FROM RESEARCH TO INDUSTRY

TOWARDS AN EFFICIENT COMMUNICATION OVERLAP

THROUGH

HARDWARE OFFLOADING

SEA conference 2018, UCAR

Julien JAEGER,
CEA, DAM, DIF, F-91297 Arpajon, France

4 APRIL 2018

www.cea.fr

MPC Context

Team overview

- Runtime system and software stack for HPC
- Team as of March 2018 (CEA and CEA/Intel/UVSQ ECR Lab)
  - 2 research scientists, 3 PhD students, 3 engineers
  - Contact: patrick.carribault@cea.fr or julien.jaeger@cea.fr
  - Partner: Paratools (1,5 MY)
- Available software
  - MPC framework
  - MALP (performance analysis tool)
  - JCHRONOSS (job scheduler for test suite on production machines)
  - Wi4MPI (MPI abstraction)
- Website for team work: http://hpcframework.com

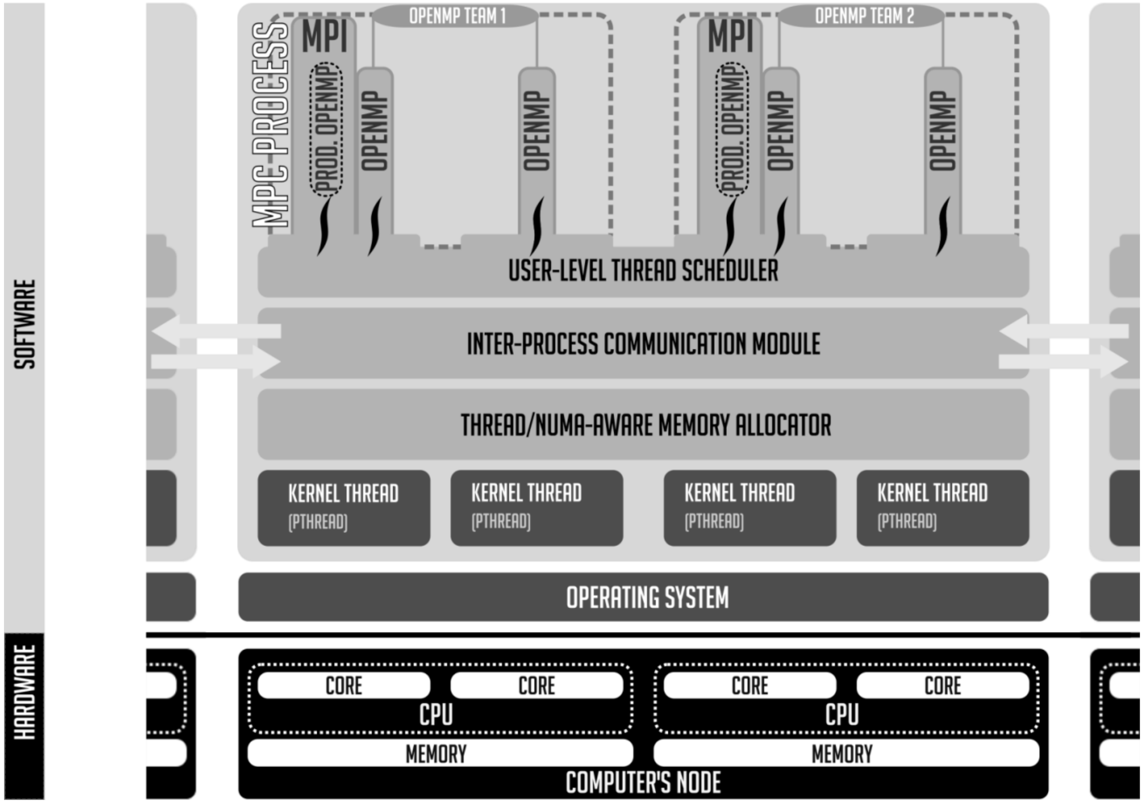
MPC framework

- Unified parallel runtime for clusters of NUMA machines
  - Idea: one process per node, compute units exploited by user-level threads
- Integration with other HPC components
  - Parallel memory allocator, compilers, debuggers, topology tool…
- Homepage: http://mpc.hpcframework.com

MPC

MPC Internals

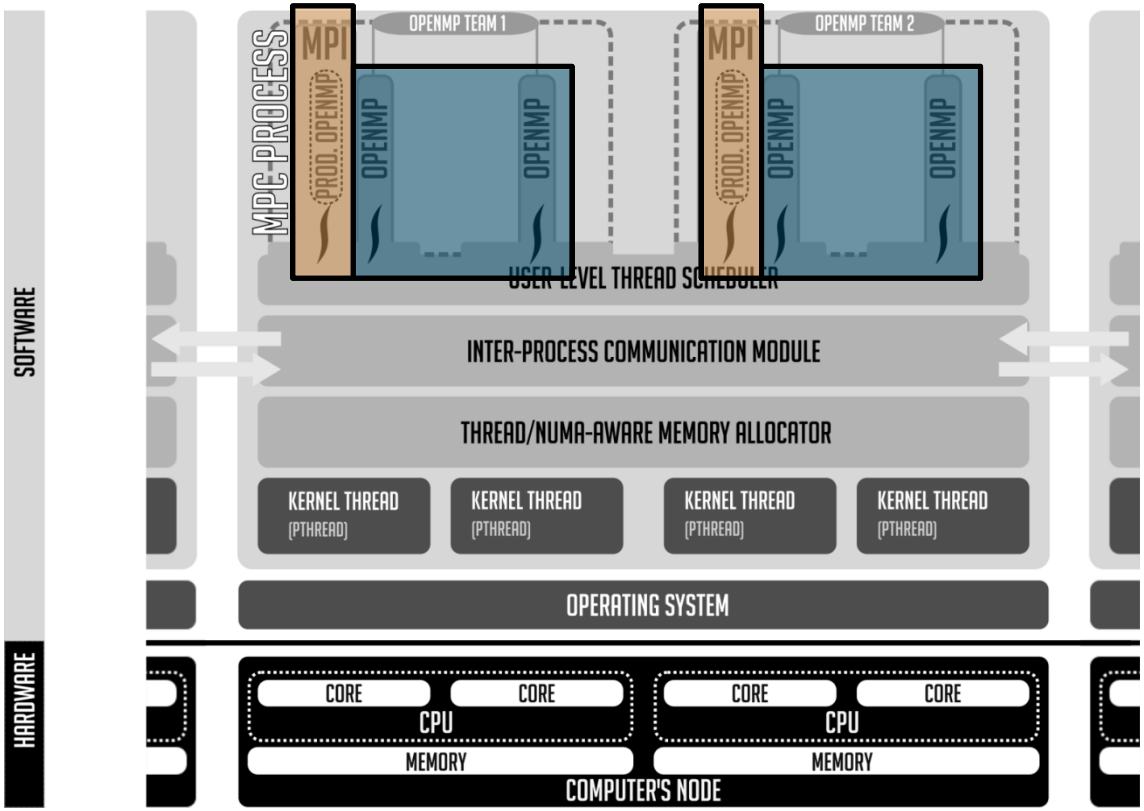
- MPC is…



MPC Internals

- MPC is…

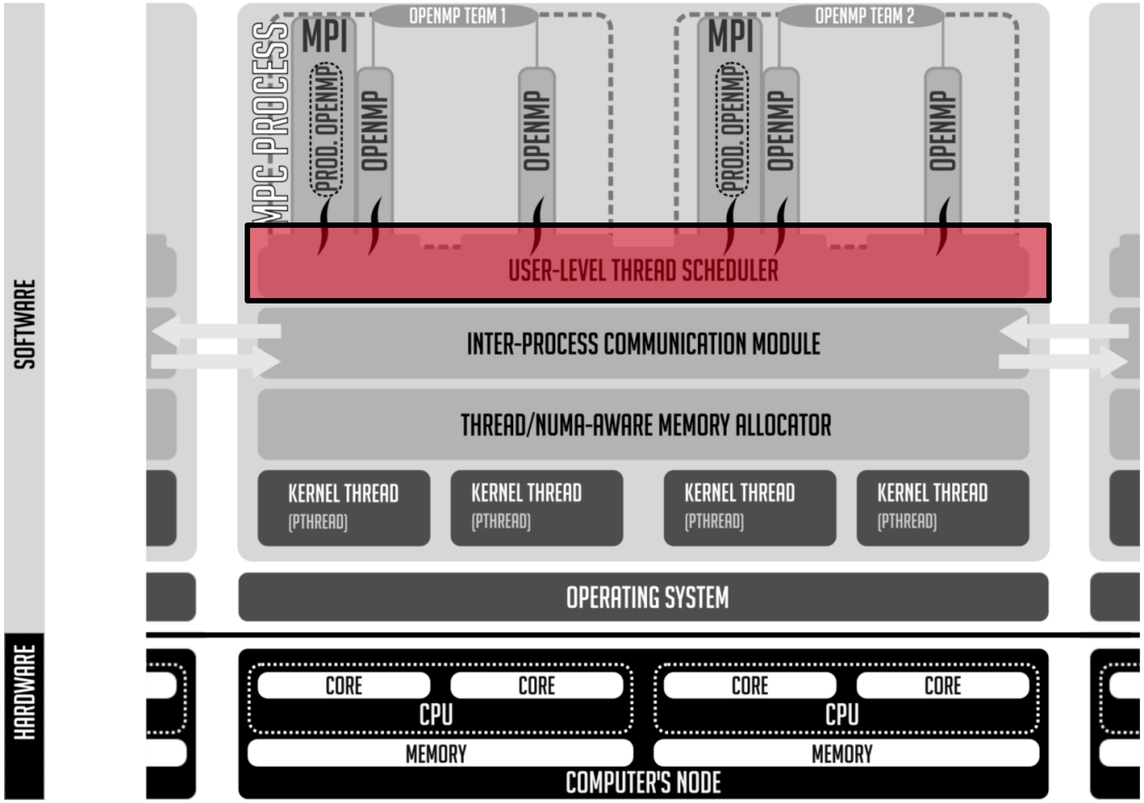
  - … MPI, OpenMP, pthread implementations…



MPC Internals

- MPC is…

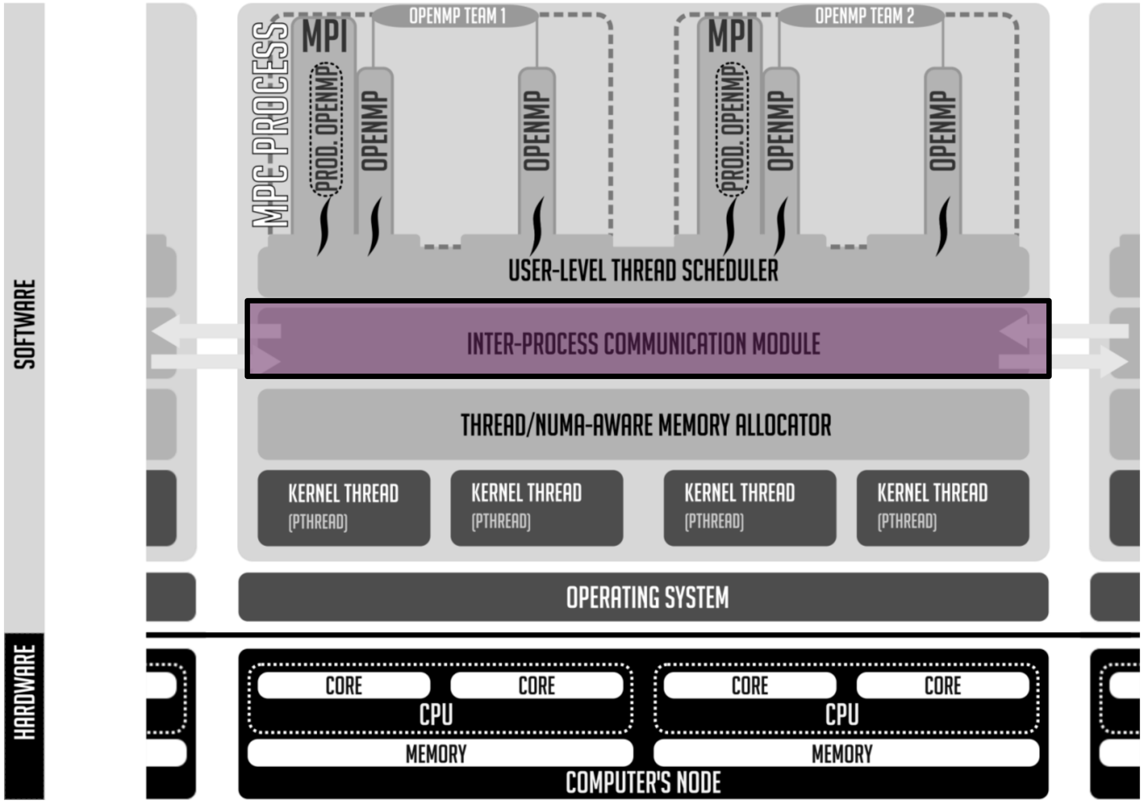
  - … MPI, OpenMP, pthread implementations…
  - … based on the same thread scheduler allowing interoperability, …



