Les caches

2MIC
Architecture matérielle

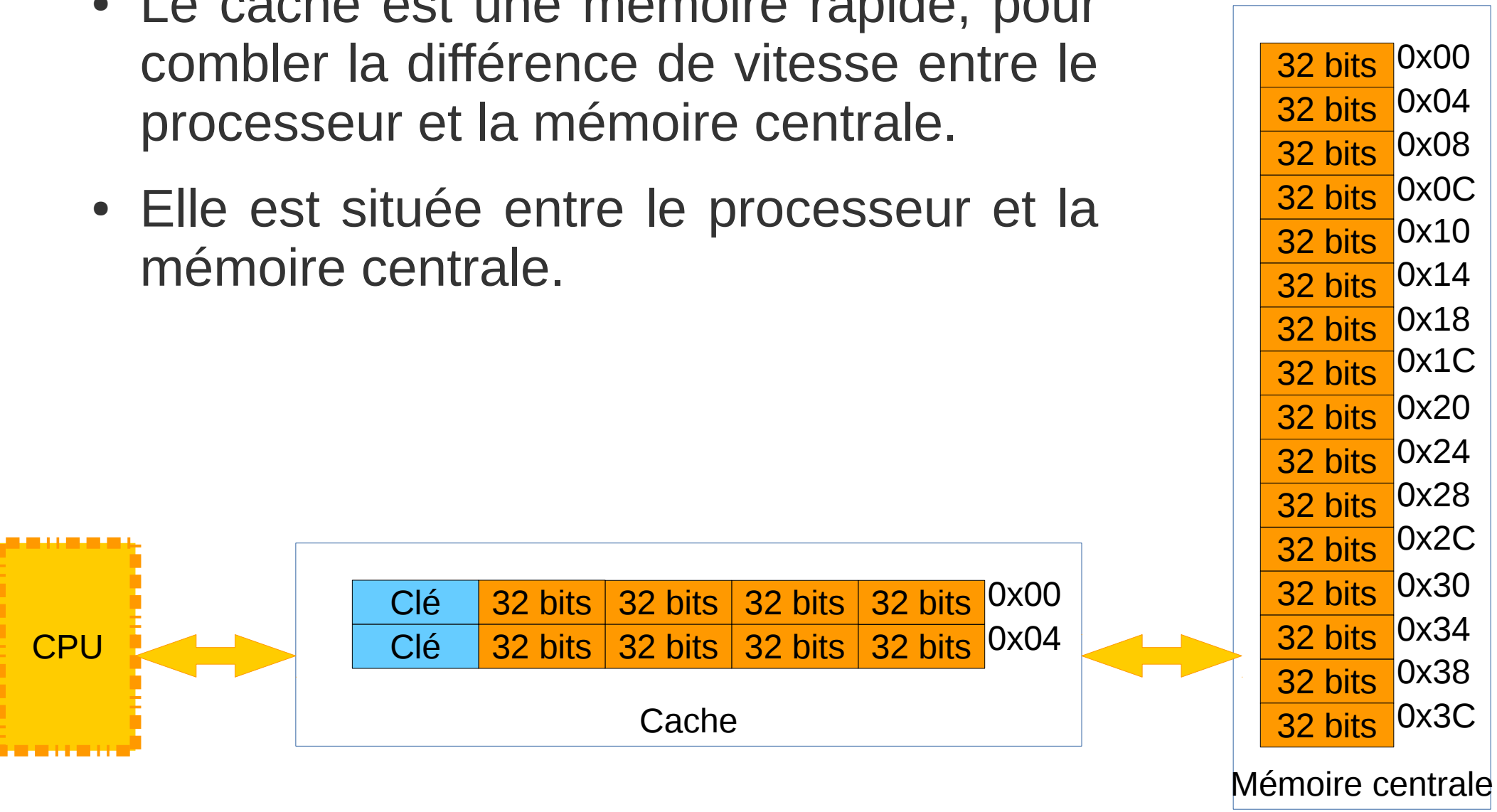
Vincent MIGLIORE

Principes fondamentaux de l'informatique

- **Principe de localité :**
  - Il y a une forte probabilité qu'une donnée dont l'adresse est proche d'une donnée récemment accédée le soit à son tour.
  - Exemple : Les instructions qui sont situées les unes à la suite des autres en mémoire centrale.
- **Principe de temporalité :**
  - Il y a une forte probabilité qu'une donnée récemment utilisée le soit de nouveau.

Le principe du cache

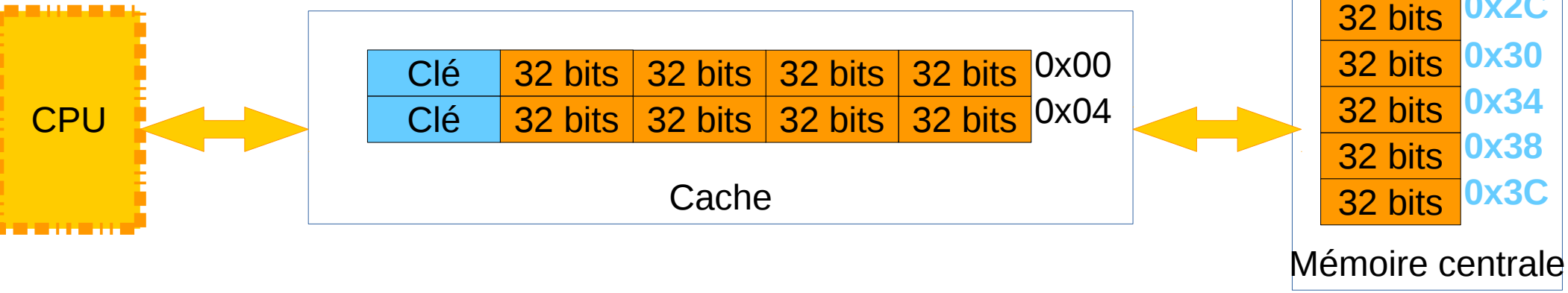
- Le cache est une mémoire rapide, pour combler la différence de vitesse entre le processeur et la mémoire centrale.
- Elle est située entre le processeur et la mémoire centrale.



Le principe du cache

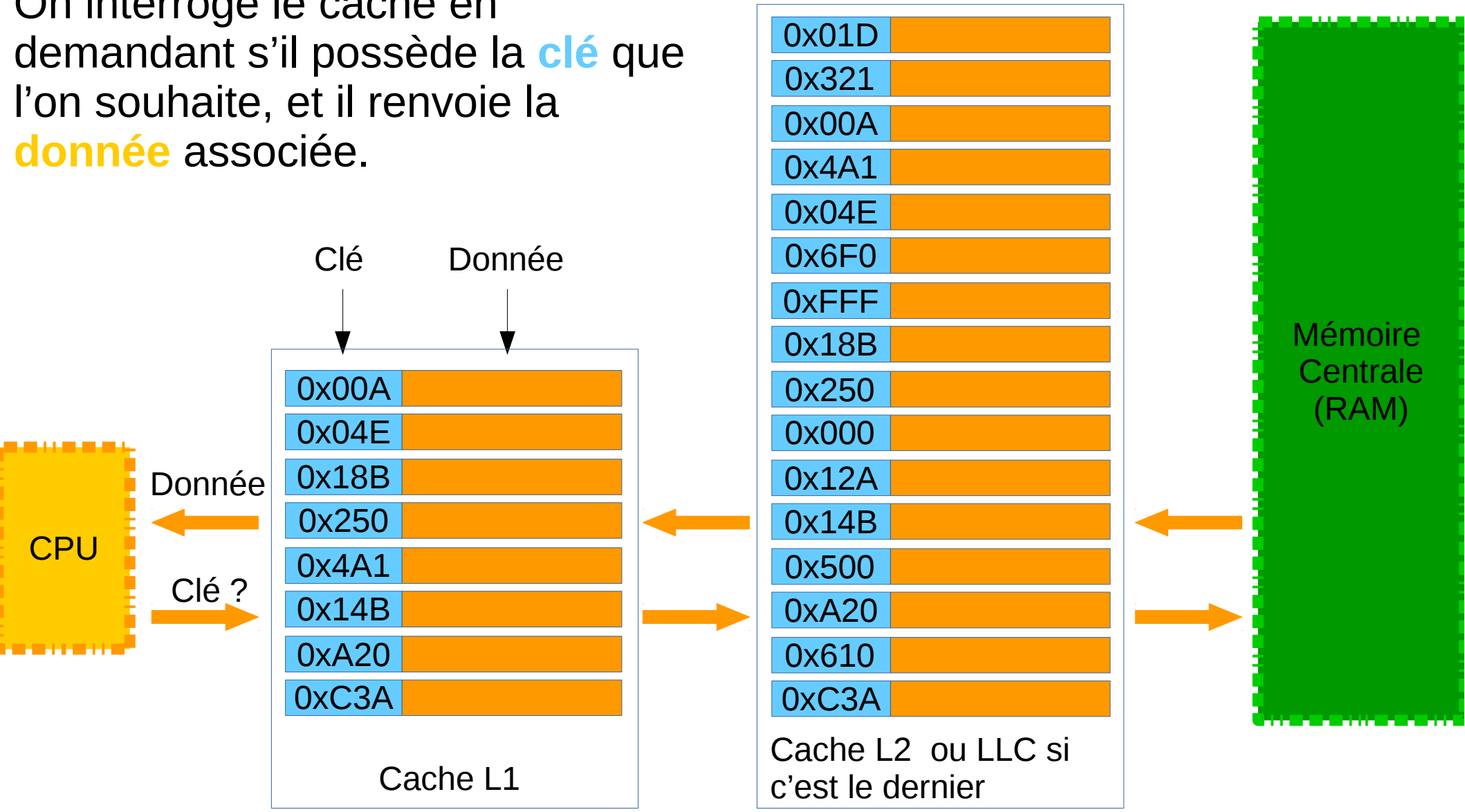
Le cache est une mémoire associative :

- Les informations sont accessibles par leur contenu, et non par une adresse.
- Une ligne de cache est une paire Clé/contenu.
- On demande une clé, en non une adresse.
- Clé = typiquement l'adresse d'une donnée en mémoire centrale.



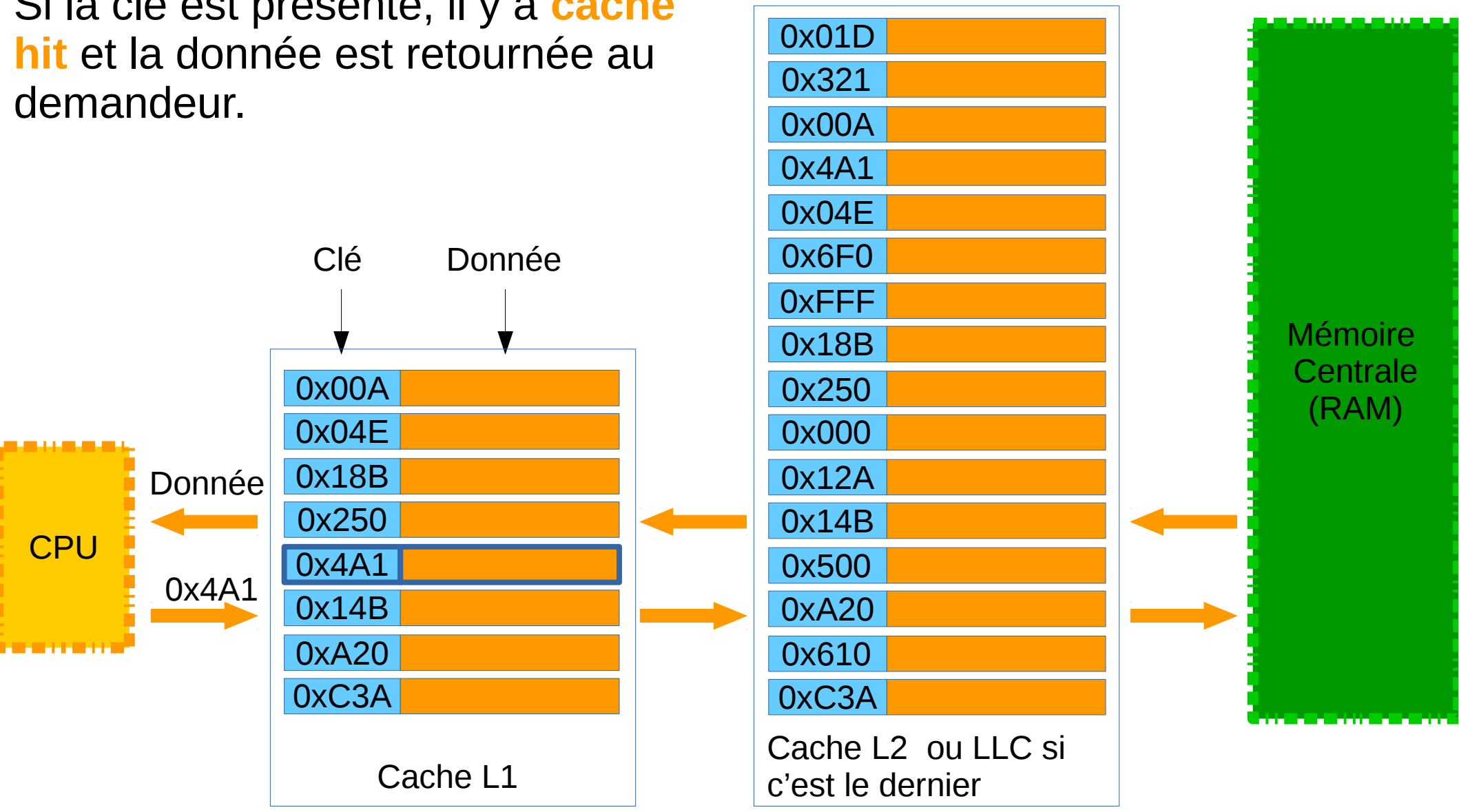
Le principe du cache

On interroge le cache en demandant s'il possède la **clé** que l'on souhaite, et il renvoie la **donnée** associée.



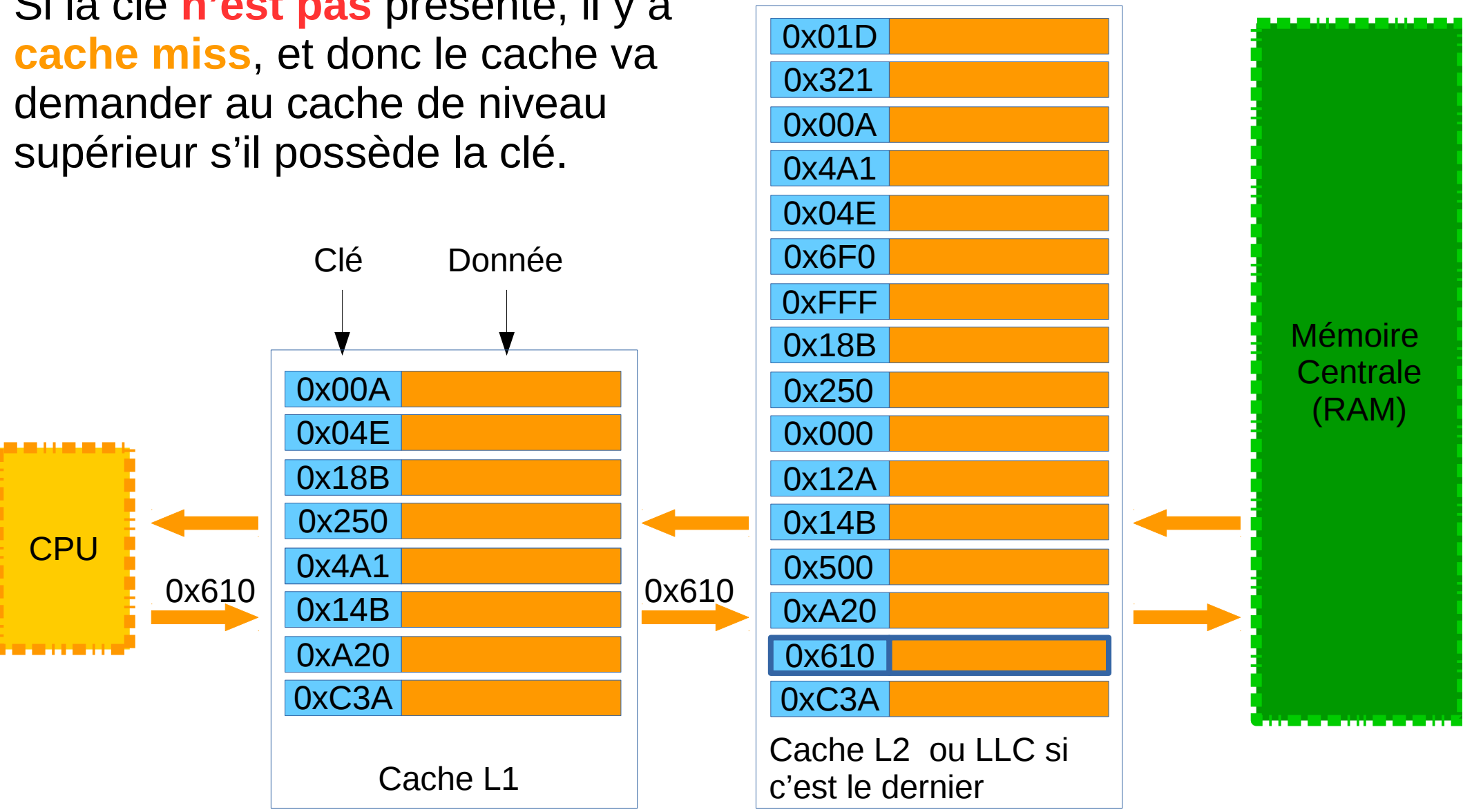
Le principe du cache

Si la clé est présente, il y a **cache hit** et la donnée est retournée au demandeur.



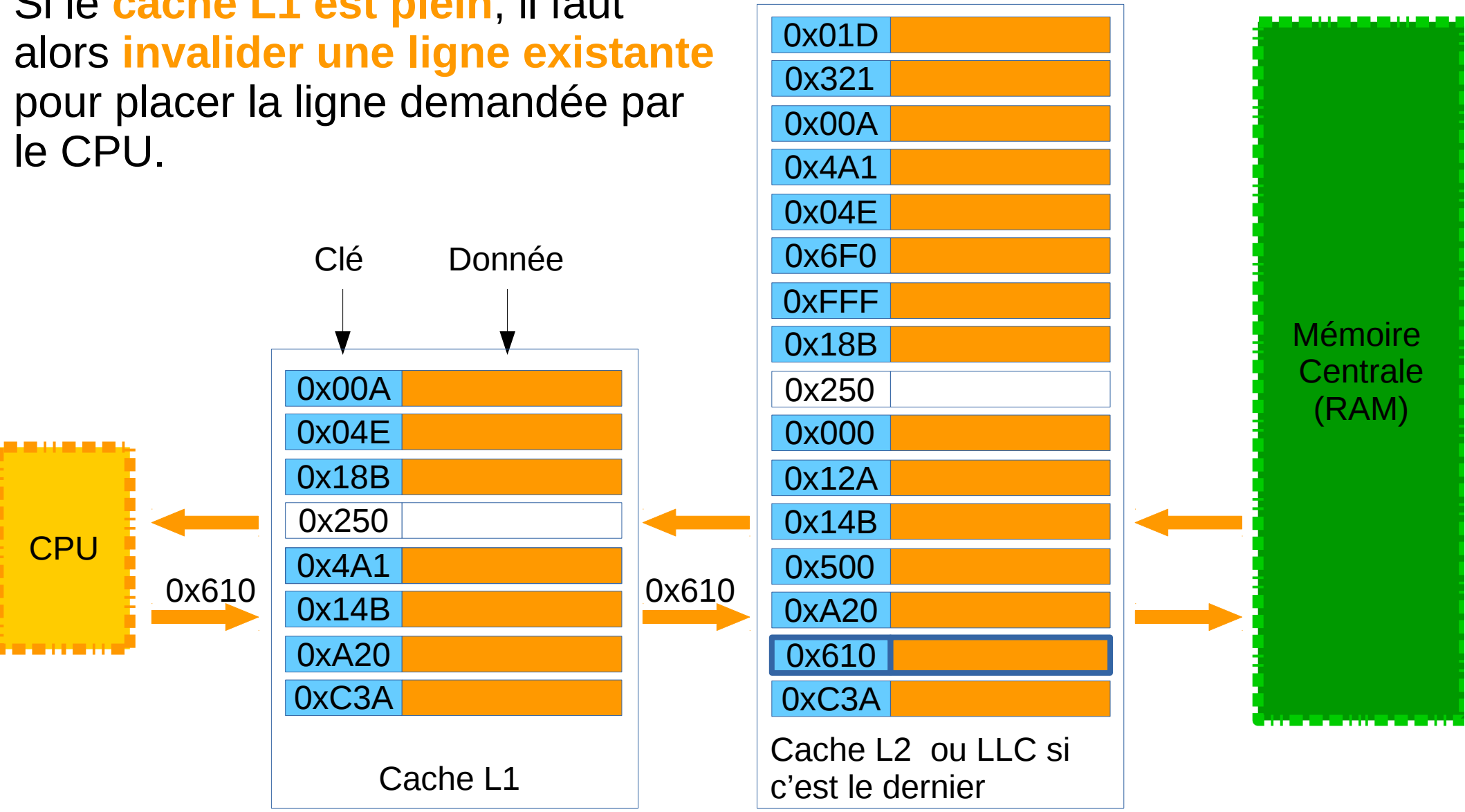
Le principe du cache

Si la clé **n'est pas** présente, il y a **cache miss**, et donc le cache va demander au cache de niveau supérieur s'il possède la clé.



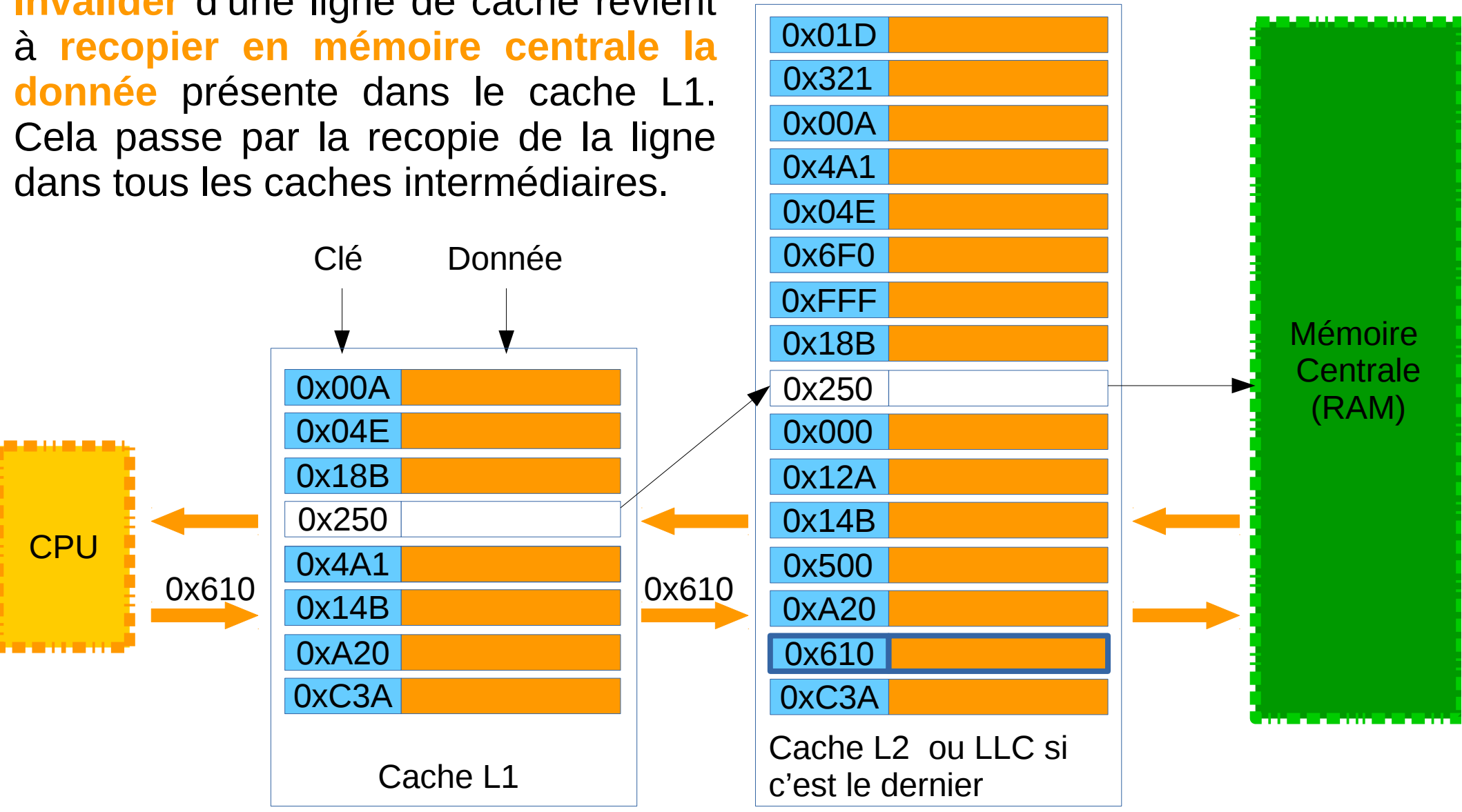
Le principe du cache

Si le **cache L1 est plein**, il faut alors **invalider une ligne existante** pour placer la ligne demandée par le CPU.



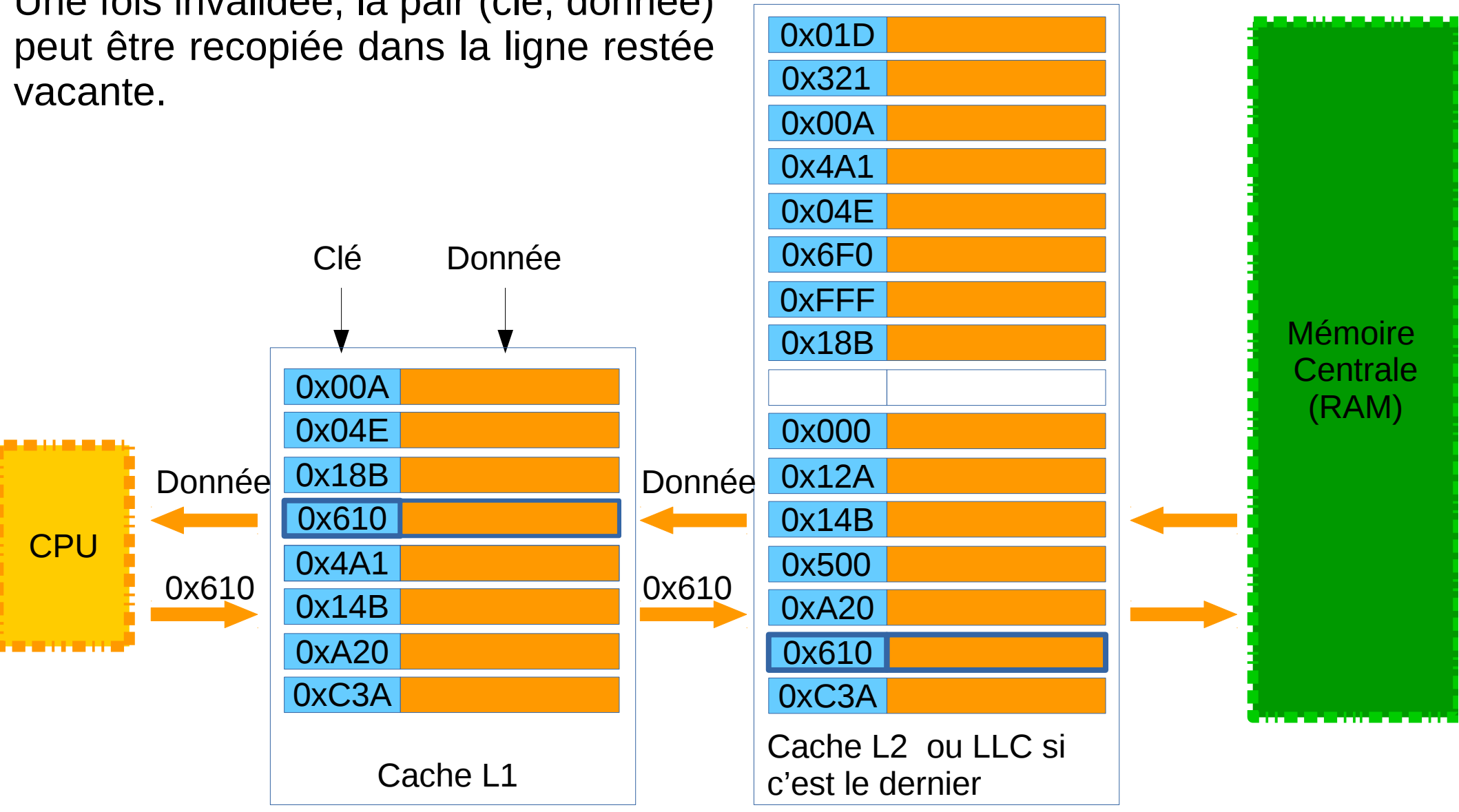
Le principe du cache

Invalider d'une ligne de cache revient à **recopier en mémoire centrale la donnée** présente dans le cache L1. Cela passe par la recopie de la ligne dans tous les caches intermédiaires.



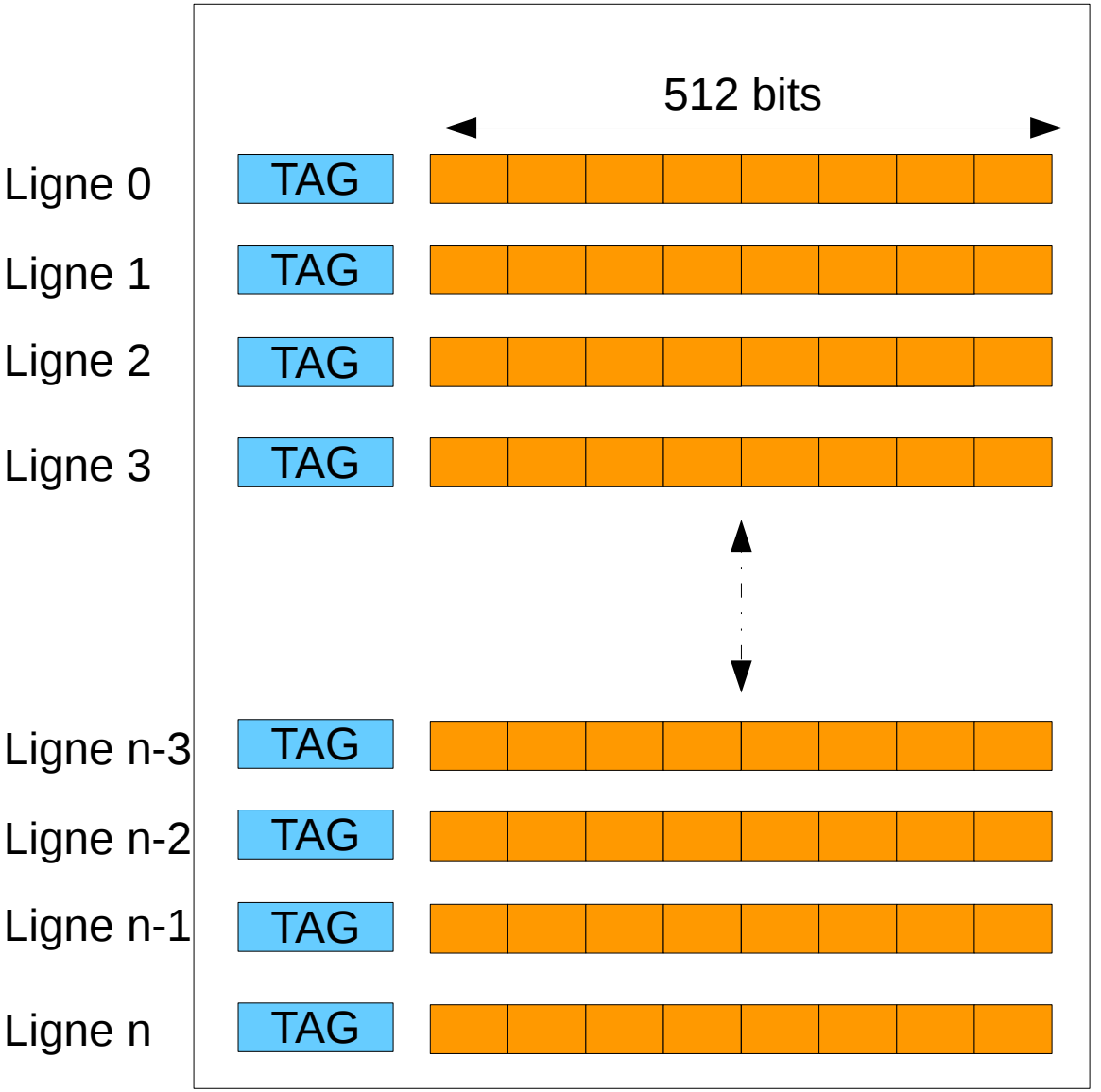
Le principe du cache

Une fois invalidée, la pair (clé, donnée) peut être recopiée dans la ligne restée vacante.



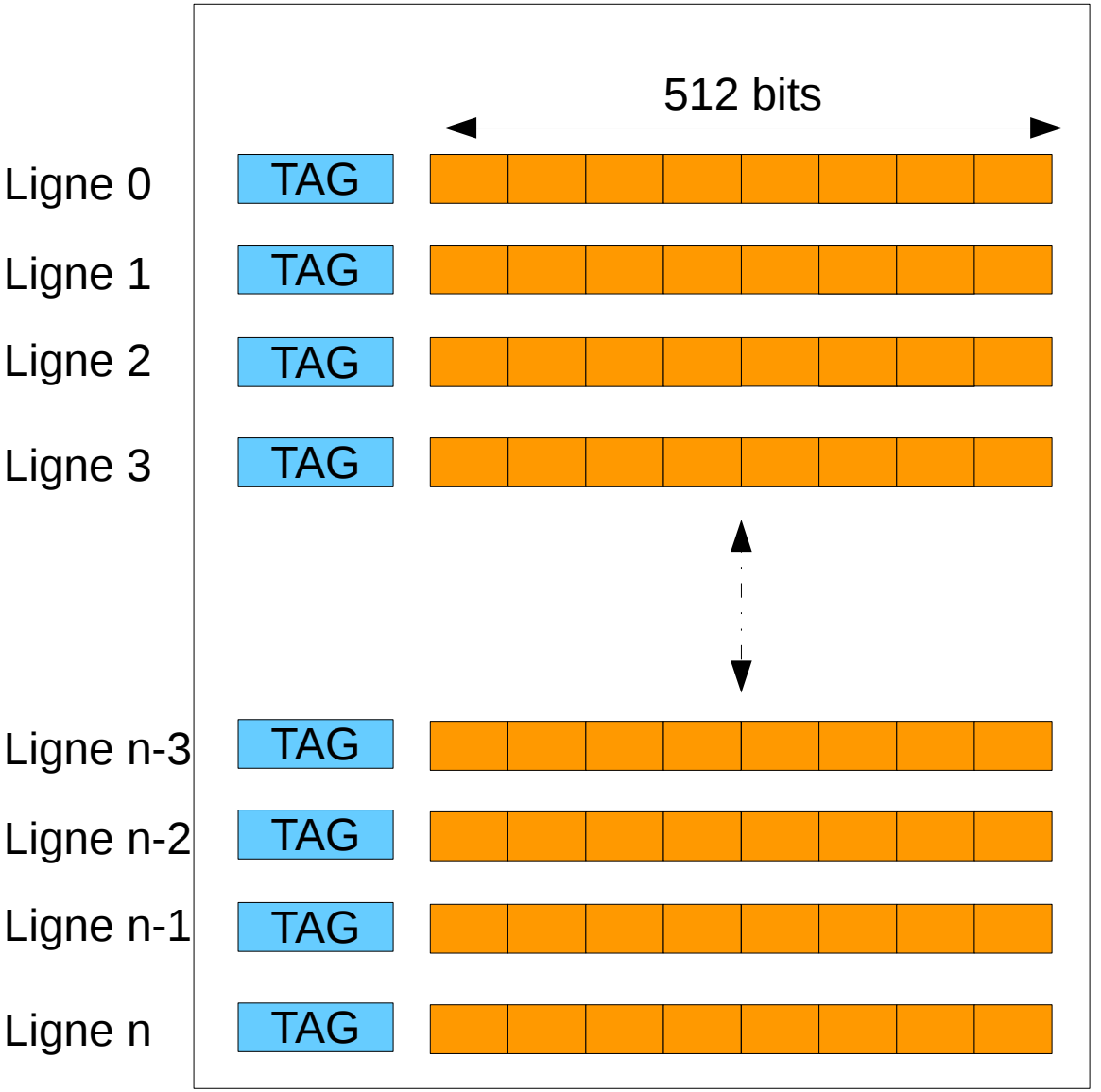
Organisation d’un cache

- Un cache est organisé en lignes, qui possèdent en général 8 données de 32 ou 64 bits.
- Pour une mémoire RAM dont les transferts sont sur 64 bits, il faut donc 8 accès mémoires pour remplir 1 ligne de cache.



