Canaux auxiliaires temporels sur processeurs Intel x86

Benoît Morgan

IRIT, INP-ENSEEIHT Toulouse, University of Toulouse

March 30, 2021

Références

Outline

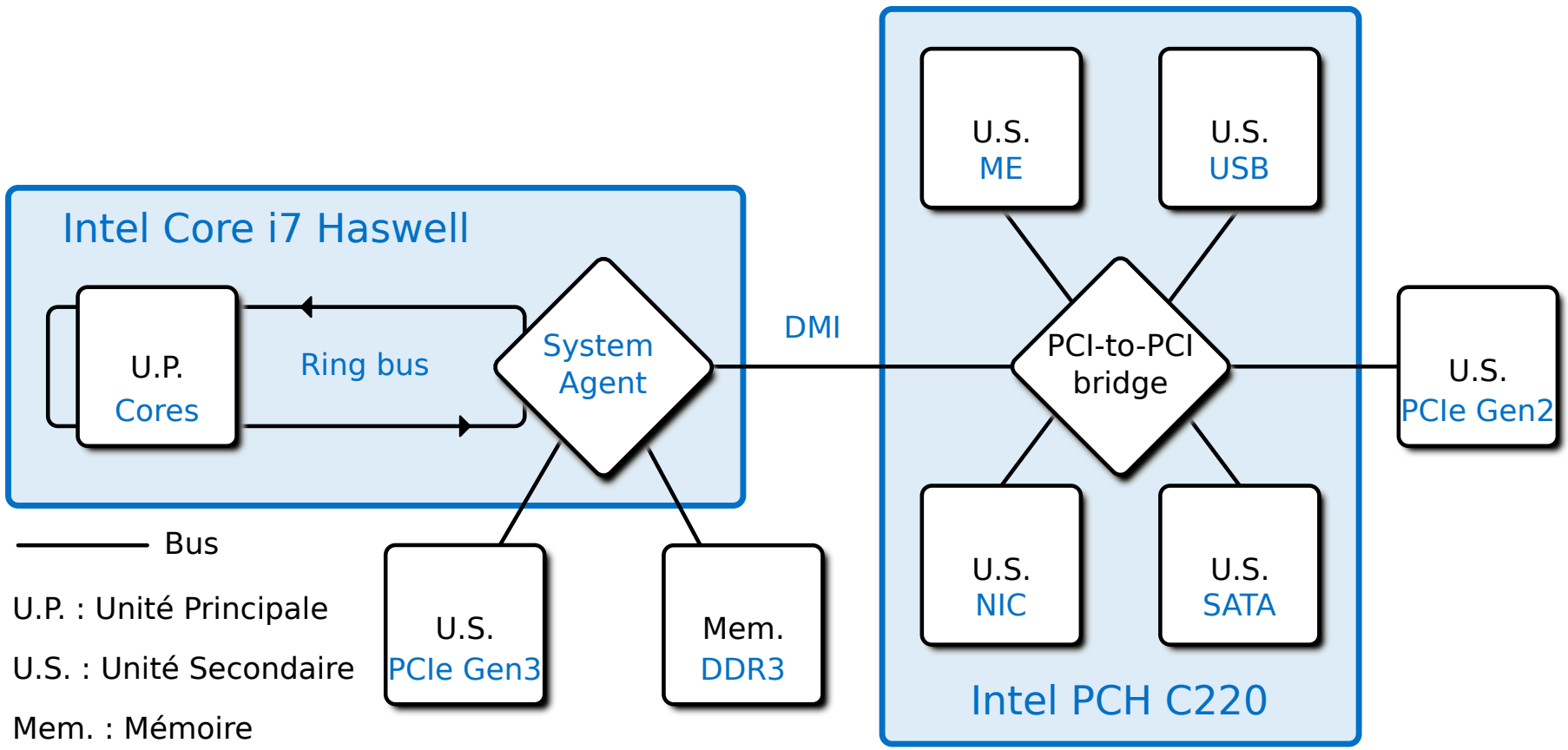
Mémoire cache

- Mémoire rapide d'accès proche du processeur
- Utile pour
  - Accès aux instructions
  - Accès aux données
  - Traduction d'adresse (mémoire virtuelle)
- Hierarchisée en niveaux
  - De plus en plus éloignés du processeur
  - De plus en plus gros en taille
  - De plus en plus partagés
  - Ayant des temps d'accès de plus en plus lent
- Limite les accès à des mémoire plus lentes

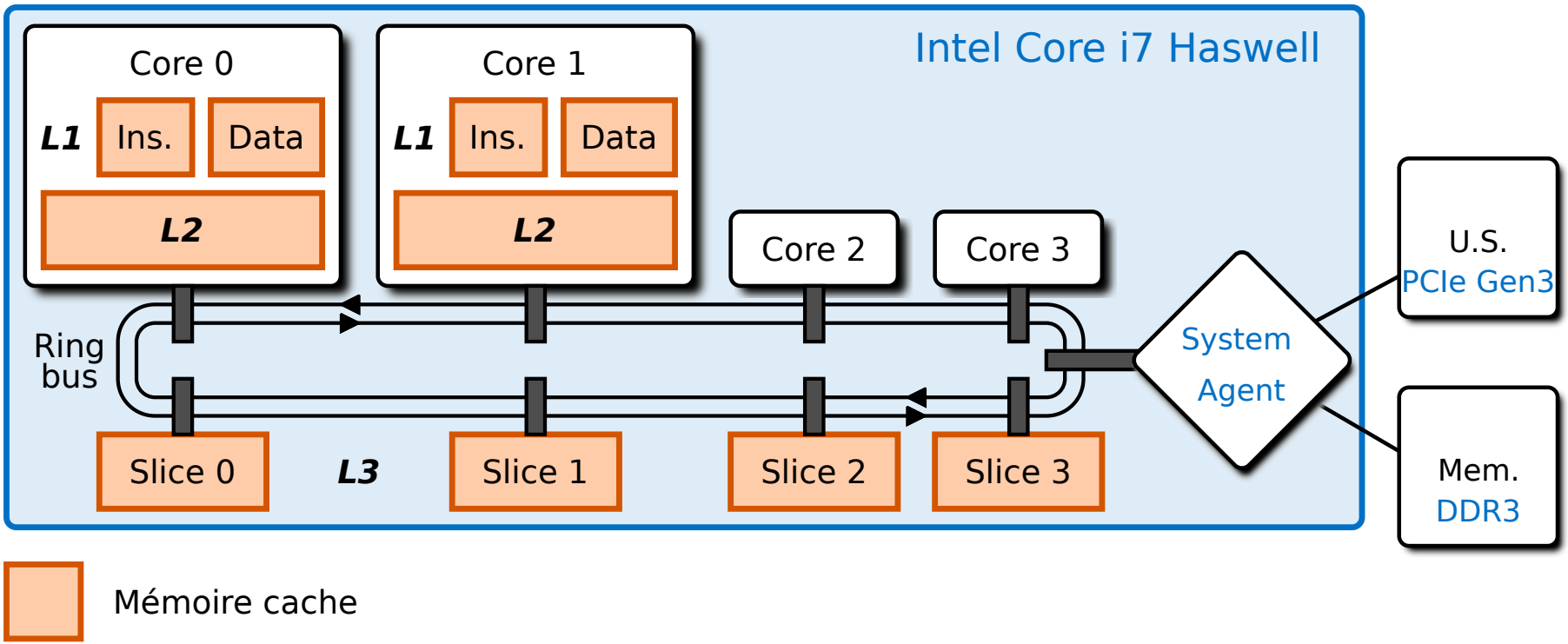
$\Rightarrow$ Augmente les performances du système

- RAM
- Périphériques

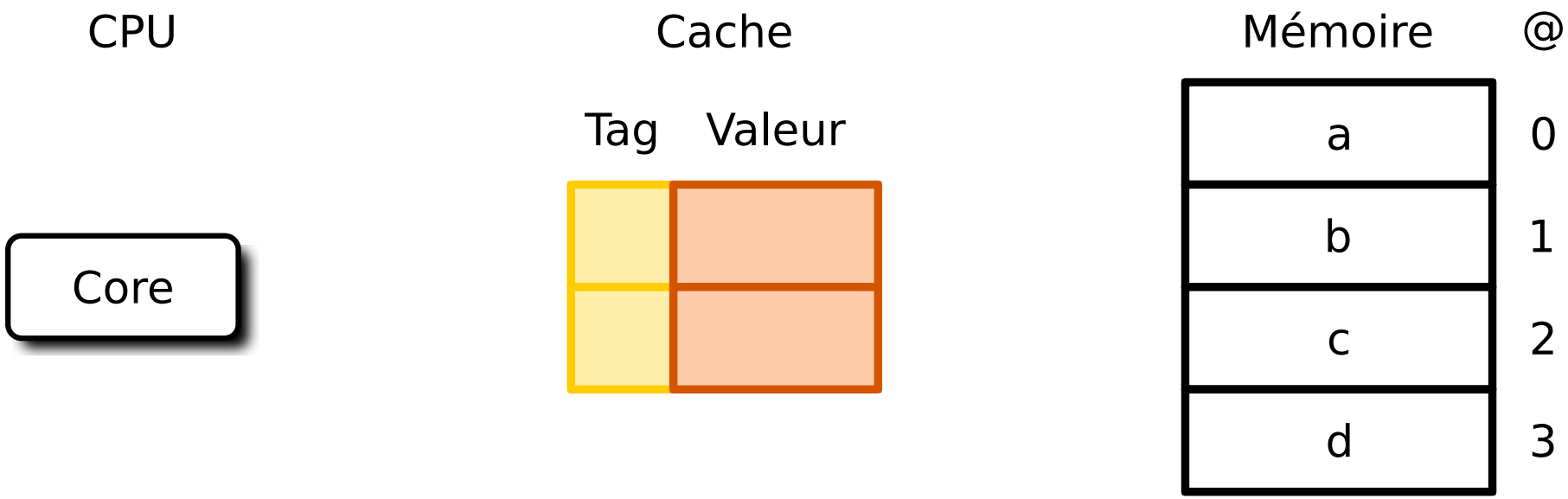
Architecture dans le cas d'Intel (1)



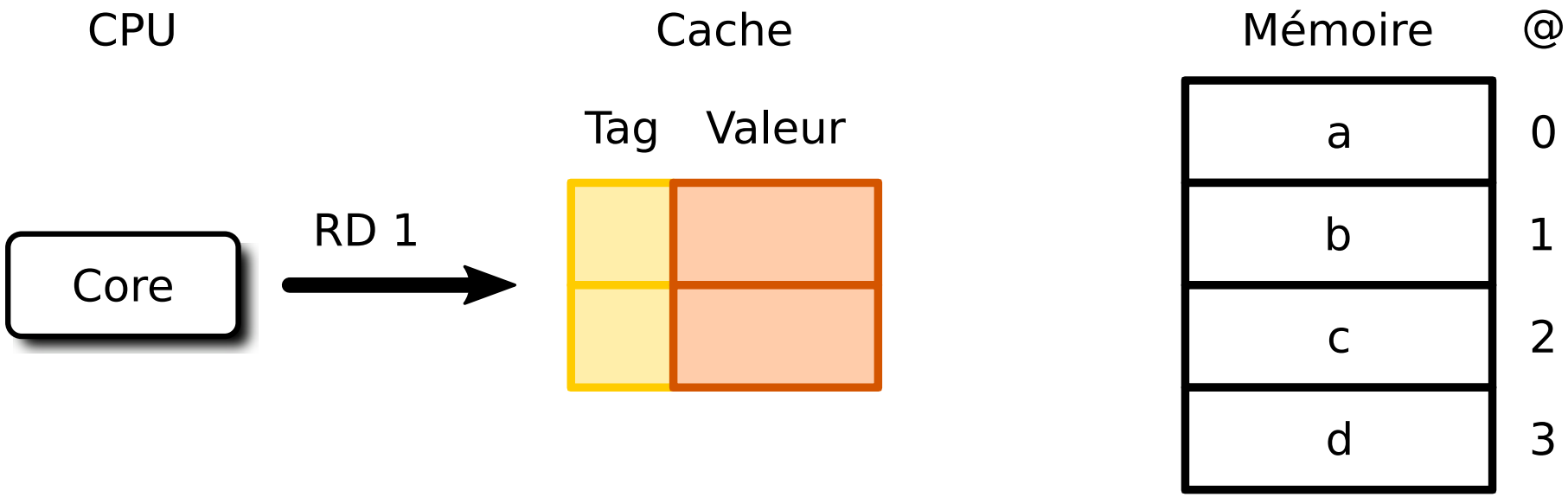
Architecture dans le cas d'Intel (2)



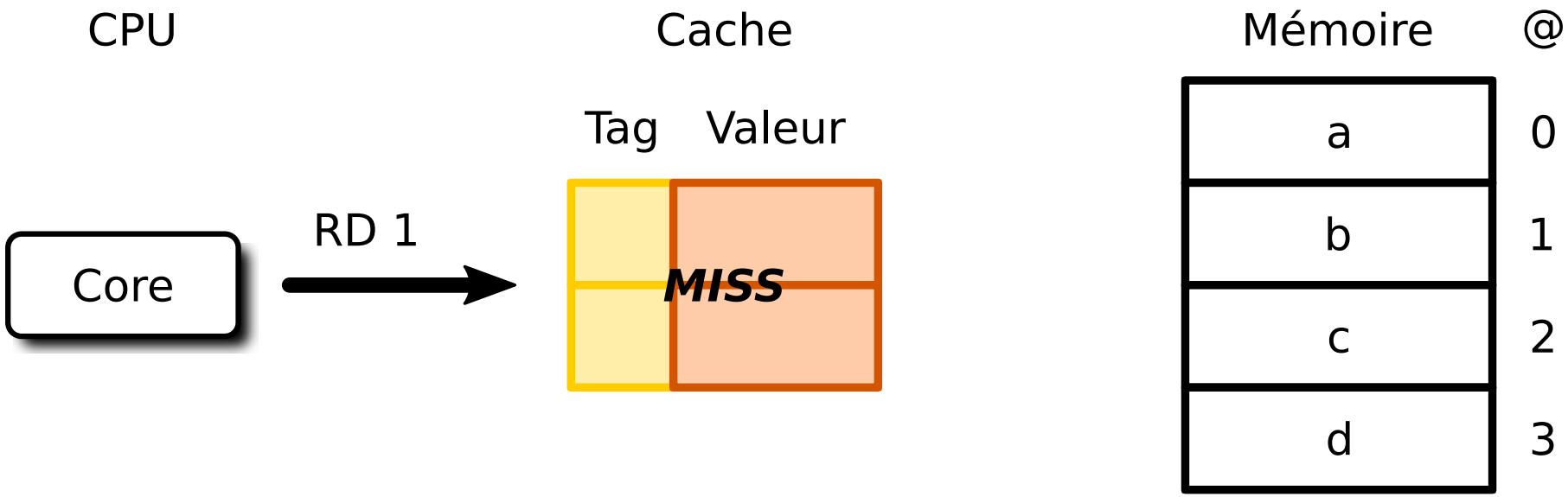
Fonctionnement basique (1)



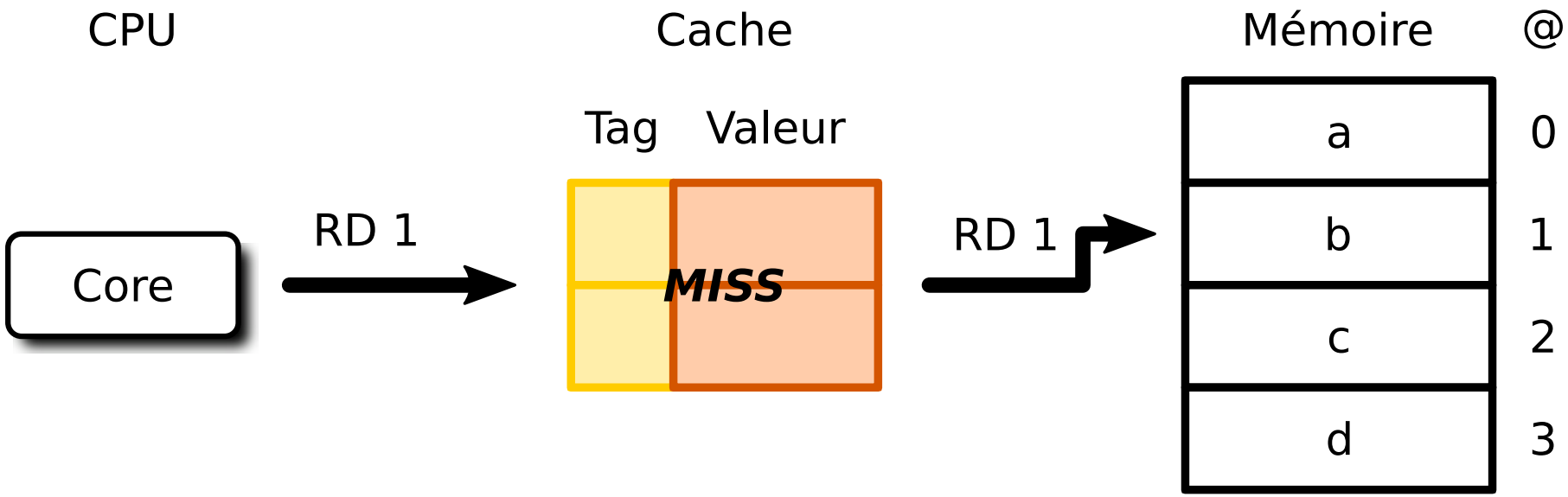
Fonctionnement basique (1)



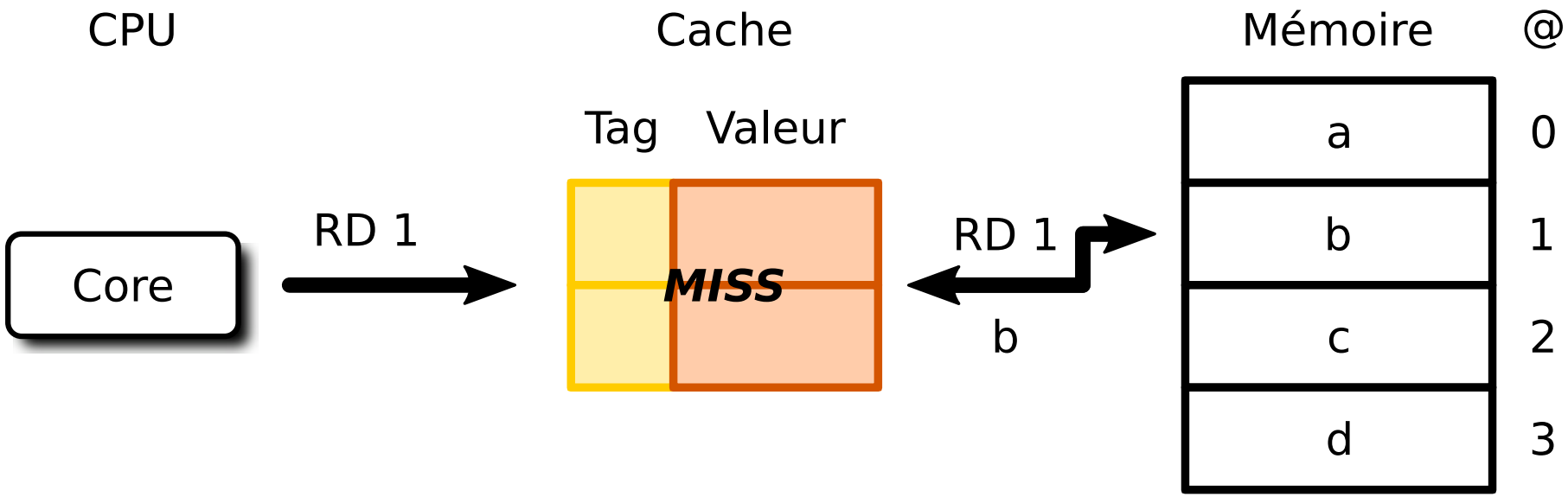
Fonctionnement basique (1)



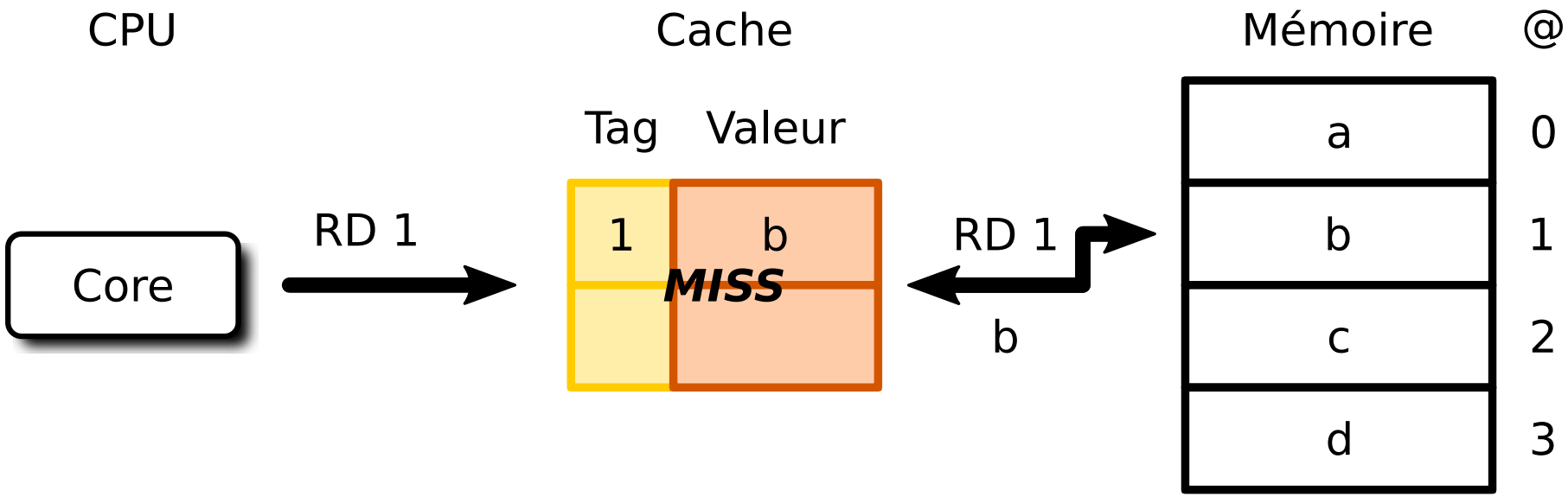
Fonctionnement basique (1)



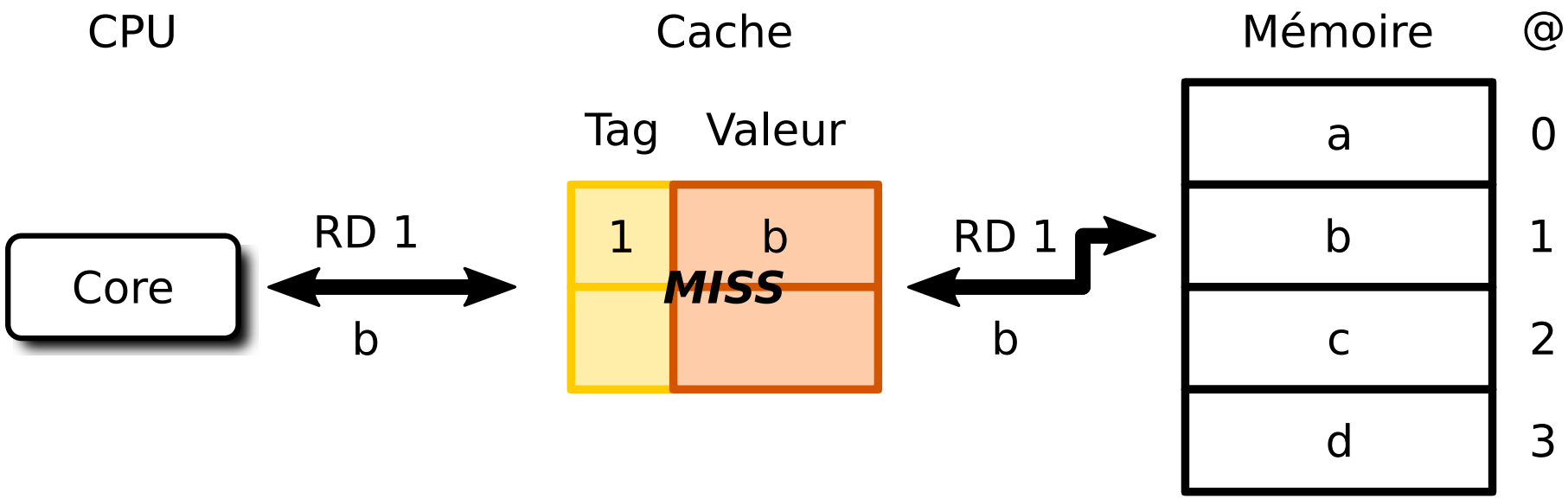
Fonctionnement basique (1)



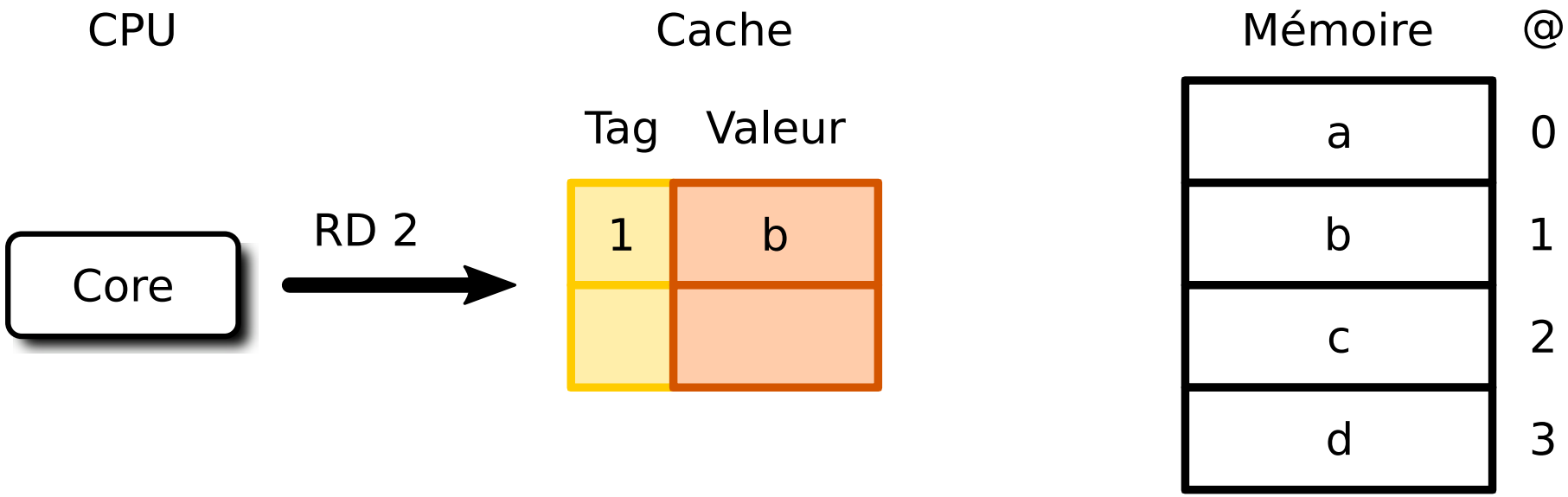
Fonctionnement basique (1)



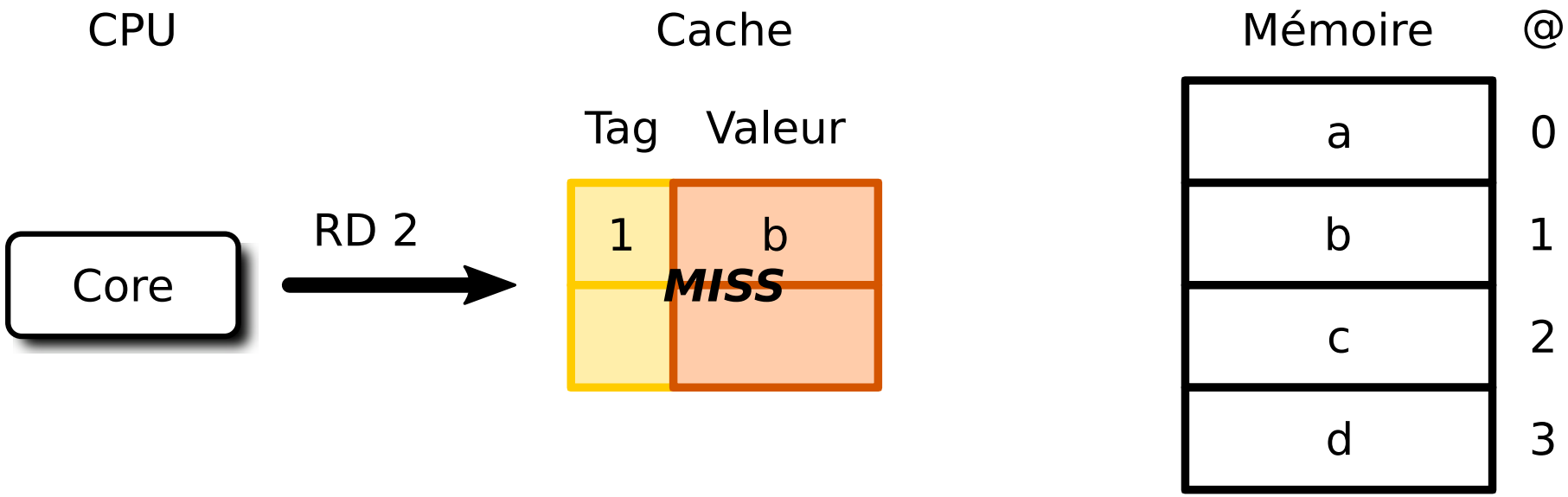
Fonctionnement basique (1)



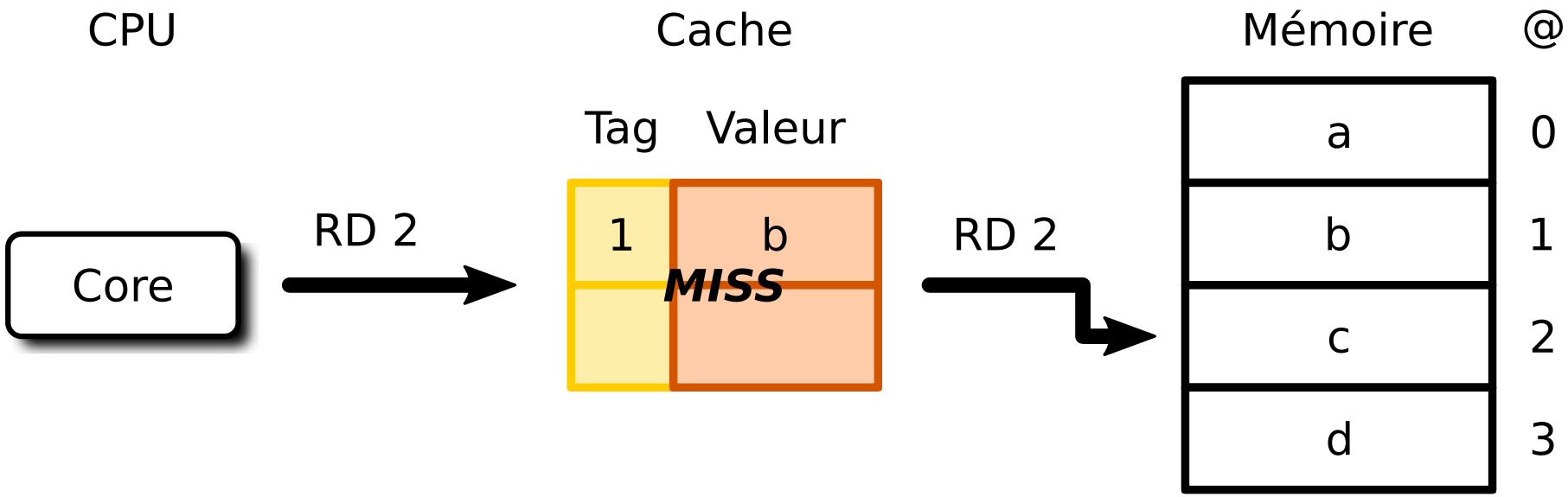
Fonctionnement basique (1)



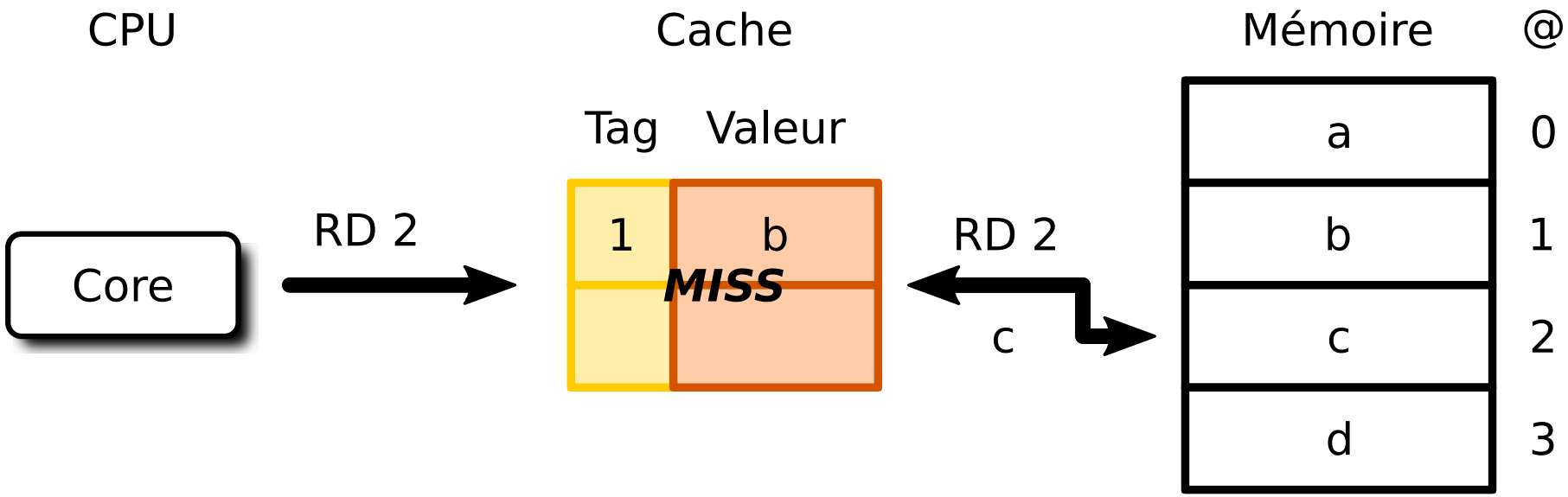
Fonctionnement basique (1)



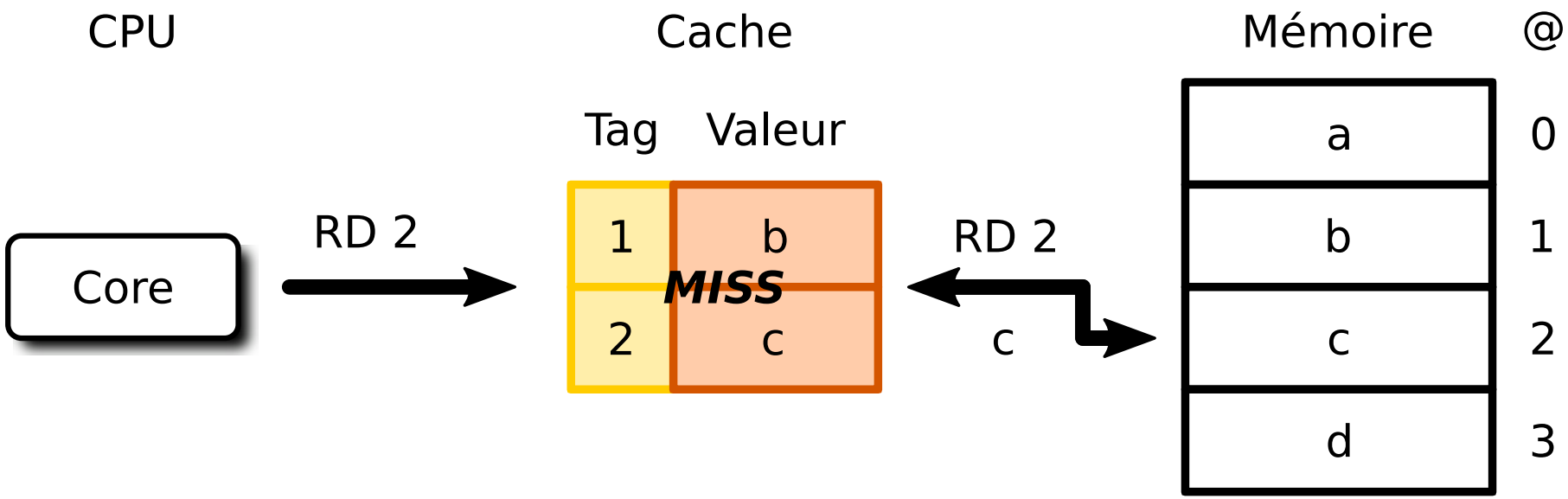
Fonctionnement basique (1)



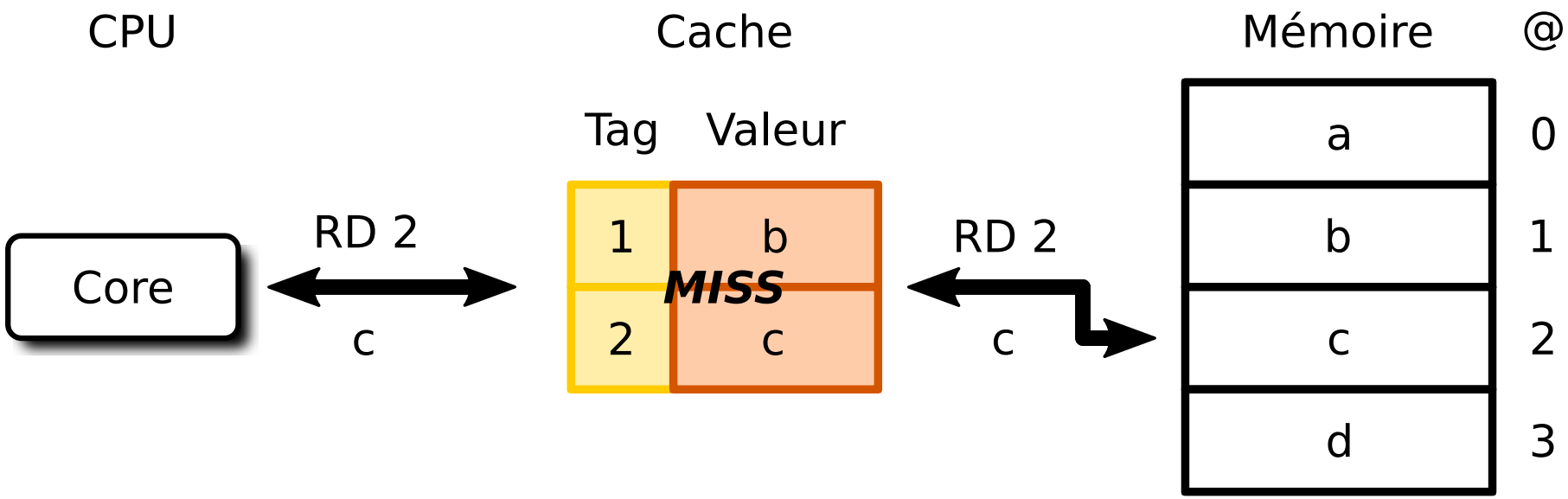
Fonctionnement basique (1)



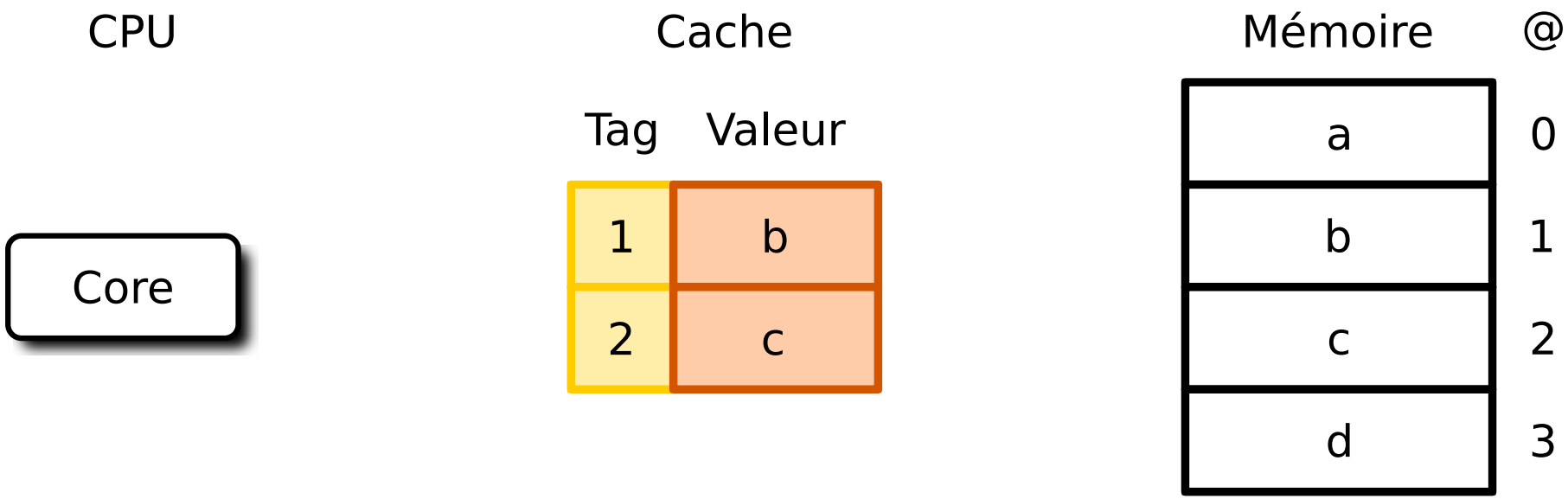
Fonctionnement basique (1)



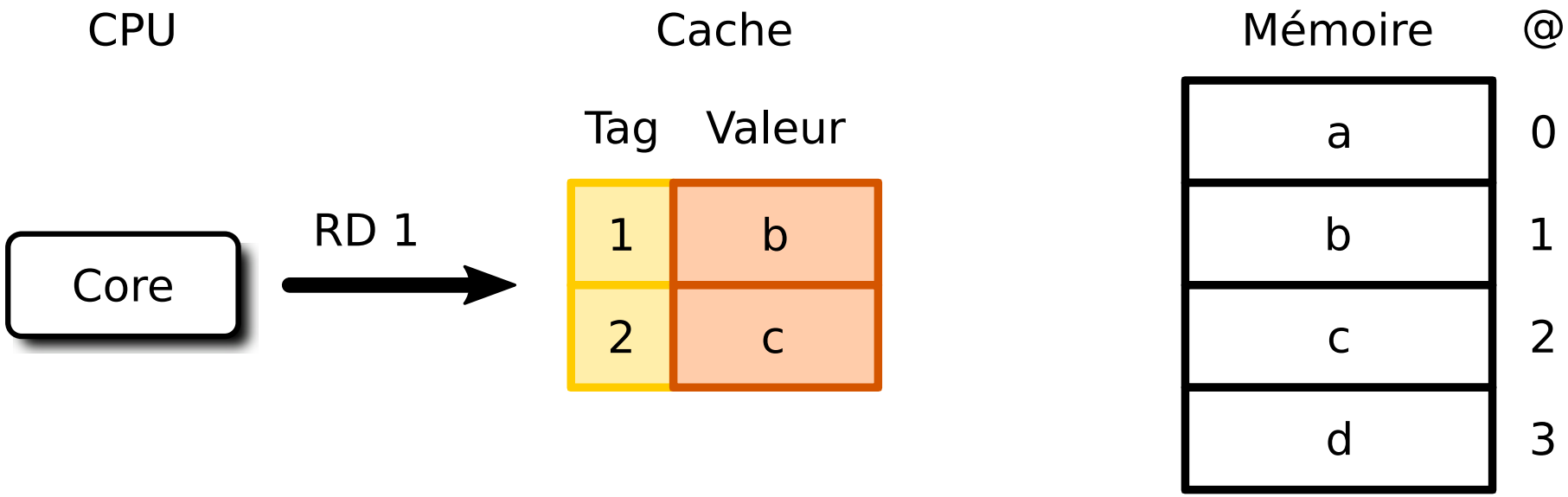
Fonctionnement basique (1)



