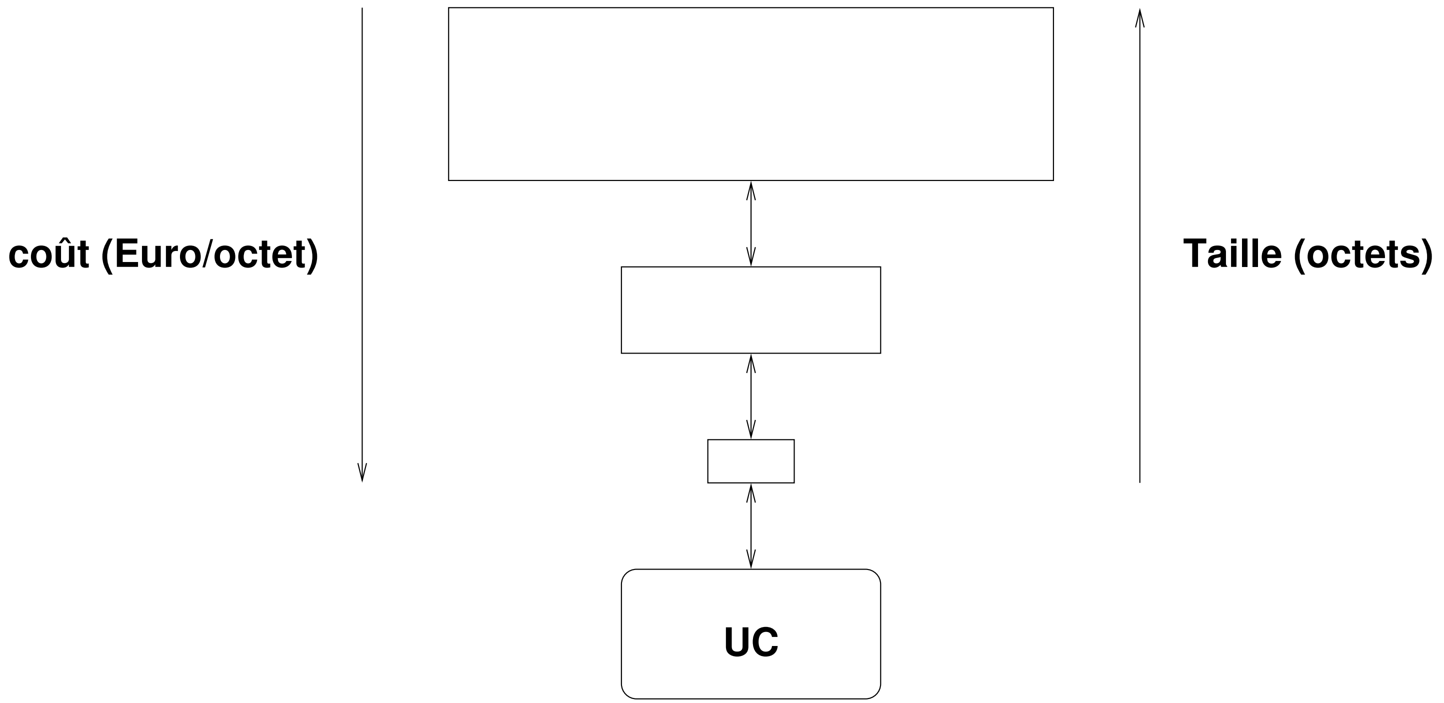
Mémoire Cache

Problème : la mémoire est beacoup plus lente que le processeur (facteur 10)

Les mémoires aussi rapides que le processeur coûtent trop chèr

La solution : une mémoire à plusieurs niveaux :



Mémoire Cache

Il y a souvent 3 couches : disque, mémoire principale, et mémoire cache

L'objectif est de faire croire que la mémoire entière est de la taille de celle que est plus grande, et du coût de celle qui est le moins chère

La rasion que cela marche souvent est le principe de localité :

