

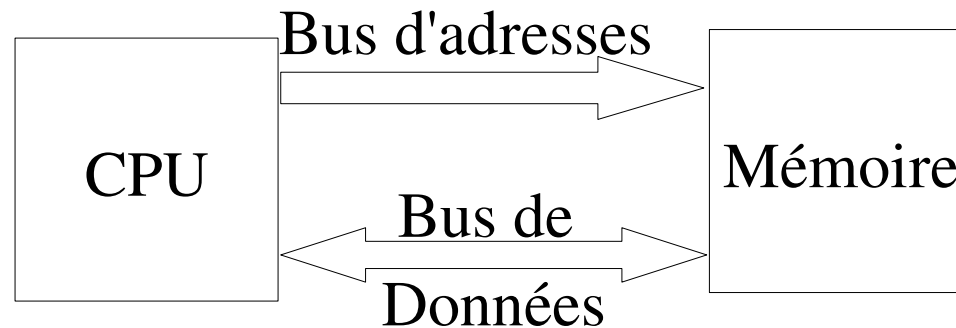
Rappels

Architecture des Processeurs

Plan

- Architecture Von Neumann
- CISC versus RISC
- Modes d'exécutions d'un processeur
- Exceptions & Interruptions
- Gestion mémoire
- Caches
- Pile d'exécution
- Entrées / Sorties

Architecture Von Neumann (1)



- Mémoire contient instructions et données
- CPU lit les instructions depuis la mémoire
- Chargement d'un programme en mémoire
 - Instructions manipulées comme des données
- Programmes peuvent s'auto-modifier

Architecture Von Neumann (2)

- PC: adresse prochaine instruction à exécuter
- SP: adresse du sommet de pile
- Registres généraux de calcul
- Registre d'état
 - Bits de conditions (Z, LT, GT, etc.)
 - Modes d'exécution
 - Niveau de protection (user/supervisor)
 - Adressage physique/virtuel
 - Masque des interruptions

CISC versus RISC

- RISC (Reduced Instruction Set Computer)
 - Choisit les instructions et les modes d'adressage les plus utiles
 - Peuvent être réalisées en hardware directement
- CISC (Complex Instruction Set Computer)
 - Choisit les instructions et les modes d'adressage qui rendent l'écriture de compilateurs plus simple
 - Nécessite l'utilisation de micro-code

CISC versus RISC

- Avantages CISC
 - Réduit le nombre d'instructions d'un programme
- Inconvénients CISC (ou supposés tels)
 - Temps d'exécution (des instructions) plus long
 - Processeur plus complexe et donc plus lent

CISC versus RISC

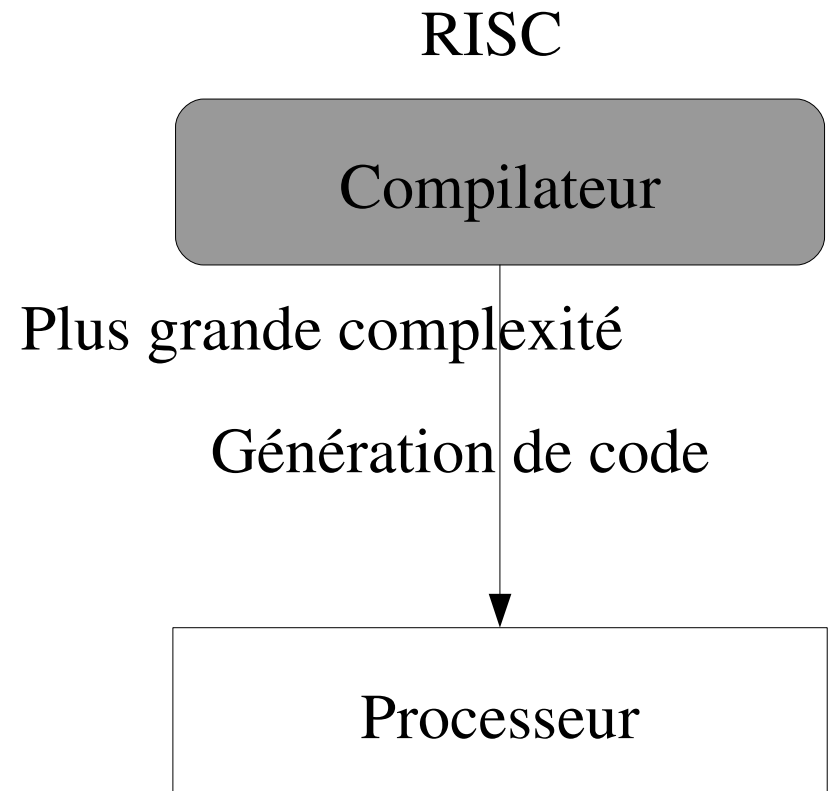
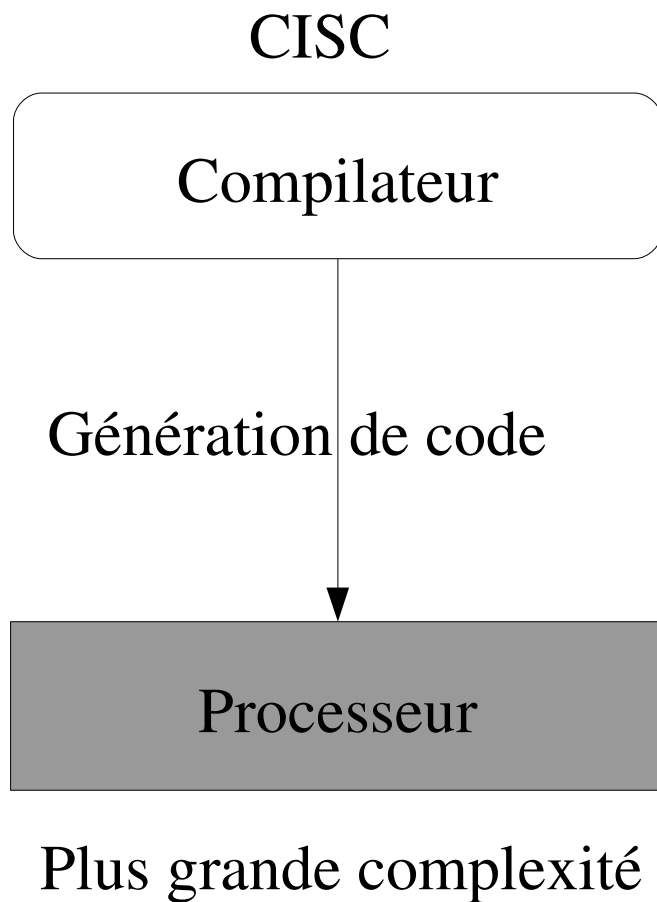
- RISC:

- Place disponible sur la puce pour de nombreux registres généraux et des caches mémoire
- écriture de compilateur et programmation assembleur difficiles

- CISC:

- Peu de place sur la puce: peu de registres et caches de petite taille
- écriture de compilateur et programmation assembleur faciles

