Performance Evaluation of MPI Benchmarks on CC-DSM Multiprocessor Architectures

Meriem ZIDOUNI
meriem.zidouni@bull.net

Ghassan CHEHAIBAR
ghassan.chehaibar@bull.net

Radu MATEESCU
radu.mateescu@inria.fr

2-3 April 2008 **Model35** **INRIA-Paris Rocquencourt**

Agenda

- Introduction
- Modeling language: LOTOS
- The CADP toolbox
- MPI benchmark: ping-pong
- LOTOS model of:
  - Send & receive primitives
  - Interconnection topology
  - Cache coherence protocol
- Functional verification
- Performance evaluation
- Conclusion & perspectives

Introduction

- **BULL** builds **supercomputers** for high-performance scientific computing
- Supercomputer =

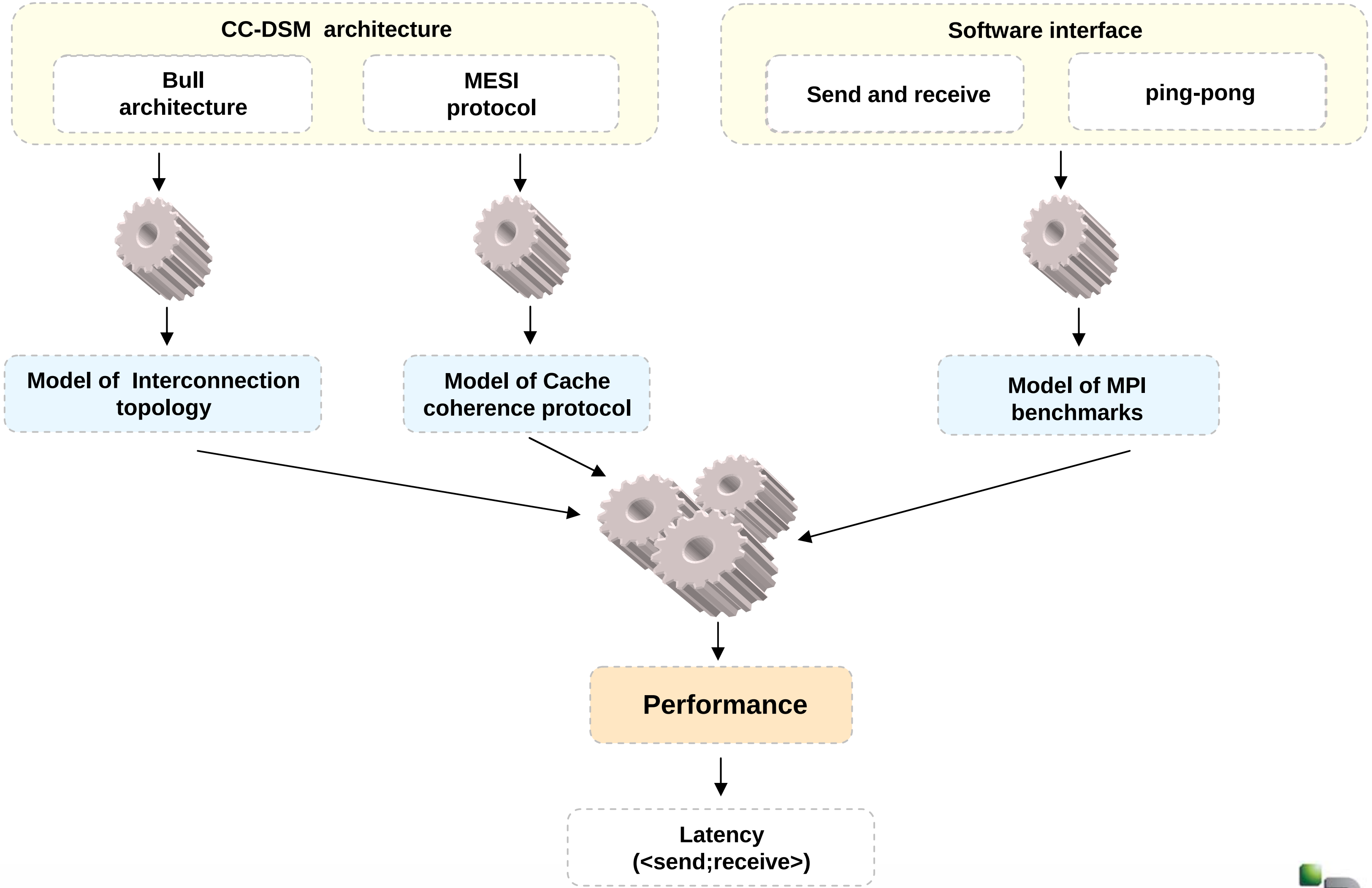
  Hardware architecture + Software interface

  (CC-DSM: *Cache Coherent-Distributed Shared Memory*)

  (MPI: *Message Passing Interface)*

- High performance supercomputer $\Rightarrow$
  - BULL has to optimize MPI implementation for its servers hardware architecture
- We need a model to evaluate performance and analyze experimental measures taking into account:
  - Cache coherence protocol and architecture topology
  - MPI software algorithm

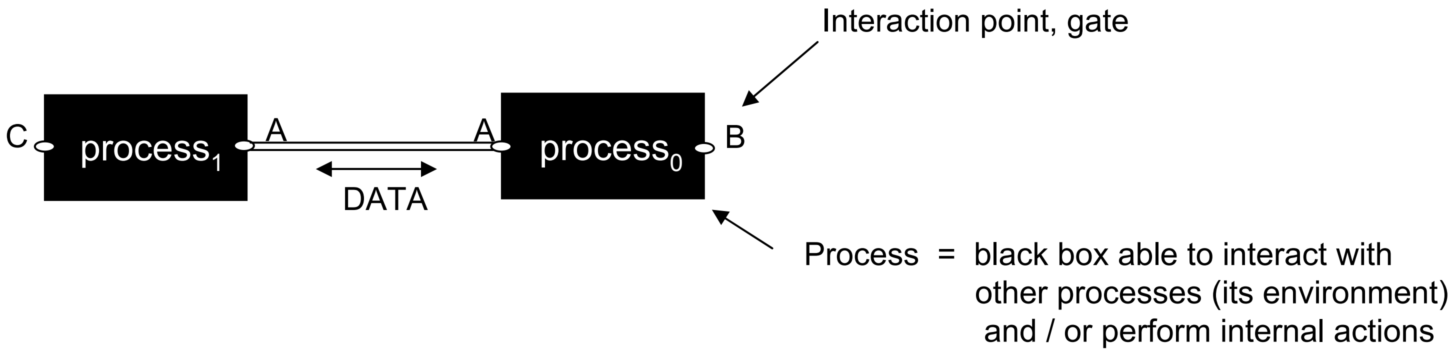
Introduction: modeling method



Modeling language: LOTOS

(*Language Of Temporal Ordering Specification*)

