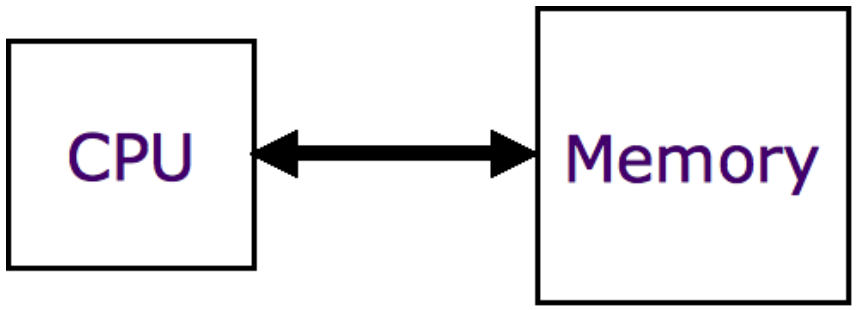
7- Mémoire

a- *Cellules mémoires (SRAM, DRAM)*

b- Problème de la mémoire (et comment s'en sortir)

c- Caches

d- Mémoire virtuelle

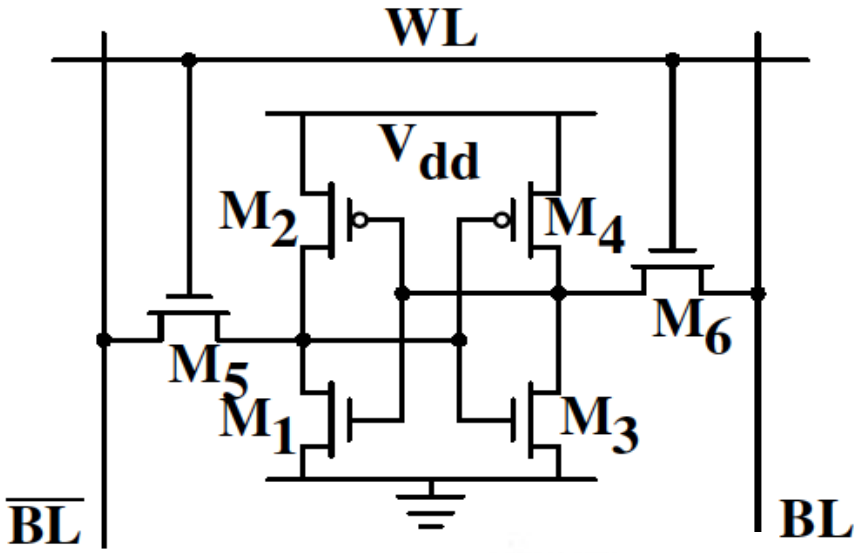


7-a Cellules mémoire: SRAM

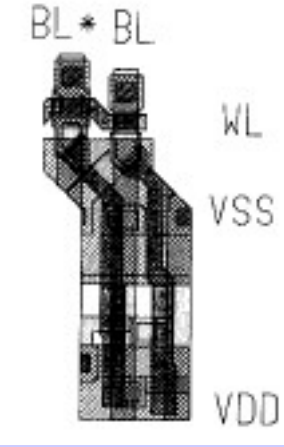
Caractéristiques Static RAM

- - 6 transistors, **prend de la place**
- + Lecture du signal quasi-instantanée sur BL en appliquant tension sur WL (signal carré)
- + Signal stable, pas besoin de rafraichissement
- - Etat (0 ou 1) maintenu par **alimentation constante** (Vdd)

Schéma



Gravure

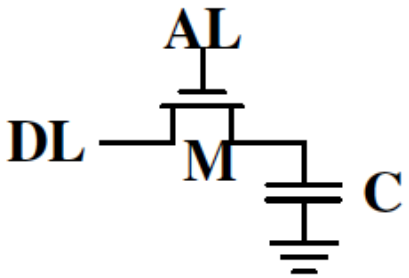


7-a Cellules mémoire: DRAM

Caractéristiques Dynamic RAM

- \+ 1 transistor, 1 capacité, peu de place
- \- Lecture/écriture en appliquant tension sur AL. **Signal pas carré, lecture/écriture pas immédiat**
- \- Lecture (et fuites) décharge la capacité, **nécessité rafraichissement**
- **\- Signal très faible.** Necessite amplificateur
- \+ Pas d'alimentation permanente

Schéma



Gravure (même échelle SRAM)

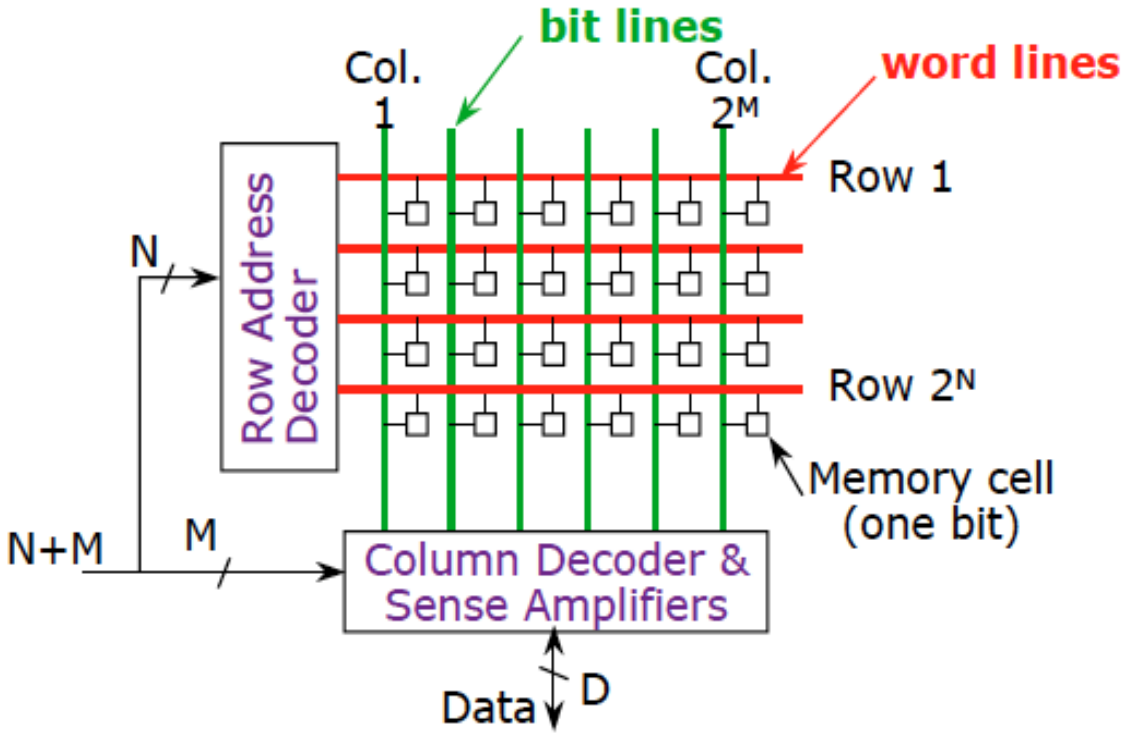
7-a Cellules mémoire

Tableau de cellules

Adresse sur N+M bits

Etapes:

- Signal RAS (row address selector) selectionne la ligne
- Signal CAS (column address selection) lit sur la colonne
- Amplification du signal (sert aussi à rafraichir signal)



1 cellule = 1 bit

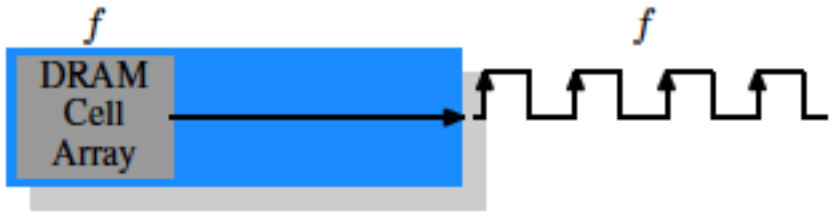
- Lecture en parallèle de plusieurs cellules/tableaux

7-a Cellules mémoire

Types de mémoire

Single Data Rate (SDRAM)

- 1 bit/cycle,
- 1 transfert/cycle,
- f=100Mhz => 100Mb/s



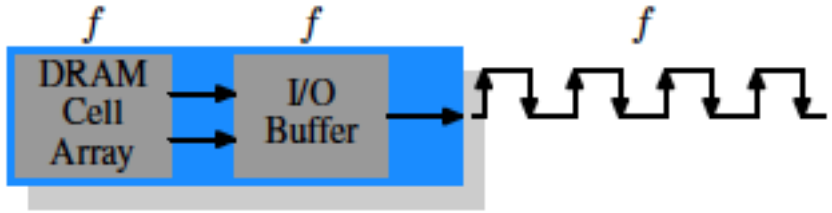
64bits/cellule: 800Mo/s

Double Data Rate (DDR SDRAM)

- 2 bits/cycle, bufferisé

! consommation: dépend de f !

