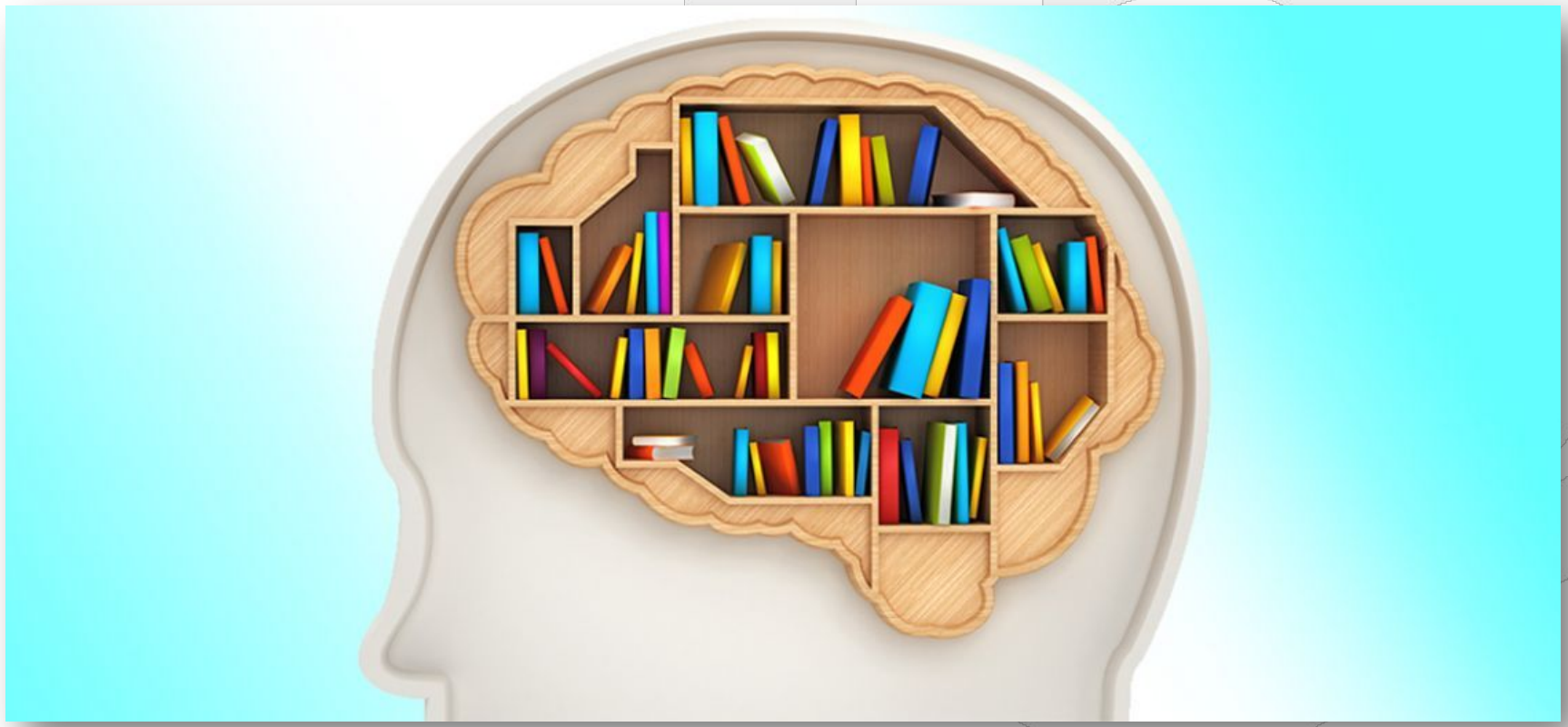
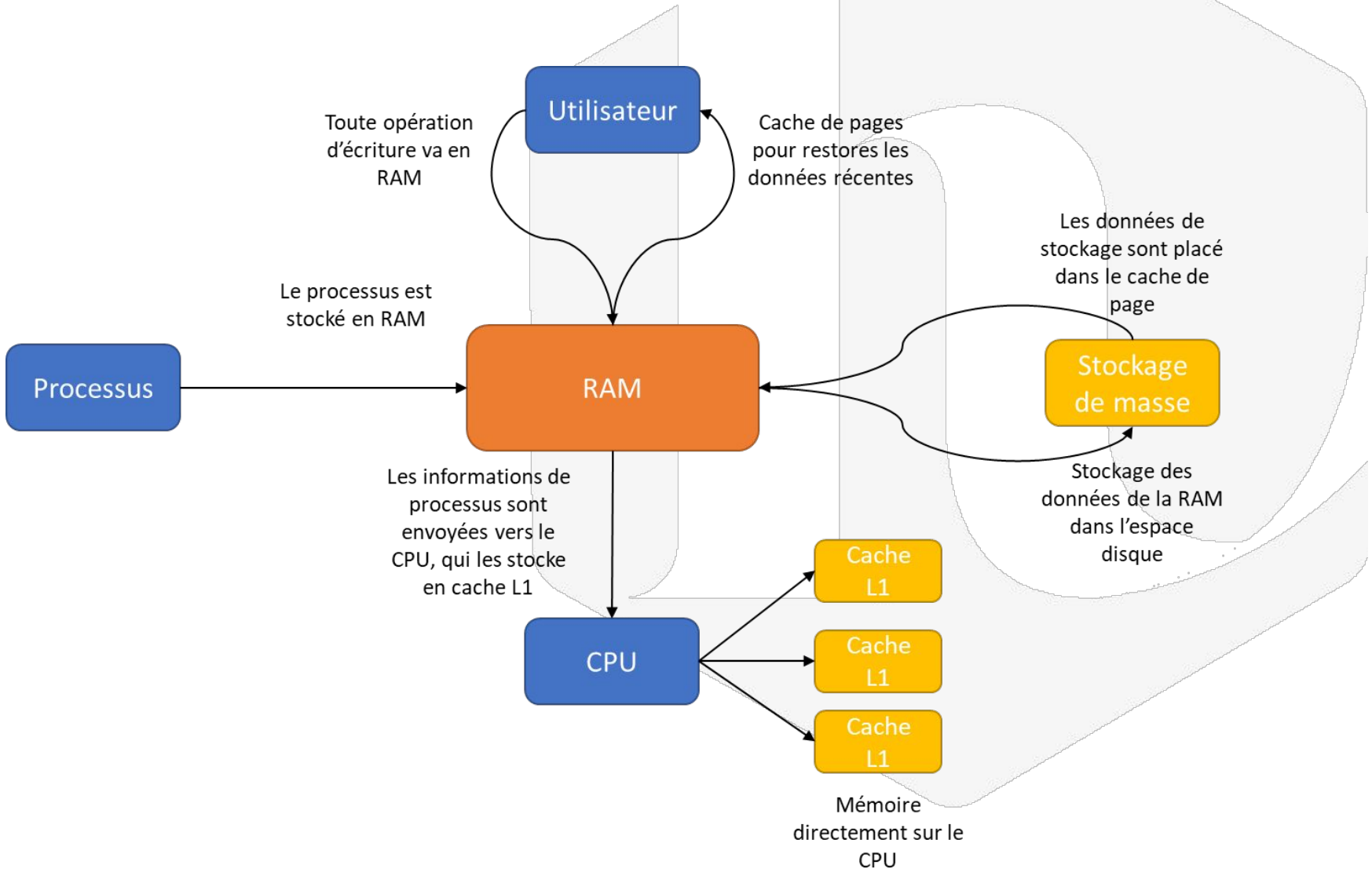
hexagone
École Supérieure d'Informatique

