Foundations of High Performance Computing

Lecture 8: MPI libraries on ORFEO and their usage

“Foundation of HPC” course

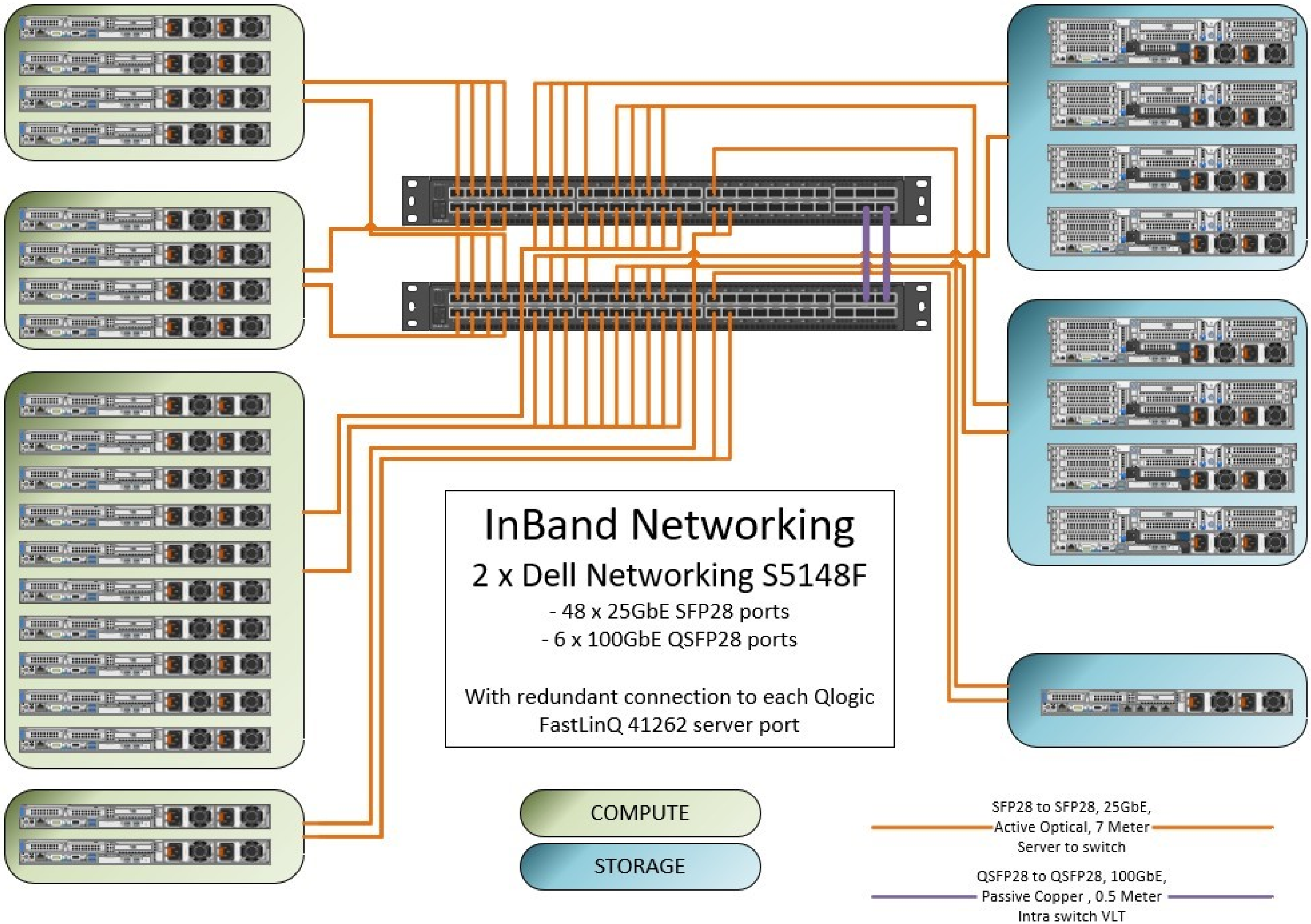
DATA SCIENCE & SCIENTIFIC COMPUTING

2022-2023 Stefano Cozzini

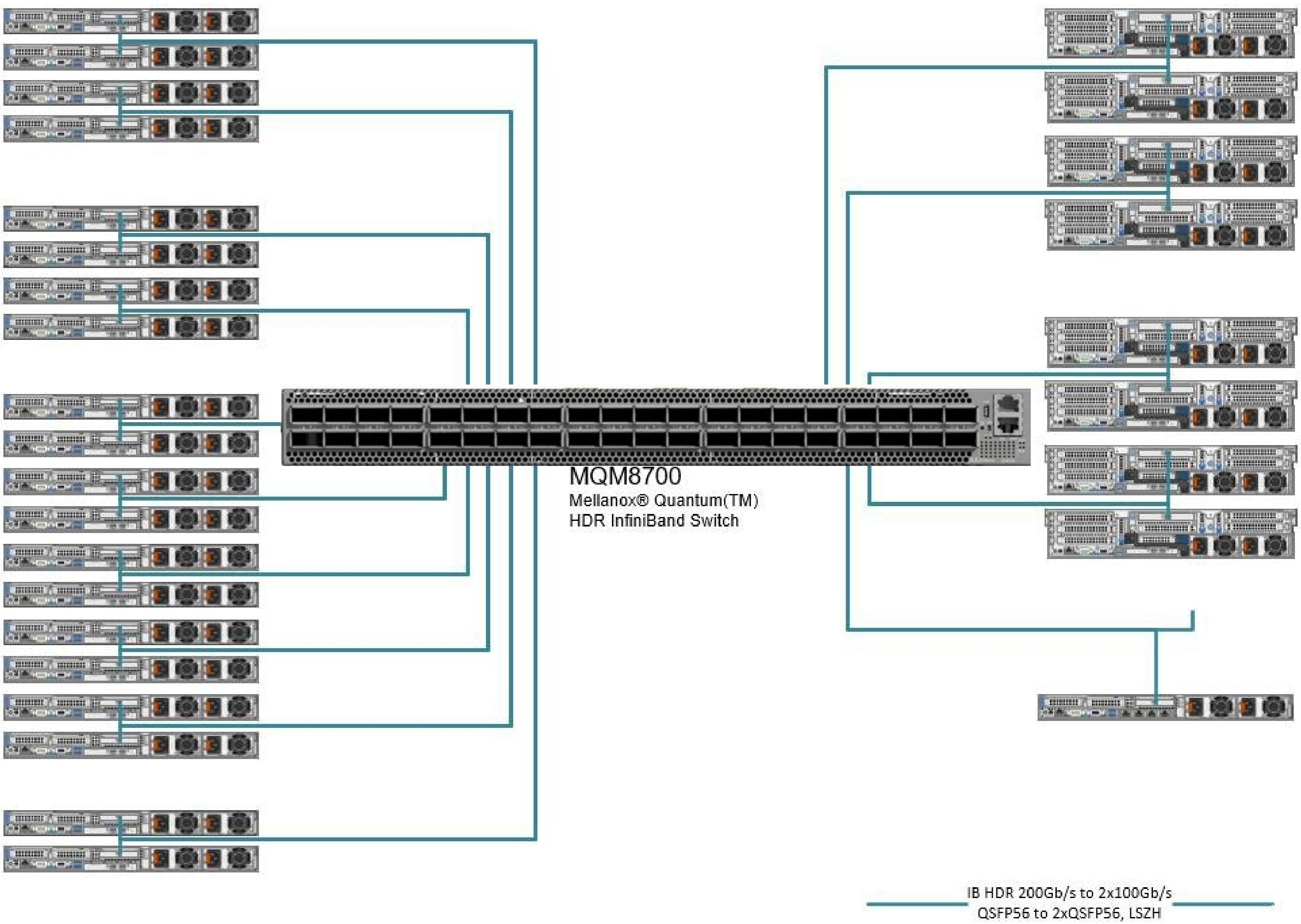
Agenda

- Recap: ORFEO networks
- Communication protocols
- MPI libraries available on ORFEO
- Measuring and understanding performance

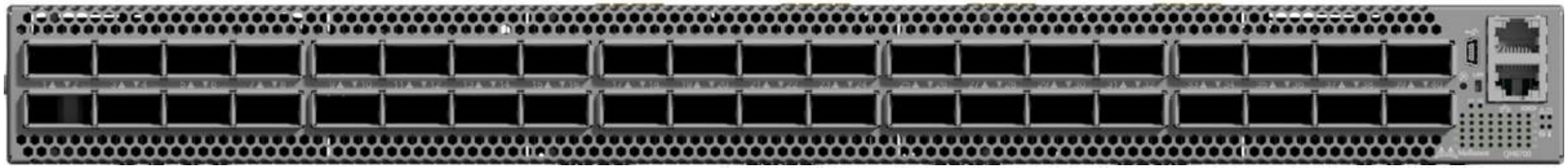
Orfeo in band management network: 25 Gbit ethernet



