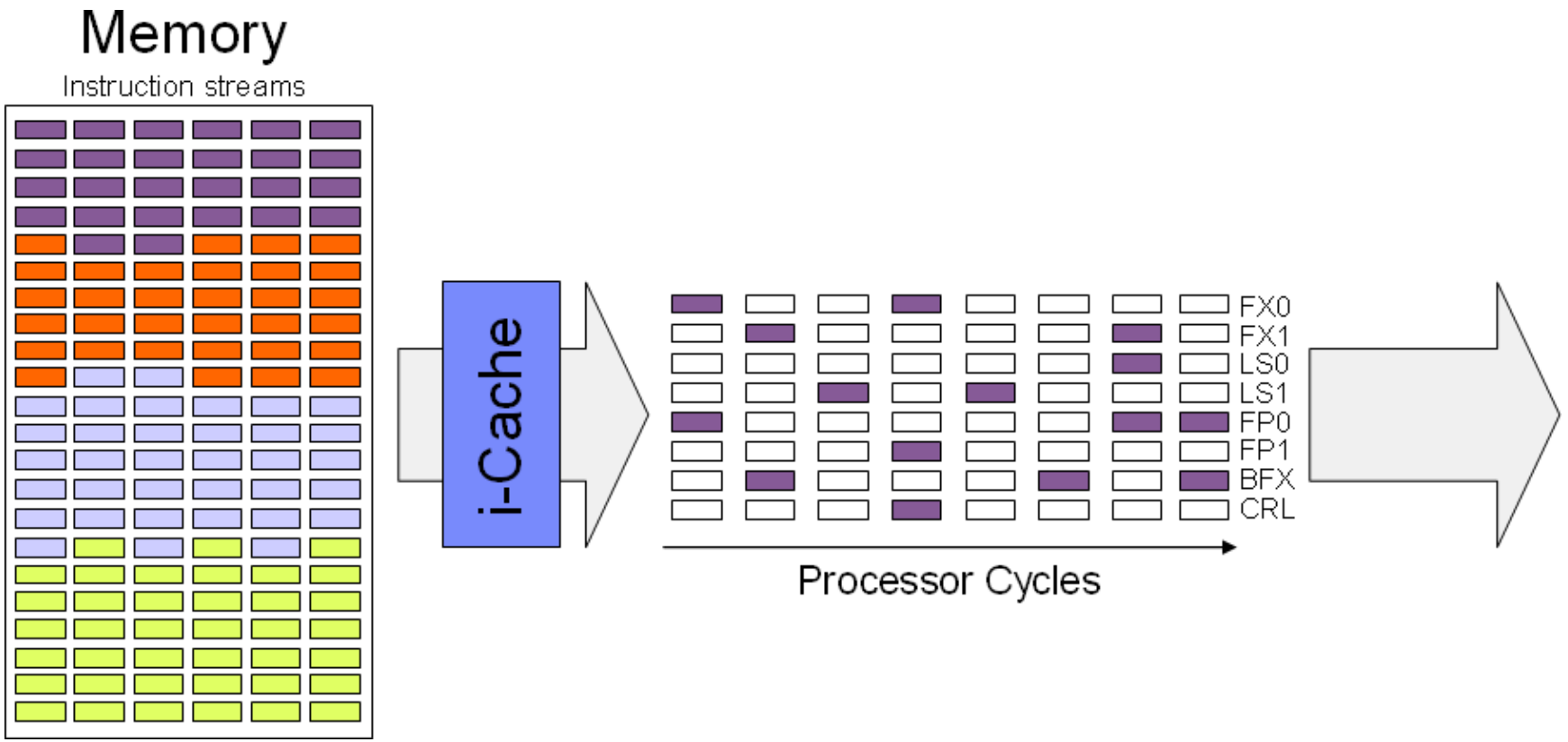
Thread Level Parallelism

- OoO & Cie ne peuvent rien contre les défauts de cache

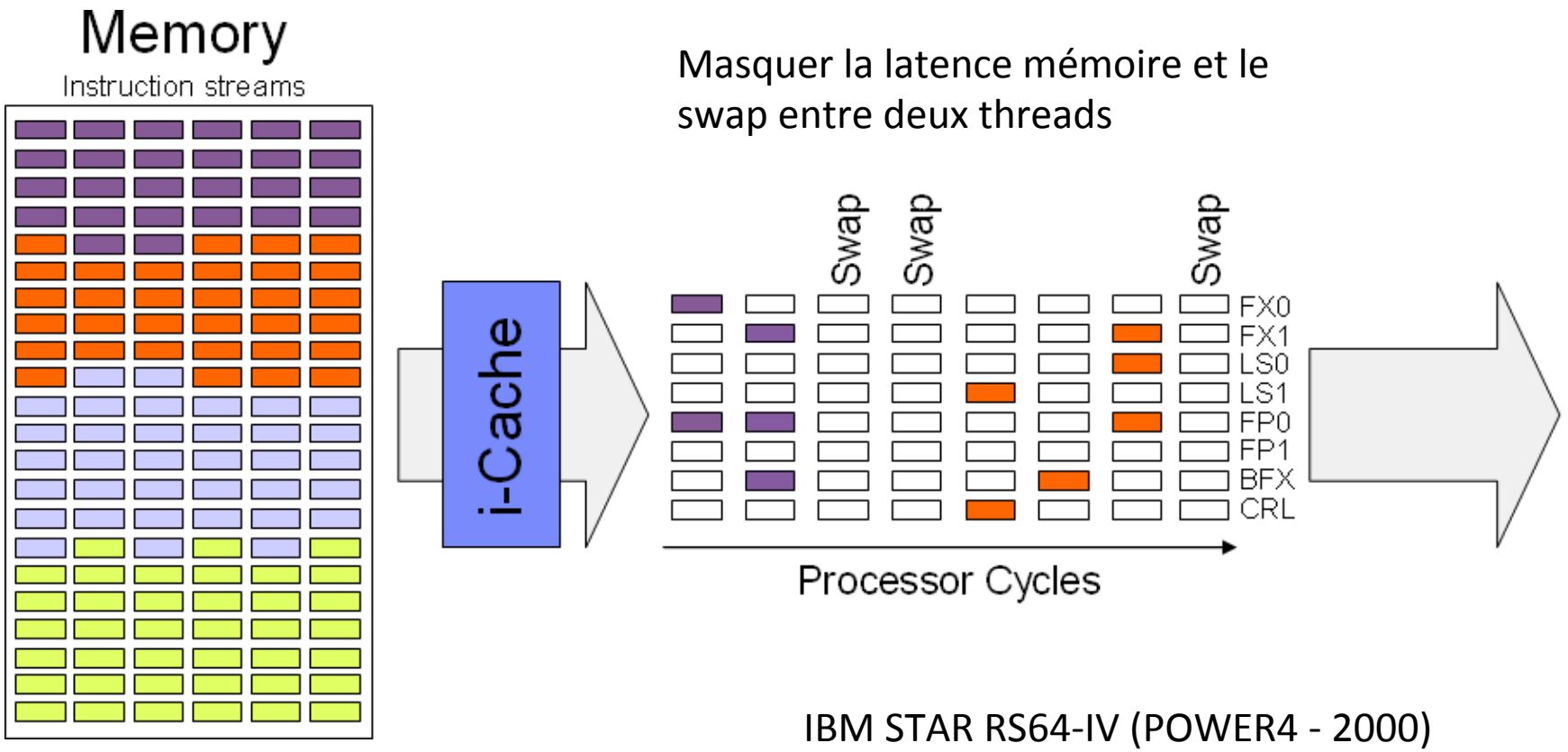
niveau	temps	cycles 2,5GHz	Exemple (cycles)
L1	<1ns	2-4	Nehalem 4
L2	2-5ns	8-20	Nehalem 10, Opteron 7, PIV 22, Core-2 14
L3	10ns	40	Nehalem 40
Mem	60ns	240	

- Prefetching
- Multithreading au niveau du pipeline

Multithreading



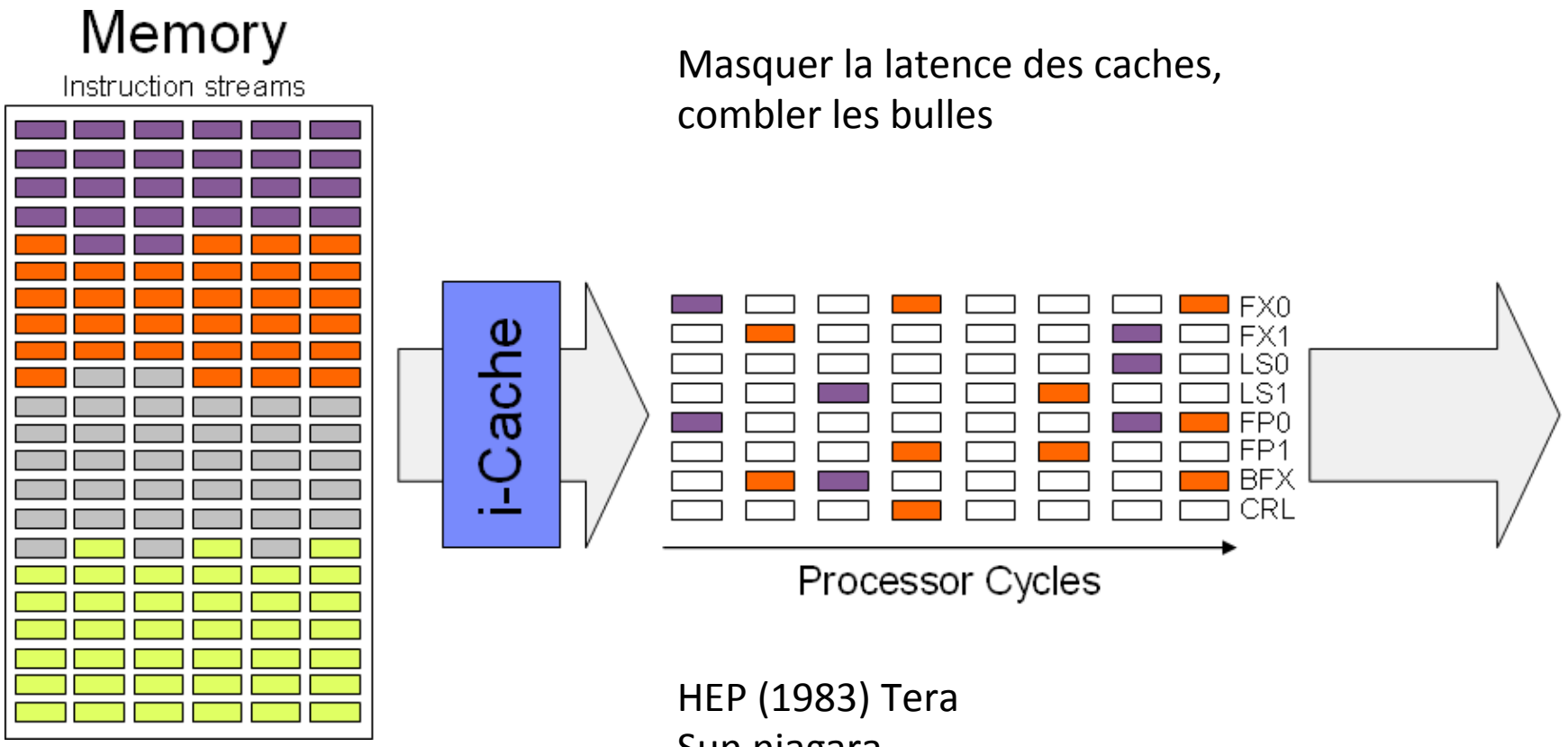
Multithreading
Coarse Grained MT



Masquer la latence mémoire et le swap entre deux threads

IBM STAR RS64-IV (POWER4 - 2000)

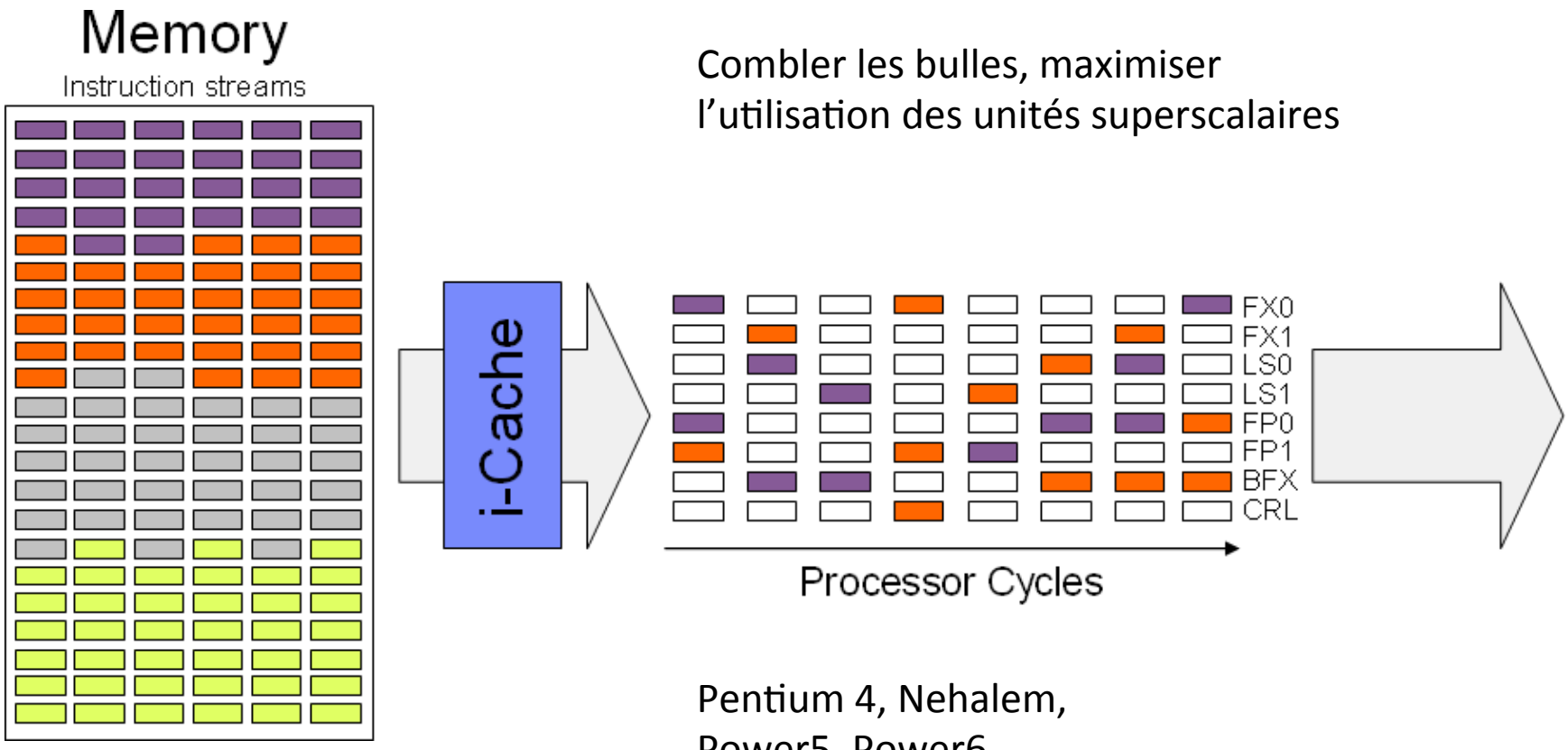
Multithreading
Fine Grained MT



Masquer la latence des caches,
combler les bulles

HEP (1983) Tera
Sun niagara

Multithreading
Simultaneous MT



Combler les bulles, maximiser
l'utilisation des unités superscalaires

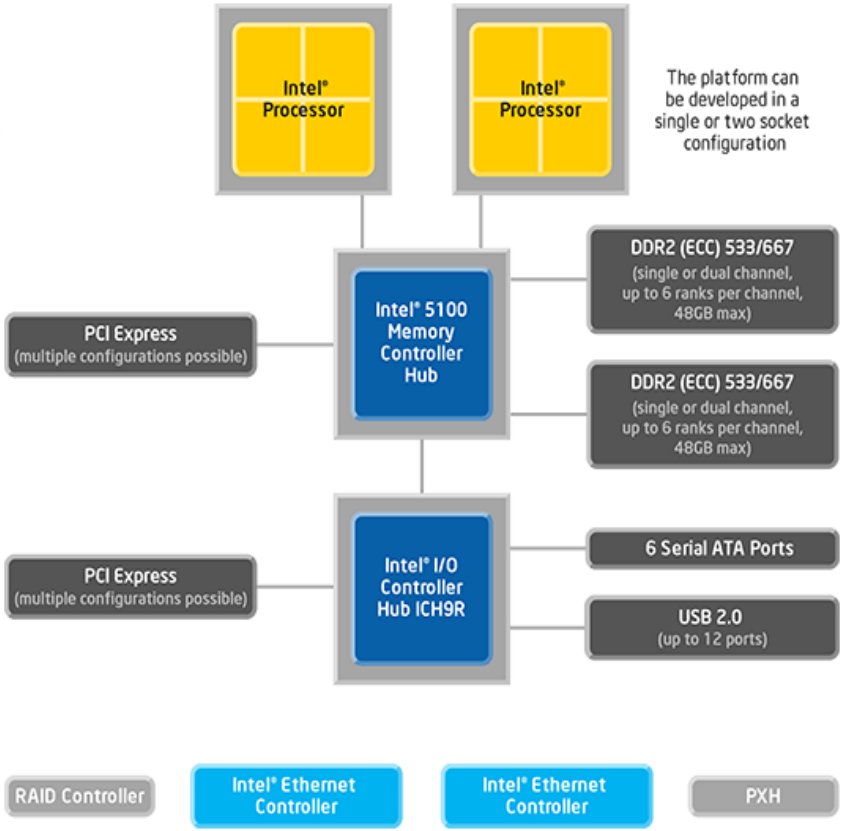
Pentium 4, Nehalem,
Power5, Power6

Multithreading
CMT – FMT - SMP

- Avantages
  - Meilleure utilisation du pipeline
  - Réactivité (lock)
  - Partage de données entre 2 threads
  - Régulation de charge
- Inconvénients
    - Partage de cache → perte de performance
    - Gestion des priorités
    - Difficultés de mise en œuvre des techniques de scrutation
- Solution
  - Désactivation
  - Appariement de threads compatibles

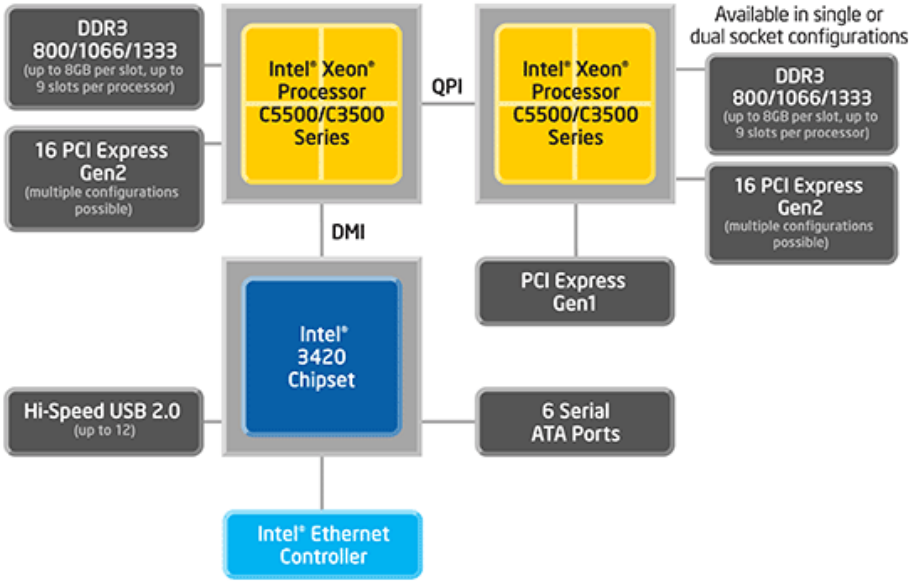
Machines multiprocesseurs à mémoire partagée

- SMP
  - Mémoire centralisée



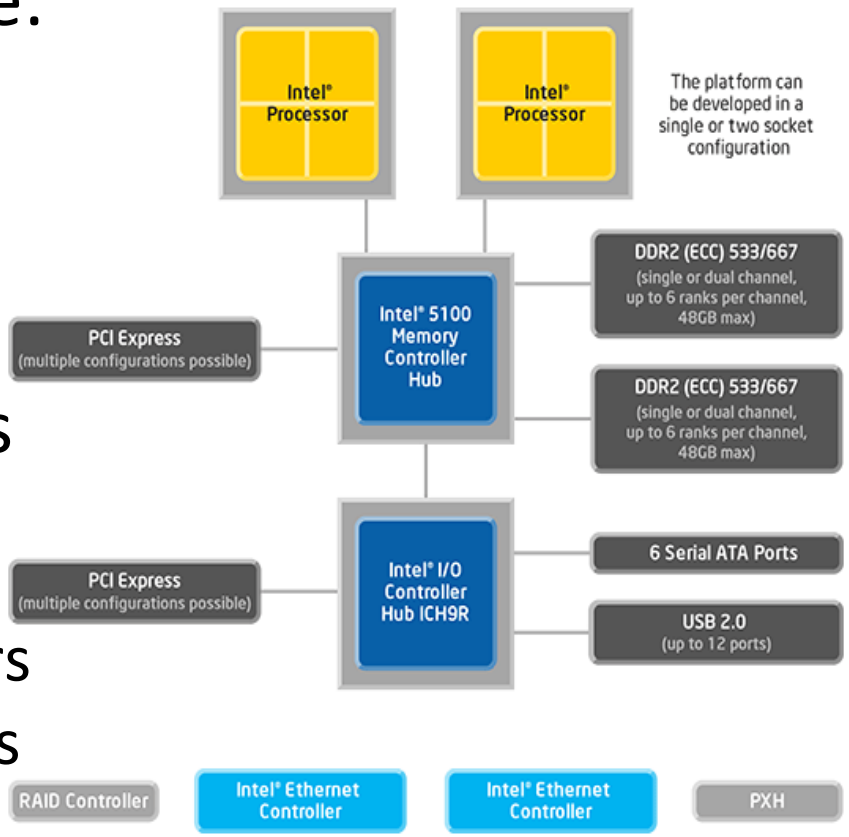
Machines multiprocesseurs à mémoire partagée

- SMP
  - Mémoire centralisée
- NUMA
  - Mémoire répartie



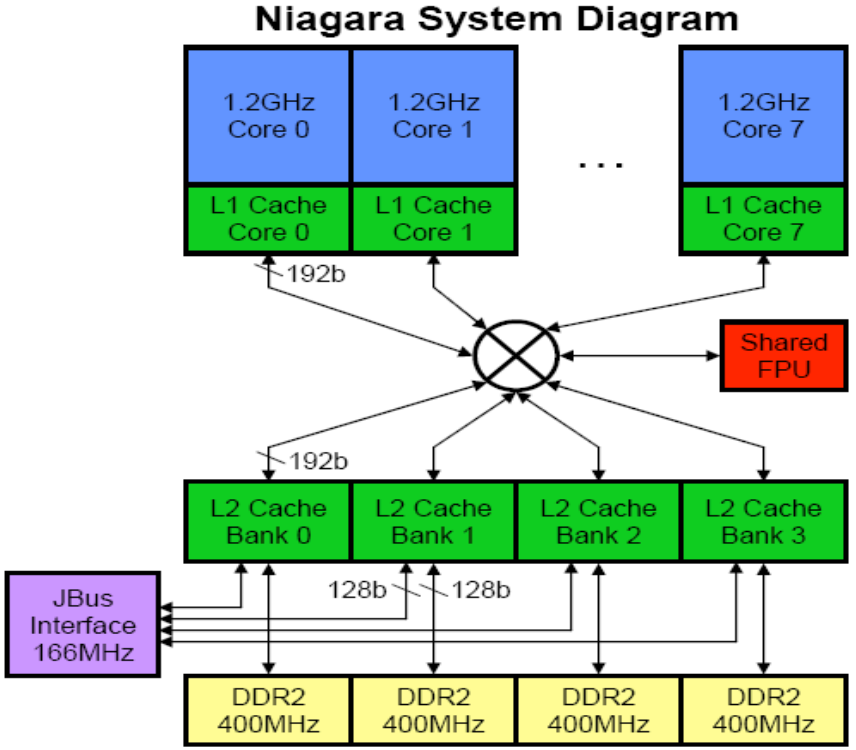
SMP

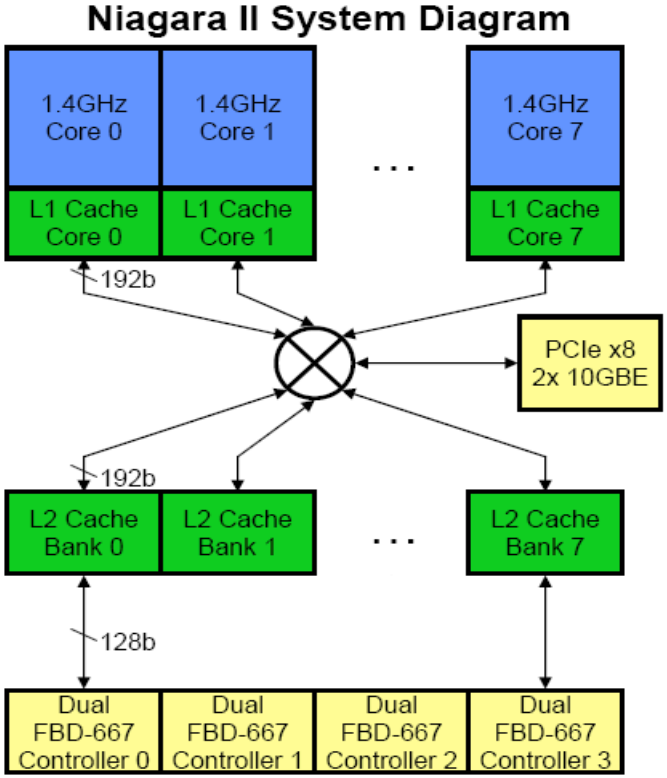
- Organisation classique:
  - Mémoire
  - Bus
  - Cache
  - Cœur
- Le bus synchronise les différents cœurs.
  - Bus unique pas plus d'une dizaine de cœurs
  - Bus en étoile (avec des filtres)



SMP

- Graup : le cache L2 est devant la mémoire





SMP
synchronisation de la mémoire

- Une même donnée peut être à la fois :
  - En mémoire,
  - Dans un cache, dans des caches,
  - Dans un registre, dans des registres,
- Quelle est la bonne version ?
  - Faire en sorte qu'il n'y ait qu'une version
    - Seulement aux yeux du programmeur
    - Donner au programmeur la même vision de la mémoire que sur une machine monoprocesseur programmée à l'aide de threads (exécutés en séquence).

SMP
synchronisation de la mémoire

→ Consistance séquentielle (Lamport 1979), Sémantique de l'entrelacement :

« Un multiprocesseur est séquentiellement consistant si toute exécution résulte d'un entrelacement des séquences d'instructions exécutées par les processeurs et qui préserve chacune des séquences. En particulier tous les processeurs voient les écritures dans le même ordre. »

- Exemple : au départ { x = 0, y = 0} on lance 2 threads :

  T1 { x = 1 ; print y }

  T2 { y = 1; print x }

  On ne devrait pas voir 0 0.

SMP
synchronisation de la mémoire

→ Consistance séquentielle (Lamport 1979), Sémantique de l'entrelacement :

« Un multiprocesseur est séquentiellement consistant si toute exécution résulte d'un entrelacement des séquences d'instructions exécutées par les processeurs et qui préserve chacune des séquences. En particulier tous les processeurs voient les écritures dans le même ordre. »

- Exemple : au départ { x = 0, y = 0} on lance 2 threads :

  T1 { x = 1 ; print y }

  T2 { y = 1; print x }

On ne devrait pas voir 0 0.