La mémoire

Des cases et des tiroirs

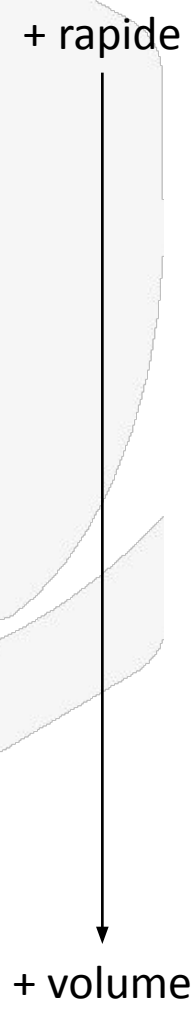


Les mémoires

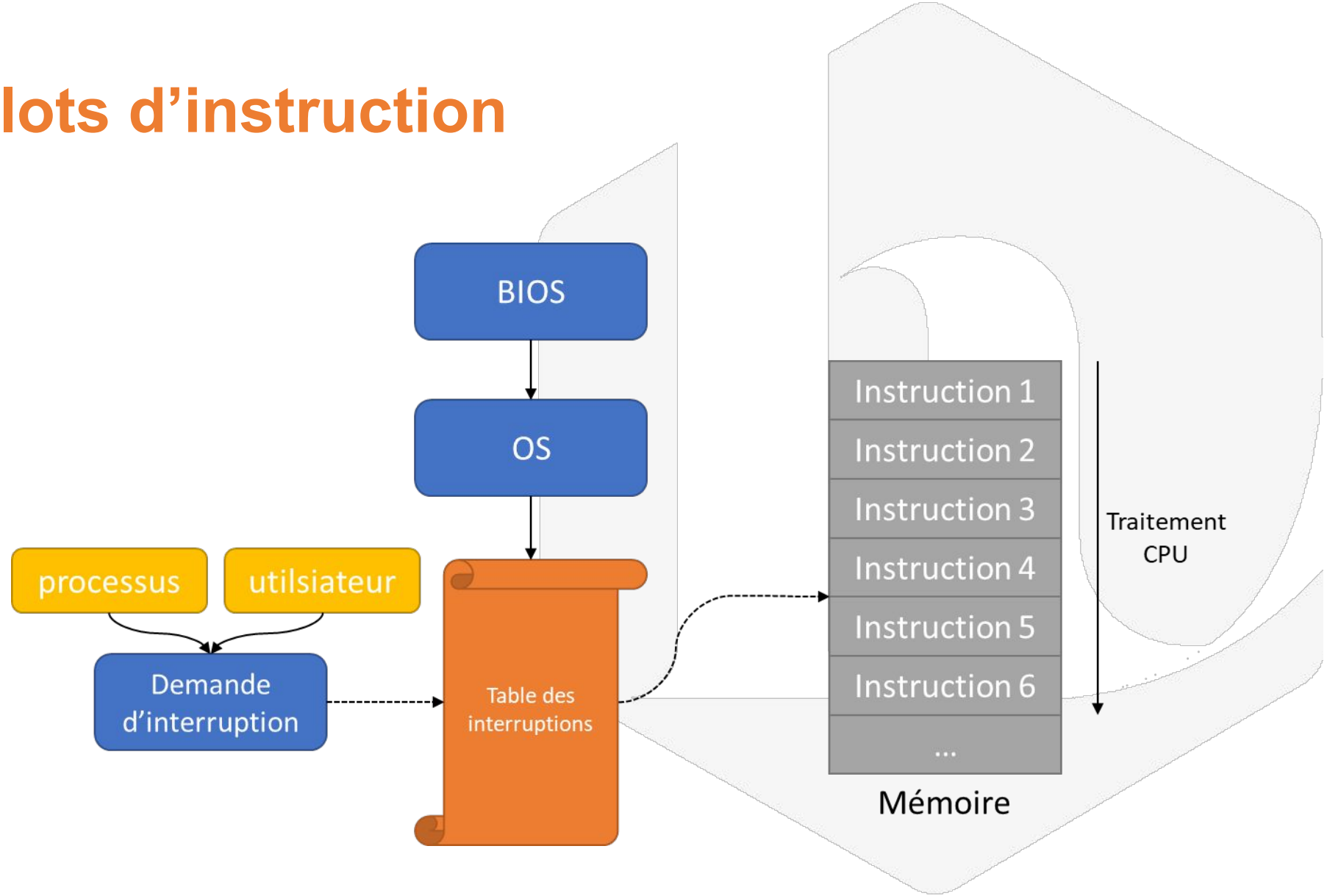


Hiérarchie des mémoires

- registres processeur
  - directement utilisables pour faire des calculs
  - stocker le contexte CPU d'une tâche (quelques dizaines d'octets)
- cache matériel primaire (cache L1 - quelques dizaines de Ko)
- cache matériel secondaire (quelques centaines de Ko)
- mémoire principale RAM (quelques centaines de Mo à quelques Go)
- disques durs locaux (quelques Go, voire To)
- espaces de stockage distants
  - disques de serveurs de fichiers, etc…



Flots d’instruction



3 acteurs principaux

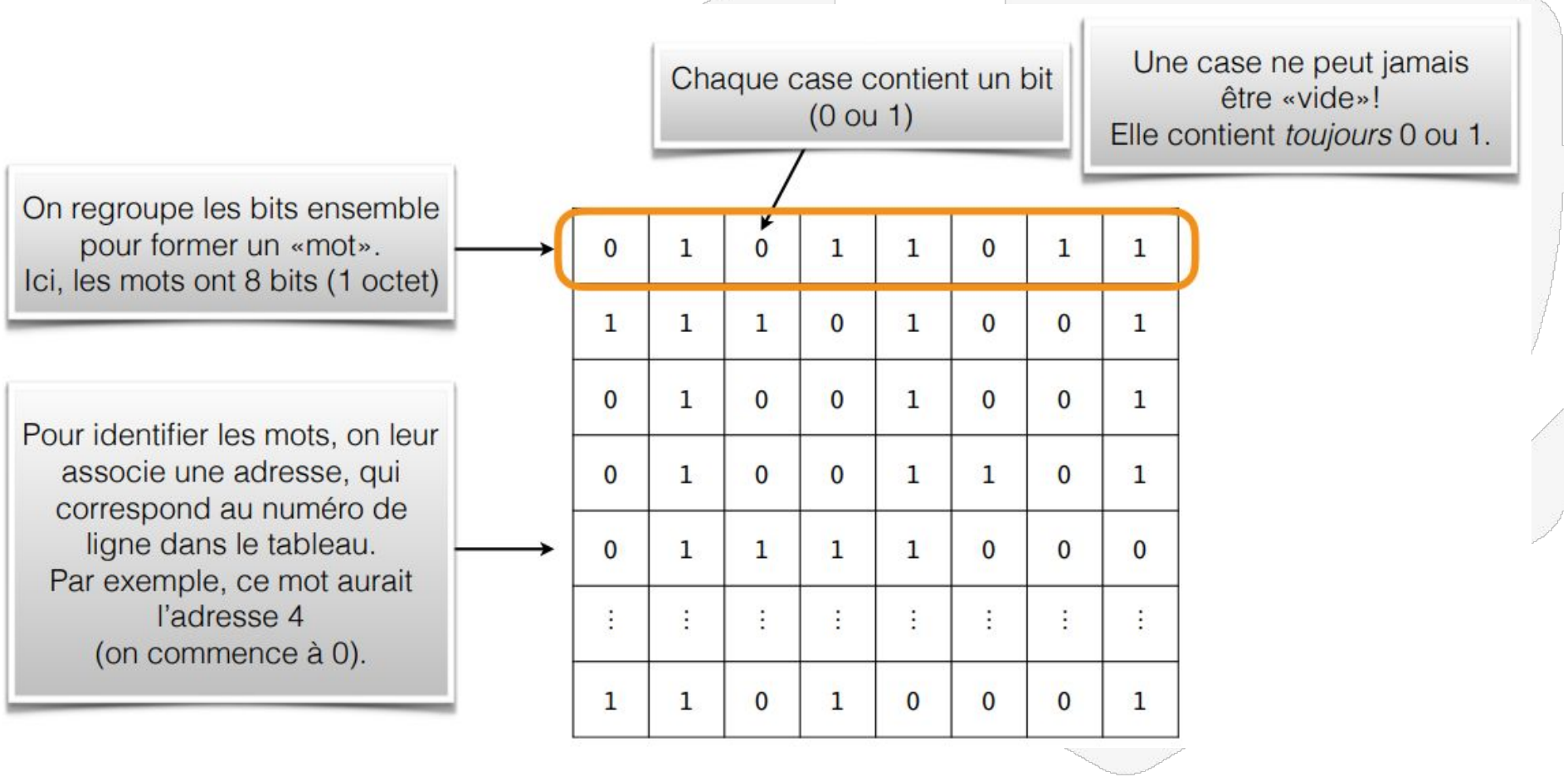
- La mémoire *Random Access Memory*
  - stocke le code à exécuter et les données associées
- Le CPU
  - réalise l'exécution du code
- Le chipset
  - Relie le CPU à la mémoire et aux périphériques

