

# Gestion mémoire (partie 1)

*L3 Informatique – Systèmes d'Exploitation*

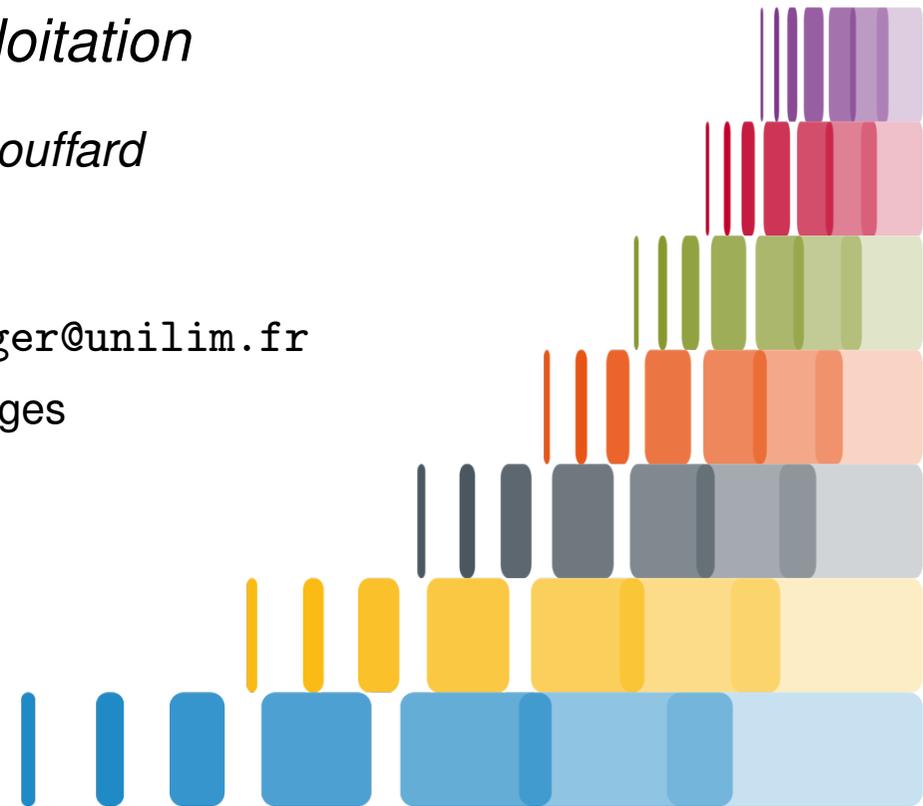
*Sources: Jean-Louis Lanet & Guillaume Bouffard*

Vincent Neiger

`vincent.neiger@unilim.fr`

Département Informatique – Université de Limoges

14 février 2018



Université  
de Limoges

# Plan

- Besoin de gestion mémoire
- Pagination
- Segmentation
- Problèmes de sécurité



# Objectifs

- Après cette présentation vous devriez être capable de
  - Comprendre les fondements de la gestion mémoire,
  - Raisonner sur les besoins et techniques de partitionnement de la mémoire,
  - Expliquer les concepts de pagination et de segmentation ainsi que leurs avantages comparés,
  - Connaitre les problèmes de sécurité induits par la gestion mémoire.



# Problème de l'allocation

- L'allocation mémoire doit permettre à un processus d'accéder à un objet défini en mémoire virtuelle (prochain cours),
- Une politique d'allocation mémoire doit résoudre:
  - La correspondance entre adresses virtuelles et adresses physique,
  - Gérer la mémoire physique,
  - Réaliser le partage d'information entre les usagers,
  - Assurer la protection mutuelle d'informations appartenant à des usages distincts.



# Et la gestion mémoire...

- Lorsque cette mémoire a été allouée à un processus il doit la gérer,
  - Créer des objets / détruire des objets,
  - Gérer les appels à des sous programmes
  - Gérer la récursivité
  - ...

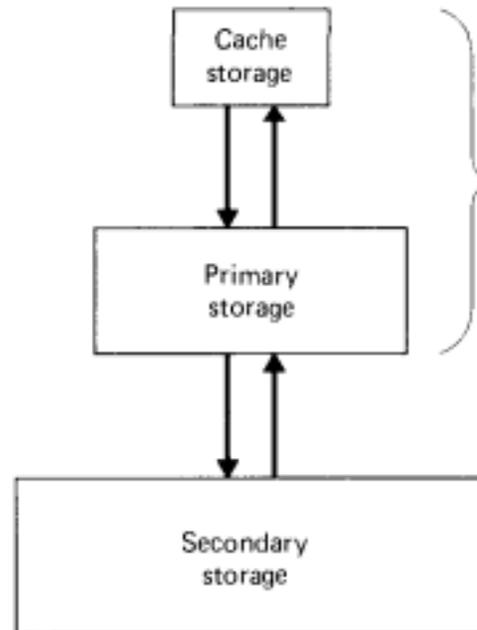


# Pourquoi gérer la mémoire ?

Temps d'accès très court  
Capacité très faible



Temps d'accès très long  
Capacité très important



Programme et données directement  
référéncés par le CPU

Programme et données doivent  
d'abord être transférés en mémoire  
principale avant tout accès par le CPU



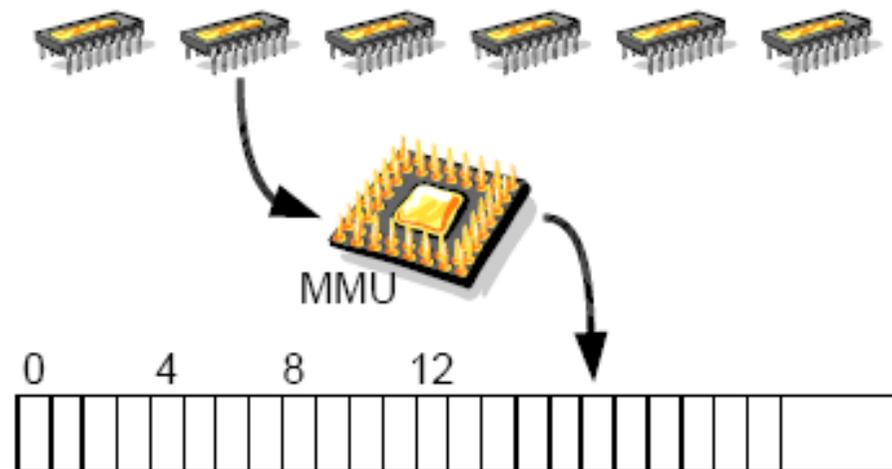
# Mémoire/Adresses physiques et logiques

- Mémoire physique:
  - la mémoire principale RAM de la machine
- Adresses physiques: les adresses de cette mémoire,
- Mémoire logique: l'espace d'adressage d'un programme,
- Adresses logiques: les adresses dans cet espace,
- Il faut séparer ces concepts car normalement, les programmes sont chargés parfois dans des positions différentes de mémoire,
  - Donc adresse physique  $\neq$  adresse logique



# Représentation de la mémoire

- Mémoire: des puces (matériel)
- Vue par le système (OS/application) via des adresses (logiciel)



# Représentation de la mémoire

- Manipulation des adresses à la main (ASM):

```
MOV 47,#0xFBBFC
```

```
MOV 74,#0xFBFBC
```

```
ADD 3,#0xFBFBC
```

```
INC#0xFBBFC
```

```
SUB #0XFBBFC,#0xFBFBC
```

– Peu clair...