SMP
synchronisation de la mémoire

→ Consistance séquentielle (Lamport 1979), Sémantique de l'entrelacement :

« Un multiprocesseur est séquentiellement consistant si toute exécution résulte d'un entrelacement des séquences d'instructions exécutées par les processeurs et qui préserve chacune des séquences. En particulier tous les processeurs voient les écritures dans le même ordre. »

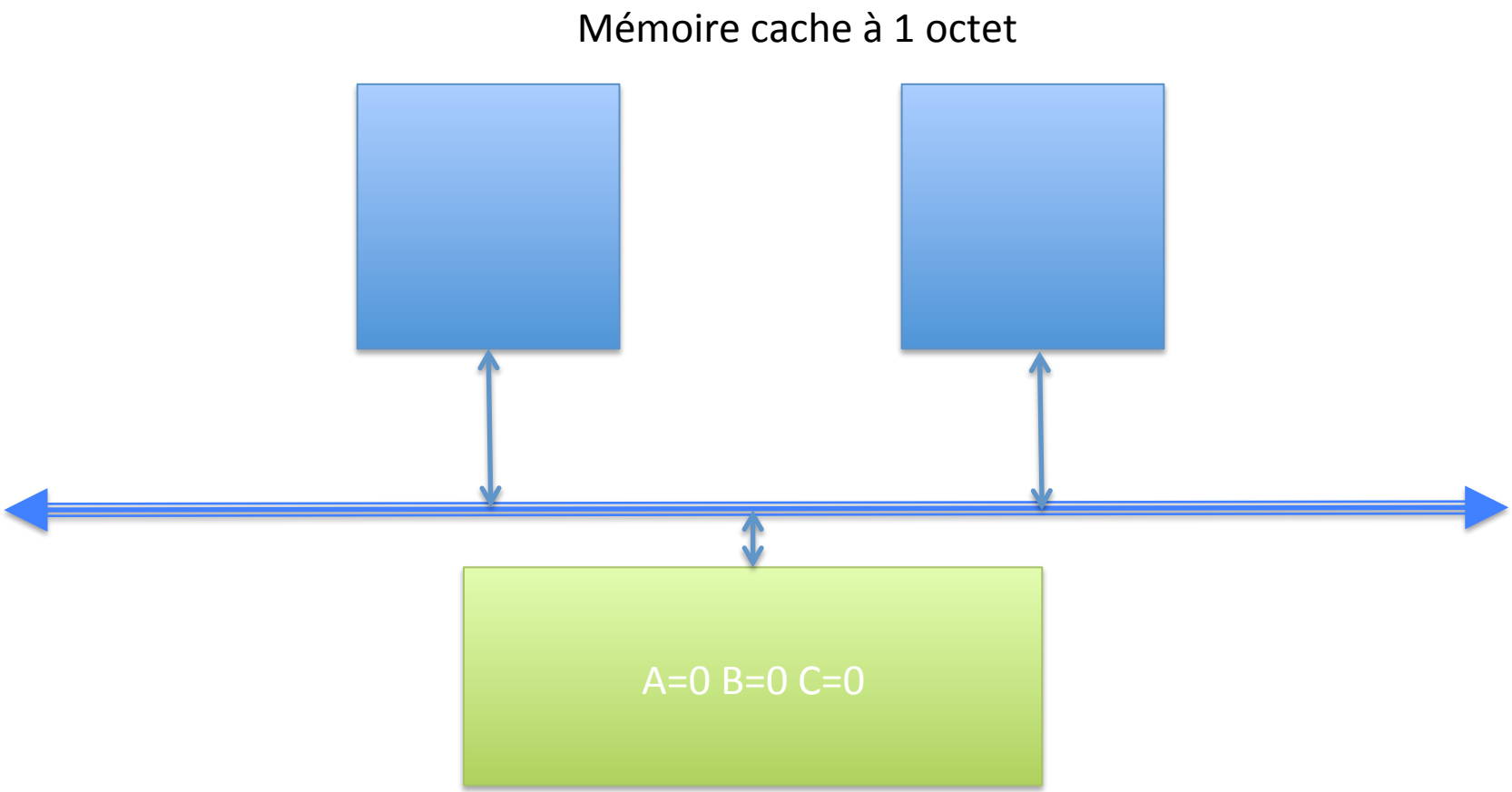
- Exemple : au départ { x = 0, y = 0} on lance 2 threads :

  T1 { x = 1 ; print y }

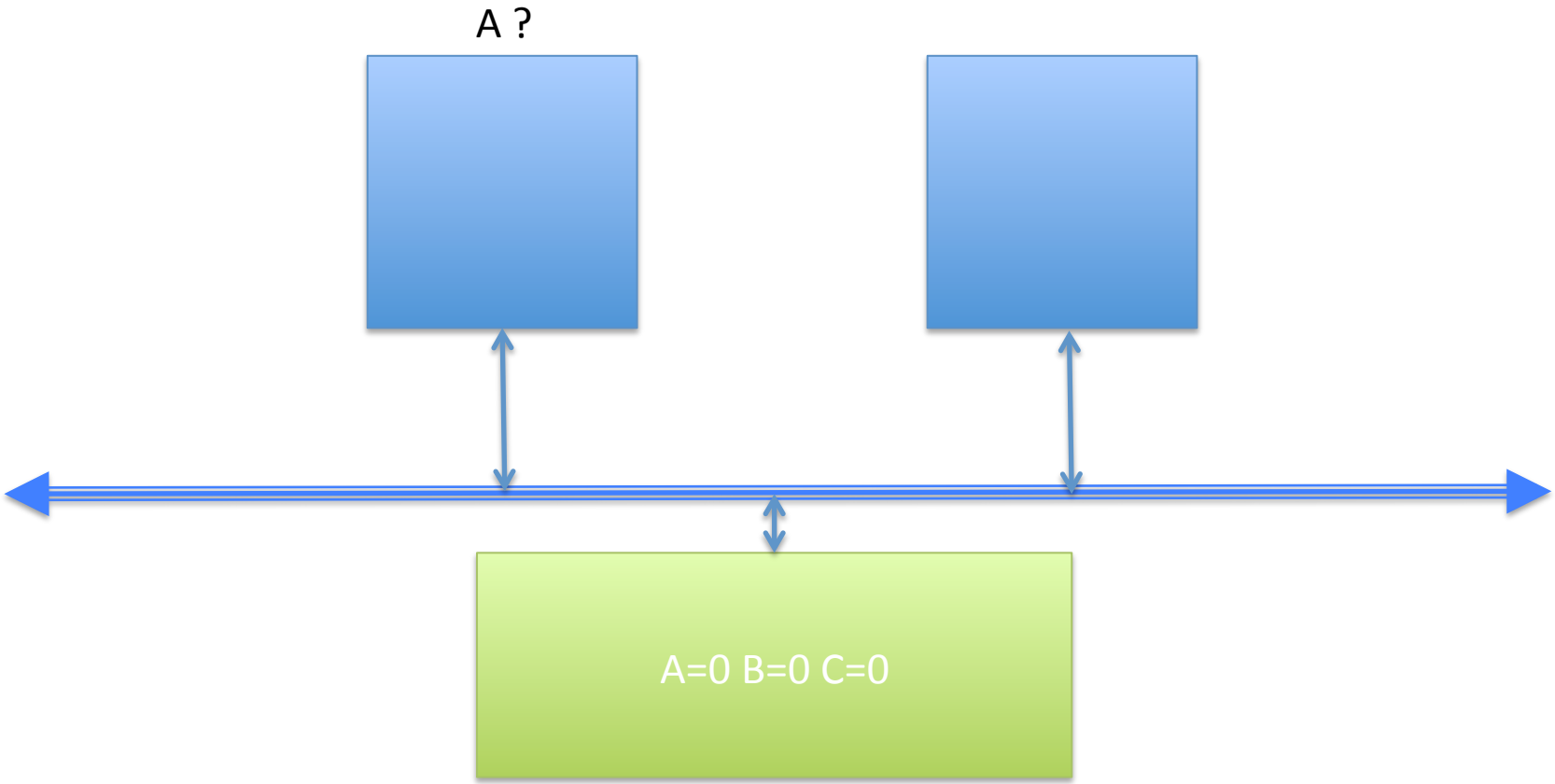
  T2 { y = 1; print x }

  On ne devrait pas voir 0 0.

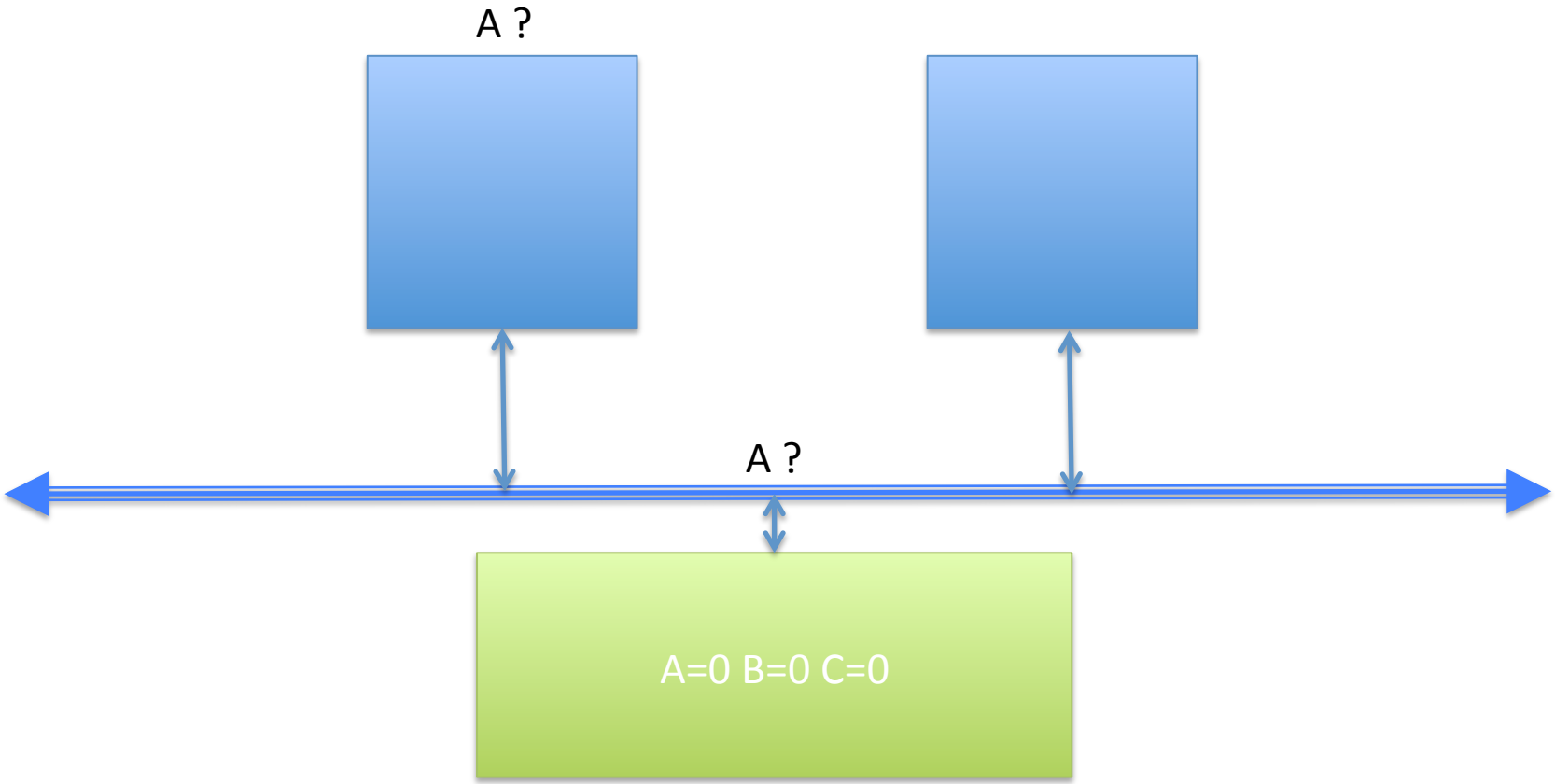
Implémentation de la cohérence séquentielle



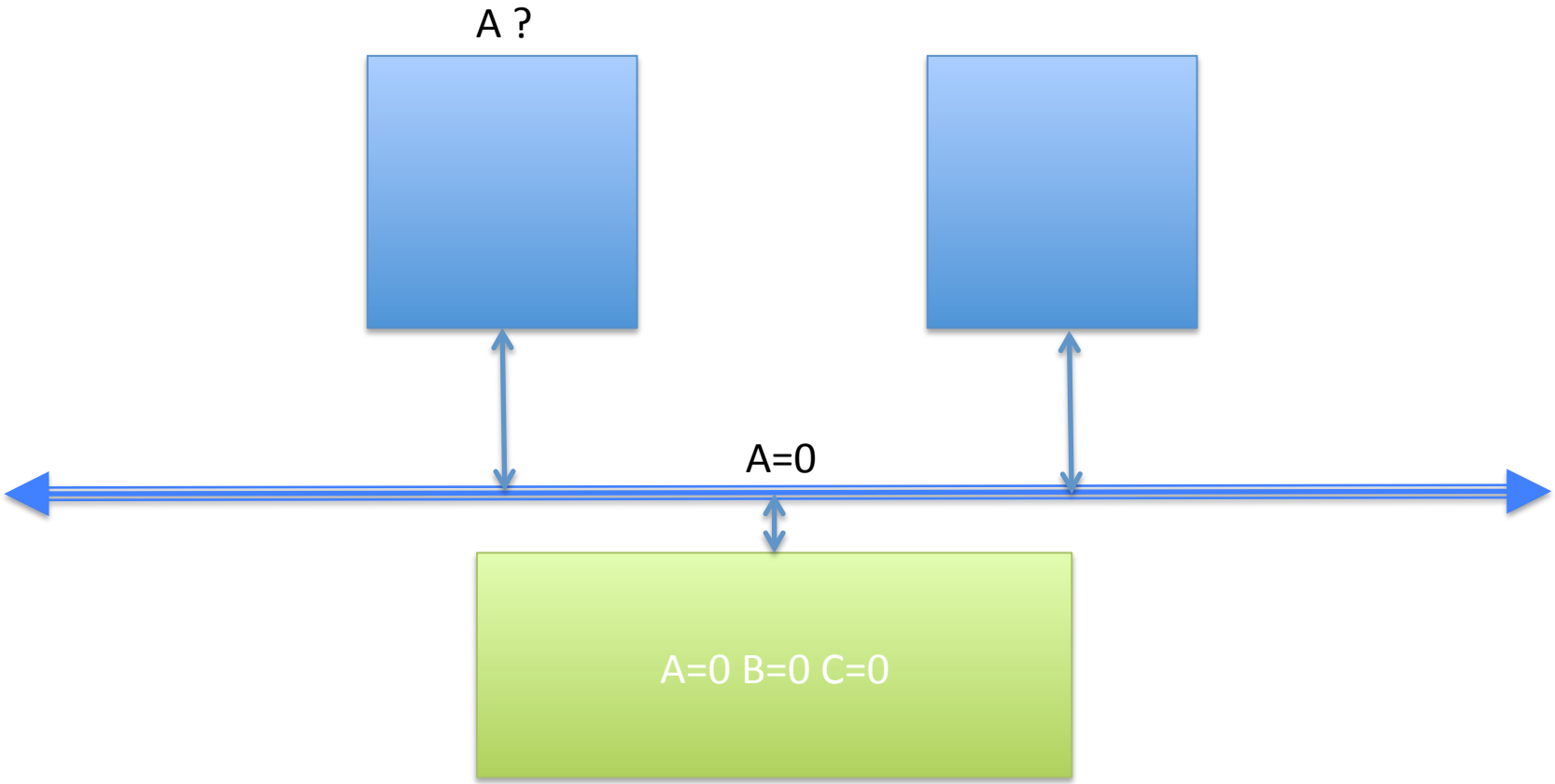
Implémentation de la cohérence séquentielle



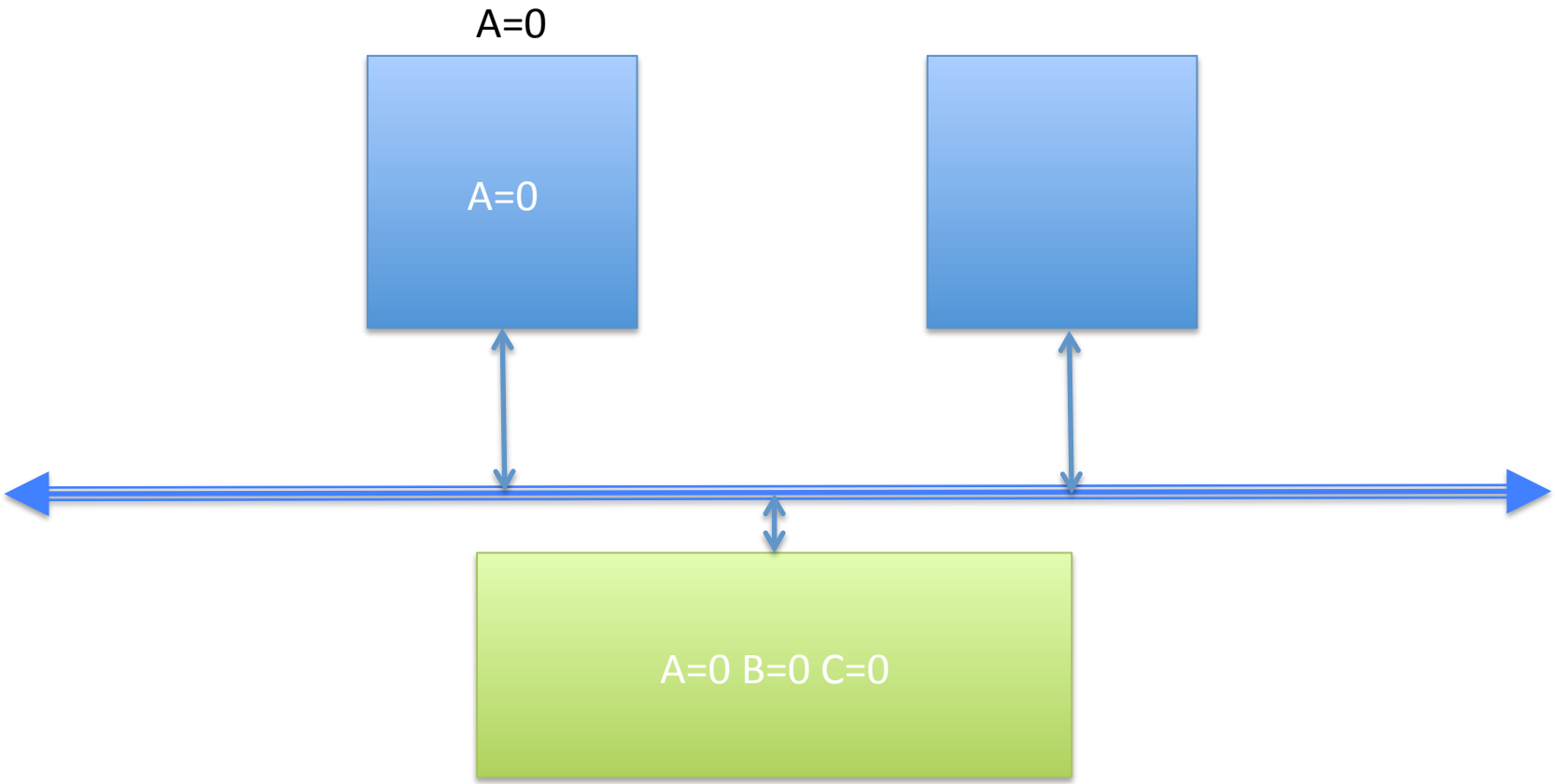
Implémentation de la cohérence séquentielle



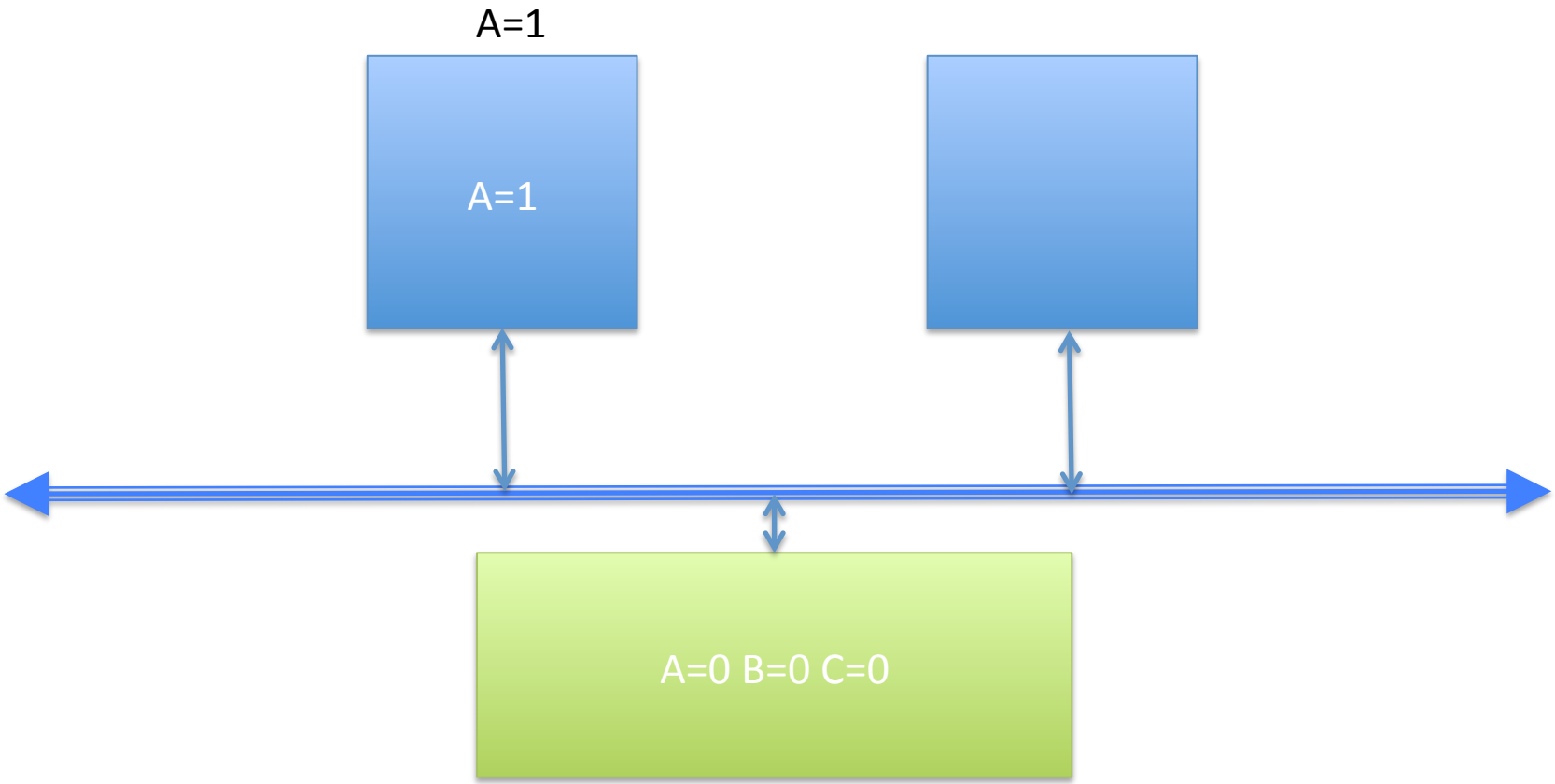
Implémentation de la cohérence séquentielle



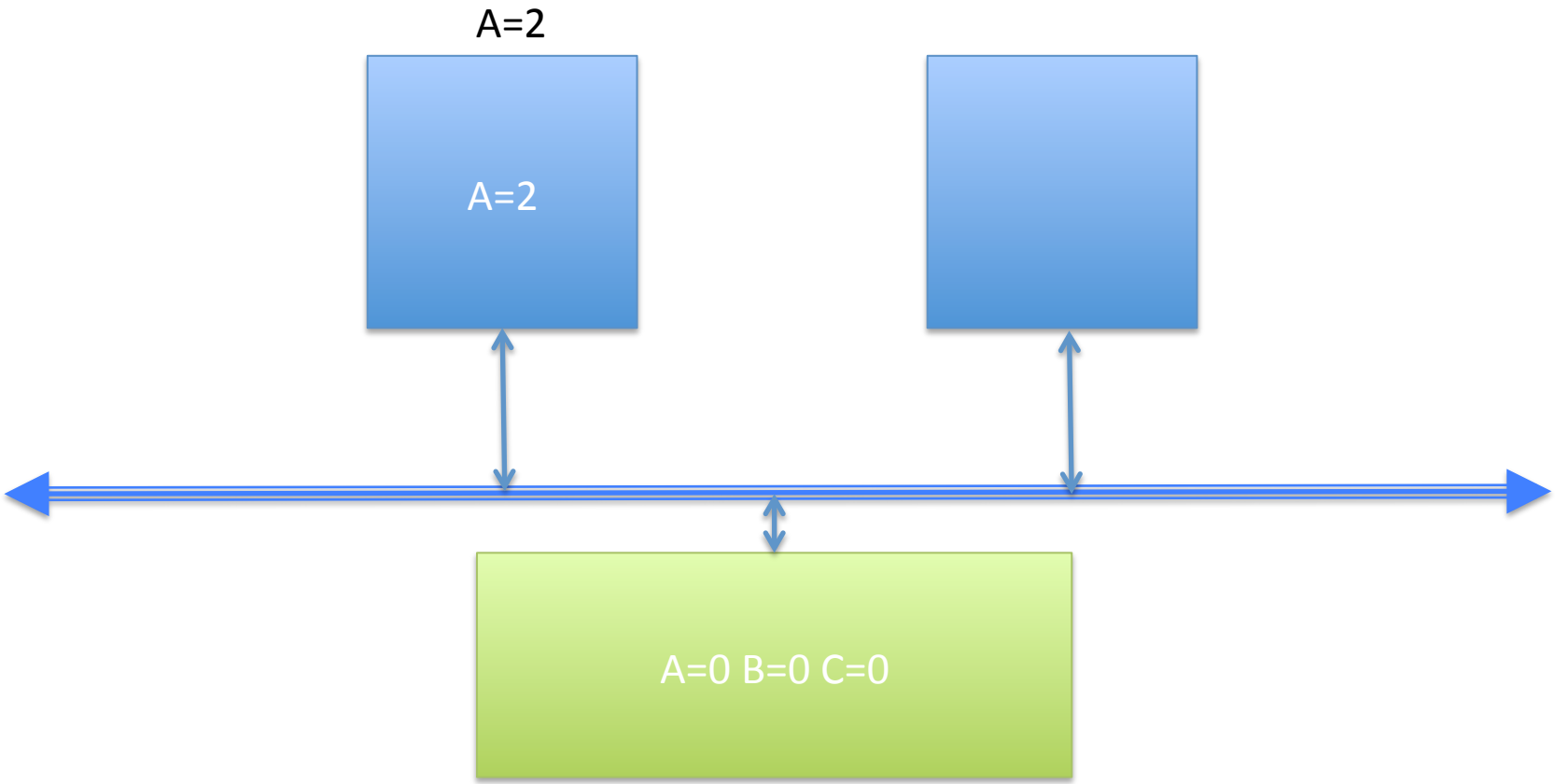
Implémentation de la cohérence séquentielle



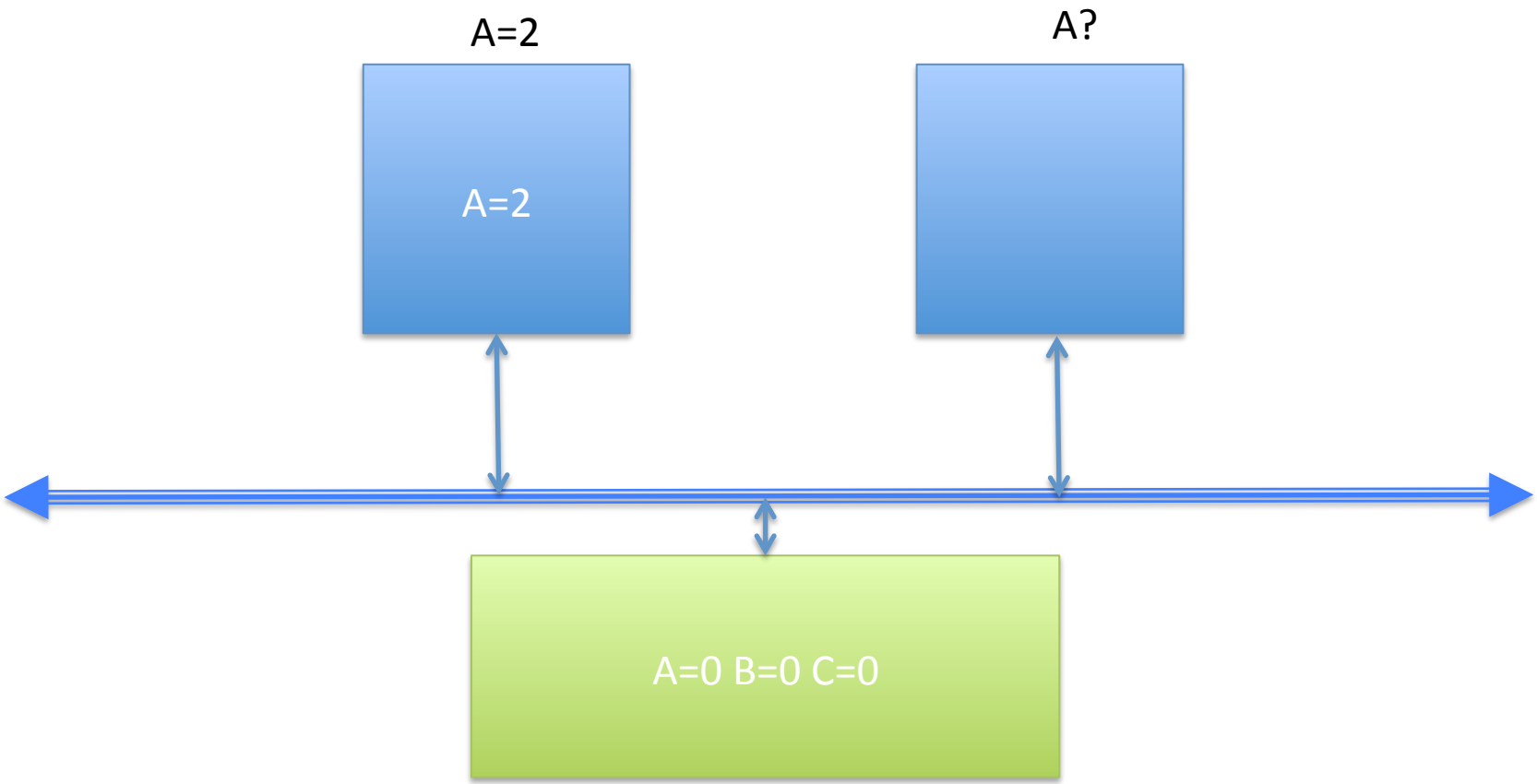
Implémentation de la cohérence séquentielle



