

Architecture ordinateur

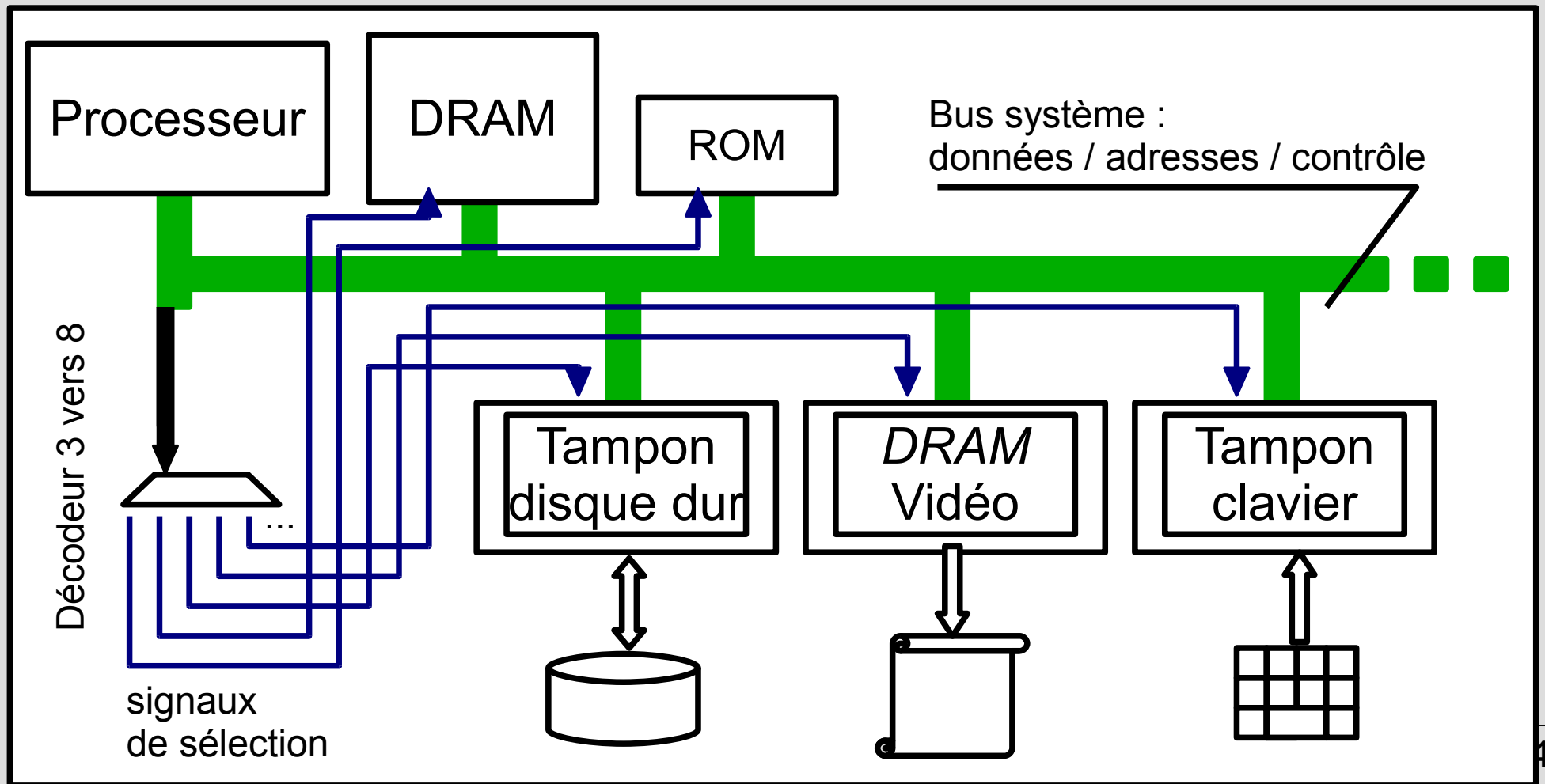
Organisation mémoire et Entrées/Sorties

Plages mémoire et E/S

- Dans une architecture à bus partagé, seule une partie des adresses active le contrôleur mémoire.
- Il reste des adresses vacantes qui permettent au processeur de désigner des périphériques et de gérer les **entrées/sorties**
- La mémoire peut correspondre à des blocs utilisant des technologies différentes :
 - ROM pour stocker le code de démarrage
 - DRAM pour stocker programmes et données chargées depuis une mémoire de masse

Plages mémoire et E/S

- Exemple de circuit auxiliaire pour sélectionner les composants par adresses



Plages mémoire et E/S

- Dans l'exemple précédent 3 bits de **poids fort** du bus d'adresse sélectionnent un module du système parmi 8 au maximum :

16 bits d'adresse

000xxxxxxxxxxxxx -> accès ROM (démarrage)

001xxxxxxxxxxxxx -> accès DRAM (mem. principale)

010xxxxxxxxxxxxx -> accès tampon disque dur

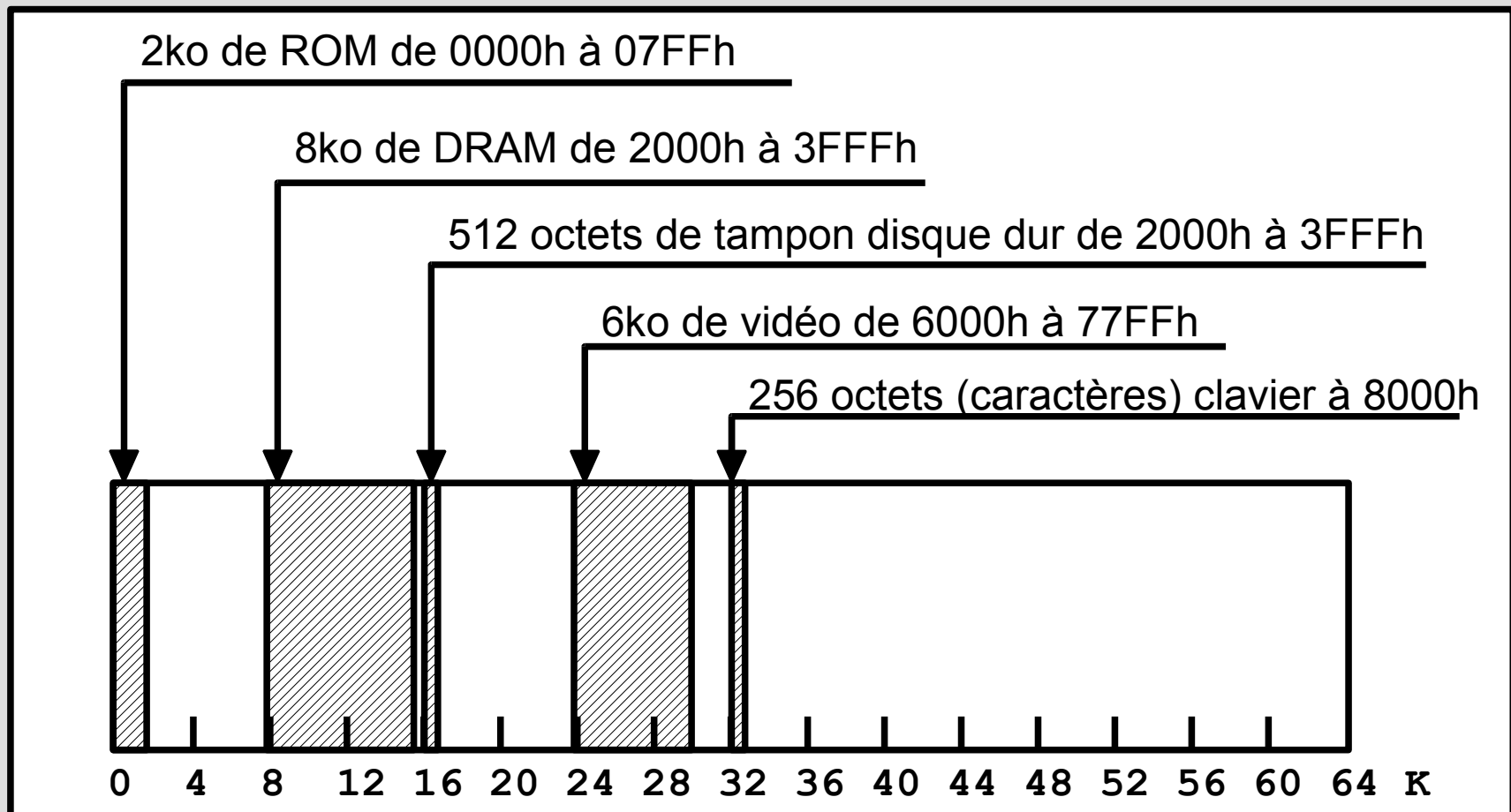
011xxxxxxxxxxxxx -> accès mémoire vidéo

100xxxxxxxxxxxxx -> accès tampon clavier

...

Plages mémoire et E/S

- Le mécanisme de sélection par **plages** définit un découpage de l'**espace d'adressage**



Exemple de plan d'espace d'adressage (système 16 bits)

Plages mémoire et E/S

- D'autres systèmes auxiliaires de sélection par adresses (circuits de logique combinatoire, comparateurs d'adresses) permettent d'organiser l'espace d'adressage de manière plus optimale

Plages mémoire et E/S

- Sur ce genre de système on dira que les Entrées/Sorties sont **mappées** en mémoire : on accède aux circuits périphériques en utilisant des adresses particulières
- Lire/Ecrire sur les périphériques se fait avec les mêmes instructions que pour la mémoire. Du point de vue programme les périphériques se comportent comme de la mémoire.
- Sur architectures Intel les E/S se font avec des instructions spéciales IN/OUT ...

Architecture ordinateur

Interruptions et timers

Interruptions

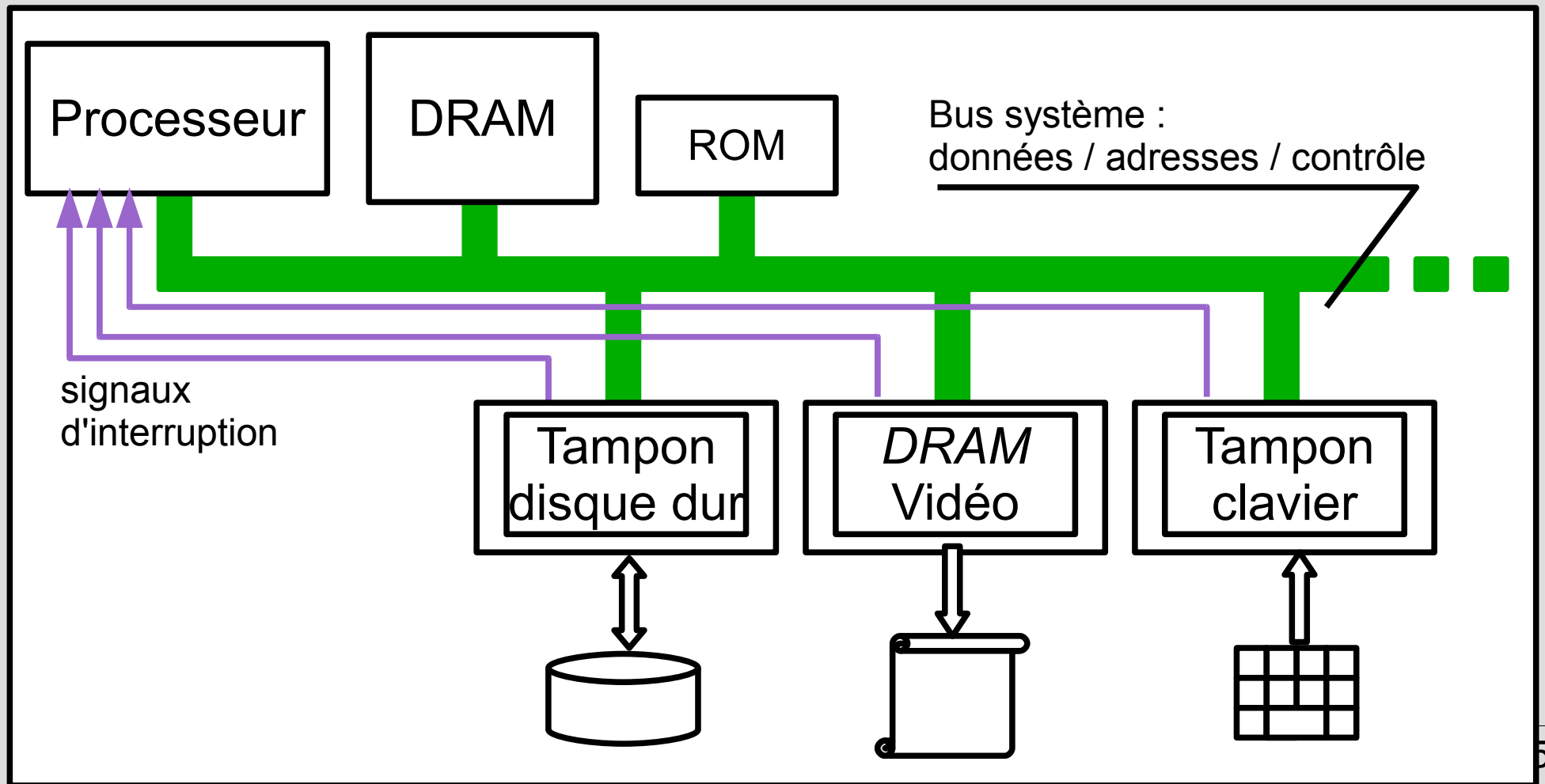
- Les interruptions sont un mécanisme implémenté au niveau matériel pour suspendre provisoirement une tâche en cours, réaliser une autre tâche, et revenir à la tâche de départ
- Le mécanisme est proche de celui de l'appel de sous-programme...
- Mais ici c'est un événement et non un appel explicite qui lance la **routine d'interruption**

Interruptions

- **Les interruptions matérielles** sont générées par un circuit périphérique qui signale au processeur un événement qui peut survenir à n'importe quel moment de l'exécution du programme
 - Une touche est enfoncée ou un bouton est cliqué
 - Une information est disponible sur un port
 - Un timer est arrivé à terme
 - Fin de balayage écran horizontal ou vertical
 - Un bloc d'échantillon sonore vient de se terminer
 - Fin de transfert automatique de mémoire (DMA)

Interruptions

- Circuits d'interruptions matérielles (considérées comme lignes du bus contrôle)

