Cours 8 : la mémoire

Charles Bouillaguet
charles.bouillaguet@univ-lille.fr

12 janvier 2021

L'ennemi n°1 de la programmation HPC :

L'ennemi n°1 de la programmation HPC :

```
double x = A[i];
```

Le « *Memory Wall* »

- La **puissance de calcul** augmente.
  - Augmentation rapide des FLOP/s.
- La vitesse de la mémoire **ne suit pas**.
  - Augmentation *moins rapide* des Go/s.

On distingue...

- Algorithmes compute-bound (ou CPU-bound).
  - Limités par FLOP/s.
- Algorithmes memory-bound.
  - limités par Go/s depuis la RAM.

Le « *Memory Wall* »

FLOPS ÷ [débit de la mémoire]

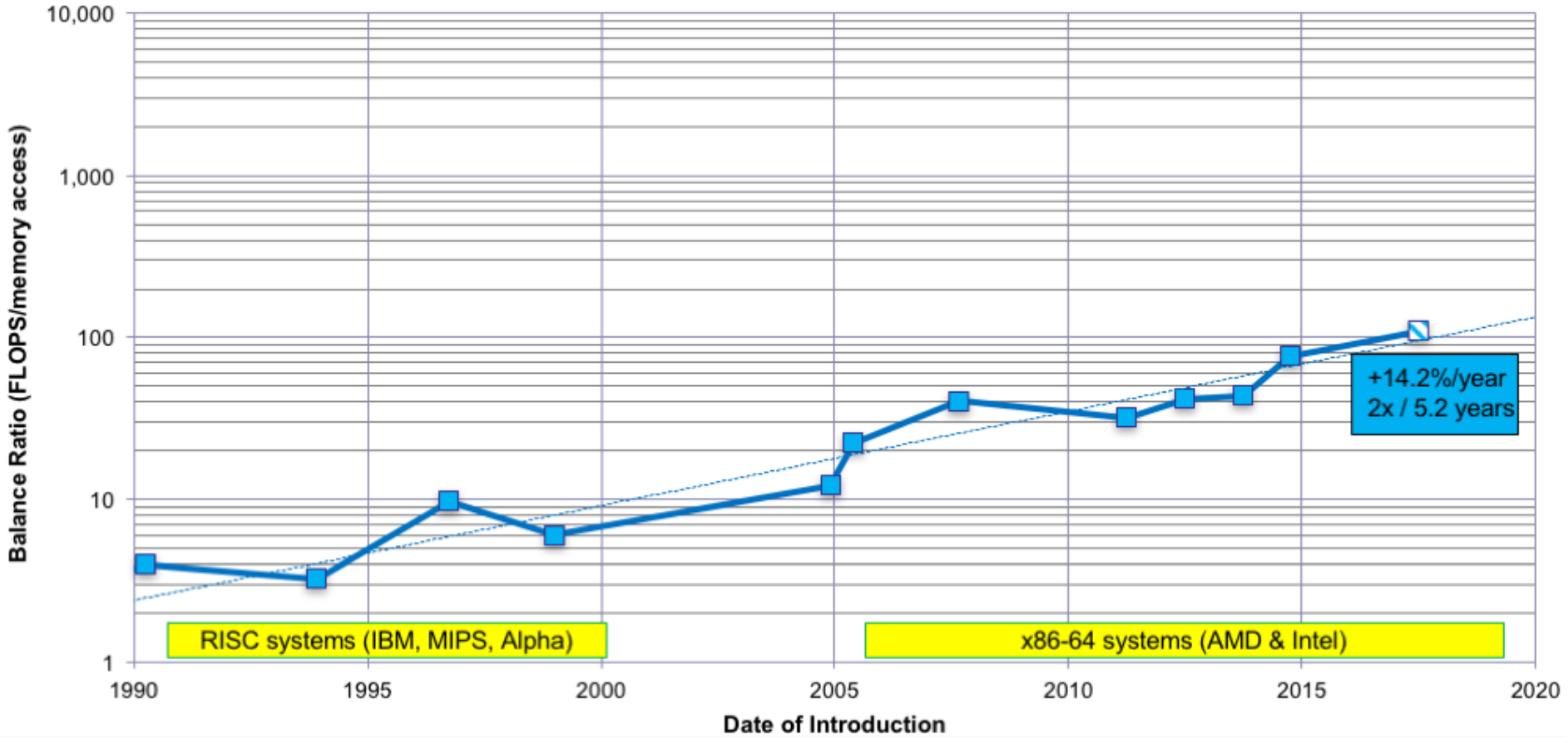


image : John McCalpin

Le « *Memory Wall* »

FLOPS ÷ [latence de la mémoire]

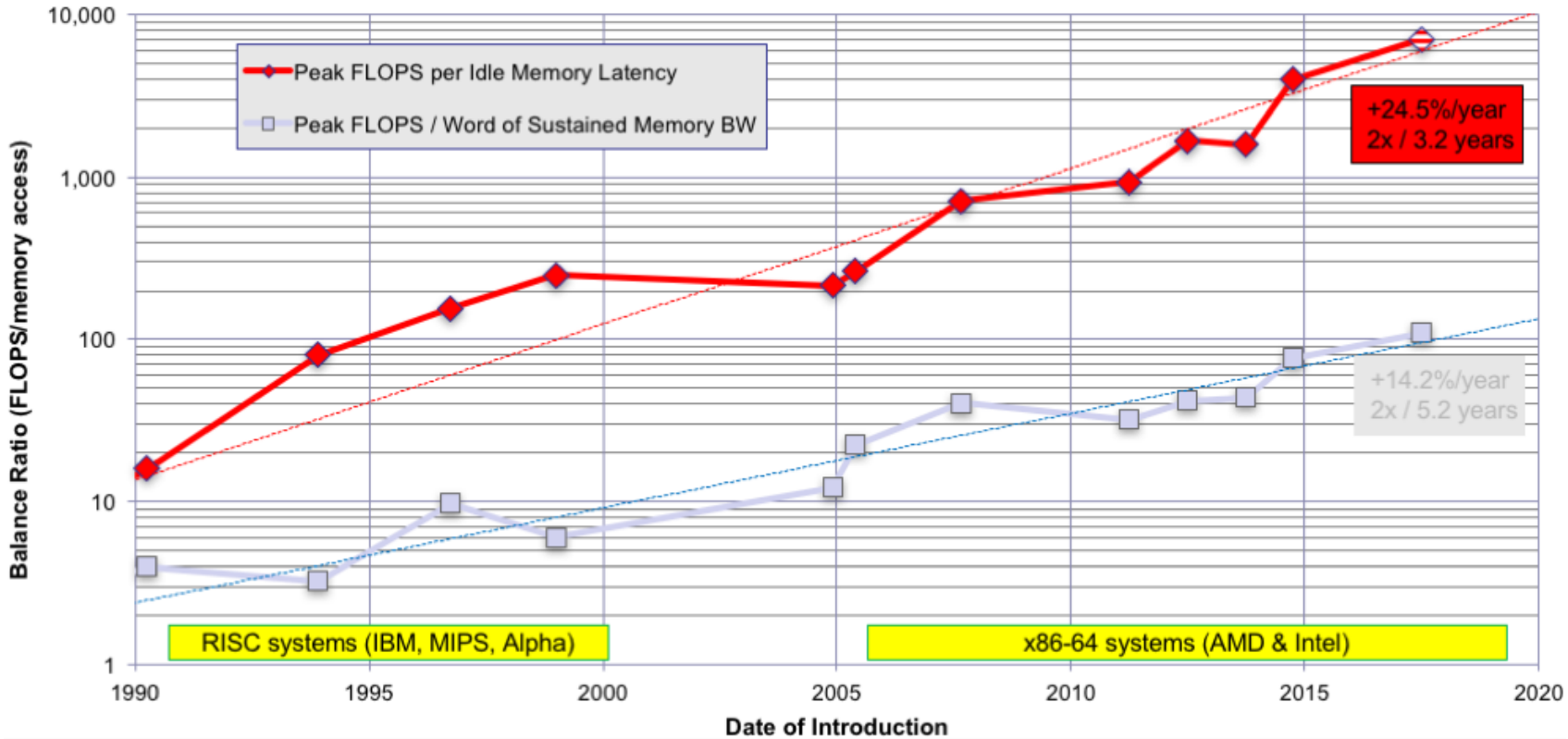


image : John McCalpin

Multicoeur : c'est l'horreur

Machine	Threads	Go/s	Speedup
Laptop	1	10.9	-
	2	10.9	1
Raspberry 3B+	1	1.8	-
	2	2.3	1.3
	4	2.0	1.1
BlueGene/Q	1	7.5	-
	2	15	2
	4	26.8	3.6
	8	27.9	3.7
	16	28.0	3.7
Cluster node	1	12.7	-
	2	24.6	1.9
	4	47.8	3.7
	8	67.6	5.3
	16	73.4	5.7

Accès non-contigus

$T =$ tableau de N entiers aléatoires dans $[0; N[$.

```
for (int i=0, x=0; i < 1000000000; i++) x = T[x];
```

En théorie

Complexité indépendante de N.

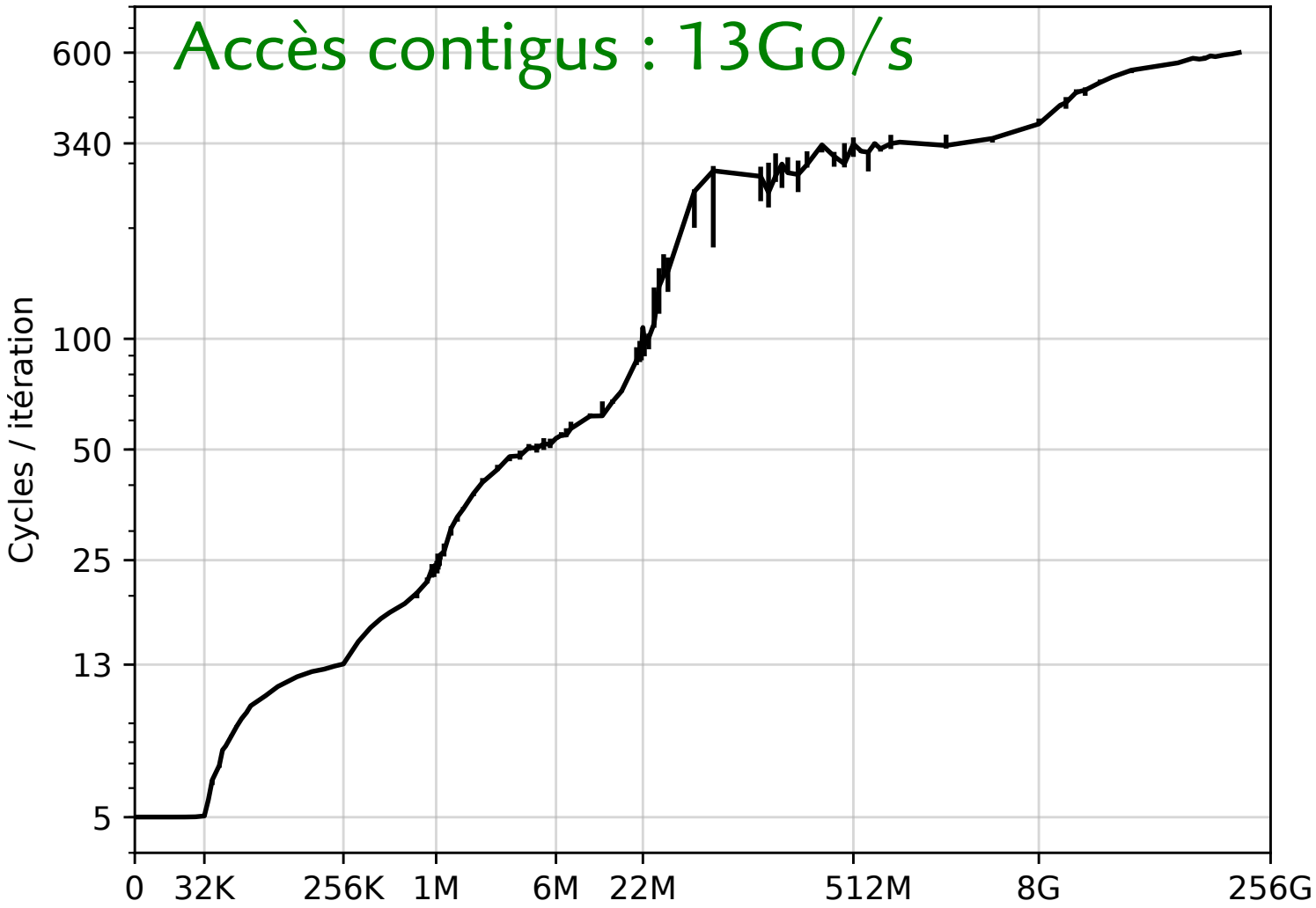
En pratique

On a affaire à la **latence** de la mémoire

Accès non-contigus

$T =$ tableau de N entiers aléatoires dans $[0; N[$.

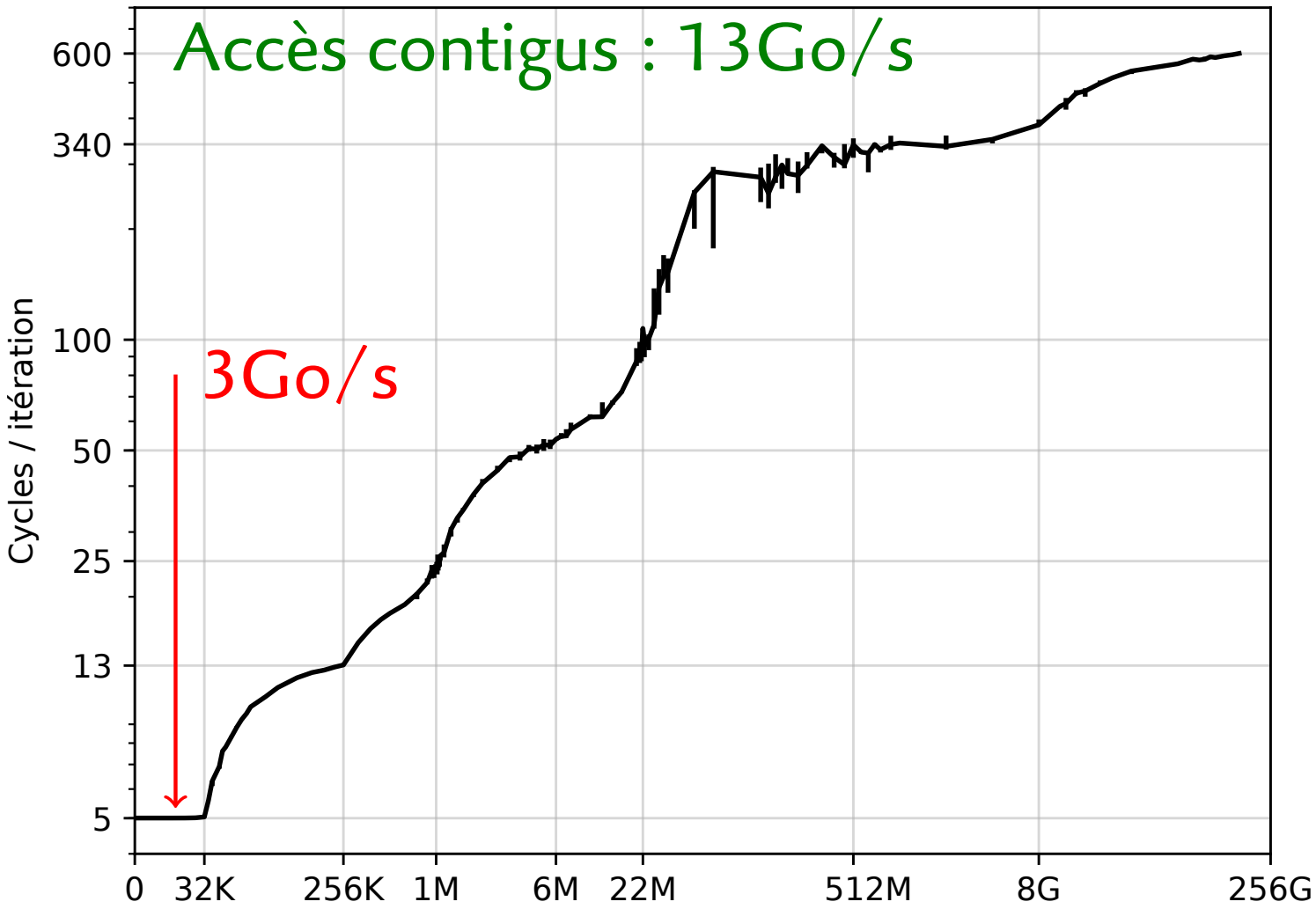
```
for (int i=0, x=0; i < 1000000000; i++) x = T[x];
```



Accès non-contigus

$T =$ tableau de N entiers aléatoires dans $[0; N[$.

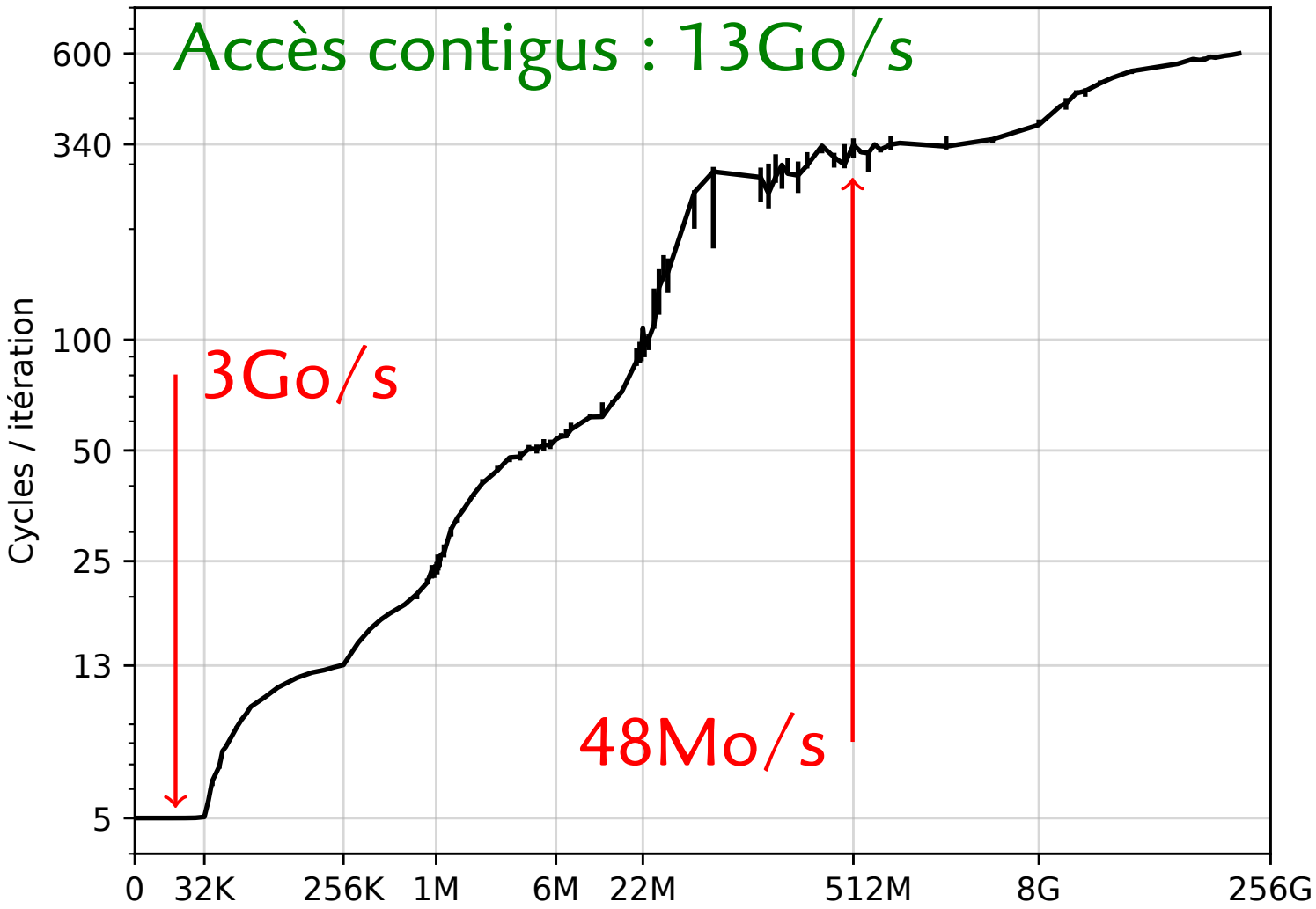
```
for (int i=0, x=0; i < 1000000000; i++) x = T[x];
```



Accès non-contigus

$T =$ tableau de N entiers aléatoires dans $[0; N[$.

