



S21

Hiérarchie mémoire

Cédric Bastoul

cedric.bastoul@unistra.fr

Université de Strasbourg

Question préliminaire

À votre avis, combien de types de mémoire possède l'ordinateur qui me sert à faire ma présentation ?

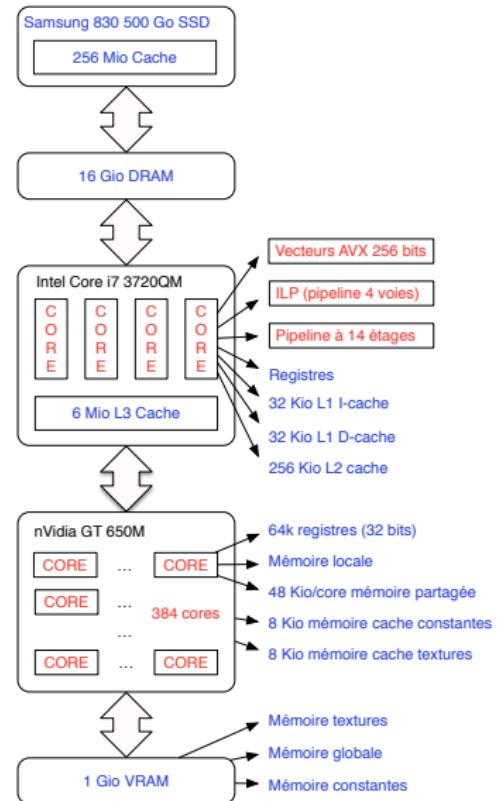
17 !!!

Analyse d'une architecture moderne

Cet ordinateur :



- 2.7 milliards de transistors
- **17 types de mémoire**
- 5 types de **parallélisme**
- Au moins 4 modèles de programmation + APIs



Objectifs de ce cours

- ▶ Comprendre le système mémoire d'un ordinateur
 - ▶ Ses principes
 - ▶ Ses technologies
 - ▶ Son organisation hiérarchique
- ▶ Comprendre l'impact du système mémoire
 - ▶ Sur les performances
 - ▶ Sur les coûts
 - ▶ Sur la durée de vie de l'information
- ▶ En tirer les conséquences
 - ▶ Pour écrire des programmes performants
 - ▶ Pour monter un ordinateur efficace
 - ▶ Pour limiter les pertes de données en cas de panne

Importance du système mémoire

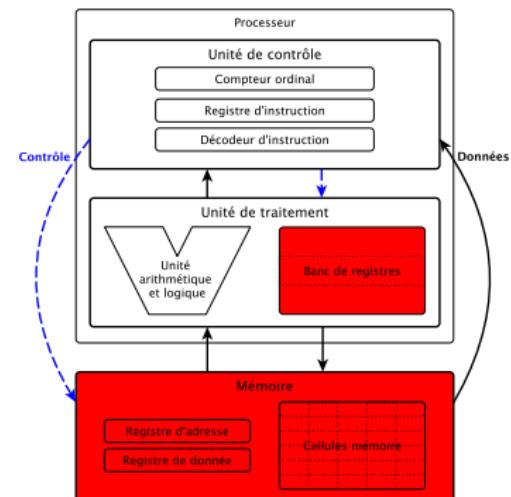
- ▶ Chaque instruction nécessite au moins un accès mémoire
 - ▶ Pour charger l'instruction
- ▶ À cela s'ajoutent les instructions de type **load et store**
 - ▶ 1/3 des instructions d'un programme en moyenne
 - ▶ Typiquement deux **load** pour chaque **store**
 - ▶ Exemple pour $A = B + C$:

```
LOAD R0, B      ; Charge B dans le registre R0  
LOAD R1, C      ; Charge C dans le registre R1  
ADD  R2, R0, R1; Place R0+R1 dans R2  
STORE R2, A     ; Stocke le contenu de R2 dans A
```

- ▶ Élément clé de la performance d'une application
 - ▶ *Compute bound* si le temps est borné par la vitesse du processeur (cryptage, vidéo, simulation etc.)
 - ▶ *Memory bound* si il est borné par la vitesse du système mémoire (recherche, big data, gestion etc.)

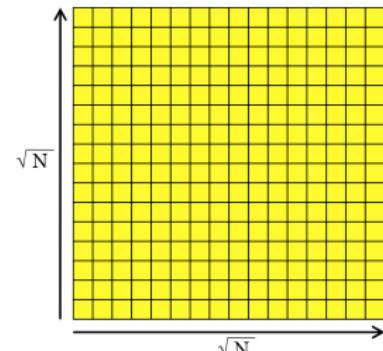
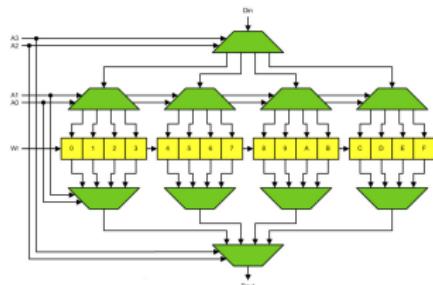
Vision idéale du système mémoire

- ▶ Pour l'instant, on a eu une vision simple du système mémoire
- ▶ Deux espaces :
 - ▶ Le banc de registres – petit nombre de cellules adressables
 - ▶ La mémoire centrale – très grand nombre de cellules adressables
- ▶ Accès à la mémoire centrale plus lent que l'accès aux registres (passage par un bus, par des registres intermédiaires...)



Limites physiques

- ▶ Temps de propagation d'un signal
 - ▶ $3 \text{ GHz} \equiv \sim 10 \text{ cm par cycle (0.33 ns)}$
 - ▶ Délais à chaque traversée de porte logique (0.5 ns à 5+ ns)
- ▶ Temps d'accès, mémoire de taille N
 $T_{\text{accès}} = T_{\text{adresse}} + T_{\text{fil}} + T_{\text{cellule}}$
 - ▶ $T_{\text{adresse}} = C_1 \times \log(N)$: temps de traversée des portes logiques pour décoder l'adresse
 - ▶ $T_{\text{fil}} = C_2 \times \sqrt{N}$: temps de traversée des fils dans un espace mémoire 2D
 - ▶ $T_{\text{cellule}} = C_3$: temps de lecture ou d'écriture dans une cellule
- ▶ Dépend de N : plus une mémoire est grande, plus elle est lente !



Hiérarchie mémoire

Motivation pour une hiérarchie mémoire

- Ce qu'on voudrait avoir idéalement :
 - ▶ Une seule mémoire de grande capacité, fonctionnant à la même vitesse que le processeur, pour un coût raisonnable
- La réalité :
 - ▶ Plus une mémoire est grande, plus elle est lente
 - ▶ Plus une mémoire est grande, plus son coût par bit est bas
 - ▶ Plus une mémoire est rapide, plus son coût par bit est élevé
- Une solution :
 - ▶ Créer un système mémoire composé de deux mémoires de types différents, l'une petite mais rapide, l'autre grande mais lente, et donnant l'illusion **la plupart du temps** d'une mémoire grande et rapide
 - ▶ Étendre cette idée à une hiérarchie de plusieurs niveaux
 - ▶ En profitant du fait que les accès ne sont pas aléatoires

Principe de localité des données

localité temporelle : si une information est accédée en mémoire, il y a de fortes chances pour qu'elle le soit une nouvelle fois prochainement

localité spatiale : si une information est accédée en mémoire, il y a de fortes chances pour que celles situées dans son voisinage le soient prochainement

- ▶ Toute la hiérarchie mémoire est construite sur ce principe
- ▶ Conséquence du modèle de von Neumann
 - ▶ Suite d'instructions en mémoire
 - ▶ Rappels de séquences d'instructions (boucles et fonctions)
 - ▶ Suites de données en mémoire (tableaux et structures)

