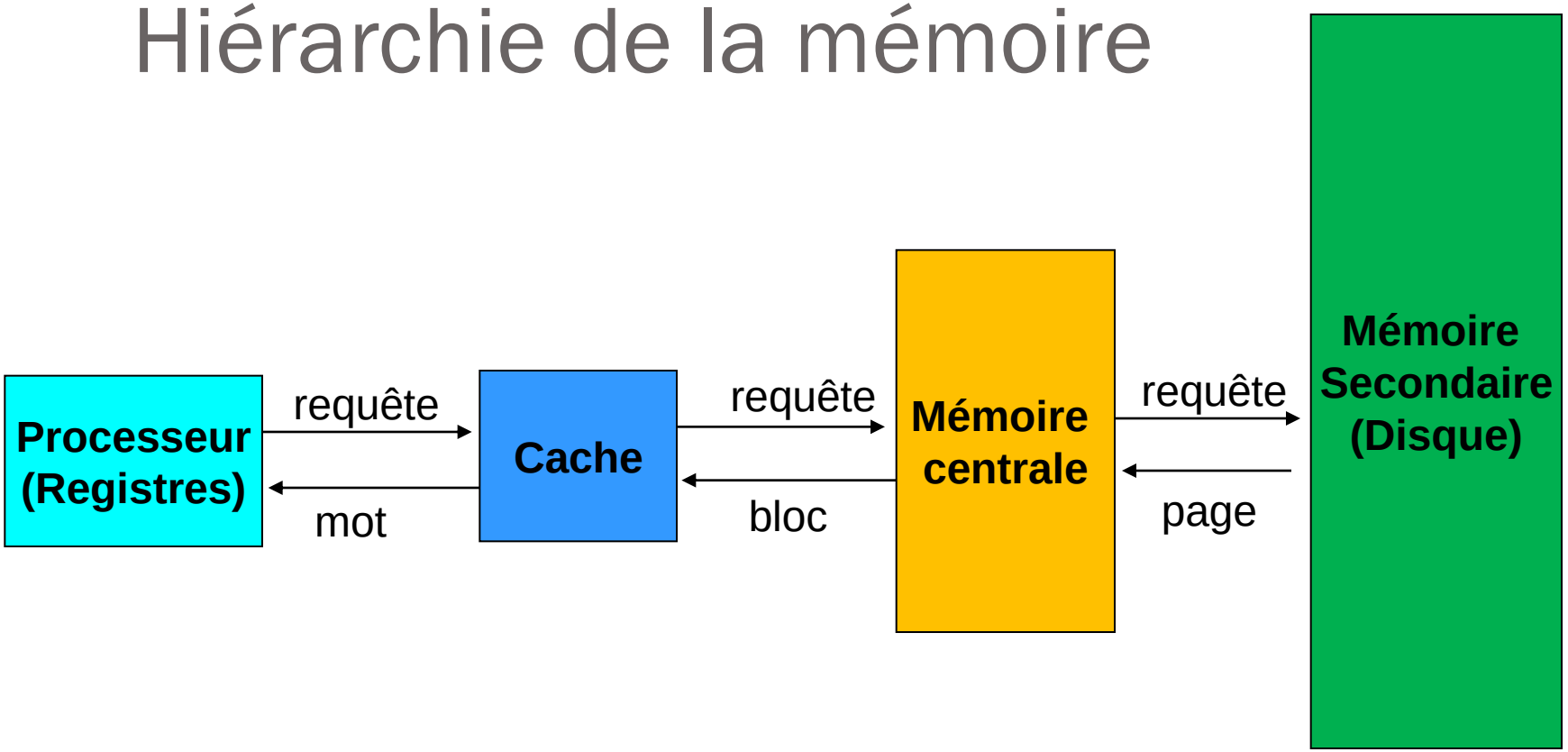
Mémoire et fichiers

Stefan Bornhofen

Loi de Parkinson

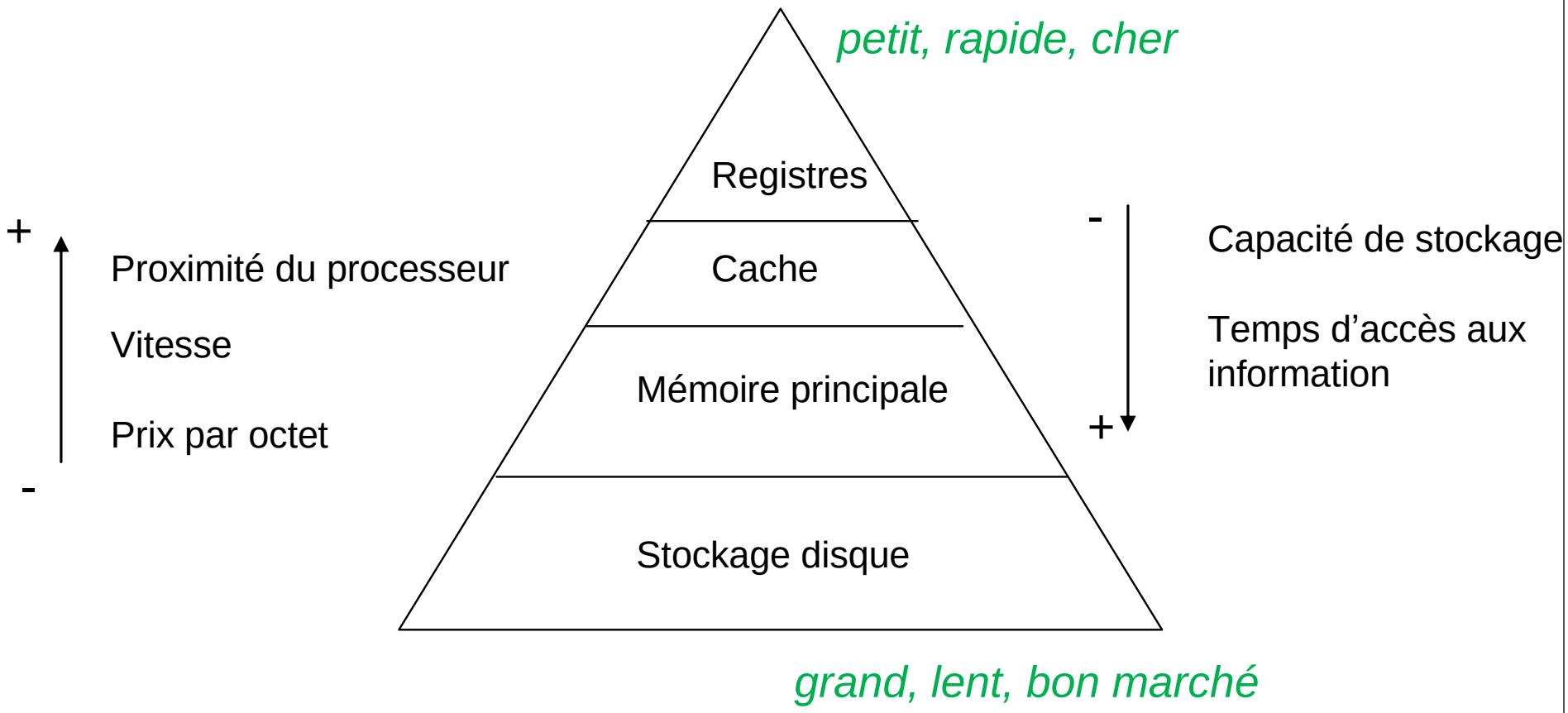
- *“Work expands so as to fill the time available for its completion.”*
- Les programmes s’acroissent pour remplir la mémoire disponible qui leur est réservée.

