

Somewhere in between of Science, Art and Nerdish Crazyyness: astroHPC

Luca Tornatore, OATs, USC-VIII meeting, June 2023 Catania






Numerical Cosmology Group @ OATS

S. Borgani, T. Castro, G. Granato, P. Monaco, G. Murante, E. Rasia, G. Taffoni, M. Valentini
L. Cappelli, A. Damiano, M. Esposito, M. Lepinzan, G. Lacopo



O U T L I N E

2 cents on HPC in present and future times
in view of tomorrow's discussion and future actions

-  HPC in its high-end reality
-  Key Messages from “many many details” and a **Key Question**
-  A worked example

HPC in its high-end reality



System name	Cores	Peak Performance (Pflop/s)	HPL (Pflop/s)	HPCG (Pflop/s)	MW
Frontier	8.7 M	1700	1200 (71%)	16 (2.9%)	22.7
Fugaku	7.6 M	540	440 (82%)	14 (0.8%)	30
LUMI	2.2 M	430	310 (72%)	3.4 (0.8%)	6
Leonardo	1.8 M	300	240 (72%)	3.1 (1 %)	7.4
Summit	2.4 M	200	150 (73%)	2.9 (1.4%)	10

top500.org -- June 2023 list

HPC: ... reality



Matrix-Matrix Mul.

Flops / byte $\sim n$

*very high arithmetic intensity
very little memory moving*

	Matrix-Matrix Mul. (Petaflops)	Matrix-Matrix Mul. (Petaflops)	HPL (Pflop/s)	HPCG (Pflop/s)	MW
Frontier	3.7 P	1700	126	126	22.7
Fugaku	7.6 M	540			
LUMI	2.2 M	430			
Leonardo	1.8 M	300			
Summit	2.4 M	200			

System of linear equations

Flops / byte ~ 1

*still high arithmetic intensity
significant memory moving*

top500.org -- June 2023 list

HPC in its high-end reality



System
name

HPL
(Pflop/s)

HPCG
(Pflop/s)

MW

Note that the HPCG
performance brings us
back to reality:

*We are now entering in
Peta-Scale era*

1%)

16 (2.9%)

22.7

14 (0.8%)

30

3.4 (0.8%)

6

3.1 (1%)

7.4

(73%)

2.9 (1.4%)

10

top500.org -- June 2023 list

HPC

The only platform **without GPUs**
Fugaku CPUs are **ARM-based vector CPUs**

- ~ 90% of HPCG performance than Frontier with 1/3rd of theoretical peak performance
- larger efficiency in HPL



					HPCG (Pflop/s)	MW
Frontier	8.7 M	1700	1200 (71%)		16 (2.9%)	22.7
Fugaku	7.6 M	540	440 (82%)		14 (0.8%)	30
LUMI	2.2 M	430	310 (72%)		3.4 (0.8%)	6
Leonardo	1.8 M	300	240 (72%)		3.1 (1 %)	7.4
Summit	2.4 M	200	150 (73%)		2.9 (1.4%)	10

top500.org -- June 2023 list

Key messages



- The “exa-scale” narrative may be misleading
a well-known fact, not something that only my cousin knows, and he told me
- Real-world scientific codes involve
 - diverse algorithms with different arithmetic intensity
 - (lots of) memory moving
 - conditional branchingas such **squeezing every theoretical flop from your supercomputer is really not trivial**
- **taking full advantage of modern architectures often requires some global re-thinking of the code**

Key messages



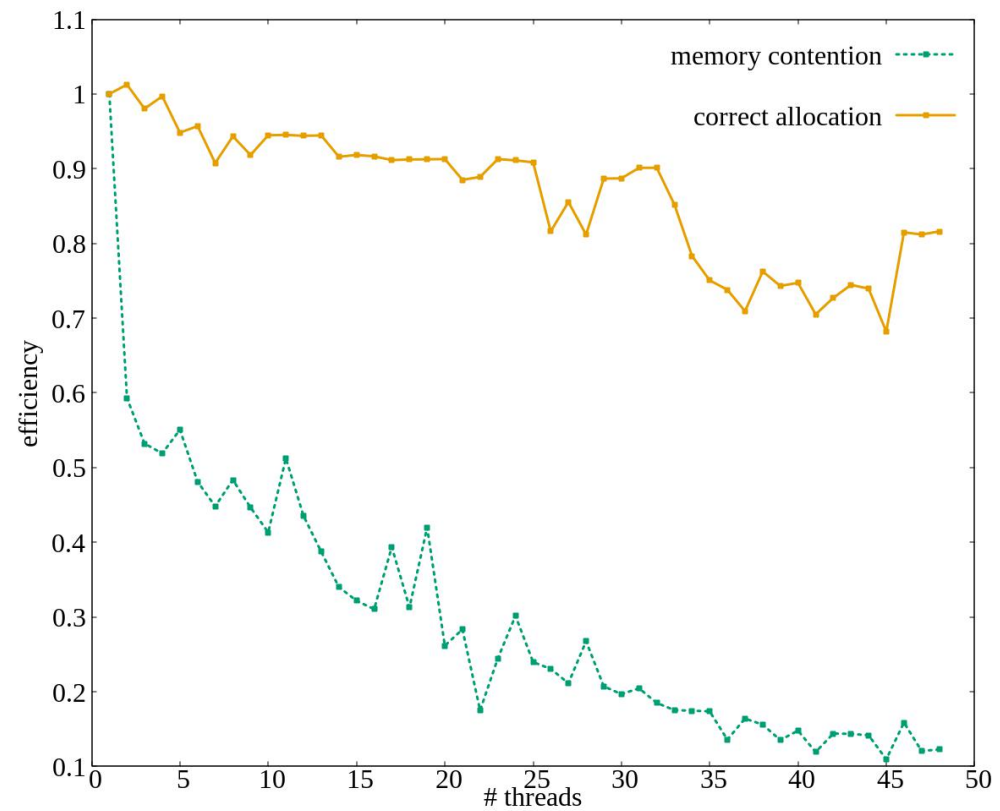
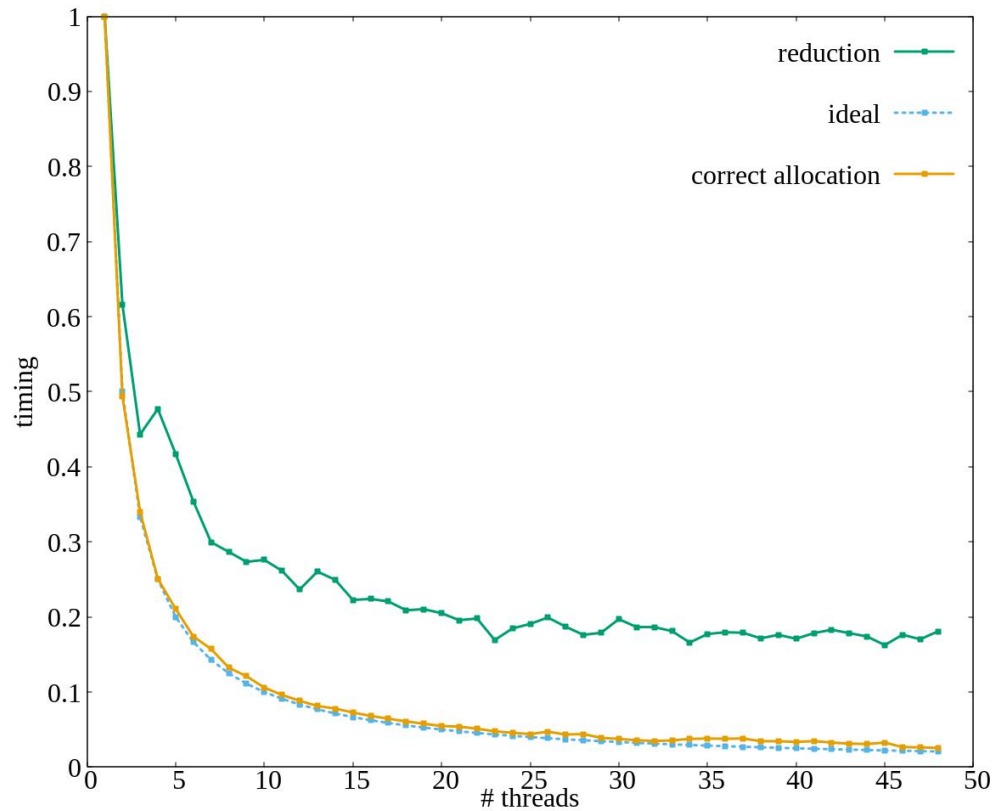
squeezing every theoretical flop from a supercomputer really is not trivial