La mémoire

- RAM composée de cellules d'un octet

- accessibles par le CPU :
  - Individuellement
  - par mots (blocs de 2, 4 ou éventuellement 8 octets)
  - adresse identifiant de manière unique la cellule ou la première cellule du mot
  - taille des adresses dépend du CPU : 16 bits, 32 bits, 64 bits

La mémoire


Chaque case contient un bit (0 ou 1)
Une case ne peut jamais être «vide»! Elle contient *toujours* 0 ou 1.
On regroupe les bits ensemble pour former un «mot». Ici, les mots ont 8 bits (1 octet)
Pour identifier les mots, on leur associe une adresse, qui correspond au numéro de ligne dans le tableau. Par exemple, ce mot aurait l'adresse 4 (on commence à 0).
0 1 0 1 1 0 1 1
1 1 1 0 1 0 0 1
0 1 0 0 1 0 0 1
0 1 0 0 1 1 0 1
0 1 1 1 1 0 0 0
1 1 0 1 0 0 0 1

La mémoire

Ici, l'exemple utilise 8 bits d'adresse, et des mots de 8 bits. Cependant, différents systèmes peuvent avoir différentes valeurs!

On utilise un nombre (fini et prédéterminé) de bits pour représenter l'adresse.

Adresse (sur 8 bits)	Donnée (mot de 8 bits)							
	b_7	b_6	b_5	b_4	b_3	b_2	b_1	b_0
0x00	0	1	0	1	1	0	1	1
0x01	1	1	1	0	1	0	0	1
0x02	0	1	0	0	1	0	0	1
0x03	0	1	0	0	1	1	0	1
0x04	0	1	1	1	1	0	0	0
⋮	⋮	⋮	⋮	⋮	⋮	⋮	⋮	⋮
0xFF	1	1	0	1	0	0	0	1

La mémoire

- La mémoire est décrite grâce à 2 informations indépendantes :
  - le nombre d'adresses possibles
    - 2^8 = 256 adresses
  - la taille des mots de la mémoire
    - 8 bits (1 octet)
- taille totale de la mémoire = quantité totale de bits pouvant être stockée dans la mémoire.

- Calcul :
  - taille mémoire = nombre d'adresses × taille d'un mot
  - = 2^8 × 1 octet = 256 octets

Adresse (sur 8 bits)	Donnée (mot de 8 bits)							
	b_7	b_6	b_5	b_4	b_3	b_2	b_1	b_0
0x00	0	1	0	1	1	0	1	1
0x01	1	1	1	0	1	0	0	1
0x02	0	1	0	0	1	0	0	1
0x03	0	1	0	0	1	1	0	1
0x04	0	1	1	1	1	0	0	0
⋮	⋮	⋮	⋮	⋮	⋮	⋮	⋮	⋮
0xFF	1	1	0	1	0	0	0	1

La mémoire : petit calcul

- Une mémoire stocke des mots de 8 bits (1 octet) et possède 2^{16} adresses.
  - Quelle est la taille totale de la mémoire en kilo-octets (Ko) ?
    - Rappel : taille mémoire = nombre d'adresses × taille d'un mot
    - Rappel : 1 kilo-octet = 210 octets

La mémoire : petit calcul

- Une mémoire stocke des mots de 8 bits (1 octet) et possède 2^{16} adresses.
  - Quelle est la taille totale de la mémoire en kilo-octets (Ko) ?
    - Rappel : taille mémoire = nombre d'adresses × taille d'un mot
    - Rappel : 1 kilo-octet = 210 octets

- réponse :
  - taille d'un mot: 1 octet
  - nombre total de mots: 216
  - taille mémoire = 1 x 2^{16} = 2^{16} octets = 2^{6} Ko = 64 Ko

La mémoire : petit calcul

- Une mémoire stocke des mots de 16 bits (2 octets) et nécessite 8 bits pour les adresser.
  - Quelle est la taille totale de la mémoire en octets?

La mémoire : petit calcul

- Une mémoire stocke des mots de 16 bits (2 octets) et nécessite 8 bits pour les adresser.
  - Quelle est la taille totale de la mémoire en octets?

- réponse :
  - taille d'un mot: 2 octets
  - bits pour les adresses: 8 bits
  - nombre d'adresses: $2^8 = 256$
  - Taille mémoire = $2 \times 2^8 = 2^9$ octets = 512 o

La mémoire : petit calcul

- Une mémoire possède une taille totale de 32 Mo et peut stocker des mots de 32 bits.
  - Combien de bits a-t-on besoin pour représenter les adresses dans cette mémoire?
  - Quelles sont les adresses minimales et maximales de cette mémoire exprimées en hexadécimal?

La mémoire : petit calcul

- Une mémoire possède une taille totale de 32 Mo et peut stocker des mots de 32 bits.
  - Combien de bits a-t-on besoin pour représenter les adresses dans cette mémoire?
  - Quelles sont les adresses minimales et maximales de cette mémoire exprimées en hexadécimal?

- réponse :
  - Taille mémoire : 32 Mo
  - Taille d'un mot : 32 bits = 4o (octets)
  - Nombre total de mots = 32 Mo / 4o = 32 x 2^{20} / 4 = 2^{25} / 2^{2} = 2^{23}
    - nous avons besoin de $\log_2(2^{23})$ = 23 bits pour représenter les adresses.

  - Les adresses minimales et maximales sont:
    - minimale: 0b00000000000000000000000 (23 bits) = 0x000000
    - minimale: 0b11111111111111111111111 (23 bits) = 0x7FFFFF

Typologie de la mémoire

- Les mémoires peuvent être:
  - volatiles: perdent leur contenu lorsqu'elles perdent leur alimentation
  - non-volatiles: conservent leur contenu même sans alimentation

- Les mémoires volatiles peuvent être :
  - statiques: n'ont pas besoin d'être lues pour conserver leurs valeurs
  - dynamiques: nécessitent un rafraîchissement périodique des données.
    - Si les données d'une mémoire dynamique ne sont pas "lues" régulièrement, elles s'effacent.

- ROM: ne peut pas être écrite (*Read Only Memory*)
- RAM: peut être écrite (*Random Access Memory*)

Le bus

- Un bus est un groupe de lignes électriques qui relie le CPU aux autres composants.
- Chaque ligne peut transférer un bit d'information à la fois.
- Bus d'adresse
  - indique l'emplacement de la mémoire visé par la transaction sur le bus
  - contrôle par CPU
  - taille du bus d'adresse (nombre de lignes) □ quantité maximum de mémoire utilisable