Hiérarchie mémoire

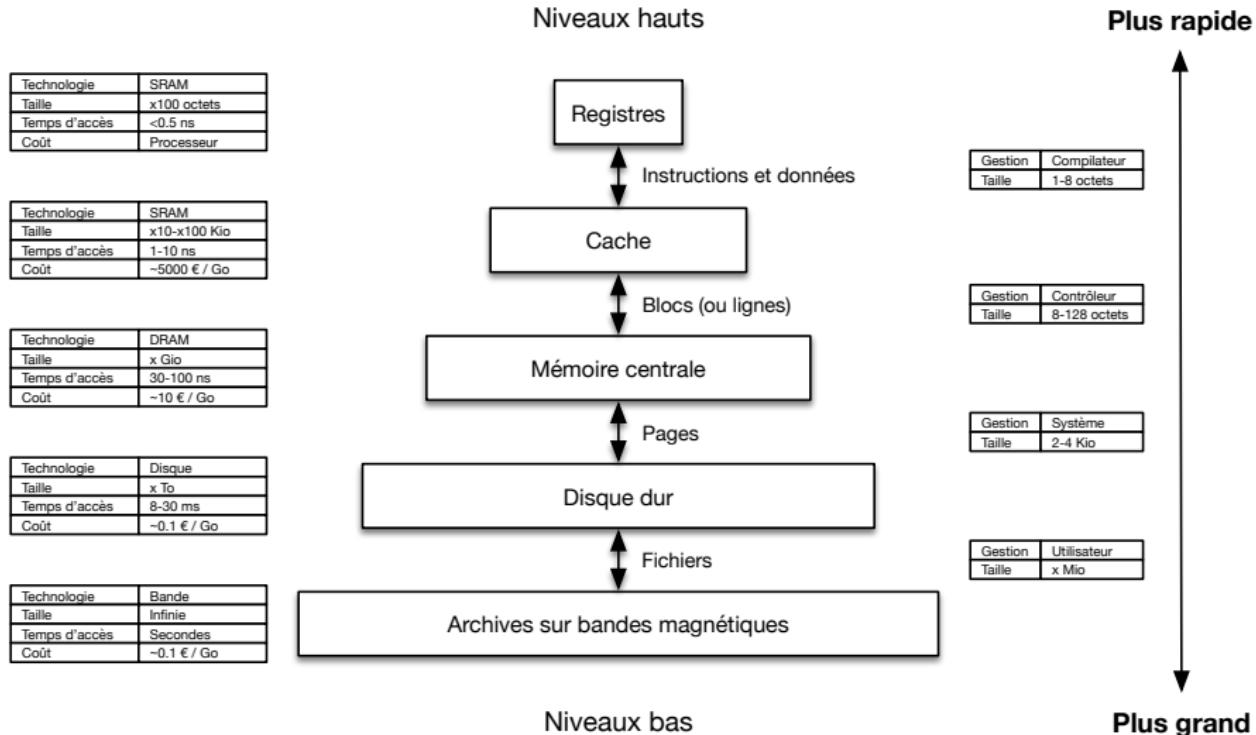


Illustration : accès en ligne ou colonne ?

Initialisation d'un tableau

[code]

```
#include <stdio.h>
#include <stdlib.h>
#include <time.h>
#define N 10000

int main() {
    double (*tab)[N] = malloc(sizeof(double[N][N]));
    clock_t time;

    time = clock();
    for (size_t i = 0; i < N; i++)
        for (size_t j = 0; j < N; j++)
            tab[j][i] = 0.;
    time = clock() - time;

    printf("Temps d'initialisation : %g s\n", ((double)time) / CLOCKS_PER_SEC);
    free(tab);
    return 0;
}
```

- ▶ Compilez et exéutez le code, notez le temps d'initialisation
- ▶ Changez l'instruction d'initialisation par « `tab[i][j] = 0.;` »
Le temps d'initialisation est-il le même ?

Technologies mémoire

Types de technologies mémoire

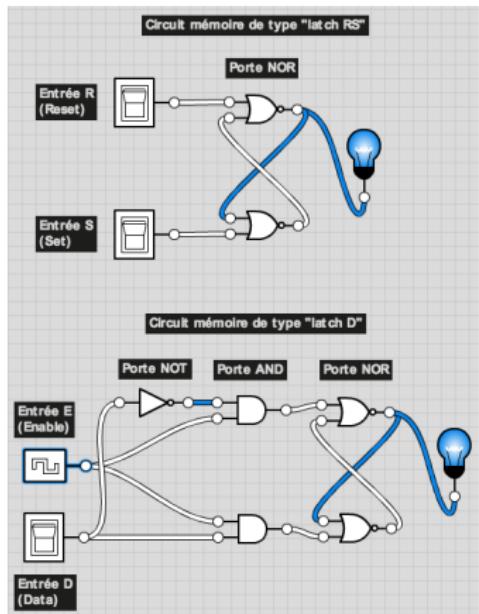
- ▶ Différentes technologies mémoire coexistent dans une architecture moderne
 - ▶ Chacune a ses forces et ses faiblesses
- ▶ La mémoire volatile perd l'information sans alimentation électrique
 - ▶ Static Random Access Memory (SRAM)
 - ▶ Dynamic Random Access Memory (DRAM)
- ▶ La mémoire persistante conserve l'information sans alimentation électrique
 - ▶ Read-Only Memory (ROM)
 - ▶ Electrically-Erasable Programmable Read-Only Memory (EEPROM ou mémoire « flash »))
 - ▶ Disques magnétiques
 - ▶ Bandes magnétiques
- ▶ Un système les utilise toutes pour différentes tâches

Circuit mémoire de base : le « latch »

- ▶ Circuit à deux états stables pouvant stocker de l'information
- ▶ Sa sortie et son état courant dépendent de son entrée et de son état précédent
- ▶ Latch RS (Reset-Set)
 - ▶ État maintenu si Reset et Set bas
 - ▶ État haut si Set haut et Reset bas
 - ▶ État bas si Set bas et Reset haut
 - ▶ Set et Reset hauts interdit
- ▶ Latch D (Data)

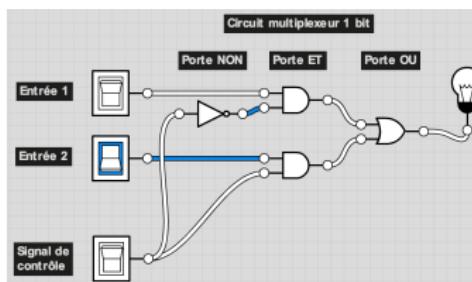
Étend le RS pour éviter l'état interdit : l'état de l'entrée Data est enregistré quand l'entrée Enable est à haut

<http://logic.ly/demo/>
(ne jouez pas trop ;-) !)



Circuit « multiplexeur »

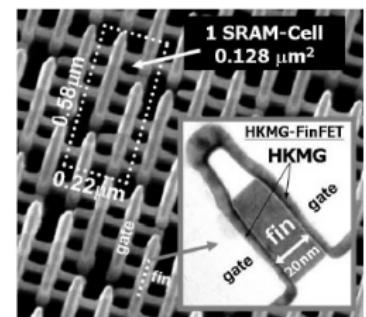
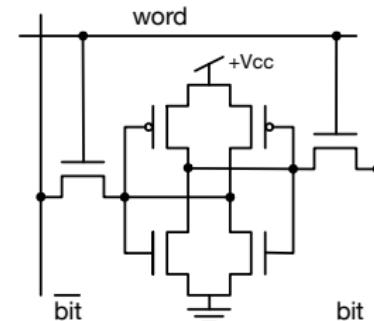
- ▶ Circuit permettant de sélectionner un signal parmi deux en utilisant un signal de contrôle
- ▶ Peut s'étendre : n signaux de contrôle permettent de sélectionner un signal parmi 2^n



- ▶ Utilisé pour « décoder » les adresses mémoire
 - ▶ Signaux de contrôle = adresse mémoire
 - ▶ Signal en sortie = « word line » (horizontal sur les dessins) permet de sélectionner la bonne cellule mémoire

Static RAM (SRAM)

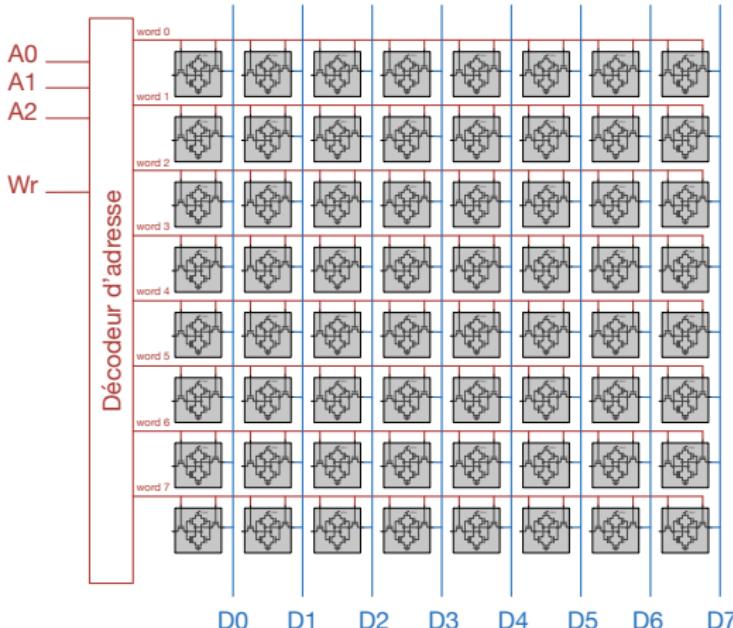
- ▶ RAM : Random Access Memory
 - ▶ Temps d'accès constant quelle que soit l'adresse ou l'information accédée avant
 - ▶ (contrairement aux bandes magnétiques qui sont un support séquentiel par ex.)
- ▶ Static : conserve son état tant qu'elle est alimentée électriquement
- ▶ Cellules mémoire proches du latch
 - ▶ Nécessitent 6 transistors par bit
 - ▶ Faible densité
 - ▶ Très rapide
 - ▶ Très chère
 - ▶ Forte consommation d'énergie
- ▶ Pour registres, buffers, caches etc.



