Université Nice Sophia Antipolis

Gestion de la Mémoire

Présentation: **Stéphane Lavirotte**
Auteurs: ... et al*

(*) Cours réalisé grâce aux documents de :
Denis Bechet, Pierre-Antoine Champin,
Stéphane Lavirotte, Jean-Paul Rigault

Mail: Stephane.Lavirotte@unice.fr
Web: http://stephane.lavirotte.com/
Université de Nice - Sophia Antipolis

Gestion Mémoire

Introduction

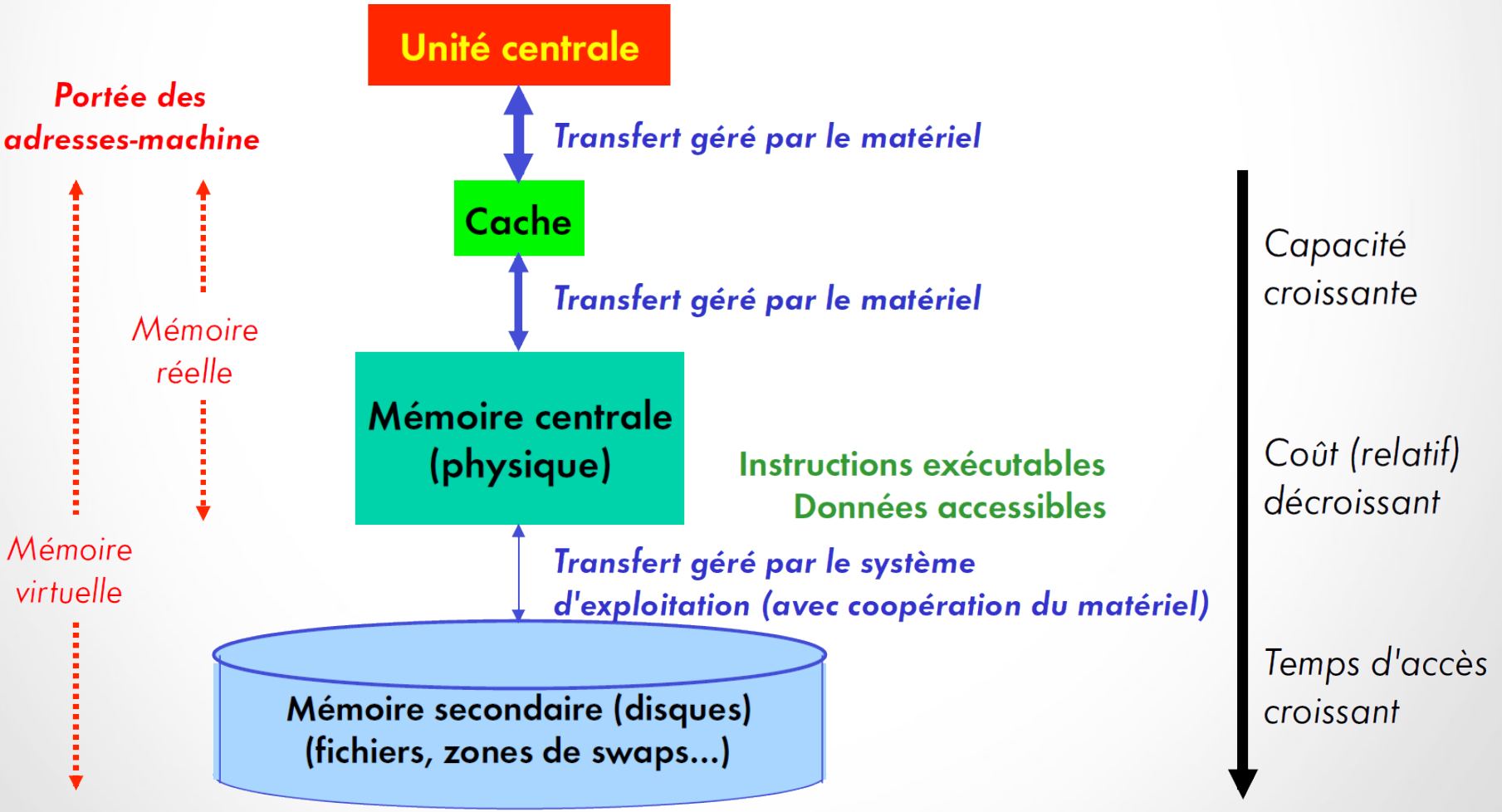
Problématiques

Université Nice Sophia Antipolis

Introduction sur la Gestion Mémoire

- ✓ Ressource importante
- ✓ Ressource hiérarchisée
  - Mémoire cache (dans le processeur)
    - très rapide, peu importante, gérée par le matériel
  - Mémoire principale (RAM)
    - rapide, importante, gérée par le SE
  - Mémoire secondaire (disques durs)
    - lente, très importante, gérée par le SE
- ✓ Gestion de la mémoire par le Système d'Exploitation
- ✓ But :
  - Offrir un espace d'adressage indépendant aux processus

Hiérarchie Mémoire



Université Nice Sophia Antipolis

Représentation des Adresses Mémoires

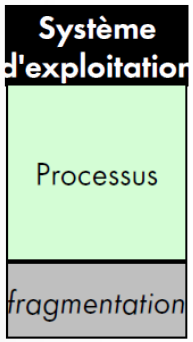
- ✓ Différentes manières de manipuler les adresses
  - Code source (pour le programmeur)
    - Adresses symboliques
      - Par exemple : `int compteur`
  - Module objet (suite à la compilation)
    - Adresses traduites
      - Par exemple le 50ème mot depuis le début d'espace mémoire
  - Module exécutable (suite au chargement)
    - Adresses absolues
      - Par exemple l'emplacement mémoire situé à l'adresse `FFF7`
- ✓ Le programmeur et les processus manipulent des adresses logiques
- ✓ Le système et le matériel manipulent des adresses physiques

Mémoire Réelle

- ✓ Contraintes de la mémoire réelle
  - Les adresses-machines ne référencent que des instructions ou données en mémoire physique
  - L'ensemble du processus en cours d'exécution doit donc être en mémoire physique
  - Les adresses doivent être contigües
- ✓ Problème
  - Gestion de la multi-programmation
- ✓ Différents modèles de gestion de la mémoire

