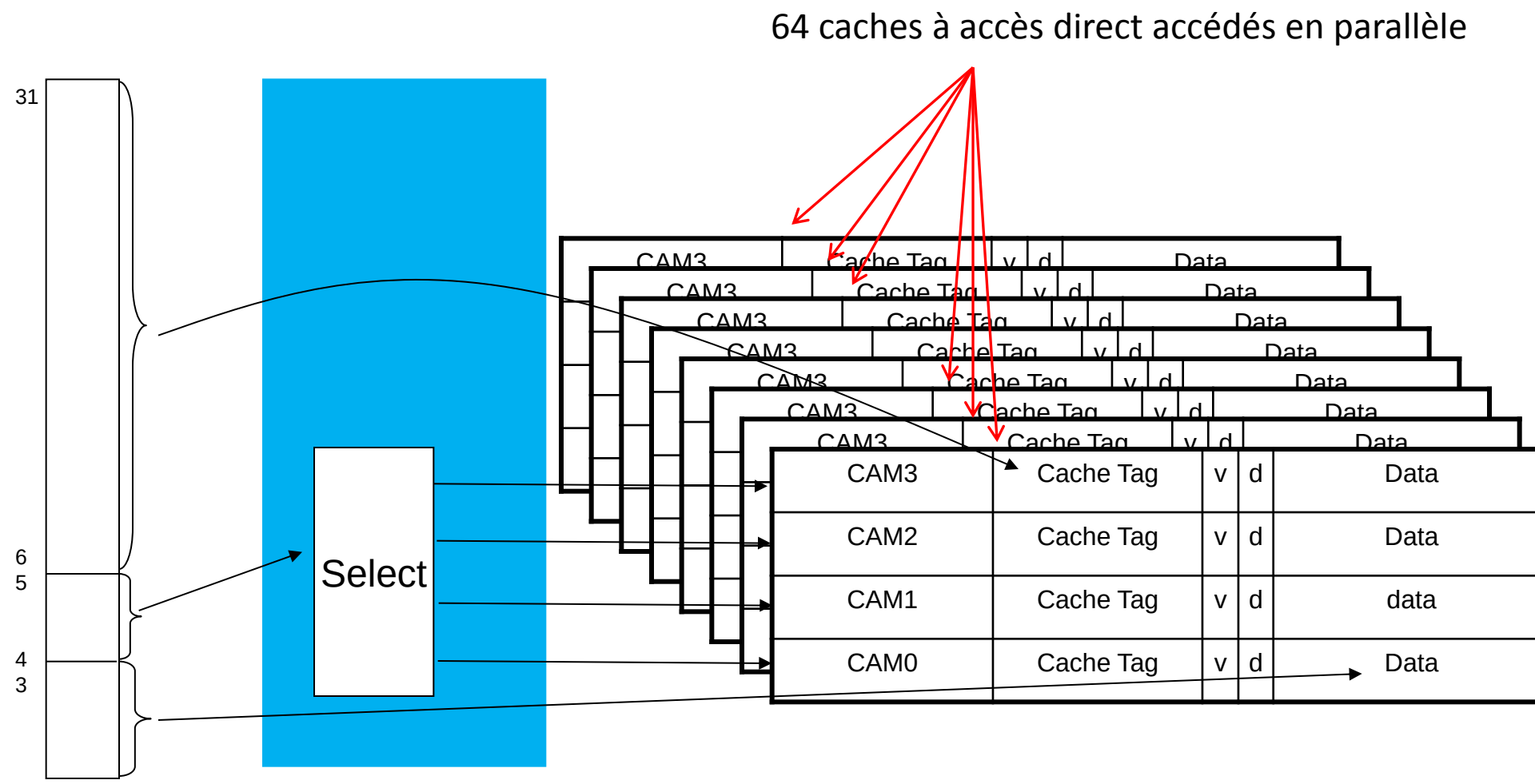
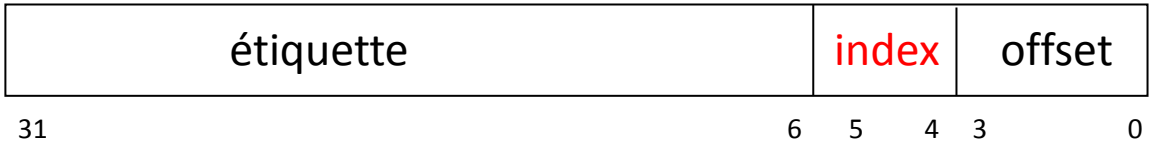
ARM940T - cache



ARM940T - cache

La mémoire cache décrite est prévue pour fonctionner avec un cœur ARM940T qui est une architecture de Harvard. Il y a donc 2 caches: 1 pour les instructions (I-cache) et 1 pour les données (D-cache)

Une structure de base (mémoire cache accès direct) est constitué de 4 lignes qui sont indexées par les bits 4 et 5 de l'adresse



Chaque ligne d'une structure de cache de base permet de stocker 16 bytes indexés par les bits 0,1,2,4 de la l'adresse.