MPC Internals

- MPC is…

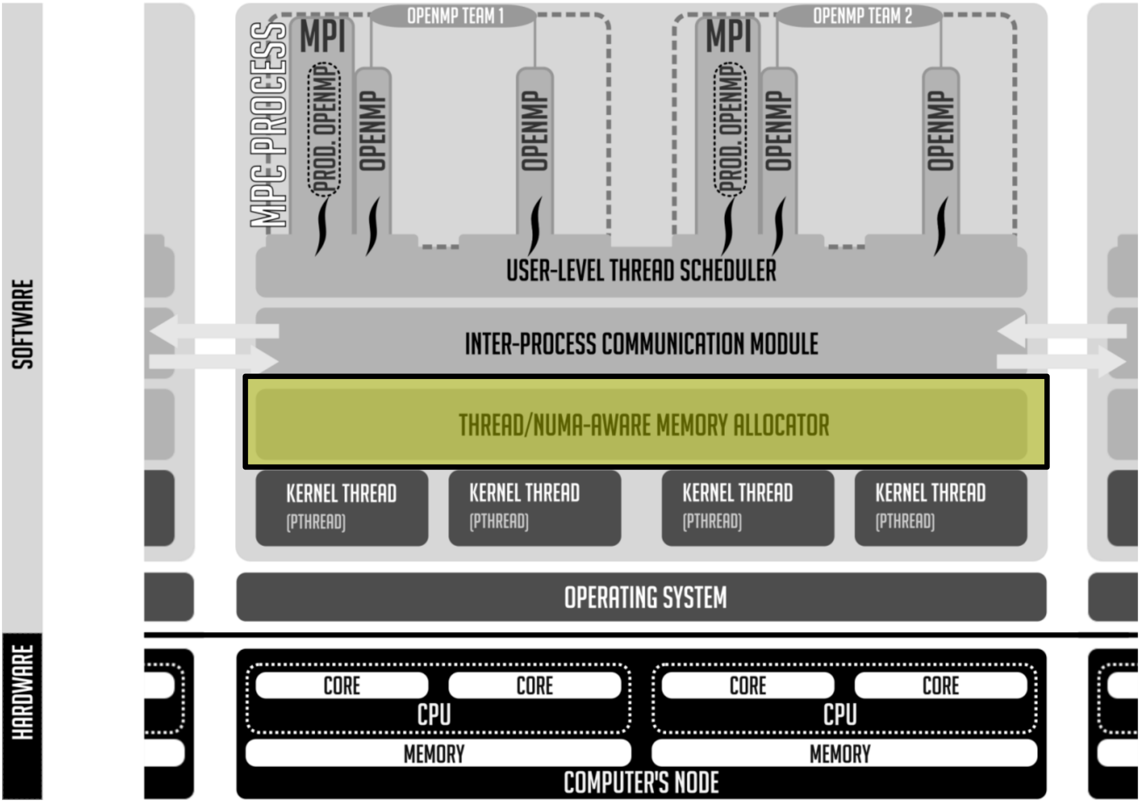
- … MPI, OpenMP, pthread implementations…
- … based on the same thread scheduler allowing interoperability, …
- …sharing the communication module for collaborative polling…



MPC Internals

- MPC is…

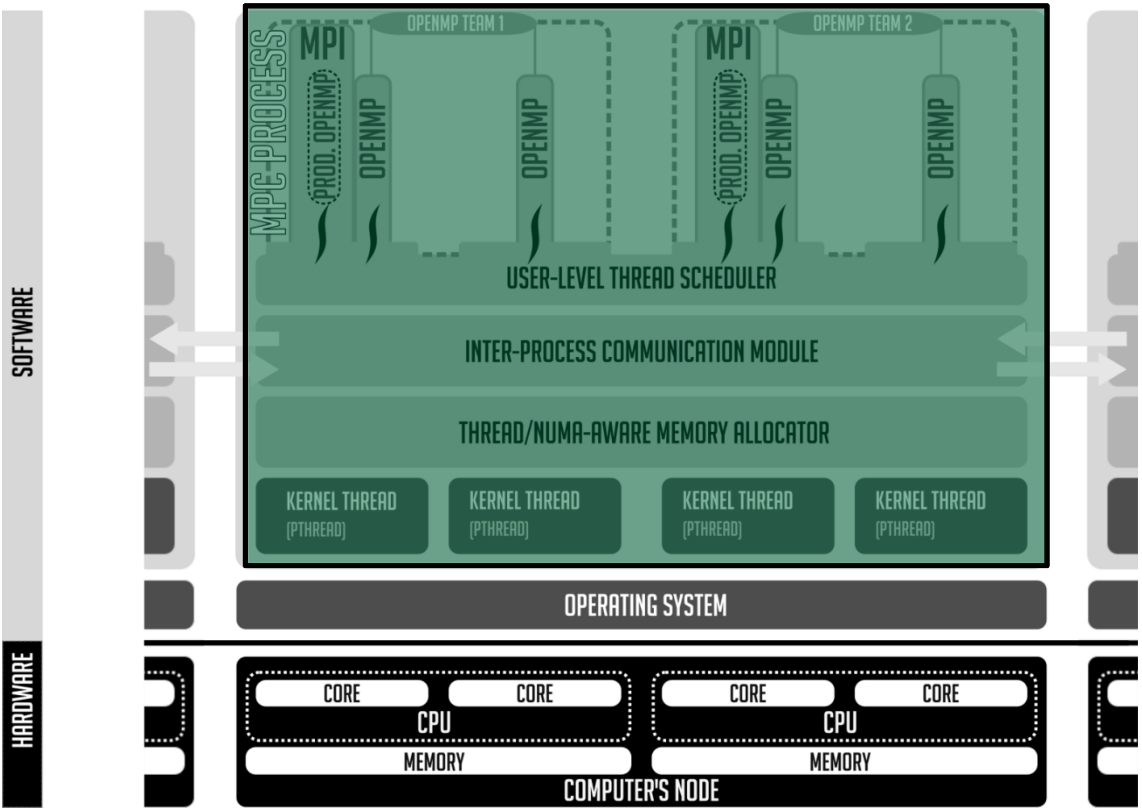
- … MPI, OpenMP, pthread implementations…
- … based on the same thread scheduler allowing interoperability, …
- …sharing the communication module for collaborative polling…
- …and a memory allocator avoiding NUMA effects…



MPC Internals

- MPC is…

- … MPI, OpenMP, pthread implementations…
- … based on the same thread scheduler allowing interoperability, …
- …sharing the communication module for collaborative polling…
- …and a memory allocator avoiding NUMA effects…
- …for **efficient hybrid MPI+X programming**



MPC – UNIFIED USER-LEVEL THREAD SCHEDULER

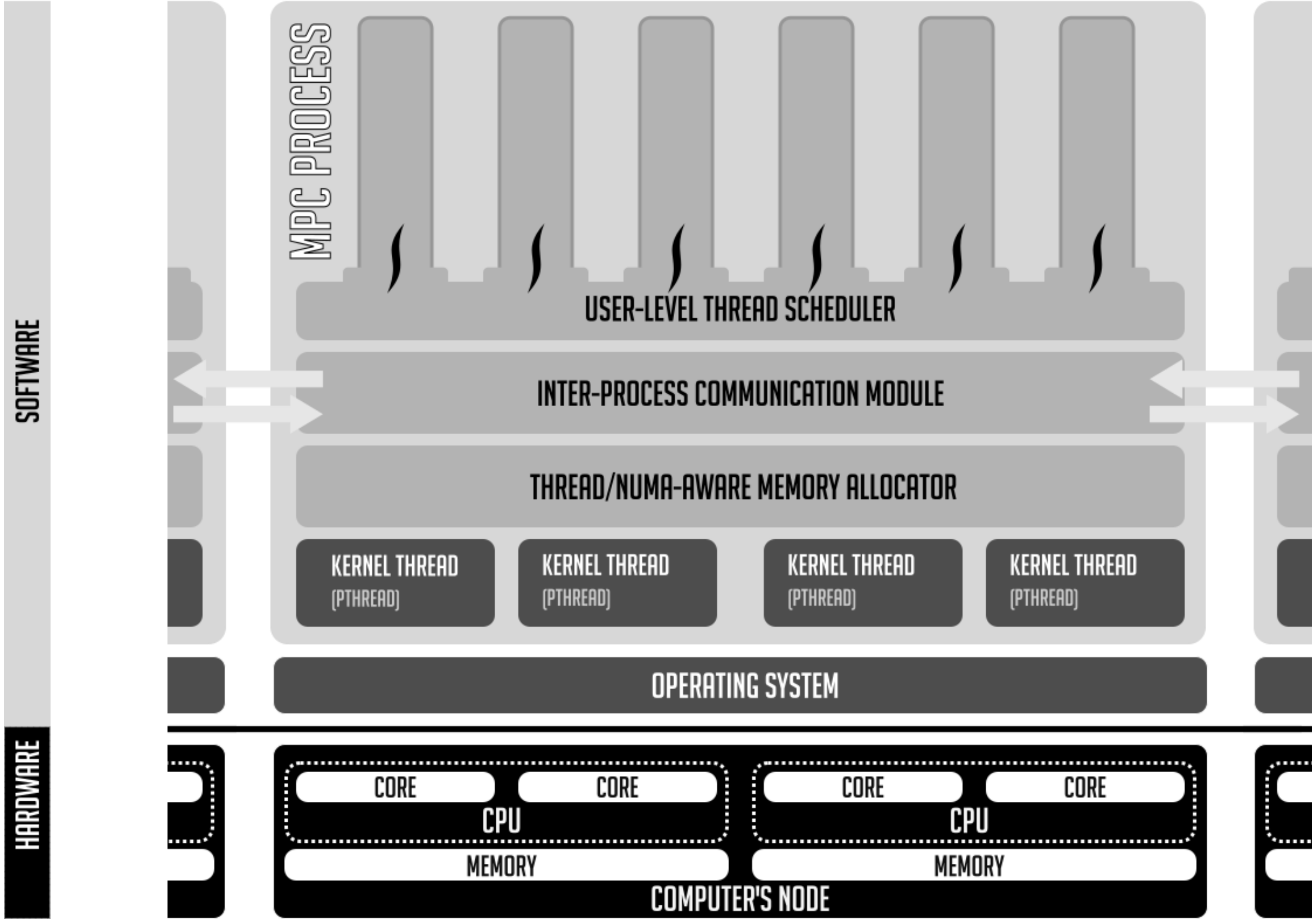
User-Level Thread Scheduler

Why user level threads ?

- Easier development
  - compare to kernel development
- Optimizations:
  - Keep only useful HPC functionalities
  - Less or no system calls (no system calls if no signal support)
- Portable:
  - OS independent
- Drawbacks:
  - Hard to debug: need specific debugger support (see tools section)
  - Architecture dependencies:
    - Optimized ASM context/switch, spinlocks, …
    - Topology detection and binding (easier thanks to HWLOC)

MPC User-Level Thread Scheduler

- MPC User-Level Thread Scheduler



MPC User-Level Thread Scheduler

MPC User-Level thread scheduler features

- MxN thread scheduler
  - M user-level threads > N kernel threads
- Optimizations
  - ASM context switches
  - Link with MPC memory allocator to ensure data locality
- Topological binding:
  - Static a priori distribution (MPI scatter/OpenMP compact)
  - On demand migration (link with memory allocator)
- MPI optimized scheduler
  - Internal dedicated task engine for message progression
- Optimized busy waiting policy
  - Use the thread scheduler to dynamically adapt busy waiting policy according to node workload
  - Busy waiting delegation to thread scheduler (e.g. “smart” mwait/futex)
- Modular approach in order to evaluate new scheduling policies
- No preemption

User-Level Thread Scheduler: 3rd Part Softwares

- Useful for use 3rd part tools/runtimes

- Features
  - Standard PThread API
  - Signal support
  - Futexes
  - No preemption

- Examples
  - Intel TBB (included in the MPC Framework package)
  - First port of Non Blocking Collectives (progression thread)

- Debugging
  - Provides a patched GDB
  - Support from Allinea DDT debugger

MPC – MPI IMPLEMENTATION

MPC – MPI: Purpose and Supported Features

- Goals
  - Smooth integration with multithreaded model
  - Low memory footprint
  - Deal with unbalanced workload

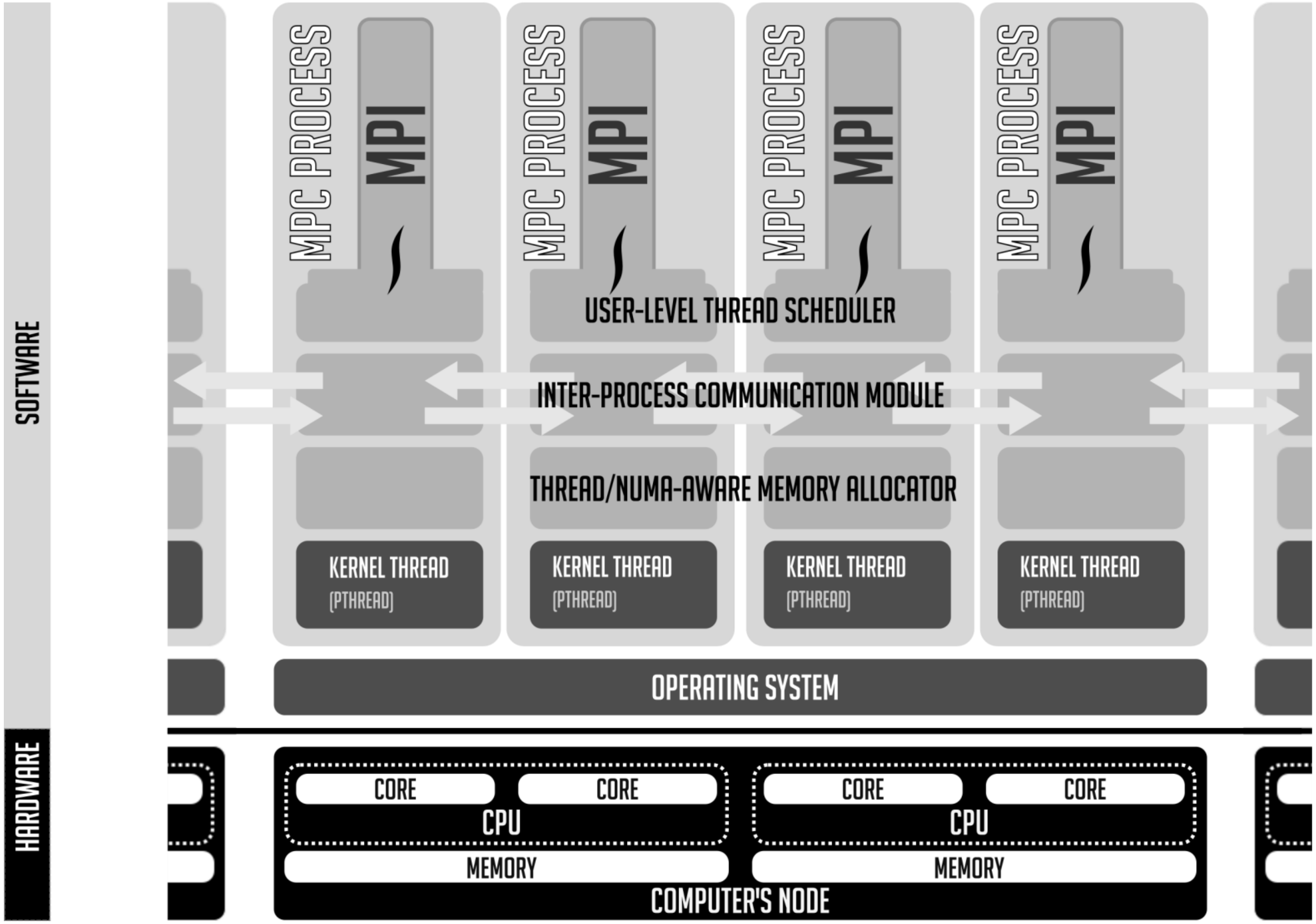
- Supported Features
  - Fully MPI 3.1 compliant
  - Handle up to MPI_THREAD_MULTIPLE level (max level)
  - MPI I/O
  - Non-blocking collectives
  - Neighborhood collectives