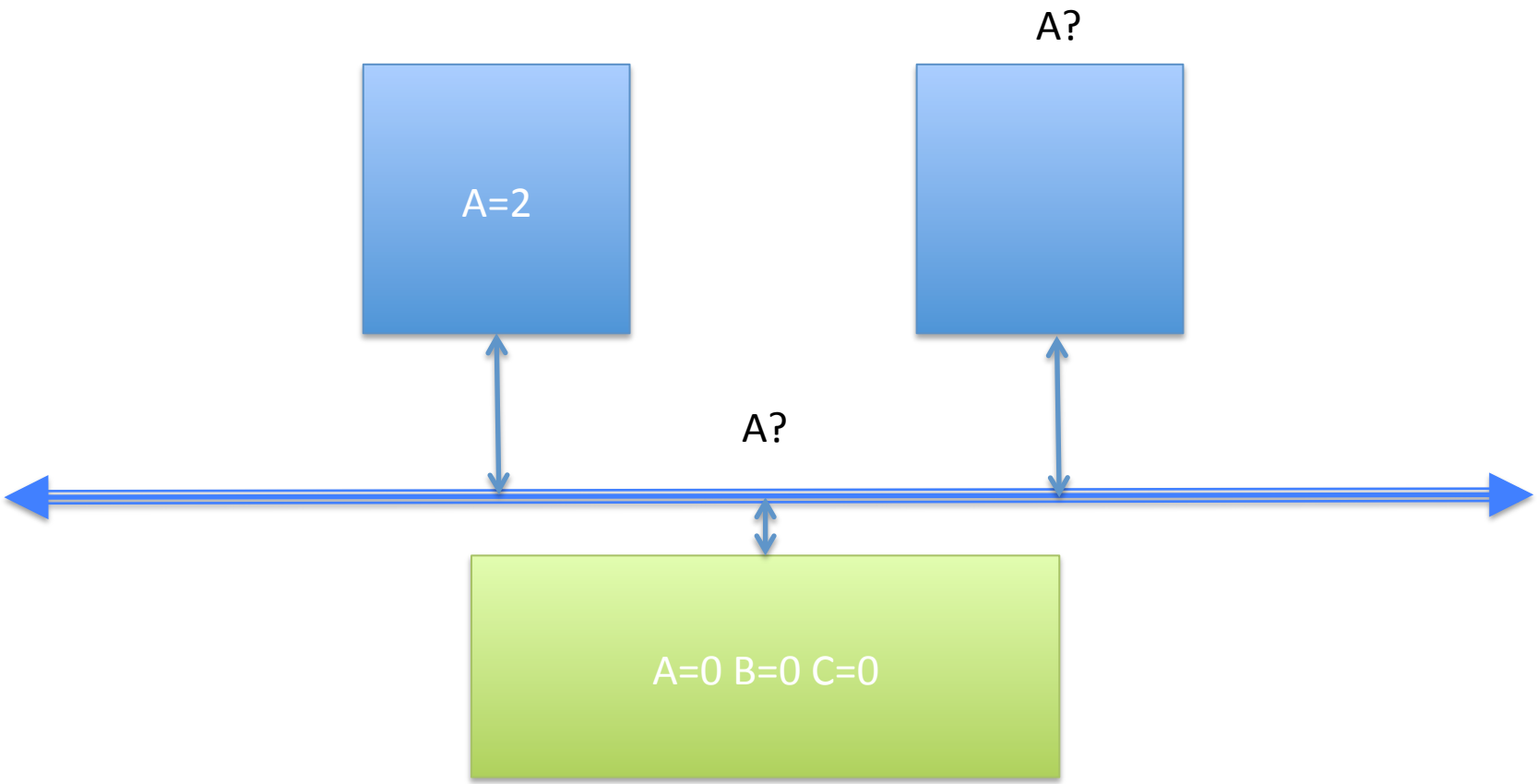
Implémentation de la cohérence séquentielle



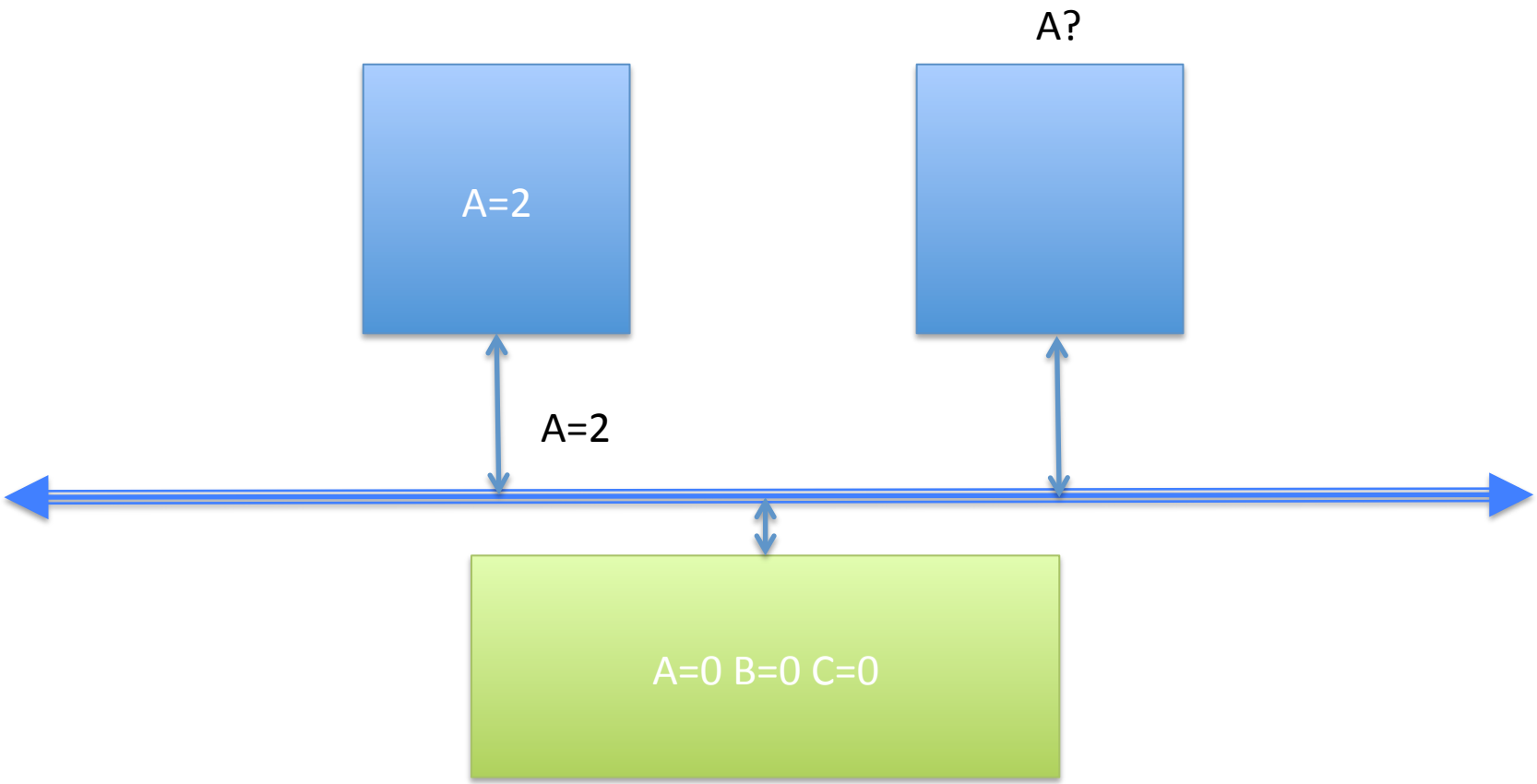
Implémentation de la cohérence séquentielle



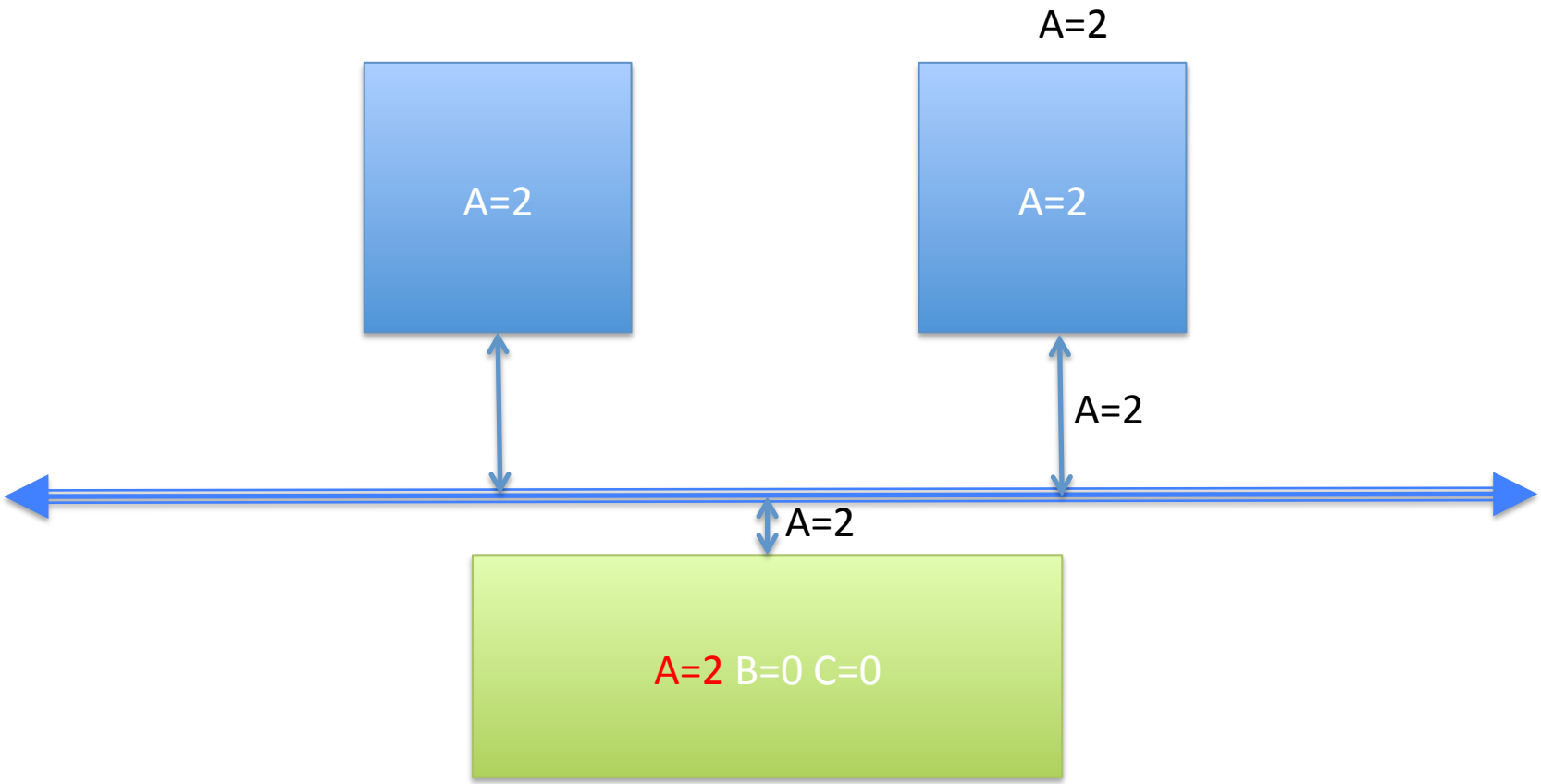
Implémentation de la cohérence séquentielle



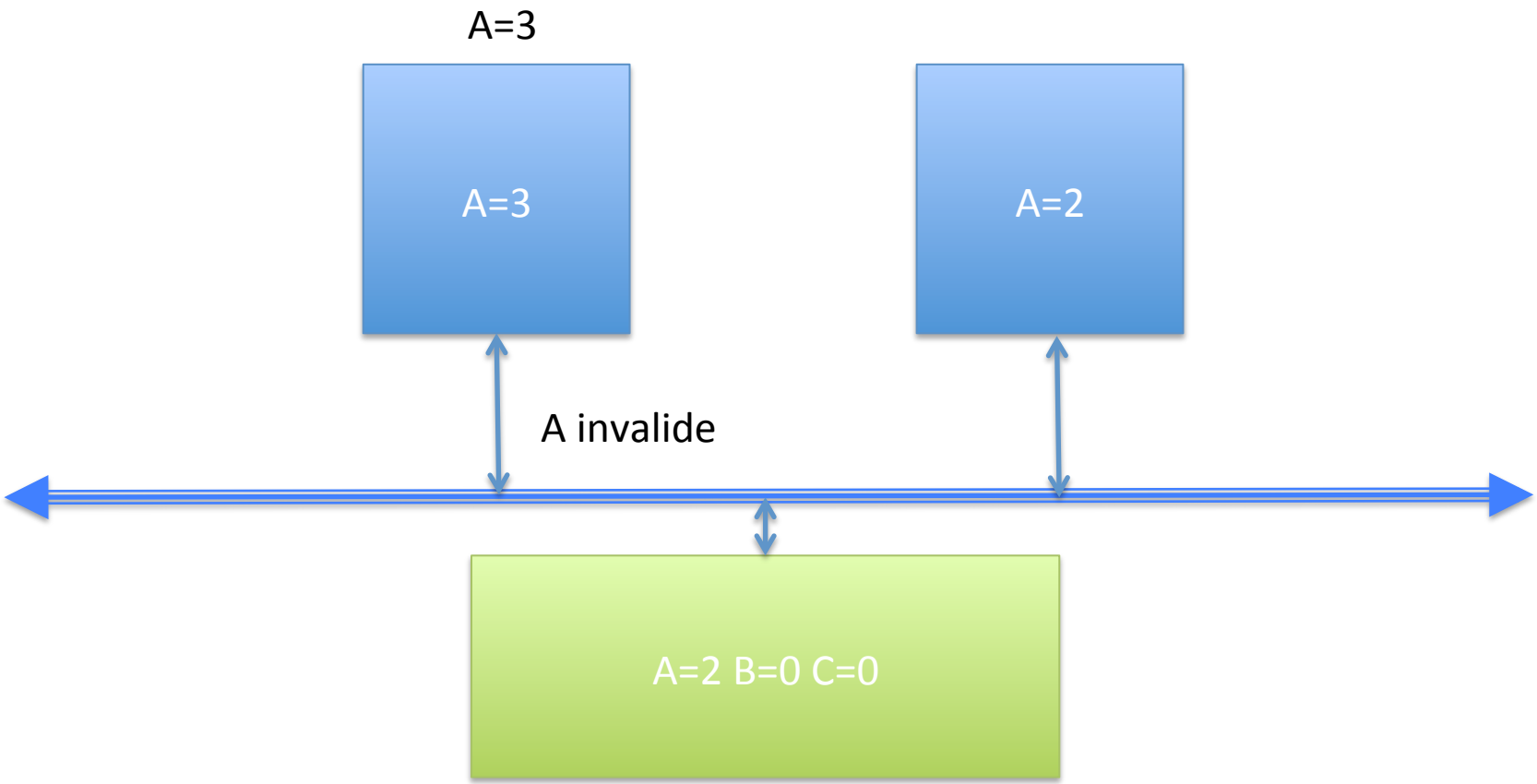
Implémentation de la cohérence séquentielle



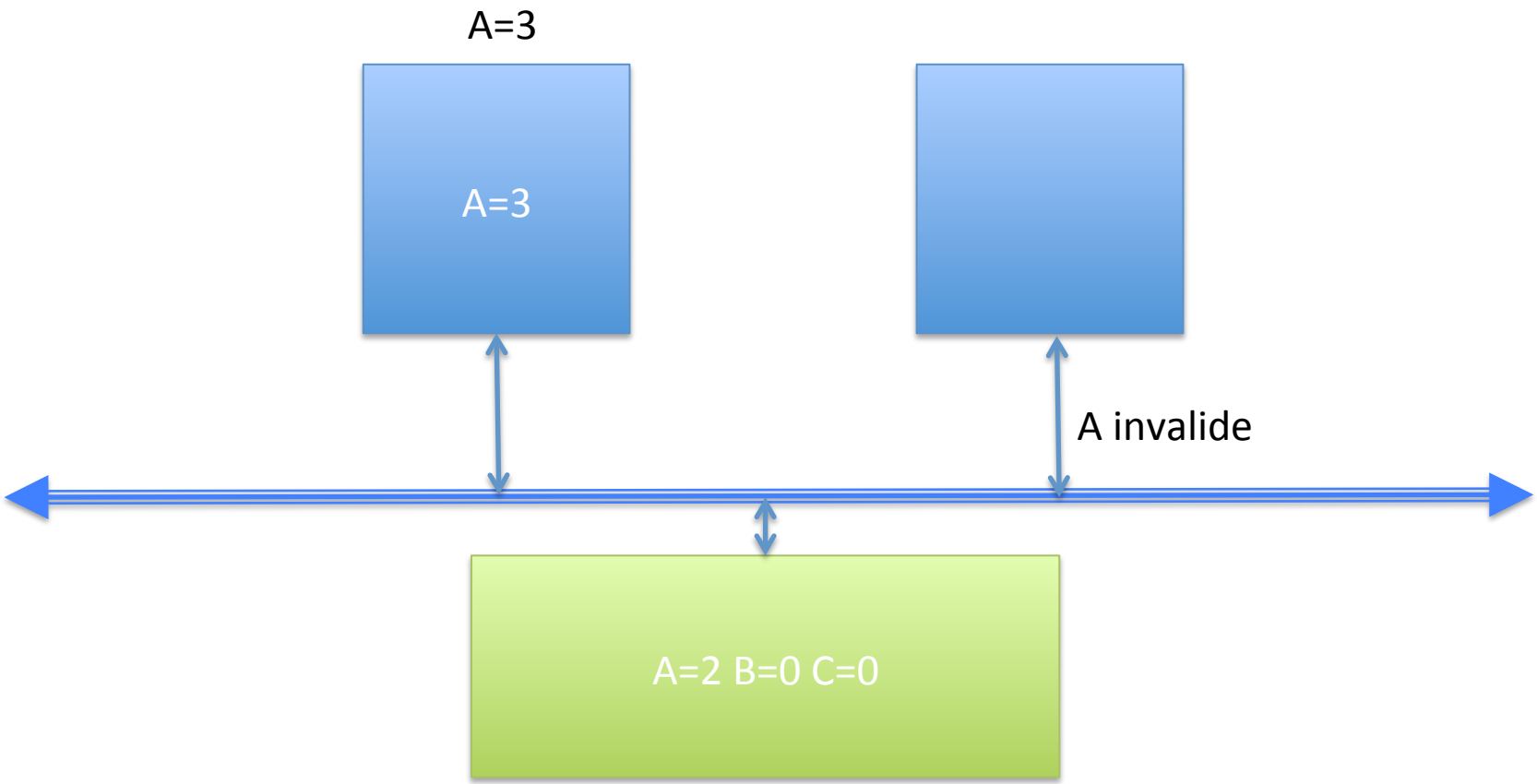
Implémentation de la cohérence séquentielle



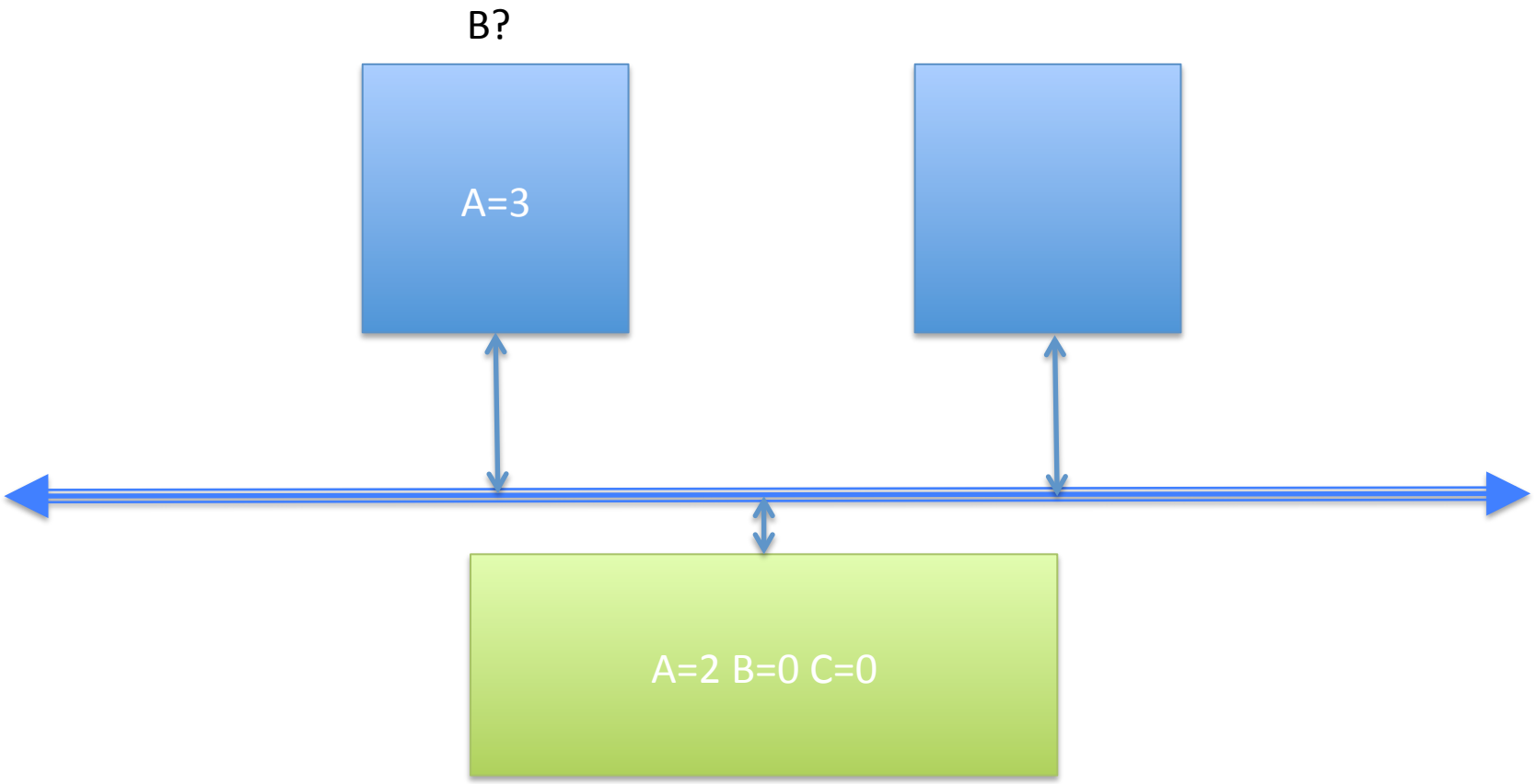
Implémentation de la cohérence séquentielle



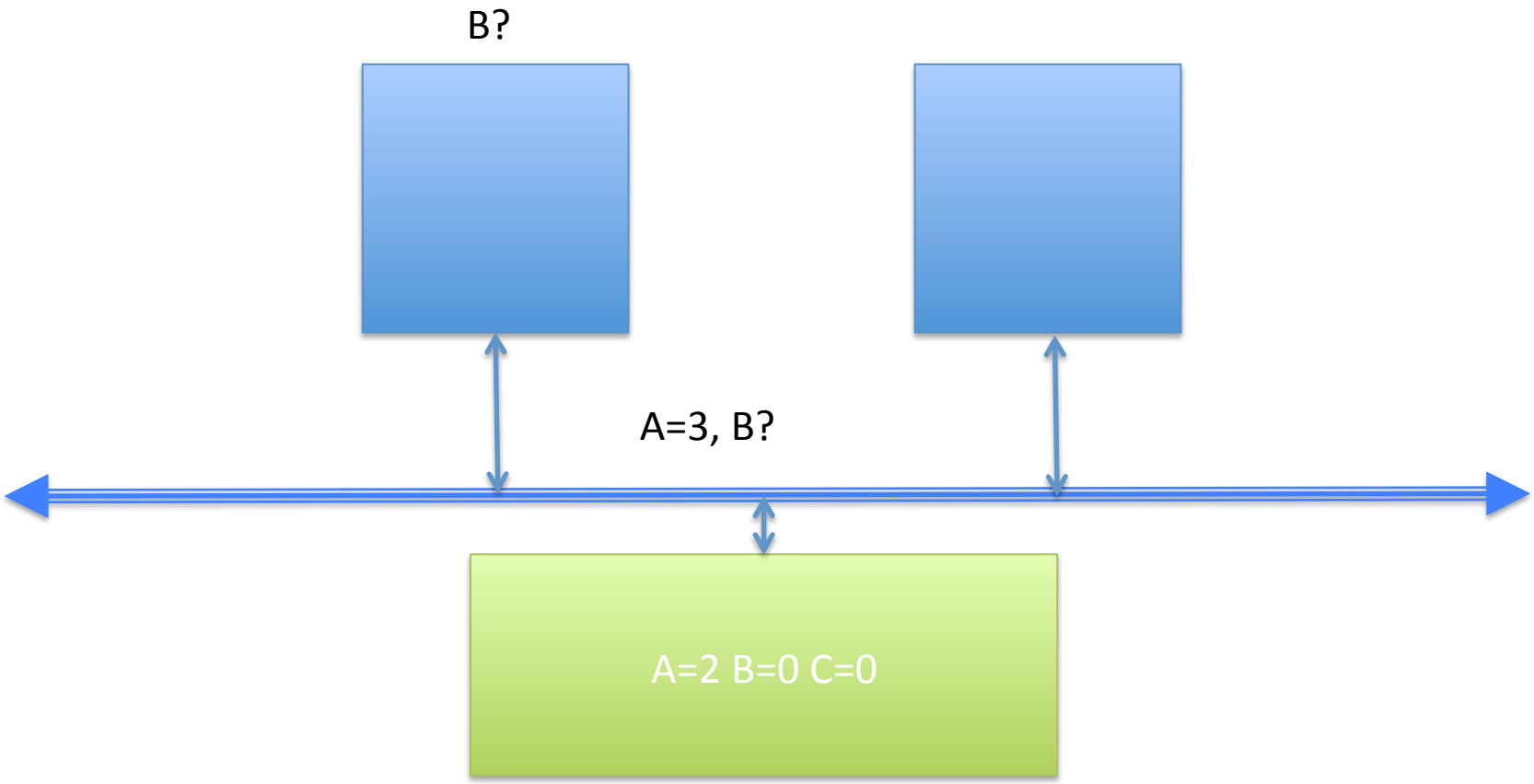
Implémentation de la cohérence séquentielle



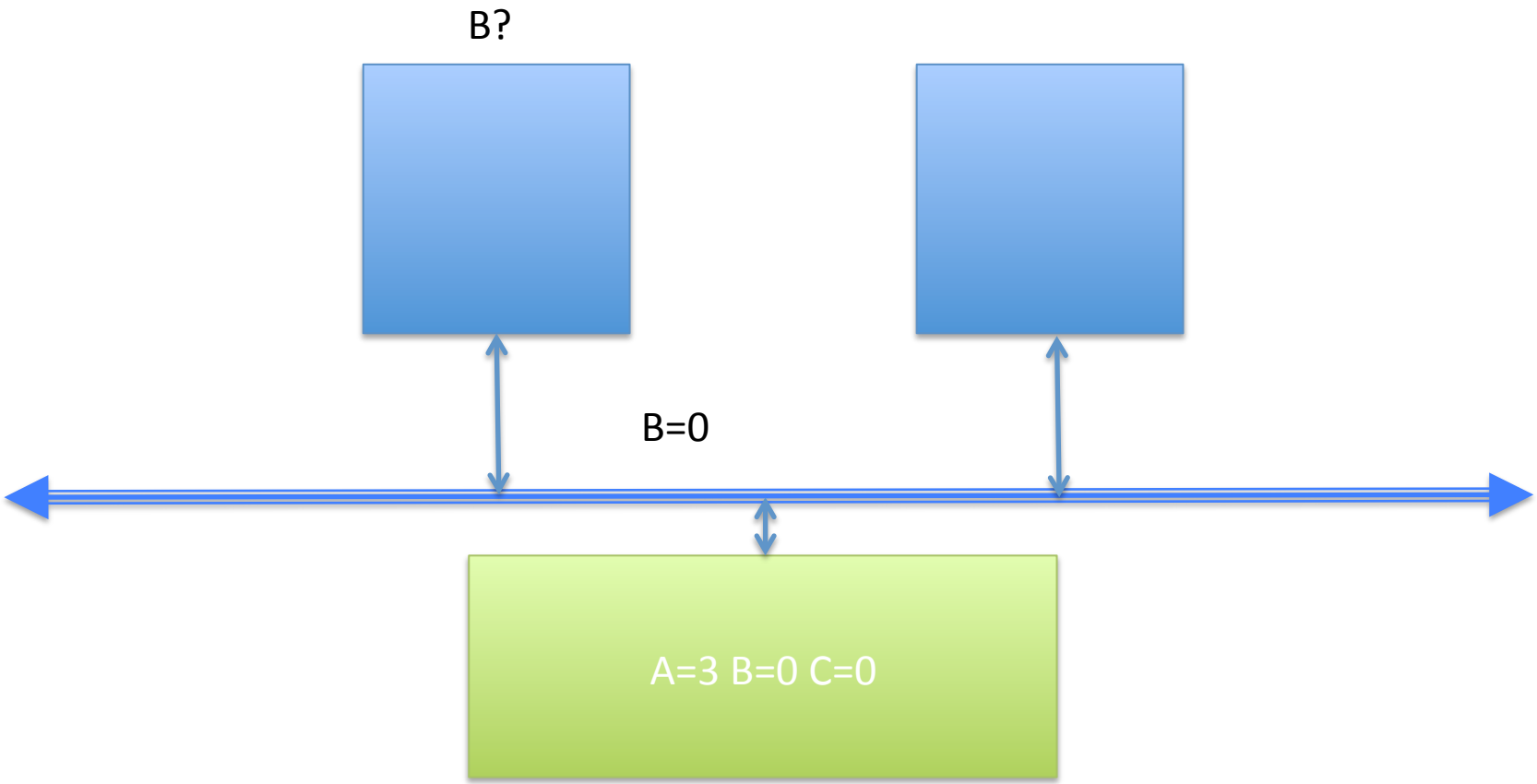
Implémentation de la cohérence séquentielle