A bird's-eye view of $0.128\mu\text{m}^2$ FinFET SRAM cells
(post silicide formation)

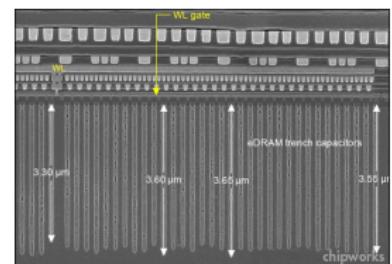
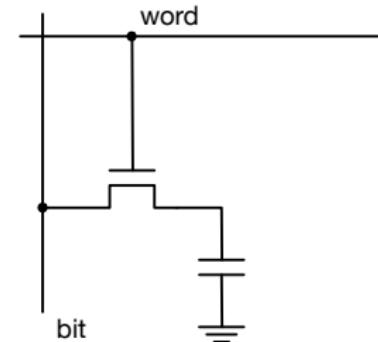
Organisation typique de la SRAM

- ▶ Tableau de cellules
- ▶ Ici 8 mots de 8 bits
- ▶ L'adresse entière est envoyée au décodeur
- ▶ Le décodeur active la ligne correspondante
- ▶ En fonction du signal Wr la ligne est lue ou écrite



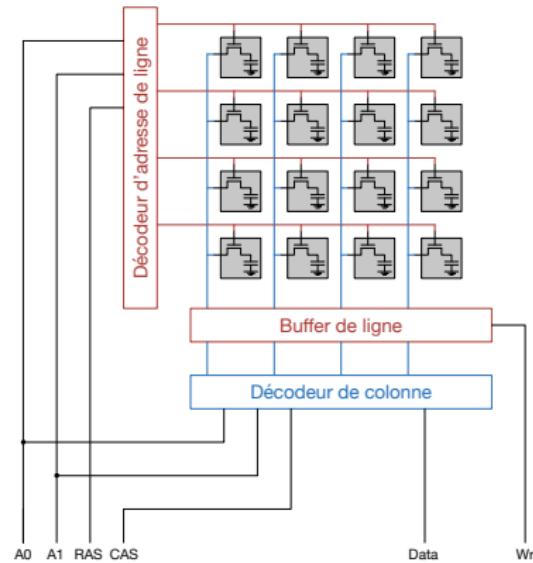
Dynamic RAM (DRAM)

- ▶ Solution plus dense que la SRAM :
 - ▶ Utiliser un simple condensateur !
 - ▶ Et un transistor pour en contrôler l'accès
 - ▶ Deux états : chargé (1) ou déchargé (0)
- ▶ Dynamic : état instable, doit être rafraîchi
 - ▶ Condensateur déchargé en 10 à 100 ms
 - ▶ Doit être rechargé périodiquement
 - ▶ Par lecture et réécriture de l'information
- ▶ Cellules mémoire simples
 - ▶ Haute densité
 - ▶ Coût réduit
 - ▶ TRÈS LENTES
- ▶ Mémoire centrale, vidéo etc.



Organisation typique de la DRAM

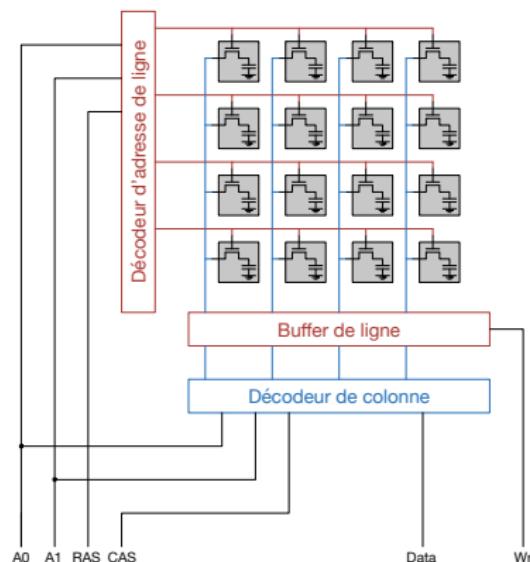
- ▶ Tableau 2D de cellules
- ▶ Ici 4×4 bits de DRAM
- ▶ L'adresse est décomposée, reçue et décodée en deux parties
 - ▶ Réduit le nombre de connecteurs
- ▶ Fonctionnement d'un accès :
 - ▶ Envoi adresse de ligne et RAS (Row Address Strobe)
 - ▶ Envoi adresse de colonne et CAS (Column Address Strobe)
 - ▶ Opération de lecture ou écriture



Exemple d'accès à une DRAM

Accès en deux temps :

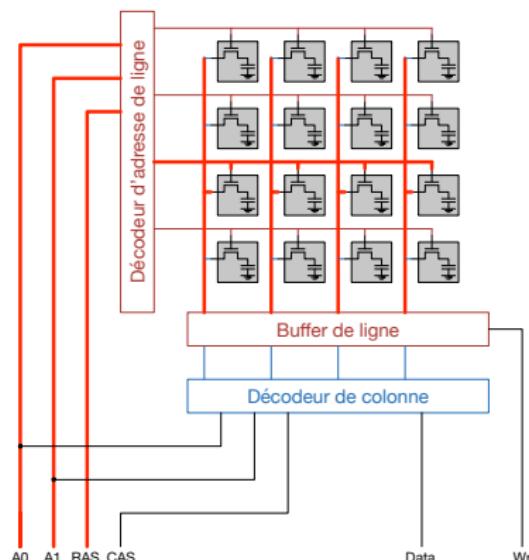
- ➊ Envoi d'une adresse de ligne et du signal RAS
 - ▶ Toute une ligne est activée
 - ▶ Elle est mise en buffer de ligne
 - ▶ La ligne est rafraîchie
- ➋ Envoi d'une adresse de colonne et du signal CAS
 - ▶ Le bit recherché est lu dans le buffer de ligne



Exemple d'accès à une DRAM

Accès en deux temps :

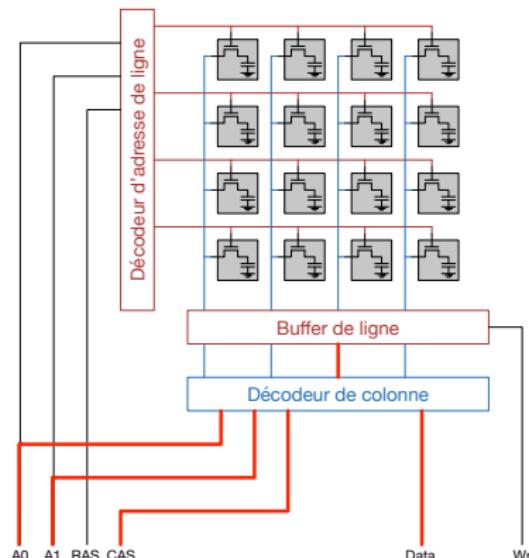
- ➊ Envoi d'une adresse de ligne et du signal RAS
 - ▶ Toute une ligne est activée
 - ▶ Elle est mise en buffer de ligne
 - ▶ La ligne est rafraîchie
- ➋ Envoi d'une adresse de colonne et du signal CAS
 - ▶ Le bit recherché est lu dans le buffer de ligne



Exemple d'accès à une DRAM

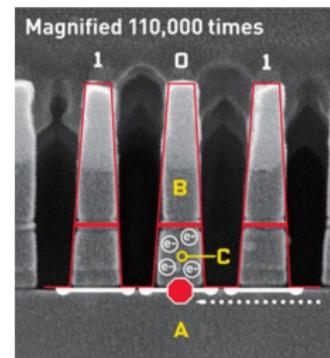
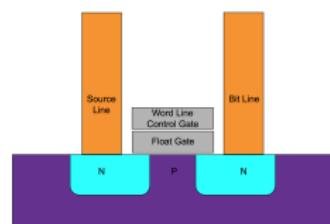
Accès en deux temps :

- ➊ Envoi d'une adresse de ligne et du signal RAS
 - ▶ Toute une ligne est activée
 - ▶ Elle est mise en buffer de ligne
 - ▶ La ligne est rafraîchie
- ➋ Envoi d'une adresse de colonne et du signal CAS
 - ▶ Le bit recherché est lu dans le buffer de ligne



