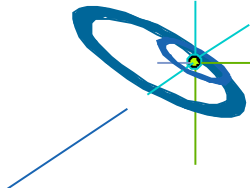




Parallel Jobs and MPI

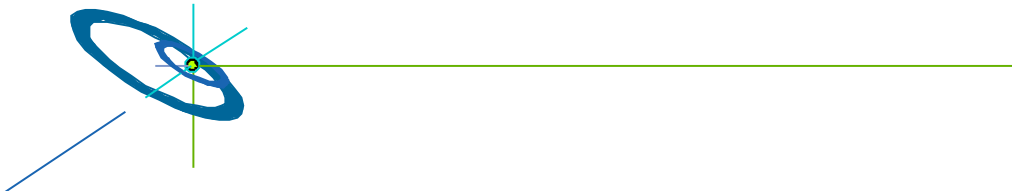
Sven Stork (HLRS)
stork@hirs.de

Int.EU.Grid Kick off Meeting



Content

- Overview of Open MPI
- Overview of PACX MPI
- Overview of Meta Open MPI
- Conclusion



Open MPI

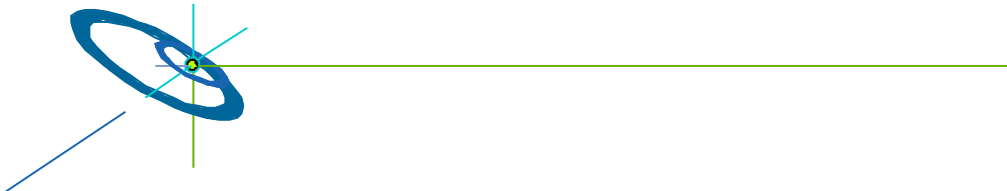
Open MPI

H L R I S



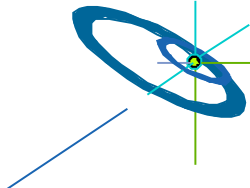
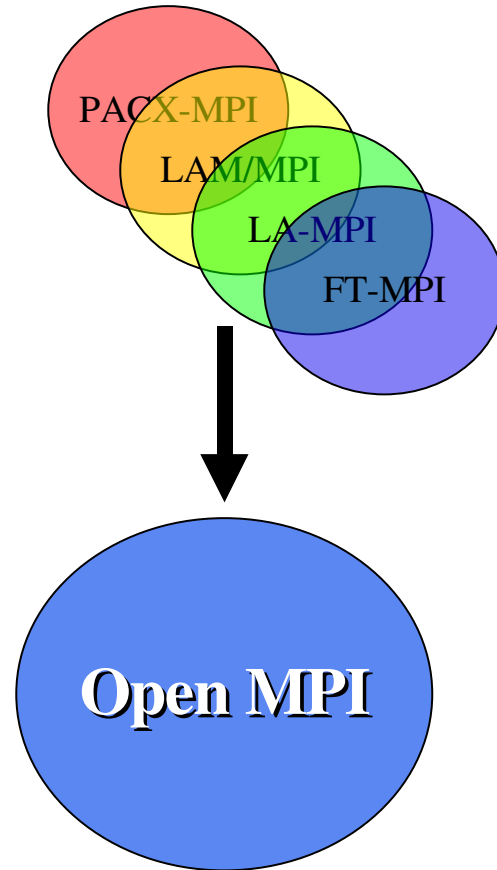
Open MPI – History

- in 2003 the developers of FT-MPI, LA-MPI, LAM/MPI decided to focus their experience and efforts on one MPI implementation instead on several
- in 2004 the real designing and coding started
- HLRS was joining Open MPI
- First release on Super Computing 2005



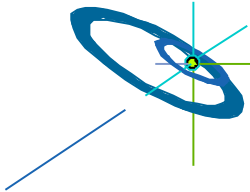
Open MPI - From Scratch

- Merge of **ideas** from
 - FT-MPI (U. of Tennessee)
 - LA-MPI (Los Alamos)
 - LAM/MPI (Indiana U.)
 - PACX-MPI (HLRS, U. Stuttgart)



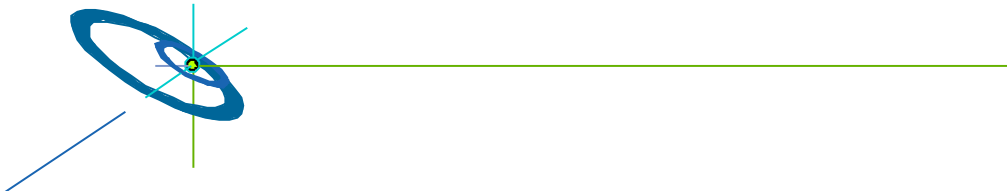
Open MPI – Who is Open MPI ?

- Founders
 - High Performance Computing Center, Stuttgart
 - Indiana University
 - Los Alamos National Laboratory
 - The University of Tennessee
- New members
 - Cisco Systems
 - Myricom
 - Sun Microsystems
 - Sandia National Labs
 - University of Huston
 - Mellanox Technologies
 - Voltaire
 - to be continued ...



Open MPI – Project Goals

- State of the art MPI implementation
 - Full support of the MPI-2 standard
 - Full thread support
 - Avoidance of old legacy code
 - Profit from long experience in MPI implementations
 - Avoiding the “forking” problem
 - Community / 3rd party involvement
 - Production-quality research platform
 - Rapid deployment for new platforms

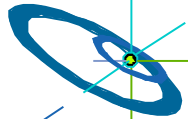
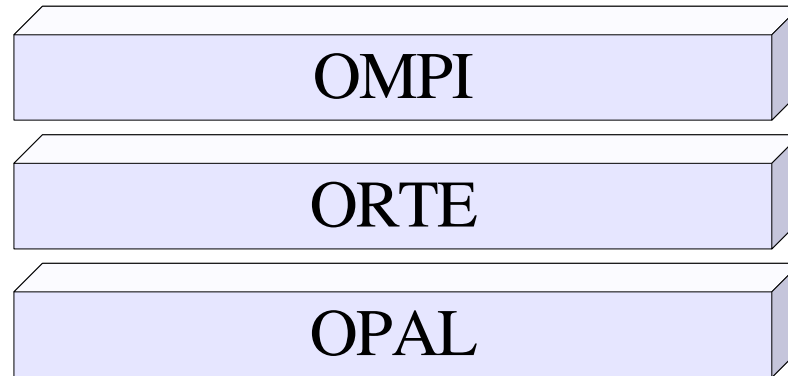


Open MPI – Design Goals

- Component architecture
- Message fragmentation / reassembly
- Design for heterogeneous environments
 - Multiple networks (run-time selection and striping)
 - Node architecture (data type representation)
- Support for automatic error detection / retransmission
- Should be portable and performant
 - Small cluster
 - “Big iron” hardware
 - “Grid” (everyone has a different definition)
 - ...

