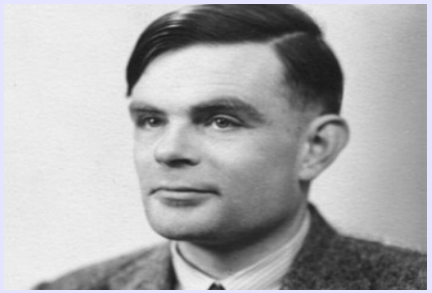
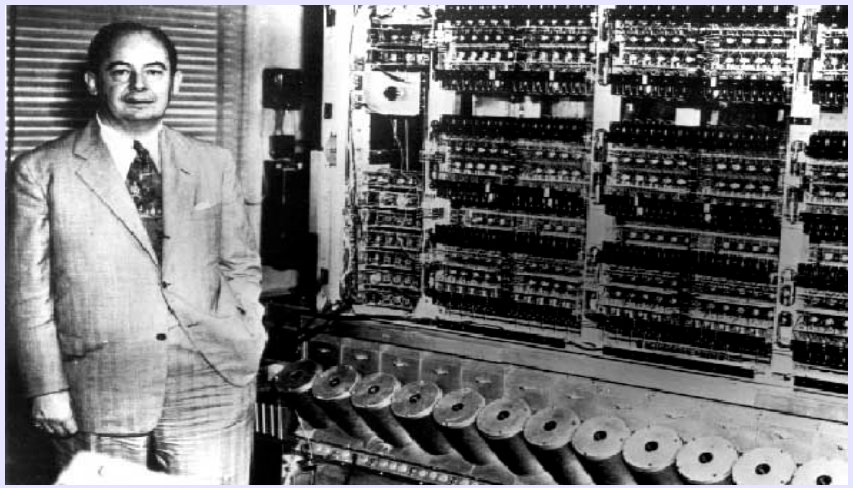




Information, Calcul et Communication

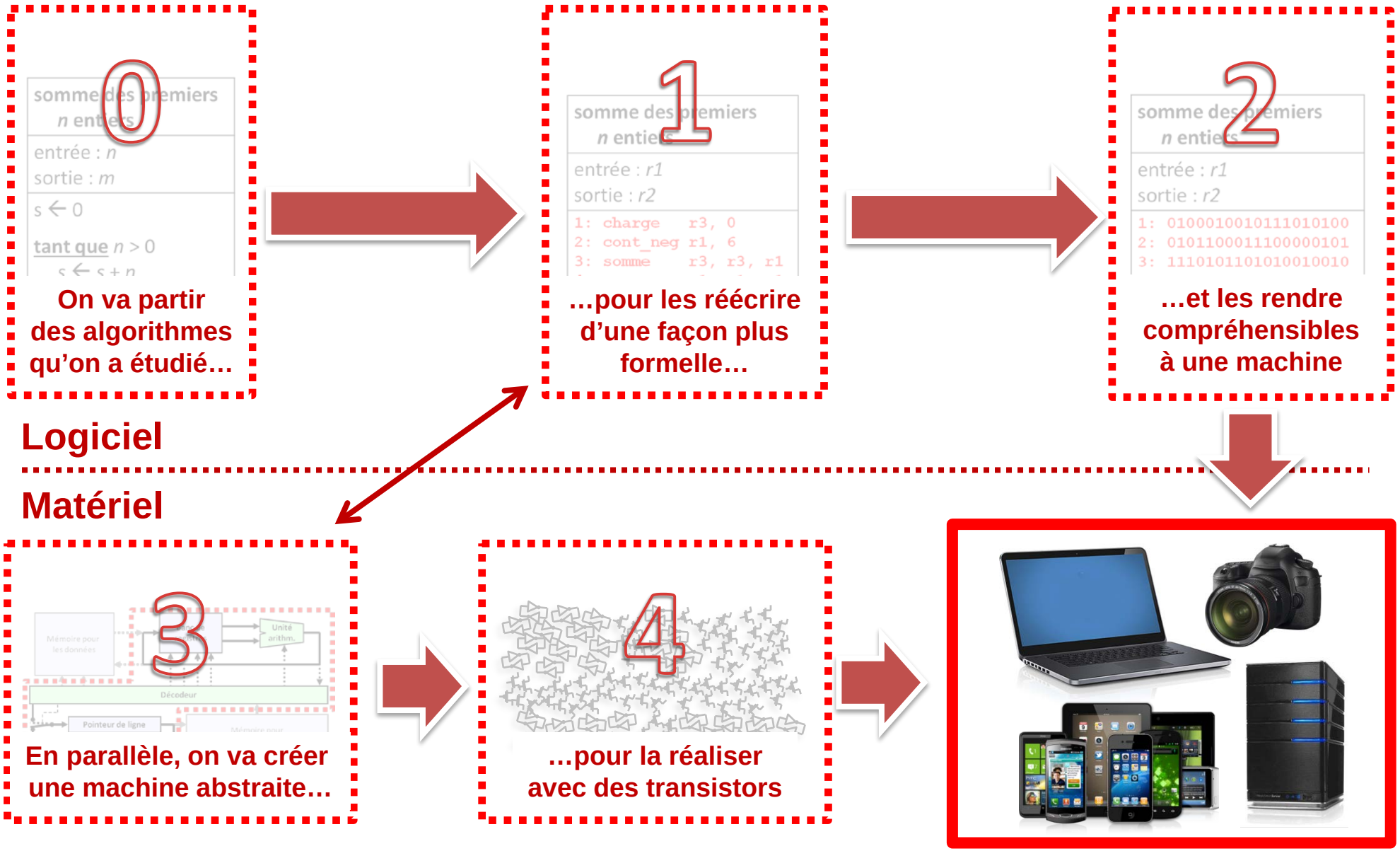
Module 3 : Systèmes

Leçon III.2: Memory Hierarchies

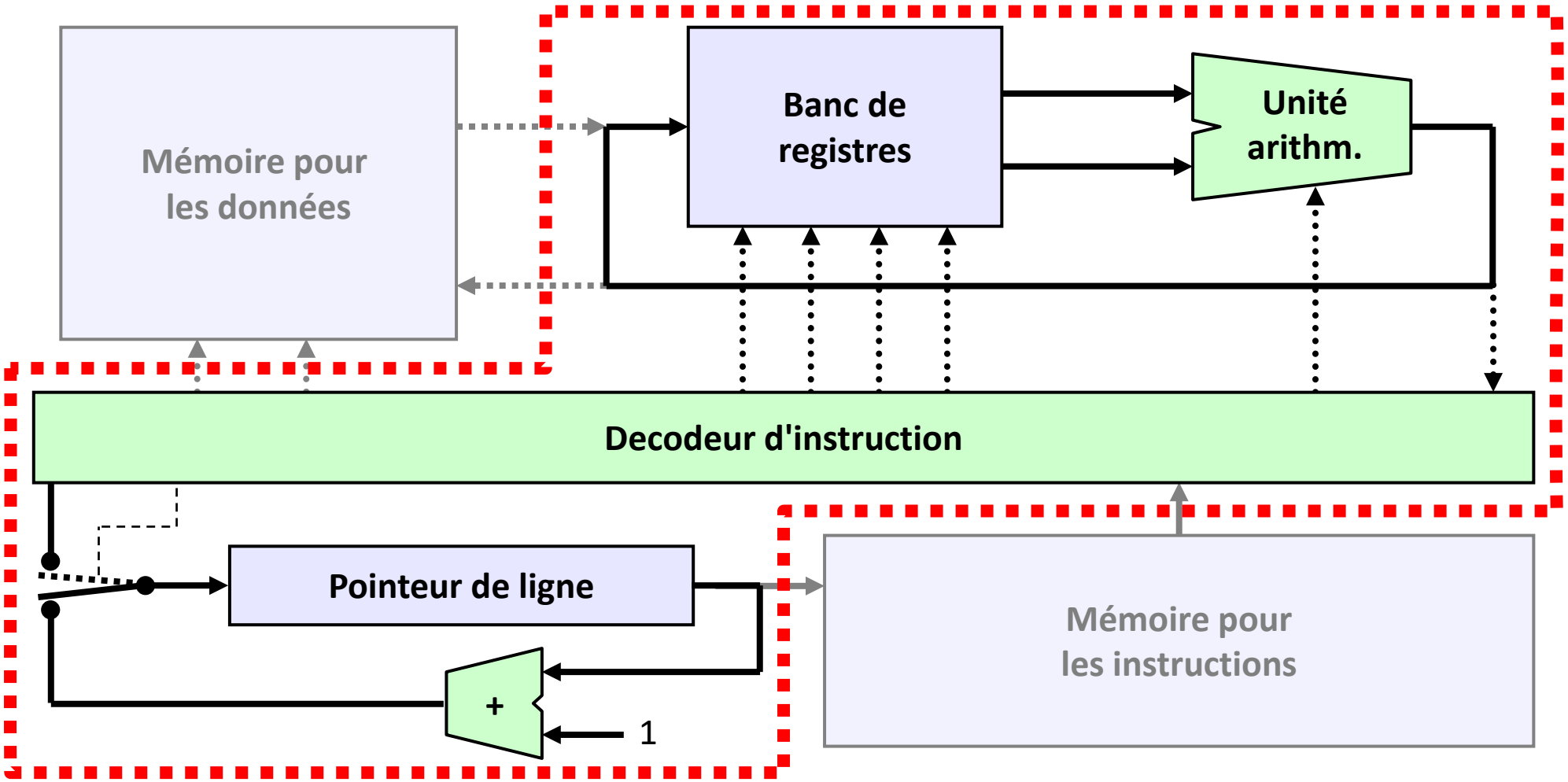
Faculté Informatique et Communications

Babak Falsafi, André Schiper, Willy Zwaenepoel

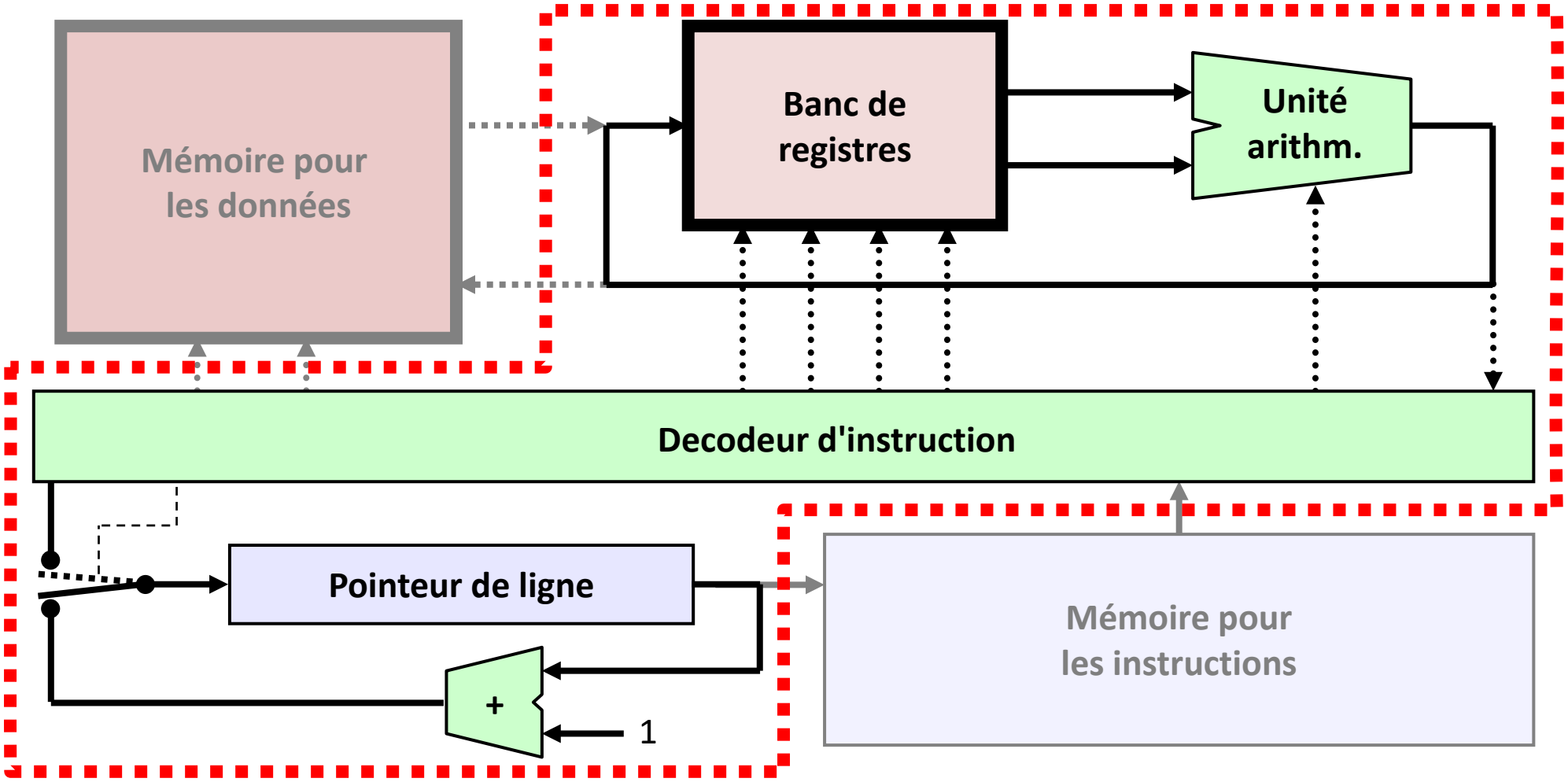
Rappel Leçon1: Des algorithmes aux ordinateurs



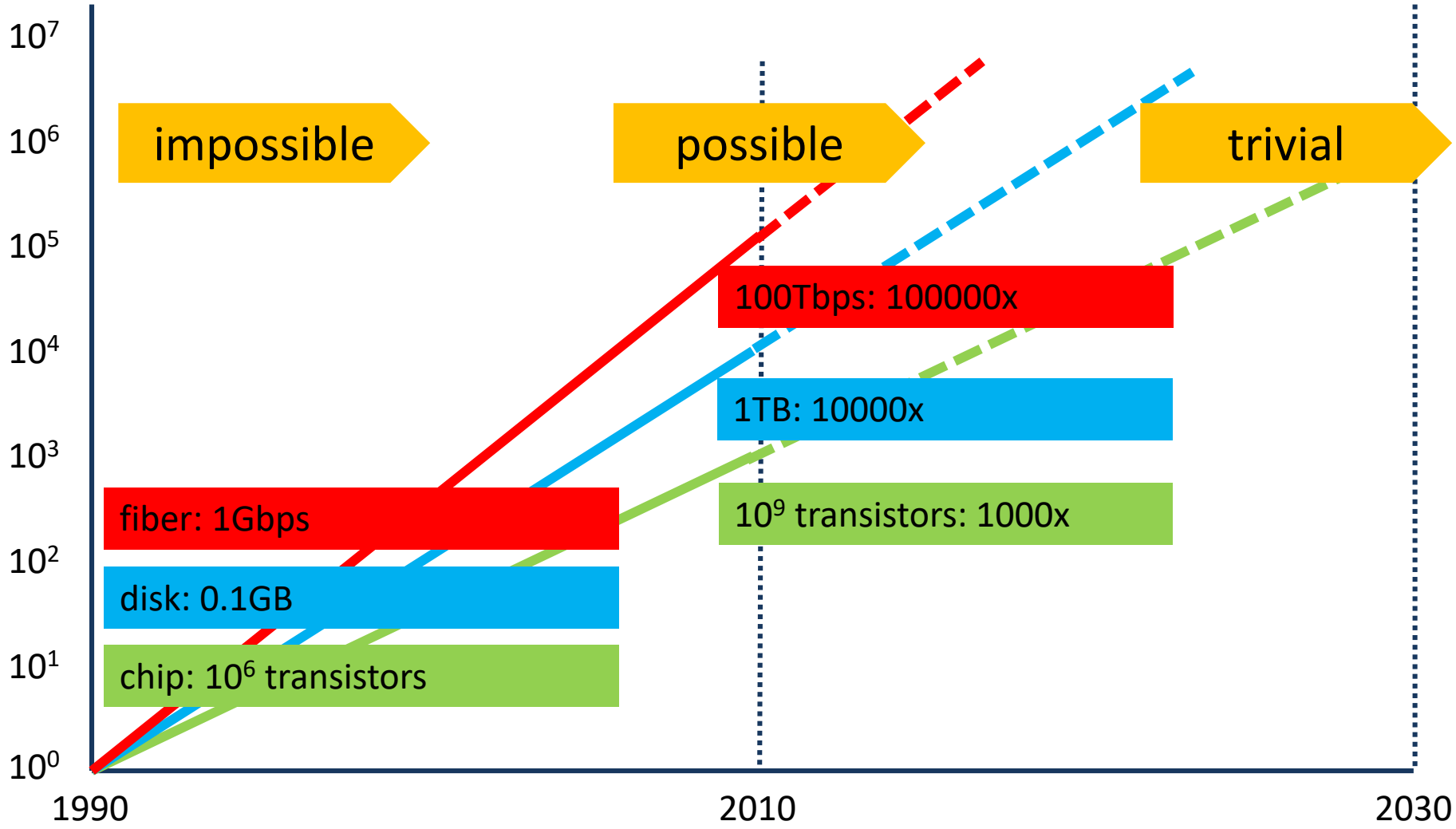
Rappel leçon1: Architecture d'un processeur



Aujourd'hui : Hiérarchie de mémoires (*première partie*)



INTRO ICC: Les accélérations spécifiques de l'Informatique



Source: Matthias Grossglauser, EPFL

La semaine prochaine: stockage de masses de données

- En 2013, on a produit 10^{21} octets de données (x10 par an)

- Organiser les données de façon à pouvoir les rechercher et manipuler efficacement (III.3)
  - Google, réseaux sociaux, administrations, finance, etc...

- Cadre général des 2 leçons:

Principe de la hiérarchisation de la mémoire

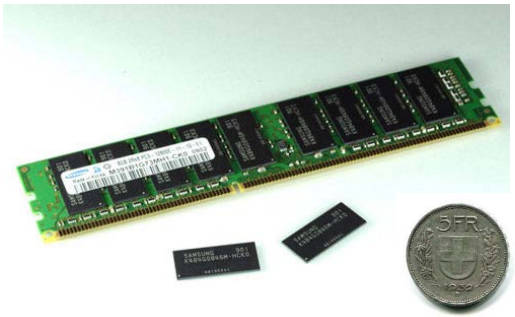
Conserver les données les plus utiles à proximité du processeur

Plan du cours:

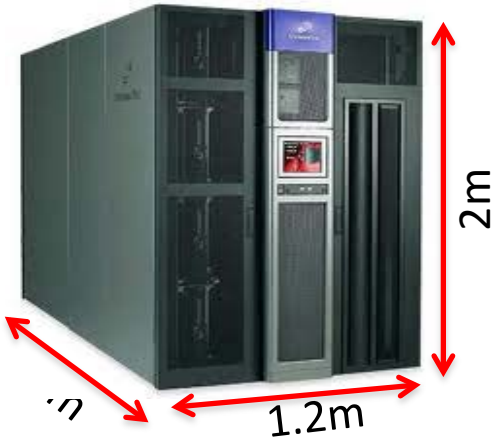
- **Principe du cache**: premier niveau de la hiérarchie de mémoire
- **Fonctionnement du cache** avec le processeur et la mémoire centrale
- **Exemple:** additionner les nombres jusqu'à n
- Le compromis entre ***localité spatiale*** et ***localité temporelle***
- Impact de l'organisation de la mémoire sur les performances d'un algorithme: exemple d'un parcours de matrice

Besoin de technologie pour stocker les données

Memory (GB)



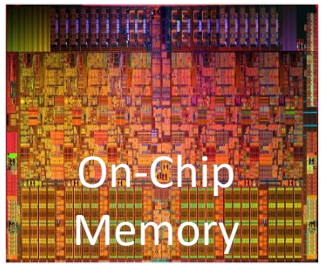
Hard Disk Array (TB)



Tape Robot (PB)

Hard Disk (TB)

Memory (MB)



Tablet FLASH (GB)

USB FLASH (GB)

1 MB = 10^6 bytes

1 GB = 10^9 bytes

1 TB = 10^{12} bytes // Tera

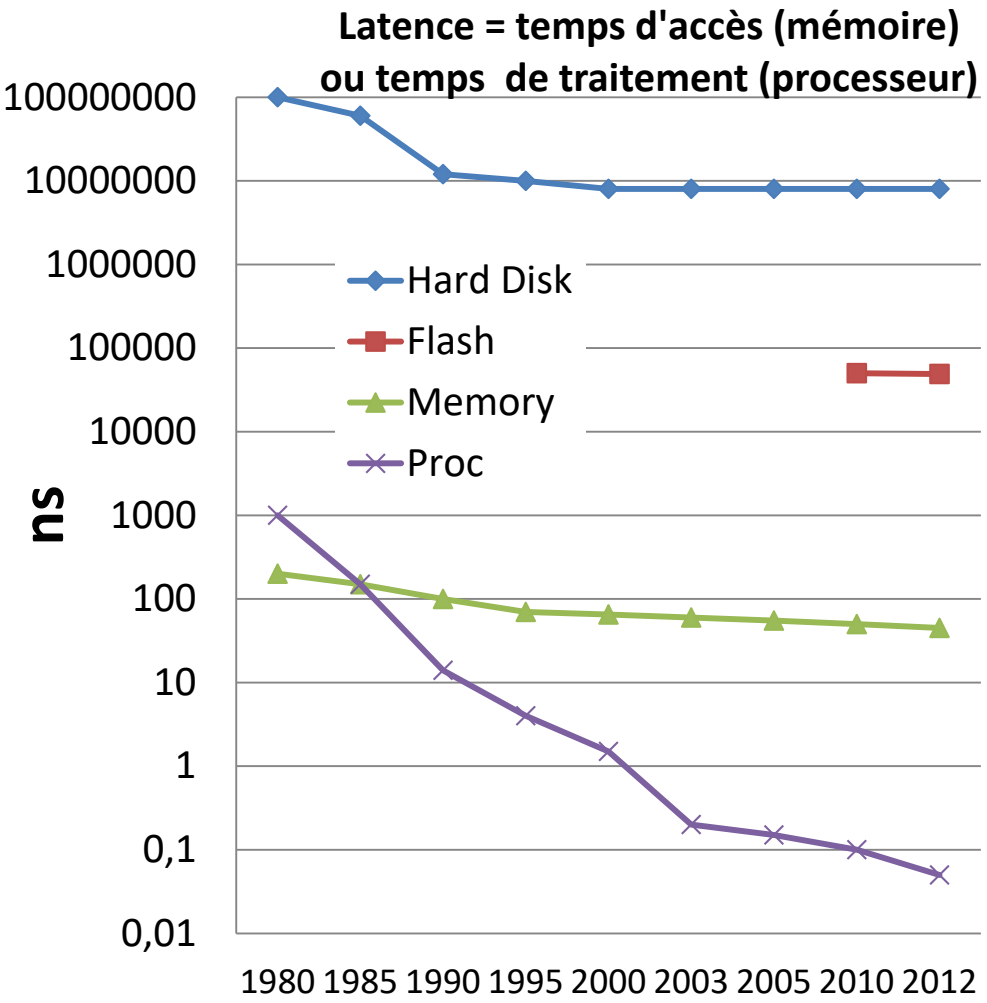
1 PB = 10^{15} bytes // Peta

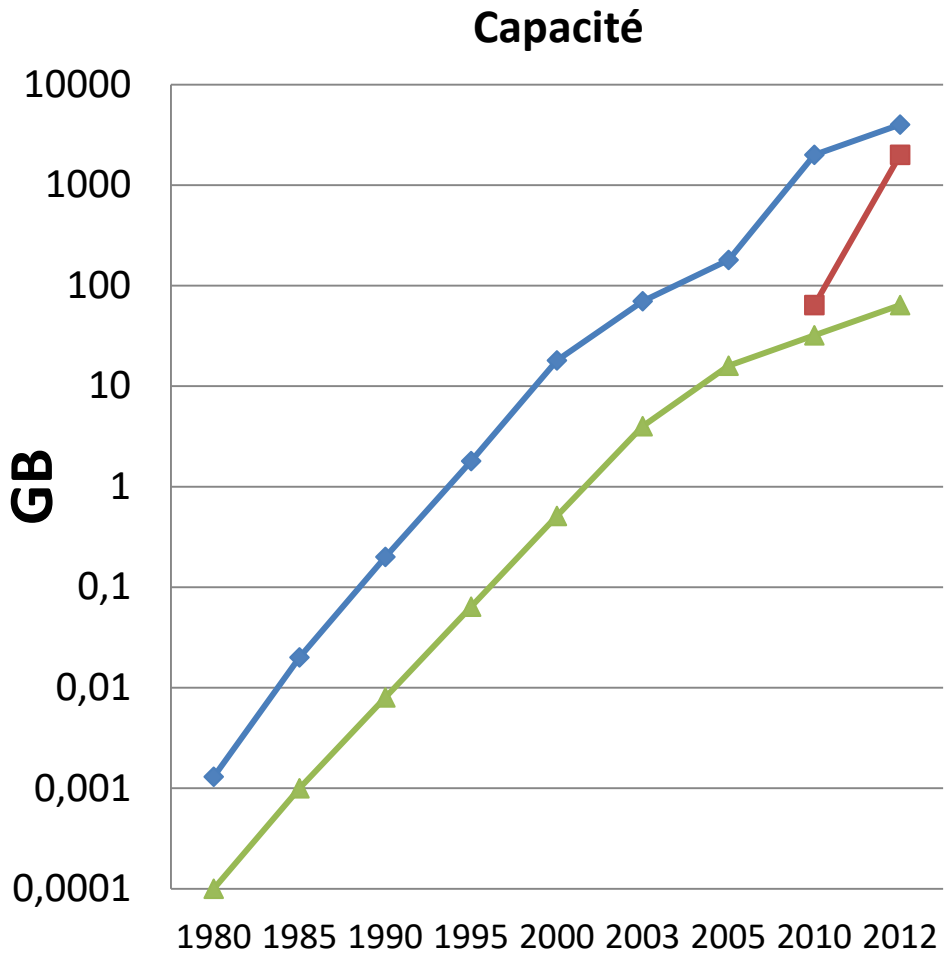
1 EB = 10^{18} bytes // Exa

1 ZB = 10^{21} bytes // Zetta

1 YB = 10^{24} bytes // Yota

Capacité vs. latence: tendances



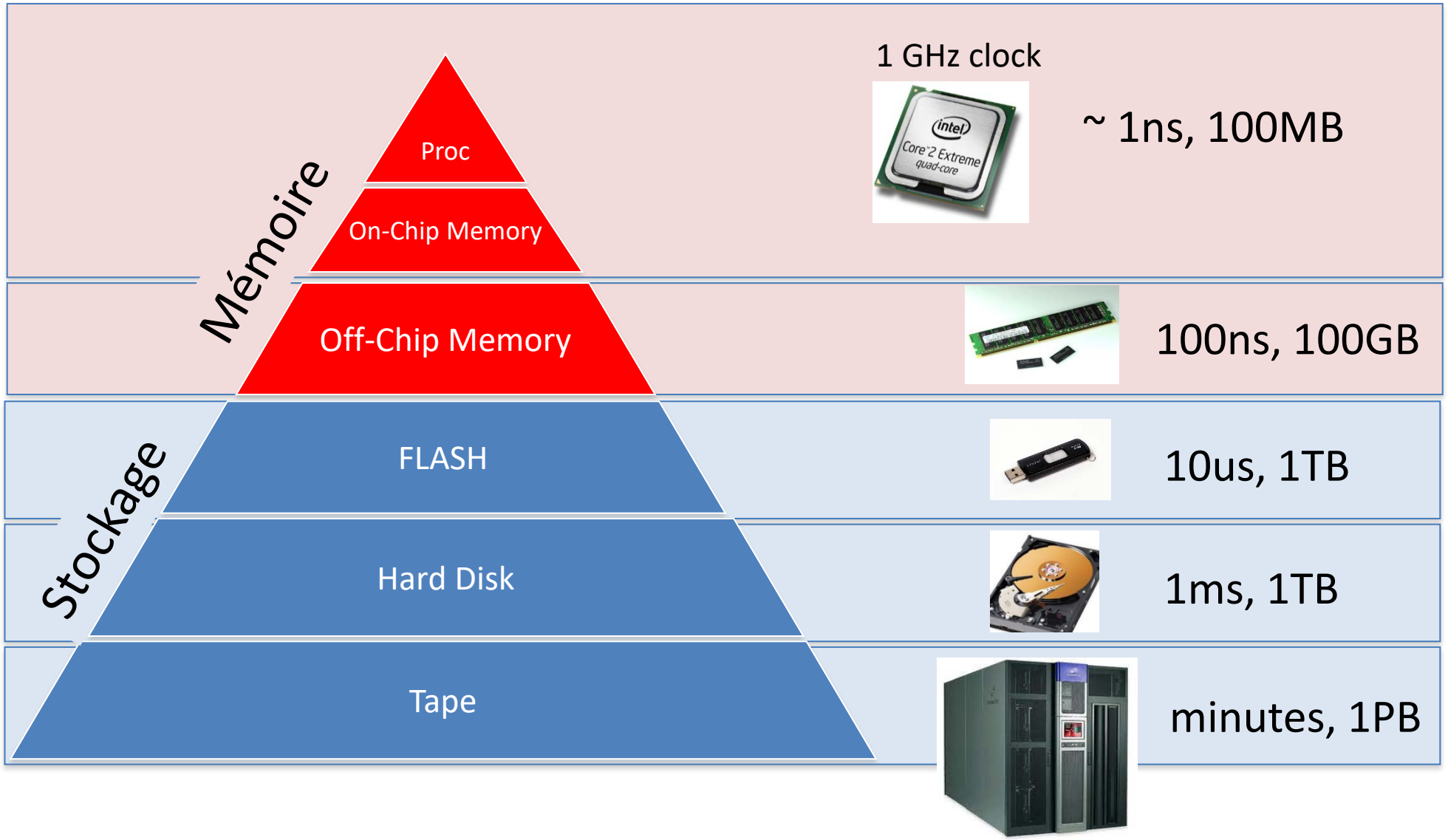


Caractéristiques de la mémoire: *capacité* et *temps d'accès*

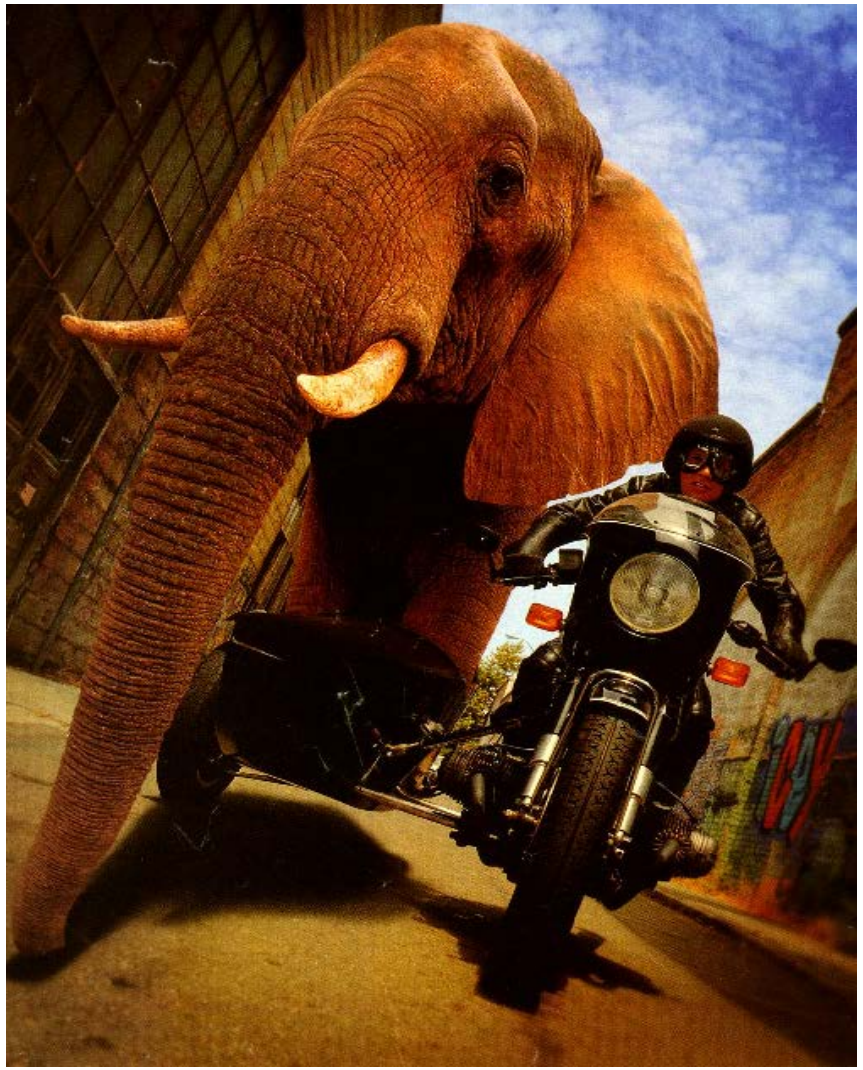
- **Compromis entre Capacité** (en octets / *Bytes, MB, GB...*) et **Temps d'accès** (en s, ns...)
- ***Stockage rapide, mais de petite taille, temporaire***
  - Mémoires temporaires
  - Données facilement disponibles
  - Données perdues lorsque éteint ➔volatile
  - Power hungry
  - Appelée "**mémoire**" (ex: la mémoire centrale)

Ou

- ***Lent, mais beaucoup plus grand, stockage permanent***
  - Pour archivage (non-volatile)
  - Données utiles pour accès ultérieur
  - Structuré et indexé
  - Accès plus lent
  - Peu gourmand en énergie
  - Appelée (mémoire de) "stockage"


Mémoire
Stockage
Proc
On-Chip Memory
Off-Chip Memory
FLASH
Hard Disk
Tape
1 GHz clock
~ 1ns, 100MB
100ns, 100GB
10us, 1TB
1ms, 1TB
minutes, 1PB

La mémoire idéale est ...