Orfeo High Speed network: 100 Gbit Infiniband



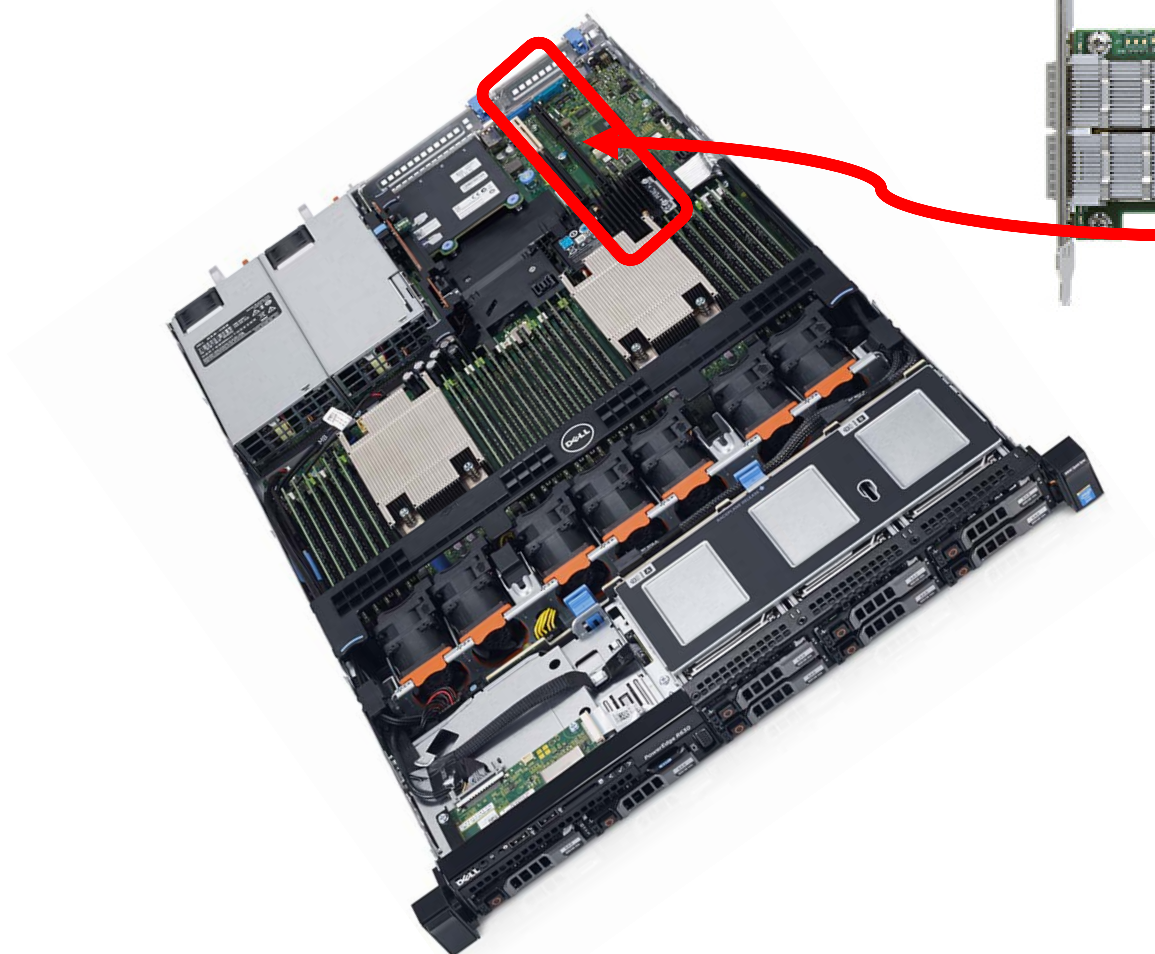
ORFEO IB network

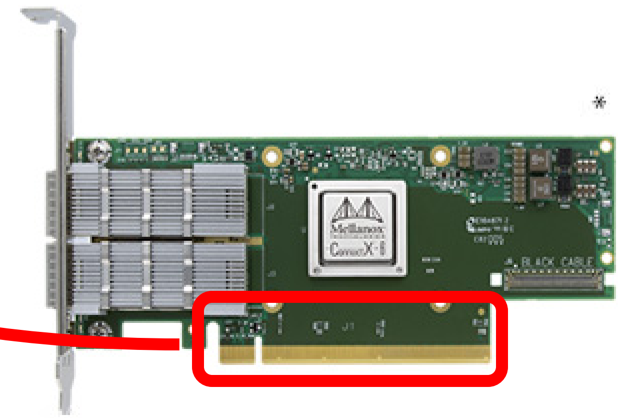


Performance
- 40 x HDR 200Gb/s ports in a 1U switch
- 80 x HDR100 100Gb/s ports (using splitter cables)
- 16Tb/s aggregate switch throughput
- Sub-130ns switch latency

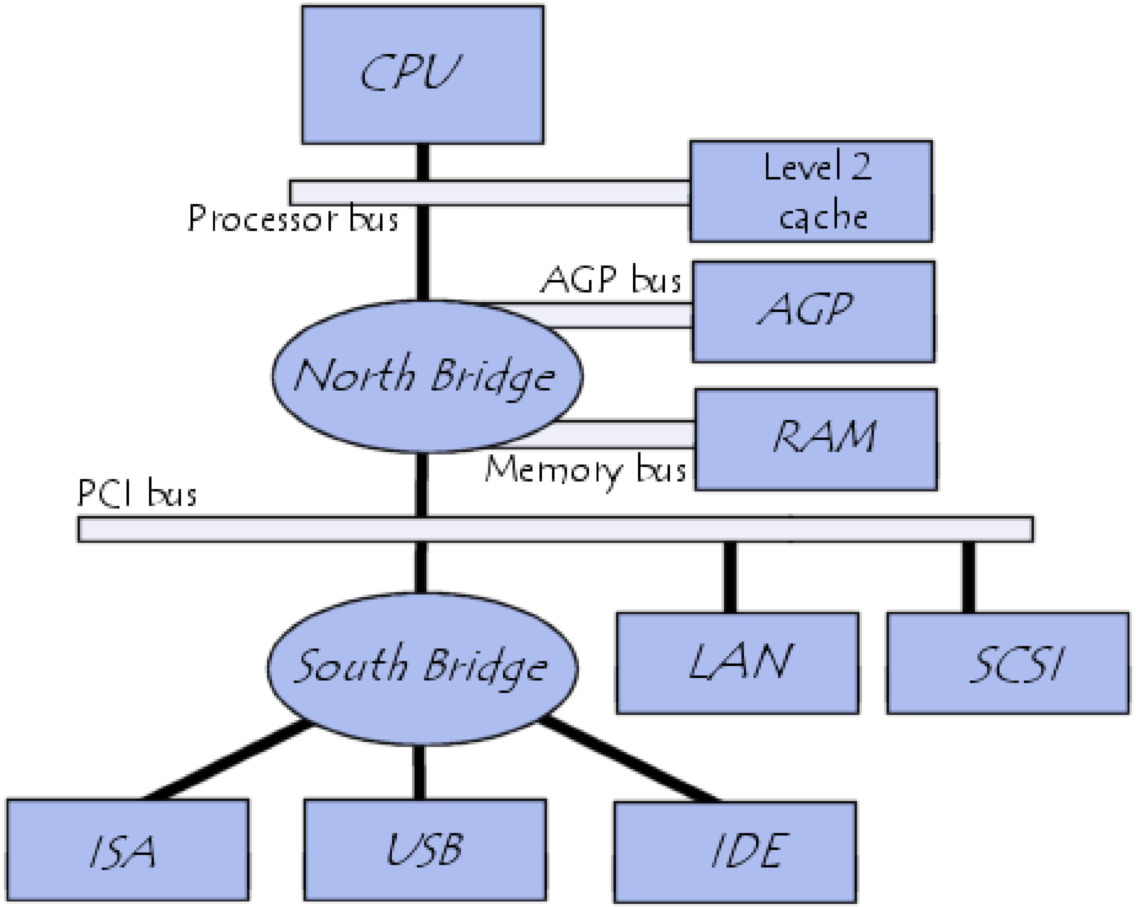
Where are the cards on the server ?