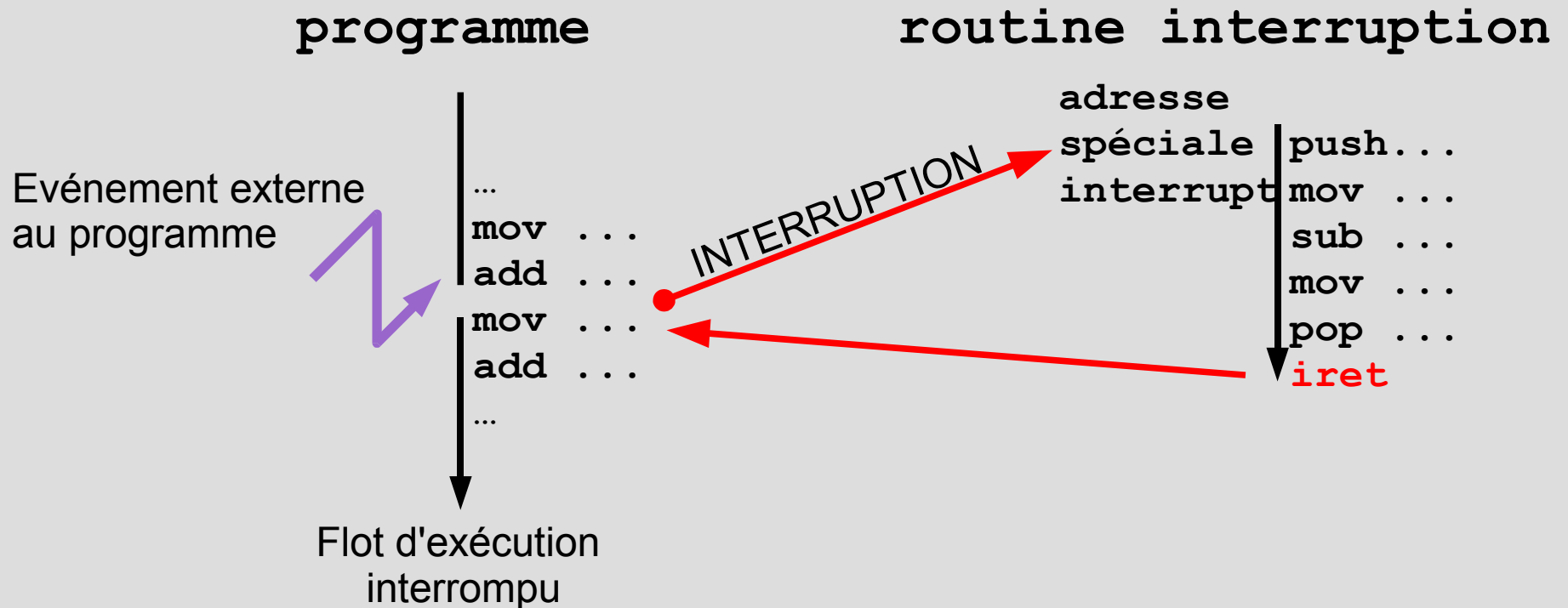


Timers

- Quand l'événement arrive, le processeur
 - termine l'instruction actuelle
 - **branche** l'exécution à l'adresse de la routine
- La routine **sauve** le **contexte d'exécution** actuel (contenu des registres) sur la pile
- Après avoir traité l'événement, la routine **restaure** le contexte d'exécution (dépiler) et retourne à l'exécution interrompue avec une instruction iret : **retour d'interruption**

Interruptions

- Au moment de l'interruption le programme n'a rien à dire : c'est le processeur qui branche...

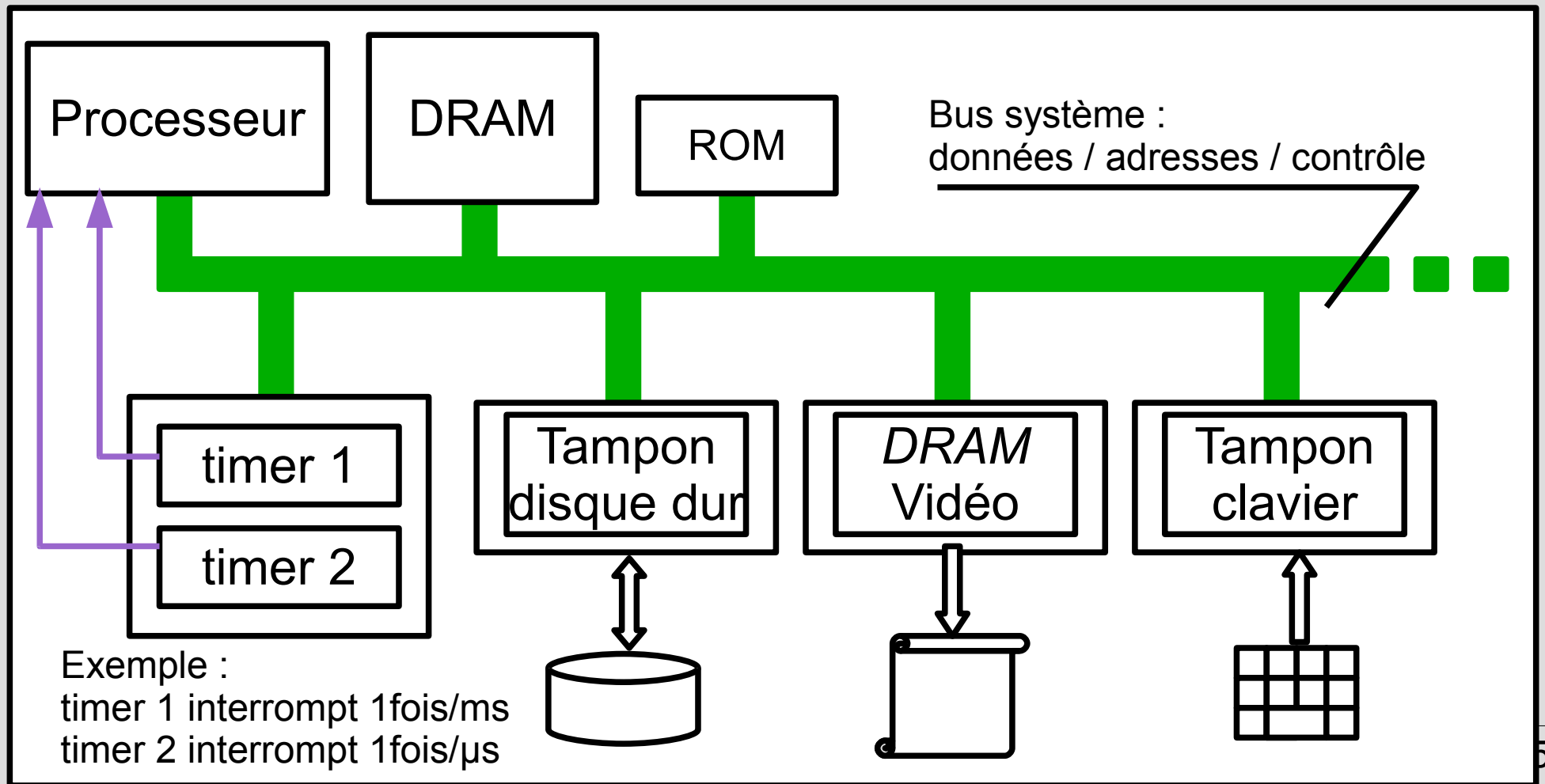


Timers

- Les *timers* sont des compteurs indépendants de l'exécution du programme sur le processeur
- Ils permettent de prévoir le lancement d'une tâche dans un délais précis, ou la répétition d'une tâche à intervalles réguliers
- Ceci nécessite d'utiliser une interruption
- Le nombre de timers matériels est limité par l'architecture : 1 timer = 1 compteur **matériel**
- Leurs fréquences sont paramétrables par le programme (accès comme périphérique)