Solid-state drive

- ▶ Mémoire de masse utilisant des circuits intégrés
 - ▶ Mémoire de type NAND flash
 - ▶ Nombre limité de cycles d'écriture
 - ▶ 1 à 3 bits par cellules (SLC, MLC, TLC)
Plus de bits → moins de durée de vie
 - ▶ Un contrôleur gère l'usure des circuits
- ▶ Information persistante même sans alimentation électrique
- ▶ Pas de partie mobile (« solid »)
 - ▶ Haute densité
 - ▶ Coût réduit
 - ▶ Durée de vie réduite
 - ▶ TRÈS TRÈS LENTS
- ▶ Utilisé comme mémoire secondaire



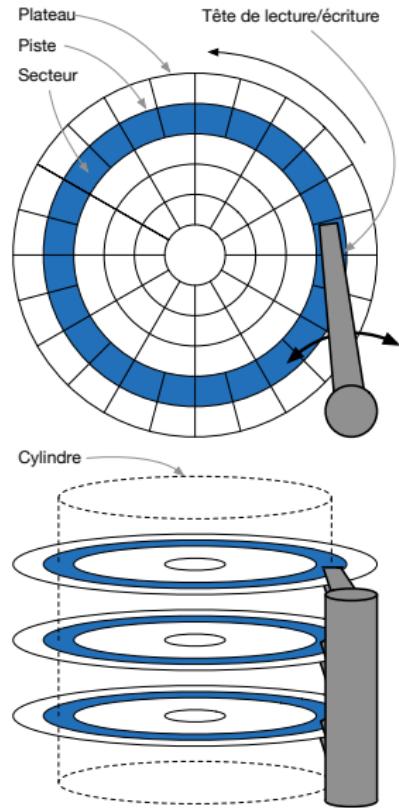
Disque dur magnétique

- ▶ Mémoire de masse utilisant des plateaux rigides recouverts d'une surface magnétique
 - ▶ Plateaux tournant à haute vitesse, jusqu'à 15000 RPM
 - ▶ Têtes capables de lire et d'écrire
- ▶ Information persistante même sans alimentation électrique
- ▶ Comprend des éléments mécaniques
 - ▶ Haute densité
 - ▶ Coût réduit
 - ▶ Durée de vie réduite
 - ▶ TRÈS TRÈS TRÈS LENTS
- ▶ Utilisé comme mémoire secondaire



Organisation typique d'un disque dur

- ▶ Organisation de la surface :
 - ▶ La surface d'un plateau est divisée en anneaux concentriques appelés *pistes*
 - ▶ Chaque piste est divisée en *secteurs*
 - ▶ Les pistes éloignées du centre ont plus de secteurs : pistes de même nombre de secteurs regroupées en *zones*
 - ▶ L'ensemble des $n^{\text{èmes}}$ pistes de chaque plateau forme un *cylindre*
- ▶ Fonctionnement d'un accès :
 - ▶ Positionnement du bras pour que la tête soit sur la bonne piste
 - ▶ Rotation pour atteindre le bon secteur
 - ▶ **Le secteur est l'unité minimale d'information sur un disque dur**



Performance d'un disque dur

- ▶ Deux principaux facteurs de performance
- ▶ Temps de recherche
 - ▶ Temps pris par le bras pour se déplacer vers la bonne piste
 - ▶ Dépend du cylindre accédé avant (encore la localité !)
 - ▶ Au pire, 20 ms, en moyenne 6-9 ms
- ▶ Temps de rotation
 - ▶ Temps pris pour arriver au bon secteur
 - ▶ Au pire une rotation complète
 - ▶ À 7200 RPM, au pire un tour 8 ms, en moyenne un demi tour 4 ms
- ▶ Accéder à un secteur peut prendre entre 10 et 30 ms
- ▶ Taille d'un secteur typique : 4096 octets (physique) mais le contrôleur de disque permet 512 octets (logique)

Point sur les technologies mémoire

- ▶ Trois principales familles aujourd'hui : SRAM, DRAM et disques durs
- ▶ Comparaisons (2014) :

Technologie	SRAM	DRAM	Disque dur
Temps d'accès	1-10 ns	30-100 ns	8-30 ms
Capacité	x100 Kio	x1 Gio	x1 To
Coût	~5000 €/Go	~10 €/Go	~0.1 €/Go
Tendance temps d'accès	Stable depuis 2004	×2 en 7-8 ans	×2 en 14-15 ans
Tendance capacité	×2 en 3 ans	×2 en 2-3 ans	×2 en 3 ans

- ▶ À comparer avec un processeur : 3 GHz → 0.33 ns
- ▶ Si un cycle prenait une seconde :
 - ▶ Accès SRAM : 3-30 secondes
 - ▶ Accès DRAM : 1 min 30 à 5 minutes
 - ▶ Accès disque dur : 1 à 3 ans !!!

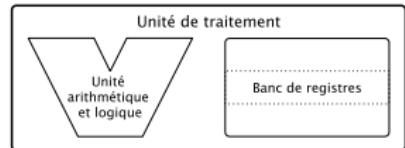
Éléments de la hiérarchie mémoire

1. Banc de registres
2. Mémoire cache
3. Mémoire centrale
4. Mémoire de masse

1. Banc de registres

Banc de registres

Cellules mémoire internes à l'unité de traitement du processeur, pouvant en général être ciblées par les instructions

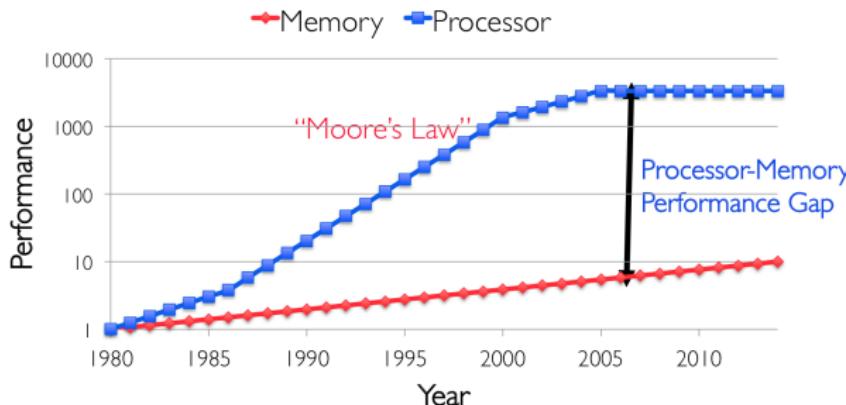


- ▶ Sommet absolu de la hiérarchie mémoire
- ▶ Technologie SRAM
- ▶ Fonctionne à la même vitesse que le processeur
- ▶ Nombre et types dépendant de l'architecture
Exemple : 16 registres généraux 64 bits sur x86-64
- ▶ Accessibles en assembleur mais pas à haut niveau
- ▶ Le compilateur tente d'optimiser leur utilisation

2. Mémoire cache

Le mur de la mémoire

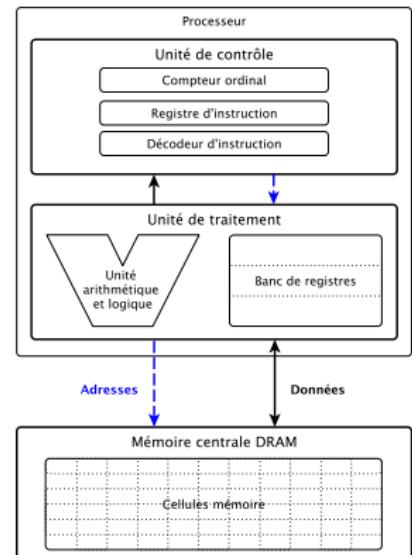
- ▶ Les processeurs sont plus rapides que les mémoires
- ▶ Mais c'est encore pire que ça !
 - ▶ La performance des processeurs a augmenté de 50% / an
 - ▶ Celle de la DRAM de seulement 7% / an
 - ▶ Dessin ci-dessous pour un seul cœur !