Organisation d’un cache

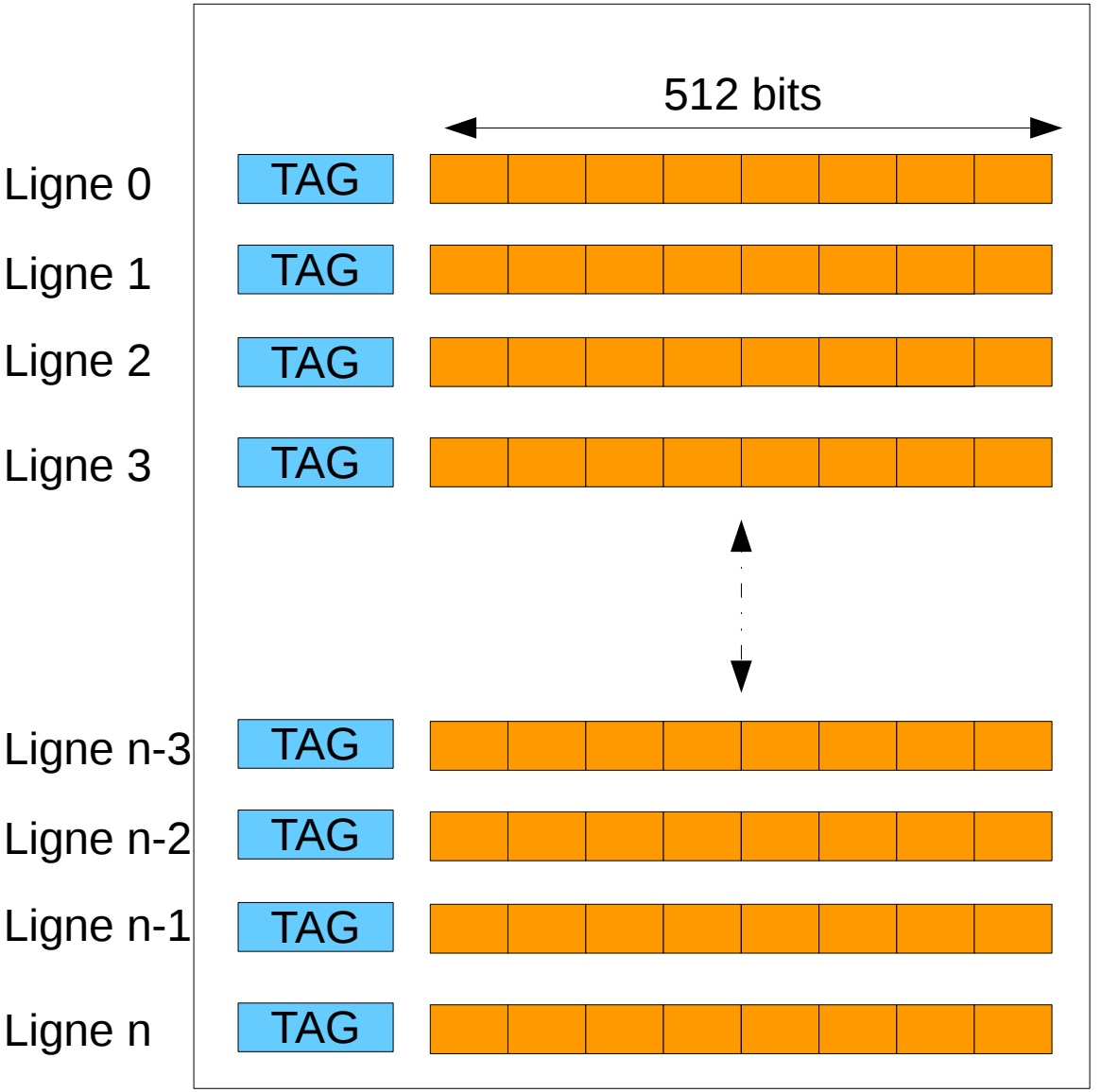
- Si on considère un cache de 16 Ko, combien il y a de lignes de cache ?



Organisation d’un cache

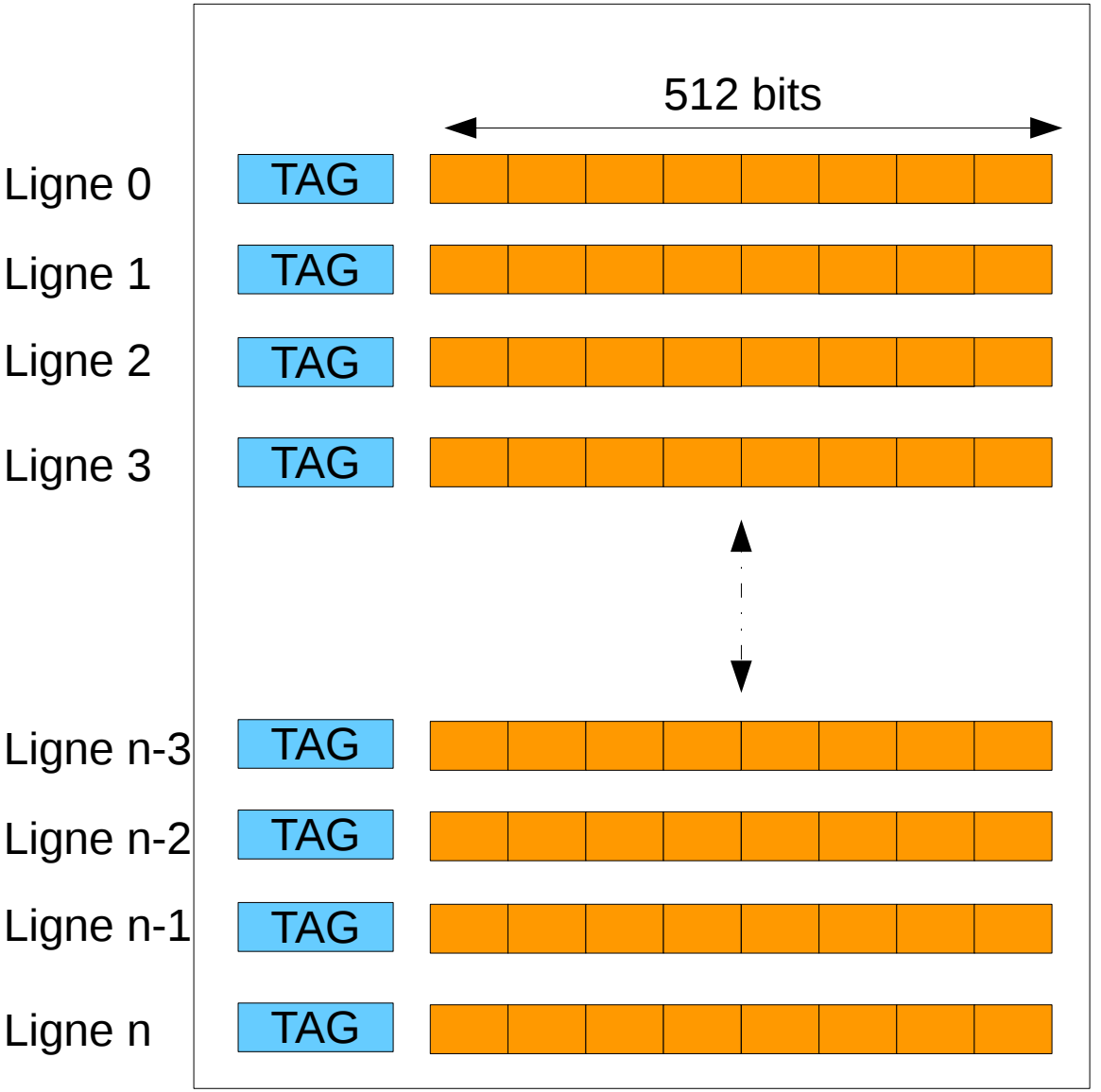
- Si on considère un cache de 16 Ko, combien il y a de lignes de cache ?
- Réponse :

$16\,\mathrm{Ko}/512\,\mathrm{bits} = 16 \times 2^{10}\,\mathrm{o}/64\,\mathrm{o} = 256$



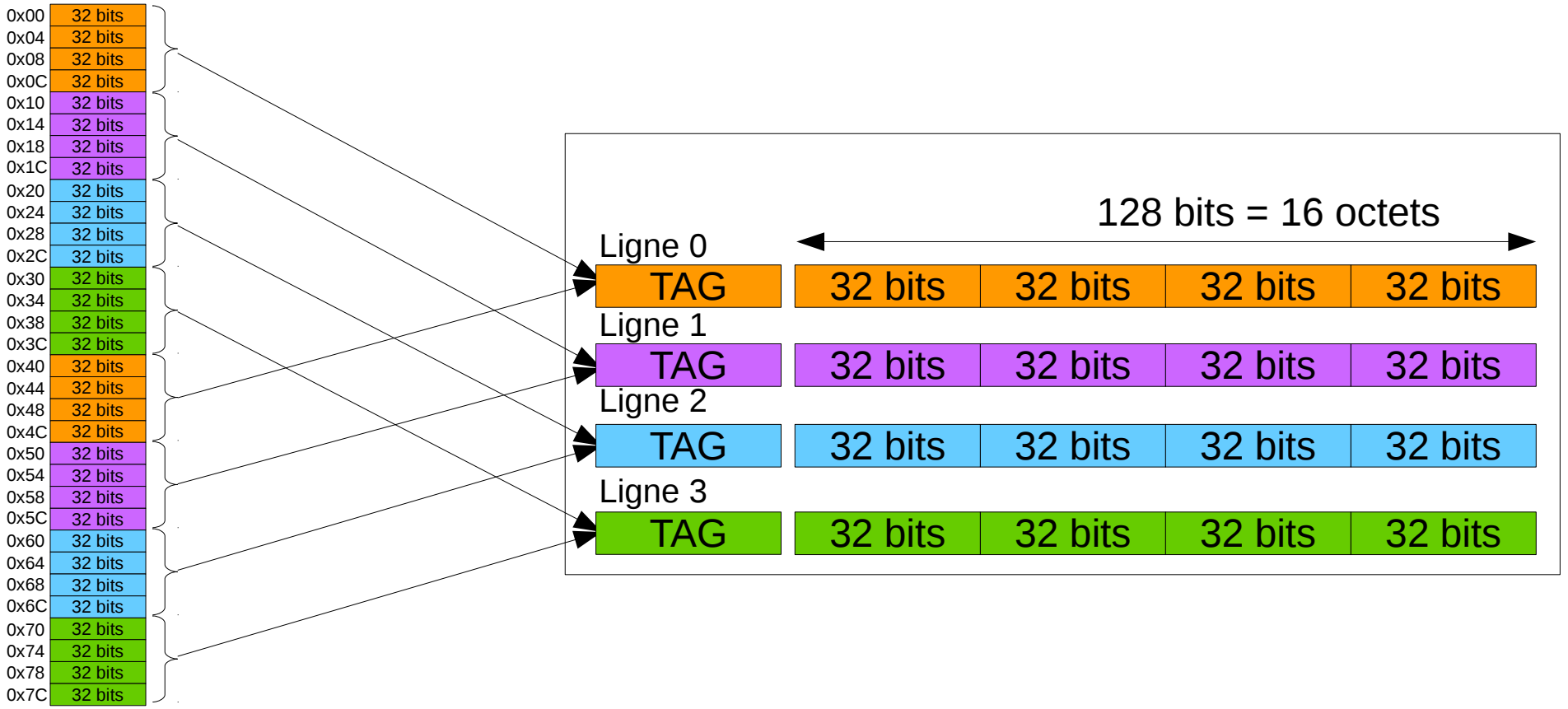
Organisation d’un cache

- Question :
  - Comment associe-t-on une ligne de cache à une adresse en mémoire centrale ?



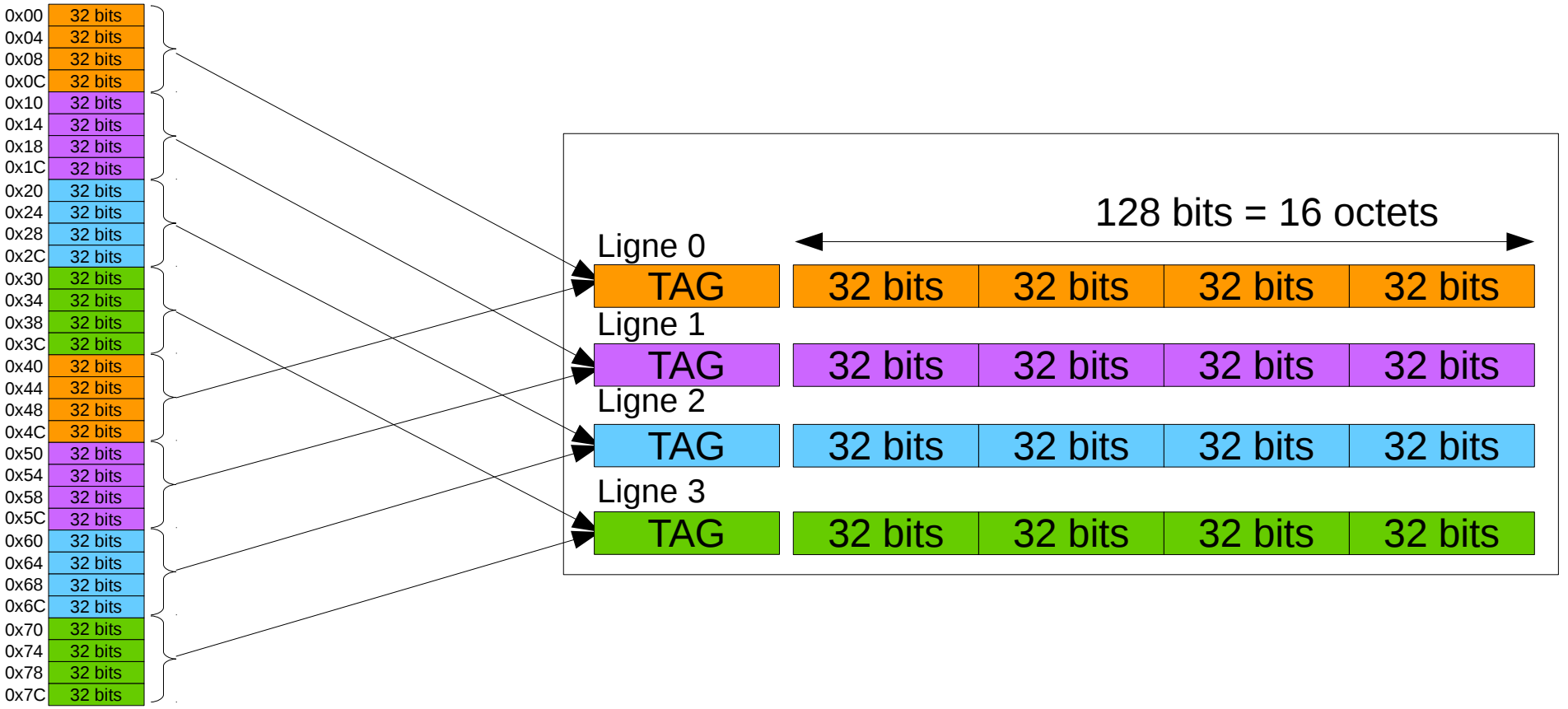
Cas du cache à correspondance pré-établie

- C’est le cas le plus simple :
  - on considère un cache de 4 lignes de 128 bits pour faire simple.
- Cela fonctionne comme un modulo



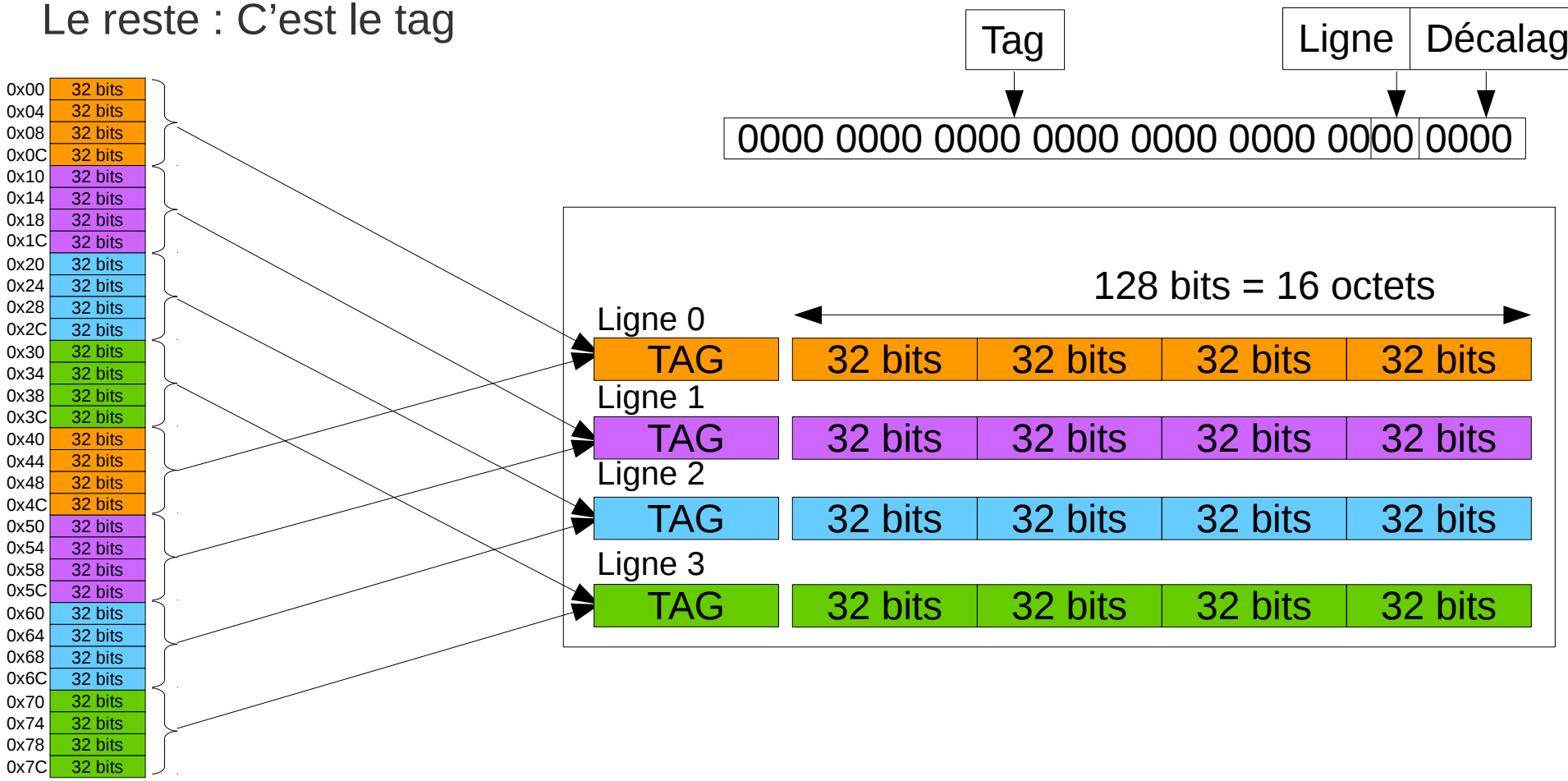
Cas du cache à correspondance pré-établie

- 0x00 → 0x0F : Ligne 0, 0x10 → 0x1F : ligne 1,
  0x20 → 0x2F : Ligne 2, 0x30 → 0x3F : ligne 3.
- 0x40 → 0x4F : Ligne 0, 0x50 → 0x5F : ligne 1,
  0x50 → 0x5F : Ligne 2, 0x50 → 0x5F : ligne 3.



Cas du cache à correspondance pré-établie

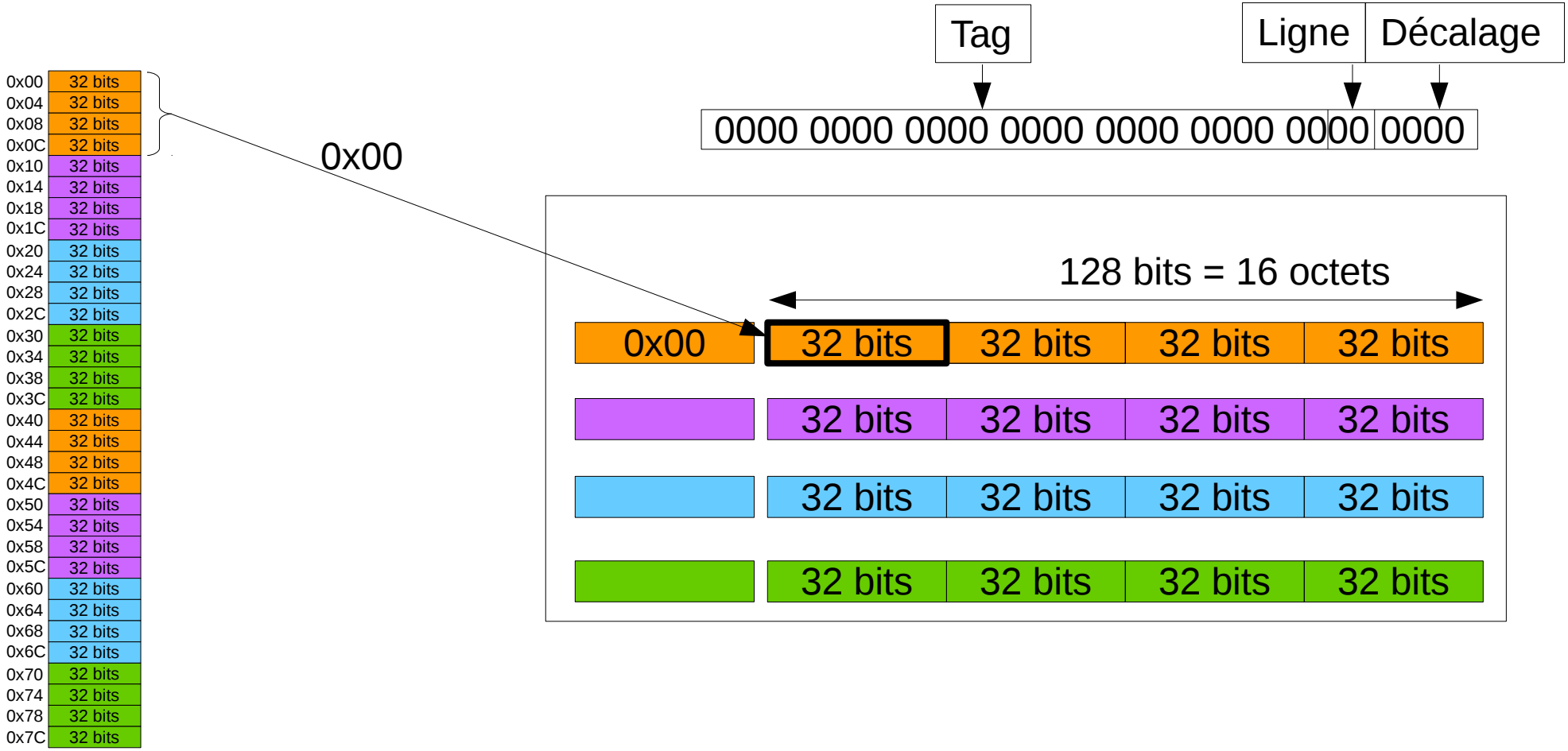
- 16 octets par ligne → 4 bits pour coder le décalage de l'octet
- 4 lignes → 2 bits pour coder la ligne
- Le reste : C'est le tag



Cas du cache à correspondance pré-établie

Adresse physique ↔ Tag | Ligne | Décalage

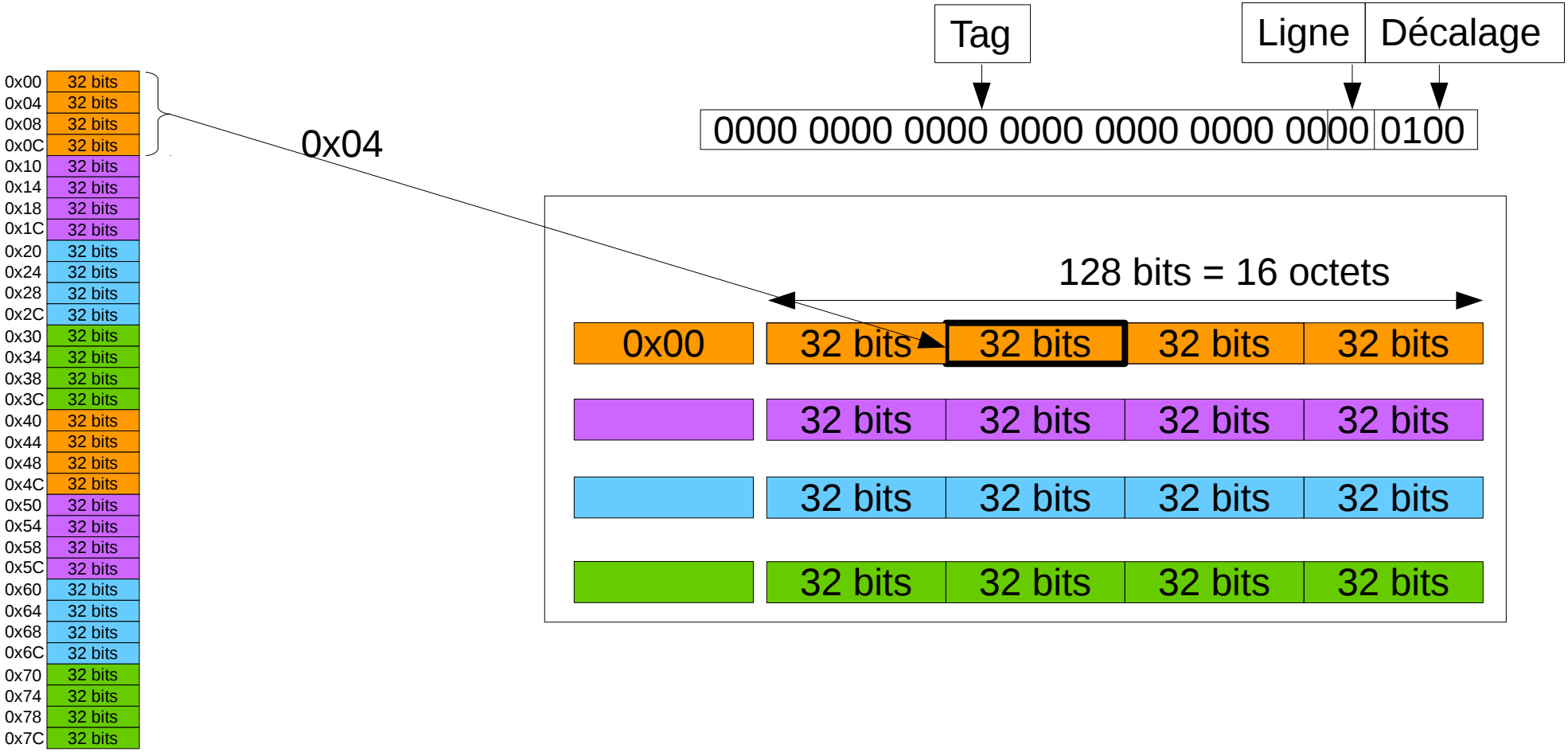
0x00 0x00 0x0 0x0



Cas du cache à correspondance pré-établie

Adresse physique ↔ Tag | Ligne | Décalage

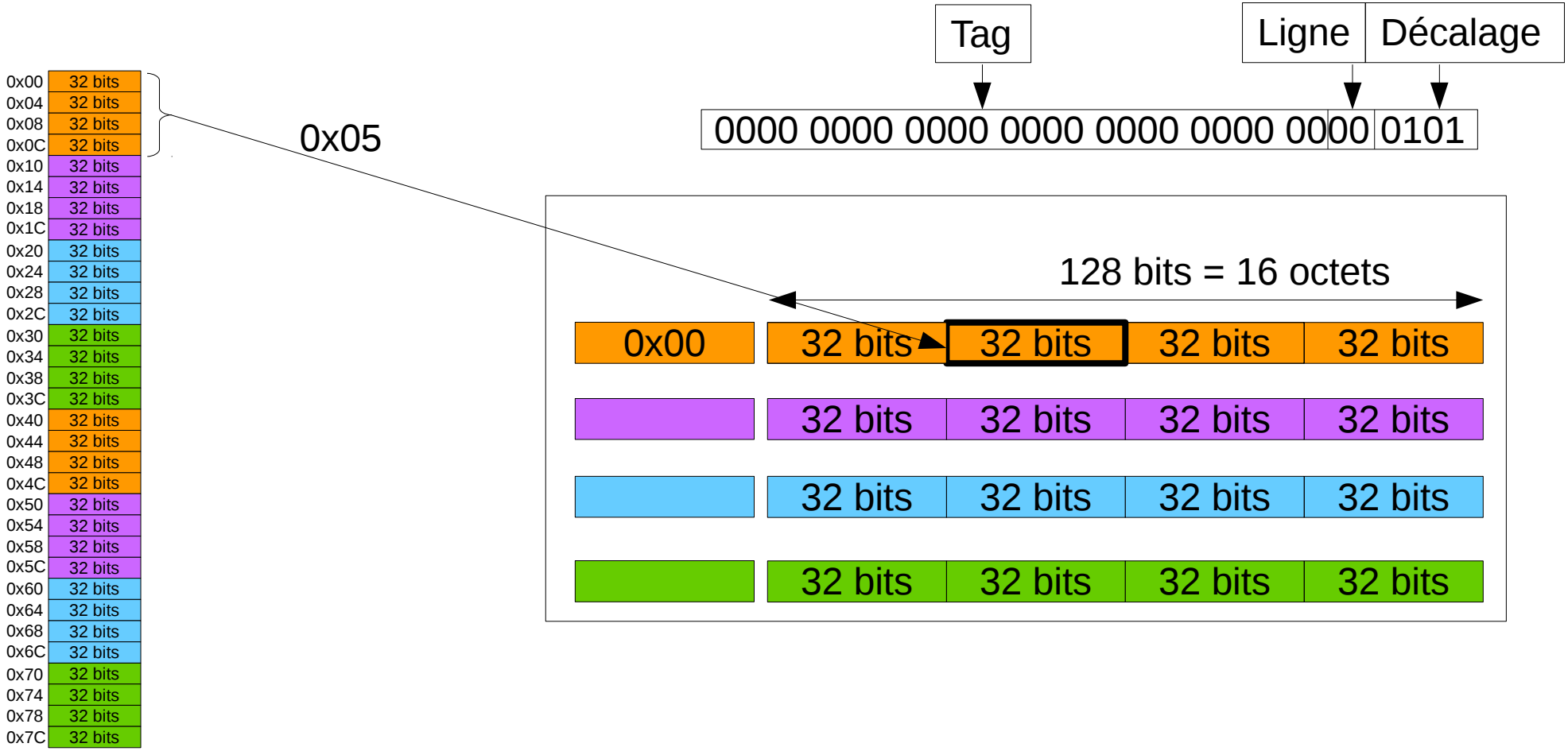
0x04 0x00 0x0 0x4



Cas du cache à correspondance pré-établie

Adresse physique ↔ Tag | Ligne | Décalage

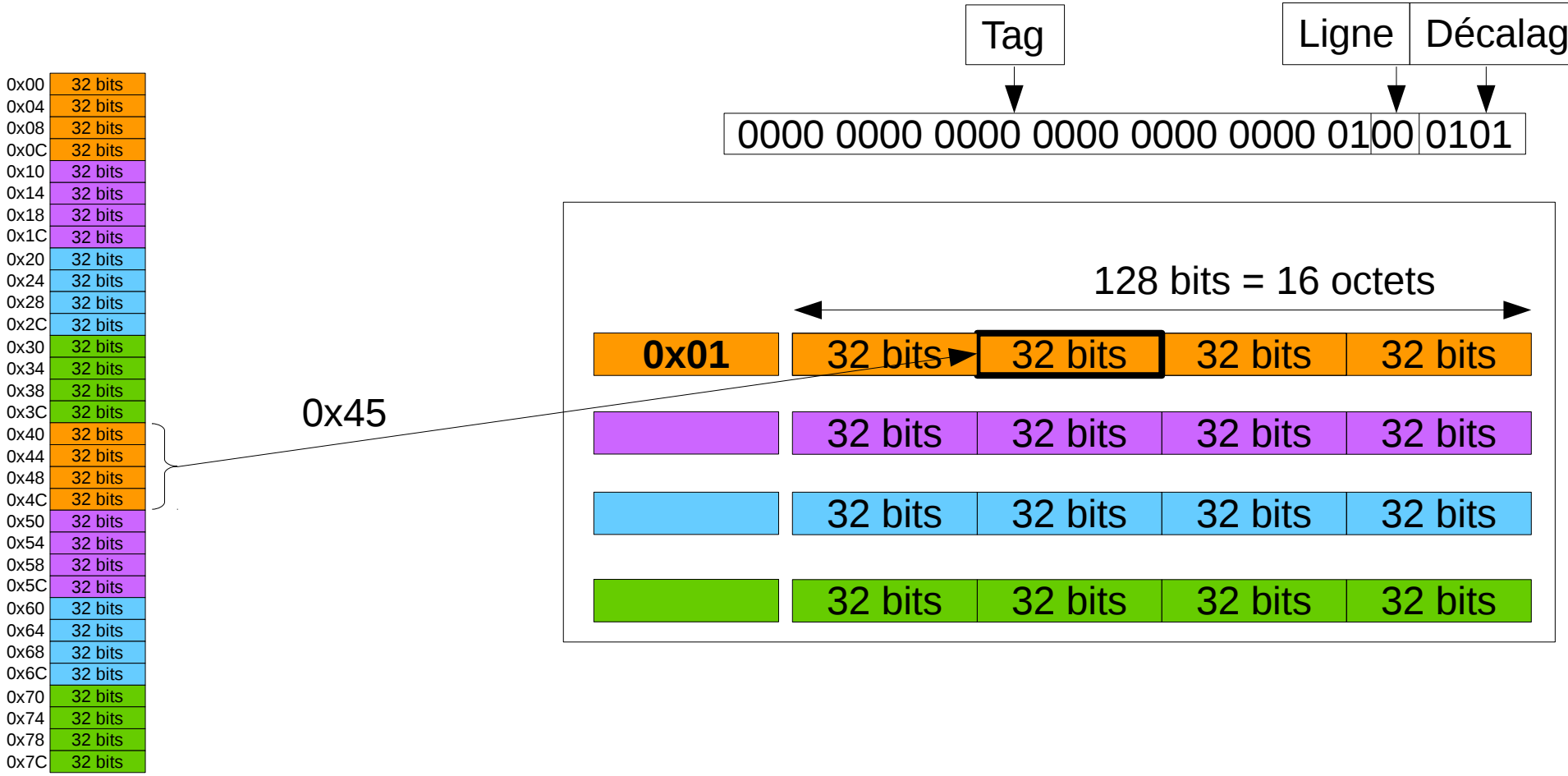
0x05 0x00 0x0 0x5



Cas du cache à correspondance pré-établie

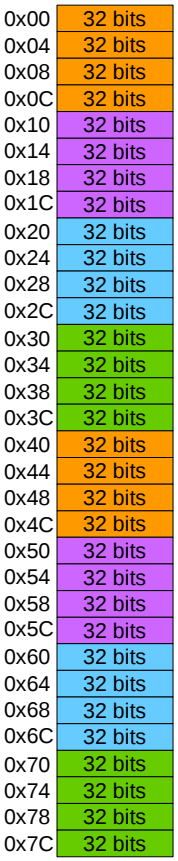
Adresse physique ↔ Tag | Ligne | Décalage

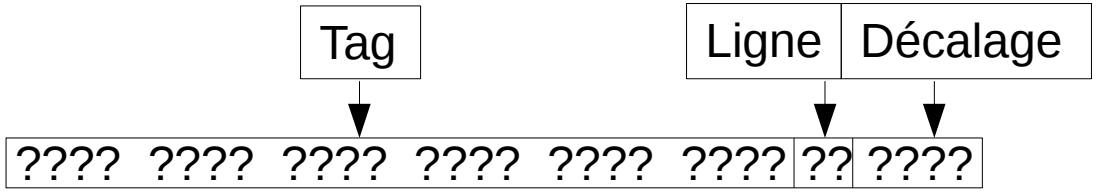
0x45 0x01 0x0 0x5

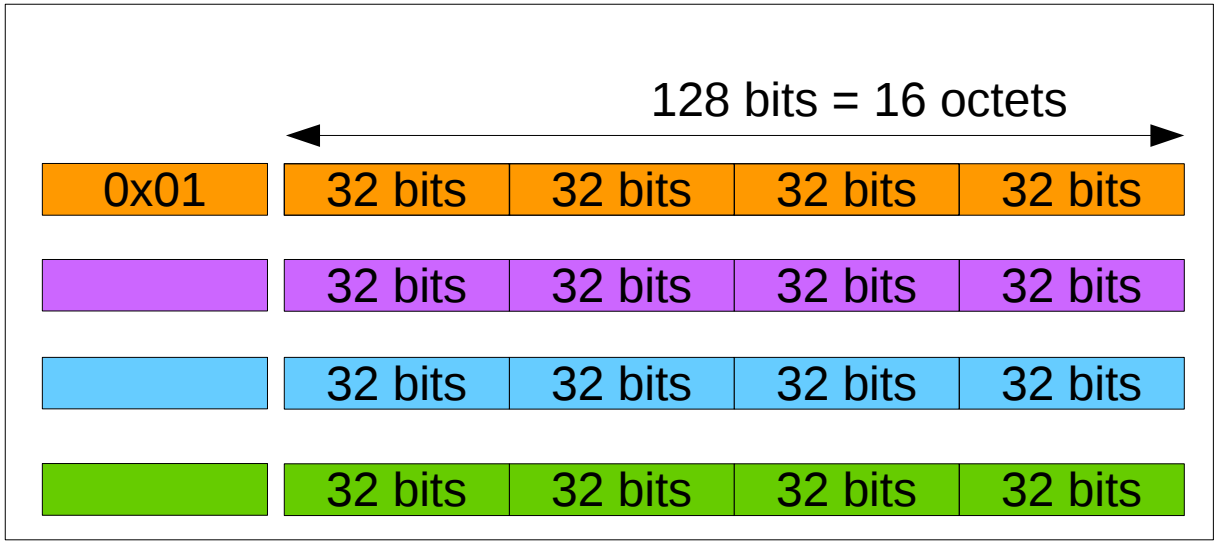


A vous de jouer

Adresse physique	↔	Tag	Ligne	Décalage
0x6F		0x ??	0x?	0x?

