Institut
Mines-Télécom

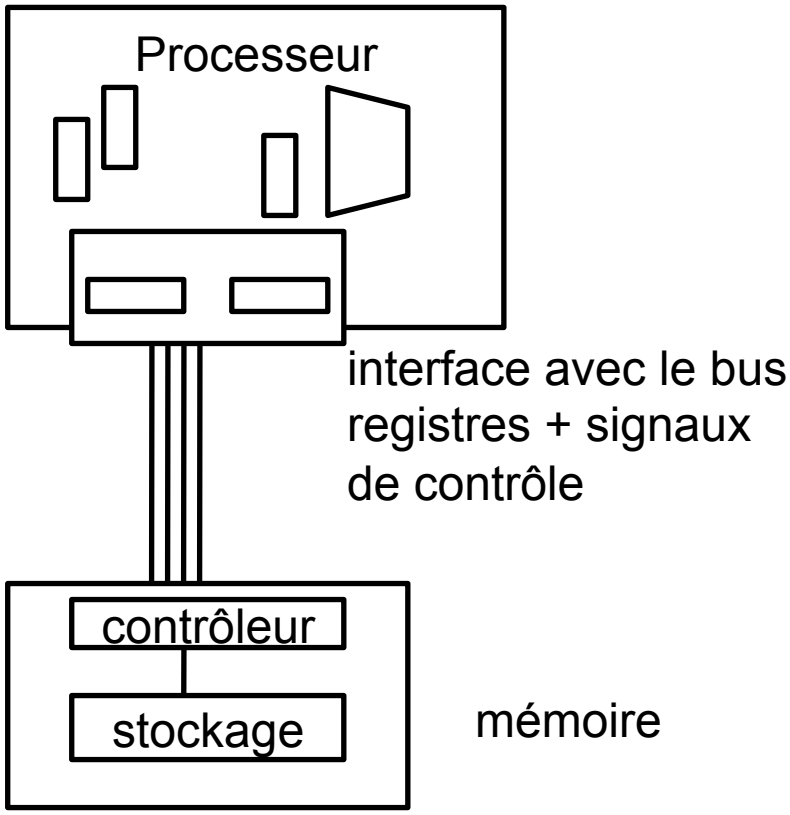
Stockage et mémoire, Du principe vers la réalité

Responsable :Thomas Robert C234-4
thomas.robert@telecom-paristech.fr

Intervenants :Tamy Boubekeur, Guillaume Duc, Gérard Mouret, Thomas Robert

La vision idéalisée de la mémoire

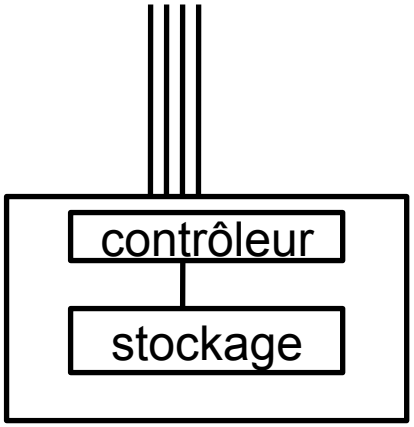
- **1 espace d'adressage linéaire**
- **Transfert simultané de w bits**
- **L'interaction : transaction**
  - Définition de l'adresse
  - Définition de la nature de l'accès (lecture/écriture)
  - Définition de la source ou de la destination locale (registre)
  - Réalisation du transfert de donnée « atomique » avec un délai τ via le bus
- **Fonctionnement interne = traitement de la requête**



Mémoire, principe et étude du cas de la DRAM

L'accès à la mémoire
Principe et exemple de mise en œuvre

- **Principe :**
  - Stockage d'une quantité d'information non triviale
  - Échanges de petites quantités d'information (vers ou depuis des registres)
  - Accès à des adresses arbitrairement réparties (d'où Random Access Memory)
  - Échange via un modèle de transaction (requête / réponse)



1) Décodage de la requête
2) Contrôle du stockage pour réaliser le transfert
   - Sélection de l'emplacement
   - Mise à jour ou diffusion du contenu

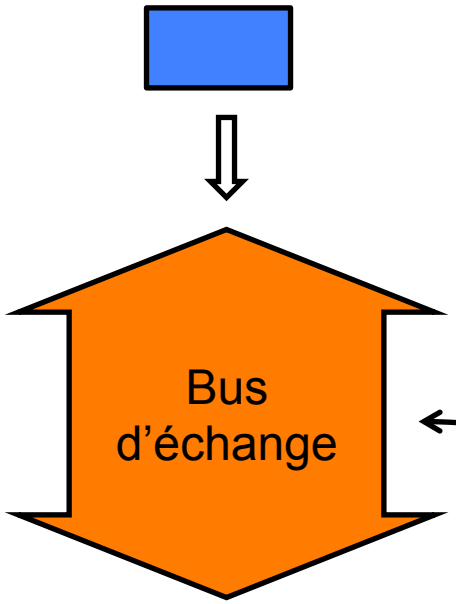
Le stockage – technologies et propriétés

- **Technologie de stockage : le problème de stabilité**
  - Static :
    - information == tension dans un circuit de 6 transistors
    - Stabilité <= alimentation du circuit de stockage (indépendant de l'information contenue
  - Dynamic :
    - information == charge sur une capacité
    - Stabilité <= consultation du contenu (détection par seuil) et rechargement périodique de la capacité
- **Comparaison**
  - Coût de production SRAM >> DRAM
  - Latence d'accès SRAM << DRAM
  - Consommation énergétique
    - Si aucun accès DRAM>SRAM
    - Si accès fréquent (lecture/écriture) SRAM ~DRAM (ou >)

Comprendre la structure
Unité de transfert, de stockage

L'information transférée

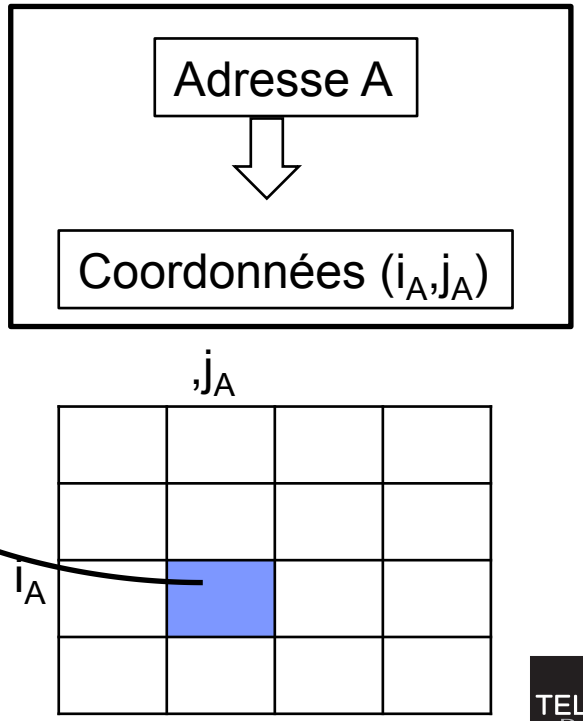
w bit = mot = 8, 16 ..64 bits



L'information stockée

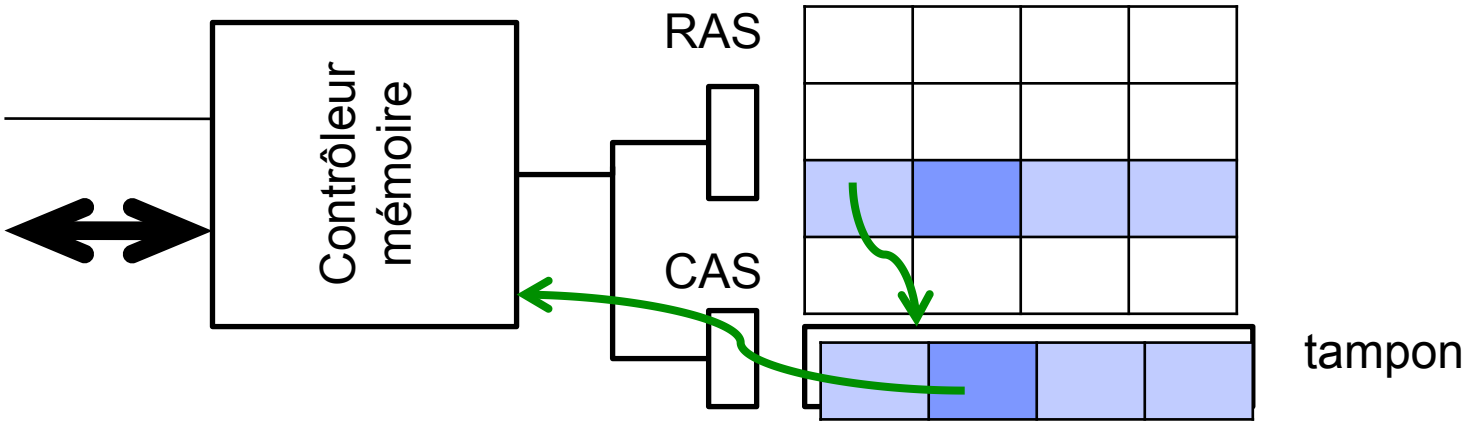
w' cellules = unité de stockage
~ supercellule