- Inter-node communications
  - TCP, InfiniBand & Portals4

- Tested up to 80,000 cores with various HPC codes

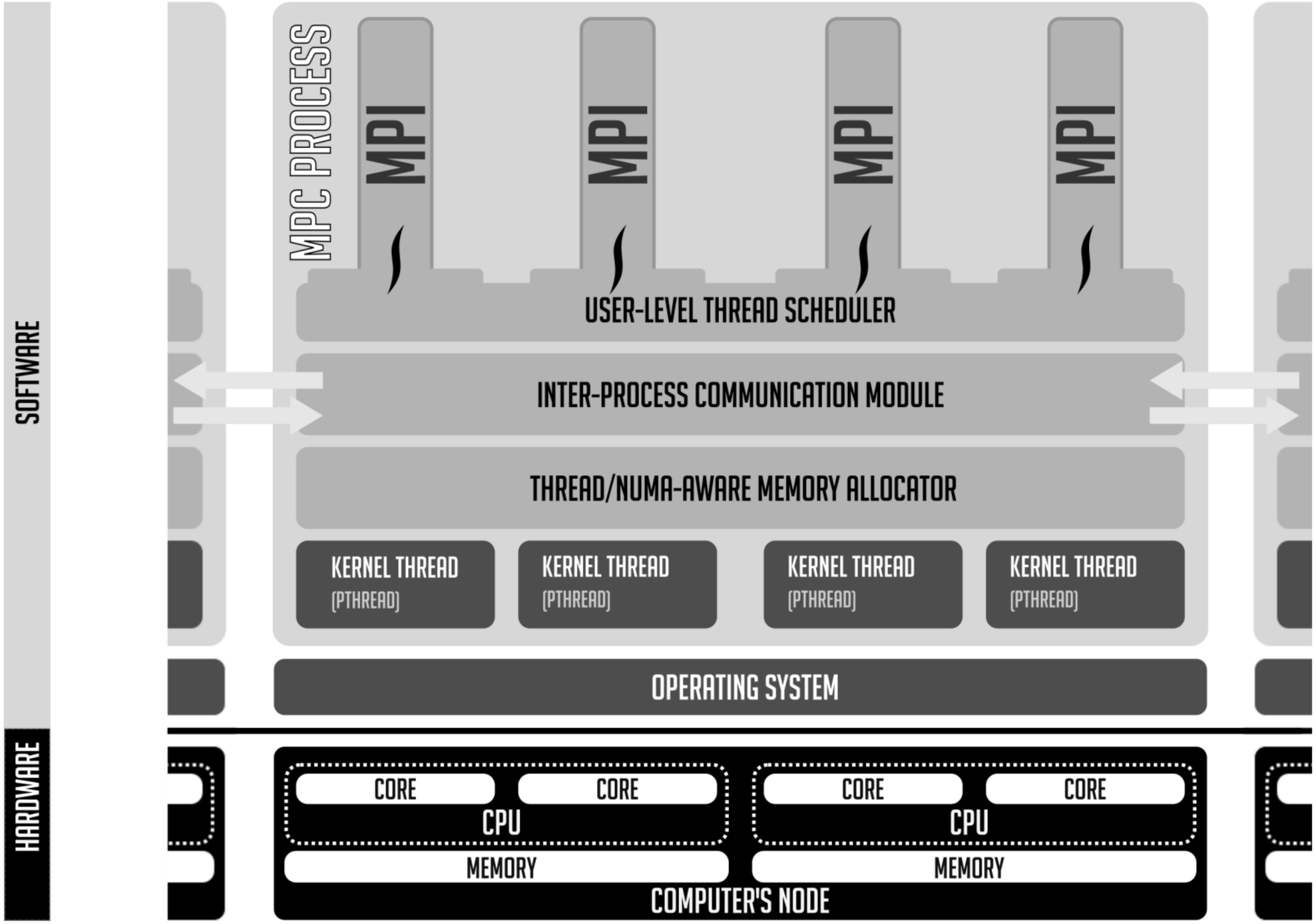
MPC – MPI Process-Based Execution Model

- Application with 4 MPI processes in 4 processes



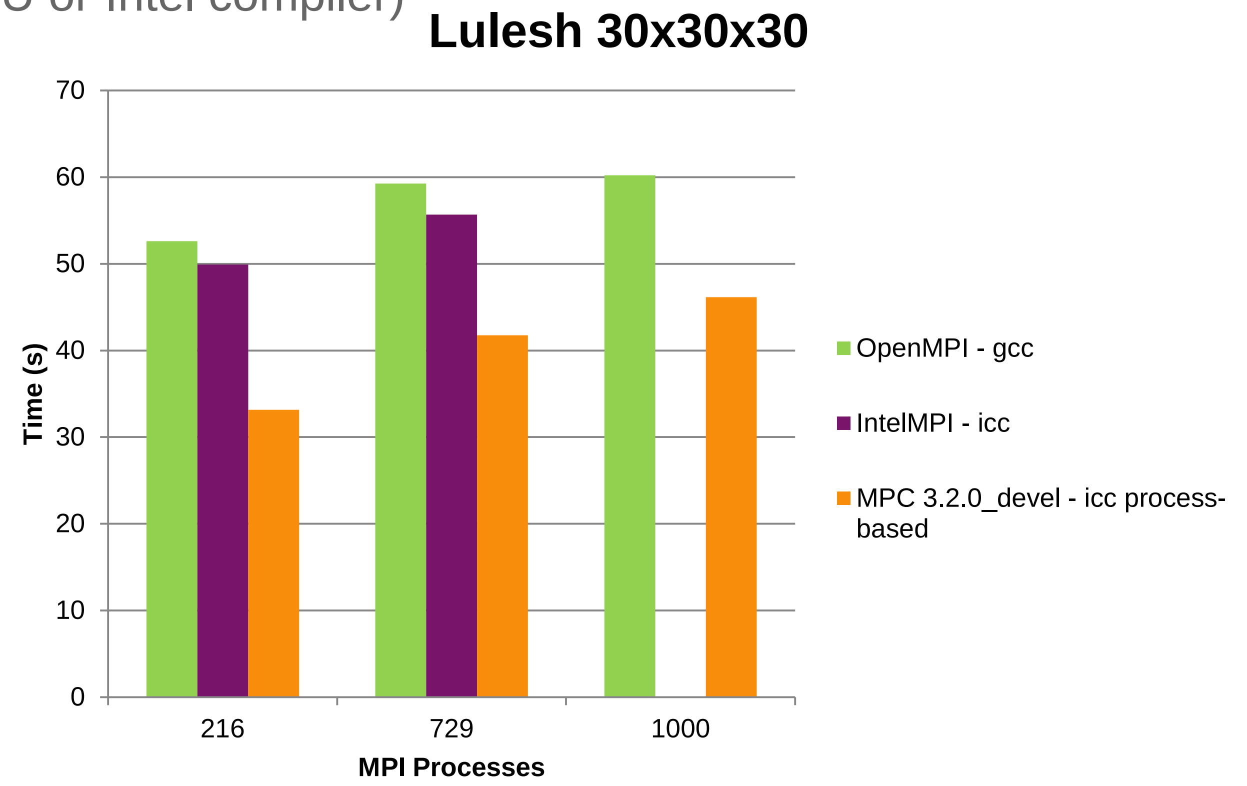
MPC – MPI Thread-Based Execution Model

- Application with 4 MPI processes in 1 process (with 4 threads)



MPI Experiment

- Architecture: Intel KNL
- Application: LULESH MPI
- Evaluation of multiple MPI implementations (OpenMPI, IntelMPI, MPC with either GNU or Intel compiler)



MPC – OPENMP IMPLEMENTATION

Runtime Stacking: OpenMP Layer

Impact of runtime stacking on OpenMP Layer

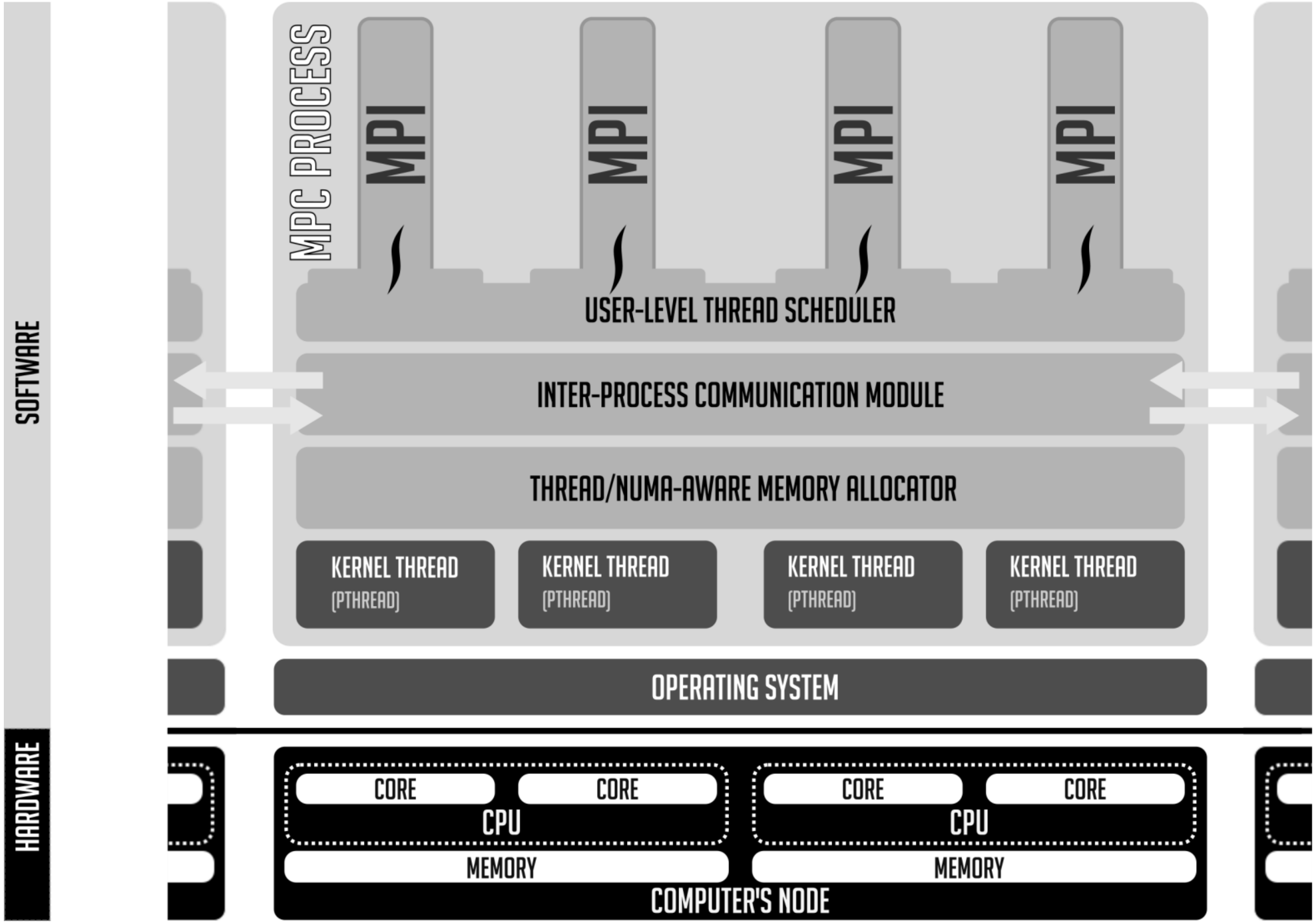
- Strong performance in fine-grain and coarse-grain approaches
  - Fine-grain
    - Optimization of launching/stopping a parallel region
    - Optimization of performing loop scheduling
  - Coarse-grain
    - Optimization of synchronization constructs (barrier, single, nowait…)
- Thread placement according to available cores (job manager + MPI runtime)
  - Oversubscribing → need to avoid busy waiting

OpenMP runtime design and implementation

- Goal: design of OpenMP runtime fully integrated into MPI runtime dealing with Granularity and Placement
- Implementation in MPC unified with optimized MPI layer