# Représentation de la mémoire

- Manipulation des adresses à la main (ASM)
  - Permet de mettre des données en un lieu précis de la mémoire
  - Complicé si on veut faire cohabiter plusieurs applications
  - Lisibilité et maintenance pitoyables



# Représentation de la mémoire

- Manipulation symbolique **explicite** des adresses : variables et pointeurs

```
int *a = 0xFBBFC
```

```
int *b = 0xFBFBFC
```

```
*a = 47
```

```
*b = 74
```

```
*b = *b+3
```

```
*a = *a+1
```

```
*b = *b-*a
```

– C'est mieux: plus haut niveau, plus clair



# Représentation de la mémoire

- Manipulation symbolique **implicite** des adresses: variables et références

```
int a = 47
```

```
int b = 74
```

```
b += 3
```

```
a++
```

```
b -= a
```



# Représentation de la mémoire

- Manipulation symbolique implicite des adresses: variables et références
  - Encore plus haut niveau, plus clair
  - Localisation (adresse) masquée (par le système qui gère la mémoire)
    - Facile d'avoir plusieurs programmes (multi-tâche, multi processus)



# Programme

- Objets d'un programme
  - fonctions, variables, constantes désignés par un nom symbolique unique
  - taille des objets calculée par compilateur
  - éditeur de liens attribue une position absolue (adresse) ou relative des objets dans espace d'adressage
- Taille et adresses des instructions et des données stockées dans programme binaire



# Rôle de la mémoire

- Stocker instructions et données des programmes exécutés par le processeur,
- Repérer les objets par leur adresse dans la mémoire
- Espace d'adressage
  - Ensemble des unités mémoire (octets) individuellement adressables depuis le CPU
  - Mécanisme(s) de construction de l'adresse de chaque unité de l'espace



# Rôle de la mémoire

- Les opérations mémoires sont plus lentes que les instructions des processeurs.
- La sécurité passe par la mémoire
  - Au minimum : zone système et zone utilisateur.
  - Si possible, protéger des zones plus fines parmi les zones utilisateurs.
- La modularité passe par la mémoire
  - Avoir plusieurs programmes chargés en parallèle.
  - Partageant du code.
- Besoin de support matériel
  - Pour pouvoir faire les vérifications de sécurité efficacement.
  - Pour permettre et faciliter le partage de code.



# Contrôle d'accès

- Pour la sûreté : avoir des zones en lecture seule pour l'utilisateur.
  - La zone système.
  - Les zones des autres utilisateurs.
- Pour la sécurité (secret) : contrôler également la lecture.
- Pour le confort : l'utilisateur peut lui-même protéger certaines zones en écritures.
- Pour l'efficacité : copie à l'écriture.
  - On interdit une zone en écriture.
  - On rattrape l'interruption de violation de droit pour sauvegarder la zone, puis autoriser l'écriture et reprendre l'exécution.



# Autres opérations

- Programme partiellement chargé en mémoire
  - Si le programme ne loge pas tout entier en mémoire centrale.
  - (La mémoire disque est plus grande que la mémoire centrale).
- Plusieurs programmes chargés en mémoire
  - Peuvent-ils partager un même sous-programme ?
  - Nécessite du code réentrant : Comment le permettre/favoriser ?
- Code relogeable
  - On veut pouvoir sauver un programme en train de s'exécuter sur le disque (par manque de place), puis plus tard le recharger en mémoire, éventuellement à un autre emplacement.



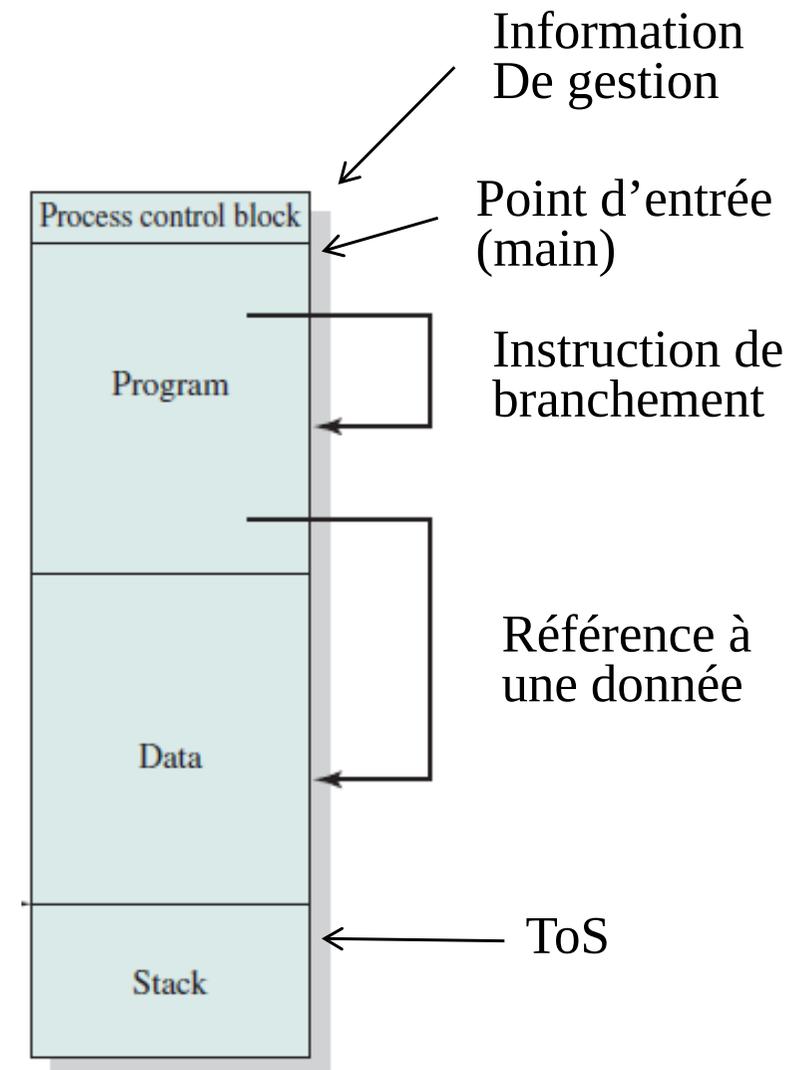
# Liaison d'adresses logiques et physiques (instructions et données)

- La liaison des adresses logiques aux adresses physiques peut être effectuée en moments différents:
  - **Compilation** : quand l'adresse physique est connue au moment de la compilation (rare)
    - p.ex. parties du SE
  - **Chargement** : quand l'adresse physique où le programme est chargé est connue, les adresses logiques peuvent être traduites (rare aujourd'hui)
  - **Exécution** : normalement, les adresses physiques ne sont connues qu'au moment de l'exécution
    - p.ex. allocation dynamique, swap : le programme peut être relogé à des adresses différentes,



# Problème d'adressage d'un processus

- Le système a besoin de connaître:
  - Le point d'entrée,
  - La pile correspondant à ce processus,
- Il doit inférer,
  - L'adresse réel du programme,
  - Les différents sauts,
  - Les adresses des données,