- ISO Standard [ISO-8807:1989]
- A Formal Description Technique for the specification of protocols and distributed systems
- Two orthogonal sub-languages:
  - Data: abstract data types (ActOne)
    - sorts and operations
    - algebraic equations
  - Processes: process algebras (~CCS, CSP, Circal)
    - parallel processes (interleaving semantics)
    - message-passing communication



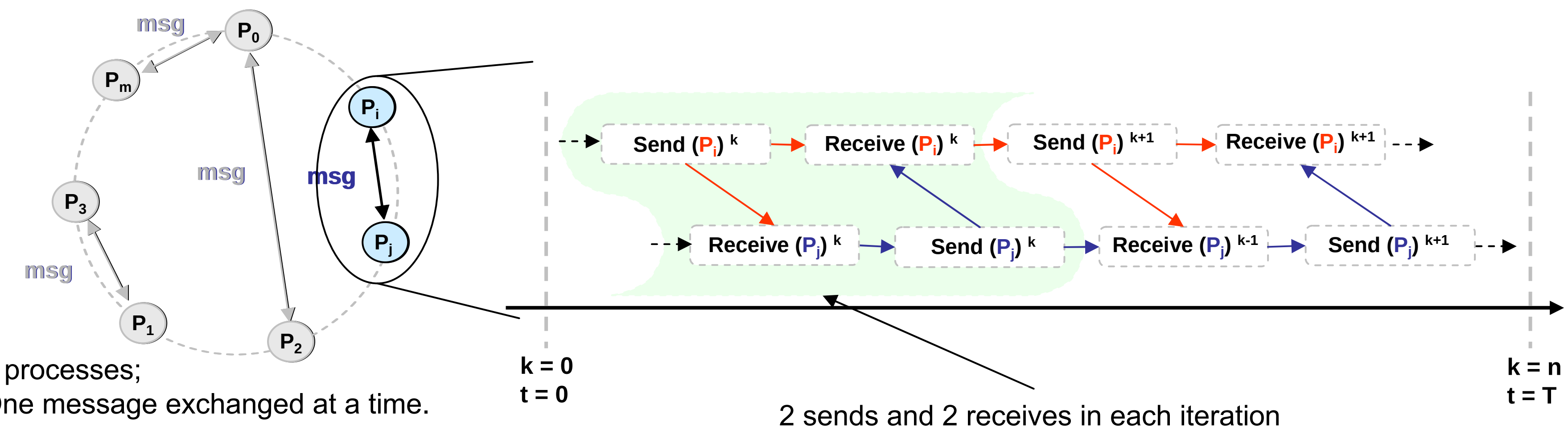
The CADP toolbox

(*Construction and Analysis of Distributed Processes*)

- Developed at INRIA Rhône-Alpes by the VASY team (http://www.inrialpes.fr/vasy/cadp)
- Toolbox for protocol and distributed systems engineering
- CADP tools useful for hardware design:
  - Compilers, translators and model generators
  - Functional verification:
    - Model checking (modal mu-calculus), equivalence checking (bisimulations)
    - Co-simulation (RTL – LOTOS)
  - Performance evaluation:
    - Functional models enriched with quantitative information (delays). Performance evaluation based on IMC theory.

MPI benchmark: ping-pong

- Benchmark ping-pong (definition):
  Alternated transmission of messages between processes
  using *send* and *receive* primitives
- **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}



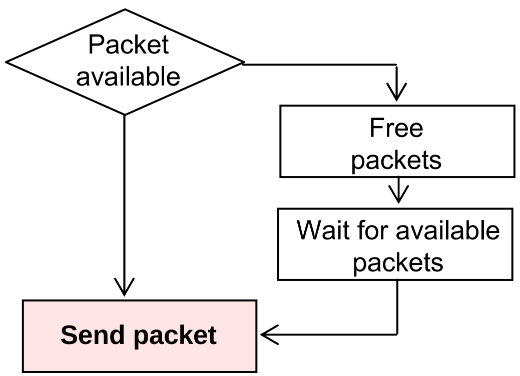
- 2 processes;
- One message exchanged at a time.

- Performance (ping-pong) = latency of message transfer from P_i to P_j (P_j to P_i)
  = T / 2n   // n: number of iterations
  = latency (< send ; receive >)

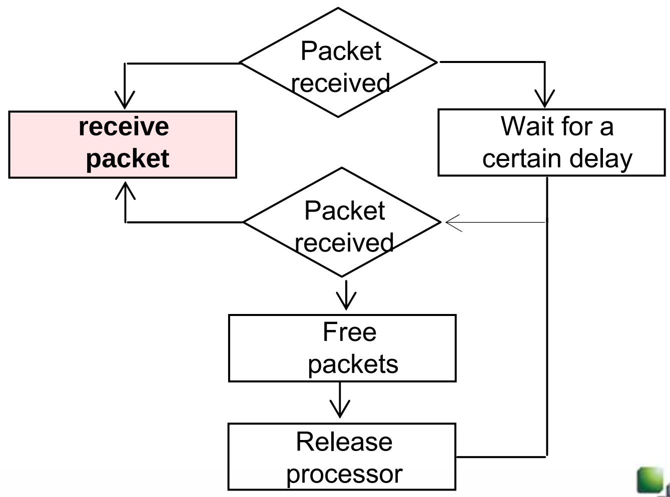
MPI library: *send* & *receive* primitives

- The data structures:
  - The exchanged message consists of a packet containing the identifier of the sender processes
  - The packets are distributed in 3 types of linked lists:
    1. list of available packets
    2. list of incoming packets
    3. list of free packets
  - 3 types of variables: pointer, lock and packet

