

Architecture des Systèmes

L3 Informatique

Partie X : Mémoire cache

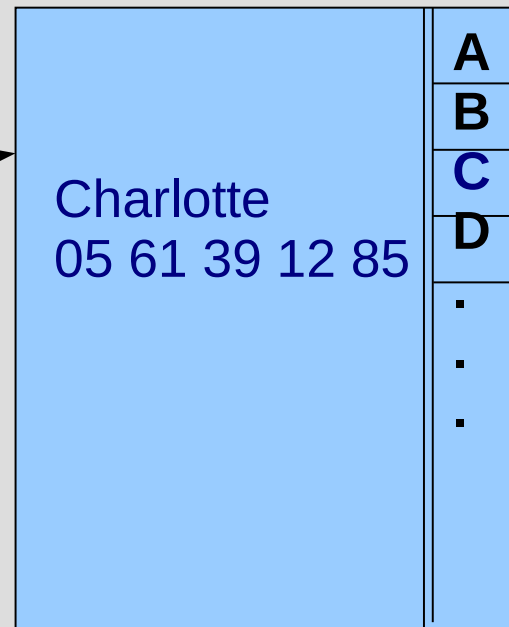
Nicolas Garric - nicolas.garric@univ-jfc.fr

Objectif

- Accéder plus rapidement à certaines informations
 - celles auxquelles on accède le plus souvent
 - celles auxquelles on est susceptible d'accéder prochainement
- Exemples :
 - cache web
 - proxy
 - cache disque
 - **cache mémoire**

Répertoire : accès direct

- Gros répertoire téléphonique
- Appel fréquent de quelques personnes seulement
- Cache du répertoire
 - 26 entrées, accès direct par première lettre
 - entrée C mémorise la dernière personne appelée dont le nom commence par C

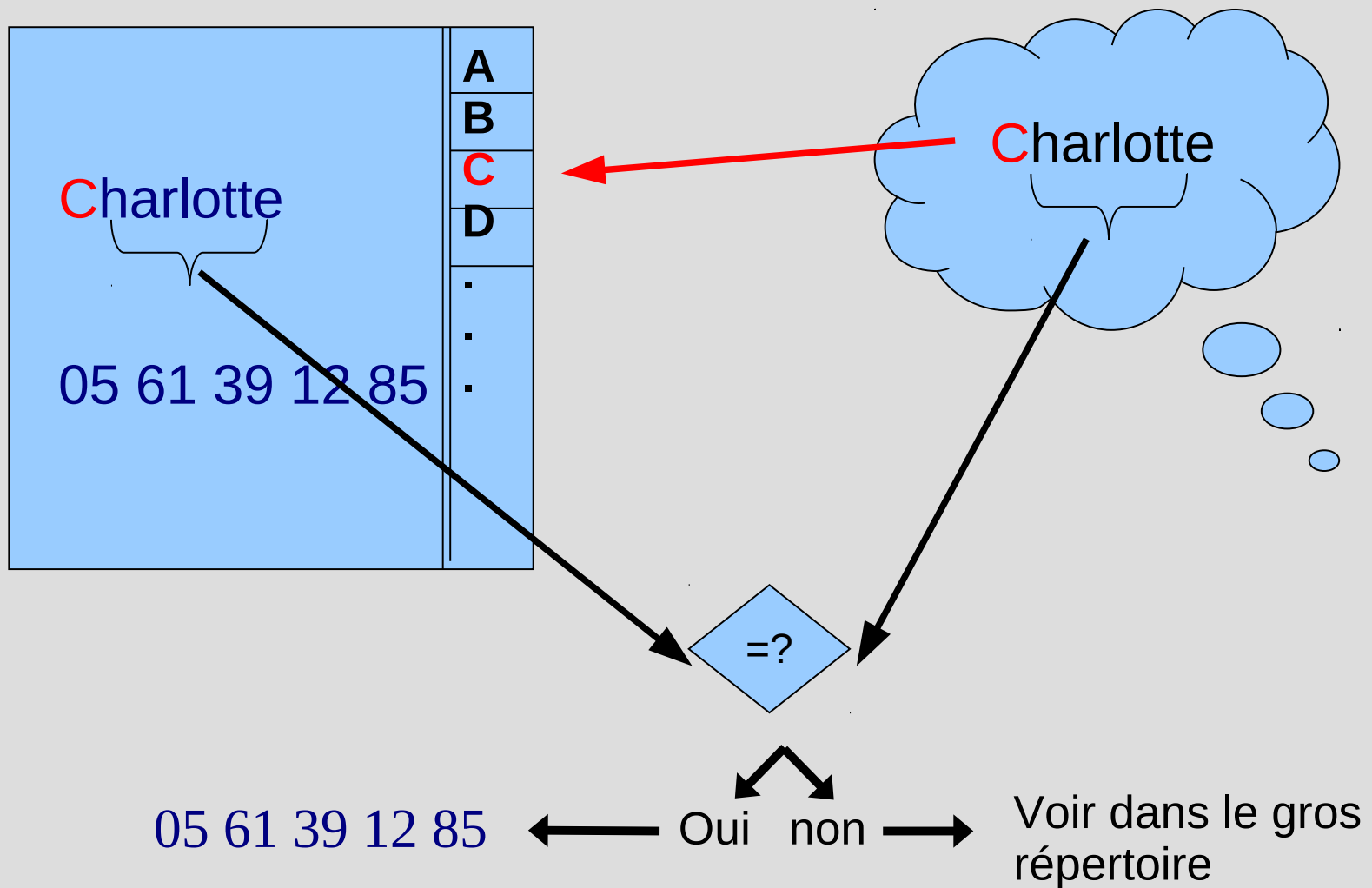


Charlotte 05 61 39 12 85	A
	B
	C
	D
	.