CISC versus RISC



RISC Philosophie

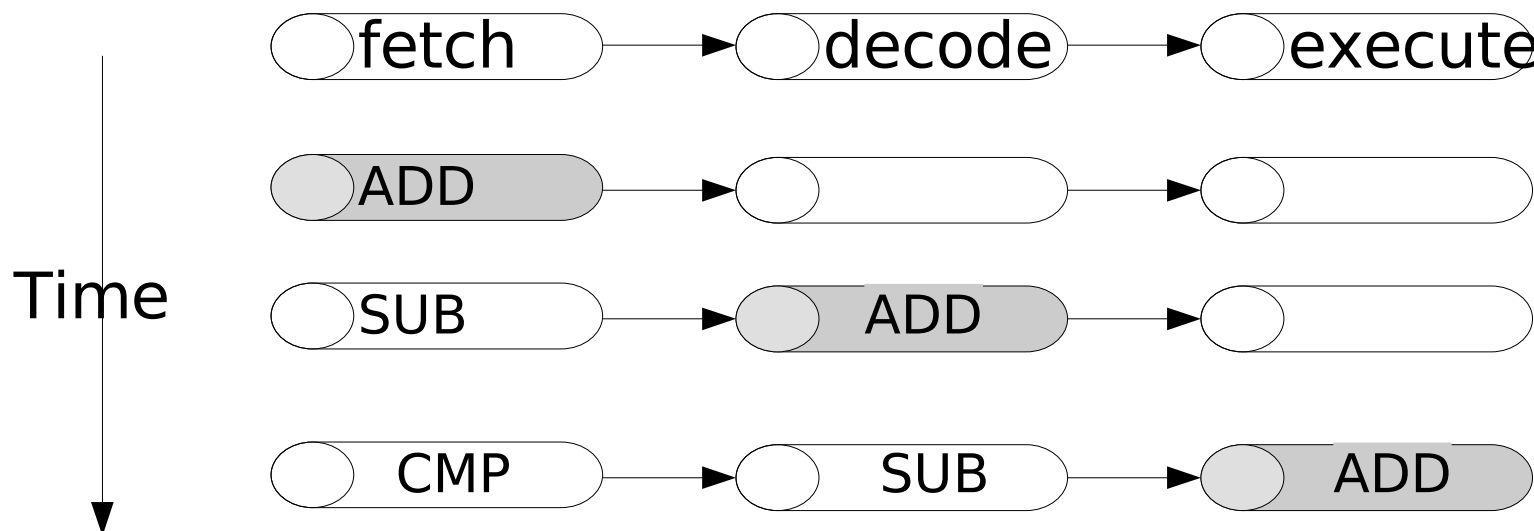
- Nombre d'instructions réduit
 - Une instruction par cycle
 - Instructions de taille fixe => facile à pré-charger
- Pipelines
 - Traitement d'une instruction est décomposé en unités élémentaires
 - Exécution de ces unités en parallèle

RISC Philosophie

- Registres
 - Grand nombre peu spécialisés (adresses ou données)
- "Load-Store"
 - processeur opère seulement sur les données des registres
 - Instructions spécifiques pour lire un mot mémoire dans un registre, et pour écrire un registre en mémoire.

RISC et Pipeline

- Exemple du ARM7 : 3 niveaux
 - Fetch: charge une instruction depuis la mémoire
 - Decode: identifie l'instruction à exécuter
 - Execute: exécute l'instruction et écrit le résultat dans le registre



Pipeline

- Instructions de branchement
 - => vidage (*flush*) du pipeline
 - prédiction de branchement pour les réduire
- Interruption
 - N'interrompt pas l'instruction se trouvant au niveau "execute"
 - Les autres instructions dans le pipeline sont abandonnées.

Mode d'exécution « superviseur »

- Accès aux ressources protégées
 - Registres d'I/O, de gestion de la mémoire
 - Registre contenant le mode d'exécution
 - Toute la mémoire physique, tous les périphériques
- Droit d'exécuter les instructions « privilégiées »
 - Masquage des interruptions
 - Changement de mode d'exécution
 - Instructions d'I/O
- Processeur démarre en mode superviseur

Mode d'exécution

« utilisateur »

- Droits restreints
 - Accès aux registres généraux
 - Exécution des instructions standard
- Passage superviseur -> utilisateur
 - Contrôlé par OS
 - Lancement d'un programme
- Passage user -> superviseur
 - Instruction « svc » (supervisor call)
 - Opération invalide ou non autorisée => exceptions
 - Interruption

Supervisor Call

- Utilisé pour invoquer un service du système
 - service id. (open, write, ...) dans un registre
 - service args (pathname, fd, ...) dans registre(s) ou dans la pile utilisateur
 - résultat / code erreur dans registre(s)
- Synchrone avec (processus / thread) appellant
 - relatif au contexte système appellant
- Appels système fournis par fonctions de bibliothèque du langage (libc)

Exceptions

- Provoquée par une condition exceptionnelle lors de l'exécution de l'instruction courante
 - Adresse mémoire invalide
 - Instruction non autorisée ou invalide
 - Division par zero
- Synchrones avec le programme exécuté
 - Détectée par le CPU
 - Traitée dans le contexte du programme courant

Interruptions

- Événement déclenché par composant matériel
 - Fin d'entrée/sortie d'un périphérique
 - Echéance de temps terminée
- Signal envoyé au processeur de l'extérieur
 - Par l'intermédiaire d'un PIC
 - Partageable ou non
- Événement asynchrone
 - Indépendant du programme courant
 - Masquable par le système

Traitement Exceptions/Interruptions (1)

- Processeur suspend l'exécution en cours
- Sauvegarde état courant
 - PC, SP, registre d'état
 - Dans des registres dédiés ou dans la pile superviseur
- Ou utilise des « shadow registers »
 - 2 jeux de registres, un par mode d'exécution

Traitement Exceptions/Interruptions (2)

- Charge nouveau contexte d'exécution :
 - Registre d'état avec mode « superviseur »
 - PC avec adresse mémoire spécifique
- Opérations de sauvegarde et de chargement
 - Non-interruptible
 - Sinon état non cohérent
 - Sauf par NMI, sur PowerPC par exemple
 - Si exception durant opérations (« double-faute »)
 - => arrêt CPU ou traitement spécial (Intel)

Exceptions/Interruptions

"Vector Table"

- Adresse de branchement
 - Dépend de l'exception/interruption
- Vers une table de vecteurs localisée
 - à adresse prédéfinie (0x00000000 par exemple)
 - dans un registre (IDTR sur Intel)
- Contient une instruction de branchement
- Table de vecteurs initialisée par le système

Vector Table Intel

- 256 entries on 32 bits
- [0 - 31] exceptions reserved by manufacturer
 - Divide by zero
 - Single step
 - Etc...
- [32 – 255] system defined
 - Traps (appels systèmes)
 - External interrupts

Memory Management

- MPU (Memory Protection Unit)
 - Restreindre l'accès du programme courant aux seules zones de la mémoire physique qui lui ont été attribuées
- MMU (Memory Management Unit)
 - Etendre l'espace mémoire accessible au delà de la mémoire physique disponible

Memory Protection Unit

- Restreindre l'accès aux zones de l'espace mémoire physique attribuées au programme courant
- Décomposition de la mémoire en régions
- Pas de translation d'adresse (mémoire physique)
ou translation simple
- Gestion statique ou remplacement global

Memory Protection Unit

- Définit au niveau matériel des "régions"
 - Adresse de début et taille
 - Droits: pas d'accès, lecture, lecture/écriture, exécutable, cachable
- Droits comparés avec le mode du processeur lors d'un accès mémoire pour déterminer si l'accès est valide ou non
 - Accès invalide => génération d'une exception