Le bus

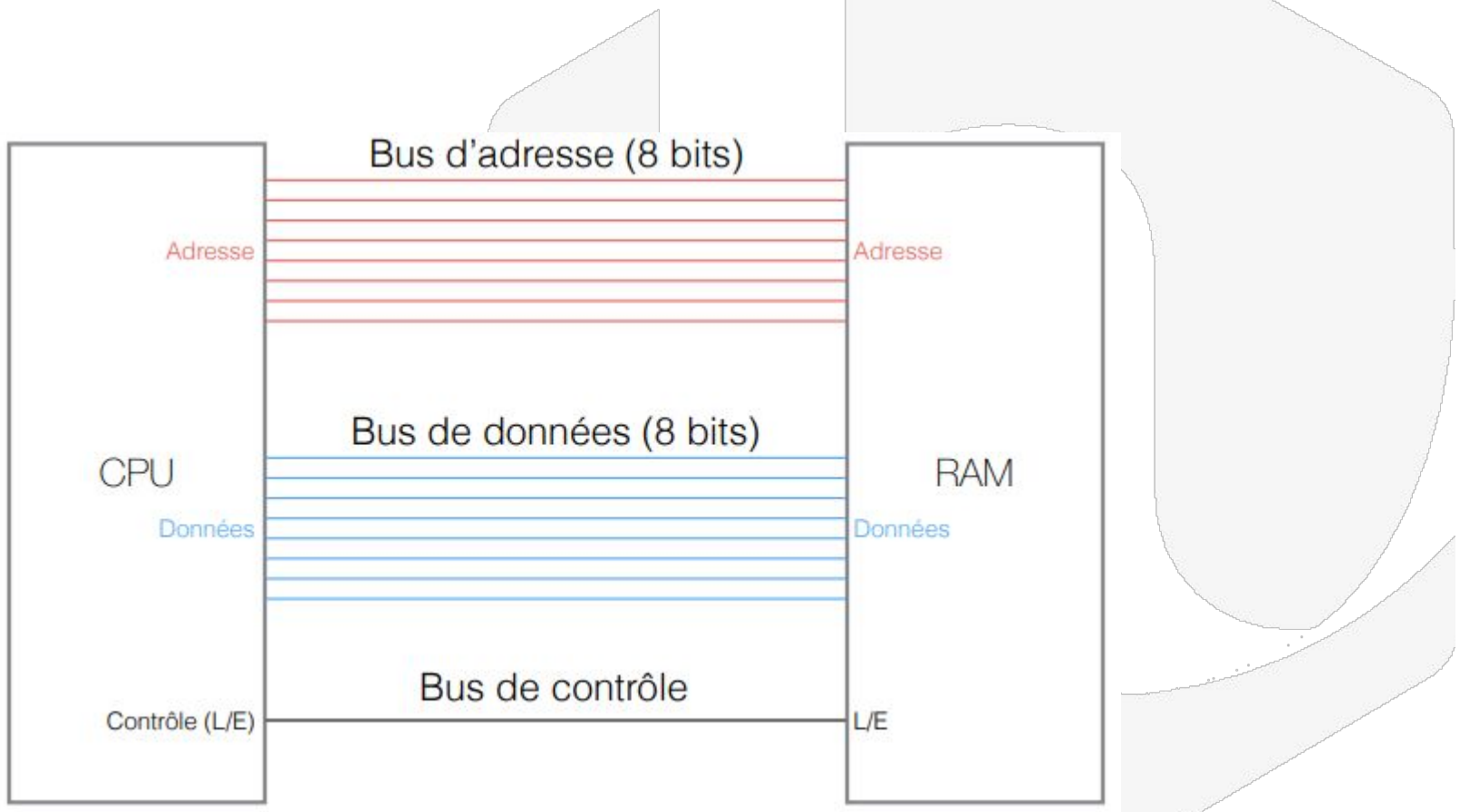
- Un bus est un groupe de lignes électriques qui relie le CPU aux autres composants.
- Chaque ligne peut transférer un bit d'information à la fois.
- Bus d'adresse
  - indique l'emplacement de la mémoire visé par la transaction sur le bus
  - contrôle par CPU
  - taille du bus d'adresse (nombre de lignes) □ quantité maximum de mémoire utilisable
    - 8 « lignes » = 2^8 adresses utilisables

Le bus

- Bus de données
  - Transfert des données.
    - peuvent circuler dans les deux sens, mais elles ne circulent que dans un seul sens à la fois.
  - taille du bus de données (le nombre de lignes) détermine la grandeur maximale des mots pouvant être transférés d'un coup.
  - je dois écrire 2 caractères ASCII (8 bits) en mémoire…
    - Chaque caractère nécessite 8 bits et le bus de données a 8 bits.
    - Il faudra donc effectuer 2 transferts pour écrire les 2 caractères en mémoire.

- Bus de contrôle
  - utilisation des bus de données et d'adresse.
  - gérer la direction des données sur le bus des données (lecture ou écriture)
  - dispose d'une horloge, qui détermine la vitesse à laquelle les données peuvent être transférées et qui synchronise les opérations

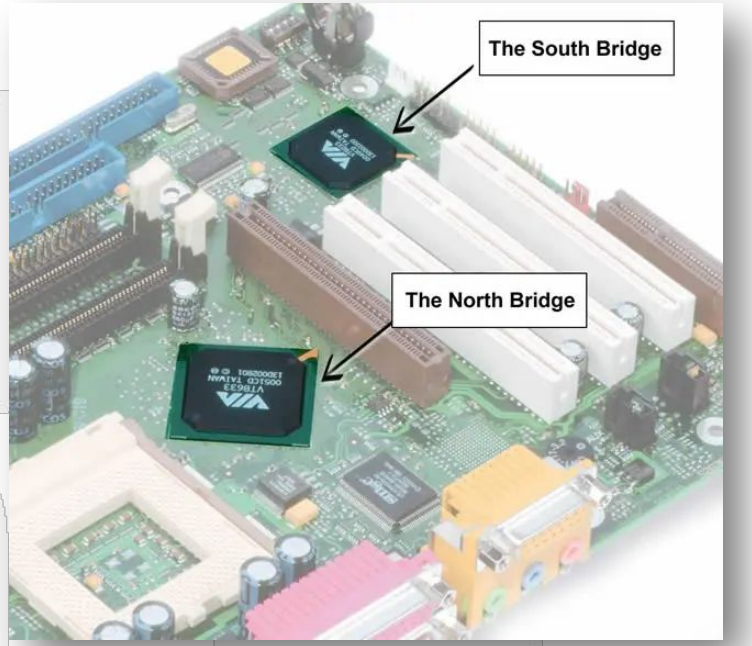
Le bus


Bus d'adresse (8 bits)
Adresse
Adresse
Bus de données (8 bits)
CPU
RAM
Données
Données
Bus de contrôle
Contrôle (L/E)
L/E

La MMU

- *Memory Management Unit*
  - *modification uniquement en ring 0*
- sur la puce du CPU, en coupure de l'accès à la RAM
- traduction des adresses logiques utilisées par le code en adresses physiques transmises sur le bus mémoire
- interdire tout accès à certaines plages de mémoire physique
- cloisonnement mémoire :
  - entre processus du système d'exploitation
  - entre l'espace mémoire noyau et la couche utilisateur