Mémoire Réelle: Partitions fixes

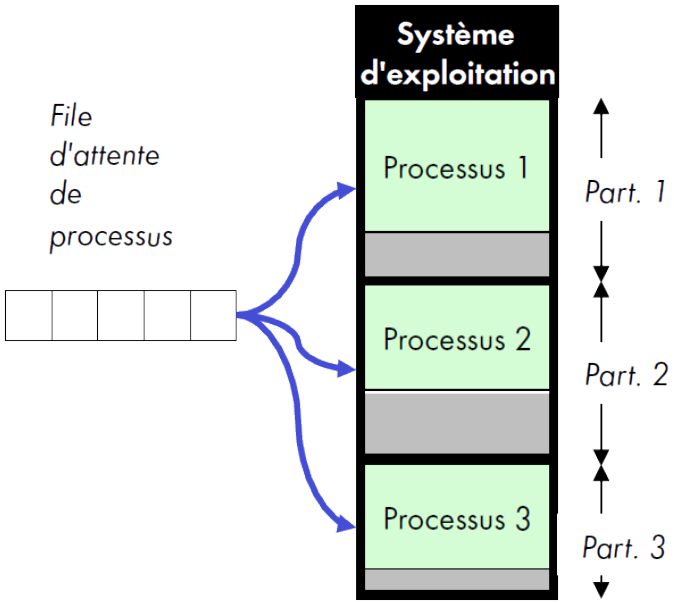
Mono-programmation



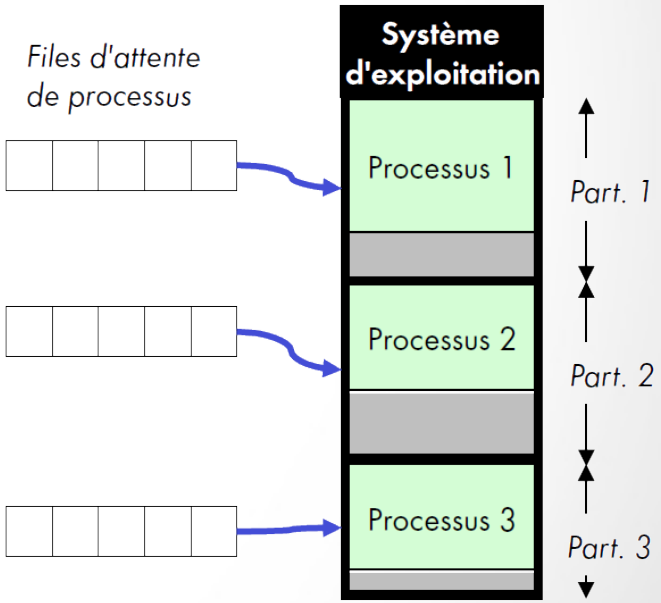
Système monotâche mono-utilisateur

Ex: MS-DOS

Multi-programmation



Partitions fixes avec une seule file d'attente

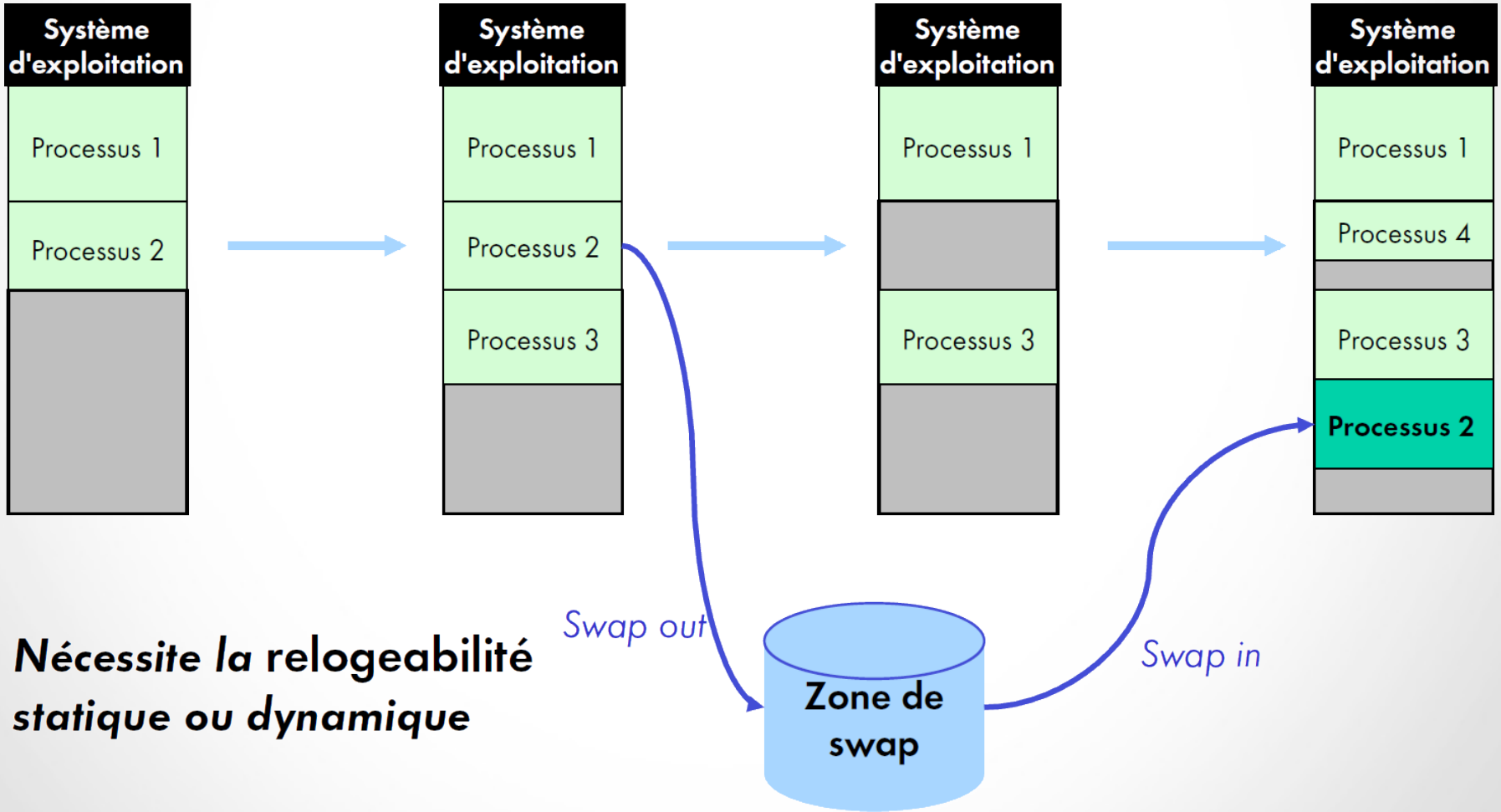


Partitions fixes avec une file d'attente par partition

Ex: mainframe d'IBM sous OS/360

Mémoire Réelle: Partitions Variables

Multi-programmation



Nécessite la relogeabilité statique ou dynamique

Problèmes liés à la Multi-programmation

- ✓ Protection
  - Interdire à un processus l'accès à l'espace mémoire des autres processus et du système d'exploitation
    - registres de protection : base + limite
    - bits de protection des pages de la mémoire
- ✓ Translation (Relocation)
  - Un processus doit pouvoir être chargé à n'importe quelle adresse mémoire
    - Translation des adresses globales lors du chargement du programme en mémoire
    - Utilisation des modes d'adressage relatif à un registre (pile/instruction/registre de segment)
- ✓ Allocation / Réallocation
  - Un programme doit obtenir la zone mémoire dont il a besoin si la mémoire est disponible
  - Et modifier ses besoins pendant son exécution