Hiérarchie de la mémoire



	Registre	Cache	Mémoire centrale	Disque
Capacité	<Ko	Mo	Go	>To
Temps d'accès (ns)	1-5	3-10	10-400	5 000 000
Coût relatif	50	10	1	0,001

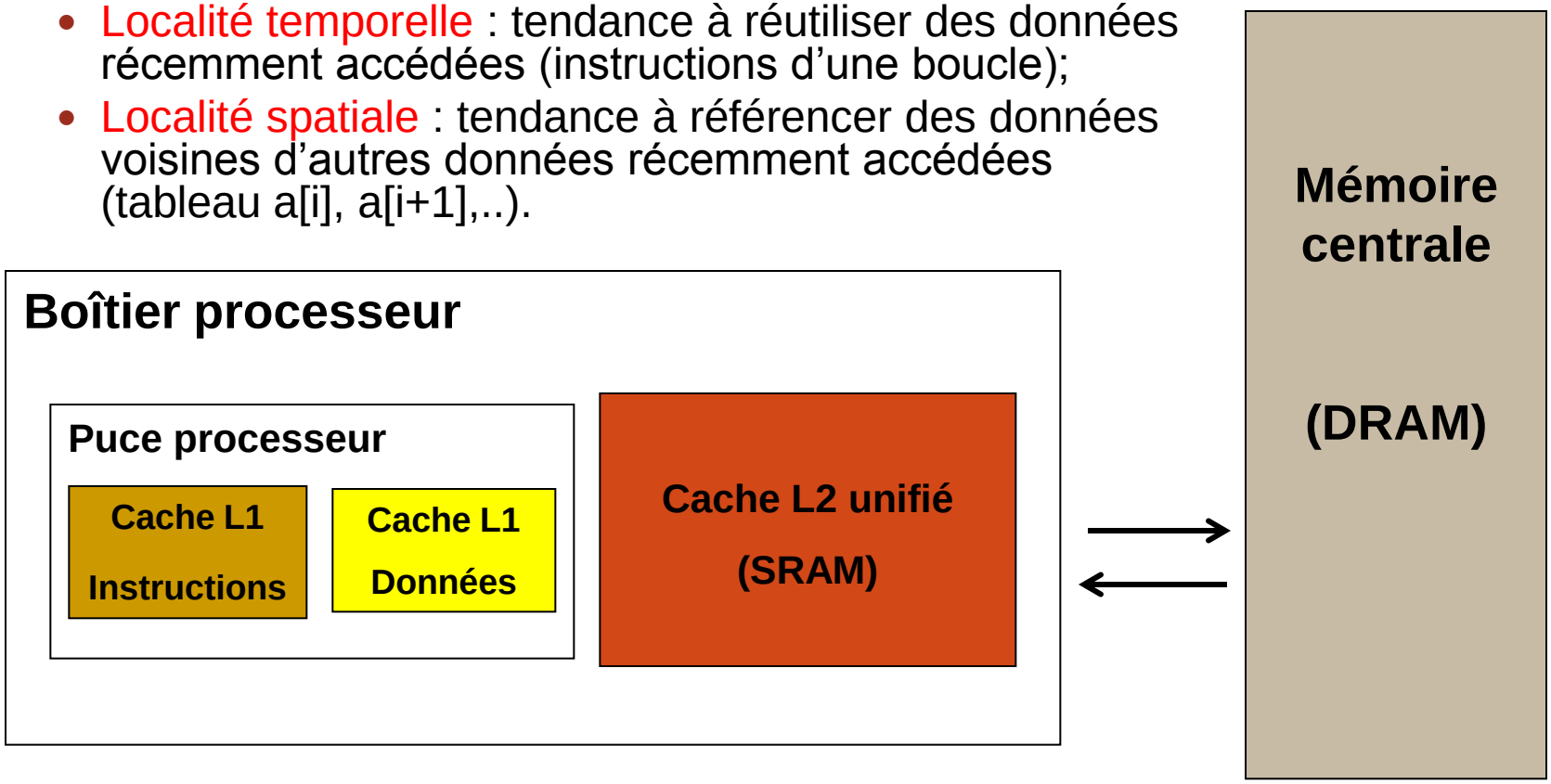
Hiérarchie de la mémoire



Cache

Eviter de rechercher en mémoire centrale des données déjà cherchées précédemment en les conservant près du processeur dans une petite mémoire à accès rapide.

