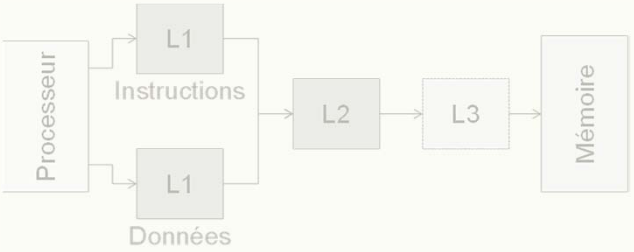
heig-vd
Haute Ecole d'Ingénierie
du Canton de Vaud

RéDS
heig-vd



ARO2

Mémoire cache et mémoire virtuelle

Basé sur le cours du prof. E. Sanchez
et le cours ASP du prof. M.Starkier

Romuald Mosqueron

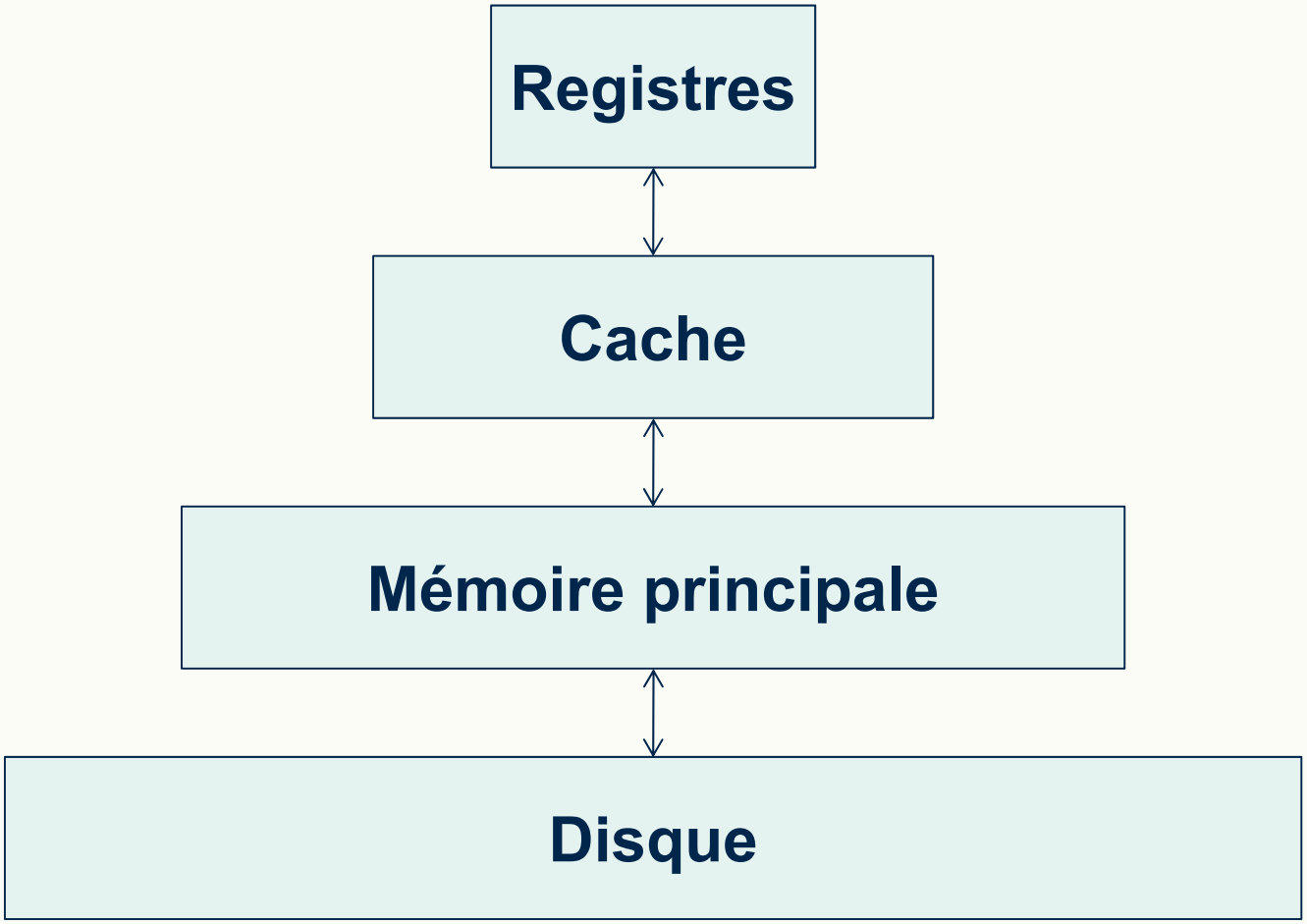
Hiérachie mémoire

32- 1024 B
0.2 – 2 ns

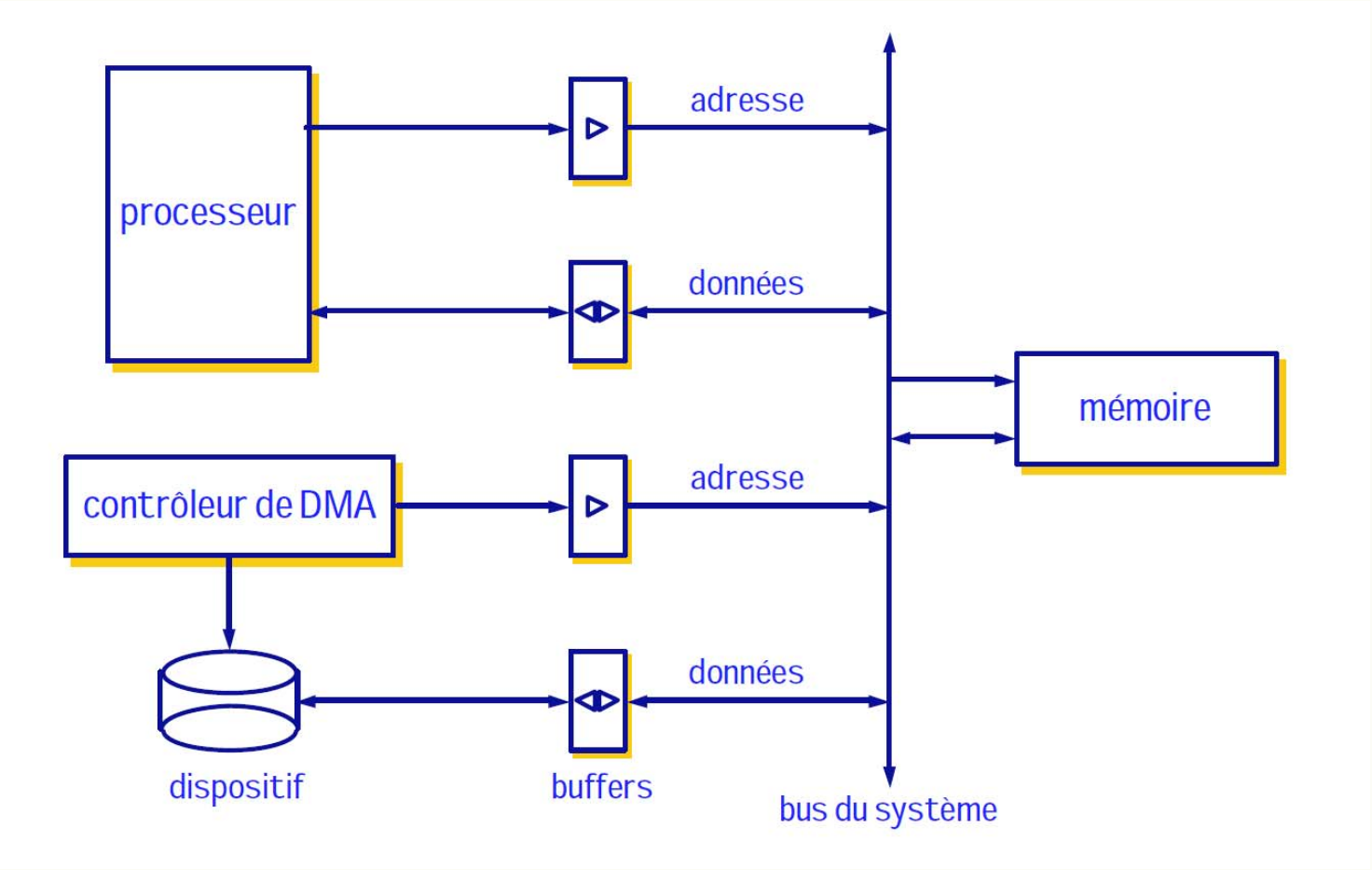
0.5 – 8MB
1-10ns

0.5 – 4 GB
10-100ns

0.5 – 2 TB
1-10ms



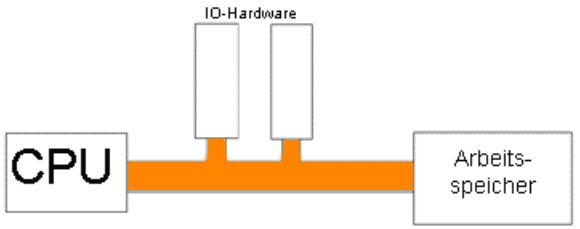
Organisation de la mémoire



DMA (Direct Memory Access)

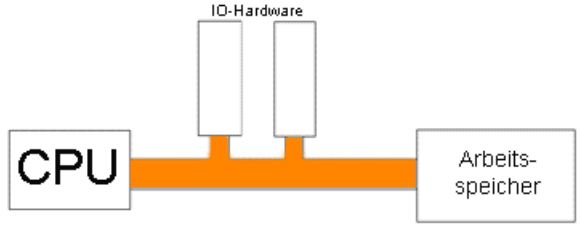
- **Définition:**

Transfert direct des données d'un périphérique sans passer par le processeur

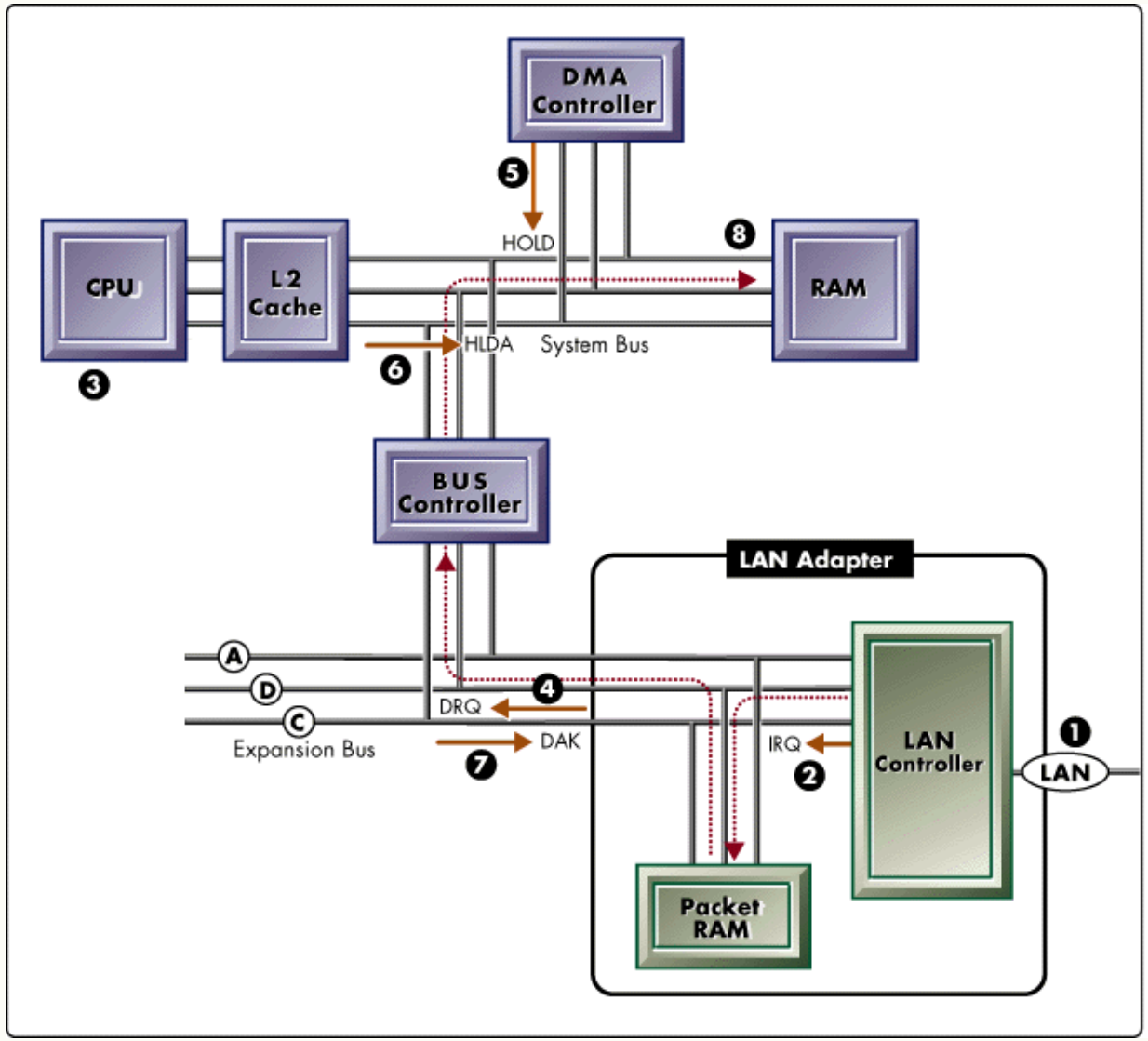


Pendant le transfert DMA :

1. Le bus est occupé par le transfert DMA
2. Le processeur exécute des instructions de la mémoire cache
3. Mais le processeur a en général priorité pour l'accès au bus

