Gestion de la hiérarchie mémoire

1)Rappel de la hiérarchie mémoire

2)La mémoire cache

3)Le concept de mémoire virtuelle

Niveau	Nom	Objets impliqués	Exemple d'opérations
13	Shell	Env utilisateur	Langage shell
12	Processus utilisateur	Processus	Fork, quit, kill, suspend, resume
11	Répertoire	Répertoire	Mkdir, remove, attach, detach
10	Périphérique	Imprimante, écran, clavier	Open, close, read, write
9	Système de Gestion de fichiers	Fichiers	Create, destroy, open, close, read, write
8	Communications	Pipes	Create, Destroy, open, close, read, write
7	Mémoire virtuelle	Segments, pages	Read, Write, Fetch
6	Mémoire de masse	Blocs de données, canaux d'E/S	Read, Write, Allocate, free
5	Processus, tâches et flots	Tâches, sémaphores, liste de tâches	Suspend, Resume, Wait, Signal
4	Interruptions	Invocation pilote (handler)	Invoke, mask, unmask, retry
3	Procédures	Appel et retour, contexte	Call, return
2	Jeu d'instructions	Pile d'évaluation, gestion des UF, microprogrammes	Load, Store, add, addf, branch, jsr, rts, rti
1	Carte et circuits	Registres, UF, bus …	Clear, activate, transfert …

La hiérarchie mémoire

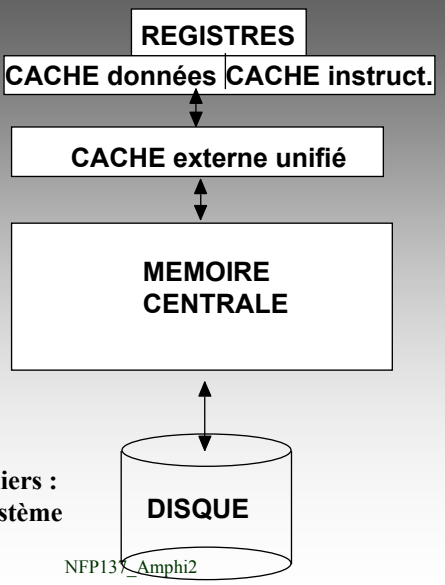
Gérée par :

le compilateur

un dispositif cablé

le système d'exploitation

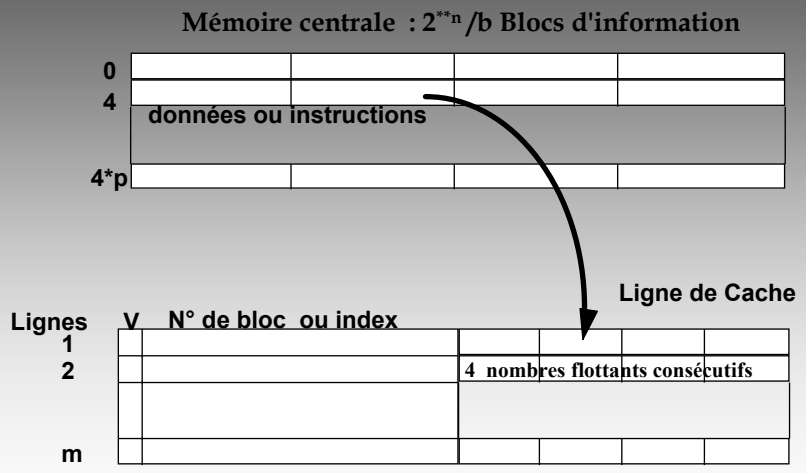
la création de fichiers : commandes ou système



- La mémoire, contenant des instructions et des données, est organisée en un ensemble de mots numérotés consécutivement à partir de 0
- Le N° associé à chacun de ces mots est l'adresse physique
- L'ensemble de ces adresses forme l'espace d'adressage physique de la mémoire

- L'adresse d'une donnée contenue dans une instruction est une adresse logique, relative au début du programme
- L'ensemble des adresses accessibles par un programme forme l'espace d'adressage logique du programme
- L'organisation de ces espaces logiques pour l'ensemble des programmes constitue la mémoire virtuelle

Le mécanisme de la mémoire cache



Paramètres

Elle est caractérisée par :

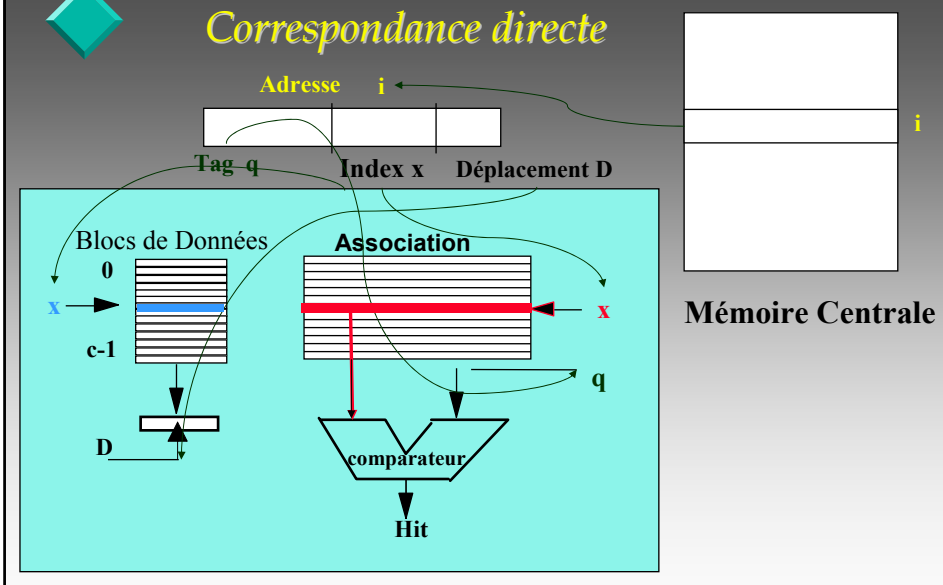
- **sa taille en KO**
- **le nombre de blocs et la taille de ses blocs**
- **le mécanisme de correspondance avec la mémoire centrale**
- **l'algorithme de remplacement des blocs**
- **la bande passante avec la mémoire**