- Géré par le matériel (hors SE)
- Principe de localité :
  - Localité temporelle : tendance à réutiliser des données récemment accédées (instructions d'une boucle);
  - Localité spatiale : tendance à référencer des données voisines d'autres données récemment accédées (tableau a[i], a[i+1],..).



Mémoire centrale

Missions du SE

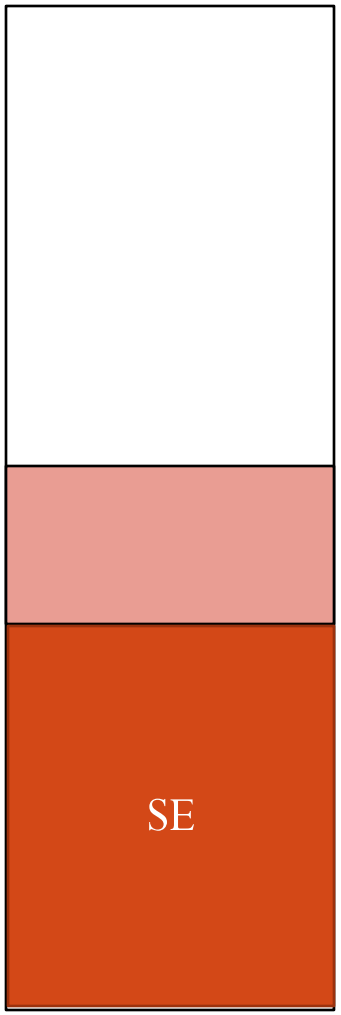
- Connaître les parties libres et occupées de la mémoire
- Allouer de la mémoire aux processus qui en ont besoin
- Libérer la mémoire d'un processus lorsque celui-ci se termine
- *Gérer le cas où la mémoire ne peut pas contenir tous les processus actifs*

Mémoire centrale

Seules les instructions stockées en mémoire centrale peuvent être exécutées par le CPU.

- En monoprogrammation
  - Mémoire réservée au SE
  - Mémoire réservée au (seul) programme à exécuter
  - Adressage direct possible

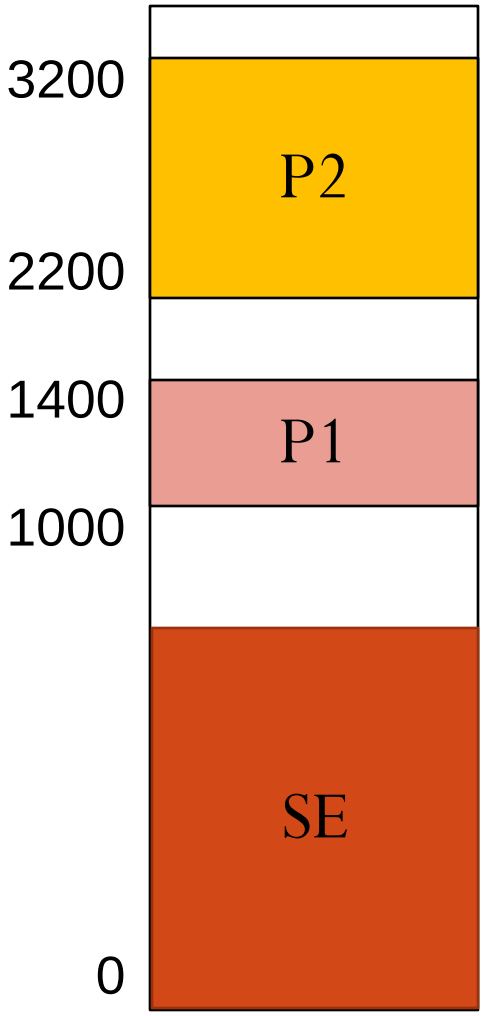
… et en multiprogrammation?



Espace d'adressage

- Base / limite
- Adresse physique = base + adresse
- Adresse maximale = base + limite

