Mémoire: cache

GIF-1001 Ordinateurs: Structure et Applications, Hiver 2019

Jean-François Lalonde

Image: une matrice de pixels

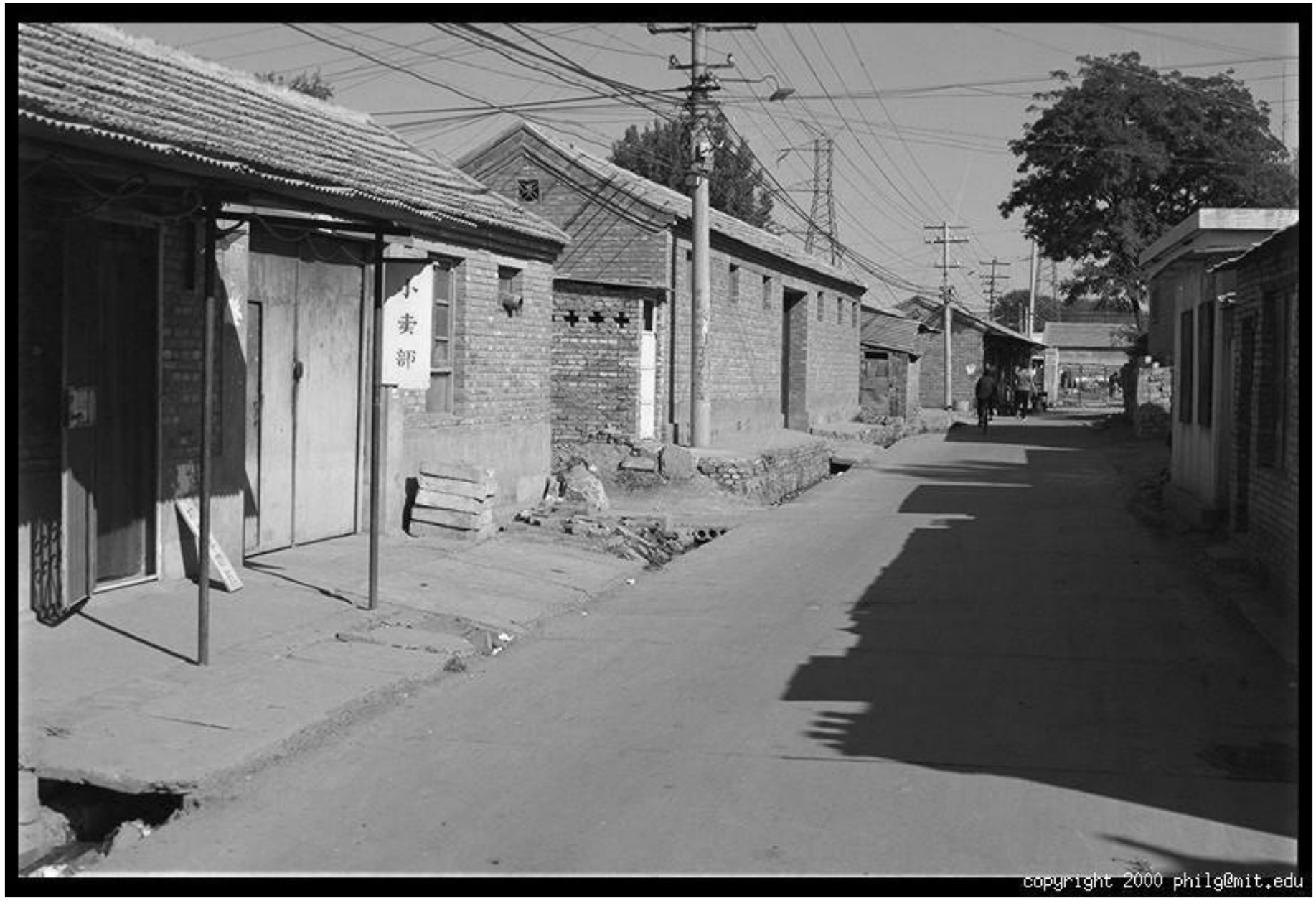


Image: une matrice de pixels

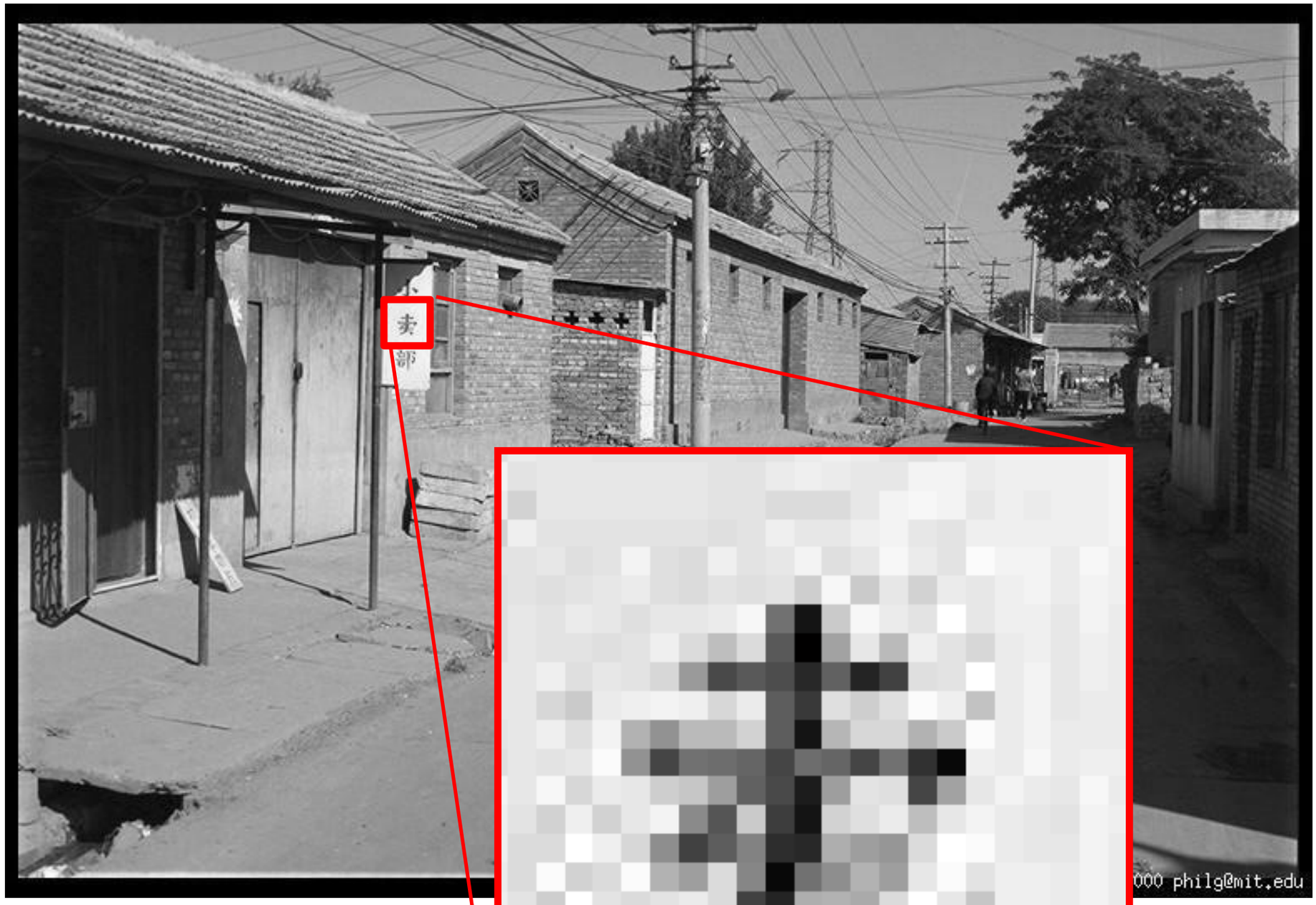
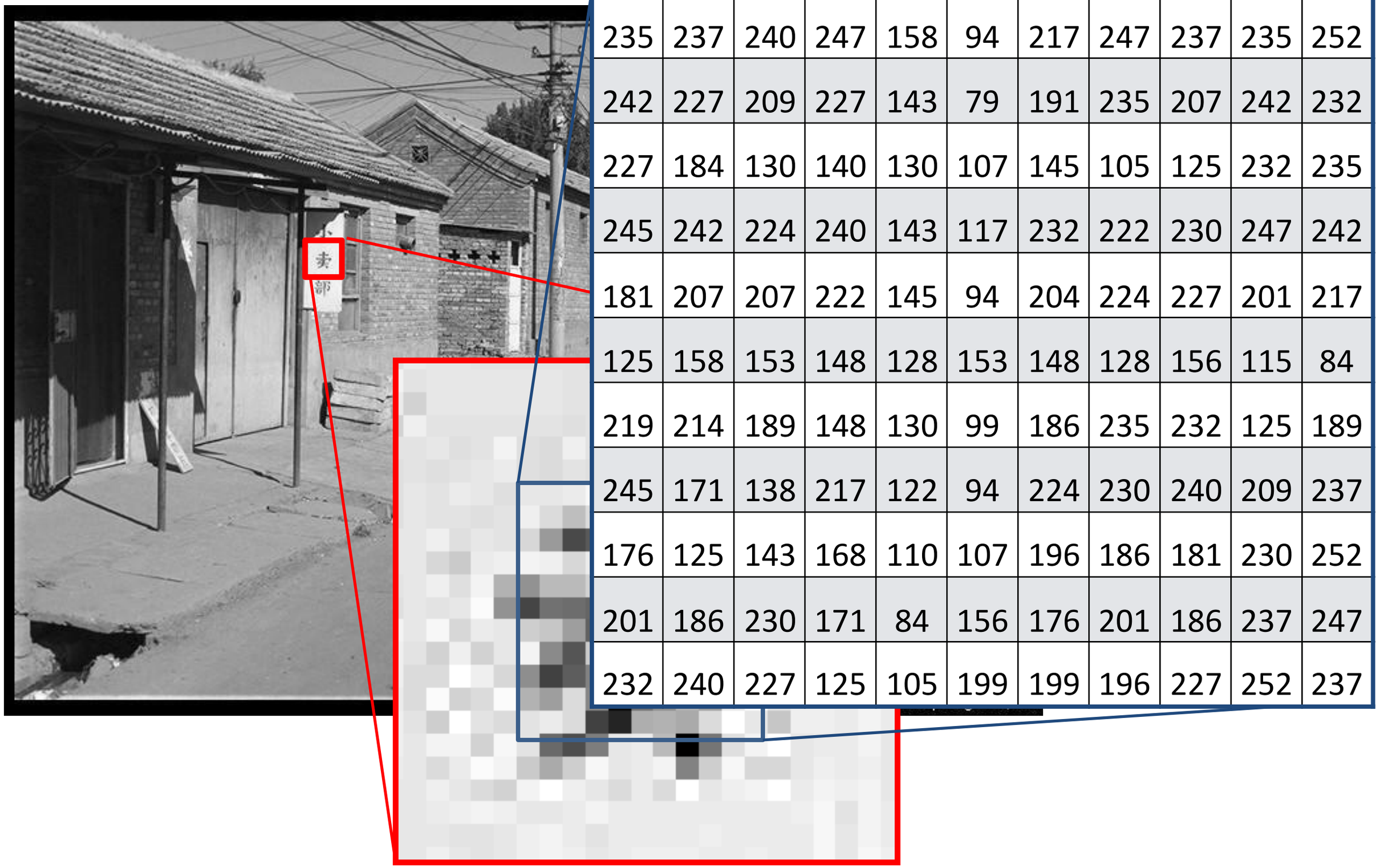


Image: une matrice de pixels

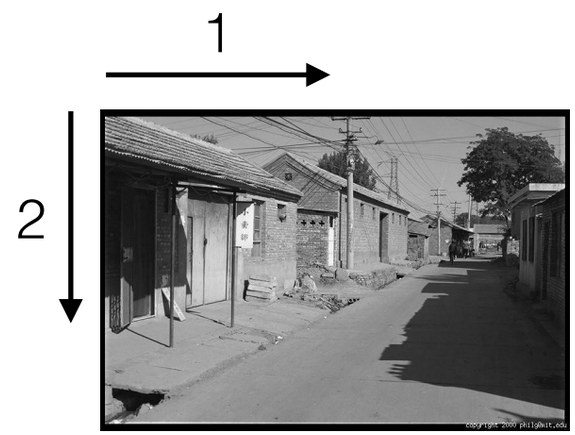


235	237	240	247	158	94	217	247	237	235	252
242	227	209	227	143	79	191	235	207	242	232
227	184	130	140	130	107	145	105	125	232	235
245	242	224	240	143	117	232	222	230	247	242
181	207	207	222	145	94	204	224	227	201	217
125	158	153	148	128	153	148	128	156	115	84
219	214	189	148	130	99	186	235	232	125	189
245	171	138	217	122	94	224	230	240	209	237
176	125	143	168	110	107	196	186	181	230	252
201	186	230	171	84	156	176	201	186	237	247
232	240	227	125	105	199	199	196	227	252	237

Aujourd’hui

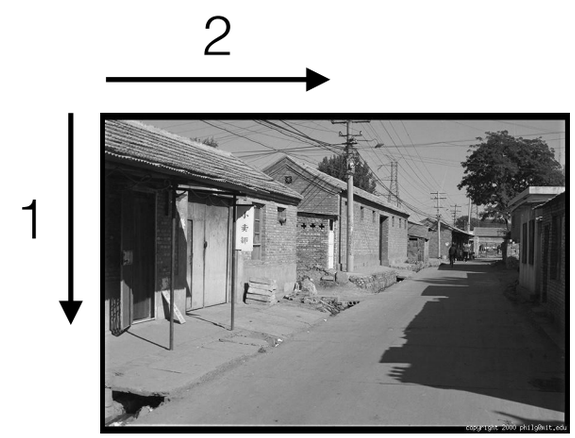
- Laquelle de ces deux versions est la plus rapide?

```
for (int i = 0; i < 256; i++) {
  for (int j = 0; j < 512; j++) {
    img[i*512 + j] = 0;
   }
 }
```



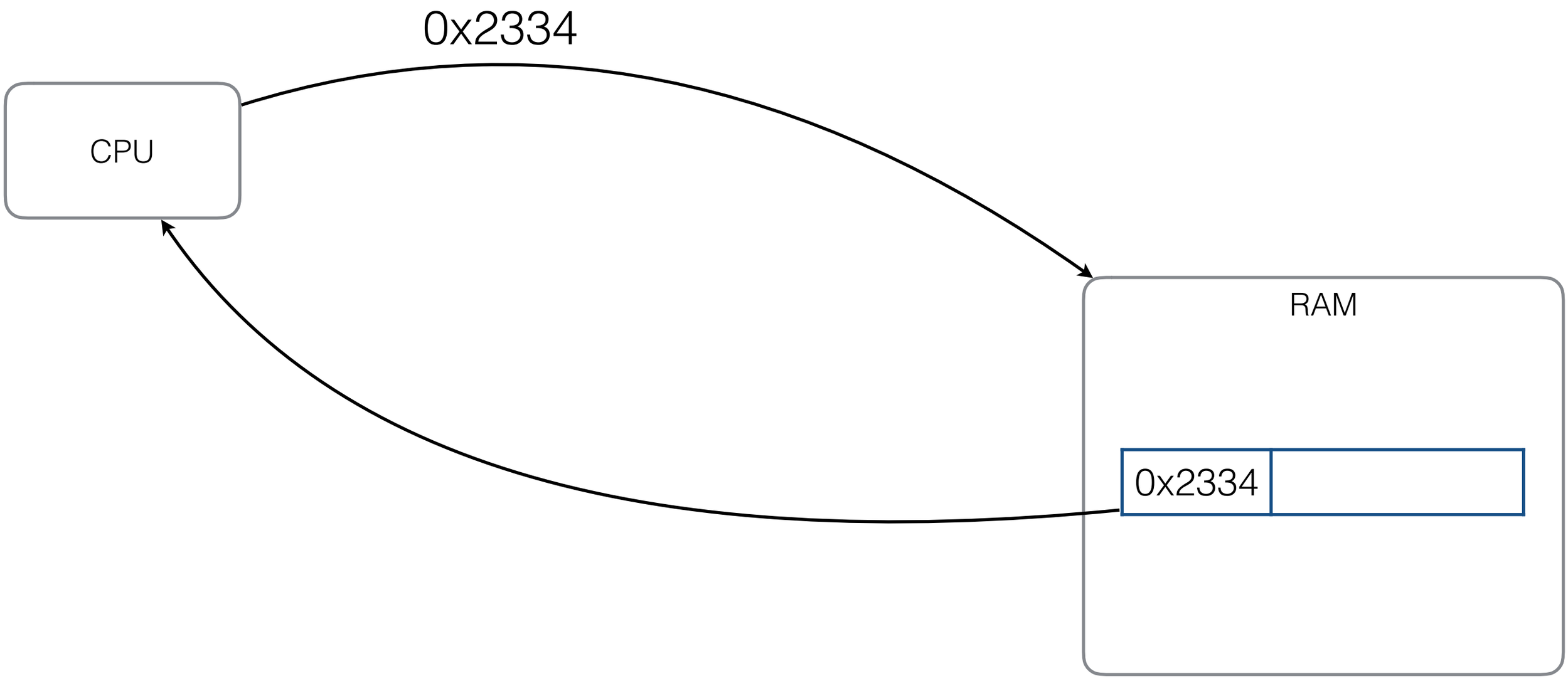
ou

```
for (int j = 0; j < 512; i++) {
  for (int i = 0; i < 256; j++) {
    img[i*512 + j] = 0;
   }
 }
```



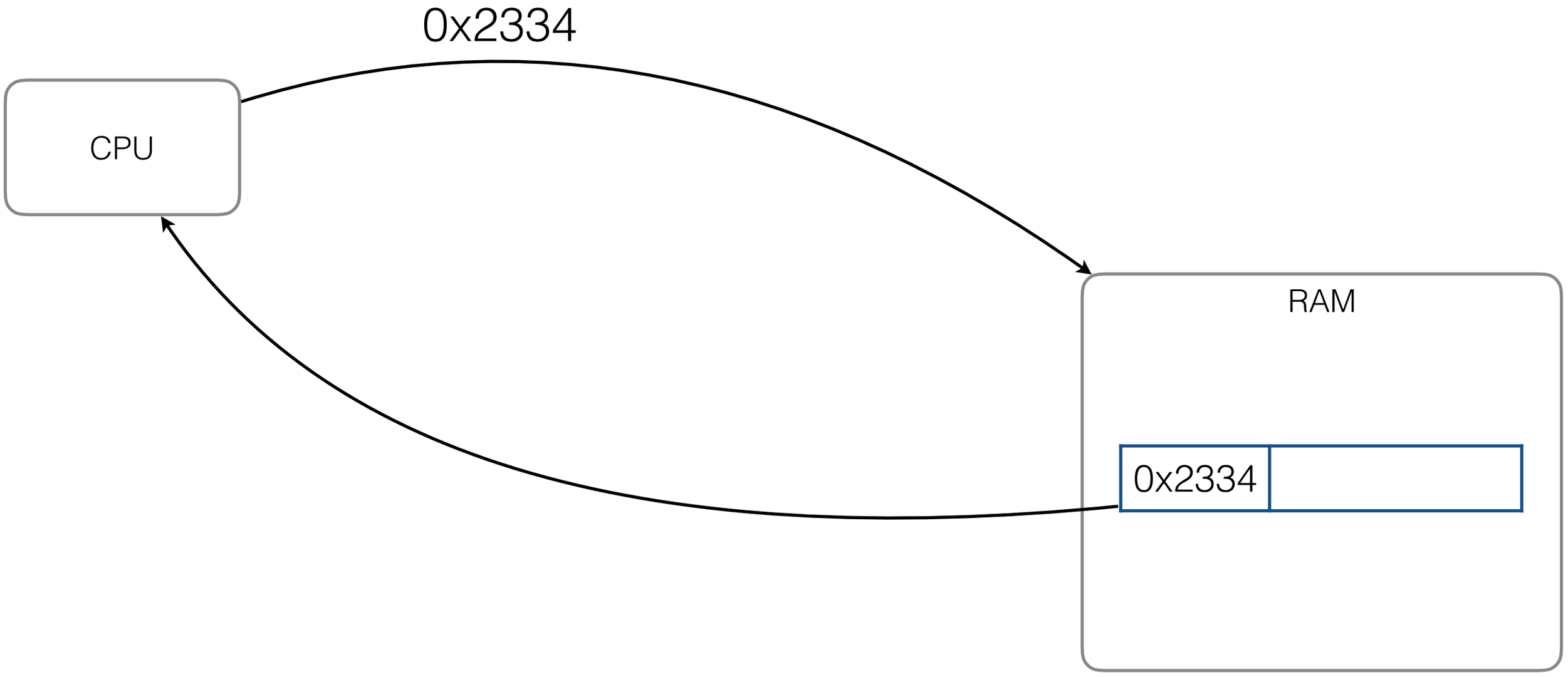
i = lignes (“y”)
j = colonnes (“x”)

