Évaluation des performances de benchmarks MPI sur des architectures multiprocesseur de type CC-DSM

Meriem ZIDOUNI
meriem.zidouni@bull.net

Ghassan CHEHAIBAR
ghassan.chehaibar@bull.net

Radu MATEESCU
radu.mateescu@inria.fr

9ième Atelier en Évaluation de Performances
Aussois 1-4 juin 2008

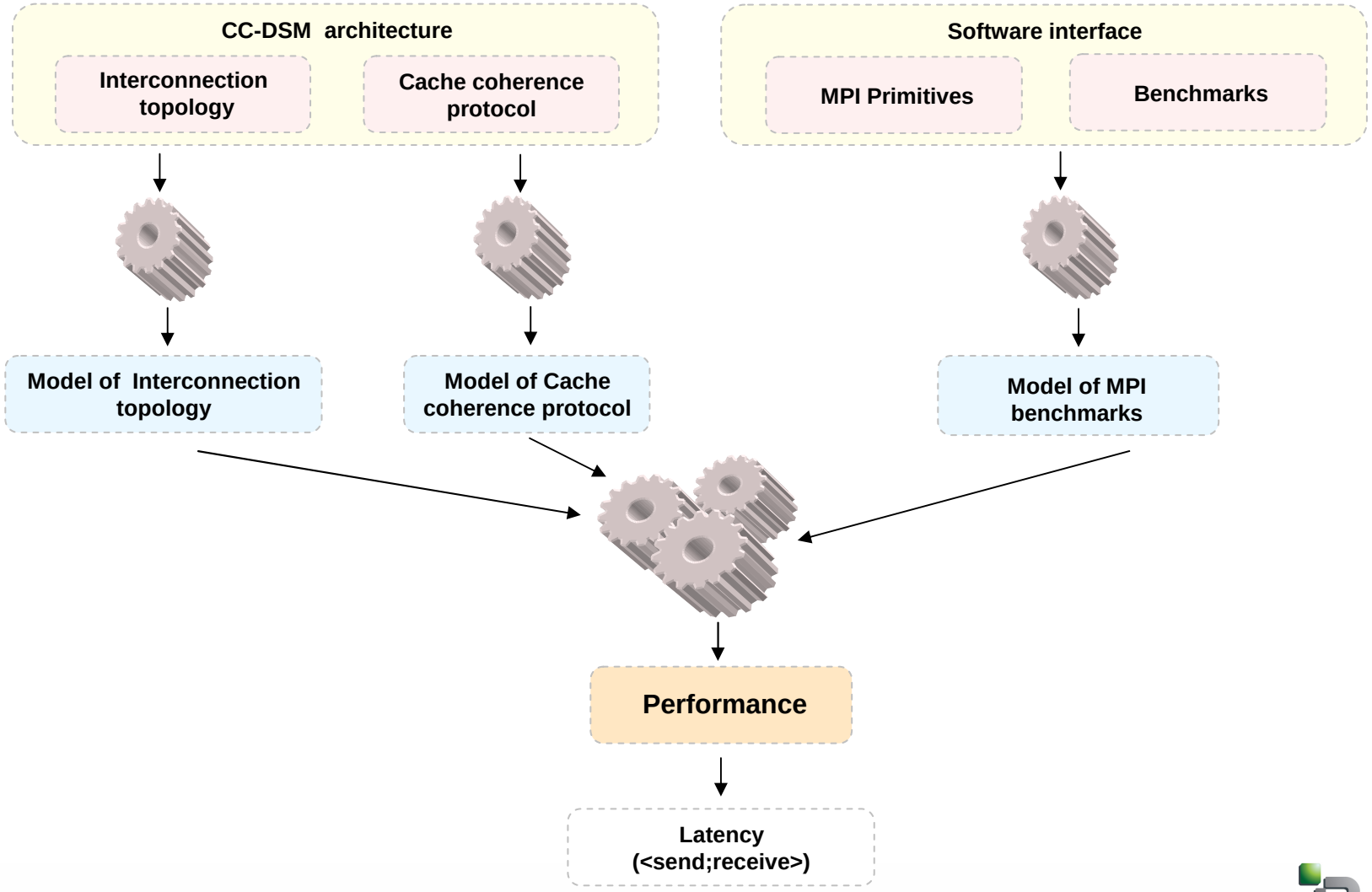
Plan

- Introduction
- Méthode de modélisation
- Données du problème :
  - Benchmark de MPI : ping-pong
  - Architecture CC-DSM
  - Protocole de cohérence des caches
  - Les latences d’accès
- Outils et environnement
- Modélisation, vérification fonctionnelle et évaluation des performances
- Résultats
- Conclusion et perspectives

