



Process placement for unbalanced architectures

Joint work with: Guillaume Mercier, François Tessier,
Charm++ team, Torsten Hoefler

Emmanuel Jeannot
Runtime Team
Inria Bordeaux Sud-Ouest

November 21st, 2012

MPI (Process-based runtime systems)

Performance of MPI programs depends on many factors that can be handled when you change the machine:

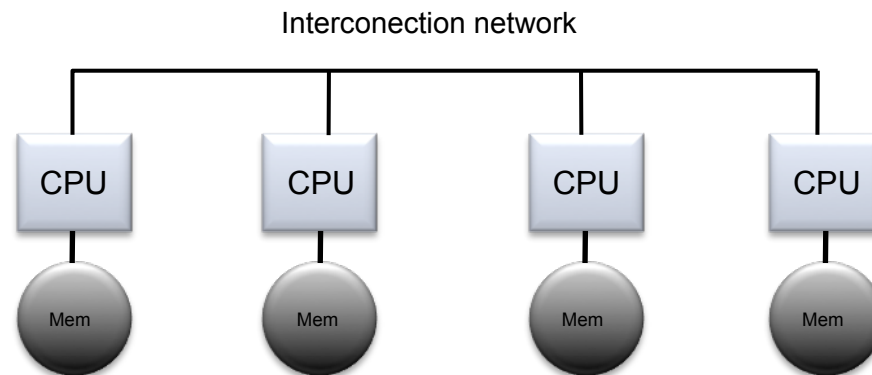
- Implementation of the standard (e.g. collective com.)
- Parallel algorithm(s)
- Implementation of the algorithm
- Underlying libraries (e.g. BLAS)
- Hardware (processors, cache, network)
- etc.

But...

Process Placement

The MPI model makes little (no?) assumption on the way processes are mapped to resources

It is often assume that the network **topology is flat** and hence the process mapping has little impact on the performance



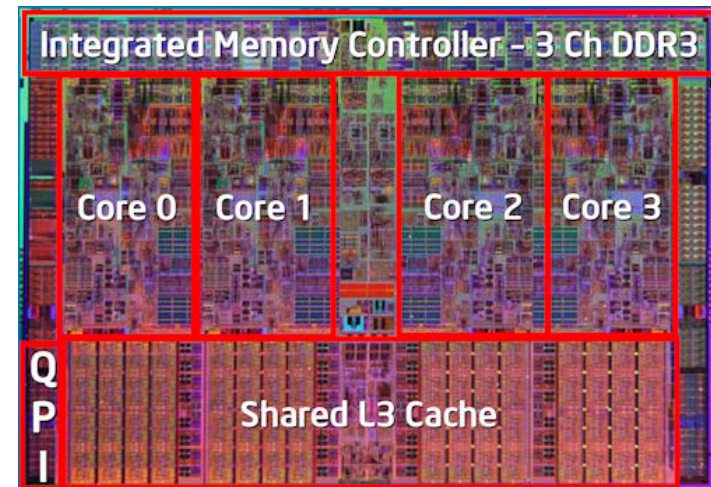
The Topology is not Flat

Due to multicore processors current and future parallel machines are hierarchical

Communication speed depends on:

- Receptor and emitter
- Cache hierarchy
- Memory bus
- Interconnection network

etc.



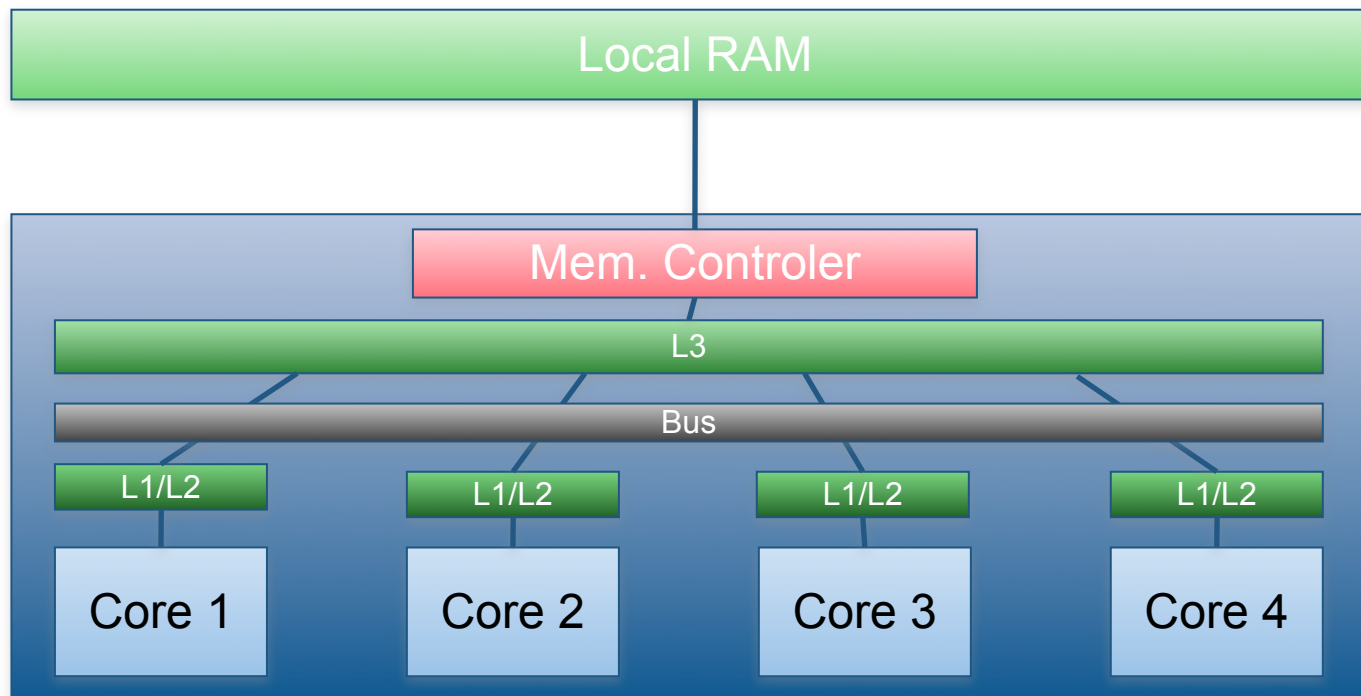
Almost nothing in the MPI standard help to handle these factors