Lecture en mémoire

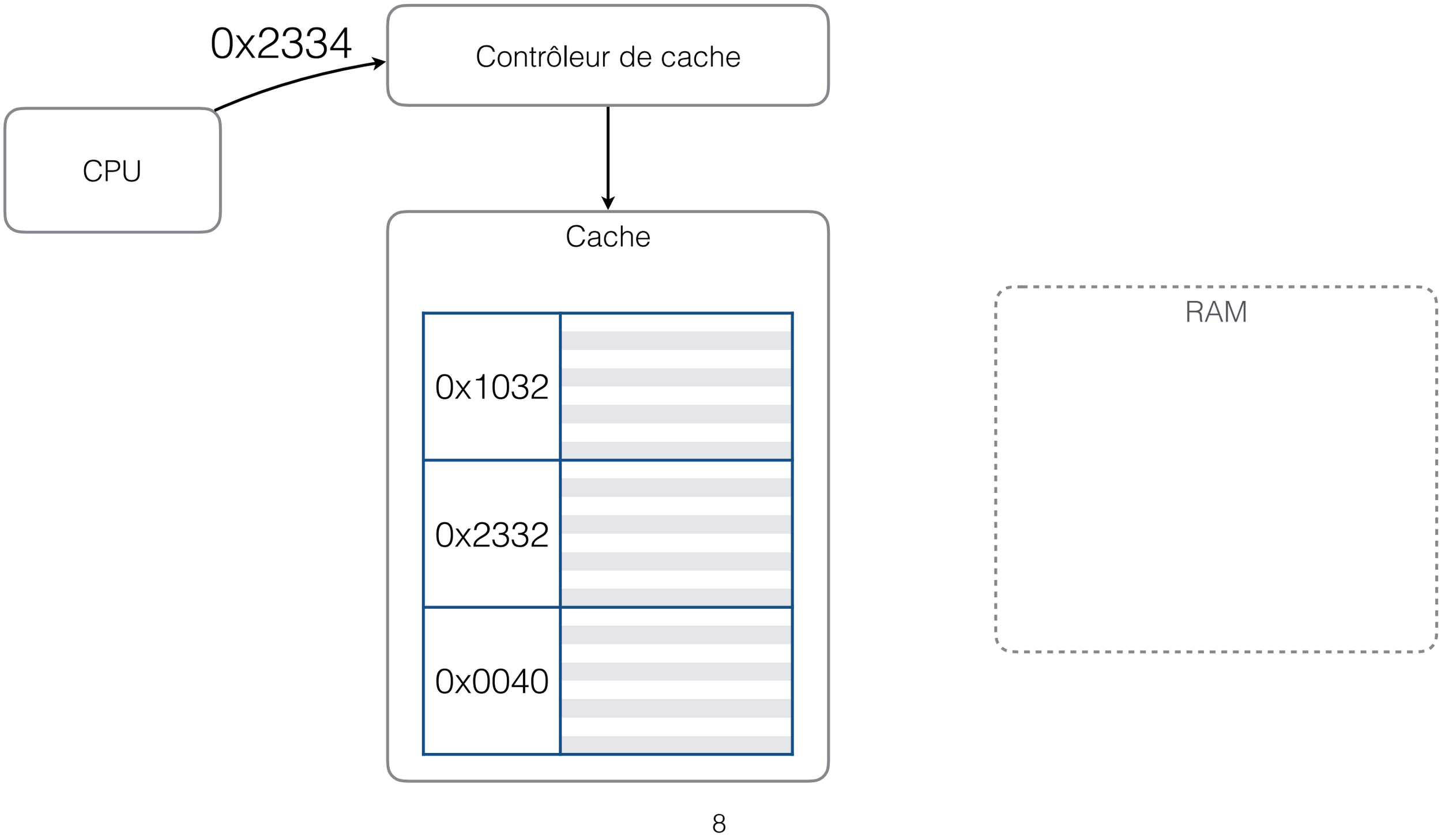


Lecture en mémoire

Comment faire pour accélérer les accès mémoire?



On rajoute une cache!



Les caches

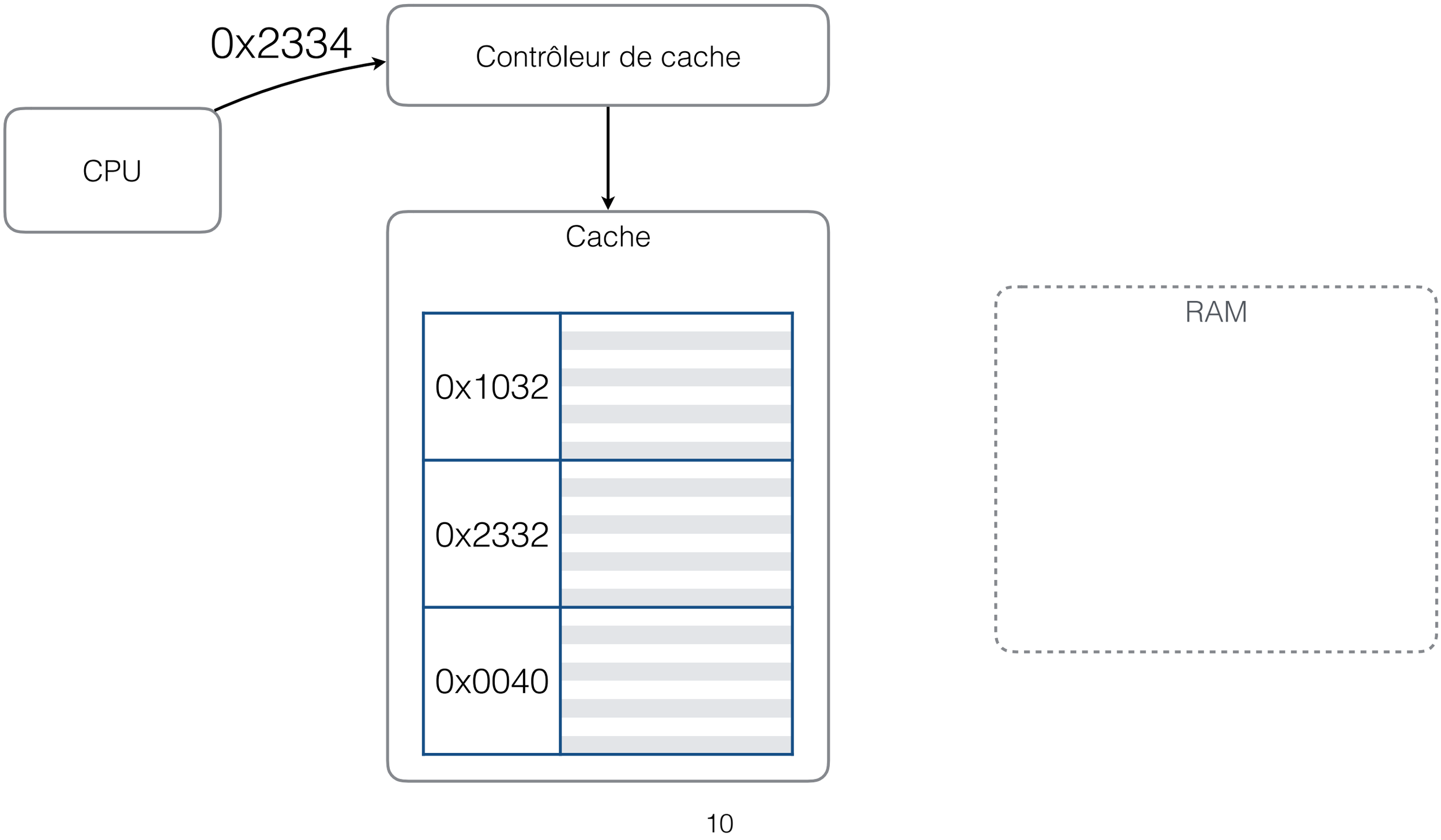
- Cache: mémoire très rapide placée entre une unité rapide et une mémoire lente afin d'accélérer les accès mémoire.

- La taille d'une cache est plus petite que la mémoire lente à laquelle est est branchée

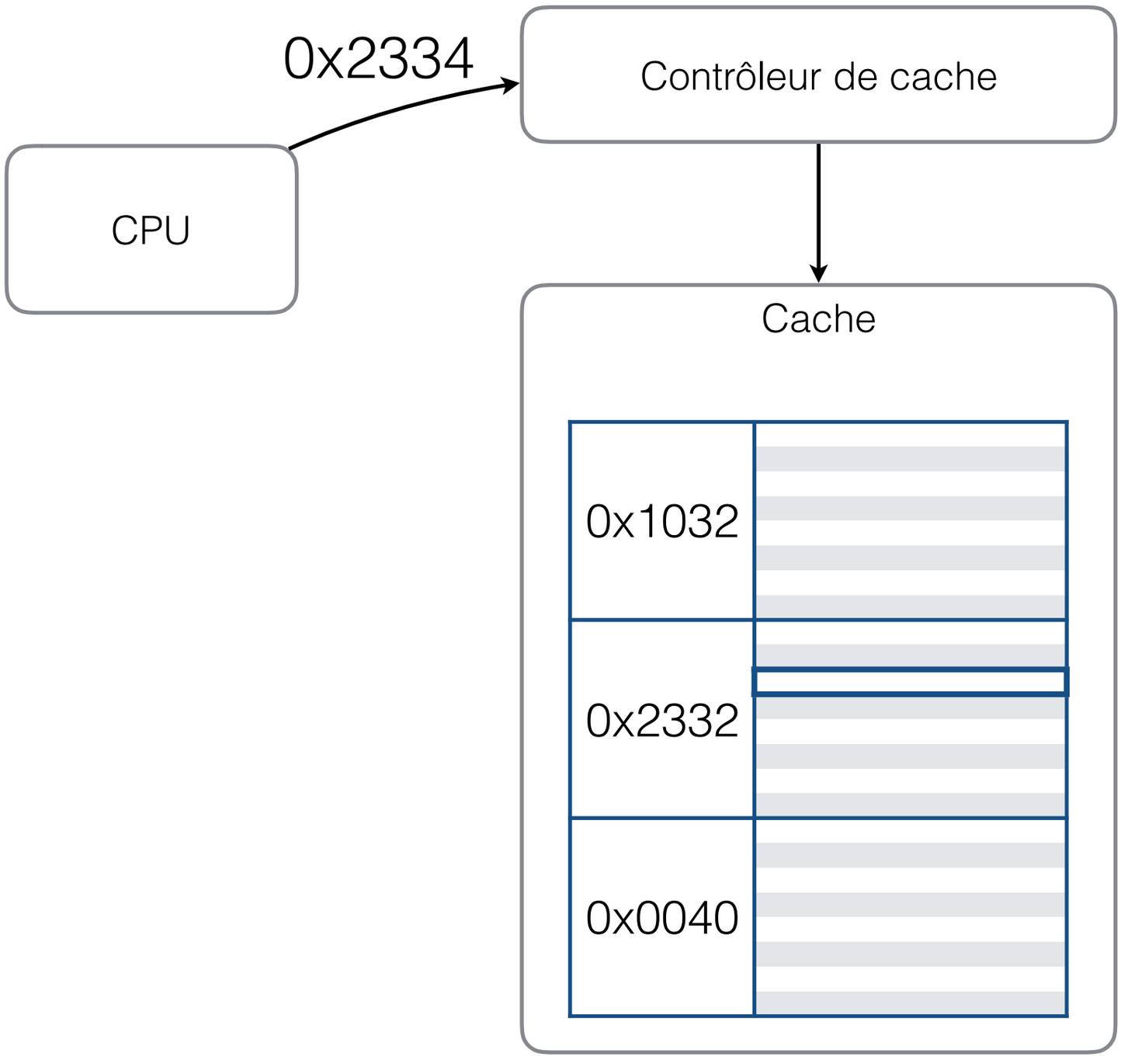
- La cache contient des *blocs* de données plutôt que des données individuelles.

Type	Temps d'accès	Vitesse de transfert
Registres	0.25 ns (proc. 4 GHz)	très rapide!
Cache (SRAM)	1—10 ns	>> 1 GB/s
Mémoire vive (SDRAM)	10—20 ns	>> 1 GB/s

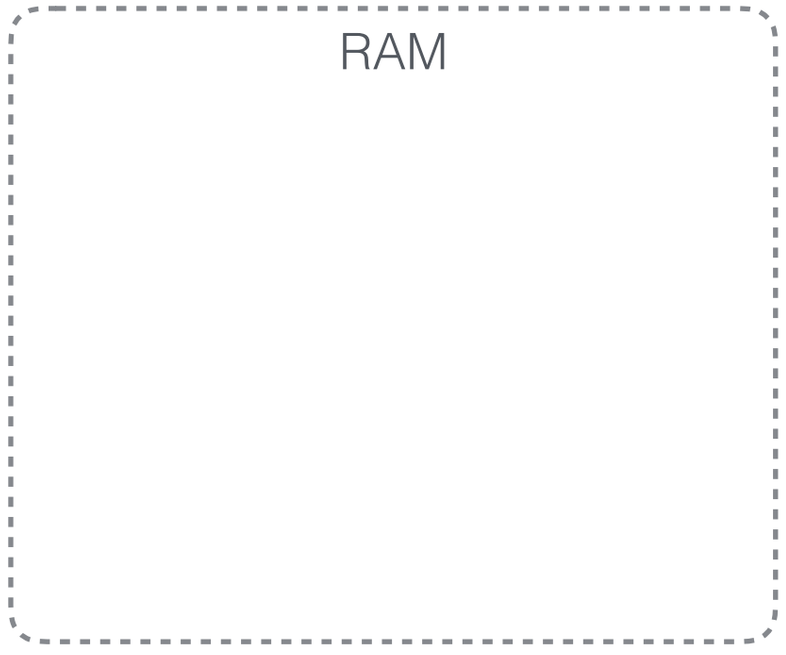
Lecture en cache



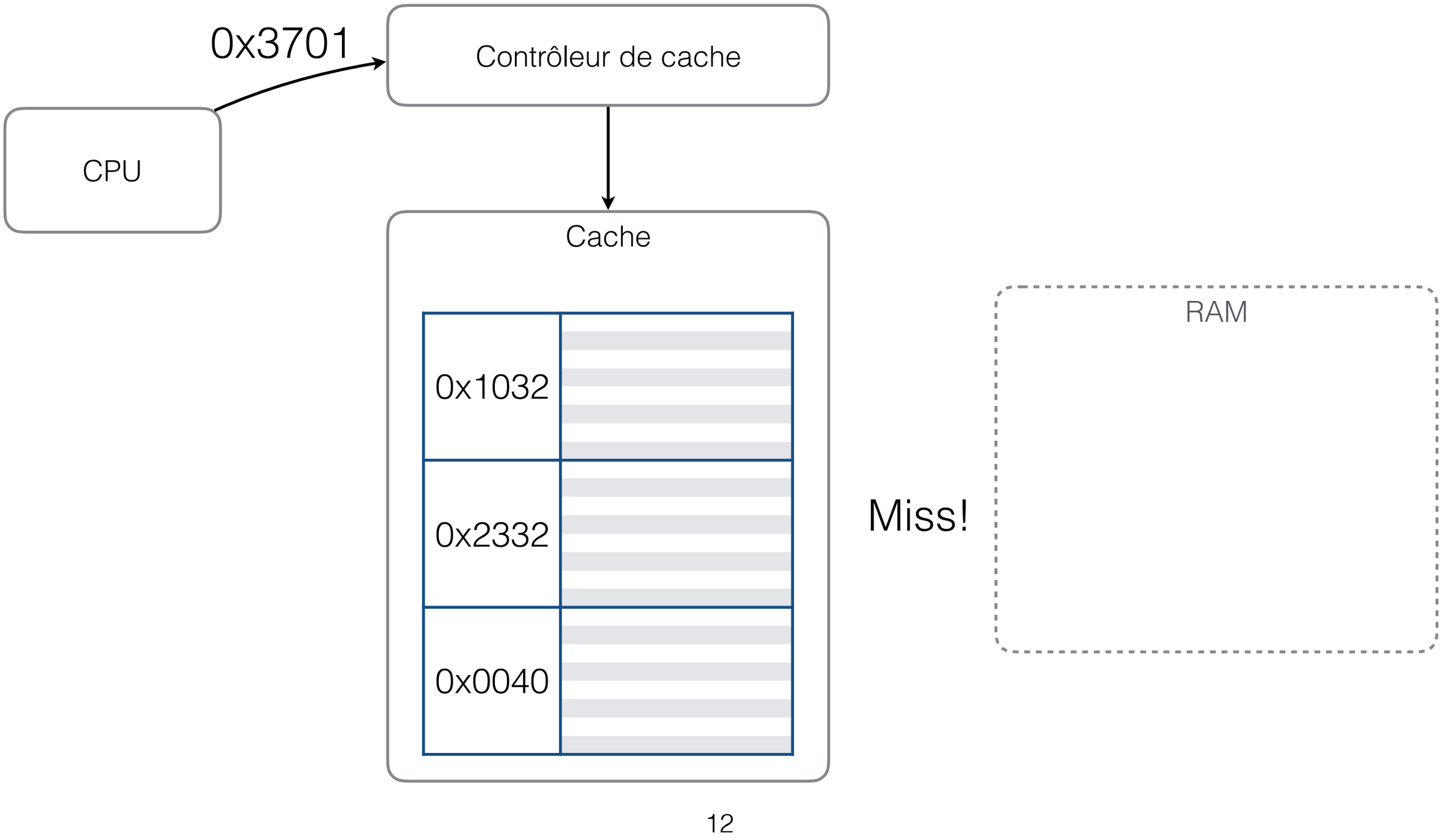
Lecture en cache



Hit!



