REDS
Reconfigurable & embedded Digital Systems

heig-vd
HAUTE ÉCOLE D'INGÉNIERIE ET DE GESTION DU CANTON DE VAUD
www.heig-vd.ch

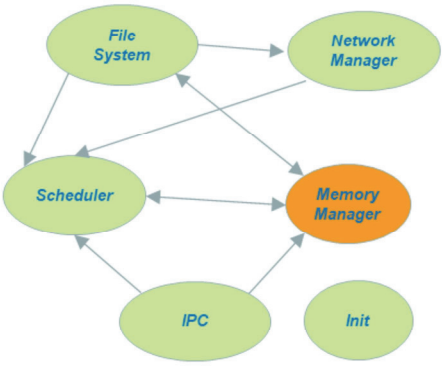
Systèmes d'exploitation
Allocation de la mémoire physique

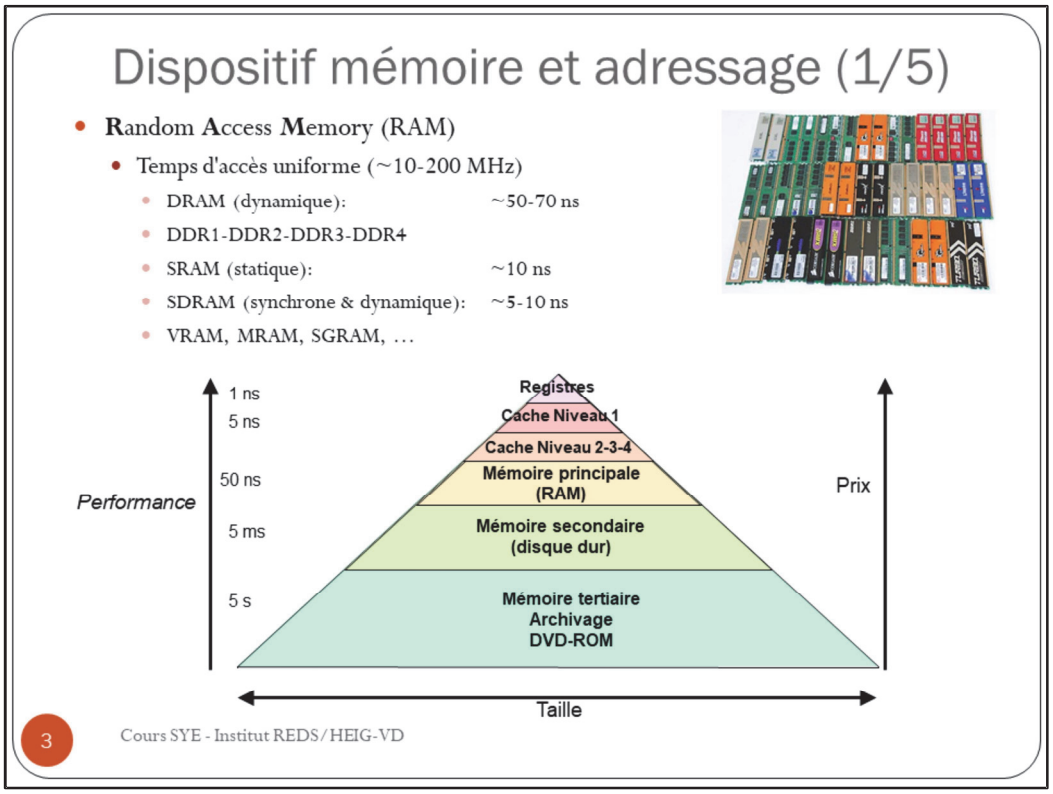
Profs Daniel Rossier, Alberto Dassatti, Salvatore Valenza

Version 3.3 (2017-2018)

