DEEP Projects

ParaStation MPI

Carsten Clauss, ParTec AG

ParTec Modular Supercomputing

The Research leading to these results has received funding from the European Commission's FP7, H2020, and EuroHPC Programmes, under Grant Agreements n° 287530, 610476, 754304, and 955606

Outline

- ParaStation overview
- Modular MPI Jobs
  - Network Bridging
  - Workflows
  - MSA awareness
- CUDA awareness
- Persistent MPI Windows

ParaStation History

- 1995: ParaStation research project (→ University of Karlsruhe)
- 1999: ParTec was founded as a spin-off
- 2005: Open source (→ ParaStation Consortium)
- since 2004: Cooperation with JSC
  - various precursor clusters
  - DEEP-System (MSA Prototype)
  - JURECA (Cluster/Booster)
  - JUWELS (Cluster/Booster)
  - JURECA DC
- since 2010: DEEP Projects
  - Cluster/Booster → Modularity
- since 2017: ParaStation Modulo

ParaStation Modulo

- ParaStation ClusterTools
  - Tools for provisioning and management
- ParaStation HealthChecker & TicketSuite
  - Automated error detection & error handling
  - Ensuring integrity of the computing environment
  - Keeping track of issues
  - Powerful analysis tools
- ParaStation MPI & Process Management
  - Runtime environment specifically tuned to the largest distributed memory supercomputers

ParaStation Process Manager

- Scalable network of MPI process management daemons running on the computational nodes:
  - Process startup and control, I/O forwarding, …
  - Precise resource monitoring
  - Proper cleanup after jobs
- PSSLURM and PSMOM:
  - Plugins to the ParaStation Management daemons
  - For tight integration with Slurm & Torque
  - Reduce number of daemons