[Source : Patterson]

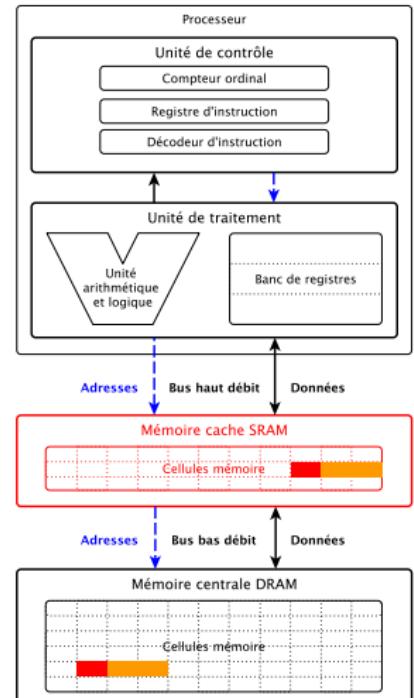
Franchir le mur de la mémoire

- ▶ On veut une mémoire de la taille de la DRAM et de la performance de la SRAM
- ▶ Impossible avec la vision idéale de l'architecture
- ▶ Idée : introduire une **mémoire cache** entre le processeur et la mémoire centrale
 - ▶ Utilisant une technologie plus efficace
 - ▶ De taille modeste pour limiter les coûts et maximiser la performance
 - ▶ Contenant les données probablement les plus utiles au processeur



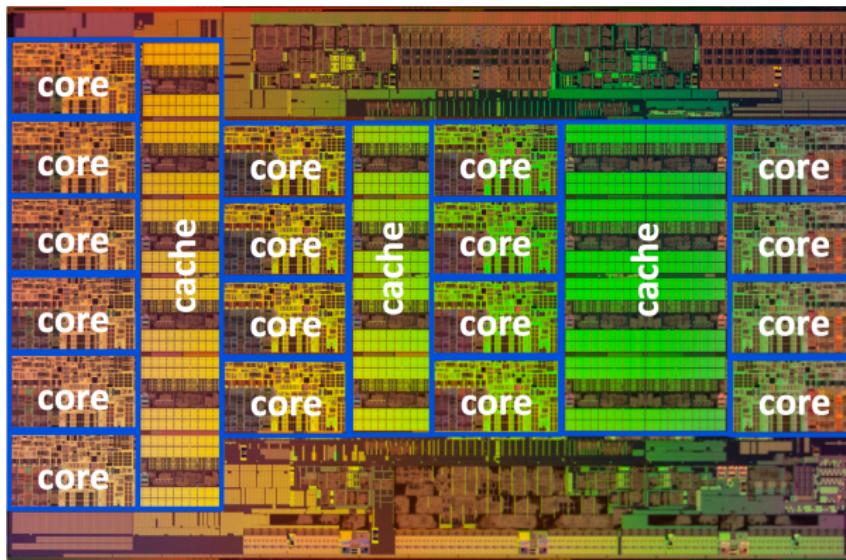
Mémoire cache

- ▶ Un cache stocke temporairement des copies d'information provenant d'une autre mémoire pour diminuer le temps d'accès à ces informations
- ▶ Quand le processeur accède une information en mémoire centrale
 - ▶ On lit tout un « bloc » d'informations consécutives et on le copie en cache
 - ▶ Tant que les prochains accès sont dans ce bloc, on utilise le cache
 - ▶ Si un accès est dans un bloc hors du cache, on le cherche et on l'y met
- ▶ Si information en cache : *cache hit*
- ▶ Sinon : *cache miss* ou « défaut »



Exemple (Intel Haswell – 2014)

- ▶ Les premiers niveaux de cache sont sur la puce du processeur pour maximiser les performances
- ▶ Espace occupé (et donc coût) important



Principes des mémoires cache

- ▶ Le cache contient une partie des données et/ou des instructions d'un programme
 - ▶ Celles les plus récemment accédées
 - ▶ Il manipule des blocs d'information
 - ▶ Il ne peut pas tout contenir !
- ▶ Il repose entièrement sur le principe de localité
- ▶ Il peut y en avoir plusieurs niveaux (typiquement 3)
- ▶ Nombreux éléments paramétrables pour chaque niveau
 - ▶ Taille du cache, taille des blocs
 - ▶ Séparation des instructions et des données ou non
 - ▶ Organisation des blocs dans le cache
 - ▶ Politique de remplacement des blocs
 - ▶ Stratégie à suivre en cas d'écriture

Bloc de données

- ▶ Le bloc est l'unité d'information manipulée par un cache
 - ▶ Composé d'informations d'adresses consécutives
 - ▶ Identifié de manière unique par la partie commune des adresses des informations dans le bloc, appelée *tag*
- ▶ Les blocs sont stockés dans des *slots* contenant :
 - ▶ Le *valid bit* V indiquant si le slot contient un bloc ou non
 - ▶ Le *dirty bit* D indiquant si l'information a été modifiée ou non
 - ▶ Le *tag* identifiant le bloc
 - ▶ Le bloc d'information



- ▶ Lors d'un accès, on compare le début de l'adresse voulue au tag, s'ils sont égaux c'est que l'information est en cache

Organisation des blocs dans le cache

- ▶ Le cache est organisé en *sets* qui définissent l'ensemble de slots où un bloc peut être stocké
- ▶ Le tag définit dans quel set un bloc peut être stocké
- ▶ Cache *direct-mapped* si autant de sets que de slots
- ▶ Cache *fully-associative* si un seul set pour tous les slots
- ▶ Cache *n-way set-associative* si chaque set contient *n* slots



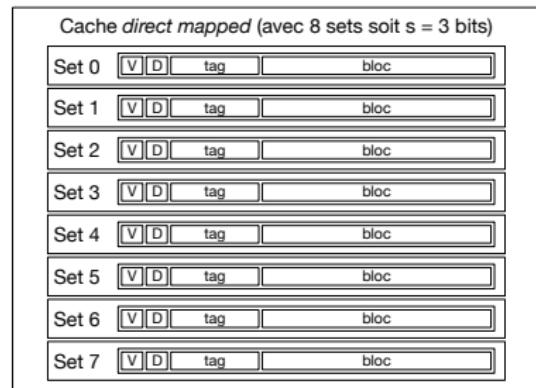
Cache *direct-mapped*

- ▶ Dispose de nombreux sets
- ▶ Un set ne peut contenir qu'un bloc
- ▶ Une partie de l'adresse (*indice de set*) détermine dans quel set un bloc va aller
- ▶ Solution très simple à réaliser
 - ▶ Mais les blocs ont beaucoup de chance d'entrer en conflit
 - ▶ Peu performant en pratique

Découpage d'une adresse de a bits



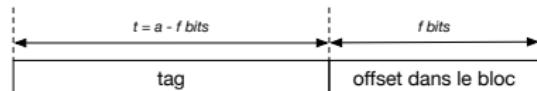
Exemple



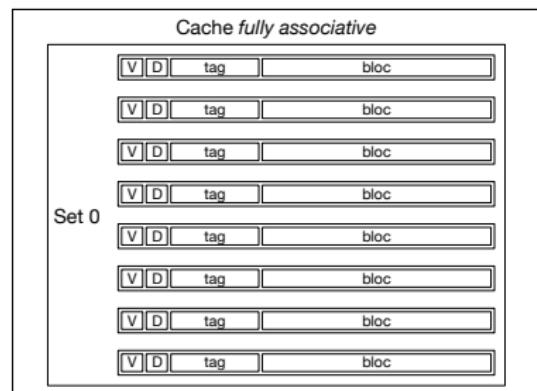
Cache *fully-associative*

- ▶ Dispose d'un seul set
- ▶ Un bloc peut prendre n'importe quelle place dans ce set
- ▶ Très complexe à réaliser
 - ▶ Pas de conflit entre les blocs
 - ▶ Plus il y a de blocs, plus la gestion est complexe
 - ▶ Vérifier la présence d'un bloc
 - ▶ Choisir un bloc à remplacer
 - ▶ Peu intéressant en pratique, réservé aux niveaux bas, ex: « *TLB* » (étudié en S31) ou caches logiciels

Découpage d'une adresse de a bits



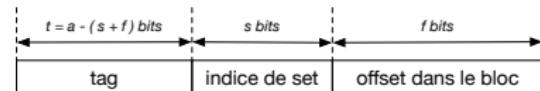
Exemple



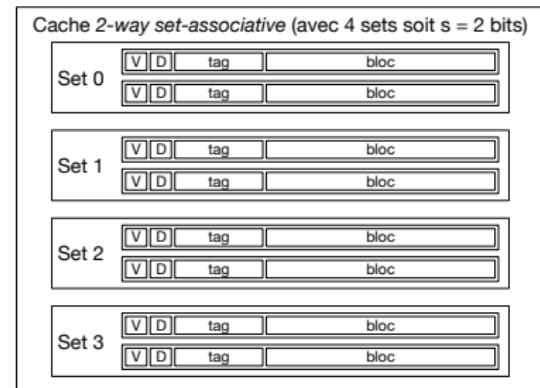
Cache *n-way set-associative*

- ▶ Intermédiaire entre *direct mapped* et *fully associative*
 - ▶ Plusieurs sets
 - ▶ Plusieurs blocs (*ways*) par set
- ▶ Une partie de l'adresse (*indice de set*) détermine dans quel set un bloc peut aller
- ▶ Un bloc peut prendre n'importe quelle place dans ce set
- ▶ Solution très performante, la plus utilisée actuellement
 - ▶ $n = 2, 4$ ou 8 ways typiques
 - ▶ Plus on baisse dans la hiérarchie, plus n augmente