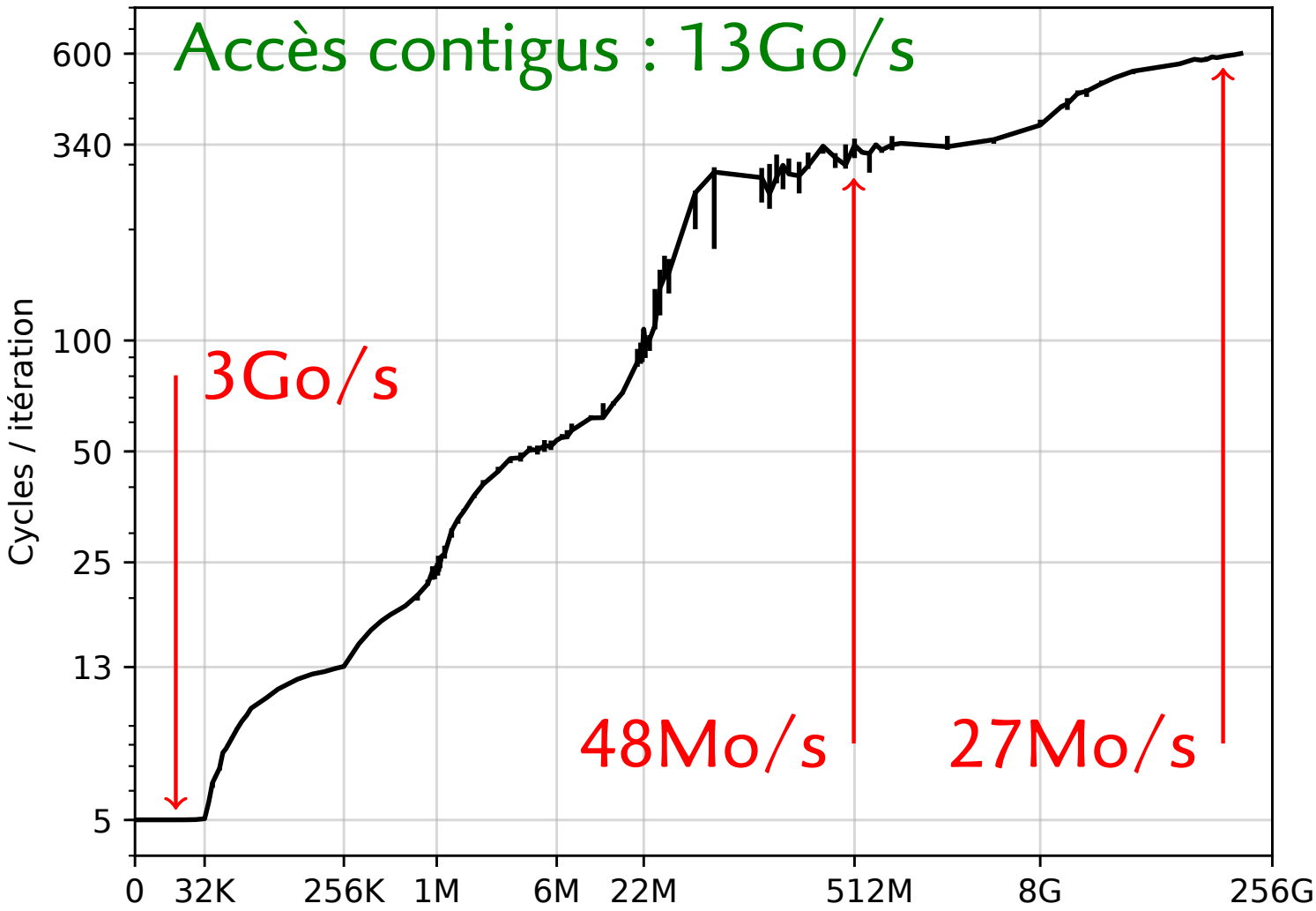
```
for (int i=0, x=0; i < 1000000000; i++) x = T[x];
```



Accès non-contigus

$T =$ tableau de N entiers aléatoires dans $[0; N[$.

```
for (int i=0, x=0; i < 1000000000; i++) x = T[x];
```



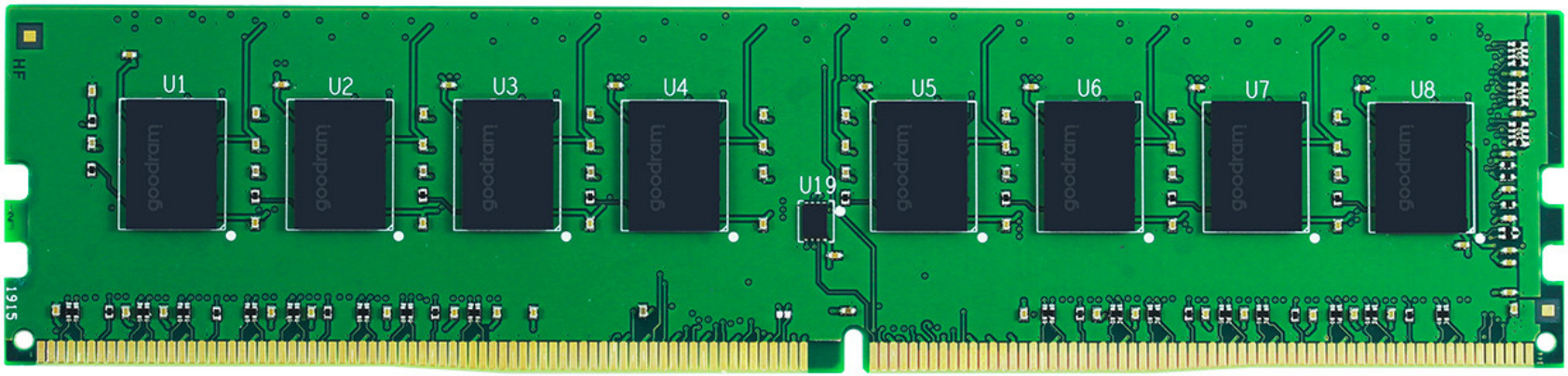
Plan

1. Description du hardware
2. La hiérarchie mémoire (caches)
3. Amélioration de la localité des données
4. (bonus) Problèmes liés à la pagination

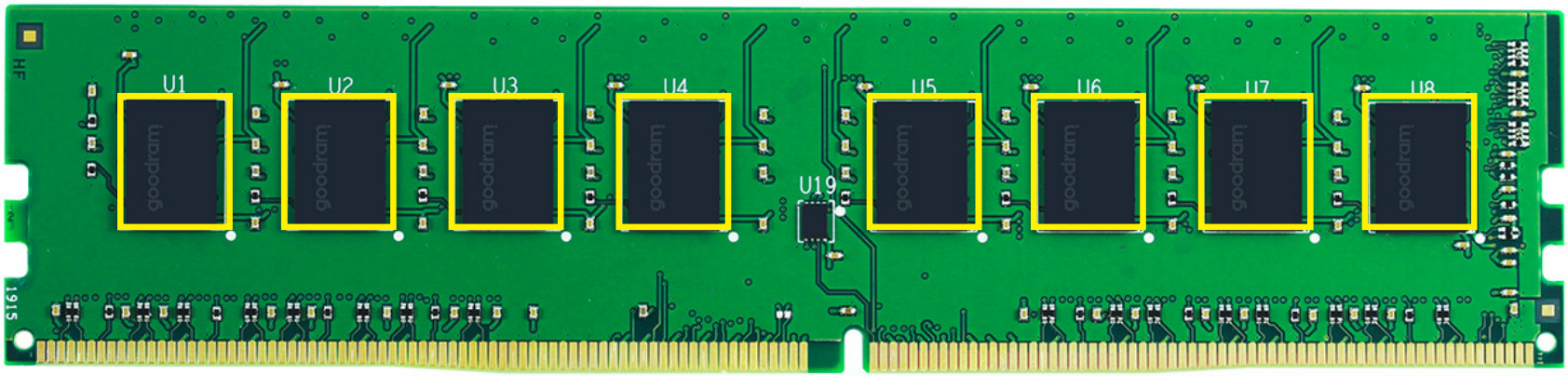
Plan

1. Description du hardware
2. La hiérarchie mémoire (caches)
3. Amélioration de la localité des données
4. (bonus) Problèmes liés à la pagination

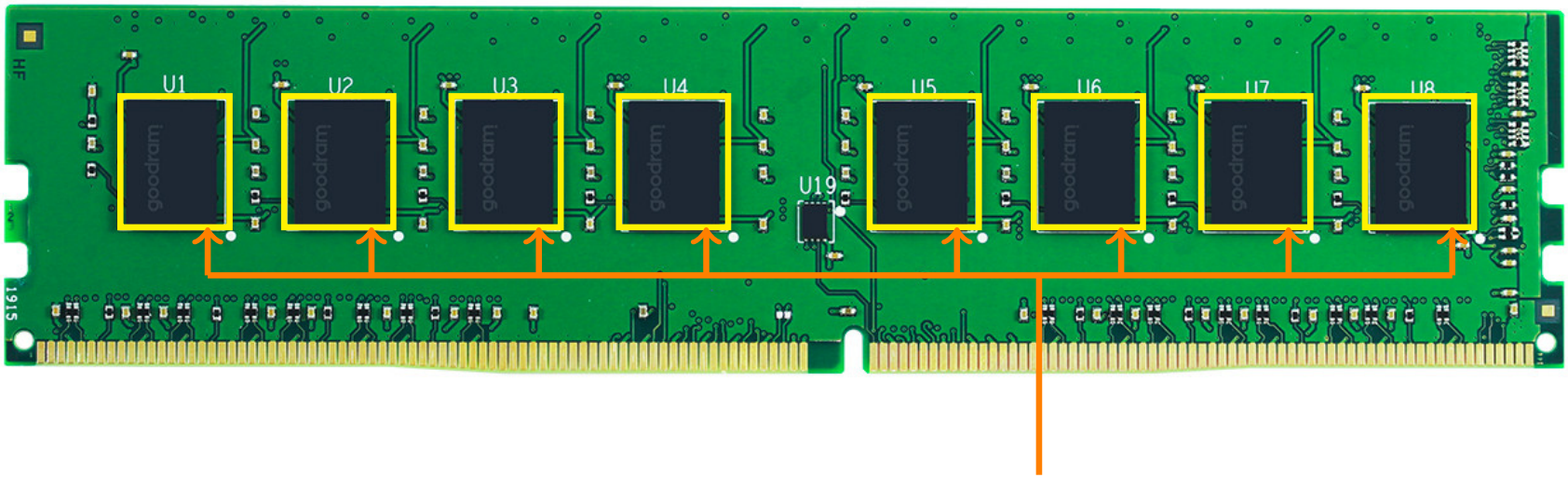
Un *DIMM* (une « barrette de RAM »)



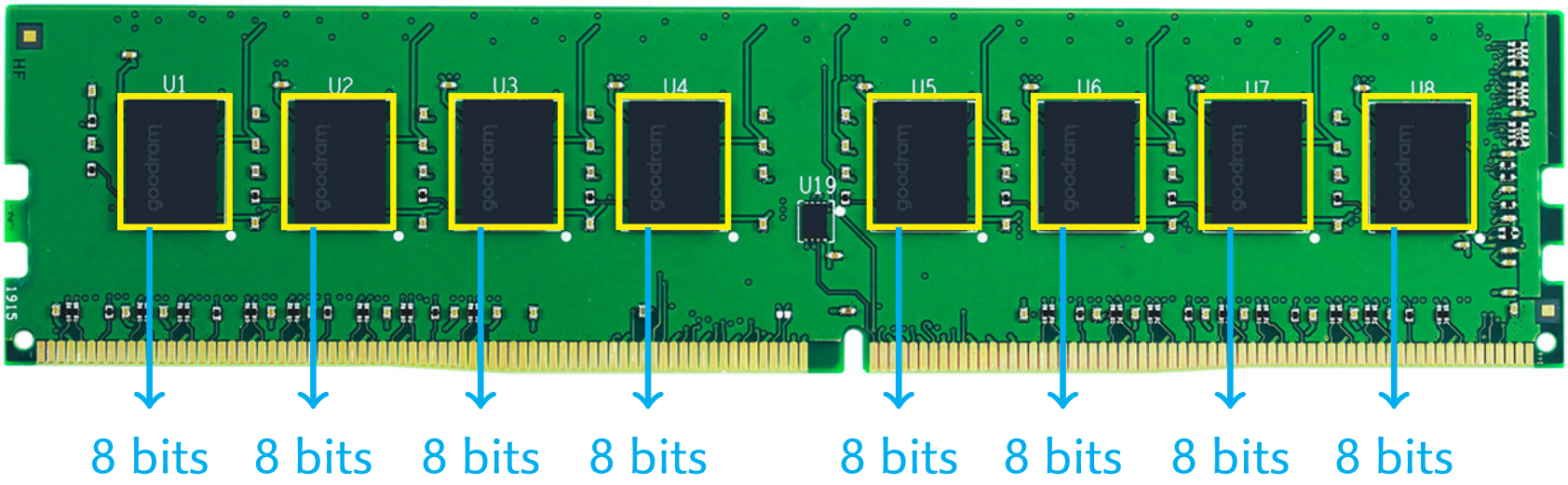
Un *DIMM* (une « barrette de RAM »)



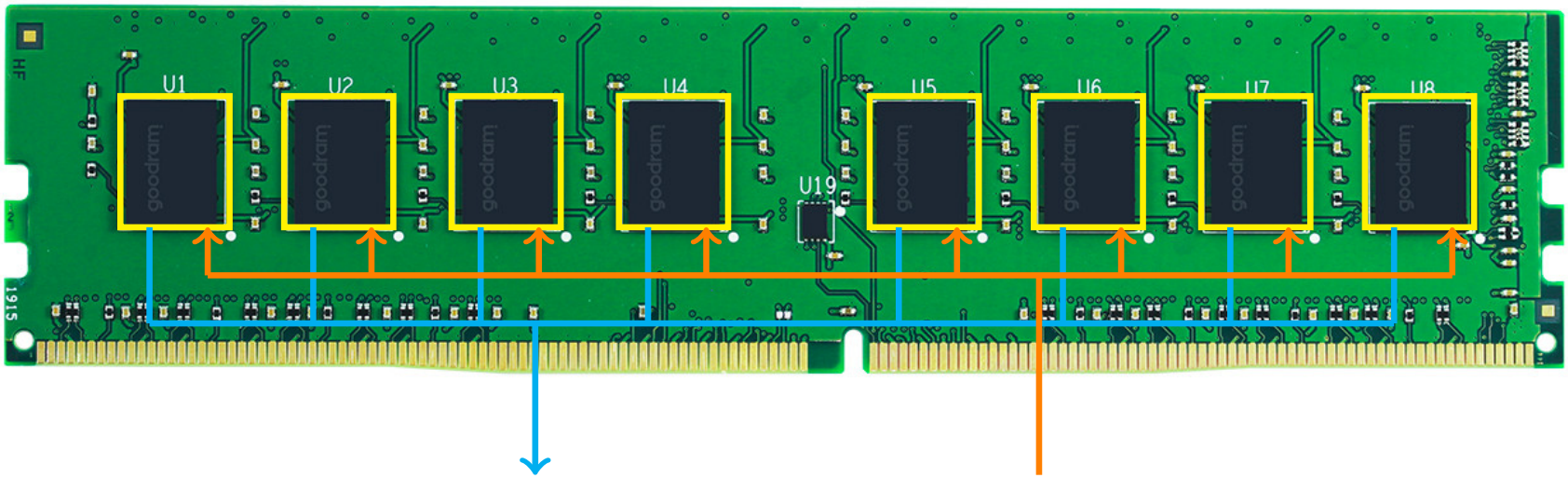
Un *DIMM* (une « barrette de RAM »)



Un *DIMM* (une « barrette de RAM »)



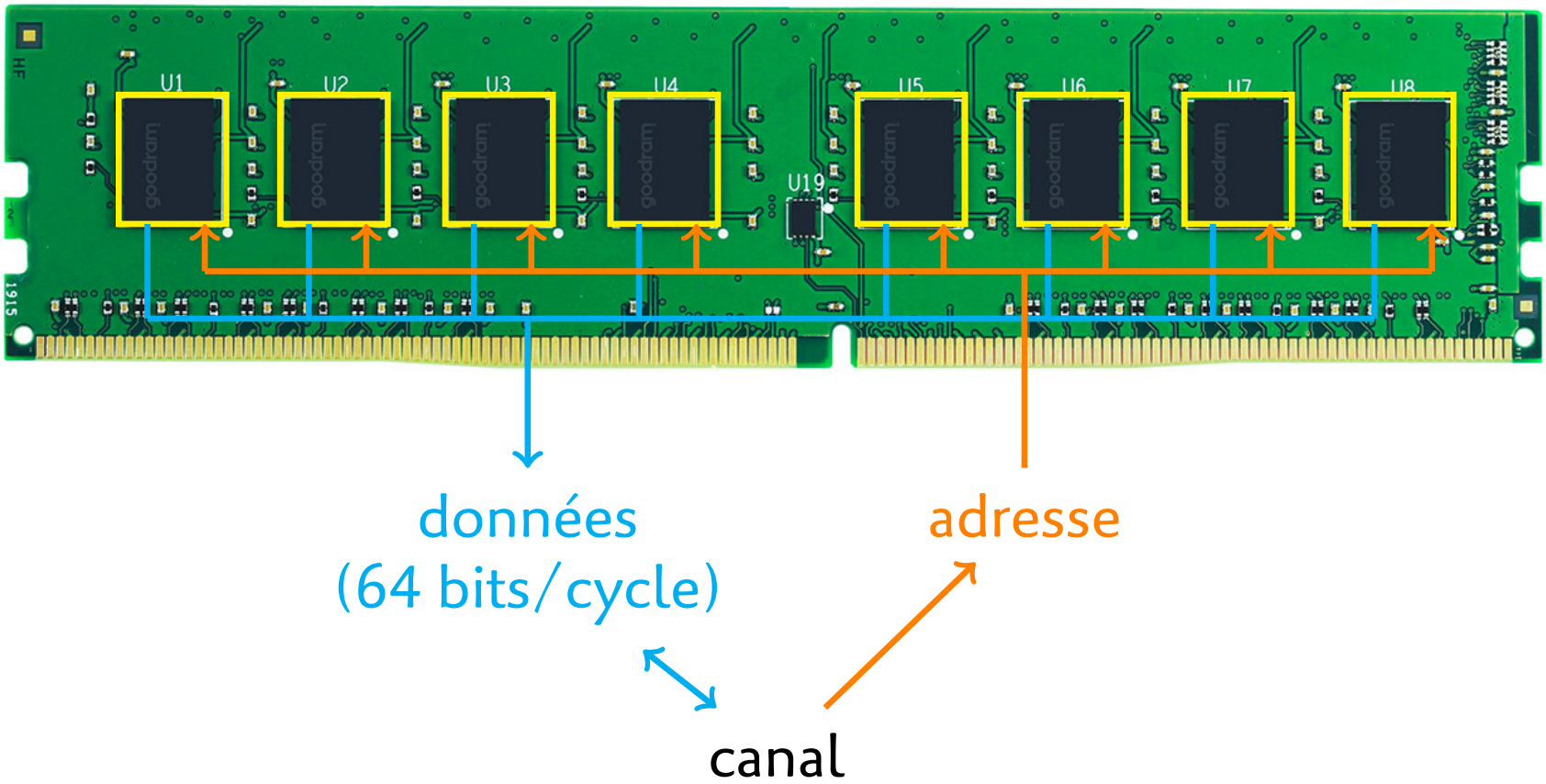
Un *DIMM* (une « barrette de RAM »)



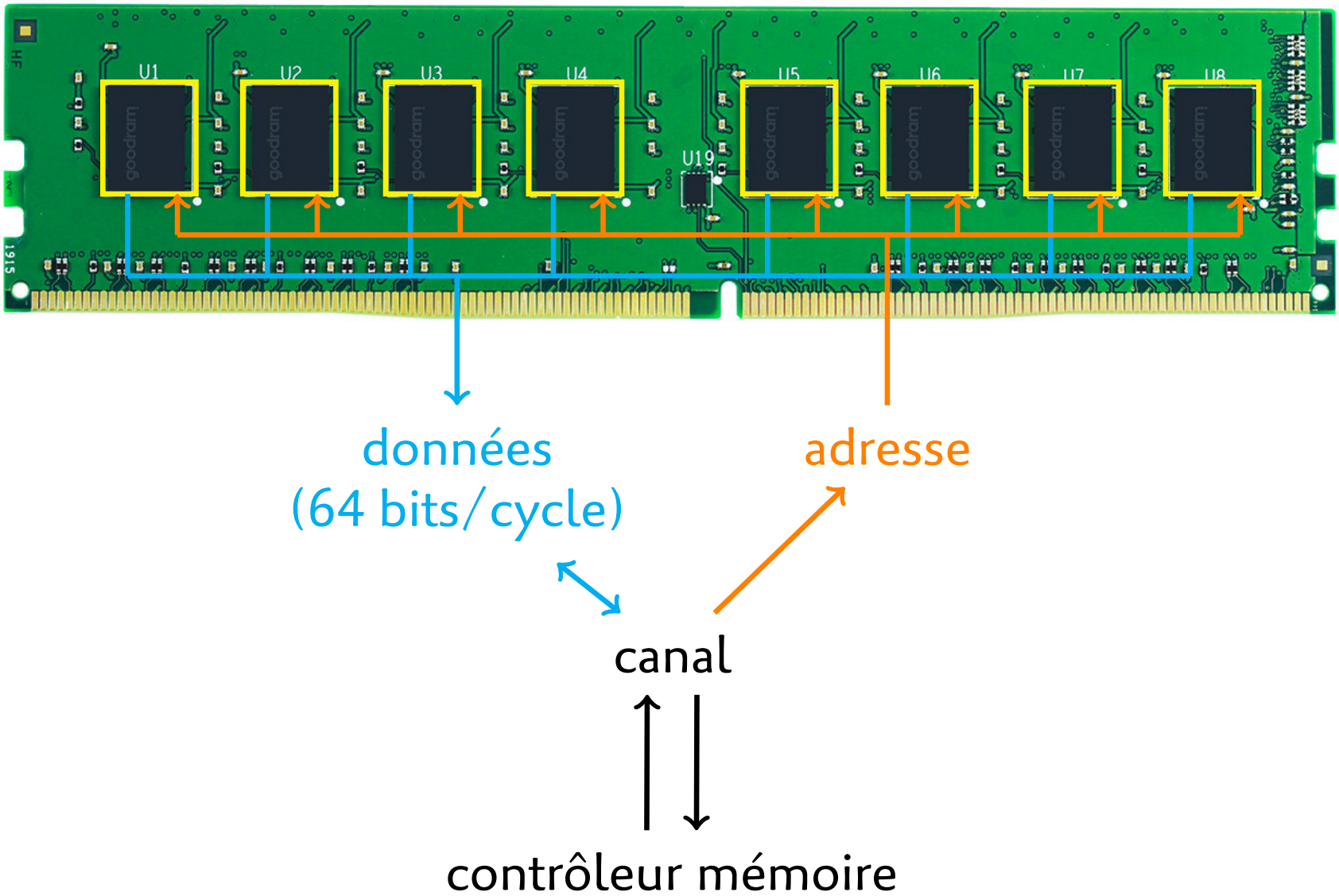
données
(64 bits/cycle)

adresse

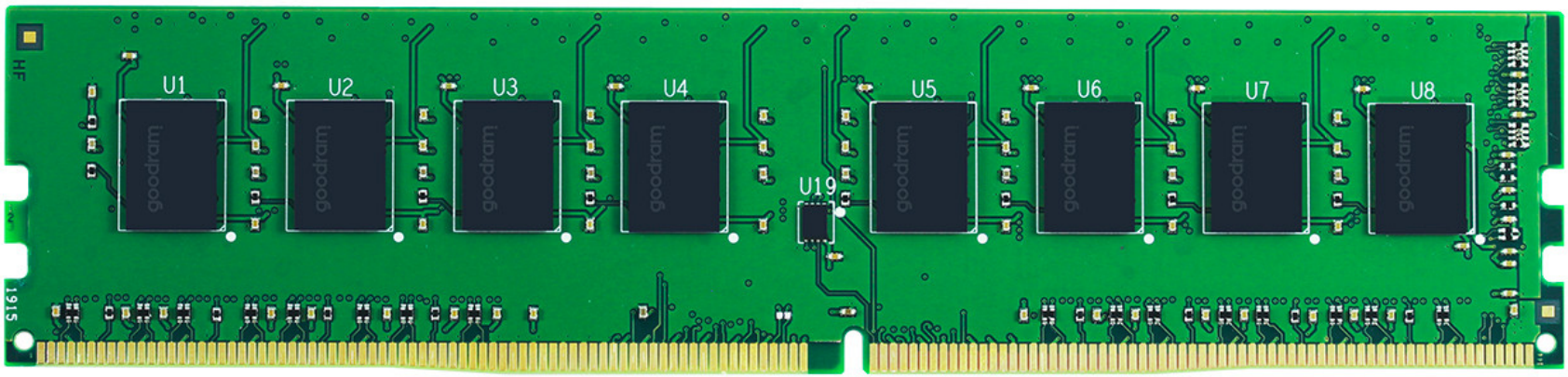
Un *DIMM* (une « barrette de RAM »)



Un *DIMM* (une « barrette de RAM »)



Un *DIMM* (une « barrette de RAM »)

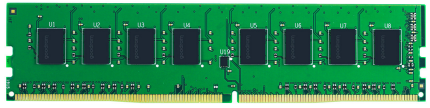


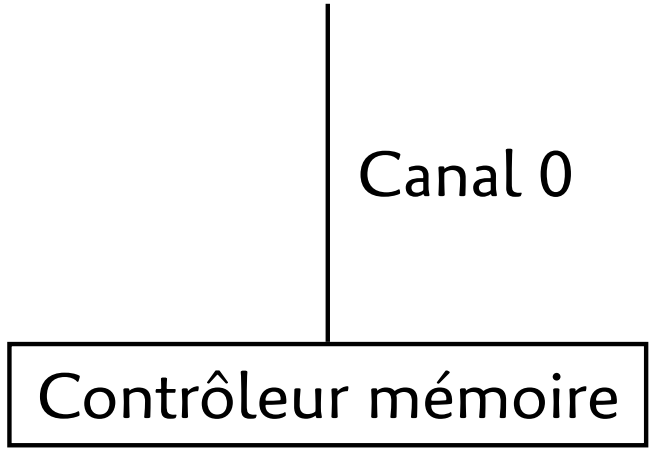
Génération	Année	Mhz	Prefetch	Go/s
DDR	2000	100–200	16	1.6–3.2
DDR2	2003	100–266	32	3.2–8.5
DDR3	2007	100–266	64	6.4–17
DDR4	2014	200–400	64	12.8–25.6