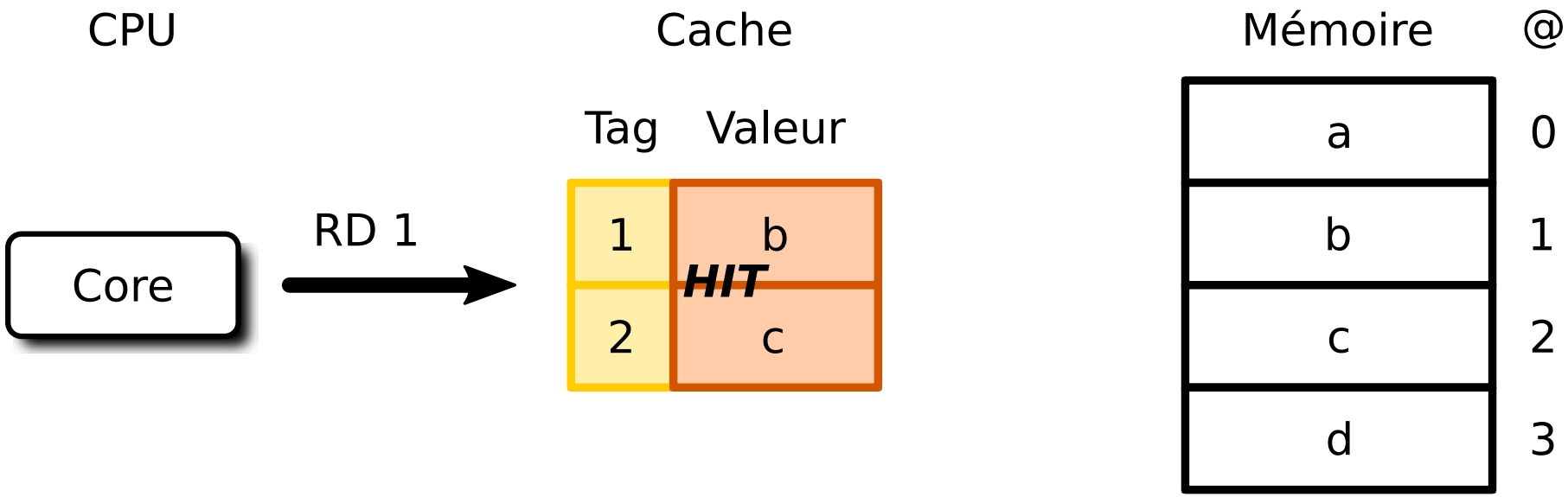
Fonctionnement basique (1)



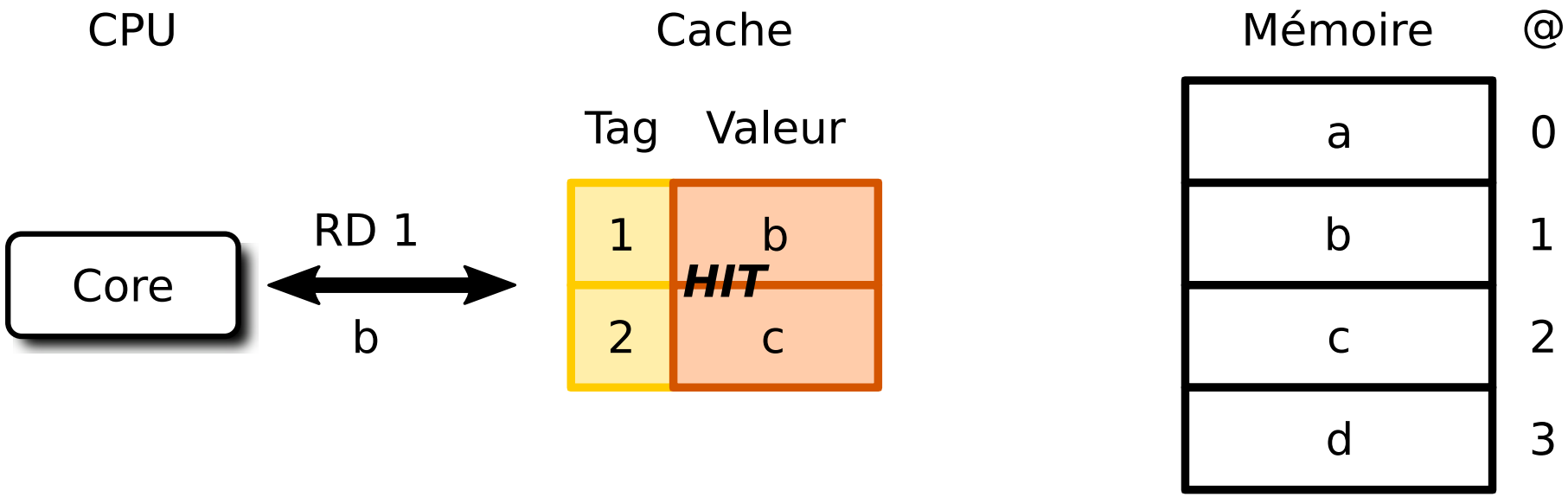
Fonctionnement basique (1)



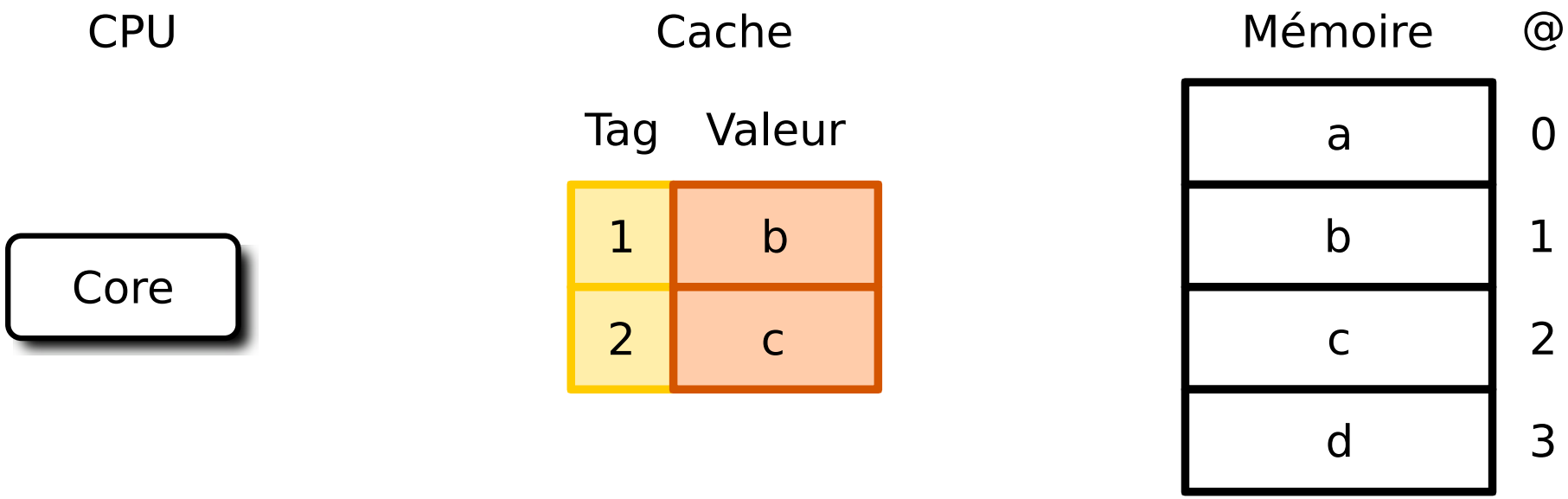
Fonctionnement basique (1)



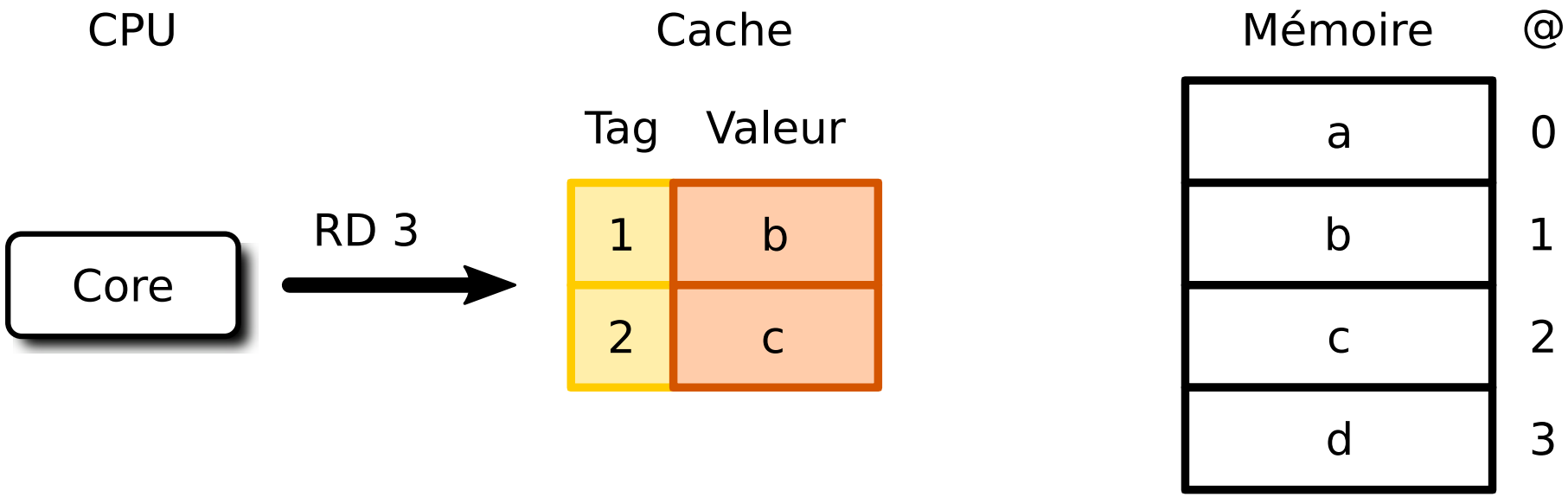
Fonctionnement basique (1)



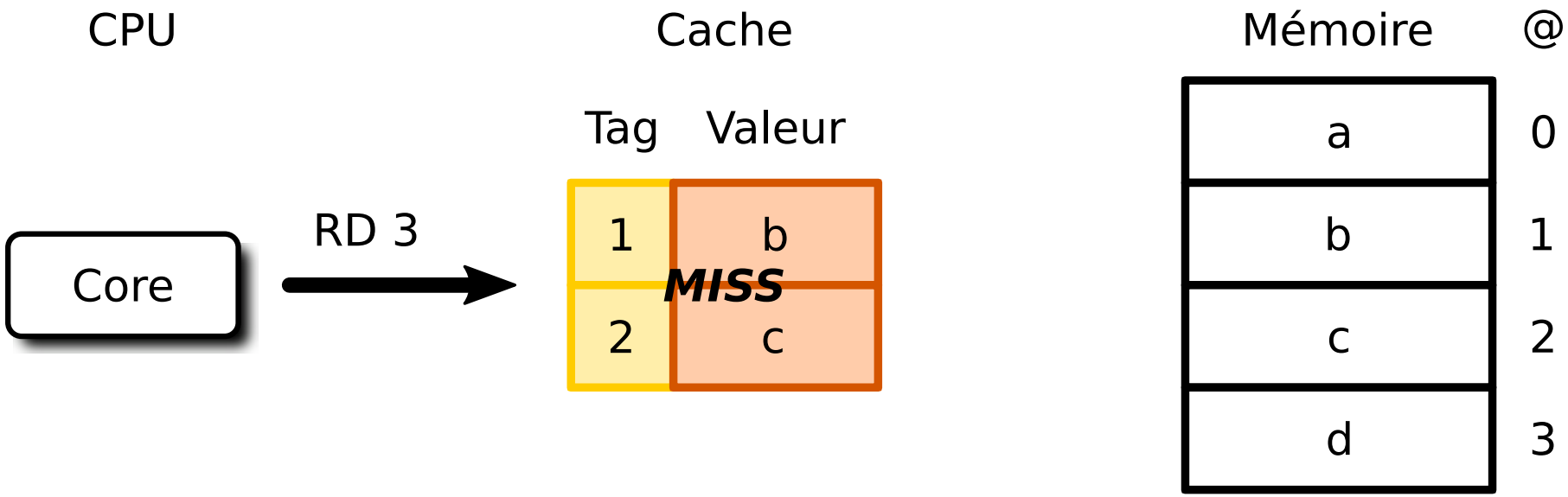
Fonctionnement basique (1)



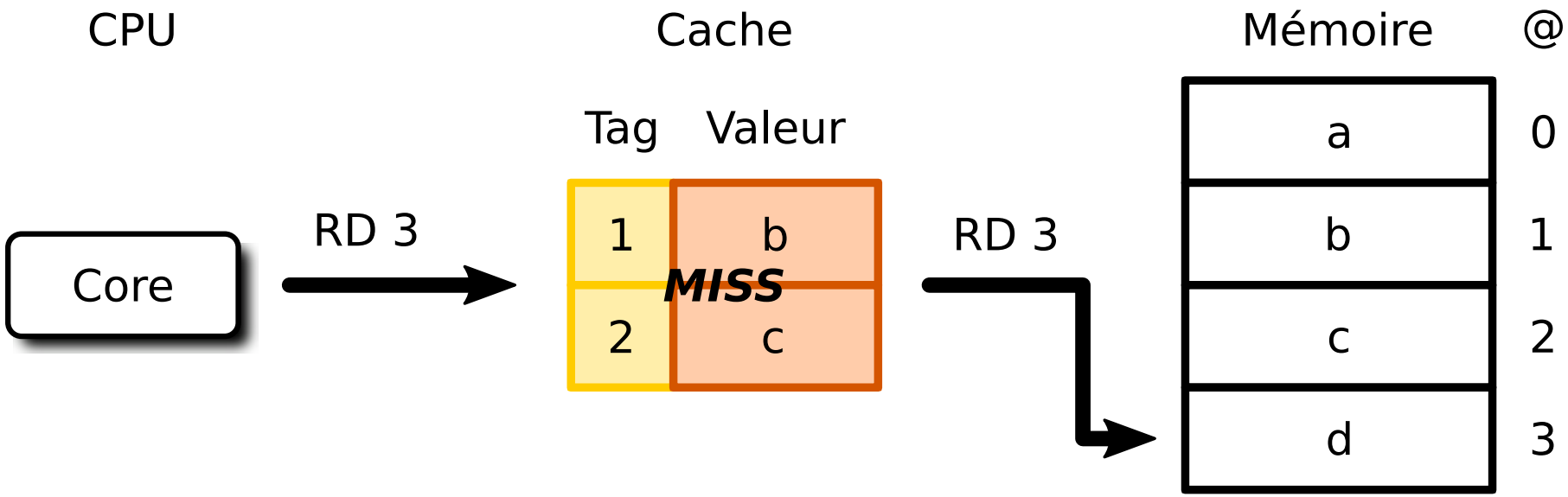
Fonctionnement basique (1)



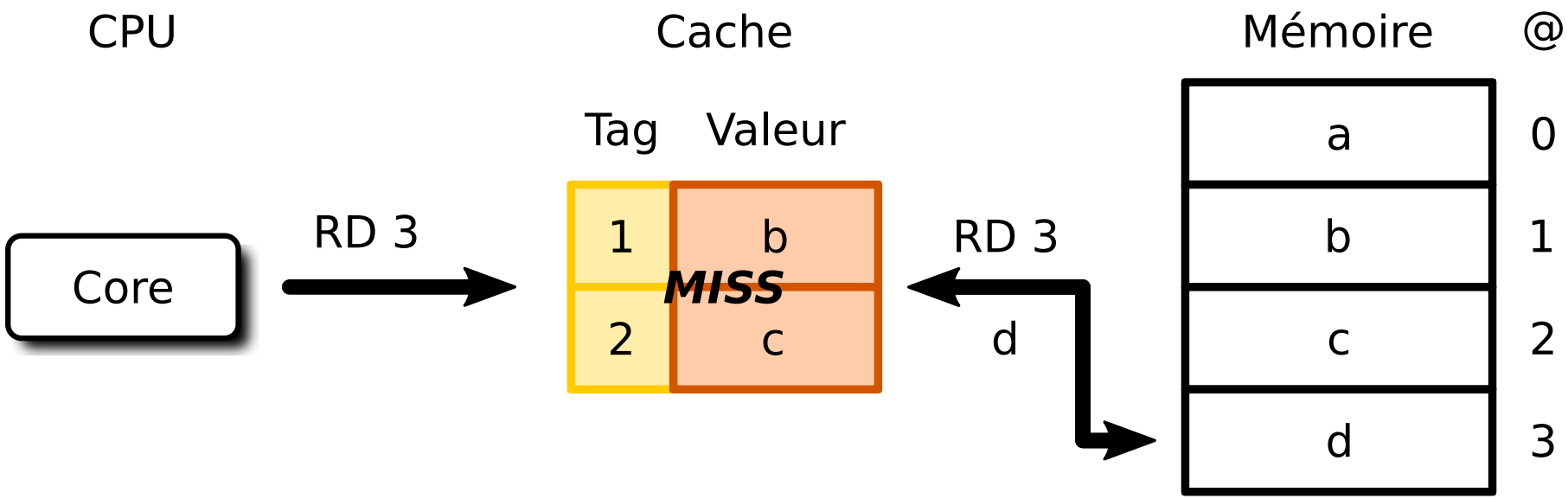
Fonctionnement basique (1)



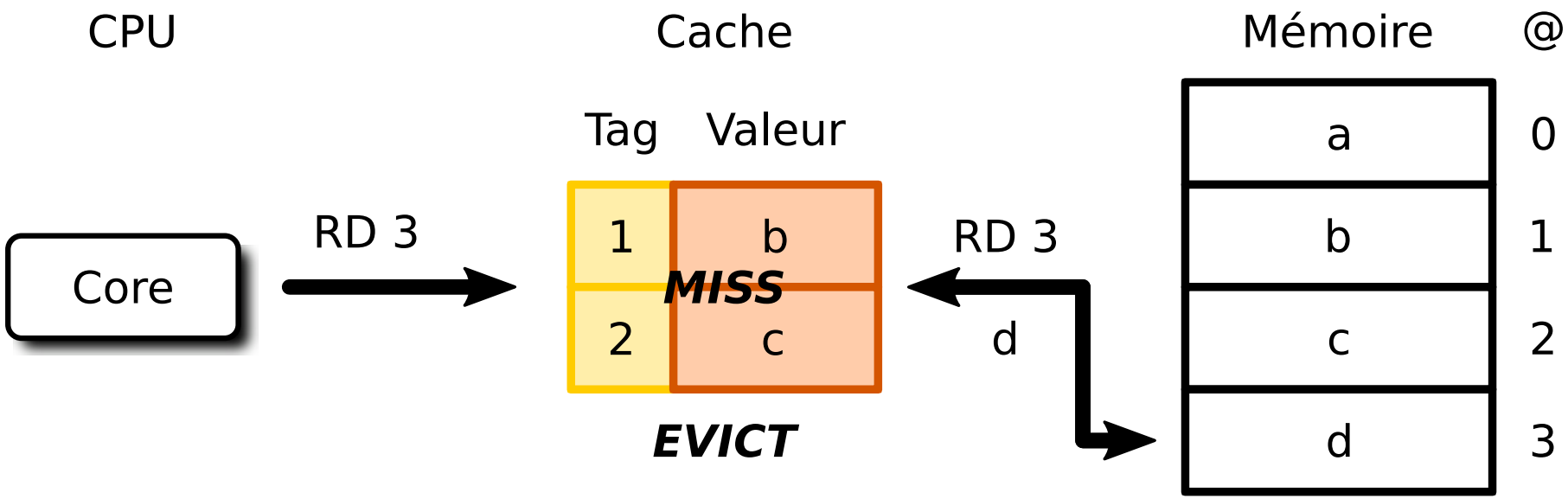
Fonctionnement basique (1)



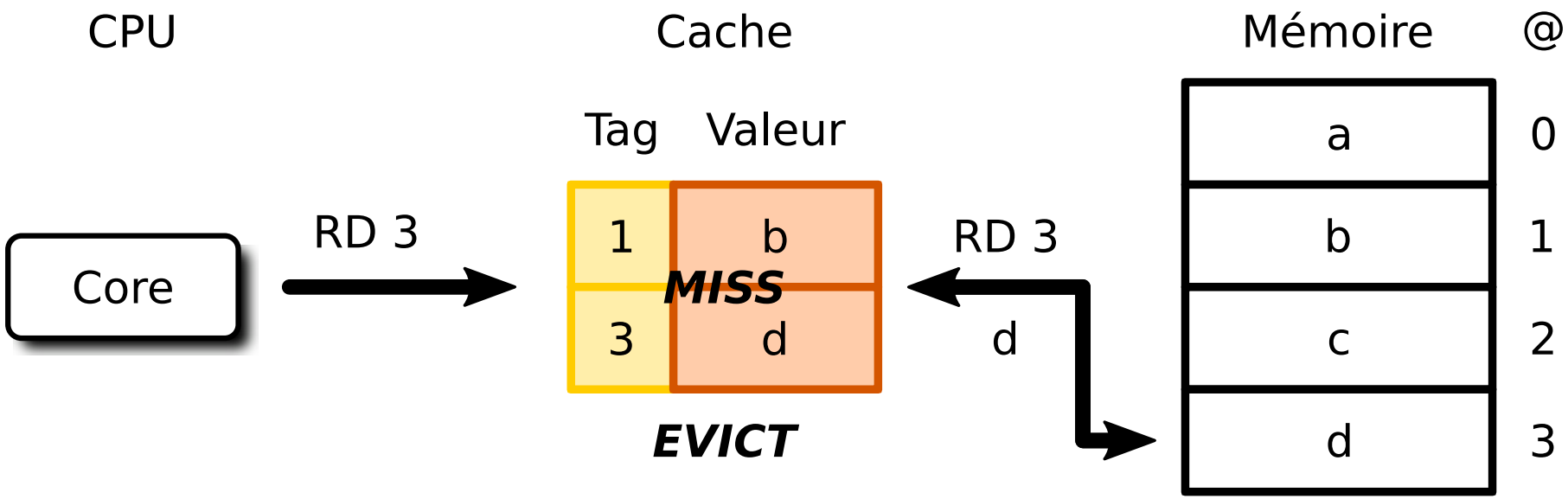
Fonctionnement basique (1)



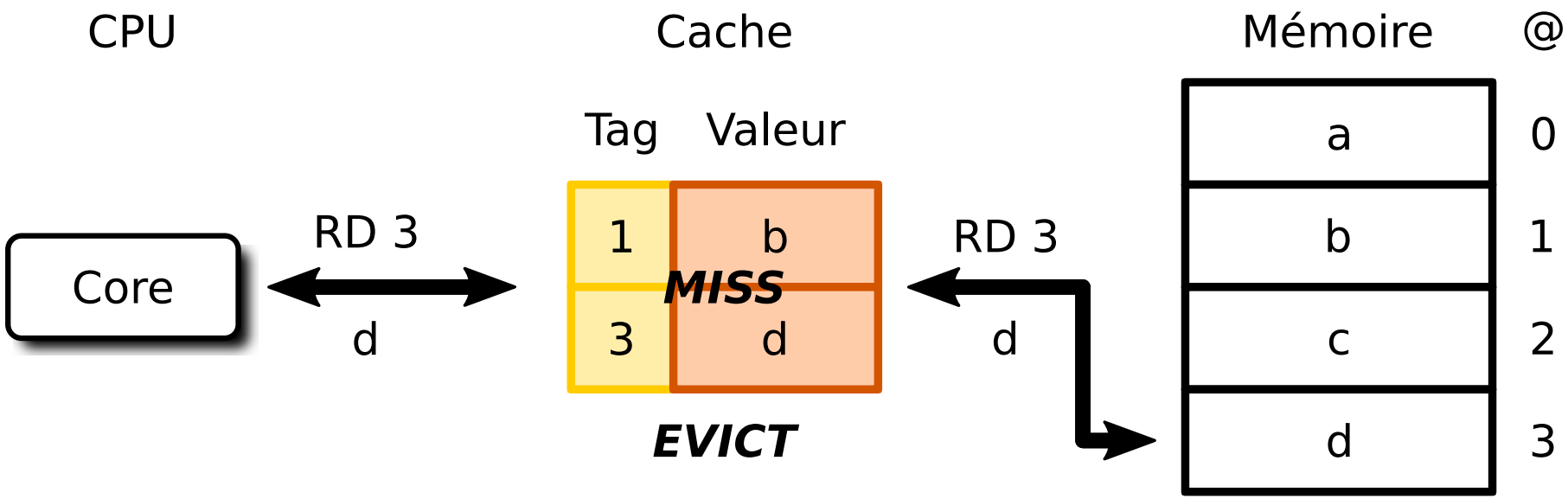
Fonctionnement basique (1)



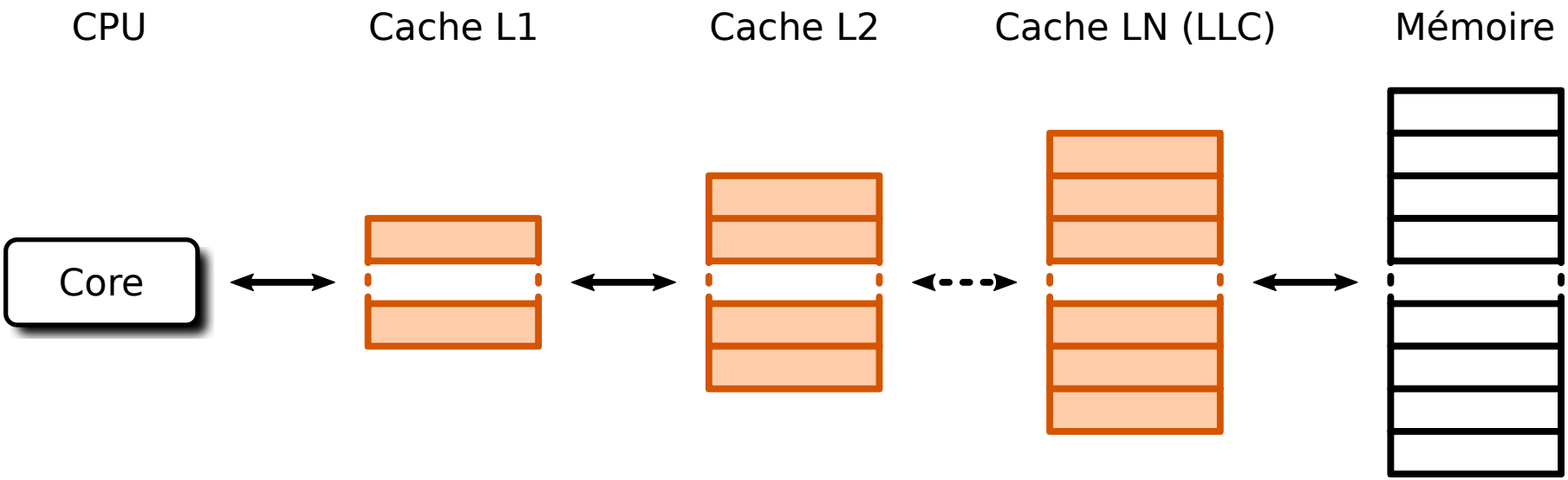
Fonctionnement basique (1)



Fonctionnement basique (1)

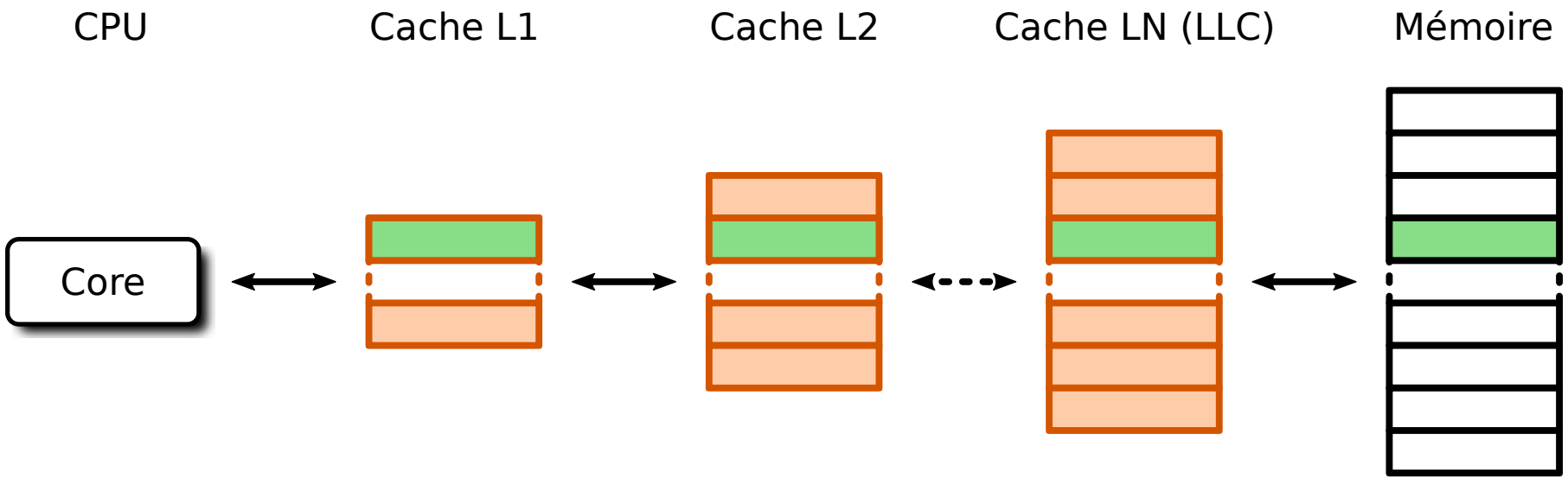


Fonctionnement basique (2)



- Il peut exister une quantité arbitraire de niveaux de cache

Fonctionnement basique (3)

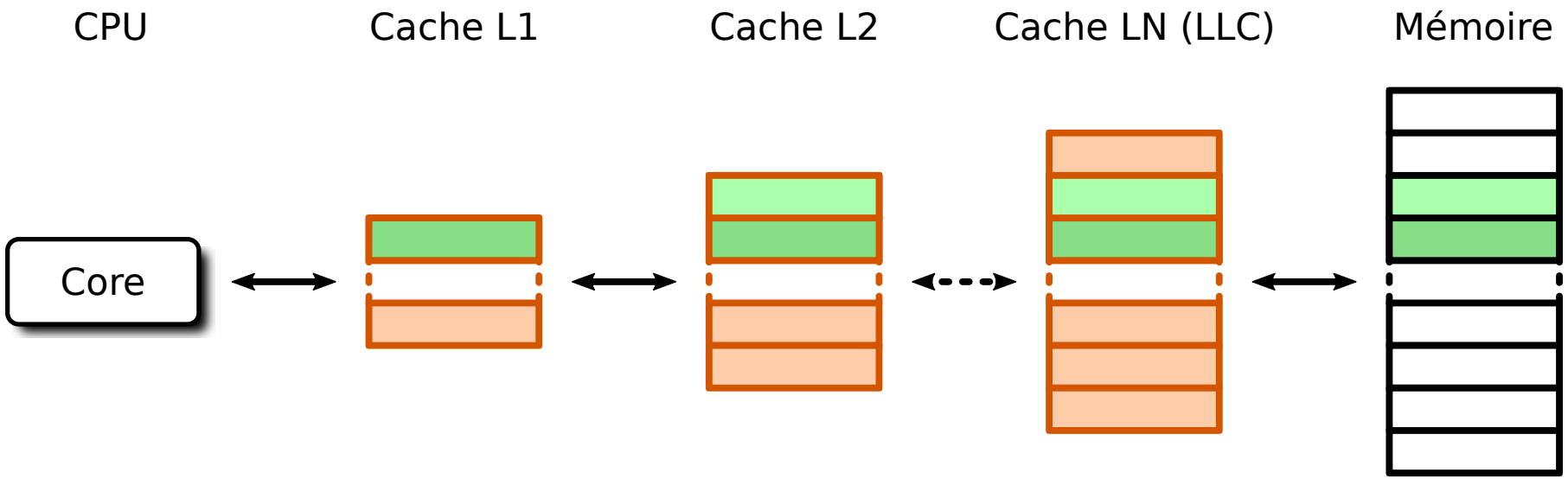


Cache inclusif : L_{k+1} inclusif de L_k

Soit $L_k^\top = \{\text{lignes de caches de niveau k chargées}\}$
L_{k+1} inclusif de $L_k \equiv \forall l_t \in L_k^\top \mid l_t$ taggée $t \Rightarrow l_t \in L_{k+1}^\top \equiv L_k^\top \subseteq L_{k+1}^\top$

Toute ligne de cachée chargée au niveau k est chargée au niveau $k+1$

Fonctionnement basique (3)

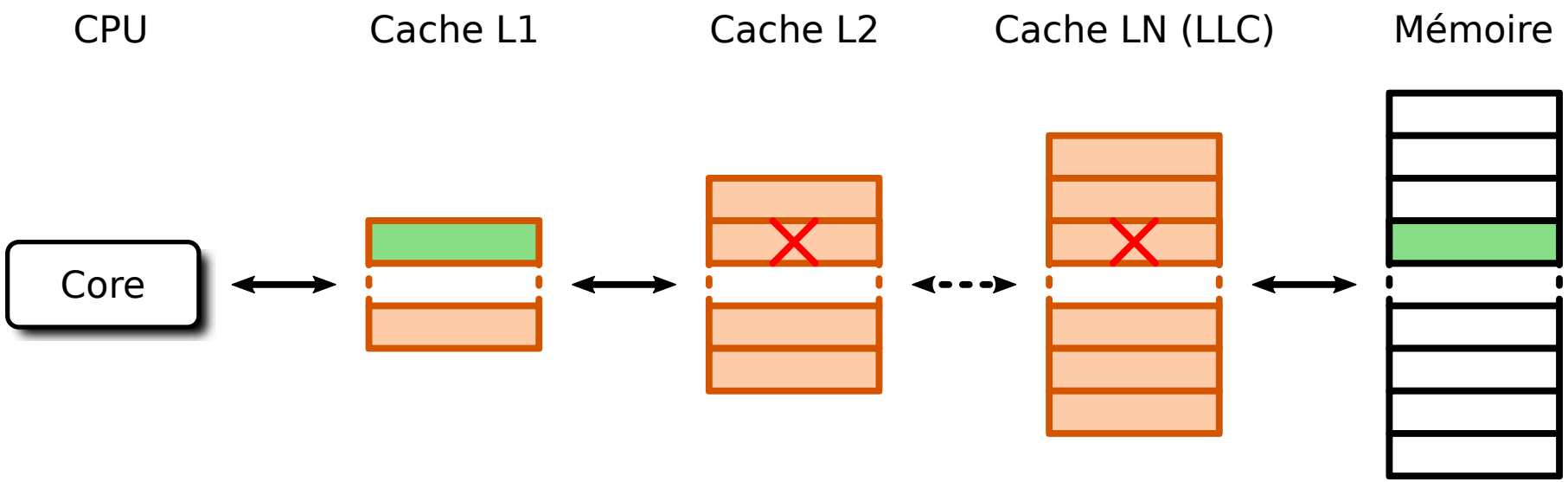


Cache inclusif : L_{k+1} inclusif de L_k

Soit $L_k^\top = \{\text{lignes de caches de niveau k chargées}\}$
L_{k+1} inclusif de $L_k \equiv \forall l_t \in L_k^\top \mid l_t$ taggée $t \Rightarrow l_t \in L_{k+1}^\top \equiv L_k^\top \subseteq L_{k+1}^\top$

Toute ligne de cachée chargée au niveau k est chargée au niveau $k+1$

Fonctionnement basique (3)



Cache inclusif : L_{k+1} inclusif de L_k

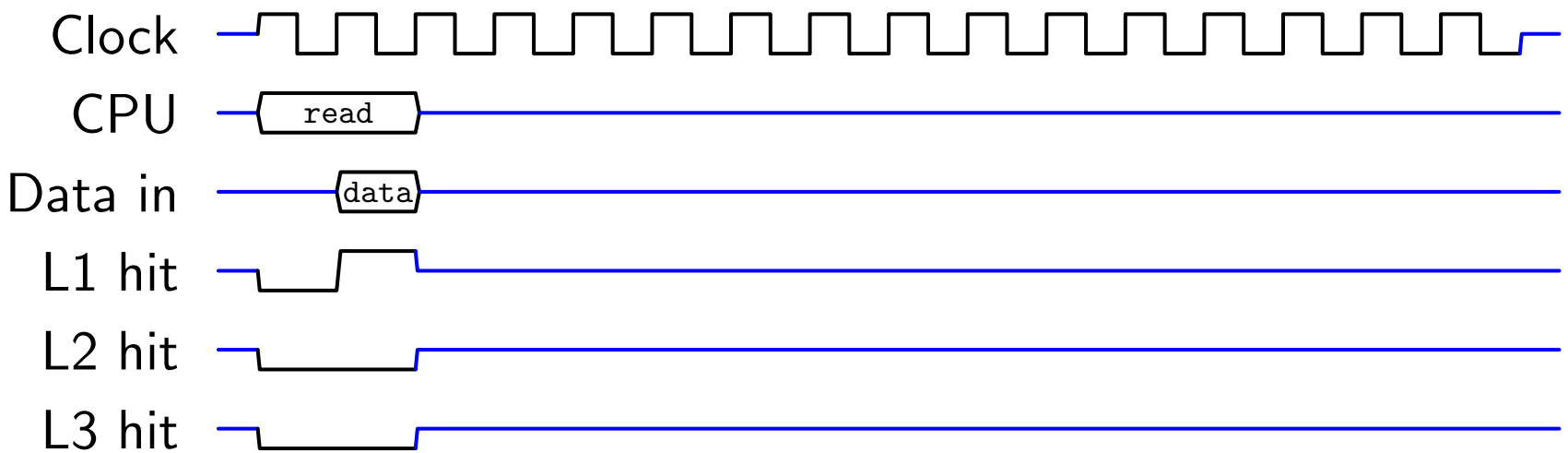
Soit $L_k^\top = \{\text{lignes de caches de niveau k chargées}\}$
L_{k+1} inclusif de $L_k \equiv \forall l_t \in L_k^\top \mid l_t$ taggée $t \Rightarrow l_t \in L_{k+1}^\top \equiv L_k^\top \subseteq L_{k+1}^\top$

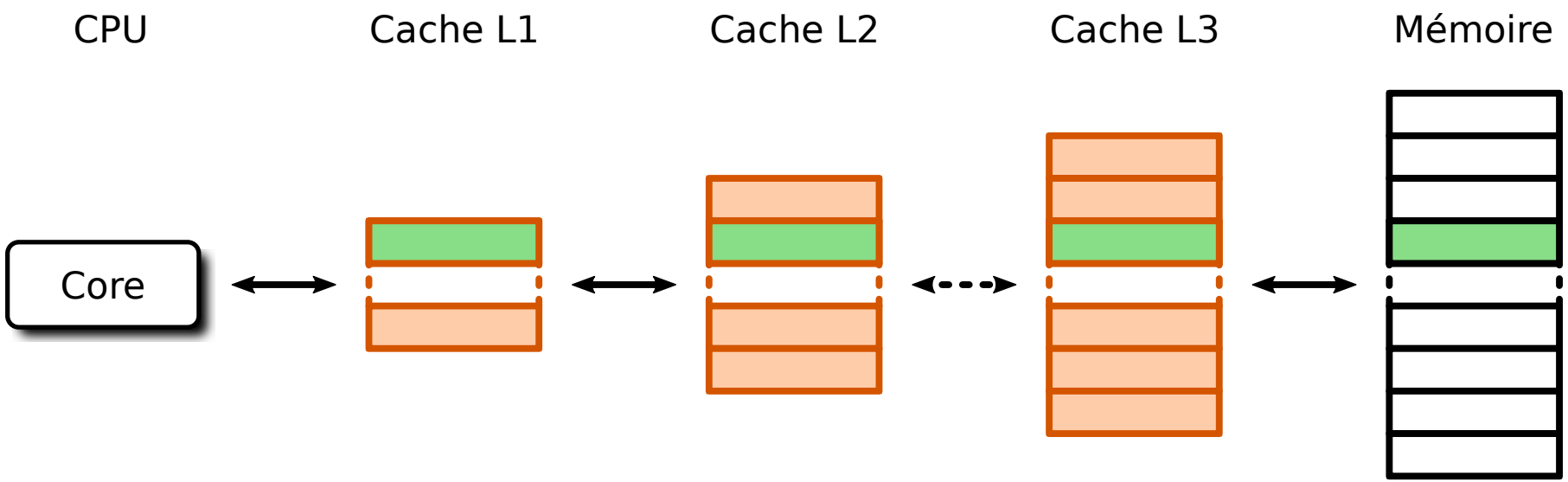
Toute ligne de cachée chargée au niveau k est chargée au niveau $k+1$

Terminologie

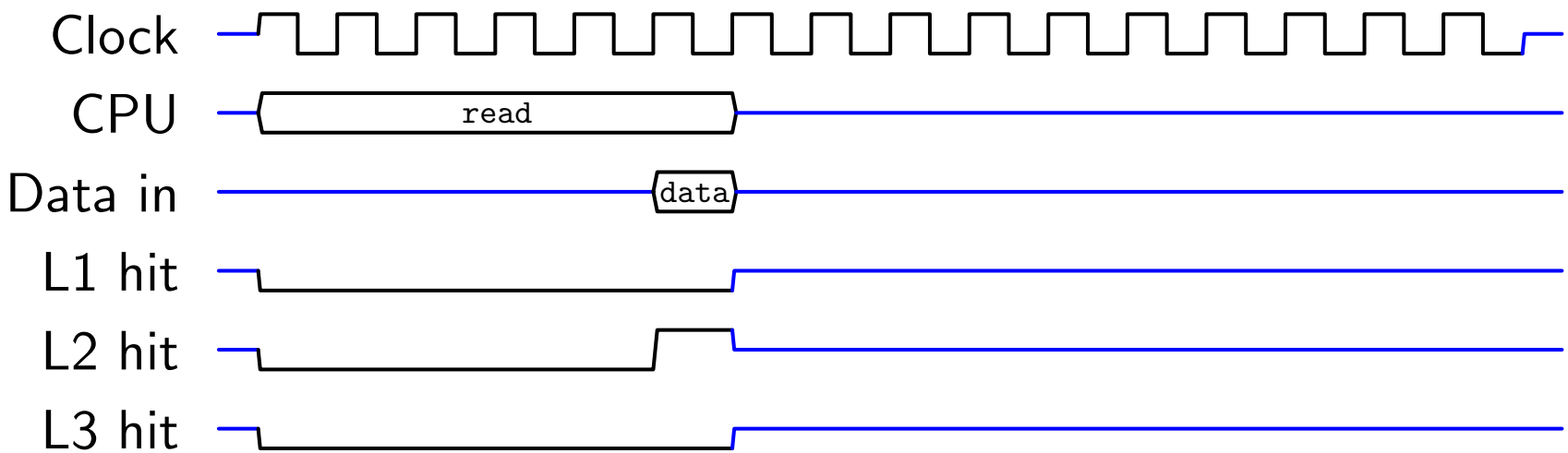
- Ligne en cache $\Rightarrow$ *cache hit*
- Ligne absente du cache $\Rightarrow$ *cache miss*
- Si *cache miss* au niveau $n \Rightarrow$ read$(n+1)$
- Inclusivité : n est inclusif s'il contient toutes les lignes de cache des niveaux supérieurs.
  - Si *cache miss* au niveau n alors *cache miss* au niveau inférieurs
  - Si *cache hit* au niveau $n-1$, alors le mot est présent au niveau n