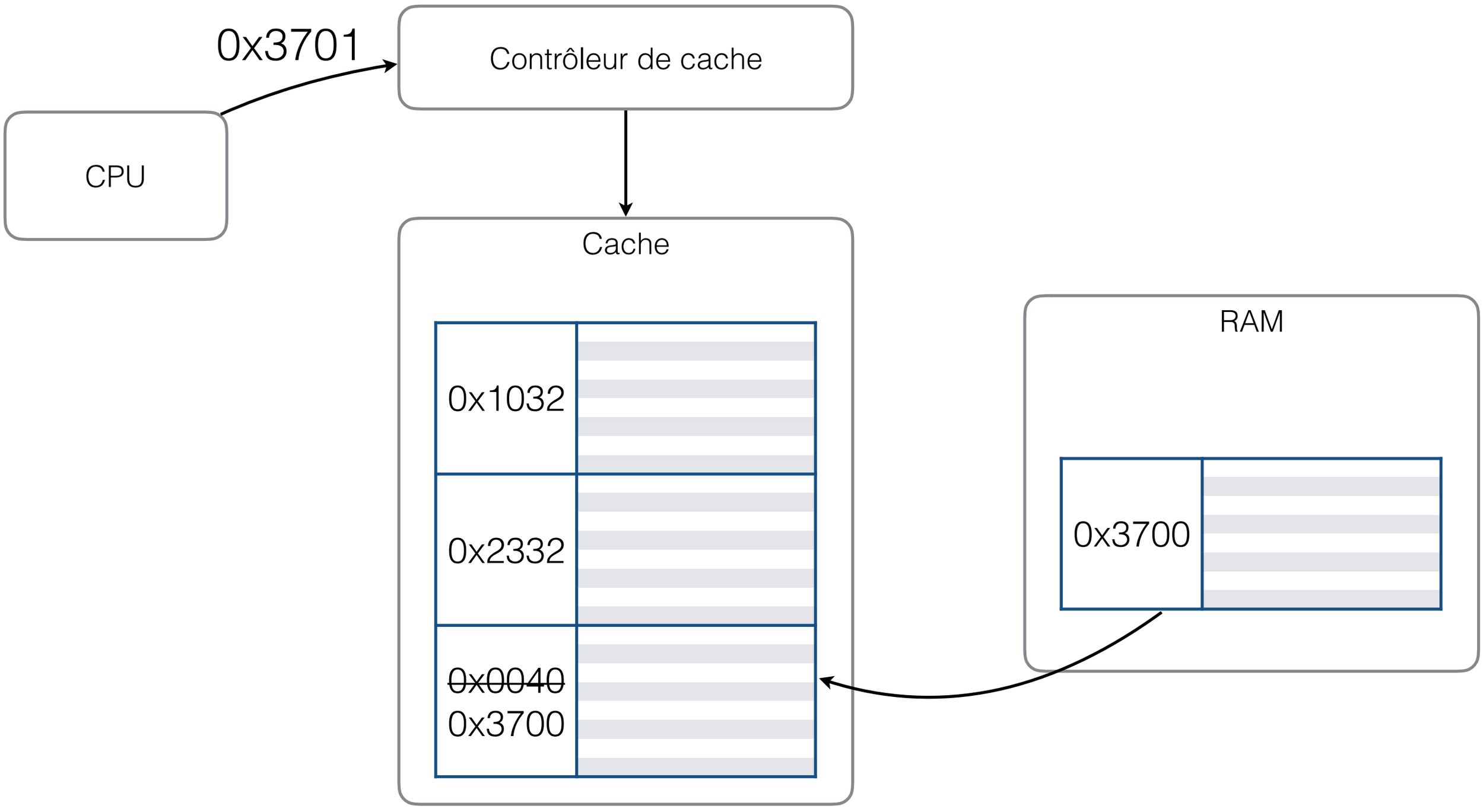
Lecture en cache



« cache miss »

- Lorsqu'il y a un « cache miss », il faut effectuer les opérations suivantes:
  - trouver un bloc à remplacer
  - copier les données de la RAM vers la cache
- Quel bloc remplacer?
  - Prendre le bloc qui a été utilisé le moins récemment, « Least Recently Used (LRU) »

Lecture en cache



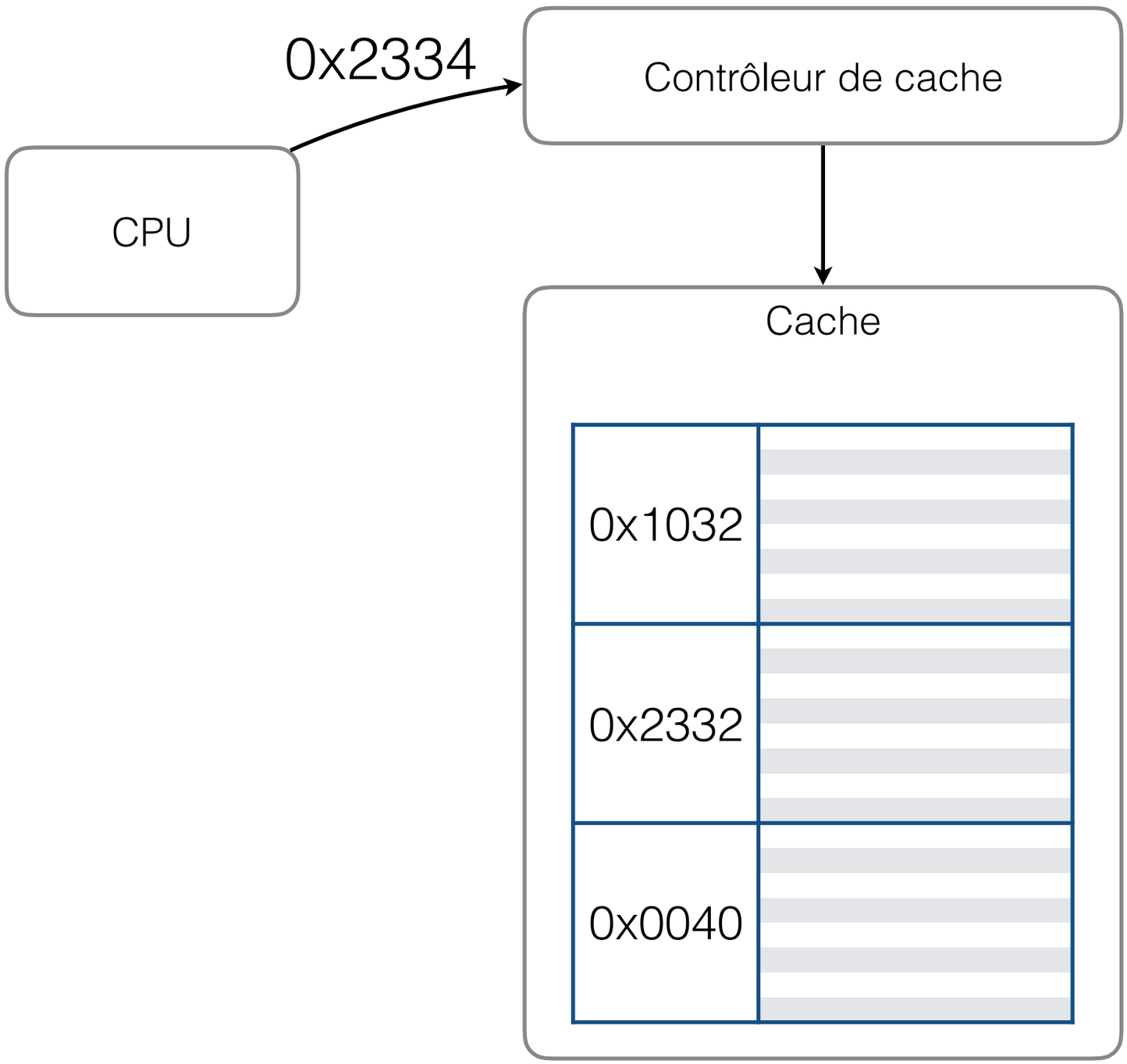
Écriture en cache

- Que faire lorsque l'on veut écrire en cache?

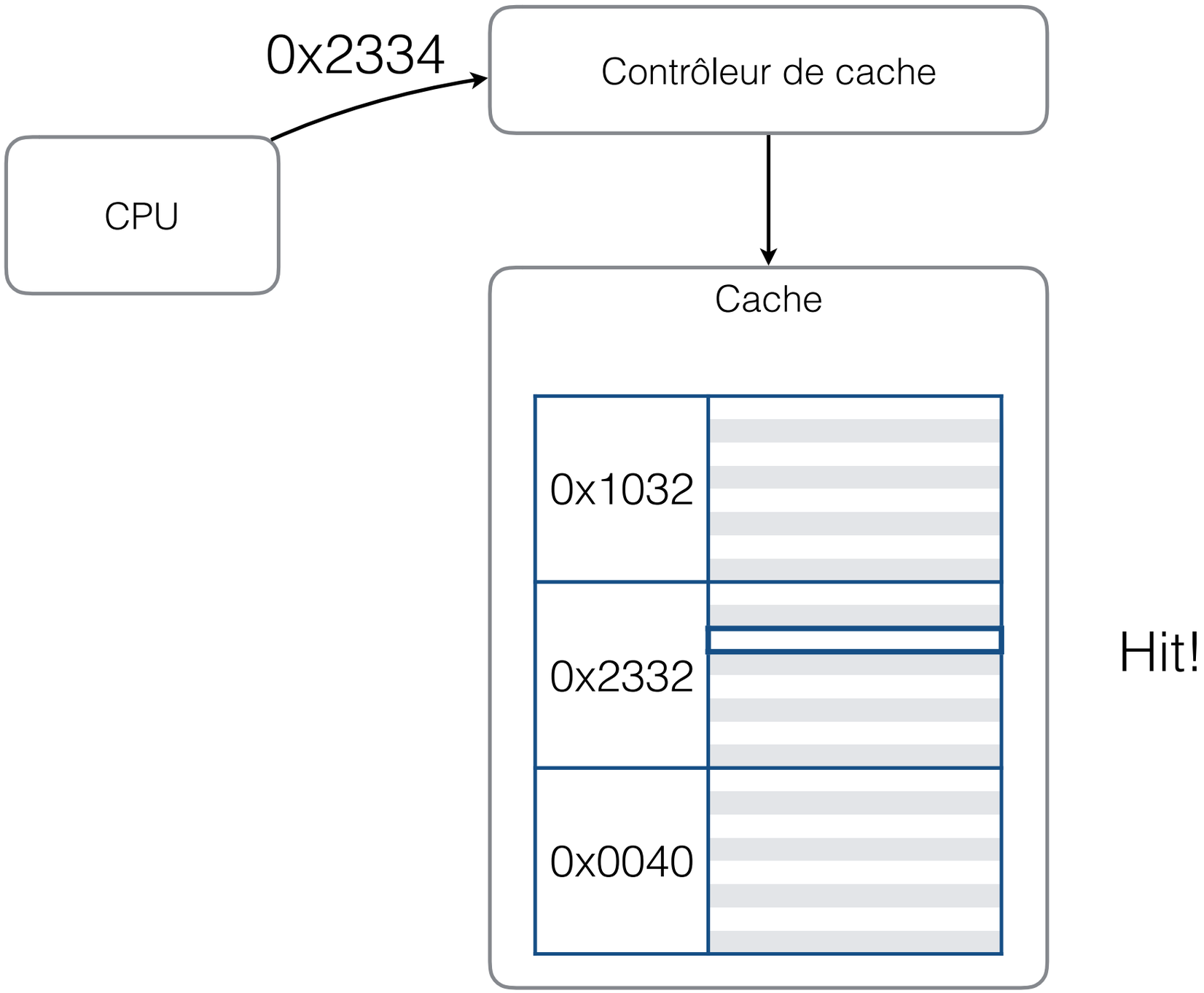
1. « **Write-through** » : écrire les changements dans la RAM au fur et à mesure

    - L'exemple précédent utilisait cette technique

Écriture en cache « write-through »

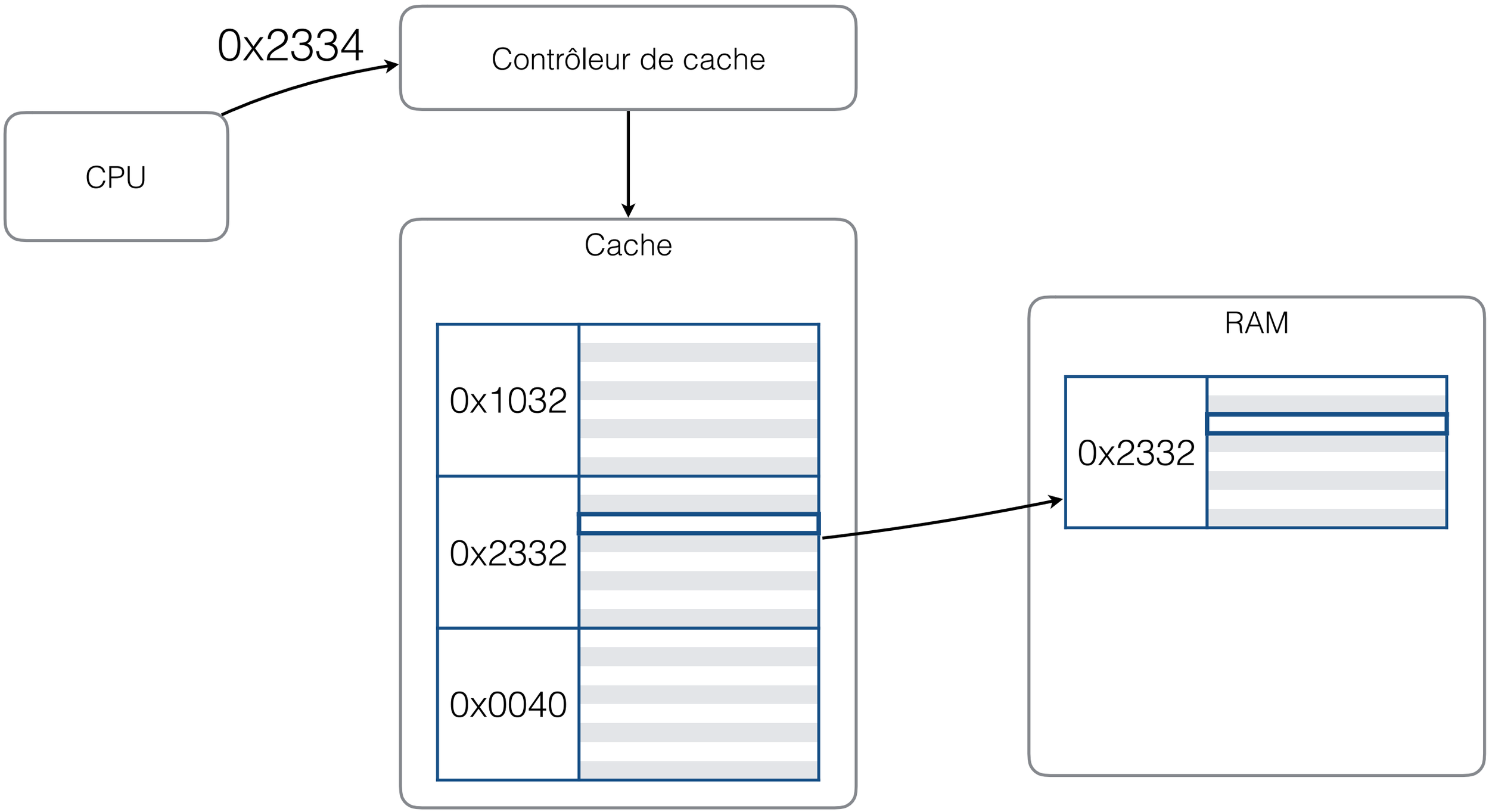


Écriture en cache « write-through »

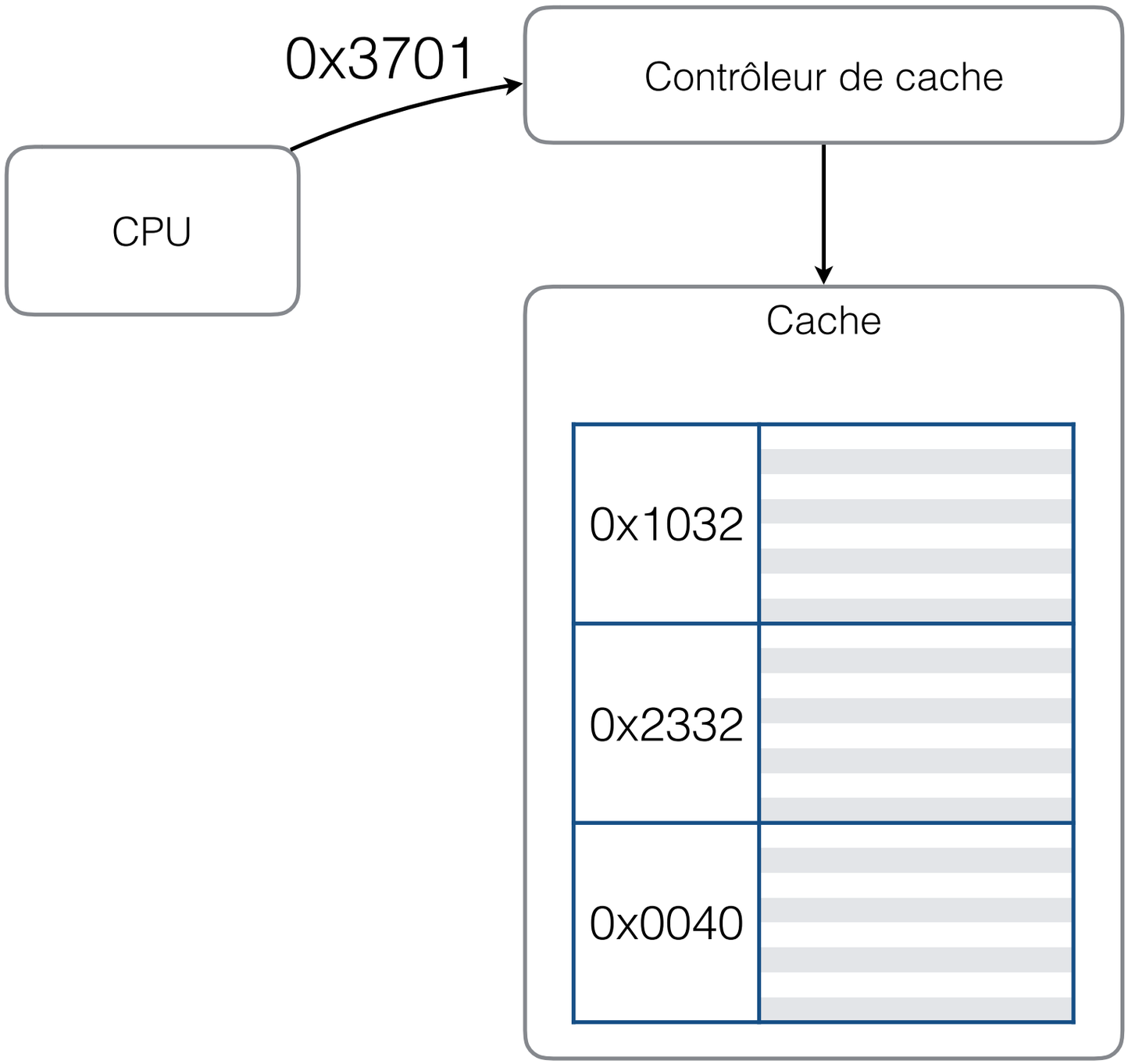


Hit!