Taille des blocs de 16 à 128 octets

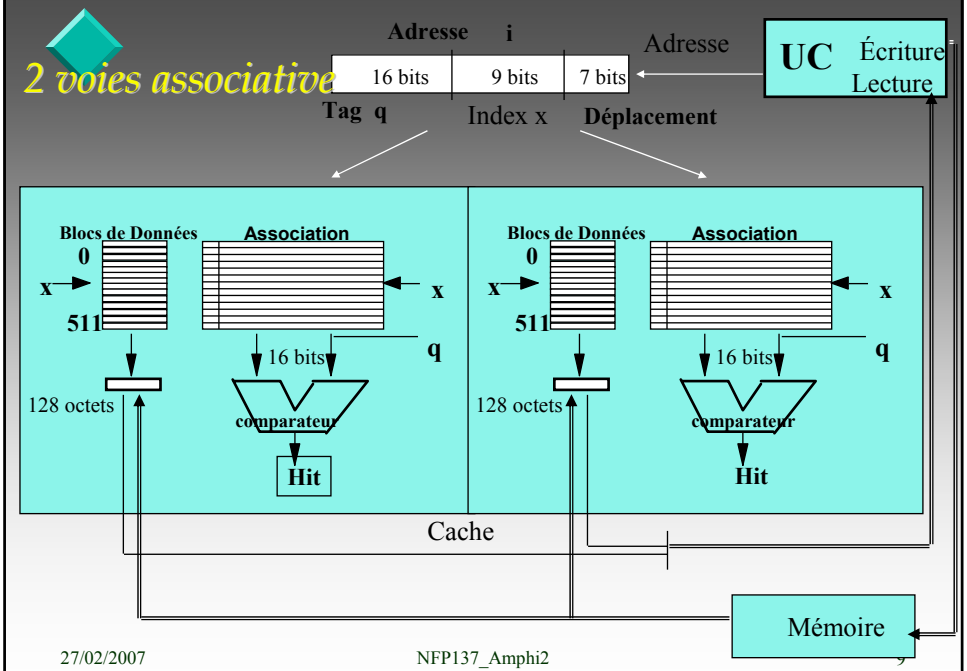
Où le bloc est-il placé dans le cache ?

Correspondance directe ou associative par ensembles

Correspondance directe



2 voies associative



Impact du cache sur la performance

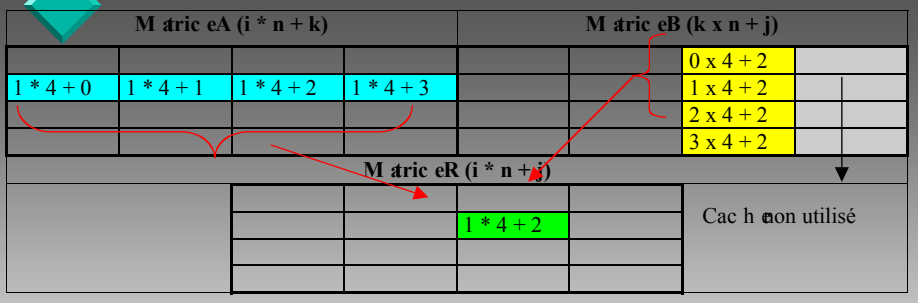
Soit **T** le temps d'accès moyen à la mémoire
Soit **k** le taux d'échec au cache

T_{succ} le temps d'accès au cache
T_{mem} le temps d'accès à la mémoire
T_{echec} le coût d'un échec cache

$$\mathbf{T} = T_{succ} * (1-\mathbf{k}) + T_{mem} * \mathbf{k} + T_{echec} * \mathbf{k}$$

Exemple : soit à calculer R = A * B matrices (n*n)

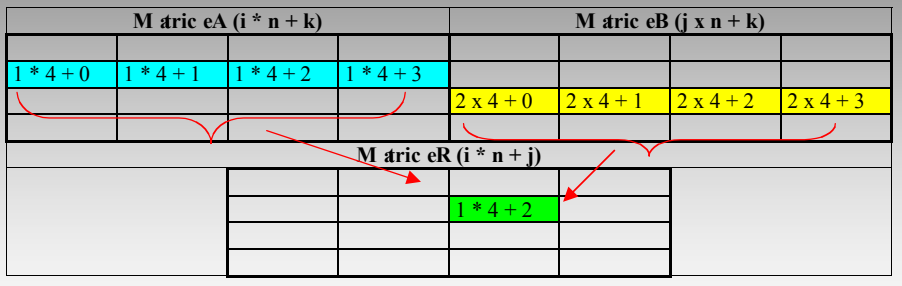
```
Pour i = 0 à n-1
        Pour j = 0 à n-1
                Pour k = 0 à n-1
                R_ij = R_ij + A_ik * B_kj
```



Coût algorithme 1 : $O(n^3)$

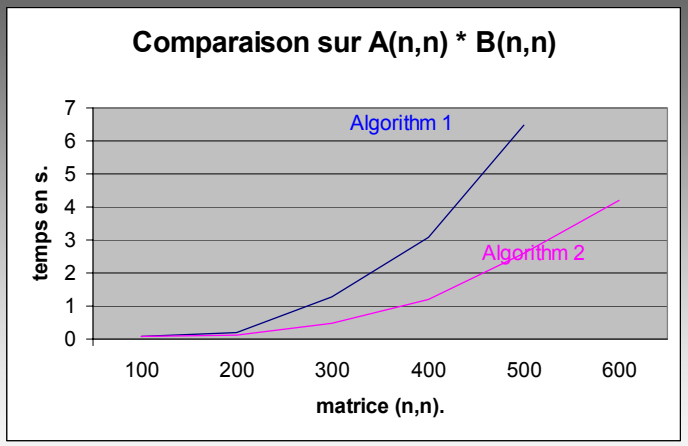
Deuxième solution :