Chaque structure de base à donc une capacité de 16x4=64 bytes

ARM940T - cache

En fait chaque ligne d'une structure de base est en fait une mémoire associative (CAM = Content Adressable Memory) où chaque entrée de la structure est répliquée 64 fois.

La capacité totale de la mémoire cache est donc 64 x 64 = 4Kbytes

Pseudo-code accès lecture

1. On décompose l'adresse en étiquette, index, offset
2. On accède la mémoire associative CAM[index] en présentant l'étiquette
3. La CAM répond oui/non une clé (Cache Tag) stockée à l'adresse CAM[index] correspond (64 comparaisons en parallèle)
4. Si oui la donnée est transférée du cache vers la mémoire. La donnée se trouve dans CAM[index] dans la structure de base dont la clé correspond à l'étiquette et l'offset détermine la donnée sur la ligne.
5. Si non, lecture d'une ligne de donnée (16 bytes) en mémoire centrale et stockage en mémoire cache. Si cache plein, remplacement d'une ligne.

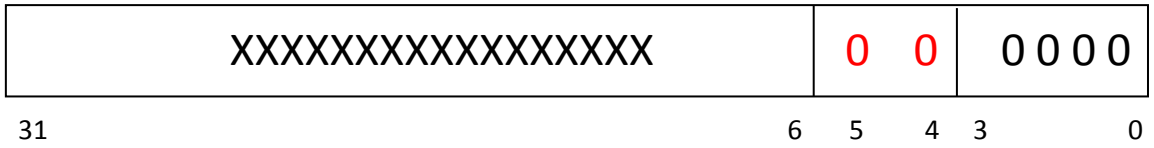
Retour sur l'exemple du cours

```
int readSet(int times, int numset) {
int setcount, value;
volatile int *newstart;
volatile int *start = (int*)0x20000;

_asm {
timesloop:
            MOV     newstart, start
            MOV     setcount, numset
setloop:
            LDR     value, [newstart, #0]
            ADD     newstart, newstart, #0x40
            SUBS    setcount, setcount, #1
            BNE     setloop
            SUBS    times, times, #1
            BNE     timesloop
}
    return value;
}
```

Retour sur l'exemple du cours

On accède des données qui se trouve aux adresses
0x20000, 0x20040, 0x20080, 0x200C0, 0x20100, etc.