Écriture en cache « write-through »

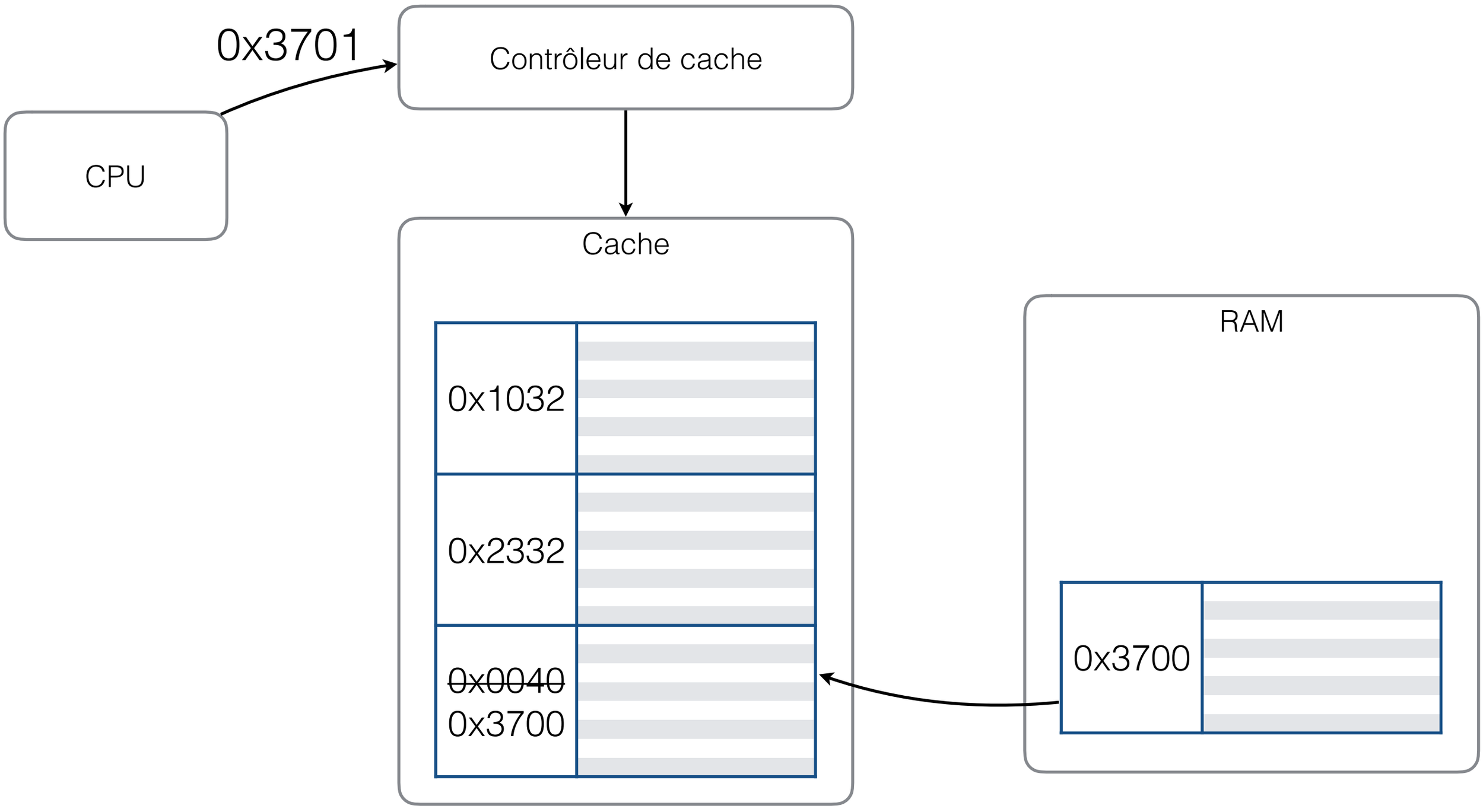


Écriture en cache « write-through »

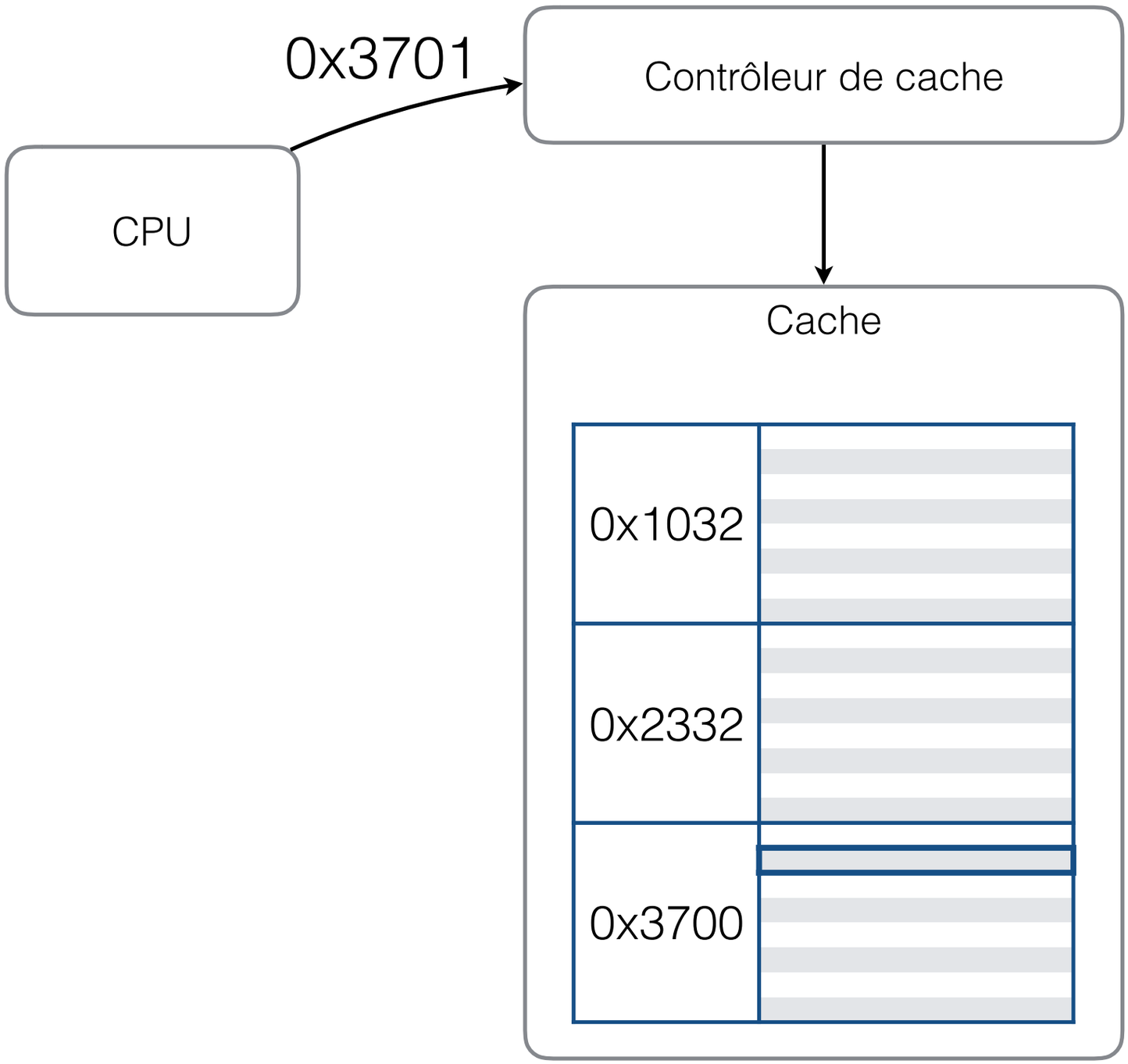


Miss!

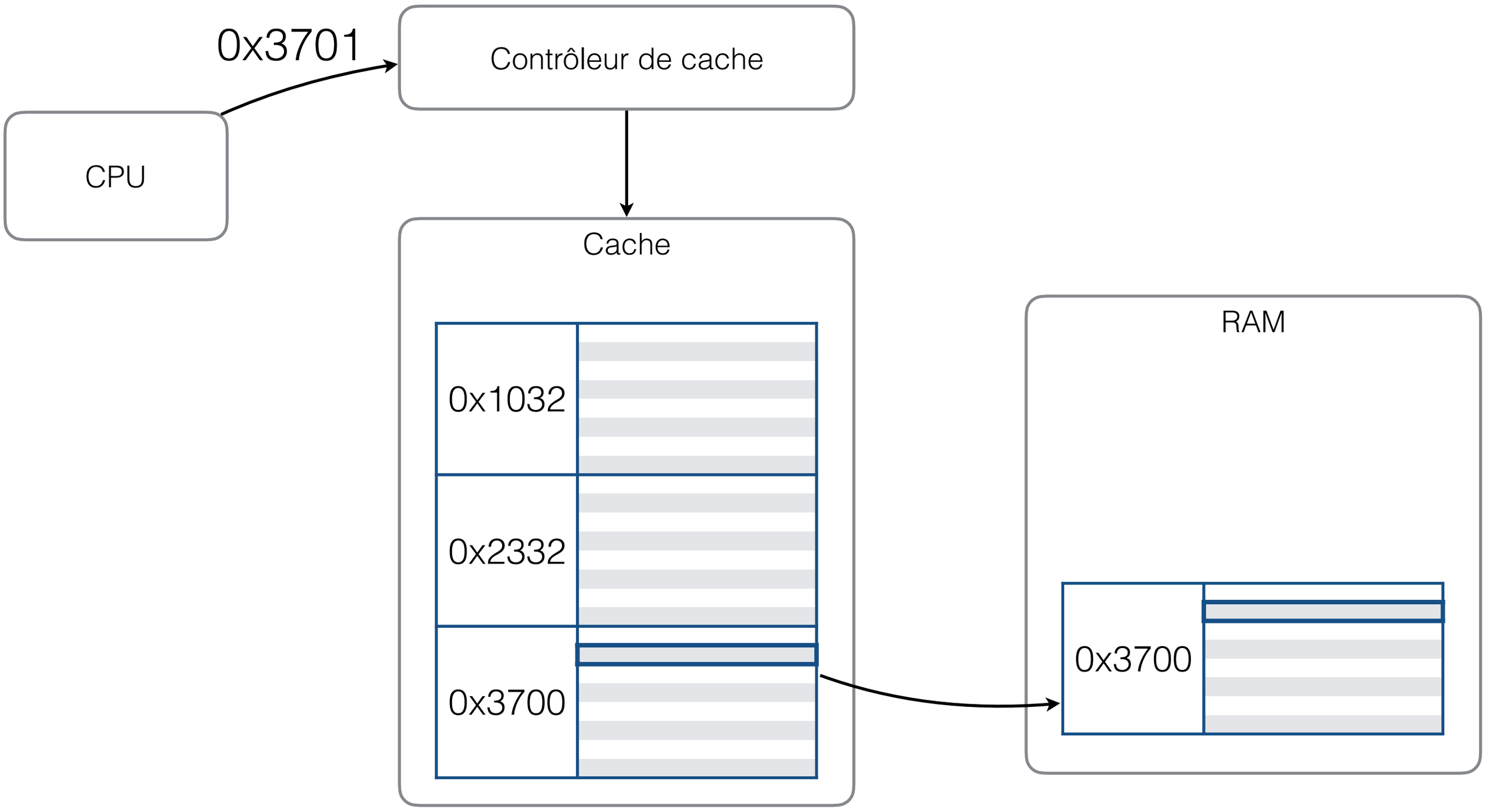
Écriture en cache « write-through »



Écriture en cache « write-through »

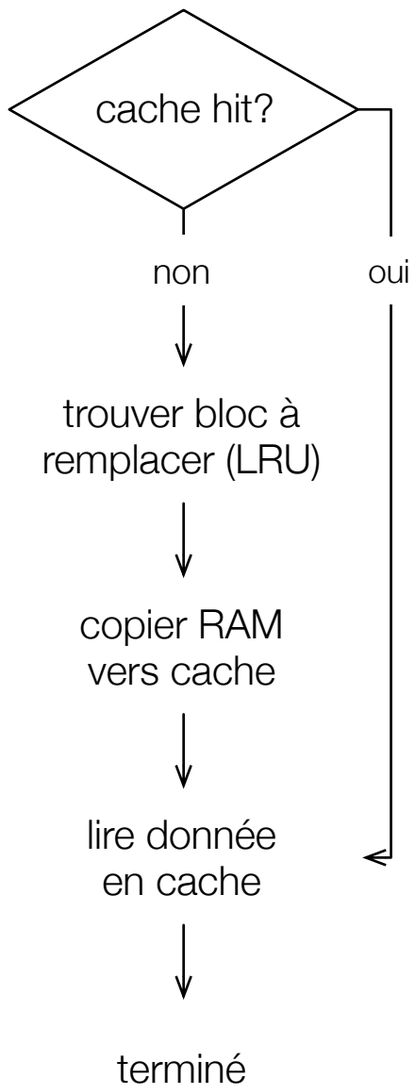


Écriture en cache « write-through »

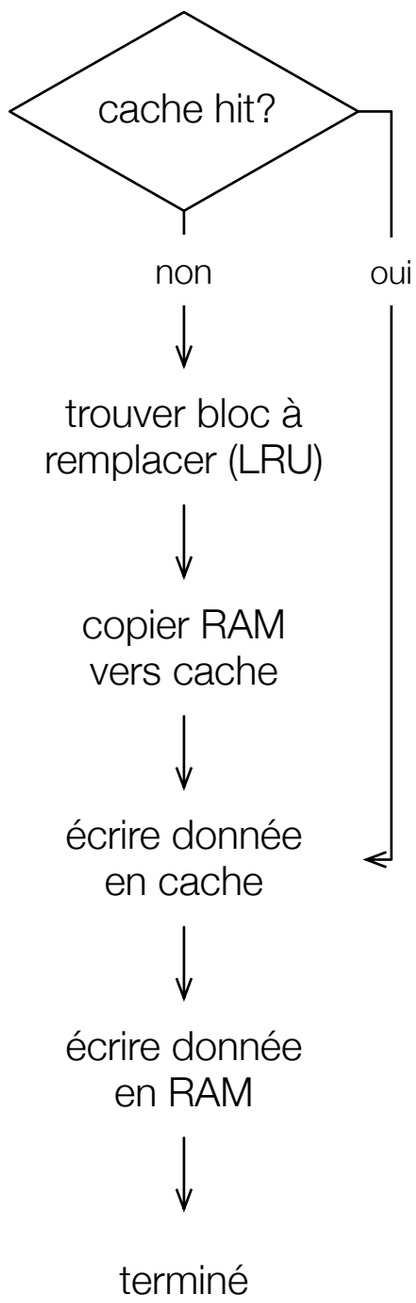


Cache « write-through »

Lecture



Écriture



Exercice #1

- Décrivez les étapes nécessaires pour que le micro-processeur écrive une donnée en mémoire si:
  - l’adresse mémoire n’est pas dans la cache;
  - le système utilise la stratégie « write-through ».

Exercice #1 — solution

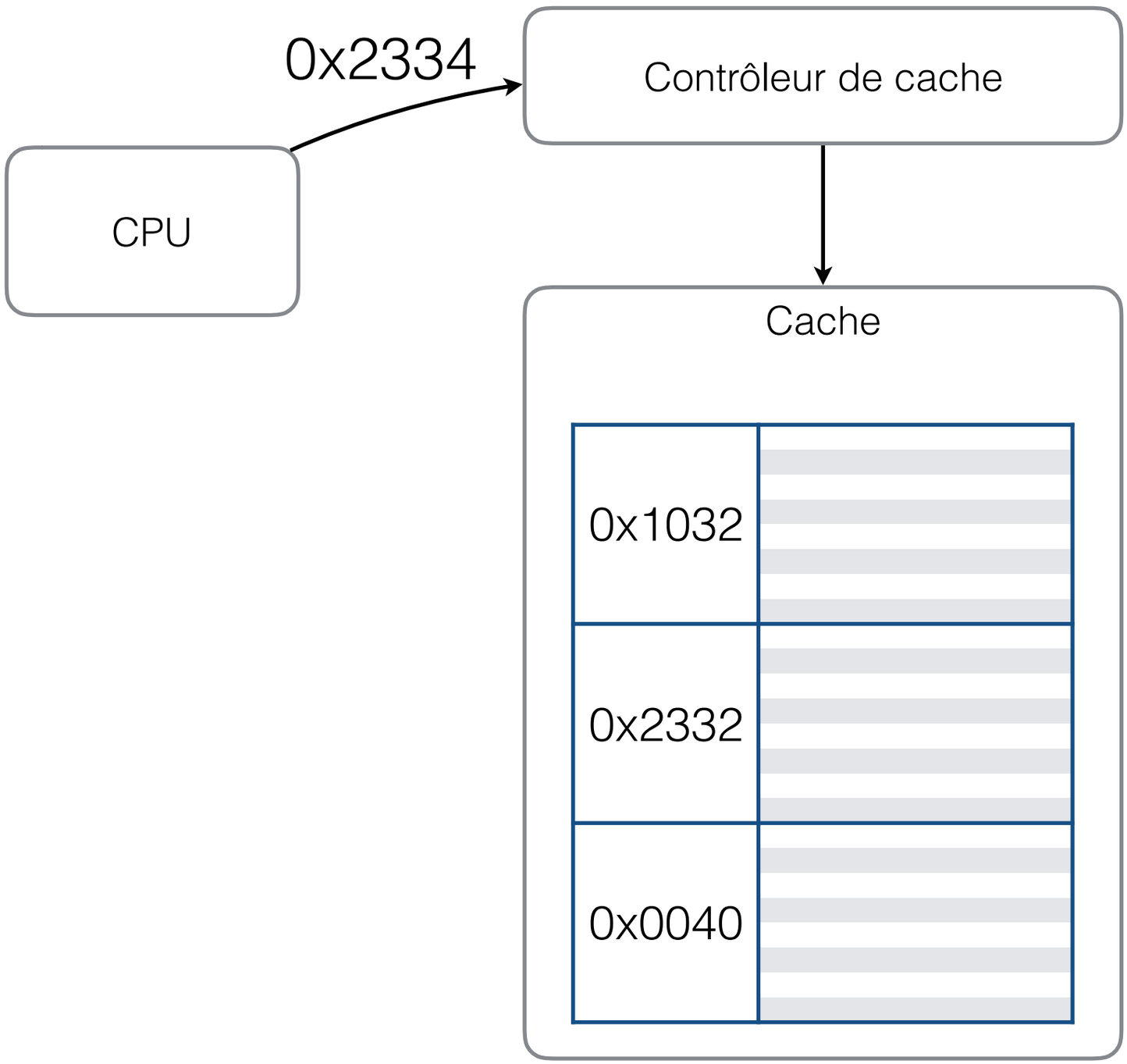
- Étapes:
  - « miss » en cache
  - trouver bloc à remplacer
  - copier bloc RAM vers cache
  - écriture en cache
  - écriture en RAM

Écriture en cache

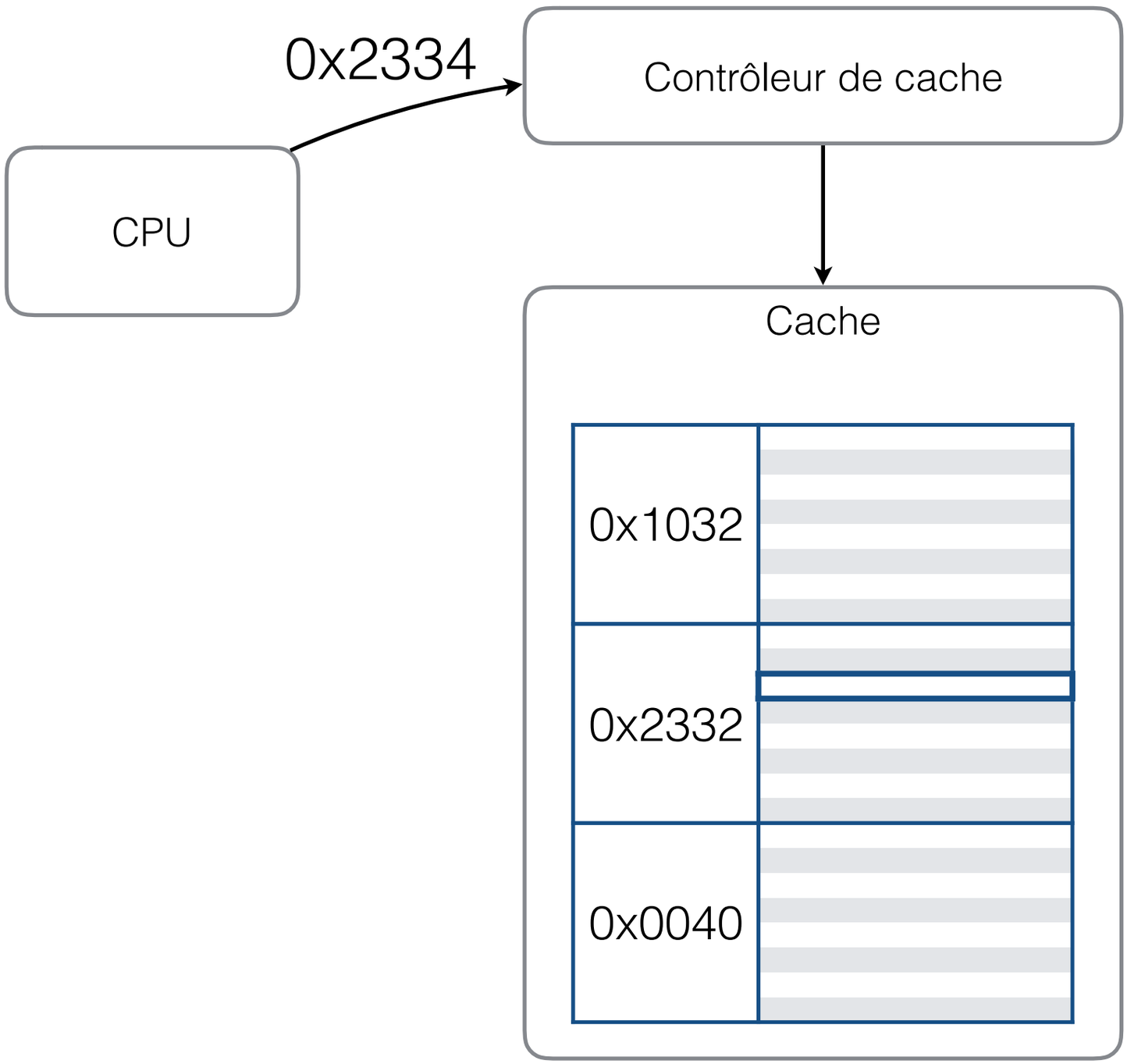
- Que faire lorsque l'on veut écrire en cache?