- Send and receive primitives:
  - Send



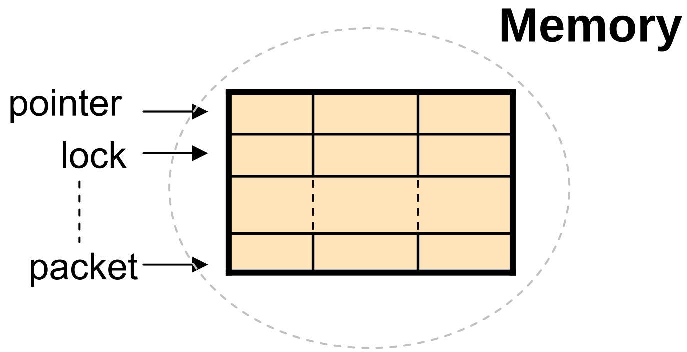
  - Receive

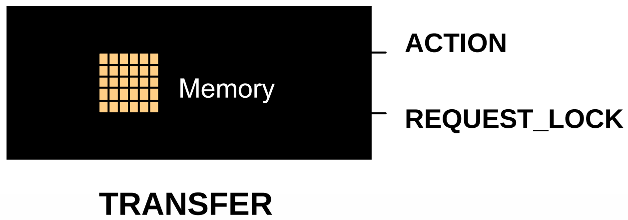


LOTOS model of send and receive primitives: data structures

- The data structures :
  - Pointers, locks and packets are defined in memory data structure
  - Memory structure is managed by LOTOS process (*TRANSFER*)

```
type Address is Natural, ID_Processor
sorts Address (*! implementedby ADT_ADDRESS *)
opns
 Local_Available_Pkt_Ptr (*! Implementedby
  ADT_LOCAL_AVAILABLE_PKT_PTR constructor external *),
 Available_Pkt_Ptr       (*! … *),
 Available_Pkt_Ptr_Lock (*! … *),
 Free_Pkt_Head_Ptr       (*! … *),
 Free_Pkt_Tail_Ptr       (*! … *),
 Incoming_Pkt_Head_Ptr  (*! … *),
 Incoming_Pkt_Tail_Ptr  (*! … *),
 Incoming_Pkt_Ptr_Lock  (*! … *),
 Pkt_Ptr                 (*! … *) : ID_Processor -> Address
endtype
```



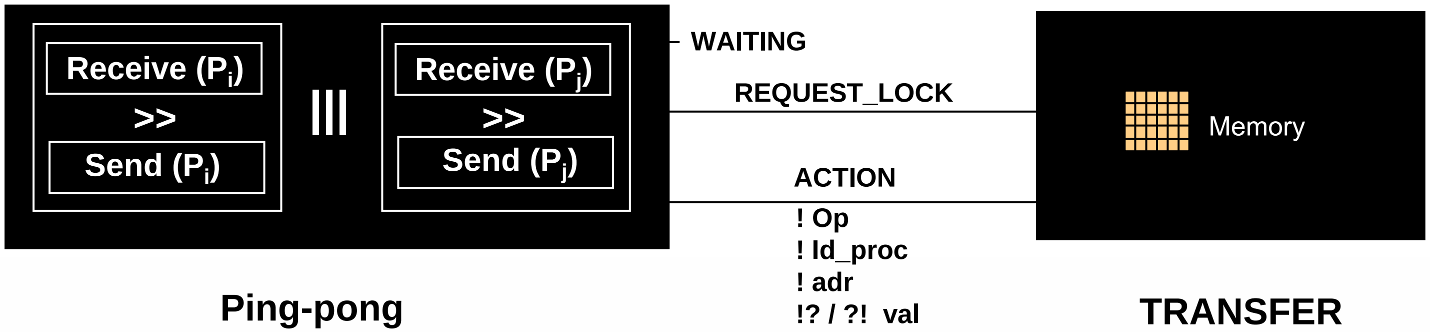


LOTOS model of send and receive primitives: control structures

- Two types of data access: load and store
- Control structures:
  - Assignment: a := b ⇒ < load(b) ; store(a,val_of_b) >
  - Test: if (a == b) ⇒ < load(a); load(b) >
  - Loop: while (a != 0) ⇒

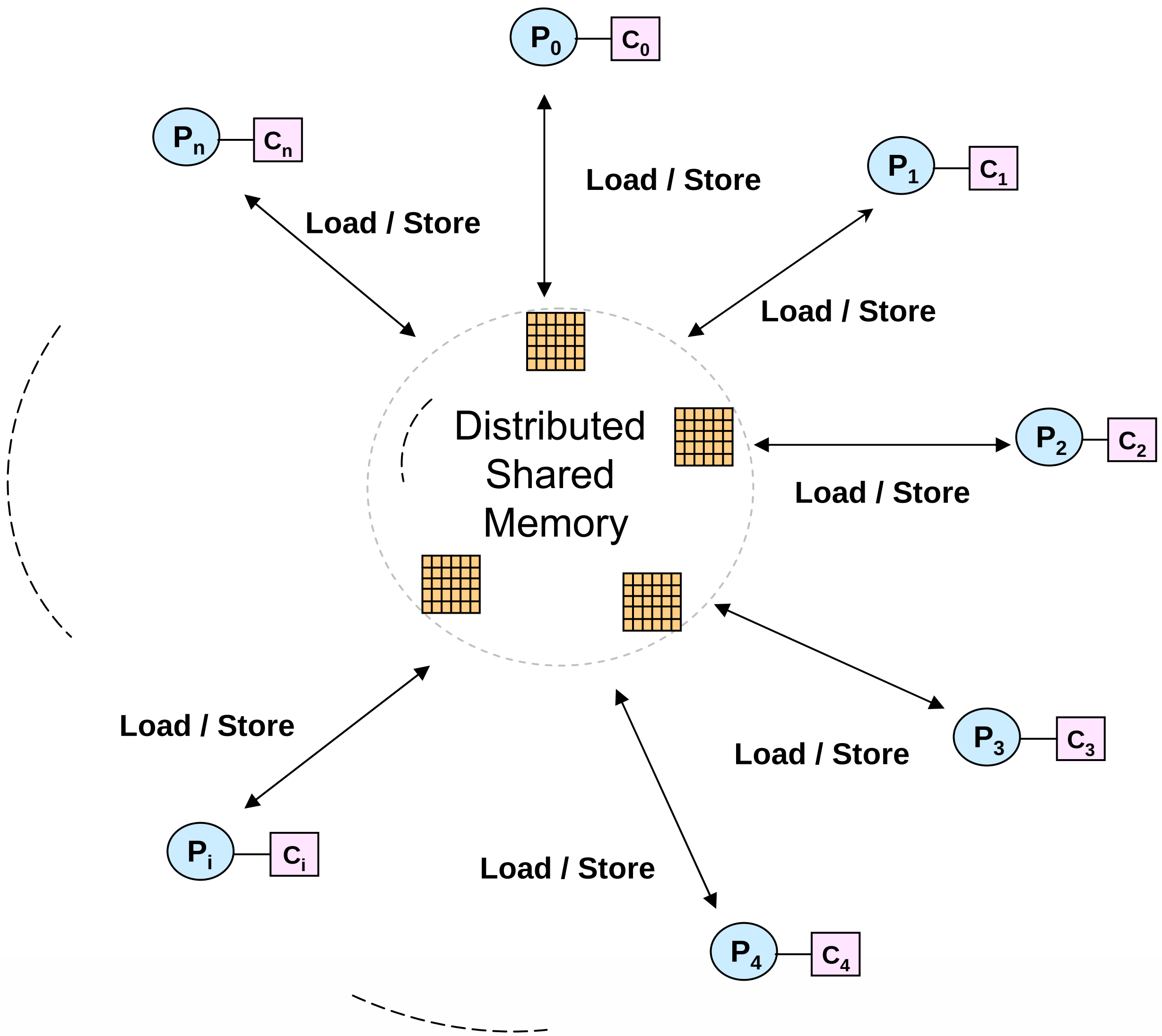
```
process Loop_While [ACTION] : exit :=
   ACTION ! a ? val_a ;
     ( [val_a <> 0]-> Loop_While [ACTION]
       []
       [val_a == 0]-> exit )
endproc
```

