

Hiérarchie mémoire

**Comment améliorer la capacité
et la rapidité d'accès à la
mémoire centrale?**

Hiérarchie mémoire

- Tous les programmes que l'on exécute et toutes les données auxquelles on accède doivent être «chargés» en mémoire centrale afin que le CPU puisse y accéder
- L'accès à la mémoire centrale est l'étape la plus lente lors de l'exécution d'une instruction
- La mémoire centrale n'est pas assez grande pour contenir le programme et les données
- Les mémoires à grande capacité sont lentes et les mémoires rapides sont chères et à faible capacité !

Modèle classique de processeur "Von Neumann"



Inconvénients "goulot d'étranglement"

- lenteur : accès mémoire les uns après les autres, avec une durée déterminée par le protocole de communication
très coûteux puisque accès mémoire en plusieurs tops d'horloge processeur
- pas adapté à l'évolution technologique : les performances du CPU doublent tous les 18 mois, alors que celles des mémoires doublent tous les 7 ans (loi de Moore)

Ordre de grandeur :

période de l'ordre de 1ns pour CPU du Pentium III,
temps d'accès DRAM de l'ordre de 40ns

Hiérarchie mémoire

Amélioration

Tenir compte des caractéristiques des accès mémoire impliquées par les langages de haut niveau

« localité des références » : on accède successivement dans le temps à des mots d'adresses consécutives

⇒ la mémoire est constituée de *plusieurs* unités conçues dans ≠ technologies.

Les unités à accès rapide contiennent des *copies* des données des unités à accès lent en cours d'utilisation (données à des adresses consécutives).



1. Classes de mémoires

ROM : Read Only Memory

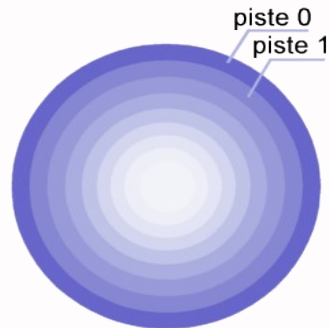
- stockage permanent (tableau logique et / ou)
- accès rapide
- utilisée pour stocker les programmes de boot, le BIOS, les micro-instructions dans le cas d'une partie contrôle micro-programmée
- modifiable : PROM, EPROM → *mémoire flash*

Hiérarchie mémoire

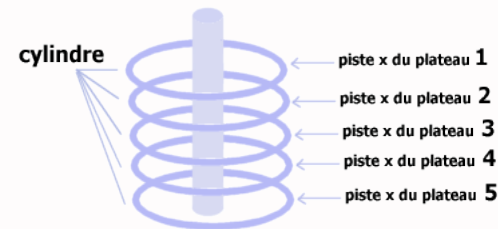
Disques durs

- disque magnétique, stockage permanent
- accès séquentiel : pas le même temps pour accéder à chaque piste
- très grande capacité, accès lent

ensemble de disques



cylindre



unité de transfert : 512 o

Hiérarchie mémoire

RAM: Random Access Memory

- volatile
- chaque mot est accessible indépendamment, avec le même temps d'accès

Mémoires statiques SRAM

Accent sur la rapidité d'accès

L'information stockée reste valide statiquement (dans des flip/flop)

→ temps d'accès mémoire \equiv temps d'acquisition de la donnée

Utilisées pour les caches

Hiérarchie mémoire

Mémoires dynamiques DRAM

Accent sur la capacité mémoire

Stockage dynamique dans des condensateurs, l'information doit être réécrite régulièrement (un millier de fois par seconde)

→ temps d'accès mémoire décomposé en

- temps d'acquisition
- temps de mise à jour

SDRAM : accès synchrone, DRAM Rambus : boîtier bus, DRAM cache : avec cache intégré

Utilisées pour la mémoire principale

Hiérarchie mémoire

	Type de mémoire	Technologie	Taille	Temps d'accès
De plus en plus loin du CPU	Registre	flip-flop	32, 64, 128	1 ps
	Cache	SRAM	8 à 512 Ko	1 à 5 ns
	Mémoire centrale	DRAM	256, 512 Mo à 1Go	40 ns
	Disque	Disque magnétique	120 Go à 200 Go	7 à 10 ms <i>10 Mo par s</i>
	Disque optique	CD-ROM	100 Go	300 ms <i>600 Ko par s</i>
	Clé USB	Mémoire flash	4 Go, 2 Go, 512 Mo, 216 Mo	<i>11 Mo par s</i>

DRAM plus grande capacité que les SRAM

SRAM plus rapides que les DRAM

SRAM consomment plus que les DRAM

SRAM plus chères que les DRAM

SRAM plus encombrantes que les DRAM

2. Mémoire multi-niveau

Coordonner des mémoires à accès rapide (chère, petite) avec des mémoires à accès lent (peu chère, grande) de façon à ce que la plus grande partie des accès se fasse vers les mémoires rapides

Basé sur le principe de localité des références : Des parties de programmes accèdent successivement dans le temps (localité temporelle) à des adresses consécutives de la mémoire (localité spatiale).

Localité dans la mémoire de données

Localité dans la pile

Localité dans la mémoire d'instructions

Des exemples de localité de référence ?



Hiérarchie mémoire

Objectif des mémoires multi-niveau :

temps d'accès mémoire du système global \equiv temps d'accès à la mémoire rapide
capacité du système global \equiv capacité de la grande mémoire

Principe utilisé entre :

- cache primaire et secondaire
- cache secondaire et mémoire centrale
- cache intégré dans la mémoire
- mémoire centrale et disque (mémoire virtuelle)