ParaStation MPI Library

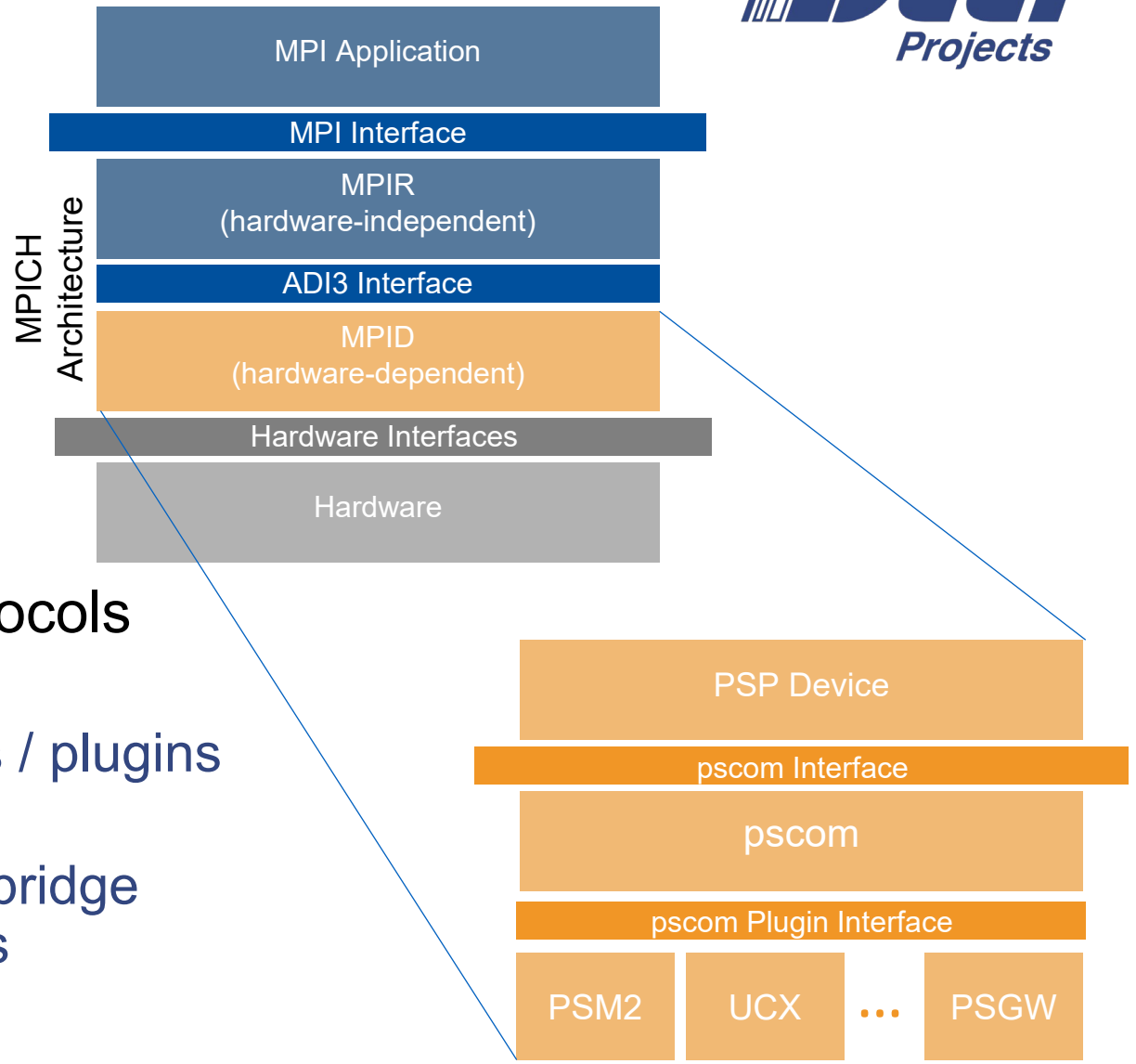
- Based on MPICH 3.3.2 (merge with 3.4.1 coming soon)
  - Maintains MPICH ABI compatibility
  - Supports all MPICH tools (tracing, debugging, …)
- MPI libraries for several compilers (especially for GCC and Intel)
- Supports a wide range of interconnect technologies, even in parallel:
  - InfiniBand on JURECA Cluster and JUWELS
  - Omni-Path on JURECA Booster
  - Extoll on DEEP projects research systems
  - BXI planned to be integrated in RED-SEA

ParaStation MPI Library

- Proven to scale up to 3,500 nodes and 140,000 procs per job
- HPL runs with ParaStation MPI:
  - JURECA & Booster: No. 29 (Top500 Nov 2017)
  - JUWELS: No. 23 (Top500 Jun 2018)
  - JUWELS Booster: No. 7 (Top500 Nov 2020)