Example on a Parallel Machine

The higher we have to go into the hierarchy the costly the data exchange

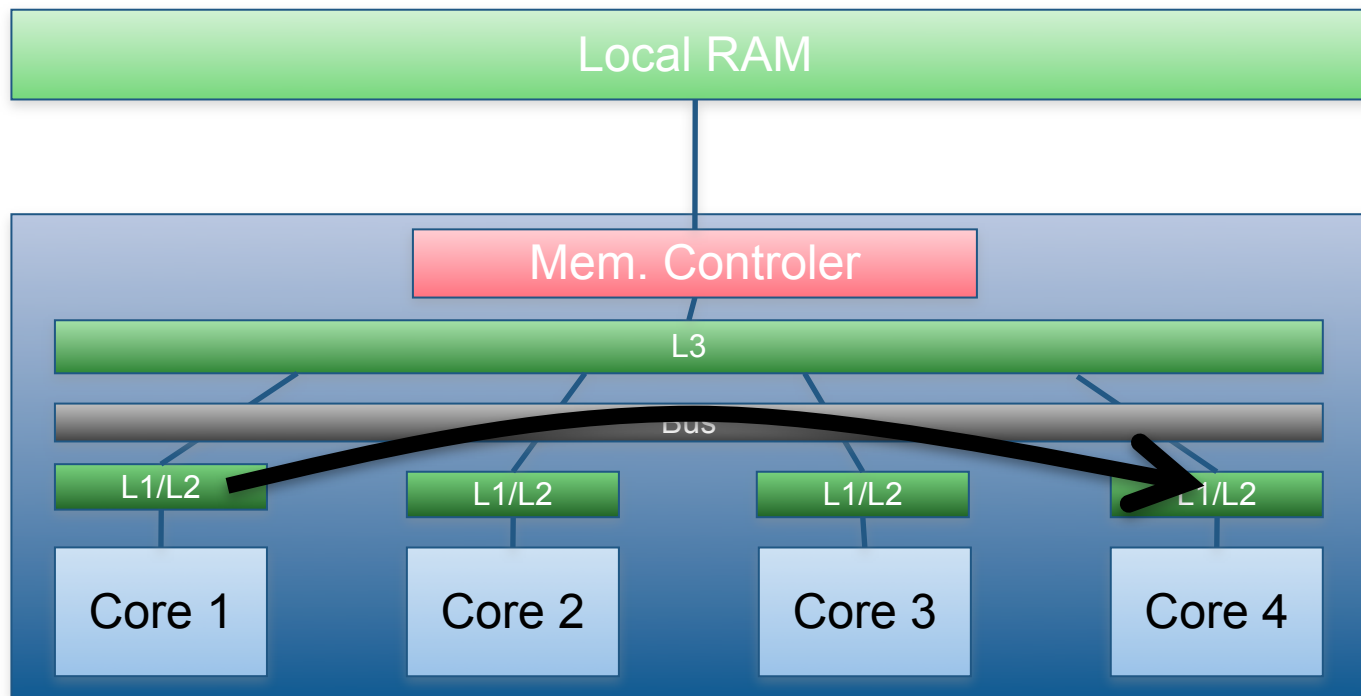
Example on a Parallel Machine

The higher we have to go into the hierarchy the costly the data exchange



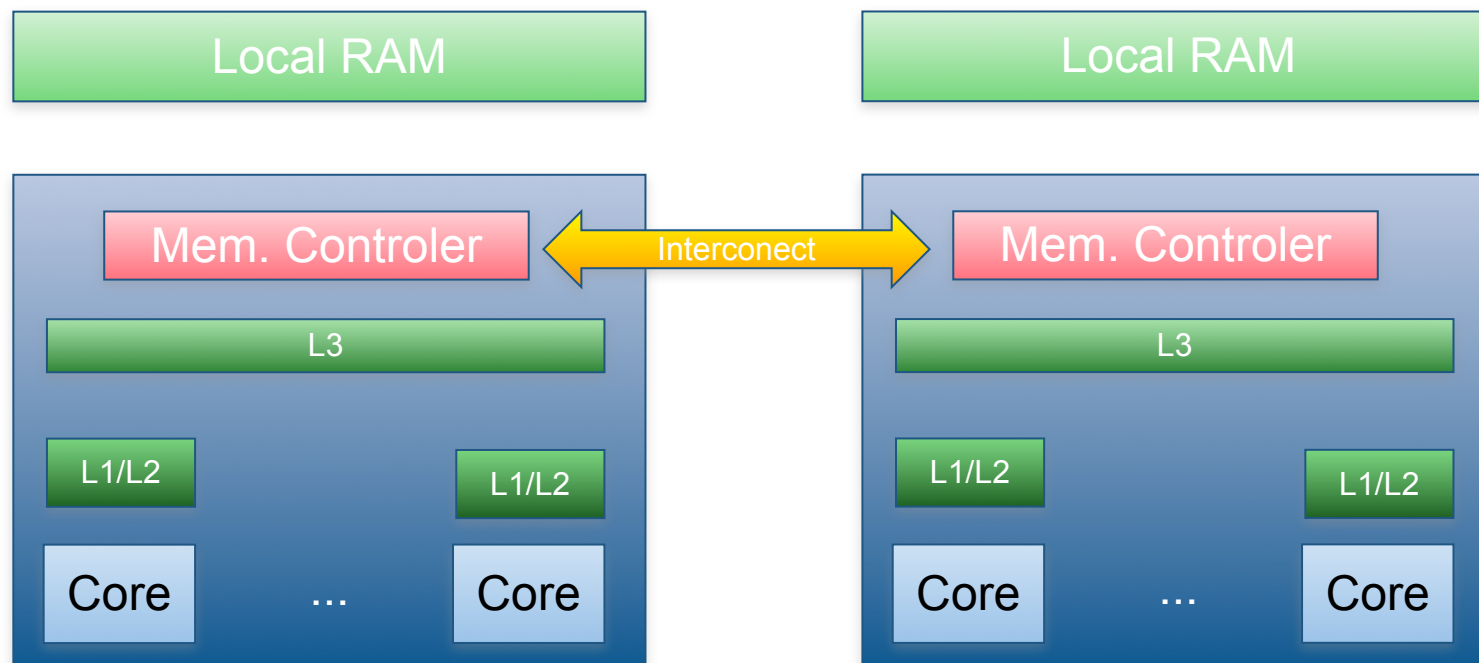
Example on a Parallel Machine

The higher we have to go into the hierarchy the costly the data exchange



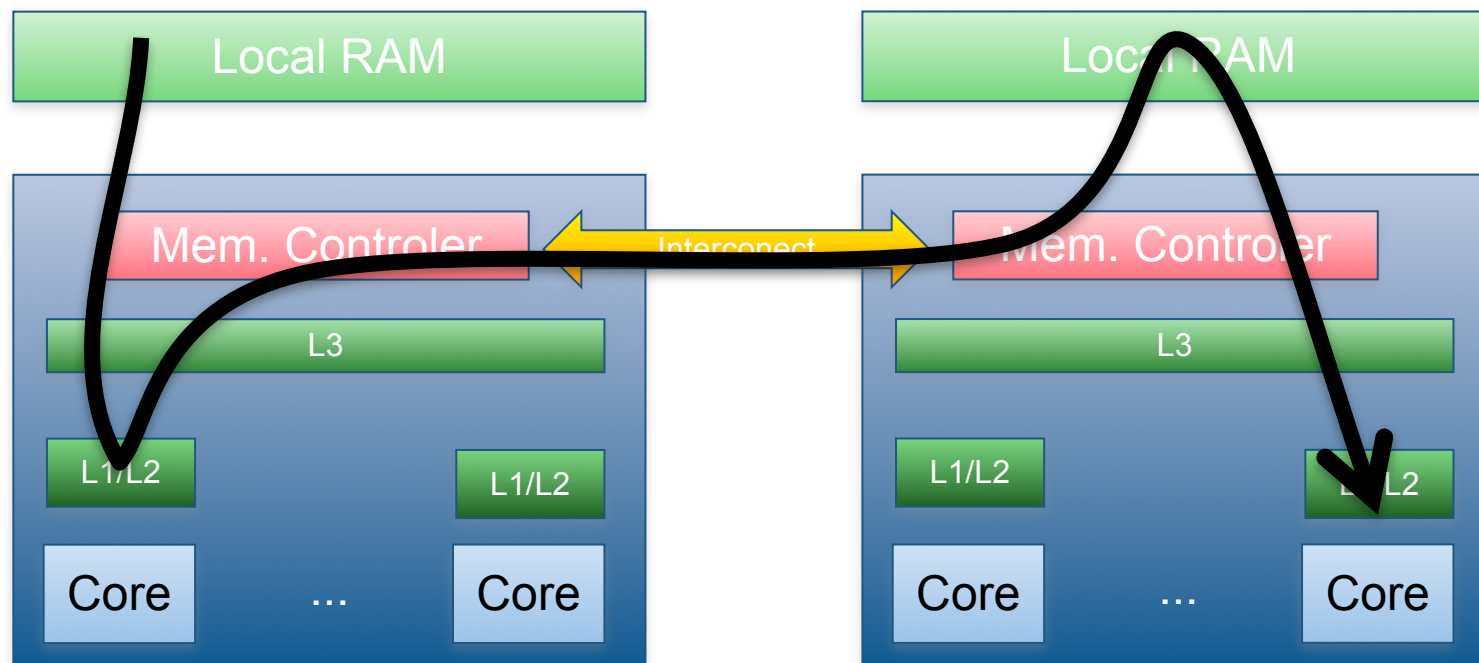
Example on a Parallel Machine

The higher we have to go into the hierarchy the costly the data exchange



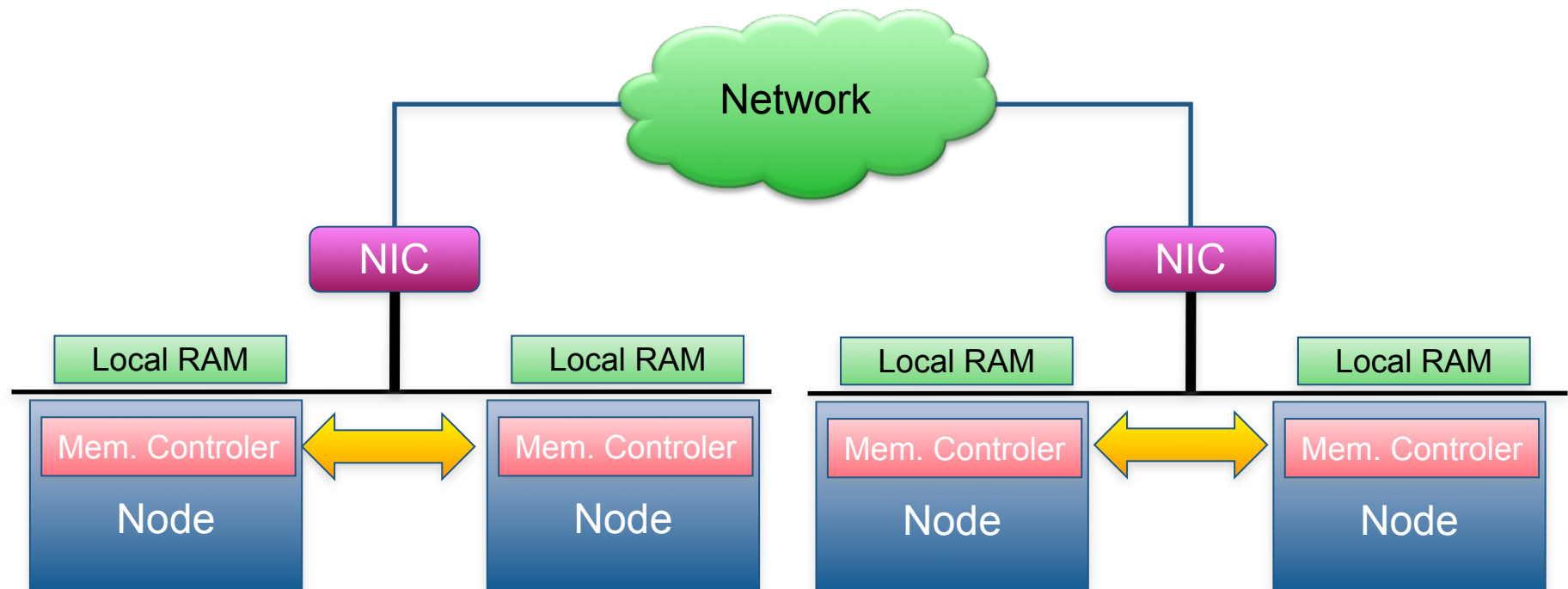
Example on a Parallel Machine

The higher we have to go into the hierarchy the costly the data exchange



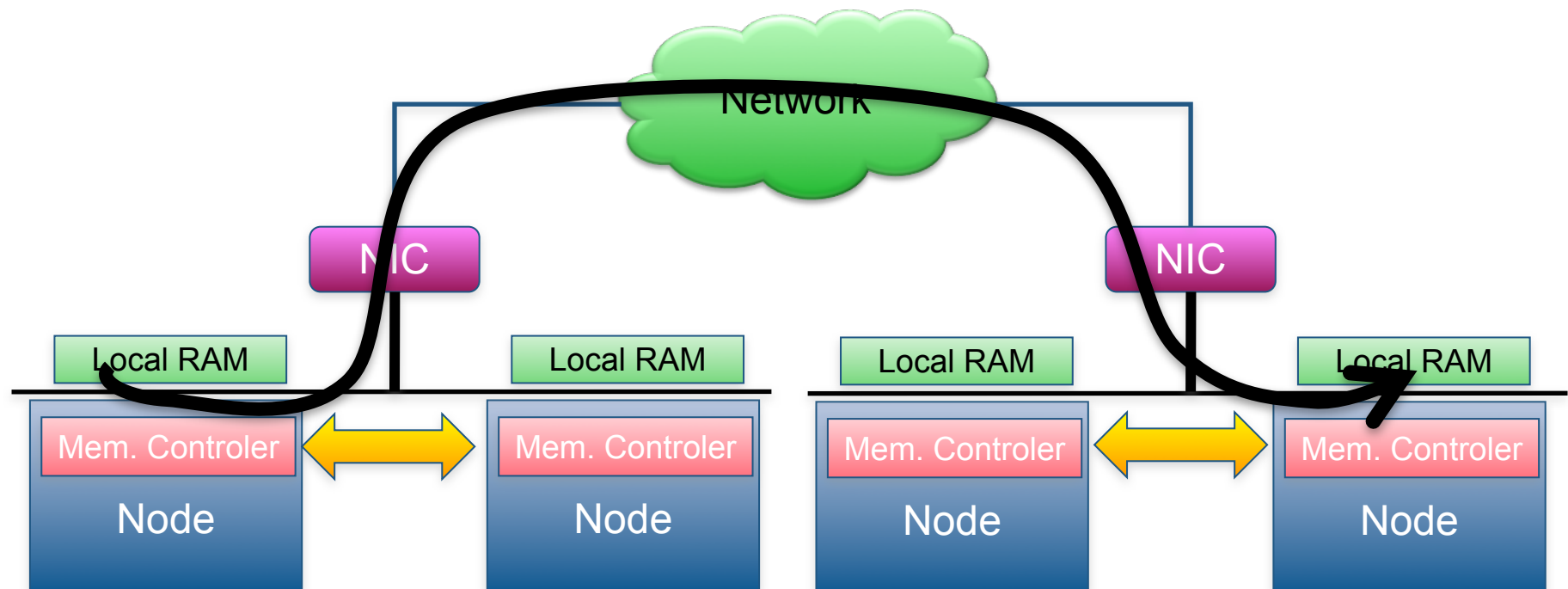
Example on a Parallel Machine

The higher we have to go into the hierarchy the costly the data exchange



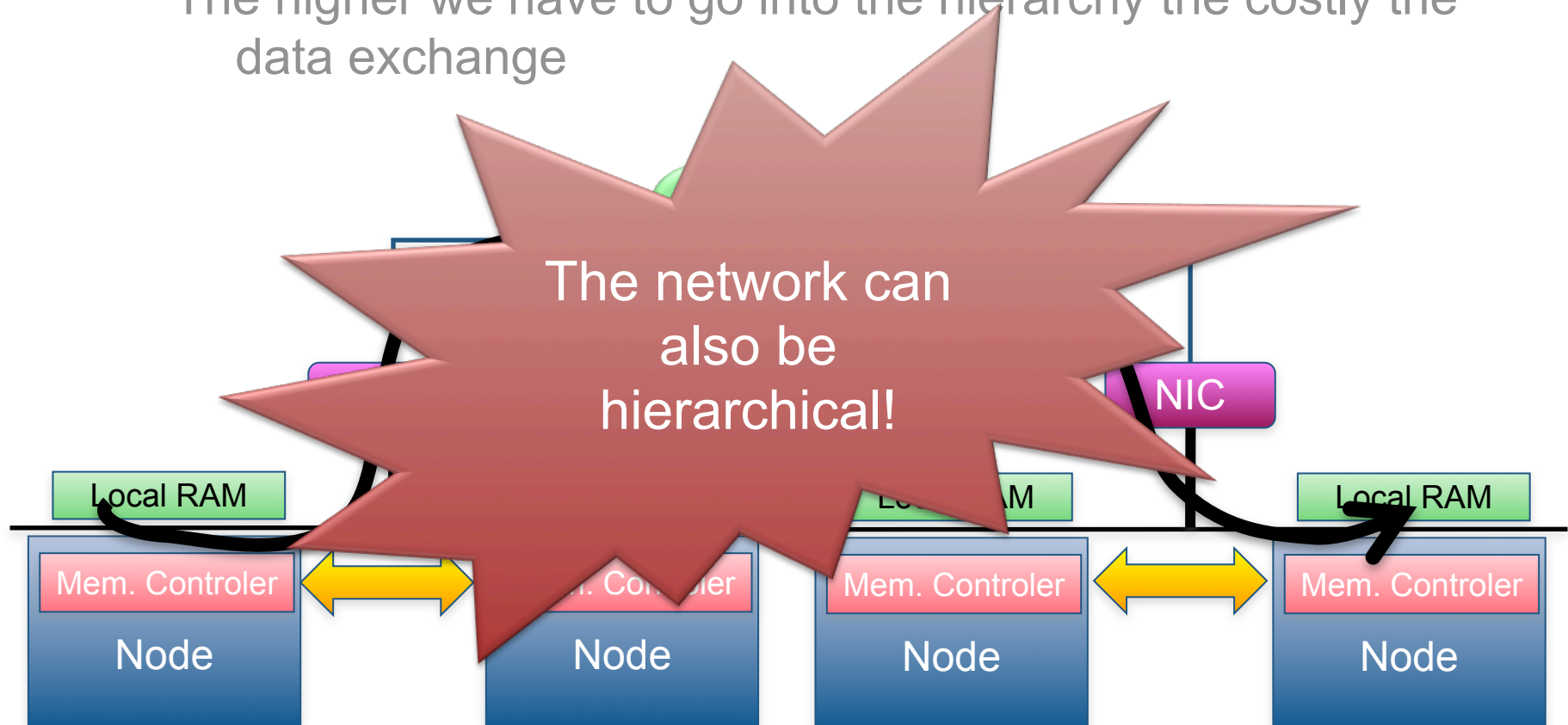
Example on a Parallel Machine

The higher we have to go into the hierarchy the costly the data exchange



Example on a Parallel Machine

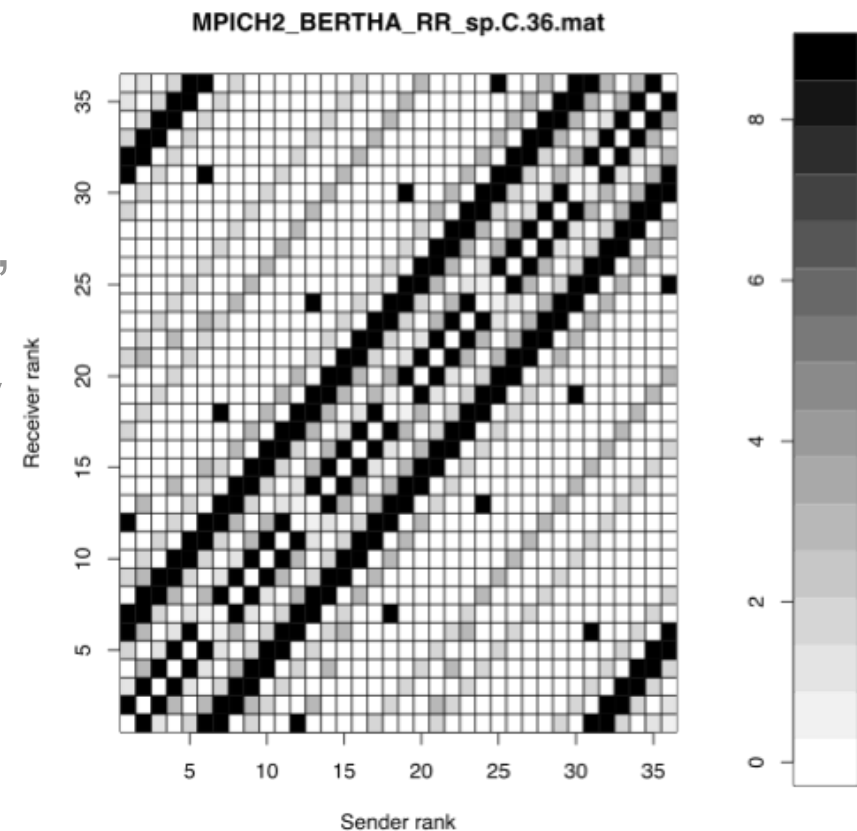
The higher we have to go into the hierarchy the costly the data exchange



Rationale

Not all the processes exchange the same amount of data

The speed of the communications, and hence performance of the application depends on the way processes are mapped to resources.