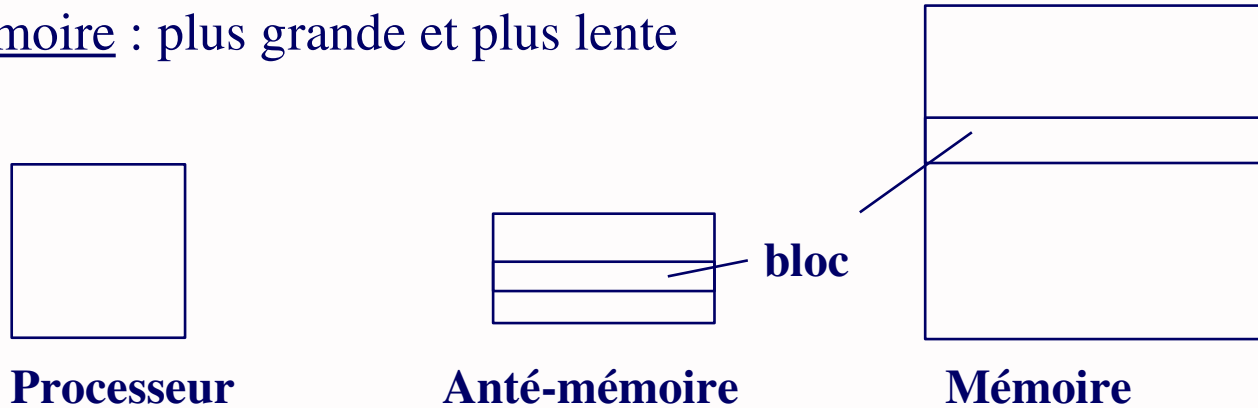
Hiérarchie mémoire

2.1. Principes

Plusieurs niveaux de mémoire, deux niveaux adjacents sont manipulés à la fois

anté-mémoire : plus proche du CPU, plus petite et plus rapide

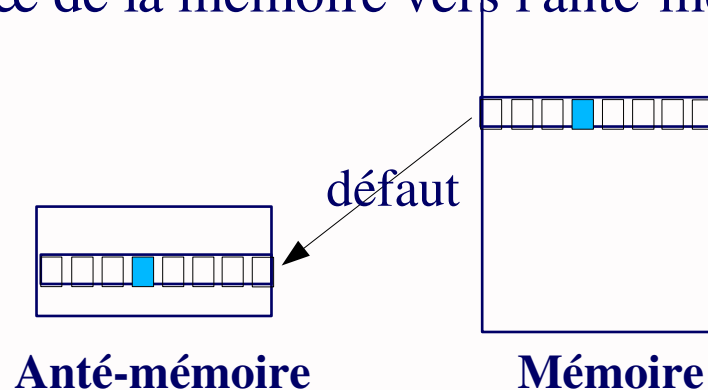
mémoire : plus grande et plus lente



unité minimale d'information: **bloc** = x mots
(par exemple 1 bloc = 8 mots)

Hiérarchie mémoire

- L'information de l'anté-mémoire est une copie d'une partie de l'information de la mémoire
- Si on trouve la donnée dans l'anté-mémoire on y accède
- Si la donnée n'est pas dans l'anté-mémoire on recopie le bloc complet qui contient cette donnée de la mémoire vers l'anté-mémoire ⇒ défaut



Selon le principe de la localité des références, si l'on vient d'accéder à une adresse d'un bloc, on accèdera bientôt à des adresses du même bloc

Hiérarchie mémoire

2.2. Performances des mémoires multi-niveau

Pe: taux d'échec d'accès à l'anté-mémoire

(i.e probabilité qu'un mot mémoire accédé ne soit pas dans l'anté-mémoire)

Ta: temps d'accès à l'anté-mémoire si succès

Te: temps de traitement si échec

$$\text{temps moyen accès mémoire} = P_e * T_e + (1 - P_e) * T_a$$

Hiérarchie mémoire

Exemple: temps d'accès anté-mémoire de 3 ns, temps d'accès mémoire de 50 ns, les blocs contiennent 8 mots donc $T_e = 8 \times 50$ ns.

Temps moyen d'accès à un mot mémoire sans hiérarchie : 50 ns

$P_e = 0,01$

```
for (j = 1; j < 100; j++)
```

$T_e = 8 \times 50$ ns

```
    traiter(tableau(i));
```

$T_a = 3$ ns

Temps moyen d'accès avec hiérarchie: $0,01 \times 8 \times 50 + 0,99 \times 3 = 6,97$ ns

$P_e = 0,9$ switch (choix) {

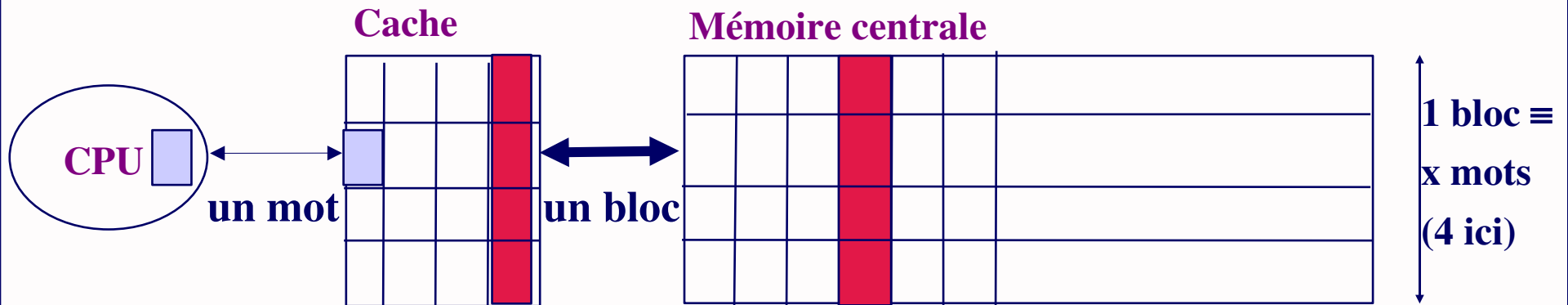
$T_e = 8 \times 50$ ns case 1: pas = 10; rienFaire(tableau, pas);

$T_a = 3$ ns case 2: pas = 50; dodo(tableau, pas);

}

Temps moyen d'accès avec hiérarchie: $0,9 \times 8 \times 50 + 0,1 \times 3 = 360,3$ ns