B = ^{t}B
Pour i = 0 à n-1
Pour j = 0 à n-1
Pour k = 0 à n-1
$\mathbf{R_{ij} = R_{ij} + A_{ik} * B_{jk}}$



Coût algorithme 2 : $O(n^2) + O(n^3)$

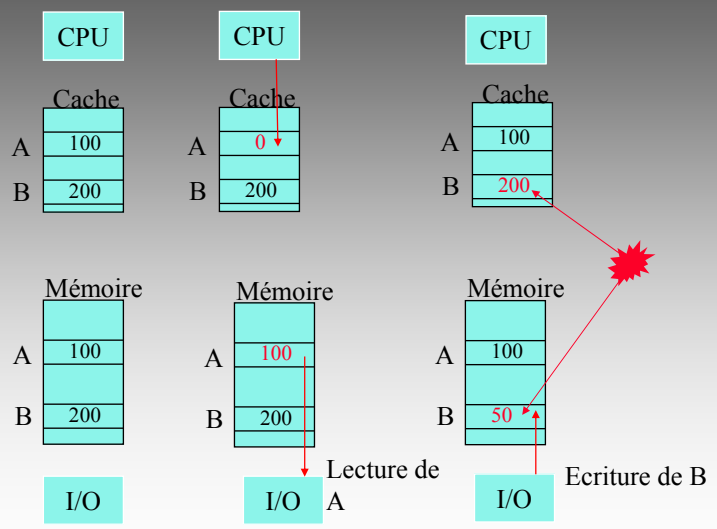
Résultat des temps d'exécution



Paramètres de la mémoire cache

Taille des blocs	32 à 128 bytes
Temps de "hit"	1 cycle
Coût des Défauts de cache	8 à 32 cycles
(Temps d'accès)	2 à 8 cycles
(Temps de transfert)	2 à 22 cycles
Taux de défaut	1 % - 20 %
Taille du cache L1 I/D	16 KO à 64 KO/ 16 KO à 64 KO

Le problème de cache-cohérence



Solutions :

Pour la lecture de la mémoire par les E/S :

Cache « write-through »

La mémoire centrale est mise à jour à chaque écriture dans le cache.

Pour l'écriture par une E/S en mémoire :

- Solution logicielle système :

Aucun bloc de buffer réservé pour les E/S ne peut être caché.
Les pages correspondantes sont marquées non-cachables

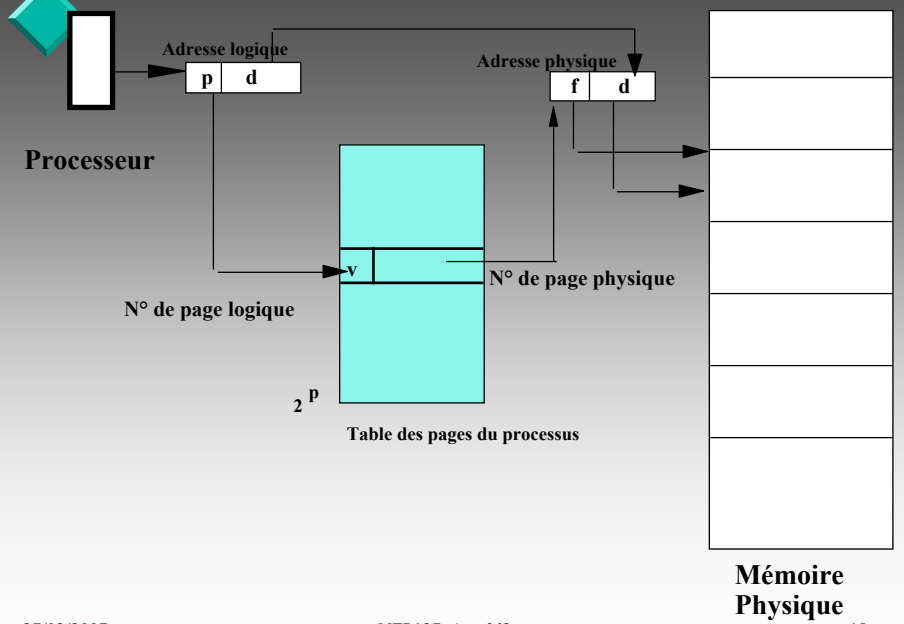
- Solution matérielle :

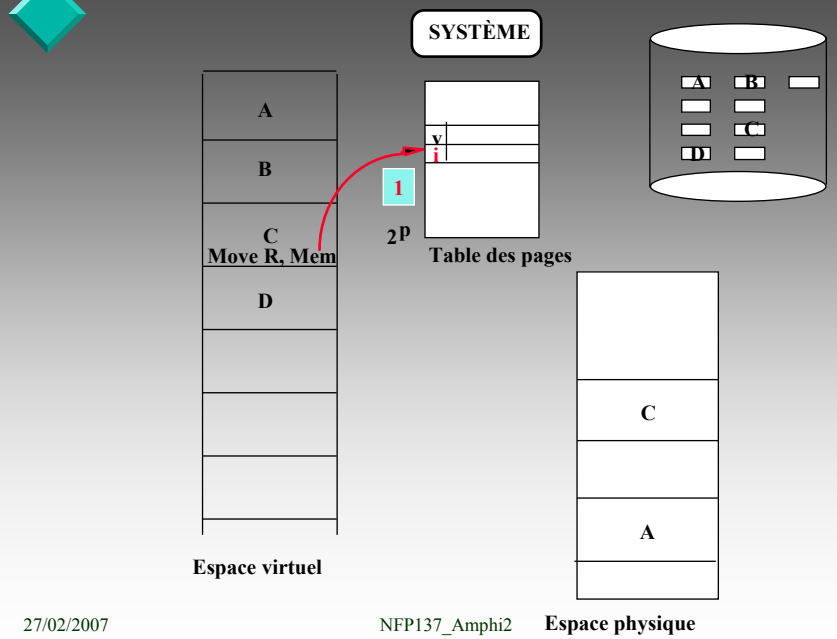
Tester les adresses E/S et vérifier si elles sont ds le cache à partir du bus mémoire. Si oui, les entrées caches sont invalidées.

