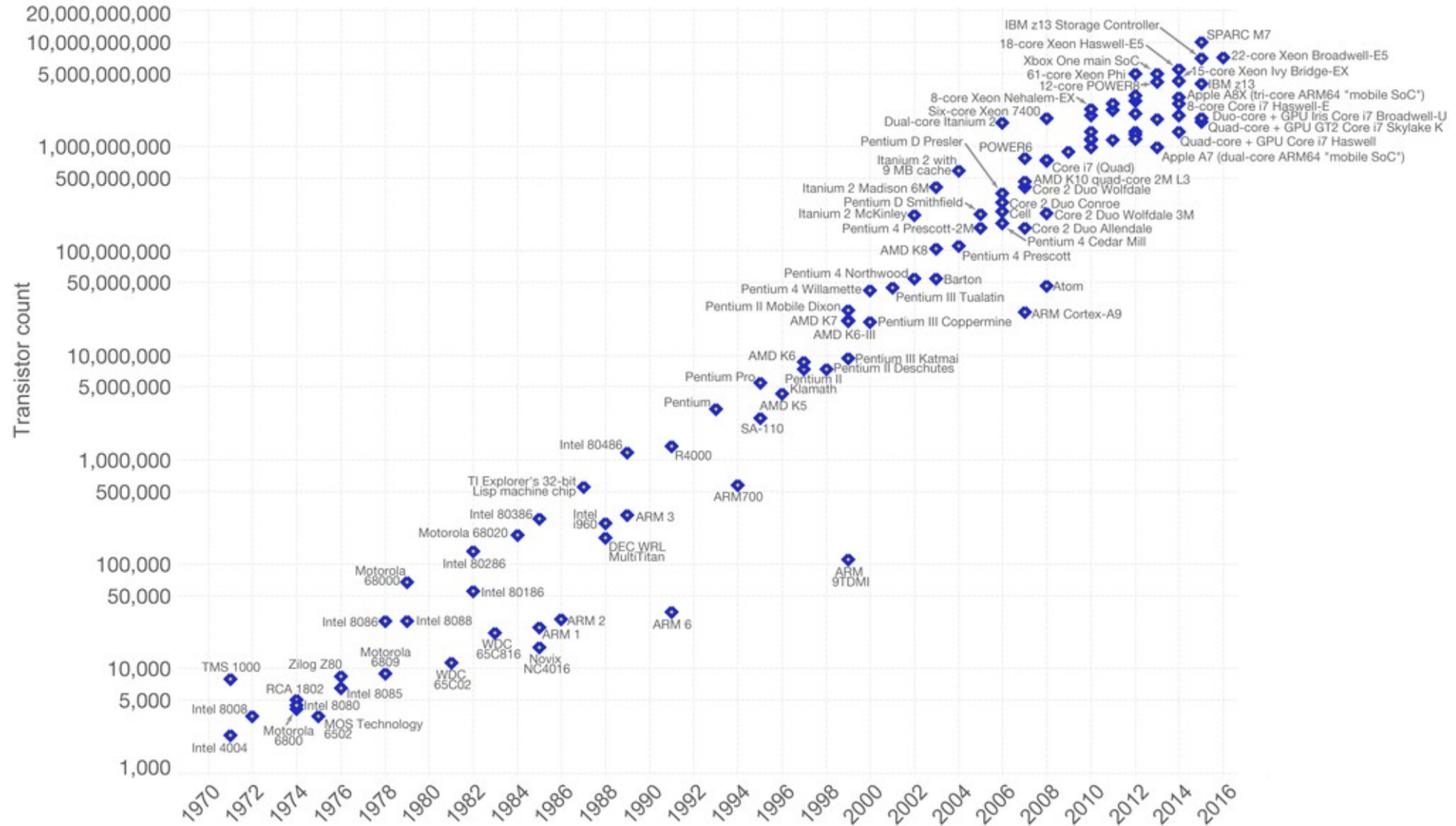
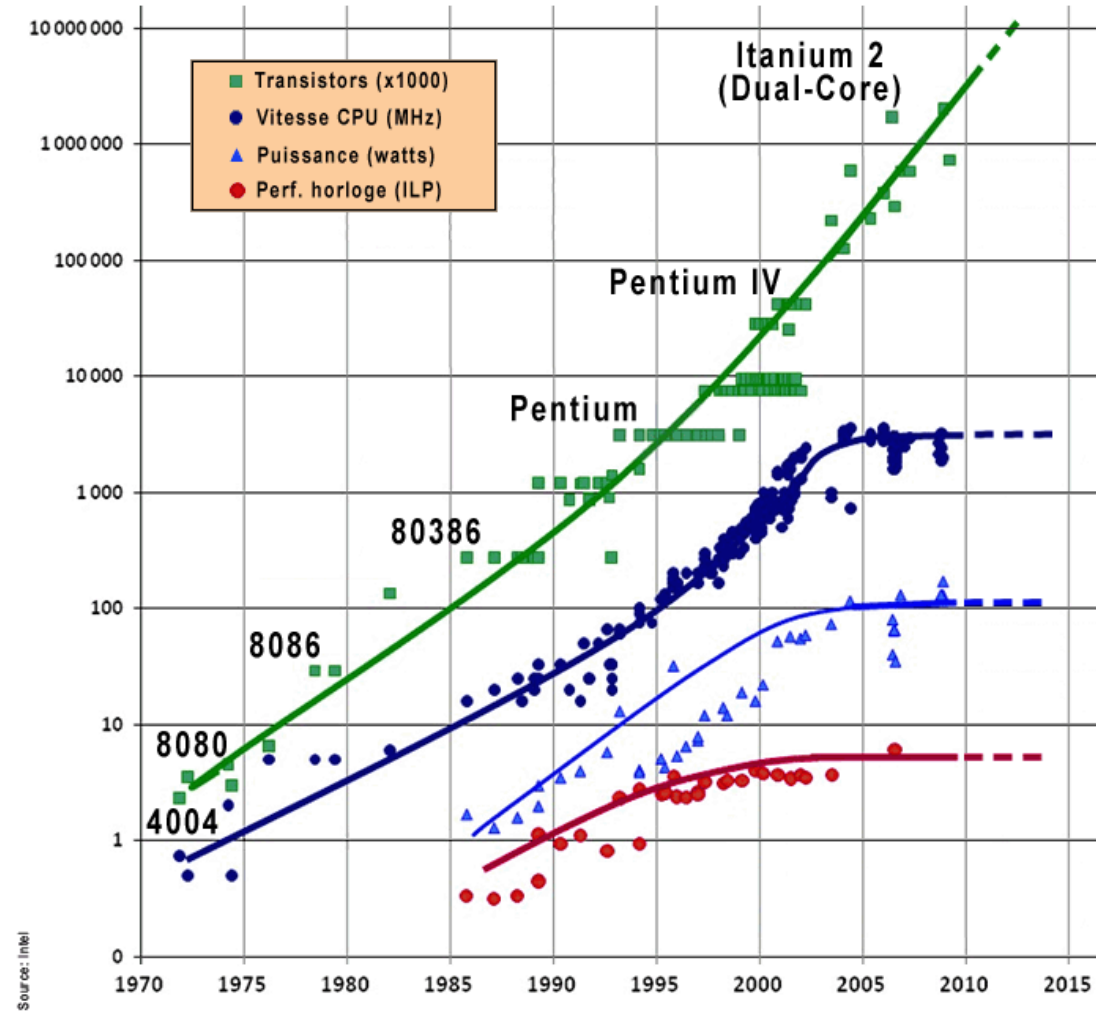