3. Mémoire cache



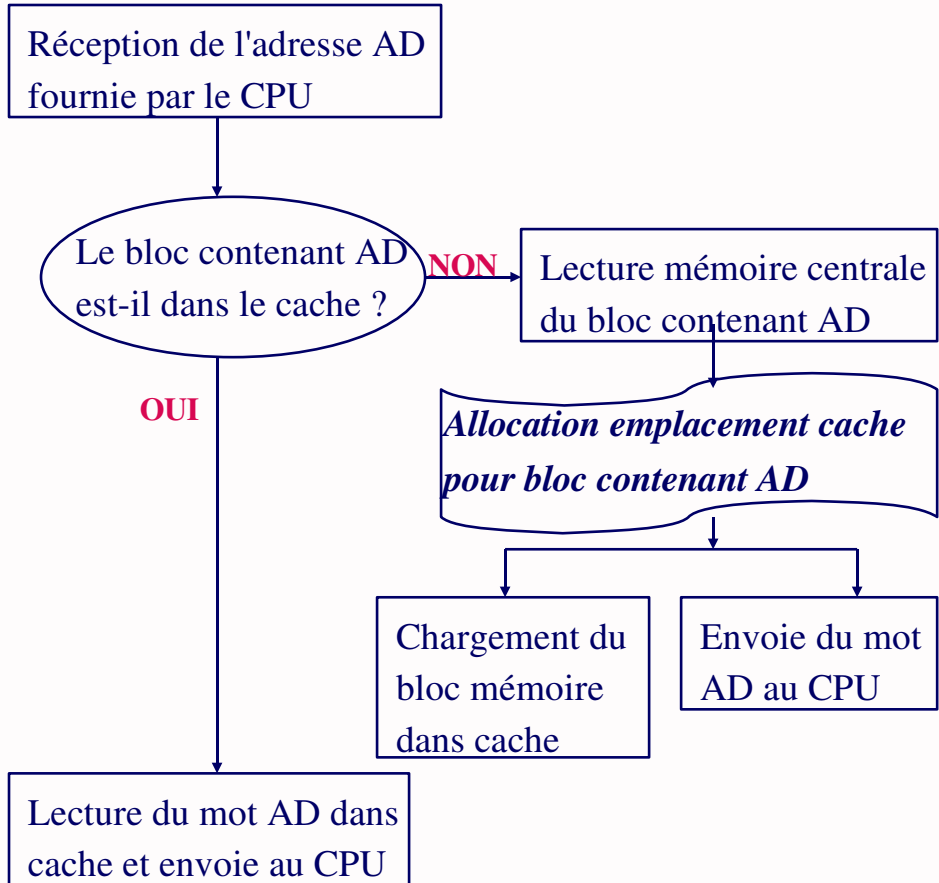
Petite mémoire entre processeur et mémoire centrale

Contient des copies des mots de la mémoire qui risquent d'être accédés par le processeur (localité des références)

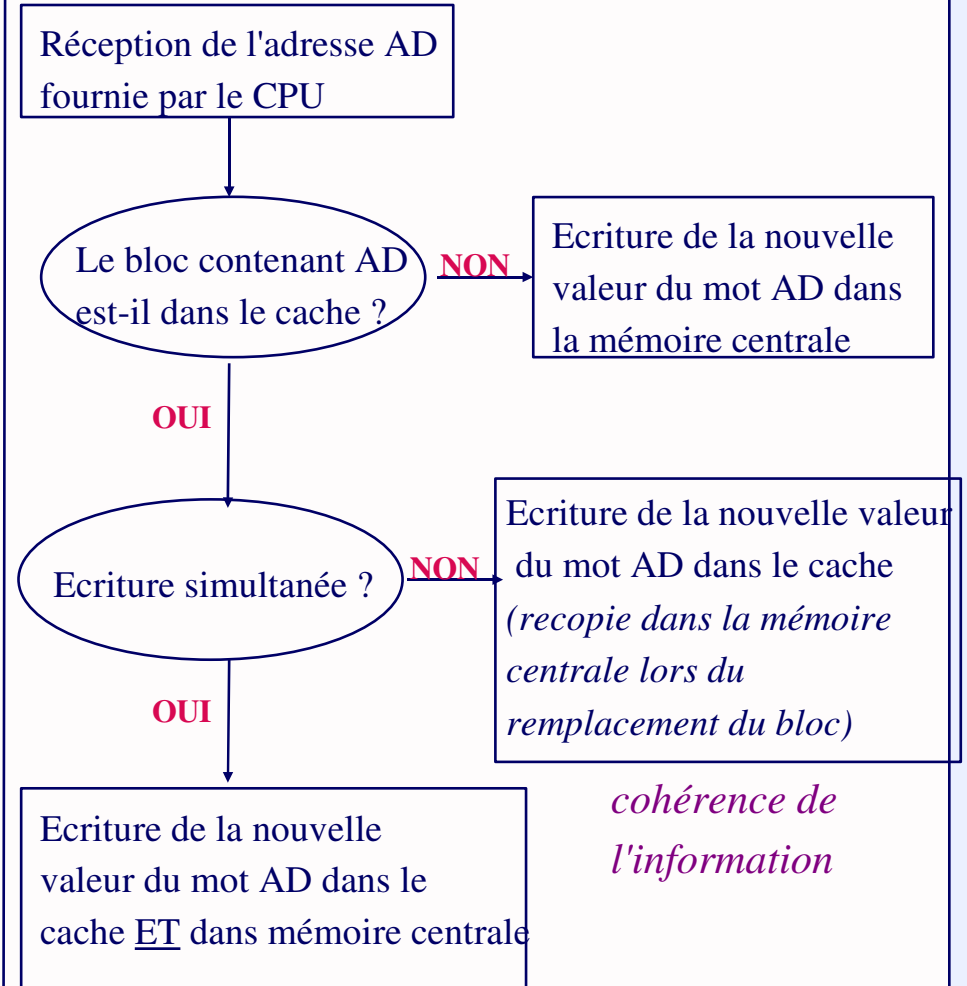
Le cache est redondant -> il n'augmente pas la capacité de la mémoire principale, mais seulement sa rapidité.