While the algorithms drive the performance,

several factors lie between our codes and the maximum performance (in terms of time-to-solution)

- the **data model**
- the quality of the implementation in **many many details**
the compilers are good but writing a great source allows them issuing an amazing code
- the capability of the code to **adapt to the platform & exploit parallelism**

A trivial example of “many many details”



This are the results for two implementation of a OpenMP processing of an array. The **green** one is correct: **why its efficiency is quickly decaying?** (*no false sharing, no naives accumulation or atomic inside loops*) **why the orange one is much better?** I've seen this in many critical codes...

Key Question for INAF



“black-box” approach

INAF focus only on science, subcontracting the HPC & code development to external experts, then just using HPC-as-a-service

Jedi approach

INAF *strongly* invest in an **internal** HPC track with a high-level & intense competence density



vital, continuous and fertile **interaction and collaboration with external** actors (HPC centers, companies, EU projects)

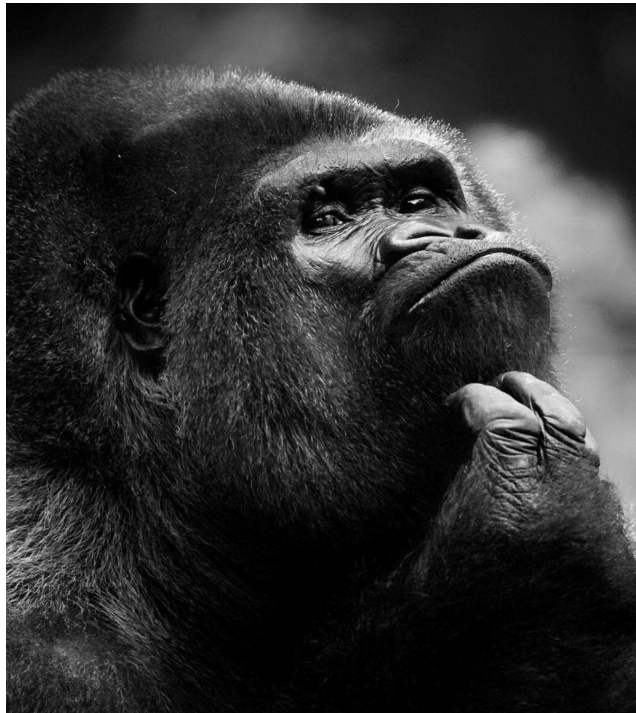
Key Question for INAF



“black-box” approach



Jedi approach



HINT (Humble INsight Thought)

- **Telescopes** of all kinds are **essential tool** for science and INAF has sharp and deep knowledge and competence on that.
- **HPC** is an essential tool, too... the quality of the code drives the science we can do

Every Key Message comes with a Key Quote



“ Two roads diverged in a yellow wood
And sorry I could not travel both
..
I took the one less traveled by,
And that has made all the difference. ”

R. Frost (it wasn't about HPC)

“ do or do not there is no try ”
you know who

A worked example

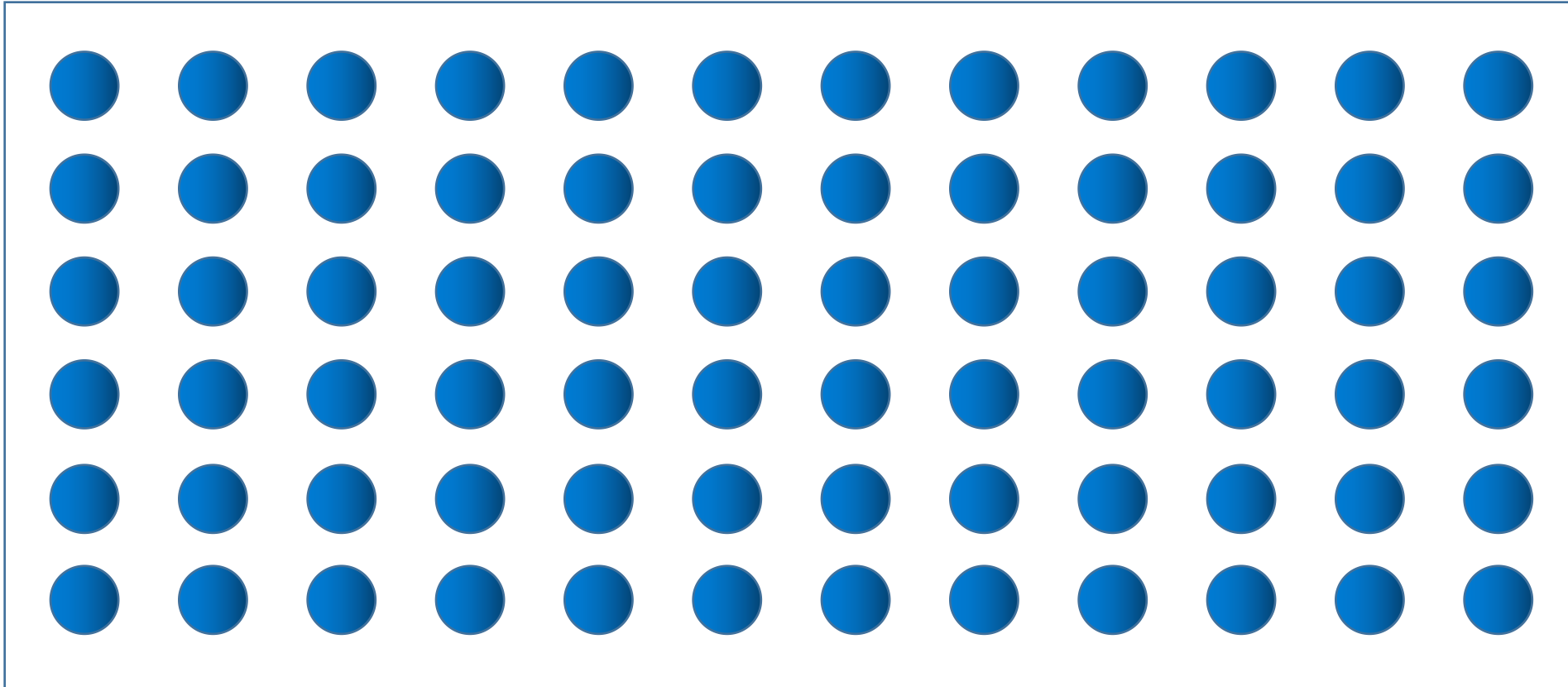


What if a **Reduce** operation, that we resolve with a MPI_Reduce (or a MPI_Ireduce), is a severe bottleneck,
like it **takes 60% to 90% of the time-to-solution** ?

Origin of the problem: see C. Gheller's and G. Lacopo's talks about a data-stacking code in radio astronomy.