Processus	Base	Limite
P1	1000	400
P2	2200	1000

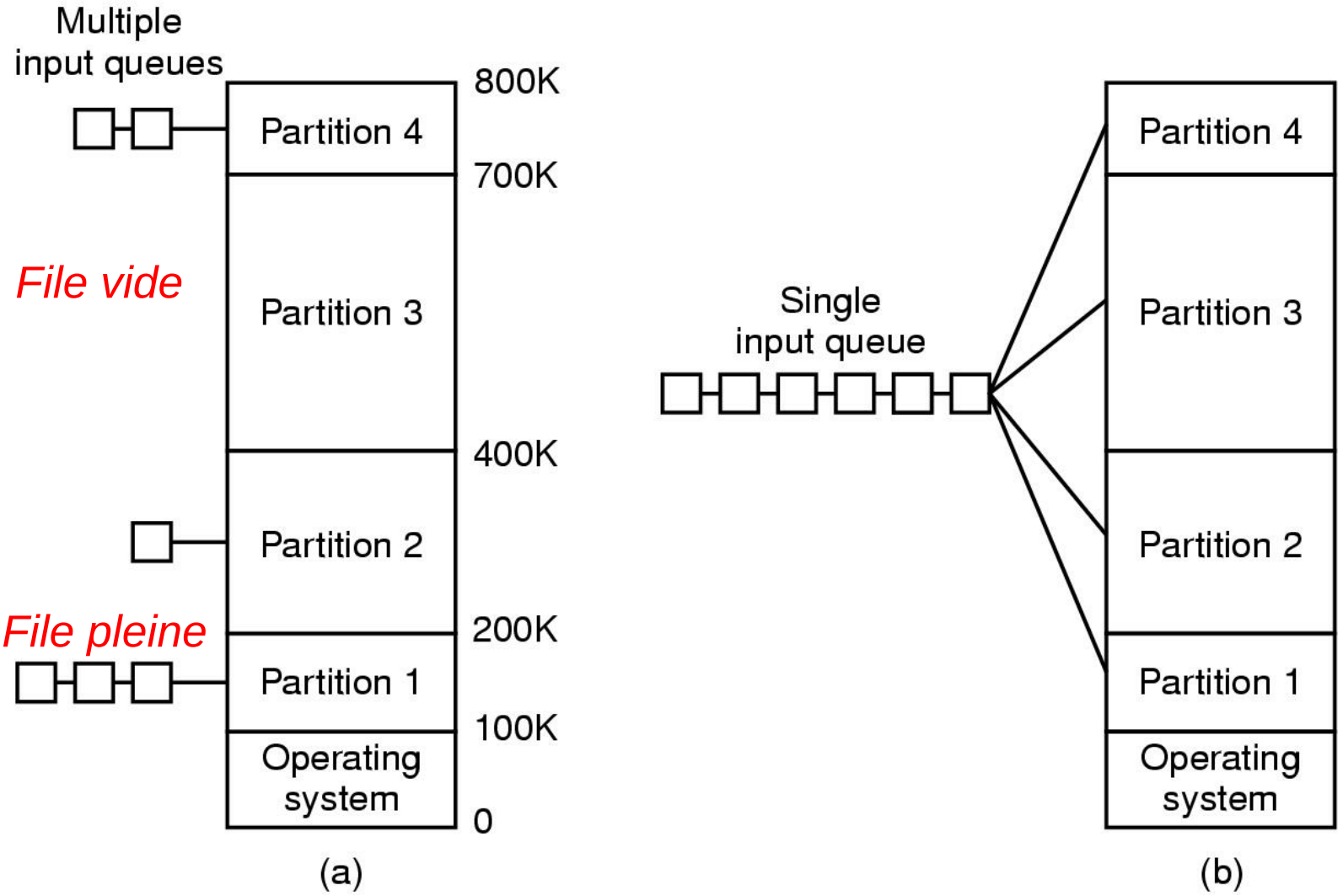


Partitions de taille fixe

Partitionner la mémoire pour contenir un nombre maximum de programmes

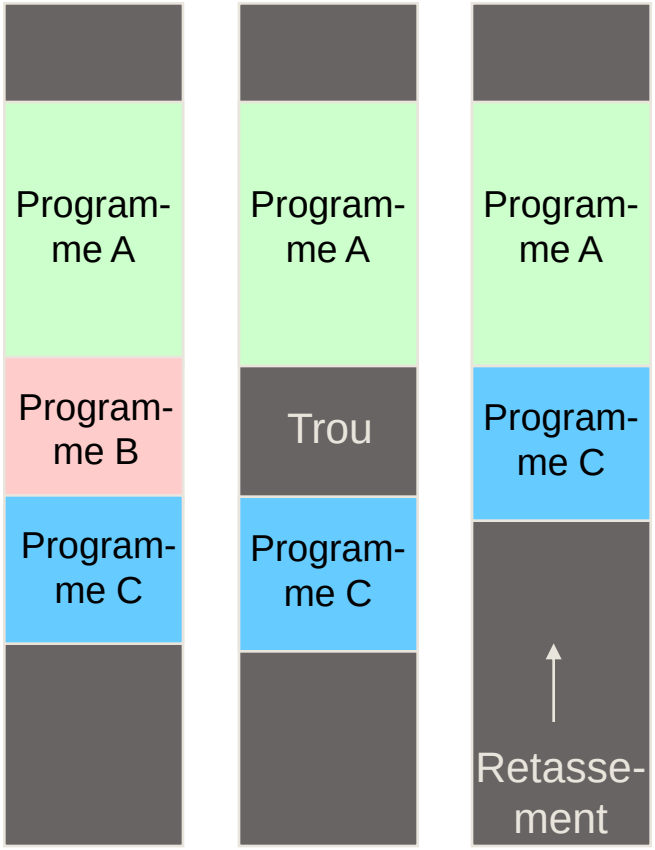
- Partitions de même grandeur? De différentes grandeurs?
- Un gros programme ne rentre que dans une grande partition
- Un petit programme rentabilise mal une grande partition
- Gaspillage de mémoire.
- Les files multiples présentent un inconvénient lorsque la file des grandes partitions est vide et celles des petites est pleine
- une alternative consiste à utiliser une seule file

Partitions de taille fixe



Partitions de taille variable

- Le nombre, la position et la taille des partitions varient dynamiquement
- Améliore l'usage de la mémoire
- mais complique sa gestion
- Lorsqu'un programme termine son exécution, il laisse un trou en mémoire
- Compactage de mémoire = relocaliser des programmes en cours d'exécution (« retassement »).



Multiprogrammation

Quand la mémoire est insuffisante pour maintenir tous les processus courants actifs.