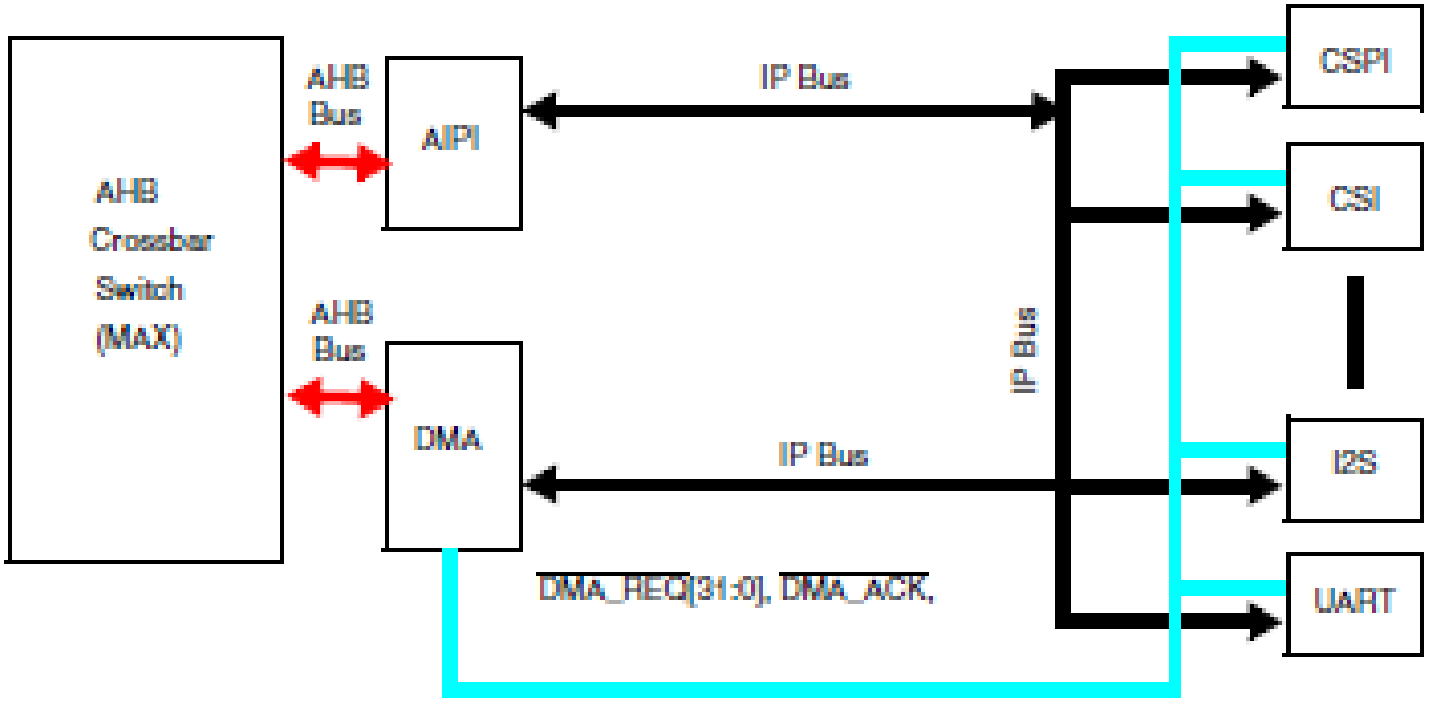


Exemple DMA


DMA Controller
5
HOLD
8
CPU
L2 Cache
RAM
HLDA
System Bus
3
6
BUS Controller
LAN Adapter
A
D
C
Expansion Bus
4
DRQ
DAK
7
IRQ
2
LAN Controller
1
LAN
Packet RAM

IMX21 Freescale

- DMA interne de microcontrôleur



Contrôleur DMA

- Partie intégrante du microcontrôleur, ou du chipset, parfois du périphérique
- Prend le contrôle du bus avec arbitration
- Instructions de transfert :
  1. Adresse source
  2. Adresse de destination
  3. Nombre d'octets à transférer
- Mécanisme de listes chaînées :
  - Suites d'instructions de transfert

Cours ARO2

MÉMOIRES CACHES

Mémoire cache (par l'exemple)

- **Exemple : CPU à 2,6GHz et mémoire avec 10ns de latence (temps de calcul de l'adresse avant de transmettre le contenu)**
  - **Si le CPU veut accéder à une information mémoire, après combien de temps va-t-il recevoir l'information ?**

    => 10×10^{-9} s
  - **Combien de cycles CPU pendant cette durée ?**

    => $10 \times 10^{-9} \times 2{,}6 \times 10^{9} = 26$ cycles

Mémoire cache

- Conclusion:
  - Même si le CPU peut faire n'importe quelle instruction (add, mult…) en quelques cycles, il doit attendre 26 cycles pour avoir les opérandes
  - Trop long pour le CPU d'aller chercher l'information en mémoire ; mais c'est pourtant là qu'elle est !

- Solution:
  - Augmenter la capacité de mémoire dite rapide

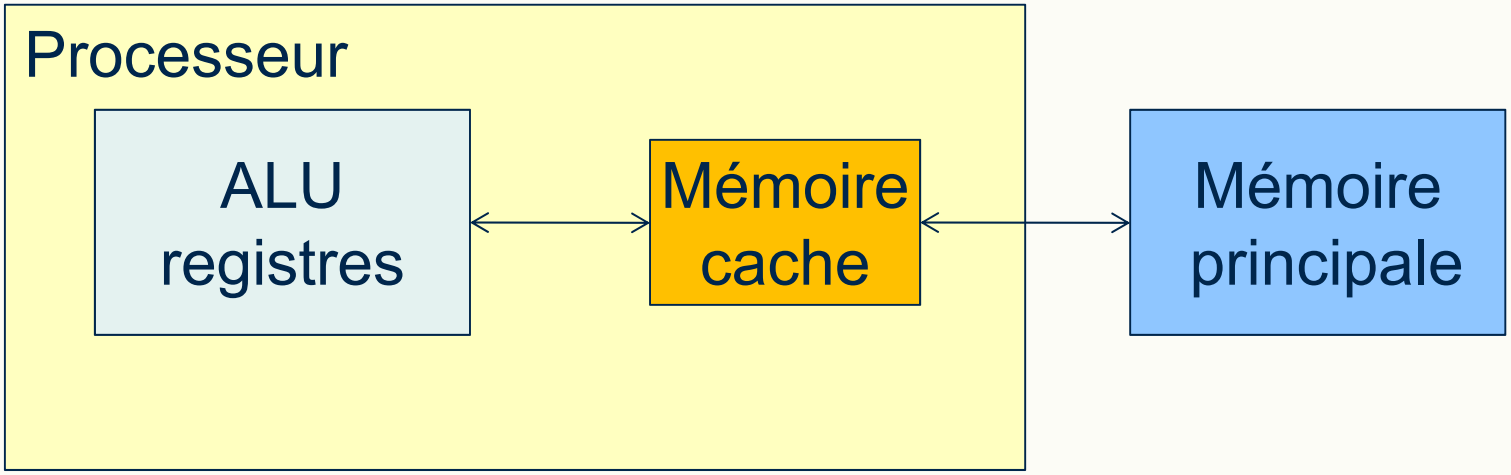
  La mémoire rapide est chère, donc trop coûteux d'avoir toute la RAM très rapide

  => on va se contenter d'une petite partie du contenu

Mémoire cache

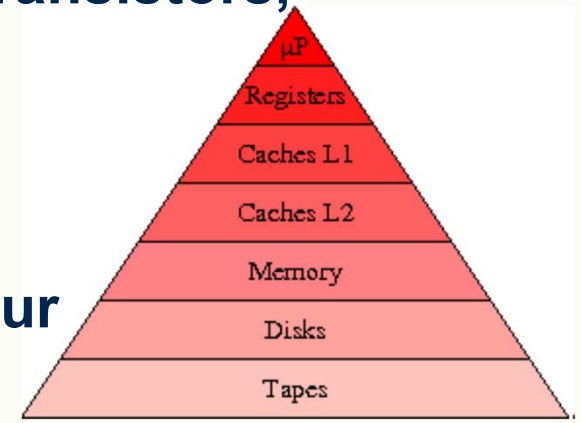
Définition: Mémoire cache (antémémoire)

- **Mémoire petite et rapide**
- **Stocke une copie des informations en mémoire RAM du processeur**
- **Située entre les unités de traitement du processeur et la mémoire principale**



Technologie de la mémoire cache

- La mémoire cache est composée de transistors, et non de condensateurs
  - Même fréquence que le CPU
  - Latence de quelques cycles CPU
  - Taille très réduite : 32Ko-128Ko pour les mémoire L1
  - Aucun contrôle sur le contenu du cache
    - Se remplit au fur et à mesure de l'exécution des programmes



- Cache pour les instructions et cache pour les données
- Souvent plusieurs niveaux de cache : Lk plus rapide mais avec une taille plus petite que Lk+1

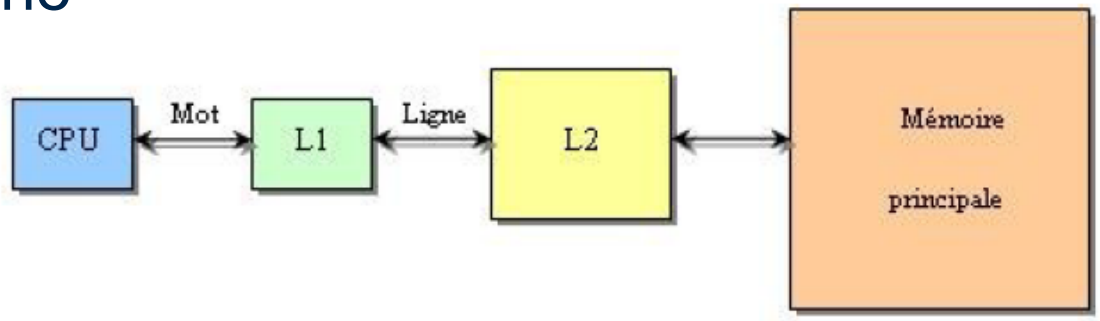
Principe de localité (locality)

- **Deux principes fondamentaux en informatique:**
  - **Localité spatiale**: le code d'un programme s'exécute toujours à l'intérieur de petites zones répétées de mémoire (des blocs correspondant à des boucles ou/et des sous-programmes). Si une adresse mémoire est utilisée, alors les adresses proches le seront probablement dans un futur proche.
  - **Localité temporelle**: les blocs s'exécutent en séquences très proches (il y a plus de chances d'accéder à une position de mémoire utilisée il y a 10 cycles qu'à une autre utilisée il y a 10000 cycles). Si une adresse mémoire est utilisée, elle le sera probablement à nouveau dans un futur proche

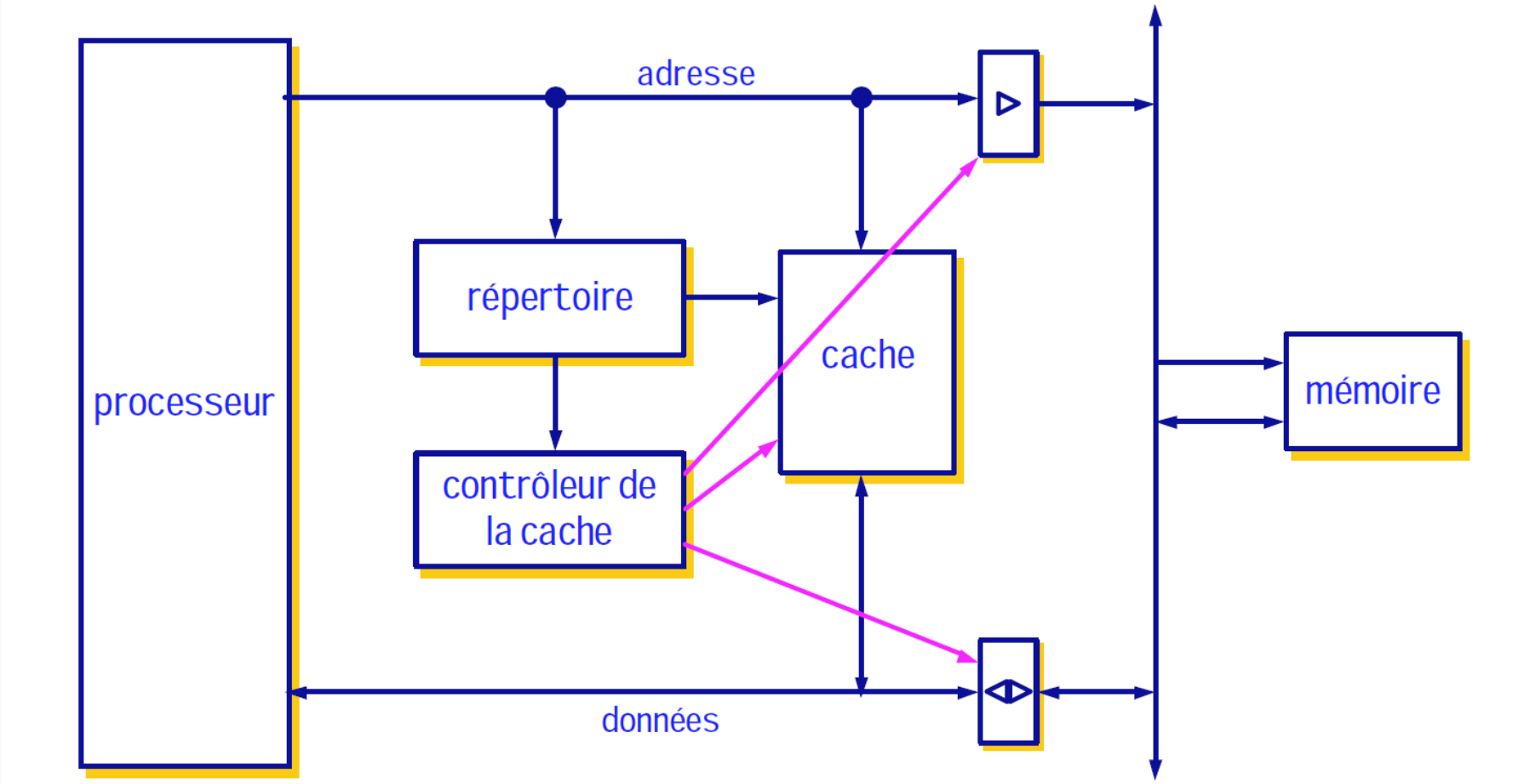
=> Ces deux principes sont à la base du principe des caches