- Plugged in the PCI-e slots..





Buses within a computer (old way)



Buses on modern HPC nodes

- Peripheral Component Interconnect (PCI) buses:
  - PCI: Developed by Intel in 1992
  - several version : v3.0 last one in 2004
  - PCI-X: designed in 1999
  - 66 MHz (can be found on older servers)
  - 133 MHz (most common on modern servers)
- PCIe: designed adopted in 2004
  - version v4.0 recently released
  - Version 2.0/version 3.0 adopted on modern HPC nodes
  - Several of them on one node with different characteristics

PCI-express speed (from wikipedia)

PCI Express link performance[30][31]

PCI Express version	Introduced	Line code	Transfer rate[i]	Throughput[i]				
				×1	×2	×4	×8	×16
1.0	2003	8b/10b	2.5 GT/s	250 MB/s	0.50 GB/s	1.0 GB/s	2.0 GB/s	4.0 GB/s
2.0	2007	8b/10b	5.0 GT/s	500 MB/s	1.0 GB/s	2.0 GB/s	4.0 GB/s	8.0 GB/s
3.0	2010	128b/130b	8.0 GT/s	984.6 MB/s	1.97 GB/s	3.94 GB/s	7.88 GB/s	15.8 GB/s
4.0	2017	128b/130b	16.0 GT/s	1969 MB/s	3.94 GB/s	7.88 GB/s	15.75 GB/s	31.5 GB/s
5.0[32][33]	*expected in Q2 2019*[34]	128b/130b	32.0 GT/s[ii]	3938 MB/s	7.88 GB/s	15.75 GB/s	31.51 GB/s	63.0 GB/s

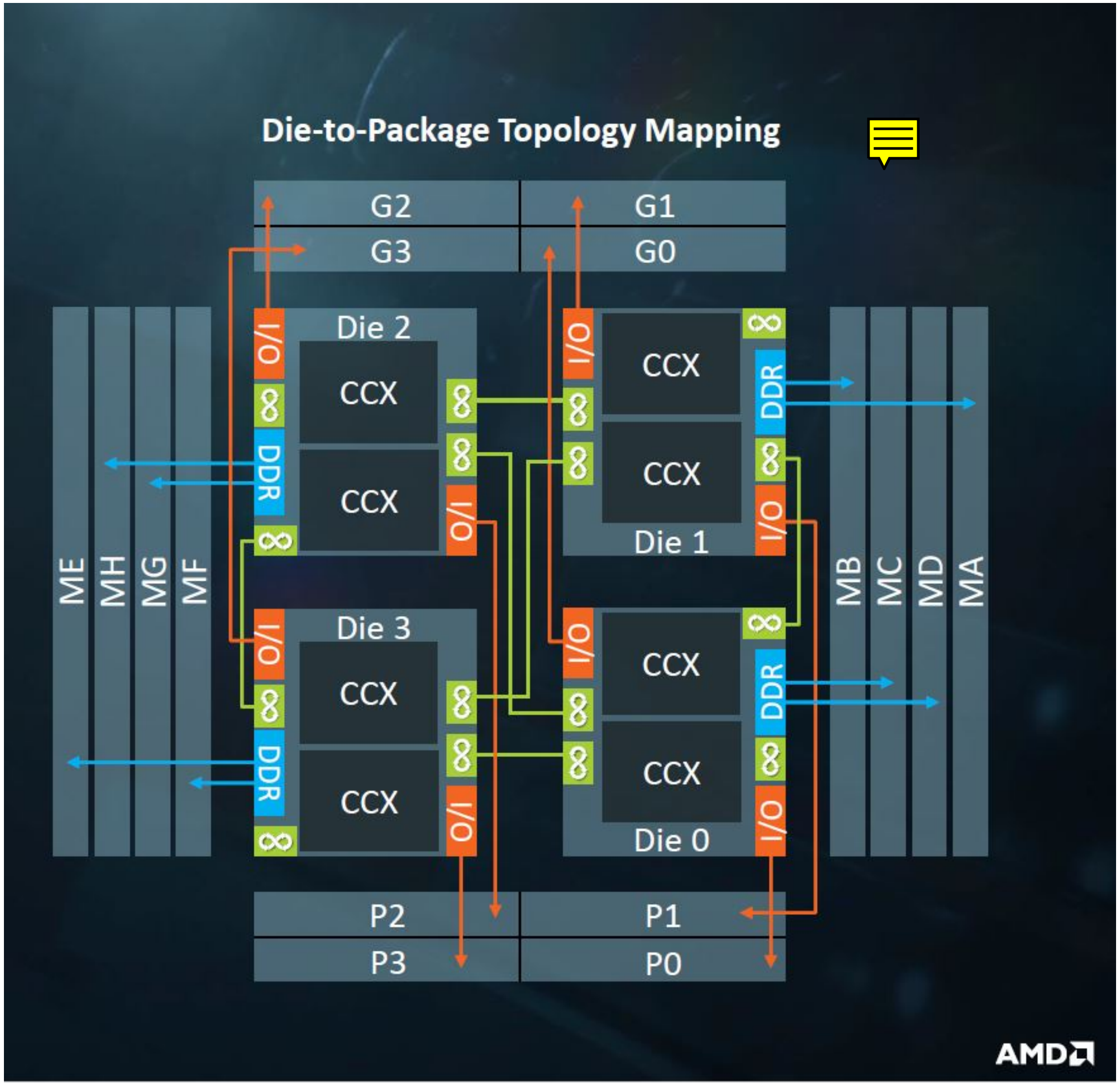
The PowerEdge R640(1U) system supports PCI express (PCIe) generation 3 expansion cards (4)