Régions MPU

- Associées à différentes parties d'un programme
 - Instructions (exécutable, read-only)
 - Données (non exécutable, read/write)
- Granularité
 - 4 octets - minimiser mémoire (automobile, etc...)
 - Puissance de 2 (en général $\geq 4\text{KB}$)
- Adresse de début d'une région est un multiple de sa taille

MPU 8 régions

~~Données
Bibliothèque 1~~

~~Données
Programme 1~~

~~Code
Programme 1~~

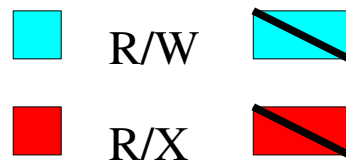
Données
Bibliothèque 2

Données
Programme 2

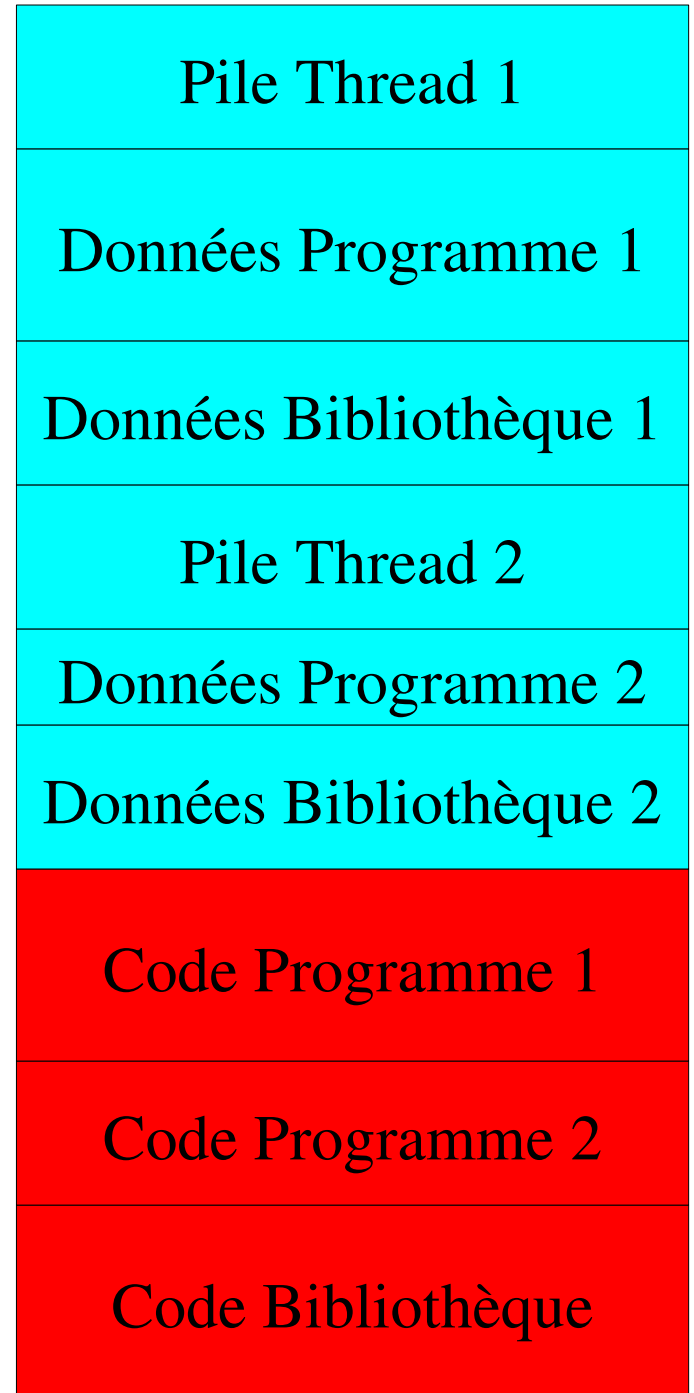
Code
Programme 2

Pile Thread
Courante

Code