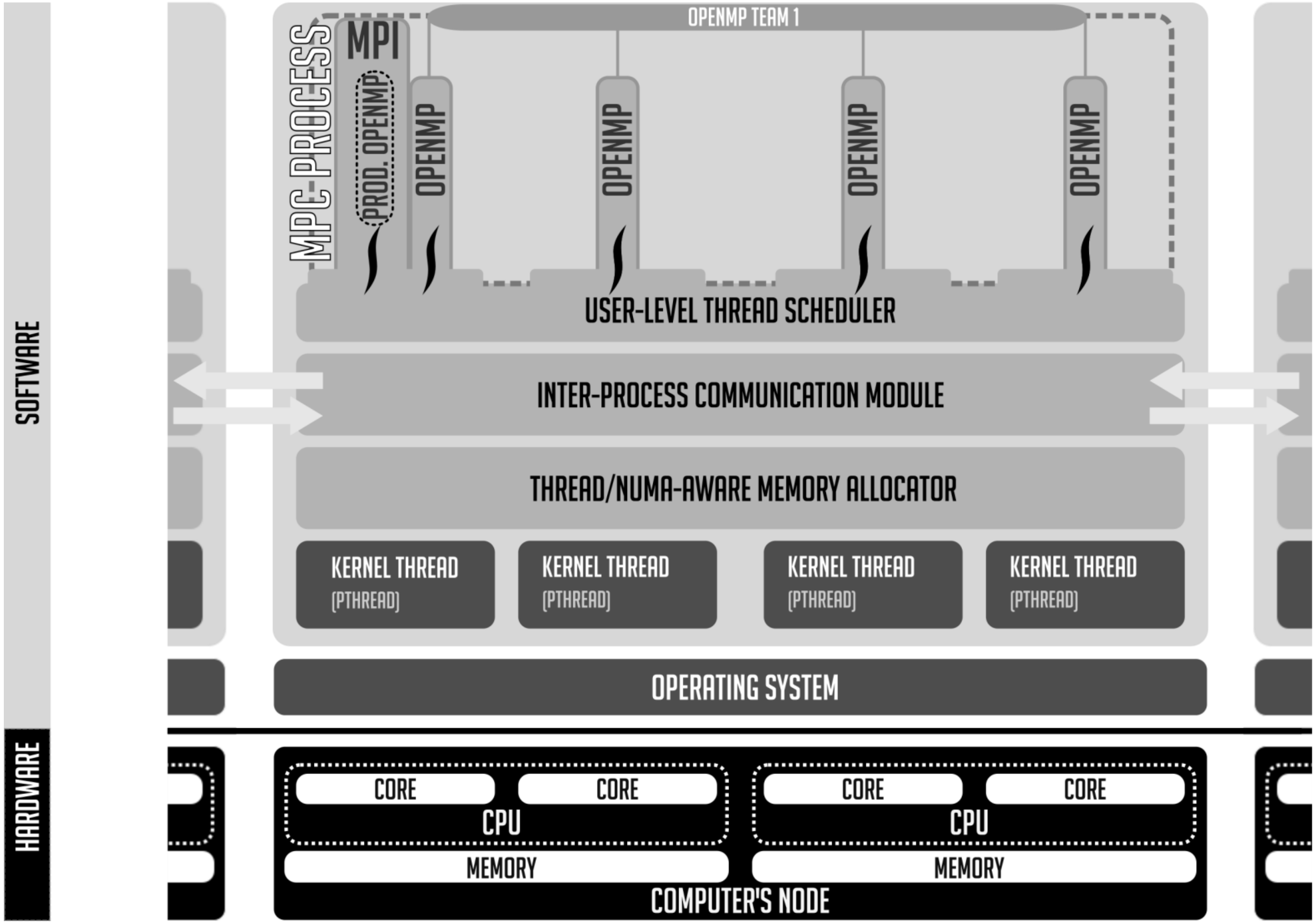
MPC Execution Model: Full MPI

- 4 MPI tasks in one process



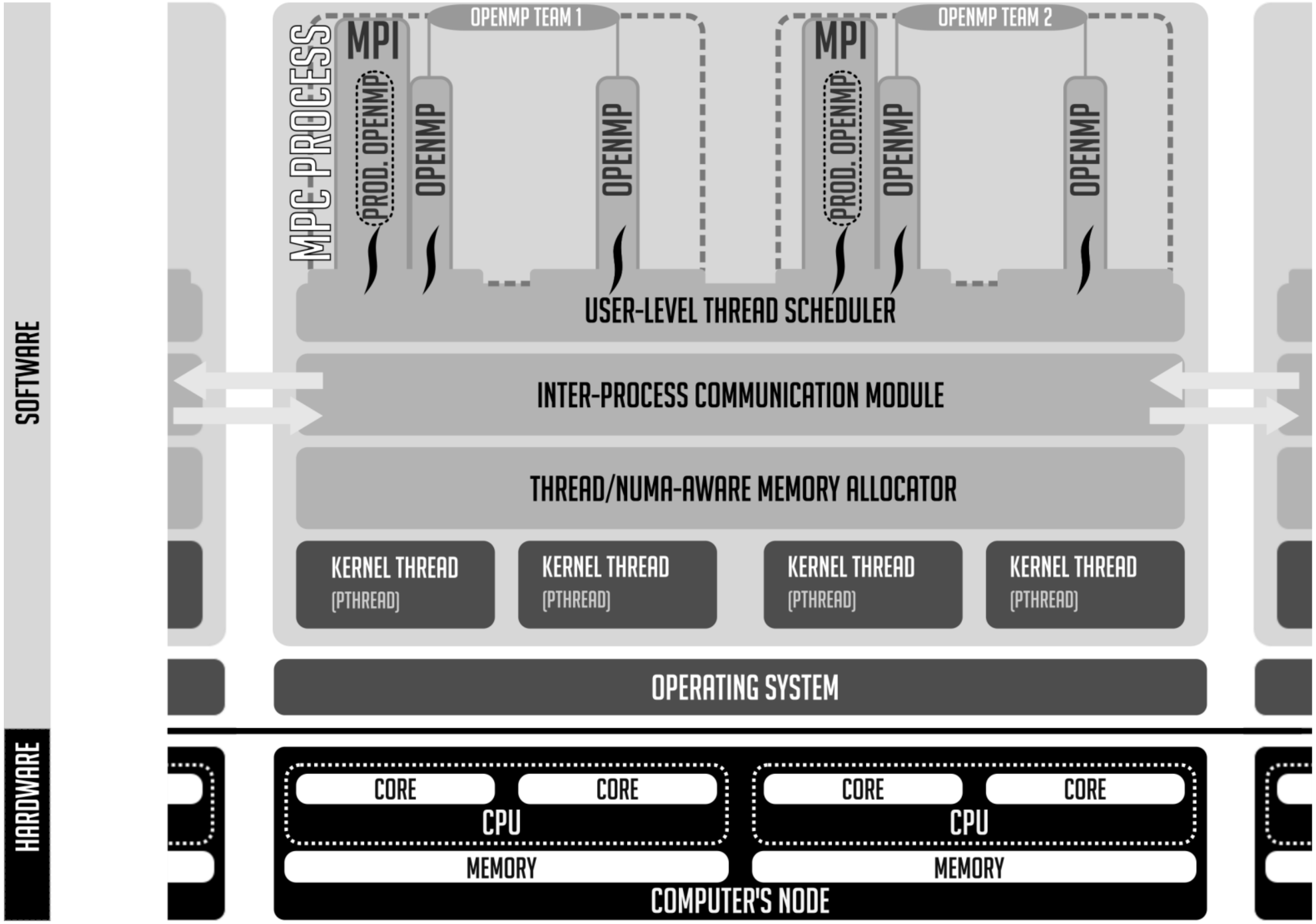
MPC Hybrid Execution Model: MPI+OpenMP (1)

- 1 MPI process + 4 OpenMP threads in one MPC process



MPC Hybrid Execution Model: MPI+OpenMP (2)

- 2 MPI processes + 4 OpenMP threads in one MPC process



User-Level Thread Scheduler: OpenMP

OpenMP threads are User-Level thread

- Smart binding thanks to information sharing with MPI runtime through thread scheduler
- Automatic MPI process and OpenMP thread placement on the node
  - Topology inheritance

Benefits

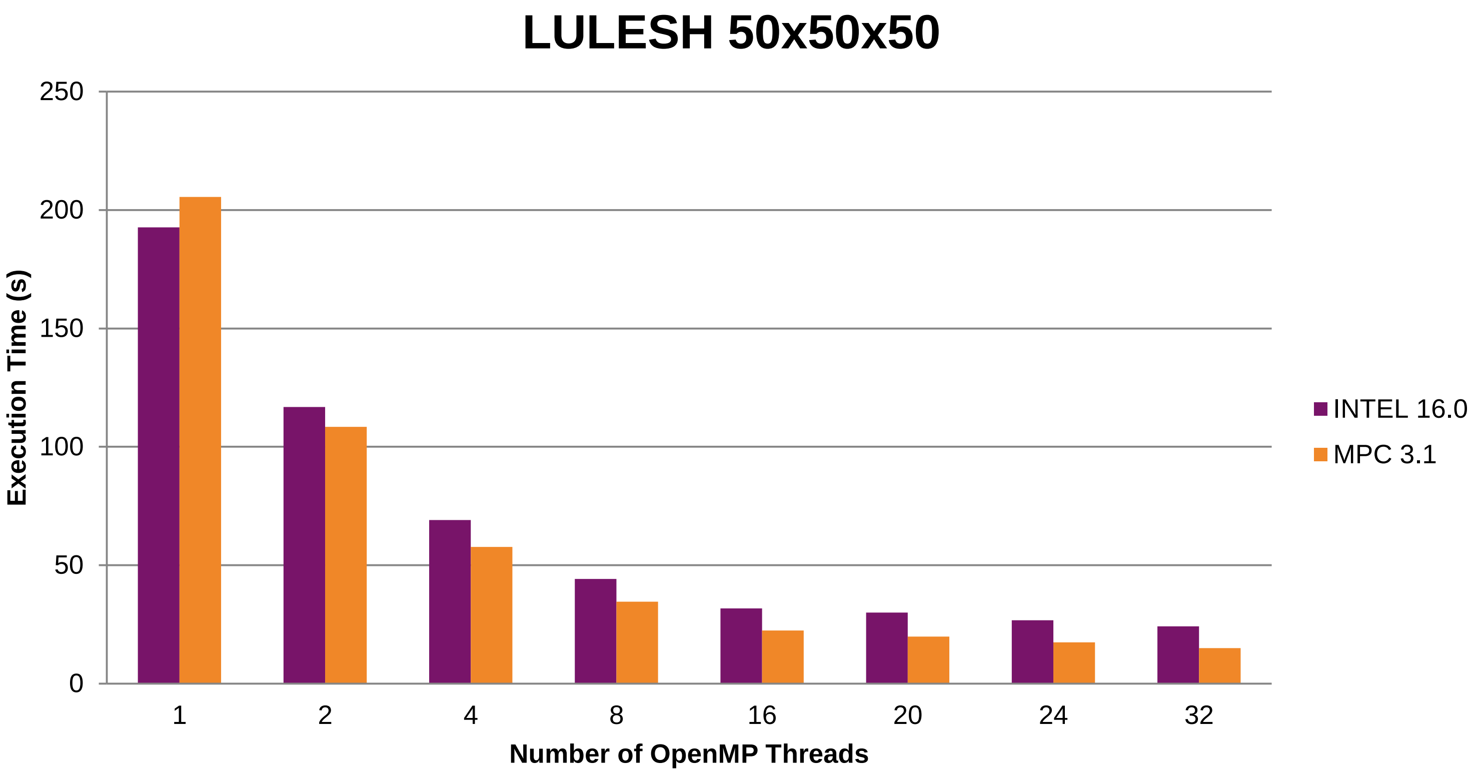
- Avoid busy waiting in multi-programming model context
  - Useful for fast thread wakeup/sleep (entering/leaving parallel regions, …)
- Data locality (link scheduler ⇔ memory allocator)

Drawbacks

- Unable to use standard TLS for #pragma threadprivate
  - ➢ Solution with extended-TLS

OpenMP Experiment

- Architecture: dual-socket 16-core Haswell
- Application: LULESH OpenMP compiled with Intel 16
- Evaluation of Intel OpenMP runtime vs. MPC OpenMP runtime (same compiler)



MPC – EXTENDED TLS AND AUTOMATIC PRIVATIZATION

Process virtualization

- Use threads (e.g. MPI processes) instead of OS processes
  - Convert standard OS process to MPC thread
  - Encapsulate OS process within threads

- Difficulties
  - How to handle global variables
  - Unable to use standard TLS for #pragma threadprivate
  - Deal with non-thread safe libraries
    - User libraries: HDF5, …
    - System libraries: getopt, …
    - Compiler libraries: libgfortran,

➔ Automatic privatization thanks to compiler support

➔ Provide patched version if not compatible yet with automatic privatization

➔ Provide non-thread safe system libraries

Extended TLS Application: Automatic Privatization

- Solution: **Automatic privatization**

- Automatically convert any MPI code for thread-based MPI compliance
- Duplicate each global variable

- Design & Implementation

- Completely transparent to the user
- When parsing or creating a new global variable: flag it as MPI thread-local
  - #pragma threadprivate : flag it as OpenMP thread-local
- Generate runtime calls to access such variables (extension of TLS mechanism)
  - Linker optimization for reduce overhead of global variable access

- Compiler support

- New option to GCC C/C++/Fortran compiler (`-fmpc-privatize`)
  - Patched GCC provided with MPC (4.8.0, on going on GCC 4.9.x and 5.x)
- ICC support automatic privatization with same flag (`-fmpc-privatize`)
  - ICC 15.0.2 and later
- On-going work for PGI compiler support