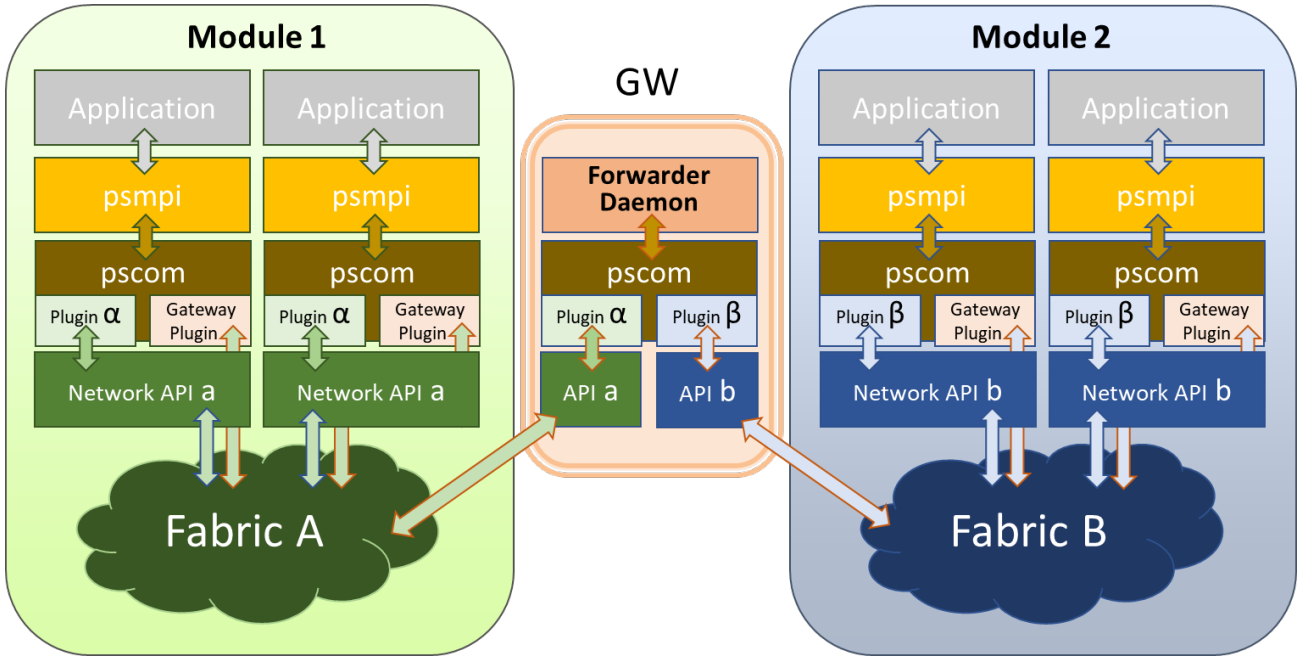
Software Architecture

- Upper (HW-independent) layers are derived from MPICH
- MPICH layers beneath ADI3 are replaced by:
  - ParaStation PSP Device, plus
  - pscom low-level communication library
- Support for various transports and protocols via pscom plugins
  - Applications may use multiple transports / plugins at the same time
  - Gateway capability via PSGW plugin to bridge transparently between different networks



Network Bridging

- Two processes communicate through a gateway if they are not directly connected by a high-speed network (e.g., IB, OPA, Extoll…)
- High-speed connections between processes and gateway daemons
- Static routing to choose a common gateway
- Virtual connection between both processes through the gateway, transparent for the application
- Virtual connections are multiplexed through gateway connections



Modular MPI Jobs

- Example for a job on 2 modules of the DEEP-EST prototype:
  - Use of srun with colon notation

```
srun … : …
```

  - Modules: Cluster (CN) + Extreme Scale Booster (ESB)

```
--partition dp-cn … : … --partition dp-esb
```

  - 8 Nodes / 64 Procs on Cluster and 16 Nodes / 256 Procs on Booster

```
-N8 -n64 … : … -N16 -n256
```

  - Use of 1 gateway node in between

```
srun --gw_num=1 …
```

```
srun --gw_num=1 --partition dp-cn -N8 –n64 ./hello_mpi_world :
--partition dp-esb –N16 –n256 ./hello_mpi_world
```