Principe de la mémoire cache

- Le processeur essaie d'accéder à un mot d'abord dans la cache, avant de passer à la mémoire principale.
  - En cas d'échec (miss), le mot est gardé dans la cache pour un accès futur.
  - En cas de succès (hit), la mémoire principale n'est pas accédée
- La fréquence des succès (hit rate) dépend de la taille de la cache et de l'algorithme exécuté par le contrôleur de cache



Principe de la mémoire cache



Terminologie

- **2 cas :**

1. **La donnée est dans le cache à la lecture**
   => cache hit

   ***Hit rate* : pourcentage d'accès avec hit (nb hit / nb accès)**

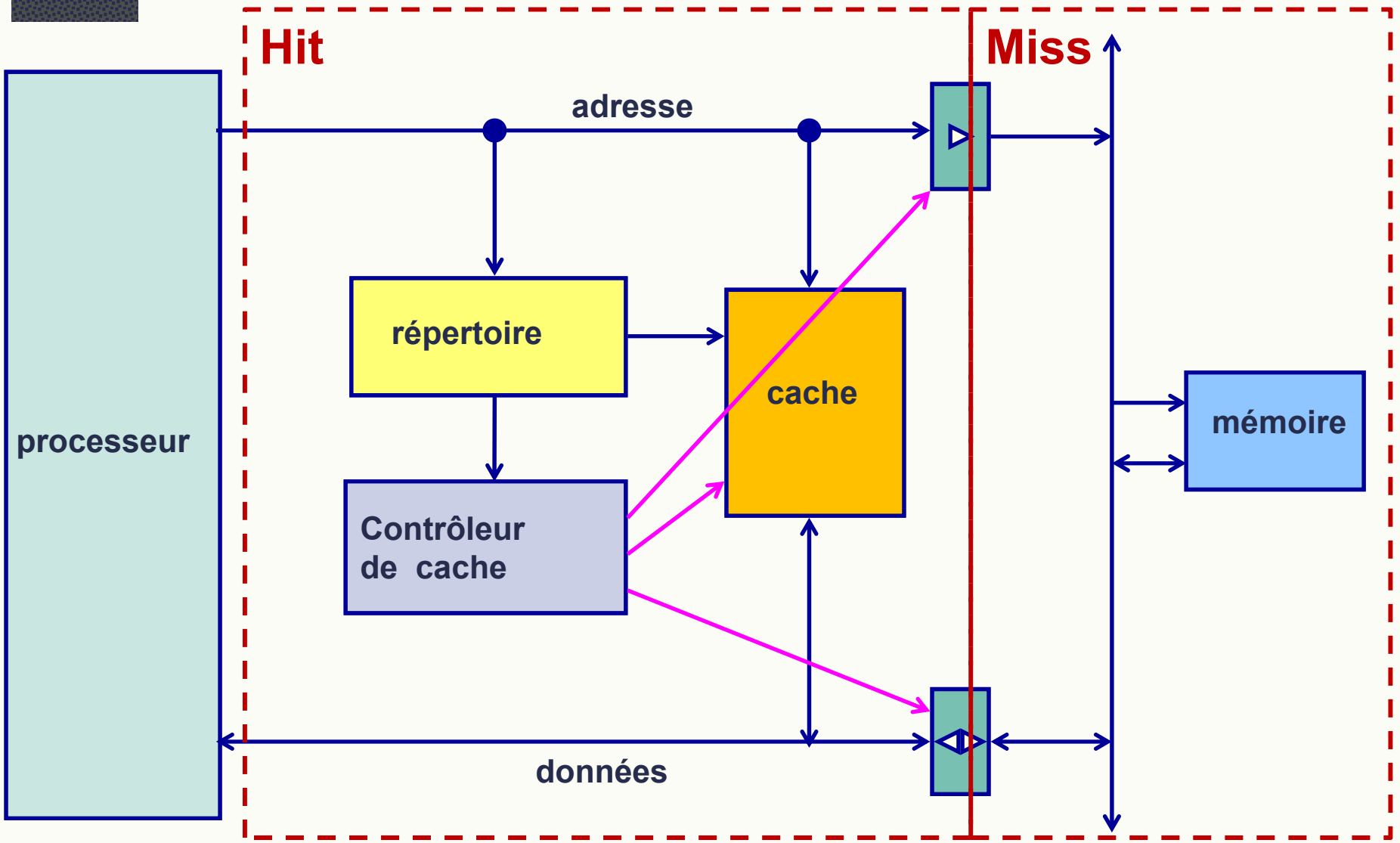
   ***Hit time* : temps d'accès à la cache**