Accès à la mémoire



- *Northbridge*

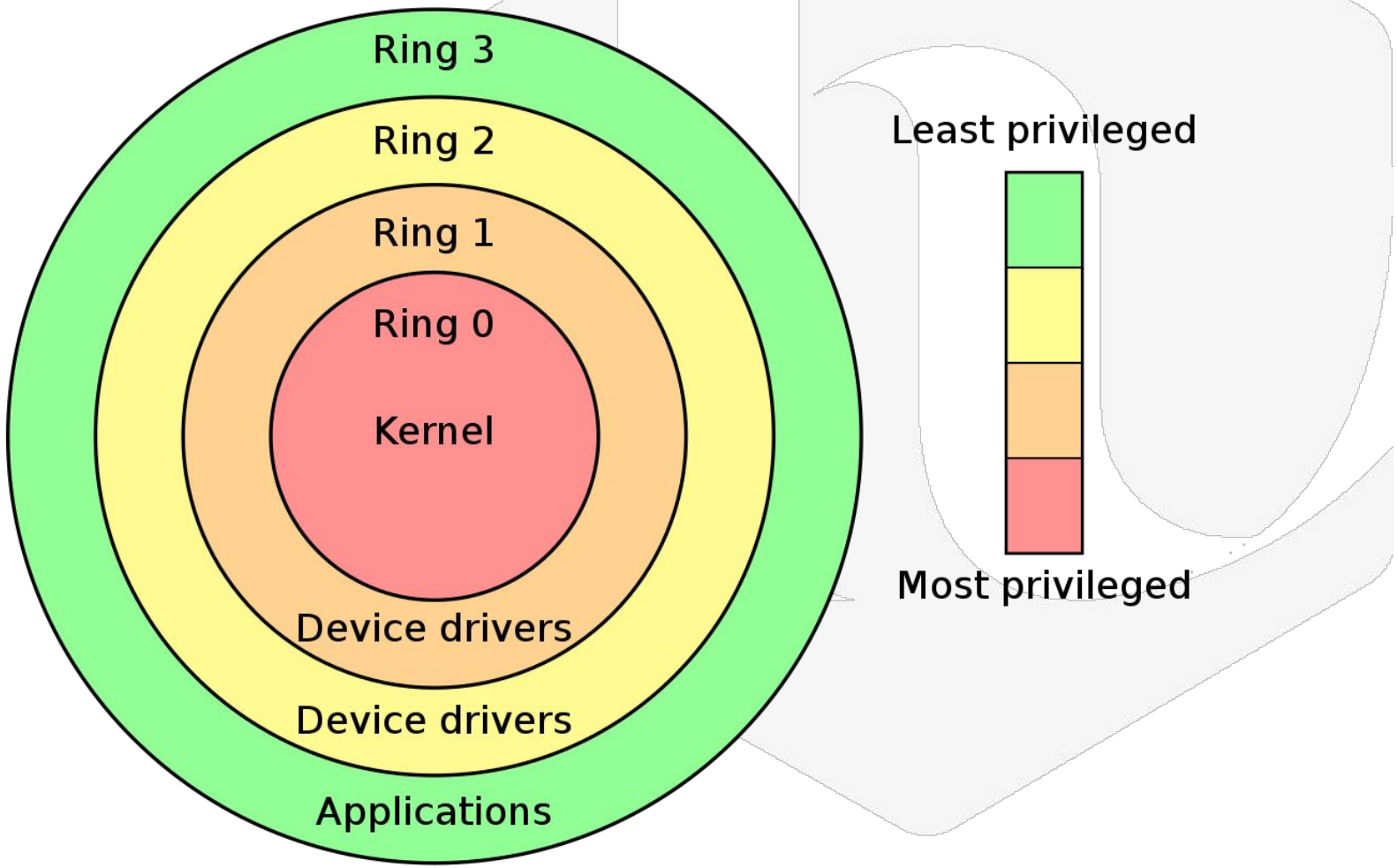
- Utilise :
  - des ports d'entrées/sorties (PIO) pour les accès rapides
  - des entrées/sorties projetées en mémoire (MMIO) pour les accès classiques
  - des accès directs à la mémoire (DMA) pour les périphériques (accès sans passer par le CPU)

- Nota : *Southbridge* accès aux périphériques

Les privilèges mémoire

- le CPU associe au flot d'instructions un niveau de privilèges courant (CPL)
- Distinction entre couche utilisateur et noyau du système d'exploitation
- Non lié aux droits des fichiers !
  - processus root et processus non privilégié même traitement !
- 4 niveaux de CPL :
  - ring 0 = noyau du système , privilèges matériels maximaux ;
  - rings 1 et 2 = surcouches noyau moins privilégiées (non utilisé) ;
  - ring 3 = ensemble des tâches en couche utilisateur.
- C’est le MMU qui gère les passages entre ring.

Les privilèges mémoire



Appel système

- un ring moins privilégié demande l'exécution d'une fonction dans un ring plus privilégié
  - Besoin d’intégrité et de confiance
- Chaque ring peut accéder aux zones mémoire affectées aux rings moins privilégiés
  - permet de lire ou d'écrire des données liées à un appel système

La segmentation

- Vision logicielle

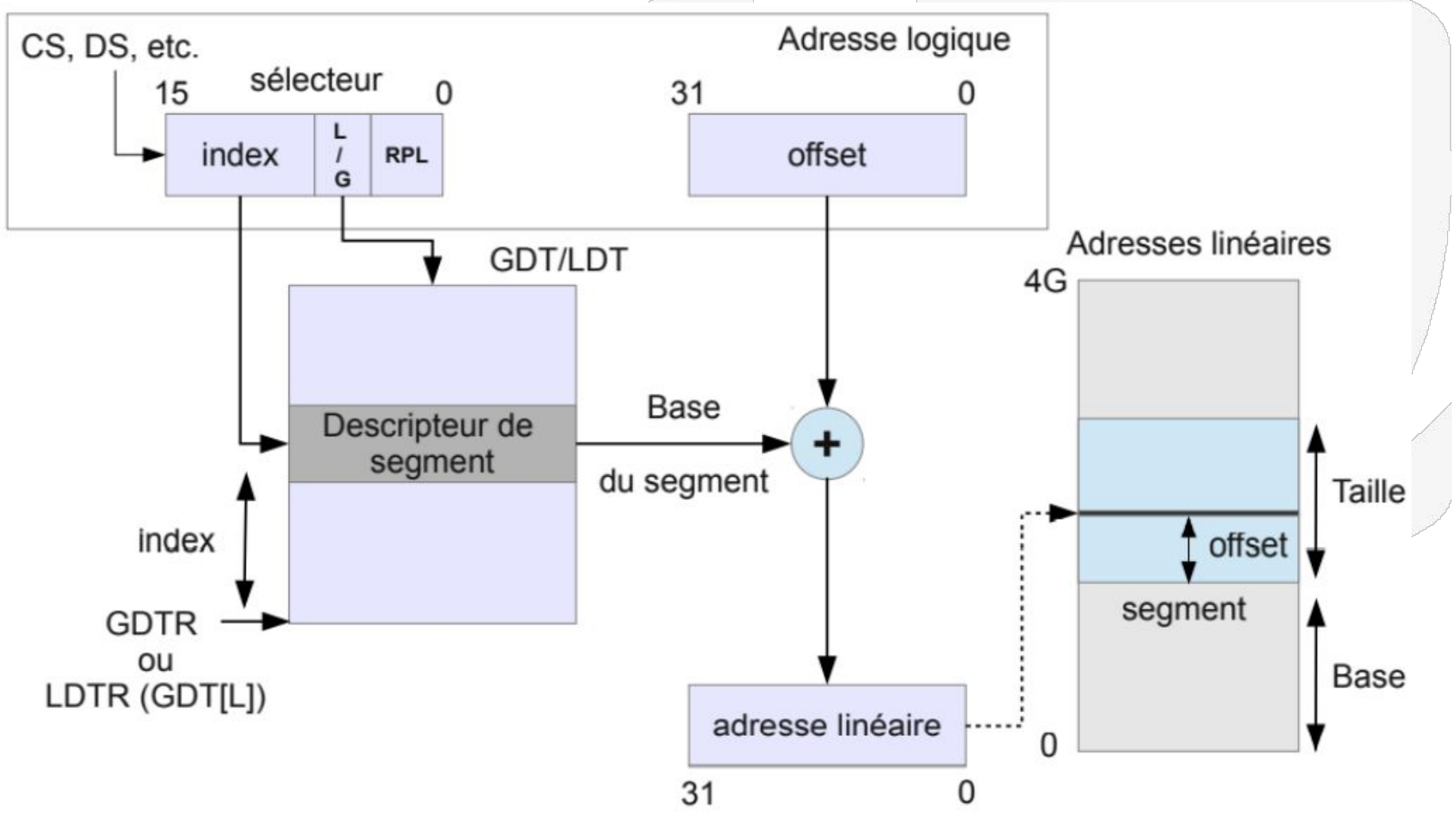
- diviser l'espace des adresses linéaires en segments de taille ajustable
  - traductibles en adresses physiques par le mécanisme de pagination

- Propriétés :
  - Un type (code, données ou segment spécial) ;
  - Un domaine de visibilité (base, taille) ;
  - Un niveau de privilèges demandé (DPL) ;
    - DPL = CPL maximal
  - Des restrictions spécifiques (code : lecture ; données : écriture).