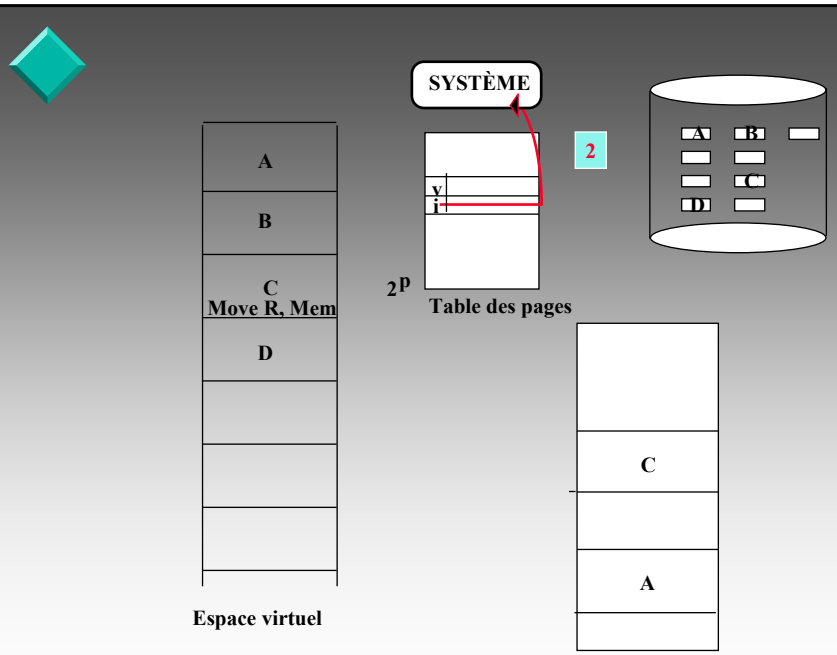
Le concept de mémoire virtuelle

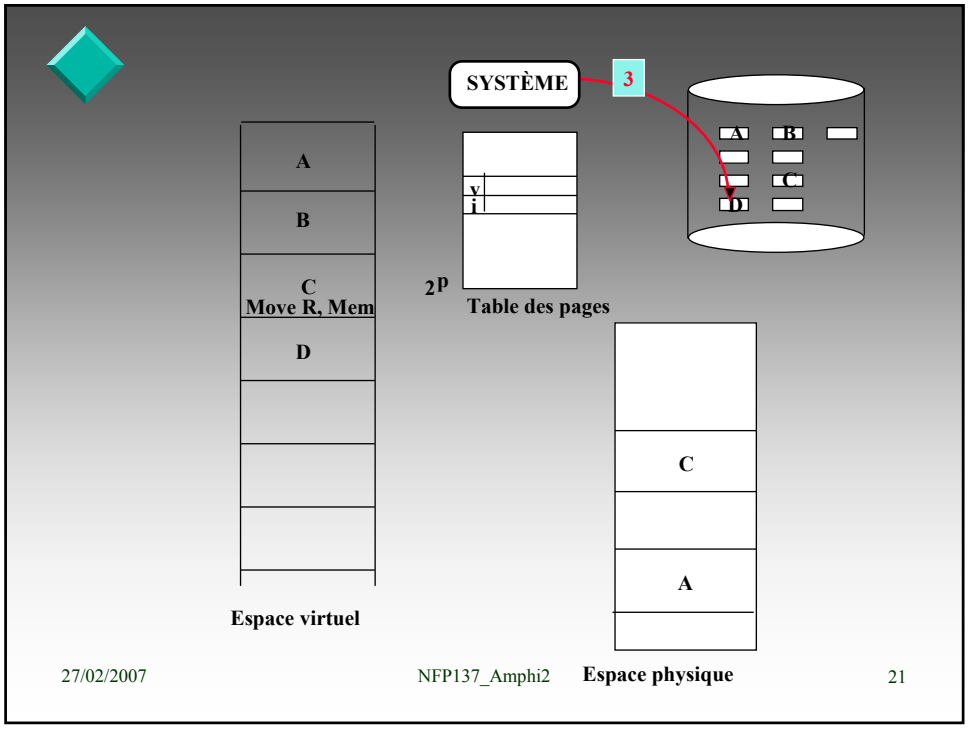
- Les espaces virtuels des processus sont divisés en pages
- Le système de gestion de la mémoire (MMU) gère les pages
- Plusieurs processus sont actifs en mémoire : seules les pages utiles sont chargées en mémoire