Official Support of MPC in Intel Compiler Man Page

- Official support of MPC since Intel Compiler v15.02
  - Extracted from the icc/icpc/ifort man page

```
> man icc
...
Feature: Privatization of static data for the MPC unified parallel runtime Requirement:
Appropriate elements of the MultiProcessor Computing (MPC) framework For more information,
see http://mpc.sourceforge.net/
...
-fmpc-privatize (L*X only) / -fno-mpc-privatize (L*X only)
Enables or disables privatization of all static data for the MultiProcessor Computing
environment (MPC) unified parallel runtime.
Architecture Restriction: Only available on Intel(R) 64 architecture
Arguments: None
Default: -fno-mpc-privatize
The privatization of all static data for the MPC unified parallel runtime is disabled.
Description:
This option enables or disables privatization of all static data for the MultiProcessor
Computing environment (MPC) unified parallel runtime.
Option -fmpc-privatize causes calls to extended thread-local-storage (TLS) resolution, run-
time routines that are not supported on standard Linux* OS distributions.
This option requires installation of another product. For more information, see Feature
Requirements.
```

Conclusion & Future Work

Overview of MPC

- MPC
  - Unified user-level thread scheduler
    - With an inter-process communication module and a thread/numa-aware allocator
- Programming models
  - Provide widely spread standards: MPI 3.1, OpenMP 3.1+, Pthread, TBB
  - Available at http://mpc.hpcframework.com (version 3.2 available)
- Runtime optimization
  - Provide unified runtime for MPI + X applications
  - New mechanism to mix thread-based programming models: Extended TLS
- Support
  - Architecture: x86, x86_64, MIC, arm (in progress)
  - Network: TCP, Infiniband and Portals4 with multi-rail
  - Resource manager: Slurm & Hydra
- Tools
  - Use HWLOC to detect topology
  - Debugger support (Allinea DDT), Profiling
  - Compiler support (Intel, GCC, PGI)

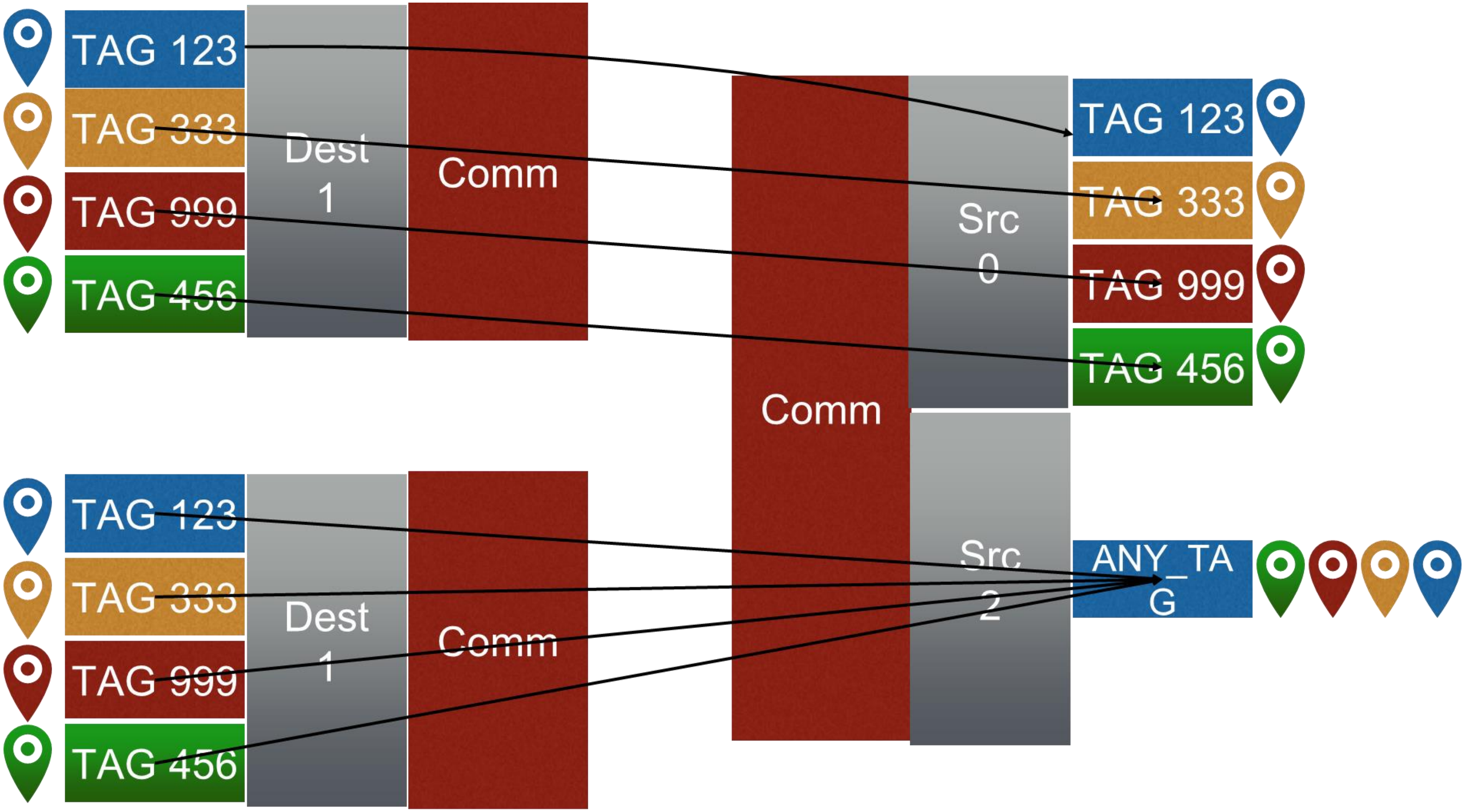
SUPPORTING ASYNCHRONOUS PROGRESSION WITH HARDWARE OFFLOADING

HARDWARE MESSAGE MATCHING

MPI and Messages

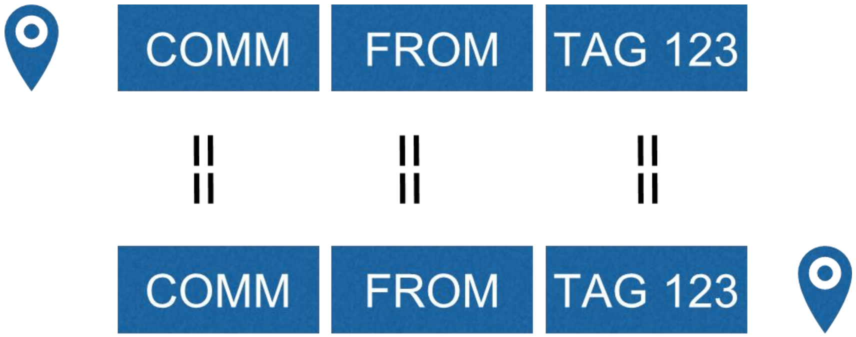
- As seen on these production MPI (using default configuration) overlap is not to be taken for granted
  - Even when using « asynchronous » calls.
  - It is important to progress these calls.
- Reasons for this are multiple (non exhaustively):
  - Optimization for latency (noise from progress)
  - No standard support for thread-multiple
    (multiple entities progressing the same requests)
  - Need for remote information when progressing messages
    (aka matching)

MPI Matching



MPI Matching

- Before copying any data, MPI has to make sure that the source and target buffers are correctly identified
  - In the continuity of their corresponding Send and Recv.



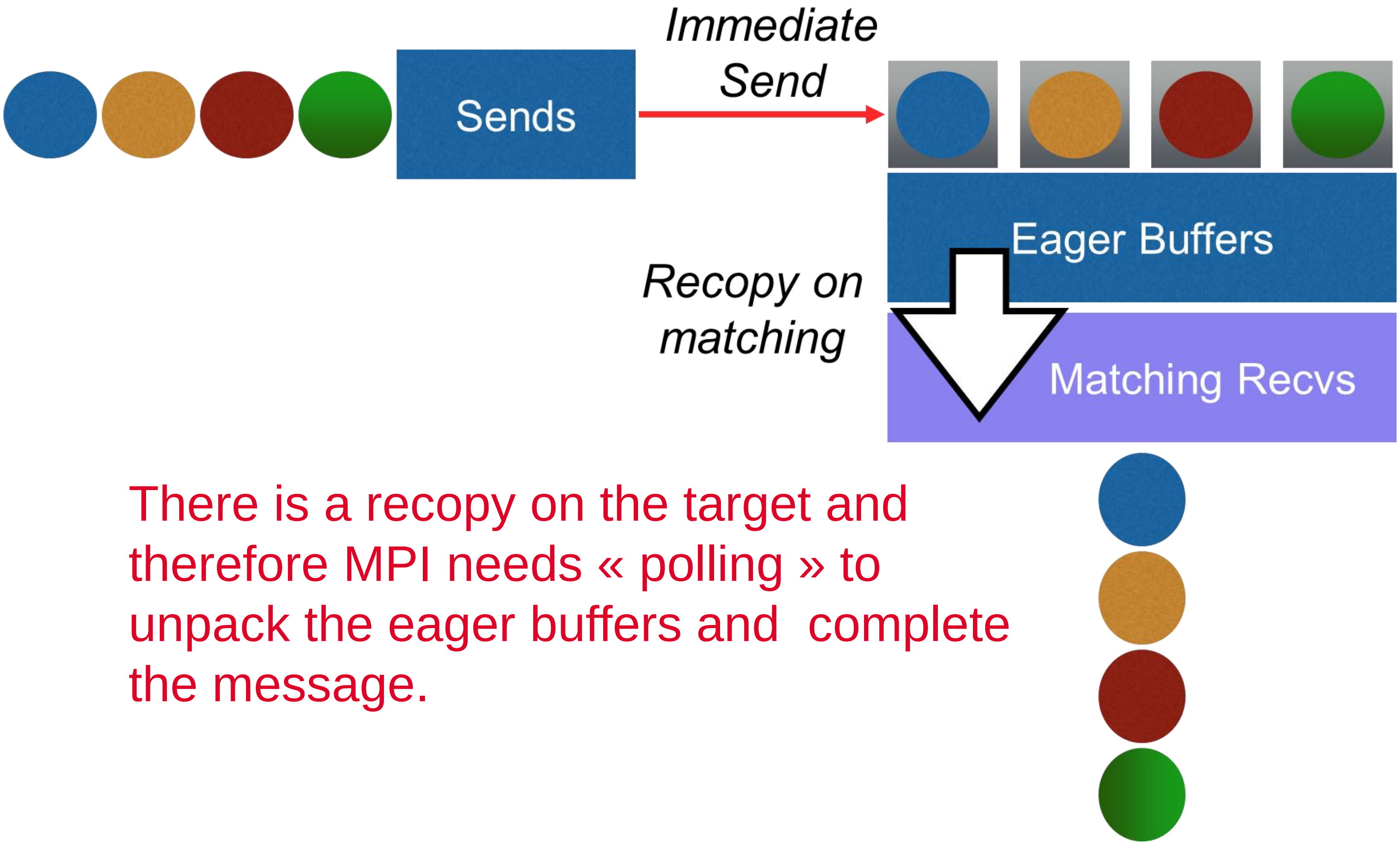
COMM | FROM | TAG 123

COMM | FROM | TAG 123

  - It makes « single copy » (aka « zero-copy » or « zero-recopy ») messages more difficult to implement.

- In practice, messages involve some recopy overhead:
  - Latency optimized eager (recopied on target)
  - Rendez-vous protocol requires a previous message to initiate the synchronization.

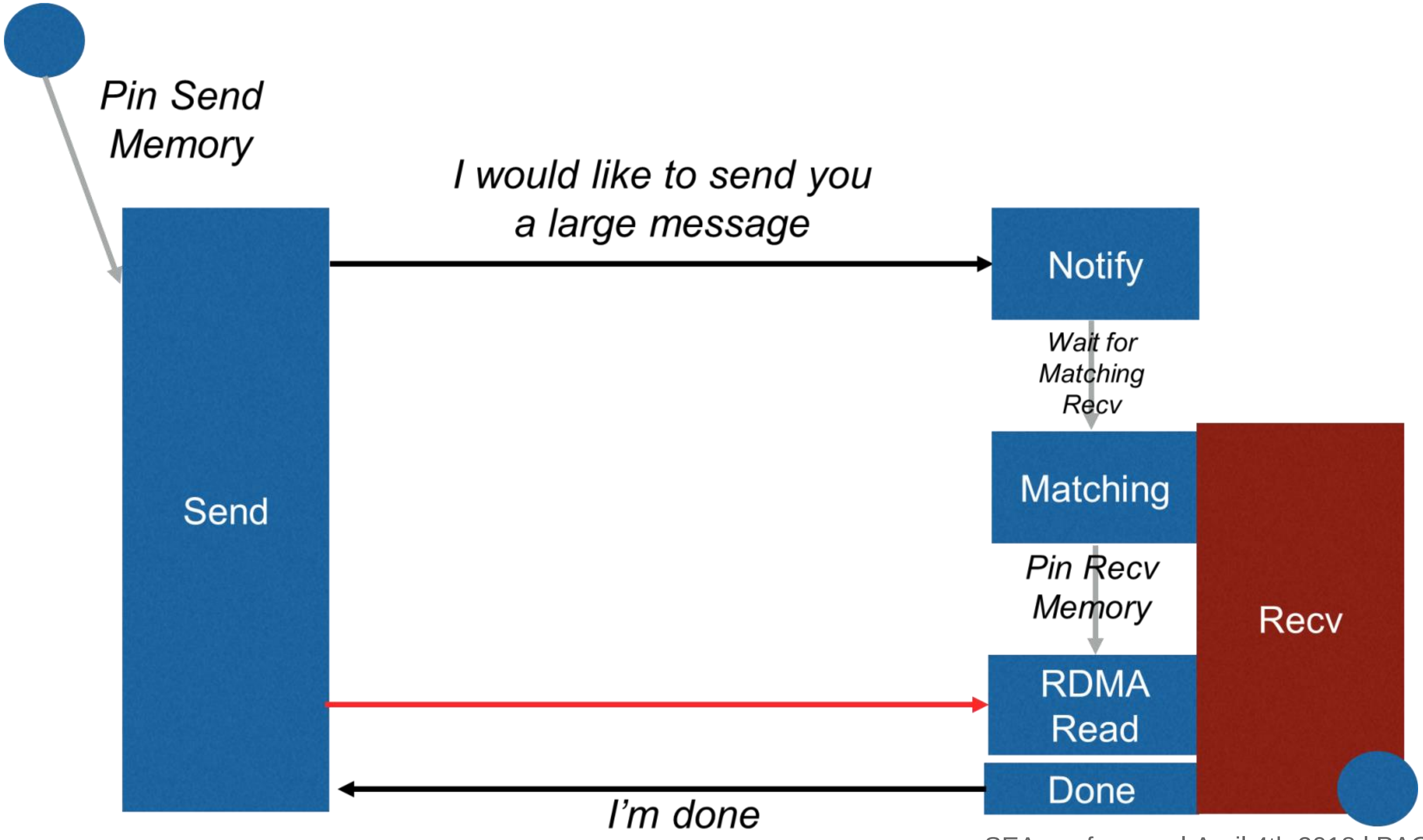
Eager Messages



- There is a recopy on the target and therefore MPI needs « polling » to unpack the eager buffers and complete the message.