w' divise w



Comprendre le traitement d'une lecture
Accès à l'adresse A codée sur 4 bits

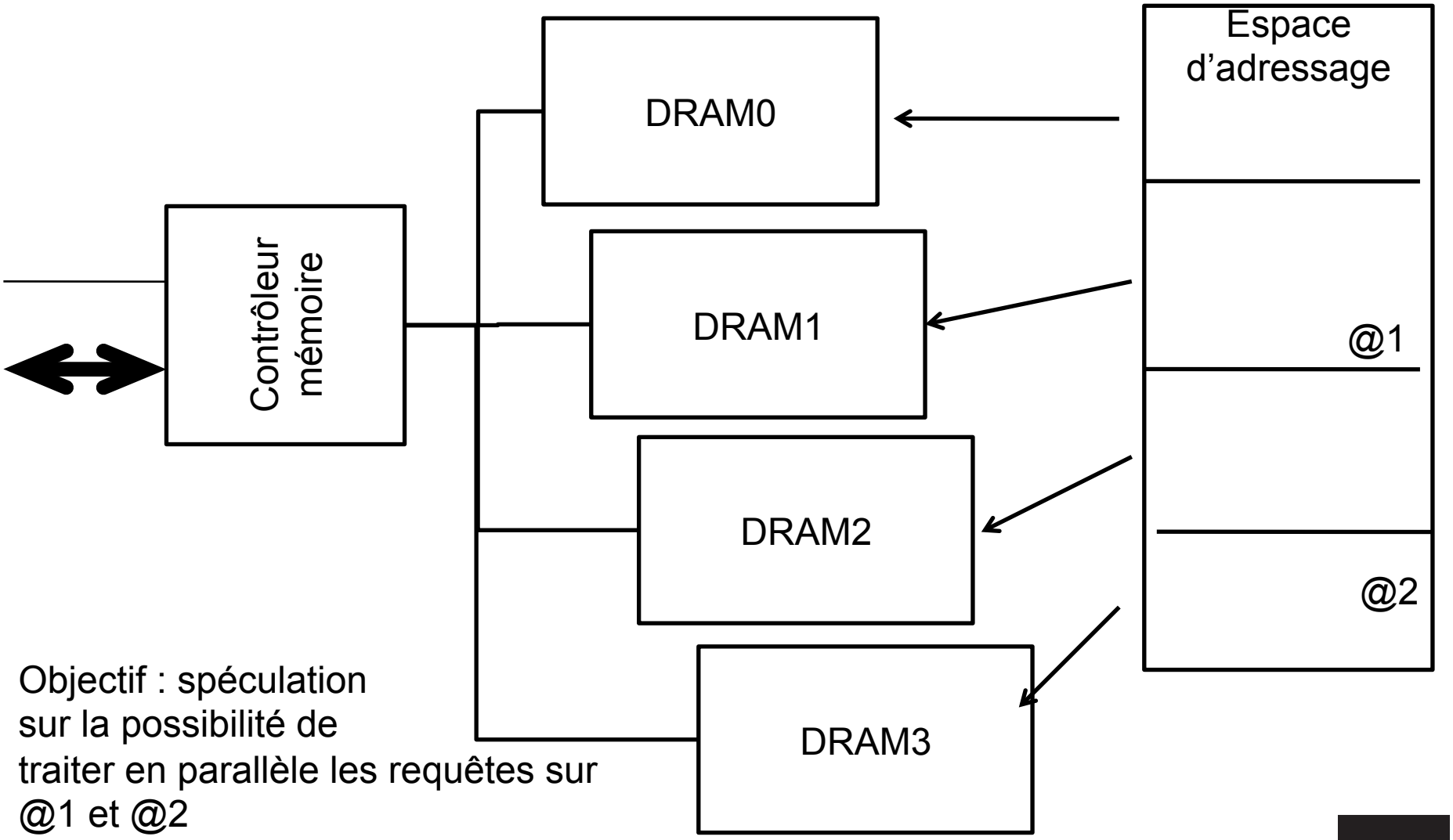
- **Remarque 1 : 2^4 =16 la taille du block**
- **Étapes :**
  - Traduction de l'adresse en coordonnées
  - Récupération de la ligne contenant la cible
  - Sélection de la cellule désirée (extraction)
  - Envoi sur un bus d'échange avec le processeur



Latence et débit

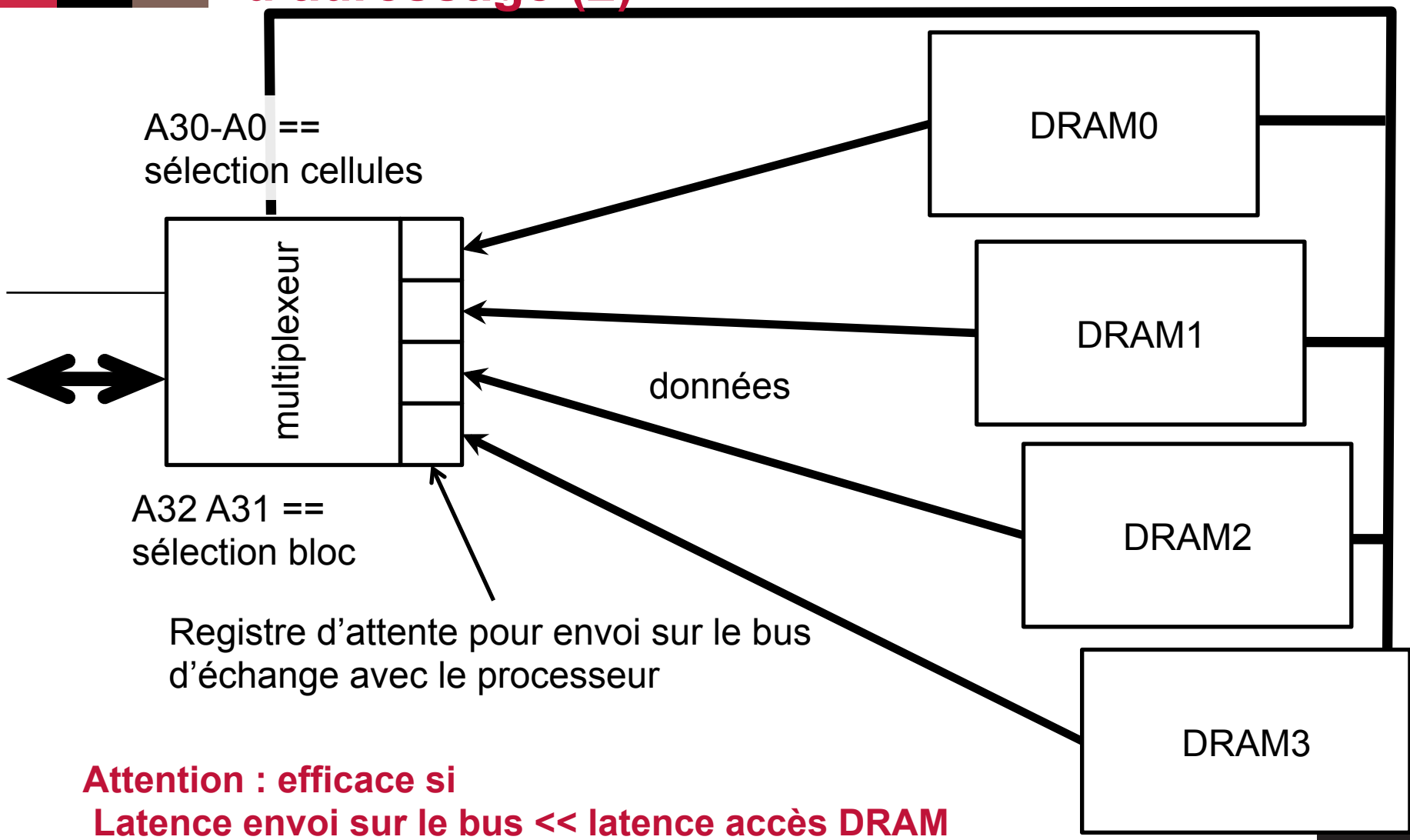
- **Latence d'accès :**
  - Temps de recopie de la ligne
  - Temps de sélection de la colonne
- **Débit :**
  - Le circuit DRAM émet le contenu d'une cellule, potentiellement << taille d'un mot
  - Le cycle doit se terminer pour sélectionner une ligne
- **Amélioration :**
  - Juxtaposition de plusieurs circuits DRAM (+ de débit)
  - Synchronisation des étapes sur une horloge ≠ acquittement (+ rapide)

Module mémoire et découpe de l'espace d'adressage (1)



Objectif : spéculation sur la possibilité de traiter en parallèle les requêtes sur @1 et @2

Module mémoire et découpe de l'espace d'adressage (2)