2. **La donnée n'est pas dans le cache**
   => cache miss

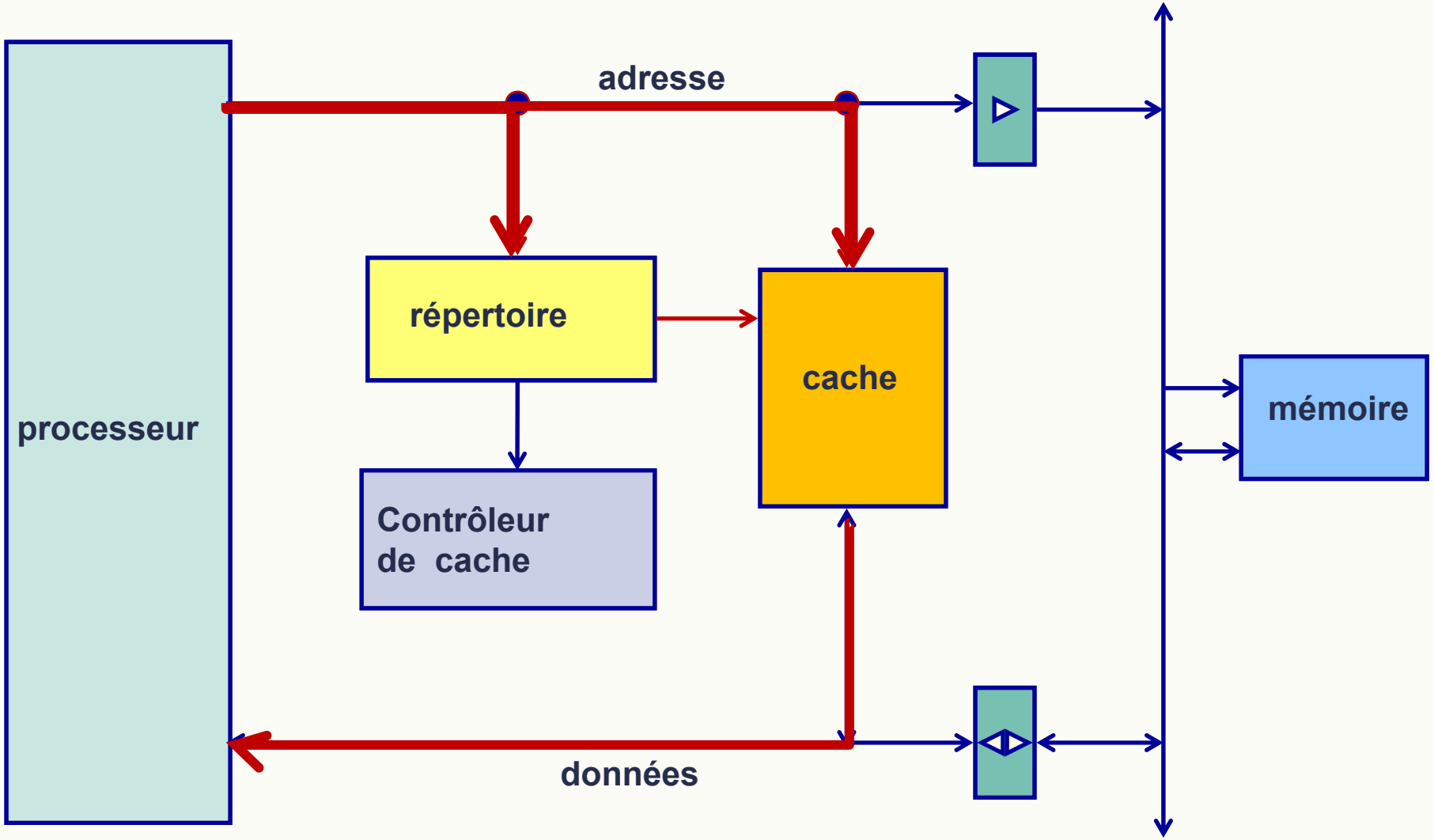
   ***Miss rate* = (1 – Hit rate)**

   ***Miss penalty* : temps d'accès mémoire principale + temps de chargement cache => Miss penalty >> Hit time**

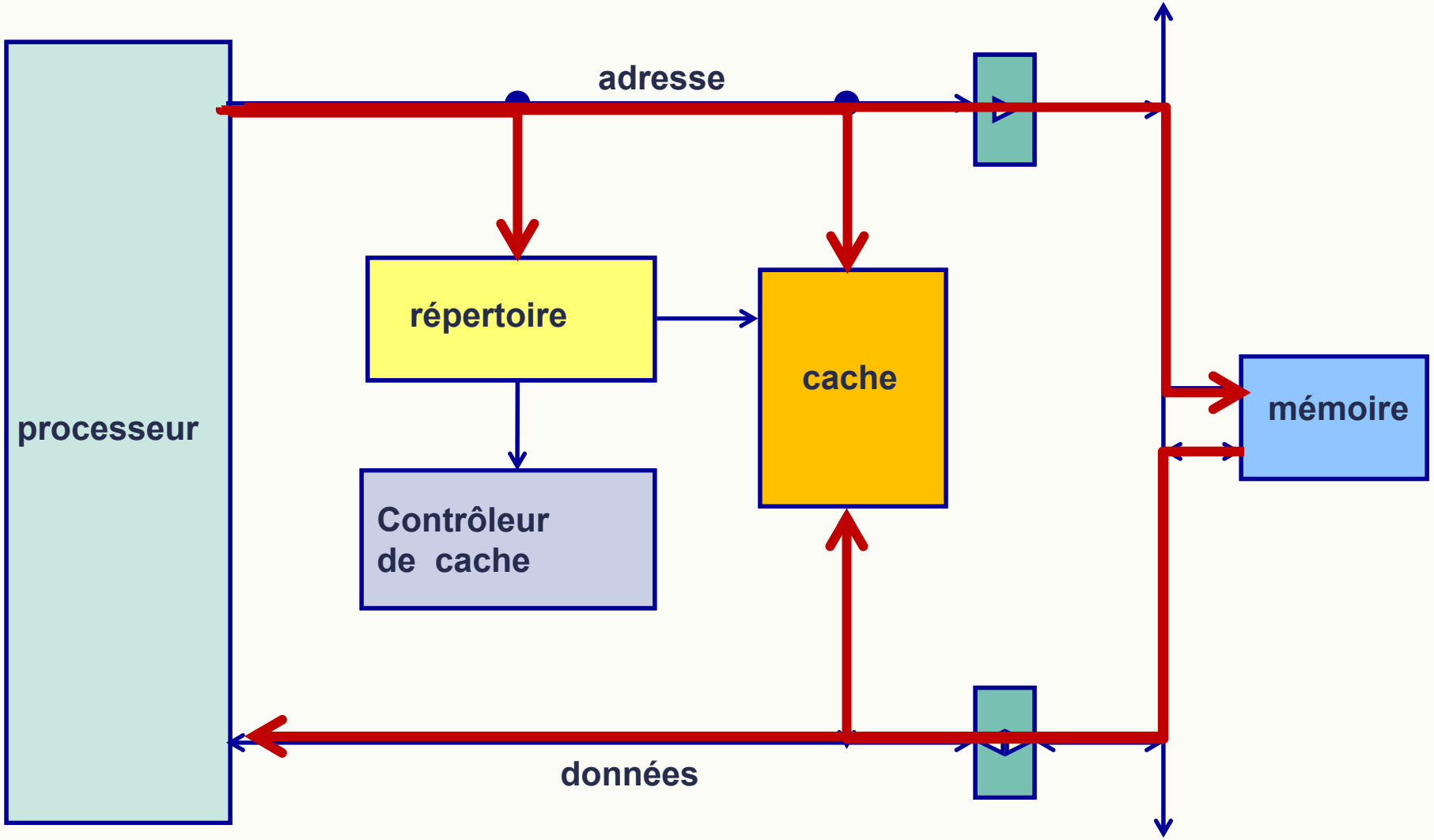
Système mémoire cache



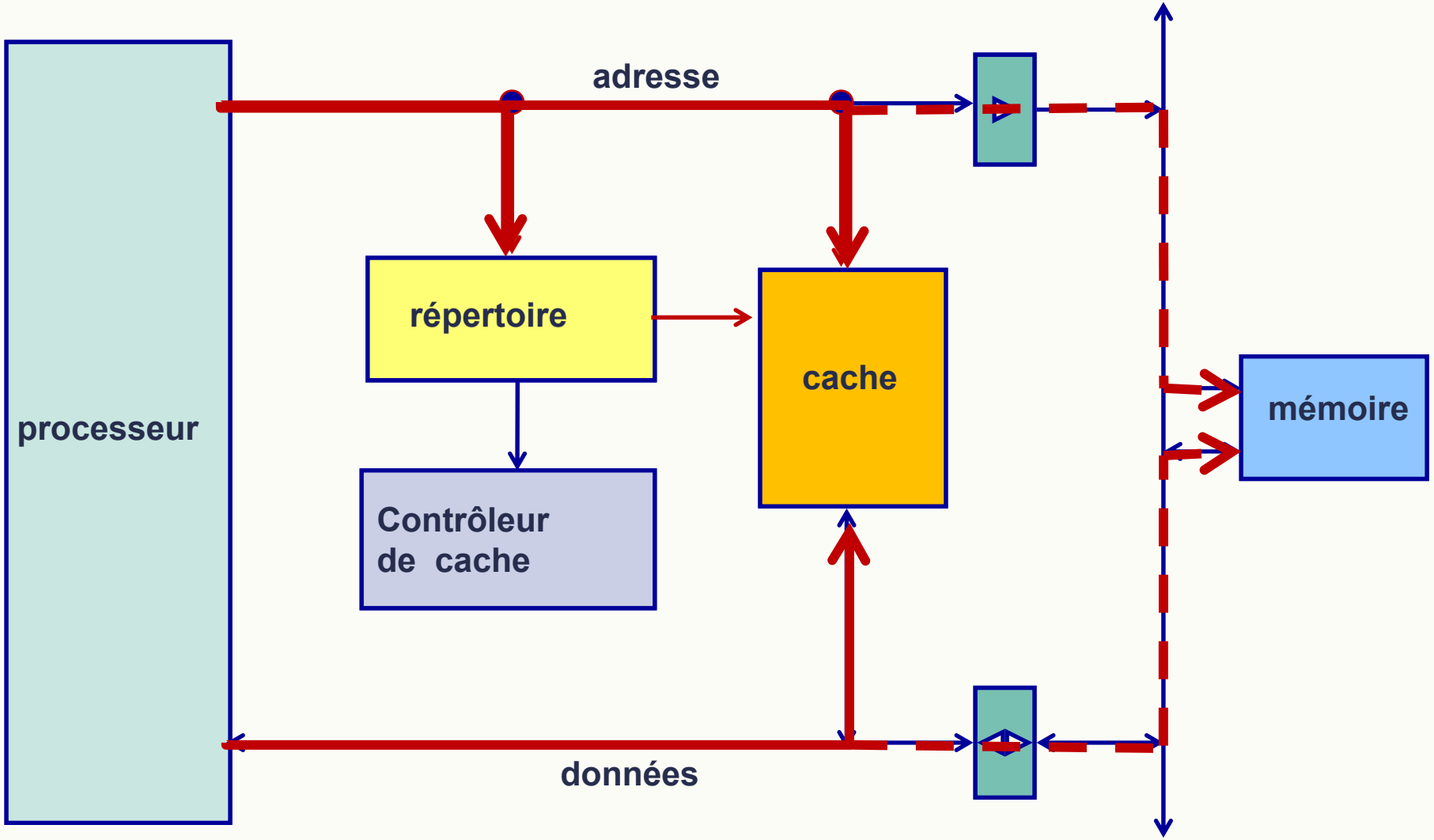
Hit (lecture)


adresse
processeur
répertoire
cache
mémoire
Contrôleur
de cache
données

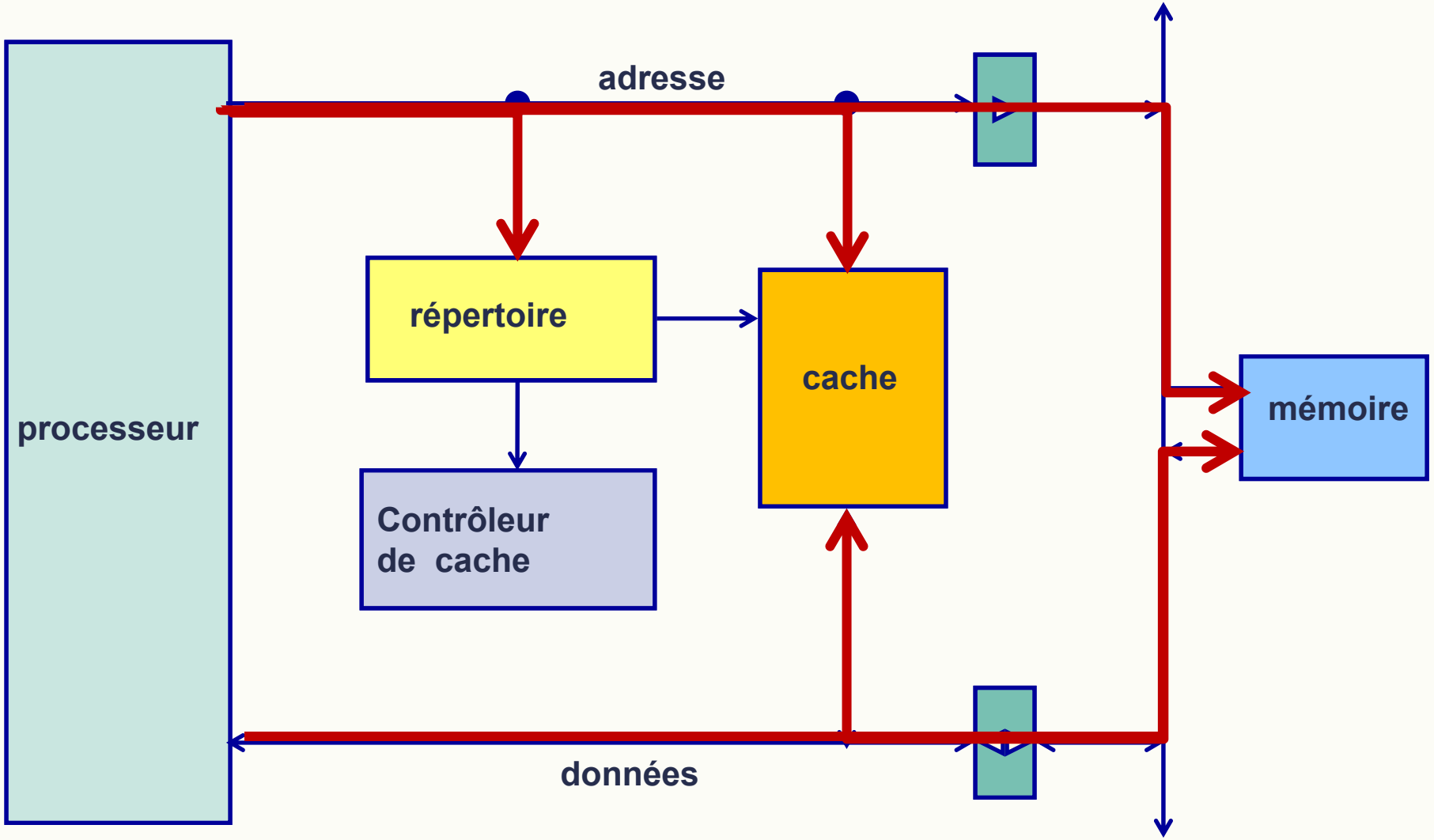
Miss (lecture)



Hit (écriture)



Miss (écriture)



Exemple : Hit rate / Miss rate

- Hit time = 2 cycles (d'horloge)
- Miss penalty = 5 cycles
- Hit rate = 80 %

- Nb cycles moyen par donnée
  = 0.8 x 2 + 0.2 x 5 = 2.6 (1.6 hit, 1 miss)
- Soit 40% (1/2.6) du temps en miss pour 20% des données

Importance de la fréquence de succès

- Si les accès à la mémoire demandent 3 wait states et la fréquence de succès est de 90%, le nombre moyen de waits states est de 10% x 3 = 0.3 wait state / cycle de mémoire

La chute de la fréquence de succès à 80% fait doubler cette valeur

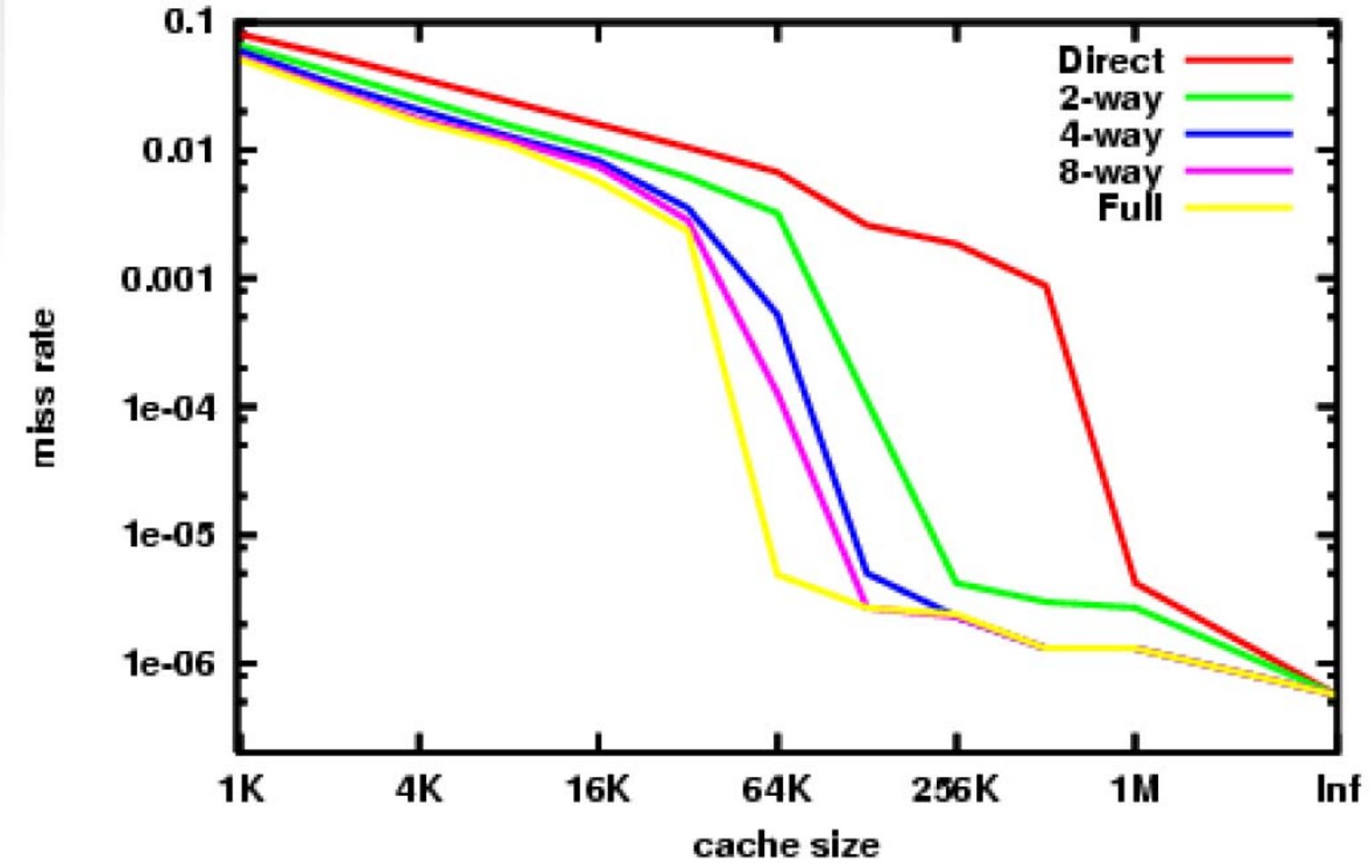
- Si une instruction demande 2 cycles sans wait state et 5 cycles avec, et si la fréquence de succès est de 80%, l'exécution de 10 instructions demande:

  (10 x 0.8 x 2) + (10 x (1 - 0.8) x 5) = 26 cycles

  dont 16 sont faits avec la cache.

C'est-à-dire: le processeur passe 40% du temps (10/26) à chercher 20% du code: pendant ce temps le bus du système est occupé par les transferts entre le processeur et la mémoire

Influence de la taille du cache et des politiques de placement sur le miss rate



Ligne de données

- Une ligne (ou bloc) de cache est la plus petite portion (groupe d'octets 8 à 64) de la cache avec une étiquette (tag) unique
  - Un cache de 64Ko aura donc 1024 lignes
- Les transferts entre cache et mémoire principale se font par lignes
- Intérêt d'avoir des lignes de plusieurs octets ?
  - RAM de répertoire plus petite que la RAM de cache
  - Transferts multi-mots (burst) plus rapides
- Pour réduire le temps de mise à jour lors d'un échec, le bus entre la mémoire et la cache est généralement plus large que celui entre la cache et le processeur

Fonctionnement de la mémoire cache

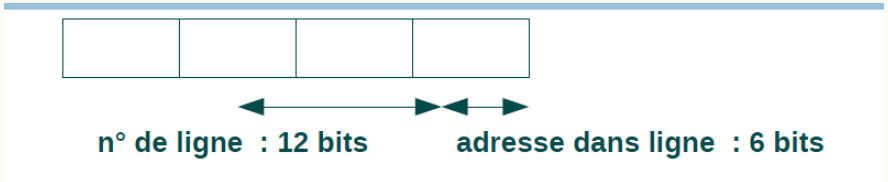
- Quand le CPU a besoin d'un octet en mémoire
  - Le CPU regarde dans le cache
    - Doit connaitre connaisse les adresses des octets
  - Si pas dans le cache, lecture en mémoire et insertion dans le cache (au besoin en écrasant des octets présents)

- Comme le cache est géré au niveau des lignes, les octets d'adresse 0 à 63 seront dans la même ligne, ceux de 64 à 127 dans une autre, etc.