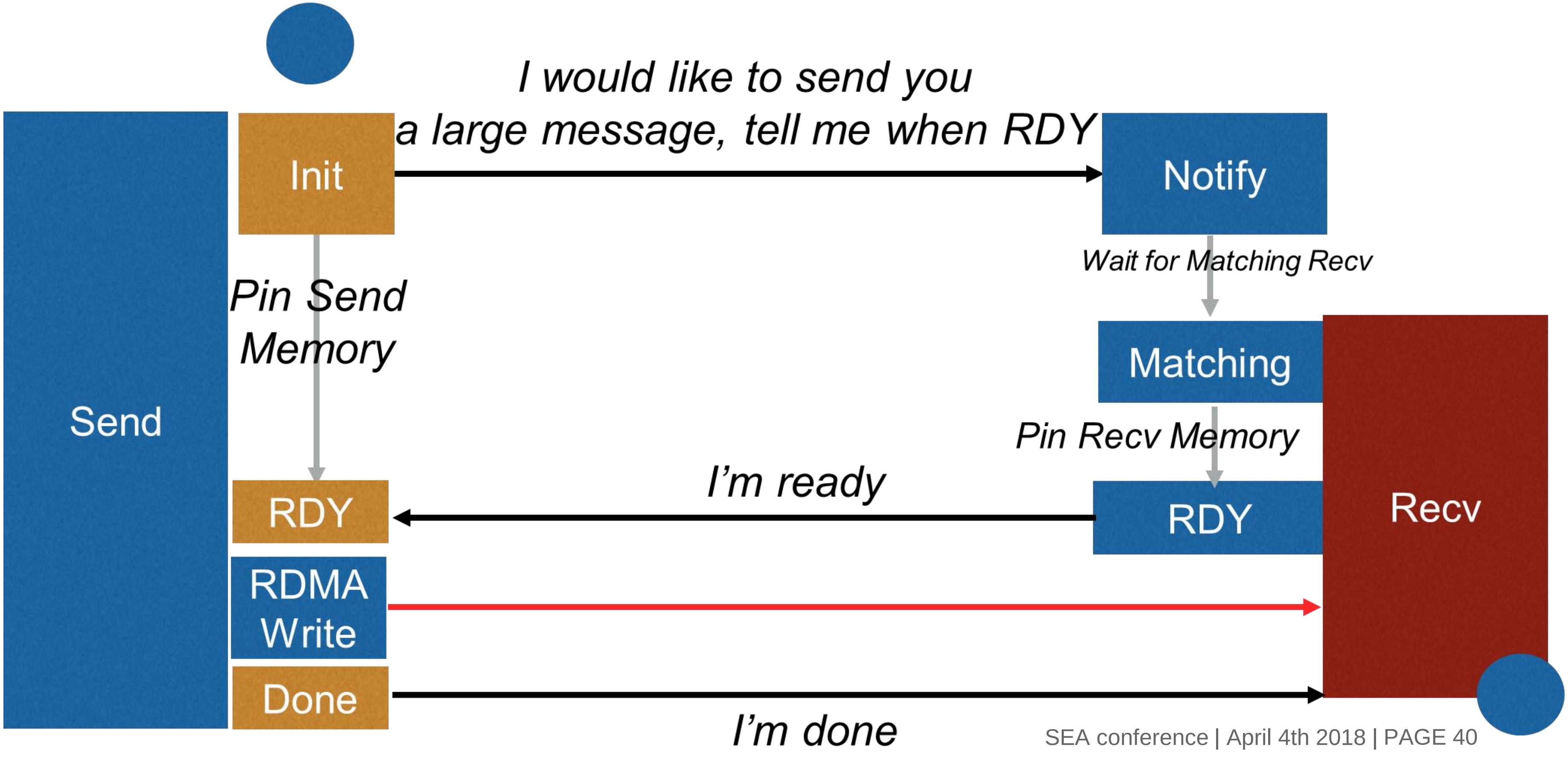
Rendez-Vous Read Example

- Payload movement is « zero-copy » but pinning and meta-data exchange to setup the rendez-vous needs polling.



Rendez-Vous Read Example

- Payload movement is « zero-copy » but pinning and meta-data exchange to setup the rendez-vous needs polling.



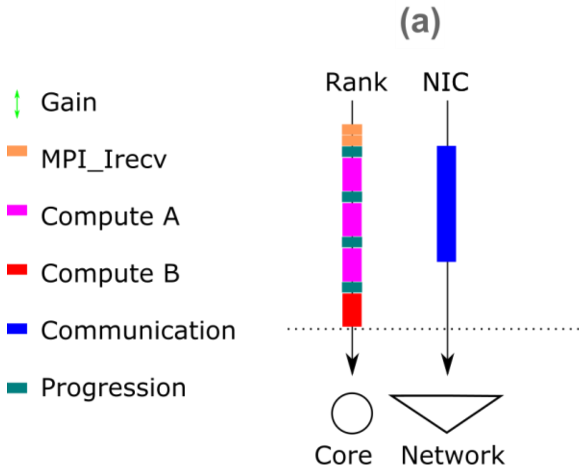
The progression issue

- How to progress messages?

The progression issue

- How to progress messages?

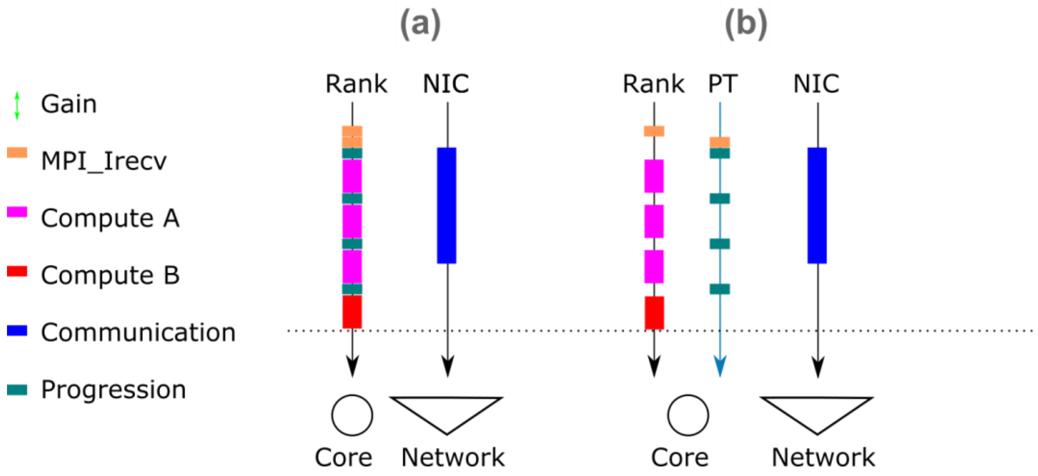
- a) on the same thread
  - need to call functions to progress the message



The progression issue

- How to progress messages?

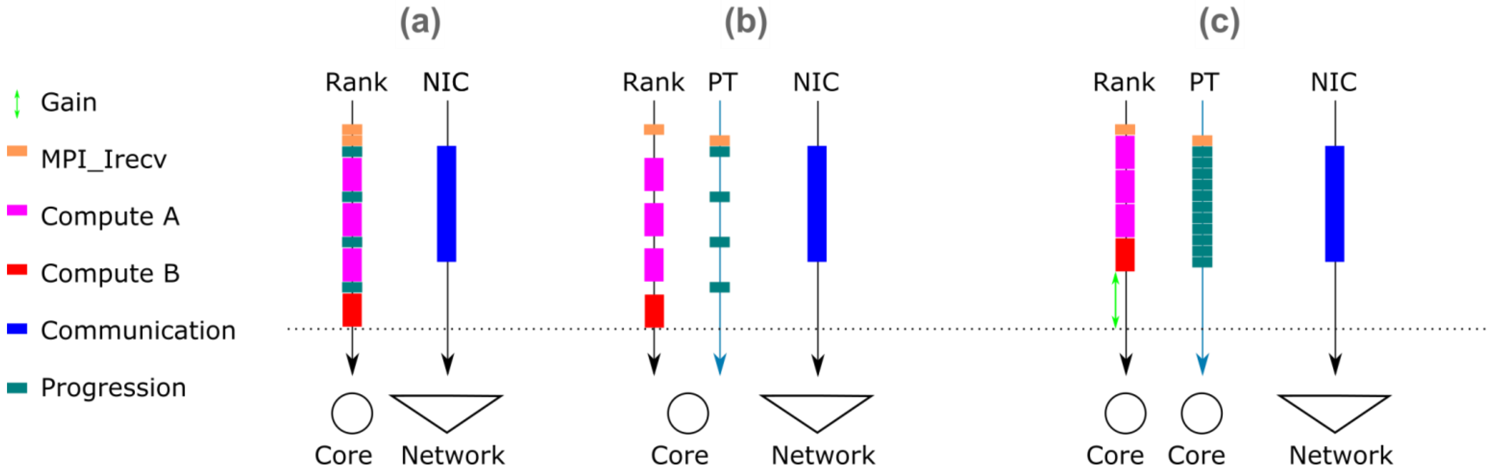
- a) on the same thread
  - need to call functions to progress the message
- b) on another thread
  - only progression on thread but competition for resources



The progression issue

- How to progress messages?

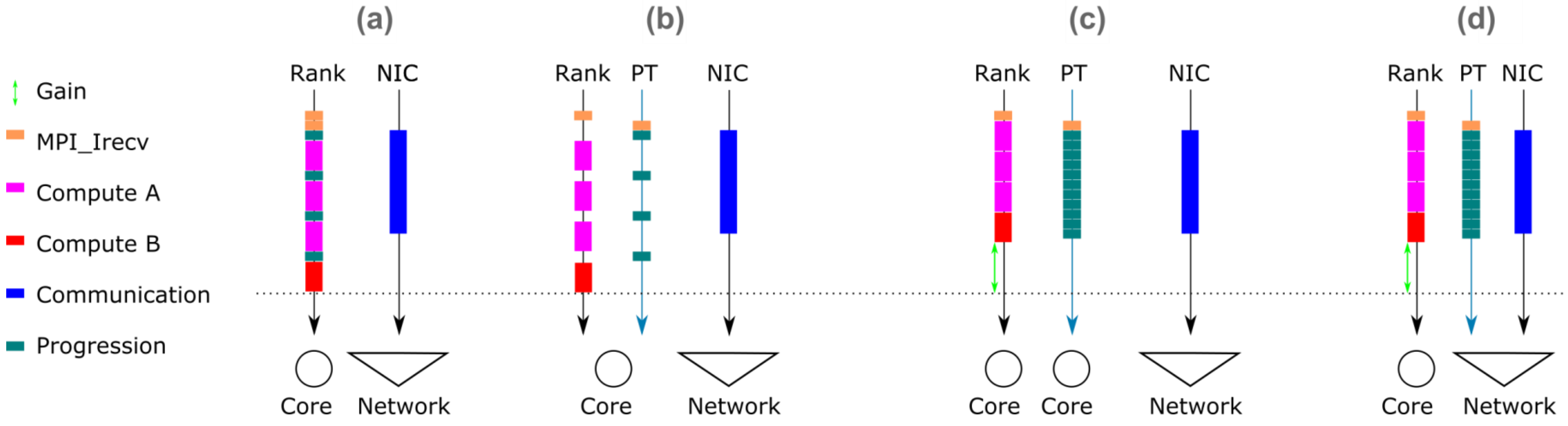
- a) on the same thread
  - need to call functions to progress the message
- b) on another thread
  - only progression on thread but competition for resources
- c) on another core
  - start and finish communications as soon as possible



The progression issue

- How to progress messages?

- a) on the same thread
  - need to call functions to progress the message
- b) on another thread
  - only progression on thread but competition for resources
- c) on another core
  - start and finish communications as soon as possible
- d) on the network card
  - if the functionality is available



MPI Matching and Overlap

- Matching prevents modern HPC network cards to fully express their potential

  - Requires information only available on the target process

- A software mechanism has to be involved to associate two MPI buffers

  - « meta-data » have to be exchanged prior to moving the actual message data.
  - This is (one of) the reasons why MPI needs to poll its message queues to enable efficient progress.

- A possible solution to this is to implement MPI matching in the hardware

  - As we are now going to present in the context of the Portals 4 message layer.

Portals 4

- Portals 4

- A high-performance messaging interface
- Developed by Sandia National Laboratories since 2017.
- Provides enhanced semantics when compared to low-level Verbs (HW Matching) and features enabling the implementation of PGAS languages (such as Put notification).

The Bull Exascale Interconnect (BXI)

- The Bull Exascale Interconnect is the first implementation of the Portals 4 communication model.