> Une suite d'accès à la mémoire n'est pas distribuée de façon aléatoire. Les adresses des accès sont regroupées en un petit nombre de groupes. Les adresses d'un groupes font référence à un petit intervalle en mémoire

Raison du principe :

> Code séquentiel plus boucles de petite taille
>
> Données organisées en tableaux et structures

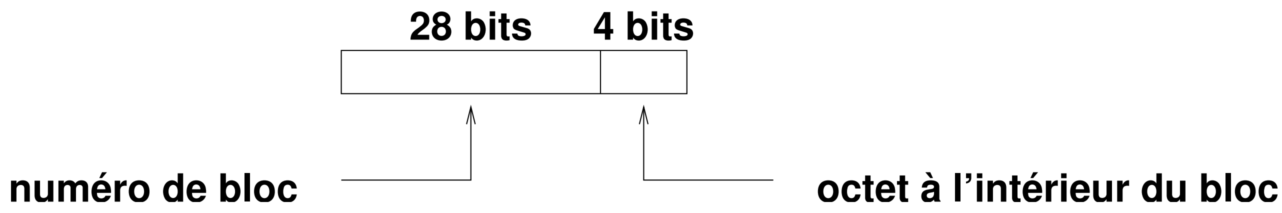
Cache, idée de base

L'idée de base est donc de copier un intervalle de code ou de données
de la mémoire lente à la mémoire rapide pour un accès plus rapide plus tard

Cet intervalle est souvent de taille 16 octets (actuellement). Nous appelons
un tel intervalle un "bloc" de mémoire

Une adresse est donc conceptuellement constituée par deux parties,
le numéro de bloc, et l'octet à l'intérieur du bloc (si les adresses de la
machine corréspondent à des octets).

Exemple (adresses de 32 bits):



Cache, chargement de blocs

Quand un bloc est référencé pour la première fois, le microprogramme le charge en mémoire cache

Si le bloc est référencé de nouveau, le cache est capable de satisfaire la référence sans accès à la mémoire principale

Quand le cache est plein, il faut choisir un bloc à supprimer

Le choix du bloc à supprimer est déterminé par le type de cache et par l'algorithme utilisé

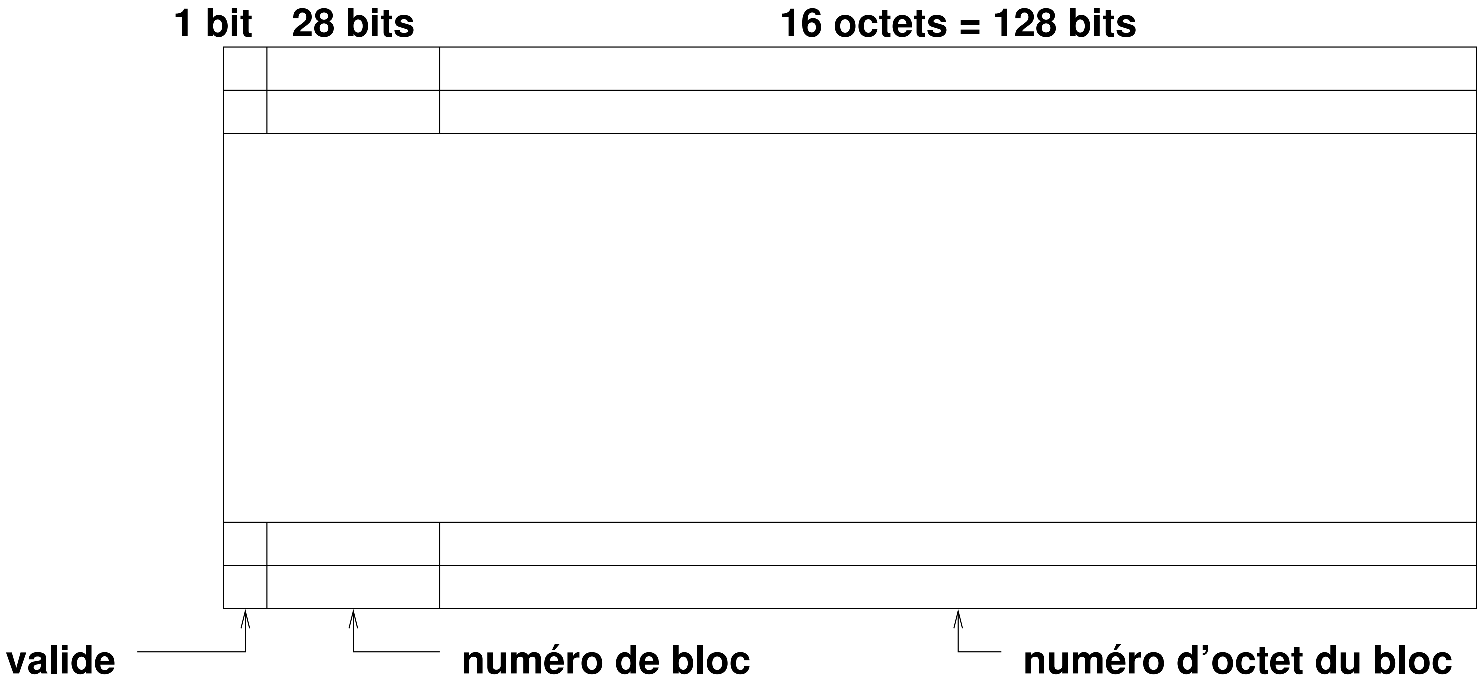
L'argorithme utilisé est souvent une version de LRU (least recently used, moins récemment utilisé), mais il y en a d'autres (random, ...)