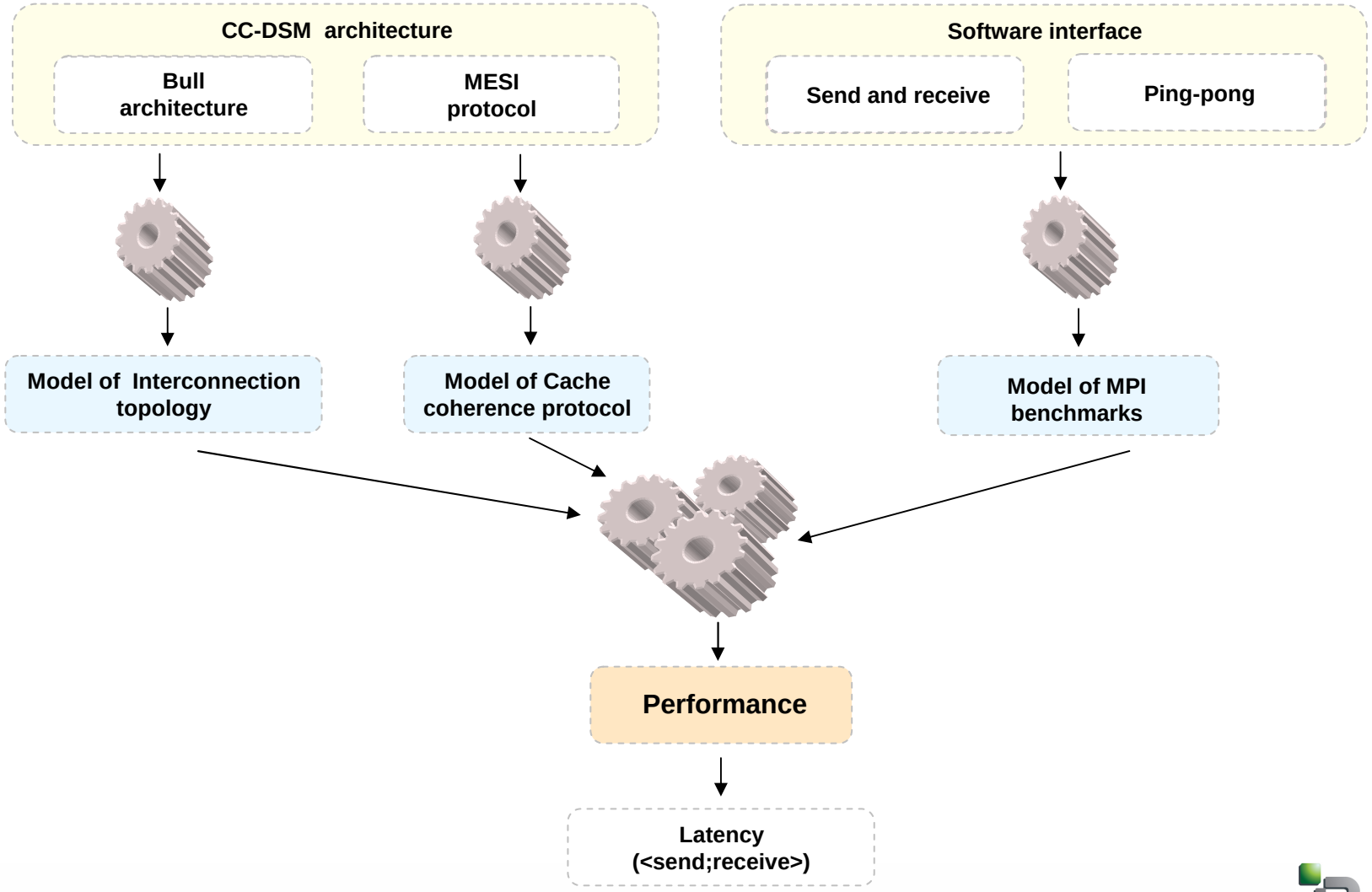
Introduction

- Bull,
  - **construit** des multiprocesseurs de type CC-DSM (*Cache Coherent-Distributed Shared Memory*) destinés au calcul scientifique haute performance (HCP : *high-Performance Computing*)
  - **fournit** une implémentation de la bibliothèque MPI (*Message Passing Interface*)
- Objectif :
  - Optimisation de l'implémentation de MPI en fonction des architectures CC-DSM pour atteindre les meilleures performances
- Proposition :
  - Une méthode formelle pour l'évaluation des performances de MPI dans les architectures CC-DSM :
    - faire les bons choix de l'architecture matérielle et d'implémentation logicielle lors de la conception;
    - analyser et justifier les mesures expérimentales

Méthode de modélisation



Méthode de modélisation

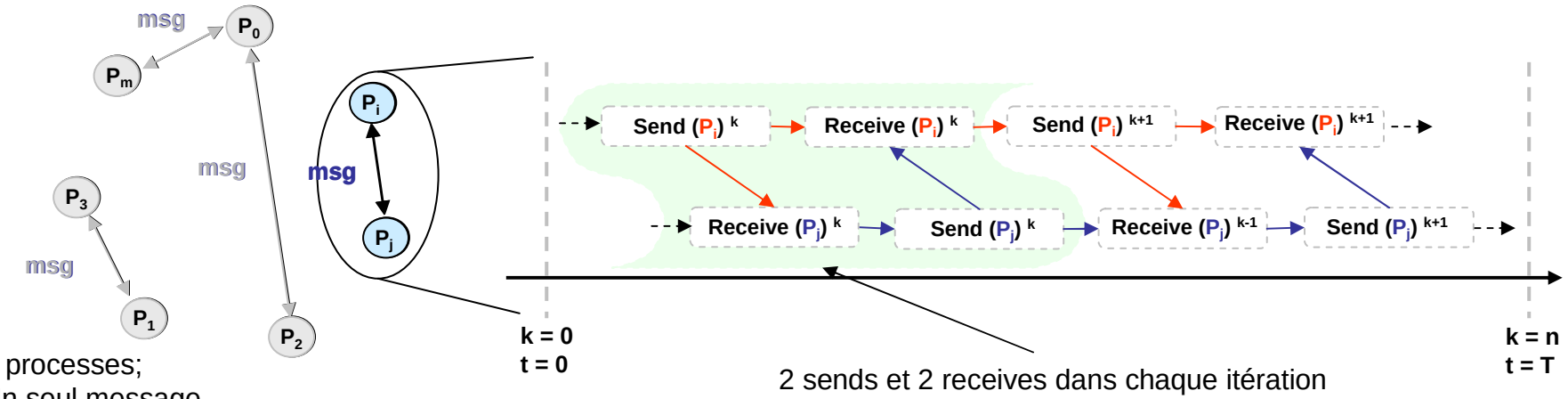


Ping-pong : benchmark de MPI

CC-DSM architecture: Bull architecture | MESI protocol

Software interface: Send and receive | Ping-pong

- Ping_pong (déf.)
  - Envois alternés de messages entre processus via les primitives *send* et *receive.*
  - **ping-pong**(P_i ,P_j) = <**send**($P_i \rightarrow P_j$)**; receive**($P_i \leftarrow P_j$)>^{n} ||| <**receive**($P_j \leftarrow P_i$)**; send**($P_j \rightarrow P_i$)>^{n}