Grande (très large capacité)

Accès rapide
(aussi rapide que l'accès aux registres du processeur)

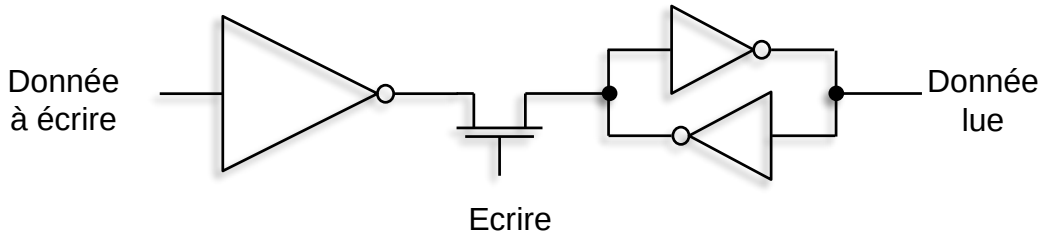
MAIS ça n'est pas possible:

- trop couteux
- consomme trop d'énergie
- utilise trop d'espace

La mémoire idéale n'existe pas ...

La technologie des registres (mémoire on-chip) est coûteuse en espace et en énergie

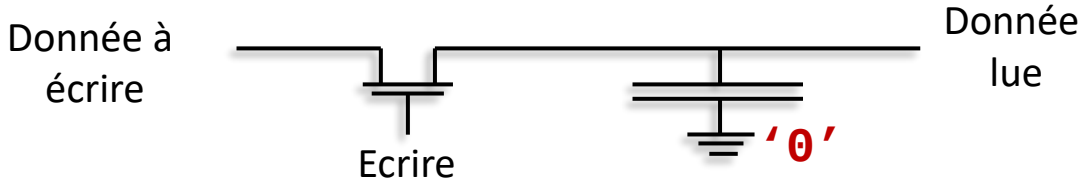
1 bit =



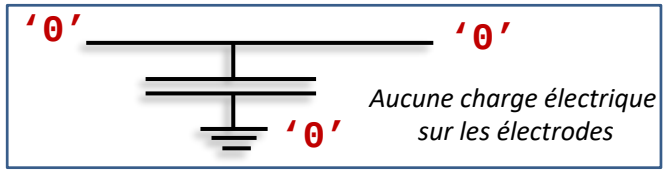
Chaque inverseur utilise deux transitors

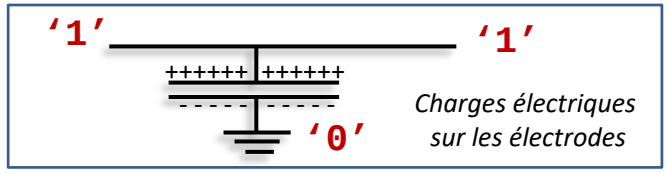
La technologie de la mémoire centrale (off-chip) demande beaucoup moins d'espace et d'énergie

1 bit =



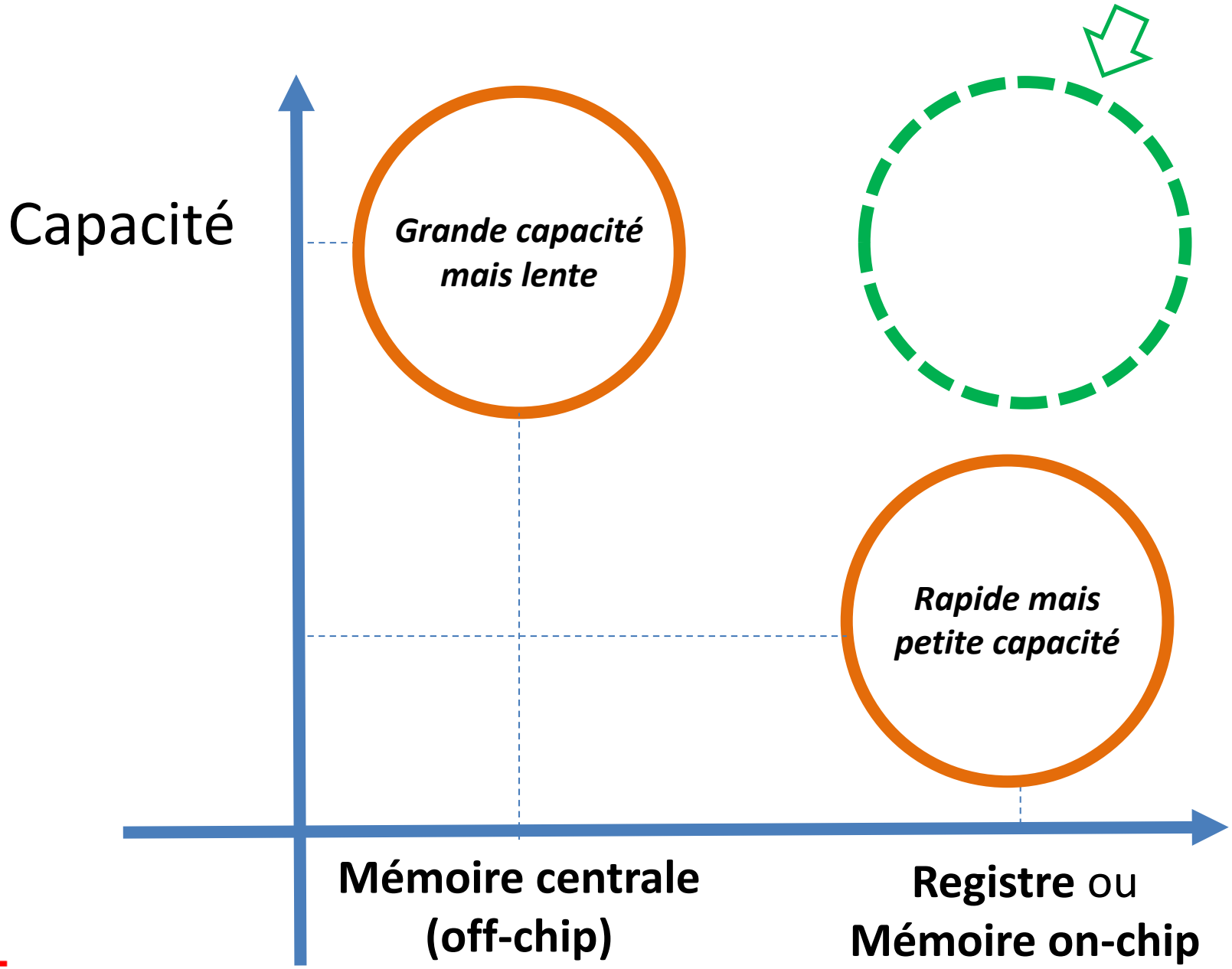
Utilisation d'un élément appelé **condensateur** = deux électrodes séparées par un isolant.





Les électrodes peuvent accumuler des charges électriques et ainsi mémoriser les niveaux de tension 0 et 1 en sortie.
Demande beaucoup moins d'énergie que l'approche on-chip
MAIS demande 100ns pour lire/écrire

QUE FAIRE ? Hiérarchiser la mémoire en utilisant un ***cache***



***Motivation**: donner l'illusion d'une mémoire **grande** et **rapide** au processeur*

Principe du cache:

► Idéalement, donner au processeur tout ce dont il a besoin en 1ns (= un cycle de l'horloge d'un processeur à 1 GHz)

• Mais toutes les données ne peuvent être gardées on-chip

Conserver on-chip seulement les données que ***l'on va utiliser prochainement***

Analogie avec la vie réelle

Musique/photos sur smartphone
- Usage temporaire
- Espace limité
- Dans votre poche, rapide

Habits dans valise
- Pour un voyage
- En fonction du temps
- Usage temporaire
- Espace limité
- Avec vous

Laptop / Cloud
- Tou(te)s les chansons/photos/films
- Archivé
- Capacité “illimitée”
- A la maison/au travail, distant

Armoire
- Tous vos habits
- Permanent
- Uniquement à la maison

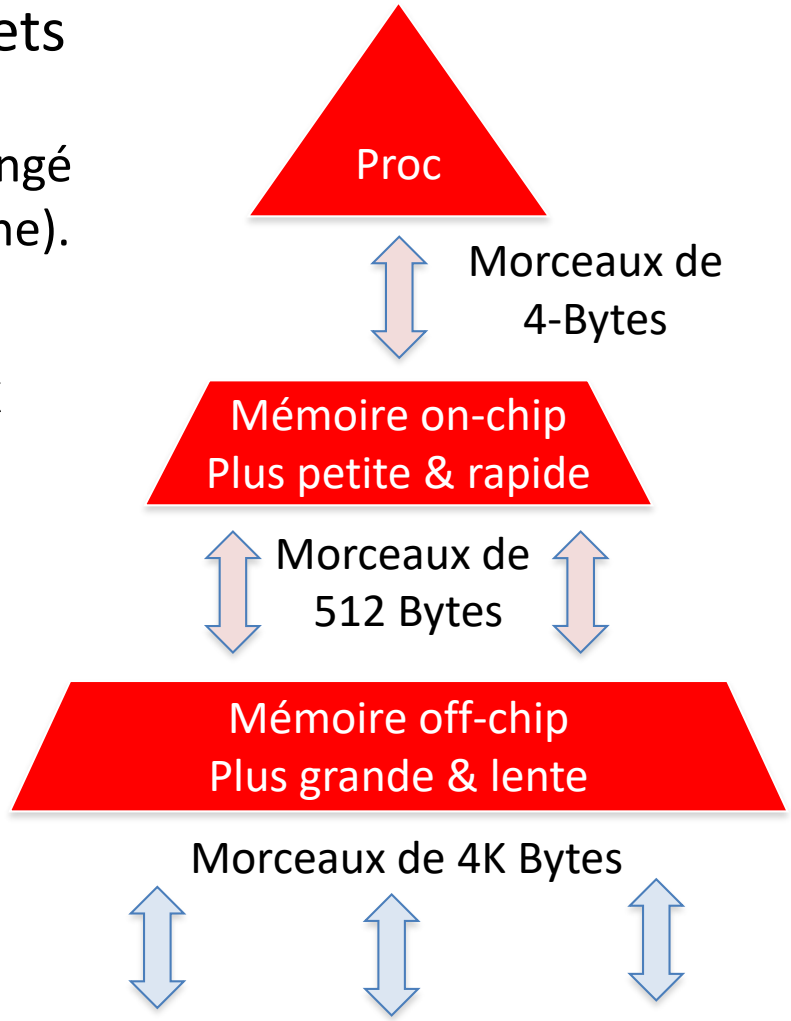
Comment compenser une faible vitesse d'accès (latence) ? Réponse: en transférant des morceaux plus gros

- La taille des données est exprimées en octets
  - Plus petite unité de mémoire
  - Mot = 4 octets(bytes) = taille du morceau échangé entre le Processeur et la mémoire on-chip (cache).

- **mémoire *on-chip* (cache)*:*** petits morceaux
  - D'un mot à quelques centaines d'octets entre le cache et la mémoire off-chip

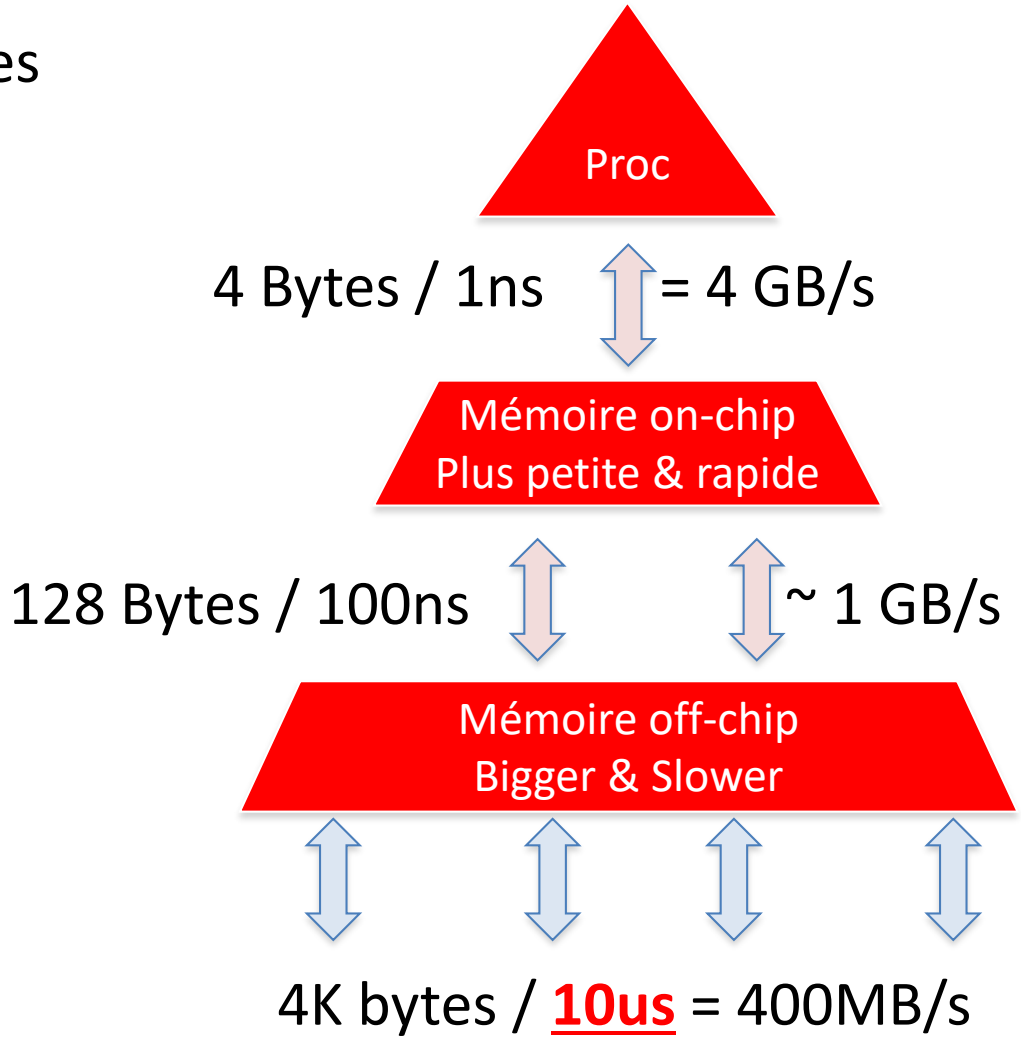
- **mémoire *off-chip***, morceaux plus grands
  - Quelques milliers d'octets

- Raisonnement niveaux inférieurs
  - Plus grands ➔ plus de capacité
  - Plus larges ➔ plus d'info déplacées par unite de temps



Taille de morceau / Latence = "Bande passante" (=débit)

- Différence de latence (temps d'accès) sur les 3 niveaux:
  - Temps d'accès d'un mot (4 Bytes)
  - **1ns** vs. **100 ns** vs. **10us**
- Mais les niveaux inf. sont plus *larges*
  - Peuvent transférer plus de données à la fois
  - Appelé "bande passante"
- Pour obtenir une bande passante élevée
  - Accès d'un morceau en parallèle
  - Avec un morceau de 4KB, la bande passante est 10 fois plus petite qu'au niveau du Proc.
  - Mais la latence est 10'000x plus élevée!



Suite du cours:

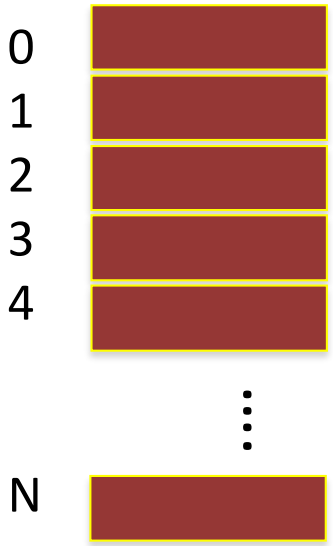
- **Principe du cache**: une mémoire ***on-chip*** = d'accès rapide pour le processeur qui lui donne d'une mémoire grande et rapide
- **Fonctionnement du cache** avec le processeur et la mémoire centrale
- **Exemple:** additionner les nombres jusqu'à n
- Le compromis entre ***localité spatiale*** et ***localité temporelle***
- Impact de l'organisation de la mémoire sur les performances d'un algorithme: exemple d'un parcours de matrice

La Mémoire du point de vue du processeur (mémoire centrale)

- Organisée comme un tableau de mots
  - chaque mot:
    - contient 4 à 8 octets (selon machine)
    - est numéroté (~ adresse postale)
    - à une ***adresse*** entre 0 et N:
      1er mot @ 0,
      2ème mot @ 1, etc.

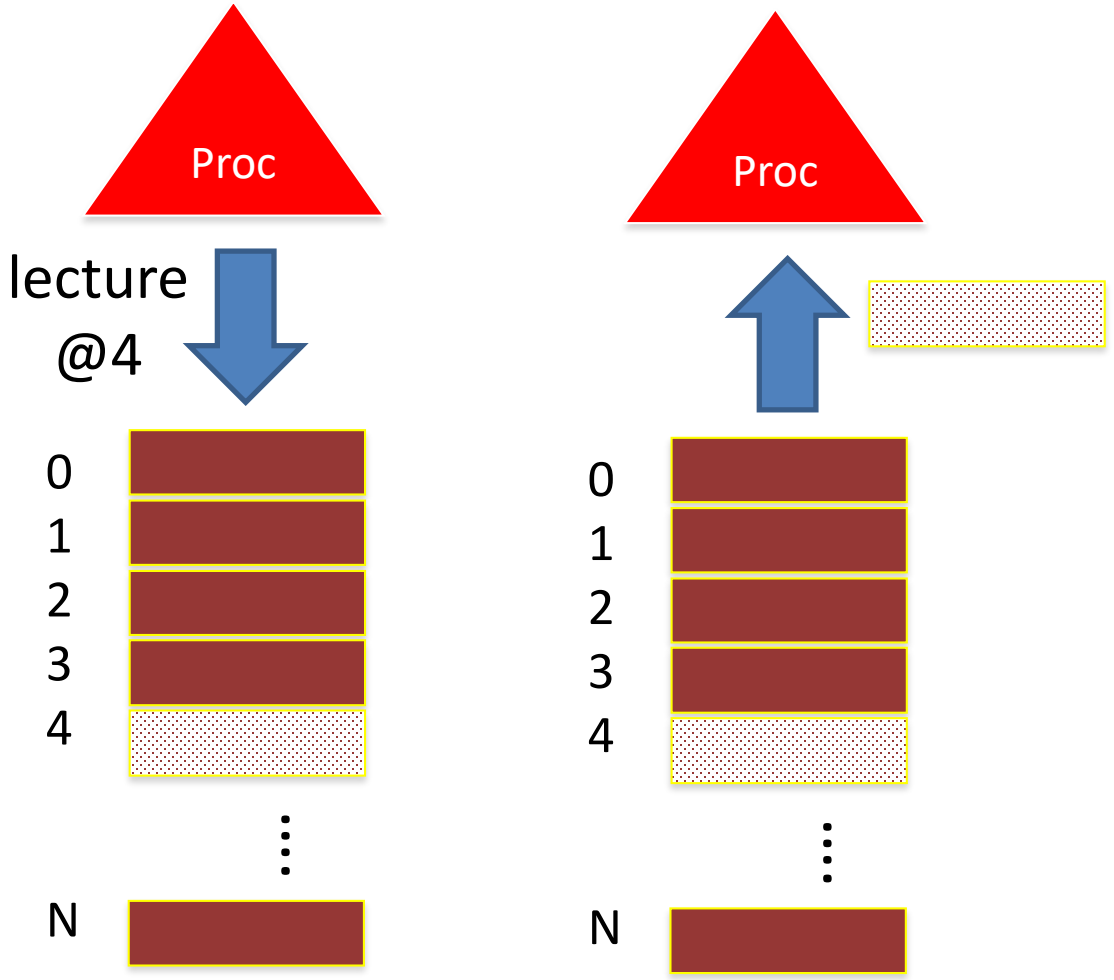
- Stocke données & instructions (III.1)
  - De multiples programmes peuvent coexister
  - Intervalle d'adresses assigné à chaque programme (segment)

Tableau de mots



Lecture d'un mot mémoire par le processeur

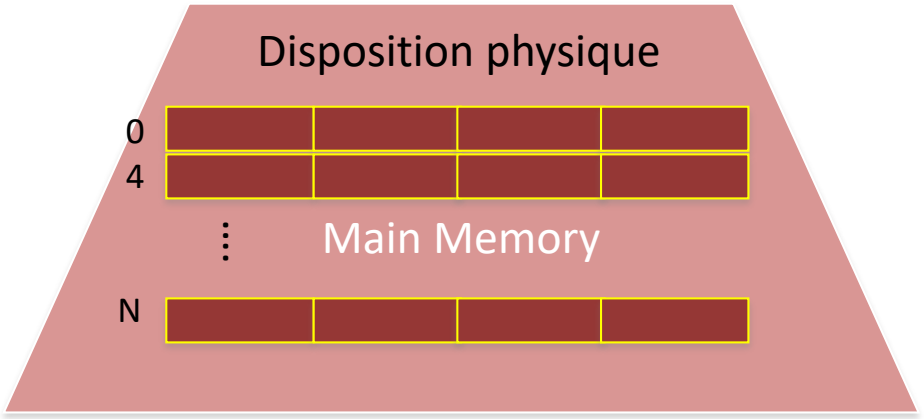
- le processeur demande à la mémoire le contenu (motif binaire) du mot d'adresse A
- La mémoire renvoie la valeur du mot au processeur
- le processeur stocke la valeur dans un registre



Organisation de la mémoire centrale

- Déplacer plus d'infos à la fois pour compenser la latence
  - la ***mémoire centrale*** est organisée en ***blocs (de 16 à 128 Bytes)***
  - le ***cache*** est aussi organisé en ***blocs***
  - le transfert entre les deux se fait par ***bloc***

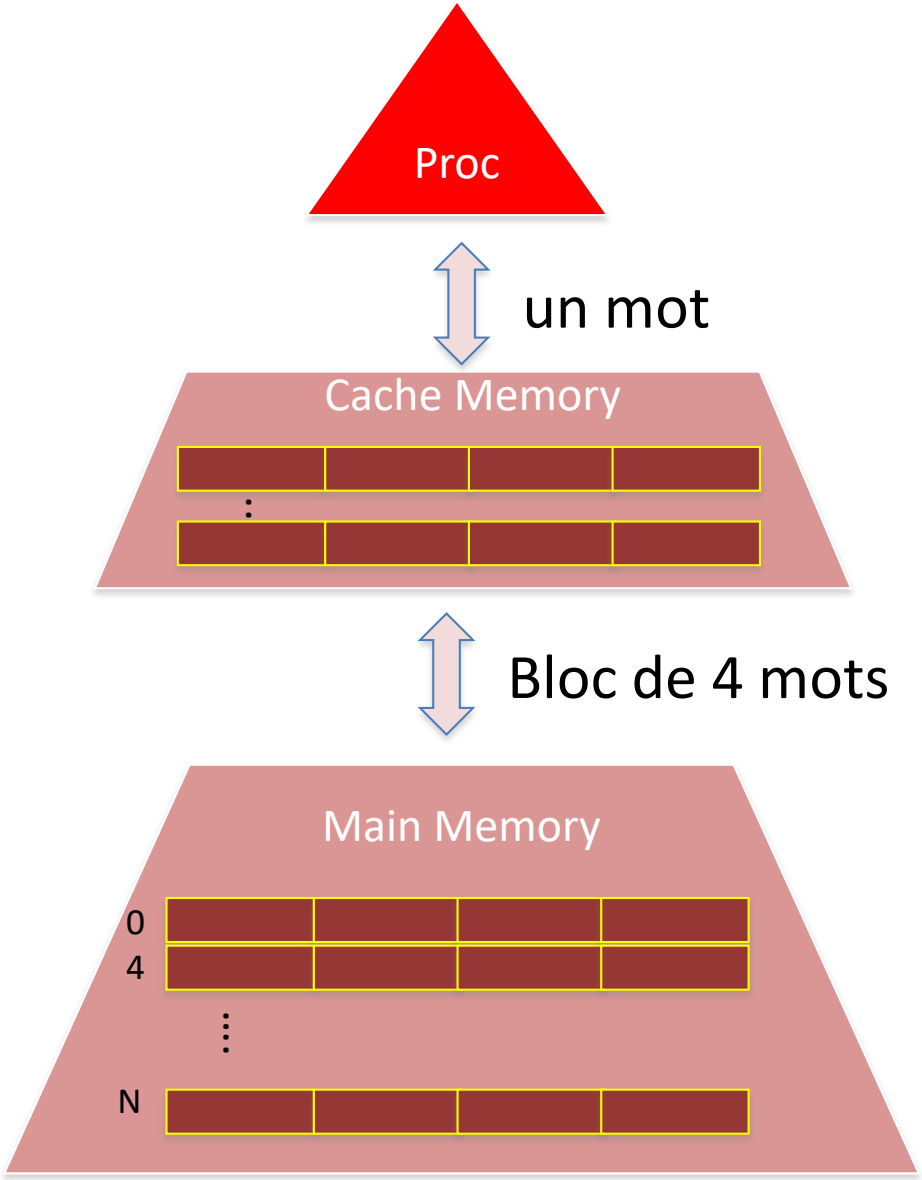
- representation de la mémoire centrale:



Bloc = 16 Bytes
= 4 mots

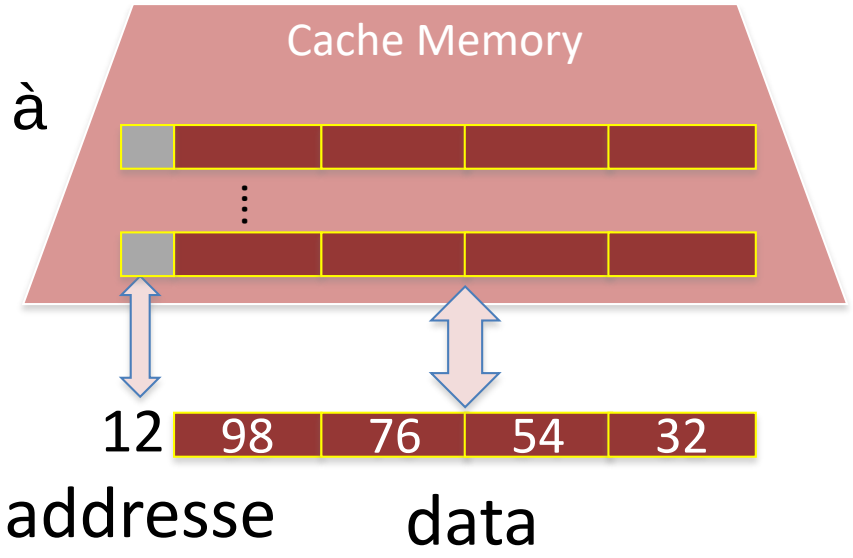
Mémoire on-chip (cache)

- ▶Mémoire plus rapide
  - 1ns à 10ns
  - Plus proche de la vitesse du processeur
- ▶Mais capacité limitée
  - 64KB à 64MB
  - Mémoire principale peut avoir 1TB
  - Ne peut contenir toutes les données/instructions
  - Infos regroupées par bloc
- ▶Q: Comment rechercher un bloc?
- ▶Q: Quels blocs en cache?



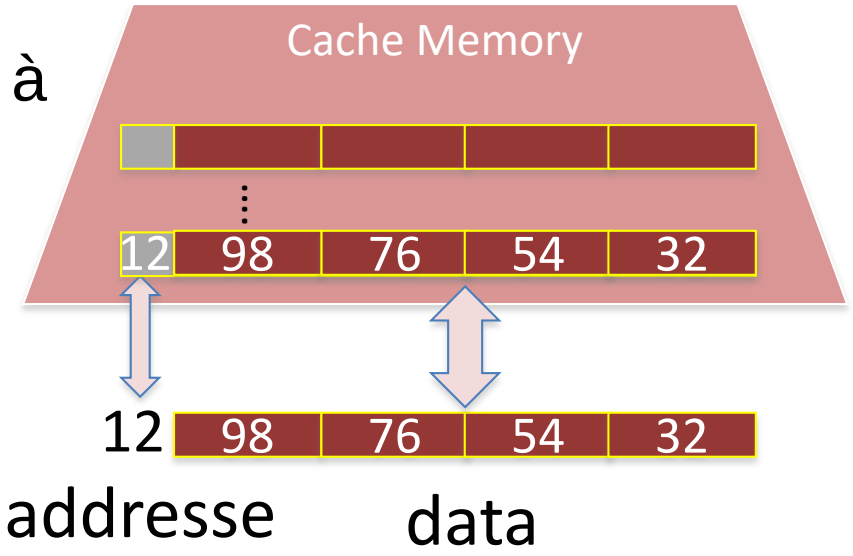
Transfert de la mémoire centrale vers le cache

- Cache:
  - Tableau composé de deux parties:
    - adresse d'un bloc de la mémoire centrale (ex: 12)
    - les données du bloc commençant à cette adresse (ex: 98765432)



Transfert de la mémoire centrale vers le cache

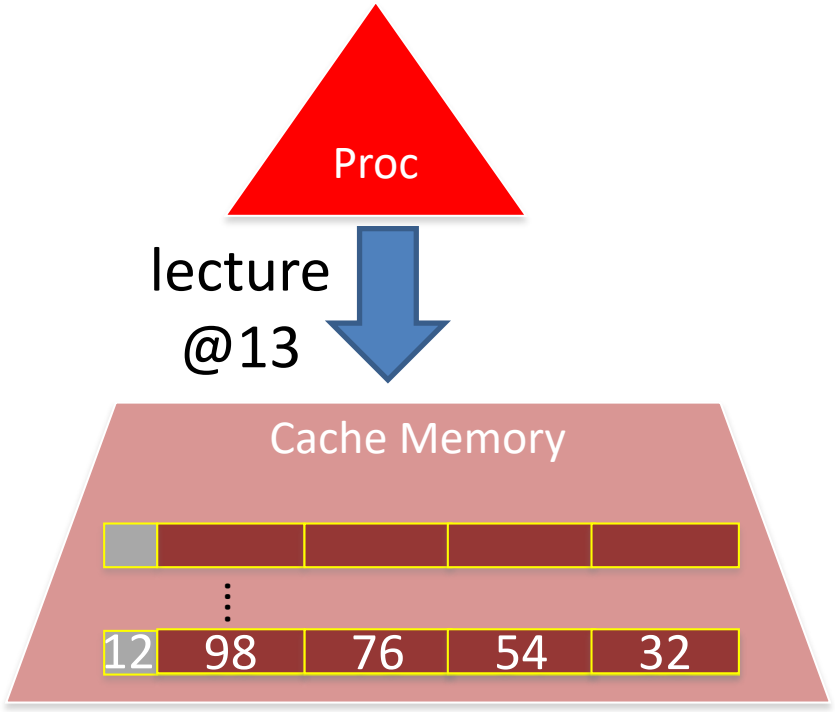
- Cache:
  - Tableau composé de deux parties:
    - adresse d'un bloc de la mémoire centrale (ex: 12)
    - les données du bloc commençant à cette adresse (ex: 98 76 54 32)



  - Plusieurs blocs de la mémoire centrale peuvent résider temporairement dans le cache