During the previous Spoke3 meeting it arose that other use cases have similar issues.

Let's start from the beginning



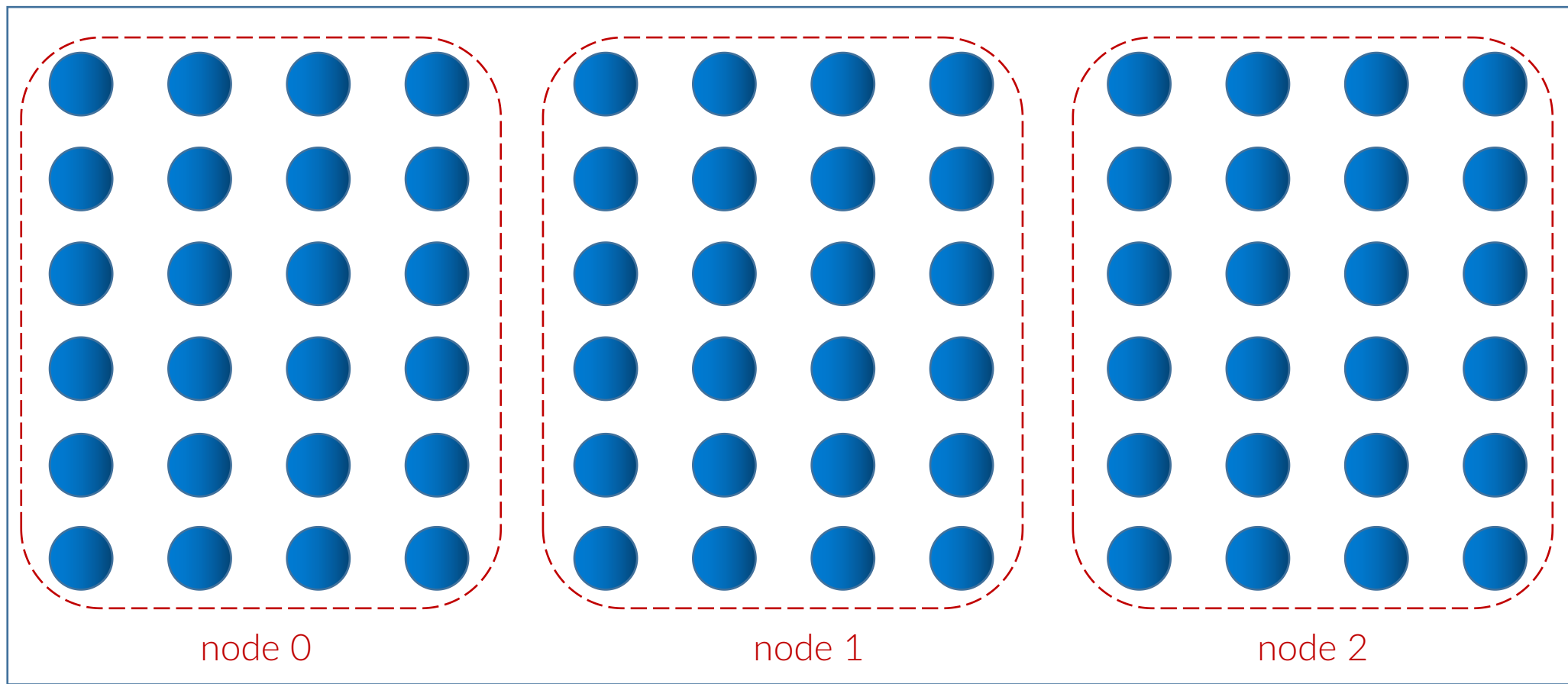
In many advanced MPI codes we routinely deal with a indistinct “lake” of MPI processes.

When writing the code we are agnostic about *where* these MPI tasks will be running.

At run-time the underlying MPI lib resolves dynamically whether a communication happens either in shared-memory or via network.

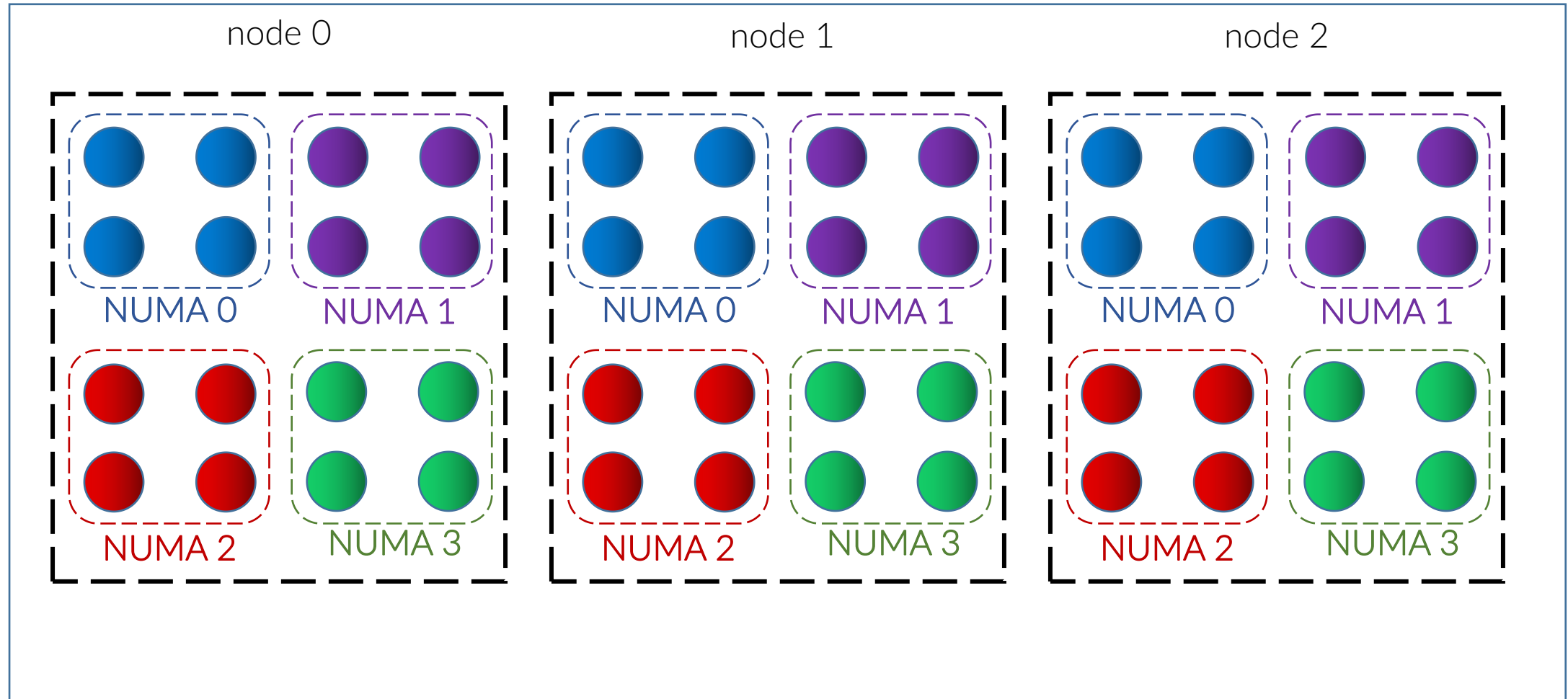
myMPI_COMM_WORLD

However, at run-time we can easily map **what tasks run in what nodes**, what tasks are running **on what sockets**, what threads are running **on what cores** and whether they are hardware threads or SMT-threads