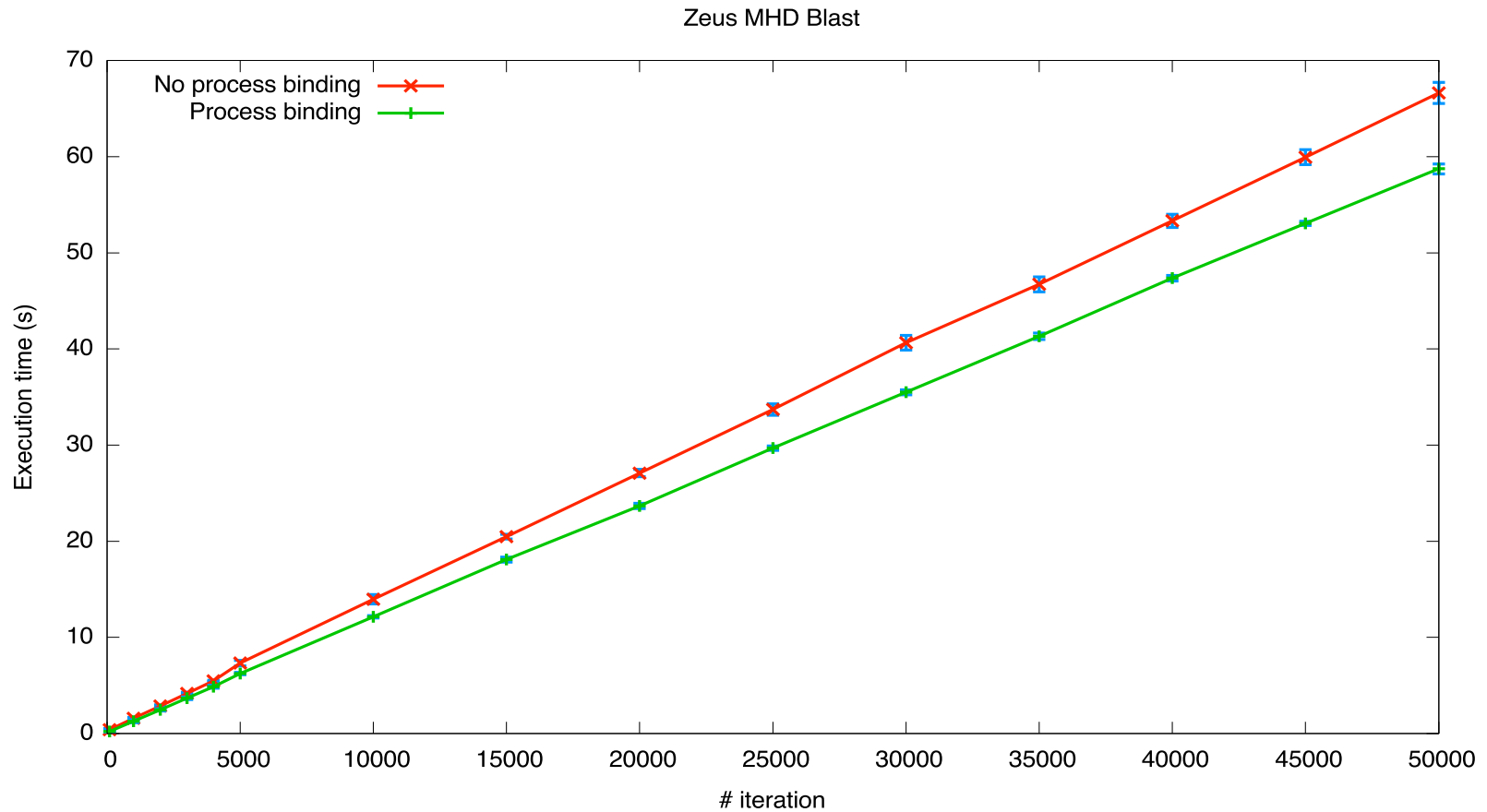
Do we Really Care: to Bind or not to Bind?

After all, the system scheduler is able to move processes when needed.

Yes, but only for shared memory system. Migration is possible but it is not in the MPI standard (see charm++)

Moreover binding provides better execution runtime stability.

Do we Really Care: to Bind or not to Bind?



Zeus MHD Blast. 64 Processes/Cores. Mvapich2 1.8. + ICC

Process Placement Problem

Given :

- Parallel machine **topology**
- Process **affinity** (communication pattern)

Map processes to resources (cores) to reduce communication cost: a nice algorithmic problem:

- Graph partitionning (Scotch, Metis)
- Application tuning [Aktulga et al. Euro-Par 12]
- Topology-to-pattern matching (TreeMatch)

Reduce Communication Cost?

But wait, my application is compute-bound!

Well, but this might not be still true in the future: **strong scaling** might not always be a solution.

Reduce Communication Cost?

Systems	2010	2018	Difference Today & 2018
System peak	2 Pflop/s	1 Eflop/s	O(1000)
Power	6 MW	~20 MW	
System memory	0.3 PB	32 - 64 PB [.03 Bytes/Flop]	O(100)
Node performance	125 GF	1,2 or 15TF	O(10) - O(100)
Node memory BW	25 GB/s	2 - 4TB/s [.002 Bytes/Flop]	O(100)
Node concurrency	12	O(1k) or 10k	O(100) - O(1000)
Total Node Interconnect BW	3.5 GB/s	200-400GB/s (1:4 or 1:8 from memory BW)	O(100)
System size (nodes)	18,700	O(100,000) or O(1M)	O(10) - O(100)
Total concurrency	225,000	O(billion) [O(10) to O(100) for latency hiding]	O(10,000)
Storage	15 PB	500-1000 PB (>10x system memory is min)	O(10) - O(100)
IO	0.2 TB	60 TB/s (how long to drain the machine)	O(100)
MTTI	days	O(1 day)	- O(10)

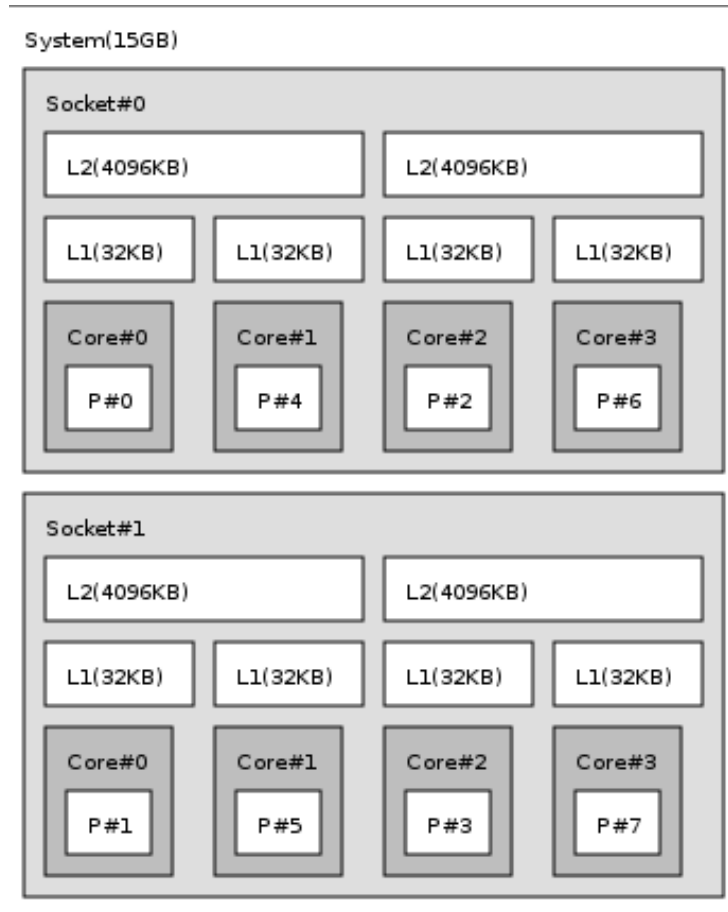
Taken from one of J. Dongarra's Talk.

Obtaining the Topology (Shared Memory)

HWLOC (portable hardware locality):

- Runtime and OpenMPI team
- Portable abstraction (across OS, versions, architectures, ...)
- Hierarchical topology
- Modern architecture (NUMA, cores, caches, etc.)
- ID of the cores
- C library to play with
- Etc

HWLOC



<http://www.open-mpi.org/projects/hwloc/>

Obtaining the Topology (Distributed Memory)

Not always easy (research issue)

MPI core has some routine to get that

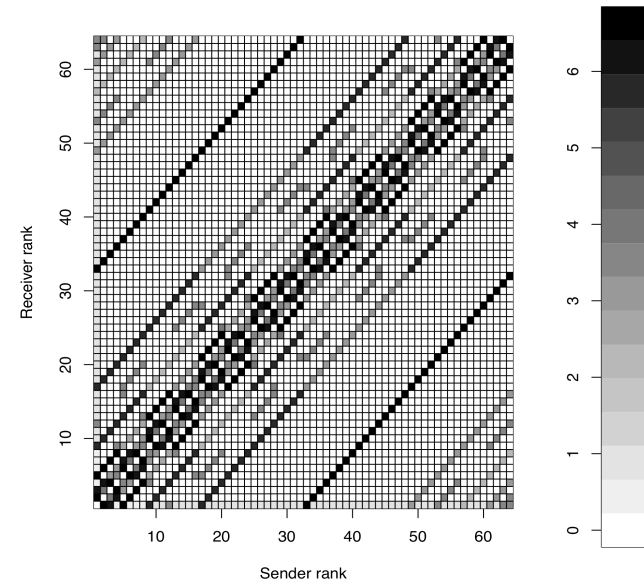
Sometime requires to build a file that specifies node adjacency

Getting the Communication Pattern

No automatic way so far...

Can be done through application monitoring:

- During execution
- With a « blank execution »



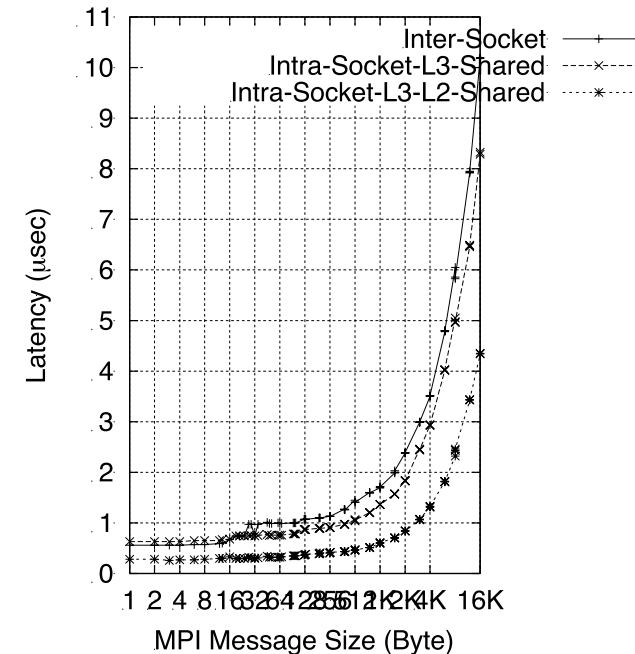
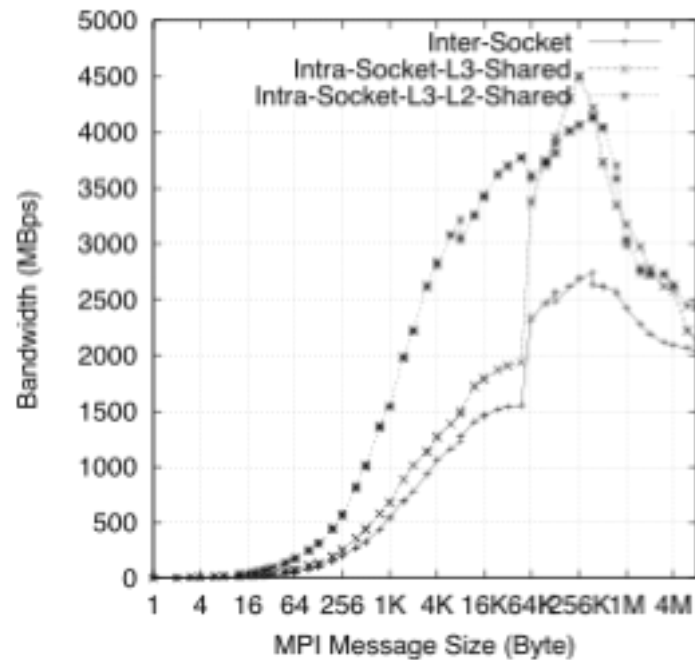
State of the Art

Process placement fairly well studied problem:

- [Träff 02]: placement through graph embedding and graph partitioning
- MPIPP [Chen et al. 2006]: placement through local exchange of processes until no gain is achievable
- [Clet-Ortega & Mercier 09]: placement through graph renumbering (Scotch)
- LibTopoMap [Hoefler & Snir 11]: placement through network model + graph partitioning (ParMetis)