Attention : efficace si
Latence envoi sur le bus << latence accès DRAM

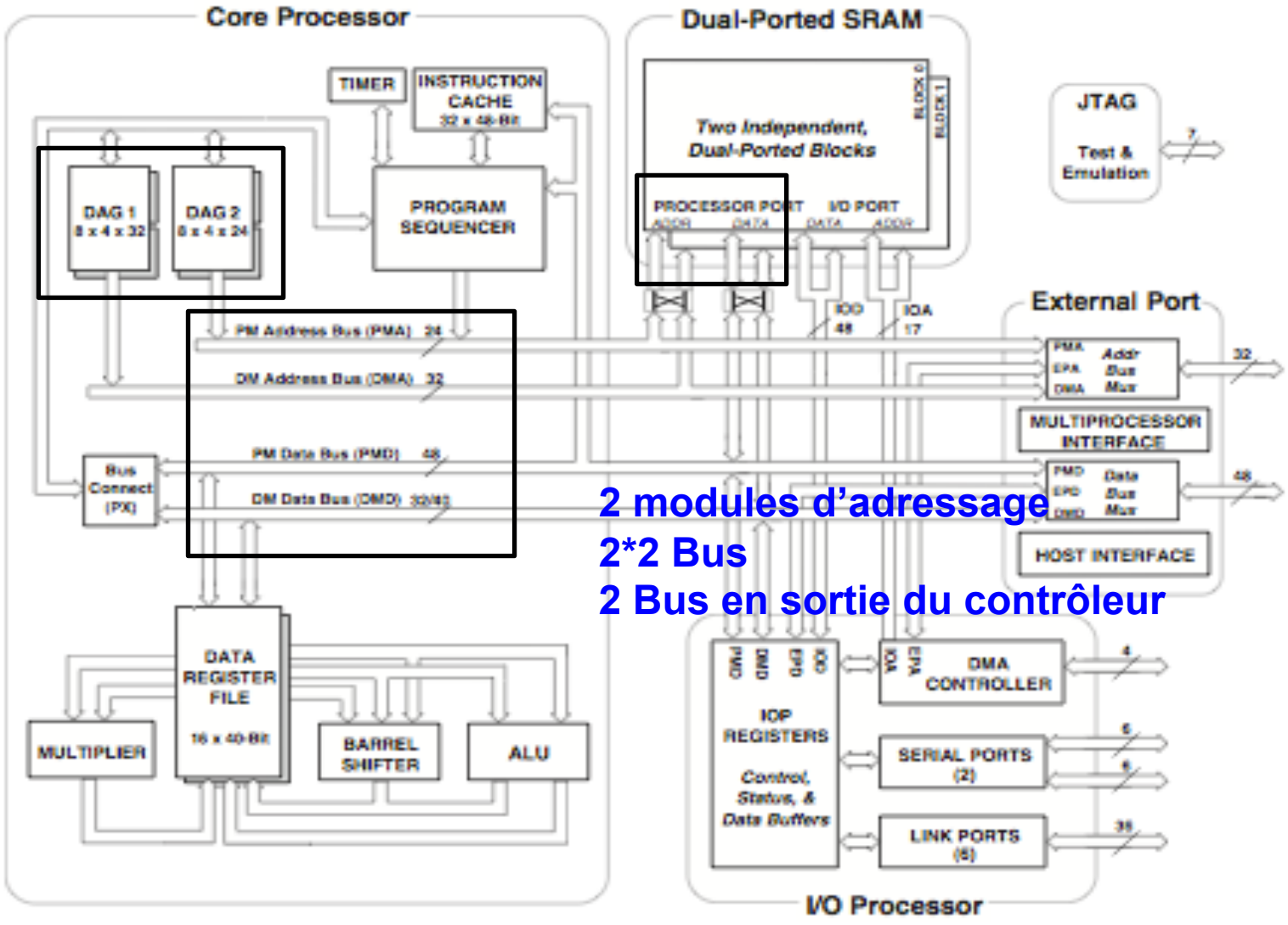
Les bus et le contrôleur

- **Observation :**
  - Une instruction est en mémoire => 1 accès pour la récupérer
  - Une instruction peut avoir besoin d'opérandes en mémoire

=> besoin de traiter plusieurs accès par instruction

- **Pb : comment l'information remonte-t-elle jusqu'au processeur ?**
- **Solutions Matérielles :**
  - Fréquence des bus très élevée + (multiplexage ou mise en file)
  - Multiplication des bus en sortie de la « mémoire » (dual bus)

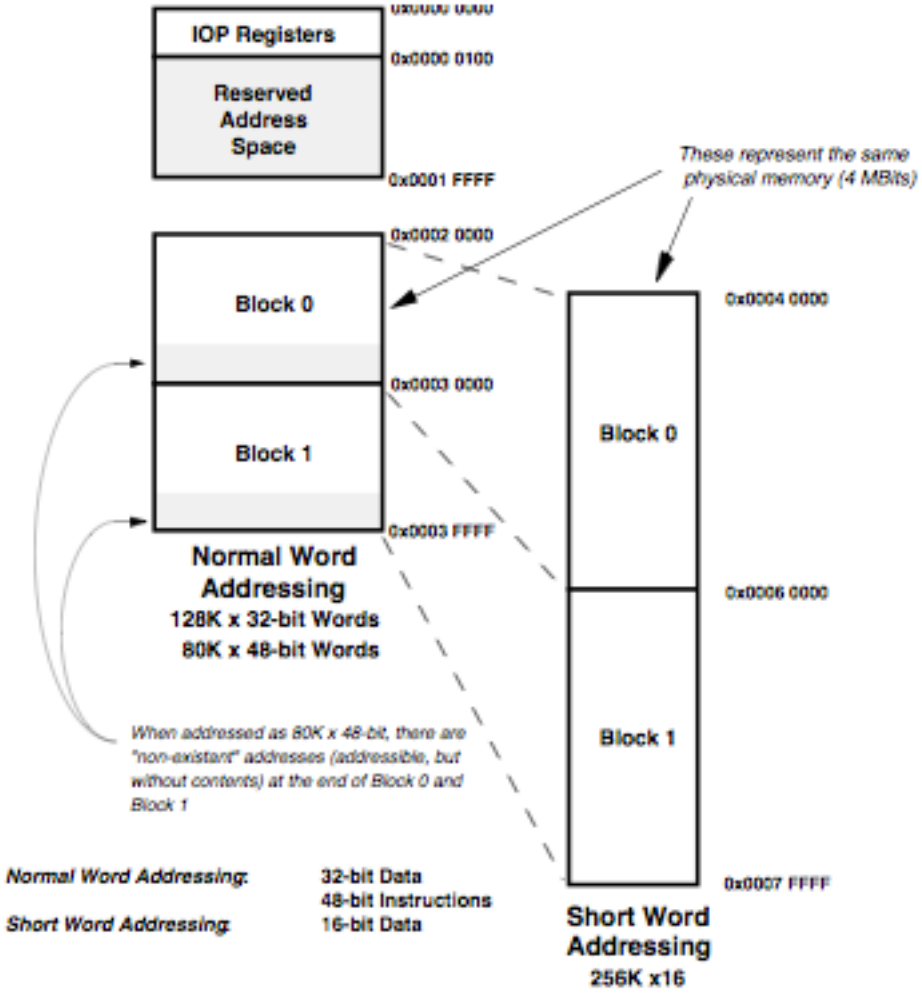
Le cas de l'architecture ADSP2106x une architecture Harvard détournée



