

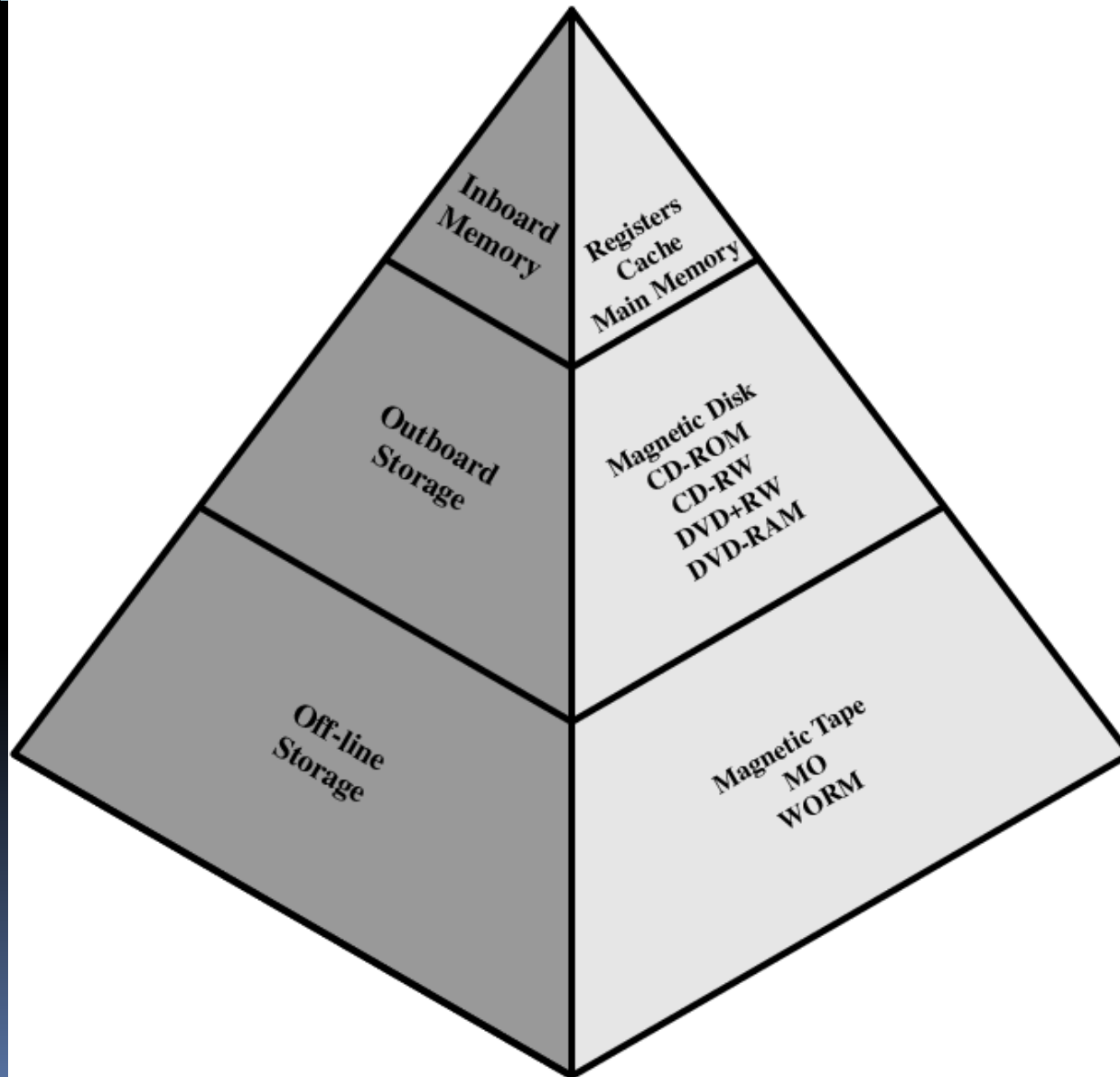


Jean-luc.dekeyser@lifl.fr

Version 2013

FONCTION MÉMOIRE

Memory Hierarchy - Diagram

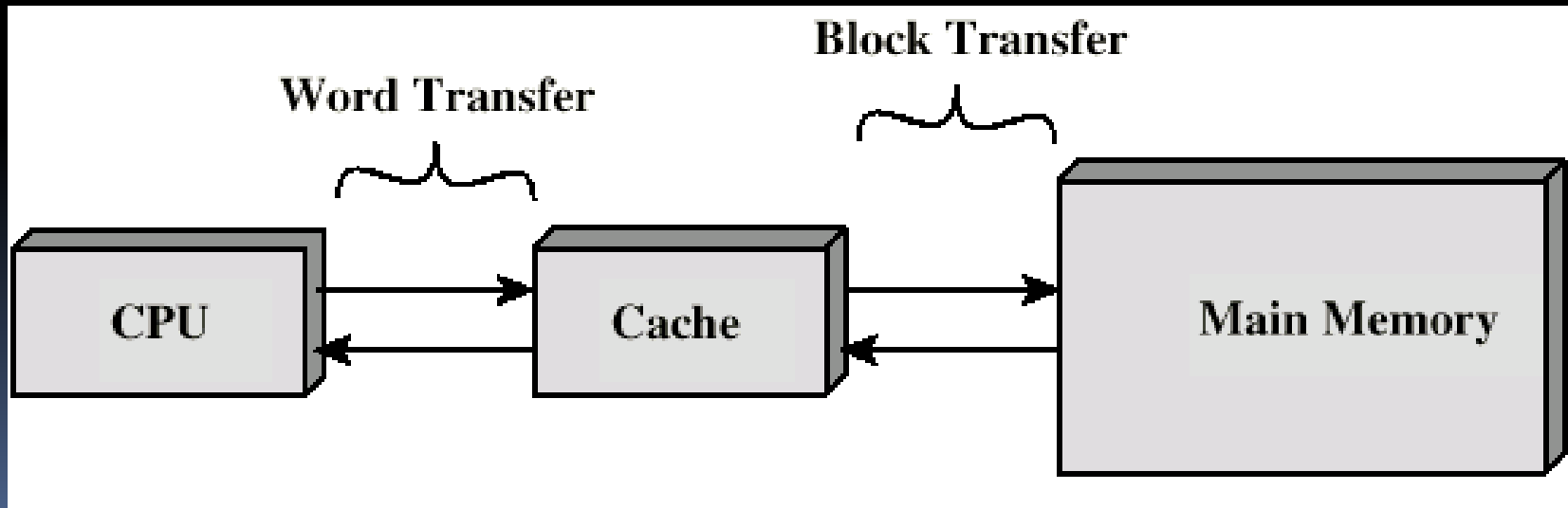


Mémoires caches

- Compromis entre la taille et la vitesse de la mémoire.
- La mémoire cache est :
 - de petite capacité
 - de grande vitesse (facteur 10 entre la cache et la mémoire conventionnelle)
- Réduire le temps d'accès mémoire moyen en conservant une mémoire importante


Cache

- Située entre le CPU et la mémoire centrale
- Peut être interne ou externe
- Ransfert par mot, par bloc





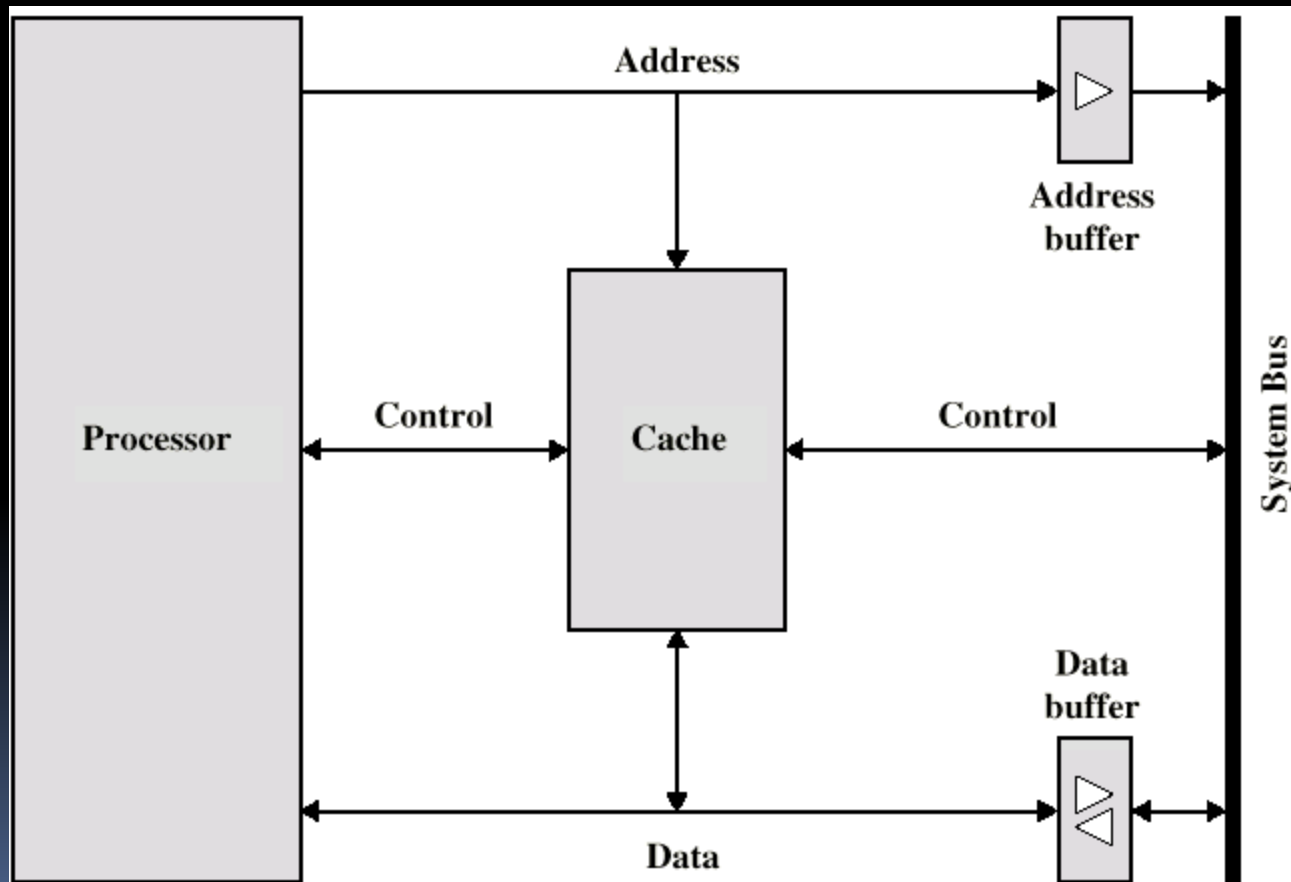
Caractéristique d'un cache

- Taille
 - Fonction de Mapping
 - Algorithme de Remplacement
 - Write Policy
 - Taille des Blocks
 - Nombre de Caches
- 

Importance de la taille

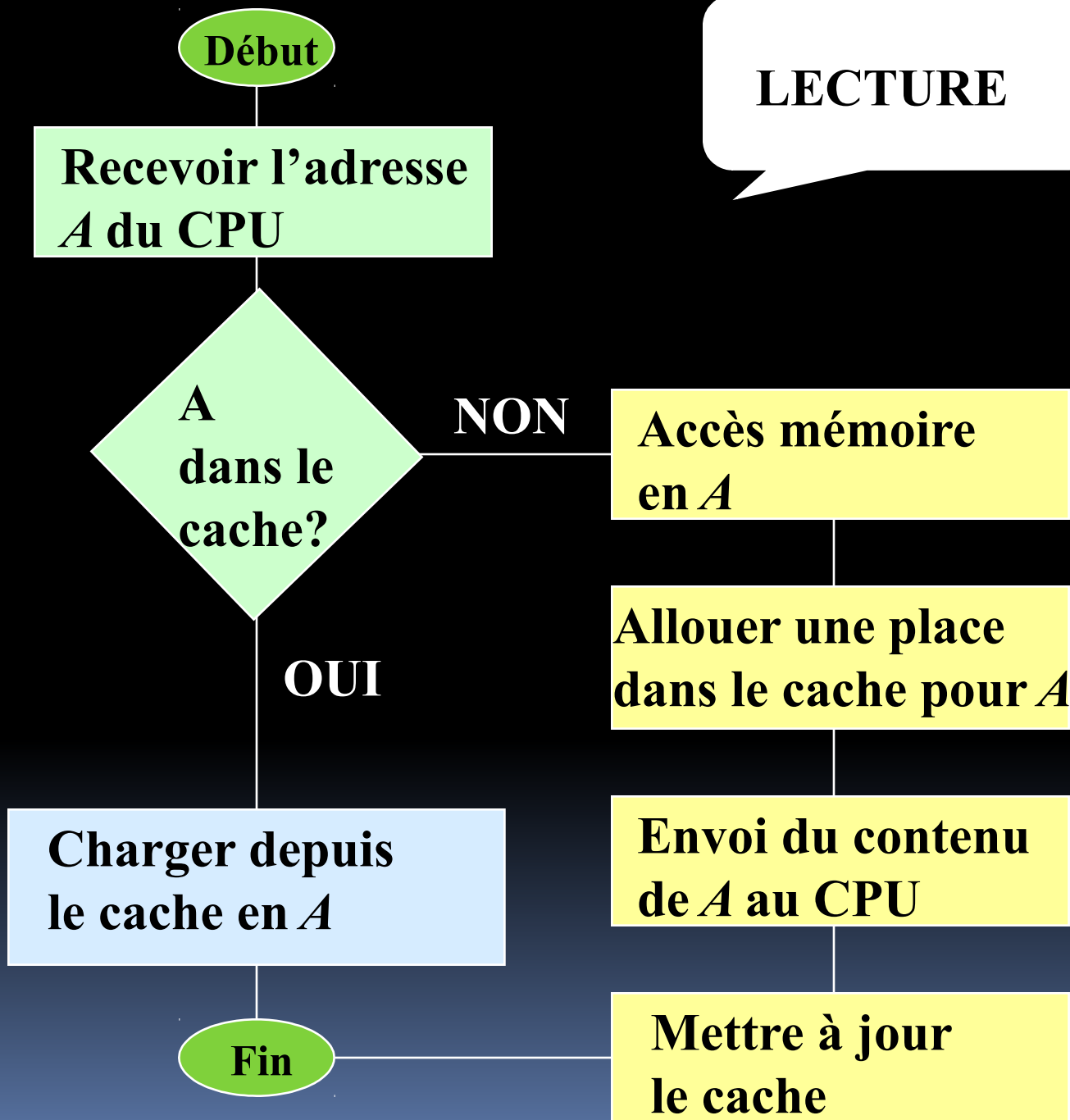
- Coût
 - Plus de cache = plus cher
- Vitesse
 - Plus de cache = plus rapide (jusque un certain point)
 - La recherche des données dans le cache prend du temps