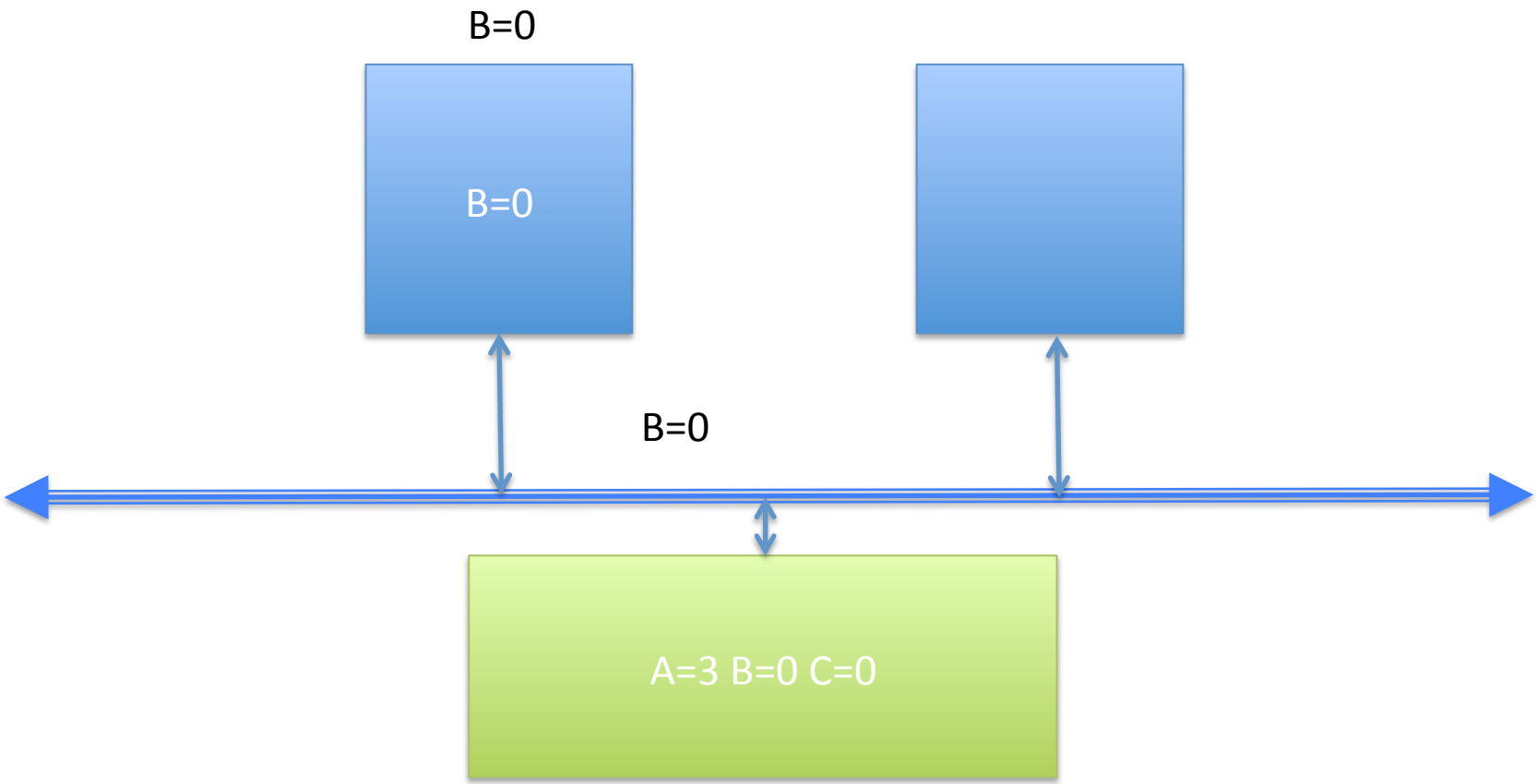
Implémentation de la cohérence séquentielle



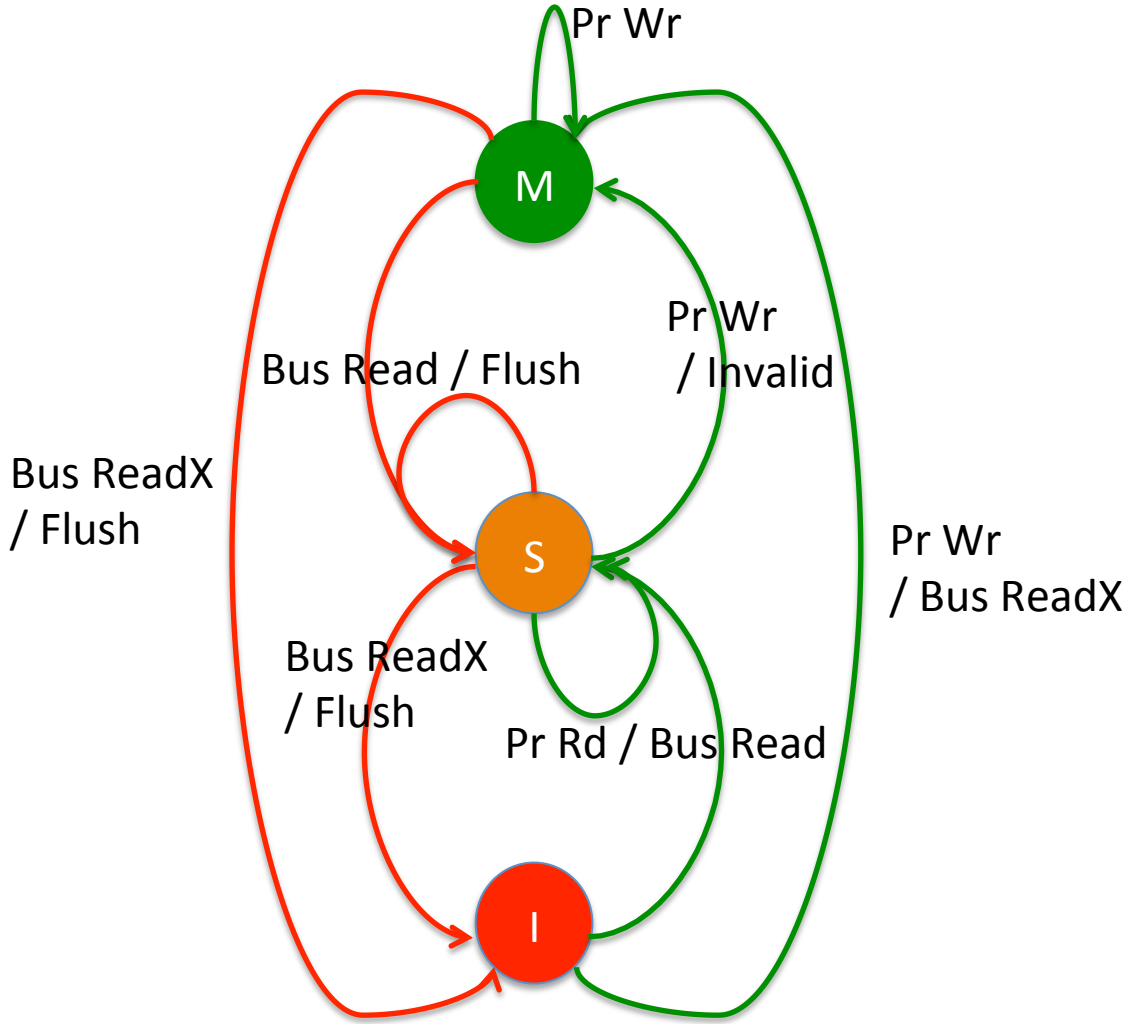
Implémentation de la cohérence séquentielle



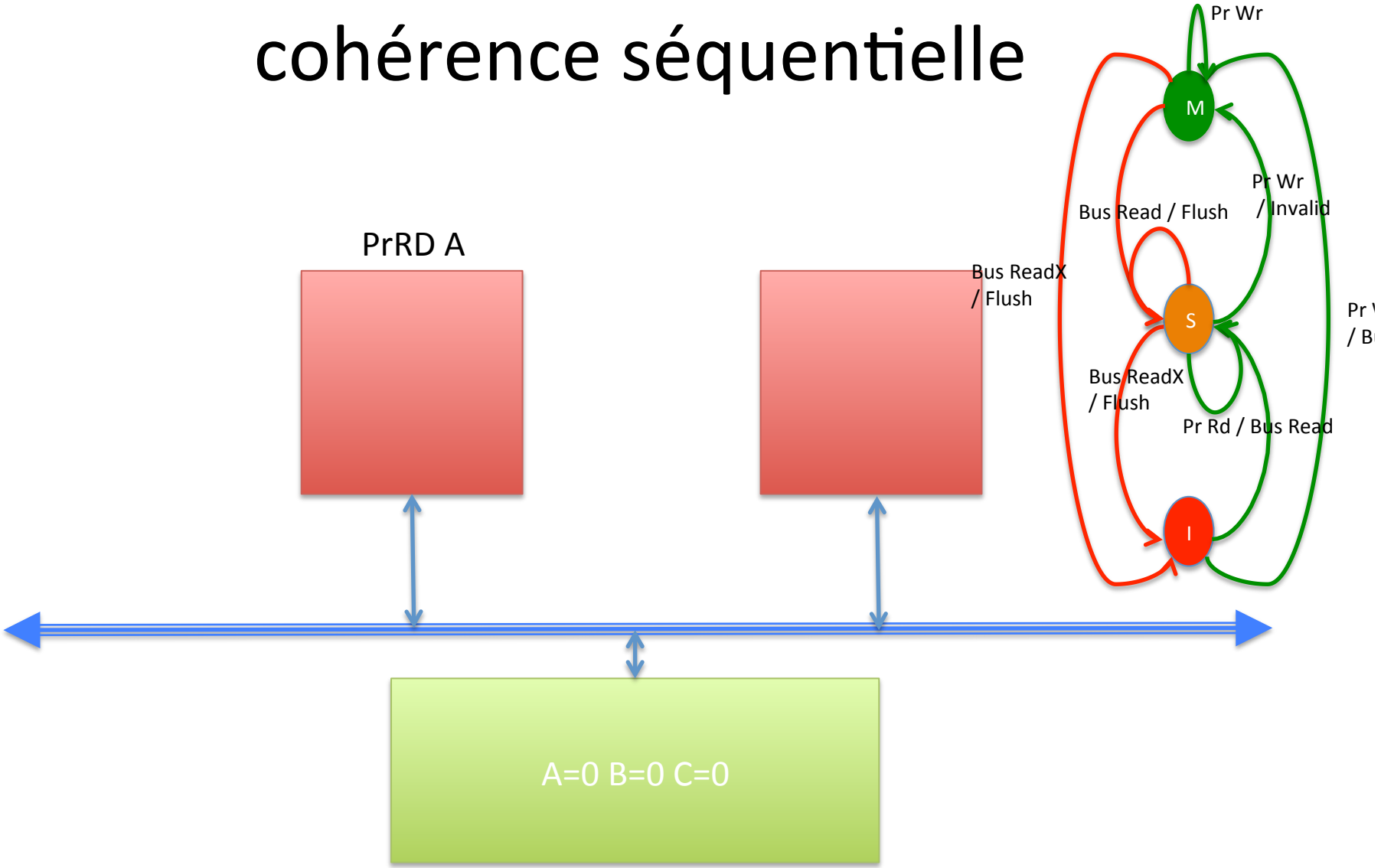
Implémentation de la cohérence séquentielle



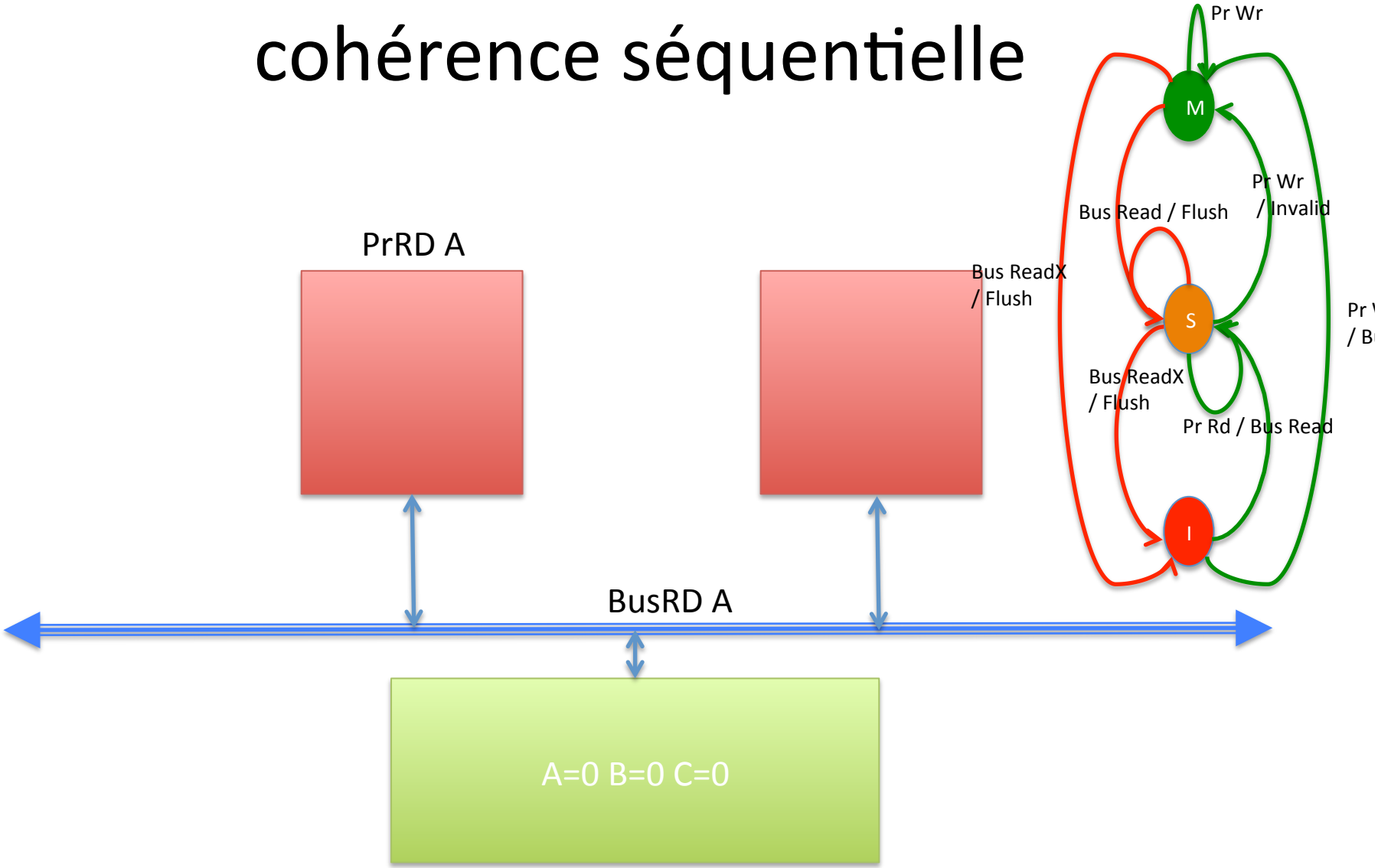
Protocole
Modified Shared Invalid (MSI)



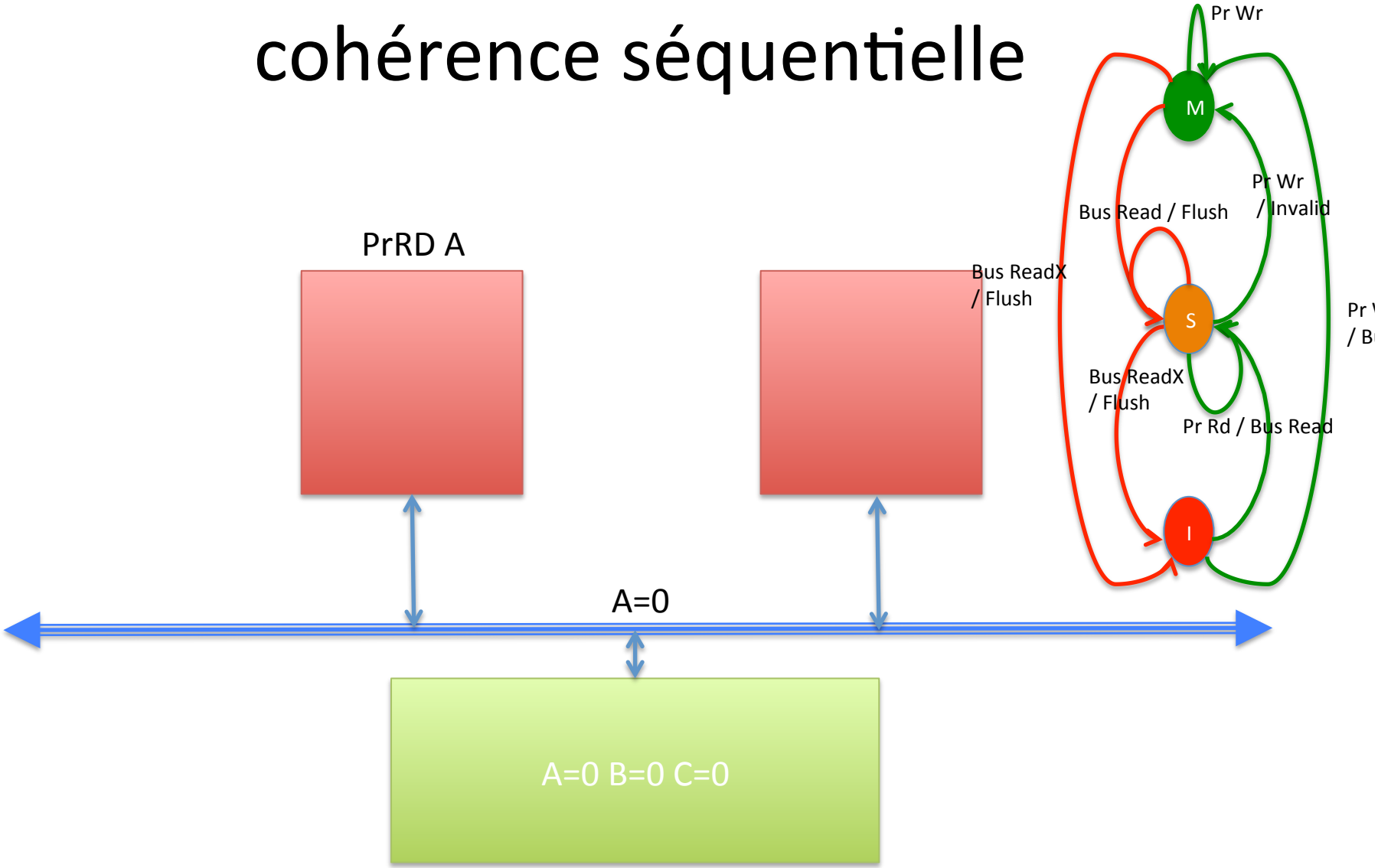
Implémentation de la cohérence séquentielle



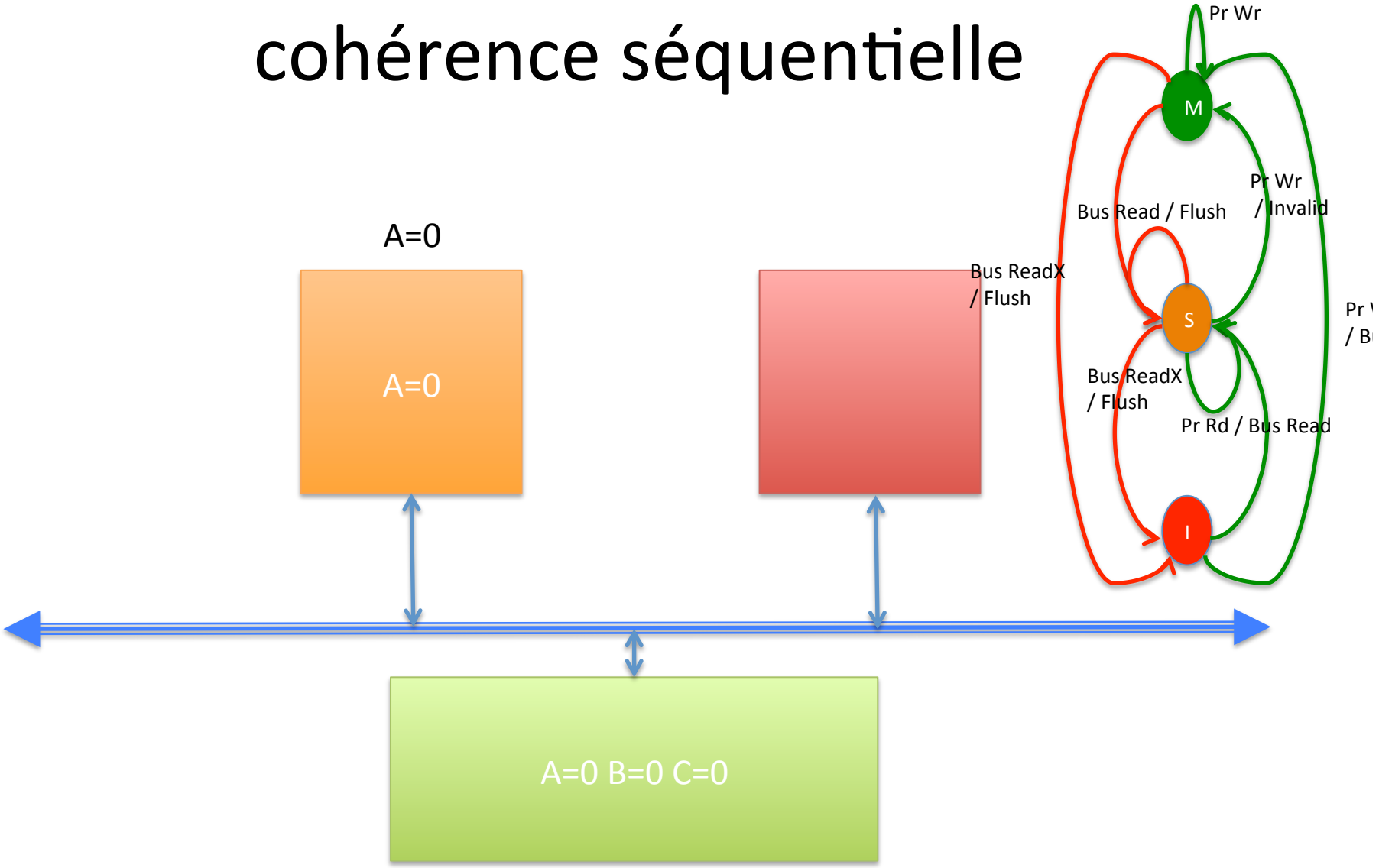
Implémentation de la cohérence séquentielle



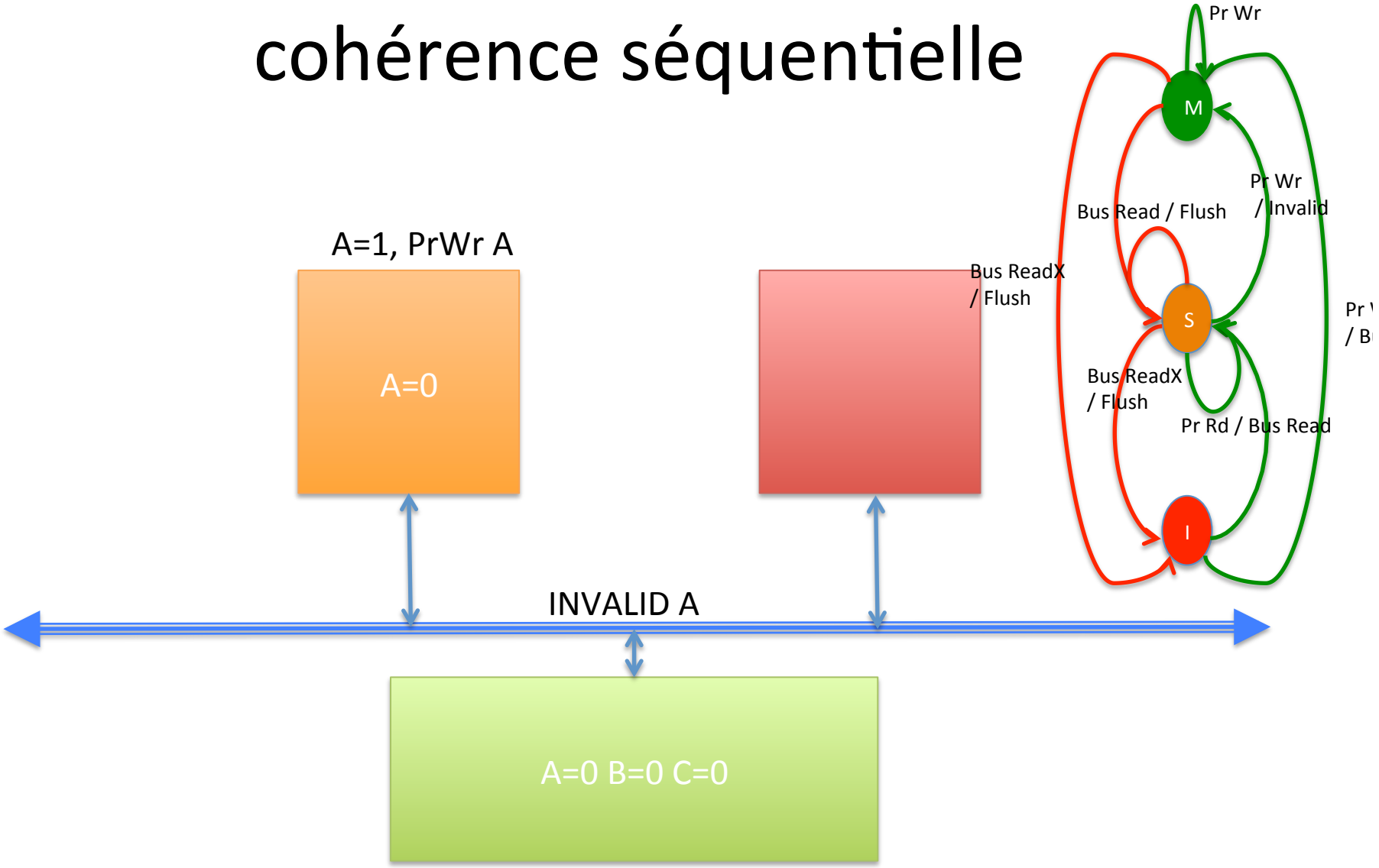
Implémentation de la cohérence séquentielle



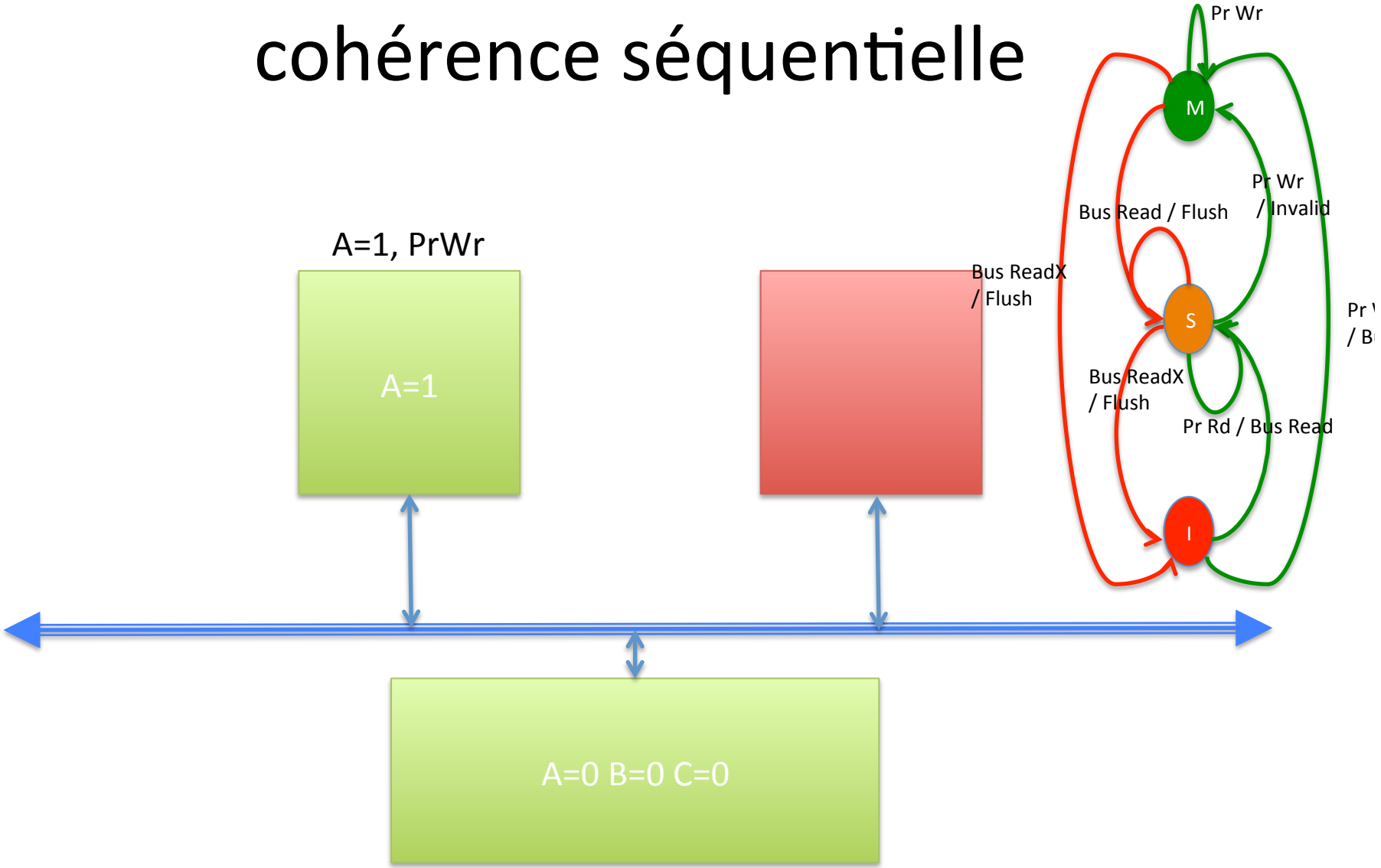
Implémentation de la cohérence séquentielle



