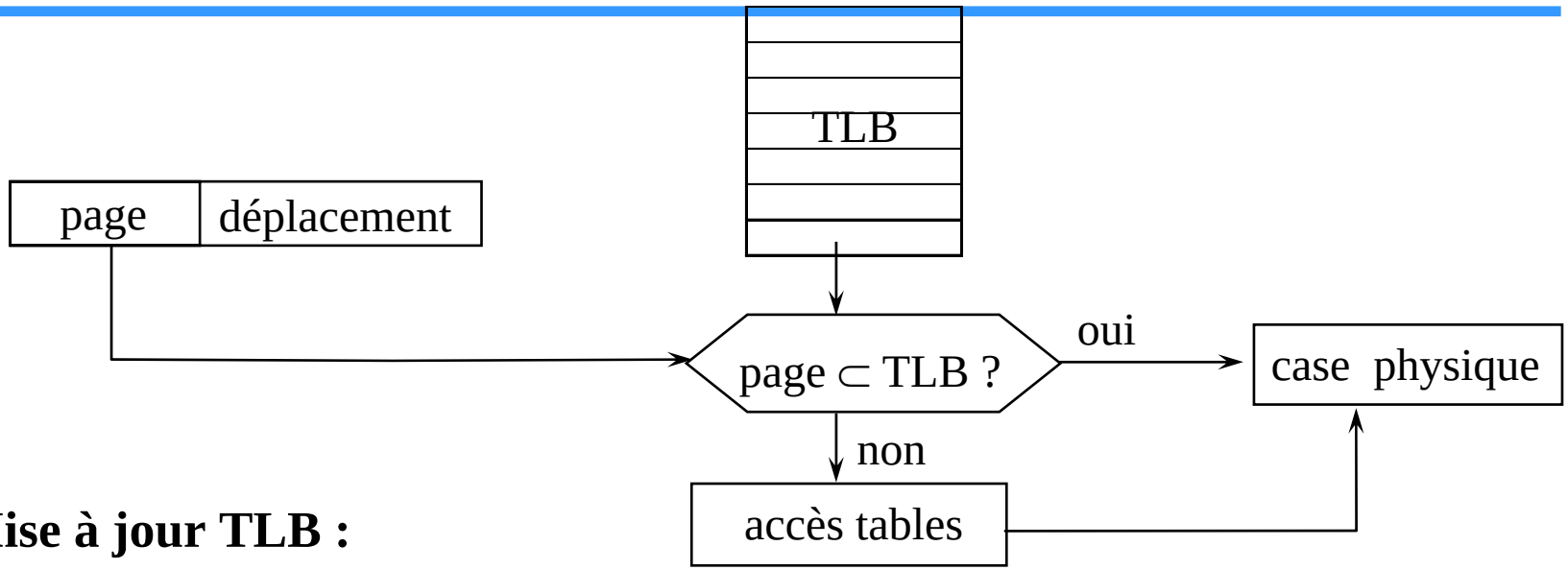
Mémoire associative : Translation Lookaside Buffer

Page	case

= cache des adresses

- TLB nombre d'entrées limité

Gestion de la TLB



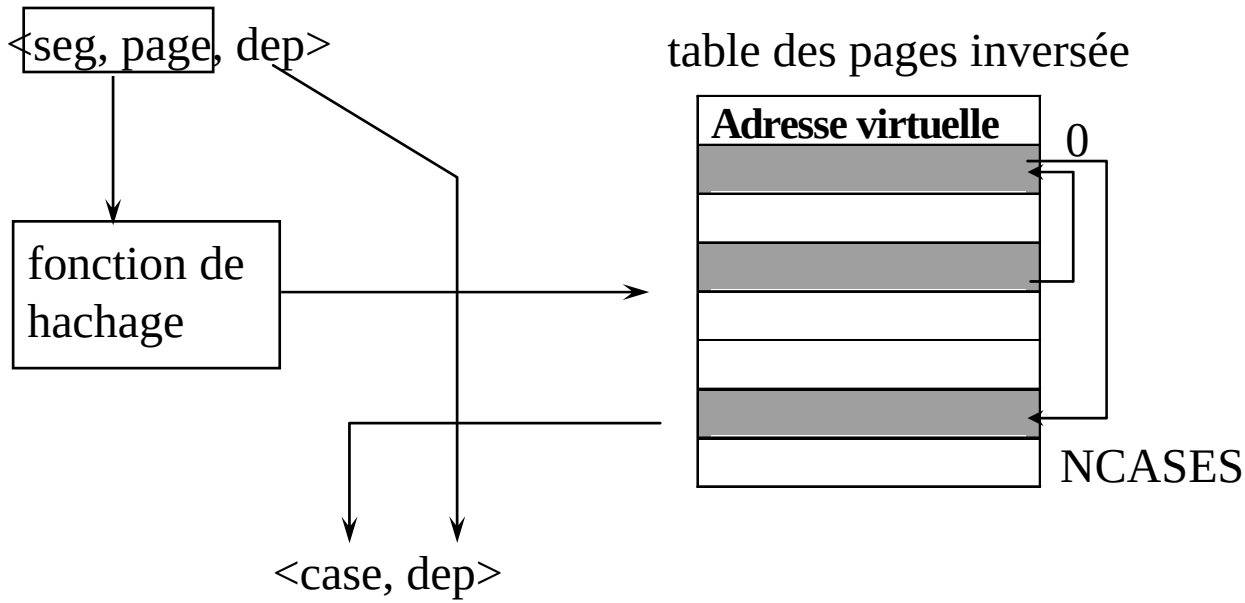
Mise à jour TLB :

- Chargement d'une nouvelle page pour le processus courant
- Commutation => invalidation de **toute** la TLB
  (automatique Intel x86 avec mise à jour PDBR)
- Déchargement page => invalidation entrée TLB
- Recouvrement (exec)

Autre approche : RS/6000

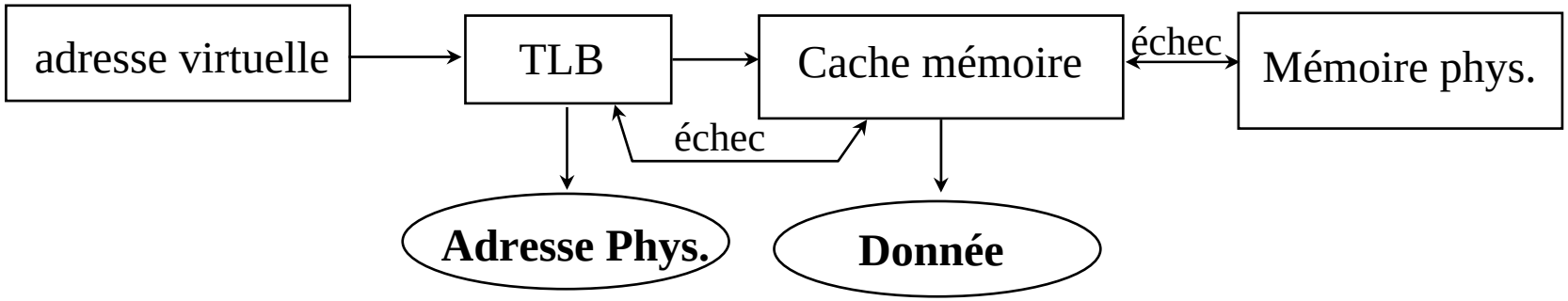
- Architecture RISC base pour AIX
- Utilisation d'une table des pages inversée = 1 entrée par **case** => taille réduite (page 4Ko, 32Mo de RAM => 128 Ko)
  1 seule table globale