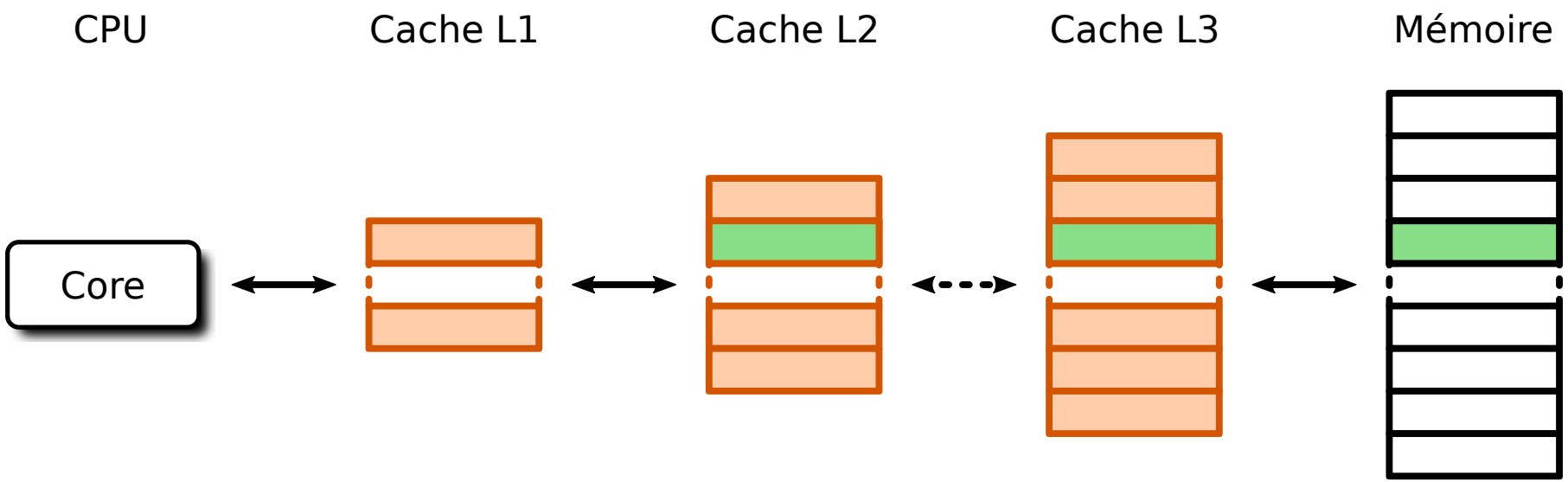
Cache hit et impact sur le temps



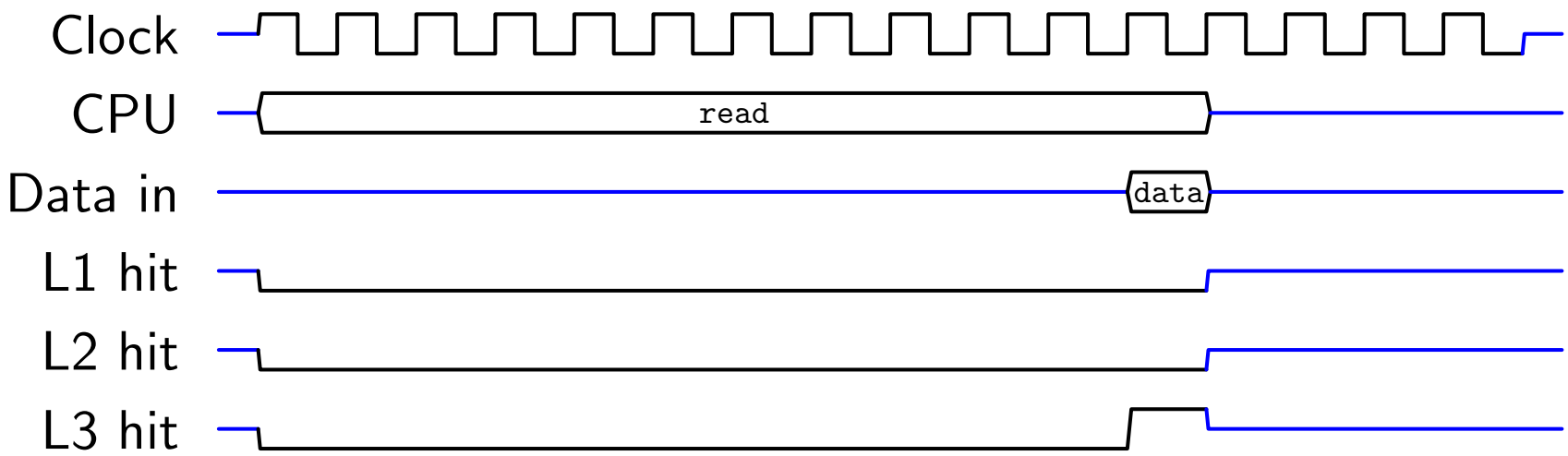


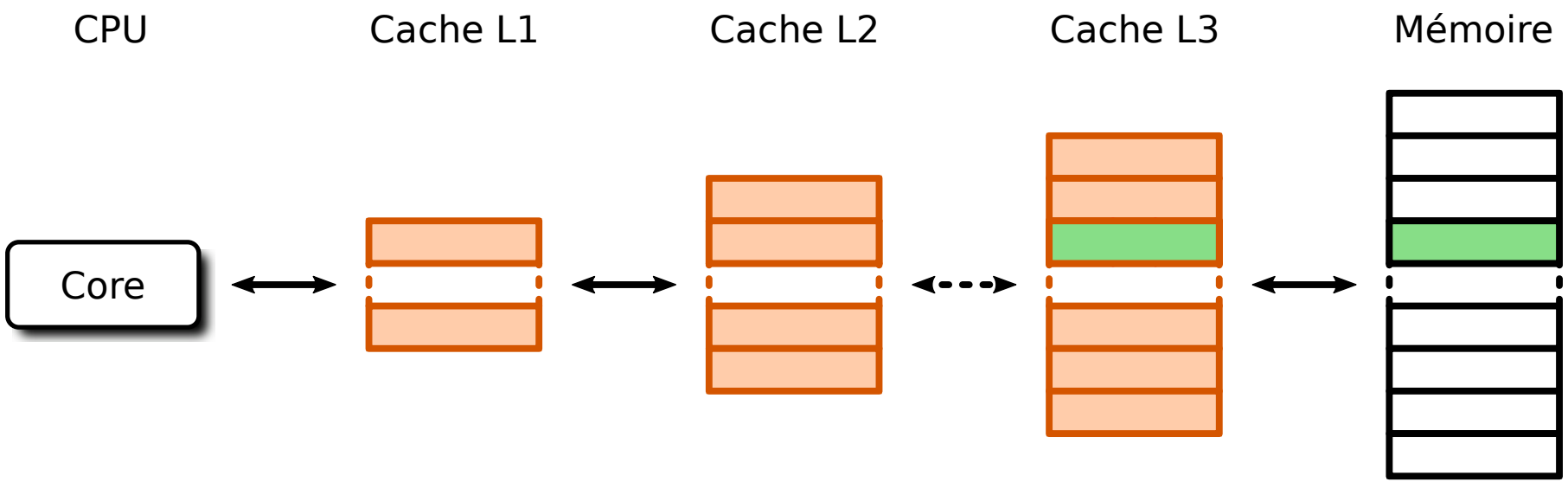
Cache hit et impact sur le temps



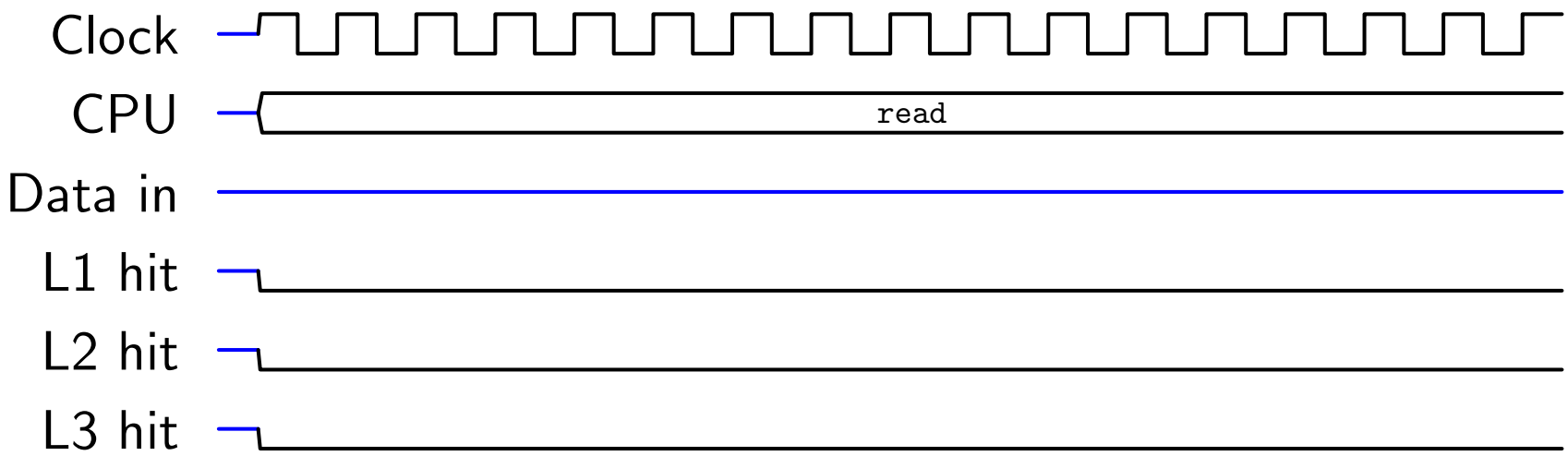


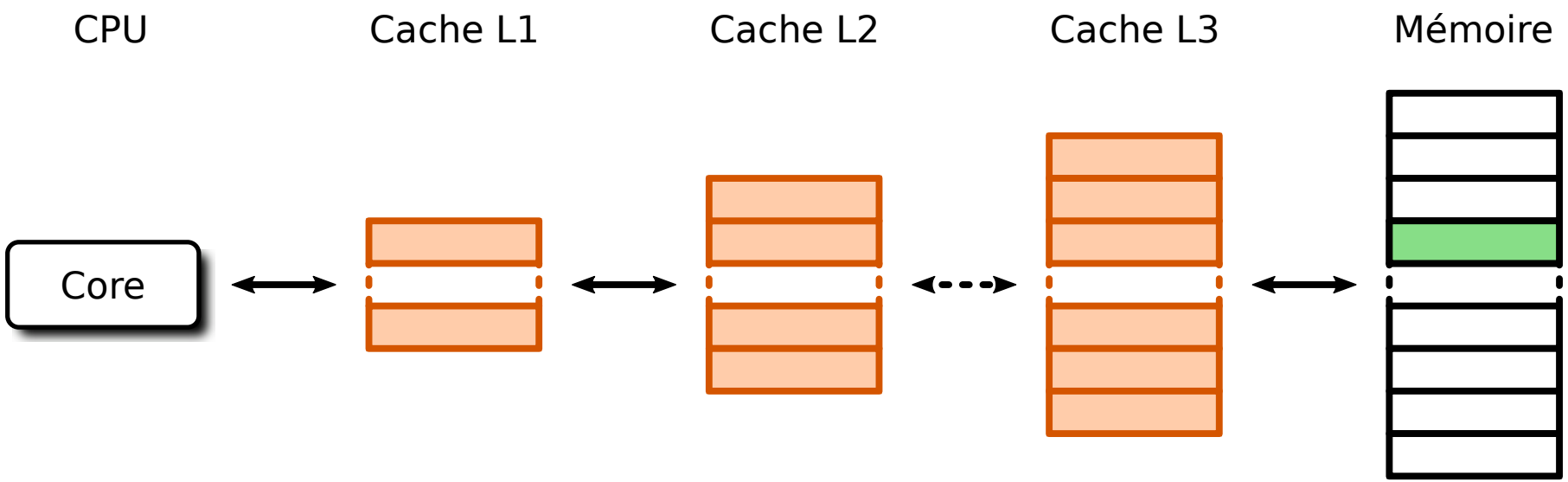
Cache hit et impact sur le temps





Cache hit et impact sur le temps



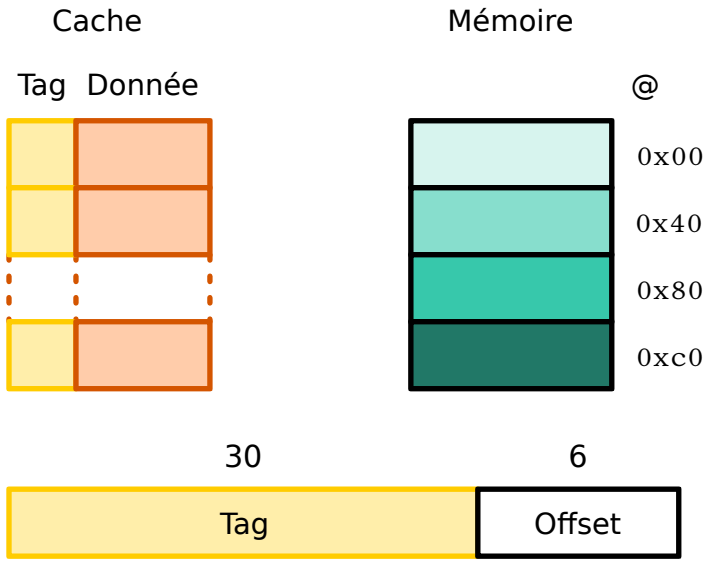


Correspondance mémoire cache

Type de mémoires cache

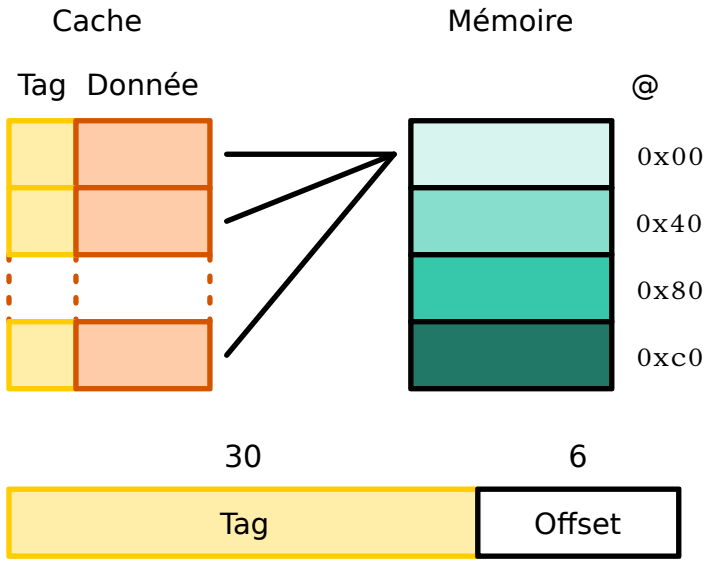
- mémoires caches complètement associatives (*fully associative cache*)
- mémoires caches directes (*direct mapped cache*)
- mémoires caches n-associatives (*n-way set associative cache*)

Correspondance mémoire cache complètement associative



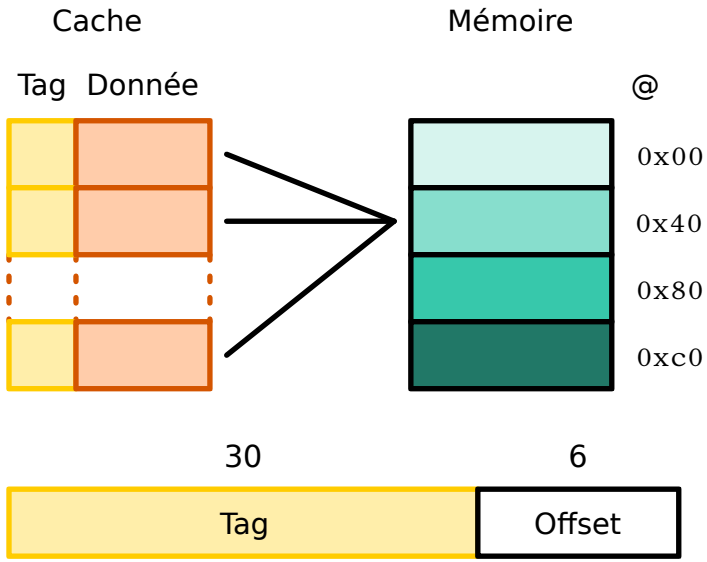
- Maximisation de l'utilisation de la mémoire
- Logique nécessaire de taille maximale

Correspondance mémoire cache complètement associative



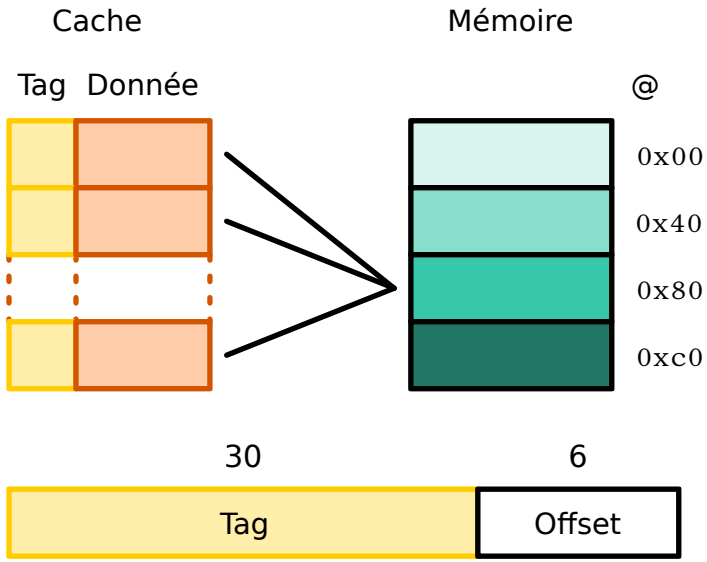
- Maximisation de l'utilisation de la mémoire
- Logique nécessaire de taille maximale

Correspondance mémoire cache complètement associative



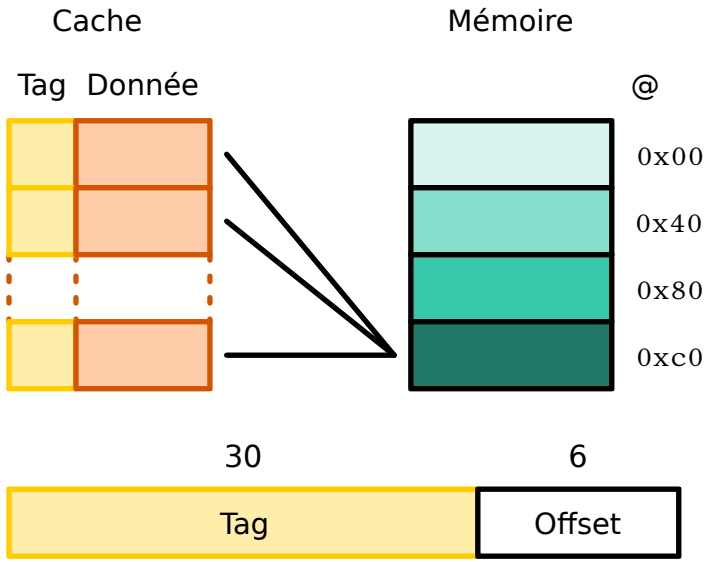
- Maximisation de l'utilisation de la mémoire
- Logique nécessaire de taille maximale

Correspondance mémoire cache complètement associative



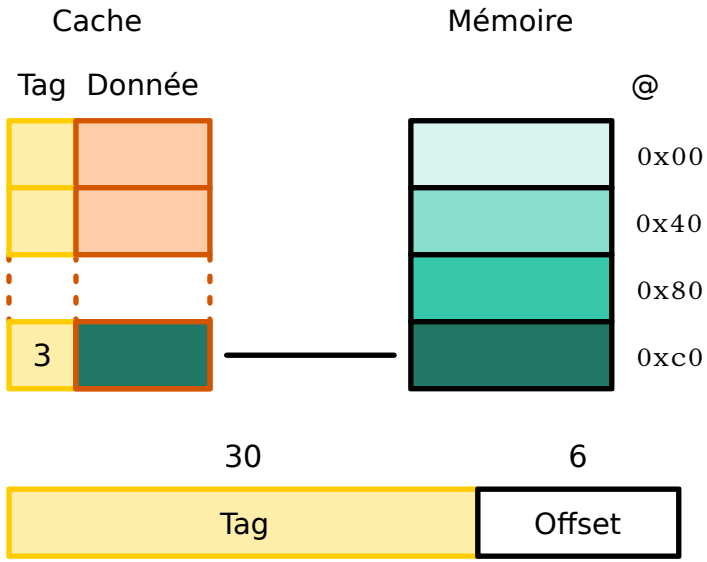
- Maximisation de l'utilisation de la mémoire
- Logique nécessaire de taille maximale

Correspondance mémoire cache complètement associative



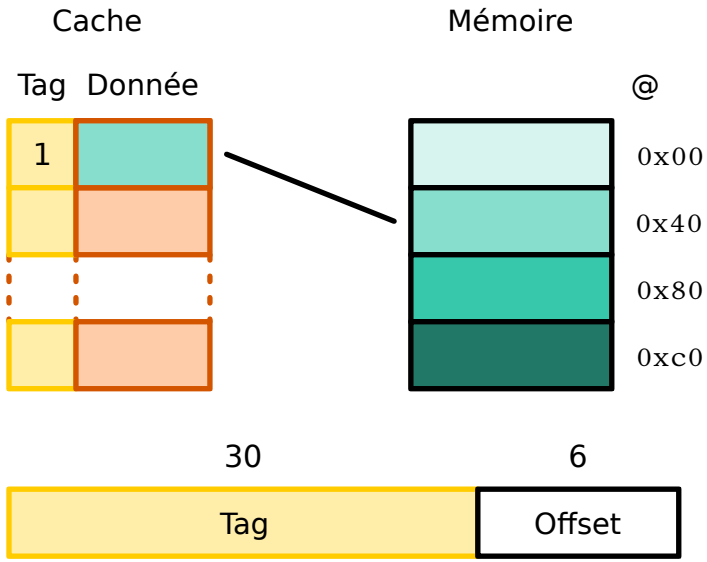
- Maximisation de l'utilisation de la mémoire
- Logique nécessaire de taille maximale

Correspondance mémoire cache complètement associative



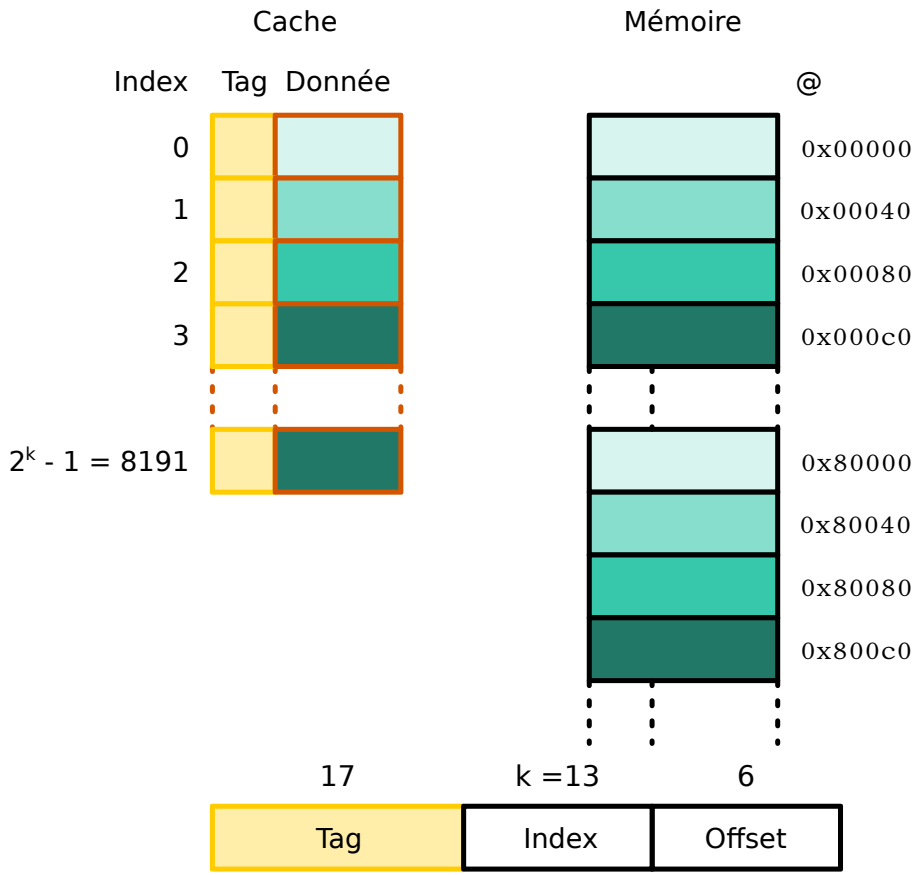
- Maximisation de l'utilisation de la mémoire
- Logique nécessaire de taille maximale

Correspondance mémoire cache complètement associative



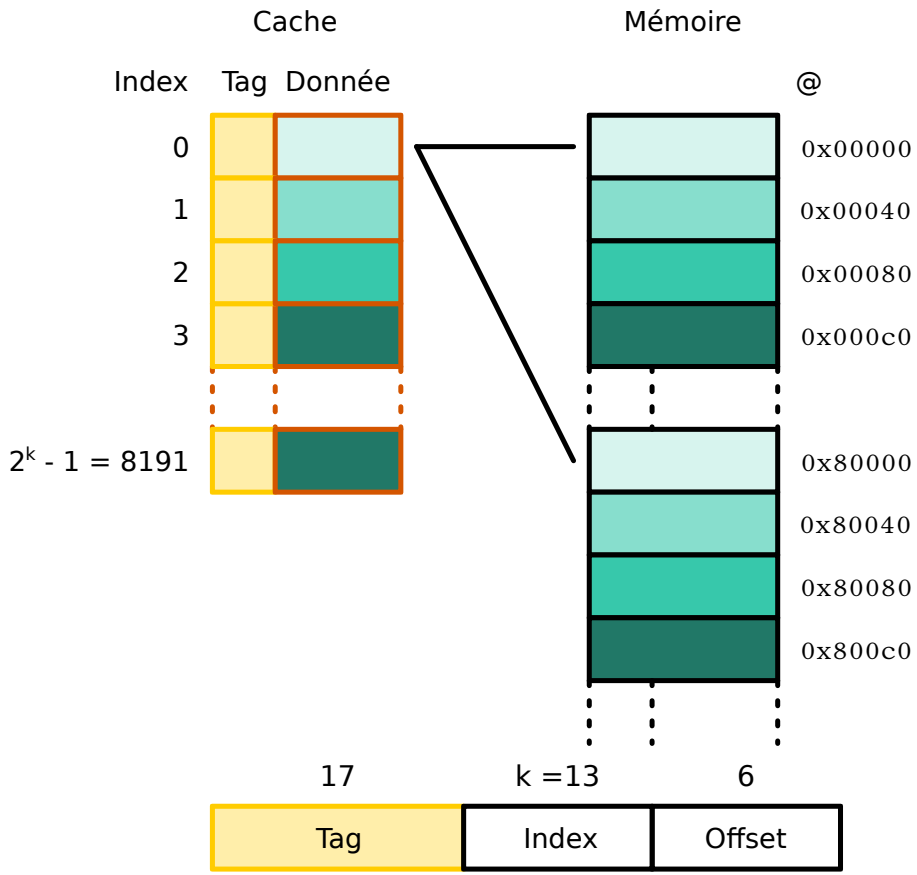
- Maximisation de l'utilisation de la mémoire
- Logique nécessaire de taille maximale

Correspondance mémoire cache directe



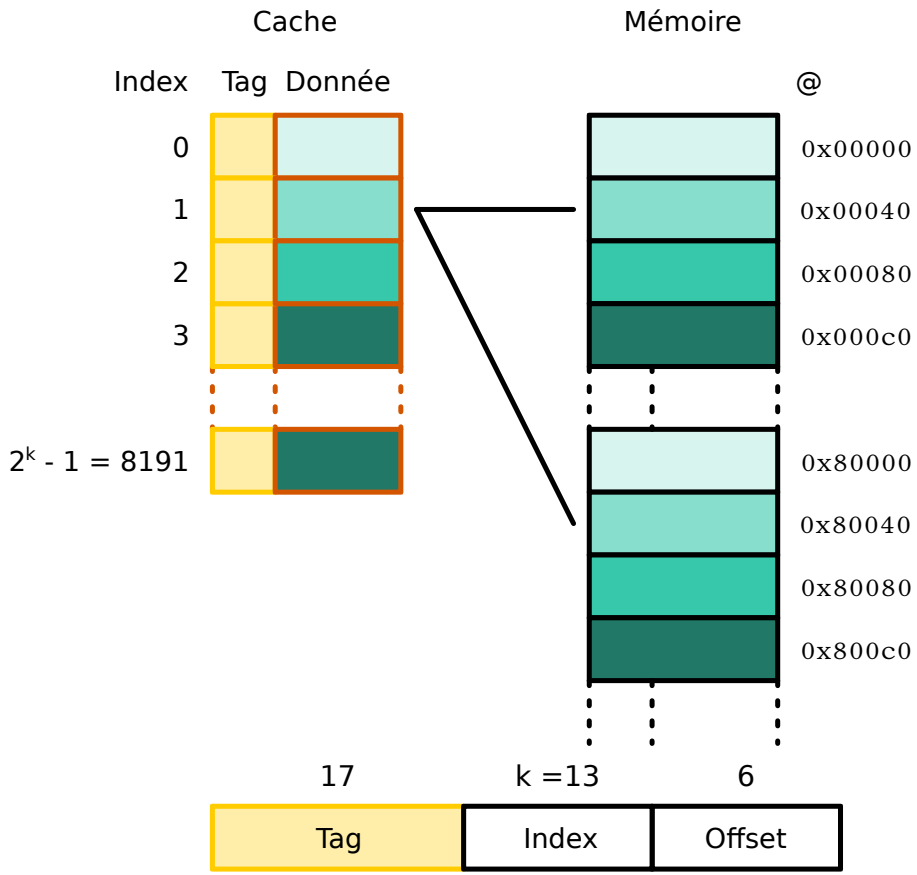
- Minimise de l'utilisation de la mémoire
- Logique nécessaire de taille minimale

Correspondance mémoire cache directe



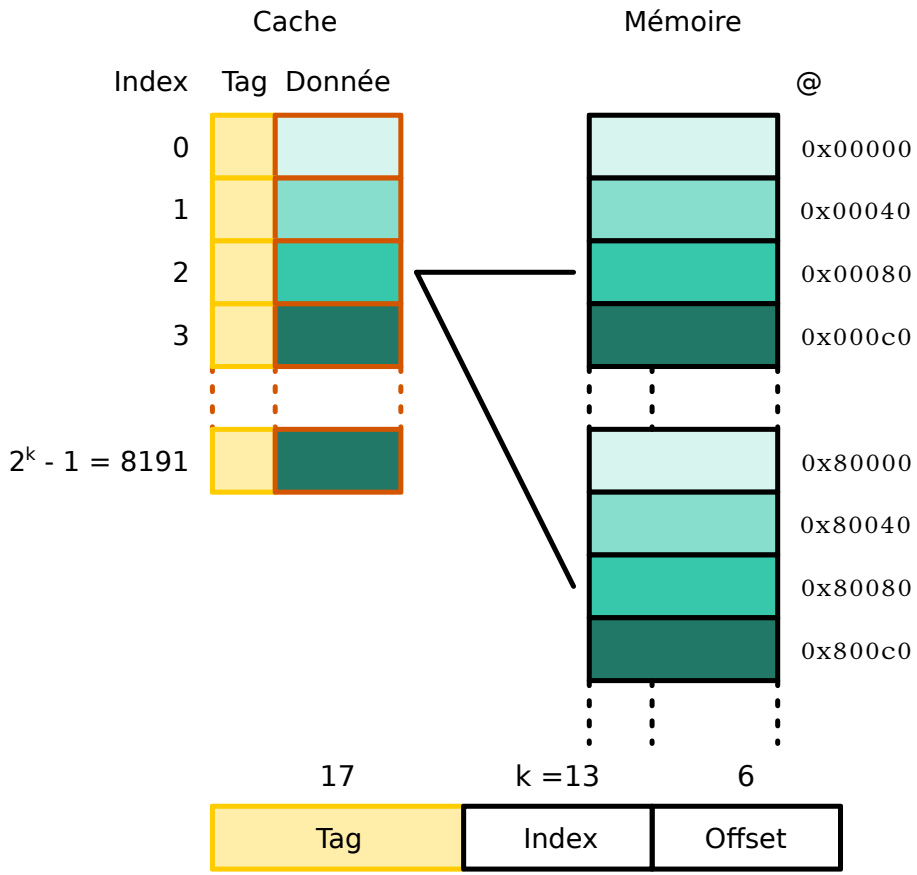
- Minimise de l'utilisation de la mémoire
- Logique nécessaire de taille minimale

Correspondance mémoire cache directe



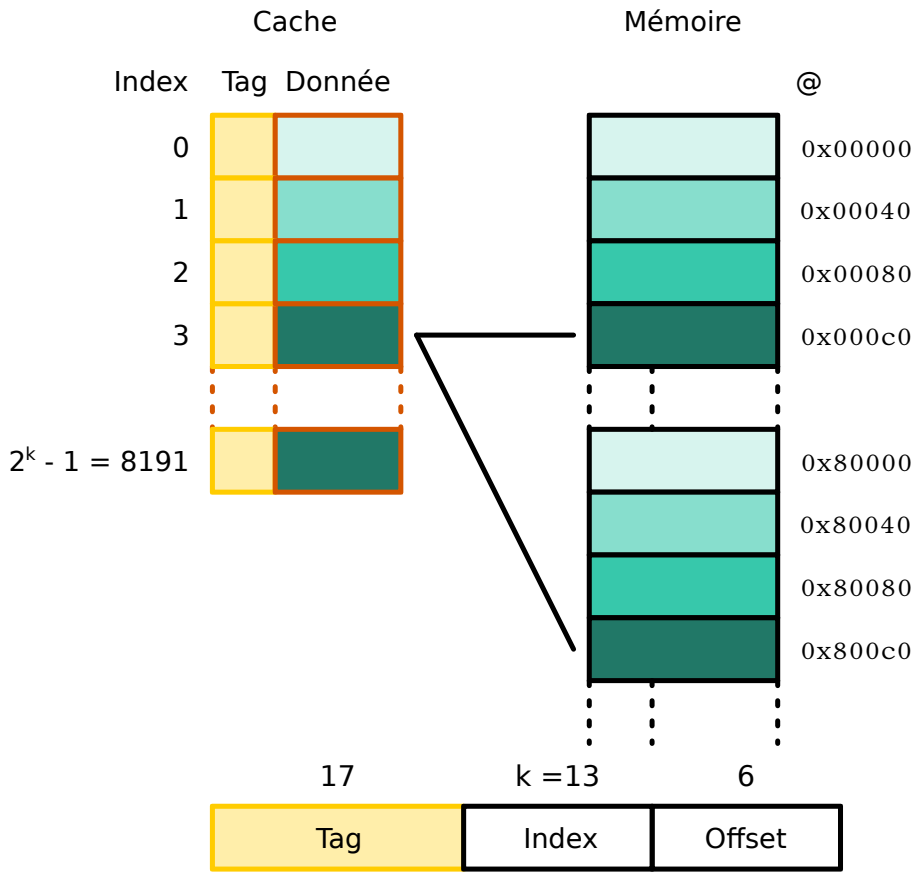
- Minimise de l'utilisation de la mémoire
- Logique nécessaire de taille minimale

Correspondance mémoire cache directe



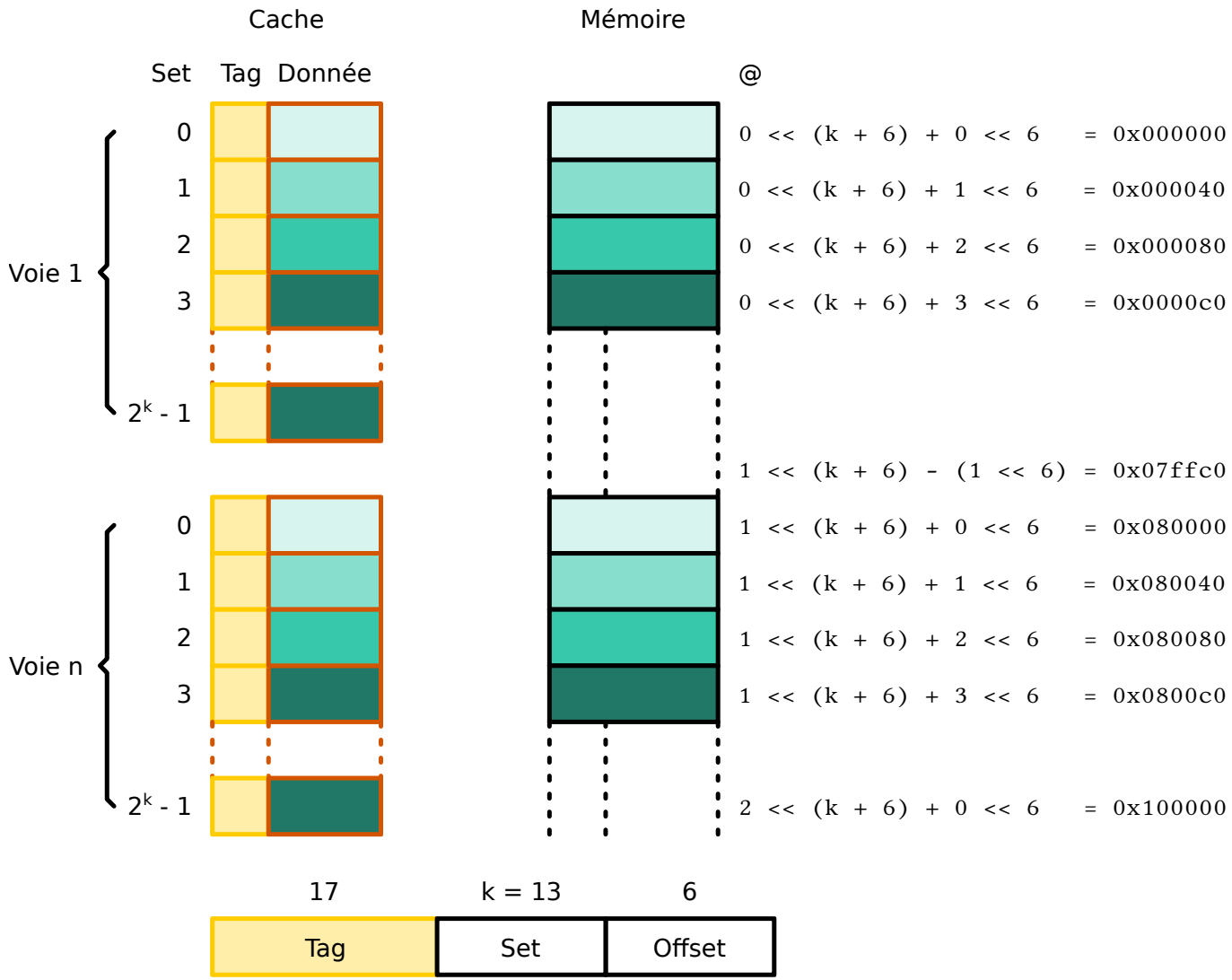
- Minimise de l'utilisation de la mémoire
- Logique nécessaire de taille minimale

Correspondance mémoire cache directe

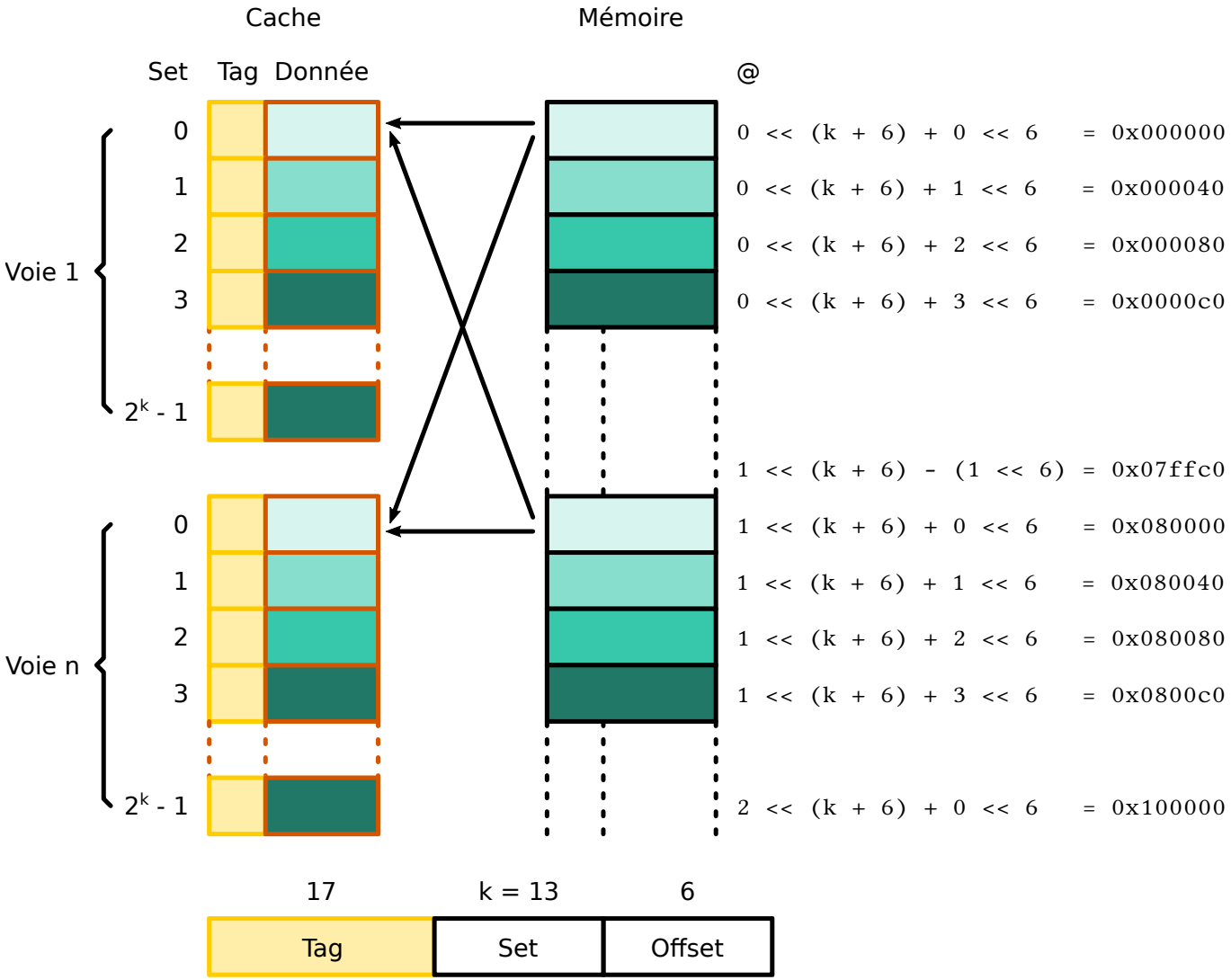


- Minimise de l'utilisation de la mémoire
- Logique nécessaire de taille minimale

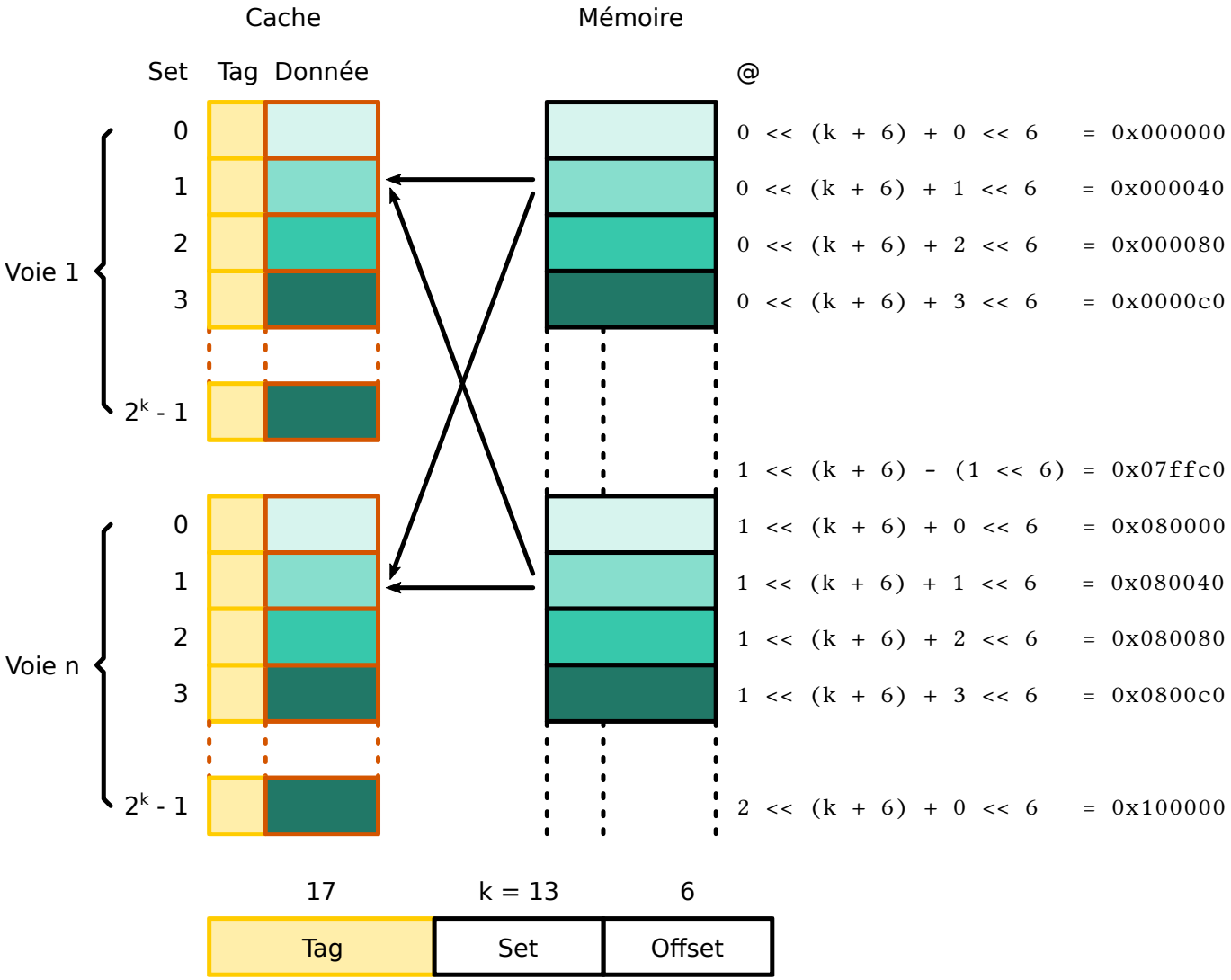
Correspondance mémoire cache n-associative