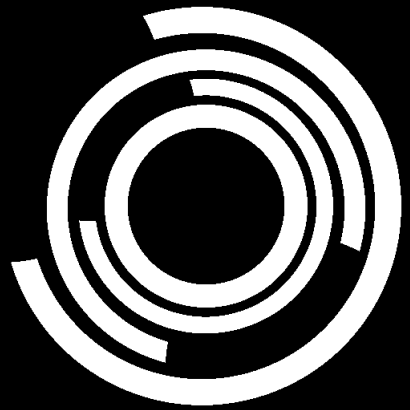
Architectures Parallèles

La fin de la course au GHZ



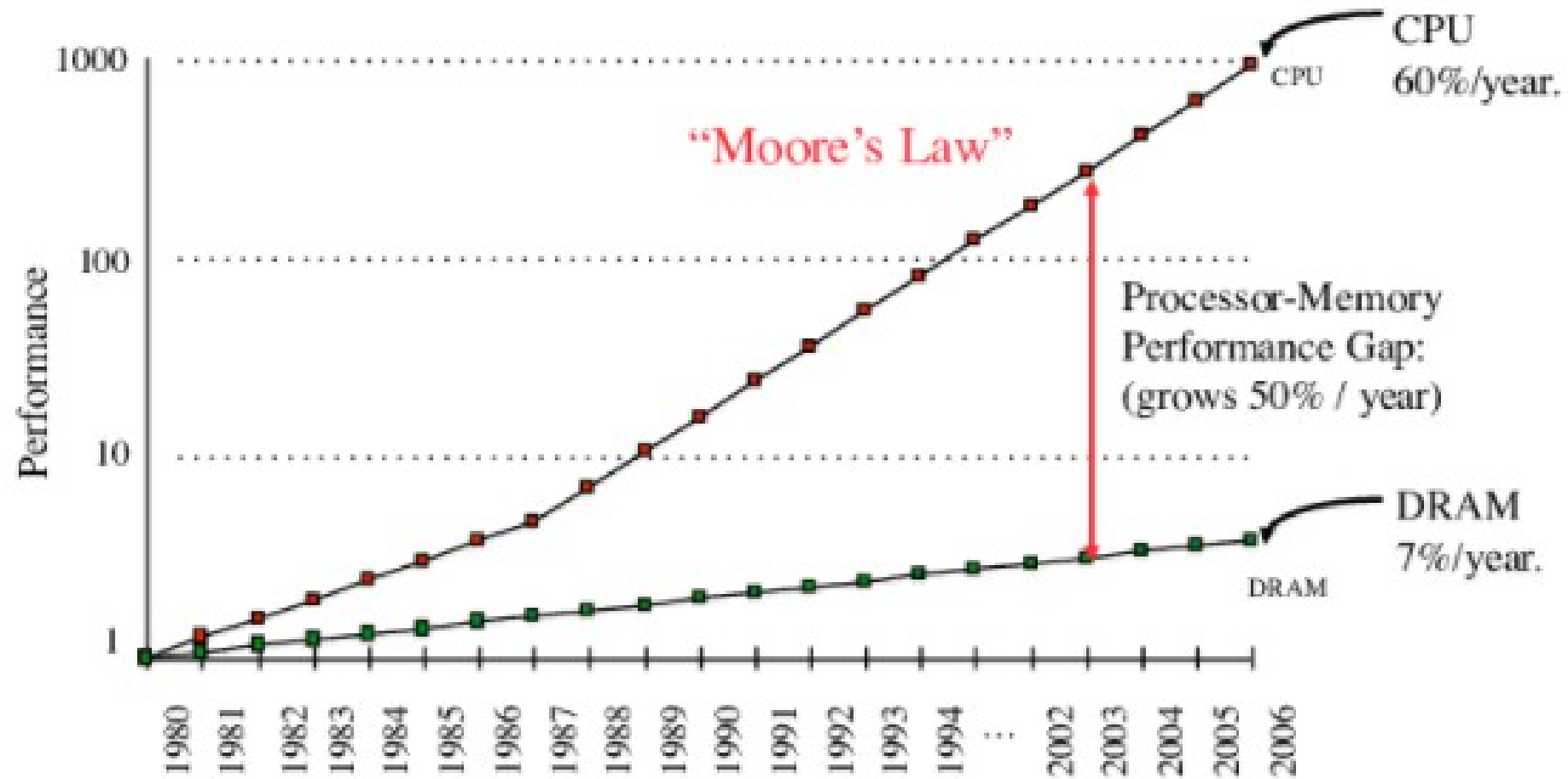
La fin de la course au GHZ





Impact de la mémoire

Mémoire vs CPU – Une guerre perdue d'avance



Mémoire vs Calcul

- Opérations mémoires
 - Lecture et écritures depuis la mémoire vers un registre
 - Repérable via l'accès à un objet de type tableau
 - On considère les variables locales en registres
 - Mesuré en octet/seconde
- Opérations de calcul
 - Toutes opérations arithmétiques natives ou composites
 - On néglige les différences dues aux pipelines ALU/FPU
 - Mesuré en FLOPS ou en cycle/éléments

Mémoire vs Calcul – Intensité arithmétique

• Quelques définitions

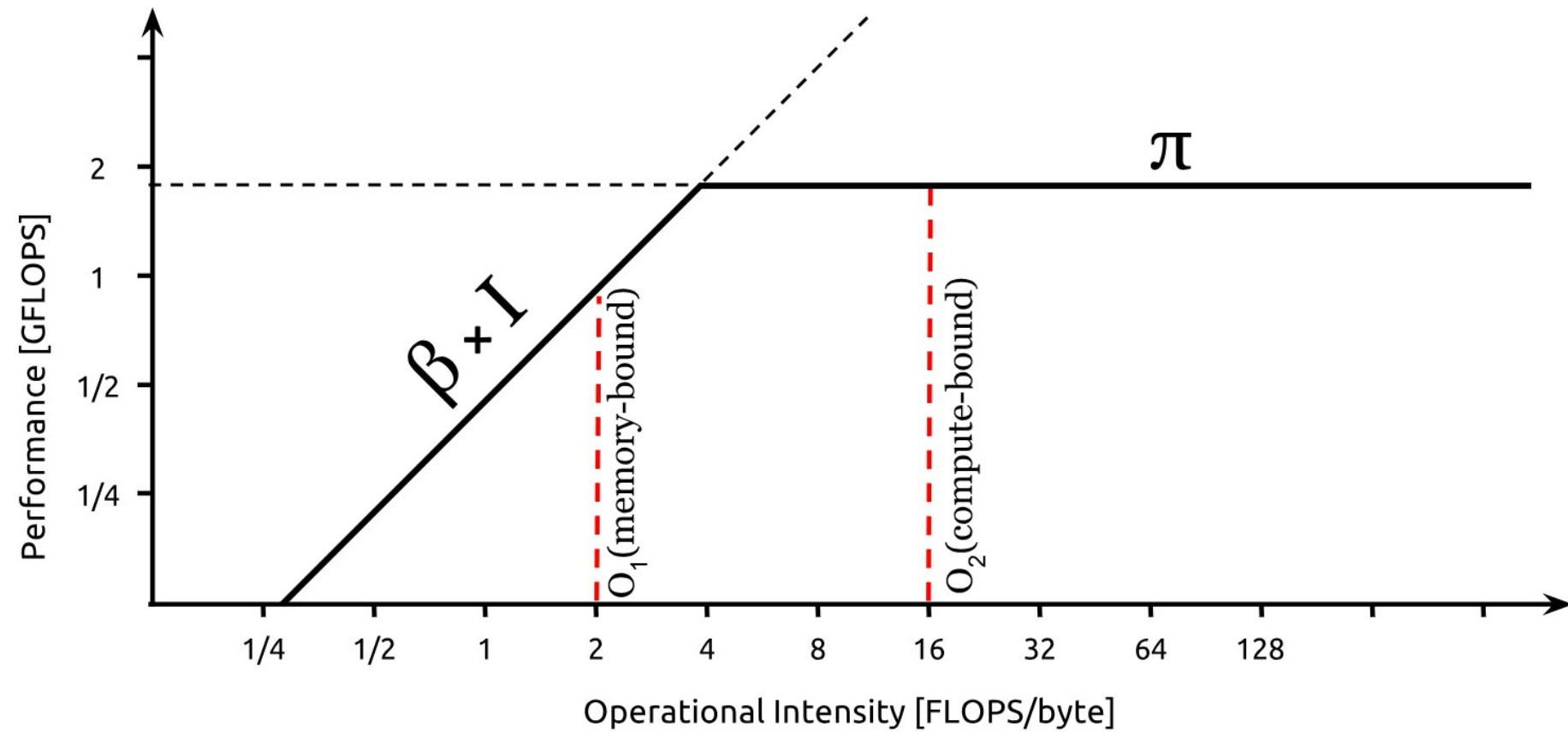
- Travail (W) : nombre d'opérations effectuées par un applicatif
- Trafic (Q) : quantité d'octets manipulées par un applicatif
- Performance pic (π) : qté maximale de travail par unité de temps
- Débit pic (β) : trafic maximal par unité de temps
- Débit pic (β) : trafic maximal par unité de temps

Intensité arithmétique: $I = \frac{W}{Q}$

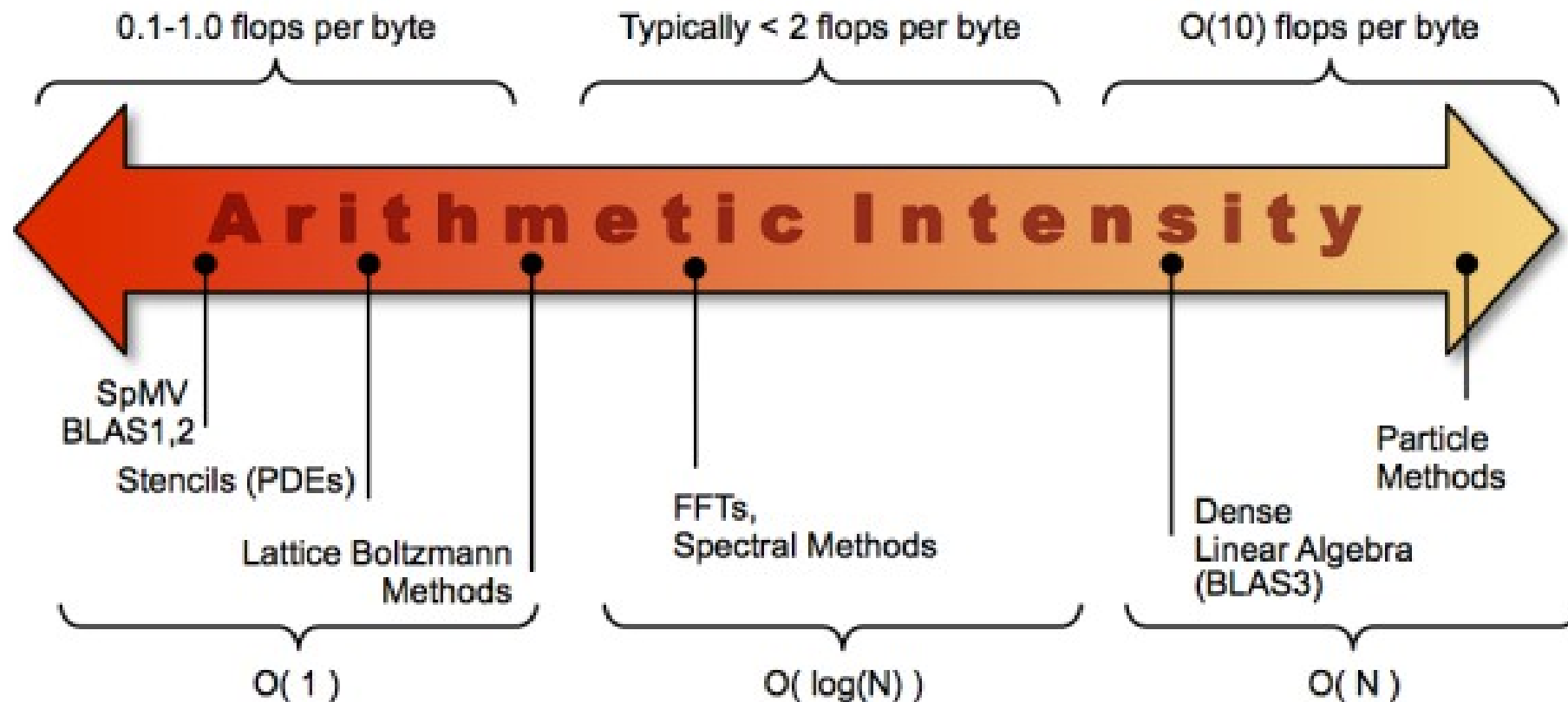
Intensité arithmétique:

Performance maximale: $P = \min \left\{ \frac{\pi}{\beta} \times I \right\}$

Mémoire vs Calcul – Le Roofline Model



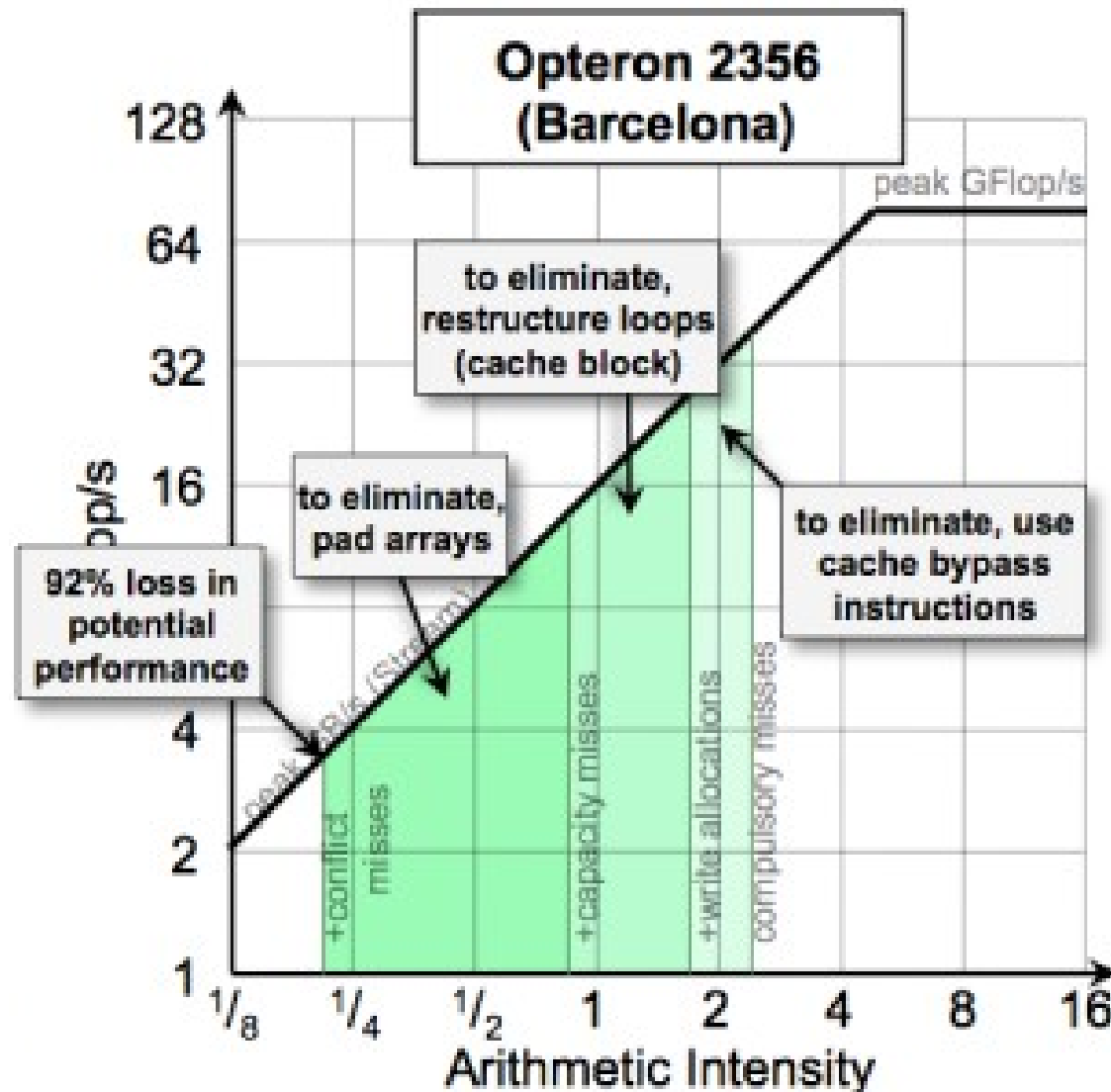
Intensité arithmétique classique



Mémoire vs Calcul – Conclusion

- Avant toutes optimisations !
 - Suis-je dans la zone Memory ou Compute-bound ?
 - **Memory bound** : se rapprocher de $\beta \times I$
 - **Compute bound** : se rapprocher de π
- Comment converger vers $\beta \times I$?
 - Amélioration de la quantité de mémoire nécessaire
 - Amélioration de l'utilisation des caches
- Comment converger vers π ?
 - Modification algorithmique
 - Parallélisme d'instructions

Impact du cache sur le Roofline Model

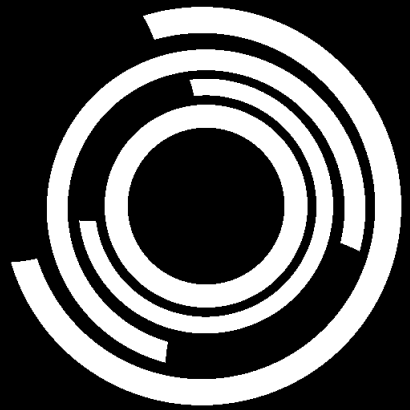


Conséquences sur les structures de données

- Conteneurs à privilégier
 - Contiguës
 - A accès unitaire
 - Ex: `std::vector`
- Et le reste ?
 - Table de hachage plutôt que dictionnaire
 - Adéquation algorithme/architecture

Structure et Tableaux

- Modèle AoS
 - Cas classique du tableau de structure
 - Efficace pour accéder à plusieurs champs par valeurs
- Modèle SoA
 - Structure contenant un tableau par champ
 - Efficace pour le traitement isolé d'un champ
- Modèle AoSoA
 - Modèle hybride permettant de maximiser l'utilisation de la hiérarchie mémoire

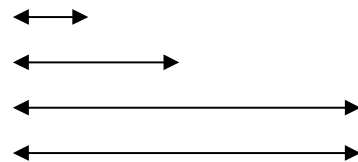
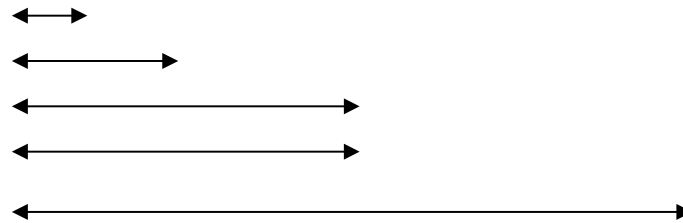


Unité de calcul vectoriel

Extensions SIMD dans les jeux d'instructions

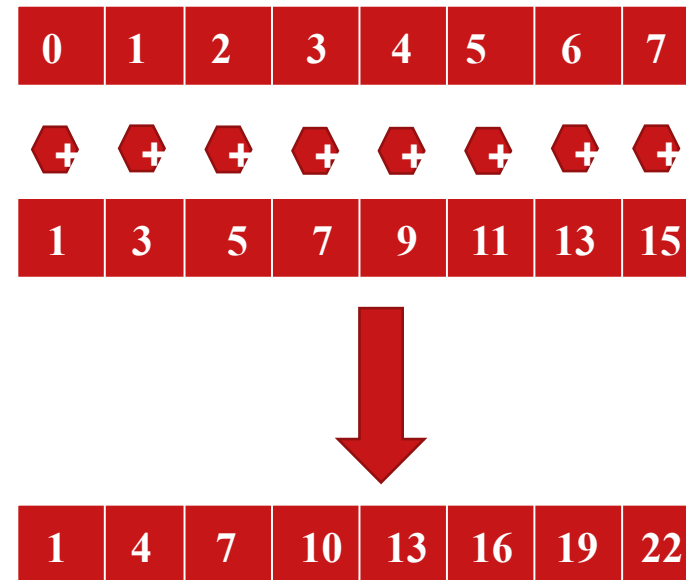
- Dans tous les jeux d'instructions
 - HP, SPARC VIS, MIPS, PowerPC AltiVec
 - ARM NEON, AARCH64, SVE
 - Intel MMX, SSE1-4, AVX, AVX2, AVX512
- Applications
 - Audio, Communication, Noyaux DSP, Graphique 2D et 3D, Images, Vidéo, Reconnaissance parole, etc
- Formes limitées d'instructions vectorielles
 - Vecteurs courts (2-64 éléments)
 - Accès mémoire avec pas unitaire
 - Transfert registres, registres mémoire et opérations sont SIMD (tous les éléments du vecteur simultanément)
 - Instructions “spéciales” multimédia