  - Wait: no access to variables



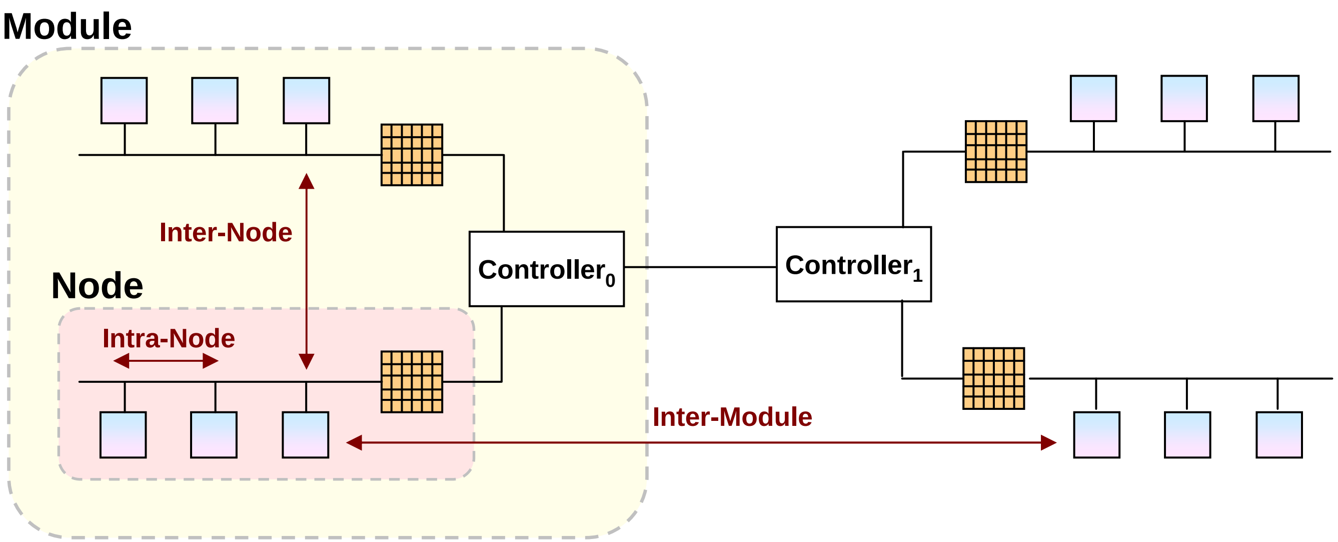
CC-DSM architecture

(*Cache Coherent-Distributed Shared Memory*)

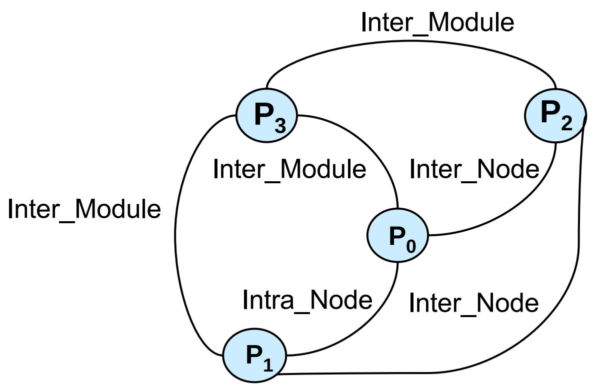


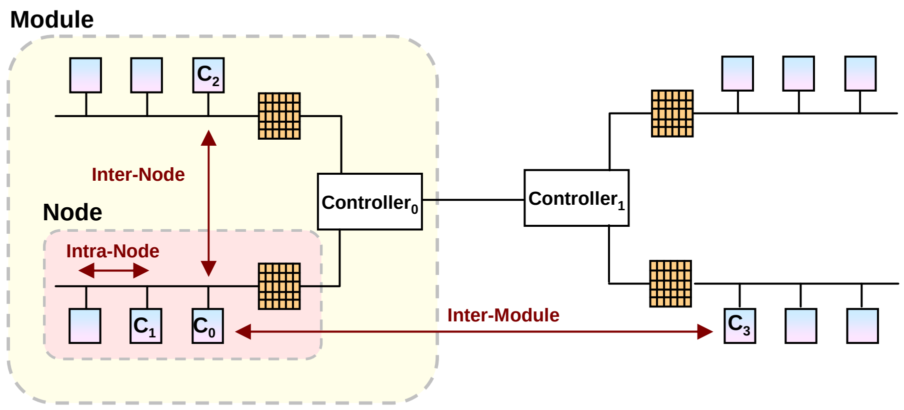
Bull architecture

- Architecture with 3 levels of distance between processors:
  - Intra-node: same node, same module
  - Inter-node: different nodes, same module
  - Inter-module: different nodes, different modules



LOTOS model of Bull architecture





Topology **[Nb_Proc][Nb_Proc] =**

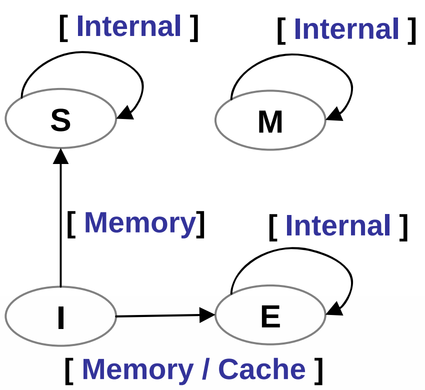
	P_0	P_1	P_2	P_3
P_0	-	Intra_Node	Inter_Node	Inter_Module
P_1	Intra_Node	-	Inter_Node	Inter_Module
P_2	Inter_Node	Inter_Node	-	Inter_Module
P_3	Inter_Module	Inter_Module	Inter_Module	-

MESI cache coherence protocol

- States of caches: Modified (M), Exclusive (E), Shared (S) and Invalid (I)
- Transfer type: Memory, Cache, Internal