La segmentation

- Liste des segments répartie sur 2 tables :
  - GDT = table globale, partagée entre toutes les tâches
  - LDT = table locale, spécifique à une tâche
- Les segments sont référencés par des sélecteurs de segments sur 16 bits
  - index dans l'une de ces deux tables
- seul le ring 0 est en mesure de définir la GDT et la LDT
- Vision simple : la segmentation sépare l'espace mémoire utilisateur de l'espace mémoire privilégié.
  - Le MMU se charge de traduire une adresse logique en une adresse linéaire
  - contrôles de privilèges et de mode d'accès.

La segmentation



Changement de CPL

- Changement de niveau de privilèges :
  - traitement d'une interruption matérielle ou d'une exception ;
  - appel système ;
  - retour du traitement d'une interruption, d'une exception ou d'un appel système (restauration du CPL de la tâche interrompue).

- Passage de l’espace mémoire Utilisateur (ring 3) vers/depuis Kernel (ring 0)

Problèmes de la segmentation

- allocation contigüe de la mémoire physique pour les tâches lancées
  - partitionnement de l'espace disponible
- problème du dimensionnement des zones allouées à chaque tâche
- réallocation de l'espace disponible
- recyclage des fragments libres isolés (défragmentation)
  - *garbage collector*
  - *déplacer les entités présentes en mémoire physique*
  - *ralentissement du processus concerné*

La pagination

- Réponse aux soucis de la segmentation
- Gestion de zones mémoire partagées entre tâches
- Principe :
  - découper l'espace des adresses linéaires en pages (blocs contigus de petite taille, typiquement 4ko) et la mémoire physique en blocs de même taille.
  - correspondance entre pages et blocs par l'intermédiaire d'une table de pages
  - Table des pages :
    - Modifiable
    - allouer et de désallouer des pages sans souci de contiguïté en mémoire physique
    - utilisation optimale des ressources disponibles

La table des pages

- La table des pages contient :
  - La validité de la page
  - Type de page (accès réservé au noyau ou autorisé en couche utilisateur)
  - permissions sur la page
  - l'adresse du bloc physique correspondant à la page
  - directives spécifiques pour la gestion du cache.

- Accès à une page non valide par un processus :
  - Exception CPU pouvant entrainer :
    - chargement de la page manquante (depuis disque par exemple)
    - duplication d'un bloc mémoire
    - extension de la pile de la tâche appelante
    - signal SIGSEGV (*Segmentation Fault*)

La pagination

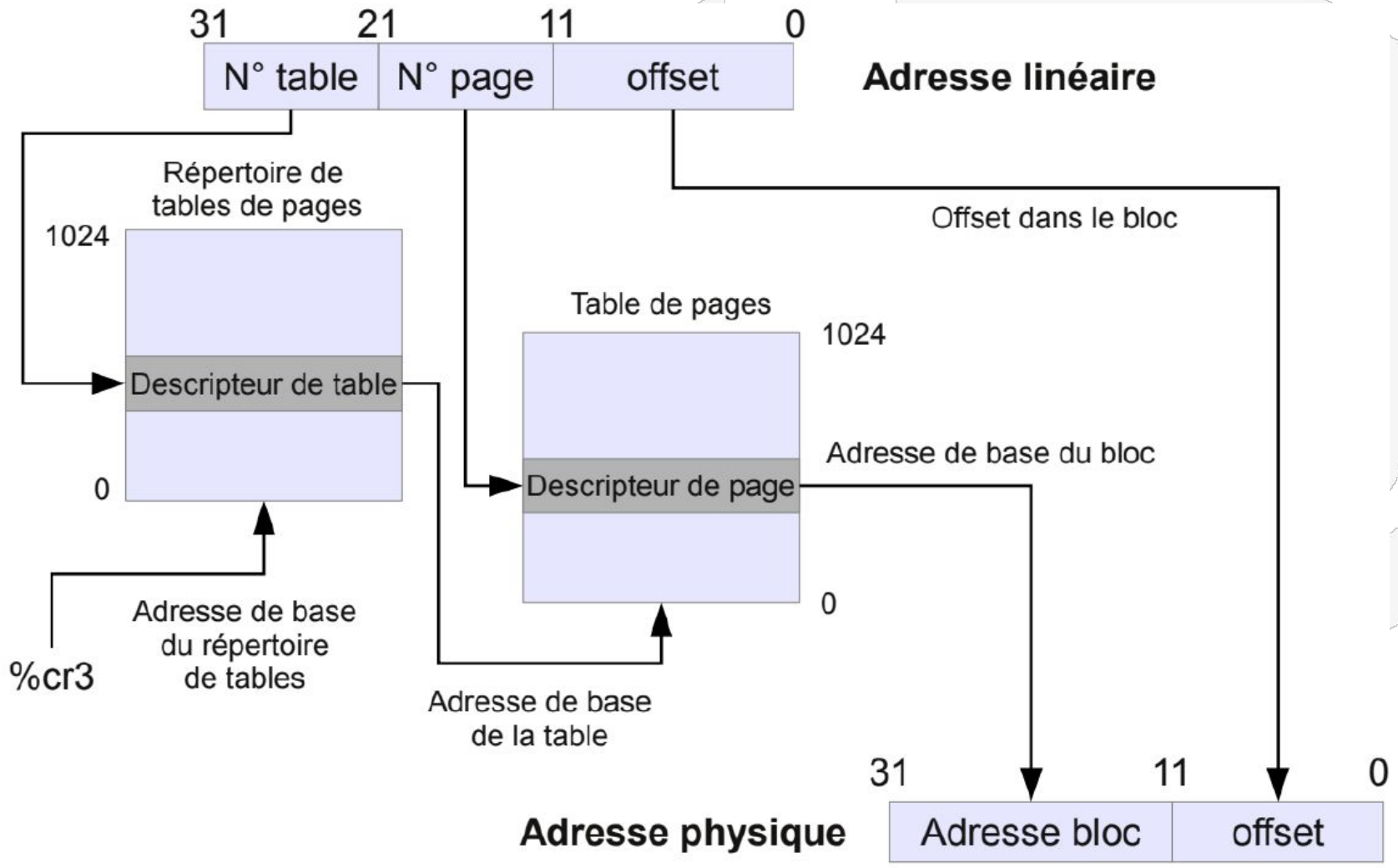
- traduction d'une adresse linéaire en adresse physique :
- octets de poids fort de l'adresse linéaire = numéro de page
- octets de poids faible = offset dans la page

- MMU utilise la table de pages pour traduire le numéro de page en numéro de bloc

- Adresse physique = le numéro de bloc (poids fort) et l’offset (poids faible)

- Attention : plusieurs tables de page peuvent être imbriquées !

La pagination



La pagination

- Permet d’ isoler les processus si chacun dispose de ses propres tables de pages
  - Stockées à des adresses différentes en mémoire.

- positif du point de vue de la sécurité
- limite les risques de propagation des incidents d'exploitation d'une tâche vers les autres tâches