Découpage d'une adresse de a bits



Exemple



Politique de remplacement des blocs

- ▶ Inutile avec un cache direct-mapped
- ▶ Avec les caches associatifs, nécessité d'un algorithme de remplacement pour déterminer quel bloc doit céder sa place quand un nouveau doit entrer dans le même set
 - ▶ Random : aléatoirement, mauvais pour la localité
 - ▶ FIFO (First-In-First-Out), mauvais pour la localité
 - ▶ LFU (Least Frequently Used) : bloc le moins accédé, très compliqué à gérer
 - ▶ LRU (Least Recently Used) : bloc qui n'a pas été accédé depuis le plus longtemps, compliqué si plus de deux voies
 - ▶ Pseudo-LRU : un bit indique le dernier bloc accédé dans le set, choix aléatoire parmi les autres
 - ▶ Solution la plus utilisée actuellement

Types de défauts de cache (*cache miss*)

- ▶ *Compulsory miss*
 - ▶ Lorsque le bloc accédé n'a jamais été accédé avant
 - ▶ Inévitable même avec un cache de taille infinie
 - ▶ On parle de *cold cache* lorsque le cache est vide
- ▶ *Capacity miss* (cas *fully-associative*)
 - ▶ Lorsqu'il n'y a plus de place pour mettre le bloc accédé
 - ▶ Inévitable si on accède plus de n blocs différents alors que le cache n'a que n slots
- ▶ *Conflict miss* (cas *direct-mapped* et *set-associative*)
 - ▶ Lorsqu'un bloc doit prendre la place d'un autre dans un set
 - ▶ Inévitable si on accède plus de n blocs différents avec le même indice de set alors que le set n'a que n slots
 - ▶ Une associativité trop faible favorise ce type de défaut

Stratégie à suivre en cas d'écriture

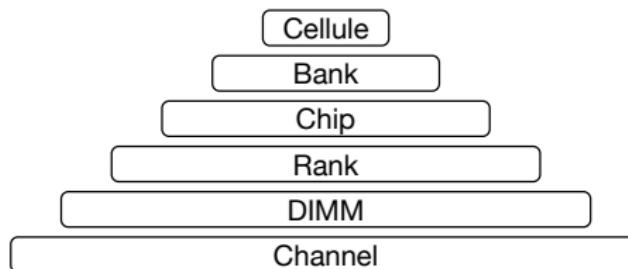
Si une information est modifiée dans le cache (cas d'un store), deux stratégies possibles

- ▶ *Write-through* : l'information est aussi écrite en mémoire
 - ▶ Pas de *dirty bit* dans ce cas
 - ▶ Facile à implémenter en matériel
 - ▶ Rapide pour les défauts en lecture (mémoire cohérente)
 - ▶ Lent en écriture
- ▶ *Write-back* : l'information n'est pas écrite en mémoire immédiatement
 - ▶ Nécessité du *dirty bit*
 - ▶ Implémentation complexe en matériel
 - ▶ Rapide en écriture
 - ▶ Remplacement d'un bloc « *dirty* » impose une écriture en mémoire centrale
 - ▶ Solution la plus utilisée actuellement

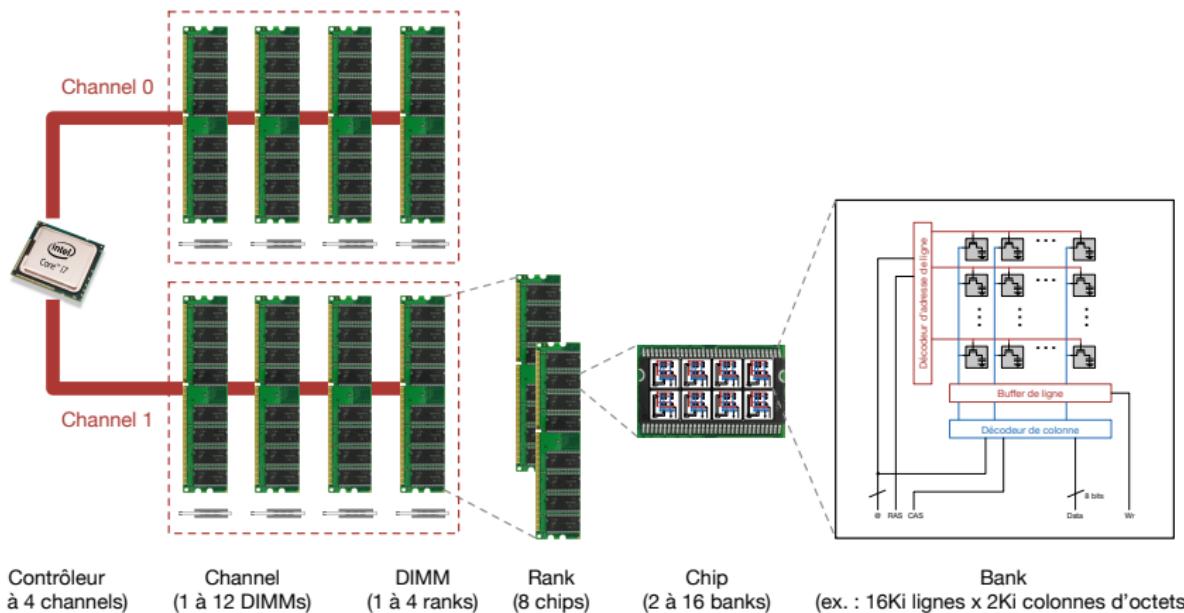
3. Mémoire centrale

Sous-système de mémoire centrale

- ▶ La mémoire centrale utilise la technologie DRAM
 - ▶ Capacité possible très supérieure à la SRAM
 - ▶ Meilleure densité que la SRAM, faible coût
 - ▶ Très lente, nécessité d'être rafraîchie (condensateurs)
- ▶ Sous-système hiérarchique et distribué contre la lenteur
 - ▶ Plusieurs éléments peuvent travailler en même temps
 - ▶ L'information est répartie sur plusieurs composants



Organisation du sous-système

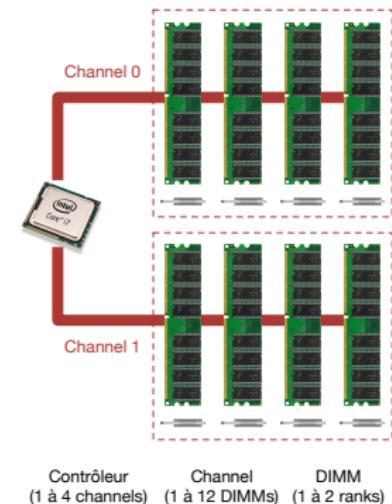


Contrôleur mémoire

- ▶ Lien entre processeur et mémoire centrale
 - ▶ Dépend du processeur et du type de mémoire
 - ▶ Répond aux requêtes d'accès aux cellules mémoire
- ▶ Assure le fonctionnement de la mémoire centrale
 - ▶ Réalise les opérations de rafraîchissement
 - ▶ Organise les requêtes pour optimiser la performance
 - ▶ Surveille la consommation d'énergie et la température (peut désactiver temporairement des éléments)
- ▶ Placé dans le processeur (solution la plus courante aujourd'hui) ou le chipset « *Northbridge* » de la carte mère
 - ▶ Processeur : plus performant, moins souple, ajoute à la surchauffe du processeur
 - ▶ Northbridge : souple (ne lie pas processeur et type de mémoire), moins performant

Channel

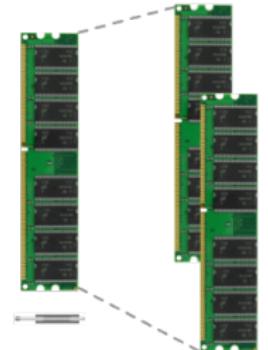
- ▶ Le contrôleur peut gérer des canaux indépendants
 - ▶ Chacun a son propre sous-contrôleur
 - ▶ Chacun dispose de son propre bus
- ▶ Améliore la bande passante
- ▶ Permet des accès simultanés
- ▶ Il faut bien placer les barettes de RAM !
 - ▶ Chaque slot est associé à un canal
 - ▶ Restrictions pour avoir le « *dual channel* »
 - ▶ Lire la documentation de la carte mère



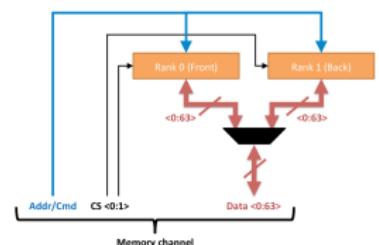
DIMM

Dual In-line Memory Module

- ▶ Plusieurs circuits mémoire « *rank* » sur une barette
- ▶ Exemple classique : un sur chaque face
- ▶ Augmente encore la capacité
- ▶ Mais interconnexion plus complexe
 - ▶ Tous les circuits reçoivent les mêmes requêtes
 - ▶ Un signal de contrôle active le bon circuit qui transmet l'information sur le canal



DIMM
(1 à 4 ranks) Rank
(8 chips)