Formats de données SIMD



Instructions SIMD parallèles et scalaires

- Types d'instructions :
 - Instructions arithmétiques et logiques
 - Instructions mémoire
 - Instructions de formatage et manipulation
 - Instructions de conversion
- Fonctionnement classique
 - Opérations verticales entre éléments du registre
 - Opération registre/registre
 - Opération registre/scalaire

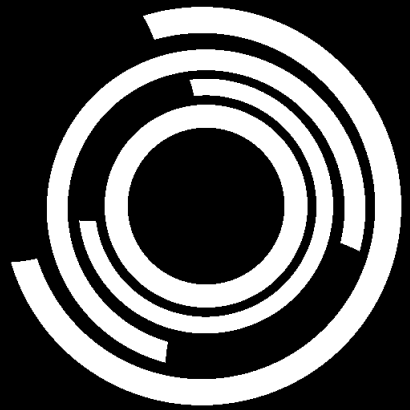


Exemple algorithmique - Valeur absolue

- Si $x(i) < 0$ alors $|x(i)| = -x(i)$
- Calcul des valeurs absolues du contenu registre SIMD
 - `__m128i v1;`
 - Créer une constante SIMD zero
 - Calculer `zero - v1`
 - Calculer `max(v1, zero - v1)`
- Notes
 - Pas de conditionnelles -> max
 - Chargement et génération des constantes explicites

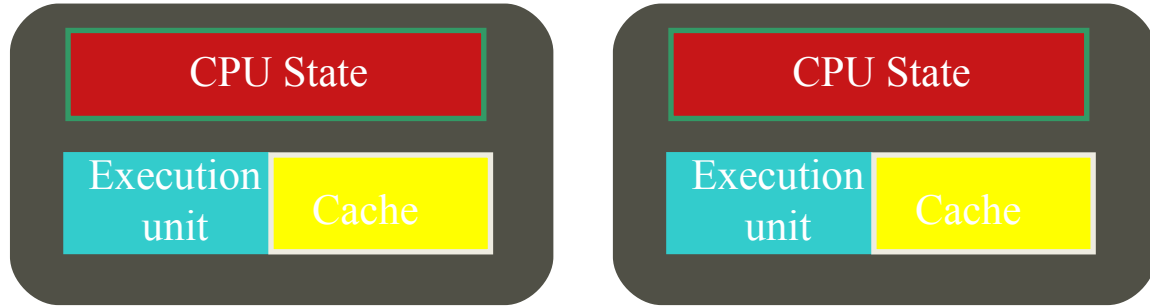
Outils de programmations SIMD

- Programmation bas-niveaux
 - Intrinsèques C INTEL, ARM, IBM
 - Assembleur inline (à déconseiller)
- Extensions de langages
 - OpenMP
 - UPC
 - Intel ISPC
- Bibliothèques
 - NSIMD

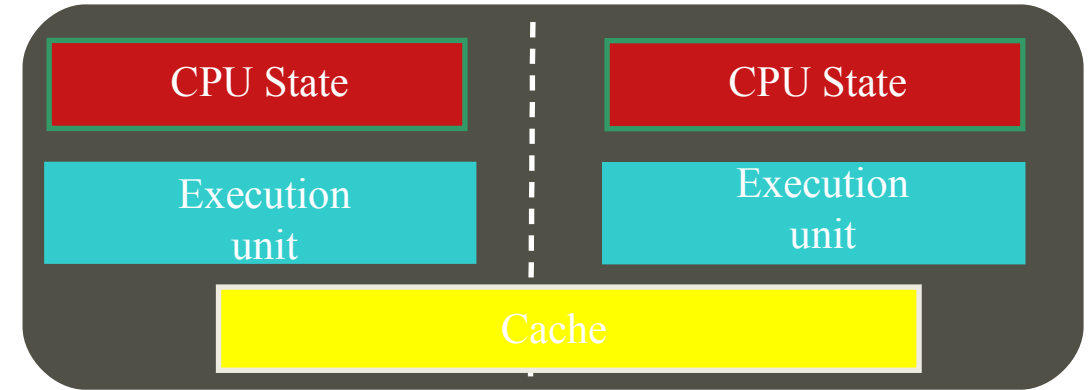


Multi et Many-core

Multiprocesseurs et multi-cœurs



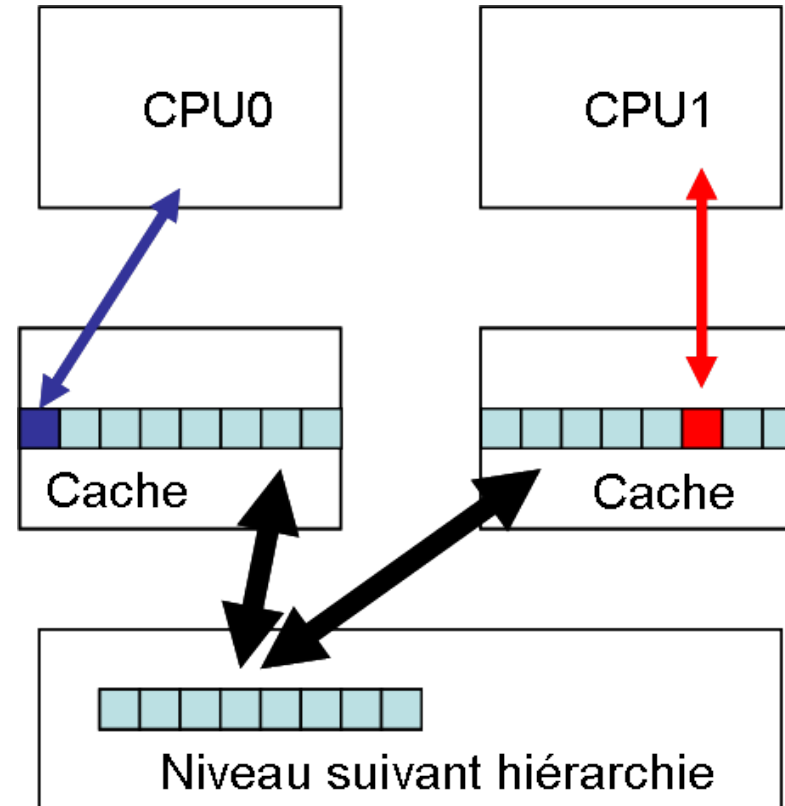
- Contient plusieurs processeurs
- Programmation parallèle de type **Simultaneous MultiProcessing** (SMP)
- Partage l'accès à la RAM
- Accès mémoire uniforme (UMA) ou non (NUMA)