Canaux

Plus de parallélisme, plus de débit

≤ 2000

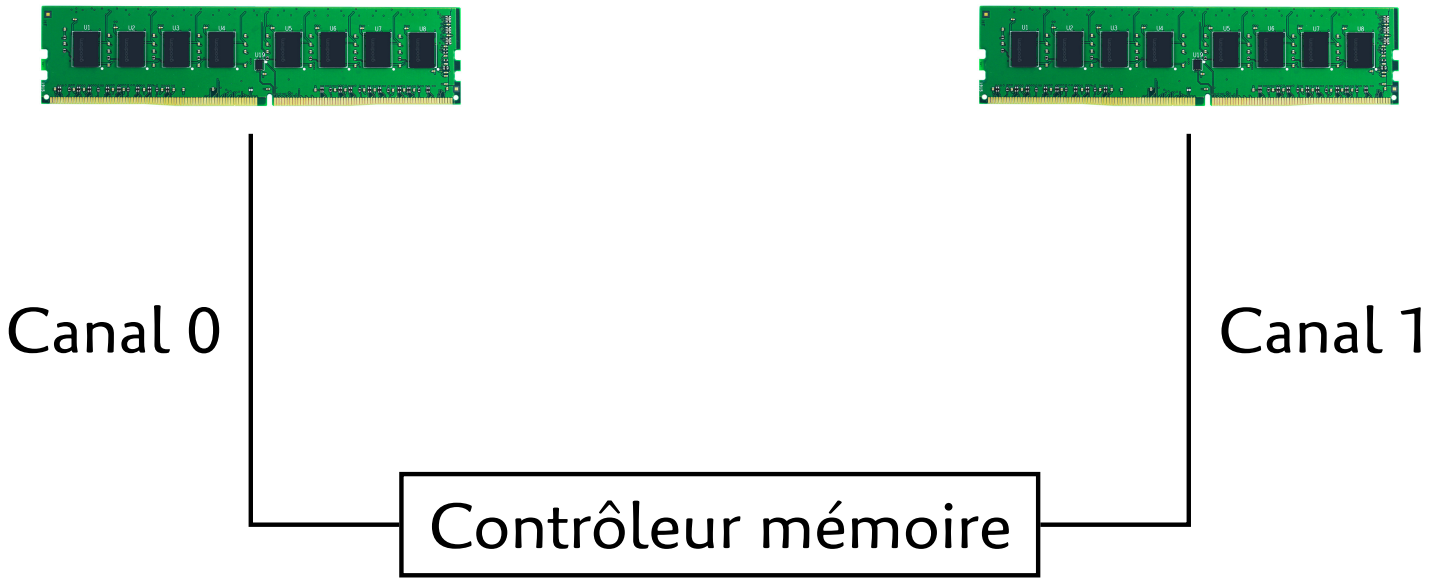




Canaux

Plus de parallélisme, plus de débit

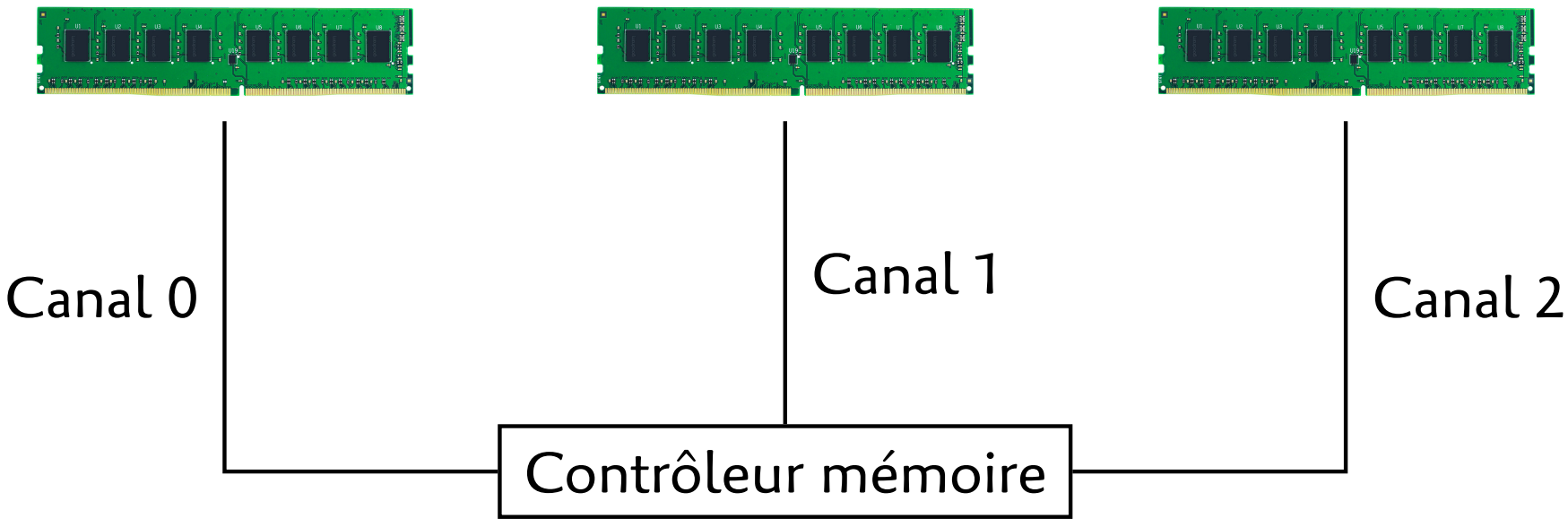
$\geq$ 2000, presque tous les « grand-public » (Core i3, i5, ...)



Canaux

Plus de parallélisme, plus de débit

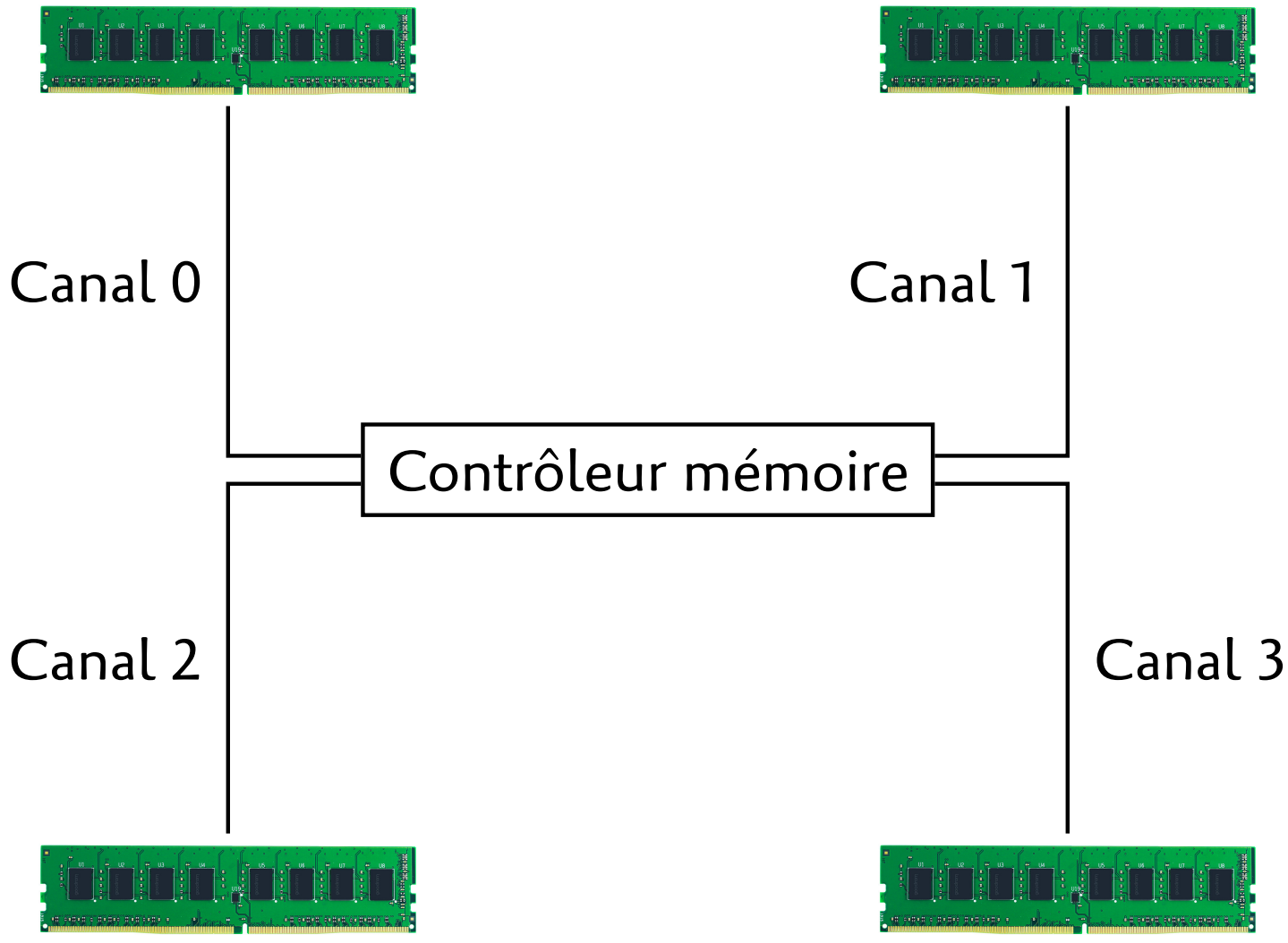
2008, Core i7 920 « Bloomfield »



Canaux

Plus de parallélisme, plus de débit

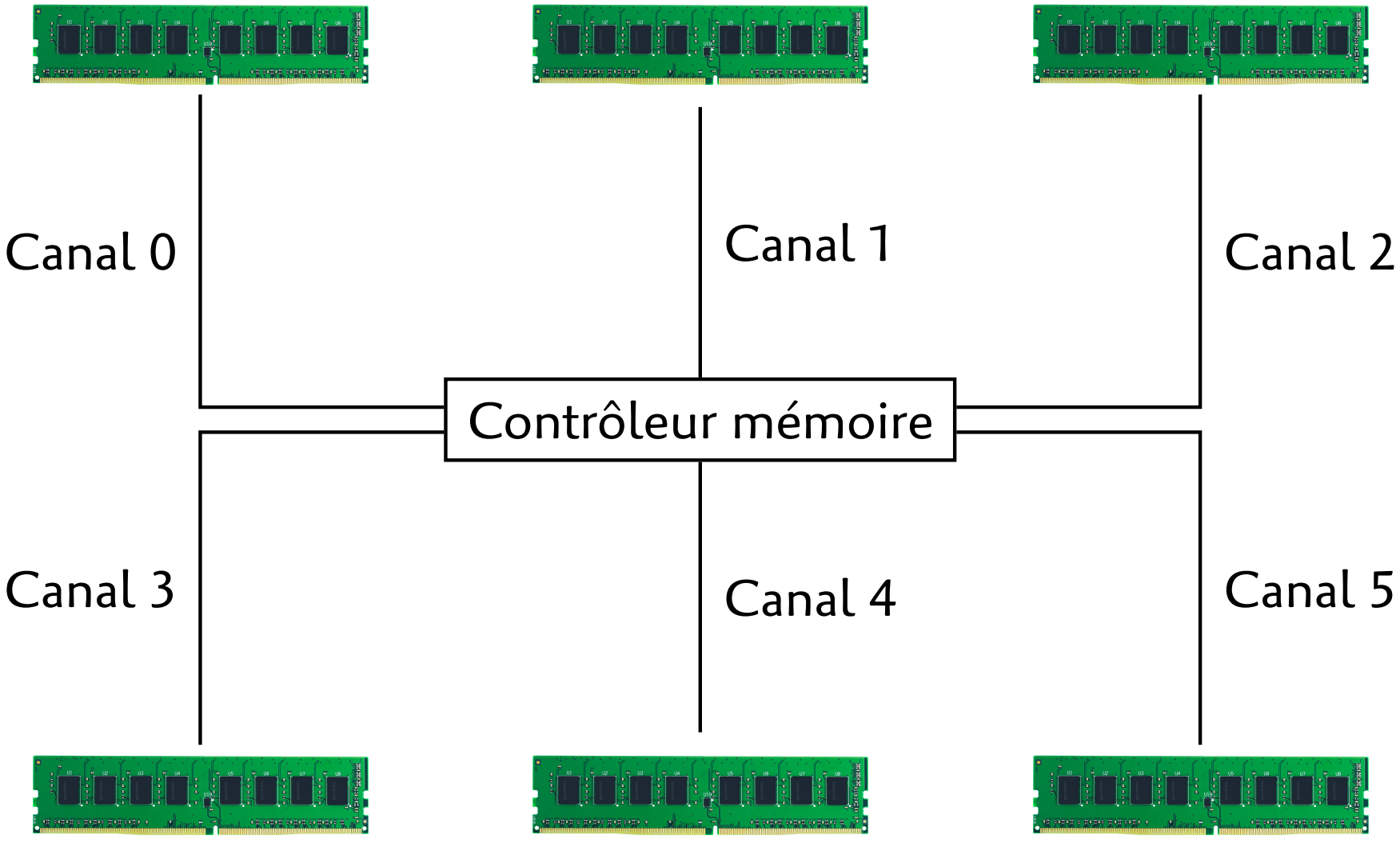
2010, AMD Opteron 6100, AMD Ryzen, Core i7/i9 « serie X »



Canaux

Plus de parallélisme, plus de débit

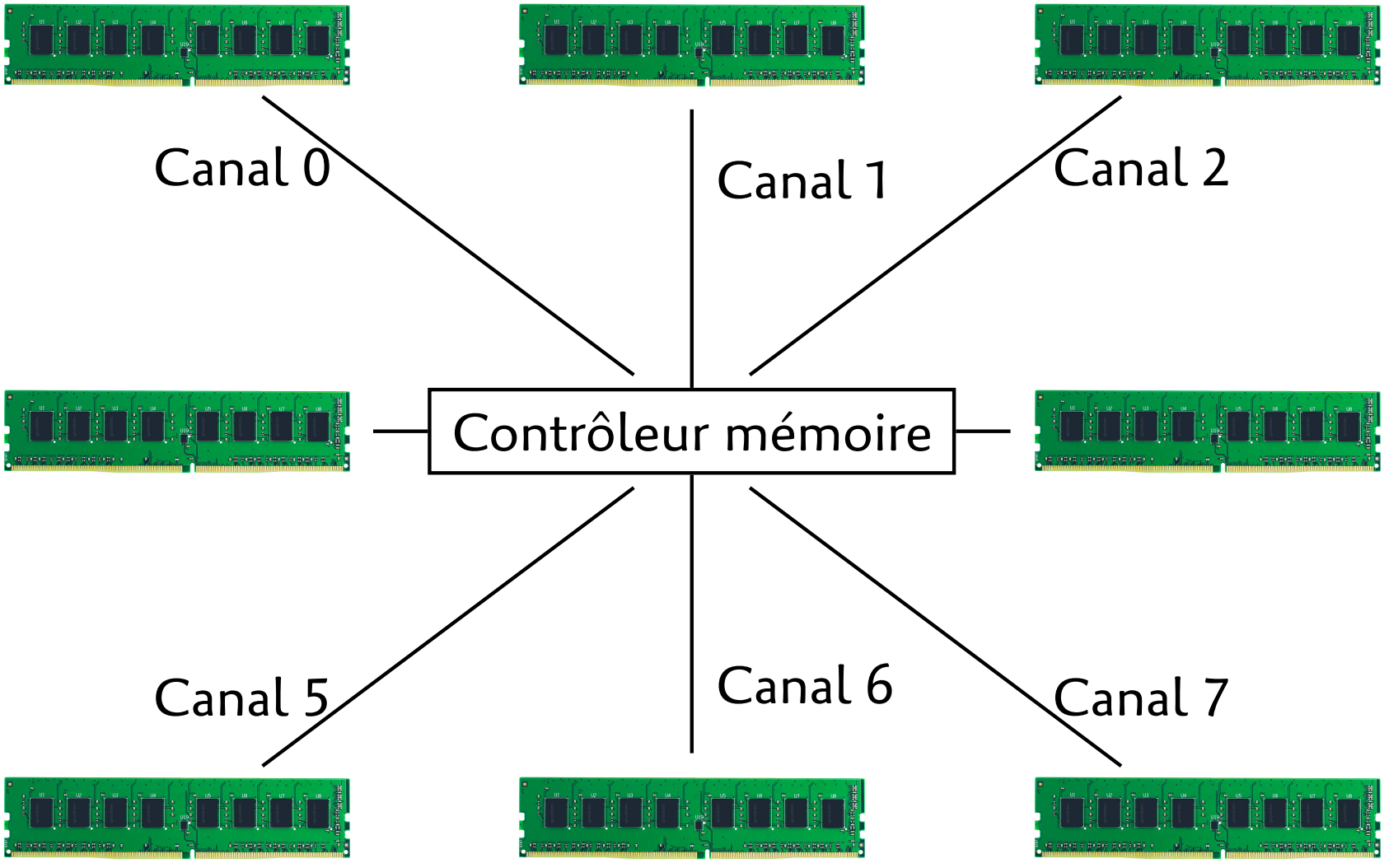
2017, Xeon Scalable (« Skylake »)



Canaux

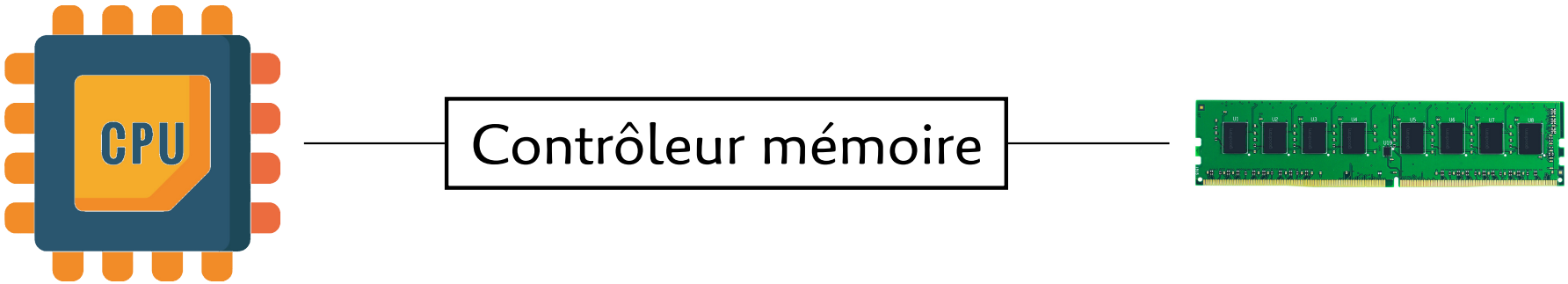
Plus de parallélisme, plus de débit

2019, AMD Epyc, IBM Power9

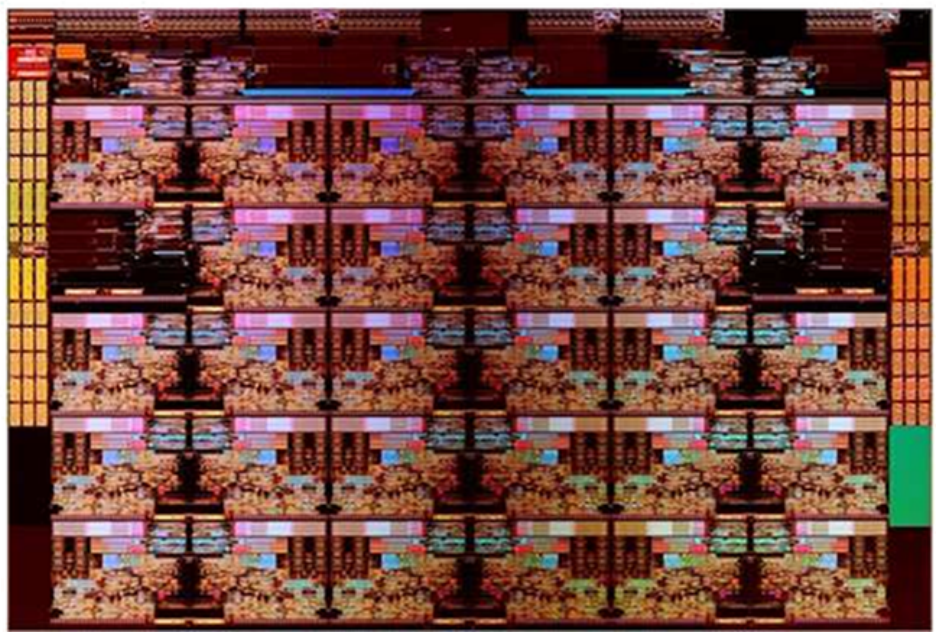


Plus près, plus vite

$\leq$ 2008 : contrôleur sur la carte mère (« chipset, north bridge »)