Workflows

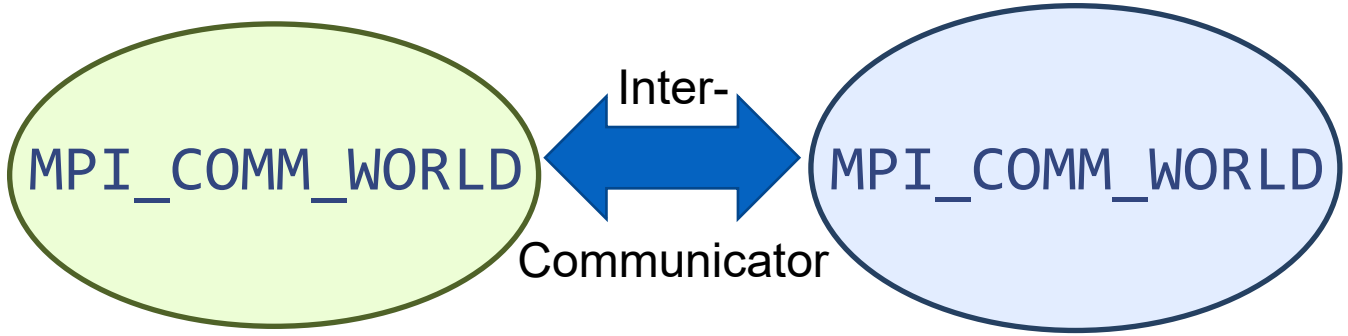
- An MPI job started with colon notation via srun will run in a *single* `MPI_COMM_WORLD`.
- Workflows may demand for multiple `MPI_COMM_WOLRDs` that may connect (and later disconnect) with each other during runtime.
- Simple job script example for such a case:

```
#!/bin/bash
#SBATCH --gw_num=1
#SBATCH --nodes=8 --partition=dp-cn
#SBATCH hetjob
#SBATCH --nodes=16 --partition=dp-esb

srun –n64  --het-group 0 ./mpi_hello_accept &
srun –n256 --het-group 1 ./mpi_hello_connect &

wait
```

Starts two *separate* `MPI_COMM_WORLDs`

Establishing Communication

- According to the MPI standard, the following functions can be used to establish connections between two separate `MPI_COMM_WORLD`s:
  - `MPI_Open_port()`
  - `MPI_Comm_accept()`
  - `MPI_Comm_connect()`
  - `MPI_Comm_disconnect()`