Organisation Typique de Cache



Mise en oeuvre

- Elle est placée entre le processeur et la mémoire (à la Von Neumann).
- On espère que l'information sera souvent présente dans le cache.
 - Principe de la localité
- L'accès est donc globalement plus rapide que dans le modèle VN.
- Même principe que la mémoire virtuelle



Localité

- Localité spatiale: probabilité d'accès à une adresse voisine
- Localité temporelle: probabilité d'accès aux mêmes adresses successivement
- Séquentialité: probabilité d'accès à l'adresse $n + 1$ (localité spatiale)

Utilisation du cache :

- Un "petit" bloc mémoire avec
 $t_{c_mem} = t_{c_cpu}$
- Programme et Data sont transférés via le cache mémoire
- L'efficacité dépend du principe de localité temporelle et spatiale des données et des instructions.

Placement des données

- Mapping des données similaire aux systèmes de mémoire virtuelle
- Implémentation hardware. Améliore également les accès mémoire système
- Plusieurs stratégies:
 - Direct mapping
 - Fully associative mapping
 - Set associative mapping

Direct Mapping

- Chaque donnée dans un seul emplacement du cache
- Calcul simple:

$$i = j \text{ modulo } m$$

i = Numéro de l'emplacement cache

j = Adresse mémoire

m = Nbre d'emplacements du cache

Direct Mapping

- Les bits de poids faible de l'adresse dans le cache et dans la mémoire sont les mêmes .
- Les bits de points fort sont rangés dans le cache.
- 2 zones sur l'adresse

TAG

Index

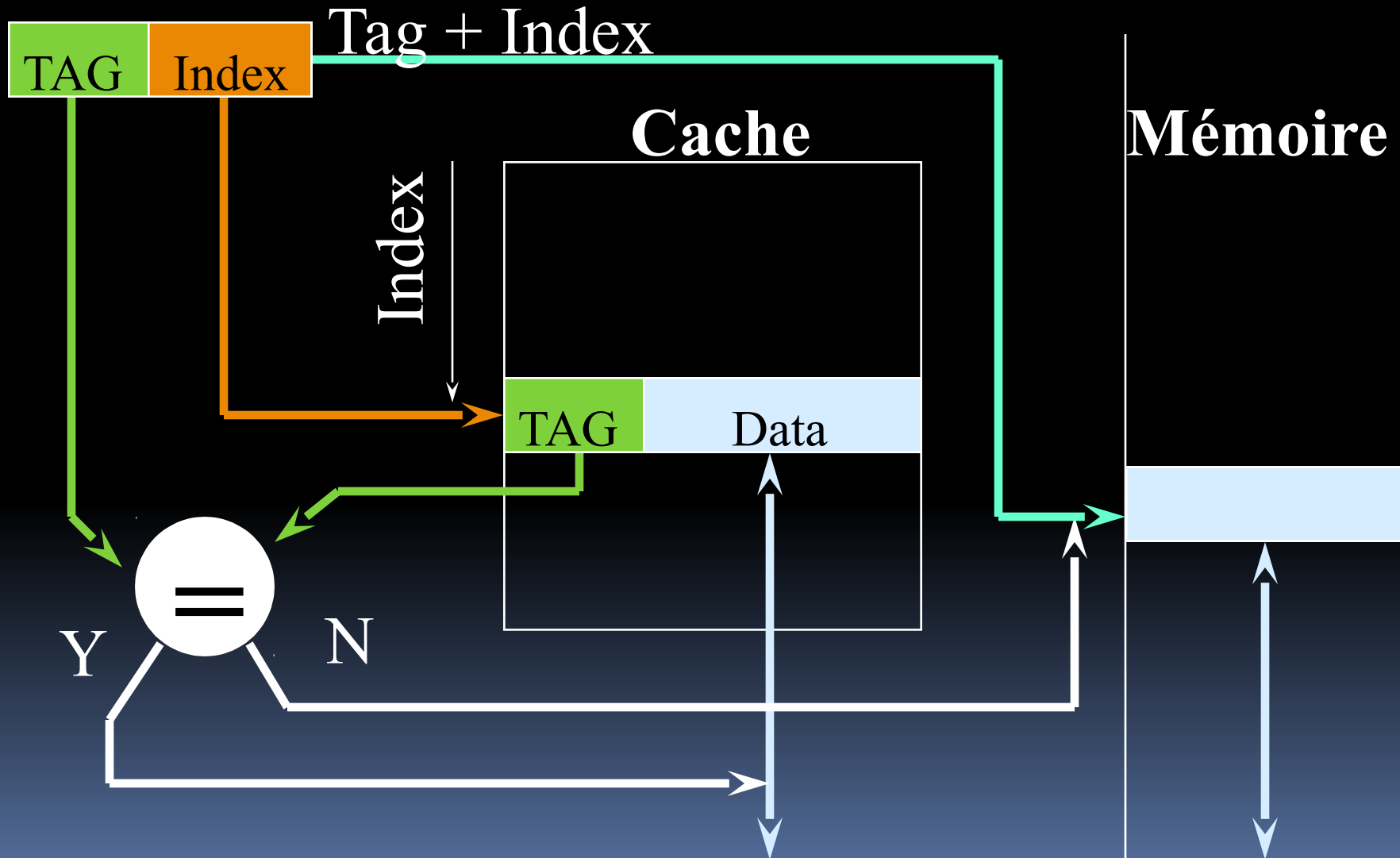
Méthode

- L'index permet l'accès à 1 emplacement du cache. Cet emplacement contient 1TAG + 1DATA
- Le TAG obtenu est comparé avec le TAG de l'adresse
- Les 2 TAG sont
 - les mêmes => Succès
 - différents => Echec

Accès mémoire

- En cas d'échec, l'adresse est envoyée à la mémoire.
 - Lecture : Le mot est envoyé au cache, il peut être envoyé en simultané au Processeur (Read-through)
 - Ecriture : Le mot est écrit dans le cache, il peut être écrit en même temps dans la mémoire (Write-through)

Direct mapping



Direct mapping par bloc

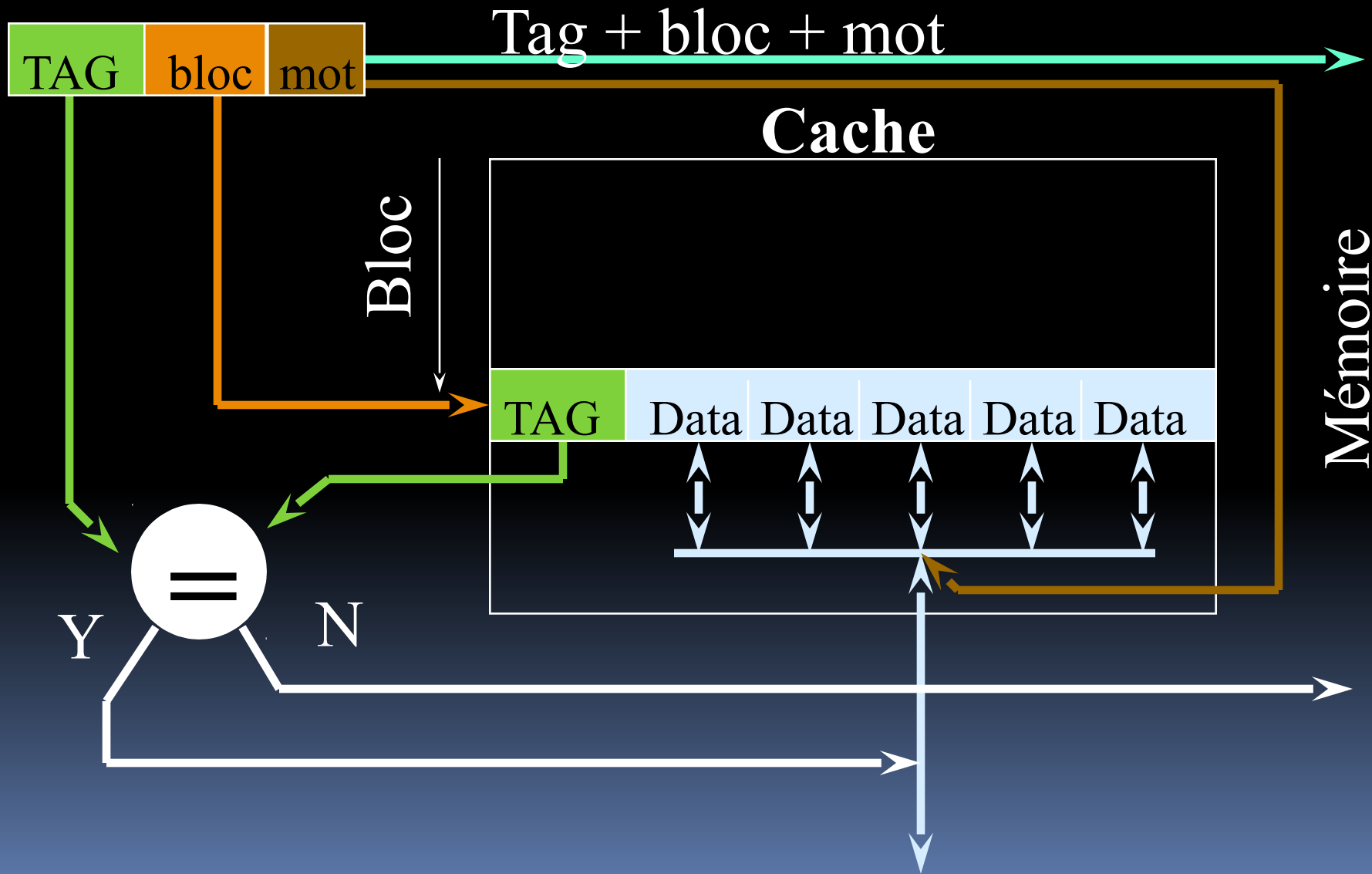
- On transfère un bloc d'adresses contiguës (= > utilisation des mémoires entrelacées)
 - plusieurs requêtes mémoire simultanées
- L'adresse est composée de 3 champs



Accès mémoire

- En cas d'échec, on transfère toujours le bloc (la ligne) qui contient le mot référencé.
 - Seul 1 bloc (ligne) du même numéro peut être présent dans le cache
 - Rangement des données qui élimine les conflits.