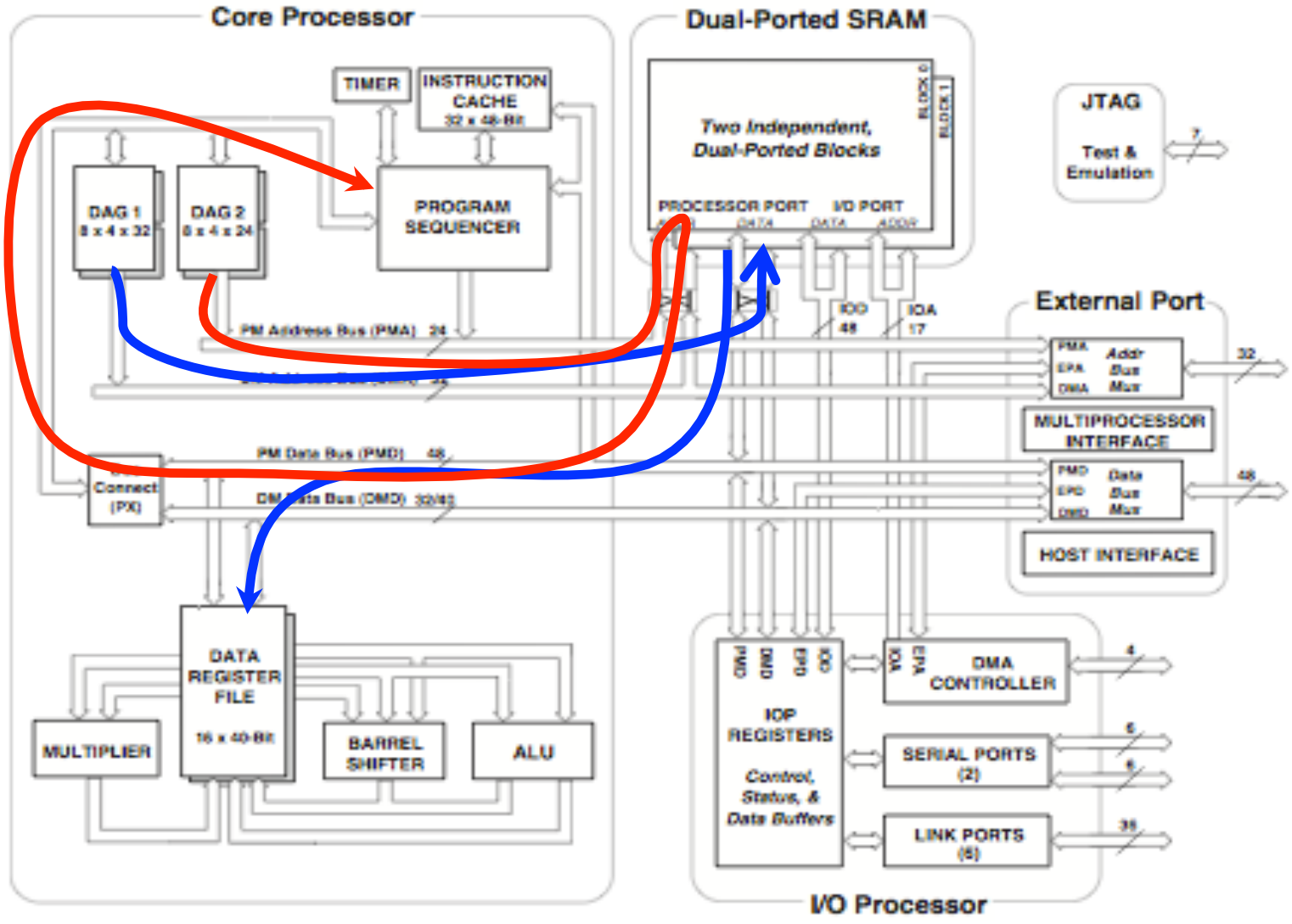
Le cas de l'architecture ADSP2106x

- **Unité d'adressage peut changer**
- **Par bloc : 32 ou 48 bit**
- **Réalité : super cellule=16**
- **Débit de transfert**
  - Les deux blocs peuvent contenir des données
  - 1 bus d'adresse et d'échange par bloc
  - 1 cycle processeur = 1 transaction mémoire

=> 2 transferts mémoire par cycles

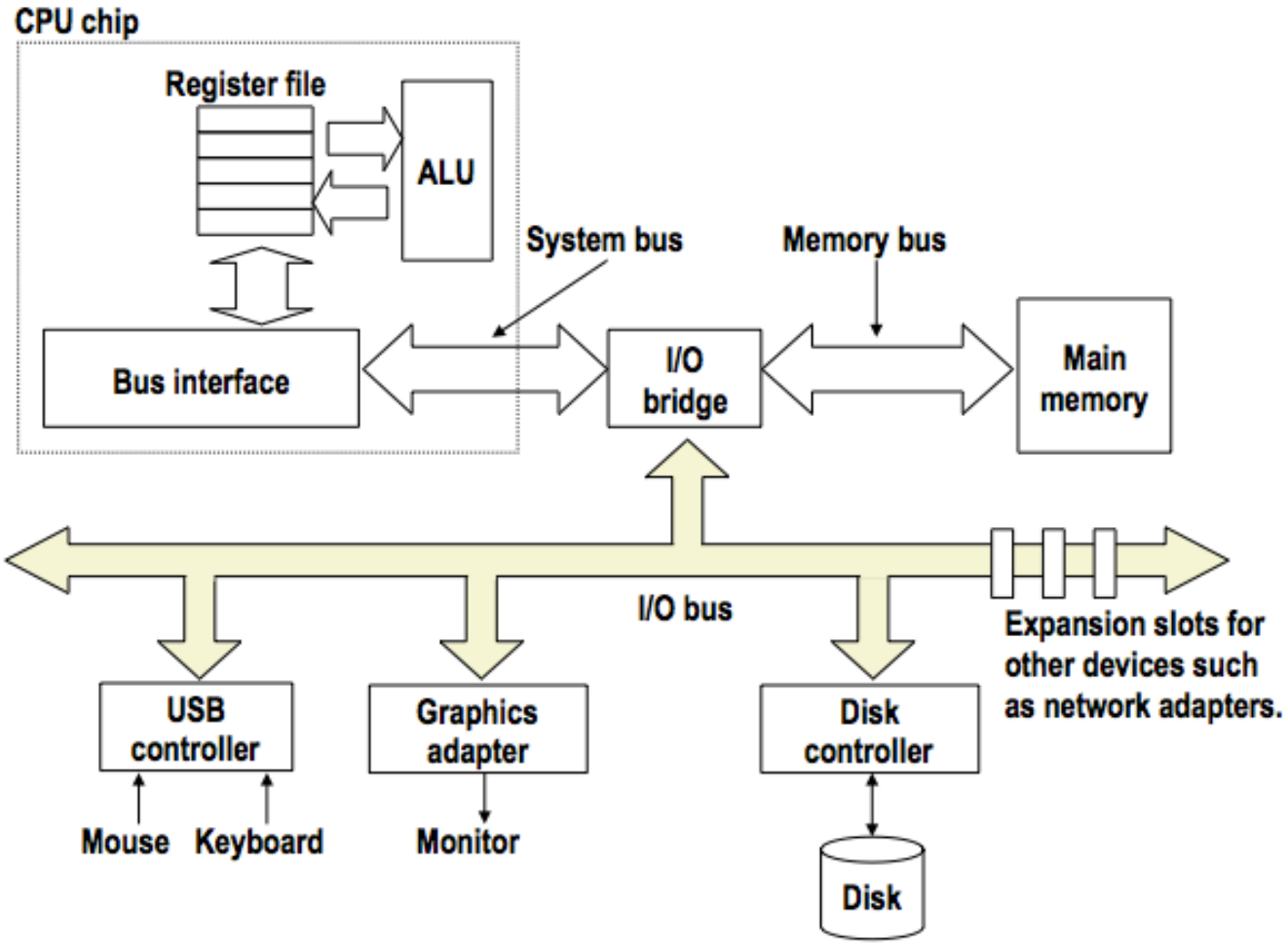


Le cas de l'architecture ADSP2106x



Les autres stockages

Architecture de l'ordinateur et stockage



Quelques mots sur les autres stockages

- **Les contraintes d'accès**
  - Granularité grossière d'accès (>>>octet)
  - Lecture synchronisée avec le processeur du contenu et filtrage
  - Le transfert de contenu avec des stockage de masse peut se faire de manière asynchrone en partageant la mémoire
- **Les performances**
  - Le temps d'accès à la donnée est souvent très long car l'accès à l'information nécessite
    - Un déplacement physique
    - l'accès au bus pour faire remonter l'information
- **Synchronisation avec le processeur**
  - Utilisation des interruptions pour dialoguer avec le processeur

Cache et hiérarchie mémoire

Le problème de performance

SRAM

Metric	1980	1985	1990	1995	2000	2005	2010	2010:1980
$/MB	19,200	2,900	320	256	100	75	60	320
access (ns)	300	150	35	15	3	2	1.5	200

DRAM

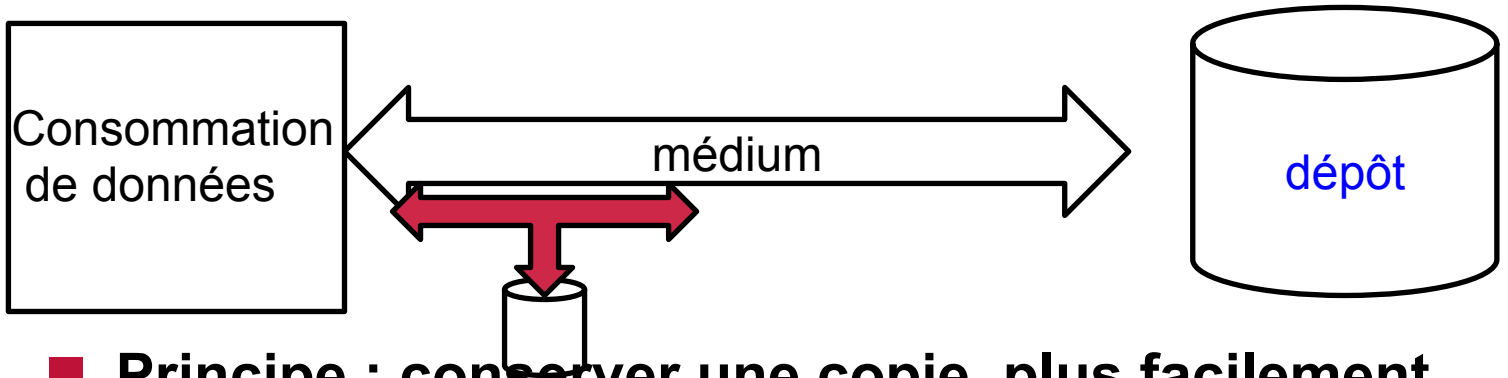
Metric	1980	1985	1990	1995	2000	2005	2010	2010:1980
$/MB	8,000	880	100	30	1	0.1	0.06	130,000
access (ns)	375	200	100	70	60	50	40	9
typical size (MB)	0.064	0.256	4	16	64	2,000	8,000	125,000

Disk

Metric	1980	1985	1990	1995	2000	2005	2010	2010:1980
$/MB	500	100	8	0.30	0.01	0.005	0.0003	1,600,000
access (ms)	87	75	28	10	8	4	3	29
typical size (MB)	1	10	160	1,000	20,000	160,000	1,500,000	1,500,000