adresse virtuelle du processus courant

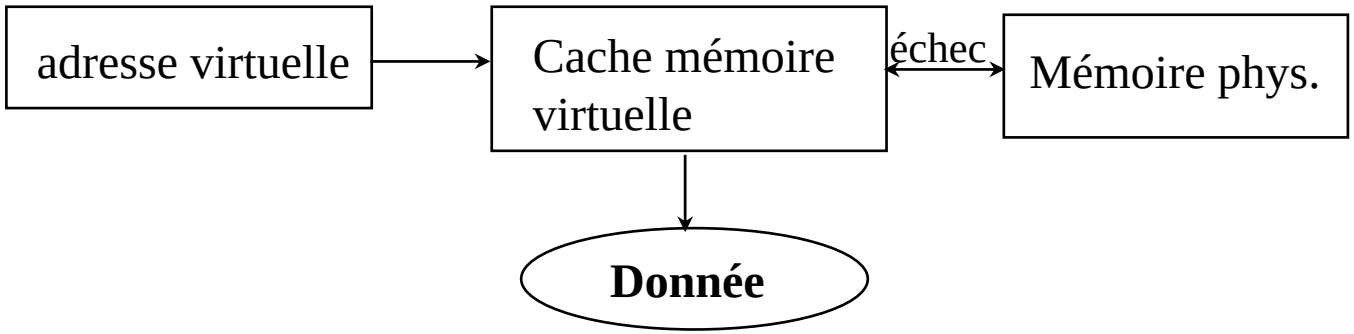


Architecture récente : Cache virtuel

- Architecture «classique» = 2 niveaux de cache mémoire



- Architecture à cache virtuel



- Avantage : 1 seul niveau + pas de «flush» à la commutation

Les processus en mémoire

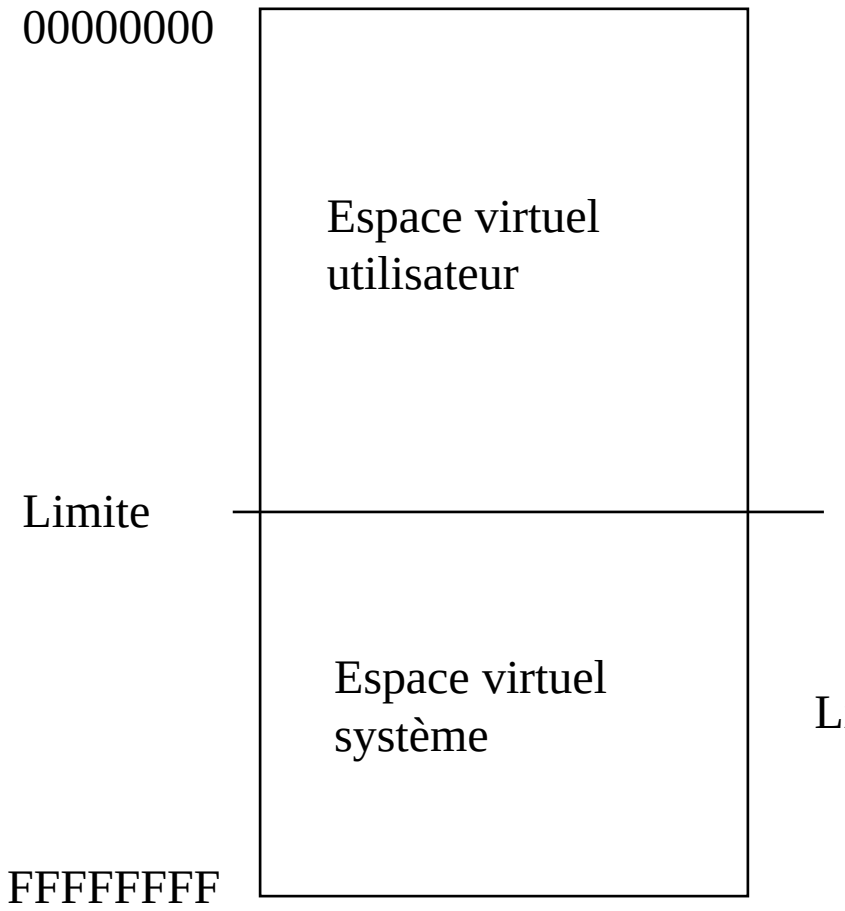
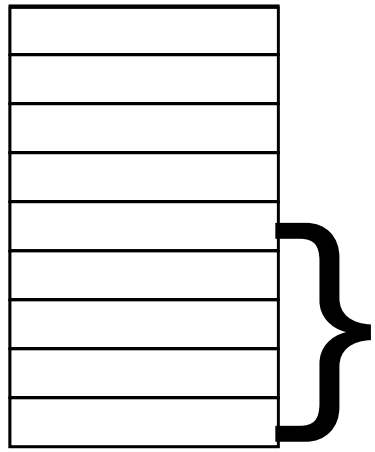


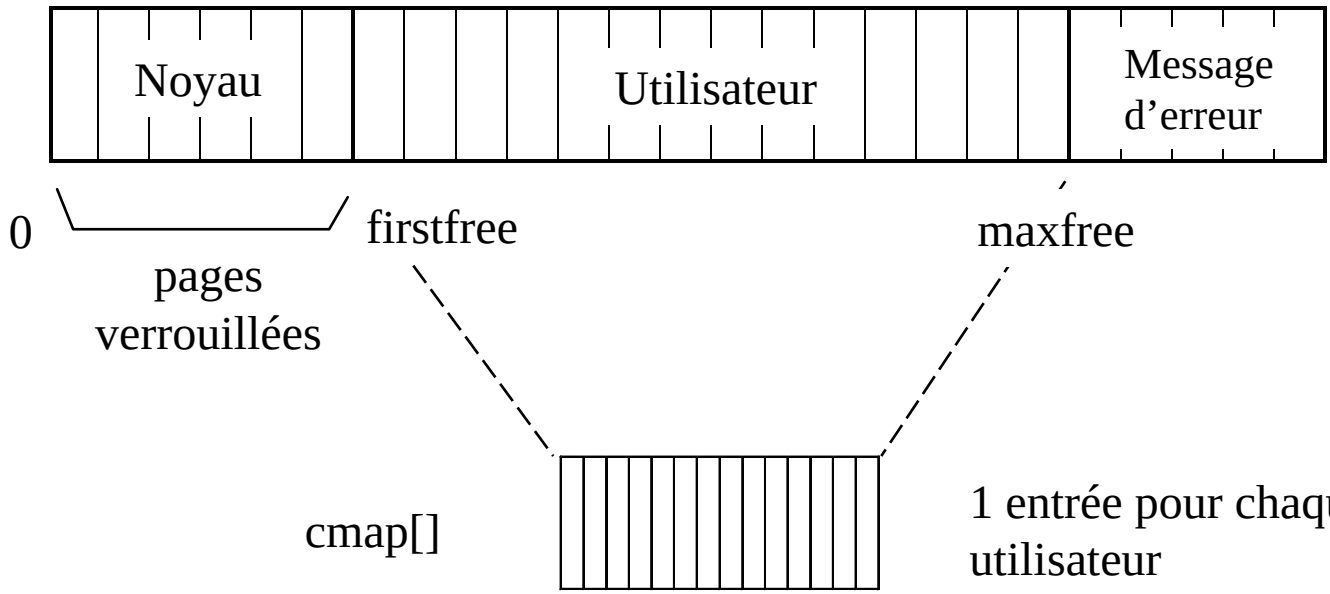
Table des pages



Limite = 80000000 NT (2 Go systeme)
C0000000 Linux – NT server (1 Go)
FFF00000 4.4 BSD (1 Mo)