Problems with quantitative knowledge

Netpipe on a 2 sockets, 8 cores each NUMA node



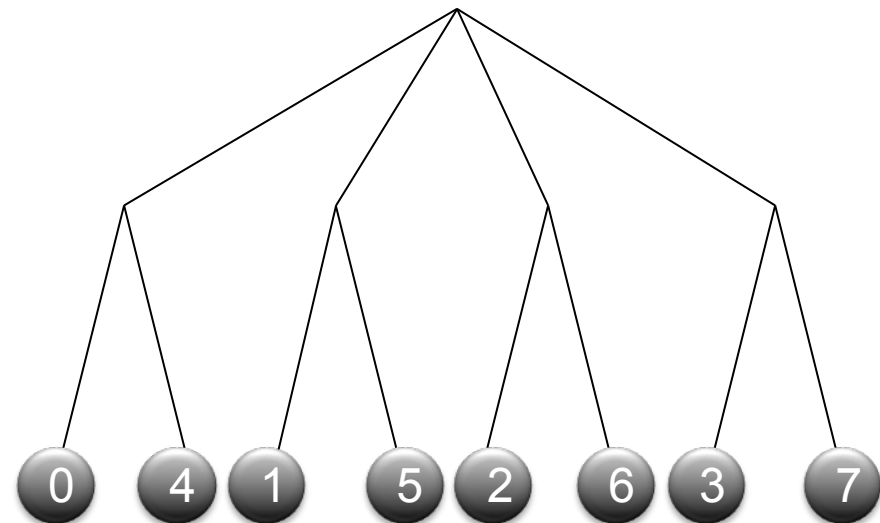
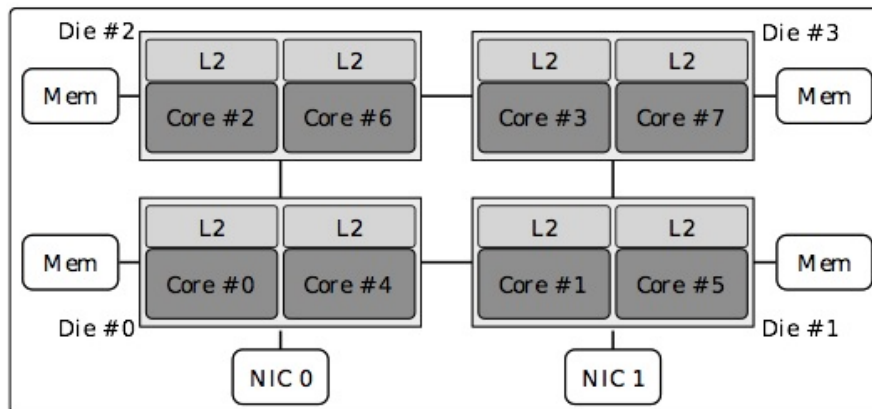
It is generally faster to use cache or to stay within a socket
But the acceleration ratio depends on message size:

- It is not linear (not affine either)
- Contention makes things even harder

Dealing with Qualitative Knowledge

Abstract the topology with a tree

Assume communication always cost more when you need to reach higher levels

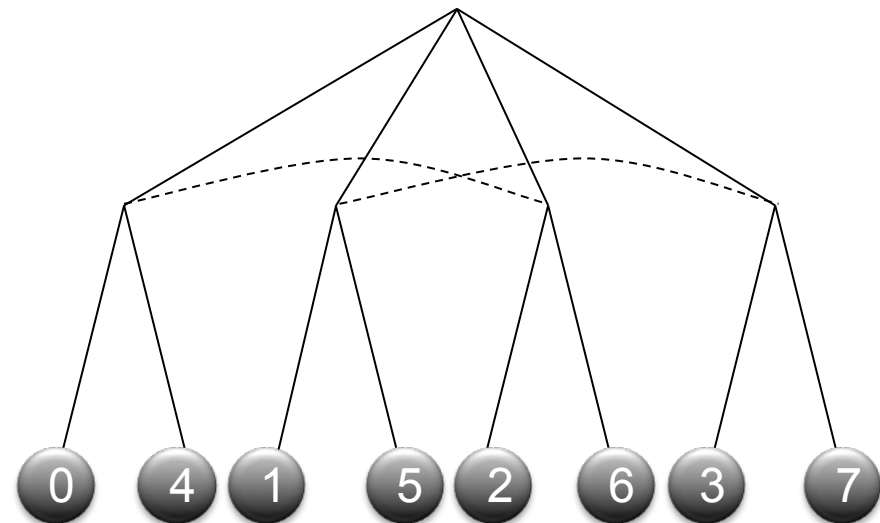
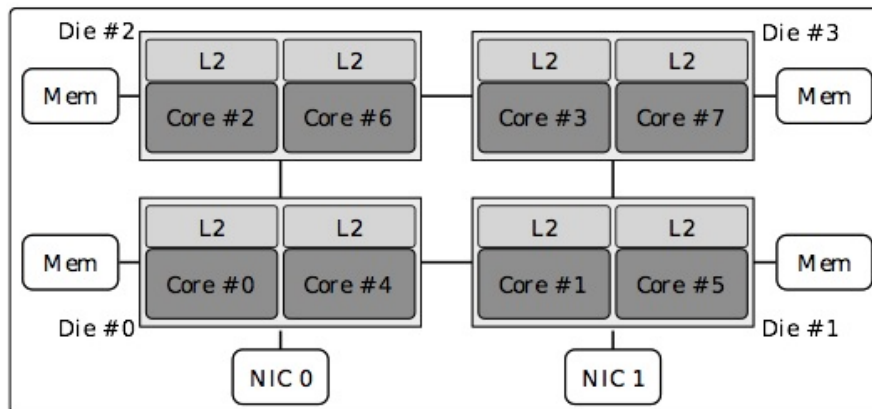


The structure is sufficient. No need to deal with latency or bandwidth.

Dealing with Qualitative Knowledge

Abstract the topology with a tree

Assume communication always cost more when you need to reach higher levels



The structure is sufficient. No need to deal with latency or bandwidth.

TreeMatch Algorithm

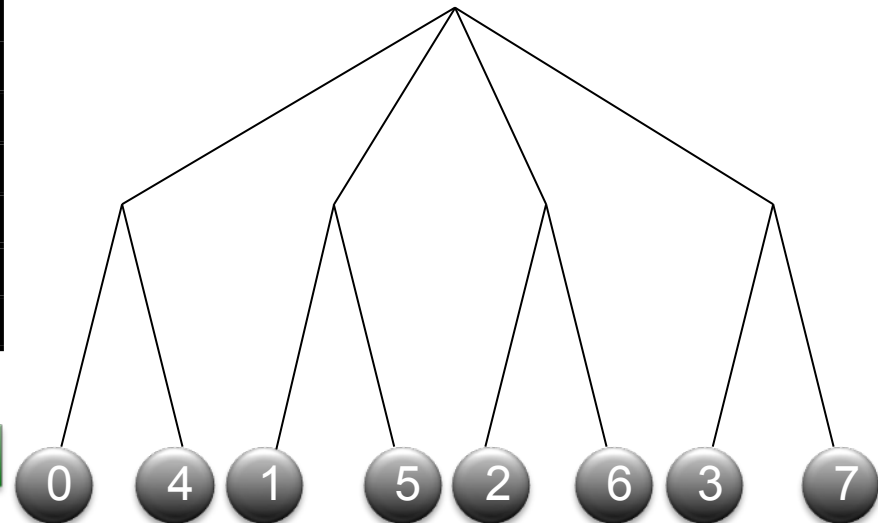
C: communication matrix

0	1000	10	1	100	1	1	1
1000	0	1000	1	1	100	1	1
10	1000	0	1000	1	1	100	1
1	1	1000	0	1	1	1	100
100	1	1	1	0	1000	10	1
1	100	1	1	1000	0	1000	1
1	1	100	1	10	1000	0	1000
1	1	1	100	1	1	1000	0



Grouped matrix

0	101	202	4
101	0	4	202
202	4	0	101
4	202	101	0



TreeMatch Algorithm

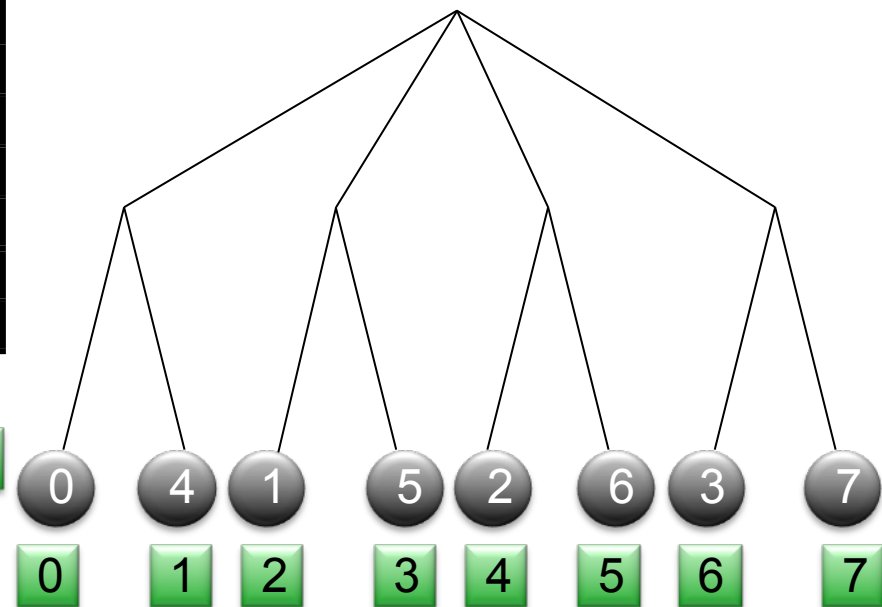
C: communication matrix

0	1000	10	1	100	1	1	1
1000	0	1000	1	1	100	1	1
10	1000	0	1000	1	1	100	1
1	1	1000	0	1	1	1	100
100	1	1	1	0	1000	10	1
1	100	1	1	1000	0	1000	1
1	1	100	1	10	1000	0	1000
1	1	1	100	1	1	1000	0



Grouped matrix

0	101	202	4
101	0	4	202
202	4	0	101
4	202	101	0



Communication matrix + Tree Topology
= Process permutation

Process reordering

TreeMatch: process permutation

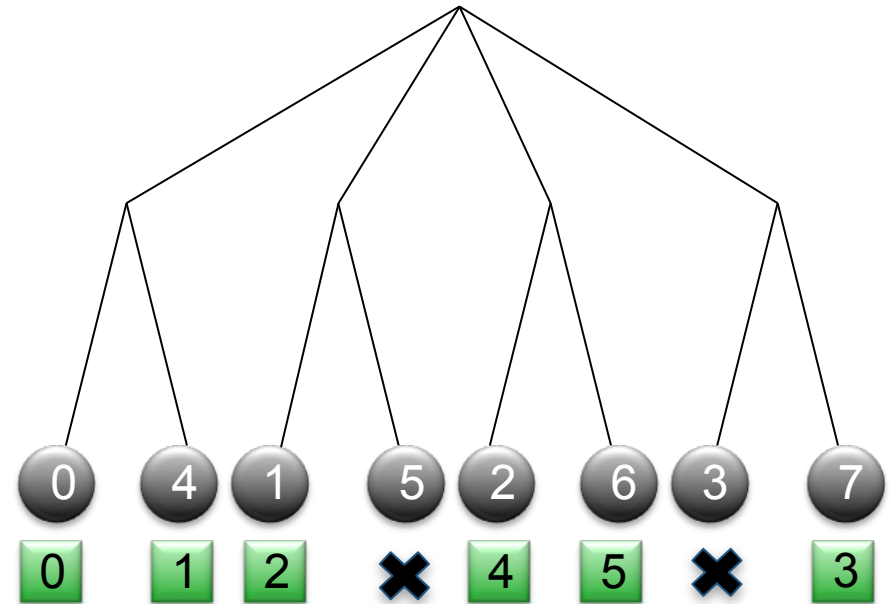
If process already bound: **rank reordering**.