Il reste donc la différence entre différentes types de cache. On utilise le mot "associativité" pour décrire cette différence

Cache associatif

Le cache est composé d'un nombre de "lignes de cache"
(ou simplement "ligne", anglais : cache lines), égal au nombre de blocs

Chaque ligne contient un bit pour indiquer si le contenu de la ligne est valide, un nombre de bits assez large pour déterminer un numéro de bloc de façon unique (dans notre exemple précédent : 28 bits), et le bloc (16 octets dans notre exemple, soit 128 bits)

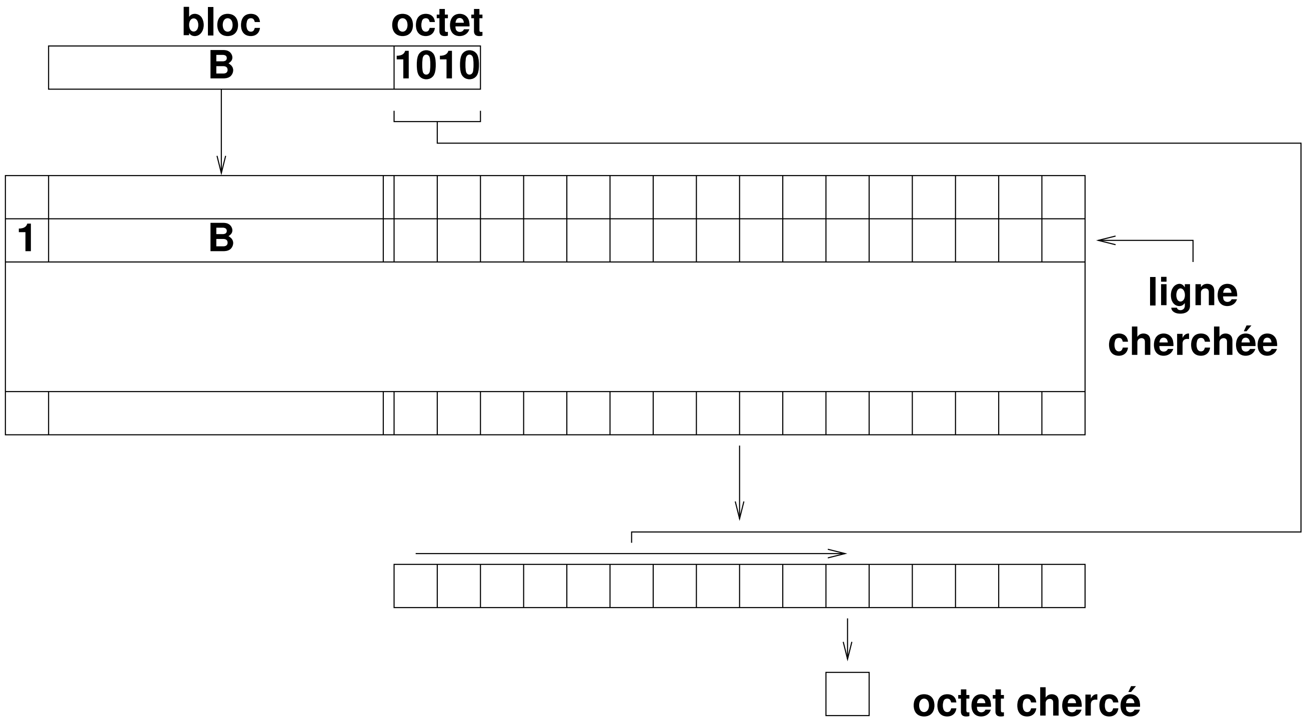