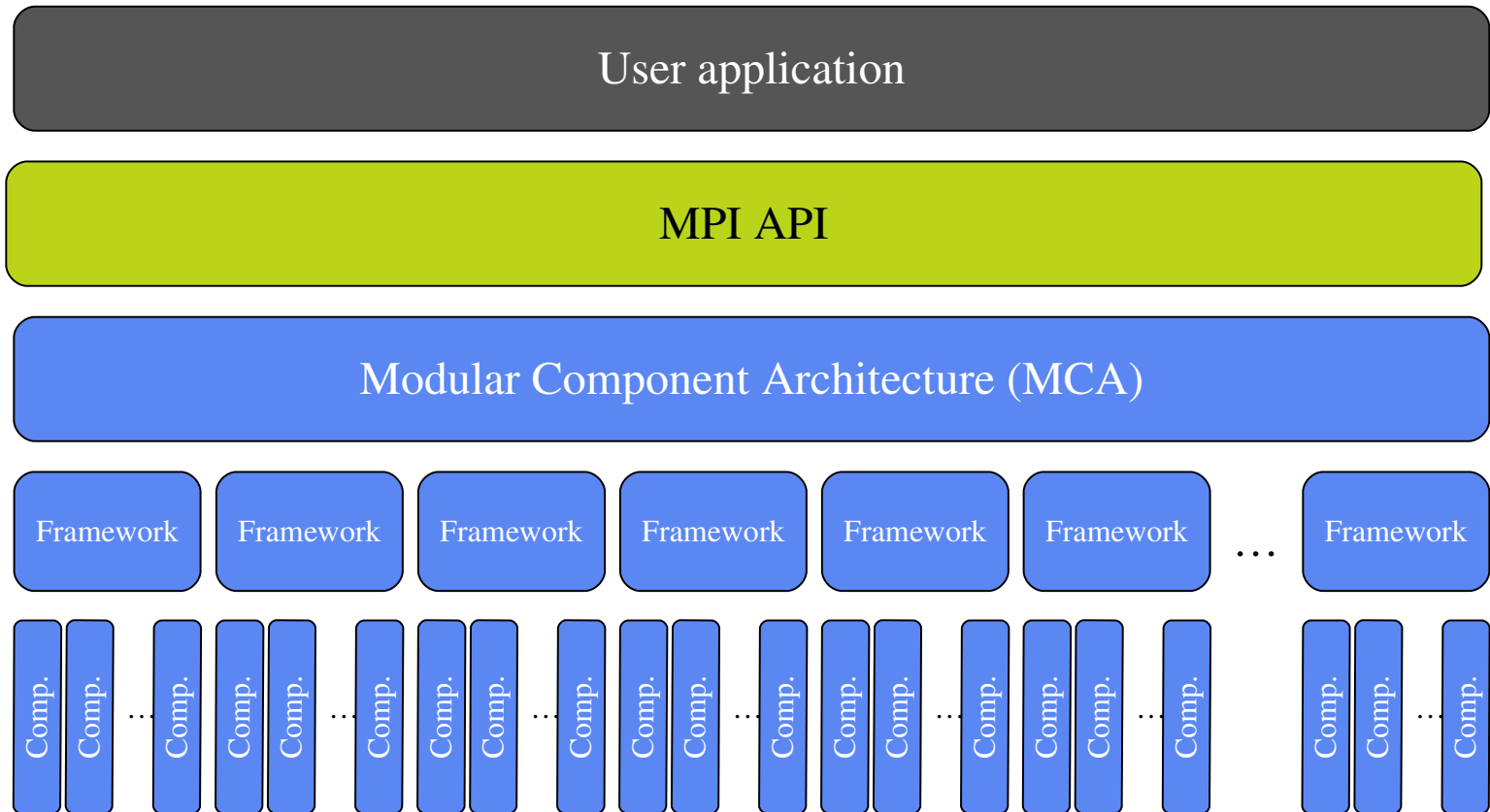
Open MPI – Design

- Open MPI consist of 3 different layers
 - OMPI : Open MPI
 - ORTE : Open Runtime Environment
 - OPAL : Open Hardware Abstraction layer



Open MPI - MCA

- MCA top level view



Open MPI – Selected Frameworks

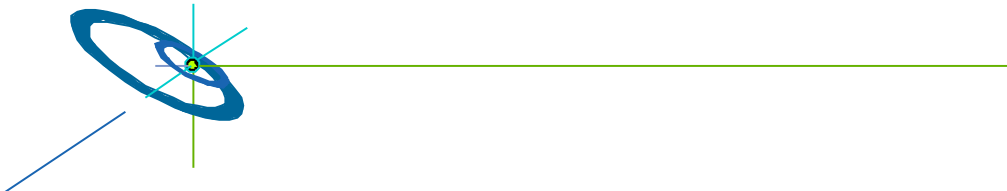
- OPAL
 - maffinity - Memory Affinity
 - paffinity - Processor Affinity
- ORTE
 - GPR - General Purpose Registry
 - IOF - I/O Forwarding
 - PLS - Process Lunch System
 - rsh
 - xcpu
 - xgrid
 - bproc
 - fork
 - poe
 - slurm
 - tm
 - RDS - Resource Discovery System

Open MPI – Selected Frameworks

- OMPI
 - BTL - Byte Transfer Layer
 - **tcp** - **TCP/IP**
 - **openib** - **Infiniband OpenIB Stack**
 - **gm/mx** - **Myrinet GM/MX**
 - **mvapi** - **Infiniband Mellanox Verbs**
 - **sm** - **Shared Memory**
 - PML - Point-2-Point Management Layer
 - **OB1** - **stripping across several transport layers**
 - **DR** - **Data Reliability**
 - COLL - Collectives
 - **basic** - **non optimized algorithms**
 - **tunes** - **optimized algorithms**
 - **sm** - **shared memory collectives**
 - **hierach** - **hierarchical**
 - OSC - One Sided Communication
 - **pt2pt** - **Emulation via point 2 point**
 - TOPO - Topology

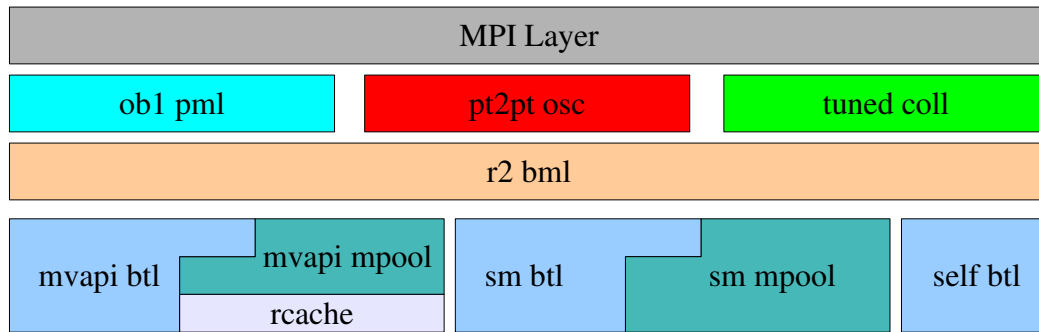
Open MPI – Modular Concept

- Each MCA framework will load all available components
- Each MCA Framework will evaluate/query each component regarding its capability to run in the current environment
- Each MCA framework will initialize the (best) runnable components and unload the others
- Composing the best fitting MPI implementation for the current system 'on the fly'