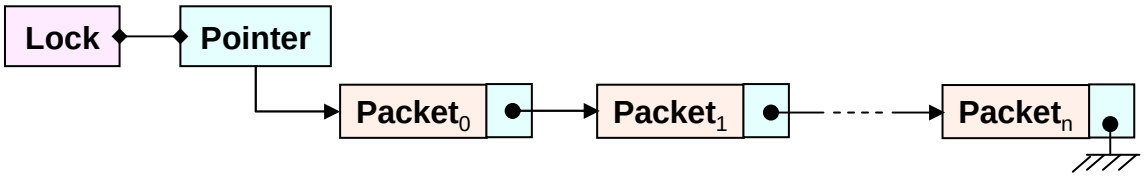


- 2 processes;
- Un seul message envoyé / reçu à la fois

- Performance (ping-pong) = latence d'un transfert d'un message de P_i à P_j (P_j à P_i)
  = T / 2n // n: nombre d'itérations
  = latency (< send ; receive >)

Send et receive : primitives de MPI

- Structures de données:
  - Un message est constitué d'un seul paquet comportant l'identificateur du processus émetteur.
  - Les paquets sont réparties dans des listes chaînées.



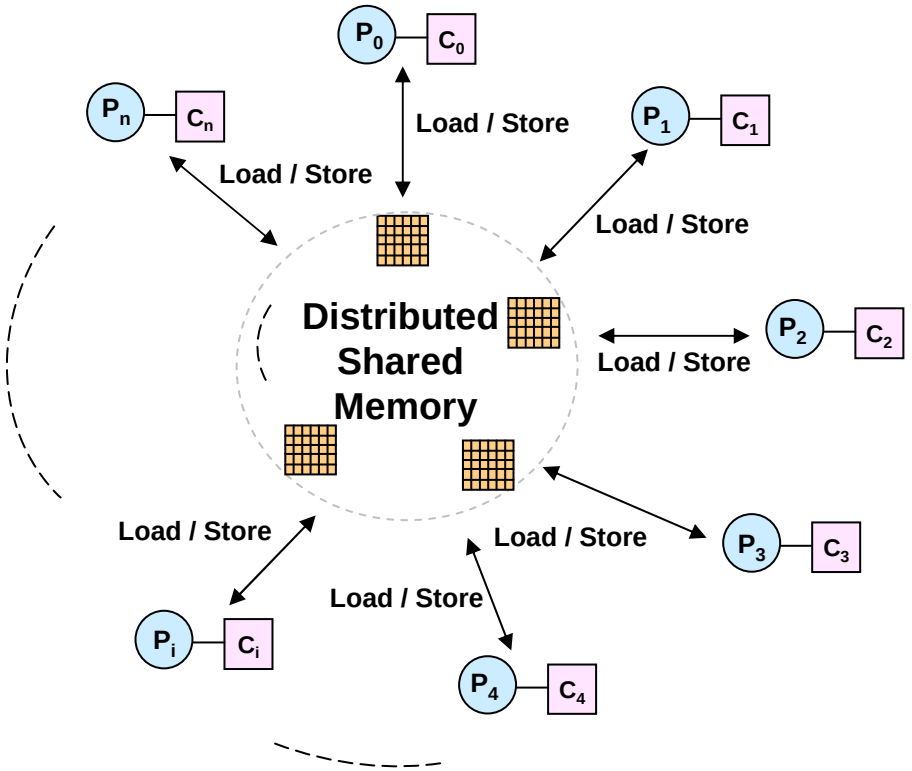
- Structures de contrôle:
  - `a := b;`
  - `if (a == b) then …`
  - `while (a != 0) do …`

Topologie de l'architecture CC-DSM de Bull

CC-DSM architecture
Bull architecture | MESI protocol

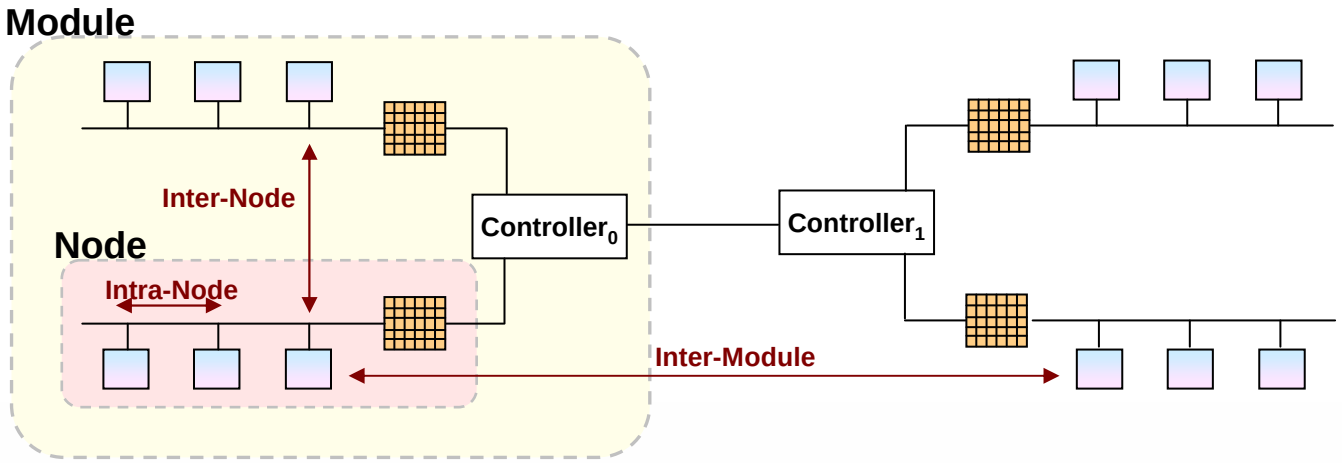
Software interface
Send and receive | Ping-pong