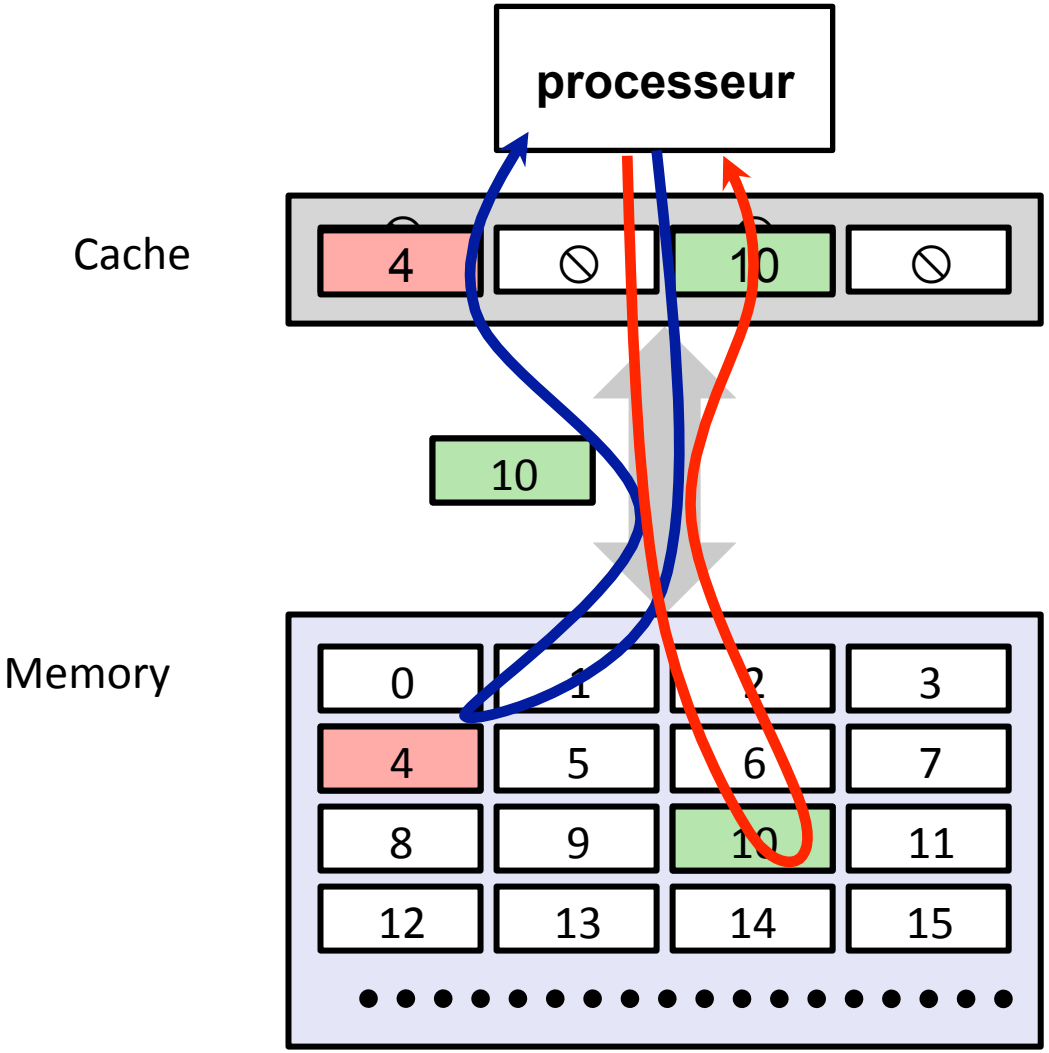
Principe d'un cache

- **Accès lent aux données dans le « dépôt »**



- **Principe : conserver une copie, plus facilement accessible de ces données sur un stockage rapide SR**
- **Question ouverte :**
  - Cette copie va-t-elle servir
  - Comment cela modifie-t-il la procédure d'accès aux données
  - Comment gère-t-on la ressource SR

Les événements élémentaires sur un cache



Consomme 4
Consomme 10
Consomme 10

Cache hit = accès à la donnée via le cache sans accès à la mémoire

Cache miss, accès à une donnée absente du cache => accès mémoire + mise à jour du cache

Placement et Identification du contenu du cache

- **Les questions clés :**
  - Quelle quantité d'information recopie-t-on ?
  - Où la dispose-t-on ?
  - Comment gère-t-on la surpopulation ?
- **Les familles de cache == une taille + stratégie de placement + une politique de remplacement**
- **Identification du contenu du cache :**
  - Observation : 1 place dans le cache => N emplacement mémoire
  - Proposition : le cache doit contenir des informations pour déterminer l'emplacement stocké

Stockage et identification du contenu du cache

- **Hypothèse : Le cache accélère l'accès à un ensemble d'octets indexés linéairement (adresses sur 2^M bits)**
- **Identification : stockage conjoint Identifiant + donnée**
- **Initialisation et cohérence avec le niveau inférieur (mémoire)**
  - État d'initialisation : valid/not valid (indique si la ligne est initialisée)
  - État d'usage : (pas de nom standard) si 0 même contenu en mémoire et dans le cache
- **1 ligne = unité de transfert/stockage**
  B octets répartis entre :
  - W bits pour l'identifiant
  - X bits pour l'état (souvant plus petit qu'un octet)
  - Y octets pour le contenu

- **Remarque : pour des raison pratique W X Y sont souvent des puissances de 2**

Placement : Quelle ligne de cache utilise-t-on

- **Stratégie de placement :**
  - Du point de vue de la ligne, une ligne peut stocker
    - un contenu d'adresse quelconque
    - er un contenu provenant d'un ensemble d'adresse restreint
  - Du point de vue de l'adresse, une adresse se place
    - au choix dans un ensemble de lignes (en fonction de leur contenu)
    - dans une seule ligne (prédéterminée par l'adresse)

@ \ ligne	Stocke un contenu d'@ quelconque	Stocke un contenu d'un ensemble d'@
Peut être placé sur plusieurs lignes	Cache associatif complet (fully associative)	Cache associatif par blocs (set associative)
Ne peut aller que sur 1 ligne	Pas cohérent	Cache par placement direct (direct map)

Le cache associatif complet (1)

- **Observation :**
  L'identifiant doit permettre de reconnaître le contenu

- **Encodage de l'Identifiant & adressage du Contenu**
  - C'est un vase communiquant
  - Adresses sur 2^M bits, 1 adresse = 1 octet

  ⇒ Contenu 2^5 octets => 2^{M-5} contenus juxtaposés
  - Identifiant Adresse d'un contenu, contenu plus grand qu'un registre

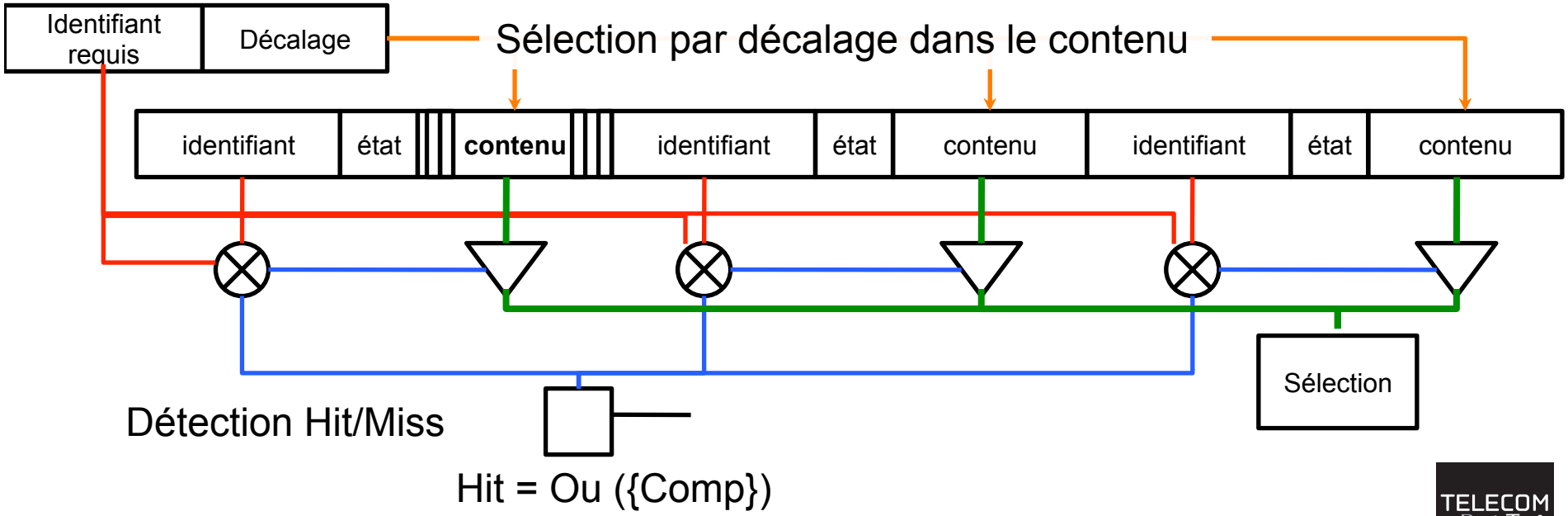
  ⇒ Récupération du contenu complet sur des registres ou filtrage

⇒ **Identifiant du contenu sur M-5 bits (poids fort de l'adresse)**

- **Identification**
  - Extraction des bits de poids fort
  - Comparaison avec l'identifiant stocké dans la ligne
  - Si correspondance sélection du contenu intéressant