Plan

- Dispositif mémoire et adressage
- Algorithme d'allocation
- Fragmentation





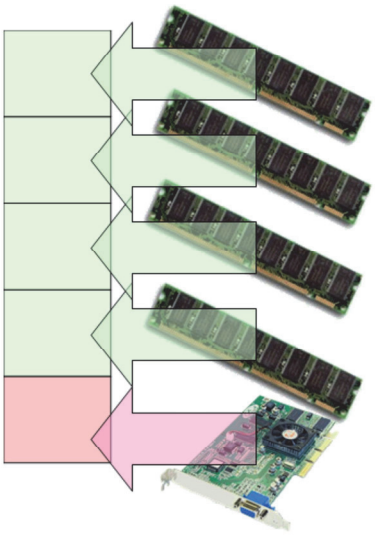
On trouve dans un système informatique toute sorte de mémoires de stockage: les plus rapides – car directement intégrées au processeur – sont les registres. La taille usuelle de ces mémoires varient généralement de 8 à 64 bits (32 bits pour la plupart des architectures). L'accès au registre est quasi-instantané. Le second type de mémoire correspond aux antémémoires (*caches*). Ce sont des mémoires embarquées dans le processeur et sont utilisées pour stocker des octets *juste avant* leur traitement ou de manière temporaire. Ces éléments mémoire ont des temps d'accès de l'ordre de quelques nanosecondes (ns). Viennent ensuite les mémoires de type RAM qui sont connectées à un bus relié au processeur. Pour accéder ces mémoires, il est nécessaire d'effectuer des requêtes sur le bus d'adresse/données et cela peut prendre jusqu'à plusieurs dizaines, voire centaines de nanosecondes.

Les mémoires secondaires sont beaucoup plus lentes et forment les mémoires dites de stockage (persistentes). Il s'agit des disques durs, des mémoires *flash*, des *sticks* mémoire USB, des *SD-cards*, etc. Les temps d'accès sont environ un million de fois plus lents que pour les mémoires principales. On parle de l'ordre de la milliseconde (ms).

On notera également qu'aujourd'hui, il est possible d'utiliser la mémoire présente dans certains périphériques (carte graphique, contrôleurs de périphériques, etc.) pour des applications nécessitant des performances élevées (ces mémoires étant généralement synchrones et très rapides).

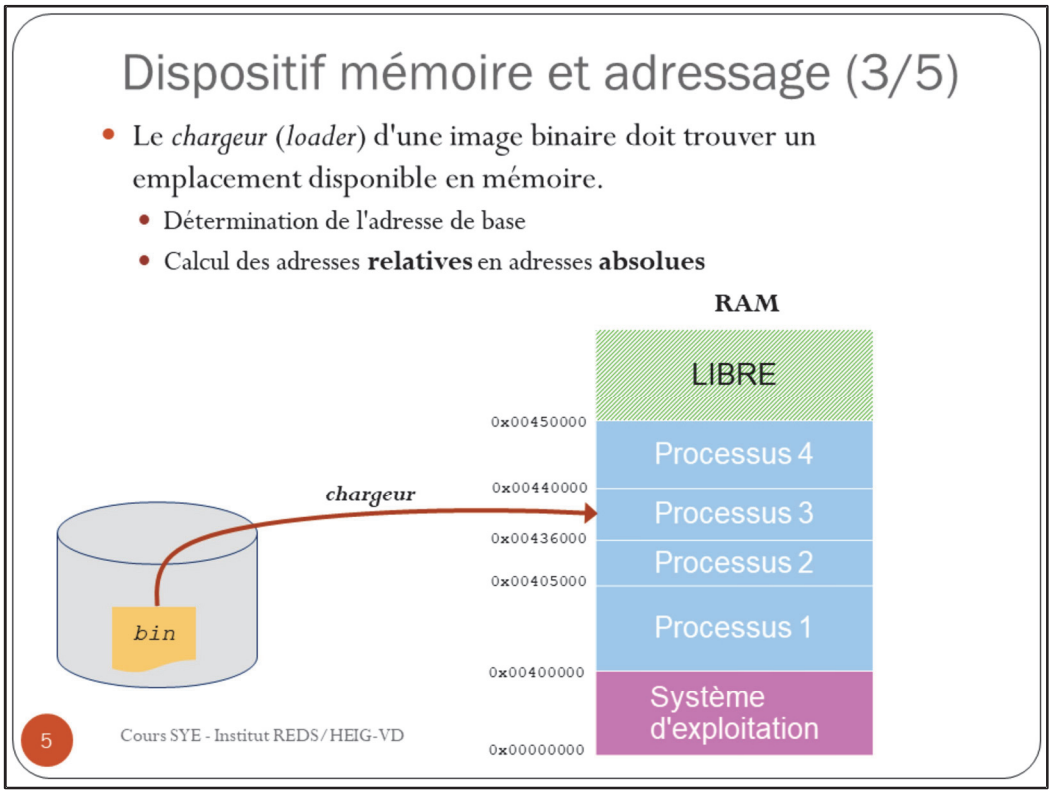
Dispositif mémoire et adressage (2/5)