Problem: how to take into account placement constraint

Dealing with already mapped applications

Problem:

- Given a hierarchical topology
- An already mapped application onto a subset of the node
- Reorder process while ensuring only this subset is used



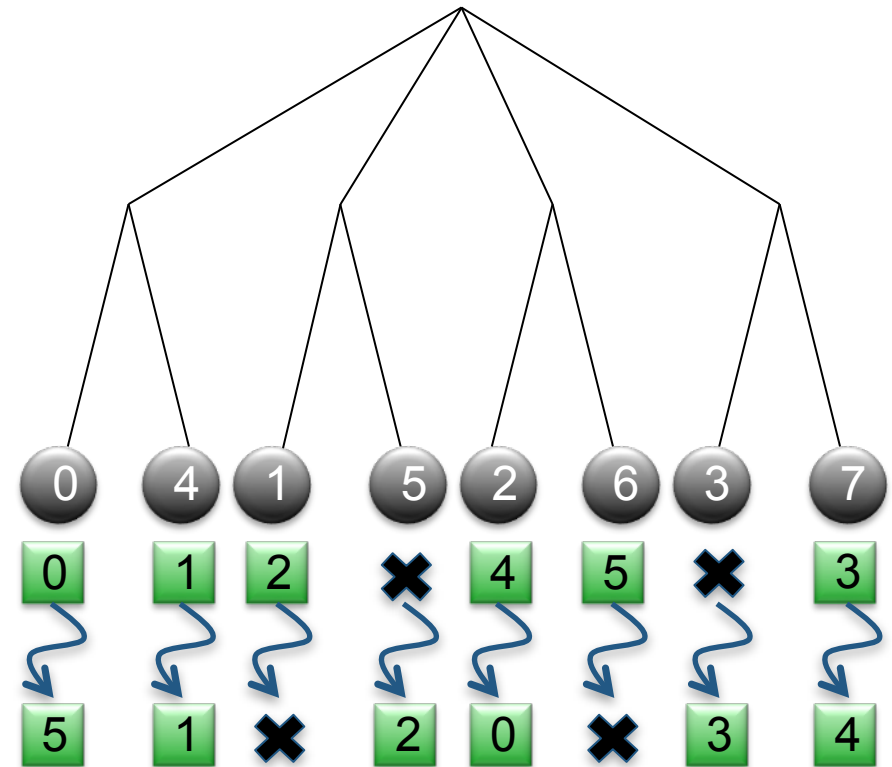
Requires to change the TreeMatch algorithm to prevent it to use some nodes

Dealing with already mapped applications

Problem:

- Given a hierarchical topology
- An already mapped application onto a subset of the node
- Reorder process while ensuring only this subset is used

Requires to change the TreeMatch algorithm to prevent it to use some nodes

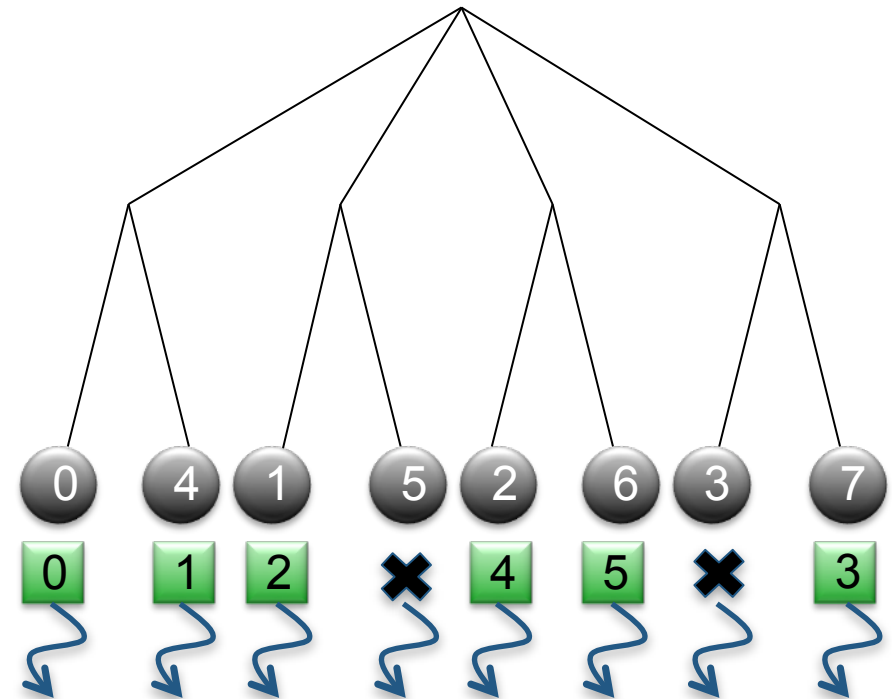


Dealing with already mapped applications

Problem:

- Given a hierarchical topology
- An already mapped application onto a subset of the node
- Reorder process while ensuring only this subset is used

Requires to change the TreeMatch algorithm to prevent it to use some nodes

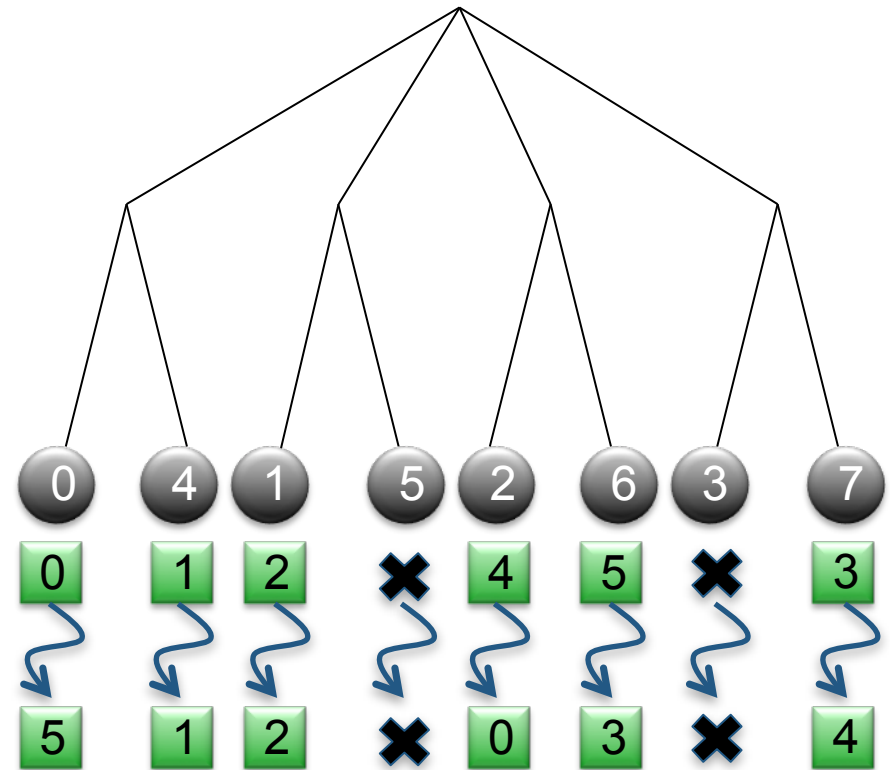


Dealing with already mapped applications

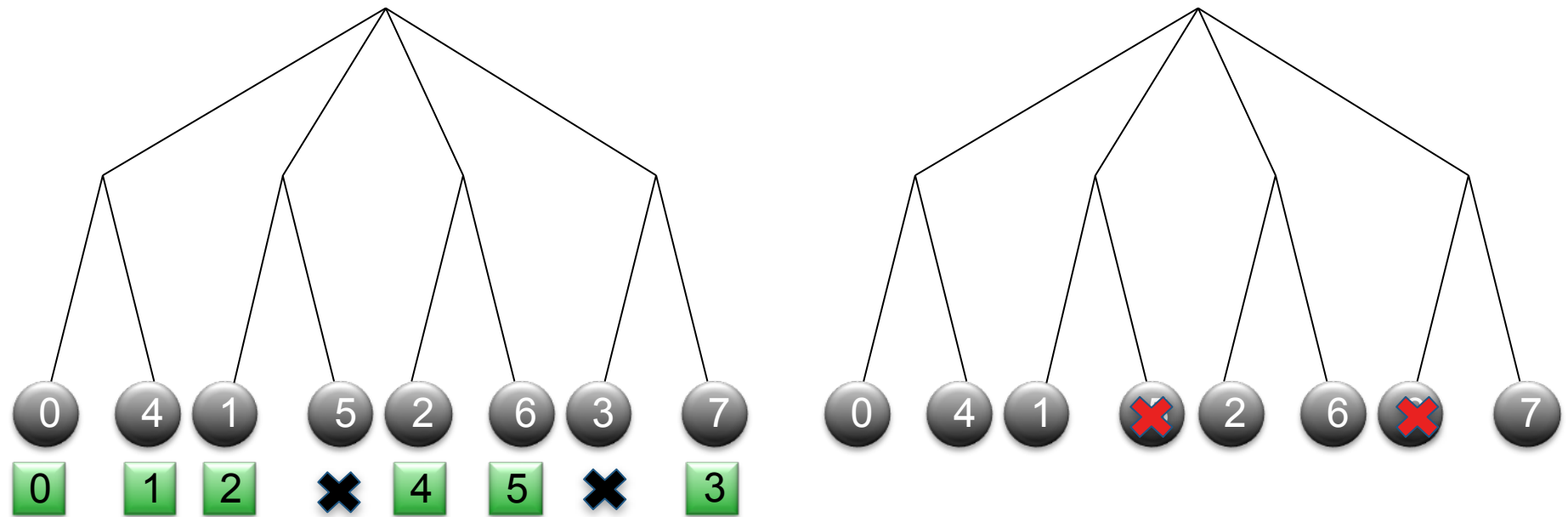
Problem:

- Given a hierarchical topology
- An already mapped application onto a subset of the node
- Reorder process while ensuring only this subset is used

Requires to change the TreeMatch algorithm to prevent it to use some nodes



New Version of TreeMatch to Deal With Unbalanced Tree



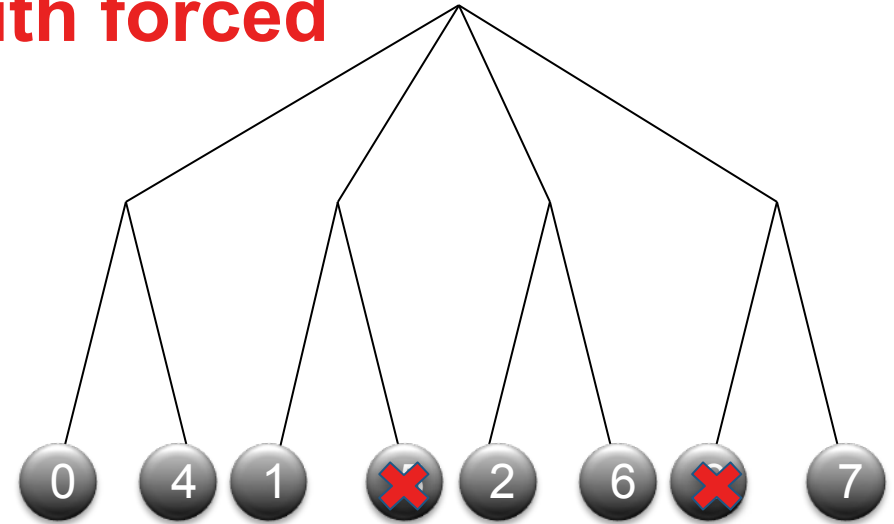
Solution:

- Extend the communication matrix with dummy nodes
- Process the tree backward by doing k-partitioning
- Force each partition to have the right number of dummy nodes
- Process recursively

Example: bi-partitionning with forced mapping

0	5	7	9	10	2
5	0	7	4	9	1
7	7	0	1	9	2
9	4	1	0	1	7
10	9	9	1	0	10
2	1	2	7	10	0

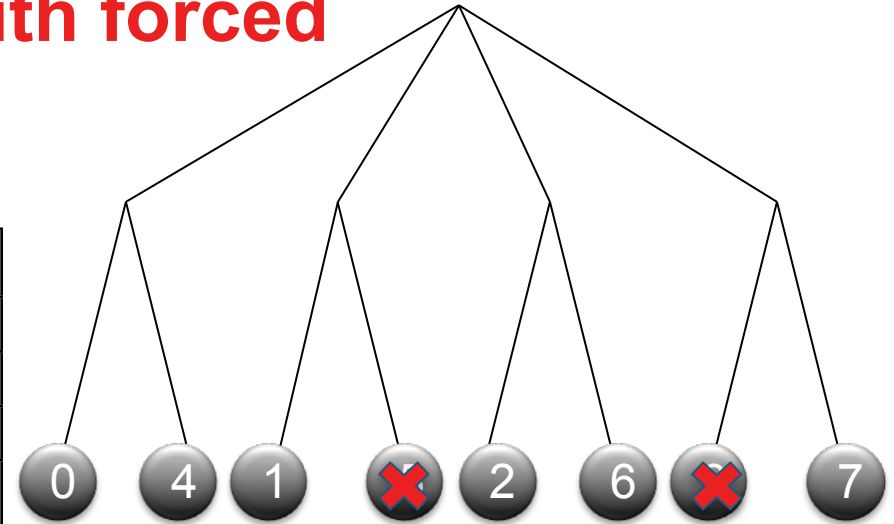
0 1 2 3 4 5



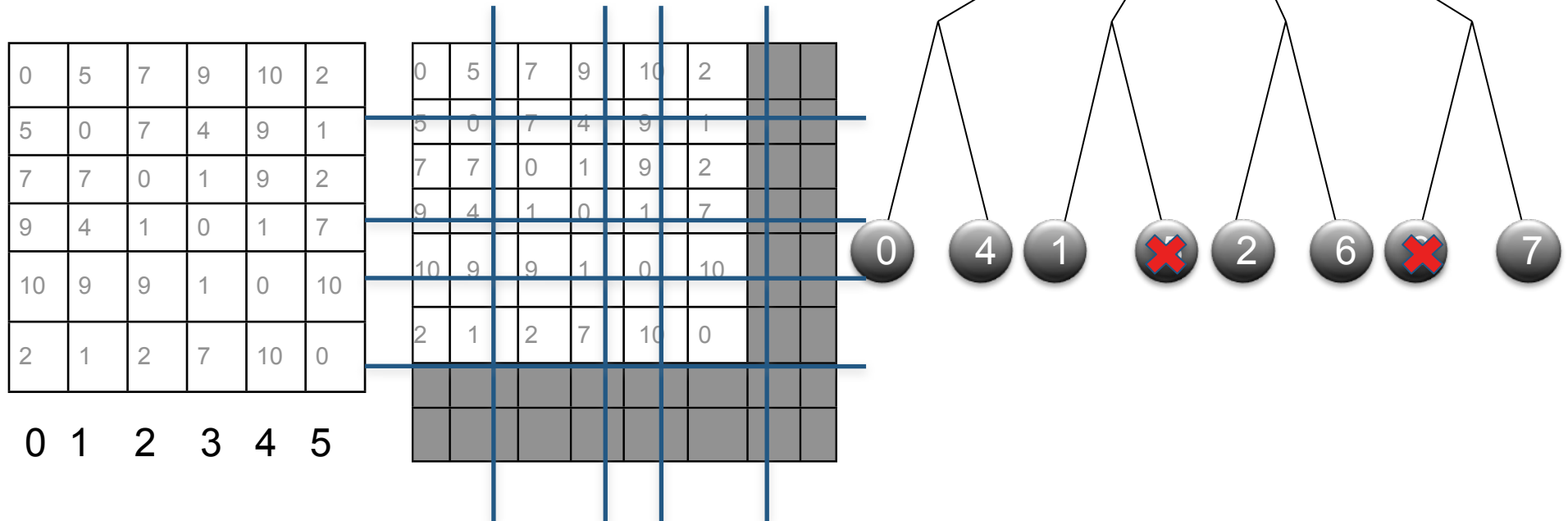
Example: bi-partitioning with forced mapping