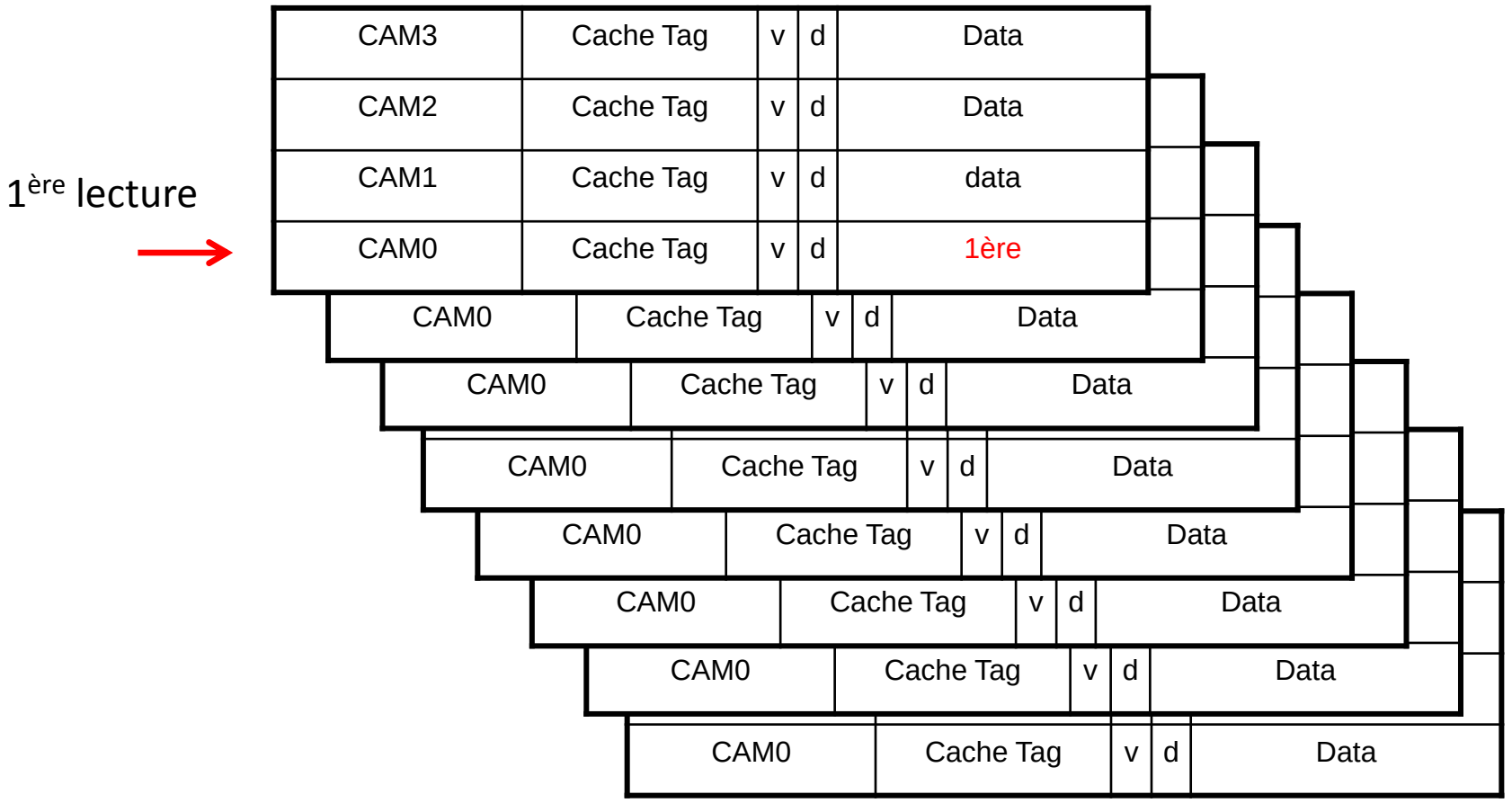


L'index de toutes les adresses vaut 0, c'est la mémoire CAM0 qui est sollicitée.
L'étiquette est différente pour tous les accès.