Timers et Interruptions

- Les timers sont des circuits périphériques



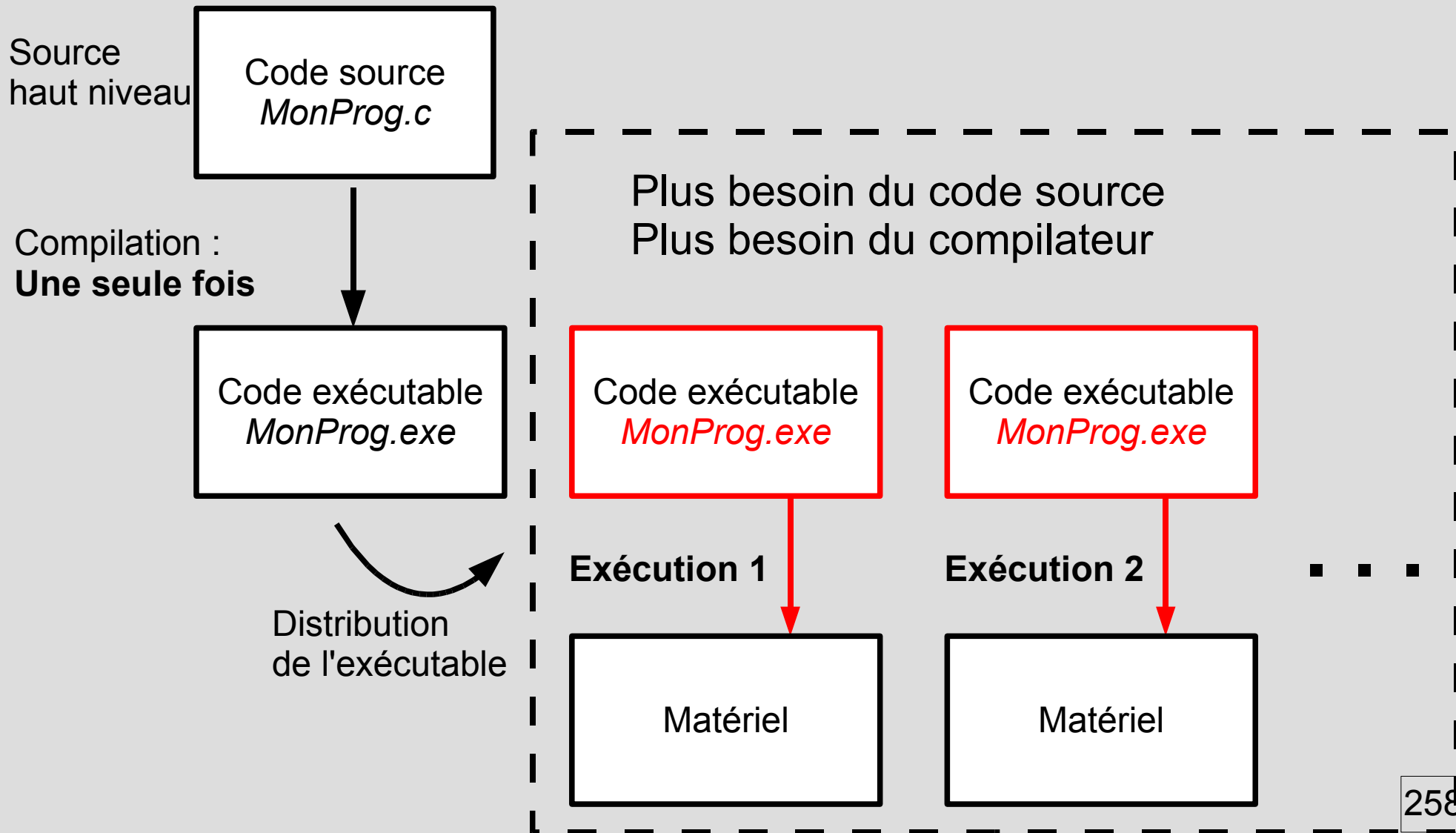
Architecture des ordinateurs

Niveaux de programmation
Compilation et interprétation

Niveaux de programmation

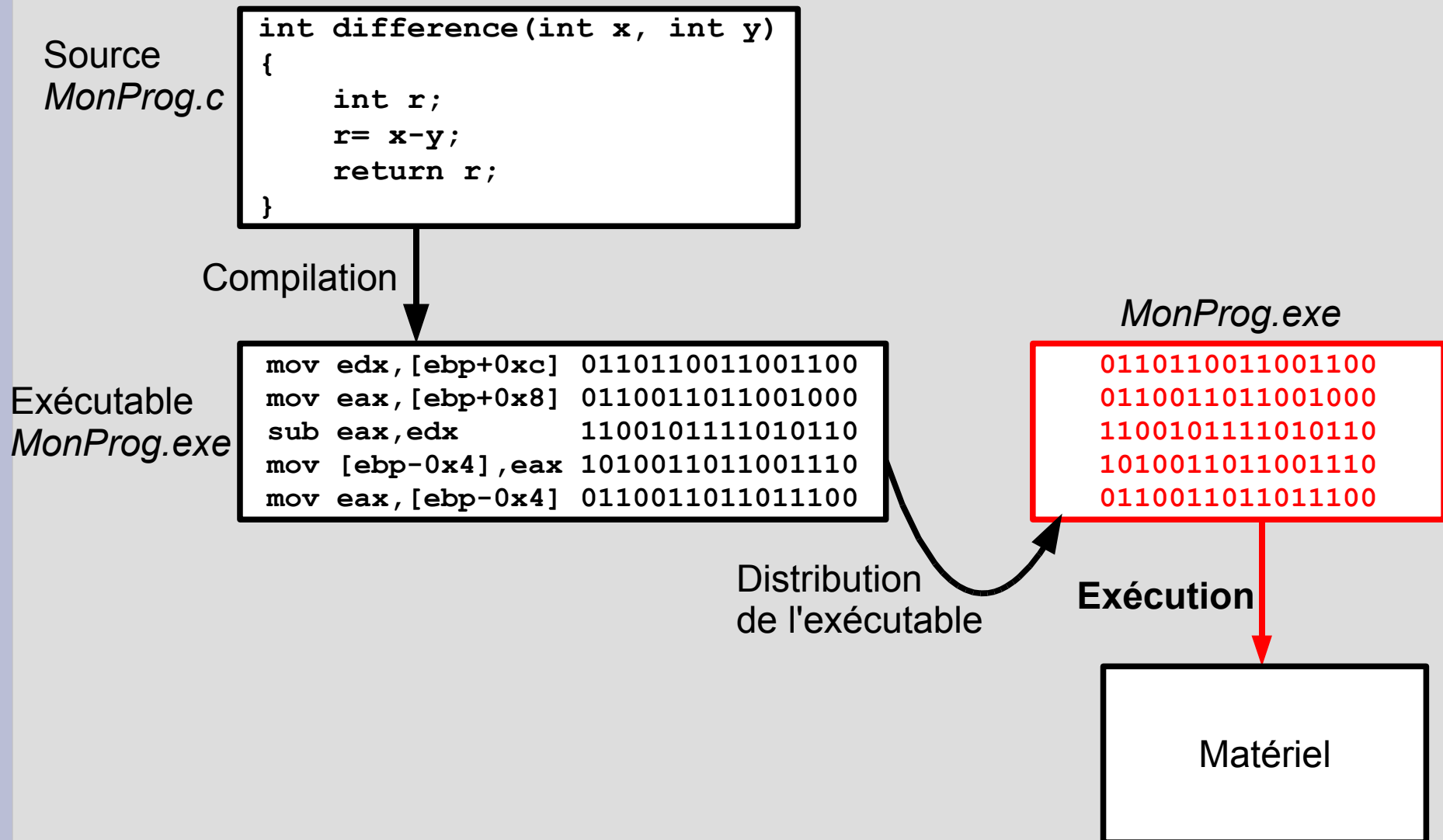


- Principe d'un langage **compilé**



Niveaux de programmation

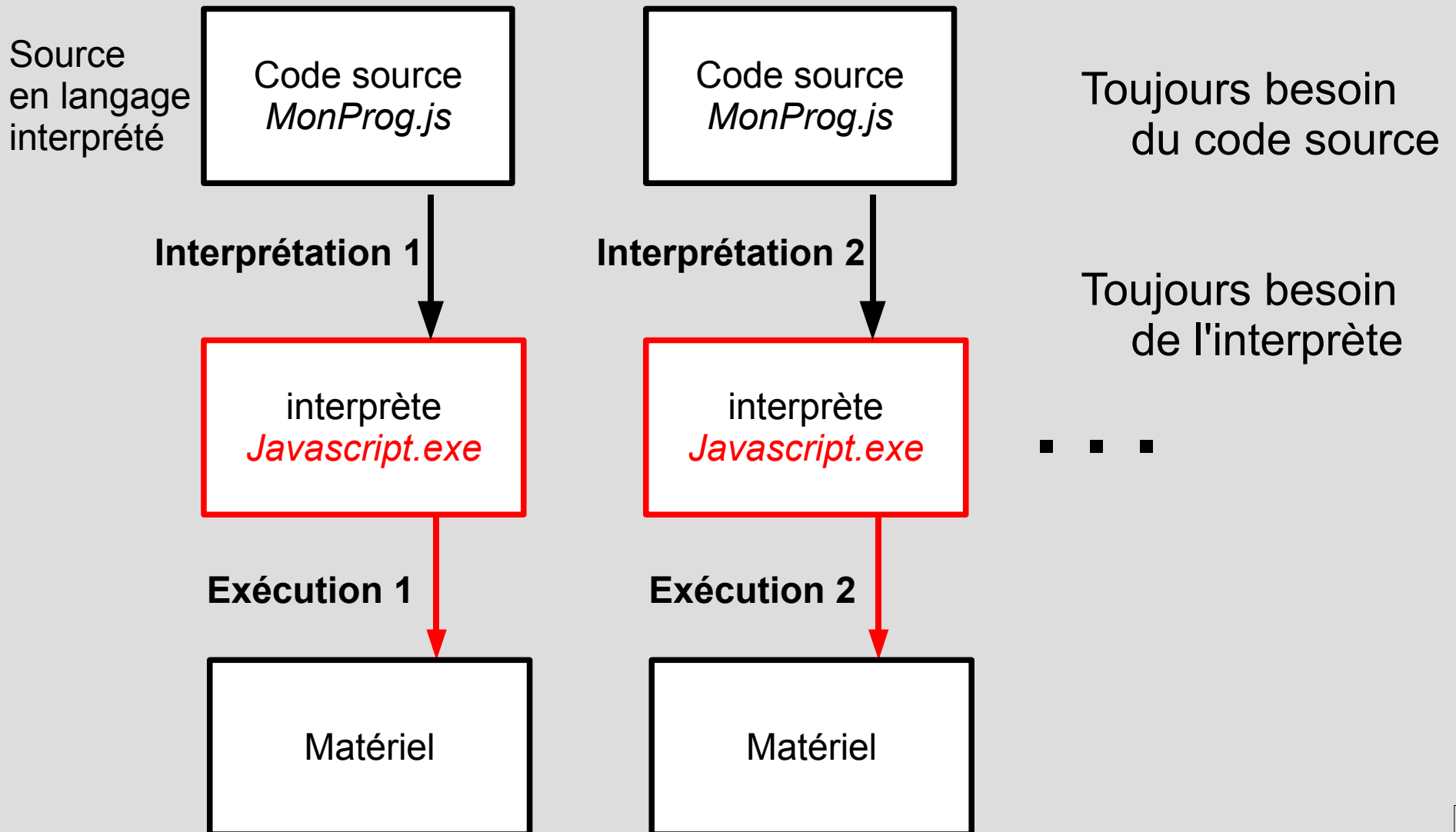
- Principe d'un langage **compilé**



Niveaux de programmation



- Principe d'un langage **interprété**



Niveaux de programmation

- Principe d'un langage interprété

Code source
de l'interprète
Javascript.c

```
char word[100];  
char prog[100000];  
  
// charger fichier prog  
...  
// boucle d'exécution  
do{  
  readNextWord(word,prog);  
  if (!strcmp(word,"if"))  
    ... tester ce qui vient  
  if (!strcmp(word,"for"))  
    ... répéter ce qui vient  
  if (!strcmp(word,"="))  
    ... affecter  
  if (!strcmp(word,"=="))  
    ... calcul booléen égalité  
  ...  
} while (progPasFini);
```

```
function factorial (n){  
  x=1;  
  for (i=2;i<=n;i++) x=x*i;  
  return x;  
}
```

MonProg.js

Interprétation :
exécution "virtuelle" ou
exécution "par procuration"

```
0110110100010100  
0110000110100100  
0010110100001010  
1100110111110110  
0100100100010110  
0111110100111100  
0110110100110100
```

Javascript.exe

compilé
(une seule fois
pour chaque
type de
processeur)

Exécution "réelle" de l'interprète
sur le processeur

Matériel

Niveaux de programmation



- Langages interprétés :
 - Avantages
 - souplesse, le code est modifiable en cours d'interprétation
 - sécurité, l'interprète vérifie l'innocuité des actions
 - langages plus "dynamiques" (ressources dynamiques...)
 - portable : il suffit que le système x ou y ait l'interprète
 - le code source est toujours accessible
 - Inconvénients
 - plus lourd, plus lent : il faut tout retraduire à chaque fois
 - il faut (re)traduire au fur et à mesure (boucles !)
 - le code source est toujours accessible

Niveaux de programmation

- Le cas java
 - java est un langage "compilé/interprété" : le **compilateur** javac convertit un fichier source .java en fichier .class qui est écrit en **bytecode**
 - Le bytecode est une sorte de jeu d'instruction indépendant de tout processeur, il n'est pas directement exécutable : un interprète de bytecode est nécessaire, c'est la **Java Virtual Machine**

Niveaux de programmation

- Principe d'un langage **compilé en bytecode**

Code source en java
MonProg.java

```
...  
  
outer:  
for (int i = 2; i < 1000; i++)  
{  
    for (int j = 2; j < i; j++)  
    {  
        if (i % j == 0)  
            continue outer;  
    }  
    System.out.println (i);  
}  
  
...
```

Compilation
en bytecode
(javac)

MonProg.class

| | |
|----------|------------------|
| iconst_2 | 00101001 |
| istore_2 | 11011001 |
| iload_2 | 11011011 |
| iload_1 | 01111011 |
| irem | 11001001 |
| ifne 25 | 0101010101101110 |
| goto 38 | 1100100111110100 |

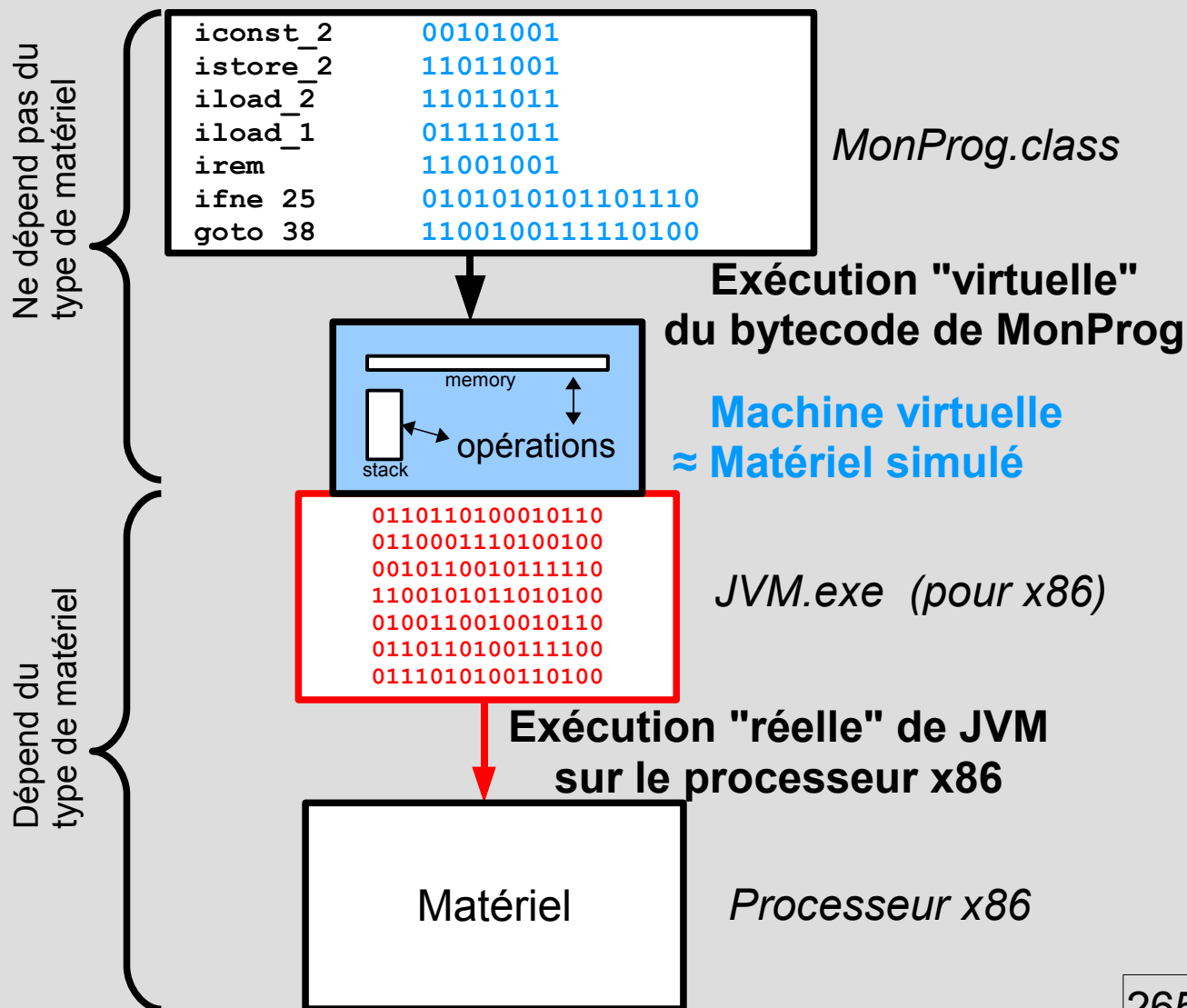
Le bytecode java est standardisé : c'est une sorte de langage machine pour une "machine" standard, la machine virtuelle java

Niveaux de programmation

- Principe d'un langage compilé en bytecode

Code source
de la machine virtuelle
JVM.c

```
char byte;  
char prog[100000];  
char memory[100000];  
int stack[10000];  
  
// charger fichier prog  
...  
// boucle d'exécution  
do{  
  byte=readNextByte(prog);  
  switch(byte)  
  {  
    case 0: ...  
    case 1: ...  
    case 2: ...  
    case 3: ...  
    case 4: ...  
    ...  
  }  
} while (progPasFini);
```



Niveaux de programmation

- Le cas java
 - Comme le bytecode est simple, la machine virtuelle est "simple" et légère et efficace.
 - La "précompilation" (une seule fois) facilite donc le travail d'interprétation (à chaque exécution) plus tard
 - L'exécution reste cependant moins performante que pour du code C ou C++ compilé
 - Les "machines virtuelles" java plus sophistiquées peuvent compiler le bytecode en langage machine natif (plate forme présente) de manière dynamique pour optimiser l'exécution et se rapprocher des performances du C ou C++

Architecture des ordinateurs

Evolutions technologiques

Puissance des processeurs

- La fréquence de l'horloge est la **fréquence du processeur** (de quelques MHz à 3GHz)
- Une mesure de puissance est le nombre de **millions d'instructions par secondes: MIPS**
 - ♦ $MIPS = Nbr_inst / (Tps_exécution \times 10^6)$
 - ♦ $MIPS = Freq / (CPI \times 10^6)$
CPI = Cycles par instruction (en moyenne)
- C'est une indication de la puissance de traitement générique du processeur

Puissance des processeurs

- Une autre mesure concerne spécifiquement le calcul avec des nombres décimaux (flottants)
 - $\text{FLOPS} = \text{Nbr_calc_flottant} / \text{Tps_exécution}$
 - L'unité est le **FLOPS** (invariant singulier/pluriel) :
Floating Point Operations Per Second
- Le calcul flottant est utilisé intensivement par le calcul scientifique et technique (simulations...) et aussi la 3D (cartes graphiques pour jeux...)
- Les performances des processeurs en calcul flottant dépendent de la présence d'une ou plusieurs unités de calcul en flottant (FPU)

Puissance des processeurs

- La puissance de traitement n'est pas le seul critère à prendre en considération :
 - Compatibilité et outils de développement logiciel
 - Interfaçage et familles de circuits auxiliaires
 - Coût du composant
 - Consommation électrique, dissipation thermique
 - Spécialités (embarqué, militaire, aérospatial...)

Progrès de l'intégration



- La loi de Moore prédit un doublement du nombre de transistors des processeurs tous les 2 ans : progression exponentielle
- Dimension actuelle de gravure: 45nm→32nm
- La miniaturisation des éléments gravés conduit à des échelles d'ordre moléculaire ~ 2015/2020 il faudra alors changer de technologie.
- Le coût d'une usine de production de nouveaux processeurs tend aussi à croître de manière exponentielle (loi de Rock)

Intégration et fréquence

- Avec des transistors (des portes logiques) commutant à plus grande fréquence on augmente le nombre de MIPS
- On sait commuter jusqu'à 500 GHz mais la dissipation thermique devient énorme
- Ces problèmes de consommation et de dissipation font que les processeurs actuels plafonnent à 3 GHz depuis quelques années...