3.1. Fonctionnement du cache

Lecture



Ecriture



cohérence de l'information

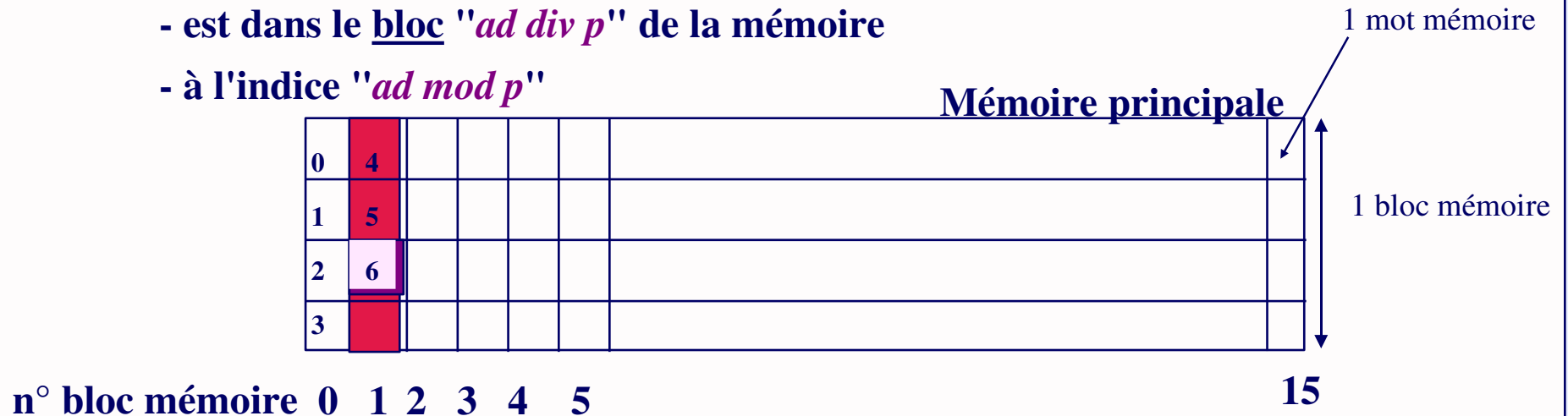
coûteux

Hiérarchie mémoire

3.2. Lien entre adresse mémoire et blocs mémoire

Si les blocs contiennent p mots, le mot d'adresse ad en mémoire :

- est dans le bloc " $ad \text{ div } p$ " de la mémoire
- à l'indice " $ad \text{ mod } p$ "



Exemple: $p = 4$, $ad = 6$

- le mot d'adresse 6 se trouve dans le bloc $6 \text{ div } 4 = 1$ de la mémoire
- avec un déplacement de $6 \text{ mod } 4 = 2$

Hiérarchie mémoire

3.3. Bloc de cache

Le cache contient beaucoup moins de blocs que la mémoire :

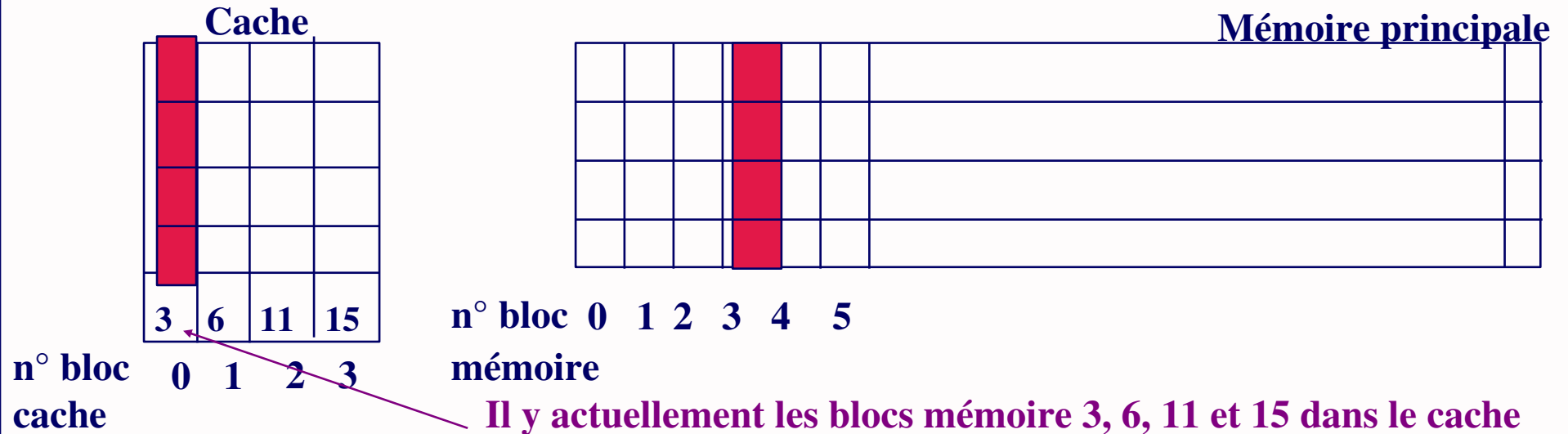
- un bloc de cache recevra successivement plusieurs blocs mémoire
- il faut donc pouvoir identifier quel est le bloc mémoire présent dans le cache

Bloc de cache:

clé	valeur du bloc
-----	----------------

Clé : identificateur du numéro de bloc mémoire

Valeur : contenu du bloc mémoire (x cases mémoire)



3.4. Où placer/chercher un bloc mémoire dans le cache ?

La place d'un bloc mémoire dans le cache est calculée à partir du numéro de bloc mémoire selon une stratégie de placement qui assure :