- sauvegarder les processus sur le disque
- … et les recharger dynamiquement

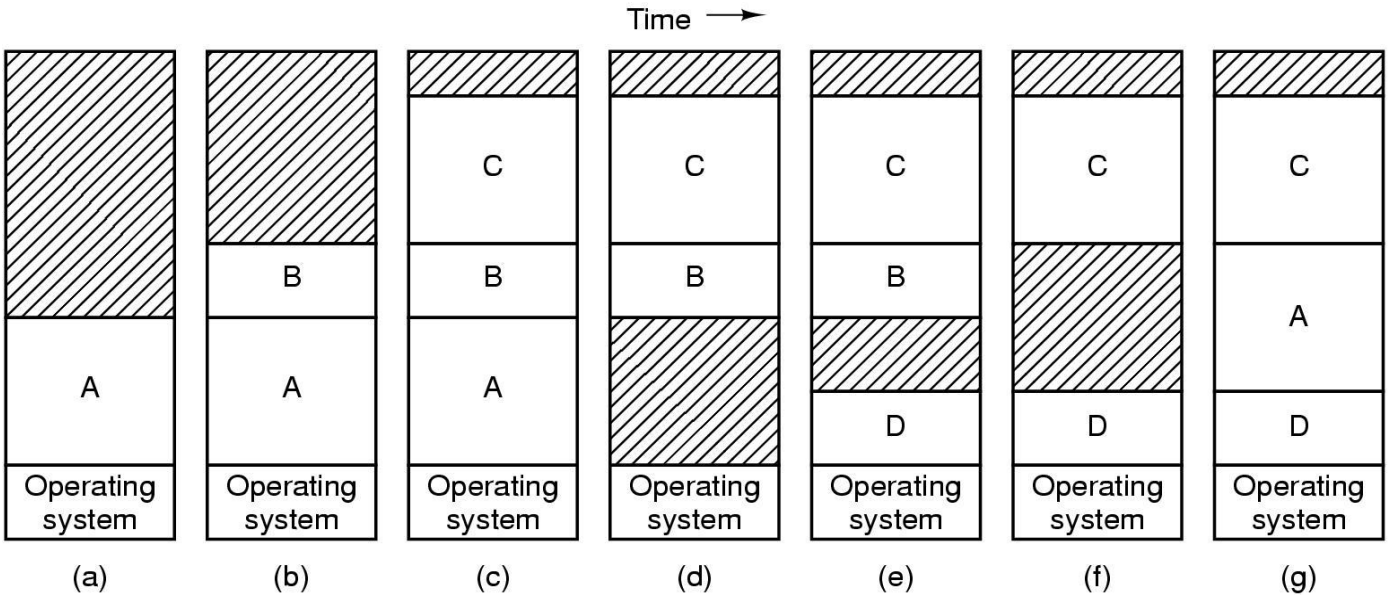
Deux approches

- Le va-et-vient (swapping) => sauvegarde du processus dans son intégralité
- La mémoire virtuelle => exécution du processus partiellement en mémoire

Le va-et-vient (*swapping*)

Lever la contrainte de dimension de la mémoire

- Elle considère chaque processus dans son intégralité:
  - Le processus est dans son intégralité en mémoire
  - ou est supprimé intégralement de la mémoire



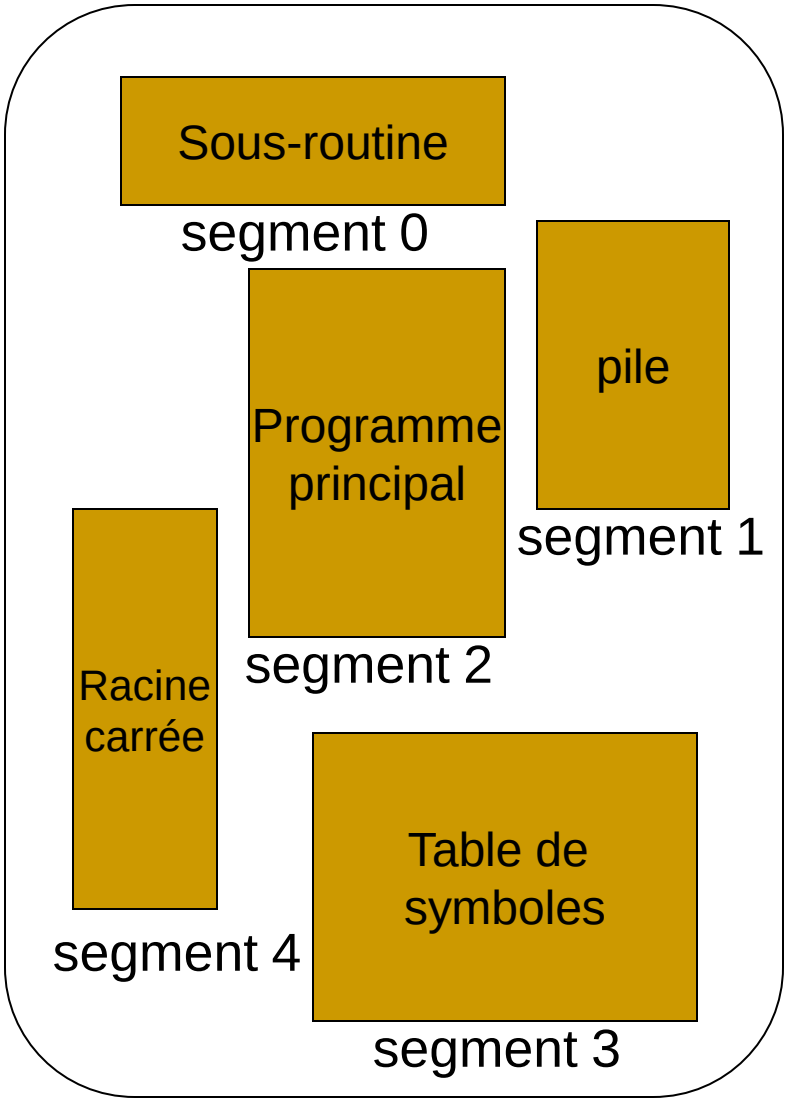
Segmentation

- Chaque processus peut avoir plusieurs segments (code, données, pile, etc.)
- Chaque segment commence à une nouvelle adresse de base dans la mémoire physique