PCI buses on ORFEO epyc nodes

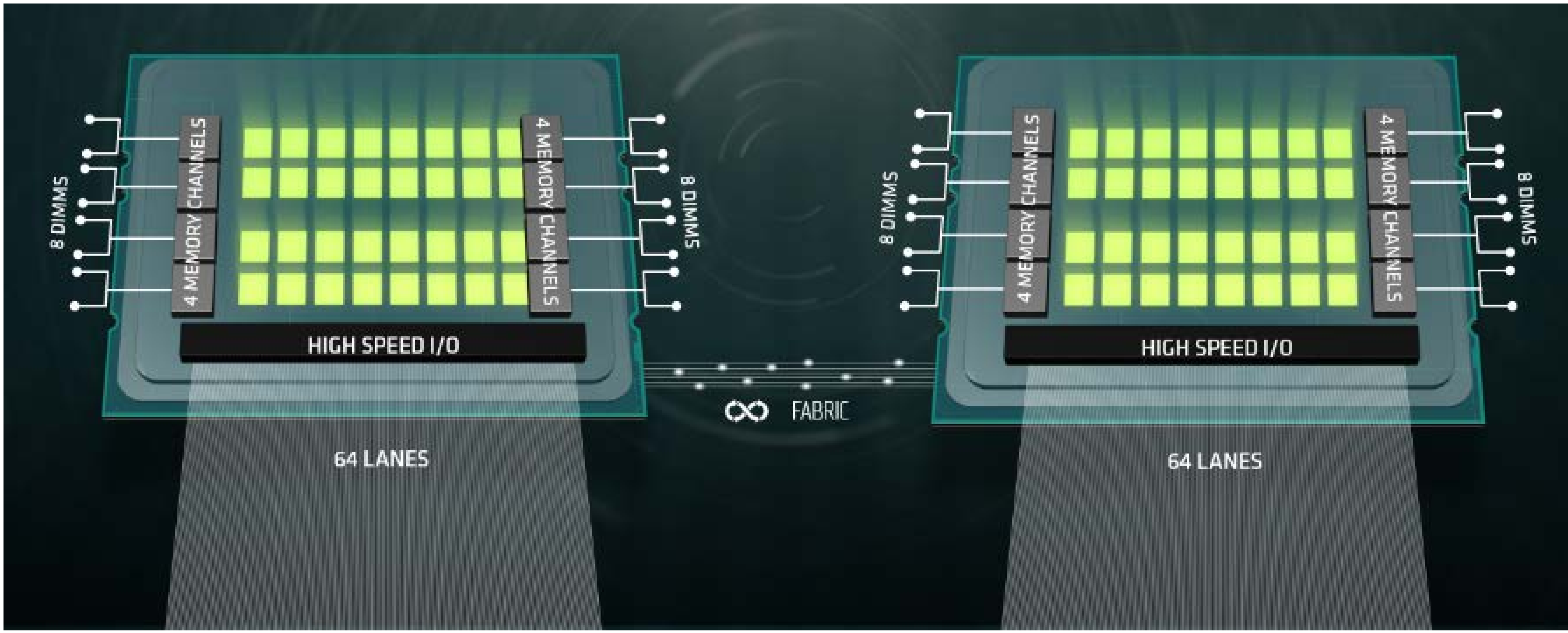
CCX is a core complex of up to 4 cores that share L3 cache. M* are the memory channels, two channels handled by each die. P* and G* are IO lanes. ∞ is the Infinity Fabric.

On a single-socket system, each die provides up to 32 PCI-E lanes using the P* and the G* IO lanes shown in Figure.