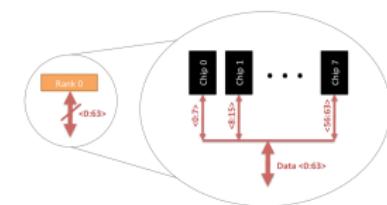


Rank

- ▶ Formé de 8 composants mémoire
- ▶ Information répartie entre les composants
 - ▶ Information de 64 bits, chaque composant en contenant 8 bits
- ▶ Vu comme un unique grand composant contenant des informations larges
- ▶ Le contrôleur n'a pas à gérer les composants individuels
- ▶ Plus rapide qu'un composant plus grand
- ▶ On ne peut pas accéder des informations plus courtes que la largeur totale



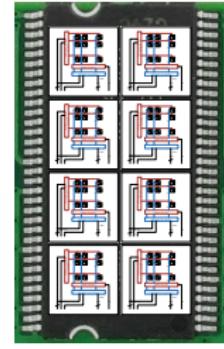
Rank
(8 chips)



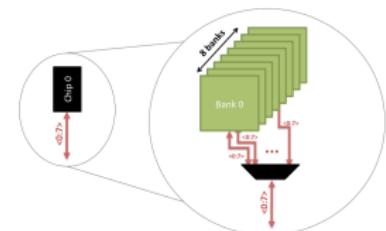
[Source : Onur Mutlu & Yoongu Kim, Carnegie Mellon Univ.]

Chip

- ▶ Formé de plusieurs banks (2 à 16)
- ▶ Vu comme un unique grand composant contenant des informations courtes
- ▶ Toutes les banks reçoivent la même requête et y répondent
- ▶ Un signal de contrôle sélectionne le bon résultat à transmettre au rank



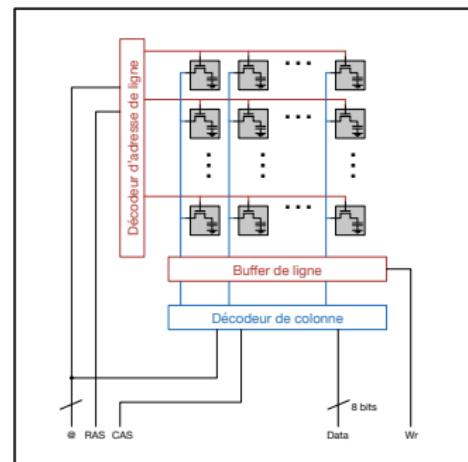
Chip
(2 à 16 banks)



[Source : Onur Mutlu & Yoongu Kim, Carnegie Mellon Univ.]

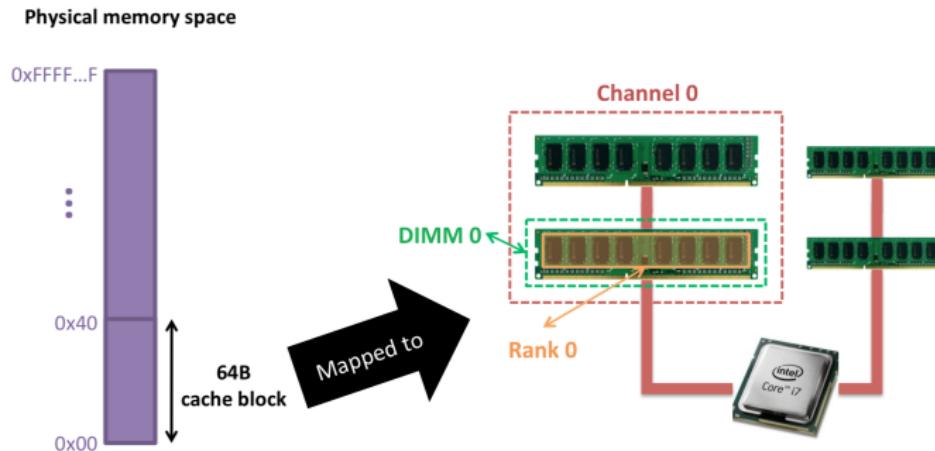
Bank

- ▶ Structure classique de DRAM qu'on a déjà étudié
- ▶ Tableau 2D de cellules pour optimiser la distance parcourue par le signal
- ▶ L'adresse est décomposée, reçue et décodée en deux parties
- ▶ Fonctionnement d'un accès :
 - ▶ Envoi adresse de ligne et RAS (Row Address Strobe)
 - ▶ Envoi adresse de colonne et CAS (Column Address Strobe)
 - ▶ Opération de lecture ou écriture



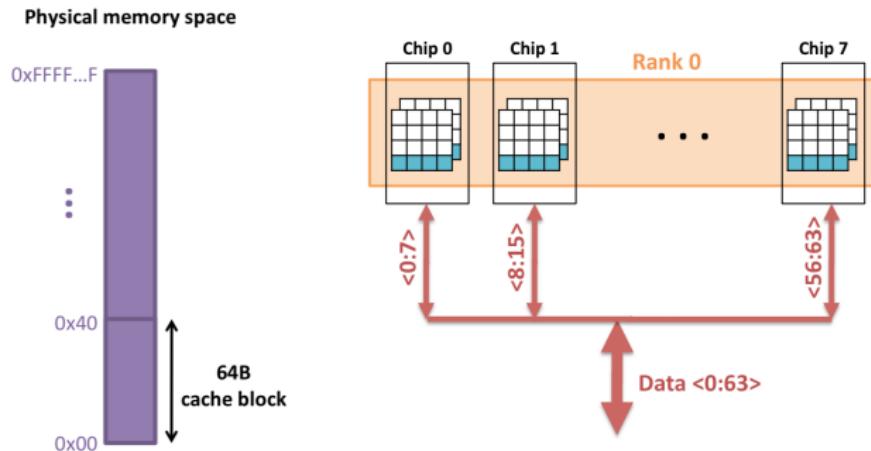
Bank
(ex. : 16Ki lignes x 2Ki colonnes d'octets)

Exemple : transfert d'un bloc de cache



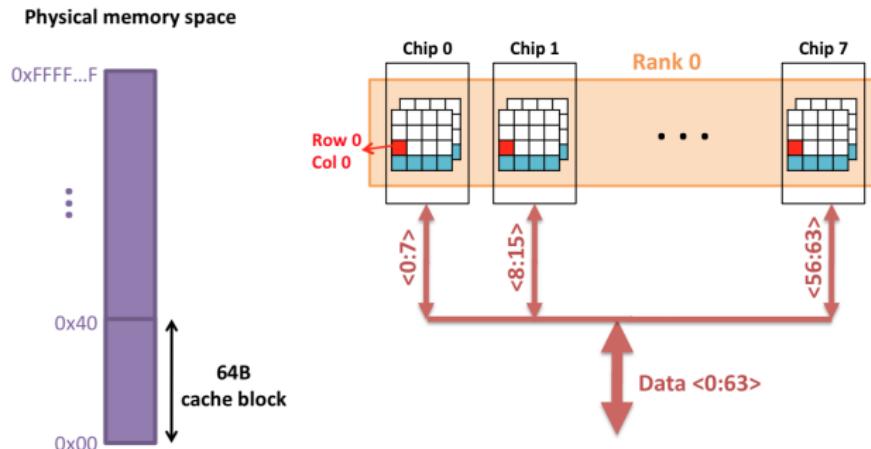
[Source : Onur Mutlu & Yoongu Kim, Carnegie Mellon Univ.]

Exemple : transfert d'un bloc de cache



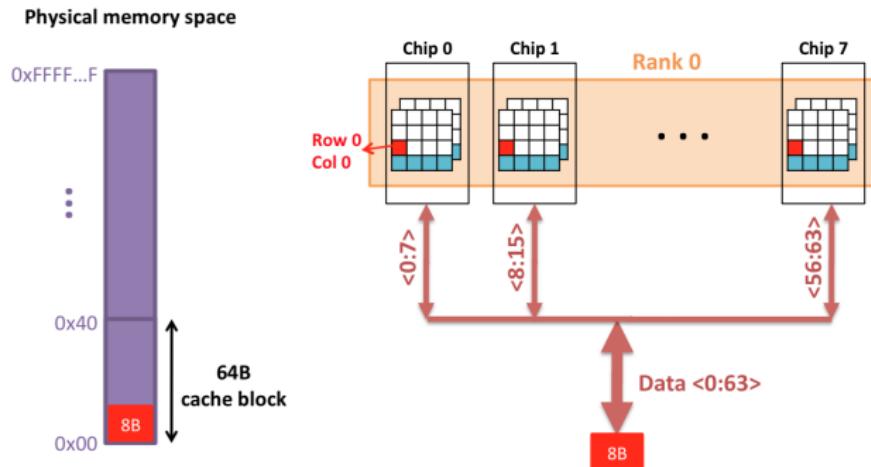
[Source : Onur Mutlu & Yoongu Kim, Carnegie Mellon Univ.]