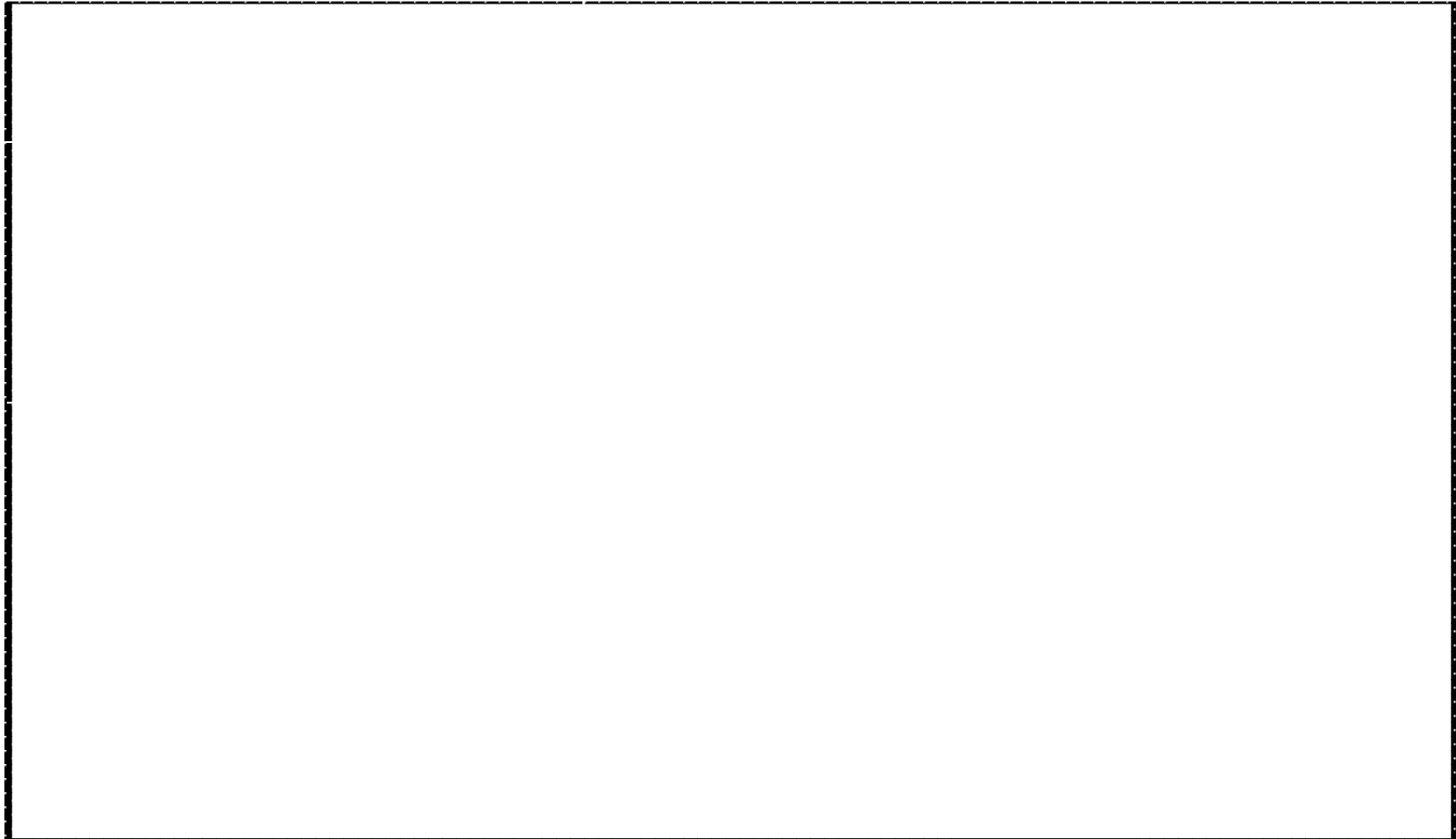
-
-
-
-

Le problème du faux partage

- Un processeur écrit dans une partie d'une ligne de cache.
- Un autre processeur écrit dans une autre partie d'une ligne de cache
- Même si chaque processeur ne modifie que sa « partie » de la ligne de cache, toute écriture dans un cache provoque une invalidation de « toute » la ligne de cache dans les autres caches.



Modèles de programmation



Outils de programmations SMP/SMT

- Programmation bas-niveaux

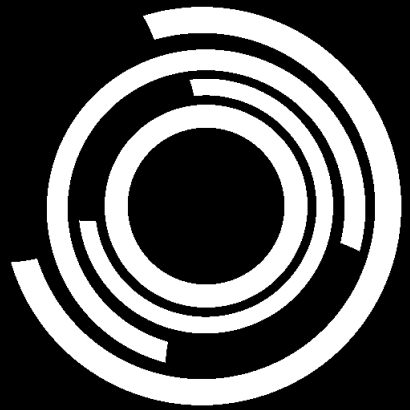
- Pthread
- WinThread

- Extensions de langages

- OpenMP
- High Performance FORTRAN
- UPC

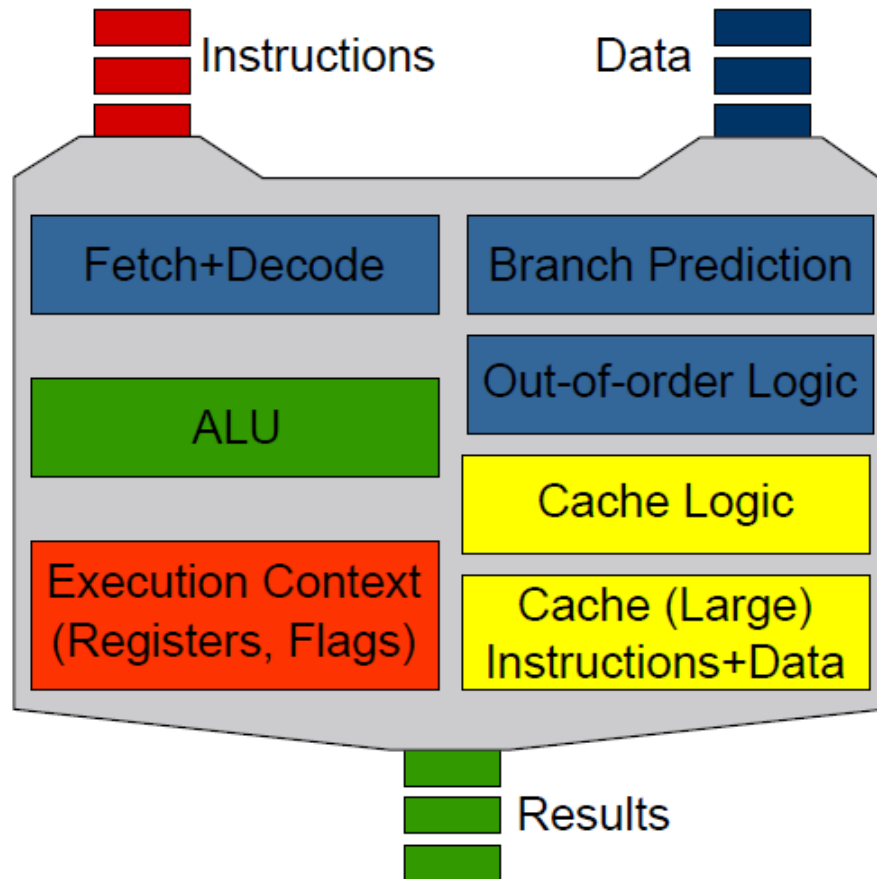
- Bibliothèques

- TBB
- Boost.Thread
- Boost.ASIO



Systeme Many- core

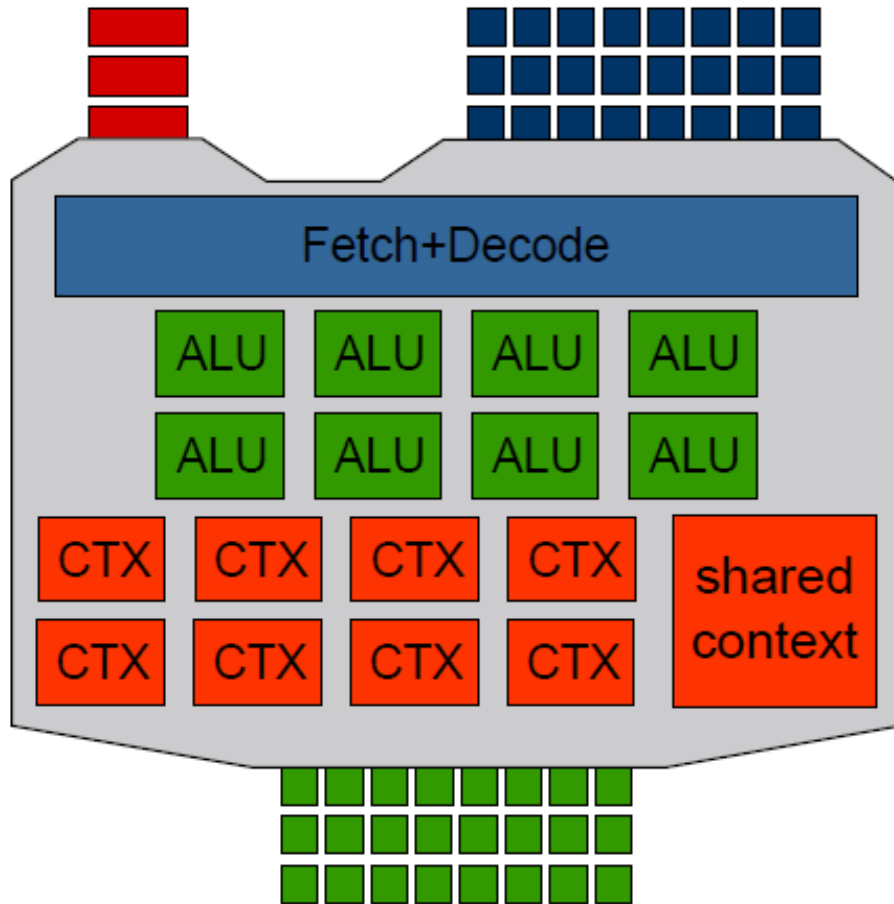
Du CPU à la GPU – Evolution architecturale



- Problèmes

- La puissance a cessé d'être proportionnelle à la DTS
- Complexité des CPUs
- **Moins de 5% de la surface du CPU participe aux calculs**

Du CPU à la GPU – Evolution



- Solutions

- Eliminer les sources de complexités :
 - Pas de pipeline
 - Pas de *out-of-order*
- Simplifier l'accès à la mémoire
 - Pas de cache
 - Synchronisation des fréquences CPU et RAM

Du CPU à la GPU – Evolution architecturale



- Conséquences
 - Cœur de calcul plus petit
 - 90% du cœur est utile au calcul
 - Multiplication des cœurs à la surface de la puce
- Impacts sur la performance
 - Parallélisme massif
 - Densité de calcul/cm² accrue

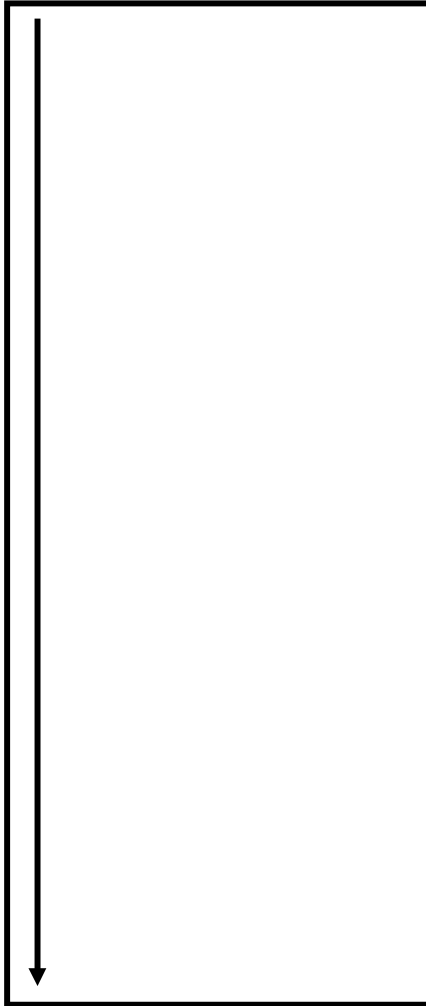
Du CPU à la GPU – Evolution

Architecture

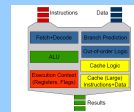


- NVIDIA Pascal
 - 3584 cœurs @ 1.5GHZ
 - Débit mémoire : 484 Go/s
 - Consommation : 250 W