Charlotte 0561391285
Denis 0561392742
Christian 0422133345

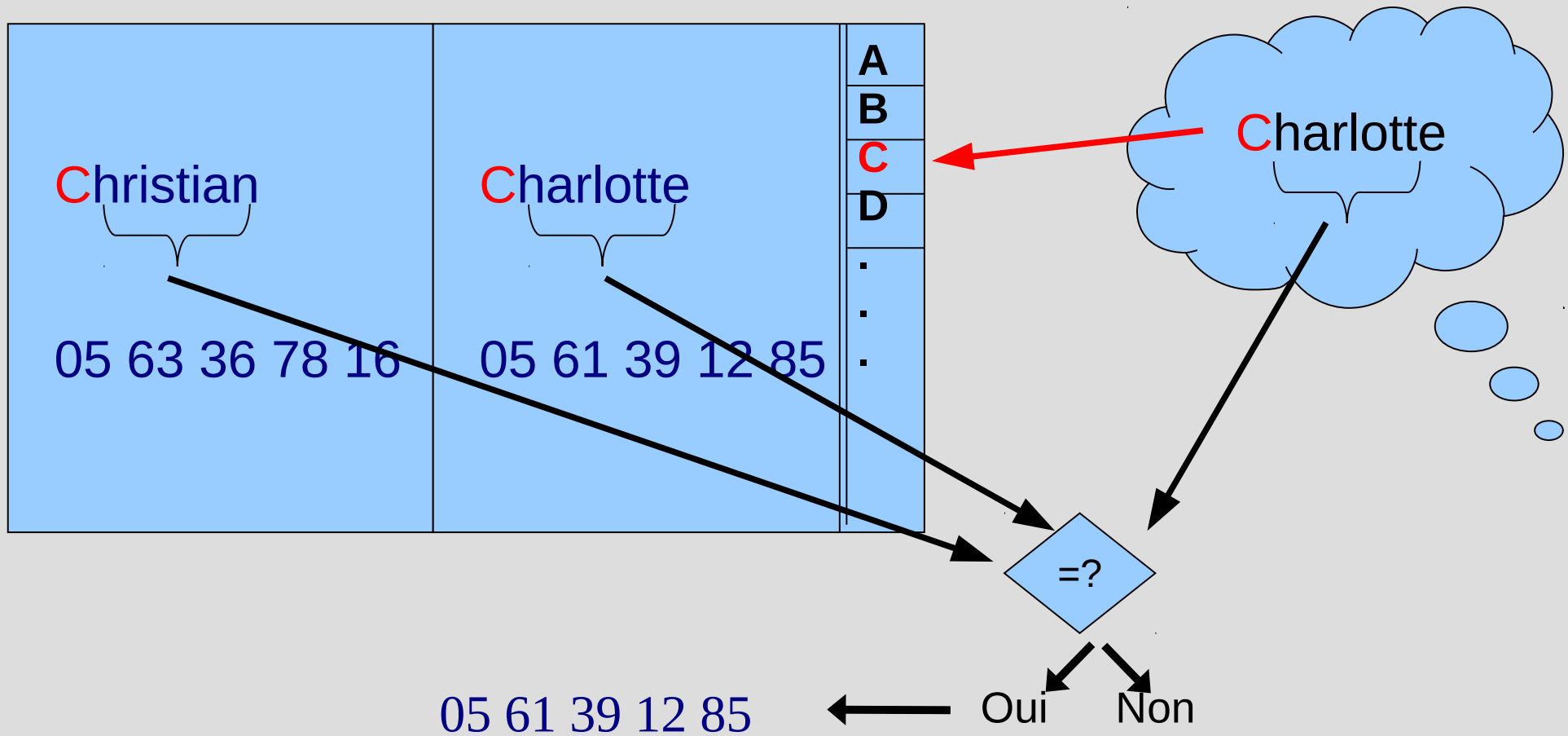
Utilisation du cache



Échec ou succès

- Problème : Charlotte et Christian
 - Appel Charlotte
 - Appel Christian → Échec
 - Appel Charlotte → Échec
 - Appel Charlotte → Succès
- Toutes les personnes dont le nom commence par C vont dans la même fiche.

Associativité par ensembles



Augmenter l'associativité

C	Chaïma	01 35 98 56 18	C	Cécile	05 68 12 34 78	C	Christian	05 63 36 78 16	C	Charlotte	05 61 39 12 85	A											
												B											
												C											
												D											
												.											
												.											
												.											