La mémoire des processus

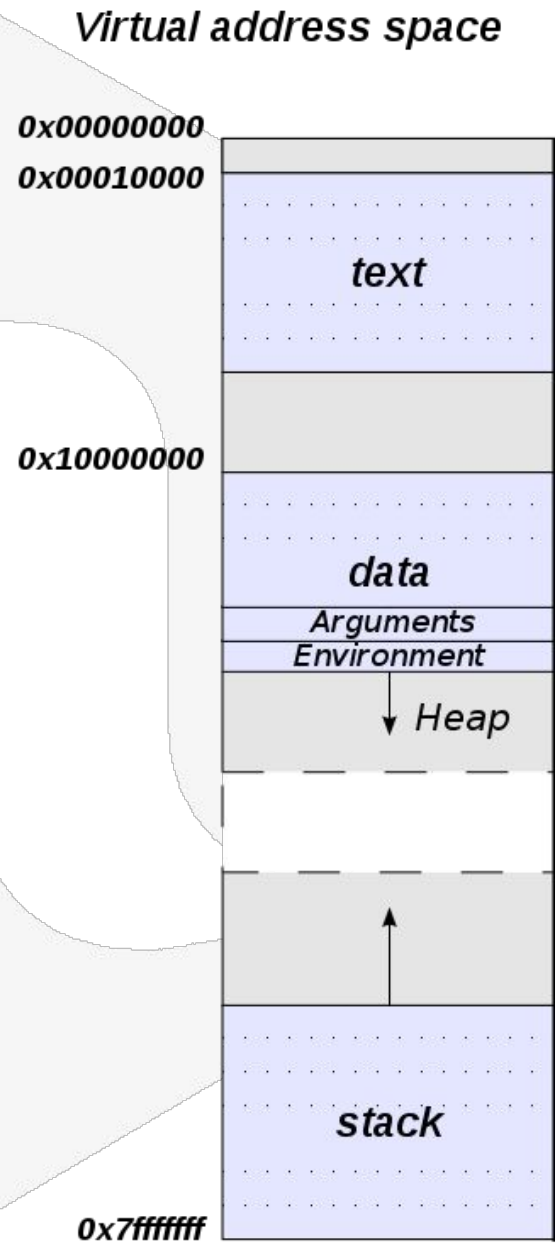
- mémoire associée à un processus en couche utilisateur en trois zones :
  - code (ou texte)
  - données, issues de l'exécutable et/ou gérées dynamiquement
  - pile, gérée dynamiquement.

- Pour optimiser, Le noyau ne charge pas immédiatement en mémoire physique le contenu d'un fichier
  - conserve la liste des correspondances entre zones mémoire et fichiers.

- L'espace mémoire d'un processus est composé
  - ensemble de zones appelées projections (*mappings*) composées
    - de pages mémoires associées à un fichier (code, librairies, etc…)
    - de pages mémoires associées à des données (tas, pile, etc...)

La mémoire des processus

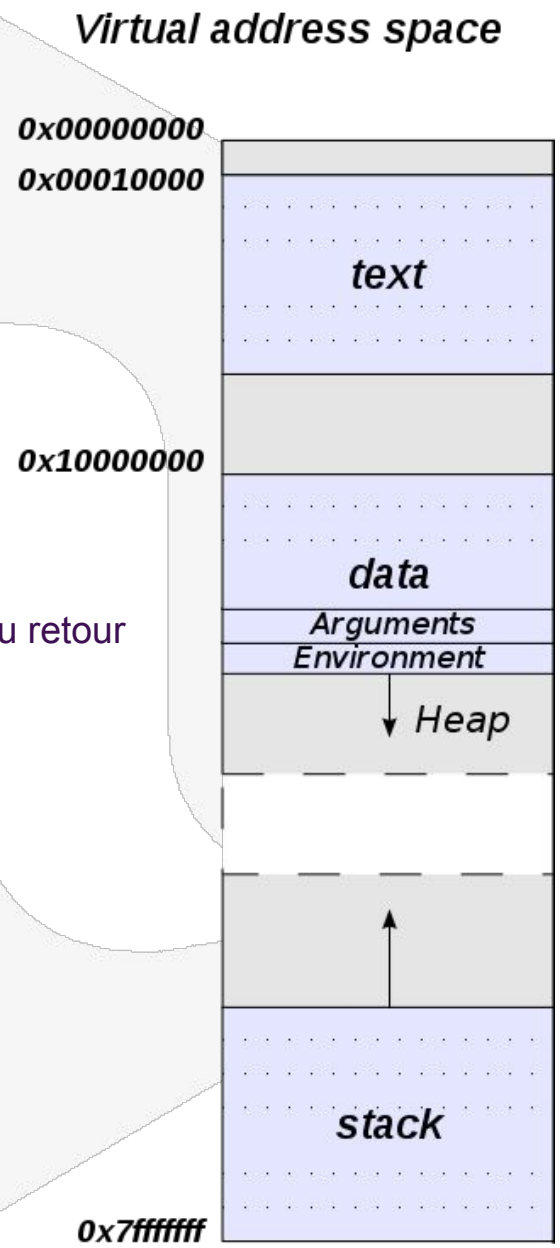
- zone de code ou *text*
  - code machine des différentes fonctions exécutables
  - chaînes de caractères et variables globales
- zone de données initiales ou *data*
  - données initialisées
  - données non-initialisées
- Zone des données de vie du processus
  - données dynamiques (variables)
  - Tas
  - pile



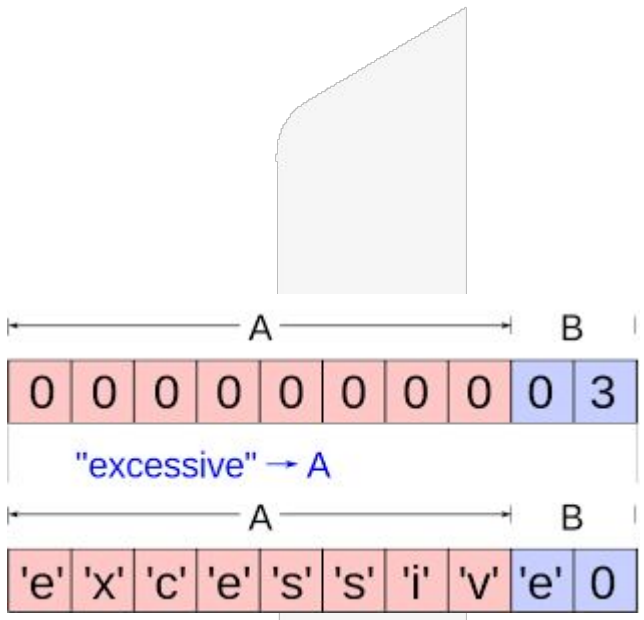
La mémoire des processus

- Focus : la pile ou *stack*
  - Taille variable, croît vers les adresses basses
  - Structurée par l'exécution des appels de fonction
    - Chaque fonction, un étage de pile
      - éléments nécessaires à la suppression de l'étage courant et au retour
      - variables locales non statiques de la fonction
      - arguments d'appel à une sous-fonction

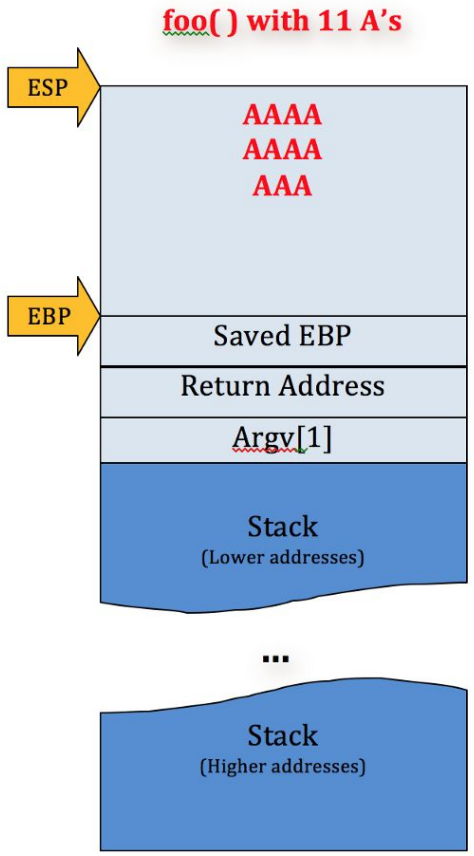
- mauvaise gestion/protection :
  - dépassement de taille ou de position (*buffer overflow*)



Buffer overflow

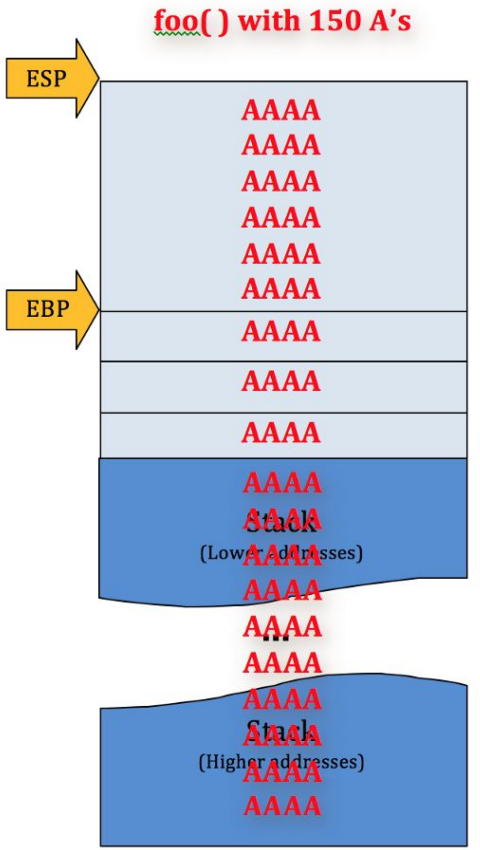


Buffer overflow



[6] CALL strcopy()