Un système hétérogène



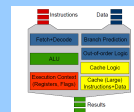
Hôte



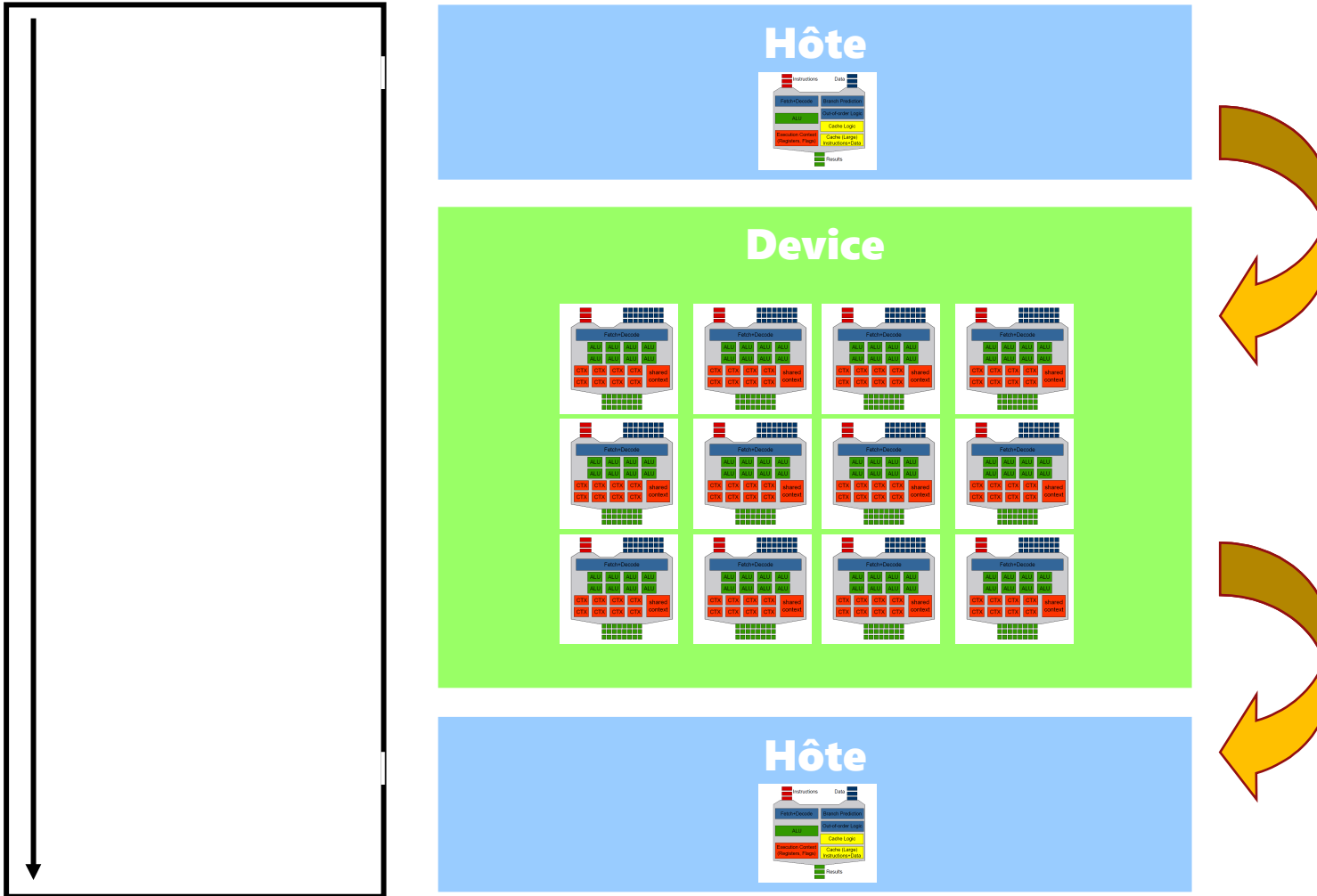
Device



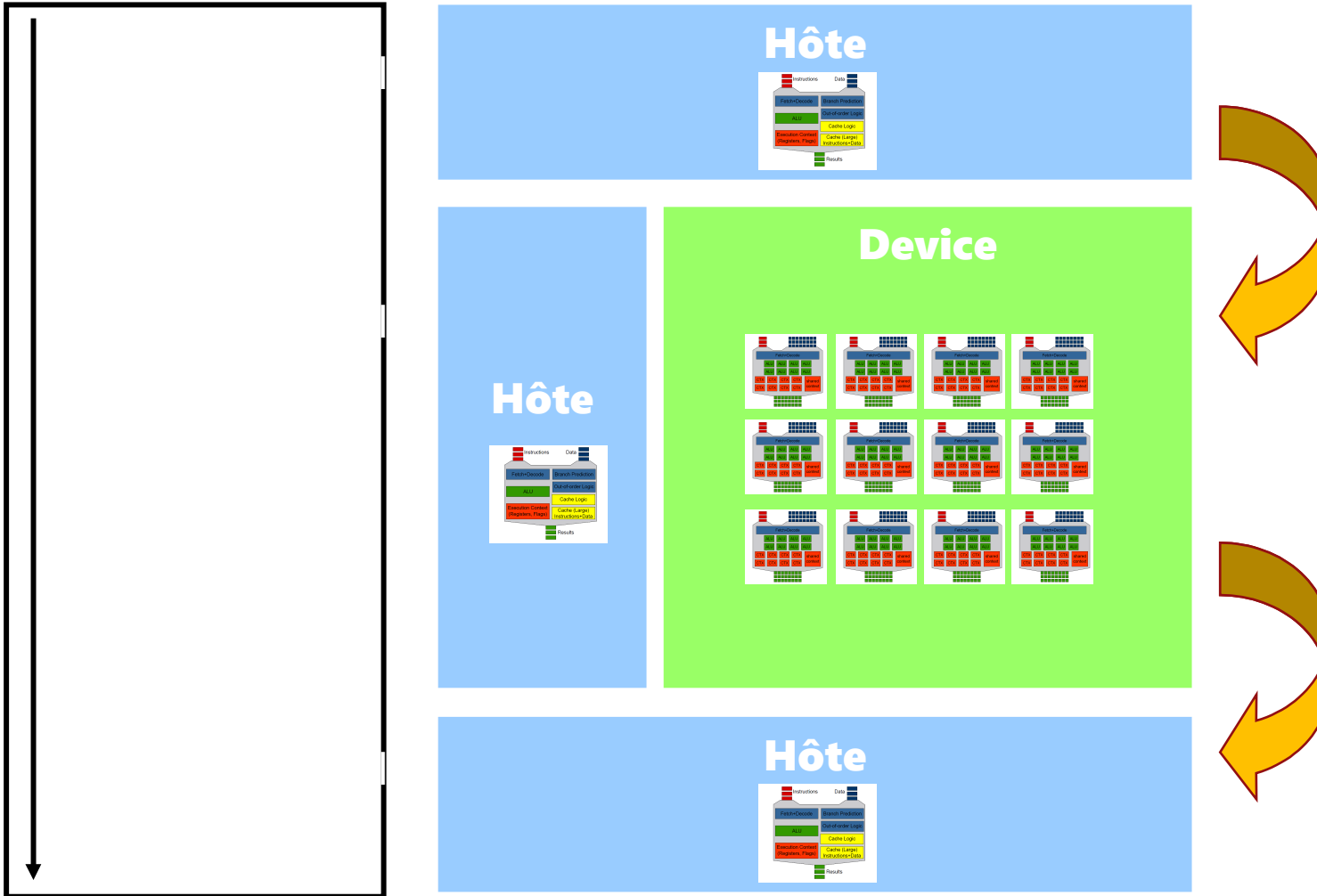
Hôte



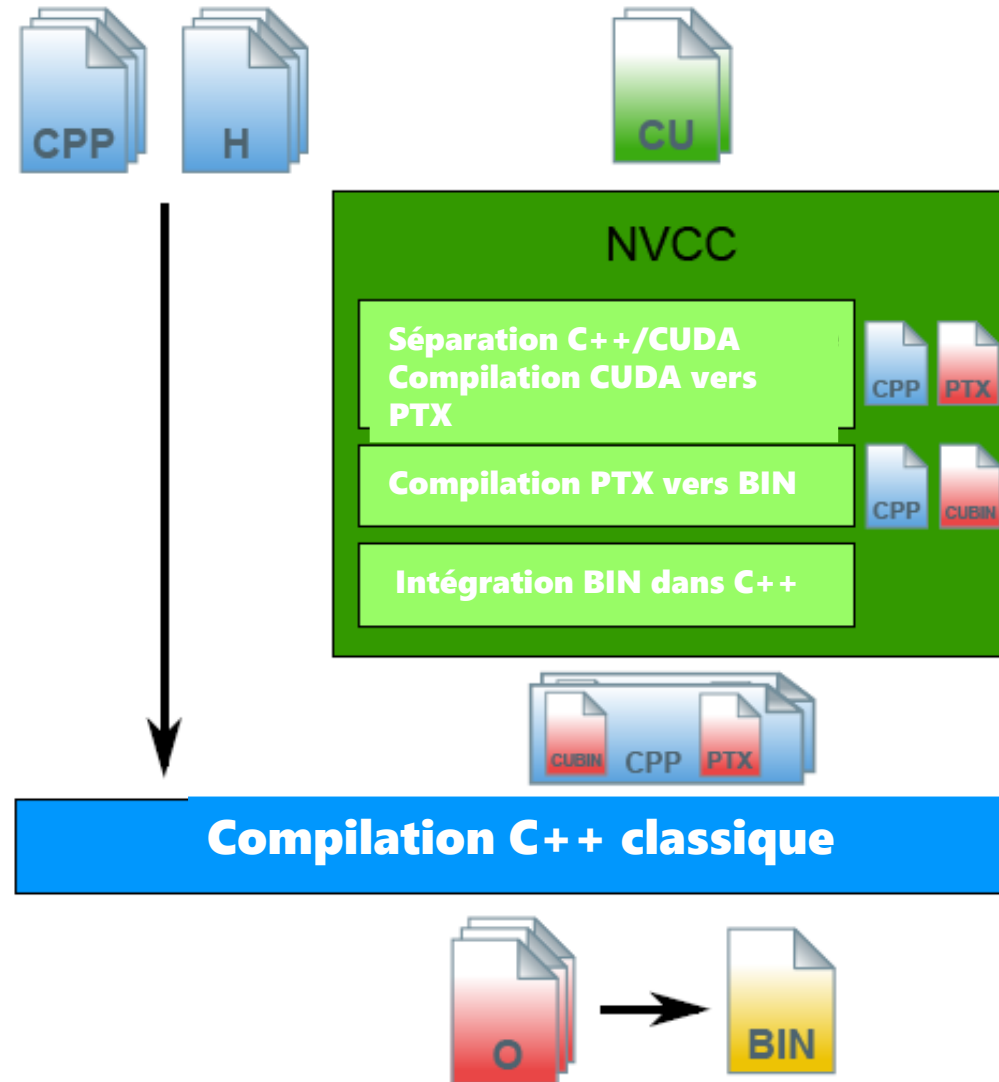
Un système hétérogène



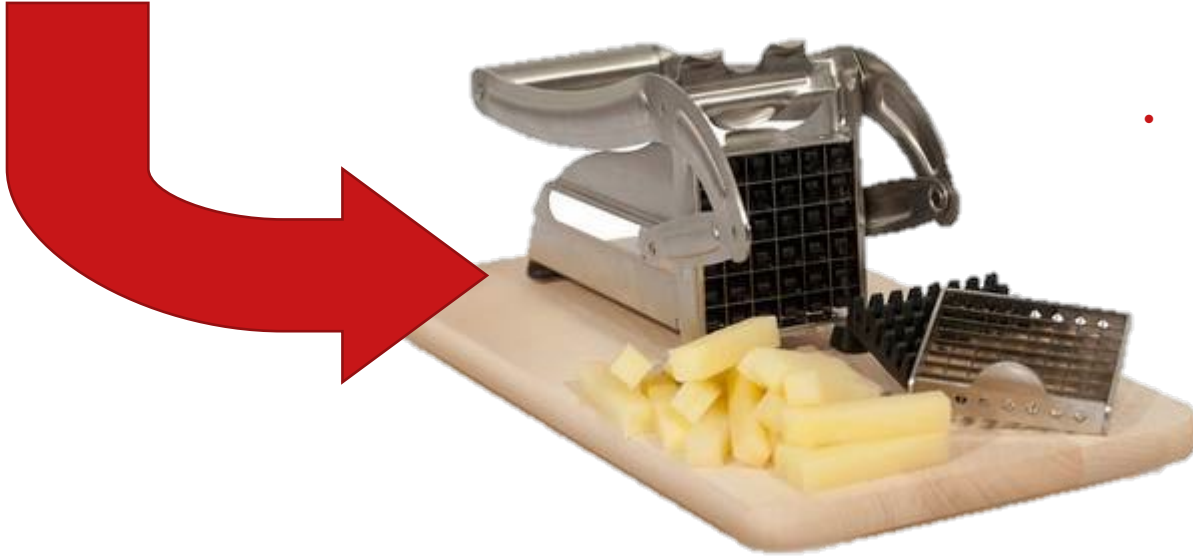
Un système hétérogène



Une compilation hétérogène



Modèle d'exécution SIMT



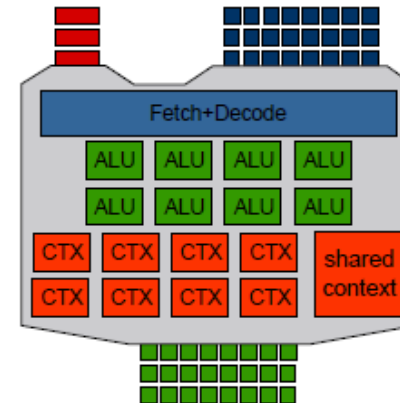
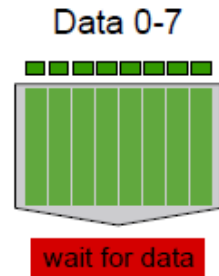
- Notion de kernel
 - Code séquentiel
 - Répliqué sur tous les cores
 - Exécutions simultanées