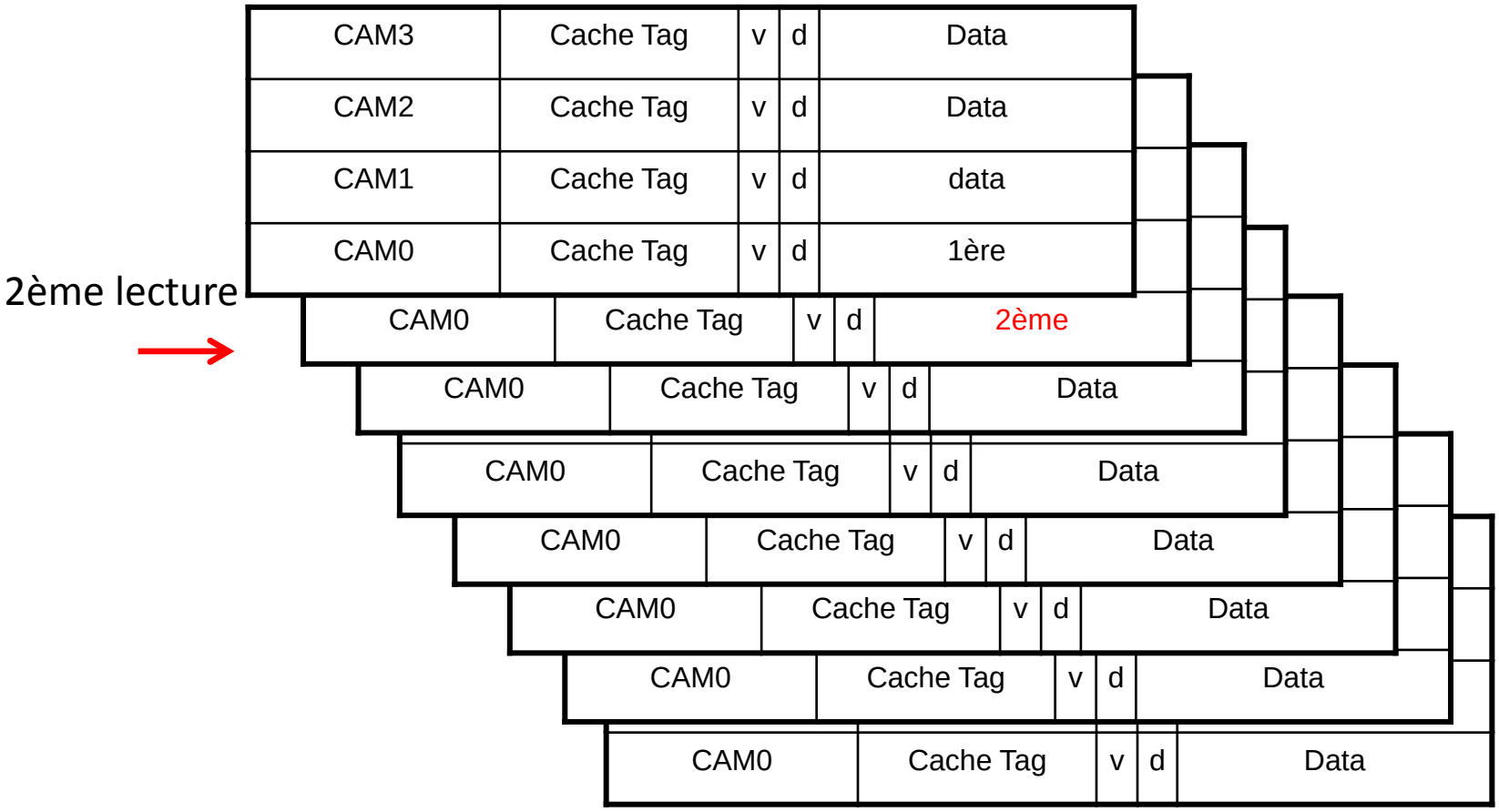
Donc, chaque accès va être stocké dans une des 64 structure parallèles différentes.