- CC-DSM : Cache Coherent-Distributed Shared Memory



Topologie de l'architecture CC-DSM de Bull

- CC-DSM : Cache Coherent-Distributed Shared Memory
- Topologie à 3 niveaux de distance entre les processeurs:
  - Intra-node : même nœud, même module
  - Inter-node : différents nœuds, même module
  - Inter-Module : différents nœuds, différents modules



Le protocole de cohérence des caches MESI

- États des caches : Modified (M), Exclusive(E), Shared(S) et Invalid (I);
- Type d'accès : Load , Store;
- Type de transfert : Memory, Cache, Internal;

Load protocol

Current state		Next state		Transfer type
Req C_{req}	C_j, j!=r	Req C_{req}	C_j, j!=r	
I	I	E	I	Memory
I	S	S	S	Memory
I	E	S	S	Memory
I	M	E	I	Cache
E/M/S	*	E/M/S	*	Internal

Store protocol

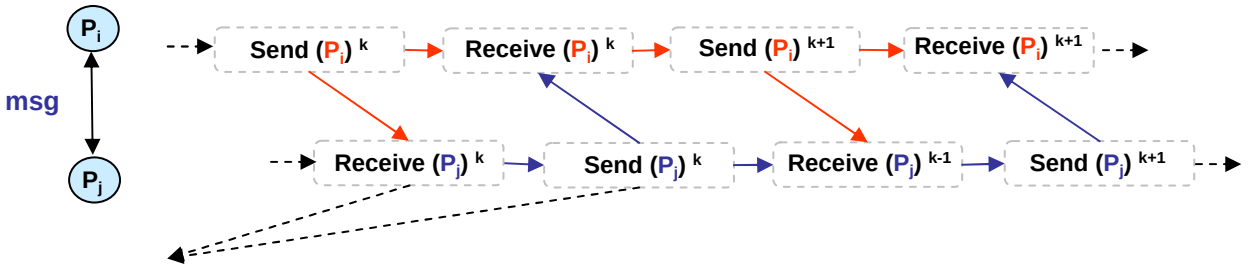
Current state		Next state		Transfer type
Req C_{req}	C_j, j!=r	Req C_{req}	C_j, j!=r	
I/S	I	M	I	Memory
I/S	S/E	M	I	Memory
I	M	M	I	Cache
E/M	*	M	*	Internal

Les latences d'accès

- La latence d'un accès dépend du :
  - type de transfert selon les états des caches;
  - niveau de transfert selon la distance entre les processeurs

Transfer type ↓ / Transfer level →

	Intra-node	Inter-node	Inter-module
Internal	I_λ	--	--
Cache	C_λ1	C_λ2	C_λ3
Memory	M_λ1	M_λ2	M_λ3



Begin < op_0; op_1; …; op_n> **End**

// $op_i \in$ { load, store}

Begin < l_0; l_1; …; l_n> **End**

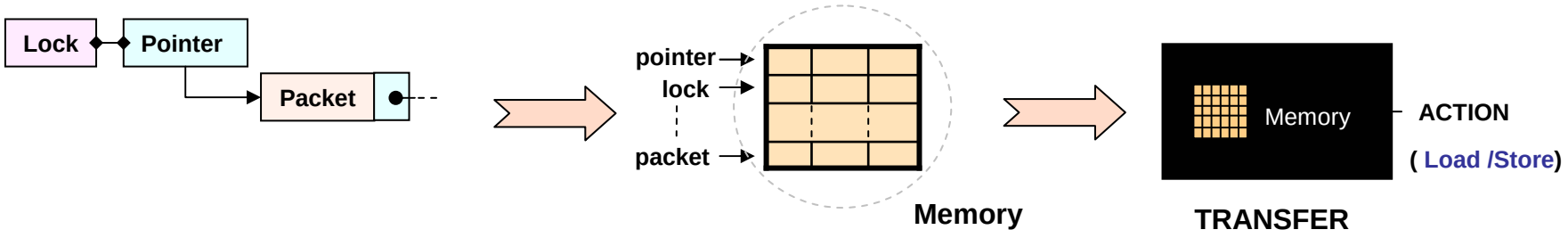
// $l_i \in$ {I_λ, C_λ1, C_λ2, C_λ3, M_λ1, M_λ2, M_λ3}

Outils et environnement

- LOTOS: Language Of Temporal Ordering Specification
  - ISO Standard [ISO-8807:1989]
  - Technique de description formelle pour la description des systèmes distribués.
- CADP: Construction and Analysis of Distributed Processes
  - Développée par l'équipe VASY de l'INRIA Rhônes-Alpes
  - Une boite à outils pour l'ingénierie des protocoles:
    - Compilateurs, générateurs de test …
    - Vérification fonctionnelle : model checking, Co-simulation …
    - Évaluation des performances basée sur la théorie des IMC
- IMC: Interactive Markov Chains
  - Thèse de Holger Hermanns (LNCS 2428)
  - Elle ajoute des caractéristiques stochastiques à l'algèbre de processus :
    - expressivité stochastique suffisante
    - compatibilité avec la théorie de l'algèbre de processus