Problèmes de Disponibilité de Mémoire

- ✓ La mémoire disponible peut être insuffisante à un moment donné

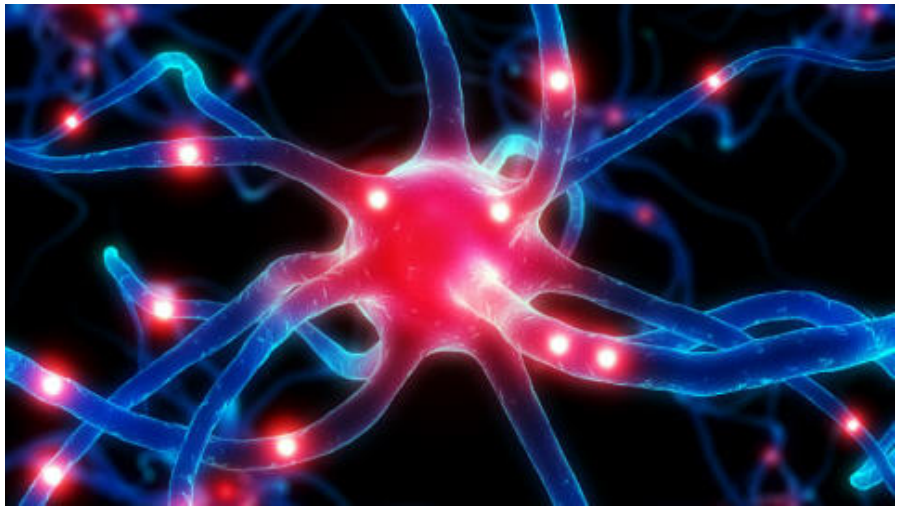
- ✓ Trois types de mécanismes pour gérer le manque de mémoire:
  1. Interdire de nouveaux processus (exemple Minix)
  2. Placer certains processus en mémoire secondaire pour libérer de l'espace mémoire en attendant que la charge mémoire diminue
     - Les processus « swapés » ne sont pas exécutables
  3. Placer des morceaux de la mémoire des processus en mémoire secondaire : mécanisme de mémoire virtuelle
     - Les processus sont partiellement exécutables

Conclusion Partielle

- ✓ Problématiques
  - Protection / Partage
  - Translation (Relocation)
  - Allocation / Réallocation (problème de fragmentation)
  - Augmentation (plus de mémoire que disponible)

- ✓ Plusieurs techniques sont utilisées pour pallier aux problèmes de la multi-programmation et au manque de mémoire
  - Mémoire Virtuelle
    - Utilisation de modes d'adressages relatifs
    - Translation des adresses (réelles ↔ virtuelles)
    - Pagination
    - Segmentation
  - Echange sur mémoire secondaire (va et vient « *swap* »)

Université Nice Sophia Antipolis



Techniques de Gestion Mémoire : Mémoire Virtuelle

D'un point de vue Espace Noyau

Gestion globale de la mémoire

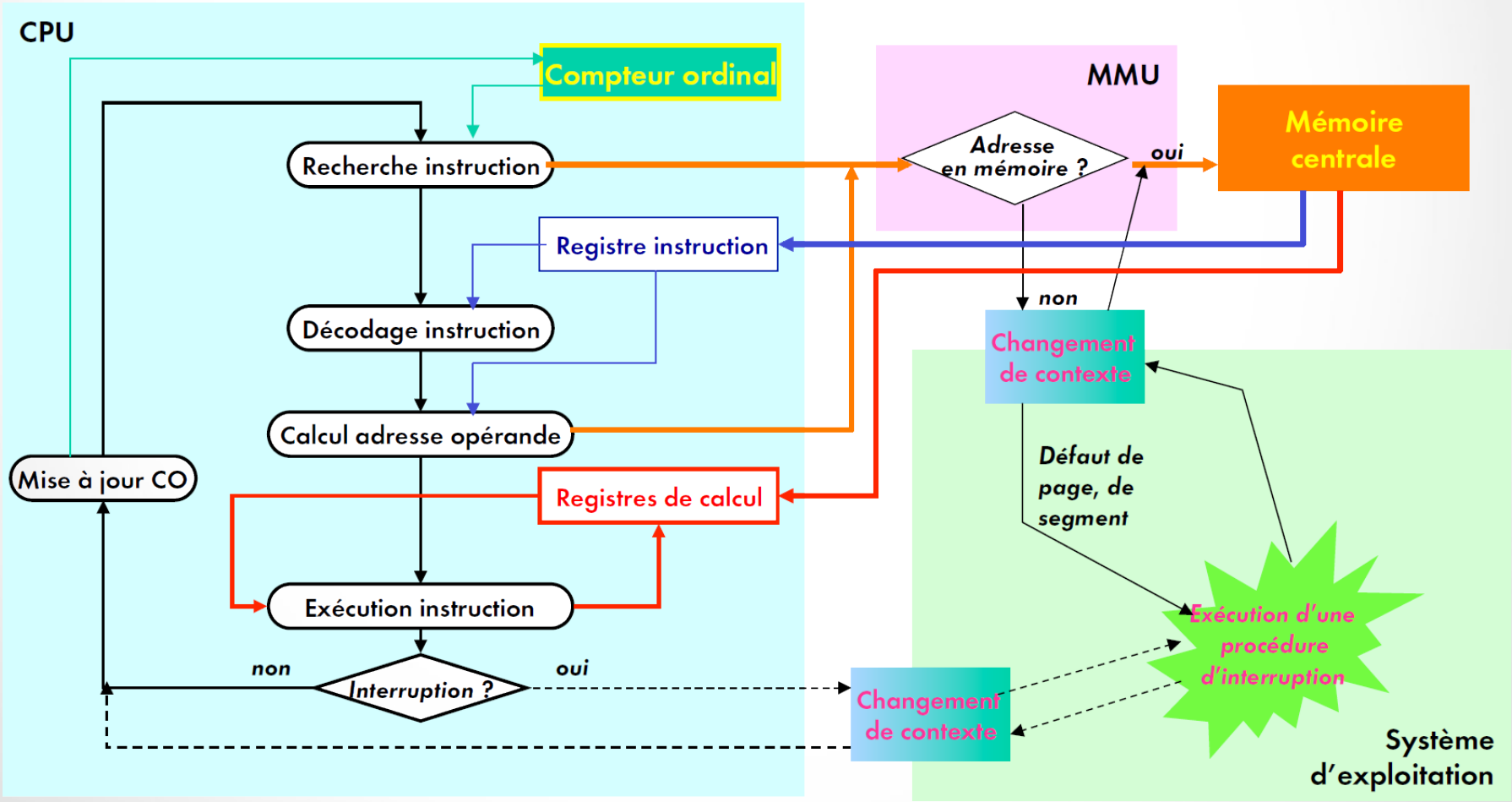
Mémoire Virtuelle

- ✓ Les adresses-machines couvrent un espace plus grand que la mémoire physique
  - Chaque processus peut même disposer de son propre espace d'adressage
- ✓ Le matériel et le système d'exploitation assurent automatiquement la montée en mémoire physique des informations utiles à l'exécution courante
  - La mémoire physique sert de cache à la mémoire virtuelle
  - On doit disposer d'un mécanisme de traduction entre l'adresse virtuelle et l'adresse physique

Traduction d'adresses

- ✓ Exécution à chaque accès mémoire
  - Coopération du matériel indispensable
- ✓ MMU (Memory Management Unit) assure la traduction en adresses physiques
  - Peu coûteux (couteux et compliqué si au niveau logiciel)[2]
  - Transparent pour les programmes
- ✓ Si l'information adressée par l'instruction courante n'est pas en mémoire physique
  - l'instruction courante est suspendue, de même que le processus qui la contient ; un déroutement est déclenché
  - le système d'exploitation prend la main et fait monter l'information en mémoire physique
  - lorsque le processus est ordonnancé à nouveau, l'instruction suspendue est reprise à l'accès mémoire