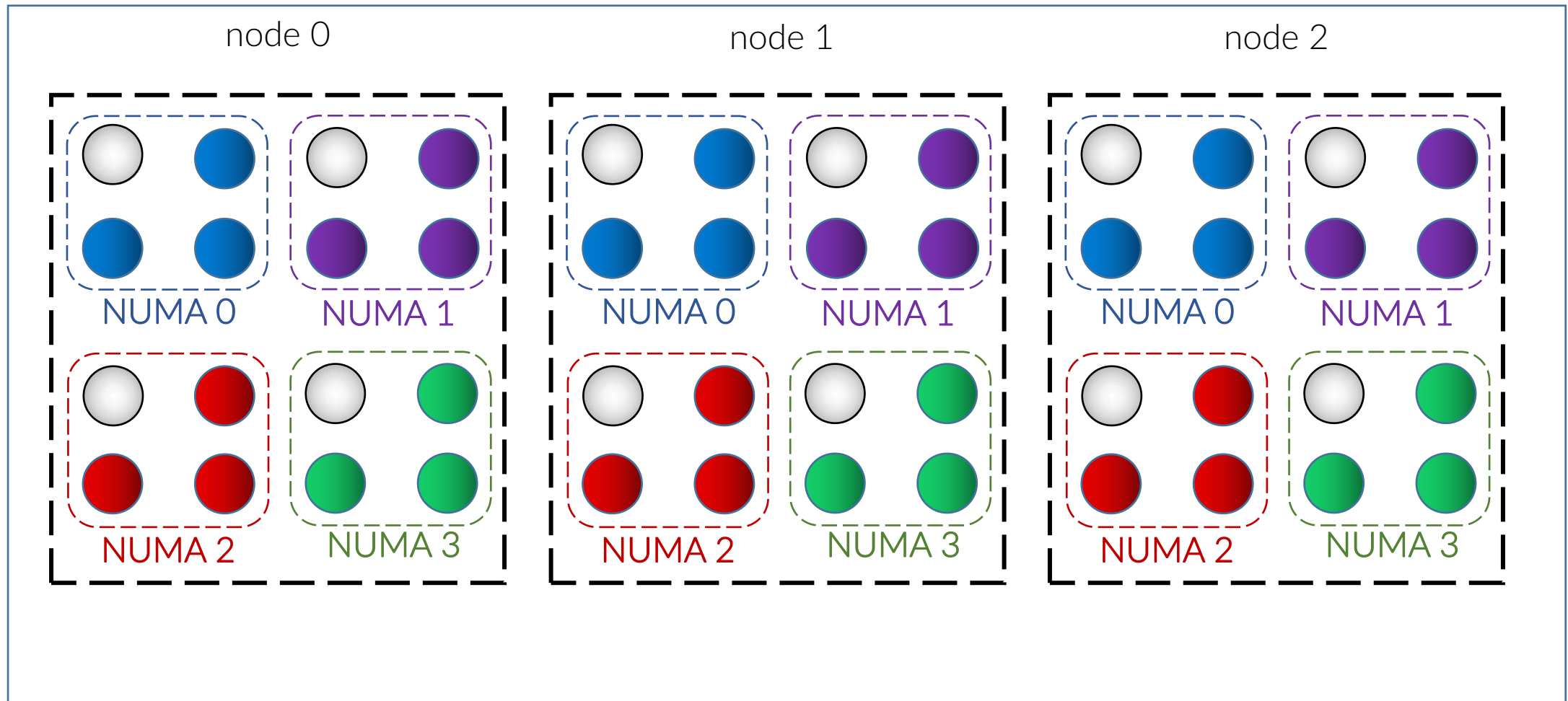
myMPI_COMM_WORLD

And we can build a hierarchy of MPI_Communicators that traces the memory hierarchy



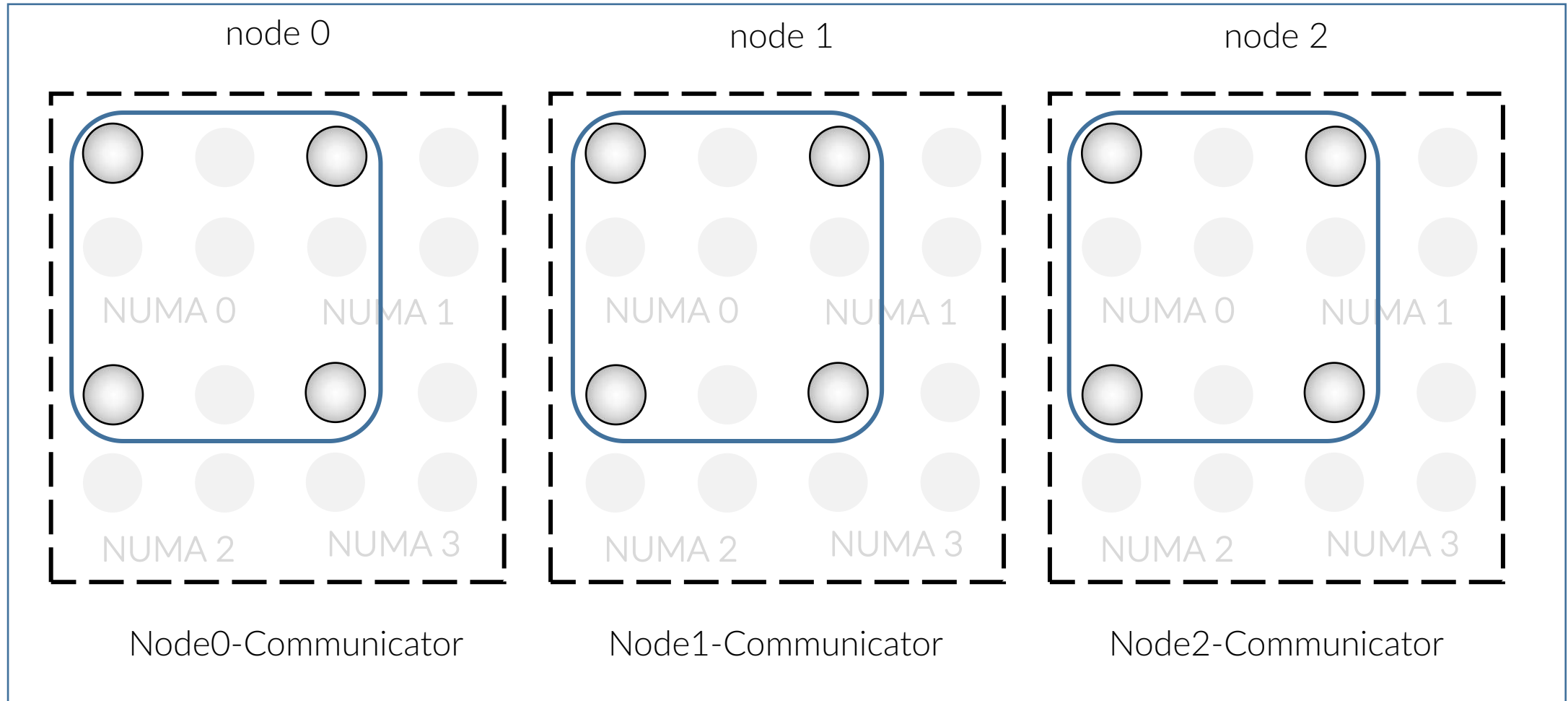
myMPI_COMM_WORLD

And we can build a hierarchy of MPI_Communicators that traces the memory hierarchy, appointing a *master* in every communicator