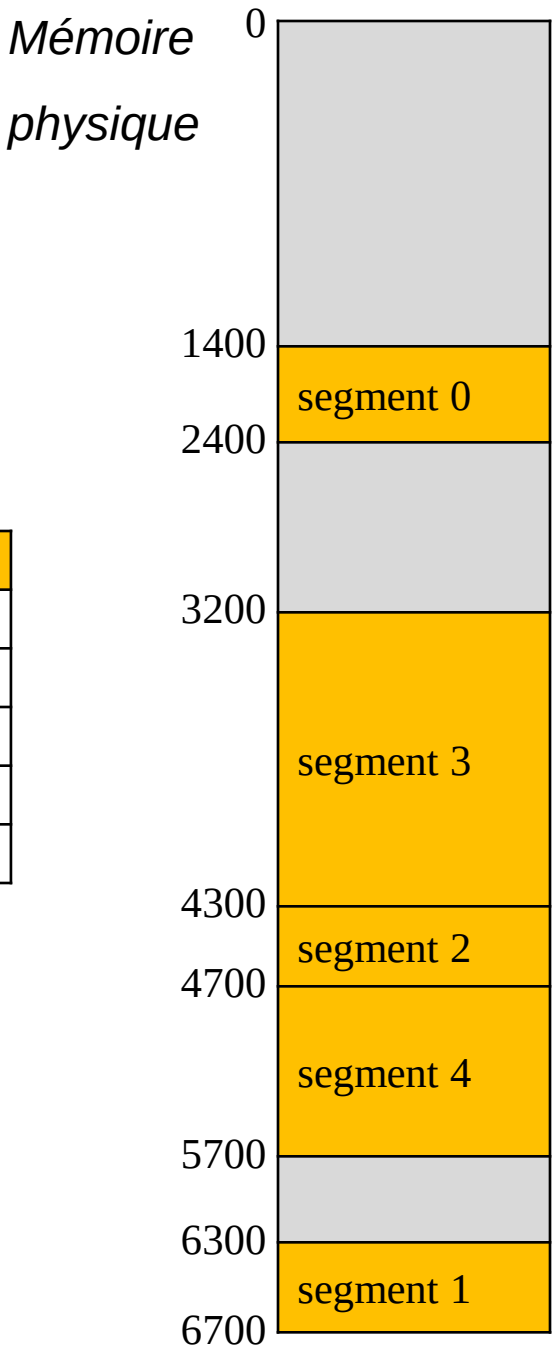
Avantages

- Les segments peuvent grandir indépendamment
- Les segments peuvent swapper indépendamment
- Les segments peuvent être partagés entre processus

Segmentation



	base	limite
0	1400	1000
1	6300	400
2	4300	400
3	3200	1100
4	4700	1000



Mémoire virtuelle

Lever la contrainte de dimension de la mémoire

- Pour exécuter un programme (ou un segment de programme), il n'est pas forcément nécessaire de le garder entièrement en mémoire
- Le SE conserve les parties (« pages ») en cours d'utilisation en mémoire et le reste sur disque.
- Quand un programme attend le chargement d'une partie de lui-même → il est en attente d'E/S

Pagination

Traiter séparément les adresses virtuelles référencées par le programme, et les adresses réelles de la mémoire physique.

- L'espace des adresses virtuelles est divisé en « pages » de taille fixe
- La mémoire physique est aussi divisée en pages de même taille
- Une adresse générée par un programme s'appelle une

adresse virtuelle = page virtuelle + offset.

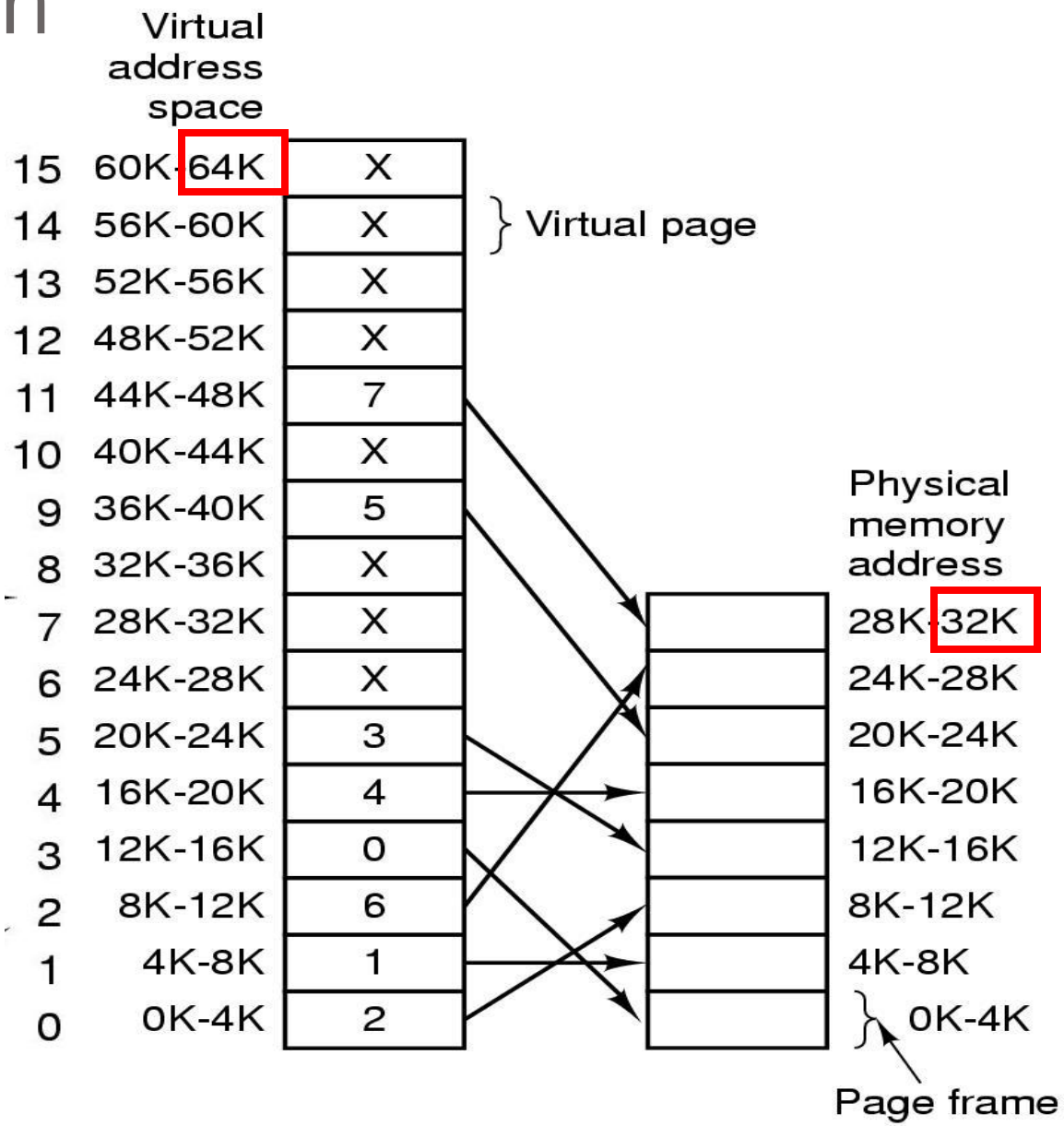
- Le SE maintient pour chaque programme une table de correspondance entre les pages virtuelles et les pages réelles.

adresse réelle= f(page virtuelle) + offset.