1. « Write-through » : écrire les changements dans la RAM au fur et à mesure
    - L'exemple précédent utilisait cette technique
2. « **Write-back** » : écrire le bloc de données en RAM *seulement lorsqu'il doit être remplacé*
    - Il faut donc se rappeler que le bloc doit être remplacé

Écriture en cache « write-back »

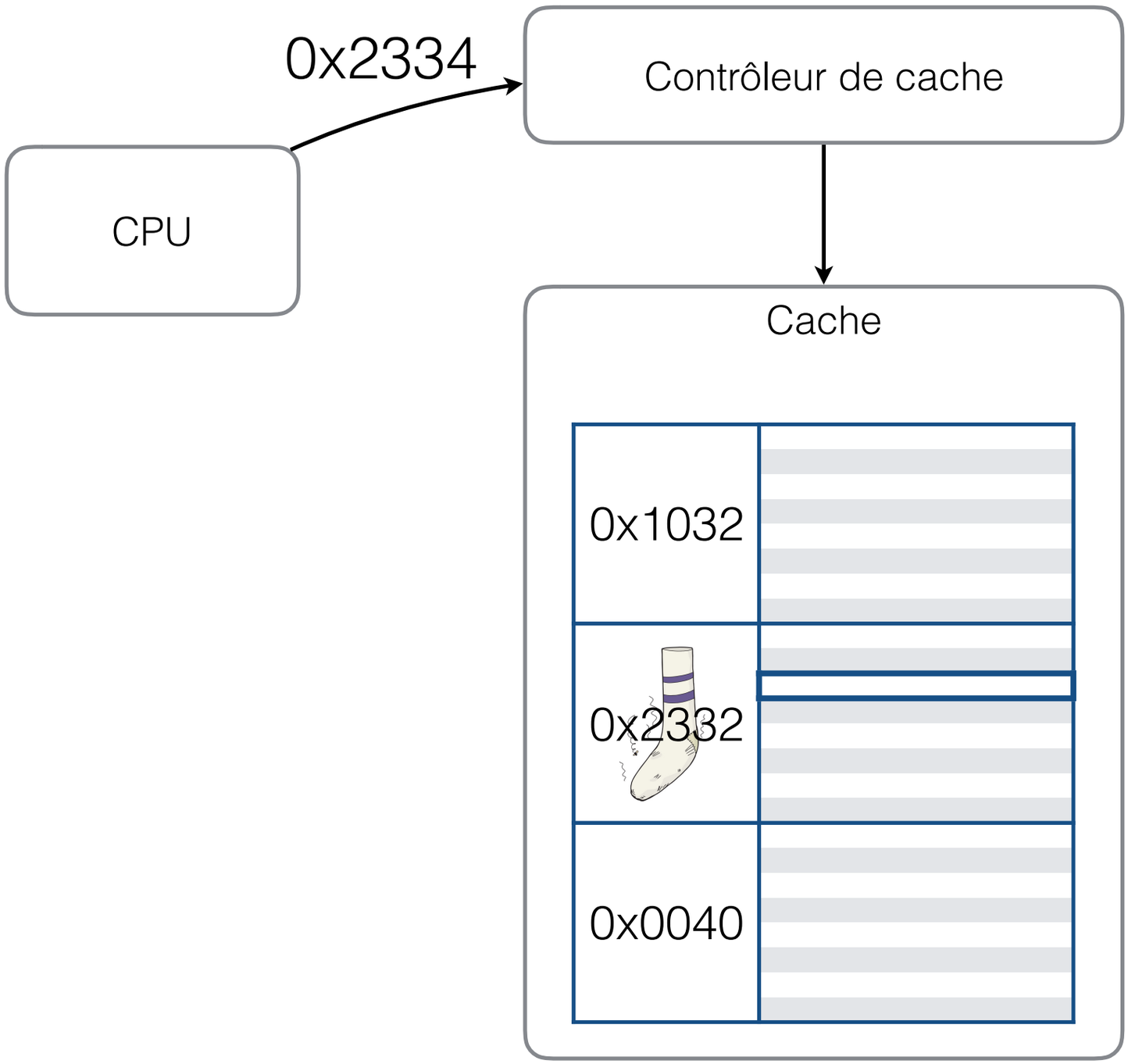


Écriture en cache « write-back »



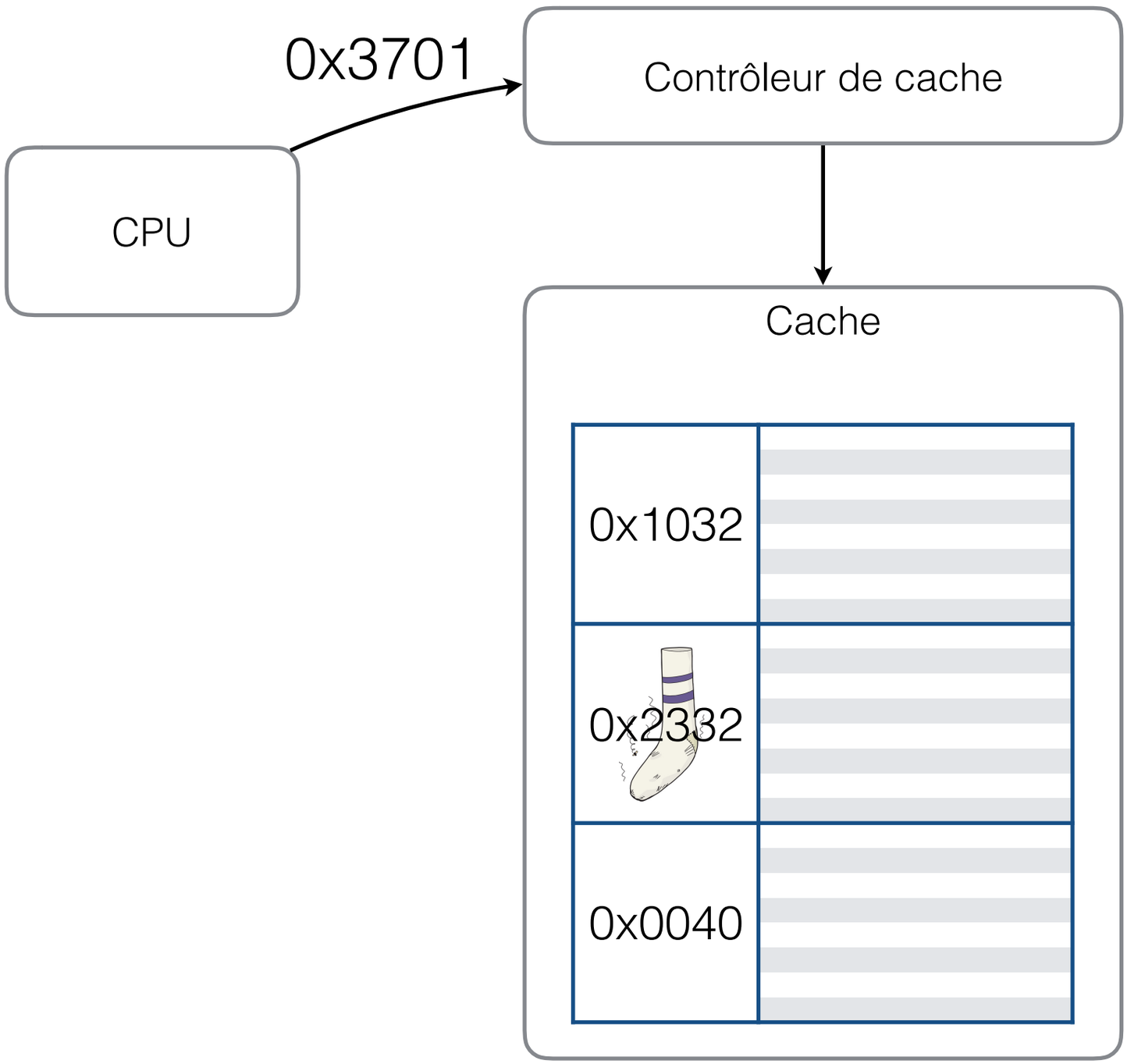
Hit!

Écriture en cache « write-back »



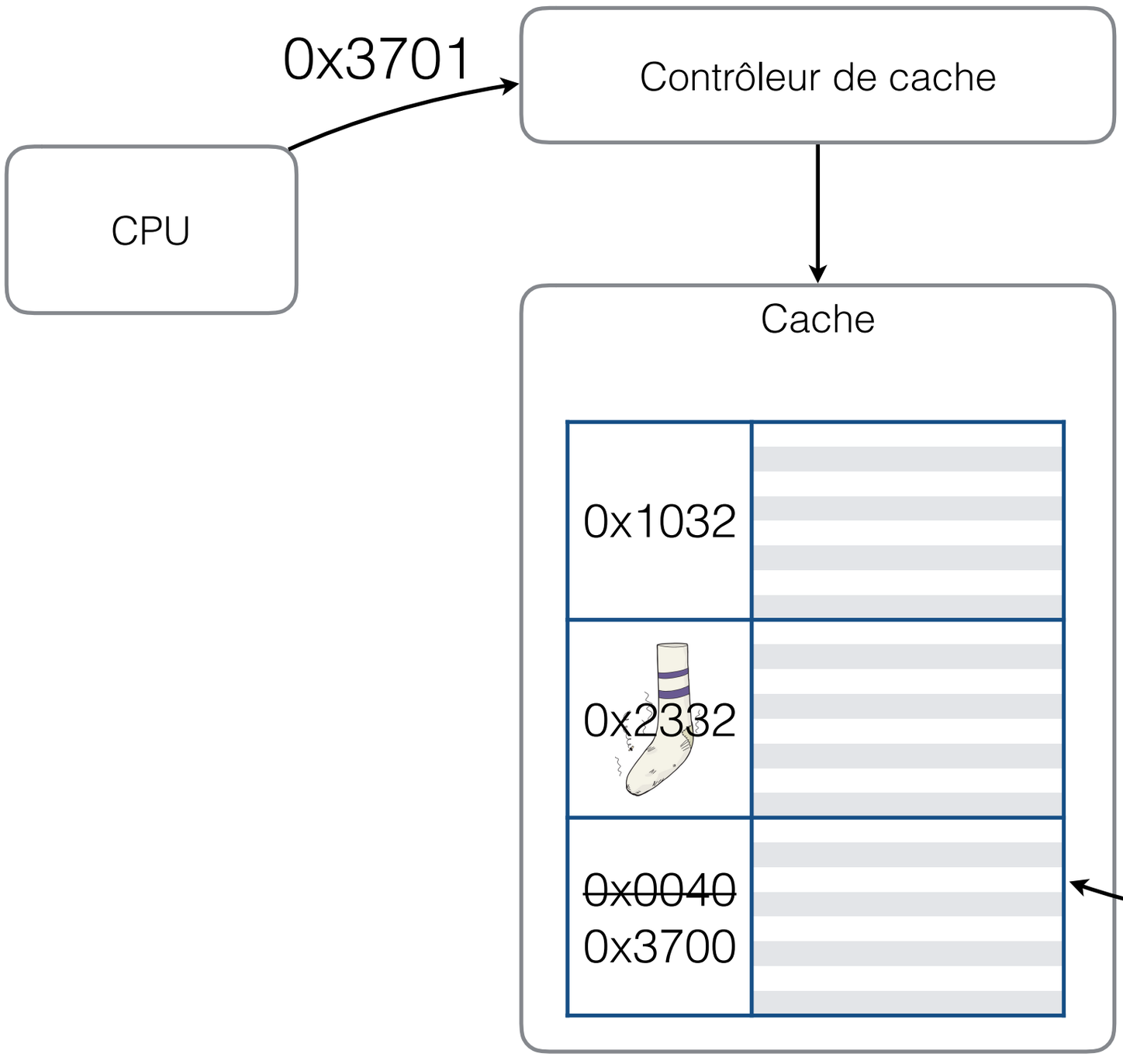
Le bloc est sale (« dirty »)!

Écriture en cache « write-back »



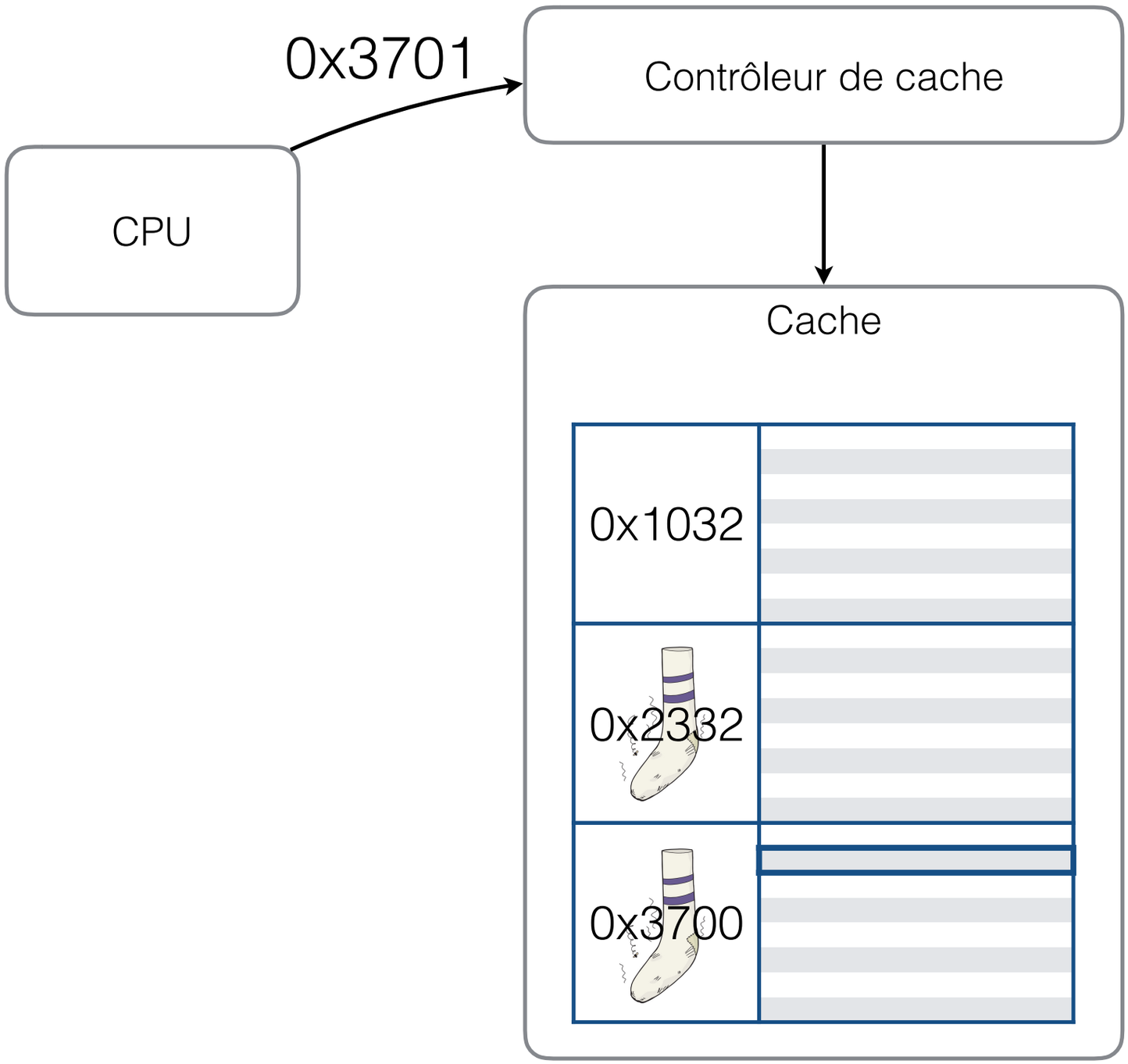
Miss!