Illustration Traduction d'adresses



Université Nice Sophia Antipolis

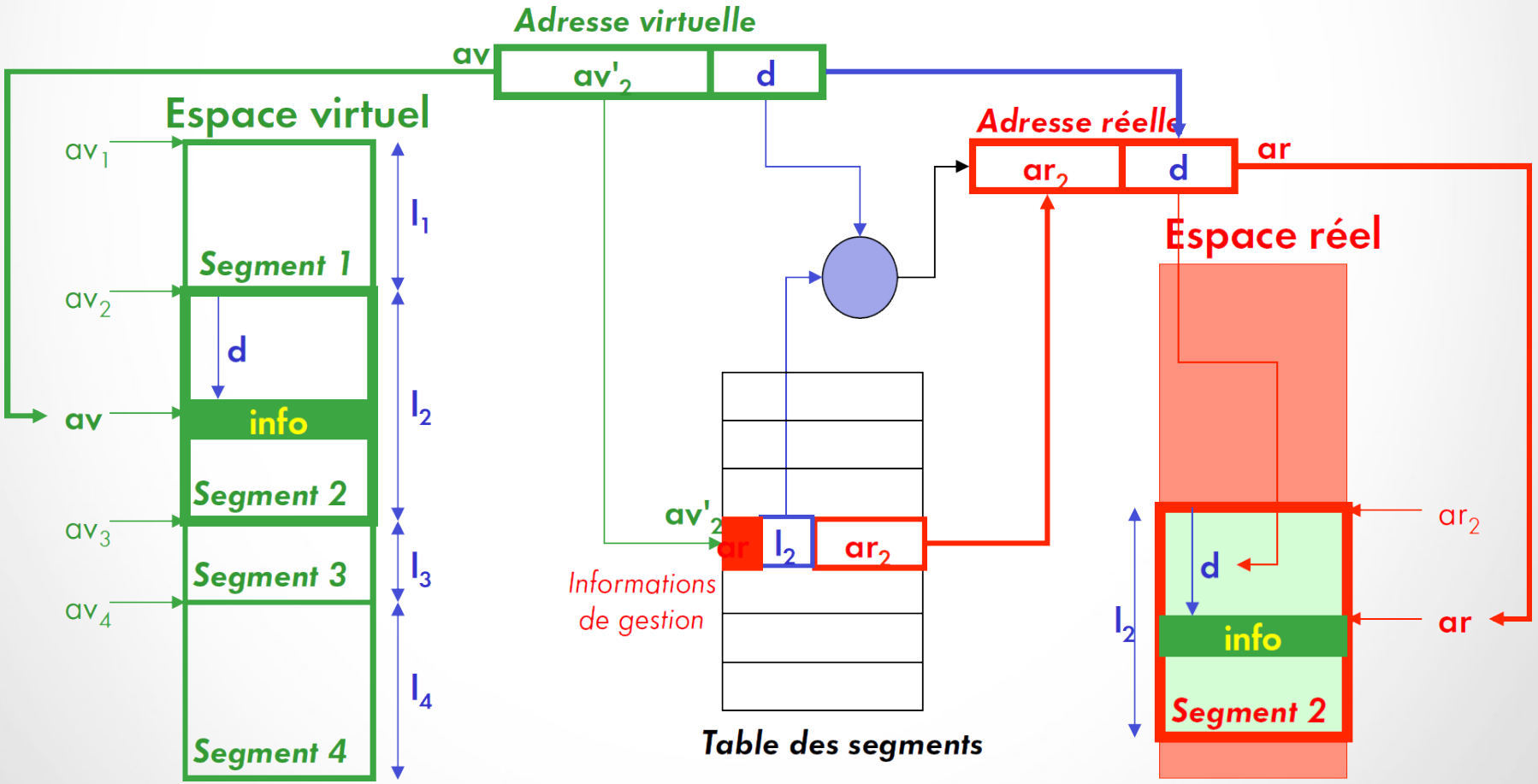
Segmentation et Pagination

- ✓ Deux modes de gestion de la mémoire virtuelle
  - Segmentation
    - Espace d'adressage décomposé en segments de tailles variables
    - Les segments peuvent avoir une signification fonctionnelle
      - Code, Données, Pile...
    - Le mécanisme de traduction doit vérifier la taille
    - L'allocation en mémoire physique est délicate
  - Pagination
    - Espace d'adressage décomposé en pages de taille fixe
    - Le découpage est aveugle (pas de sens fonctionnel)
    - Les mécanismes de traduction et d'allocation sont simples
    - Le remplacement de pages est délicat
- ✓ Systèmes hybrides
  - Pagination segmentée
  - Segmentation paginée (modèle Linux)

Segmentation

- ✓ Vue cohérente avec celle de l'utilisateur
  - L'exemple le plus connu est l'Intel 8086 et ses quatre registres :
    - CS, pour *Code Segment* : pointe vers le segment contenant le programme courant
    - DS, pour *Data Segment* : pointe vers le segment contenant les données du programme en cours d'exécution
    - ES, pour *Extra Segment* : pointe vers le segment dont l'utilisation est laissée au programmeur
    - SS, pour *Stack Segment* : pointe vers le segment contenant la pile
- ✓ Association de la protection avec le segment
  - Protection suivant le segment (lecture seule, exécution, ...)
  - Empêcher les accès illégaux (*segmentation fault*)
- ✓ Partage de segments entre processus
  - Quand deux processus pointent sur la même entrée de la table