Modélisation en LOTOS : les primitives

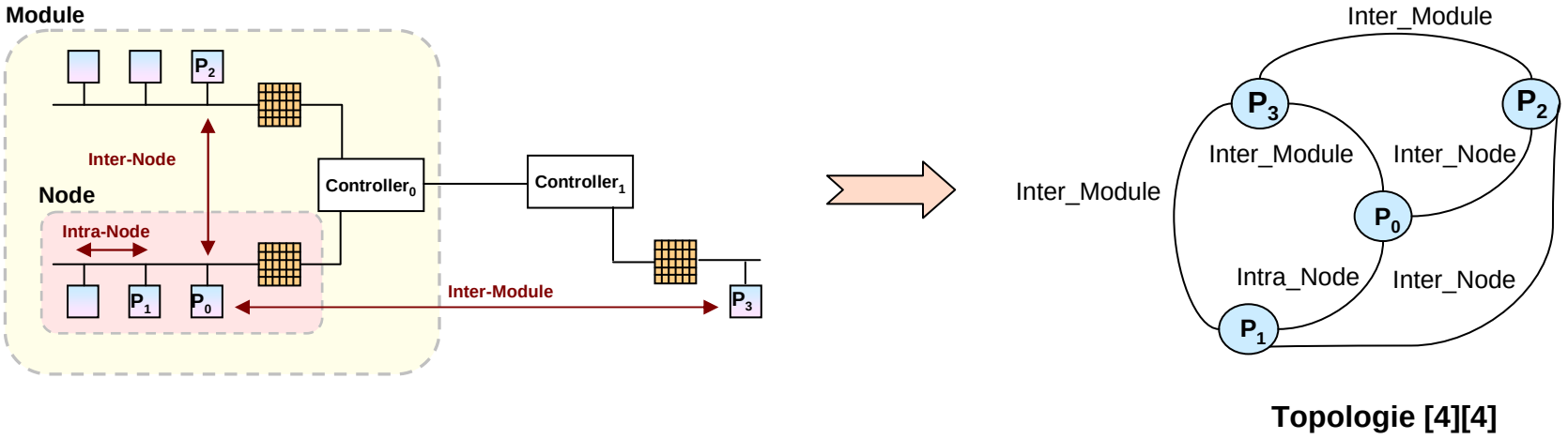
- Structures de données en LOTOS
  - Deux modes d'accès aux variables : **Load**, **Store**



- Structures de contrôle en LOTOS:
  - `a := b` ⇒ `< load(b) ; store(a,val_of_b) >`
  - `if (a == b)` ⇒
    ```
    < load(a) ; load(b) >
      ( [val_a == val_b] -> …
            []
        [val_a <> val_b] -> … )
    ```
  - `while (a != 0)` ⇒
    ```
    process Loop_While [ACTION] : exit :=
          ACTION !load ! a ? val_a ;
          ( [val_a <> 0]-> Loop_While [ACTION]
                []
          [val_a == 0]-> exit )
    endproc
    ```

Modélisation en LOTOS: la topologie

- Topologie est un graphe G = (S, E, A) :
  - Sommet S = { P_0, P_1,…, P_n}
  - Étiquette E = { Intra_node, Inter_node, Inter_module}
  - Arête A ⊆ S x E x S

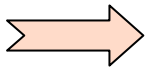


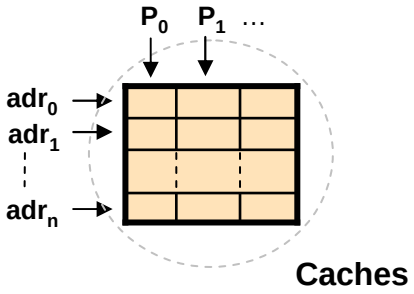
Topologie [4][4]