Bibliothèque



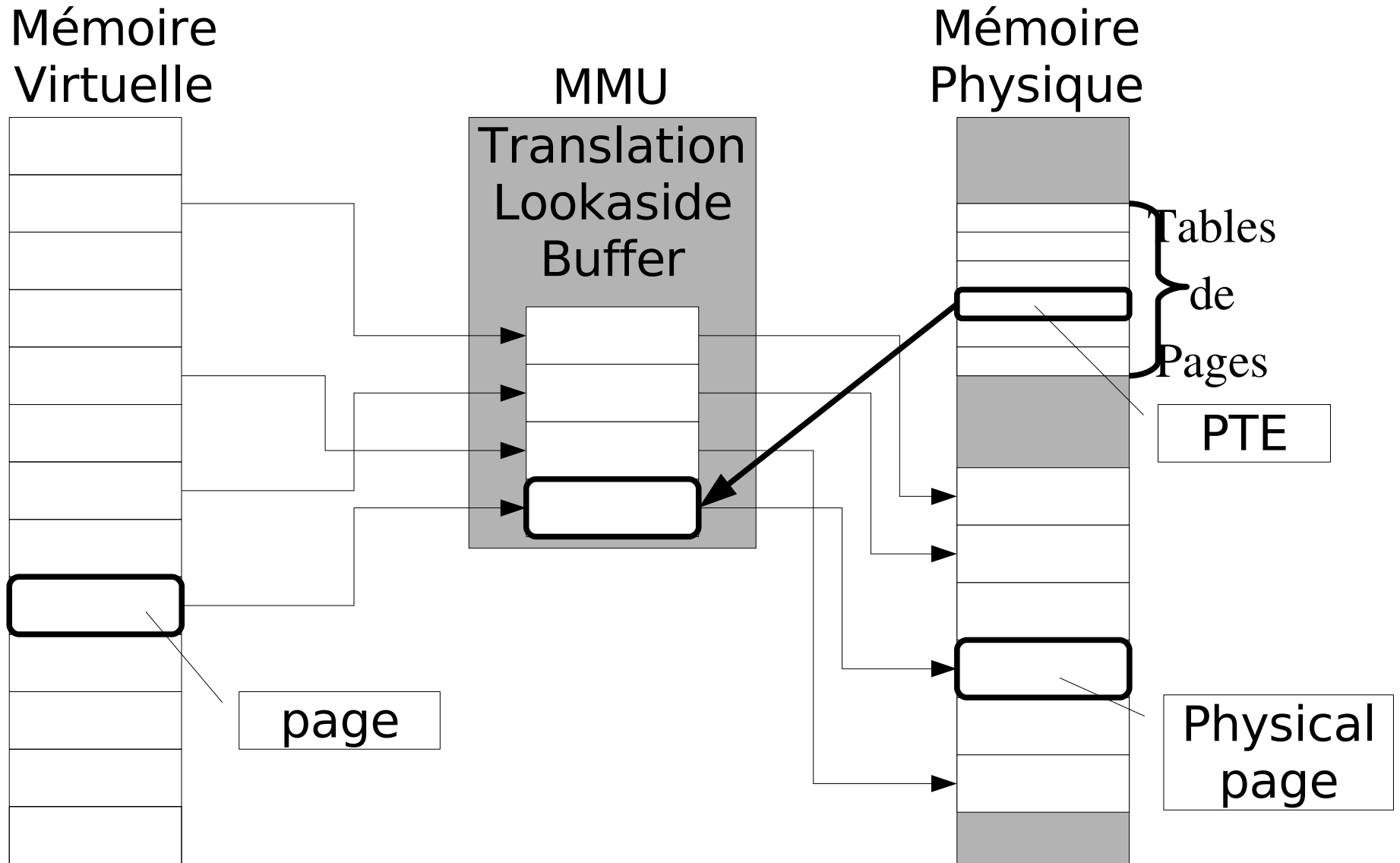
Région invalide



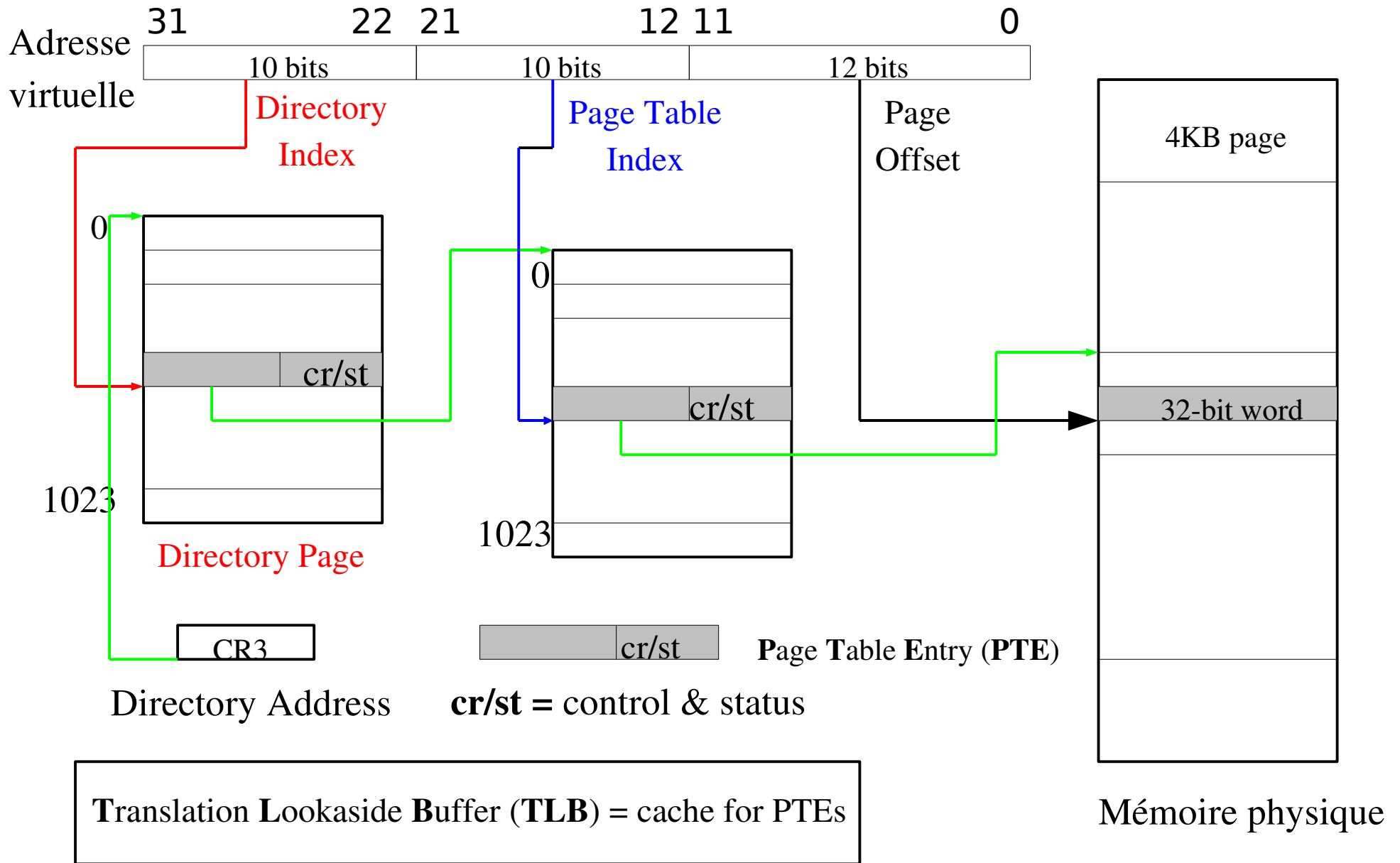
Memory Management Unit

- Notion d'espace d'adressage virtuel
 - Permet de « mapper » des fichiers dans l'espace d'adressage d'un processus
 - Contient souvent grandes zones non définies
- Décomposition de la mémoire en pages (en général de même taille 4KB ou 8KB)
- Gestion dynamique fine
- Conversion adresse virtuelle => adresse physique
 - utilisation structure "arborescente" multi niveaux
 - Intel : 2 niveaux (1 Directory + Page Tables)

Memory Management Unit (MMU)