- 2 transferts/cycle (front montant et descendant de l'horloge)
- f=100Mhz=>200Mb/s par cellule



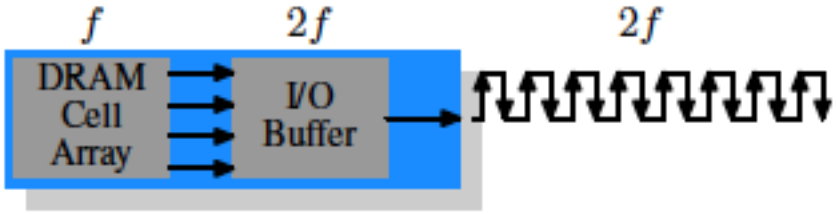
64bits/cellule: 1600Mo/s

7-a Cellules mémoire

Types de mémoire

DDR2 SDRAM

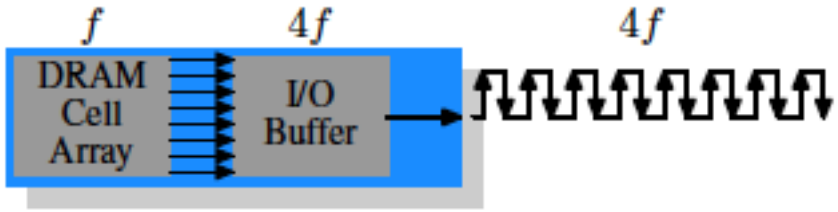
- 4 accès/cycle, bufferisé
- Fréquence x2 de transfert, sur les deux fronts
- f=200Mhz => 800Mb/s



64bits/cellule: 6400Mo/s

DDR3 SDRAM

- 8 accès/cycle, bufferisé
- Fréquence x4 de transfert, surles 2 fronts
- f=233Mhz => 1864Mb/s

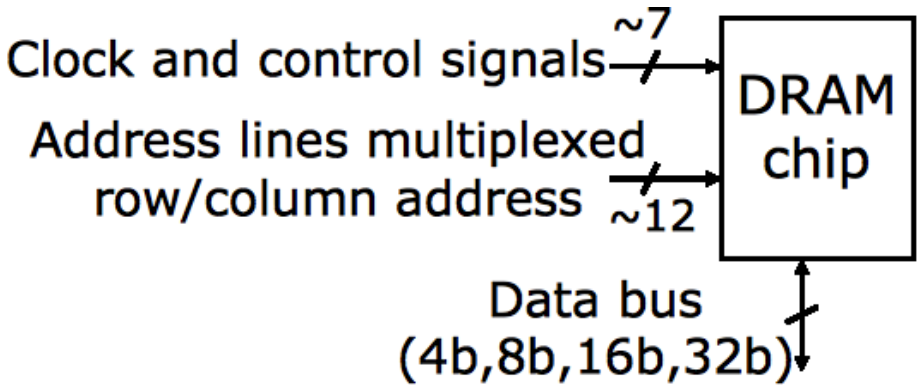


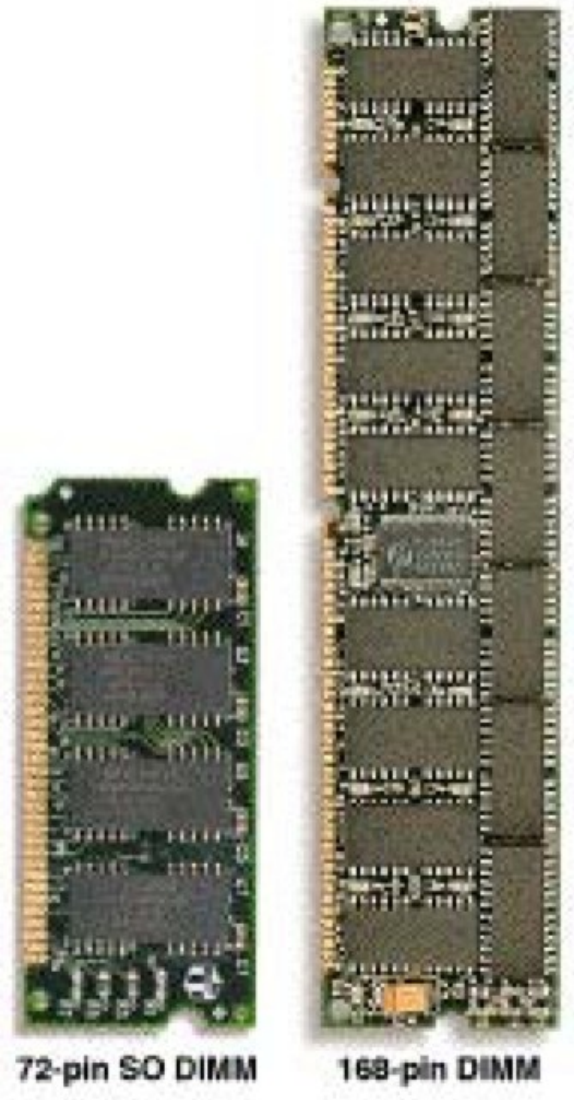
64bits/cellule: 14900Mo/s

7-a Cellules mémoires

Packaging

- DIMM (Dual Inline Memory Module): plusieurs tableaux de cellules connectés en parallèle, avec buffers
- Plusieurs chips nécessaires pour fournir les données (64 bits = 16x4 bits par exemple)





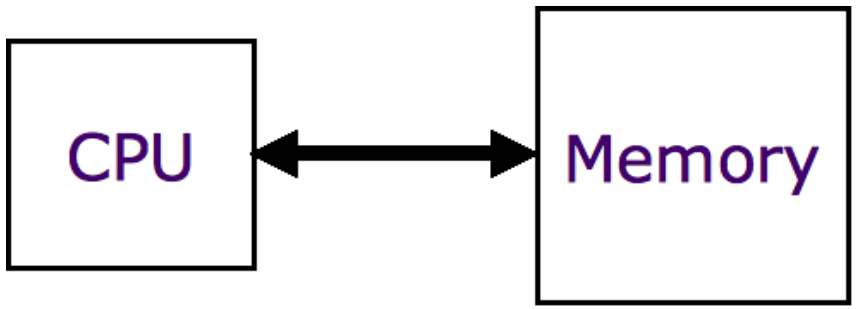
7- Mémoire

a- Cellules mémoires (SRAM, DRAM)

b- *Problème de la mémoire (et comment s'en sortir)*

c- Caches

d- Mémoire virtuelle



7-b Problème de la mémoire

Idéalement:

- Mémoire très grande, temps d'accès = 1 cycle, débit illimité

7-b Problème de la mémoire

Idéalement:

- Mémoire très grande, temps d'accès = 1 cycle, débit illimité.

En réalité, limitée par:

- **Latence**: temps pour un accès à la mémoire.