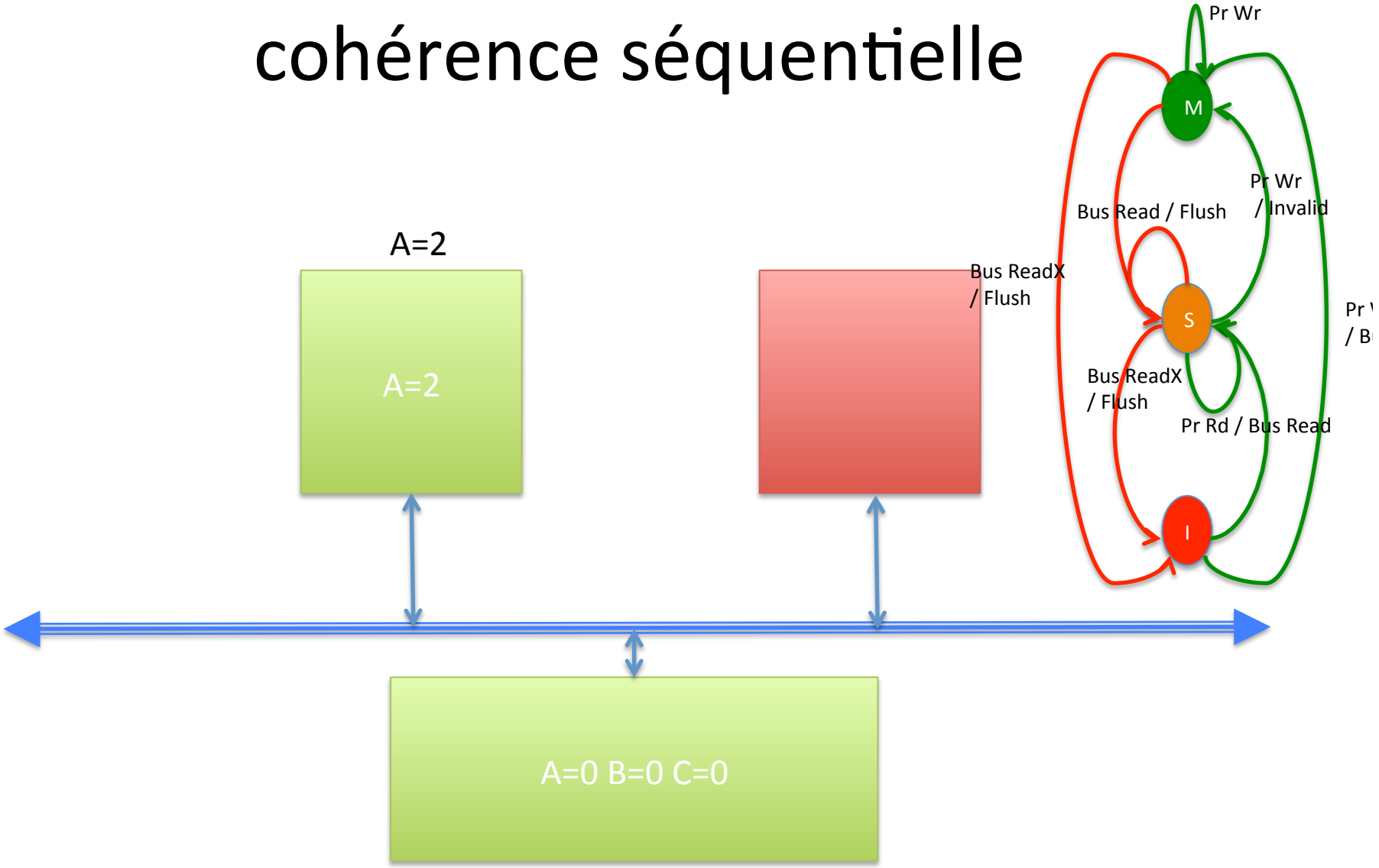
Implémentation de la cohérence séquentielle



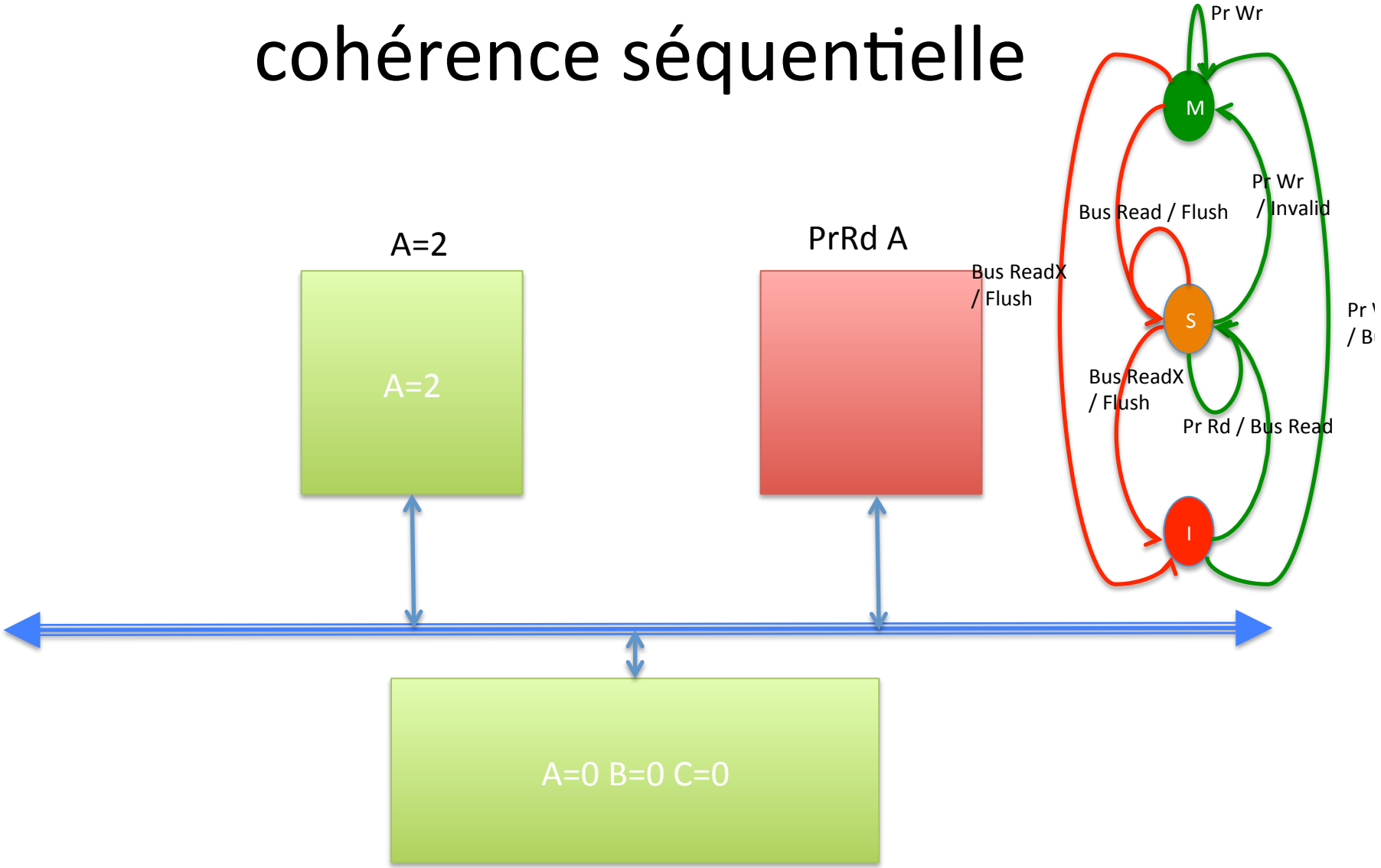
Implémentation de la cohérence séquentielle



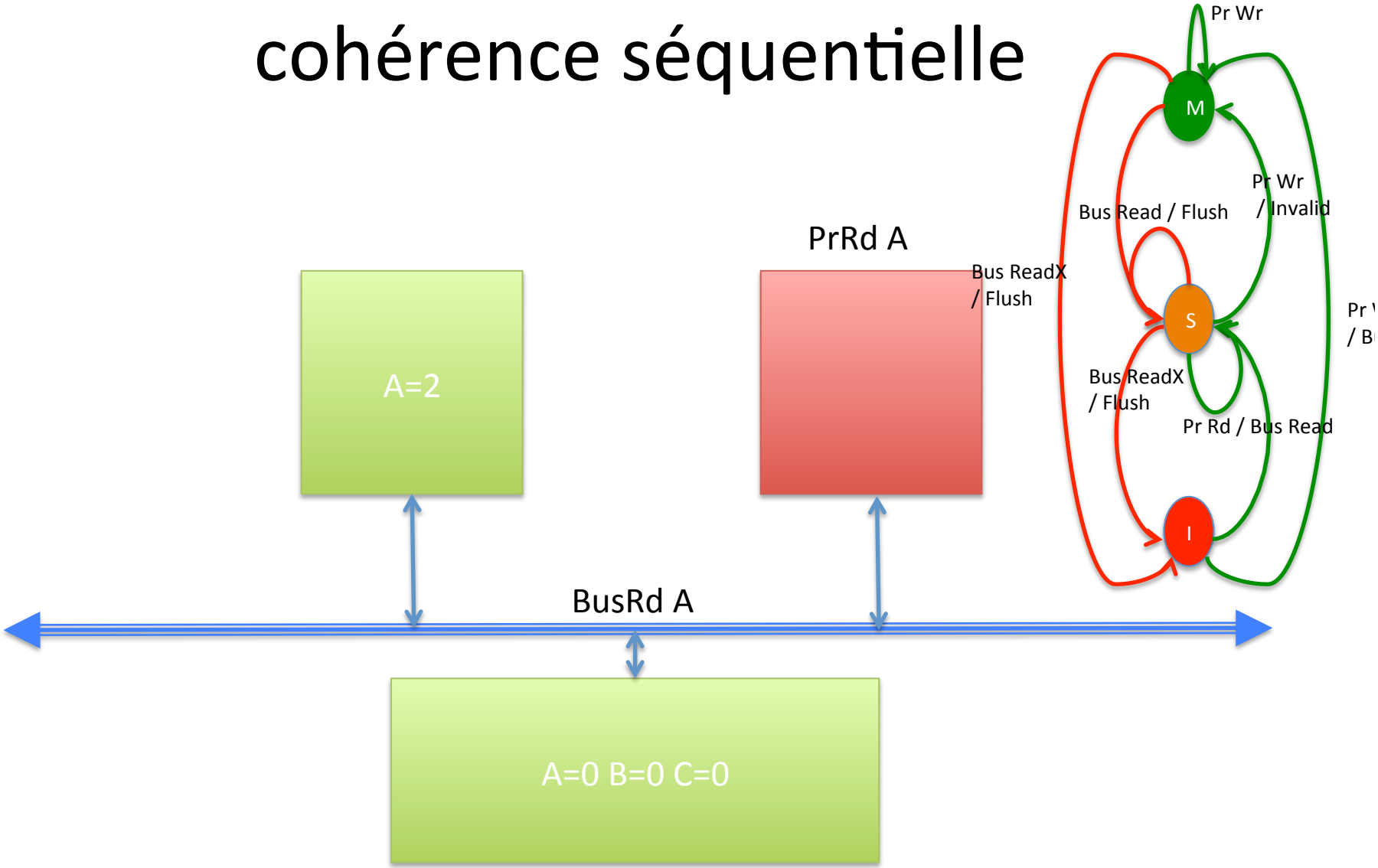
Implémentation de la cohérence séquentielle



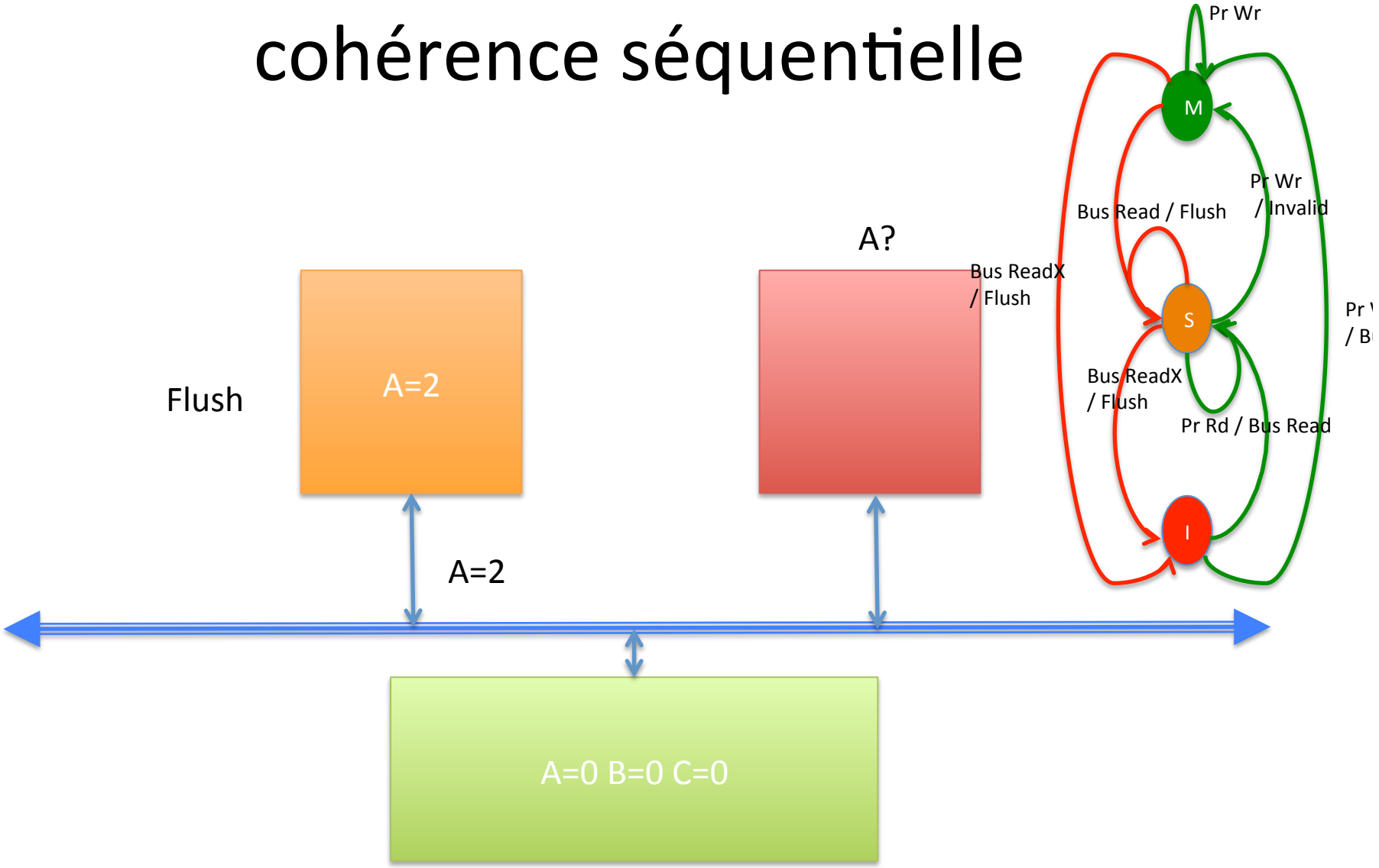
Implémentation de la cohérence séquentielle



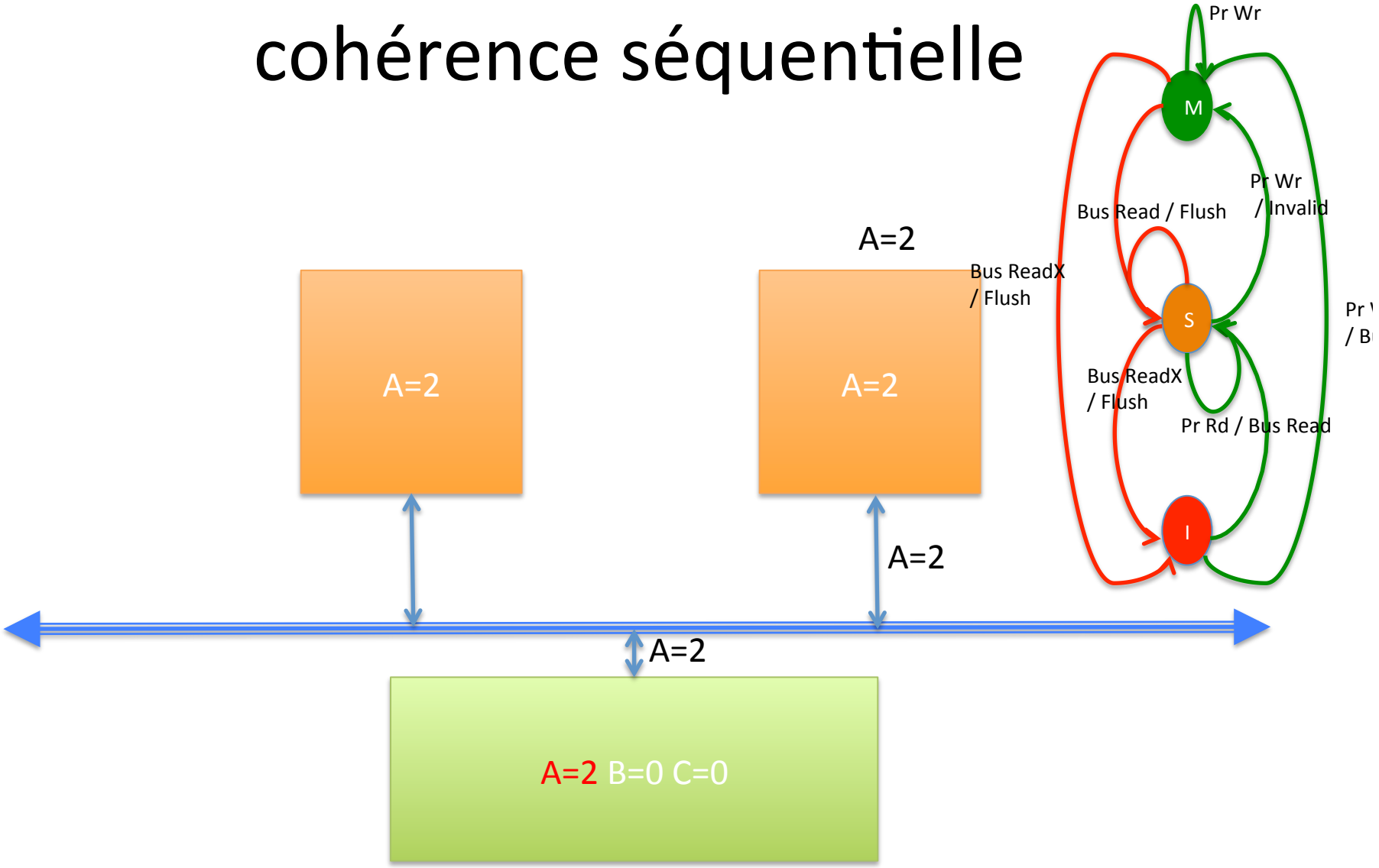
Implémentation de la cohérence séquentielle



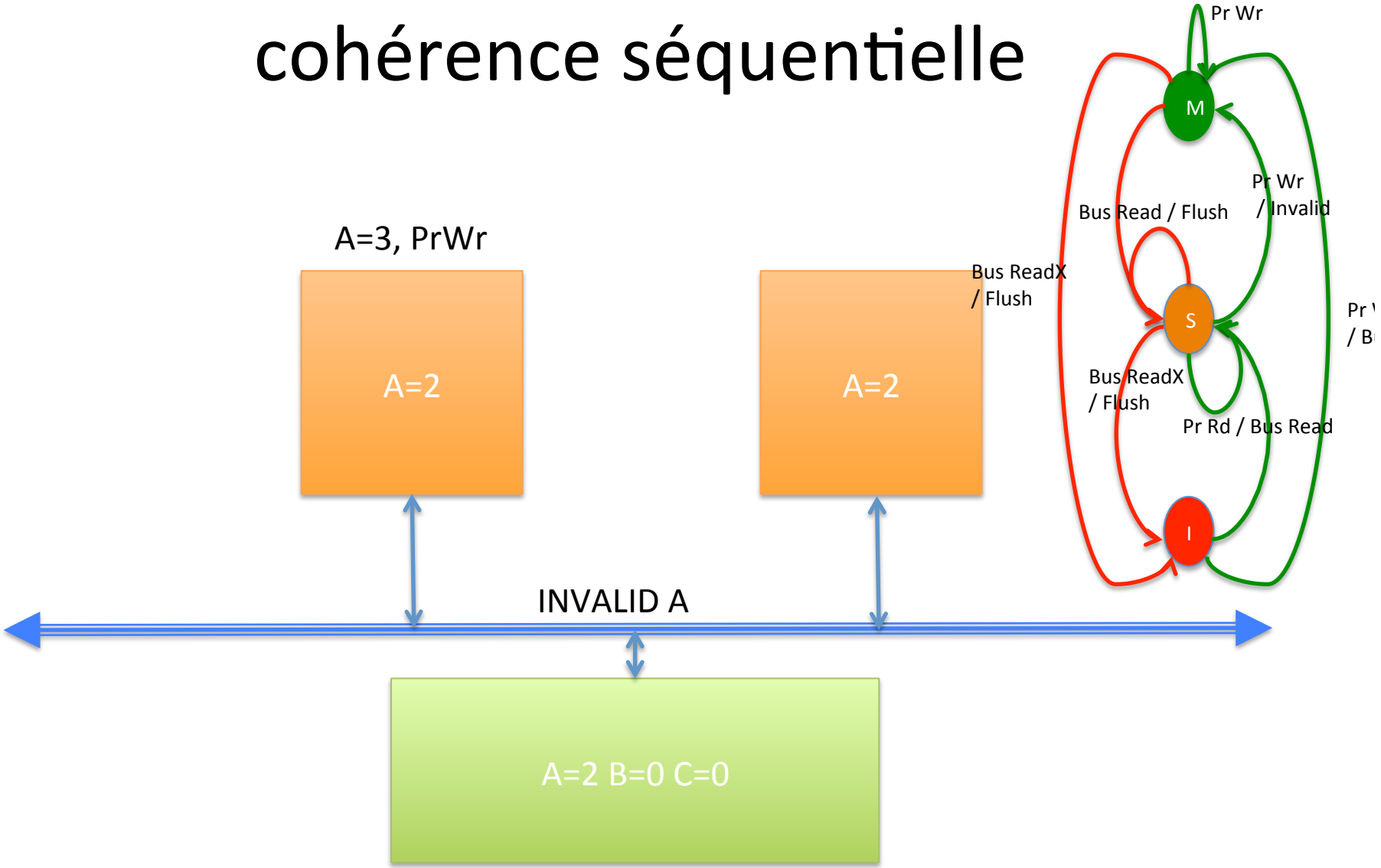
Implémentation de la cohérence séquentielle



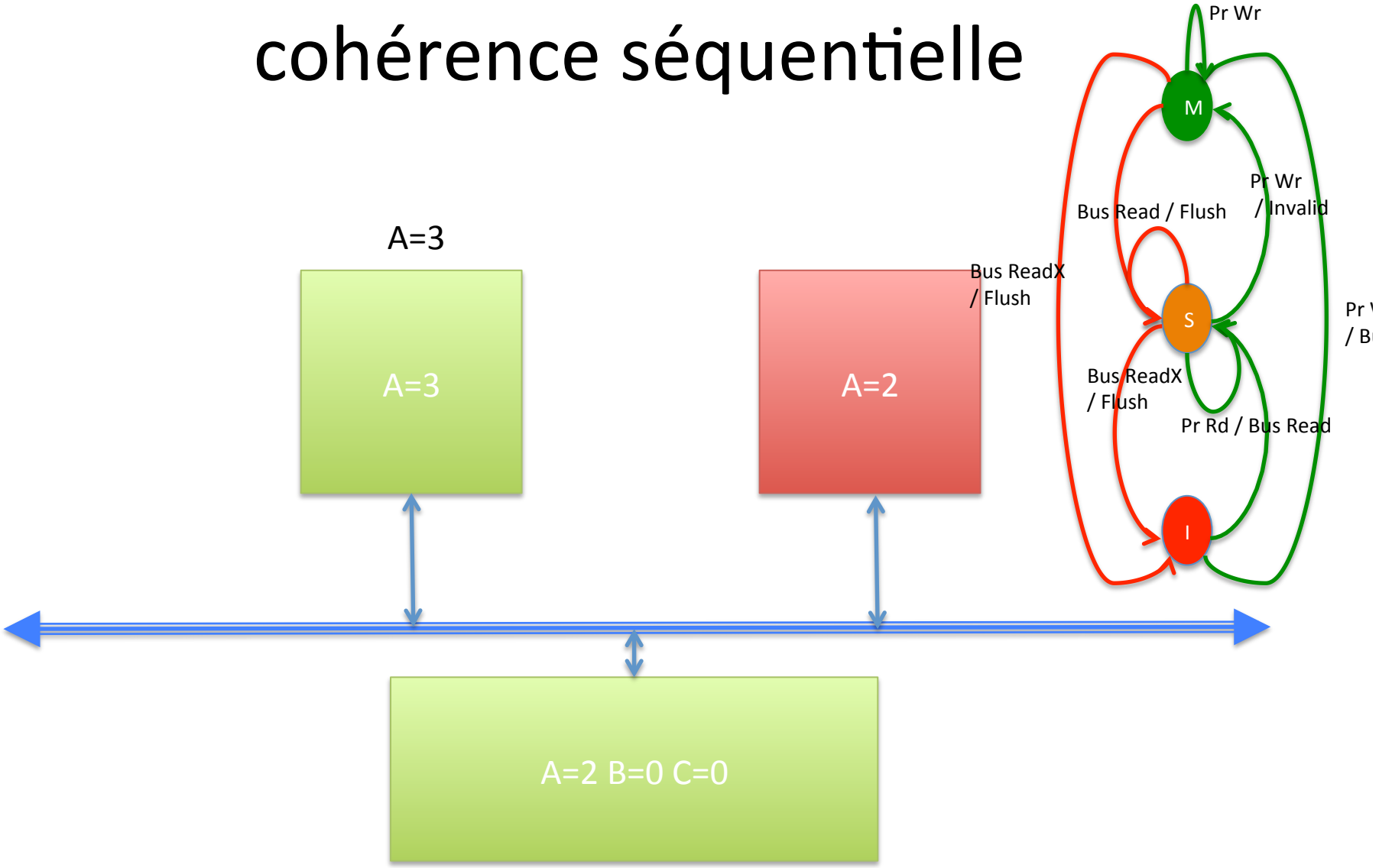
Implémentation de la cohérence séquentielle



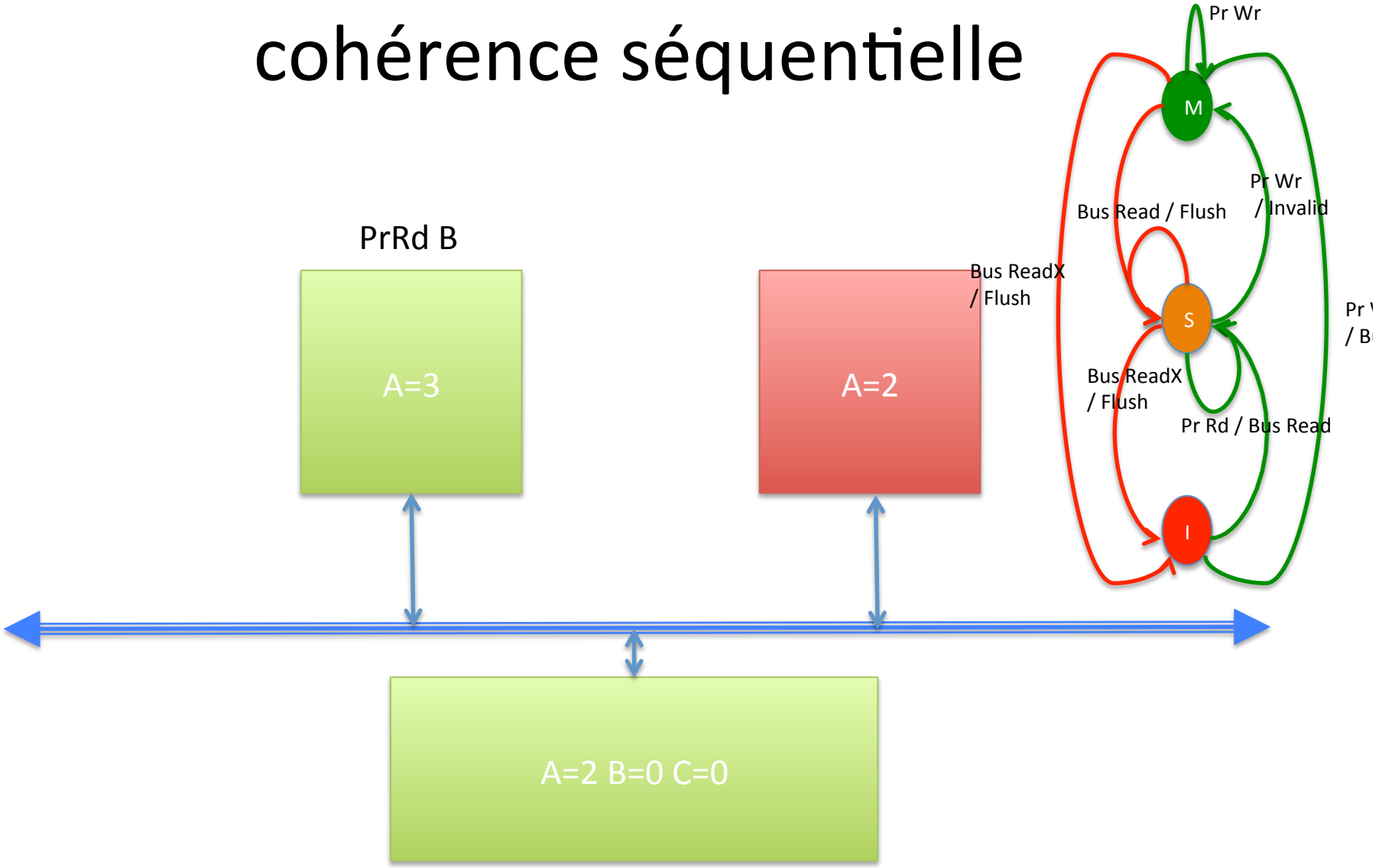
Implémentation de la cohérence séquentielle