Open MPI – MCA Example

- Example : OMPI Layer

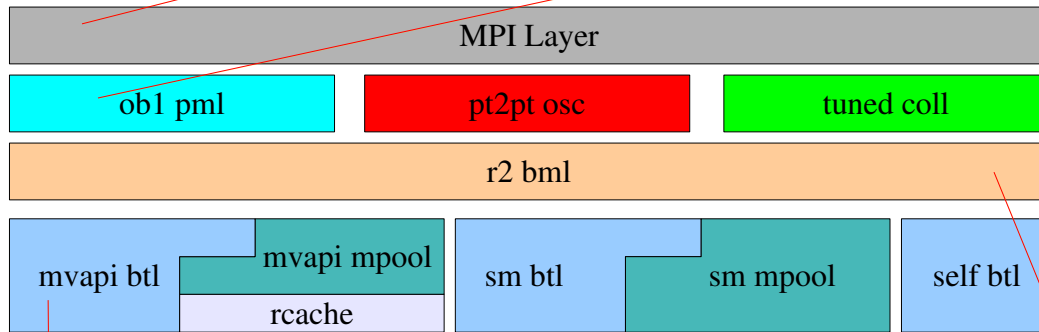


Open MPI - MCA Example

- Example : OMPI Layer

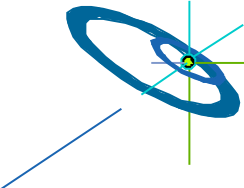
•check parameter
•calls the responsible component

•select protocol to use
•management fragmentation



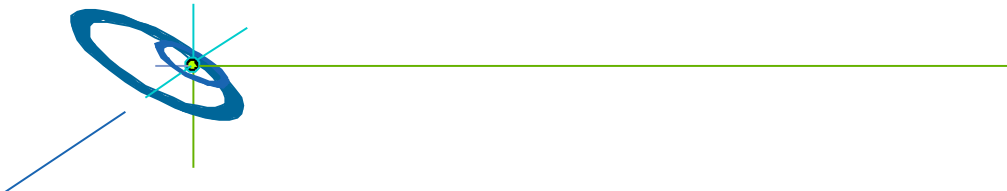
•transfers the data via Infiniband

•decide which BTL to use for the transport



Open MPI – ompi_info

- the **ompi_info** utility display several information about the installed Open MPI implementation
 - Open MPI version
 - build information
 - compiler settings
 - compile time configuration
 - installed components
 - component parameters



Open MPI – ompi_info parameter example

- Example : TCP BTL parameters

> *ompi_info --param btl tcp*

MCA btl: parameter "btl_base_debug" (current value: "0")

If btl_base_debug is 1 standard debug is output, if > 1 verbose debug is output

MCA btl: parameter "btl" (current value: <none>)

Default selection set of components for the btl framework (<none> means "use all components that can be found")

select which network interfaces to use

MCA btl: parameter "btl_base_verbose" (current value: "0")

Verbosity level for the btl framework (0 = no verbosity)

MCA btl: parameter "btl_tcp_if_include" (current value: <none>)

MCA btl: parameter "btl_tcp_if_exclude" (current value: "lo")

select which network interfaces NOT to use

MCA btl: parameter "btl_tcp_free_list_num" (current value: "8")

MCA btl: parameter "btl_tcp_free_list_max" (current value: "-1")

set send buffer size of the socket

MCA btl: parameter "btl_tcp_free_list_inc" (current value: "32")

MCA btl: parameter "btl_tcp_sndbuf" (current value: "131072")

MCA btl: parameter "btl_tcp_rcvbuf" (current value: "131072")

set recv buffer size of the socket

MCA btl: parameter "btl_tcp_endpoint_cache" (current value: "30720")

MCA btl: parameter "btl_tcp_exclusivity" (current value: "0")

MCA btl: parameter "btl_tcp_eager_limit" (current value: "65536")

MCA btl: parameter "btl_tcp_min_send_size" (current value: "65536")

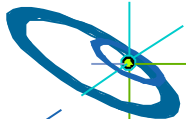
MCA btl: parameter "btl_tcp_max_send_size" (current value: "131072")

MCA btl: parameter "btl_tcp_min_rdma_size" (current value: "131072")

MCA btl: parameter "btl_tcp_max_rdma_size" (current value: "2147483647")

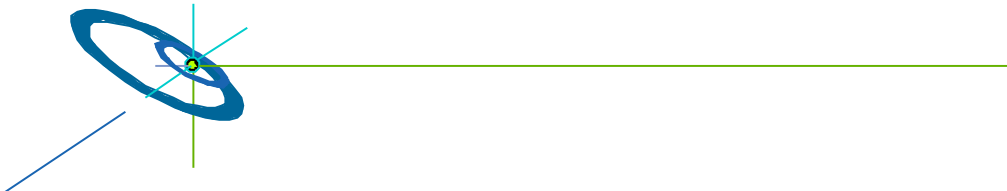
MCA btl: parameter "btl_tcp_flags" (current value: "10")

MCA btl: parameter "btl_tcp_priority" (current value: "0")



Open MPI – Exampels Parameters

- selecting which low level transport to use/ not to use
 - `mpiexec -n 4 -mca btl mvapi,sm,self Killer_App`
 - `mpiexec -n 4 -mca btl ^tcp Killer_App`
- manipulating the startup mechanism
 - `mpiexec -n 4 -mca pls_rsh_agent /usr/bin/ssh.orig Killer_App`
- performance tuning
 - `mpiexec -n 4 -mca mpi_leave_pinned 1 Killer_App`
 - `mpiexec -n 4 -mca btl_tcp_rcvbuf 514288 Killer_App`



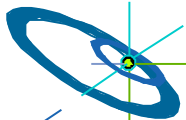
Open MPI - Installation

- Open MPI uses autotools:
 - ./configure
 - make
 - make install
- Features:
 - building as shared library and/or static library
 - enable/disable thread support
 - a lot more ...
- binary files and libraries must be found by the system
 - `$(PREFIX)/bin -> $PATH`
 - `$(PREFIX)/lib -> $LD_LIBRARY_PATH`



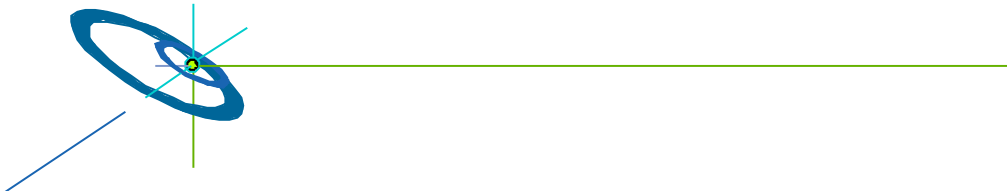
Open MPI – startup

- Open MPI supports both **mpirun** and **mpiexec**
- ORTE supports several startup mechanisms natively
 - rsh/ssh based
 - PBS/Torque
 - SLURM
 - Xgrid
- The startup mechanism can be forced/selected via an MCA parameters