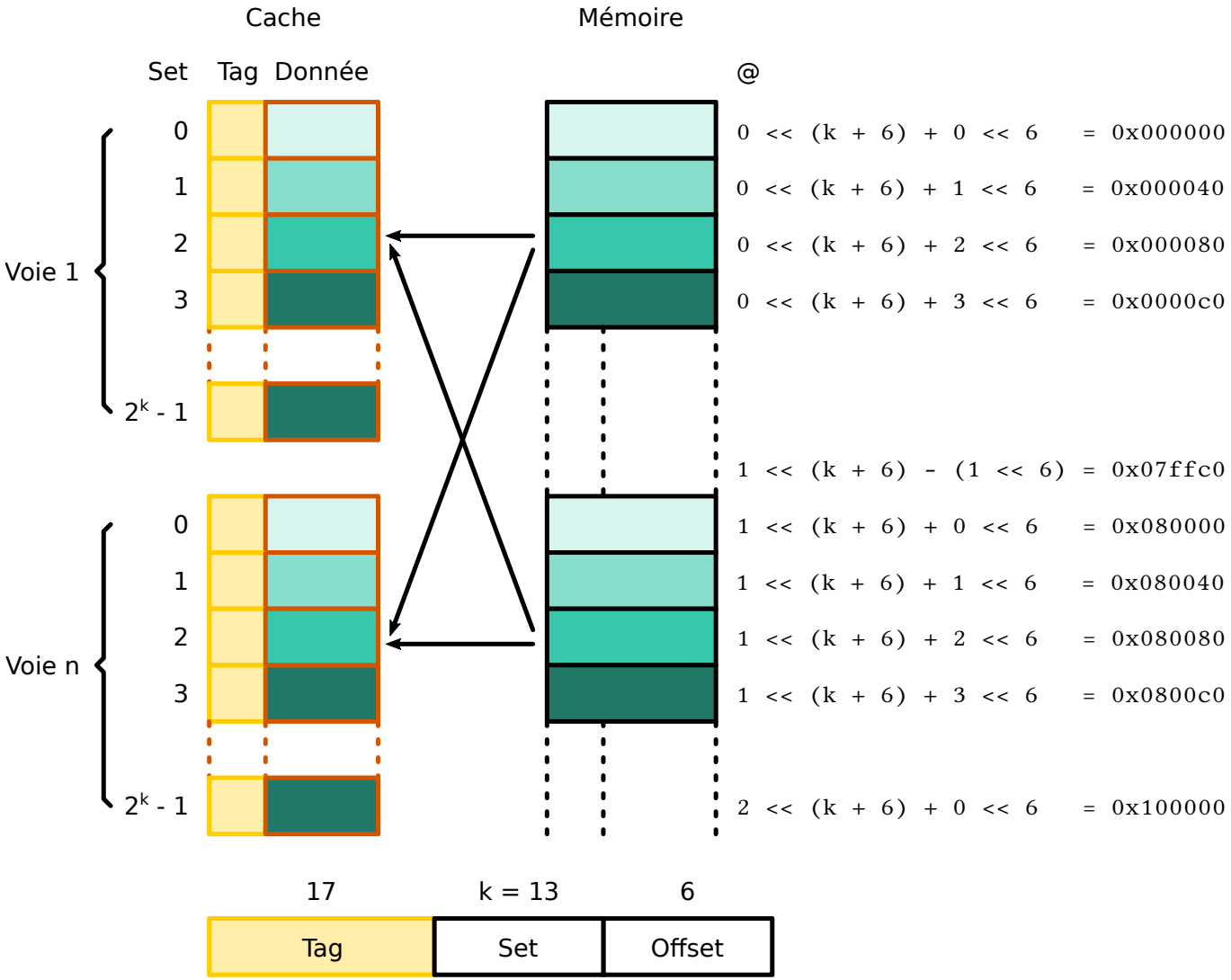
Correspondance mémoire cache n-associative



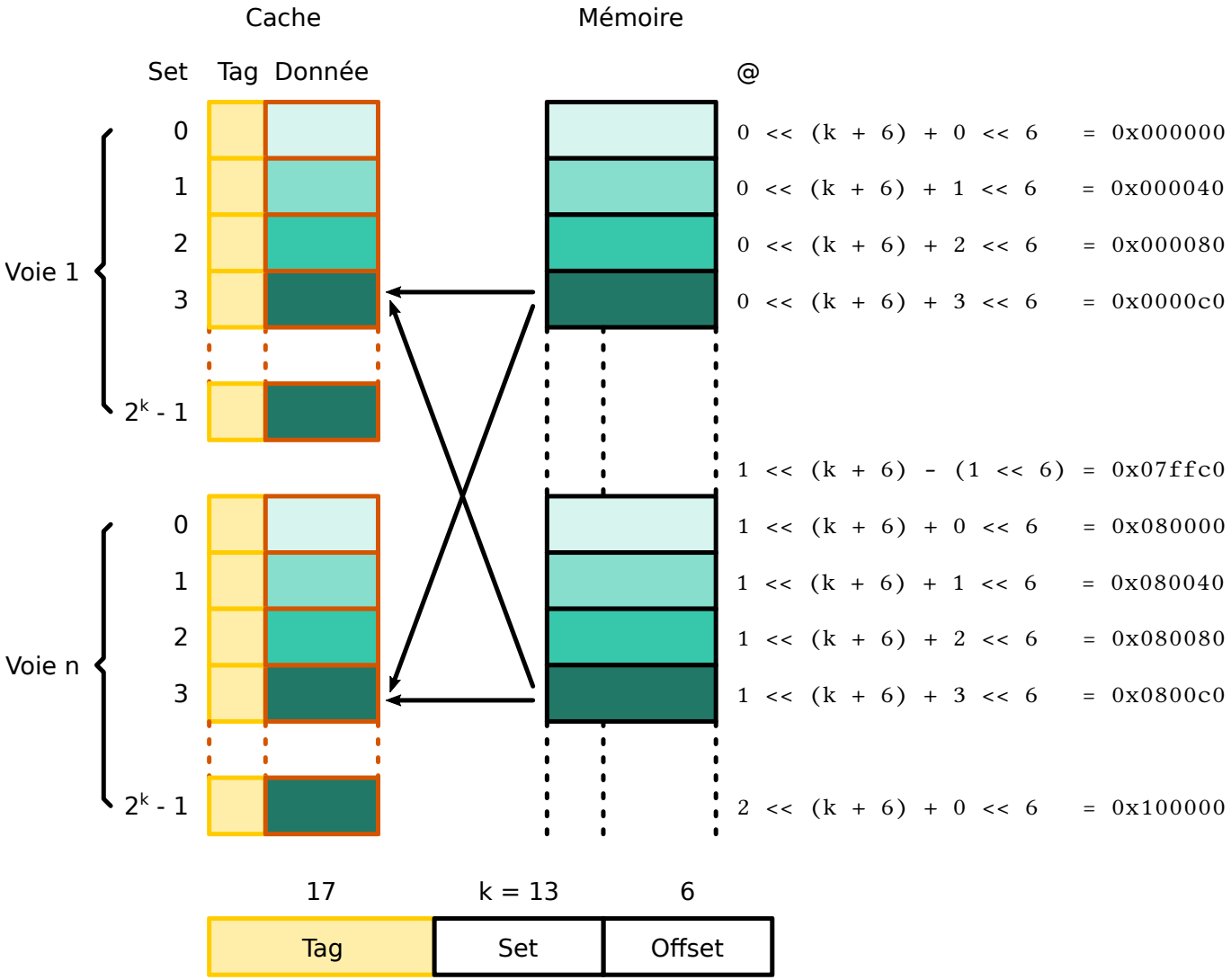
Correspondance mémoire cache n-associative



Correspondance mémoire cache n-associative



Correspondance mémoire cache n-associative



Paramètres des niveaux de cache

- Taille d'une ligne de cache : l
- Cache *n-way associative* : n lignes de cache par *set*
- Nombre de *sets* pour un cache : s
- Taille du cache de niveau $k | k < LLC = S_k = l \times n \times s$
- LLC et multicœur :
  Correspondance composée
  Un même set est présent sur les k cœurs du processeur
  Chaque ligne y sont réparties selon une fonction de correspondance donnée

Politique d'éviction des lignes de cache

Si un cache est n-associatif $\rightarrow$ politique d'éviction

- Si un *set* est plein, un ligne est supprimée : *éviction*
- Intel applique (ait) l'algorithme *Last Recently Used*

Correspondance des caches

Niveau L1

Adress virtuelle $\leftrightarrow$ cache set

Niveau L2 et L3 = LLC

Adress physique $\leftrightarrow$ cache set

Paramètres des niveaux de cache

Pour les processeur i^7 : $N = 3 \Rightarrow$ LLC $= 3$. On étudie pas ici le L_1.
Nombre de cœurs : $c = 4$. Taille de ligne de cache : $l = 64$

Niveau L2

- Taille du cache de niveau 2 : $S_2 = 256k$
- Associativité : $n_2 = 4$
- Nombre de *sets* : $s_2 = \frac{S_2}{n_2 \times l} = 1024$

Niveau L3 = LLC

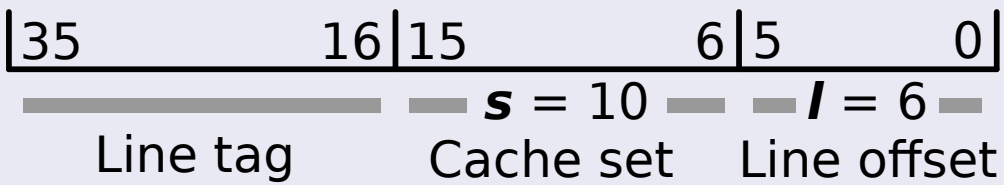
- Taille du cache de niveau LLC : $S_3 = 8M$
- Associativité : $n_3 = 16$
- Nombre de *sets* : $s_3 = \frac{S_3}{n_3 \times c \times l} = 2048$

Correspondance du cache L2

Paramètres de l'architecture

- Taille des adresses physiques : $a = 36$ bits
- Nombre de *sets* : $s_2 = 1024 \Leftrightarrow 10$ bits
- Taille de ligne de cache : $l = 64 \Leftrightarrow 6$ bits

Correspondance adressage $\leftrightarrow$ cache set



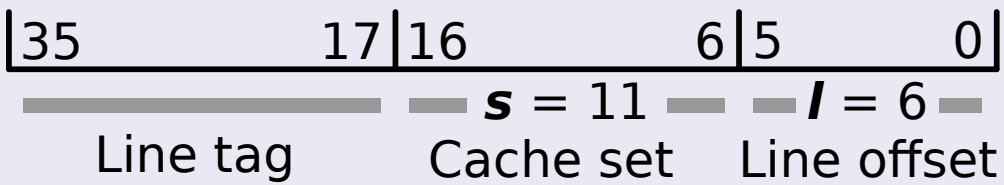
Cache physical mapping

Correspondance du cache L3 (1)

Paramètres de l'architecture

- Taille des adresses physiques : $a = 36$ bits
- Nombre de *sets* : $s_3 = 2048 \Leftrightarrow 11$ bits
- Taille de ligne de cache : $l = 64 \Leftrightarrow 6$ bits

Correspondance adressage $\leftrightarrow$ cache set



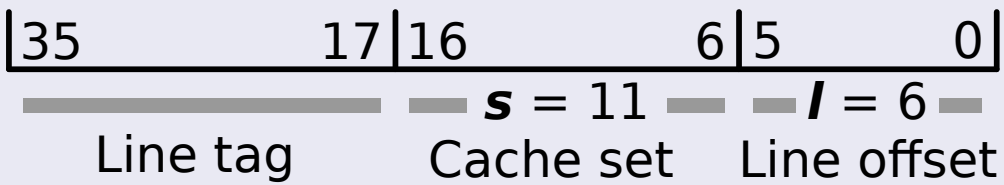
Cache physical mapping

Correspondance du cache L3 (1)

Paramètres de l'architecture

- Taille des adresses physiques : $a = 36$ bits
- Nombre de *sets* : $s_3 = 2048 \Leftrightarrow 11$ bits
- Taille de ligne de cache : $l = 64 \Leftrightarrow 6$ bits

Correspondance adressage $\leftrightarrow$ cache set



Cache physical mapping

Remarque : un cache set est représenté c fois, 1 fois par slice.
Conséquence : partager un cache set ne suffit pas forcément pour obverser un autre processus...

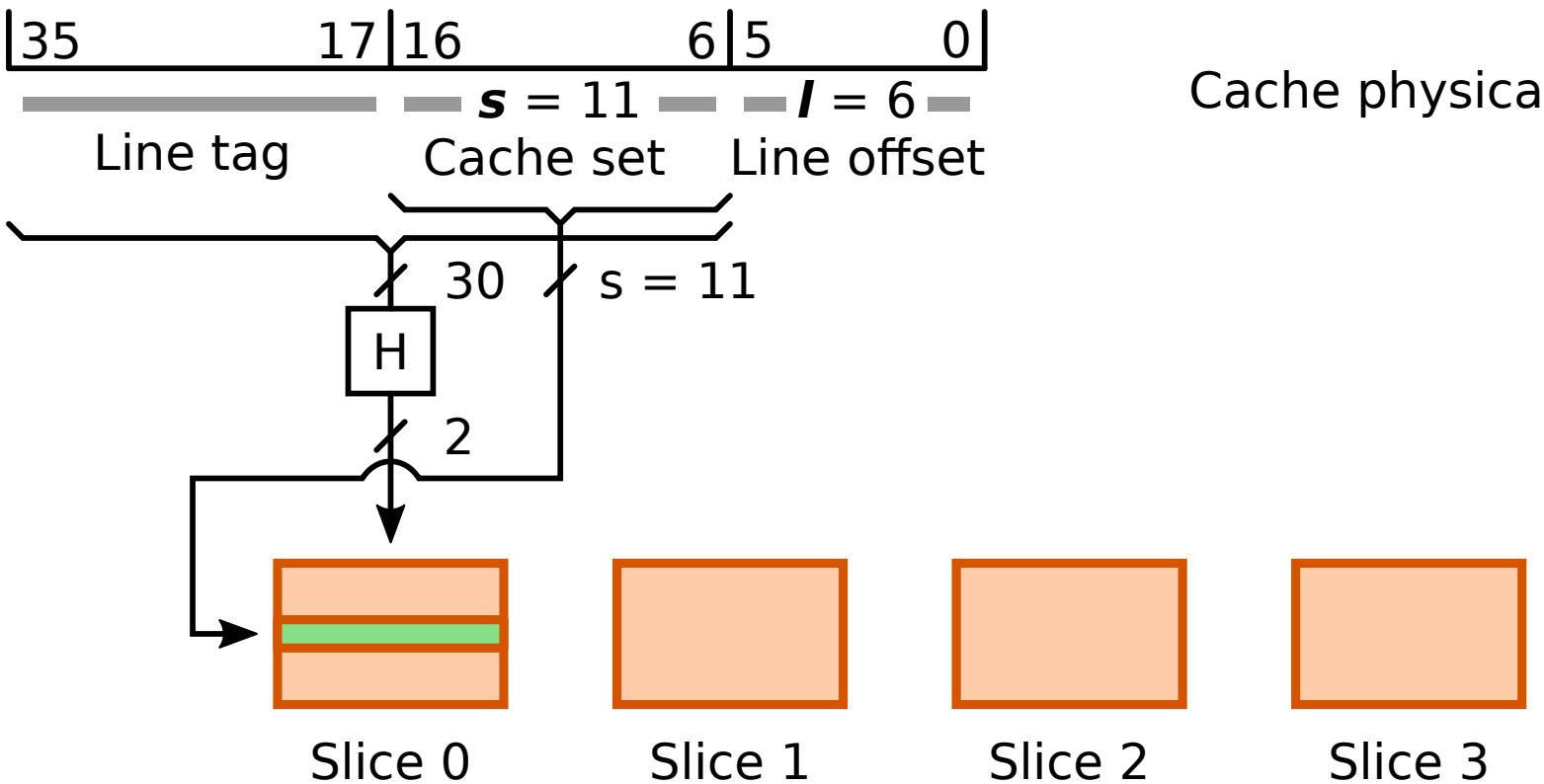
Correspondance du cache L3 (2)

Fonction de correspondance du cache slice

$H : S_3 \times T_3 \rightarrow C$

Associe une ligne de cache au slice à l'aide des bits de tag et de cache set.

Fonction de hashage non linéraire.



Cache physical mapping

Cache et programme utilisateur

En *x86* il est possible d'explicitement vider une ligne de cache depuis l'espace utilisateur. C'est à dire depuis un processus s'exécutant en mode utilisateur ($\neq$ noyau).

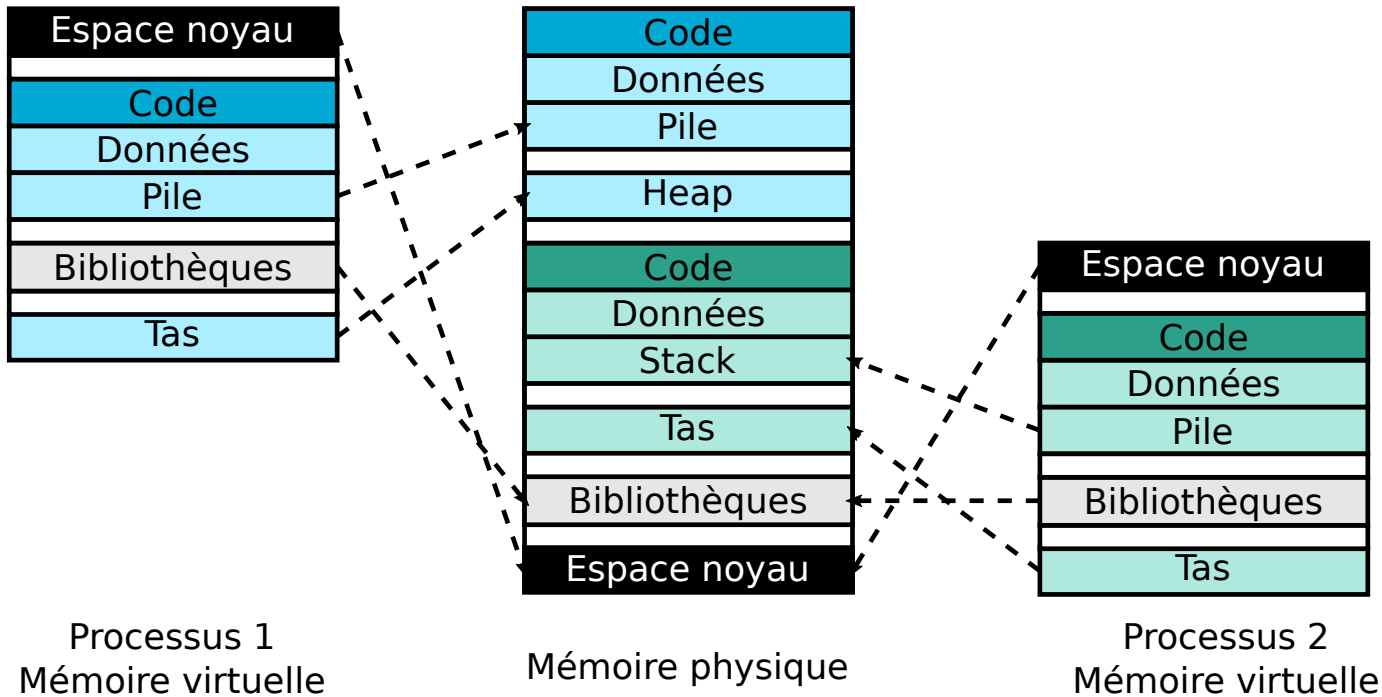
Instruction `clfush`

`clflush <référence mémoire virtuelle>`
Permet de vider la ligne de cache associée à l'édresse virtuelle en paramètre.
Exemple : `clflush (%rax)`
Exemple : `clflush étiquette`

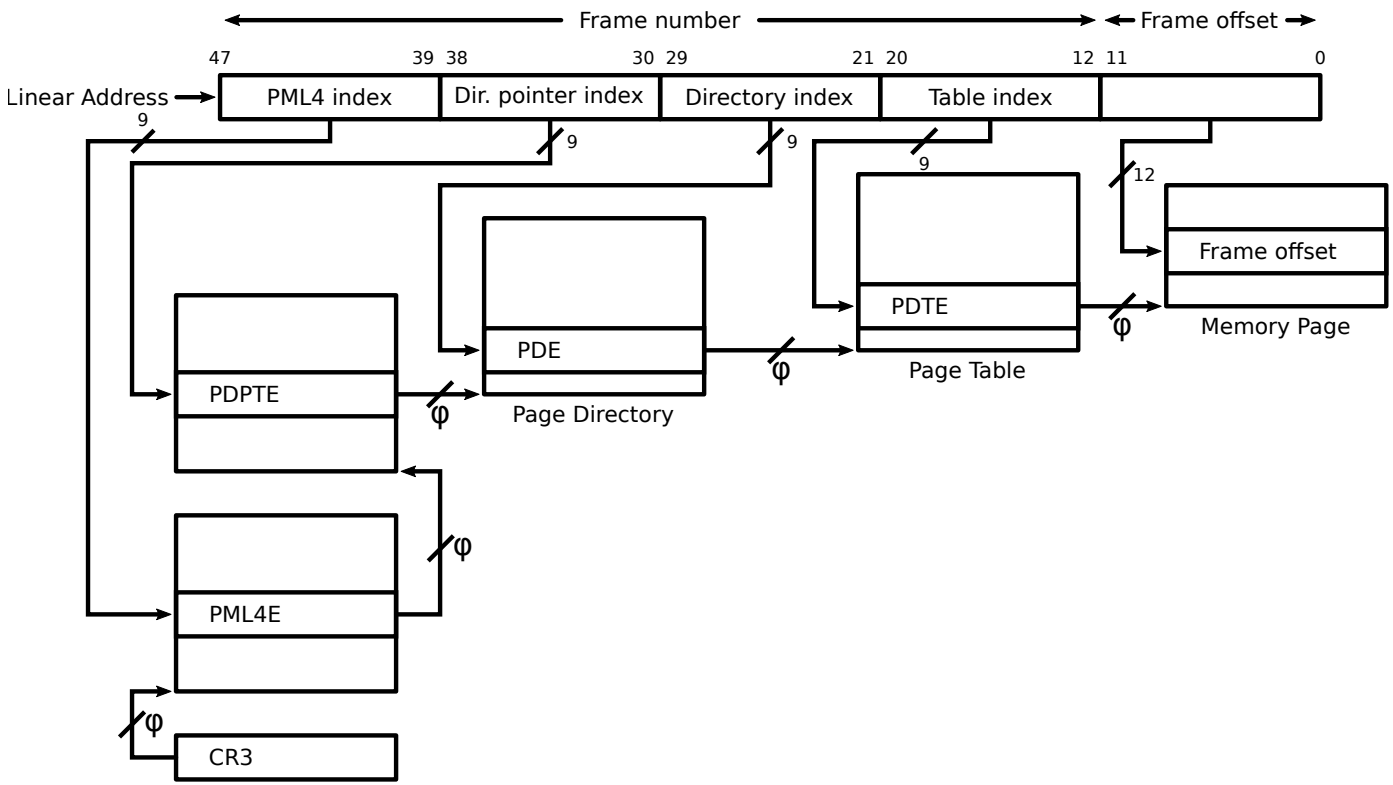
Conséquence : un utilisateur peut vider n'importe quelle ligne de cache à partir du moment où elle est adressée dans son espace d'adressage virtuel !

Outline

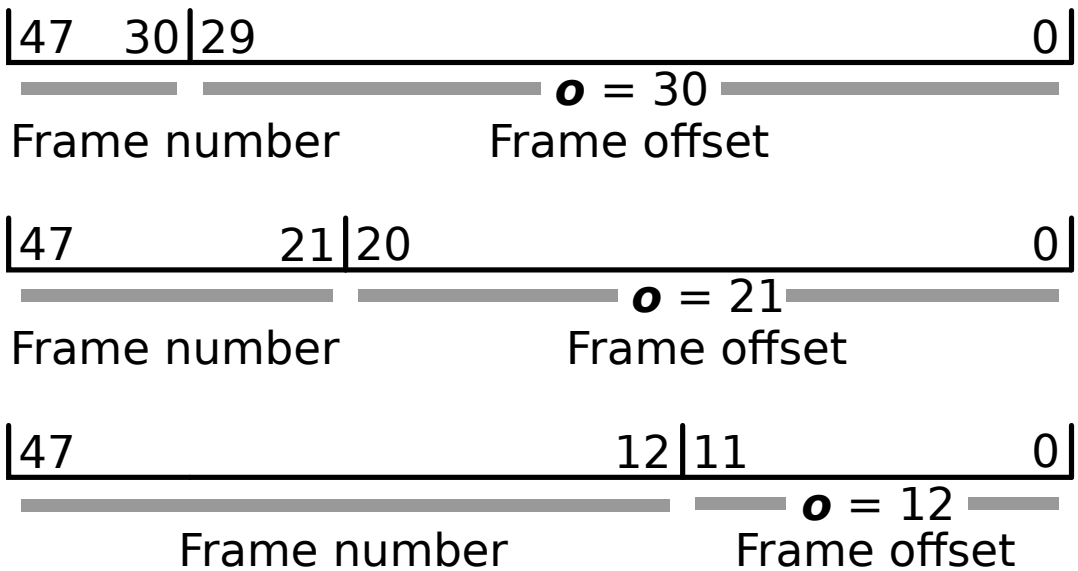
Espace mémoire virtuel des processus UNIX



Pagination à 4 ko avec la MMU Intel



Tailles des pages



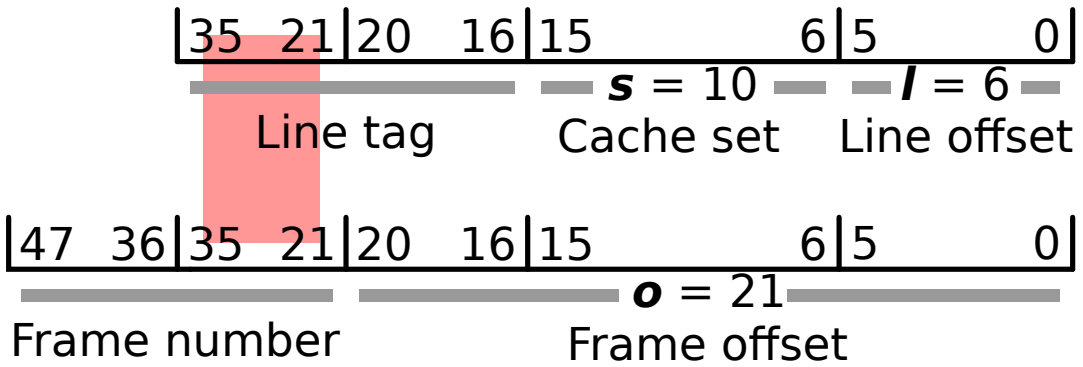
1GB pages

2MB pages

4kB pages

Tailles des pages et correspondance du cache (1)

Pagination à 2Mo et correspondance avec le cache L2.



Cache physical mapping

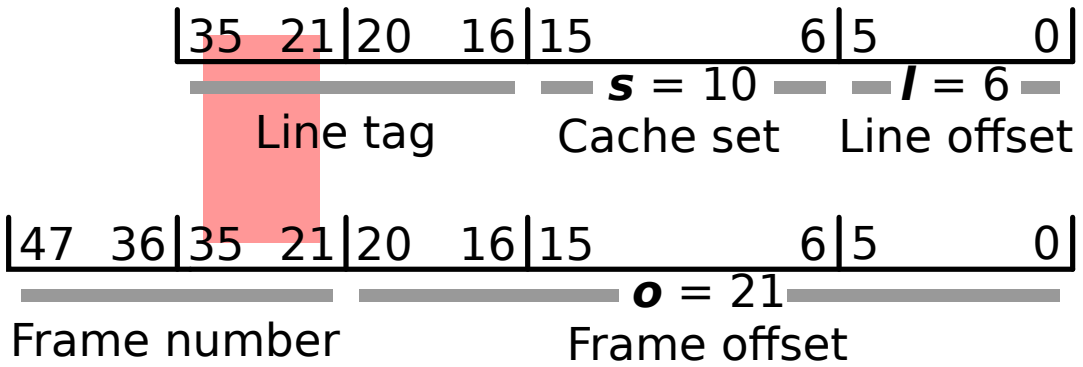
MMU virtual addressing

Les 21 bits de poids faible de l'adresse virtuelle ne sont pas traduits :

- Accès aux 6 bits *ligne offset*
- Accès aux 10 bits *cache set*
- Accès à 5 bits de tag
- Seuls 15 bits de poid fort de tag sont inconnus

Tailles des pages et correspondance du cache (1)

Pagination à 2Mo et correspondance avec le cache L2.



Cache physical mapping

MMU virtual addressing

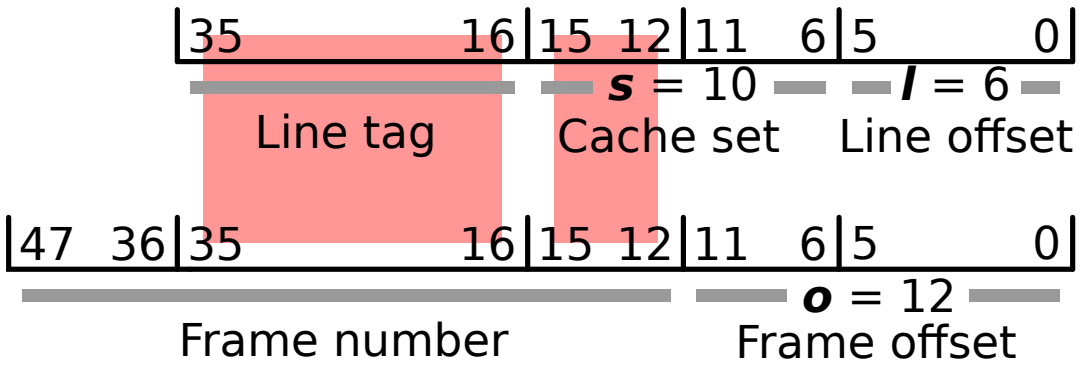
Les 21 bits de poids faible de l'adresse virtuelle ne sont pas traduits :

- Accès aux 6 bits *ligne offset*
- Accès aux 10 bits *cache set*
- Accès à 5 bits de tag
- Seuls 15 bits de poid fort de tag sont inconnus

Conséquence : associativité de 4 $\rightarrow$ avec une page de 2Mo je peux obtenir $32 > 4$ lignes de caches différentes, donc remplir un cache set.

Tailles des pages et correspondance du cache (2)

Pagination à 2Mo et correspondance avec le cache L2.



Cache physical mapping

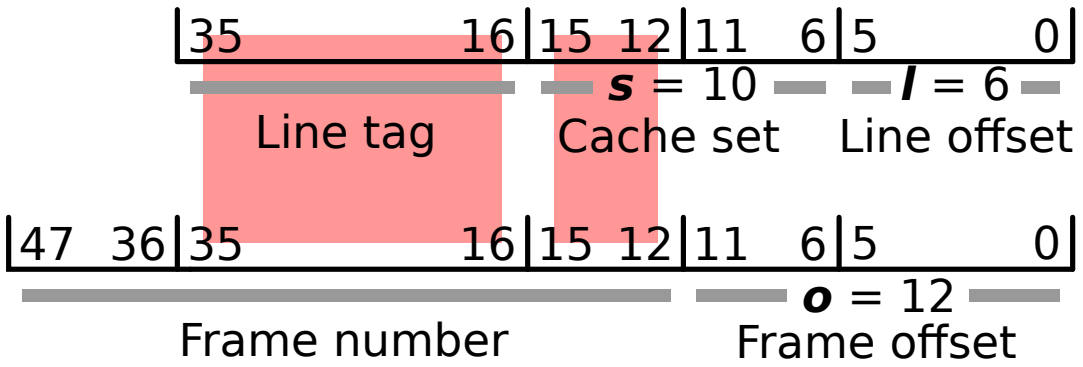
MMU virtual addressing

Les 12 bits de poids faible de l'adresse virtuelle ne sont pas traduits :

- Accès aux 6 bits *ligne offset*
- Accès aux 6 bits de poid faible de *cache set*
- Les 4 bits de poid fort de *cache set* sont inconnus
- Les 20 bits de tag *tag* sont inconnus

Tailles des pages et correspondance du cache (2)

Pagination à 2Mo et correspondance avec le cache L2.



Cache physical mapping

MMU virtual addressing

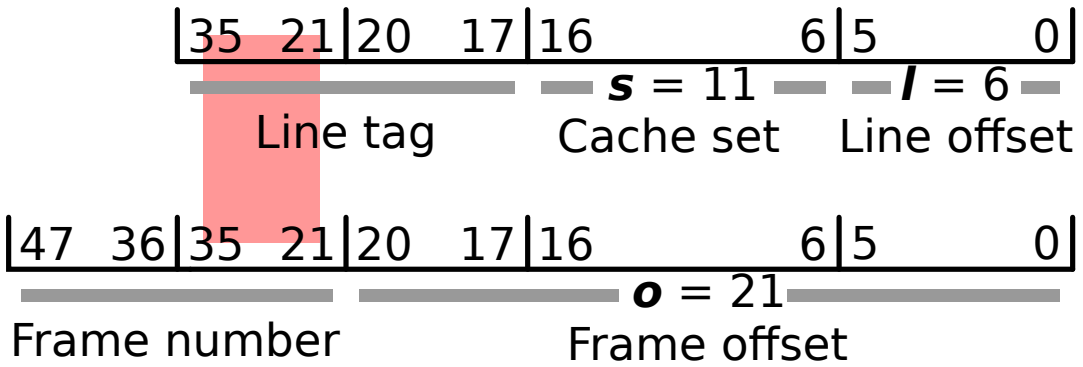
Les 12 bits de poids faible de l'adresse virtuelle ne sont pas traduits :

- Accès aux 6 bits *ligne offset*
- Accès aux 6 bits de poid faible de *cache set*
- Les 4 bits de poid fort de *cache set* sont inconnus
- Les 20 bits de tag *tag* sont inconnus

Conséquence : je ne sais pas remplir "facilement" un cache set en particulier.

Tailles des pages et correspondance du cache (3)

Pagination à 2Mo et correspondance avec le cache L3.



Cache physical mapping

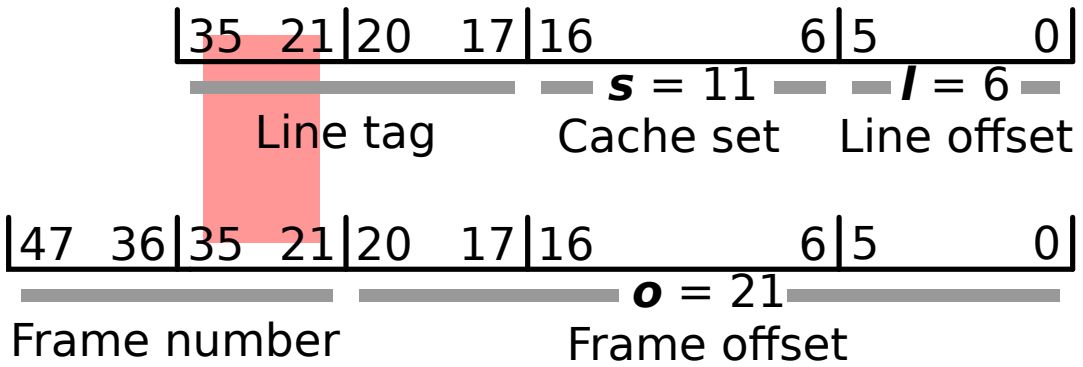
MMU virtual addressing

Les 21 bits de poids faible de l'adresse virtuelle ne sont pas traduits :

- Accès aux 6 bits *ligne offset*
- Accès aux 13 bits *cache set*
- Accès à 2 bits de tag
- Seuls 15 bits de poid fort de tag sont inconnus

Tailles des pages et correspondance du cache (3)

Pagination à 2Mo et correspondance avec le cache L3.



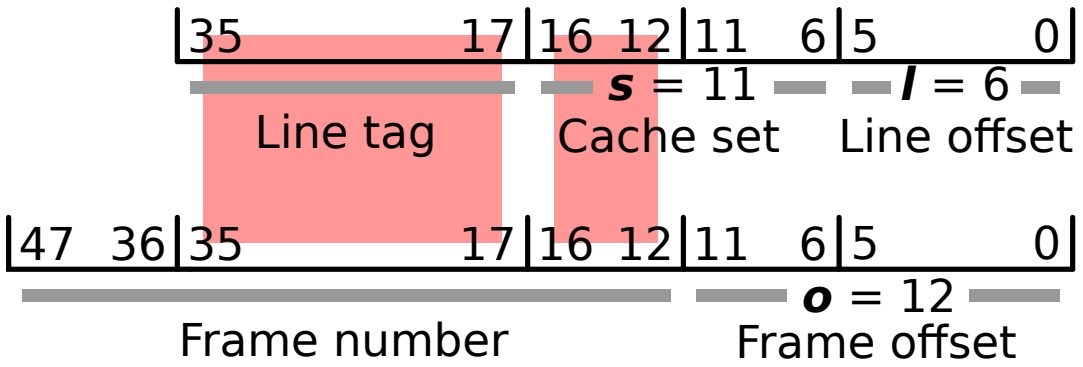
Les 21 bits de poids faible de l'adresse virtuelle ne sont pas traduits :

- Accès aux 6 bits *ligne offset*
- Accès aux 13 bits *cache set*
- Accès à 2 bits de tag
- Seuls 15 bits de poid fort de tag sont inconnus

Conséquence : associativité de 16 $\rightarrow$ avec 4 pages de 2Mo je peux obtenir 16 lignes de caches différentes, donc remplir un cache set.

Tailles des pages et correspondance du cache (4)

Pagination à 2Mo et correspondance avec le cache L3.



Cache physical mapping

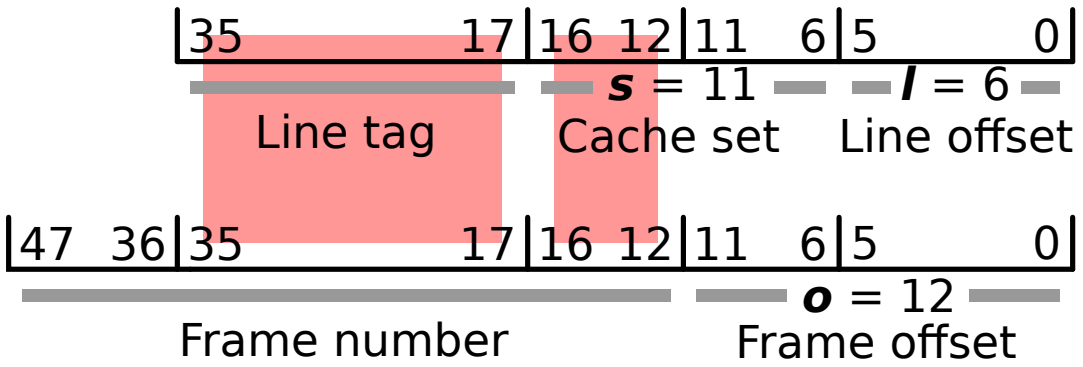
MMU virtual addressing

Les 12 bits de poids faible de l'adresse virtuelle ne sont pas traduits :

- Accès aux 6 bits *ligne offset*
- Accès aux 6 bits de poid faible de *cache set*
- Les 7 bits de poid fort de *cache set* sont inconnus
- Les 17 bits de *tag* sont inconnus

Tailles des pages et correspondance du cache (4)

Pagination à 2Mo et correspondance avec le cache L3.



Cache physical mapping

MMU virtual addressing

Les 12 bits de poids faible de l'adresse virtuelle ne sont pas traduits :

- Accès aux 6 bits *ligne offset*
- Accès aux 6 bits de poid faible de *cache set*
- Les 7 bits de poid fort de *cache set* sont inconnus
- Les 17 bits de *tag* sont inconnus

Conséquence : je ne sais pas remplir "facilement" un cache set en particulier.

Outline

Objectif d'une attaque temporelle sur les caches

Canal auxiliaire

Extraire de l'information d'un processus cible

Canal caché

Communiquer à l'aide d'un canal non autorisé

Mémoire partagée (1)

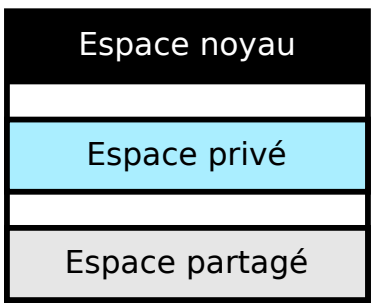
Définition

Les deux processus partagent une adresse virtuelle qui va nécessairement être traduite en même mémoire physique.

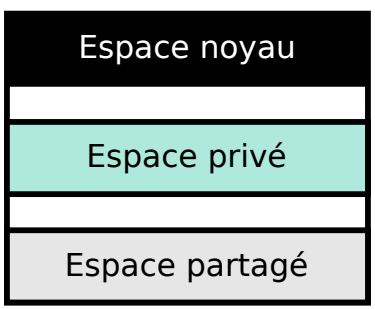
Conséquence 1 : même tag, même cache set.

Conséquence 2 : `clflush <@partagée>` flushe la ligne partagée.