- Augmentation de la taille du cache
- Risque de nombreux emplacements vides dans le cache

Associativité totale

- Au maximum : un seul ensemble qui peut contenir n'importe quel prénom
 - plus difficile de trouver un prénom
 - pas d'index
 - besoin de comparer les 26 prénoms
 - Où mettre un nouveau prénom ?
 - emplacement vide
 - et s'ils sont tous pleins ?
 - enlever un prénom au hasard
 - enlever le moins récemment utilisé

En résumé

- Cache à accès direct : très rapide
 - Je regarde la page C. Si ce n'est pas le bon, je regarde dans le gros répertoire.
 - Si j'appelle un prénom commençant par C, je le mets dans la page C (et écrase celui qui y était déjà).
- Cache associatif
 - Plusieurs emplacements pour chaque lettre
 - Moins rapide
 - besoin de regarder plusieurs entrées dans un ensemble
 - Si l'ensemble est plein, quelle entrée on remplace ?

Efficacité du cache

- Pour que le cache ait un intérêt il faut que les pré-noms appelés soient plus souvent dans le cache que dans le gros répertoire.
- Dans le cadre d'un programme informatique, l'utilisation d'un cache est efficace car :
 - une instruction ou une donnée accédée a plus de chances d'être accédée par la suite (utilisation de variables et de boucles) : **localité temporelle**
 - une instruction voisine d'une instruction exécutée a plus de chances d'être exécutée : **localité spatiale**

Mémoire cache

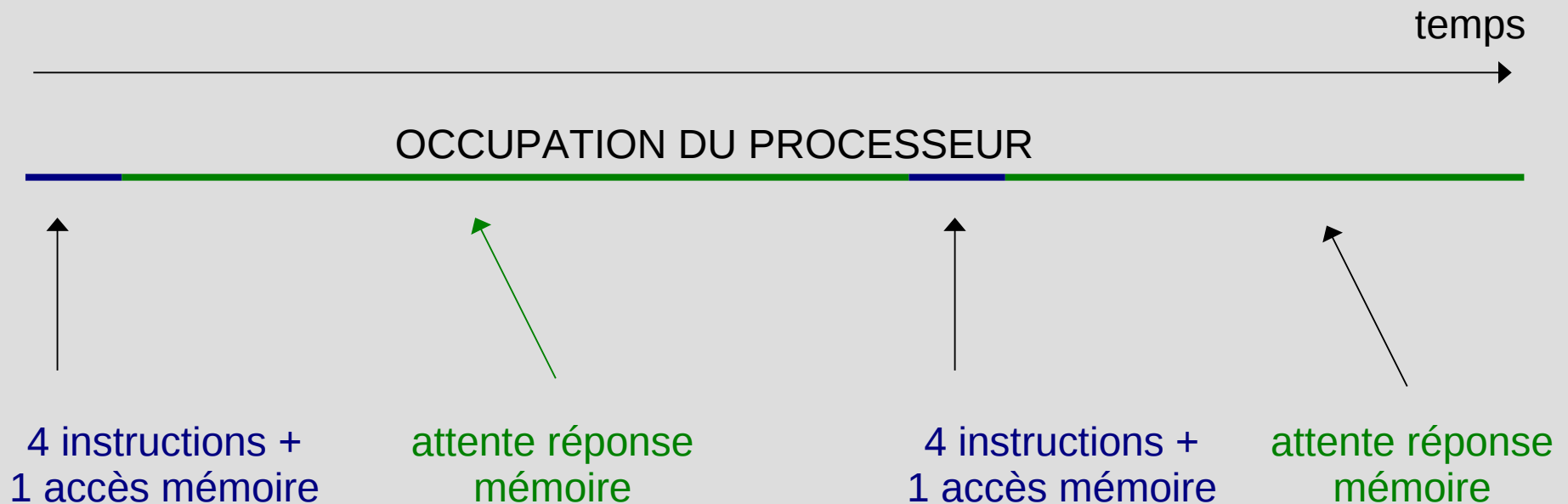
- Pourquoi a-t-on besoin de caches entre la mémoire et le processeur ?
- Comment ça marche ?
- Optimisation des paramètres
- Performances

Caractéristiques des mémoires

	Temps d'accès	Capacité	Coût par méga_octet
Disque	5 ms	500 Go	0,0002 €
Mémoire RAM dynamique	10 ns	1 Go	0,02 €
Mémoire RAM statique	5 ns	500 Ko	20 €
Registres	1 ns	64 de 4 octets	+++ €

Caractéristiques des programmes

- 20% de branchements
- 20% d'accès à la mémoire
- Temps d'accès à la mémoire centrale > 20 instructions
 - 5 instructions = 1 accès à la mémoire