Cache associatif

Initialement, le bit "valide" est 0 pour toutes les lignes

Quand un bloc est chargé, le bit correspondant est mis à 1

Quand un accès à l'adresse A est tenté, le microprogramme vérifie si le bloc est dans le cache:



Cache associatif

Pour que l’accès soit rapide, le numéro de bloc cherché doit être comparé en parallèle avec toutes les lignes simultanément

Ceci nécessite un circuit spécial, très coûteux

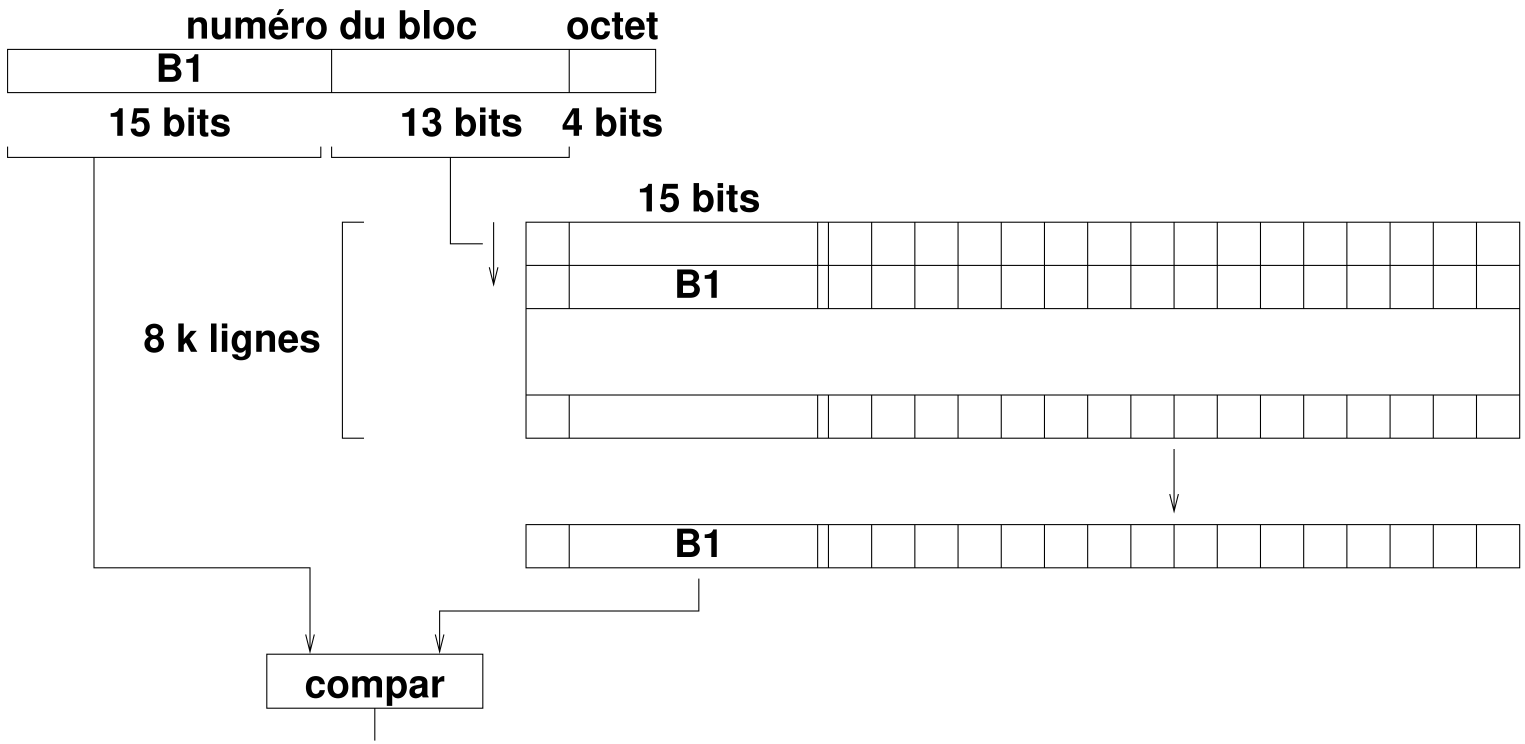
Pour cette raison, on n’utilise pas souvent un cache associatif