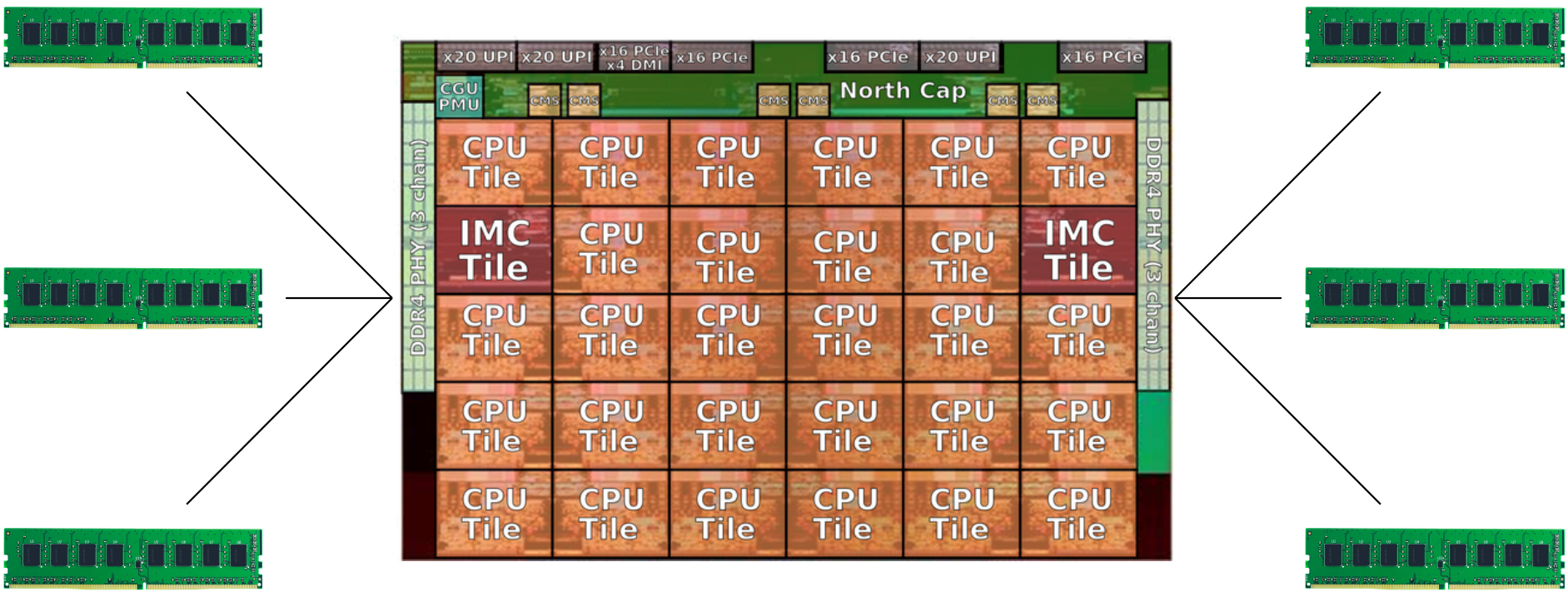


Plus près, plus vite

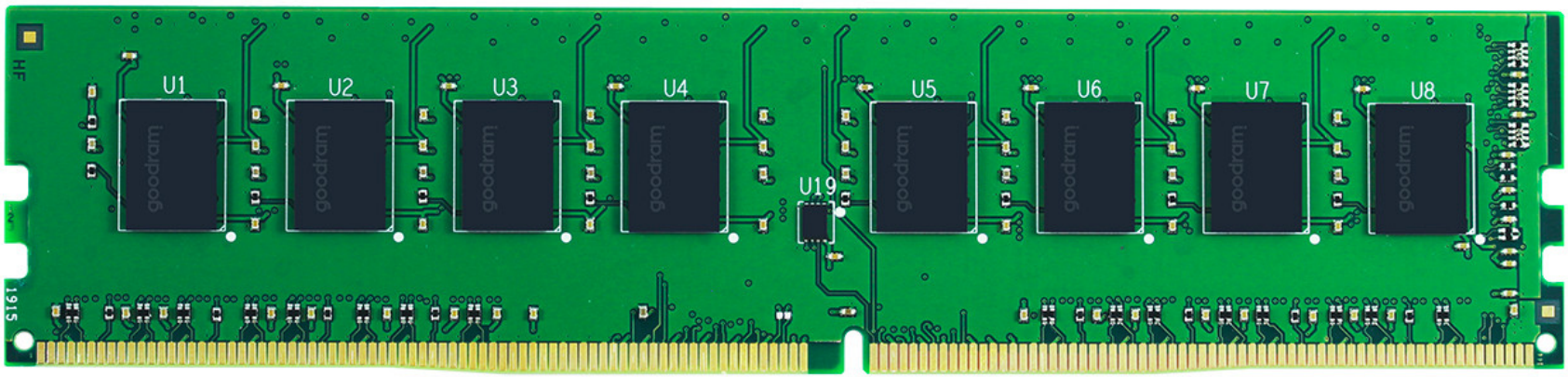


Plus près, plus vite

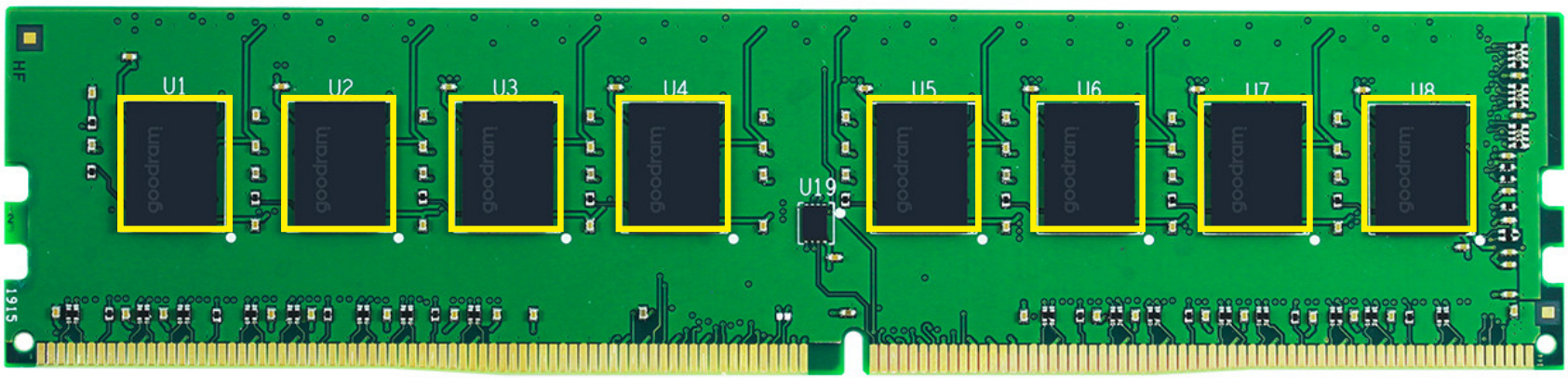
$\geq$ 2008, contrôleur(s) dans le cpu



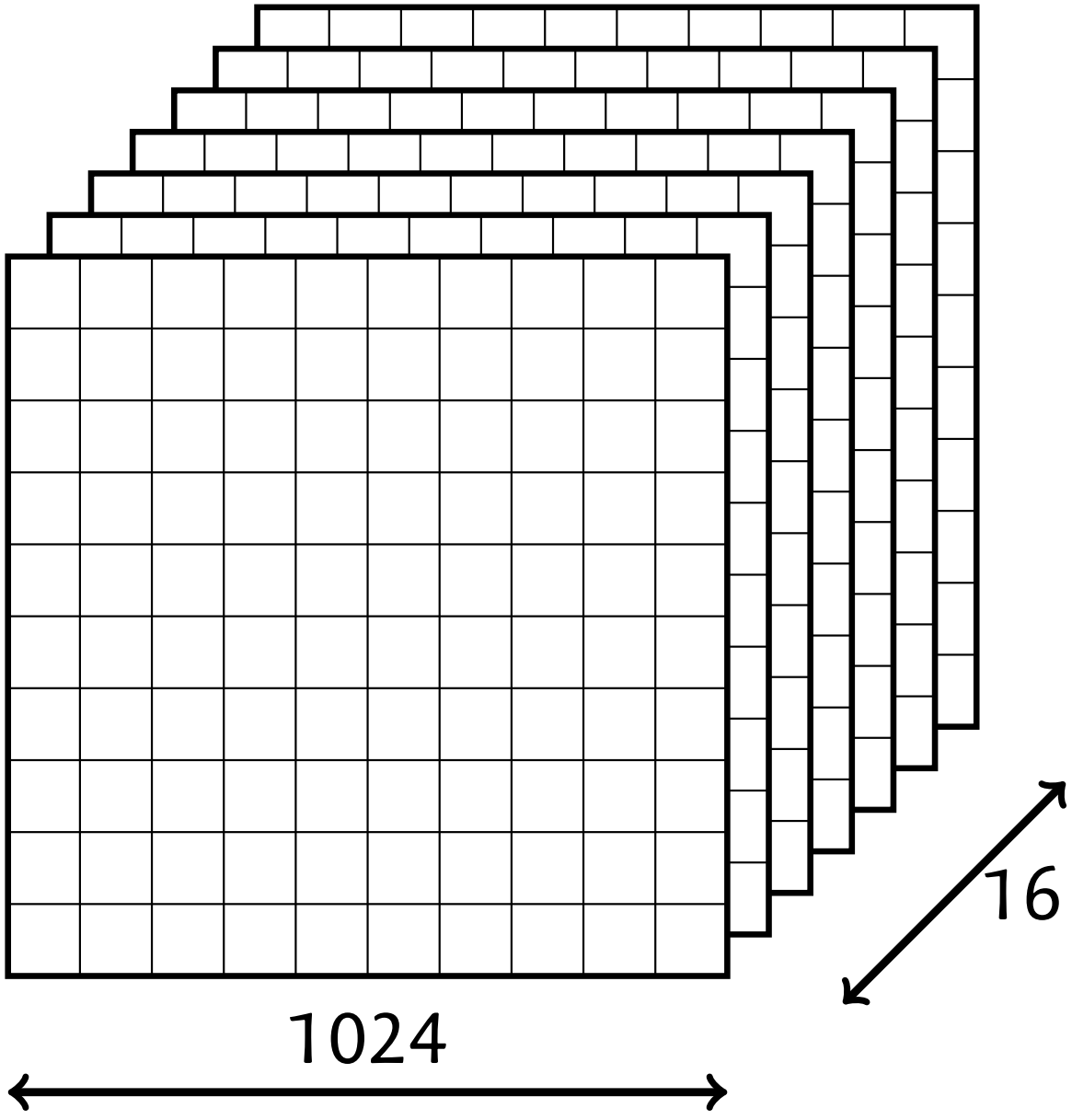
Plus dans les détails



Plus dans les détails



Rows, Columns and Banks



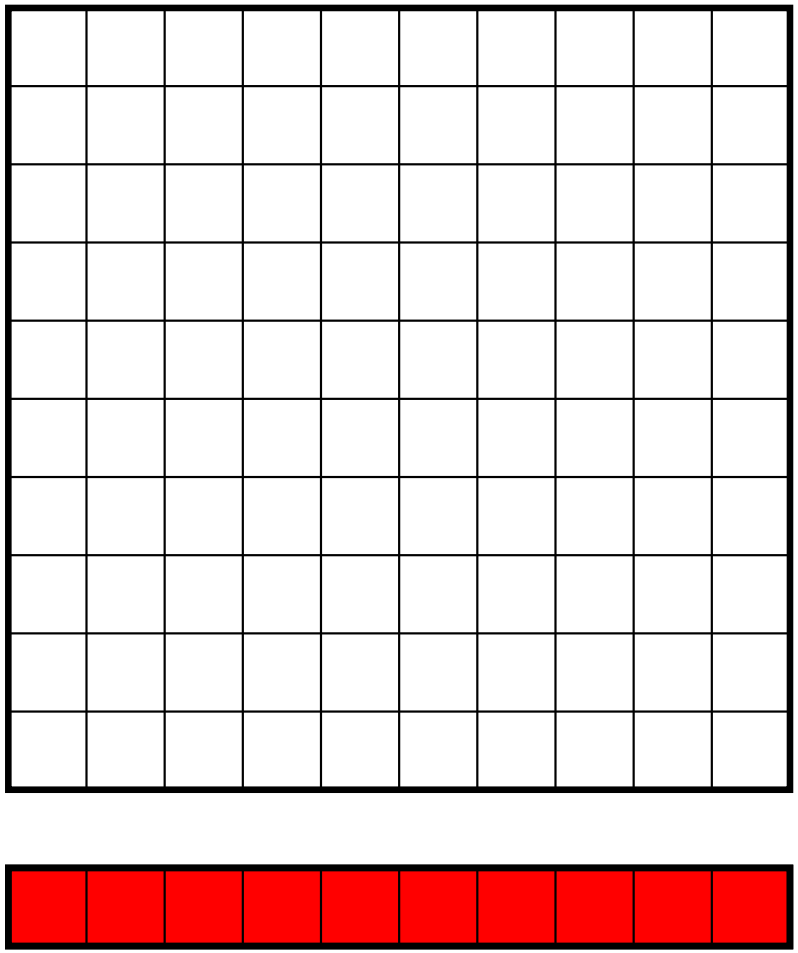
4 bits $\mapsto$ bank

10 bits $\mapsto$ column

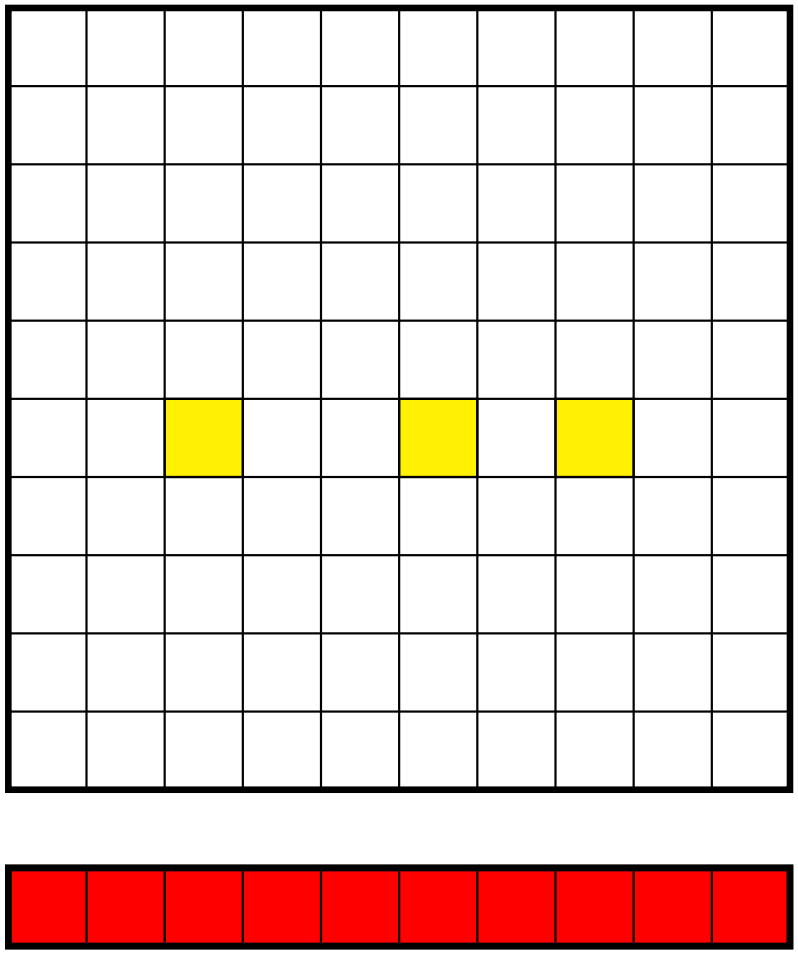
x bits $\mapsto$ row

(pour DDR4)

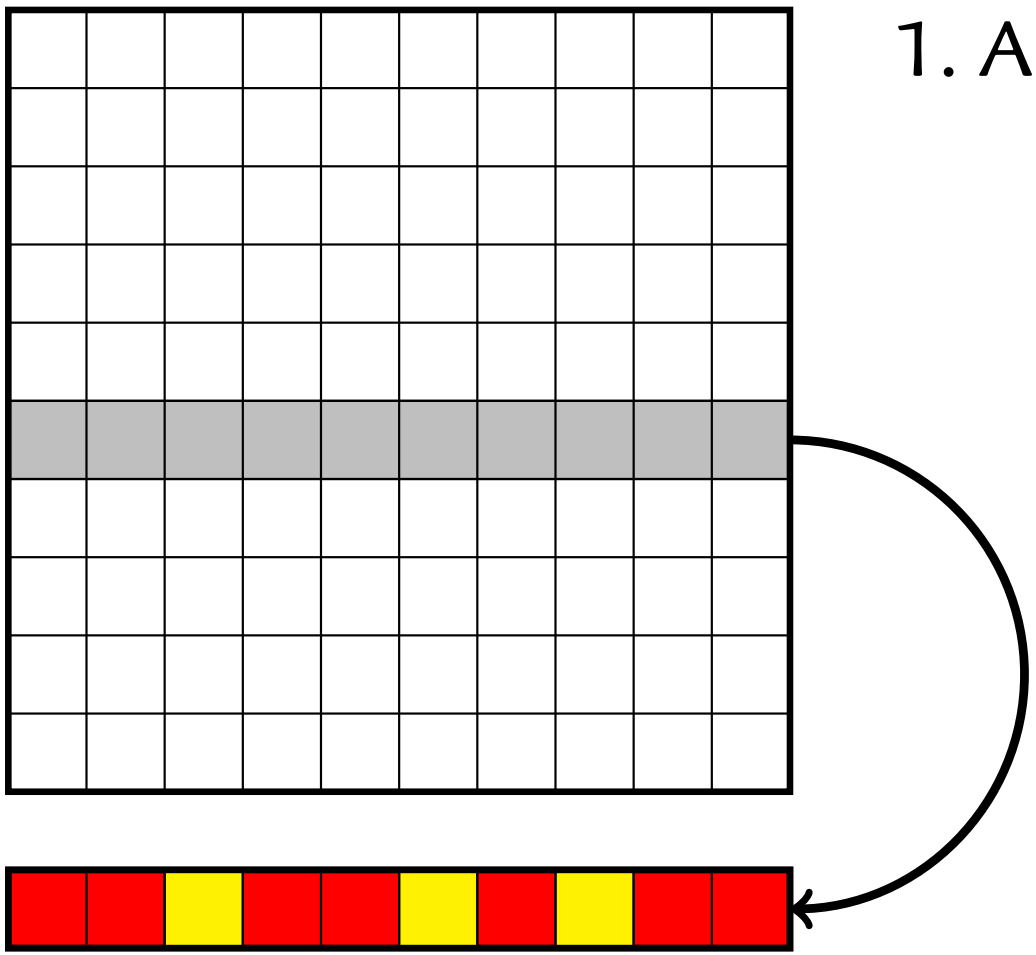
Rows, Columns, ...



Rows, Columns, ...

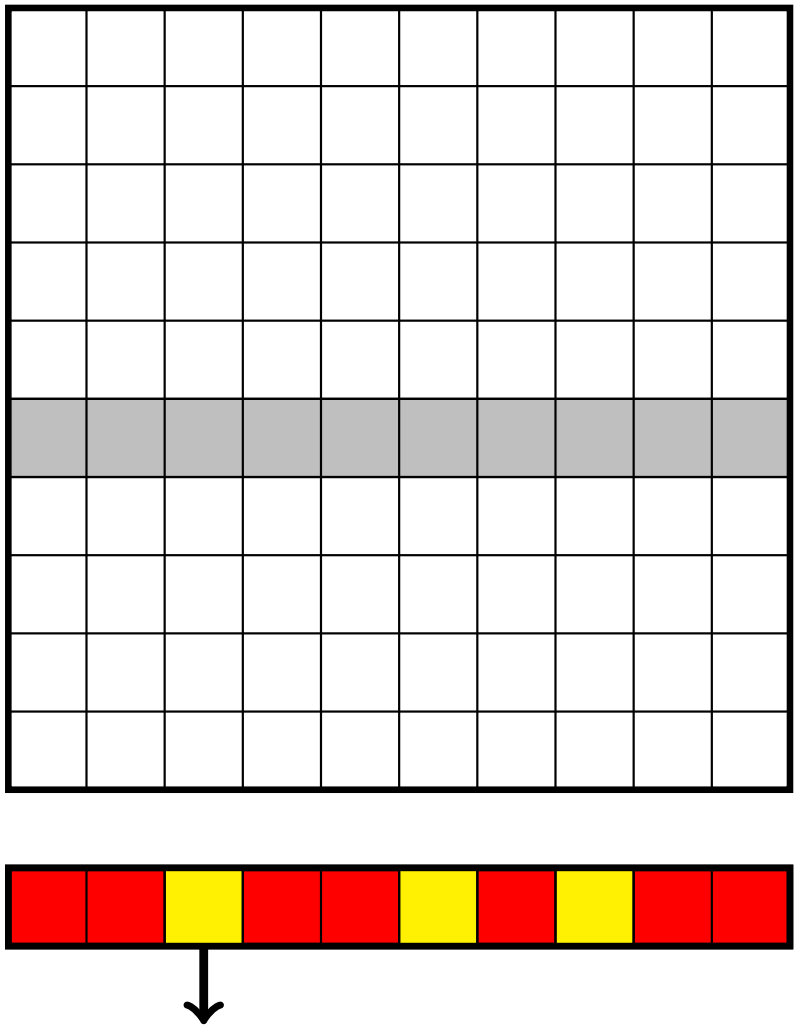


Rows, Columns, ...



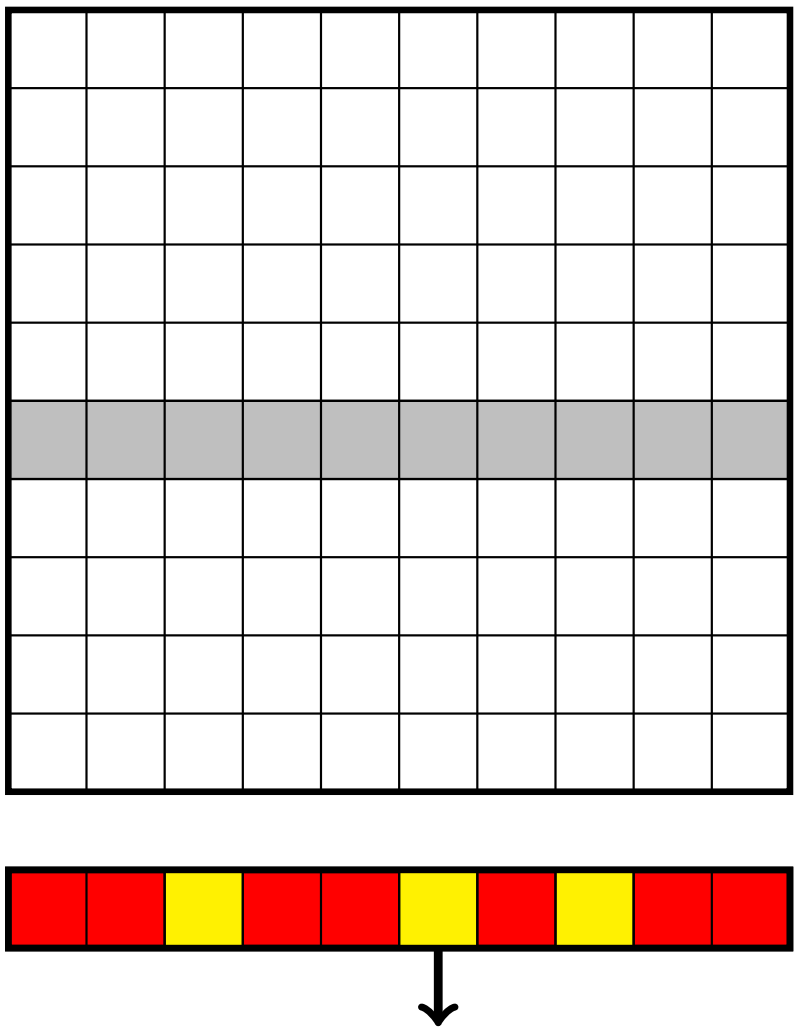
1. ACTivate row

Rows, Columns, ...



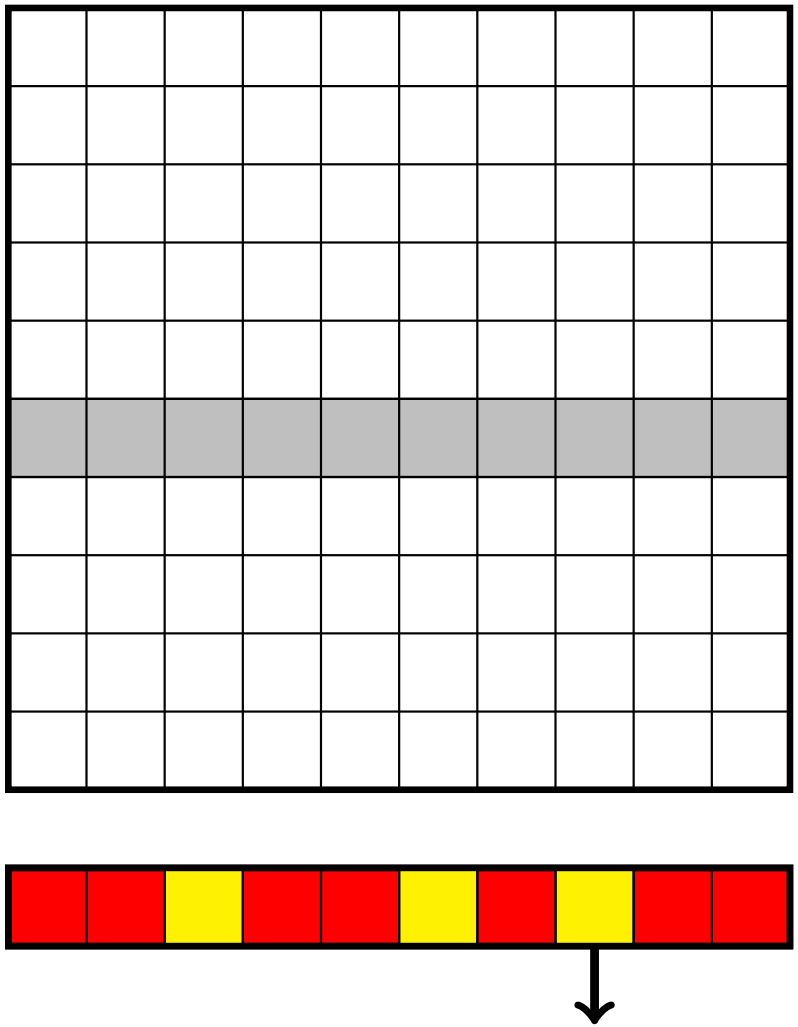
1. ACTivate row
2. READ column

Rows, Columns, ...