Latence >> cycle CPU

- **Bande passante**: nombre d'accès par unité de temps

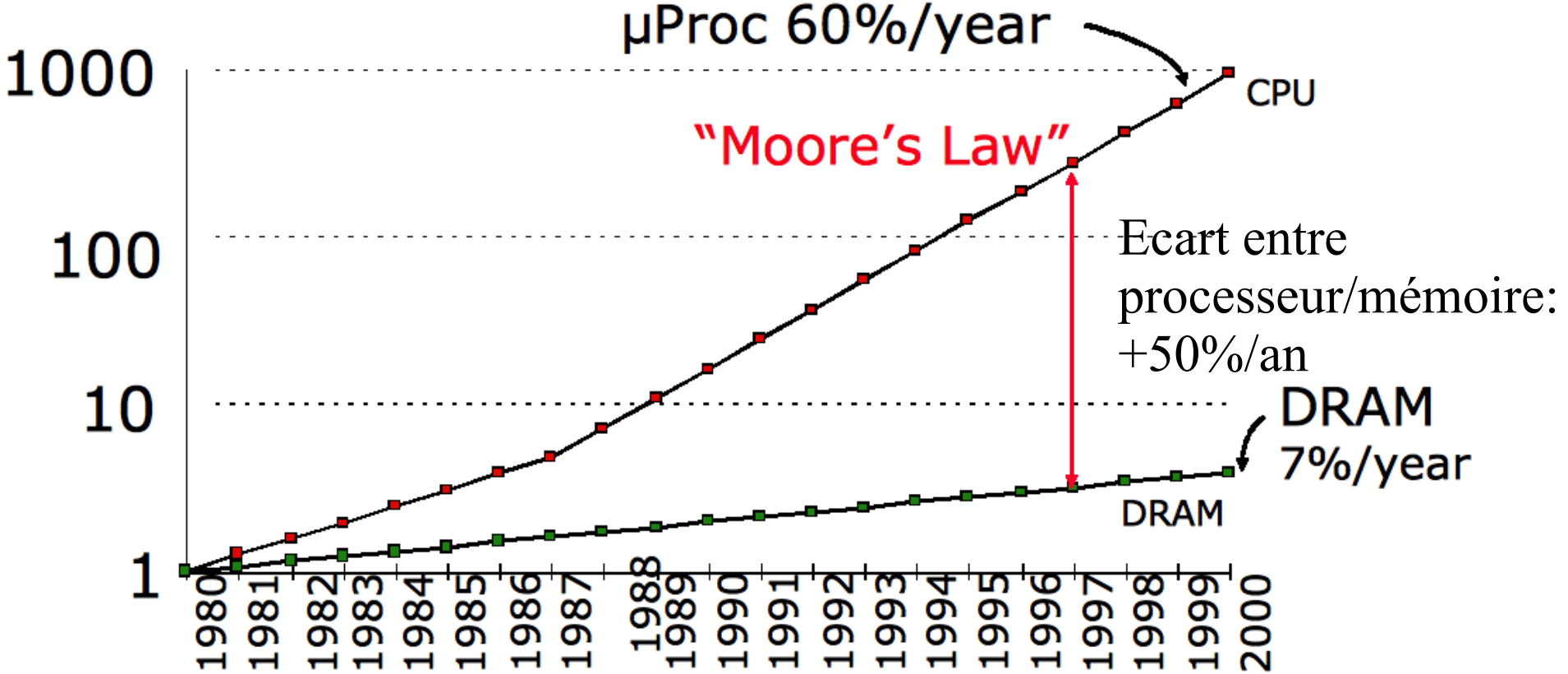
Si une fraction *m* d'instructions accède à la mémoire, il faut

1+*m* références mémoire / instruction

(chaque instruction est aussi en mémoire)

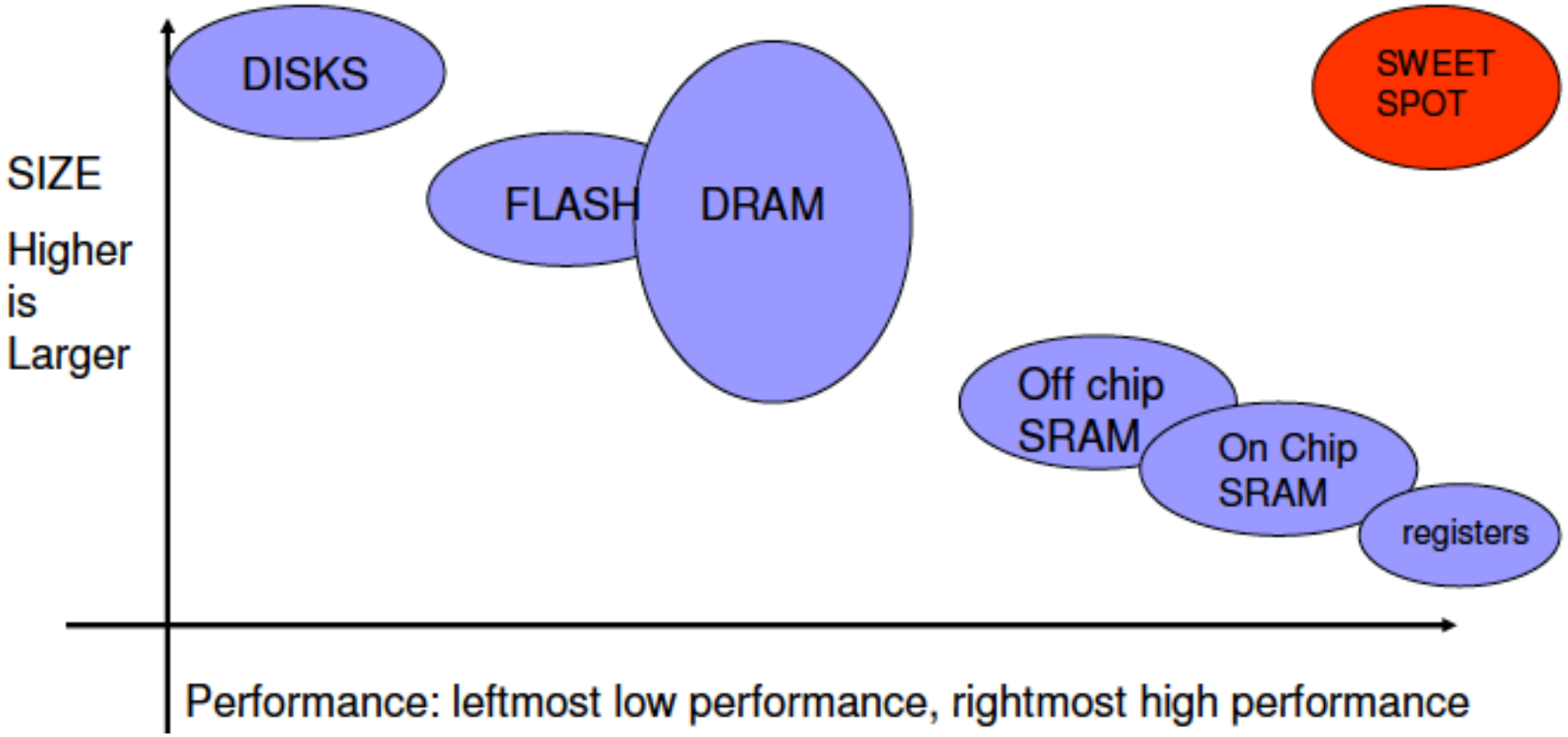
1 cycle par instruction => *1*+*m* référence mémoire par cycle

7-b Problème de la mémoire



- Pour un processeur superscalaire à 2Ghz 4 instructions/cycle, une DRAM à 100ns l'accès => 800 instructions pour un accès !

7-b Technologie disponible



7-b Technologie disponible

Type	Taille	Latence	Prix
Registre	< 1Ko	<1ns	€€€€
SRAM sur le processeur	8Ko-8Mo	<10ns	€€€
SRAM hors processeur	1Mo-16Mo	<20ns	€€
DRAM	64Mo-1To	<100ns	€
Flash	64Mo-32Go	<100us	c
Disque	40Go-1Po	<20ms	~0

7-b Comment s'en sortir ?

Accès mémoire suivent souvent une régularité prédictible

Localité temporelle: lorsqu'une cellule mémoire est accédée (en lecture ou écriture), un nouvel accès à cette cellule peu de temps après

Localité spatiale: lorsqu'une cellule mémoire est accédée (en lecture ou écriture), un accès à une **cellule mémoire proche** a lieu peu de temps après.