- Espace d'adressage **linéaire**
- Accès uniformes
  - Mémoire principale (RAM)
  - Cartes graphiques
  - Périphériques
  - …



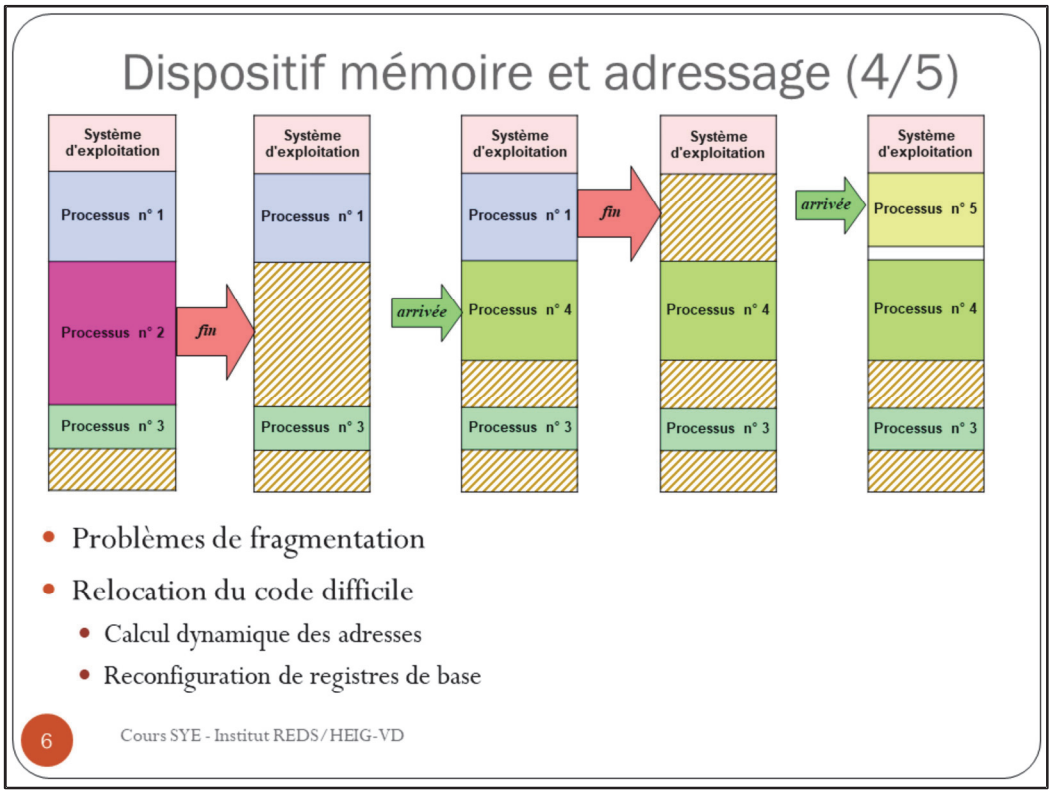
Un espace d'adressage physique linéaire permet d'accéder non seulement la RAM, mais également toutes les mémoires embarquées disponibles sur la plate-forme, y compris les registres de contrôleurs de périphériques (*memory-mapped I/O*). Ainsi, l'accès à ces différentes zones mémoires peuvent être faites avec les mêmes instructions de lecture et d'écriture.