Écriture en cache « write-back »



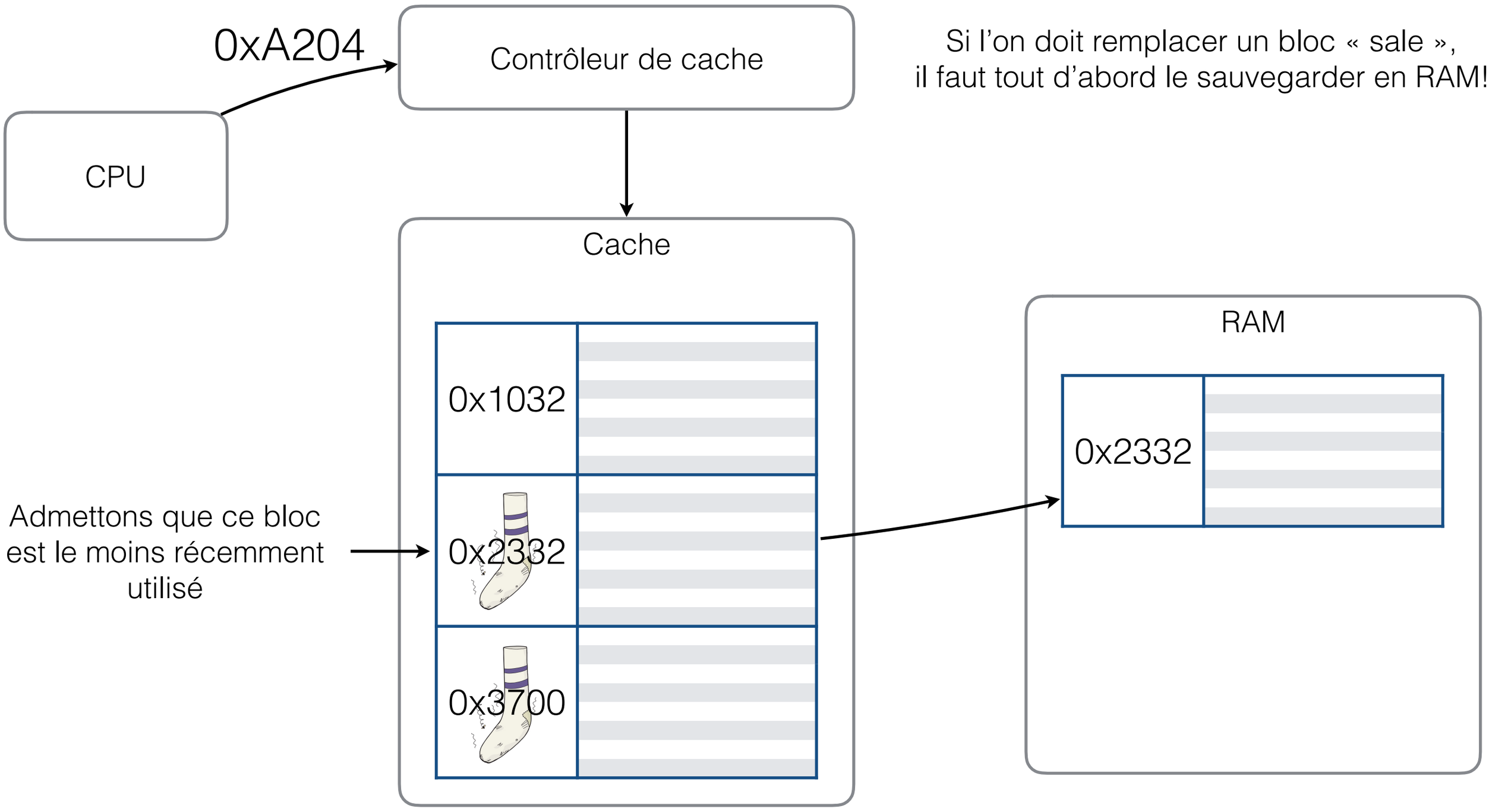
Le bloc à remplacer est propre, alors pas besoin de l'écrire en RAM avant de le remplacer!

Écriture en cache « write-back »



On peut ensuite écrire dans le bloc en cache, et l'identifier comme « sale »

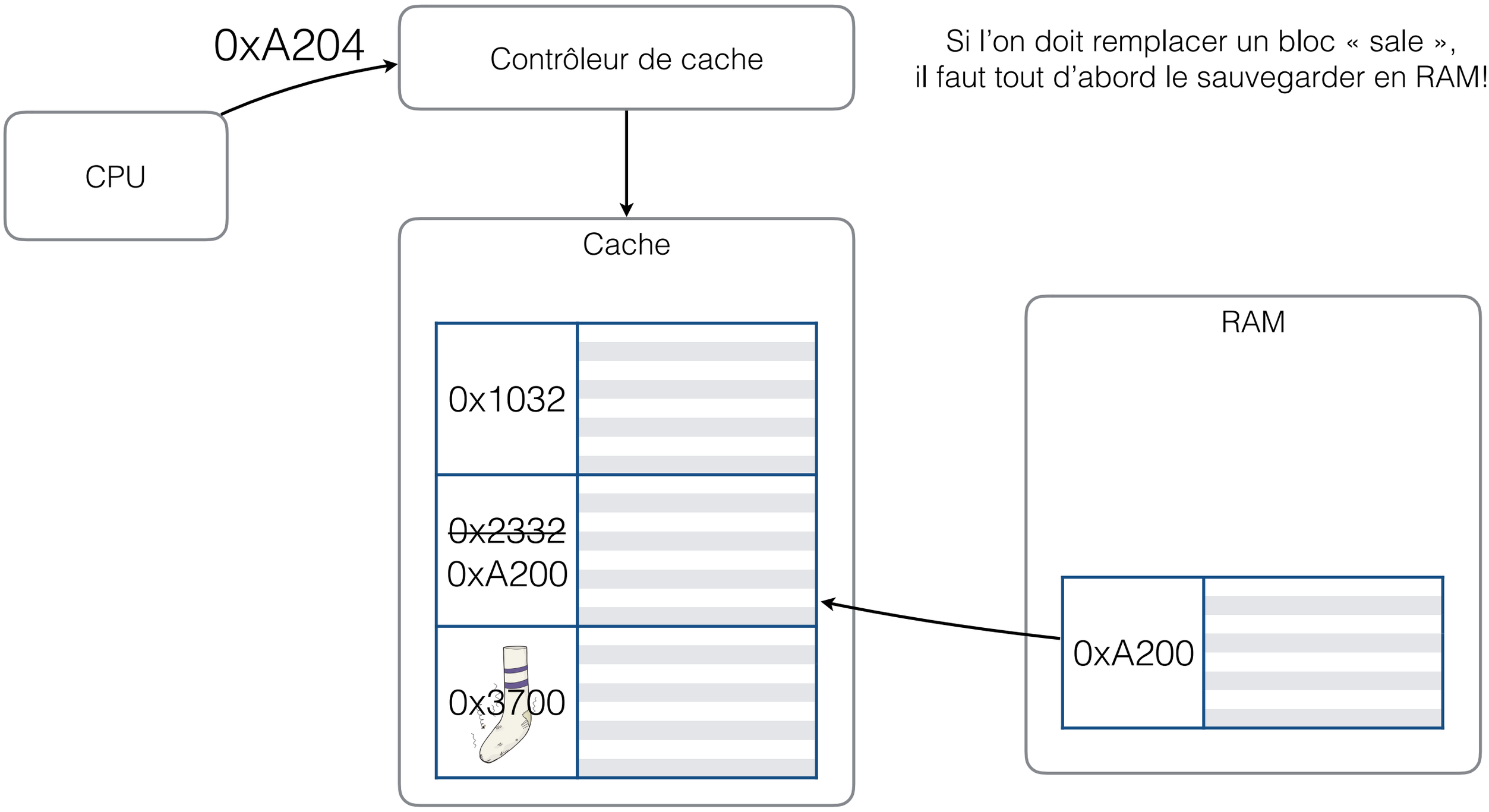
Écriture en cache « write-back »



Si l'on doit remplacer un bloc « sale »,
il faut tout d'abord le sauvegarder en RAM!

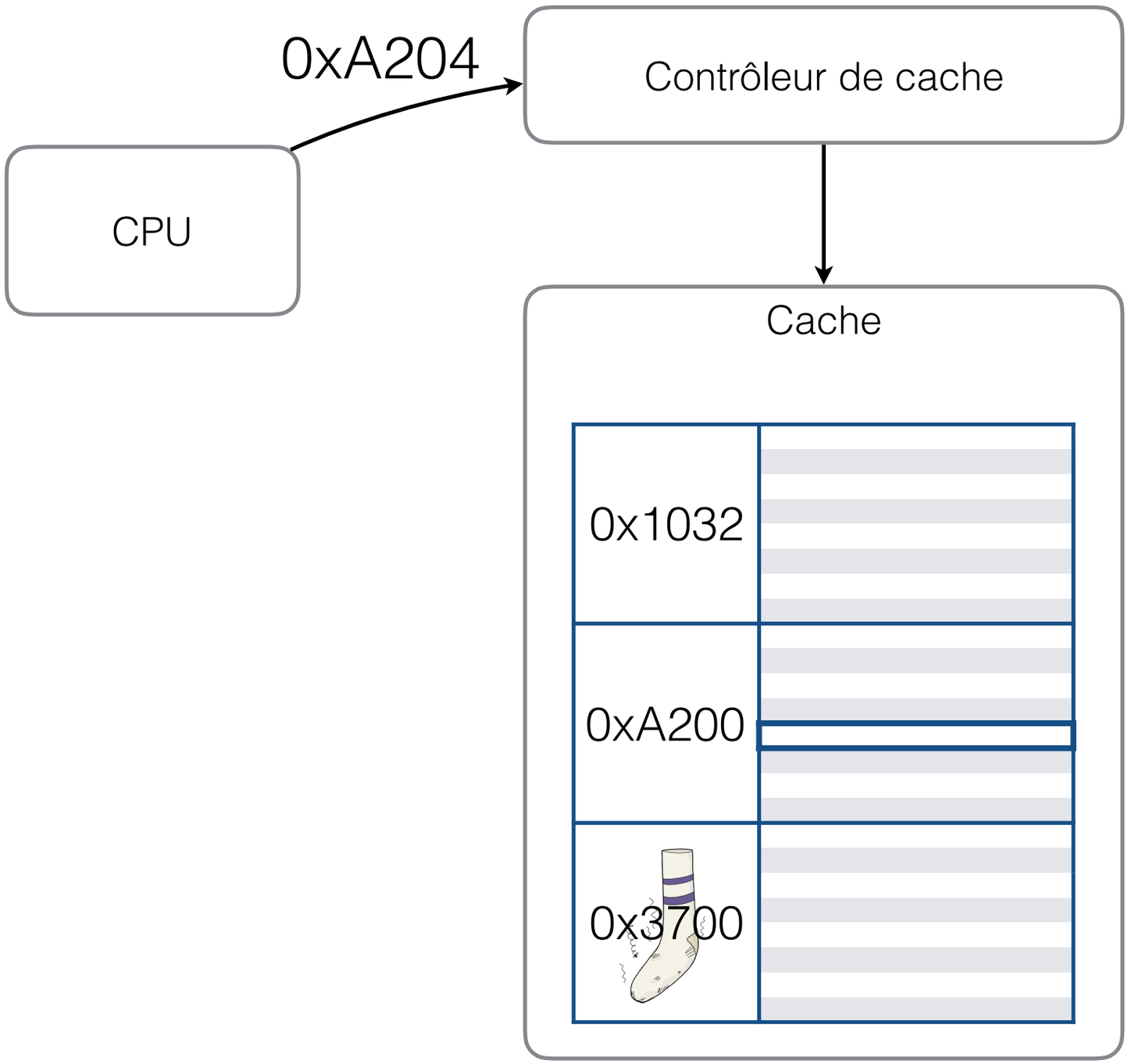
Admettons que ce bloc est le moins récemment utilisé

Écriture en cache « write-back »



Si l'on doit remplacer un bloc « sale »,
il faut tout d'abord le sauvegarder en RAM!

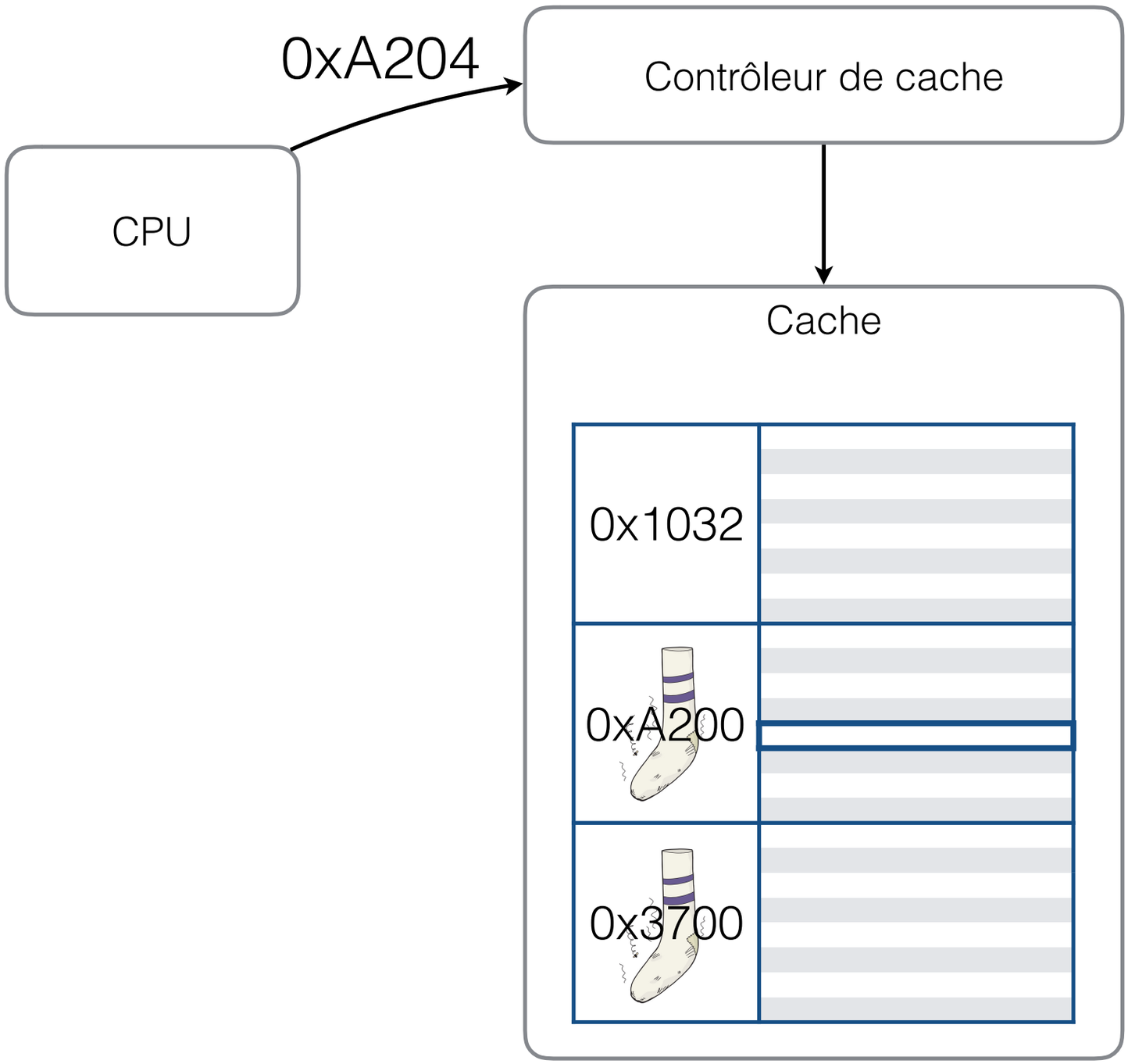
Écriture en cache « write-back »



Si l'on doit remplacer un bloc « sale »,
il faut tout d'abord le sauvegarder en RAM!

L'écriture peut être effectuée ensuite.

Écriture en cache « write-back »



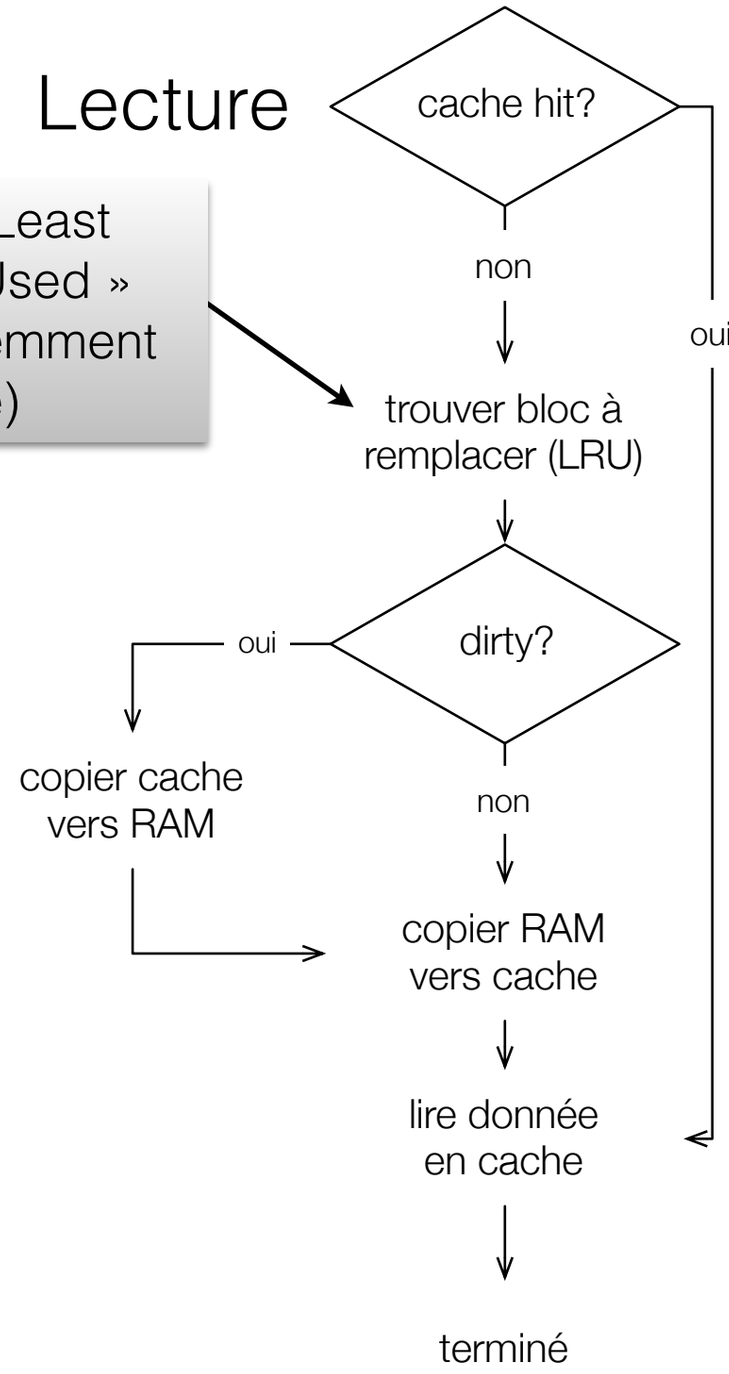
Si l'on doit remplacer un bloc « sale »,
il faut tout d'abord le sauvegarder en RAM!

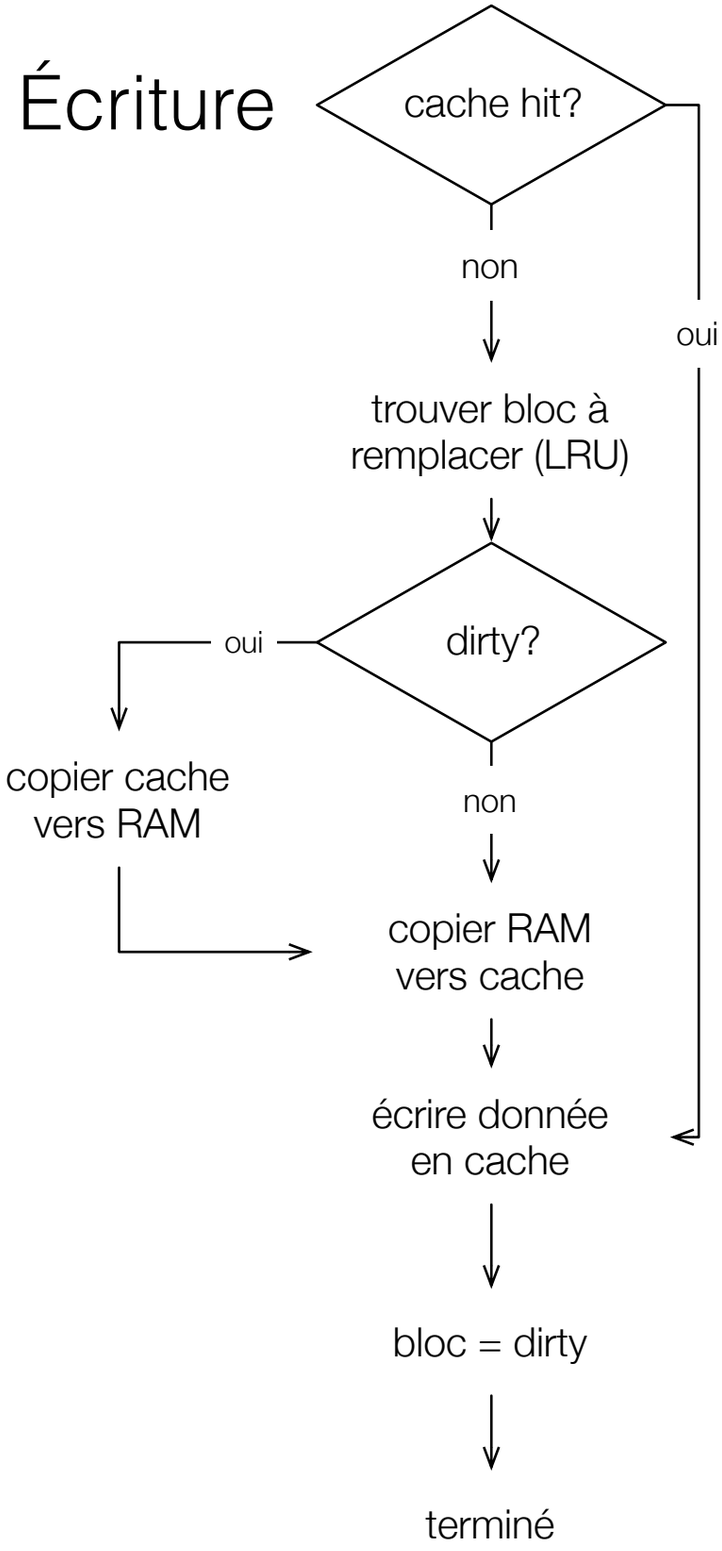
L'écriture peut être effectuée ensuite.