Illustration Segmentation



Université Nice Sophia Antipolis

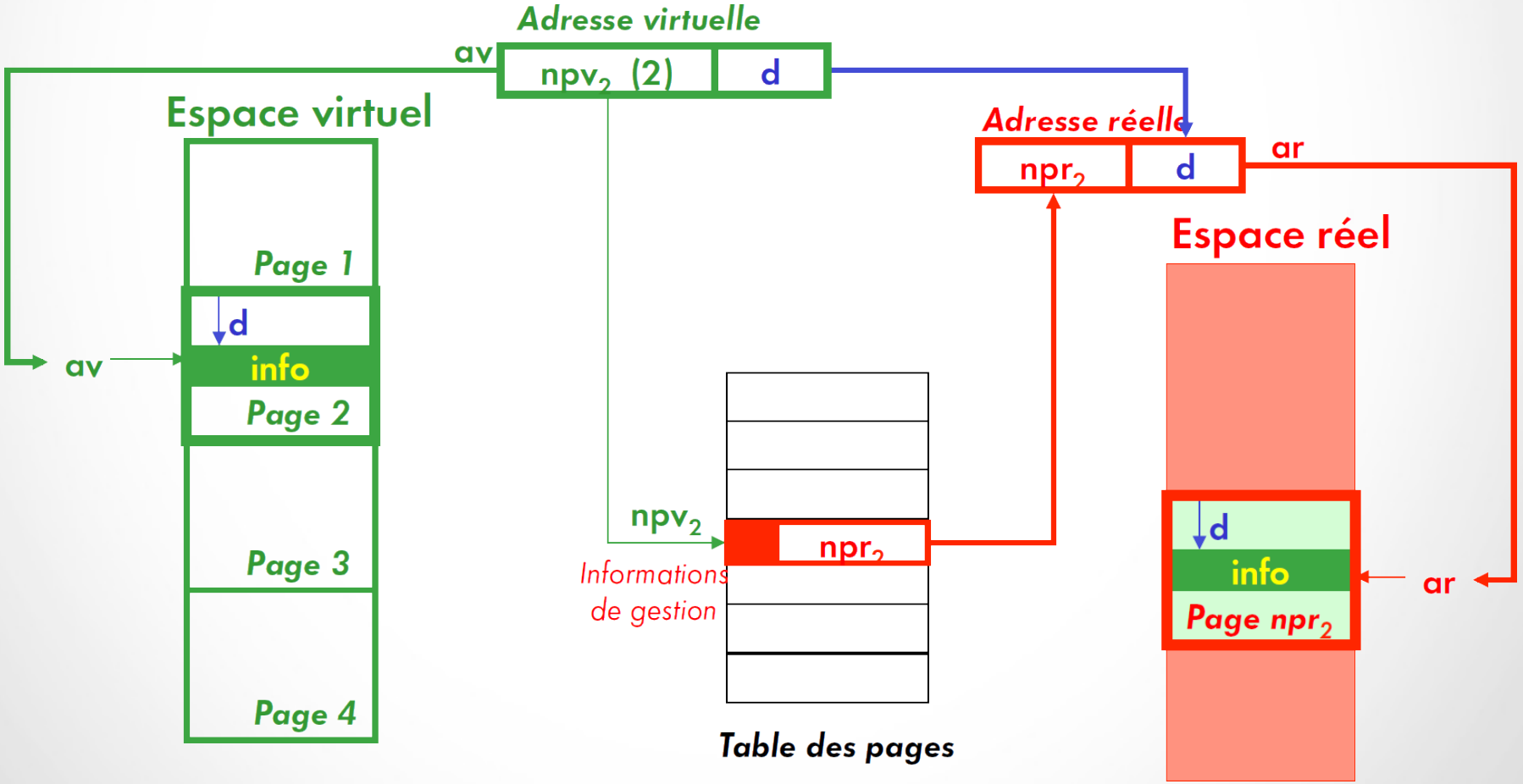
Illustration Segments : cat /proc/*pid*/maps

Adresses	*Droits*	*Déplacement*	*Device*	*I-numéro*		
08048000-08097000	r-xp	00000000	03:02	22438	*zsh*	*text*
08097000-0809b000	rw-p	0004e000	03:02	22438		*data*
0809b000-080ce000	rwxp	00000000	00:00	0		*bss*
40000000-4000a000	r-xp	00000000	03:02	30482	*ld.so*	*text*
4000a000-4000b000	rw-p	00009000	03:02	30482		*data*
40010000-40012000	r-xp	00000000	03:02	30514	*libtermcap.so*	*text*
40012000-40013000	rw-p	00001000	03:02	30514		*data*
40013000-40014000	rw-p	00000000	00:00	0		*bss*
40014000-400a5000	r-xp	00000000	03:02	30486	*libc.so*	*text*
400a5000-400ad000	rw-p	00090000	03:02	30486		*data*
400ad000-400b9000	rw-p	00000000	00:00	0		*bss*
400b9000-400c0000	r-xp	00000000	03:02	30504	*libnss_files*	*text*
400c0000-400c1000	rw-p	00006000	03:02	30504		*data*
400c1000-400c2000	rw-p	00000000	00:00	0		*bss*
bfffd000-c0000000	rwxp	ffffe000	00:00	0	*pile du processus*	

Pagination

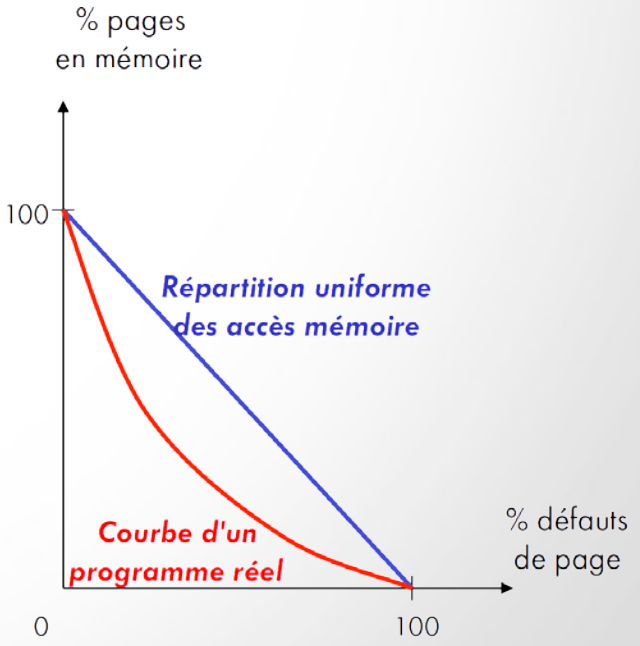
- ✓ La pagination permet d'avoir en mémoire un processus donc les adresses sont non contigües
- ✓ Pour réaliser cela, on partage l'espace d'adressage du processus et la mémoire physique en
  - **Cadres de page** (frames): mémoire physique découpée en zones de taille fixe
    - Taille: puissance de 2 ⇒ entre 512 bytes et 8192 bytes.
  - La mémoire logique est également subdivisée en blocs de la même taille appelés **pages**
    - Taille pages = Taille cadres
    - Adresse logique = numéro de page + déplacement dans le page
  - **Table des pages**: liaison entre numéro de page et cadre de page (une table par processus). On conserve l'emplacement des pages dans une tables de transcodage.
    - Table de pages: traduit l'adresse logique en adresse physique.

Illustration Pagination



Remplacement de Page

- ✓ Quelle page remplacer lorsqu'une nouvelle page doit monter en mémoire physique ?
- ✓ Principe d'optimalité (Peter Denning)
  - Remplacer la page qui ne sera pas utilisée dans le temps le plus long
    - Problème : principe non causal !
    - Approximations liées au principe de localité
- ✓ Principe de localité
  - Agrégation des références
  - Pratiques de programmation structurée
    - Instruction séquentielles
    - Boucles
    - Absence de `goto`
  - Modularité
  - Structures de données
    - Records
    - Tableaux



Université Nice Sophia Antipolis

Algorithmes Remplacement de Page

- ✓ Algorithmes globaux de remplacement de pages
  - FIFO: First In First Out
    - Problème de l'anomalie FIFO
  - LRU: *Least Recently Used*
    - Difficile à implémenter
  - NRU: *Not Recently Used*
    - Le plus courant (souvent confondu avec LRU)
  - NFU: *Not Frequently Used*
    - Incrémente un compteur à chaque utilisation
  - *Aging*
    - Variante de NFU
- ✓ Gestion par processus (« *Working Set* »)
  - Prédiction des pages utiles de chaque processus

Segmentation x Pagination

Considération	Segmentation	Pagination
Connaissance du mécanisme par le programmeur nécessaire ?	Oui	Non
Combien d'adresses linéaires sont présentes ?	Plusieurs	1
L'adressage total peut-il dépasser la mémoire physique?	Oui	Oui
Le code et les données peuvent-ils être distingués et protégés séparément ?	Oui	Non
La taille peut-être être changée dynamiquement ?	Oui	Non
Le partage des procédures entre les utilisateurs est-il facilité?	Oui	Non
Pourquoi cette technique a été inventée ?	Séparation Partage Protection	Augmentation mémoire physique