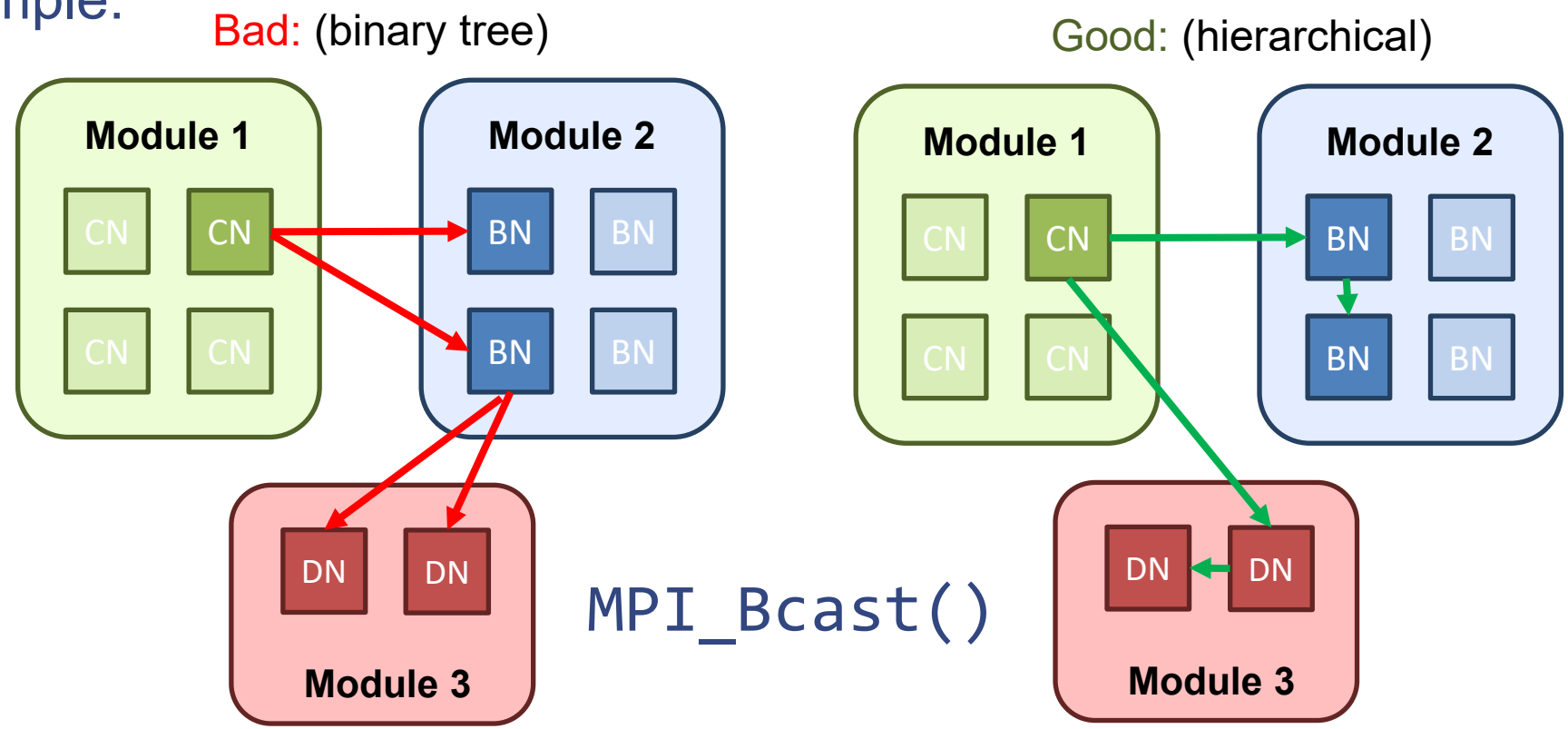


- ParaStation MPI supports all these functions – even for connections across module boundaries.
- …`MPI_Comm_spawn()` is supported, but currently not quite well for the inter-module case.

→ Is there a demand for this on application side?

MSA Awareness

- Modularity-aware MPI Collectives:
  - Optimized patterns for collectives that take the modularity into account
  - Assumption: Inter-module communication is the bottleneck
  - Example:



Hierarchical Collectives

- General rules used here to optimize collectives:
  1. First do all module-internal gathering and/or reduction operations — if required.
  2. Then perform the inter-module operation with only one process per module.
  3. Finally, distribute the data within each module in a strictly module-local manner.

- Multi-level hierarchy awareness:

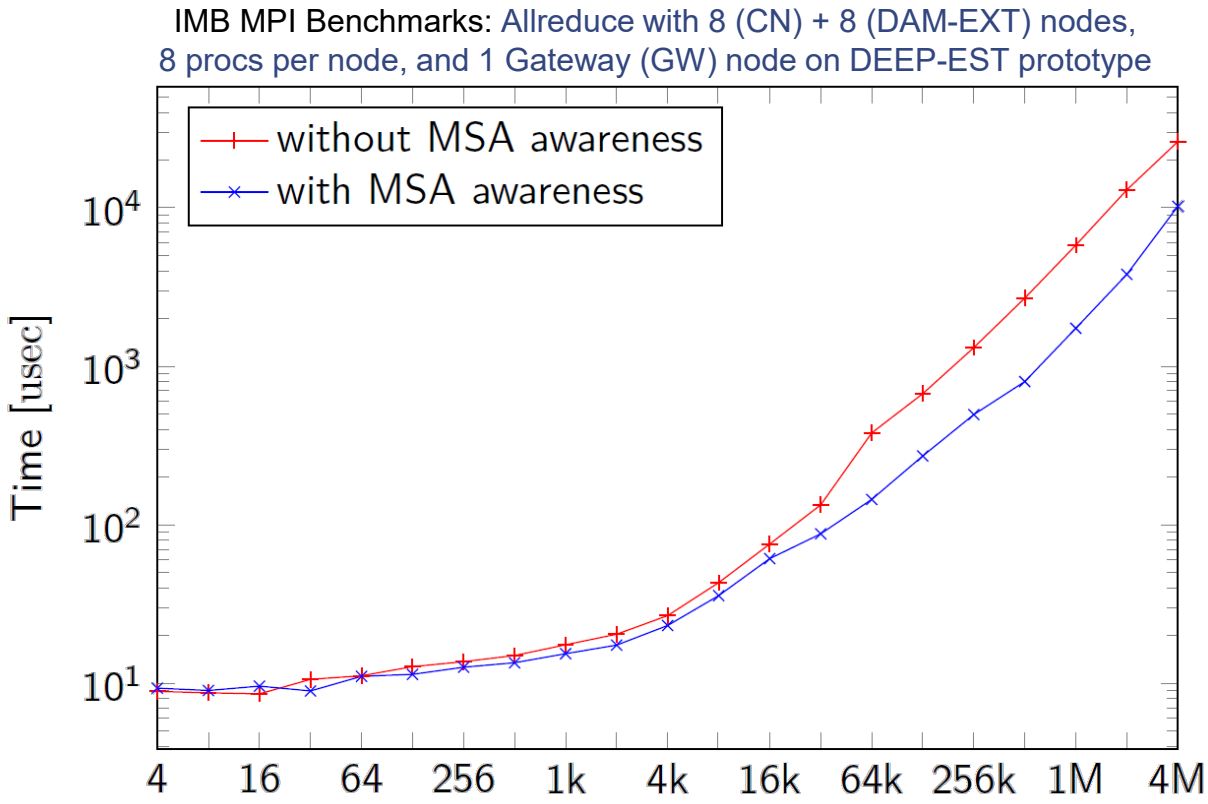
  → Apply this set of rules *recursively:* First on module level, then on node level…