Direct mapping par bloc



Avantages / Inconvénients

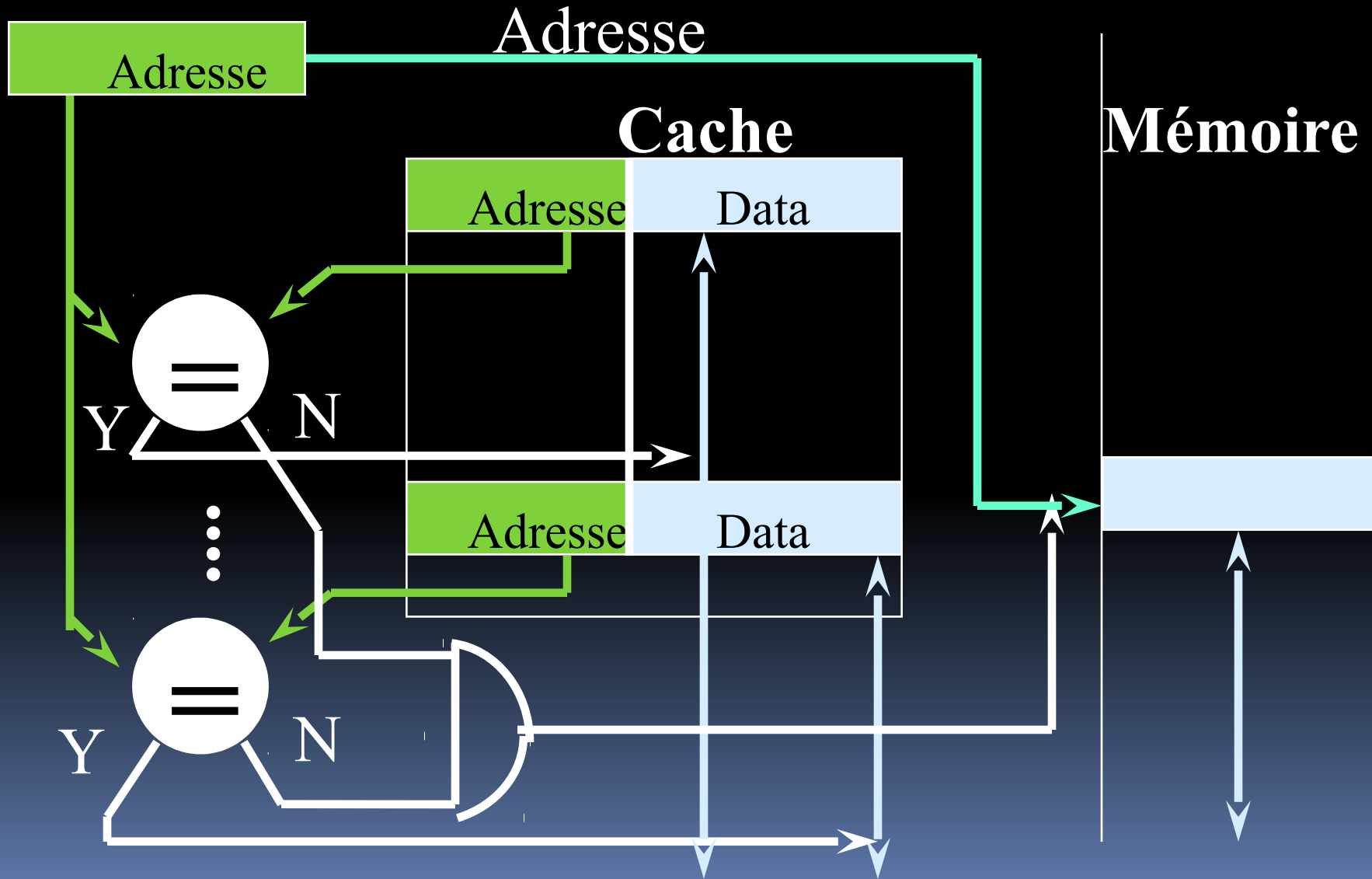
- Pas d'algo remplacement
- Hard simple et peu coûteux
- Rapide
- Ratio de succès faible (cf. associative)
- Performances décroissent si même Index

◆ **Tendance : Direct Mapping adapté aux mémoires caches plus grandes**

Fully associative mapping

- Mémoire associative: L'adresse est simultanément comparée à toutes les adresses rangées dans le cache.
- Par bloc ou par mot (cf. Direct Mapping)
- Un bit de validité associé à chaque entrée
- Algorithme de remplacement hardware

Fully associative mapping



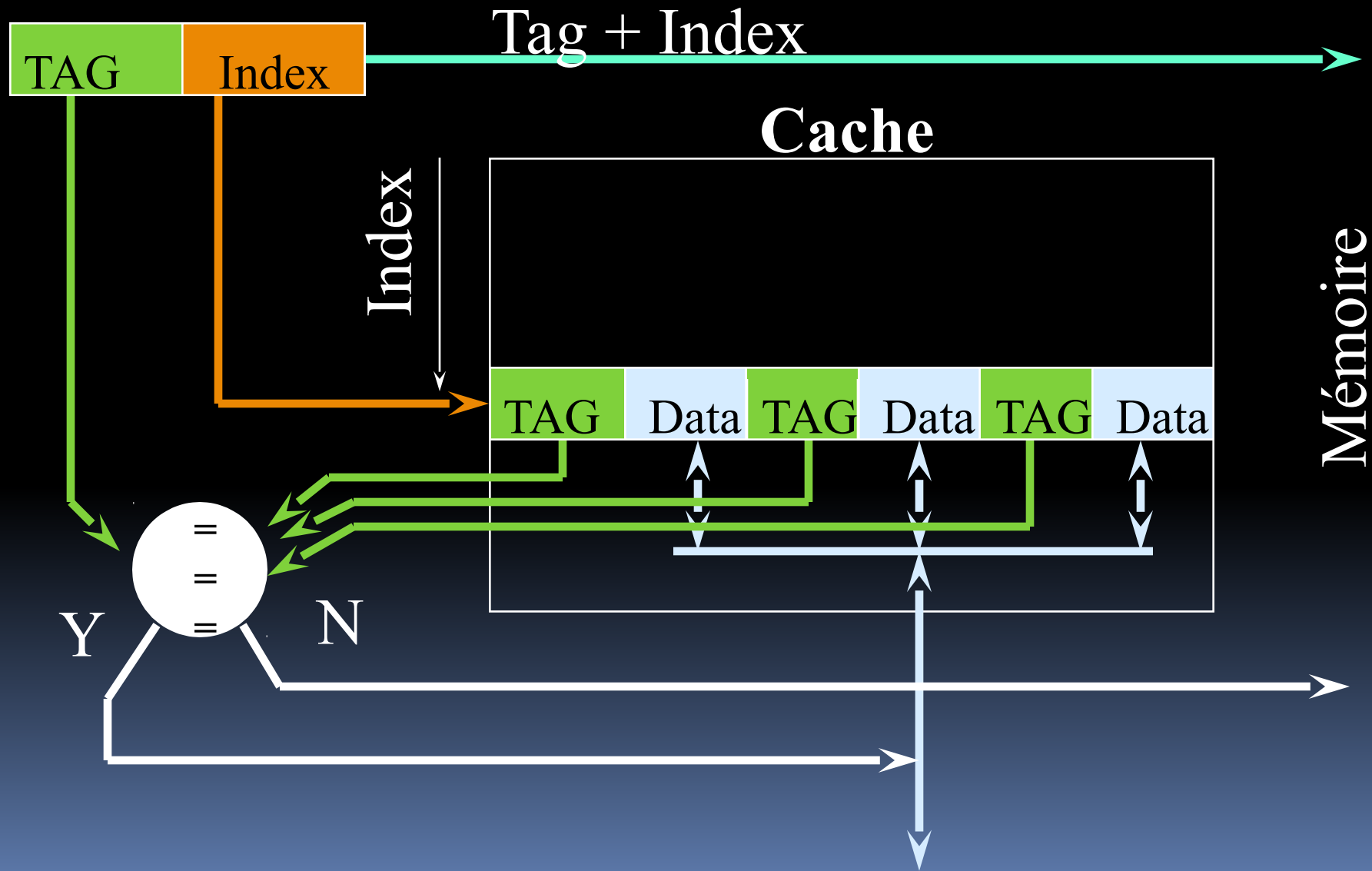
Avantages / Inconvénients

- Flexibilité
- Plusieurs lignes dans le cache avec le même numéro.
- Coût hardware du comparateur
- Algo de remplacement (lequel enlever ?)
- Taille réduite

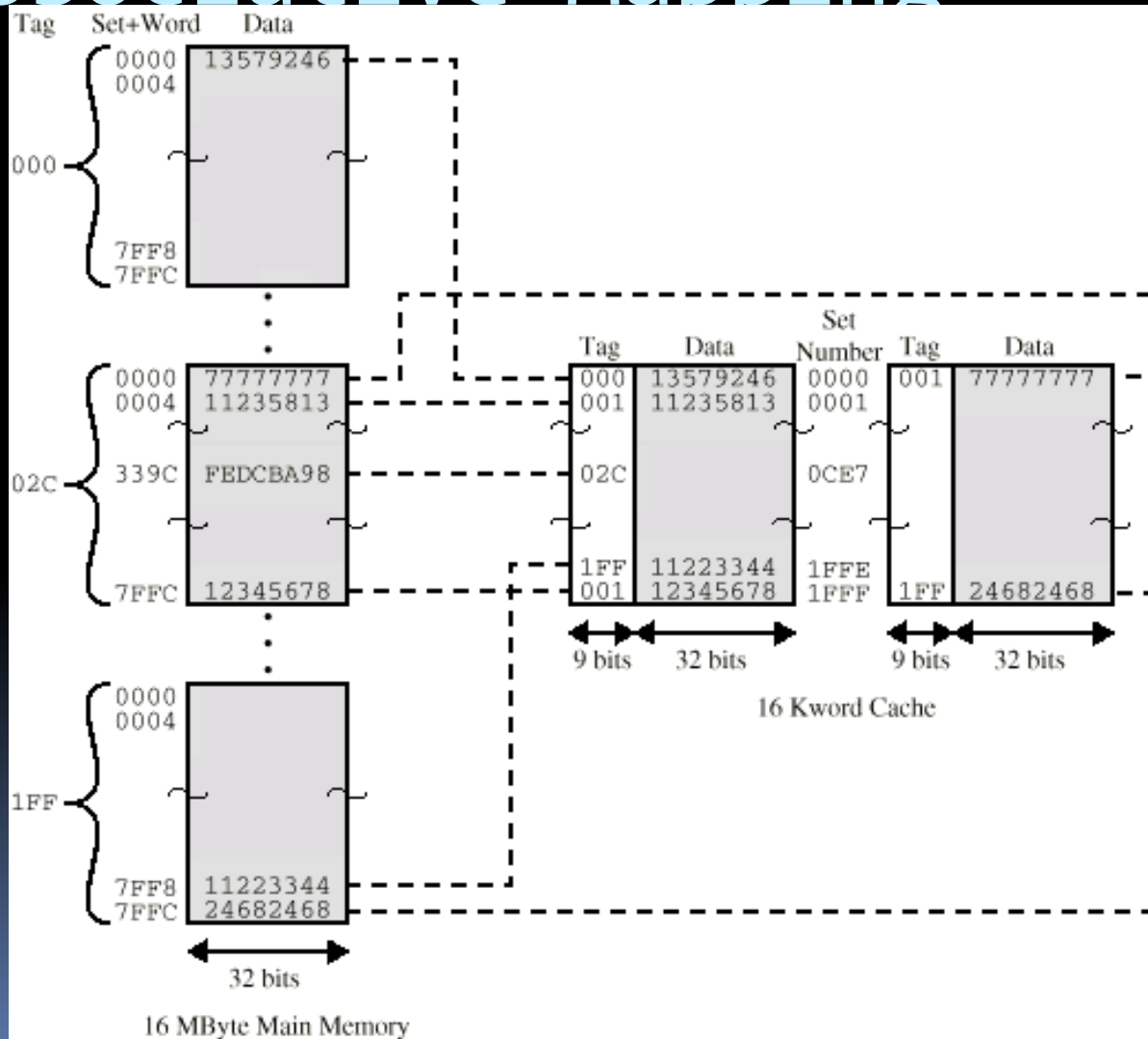
Set associative mapping

- Compromis entre direct et full associative mapping
- Permet un nombre limité de blocs (lignes) avec le même index mais des TAG différents.
- Le cache est divisé en "set" de blocs. Chaque "set" contient le même nombre de blocs.
- Chaque bloc a son propre TAG qui associé à l'index, identifie le bloc.

Set associative mapping



Exemple Two Way Set Associative Mapping




Avantages / Inconvénients

- Moins de comparateurs.
- Algo de remplacement seulement sur le set
- Plus courant avec 2 blocs par ligne



Stratégies de chargement

- Trois stratégies de chargement de donnée ou bloc de données de la mémoire vers le cache
 - A la demande
 - Pré chargement
 - Chargement sélectif
- 

1) A la demande

- On charge lorsqu'il y a eu un échec
 - Bloc non présent dans le cache
- La plus simple
- Pas de hardware

2) Pré chargement

- On charge avant la référence au bloc.
- Exemple : charge le bloc $(i+1)$ quand le bloc (i) est référencé pour la 1^{ère} fois.
 - 50% d'échec en moins sur un cache assez grand.
 - Sur un cache petit, on peut écraser un autre bloc qui pourrait être encore référencé.

3) Chargement sélectif

- Seuls certains blocs seront chargés en cache
- Certains blocs restent toujours en mémoire
- Accès direct à la mémoire reste possible
 - Exemple : des données communes en Multiprocesseurs sont plus facilement gérées en mémoire commune.

Opérations d'écriture


- Consistance entre le cache et la mémoire
 - les 2 versions de la donnée peuvent être différentes.
- Nécessité de conserver la cohérence dans des systèmes multi processeurs (mémoire partagée et caches multiples) ou avec des I/O sur la mémoire...

Write-through

- Chaque écriture dans le cache est répétée sur la mémoire.
- Le cache est plus efficace alors en lecture qu'en écriture
- En moyenne (Smith 1982) 3 à 10 lectures entre 2 écritures.