=> Revient à découper la mémoire en blocs de la taille d'une ligne de cache : transferts mémoire ↔ cache par bloc

Stratégies de placement

- Si le processeur demande à lire l'adresse mémoire X
  - Où placer l'information contenue en X (et celles des octets du bloc de X) dans le cache ?
  - Le cache doit mémoriser l'adresse de base de la ligne : pour des lignes de 2n octets, adresse de base b = X / 2n
  - par exemple



- Stratégies de placement (cablées, non logicielles)
  - Sur une ligne dépendante de l'adresse : direct mapped
  - N'importe où : cache fully associative
  - Dans un sous ensemble de lignes : set associative

Le répertoire de la cache

- Le répertoire de la cache a deux fonctions:
  - indiquer si le mot accédé par le processeur se trouve dans la cache (signal match)
  - si c'est le cas, indiquer l'adresse de la cache où se trouve le mot

- Le répertoire de la cache peut donc être vu comme une mémoire associative: on parle alors d'une mémoire cache complètement associative.
- Dans ce cas, un mot de la mémoire principale peut être stocké n'importe où dans la cache

Recherche dans le cache

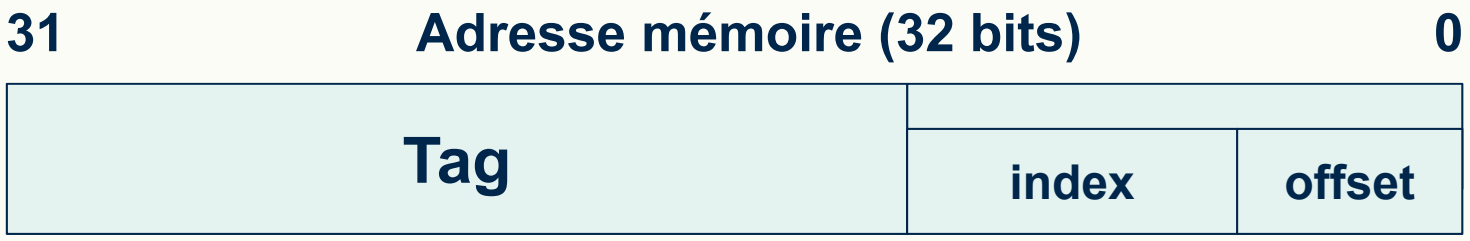
On regarde dans le cache très souvent pour savoir si une ligne est présente

- L'opération doit être rapide

- **Fully Associative**
  - Il faut regarder toutes les lignes : trop lent séquentiellement
  - Recherche en parallèle, très coûteux financièrement car circuits compliqués
- **Direct Mapped**
  - Une seule ligne à regarder
- **Set Associative**
  - Un seul set à regarder, mais recherche séquentielle ou en parallèle (moins coûteux car moins de lignes) à l'intérieur

Mémoire cache associative par ensemble (direct mapped cache)

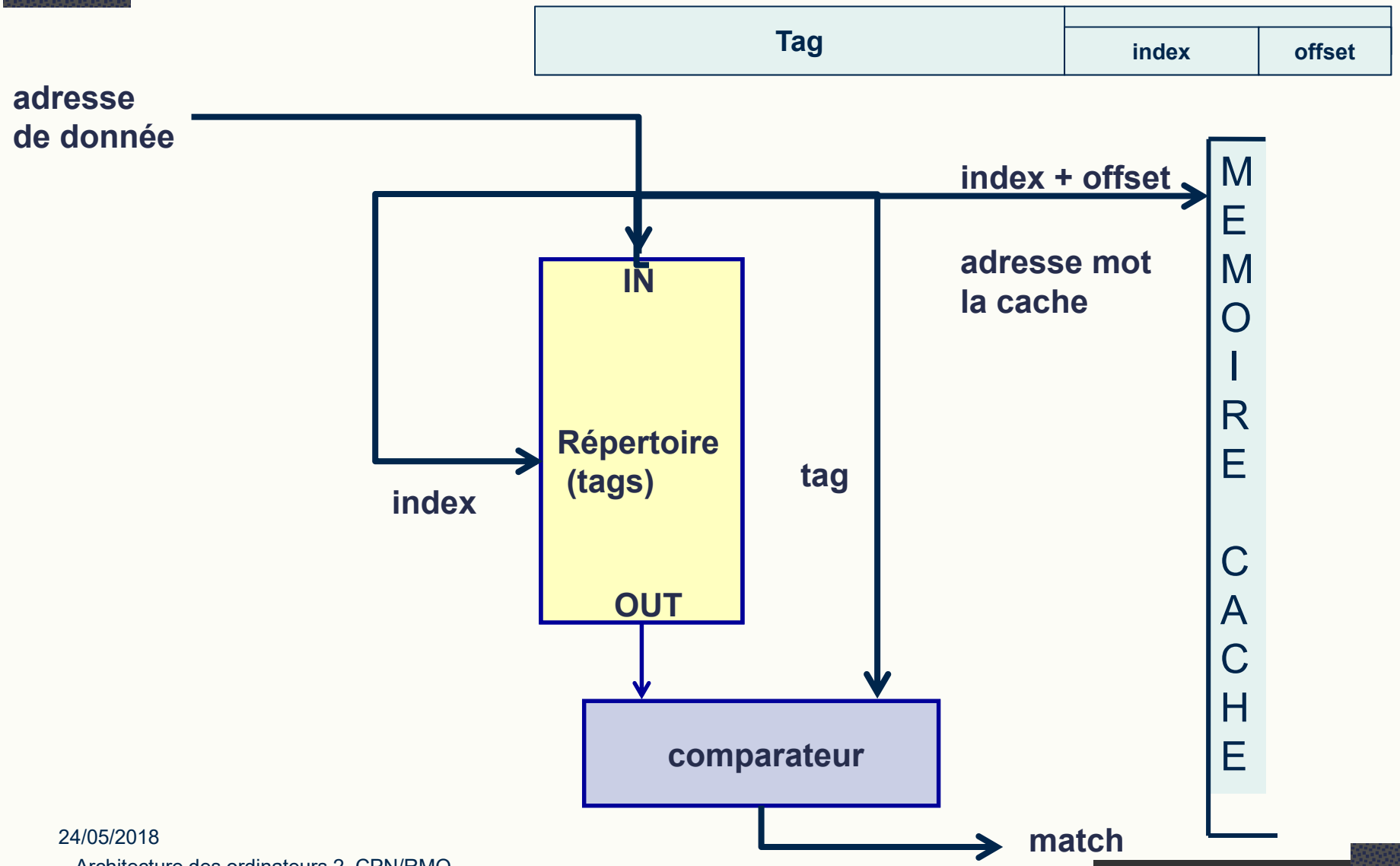
- La cache est adressée par les bits de poids faible de l'adresse
- Le répertoire
  - contient un tag pour chaque adresse du cache
  - le tag contient les bits de poids fort de l'adresse et un bit de validité



Mémoire cache associative par ensemble (direct mapped cache)

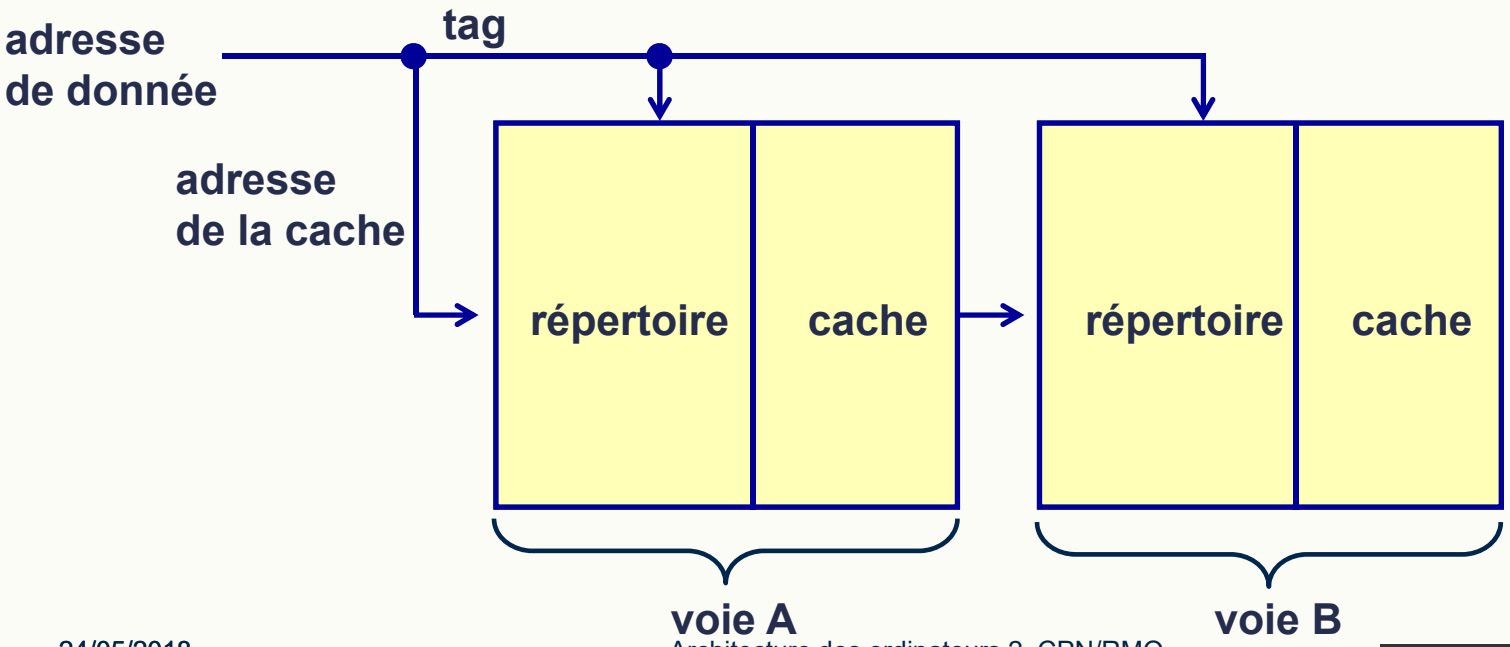
- La mémoire principale est divisée en ensembles et la cache est complètement associative pour chaque ensemble
- L'adresse physique est divisée en deux parties:
  - les bits de poids faible contiennent l'adresse de l'ensemble (adresse de la cache): toutes les adresses physiques qui partagent ces bits seront stockées à la même position dans la cache
  - les bits de poids fort forment une étiquette (tag), à comparer avec la valeur stockée dans le répertoire pour l'ensemble sélectionné
- Plus rapide que la cache complètement associative: la cache est accédée sans attendre la réponse du répertoire
- Au moment du démarrage, le contenu de la cache est quelconque. Chaque ligne de la cache contient un bit de validité (valid bit): il est mis à 1 si le contenu de la ligne est valable

Mémoire cache associative par ensemble (direct mapped cache)



Mémoire cache associative à plusieurs voies (N-way set associative cache)

- Constitué de plusieurs caches par ensemble couplées
- Permet de stocker plusieurs mots ayant les mêmes LSB d'adresse (index)



Mémoire cache associative à plusieurs voies (N-way set associative cache)

- Dans une cache associative par ensembles, un mot de la mémoire principale peut être stocké à une seule position dans la cache. Ce type de cache est appelé également direct-mapped. Si le processeur accède à deux mots dont les adresses partagent les bits de poids faible (l'adresse de l'ensemble), un échec arrive
- Ce problème est résolu dans une cache associative par ensembles à N voies: un mot de la mémoire principale peut être stocké en N positions différentes de la cache
- En général:
  - doubler l'associativité implique une diminution de 20% de la fréquence d'échec
  - doubler la taille de la cache implique une diminution de 69% de la fréquence d'échec

Algorithmes d'écriture

- Write-through: la mémoire principale est toujours modifiée lors d'un cycle d'écriture, qu'il s'agisse d'un succès ou d'un échec
- Copy-back ou write-back: l'écriture se fait seulement dans la cache. Pour garder la cohérence avec la mémoire principale, on peut écrire dans la mémoire principale toute ligne qu'on enlève de la cache avec une information valable. Pour éviter des écritures inutiles, on ajoute un bit par ligne, le dirty bit, pour indiquer que le contenu de la cache est différent de celui de la mémoire
- Dans tous les cas, on peut améliorer le temps d'écriture en ajoutant un buffer d'écriture (write buffer): les données sont écrites dans ce buffer et le contrôleur de cache se charge de les envoyer par la suite dans la mémoire

Cache miss, politique de remplacement

- Si info non présente dans le cache, on doit l'y mettre
  - Problème si cache plein ou ligne pleine : il faut choisir une ligne à remplacer
    - Si direct mapped pas le choix
    - Si fully associative trop de choix
    - Si set associative un peu de choix
- Politiques de remplacement (stratégies associatives)
  - Aléatoire : on choisit une ligne au hasard parmi les candidates
  - FIFO : premier entré – premier sorti
  - Least Recently Used : on supprime une ligne qui n'a pas été utilisée depuis longtemps
    - Il faut se souvenir de cette information
- Influence le miss rate (nb pages non présentes / nb pages cherchées)

Algorithme d'effacement

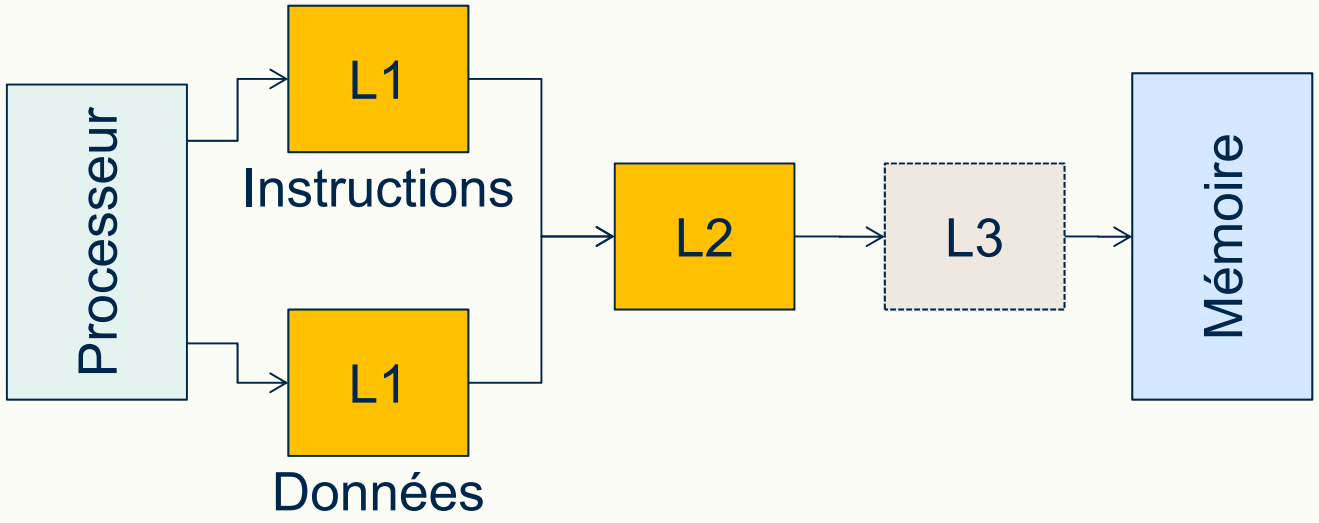
- Important : choix de la ligne à effacer quand une nouvelle ligne doit être écrite

- 3 méthodes (exemples) :
- Méthode LRU (least recently used): chaque ligne de la cache (même ensemble mais voies différentes) possède des bits pour indiquer l'ordre d'utilisation des voies.