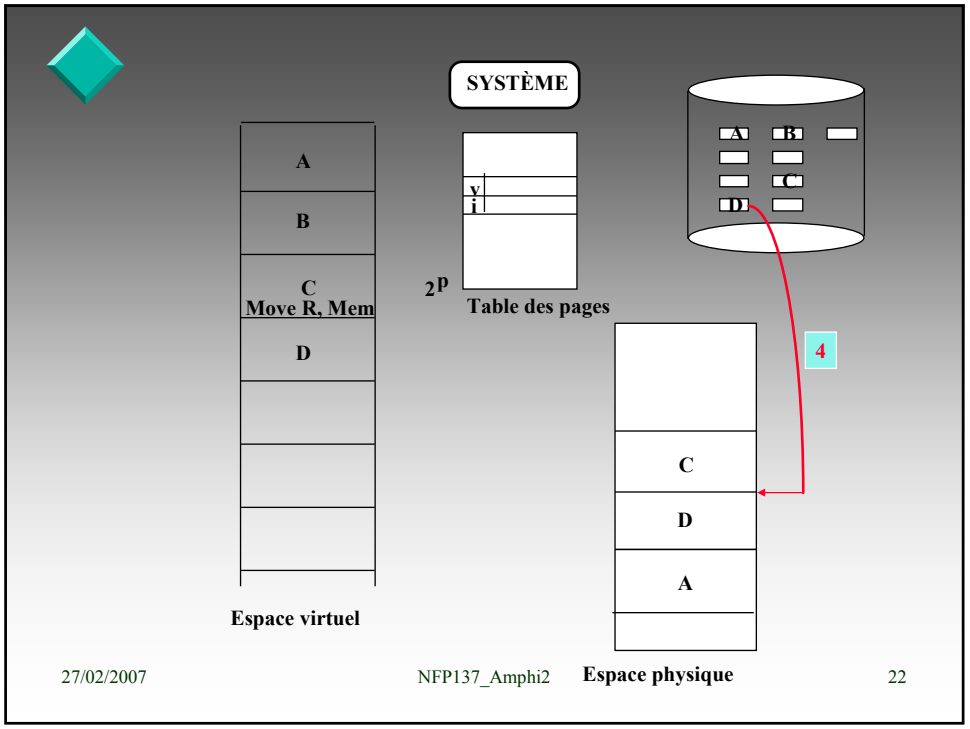
➪ Il est donc possible de partager du code ou des données entre plusieurs processus

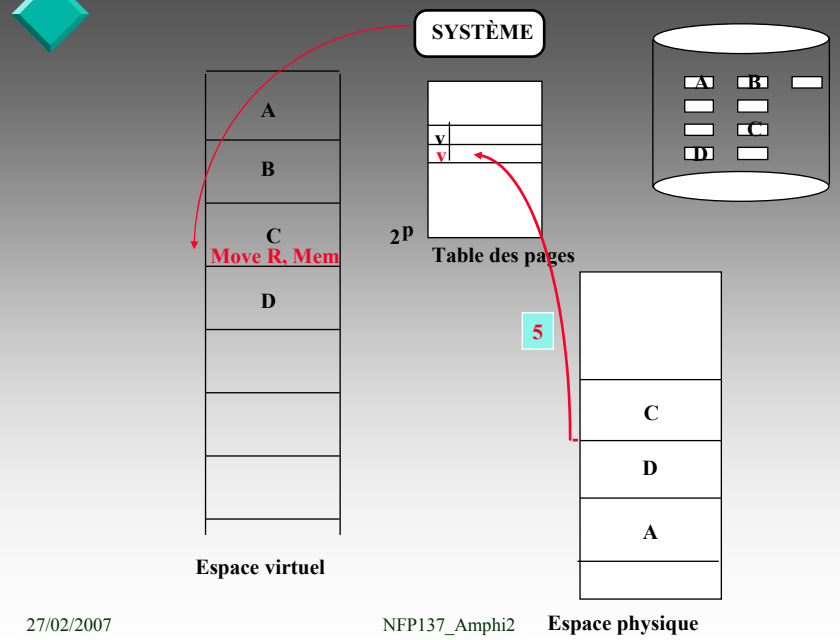



SYSTÈME
A
B
C
Move R, Mem
D
v
i
1
2^p
Table des pages
A
B
C
D
C
A
Espace virtuel
Espace physique


SYSTÈME
2
A
B
C
Move R, Mem
D
v
i
2^p
Table des pages
A
B
C
D
C
A
Espace virtuel
Espace physique

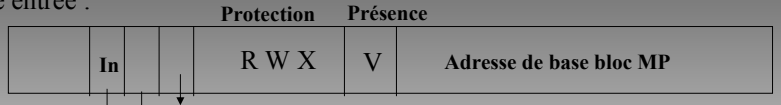

SYSTÈME
3
A
B
C
Move R, Mem
D
v
i
2^p
Table des pages
A
B
C
D
C
A
Espace virtuel
Espace physique


SYSTÈME
A
B
C
Move R, Mem
D
v
i
2^p
Table des pages
A
B
C
D
4
C
D
A
Espace virtuel
Espace physique



Format d'une entrée de la table des pages

Une entrée :



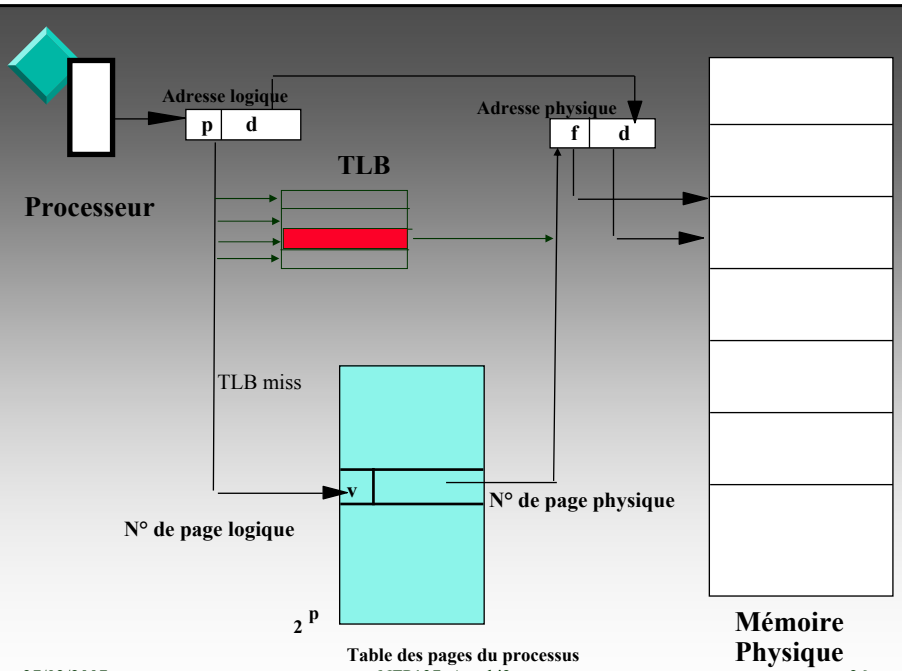
Modification :
Positionnée lorsqu'une page est écrite

Référence :
Positionné à chaque accès à la page (L/E)

Cachable :
Ces pages marquées ne peuvent être transférées en cache.