Open MPI – Requirements

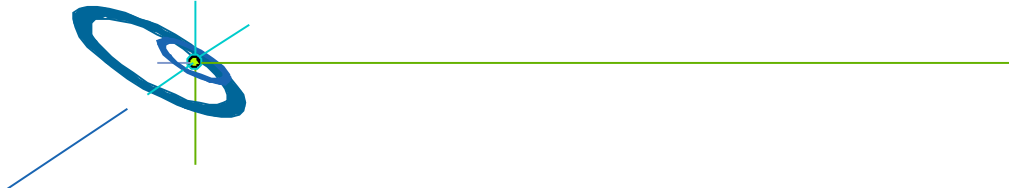
- The orte daemons need an open TCP/IP Port for incoming connections
- Different requirements for the different PLS
 - ssh requires login without password (e.g. public keys)
 - Pbroc requires default requirements
- software installation
 - Open MPI needs to be installed on the HN and CN



PACX MPI

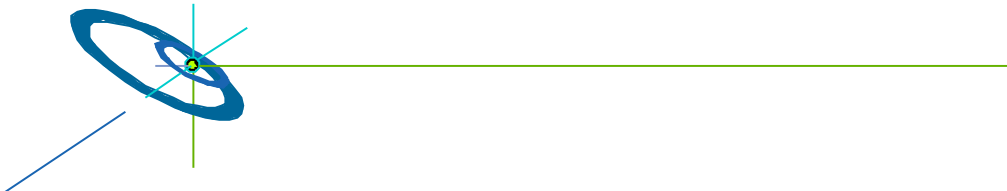
PACX MPI

H L R I S



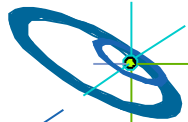
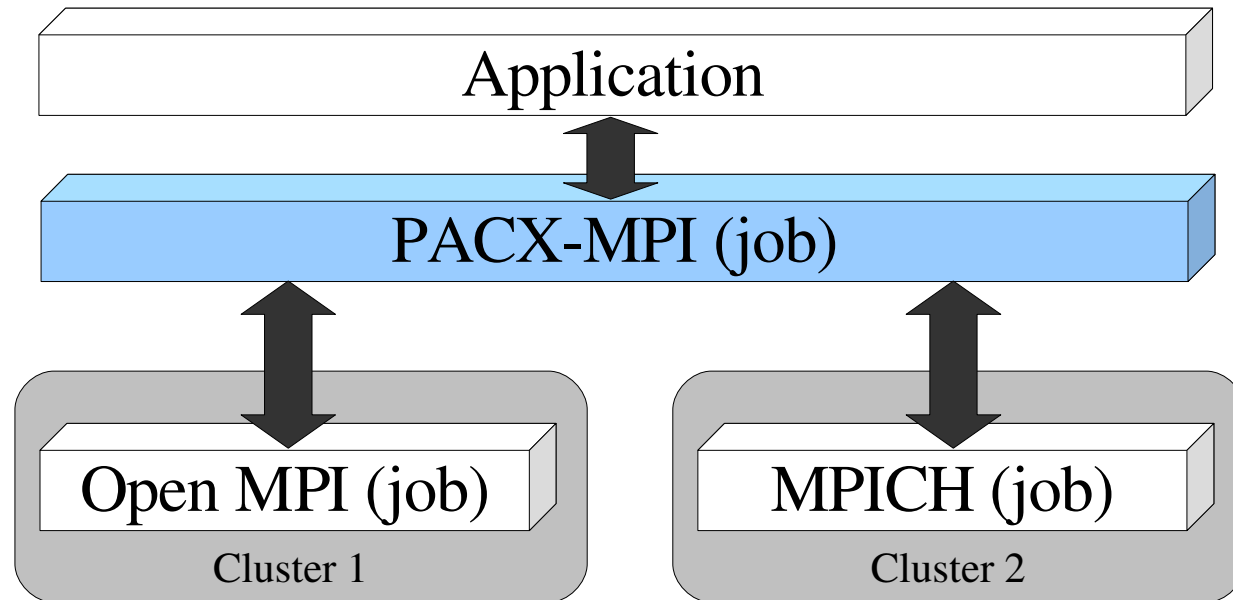
PACX MPI – Overview

- A middleware for seamlessly run a MPI-application on a network of parallel computers
- originally dev. in 1995 to connect Vector+MPP
- PACX-MPI is an optimized standard-conforming MPI-implementation, application just needs to be **recompiled(!)**
- PACX-MPI uses locally installed, optimized vendor implementations for cluster inter communication



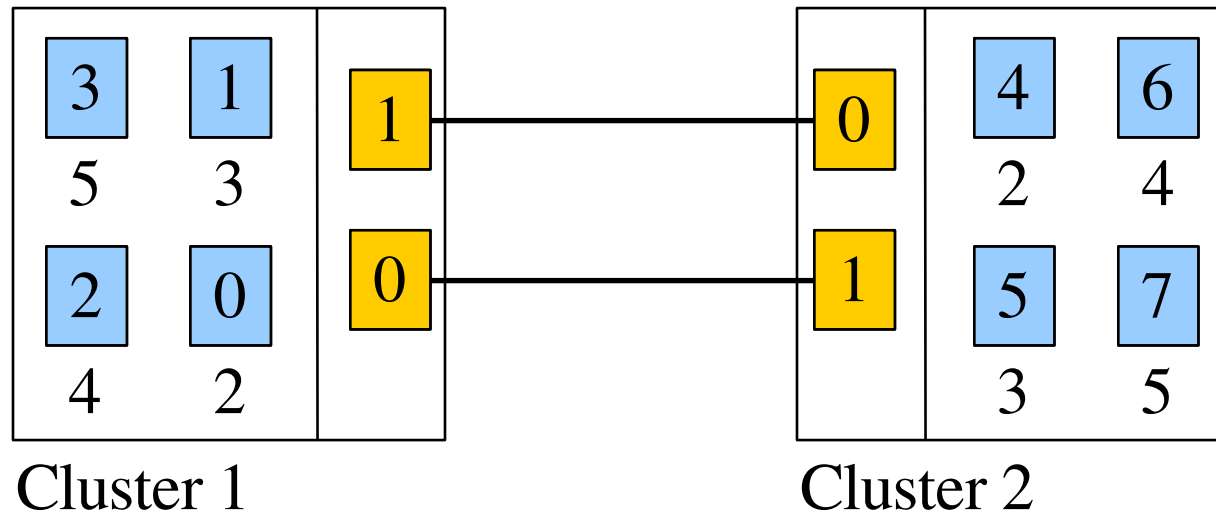
PACX MPI – Design

- PACX-MPI start a MPI job in each cluster
- PACX-MPI “merge/manage” these MPI jobs internally and emulate transparently a bigger MPI job to the application