The BXI card

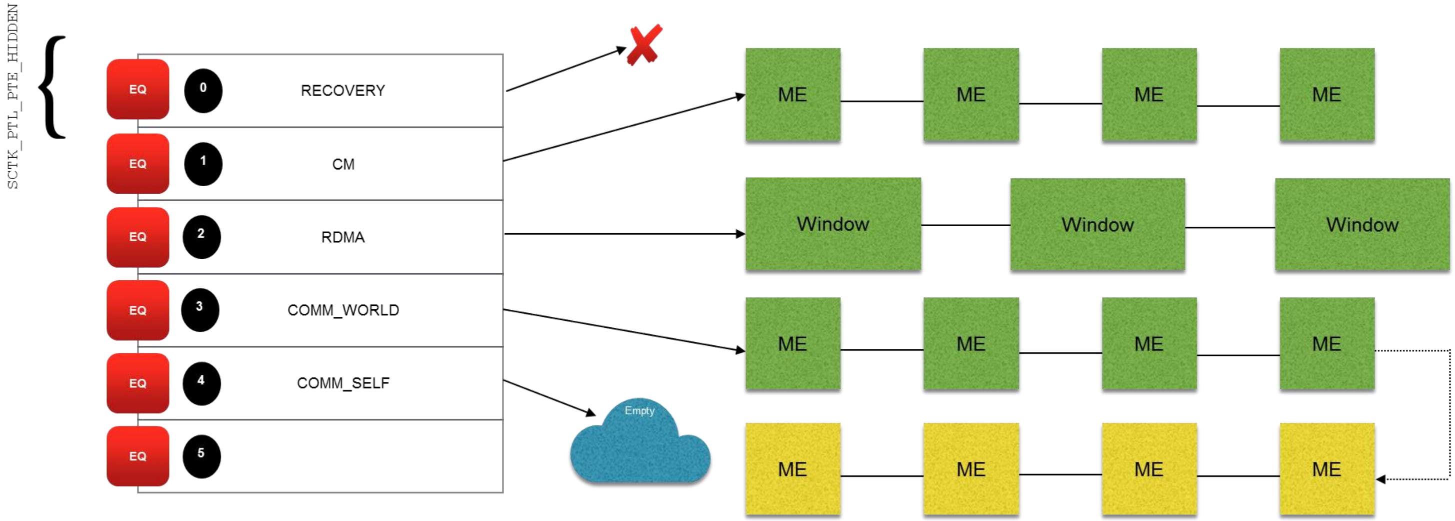
A BXI Switch

Bull
atos technologies

BXI

Portals 4 in the MPC Runtime

- The Portals 4 communication model with the help of the BXI has been integrated to the MPC runtime.
  - To do so MPC relied on a mapping of communicators (round-robin) to the Portal 4 table
  - ME=Matching List Entry



== overflow

Portals 4 MatchBit in MPC

- The 64 bits matchbit is handled as follows in MPC:

  - No need for communicator as it is mapped to the table entry (currently limits communicator number to table size)
  - 32 bits for the tag
  - 16 bits for the source rank
  - 16 bits for the sequence number

Source 16 bits	Sequence ID 16 bits	TAG 32 bits

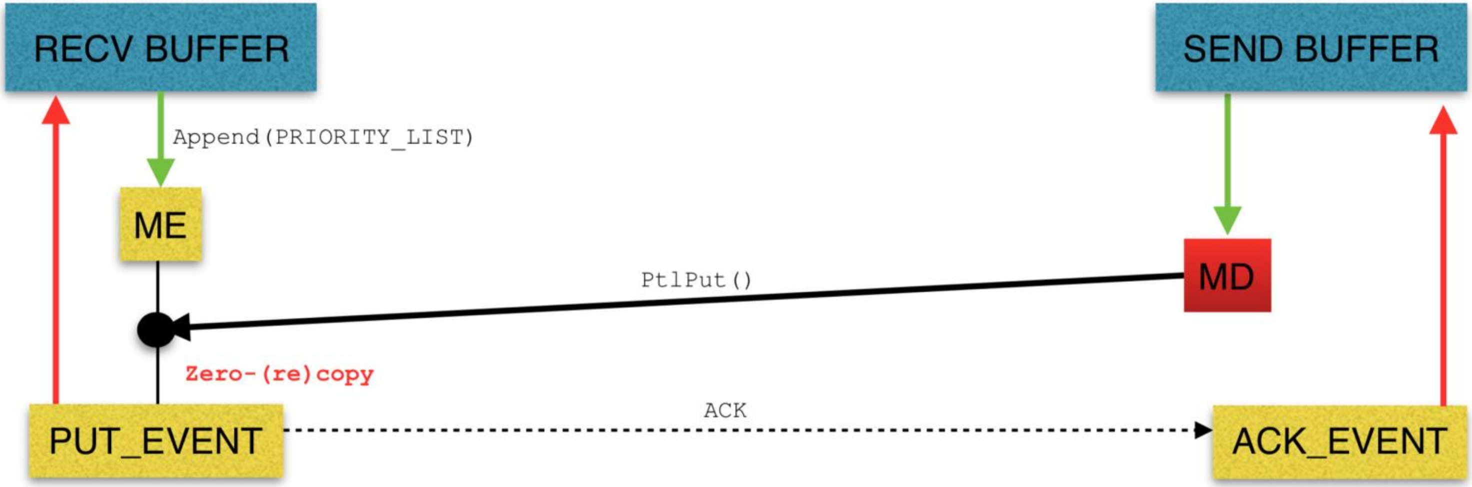
- It allows a full offload of the matching in the HCA

  - Matching a Send and a Recv without requiring CPU arbitration.

Hardware Matching

- A key advantage of Portals 4’s design is that it allows the matching to take place in the hardware.

- It is therefore possible to route a message end-to-end in an asynchronous manner.
- MD = Memory Descriptor



- Target message address is resolved within the hardware, allowing the put event to **directly** target the user-space MPI buffer.

- All steps involved in progressing the message are done by the HCA.

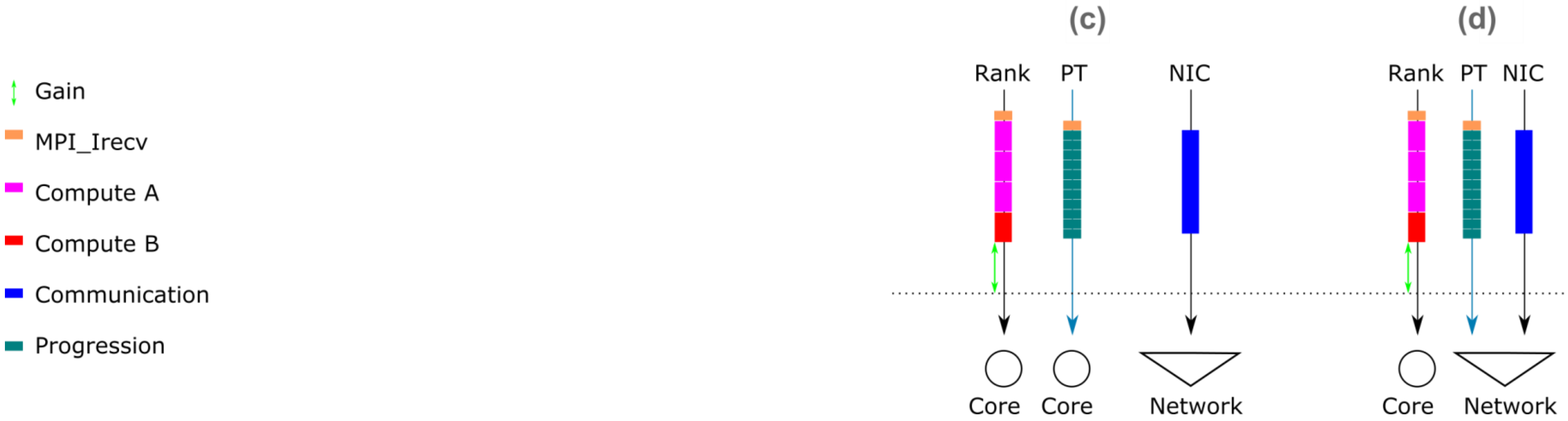
TOWARDS EFFICIENT HARDWARE COLLECTIVE OFFLOAD

Offloading collectives

- In addition to message matching, it will also be possible to offload collective communications on the network card
  - All point-to-point communications involved in the collective
  - /!\ Also handling the unfolding of the collective algorithm for all MPI processes
- Problems that may occur
  - Collective algorithm for all MPI processes are handle by one core
  - Not only communications but also compute power is now required and may take time on the core
  - May infer artificial serialization/ordering between messages

Offloading

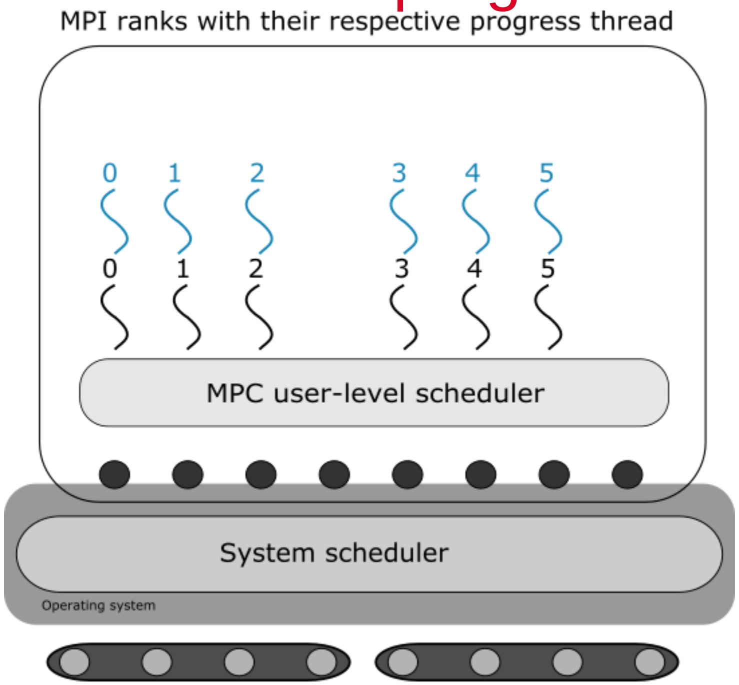
- To simulate this behavior, we offloaded collective handling to reserved core
  - Seems similar to collective messages being grouped on network card processor

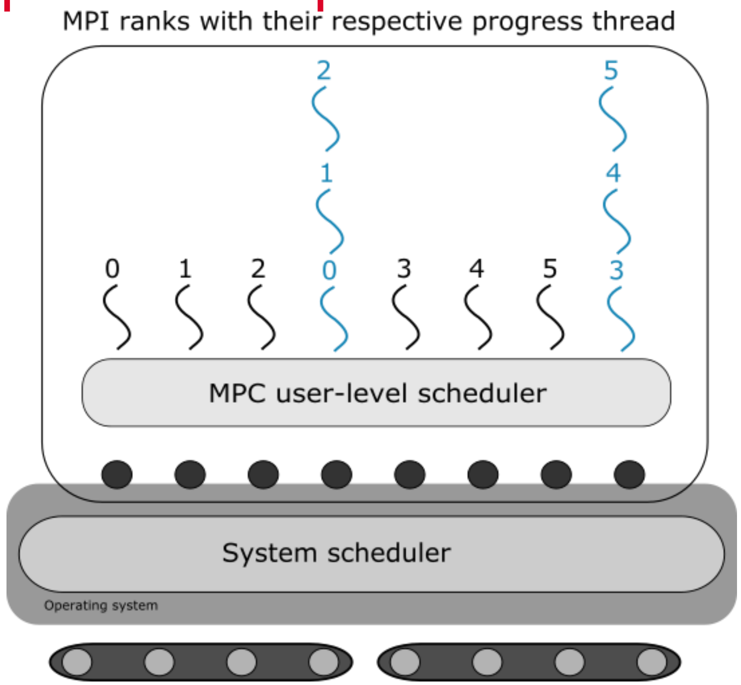


- What happens when messages are folded on a small number of reserved core

Dedicated resources for progress threads

- MPC thread-based implementation allowed us to try different progress thread placement policies



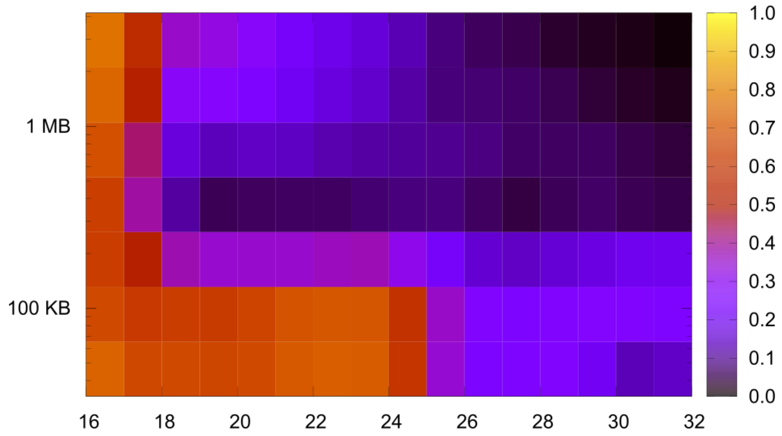


- Left: threads are bound to the same core (can be another hyperthread)
- Right: threads are bound to the closest available core

Dedicated resources for progress threads

- Having dedicated resources increase overlap ratio, and may also provide better performance