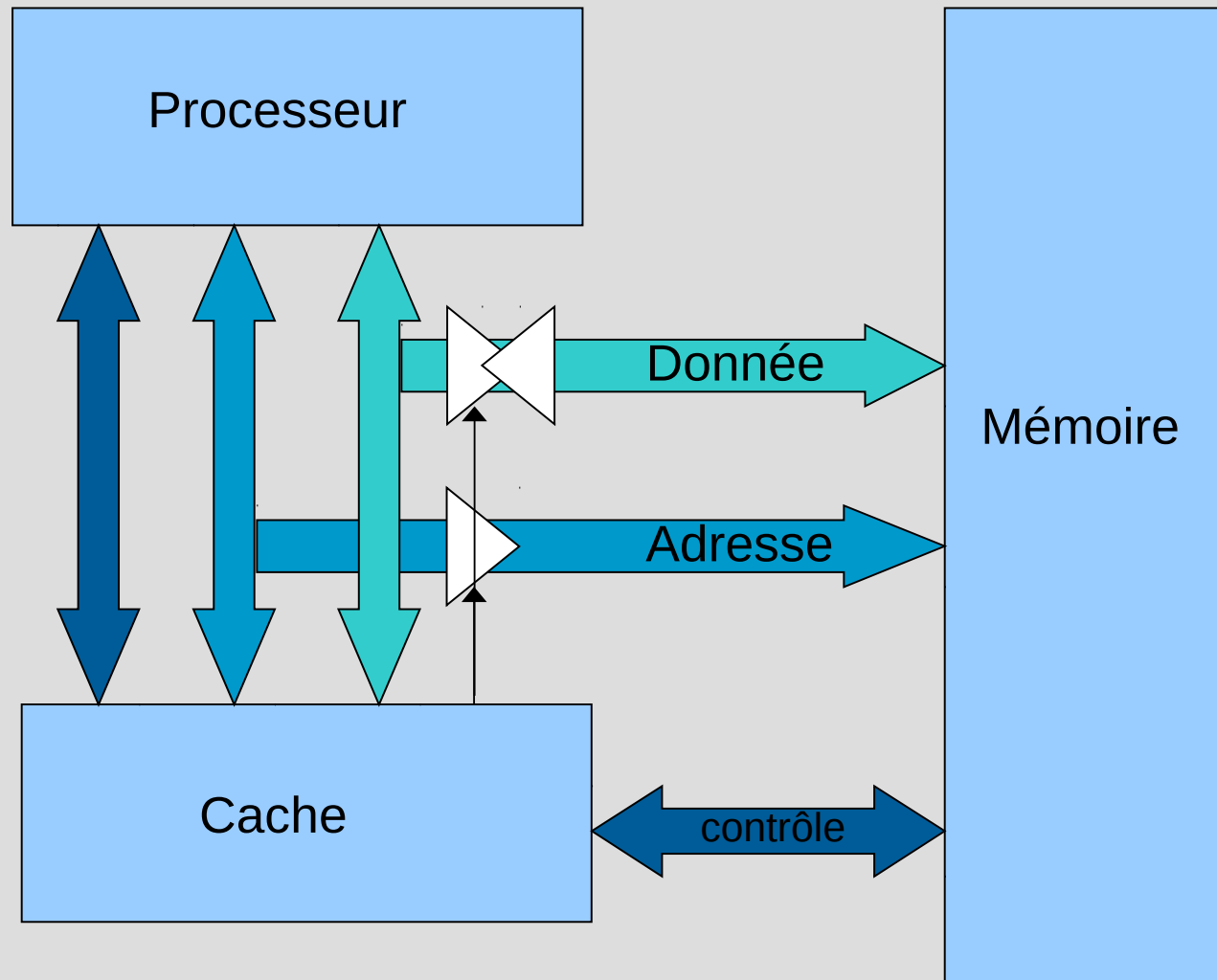
Mais les instructions aussi doivent être chargées depuis la mémoire !

Évolution

Année	Nombre d'instructions pouvant être exécutées durant un accès à la mémoire
1995	10
2000	30
2005	60

- La mémoire est de moins en moins rapide par rapport au processeur

Architecture générale



Fonctionnement général (1)

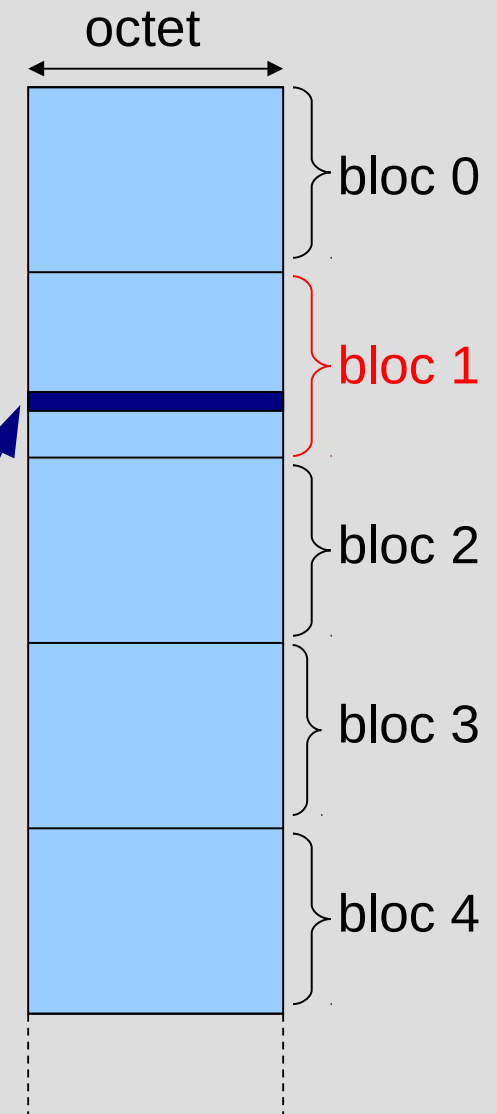
- Lecture par le processeur d'une instruction (ou d'une donnée)
- Recherche dans le cache (**comment ?**)
- Si succès, le cache renvoie l'instruction sur le bus de données
- Si échec, accès à la mémoire
 - la mémoire renvoie l'instruction sur le bus de données
 - l'instruction est automatiquement recopiée dans le cache (**où ?**)