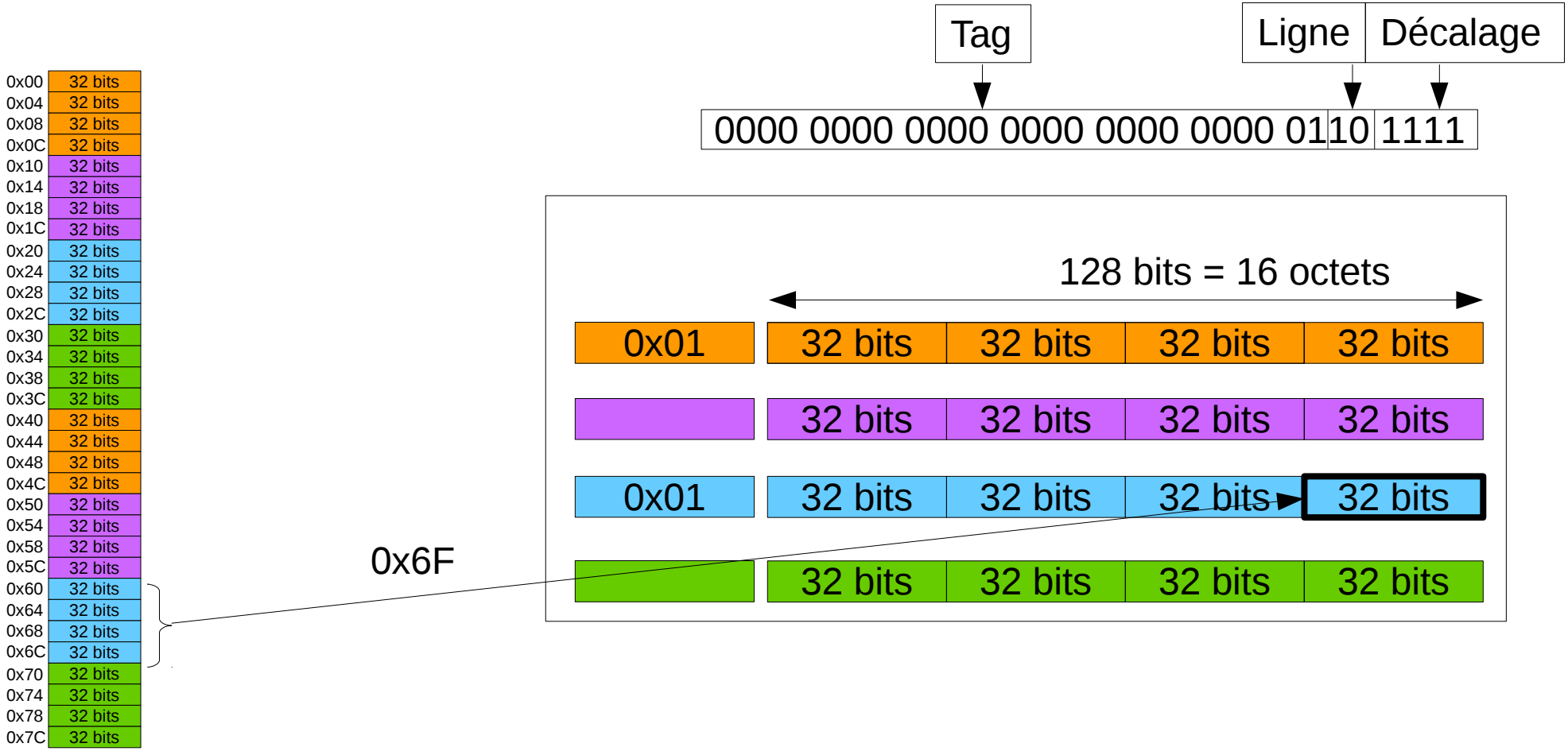






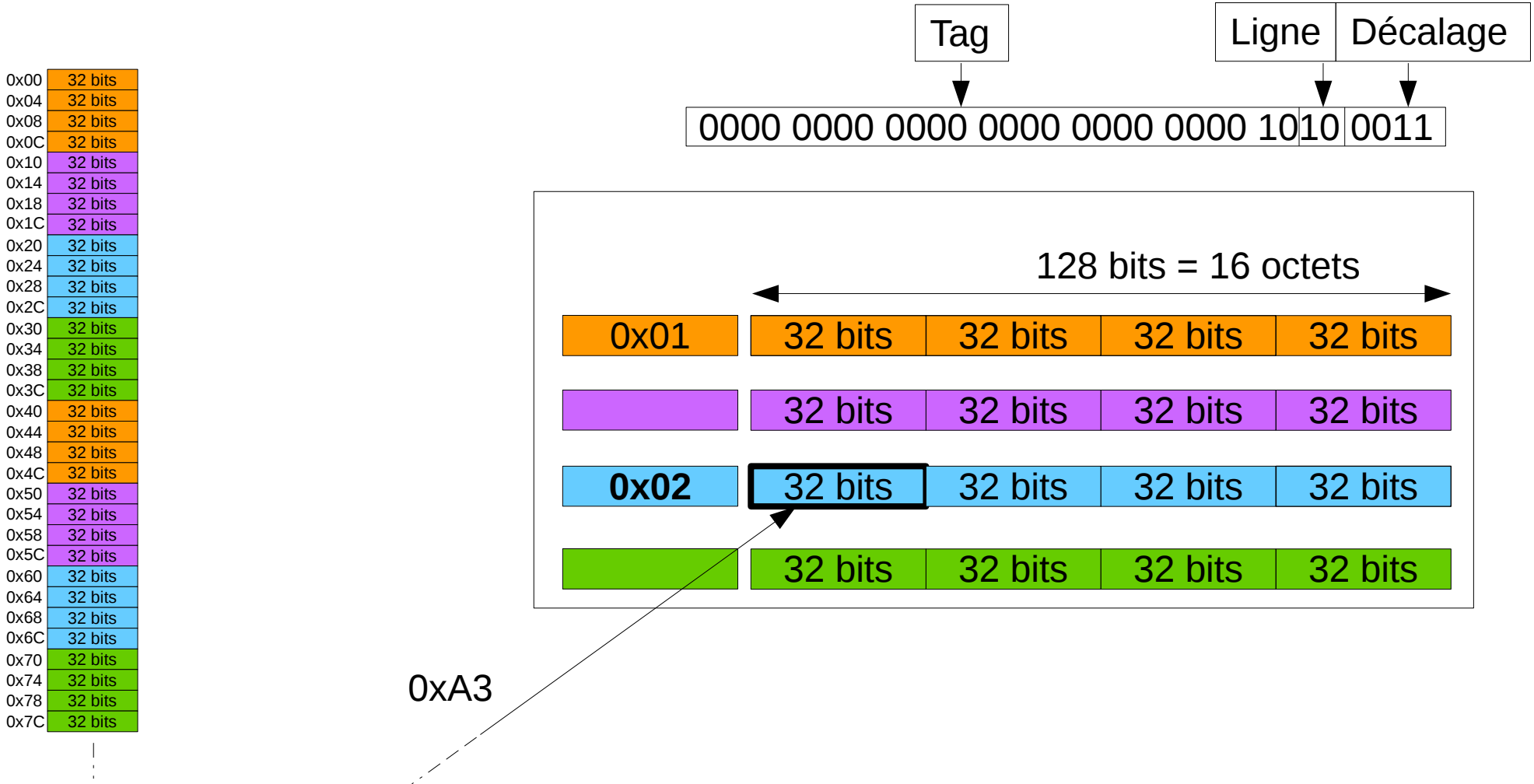
A vous de jouer : Réponse

Adresse physique	↔	Tag	Ligne	Décalage
0x6F		0x01	0x2	0xF



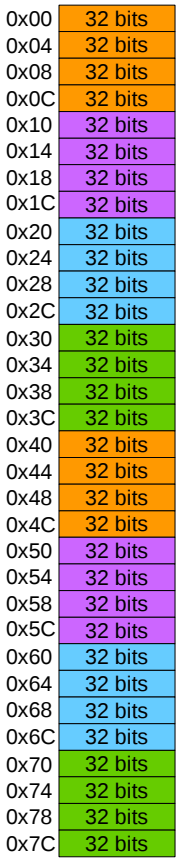
Dernier exemple

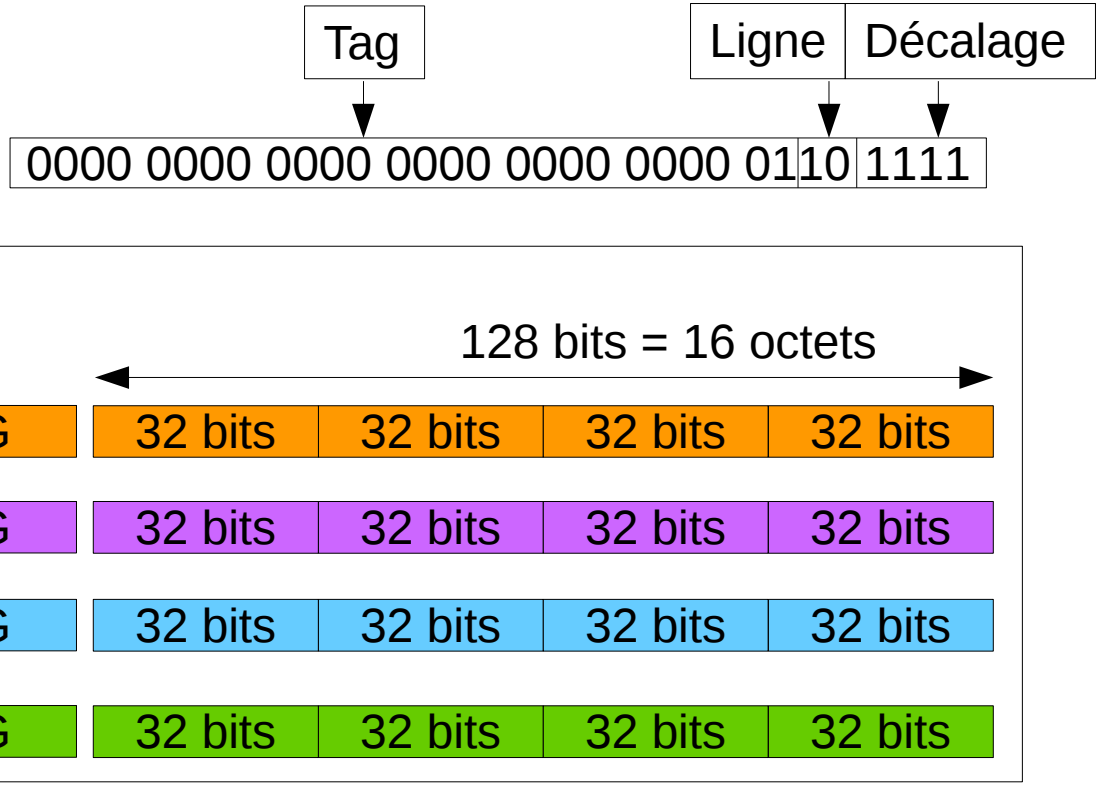
Adresse physique	↔	Tag	Ligne	Décalage
0xA3		0x02	0x2	0x3



Exercice de conclusion

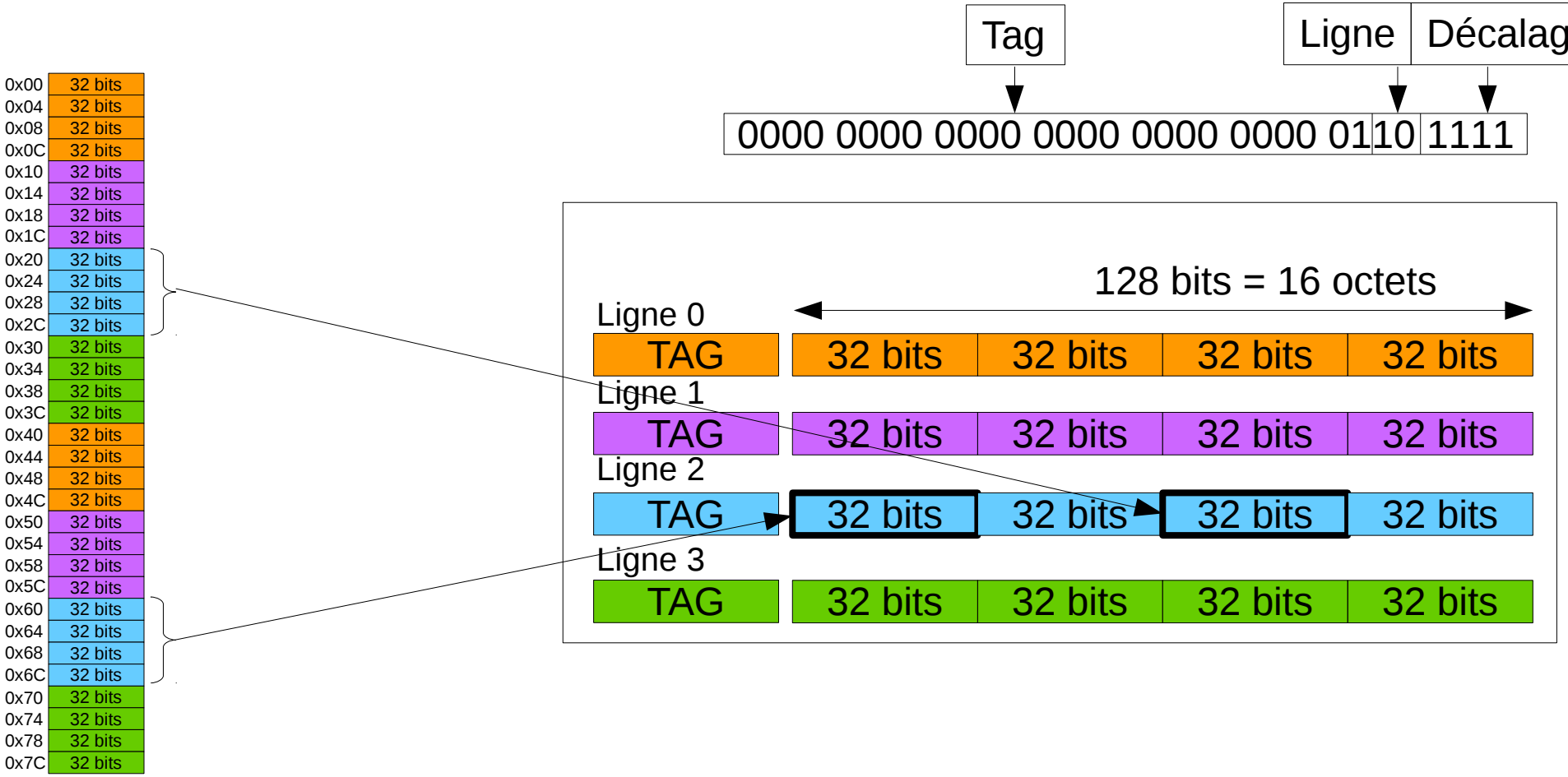
- Dans cette configuration, calculez le TAG, la ligne et le décalage des adresses suivantes : 0x28, 0x60





Exercice de conclusion

- 0x28 → 0b0010 1000 → TAG : 0, LIGNE : 2, DECA. : 8
- 0x60 → 0b0110 0000 → TAG : 1, LIGNE : 2, DECA. : 0

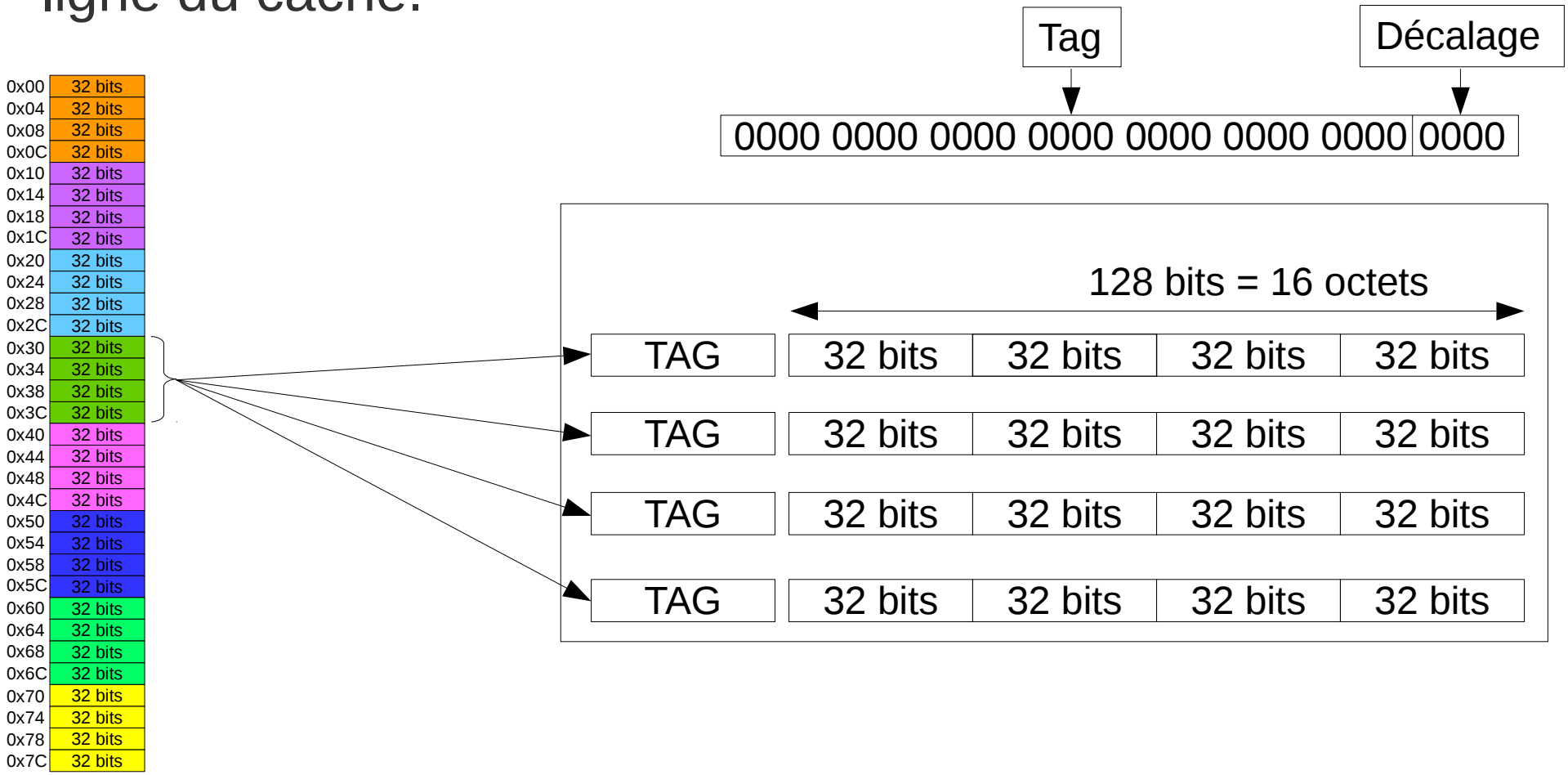


Avantages et inconvénients du cache à correspondance pré-établie

- Avantage :
  - Simplicité : Le cache décode la ligne et le TAG depuis l'adresse, puis est capable en 1 cycle de savoir si la donnée est présente ou non.
- Désavantage :
  - C'est son caractère déterministe (à une adresse correspond une ligne de cache). Cela signifie que la performance est fortement dépendante du programme actuel et du comportement des autres programmes qui tournent en parallèle.

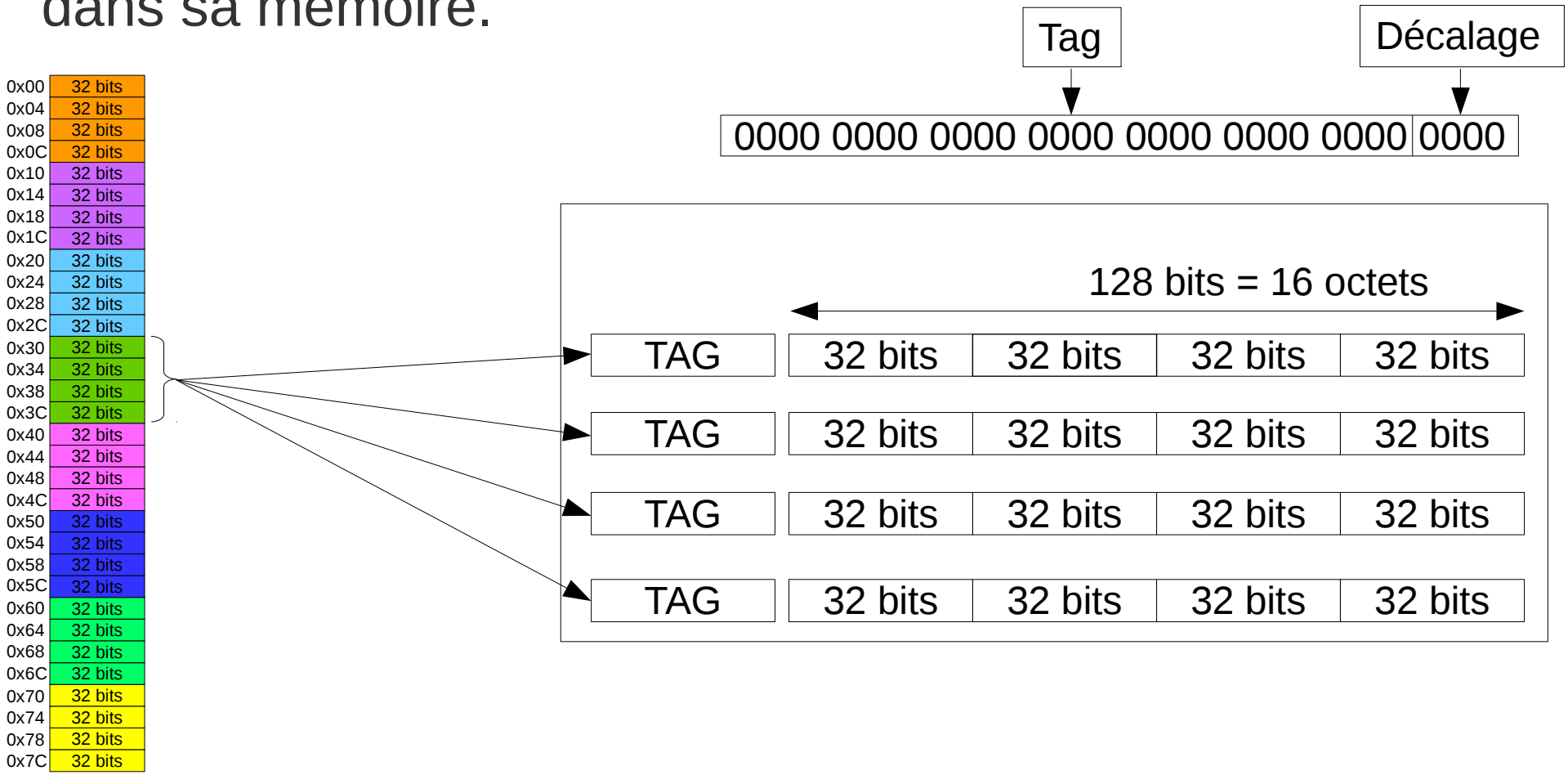
The ultimate cache : Totalement associatif

- Il n'y a aucun lien de correspondance ligne ↔ adresse, chaque adresse peux être associée à n'importe quelle ligne du cache.



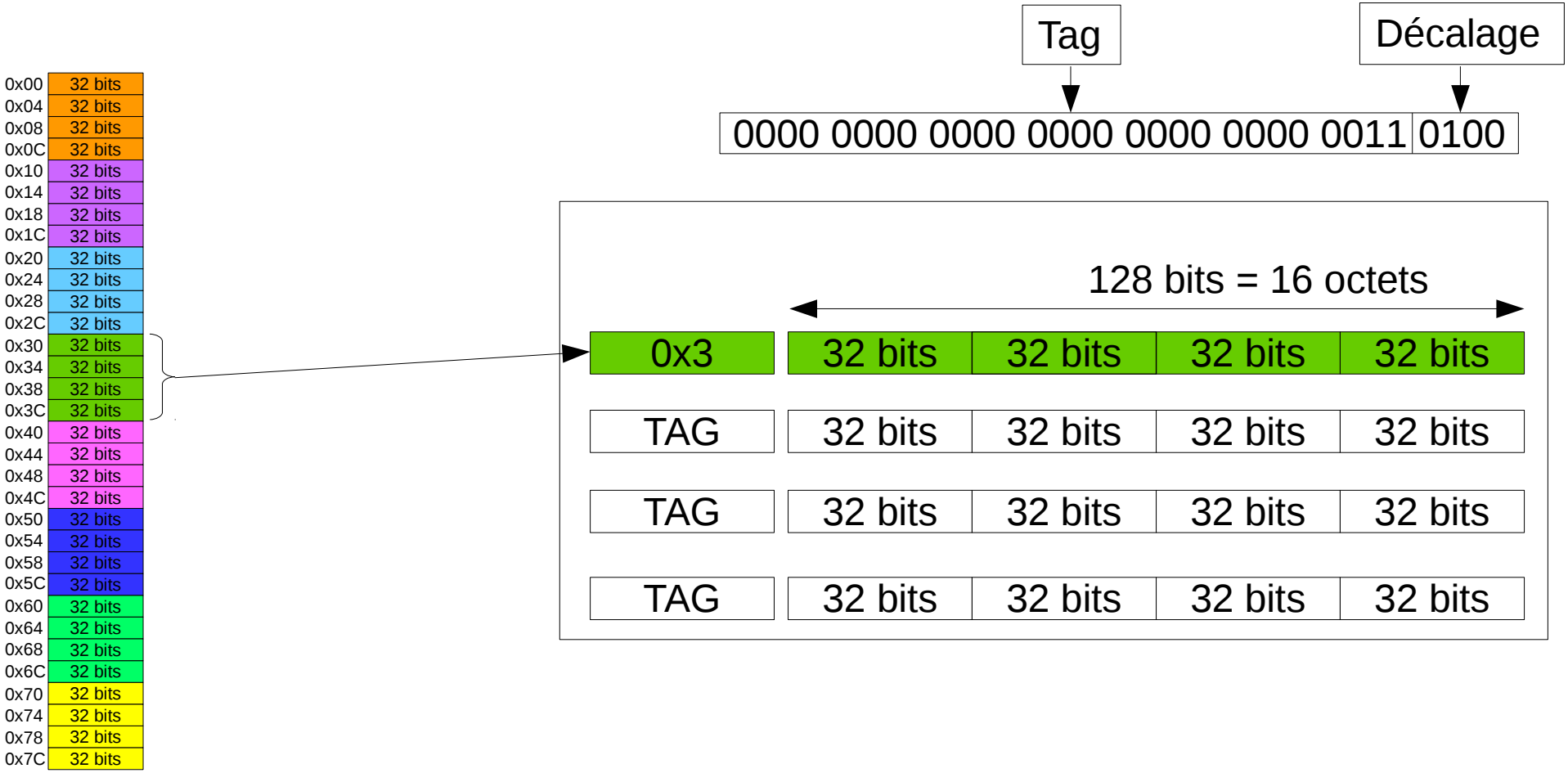
The ultimate cache : Totalement associatif

- En contre-partie, le cache doit vérifier le TAG de chaque ligne avant de savoir si la donnée est présente ou non dans sa mémoire.



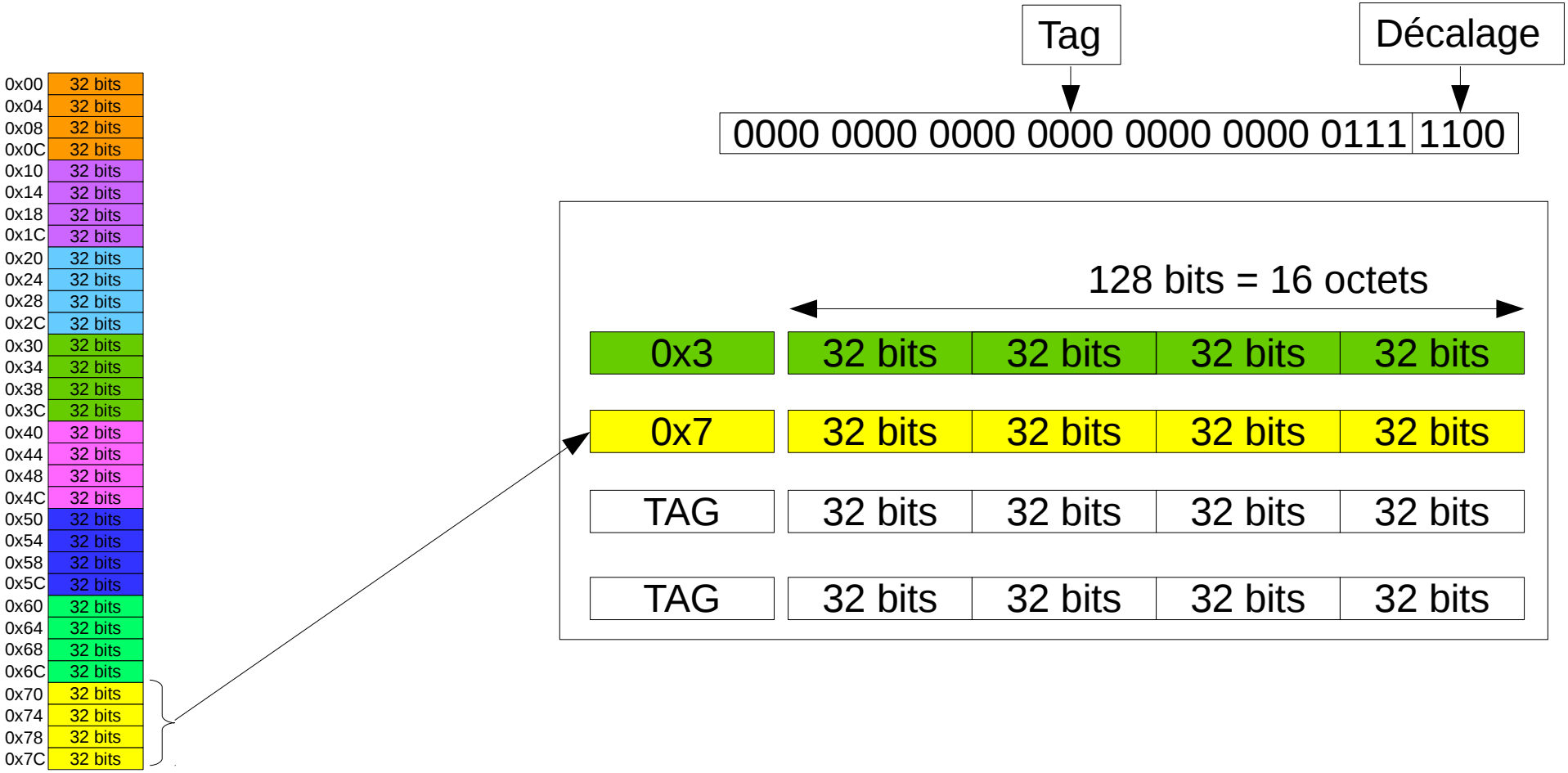
The ultimate cache : Totalement associatif

Adresse physique demandée : 0x34



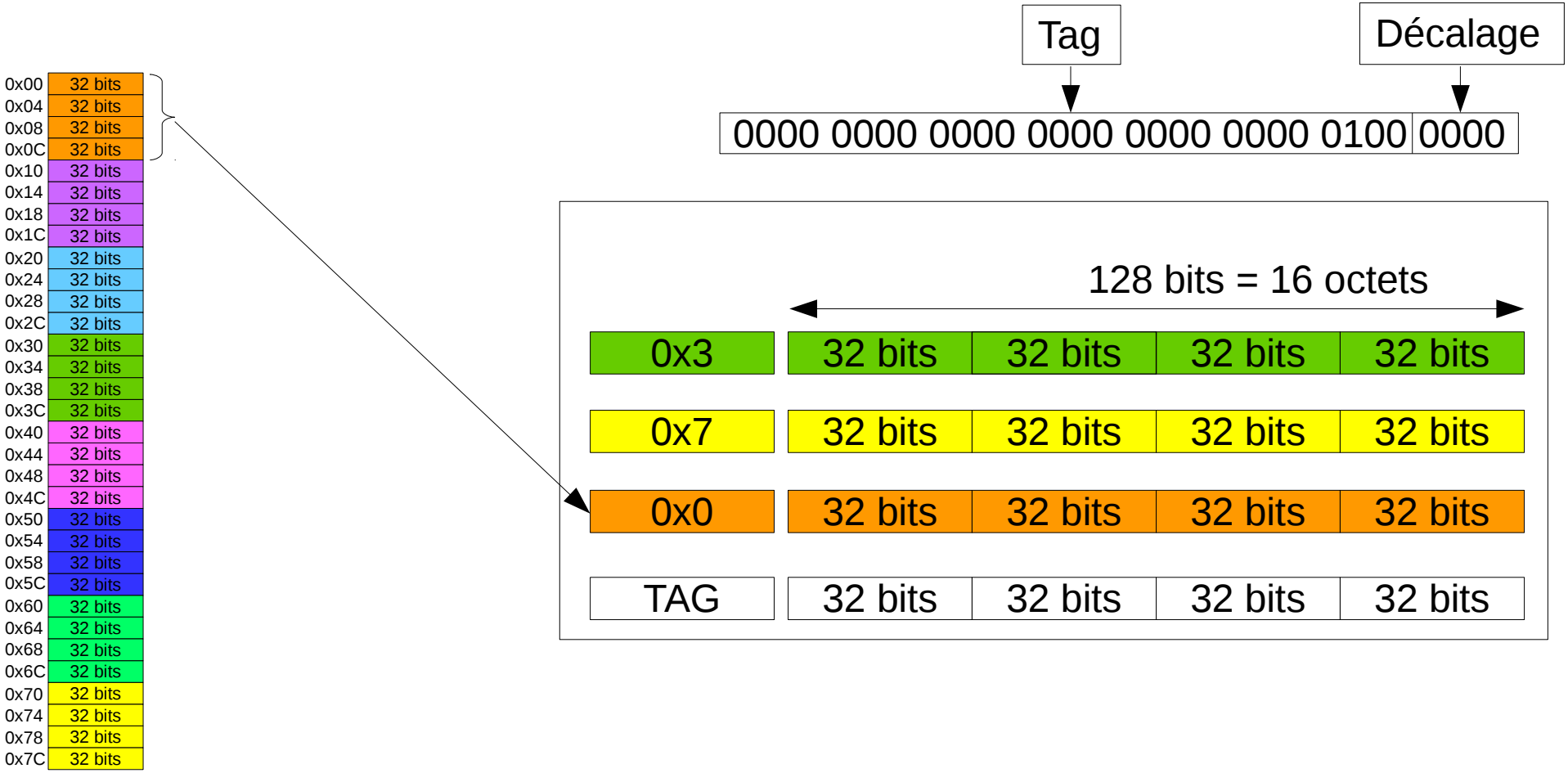
The ultimate cache : Totalement associatif