Du point de vue de l'OS, un espace d'adressage linéaire facilite grandement la gestion mémoire, puisque différentes zones mémoire correspondent à différents intervalles *(ranges)* d'adresses.



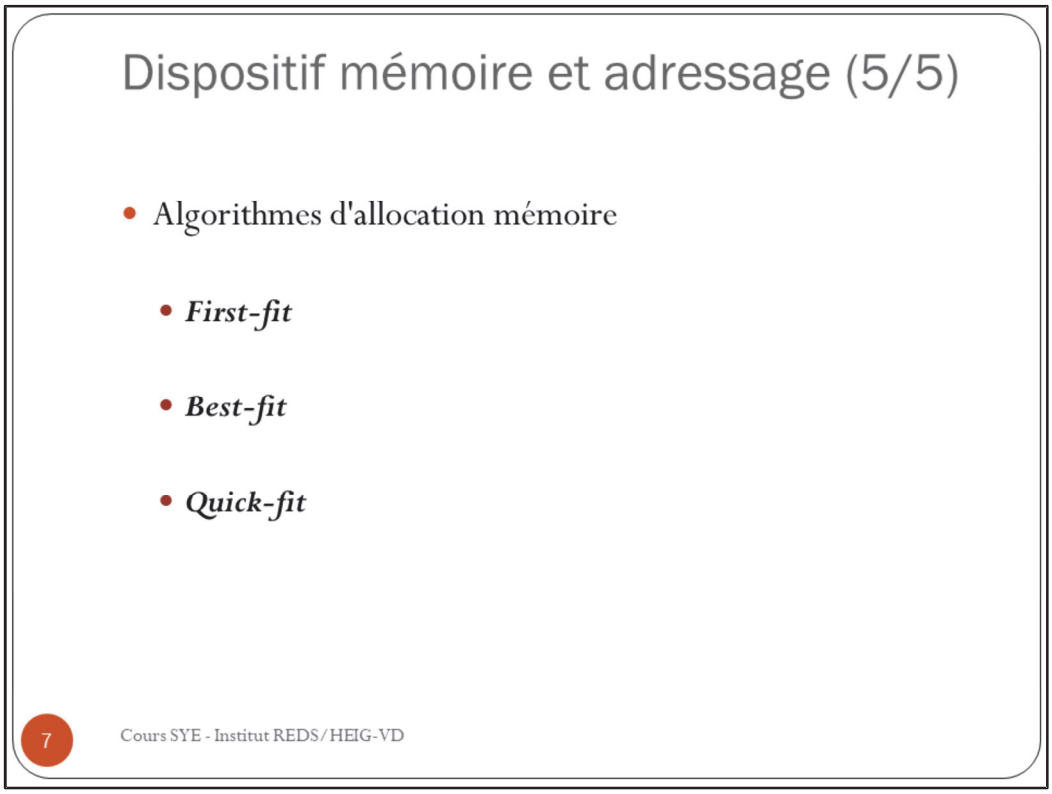
Le chargeur, i.e. partie du code de l'appel système *exec()*, doit trouver un emplacement mémoire afin de pouvoir placer l'image binaire. Par conséquent, toutes les adresses utilisées par le code binaire devront être réajustées afin que les sauts et autres références aux objets du programme puissent être corrects.