Fonctionnement général (2)

- pour profiter de la localité spatiale
 - la mémoire renvoie l'instruction et ses voisines (**combien ? lesquelles ?**)
 - elles sont recopiées dans le cache (**où ?**)
- Cas de l'écriture d'une donnée
 - si la donnée est dans le cache : **écriture dans le cache ? dans la mémoire ? dans les 2 ?**
 - si la donnée n'est pas dans le cache : écriture dans la mémoire. **Recopie dans le cache ?**

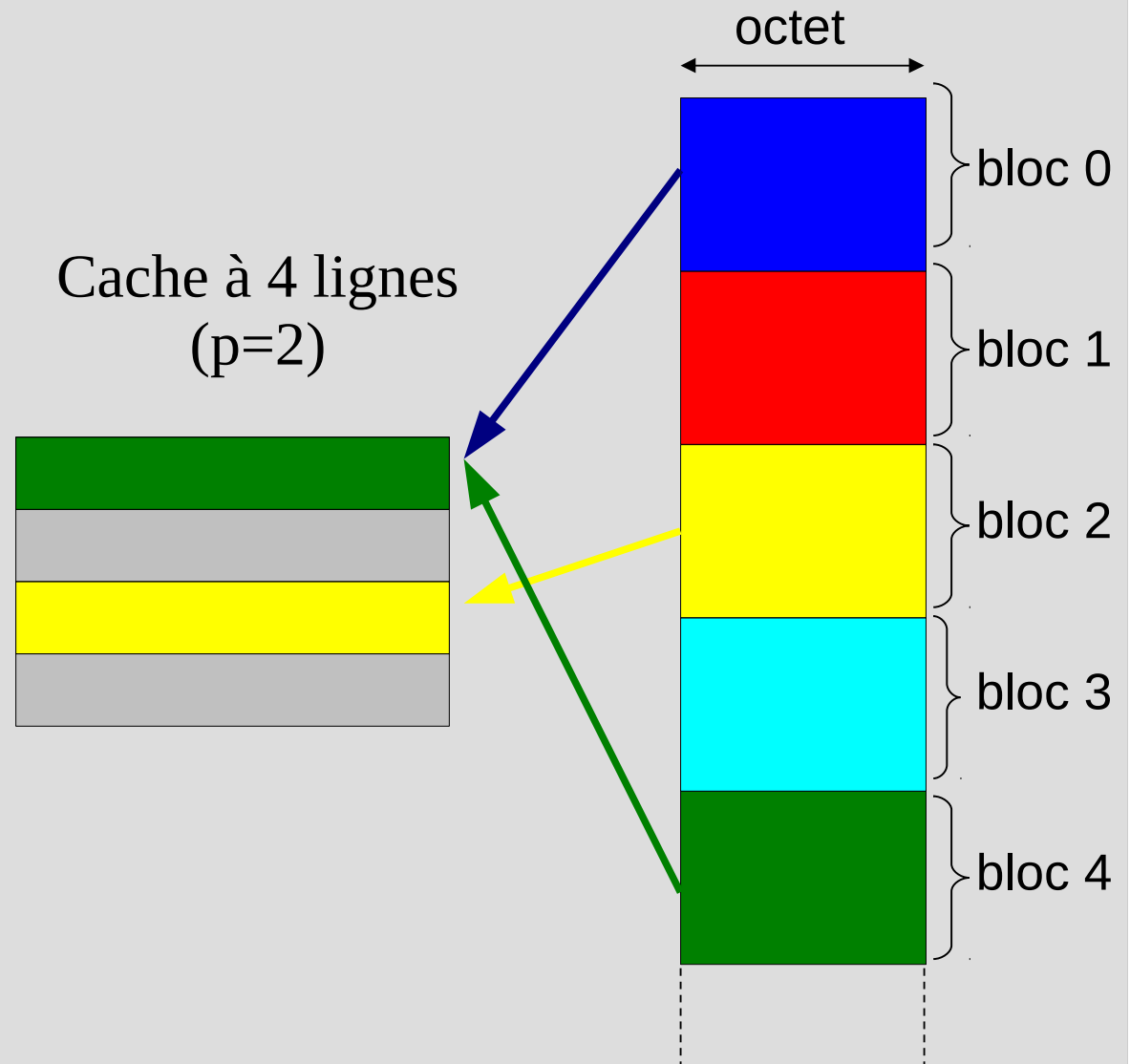
Découpage en blocs

- On découpe l'espace mémoire en blocs de taille 2^n .
- Ainsi les n bits de poids faible d'une adresse indiquent l'adresse dans le bloc et les bits de poids fort le numéro de bloc
- Exemple : $n=5 \Rightarrow$ blocs de 32 octets
- L'octet d'adresse **111000** :
 - se situe dans le **bloc 1**
 - son adresse dans le bloc est **11000**



Cache direct (1)

- Un cache direct contient 2^p lignes
- Chaque ligne est de la taille d'un bloc
- Les p bits de poids faibles du numéro de bloc indiquent dans quelle ligne du cache est chargé un bloc.



Cache direct (2)

- Pour savoir quel bloc est stocké dans une ligne du cache il faut stocker les premiers bits du numéro de bloc : on appelle ce champ **l'étiquette** du bloc
- Une adresse mémoire se décompose donc en 3 :
 - l'adresse dans le bloc (ex = 5bits)
 - le numéro de ligne du cache correspondant (ex = 2 bits)
 - l'étiquette identifiant le bloc
- Exemple : adresse de 16 bits, taille bloc = 32 octets et 4 lignes de cache
 - 010010101 11 00011

Exemples de lecture

- Lecture de :