Adresse physique demandée : 0x7C



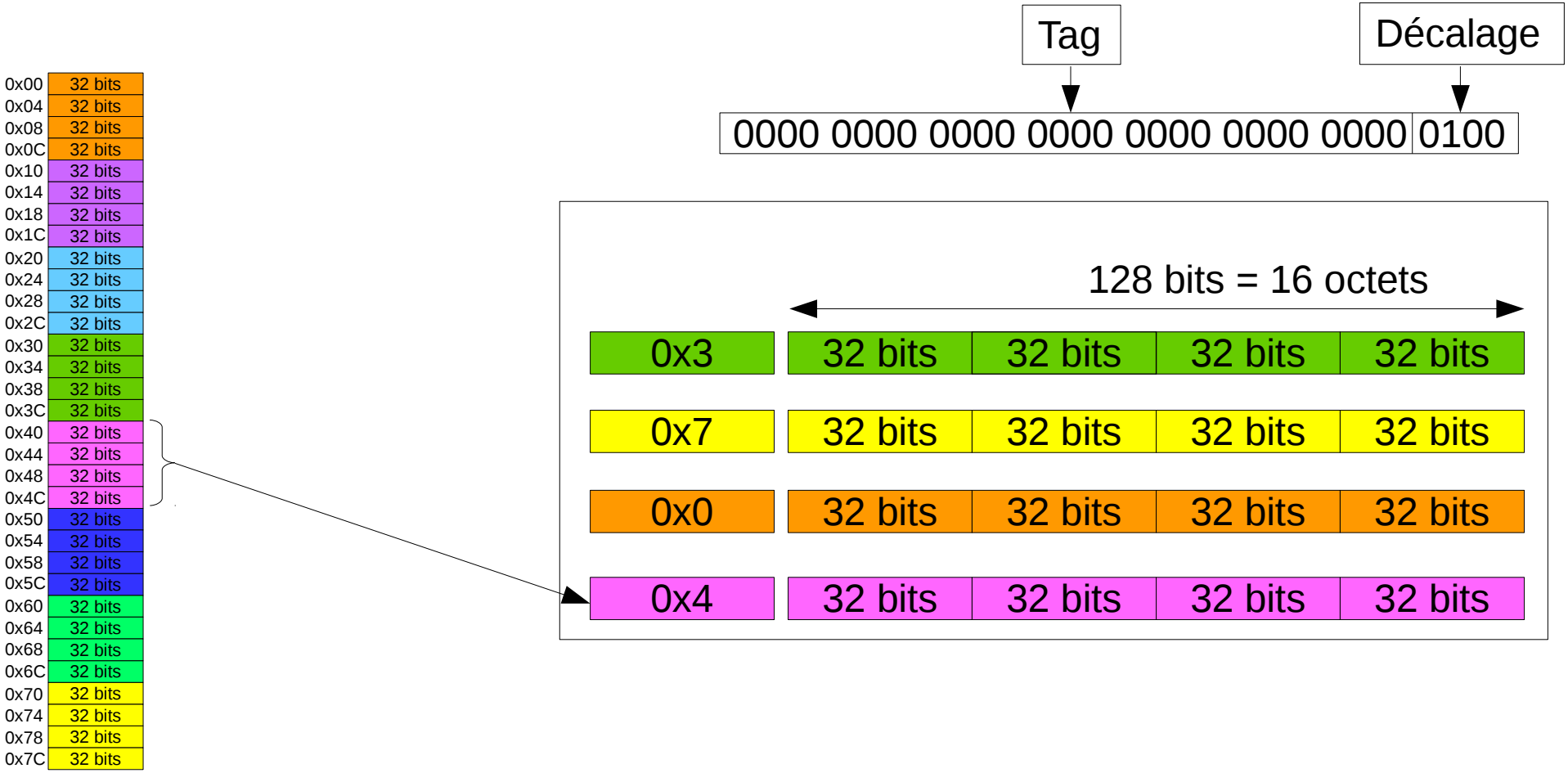
The ultimate cache : Totalement associatif

Adresse physique demandée : 0x04



The ultimate cache : Totalement associatif

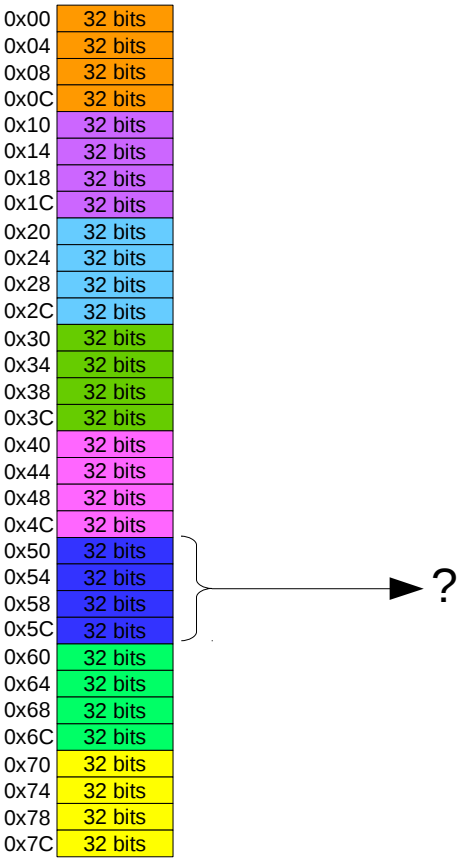
Adresse physique demandée : 0x40

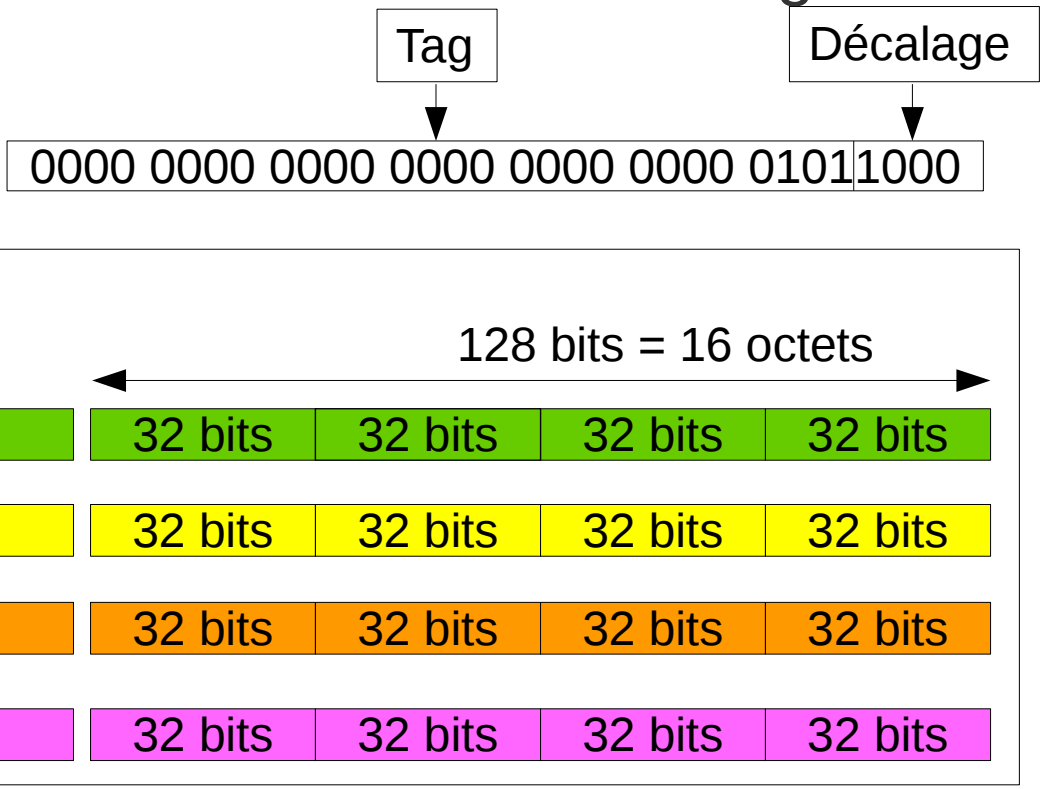


The ultimate cache : Totalement associatif

Adresse physique demandée : 0x58

Le cache étant rempli, comment choisir la ligne à supprimer ?



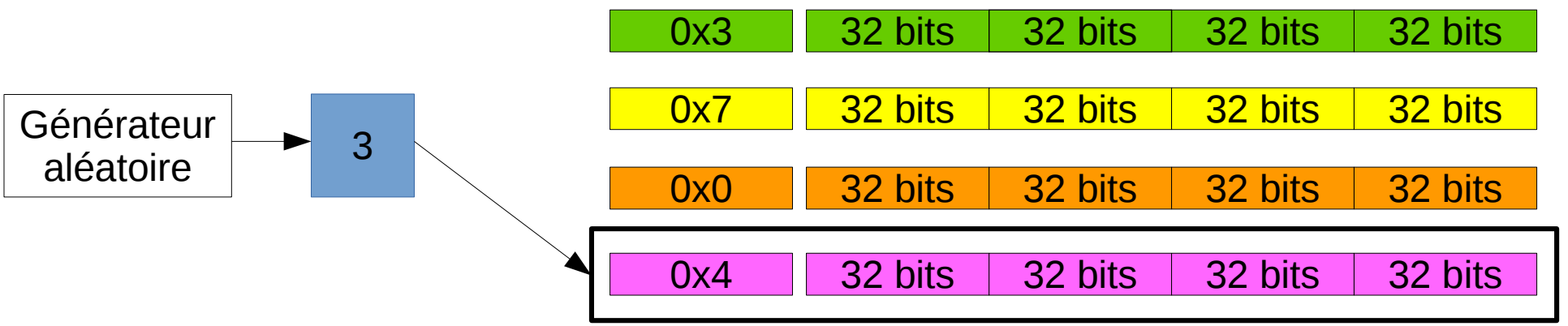


Algorithmes de remplacement

Algorithme de remplacement d'une ligne de type **Random** :

Le remplacement d'une ligne de cache se fait en sélectionnant aléatoirement une ligne

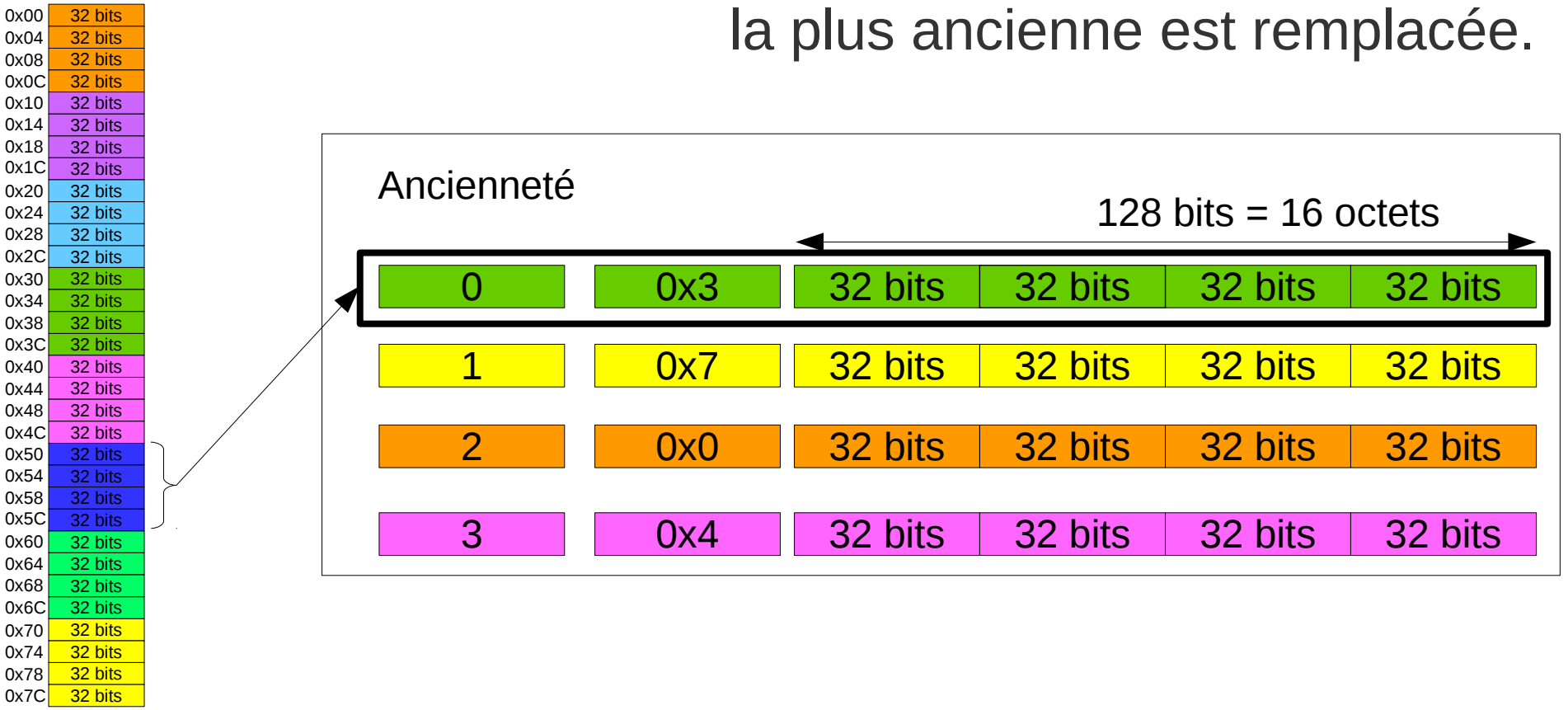
Simple dans sa conception, mais on peut très bien malencontreusement remplacer une ligne qui est fortement utilisée.



Algorithmes de remplacement

Algorithme de remplacement d'une ligne de type **FIFO / Round Robin** :

First **I**n **F**irst **O**ut → La ligne dont l'ajout dans le cache est la plus ancienne est remplacée.

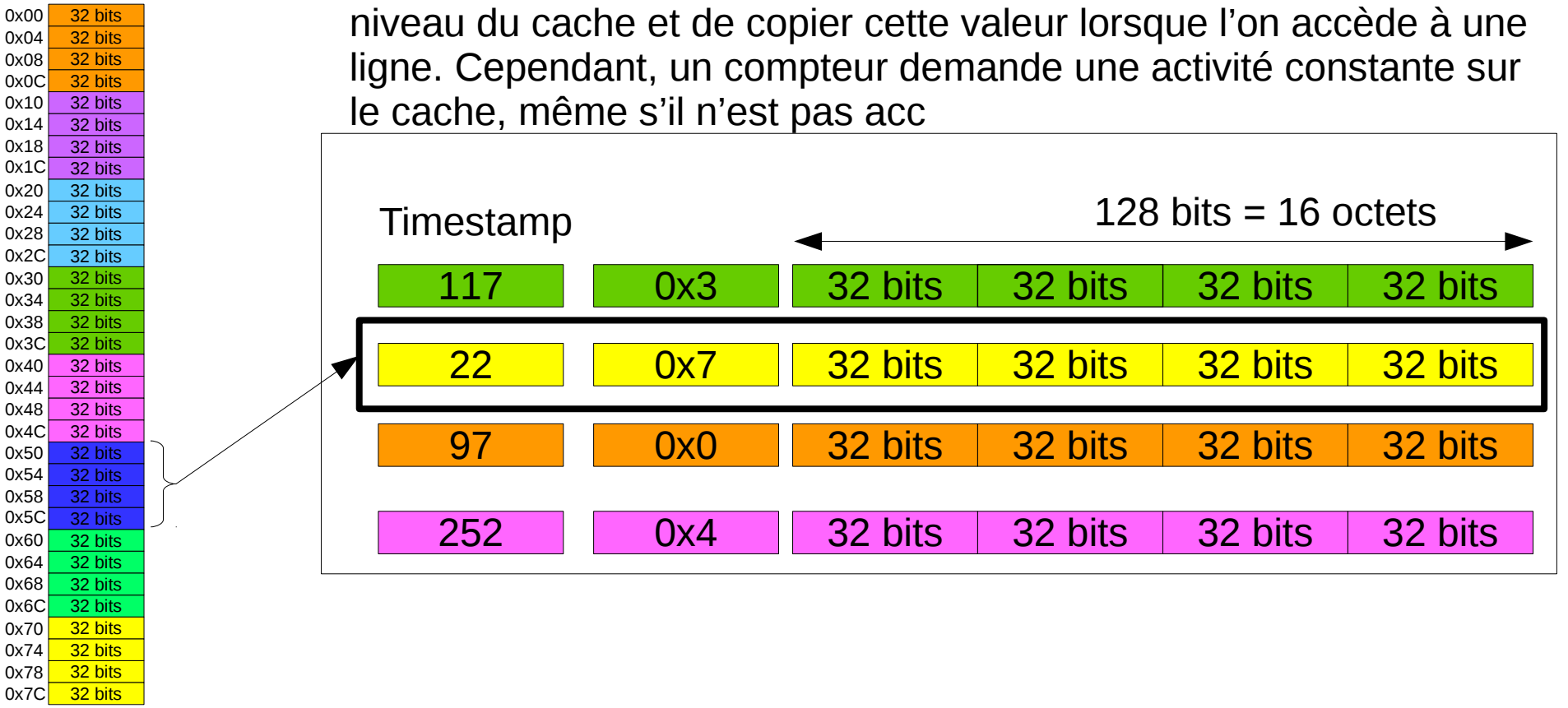


Algorithmes de remplacement

Algorithme de remplacement d'une ligne de type **LRU** :

Least **R**ecently **U**sed → Ligne dont l'accès est le plus lointain.

La méthode la plus simple directe est d'ajouter un compteur au niveau du cache et de copier cette valeur lorsque l'on accède à une ligne. Cependant, un compteur demande une activité constante sur le cache, même s'il n'est pas acc



Algorithmes de remplacement

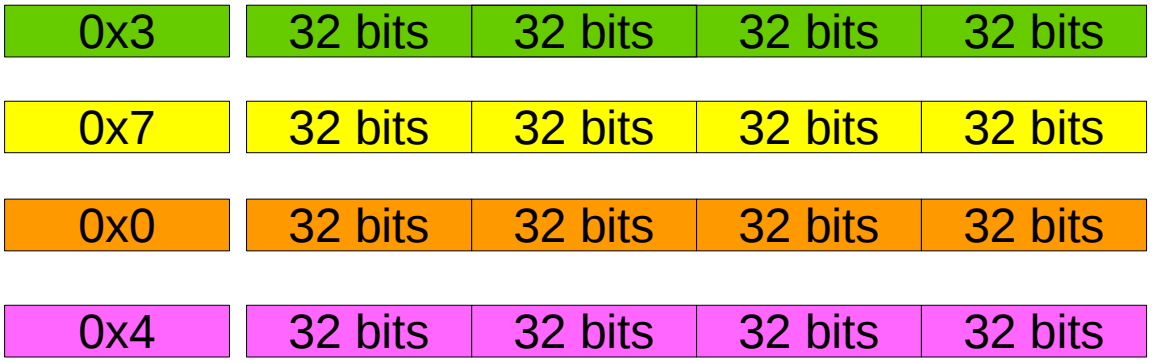
Algorithme de remplacement d'une ligne de type **LRU** :

Méthode matricielle.

Lorsque l'on accède à la ligne de cache i, on positionne à 1 la ligne i de la matrice, et on met à 0 la colonne i.

Accès à la ligne 2 :

0	0	0	0
0	0	0	0
0	0	0	0
0	0	0	0



Algorithmes de remplacement

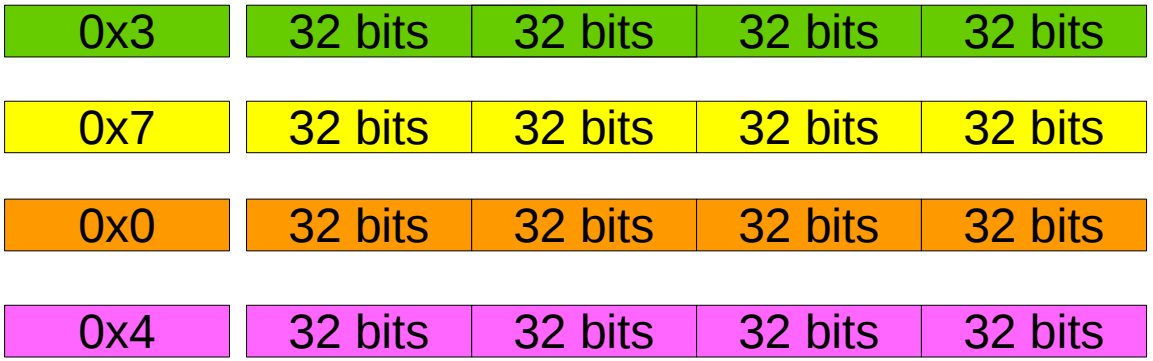
Algorithme de remplacement d'une ligne de type **LRU** :

Méthode matricielle.

Lorsque l'on accède à la ligne de cache i, on positionne à 1 la ligne i de la matrice, et on met à 0 la colonne i.

Accès à la ligne 2 :

0	0	**0**	0
0	0	**0**	0
1	**1**	**0**	**1**
0	0	**0**	0



Algorithmes de remplacement

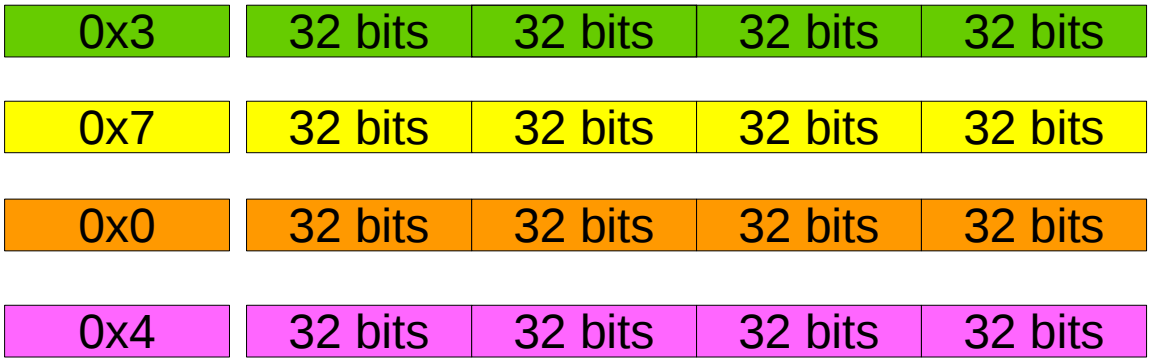
Algorithme de remplacement d'une ligne de type **LRU** :

Méthode matricielle.

Lorsque l'on accède à la ligne de cache i, on positionne à 1 la ligne i de la matrice, et on met à 0 la colonne i.

Accès à la ligne 0 :

0	**1**	**1**	**1**
0	0	0	0
0	1	0	1
0	0	0	0



Algorithmes de remplacement

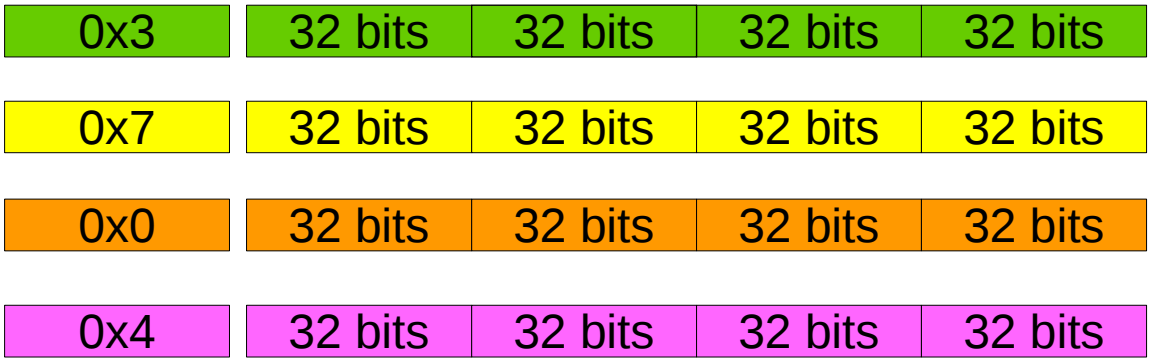
Algorithme de remplacement d'une ligne de type **LRU** :

Méthode matricielle.

Lorsque l'on accède à la ligne de cache i, on positionne à 1 la ligne i de la matrice, et on met à 0 la colonne i.

Accès à la ligne 1 :

0	**0**	1	1
1	**0**	**1**	**1**
0	**0**	0	1
0	**0**	0	0



Algorithmes de remplacement

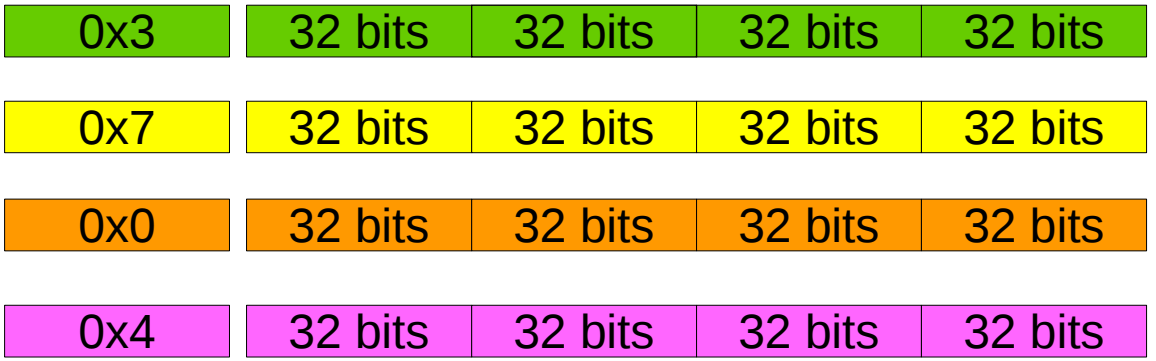
Algorithme de remplacement d'une ligne de type **LRU** :

Méthode matricielle.

Lorsque l'on accède à la ligne de cache i, on positionne à 1 la ligne i de la matrice, et on met à 0 la colonne i.

Accès à la ligne 3 :

0	0	1	**0**
1	0	1	**0**
0	0	0	**0**
1	**1**	**1**	**0**



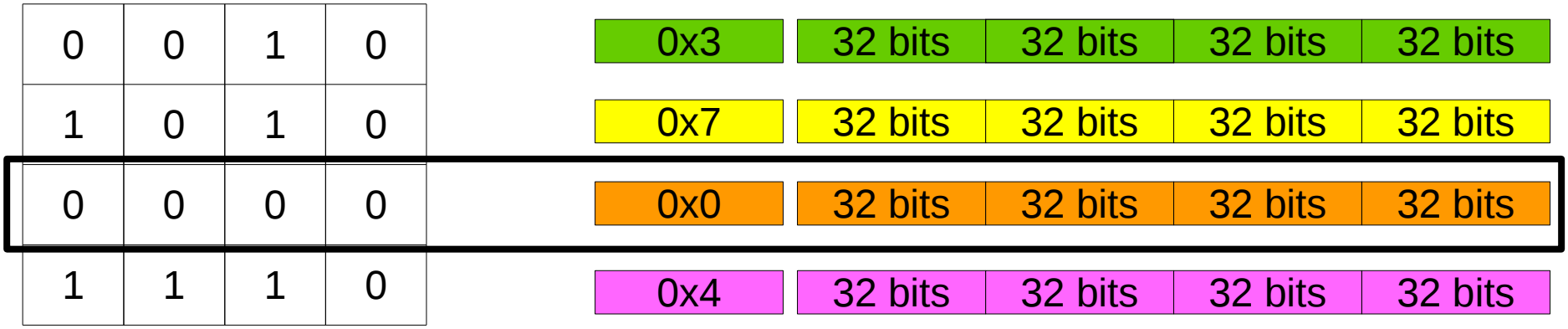
Algorithmes de remplacement

Algorithme de remplacement d'une ligne de type **LRU** :

Méthode matricielle.

Lorsque l'on accède à la ligne de cache i, on positionne à 1 la ligne i de la matrice, et on met à 0 la colonne i.

Remplacement d'une ligne de cache :



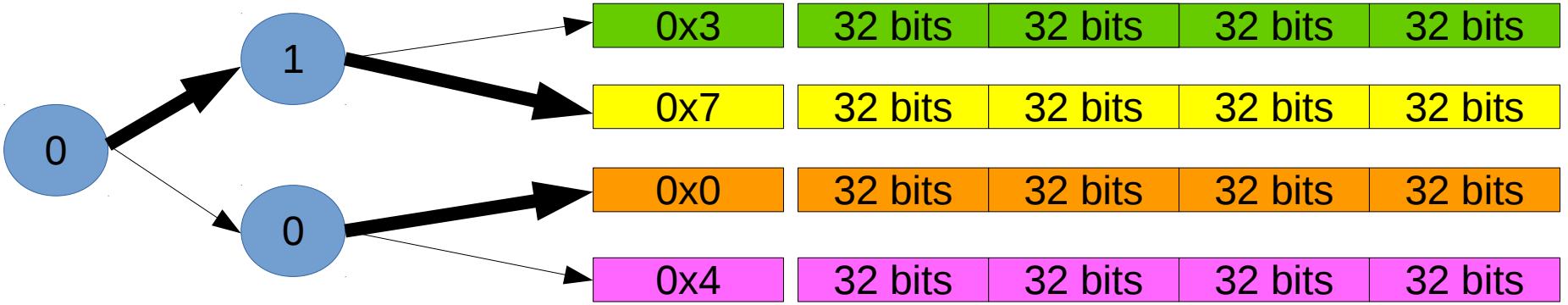
Défaut : cette méthode demande N^2 bits, N étant le nombre de lignes

Algorithmes de remplacement

Algorithme de remplacement d'une ligne de type **PLRUt** :

Méthode basé sur un arbre binaire.

On associe à chaque nœud de l'arbre un bit, qui sélectionne une branche ou l'autre en fonction de sa valeur. Dans cet exemple, le bit à 0 sélectionne la branche du haut, et un 1 sélectionne la branche du bas. Lorsque l'on accède à une ligne, on va inverser toutes les branches qui pointent vers la ligne.



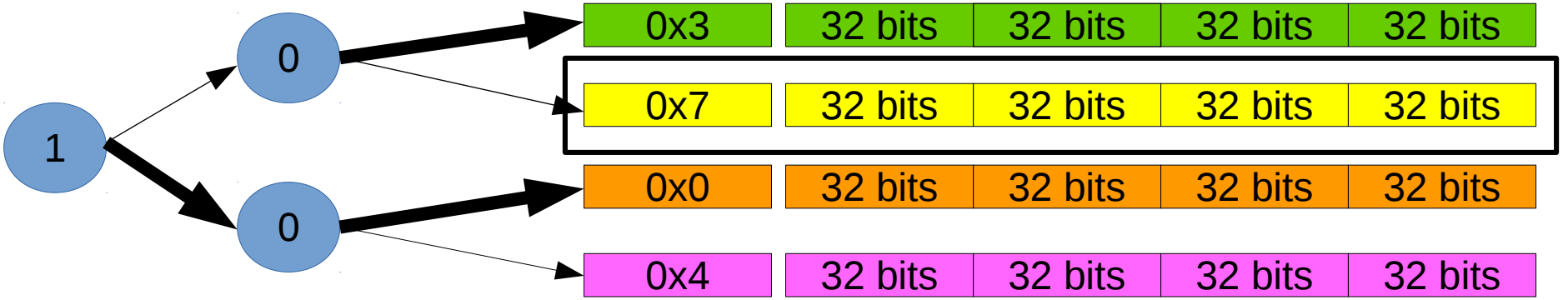
Algorithmes de remplacement

Algorithme de remplacement d'une ligne de type **PLRUt** :

Méthode basé sur un arbre binaire.

On associe à chaque nœud de l'arbre un bit, qui sélectionne une branche ou l'autre en fonction de sa valeur. Dans cet exemple, le bit à 0 sélectionne la branche du haut, et un 1 sélectionne la branche du bas. Lorsque l'on accède à une ligne, on va inverser toutes les branches qui pointent vers la ligne.

Accès à la ligne 1 :



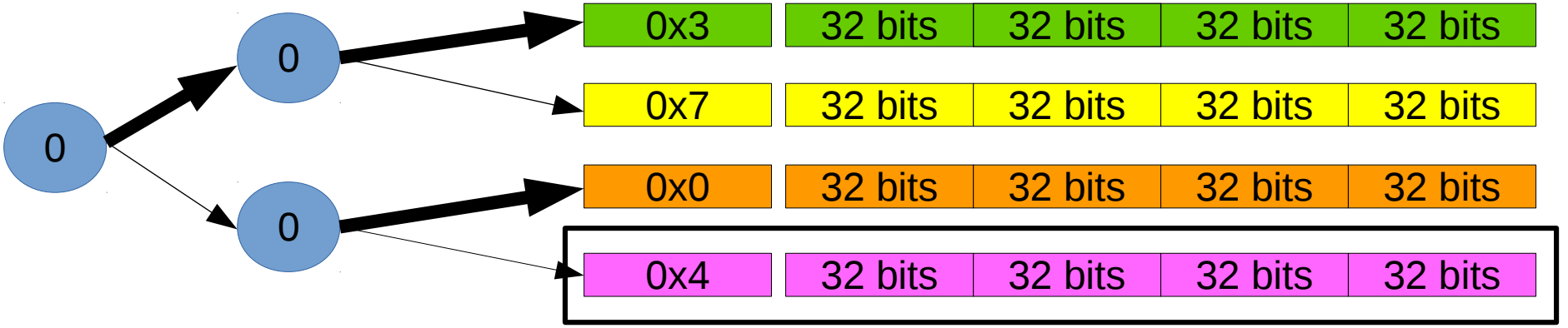
Algorithmes de remplacement

Algorithme de remplacement d'une ligne de type **PLRUt** :

Méthode basé sur un arbre binaire.

On associe à chaque nœud de l'arbre un bit, qui sélectionne une branche ou l'autre en fonction de sa valeur. Dans cet exemple, le bit à 0 sélectionne la branche du haut, et un 1 sélectionne la branche du bas. Lorsque l'on accède à une ligne, on va inverser toutes les branches qui pointent vers la ligne.

Accès à la ligne 3 :

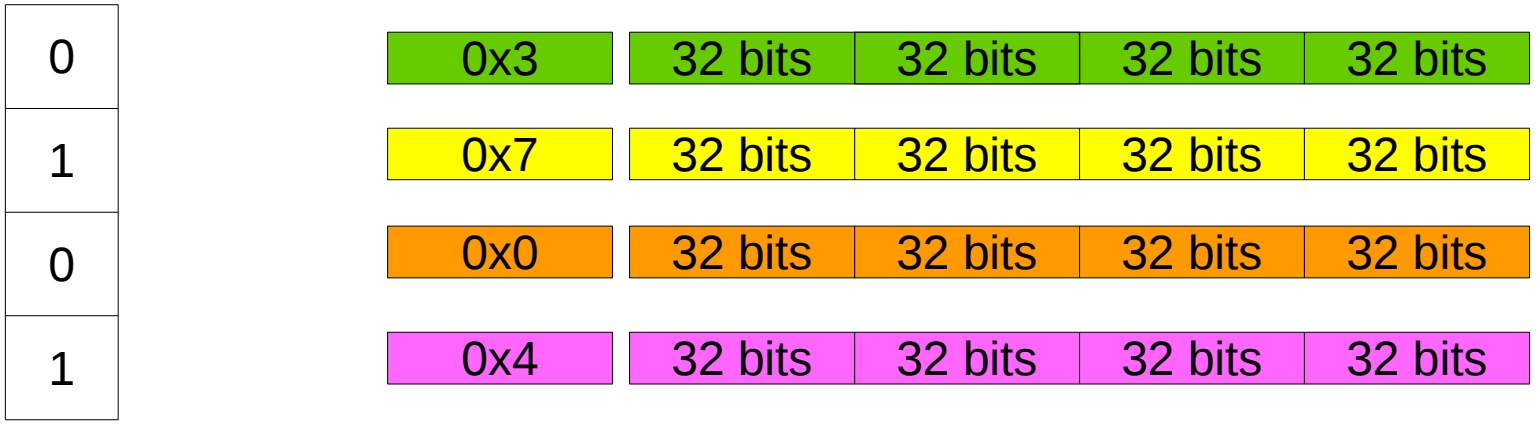


Défaut : L'histoire de chaque nœud est partiellement pris en compte. Le nœud parent par exemple divise en deux groupes les nœuds, gommant une partie de l'histoire de nœuds enfants.