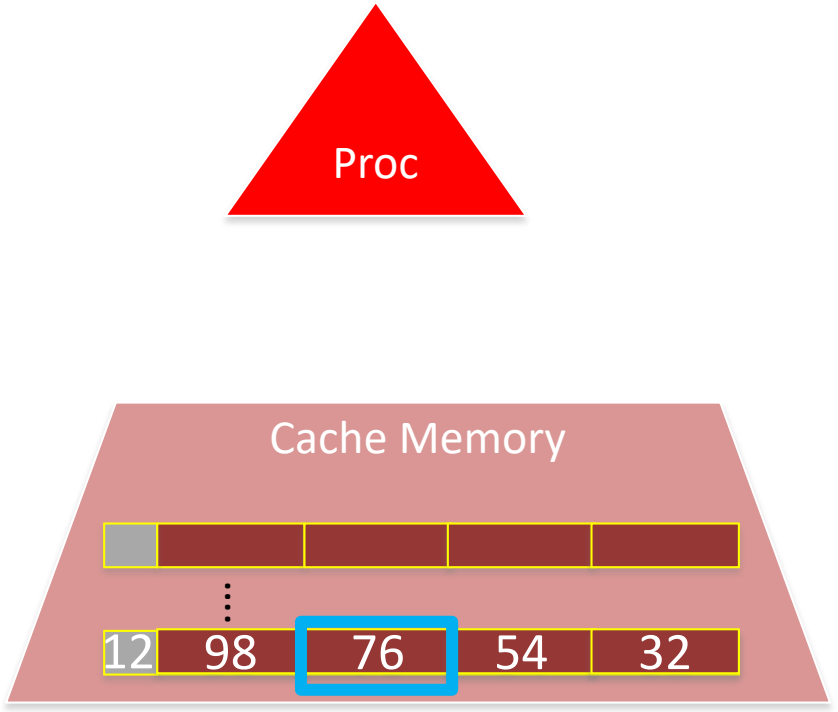
Recherche de données par le Processeur

- ***Le processeur envoie l'adresse du mot recherché***



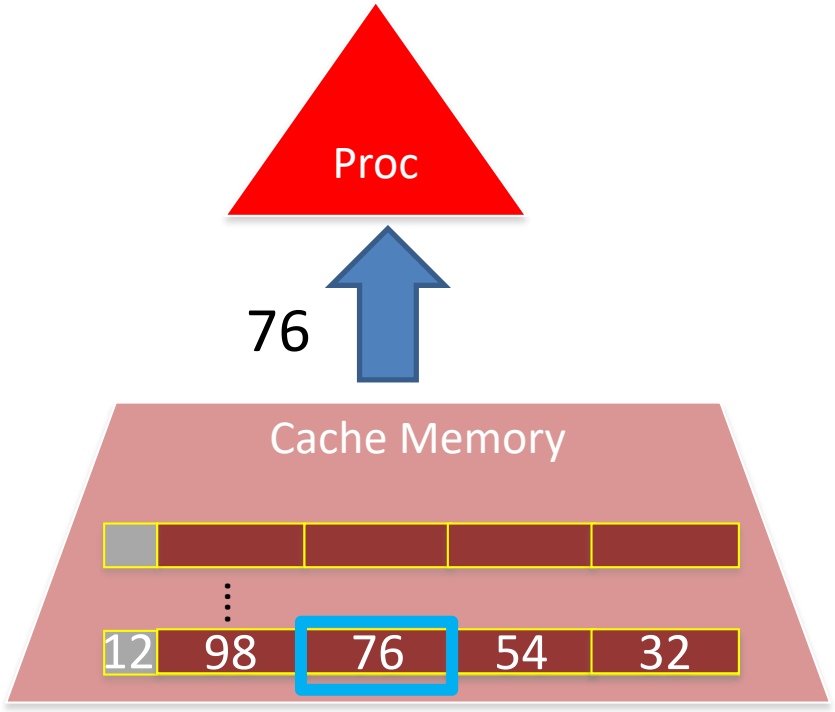
Recherche de données par le Processeur

- ***Le cache vérifie si la donnée est présente dans le cache***



Recherche de données par le Processeur

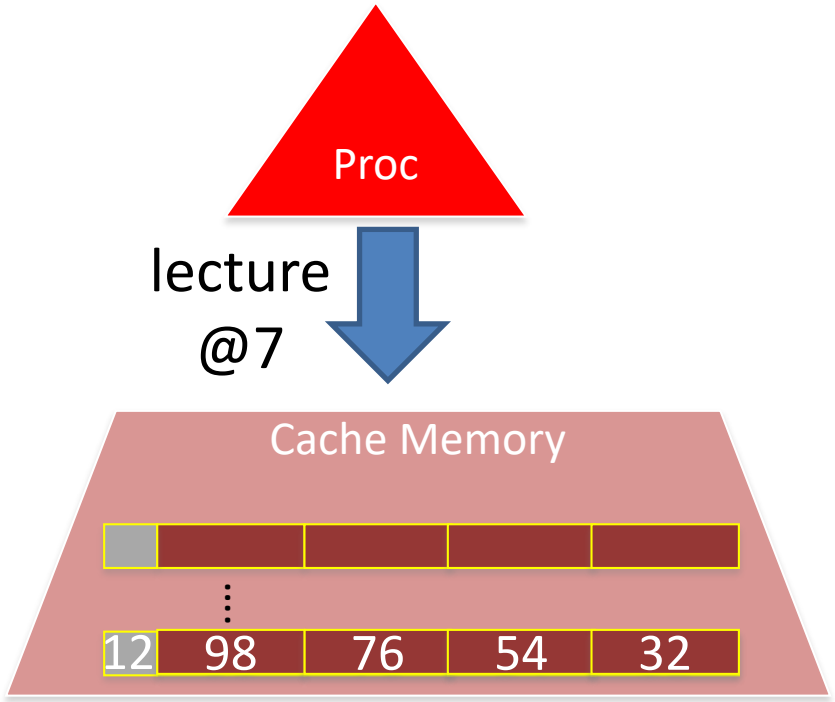
- ***Si oui, le cache envoie le mot (1ns)***



Recherche de données par le Processeur (cas échec)

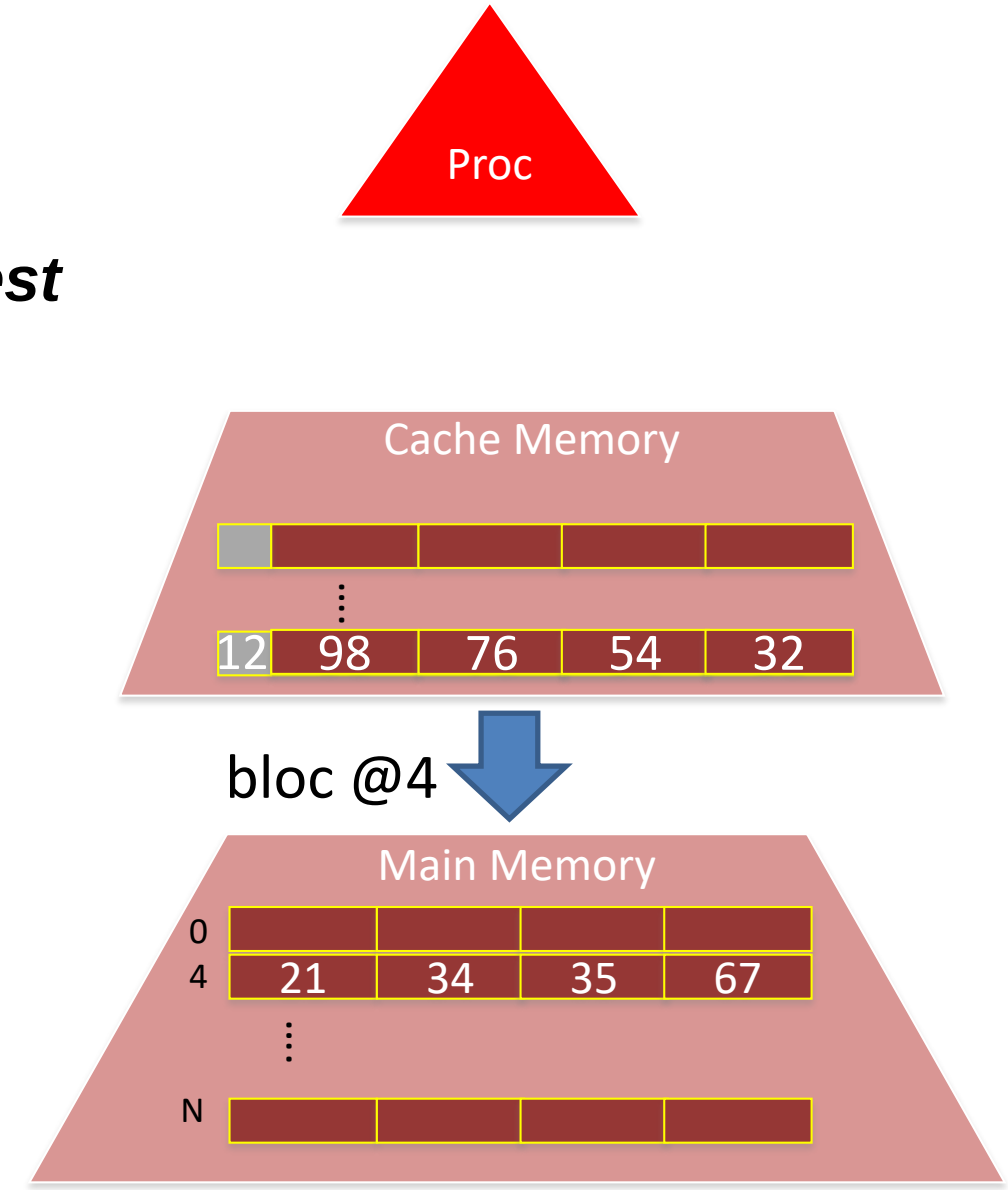
Toutes les données ne peuvent pas être dans le cache !

- ***Le processeur envoie l'adresse du mot recherché***



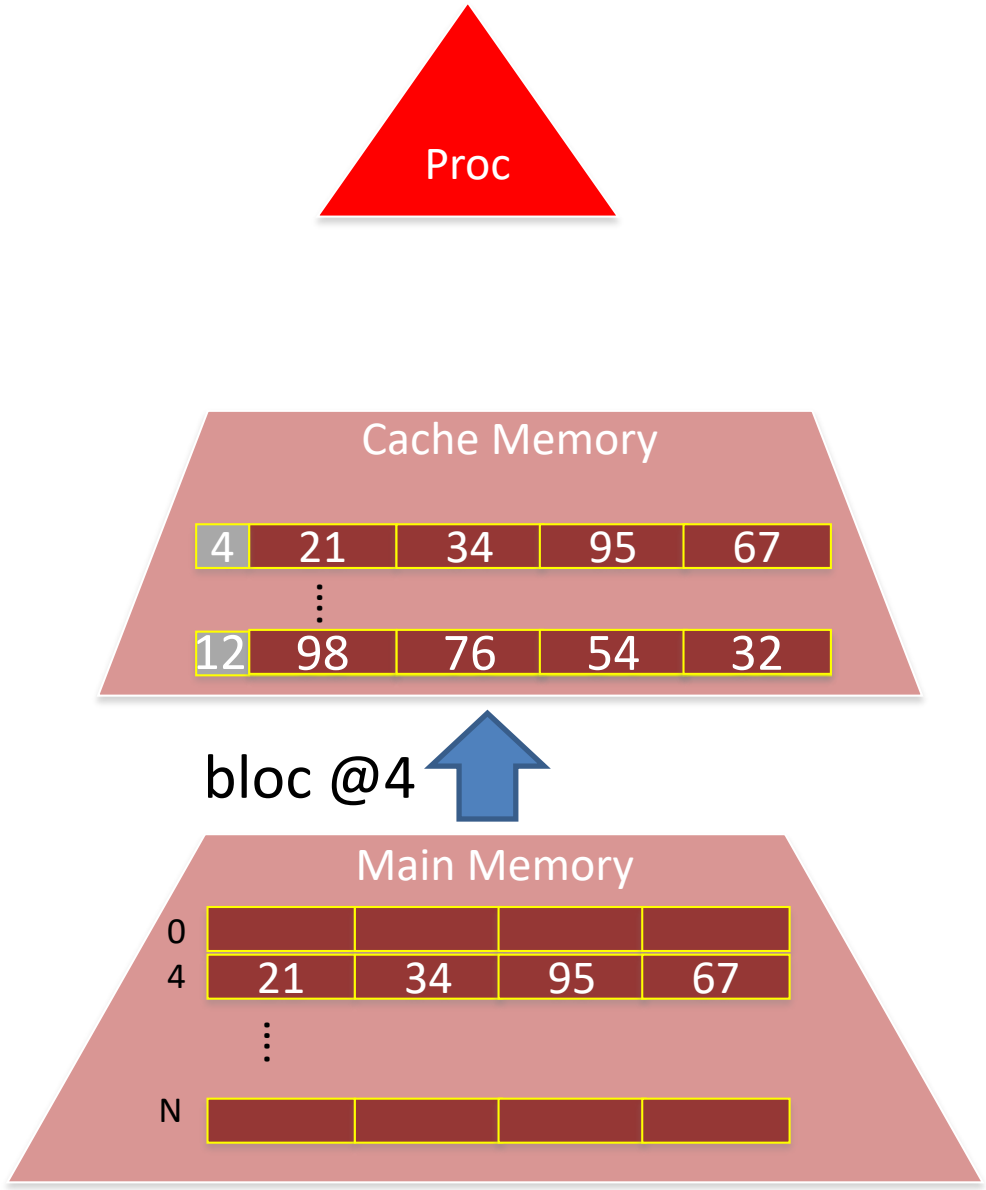
Recherche de données par le Processeur (cas échec)

- ***Le cache vérifie si la donnée est présente dans le cache***
- ***Si non, il faut charger un bloc de la mémoire vers le cache (100ns)***



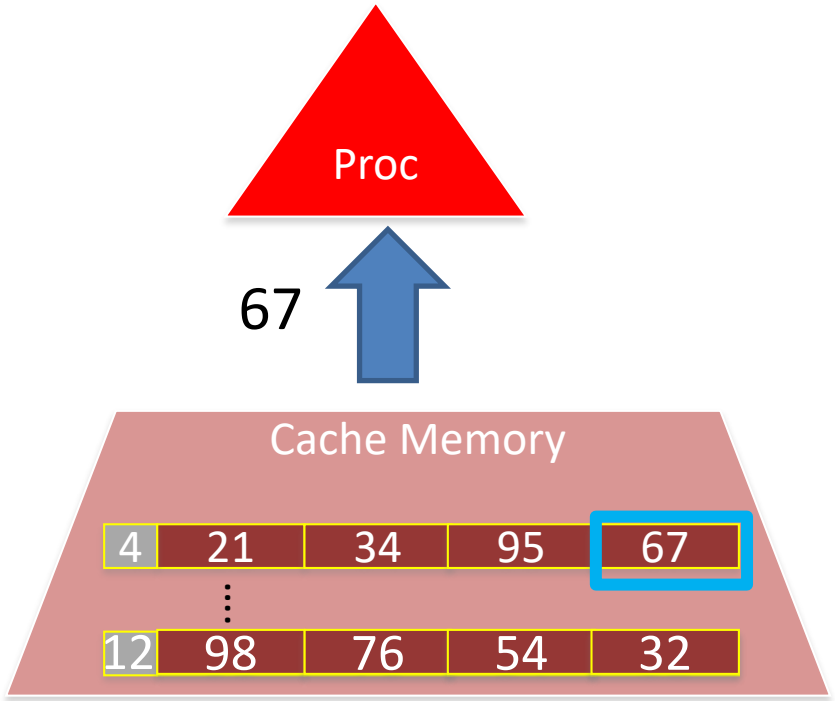
Recherche de données par le Processeur (cas échec)

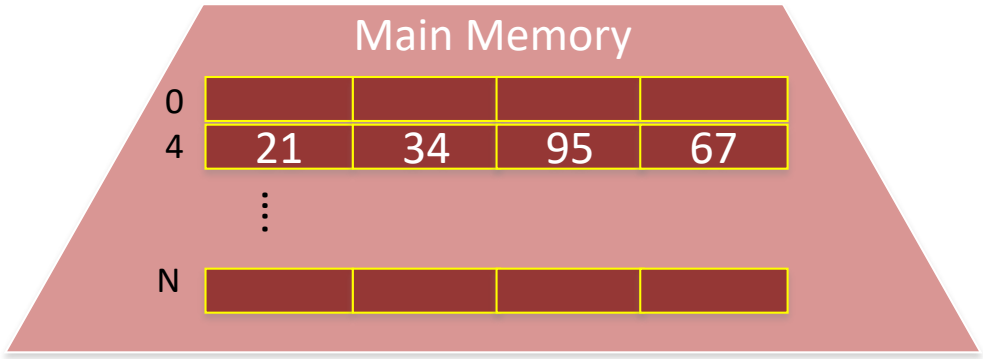
- ***Si non, il faut charger un bloc de la mémoire vers le cache (100ns)***



Recherche de données par le Processeur (cas échec)

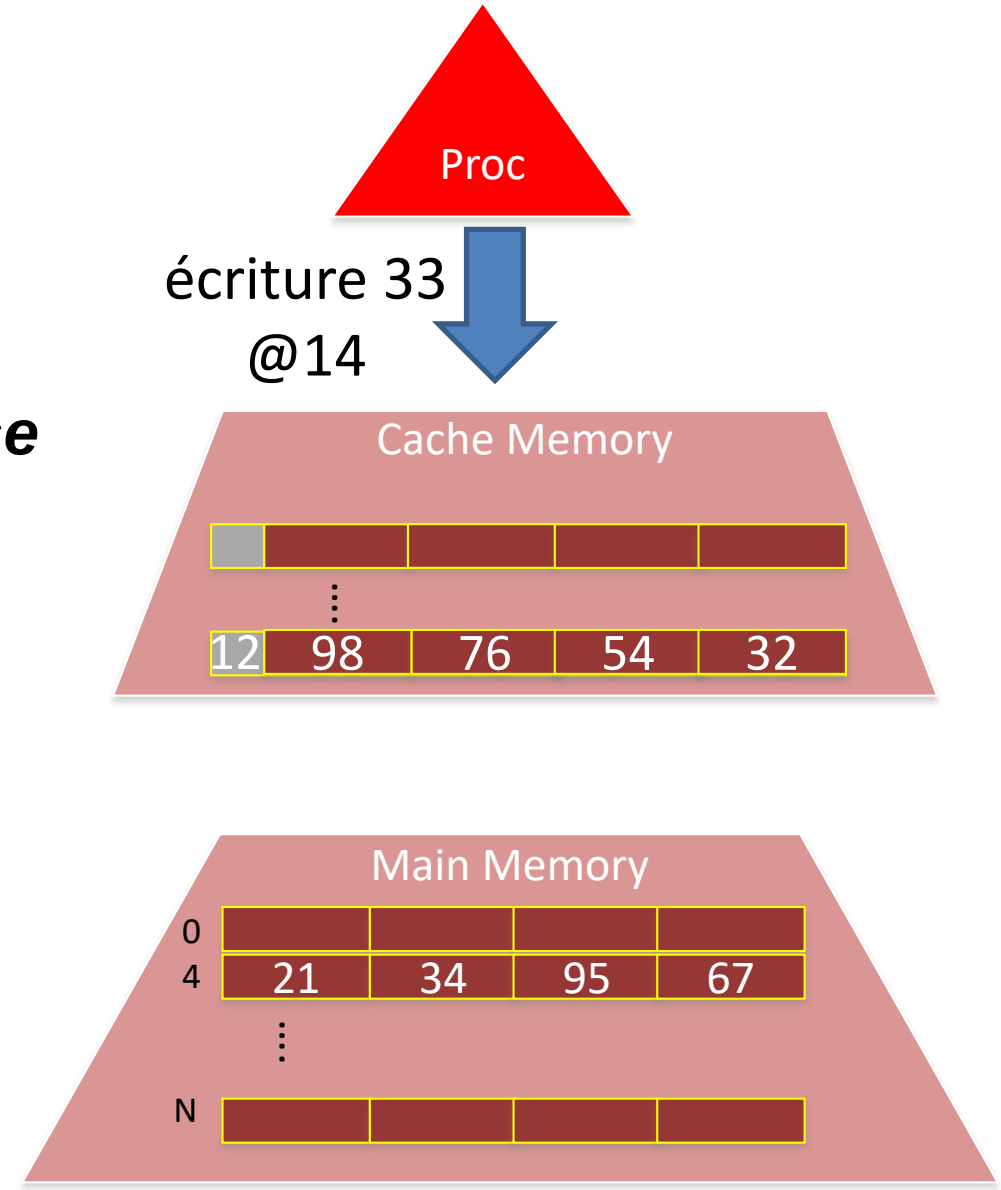
- ***le coût est proche d'une lecture sans cache***





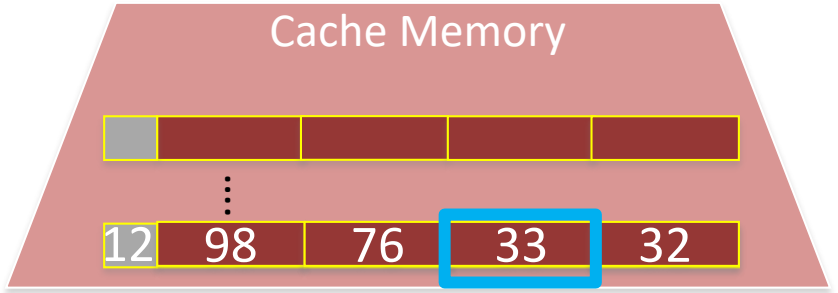
Ecriture d'une donnée par le Processeur

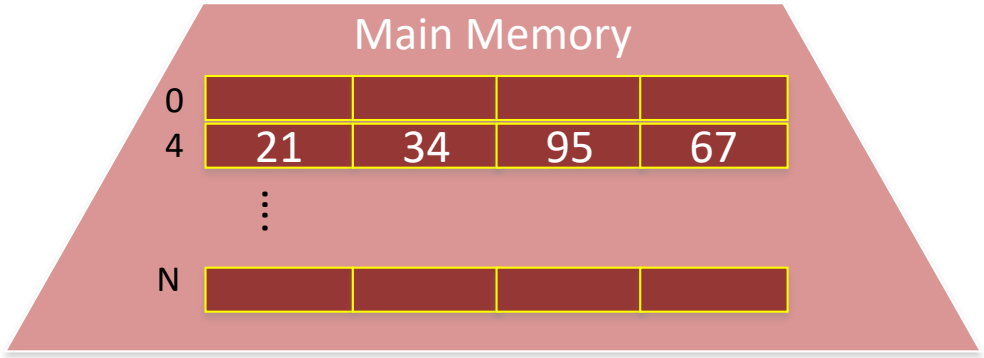
- ***Le processeur envoie l'adresse et la valeur du mot à écrire***



Ecriture d'une donnée par le Processeur

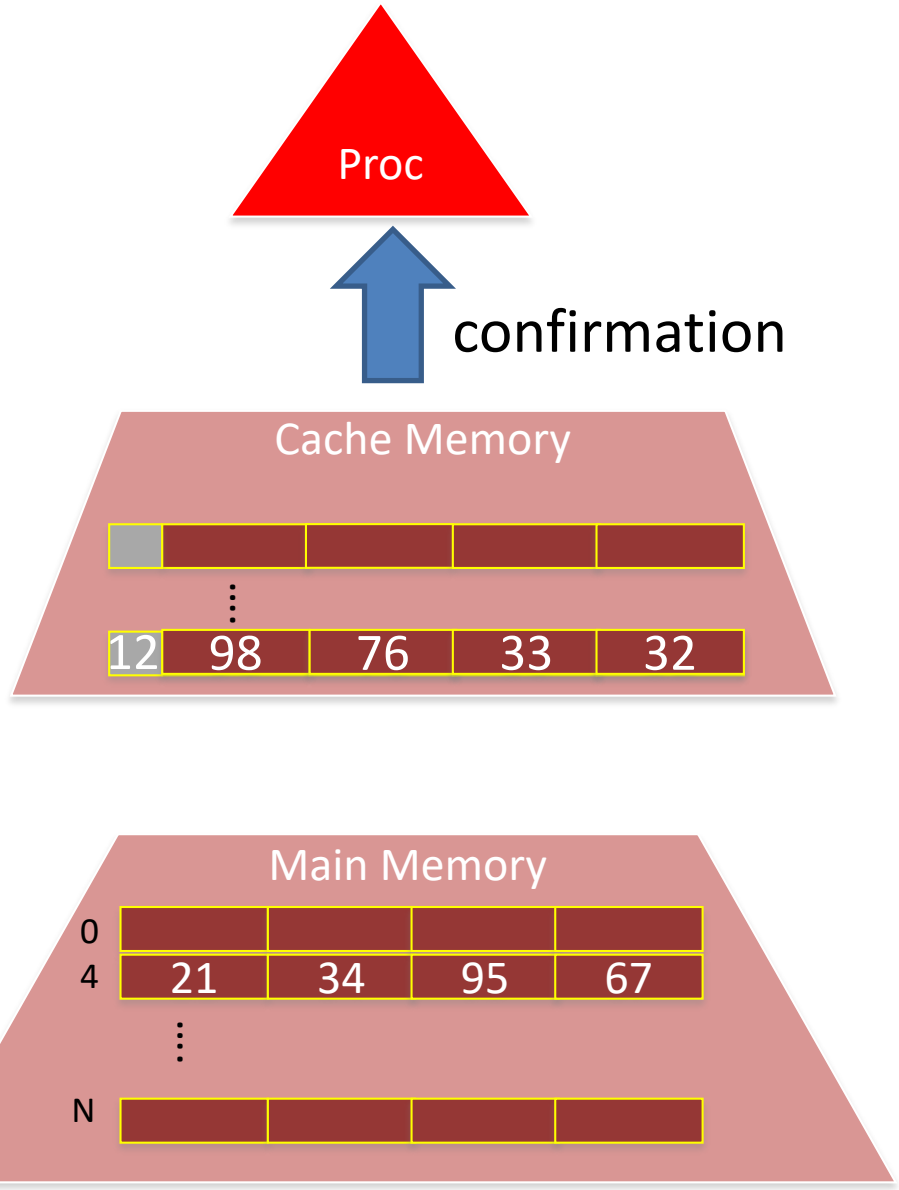
- Le cache vérifie si la donnée est présente dans le cache
- ***Si oui, la valeur est écrite dans le cache (1ns)***





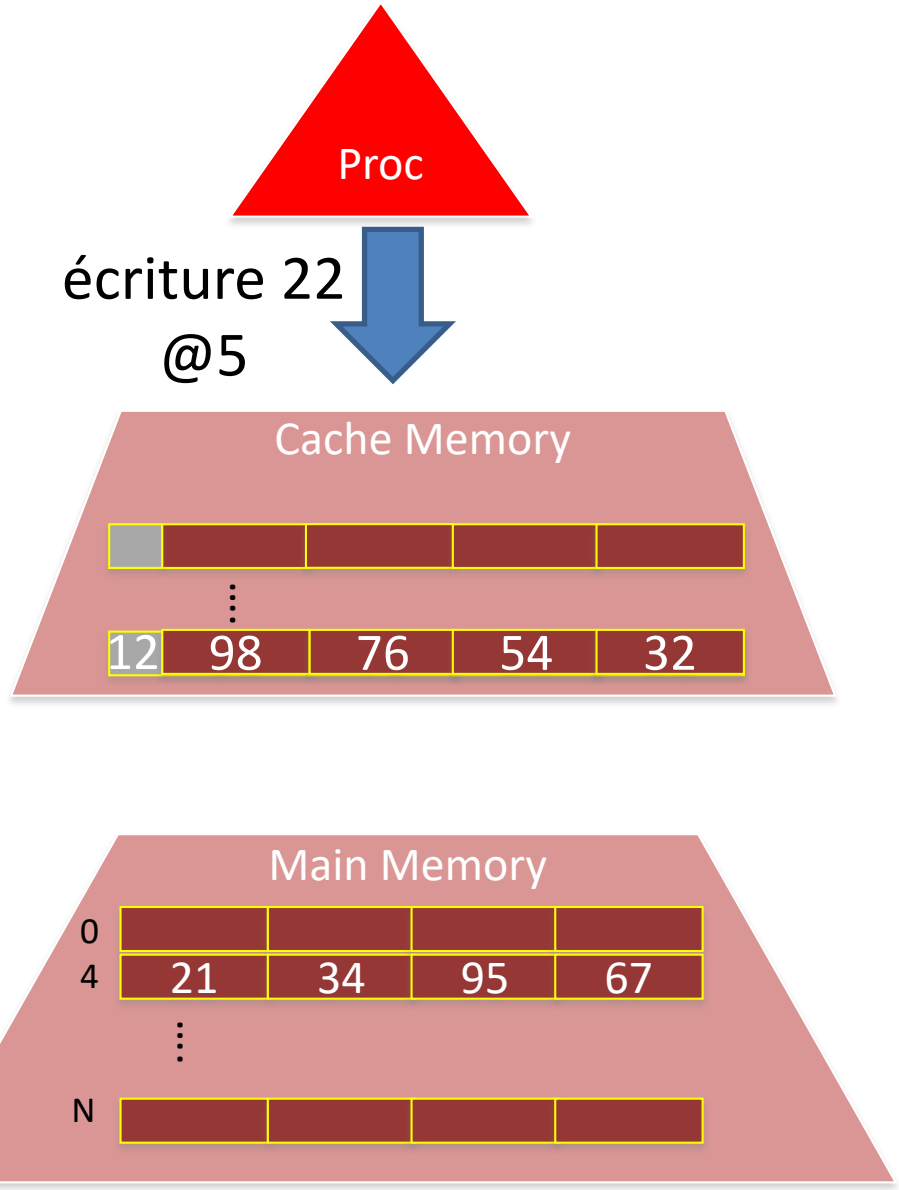
Ecriture d'une donnée par le Processeur

- ***envoie une confirmation au processeur.***



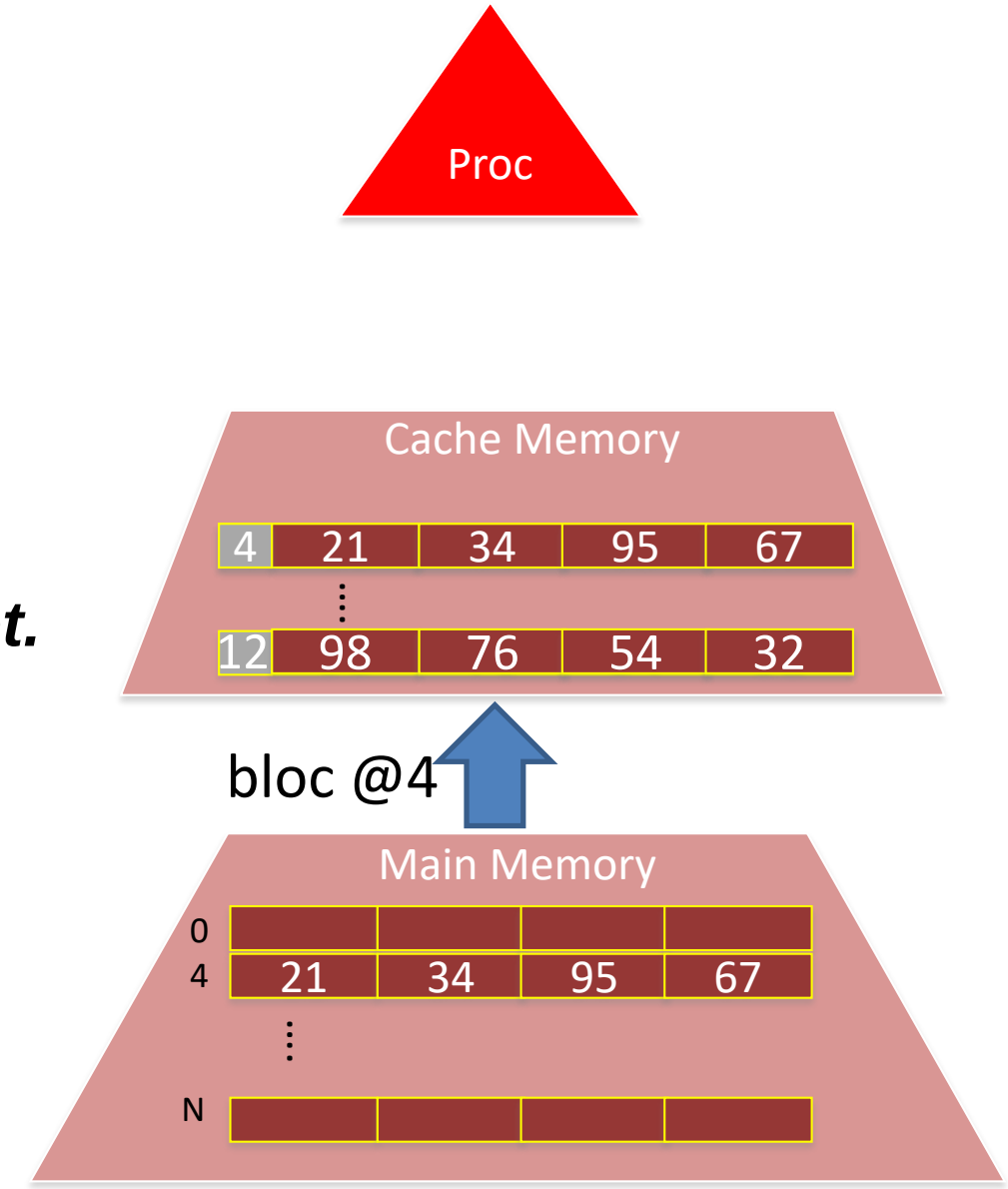
Ecriture d'une donnée par le Processeur (échec)