Exemple de localité temporelle: les instructions dans une boucle sont accédées plusieurs fois

Exemple de localité spatiale: les instructions dans une séquence sont accédées les uns après les autres.

7-b Futur proche

Dans une boucle

```
    for (i=0; i<N; i++) {
S1        … = A[i]
S2        … = A[i + K]
    }
```

Combien d'itérations séparent deux utilisations successives d'un même élément de tableau ? **K**

Idée principale

- Garder A[i] lu en S2 dans une mémoire rapide jusqu'à sa prochaine réutilisation par S1 K itérations plus tard.
- Si stocké dans registres, nécessaire au moins **K** registres

7-b Cellule mémoire proche

Dans une boucle

```
for (i=0; i<1000; i++) {
  for (j=0; j<1000; j++) {
      … A[i*1000+j] …
      … B[j*1000+i]  ...
} }
```

Localité spatiale pour A: très bonne, les cellules mémoires accédées consécutivement

Localité spatiale pour B: mauvaise, les cellules accédées consécutivement sont séparées par 1000 éléments !

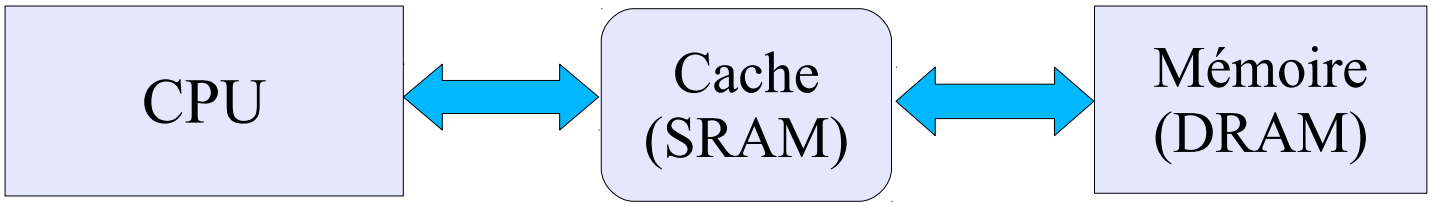
7-b Localité spatiale et temporelle

Donald J. Hatfield, Jeanette Gerald: Program Restructuring for Virtual Memory. IBM Systems Journal 10(3): 168-192 (1971)

7-b Hiérarchie mémoire

Stratégie:

- Réduire la latence moyenne en utilisant une mémoire rapide et plus petite, appelée **cache**
- Utiliser la localité temporelle et stocker les éléments recemment accédés dans le cache
- Utiliser la localité spatiale pour stocker les éléments proches de ceux déjà accédés



7-b Hiérarchie mémoire

A chaque accès mémoire:

- Vérifie si l'élément est dans le cache: si oui, c'est un **cache hit** (faible latence),
- Sinon c'est un **cache miss** (grosse latence), le chercher en mémoire. Le placer dans le cache ainsi que ses voisins.

Gain attendu:

T1 latence pour un hit, *T2* latence pour un miss (et accès mémoire)

h: fraction des références mémoires faisant un hit (hit ratio)

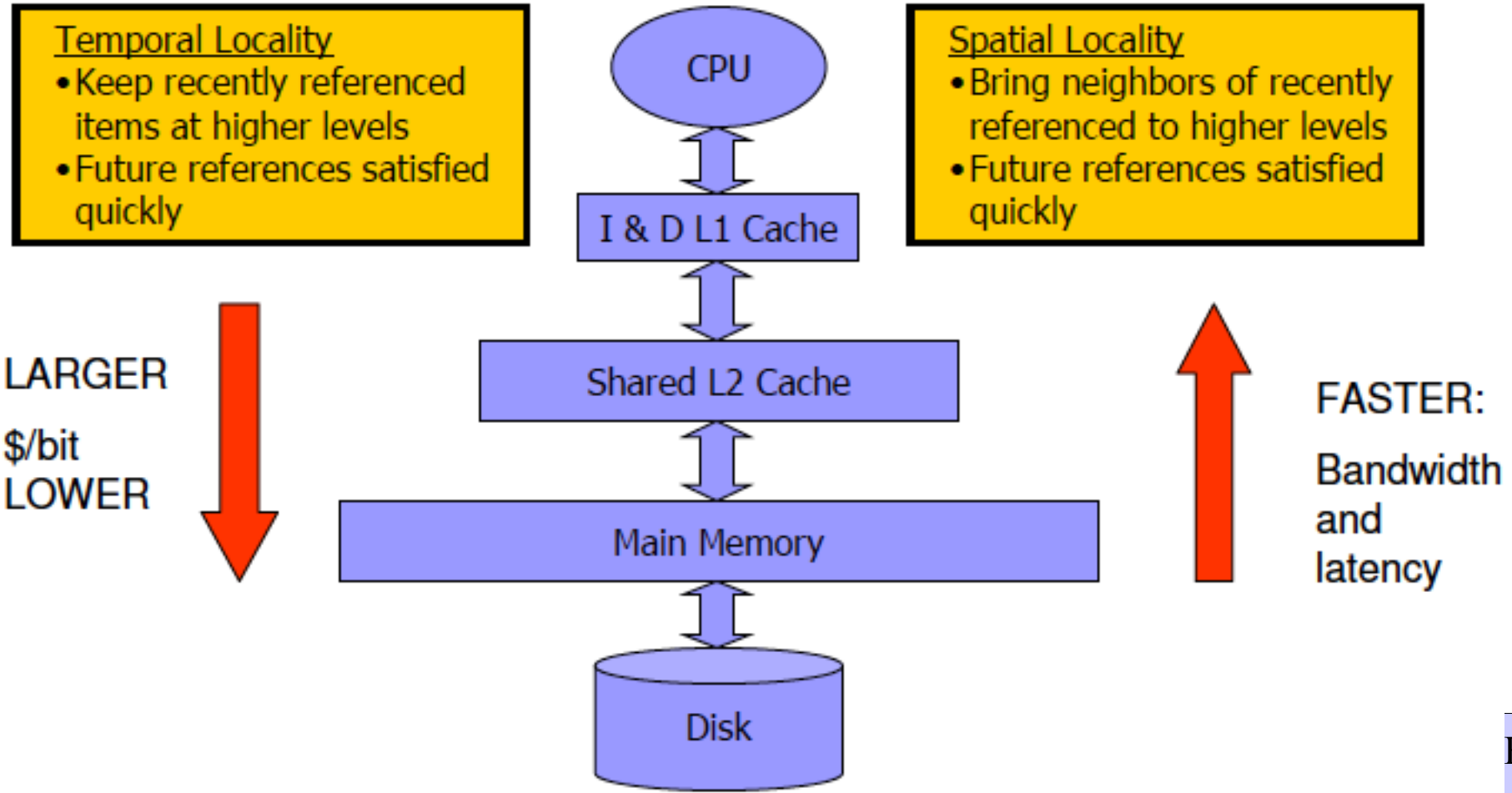
Latence moyenne = *h.T1 + (1 - h). T2*

Si *h=0,5,* au mieux latence moyenne = ½ T2... Gain d'un facteur 2 seulement, quelque soit la vitesse du cache !!

=> Besoin d'avoir h proche de 1

7-b Hiérarchie mémoire

Généralisation à plusieurs niveaux de cache



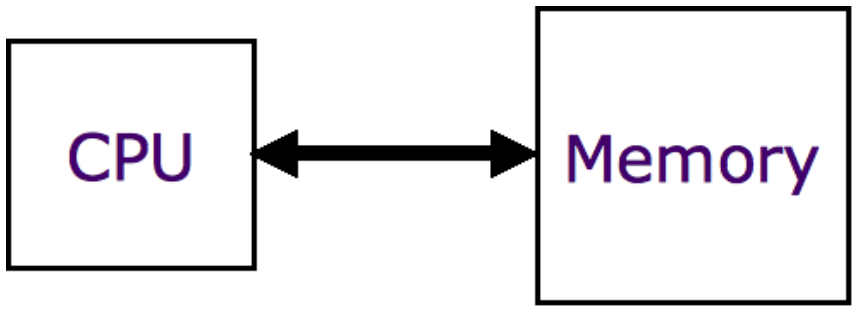
7- Mémoire

a- Cellules mémoires (SRAM, DRAM)

b- Problème de la mémoire (et comment s'en sortir)

c- *Caches*

d- Mémoire virtuelle



7-c Fonctionnement du cache

Points clés sur le fonctionnement

- **Identification:** comment on trouve une donnée dans le cache ?
- **Placement:** où sont placées les données dans le cache ?
- **Politique de remplacement:** comment faire de la place pour de nouvelles données ?
- **Politique d'écriture:** comment propager les changements de données ?
- Quelles autres stratégies pour améliorer le hit ratio ?