- Usage: Set environment variables…
  - `PSP_MSA_AWARENESS=1`
  - `PSP_MSA_AWARE_COLLOPS=1`
  - `PSP_SMP_AWARE_COLLOPS=1`

As these features are not always beneficial and/or are still experimental, they are disabled by default!

Performance Improvement

- Improvement heavily depends on the setting, for example:
  - number of processes / gateway nodes involved
  - rank distribution in communicator
  - message sizes of the pattern
  - …and the pattern itself
- Currently supported patterns:
  - `MPI_Bcast / MPI_Ibcast`
  - `MPI_Reduce / MPI_Ireduce`
  - `MPI_Allreduce / MPI_Iallreduce`
  - `MPI_Scan / MPI_Iscan`
  - `MPI_Barrier`



API Extensions

- Besides this kind of transparent MSA awareness, there is also the possibility for the application to adapt to modularity explicitly.
- API additions by ParaStation MPI for retrieving topology information:
  - Querying the module ID via the `MPI_INFO_ENV` object:

```
MPI_Info_get (MPI_INFO_ENV , "msa_module_id", …, value, …);
```

  - Splitting communicators according to the topology by utilizing a newly added split type for `MPI_Comm_split_type()`:

```
MPI_Comm_split_type (oldcomm, MPIX_COMM_TYPE_MODULE, …,
                     &newcomm);
```

CUDA awareness

- In the first instance, CUDA awareness just means that an application is allowed to pass GPU-Device pointers to the MPI.
- Otherwise, if the memory is not managed by the CUDA runtime, an explicit staging is required by the application.

Back then, without CUDA awareness:

```
cudaMemcpy(temp_buffer_on host,                    // <- Staging
  buffer_in_device_memory, cudaDeviceToHost);

MPI_Send(temp_buffer_on_host, …);
```

Today, with CUDA awareness:

```
MPI_Send(buffer_in_device_memory, …);
```

CUDA and ParaStation MPI

- CUDA awareness supported by the following MPI APIs:
  - Point-to-point (e.g., `MPI_Send`, `MPI_Recv`, ...)
  - Collectives (e.g., `MPI_Allgather`, `MPI_Reduce`, ...)
  - One-sided (e.g., `MPI_Put`, `MPI_Get`, ...)
  - Atomics (e.g., `MPI_Fetch_and_op`, `MPI_Accumulate`, ...)
- CUDA awareness for all transports / pscom plugins via staging
- CUDA optimization / GPUDirect: UCX plugin (`pscom4ucp`)

CUDA and ParaStation MPI

- Ability to query CUDA awareness at compile time:

```
#if defined(MPIX_CUDA_AWARE_SUPPORT) && MPIX_CUDA_AWARE_SUPPORT
printf("The MPI library is CUDA-aware\n");
#endif
```

- …and also at runtime via API extensions:

```
if (MPIX_Query_cuda_support())
    printf("The CUDA awareness is activated\n");
```

```
MPI_Info_get(MPI_INFO_ENV, "cuda_aware", … , value, …);
```

Similar to Open MPI's CUDA extensions

- As CUDA awareness adds some cycles to latency, it is disabled by default!