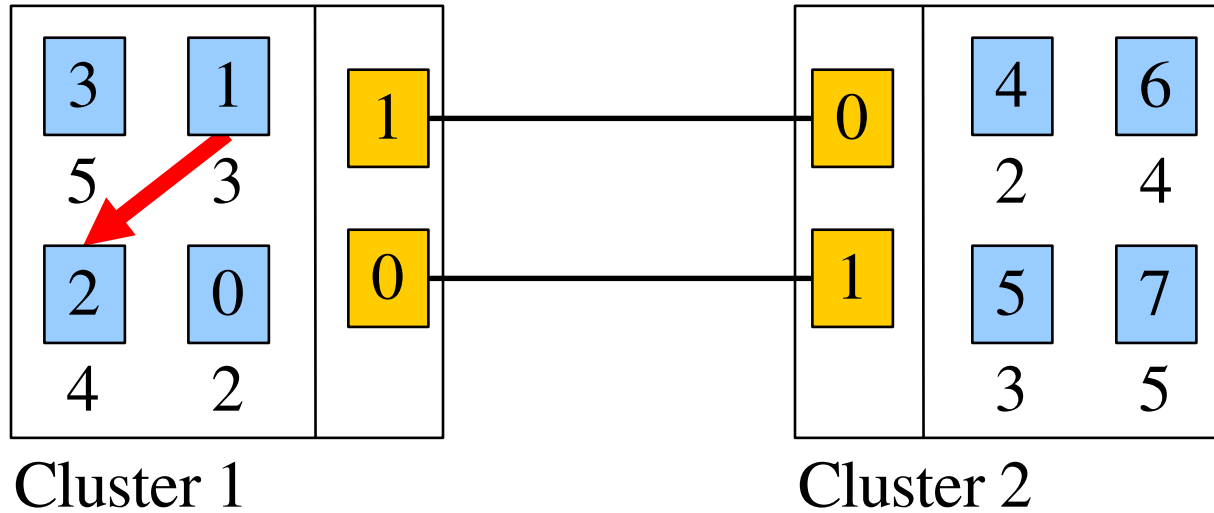
PACX MPI – Design

- PACX-MPI maps the MPI process ranks of the big job to the processes of the clusters
- PACX-MPI start 2 additional,hidden MPI processes on the local MPI jobs for the external communication
- rank 0 of the local MPI jobs is always the **out_daemon**
- rank 1 of the local MPI jobs is always the **in_daemon**



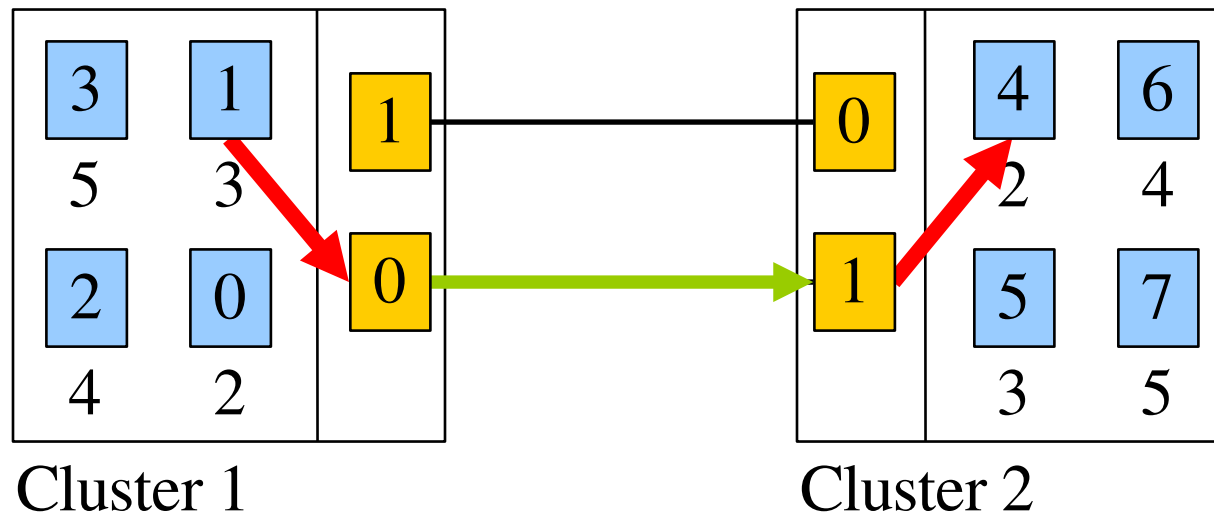
PACX MPI – Design

- Internal Communication
 - communication between processes that resides in the same cluster is performed via the local, optimized MPI



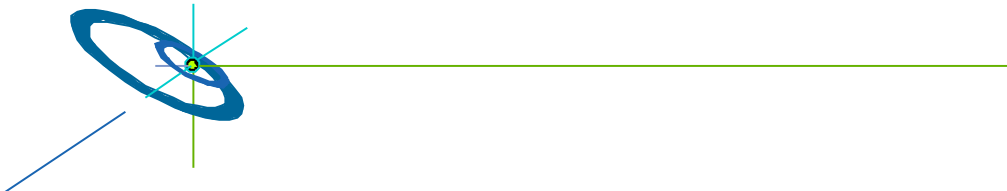
PACX MPI – Design

- External Communication
 - Send message to `out_daemon` using native MPI.
 - The **out_daemon** sends message to destination host over network using "a protocol" (TCP, ATM, SSL)
 - The **in_daemon** sends message to destination with MPI.



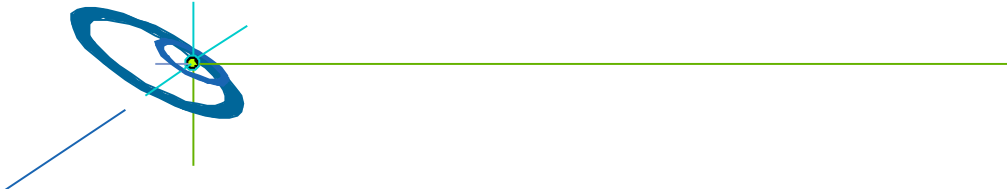
PAXC MPI - how to install

- PACX is auto tools based
 - ./configure
 - make
 - make install
- important configure switches
 - --prefix=<install target>
 - --with-mpi-dir=<path to local MPI implementation>



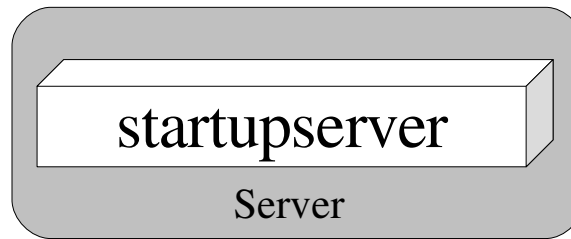
PACX MPI – usage

- PACX-MPI compiler wrappers
 - `pacxcc`
 - `pacxfc`
 - `ppacxcc`
 - `ppacxfc`
- compiling with PACX
 - `pacxcc -c hello.c`
 - `pacxcc -o hello hello.o`