- ***Le processeur envoie l'adresse et la valeur du mot à écrire***



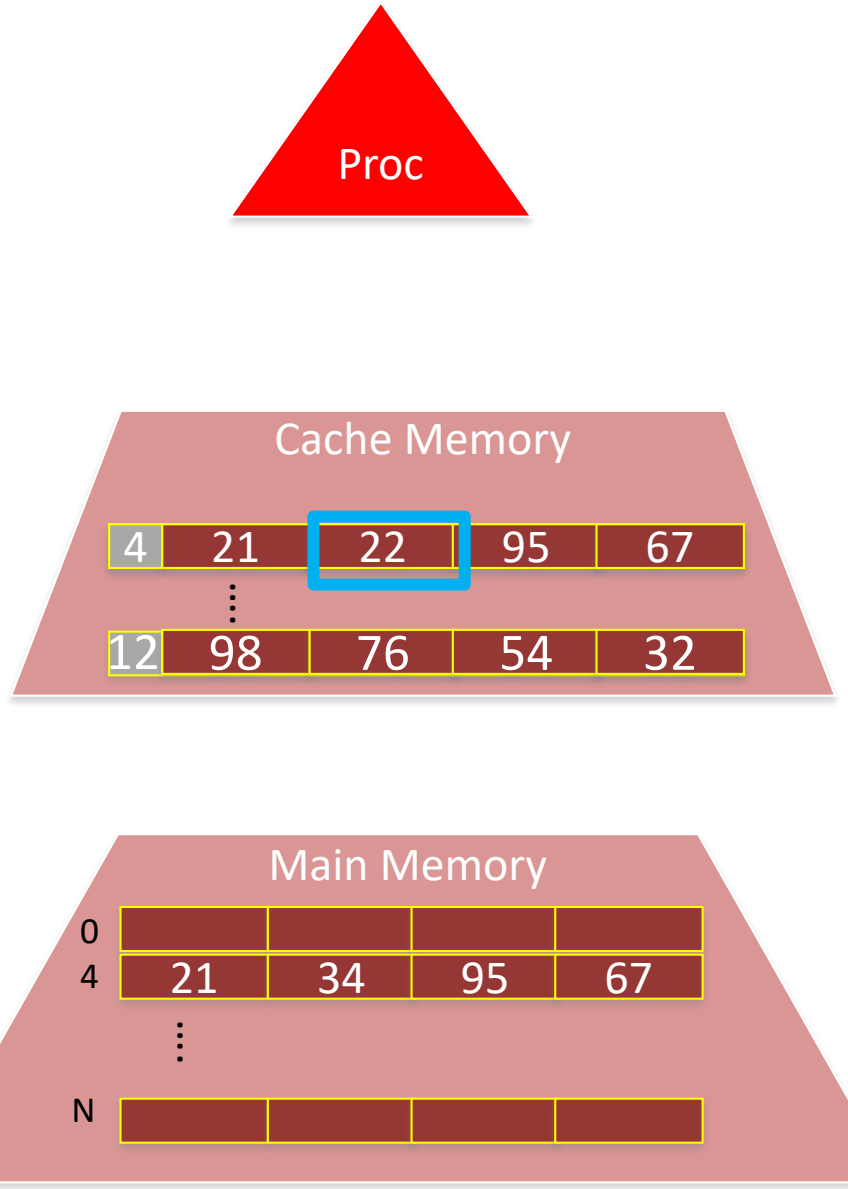
Ecriture d'une donnée par le Processeur (échec)

- Le cache vérifie si la donnée est présente dans le cache
- ***Si non, il faut charger un bloc de la mémoire vers le cache (100ns) puis écrire dans le mot.***



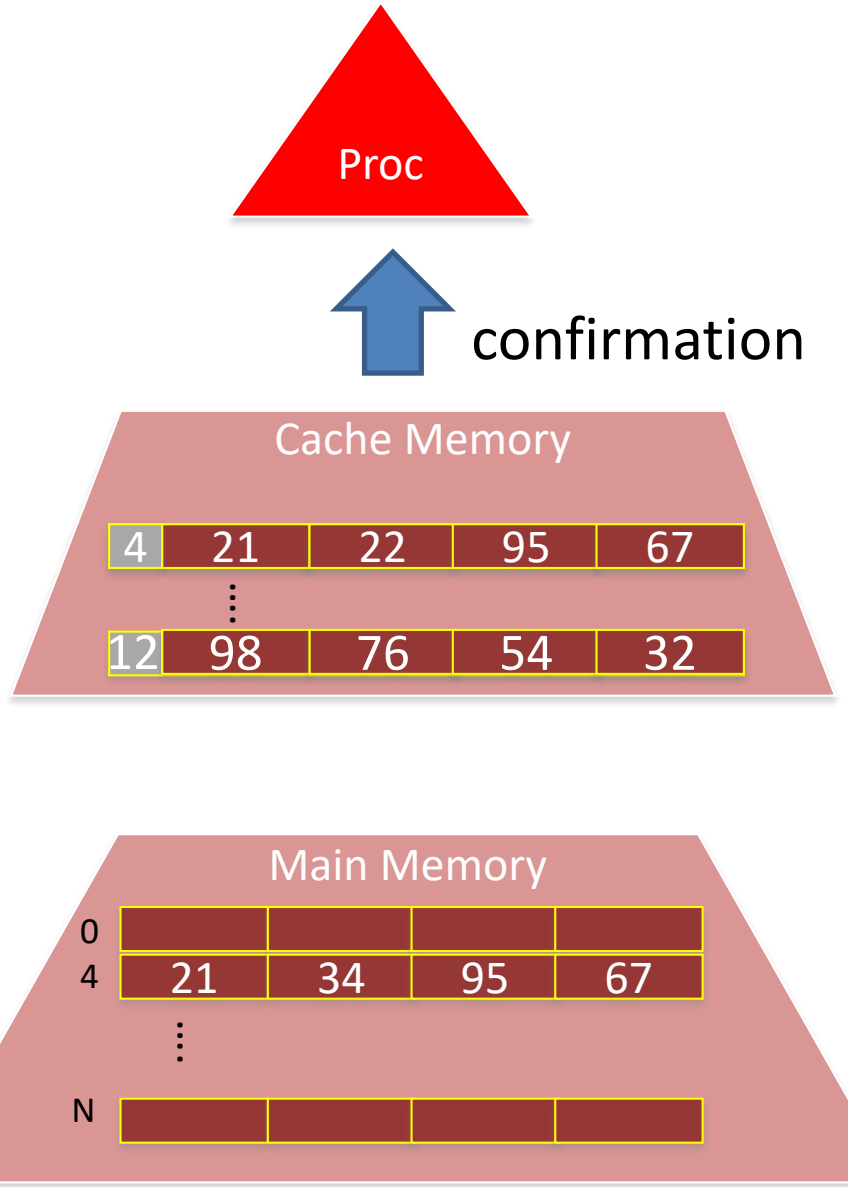
Ecriture d'une donnée par le Processeur (échec)

- Le cache vérifie si la donnée est présente dans le cache
- ***Si non, il faut charger un bloc de la mémoire vers le cache (100ns) puis écrire dans le mot.***



Ecriture d'une donnée par le Processeur (échec)

- ***envoie une confirmation au processeur.***



Que faire si le cache est plein ?

- Il faut remplacer un bloc
- Lequel ?
- Une solution courante: celui qui a été utilisé le moins récemment (LRU = Least Recently Used)

► Ne pas oublier de recopier le bloc en mémoire
le cache contient les données les plus récentes des variables

Suite du cours:

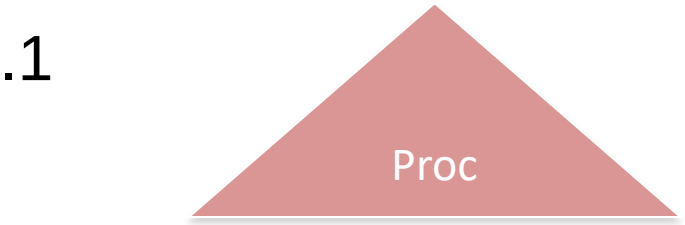
- **Principe du cache**: une mémoire ***on-chip*** = d'accès rapide pour le processeur qui lui donne d'une mémoire grande et rapide
- **Fonctionnement du cache** avec le processeur et la mémoire centrale
- **Exemple:** additionner les nombres jusqu'à n
- Le compromis entre ***localité spatiale*** et ***localité temporelle***
- Impact de l'organisation de la mémoire sur les performances d'un algorithme: exemple d'un parcours de matrice

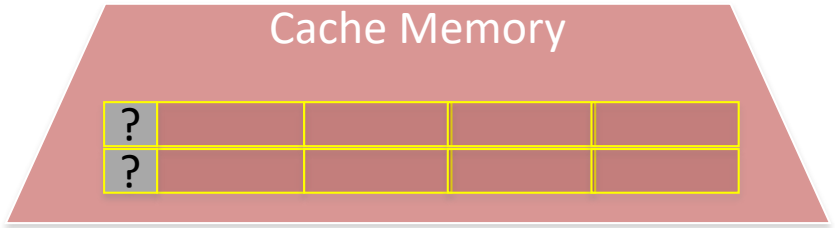
Exemple: Somme des nombres jusqu'à n

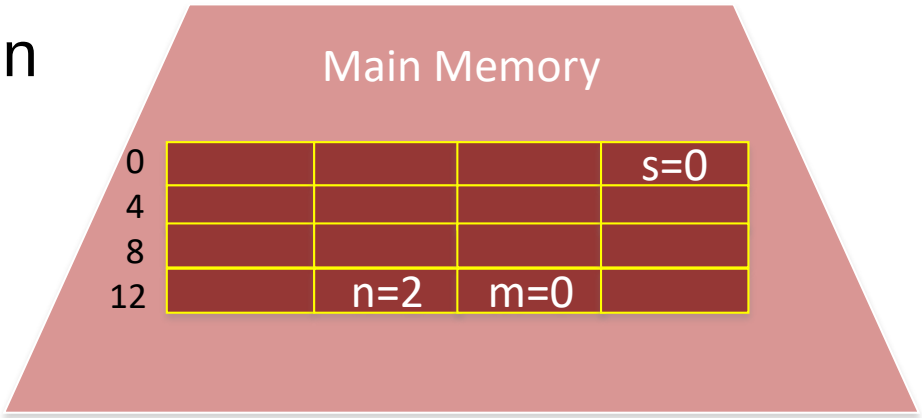
Exemple proche de la leçon III.1
Supposons:

- Cache avec 2 blocs
- Mémoire avec 4 blocs
- s @3
- n @13
- m @14
- s et m initialement 0
- n initialement 2

Somme des nombres de 0 à n
Tant que n > 0
    m ← m + n
    n ← n – 1
s ← m







Exemple: Somme des nombres jusqu'à n

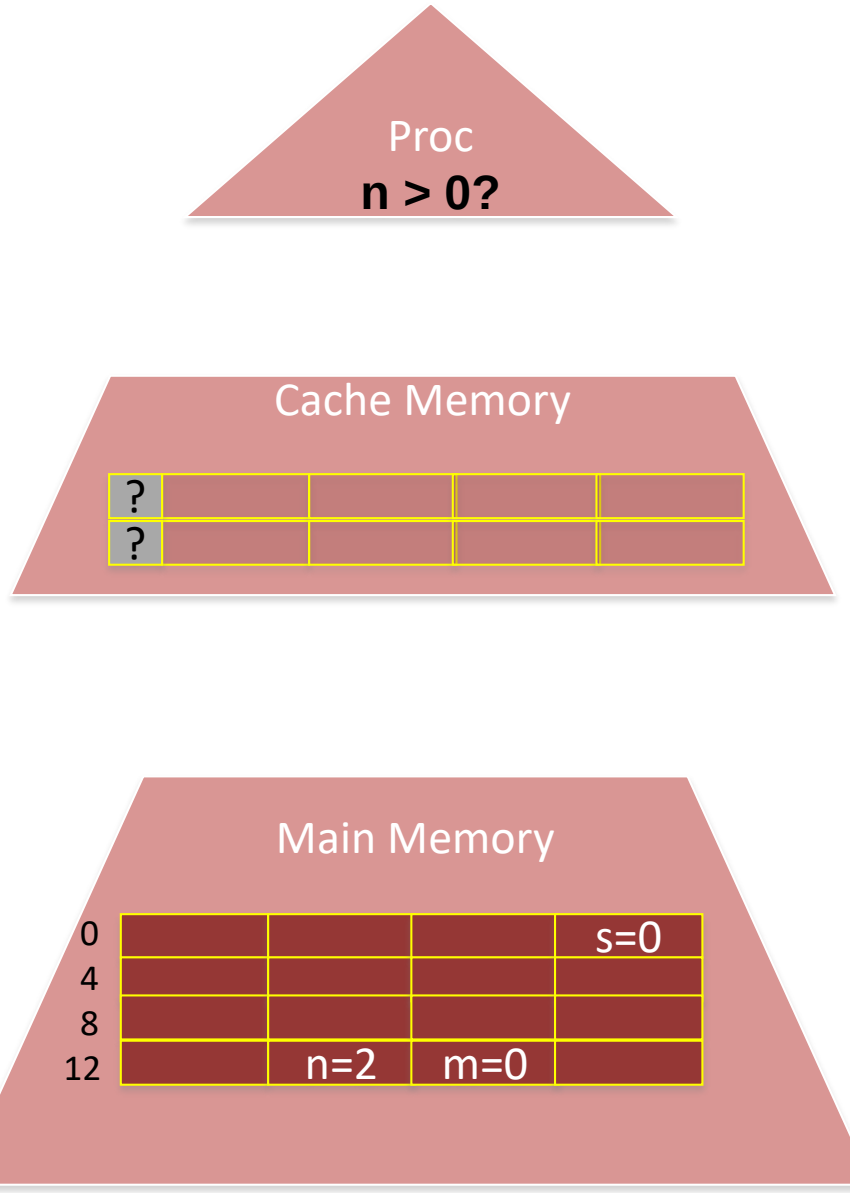
Algorithme:

Tant que n > 0
    m ← m + n
    n ← n – 1

s ← m

Actions du proc:

Pas 1: n > 0?



Exemple: Somme des nombres jusqu'à n

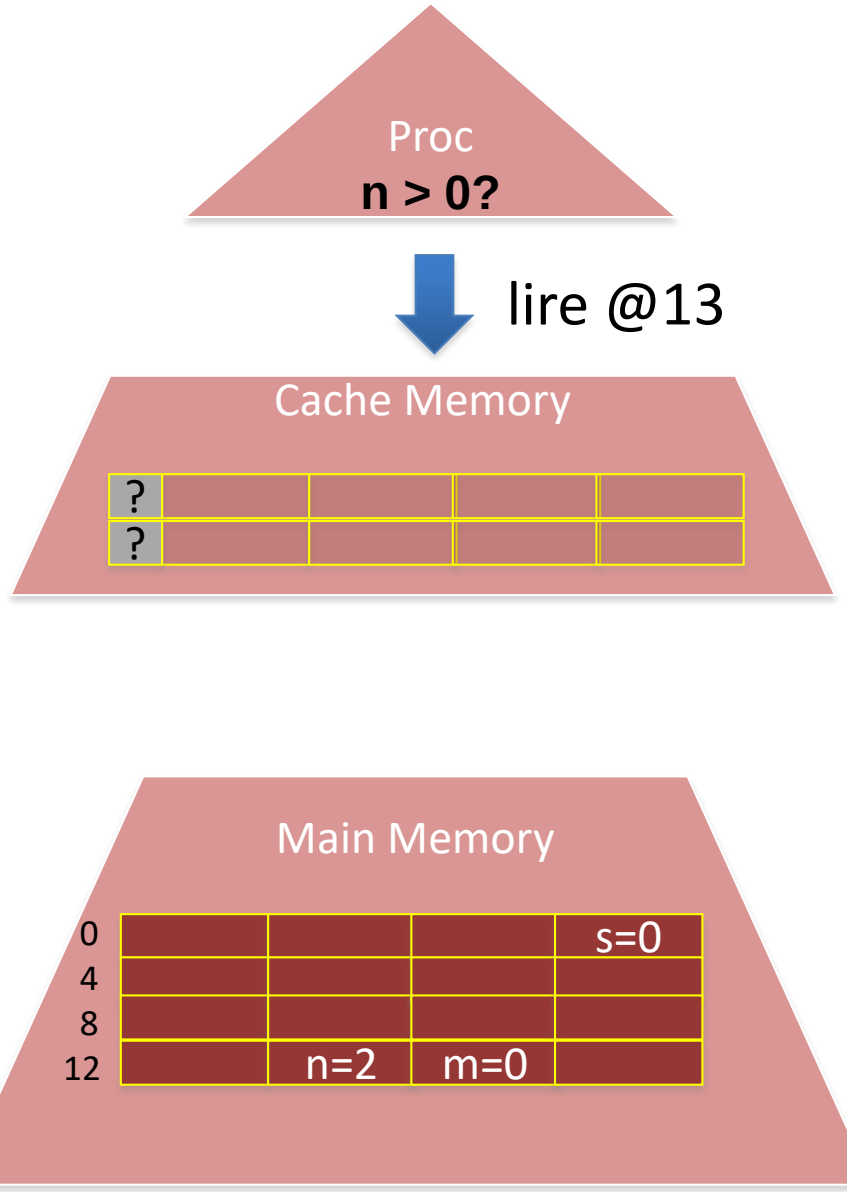
Algorithme:

Tant que n > 0
    m ← m + n
    n ← n – 1

s ← m

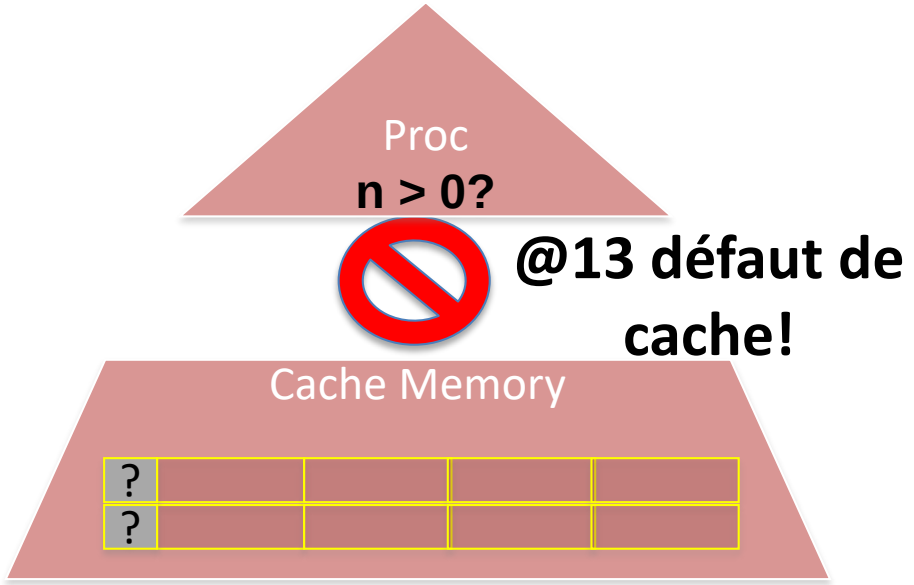
Actions du proc :

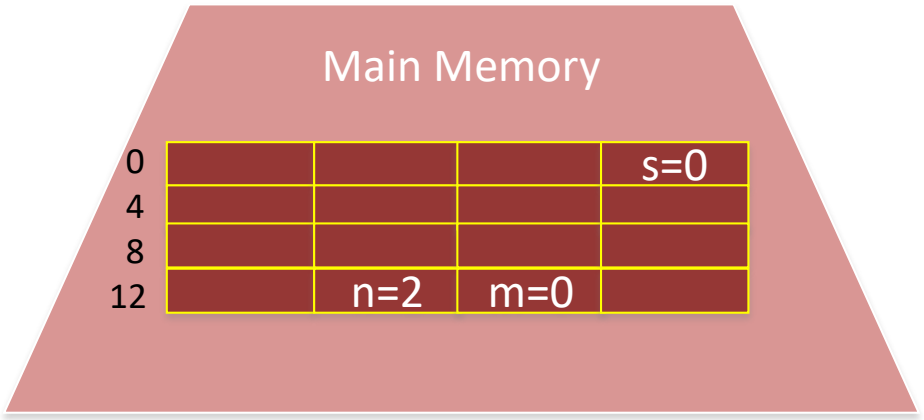
Pas 1: n > 0?
Pas 2: lire @13



Exemple: Somme des nombres jusqu'à n

Algorithme:	Actions du proc :
Tant que n > 0	Pas 1: n > 0?
m ← m + n	Pas 2: lire @13
n ← n – 1	Pas 3: @13 pas en cache (défaut de cache!)
s ← m	





Exemple: Somme des nombres jusqu'à n

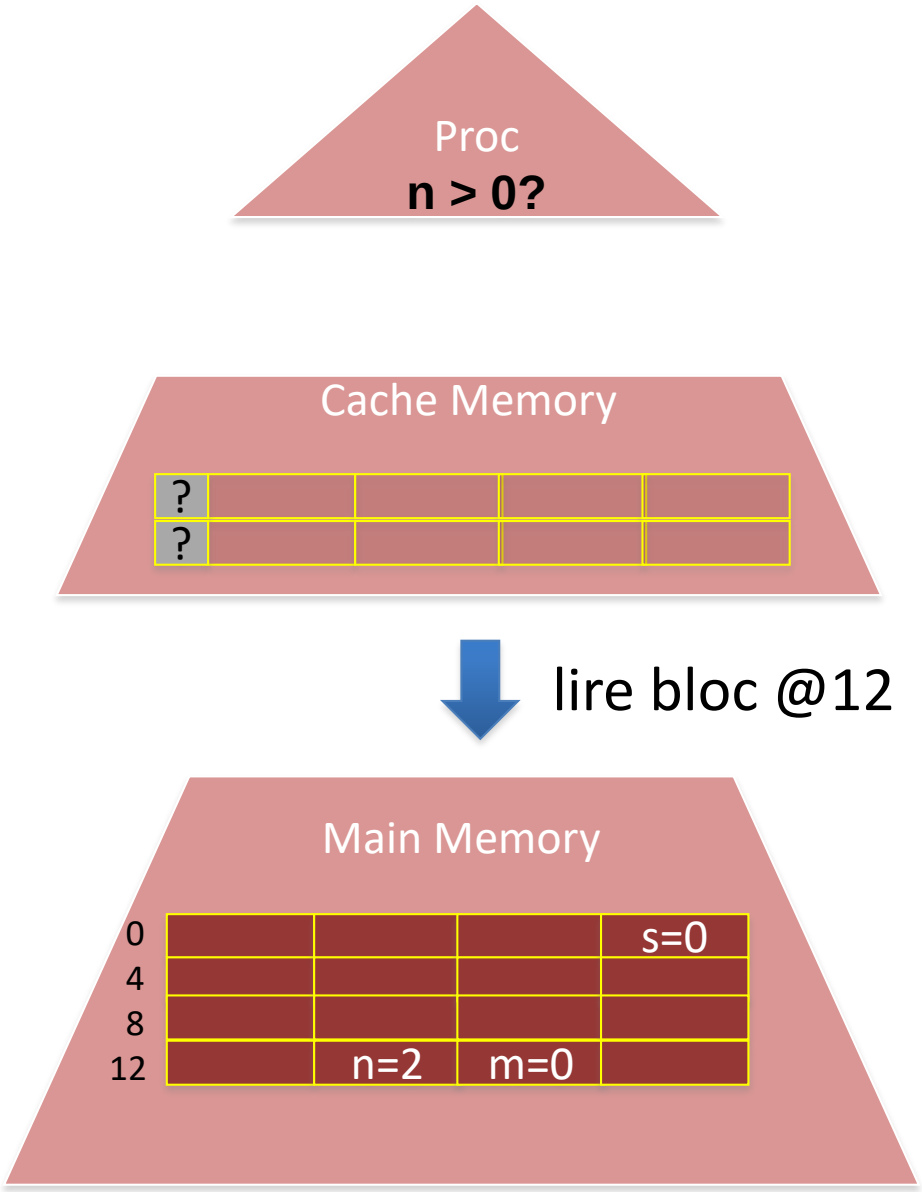
Algorithme:

Tant que n > 0
    m ← m + n
    n ← n – 1

s ← m

Actions du proc :

Pas 1: n > 0?
Pas 2: lire @13
Pas 3: @13 pas en cache
(défaut de cache!)
Pas 4: lire bloc @12



Exemple: Somme des nombres jusqu'à n

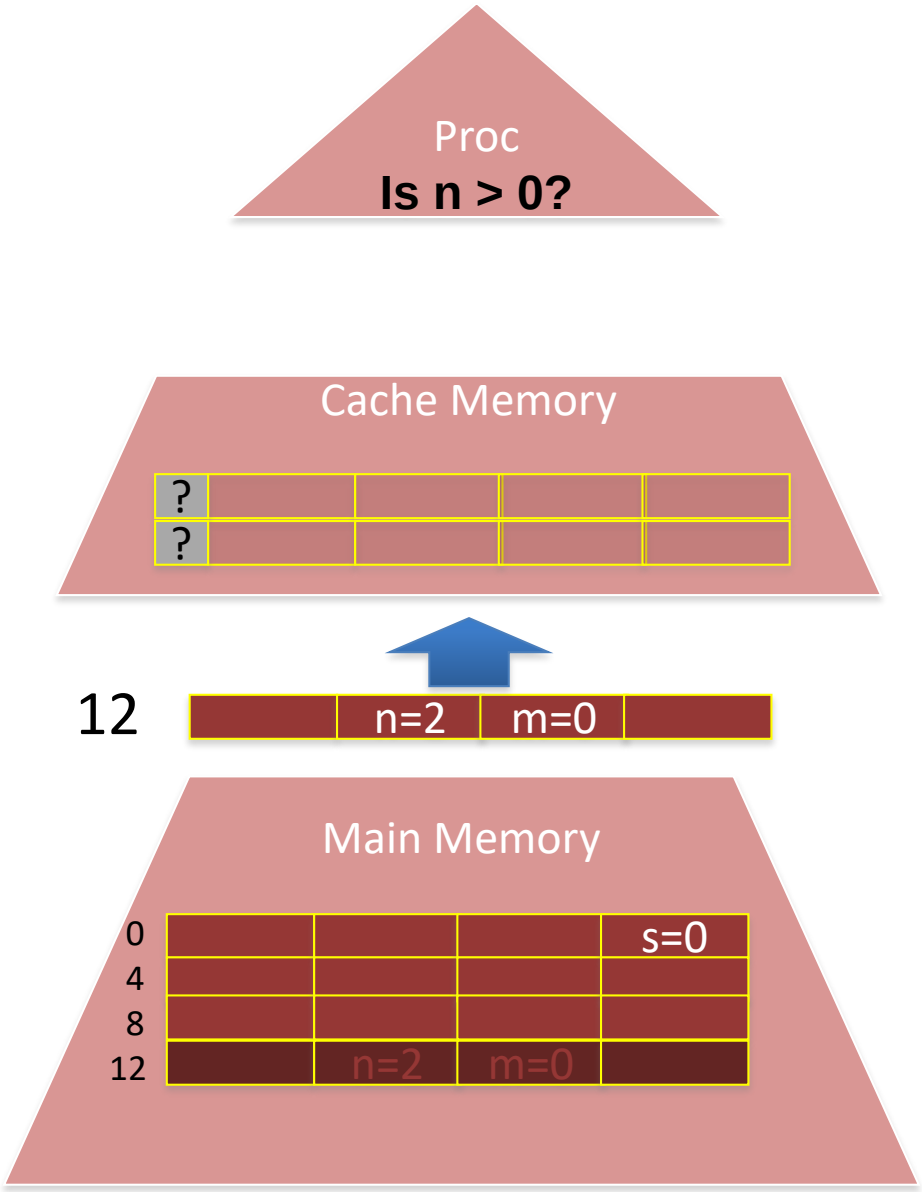
Algorithme:

Tant que n > 0
    m ← m + n
    n ← n – 1

s ← m

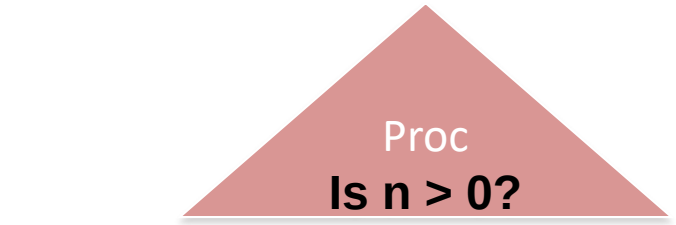
Actions du proc :

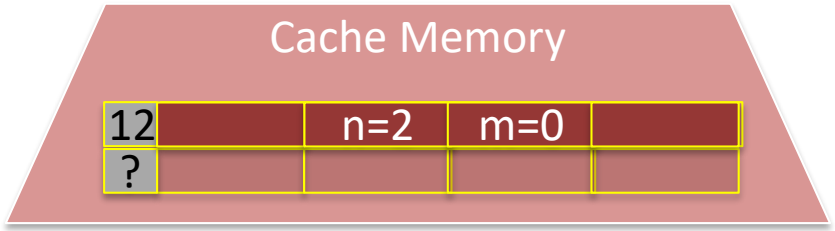
Pas 1: n > 0?
Pas 2: lire @13
Pas 3: @13 pas en cache
(défaut de cache!)
Pas 4: lire bloc @12

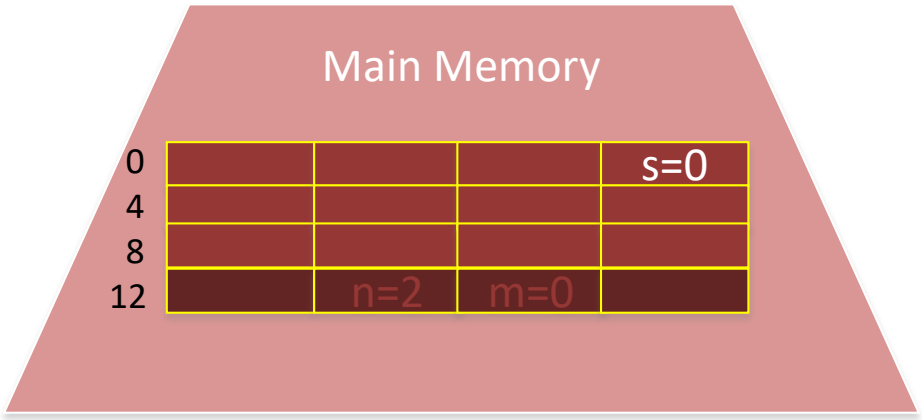


Exemple: Somme des nombres jusqu'à n

Algorithme:	Actions du proc :
Tant que n > 0	Pas 1: n > 0?
m ← m + n	Pas 2: lire @13
n ← n – 1	Pas 3: @13 pas en cache (défaut de cache!)
s ← m	Pas 4: lire bloc @12
	Pas 5: placer bloc @12

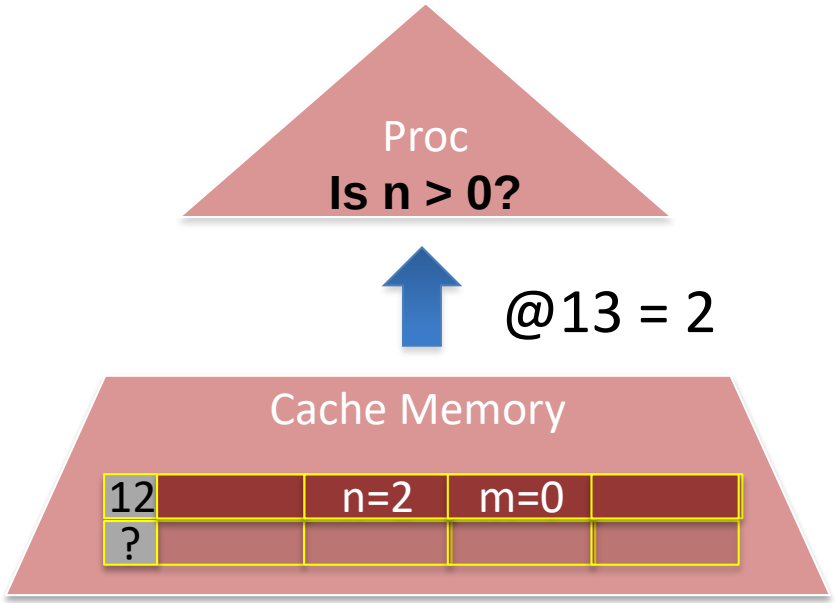


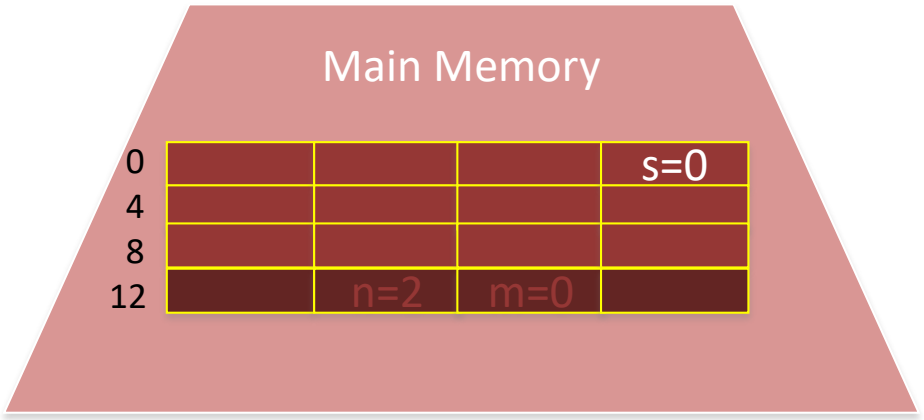




Exemple: Somme des nombres jusqu'à n

Algorithme:	Actions du proc :
Tant que n > 0	Pas 1: n > 0?
m ← m + n	Pas 2: lire @13
n ← n – 1	Pas 3: @13 pas en cache (défaut de cache!)
s ← m	Pas 4: lire bloc @12
	Pas 5: placer bloc @12
	Pas 6: retourner 2