# Problème d'adressage d'un processus : hardware

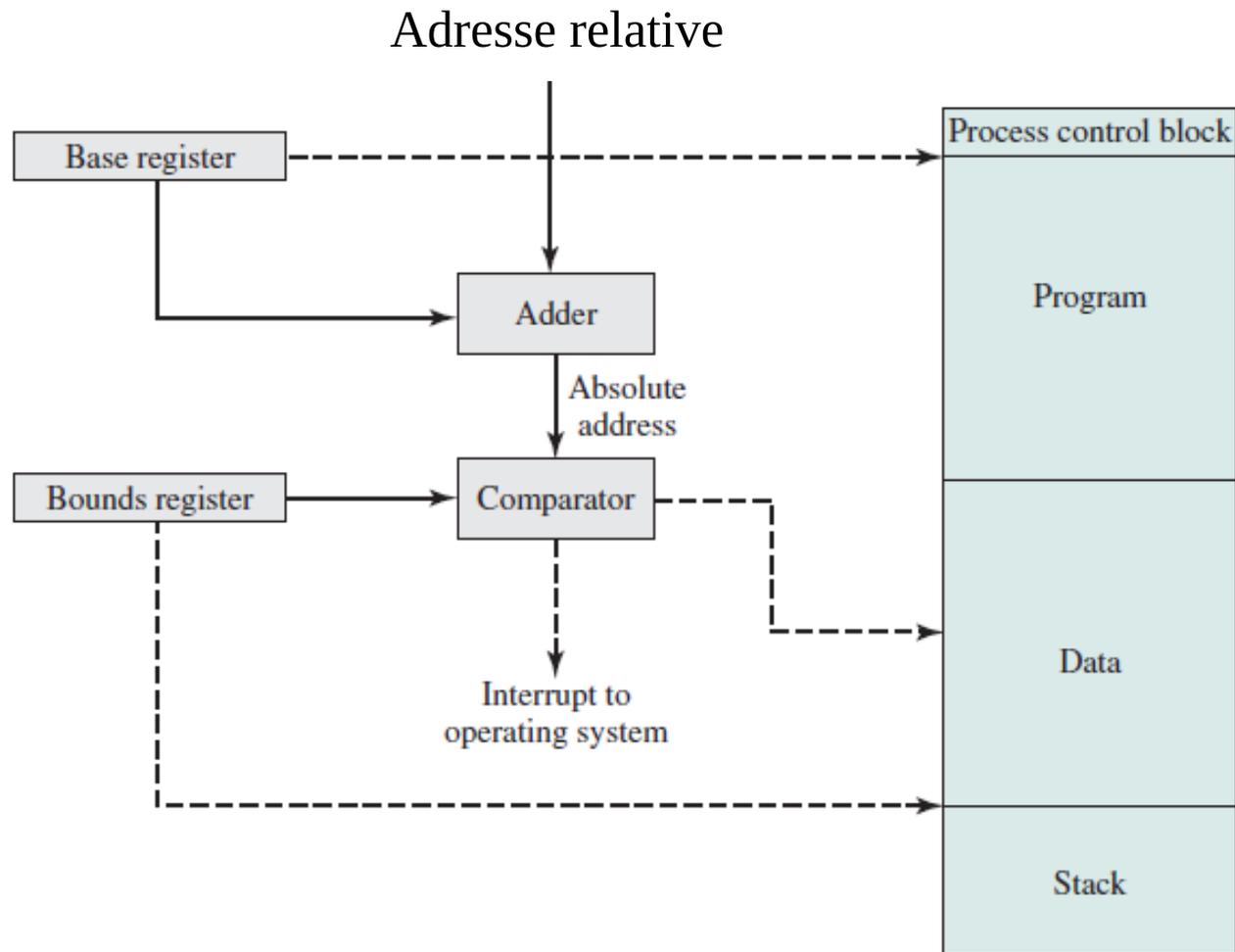


Image du processus en mémoire

# Code relogeable

- On veut pouvoir déplacer le code une fois chargé en mémoire
- Il suffit que le compilateur référence les adresses du code par décalage par rapport à un registre de base (par exemple le début de la zone utilisateur).
- Permet un calcul dynamique des adresses réelles du code.
- Peut aussi se faire au niveau logiciel (génération de code) sans support matériel.
- L'adresse  $p + k$  est un saut relatif de  $d$  par rapport à  $pc$  ou de  $k$  par rapport à la *base* qui vaut ici  $p$  pendant l'exécution.



# Plan

- Besoin de gestion mémoire
- Pagination
- Segmentation
- Problèmes de sécurité



# Et pour quelques termes de plus...

- Frame : la mémoire vive est composée de zones de même taille, appelées *cadres* (*frames* en anglais).
- Pages : un bloc de donnée de longueur fixe résidant en mémoire secondaire. Une page peut être copiée temporairement dans une frame,
- Segment : un bloc de donnée de longueur variable résidant en mémoire secondaire. Un segment peut être copié temporairement en mémoire principale.



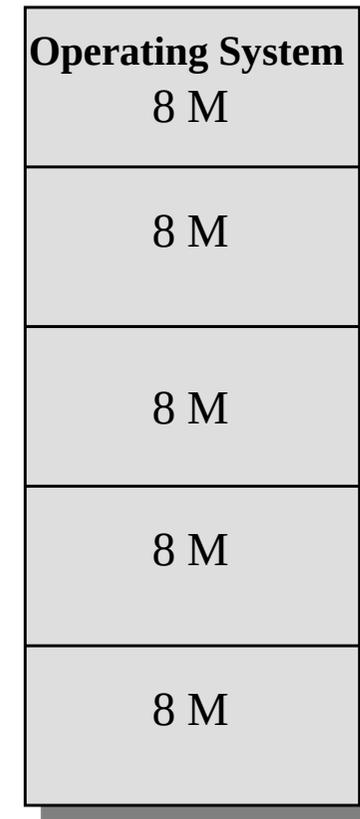
# Techniques de gestion mémoire

- Partitionnement à taille fixe
  - Divise la mémoire en partitions lors du boot, les tailles peuvent être identiques ou pas mais fixes,
  - Mécanisme simple souffrant de la fragmentation interne
- Partitionnement à taille variable
  - Les partitions sont créées lors du chargement des programmes
  - Ne crée pas de fragmentation interne mais externe,
- Pagination simple
  - Divise la mémoire en pages de taille fixes et charge les programmes dans les pages disponibles
  - Pas de fragmentation externe, mais légère fragmentation interne



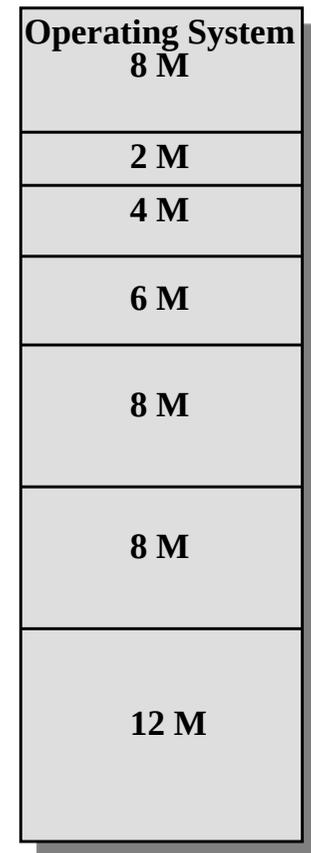
# Partitionnement à taille fixe

- Main memory divided into static partitions
- Simple to implement
- Inefficient use of memory
  - Small programs use entire partition
  - Maximum active processes fixed
  - Internal Fragmentation



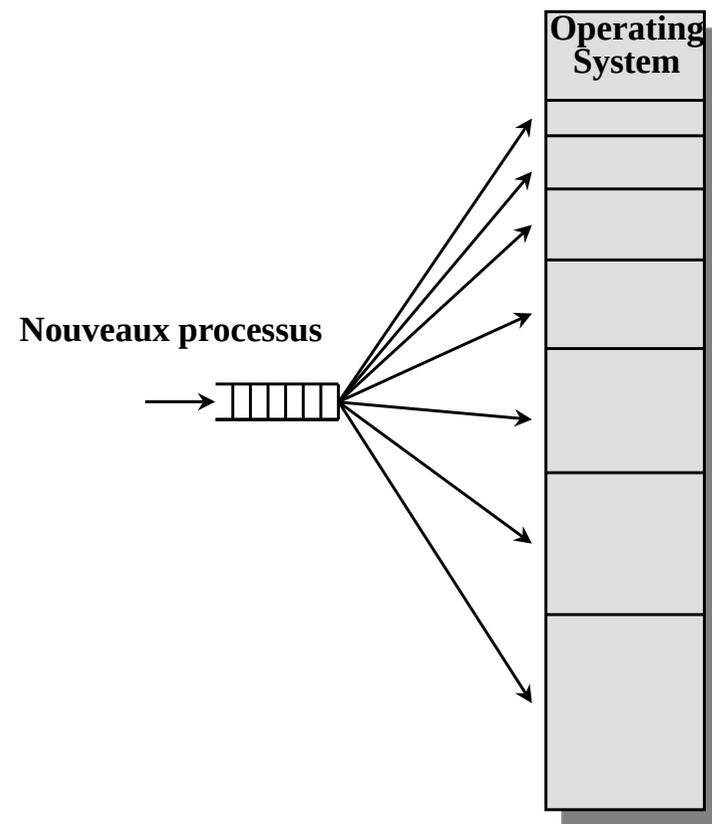
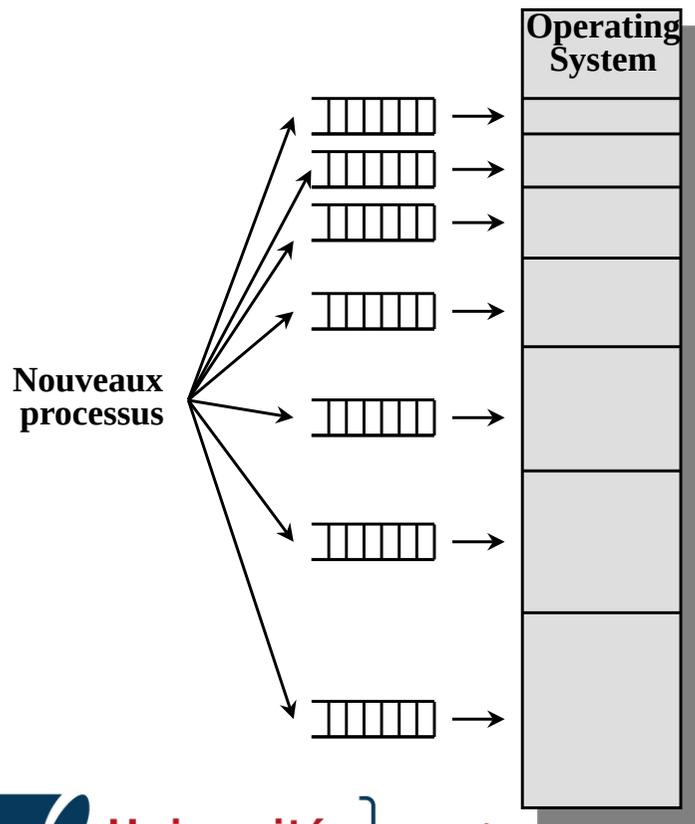
# Partitionnement à taille fixe