On se rappelle que pour un système monoprogrammé typiquement, l'image binaire peut contenir directement des adresses absolues. L'image est donc toujours chargée à la même adresse et aucun réajustement n'est nécessaire. Ce n'est pas le cas bien entendu pour les systèmes multiprogrammés.



Le mouvement des processus (va-et-vient entre le disque et la mémoire) génère une fragmentation au fil du temps. Au bout d'un certain temps, la mémoire se trouve très fragmentée et il devient difficile de trouver un emplacement disponible pour charger un processus, alors que la somme des *trous* mémoire constituerait une taille mémoire suffisamment grand.

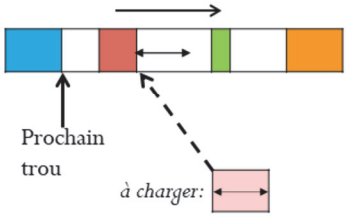
La solution serait de défragmenter la mémoire. Or, la défragmentation mémoire a pour conséquence de modifier l'emplacement du code – donc des instructions – et des données. Il est nécessaire de réajuster les adresses et de réadapter les instructions (instruction de saut, accès mémoire, etc.). Ce mécanisme peut s'avérer très complexe et lent. C'est pourquoi, lorsque l'on fait ce genre d'allocation directement dans la mémoire physique, les processeurs modernes comportent des registres permettant de stocker **l'adresse de base** du processus, et tous les accès mémoire se font de manière relative par rapport à cette adresse: autrement dit, les adresses en mémoire sont des ***offsets*** qui sont additionnés à l'adresse de base. Ainsi, bouger un processus en mémoire signifie modifier l'adresse de base dans le registre de base.



Nous présentons par la suite trois algorithmes fondamentaux permettant l'allocation d'une zone mémoire contiguë dans la mémoire physique. Il s'agit des algorithmes *first-fit, best-fit* et *quick-fit*.

Allocation *First-Fit* (1/1)

- Recherche de la première zone libre satisfaisante
  - Utilisation d'un pointeur courant (au départ sur la première adresse disponible)
  - Recherche circulaire
- Génère des résidus de toute taille
- Implémentation simple et performance élevée



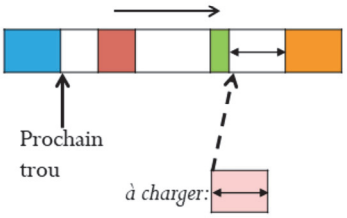
L'algorithme *first-fit* est très simple à implémenter et facile à comprendre : il consiste à parcourir l'espace d'adressage (espace mémoire) de manière **séquentielle** et d'identifier une zone contiguë de **taille satisfaisante** correspondante à la taille du bloc que l'on souhaite charger. Comme le parcours s'effectue séquentiellement, on peut très bien utiliser un pointeur mémorisant la position courante afin de poursuivre la recherche à partir de cet endroit, lors d'une prochaine allocation.

Cette méthode d'allocation ne tient pas compte de la taille des zones disponibles présentes dans la mémoire à un moment donné. C'est pourquoi, au fil du temps, alors que les blocs sont alloués et retirés de la mémoire, des trous de tailles différentes vont apparaître, favorisant ainsi une fragmentation élevée de la mémoire. En prenant systématiquement le premier "trou" disponible quelque soit sa taille, le reste de la mémoire non utilisée (résidu) peut être de taille variable et empêcher rapidement une allocation d'un bloc contigu, alors même que la somme totale des résidus de la mémoire peut être supérieure à la taille du bloc recherché.

L'algorithme suivant permet de réduire ce problème.