0	5	7	9	10	2
5	0	7	4	9	1
7	7	0	1	9	2
9	4	1	0	1	7
10	9	9	1	0	10
2	1	2	7	10	0
0	1	2	3	4	5

0	5	7	9	10	2		
5	0	7	4	9	1		
7	7	0	1	9	2		
9	4	1	0	1	7		
10	9	9	1	0	10		
2	1	2	7	10	0		



Example: bi-partitioning with forced mapping

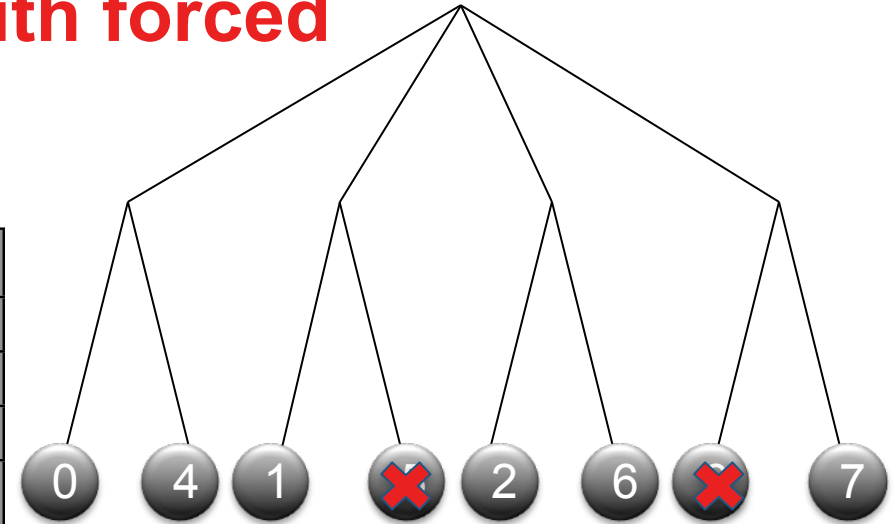


0	4	9	
4	0	4	
9	4	0	

Example: bi-partitioning with forced mapping

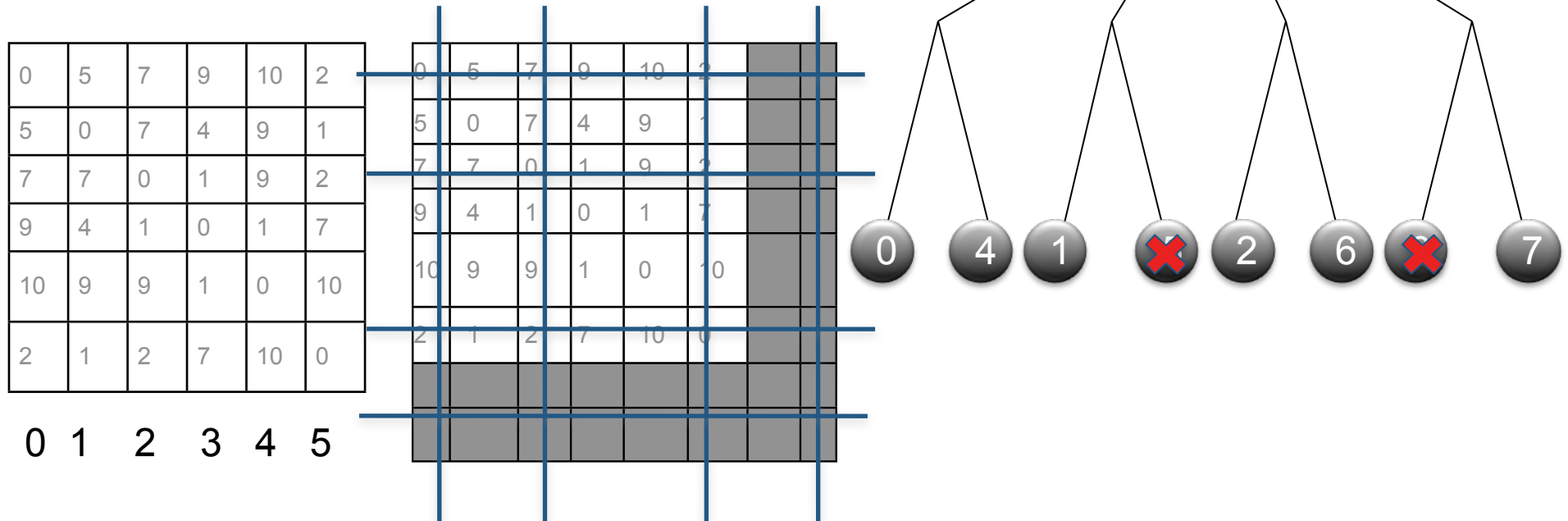
0	5	7	9	10	2
5	0	7	4	9	1
7	7	0	1	9	2
9	4	1	0	1	7
10	9	9	1	0	10
2	1	2	7	10	0
0	1	2	3	4	5

0	5	7	9	10	2		
5	0	7	4	9	1		
7	7	0	1	9	2		
9	4	1	0	1	7		
10	9	9	1	0	10		
2	1	2	7	10	0		



0	4	9	
4	0	4	
9	4	0	

Example: bi-partitioning with forced mapping



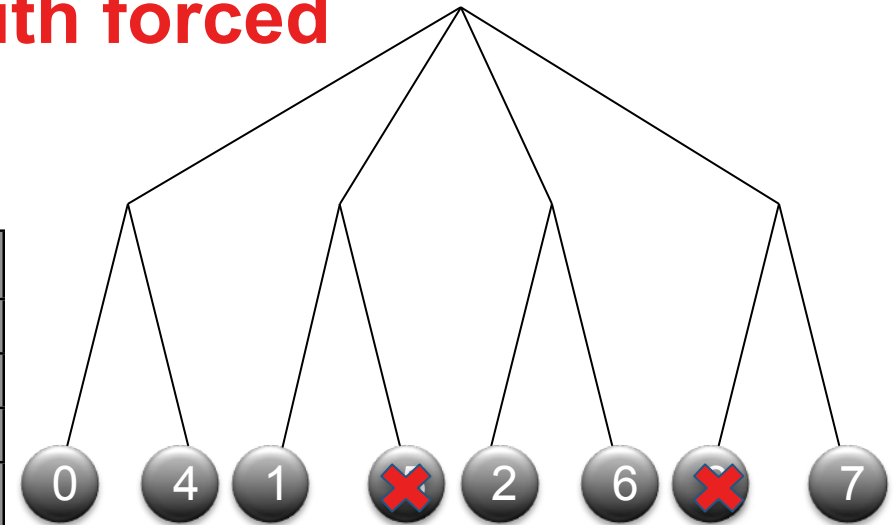
0	4	9	
4	0	4	
9	4	0	

0	7	2	
7	0	2	
2	2	0	

Example: bi-partitioning with forced mapping

0	5	7	9	10	2
5	0	7	4	9	1
7	7	0	1	9	2
9	4	1	0	1	7
10	9	9	1	0	10
2	1	2	7	10	0
0	1	2	3	4	5

0	5	7	9	10	2		
5	0	7	4	9	1		
7	7	0	1	9	2		
9	4	1	0	1	7		
10	9	9	1	0	10		
2	1	2	7	10	0		



0	4	9	
4	0	4	
9	4	0	

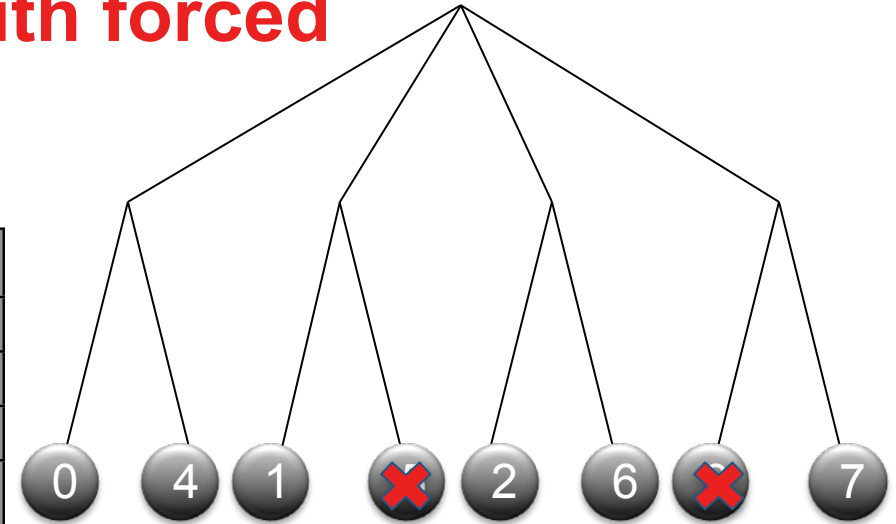
0	7	2	
7	0	2	
2	2	0	

Example: bi-partitioning with forced mapping

0	5	7	9	10	2
5	0	7	4	9	1
7	7	0	1	9	2
9	4	1	0	1	7
10	9	9	1	0	10
2	1	2	7	10	0

0 1 2 3 4 5

0	5	7	9	10	2		
5	0	7	4	9	1		
7	7	0	1	9	2		
9	4	1	0	1	7		
10	9	9	1	0	10		
2	1	2	7	10	0		



0	7	2	
7	0	2	
2	2	0	

0	4	9	
4	0	4	
9	4	0	

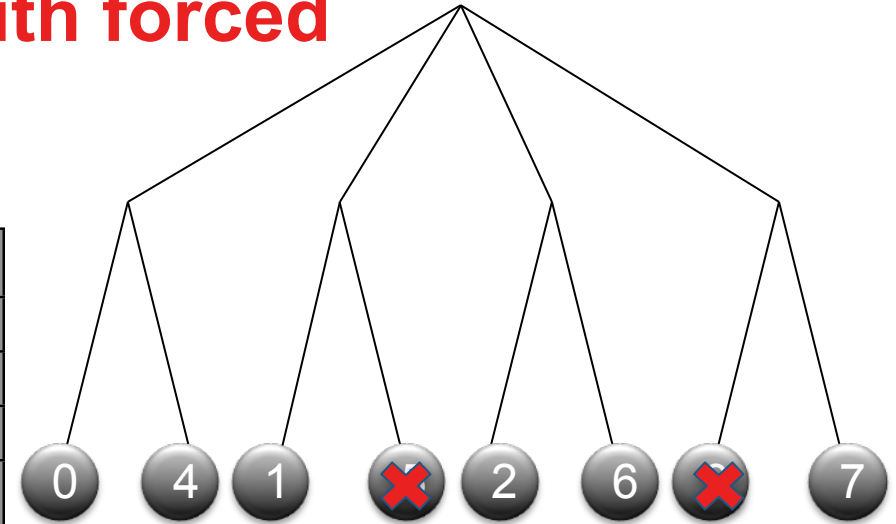
0	7	2	
7	0	2	
2	2	0	

Example: bi-partitioning with forced mapping

0	5	7	9	10	2
5	0	7	4	9	1
7	7	0	1	9	2
9	4	1	0	1	7
10	9	9	1	0	10
2	1	2	7	10	0

0 1 2 3 4 5

0	5	7	9	10	2		
5	0	7	4	9	1		
7	7	0	1	9	2		
9	4	1	0	1	7		
10	9	9	1	0	10		
2	1	2	7	10	0		



0	7	2	
7	0	2	
2	2	0	

0	4	9	
4	0	4	
9	4	0	

0	7	2	
7	0	2	
2	2	0	

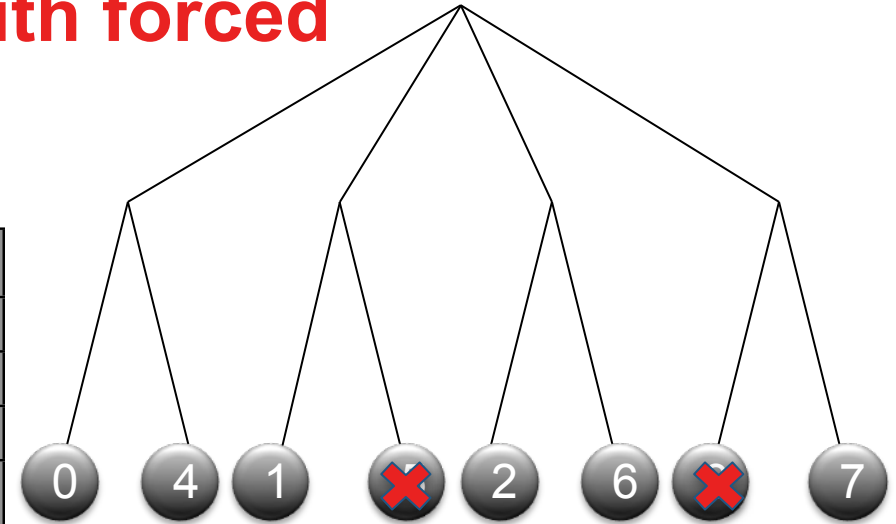
0	2
2	0

Example: bi-partitioning with forced mapping

0	5	7	9	10	2
5	0	7	4	9	1
7	7	0	1	9	2
9	4	1	0	1	7
10	9	9	1	0	10
2	1	2	7	10	0

0 1 2 3 4 5

0	5	7	9	10	2		
5	0	7	4	9	1		
7	7	0	1	9	2		
9	4	1	0	1	7		
10	9	9	1	0	10		
2	1	2	7	10	0		