- Différentes tailles de partitions
  - Les petits programmes sont alloués aux petites partitions
  - Le problème de la fragmentation reste.
- Algorithme de placement
  - Utiliser la plus petite partition possible,
  - Une file d'attente par partition,
  - Inefficacité si les grosses partitions ne sont pas requises.



# File d'attente

- Soit une file d'attente par partition soit une file d'attente et allocation First Fit.



# Techniques de gestion mémoire

- Partitionnement à taille fixe
  - Divise la mémoire en partitions lors du boot, les tailles peuvent être identiques ou pas mais fixes,
  - Mécanisme simple souffrant de la fragmentation interne
- Partitionnement à taille variable
  - Les partitions sont créées lors du chargement des programmes
  - Ne crée pas de fragmentation interne mais externe,
- Pagination simple
  - Divise la mémoire en pages de taille fixes et charge les programmes dans les pages disponibles
  - Pas de fragmentation externe, mais légère fragmentation interne

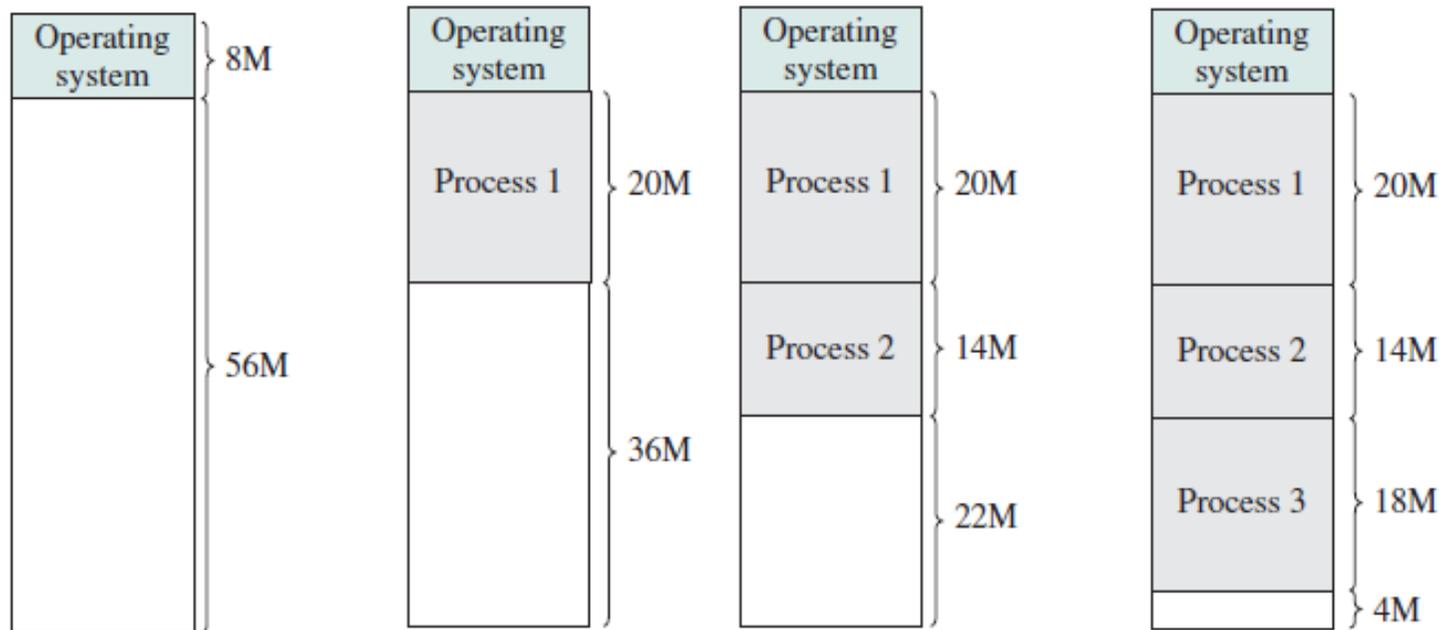


# Partitionnement dynamique

- Les partitions sont de longueurs variables et leur nombre n'est pas fixe,
- Un processus reçoit exactement la mémoire dont il a besoin,
- Éventuellement ceci crée des trous dans la mémoire (espace alloué puis dés-alloué) : fragmentation externe.