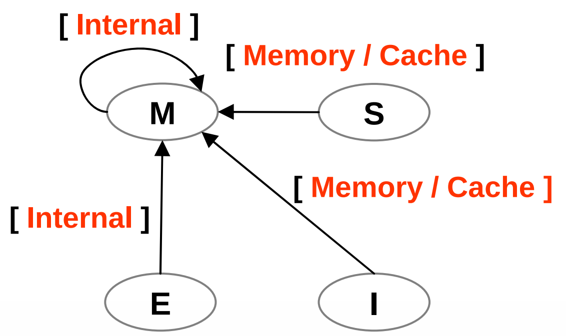
Load protocol

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



Store protocol

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



LOTOS model of cache coherence protocol

Caches **[Size_Memory][Nb_Proc] =**

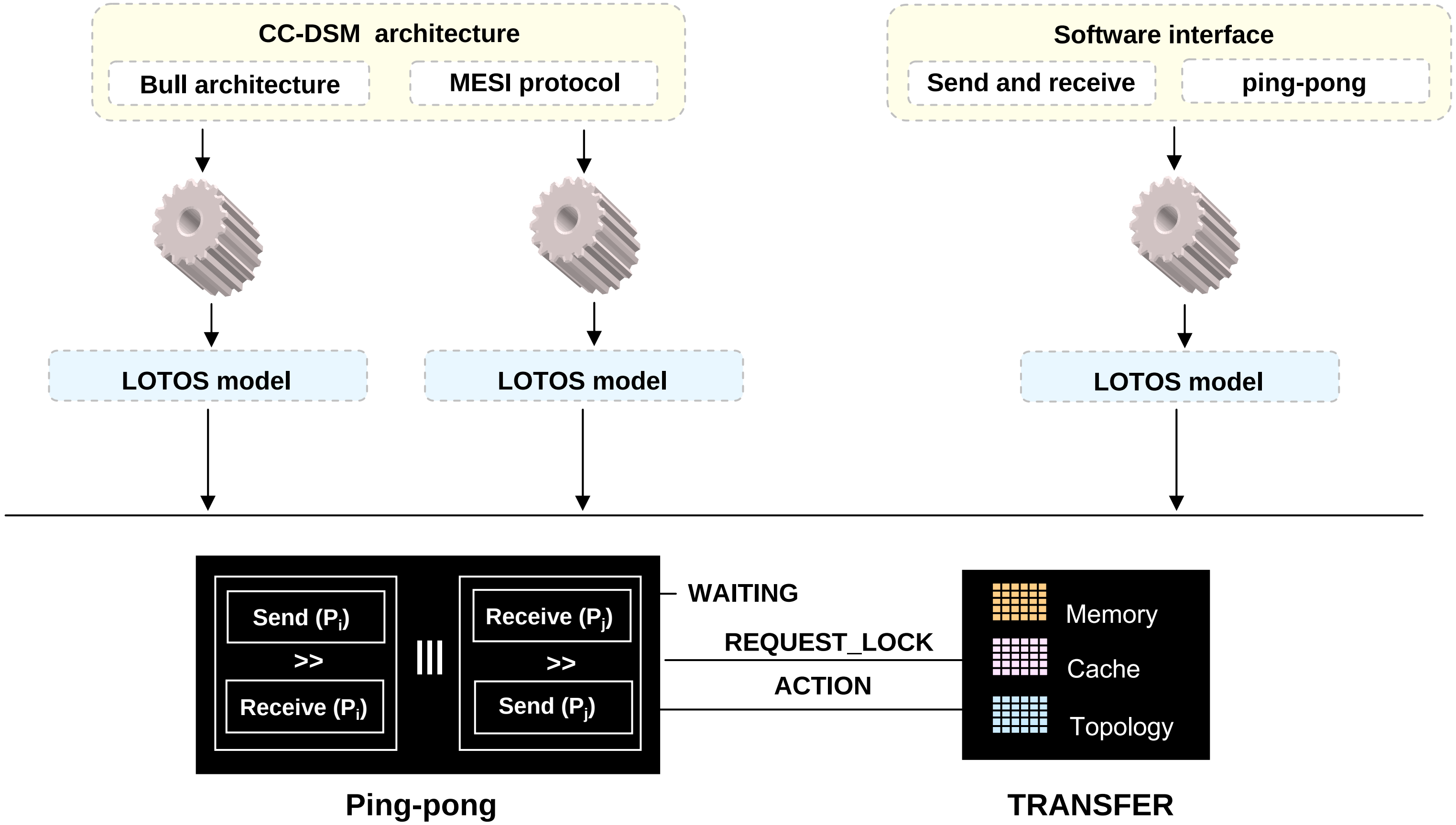
	P_0	P_1	P_2	P_3
adr_0	I	I	M	I
adr_1	S	I	S	S
adr_2	E	I	I	I
adr_3	I	I	I	I

```
type Cache is Address, ID_Action, ID_Processor
 sorts
  Cache (*! implementedby ADT_CACHE external *)
 opns
  Init_Cache      (*! implementedby ADT_INIT_CACHE constructor external *):-> Cache
  Update_Cache    (*! implementedby ADT_UPDATE_CACHE external *):
                                 Cache,ID_Action,Address,ID_Processor -> Cache
endtype
```

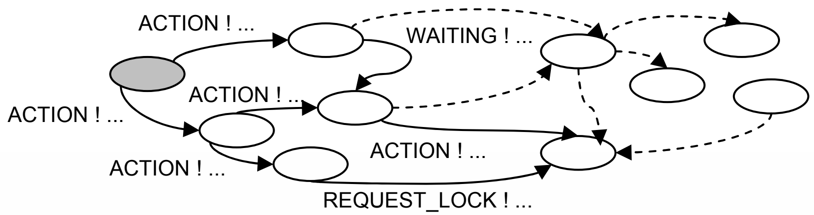
Load protocol → `Update_Cache (Caches,LOAD,adr,ID_Processor)`

Store protocol → `Update_Cache (Caches,STORE,adr,ID_Processor)`

Ping-pong model

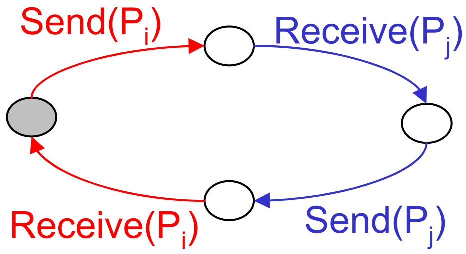


Ping_pong.bcg =

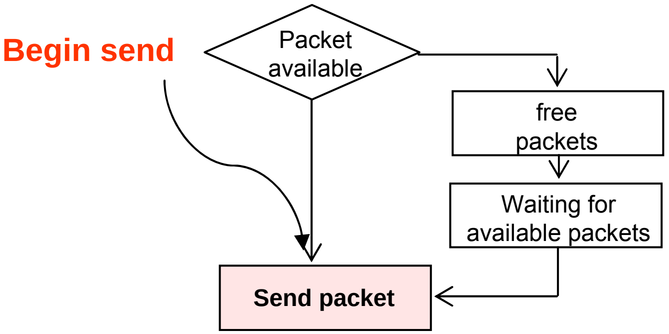


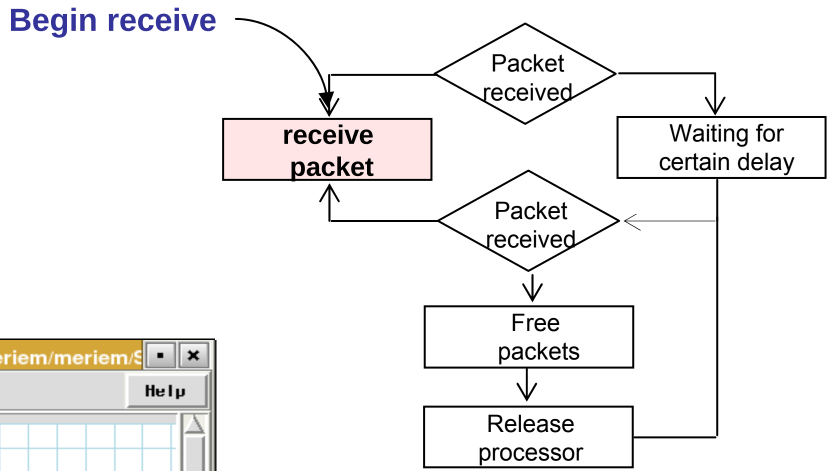
- **159,029 states**
- **2,719,74 transitions**