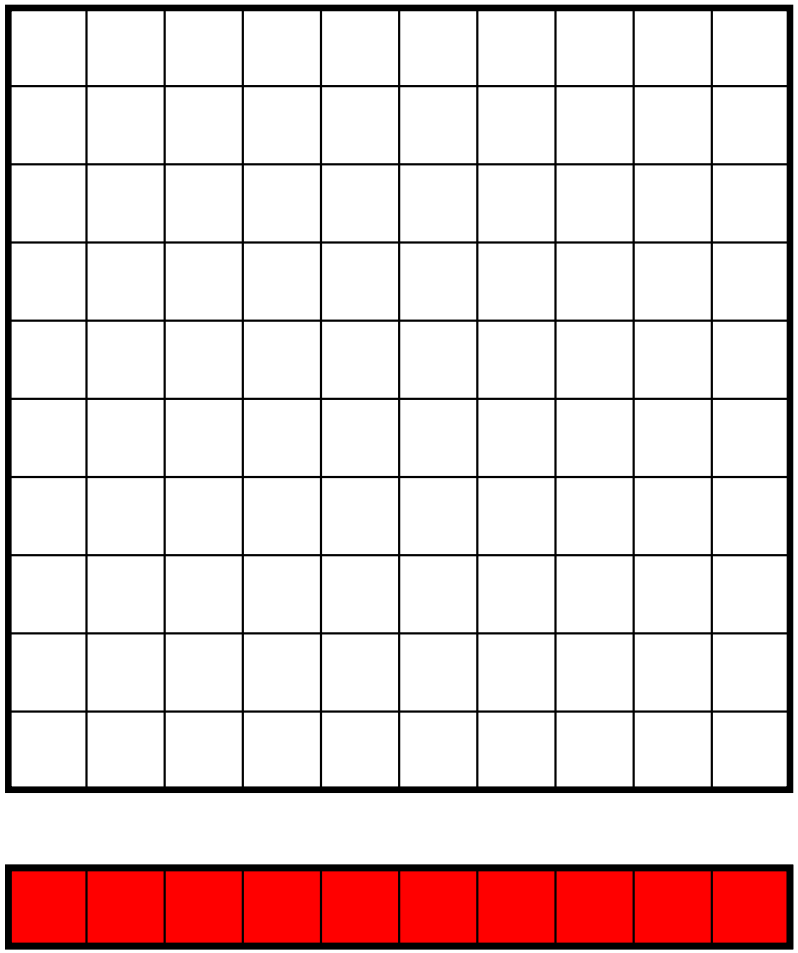
1. ACTivate row
2. READ column
3. READ column

Rows, Columns, ...



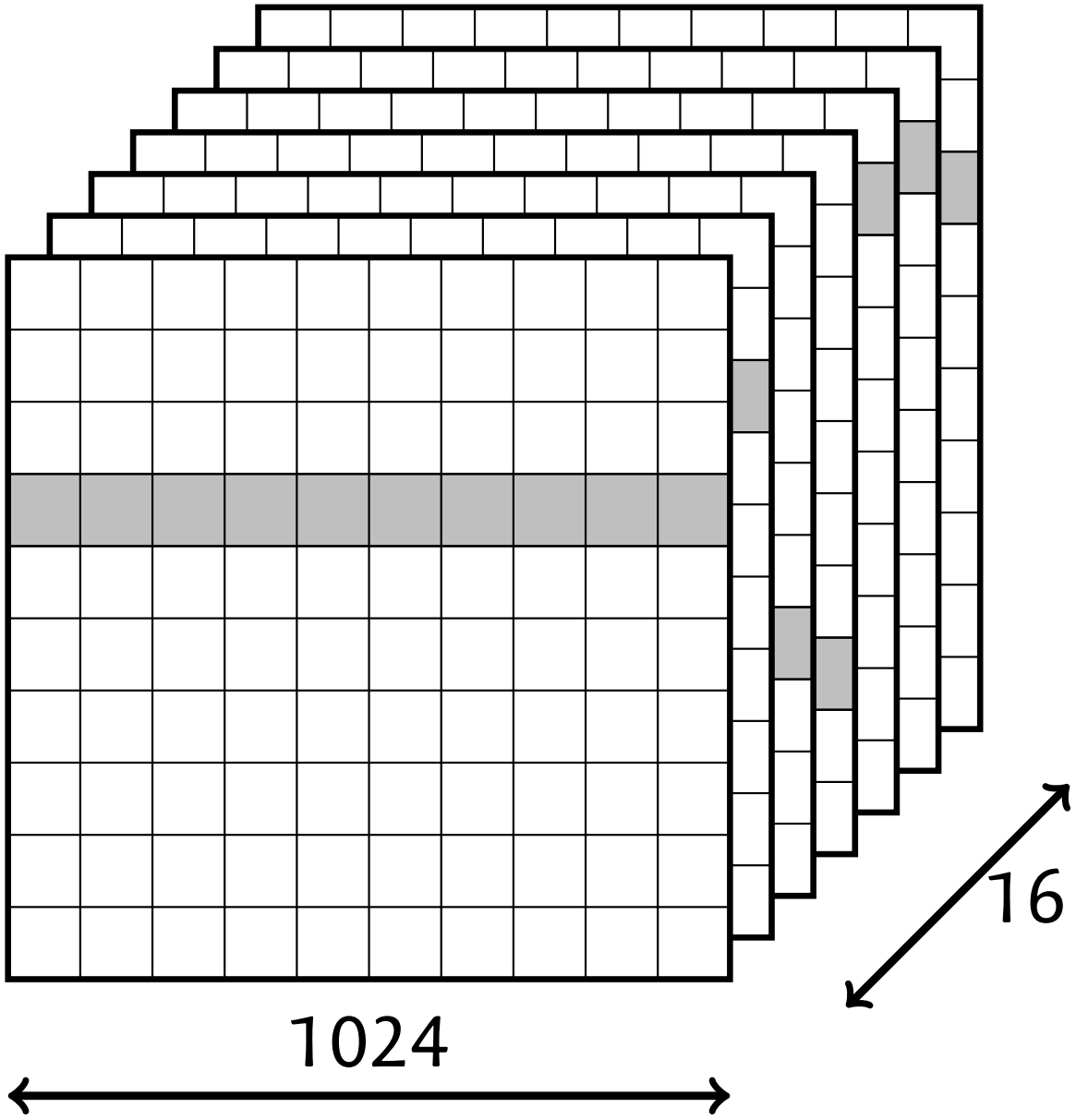
1. ACTivate row
2. READ column
3. READ column
4. READ column

Rows, Columns, ...



1. ACTivate row
2. READ column
3. READ column
4. READ column
5. PREcharge (reset)

Rows, Columns and Banks



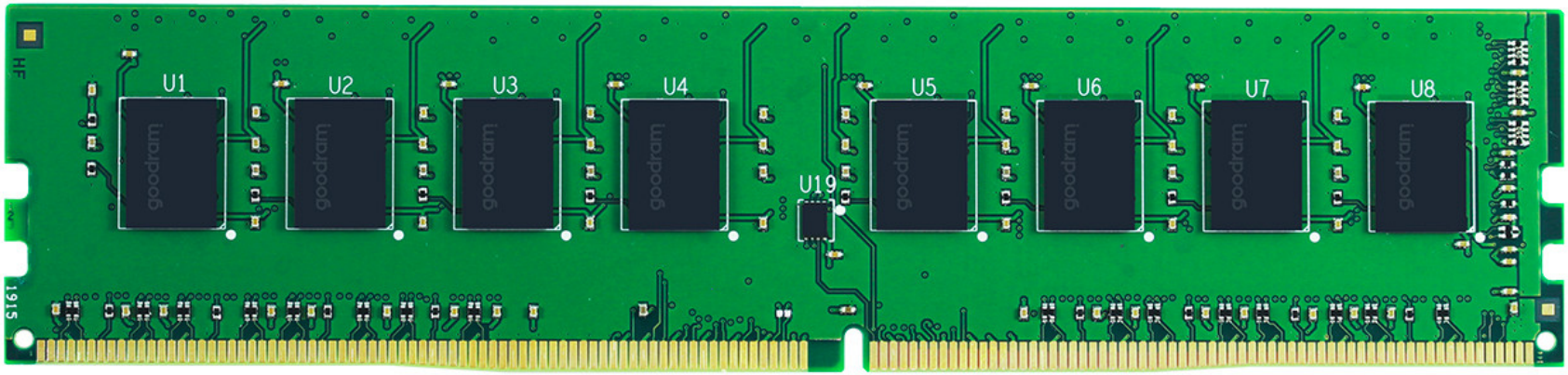
4 bits $\mapsto$ bank

10 bits $\mapsto$ column

x bits $\mapsto$ row

(pour DDR4)

En 2020 : la DDR4

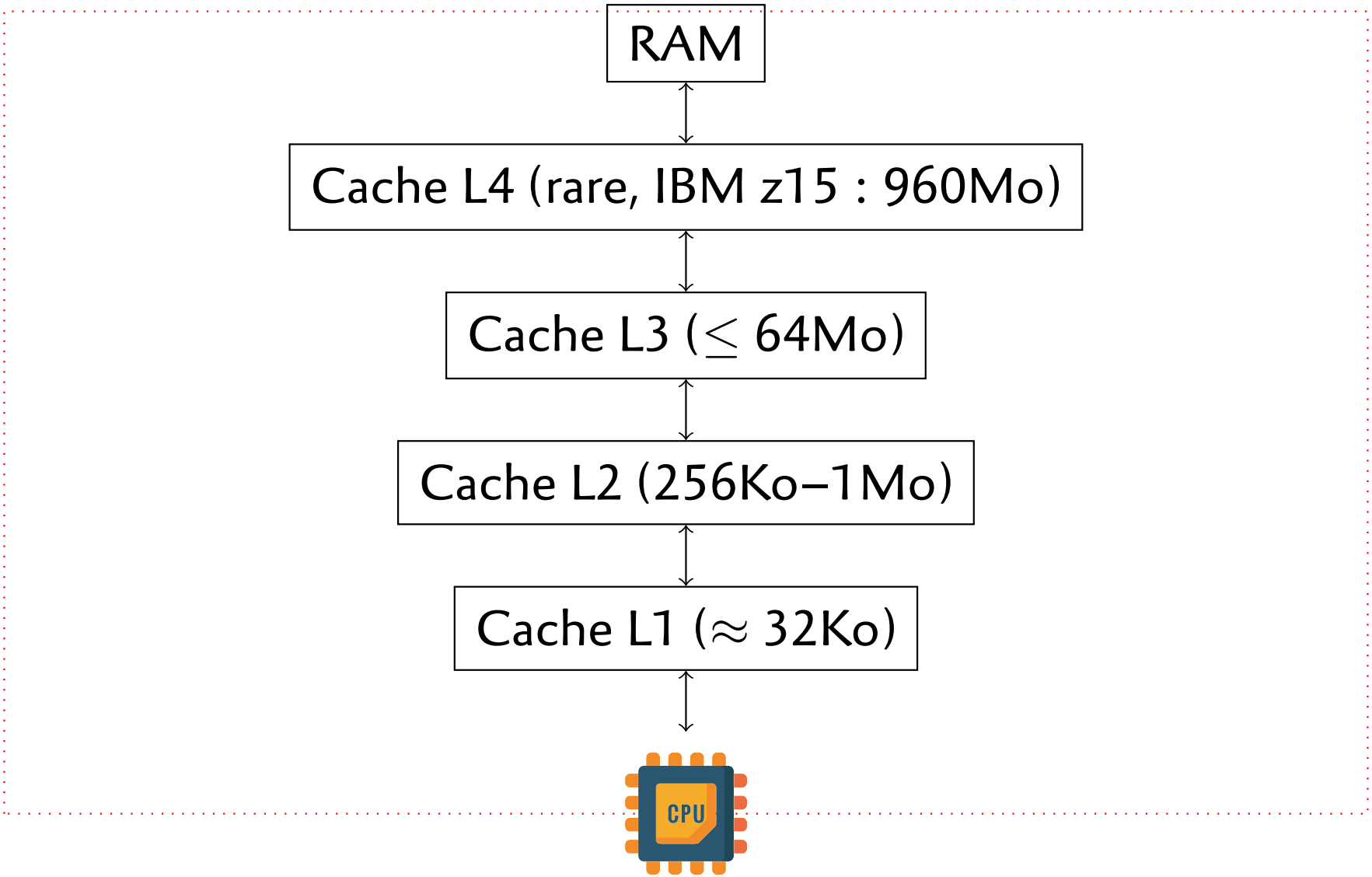


Standard	Freq. (Mhz)	Go/s	CL min.	Latence (ns)
DDR4-1600	200	12.8	10	12.5
DDR4-1866	233	14.9	12	12.9
DDR4-2133	266	17.0	14	13.1
DDR4-2400	300	19.2	15	12.5
DDR4-2666	333	21.3	17	12.8
DDR4-2933	366	23.5	19	13.0
DDR4-3200	400	25.6	20	12.5

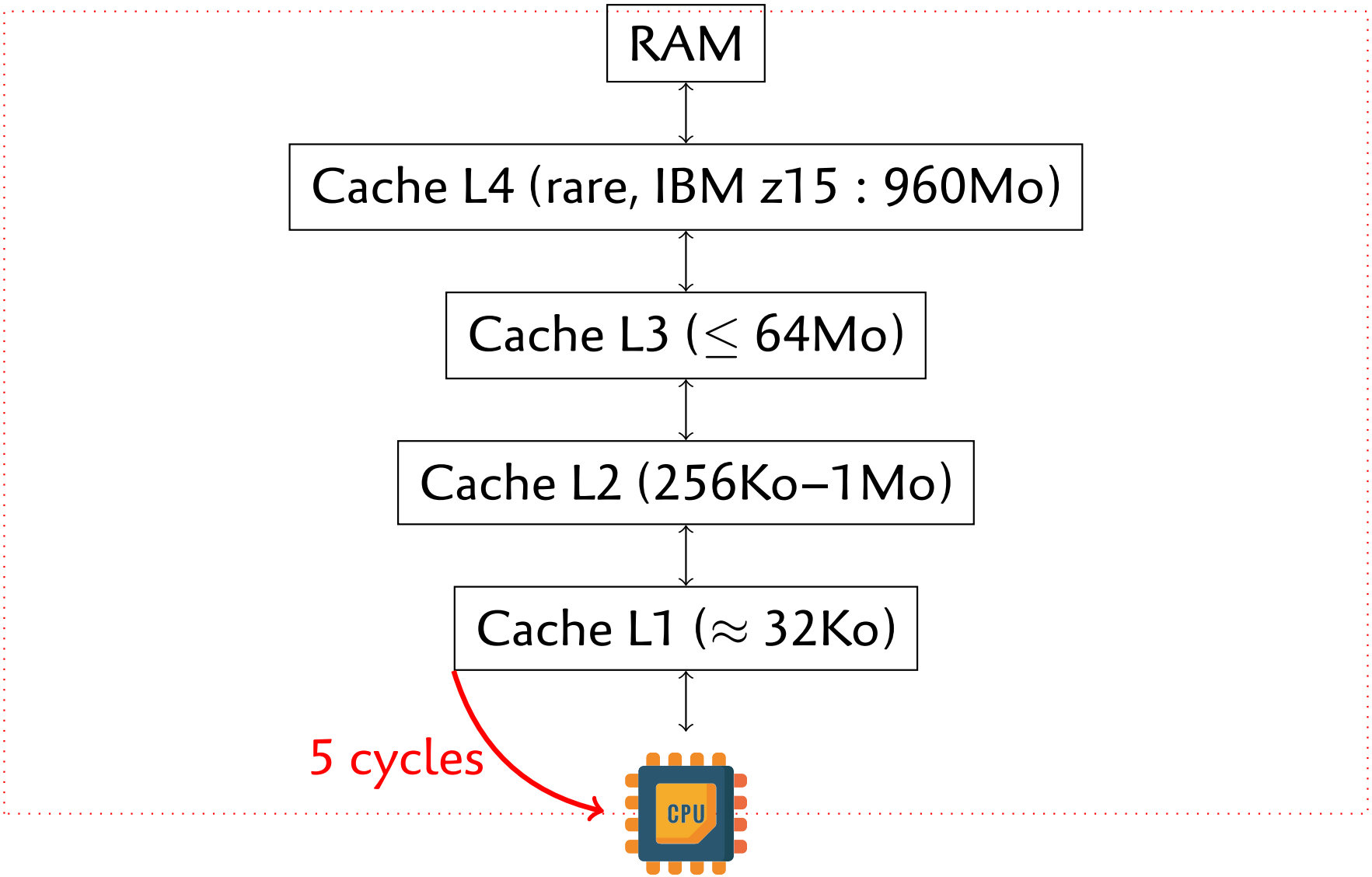
Plan

1. Description du hardware
2. La hiérarchie mémoire (caches)
3. Amélioration de la localité des données
4. (bonus) Problèmes liés à la pagination

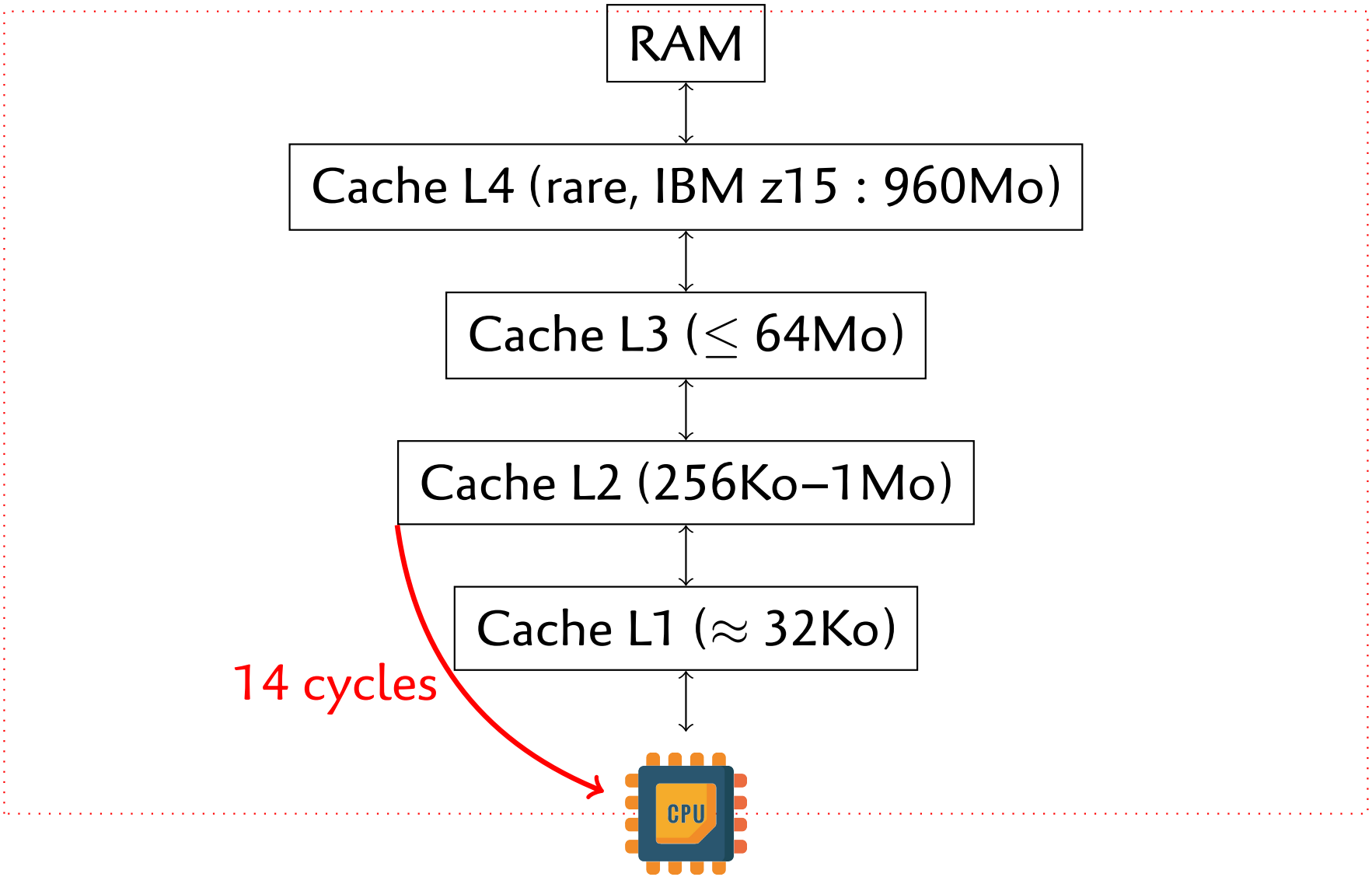
La hiérarchie mémoire (les caches)



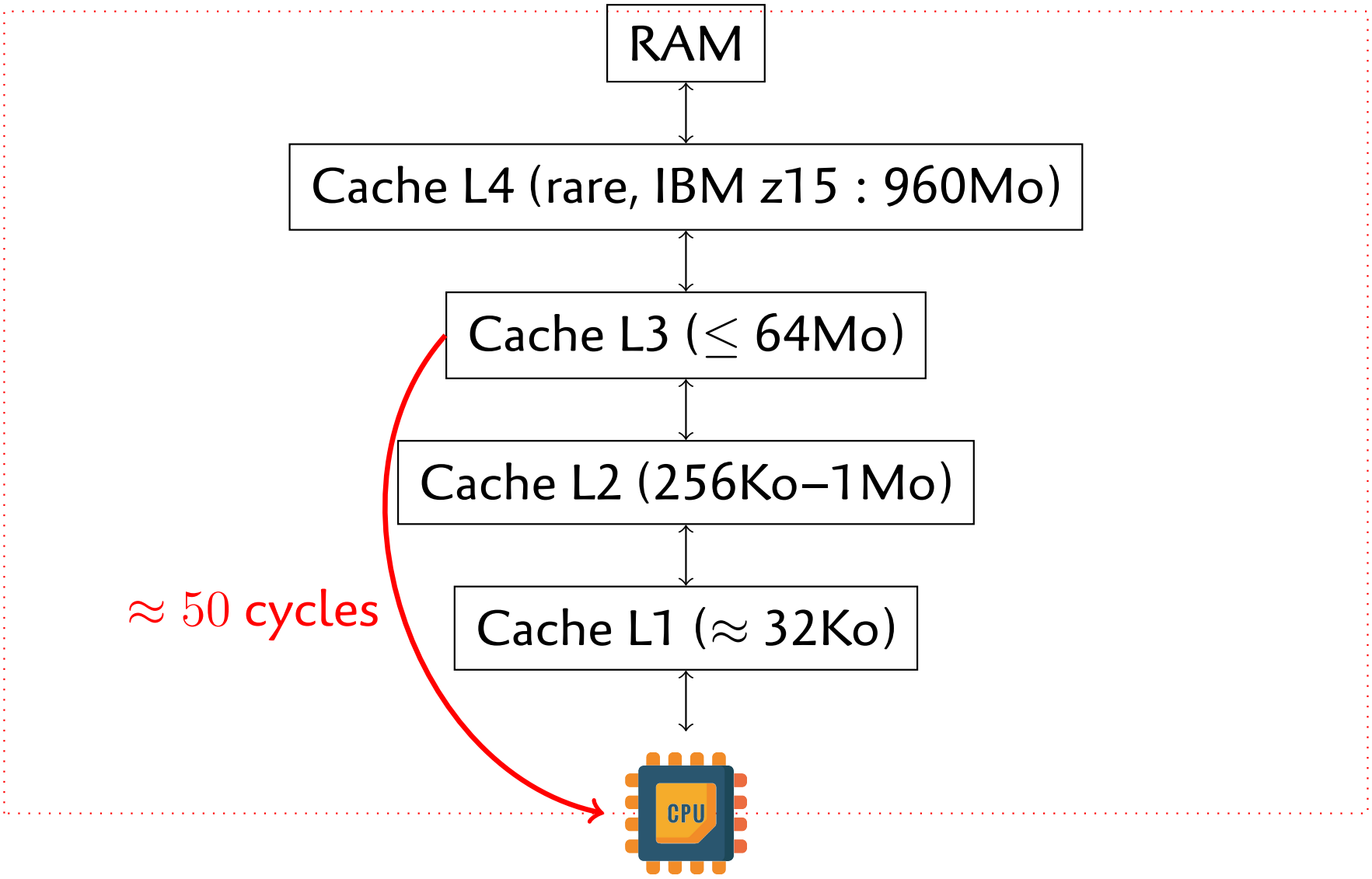
La hiérarchie mémoire (les caches)