Cache direct

Un bloc ne peut pas être placé n'importe où dans le cache

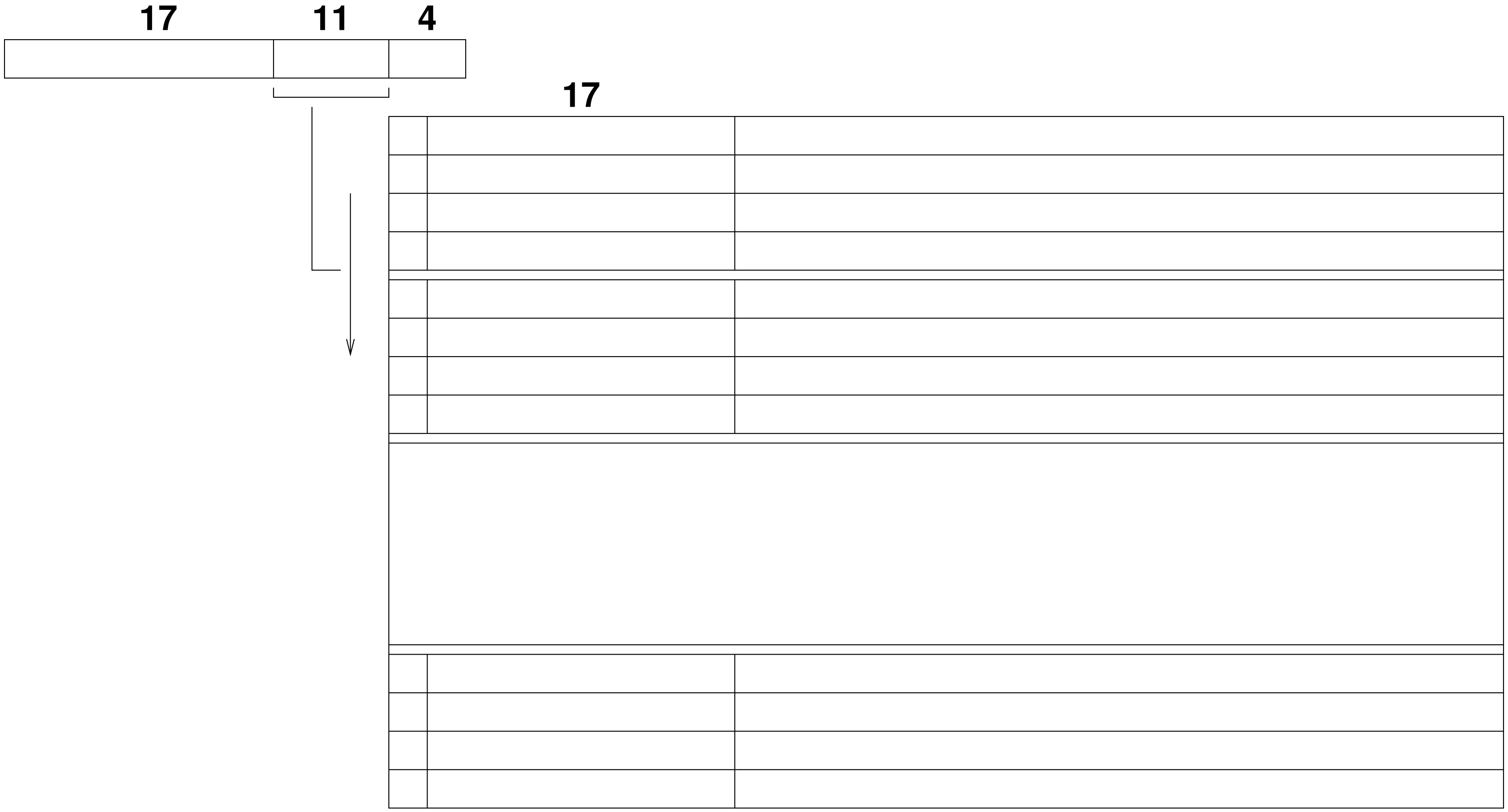
Chaque bloc en mémoire principale a sa place prédéterminé par les bit les moins sigificatifs de son numéro.