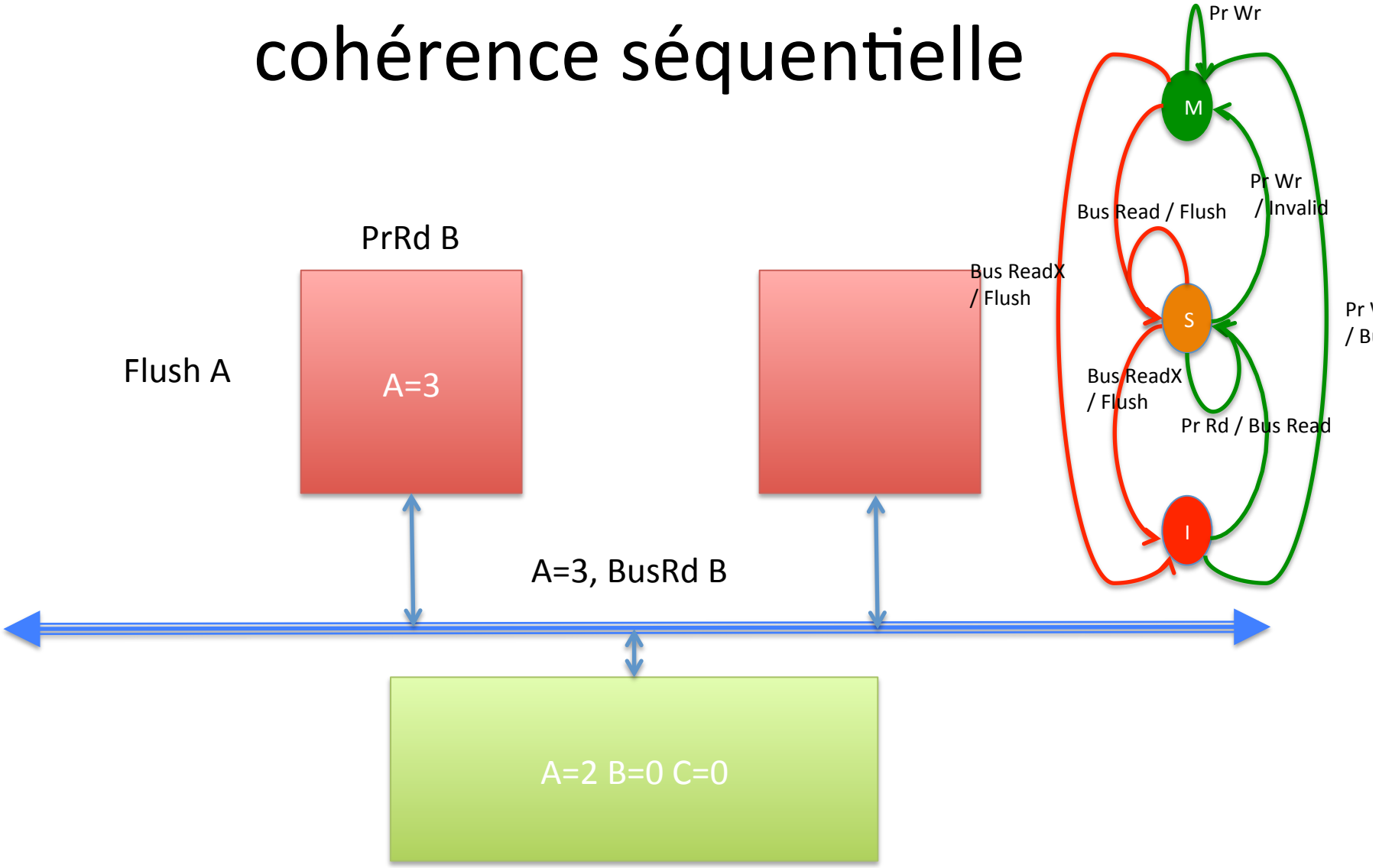
Implémentation de la cohérence séquentielle



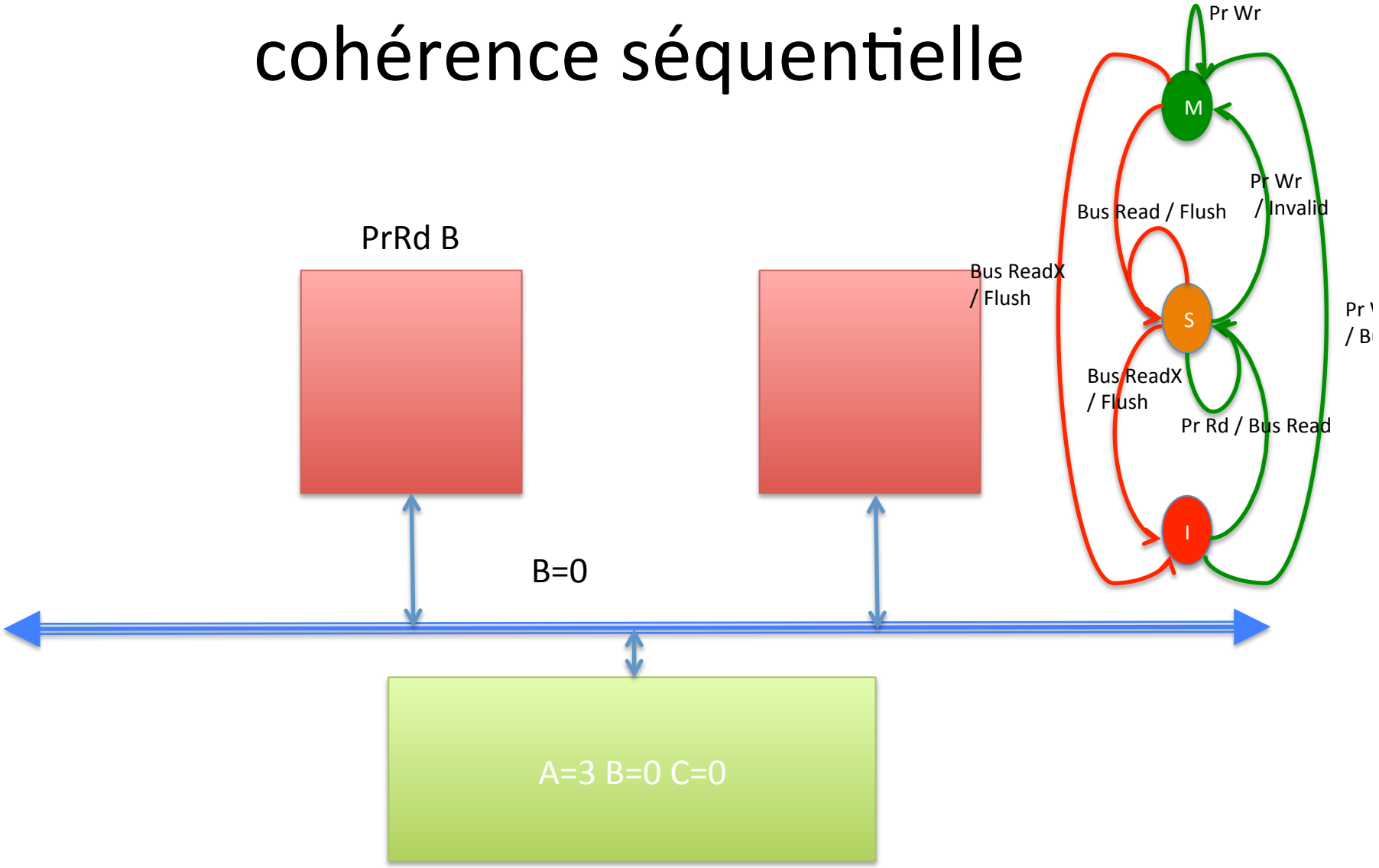
Implémentation de la cohérence séquentielle



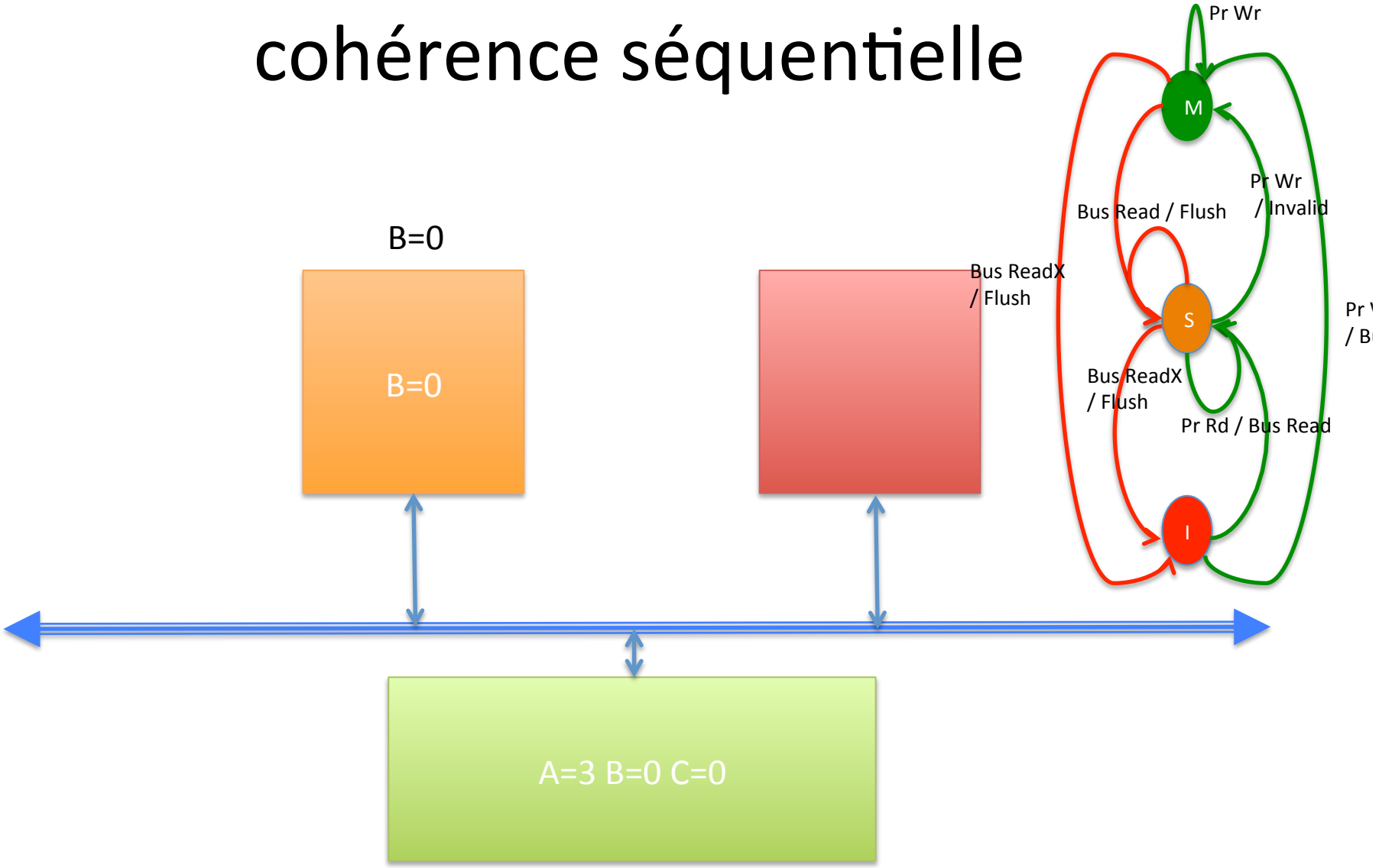
Implémentation de la cohérence séquentielle



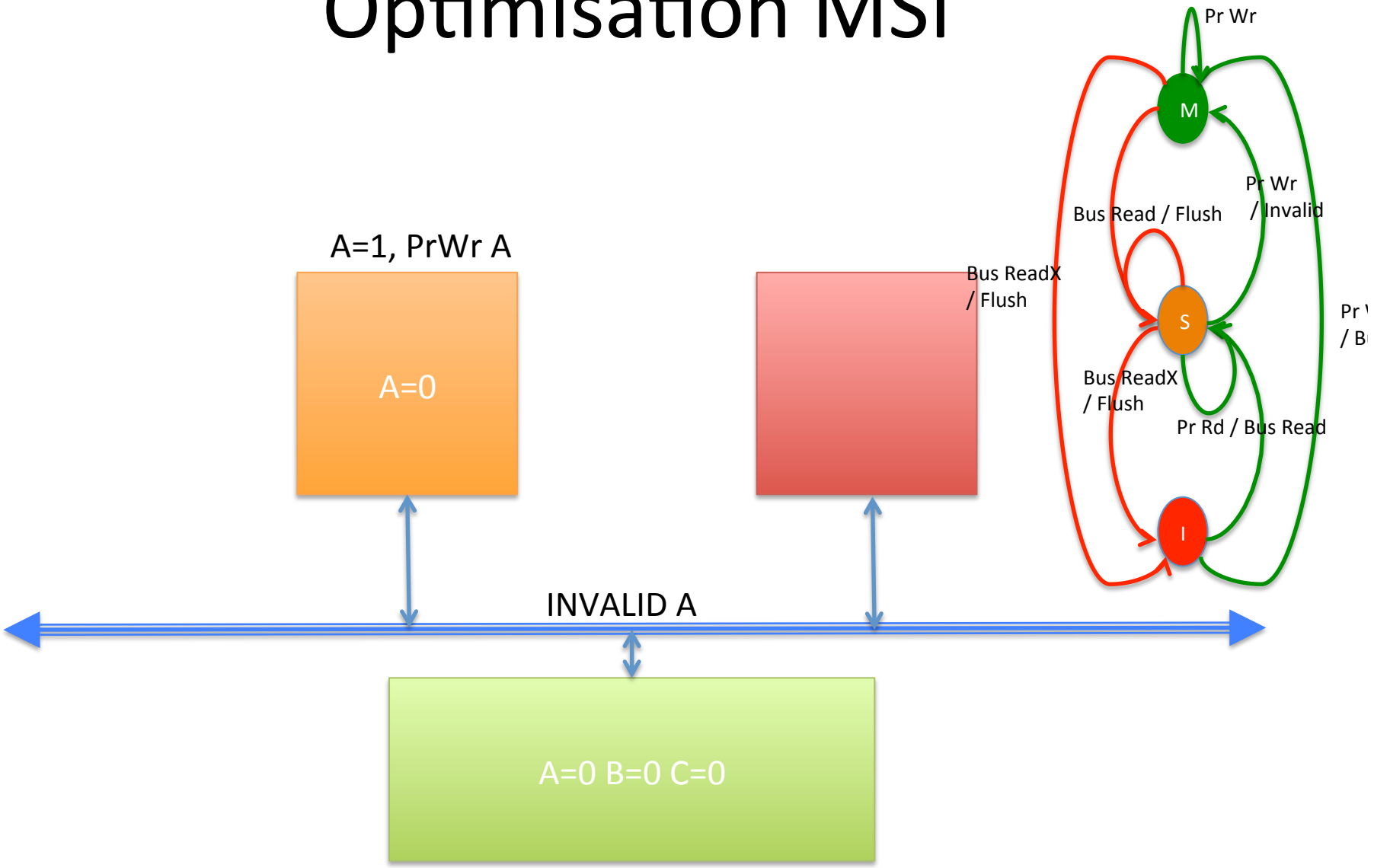
Implémentation de la cohérence séquentielle



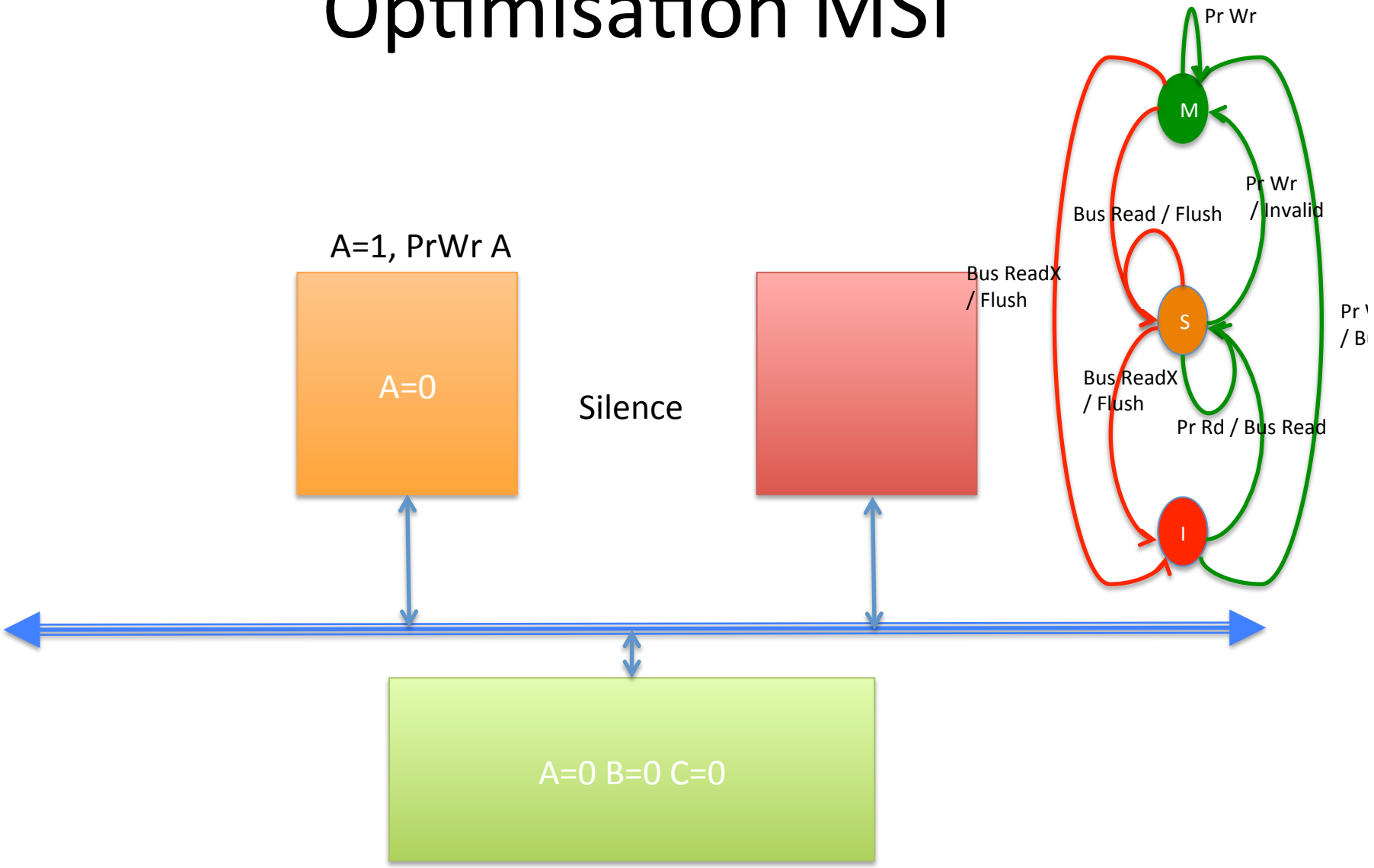
Implémentation de la cohérence séquentielle



Optimisation MSI

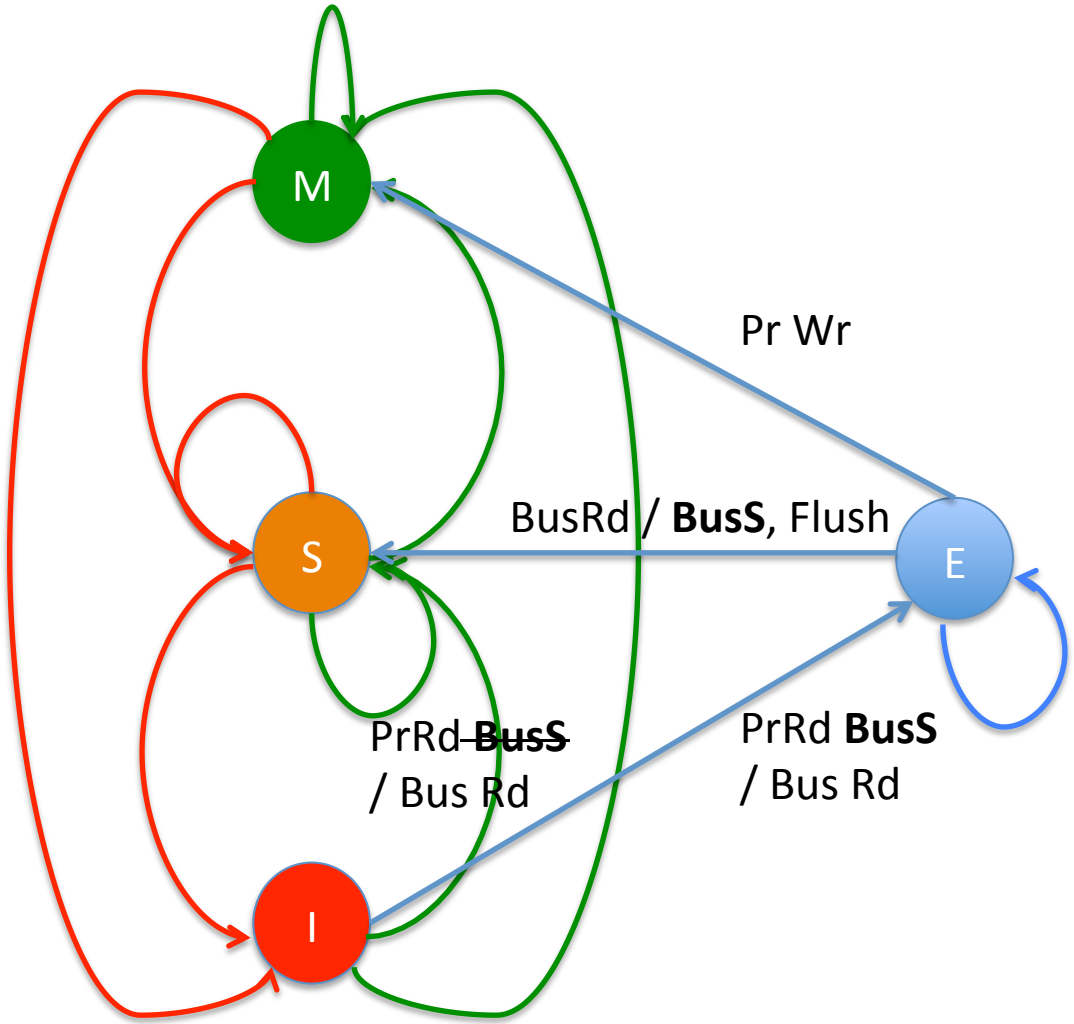


Optimisation MSI



Protocole MESI
Modified Exclusive Shared Invalid

- Transition silencieuse si variable non partagée
  - Nécessite de savoir si la valeur provient d'un cache ou de la mémoire



Instruction atomique

type __sync_fetch_and_add (*type* *ptr, *type* value, ...)

type __sync_val_compare_and_swap (type *ptr, type oldval type newval, ...)

...

- Réalisation
  - Bloquer le bus
  - Passer en *modified* et conserver la donnée jusqu'à l'écriture
  - Passer en *modified* et observer si la donnée est toujours là au moment de l'écriture
- Voir la future norme du C

Le problème du faux partage
False sharing

- 2 variables indépendantes peuvent être sur la même ligne de cache

```
int x,y;
```

Cette ligne de cache peut faire du *ping-pong* entre deux processeurs.

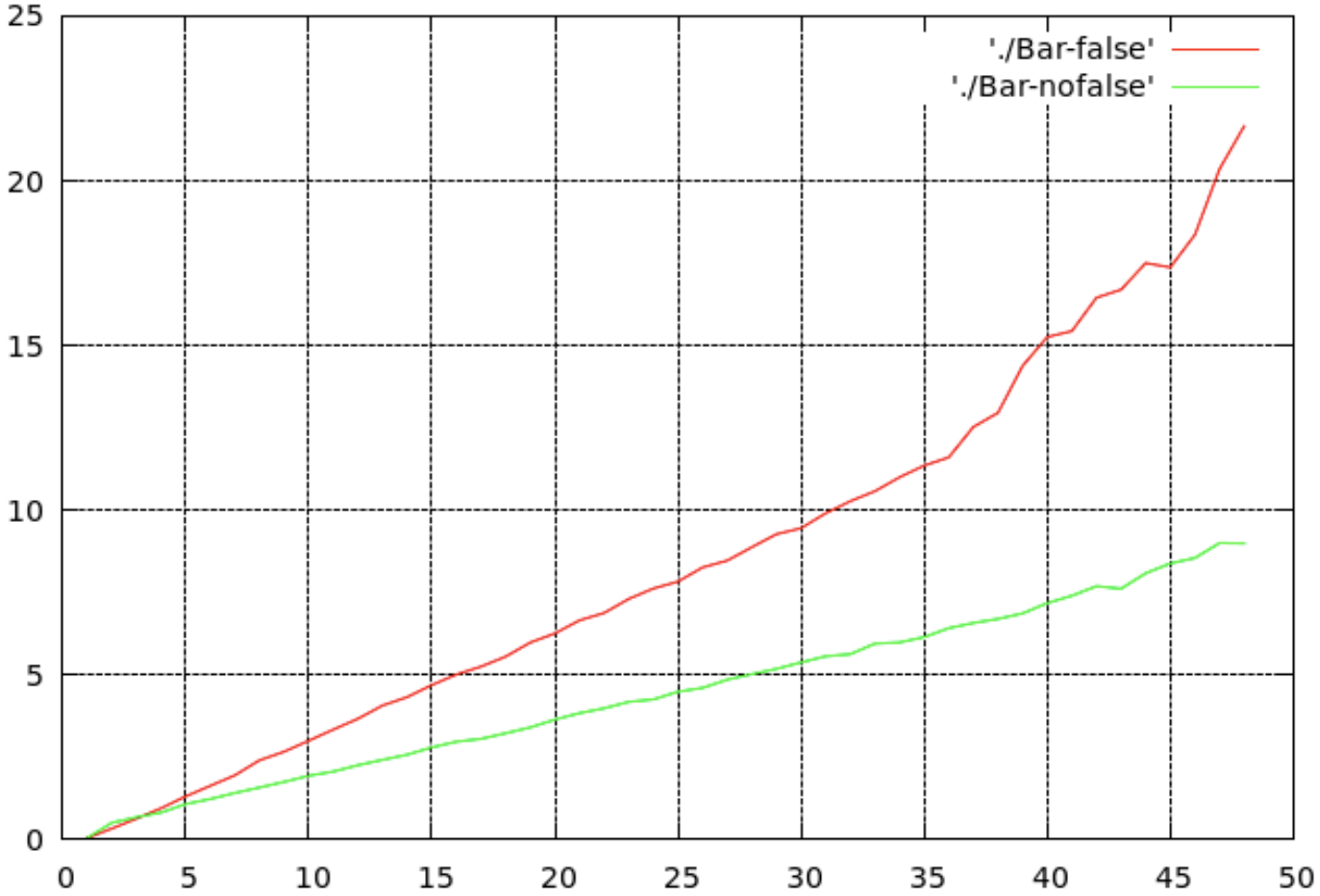
```
int Tab[nb_threads] ;
#pragma parallel for omp
For (i=…)
    Tab[omp_get_thread_num()] = f(i) ;
```

- Solutions
  - Utiliser des variable locale au thread
  - Faire du bourage d'octets (padding)
  - Utiliser des directives d'alignement

```
int x __attribute__((__aligned__(64)));
```

    - Voir la future norme du C

Influence du false sharing sur une barrière



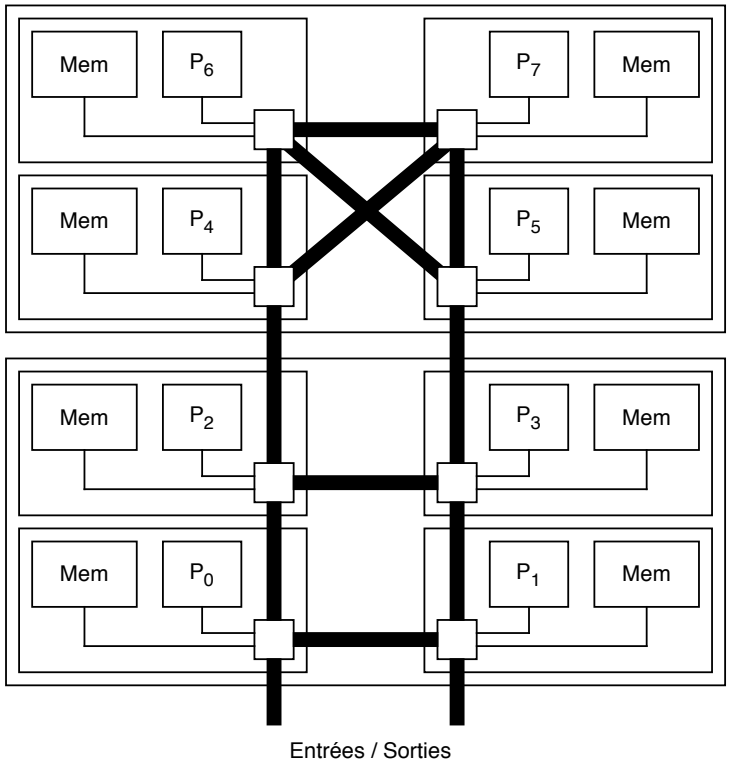
SMP

- Machines faciles à programmer
- Bus : goulet d'étranglement
  - Contention, latence mémoire importante
- Mémoire centralisée
  - Limite le nombre de processeurs