Functional verification: ping-pong behavior

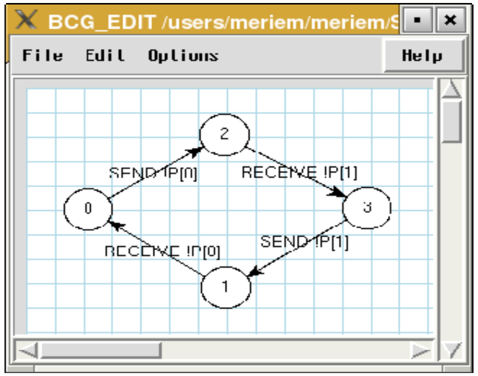


Expected behavior






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

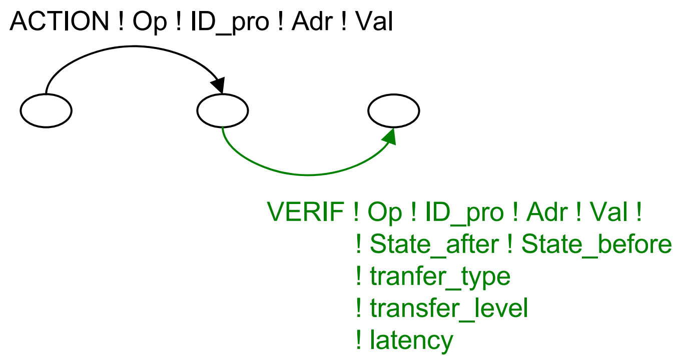


Obtained behavior

Functional verification:
cache coherence protocol & mutual exclusion

Cache coherent protocol

- update of cache state
- transfer types
- transfer levels
- transfer latency

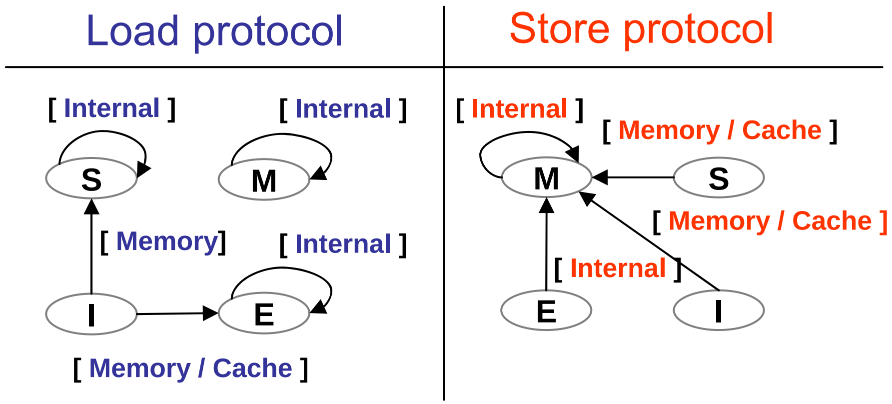


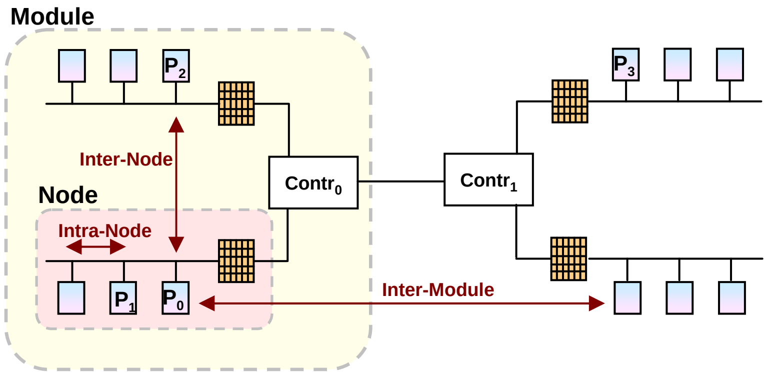

```
library "macros.mcl" end_library
   [ true*.
      ( Action_State_Before ('LOAD','0','I','I') and
        not Action_State_After('LOAD','0','I','I','E','I','MEMORY'))
   ] false
```

Mutual exclusion

```
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
```

Performance evaluation: access latencies





Transfer level →

	Intra_Node	Inter_Node	Inter_Module
Internal	$I\_\lambda 1$	$I\_\lambda 2$	$I\_\lambda 3$
Cache	$C\_\lambda 1$	$C\_\lambda 2$	$C\_\lambda 3$
Memory	$M\_\lambda 1$	$M\_\lambda 2$	$M\_\lambda 3$