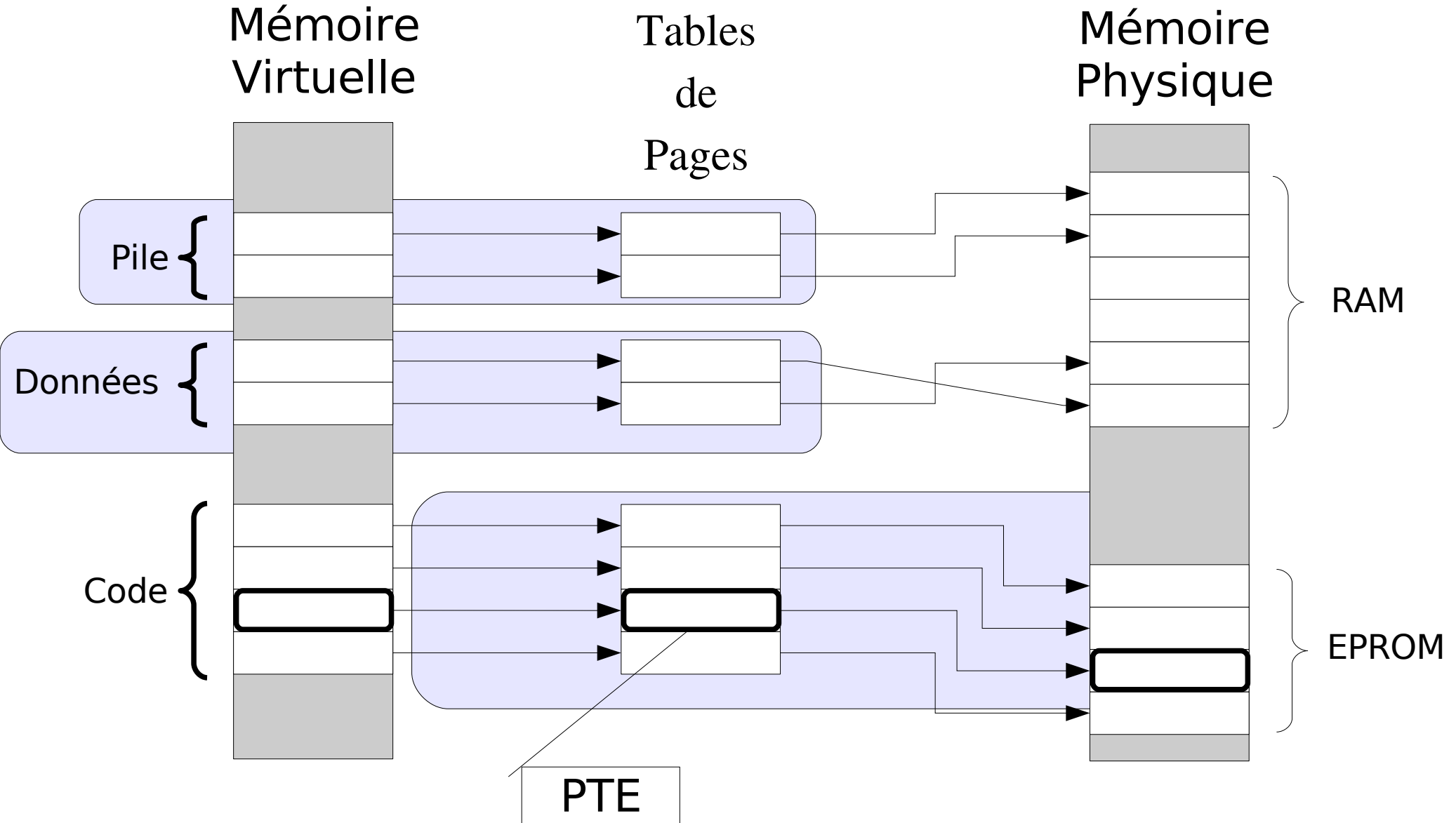
MMU Intel



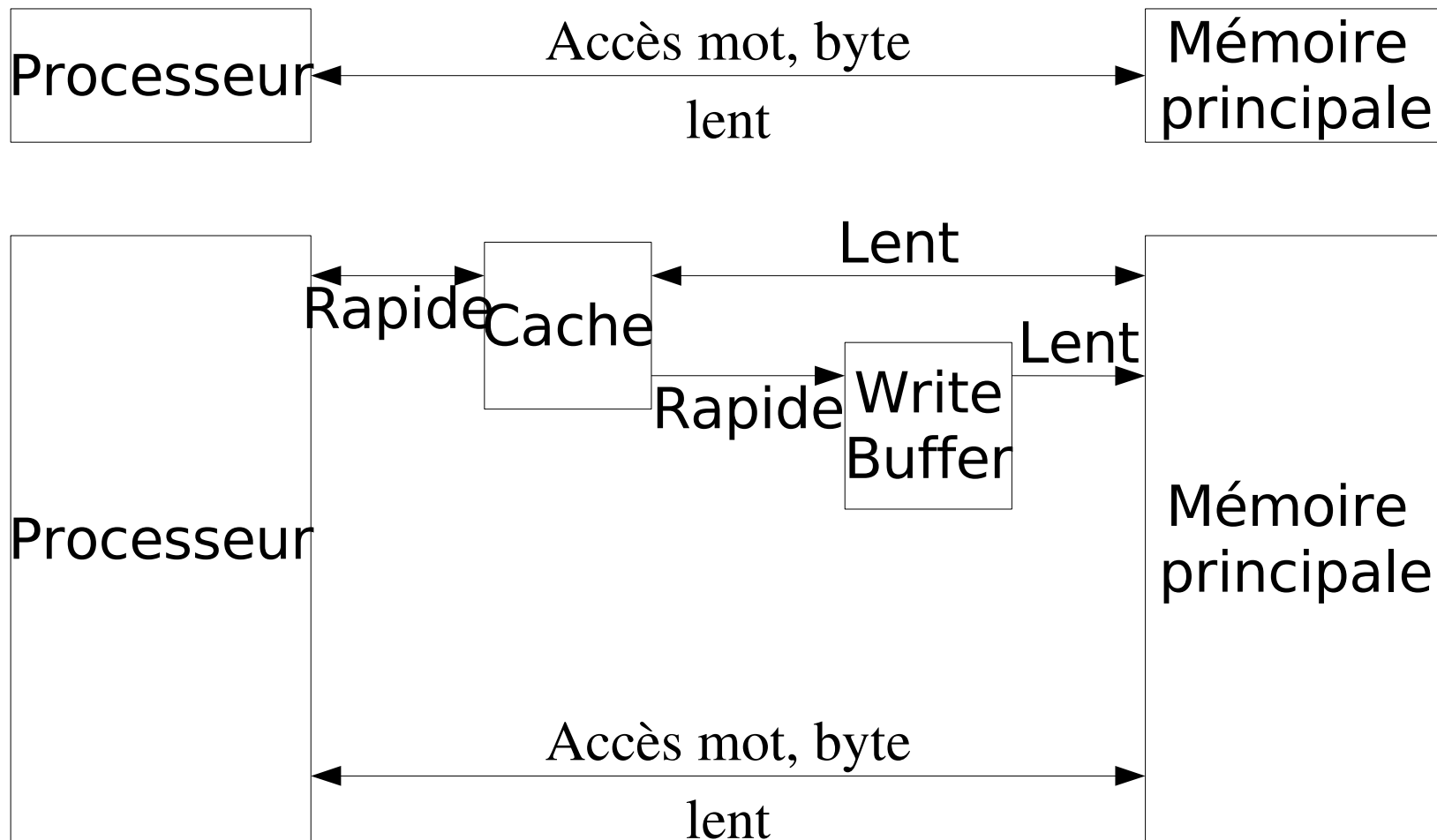
Page Table Entry

- Décrit une page d'un espace virtuel
- Adresse de la page physique associée, si valide
- Bits de contrôle
 - Page physique associée valide
 - Droits d'accès (lecture, écriture, exécution)
- Bits de status
 - Page accédée
 - Page modifiée

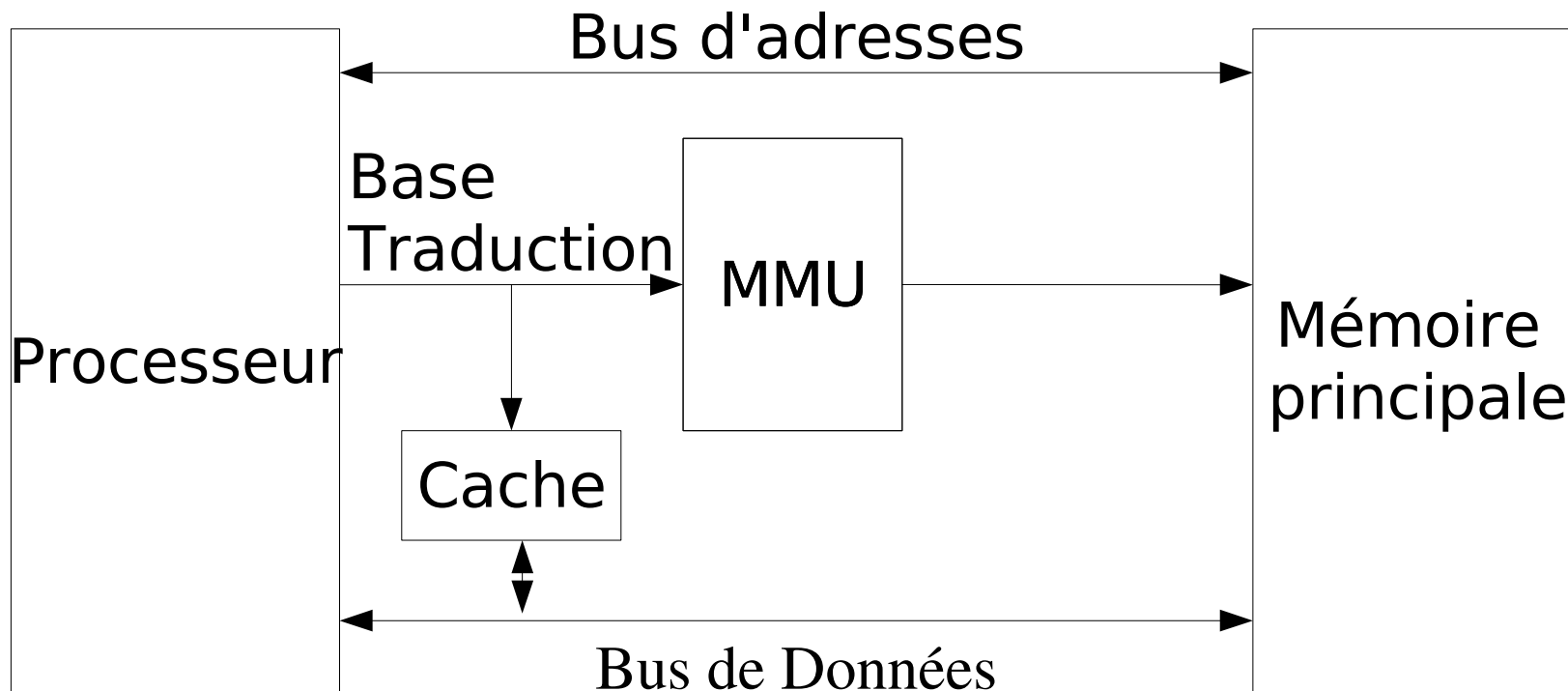
Exemple de "Mapping"