Write-back

- L'écriture n'est réalisée que lors du remplacement de bloc (I/O passent par le cache)
 - Simple write-back:
tous les blocs remplacés sont écrits
- 



Write Back (2)


- Ecriture seulement des blocs modifiés: 1 bit de Flag par bloc

Algorithme de remplacement

- Problème : Amener un nouveau bloc quand le cache est plein !
- Direct Mapping => pas de choix possible (le même index).
- Reprise des algorithmes des systèmes de mémoire virtuelle
- Implémentation hardware
- En même temps que le chargement du bloc depuis la mémoire



Aléatoire

- Méthode la plus simple
 - On choisit un bloc au hasard.
 - Approximation du hasard
 - Compteur incrémenté en fonction de l'horloge
 - La valeur du compteur donne le numéro du bloc à remplacer
- 

FIFO

- On écrase le bloc le plus ancien dans le cache.
- Implémentation
 - FIFO d'adresses
 - Compteur incrémental
 - Fully associative : 1 seul compteur
 - Set associative: 1 compteur par set

LRU (Least Recently Used)

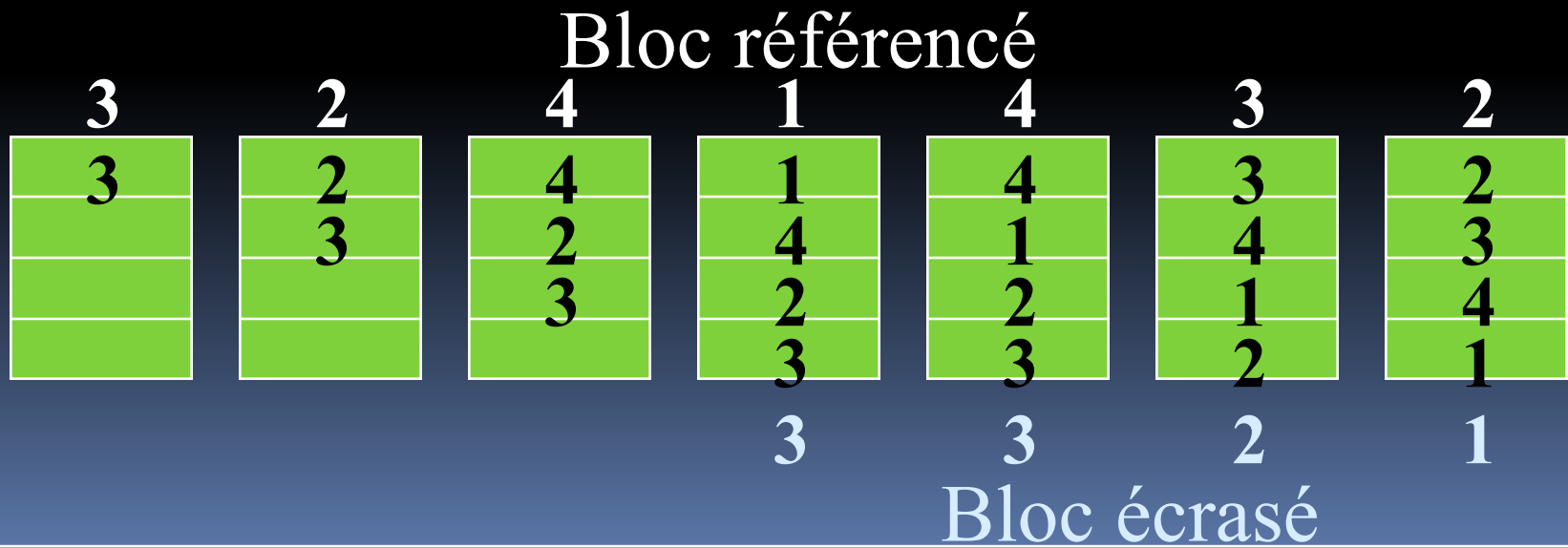
- On écrase le bloc le moins récemment utilisé .
- Implémentation
 - Par compteur
 - Par pile de registres
 - Par matrice de références
 - Par méthodes approximatives

LRU par compteur

- 1 Compteur par bloc
- Incrément régulier du compteur
- A chaque référence on recopie le compteur dans le bloc concerné
- On choisit le plus vieux en fonction de la valeur des compteurs

LRU par pile de registres

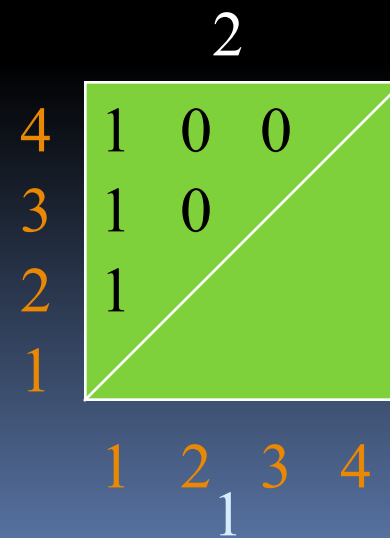
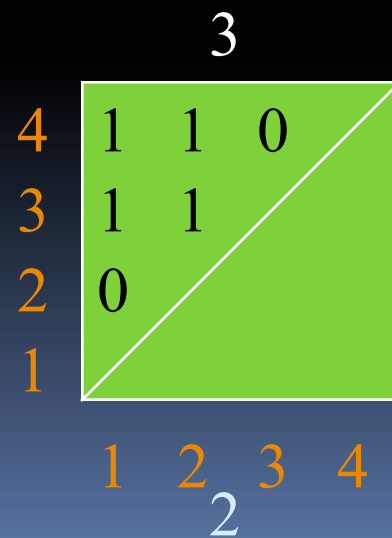
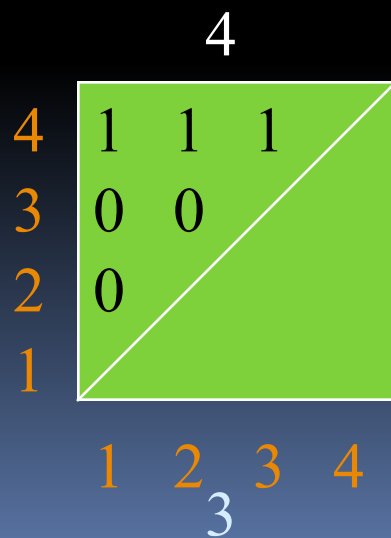
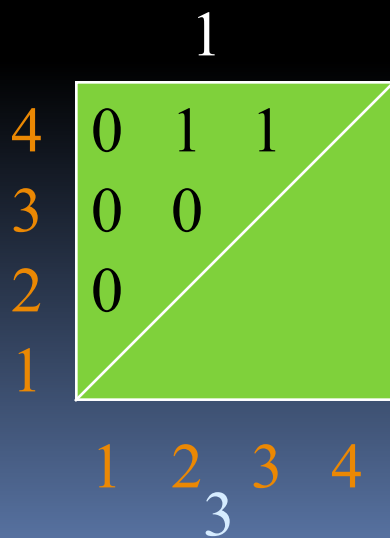
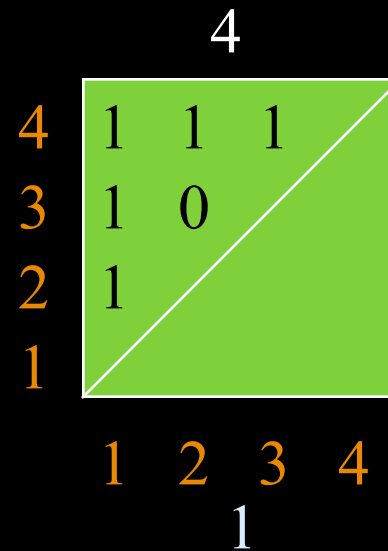
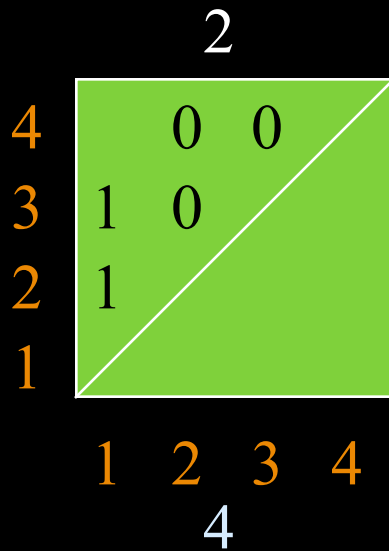
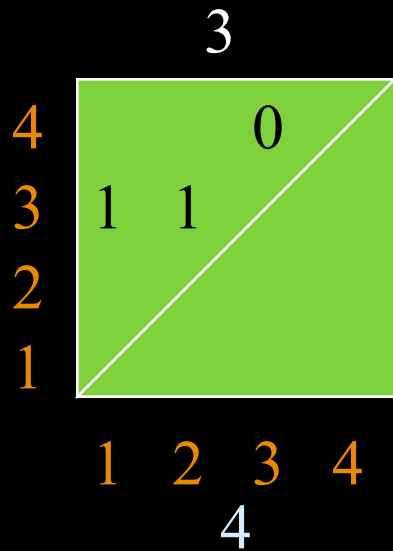
- Pile de registres (par set) :
- On place en sommet de pile le numéro du bloc référencé
- nb de registre = nb bloc par set



LRU par matrice de références

- Matrice de bit
- Solution avec une matrice triangulaire
- B Blocs => matrice $B \times B$ (sans diagonale)
- Une référence au bloc l
 - Les bits de la ligne l sont mis à 1
 - Les bits de la colonne l sont mis à 0
- Le bloc LRU a des 0 sur sa ligne et des 1 sur sa colonne

Matrice de références

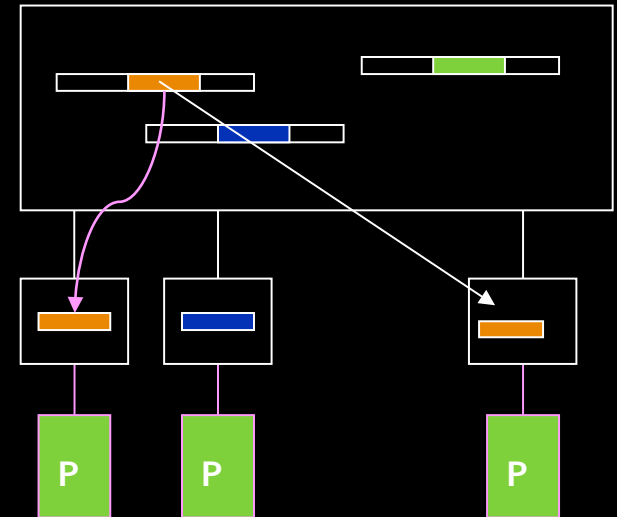


LRU et méthodes approximatives

- Un grand nombre de blocs impose des méthodes approximatives
- Résolution par sous groupe
- Choix d'une solution proche du LRU
- Exemple sur le 80486 pseudo LRU

Cache et multiprocesseurs


- Réduction de la latence
 - Duplication proche du processeur
- Réduction du bandwidth
- Plusieurs processeurs partagent des data efficacement



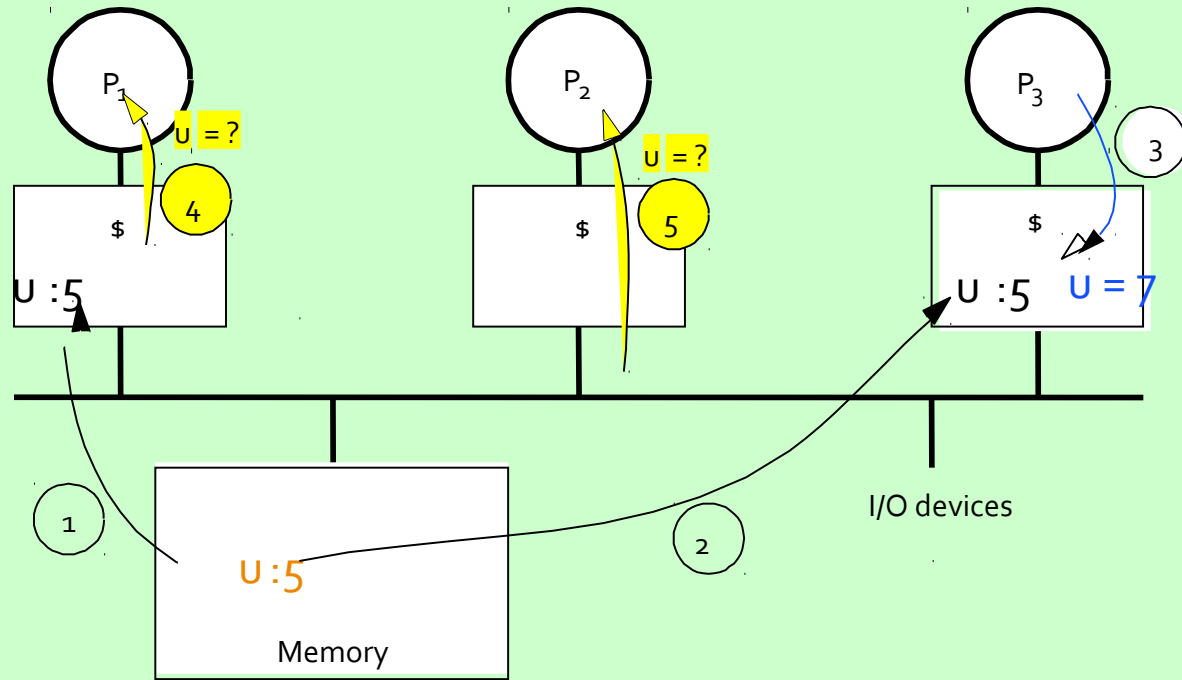
• **store & load exécutés sur différents processeurs?**



Fonctionnement

- Pas de problème en lecture
 - En écriture, il faut mettre à jour toutes les copies de la donnée
 - Le Write-through n'est pas suffisant : il ne met pas à jour les copies des autres caches
- 

Exemple Cohérence de Cache



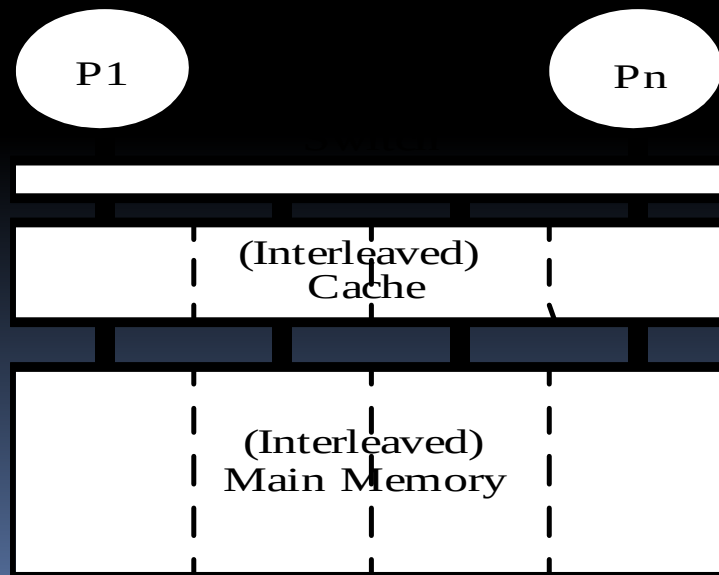
- ❑ Processeurs ont différentes valeurs de u après (3)
- ❑ Inacceptable et fréquent!