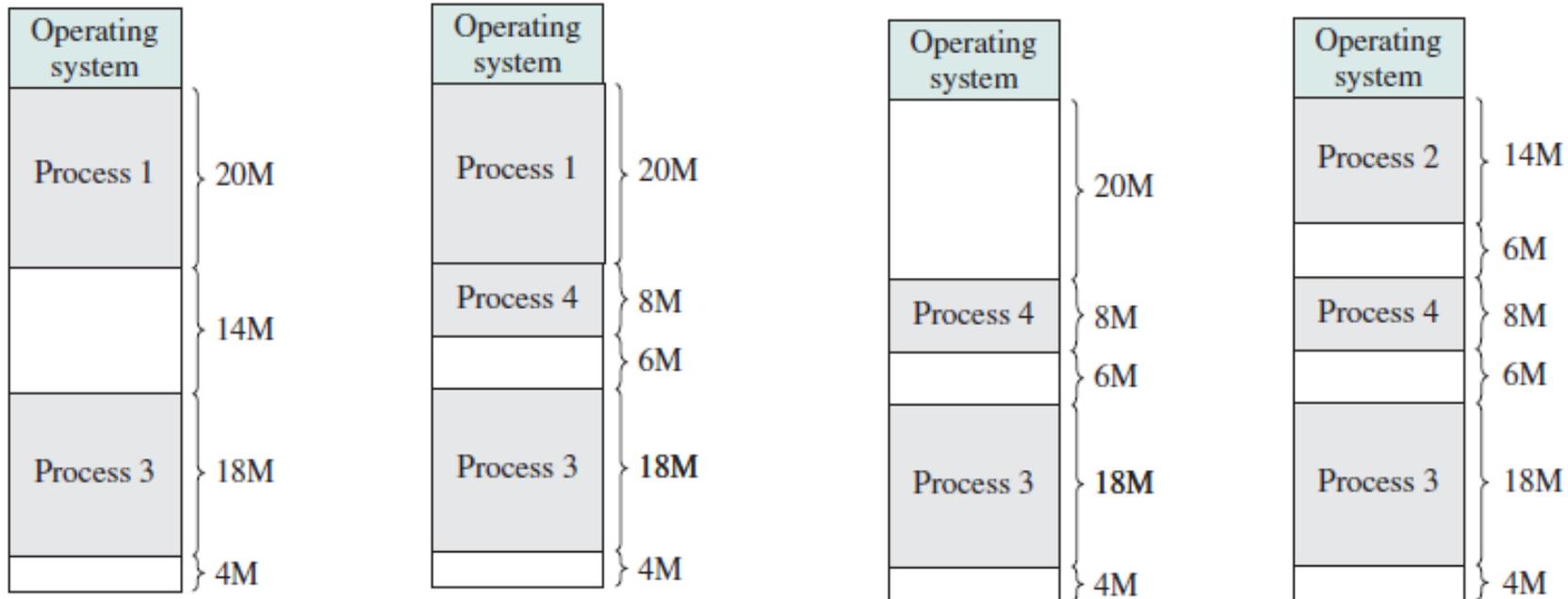
# Fragmentation



Quatre processus doivent être chargés : 20, 14, 18 et 8 Mo



# Fragmentation



# Fragmentation mémoire

- Les statistiques montrent que pour N blocs alloués,  $0.5*N$  blocs sont perdus par fragmentation;
- Solution le compactage.
  - Le CPU déplace les processus de manière à les rendre contigus.
  - La mémoire est relâchée par bloc.
  - L'algorithme de compactage doit être exécuté de temps en temps
    - Fréquence ?
    - A quel moment ?

# Fragmentation

- Réunir 2 blocs libres adjacents
  - Minimiser fragmentation
- Immédiatement, lors de la libération d'un bloc
  - Retrouver éventuel bloc libre adjacent du bloc libéré
  - Liste des blocs libres par adresses croissantes
  - Pénalise opération de libération
- Plus tard
  - À partir d'un seuil minimum de blocs libres
  - Lorsque allocation non satisfaite immédiatement



# Algorithme d'allocation

- Allocation de régions de tailles variables
  - Espace libre = ensemble de blocs de  $\neq$  tailles
- Taille bloc libre  $>$  taille demandée
  - Reste bloc libre de plus petite taille
- Fragmentation externe
  - Somme taille des zones libres  $>$  taille demandée
  - Taille de chaque zone libre  $<$  taille demandée



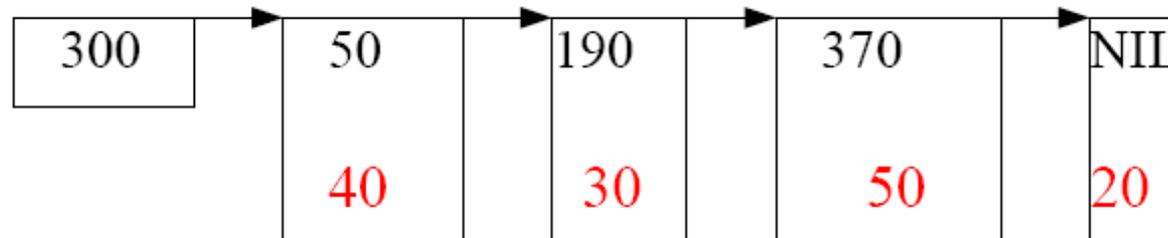
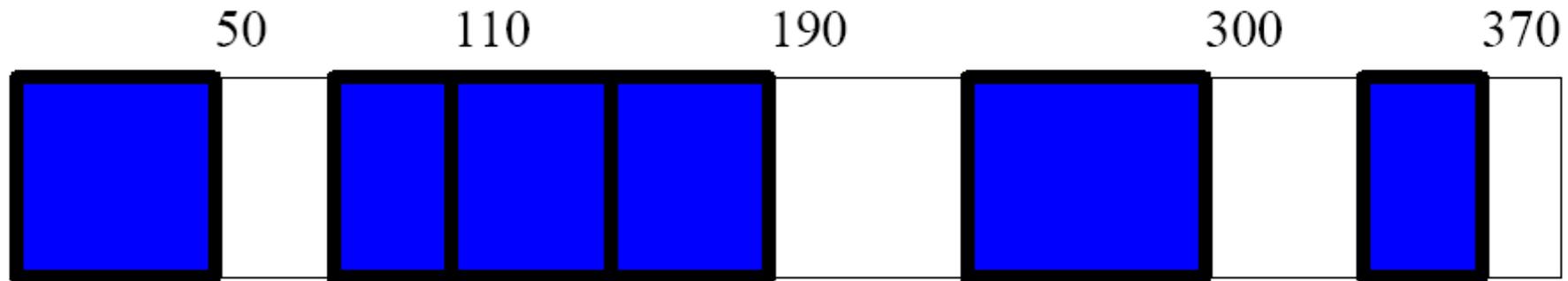
# Allocateur First fit

- Alloue le premier bloc libre qui convient
- Liste de blocs libres gérée en FIFO ou par adresses croissantes
- Chaque bloc libre contient
  - Sa taille
  - Adresse bloc suivant dans la liste
- Allocation doit au pire parcourir toute la liste