Exemple: Somme des nombres jusqu'à n

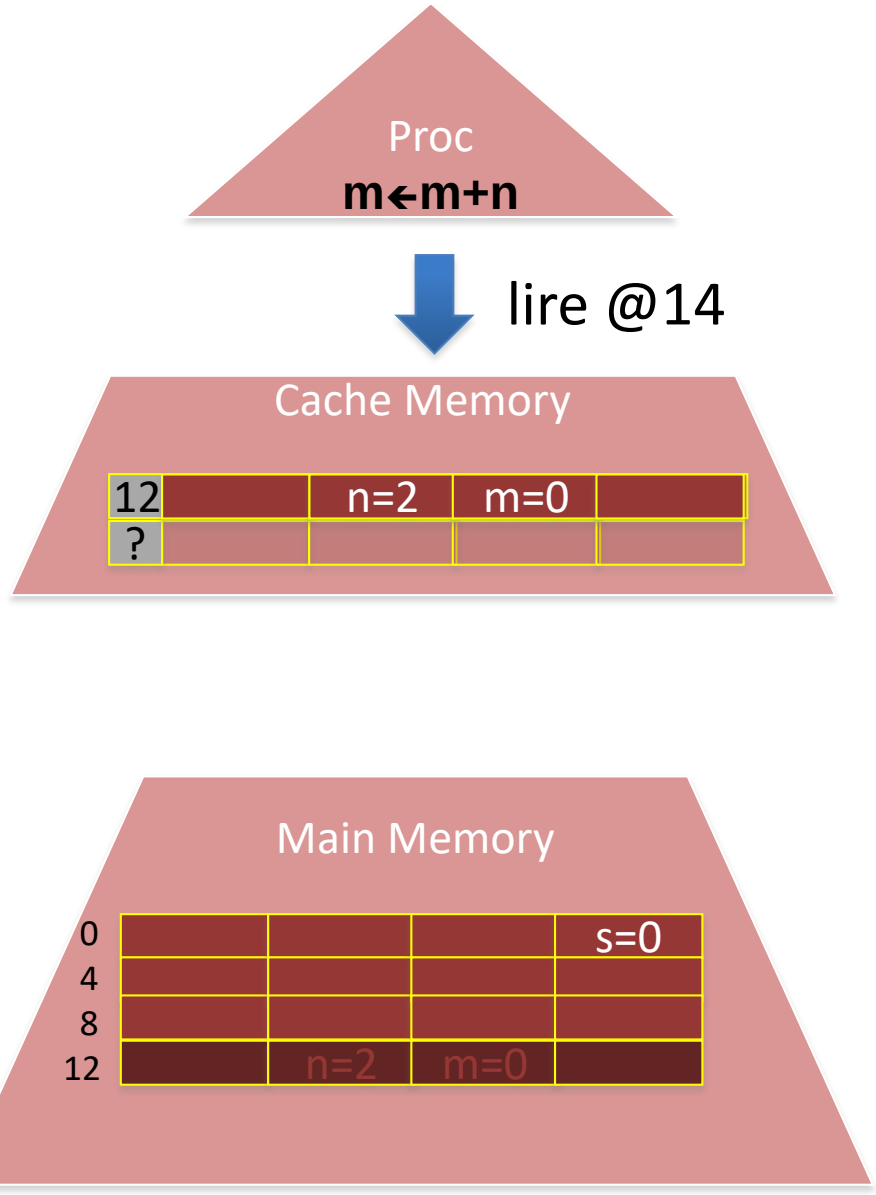
Algorithme:

Tant que n > 0
    m ← m + n
    n ← n – 1

s ← m

Actions du proc :

Pas 1: n > 0?
Pas 2: lire @13
Pas 3: @13 pas en cache
(défaut de cache!)
Pas 4: lire bloc @12
Pas 5: placer bloc @12
Pas 6: retourner 2
Pas 7: lire @14



Exemple: Somme des nombres jusqu'à n

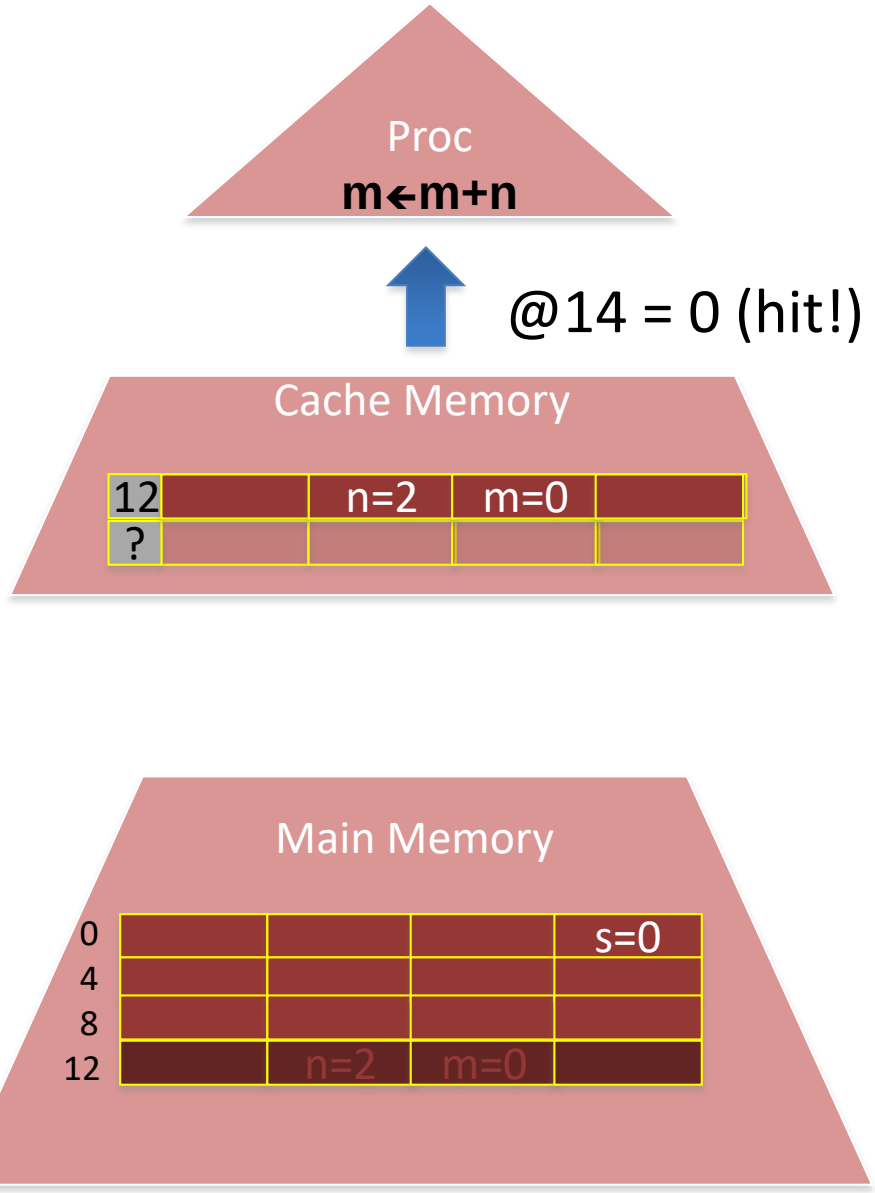
Algorithme:

Tant que n > 0
    m ← m + n
    n ← n – 1

s ← m

Actions du proc :

Pas 1: n > 0?
Pas 2: lire @13
Pas 3: @13 pas en cache
(défaut de cache!)
Pas 4: lire bloc @12
Pas 5: placer bloc @12
Pas 6: retourner 2
Pas 7: lire @14
(en cache!)
Pas 8: retourner 0



Exemple: Somme des nombres jusqu'à n

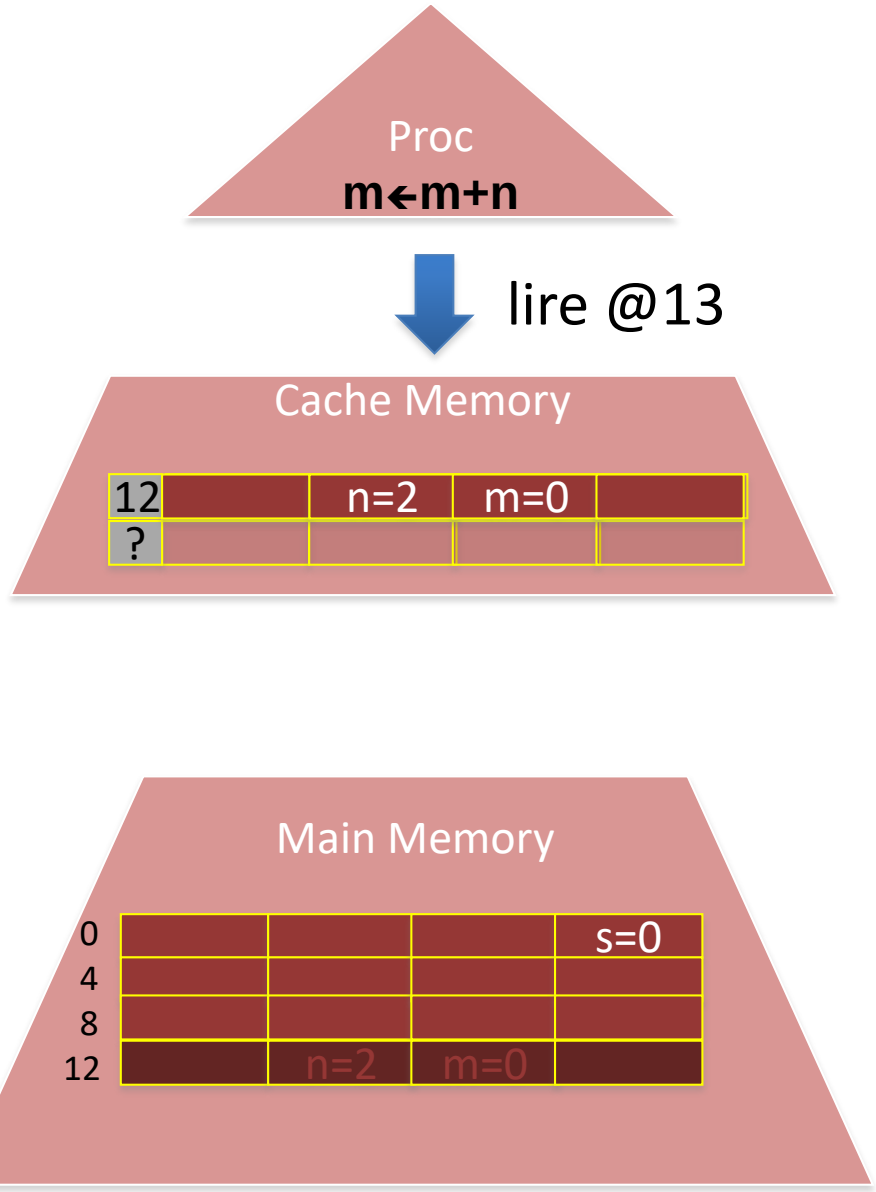
Algorithme:

Tant que n > 0
    m ← m + n
    n ← n – 1

s ← m

Actions du proc :

Pas 1: n > 0?
Pas 2: lire @13
Pas 3: @13 pas en cache
(défaut de cache!)
Pas 4: lire bloc @12
Pas 5: placer bloc @12
Pas 6: retourner 2
Pas 7: lire @14
(en cache!)
Pas 8: retourner 0
Pas 9: lire @13



Exemple: Somme des nombres jusqu'à n

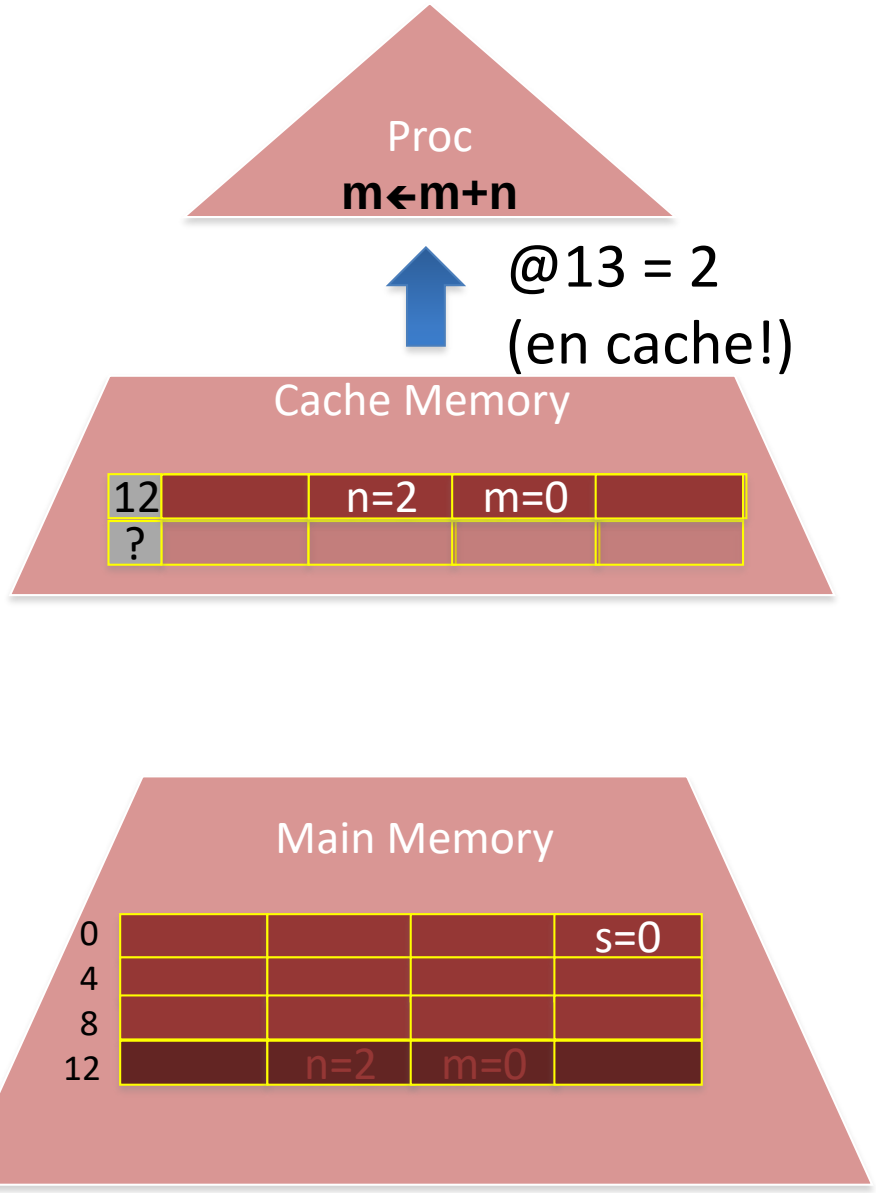
Algorithme:

Tant que n > 0
    m ← m + n
    n ← n – 1

s ← m

Actions du proc :

Pas 1: n > 0?
Pas 2: lire @13
Pas 3: @13 pas en cache
(défaut de cache!)
Pas 4: lire bloc @12
Pas 5: placer bloc @12
Pas 6: retourner 2
Pas 7: lire @14
(en cache!)
Pas 8: retourner 0
Pas 9: lire @13
Pas 10: retourner 2



Exemple: Somme des nombres jusqu'à n

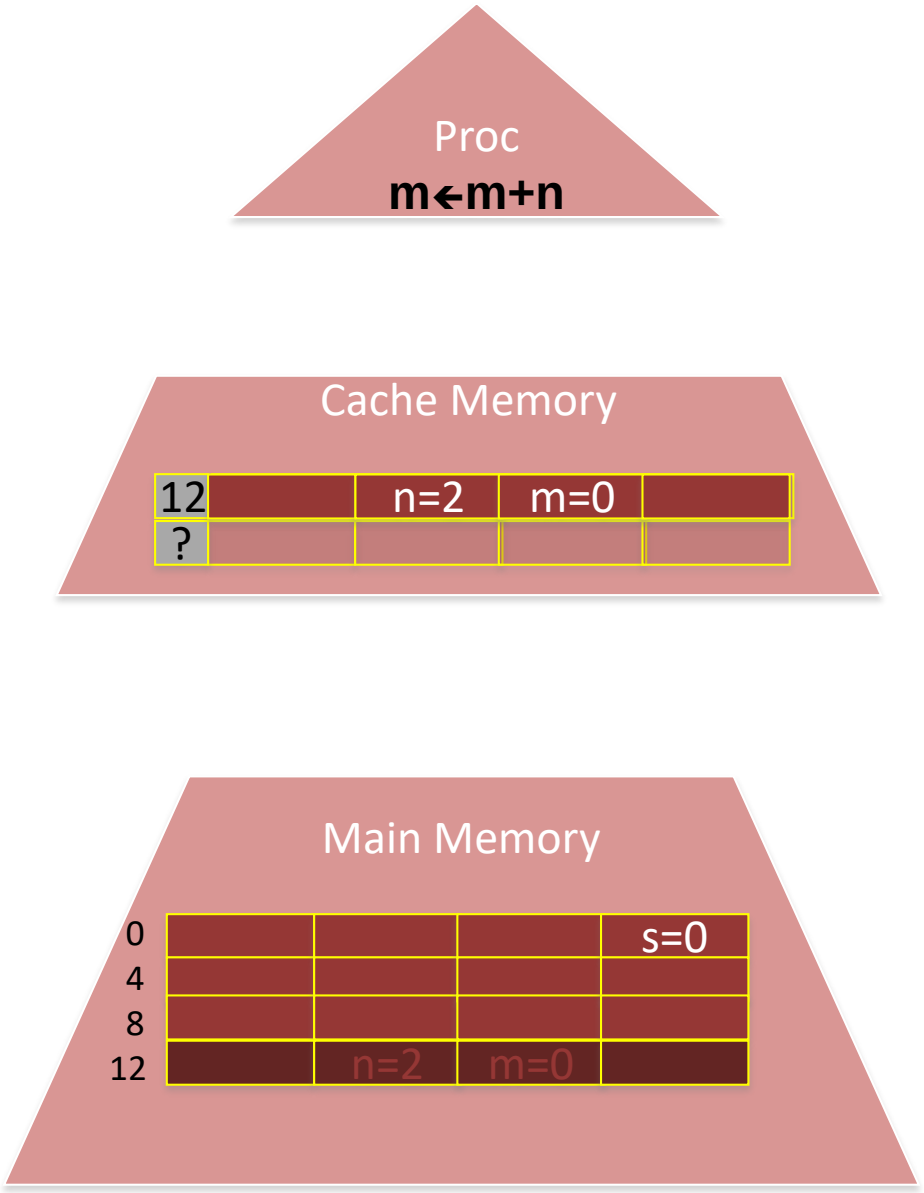
Algorithme:

Tant que n > 0
    m ← m + n
    n ← n – 1

s ← m

Actions du proc :

Pas 1: n > 0?
Pas 2: lire @13
Pas 3: @13 pas en cache
(défaut de cache!)
Pas 4: lire bloc @12
Pas 5: placer bloc @12
Pas 6: retourner 2
Pas 7: lire @14
(en cache!)
Pas 8: retourner 0
Pas 9: lire @13
Pas 10: retourner 2
Pas 11: add m, n (0 + 2)



Exemple: Somme des nombres jusqu'à n

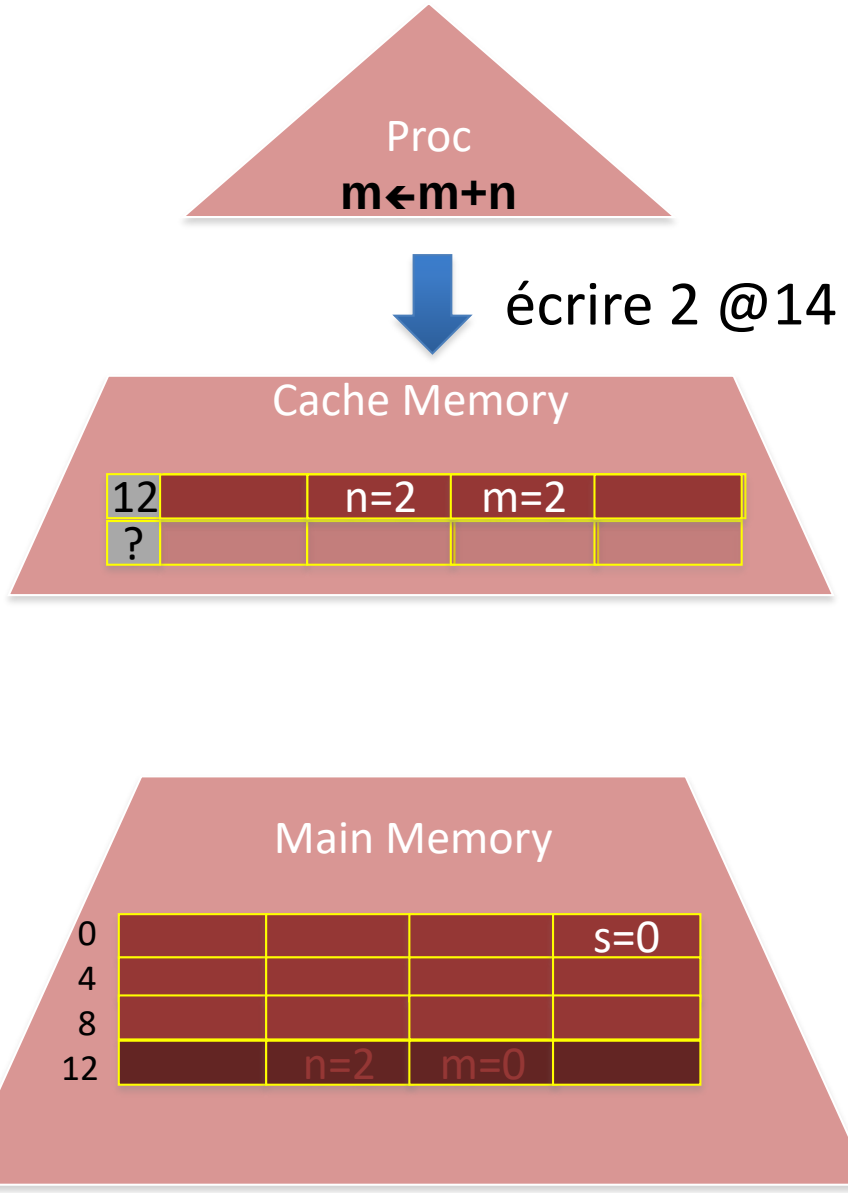
Algorithme:

Tant que n > 0
    m ← m + n
    n ← n – 1

s ← m

Actions du proc :

Pas 1: n > 0?
Pas 2: lire @13
Pas 3: @13 pas en cache
(défaut de cache!)
Pas 4: lire bloc @12
Pas 5: placer bloc @12
Pas 6: retourner 2
Pas 7: lire @14
(en cache!)
Pas 8: retourner 0
Pas 9: lire @13
Pas 10: retourner 2
Pas 11: add m, n (0 + 2)
Pas 12: écrire 2 @14



Exemple: Somme des nombres jusqu'à n

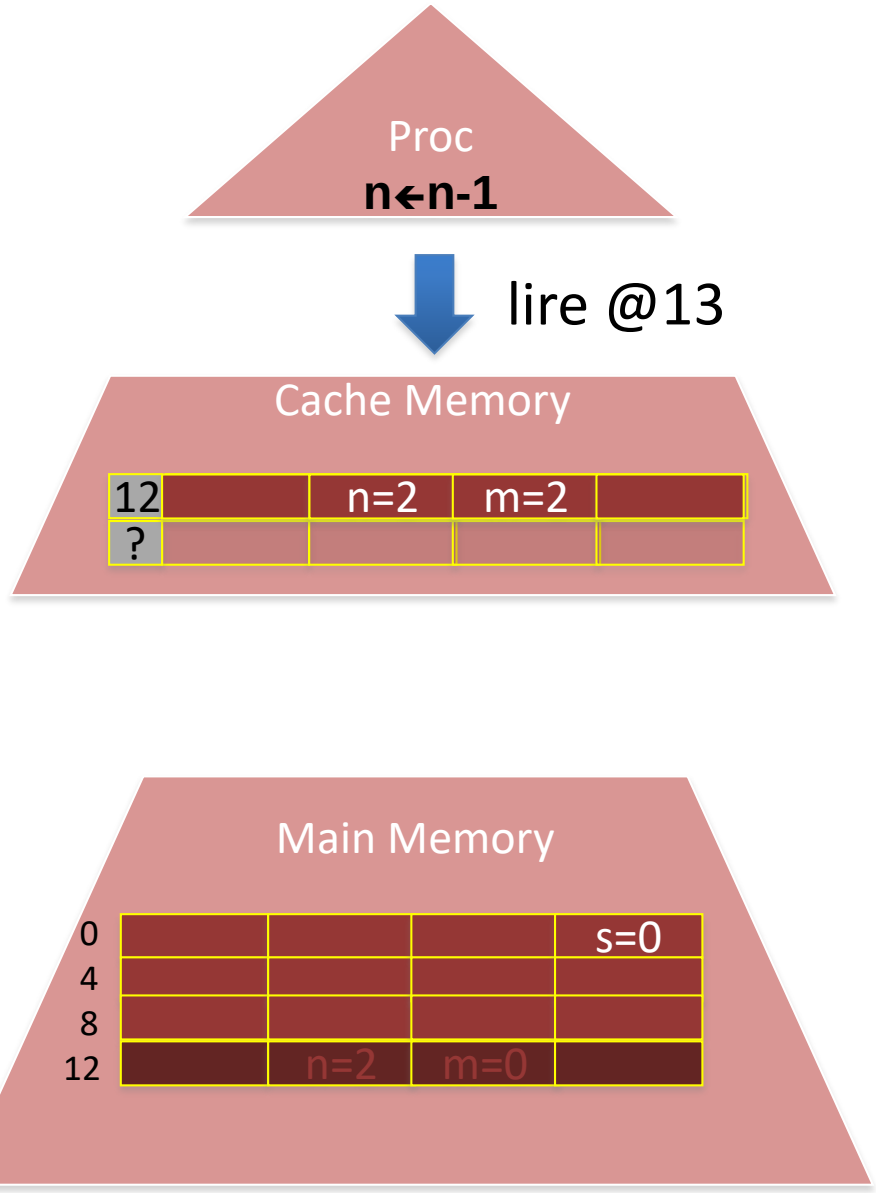
Algorithme:

Tant que n > 0
    m ← m + n
    n ← n – 1

s ← m

Actions du proc :

Pas 1: n > 0?
Pas 2: lire @13
Pas 3: @13 pas en cache
(défaut de cache!)
Pas 4: lire bloc @12
Pas 5: placer bloc @12
Pas 6: retourner 2
Pas 7: lire @14
(en cache!)
Pas 8: retourner 0
Pas 9: lire @13
Pas 10: retourner 2
Pas 11: add m, n (0 + 2)
Pas 12: écrire 2 @14
Pas 13: lire @13



Exemple: Somme des nombres jusqu'à n

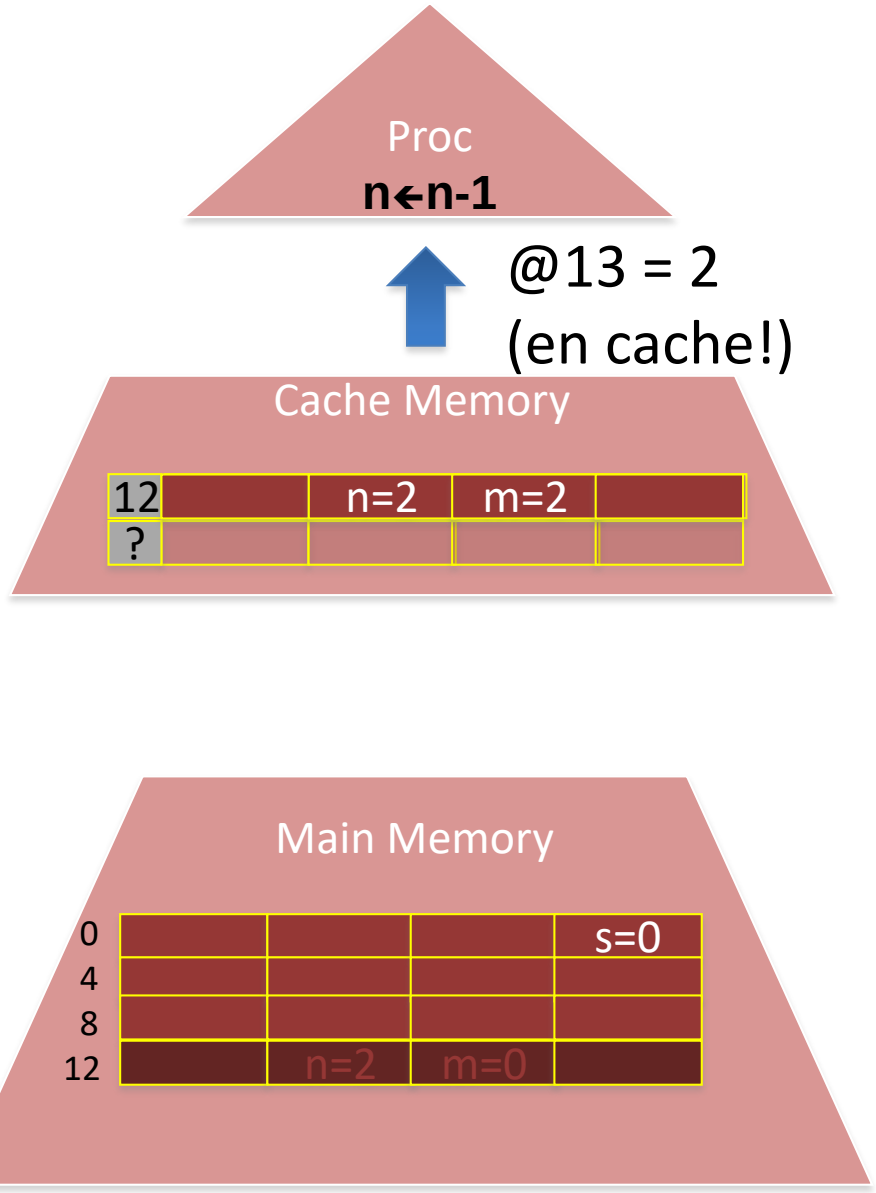
Algorithme:

Tant que n > 0
    m ← m + n
    n ← n – 1

s ← m

Actions du proc :

Pas 1: n > 0?
Pas 2: lire @13
Pas 3: @13 pas en cache
(défaut de cache!)
Pas 4: lire bloc @12
Pas 5: placer bloc @12
Pas 6: retourner 2
Pas 7: lire @14
(en cache!)
Pas 8: retourner 0
Pas 9: lire @13
Pas 10: retourner 2
Pas 11: add m, n (0 + 2)
Pas 12: écrire 2 @14
Pas 13: lire @13
Pas 14: retourner 2



Exemple: Somme des nombres jusqu'à n

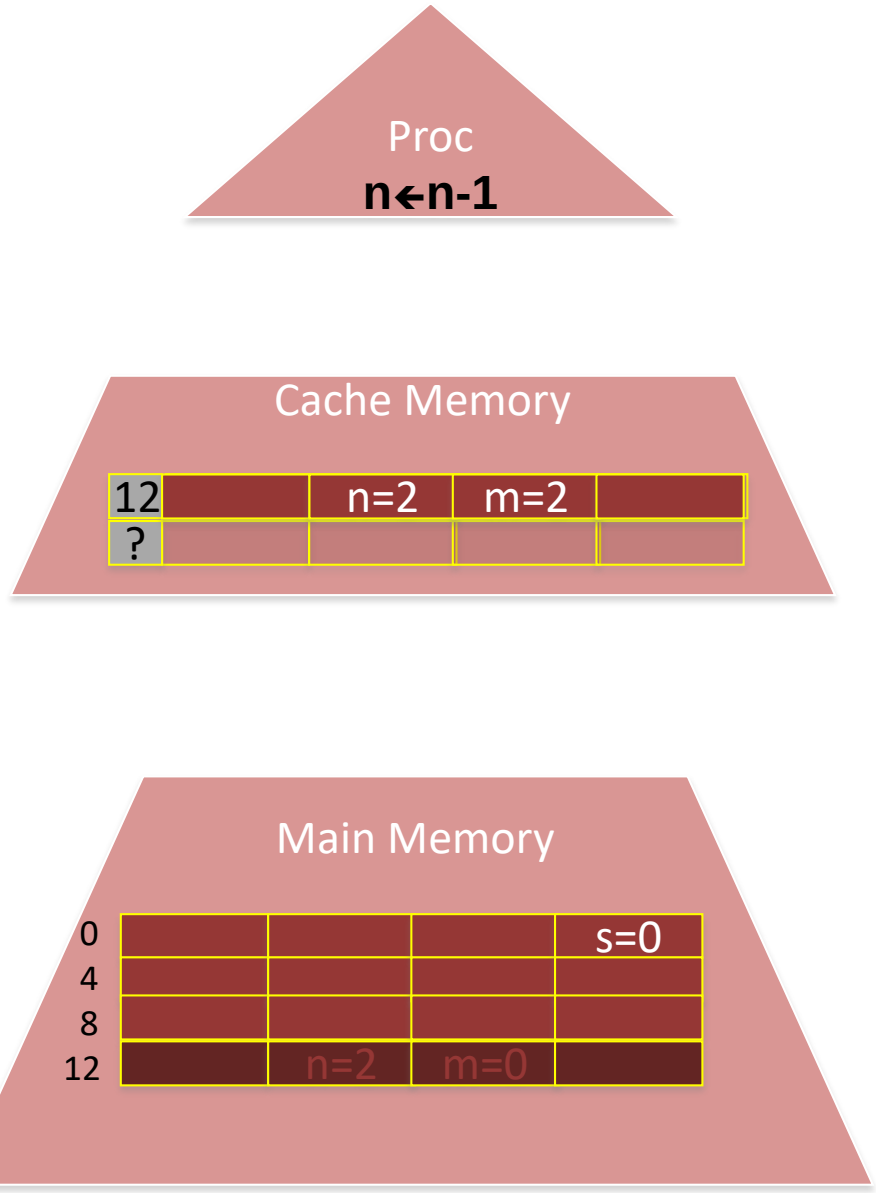
Algorithme:

Tant que n > 0
    m ← m + n
    n ← n – 1

s ← m

Actions du proc :

…
Pas 13: lire @13
Pas 14: retourner 2
Pas 15: soustraire 1, n



Exemple: Somme des nombres jusqu'à n

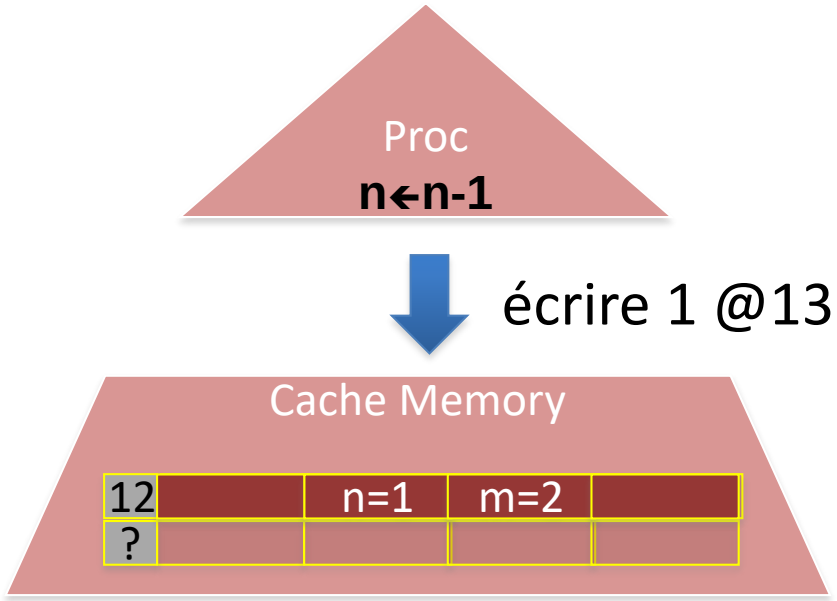
Algorithme:

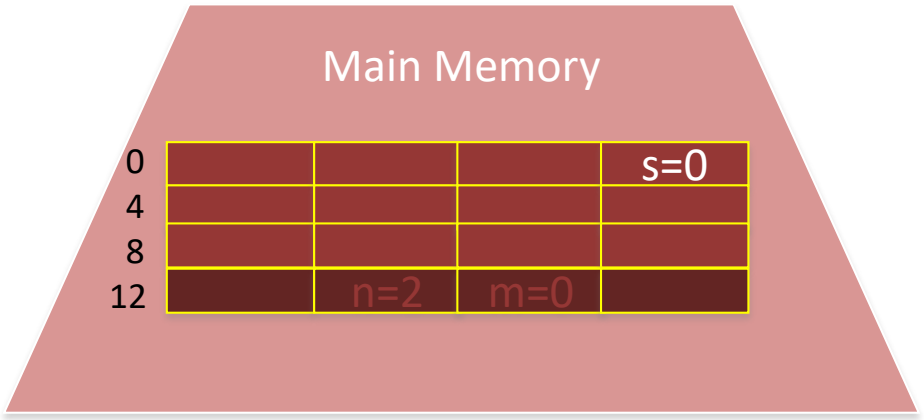
Tant que n > 0
    m ← m + n
    n ← n – 1

s ← m

Actions du proc :

…
Pas 13: lire @13
Pas 14: retourner 2
Pas 15: soustraire 1, n
Pas 16: écrire 1 @ 13





Exemple: Somme des nombres jusqu'à n

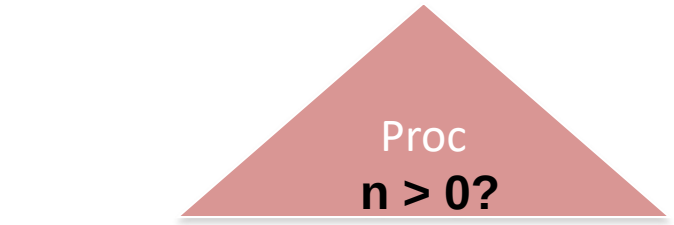
Algorithme:

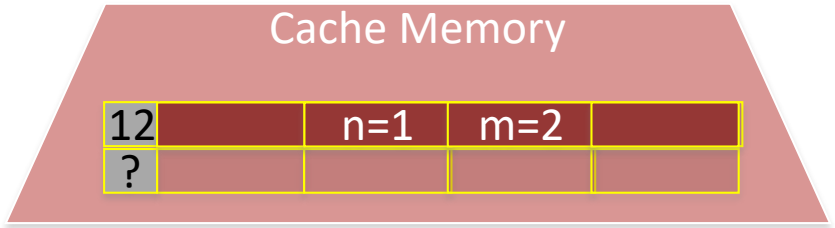
Tant que n > 0
    m ← m + n
    n ← n – 1

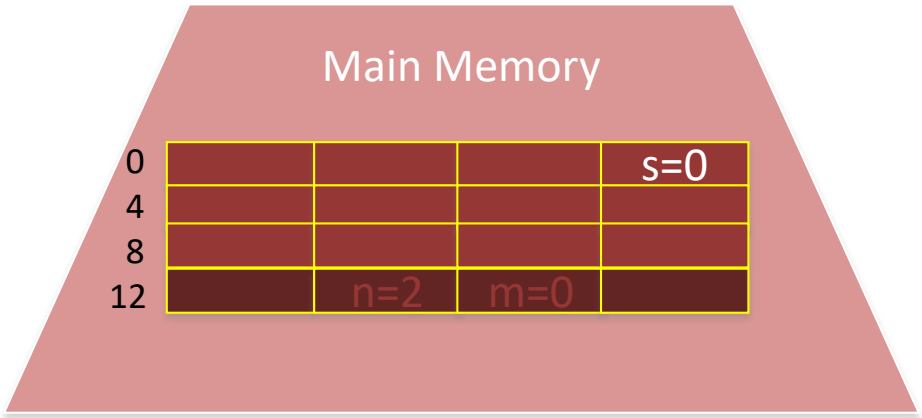
s ← m

Actions du proc :

…
Pas 13: lire @13
Pas 14: retourner 2
Pas 15: soustraire 1, n
Pas 16: écrire 1 @ 13
Pas 17: n > 0?







Exemple: Somme des nombres jusqu'à n

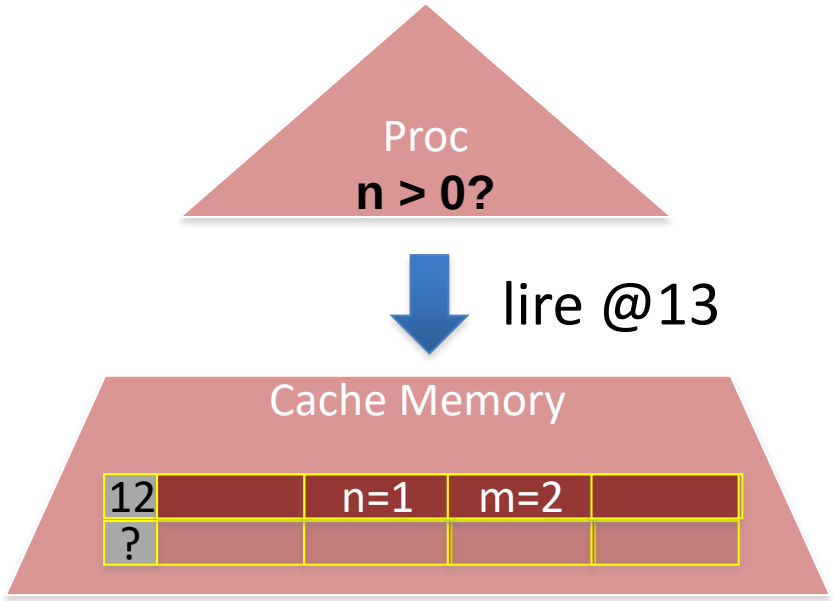
Algorithme:

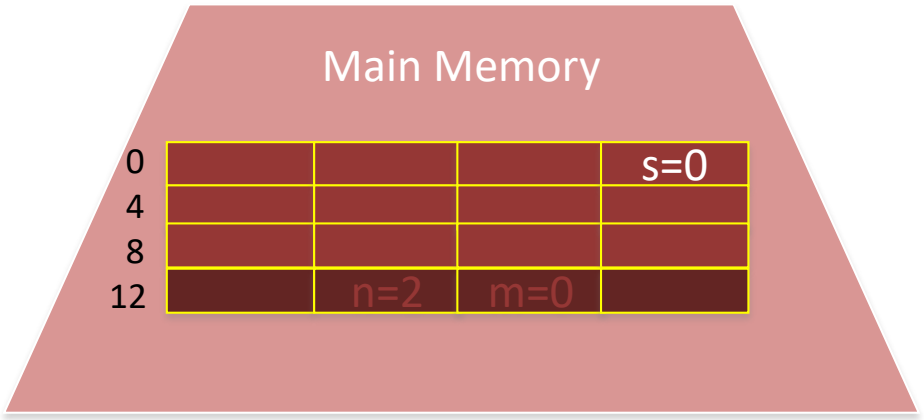
Tant que n > 0
    m ← m + n
    n ← n – 1

s ← m

Actions du proc :

…
Pas 13: lire @13
Pas 14: retourner 2
Pas 15: soustraire 1, n
Pas 16: écrire 1 @ 13
Pas 17: n > 0?
Pas 18: lire @13





Exemple: Somme des nombres jusqu'à n

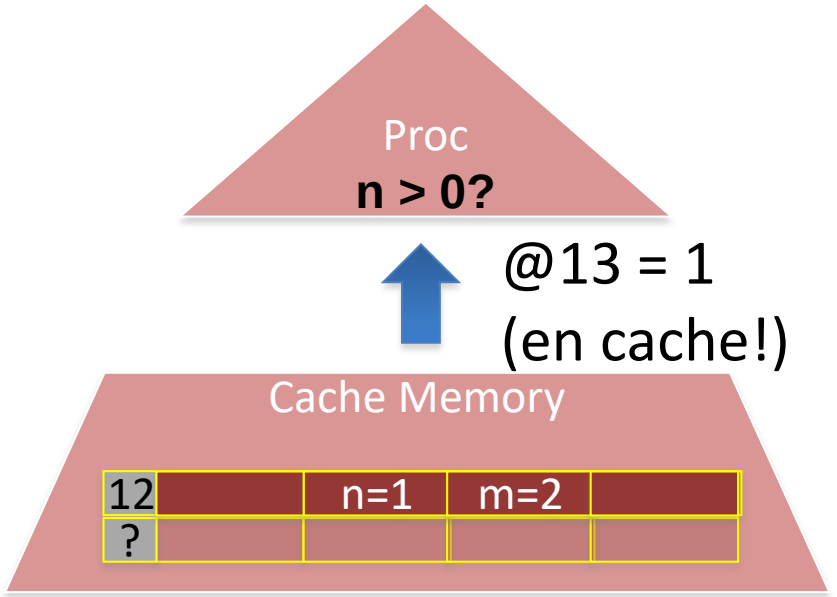
Algorithme:

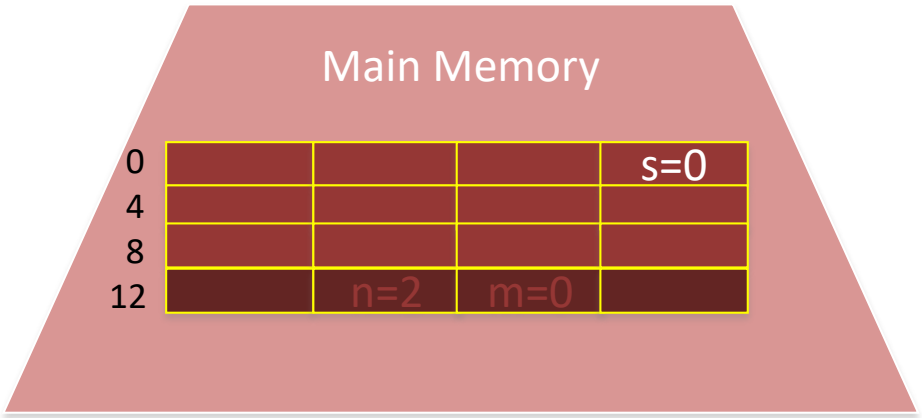
Tant que n > 0
    m ← m + n
    n ← n – 1

s ← m

Actions du proc :

…
Pas 13: lire @13
Pas 14: retourner 2
Pas 15: soustraire 1, n
Pas 16: écrire 1 @ 13
Pas 17: n > 0?
Pas 18: lire @13
Pas 19: retourner 1





Exemple: Somme des nombres jusqu'à n

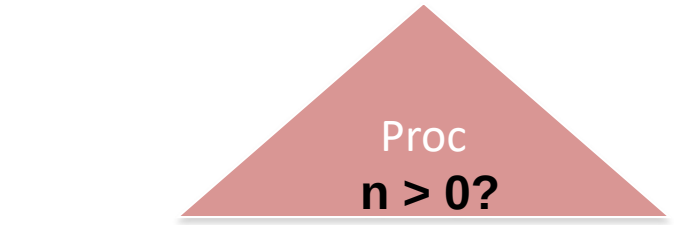
Algorithme:

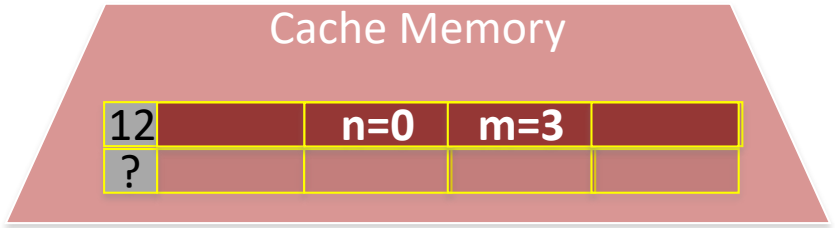
Tant que n > 0
    m ← m + n
    n ← n – 1

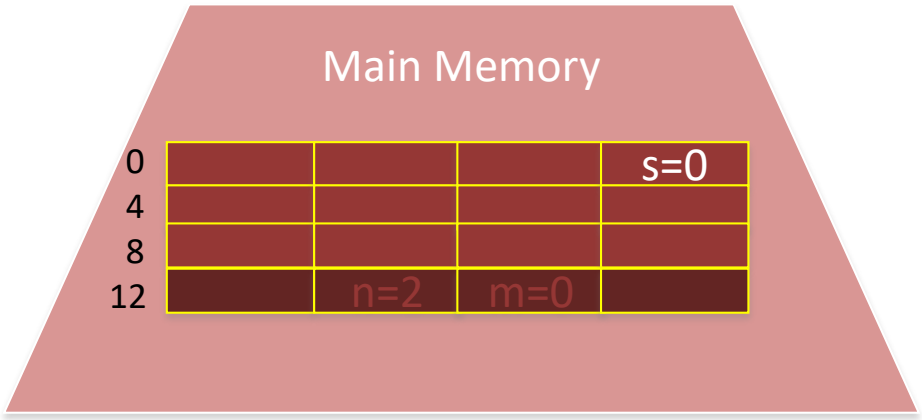
s ← m

Actions du proc :

…
Pas 13: lire @13
Pas 14: retourner 2
Pas 15: soustraire 1, n
Pas 16: écrire 1 @ 13
Pas 17: n > 0?
Pas 18: lire @13
Pas 19: retourner 1
…dix pas plus tard…
Pas 29: n > 0?







Exemple: Somme des nombres jusqu'à n

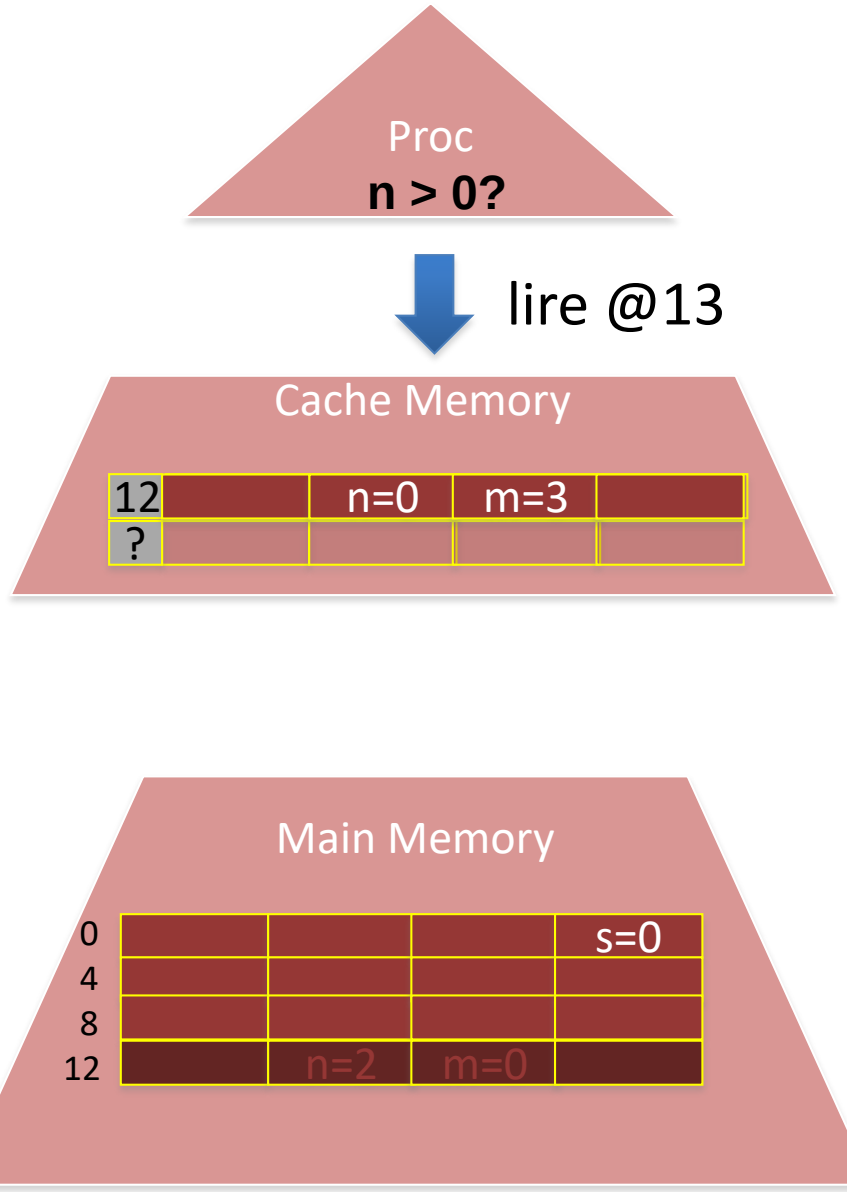
Algorithme:

Tant que n > 0
    m ← m + n
    n ← n – 1

s ← m

Actions du proc :

…
Pas 13: lire @13
Pas 14: retourner 2
Pas 15: soustraire 1, n
Pas 16: écrire 1 @ 13
Pas 17: n > 0?
Pas 18: lire @13
Pas 19: retourner 1
…dix pas plus tard…
Pas 29: n > 0?
Pas 30: lire @ 13



Exemple: Somme des nombres jusqu'à n

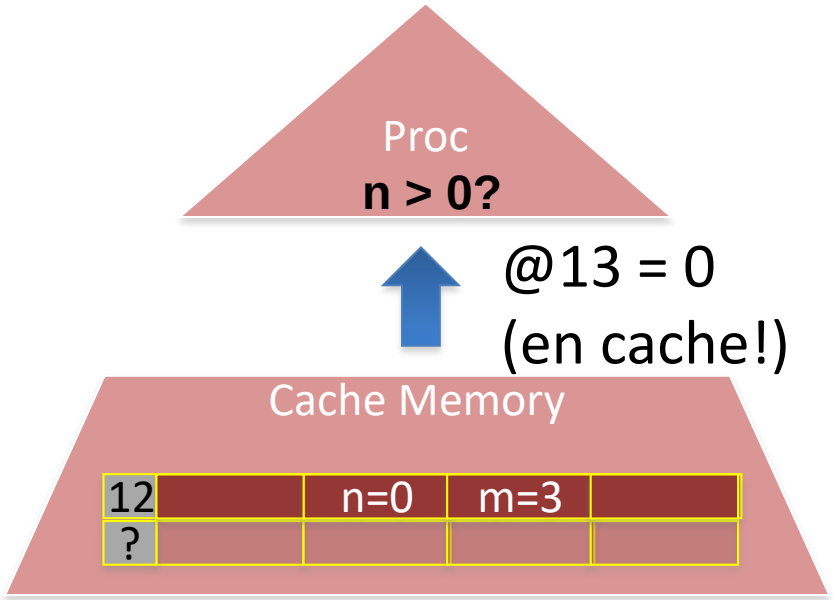
Algorithme:

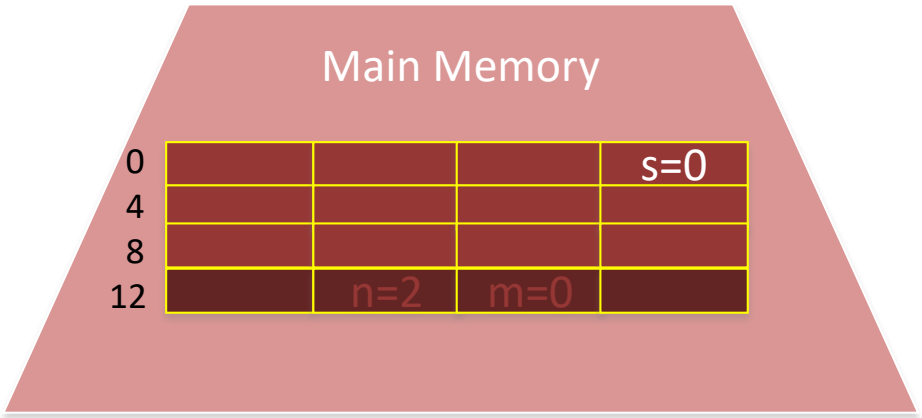
Tant que n > 0
    m ← m + n
    n ← n – 1

s ← m

Actions du proc :

…
Pas 13: lire @13
Pas 14: retourner 2
Pas 15: soustraire 1, n
Pas 16: écrire 1 @ 13
Pas 17: n > 0?
Pas 18: lire @13
Pas 19: retourner 1
…dix pas plus tard…
Pas 29: n > 0?
Pas 30: lire @ 13
Pas 31: retourner 0





Exemple: Somme des nombres jusqu'à n

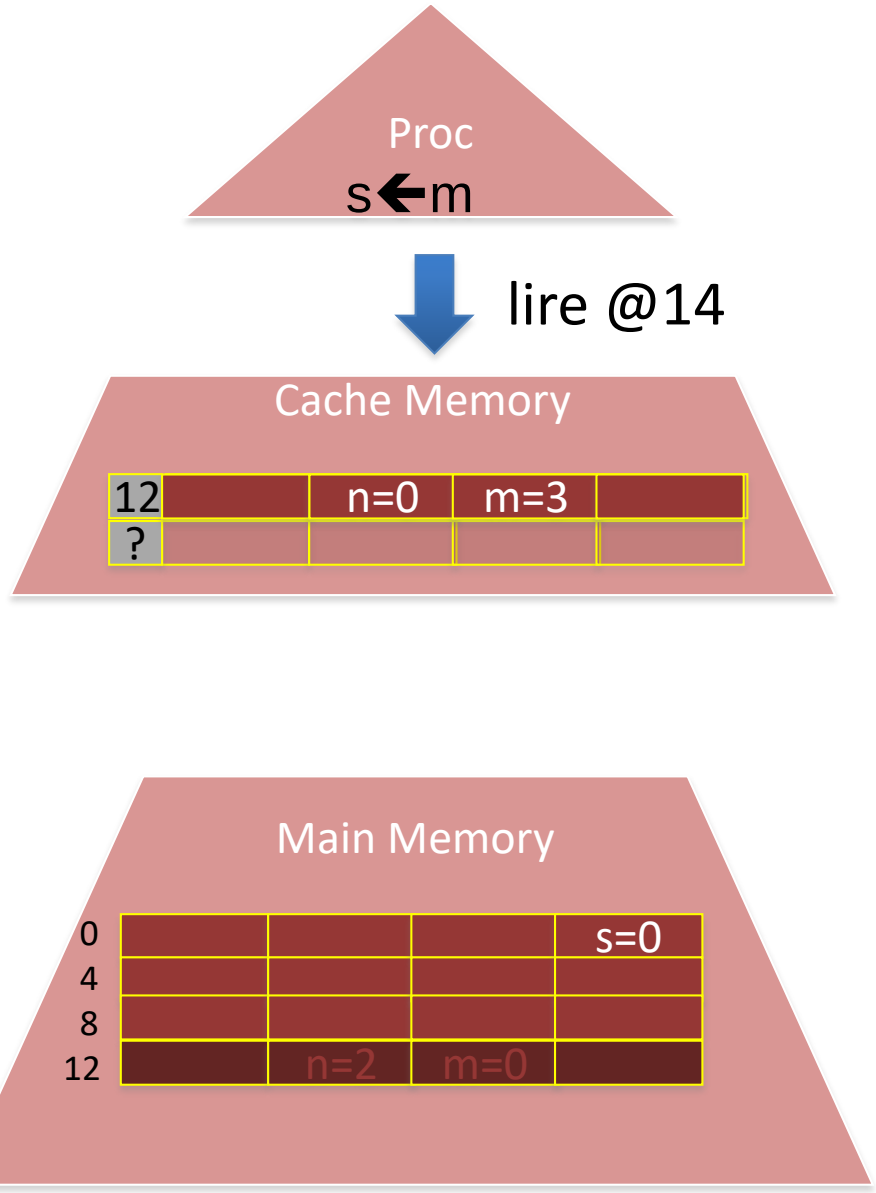
Algorithme:

Tant que n > 0
    m ← m + n
    n ← n – 1

s ← m

Actions du proc :

…
Pas 13: lire @13
Pas 14: retourner 2
Pas 15: soustraire 1, n
Pas 16: écrire 1 @ 13
Pas 17: n > 0?
Pas 18: lire @13
Pas 19: retourner 1
…dix pas plus tard…
Pas 29: n > 0?
Pas 30: lire @ 13
Pas 31: retourner 0
Pas 32: lire @ 14



Exemple: Somme des nombres jusqu'à n

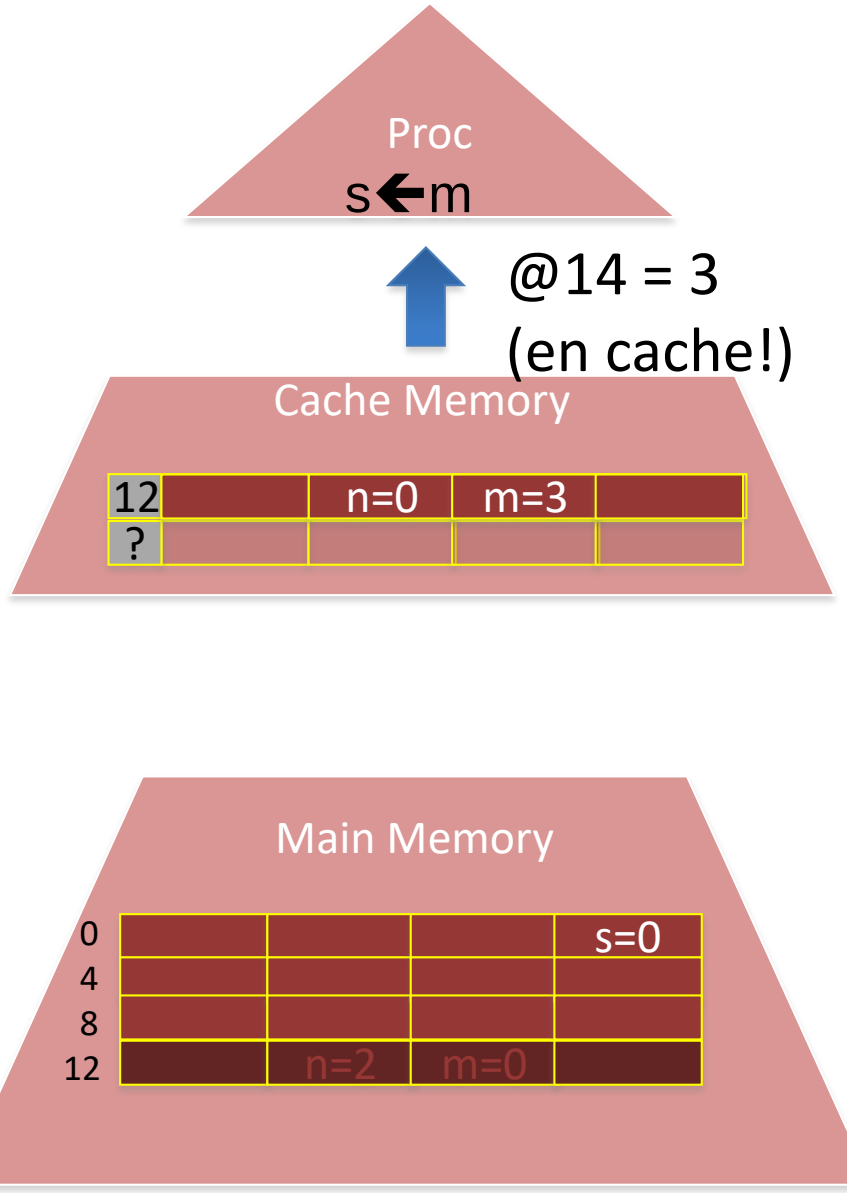
Algorithme:

Tant que n > 0
    m ← m + n
    n ← n – 1

s ← m

Actions du proc :

…
Pas 13: lire @13
Pas 14: retourner 2
Pas 15: soustraire 1, n
Pas 16: écrire 1 @ 13
Pas 17: n > 0?
Pas 18: lire @13
Pas 19: retourner 1
…dix pas plus tard…
Pas 29: n > 0?
Pas 30: lire @ 13
Pas 31: retourner 0
Pas 32: lire @ 14
Pas 33: retourner 3



Exemple: Somme des nombres jusqu'à n

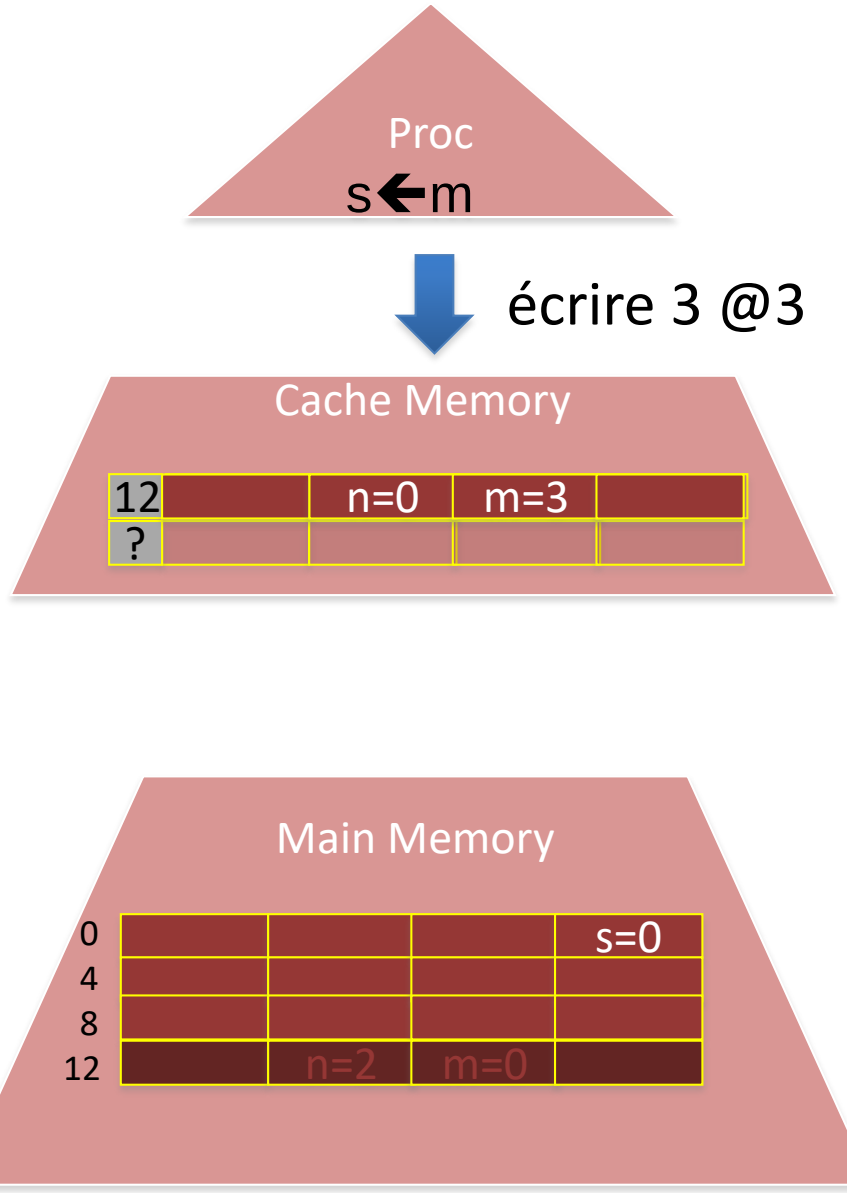
Algorithme:

Tant que n > 0
    m ← m + n
    n ← n – 1

s ← m

Actions du proc :

…
Pas 13: lire @13
Pas 14: retourner 2
Pas 15: soustraire 1, n
Pas 16: écrire 1 @ 13
Pas 17: n > 0?
Pas 18: lire @13
Pas 19: retourner 1
…dix pas plus tard…
Pas 29: n > 0?
Pas 30: lire @ 13
Pas 31: retourner 0
Pas 32: lire @ 14
Pas 33: retourner 3
Pas 34: écrire 3 @ 3



Exemple: Somme des nombres jusqu'à n

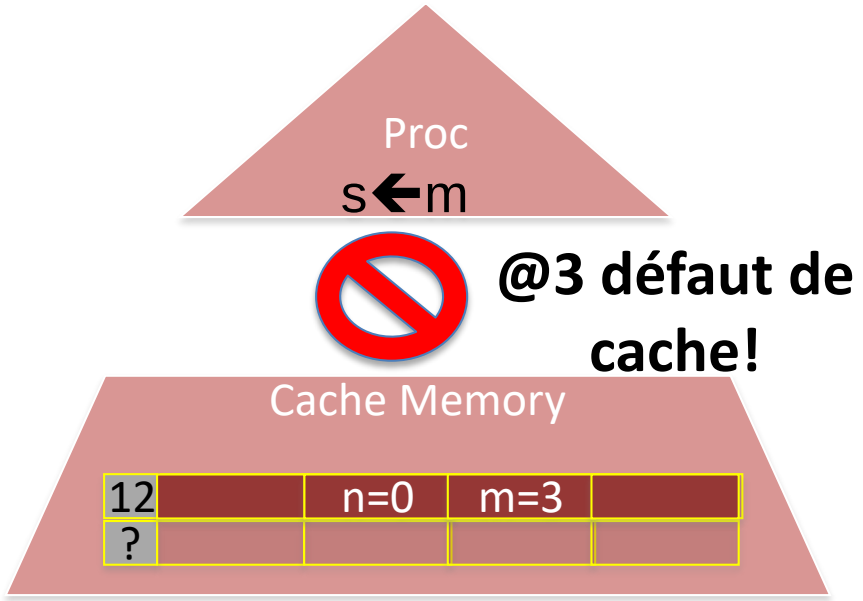
Algorithme:

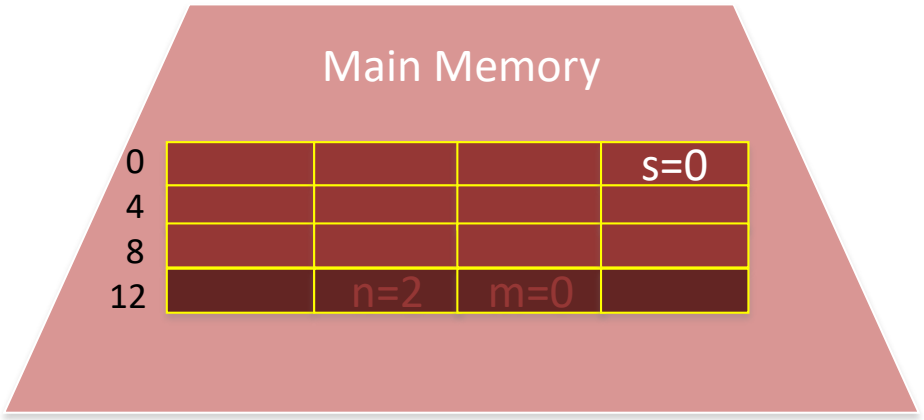
Tant que n > 0
    m ← m + n
    n ← n – 1

s ← m

Actions du proc :

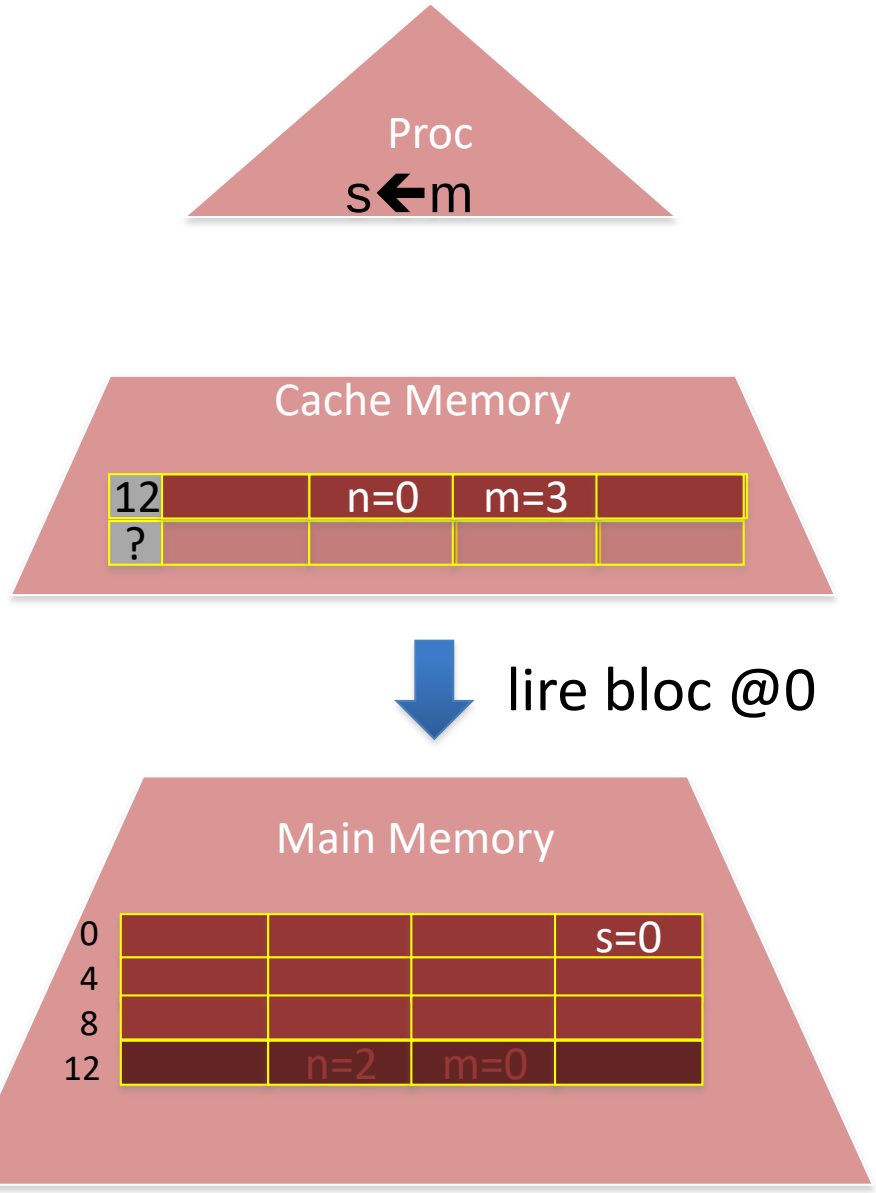
…
Pas 34: écrire 3 @3
Pas 35: @3 pas en cache
(défaut de cache!)





Exemple: Somme des nombres jusqu'à n

Algorithme:	Actions du proc :
Tant que n > 0	…
m ← m + n	Pas 34: écrire 3 @3
n ← n – 1	Pas 35: @3 pas en cache (défaut de cache!)
s ← m	Pas 36: lire bloc @ 0



Exemple: Somme des nombres jusqu'à n

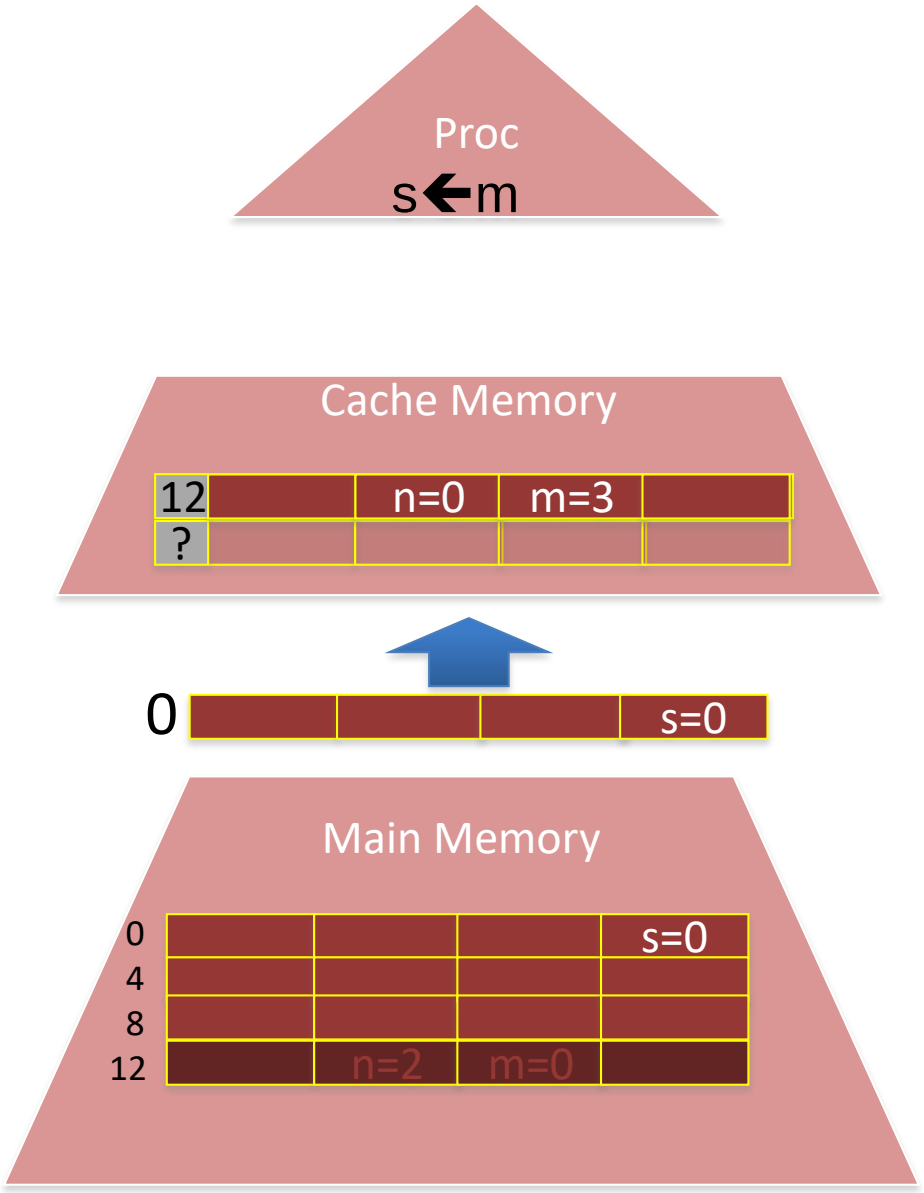
Algorithme:

Tant que n > 0
    m ← m + n
    n ← n – 1

s ← m

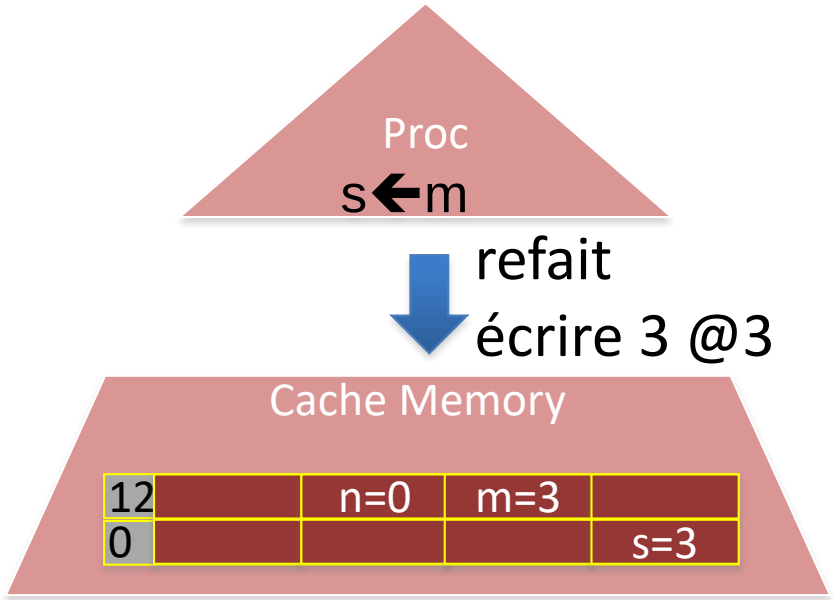
Actions du proc :

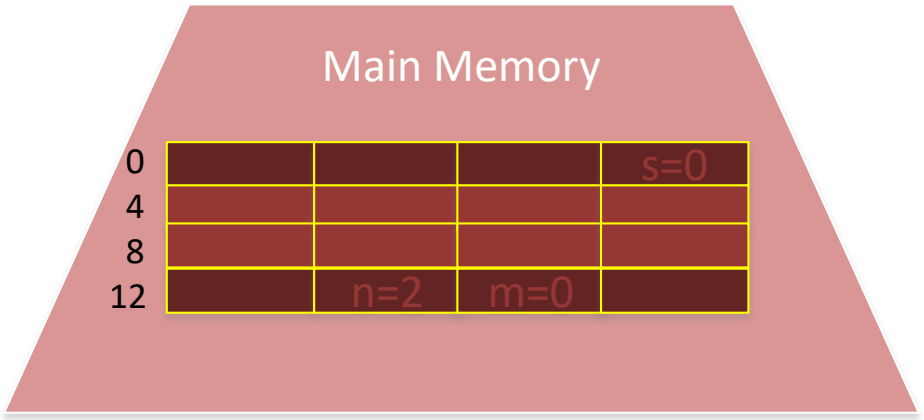
…
Pas 34: écrire 3 @3
Pas 35: @3 pas en cache
(défaut de cache!)
Pas 36: lire bloc @ 0
Pas 37: placer bloc @ 0



Exemple: Somme des nombres jusqu'à n

Algorithme:	Actions du proc :
Tant que n > 0	…
m ← m + n	Pas 34: écrire 3 @3
n ← n – 1	Pas 35: @3 pas en cache (défaut de cache!)
s ← m	Pas 36: lire bloc @ 0
	Pas 37: placer bloc @ 0
	Pas 38: refaire écrire 3 @ 3
	Terminé!





Quelle a été la vitesse de votre programme?

- 1ns pour presque tous les pas (processeur de 1 GHz)
  - Tous les accès en cache: 1ns
  - Défauts de cache: 100ns (pour obtenir le bloc de la mémoire)

- Tant que les données sont en cache, pas besoin d'accéder la mémoire.
  Le cache rend la mémoire à la fois grande et rapide!

Bon bilan accès mémoire: 6n accès cache+ 2 défauts de cache

- coût pour le premier passage:
  - 4 lectures dont 1 défaut & 2 écritures
- (n-1) passages suivants: aucun défaut de cache
- fin : 1 lecture et 1 écriture (défaut de cache)

Que ce passe-t-il si m & n sont dans des blocs différents?

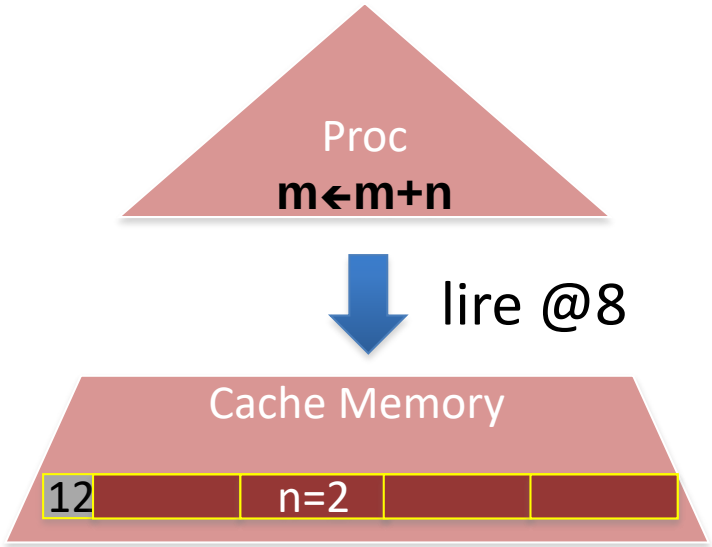
Algorithme:

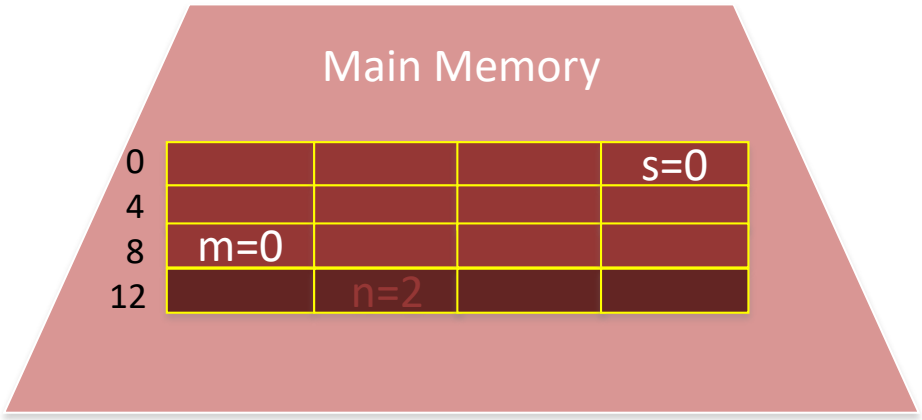
Tant que n > 0
    m ← m + n
    n ← n – 1

s ← m

Actions du proc:

Pas 1: n > 0?
Pas 2: lire @13
Pas 3: @13 pas en cache
(défaut de cache!)
Pas 4: lire bloc @12
Pas 5: placer bloc @12
Pas 6: retourner 2
Pas 7: **lire @8**





Que ce passe-t-il si m & n sont dans des blocs différents?

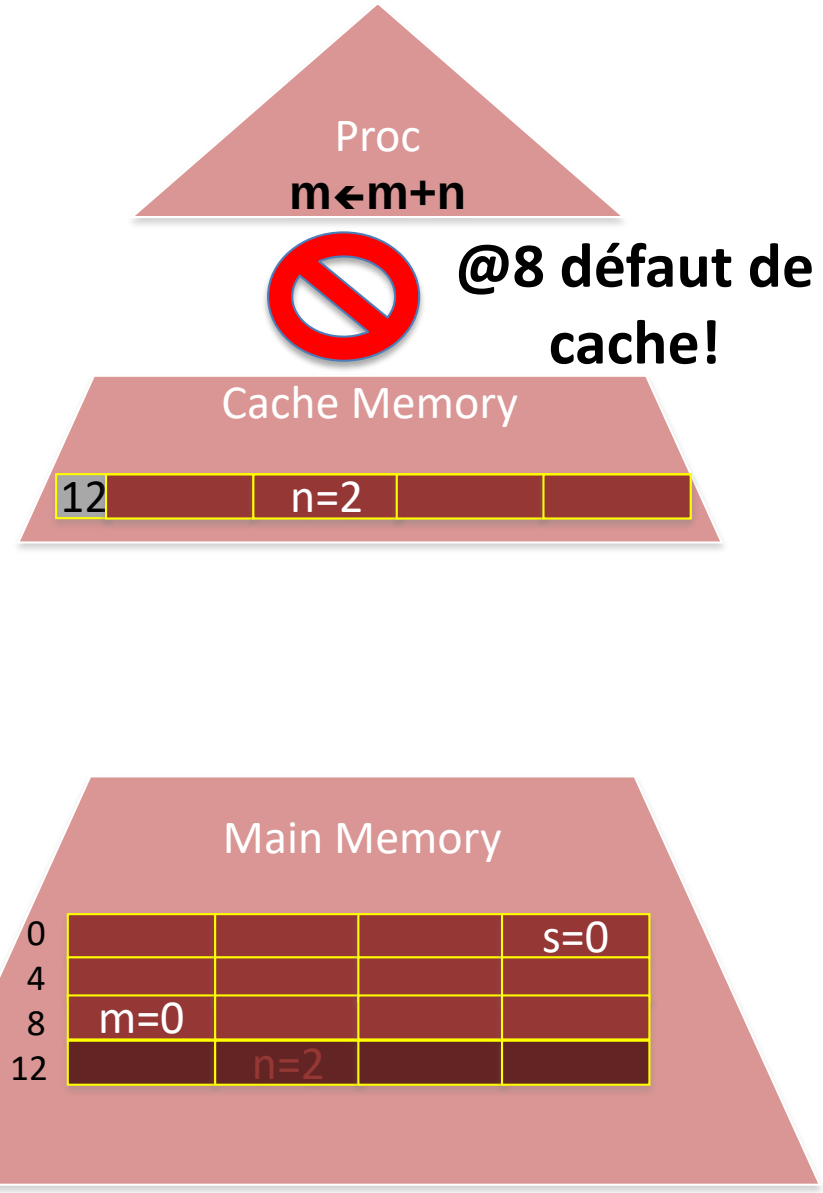
Algorithme:

Tant que n > 0
    m ← m + n
    n ← n – 1

s ← m

Actions du proc:

Pas 1: n > 0?
Pas 2: lire @13
Pas 3: @13 pas en cache
(défaut de cache!)
Pas 4: lire bloc @12
Pas 5: placer bloc @12
Pas 6: retourner 2
Pas 7: **lire @8**
Pas 8: @8 pas en cache
(défaut de cache!)



Que ce passe-t-il si m & n sont dans des blocs différents?

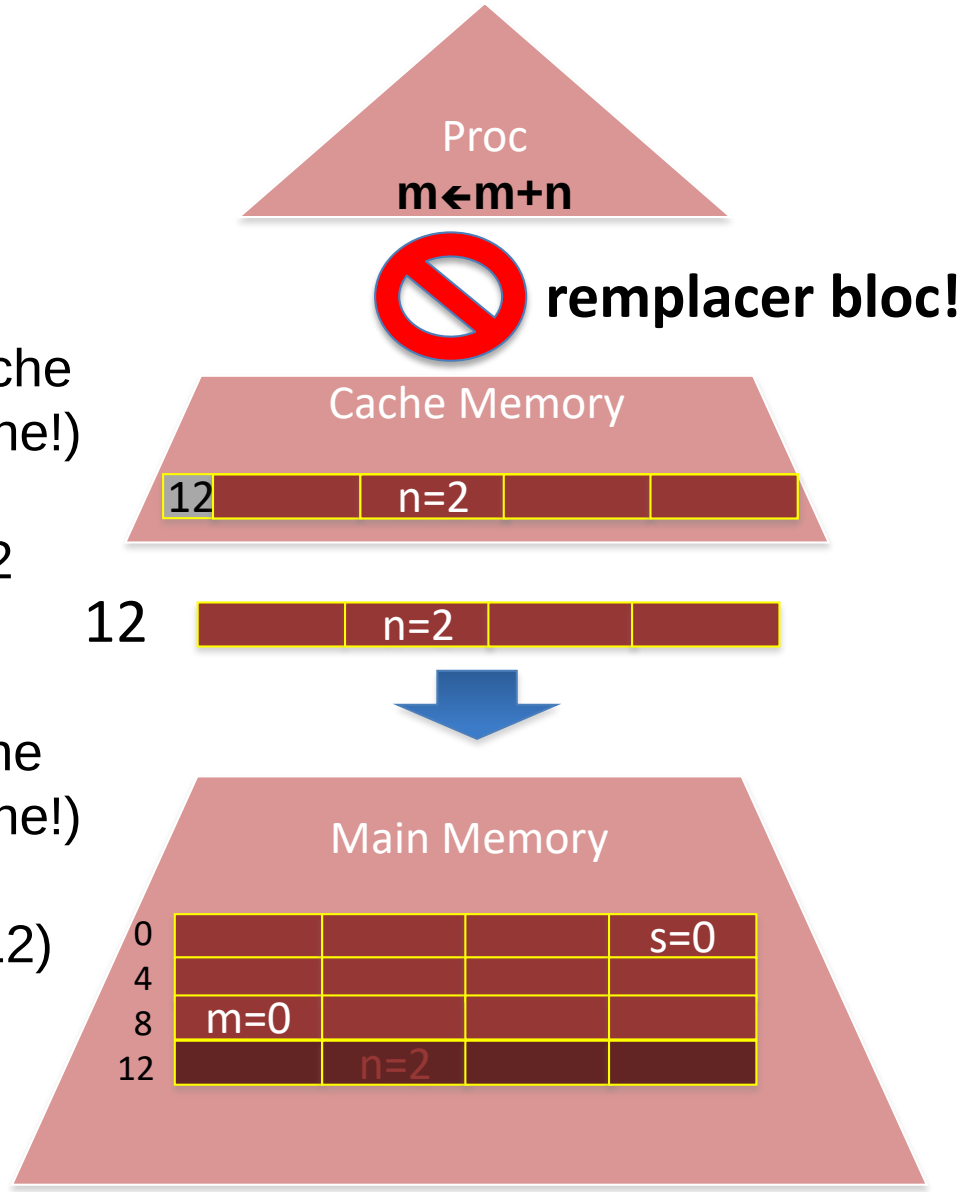
Algorithme:

Tant que n > 0
    m ← m + n
    n ← n – 1

s ← m

Actions du proc:

Pas 1: n > 0?
Pas 2: lire @13
Pas 3: @13 pas en cache
(défaut de cache!)
Pas 4: lire bloc @12
Pas 5: placer bloc @12
Pas 6: retourner 2
Pas 7: **lire @8**
Pas 8: @8 pas en cache
(défaut de cache!)
Pas 9: @12 en cache
(remplacer @12)



Que ce passe-t-il si m & n sont dans des blocs différents?

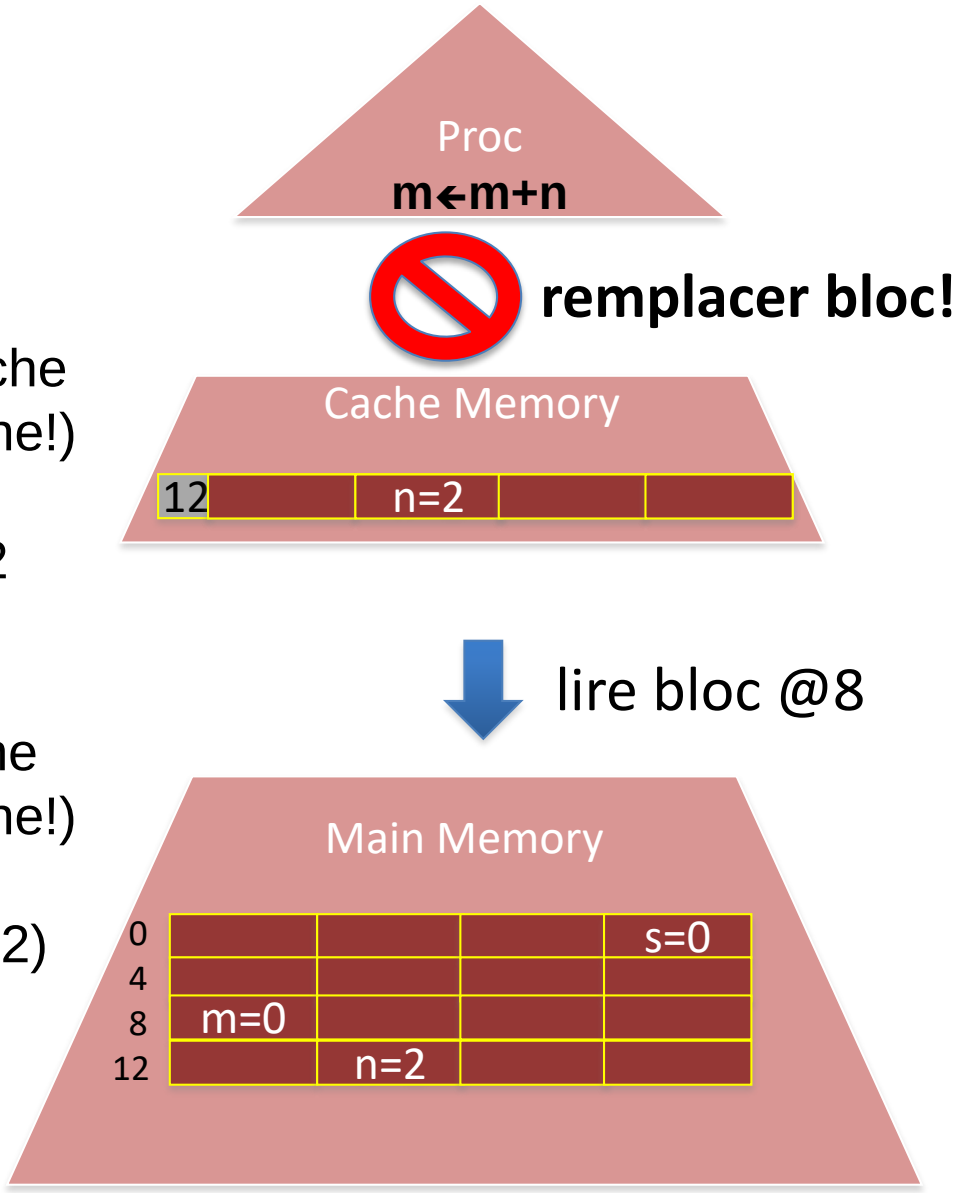
Algorithme:

Tant que n > 0
    m ← m + n
    n ← n – 1

s ← m

Actions du proc:

Pas 1: n > 0?
Pas 2: lire @13
Pas 3: @13 pas en cache
(défaut de cache!)
Pas 4: lire bloc @12
Pas 5: placer bloc @12
Pas 6: retourner 2
Pas 7: **lire @8**
Pas 8: @8 pas en cache
(défaut de cache!)
Pas 9: @12 en cache
(remplacer @12)
Pas 10: lire bloc @8



Que ce passe-t-il si m & n sont dans des blocs différents?

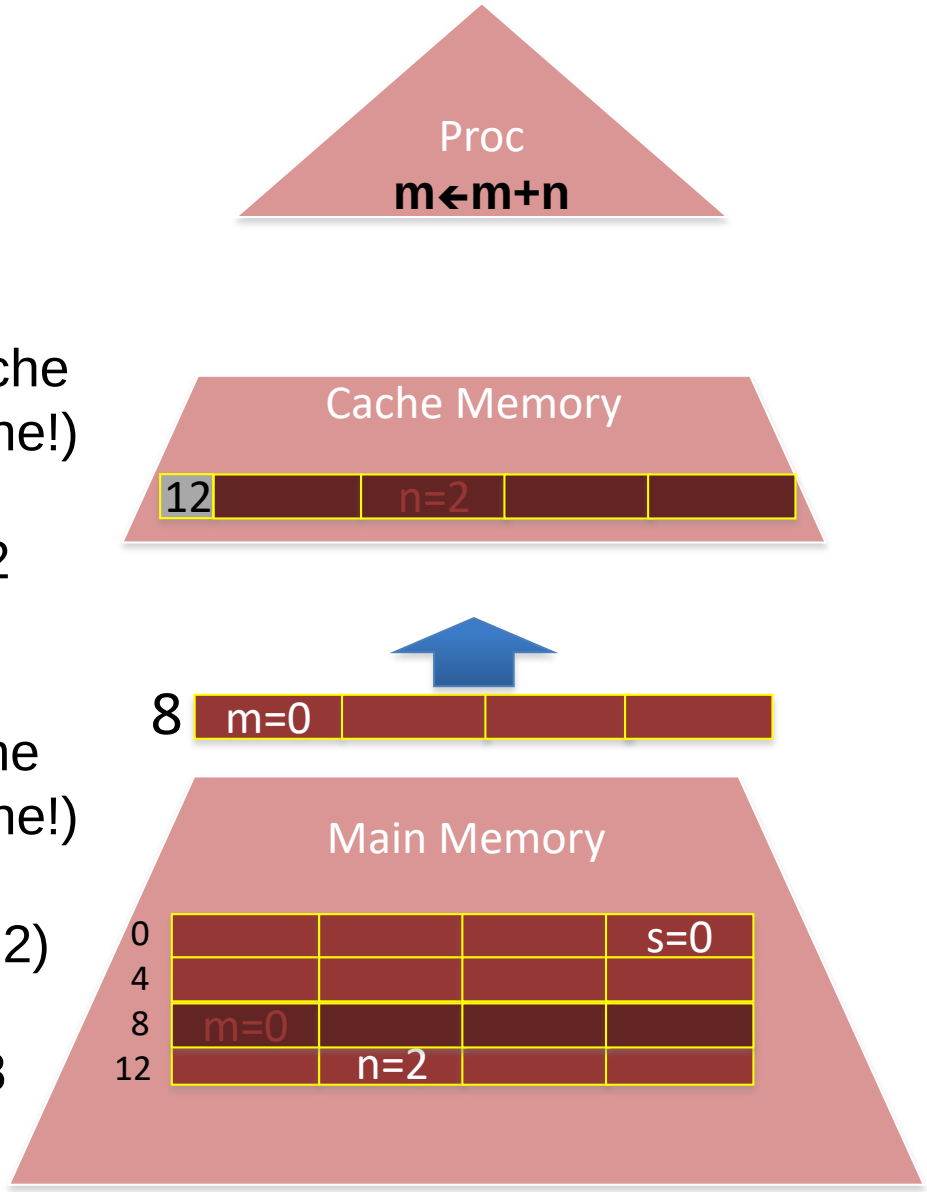
Algorithme:

Tant que n > 0
    m ← m + n
    n ← n – 1

s ← m

Actions du proc:

Pas 1: n > 0?
Pas 2: lire @13
Pas 3: @13 pas en cache
(défaut de cache!)
Pas 4: lire bloc @12
Pas 5: placer bloc @12
Pas 6: retourner 2
Pas 7: **lire @8**
Pas 8: @8 pas en cache
(défaut de cache!)
Pas 9: @12 en cache
(remplacer @12)
Pas 10: lire bloc @8
Pas 11: placer bloc @8



Que se passe-t-il si m & n sont dans des blocs différents?