- **le plus bas taux d'échec de recherche du bloc**

le bloc cherché est bien dans le cache à la place calculée

- **un coût réduit de recherche du bloc**

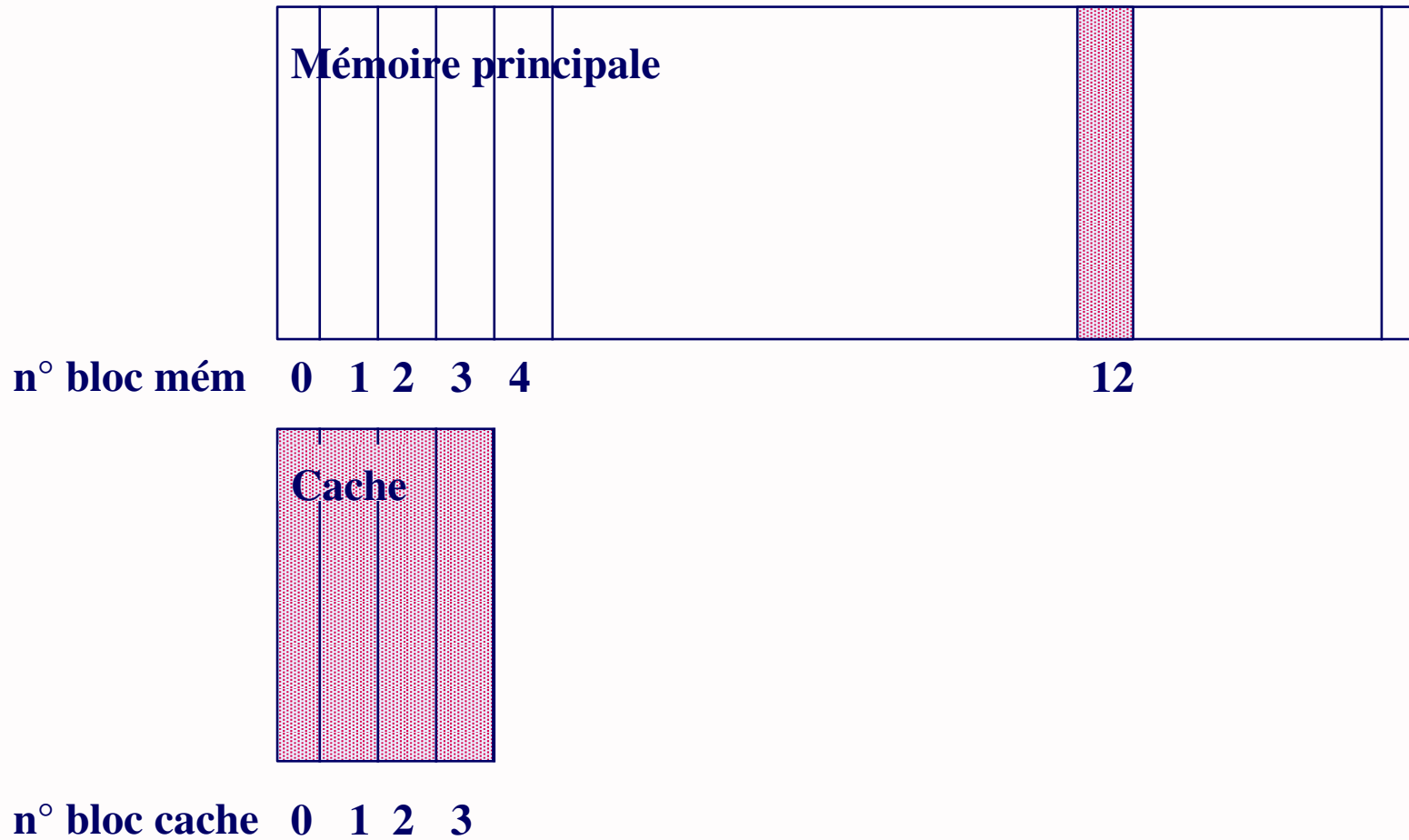
un bloc mémoire a peu d'emplacements possibles dans le cache

3 stratégies

- **caches associatifs**
- **caches à accès direct**
- **caches associatifs par ensemble de blocs**

Hiérarchie mémoire

Cache associatif: Un bloc mémoire peut se trouver n'importe où dans le cache

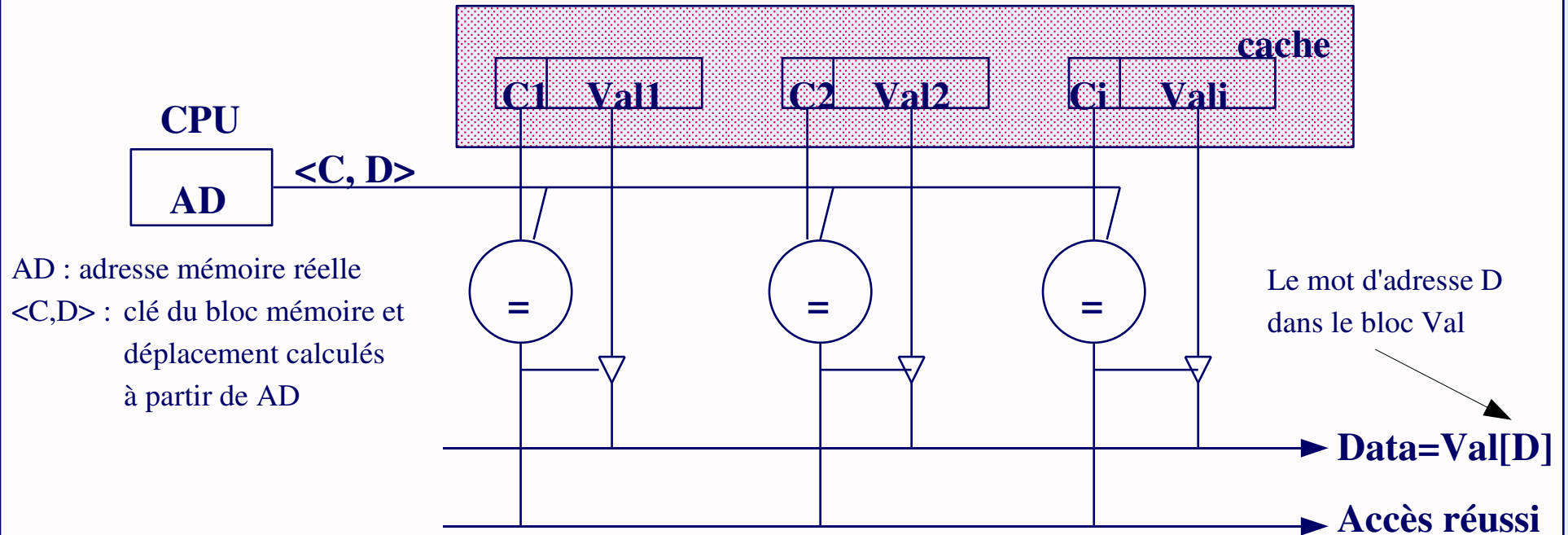


Hiérarchie mémoire

Recherche d'un mot dans un cache de taille $N \rightarrow O(N)$ comparaisons

Pour un accès en $O(1)$, ajout d'une logique de comparaison des clés (rapide mais coûteux)