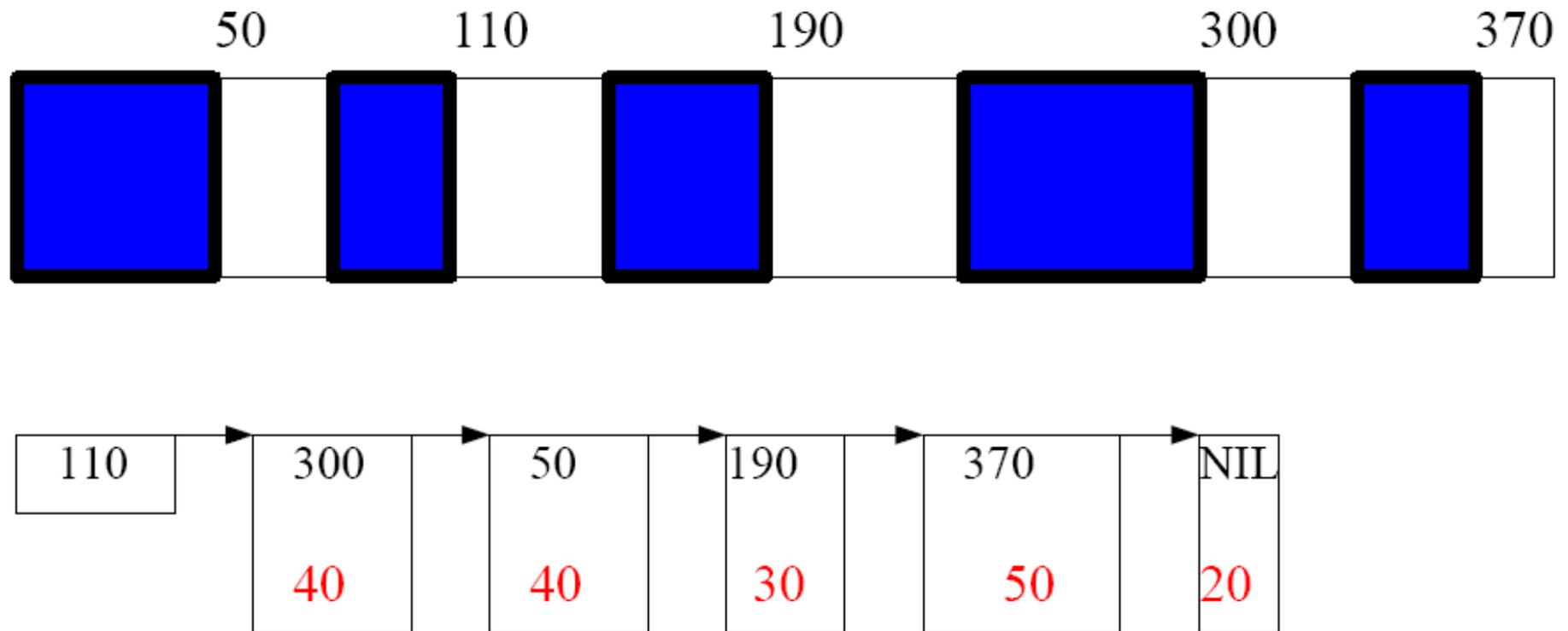
# First fit

□ bloc libre    ■ bloc occupé    190 : adresse    40 : taille



# First fit

Libération bloc de taille 40 à l'adresse 110



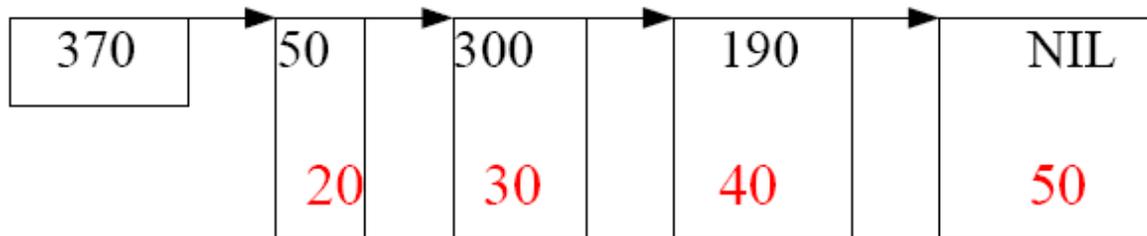
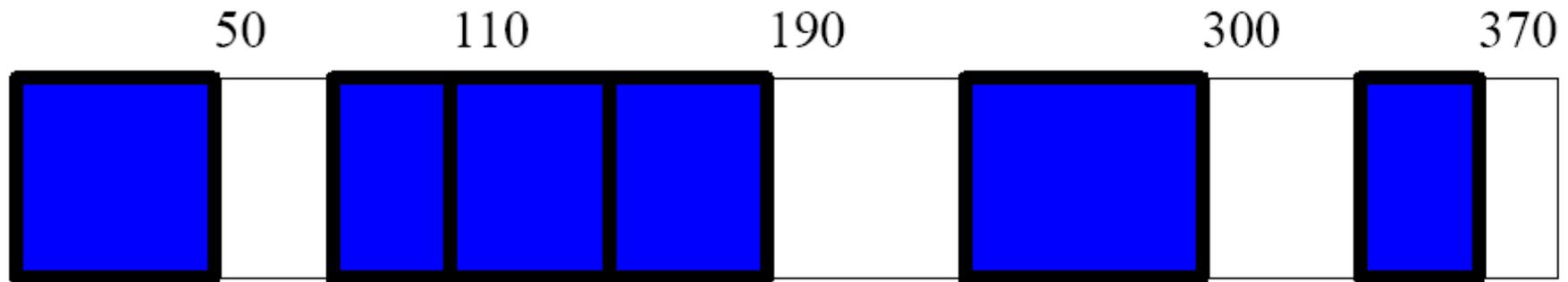
# Allocateur Best fit

- Alloue depuis le bloc libre qui laisse le plus petit résidu
- Liste de blocs libres gérée par tailles croissantes
- Chaque bloc libre contient
  - Sa taille
  - Adresse bloc suivant dans la liste
- Libération rapide
- Allocation doit au pire parcourir toute la liste



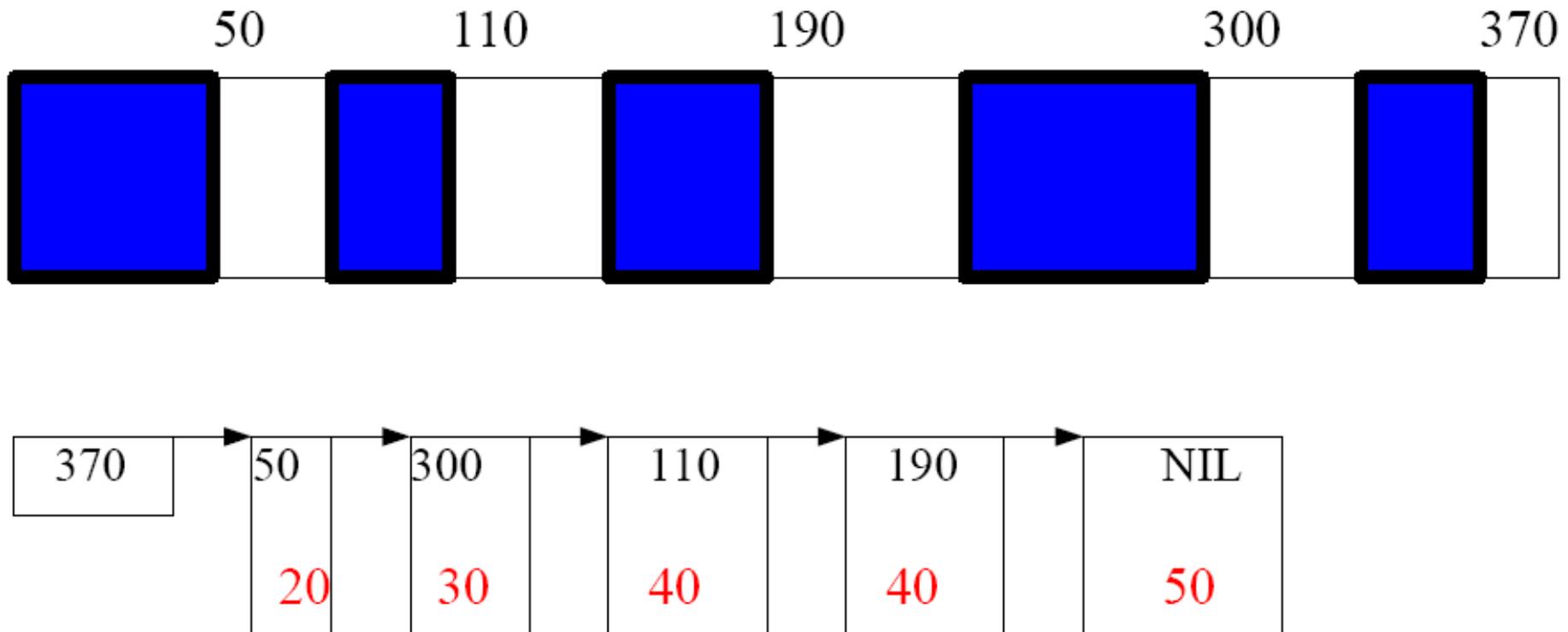
# Best fit

□ bloc libre    ■ bloc occupé    190 : adresse    40 : taille



# Best fit

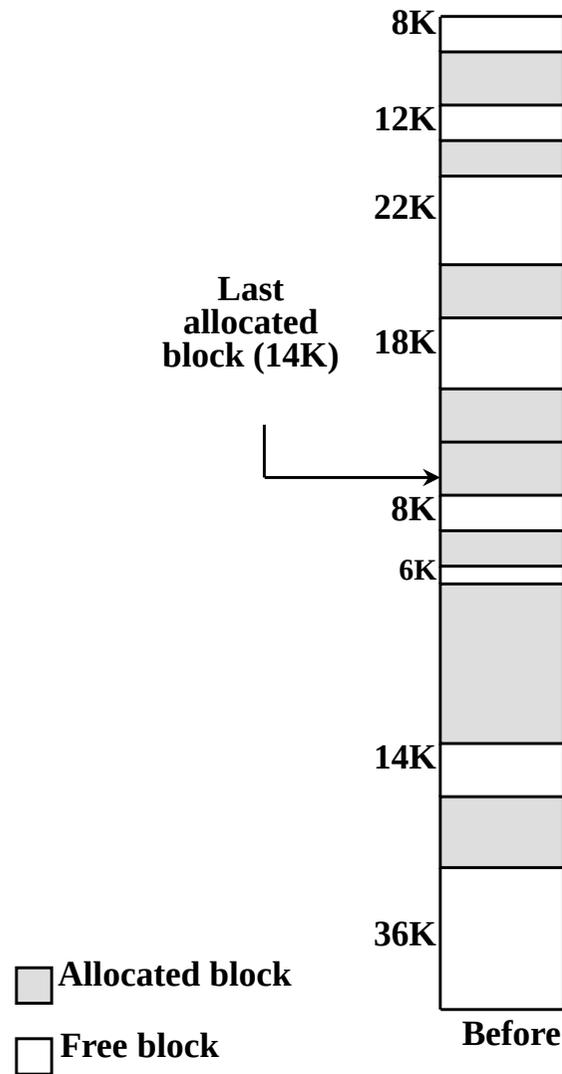
Libération bloc de taille 40 à l'adresse 110



# Worst fit

- Plus grand résidu,
- On choisit la zone telle que la taille restante soit la plus grande possible,
- On combat efficacement l'émiettement.