Allocation *Best-Fit* (1/2)

- Recherche de la zone **la "mieux" adaptée**



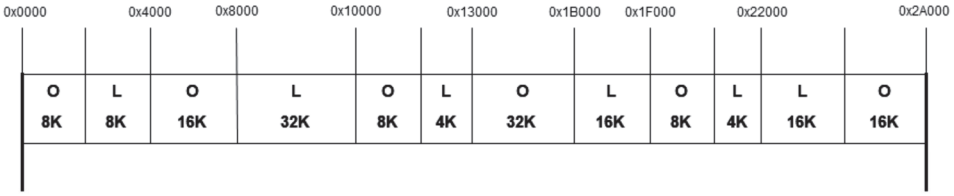
- Génère un résidu minimal
  - Préserve les "grands" trous
- Nécessaire de parcourir **toute la mémoire**

L'algorithme *best-fit* tient compte de la taille des résidus, c-à-d que l'algorithme inspecte l'ensemble de la mémoire à la recherche de la zone **générant le plus petit résidu**.

Cette approche permet de préserver les zones de grande taille afin de donner une chance élevée aux demandes futures.

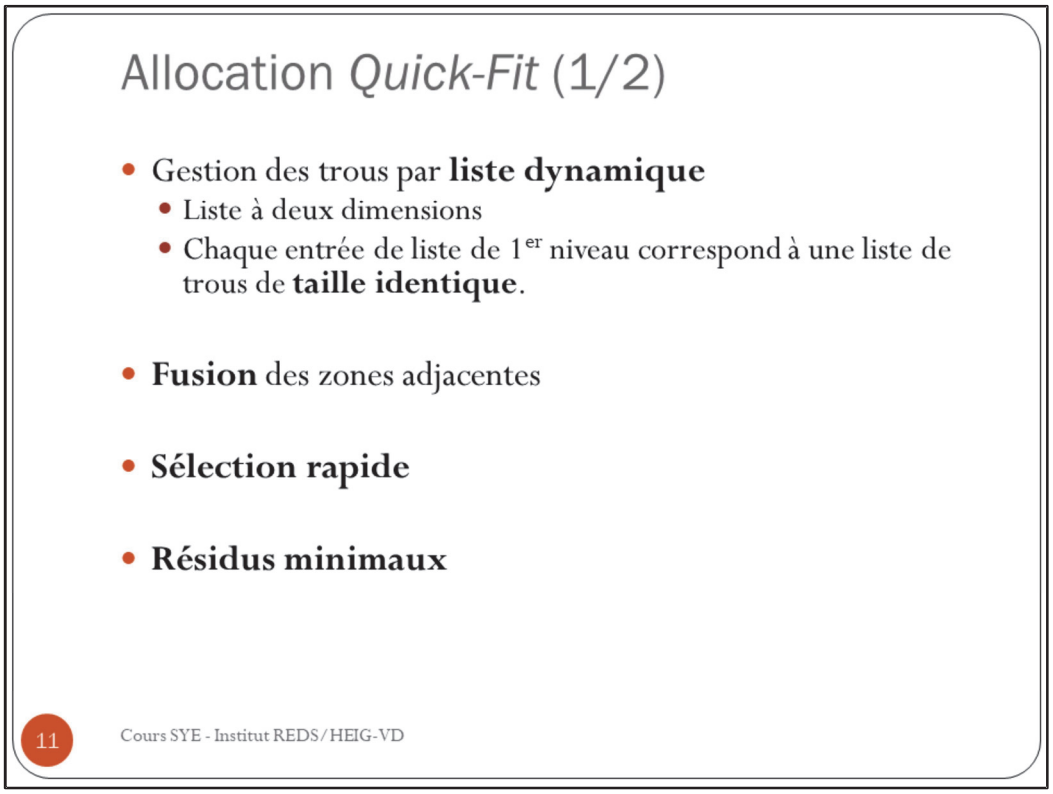
Allocation *Best-Fit* (2/2)

- Les processus suivants arrivent dans cet ordre :
  - **P_1 (18K)**
  - **P_2 (8K)**
  - **P_3 (1K)**
- Comment ces processus seront alloués lors d'une allocation de type *Best-Fit* avec la configuration mémoire ci-dessous ?