On décrit l'exécution si numset = 65

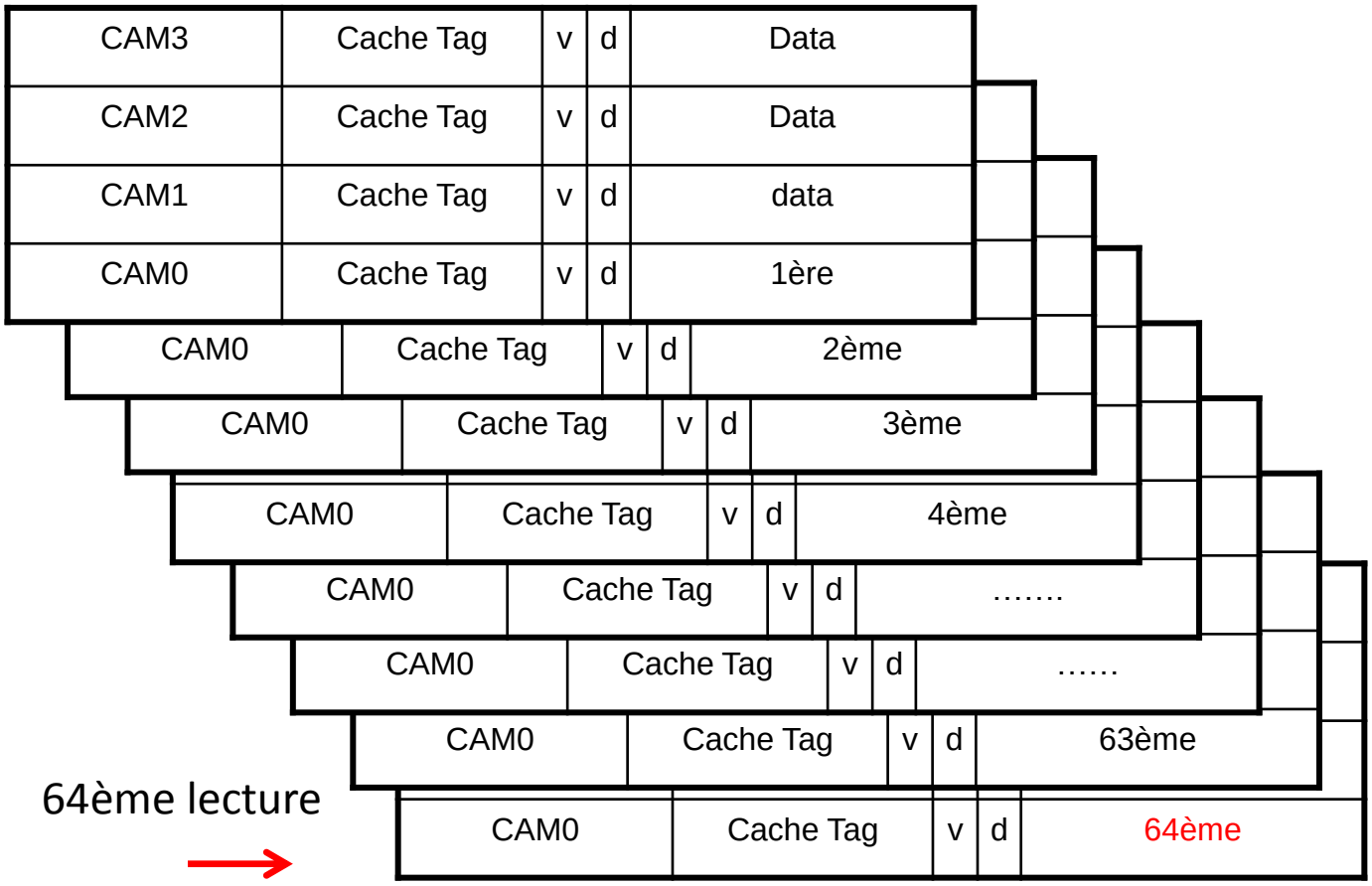
Allocation round-robin - setloop



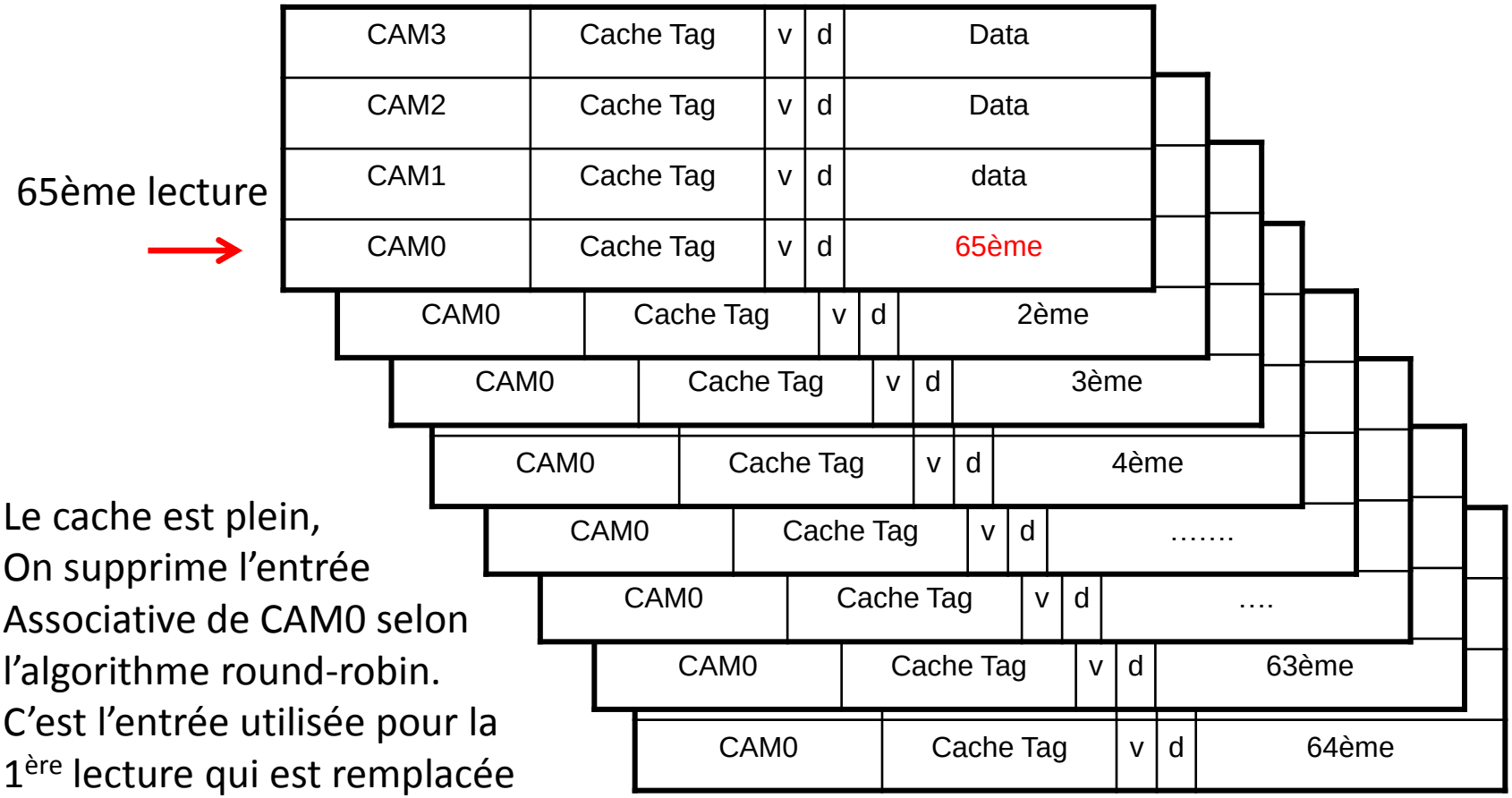
Allocation round-robin - setloop



Allocation round-robin - setloop



Allocation round-robin - setloop



Le cache est plein,
On supprime l'entrée Associative de CAM0 selon l'algorithme round-robin.
C'est l'entrée utilisée pour la 1ère lecture qui est remplacée