In total 128 IO lines



2 socket layout:



in a two-socket (2S) configuration, half the IO lanes of each die are used to connect to one of the dies on the other socket by using the G* IO lanes configured as Infinity Fabric. This leaves the socket with the P* IO lanes for a total of 64 PCI-E lanes and, thus, still 128 PCI-E lanes for the platform.

PCI buses on ORFEO epyc nodes

- Let us use hwloc and lstopo

Lstopo graphical output

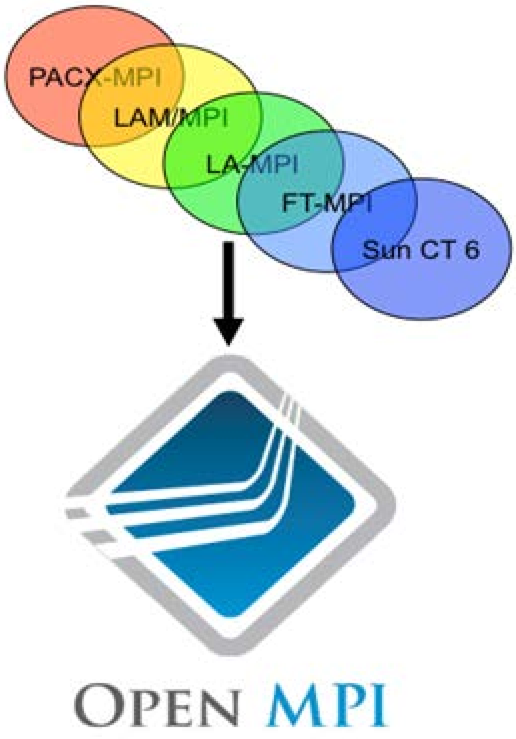
- 8 NUMA regions

MPI libraries available on ORFEO

- Open-MPI
  - Open-source
  - Portable and efficient
- Intel MPI
  - closed source
  - Fits perfectly the intel architecture

Open MPI

- Evolution of several prior MPI's Open source project and community
- Production quality
- Vendor-friendly
- Research- and academic-friendly
- MPI-3.1 compliant

