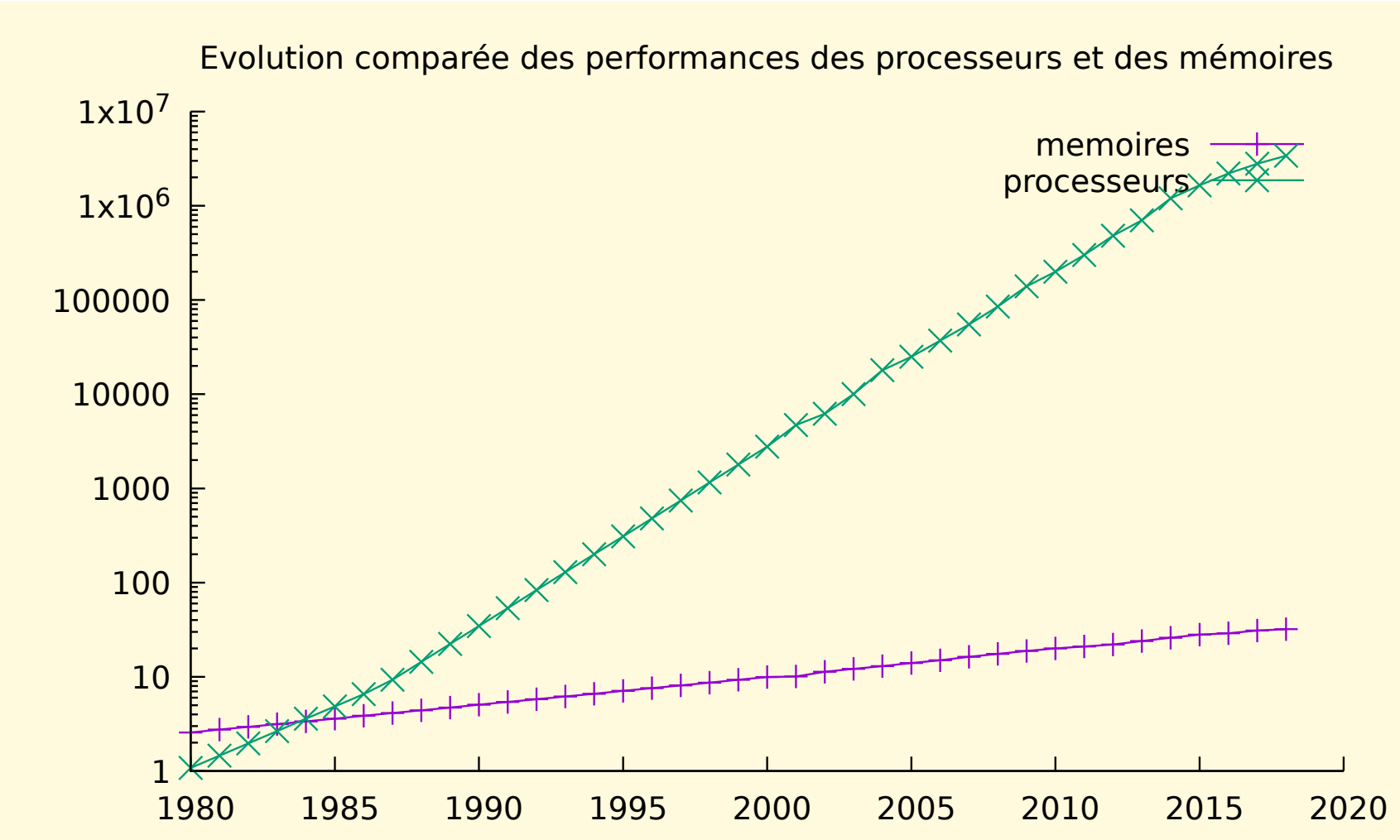
Hiérarchie mémoire

Alain MÉRIGOT

Université Paris Saclay

Position du problème

Evolution technologique comparée des processeurs et des mémoires



Principes de localité

– **Localité temporelle**

Si une information (instruction ou donnée) est manipulée par le processeur, il est très probable qu'elle sera également utilisée peu de temps après