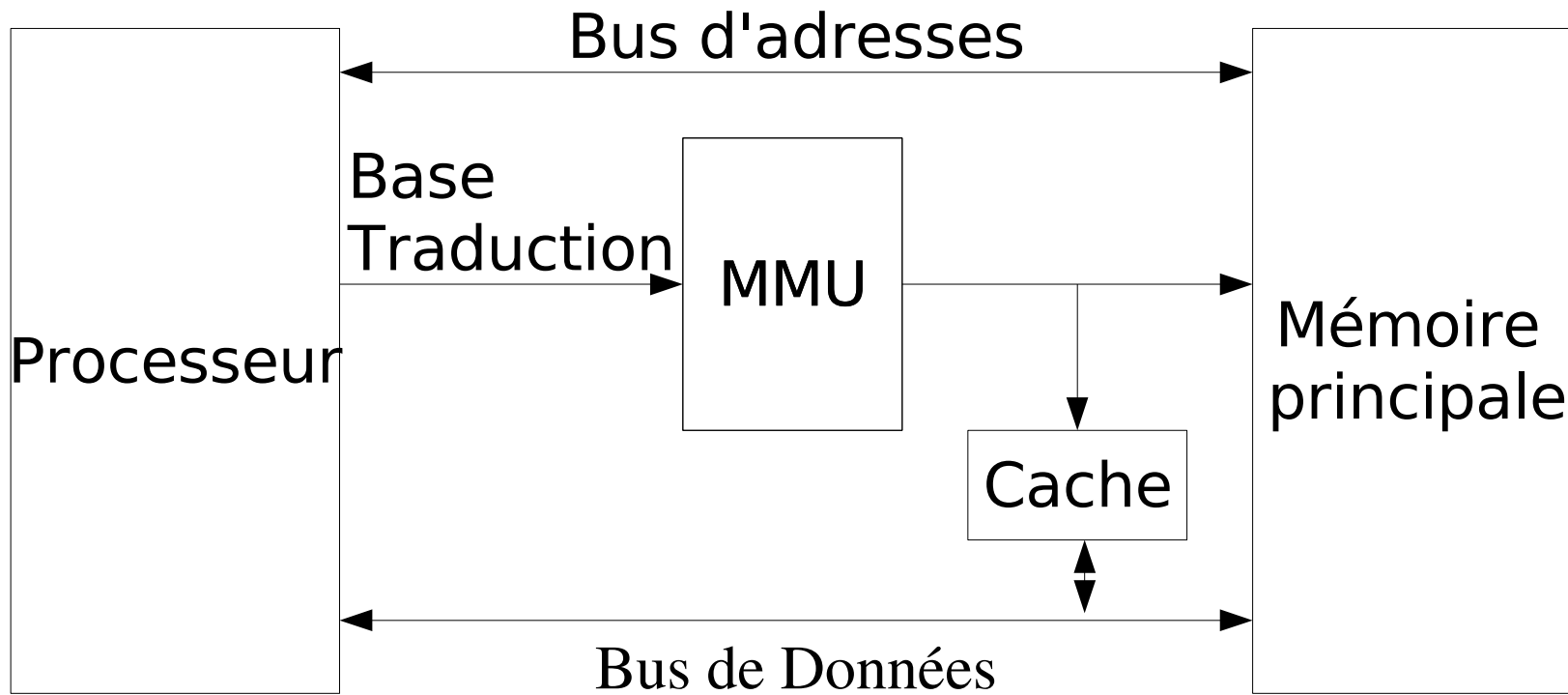
Caches Mémoire



Cache Logique



Cache Physique



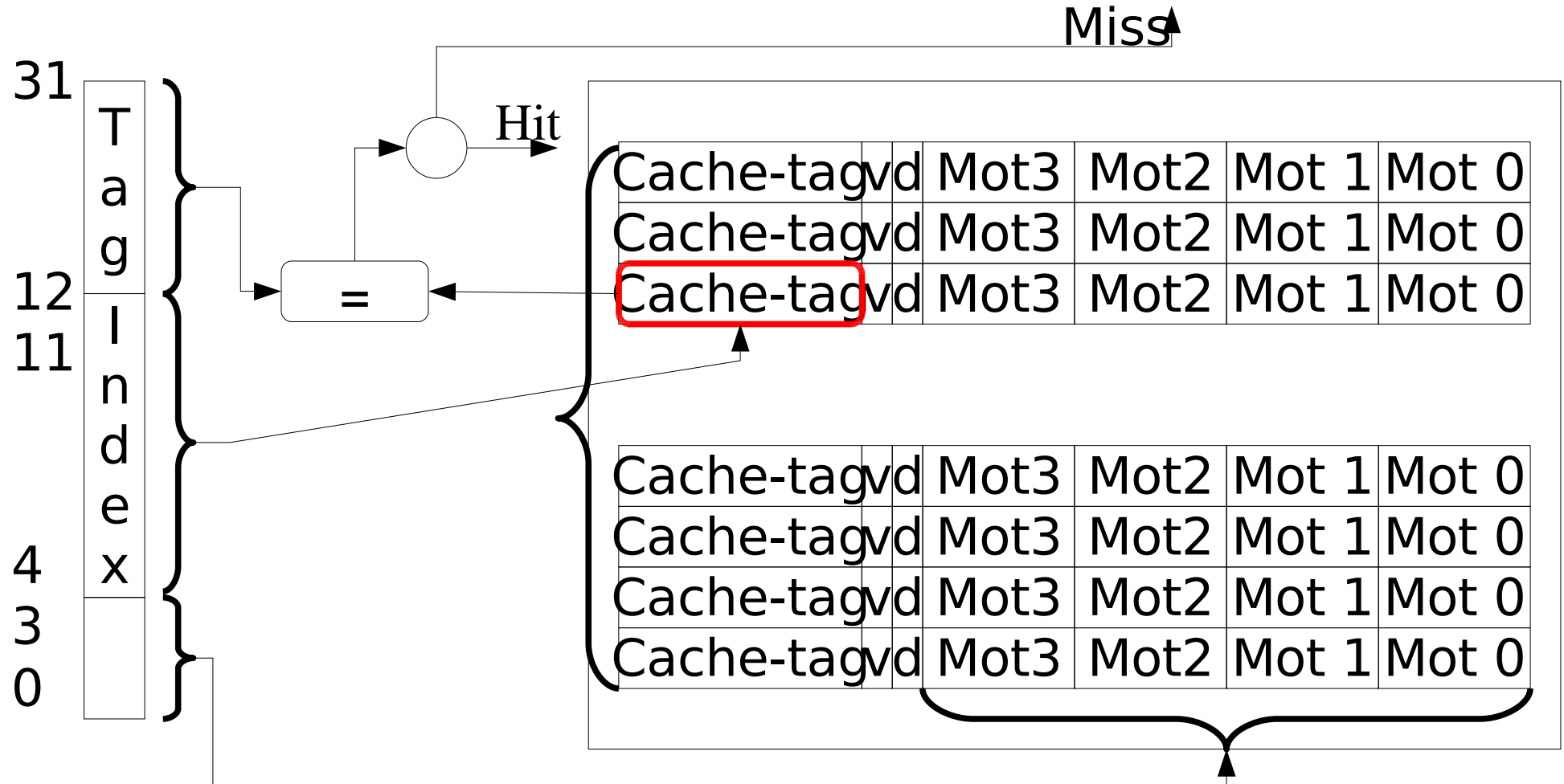
Composition d'un Cache

- Cache est décomposé en « lignes de cache »
- Ligne chargée avec des "blocs"
 - Zones contiguës de mots mémoire
 - Lignes identifiées à partir de l'adresse
- États
 - Ligne valide ou non
 - Ligne modifiée ou non

Caches - Politique de gestion

- Writethrough