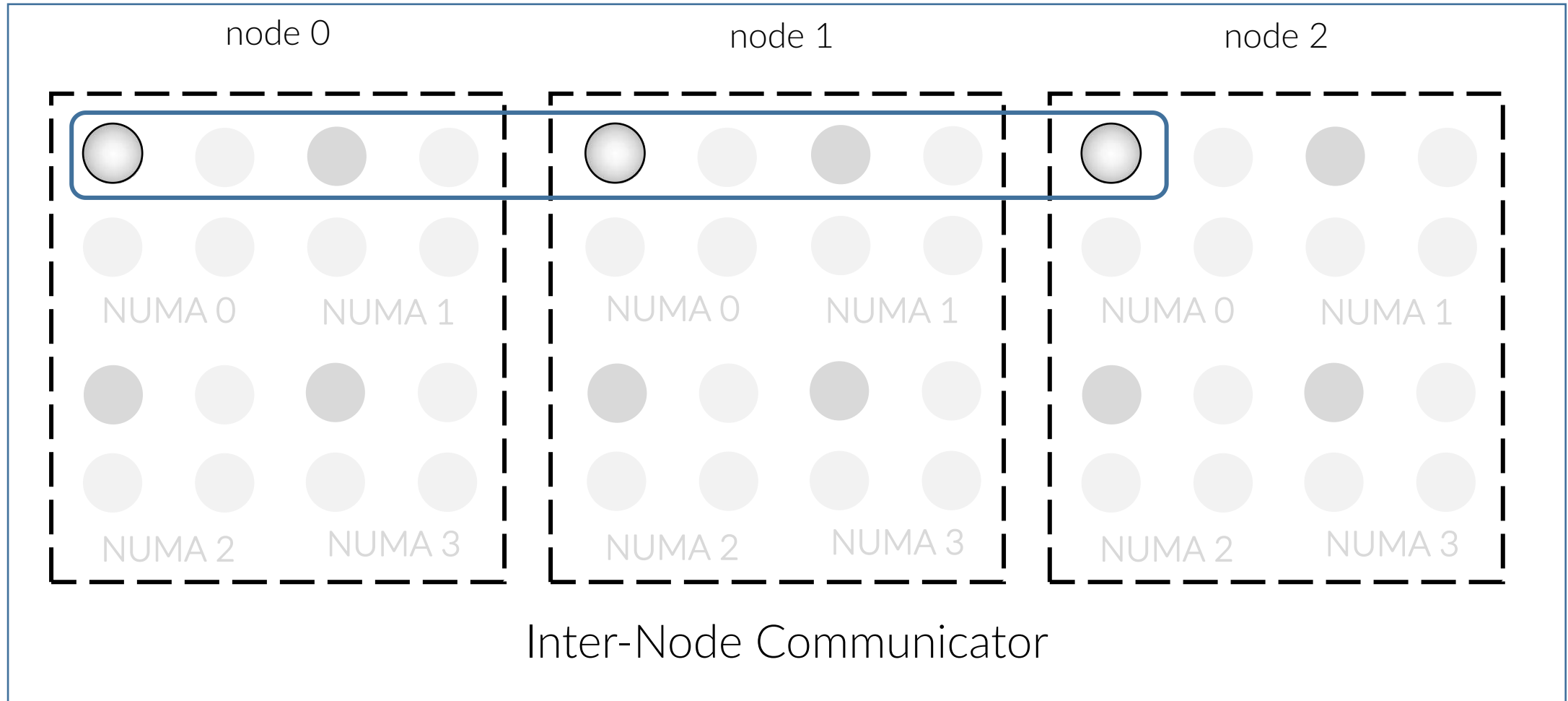
myMPI_COMM_WORLD

And we can build a hierarchy of MPI_Communicators that traces the memory hierarchy, appointing a *master* in every communicator that participates in the upper-level communicator



myMPI_COMM_WORLD

And we can build a hierarchy of MPI_Communicators that traces the memory hierarchy, appointing a *master* in every communicator that participates in the upper-level communicator



myMPI_COMM_WORLD

And we can build a hierarchy of `MPI_Communicators` that traces the memory hierarchy, appointing a *master* in every communicator that participates in the upper-level communicator

- At every level, if appropriate, **MPI tasks can expose shared-memory windows;** “communications” can then be crafted as in shared-memory paradigm (pay attention to **synchronization**, it may be non-trivial)
- If appropriate, masters at every level can expose shared-memory windows in the upper-level communicator, and so on;
- When appropriate, the masters **at the final level expose memory windows** in the top-level communicator and may operate either through **RDMA** or normal **message-passing**.

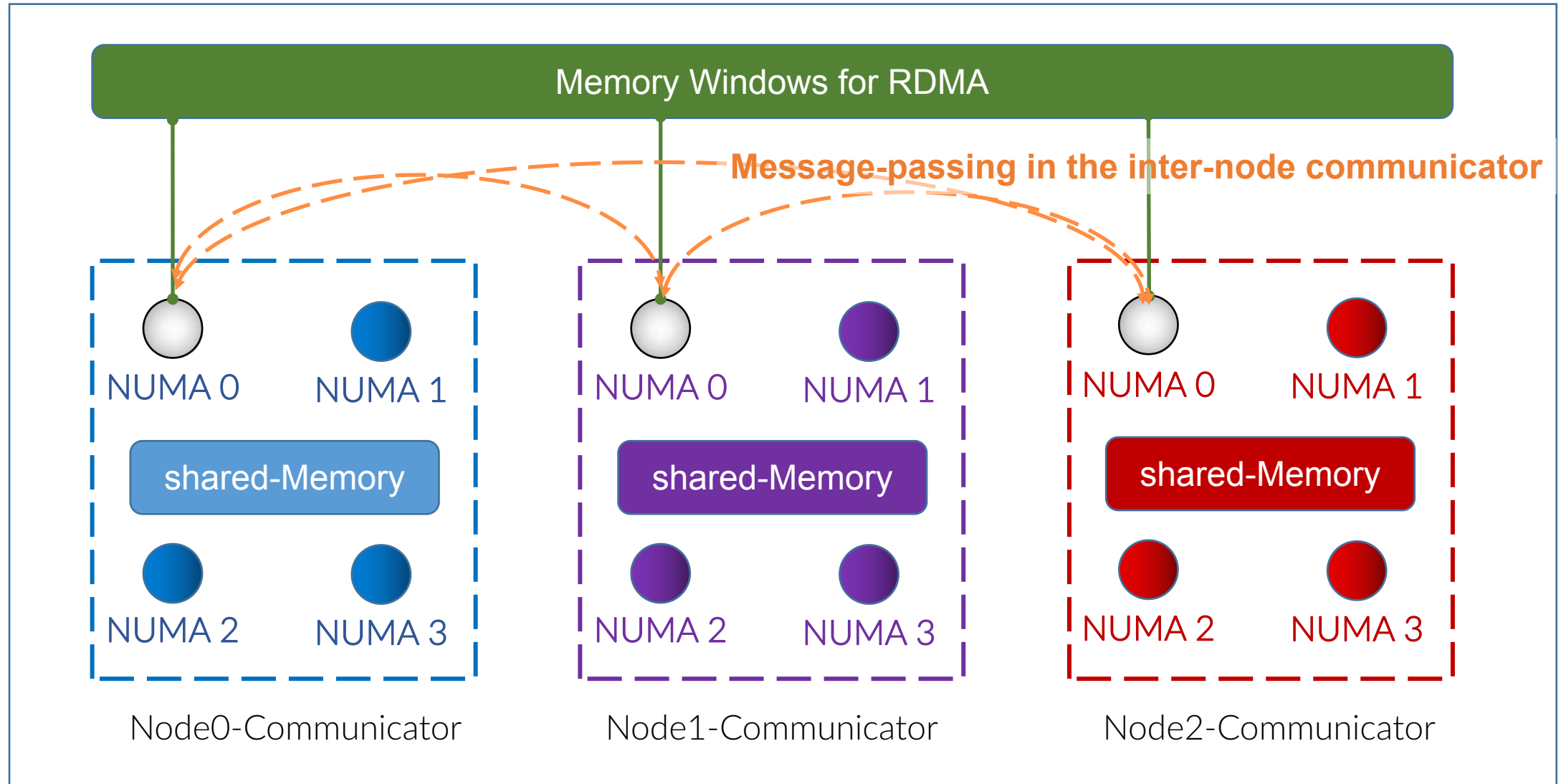
Typically the top-level communicator is a inter-node communicator