Différentes techniques

- Plusieurs techniques pour garder la cohérence des caches
 - Cache partagé
 - Objet "non-caché"
 - Surveillance de Bus
 - Ecriture diffusion: Broadcast write
 - Méthode de la directrice
 - MESI

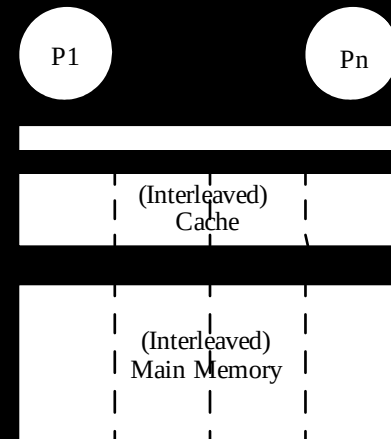
Cache partagé

- Alliant FX-8
 - Début des 80's
 - 8 x 68020s avec x-bar et 512 KB interleaved cache
- Bientôt sur nos microprocesseurs...



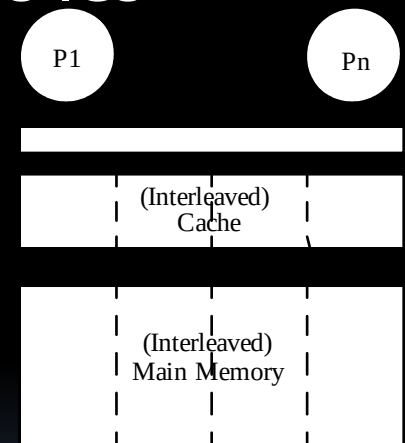
Avantages

- Placement dans le cache standard
 - Une seule copie existe
- Partage à grain fin
- Interférence positive
 - 1 proc peut prefetcher data pour un autre
- Taille mémoire cache totale plus petite.
- Partage d'une ligne de data sans effet "ping-pong"
 - Faux partage



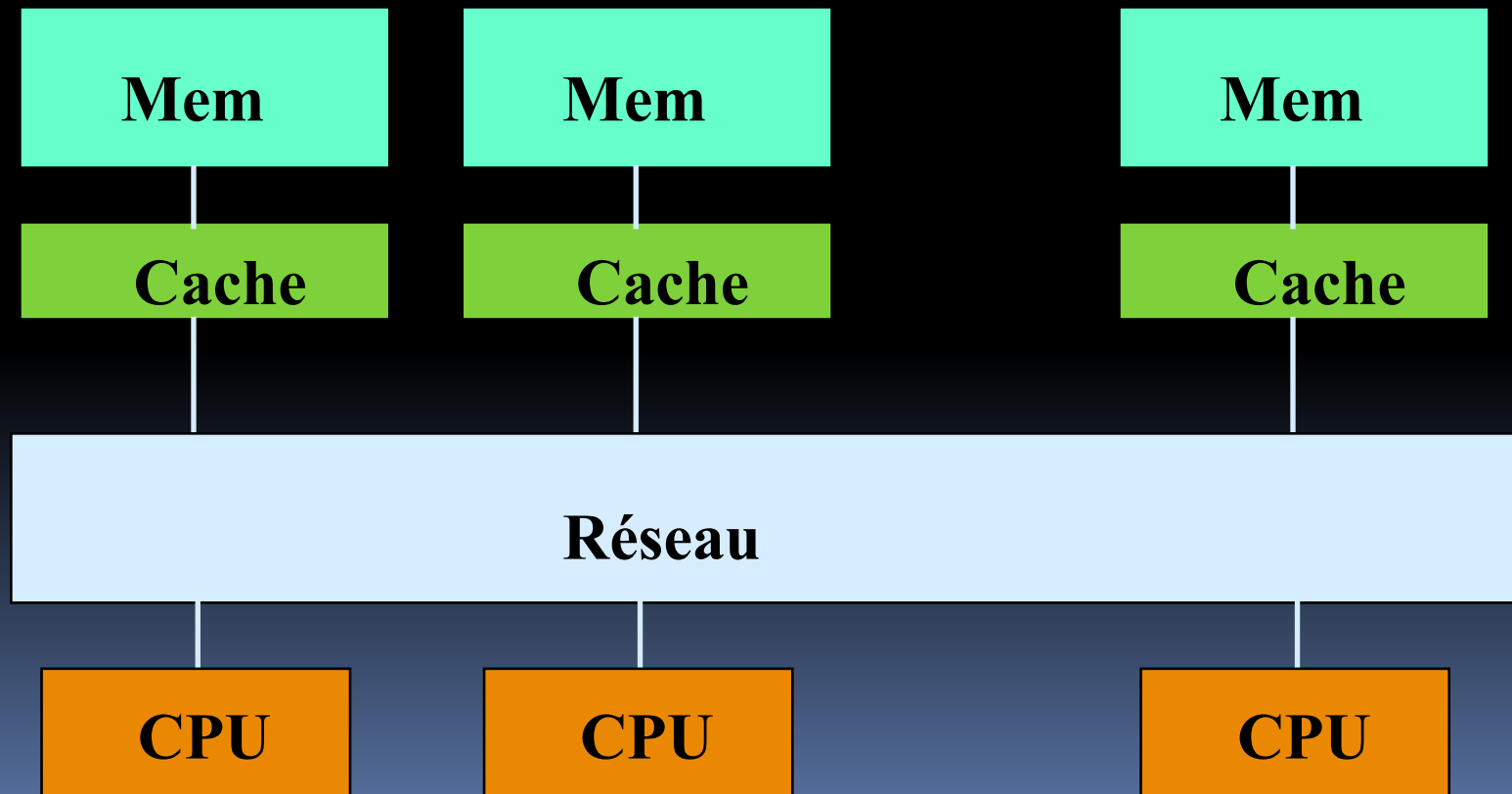
Désavantages

- Limitation fondamentale du BW
- Augmentation de la latence pour tous les accès
 - X-bar
 - Cache plus grand
- Interférence négative
 - 1 proc peut écraser les data utiles pour un autre
- Beaucoup de caches L2 sont partagés



Mémoire multi-bancs

- Ne réduit pas les conflits d'accès au réseau.



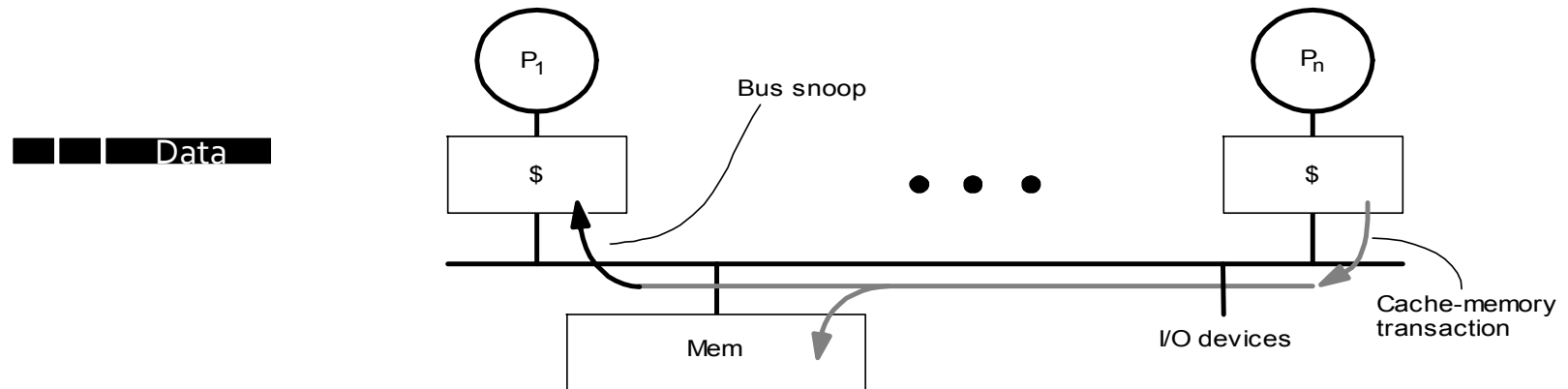
Objets non cachés

- Les objets partageables accédés en écriture sont présents en mémoire
- Ils ne peuvent pas venir dans le cache
- Contrôle software:
 - sémaphores.
 - sections critiques.

Surveillance de bus

- Multiprocesseurs autour d'un bus.
- 1 contrôleur de cache / CPU :
 - détecte les écritures sur le Bus .
 - Si l'adresse \in au cache, on invalide l'entrée.
- On utilise nécessairement le Write-Through ou le write back

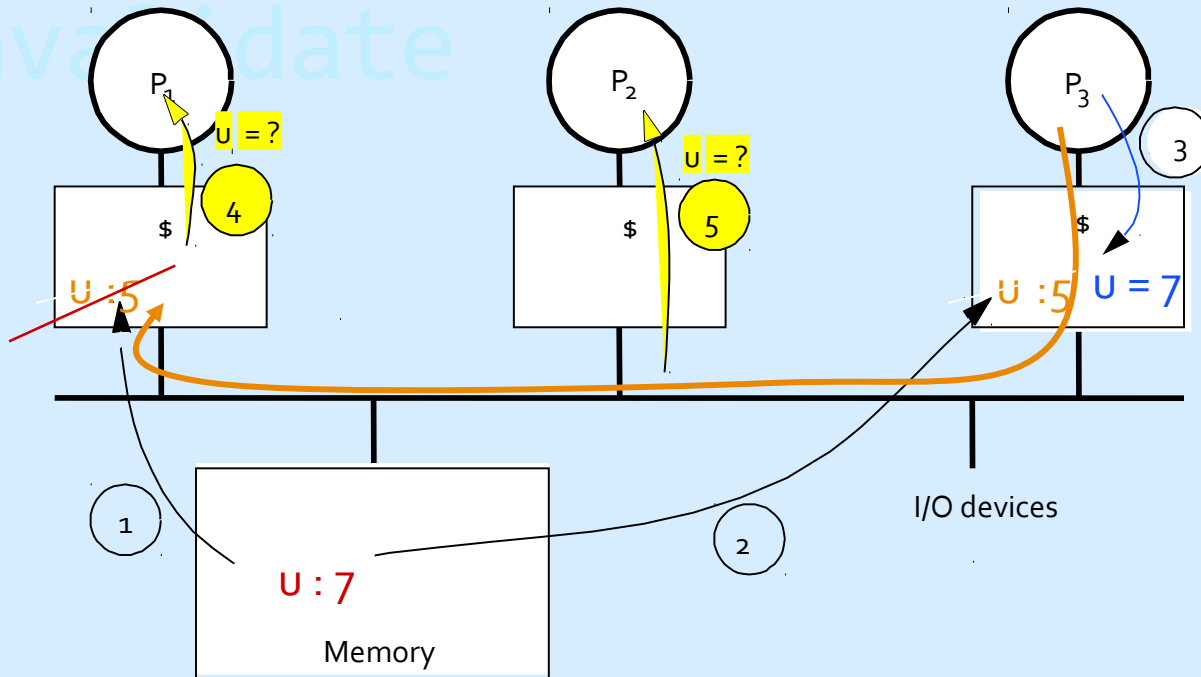
Protocole Snoopy



- Contrôleur de cache “snoops” toutes les transactions sur le bus
 - action pour assurer la cohérence
 - invalide, update, ou suppression de la valeur
 - dépend de l'état du bloc et du protocole de cohérence