 - Toute modification est immédiatement répercutée en mémoire centrale => lent
- Writeback (utilisation du dirty bit)
 - Écrit dans une ligne de cache valide
 - Ligne de cache contient des données plus récentes
 - Ligne écrite quand nécessité de remplacer la ligne
- Politiques de remplacement
 - Round-robin ou pseudo aléatoire

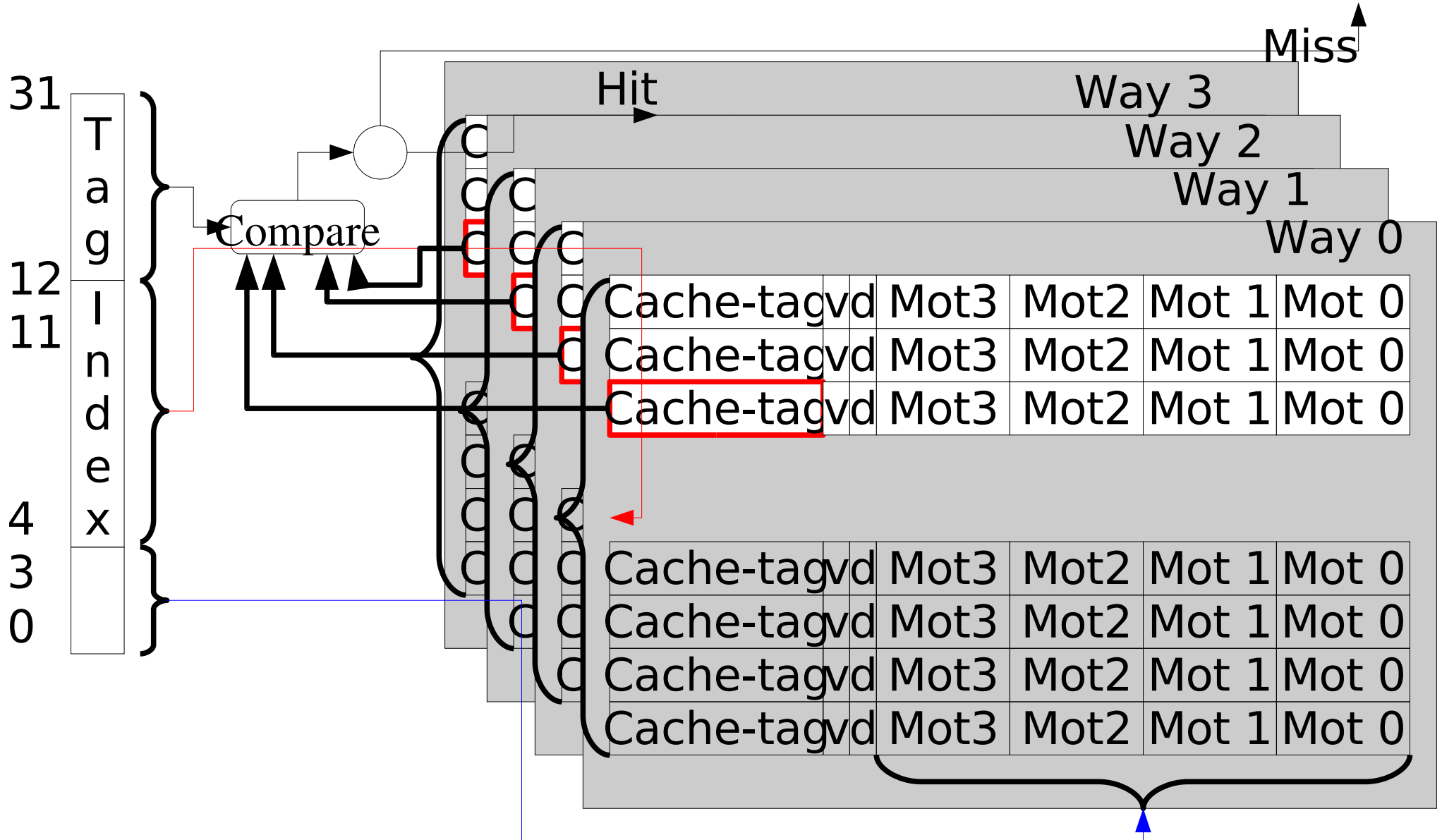
Architecture d'un cache



V = valide

D = dirty (modifié)