# Les différents algorithmes...

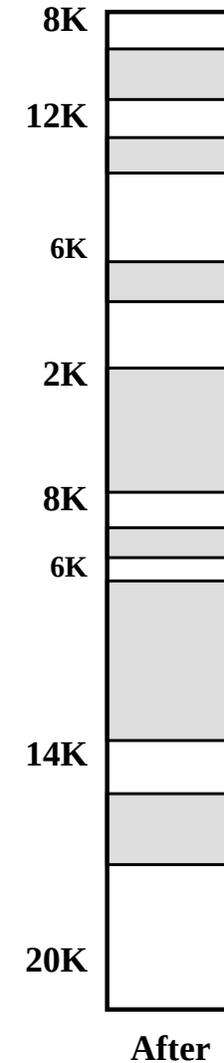


## Allocate 18K

First Fit

Next Fit

Best Fit



# Techniques de gestion mémoire

- Partitionnement à taille fixe
  - Divise la mémoire en partitions lors du boot, les tailles peuvent être identiques ou pas mais fixes,
  - Mécanisme simple souffrant de la fragmentation interne
- Partitionnement à taille variable
  - Les partitions sont créées lors du chargement des programmes
  - Ne crée pas de fragmentation interne mais externe,
- Pagination simple
  - Divise la mémoire en pages de taille fixes et charge les programmes dans les pages disponibles
  - Pas de fragmentation externe, mais légère fragmentation interne



# Pagination simple

- La mémoire est partitionnée en petits morceaux de même taille: les **pages physiques** ou ‘cadres’ ou ‘frames’
- Chaque processus est aussi partitionné en petits morceaux de même taille appelés **pages** (logiques)
- Les pages logiques d’un processus peuvent donc être assignés aux cadres disponibles n’importe où en mémoire principale
- Conséquences:
  - un processus peut être éparpillé n’importe où dans la mémoire physique.
  - la fragmentation **externe** est éliminée



# Pagination

- Le principe de la pagination est simple
  - l'espace d'adressage de chaque programme est partagé en blocs de **taille fixe** (des tailles de 4-16KB sont courantes),
  - les **pages** utilisées sont chargées en mémoire.
  - le restant de l'espace d'adressage est stocké sur le disque dur.
- Les adresses physiques des pages en mémoire sont stockées dans une **page table**.
- Chaque processus dispose de sa **page table**, qui fait office de multiples **base register**.



# Exemple de chargement de processus

Frame number	Main memory
0	
1	
2	
3	
4	
5	
6	
7	
8	
9	
10	
11	
12	
13	
14	

(a) Fifteen Available Pages

Frame number	Main memory
0	A.0
1	A.1
2	A.2
3	A.3
4	
5	
6	
7	
8	
9	
10	
11	
12	
13	
14	

(b) Load Process A

Frame number	Main memory
0	A.0
1	A.1
2	A.2
3	A.3
4	B.0
5	B.1
6	B.2
7	
8	
9	
10	
11	
12	
13	
14	

(b) Load Process B

Frame number	Main memory
0	A.0
1	A.1
2	A.2
3	A.3
4	B.0
5	B.1
6	B.2
7	C.0
8	C.1
9	C.2
10	C.3
11	
12	
13	
14	

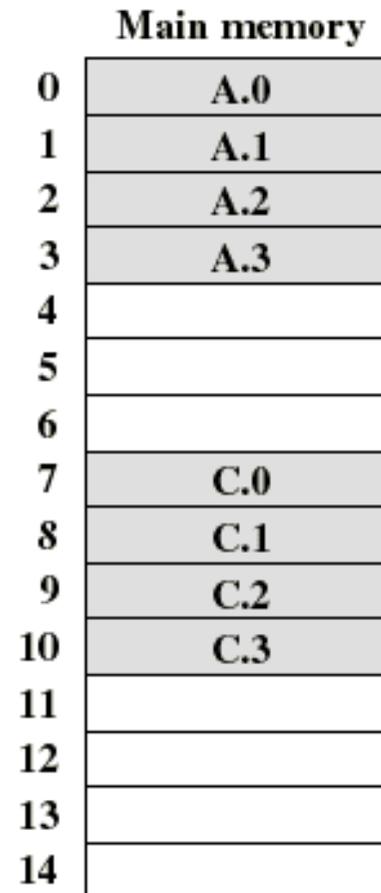
(d) Load Process C

- Supposons que le processus B se termine ou soit suspendu

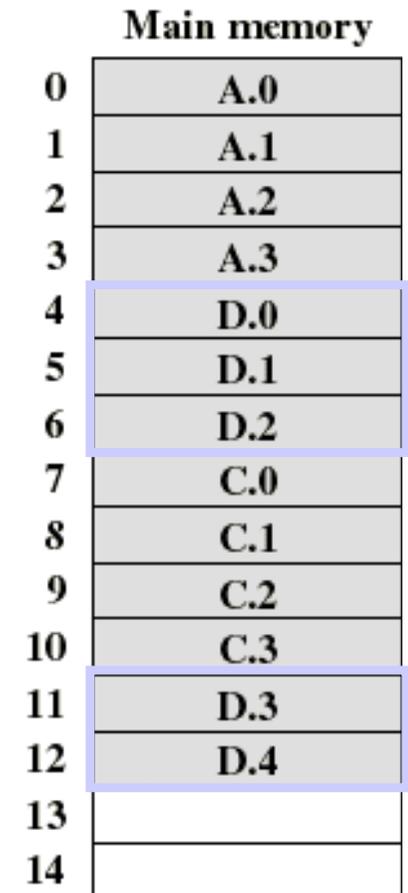


# Exemple de chargement de processus

- Nous pouvons maintenant transférer en mémoire un programme D, qui demande 5 pages
  - bien qu'il n'y ait pas 5 pages contigües disponibles
- La fragmentation externe est limitée au cas où le nombre de pages disponibles n'est pas suffisant pour exécuter un programme en attente
- Seule la dernière page d'un programme peut souffrir de **fragmentation interne** (moy. 1/2 cadre par proc)

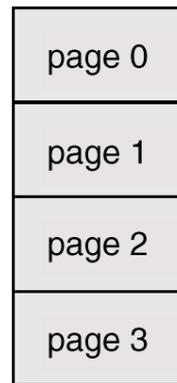


(e) Swap out B



(f) Load Process D

# Tableaux de pages

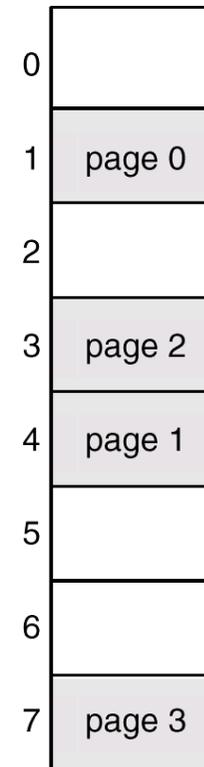


logical  
memory

0	1
1	4
2	3
3	7

page table

frame  
number



physical  
memory

Les entrées dans le tableau  
de pages sont aussi  
appelées *descripteurs* de  
pages



# Tables de pages

0	0
1	1
2	2
3	3

Process A  
page table

0	—
1	—
2	—

Process B  
page table

0	7
1	8
2	9
3	10

Process C  
page table

0	4
1	5
2	6
3	11
4	12

Process D  
page table

13
14

Free frame  
list

- Le SE doit maintenir une **table de pages** pour chaque processus
- Chaque descripteur de pages contient le numéro de frame où la page correspondante est physiquement localisée
- Une table de pages est indexée par le numéro de la page afin d'obtenir le numéro de la frame
- Une liste de frames disponibles est également maintenue (*free frame list*)