- Interaction avec les données
 - 1 instance de kernel = 1 donnée
 - Les boucles deviennent « spatiales »
 - Métaphores du presse frites

Problématique de la latence mémoire

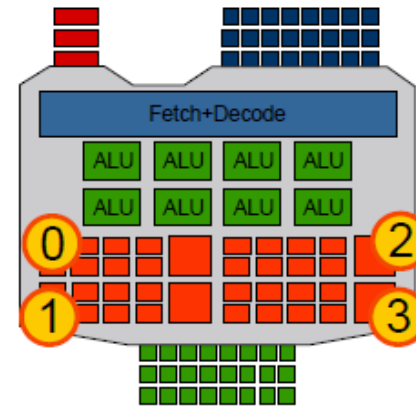
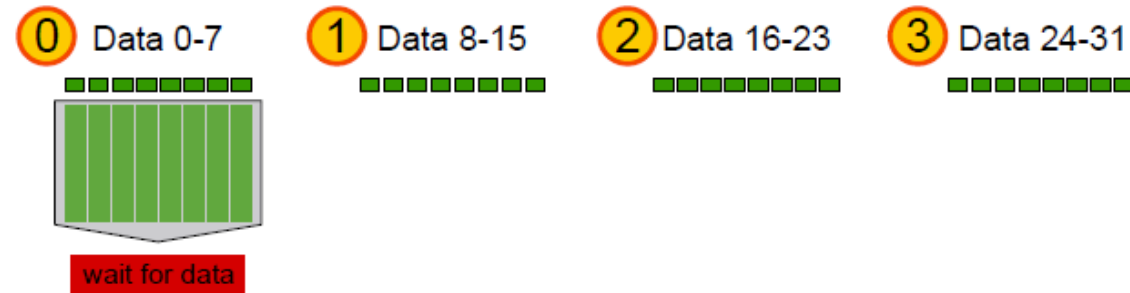
- La latence d'accès à la mémoire est largement supérieur au temps d'accès registre (x400-600)
- Mais le cœur de calcul de base de la GPU est dépourvu de tout les systèmes nécessaires à pallier à ces soucis

Idée : Utiliser le parallélisme de donnée pour occuper les cœurs pendant les phase d'attentes.