  Pour une cache à 4 voies il y a 24 possibilités (4!): il faudrait 5 bits de codage au minimum. Pour une cache à 16 voies il faudrait 45 bits...
- Méthode NLU (not last used): on n'écrit pas dans la dernière voie accédée. Une RAM stocke cette information pour chaque ligne
- Remplacement aléatoire

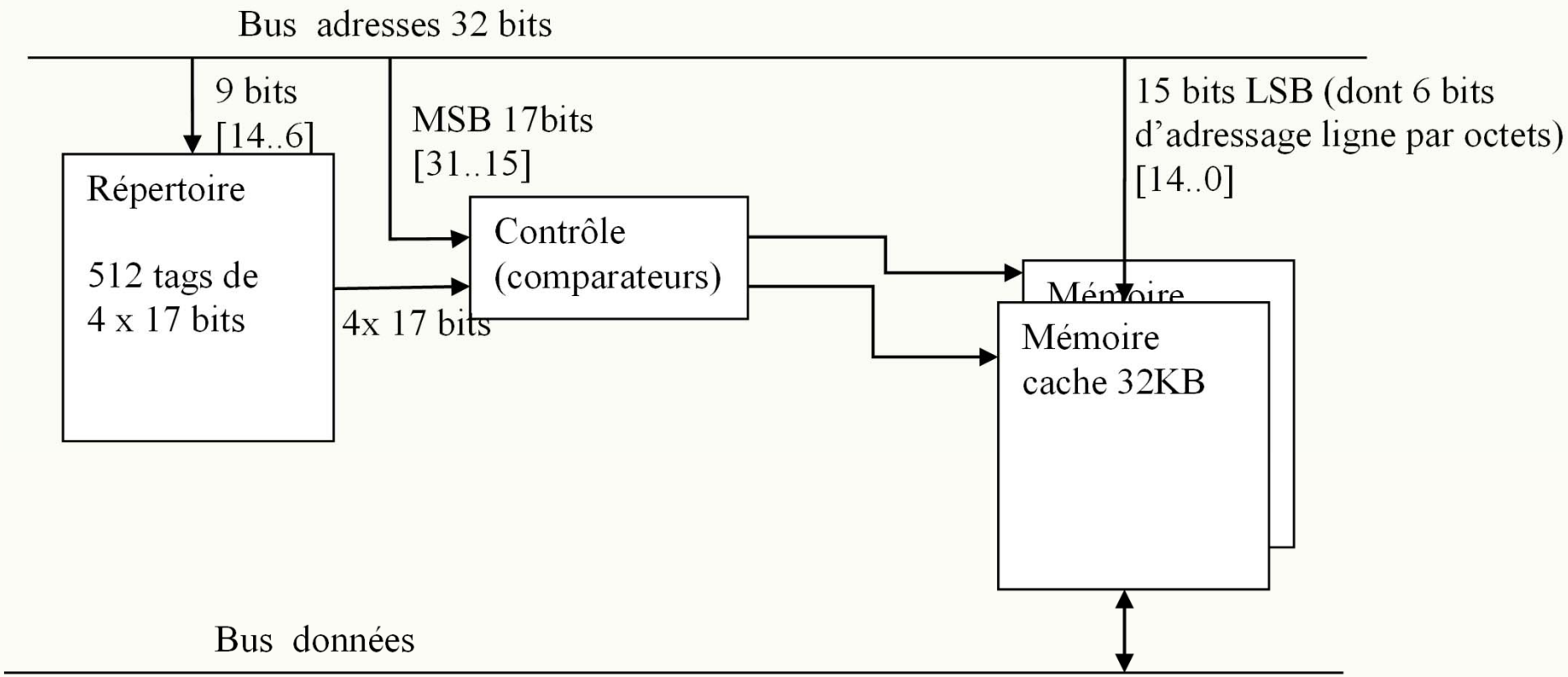
Caches multi-niveaux

- Niveaux 1, 2 et 3 => L1, L2, L3
- Exemple :
  - Opteron 2xL1 64kB, 1 L2 1MB



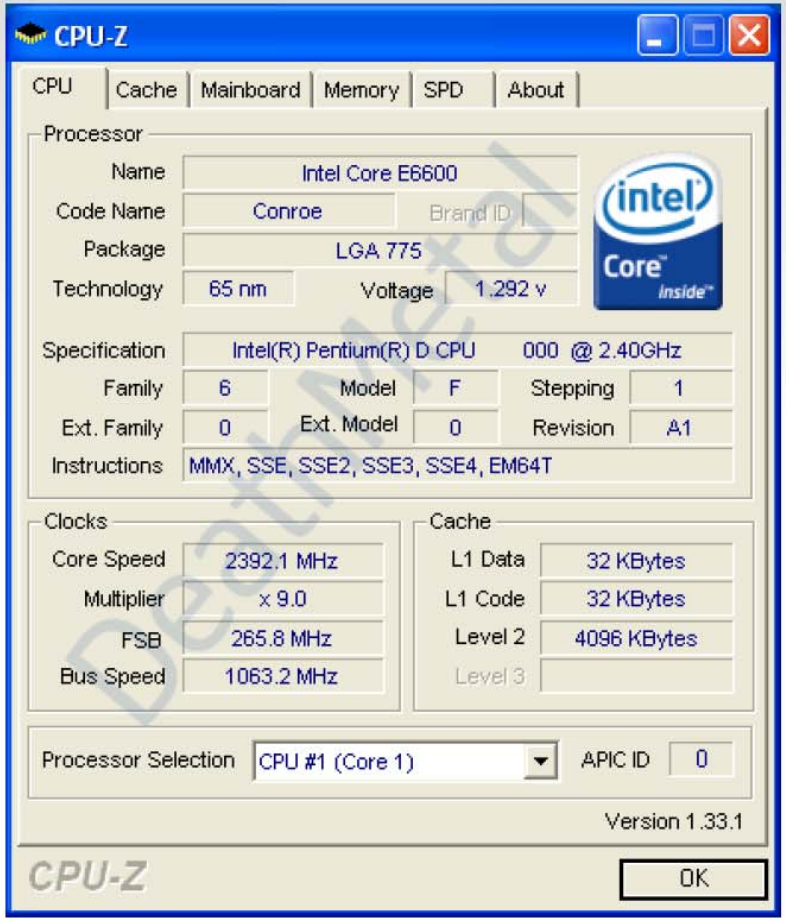
Mémoires caches niveau 1

- **128KB, 4 voies associative de 32KB**
- **taille de ligne : 64 octets**



Exemple : Core 2 Duo E6600

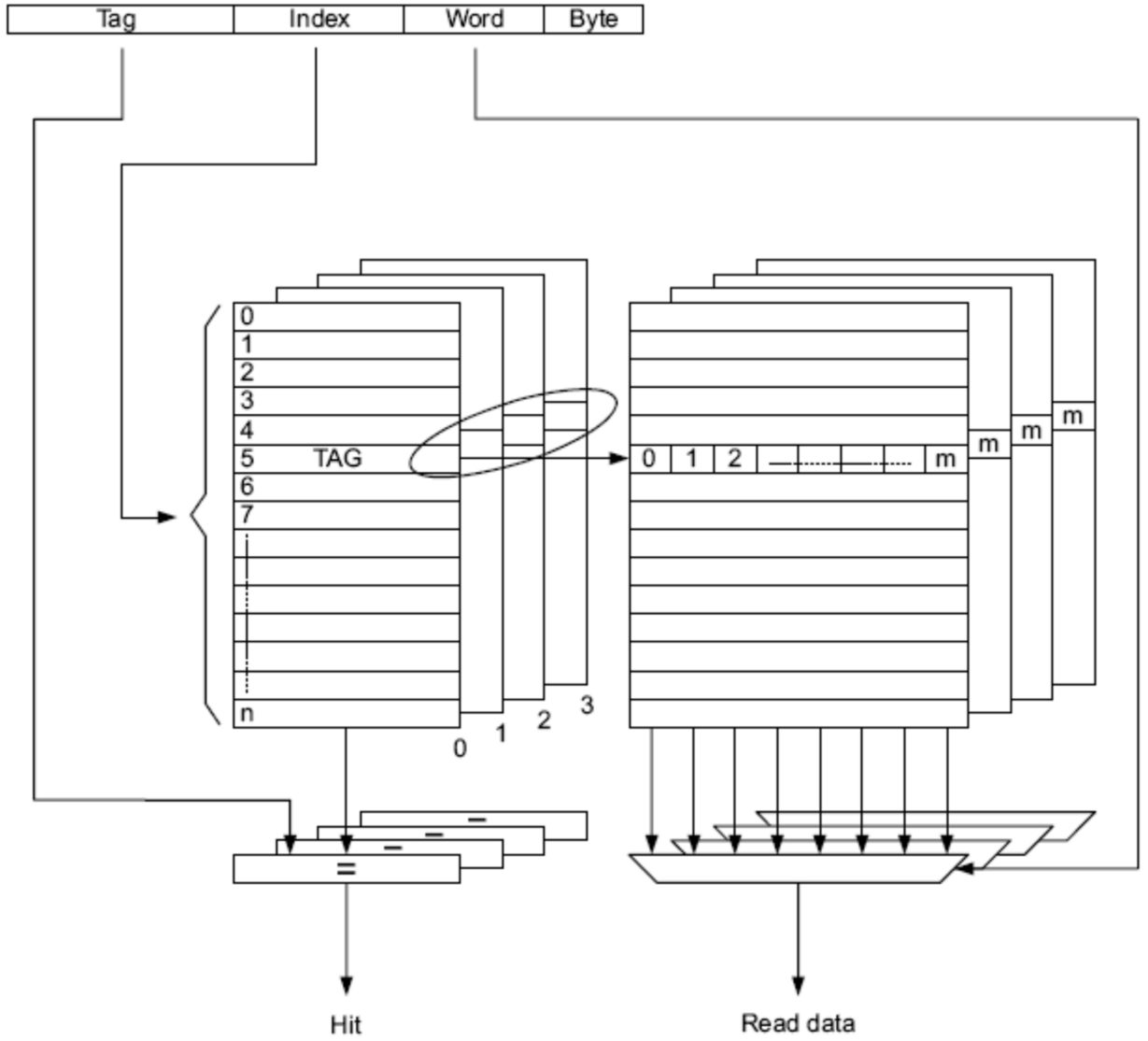
- L1 Data cache
  - 2 x 32 KBytes
  - 8-way set associative
  - 64-byte line size
- L1 Instruction cache
  - 32 Kbytes
  - 8-way set associative
  - 64-byte line size
- L2 cache
  - 4096 KBytes,
  - 16-way set associative
  - 64-byte line size



Exemple : Intel i7 4 coeurs (Sandy Bridge)

- L1 Data cache (1 par coeur)
  - 32 KBytes
  - 8-way set associative
  - 64-byte line size
- L1 Instruction cache
  - idem Data cache
- L2 cache (1 par coeur)
  - 256 KBytes
  - 16-way set associative
  - 64-byte line size
- L3 cache (commun aux 4 coeurs
  - contient le processeur graphique

Cache architecture 1



Position de la cache

- Cache physique: la cache est placée après le gestionnaire de mémoire virtuelle (MMU): elle reçoit une adresse physique
- Cache logique: la cache est placée avant le gestionnaire de mémoire virtuelle (MMU): elle reçoit une adresse virtuelle. Ce cas demande une cache moins rapide. Mais une adresse physique peut être traduite à plusieurs adresses logiques: si elles sont toutes dans la cache, l'écriture dans l'une laisse les autres inchangées