Algorithmes de remplacement

Algorithme de remplacement d'une ligne de type **PLRUm** :

Chaque ligne est associé à un bit. Lorsque la ligne est accédée, en positionne à 1 le bit de la ligne. Lorsque tous les bits sont à 1, on met à 0 tous les bits sauf celui qui vient d'être positionné à 1.

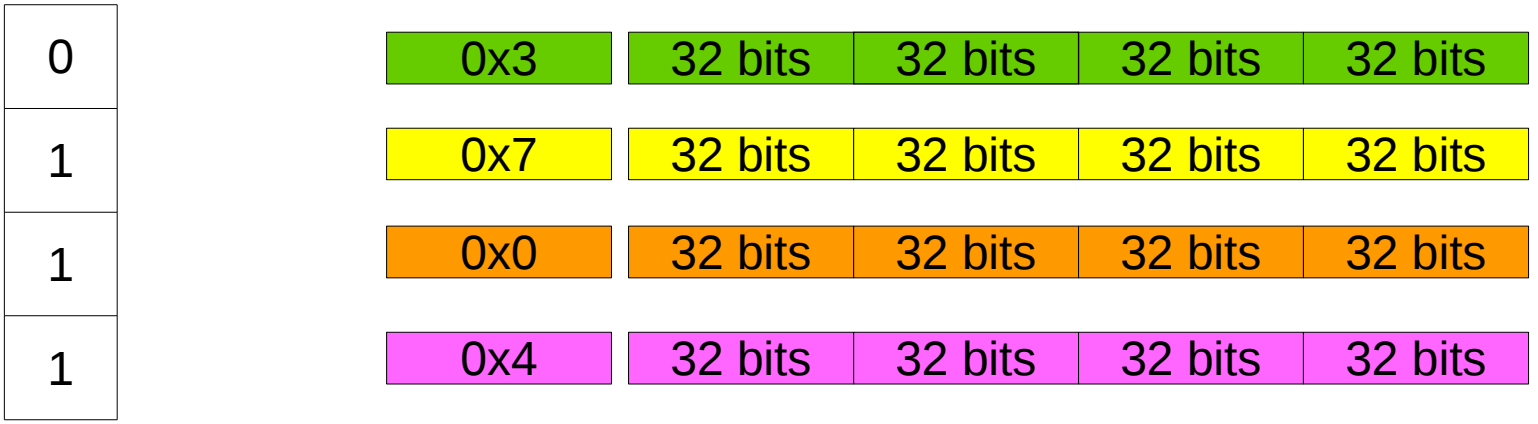


Algorithmes de remplacement

Algorithme de remplacement d'une ligne de type **PLRUm** :

Chaque ligne est associé à un bit. Lorsque la ligne est accédée, en positionne à 1 le bit de la ligne. Lorsque tous les bits sont à 1, on met à 0 tous les bits sauf celui qui vient d'être positionné à 1.

Accès à la ligne 2 :



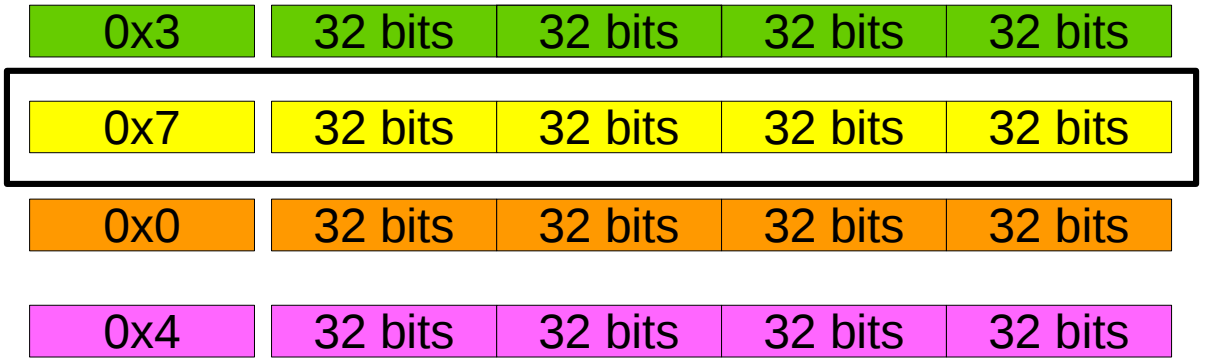
Algorithmes de remplacement

Algorithme de remplacement d'une ligne de type **PLRUm** :

Chaque ligne est associé à un bit. Lorsque la ligne est accédée, en positionne à 1 le bit de la ligne. Lorsque tous les bits sont à 1, on met à 0 tous les bits sauf celui qui vient d'être positionné à 1.

Accès à la ligne 0 :

1
0
0
0



Quand il y a plusieurs lignes à 0, on remplace celle avec l'indice le plus faible. C'est une méthode très populaire.

Benchmark des solutions existantes

Valeur < 1 → Moins de cache miss que LRU

Policy	*Cache size (KB)*	*Ways*	*All benchmarks*			*Representative benchmarks*		
			Hit ratio (%)	*Miss ratio / LRU*	*Average Miss ratio / LRU*	*Hit ratio (%)*	*Miss ratio / LRU*	*Average Miss ratio/ LRU*
1-bit	8	4	96.32	0.987	0.985	97.04	0.984	0.996
	16	4	97.15	0.983		97.90	1.010	
	32	4	98.28	0.999		98.74	0.997	
	64	4	98.56	0.972		99.10	0.991	
LRU	8	4	96.27	1.000	1.000	97.00	1.000	1.000
	16	4	97.10	1.000		97.92	1.000	
	32	4	98.27	1.000		98.74	1.000	
	64	4	98.51	1.000		99.09	1.000	
MPLRU	8	4	96.26	1.002	0.997	97.01	0.994	0.997
	16	4	97.13	0.988		97.94	0.989	
	32	4	98.27	1.001		98.73	1.007	
	64	4	98.52	0.998		99.09	1.000	
PLRUm	8	4	96.34	0.981	0.981	97.07	0.975	0.976
	16	4	97.17	0.977		97.94	0.988	
	32	4	98.30	0.982		98.78	0.968	
	64	4	98.54	0.985		99.11	0.974	
PLRUt	8	4	96.26	1.003	0.999	97.01	0.995	0.998
	16	4	97.13	0.989		97.94	0.988	
	32	4	98.27	1.003		98.73	1.010	
	64	4	98.51	1.000		99.09	1.000	
Random	8	4	96.25	1.004	0.999	96.92	1.024	1.021
	16	4	97.15	0.984		97.84	1.037	
	32	4	98.26	1.007		98.71	1.026	
	64	4	98.52	0.999		99.09	0.998	
Round Robin	8	4	96.18	1.023	1.018	96.86	1.045	1.031
	16	4	96.98	1.042		97.79	1.063	
	32	4	98.27	1.000		98.75	0.990	
	64	4	98.50	1.008		99.06	1.026	

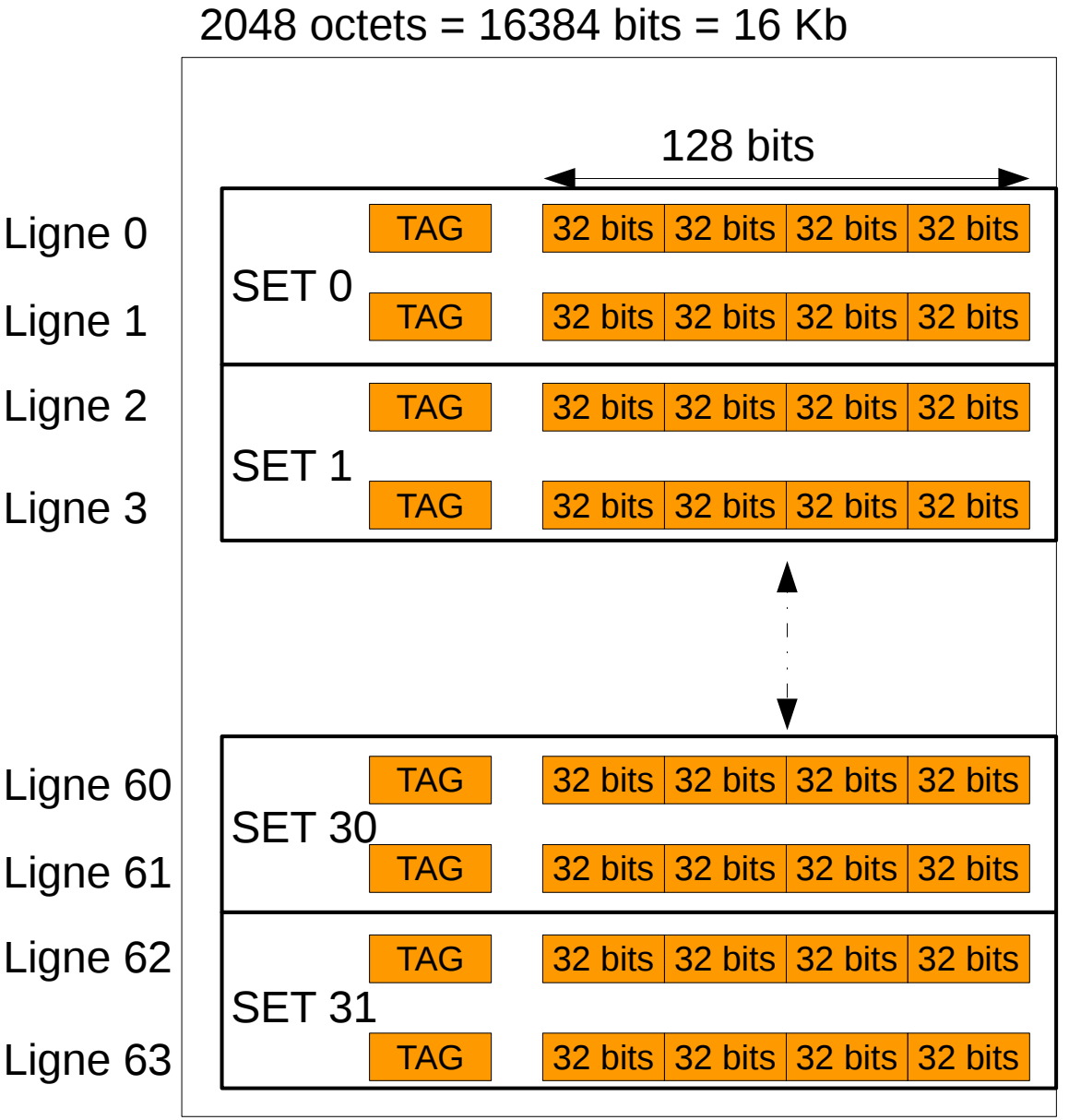
Representative benchmark =

Application usuelle dans le domaine de l'embarqué : Automobile / Télécommunications / Réseau

Thèse GILLES Damien / ARM

Solution classique: Le cache n-associatif

- Solution à mis chemin entre totalement associatif et à correspondance directe.
- Les lignes sont regroupés en « **sets** » contenant ***n*** lignes de cache.
- Il y a correspondance pré-établie pour sélectionner le **set**.
- Au sein d'un set, on utilise une associativité totale.
- Il « suffit » de vérifier seulement les TAG au sein d'un même **set**.



Un peu de vocabulaire

- Cache à correspondance pré-établie → Direct mapped
- Cache totalement associatif → Fully associative
- Cache n-associatif → n-associative / n-Way

Politique d’écriture dans la mémoire de niveau supérieure

- Write-through :
  - Dès qu’une donnée en cache est modifiée, elle est remontée à la mémoire centrale. Cette politique est utilisée pour le cache d’instruction.
- Write-back :
  - Remontée à la mémoire centrale lorsque la donnée est invalidée en cache (c’est l’exemple que je vous ai présenté). C’est le cas le plus courant car limite les accès mémoire.

Exemple Intel Core i7 – 3820 CPU

L1 cache :

- instruction cache 32KB
  8-way write-through per core
- data cache 32KB
  8-way write-back per core
- cache latency : 3 clock cycles

– L2 cache :

- 256KB **8-way write-back** unified cache **per core**
- Cache latency : 12 clock cycles

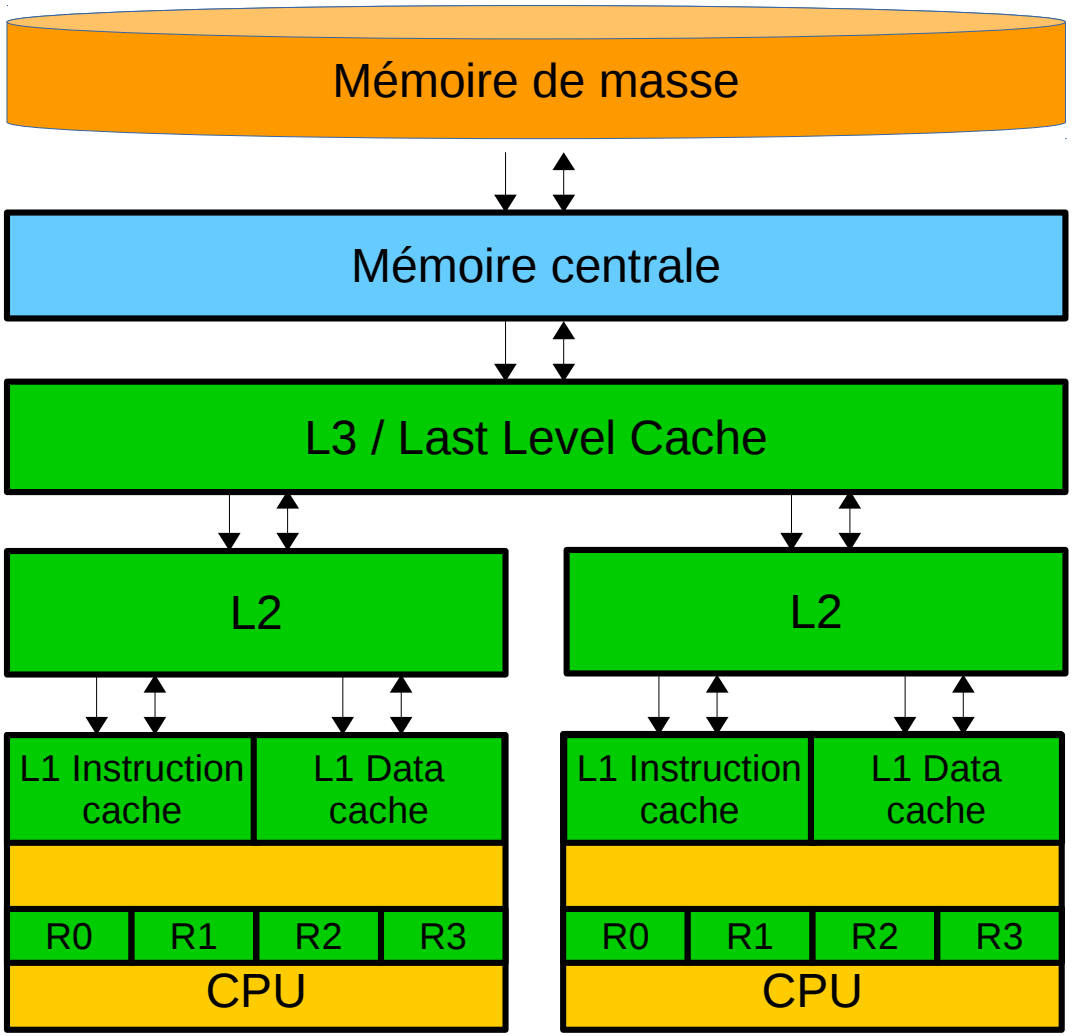
– L3 cache :

- 10MB **20-way write-back** unified cache **shared by all cores**
- Cache latency : 26-31 clock cycles

– TLBs :

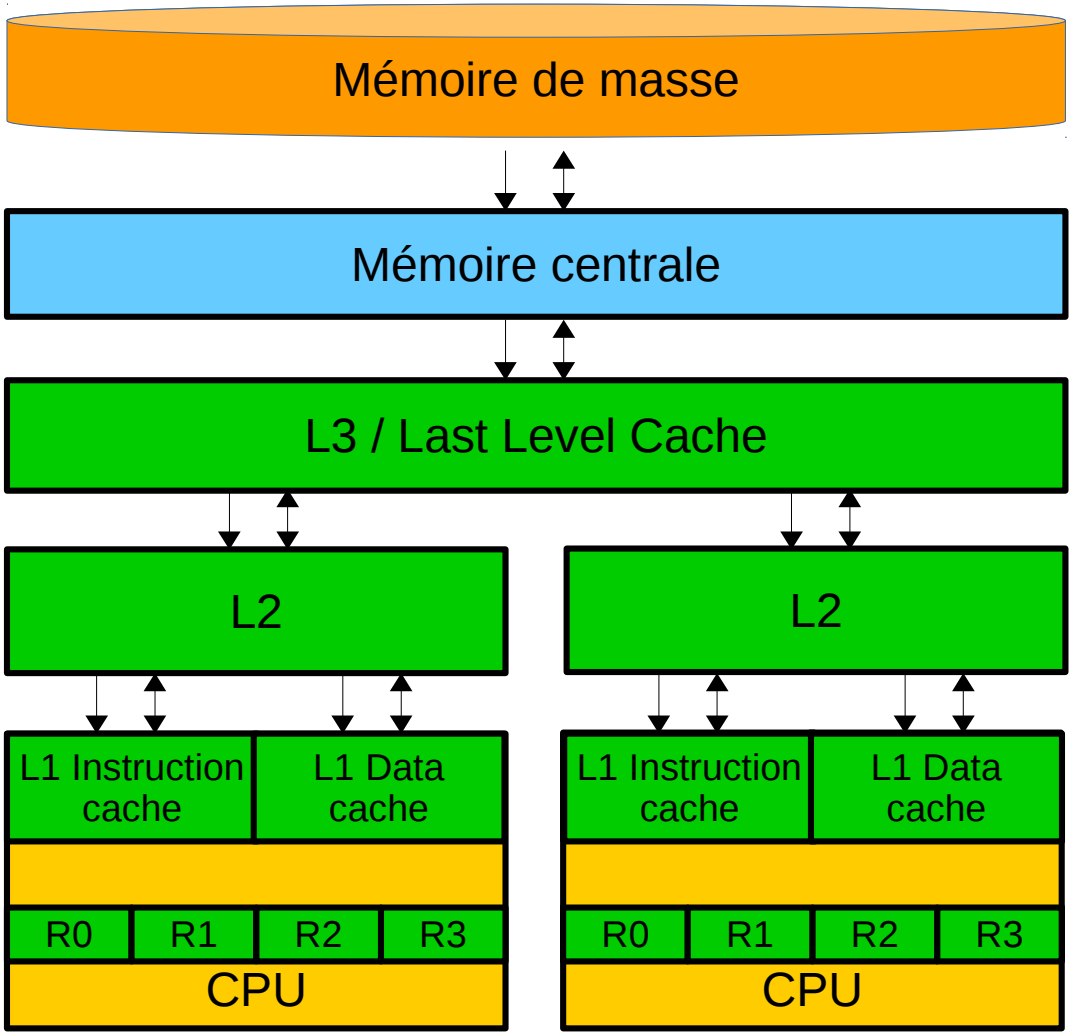
- L1 : 4K pages, 64 entries, 4-way
- L2 : 2K pages, 512 entries, 4-way

– Caches + TLBs uses pseudo LRU replacement policy

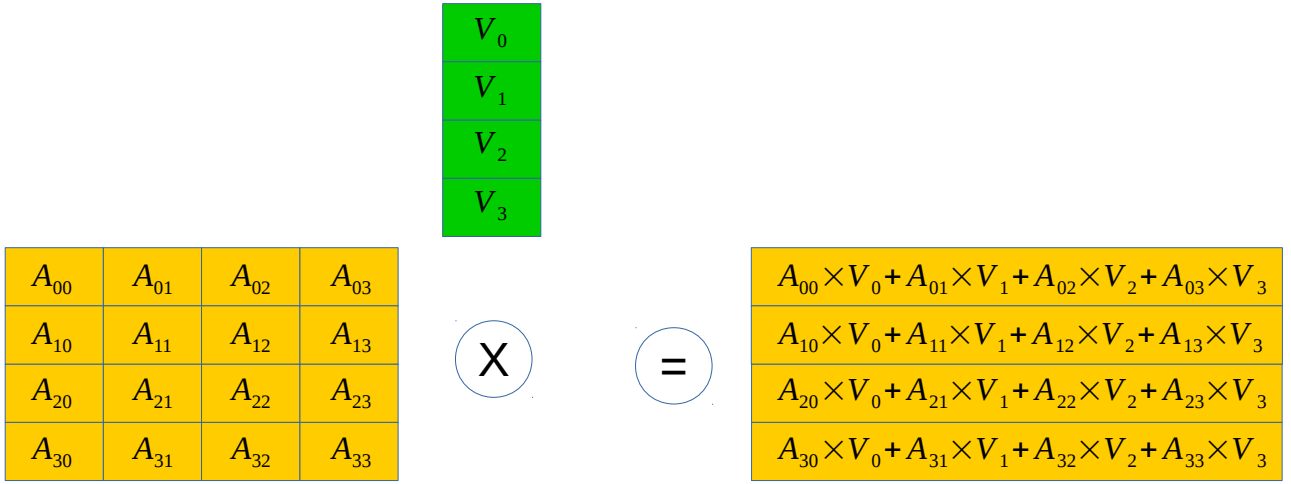


Temps d’accès selon le niveau de la hiérarchie mémoire

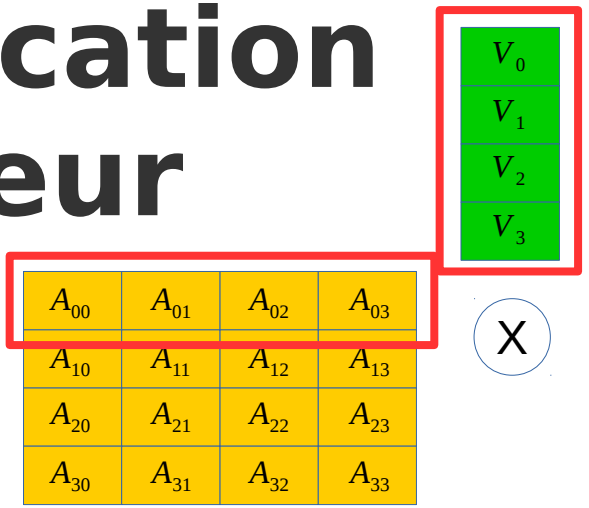
Temps d’accès	Taille
16 µs	250 Go~ 2 To
100 ns	1 Go ~ 16 Go
21 ns	10 Mo
7 ns	256 Ko
0.5 ns	32 Ko
0 ns ?	32/64 bits



Application : Multiplication Matrice-vecteur

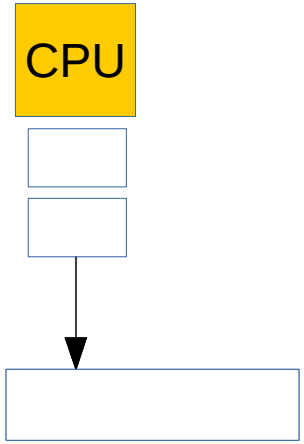


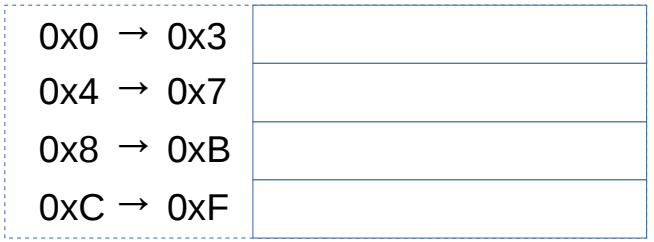
Exemple : Multiplication matrice - vecteur

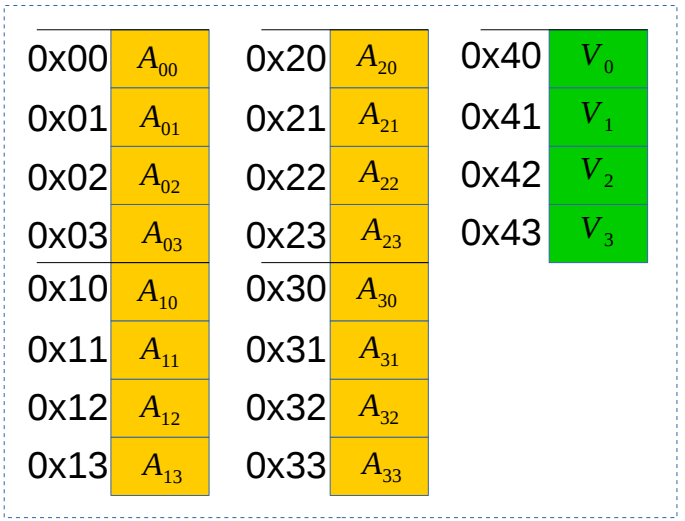


Direct mapped

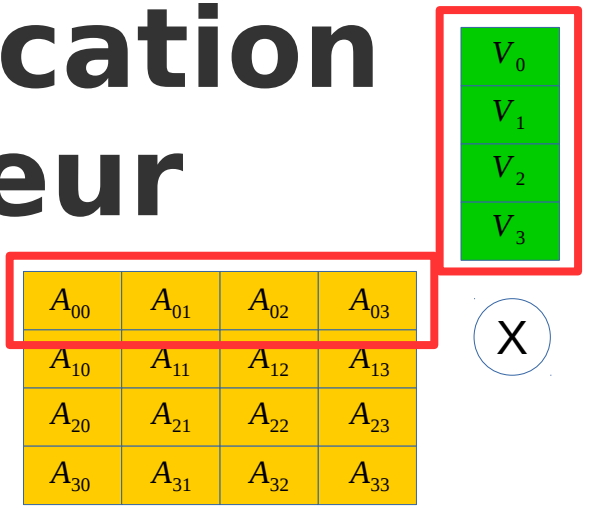
- Cache miss : 0
- Replacement : 0





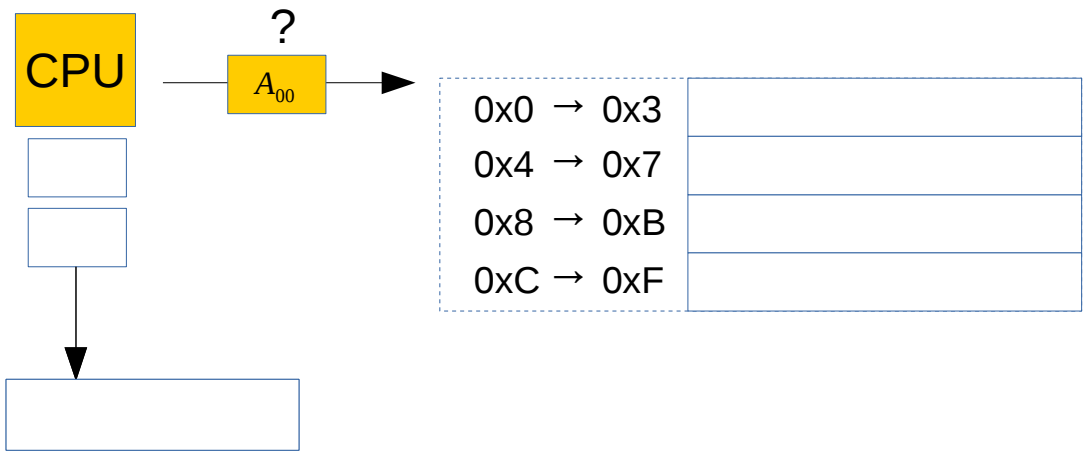


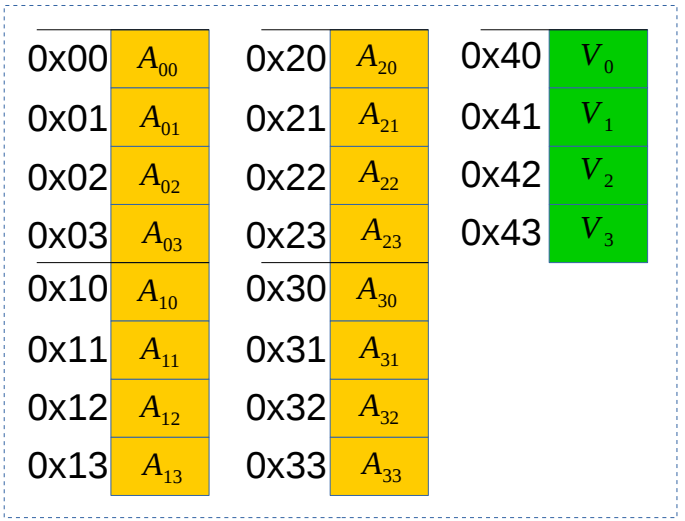
Exemple : Multiplication matrice - vecteur



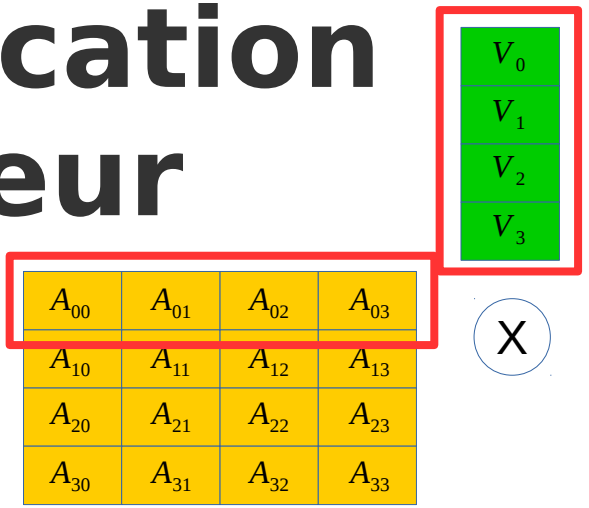
Direct mapped

- Cache miss : 0
- Replacement : 0



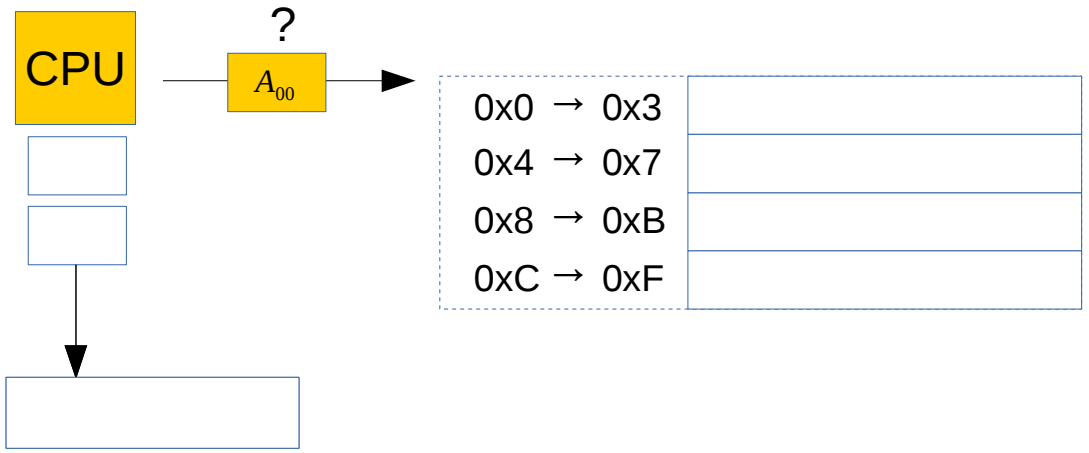


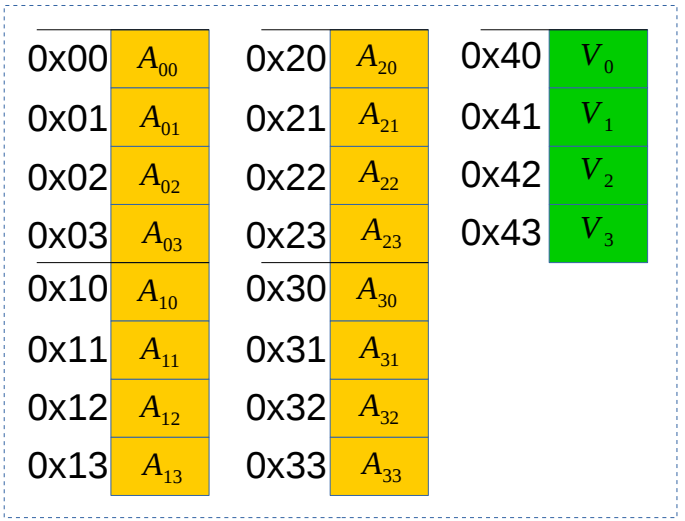
Exemple : Multiplication matrice - vecteur



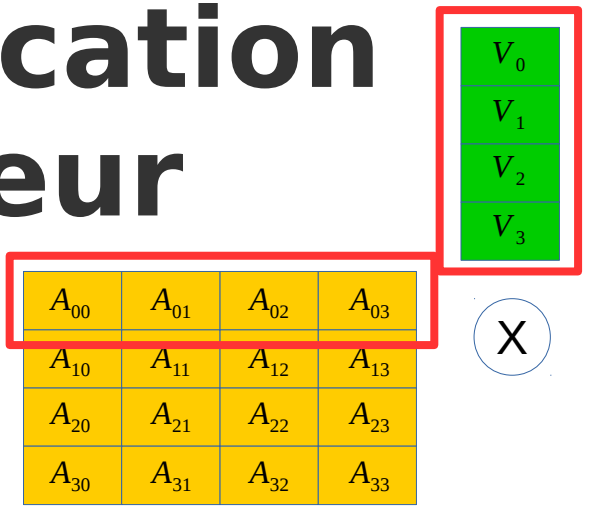
Direct mapped

- Cache miss : 1
- Replacement : 0



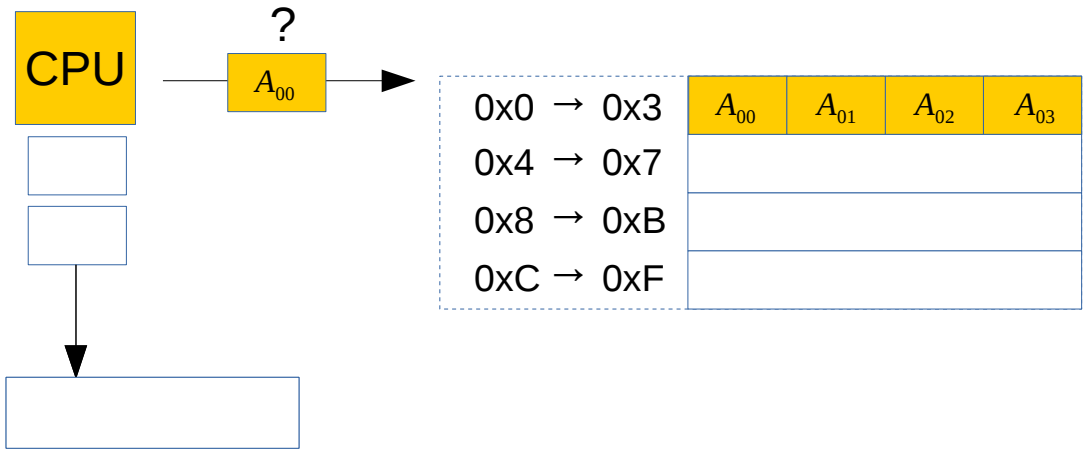


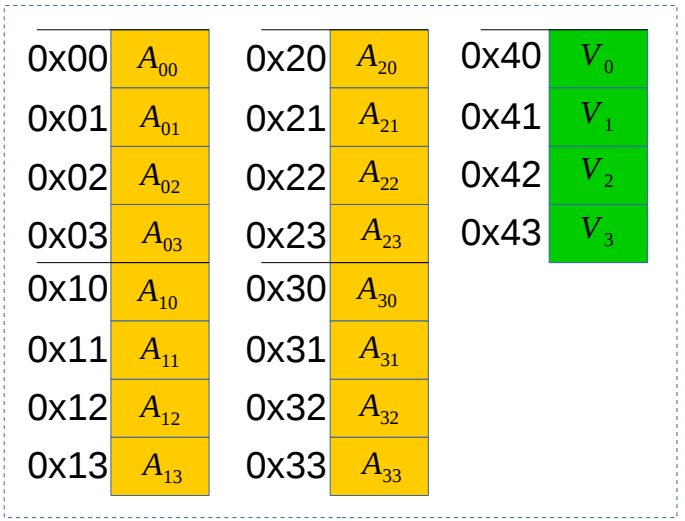
Exemple : Multiplication matrice - vecteur