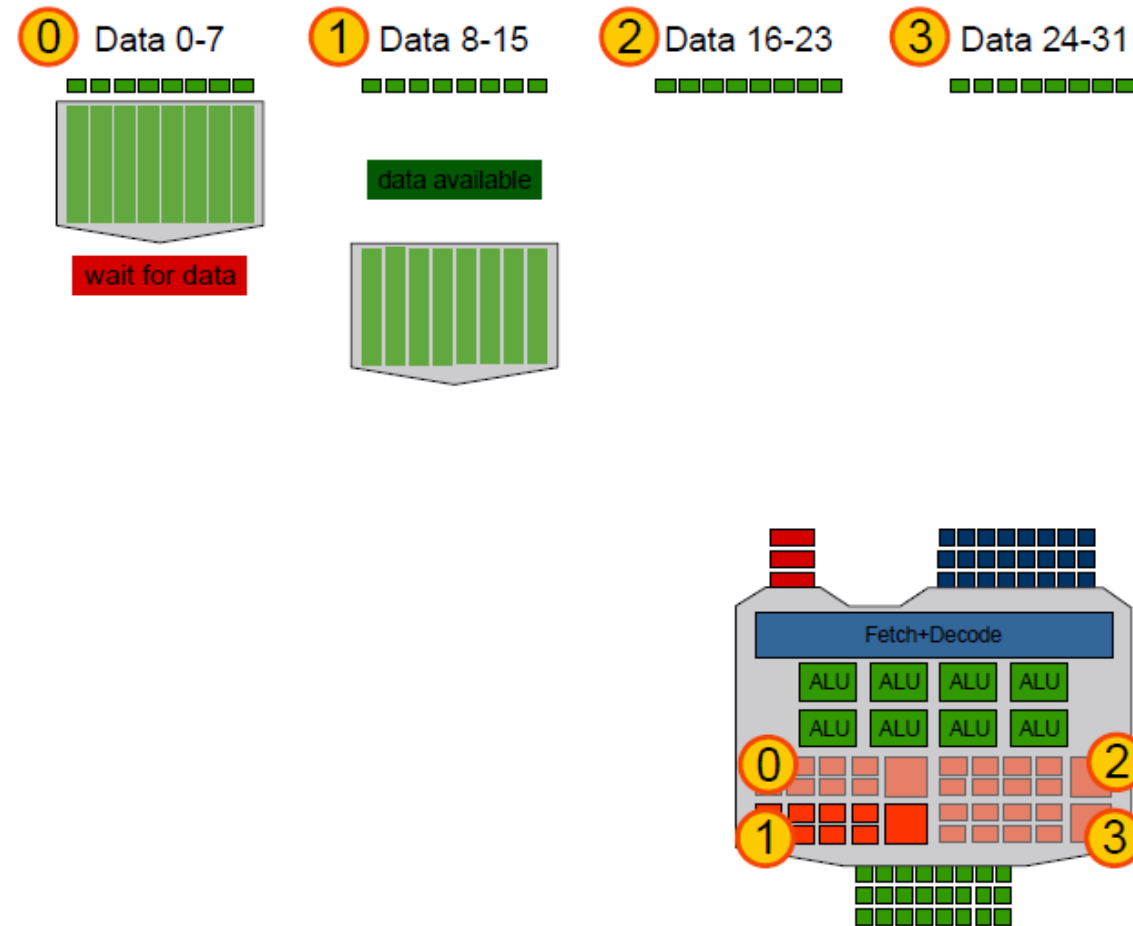
Latence mémoire - Résolution



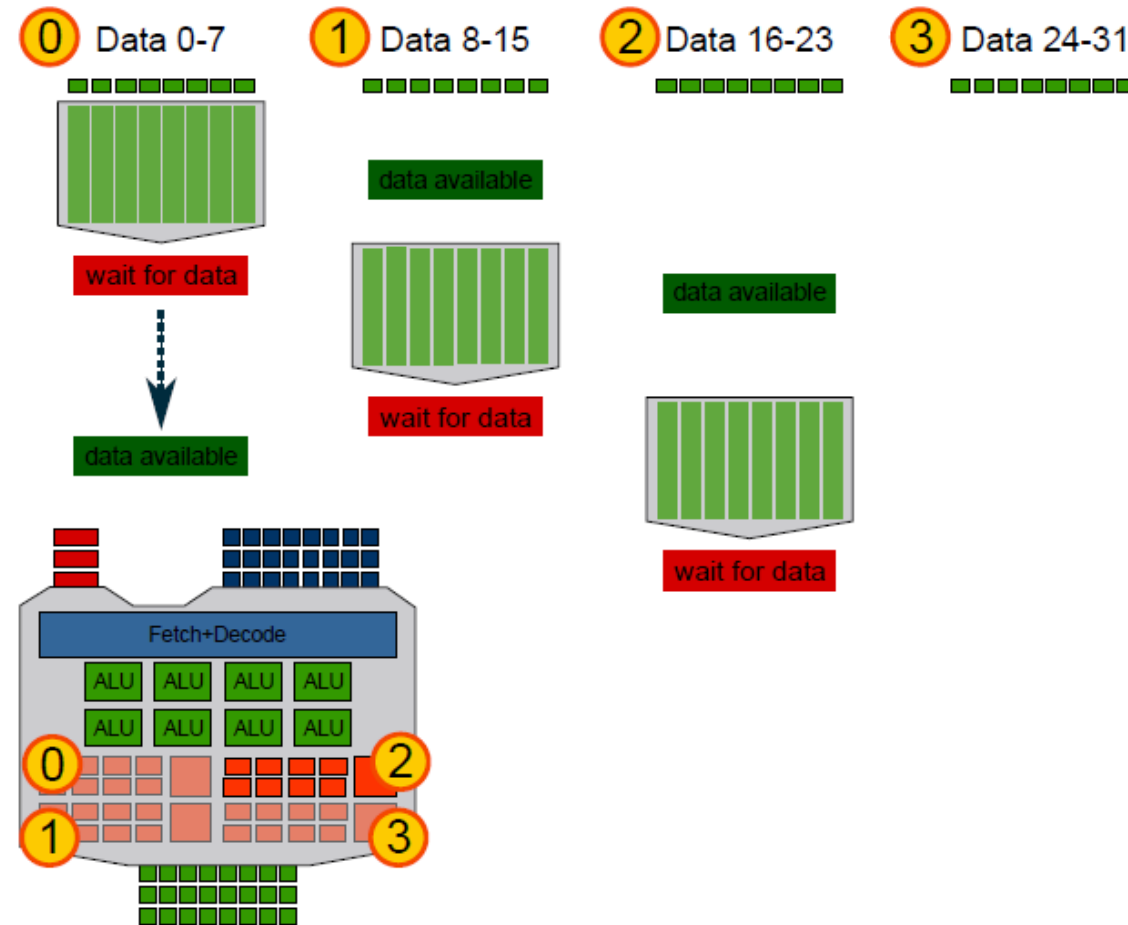
Latence mémoire - Résolution



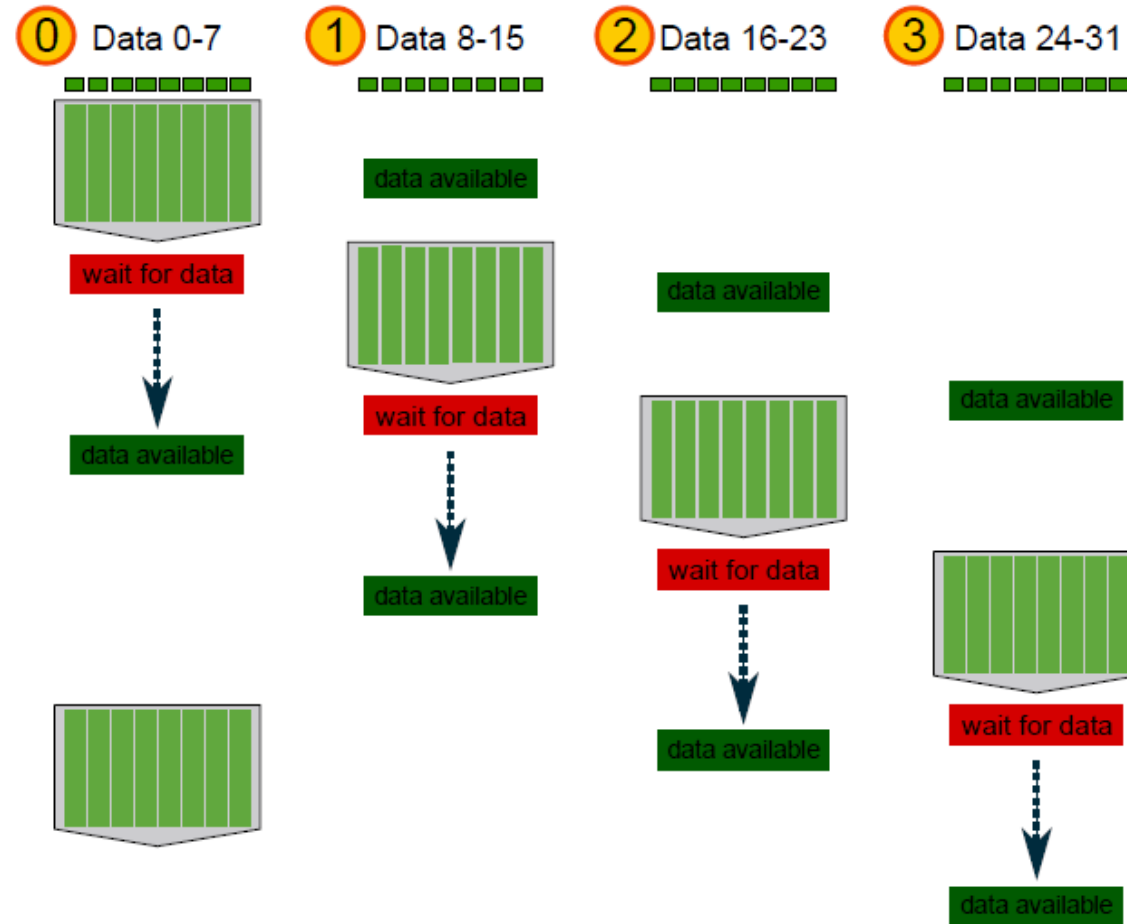
Latence mémoire - Résolution



Latence mémoire - Résolution



Latence mémoire - Résolution



Modèle d'exécution SIMT


```
// Données à traiter
double A[dim0 * dim1], B[dim0 * dim1], C[dim0 * dim1];

// Cas séquentiel CPU - Itération étendue dans le temps
for (int i1 = 0; i1 < dim1; i1++)
    for (int i0 = 0; i0 < dim0; i0++)
        A[i0 + dim1 * i1] = B[i0 + dim1 * i1] + C[i0 + dim1 * i1];
```

Modèle d'exécution SIMT

```
// Données à traiter
double A[dim0 * dim1], B[dim0 * dim1], C[dim0 * dim1];

// Cas séquentiel CPU - Itération étendue dans le temps
for (int i1 = 0; i1 < dim1; i1++)
    for (int i0 = 0; i0 < dim0; i0++)
        A[i0 + dim1 * i1] = B[i0 + dim1 * i1] + C[i0 + dim1 * i1];
```



Modèle d'exécution SIMT

```
// Données à traiter
double A[dim0 * dim1], B[dim0 * dim1], C[dim0 * dim1];

// Cas parallèle GPU - Itération étendue dans l'espace des cores
int i0 = threadIdx.x;
int i1 = blockIdx.x;
A[i0 + dim1 * i1] = B[i0 + dim1 * i1] + C[i0 + dim1 * i1];
```