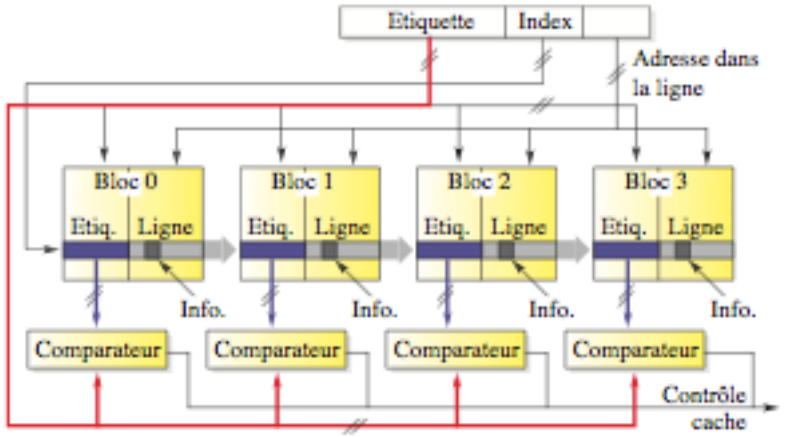
Le cache associatif complet (1)

- **Adresse == id de 1 ligne + 1 position dans la ligne**
- **Recherche dans le cache (3 ligne, exemple jouet)**
  - Comparaison identifiant requis / identifiant de ligne
  - En parallèle sélection de la fraction souhaitée du contenu
  - Le comparateur contrôle le sélecteur qui laisse passer (ou pas le contenu de la ligne

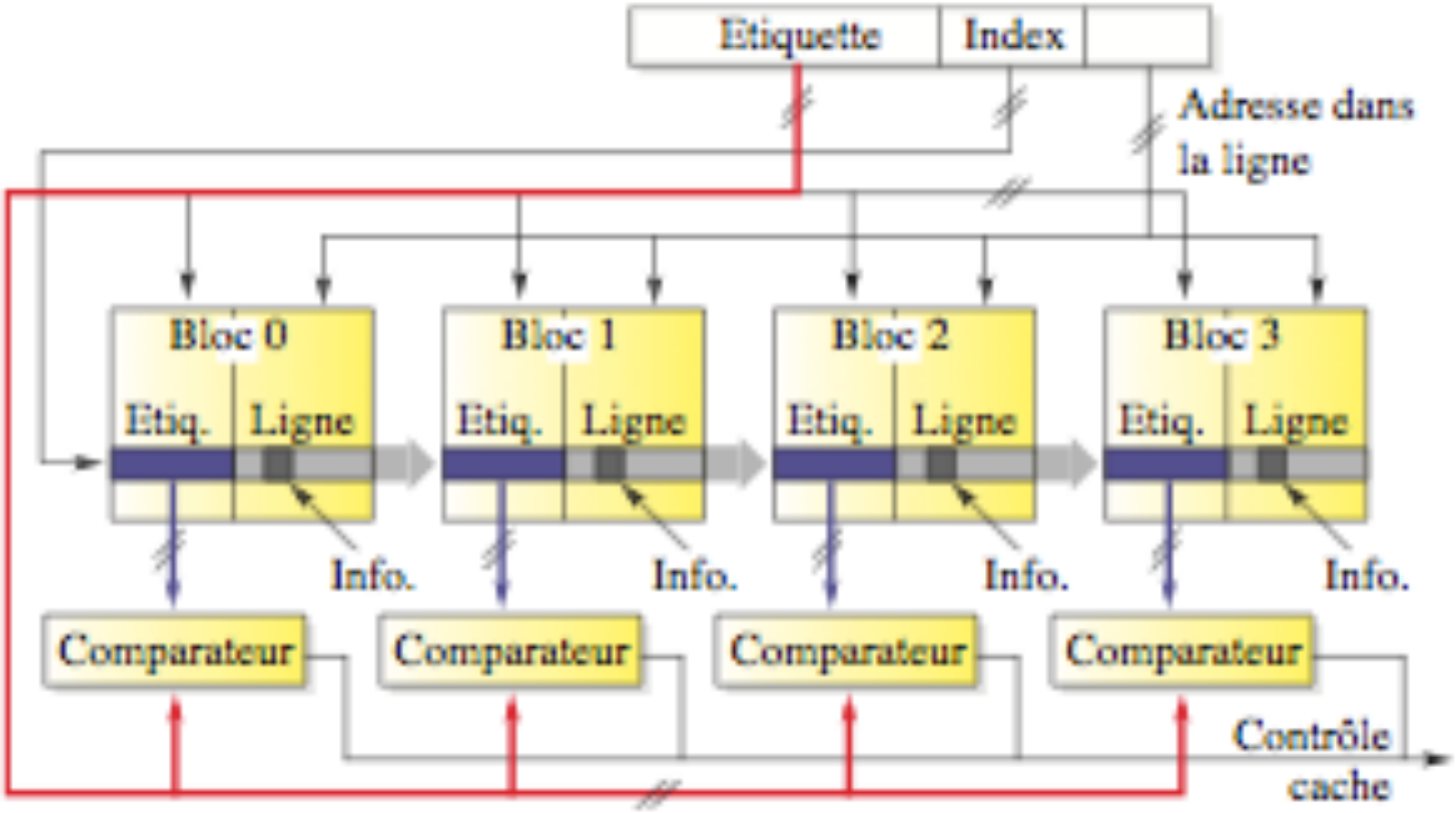


Un compromis 1 comparateur pour 1 ensemble de lignes

- **Principe :**
  - fragmenter le cache en colonnes =>
    N ensembles de P entrées, 1 comparateur pour 1 ensemble
  - Placement des contenus en fonction
    - d'un critère de répartition dans la mémoire
    - d'un critère de fraicheur / fréquence de l'accès …