Intégration et fréquence



- **L'intégration** continue de progresser mais les fréquences stagnent :
 - On peut faire des processeurs toujours plus petits et moins consommateurs (mais pas plus puissants)
 - On peut faire des processeurs qui font plus de choses (mais pas une seule chose plus vite)

Architecture des ordinateurs

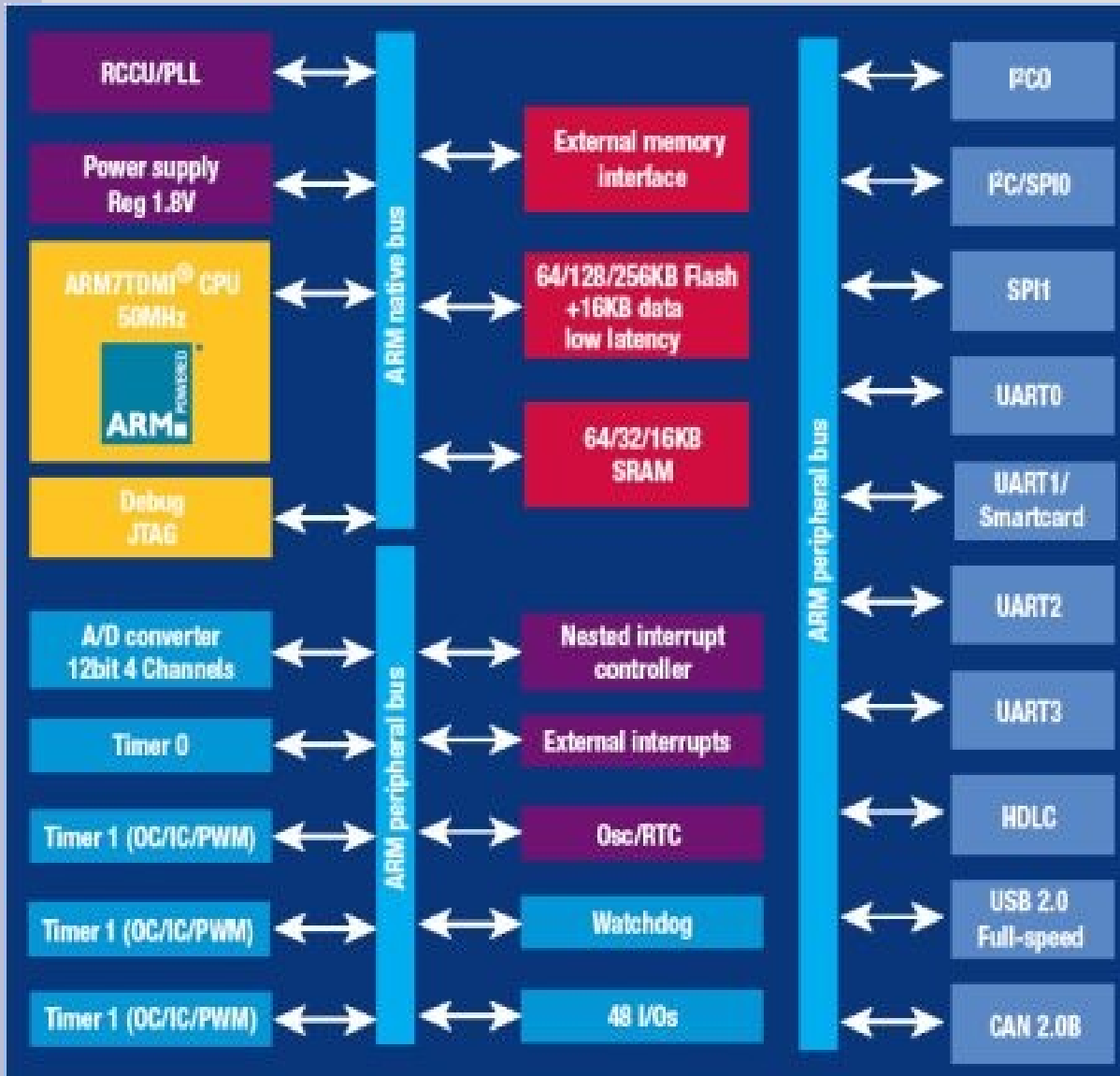
Systemes embarqués,
Systemes intégrés

Systemes intégrés



- Un **microprocesseur** d'ordinateur est spécialisé sur les fonctions de traitement de l'information, il nécessite en plus des composants de mémoire, d'entrées/sorties, de gestion d'énergie...
- Un **microcontrôleur** est un composant qui regroupe **sur un seul circuit intégré** l'ensemble des fonction d'un système informatique : c'est un ordinateur sur une puce

Systemes integrés



Exemple de microcontrôleur à base d'ARM7TDMI :

STR710

ST Microelectronics



Systemes intégrés

- Les microcontrôleurs 8 bits les plus simples sont aussi les plus économiques et restent très utilisés dans les applications embarquées nécessitant un "composant programmable"
 - Télécommandes, correction d'erreur, cryptage
 - Interfaces, adaptateurs de protocoles
 - Gestion de servomoteurs, asservissements
 - Gestion d'alimentation, chargeurs de batteries
 - Jouets, jeux interactifs, gadgets divers ...

Systemes intégrés

- Ces microcontrôleurs industriels présentent de nombreuses entrées sorties, des *timers* permettant de réaliser des tâches à intervalles très réguliers, de faibles latences en réponse aux entrées imprévues (interruptions) ...
- Quantité de mémoire et puissance de calcul restent souvent très limité

Systemes intégrés

- A côté des applications légères s'est développé un besoin de puissance en électronique embarquée :
 - Téléphonie mobile
 - Terminaux et gadgets multimédias
 - GPS, afficheurs automobile ...
 - Routeurs, modems
 - Périphériques : imprimantes, disques durs...

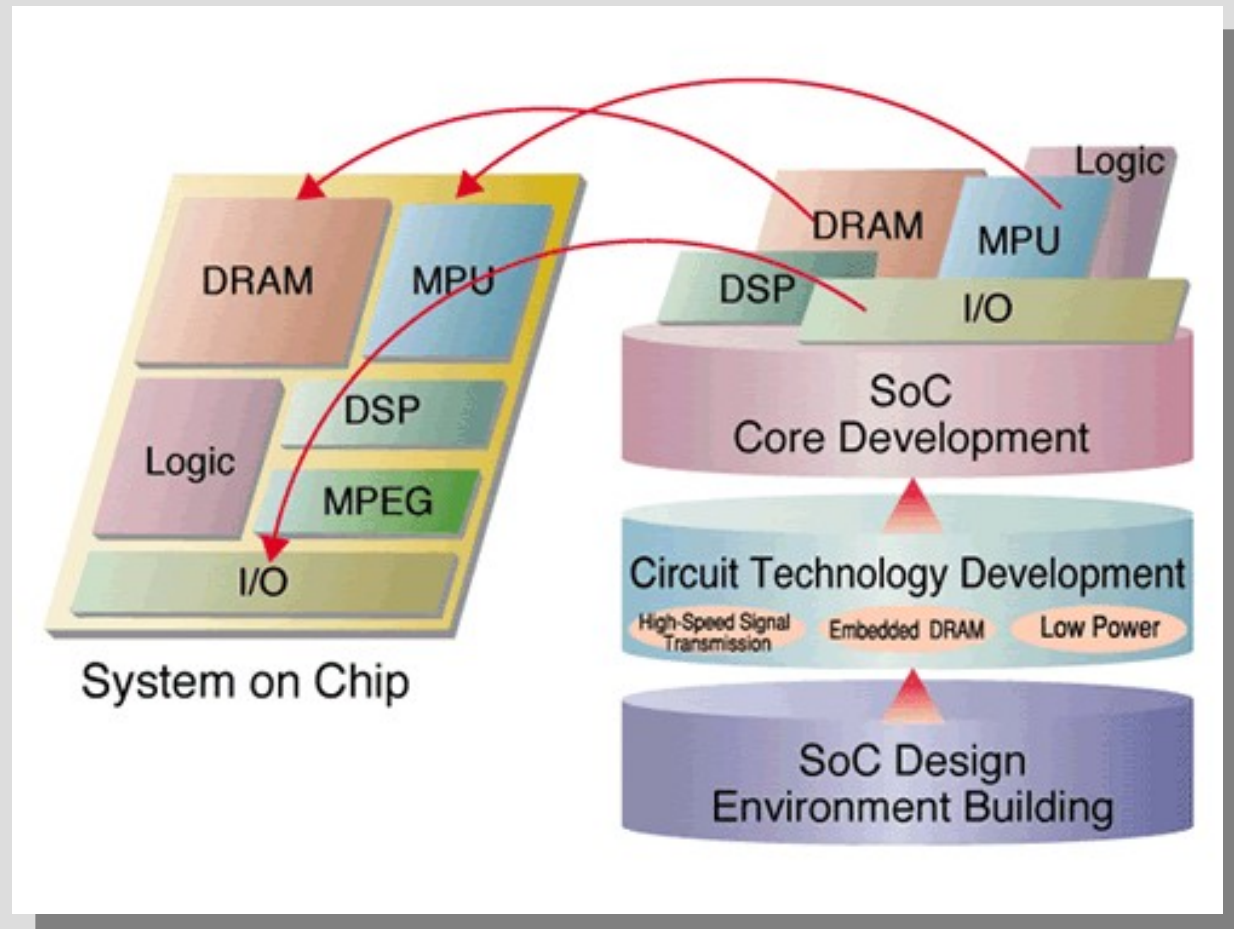
Systemes integrés

- Pour ces systemes puissants et complexes mais portable il y a aussi nécessité de regrouper toutes les fonctions en un seul composant :
 - Légèreté et compacité
 - Robustesse (minimum de soudures)
 - Economie d'énergie (bus réduits, faibles niveaux)
 - Faibles coûts de production (pas d'assemblage)
- On parle de plus en plus de SoC :
System on Chip

Systemes intégres



L'idée est de développer un systme complet à partir de bibliothèques de briques matérielles (blocs IPs) et de briques logicielles (μ noyaux, bibliothèques spécialisées...)



Systemes intégrés



- Différents modules plus ou moins spécialisés :
 - MPU Micro Processor Unit
synonyme de CPU (coeur processeur polyvalent)
 - Entrées/Sorties numériques ou analogiques
Ports de communication, contrôleurs d'affichage...
DAC Digital to Analog Converter
ADC Analog to Digital Converter
 - DSP Digital Signal Processor
traitement du signal numérique (filtrage, contrôle...)
 - Spécialités, codage/décodage mp3 ...

Systemes intégrés



- Pour de grosses séries un fabricant peut investir dans la fabrication de masques et lancer une production de composants ASICs
Application-Specific Integrated Circuits
- Pour du prototypage, des petites séries ou pour être plus réactif il est possible d'utiliser de la logique programmable : FPGAs
Field-Programmable Gate Arrays

Architecture système

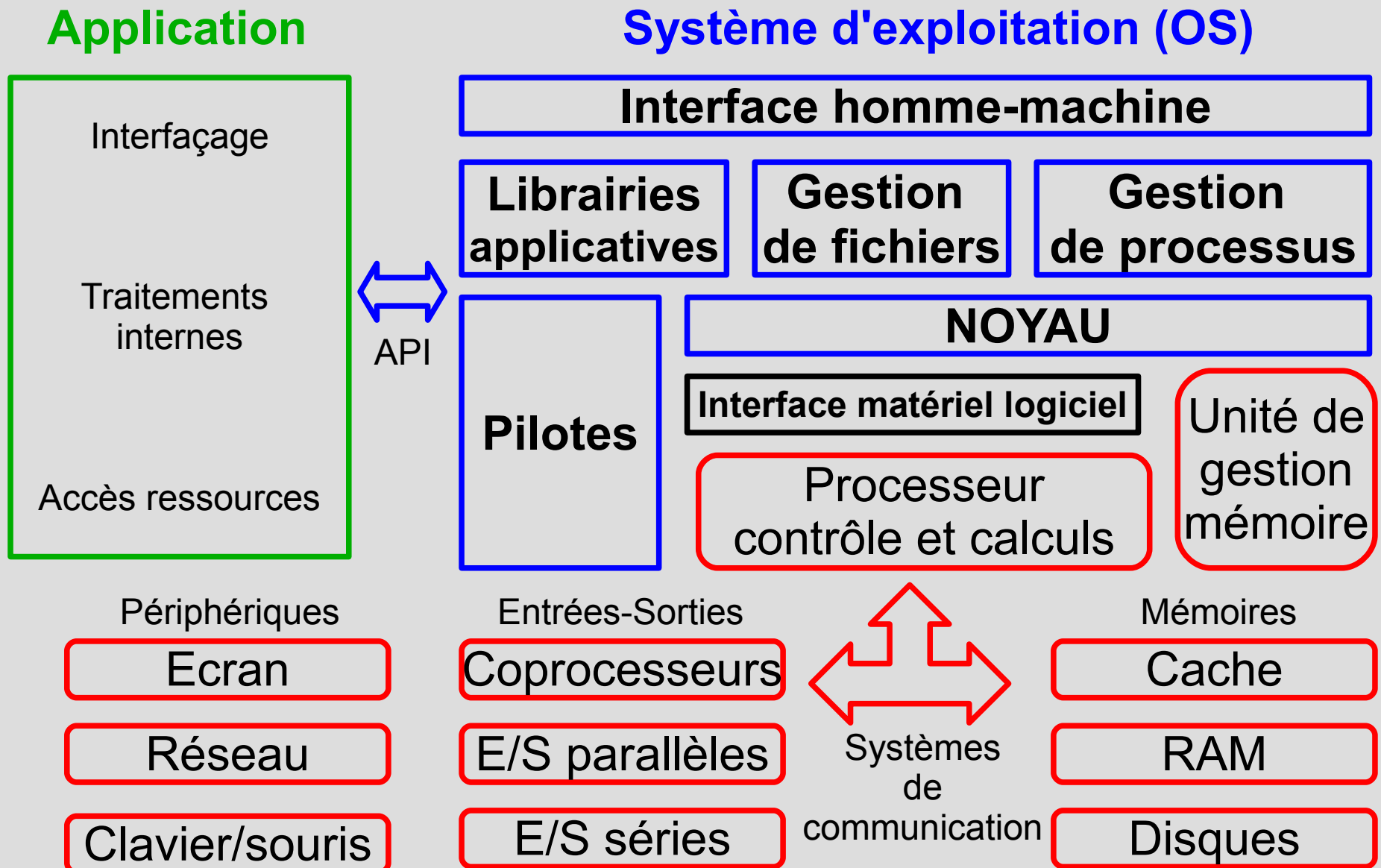
Notions sur le noyau système et architectures évoluées

Architecture système



- Un système informatique moderne est doté d'un ensemble de modules logiciels destinés à faciliter et sécuriser l'accès aux ressources matérielles pour l'utilisateur et le développeur. C'est le **système d'exploitation = OS** :
 - L'interface du système d'exploitation : menus, fenêtres
 - Les fonctions standard du système d'exploitation utilisées par API : *Application Programming Interface*
 - La gestion bas niveau des ressources matérielles : c'est le rôle du **noyau** du système (*kernel*) et des **pilotes** de périphériques