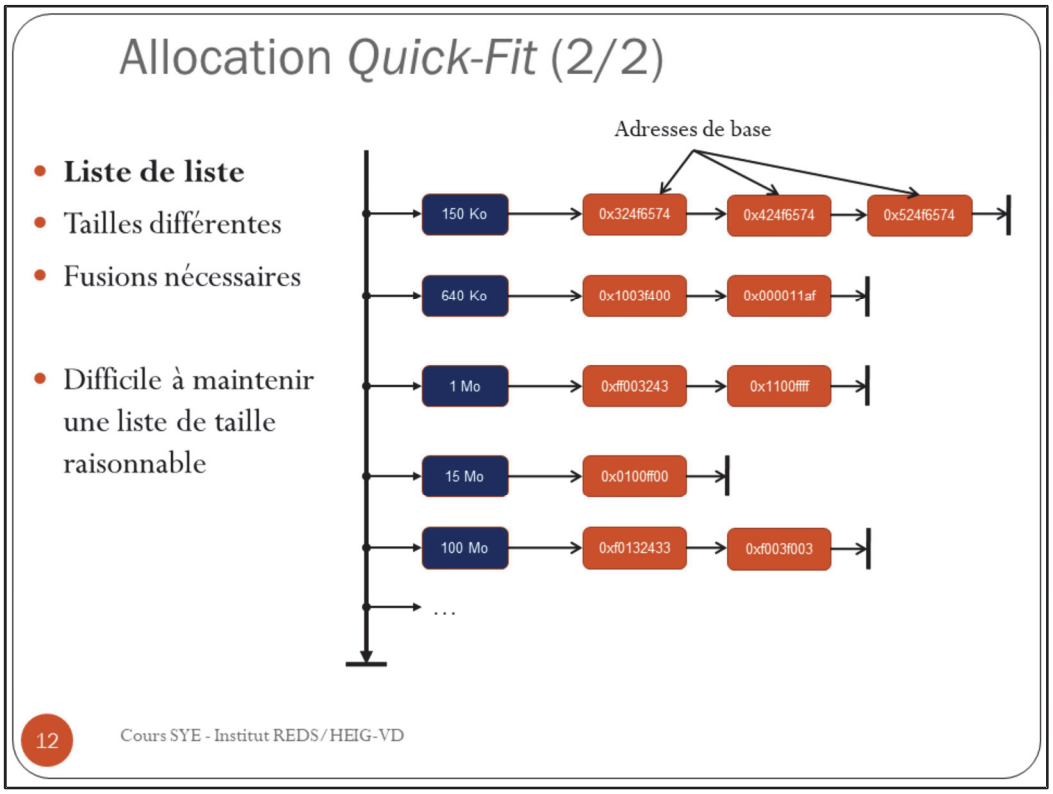
O: Occupé
L: Libre

- Adresse physique de P1: _____
- Adresse physique de P2: _____
- Adresse physique de P3: _____



L'algorithme *quick-fit* consiste à gérer une liste dynamique à deux niveaux : chaque entrée de la liste contient une liste de trous de taille identique.

Chaque requête d'allocation ou de retrait exige une mise à jour de la liste. Lorsque la mémoire est mise à forte contribution, une telle liste peut contenir beaucoup d'entrées. C'est pourquoi, il est aussi nécessaire, périodiquement, de *fusionner* des trous adjacents en mettant à jour la liste. Cette opération pouvant demander un certain temps CPU, on peut l'imaginer qu'elle s'effectue en arrière-plan (*background*).



Une entrée de la liste de 1er niveau va contenir la taille des trous associés et une référence vers une liste d'adresses. Ces adresses correspondent aux adresses de base des zones mémoires libres.

Références

- A. Silberschatz et al.:
  "Operating Systems Concepts", 6th edition, Wiley
- Andrew Tanenbaum,
  "Systèmes d'exploitation", 2ème édition