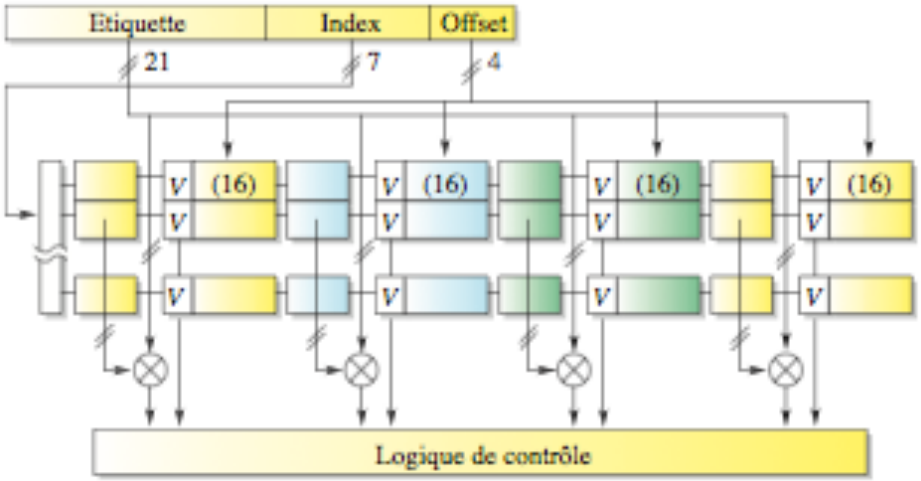


Comment cela marche dans le détail

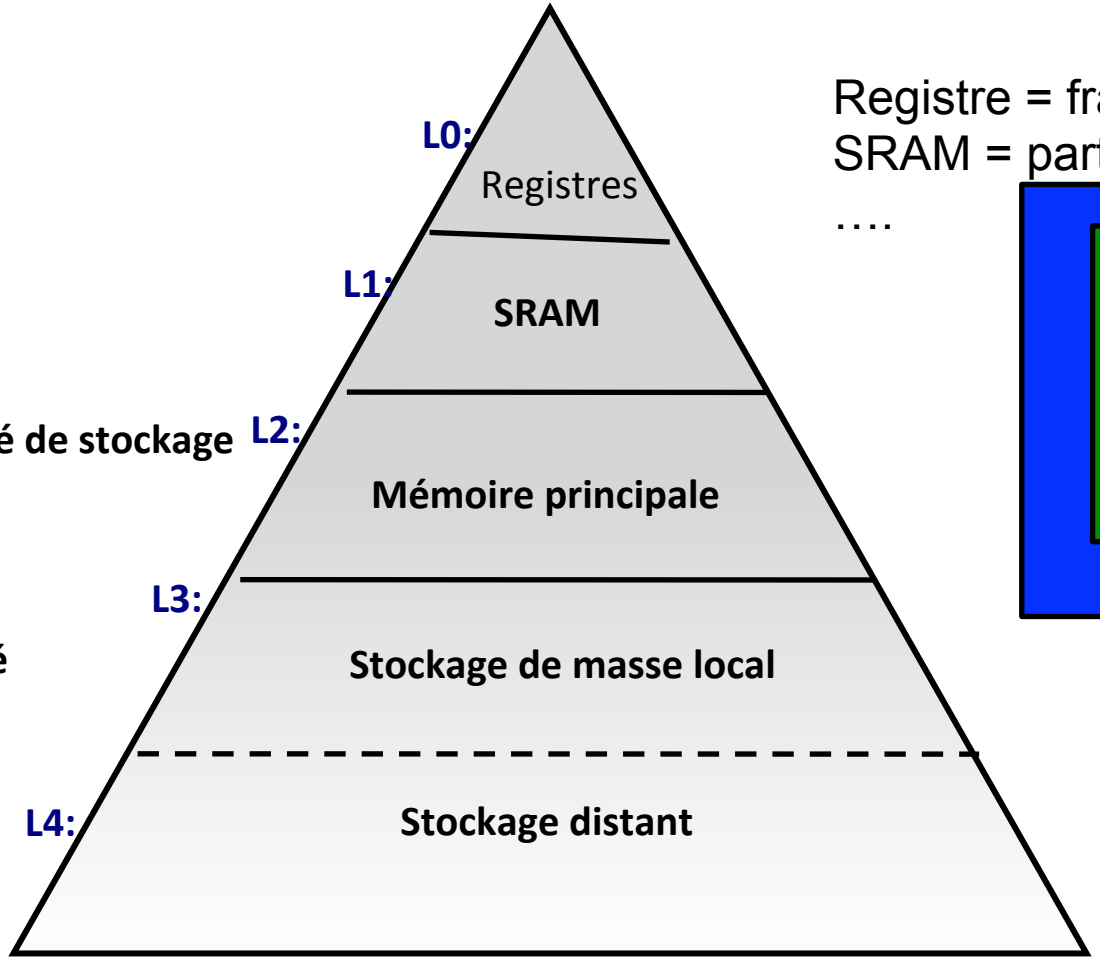


Politique least recently used sur 2 blocs

- **LRU sur deux bloc == complément du dernier lu**
- **Encodage de LRU sur 1 bit**
- **Avantage**
  - Occupation des blocs indépendante pour chaque entrée
  - Exemple : I80486



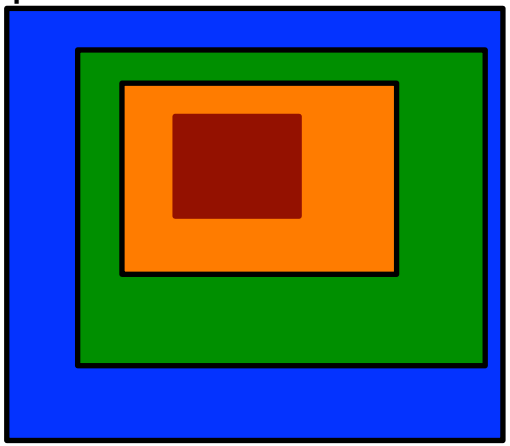
Hiérarchie mémoire au sens de l'inclusion



- de capacité
+rapide
+ cher / unité de stockage

+ de capacité
-cher
-rapide

Registre = fraction (SRAM)
SRAM = partie de DRAM
….

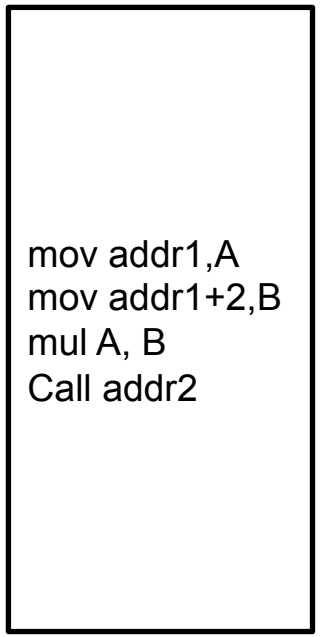


Référencement continu de l'information
=> adressage du contenu du disque dur ?

Mémoire virtuelle, le besoin, la mise en œuvre

Du code à l'adresse physique

Binaire



- **Ce que l'on ne veut pas**
  - Placement *en dur* des données et fonctions
- **Pourquoi ?**

⇒ **rend le « partage » de la mémoire plus complexe entre deux fonctions**

⇒ **Conception du logiciel requiert une vue globale de la mémoire**

- **Solution**

  Généraliser l'adressage indexé base + addr
- **En pratique :**
  - Adresse = 2 composantes, 1 segment + 1 décalage
  - Résolution de l'adresse :
    - Segment défini par un registre
    - Décalage défini dans l'instruction

Un code relogeable (1)

- **Peut on déplacer les codes suivant à n'importe qu'elle adresse :**

```
mov ax, 4
mov bx, 7
add ax, bx
mov bx, 2
mul ax, bx
```

```
mov ax, 4
mov bx, [ax]//@ indexé
mov ax,
add  ax, bx
```

Un code relogeable (2)

- **Un code est relogeable si l'on peut le charger à un emplacement quelconque en mémoire avant le début de son exécution**
- **Solution pour un code relogeable :**
  - Usage systématique d'adressage indexé
  - Créer l'illusion d'espace d'adressage séparés pour chaque application
- **Mémoire virtuelle : mécanisme permettant de créer un niveau d'indirection entre une adresse et le contenu mémoire**
  - Adresse physique = Traduction (adresse virtuelle)

Adressage Virtuel

- **Limites de l'Adressage physique :**
  - 1 adresse = 1 cellule de stockage (sauf exception)
    => espace d'adresse limité par le nombre de cellules mémoire
  - 1 fonction d'adressage => construit l'adresse physique :
    - De la prochaine instruction
    - Des opérandes résidant en mémoire
- **Objectifs des Adresses virtuelles :**
  1. Permettre d'avoir un espace d'adressage plus grand
  2. Partager efficacement la mémoire
  3. Gérer des contraintes de contrôle d'accès
- **Idée : une donnée dans un programme possède une adresse indépendamment de son stockage (mémoire, disque)**

Espace d'adressage virtuel

- **Espace d'adressage virtuel :**
  Intervalle d'adresse permettant de référencé des données indépendamment de leur stockage effectif

- **Adresse Virtuelle dans la pratique**
  - Adresses dans $[0,2^N]$ avec N très grand
  - Découplage entre adresse et présence en mémoire (hypothèse, si absent de la mémoire => présent sur le disque)
  - Adresse dans le « code » = mode d'adressage dans l'espace virtuel

- **Pb : Comment indiquer au processeur où trouver l'information ?**
  - Des mécanismes matériels pour traduire l'adresse virtuelle
  - Des données permettant de conserver l'information sur ce qui est en mémoire

La segmentation dans les processeurs

- **Idée découper la mémoire en N segments de taille fixe (2^x), décomposer l'adresse en 2 composante :**
  - Le segment (identifie le tronçon
  - Le décalage
- **Traduction de l'adresse :**
  - Segment stocké dans un registre
  - Multiplication par 2^x puis addition avec le « décalage »

- **Exemple : x86, CS registre pour le segment de code => calcul de l'adresse d'un saut**
- **Problème toujours autant d'adresses que de mémoire**

Organisation en Pages

- **Espace virtuel décomposé en $M/2^k$ pages,**
  (k usuellement vaut 10, 11 ou 12 pour des pages de 1, 2 ou 4 kilo octets)
- **Espace d'adressage physique**
  - Découpé en page
  - Chaque page physique contient une page virtuelle
  - Données en mémoire pour faire la traduction @Virt => @ Physique
- **La page = unité d'association espace physique / virtuel**

⇒ **Besoin de stocker ces correspondances,**
+ la page est petite, + il y a de correspondances

- **Comment stocker cette association ?**

Adresse virtuelle, la table de page simple

- **Espace d'adressage = N pages => tableau à N entrées**
- **Chaque entrée = 1 page virtuelle**
  - N° de page = index dans la table => facile de retrouver l'entrée
  - Contenu de l'entrée :
    - l'adresse du début de page physique où la page est présente (si elle est en mémoire)
    - Information sur les contraintes d'accès (lecture / écriture)
    - La table est stockée en mémoire
- **Sur le processeur**
  - 1 registre pour retrouver la table des pages PTBR (Page Table Base Register)
  - 1 bloc de traduction
    1. Consulte la table de page pour trouver le numéro de page physique
    2. Si présent Construit l'adresse physique en additionnant NumPage*2^k+décalage dans la page