Architecture système



Architecture système

- Sur un système informatique polyvalent on souhaite pouvoir utiliser simultanément plusieurs applications
- Un **processus** est une application en cours d'utilisation
- On parle de système **multitâche** quand le noyau est capable d'arbitrer dynamiquement la ressource processeur pour les différents processus
 - **ordonnanceur**
routine appelée à intervalle fixe par interruption (timer)
- L'autre rôle du noyau est d'allouer la ressource mémoire aux différents processus
 - **gestionnaire de mémoire**
composant logiciel s'appuyant sur une unité matérielle

Architecture système

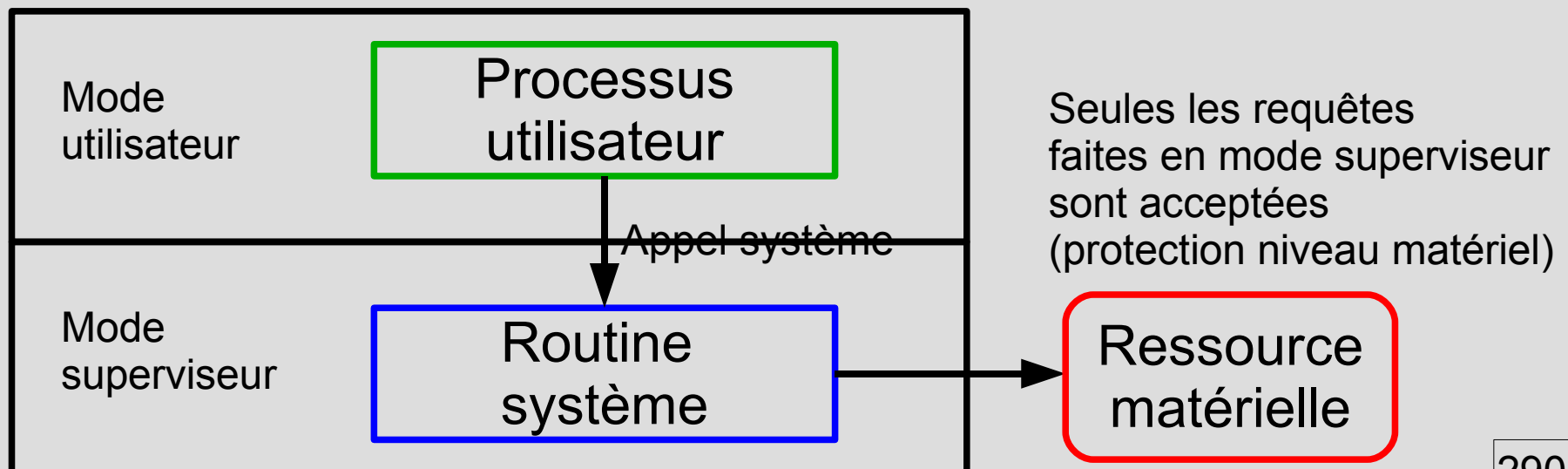


- Pour maintenir l'intégrité du fonctionnement d'ensemble, le système doit se voir réserver certains privilèges inaccessibles aux applications
- Cette protection doit opérer au niveau matériel, le processeur dispose de 2 modes d'exécution :
 - Le **mode utilisateur** pour les applications et les modules non critiques du système
 - Le **mode superviseur** qui bénéficie des privilèges et est réservé au noyau et aux pilotes de périphériques

Architecture système



- Pour les processus utilisateur, l'obtention des services critiques du système (allocations de ressources matérielles : processeur, mémoire, E/S) passe par des **appels systèmes**
- L'appel système est une interruption logicielle qui passe le processeur en mode superviseur



Architecture système



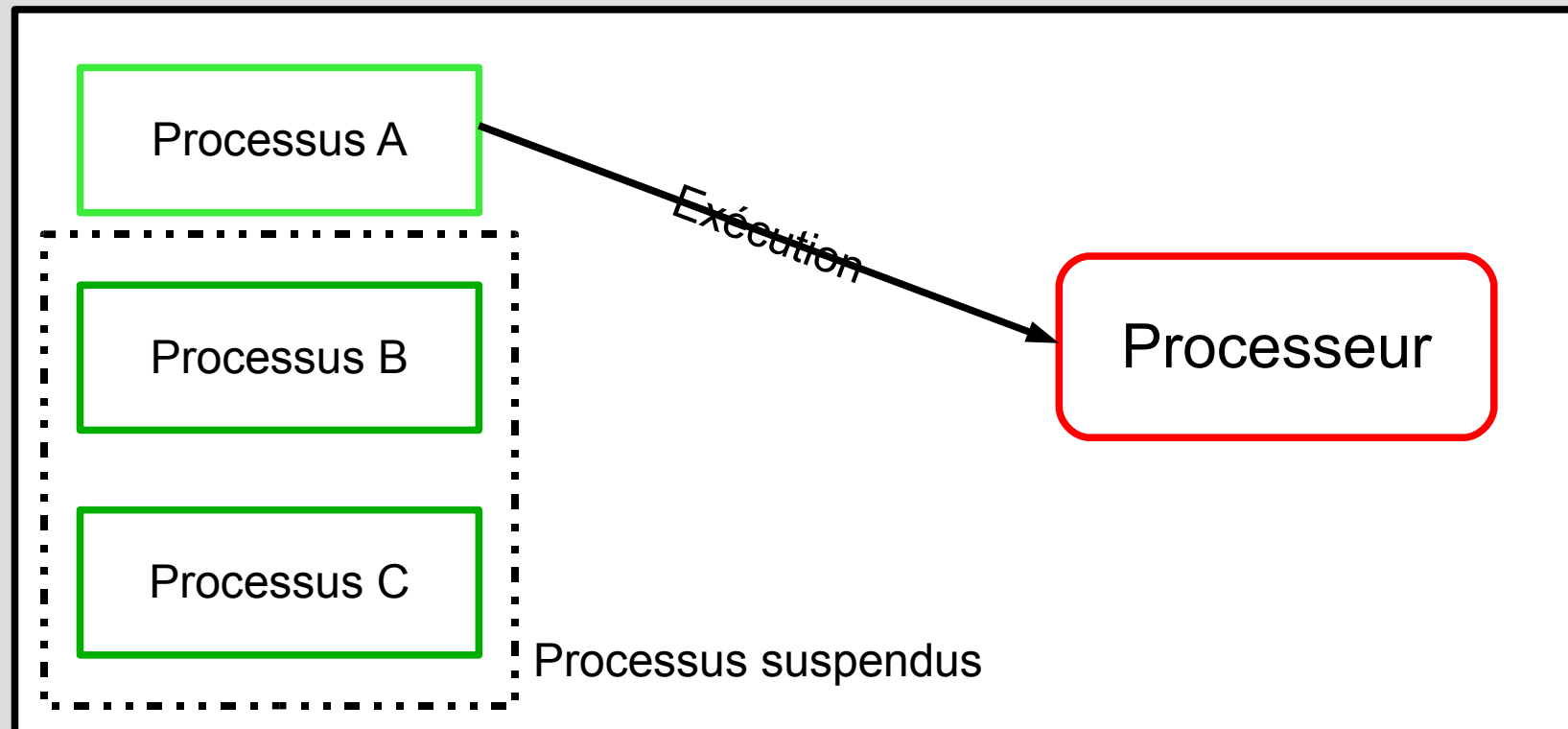
- Le noyau est l'ensemble des modules logiciels du système d'exploitation fonctionnant en mode superviseur
- Tout processus utilisateur peut demander des ressources matérielles mais il doit passer par un appel système. Il est impossible de basculer en mode superviseur sans utiliser un tel appel.
- Si le noyau système est correctement conçu, la demande n'aboutit que si elle ne remet pas en cause l'intégrité de l'ensemble ou les droits utilisateurs (sur les systèmes multi-utilisateurs)

Architecture système

**Multitâche préemptif
et ordonnanceur**

Multitâche

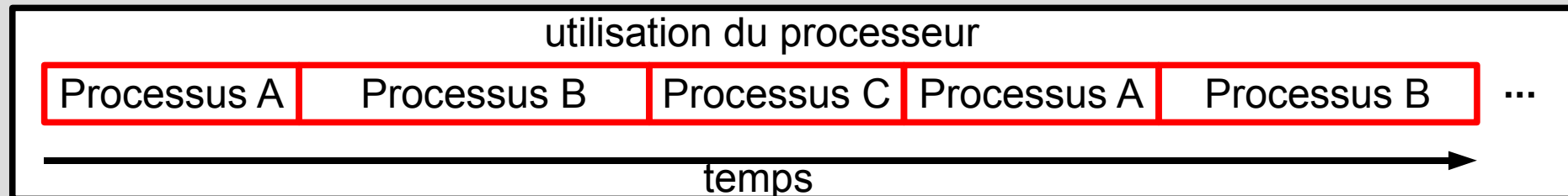
- Problème du multitâche : le processeur est unique et ne peut exécuter qu'un seul processus à la fois à un instant donné



Multitâche



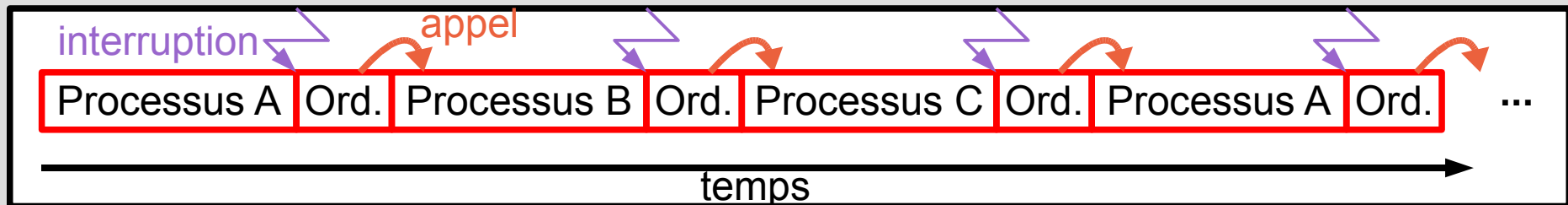
- Sur un système mono-processeur le principe du multitâche est celui du temps partagé : chaque processus utilisera le processeur à tour de rôle pour avancer dans son exécution



- L'**ordonnanceur** du noyau est responsable d'assurer automatiquement le passage sécurisé d'un processus à un autre
- Pour donner l'illusion d'activités simultanées le quantum de temps processeur alloué à chaque segment d'exécution est court (~1ms)

Architecture système

- On parle de multitâche préemptif quand le passage d'un processus au suivant est géré par une routine appelée à intervalles réguliers par interruption.
- Le processus en cours est automatiquement suspendu
- La routine d'interruption est responsable de choisir le prochain processus à rendre actif :
c'est l'**ordonnanceur**



- Cette solution est robuste car elle garanti un temps d'exécution équitable à chaque processus
- Il est possible de "tuer" les processus plantés

Architecture système

- L'ordonnanceur doit sauver en mémoire les informations du processus suspendu pour pouvoir le relancer plus tard en les rechargeant : c'est la **commutation de contexte** (on remet en place les registres, les codes condition et le PC)
- Un timer matériel déclenche l'interruption pour activer l'ordonnanceur à intervalles périodiques
- Il fixe le **quantum** de temps alloué aux processus : typiquement 1ms

Architecture système

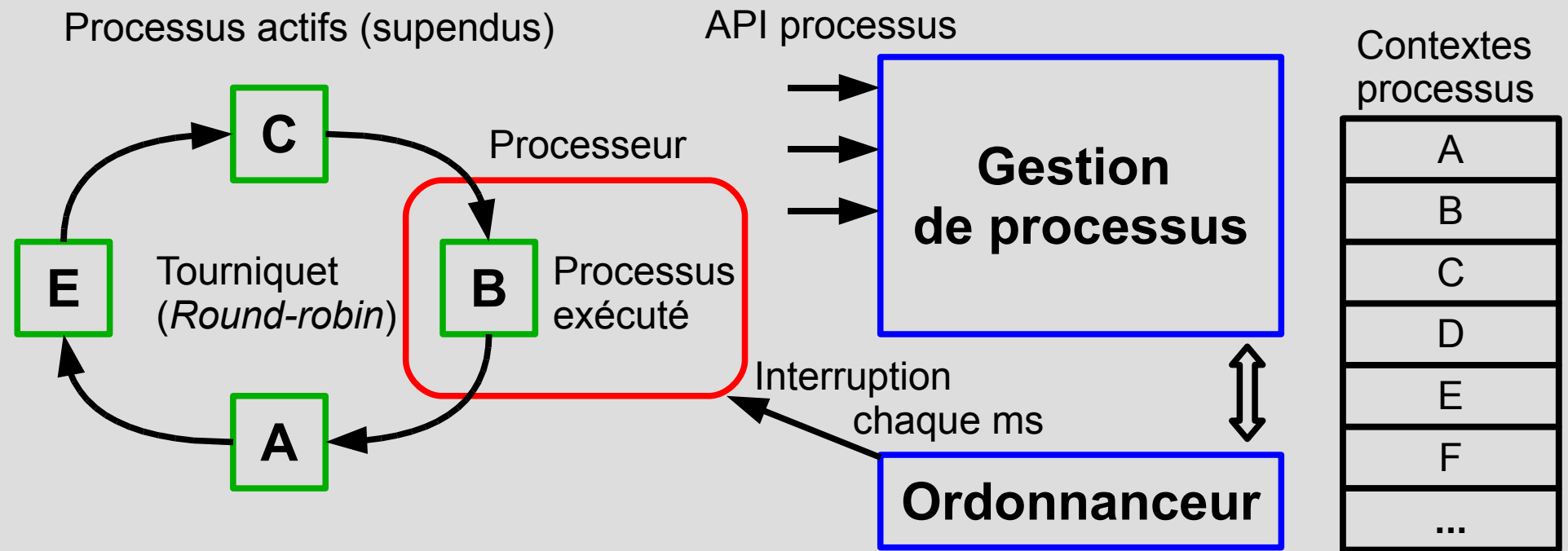


- L'ordonnanceur est un module logiciel qui s'exécute en contexte d'interruption: son temps d'exécution doit être aussi court que possible
- Sur un système moderne évolué (Linux, OSX, Windows ...) la gestion des processus met en jeu de nombreux autres mécanismes, priorités, attentes d'E/S ...
- Il y a donc un module "gestion de processus" de plus haut niveau pour diriger l'activité de l'ordonnanceur

Architecture système



- Schéma général du multitâche préemptif



Processus en attente d'événement E/S



API processus :