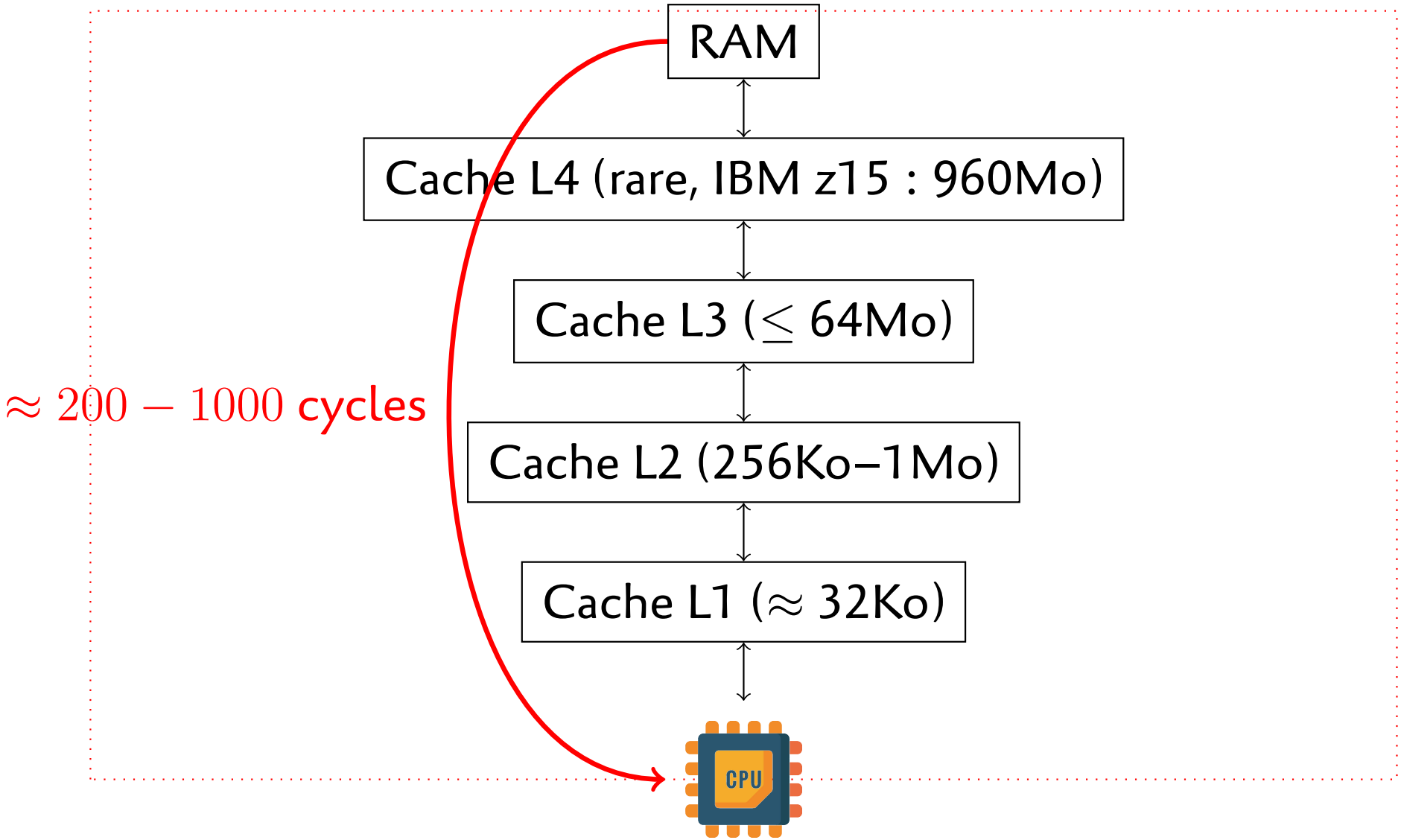
La hiérarchie mémoire (les caches)



La hiérarchie mémoire (les caches)



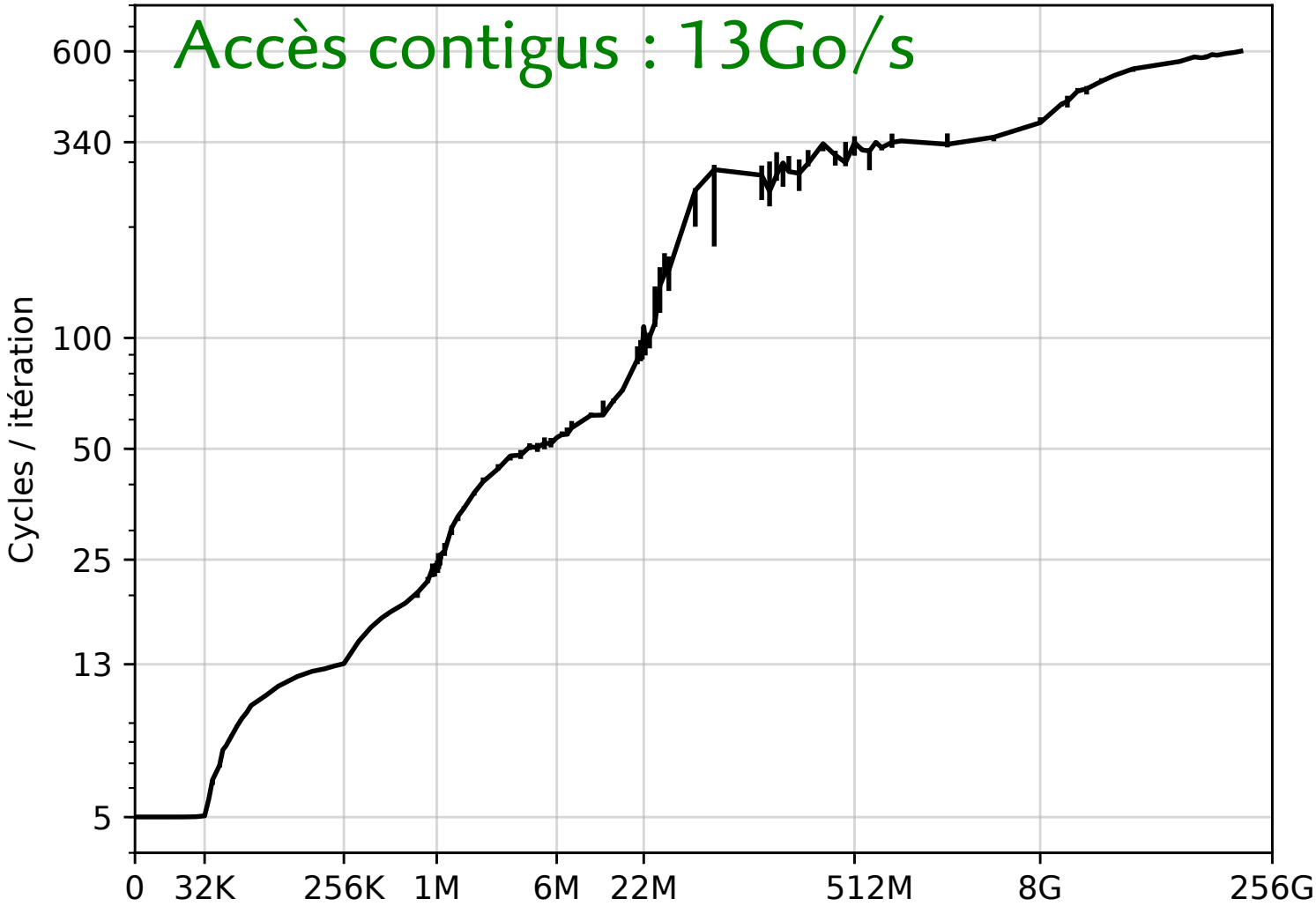
La hiérarchie mémoire (les caches)



Effet de la hiérarchie mémoire

$T =$ tableau de N entiers aléatoires dans $[0; N[$.

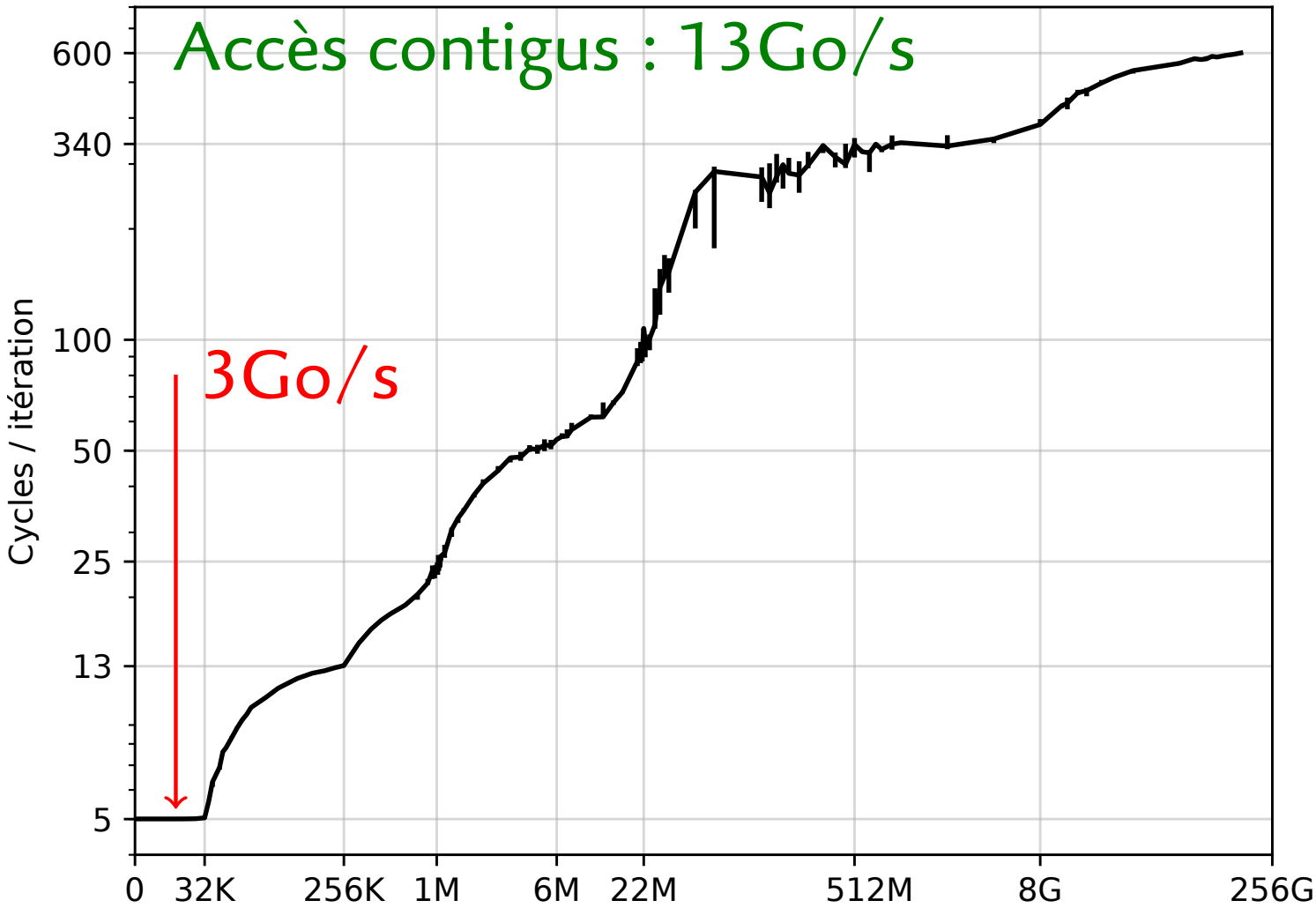
```
for (int i=0, x=0; i < 1000000000; i++) x = T[x];
```



Effet de la hiérarchie mémoire

$T =$ tableau de N entiers aléatoires dans $[0; N[$.

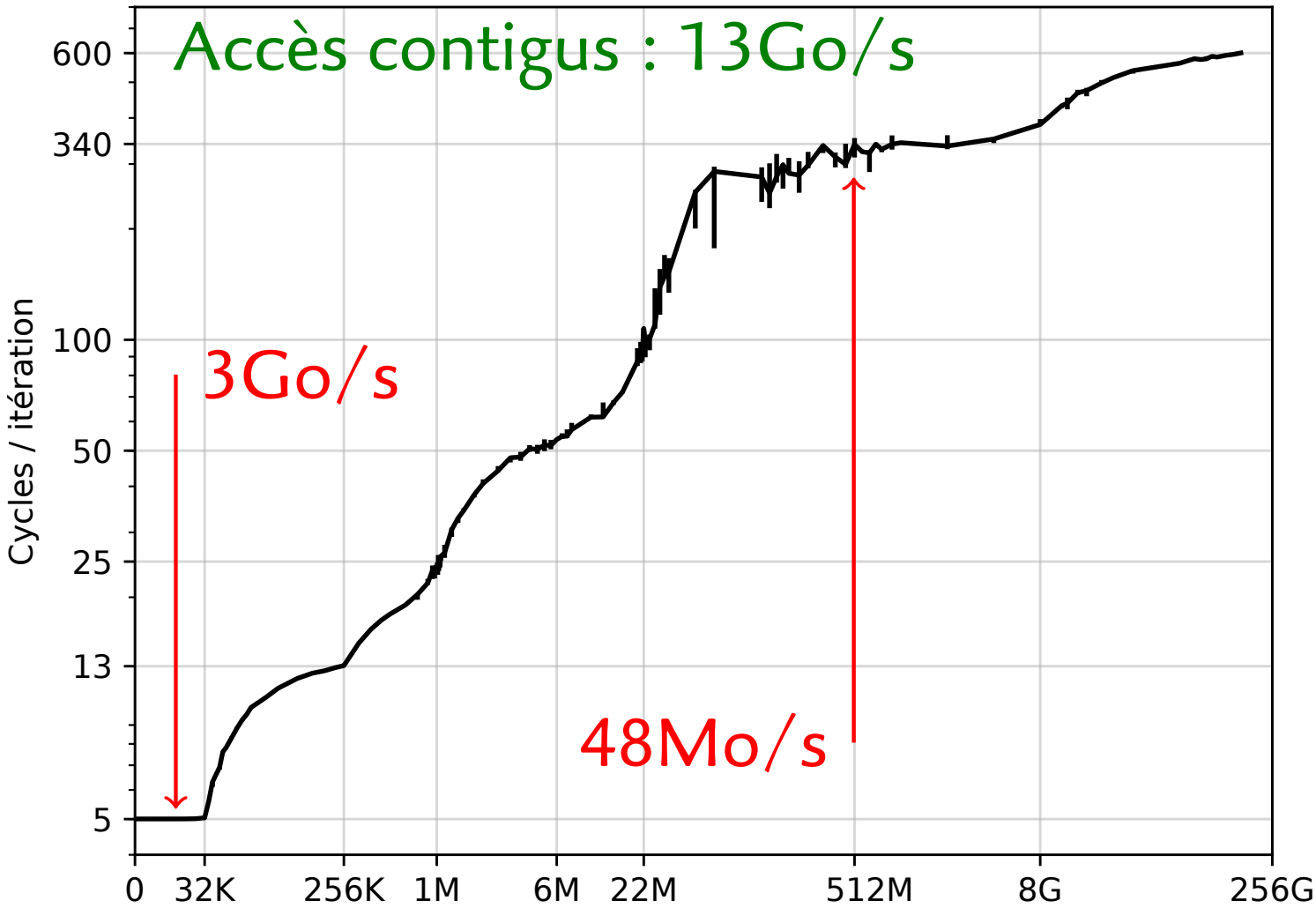
```
for (int i=0, x=0; i < 1000000000; i++) x = T[x];
```



Effet de la hiérarchie mémoire

$T =$ tableau de N entiers aléatoires dans $[0; N[$.

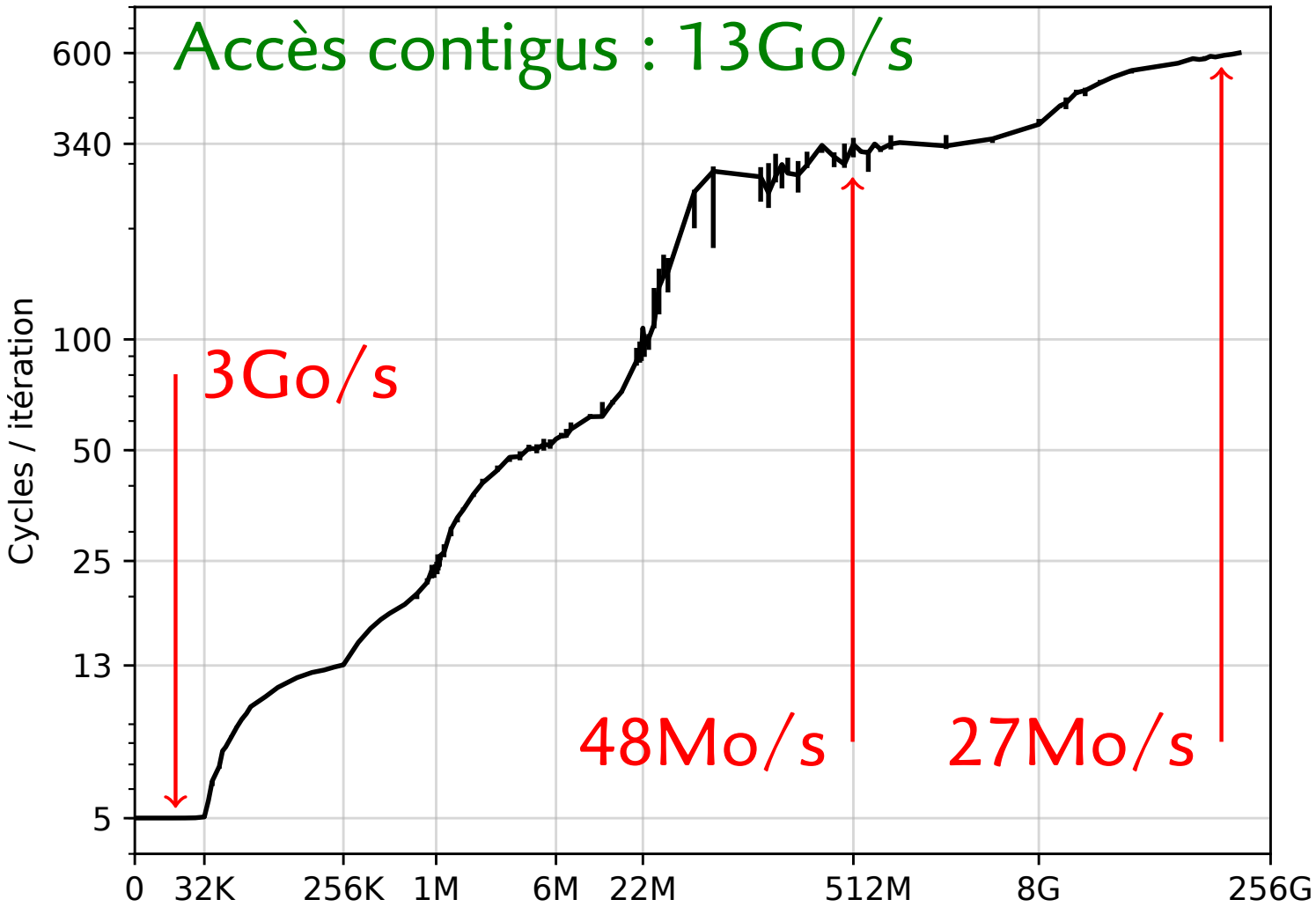
```
for (int i=0, x=0; i < 1000000000; i++) x = T[x];
```



Effet de la hiérarchie mémoire

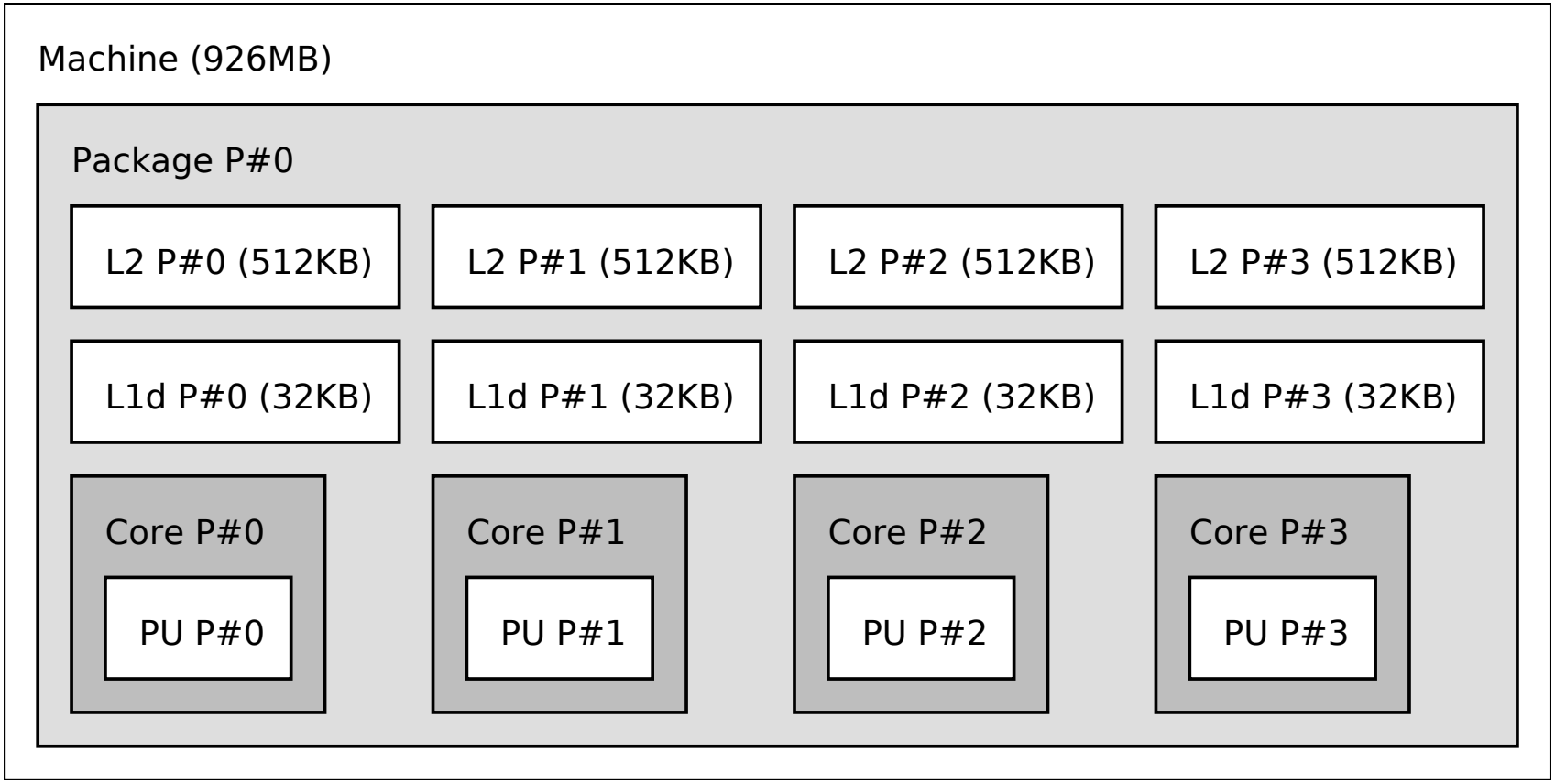
$T =$ tableau de N entiers aléatoires dans $[0; N[$.

```
for (int i=0, x=0; i < 1000000000; i++) x = T[x];
```