Cours ARO2

MEMOIRE VIRTUELLE

Mémoire virtuelle

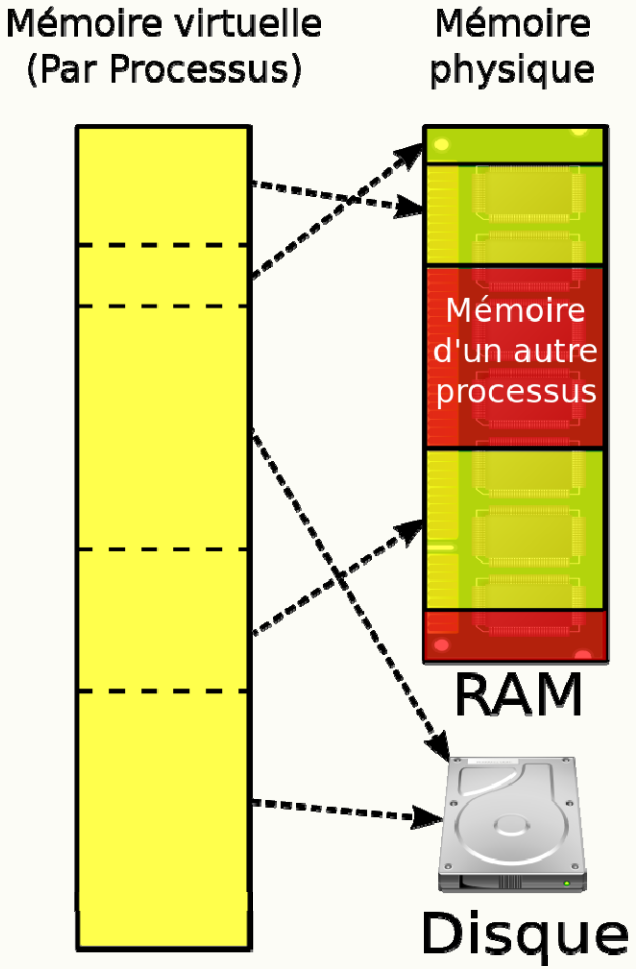
Définition:En informatique, le mécanisme de mémoire virtuelle a été mis au point dans les années 1960. Il repose sur l'utilisation de traduction à la volée des adresses (virtuelles) vues du logiciel, en adresses physiques de mémoire vive. La mémoire virtuelle permet :

- **d'utiliser de la mémoire de masse comme extension de la mémoire vive**
- **d'augmenter le taux de multiprogrammation ;**
- **de mettre en place des mécanismes de protection de la mémoire ;**
- **de partager la mémoire entre processus.**

- **Espace du disque dur interne d'un ordinateur qui vient seconder la mémoire vive, Elle se concrétise par un fichier d'échanges (fichier swap), lequel contient les données non sollicités constamment. La mémoire virtuelle, comme son nom l'indique, sert à augmenter artificiellement la mémoire vive. Elle est aussi moins performante.**

Source wikipédia

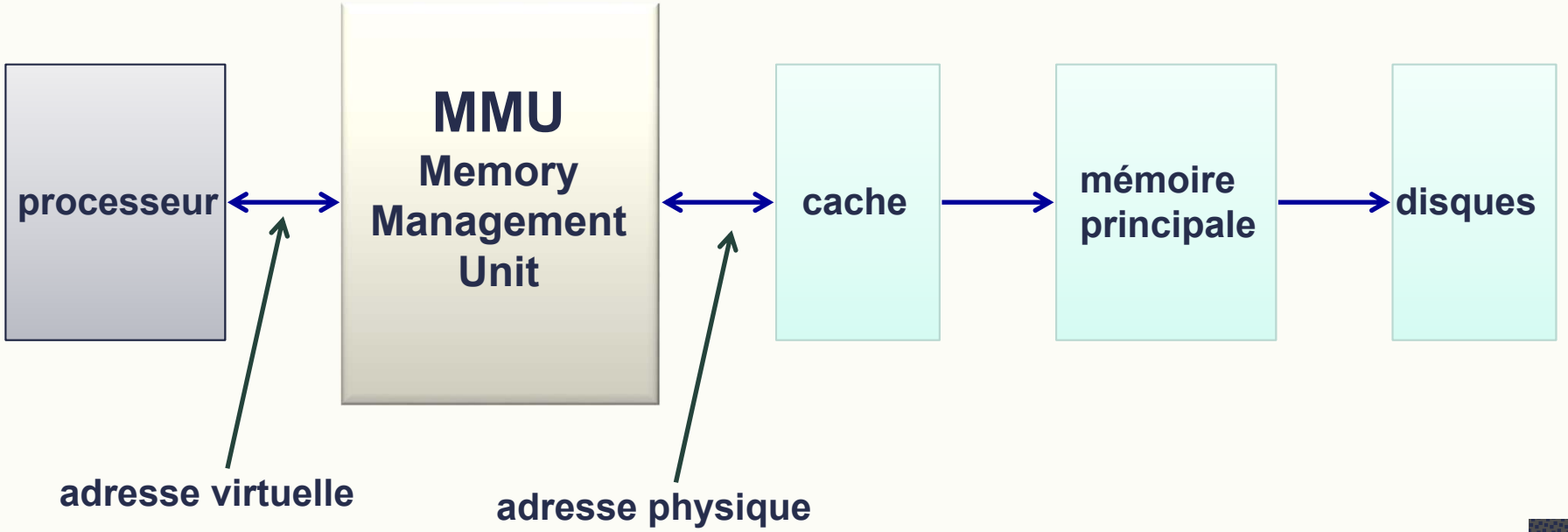
Mémoire virtuelle



Source wikipédia

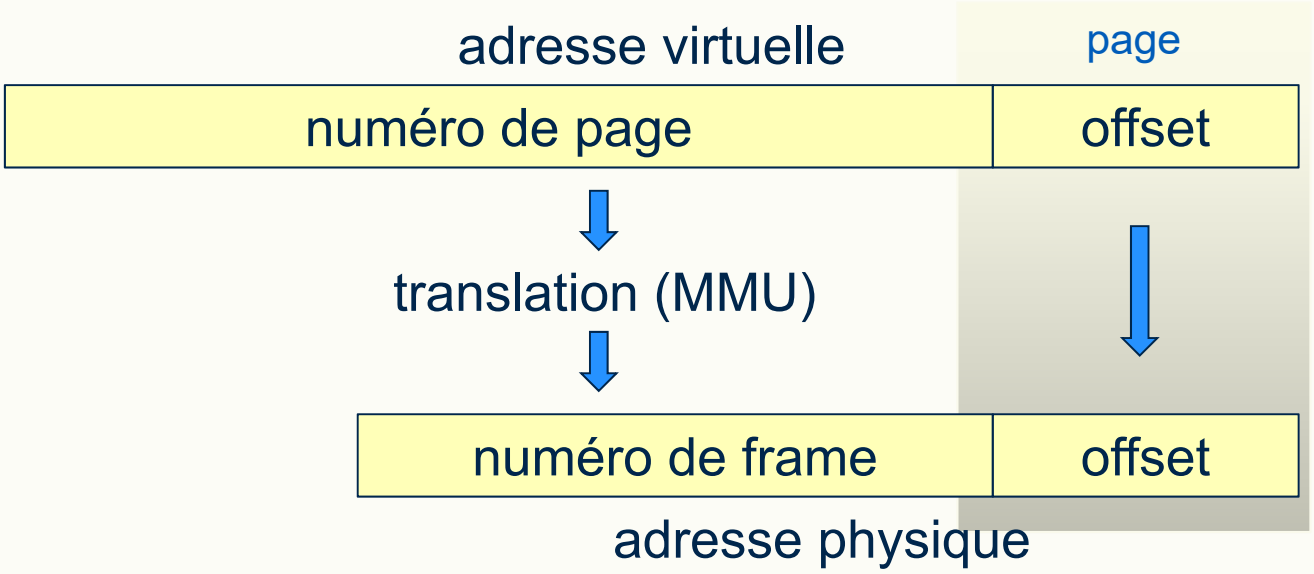
Mémoire virtuelle

- Translation d'adresse
- Taille mémoire virtuelle > taille mémoire physique
- Protection des données => mode multitâche, multiutilisateur, noyau / utilisateur (Linux),

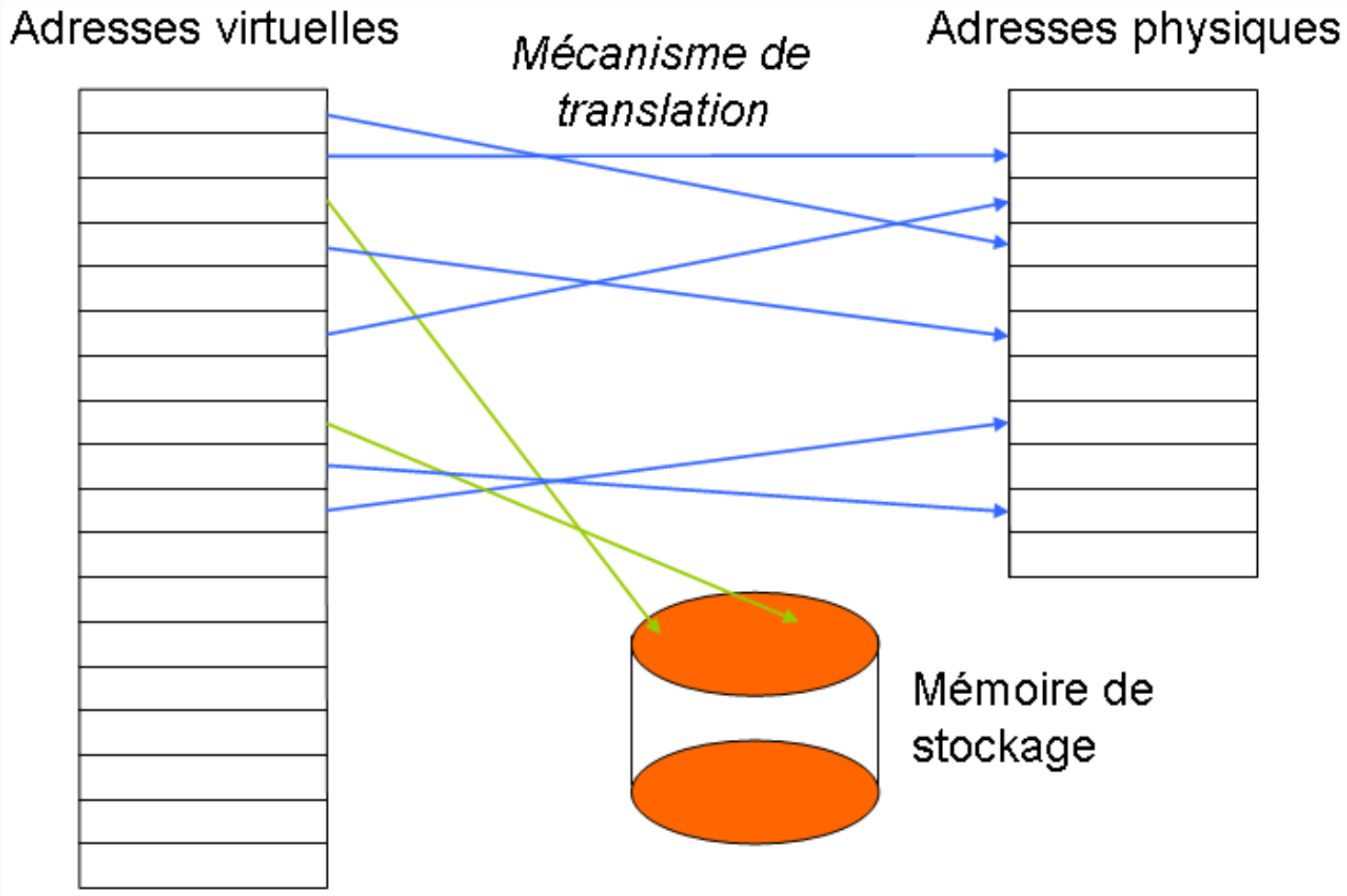


Mémoire virtuelle

- **Mémoire partitionnés en blocs :**
  - **Pages (virtuel) ⇔ Frames (pages physique)**
  - **Numéro de page translaté en numéro de frame**
  - **Offset (déplacement) inchangé**