# Pagination - Adressage

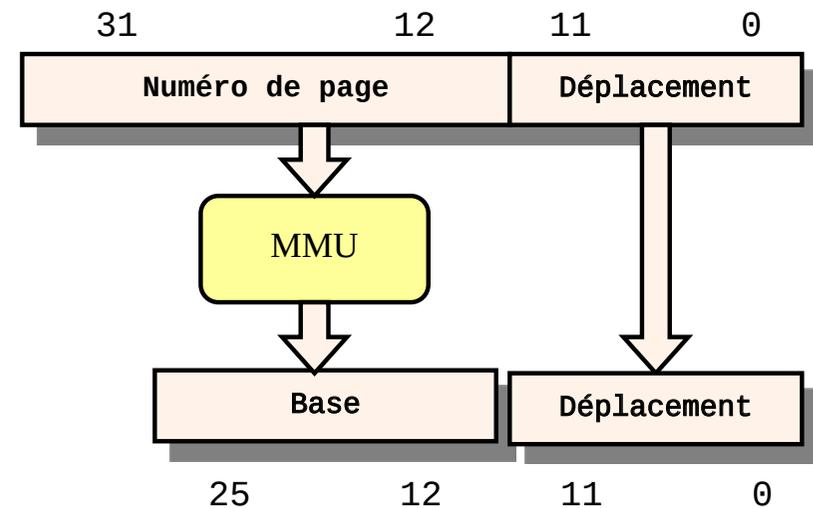
Une adresse virtuelle est donc composée de deux parties : un **numéro de page** et un **déplacement** dans la page.

Exemple:

Largeur des adresses = 32bits

Taille des pages = 4KB

Taille de la mémoire = 64MB



Cette transformation, réalisée par la MMU, permet de travailler avec des espaces d'adressage plus grands que la mémoire physique, et d'autre part de référencer des données sur une même page avec peu de bits.



# Pagination - *Page faults*

- Une adresse virtuelle peut donc référencer
  - soit une page en mémoire (*page hit*)
  - soit une page sur le disque dur (*page miss*).
- Si la page cherchée se trouve sur le disque dur, le MMU engendre une interruption (*page fault*)
- Une procédure se charge de transférer les données en mémoire (une opération qui demande plusieurs millions de cycles d'horloge).



# Pagination - Remplacement

- Lors qu'une nouvelle page est transférée du disque dur à la mémoire, il est parfois nécessaire d'en transférer une autre de la mémoire au disque dur (*swap*).
- Étant donné le "prix" très élevé d'un *page fault*, l'algorithme utilisé pour décider quelle page va être "swappée" est fondamental pour la performance d'un système. L'algorithme de choix pour cette opération est le **LRU** (*least recently used*).

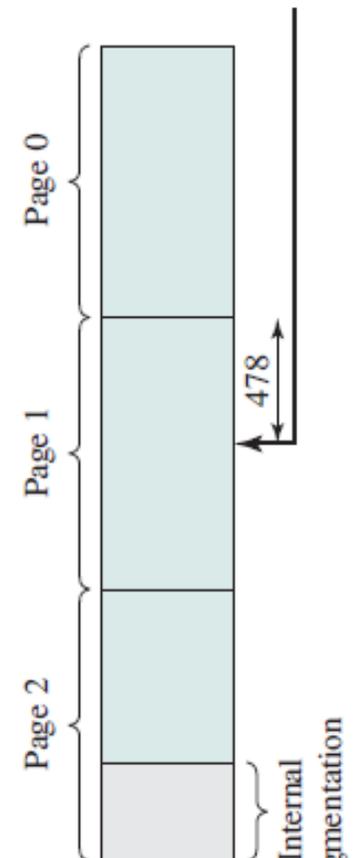


# Traduction d'adresses

- L'adresse logique est facilement traduite en adresse physique
  - car la taille des pages est une puissance de 2
    - les pages débutent toujours à des adresses qui sont puissance de 2
    - qui ont autant de 0s à droite que la longueur de l'offset
    - donc ces 0s sont remplacés par l'offset
  - Ex: si 16 bits sont utilisés pour les adresses et que la taille d'une page = 1K (1024 bits) on a besoin de 10 bits pour le décalage, laissant ainsi 6 bits pour le numéro de page
  - L'adresse logique (n,m) est traduite à l'adresse physique (k,m) en utilisant n comme index sur la table des pages et en le remplaçant par l'adresse k trouvée.

Logical address =  
Page# = 1, Offset = 478

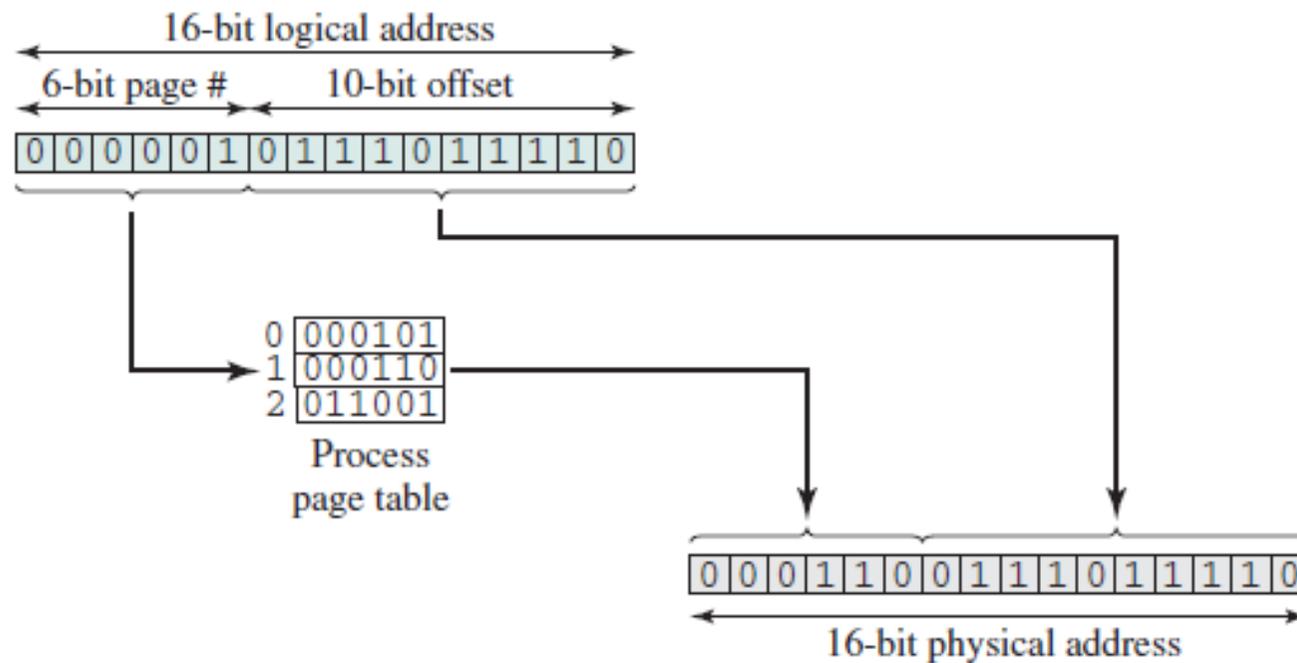
00000101111011110



(b) Paging  
(page size = 1K)



# Traduction d'adresse (logique-physique) pour la pagination



# Techniques de gestion mémoire (next)

- Segmentation simple (*prochain cours*)
  - Divise les programmes en segments
  - Pas de fragmentation interne, faible fragmentation externe
- Mémoire virtuelle paginée (*prochain cours*)
  - Basée sur un mécanisme de pages, mais pas toutes en mémoire centrale,
  - Autorise un vaste espace de mémoire virtuelle
  - Surcout d'exécution
- Mémoire virtuelle segmentée (*prochain cours*)
  - Basée sur un mécanisme de segments, mais pas toutes en mémoire centrale,
  - Facilité pour partager des modules.



Any question ?



Université  
de Limoges

FACULTÉ  
DES SCIENCES  
ET TECHNIQUES