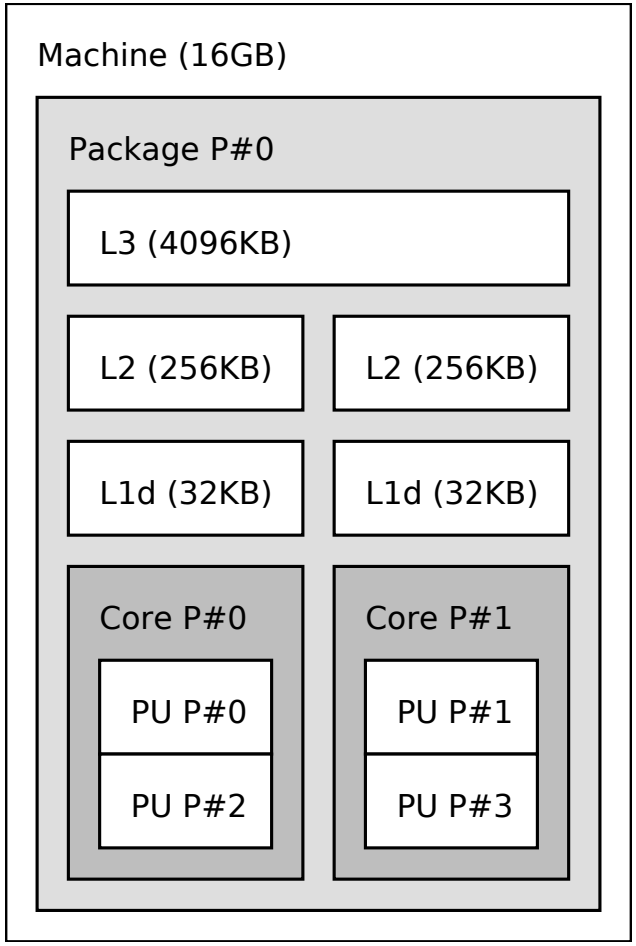
Caches : grande variabilité

Raspberry Pi 3B+ (ARM Cortex A53)



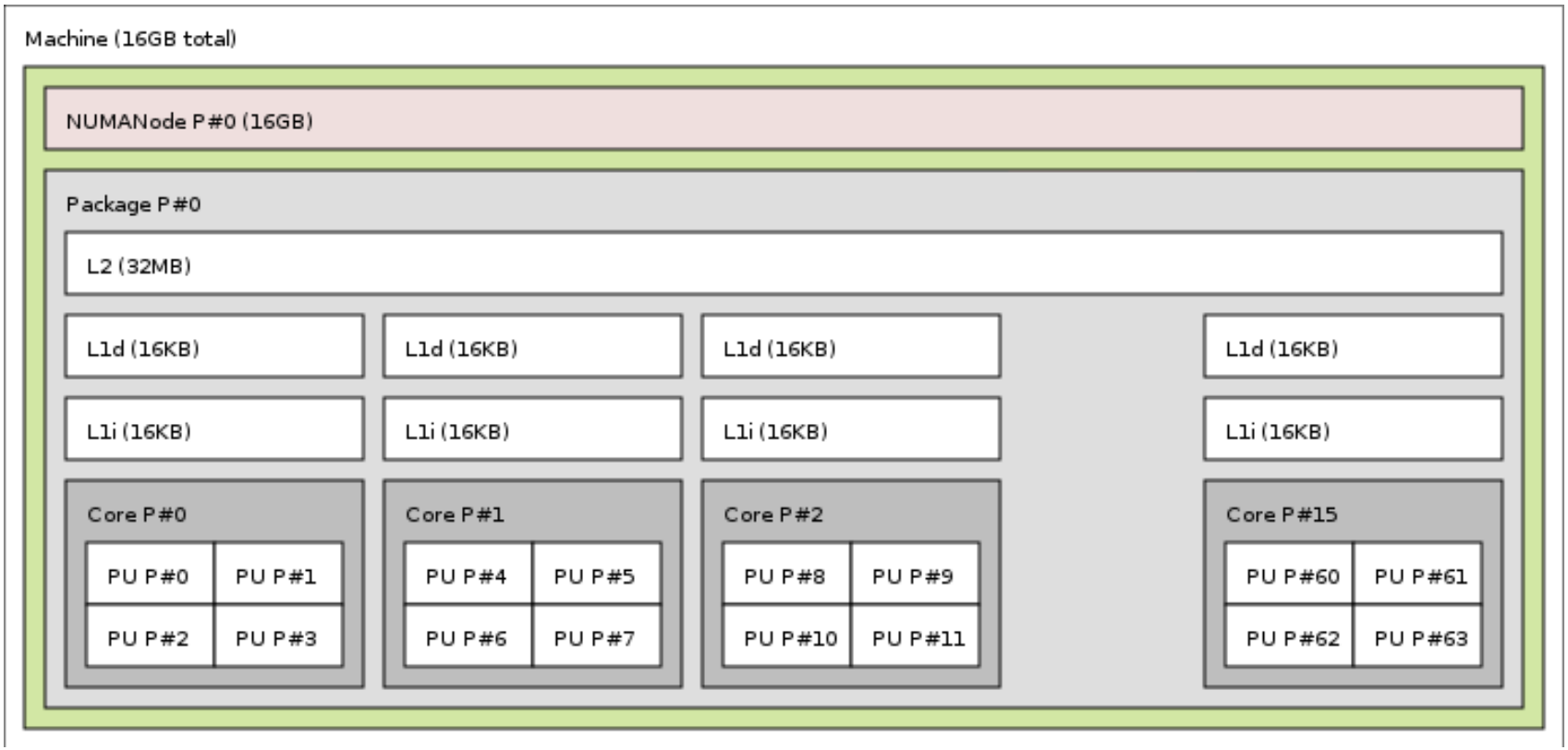
Caches : grande variabilité

Mon laptop (Intel Core i7 6600U)



Caches : grande variabilité

IBM BlueGene/Q (PowerPC A2)

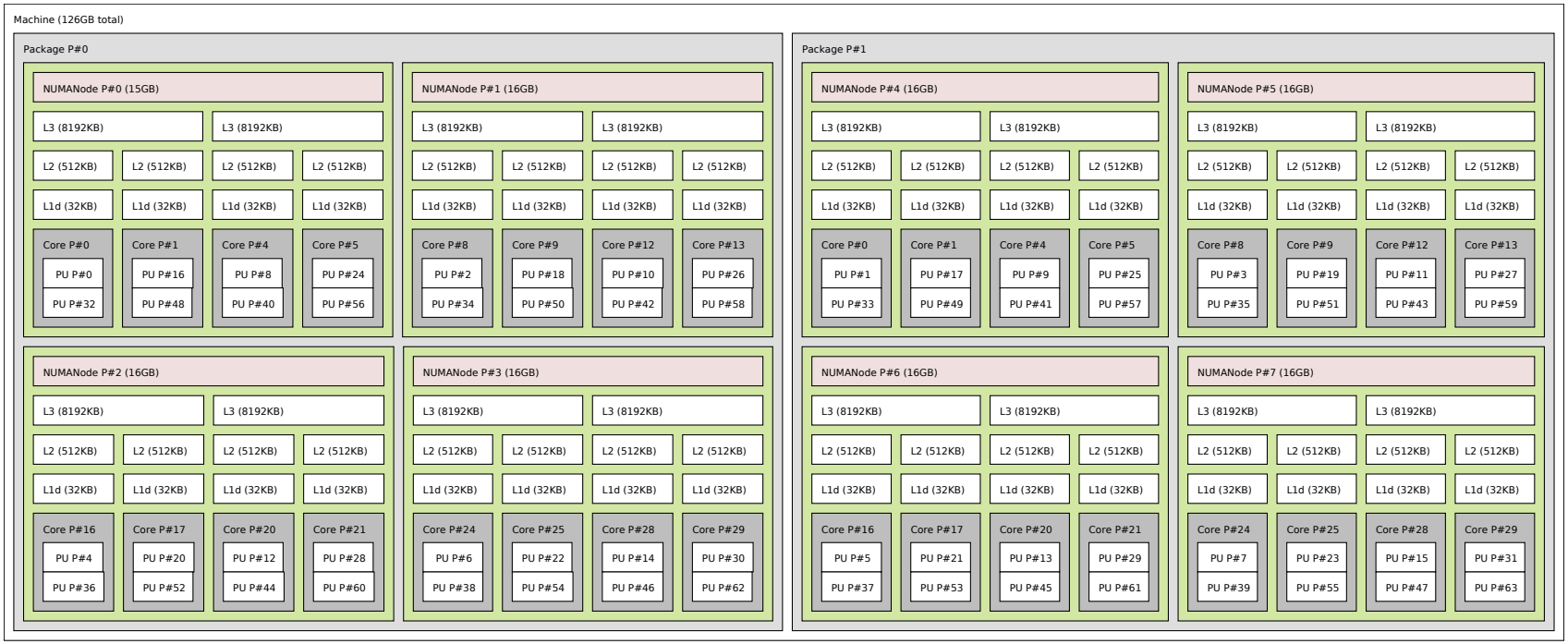


Caches : grande variabilité

Noeud d'un cluster récent (2 × Intel Xeon Gold 6130)

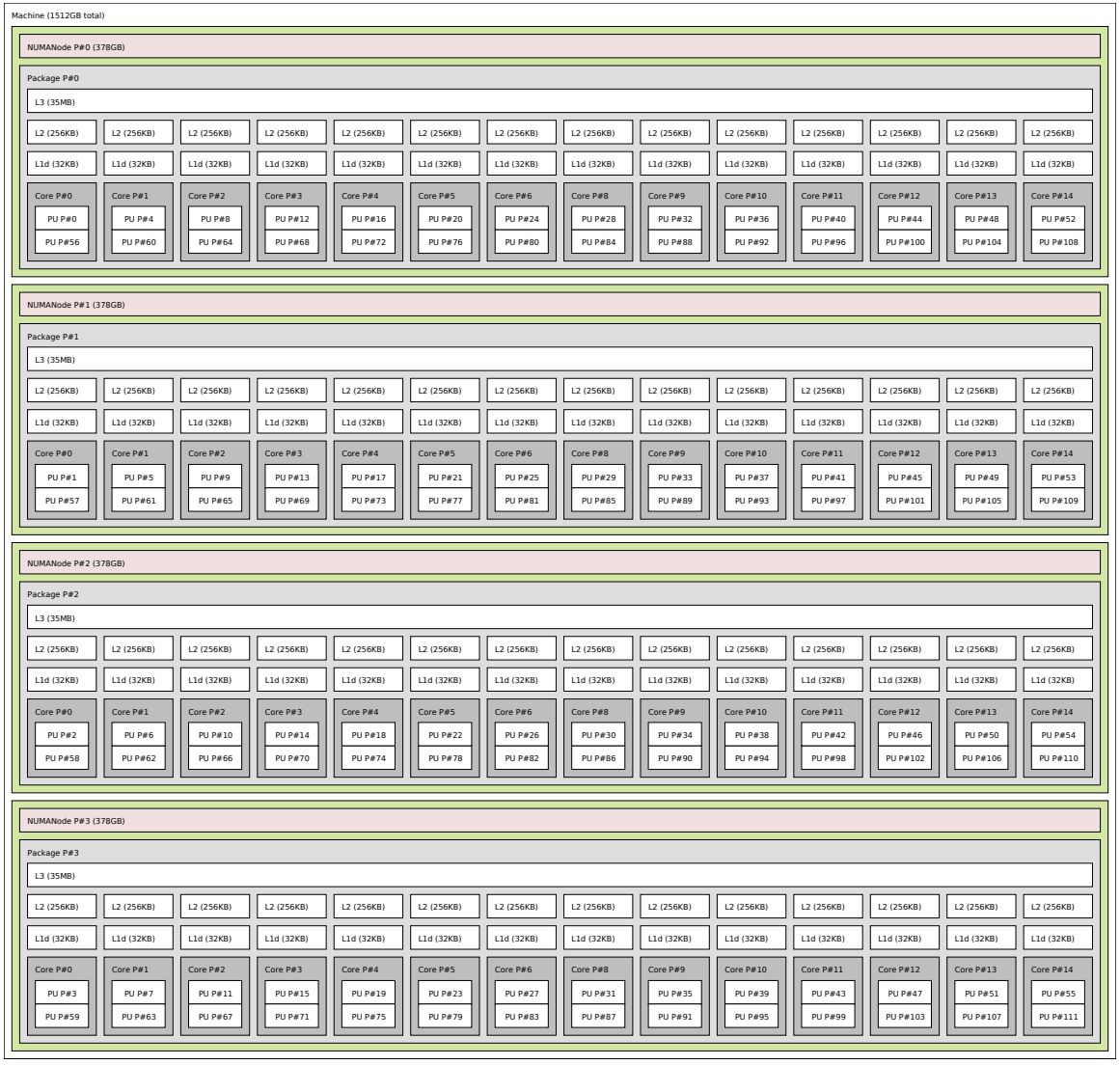
Caches : grande variabilité

Noeud d'un autre cluster récent (2 × AMD EPYC 7301)



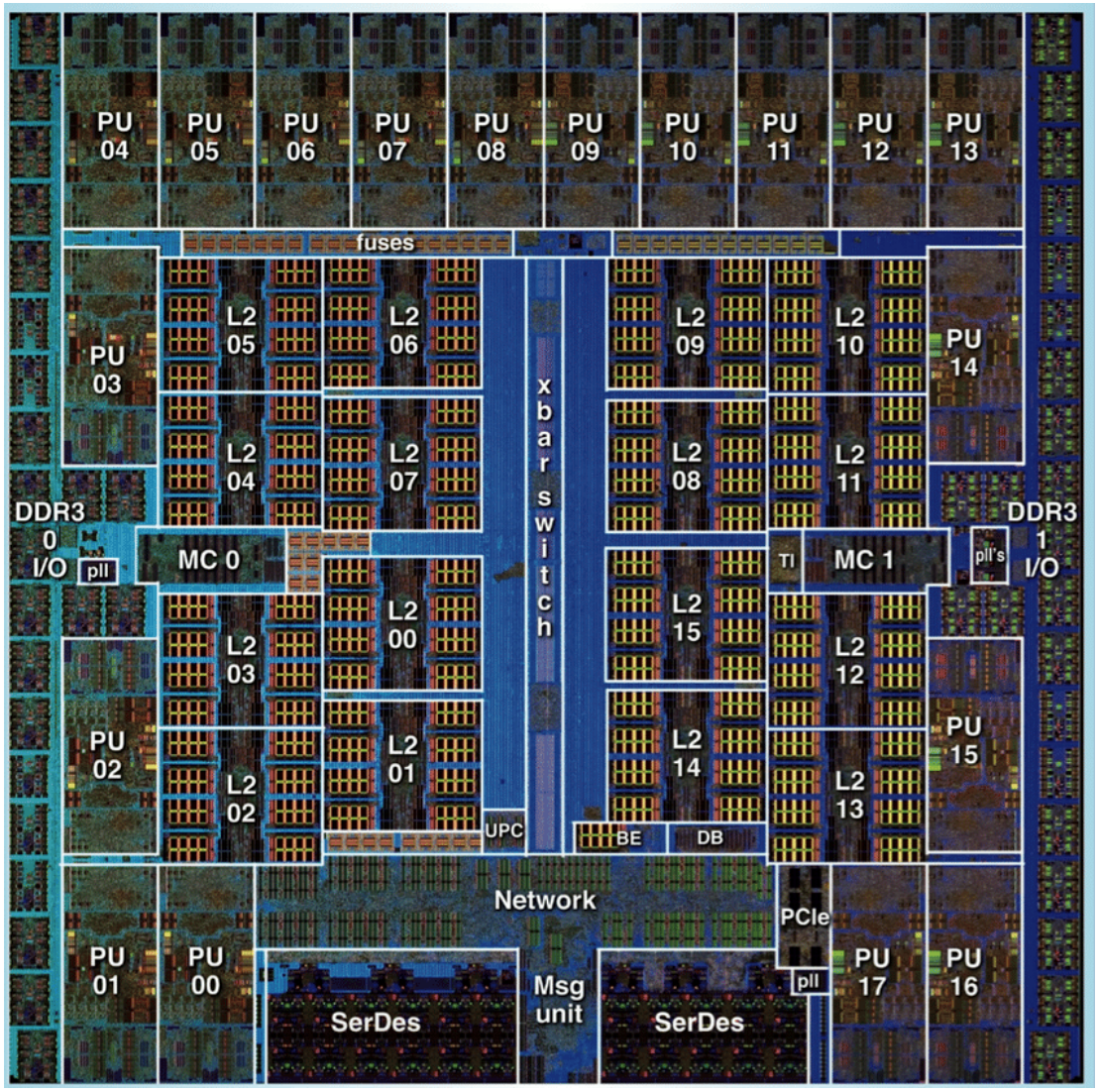
Caches : grande variabilité

Fat Node (4 × Intel Xeon E7-4850 v3) + 1.5To de RAM



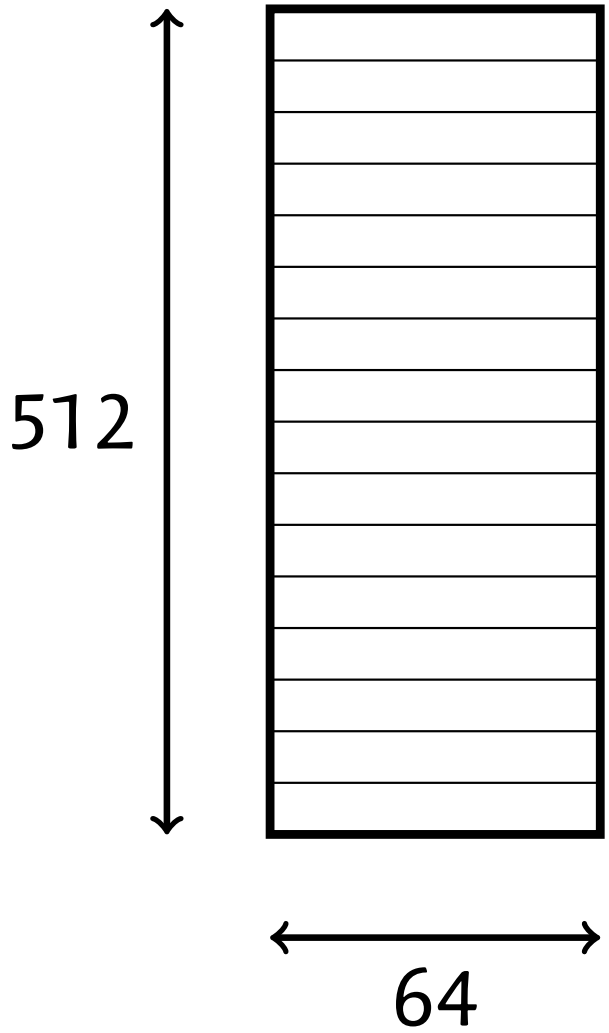
Caches : ça prend de la place !

PowerPC A2



Organisation d'un cache

Cache L1 typique

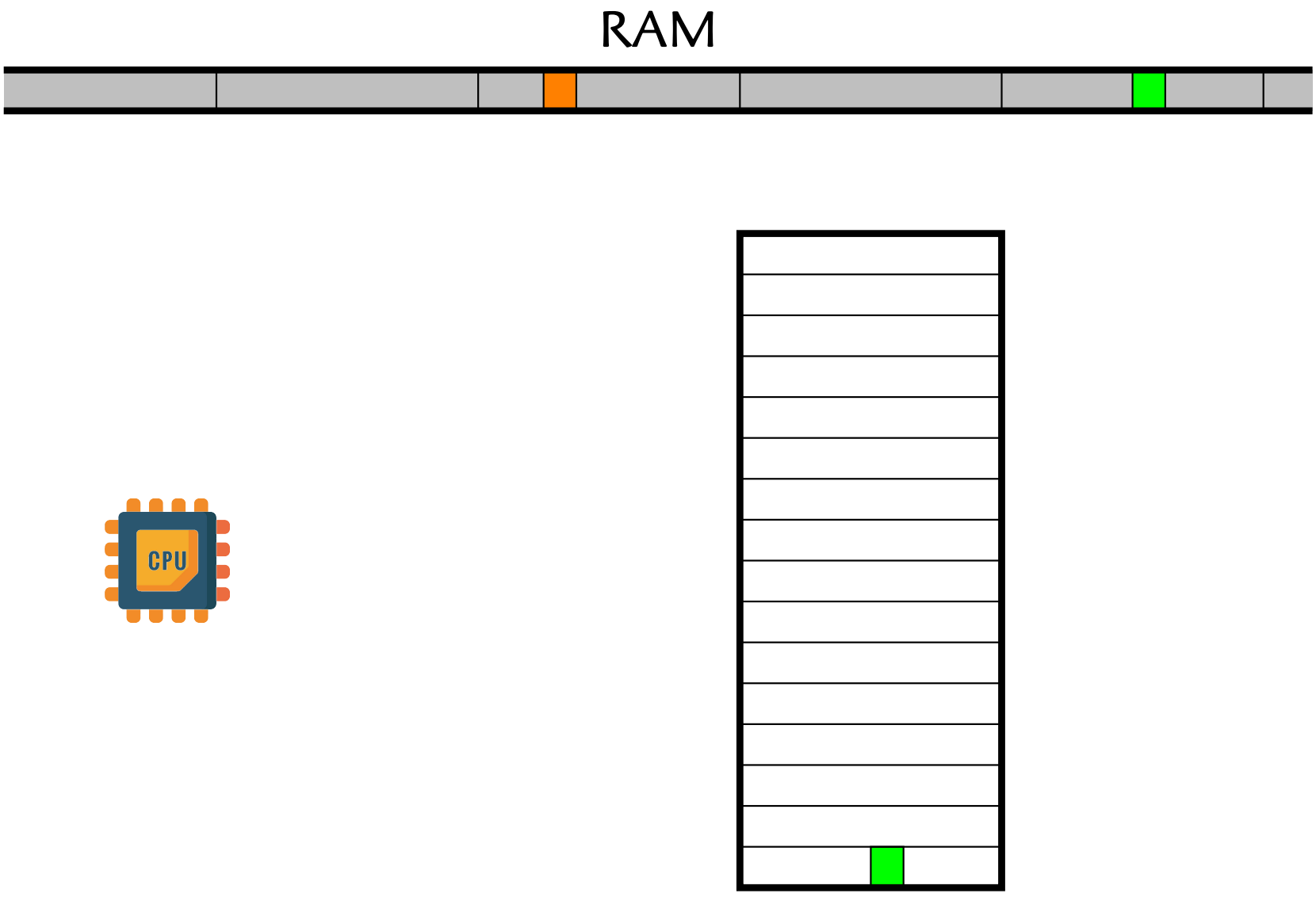


Ligne de cache

= 64 octets alignés

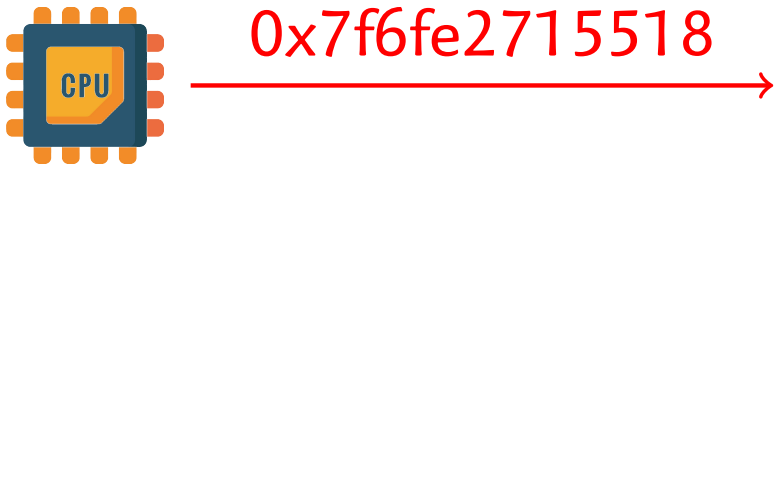
(l'adresse du 1er est un multiple de 64)