Et le bloc est identifié comme « sale »

Cache « write-back »

LRU = « Least Recently Used » (moins récemment utilisé)





Récapitulation

Write-through

Mets la RAM à jour
à chaque écriture

Write-back

Cache « hit » vs « miss »

- Lorsqu'il veut une donnée, le microprocesseur cherche d'abord dans la cache.
  - Si la donnée s'y trouve, nous avons un « hit ».
  - Si la donnée ne s'y trouve pas, nous avons un « miss »
    - La donnée est transférée de la mémoire vers la cache. Les données adjacentes sont également transférées.
- Le « hit ratio » est la probabilité de retrouver une donnée dans la mémoire cache.

Principe de localité

- Phénomène par lequel les *emplacements mémoire adjacents* sont fréquemment accédés
  - Ex: les instructions d'un programme sont toutes adjacentes en mémoire

- Survient « naturellement » (instructions d'un programme), mais nous avons un rôle à jouer!

La cache L1

- La cache L1 est habituellement à l'intérieur du même circuit intégré que le microprocesseur, souvent imbriquée dans l'architecture même du processeur.

- Petite, car espace sur le microprocesseur limité (ex: 64Ko par coeur pour les Intel i7)

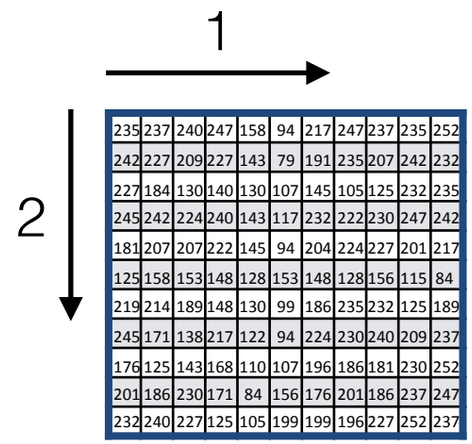
- Divisée en deux pour les données et pour les programmes.

- Très utilisée.

Retour

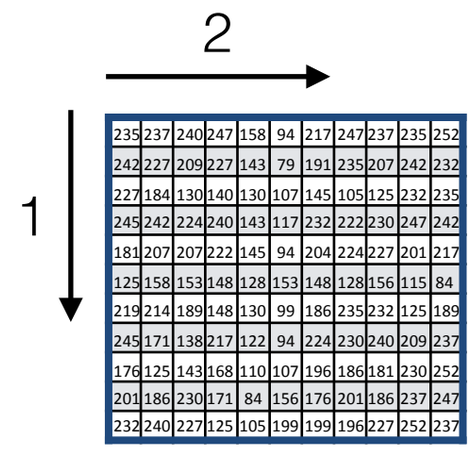
- Laquelle de ces deux versions est la plus rapide?

```
for (int i = 0; i < 256; i++) {
  for (int j = 0; j < 512; j++) {
    img[i*512 + j] = 0;
   }
 }
```



ou

```
for (int j = 0; j < 512; i++) {
  for (int i = 0; i < 256; j++) {
    img[i*512 + j] = 0;
   }
 }
```



i = lignes
j = colonnes

Ordre d’accès mémoire

1	2	3	4
5	6	7	8
9	10	11	12
13	14	15	16

1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16

```
for (int i = 0; i < 256; i++) {
  for (int j = 0; j < 512; j++) {
    img[i*512 + j] = 0;
   }
 }
```

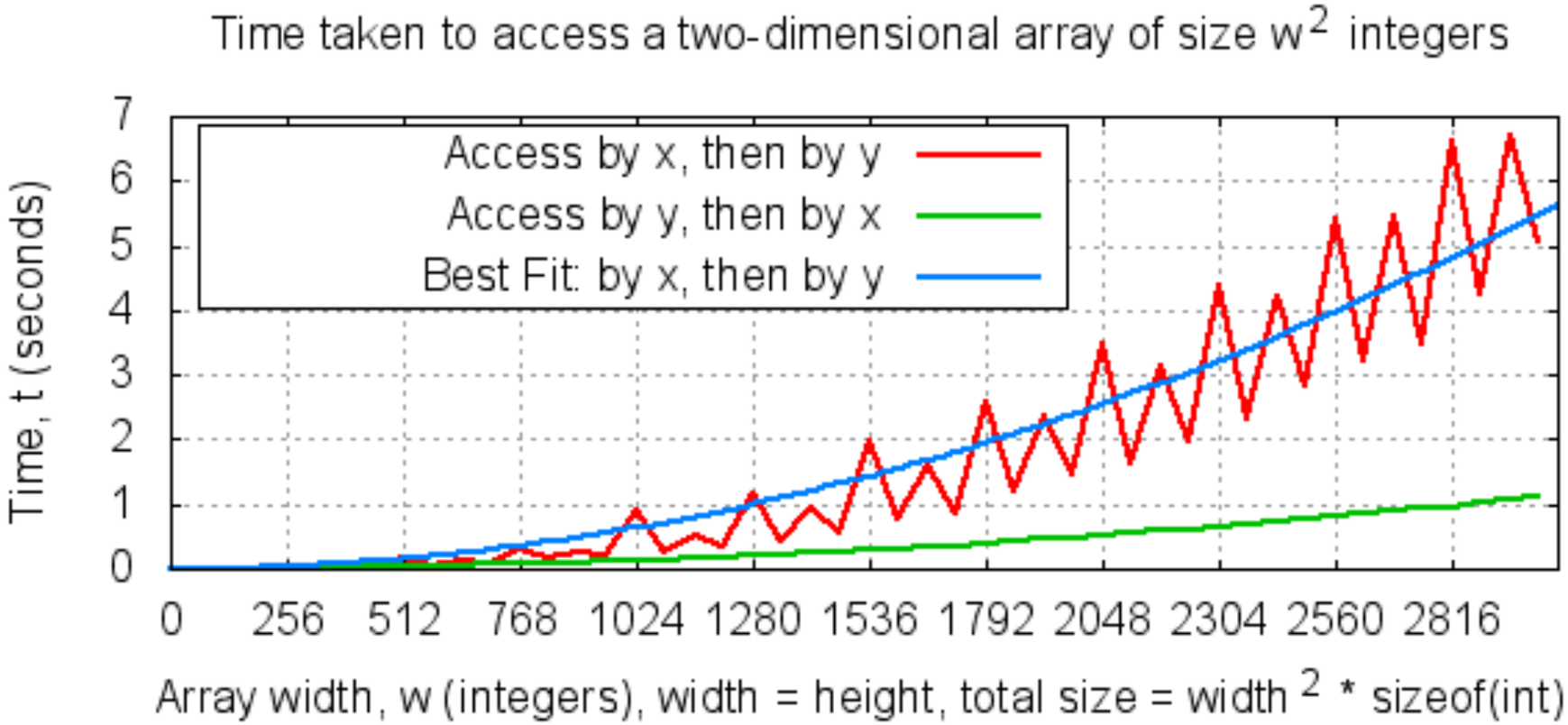
Ordre d’accès mémoire

1	2	3	4
5	6	7	8
9	10	11	12
13	14	15	16

1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16

```
for (int j = 0; j < 512; i++) {
  for (int i = 0; i < 256; j++) {
    img[i*512 + j] = 0;
   }
 }
```

Ordre d'accès mémoire

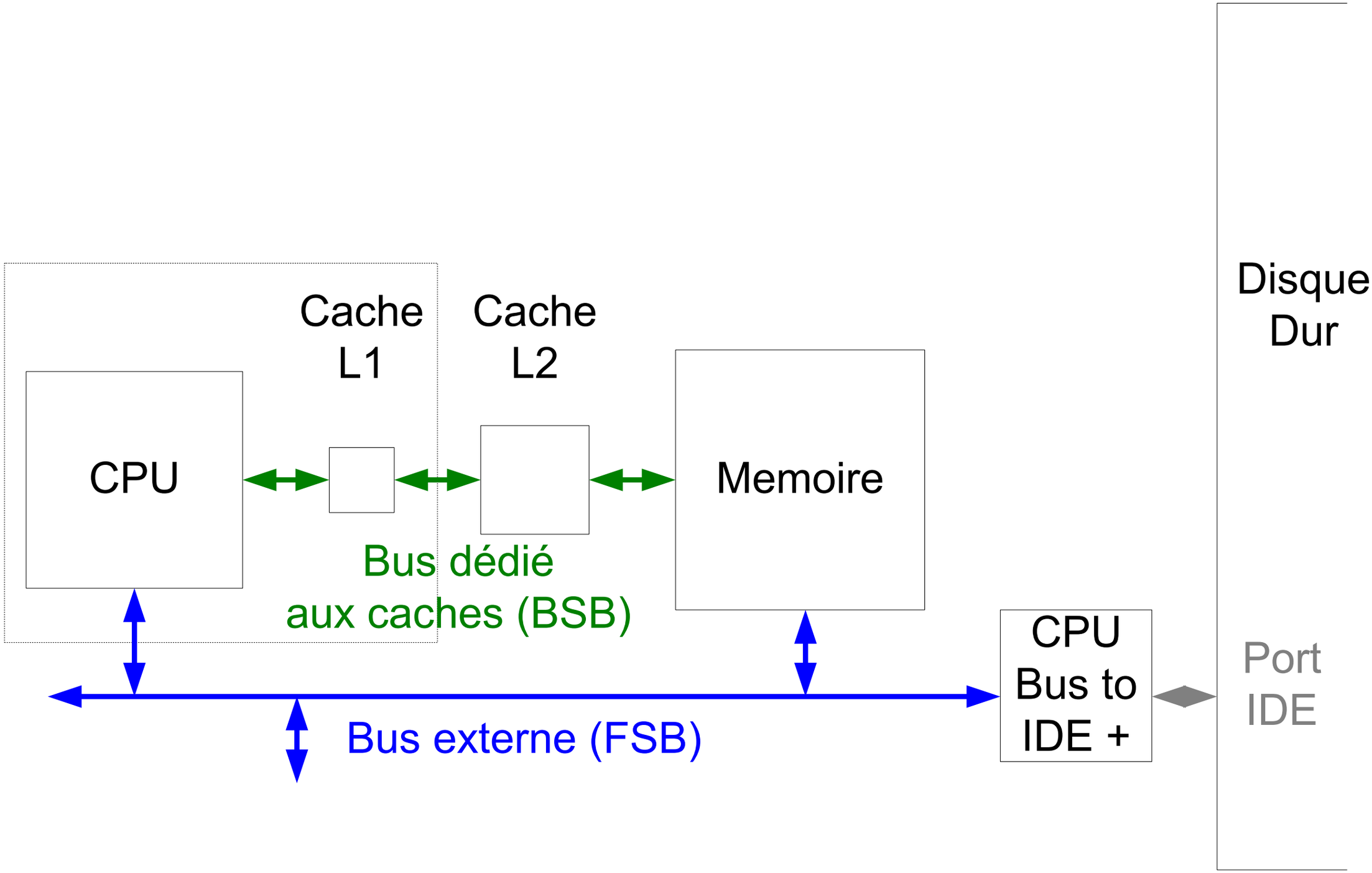


crédit: http://www.tophatstuff.co.uk/archive.php?p=119

Les caches L2, L3 et autres

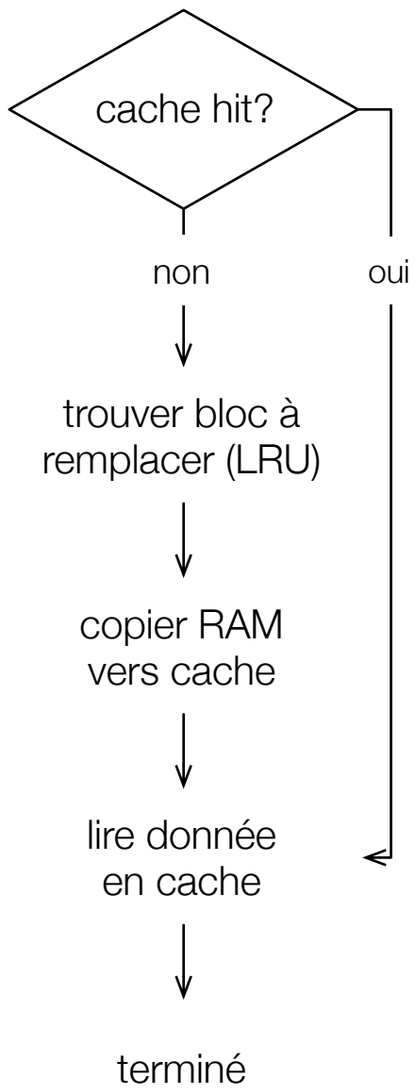
- Les ordinateurs modernes ont au moins deux niveaux de cache et souvent trois.
- La cache L2 est plus grosse que la cache L1 (256ko à 2Mo).
- Les caches L2 et L3 sont habituellement à l'extérieur du microprocesseur (mais de plus en plus à l'intérieur). Indépendantes de l'architecture du microprocesseur lorsqu'à l'extérieur.
- Habituellement, les données et les instructions sont gérées de la même façon dans les caches L2 et L3 (elles sont dites unifiées). Cependant, les instructions et les données sont de plus en plus souvent séparées dans les caches de niveau 2 (L2) voire de niveau 3.
- Moins rapide que L1 mais environ 10 fois plus rapide que la mémoire.
- Dans certains systèmes, il y a 4 niveaux de cache…
- Dans les ordinateurs modernes, il y a des caches dans les disques durs, et pour plusieurs périphériques. Ces caches contiennent les données du disque ou celles du périphérique qui les sont plus susceptibles d'être accédées prochainement.

Hiérarchie de caches

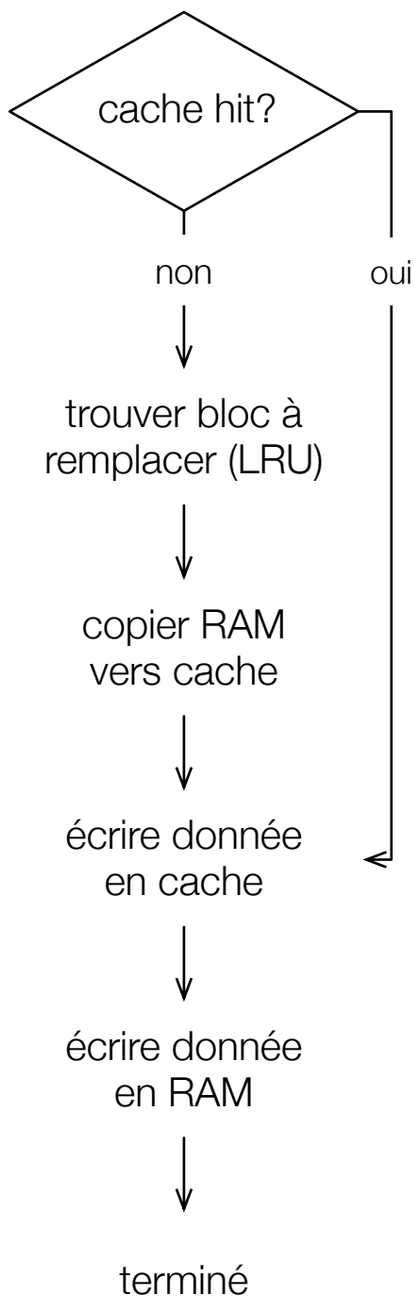


Cache « write-through »