Mémoire Virtuelle: Avantages et Inconvénients

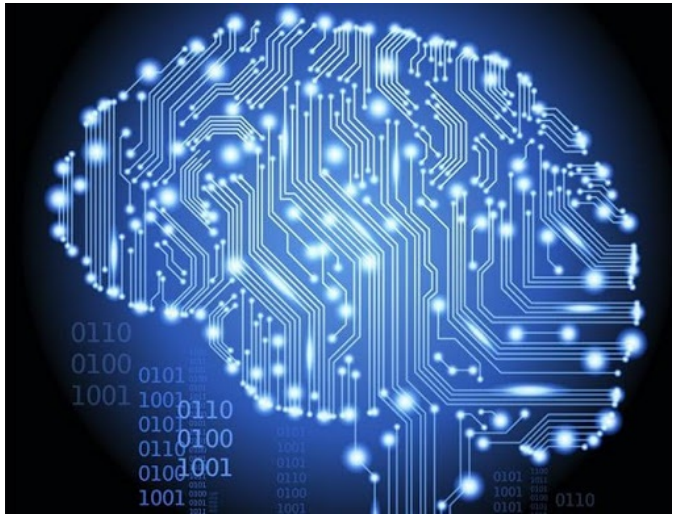
- ✓ Avantages
  - Libération de la contrainte de la taille de l'espace physique
  - Taux de multi-programmation élevé
  - « Relogeabilité » immédiate
  - Partage de code et de données ; « mapping de fichiers »

- ✓ Inconvénients
  - Perte de performances
  - D'autant plus efficace que l'on a plus de mémoire physique
  - Réglage délicat : taille des pages, remplacement

Université Nice Sophia Antipolis

Conclusion sur les Systèmes et la Gestion Mémoire

- ✓ Type de gestionnaires
  - Système mono-programmé ou multi-programmé
  - Avec ou sans MMU (traduction matérielle en adresses physiques)
  - Avec ou sans pagination
  - Avec ou sans segmentation
  - Avec ou sans échange avec la mémoire secondaire (swaping sur disque dur)
- ✓ Ex: Linux
  - Système multi-programmé
  - Avec (Linux vanilla) ou sans MMU (uClinux)
  - Avec Segmentation Paginée
  - Avec ou sans mémoire secondaire (ajout à la demande)



Gestion Dynamique de la Mémoire

D'un point de vue Espace Utilisateur

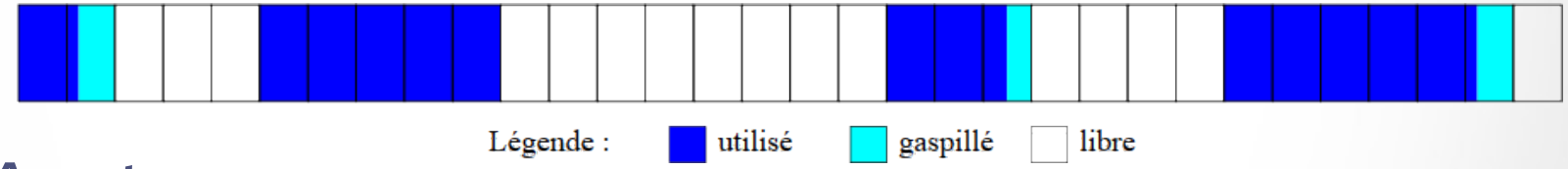
Gestion du tas

Rappels

- ✓ Allocation statique
  - s'applique aux variables locales
  - stockage dans la pile (*stack*)
  - allocation en entrant dans la fonction/procédure
  - libération en sortant de la fonction/procédure
  - gérée automatiquement par le compilateur
- ✓ Allocation dynamique
  - stockage dans le tas (*heap*)
  - allocation/libération à la demande du programmeur
    - `malloc` / `free`
    - `new` / `delete`
- ✓ Gestion de l'allocation dynamique:
  - Bibliothèques standards (`malloc` / `free`)
  - Langage lui-même (`new` / `delete`)
  - De manière implicite (ramasse-miettes)

Méthode: Carte de Bits

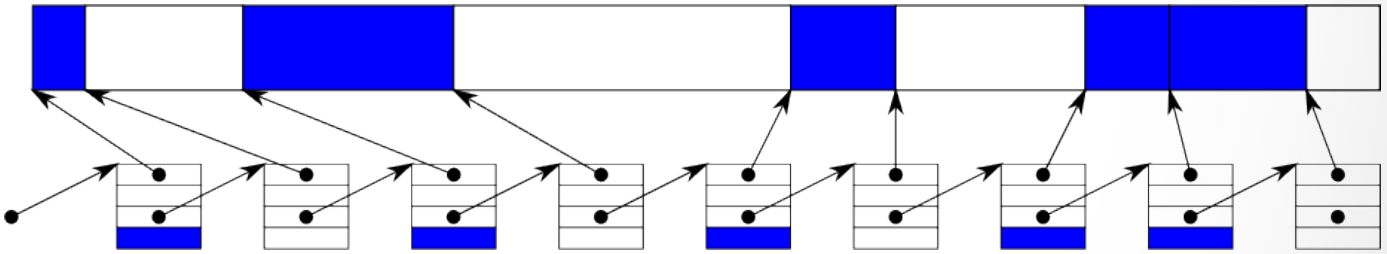
- ✓ Carte de Bits
  - le tas est subdivisé en blocs de taille fixe
  - on établit « une carte » (un tableau de bits) dont chaque bit représente l'état (libre/alloué) d'un bloc



  - Avantage
    - Surcoût mémoire raisonnable
  - Inconvénients
    - Allocation coûteuse en temps (recherche d'espace libre)
    - Fragmentation interne:
      - Certains blocs marqués comme occupés ne sont pas totalement utilisés
  - Peu utilisé en pratique

Méthode: Liste Chainée

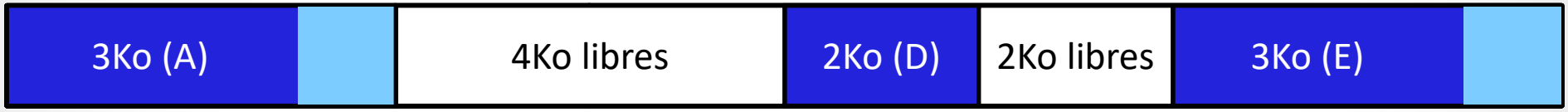
- ✓ Liste chaînée
  - Liste chaînée des segments libres et occupés
  - Exemple de représentation possible: chaque maillon = une zone (adresse, taille), son état (alloué ou libre) et pointe vers le maillon suivant



  - Lors d'une allocation mémoire, si plusieurs zones libres de taille suffisante sont disponibles, laquelle choisir?
    - *First fit* : premier emplacement suffisant (méthode la plus simple et la plus rapide)
    - *Best fit* : cherche la zone libre la plus petite, c.à.d. la plus *proche* de la taille recherchée (laisse des petits trous)
    - *Worse fit* : cherche la zone libre la plus grande (fragmentation à terme)

Méthode: Buddy

- ✓ Buddy
  - Méthode basée sur le principe dichotomique, permettant de diviser la taille du tas par des puissances de 2