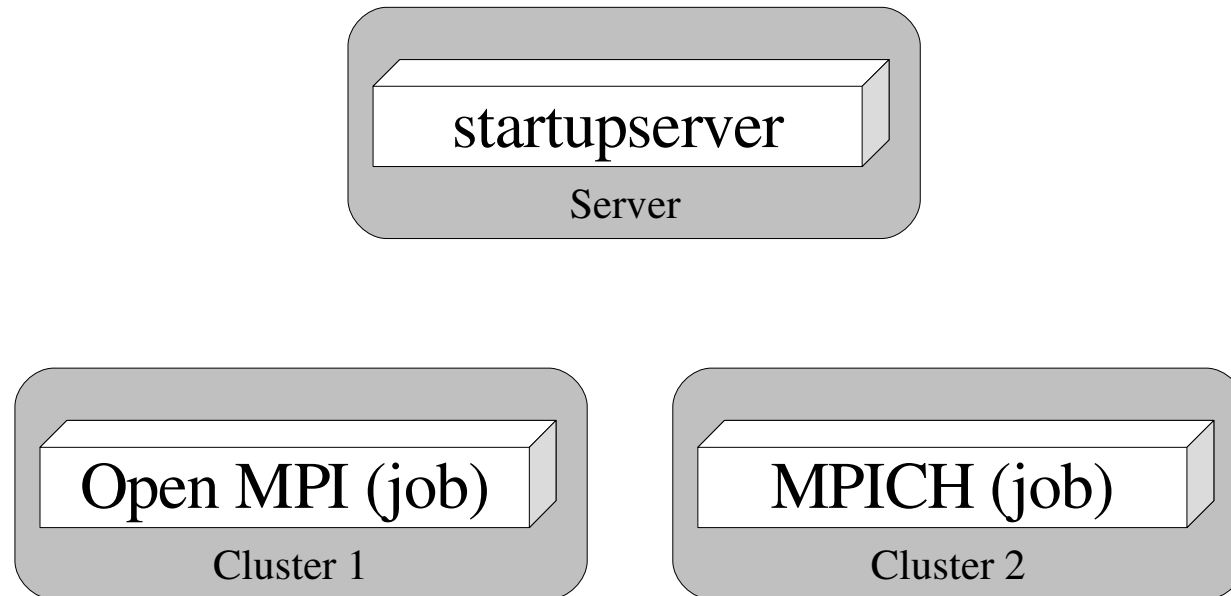
PACX MPI – usage

- Running PACX MPI applications
 - start the PACX **startupserver** with the number of clusters on a central reachable node
 - e.g. **Server : ./startupserver 2**



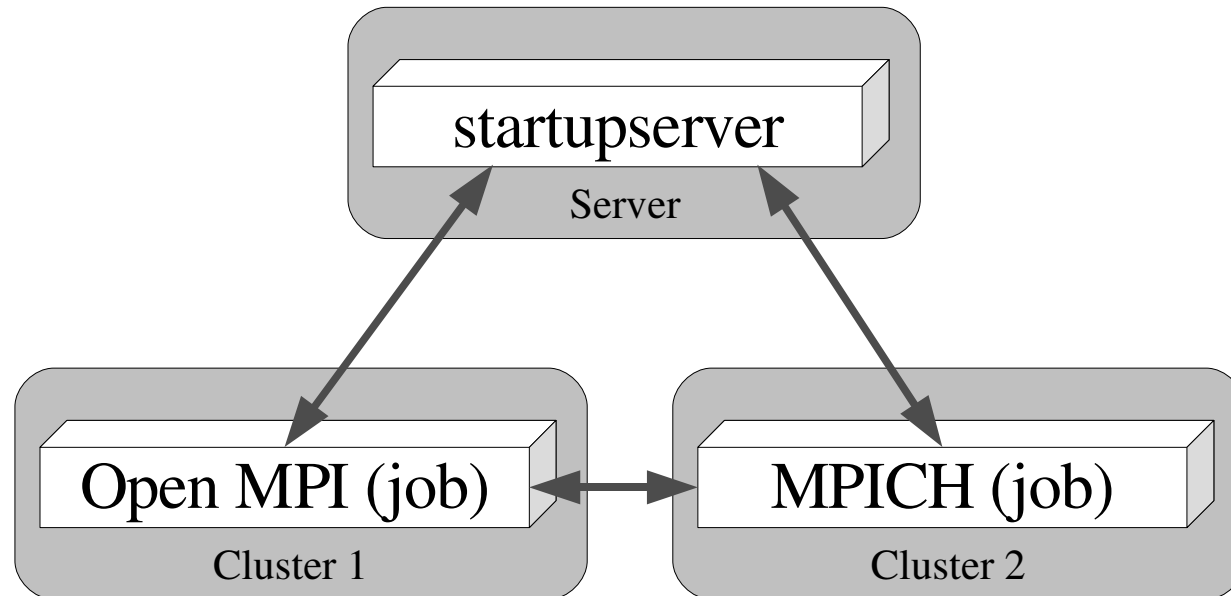
PACX MPI – usage

- Running PACX MPI applications
 - start the application on each cluster with the required number of processes (+2 additional daemon processes)
 - e.g. Cluster 1 HN : `mpirun -np 3 hello`
 - e.g. Cluster 2 HN : `mpirun -np 3 hello`



PACX MPI – usage

- Running PACX MPI applications
 - the applications connect and sync via the **startupserver** their corresponding connection information
 - the parallel programs (I/O daemons) connect to each other



PACX MPI – Requirements + Configuration

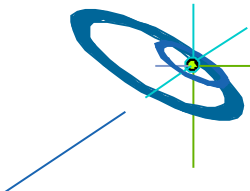
- the **startupserver** requires requires one public reachable port, the port number can be specified as command line argument
- the first (n-1) jobs need 2 public reachable ports (at least the node where the in_daemon/out_daemon is running)
- the connection ports can be configure via the “.netfile” configuration file
- the requirements of the locally installed MPI implementations
- software installation
 - HN : PACX-MPI for the local MPI implementation (required for compilation)
 - CN : PACX-MPI is statically linked to the binary, therefore no extra software than the local MPI requirements need to be installed