Modélisation en LOTOS : le protocole de cohérence des caches

- États des caches

M, S, E, I



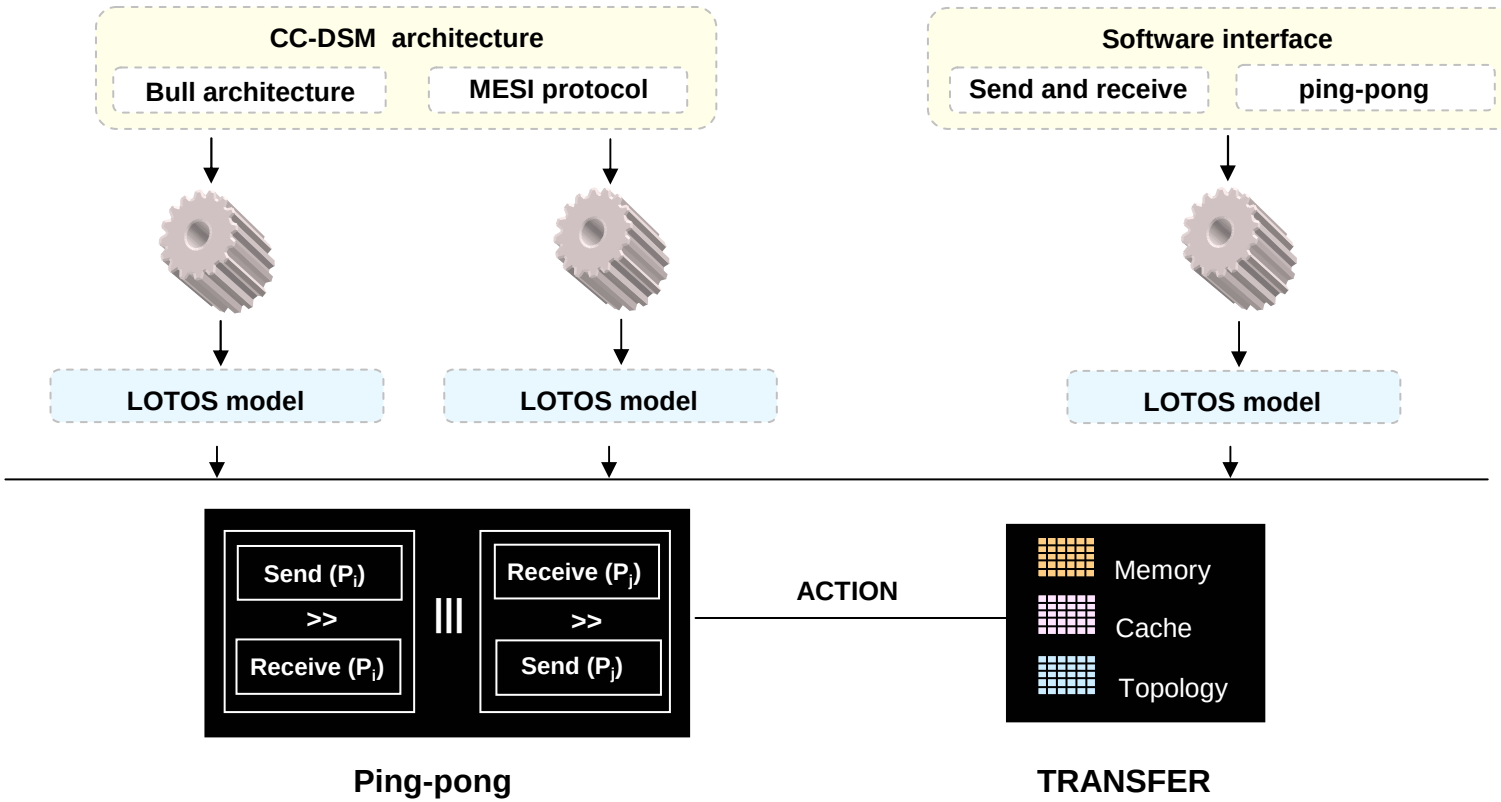


- Load/Store protocol

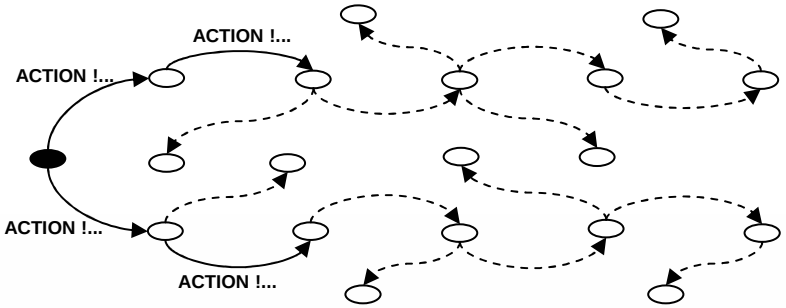
Current state		Next state		Transfer type
Req C_{req}	C_j, j!=r	Req C_{req}	C_j, j!=r	
I	I	E	I	Memory
I	S	S	S	Memory
I	E	S	S	Memory
I	M	E	I	Cache
E/M/S	*	E/M/S	*	Internal

Update_Caches : Caches × { load, store} × Address × Porcessor → Caches

Model LOTOS du ping-pong



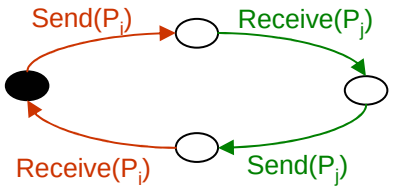
Ping_pong.bcg =



- 159,029 states
- 2,719,740 transitions

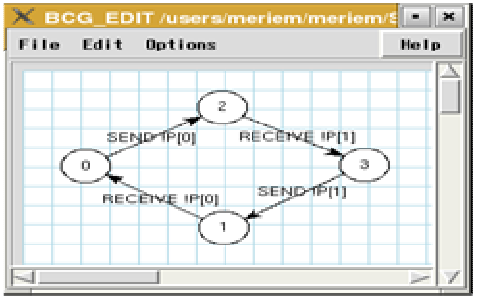
Vérification fonctionnelle du modèle

- Comportement du ping_pong :



Comportement attendu

```
"ping_pong_behaviour.bcg" =
      branching reduction of
      hide all but SEND,RECEIVE
      in "ping_pong.bcg"
```



Comportement obtenu

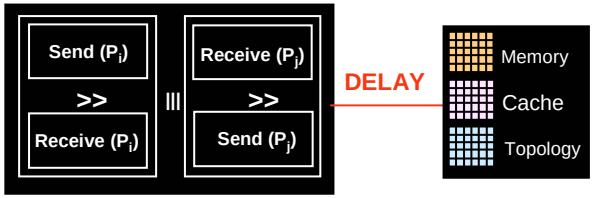
- Le protocole de cohérence des caches :
  - Le changement des états pour les opération *load* et *store*
  - La cohérence entre les états, les types et les niveaux de transferts

- Exclusion mutuelle

```
library "macros.mcl" end_library
  macro MUTEX (id_proc_1,id_proc_2,adr)=
    [ true*.
      Take_Lock (id_proc_1,adr).(not Release_Lock (id_proc_2,adr))*.Take_Lock(id_proc_2,adr)
    ] false
  end_macro
```

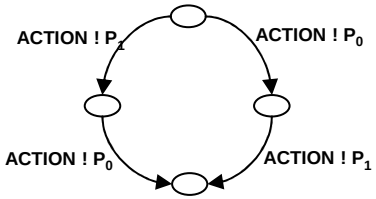
Le modèle IMC du ping-pong (1/2)

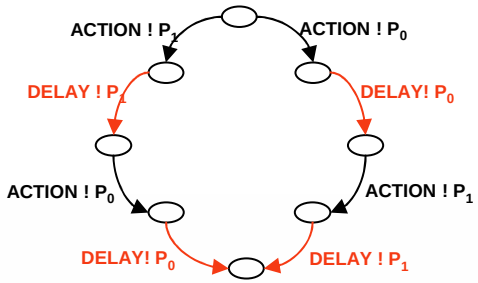
- Cas N°1:
  transformation de la porte ACTION en une porte qui représente un délais Markovien
  - X **Incorrecte** : des synchronisations sur la porte ACTION



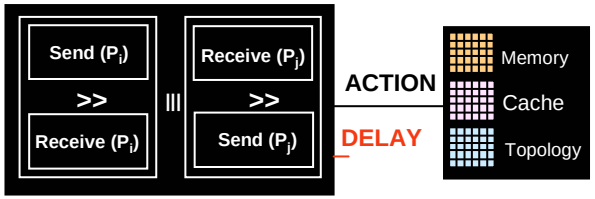
Ping-pong **Transfer**

- Cas N°2 :
  ajout d'une porte DELAY dans le processus *transfer*
  - X **Incorrecte** : la concurrence des processus parallèles dans le temps n'est pas préservée.