- 0000000000 00 00011
- 0100111110 01 00111
- 0000000000 00 00000

Échec

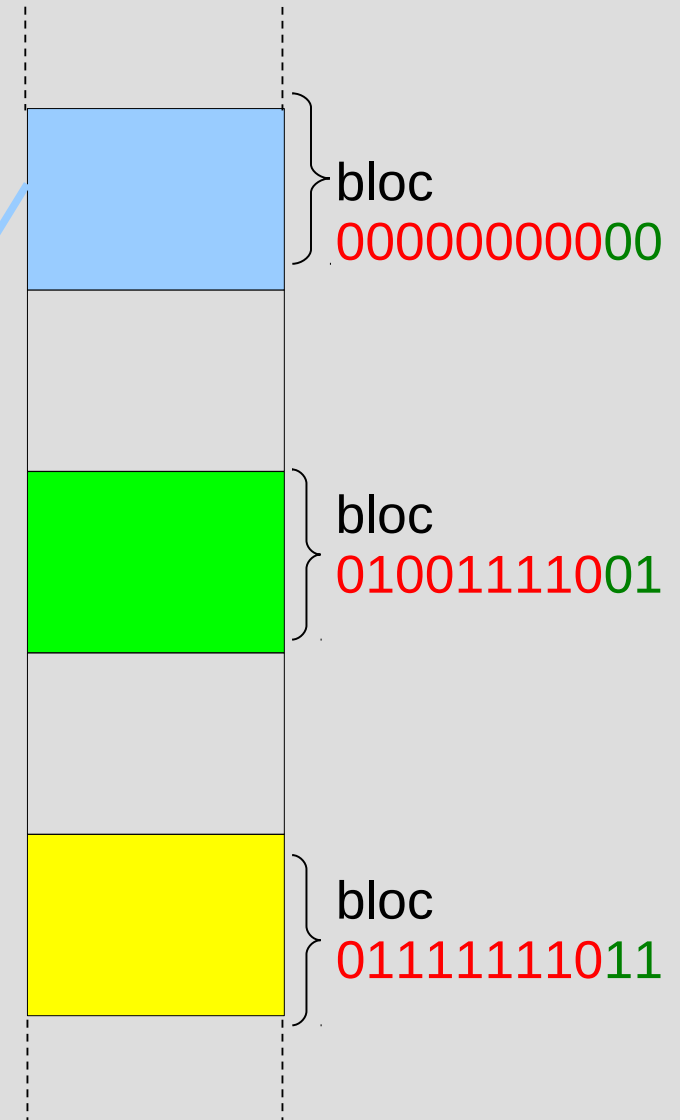
Échec

Succès

Cache à 4 lignes

	0000000000	
	0100111110	
	010011011	
	000110000	

bit de validité

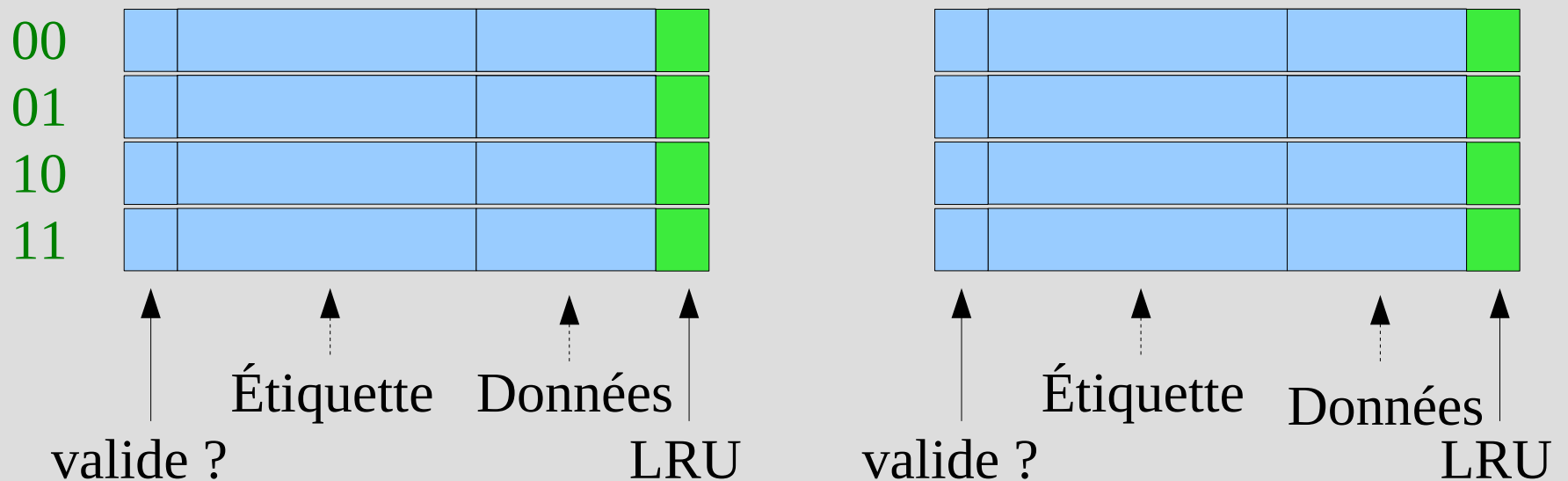


Cas d'une écriture

- en cas de succès :
 - soit **écriture simultanée** (*write-through*) dans le cache et en mémoire
 - soit **écriture différée** en mémoire (*write-back*) lors du remplacement du bloc par un autre
- en cas d'échec :
 - soit **écriture allouée** : bloc copié dans le cache
 - soit **écriture non allouée** : écriture seulement en mémoire

Associativité par ensembles

- Pour chaque ligne \rightarrow k entrées (ici k=2)



- Pour chaque adresse : 2 entrées possibles du cache
- Si toutes les entrées sont pleines ?

LRU

- LRU = Last Recently Used
- Si toutes les entrées sont pleines, le bloc est chargé à la place de celui le moins récemment utilisé.
- Exemple d'algorithme :
 - chaque entrée possède une valeur LRU correspondant à son ancienneté (entre 1 et le nombre d'entrées)
 - chaque utilisation d'une entrée remet à 1 son LRU et met à jour les autres entrées