Open MPI & MetaComputing

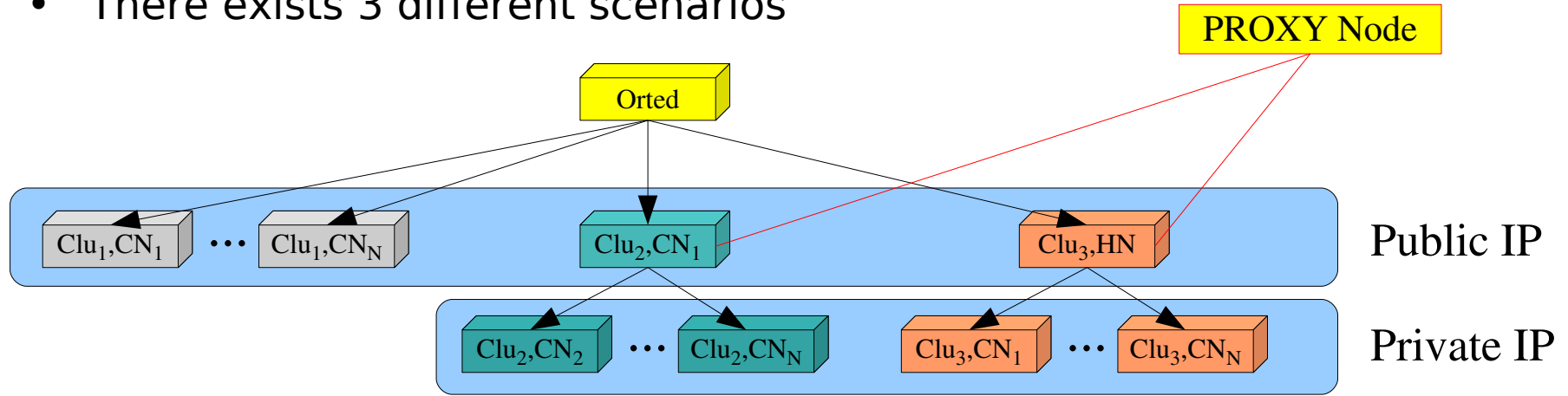
Meta enabled Open MPI - Goals

- Enable Open MPI to run applications across multiple clusters
- Only one MPI that is spawning across the whole GRID
- Usage of high speed interconnects for intra cluster communication
- best possible transport for inter cluster communication
- Handling of heterogeneity issues
 - architectures
 - endianness
- Encryption of inter cluster communication
- Compression of inter cluster communication



Open MPI & MetaComputing - Analysis

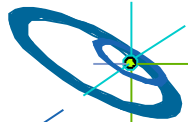
- There exists 3 different scenarios



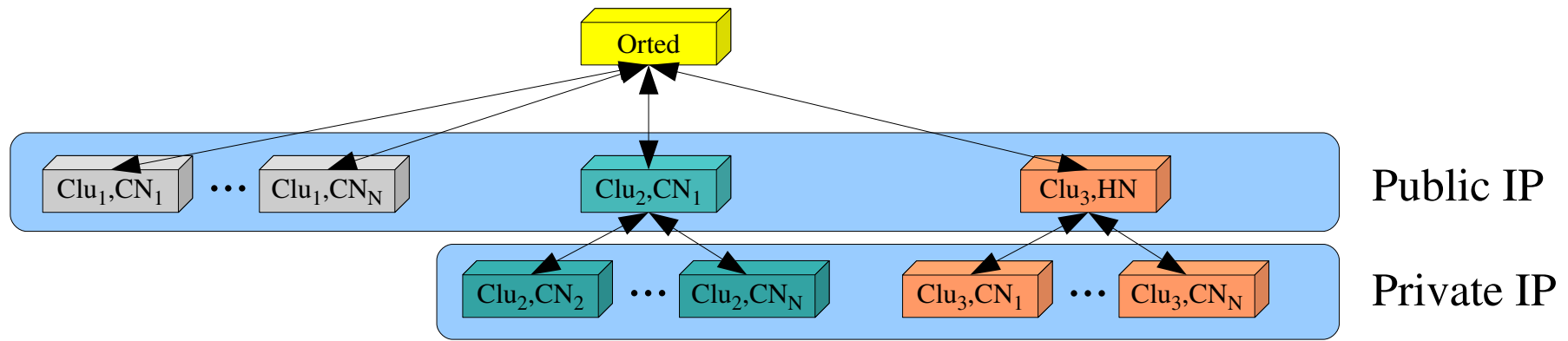
- every computation node has a public IP address

- only one of the CN has a public IP address
- all other CN are not directly reachable

- one of the HN has a public IP address
- non of the CN is directly reachable



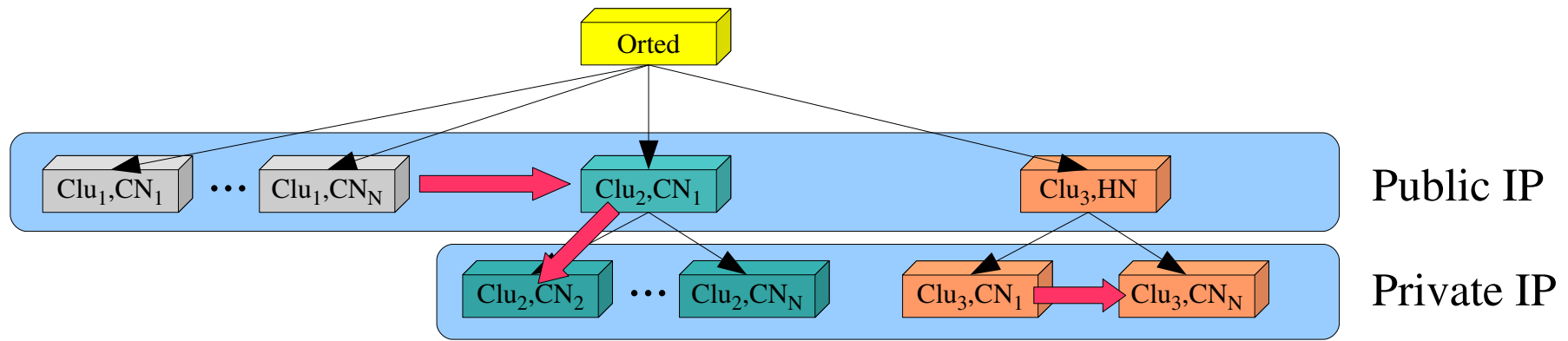
Open MPI & MetaComputing – Configuration



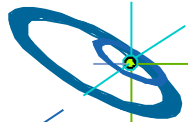
- The ORTE daemon starts the processes
- After the processes have been started each process/proxy send connection information back to the ORTE daemon
- The collected information is distributed back to the processes/proxy's