7-c Structure du cache et identification

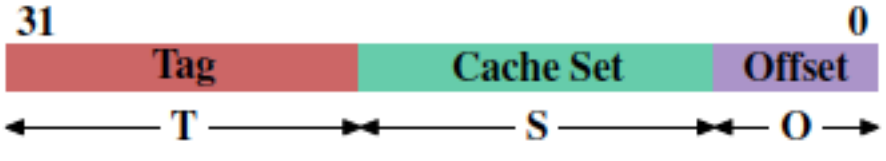
Ligne de cache: bloc de données correspondant à un bloc de données consécutives en mémoire (par ex., 128 octets = 1 ligne)

Les lignes de cache sont organisées en **ensembles de lignes (sets)**

Associativité: nombre de lignes d'un set (identique pour tous les sets)

Correspondance adresse mémoire et place dans le cache:

- Une partie est gardée, l'"adress tag", pour pouvoir l'identifier
- Une partie de l'adresse va correspondre au numéro de set
- La dernière partie de l'adresse correspond à la position de la donnée dans la ligne de cache



4-a Structure du cache et identification

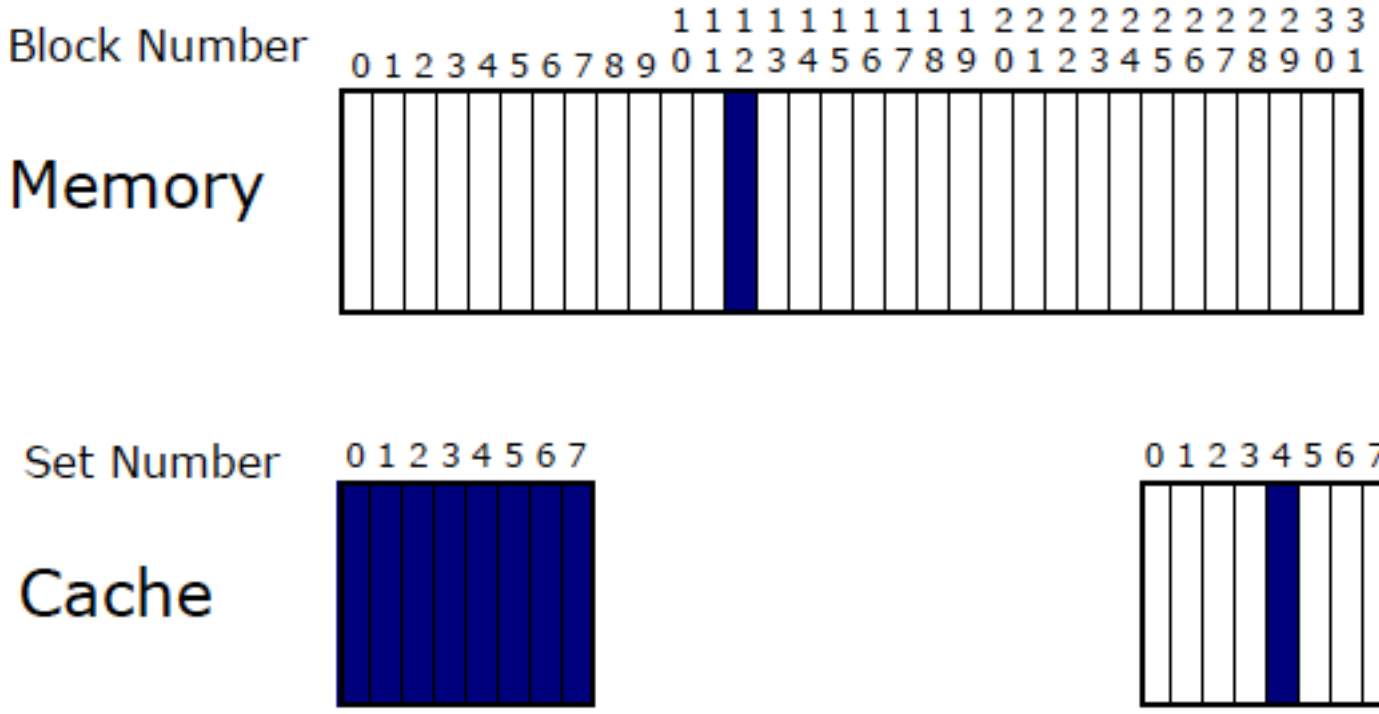
Donnée à l'adresse

01001	01	011

Set	Offset 000	Offset 001	Offset 010	Offset 011	Offset 100	Offset 101	Offset 110	Offset 110	Tag
00									
01				data					01001
10									
11									

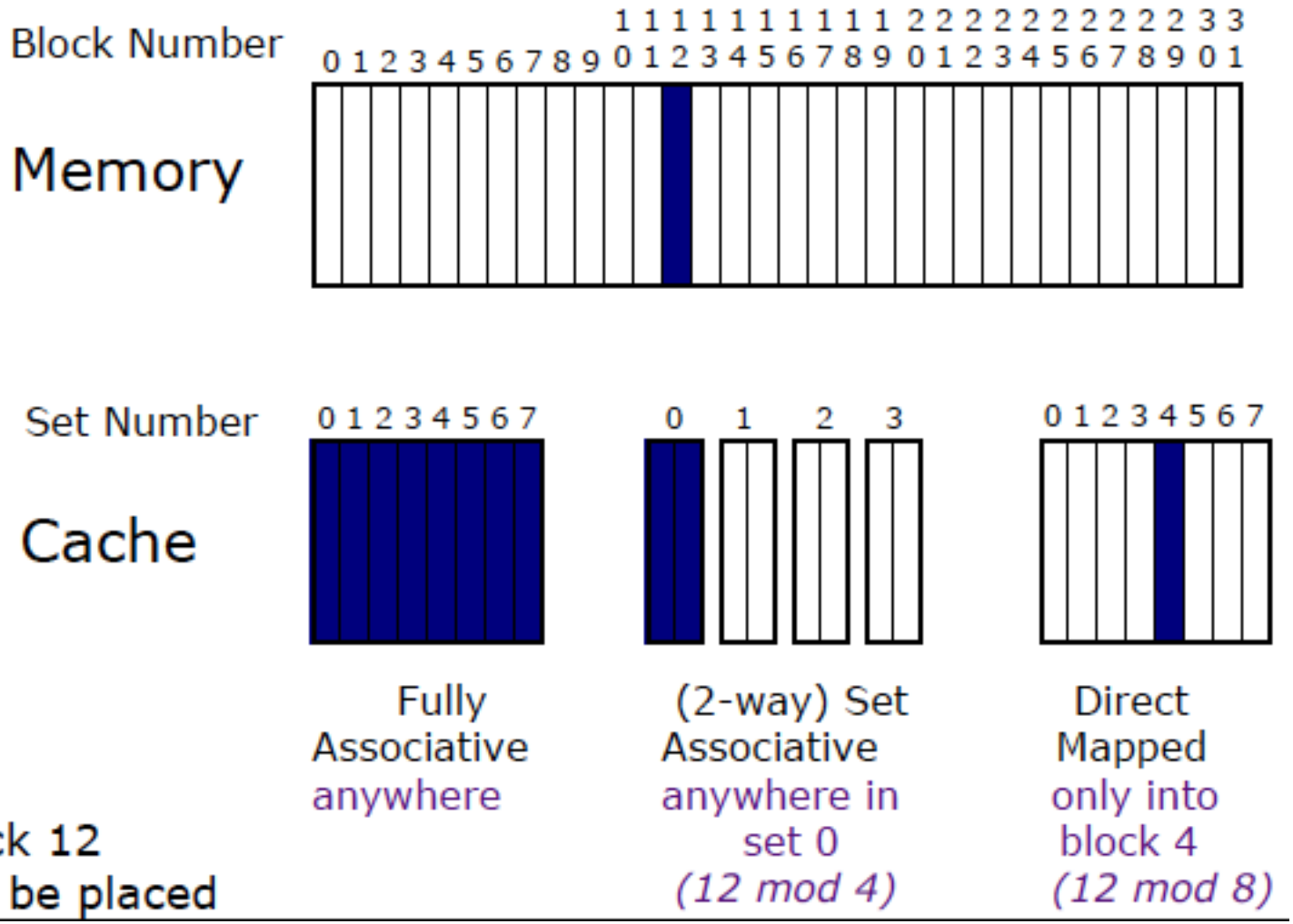
7-c Placement

Où placer dans le cache la donnée venant de l'adresse 12 ?