– **Localité spatiale**

Si une information est manipulée par le processeur, il est très probable qu'une information située dans une emplacement mémoire proche sera également utilisée peu de temps après

La hiérarchie mémoire

Différents niveaux hiérarchiques :

– Processeur

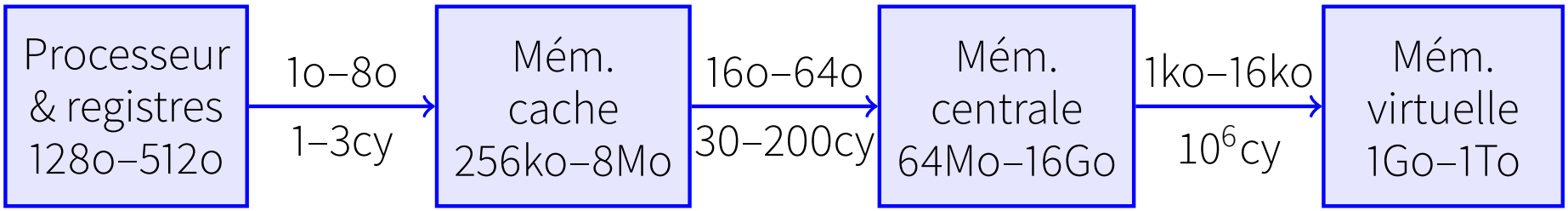
– Mémoire *cache*

– Mémoire centrale

– Mémoire vituelle

Echange entre cache et mémoire centrale : *lignes de cache*

Echange entre mémoire centrale et mémoire virtuelle : *pages*

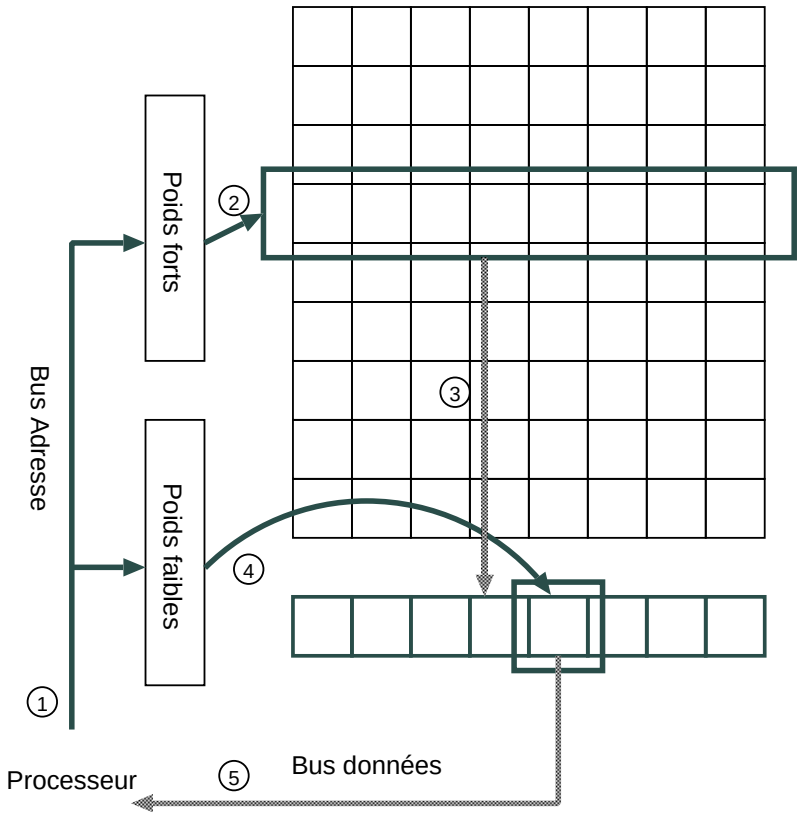


La mémoire centrale

Organisée comme une matrice de points mémoire élémentaires.
La lecture d'un mot mémoire procède en plusieurs étapes:

1. envoi de l'adresse par le processeur
2. sélection d'une ligne avec la partie haute de l'adresse
3. lecture de la ligne
4. sélection du bit (ou mot) dans la ligne avec la partie basse de l'adresse
5. envoi de l'information au processeur

La phase 3 est de très loin la plus coûteuse



La mémoire centrale (cont.)