Etude de cas : 4.3BSD

Représentation de la mémoire physique

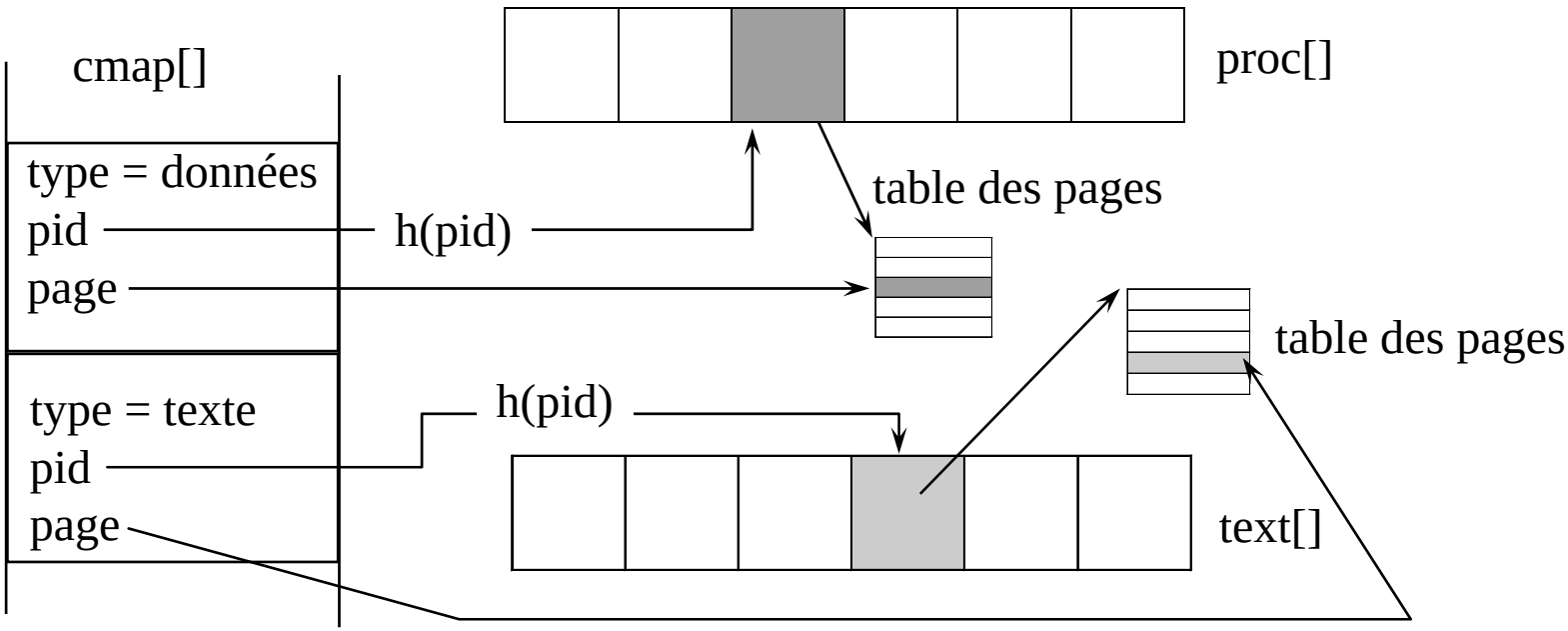


1 entrée pour chaque case utilisateur

(1 entrée par groupe de pages) => taille plus réduite

Structure de contrôle

- La structure cmap:
  - Noms : ID processus, Numéro de page, type (pile, données, texte)
  - Liens sur la freelist (listes des cases libres)
  - Synchronisation : verrous (pendant les chargements/déchargements)
  - Informations utiles pour le cache des pages de code



Etat d’une page

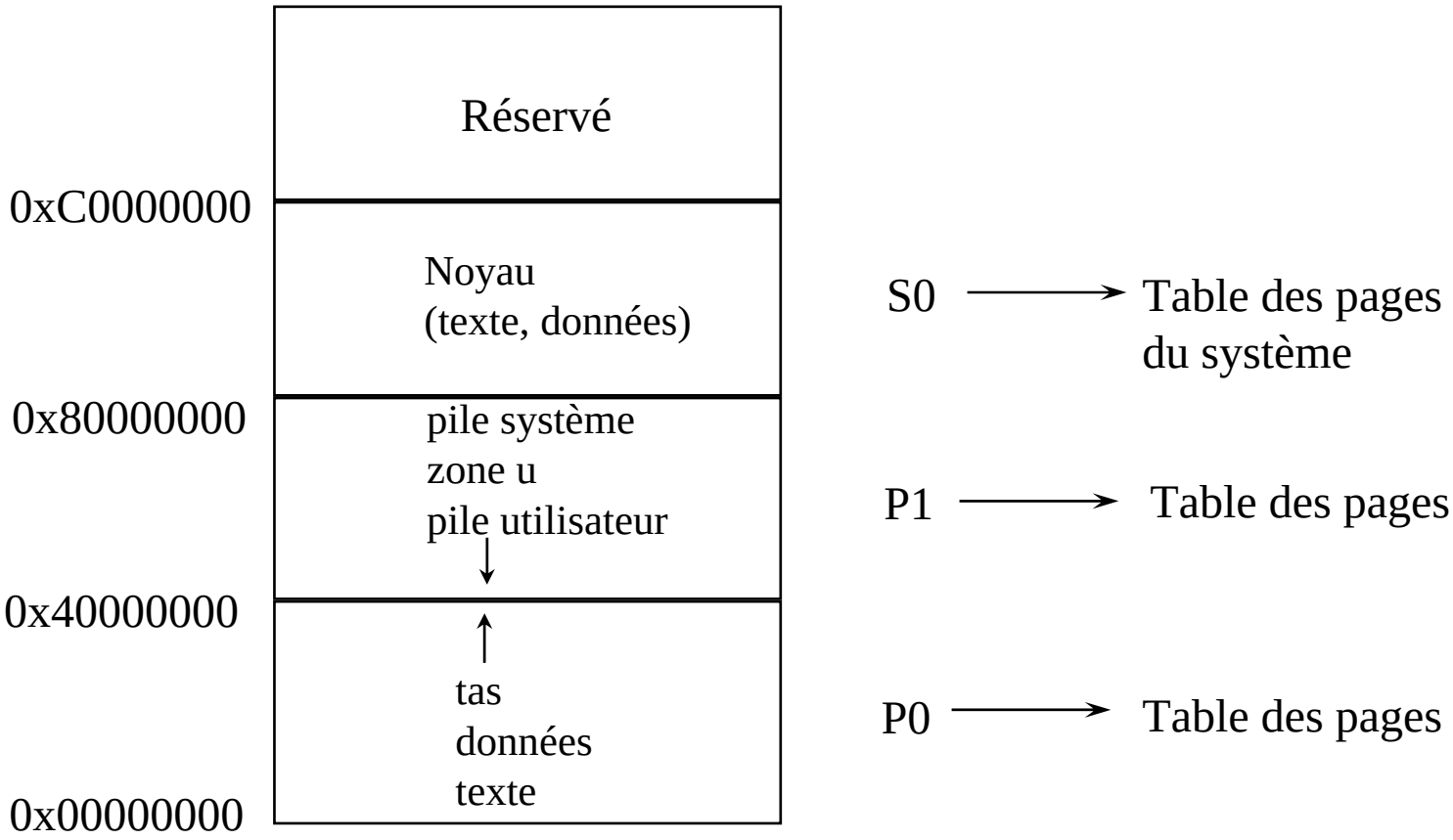
- Résidente : présente en mémoire physique
- Chargée-à-la-demande (Fill-on-demand):
  - Page non encore référencée qui doit être chargée au premier accès
  - 2 types :