Exemple : transfert d'un bloc de cache



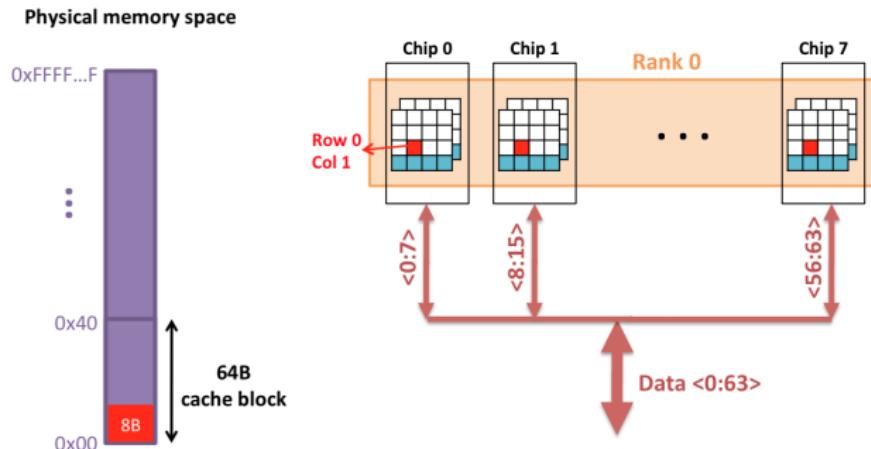
[Source : Onur Mutlu & Yoongu Kim, Carnegie Mellon Univ.]

Exemple : transfert d'un bloc de cache



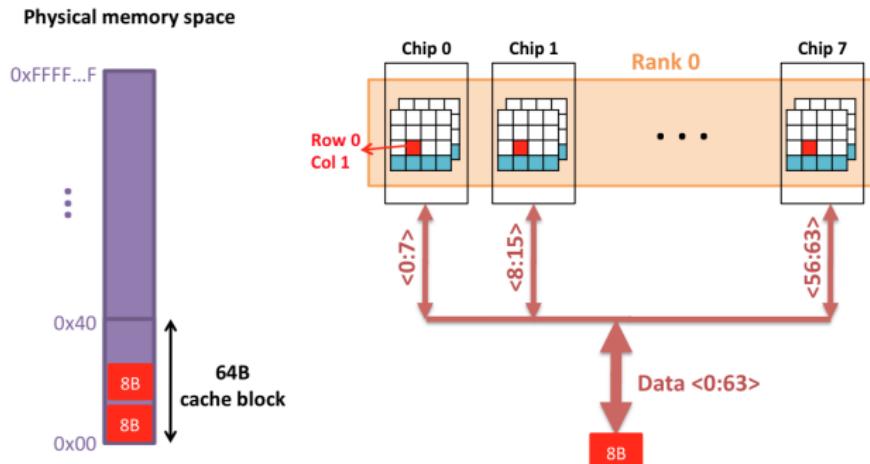
[Source : Onur Mutlu & Yoongu Kim, Carnegie Mellon Univ.]

Exemple : transfert d'un bloc de cache



[Source : Onur Mutlu & Yoongu Kim, Carnegie Mellon Univ.]

Exemple : transfert d'un bloc de cache



[Source : Onur Mutlu & Yoongu Kim, Carnegie Mellon Univ.]

- ▶ La lecture d'un bloc de 64 octets demande 8 accès
- ▶ 8 cellules sont lues en séquence

4. Mémoire de masse

Sous-système de stockage de masse

- ▶ Le stockage de masse utilise les disques durs
 - ▶ Capacité très supérieure à la mémoire centrale
 - ▶ Forte densité d'information, très faible coût
 - ▶ Information persistante sans alimentation électrique
 - ▶ Extrêmement lent
- ▶ Environnement personnel typique : disque dur unique
 - ▶ Attention aux sauvegardes !
- ▶ Environnement professionnel typique : plusieurs disques
 - ▶ Améliorer les performances
 - ▶ Disposer d'une plus grande capacité
 - ▶ Garantir le fonctionnement en cas de panne
 - ▶ Éviter la perte de données

RAID

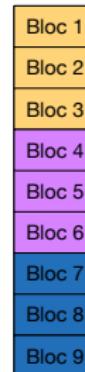
Redundant Array of Independent Disks

- ▶ Rassemble plusieurs disques en une seule unité logique
 - ▶ Utilisation transparente pour l'utilisateur
 - ▶ Gestion par logiciel (système d'exploitation ou application dédiée) ou matériel (carte contrôleur RAID)
- ▶ Objectifs :
 - ▶ Disposer d'unités de capacité supérieure à un disque seul
 - ▶ Construire des unités plus performantes par l'utilisation de plusieurs disques en parallèle
 - ▶ Sécuriser les données en affectant certains disques ou parties de disques à la redondance d'information
- ▶ Trois niveaux couramment utilisés :
 - ▶ RAID 0 : entrelacement de disques « *striping* »
 - ▶ RAID 1 : disques redondants « *mirroring* »
 - ▶ RAID 5 : entrelacement et parité répartie

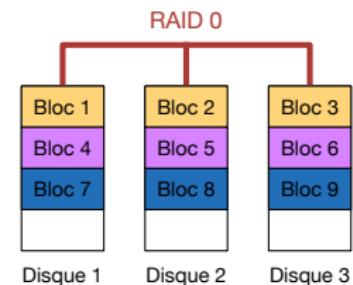
RAID 0 « striping »

- ▶ Espace logique découpé en bandes entrelacées sur les disques physiques
- ▶ Taille : celle du plus petit disque multipliée par le nombre de disques
 - ▶ Quand le plus petit disque est plein, on ne peut plus répartir l'information
 - ▶ Préférer des disques de même taille !
- ▶ Améliore la performance en distribuant la charge sur plusieurs disques
- ▶ Si un seul disque tombe en panne, toutes les données sont perdues
- ▶ Pour les besoins de performance où l'intégrité des données n'est pas critique

Organisation logique



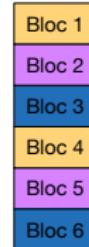
Organisation physique



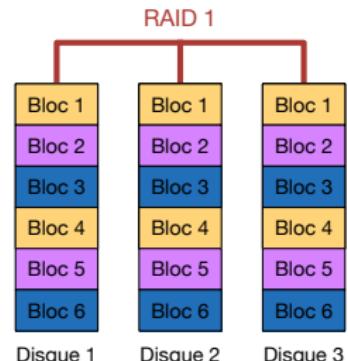
RAID 1 « *mirroring* »

- ▶ Espace logique découpé en bandes dupliquées sur tous les disques physiques
- ▶ Taille : celle du plus petit disque
 - ▶ Espace excédentaire perdu
 - ▶ Préférer des disques de même taille !
- ▶ Distribution possible de la charge en lecture, mais écriture simultanée pour que les disques restent interchangeables
- ▶ Résiste à une panne de tous les disques sauf un
- ▶ Intéressant pour la performance et (surtout) la fiabilité à deux disques

Organisation logique



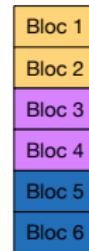
Organisation physique



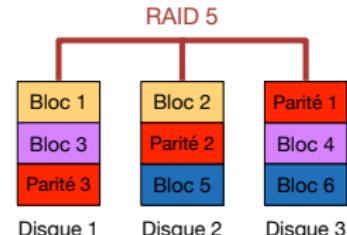
RAID 5 « *striping and parity* »

- ▶ Espace logique découpé en bandes entrelacées sur les disques physiques avec des bandes additionnelles de parité
 - ▶ Parité : XOR de chaque bit des bandes
 - ▶ Peut reconstruire une bande manquante
- ▶ Taille : celle du plus petit disque multipliée par le nombre de disques moins un
 - ▶ Préférer des disques de même taille !
 - ▶ Équivalent d'un disque pour la parité
- ▶ Améliore la performance en distribuant la charge sur plusieurs disques
- ▶ Résiste à une panne d'un des disque
- ▶ Répandu dans les baies de disques

Organisation logique



Organisation physique



That's all folks

Encore beaucoup de possibilités et de choses à découvrir,
mais ceci est une autre histoire...

Messages à ne pas oublier

- ▶ La mémoire est organisée en **hiérarchie** pour des raisons de coût, de taille et de performance
- ▶ L'organisation en hiérarchie fonctionne bien grâce au **principe de localité des données**
- ▶ Un programme doit être écrit en respectant ce principe pour une meilleure performance
- ▶ Plusieurs technologies mémoire ayant chacune leurs forces et leurs faiblesses coexistent et sont utilisées à différents étages de la hiérarchie
- ▶ La **mémoire cache** joue un rôle fondamental pour masquer la différence de performance entre les registres du processeur et la mémoire centrale