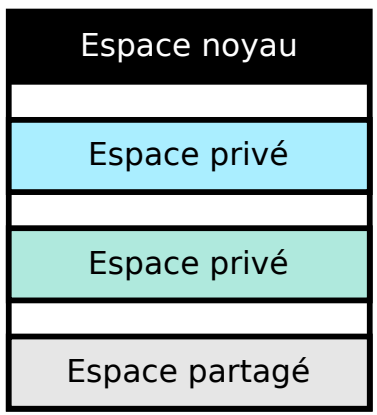
Mémoire partagée (2)



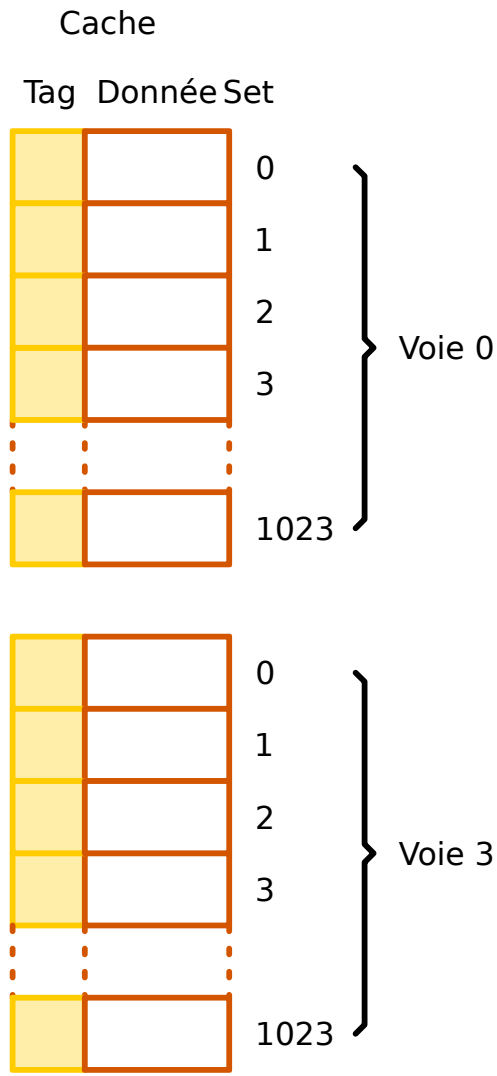
Processus 1
Mémoire virtuelle



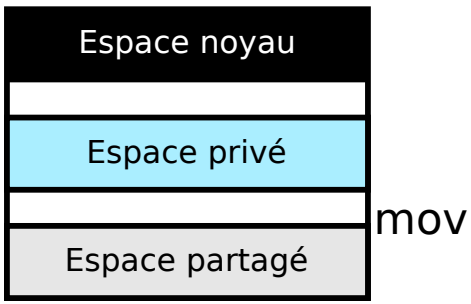
Processus 2
Mémoire virtuelle



Mémoire physique

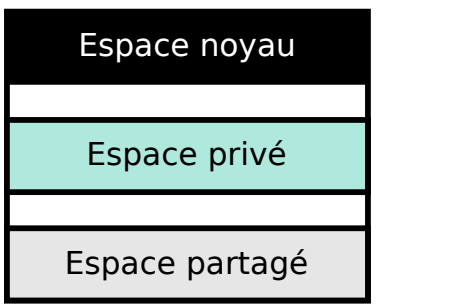


Mémoire partagée (2)

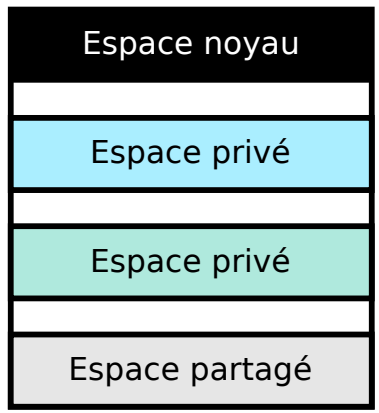


mov

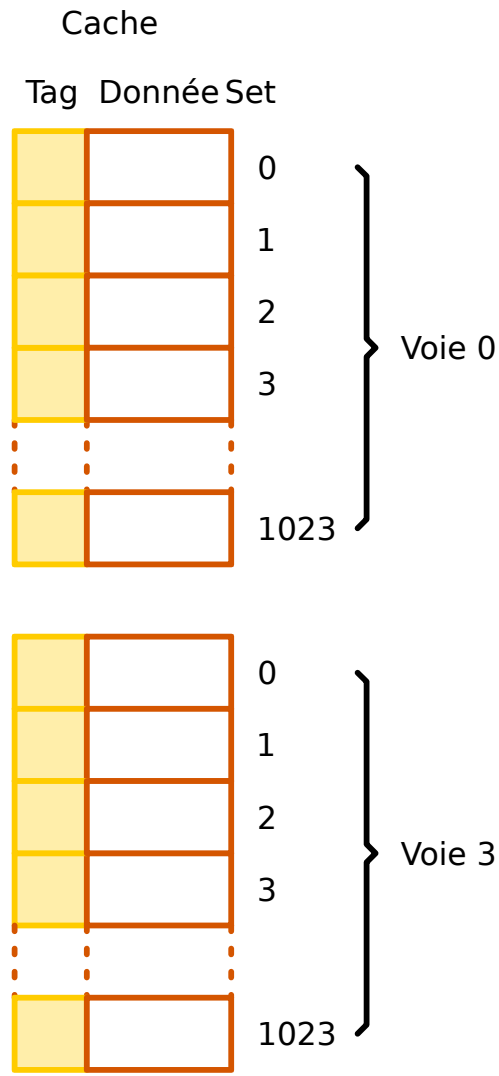
Processus 1
Mémoire virtuelle



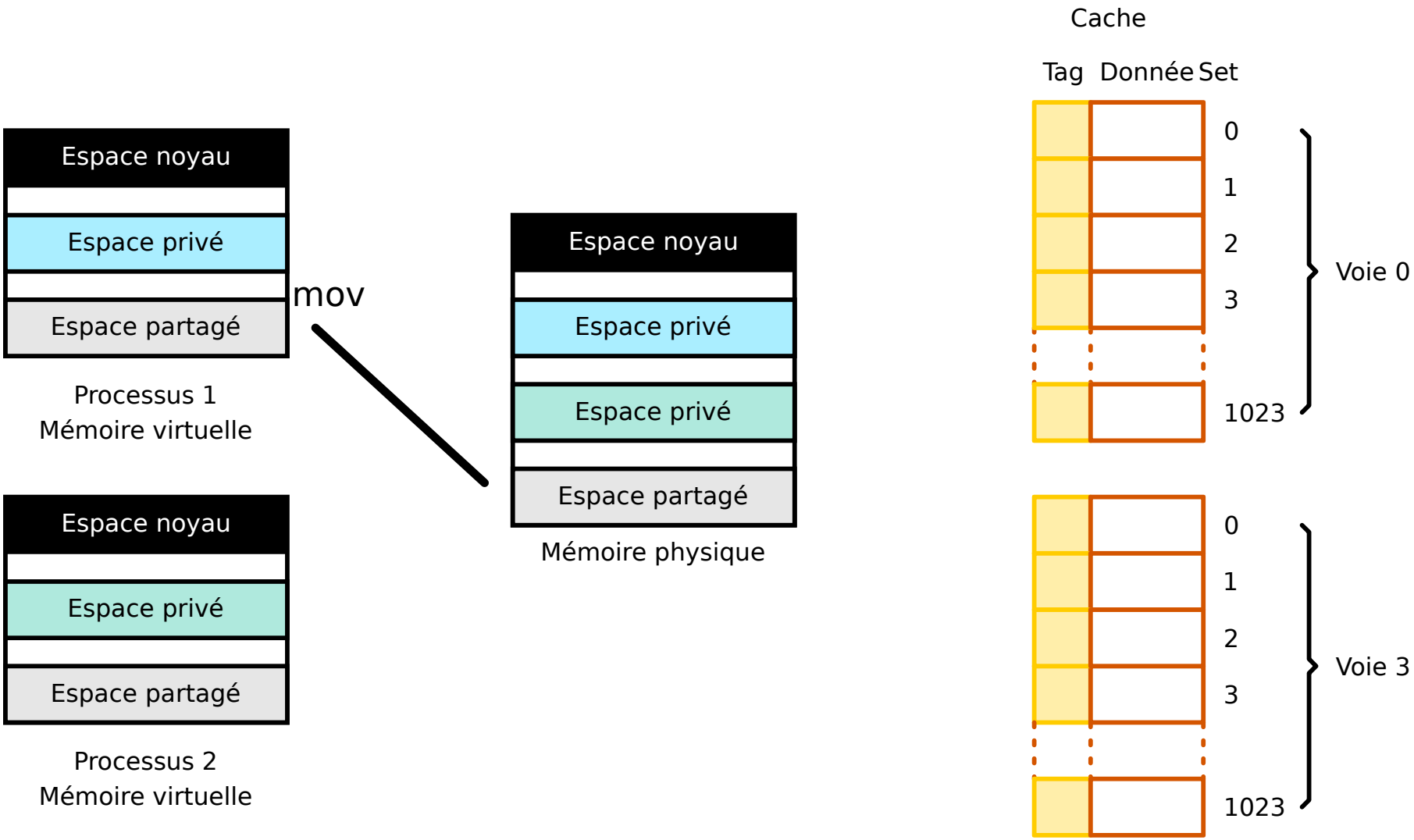
Processus 2
Mémoire virtuelle



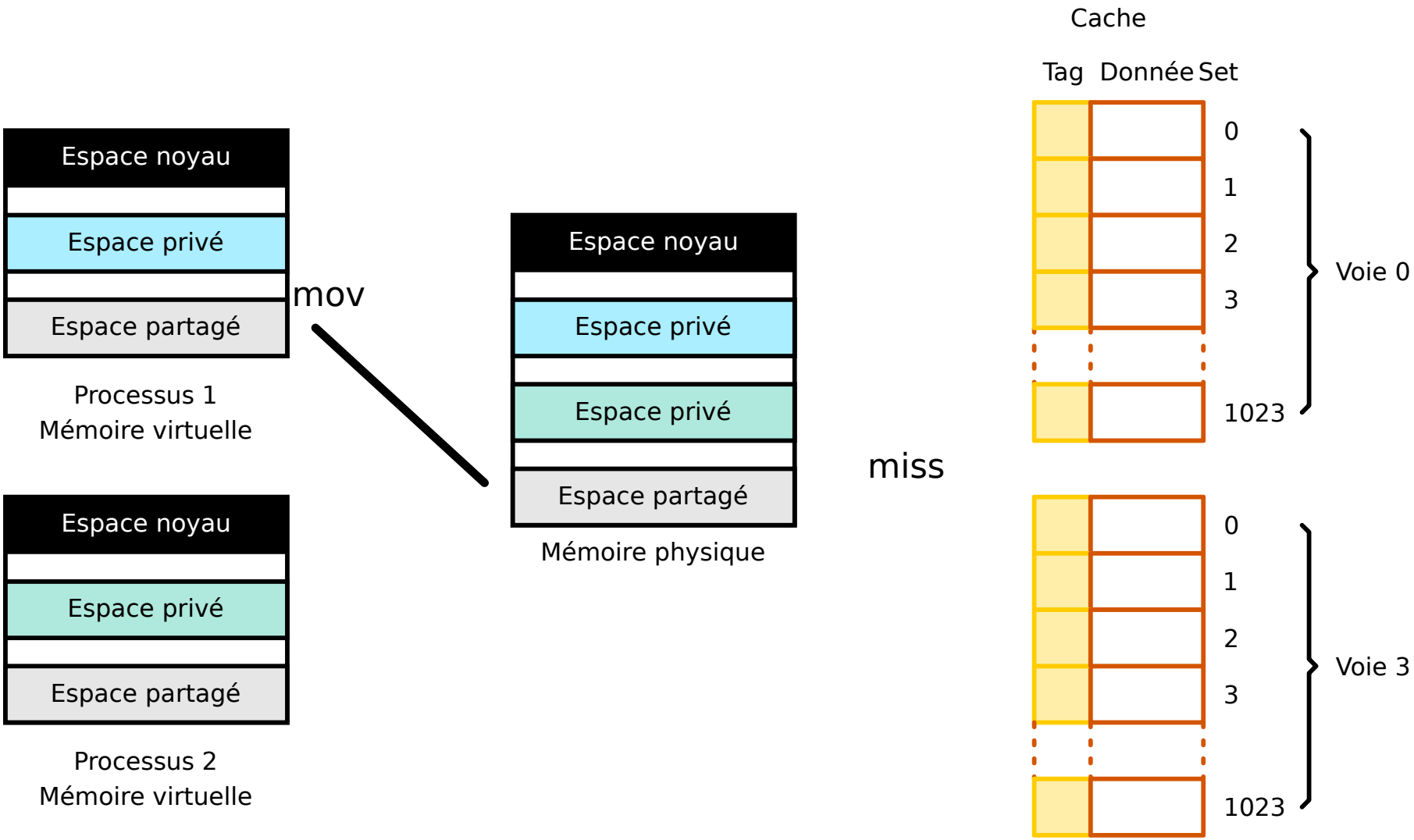
Mémoire physique



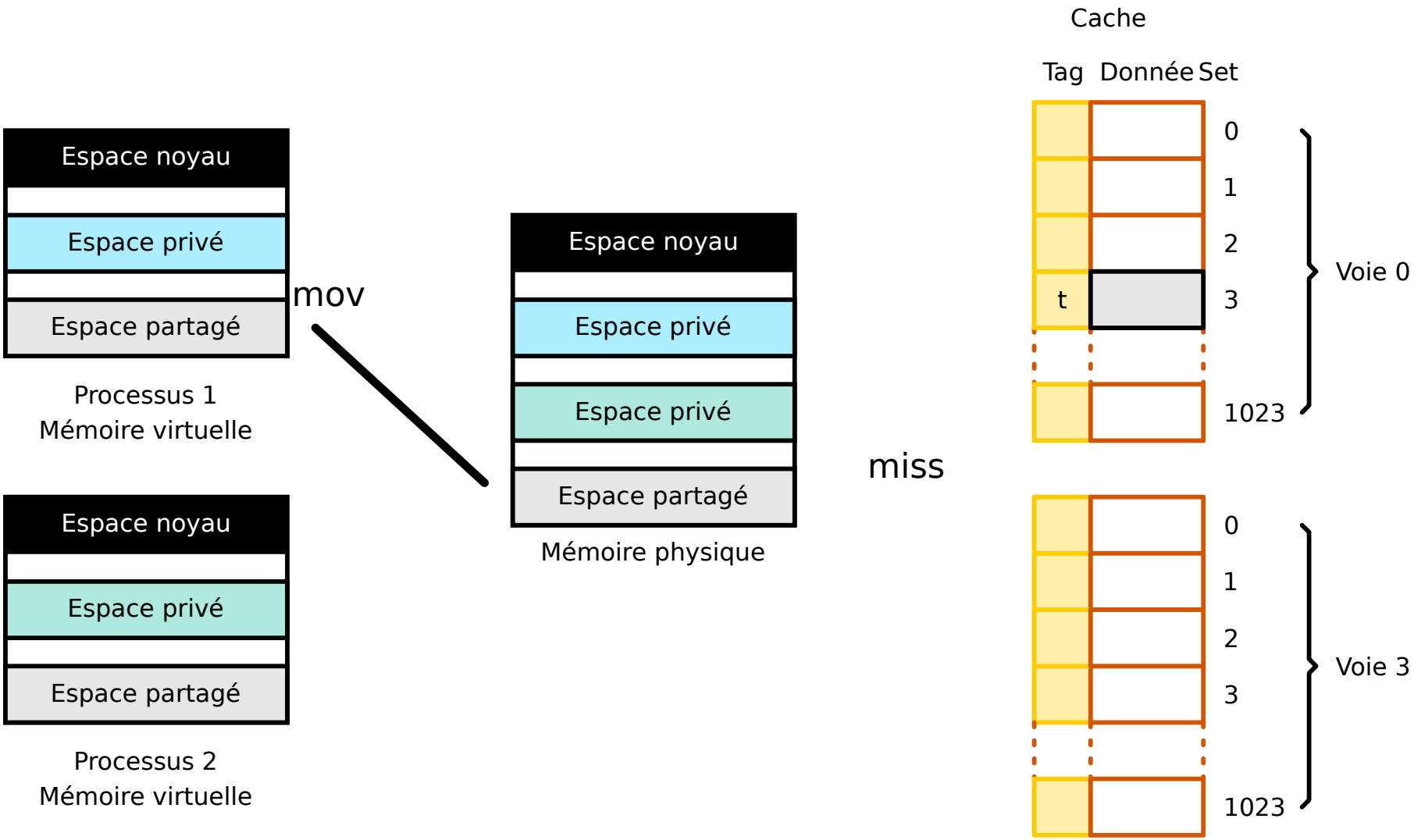
Mémoire partagée (2)



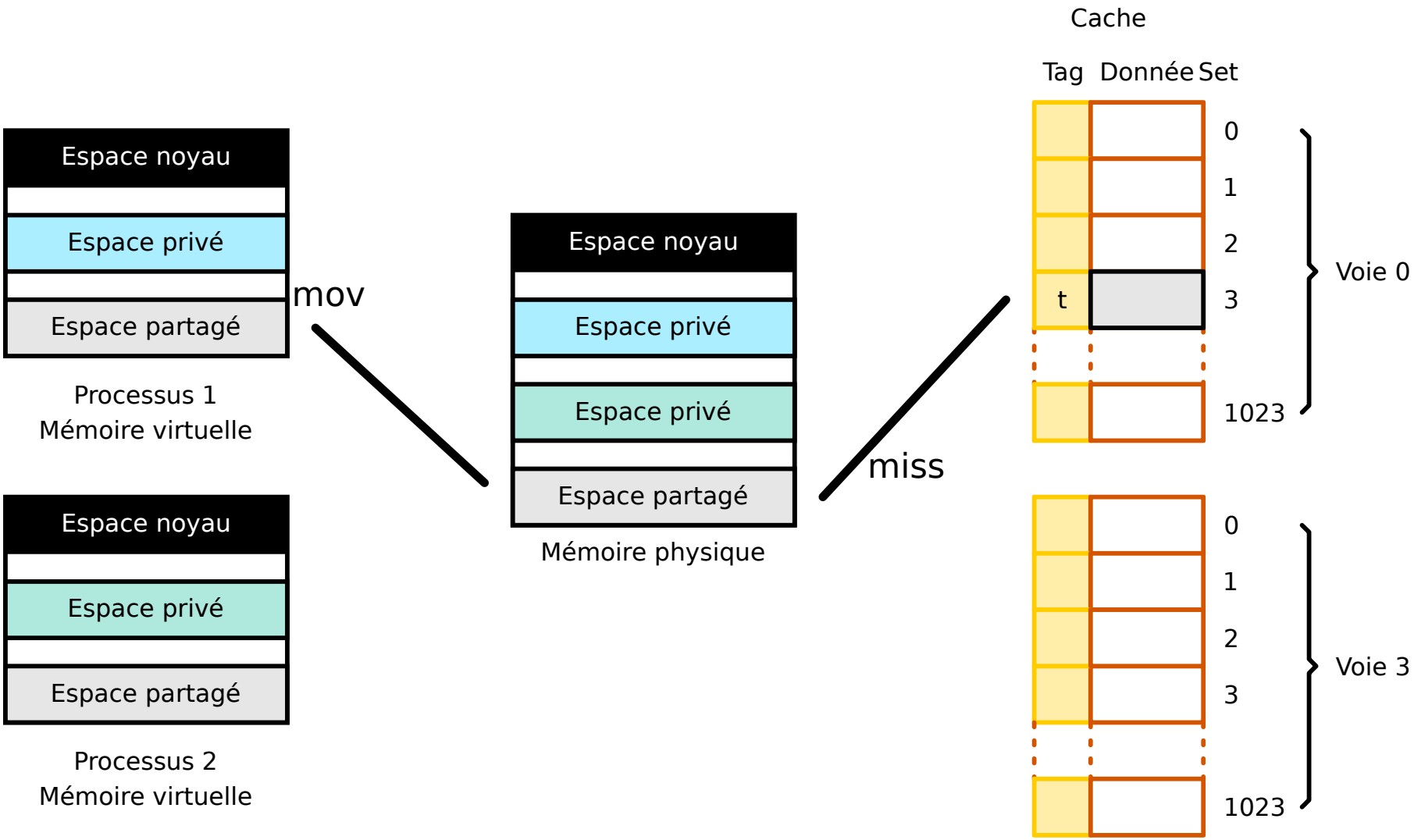
Mémoire partagée (2)



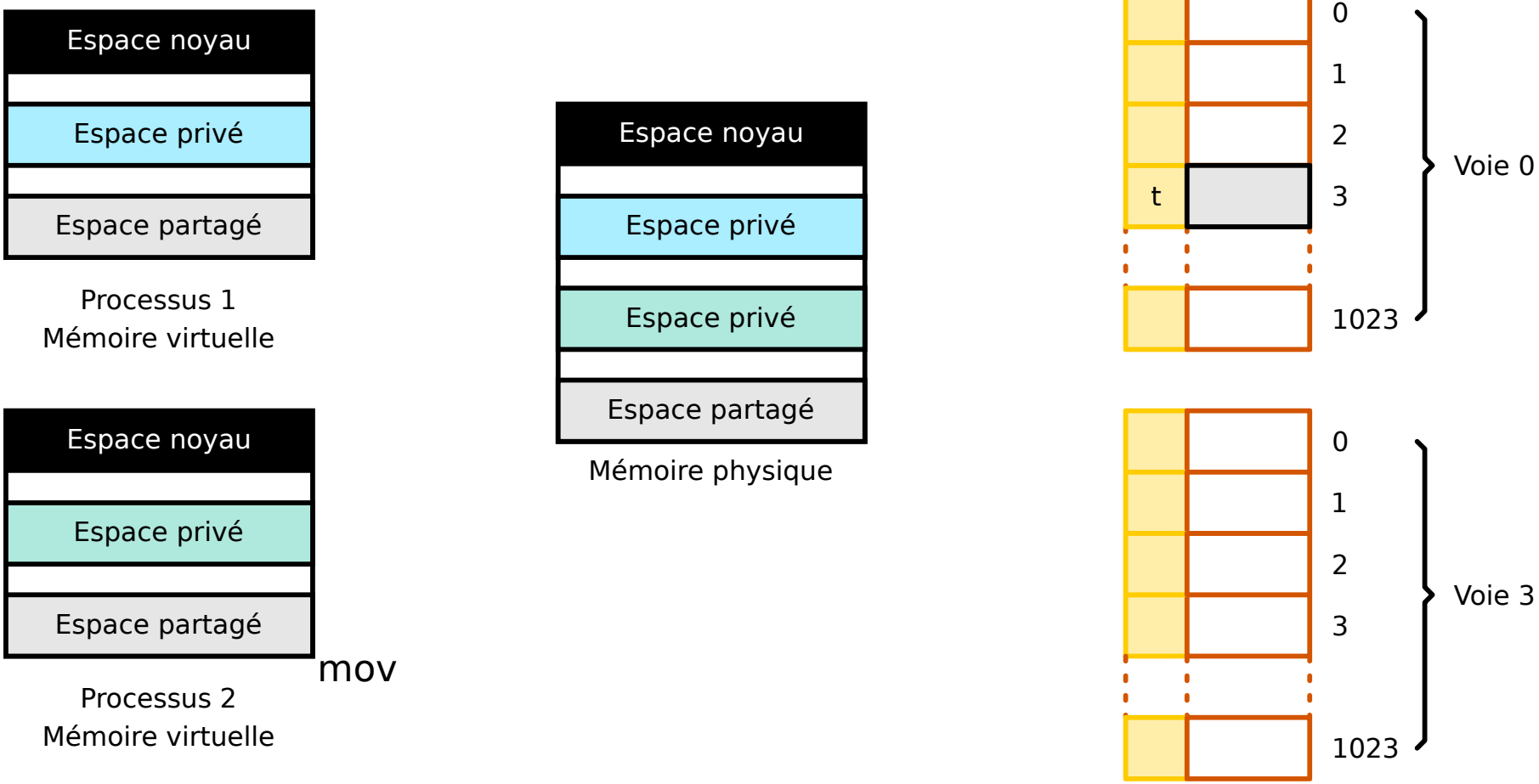
Mémoire partagée (2)



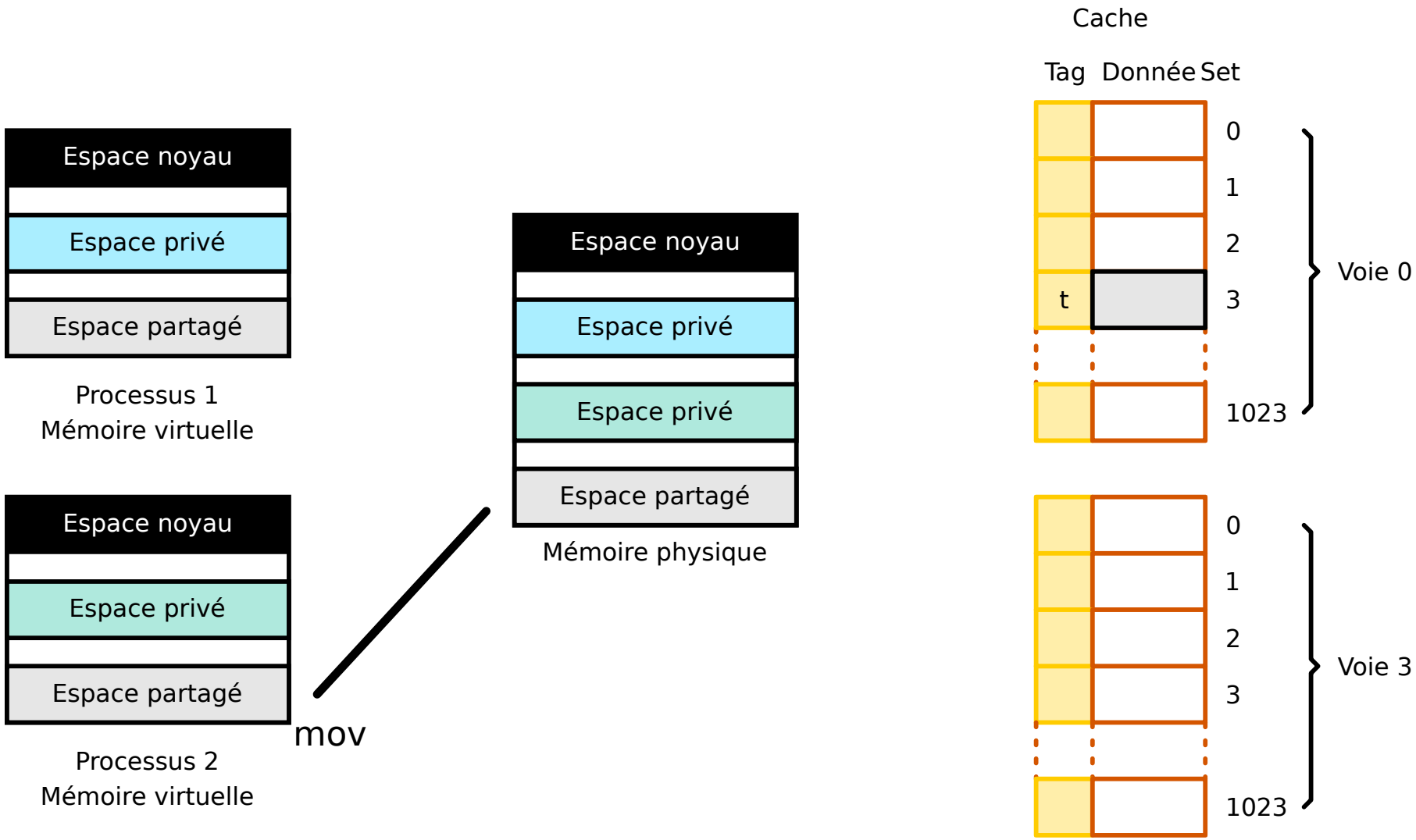
Mémoire partagée (2)



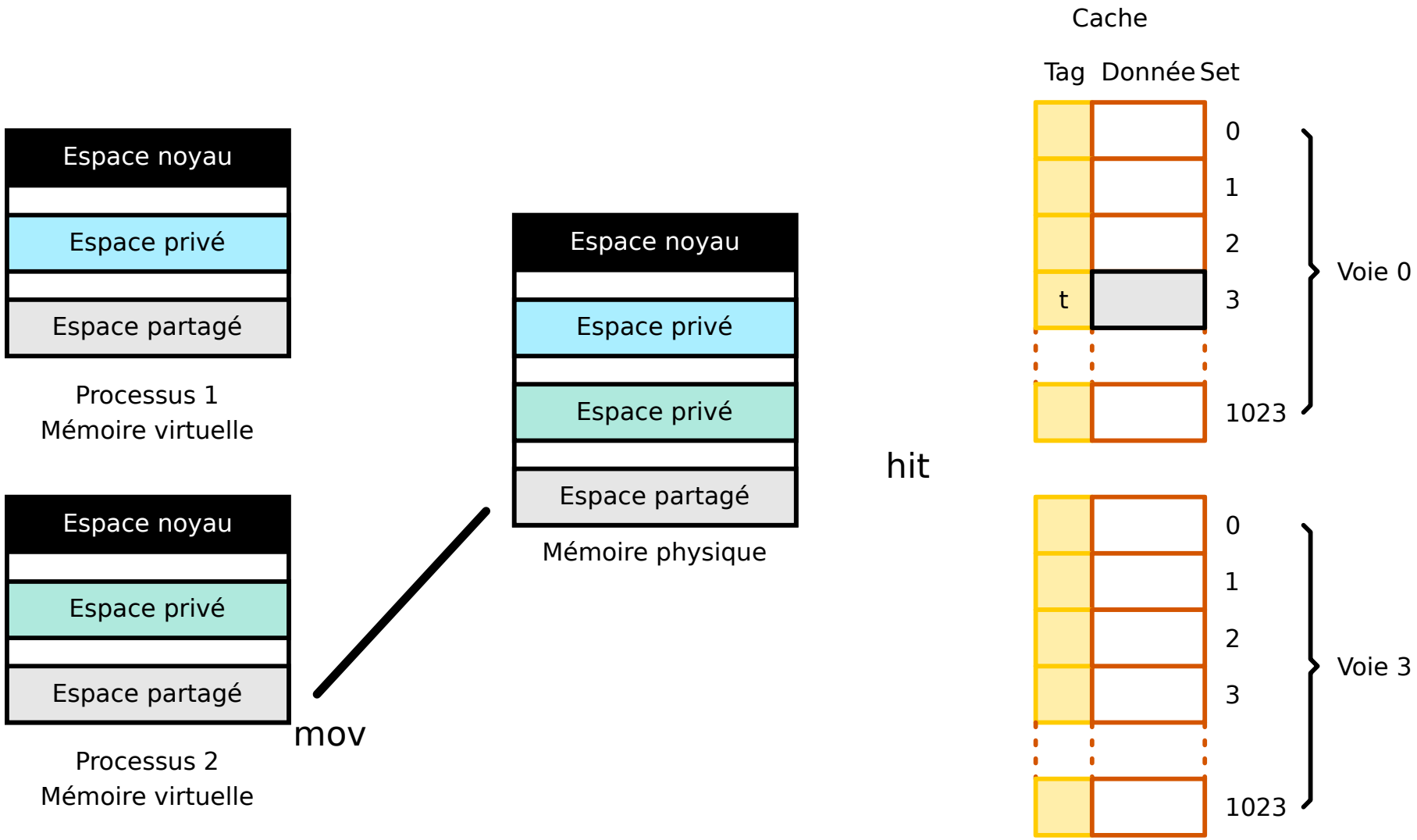
Mémoire partagée (2)



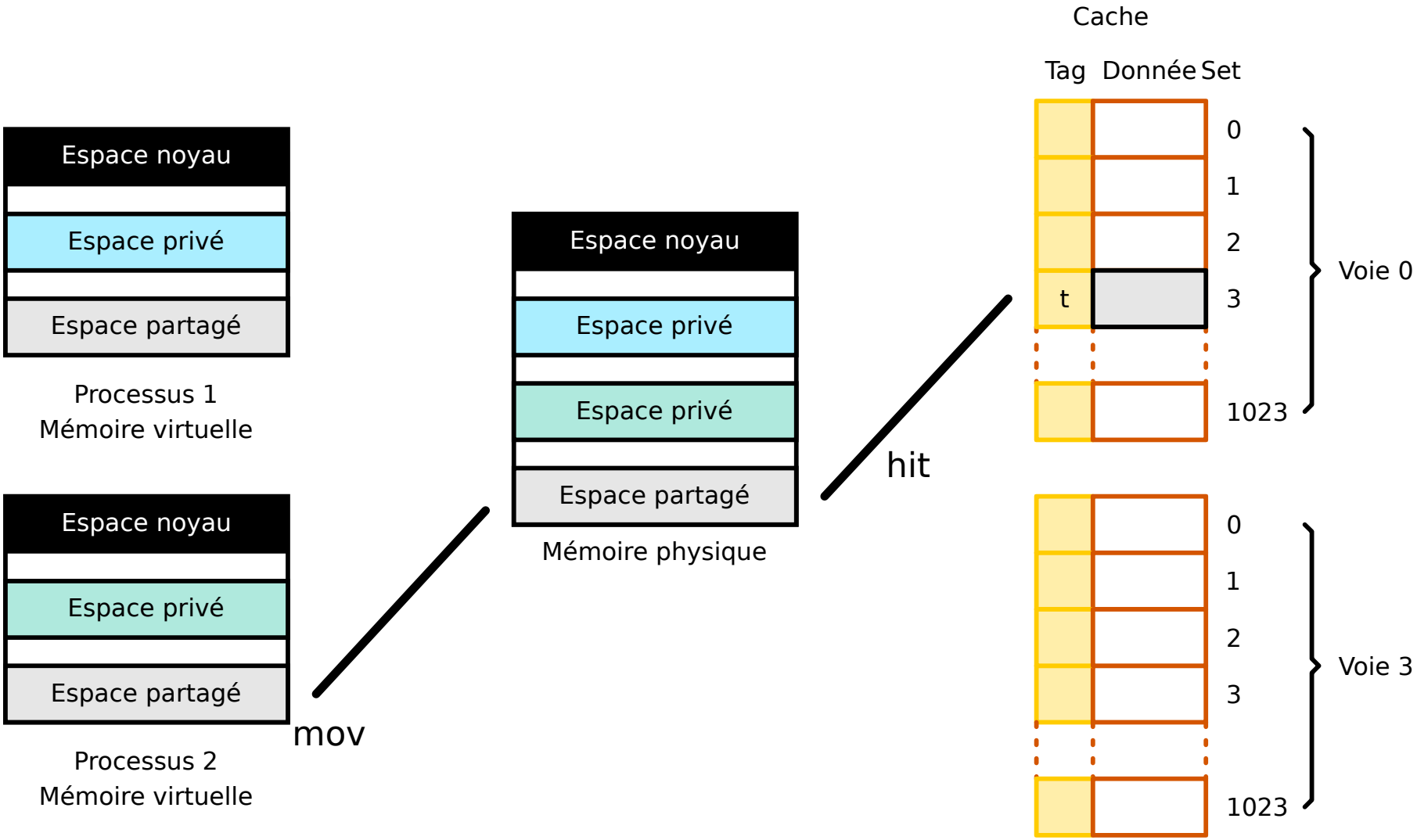
Mémoire partagée (2)



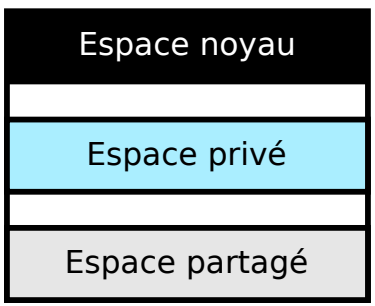
Mémoire partagée (2)



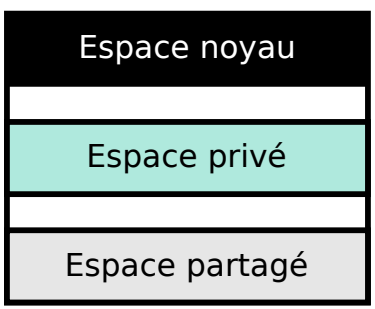
Mémoire partagée (2)



Mémoire partagée (2)

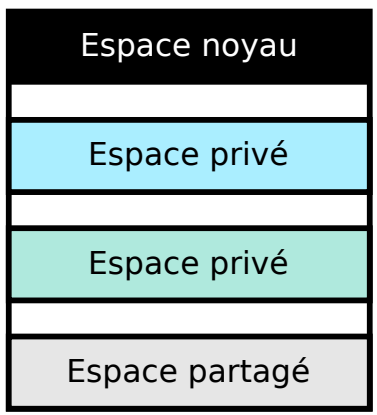


Processus 1
Mémoire virtuelle

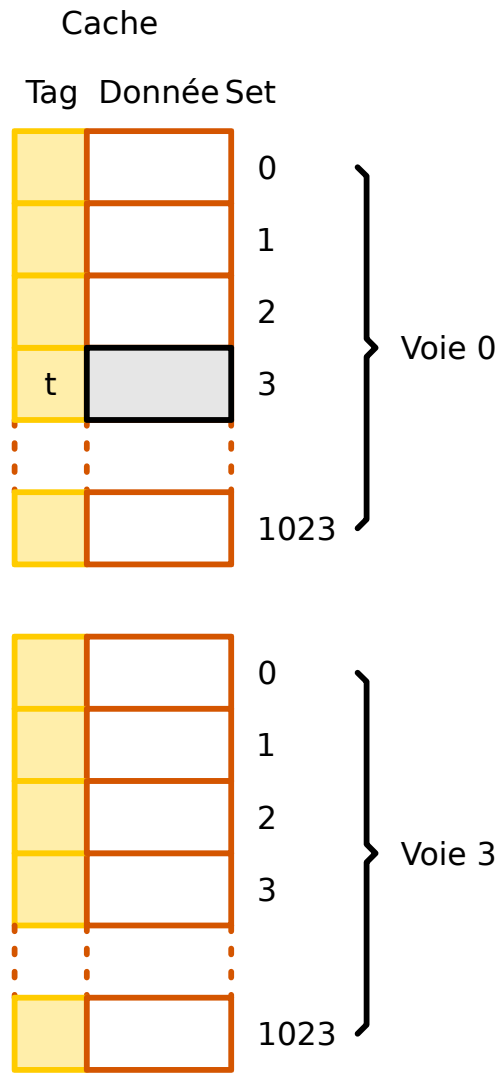


Processus 2
Mémoire virtuelle

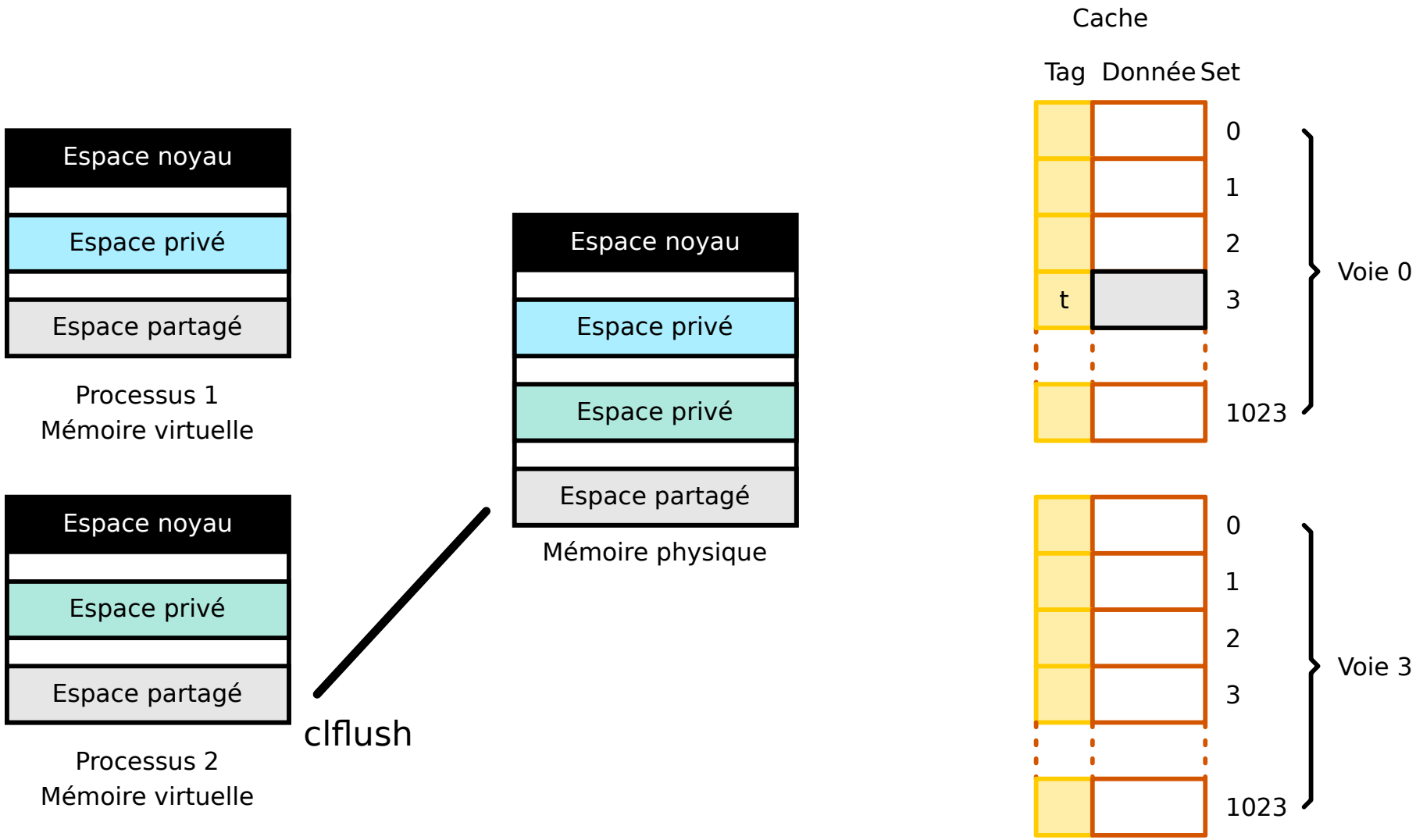
clflush



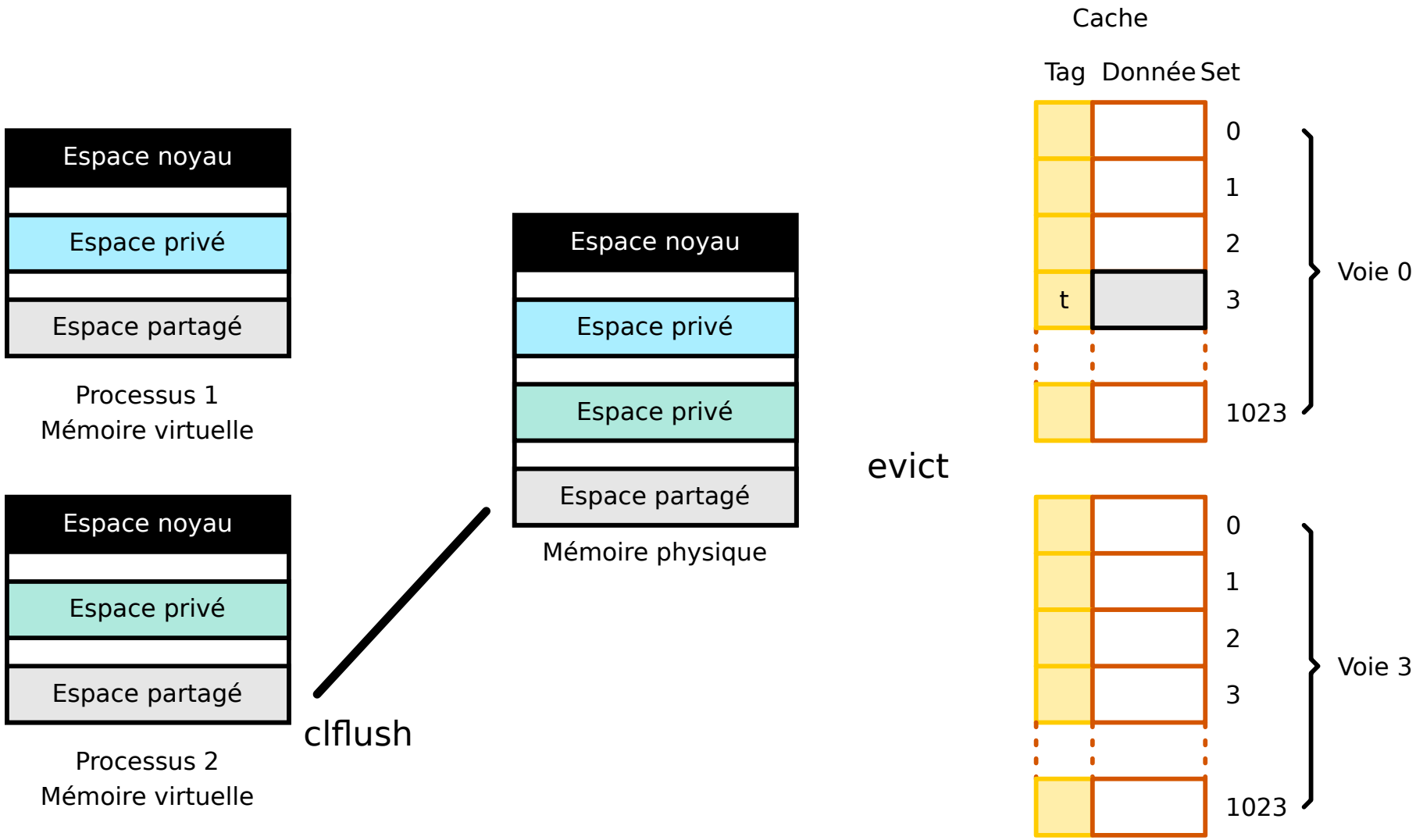
Mémoire physique



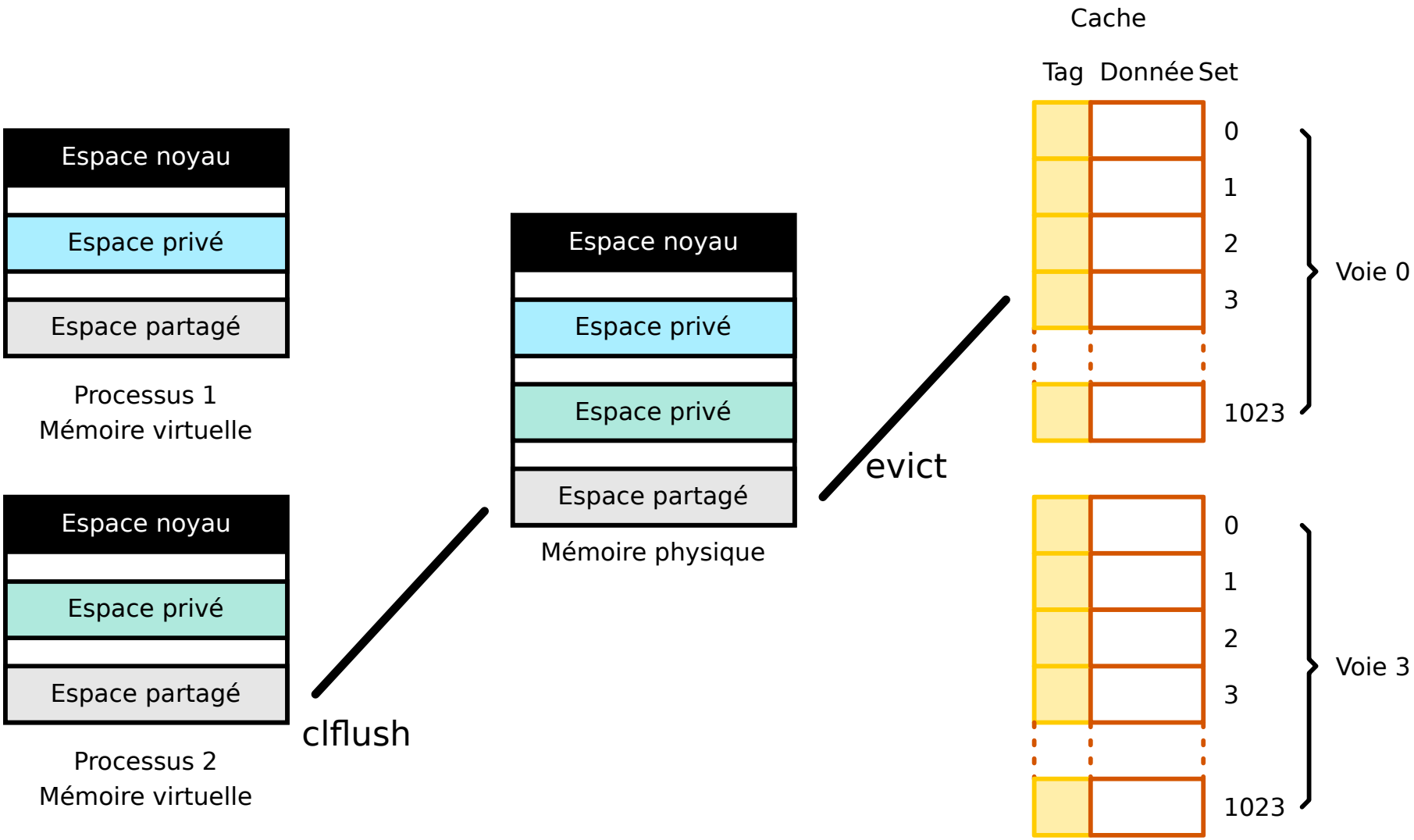
Mémoire partagée (2)