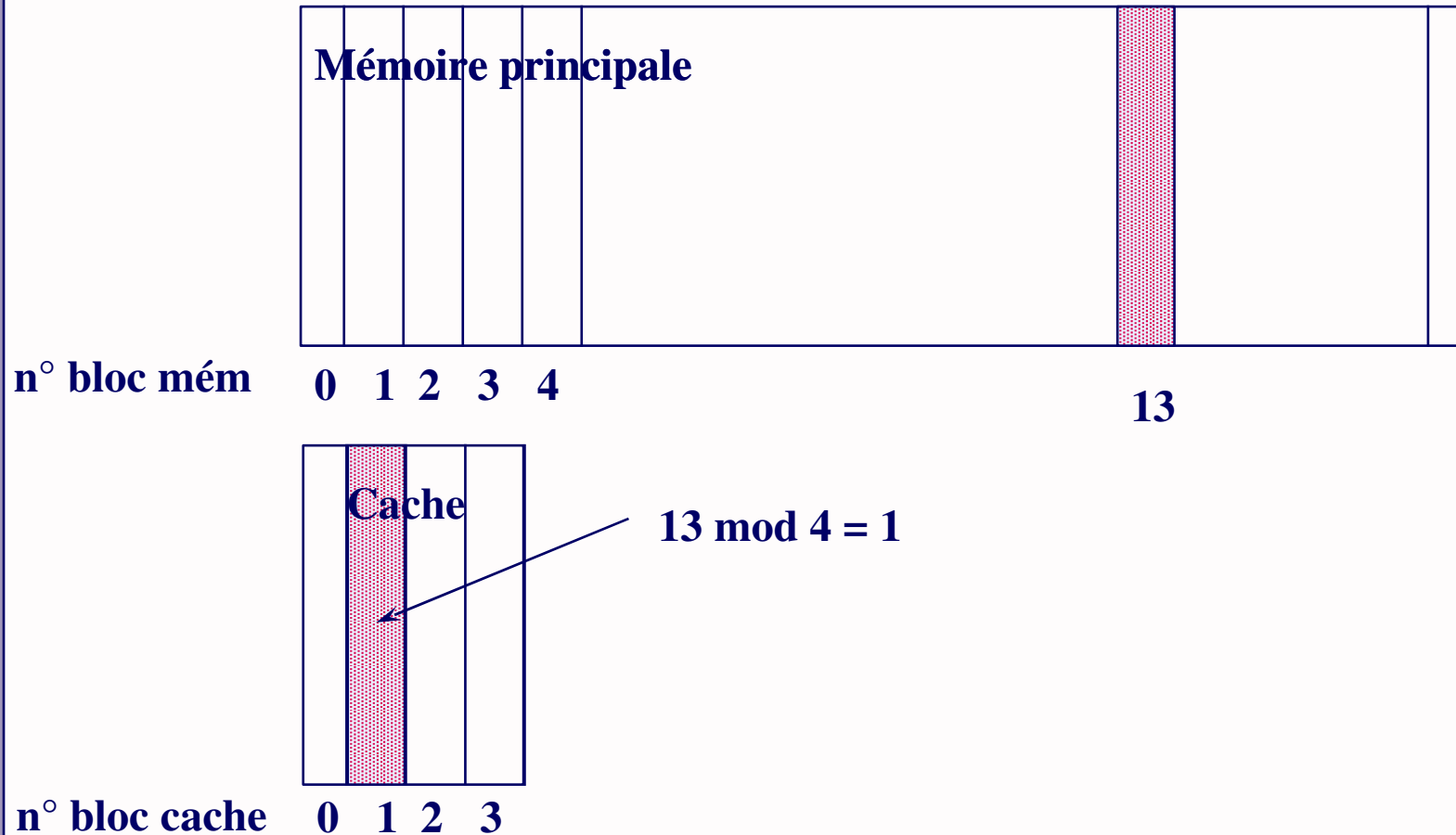
\Rightarrow CAM: Content Adressable Memories



Hiérarchie mémoire

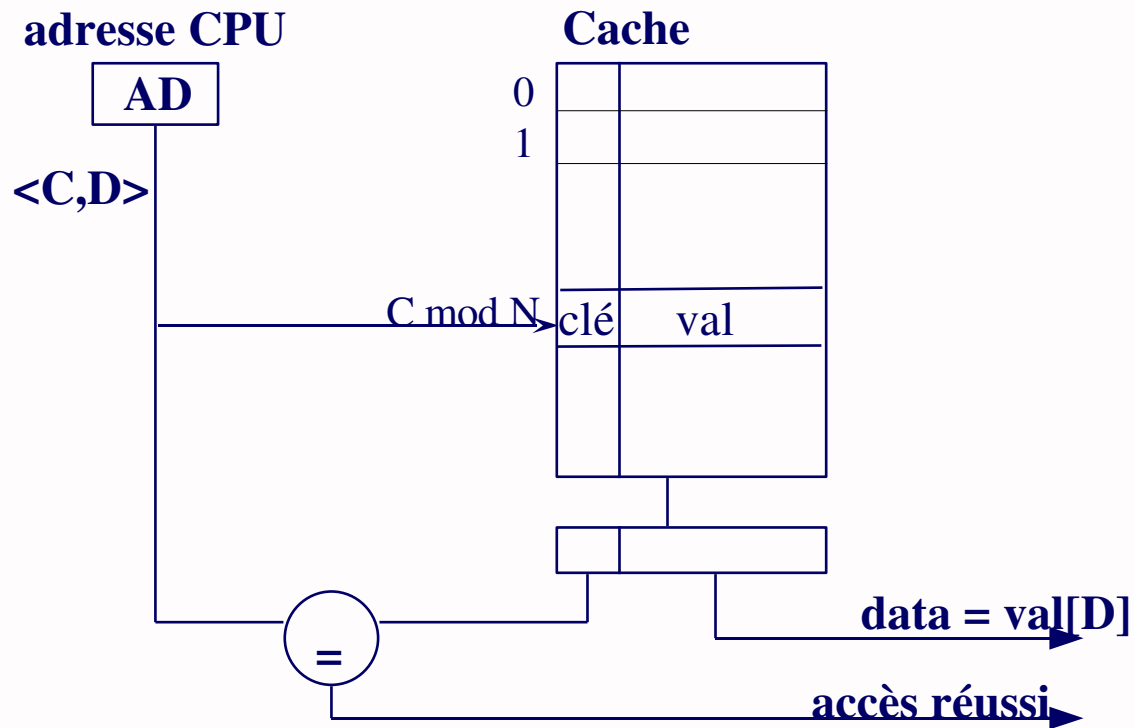
Cache à accès direct: Un bloc mémoire a un *unique* emplacement possible dans le cache