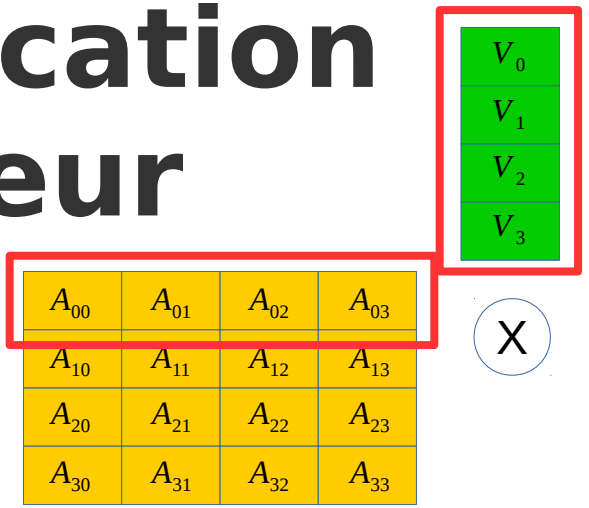
Direct mapped

- Cache miss : 1
- Replacement : 0



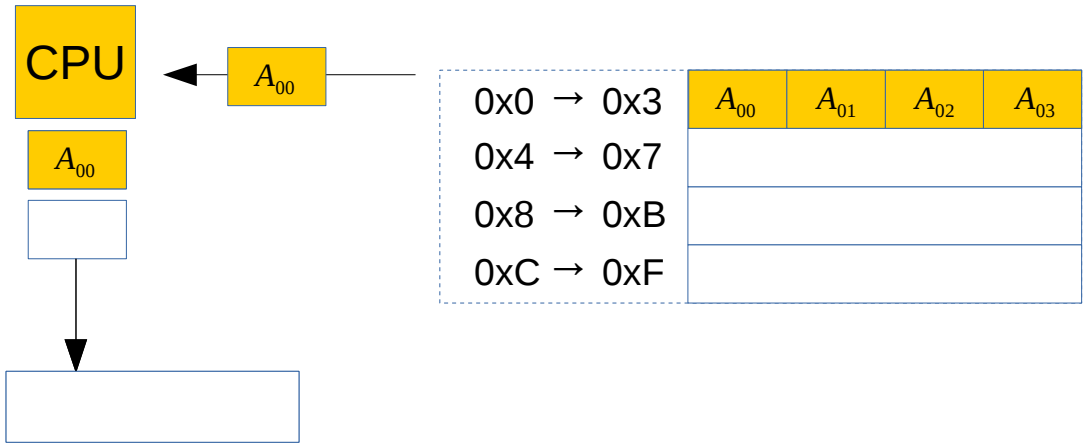


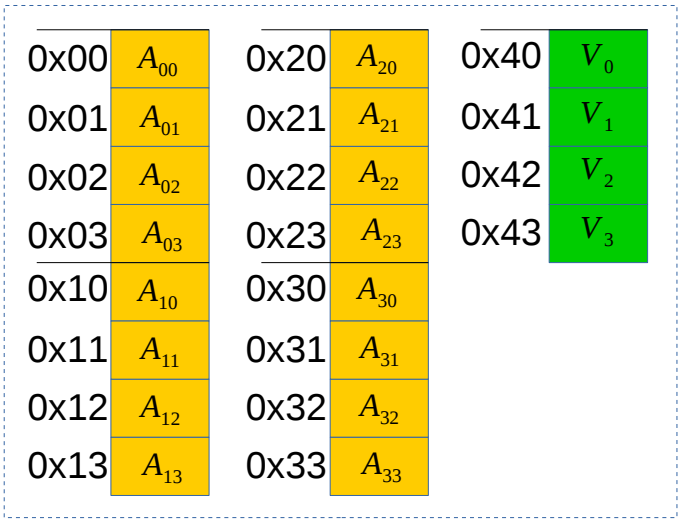
Exemple : Multiplication matrice - vecteur



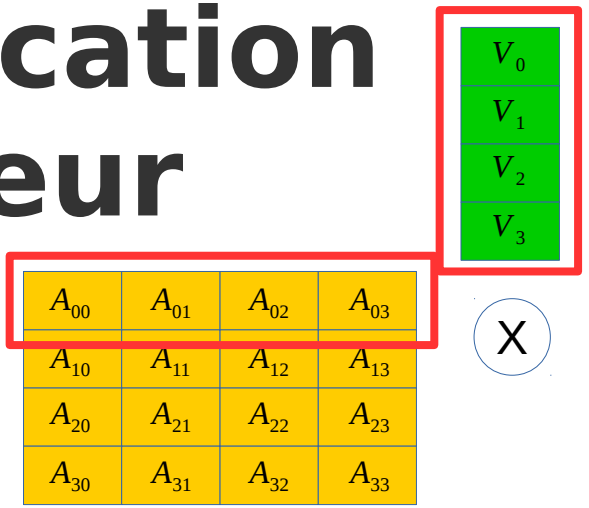
Direct mapped

- Cache miss : 1
- Replacement : 0



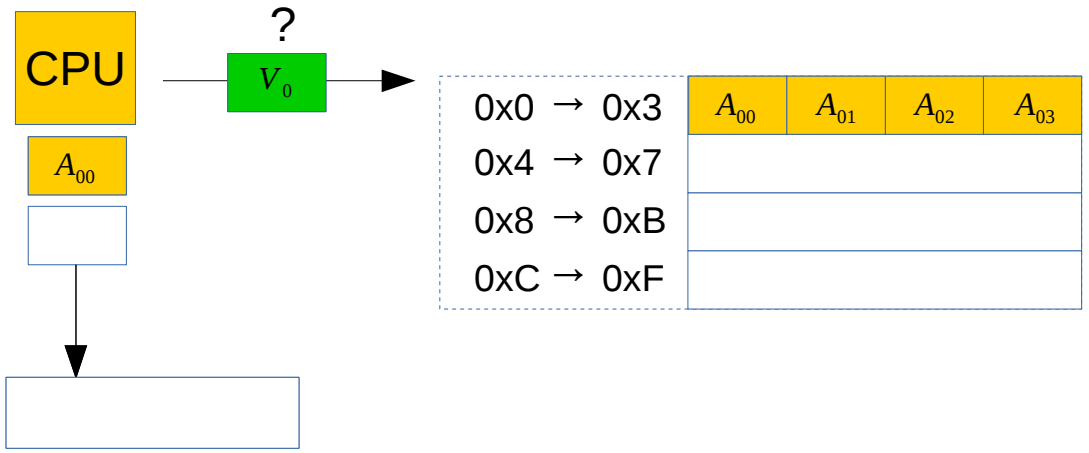


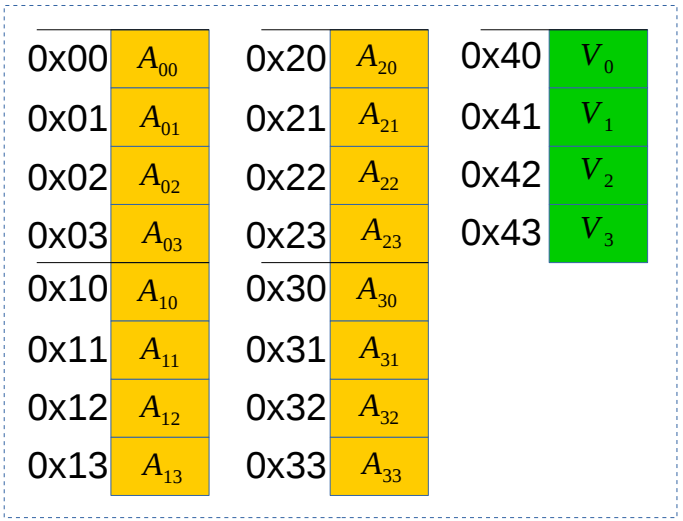
Exemple : Multiplication matrice - vecteur



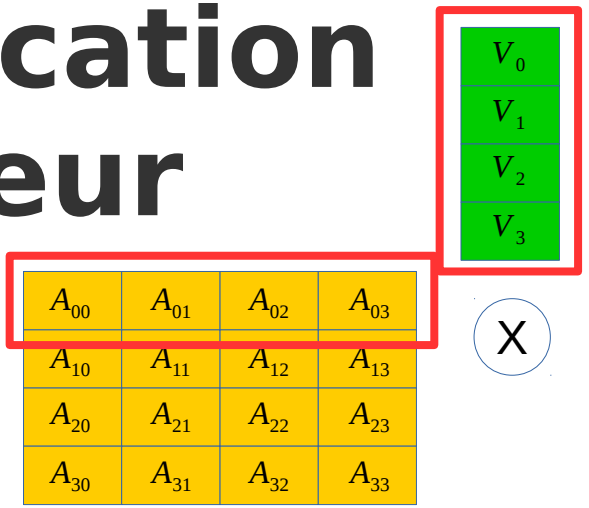
Direct mapped

- Cache miss : 1
- Replacement : 0



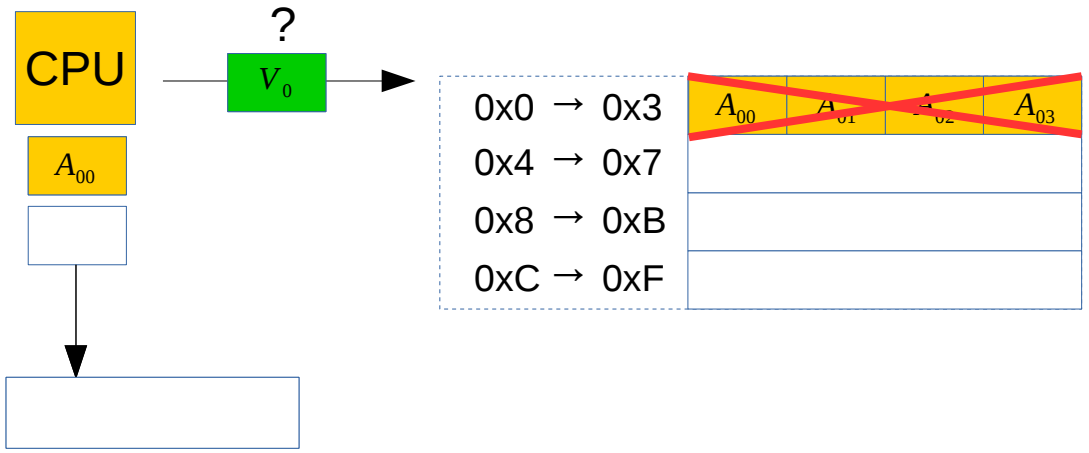


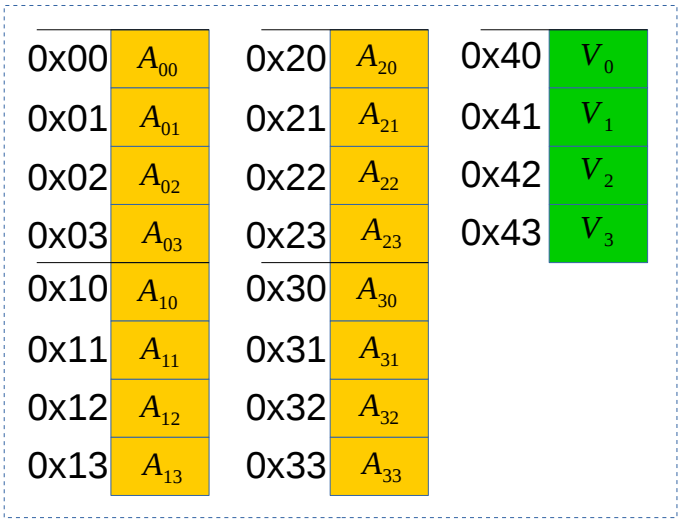
Exemple : Multiplication matrice - vecteur



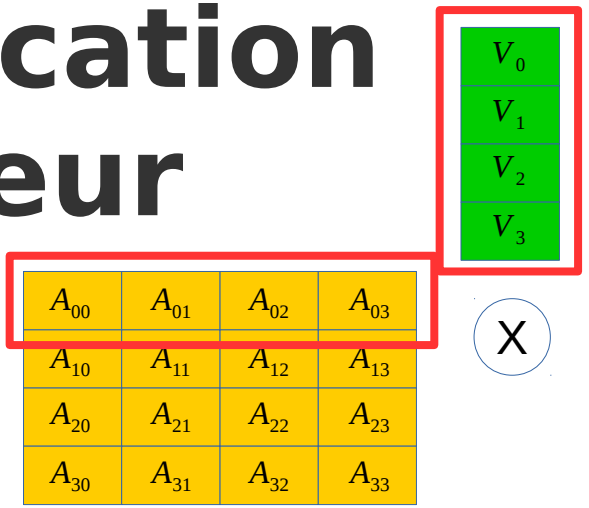
Direct mapped

- Cache miss : 2
- Replacement : 1



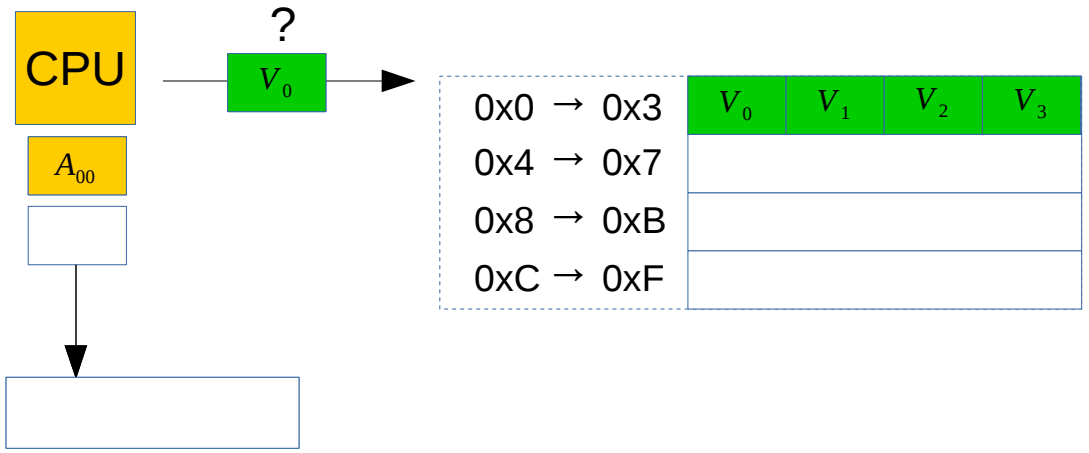


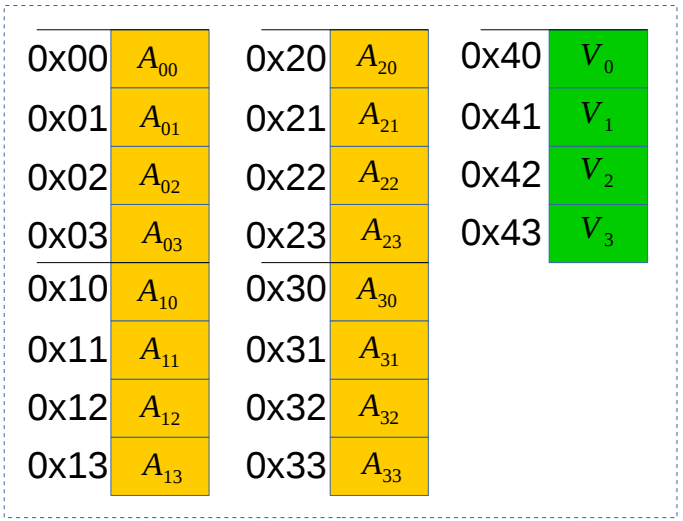
Exemple : Multiplication matrice - vecteur



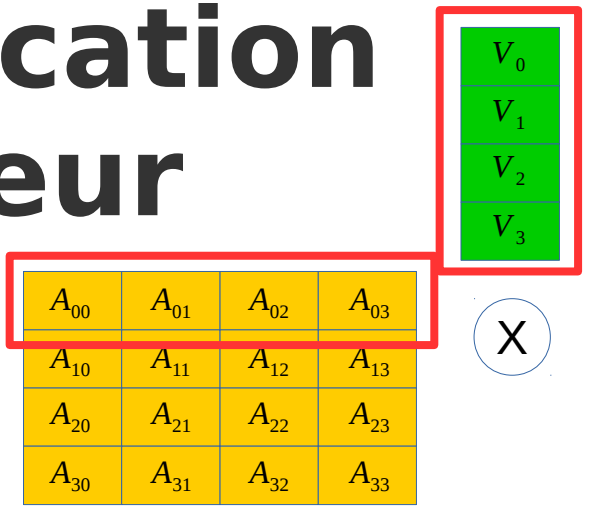
Direct mapped

- Cache miss : 2
- Replacement : 1



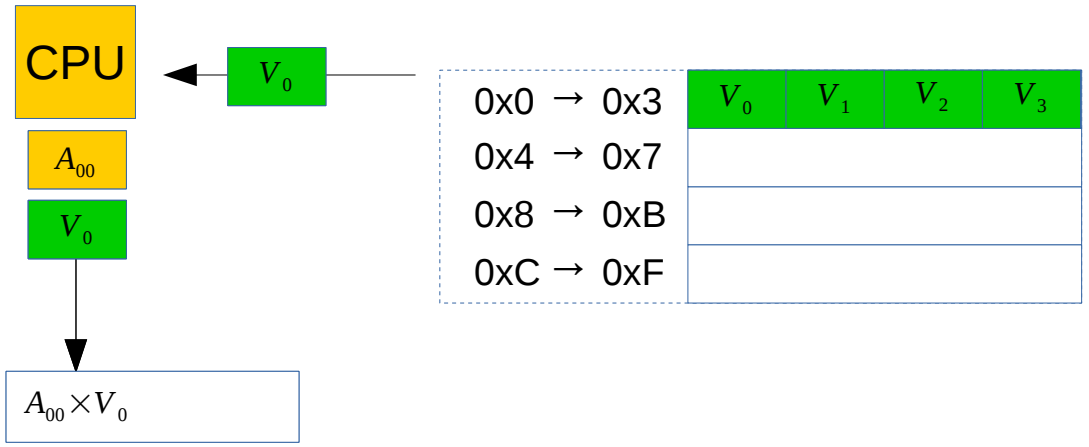


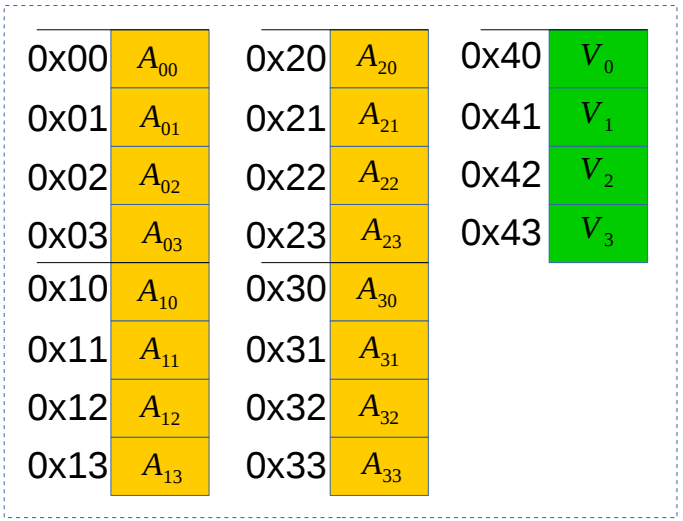
Exemple : Multiplication matrice - vecteur



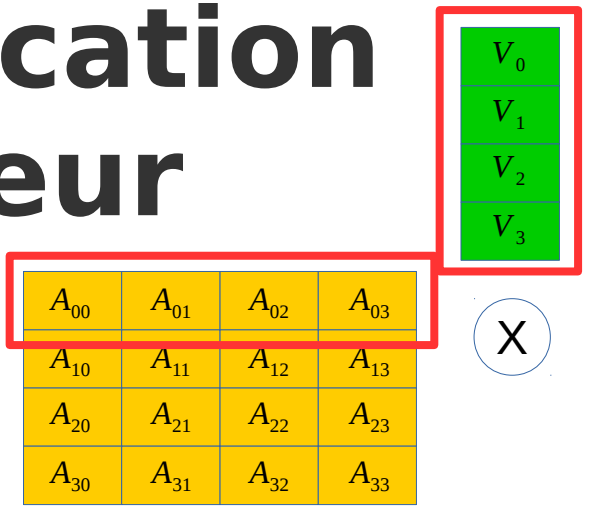
Direct mapped

- Cache miss : 2
- Replacement : 1



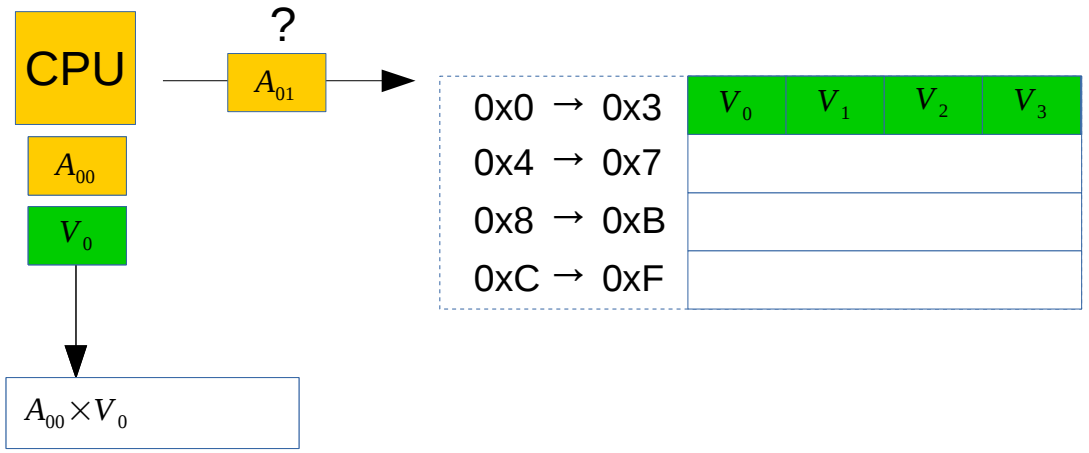


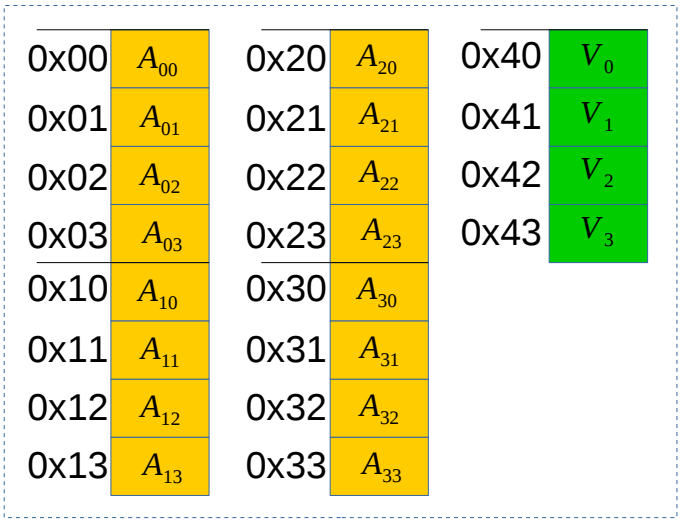
Exemple : Multiplication matrice - vecteur



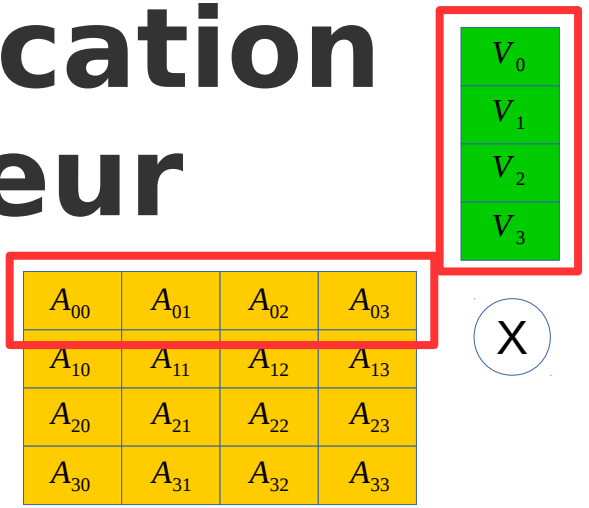
Direct mapped

- Cache miss : 2
- Replacement : 1



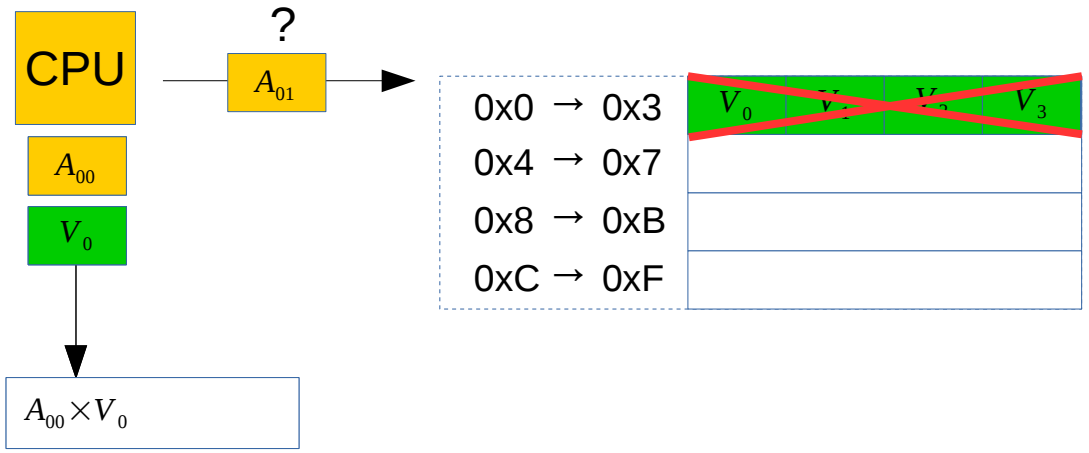


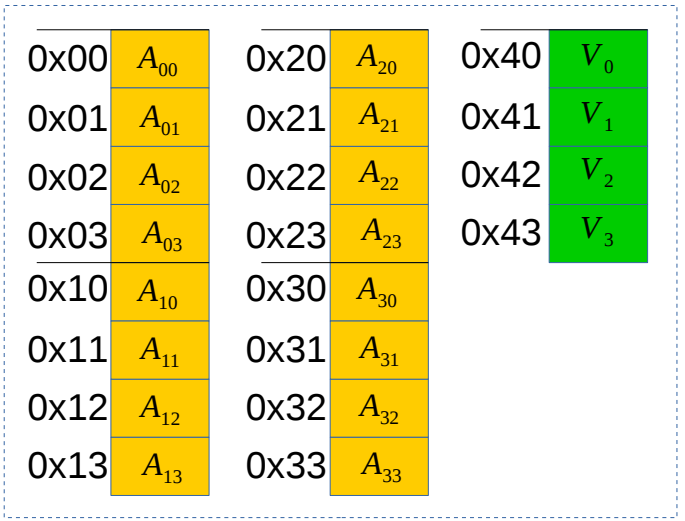
Exemple : Multiplication matrice - vecteur



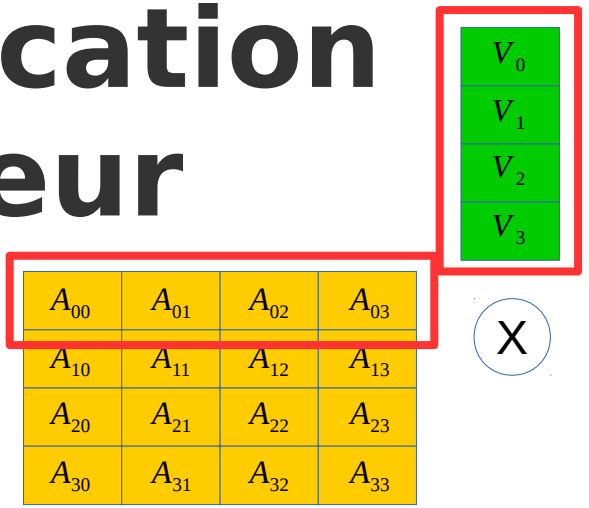
Direct mapped

- Cache miss : 3
- Replacement : 2



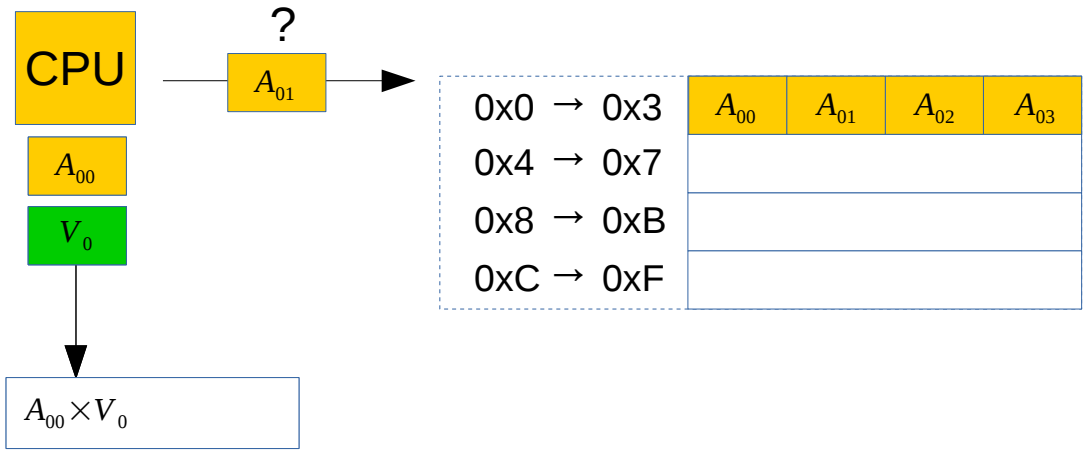


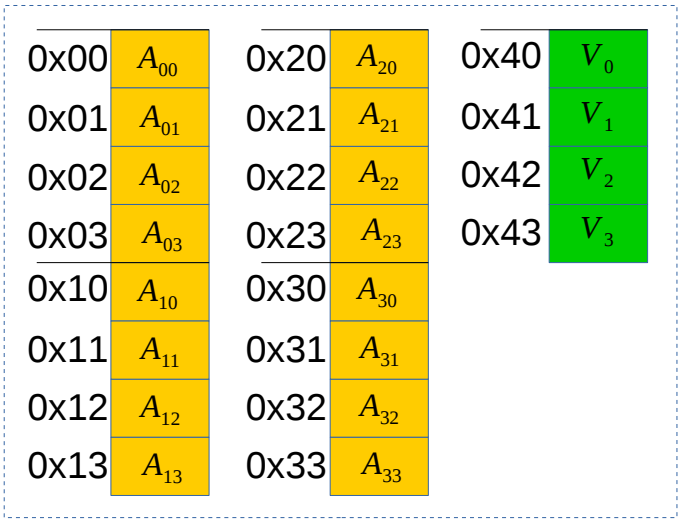
Exemple : Multiplication matrice - vecteur



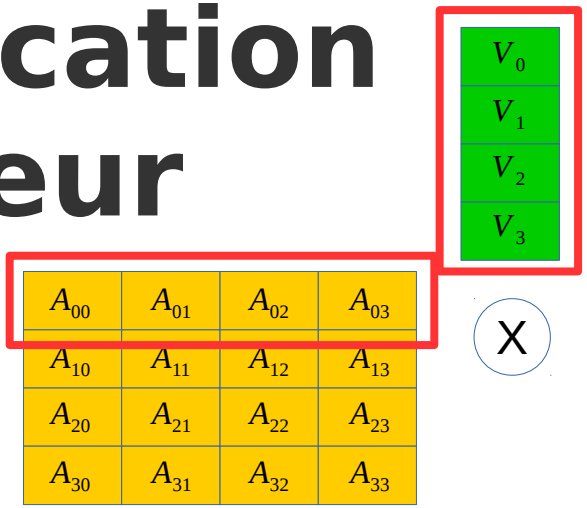
Direct mapped

- Cache miss : 3
- Replacement : 2



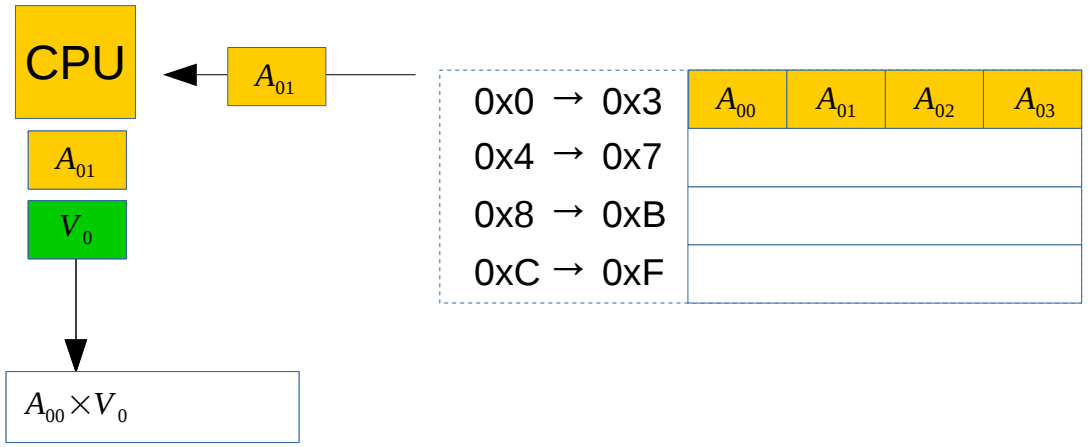


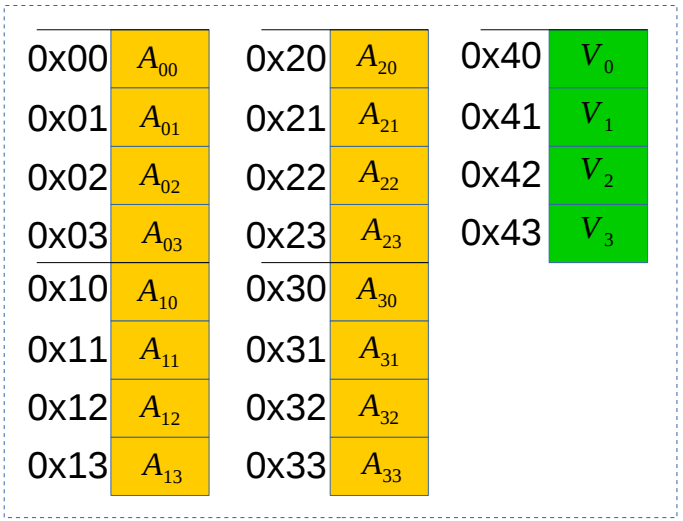
Exemple : Multiplication matrice - vecteur