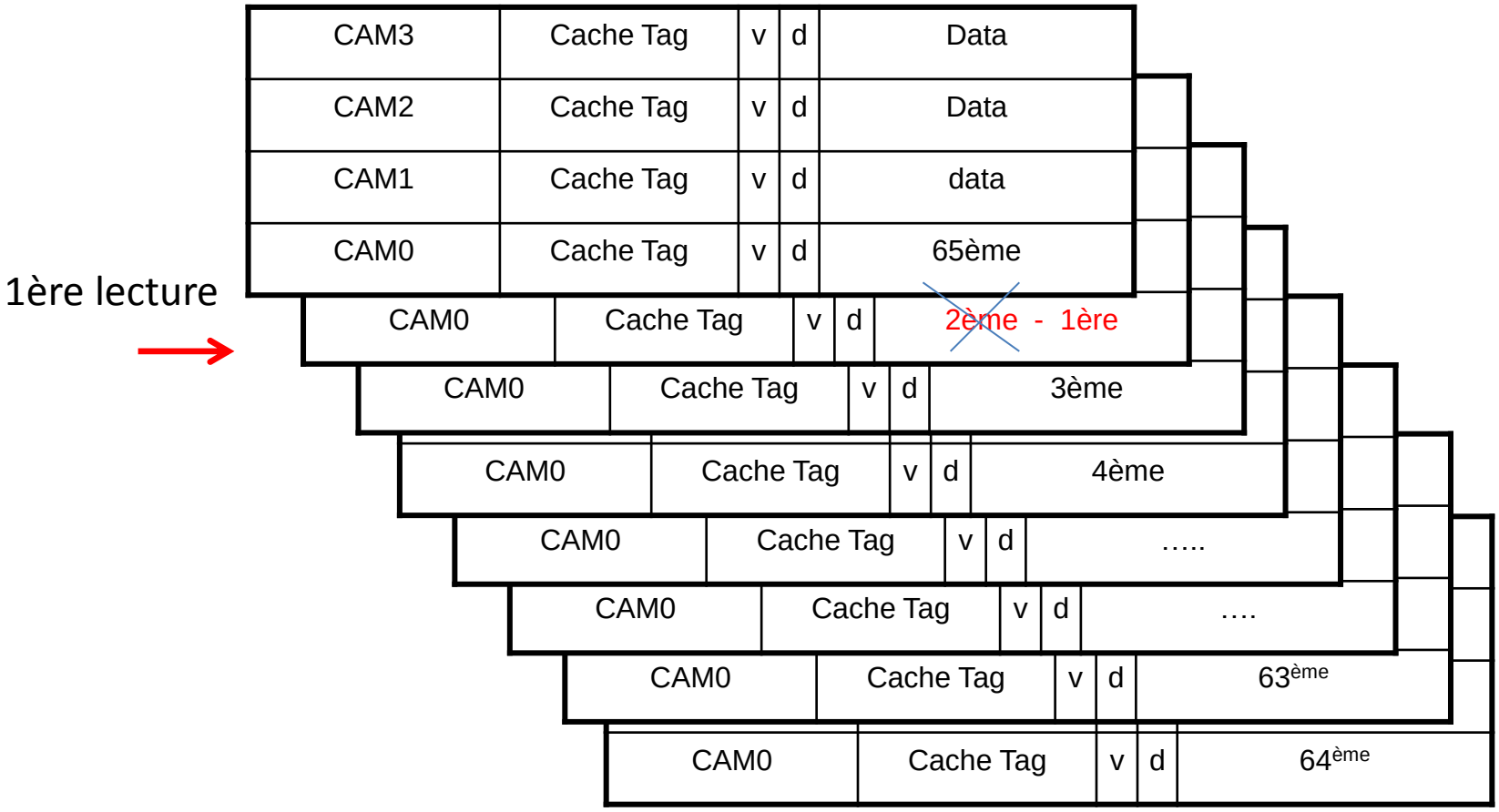
Allocation round-robin

Après l'exécution de la boucle setloop la mémoire associative est pleine et contient les données qui correspondent aux lectures 2, 3, 4, …, 65.

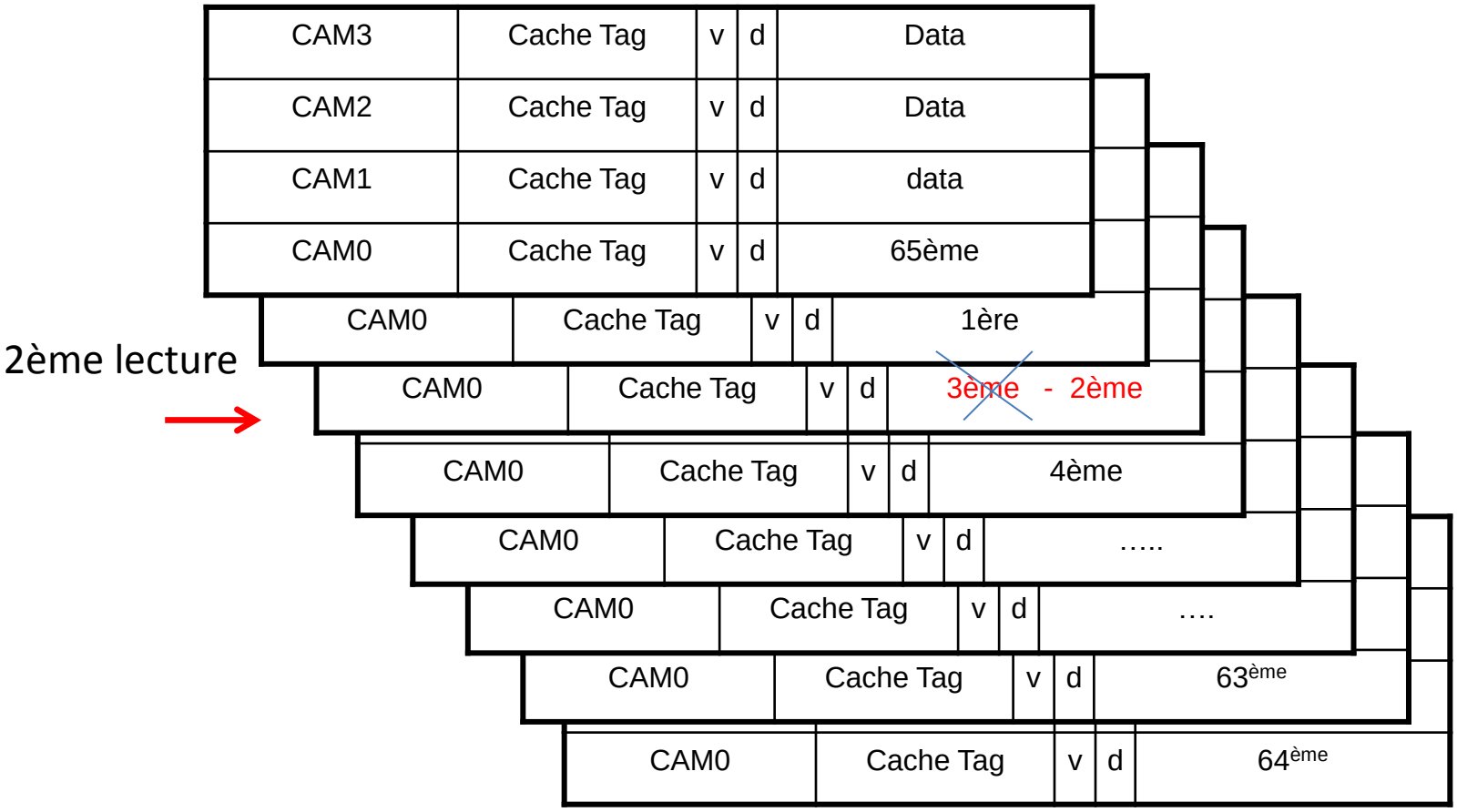
Le contenu de la première lecture a été remplacée par le contenu de la 65ème.