Solution pour améliorer la performance et la taille de la table de traduction AV-AR

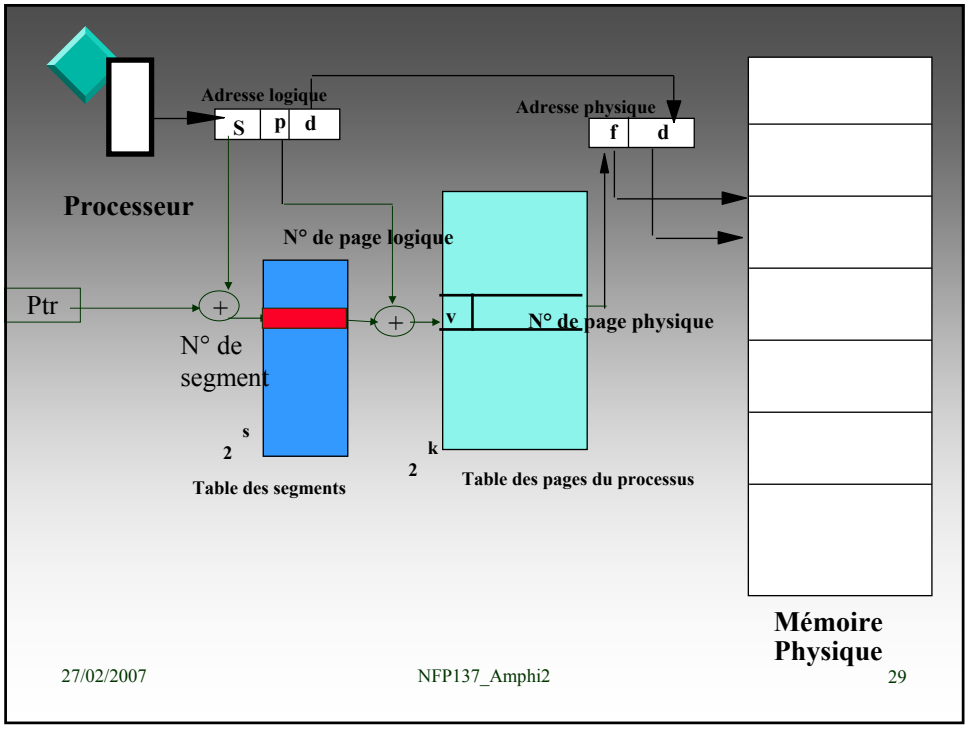
- Taille de la table
  - pages de 4KO (2 12) sur 4GO (2 32) soit (2 20) entrées => 1 Méga entrées > 4 MO par processus
- La table des pages est en mémoire
  - 2 accès mémoire pour une information en mémoire
  - Présence d'un cache (TLB) hard ou soft
- TLB
  - Cache ds le MMU
  - Données de contexte de la tâche
- Segmentation
  - Introduction d'une segmentation de la MV pour diminuer la taille de la table.



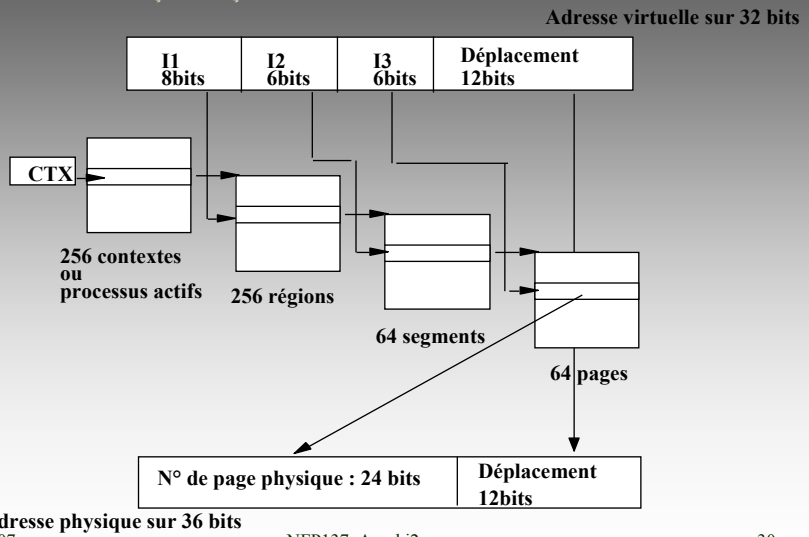
Mémoire virtuelle segmentée

- Pagination : espace d'adressage linéaire
- Segmentation : collection discontinue d'espaces d'adressage linéaire :
  - Une adresse virtuelle segmentée Ad = [S | d = AV]
    - S = N° de segment
    - d = déplacement dans le segment = Adresse Virtuelle
  - Exemple d'utilisation : un segment par module fonctionnel
    - Code source S=0, données initialisées S=1, pile S=2, tas S=3 ….
  - Des allocations dynamiques peuvent être réalisées dans de nouveaux segments à l'exécution (création d'un seul module .exe pour une MV)
  - Simplifie la gestion de code ré-entrant (N code de tâches issue de N activations de processus)
  - Possibilité d'accorder des droits (RWX) différents à chaque segment

- SA = [s |d]
  - S = sélecteur de segment -> (Base, Limite)
  - VA = Base + d = [10-bit DIR] | 10 bit Page | 12 bit Déplac.]
  - DIR = Répertoire de Pages -> pointeur 32 bits 2nd niveau de pages
  - Page = 2nd niveau de table des pages -> 20 bit adresse de bloc b
  - Adresse Physique = [b | Déplac]