Améliorations des accès mémoire par accès à plusieurs mots contigus

- Mémoire élargie : utilisation de plusieurs boitiers lus simultanément et envoyés en parallèle sur un bus élargi
- Mémoire entrelacée : utilisation de plusieurs boitiers lus simultanément et envoyés séquentiellement
- Accès en rafale (ou *burst mode*) : boitier mémoire pouvant envoyer séquentiellement n mots en réponse à une demande de lecture. La quasi totalité des boitiers mémoire ont actuellement cette fonctionnalité

Les mémoires actuelles (DDR SDRAM) possèdent également une sorte de *cache* interne mémorisant la ligne récemment lue.

Les mémoires *cache*

Stockent une partie des données de la mémoire centrale

Organisée en *lignes de cache*

Lors d'un accès mémoire :

1. On cherche si la donnée est présente dans le cache
2. Si oui, on envoie la donnée au processeur.
   Temps t_a (temps d'accès réussi).
3. Si non
   1. On lit la ligne contenant la donnée en mémoire et on la copie dans le cache.
   2. on envoie la donnée au processeur au processeur.

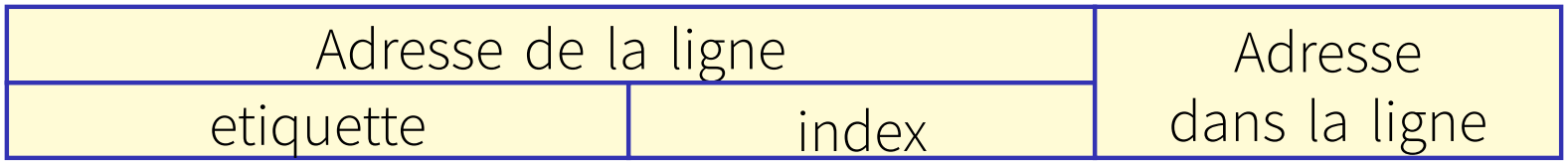
   Temps t_e temps d'échec
   Probabilité d'un échec τ_e.

Ou stocker une ligne dans le cache ?

Plusieurs politiques :

- En un emplacement unique dépendant de son adresse : **cache à correspondance directe**.
  Par exemple on prendra les poids faibles de l'adresse pour déterminer l'emplacement dans le cache
- En un emplacement quelconque : **cache associatif**
  Evite que des lignes avec les mêmes poids faibles d'adresse ne s'*écrasent* mutuellement, mais nécessite une recherche complexe
- En un emplacement quelconque dans un ensemble déterminé par l'adresse : **cache associatif par ensemble**.
  Bon compromis, les ensembles font quelques lignes (4, 8...)

Comment repérer si une ligne est présente dans le cache ?



On distingue différent champs dans une adresse de donnée:

déplacement l'adresse de l'octet dans la ligne

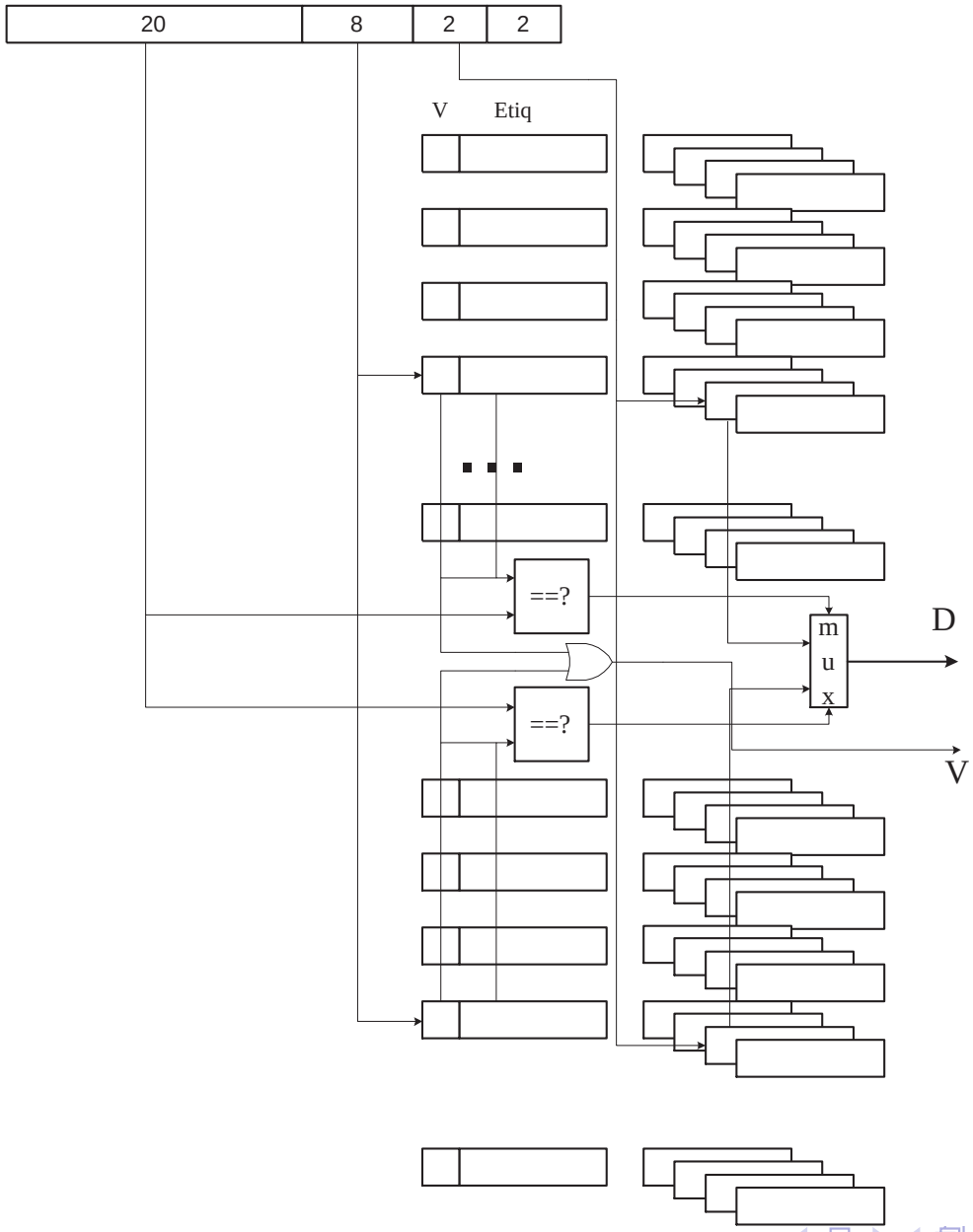
index la partie servant à déterminer la place de la ligne dans le cache (pour les caches à correspondance directe ou associatifs par ensemble)

étiquette Les poids forts qui servent à identifier la ligne de cache

Exemple : cache de 1024 ensembles contenant 4 lignes de 4 mots

- adresse de l'octet dans le mot (2b) + adresse mot dans la ligne (2b) : deplacement 4 bits
- index : 10 bits (repère un ensemble parmi 1024)
- étiquette : $32 - 10 - 4 = 18$ bits

Exemple de cache associatif : 256 ensembles de 2 lignes



Comment remplacer une ligne ?

Si on insère une nouvelle ligne dans le cache, il faut en supprimer une.

Si le cache est (partiellement) associatif, comment choisir la ligne à supprimer ?

Différentes politiques :

- aléatoirement
- la plus ancienne
- la moins récemment utilisée

Cas des écritures

Deux politiques dans le cas d'une écriture :

- **Cache à écriture simultanée (*write-through*)** On écrit simultanément dans le cache et la mémoire pour assurer la cohérence des information.
  Problème : encombrement du bus
- **Cache à réécriture (*write-back*)** On récrit la ligne en mémoire au moment où elle est supprimée du cache.
  Plus complexe, mais charge moins le bus.

Cache instruction et données

On peut avoir ou non les instructions et les données dans la même cache.

- **Caches séparés :** un cache pour les instructions et un pour les données.
  Permet d'éviter les conflits d'accès à la mémoire dans le processeur
- **Caches unifiés :** un cache unique stockant instructions et données
  Permet une meilleure utilisation de la capacité du cache

Amélioration des caches

Idéalement, il faudrait réduire t_a, t_e et τ_e.
Conduit à des exigences la plupart du temps contradictoires.