  - Avantages
    - Efficace en temps (logarithmique par rapport à la taille de la mémoire)
    - Malgré la fragmentation, efficace
  - Inconvénients
    - Fragmentation interne (contrainte sur la taille des zones)
    - Fragmentation externe
- ✓ Méthode utilisée dans le noyau Linux

Université Nice Sophia Antipolis

Gestion de l'Espace Virtuel des Processus

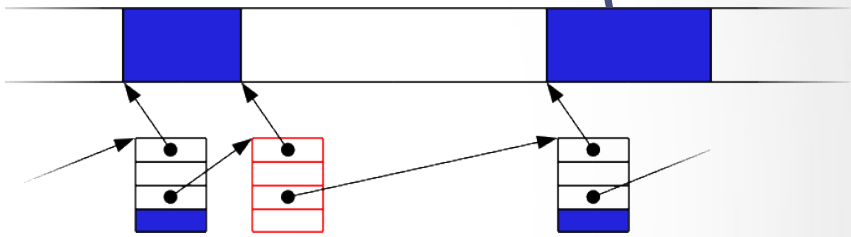
- ✓ Demandes de création/modification de l'espace virtuel utilisateur par les processus en mode utilisateur :
  - Création initiale des zones du code, des données et de la pile
    - `execve()`
  - Chargement des bibliothèques dynamiques
    - à la fin du `execve()`
  - Extension du tas vers le haut
    - `brk()` et indirectement `malloc()`
  - Mémoire partagée avec les fonctions IPC
    - `shmat()` et `shmdt()`
  - Projection de fichier avec les fonctions
    - `mmap()` et `munmap()`
  - ...

Annexes

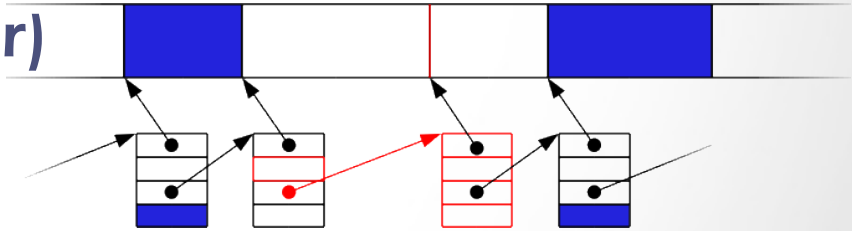
Pour comprendre certains mécanismes plus en détail

Détails Gestion Liste Chainée : Allocation

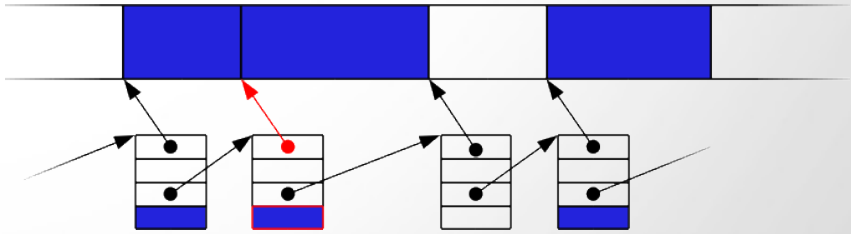
1. On recherche une zone libre de taille suffisante (de taille ≥ à la taille réclamée)



2. On fractionne la zone libre en deux (la première ayant exactement la taille à allouer)

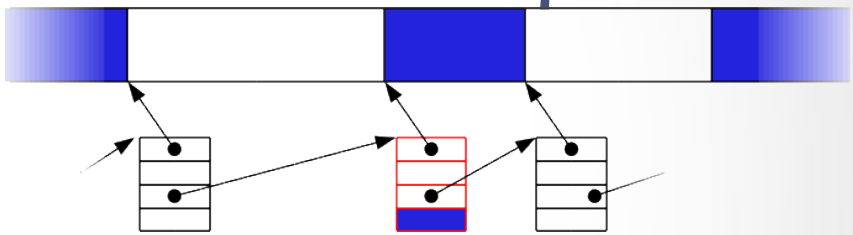


3. On marque la première comme allouée et on retourne son adresse

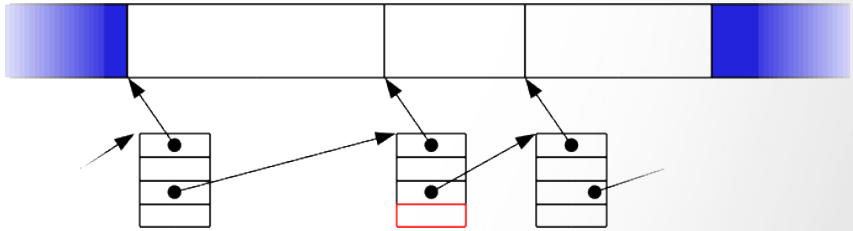


Détails Gestion Liste Chainée : Libération

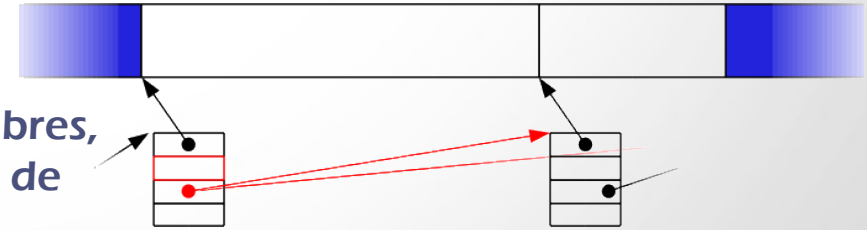
1. On parcours la liste jusqu'à trouver le maillon pointant vers la zone à libérer



2. On marque ce maillon comme libre. Attention à la **fragmentation externe**



3. Le cas échéant, on fusionne la zone libre avec les zones libres voisines

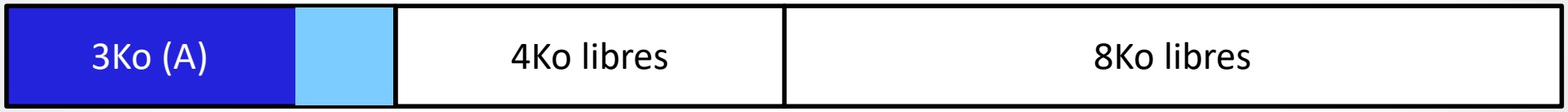


Seuls les voisins immédiats peuvent être libres, donc la fusion ne pénalise pas l'opération de libération (temps constant).