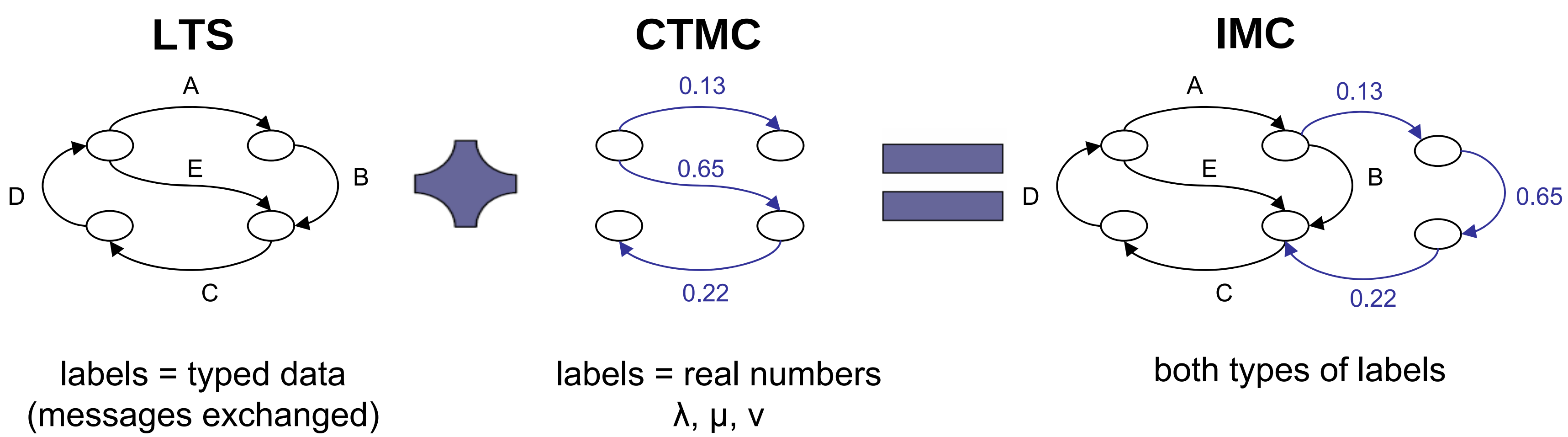
↑

Transfer type

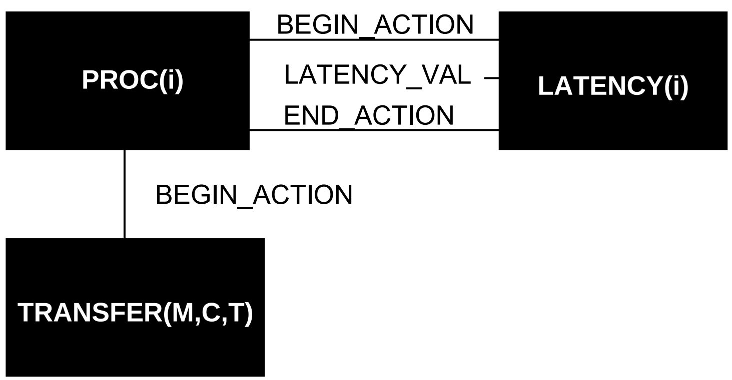
Latencies for load and store access

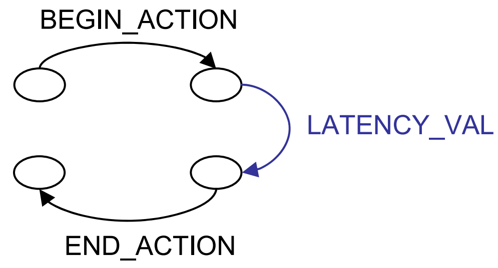
Performance evaluation: Interactive Markov Chains (IMC)

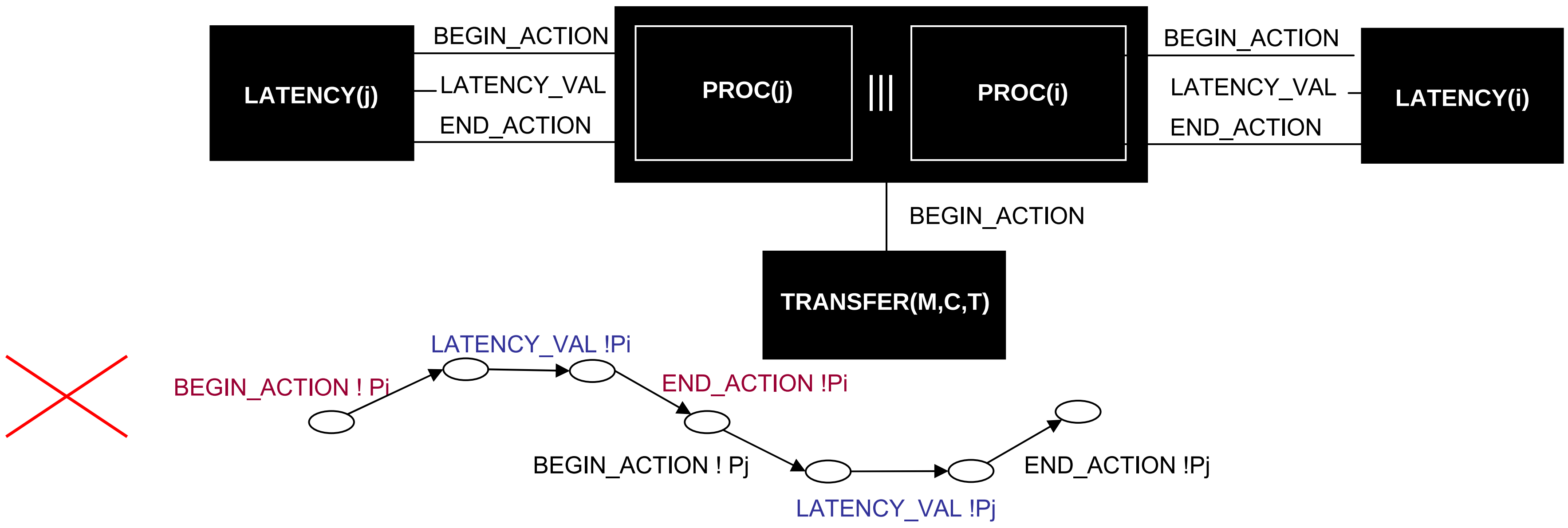
- Defined in H. Hermanns' PhD thesis (LNCS 2428)
- It adds stochastic features to process algebra, still providing:
  - sufficient stochastic expressivity
  - compatibility with process algebra theory
  - useful compositionality results