Let's simplify the example:

- 4 sockets per node, all RAM is shared at node-level
 - 1 NUMA region per socket
 - let's place 1 MPI task per socket
- (could be more, but let's keep it the simplest possible)

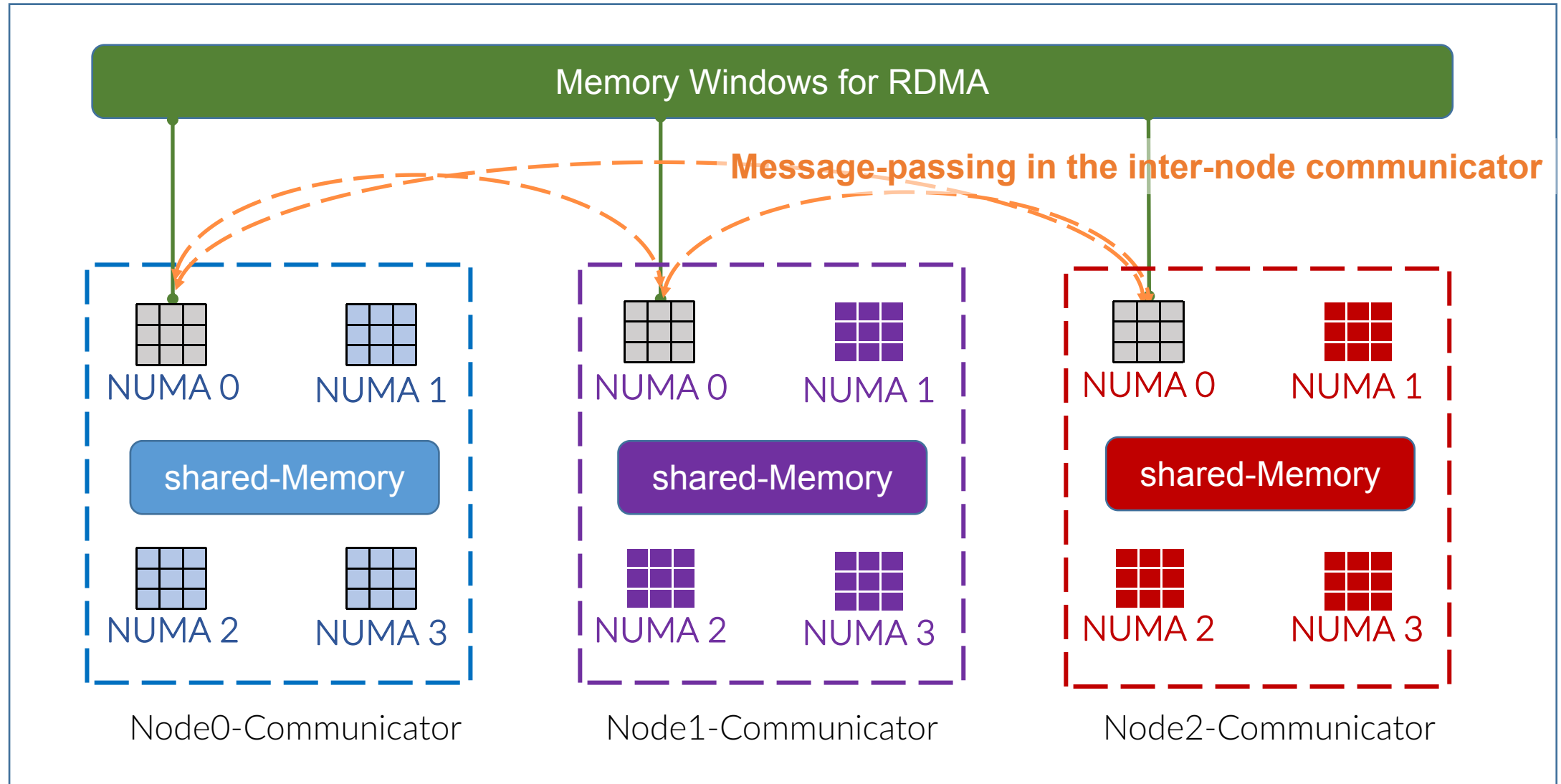
Then, at node level the MPI tasks can operate with shared-memory paradigm, while inter-node they can both use RDMA or messages.

Let's simplify the example



myMPI_COMM_WORLD

Of course, you may want to populate the sockets with threads



myMPI_COMM_WORLD

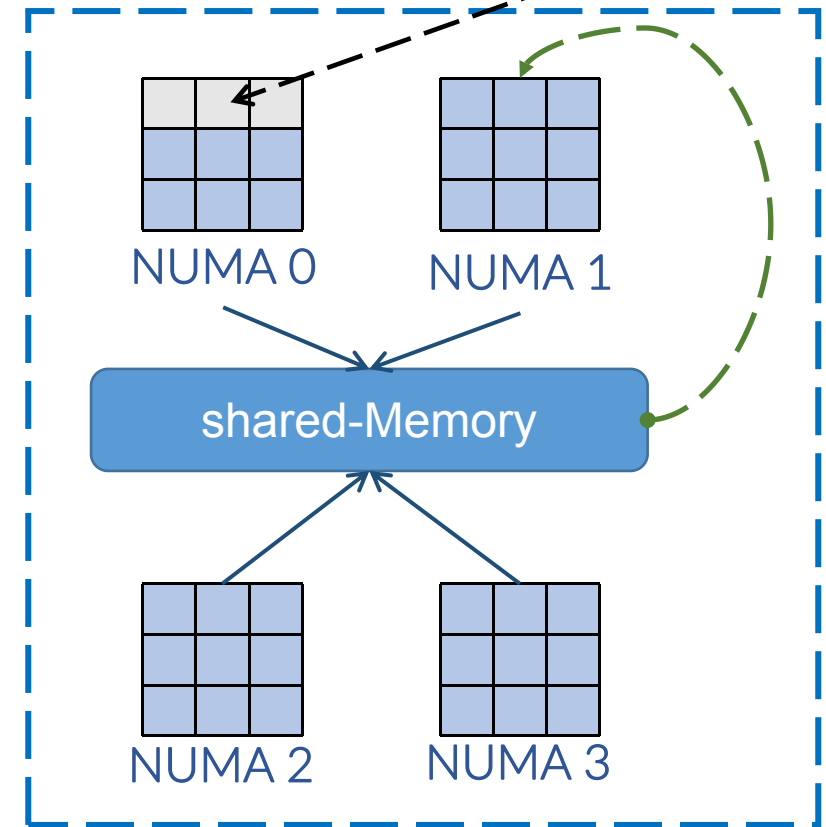
Back to the REDUCE problem



Then, **our reduce** proceeds in 2 steps (that overlaps in time)