- Créer nouveau processus
- Terminer processus
- Attente d'événement E/S

Architecture système

**Gestion mémoire
mémoire virtuelle
mémoire cache**

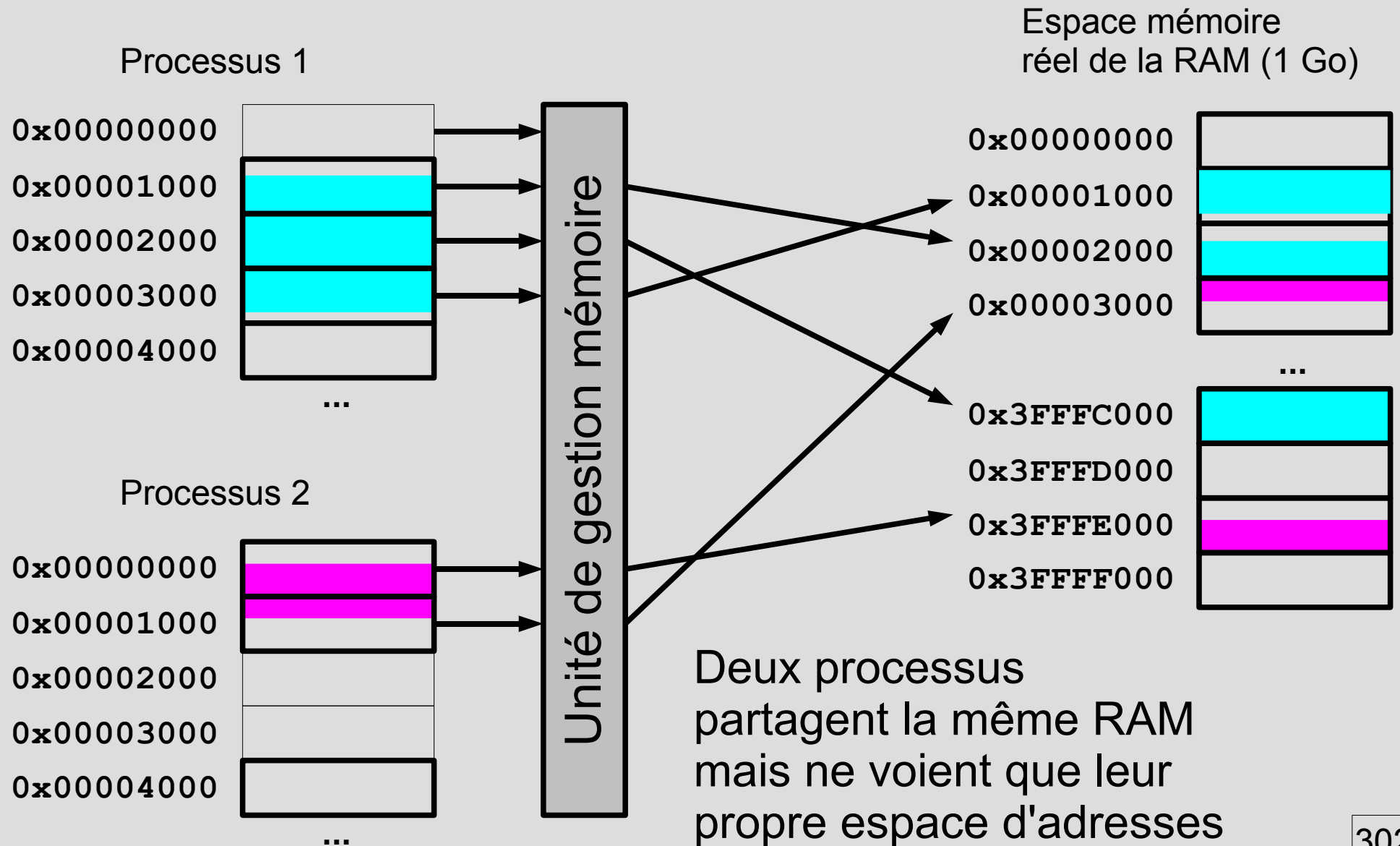
Organisation mémoires

- Sur les systèmes polyvalents évolués on souhaite que chaque processus dispose d'un espace mémoire privé
- De nombreux processus souhaitent disposer d'une importante quantité de mémoire et ne pas tenir compte des adresses des autres processus
- Ceci conduit d'une part à utiliser une mémoire de masse (disque dur) pour stocker une partie des informations des processus lorsque la RAM est pleine, d'autre part à utiliser des adresses logiques distinctes des adresses RAM réelles

Organisation mémoires

- Le mécanisme utilisé est celui de la mémoire paginée, l'espace mémoire est découpé en blocs de tailles fixes (**pages mémoire**, typiquement 4ko)
- Un système matériel établit les correspondances entre les adresses logiques utilisées par le programme dans son espace d'adressage privé et les adresses réelles des pages dans la RAM
- Ce dispositif est l'unité de gestion de mémoire MMU : ***Memory Management Unit***
- Les processeurs récents en sont tous pourvus

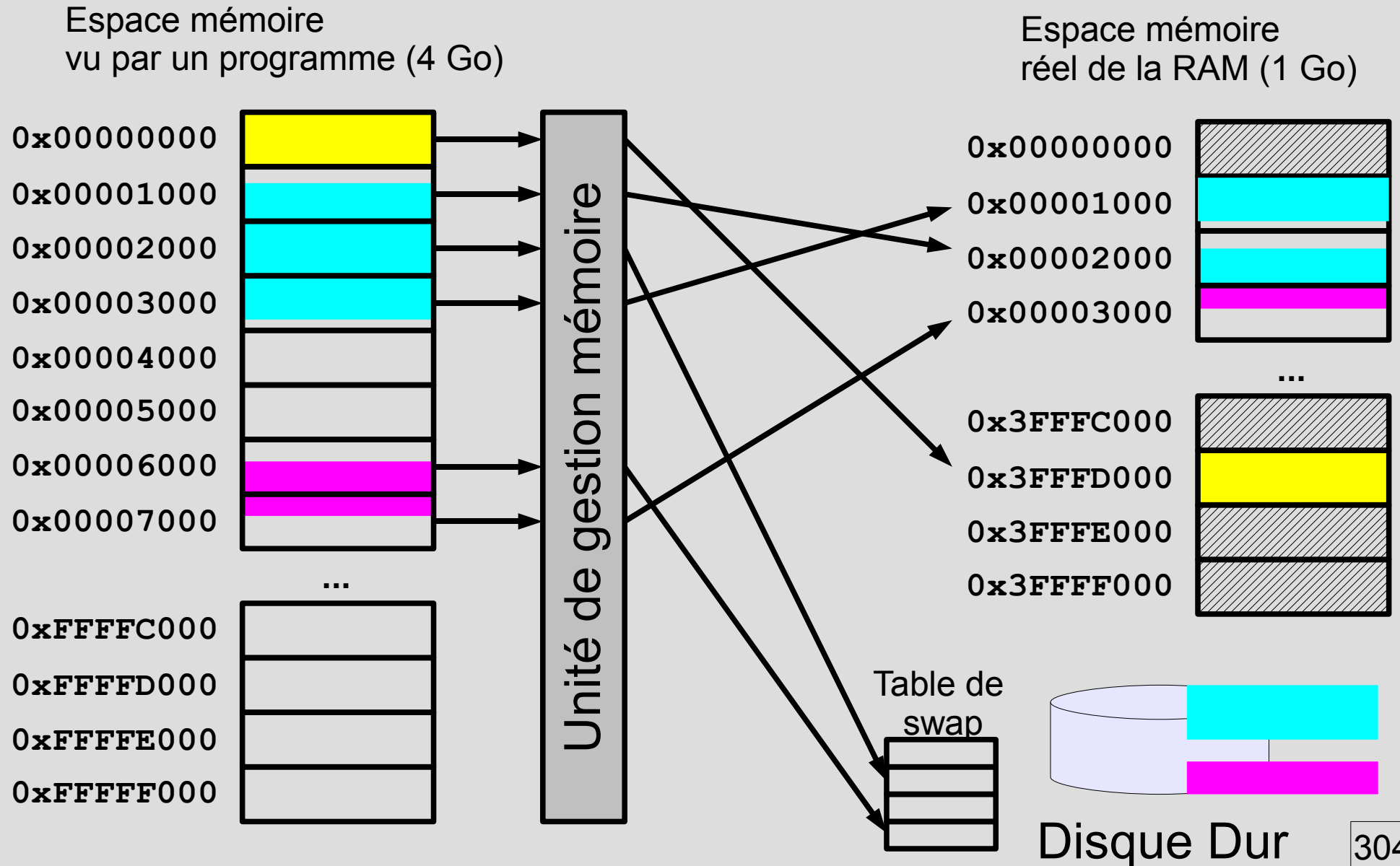
Organisation mémoires



Organisation mémoires

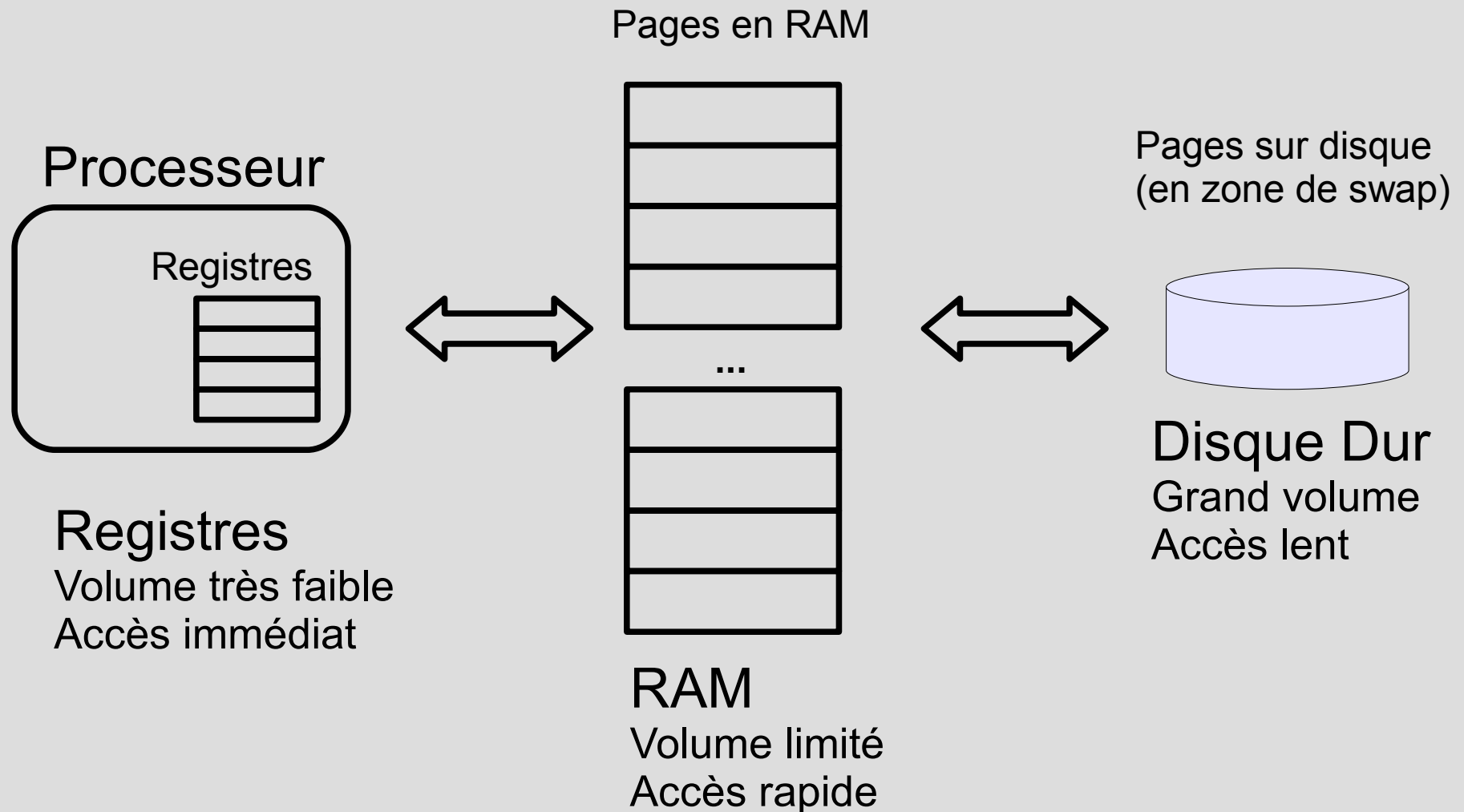
- Lorsque la RAM est pleine on transfère des pages vers le disque dur sur une zone de stockage temporaire : mécanisme de *swap*
- Si un processus accède à une adresse qui correspond à une page qui n'est pas en RAM, la MMU déclenche une routine du noyau qui doit accéder au disque dur pour rapatrier la page en RAM
- Si il faut renoncer à maintenir en RAM une autre page (faute de place) on choisit la plus anciennement accédée pour "swapper"

Organisation mémoires



Organisation mémoires

Organisation conceptuelle de la mémoire virtuelle



Organisation mémoires

- Ce mécanisme de **mémoire virtuelle** permet de faire fonctionner en multitâche un ensemble d'applications totalisant en mémoire réservée plus que le volume de la RAM réelle
- Cependant les temps d'accès au disque dur sont longs. L'ensemble du système fonctionne mais au ralenti...
- Il faut alors renoncer à utiliser autant d'applications en même temps, ou augmenter la mémoire vive réelle (RAM)

Organisation mémoires

- Les mémoires RAM utilisées pour la mémoire principale (technologie DRAM) ne progressent pas autant en vitesse que la fréquence des processeurs
- Processeurs ~ 3 Ghz $\rightarrow 0,33$ ns entre 2 instructions
- DRAM latence accès ~ 10 ns
(Les DDR3 de dernière génération augmentent les débits sur des séquences de données contigües mais la latence en accès aléatoire reste élevée)
- Dans cette situation le processeur passe la plupart de son temps à attendre les informations (instructions ou données) demandées à la RAM