Au début de l'exécution de la deuxième boucle setloop, la prochaine ligne à être allouée est celle de la deuxième mémoire associative

Début deuxième exécution setloop



Début deuxième exécution setloop



Allocation round-robin

On voit qu'avec l'algorithme round-robin la donnée lue ne se trouve jamais dans la mémoire cache.

C'est un exemple (pathologique) de programme qui montre qu'il existe un programme pour lequel l'algorithme d'allocation round-robin ne fonctionne pas bien.

Allocation random

Random veut dire aléatoirement avec une distribution uniforme.

Cette fois, on dessine les 64 entrées de CAM0 en ligne

Après l'exécution des 64 premières lectures de la première boucle setloop on a,

8ème	60ème	23ème	49ème	17ème	52ème	1ère	64ème	…….	12ème

Après l'exécution de la première boucle setloop on a

8ème	60ème	23ème	49ème	65ème	52ème	1ère	64ème	…….	12ème

A la 65ème lecture il y a eu cache miss (comme les autres accès) mais pour la première fois le cache est plein et une entrée (aléatoire) est supprimée du cache.

Une donnée n'est pas présente dans le cache, c'est une parmi {1,2,…,64} avec probabilité uniforme.

Deuxième exécution de setloop

A début de l'exécution de la deuxième boucle setloop, le mémoire associative CAM0 est dans l'état

$8^{ème}$	$60^{ème}$	$23^{ème}$	$49^{ème}$	$65^{ème}$	$52^{ème}$	$1^{ère}$	$64^{ème}$	.....	$12^{ème}$

Il y a toujours une lecture sur les 64 premières qui va donne un cache miss, i.e.

P(#cache miss >1) = 1

(probabilité que le nombre de cache miss est plus grand que 1 = 1)

On note i la lecture qui va donner le premier cache miss. C'est n'importe laquelle parmi les 64 premières lectures avec une probabilité uniforme (=1/64).

La lecture i est celle qui a été remplacée par la $65^{ème}$.

$$\{1, 2, 3, \ldots, \hat{\imath}, \ldots, 64\} \bigcup \{65\}$$

Deuxième exécution de setloop

Si i=64 il va y avoir un cache miss à la 64ème lecture et un deuxième si et seulement si la ligne allouée pour stocker la 64ème lecture est la 65ème.

Si i=63 il va y avoir un cache miss à la 63ème lecture et un deuxième si et seulement si la ligne allouée pour stocker la 63ème lecture est la 64ème ou la 65ème .

idem pour toute les valeurs de i<64

$$\mathrm{P}(\#\text{cache miss} > 2) = \sum_{i=1}^{64} \frac{1}{64} \sum_{j=i+1}^{65} \frac{1}{64}$$

Deuxième exécution de setloop

$$\mathrm{P}(\#\text{cache miss} > 3) = \sum_{i=1}^{64} \frac{1}{64} \sum_{j=i+1}^{65} \frac{1}{64} \sum_{k=j+1}^{65} \frac{1}{64}$$

Pour rappel:

i est le numéro de la lecture qui à été remplacée par la 65ème à la fin de la première boucle setloop (1er cache miss de la deuxième boucle)

Lors de la lecture i il y a cache miss, la donnée de la lecture i est insérée dans le cache à la place de la donnée de la lecture j.

Lors de la lecture j, il y a cache miss, la donnée de la lecture j est insérée dans le cache à la place de la donnée de la lecture k.

Si i<j<k alors il y a au moins 3 cache miss.

Deuxième exécution de setloop

A la fin de la deuxième exécution de setloop, la mémoire cache est dans l'état

$$\{1, 2, 3, \ldots, \hat{\imath}, \ldots, 64\} \bigcup \{65\}$$

Avec i (qui n'est pas dans le cache) est 1, 2, .. Ou 64 avec probabilité uniforme.
C'est le même état que l'état initial de la deuxième exécution de la boucle.

On a donc la même formule pour les probabilités

P(#cache miss > n)

Pour le calcul des probabilités on utilise matlab et on obtient

E(#cache miss) = 1.72