block 12
can be placed

7-c Placement



7-c Placement

Associativité

- Une donnée peut être placée
  - Dans n'importe quelle ligne de cache d'un set
  - Et dans un seul set
- Cas extrêmes: complètement associatif (un set) ou direct map (un set=une ligne de cache).

La donnée contenue à une adresse est-elle dans le cache ?

- Calcul du set où devrait être la donnée, en fonction de l'adresse
- Pour chaque ligne de cache: (en parallèle)
  - Vérification du “tag” avec l'adresse cherchée

7-c Remplacement: direct map (1)

Exemple d'un cache direct map:

for (i=0; i<200; i++) A[i]=B[i];

→ Les accès sont alternés: B[0], A[0], B[1], A[1], ...

On considère

- A commence à l'adresse 2048, B à l'adresse 4096. A et B des tableaux d'entiers sur 64 bits (8 octets/élément)
- Un cache direct map,
  - Taille de 2048 octets, 64 lignes de cache de 32 octets (4 éléments de tableau/ligne)
- A[i] dans la ligne de cache (2048+8i)/32 modulo 64

→ A[0], A[1], A[2], A[3] et B[0],B[1],B[2],B[3] sont placés dans la même ligne de cache.

7-c Remplacement: direct map (1)

Cas défavorable d'accès

Accès à B[0]	**MISS**	Place B[0],B[1],B[2],B[3] dans la ligne de cache 0	
Accès à A[0]	**MISS**	Place A[0],A[1],A[2],A[3] dans la ligne de cache 0	Enlève B[0],B[1],B[2],B[3]

7-c Remplacement: direct map (1)

Cas défavorable

Accès à B[0]	**MISS**	Place B[0],B[1],B[2],B[3] dans la ligne de cache 0	
Accès à A[0]	**MISS**	Place A[0],A[1],A[2],A[3] dans la ligne de cache 0	Enlève B[0],B[1],B[2],B[3]
Accès à B[1]	**MISS**	(Re)place B[0],B[1],B[2],B[3] dans la ligne de cache 0	Enlève A[0],A[1],A[2],A[3]
Accès à A[1]	**MISS**	(Re)place A[0],A[1],A[2],A[3] dans la ligne de cache 0	Enlève B[0],B[1],B[2],B[3]
Accès à B[2]	**MISS**	(Re)place B[0],B[1],B[2],B[3] dans la ligne de cache 0	Enlève A[0],A[1],A[2],A[3]
Accès à A[2]	**MISS**	(Re)place A[0],A[1],A[2],A[3] dans la ligne de cache 0	Enlève B[0],B[1],B[2],B[3]
Accès à B[3]	**MISS**	(Re)place B[0],B[1],B[2],B[3] dans la ligne de cache 0	Enlève A[0],A[1],A[2],A[3]
Accès à A[3]	**MISS**	(Re)place A[0],A[1],A[2],A[3] dans la ligne de cache 0	Enlève B[0],B[1],B[2],B[3]

7-c Remplacement: direct map (2)

Cas favorable: réordonne les accès (vectorisation)

Accès à B[0]	**MISS**	Place B[0],B[1],B[2],B[3] dans la ligne de cache 0	
Accès à B[1]	**HIT**		
Accès à B[2]	**HIT**		
Accès à B[3]	**HIT**		
Accès à A[0]	**MISS**	Place A[0],A[1],A[2],A[3] dans la ligne de cache 0	Enlève B[0],B[1],B[2],B[3]
Accès à A[1]	**HIT**		
Accès à A[2]	**HIT**		
Accès à A[3]	**HIT**		

7-c Remplacement: caches associatifs

Quand un set est plein, quelle ligne du set remplacer ?

- Au hasard
- La donnée la moins récemment utilisée (Least Recently Used, LRU)
  - A chaque accès, on doit mettre à jour un indice de "fraicheur" de la donnée et des autres données du set
  - Utilisé sur caches avec petite associativité (2,4,8)
- Une ligne, à tour de rôle (méthode tourniquet).
  - Utilisé sur caches avec grande associativité

7-c Remplacement: caches associatifs

Exemple

for (i=0; i<200; i++) A[i] = B[i];

On considère;

- Un cache 2-associatif (associativité de valeur 2).
  - Taille de 2048 octets,
  - Lignes de cache de 32 octets (4 éléments de tableau par ligne)
  - 32 sets de 2 lignes chacun

A[i] est placé dans le set (2048+8i)/32 modulo 32

→ A[0],A[1],A[2],A[3] et B[0],B[1],B[2],B[3] sont dans le même set, ainsi que A[128],A[129],A[130],A[131] et B[128],B[129],B[130],B[131]

7-c Remplacement: caches associatifs

Accès à B[0]	**MISS**	Place B[0],B[1],B[2],B[3] dans premier set	
Accès à A[0]	**MISS**	Place A[0],A[1],A[2],A[3] dans premier set	
Accès à B[1]	**HIT**		
Accès à A[1]	**HIT**		
...	...	...	...
Accès à B[128]	**MISS**	Place B[128],B[129], B[130],B[131] dans premier set	Enlève B[0],B[1],B[2],B[3] car moins récemment accédée
Accès à A[128]	**MISS**	Place A[128],A[129], A[130],A[131] dans premier set	Enlève A[0],A[1],A[2],A[3] car moins récemment accédée
Accès à B[129]	**HIT**		
Accès à A[129]	**HIT**		

7-c Politique d'écriture

Quelle mise à jour du cache si on écrit une donnée vers la mémoire ?

Cache hit:

- **Write through**: On écrit dans le cache et la mémoire. Reste simple même si augmente l'utilisation de la bande passante mémoire
- **Write back**: On écrit dans le cache. La donnée est écrite en mémoire seulement si enlevée du cache. Un bit (appelé dirty bit) permet d'éviter copie vers mémoire si donnée pas écrite.

Cache miss (pour une écriture):

- **No write allocate**: ne pas ramener la donnée écrite en cache
- **Write allocate**: ramène la donnée en cache

Combinaisons possibles: write through et no write allocate

7-c Causes de cache miss

Cache miss obligatoire (compulsory miss):

- on accède pour la première fois à la donnée

Cache miss de capacité (capacity miss):

- La donnée a déjà été accédée mais a été enlevée du cache car le volume de données accédé depuis excède la taille du cache

Cache miss de conflit (conflict miss):

- La donnée a déjà été accédée mais a été enlevée du cache car d'autres données l'ont évincée de son set. Il est possible qu'il reste de la place ailleurs dans le cache (problème dû à la politique de placement)

7-c Prefetching: améliorer fonctionnement caches

Fonctionnement de base: demande une donnée ou instruction en mémoire quand on fait un accès

Prefetching: ne pas attendre et demander à l'avance les données ou instruction dont on aura besoin après

→ *mécanisme clé pour masquer la latence mémoire*

Faire du prefetching sur quoi ?

- Instructions, flot des instructions prévisibles (sauf si branchement)
- Parcours de tableaux (adresses) réguliers

Comment faire le prefetching (préchargement) ?