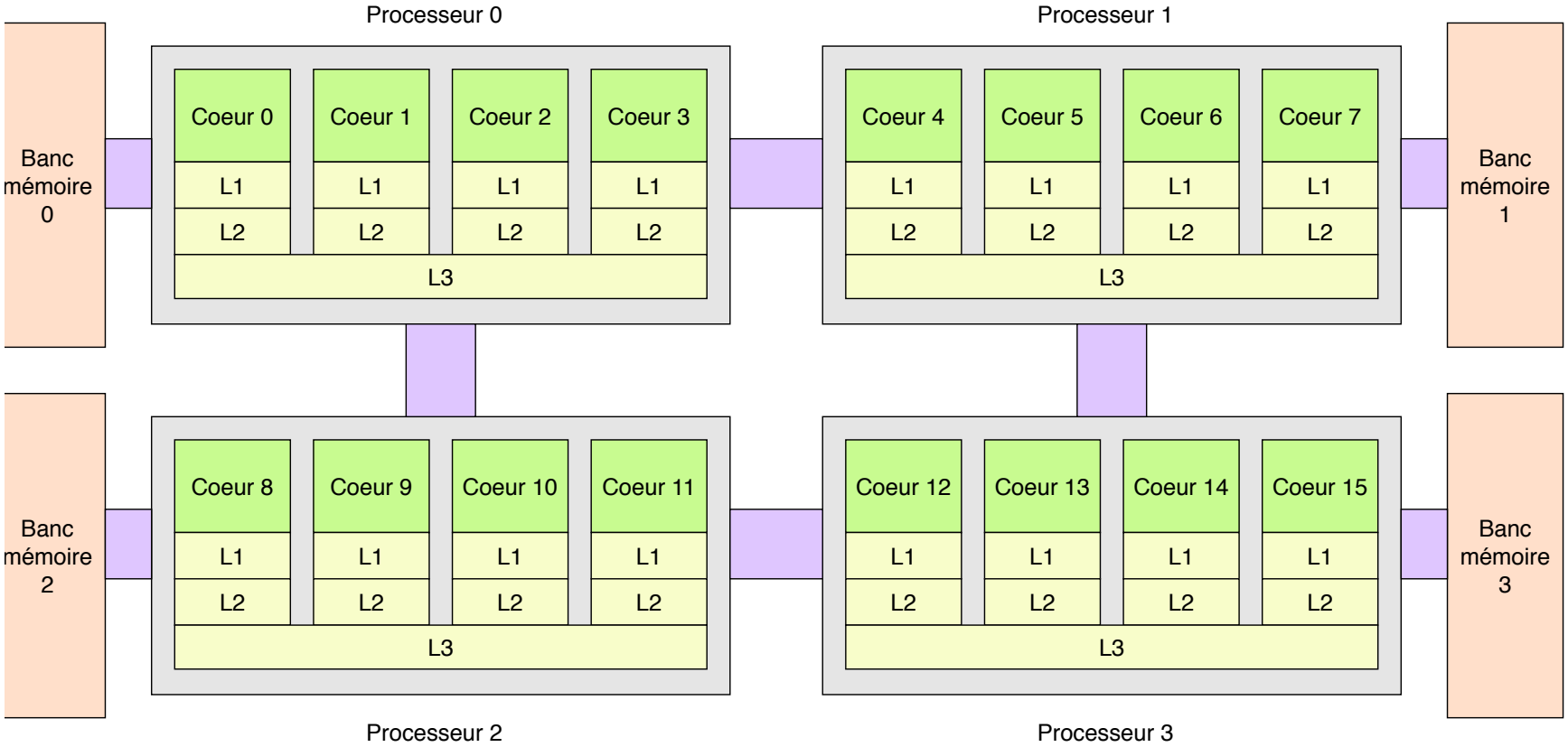
- L'apparition du multicœur condamne cette approche pour le calcul hôte performance

NUMA
non uniform memory access

- Nœud NUMA
  - Briques : processeurs + mémoires
- Réseau de communication inter nœud
  - La latence mémoire dépend de la distance entre le processeur et la mémoire en jeu.
  - Hagrid, facteur numa =
    - 1.1
    - 1.25
    - 1.4



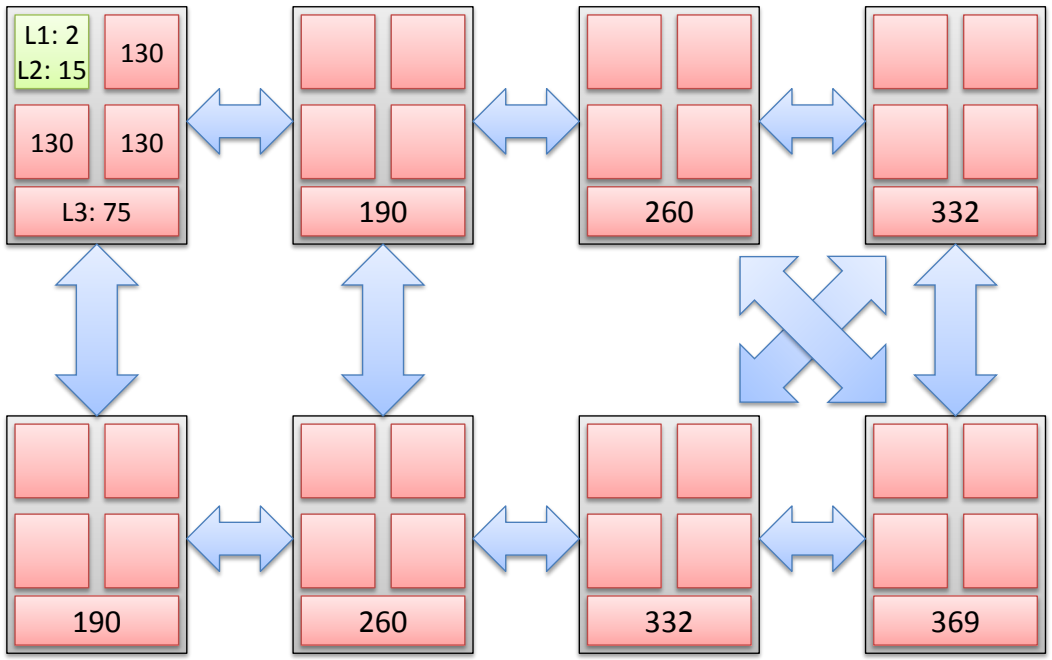
Machine NUMA
opteron 4 x 4



Type d'accès	Accès local	Accès au banc voisin	Accès au banc opposé
Lecture	83 ns	98 ns ($\times$ 1,18)	117 ns ($\times$ 1,41)
Ecriture	142 ns	177 ns ($\times$ 1,25)	208 ns ($\times$ 1,46)

Opteron 8 sockets

Cache access latency



Memory is a bit faster than L3, but it's complicated…

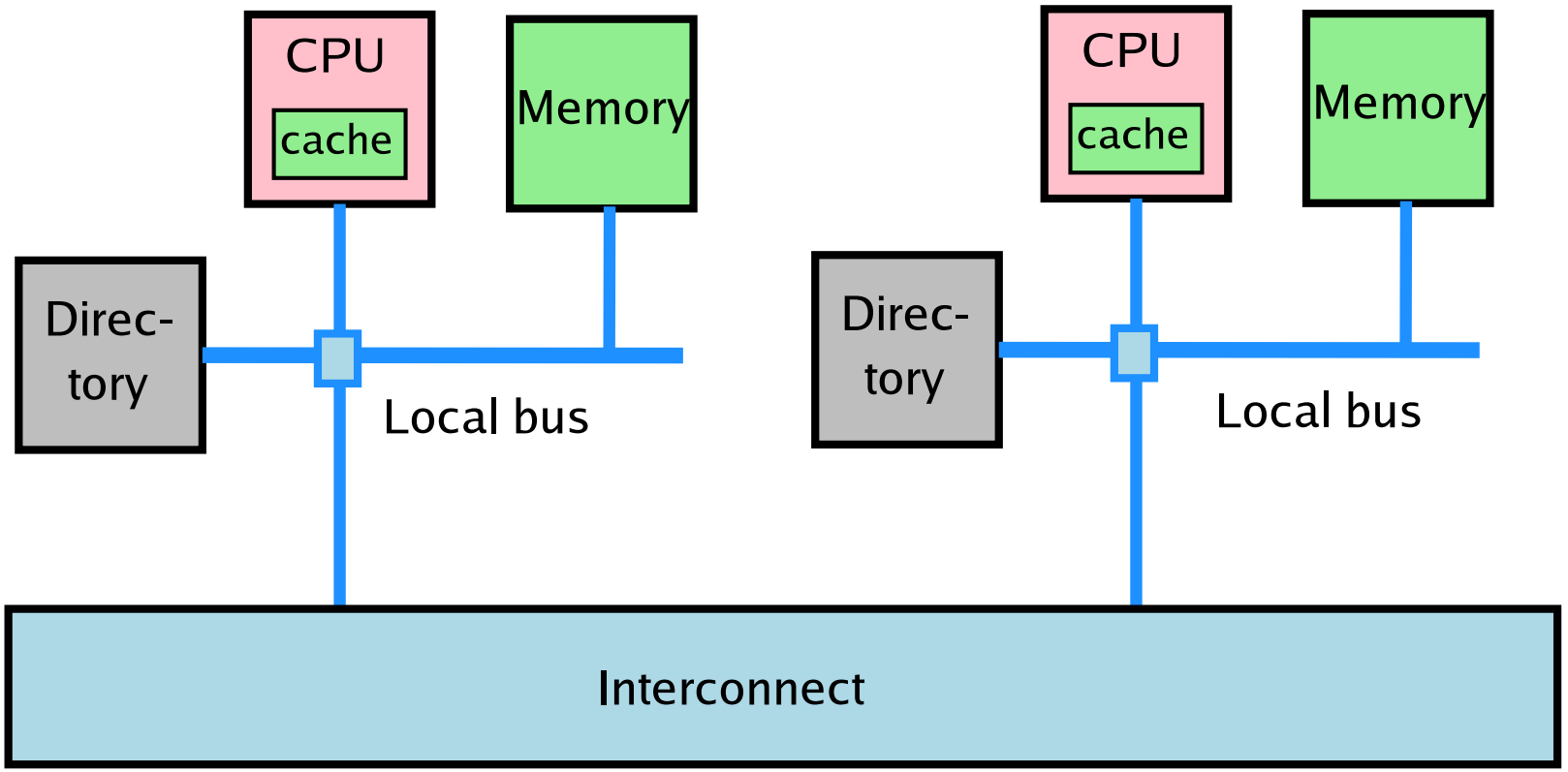
MESI sur NUMA

	Clean/Dirty	Unique?	Can Write?	Can Forward?	Can Silent Transition to	Comments
Modified	Dirty	Yes	Yes	Yes		Must writeback to share or replace
Exclusive	Clean	Yes	Yes	Yes	MSIF	Transitions to M on write
Shared	Clean	No	No	No	I	Does not forward
Invalid	NA	NA	NA	NA		Cannot Read
Forwarding	Clean	Yes	No	Yes	SI	Must invalidate other copies to write

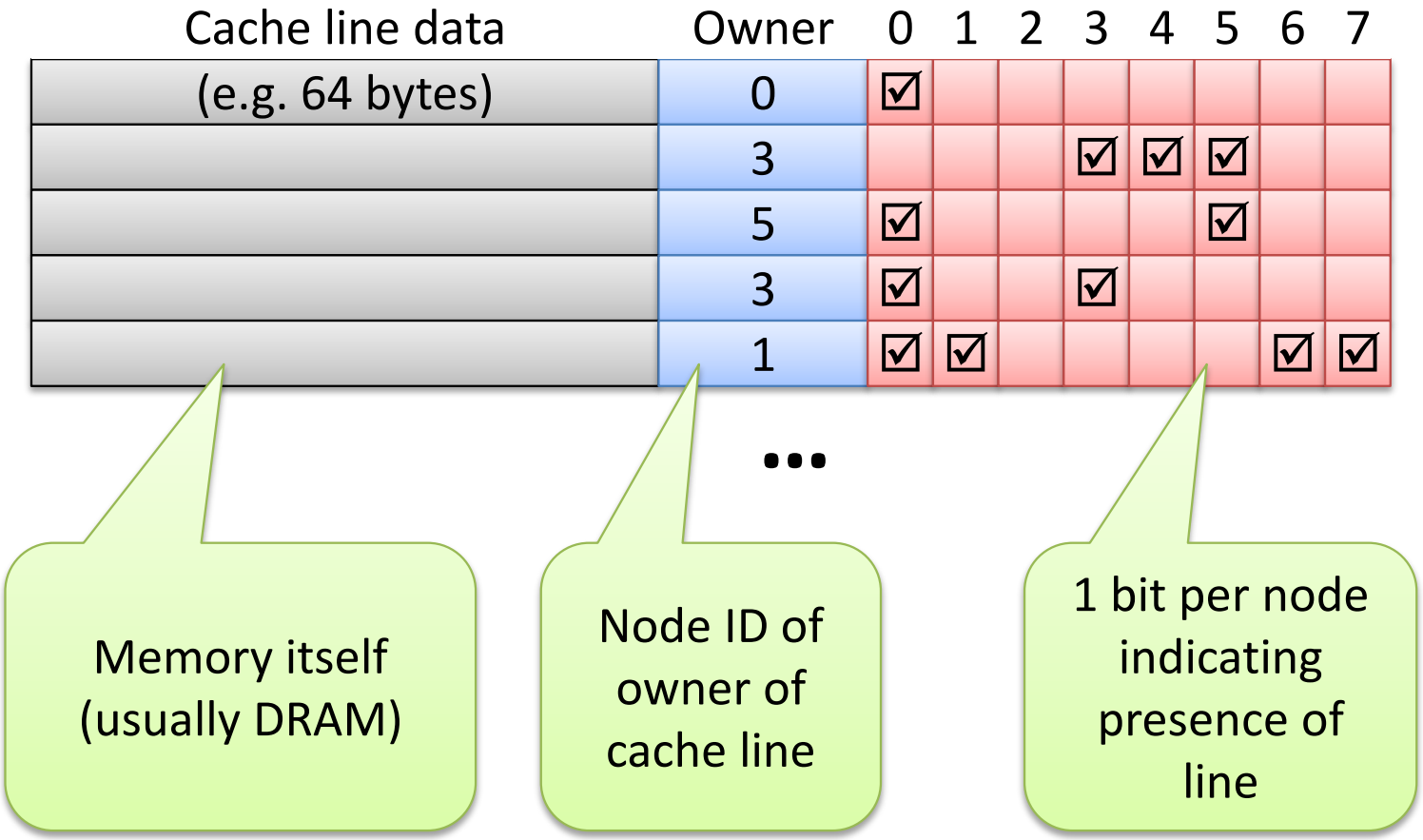
	Clean/Dirty	Unique?	Can Write?	Can Forward?	Can Silent Transition to	Comments
Modified	Dirty	Yes	Yes	Yes	O	Can share without writeback
Owned	Dirty	Yes	Yes	Yes		Must writeback to transition
Exclusive	Clean	Yes	Yes	Yes	MSI	Transitions to M on write
Shared	Either	No	No	No	I	Shared can be dirty or clean
Invalid	NA	NA	NA	NA		Cannot Read

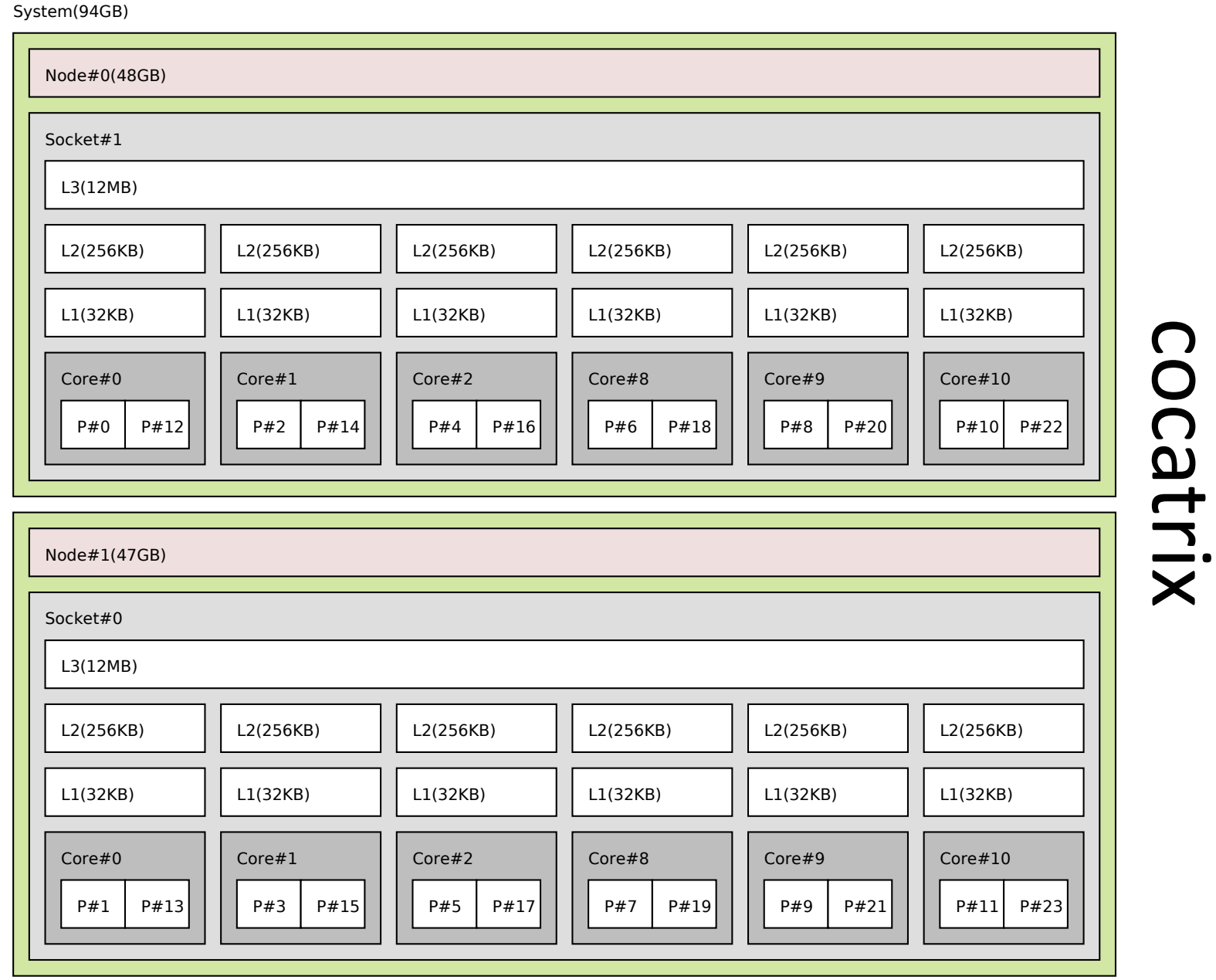
	Clean/Dirty	Unique?	Can Write?	Can Forward?	Can Silent Transition to	Comments
Modified	Dirty	Yes	Yes	Yes		Must writeback to share or replace
Exclusive	Clean	Yes	Yes	Yes	MSI	Transitions to M on write
Shared	Clean	No	No	Yes	I	Shared implies clean, can forward
Invalid	NA	NA	NA	NA		Cannot Read

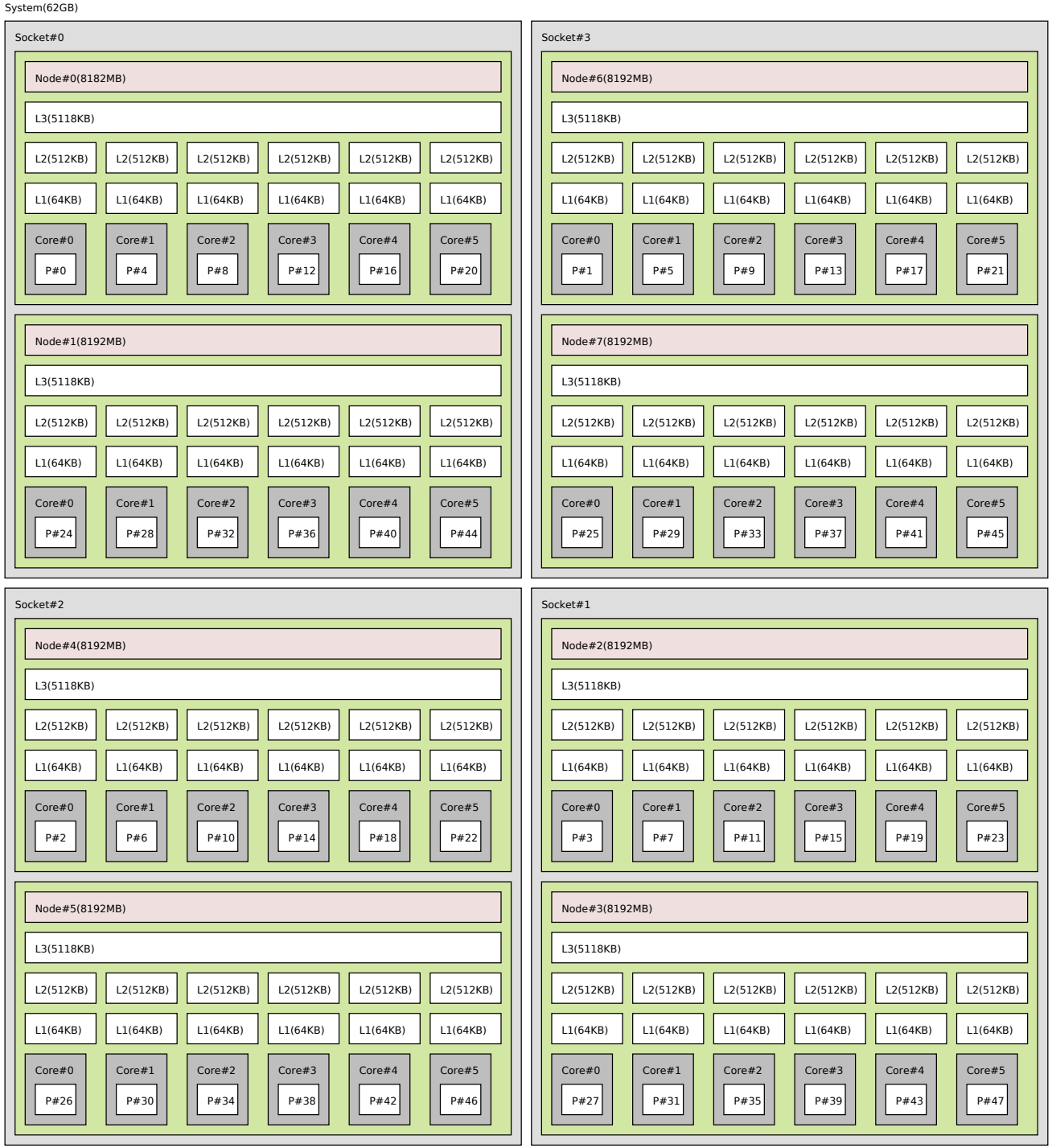
Répertoire


CPU
cache
Memory
Direc-
tory
Local bus
CPU
cache
Memory
Direc-
tory
Local bus
Interconnect

Répertoire


Cache line data
Owner
0 1 2 3 4 5 6 7
(e.g. 64 bytes)
0
3
5
3
1
...
Memory itself (usually DRAM)
Node ID of owner of cache line
1 bit per node indicating presence of line


System(94GB)
Node#0(48GB)
Socket#1
L3(12MB)
L2(256KB)
L2(256KB)
L2(256KB)
L2(256KB)
L2(256KB)
L2(256KB)
L1(32KB)
L1(32KB)
L1(32KB)
L1(32KB)
L1(32KB)
L1(32KB)
Core#0
P#0
P#12
Core#1
P#2
P#14
Core#2
P#4
P#16
Core#8
P#6
P#18
Core#9
P#8
P#20
Core#10
P#10
P#22
Node#1(47GB)
Socket#0
L3(12MB)
L2(256KB)
L2(256KB)
L2(256KB)
L2(256KB)
L2(256KB)
L2(256KB)
L1(32KB)
L1(32KB)
L1(32KB)
L1(32KB)
L1(32KB)
L1(32KB)
Core#0
P#1
P#13
Core#1
P#3
P#15
Core#2
P#5
P#17
Core#8
P#7
P#19
Core#9
P#9
P#21
Core#10
P#11
P#23
cocatrix


System(62GB)
Socket#0
Node#0(8182MB)
L3(5118KB)
L2(512KB) L2(512KB) L2(512KB) L2(512KB) L2(512KB) L2(512KB)
L1(64KB) L1(64KB) L1(64KB) L1(64KB) L1(64KB) L1(64KB)
Core#0 P#0
Core#1 P#4
Core#2 P#8
Core#3 P#12
Core#4 P#16
Core#5 P#20
Node#1(8192MB)
L3(5118KB)
L2(512KB) L2(512KB) L2(512KB) L2(512KB) L2(512KB) L2(512KB)
L1(64KB) L1(64KB) L1(64KB) L1(64KB) L1(64KB) L1(64KB)
Core#0 P#24
Core#1 P#28
Core#2 P#32
Core#3 P#36
Core#4 P#40
Core#5 P#44
Socket#3
Node#6(8192MB)
L3(5118KB)
L2(512KB) L2(512KB) L2(512KB) L2(512KB) L2(512KB) L2(512KB)
L1(64KB) L1(64KB) L1(64KB) L1(64KB) L1(64KB) L1(64KB)
Core#0 P#1
Core#1 P#5
Core#2 P#9
Core#3 P#13
Core#4 P#17
Core#5 P#21
Node#7(8192MB)
L3(5118KB)
L2(512KB) L2(512KB) L2(512KB) L2(512KB) L2(512KB) L2(512KB)
L1(64KB) L1(64KB) L1(64KB) L1(64KB) L1(64KB) L1(64KB)
Core#0 P#25
Core#1 P#29
Core#2 P#33
Core#3 P#37
Core#4 P#41
Core#5 P#45
Socket#2
Node#4(8192MB)
L3(5118KB)
L2(512KB) L2(512KB) L2(512KB) L2(512KB) L2(512KB) L2(512KB)
L1(64KB) L1(64KB) L1(64KB) L1(64KB) L1(64KB) L1(64KB)
Core#0 P#2
Core#1 P#6
Core#2 P#10
Core#3 P#14
Core#4 P#18
Core#5 P#22
Node#5(8192MB)
L3(5118KB)
L2(512KB) L2(512KB) L2(512KB) L2(512KB) L2(512KB) L2(512KB)
L1(64KB) L1(64KB) L1(64KB) L1(64KB) L1(64KB) L1(64KB)
Core#0 P#26
Core#1 P#30
Core#2 P#34
Core#3 P#38
Core#4 P#42
Core#5 P#46
Socket#1
Node#2(8192MB)
L3(5118KB)
L2(512KB) L2(512KB) L2(512KB) L2(512KB) L2(512KB) L2(512KB)
L1(64KB) L1(64KB) L1(64KB) L1(64KB) L1(64KB) L1(64KB)
Core#0 P#3
Core#1 P#7
Core#2 P#11
Core#3 P#15
Core#4 P#19
Core#5 P#23
Node#3(8192MB)
L3(5118KB)
L2(512KB) L2(512KB) L2(512KB) L2(512KB) L2(512KB) L2(512KB)
L1(64KB) L1(64KB) L1(64KB) L1(64KB) L1(64KB) L1(64KB)
Core#0 P#27
Core#1 P#31
Core#2 P#35
Core#3 P#39
Core#4 P#43
Core#5 P#47

Bourfouf

Outils pour le placement

Libnuma (linux)

- Placement des threads
  - int sched_setaffinity(pid_t pid, size_t cpusetsize, cpu_set_t *mask);
- Politiques placement mémoire
  - Local, distant, interleave
- Numactl en ligne de commande

- Allocation first touch
  - Allocation paresseuse des pages
  - Placement réalisé à la première utilisation
    - Suivant la politique définie
    - Par défaut la page est placée sur le nœud numa du cœur ayant causé l'allocation
  - Nécessite la connaissance de l'architecture pour faire des placements optimaux

Stratégie Placement thread / mémoire

- Stratégie de placement thread / mémoire
  - Déterminer les dimensions où l'application à le comportement le plus régulier possible
  - Découper le travail en tâches équilibrées et les affecter de façon statique aux threads
  - Réaliser l'allocation physique des pages
    - Boucle d'initialisation à blanc réalisé par chaque thread
  - Lancement de l'application

- Commentaires :
  - Stratégie bien comprise par les programmeurs OpenMP un peu expérimentés
  - Ne s'applique pas aux problèmes irréguliers
    - « move on next touch »
  - Ne tient pas compte de la charge mémoire des nœuds
    - Ni de la consommation de la bande passante

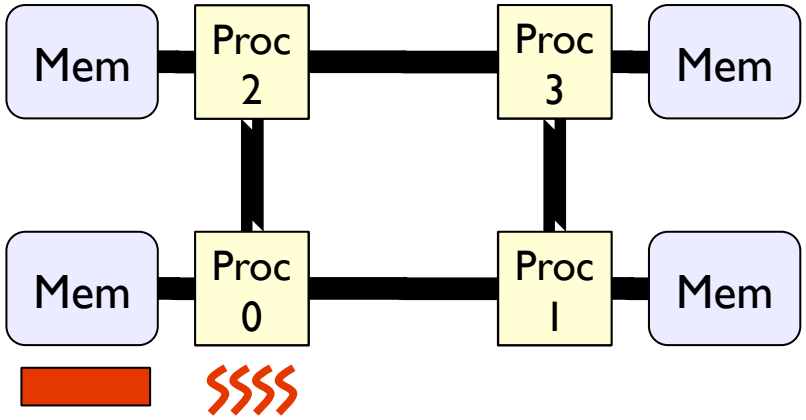
TP1 sur Boursouf
initialisation séquentielle