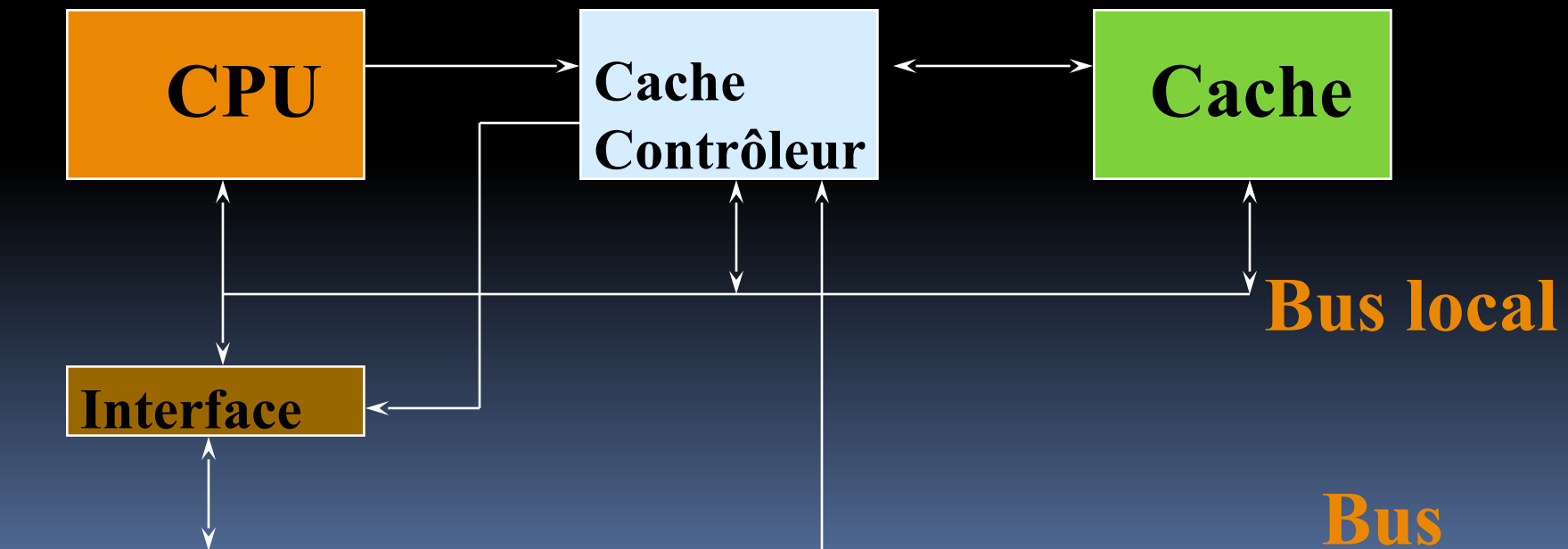
Example: Write-thru

Invalidate



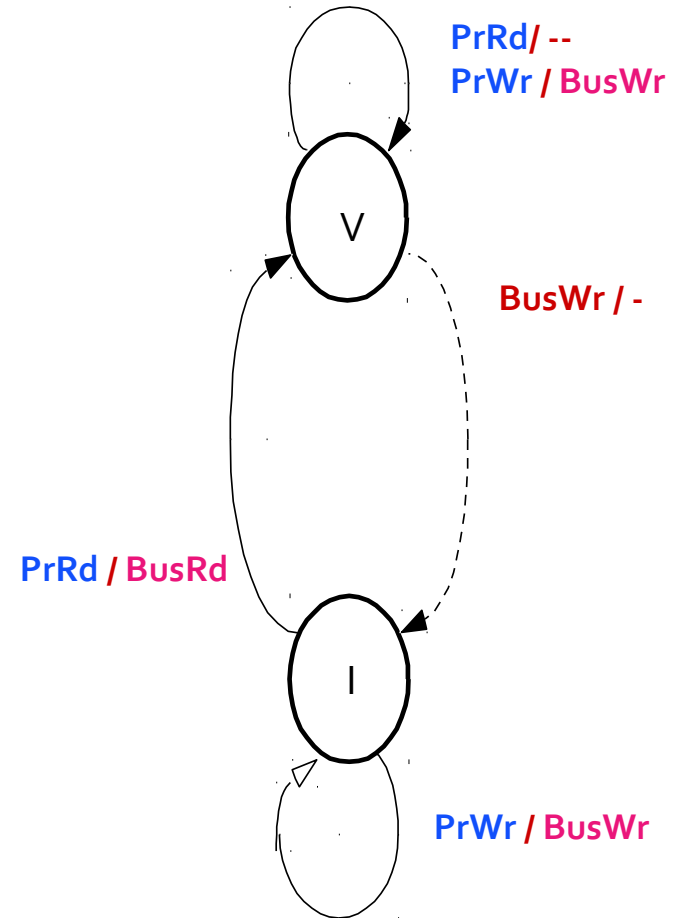
Fonctionnement du contrôleur

- 2 stratégies
 - comparer le TAG de l'index
 - ou invalider l'index \forall TAG



Protocole Write-through Invalidate

- Write through, write no-allocate caches
- 2 états par bloc dans chaque cache
 - bits hardware associés aux blocs présents dans le cache
 - Autres blocs Invalide (non présent)
- Write invalide tous les autres caches
 - Lectures multiples simultanées mais write invalide les copies
 - Ordre des écritures est celui du Bus

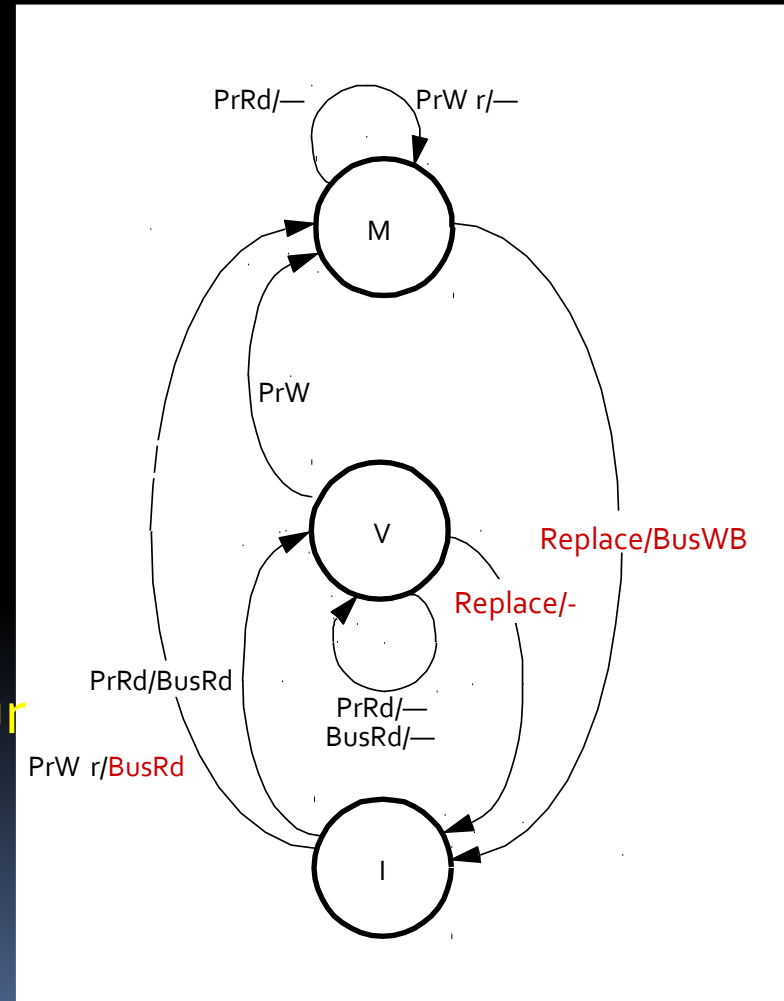


Write-through vs. Write-back

- Write-through est simple
 - Tous les write sont observables
 - Tous les write vont sur le bus
 - Utilise beaucoup de bandwidth
- Write-back absorbe la plupart des write par des succès aux caches
 - Pas d'accès sur le Bus
- Comment assurer la cohérence?
 - Protocole plus sophistiqué!

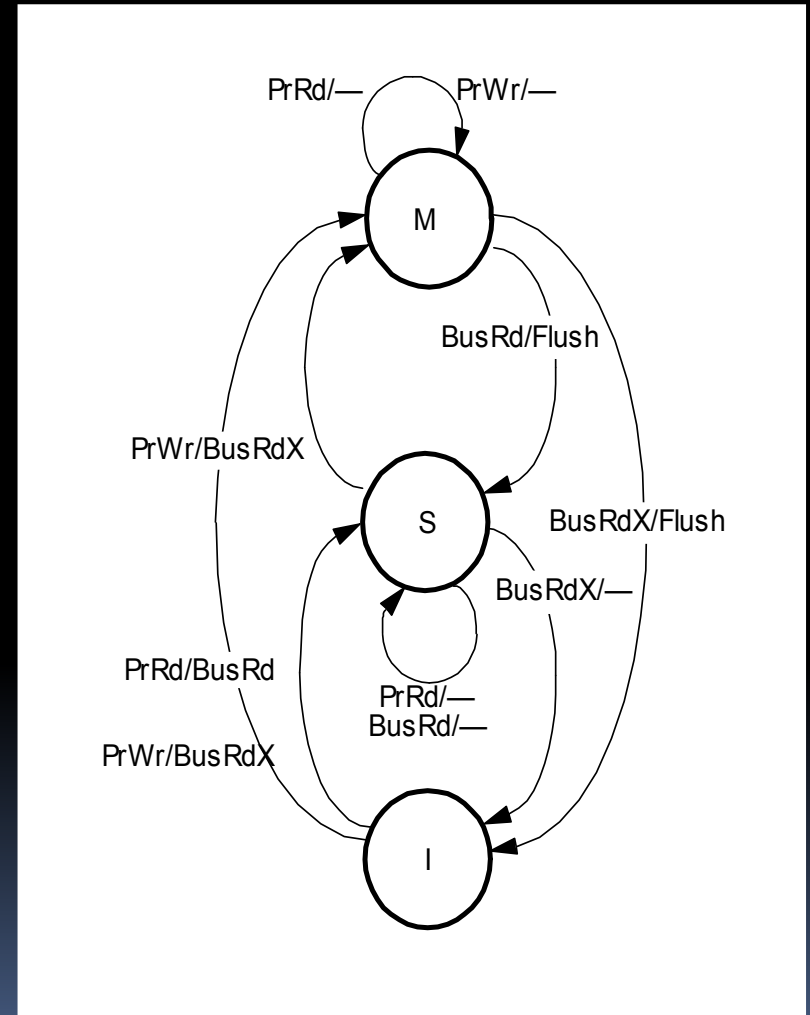
Write-back monoprocesseur

- 2 opérations processeur
 - PrRd, PrWr
- 3 états
 - invalid, valid (clean), modified (dirty)
- 2 transactions bus :
 - read (BusRd), write-back (BusWB)
 - Valid = "shared"
 - Modified = "exclusive"
- Introduire 1 nouvelle transaction sur le Bus en multiproc
 - read-exclusive: read pour modifier

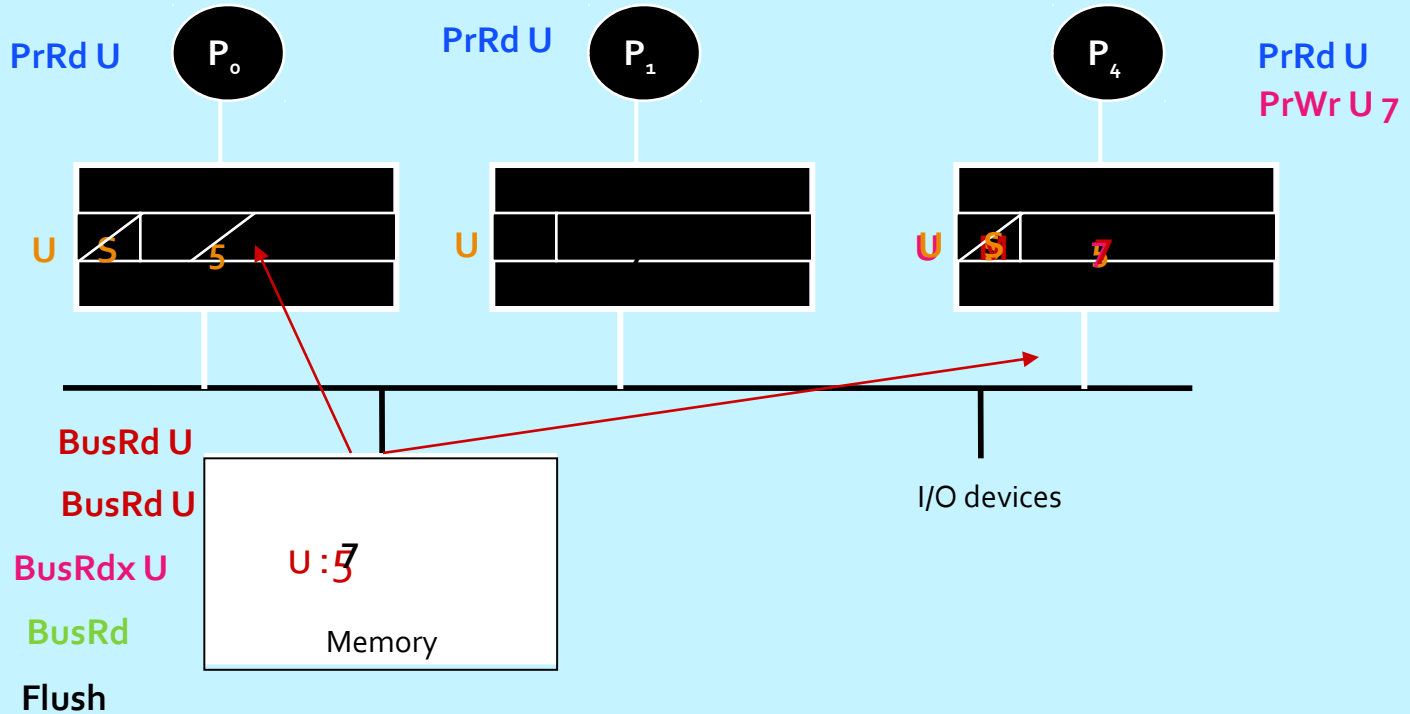


Protocole MSI Invalidate

- Read entraîne "shared"
- Devenir propriétaire exclusif avant le Write
 - BusRdx produit une invalidation des autres
 - Si bloc modifier dans un autre cache, on récupère (1 seul en M)
- Remplacement?
 - S->I, M->I



Example: Write-Back Protocol



Protocole trop coûteux!