Le bloc mémoire M se place dans le bloc $M \bmod N$ du cache
où N est le nombre de blocs du cache



Hierarchie mémoire

Sacrifier le parallélisme des caches associatifs pour utiliser des RAM ordinaires



Exemple

Cache à accès direct organisé en 4 blocs de 16 mots; lecture du mot d'adresse 178

cache

8	17	6	5

0 1 2 3

mémoire

Bloc 0 adresses 0..15
Bloc 1 adresses 16..31
.
.
.
Bloc 11 adresses 176..192

1. *Quel est le bloc mémoire correspondant ?*



2. *Où se trouve ce bloc dans le cache ?*



3. *Y a-t-il défaut de cache ?*



Exemple

Cache à accès direct organisé en 4 blocs de 16 mots; lecture du mot d'adresse 178

cache

8	17	6	5
0	1	2	3

mémoire

Bloc 0 adresses 0..15
Bloc 1 adresses 16..31
.
.
.
Bloc 11 adresses 176..192

1. *Quel est le bloc mémoire correspondant ?*

Le mot d'adresse 178 est dans le bloc
 $178 \div 16 = 11$ de la mémoire

2. *Où se trouve ce bloc dans le cache ?*

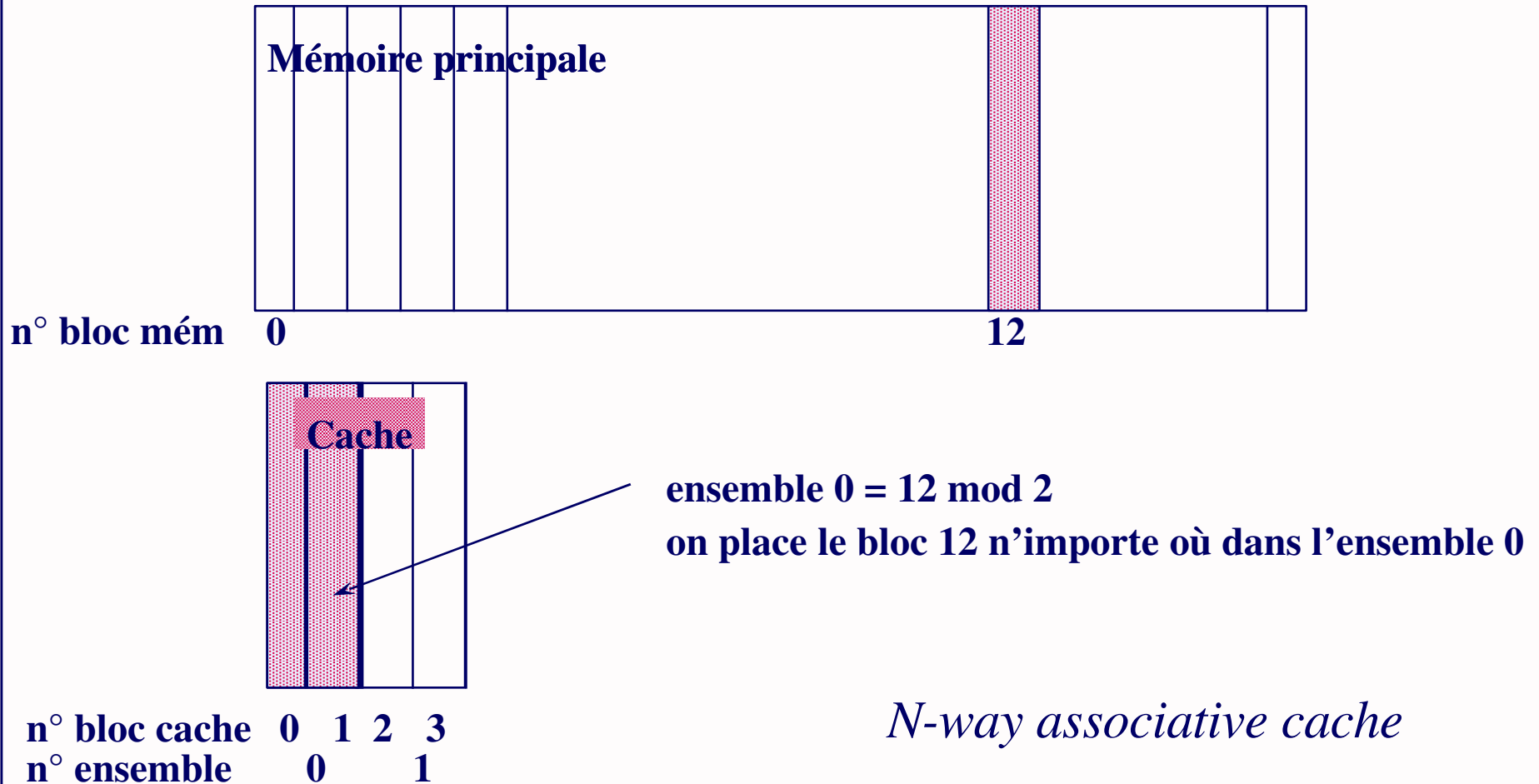
le bloc 11 mémoire se cherche dans le bloc
 $11 \bmod 4 = 3$ du cache