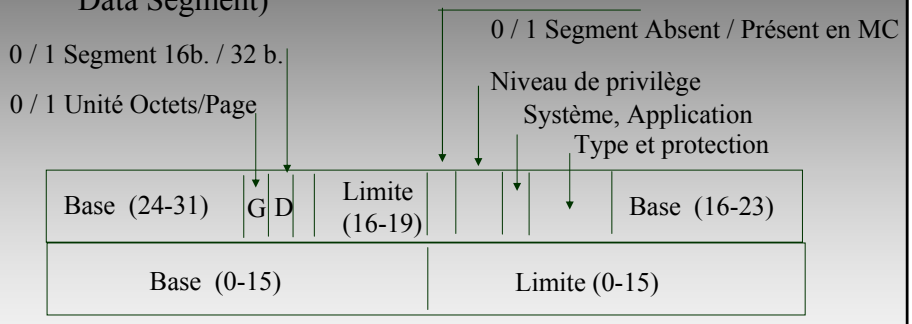

Adresse logique
S
p
d
Adresse physique
f
d
Processeur
N° de page logique
Ptr
N° de segment
s
2
v
N° de page physique
k
2
Table des segments
Table des pages du processus
Mémoire Physique

Super Sparc


Adresse virtuelle sur 32 bits
I1 8bits
I2 6bits
I3 6bits
Déplacement 12bits
CTX
256 contextes ou processus actifs
256 régions
64 segments
64 pages
N° de page physique : 24 bits
Déplacement 12bits
Adresse physique sur 36 bits

Exemple : Intel Pentium

- Le Pentium a 6 registres segment (CS Code Segment, DS Data Segment)

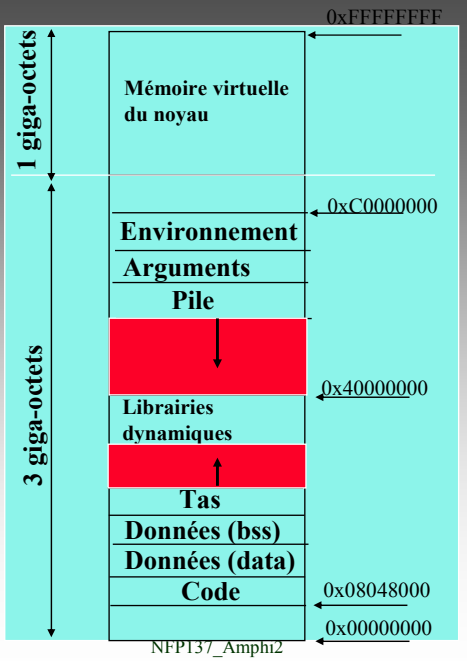


Page de 4 KO ; Adresse de base 32 b. ; Limite 20b. => Taille segment = 1MO ou 4GO

Paramètres de la mémoire virtuelle

Taille des pages	512 - 8192 bytes
Temps de “hit”	1 à 10 cycles
Coût des Défauts de pages	1.000.000 à 4.000.000 cycles
(Temps d’accès)	990.000 à 3.900.000 cycles
(Temps de transfert)	10.000 à 100.000 cycles
Taux de défaut	0,000001 % - 0,001 %
Taille mémoire	16 MO à 8.192 MO

MV-UNIX

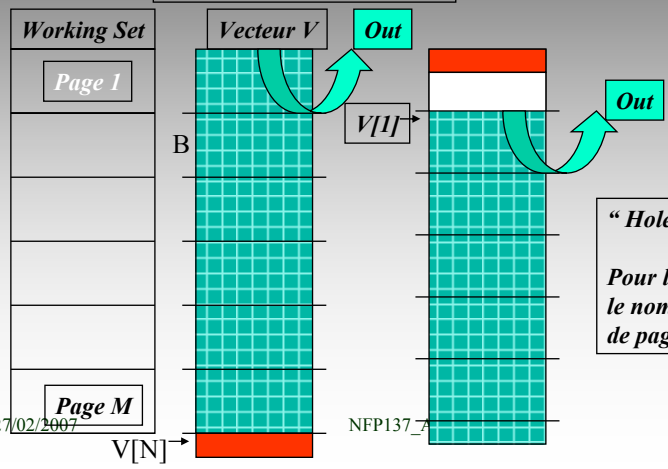


- Processus de gestion mémoire
  - Swapper (PID=0) : activé toutes les 4s. pour tester si le processus doit être swappé :
    - Candidats Swap out : les inactifs depuis longtemps, les présents depuis longtemps
  - Démon Init (PID=1) : engendre tous les autres processus
  - Gestionnaire de pages (PID=2)
    - Page à la demande
    - Remplacement global des pages modifiées
    - Mécanisme automatique de libération de pages
      - Si le nombre de pages libres descend au dessous d'un seuil fixé, un démon est activé pour swapper les pages.
- Appels système
  - p = malloc (int size) : allocation dynamique de size octets ds le tas
  - free (p) : libération de la zone pointée par p
  - Brk (end_data_segment) augmente le segment de données alloué au processus

Un exemple : influence de la politique de gestion du remplacement des pages. (LRU)

```
Soit : Real V[N] ;
       for j = 1 to P
              for i =1 to N
                     V[i] = f(i,j) ;
```

*Avec N = M*B + 1*
V[1..N-1] sont en mémoire



" Hole effect "

Pour la decomposition LU le nombre de défauts de pages est en $O(N^3)$

Algorithme de remplacement des pages

- FIFO
  - L'OS maintient une liste ordonnée des pages utilisées et sélectionne la plus ancienne pour être remplacée.
    - La nouvelle page est insérée en tête de liste
    - La page en queue de liste est remplacée
- LRU
  - Chaque page mémorise l'instant du dernier accès et la page la plus vieille est candidate pour être remplacée
  - Nécessite un support hard et une approximation pour gérer le temps efficacement.
- LFU, NFU, NRU
  - Least Frequently use, Not Frequently Use, Not Recently Use …