    Fill-from-text : lue depuis un exécutable

    Zero-fill: page de pile ou de donnée créée avec des 0
- Déchargée (Outswapped)

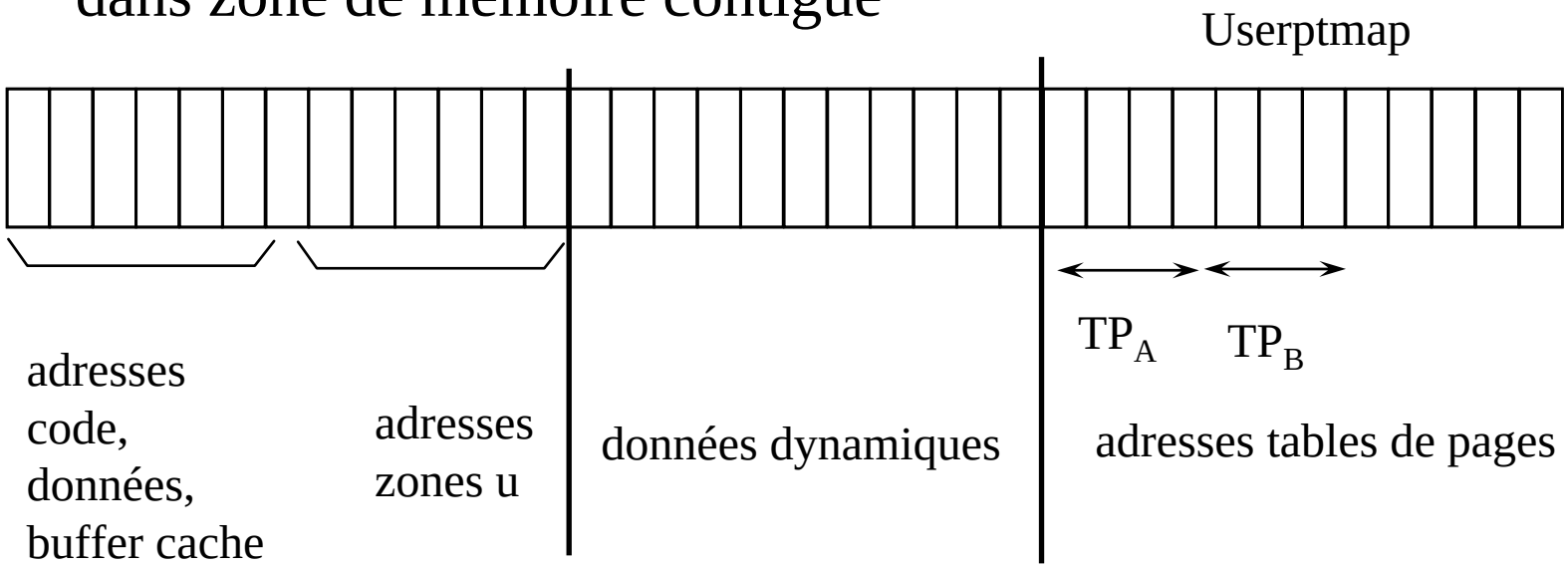
Structure de l'espace d'adressage

4.3BSD sur VAX-11 - adresses sur 32 bits => 4Go



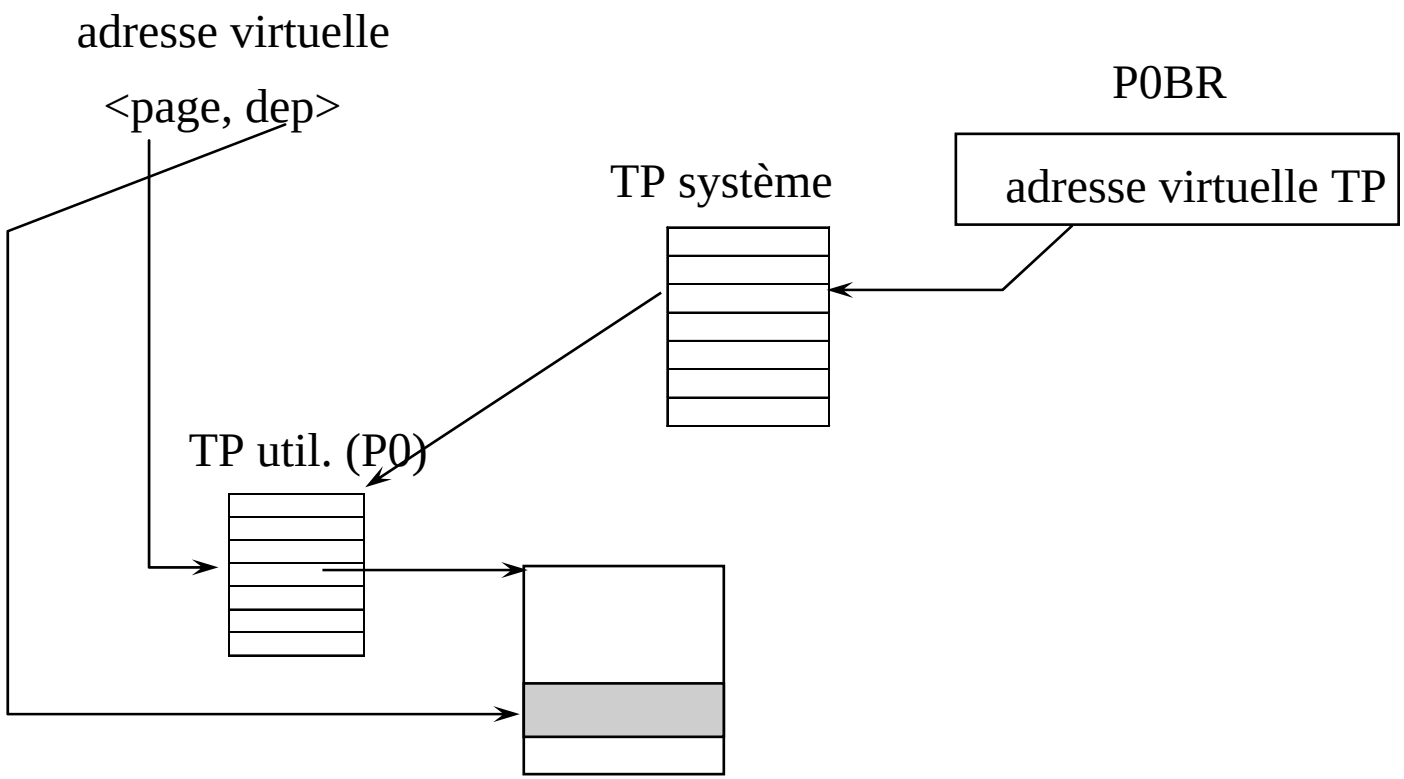
Organisation de l’espace virtuel noyau

- Table des pages du système (TPS) allouée statiquement dans zone de mémoire contiguë



- Tables des pages des processus contiguë dans l’espace virtuel du noyau

Accès aux données utilisateur



Double indirection (passage par la table du système)
=> fait une seule fois ensuite l'adresse est dans la TLB !