3. *Accès cache*

Le bloc 3 du cache contient le bloc 5 de la mémoire : il y a défaut de cache
 → copie du bloc 11 mémoire dans le bloc 3 du cache

Hiérarchie mémoire

Cache associatif par ensemble de blocs: mot mémoire placé dans le cache n'importe où dans un ensemble de blocs. Le numéro d'ensemble est calculé comme dans le cas accès direct : $M \bmod N$ où N est le nombre d'ensembles du cache



3.5. Quel bloc remplacer lors d'un défaut de cache?

- **cache à correspondance directe: une seule place possible → on écrase le bloc correspondant**
- **cache associatif: on écrase le bloc qui n'a pas été utilisé depuis le plus longtemps (Last Recently Used), ou celui qui est le moins souvent utilisé (Least Frequently Used)**
- **cache associatif par ensemble de blocs: dans l'ensemble associé au bloc mémoire, on utilise la stratégie LRU ou LFU**

Récapitulatif

- cache à accès direct: recherche facile **mais** une seule place possible \Rightarrow fort taux d'échec

Exemple:

pour un cache de 4 blocs, les blocs mémoire 1, 5, 9, 13, sont tous égaux à

1 modulo 4 et se placent tous dans le bloc 1 du cache

\Rightarrow ils ne peuvent donc pas être dans le cache en même temps

- cache associatif: moins d'échec car on peut mettre le bloc où l'on veut **mais** recherche difficile (caches chers)
- cache associatif par ensemble de blocs : compromis entre les 2

3.6. Éléments de conception des caches

- Taille du cache : assez petite pour que le coût global des accès soit proche de celui de la mémoire centrale seule mais assez grande pour être efficace.

Ex : IBM 360 (1968) 16Ko, Intel 486 (1989) 8Ko, Intel Pentium (1993) 8Ko/8Ko + 512 Ko, Itanium (2001) 16Ko/16Ko + 96Ko + 4 Mo

- Taille des blocs : assez grande pour accéder pendant longtemps à des données consécutives mais pas trop grande car sinon il y a peu de blocs et donc on écrase des blocs utiles

- Nombre de caches : la taille des composants diminue donc il est possible d'intégrer dans la puce plusieurs caches.

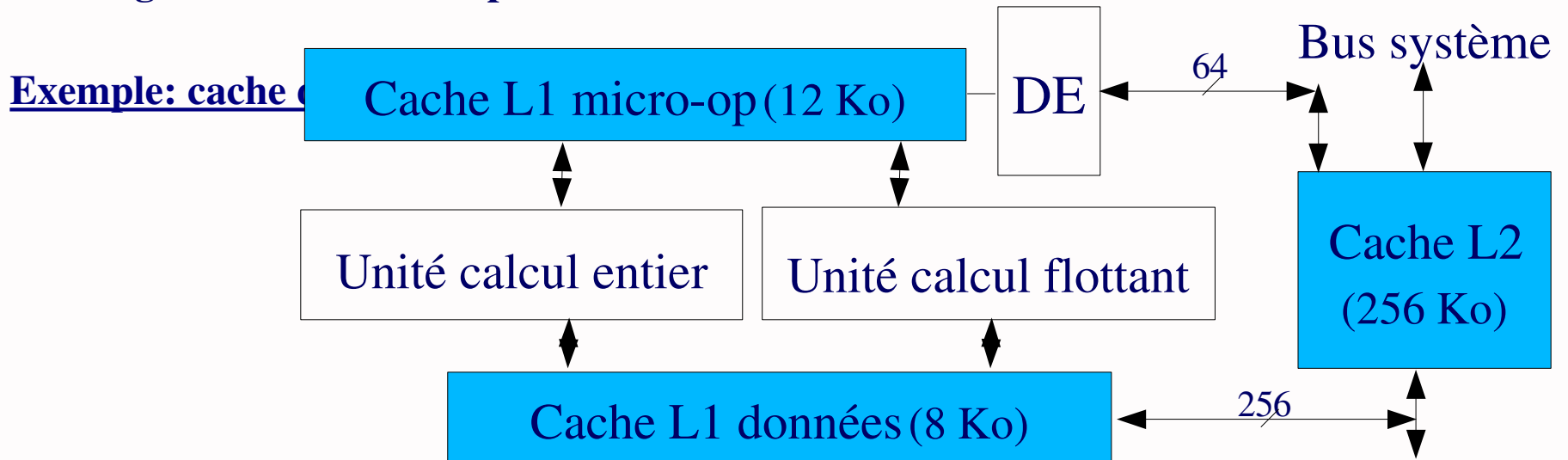
En général, un (ou deux) cache interne L1 plus un cache externe L2.

L1 : unifié ou séparé (instructions/données).

Hiérarchie mémoire

Exemple: cache du Pentium III

- 2 niveaux de cache :
 - cache primaire de 32 Ko (16ko instructions, 16ko données)
 - cache secondaire de 256 Ko
- accès au cache en 1 cycle
- organisation associative par ensemble de blocs (8 blocs par ensembles, **1 bloc = 16 octets**)
- stratégie de remplacement: bloc le moins récemment utilisé
- stratégies d'écriture: recopie immédiate



4. Pagination et mémoire virtuelle

- But de la pagination: obtenir une mémoire virtuelle à très grande capacité + simplifier la gestion multi processus
- Mémoire virtuelle = mémoire centrale + disque
 - page: unité de transfert (bloc)
 - adresse virtuelle: clé d'un mot de la mémoire virtuelle
 - adresse physique: adresse réelle du mot dans le disque ou la mémoire centrale
- La gestion des défauts de page est faite par software (routine d'interruption)

Remarque: accès disque: mili-seconde, 100 000 cycles horloge processeur
accès mémoire:nano-seconde, 1 à 10 cycles horloge processeur

Conclusion