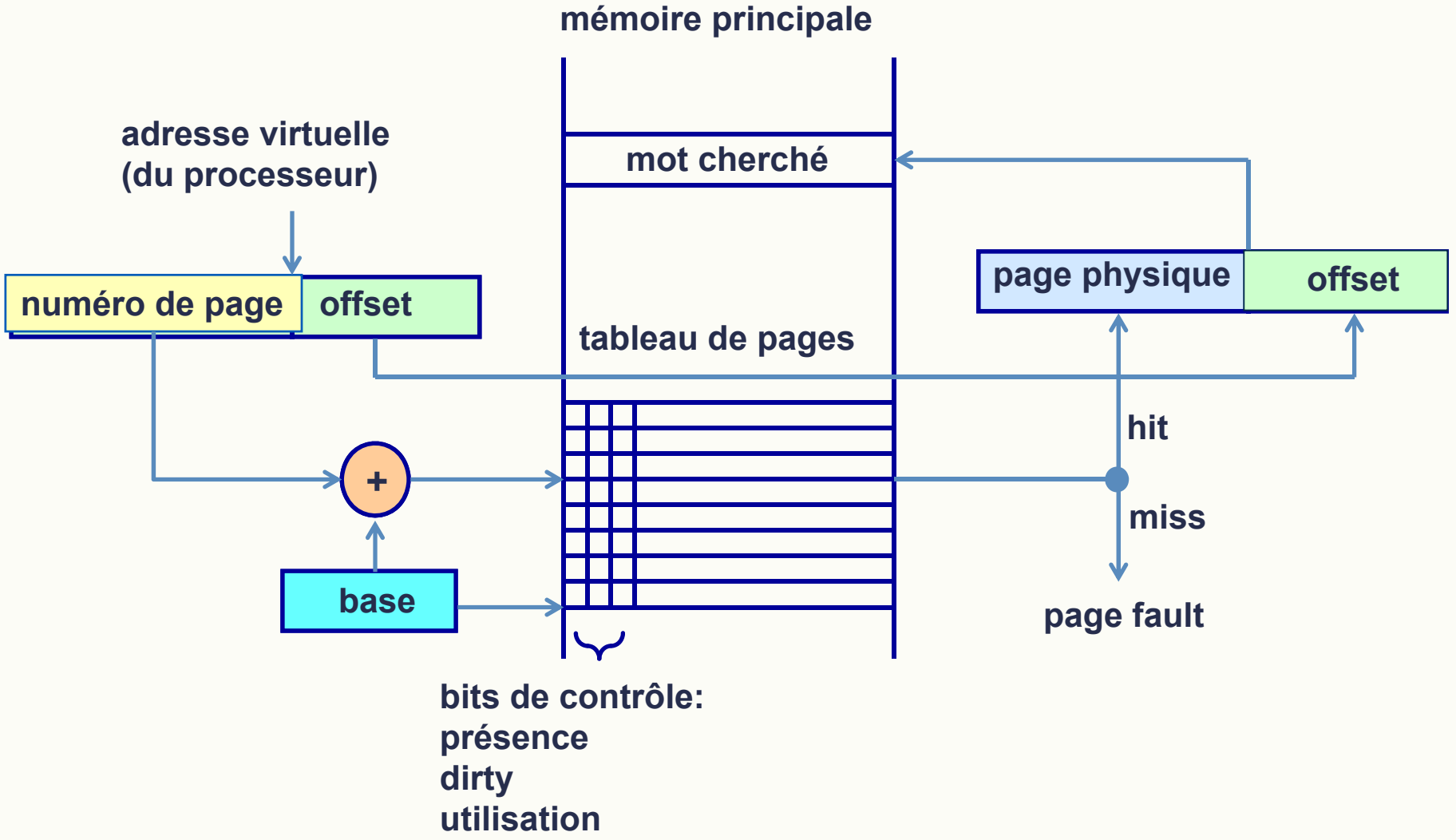


Translation d'adresses



wikipédia

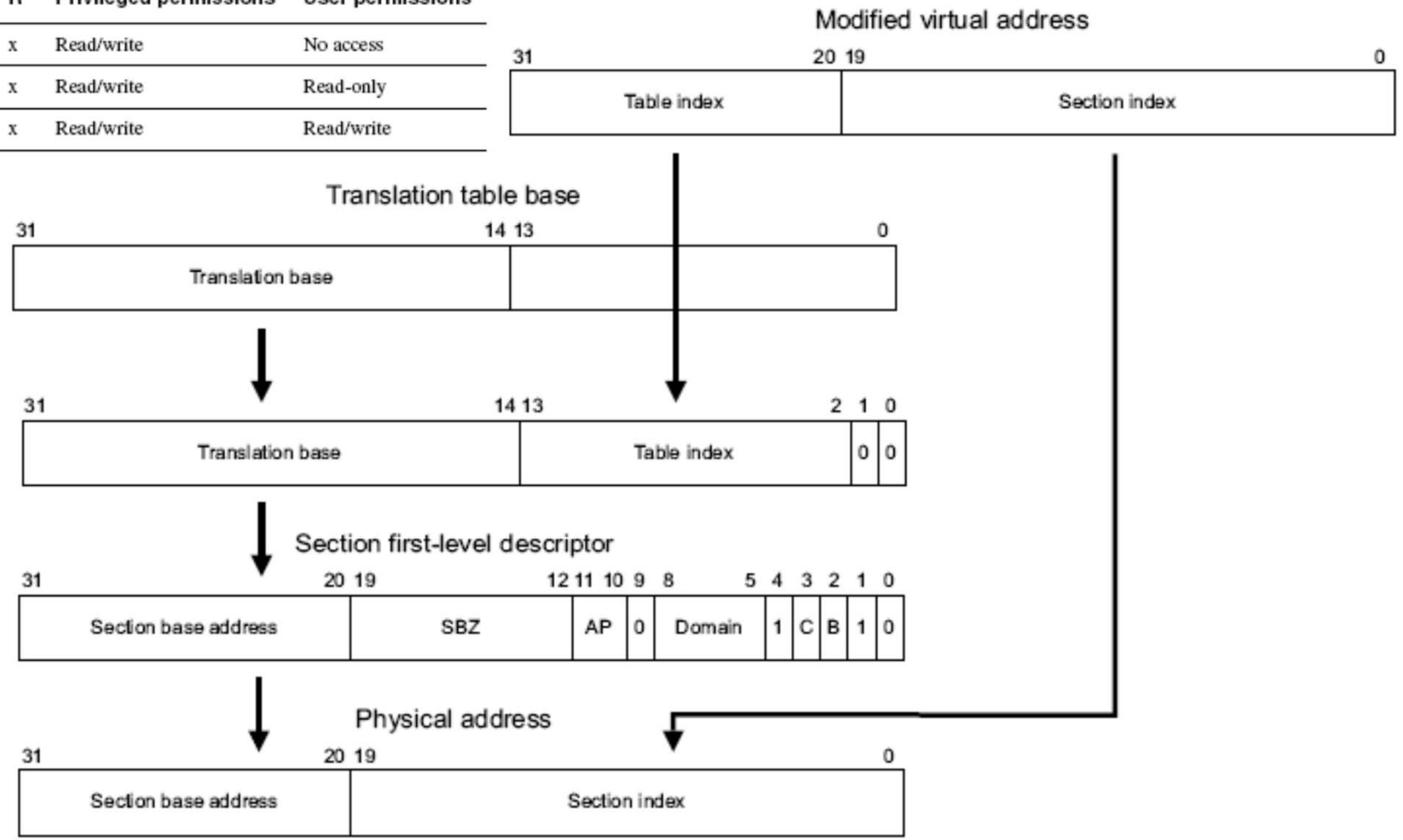
Accès à une adresse physique



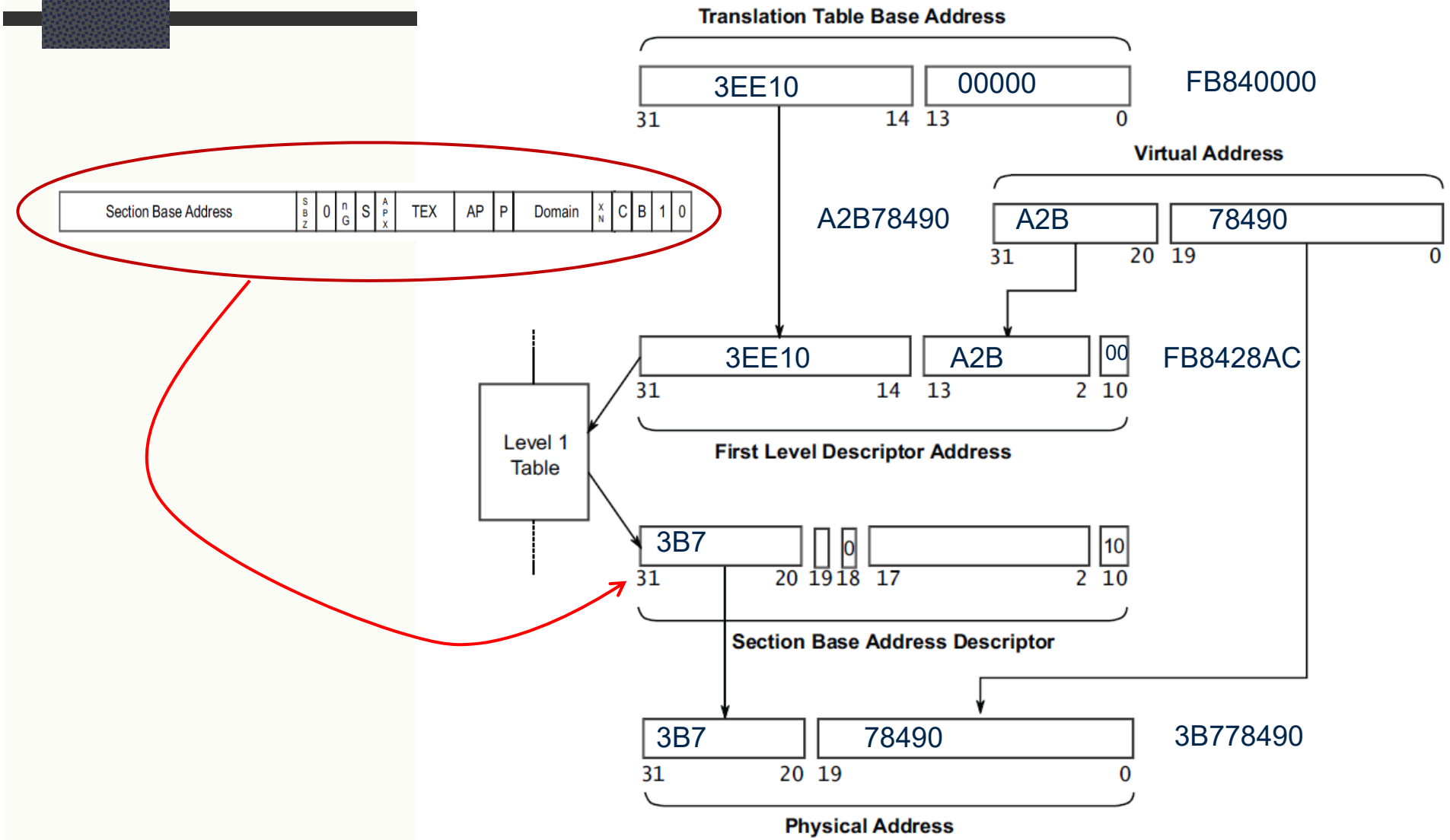
Virtual memory

Table 3-12 Interpreting access permission (AP) bits (continued)

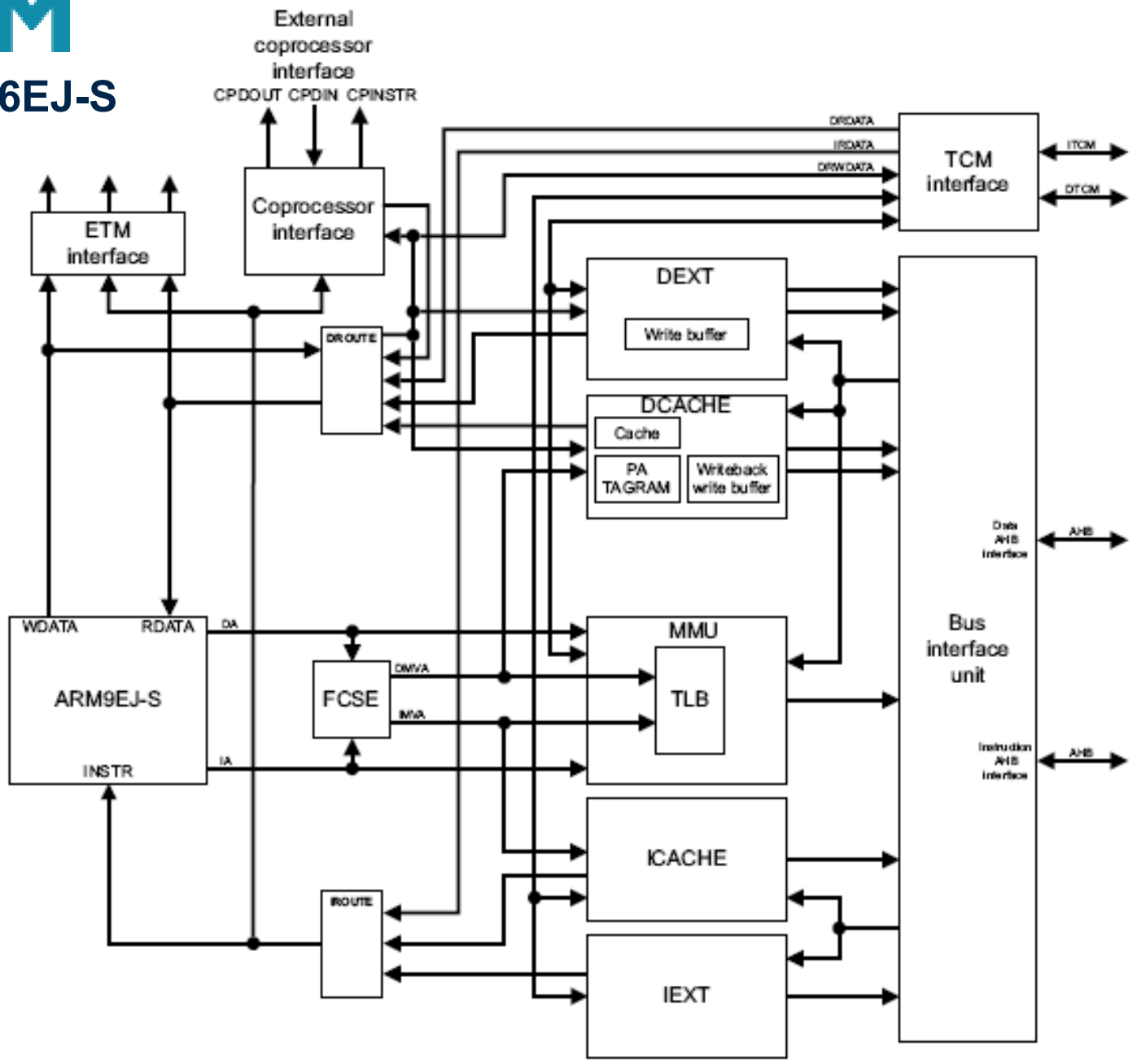
AP	S	R	Privileged permissions	User permissions
0 1	x	x	Read/write	No access
1 0	x	x	Read/write	Read-only
1 1	x	x	Read/write	Read/write



Cortex-A8:
Génération d'une adresse physique



ARM926EJ-S


External coprocessor interface
CPDOUT CPDIN CPINSTR
Coprocessor interface
ETM interface
DROUTE
IROUTE
DEXT
Write buffer
DCACHE
Cache
PA TAGRAM
Writeback write buffer
ICACHE
IEXT
MMU
TLB
FCSE
DMVA
IMVA
WDATA
RDATA
DA
ARM9EJ-S
INSTR
IA
DRDATA
IRDATA
DRWDATA
TCM interface
ITCM
DTCM
Data AHB interface
AHB
Instruction AHB interface
AHB
Bus interface unit