Taux d'échec / nombre d'entrées

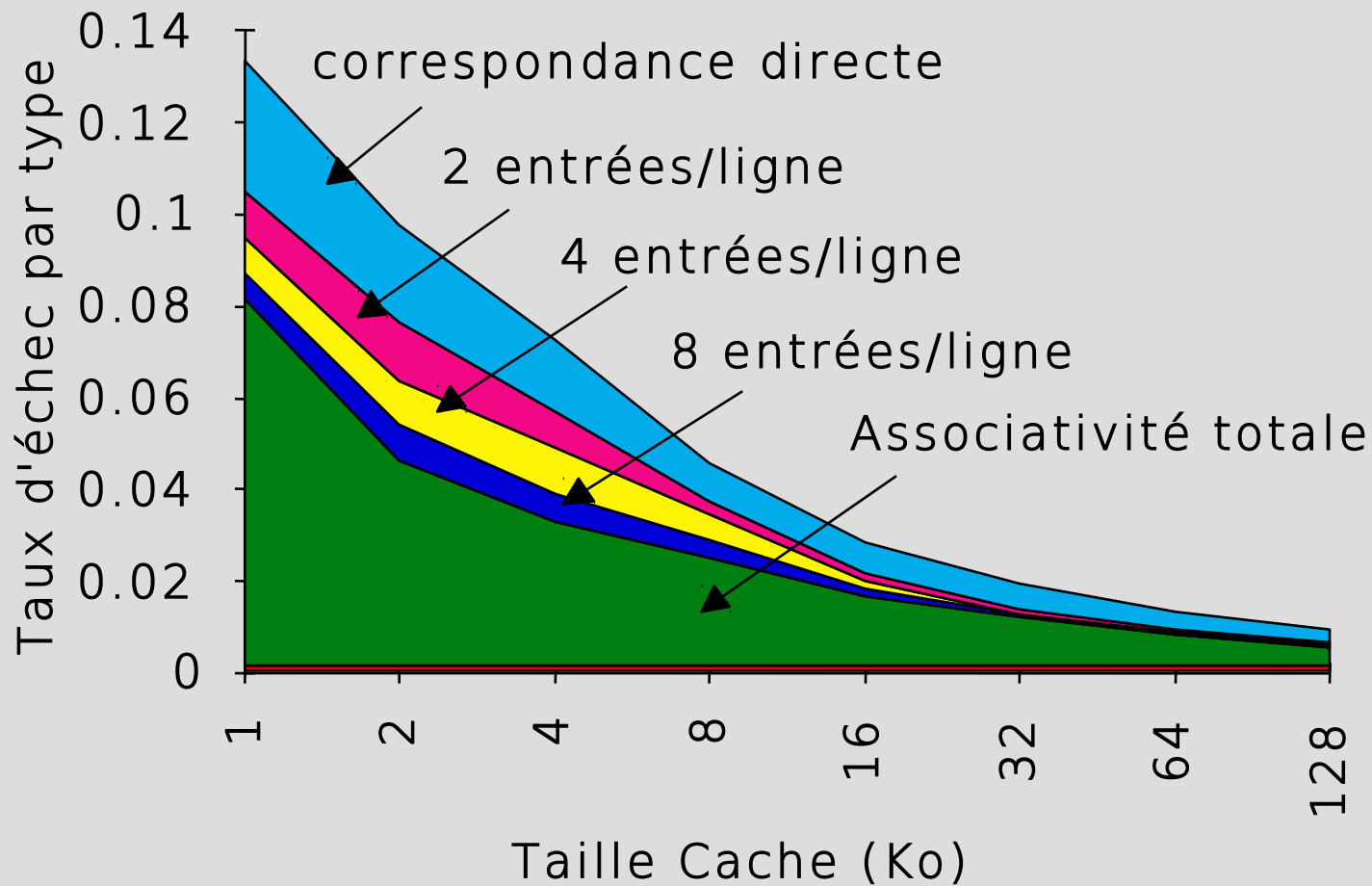
associativité	2 entrées/ligne		4 entrées/ligne		8 entrées/ligne	
Taille	LRU	hasard	LRU	hasard	LRU	hasard
16 Ko	5,18%	5,69%	4,67%	5,29%	4,39%	4,96%
64 Ko	1,88%	2,01%	1,54%	1,66%	1,39%	1,53%
256 Ko	1,15%	1,17%	1,13%	1,13%	1,12%	1,12%

LRU = plus complexe à mettre en oeuvre

Associativité totale

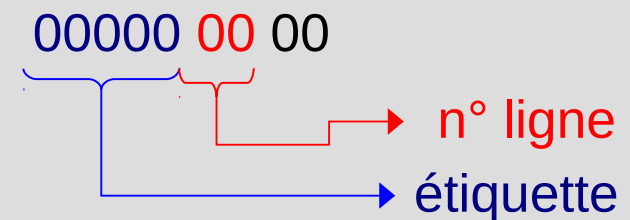
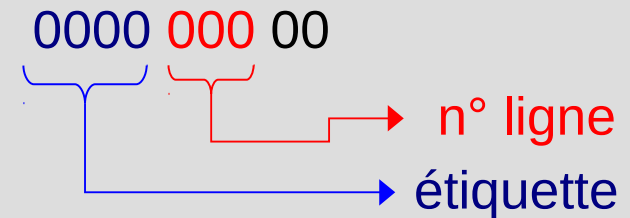
- Cache = 1 ligne avec n entrées
- Un bloc peut aller dans n'importe quelle entrée
- Pour savoir si un bloc est dans le cache :
 - Comparaison des étiquettes de toutes les entrées !
- Algorithme LRU :
 - Gérer le champs LRU de toutes les entrées !

Taux d'échec / nombre d'entrées



Continuum d'associativité

- Accès direct
 - n ensembles à 1 entrée
- Associativité par ensemble
 - n/a ensembles à a entrées
- Associativité totale
 - 1 ensemble à n entrées



Pourquoi le cache est efficace ?

- Localité temporelle
 - accès à une variable
 - boucles
- Localité spatiale
 - parcours d'un tableau : accès à `tab[0]`
 - ramène dans le cache `tab[0]` à `tab[x]` ($x \leq \text{taille bloc}$)
 - boucles

Caches séparés

- On peut utiliser 2 caches distincts :
 - un pour les instructions
 - un pour les données
- Les opérations mémoires sur les données et les instructions peuvent se faire en parallèle (utile si pipeline)
- Ex cache L1 données Pentium 4 :
 - taille = 8 Ko
 - associatif à 4 entrées
 - taille d'un bloc = 64 octets
 - écriture directe (write-through)

Cache de niveau 2

- Pour améliorer les performances on peut ajouter un cache de niveau 2 (L2 = Level2)
- Ex cache L2 Pentium 4 :
 - taille = 1 Mo
 - associatif à 8 entrées
 - taille d'un bloc = 128 octets
 - écriture différée (write-back)
- Ce cache est souvent unifié (données + instructions)

Cache de niveau 3

- On peut ajouter un cache de niveau 3.
 - Situé sur la carte processeur
 - 16 Mo de SRAM
- Les différents caches sont inclusifs :
 - le cache L3 contient les données du cache L2
 - le cache L2 contient les données du cache L1