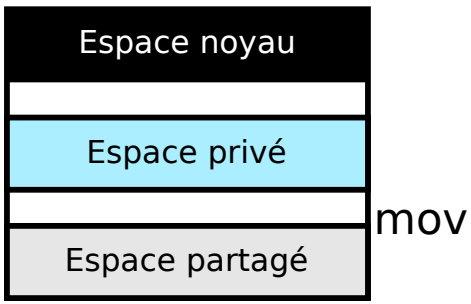
Mémoire partagée (2)



Mémoire partagée (2)

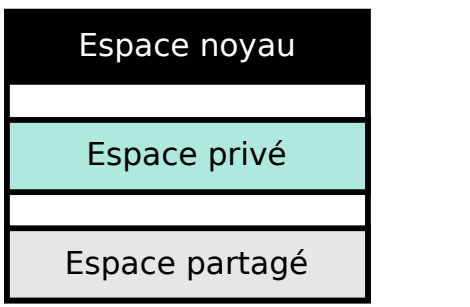


Mémoire partagée (2)

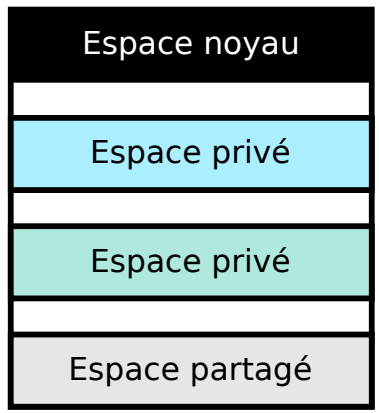


mov

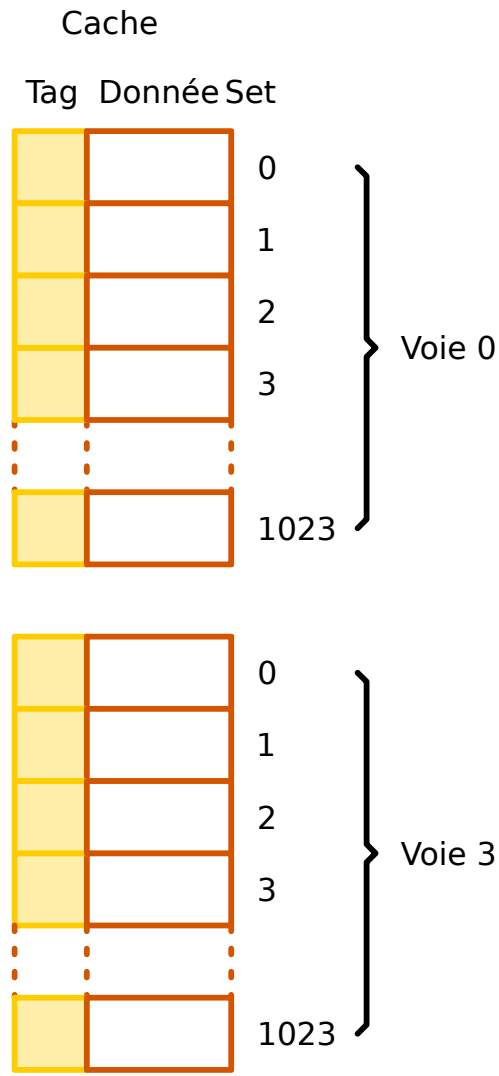
Processus 1
Mémoire virtuelle



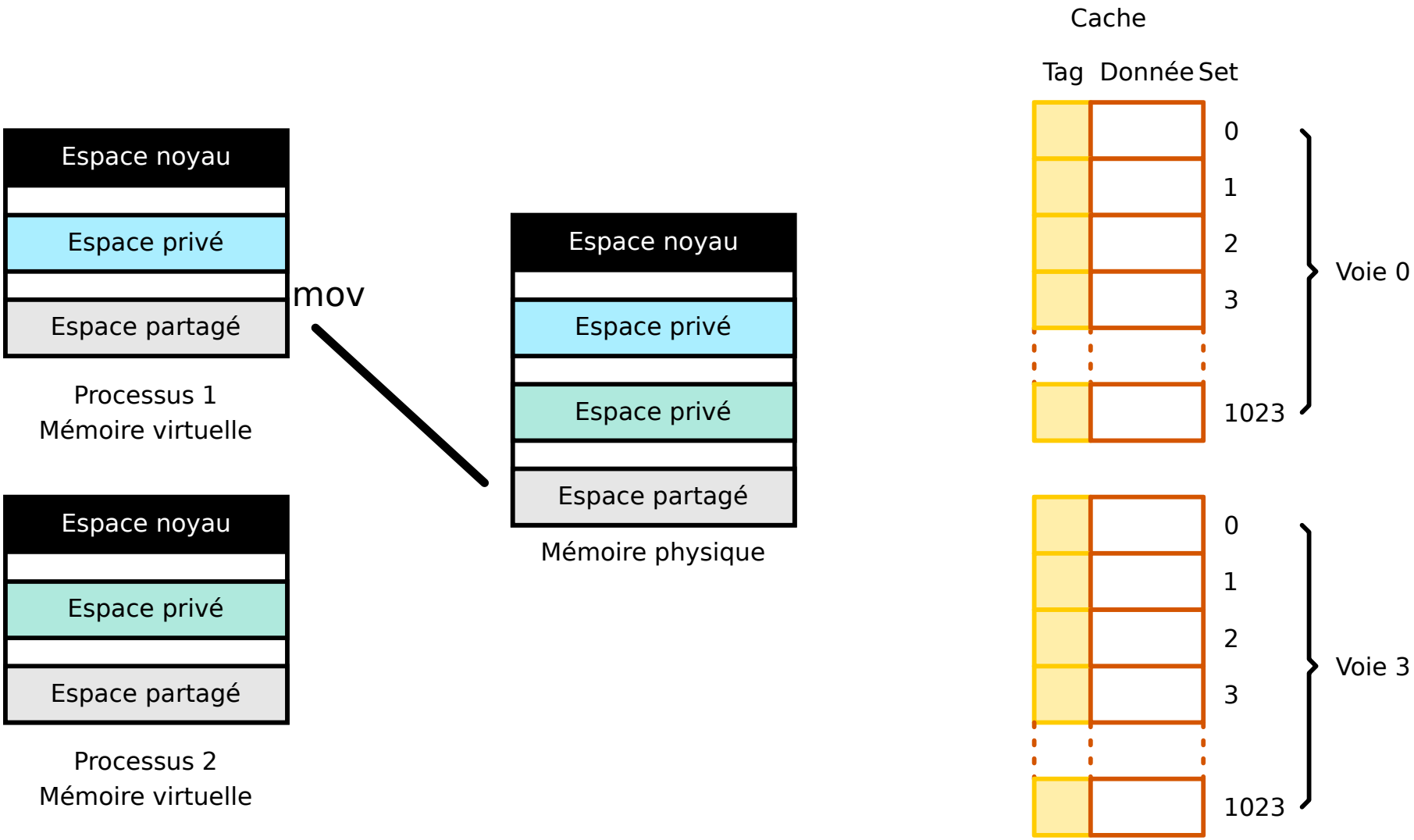
Processus 2
Mémoire virtuelle



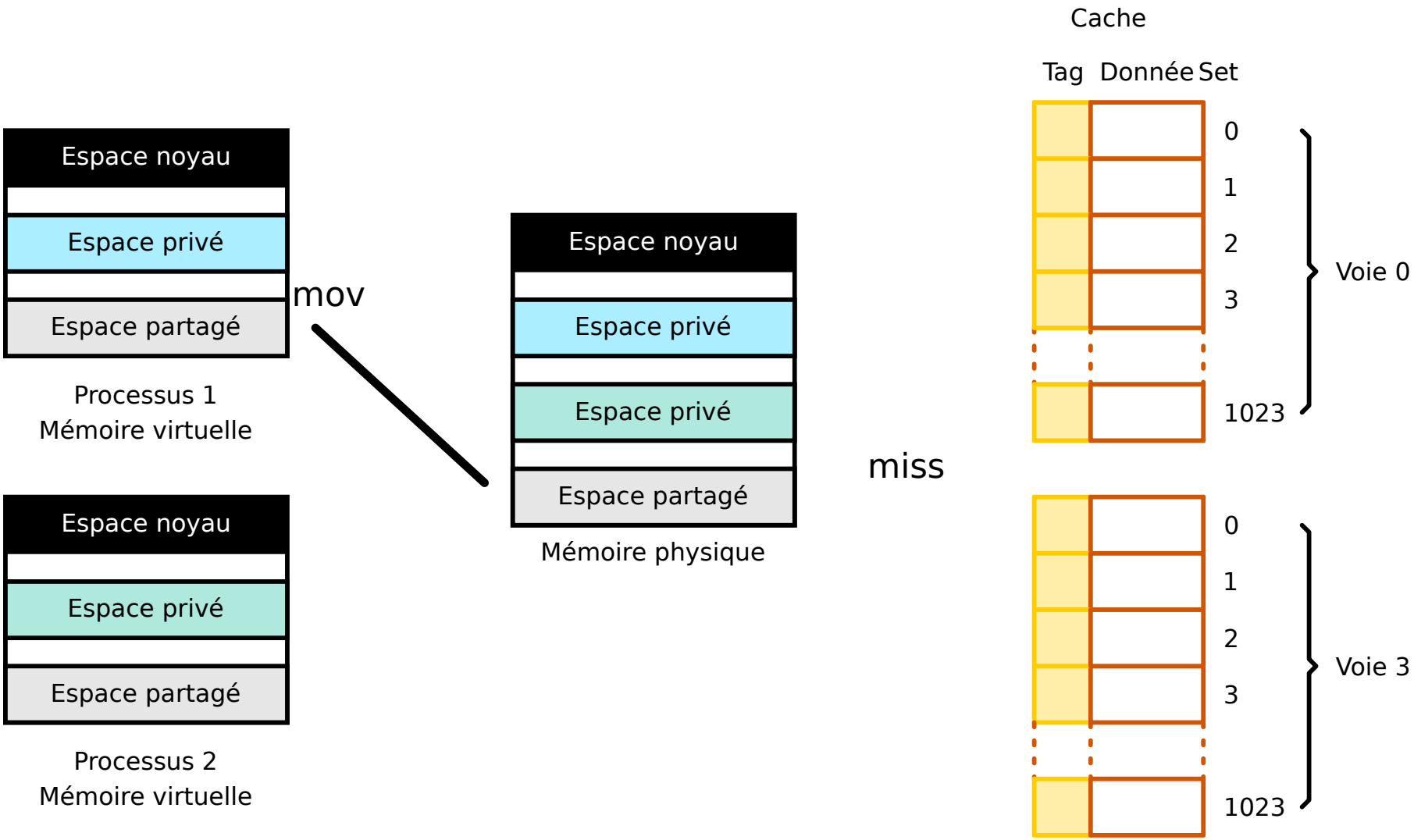
Mémoire physique



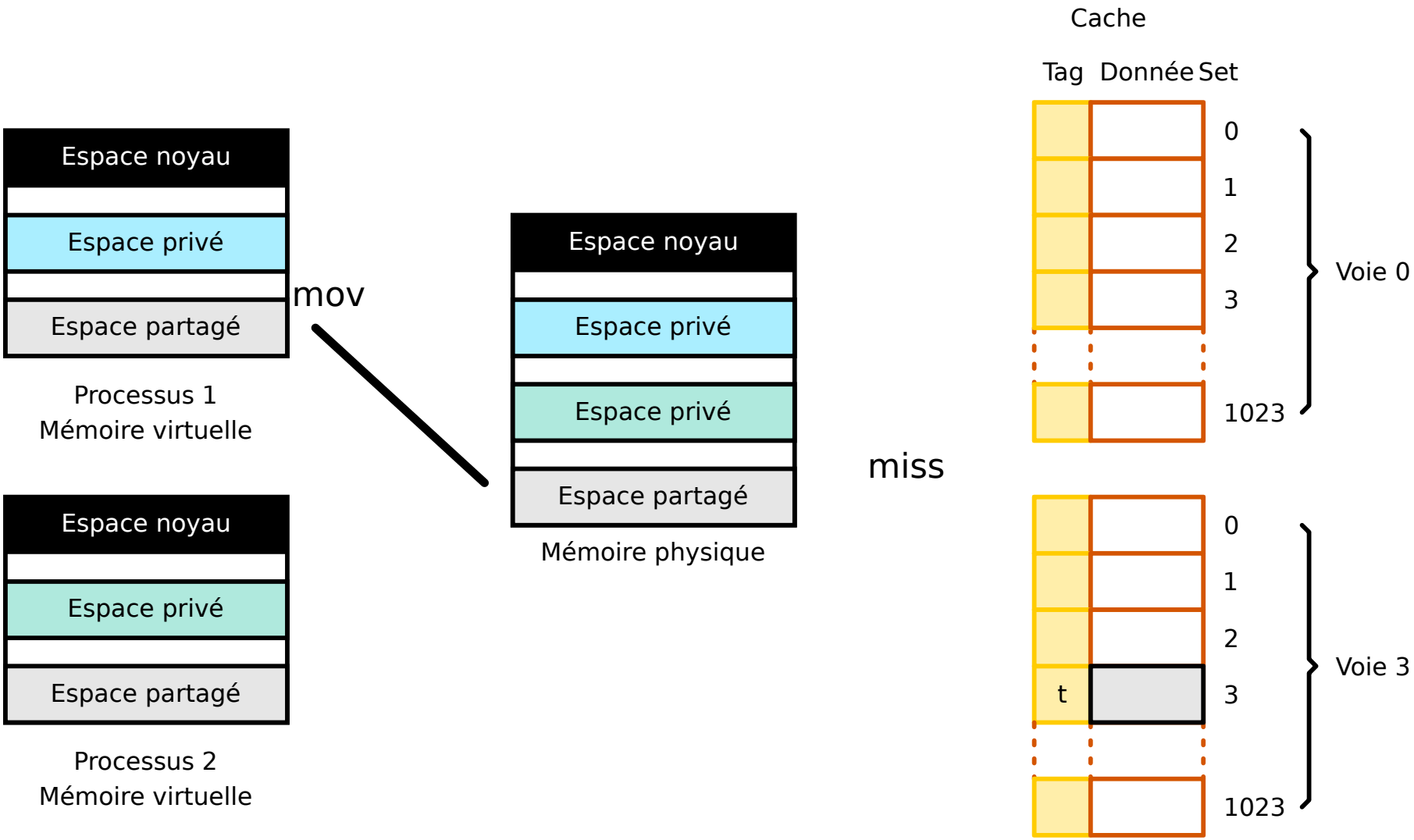
Mémoire partagée (2)



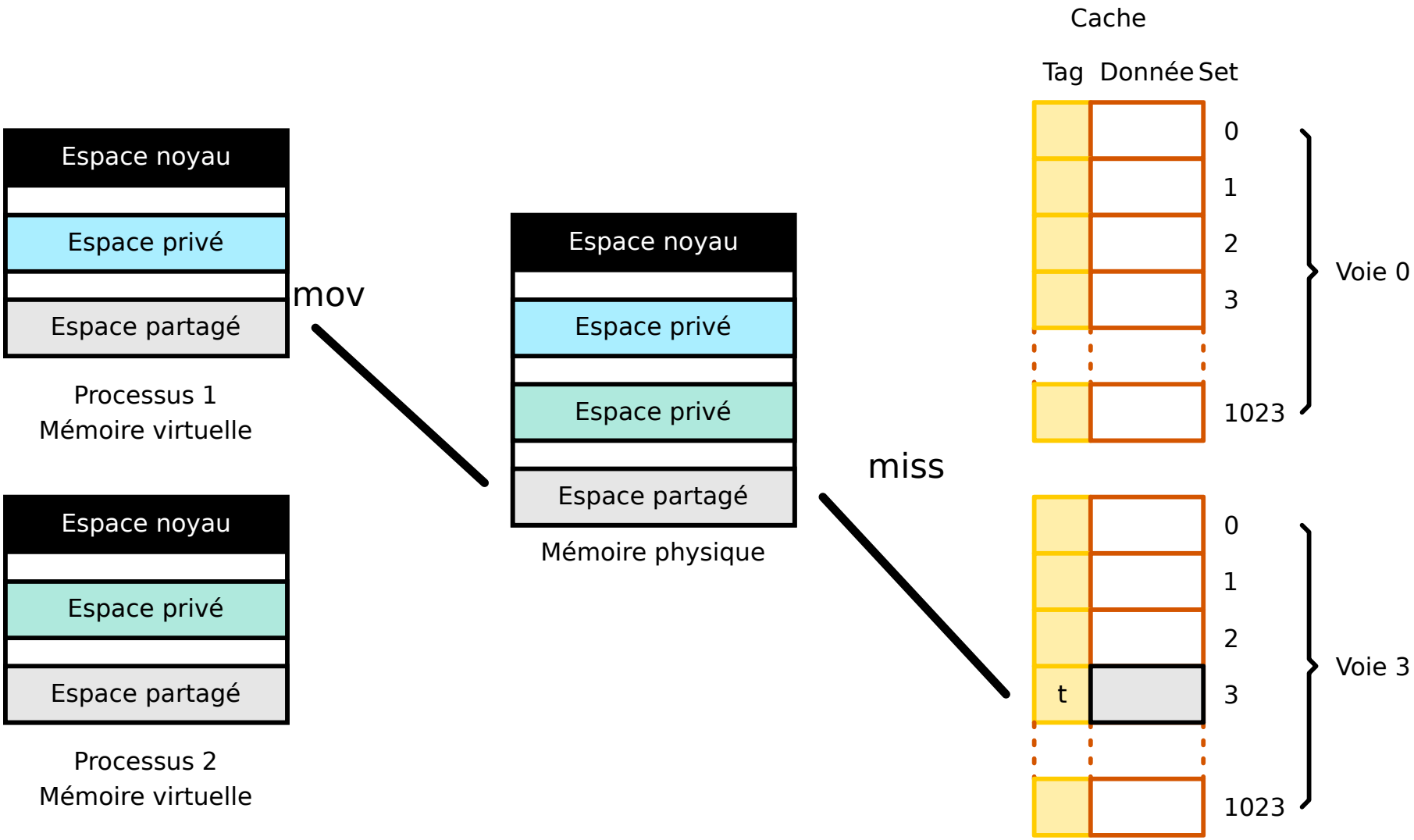
Mémoire partagée (2)



Mémoire partagée (2)



Mémoire partagée (2)



Cache set partagé (1)

Définition

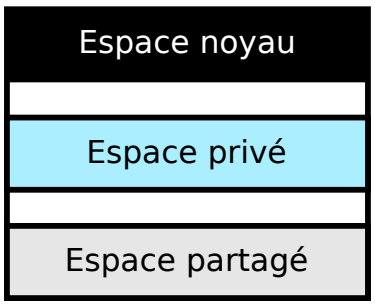
Les deux processus ne pas partagent pas une adresse virtuelle qui va nécessairement être traduite en même mémoire physique.

Conséquence 1 : impossible d'utiliser `clflush`.
En outre, il partagent très probablement le même cache set pour plusieurs lignes de cache.

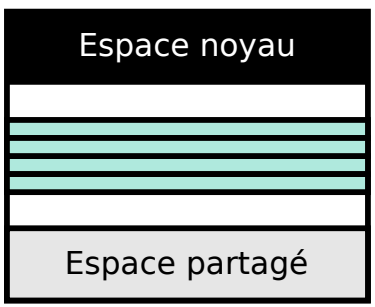
Remarque : lire n *ways* de lignes de cache différentes vident nécessairement un cache set de toutes les autres lignes.

Conséquence 2 : c'est à peu près équivalent à un `clflush` d'une ligne de cache non partagée.

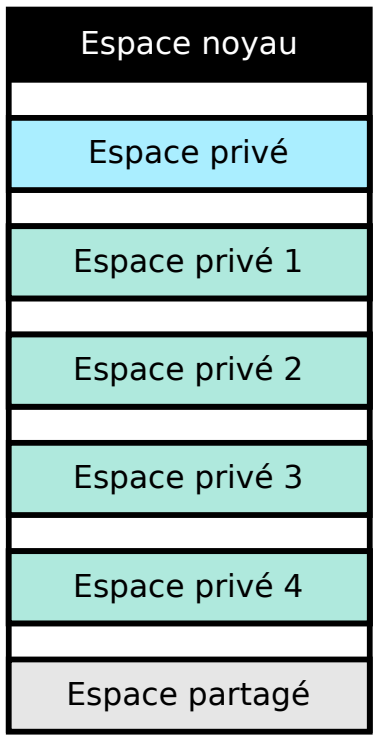
Cache set partagé (2)



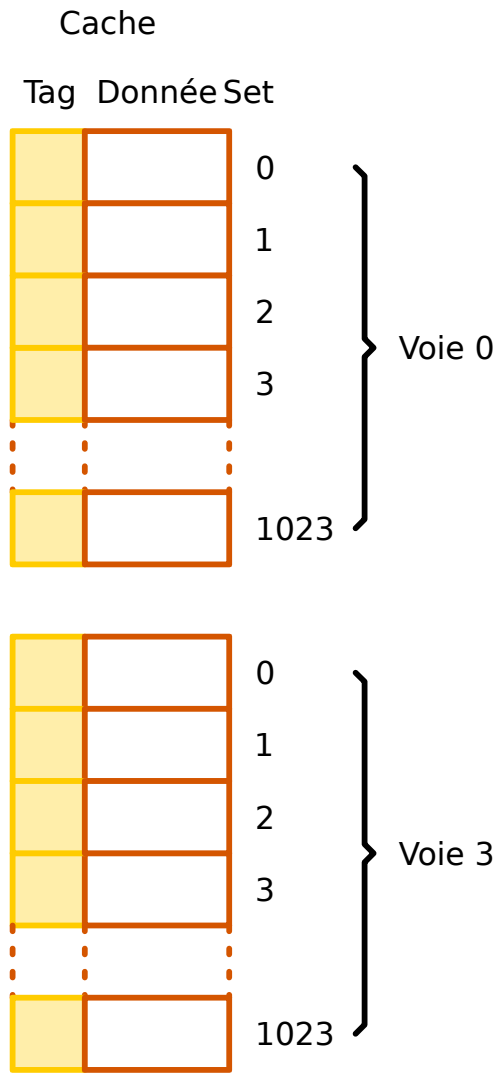
Processus 1
Mémoire virtuelle



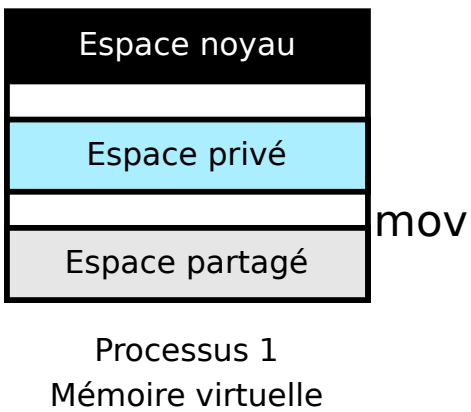
Processus 2
Mémoire virtuelle

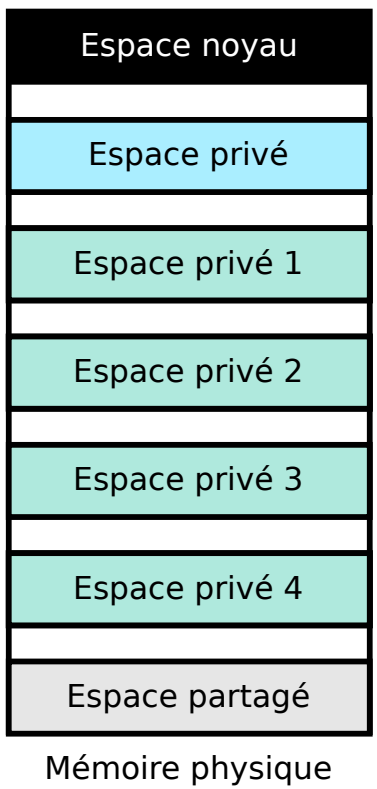


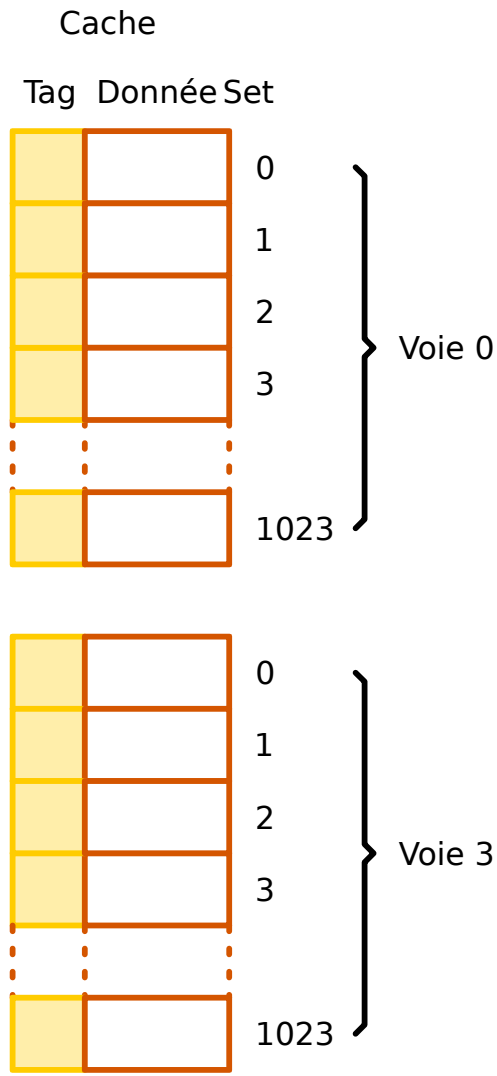
Mémoire physique



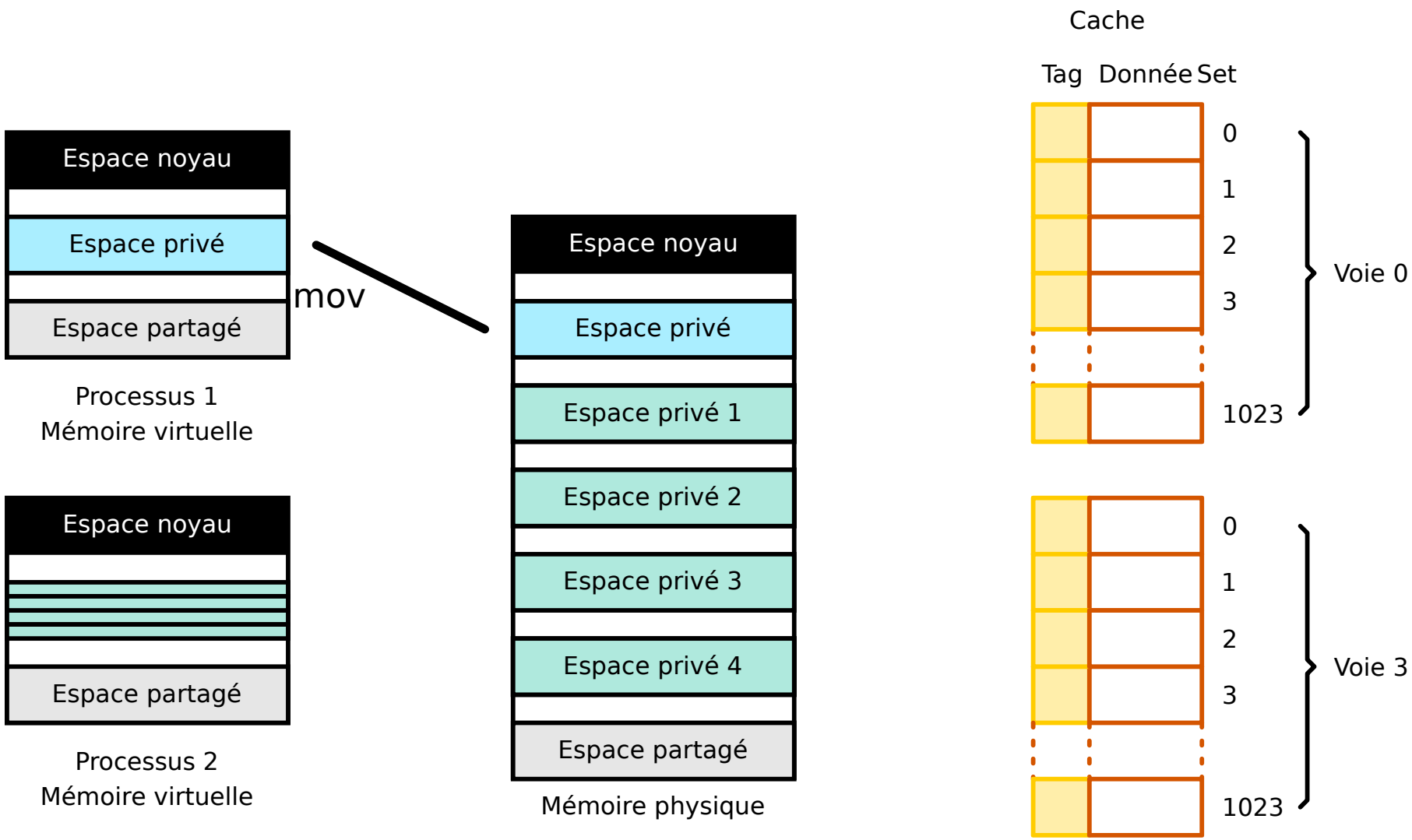
Cache set partagé (2)



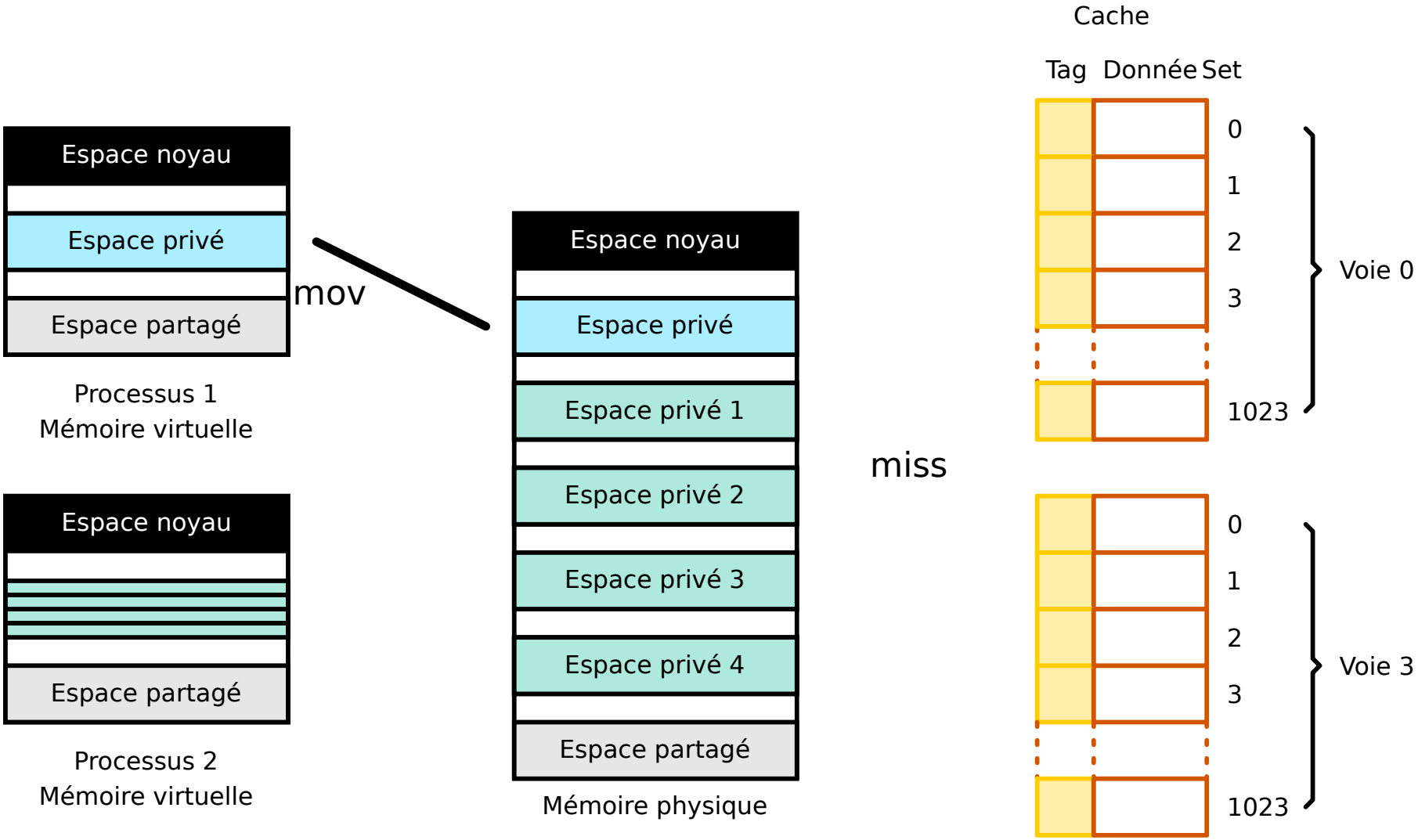




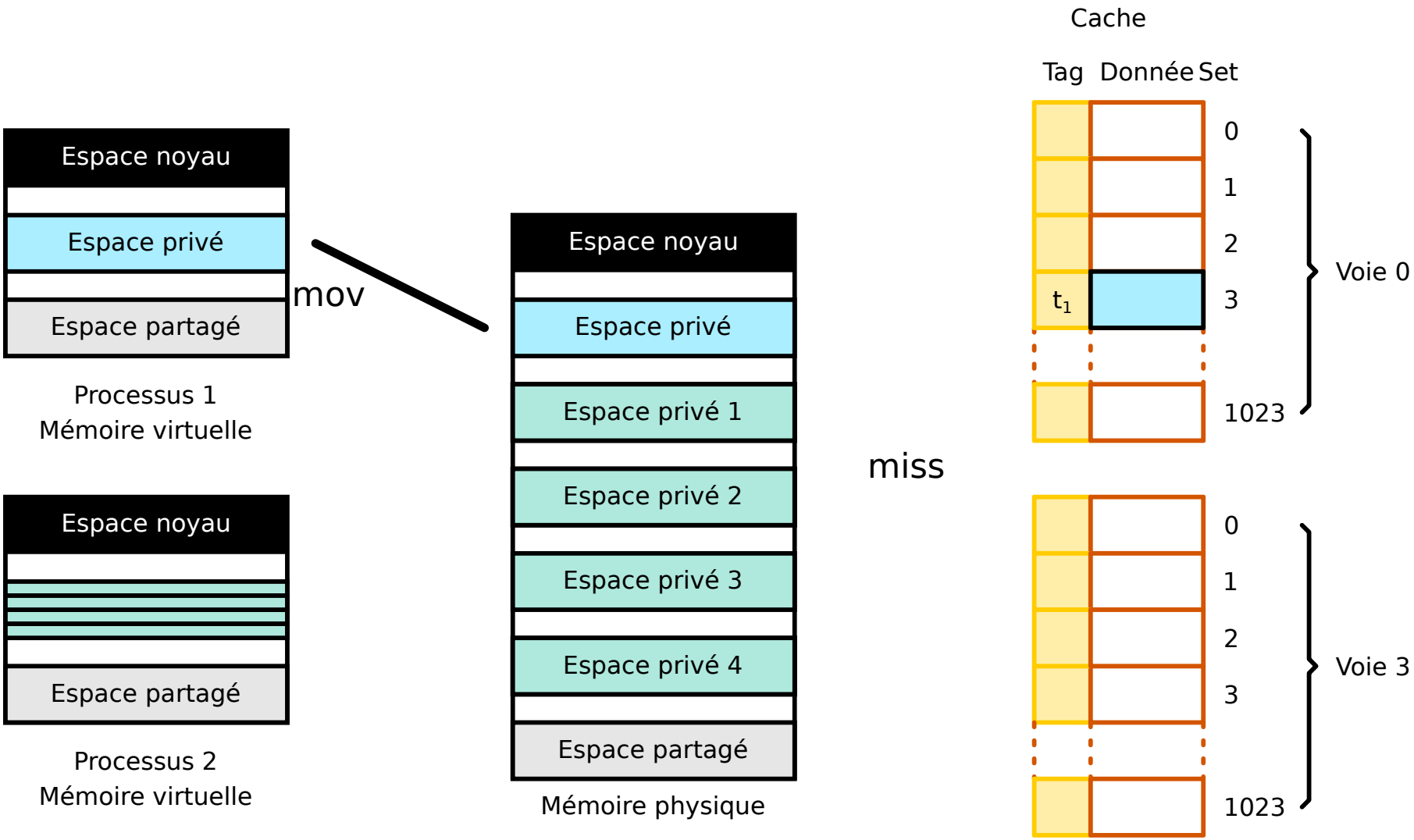
Cache set partagé (2)



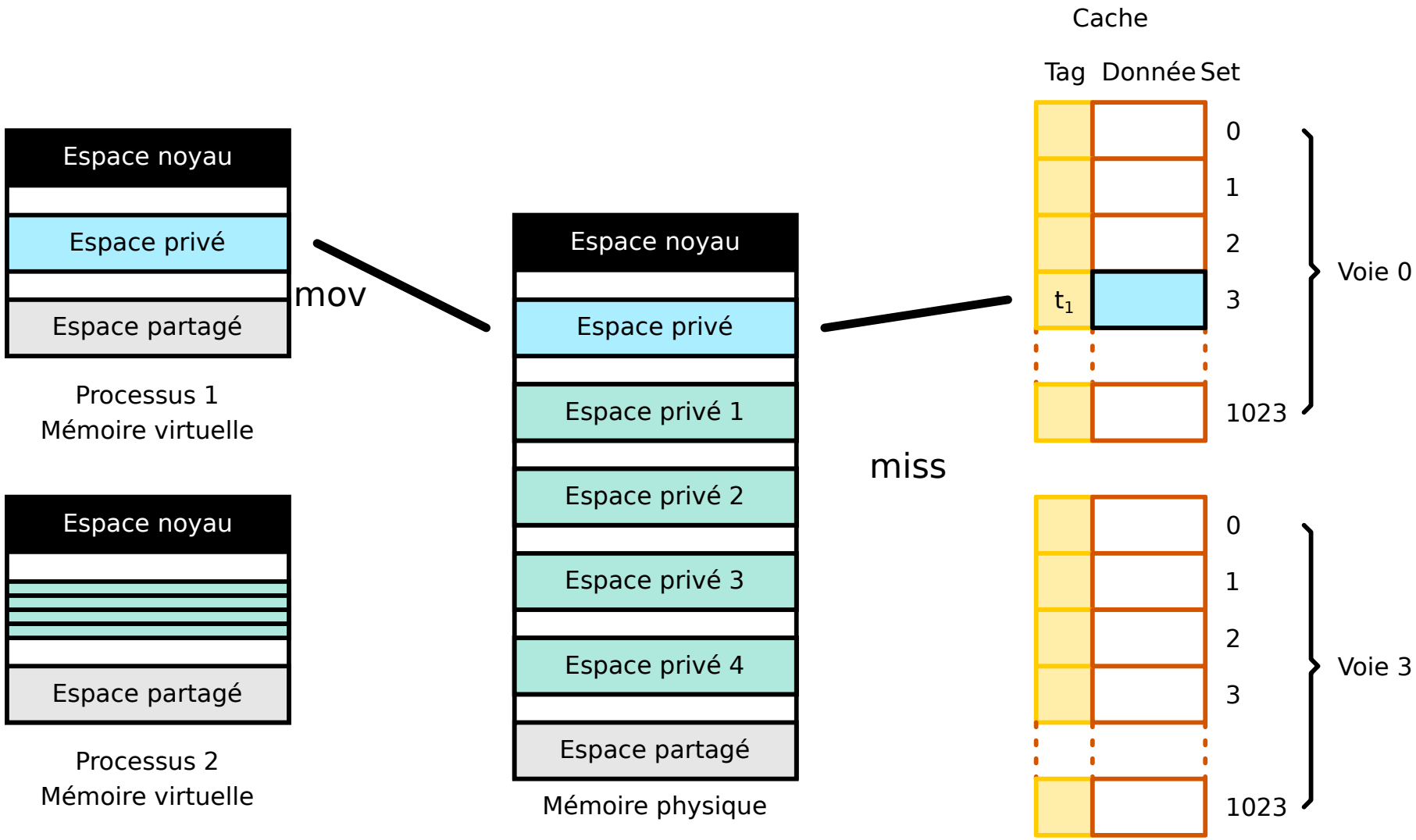
Cache set partagé (2)



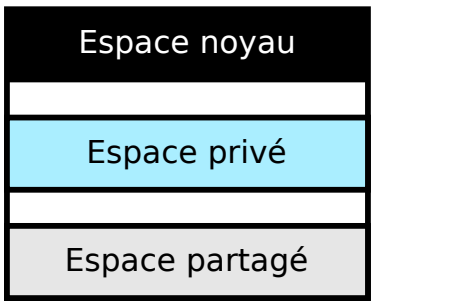
Cache set partagé (2)



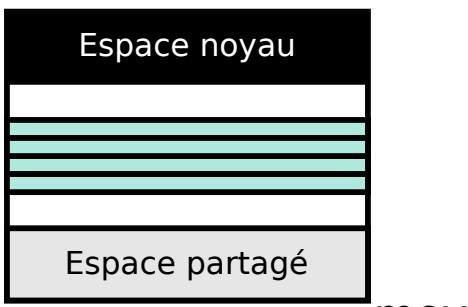
Cache set partagé (2)



Cache set partagé (2)