Exemple (taille du cache 128 ko, adresses à 32 bits, taille d'un bloc 16 octets) :



Cache associatif à N groupes

C'est une solution intermédiaire entre associatif et direct (ex N=4, taille 128 ko)



Cache associatif à N groupes

Ce type de cache est moins sensible à certains motifs d'accès

Le cache associatif et le cache direct sont des cas spéciaux de ce cache (N = 1 pour le cache direct, N = M pour le cache associatif, où M est le nombre de lignes)

Une bonne valeur de N est 2–4.

N est parfois appelé l'"associativité du cache"

Choix des paramètres

Taille d'un bloc

Le surcoût dû au circuits supplémentaires dépend du nombre de blocs

Pour une taille fixe de cache, il est donc préférable d'avoir des blocs de taille importante

Mais cela augmente la probabilité qu'une partie du bloc sera inutile

Une bonne valeur est 4 ou 8 mots (de 32 ou 64 bits selon la machine)

Associativité

Le coût augmente de manière considérable avec l'associativité

Il est rare pour un programme d'avoir besoin d'une associativité supérieure à 4

Une associativité de 2 est le minimum

Une bonne valeur est 2 ou 4

Influence du cache sur la programmation

C'est une bonne idée d'aligner le code et les données sur les lignes de cache

Un sous programme et un tableau devrait donc commencer sur une adresse multiple de la taille d'un bloc

Le compilateur s'occupe du code et malloc s'occupe des données

Le compilateur s'occupe également des données statiques

Il faut éviter des boucles et des tableaux dont la taille dépasse celle du cache (pour un cache associatif, c'est la seule considération)

Pour un cache direct, il faut éviter une distance entre appelant et appelé multiple de la taille du cache, ainsi que la manipulation simultanée de données à distance multiple de la taille du cache

Ces considération influencent rarement la programmation d'applications, mais souvent la programmation système (compilateurs, gestionnaires de la mémoire, ...)

Stratégie d’écriture du cache

Écriture immédiate

Chaque écriture provoque une écriture en mémore principale

Si le nombre d’écriture est considérable, ça peut diminuer l’efficacité du cache

C’est facile à implèmenter

Écriture différée

Écriture en mémoire prinicpale uniquement en cas de besoin (changement de processus, écriture sur disque, synchronisation)

Plus difficile à implémenter