Performance evaluation: insertion of Markovian delays in ping-pong specification

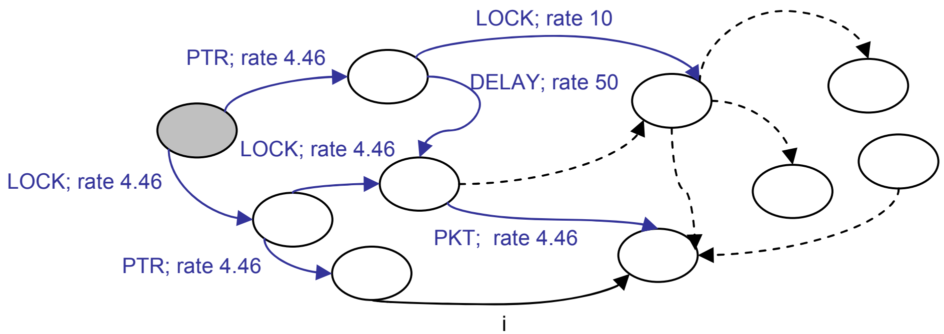






Performance evaluation: generation of MC of ping-pong

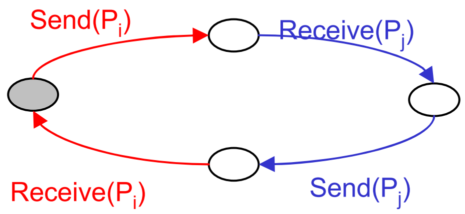
```
"ping_pong.bcg" = generation of "ping_pong.lotos";
"model.bcg"    = branching reduction of
                 hide all BEGIN_ACTION, END_ACTION, REQUEST_LOCK, WAITING
                 in "ping_pong.bcg";
"markovian_ping_pong.bcg" = branching stochastic reduction of total rename
           "DELAY"                                          -> "DELAY; rate 50",
           "LATENCY ! Incoming_Pkt_Ptr_Lock[P0] ! M_FSB_1" -> "LOCK; rate 4.46",
           "LATENCY ! Incoming_Pkt_Ptr_Lock[P0] ! C_FSB"   -> "LOCK; rate 10
            ...
           in "model.bcg";

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

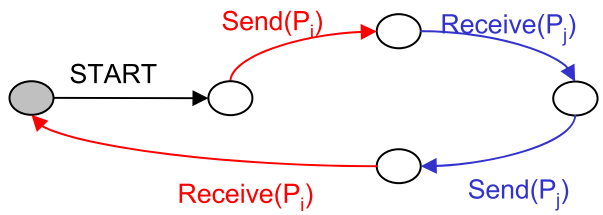


markovian_ping_pong.bcg

computes the corresponding equilibrium ("steady-state") probability distribution on the long run using the Gauss/Seidel algorithm

Performance evaluation: results





- **Throughput** (START): START transition frequency evaluated by BCG_STEADY
- **Latency** = 1/(2 * Throughput(START))

	Latency (μs)			
	Primitives SR1		Primitives SR2	
	Protocol A	Protocol B	Protocol A	Protocol 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**	
Req C_{req}	C_j, j!=r	Res C_{req}	C_j, j!=r
I	I	E	I
I	S	S	S
I	E	S	S
I	M	~~E~~ S	X S
E/M/S	*	E/M/S	*

- 2 variables for each process
- Lock-free implementation with fixed-size buffers

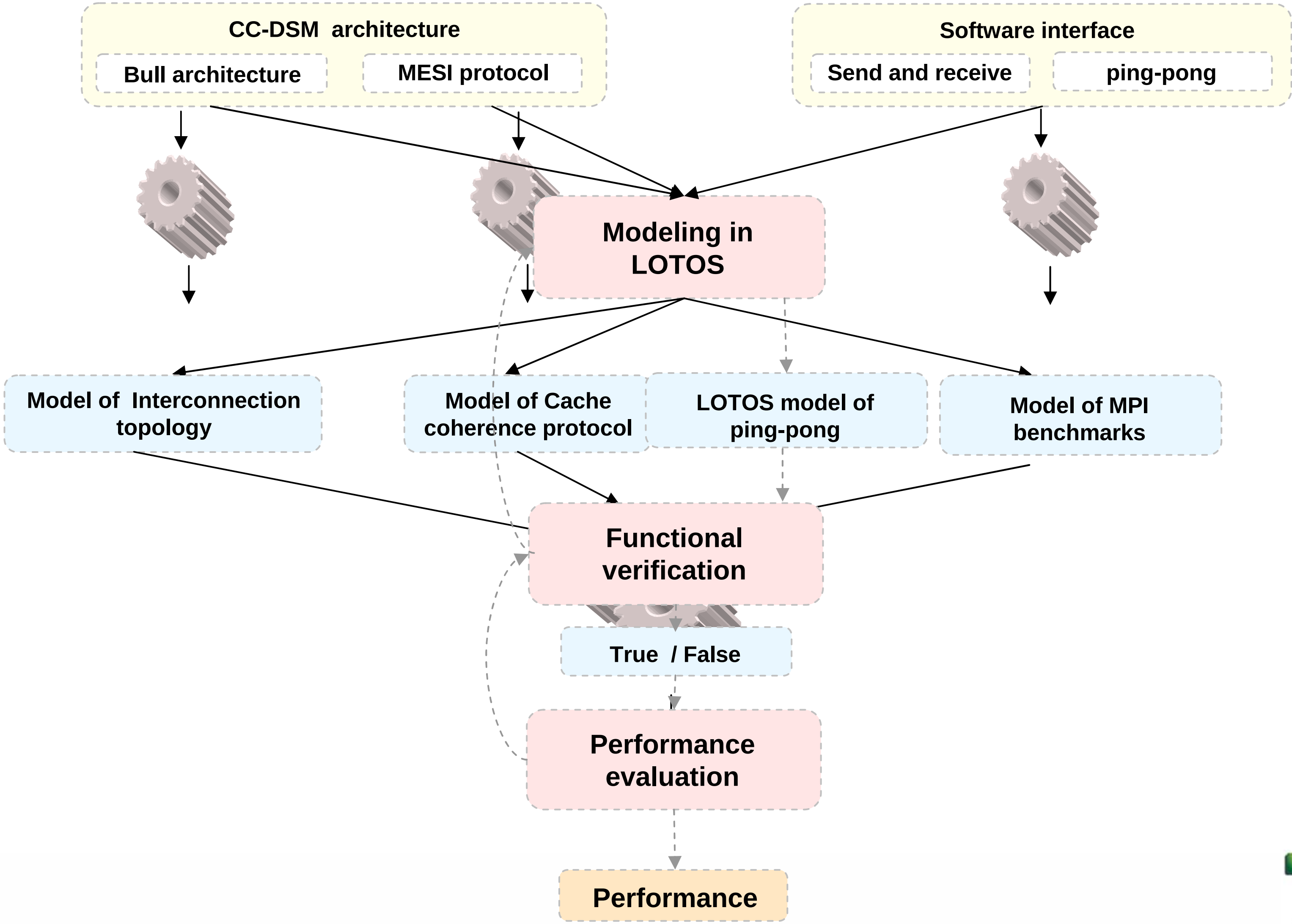
Performance evaluation: results

- **Latency** = 1/(2 *Throughput (START))
- **Throughput** (VAR): frequency of transitions corresponding to misses made on the variable VAR
- **Nb_Misses** (VAR): number of misses of the variable VAR during the **Latency** period

- **Nb_Misses** (VAR)= Latency *Throughput (VAR)

	Number of misses									
	Primitives SR1						Primitives SR2			
	Protocol A			Protocol B			Protocol A		Protocol 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
Model of Interconnection topology
Model of Cache coherence protocol
LOTOS model of ping-pong
Model of MPI benchmarks
Functional verification
True / False
Performance evaluation
Performance

Conclusion

- Modeling in LOTOS:
  - send and receive primitives
  - cache coherence protocol
  - interconnection topology
- Functional verification of the ping-pong model
- Performance evaluation of the ping-pong model:
  - Consistency of the obtained results
  - The obtained results are comforted by the experimental measures
  - Comparison of latencies of 2 MPI primitives in 3 different topologies and 2 different cache coherency protocols

Perspectives

- Current work …
  - Performance evaluation of *barriers* primitives
- ... Ongoing work
  - Automation of the proposed method
  - Taking into account the different phases of transfers in the protocol cache coherence model
  - Generalization of the method
  - ...