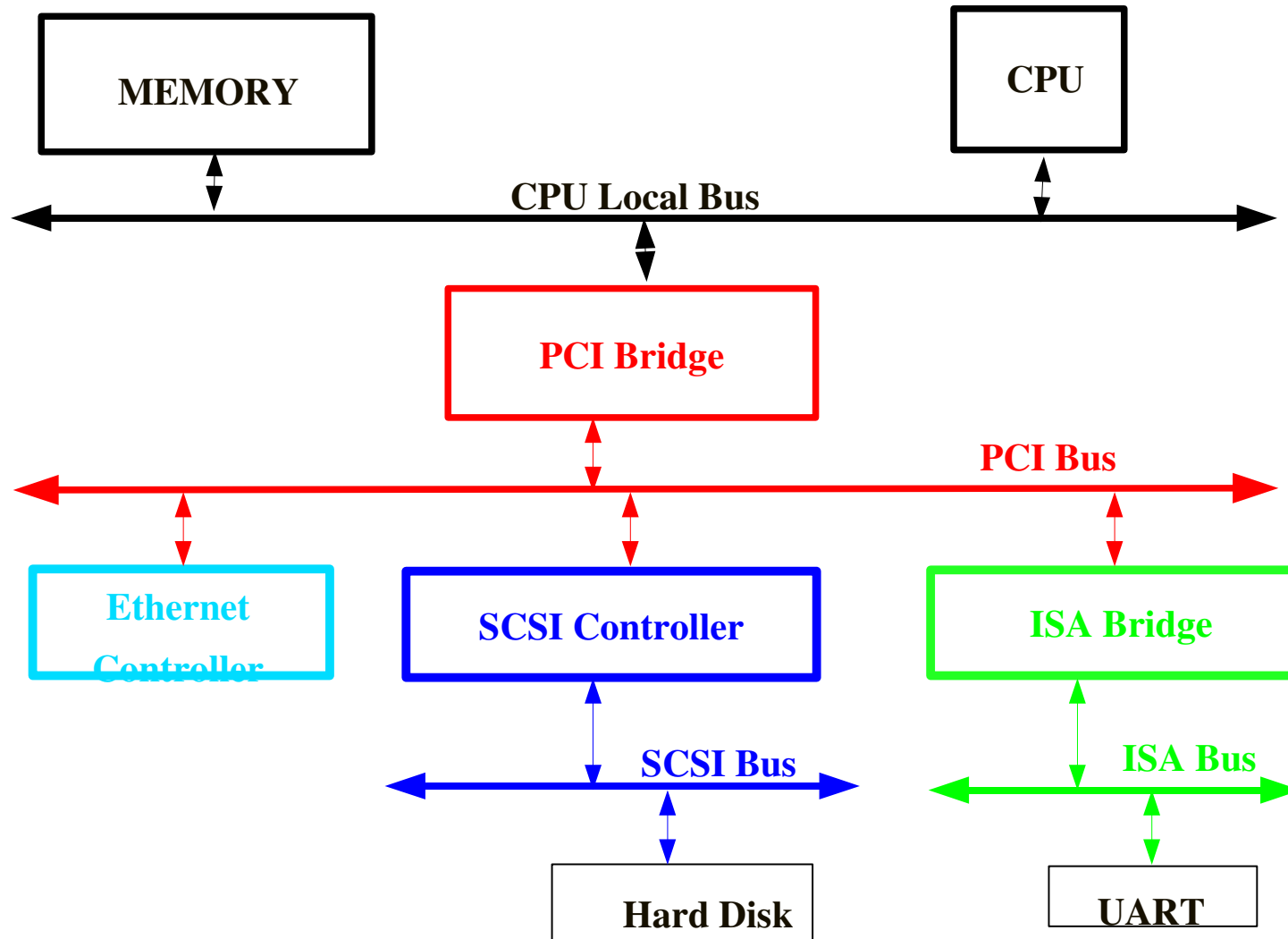
Caches Associatifs



Caches - contrôle

- Flush: vider tout le contenu
- Clean: écrire toutes les données modifiées en mémoire
- Possibilité d'effectuer ces opérations par portion (granularité: la ligne)
- Opérations pour I caches et D caches
- Possibilité de verrouiller du code et des données en cache

Physical I/O Architecture



Device Registers

- Périphériques gérés par registres
 - Contrôle (start/stop, interruptions, etc.)
 - Etat (interruptions, etc.)
- Registres accessibles
 - Instructions spéciales d'I/O (Intel)
 - Adresses mémoire spécifiques
 - Qui ne doivent pas aller dans le cache mémoire !
- Etat stoppé à power-on/après hardware reset
- Pb lors de re-démarrage « à chaud »

Transferts de Données

- Indirects à travers des registres
 - Simple mais pas efficace
 - Devices lents (clavier, souris, etc)
- Par accès direct en mémoire (**DMA**)
 - Données copiées par device dans buffers du driver
 - Efficace (copie en parallèle avec CPU)
 - Devices rapides (disques, ethernet, USB, etc.)
 - Complexe
 - Buffers pas dans cache / « bus snooping » (Intel)
 - Continue après arrêt brutal puis re-démarrage à chaud

Données globales / locales

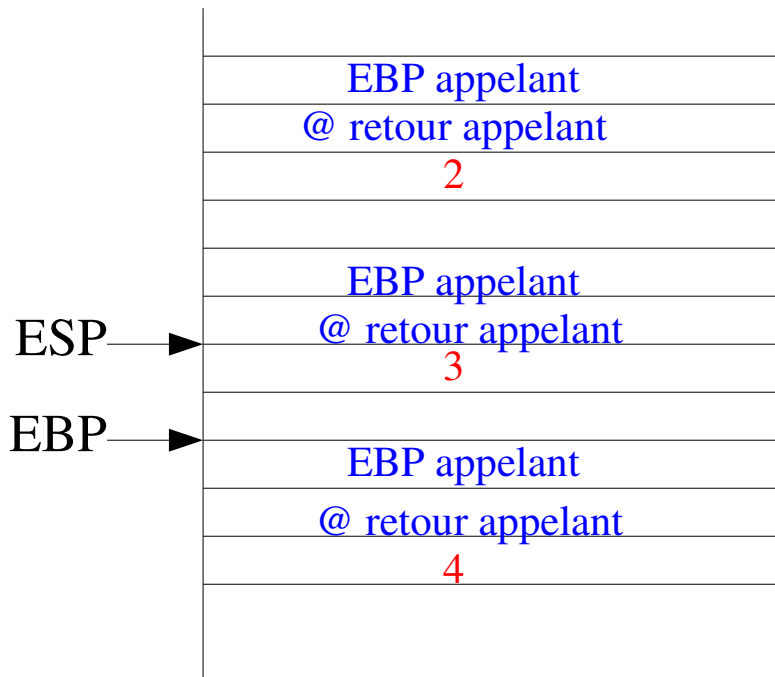
```
extern char c_array[];    /* global public to all
    files */
static int i_array[2048]; /* global private to file
    */

    int /* return value */
func (int i, char* name) { /* call arguments */
    int* ptr;              /* local to func() */
    struct example_t ex;   /* local to func() */
    static char c_pfgd[100]; /* global private to
    func() */
```

Pile d'exécution

- Fonctions langage C
 - Paramètres d'appel
 - Contexte de retour
 - Variables locales
- Support récursivité
 - `fact(n) {if n <= 2 return n; else return n * fact(n-1);}`
- Nombre de paramètres variable
 - `printf(...)`

Pile d'exécution – Intel



Stack Bottom

```
unsigned int fact(unsigned int n) {
    return (n <= 2) ? n : n * fact(n - 1);
}
```

fact:

```
pushl %ebp
movl %esp, %ebp
subl $8, %esp
cmpl $2, 8(%ebp)
jbe .L2
movl 8(%ebp), %eax
subl $1, %eax
movl %eax, (%esp)
call fact
movl %eax, %edx
```

```
imull 8(%ebp), %edx
movl %edx, -4(%ebp)
jmp .L4
```

.L2:

```
movl 8(%ebp), %eax
movl %eax, -4(%ebp)
```

.L4:

```
movl -4(%ebp), %eax
leave
ret
```