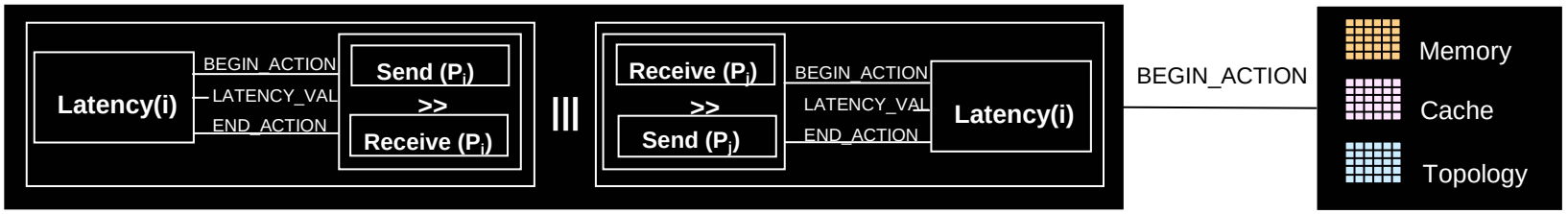
- Cas N°3 :
  ajout d'une porte DELAY dans les processus *send* et *receive*
  - √ **Correcte** : mais nécessite beaucoup de modifications dans le code des primitives send et receive.



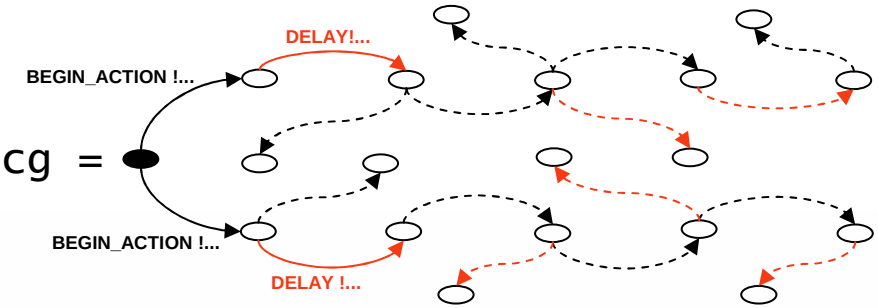
Ping-pong **Transfer**

Le modèle IMC du ping-pong (2/2)

- Gestion des latences des transferts dans un nouveau processus « Latency »:
  - Le délai de chaque action (`LATENCY_VAL`) est inséré entre son début et sa fin (`BEGIN_ACTION`, `END_ACTION`)
  - Le début et la fin de chaque action permet de prendre en compte les différentes phases d'un transfert
  - Modification de la loi de distribution sans affecter la spécification LOTOS



`Ping_pong_IMC.bcg =`



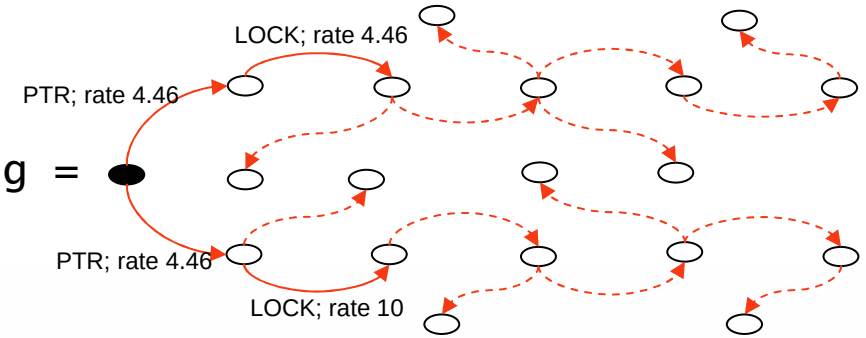
- **1,053,795 states**
- **2,254,880 transitions**

Génération de CM du modèle Markovien

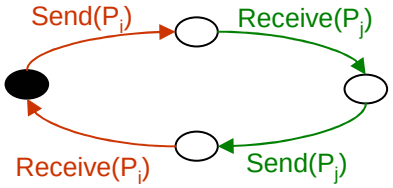
```
"ping_pong.bcg" = generation of "ping_pong.lotos";
"model.bcg "    = branching reduction of
                  hide all but LATENCY_VAL in "ping_pong.bcg";
"markovian_ping_pong.bcg" = branching stochastic reduction of total rename
          "LATENCY_VAL ! Lock_A ! M_FSB_1" -> "LOCK; rate 4.46",
          "LATENCY_VAL ! Lock_B ! C_FSB"   -> "LOCK; rate 10
           ...
          in "model.bcg";

 % bcg_steady -thr -append Rate_Intra_Node.csv
```

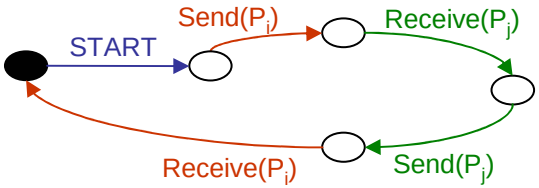
- BCG_STEADY : calcul l'état d'équilibre de la distribution de probabilité sur le long terme, et fournit des mesures de débit pour chacune des étiquettes de transitions.