0	7	2	
7	0	2	
2	2	0	

0	4	9	
4	0	4	
9	4	0	

0	7	2	
7	0	2	
2	2	0	

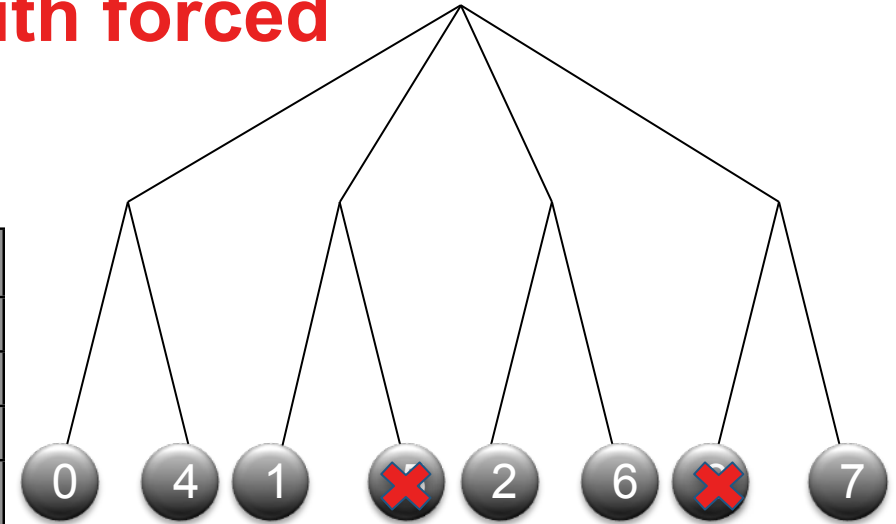
0	2
2	0

Example: bi-partitioning with forced mapping

0	5	7	9	10	2
5	0	7	4	9	1
7	7	0	1	9	2
9	4	1	0	1	7
10	9	9	1	0	10
2	1	2	7	10	0

0 1 2 3 4 5

0	5	7	9	10	2		
5	0	7	4	9	1		
7	7	0	1	9	2		
9	4	1	0	1	7		
10	9	9	1	0	10		
2	1	2	7	10	0		



0	7	2	
7	0	2	
2	2	0	

0	4	9	
4	0	4	
9	4	0	

0	7	2	
7	0	2	
2	2	0	

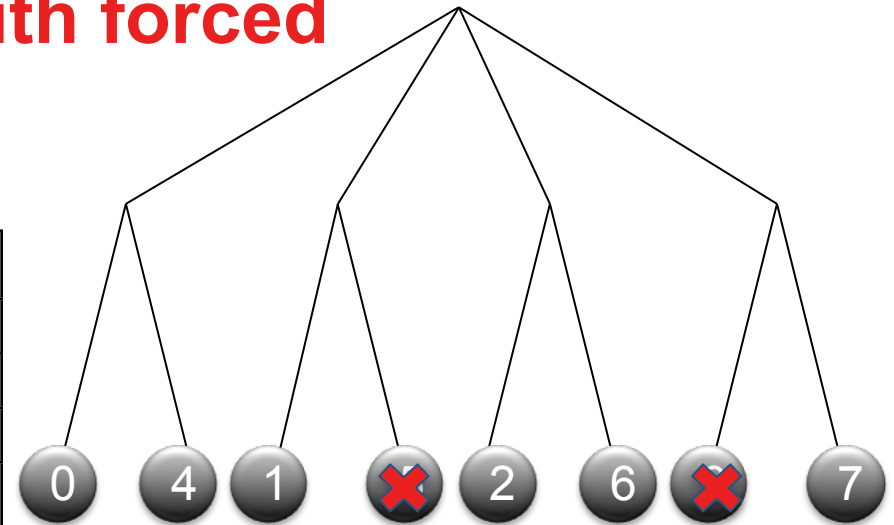
0	2
2	0

Example: bi-partitioning with forced mapping

0	5	7	9	10	2
5	0	7	4	9	1
7	7	0	1	9	2
9	4	1	0	1	7
10	9	9	1	0	10
2	1	2	7	10	0

0 1 2 3 4 5

0	5	7	9	10	2		
5	0	7	4	9	1		
7	7	0	1	9	2		
9	4	1	0	1	7		
10	9	9	1	0	10		
2	1	2	7	10	0		



0	7	2	
7	0	2	
2	2	0	

0	4	9	
4	0	4	
9	4	0	

0	7	2	
7	0	2	
2	2	0	

0	2
2	0

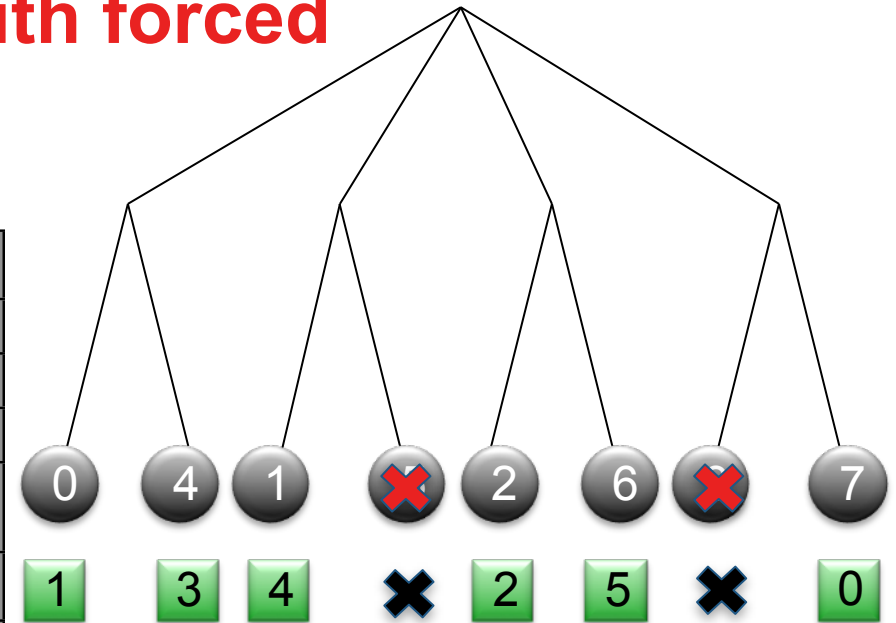
0	

Example: bi-partitioning with forced mapping

0	5	7	9	10	2
5	0	7	4	9	1
7	7	0	1	9	2
9	4	1	0	1	7
10	9	9	1	0	10
2	1	2	7	10	0

0 1 2 3 4 5

0	5	7	9	10	2		
5	0	7	4	9	1		
7	7	0	1	9	2		
9	4	1	0	1	7		
10	9	9	1	0	10		
2	1	2	7	10	0		



0	7	2	
7	0	2	
2	2	0	

0	4	9	
4	0	4	
9	4	0	

0	7	2	
7	0	2	
2	2	0	

0	2
2	0

0	

Implementation in MPI

MPI_Dist_graph_create

Outputs a new communicator with new ranks:

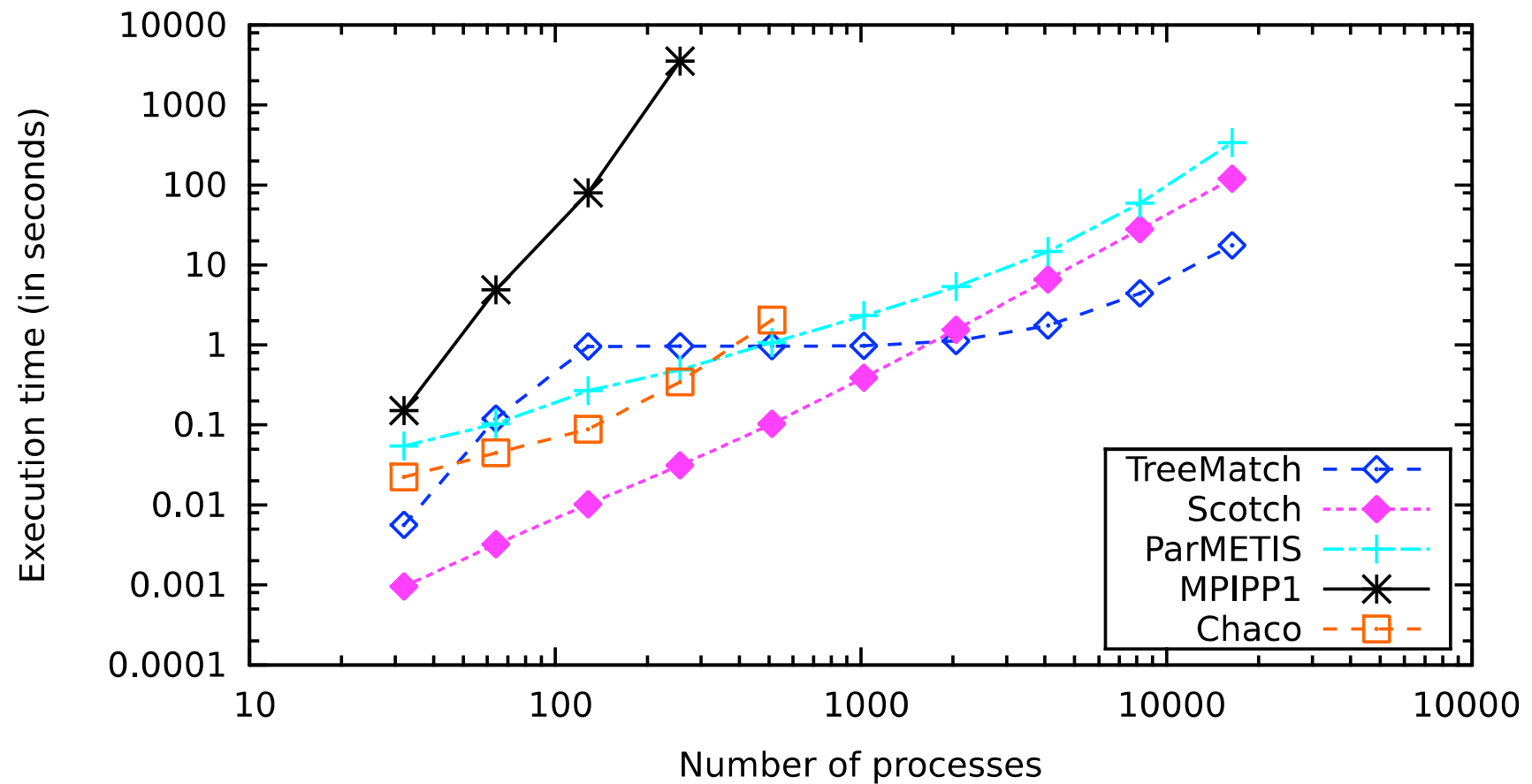
Advantages:

- Standard: MPI_Dist_graph_create routine
- Dynamic: can be done at runtime several times
- Flexible: can take into account placement constraints

Integrated into OpenMPI

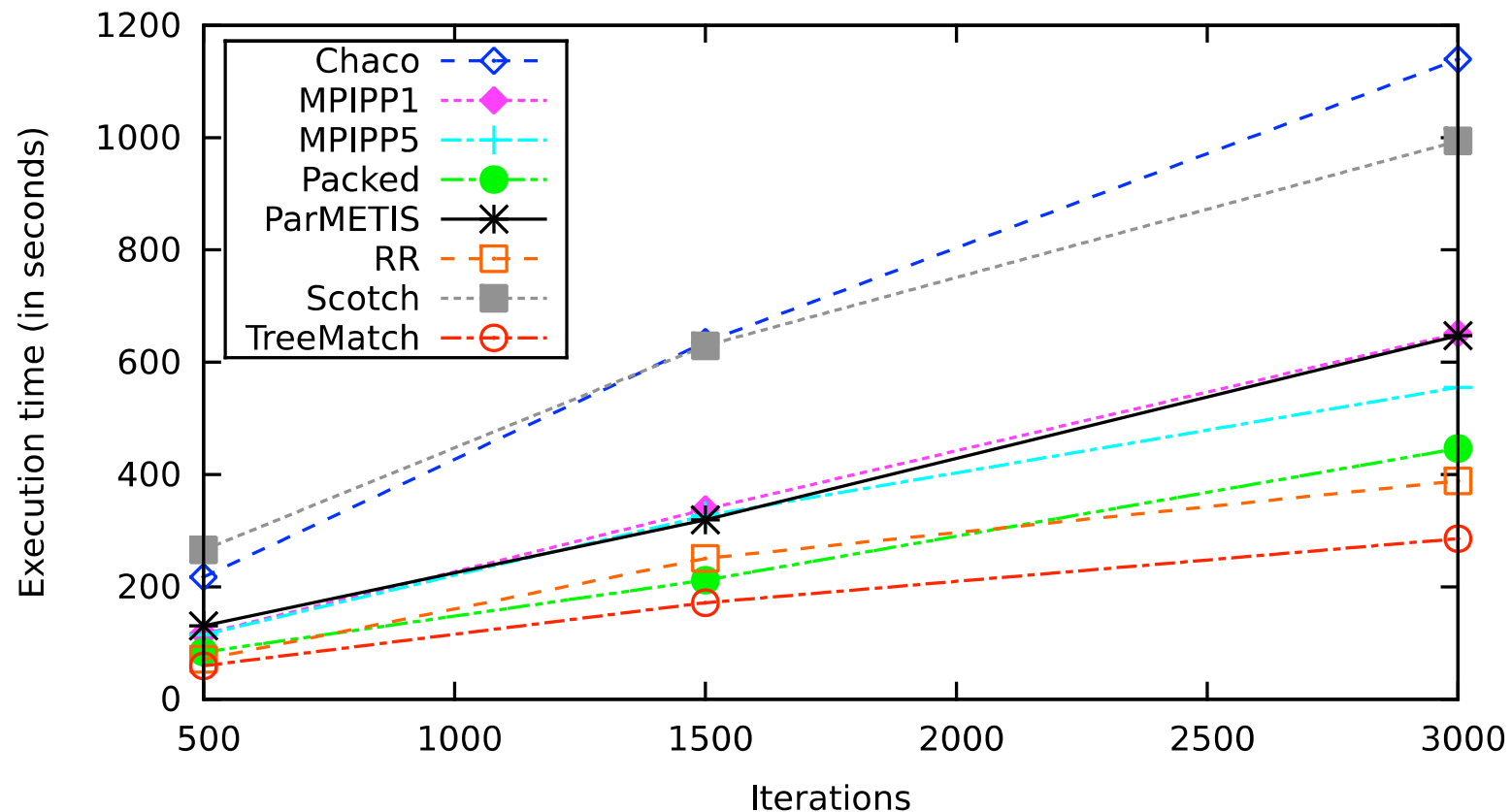
Results: placement computation time

Mapping time comparison between TreeMatch
and other graph partitionners on large communication patterns