Réduction du taux d'échec τ_e

- Taux d'échec lors de la première utilisation d'une donnée
  - augmenter la taille des lignes
  - anticiper la demande (*prefetch*)
- Taux d'échec du à une capacité trop faible: augmenter la taille
- Taux d'échec du à des conflits : augmenter l'associativité

Réduction du temps d'accès Utiliser des caches simples (associativité faible, écriture simultanée)

Réduction du temps d'échec

- Caches non bloquant :
  - exécution des requètes dans le désordre
  - priorité aux lectures sur les écritures
- Caches à plusieurs niveaux Deux voire trois niveaux sont courants

Amélioration des caches (cont.)

On utilise *plusieurs niveau de caches* successifs pour permettre de respecter les exigences simultanées (et incompatibles) de réduction du temps d'accès t_a, du temps d'échec t_e et du taux d'échec τ_e.

le cache de niveau 1 (L1) est le plus proche du processeur. Il doit fournir des données en 1 cycle et pour cela être le plus simple possible, quitte à avoir un fort taux d'échec. Il est toujours dans le circuit du processeur.

les caches de niveau supérieur (L2 et souvent L3) doivent limiter les accès mémoire. Il sont de plus grande taille et plus sophistiqués, mais leur temps d'accès est plus important (2–4 cy typ.) Le cache L2 est actuellement intégré au processeur, alors que L3 peut être externe ou intégré.

Caches à plusieurs niveaux

	Cache L1	Cache L2
Objectifs	Cache rapide pour alimenter le processeur (sur le même circuit)	Réduire le temps d'échec du cache L1 et limiter au maximum les accès à la mémoire
Type de cache	deux caches séparés pour données et instructions	unifié instructions/données
Temps accès	1 cy	2-3 cy
Temps d'échec	2-3 cy	30-200 cy
Taille	2×16–32 ko	256 ko (qq Mo pour L3)
Associativité	Corr. directe ou faible associativité (pour $\searrow t_a$)	Associatif par ensemble (2-8 lignes/ens.)
Écriture	Écriture simultanée (pour simplifier le cache)	Réécriture (pour limiter le trafic mémoire)

Evaluation de performances

Il est nécessaire de rendre en compte les caches dans les évaluation de performances.

$$t_{prog} = n_i \times T_c \times (CPI + MPI \times \tau_e \times t_e) \quad (1)$$

CPI : nombre de cycles par instruction en moyenne
MPI : nombres d'accès mémoire par instruction en moyenne
τ_e taux d'échec du cache L1
t_e temps d'échec (en cycles) du cache L1
Dans les cas des caches à plusieurs niveaux, t_e prend en compte les différentes caractéristiques des caches :

$$t_e = t_{eL1} + t_{eL2} \times \tau_{eL2}$$

Mémoire virtuelle

Dans la plupart des systèmes modernes, on peut utiliser une partie de la mémoire de masse (disque) pour étendre la mémoire centrale. **mémoire virtuelle**

En fonction des besoins, les informations sont transférées au disque ou recopiées en mémoire centrale par *pages* (typ. 1-4 ko).

Vus les temps **très** importants d'accès à un disque, on cherche à limiter le taux d'échec (*associativité totale*).

Un programme et ses données ne sont donc pas toujours au même emplacement en mémoire centrale suivant la place disponible.

Par contre, ils gardent toujours les mémes adresses vues du processeur (*adresses virtuelles*), alors que les emplacement en mémoire ont changés (*adresses physiques*).

Mémoire virtuelle (cont.)

Le système d'exploitation crée une correspondance entre les adresses virtuelles et les adresses physiques dans une *table des pages*.

Ces tables peuvent être de grande taille (qq Mo) et résident en mémoire.

Tous les accès mémoire ont besoin de faire une traduction entre les espaces virtuels et physiques avec la table des pages.

Pour accélérer cette traduction, un mécanisme du processeur permet de faire les correspondances entre adresses virtuelles et adresses physiques *avant* l'accès au cache.

Il s'agit d'une petite table qui mémorise les dernières traductions *Translation Lookahead buffer TLB*