Défaut de page - pagein

```
Pagein(adresse virtuelle) {
    Verrouiller la table des pages
    Si (adresse non valide) {
         envoyer SIGSEGV au processus;
         aller fin;
    Si  (page dans le cache des pages) { // page de code
         extraire page du cache
         mise à jour de table des pages;
         tant que (contenu page non valide)
                 sleep(contenu_valide);
    }
    sinon {
         attribuer une nouvelle page;
         Si (page non précédemment chargée et «Zero-fill») initialisée à 0
         sinon {
                 lire la page depuis le périphérique de swap ou fichier exécutable
                 sleep(E/S);
         }
    }
    wakeup(contenu-valide);
    Positionner bit valide; Effacer bit modifié;
    Recalculer priorité du processus;
    Déverrouiller;
}
```

Remplacement de pages

- Objectif: minimiser le nombre de défauts de page
- Idée : exploiter la localité des programmes
  - «Une page anciennement utilisée a une faible probabilité d'être référencée dans un futur proche»
- Algorithme LRU (**Least** Recently Used) trop coûteux

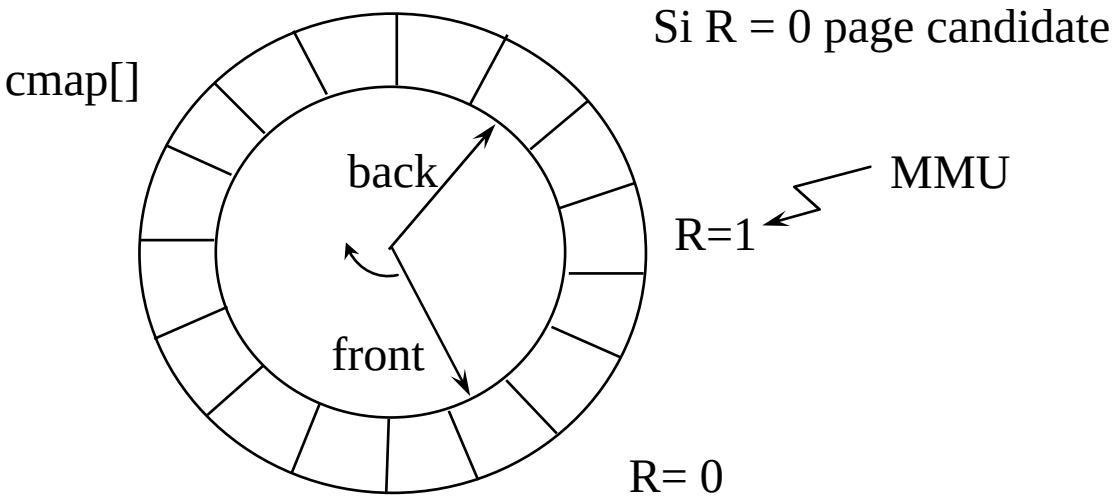
  => approximation de LRU : NRU «**Not** Recently Used»
- Choix d'un remplacement **global**

  => meilleure répartition des pages

  moins bon contrôle de nombre de défauts de page

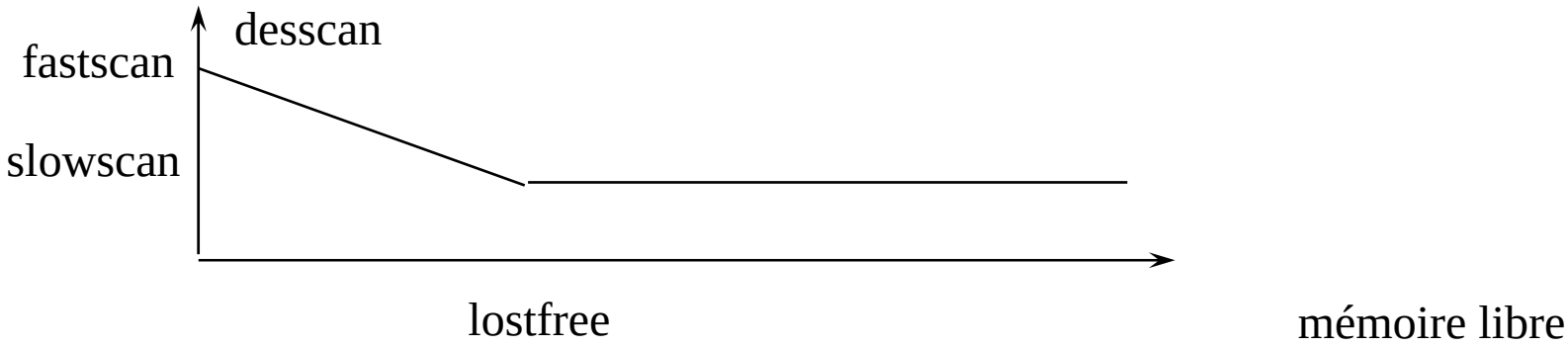
Implémentation du NRU

- Objectif : maintenir une liste de cases libres avec une taille minimum = freelist (taille = freemem).
- Utilisation du bit de référence positionné par la MMU
- 2 passes : 1) Mettre à 0 le bit de référence
  2) Tester (plus tard) ce bit, si toujours à 0 la page peut être récupérée si nécessaire



Le démon de pagination

- Maintient le nombre de cases libres au-dessus d'un seuil
- Réveillé 4 fois par seconde pour tester les cases
- Choix des pages victimes (NRU) à insérer dans freelist
  - si les victimes ont été modifiées, lancer une écriture asynchrone sur le swap
  - écriture terminée => insertion freelist
- Paramètres de bases :
  - Nombre de pages à tester (descan) en moyenne 20 à 30 % des pages testées par seconde
  - Arrêt du démon lorsque freemem > lostfree (= 25% mémoire utilisateur)



Le démon de pagination (2)

- Autres paramètres :
  - desfree : nombre de cases libres à maintenir par le démon (1/8 4.3BSD, 7% 4.4BSD (free_target), 6.25% System V R4)
  - minfree : nombre de cases minimum pour le système (1/16 4.3BSD, 5% 4.4BSD, 3% System VR4)
- Si freemem < minfree activer stratégie de swap
  - => déchargement de processus en entier
  - le démon n’arrive plus à maintenir assez de cases libres

Gestion du swap

- Gérer par le **swapper** (processus 0)
- rôle : charger (swapin) / décharger (swapout) des processus
- Dans les Unix récents intervient uniquement dans les cas de pénurie de mémoire importante

Quand décharger un processus ?

- 3 cas :
  - 1) Userptmap fragmentée ou pleine : impossible d'allouer des pages contiguës pour les tables des pages (propre à 4.3BSD)
  - 2) Plus assez de mémoire libre
    freemem < minfree (BSD)
    < GPGSLO (SVR4)
  - 3) Processus inactifs plus de 20 secondes
    (exemple : un utilisateur ne s'est pas déconnecté)
- => le processus victime est entièrement déchargé
  - Toutes les pages + zone u + tables des pages

Quel processus évincé

- 2 critères :
  - Temps processus endormi en mémoire
  - Taille du processus
- Choisi d’abord les processus endormis depuis plus de 20 sec. (maxslp)
- Si non suffisant : les 4 plus gros processus
- Si non suffisant : ???