La table des pages est maintenue par le SE => lent

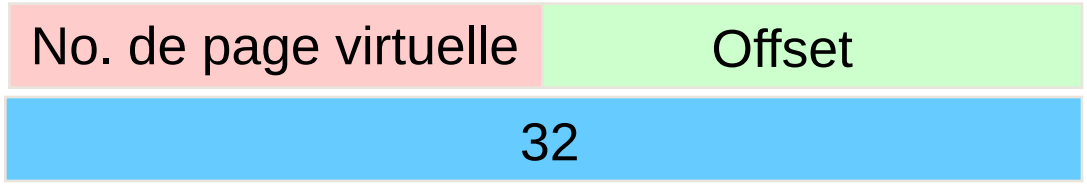
La **MMU** (Memory Management Unit) fait la correspondance rapide entre les adresses virtuelles et les adresses physiques, comme un cache.

Pagination


Virtual address space
15 60K-64K X
14 56K-60K X
13 52K-56K X
12 48K-52K X
11 44K-48K 7
10 40K-44K X
9 36K-40K 5
8 32K-36K X
7 28K-32K X
6 24K-28K X
5 20K-24K 3
4 16K-20K 4
3 12K-16K 0
2 8K-12K 6
1 4K-8K 1
0 0K-4K 2
Virtual page
Physical memory address
28K-32K
24K-28K
20K-24K
16K-20K
12K-16K
8K-12K
4K-8K
0K-4K
Page frame

Adresse virtuelle

- L'adresse virtuelle est scindée en deux champs : Les derniers bits définissent un offset (adresse dans la page), le reste définit le numéro de page virtuelle.



Adresse virtuelle

Exemple

Adresses virtuelles de 32 bits et pages de 2 Ko

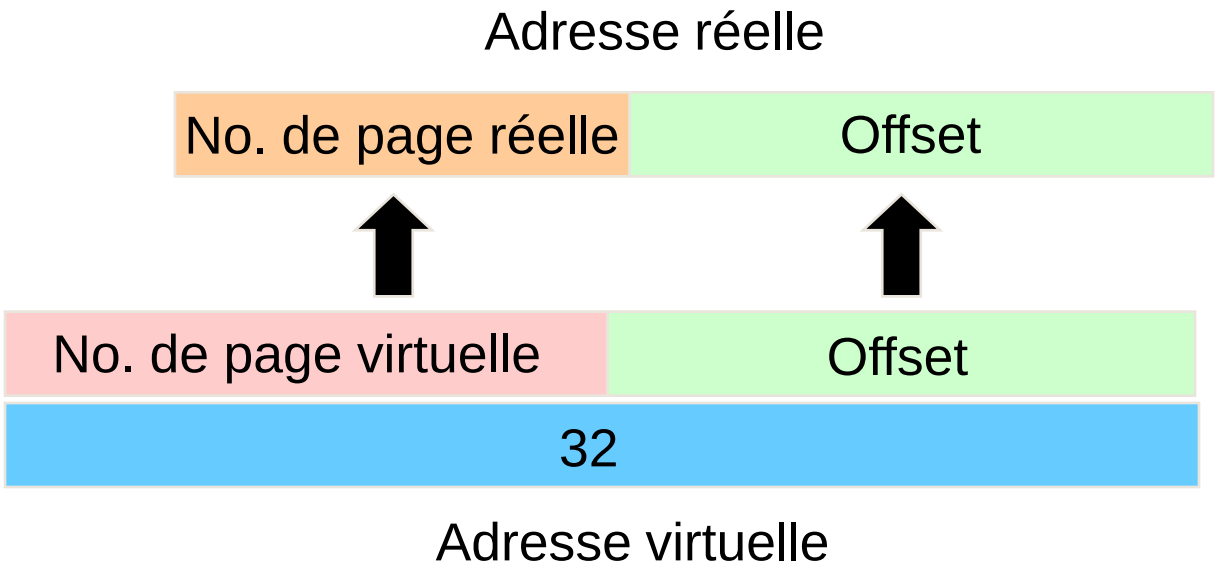
⇒ L'offset aura 11 bits (2^{11} = 2 K)

⇒ Le numéro de page virtuelle aura 32 - 11 = 21 bits.