Open MPI & MetaComputing – Data Transfers

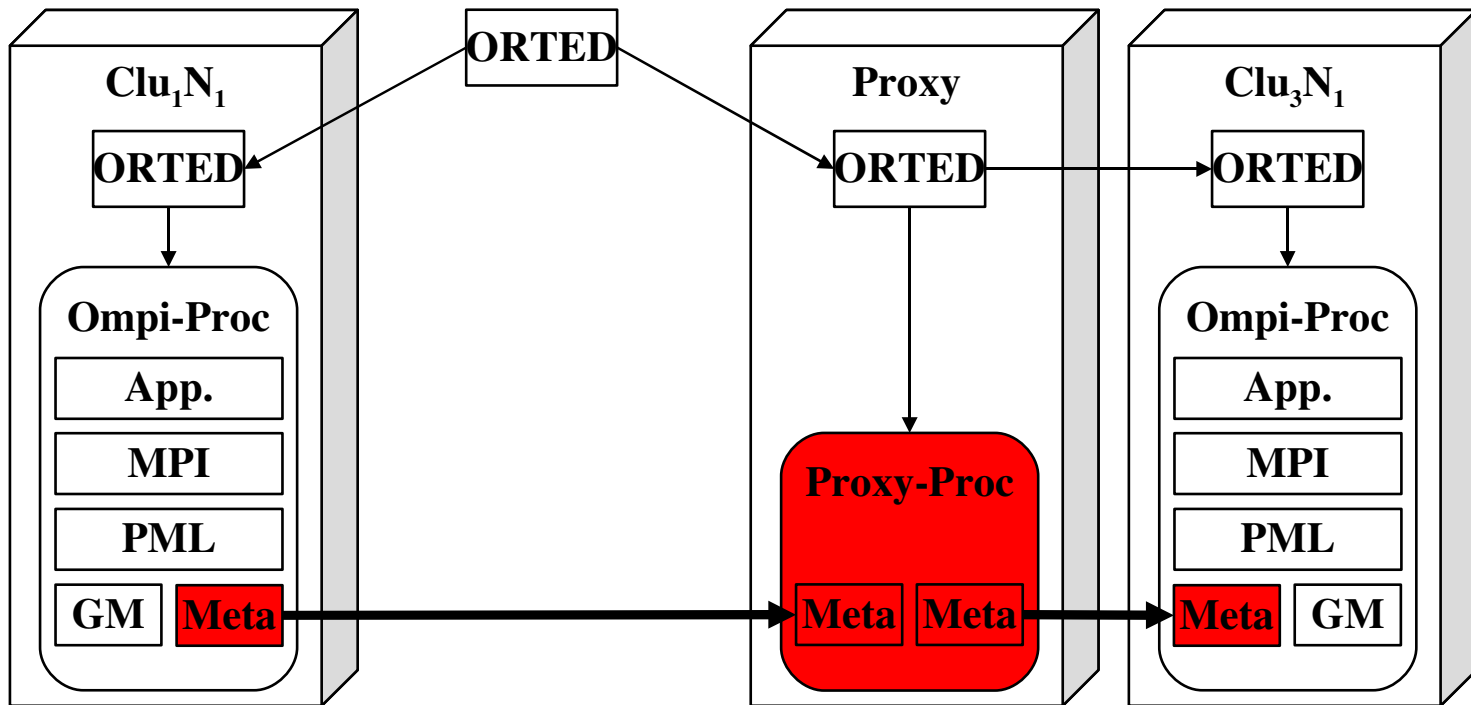


- The cluster internal communication is handled by the fast cluster interconnect BTL
- For the cluster external communication the data needs to be send to the nearest PROXY



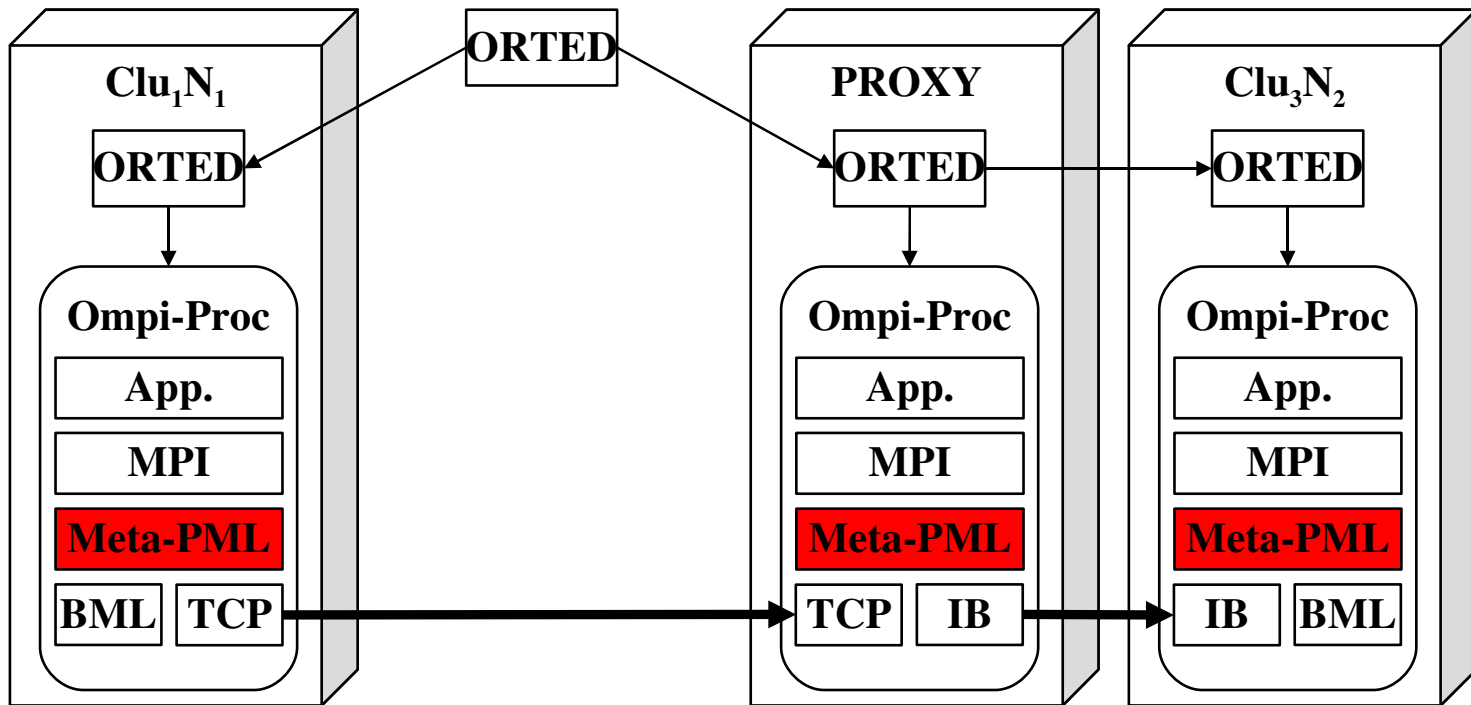
Open MPI & MetaComputing – Proposed Solutions

- META BTL
 - META BTL transfers data to processes that do not reside in the same cluster



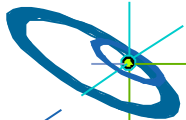
Open MPI & MetaComputing – Proposed Solutions

- META PML
 - start addition MPI processes on the PROXY nodes
 - add new META PML that manage the data transfer with existing BTLs



Open MPI & MetaComputing – Proposed Solutions

- META BTL
 - Pro
 - **fits into Open MPI semantic**
 - **automatically benefits of new PML components (e.g. DR)**
 - Contra
 - **requires META/BTL for every interconnect**
 - **requires an extra PROXY process to be started by the ORTED**
- META PML
 - Pro
 - **re-use of existing BTL**
 - Contra
 - **requires a META PML for every new functionality (e.g. DR)**
 - **doesn't fit in the current Open MPI semantic**
 - **requires manipulation of the topology information**



Conclusion

- current state
 - using Open MPI on each Cluster
 - using PAXC MPI as bridge between clusters
- future
 - extend Open MPI to work on Grid
 - Update the Open MPI implementation on the clusters and get rid of PACX MPI