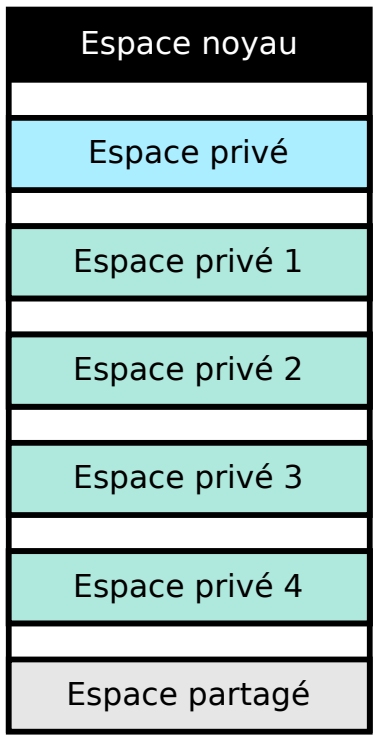


Processus 1
Mémoire virtuelle

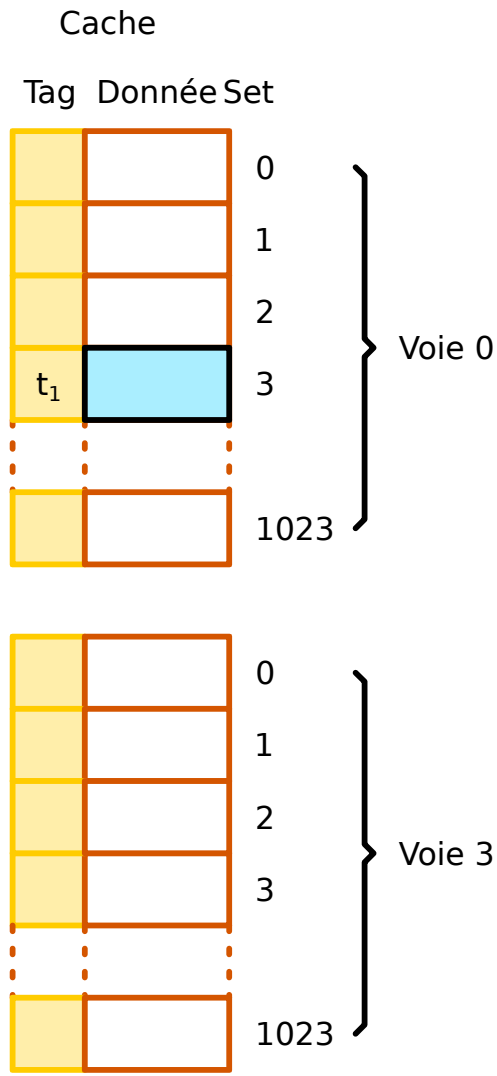


mov

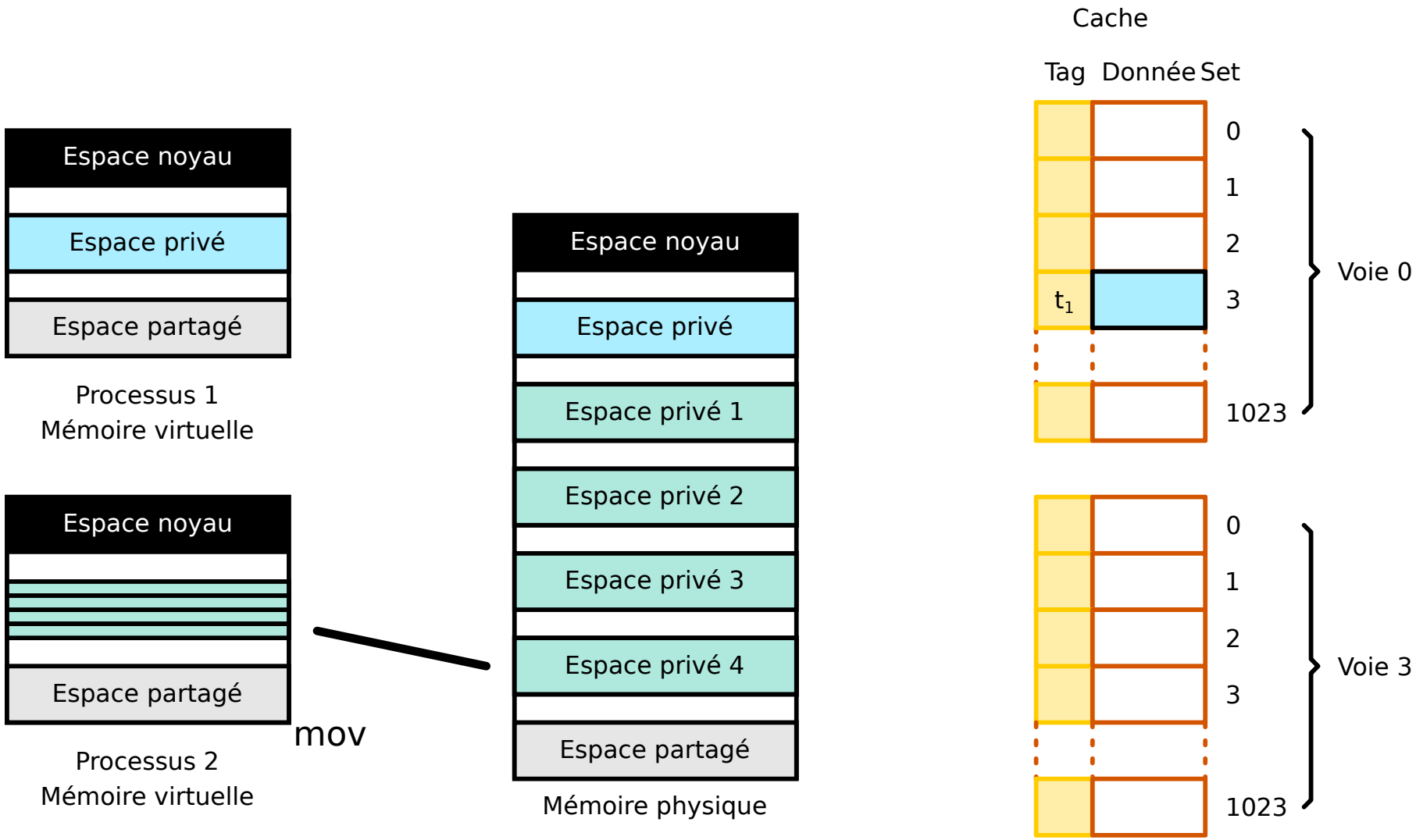
Processus 2
Mémoire virtuelle



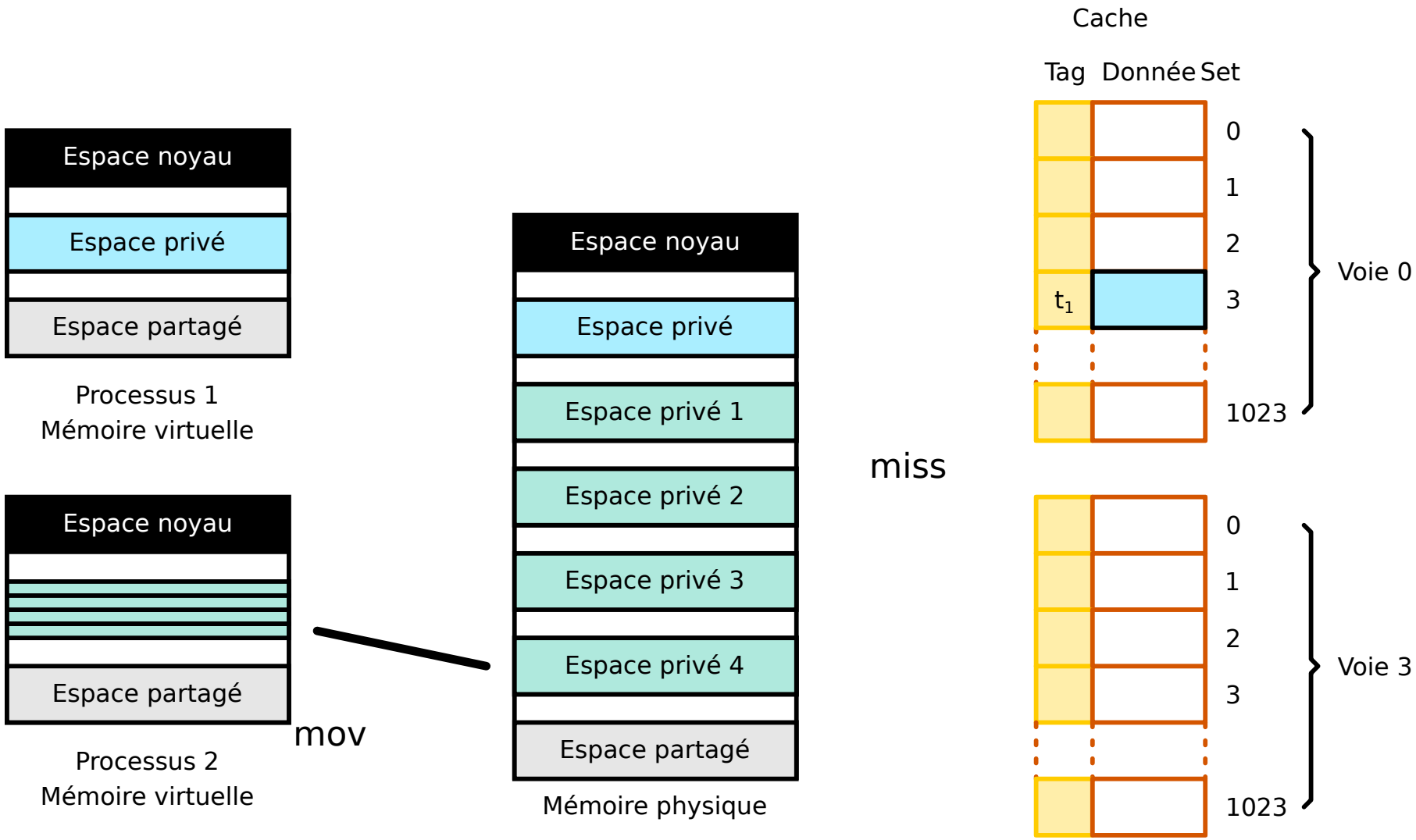
Mémoire physique



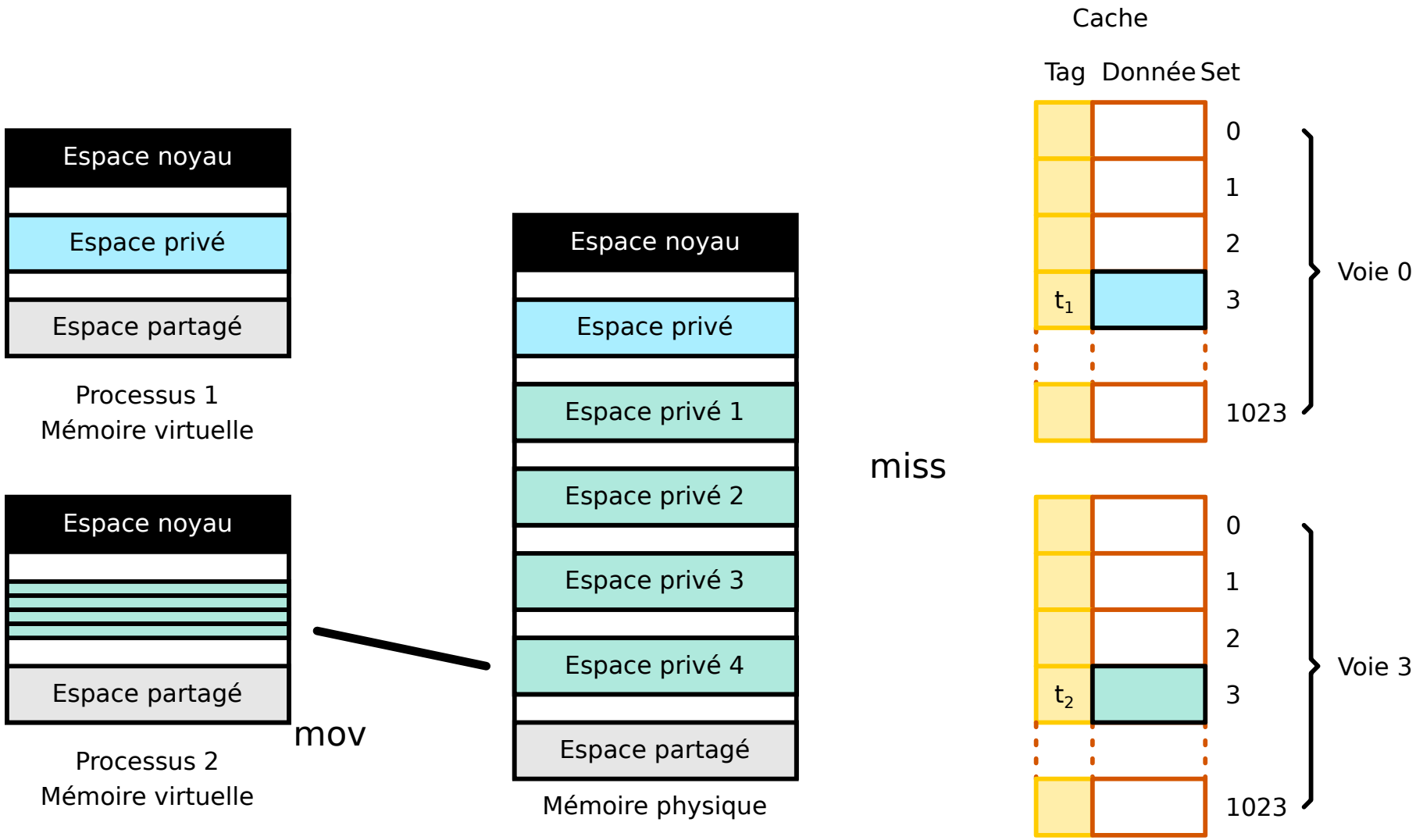
Cache set partagé (2)



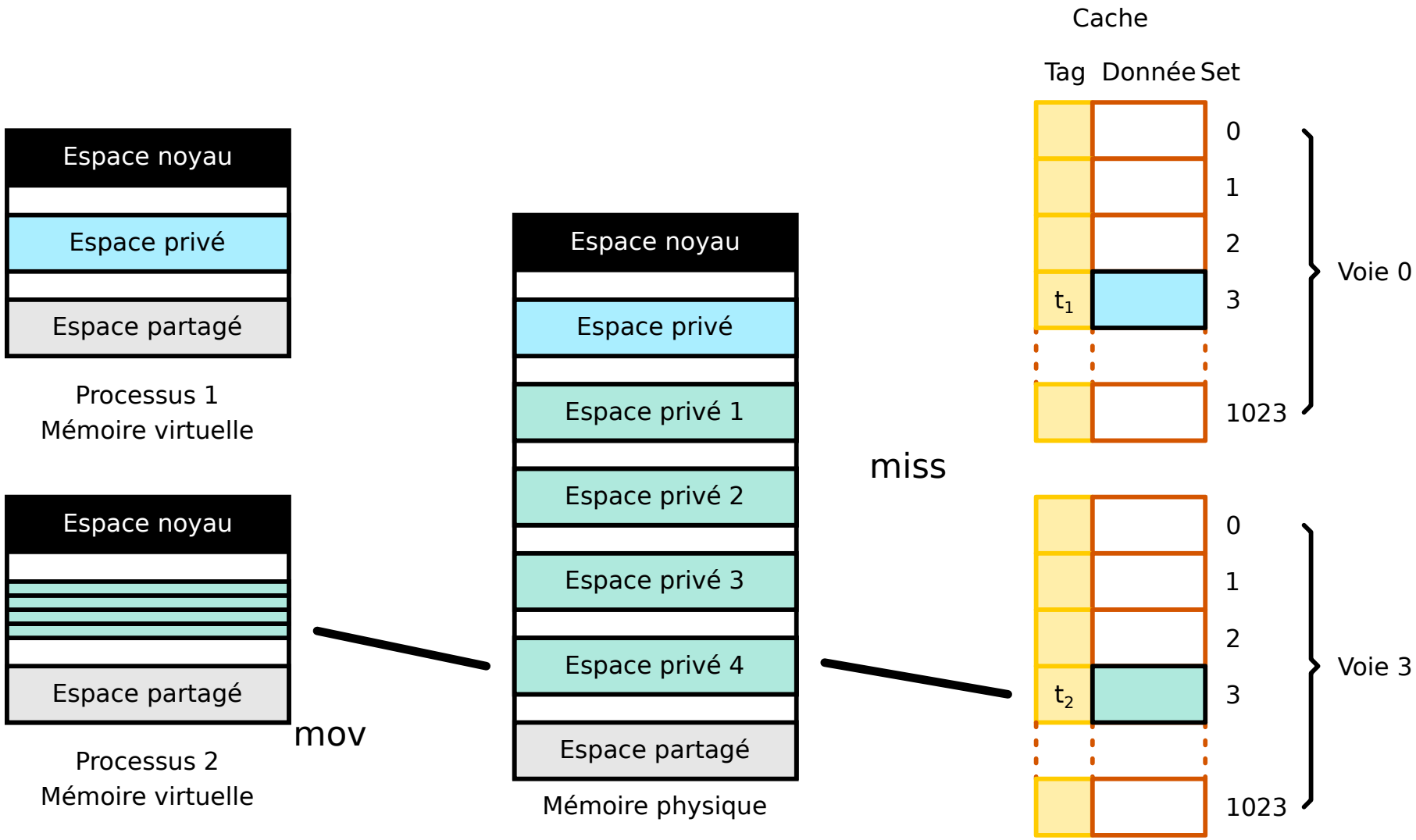
Cache set partagé (2)



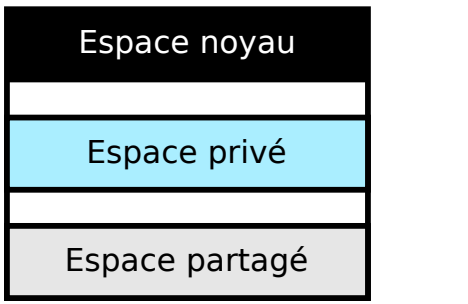
Cache set partagé (2)



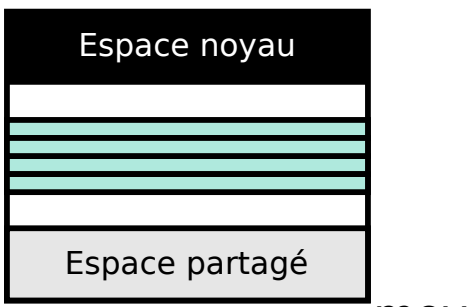
Cache set partagé (2)



Cache set partagé (2)

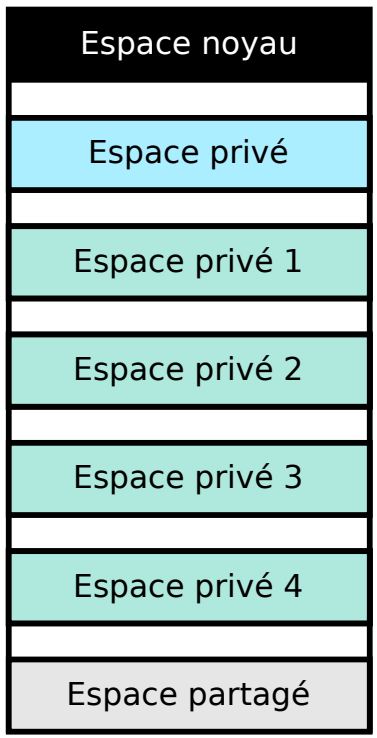


Processus 1
Mémoire virtuelle

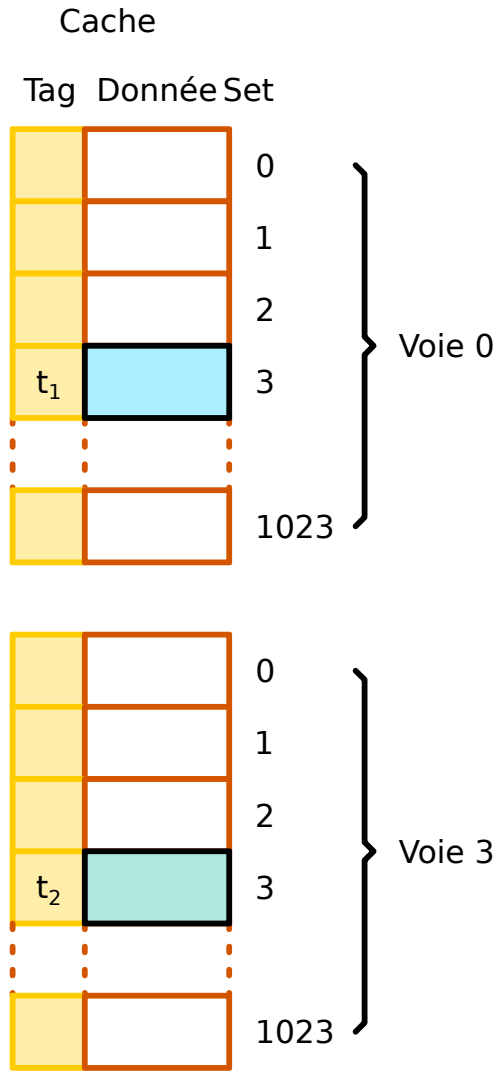


mov

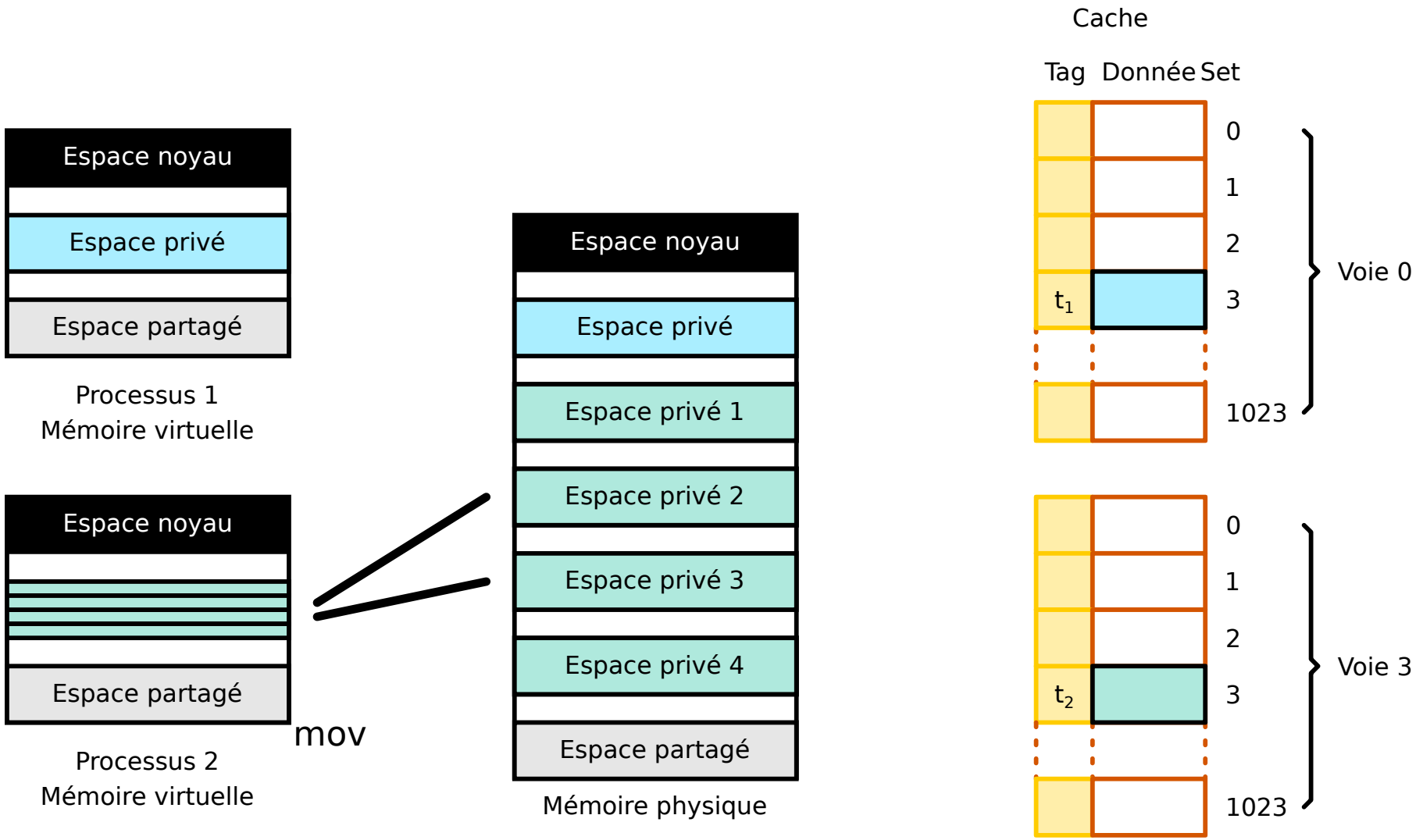
Processus 2
Mémoire virtuelle



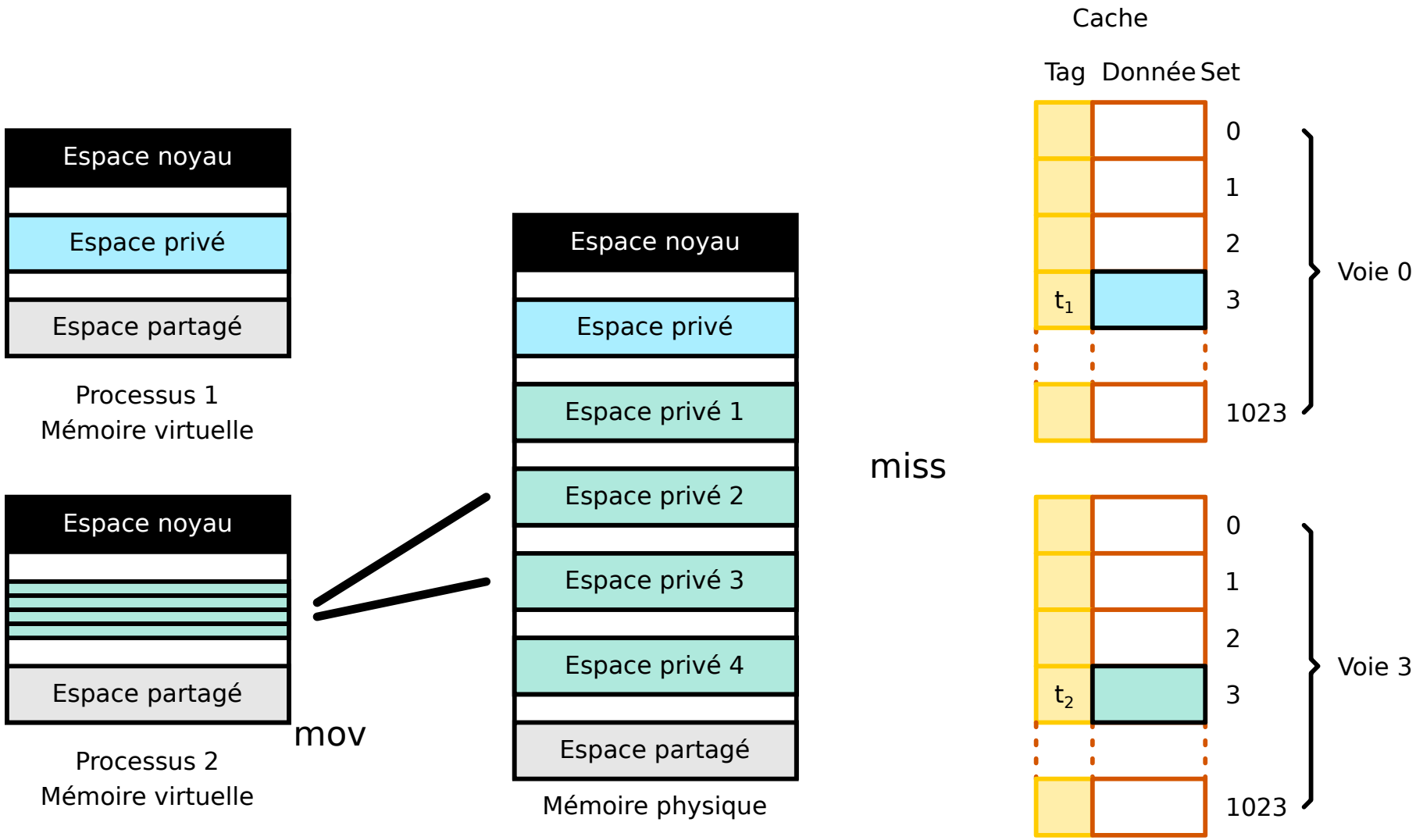
Mémoire physique



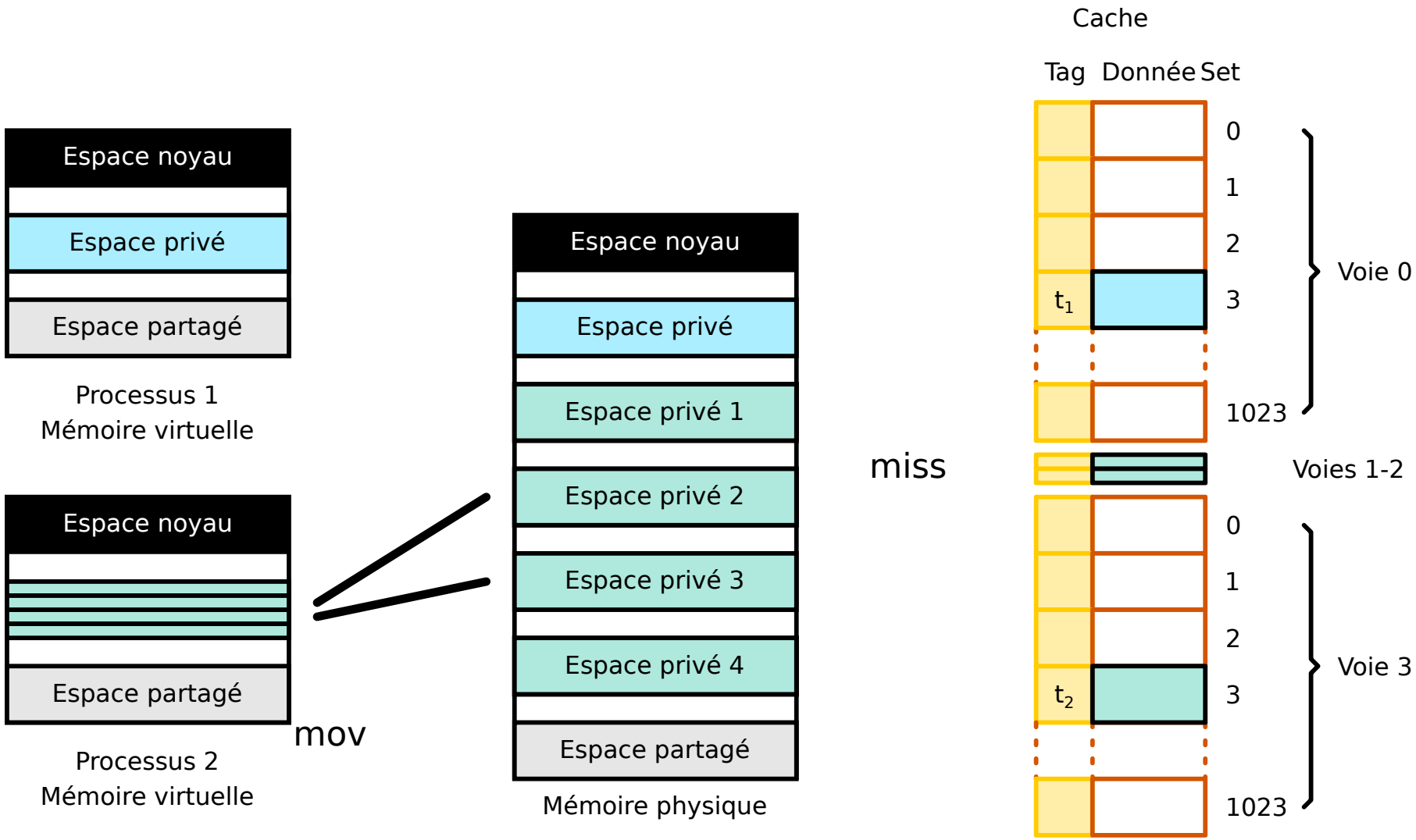
Cache set partagé (2)



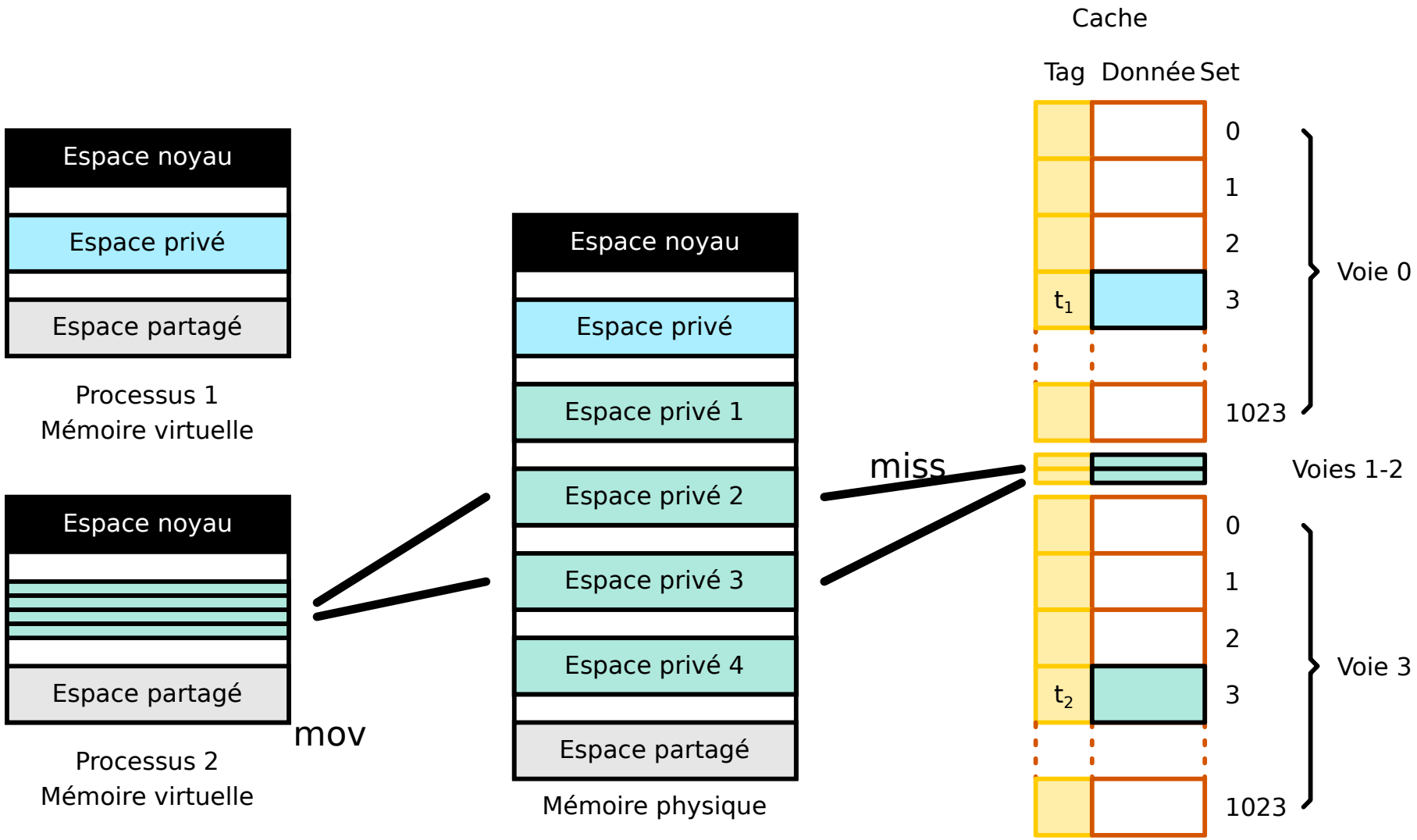
Cache set partagé (2)



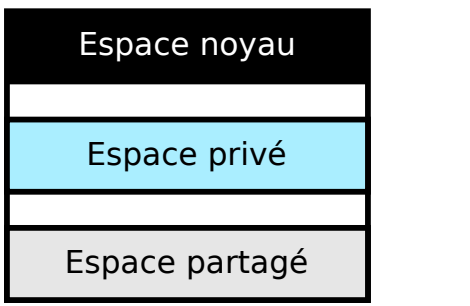
Cache set partagé (2)



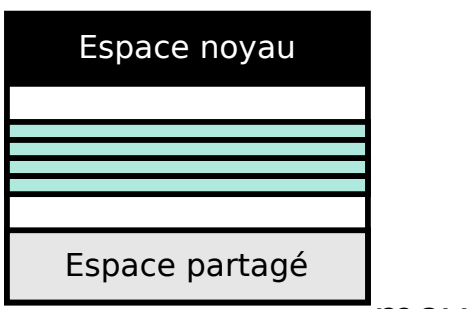
Cache set partagé (2)



Cache set partagé (2)

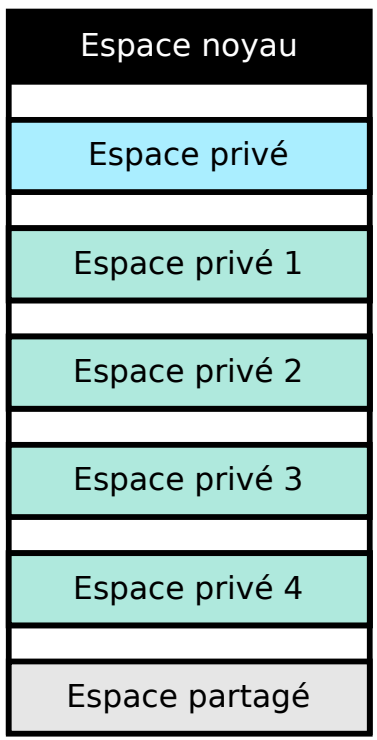


Processus 1
Mémoire virtuelle

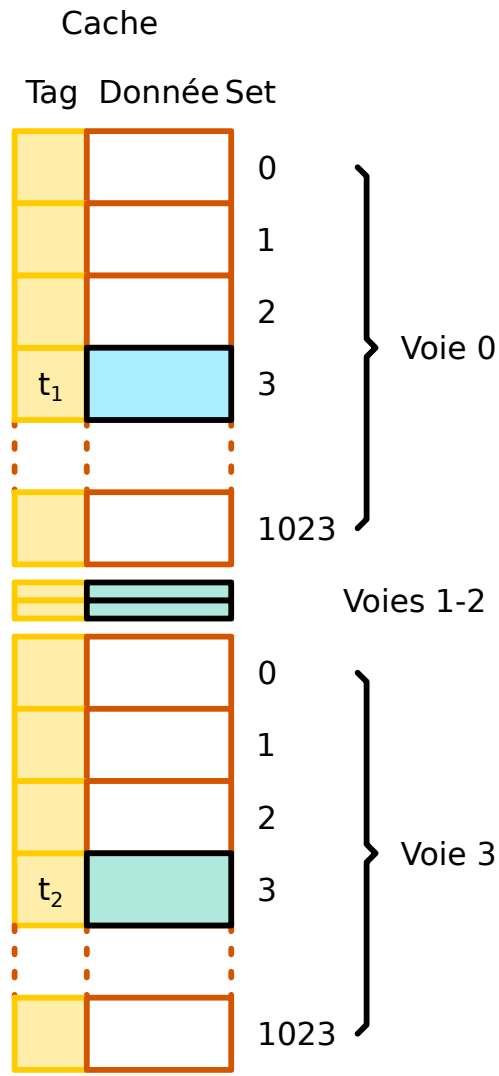


Processus 2
Mémoire virtuelle

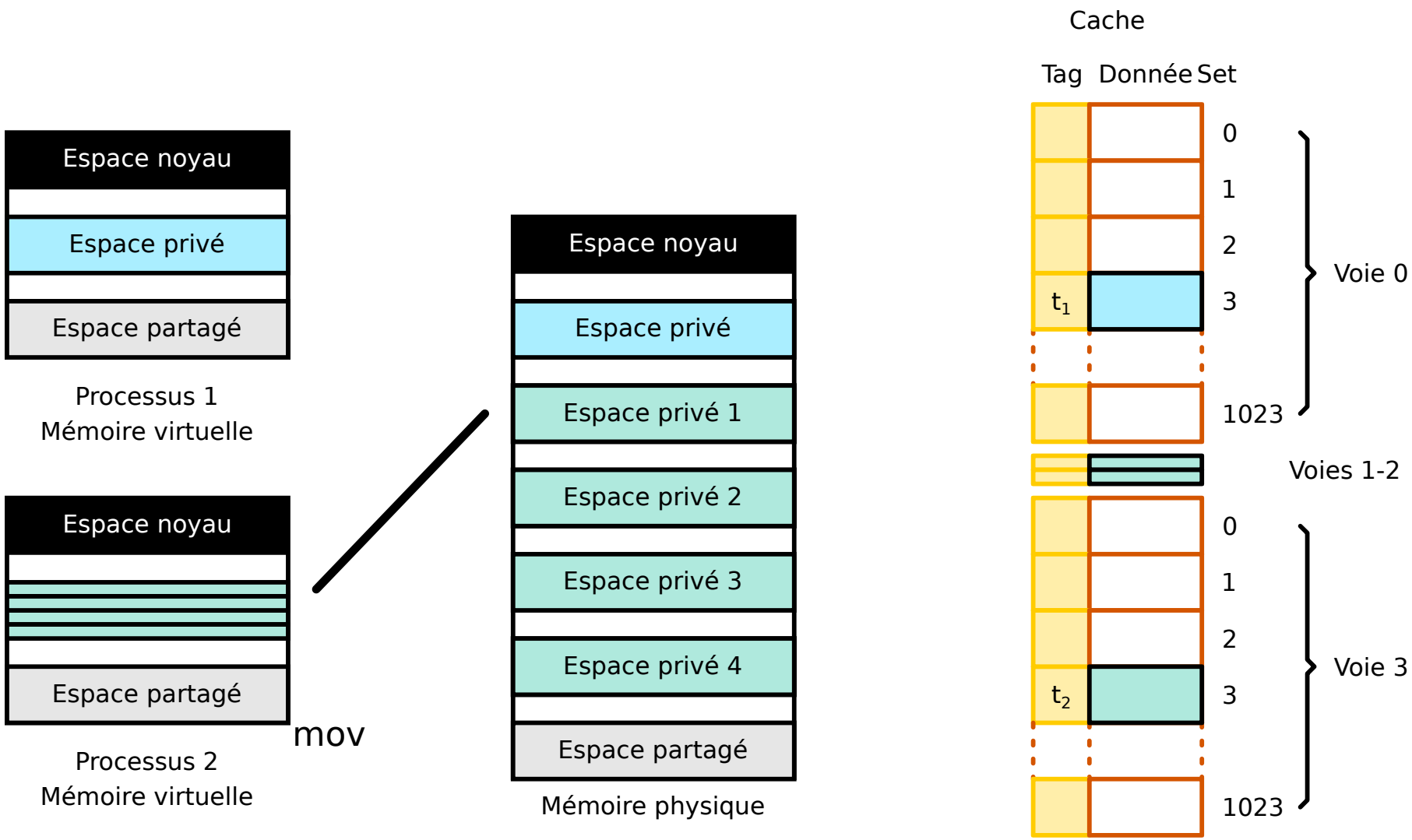
mov



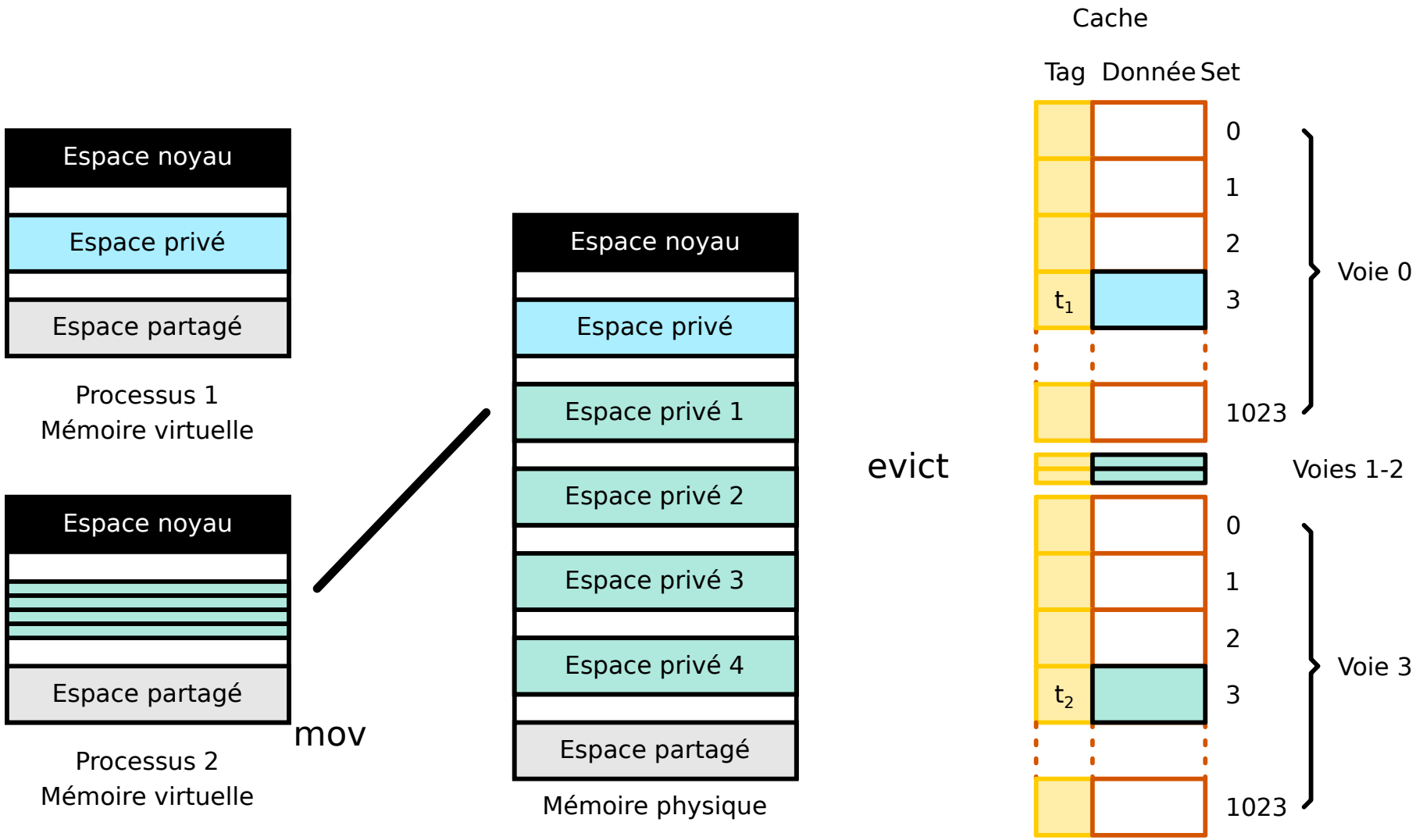
Mémoire physique



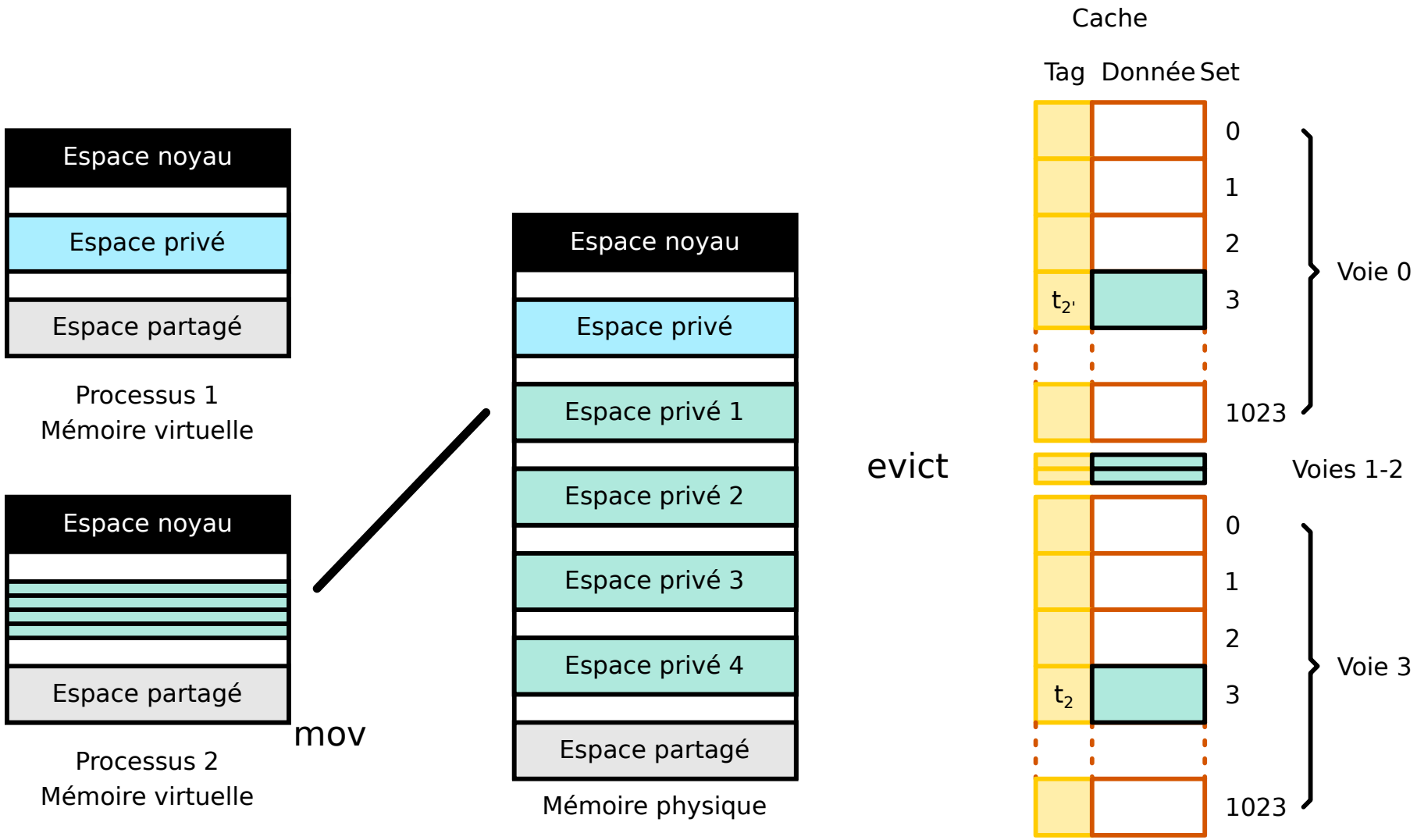
Cache set partagé (2)