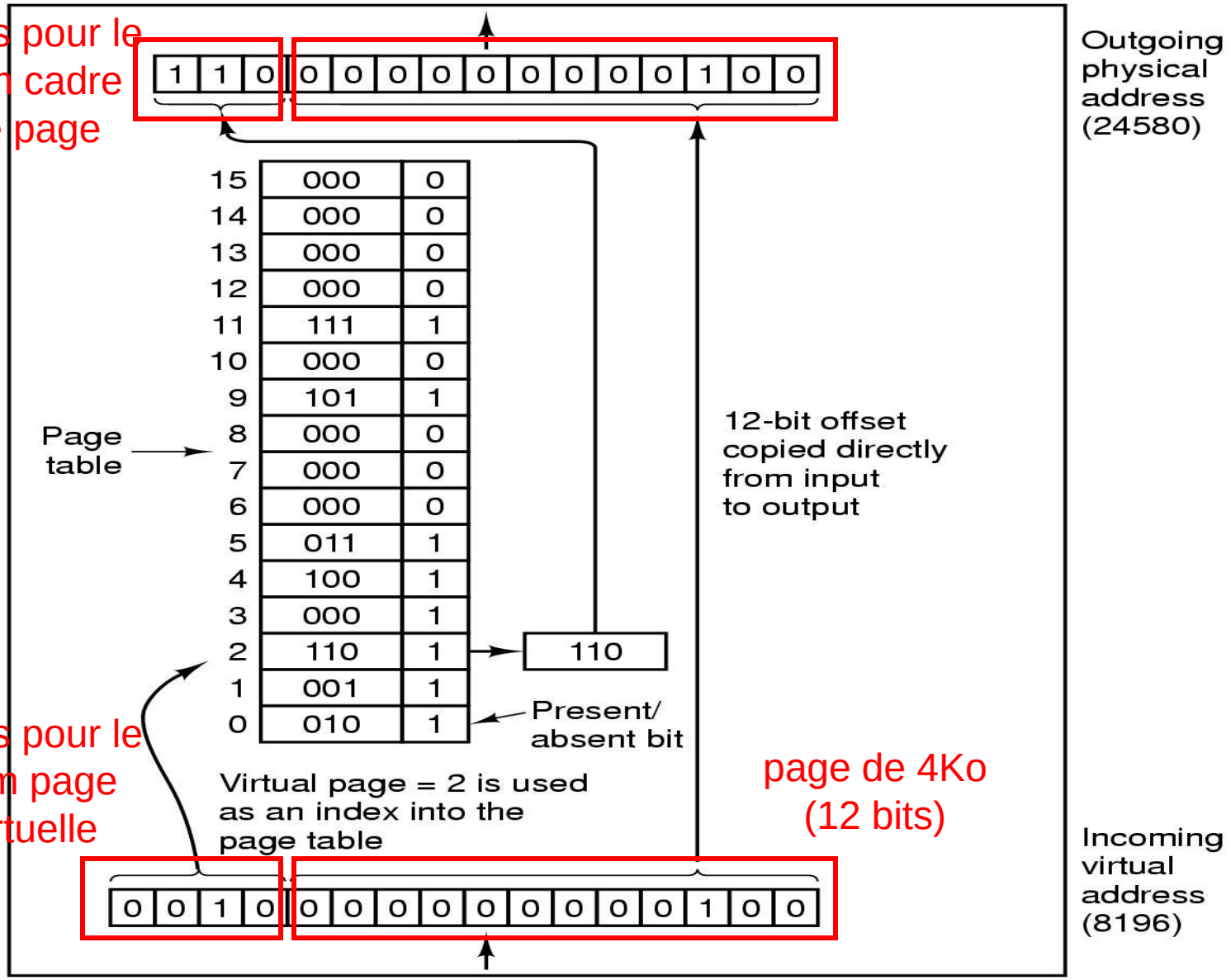
⇒ L'espace d'adresses virtuelle est découpé en 2 M pages de 2 Ko

Traduction

Traduction d ’une adresse virtuelle en adresse réelle



Traduction


3 bits pour le num cadre de page
1 1 0 0 0 0 0 0 0 0 0 0 1 0 0
Outgoing physical address (24580)
15 000 0
14 000 0
13 000 0
12 000 0
11 111 1
10 000 0
9 101 1
8 000 0
7 000 0
6 000 0
5 011 1
4 100 1
3 000 1
2 110 1
1 001 1
0 010 1
Page table
110
Present/ absent bit
12-bit offset copied directly from input to output
4 bits pour le num page virtuelle
Virtual page = 2 is used as an index into the page table
page de 4Ko (12 bits)
0 0 1 0 0 0 0 0 0 0 0 0 0 1 0 0
Incoming virtual address (8196)

Défaut de page

- Si le bit V (valid-bit) est 1, c'est que la page est en mémoire réelle. Il remplace alors le numéro de page virtuelle par le numéro de page réelle qui se trouve dans la rangée en question.
- Si le bit V (valid-bit) est 0 (« défaut de page »), il lit l'adresse disque de la page en question, charge la page en mémoire et inscrit dans la rangée correspondante de la table des pages le numéro de page réelle où la page a été placée. Il met ensuite le bit V à 1.

La victime

Algorithmes de remplacement de page

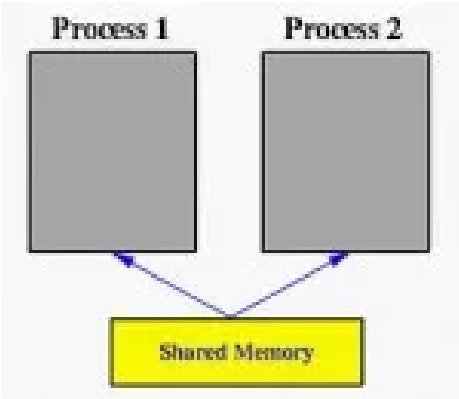
- *Optimal* (évincer la page qui sera appelée le plus tard possible dans l'avenir => impossible à savoir à l'avance)
- FIFO (first in first out)
- LRU (least recently used)
- ...

La victime doit-elle être récrite sur le disque?

- 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 modification (*dirty bit*) sur chaque descripteur de page indique si la page a été changée

Mémoire partagée

- Méthode de communication inter-processus
- La mémoire partagée permet à plusieurs processus d'accéder à la même zone mémoire
- Partager des pages mémoire
- Lorsqu'un processus modifie une telle page, tous les autres processus voient la modification.



Mémoire partagée

- shmget() crée ou récupère le segment de mémoire partagée
- shmctl() permet d'intervenir sur ce segment (modification des droits d'écriture, suppression, …)
- shmat() permet à un processus de s'attacher ce segment
- shmdt() permet à un processus de se détacher du segment.

```c
#include <stdio.h>
#include <sys/shm.h>
#include <sys/stat.h>

int main ()
{
  int segment_id;
  char* shared_memory;
  const int shared_segment_size = 0x6400;
  int key = 5678; // any nice integer

  // Alloue le segment de mémoire partagée.
  segment_id = shmget (key, shared_segment_size, IPC_CREAT | 0666);

  // Attache le segment de mémoire partagée.
  shared_memory = (char*) shmat (segment_id, 0, 0);
  printf ("mémoire partagée attachée à l'adresse %p\n", shared_memory);
  // Écrit une chaîne dans le segment de mémoire partagée.
  sprintf (shared_memory, "Hello, world.");
  // Détache le segment de mémoire partagée.
  shmdt (shared_memory);

  // Libère le segment de mémoire partagée.
  shmctl (segment_id, IPC_RMID, 0);

  return 0;
}
```

Utiliser « ipcs –m » pour afficher la mémoire partagée