	1	2	4	8	16	32
addseq:	1095	1152	1053	1151	1152	1149
addparastat:	1703	965	909	605	654	632
addparadyna:	1698	794	746	549	564	535
sumseq:	237	239	239	239	239	238
sumparastat:	5167	43172	43798	23559	16465	14974
sumparapart:	340	275	275	195	202	207
addparastat:	0,64	1,19	1,16	1,90	1,76	1,82
addparadyna:	0,64	1,45	1,41	2,09	2,04	2,15
sumparastat:	0,05	0,01	0,01	0,01	0,01	0,02
sumparapart:	0,70	0,87	0,87	1,22	1,18	1,15

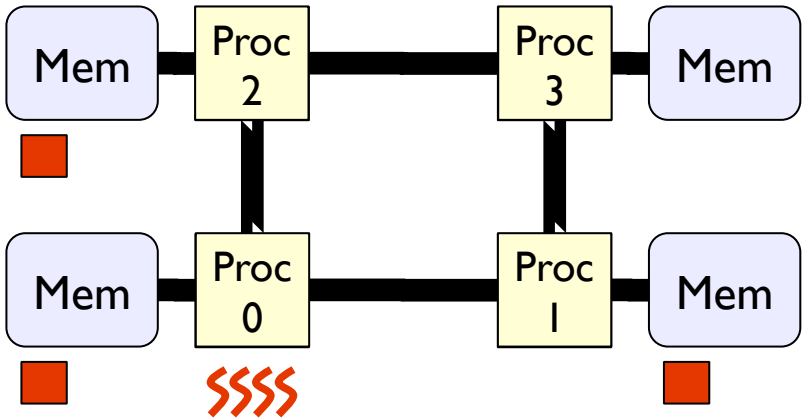
TP1 sur Boursouf
initialisation parallèle

	1	2	4	8	16	32
addseq:	1616	1794	1897	1889	1884	1878
addparastat:	956	510	260	132	87	57
addparadyna:	960	845	615	505	499	425
sumseq:	379	459	502	481	480	477
sumparastat:	4430	7828	37887	21038	15564	13635
sumparapart:	222	112	56	28	17	23
addparastat:	1,14	2,26	4,05	8,72	13,24	20,16
addparadyna:	1,14	1,36	1,71	2,28	2,31	2,70
sumparastat:	0,05	0,03	0,01	0,01	0,02	0,02
sumparapart:	1,07	2,13	4,27	8,53	14,05	10,36

Contention mémoire



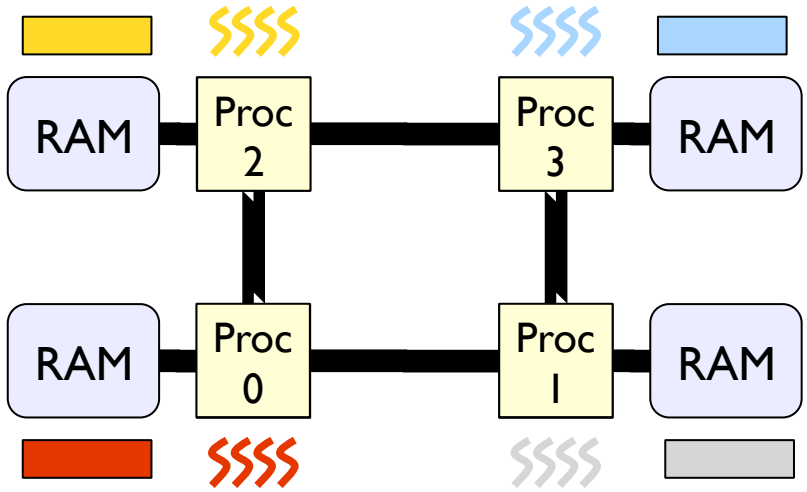
Local: 3621 Mo/s



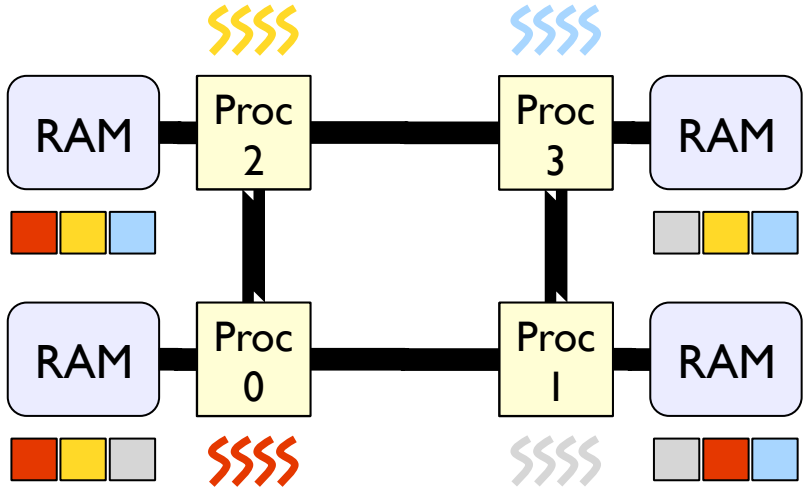
Local + voisins: 3940 Mo/s

- Application synthétique
  - Deux placements mémoire différents
    - Allouer localement
    - Répartir les pages mémoire sur les nœuds voisins
- Résultats
  - Sur une machine non chargée, la solution répartie est la meilleure

Le problème de la contention mémoire



Local: 4 x 3635 Mo/s



Local + voisins: 4 x 2257 Mo/s

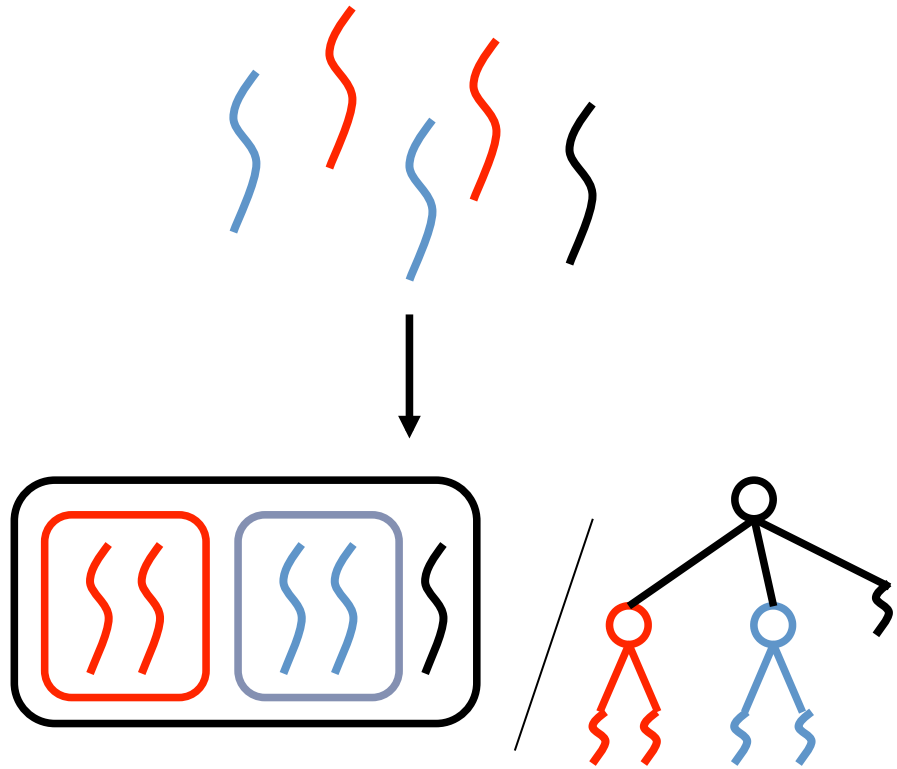
- Application synthétique
  - Deux placements mémoire différents
    - Allouer localement
    - Répartir les pages mémoire sur les nœuds voisins
- Résultats
  - Sur une machine chargée, la solution locale est la meilleure

Conclusion

- Le tournevis
  - Solution universelle et performante
  - Mais coûteuse en manpower et en matière grise
    - Cercle de plus en plus restreint d'experts
- Portabilité des performances
  - Expression structuré du parallélisme
    - Parallélisme imbriqué
  - Modélisation du matériel
    - Organisation hiérachique de la machine
  - Projection dynamique du paraléllisme sur le matériel
    - Lors de l'exécution

Des bulles pour exprimer les affinités

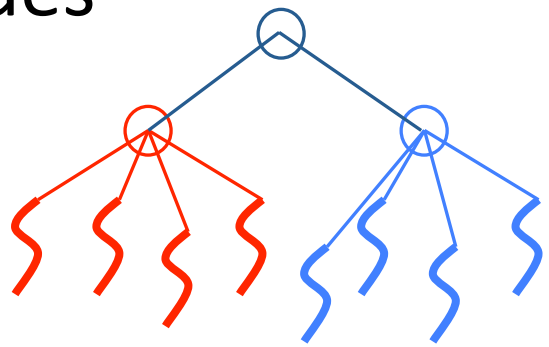
- Regrouper les threads en relation
  - Partage de données
  - Synchronisations
  - Interactions avec les E/S
- La plateforme BubbleSched
  - Fournit des primitives de haut niveau pour créer/ déplacer/éclater des bulles
  - Possibilité d'imbriquer des bulles



http://runtime.bordeaux.inria.fr/bubblesched/

Ordonnancement de bulles sur architectures hiérarchiques

- Architecture modélisée à l'aide de listes d'ordonnancement
- Structure du parallélisme capturée dans des bulles
- Ordonnancer = projeter un arbre dynamique de threads et de bulles sur un arbre de listes d'ordonnancement

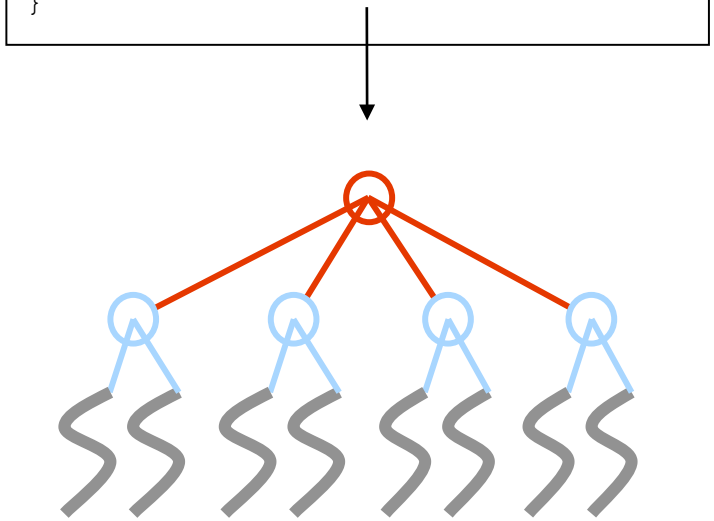


ForestGOMP: un support exécutif OpenMP conçu pour les architectures hiérarchiques

- Extension de GNU OpenMP
- Les sections parallèles génèrent des bulles
- Gestion efficace du parallélisme imbriqué
  - In nested we trust !
- Synchronisations adaptées à l'architecture

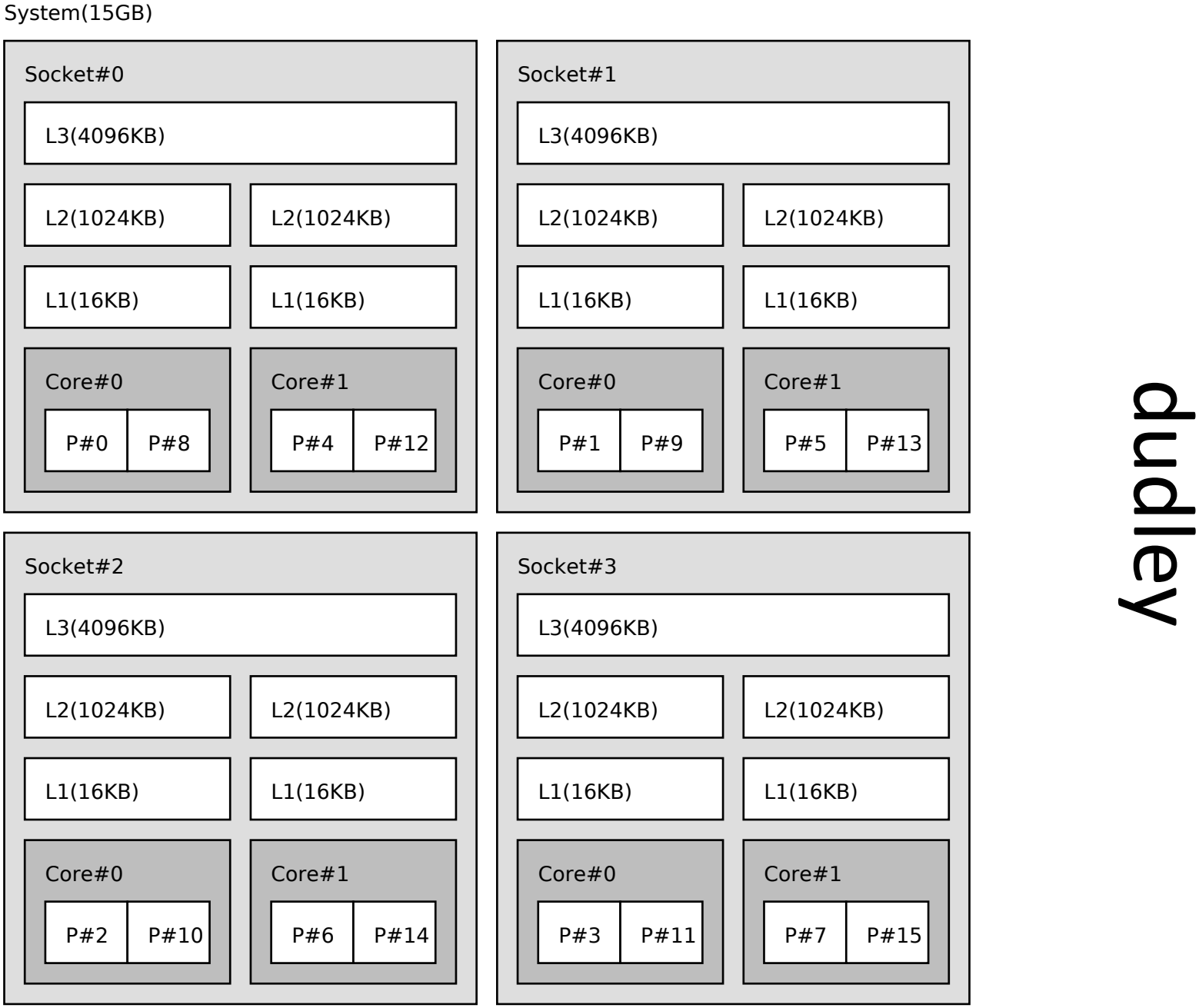
```
void job()
{
   ...

#pragma omp parallel for
  for (int i=0; i<MAX; i++)
    {
      ...
#pragma omp parallel for
num_threads (2)
      for (int k=0; k<MAX; k++)
        ...
    }
}
```

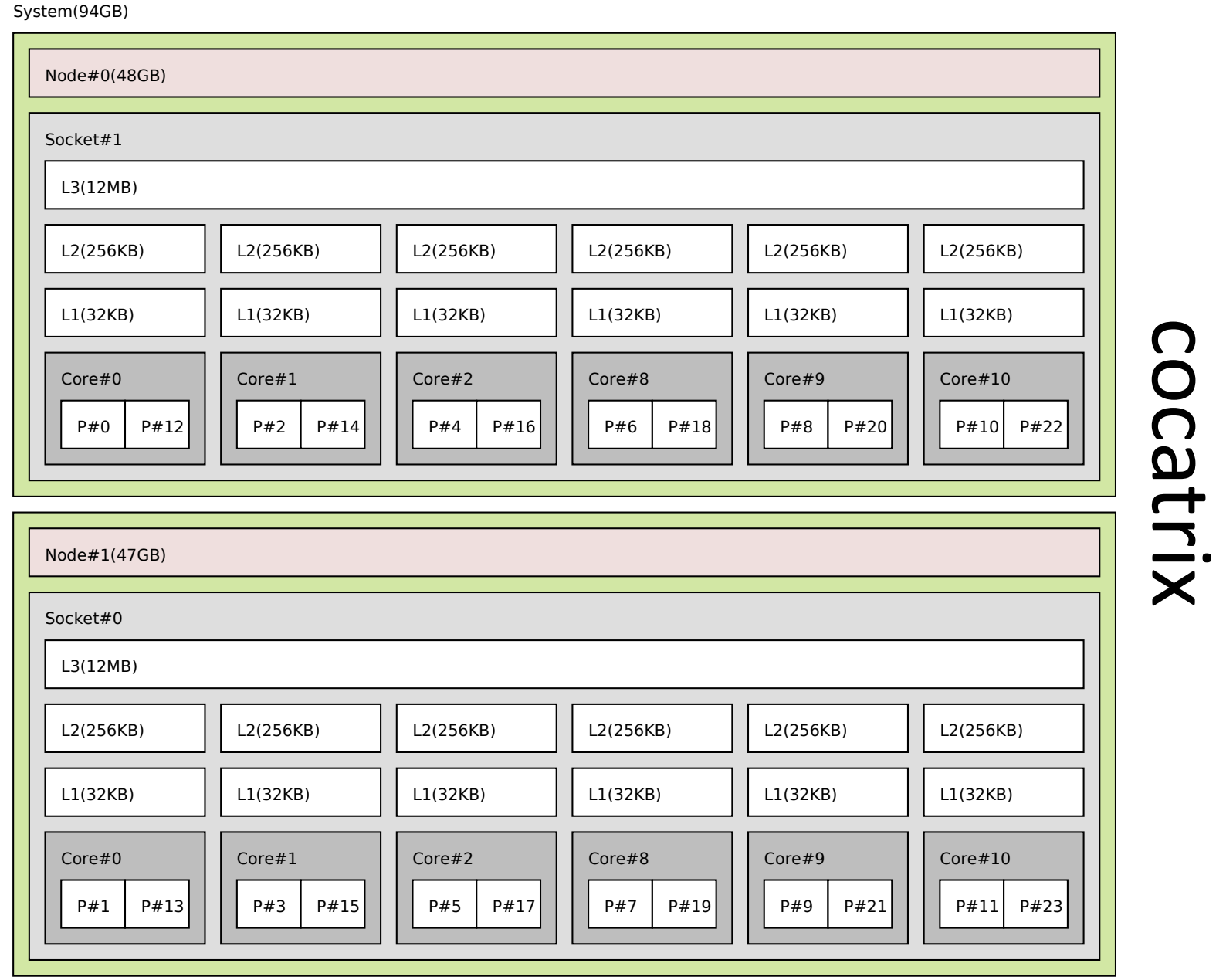


Conclusion Archi

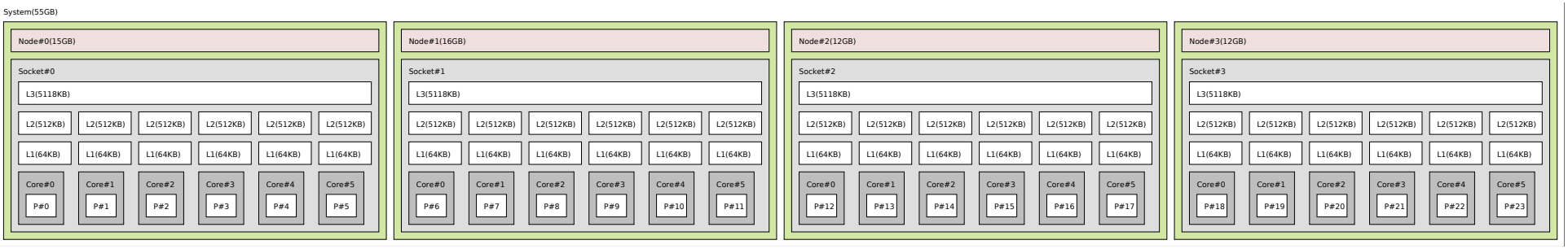
- ILP :
  - trop complexe, énergivore
    - EPIC (itanium) : impasse, les compilateurs ne sont pas assez puissant.
  - Retour en arrière avec le multicore
- Multicore
  - Inévitable
  - Pénalisé par la cohérence de cache séquentiel
- Quel avenir pour l'approche mémoire commune ?


System(15GB)
Socket#0
L3(4096KB)
L2(1024KB)
L2(1024KB)
L1(16KB)
L1(16KB)
Core#0
P#0
P#8
Core#1
P#4
P#12
Socket#1
L3(4096KB)
L2(1024KB)
L2(1024KB)
L1(16KB)
L1(16KB)
Core#0
P#1
P#9
Core#1
P#5
P#13
Socket#2
L3(4096KB)
L2(1024KB)
L2(1024KB)
L1(16KB)
L1(16KB)
Core#0
P#2
P#10
Core#1
P#6
P#14
Socket#3
L3(4096KB)
L2(1024KB)
L2(1024KB)
L1(16KB)
L1(16KB)
Core#0
P#3
P#11
Core#1
P#7
P#15

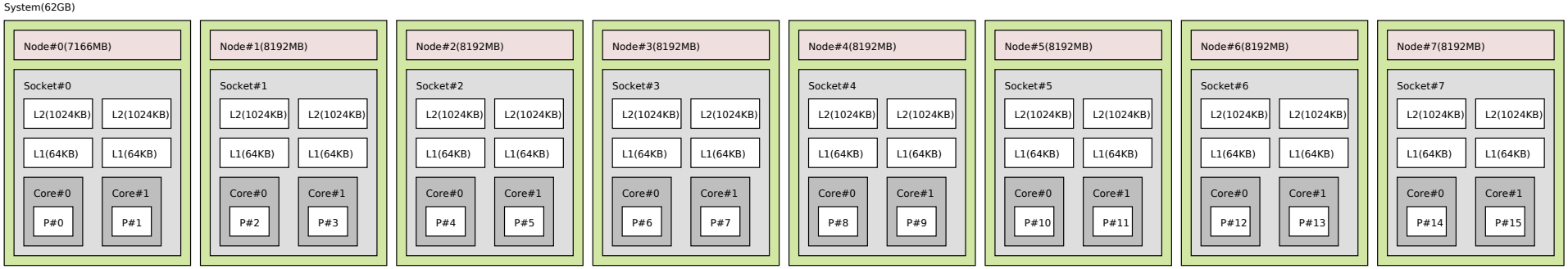
dudley

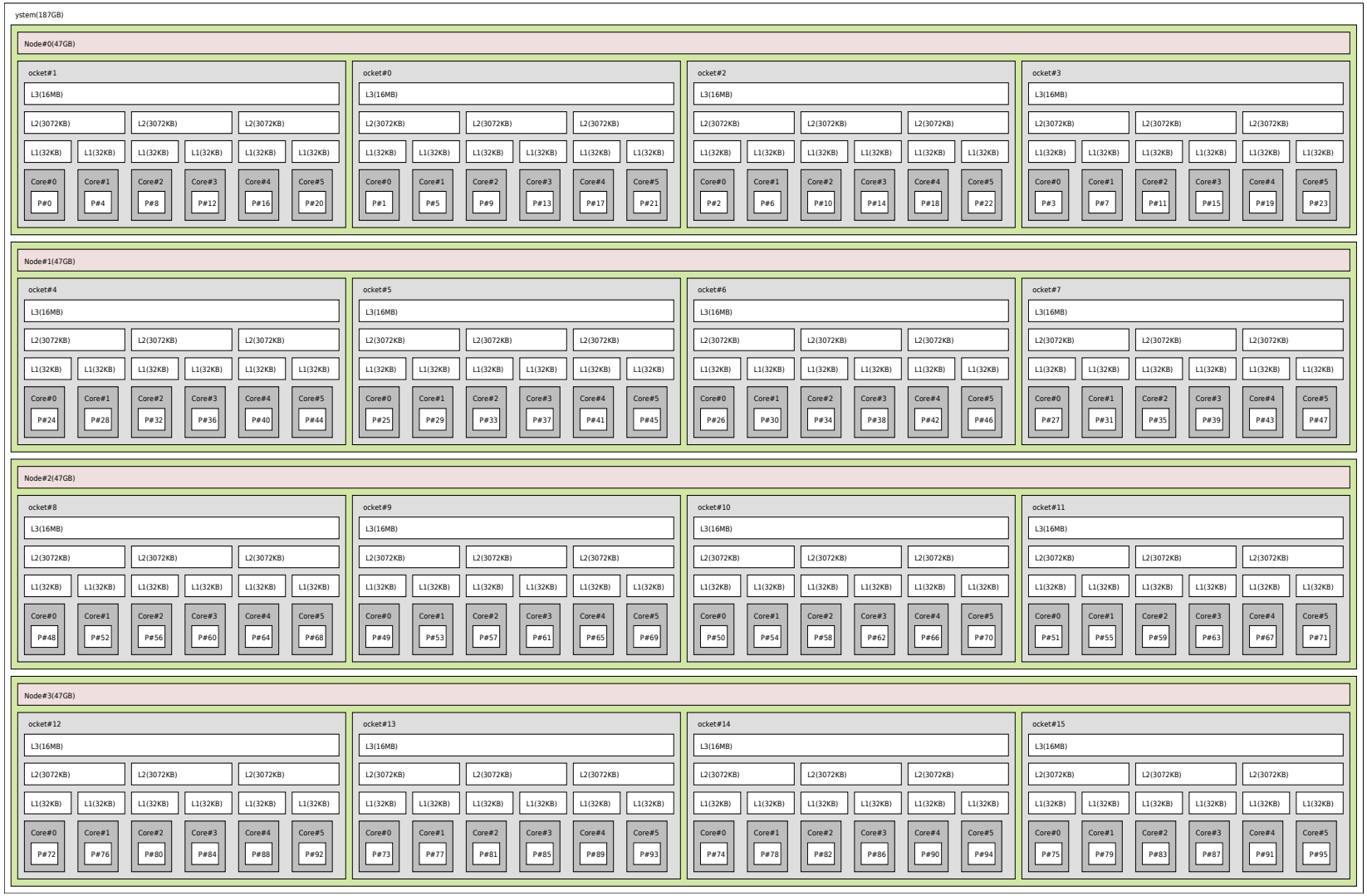

System(94GB)
Node#0(48GB)
Socket#1
L3(12MB)
L2(256KB)
L2(256KB)
L2(256KB)
L2(256KB)
L2(256KB)
L2(256KB)
L1(32KB)
L1(32KB)
L1(32KB)
L1(32KB)
L1(32KB)
L1(32KB)
Core#0
P#0
P#12
Core#1
P#2
P#14
Core#2
P#4
P#16
Core#8
P#6
P#18
Core#9
P#8
P#20
Core#10
P#10
P#22
Node#1(47GB)
Socket#0
L3(12MB)
L2(256KB)
L2(256KB)
L2(256KB)
L2(256KB)
L2(256KB)
L2(256KB)
L1(32KB)
L1(32KB)
L1(32KB)
L1(32KB)
L1(32KB)
L1(32KB)
Core#0
P#1
P#13
Core#1
P#3
P#15
Core#2
P#5
P#17
Core#8
P#7
P#19
Core#9
P#9
P#21
Core#10
P#11
P#23
cocatrix

fridulva



hagrid




ystem(187GB)
Node#0(47GB)
ocket#1
ocket#0
ocket#2
ocket#3
L3(16MB)
L2(3072KB)
L1(32KB)
Core#0 P#0
Core#1 P#4
Core#2 P#8
Core#3 P#12
Core#4 P#16
Core#5 P#20
Core#0 P#1
Core#1 P#5
Core#2 P#9
Core#3 P#13
Core#4 P#17
Core#5 P#21
Core#0 P#2
Core#1 P#6
Core#2 P#10
Core#3 P#14
Core#4 P#18
Core#5 P#22
Core#0 P#3
Core#1 P#7
Core#2 P#11
Core#3 P#15
Core#4 P#19
Core#5 P#23
Node#1(47GB)
ocket#4
ocket#5
ocket#6
ocket#7
Core#0 P#24
Core#1 P#28
Core#2 P#32
Core#3 P#36
Core#4 P#40
Core#5 P#44
Core#0 P#25
Core#1 P#29
Core#2 P#33
Core#3 P#37
Core#4 P#41
Core#5 P#45
Core#0 P#26
Core#1 P#30
Core#2 P#34
Core#3 P#38
Core#4 P#42
Core#5 P#46
Core#0 P#27
Core#1 P#31
Core#2 P#35
Core#3 P#39
Core#4 P#43
Core#5 P#47
Node#2(47GB)
ocket#8
ocket#9
ocket#10
ocket#11
Core#0 P#48
Core#1 P#52
Core#2 P#56
Core#3 P#60
Core#4 P#64
Core#5 P#68
Core#0 P#49
Core#1 P#53
Core#2 P#57
Core#3 P#61
Core#4 P#65
Core#5 P#69
Core#0 P#50
Core#1 P#54
Core#2 P#58
Core#3 P#62
Core#4 P#66
Core#5 P#70
Core#0 P#51
Core#1 P#55
Core#2 P#59
Core#3 P#63
Core#4 P#67
Core#5 P#71
Node#3(47GB)
ocket#12
ocket#13
ocket#14
ocket#15
Core#0 P#72
Core#1 P#76
Core#2 P#80
Core#3 P#84
Core#4 P#88
Core#5 P#92
Core#0 P#73
Core#1 P#77
Core#2 P#81
Core#3 P#85
Core#4 P#89
Core#5 P#93
Core#0 P#74
Core#1 P#78
Core#2 P#82
Core#3 P#86
Core#4 P#90
Core#5 P#94
Core#0 P#75
Core#1 P#79
Core#2 P#83
Core#3 P#87
Core#4 P#91
Core#5 P#95
Bertha