Markovian_ping_pong.bcg =



Les résultats (2/3)





- **Throughput** (START): la fréquence de la transition START calculée par BCG_STEADY
- **Latence** = 1/(2 * Throughput(START))

	Latence (µs)			
	Primitives SR1		Primitives SR2	
	Protocole A	Protocole B	Protocole A	Protocole B
Intra-node	1	2.45	0.65	0.85
Inter-node	3.28	5.71	1.69	2.55
Inter-module	5.52	9.64	2.79	4.22

Current state		Next state		Transfer type
Req C_{req}	C_j, j!=r	Req C_{req}	C_j, j!=r	
I	I	E	I	Memory
I	S	S	S	Memory
I	E	S	S	Memory
I	M	~~E~~ S	~~X~~ S	Cache
E/M/S	*	E/M/S	*	Internal

- Basées sur le principe producteur-consommateur
- Les listes sont remplacées par des tampons sans verrous

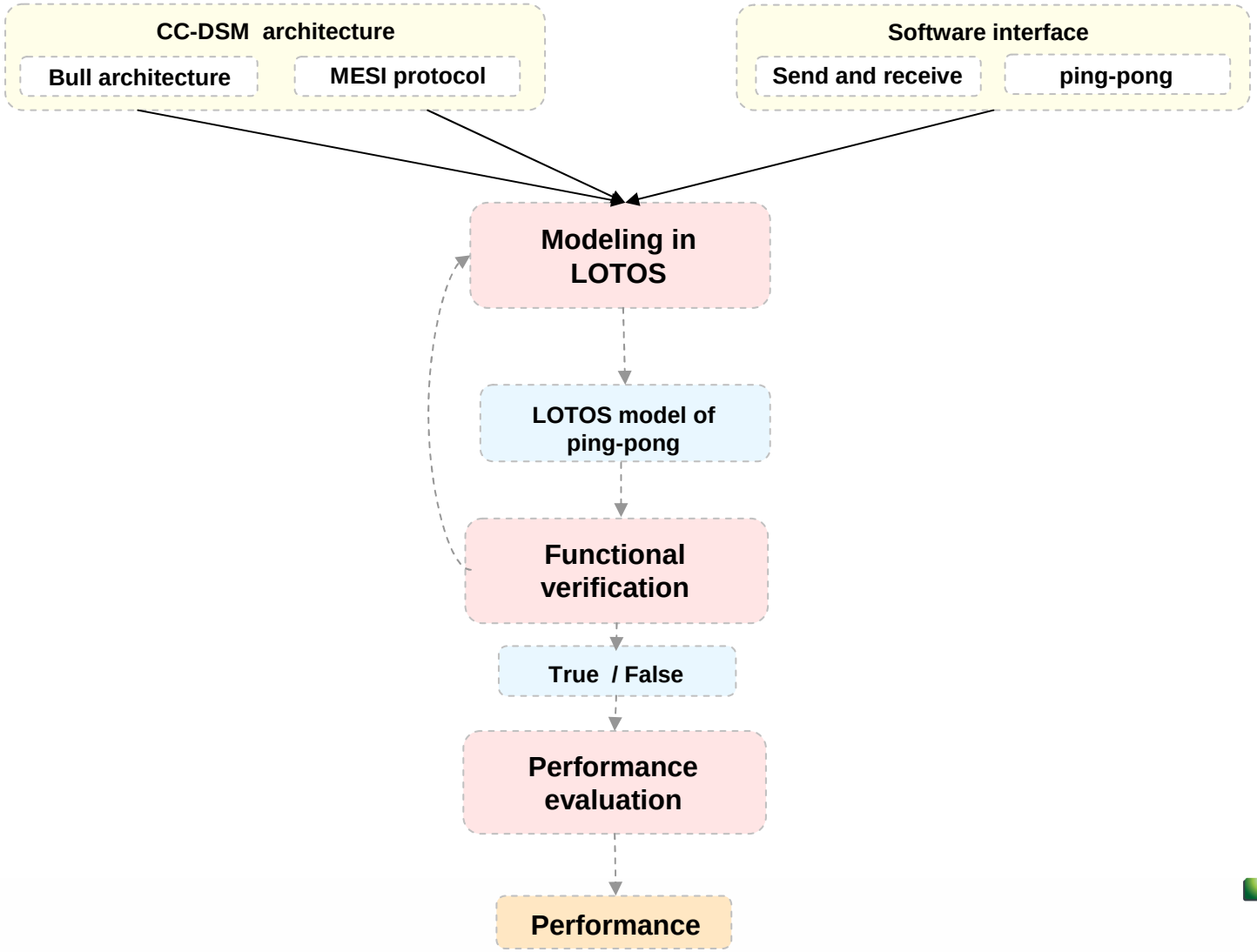
Les résultats (2/2)

- **Latence** = 1/(2 ∗Throughput (START))
- **Throughput** (VAR) : la fréquence des transitions qui correspondent aux misses effectuées sur la variable VAR
- **Nb_Misses** (VAR) : nombre de misses effectuées sur la variable VAR pendant une durée Latence

- **Nb_Misses** (VAR) = Latence ∗Throughput (VAR)

	Nombre de miss									
	Primitives SR1						Primitives SR2			
	Protocole A			Protocole B			Protocole A		Protocole B	
	Packet	**Pointer**	**Lock**	**Packet**	**Pointer**	**Lock**	**Packet**	**Pointer**	**Packet**	**pointer**
Intra-node	4	8	7	6	14	15	4	7	4	8
Inter-node	4	9	7	6	15	13	4	8	5	10
Inter-module	4	9	7	6	13	15	4	8	5	10

Conclusion


CC-DSM architecture
Bull architecture
MESI protocol
Software interface
Send and receive
ping-pong
Modeling in LOTOS
LOTOS model of ping-pong
Functional verification
True / False
Performance evaluation
Performance

Conclusion

- Modélisation en LOTOS :
  - les primitives send et receive,
  - le protocole de cohérence des caches,
  - et la topologie du système.
- Vérification fonctionnelle du modèle LOTOS du ping-pong
- Évaluation des performances du modèle ping-pong :
  - Obtention de résultats cohérents et conformes aux mesures expérimentales.
  - Analyse et comparaison des latences dans le cas de 2 primitives différentes de MPI, 2 protocoles de cohérence des caches et 3 topologies différentes

Perspectives

- Résoudre le problème d'explosion d'espace d'états
- Traduction automatique d'un algorithme MPI en LOTOS :
  - Structures de données
  - Structures de contrôle : if, then, else, while …
- Modélisation des conflits dans les accès aux données
- Généralisation de la méthode
- …

Merci

???