→ Set `PSP_CUDA=1` to enable it.

Persistent MPI Windows

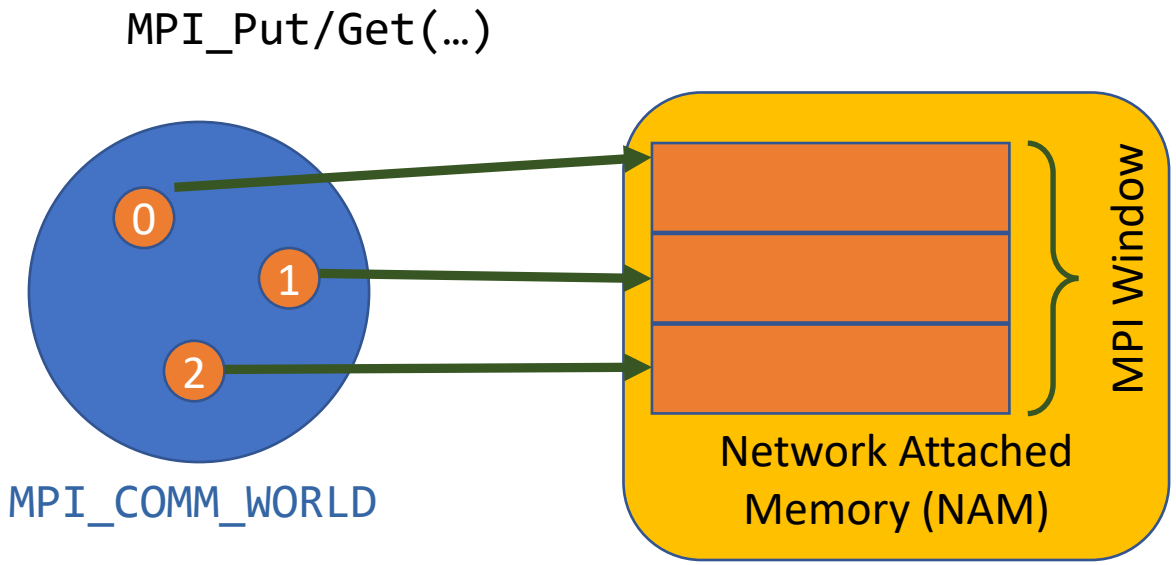
- Extension stemming from DEEP-EST: persistent MPI RMA windows
- Primarily developed for addressing so-called *Network Attached Memory*

  …but persistent RMA windows can also be built with *shared-memory* on common compute nodes as well!

- The idea of Network Attached Memory (NAM):
  - Network nodes without (significant) compute power, but equipped with a lot of fast and byte-addressable memory
  - Plus an interconnect technology that allows direct RDMA Put/Get operations onto this memory from remote compute nodes
- How to integrate this into the world of MPI and its RMA interface?