- Prefetching matériel (tables de prédiction des accès mémoire)
- Prefetching logiciel (instructions à placer par compilateur)

7-c Prefetching matériel

Quelles stratégie pour prefetcher des données (ou instruction)?

- Prefetch sur miss: lorsqu'on fait un miss sur le bloc de données *b* (une ligne de cache), prefetcher *b+1*
- Prefetcher une ligne de cache à l'avance
- Prefetch avec incrément
  - Si la séquence d'adresses est *b, b+1, b+2*, prefetcher *b+3*, *b+4*...
  - Si la séquence d'adresses est *b, b+2, b+4*, prefetcher *b+6, b+8*...
  - Utilise des prédicteurs d'adresse, basé sur un historique

Exemple Power5: 8 flux d'adresses indépendants prefetchés avec incrément, jusqu'à 12 lignes de cache d'avance

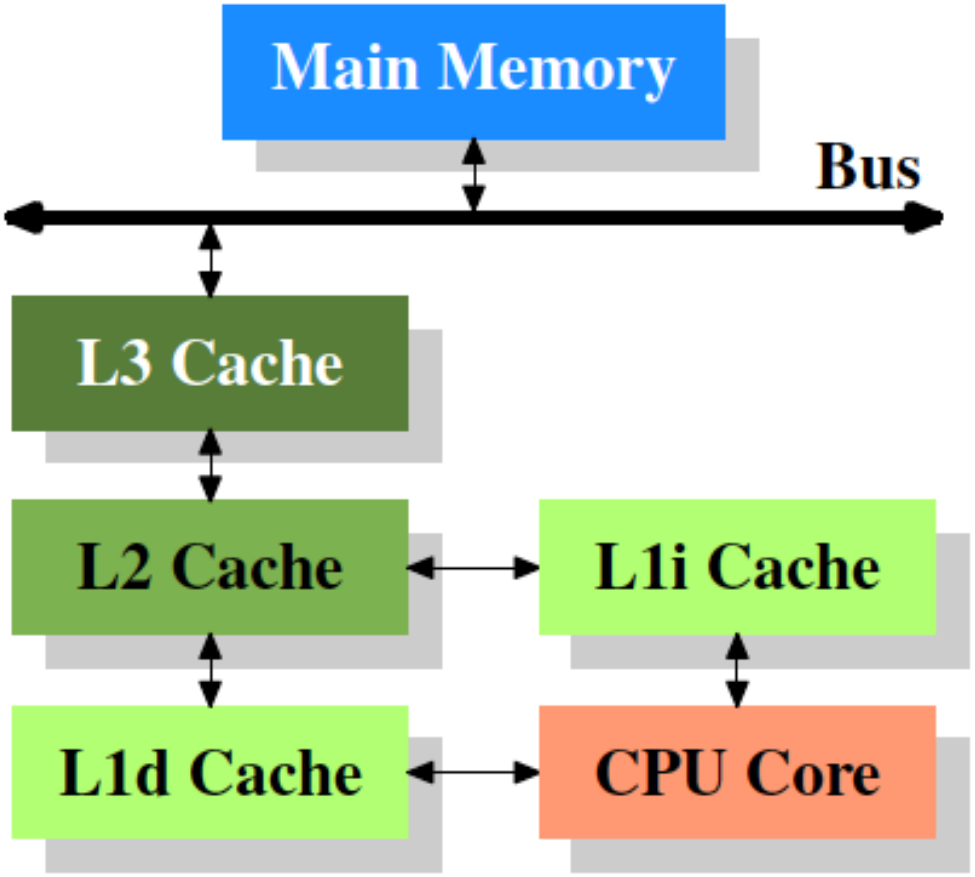
7-c Quelques exemples de cache

Cache parfois séparé entre instructions et données

Hiérarchie de caches

Exemple Power4

- L1 instruction: direct map
- L1 données: 2-associatif
- L2: 1.44Mo, 8-associatif
- L3: 32Mo, 8-associatif



7-c Quelques exemples de caches

	Intel Madison 6Mo (Itanium)	IBM Power4+	IBM Power5
Frequence (Ghz)	1.5	1.7	1.9
Latence L2	5 cycles: 3.3 ns	12 cycles: 7.1 ns	13 cycles: 6.8 ns
Latence L3	14 cycles: 9.3 ns	123 cycles: 72.3 ns	87 cycles: 45.8 ns
Latence mémoire	224 cycles: 149ns	351 cycles: 206ns	220 cycles: 116ns
Copie d'éléments consécutifs d'un tableau	5.07Go/s	8.37Go/s	17.9Go/s

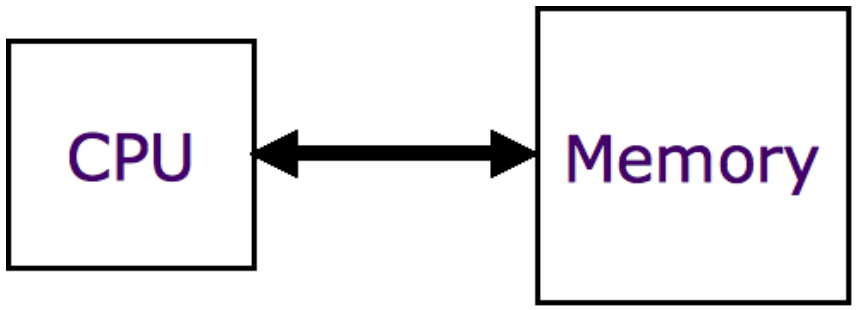
7- Mémoire

a- Cellules mémoires (SRAM, DRAM)

b- Problème de la mémoire (et comment s'en sortir)

c- Caches

d- Mémoire virtuelle



7-d Mémoire virtuelle

Simplifie prog. accès mémoire

Découple l'espace d'adressage vu par l'utilisateur de la mémoire physique.

Les processus ne voient et ne peuvent adresser que leur mémoire

Mécanisme de pages:

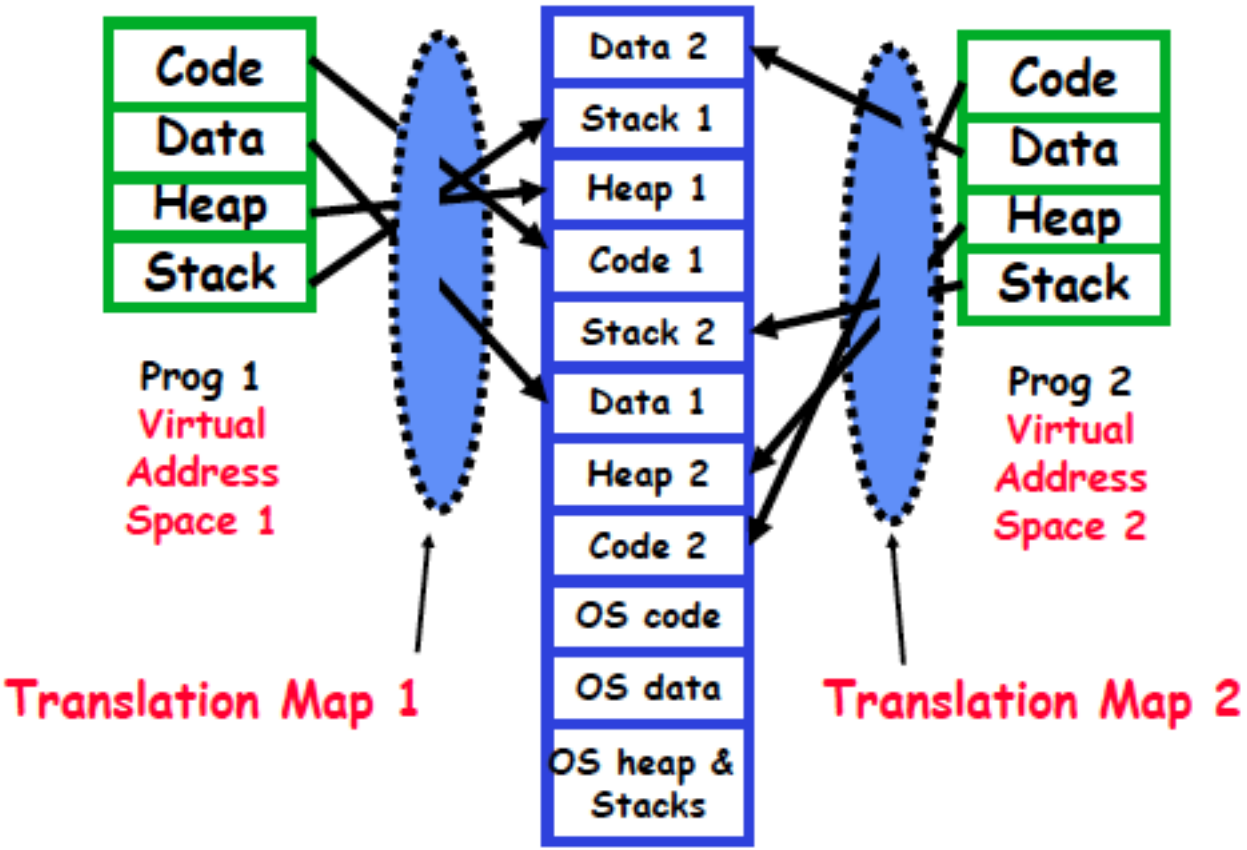
- Mémoire gérée par blocs de 2^k cellules mémoire (4Ko ou 2Mo sur x86), des **pages**
- Chaque page virtuelle correspond à une page réelle