Organisation mémoires

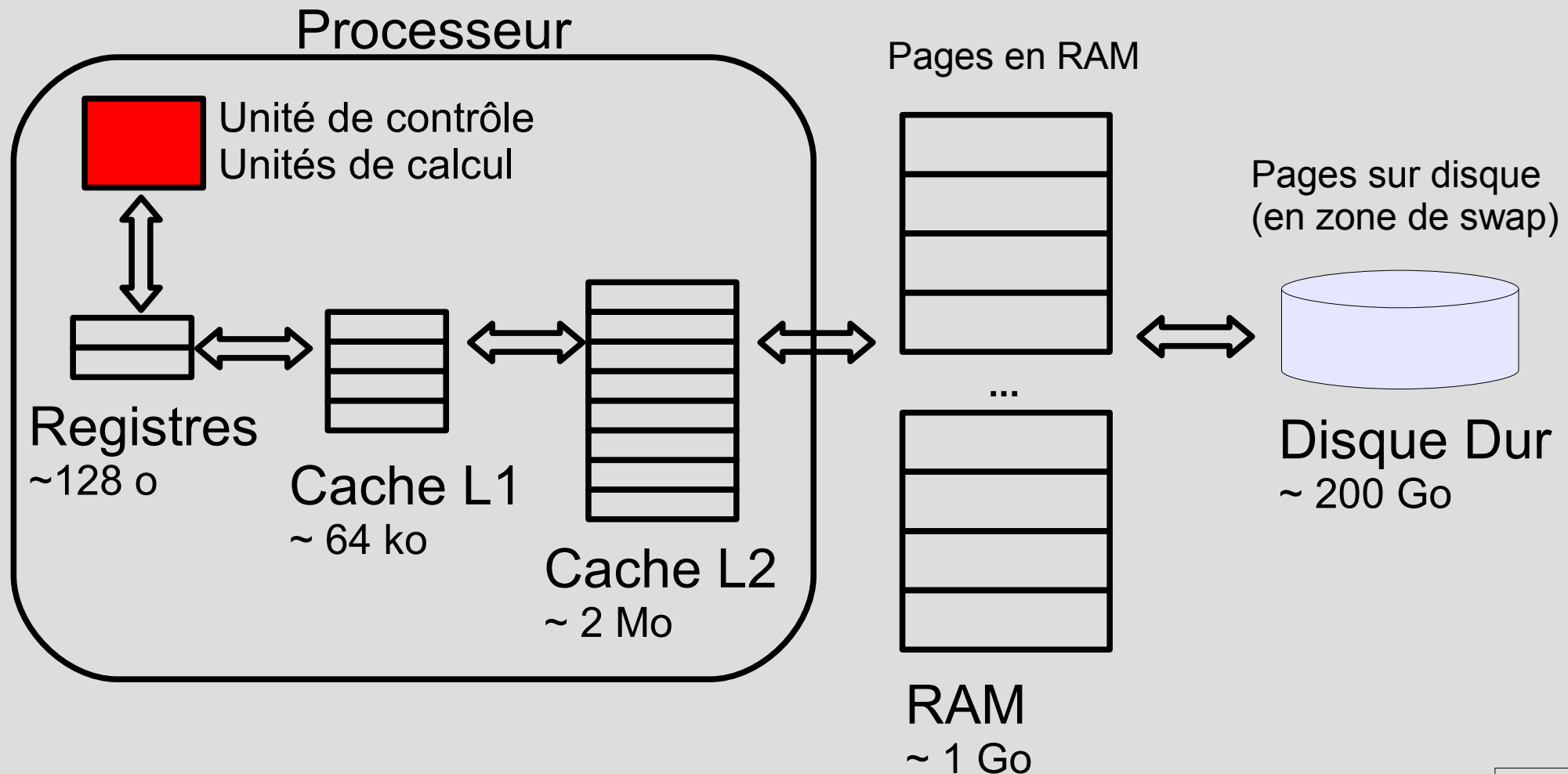


- Pour éviter cette situation on insère une mémoire beaucoup plus rapide (technologie SRAM) entre la mémoire principale et le processeur
- Il s'agit de la **mémoire cache**
- Le principe est comparable avec celui de mémoire virtuelle : une mémoire rapide mais limitée proche du processeur, une mémoire plus lente mais plus volumineuse en soutien

Organisation mémoires



Organisation conceptuelle de la mémoire virtuelle



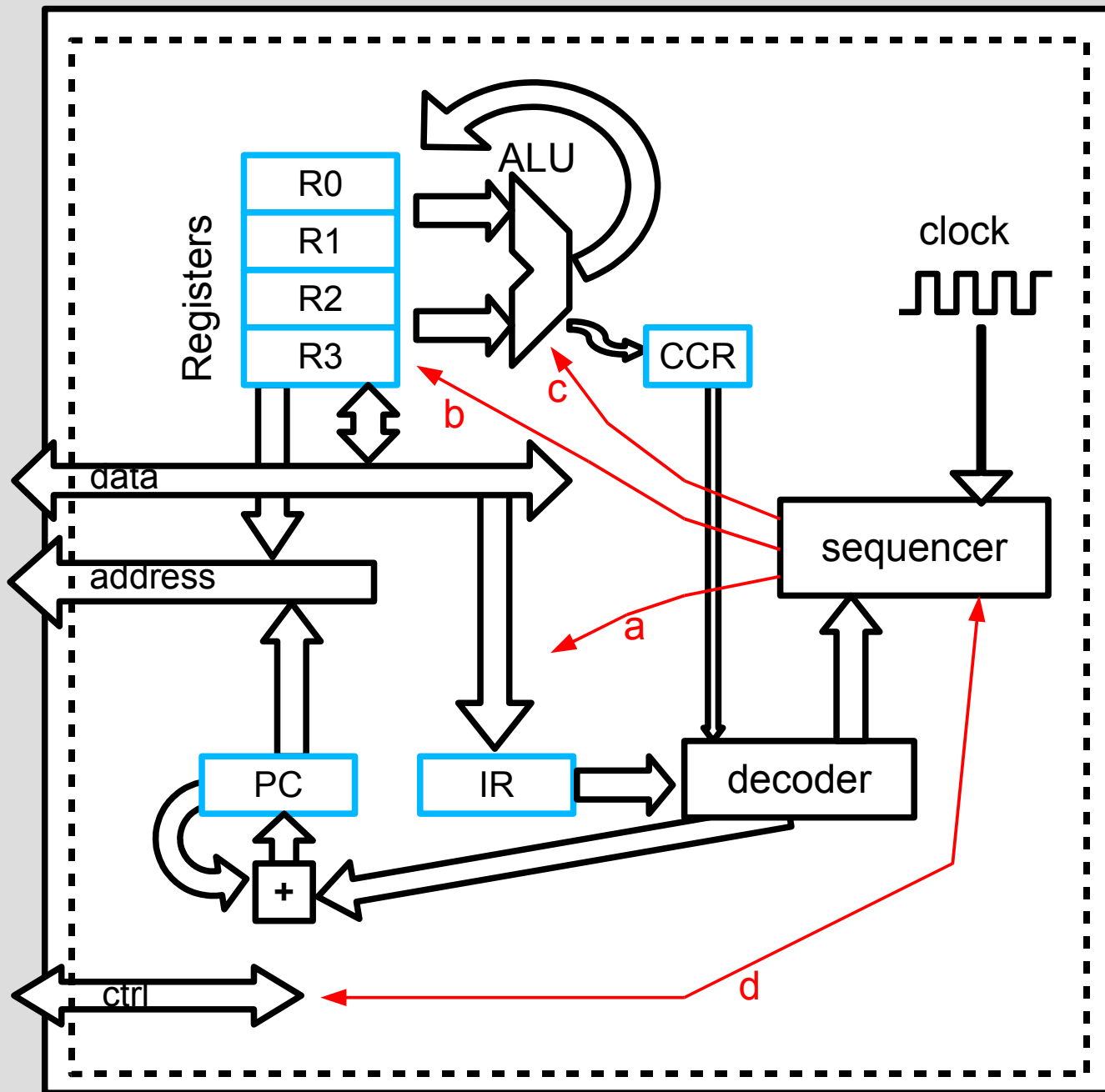
Organisation mémoires

- Pour que ce principe de **hiérarchie de mémoire** soit efficace il faut respecter
 - Localité temporelle : les programmes accèdent souvent aux mêmes données
 - Localité spatiale : les programmes accèdent à des données contiguës
- Ces conditions sont souvent respectées par les applications. Si ce n'est pas le cas, les performances se dégradent.

Architecture système

**Traitements en pipeline,
parallélisme et multicoeurs**

Processeur



Modèle générique de μ -processeur

PC = Program Counter
 IR = Instruction Register
 CCR = Condition Code Register

ALU = Arithmetic & Logic Unit

a : chargement instruction
 b : sélection opérandes
 c : sélection opération
 d : contrôle des accès mémoire et E/S

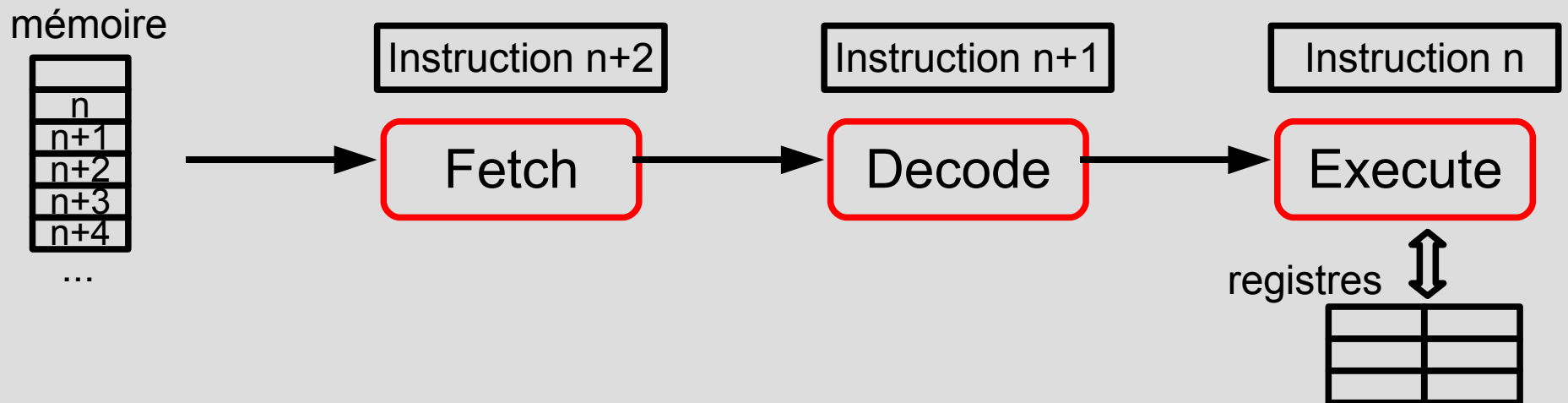
Architecture système

- L'exécution d'une instruction met en jeu plusieurs phases successives :
 - Chargement de l'instruction depuis la mémoire vers le registre d'instruction : **fetch**
 - Décodage de l'instruction pour préparer l'exécution : **decode**
 - L'exécution proprement dite : **execute**
- Chaque phase utilise des circuits différents...

Architecture système



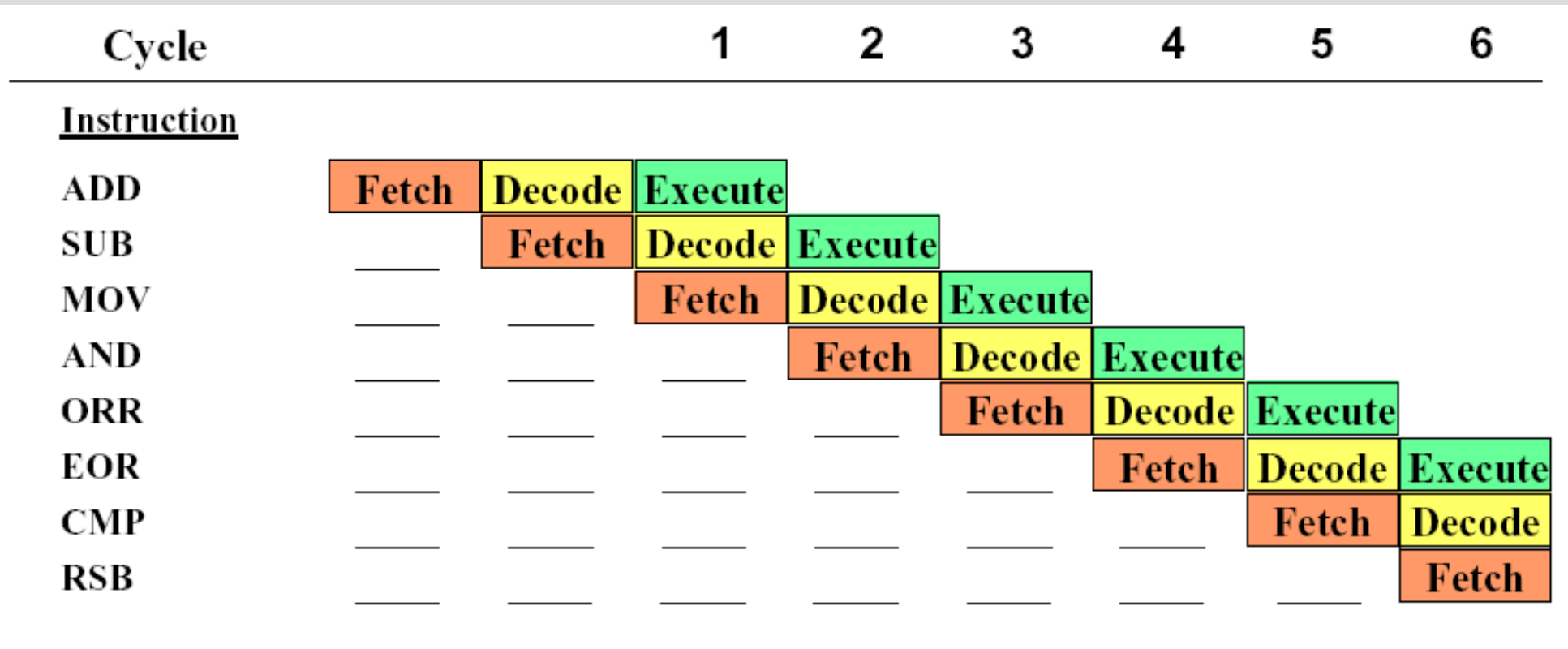
- Les circuits responsables d'une phase sont donc disponibles pour l'instruction suivante dès que l'instruction en cours atteint la phase suivante



- C'est le principe d'exécution en ***pipeline*** qui permet de traiter simultanément autant d'instruction que le nombre d'étapes ou **étages** du pipeline.