Détails Méthode Buddy: Allocations de blocs

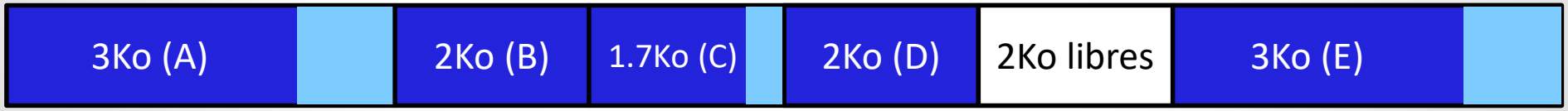
Tas de 16 Ko

Allocation d'un bloc de 3Ko (Bloc A)



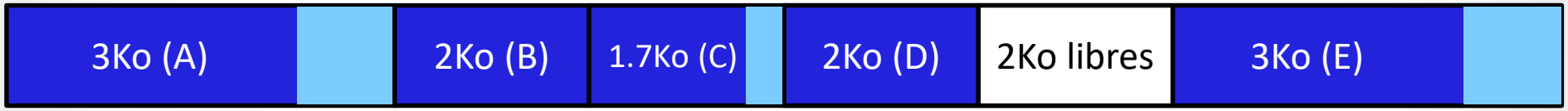
Allocation d'un bloc de 2Ko (Bloc B)

Allocation d'un bloc de 1.7Ko (C), 2Ko (D) et 3Ko (E)

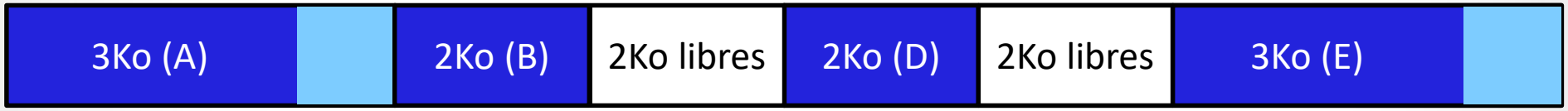


Détails Méthode Buddy: Libération et Fusion de blocs

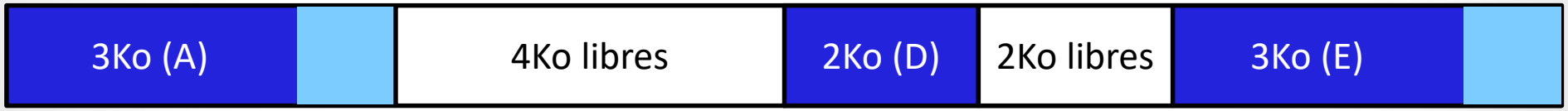
Etat courant



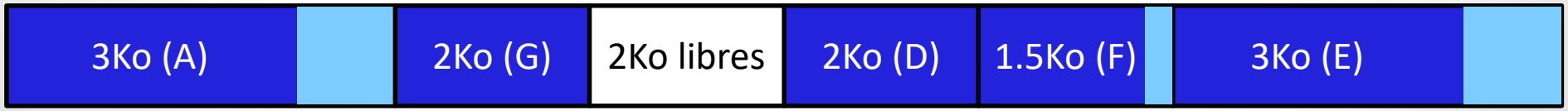
Libération de (C)



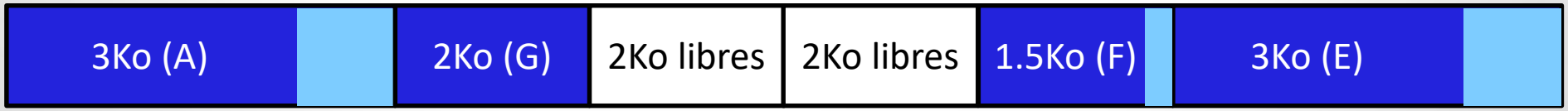
Si (B) est libéré alors fusion des blocs



Allocation d'un bloc de 1.5Ko (F) et d'un bloc de 2Ko (G)

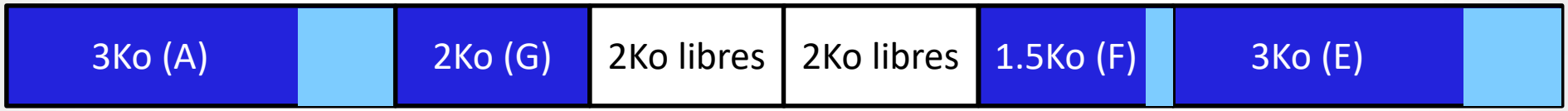


Mais il n'est possible de fusionner que des blocs séparés. Libération de (D)

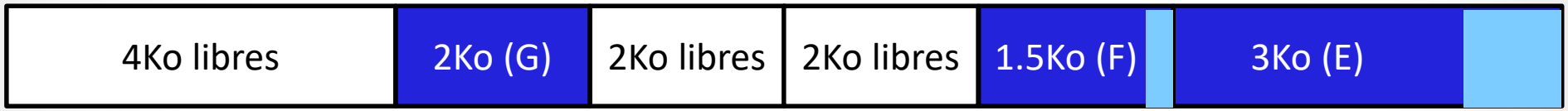


Détails Méthode Buddy: Encore un pour la route...

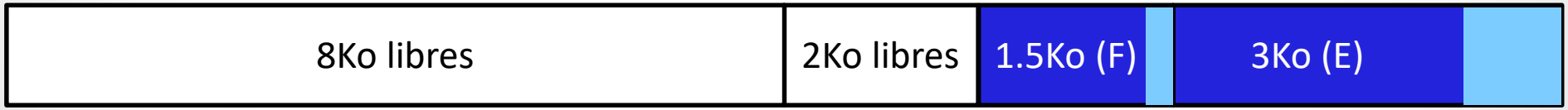
Etat courant



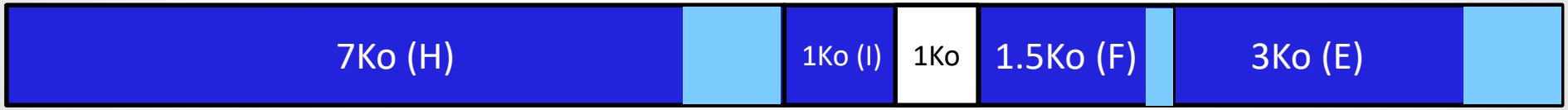
Libération de (A)



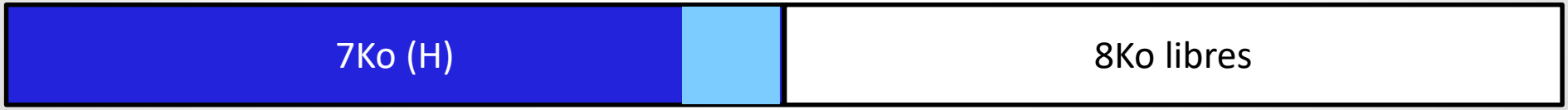
Si (G) est libéré alors fusion des blocs



Allocation d'un bloc de 7Ko (H) et d'un bloc de 1Ko (I)



Libération des blocs (F), (E), (I). Etc. etc. etc.



Pagination: Conséquence sur le Programmation

- ✓ Pagination: abstraction invisible au programmeur mais a des conséquences sur la gestion de la mémoire
- ✓ Ex: supposons que la taille d'une page soit de 128 sizeof(int) et qu'un tableau soit stocké ligne par ligne:

tab[0][0]	tab[0][1]	…	tab[0][128]	Première page
…				Autres pages
tab[127][0]	tab[127][1]	…	tab[127][127]	Dernière page

- ✓ Les deux codes suivants ne sont pas équivalents en termes de gestion mémoire (le premier peut nécessiter des swaps)

```
int tab[128][128] ;
for(int j = 0 ; j<128 ; j++)
   for(int i = 0 ; i<128 ; i++)
      tab[i][j] = 0 ;
/* parcours les pages */
```

```
int tab[128][128] ;
for(int i = 0 ; i<128 ; i++)
   for(int j = 0 ; j<128 ; j++)
      tab[i][j] = 0 ;
/* reste dans une page  */
```