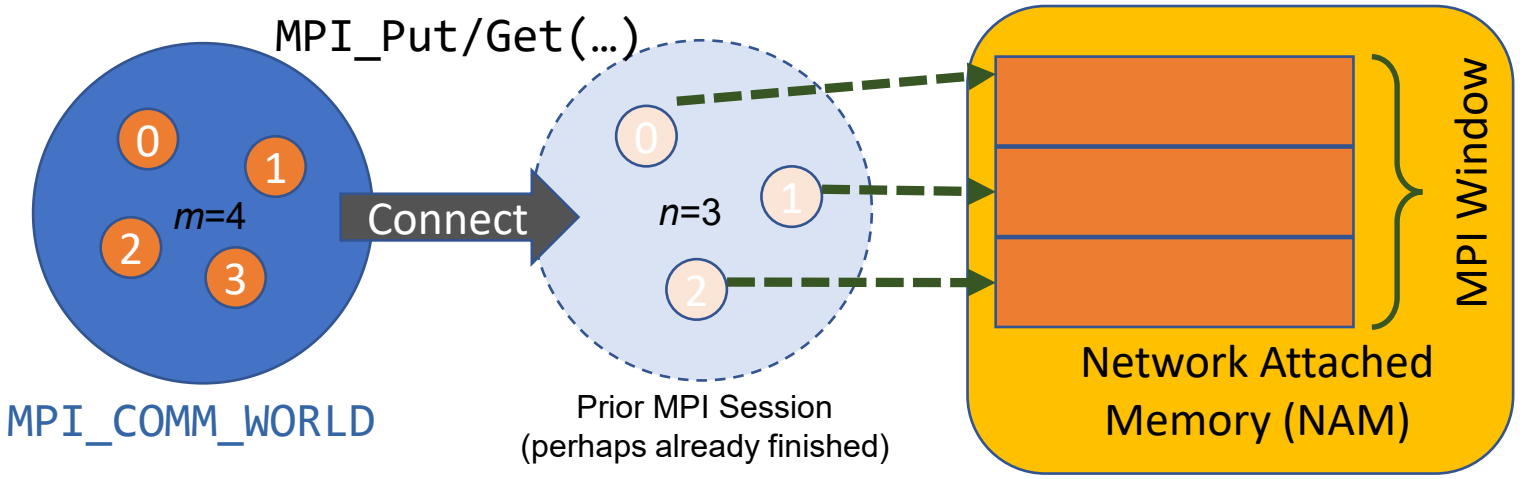
Persistent MPI Windows

- Idea followed in DEEP-EST:
  - Use `MPI_Win_allocate()` with special MPI Info key/value pairs to allocate
  - One NAM region is then associated with each MPI rank in the window
  - Use common `MPI_Put/Get()` operations for accessing these regions
  - Persistency: freeing the MPI window does *not* release the associated memory



Persistent MPI Windows

- Scenario for application workflows:
  - Prior MPI session has called `MPI_Win_allocate()` for NAM with n ranks
  - A new MPI session with m ranks now wants to attach to that window
  - New session calls `MPI_Comm_connect()`, returning an inter-comm and uses this inter-comm for creating an RMA window object by attaching
  - Window now has as many NAM regions as ranks n in former session, and NAM regions are addressable by remote ranks in the inter-comm



Summary and Outlook

- Which of these features are of interest for DEEP-SEA Applications?
  - Support for modular MPI jobs? (i.e., jobs across multiple MSA modules)
  - Transparent features to optimize communication on MSA systems?
  - API extensions to adapt applications explicitly to modularity?
  - Support for workflows (i.e., jobs with multiple `MPI_COMM_WORLD`s) via
    - *MPI_Comm_connect/accept()?*
    - *MPI_Comm_spawn()?*
    - *Persistent RMA windows?*
  - Awareness/interoperability for CUDA and/or other programming models?
- Any further demands, ideas, or special wishes towards MPI support?

Resources and Contact

- ParaStation MPI description on the DEEP Projects webpage:
  https://www.deep-projects.eu/software/programming-environment/parastation-mpi.html
- Documents on the BSCW: DEEP-SEA → Seminar → ParaStation MPI
  https://bscw.zam.kfa-juelich.de/bscw/bscw.cgi/3597441
- ParaStation MPI as open-source on GitHub:
  https://github.com/ParaStation/psmpi
- For further questions and/or discussions just contact me directly:
  clauss@par-tec.com

DEEP-SEA

Software for Exascale Architectures