Le swapper

- Algorithme de sched

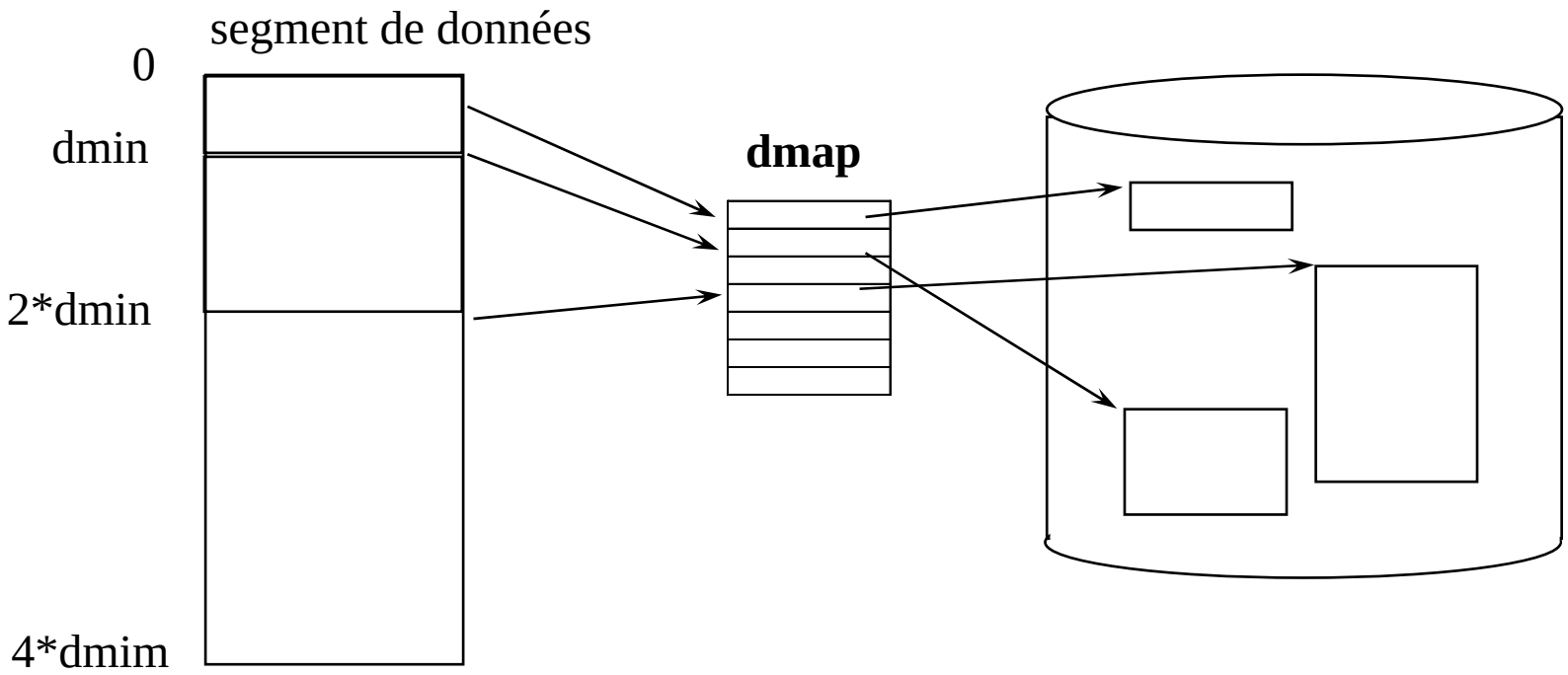
```
boucle
    recherche processus SRUN et non SLOAD le plus ancien
    si non trouvé
        alors sleep (&runout, PSWP); continuer
    sinon
        si swapin(p); continuer
        /* place insuffisante en mémoire */
        Si existe processus endormis ou en mémoire depuis longtemps
          alors swapout(p); continuer
          sinon sleep(&runin, PWSP);
        fin si
    fin si
fin boucle
```

Gestion de l’espace de swap

- Une ou plusieurs partitions (sans système de fichiers)
- Le swap est préalloué à la création du processus (pour les données et la pile)
  - => pouvoir toujours décharger un processus
- Swap du code :
  - Code déjà présent sur disque dans système de fichier
  - Swappé pour des raisons de performance !
  - Code swappé uniquement si plus utilisé par un processus en mémoire (champs x_ccount indique le nombre de processus en mémoire utilisant le code)

Espace de swap (2)

- Pour chaque segment une structure dmap stocké dans zone U
  - Premier bloc de taille 16K (= dmim)
  - Chaque bloc suivant est le double du précédent



Attention: 1 seule copie pour le code => dmap du code dans struct text

Algorithme swapout

- Swapout : décharger un processus sur disque

1- Allouer espace de swap pour zone U et table des pages
2- Décrémenter x_ccount, si x_ccount = 0 décharger les pages de code
3- Décharger les pages résidentes et modifiées sur le swap
4- Insérer toutes les pages déchargées dans freelist
5- Décharger table des pages, zone U, pile système
6- Libérer zone U
8- Mémoriser dans struct proc l'emplacement zone U sur disque
7- Libérer tables des pages dans Userptmap

Algorithme swapin

- swapin : chargement d’un processus

1- Allouer table de pages dans Userptmap
2- Allouer une zone U
3- Lire table des pages, zone U
4- Libérer espace table des pages, zone U sur disque
5- charger éventuellement le code et l’attacher au processus
6- Si le processus à l’état prêt (SRUN), l’insérer dans file des processus prêts

Création d'un processus

- BSD : données et pile dupliquées, code partagé
- **Swap** :
  - Allouer espace sur le swap pour le fils (données pile)
  - Espace pour le texte déjà alloué par le père (exec)
- **Table des pages** :
  - Allouer des pages pour les tables de pages du fils
    (trouver des entrées contiguës dans Userptmap, prendre des cases dans la freelist)
- **Zone U** :
  - créer une nouvelle zone U avec le contenu de la zone U du père
- **Code** :
  - Ajouter le fils dans la liste des processus partageant le code
  - x_count++, x_ccount++

Création de processus (2)

- **Données et pile** :
  - Pages référencées par les segments de données et de pile copiées
  - Pages marquées modifiées
  - Pages swappées copiées

=> très coûteux => Création d'un nouvel appel le **vfork**

- Constatation : le fork et très souvent suivi d'un exec

  => recopie inutile !