Results

Processes binding on ZEUS-MP/2 CFD application - metric : msg - processus : 256

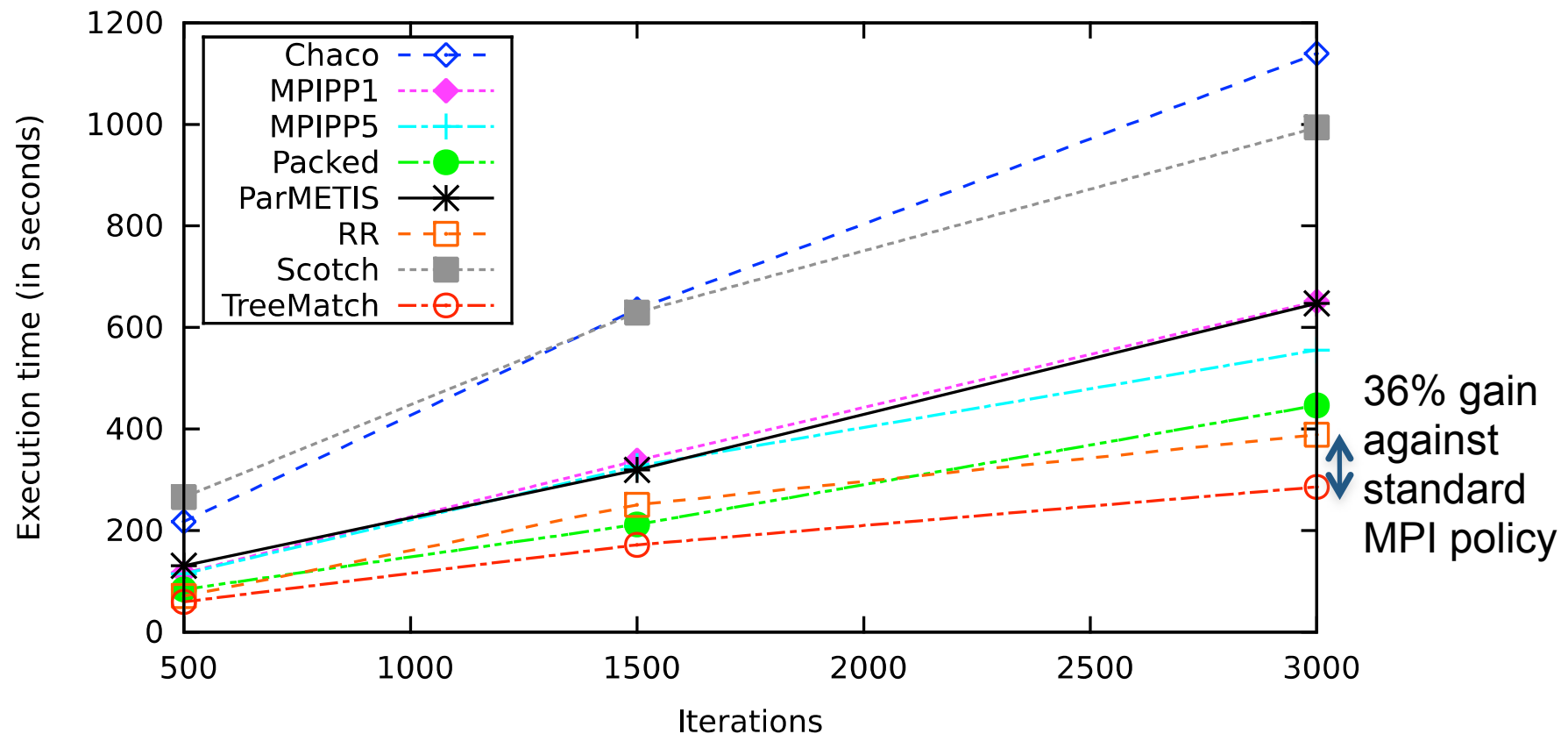


64 nodes linked with an Infiniband interconnect (HCA: Mellanox Technologies, MT26428 ConnectX IB QDR).

Each node features two Quad-core INTEL XEON NEHALEM X5550 (2.66 GHz) processors.

Results

Processes binding on ZEUS-MP/2 CFD application - metric : msg - processus : 256

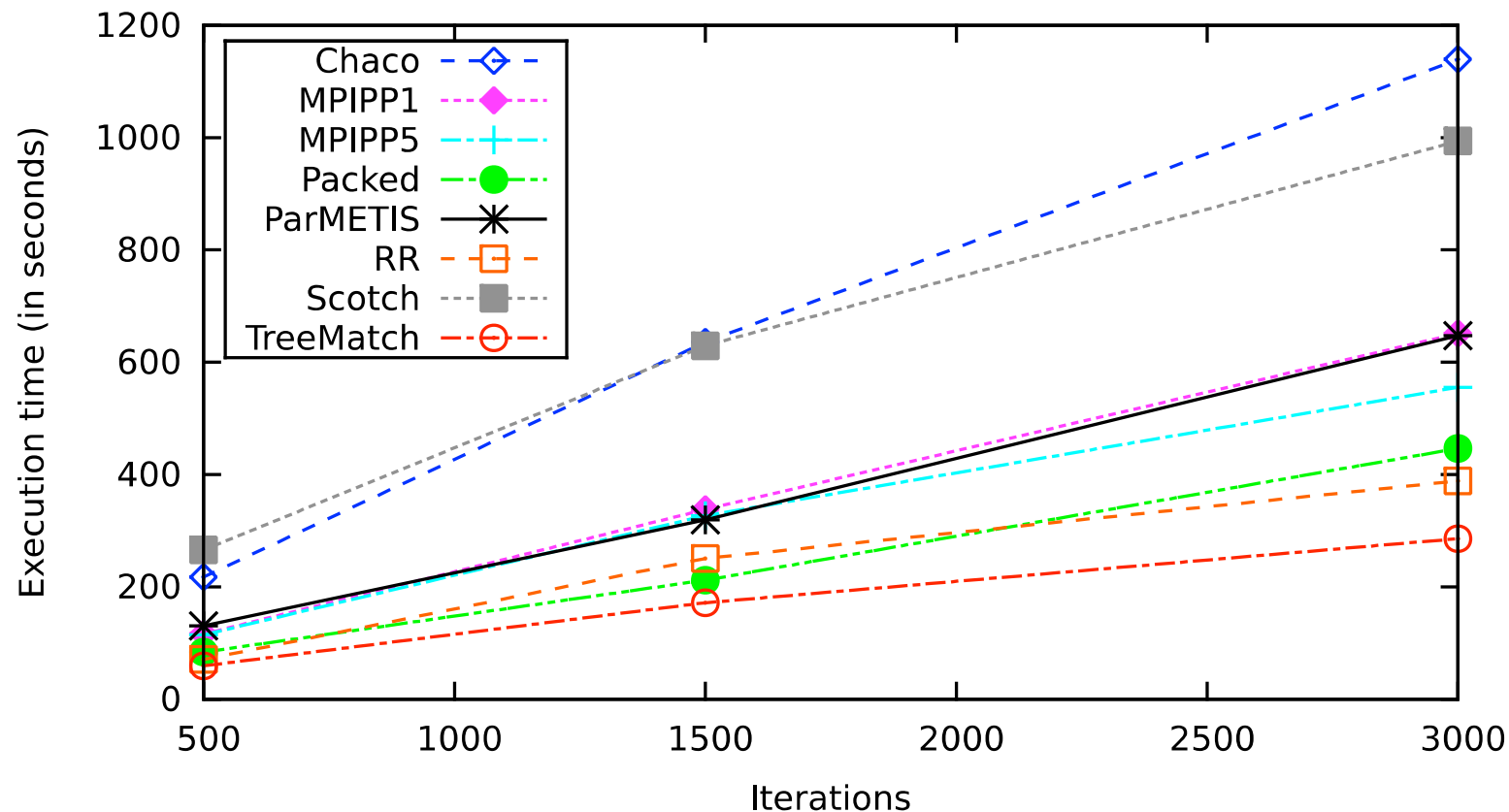


64 nodes linked with an Infiniband interconnect (HCA: Mellanox Technologies, MT26428 ConnectX IB QDR).

Each node features two Quad-core INTEL XEON NEHALEM X5550 (2.66 GHz) processors.

Results

Processes binding on ZEUS-MP/2 CFD application - metric : msg - processus : 256

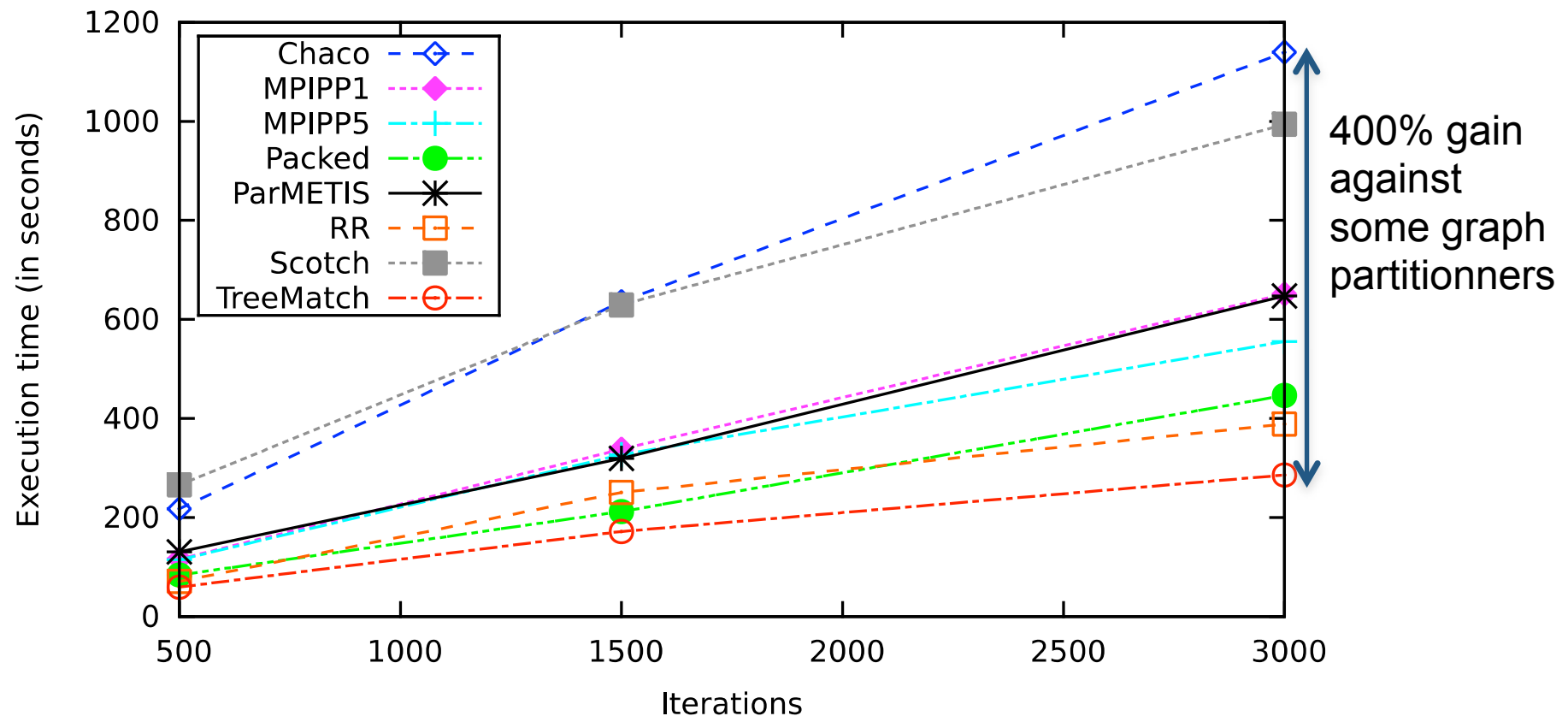


64 nodes linked with an Infiniband interconnect (HCA: Mellanox Technologies, MT26428 ConnectX IB QDR).

Each node features two Quad-core INTEL XEON NEHALEM X5550 (2.66 GHz) processors.

Results

Processes binding on ZEUS-MP/2 CFD application - metric : msg - processus : 256



64 nodes linked with an Infiniband interconnect (HCA: Mellanox Technologies, MT26428 ConnectX IB QDR).

Each node features two Quad-core INTEL XEON NEHALEM X5550 (2.66 GHz) processors.

Conclusion

To ensure performance portability one must take into account the topology of target machine

Process placement according to application behavior and topology helps in increasing performance

(Potential) collaborations:

- TreeMatch in LibTopoMap (with Torsten)
- TreeMatch as LB in Charm++ (with Sanjay)
- TreeMatch in Dist_graph_create in MPICH2 (with Pavan)
- Mixing communication affinity and I/O affinity (With Rob)

Thanks!



CCDSC 2012
www.inria.fr