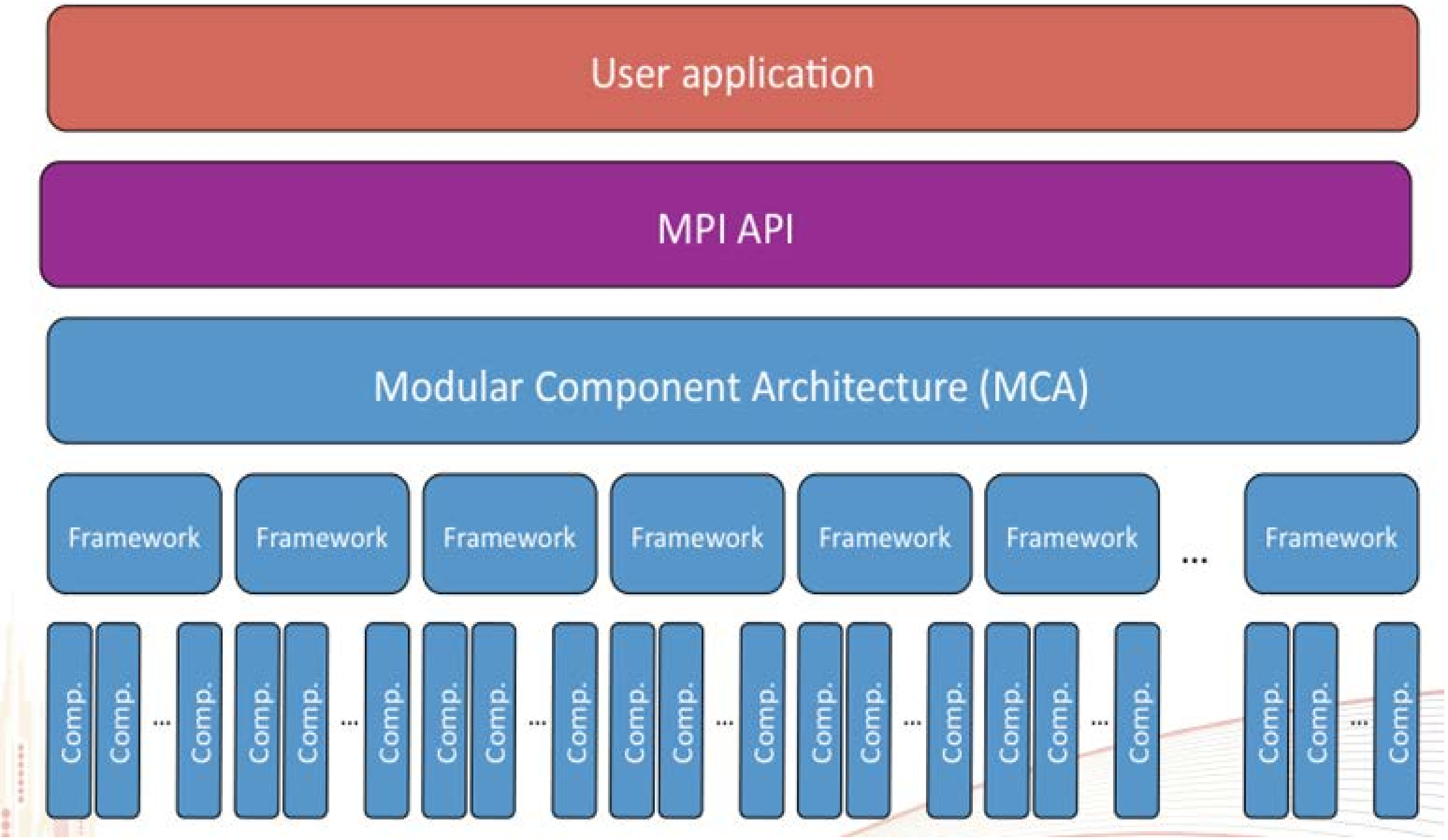


https://www.open-mpi.org/

openMPI: Plugins for HPC

- Uses Modular Component Architecture (MCA)
- Run-time plugins for combinatorial functionality
  - Underlying point-to-point network support
  - Different MPI collective algorithms
  - back-end run-time environment / scheduler support
- Extensive run-time tuning capabilities
  - Allow user or system administrator to tweak performance for a given platform

Plugin high level view



Lots and lots of plugin type

- Back-end network
- Resource manager support
- Operating system support
- All can be loaded (or not) at runtime
- Choice of network is a runtime decision

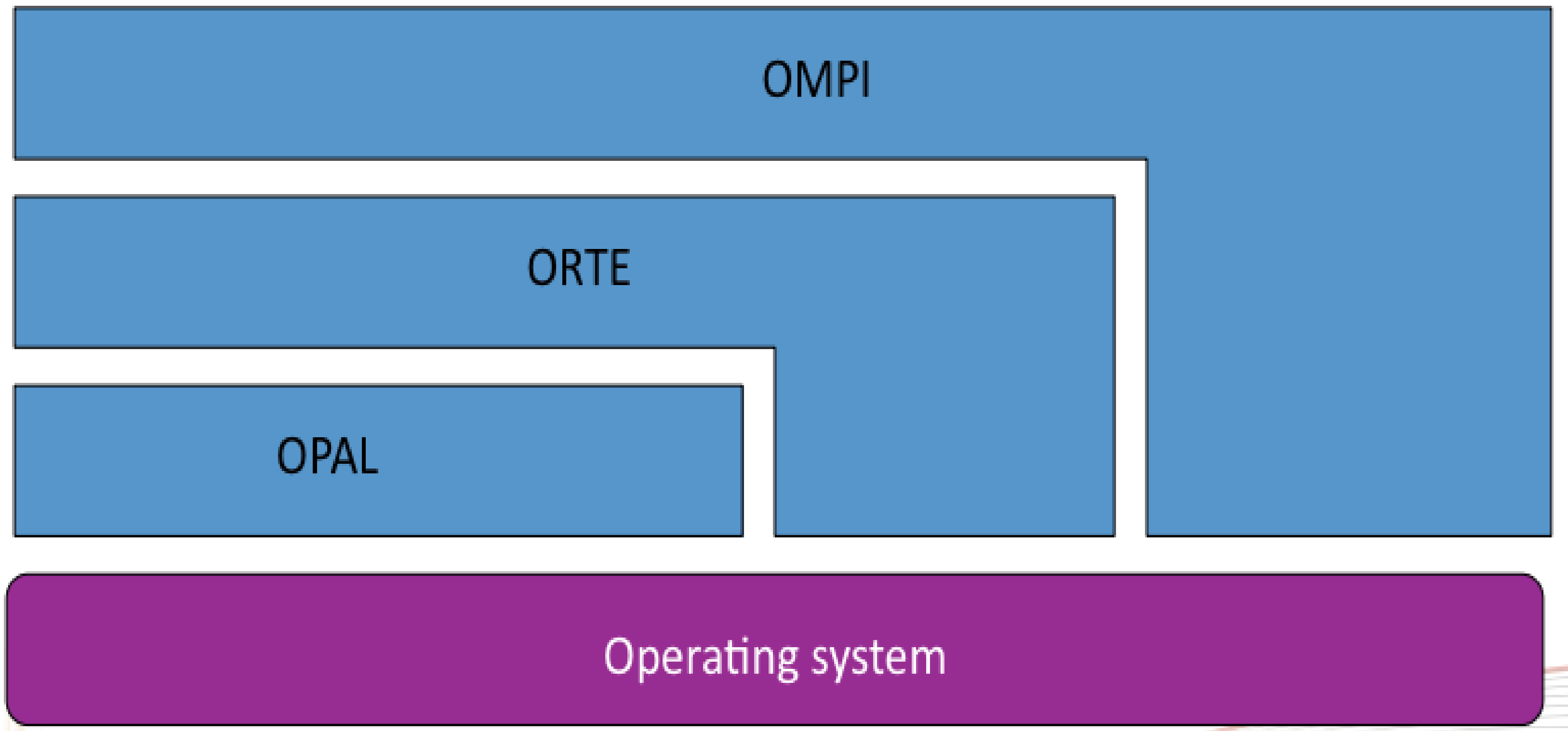
MPI frameworks (version 4.x)

- bml: BTL multipliexing layer
- btl: Byte transport layer
- coll: MPI collectives
- fbtl: MPI file byte transfer layer
- fcoll: MPI file collectives
- fs: MPI file management
- hook: Generic hooks
- io: MPI IO
- mtl: Matching transport layer
- op: MPI reduction operations
- osc: MPI one sided communications
- pml: MPI point-to-point communications
- sharedfp: MPI shared file pointer operations
- topo: MPI topologies
- vprotocol: Virtual protocol API interposition