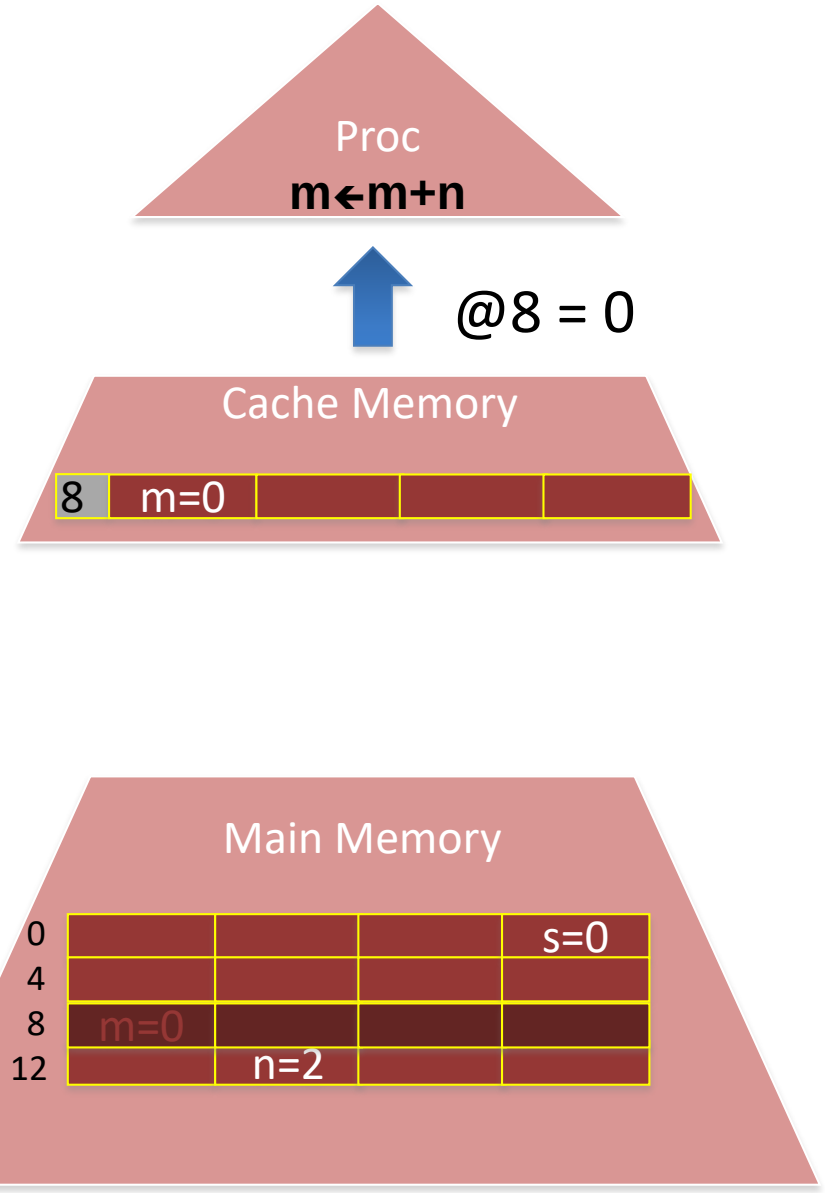
Algorithme:

Tant que n > 0
    m ← m + n
    n ← n – 1

s ← m

Actions du proc:

Pas 1: n > 0?
Pas 2: lire @13
Pas 3: @13 pas en cache
(défaut de cache!)
Pas 4: lire bloc @12
Pas 5: placer bloc @12
Pas 6: retourner 2
Pas 7: **lire @8**
Pas 8: @8 pas en cache
(défaut de cache!)
Pas 9: @12 en cache
(remplacer @12)
Pas 10: lire bloc @8
Pas 11: placer bloc @8
Pas 12: retourner 0



Exercice

- Q: Si m & n sont dans des blocs différents, un cache d'un seul bloc conduit à un défaut de cache à chaque référence, en cas d'accès alternés à ces deux variables. Quel est le temps nécessaire pour exécuter le programme?

Suite du cours:

- **Principe du cache**: une mémoire ***on-chip*** = d'accès rapide pour le processeur qui lui donne d'une mémoire grande et rapide
- **Fonctionnement du cache** avec le processeur et la mémoire centrale
- **Exemple:** additionner les nombres jusqu'à n
- Le compromise entre ***localité spatiale*** et ***localité temporelle***
- Impact de l'organisation de la mémoire sur les performances d'un algorithme: exemple d'un parcours de matrice

D'où provient la réussite dans l'accès au cache?

Lié à la localité des accès

Défaut de cache une fois sur le bloc @12

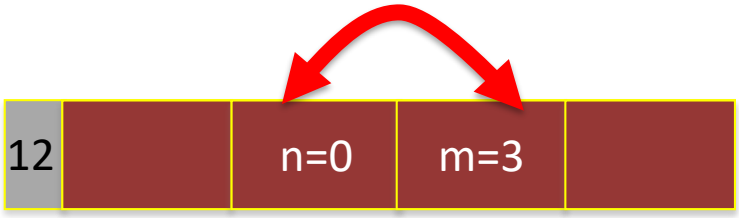
Mais en cache une douzaine de fois

Pourquoi?

Variables liées placées dans le même bloc (n & m)

Appelé ***"localité spatiale"***

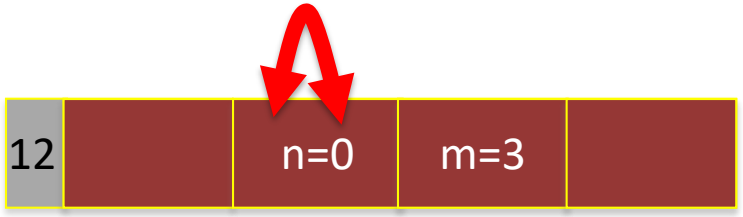
Si un élément est accédé en mémoire, son voisin a aussi de fortes chances d'être accédé (ex: tableau).



Mêmes variables accédées plusieurs fois de suite (ex: n)

Appelé ***"localité temporelle"***

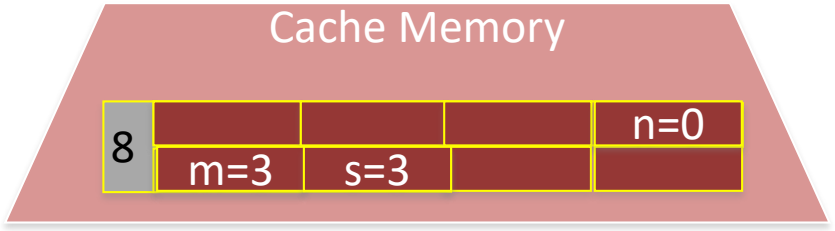
Un élément accédé récemment a de fortes chances d'être accédé dans le futur.



Localité & taille des blocs

- Avantage de blocs plus grands ➔ plus de localité spatiale
  - supposons n, m et s à des adresses consécutives

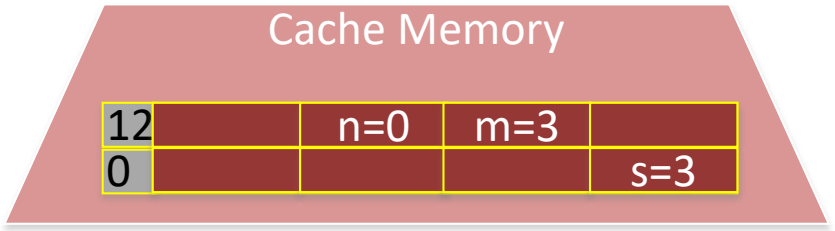
1 bloc de 8 mots

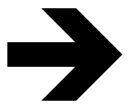


- Inconvénient de blocs plus grands:

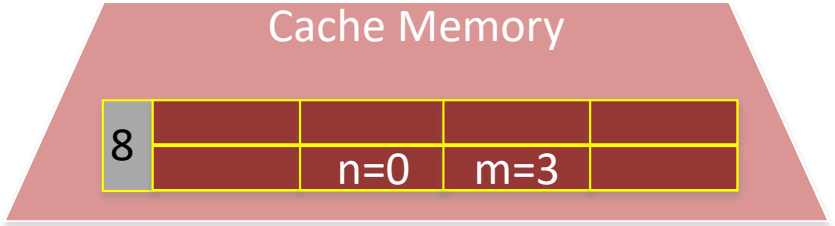
➔ moins de localité temporelle

2 blocs de 4 mots

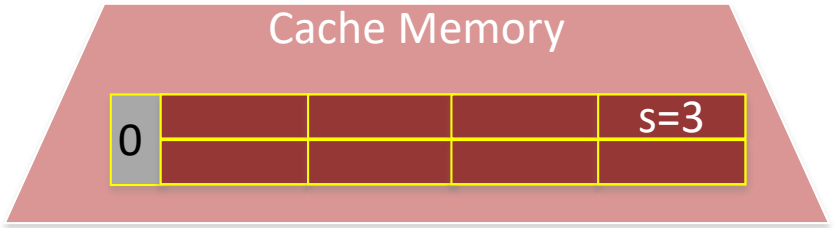




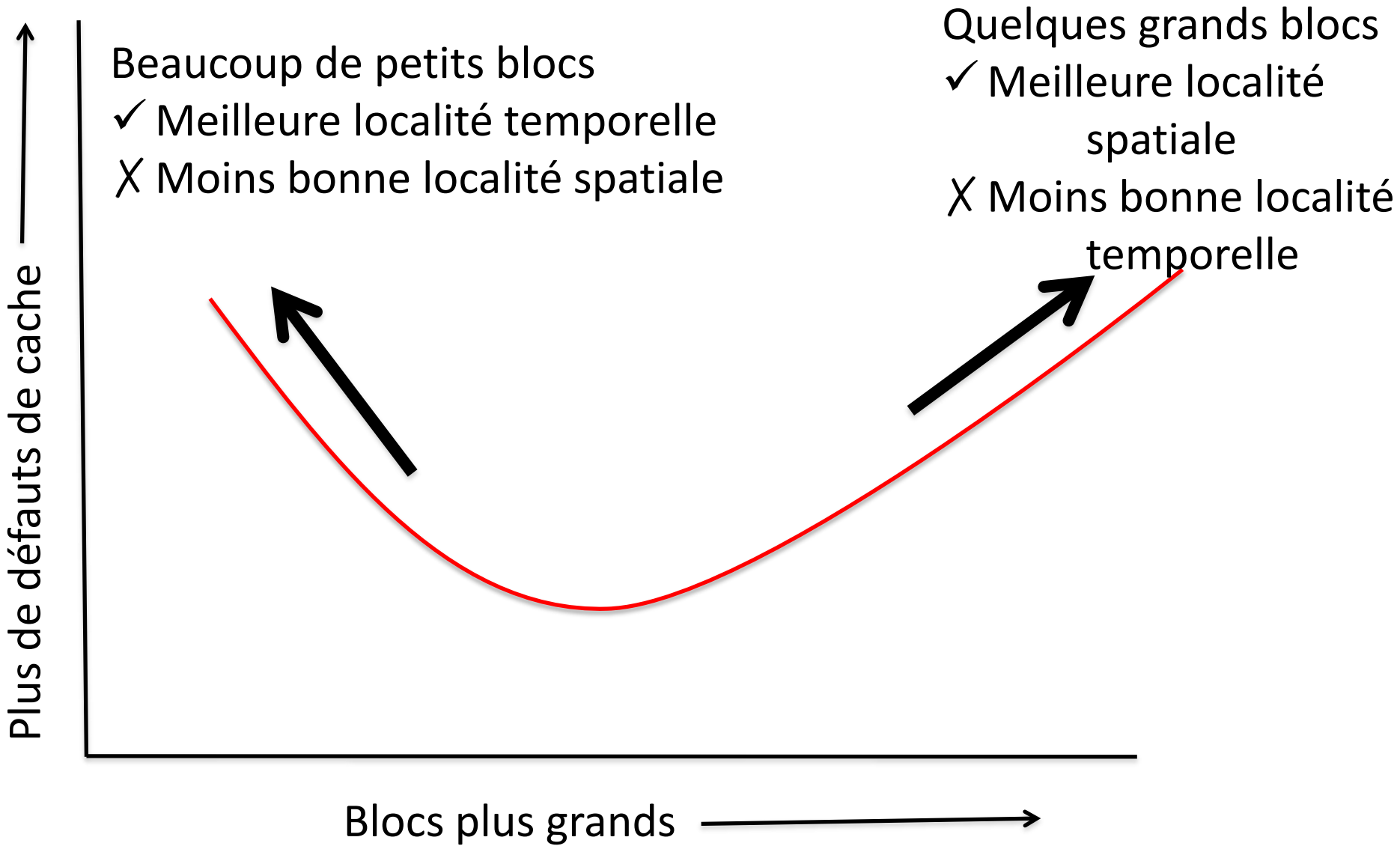
1 bloc de 8 mots



ou



Localité spatiale vs. temporelle & Taille des blocs



Suite du cours:

- **Principe du cache**: une mémoire ***on-chip*** = d'accès rapide pour le processeur qui lui donne d'une mémoire grande et rapide
- **Fonctionnement du cache** avec le processeur et la mémoire centrale
- **Exemple:** additionner les nombres jusqu'à n
- Le compromis entre ***localité spatiale*** et ***localité temporelle***
- Impact de l'organisation de la mémoire sur les performances d'un algorithme: exemple d'un parcours de matrice

Exemple: Addition des éléments d'une matrice

Addition des éléments de M

$$s \leftarrow \sum_{i=0}^{3} \sum_{j=0}^{3} M(i,j)$$

La somme ne dépend pas de la façon dont les éléments sont ajoutés

$$s \leftarrow \sum_{j=0}^{3} \sum_{i=0}^{3} M(i,j)$$

Mais la façon de faire la somme peut influencer le temps de calcul

Matrix M(i,j)

M(0,0)	M(0,1)	M(0,2)	M(0,3)
M(1,0)	M(1,1)	M(1,2)	M(1,3)
M(2,0)	M(2,1)	M(2,2)	M(2,3)
M(3,0)	M(3,1)	M(3,2)	M(3,3)

Addition des éléments de M: Disposition en mémoire

Pourquoi une différence entre ces deux options ?

- Les données sont stockées en mémoire par lignes
- M a 2 indices: lignes & colonnes
- La mémoire est un tableau de mots à 1 dimension

Supposons que M commence au mot d'adresse 0

Matrice M(i,j)

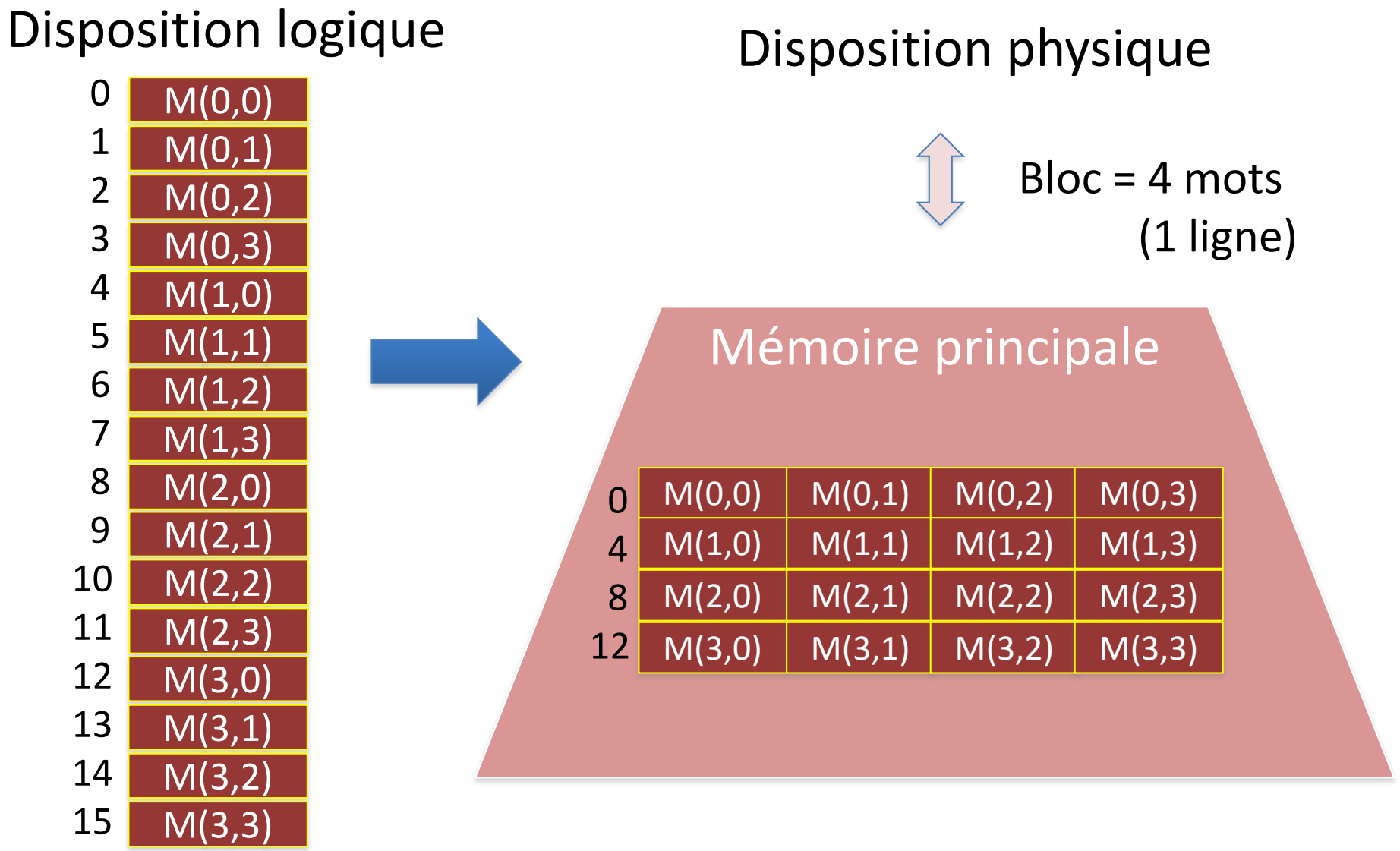
M(0,0)	M(0,1)	M(0,2)	M(0,3)
M(1,0)	M(1,1)	M(1,2)	M(1,3)
M(2,0)	M(2,1)	M(2,2)	M(2,3)
M(3,0)	M(3,1)	M(3,2)	M(3,3)

M est stockée par ligne

Disposition logique

0	M(0,0)
1	M(0,1)
2	M(0,2)
3	M(0,3)
4	M(1,0)
5	M(1,1)
6	M(1,2)
7	M(1,3)
8	M(2,0)
9	M(2,1)
10	M(2,2)
11	M(2,3)
12	M(3,0)
13	M(3,1)
14	M(3,2)
15	M(3,3)

Disposition logique vs. physique (par bloc)



Notation mathématique, graphique & algorithmique

Addition par lignes

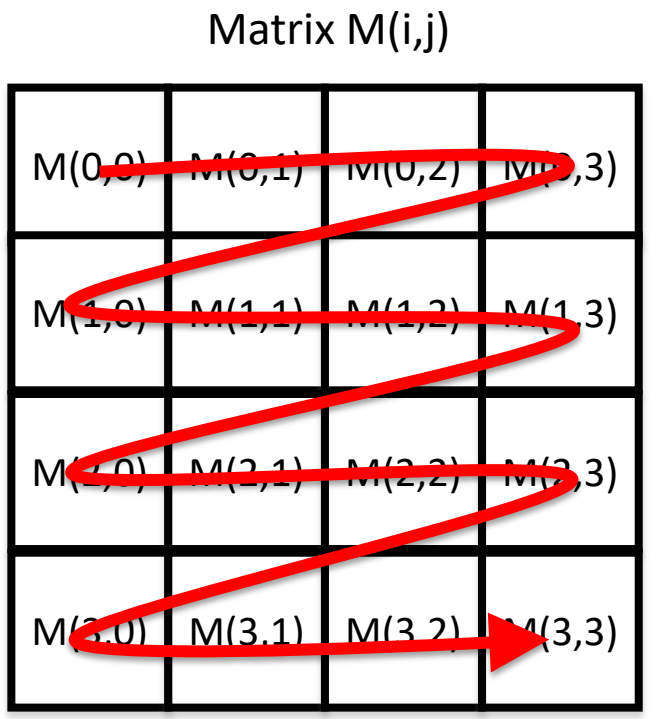
Notation mathématique

$$s \leftarrow \sum_{i=0}^{3}\sum_{j=0}^{3} M(i,j)$$

Notation algorithmique

Tant que $i \leq 3$
  j ← 0 (une fois par ligne)
  Tant que $j \leq 3$
    s ← s + M(i,j)
    j ← j + 1
  i ← i + 1

Notation graphique

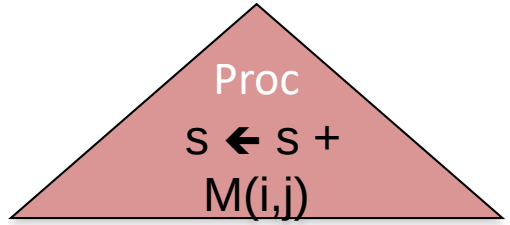


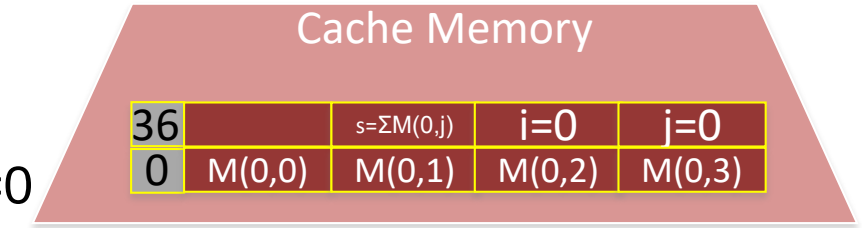
Exemple: Addition par lignes

Algorithme:

```
Tant que i ≤ 3
    j ← 0
    Tant que j ≤ 3
       s ← s + M(i,j)
       j ← j + 1
    i ← i + 1
```

- un défaut de cache pour i
- un défaut de cache pour M(0,0)

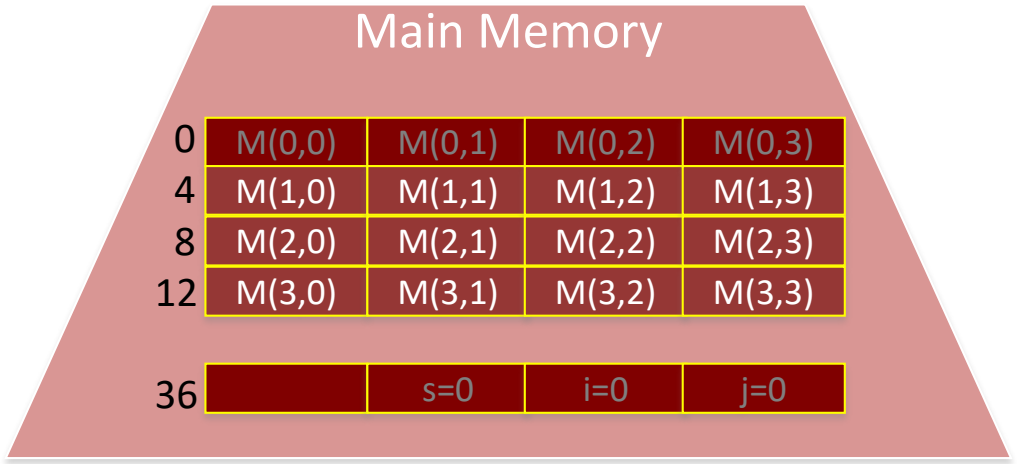




- Donc tous les accès ultérieurs pour la ligne i=0 seront en cache

- donc,

$$s \leftarrow s + \sum_{j=0}^{3} M(0,j)$$



Exemple: Addition par lignes

- Pour chaque ligne ultérieure (valeur de i)
  - seulement un défaut de cache pour amener la ligne en cache

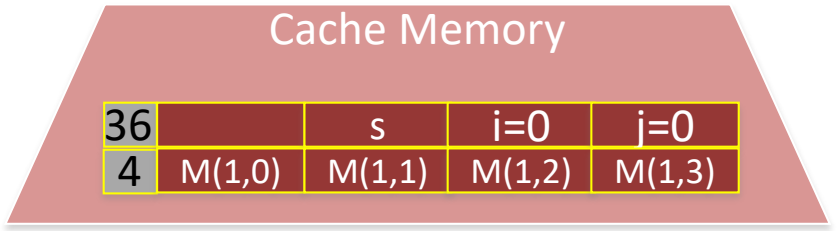
$$s \leftarrow s + \sum_{j=0}^{3} M(1,j)$$

$$s \leftarrow s + \sum_{j=0}^{3} M(2,j)$$

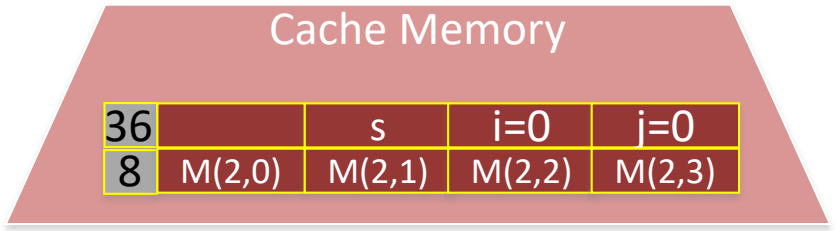
$$s \leftarrow s + \sum_{j=0}^{3} M(3,j)$$

Contenu du cache par ligne

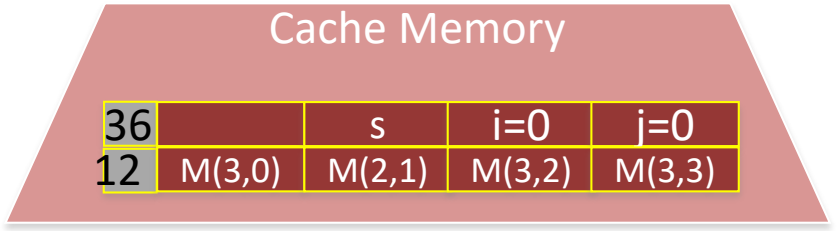
i=1



i=2



i=3



Exemple: Addition par colonnes

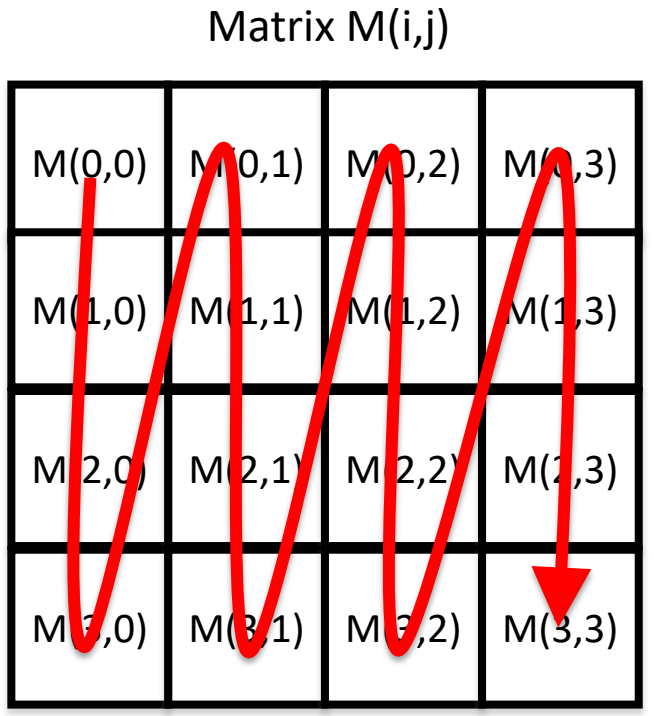
Notation mathématique

$$s \leftarrow \sum_{j=0}^{3} \sum_{i=0}^{3} M(i,j)$$

Notation algorithmique

Tant que $j \leq 3$
  $i \leftarrow 0$ (une fois par colonne)
  Tant que $i \leq 3$
    $s \leftarrow s + M(i,j)$
    $i \leftarrow i + 1$
  $j \leftarrow j + 1$

Notation graphique



Exemple: Addition par colonnes

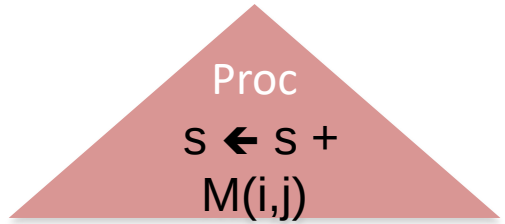
Algorithme:

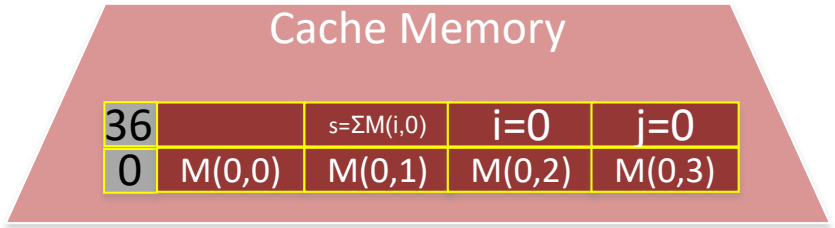
```
Tant que j ≤ 3
    i ← 0
    Tant que i ≤ 3
        s ← s + M(i,j)
        i ← i + 1
    j ← j + 1
```

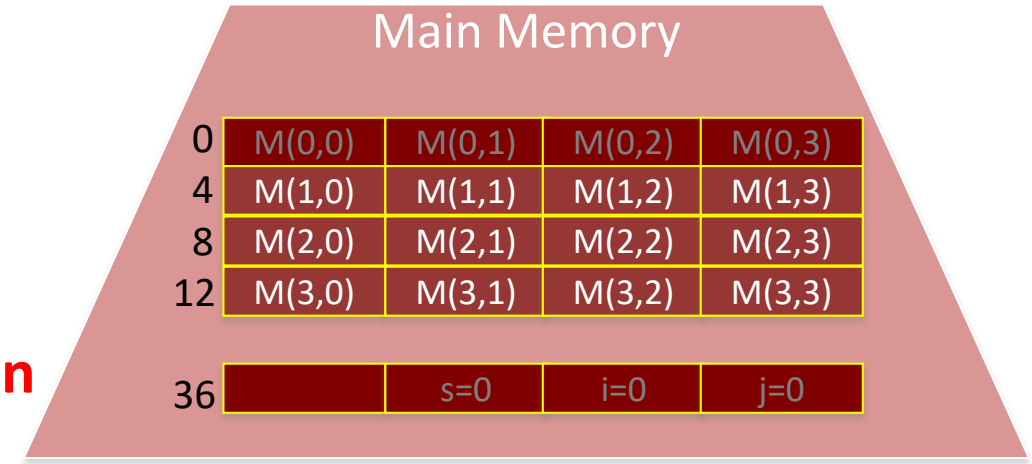
- Maintenant, addition de M(0,0) avec M(1,0), M(2,0) and M(3,0)
- Uniquement M(0,0) en cache
- Donc, accès à M(1,0), M(2,0) and M(3,0), pour colonne j=0 provoquent un défaut de cache!
- Comséquence:

$$s \leftarrow s + \sum_{i=0}^{3} M(i,0)$$

provoque quatre défauts de cache, un pour chaque ligne!







Exercice

- Q: Essayons de comparer les differences de performance. Supposons que chaque operation prend un tic d'horloge du processeur, et qu'un défaut de cache prend 100 tics d'horloge. Calculer le nombre de tics d'horloge.

Addition des éléments de la ligne 0

$$s \leftarrow s + \sum_{j=0}^{3} M(0,j)$$

Nombre de tics d'horloge =

Addition des éléments de la colonne 0

$$s \leftarrow s + \sum_{i=0}^{3} M(i,0)$$

Nombre de tics d'horloge =

Résumé

- Hiérarchies de mémoires
  - Différence de technologie: vitesse vs. capacité
- Deux types de stockage
  - Mémoire pour le calcul
  - Mémoire de stockage pour l'archivage de données
- Cache & localité
  - Permet de rendre la mémoire grande & rapide
- Les algorithmes peuvent aussi conduire à des performances différentes
  - Par ex., 4x entre accès d'une matrice par lignes & colonnes