Utilité:

- Chaque processus (programme s'exécutant) a l'impression d'avoir toute la mémoire → **gestion transparente du partage mémoire**
- Compartimente la mémoire par processus → **protection entre utilisateurs/programmes**

7-d Mémoire virtuelle

Lien entre mémoire réelle et physique: chaque processus a sa mémoire virtuelle



7-d Mémoire virtuelle

Traduction à la volée des adresses virtuelles ↔ adresses physiques:

Fait par le matériel: la MMU (**memory management unit**)

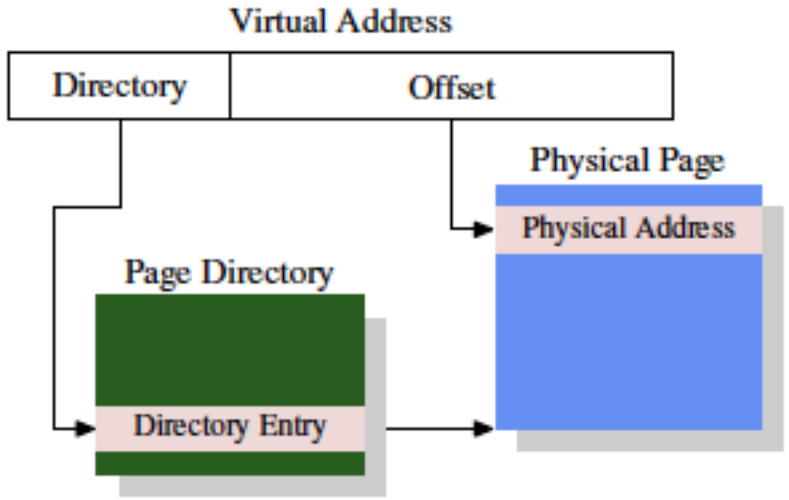
- Les caches peuvent être soit en adresses virtuelles soit adresses physiques

7-d Mémoire virtuelle

Traduction à la volée des adresses virtuelles ↔ adresses physiques:

Conversion par table de pages: l'adresse virtuelle est décomposée en deux

- Une partie (directory) sert d'index dans la table de conversion
- Le reste (offset) sert de déplacement dans la page.



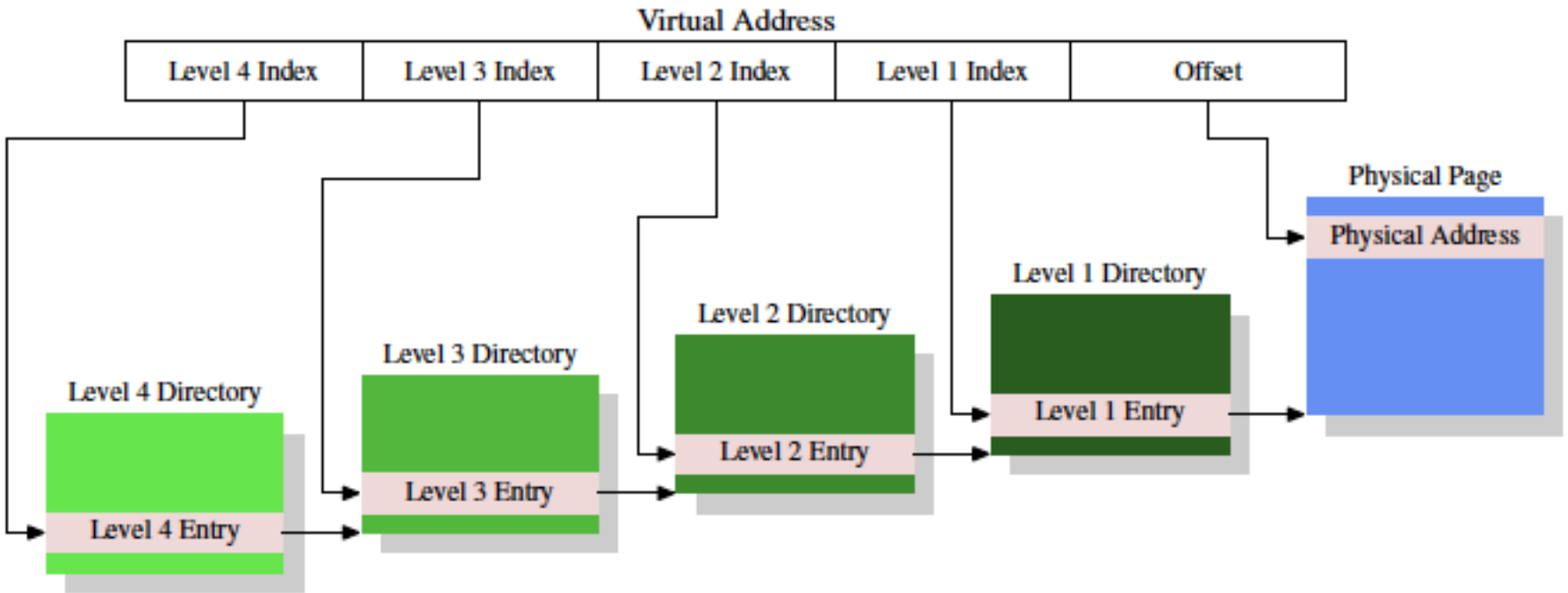
7-d Mémoire virtuelle

Traduction à la volée des adresses virtuelles ↔ adresses physiques:

Si la page fait une taille de 4Ko, l'offset fait 12 bits, le champ directory fait 20 bits

→ **Table des pages avec 2^{20} entrées !** (la plupart des entrées sont vides)

Solution: Multiples niveaux de tables



7-d Mémoire virtuelle

Traduction à la volée des adresses virtuelles ↔ adresses physiques:

Toutes les tables de pages sont en mémoire

→ Beaucoup d'accès mémoire à chaque accès demandé !

Solution: Mettre dans un cache les traductions

Dans le L1 ou L2:

- Risque que les données soient évincées par d'autres données
- Augmente l'utilisation de la bande passante au cache

Dans un cache dédié, le TLB (Translation Look-Ahead Buffer)

- Petit, reservé à la traduction des adresses des pages

7- Mémoire: conclusion

- **Les mémoires sont plus lentes que les processeurs**

Ca ne va pas s'améliorer !

- Mécanismes essentiels pour contrer ce problème
  - Les caches: nécessite d'exprimer de la **localité**
    - De la localité spatiale: accéder à des données proches les unes des autres
    - De la localité temporelle: réutiliser les mêmes données à un intervalle de temps faible

→ Impact sur la façon de programmer ! (cf p224)

  - Le prefetch: prédit les prochaine adresses accédées en fonction des accès passés

→ Impact sur le choix/parcours des structures de données ! (cf p224)