Fautes de cache



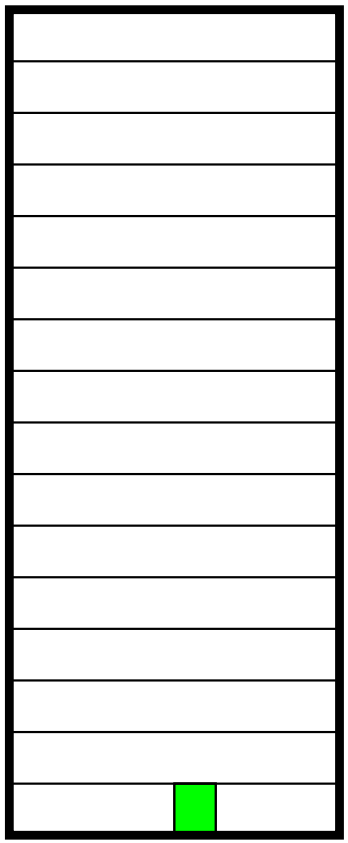
Fautes de cache


```
x = A[i];
```



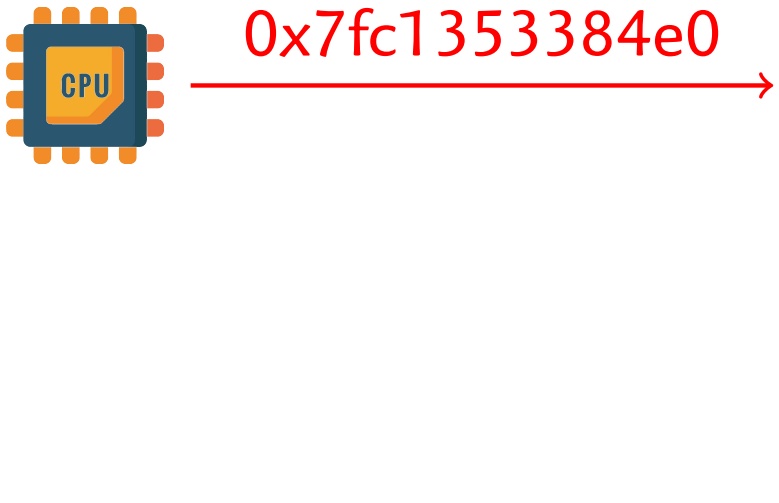
Fautes de cache


```
x = A[i];
```

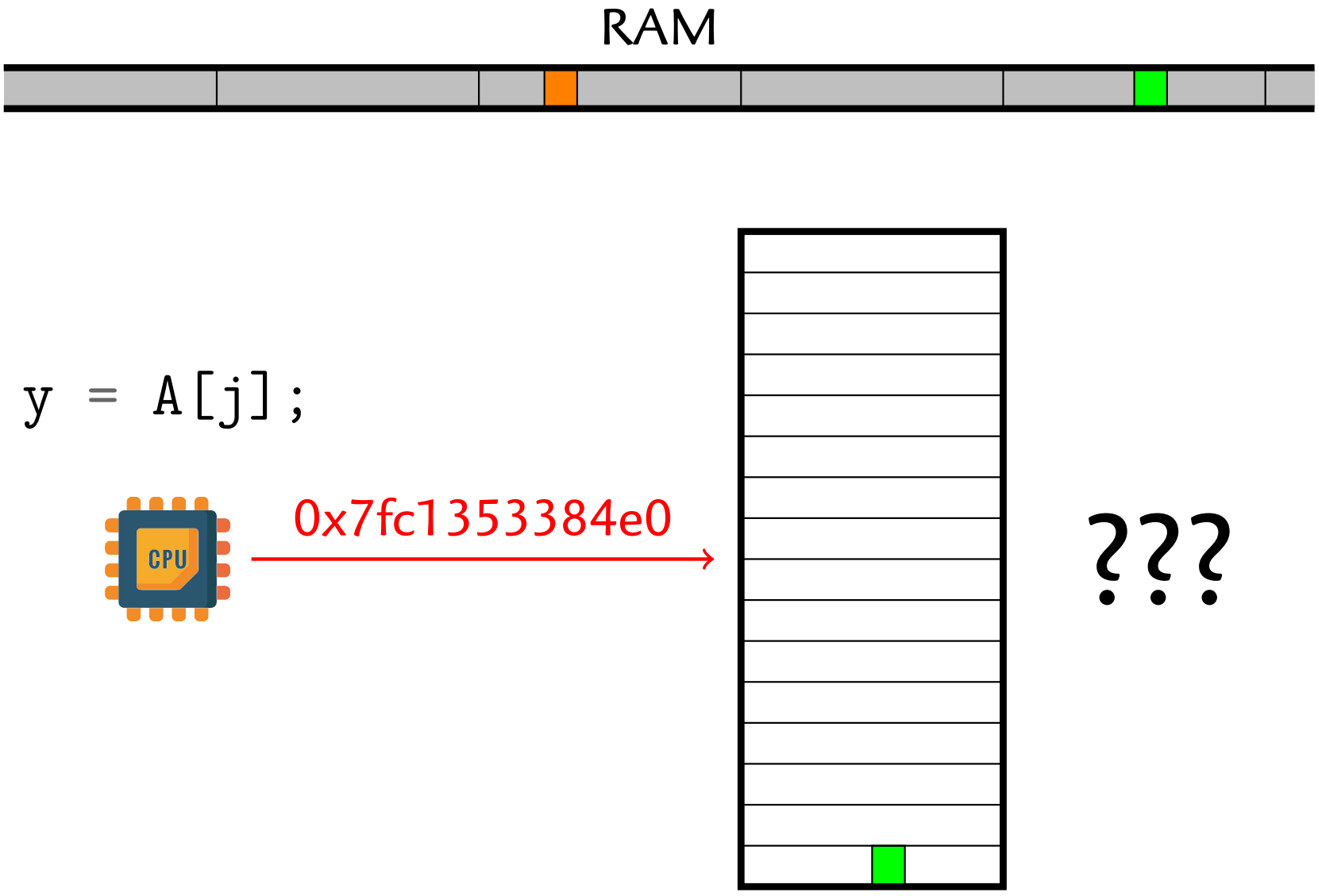


Fautes de cache

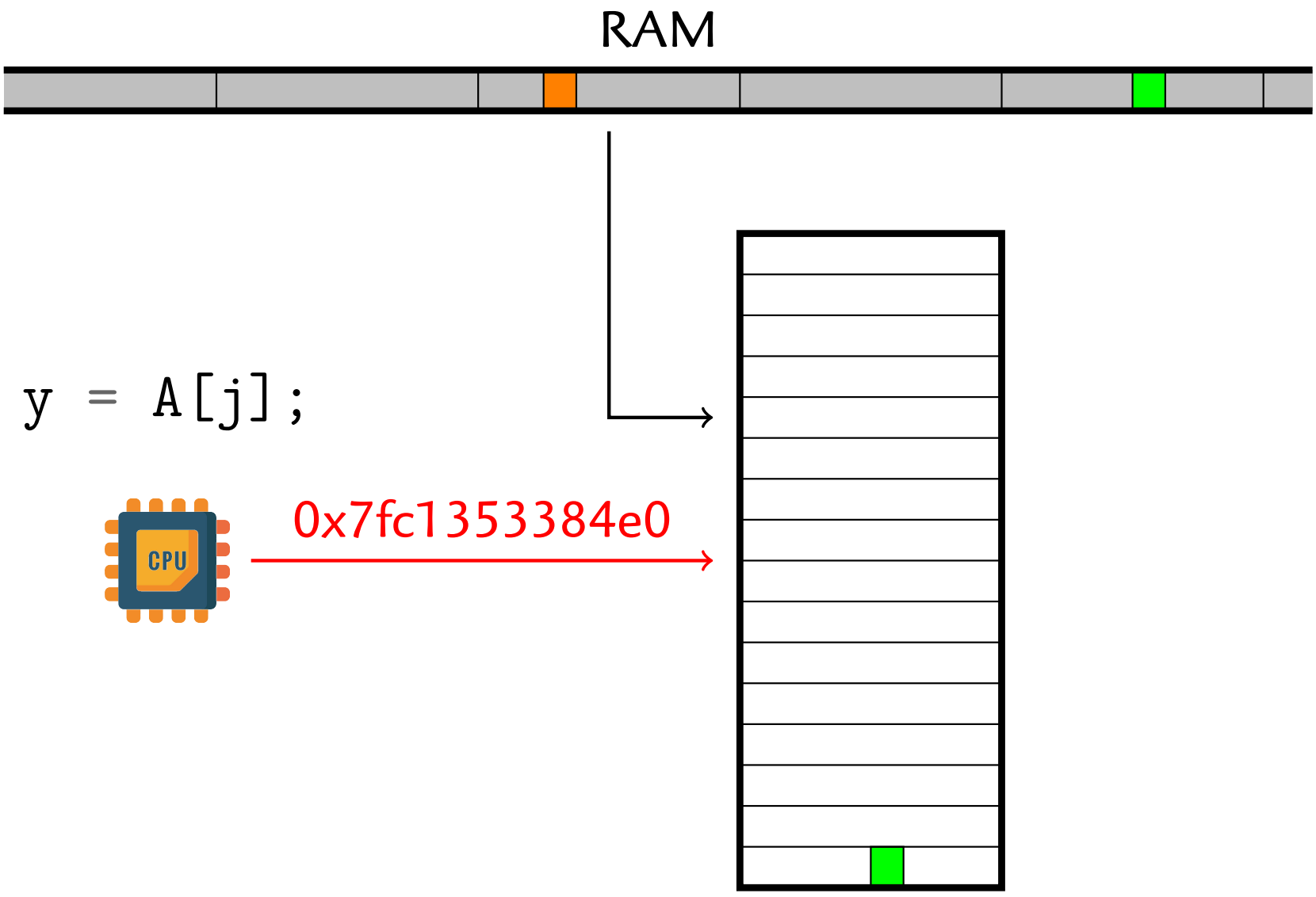

```
y = A[j];
```



Fautes de cache

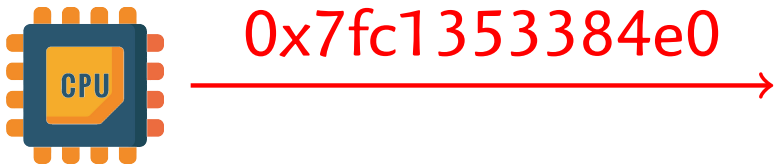


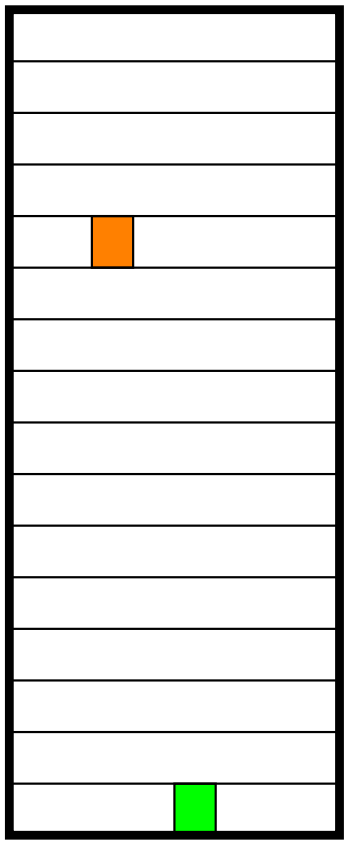
Fautes de cache



Fautes de cache

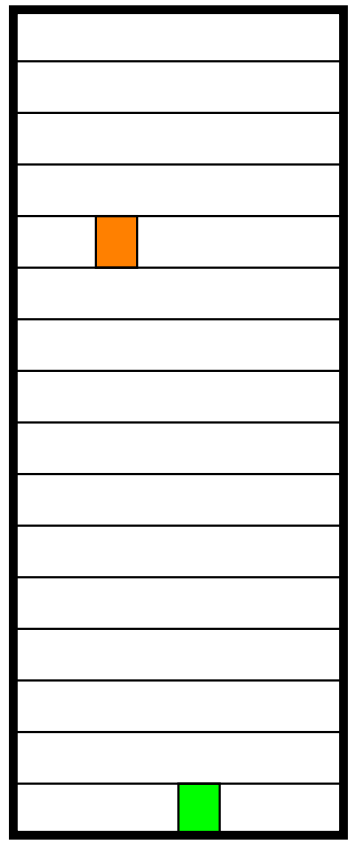

```
y = A[j];
```





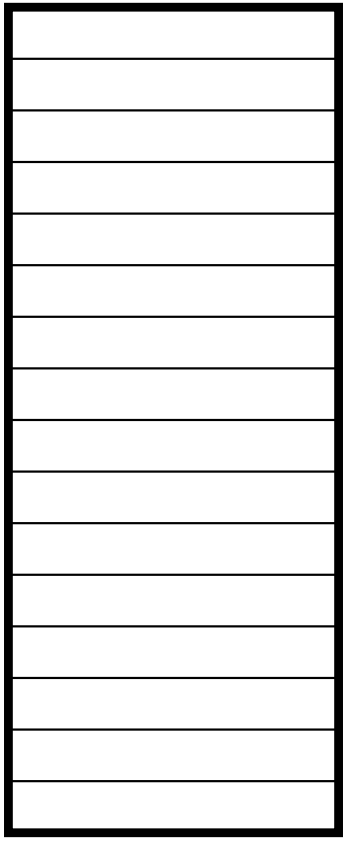
Fautes de cache


```
y = A[j];
```



Fautes de cache (suite)

111111000100111000101010101000011000

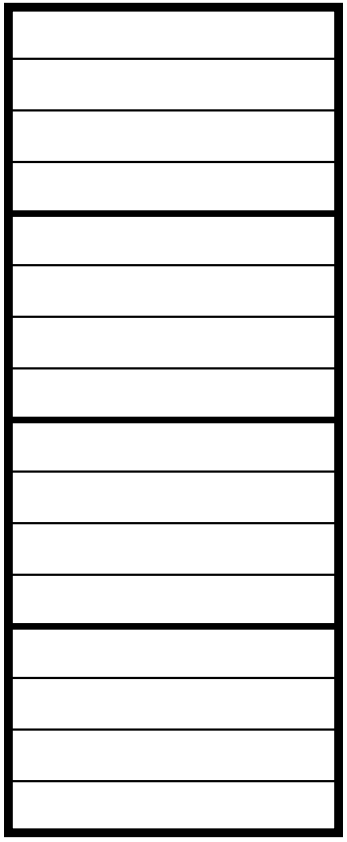


En cas de faute :

- Quelle ligne évincer ?
- Et les écritures ?
- Quels emplacements pour une ligne donnée ?

Fautes de cache (suite)

111111000100111000101010101000011000



En cas de faute :

- Quelle ligne évincer ?
- Et les écritures ?
- Quels emplacements pour une ligne donnée ?

Petits exemples

Recopie de tableau 2D

```
/* Mauvais */
for (int i = 0; i < N; i++)
    for (int j = 0; j <N; j++)
        dst[j][i] = src[j][i];

/* Bon */
for (int i = 0; i < N; i++)
    for (int j = 0; j < N; j++)
        dst[i][j] = src[i][j];
```

Petits exemples

Recopie de tableau 2D

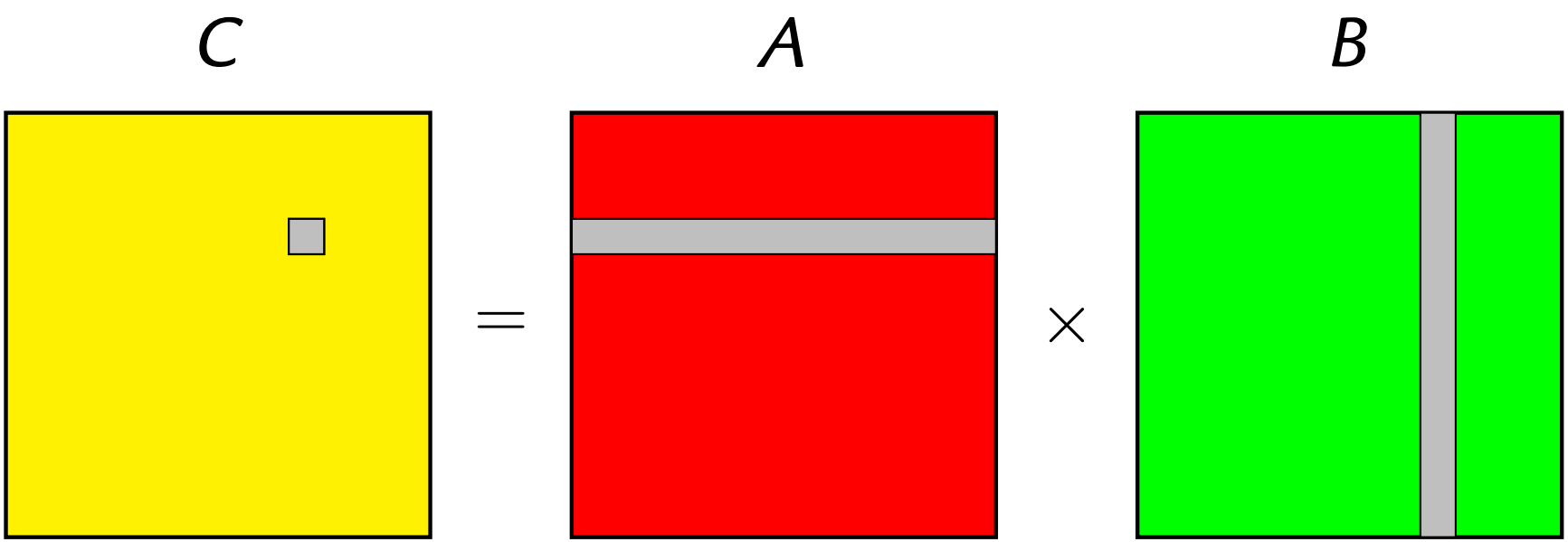
```
/* Mauvais */
for (int i = 0; i < N; i++)
    for (int j = 0; j <N; j++)
        dst[j][i] = src[j][i];

/* Bon */
for (int i = 0; i < N; i++)
    for (int j = 0; j < N; j++)
        dst[i][j] = src[i][j];
```

Petits exemples

Produit de matrice naïf

```
for (int i = 0; i < N; i++)
  for (int j = 0; j < N; j++)
    for (int k = 0; k < N; k++)
      C[i * N + j] += A[i * N + k] * B[k * N + j];
```



Petits exemples

Produit de matrice naïf

```
for (int i = 0; i < N; i++)
  for (int k = 0; k < N; k++)
    for (int j = 0; j < N; j++)
      C[i * N + j] += A[i * N + k] * B[k * N + j];
```

Astuce #1

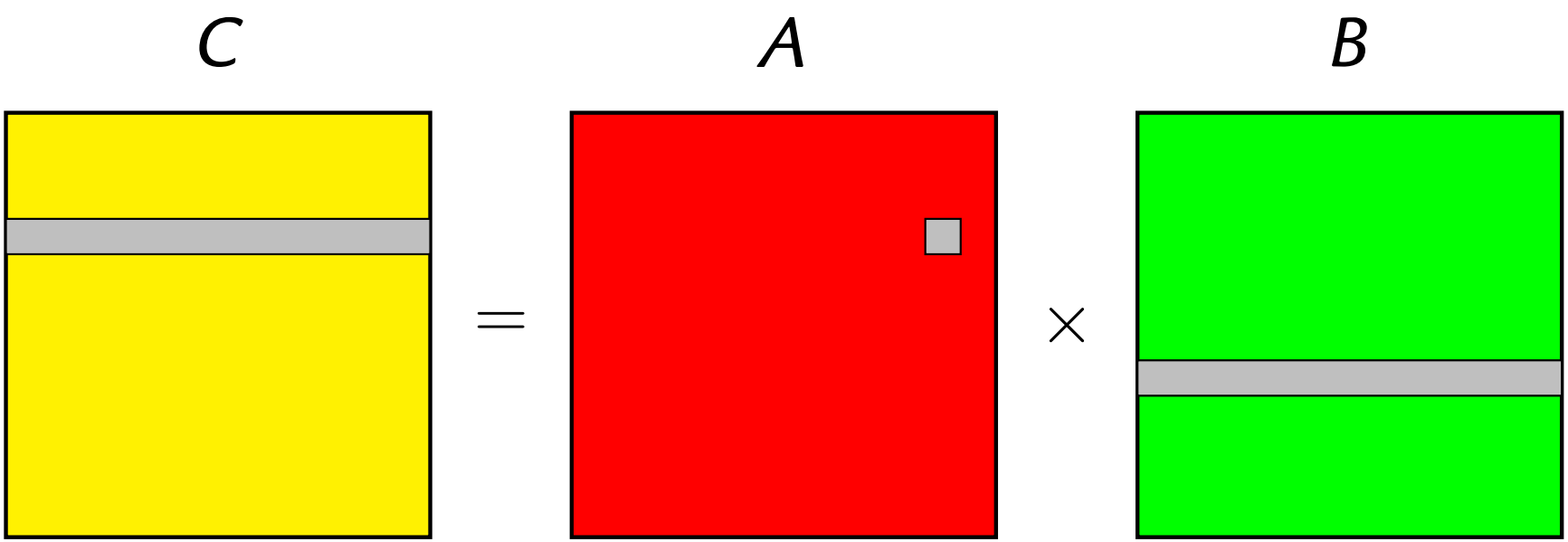
- Permuter les boucles sur j et k.

$\Rightarrow$ Accès contigüs (localité spatiale)

Petits exemples

Produit de matrice naïf

```
for (int i = 0; i < N; i++)
  for (int k = 0; k < N; k++)
    for (int j = 0; j < N; j++)
      C[i * N + j] += A[i * N + k] * B[k * N + j];
```



Petits exemples

Produit de matrice naïf

```
transpose(B);
for (int i = 0; i < N; i++)
  for (int j = 0; j < N; j++)
    for (int k = 0; k < N; k++)
      C[i * N + j] += A[i * N + k] * B[j * N + kj];
```

Astuce #2

- Pré-transposer B.
- Bonus : possibilités de vectorisation.

Petits exemples

Produit de matrice naïf

```
transpose(B);
for (int i = 0; i < N; i++)
  for (int j = 0; j < N; j++)
    for (int k = 0; k < N; k++)
      C[i * N + j] += A[i * N + k] * B[j * N + kj];
```

Bilan

- Petites matrices : pas rentable
  - surcoût trop élevé.
- Grandes matrices : gain
  - $N = 3200 : 177s \rightsquigarrow 63s$.

Petits exemples

Produit de matrice naïf

Astuce #2

- Produit par blocs.
- Défaut : récursif au lieu d'itératif.
- Avantage : les petits blocs tiennent en cache.