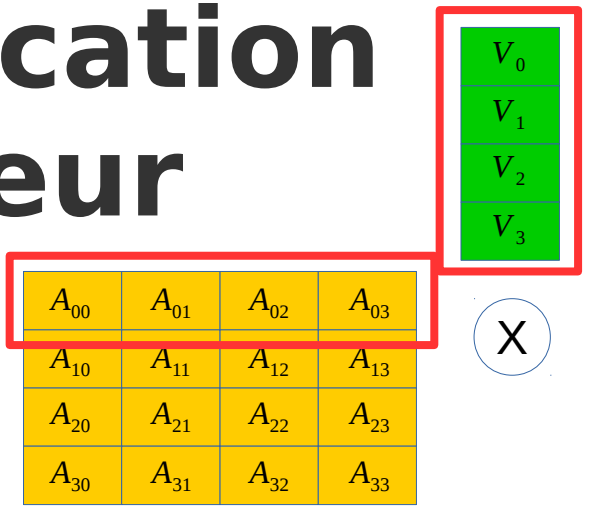
Direct mapped

- Cache miss : 3
- Replacement : 2



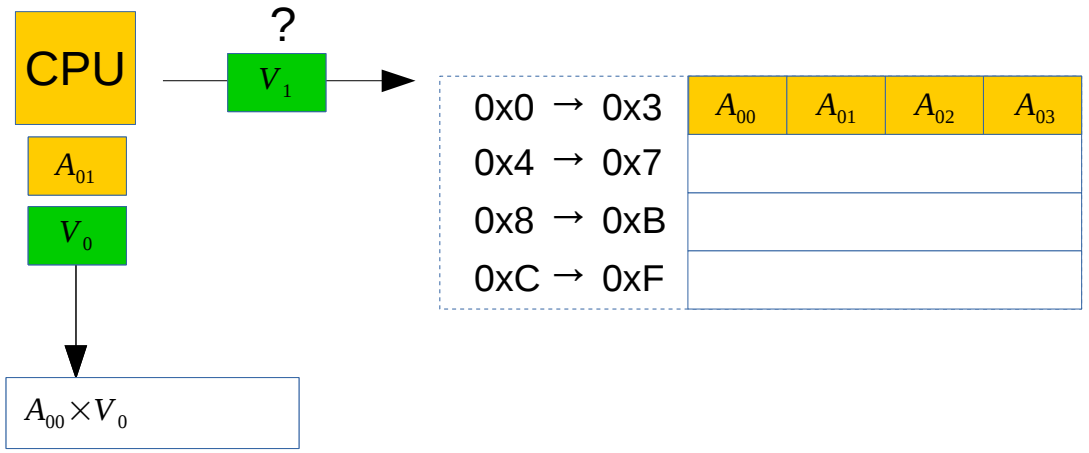


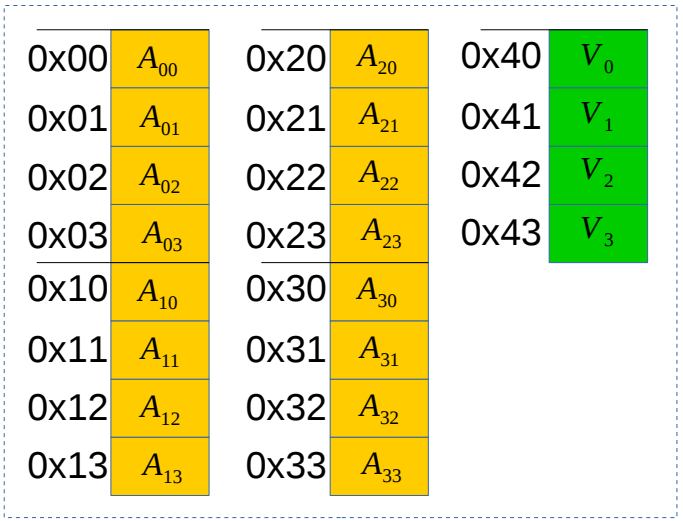
Exemple : Multiplication matrice - vecteur



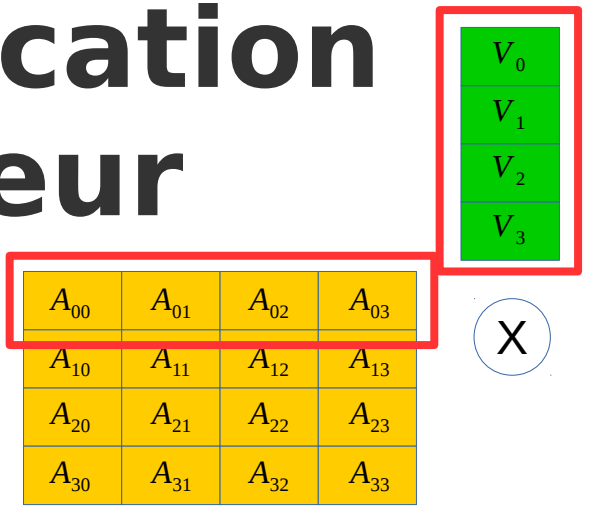
Direct mapped

- Cache miss : 3
- Replacement : 2



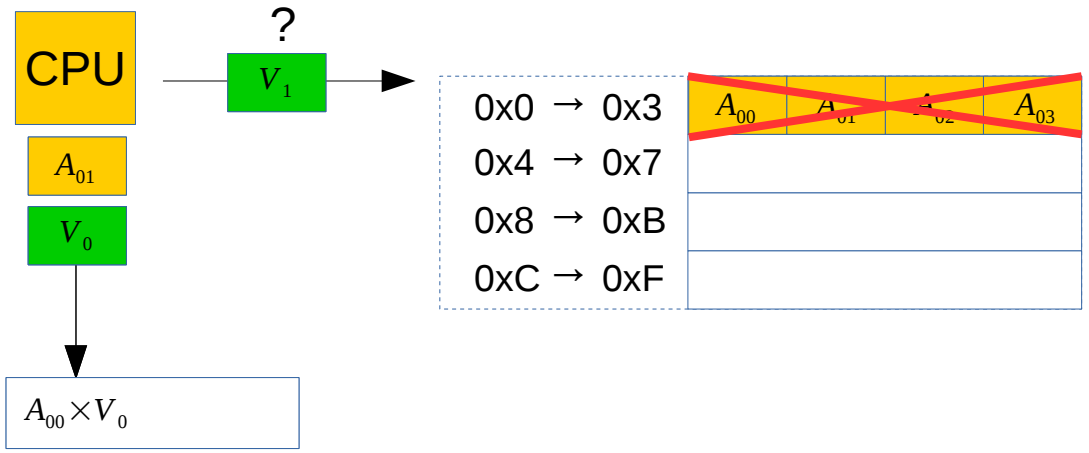


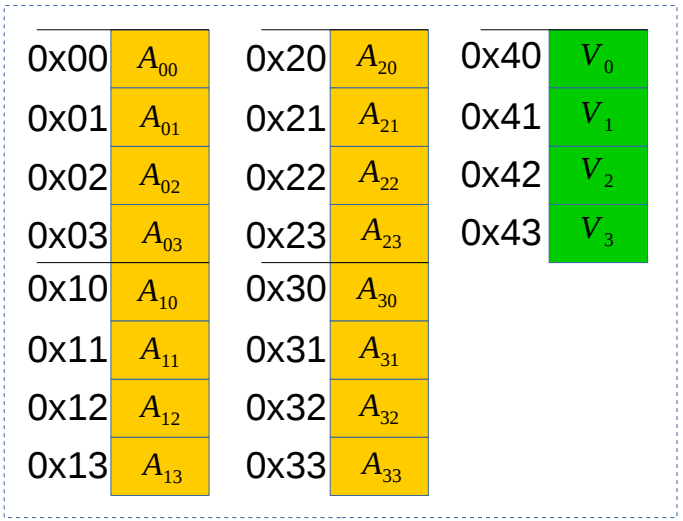
Exemple : Multiplication matrice - vecteur



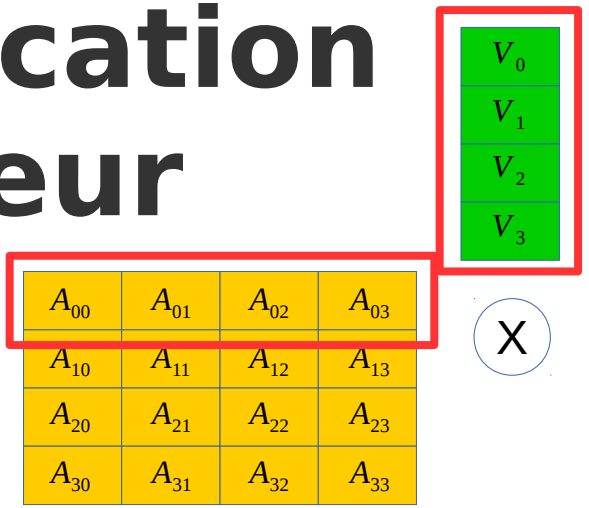
Direct mapped

- Cache miss : 4
- Replacement : 3



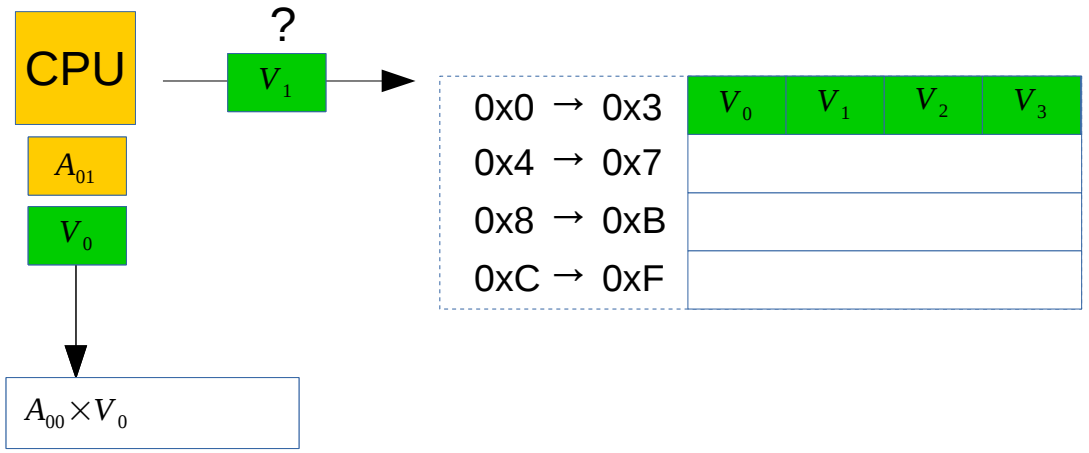


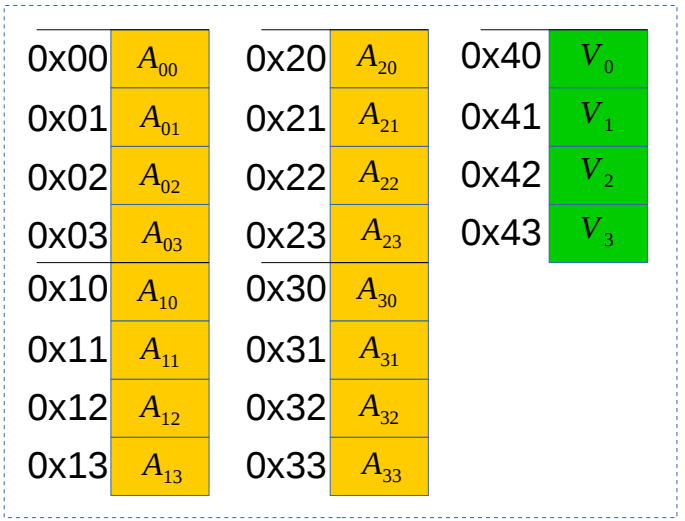
Exemple : Multiplication matrice - vecteur



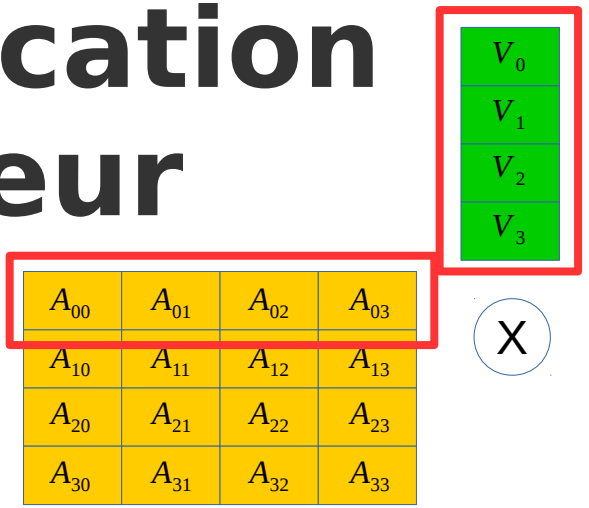
Direct mapped

- Cache miss : 4
- Replacement : 3



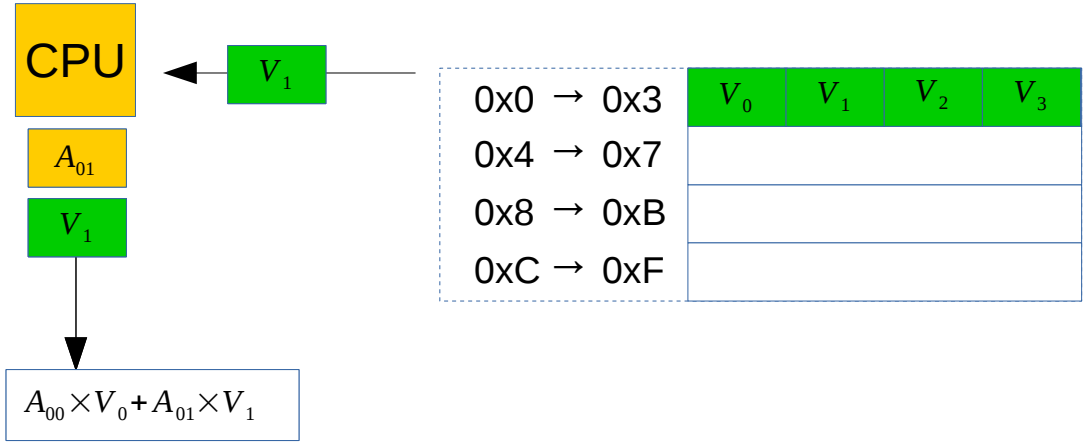


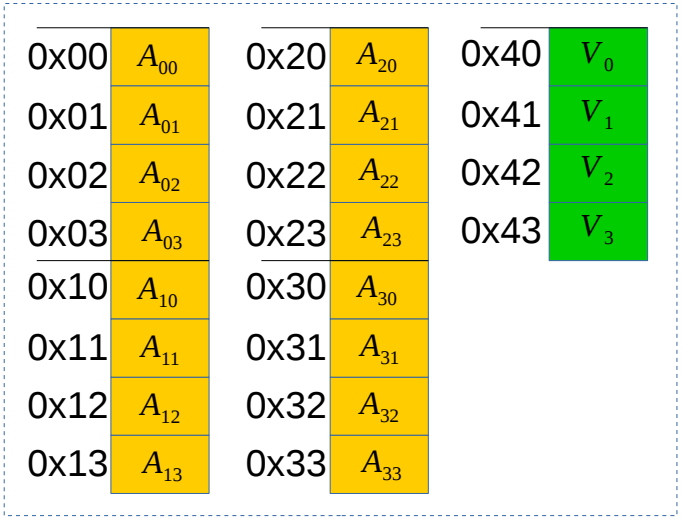
Exemple : Multiplication matrice - vecteur



Direct mapped

- Cache miss : 4
- Replacement : 3

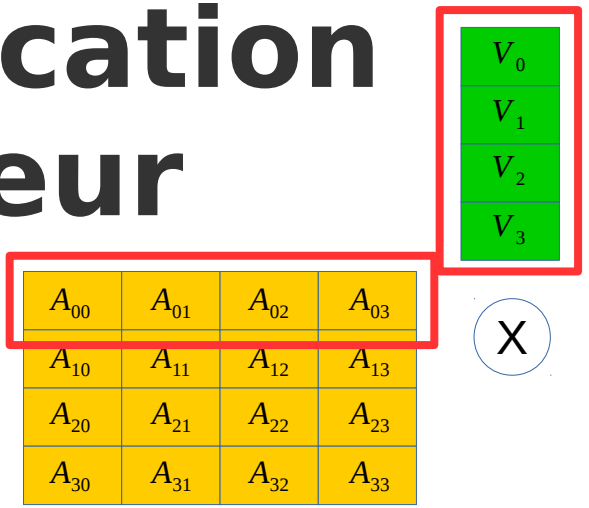




Exemple : Multiplication matrice - vecteur

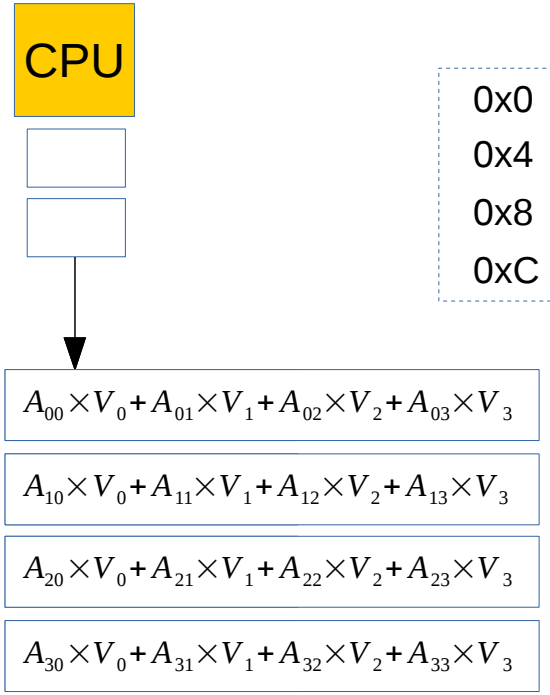
Quelques opérations plus tard ...

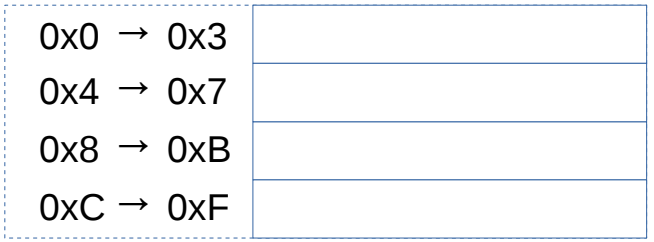
Exemple : Multiplication matrice - vecteur

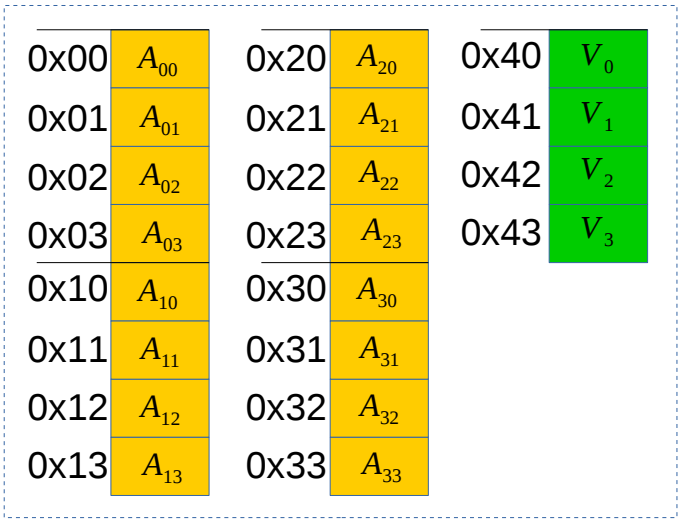


Direct mapped

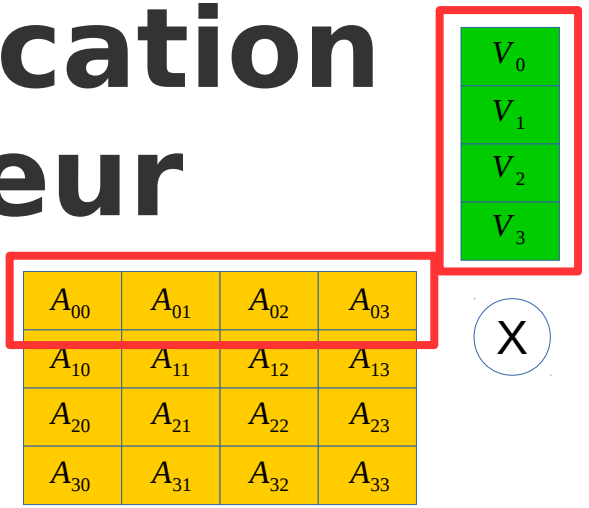
- Cache miss : 32
- Replacement : 31





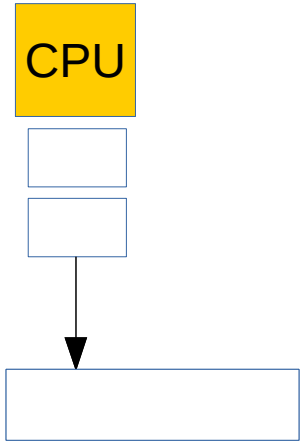


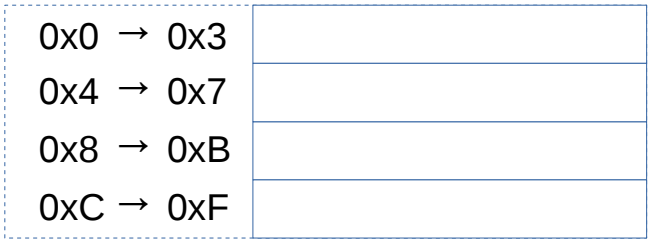
Exemple : Multiplication matrice - vecteur

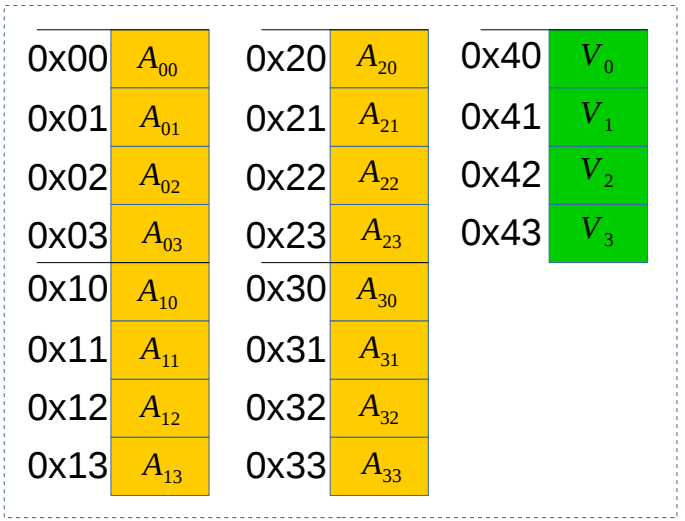


Fully associative

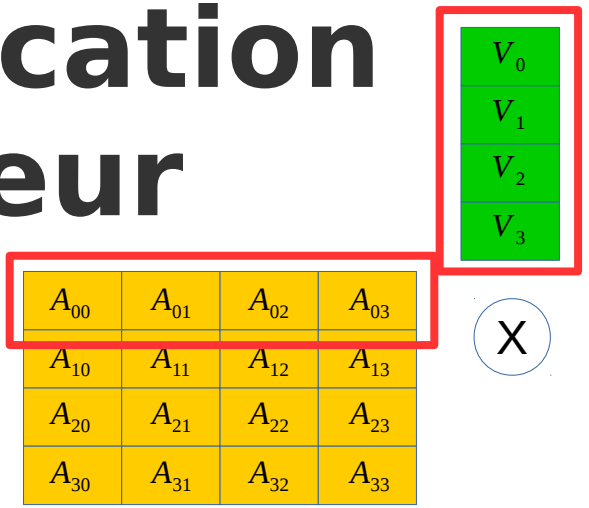
- Cache miss : 0
- Replacement : 0





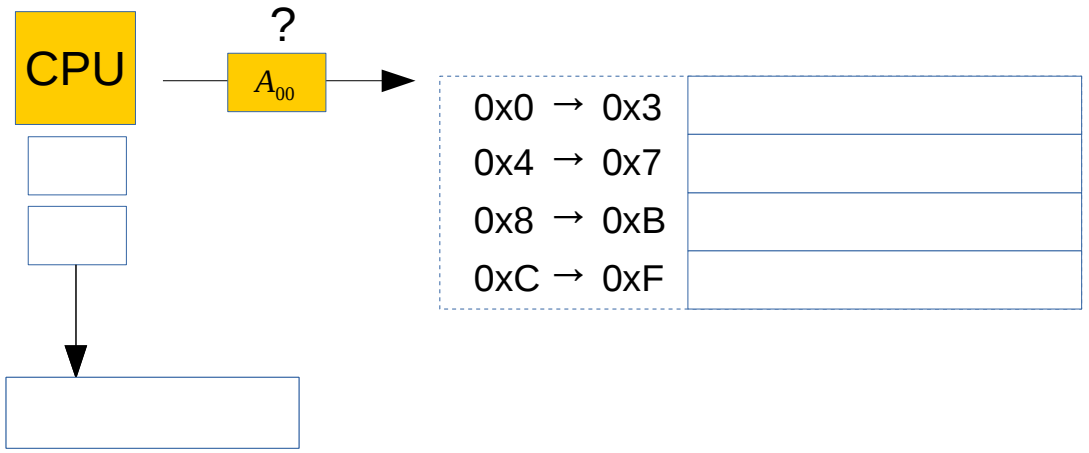


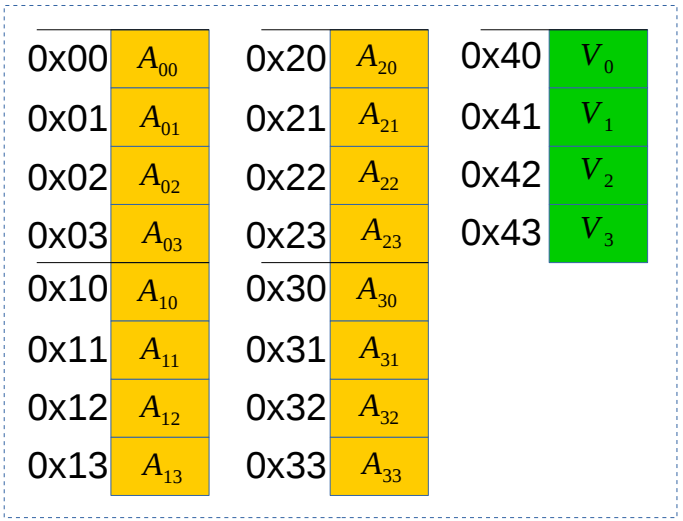
Exemple : Multiplication matrice - vecteur



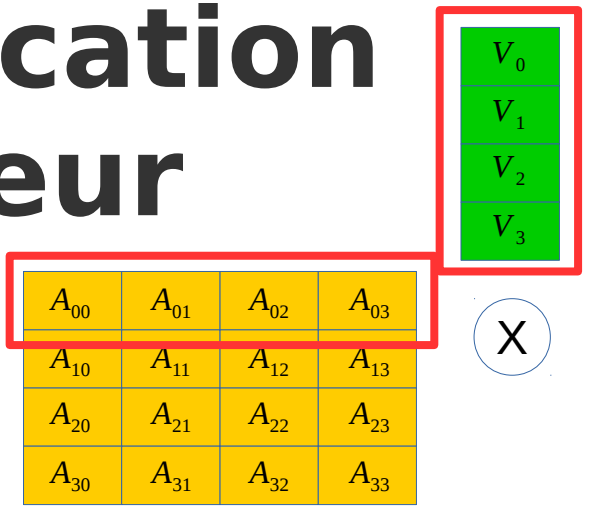
Fully associative

- Cache miss : 0
- Replacement : 0



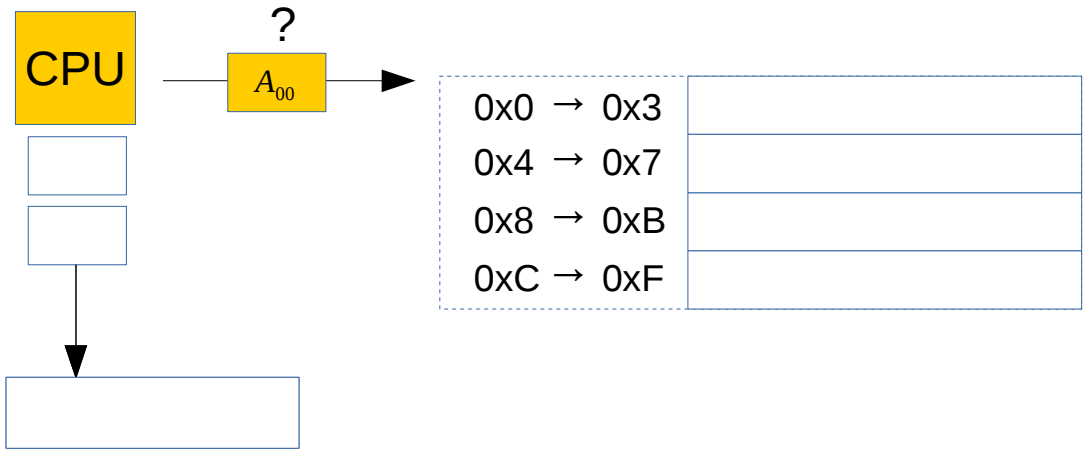


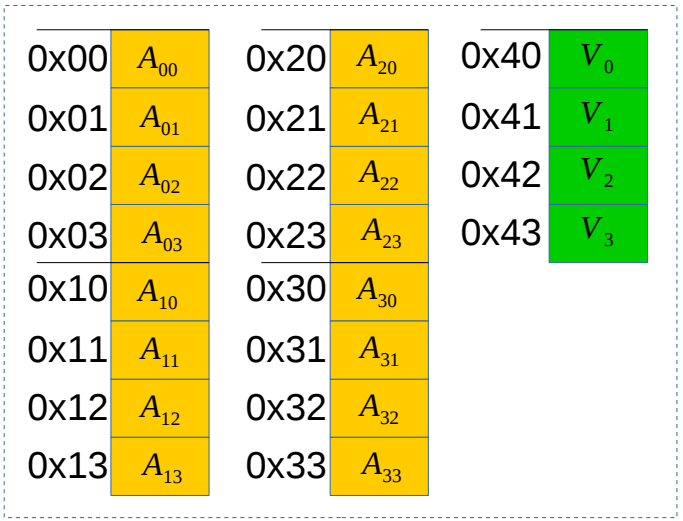
Exemple : Multiplication matrice - vecteur



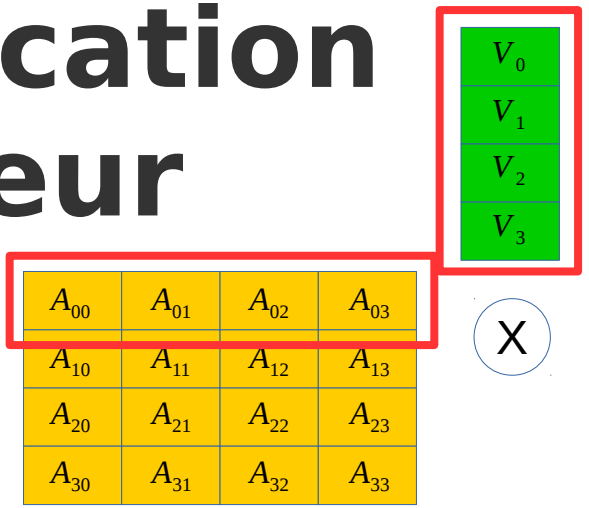
Fully associative

- Cache miss : 1
- Replacement : 0



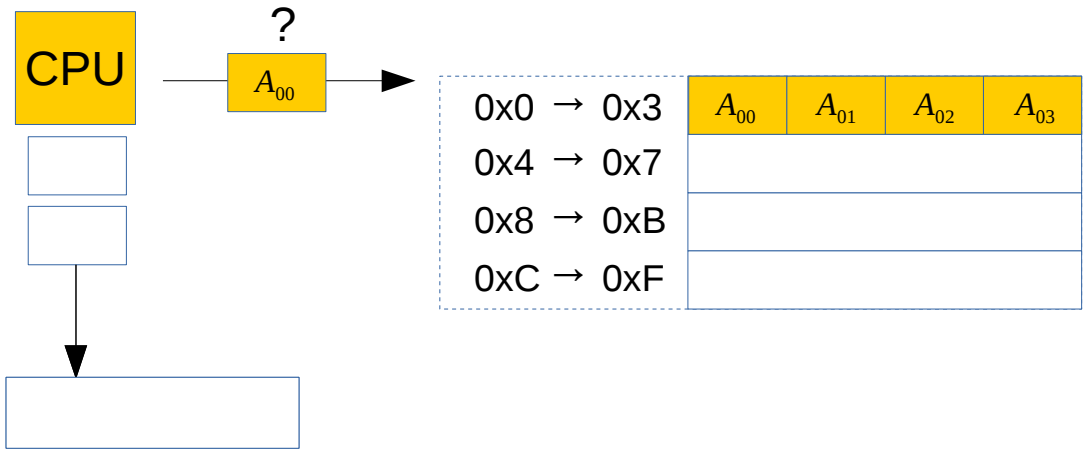


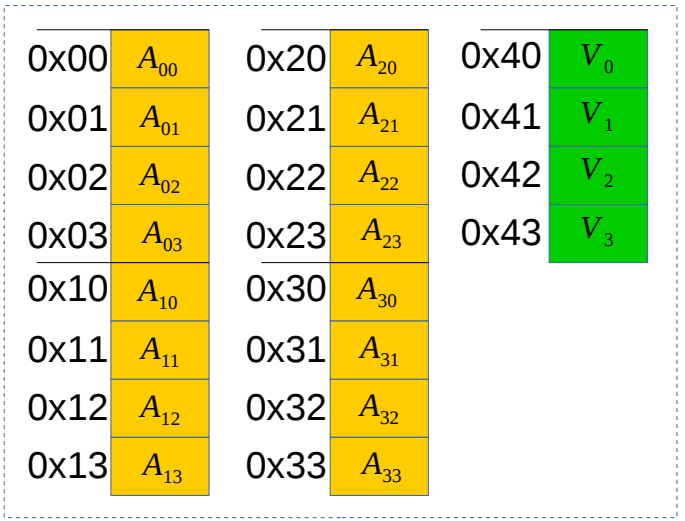
Exemple : Multiplication matrice - vecteur



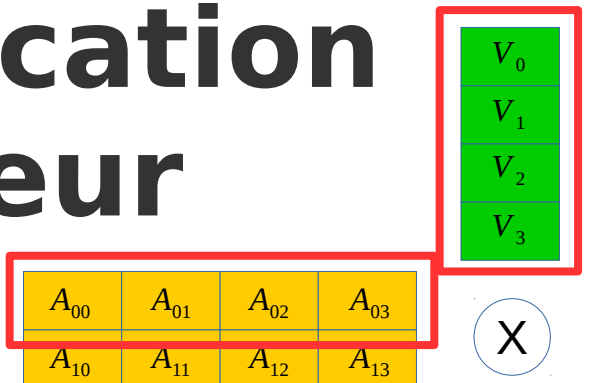
Fully associative

- Cache miss : 1
- Replacement : 0



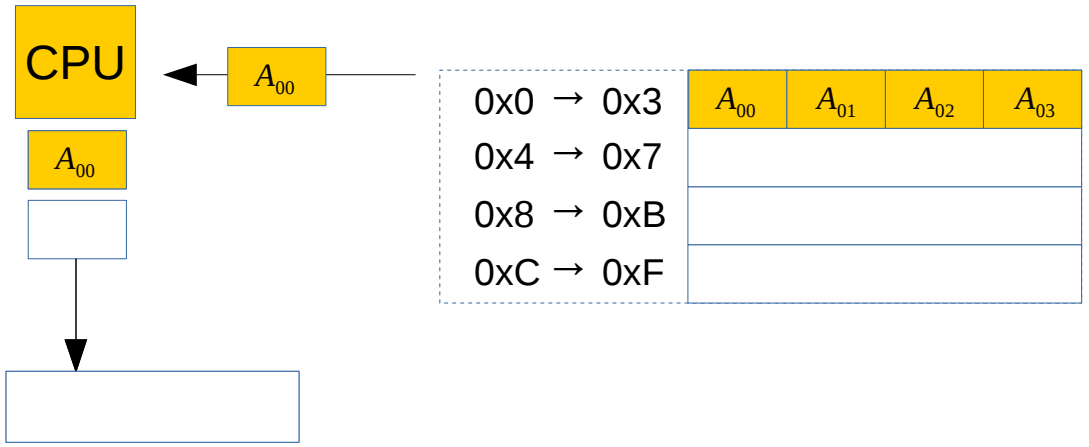


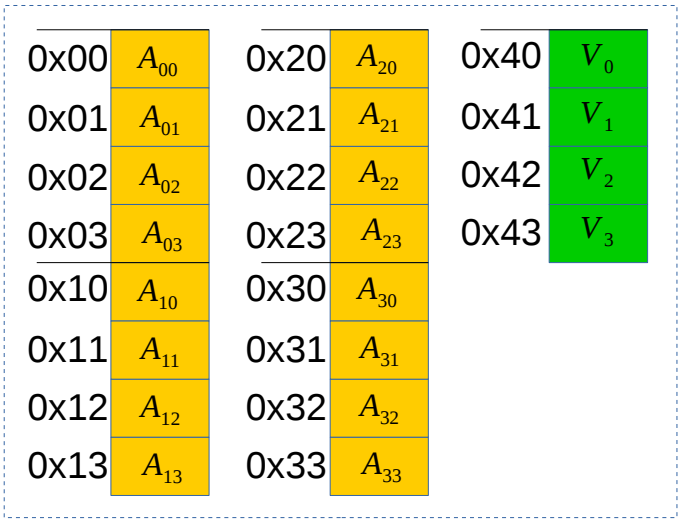
Exemple : Multiplication matrice - vecteur