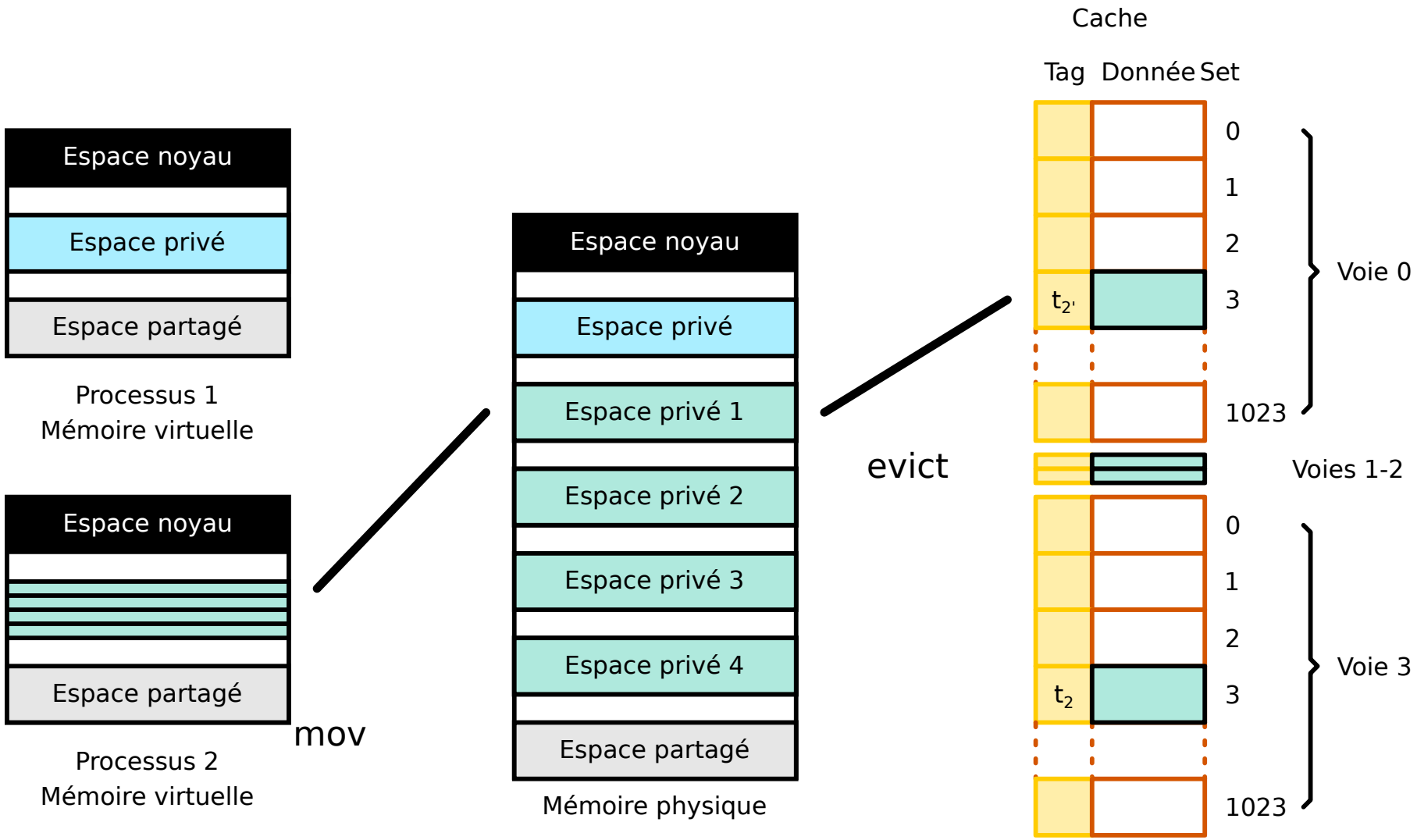
Cache set partagé (2)



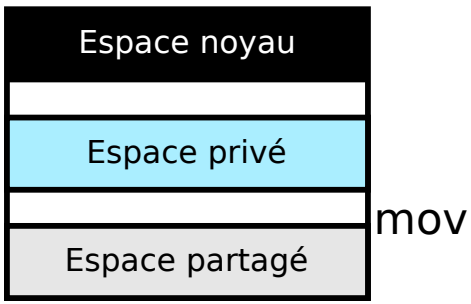
Cache set partagé (2)



Cache set partagé (2)

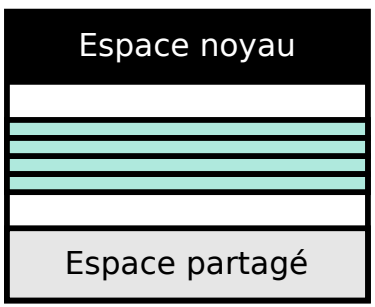


Cache set partagé (2)

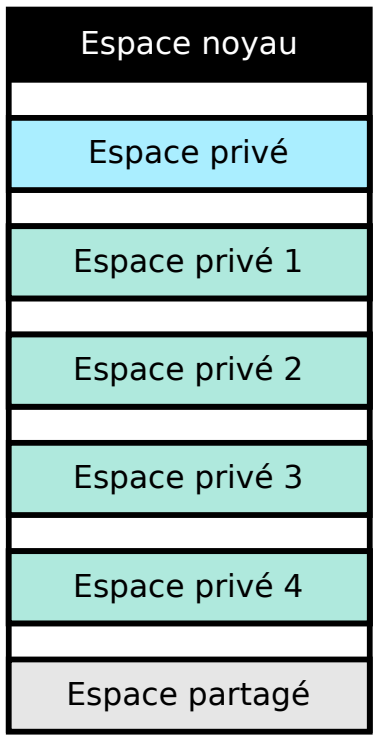


mov

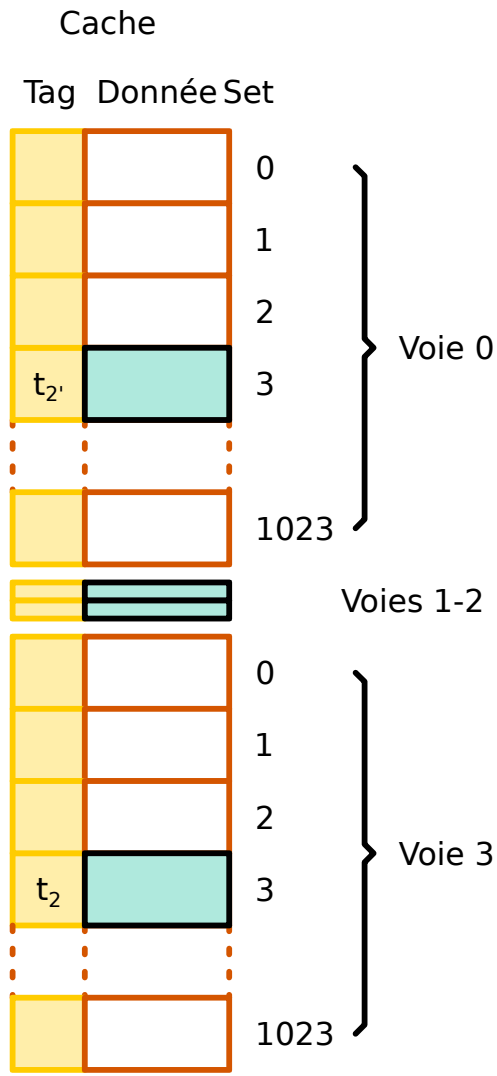
Processus 1
Mémoire virtuelle



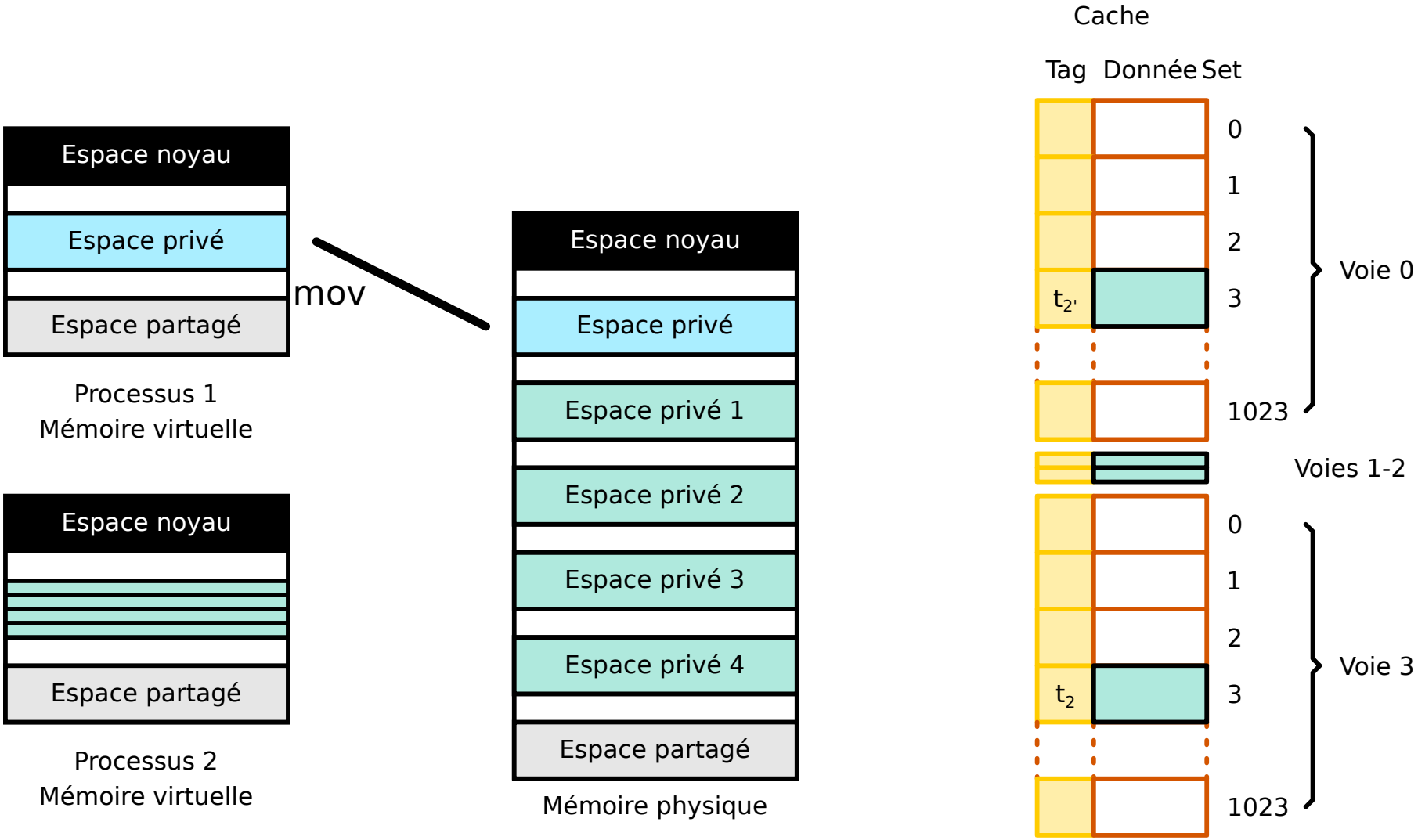
Processus 2
Mémoire virtuelle



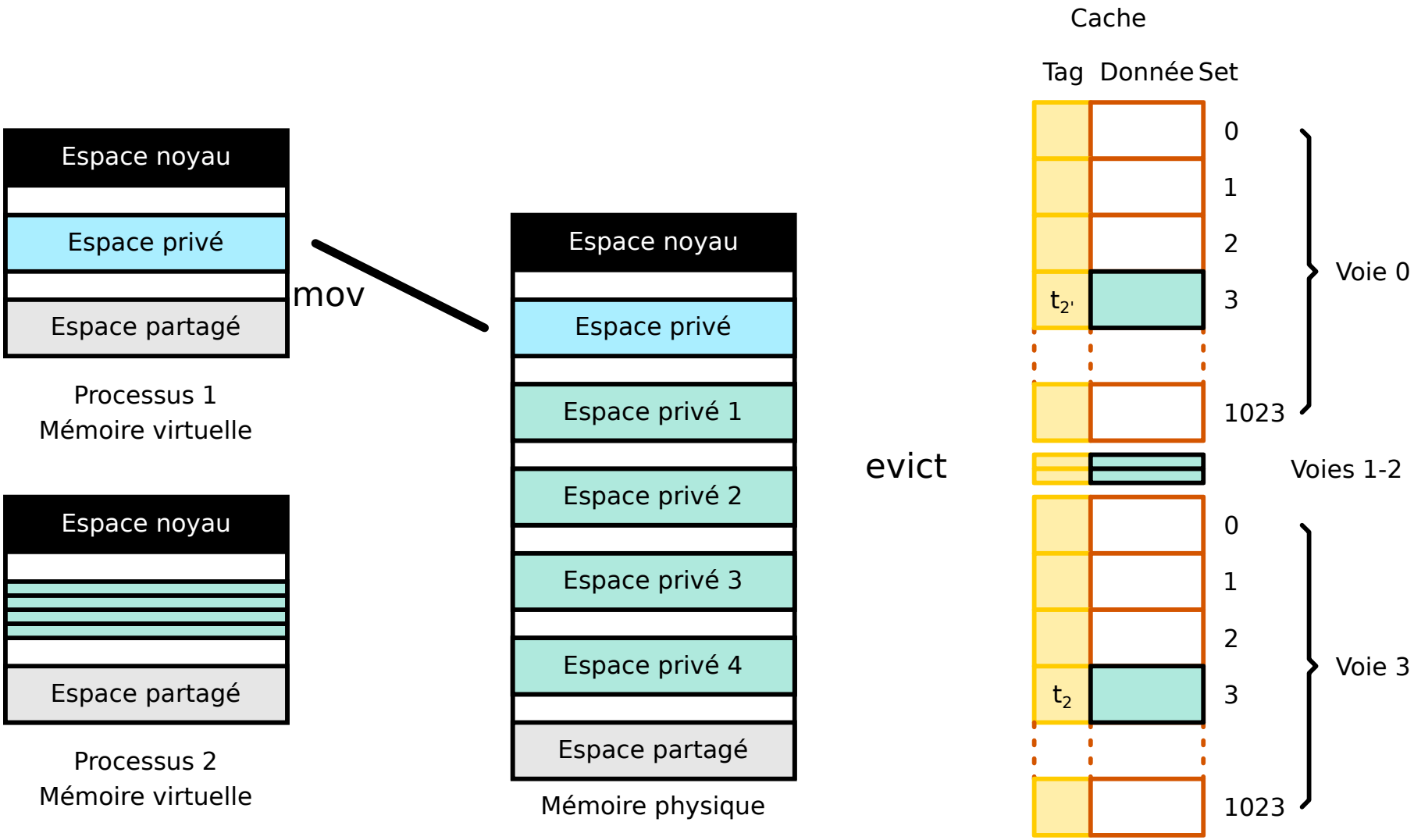
Mémoire physique



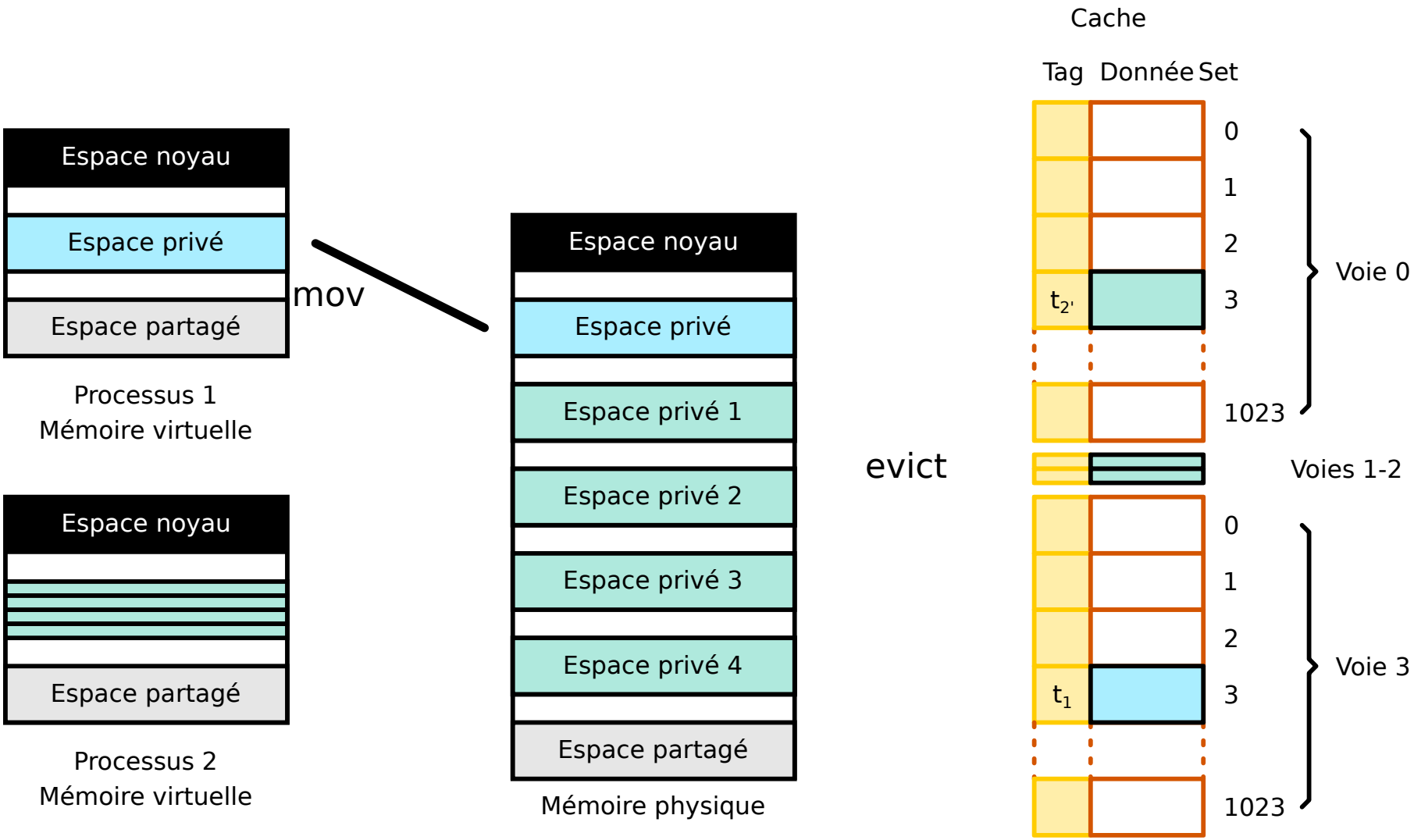
Cache set partagé (2)



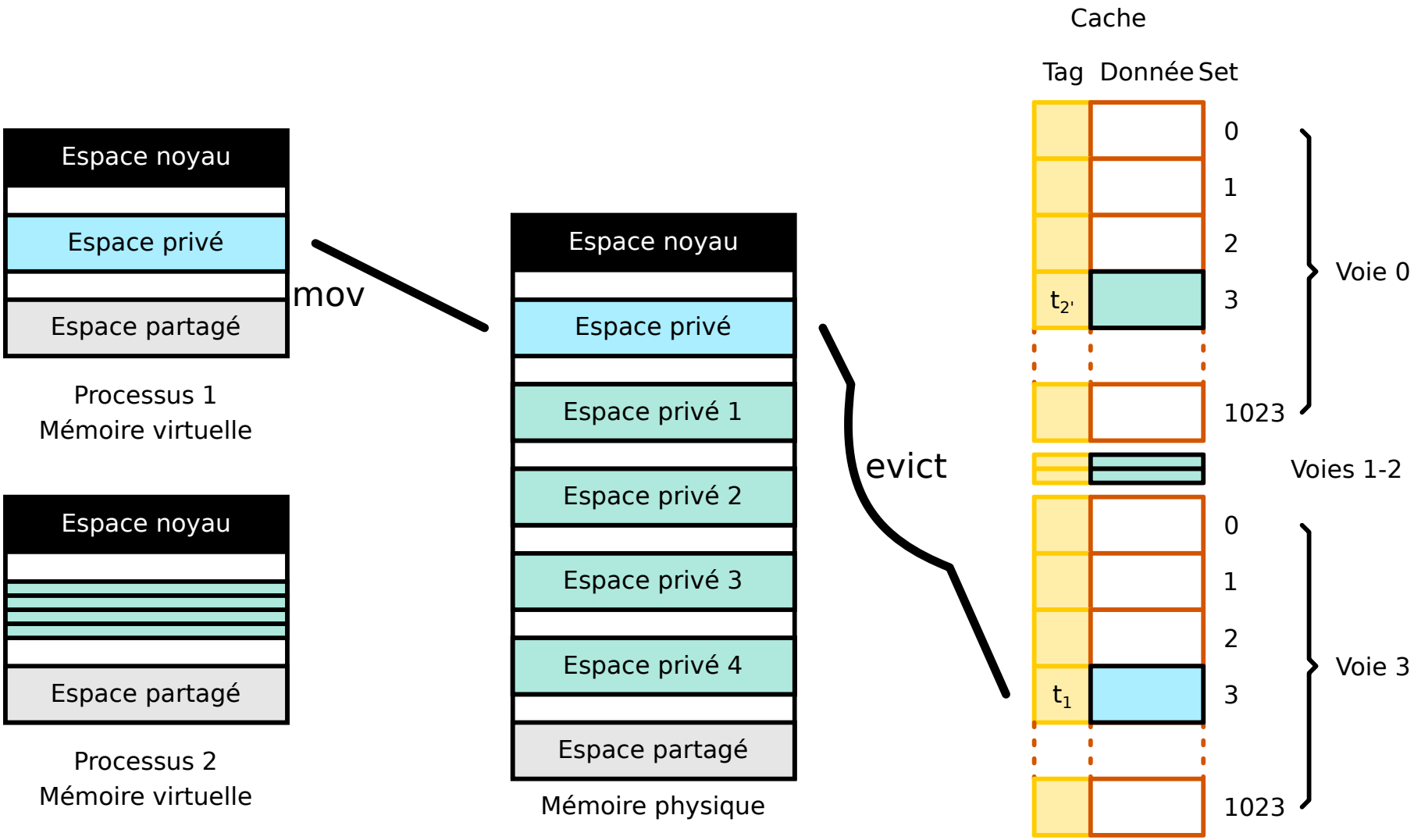
Cache set partagé (2)



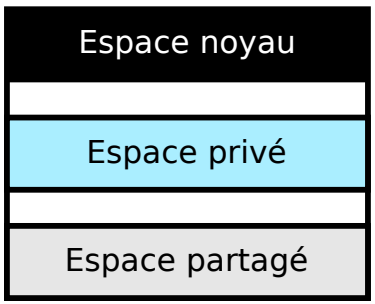
Cache set partagé (2)



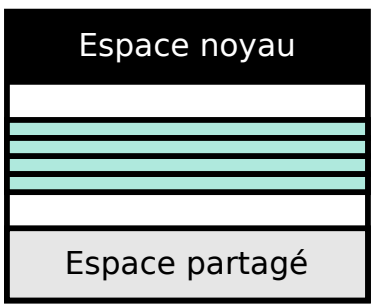
Cache set partagé (2)



Cache set partagé (2)

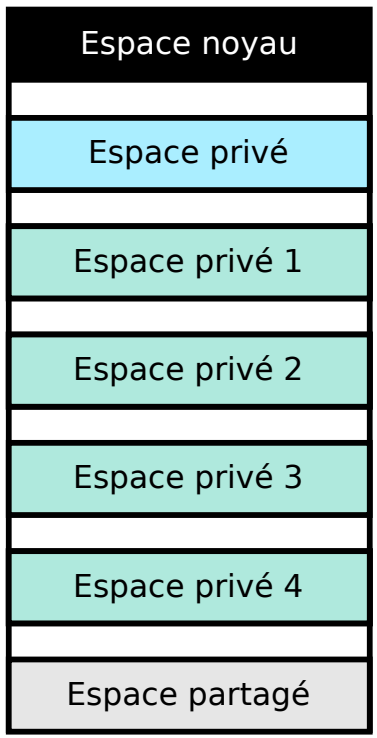


Processus 1
Mémoire virtuelle

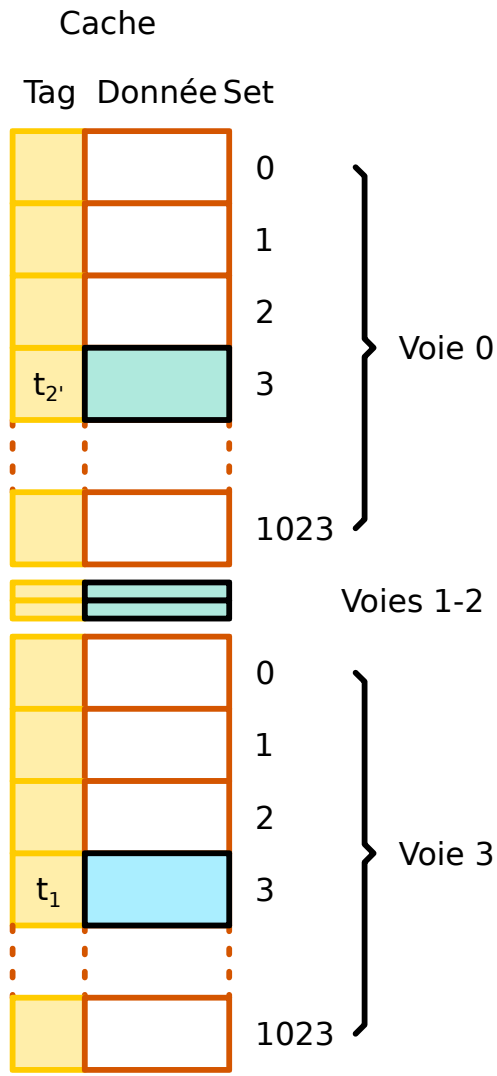


Processus 2
Mémoire virtuelle

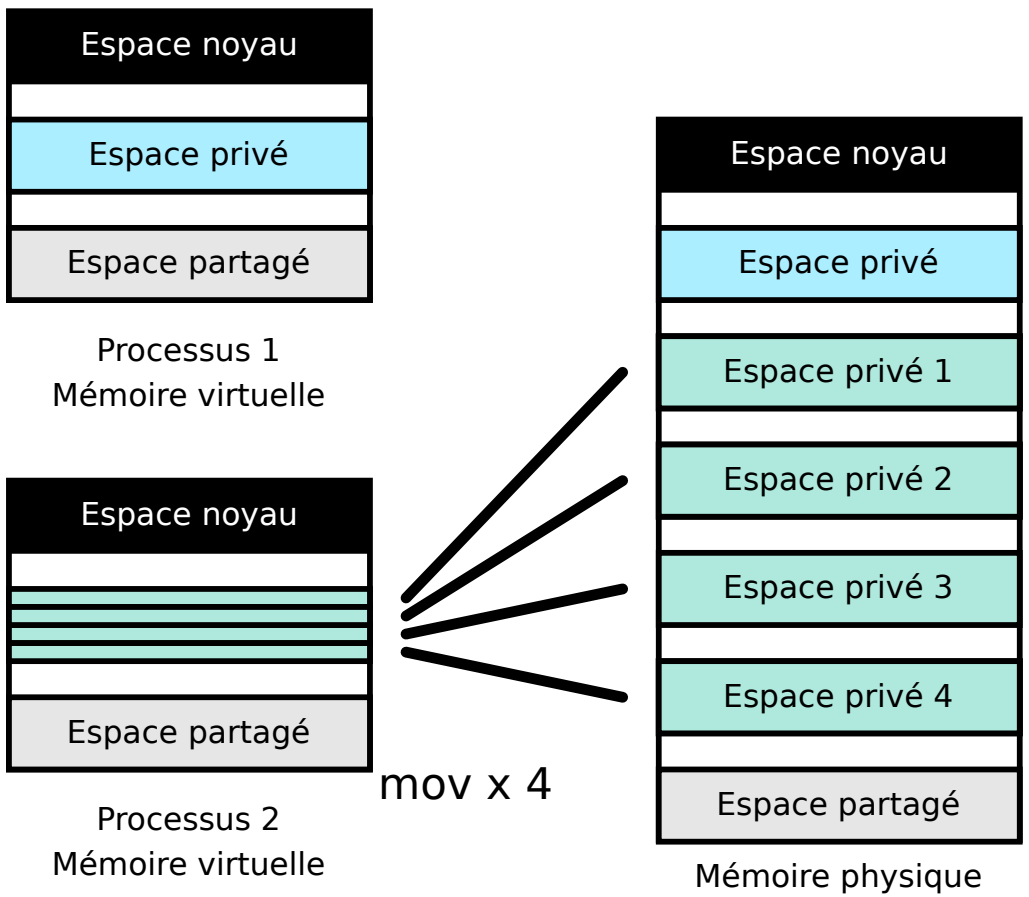
mov x 4

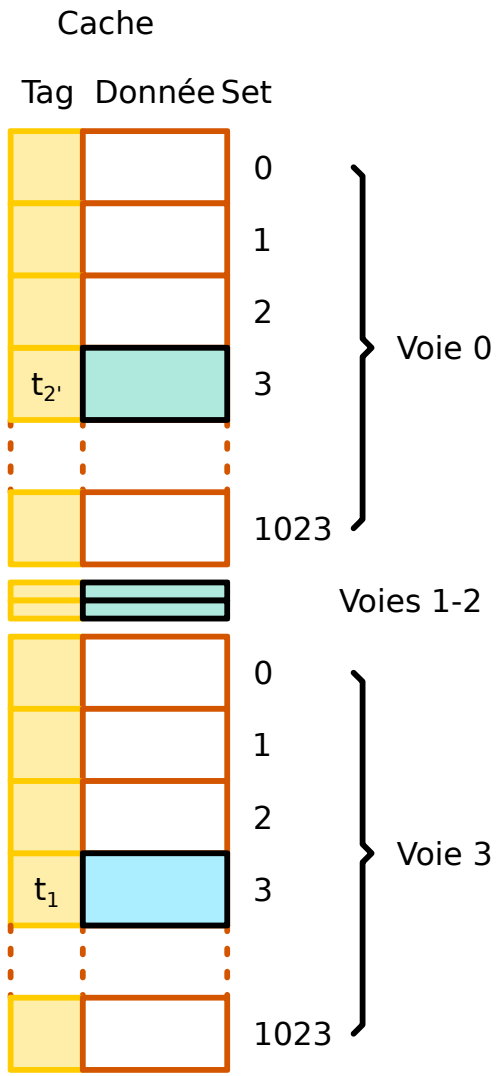


Mémoire physique

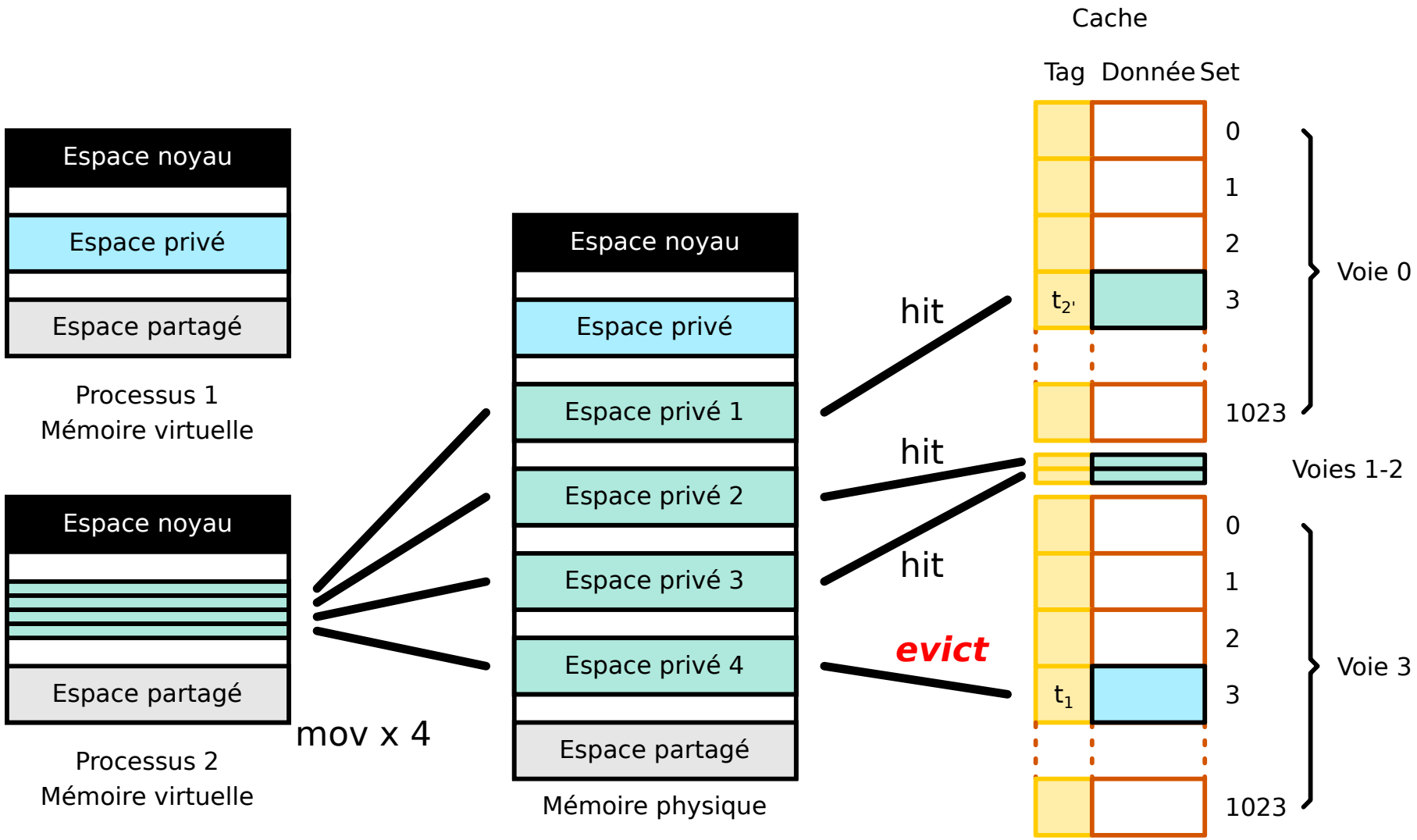


Cache set partagé (2)

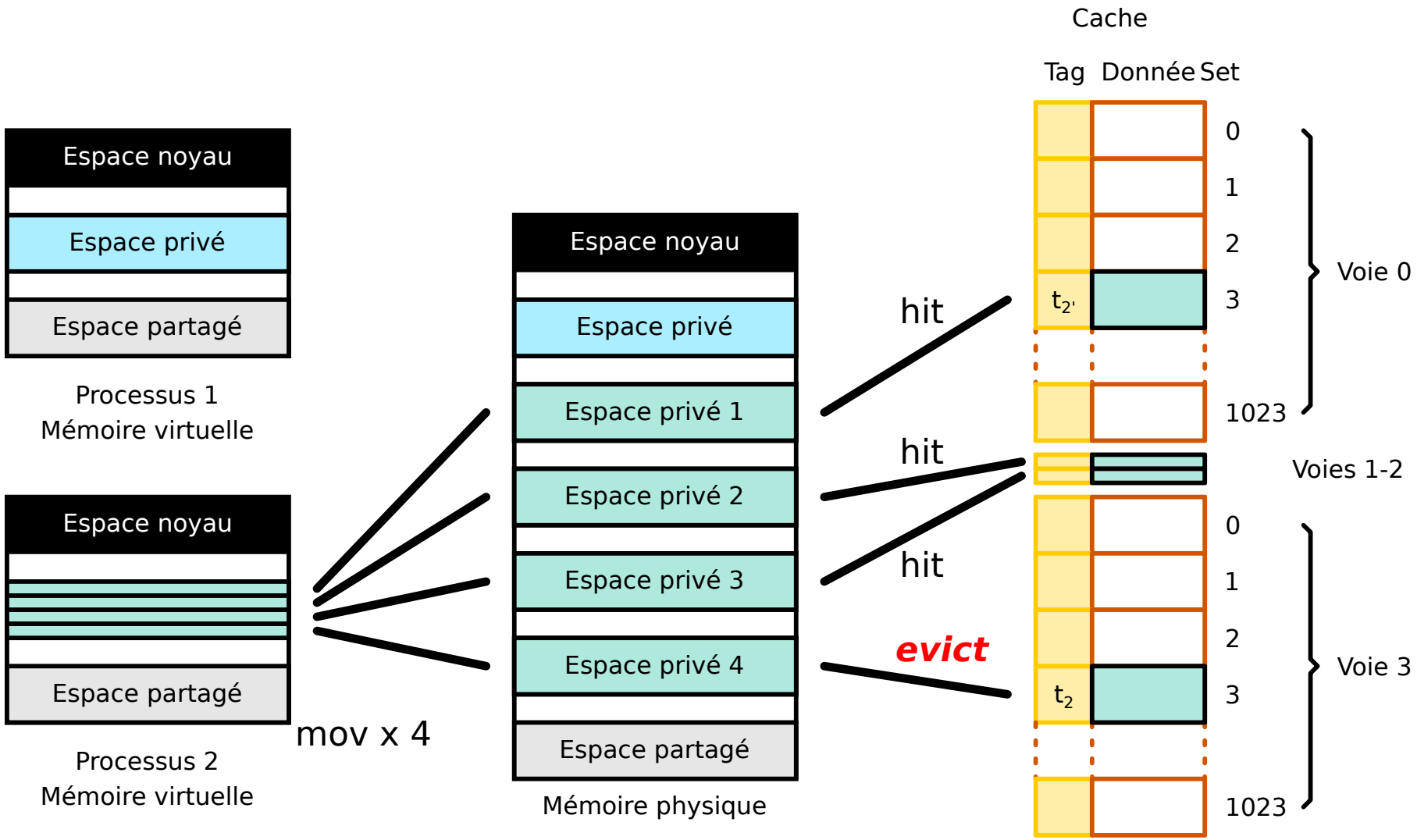




Cache set partagé (2)



Cache set partagé (2)



Attaque FLUSH+RELOAD

Prérequis : mémoire partagée

Définition

1. Étape **flush** : Éviction des addresses partagées avec `clflush` ;
2. Attente de l'exécution du processus cible ou source ;
3. Étape **reload** : accès aux addresses partagée avec mesure de temps.

Si accès rapide, l'attaquant y a accédé.

Attaque PRIME+PROBE

Prérequis 1 : connaissance du cache set
Prérequis 2 : ensemble d'éviction : allocation d'un ensemble d'éviction pour un set contenant n lignes différentes.

Définition

1. Étape **prime** : Éviction des addresses du cache set cible en accédant au lignes de l'ensembles d'éviction correspondant ;
2. Attente de l'exécution du processus cible ou source ;
3. Étape **probe** : accès à toutes les lignes du cache set et mesure du temps.

Si accès lent, l'attaquant a accédé à la ligne de cache cible congrue avec le cache set. I.e une ligne de cache du cache set aura été évictée.

Application : secret cryptographique (1)

```
uint8_t encrypt(uint8_t secret, uint8_t bloc) {
  return bloc ^ secret;
}
```

Side channel temporel ?

Application : secret cryptographique (1)

```
uint8_t encrypt(uint8_t secret, uint8_t bloc) {
  return bloc ^ secret;
}
```

Side channel temporel ?

Reponse : non.

Application : secret cryptographique (2)

```
uint8_t kdf[0x100];
uint8_t encrypt(uint8_t secret, uint8_t bloc) {
  return bloc ^ kdf[secret];
}
```

Side channel temporel ?

Application : secret cryptographique (2)

```
uint8_t kdf[0x100];
uint8_t encrypt(uint8_t secret, uint8_t bloc) {
  return bloc ^ kdf[secret];
}
```

Side channel temporel ?
Reponse : oui.

Application : canal caché

Canal caché avec mémoire partagée

- 1 ligne pour transmettre le symbole 0
- 1 ligne pour transmettre le symbole 1

Demo time !