Architecture système

- Ainsi sur ARM7 il y a 3 cycles entre le début et la fin du traitement d'une instruction mais 6 cycles suffisent pour exécuter 6 instructions



Architecture système

- Ce principe est valable tant que les instructions se suivent de manière prévisible (pas de branchement) et tant que la phase d'exécution ne perturbe pas le fonctionnement des autres phases (pas d'accès mémoire)
- Dans le cas contraire le flux est interrompu ce qui diminue la performance moyenne... il faut relancer un nouveau train d'instructions

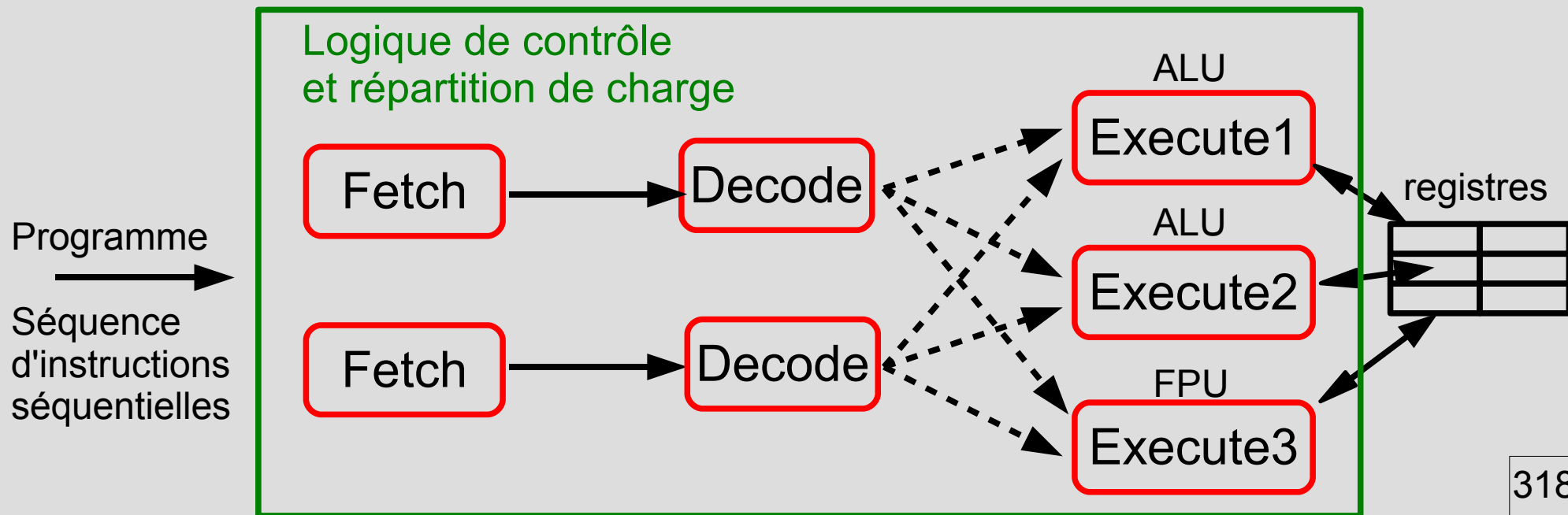
Architecture système

- Les processeurs évolués (Intel core ...) utilisent un nombre croissant d'étages au niveau du pipeline (entre 10 et 15 actuellement)
- Ceci favorise les sections de code dont le déroulement est prévisible
- L'utilisation de techniques de prévision dynamique réduit l'impact des branchements
- La présence de la mémoire cache L1 réduit les pénalités d'accès mémoire

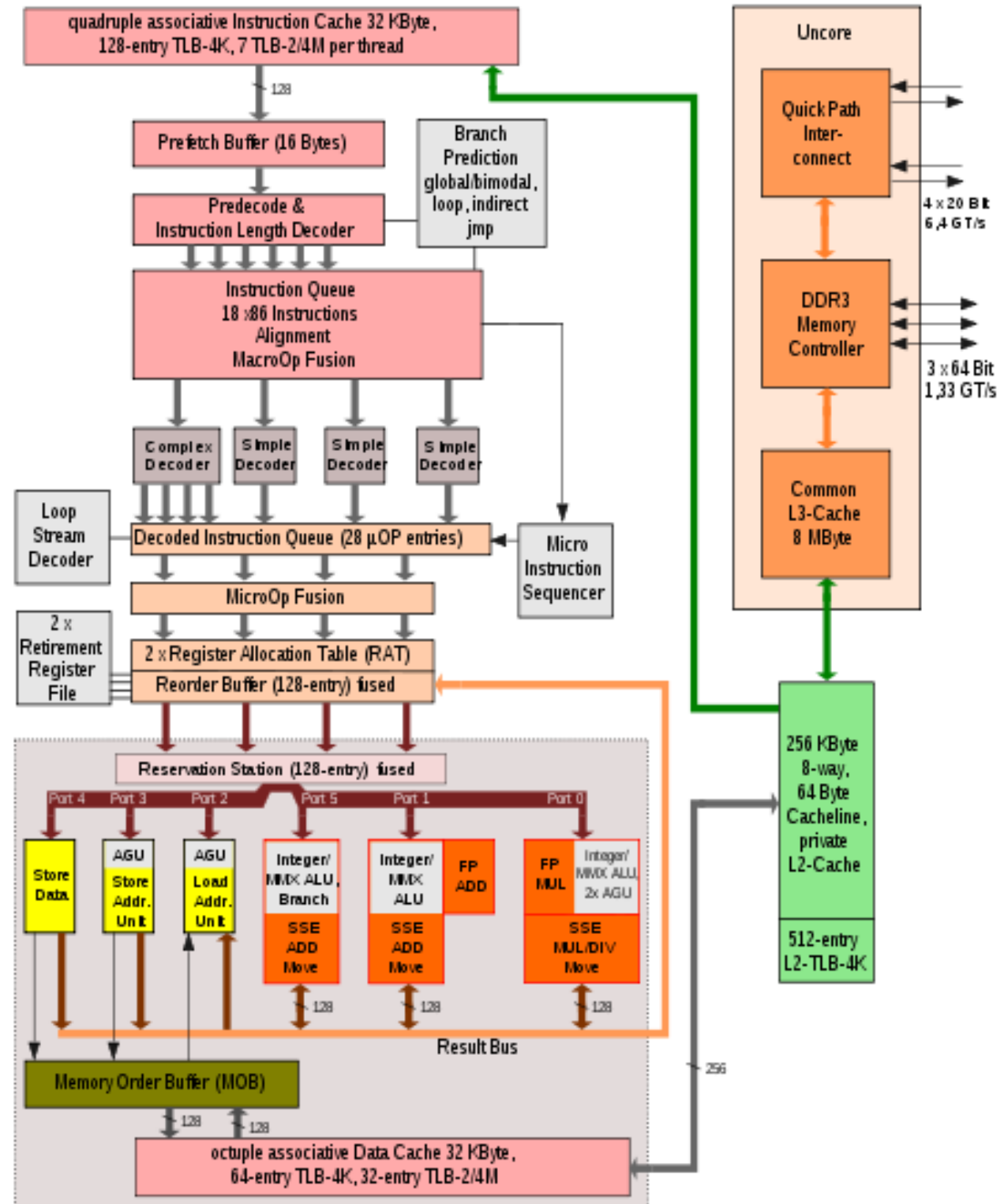
Architecture système



- Les processeurs évolués comportent **plusieurs unités d'exécution**, par exemple pour traiter des entiers et des flottants
- Il est possible d'exécuter **simultanément** plusieurs instruction à un même niveau pipeline
Ces architectures sont dites **superscalaires**



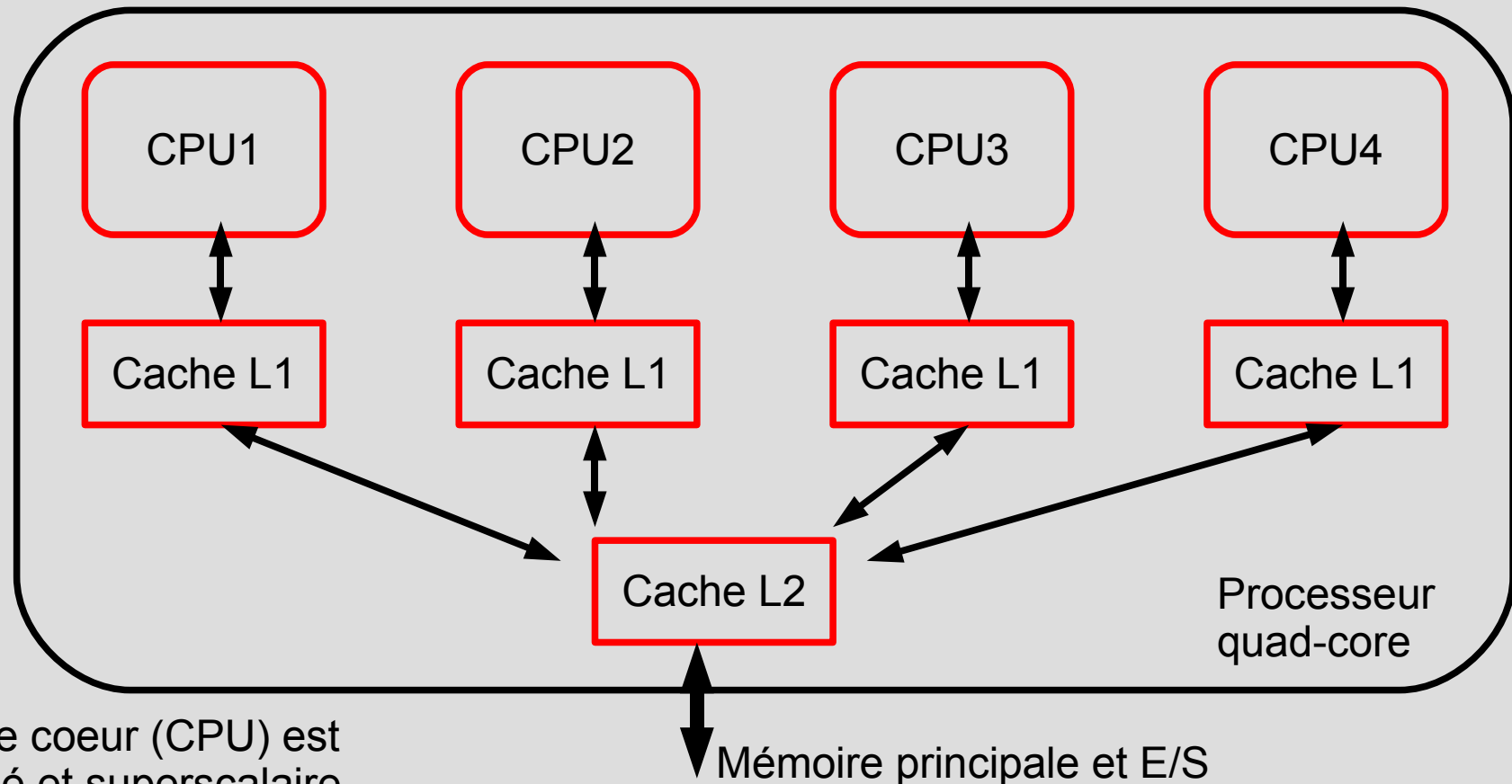
Exemple d'architecture pipelinée superscalaire Intel Nehalem



Architecture système



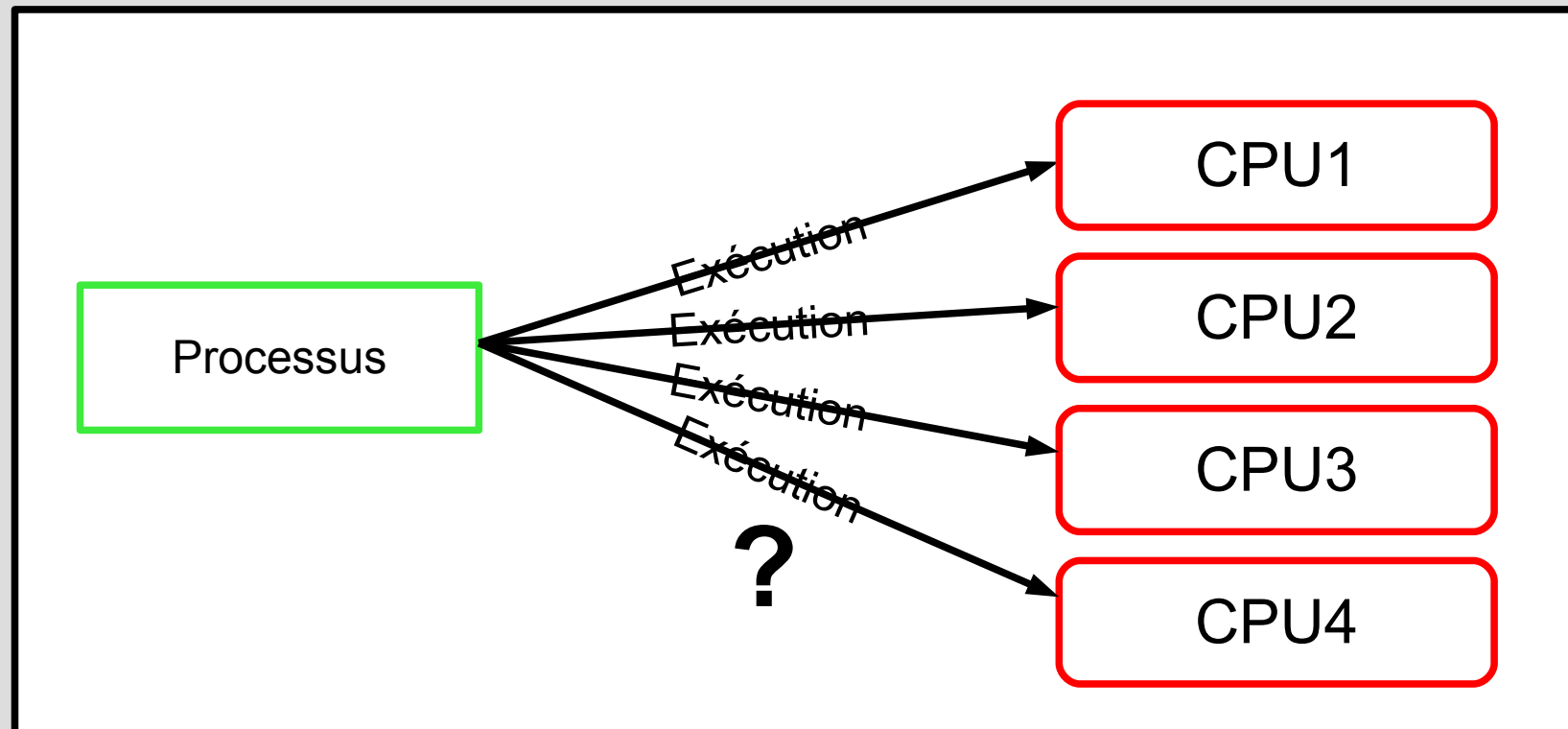
- L'évolution actuelle se fait vers des composants à plusieurs coeurs : chaque coeur est en fait équivalent à un processeur (CPU) complet incluant une mémoire cache L1



chaque coeur (CPU) est pipeliné et superscalaire

Architecture système

- L'exploitation optimale de ce type de processeur pose des problèmes particuliers : programmation du parallélisme



Architecture système

- Le développement se fait vers des processus multiples - ***multithread*** - pour exploiter ce parallélisme matériel