1) **within every node** the threads of MPI tasks in the sockets performs a very efficient reduce in shared-memory; we use a RING for that. The result is saved in the target task, if it belongs to the node

threads dedicated to MPI / operations in upper-level communicator

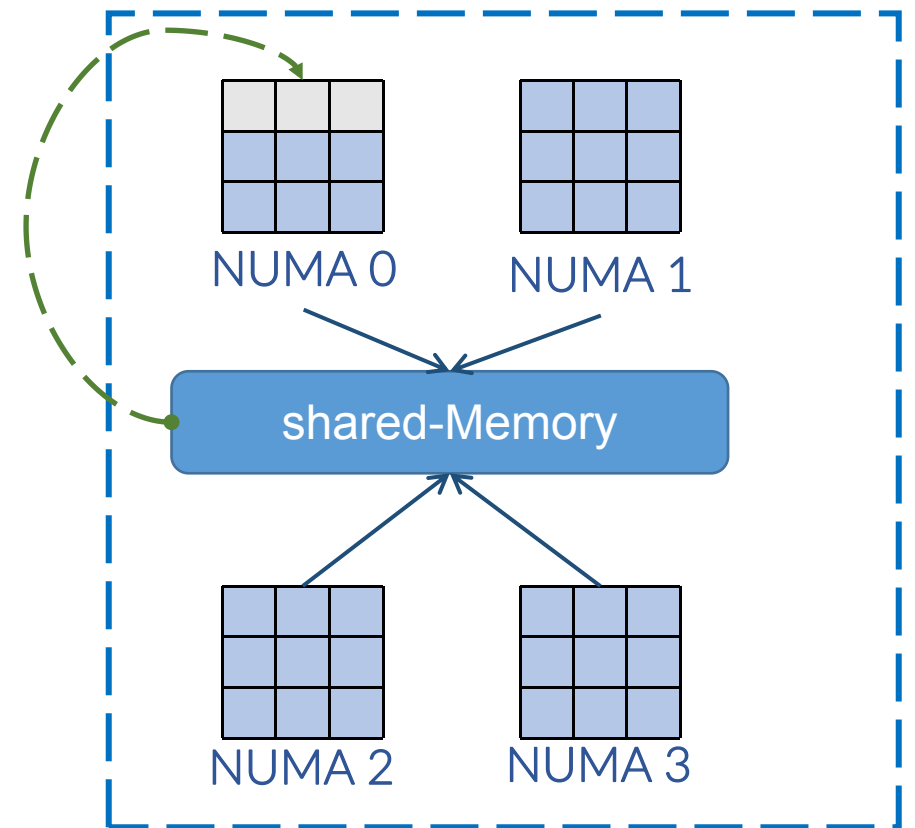


Back to the REDUCE problem

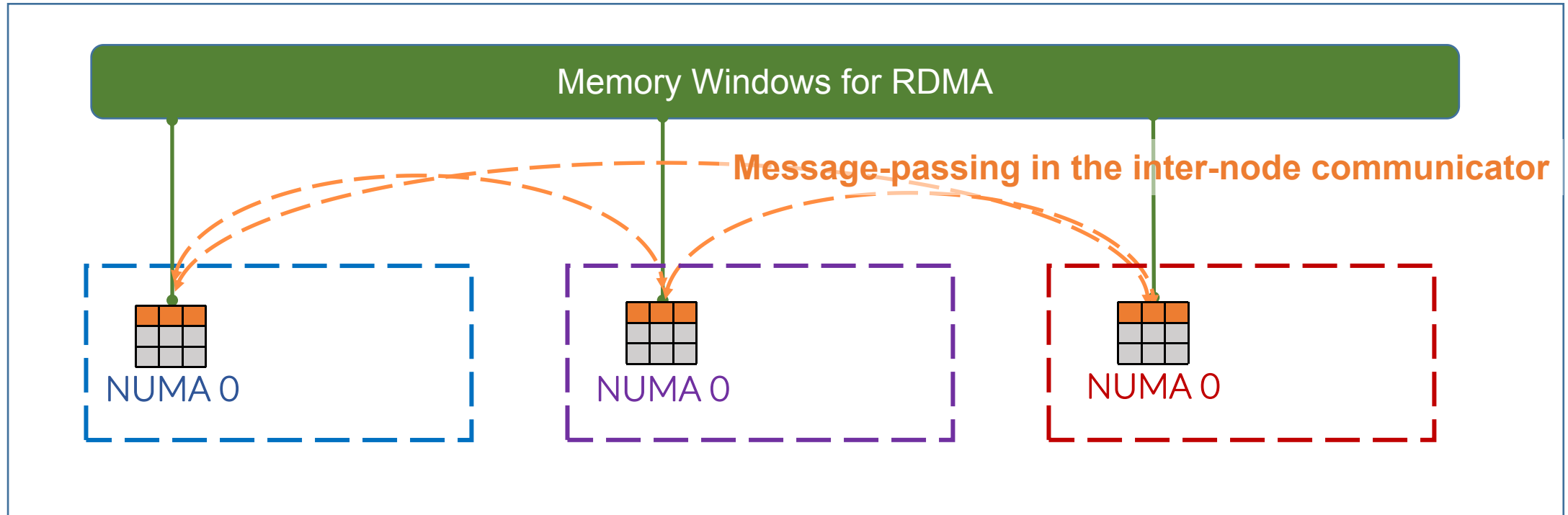


Then, **our reduce** proceeds in 2 steps (that overlaps in time)

1) **within every node** the threads of MPI tasks in the sockets performs a very efficient reduce in shared-memory; we use a RING for that. The result is saved in the master task, if the target task does not belong to this node



2) **internode**, the dedicated threads in the node masters perform the reduce among their partial results; that may happens either with a MPI_Reduce/ MPI_Ireduce in their inter-node communicator or via RDMA access (we implement a RING among them too)



The master of the node that contains the target task writes the result in the target task's memory via shared-memory.