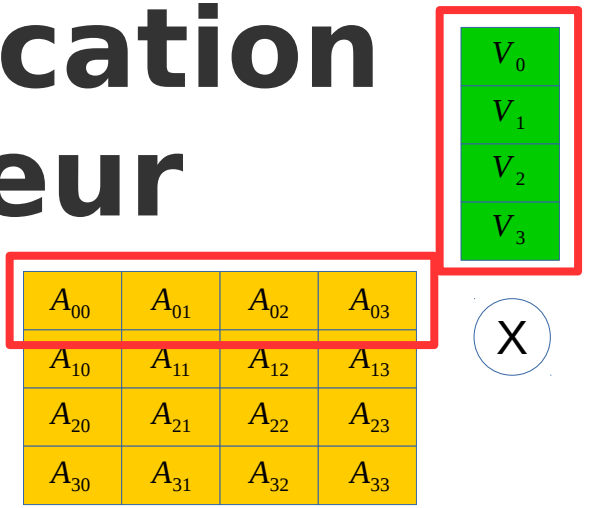
Fully associative

- Cache miss : 1
- Replacement : 0



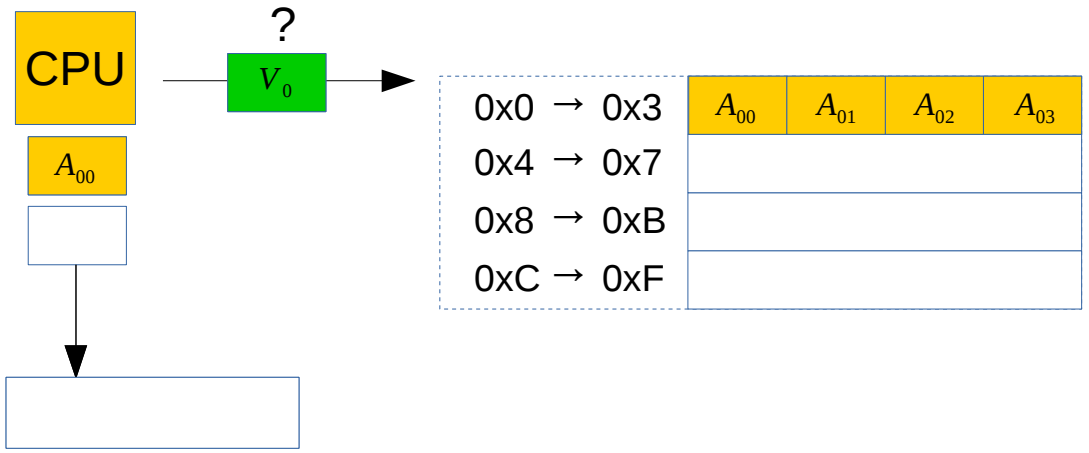


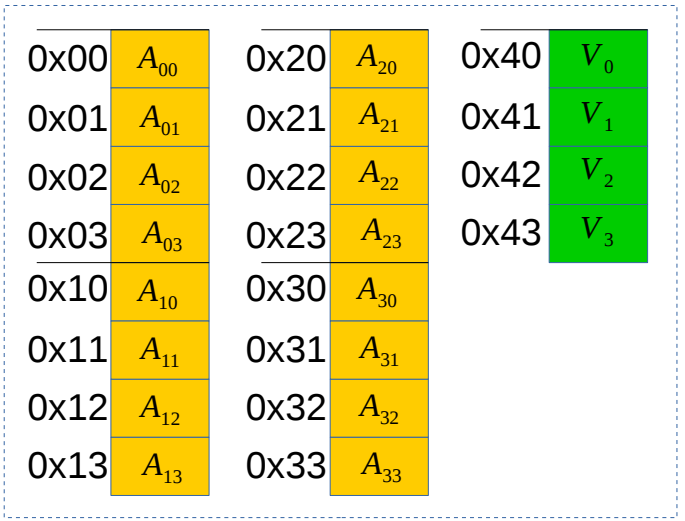
Exemple : Multiplication matrice - vecteur



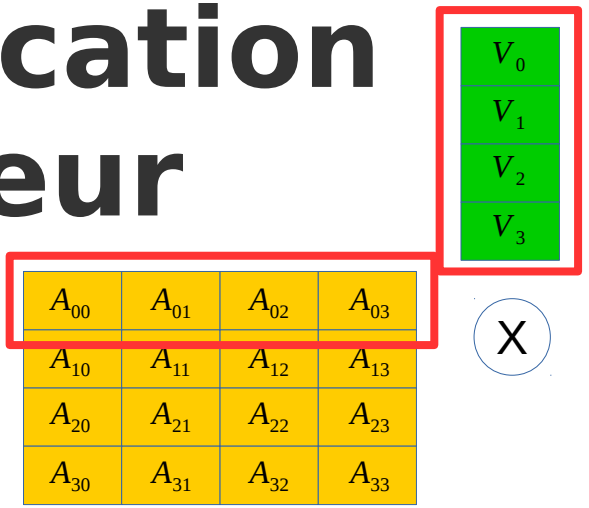
Fully associative

- Cache miss : 1
- Replacement : 0



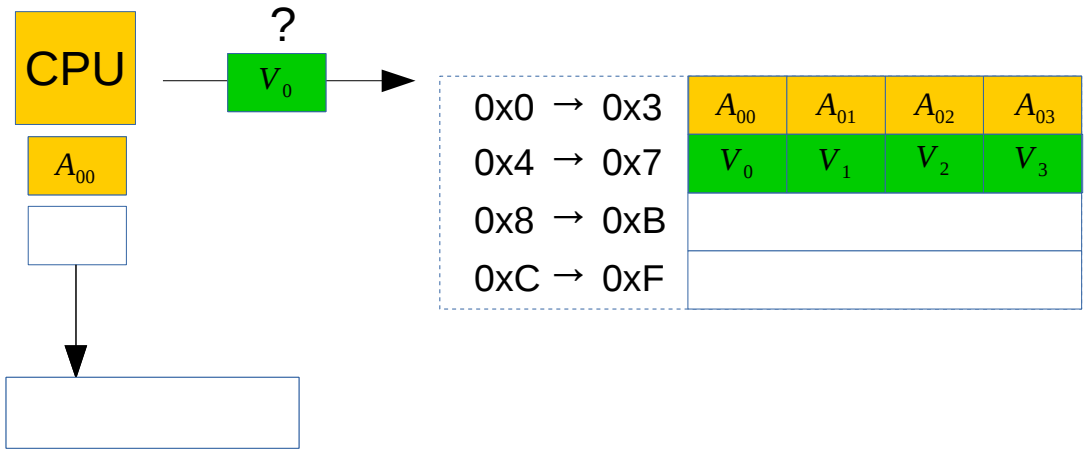


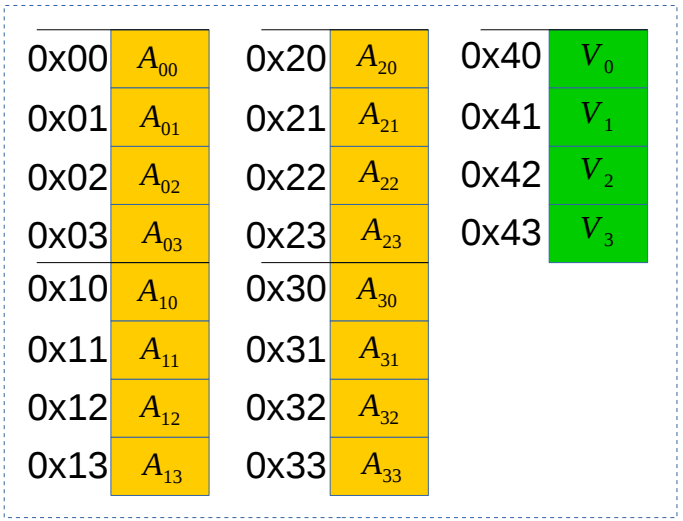
Exemple : Multiplication matrice - vecteur



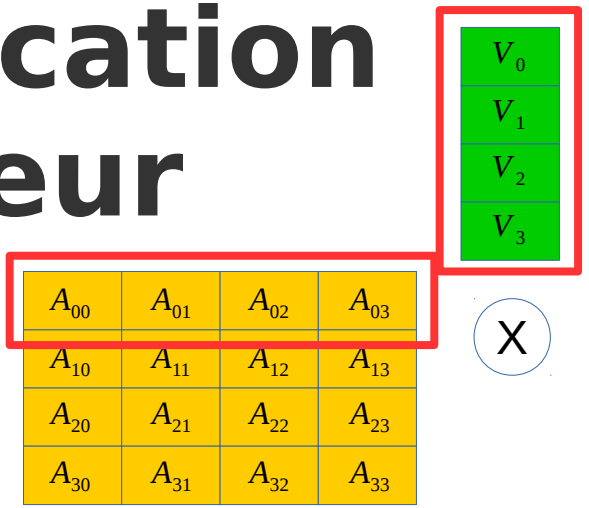
Fully associative

- Cache miss : 2
- Replacement : 0



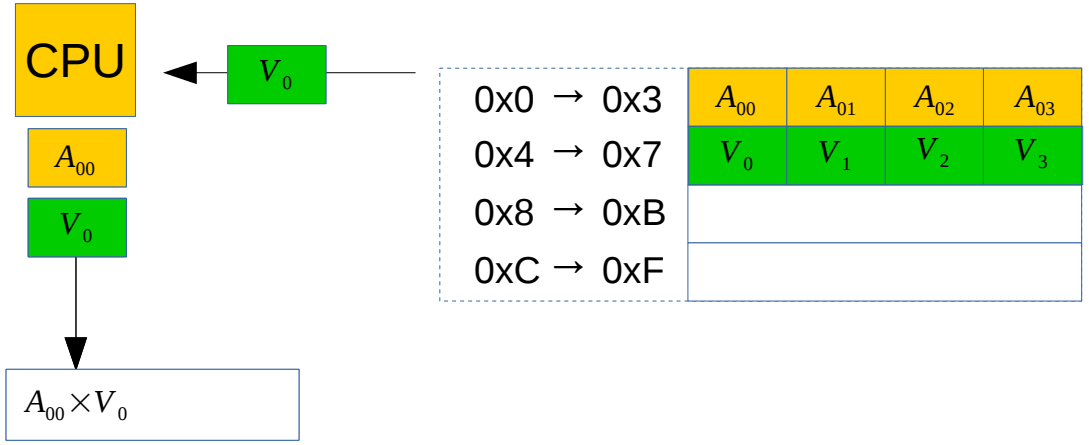


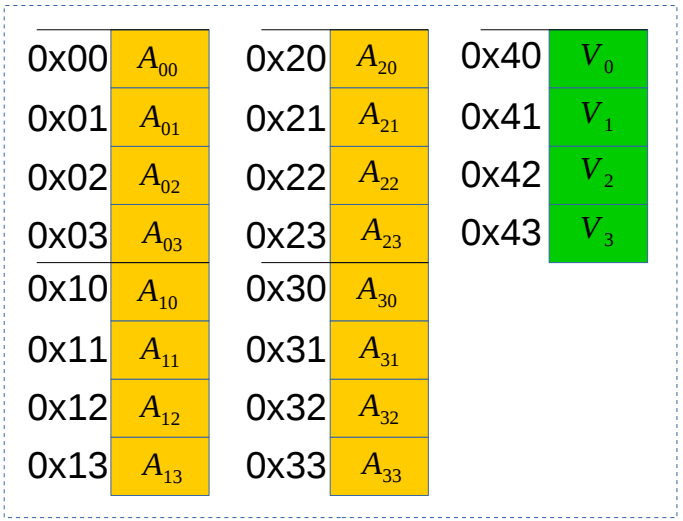
Exemple : Multiplication matrice - vecteur



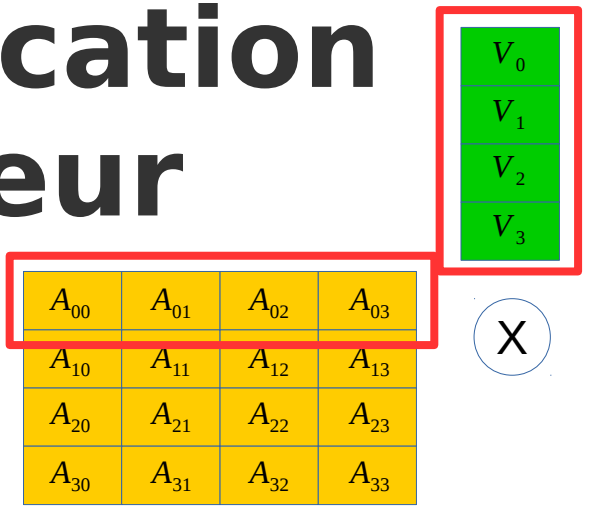
Fully associative

- Cache miss : 2
- Replacement : 0



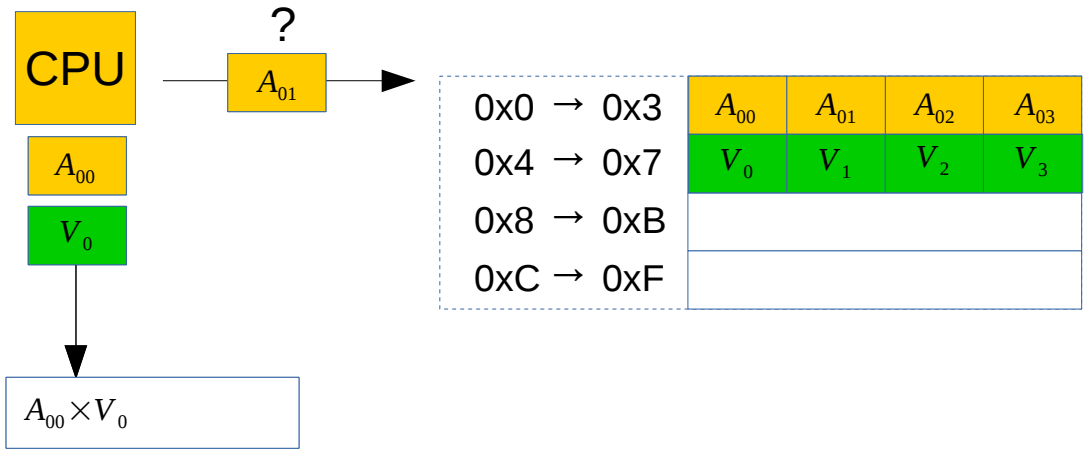


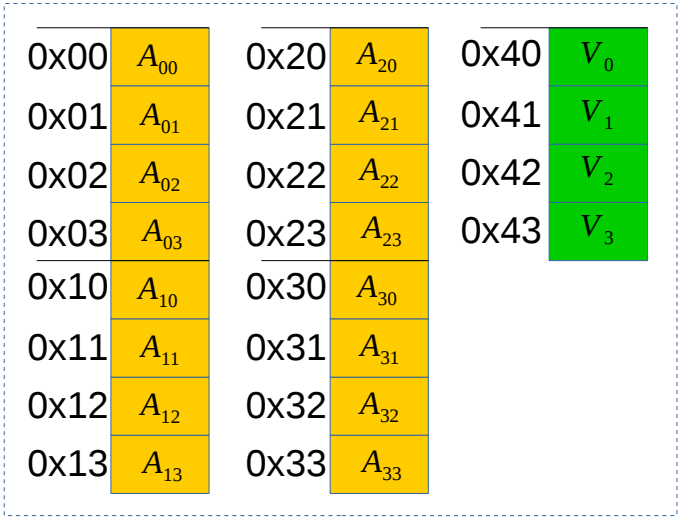
Exemple : Multiplication matrice - vecteur



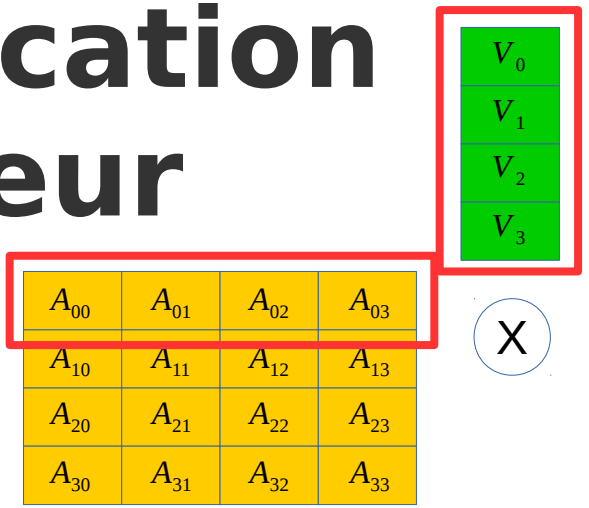
Fully associative

- Cache miss : 2
- Replacement : 0



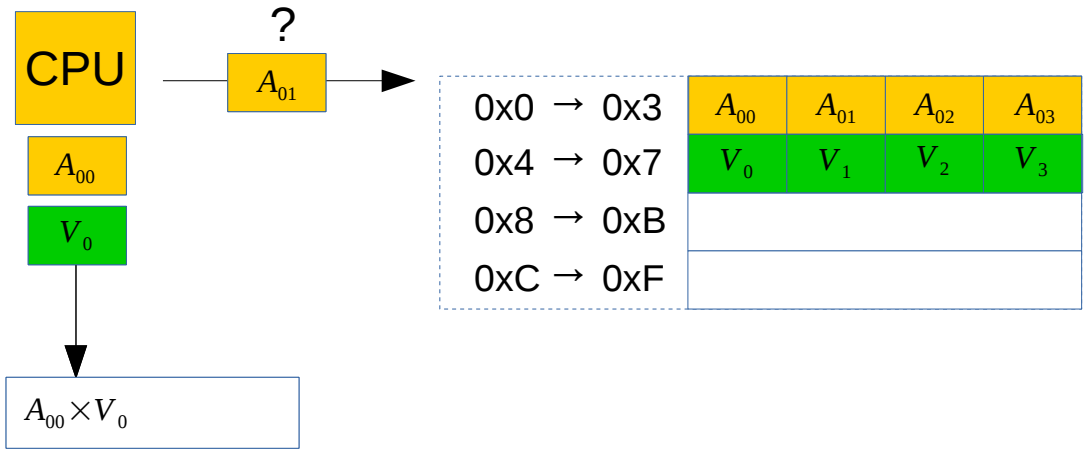


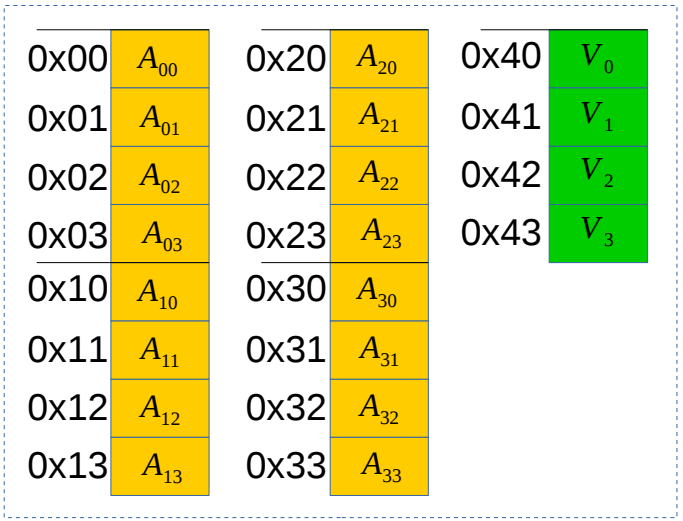
Exemple : Multiplication matrice - vecteur



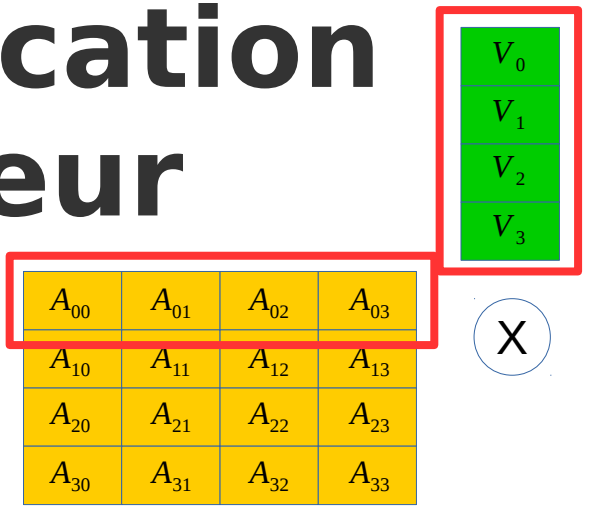
Fully associative

- Cache miss : 2 / Cache hit : 1
- Replacement : 0



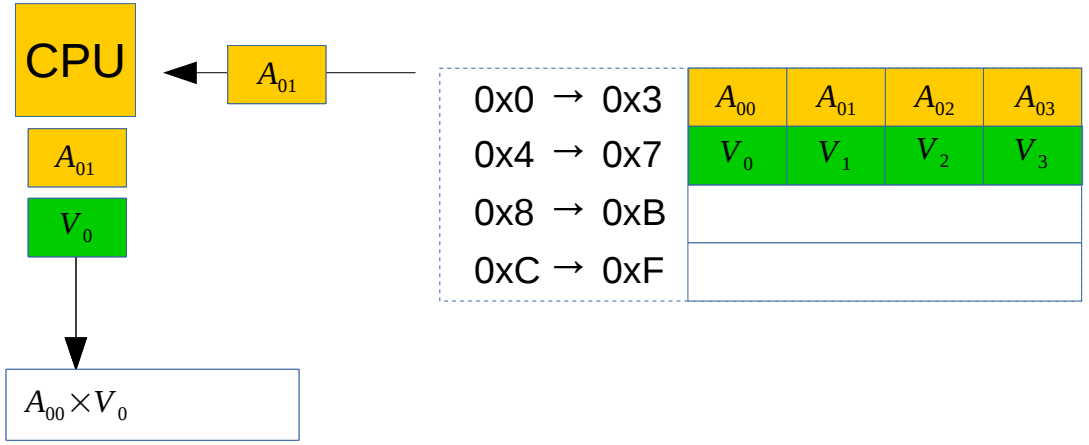


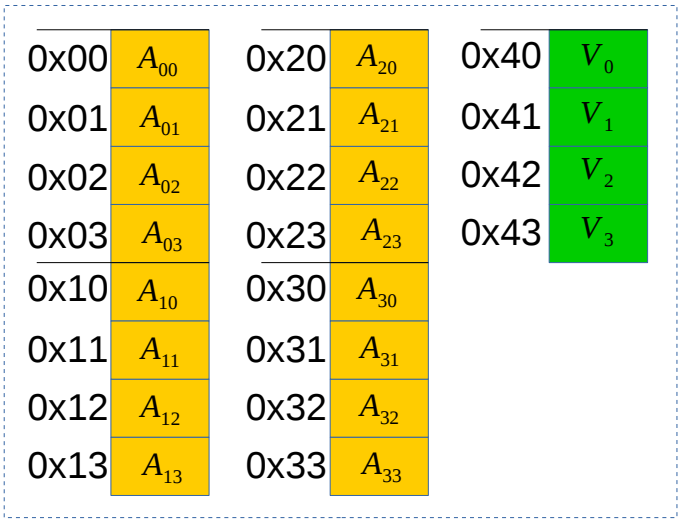
Exemple : Multiplication matrice - vecteur



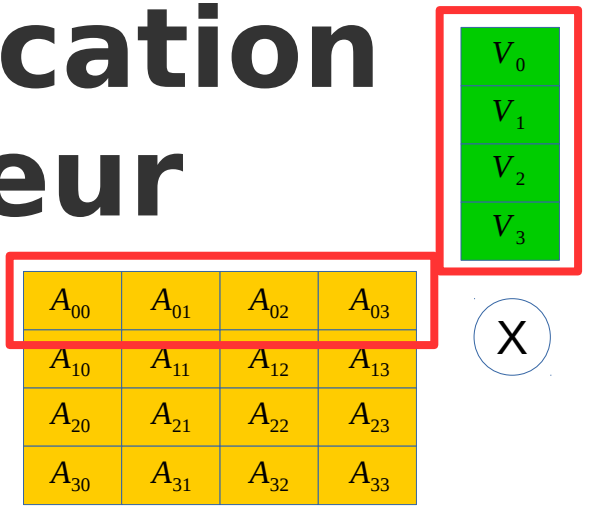
Fully associative

- Cache miss : 2 / Cache hit : 1
- Replacement : 0



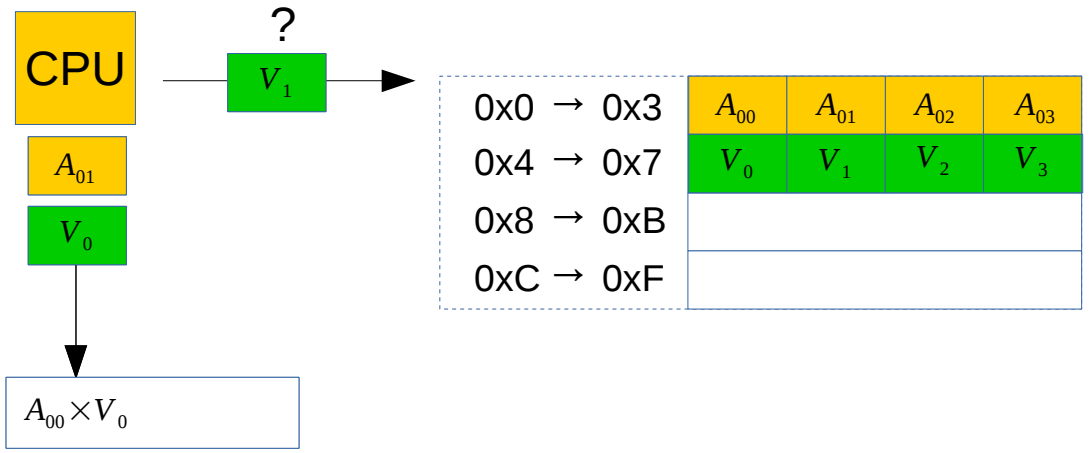


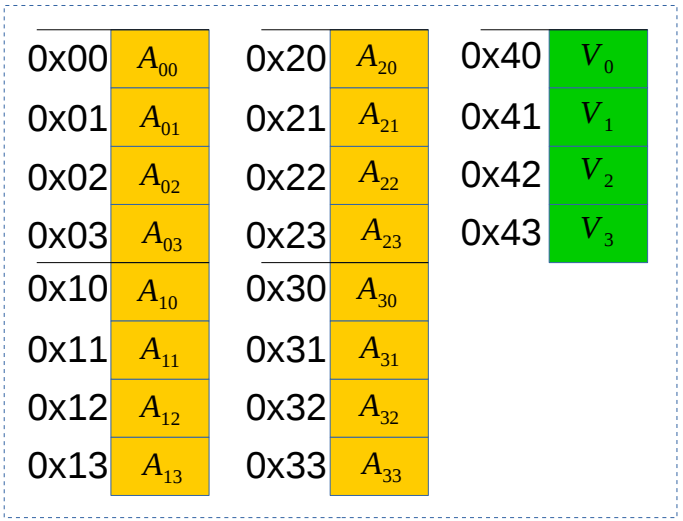
Exemple : Multiplication matrice - vecteur



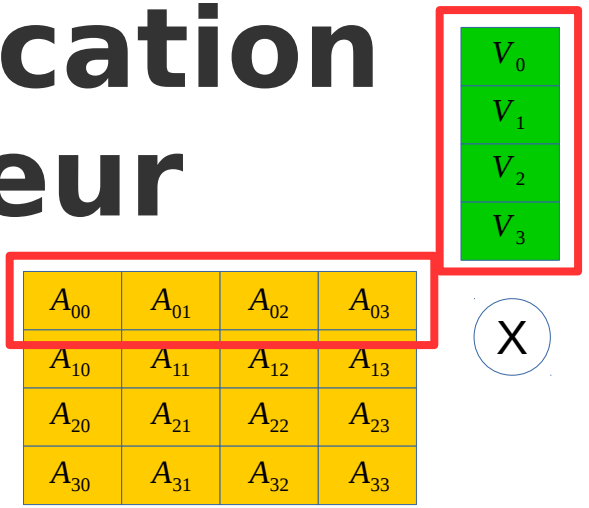
Fully associative

- Cache miss : 2 / Cache hit : 1
- Replacement : 0



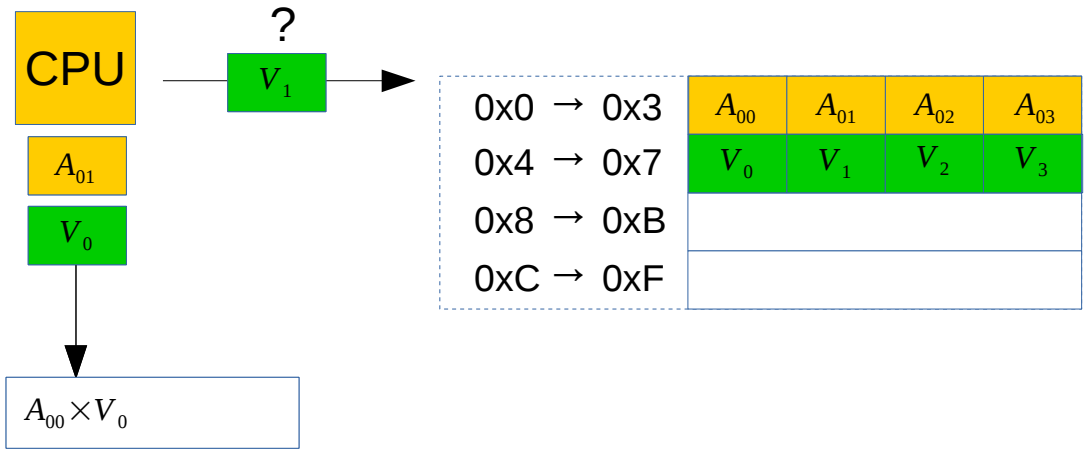


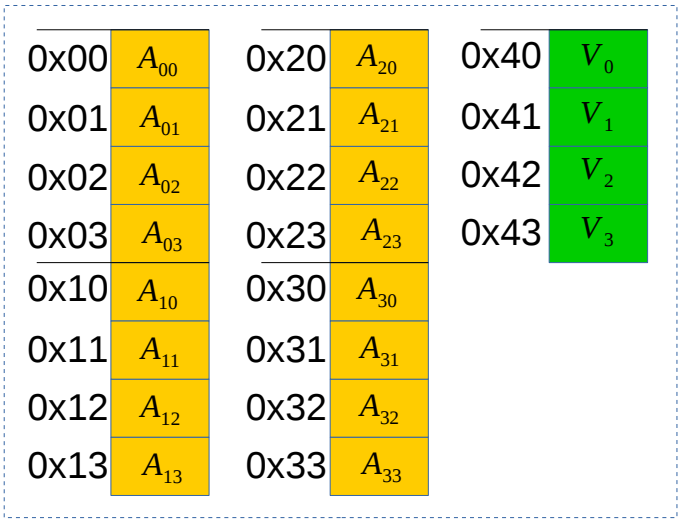
Exemple : Multiplication matrice - vecteur



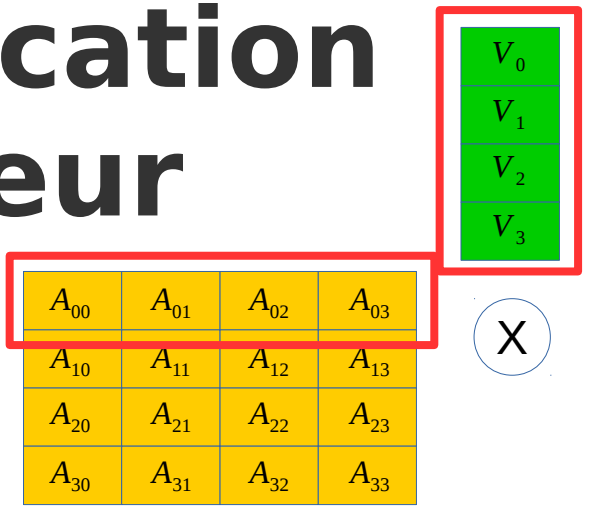
Fully associative

- Cache miss : 2 / Cache hit : 2
- Replacement : 0



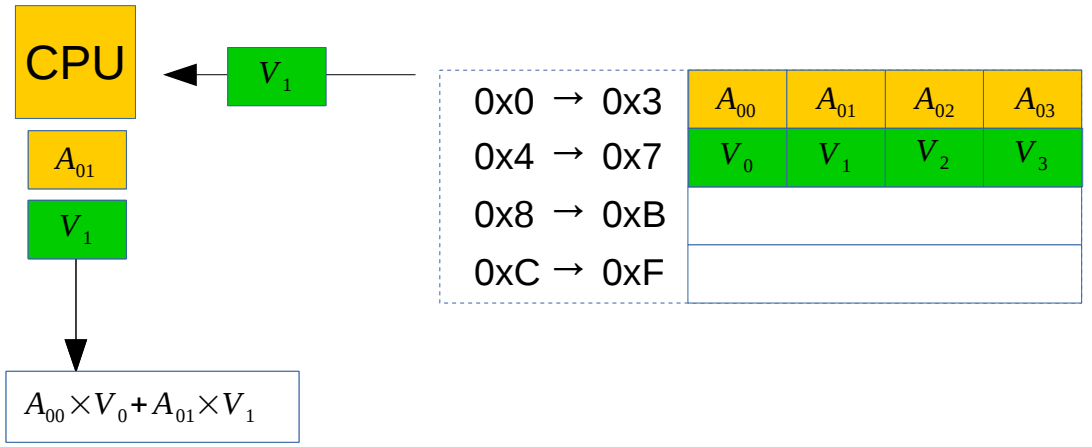


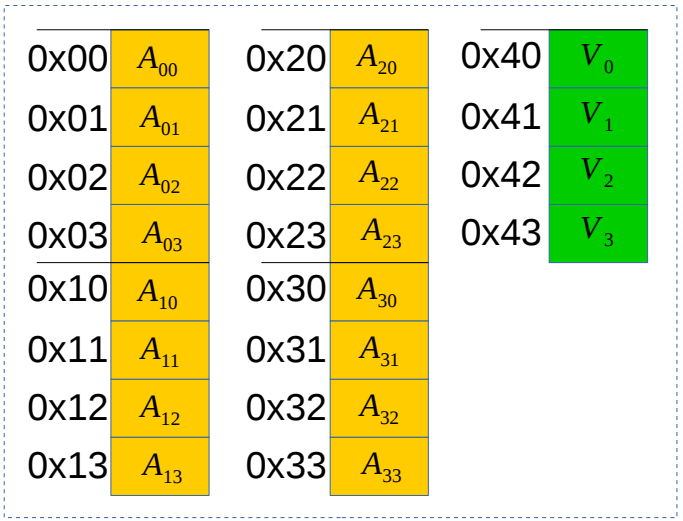
Exemple : Multiplication matrice - vecteur



Fully associative

- Cache miss : 2 / Cache hit : 2
- Replacement : 0





Exemple : Multiplication matrice - vecteur

Quelques opérations plus tard ...

Conclusion

Direct-mapped :

- Cache hit : 0
- Cache miss : 32
- Remplacement : 31

Fully associative :

- Cache hit : 27
- Cache miss : 5
- Remplacement : 1

Application numérique :

- Cache hit = 0.5 ns / Cache miss : 50 ns / Remplacement : 50 ns

Conclusion

Direct-mapped :

- Cache hit : 0
- Cache miss : 32
- Remplacement : 31

Fully associative :

- Cache hit : 27
- Cache miss : 5
- Remplacement : 1

Application numérique :

- Cache hit = 0.5 ns / Cache miss : 50 ns / Remplacement : 50 ns

Direct-mapped :

3.15 µs

Fully associative :

313 ns