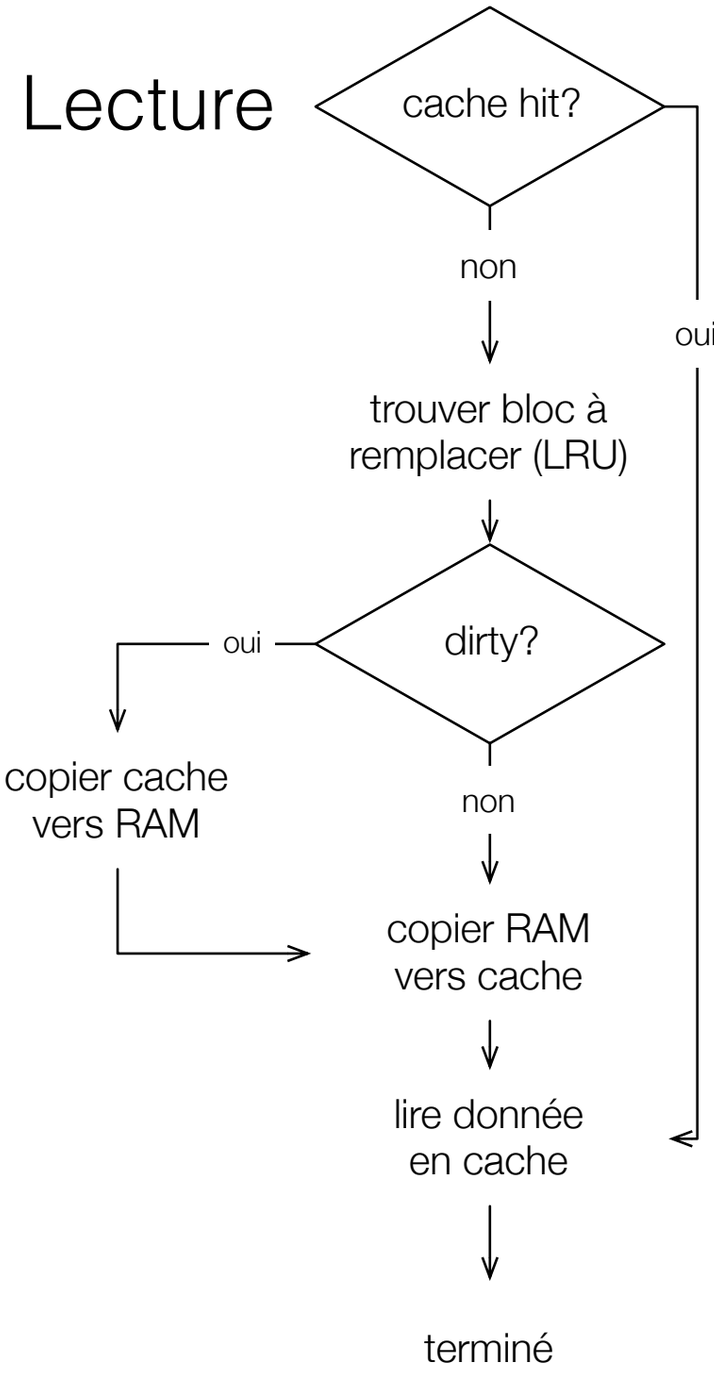
Lecture

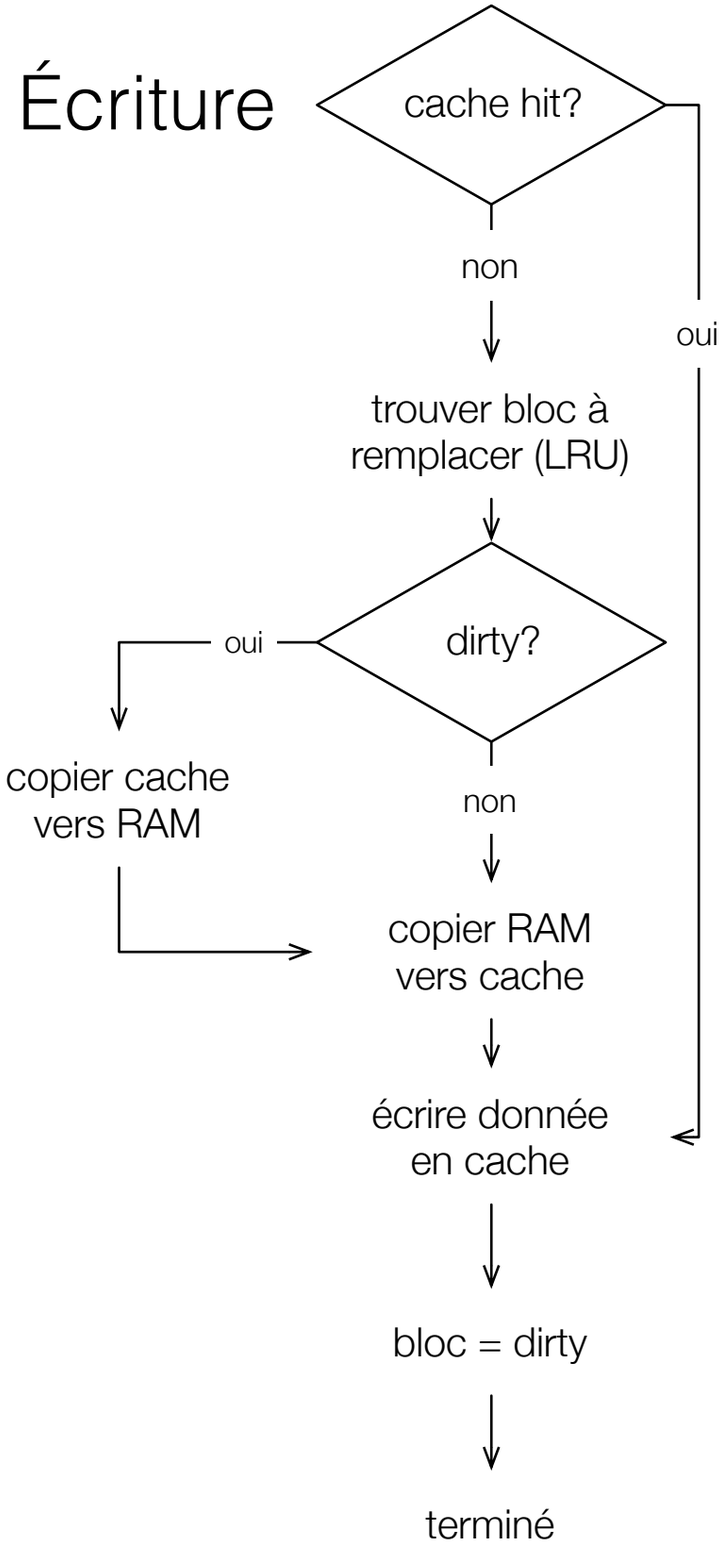


Écriture



Cache « write-back »





Exercice #2

- Décrivez les étapes nécessaires pour que le micro-processeur écrive une donnée en mémoire si:
  - l’adresse mémoire n’est dans aucune cache;
  - il y a deux niveaux de cache (L1 et L2);
  - le système utilise la stratégie “write-through”.

Exercice #2 — solution

- Étapes:
  - « miss » en cache L1
  - trouver bloc à remplacer en cache L1
  - copier bloc de cache L2 à L1
    - « miss » en cache L2
    - trouver bloc à remplacer en cache L2
    - copier bloc de RAM à L2
- écriture en cache L1
- écriture en cache L2
  - écriture en RAM

La mémoire principale, une cache du disque dur?

- La mémoire contient des données et des instructions regroupées en pages (souvent 4K).
- La mémoire fournit des informations aux caches.
- Si les données ne sont pas trouvées dans la mémoire (page fault), il faut chercher dans le disque dur (très long). La page contenant la donnée recherchée est habituellement transférée en mémoire. Elle remplace souvent une page déjà en place qu'il faut sauvegarder d'abord (swap page).

F.A.Q.

- Q. Où est la table des pages?
  - A. En mémoire
- Q. Combien d'accès mémoire doit-on faire pour accéder à des données?
  - A. 2: un pour accéder à la table des pages, un pour accéder aux données
- Q. Que faire pour éviter les accès mémoires doublés?
  - A. On utilise une "cache" spécialisée: le TLB!

Traduction d’adresse, MMU et TLB

- Le MMU d’une mémoire paginée est plus complexe que celui d’une mémoire allouée de façon contiguë. En effet, pour tous les accès à la mémoire, il faut accéder à la table de pages afin de décoder l’adresse à lire. Comme la table de page est souvent volumineuse, elle est elle-même en mémoire: il faut faire deux lectures de la mémoire pour aller chercher une instruction, ce qui est très long et quasi-inadmissible.

- Le TLB (“Translation Lookaside Buffer”), est une table de registres très rapides à l’intérieur du CPU. Le TLB contient quelques entrées de la table de pages, les dernières utilisées. Il s’agit d’une forme de cache. Pour faire de la traduction d’adresse, le MMU regarde d’abord si la page à traduire est dans le TLB. Puisque le TLB contient les dernières entrées de la page de table utilisées, alors le MMU y trouve presque toujours la page à lire. Lorsqu’un hit se produit dans le TLB, le temps d’accès à la mémoire n’est pas augmenté par la pagination.

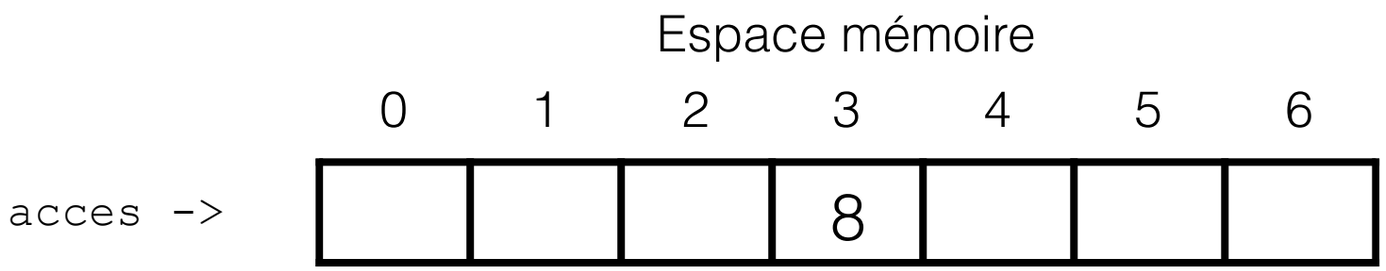
- Un TLB typique contient 8 à 4096 entrées de page de table. Lorsque la page est trouvée dans le TLB, ½ cycle à 1 cycle d’horloge est requis pour faire la translation d’adresse. Lorsque la page n’est pas dans le TLB, il faut faire deux accès mémoire… Le taux de succès dans le TLB est entre 99% et 99.99%!

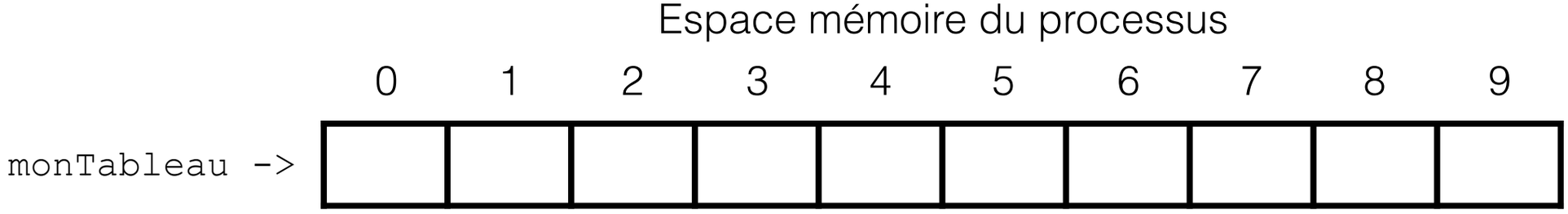
Meltdown

- Façon d’exploiter un ordinateur
  - prendre contrôle
  - copier des données protégées
- Exploite la façon dont les microprocesseurs sont conçus.
- Deux concepts clés pour en comprendre son fonctionnement:
  - les pipelines
  - les caches

https://meltdownattack.com

Meltdown



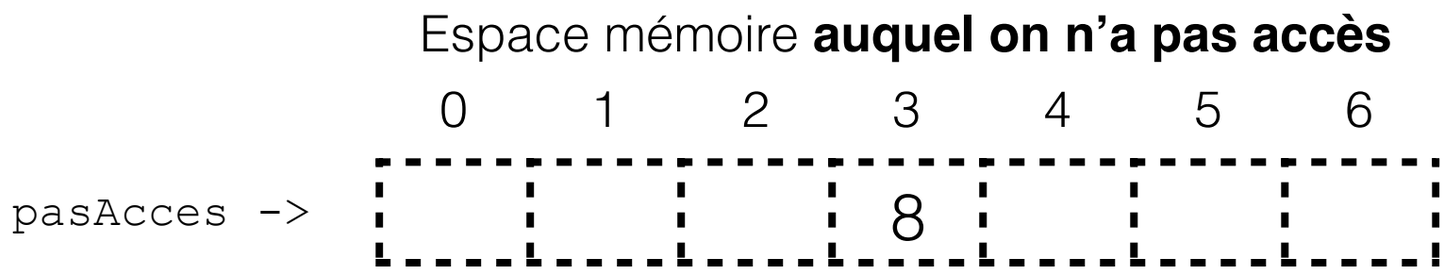


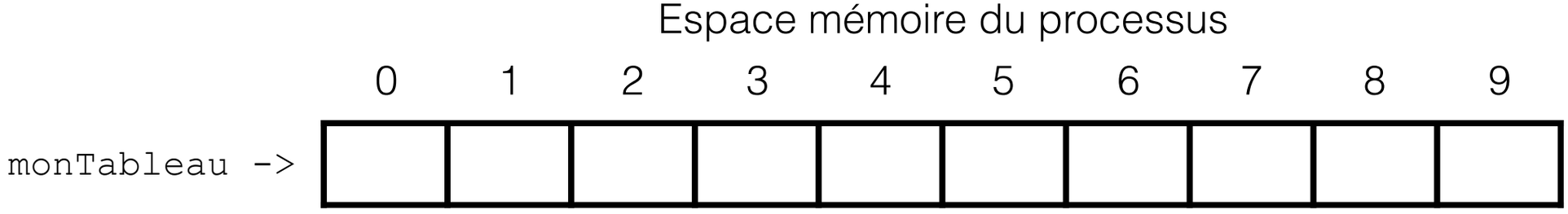
Code à exécuter

```
a = monTableau[acces[3]]
```

Ce code place le 8e élément de `monTableau` dans la variable `a`
(car le 3e élément de `acces` est 8).

Meltdown



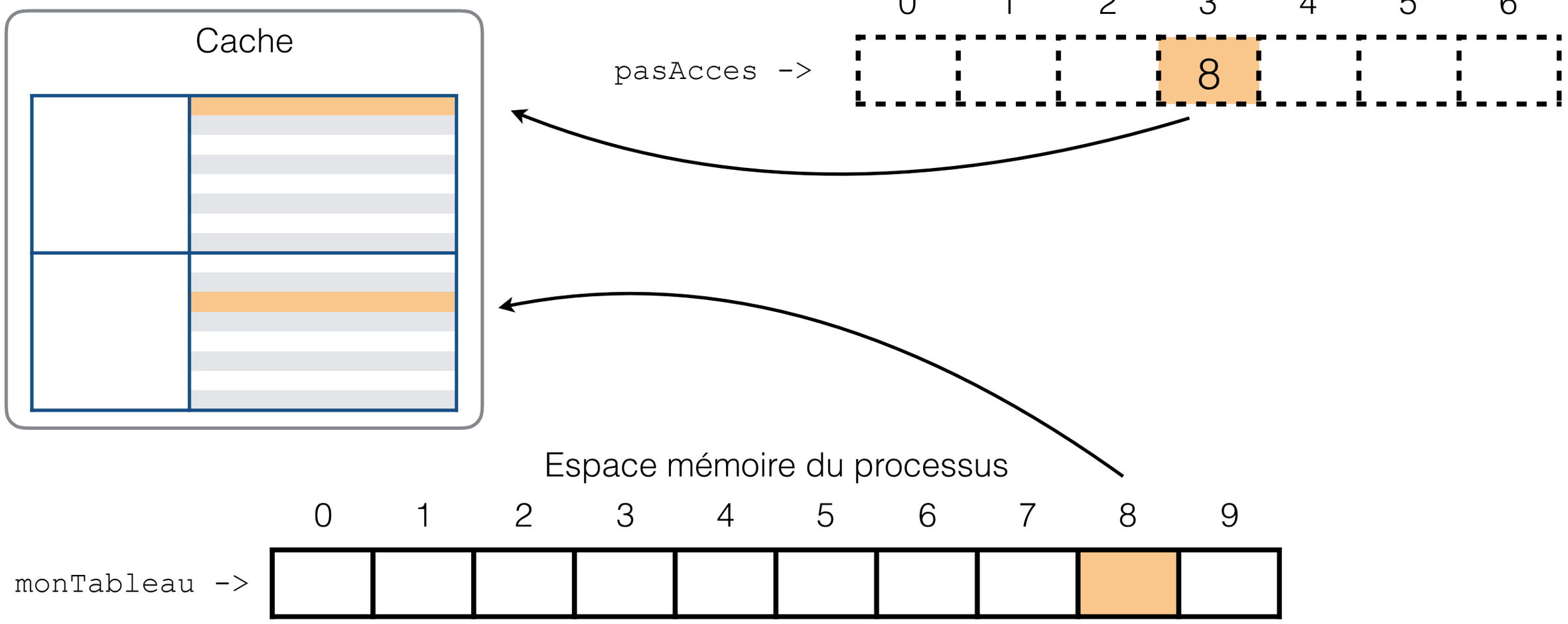


Code à exécuter

```
a = monTableau[pasAcces[3]];
```

Ce code soulève une exception! Nous n'avons pas accès au 3e élément de `pasAcces`…

Meltdown

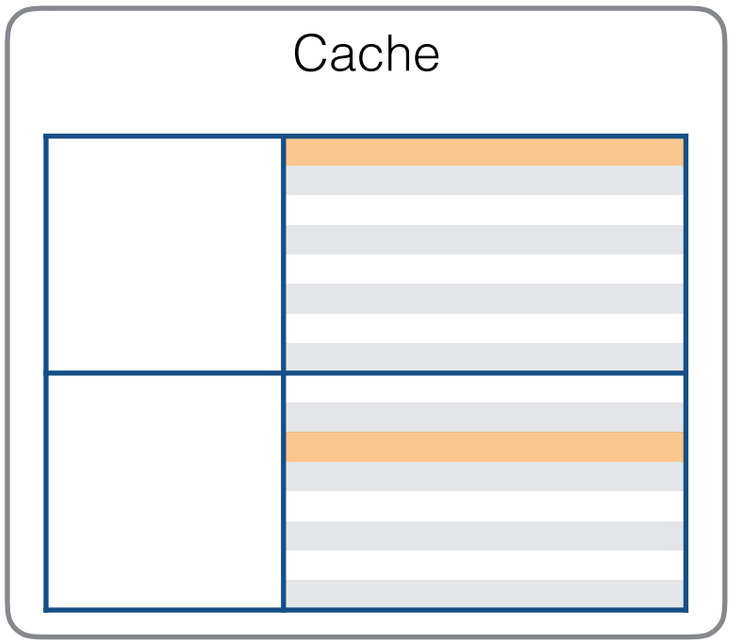


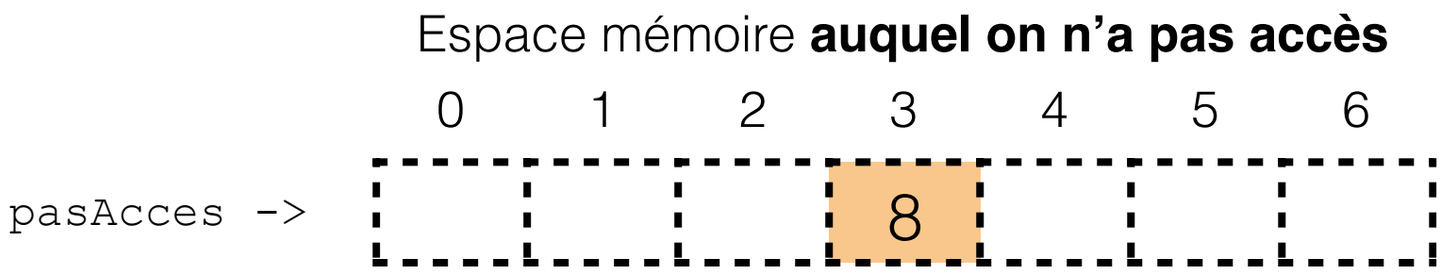
Code à exécuter

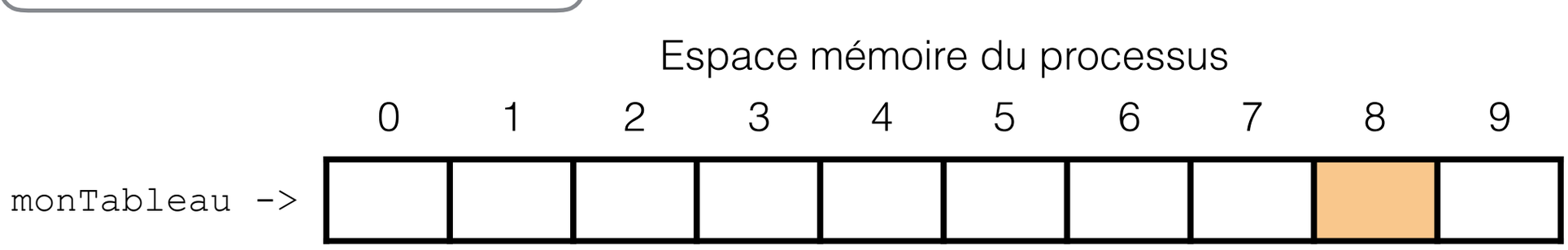
```
a = monTableau[pasAcces[3]];
```

Avant de soulever l'exception, la donnée (inaccessible) **et l'élément correspondant de monTableau** ont été mises en cache (lors de l'étape « pre-fetch » du pipeline)

Meltdown







Il ne reste plus qu'à boucler sur chaque élément de monTableau,
et de compter le temps que ça prend pour les accéder.

Quelle est la valeur de l'espace mémoire (supposément) inaccessible?

L'indice de monTableau qui est dans la cache, donc qui est accessible le plus rapidement!

Mémoire virtuelle vs cache

- Cache
  - accélérer la vitesse d’accès mémoire
- Mémoire virtuelle
  - augmente la “quantité perçue” de mémoire disponible
  - indépendant de l’architecture

Références et exercices

- Références
  - Irv Englander: chapitre 7 (jusqu'à 7.6), sections 8.3 (caches seulement), chapitre 9