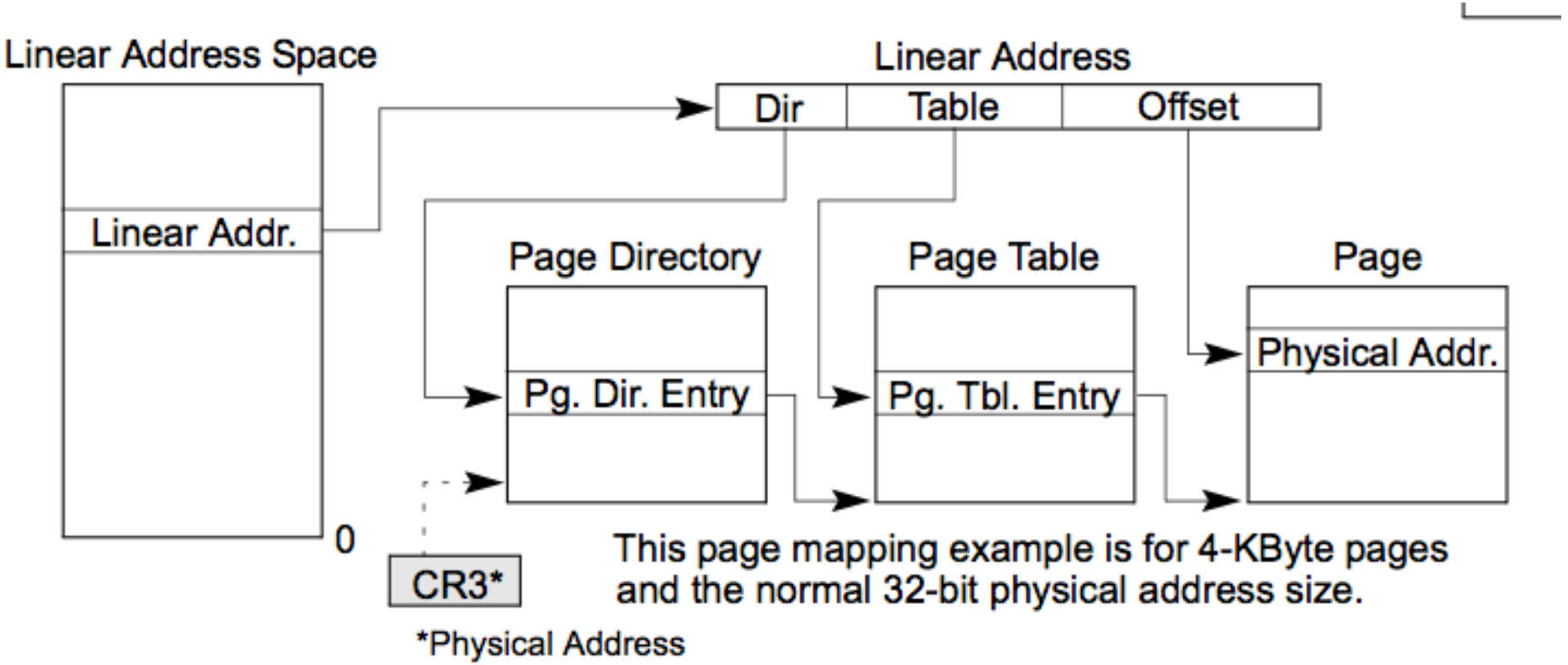
    2bis. Si absent branchement / interruption pour charger la page

Multi programmation et pages de table multi niveaux

- **Pb : multiprogrammation = Partager les ressources du processeur entre P programmes**
- **Pour la mémoire => P espaces virtuels => P pages tables**
- **Si les espaces virtuels sont grands (N taille de la table) => P tables de taille N = risque d'explosion**
- **Application numérique :**
  - Adressage virtuel sur 32 bits pour des pages de 4ko => 2^{20} pages pour P processus (sur Unix se compte en milliers)

  $\Rightarrow$ 2^{32} entrées à stocker en mémoire (saturation)
- **Solution : Traduction en plusieurs étapes**

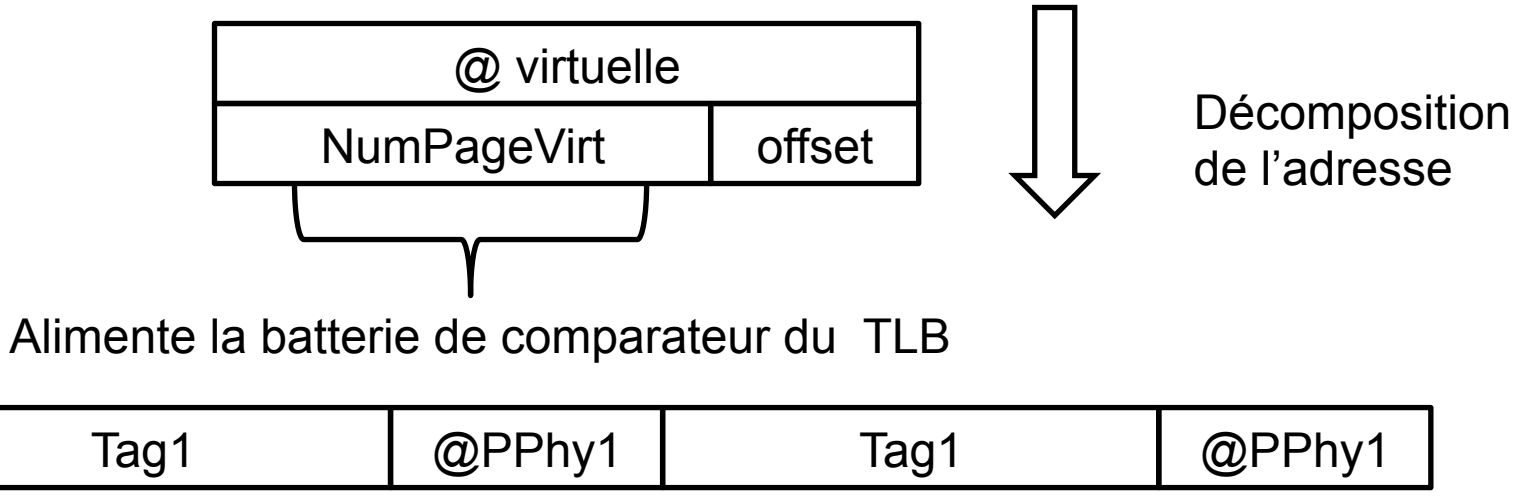
L'exemple du IA 32



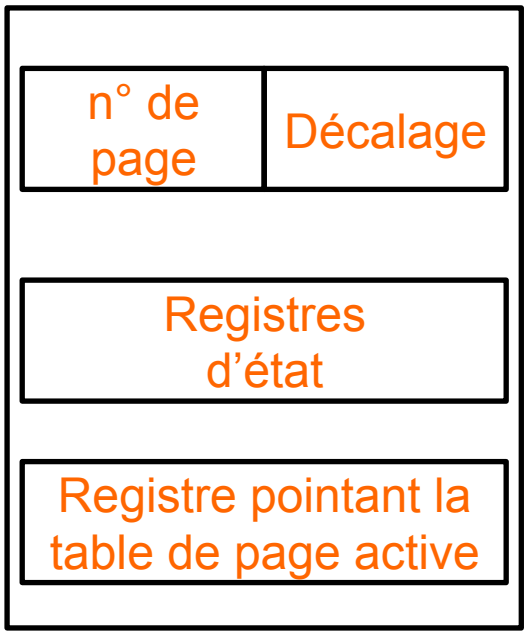
Hiérarchisation de la table de page => stockage dans l'espace virtuel
+ chargement en mémoire des pages contenant les tables utiles

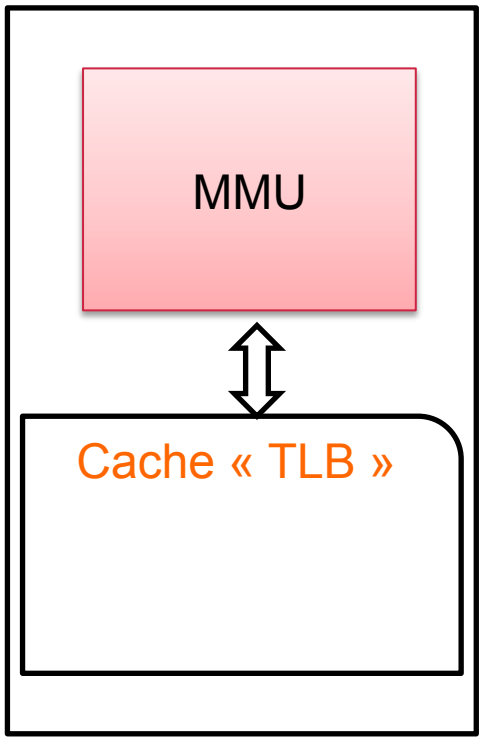
Le mécanisme de TLB : cache de traduction

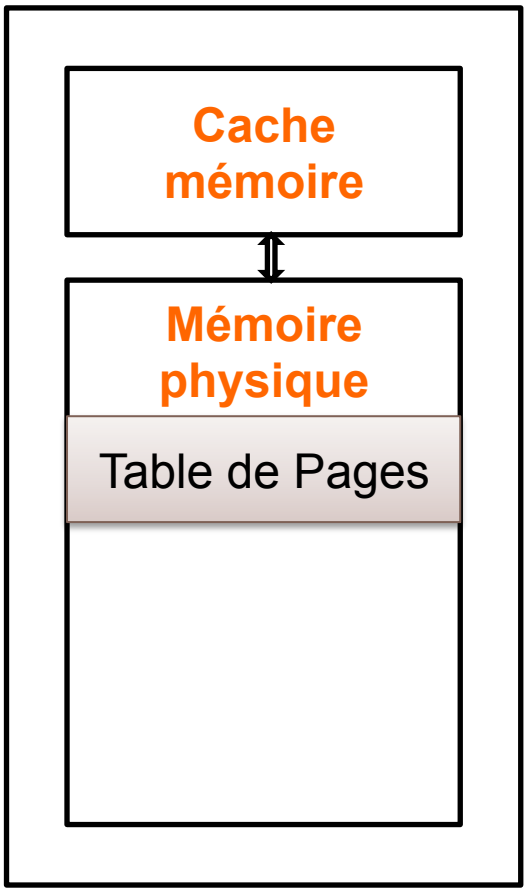
- **TLB : Translation Look-aside Buffer**
  - Cache pour stocker le résultat de la traduction
  - Décomposition du numéro de Page pour construire un cache associatif (complet ou par bloc)
- **Exemple d'un cache associatif complet**



Un accès de A à Z







1) Interroger le TLB puis la tables des pages pour l'entrée identifiée par n ° page dans la table courante
2) Vérifier la validité de l'accès
3) Construire l'@ Physique