Copies value of Argv[1] into space reserved for local variable c



~~Saved EBP~~

~~Return Address~~

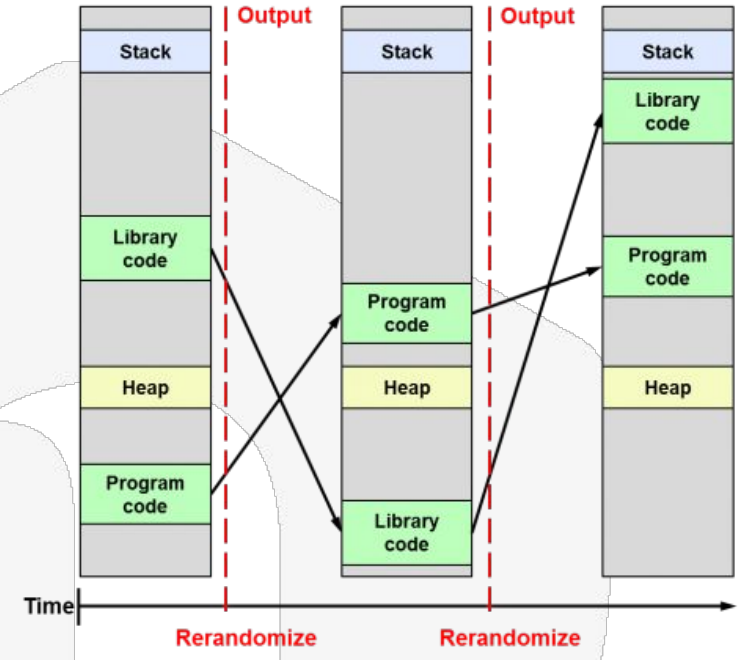
~~Argv[1]~~

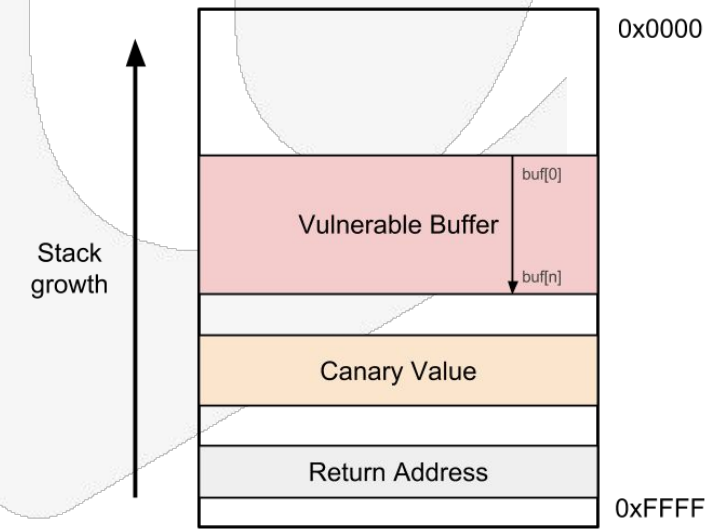
[6] CALL strcopy()

Copies value of Argv[1] into space reserved for local variable c

Buffer overflow : protections

- Randomisation des adresses en mémoire
  - *Address Space Layout Randomization* (ASLR)
- Canaries
  - *Stack-Smashing Protector*
  - Zone de contrôle de l'intégrité de la mémoire
- Pile mémoire non exécutable





Buffer overflow

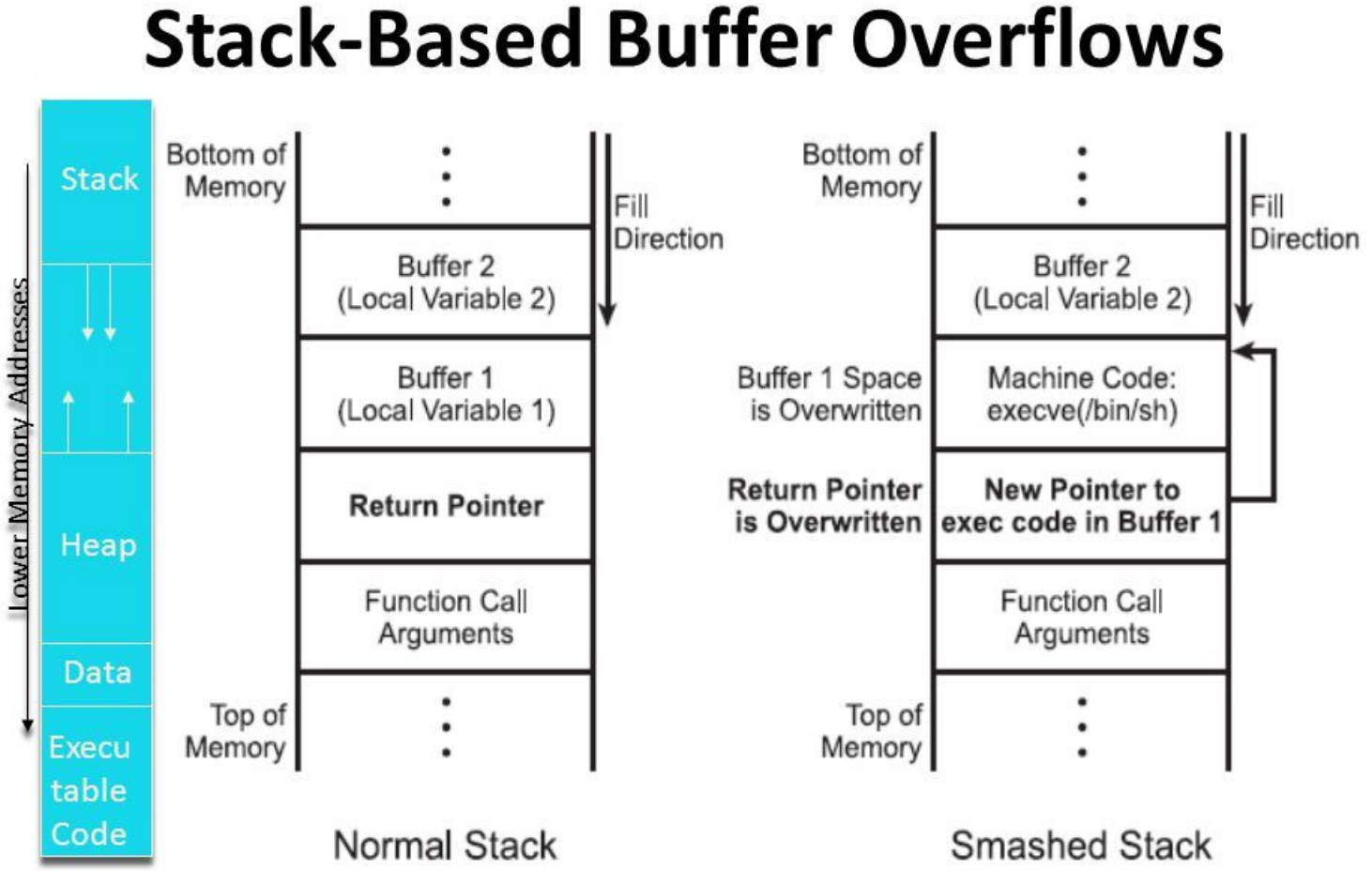


EXHIBIT 10.2 A normal stack and a stack with a buffer overflow.