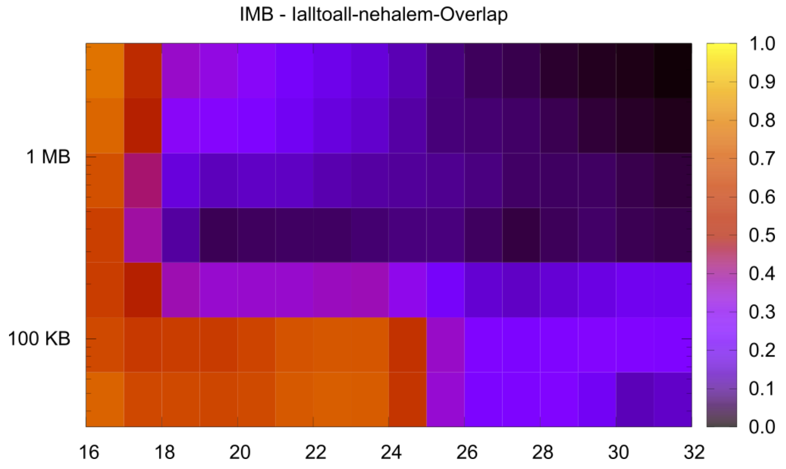
IMB - Ialltoall-nehalem-Overlap

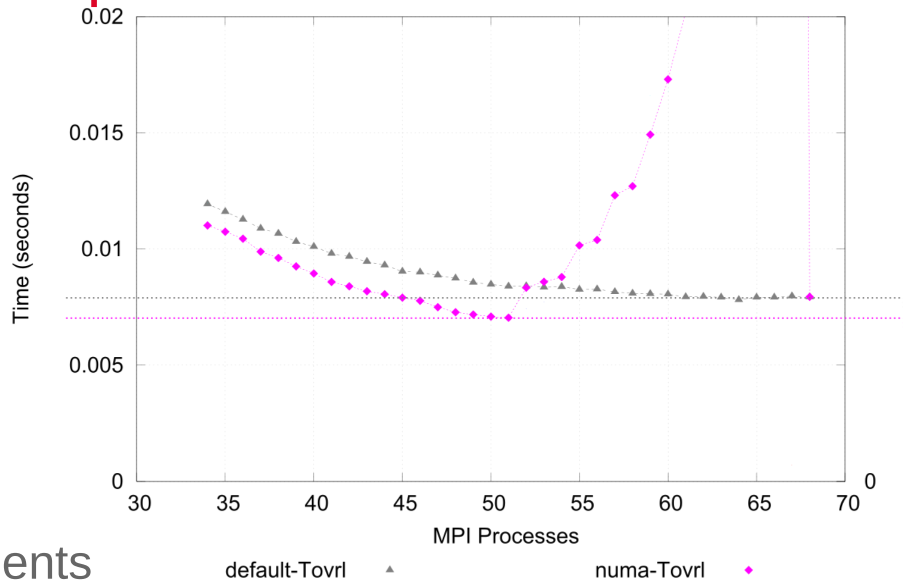


- Left: overlap with different placements
  - 16: best - all progress threads have their own cores
  - 24: good compromise – 3 progress threads per available core
  - >24: some progress threads are bound to the core of their MPI processes

Dedicated resources for progress threads

- Having dedicated resources increase overlap ratio, and may also provide better performance



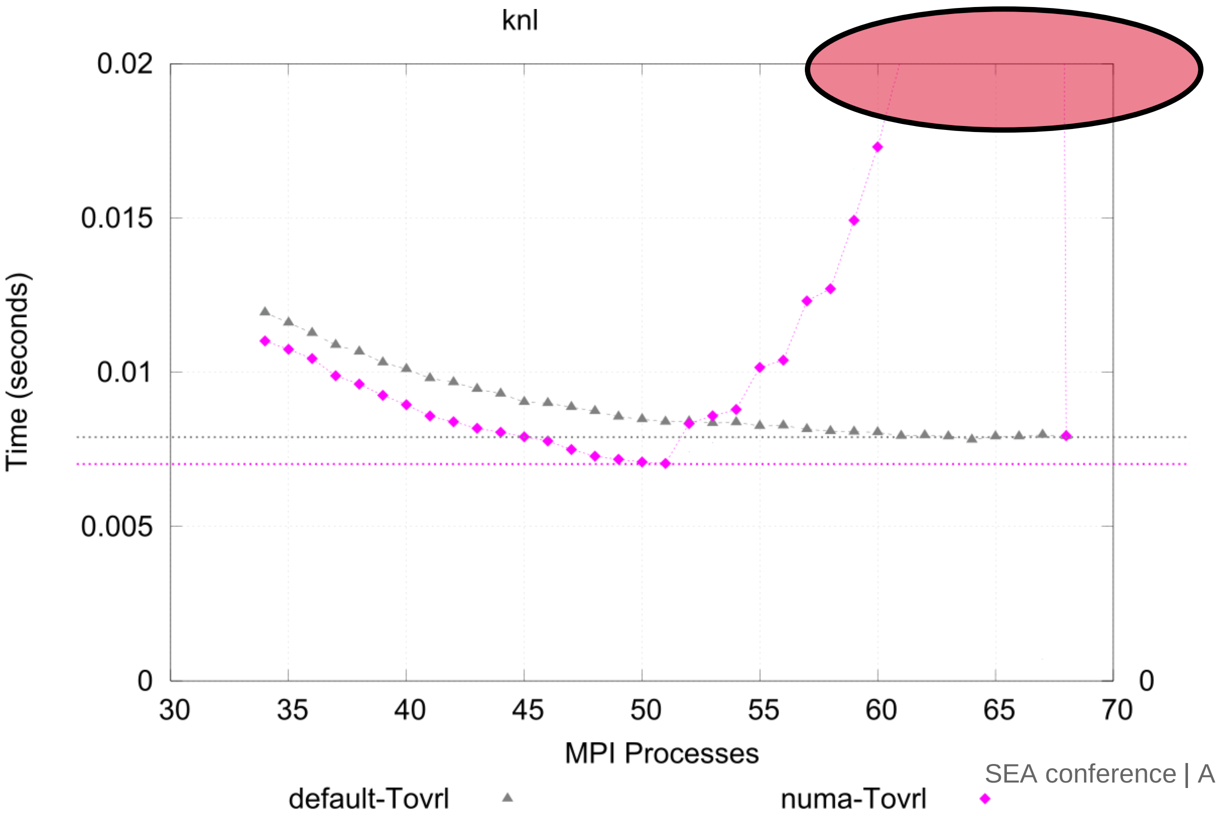


- Left: overlap with different placements
  - 16: best - all progress threads have their own cores
  - 24: good compromise – 3 progress threads per available core
  - >24: some progress threads are bound to the core of their MPI processes

- Right: alltoall performance on KNL
  - Grey: regular alltoall – Purple: ialltoall with thread placement according to left
  - 3 PTs per core provides overall best performances

Dedicated resources for progress threads

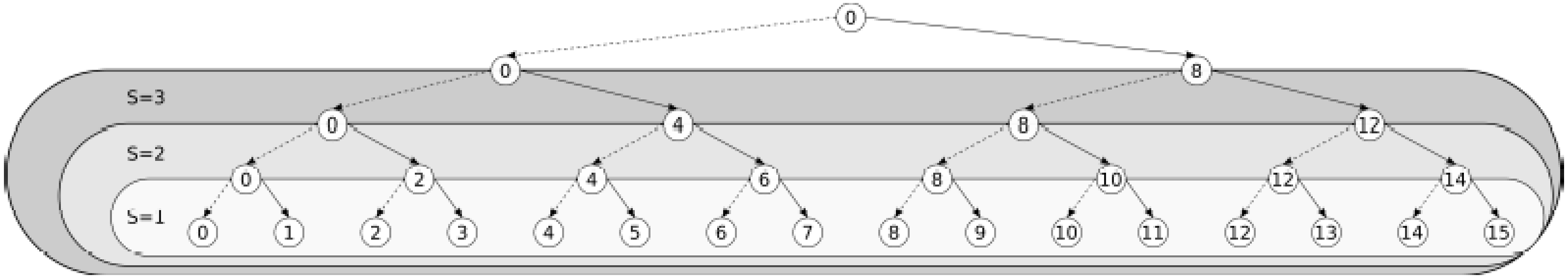
- Huge impact when all communications are folded on few cores
  - Especially for intra-nodes comm
  - Similar to what will happen when collectives will be offloaded to NIC
  - May still not be a viable solution for full MPI or MPI+MPI applications

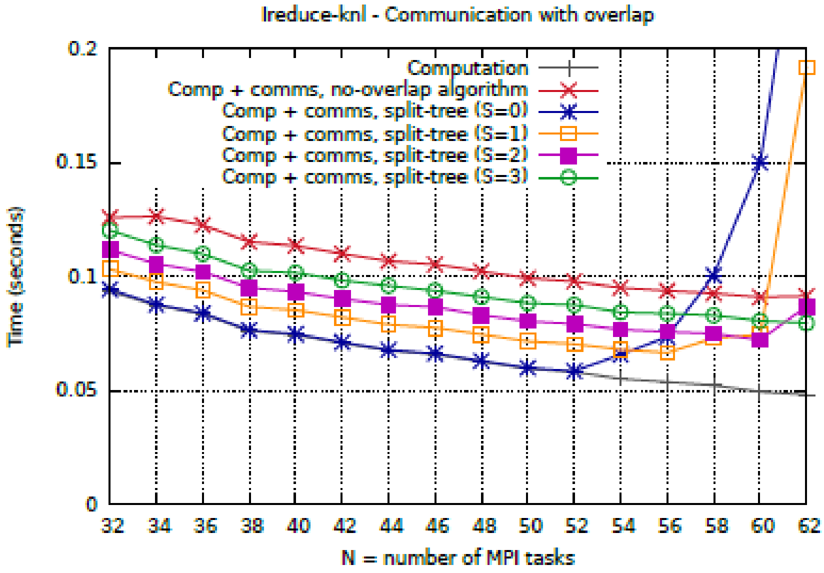


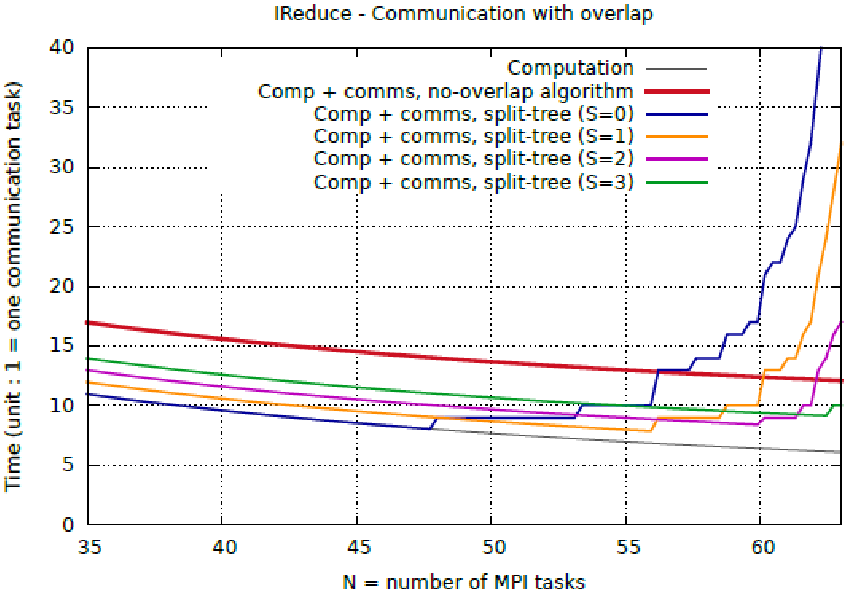
A solution: split-tree to reduce PTs cumul overhead

- Do the “heavy”parts of the tree on the MPI processes

- Try to reduce the overhead of having numerous PTs on few cores

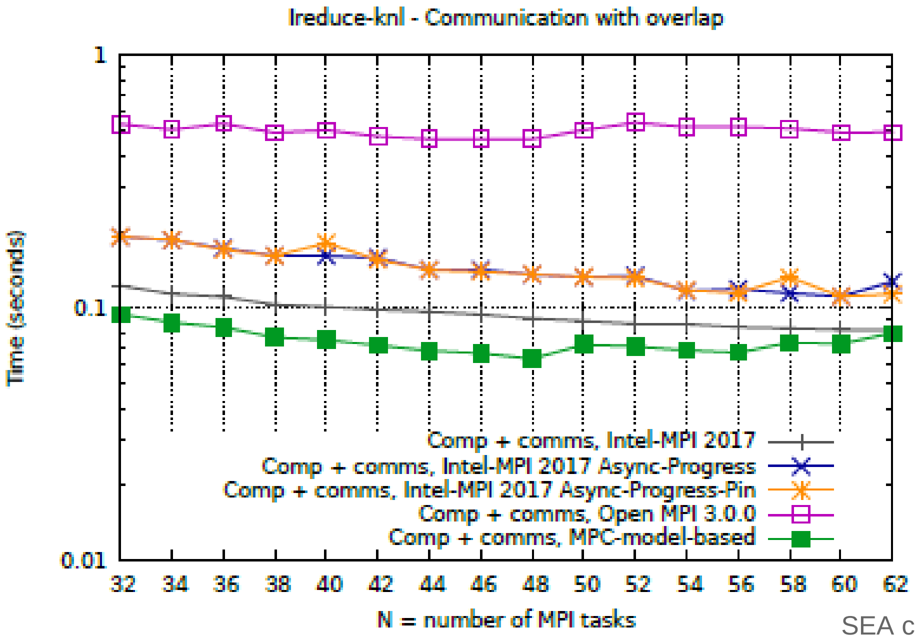






Collective offload: hardware + software support

- To realize collective offload on NIC, hardware support is needed…
- …but might be not enough to observe performance gain
  - Necessity to adapt algorithm to the workload and number of cores available to handle offloaded collectives



CONCLUSION

Conclusion

- In this presentation we have shown various points limiting asynchronous progression…

- …along with solutions by offloading services to a device (e.g., the BXI and Portals 4 through the MPC framework) to improve overlap
  - Hardware matching limiting the necessity of polling on CPUs
  - Collectives offloading to limit CPU time necessary to unfold the algorithm

- These evolutions in network cards which are bound to provide matching in a more generalized manner should open a lot of opportunities to HPC hardware.
  - But, as always, may also bring new issues to handle properly to obtain best performances.

Commissariat à l’énergie atomique et aux énergies alternatives
T. +33 (0)1 69 26 40 00
Etablissement public à caractère industriel et commercial | RCS Paris B 775 685 019