- vfork : pas de recopie en attendant le exec
  - Père et fils partagent le même espace d'adressage
  - Création uniquement de proc, zone U, table des pages
  - Père reste bloquer jusqu'à ce que le fils fasse exec ou exit (pb de cohérence)

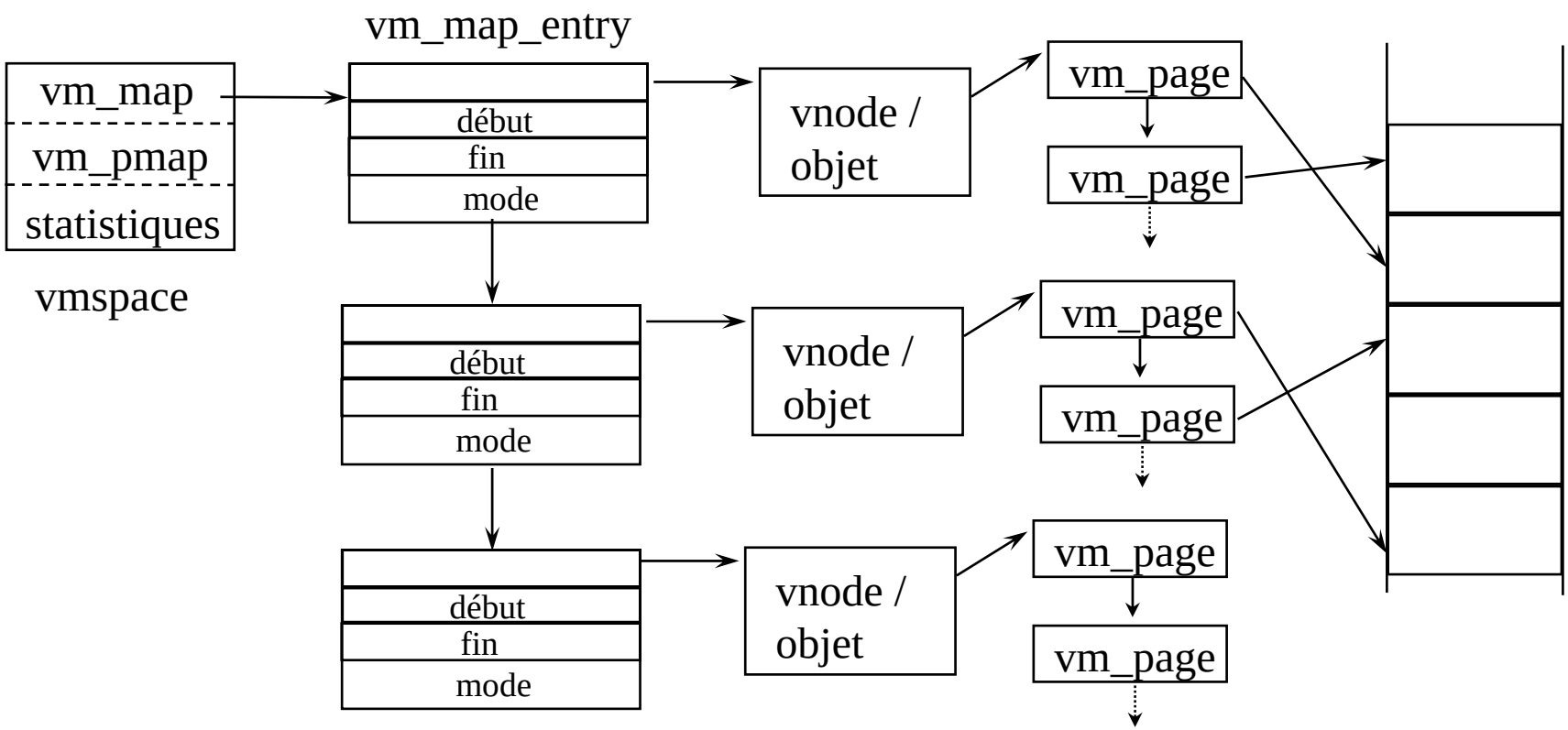
Les nouveaux systèmes

- Système V Release 5
- Solaris
- 4.4 BSD
- Linux

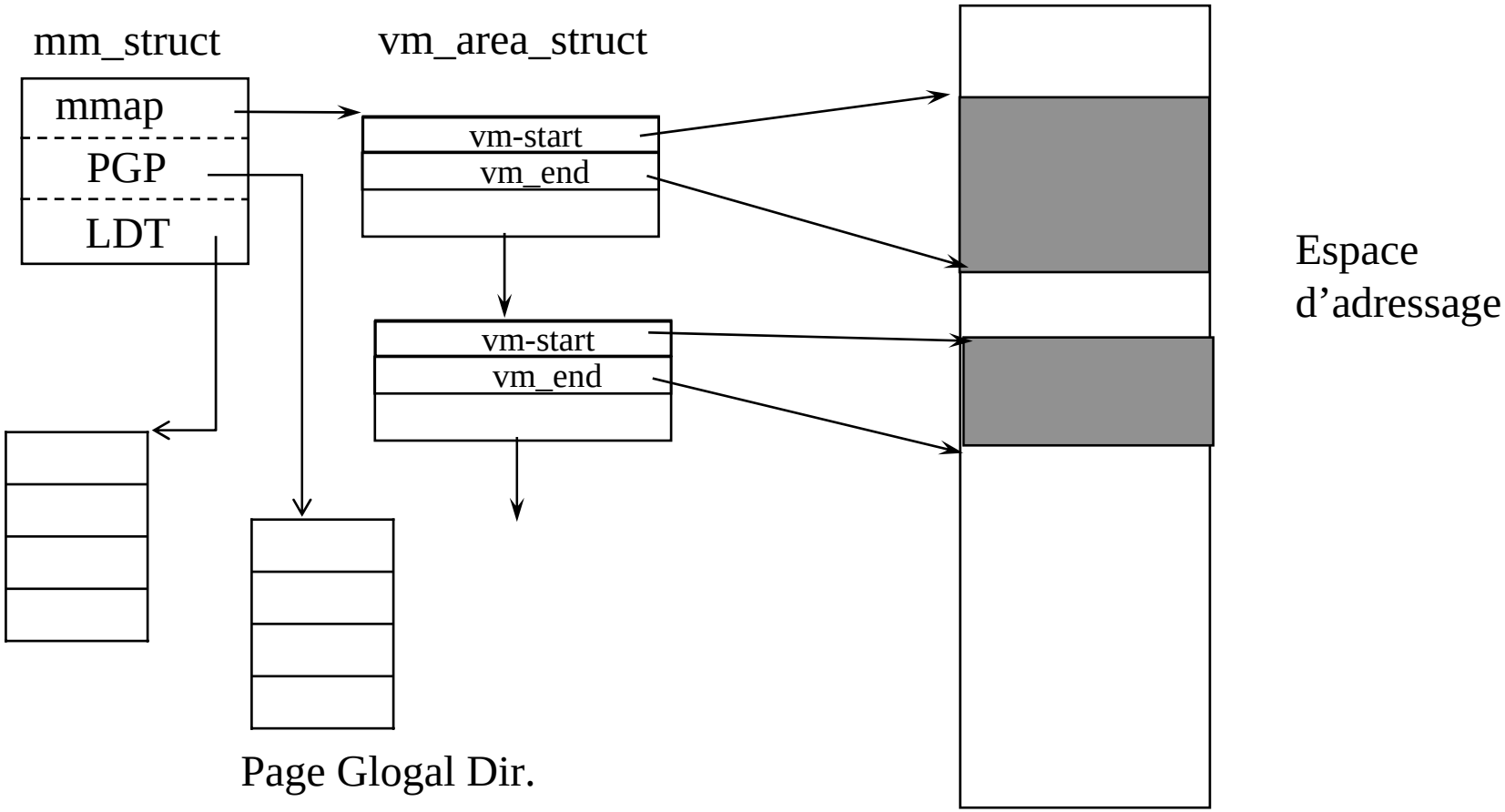
} Mémoires virtuelles très proches

- Nouveautés :
  - Structures générales
  - Fichiers «mappés»
  - Copie-sur-écriture