- Problème: 1 Read&Write produit deux actions sur le bus, même sans partage.
- BusRd (I->S) suivi BusRdX (S->M)
 - Pas très bon pour un monoprocesseur ☹️

Protocole MESI (4-state)


- Ajoute l'état *exclusive*
 - distingue exclusive (writable) et owned (written)
- Etat
 - invalid
 - exclusive ou *exclusive-clean* (seul ce cache a une copie, mais pas modifiée)
 - shared (2 caches au moins ont une copie)
 - modified (dirty)
- I -> E sur PrRd si aucun cache n'a de copie

Mémoire entrelacée

- Réduire la différence entre vitesse de la mémoire et vitesse du processeur
- Augmenter le débit de la mémoire de telle sorte que plusieurs mots soient accédés en même temps
- Equivalence entre débit mémoire, débit bus et débit CPU




Construction

- La mémoire est construite avec plusieurs modules.
 - Chaque module travaille de façon indépendante.
 - Ces modules sont connectés à un ou des bus ou un réseau de connexion.
- 



Fonctionnement

- Lorsqu'une adresse est présentée à un module, il faut un cycle mémoire pour réaliser la lecture/écriture.
 - Idée: Présenter différentes adresses à différents modules en même temps.
- 



Organisation

- La mémoire supporte un adressage linéaire
- L'organisation de la mémoire dépend de la projection de cet adressage sur la mémoire physique

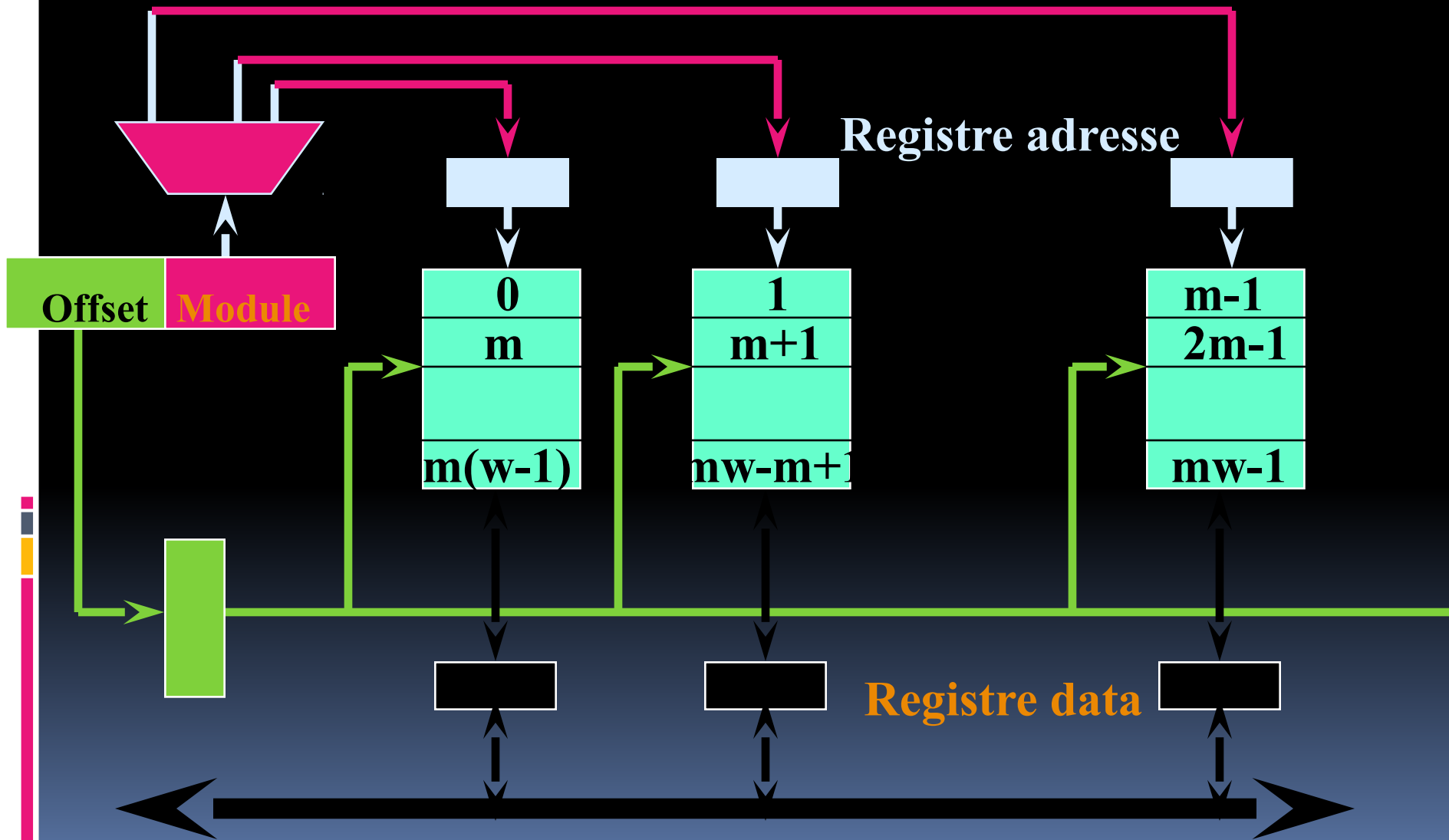
Accès par bloc

- En plus de l'accès aléatoire, on utilise souvent un accès par bloc. (Bloc d'adresses consécutives)(cf. cache).
- On peut prendre en compte cette contrainte pour améliorer le débit de la mémoire lors d'accès à des blocs.

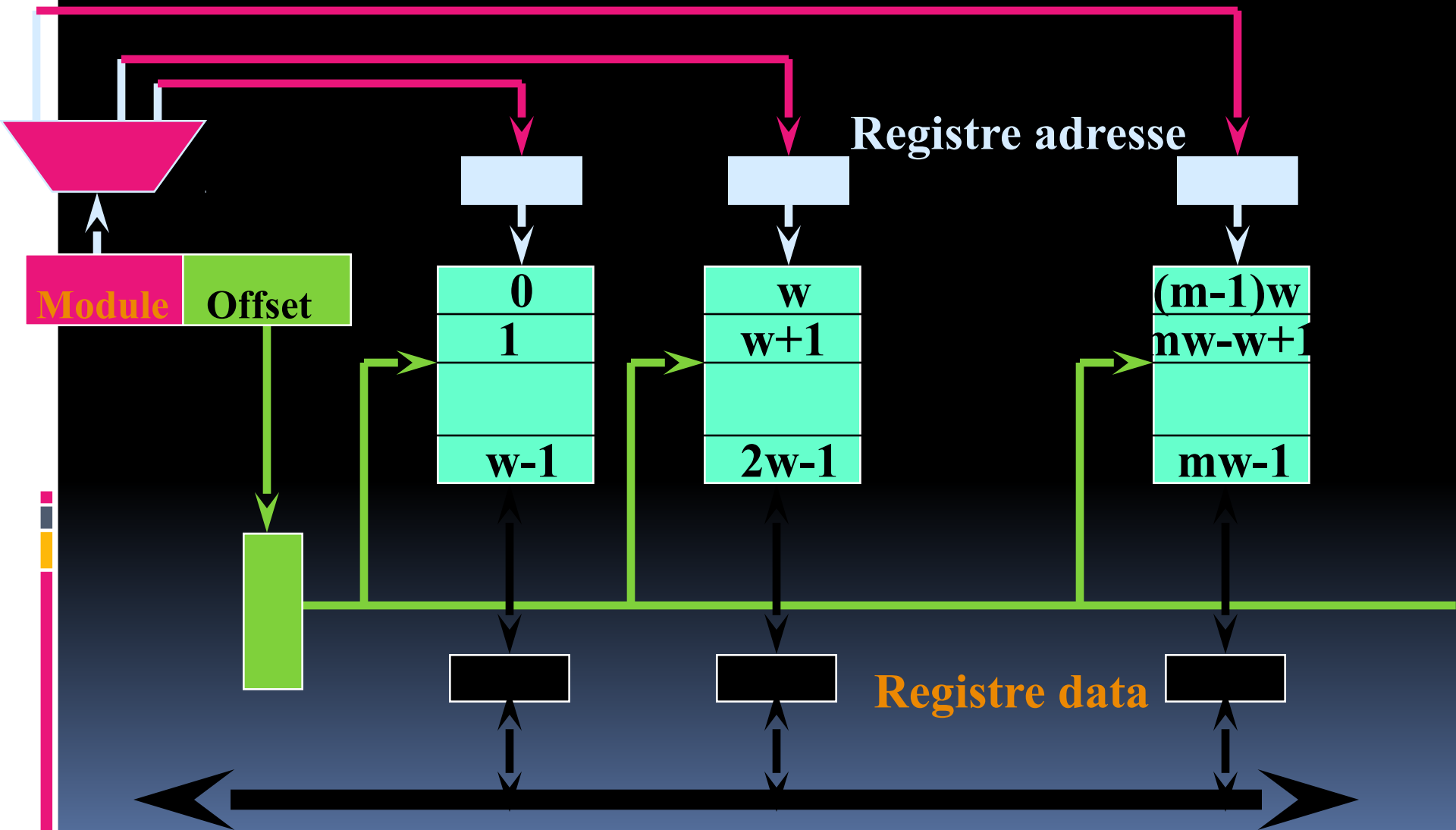
Adressage

- ◆ **Deux formats d'adressage existent**
 - **Low order interleaving** : les bits de poids faible référencent le module, les bits de poids fort donnent le déplacement dans le module.
 - **high order interleaving** : les bits de poids fort référencent le module, les bits de poids faible donnent le déplacement dans le module.

Low-order interleaving




High-order interleaving





Parallèle/pipeline

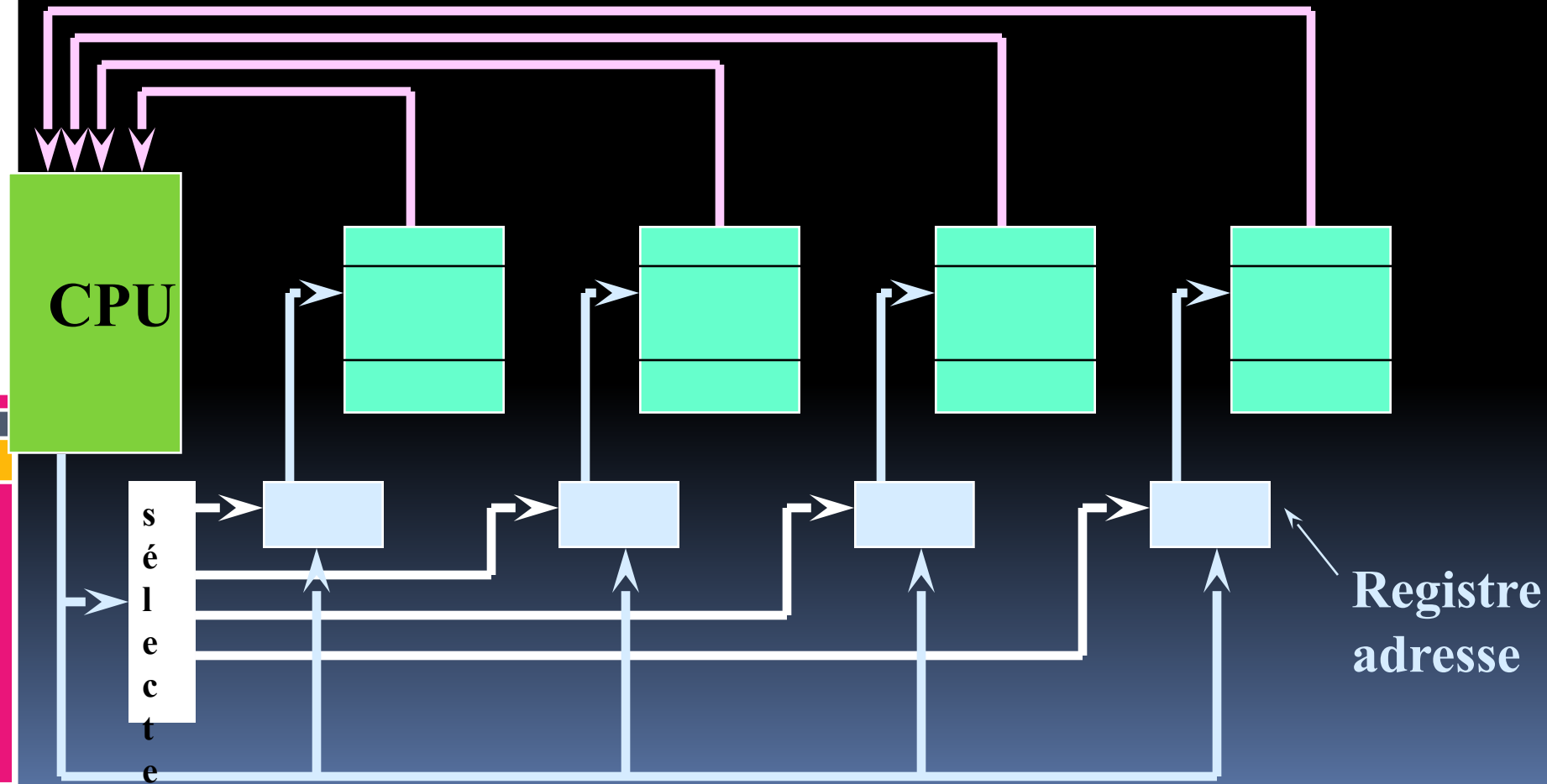
- Les accès à des données multiples peuvent alors être
 - parallélisés : tous les bancs démarrent en même temps
 - pipelinés. Les bancs sont activés successivement
- 