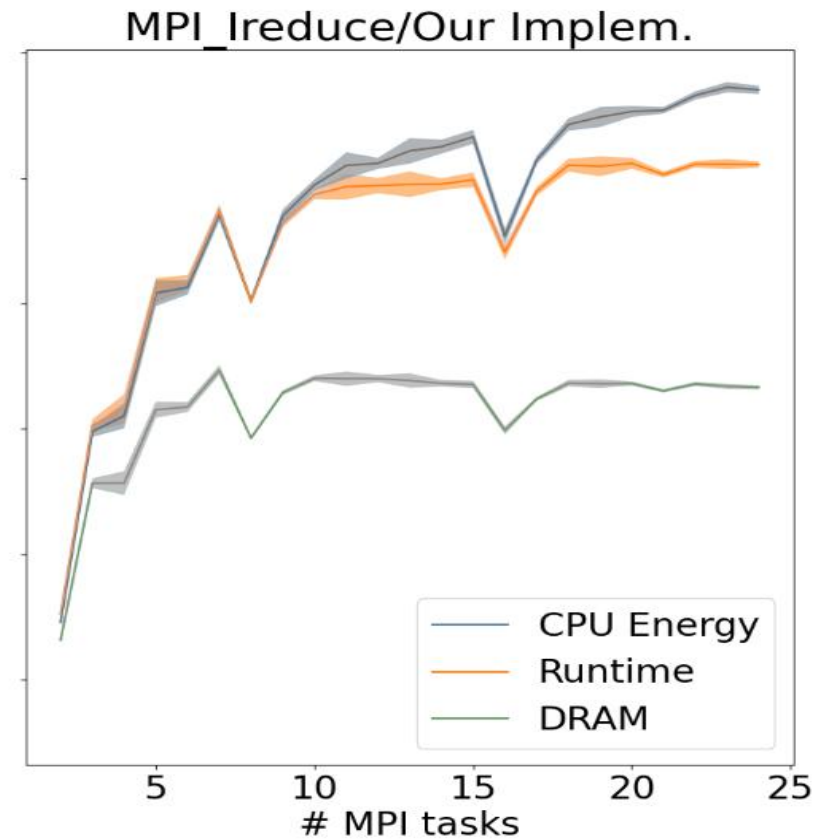
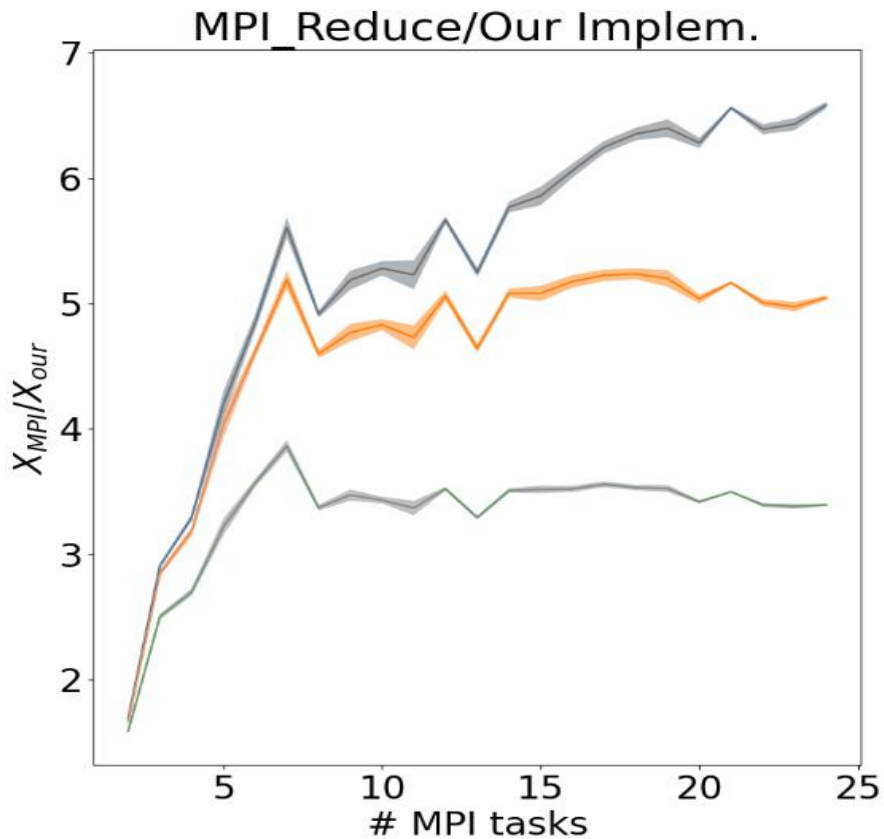
While this inter-node reduce happens, the next reduce starts at node-level, and so on

Back to the REDUCE problem



This brings a significantly faster reduce, 3 to 7 times (taken from G. Lacopo's talks; **focus on orange lines**), at the cost of a larger memory usage.

The inter-node is also faster (results not yet ready, though)

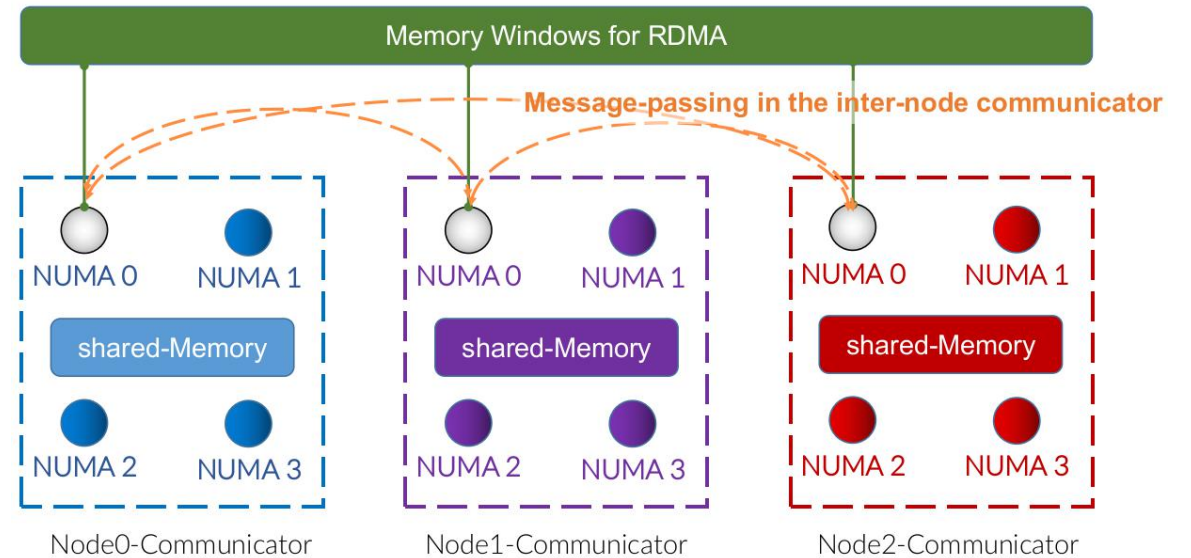


A general comment



The REDUCE problem is just an example, all in all not that complex, of a refined code that implements **explicit NUMA awareness**.

In general, you may place 1 MPI task per GPU, and saturate the rest with nested OpenMP parallelism levels with different binding policies.



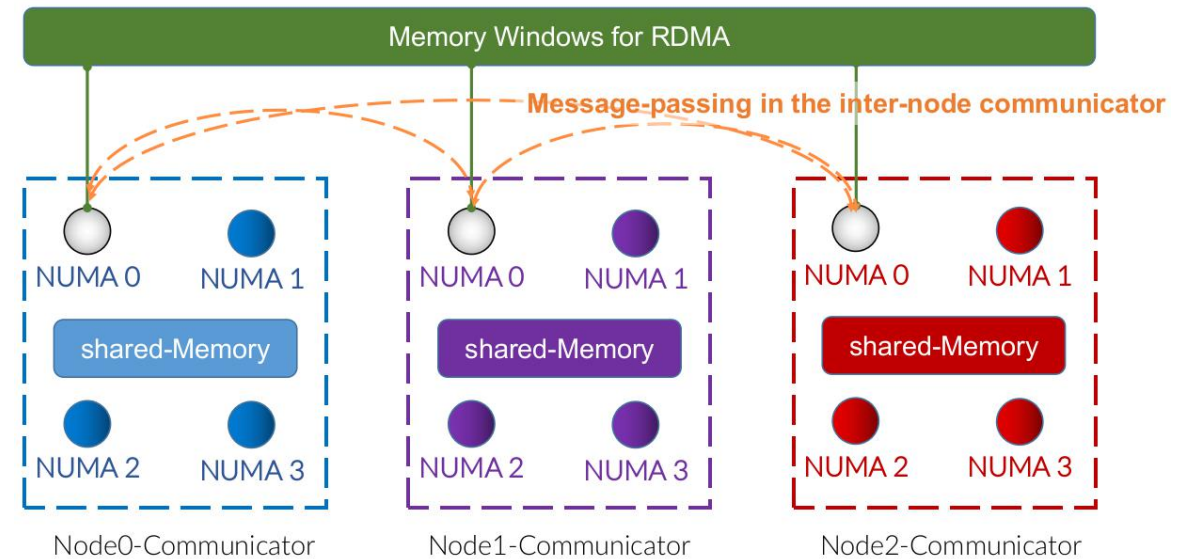
A general comment



More generally, that allows to reduce the *communication surface* while saturating the CPUs computational power.

Domain-Decomposition, for instance, can be re-adapted to **node-domains**, while exploiting **functional decomposition**, by a task-based approach, either via OpenMP or OpenACC, **within the nodes**.

Reserving some threads @ node-master task to MPI, and all the others to internal tasks and/or GPU management, may ease a lot the famous overlap of computation and communication.



CONCLUSIONS