Structure d’un espace 4.4BSD

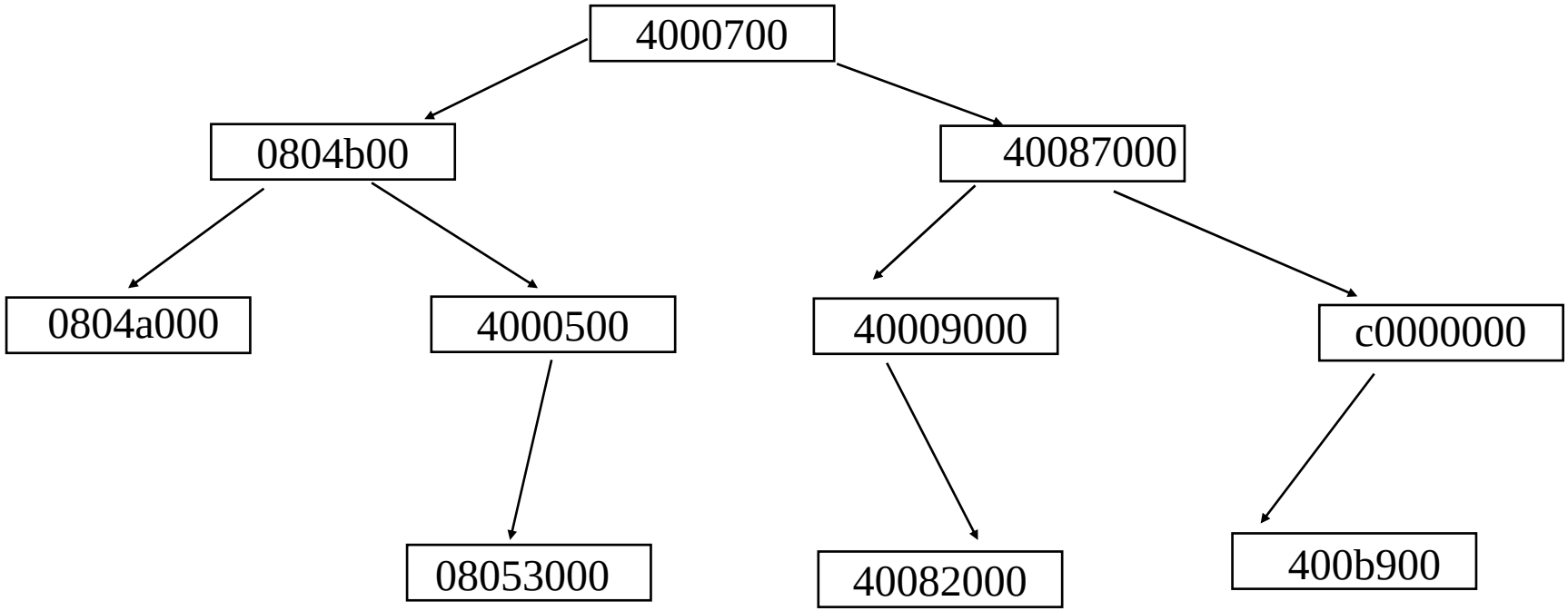


Structure dans Linux



Organisation de l’espace mémoire d’un processus

- Lorsque beaucoup de régions
  - Liste de région => arbre des régions (arbre AVL)
- Ex : linux : /proc/pid-processus/maps



Visualisation mémoire sous linux

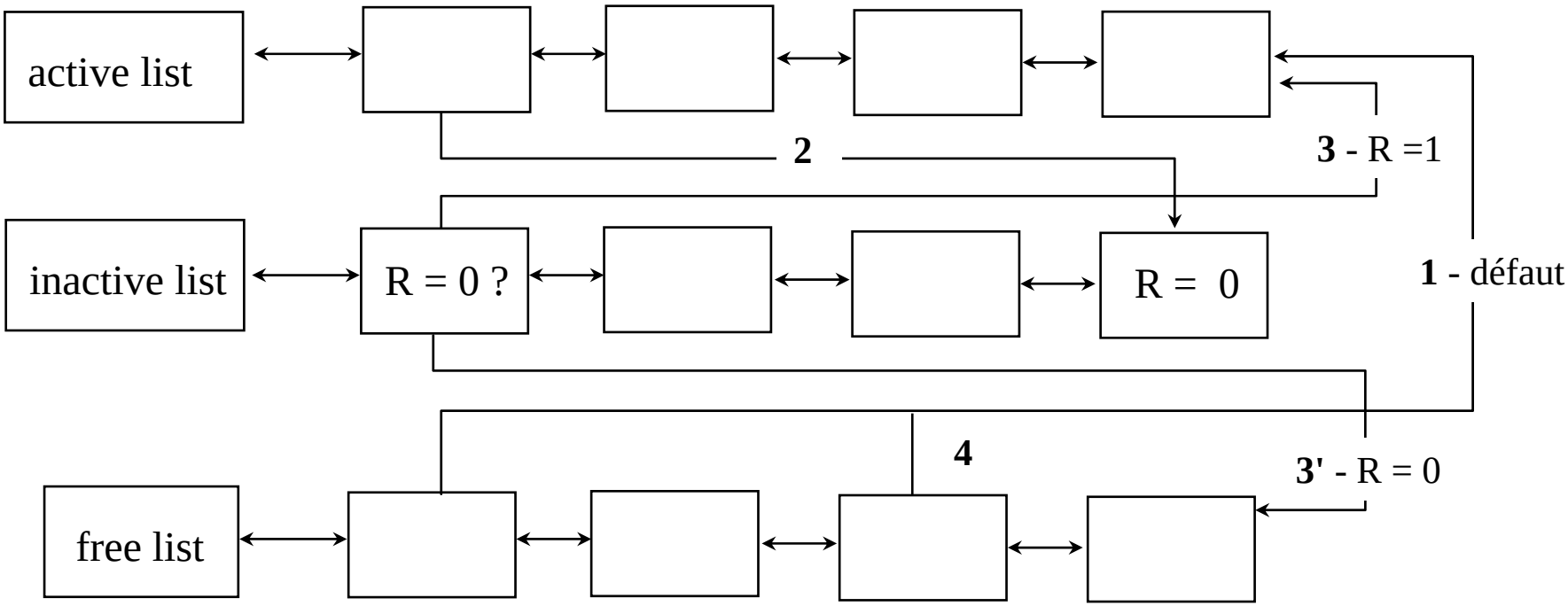
```
# more /proc/1/maps
08048000-0804f000 r-xp 00000000 03:06 80252      /sbin/init
0804f000-08051000 rw-p 00006000 03:06 80252      /sbin/init
08051000-08055000 rwxp 00000000 00:00 0
40000000-40012000 r-xp 00000000 03:06 69906      /lib/ld-2.1.3.so
40012000-40013000 rw-p 00011000 03:06 69906      /lib/ld-2.1.3.so
40013000-40014000 rw-p 00000000 00:00 0
4001d000-400fc000 r-xp 00000000 03:06 69912      /lib/libc-2.1.3.so
400fc000-40101000 rw-p 000de000 03:06 69912      /lib/libc-2.1.3.so
40101000-40104000 rw-p 00000000 00:00 0
bfffe000-c0000000 rwxp fffff000 00:00 0
```

Les objets et paginateurs

- Un **paginateur** par type d'objet
  => chargement/déchargement des pages de l'objet

- Structure vm_pmap : dépendante de la machine
  Conversion adresse physique <--> adresse logique
  Fonction de manipulation de la table de page
  - Gérer les protections (copie-sur-écriture)
  - Mise à jour
  - Création ..

Remplacement de pages



Algorithme: Fifo avec seconde chance

Optimisation : copie sur écriture

- Objectif : éviter les recopies du fork
- Autoriser le partage en écriture
  - segment de pile de données partagées, les pages sont recopiées uniquement si elles sont modifiées