Rappel : format d'une entrée

				Protection	Présence	
	In	R	M	R W X	V	**Adresse de base bloc MP**

- Support matériel pour le remplacement des pages
  - Bit R, Page récemment référencée
  - Bit M, Page modifiée en MC
  - R est mis à 0 périodiquement (Ex. à chaque commutation de contexte)
- Support de l'OS
  - Au 1er référencement de la page, R=1 et M=0
  - RAZ de R périodiquement et R=1 a chaque référence de page
  - Sur une modification du contenu de la page M=1

Phénomène d'écroulement

- De nombreux défauts de page (voir exemple T35)
  - Résultats : page-in et page-out très fréquent => perte de performances
- Causes
  - Trop de processus nécessitant plus de mémoire que disponible
    - Réduire le nombre de processus activables
  - Pas assez de pages en mémoire pour les processus actifs
    - Pré-charger des pages formant le Working Set nécessaire
  - Mauvaise localité des données accédées
    - Modifier la programmation
  - Trop de pages en mémoire pour une tâche => trop de défauts pour les autres processus
    - Libérer les pages en dehors du Working Set

Taille du Working Set

- Le principe de séquentialité des données et du code induit un nombre restreint de pages récemment référencées. Cet ensemble de pages se modifie avec le temps.
- Objectif : déterminer un Working Set qui minimise les défauts de pages.
  - Politique de chargement :
    - À la demande : les pages sont allouées seulement sur un défaut
    - Par anticipation : le programmeur peut anticiper l'identification des pages qui seront nécessaires.

    - Fixe ou Variable : le processus possède un nombre fixe (ou variable de pages allouables)
  - Optimum : Allocation globale d'un nombre variable de pages ce qui permet au système de maintenir un WS nécessaire et de gérer les pages selon leur fréquence d'accès.
- Gestion des pages (libération et pages libre)
  - Le système identifie périodiquement les pages les moins référencées ou les plus anciennes et les swappe sur disque => maintient d'une liste de pages libre
- Taille optimale
  - Analyse
    - Taille moyenne d'un processus = s
    - Taille de la page = p et taille d'une entrée de la table des pages =e

- Politique d'allocation :
  - Local ou global : les pages du processus provoquant le défaut (local) sont candidates au remplacement ou toutes les pages du système (global)
  - Nombre de pages nécessaires par tâche = (s/p)
  - Taille de la table des pages par tâche = (s/p)*e
- Overhead = (s*e) / p
- Min d(Overhead)/dp = 0 => $-s*e/p^2 = 0$ => p = sqrt(2s*e)
  - Ex : S= 2MB, e = 8 => optimum = 5793 octets => pages = 4Ko à 8Ko

❖ Pages verrouillées
- E/S en DMA (voir cours N°1)
- Pages associées à des buffers d'E/S (adresse physique pour le buffer d'E/S)
- Pages verrouillées en mode noyau

Ex: Windows NT

❖ Structures de données
- Une table des pages par processus
- Une BD des pages
  - Utilisé par le gestionnaire de MV pour enregistrer l'état des pages physiques
  - Pages sont :
    - Valides : en cours d'utilisation
    - Zero : libre et initialisées à 0 ou free (non init.)
    - Standby : hors du WS mais utilisées
    - Modified : hors du WS, modifiées et pas swappées
    - Bad : erreur physique

❖ Politique et Working Set
- Page à la demande
- FIFO local pour le remplacement
  - Chaque processus a une limite max et min du WS
- Réglage automatique du WS
  - Si la mémoire du processus est utilisée en totalité, le gestionnaire de MV swappe les pages pour maintenir le WS à son minimum.

EXERCICES

Compréhension du mécanisme de Mémoire Virtuelle

EXERCICE1 : Soit le programme suivant :

Adresse de début 1020

Debut : I1 : Charger le mot d'adresse 6144 dans R0
I2 : Empiler R0 (écriture puis SP--)
I3 : Appeler une procédure située à l'adresse 5120 et placer l'adresse de retour en pile
I4 : Décrémenter le pointeur de pile de 16
I5 : Comparer la valeur en sommet de pile à la constante 4
I6 : Effectuer un branchement à l'adresse 5652 si égalité
Fin

La taille des pages est de 512 octets.
Chaque instruction fait 4 octets.
La pile est située à l'adresse 8192 et croît vers les adresses décroissantes

0	0
1	512
2	1024
3	1536
4	2048
5	2560
6	3072
7	3584
8	4096
9	4608
10	5120
11	5632
12	6144
13	6656
14	7168
15	7680
16	8192

Donner la liste ordonnée des pages utilisées par ce programme :

EXERCICE2 :

On souhaiterait rajouter à un processeur classique, muni de registres 32 bits, d'un chemin de données 32 bits, d'une mémoire centrale de plusieurs millions d'octets adressable par 32 bits d'adresse, un gestionnaire de mémoire virtuelle, fournissant des pages à la demande à chaque défaut rencontré.

On considère des pages de 4K octets et des vecteurs de flottants simple précision (4 octets) inférieurs à 1024 éléments.

Identifier le problème crucial à résoudre pour implémenter une nouvelle instruction qui serait :

V1 <- V1 + V2 V1, V2 vecteur de taille n ?

Envisager une solution dans la réalisation de l'instruction d'addition vectorielle pour contourner cette difficulté.

Influence de la MV sur la performance des programmes

- Pages de 2KO soit 512 mots de 32b.
- A : array (1..512, 1..512) of integer ; stocké par lignes

cas1

```
for j=1 to 512
   for i=1 to 512
   A(I,J) = 0 ;
```

cas2

```
for i=1 to 512
   for j=1 to 512
   A(I,J) = 0 ;
```

Nombre de défauts de page dans chaque cas ?