Accès parallèle

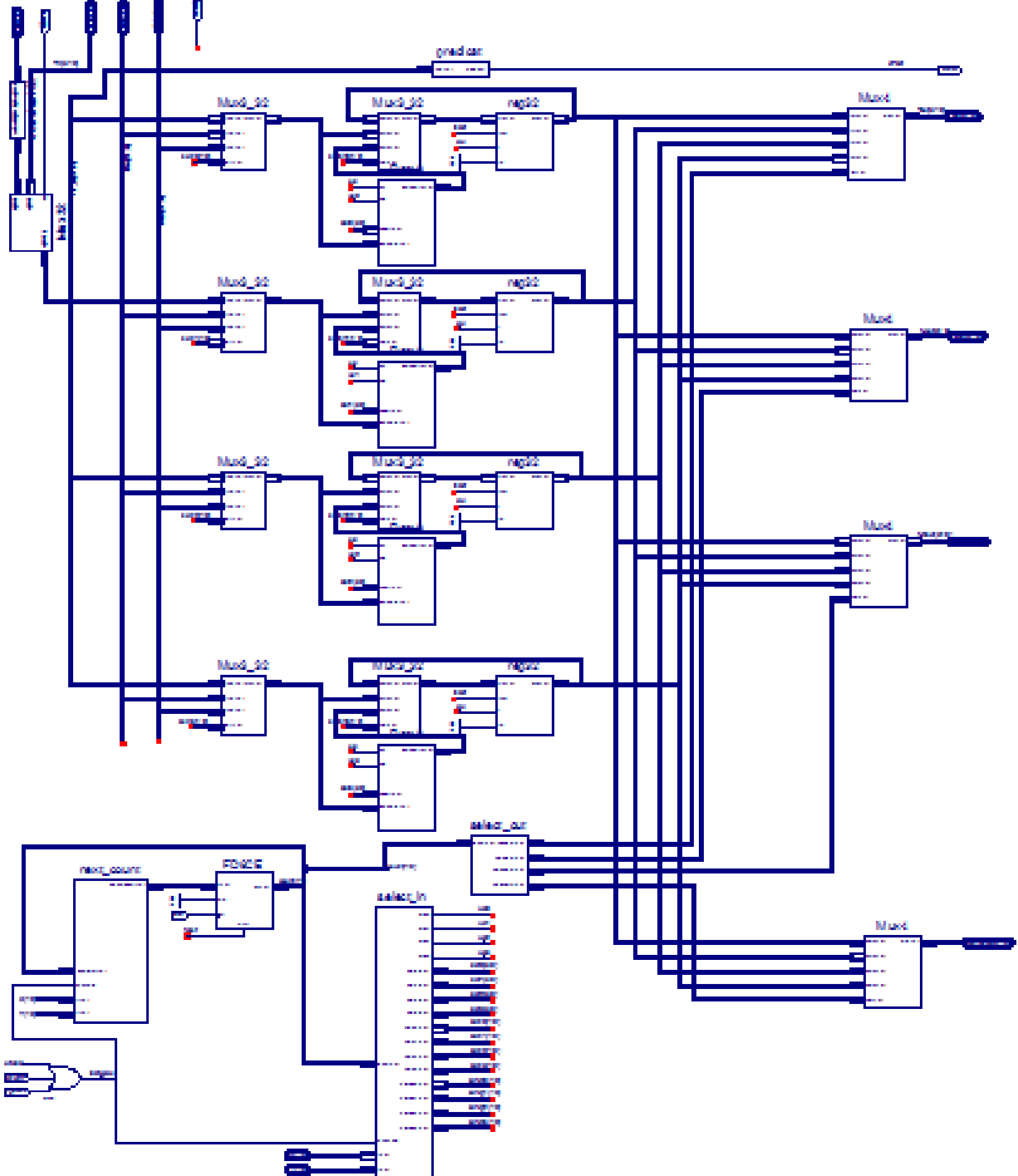
- La largeur du bus de donnée est égale au nombre de bancs qui travaillent en parallèle
- Les bancs travaillent
 - soit sur la même adresse,
 - soit sur des adresses différentes

Accès parallèle

Bus data extra Large



Homade

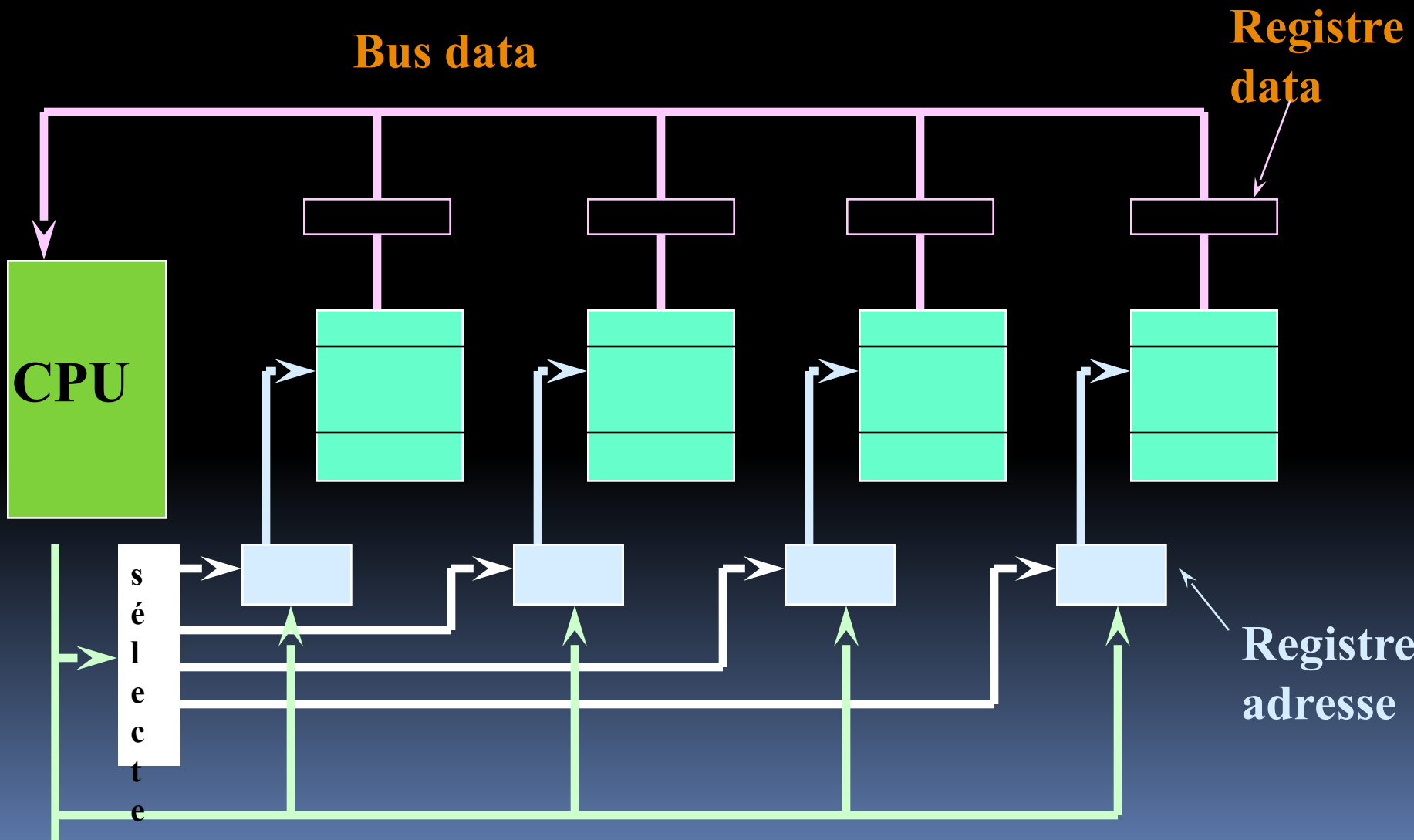




Accès pipeline

- Les adresses sont envoyées à chaque cycle vers un banc différent
- Une adresse différente peut être produite à chaque cycle (vitesse du CPU)
- La même adresse à tous les bancs en un seul cycle

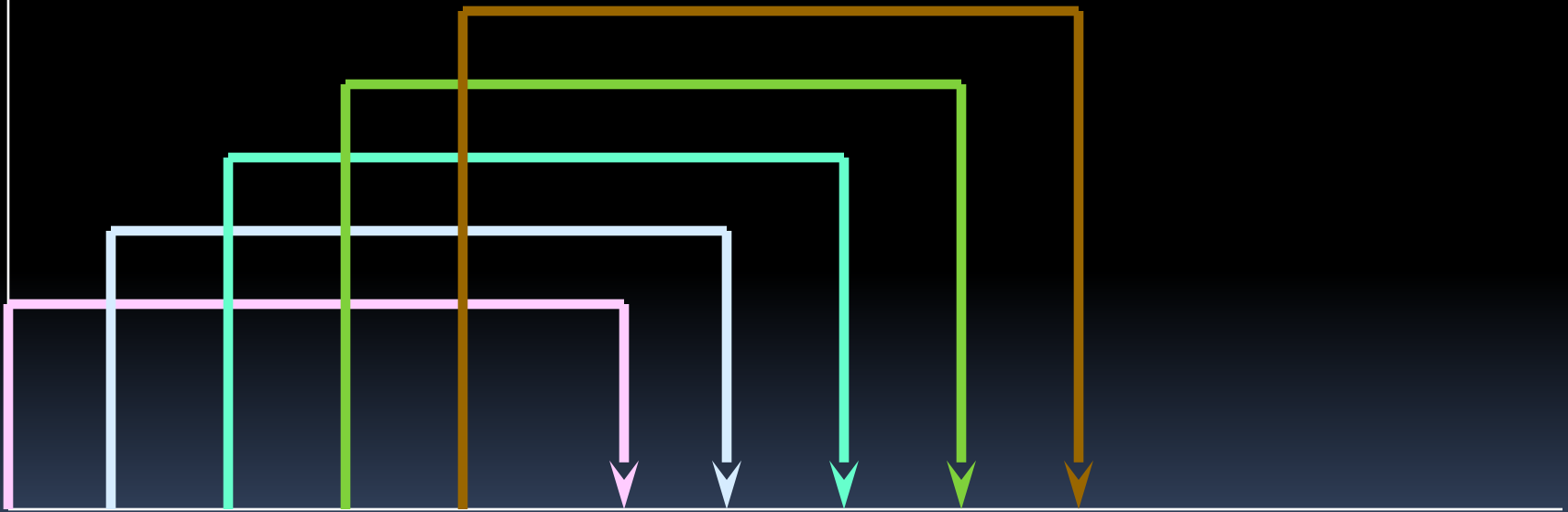
Accès pipeline



Adresses différentes

Modules

Temps d'accès mémoire



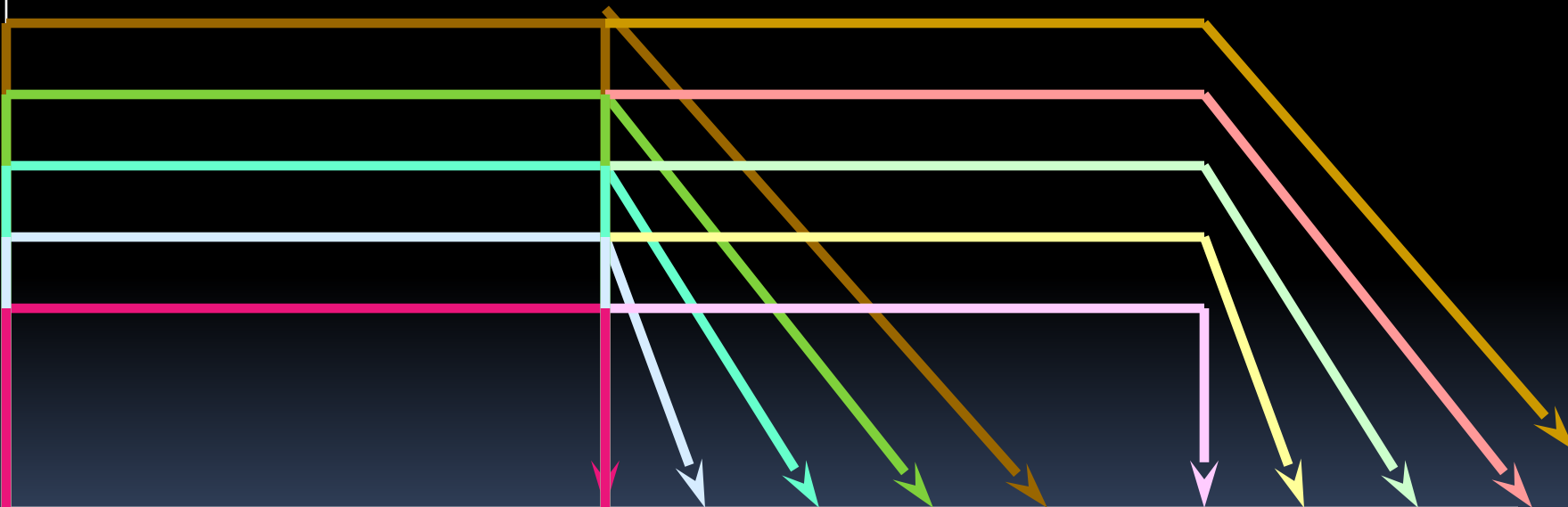
Temps

Même adresse

Modules

Temps d'accès mémoire

Deuxième accès



Nouvelle
adresse

Temps