OpenMPI software stack

- Three main section:
- OpenMPI layer (OMPI)
- OpenMPI Run-Time environment (ORTE)
- Open Portability Access Layer (OPAL)
- OMPI→ ORTE→OPAL

Graphical view



Ompi_info...

```
[cozzini@login02 ~]$ srun -n1 ompi_info | grep btl
srun: Warning: can't run 1 processes on 2 nodes, setting nnodes to 1
                 MCA btl: self (MCA v2.1.0, API v3.1.0, Component v4.1.4)
                 MCA btl: ofi (MCA v2.1.0, API v3.1.0, Component v4.1.4)
                 MCA btl: openib (MCA v2.1.0, API v3.1.0, Component v4.1.4)
                 MCA btl: tcp (MCA v2.1.0, API v3.1.0, Component v4.1.4)
                 MCA btl: usnic (MCA v2.1.0, API v3.1.0, Component v4.1.4)
                 MCA btl: vader (MCA v2.1.0, API v3.1.0, Component v4.1.4)
                MCA fbtl: posix (MCA v2.1.0, API v2.0.0, Component v4.1.4)
[cozzini@login02 ~]$ srun -n1 ompi_info | grep pml
srun: Warning: can't run 1 processes on 2 nodes, setting nnodes to 1
                 MCA pml: v (MCA v2.1.0, API v2.0.0, Component v4.1.4)
                 MCA pml: cm (MCA v2.1.0, API v2.0.0, Component v4.1.4)
                 MCA pml: monitoring (MCA v2.1.0, API v2.0.0, Component v4.1.4)
                 MCA pml: ob1 (MCA v2.1.0, API v2.0.0, Component v4.1.4)
                 MCA pml: ucx (MCA v2.1.0, API v2.0.0, Component v4.1.4)
```

Point to Point component Frameworks

- Byte Transfer Layer (BTL)
  - Abstracts lowest native network interfaces
- Point-to-Point Messaging Layer (PML)
  - Implements MPI semantics, message fragmentation, and striping across BTLs

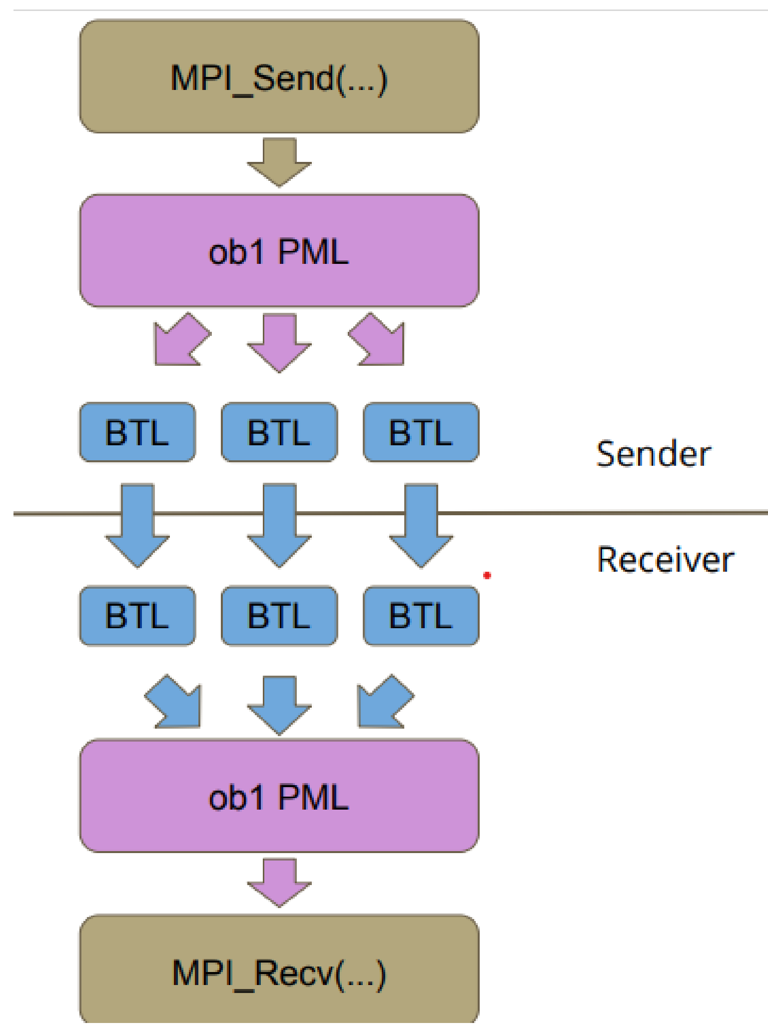
Shall we need to know all this ?

PML details..

- There are several PMLs to choose from:
  - ob1: Multi-device, multi-rail engine
    - Uses BTL components (byte transfer layer)
  - cm: Engine for matching network layers
    - Uses MTL components (matching transport layer)
  - ucx: Uses the UCX communication library (Unified Communications X)

ob1: Multi-Device, Multi-Rail Engine

- ob1 will:
  - 1. Pick BTL instance(s) that can reach a given peer
  - 2. Split large messages across relevant BTL instances
  - 3. Re-assemble messages at the receiver
- ob1 was Open MPI's original point-to-point transport engine and still works well in many environments.

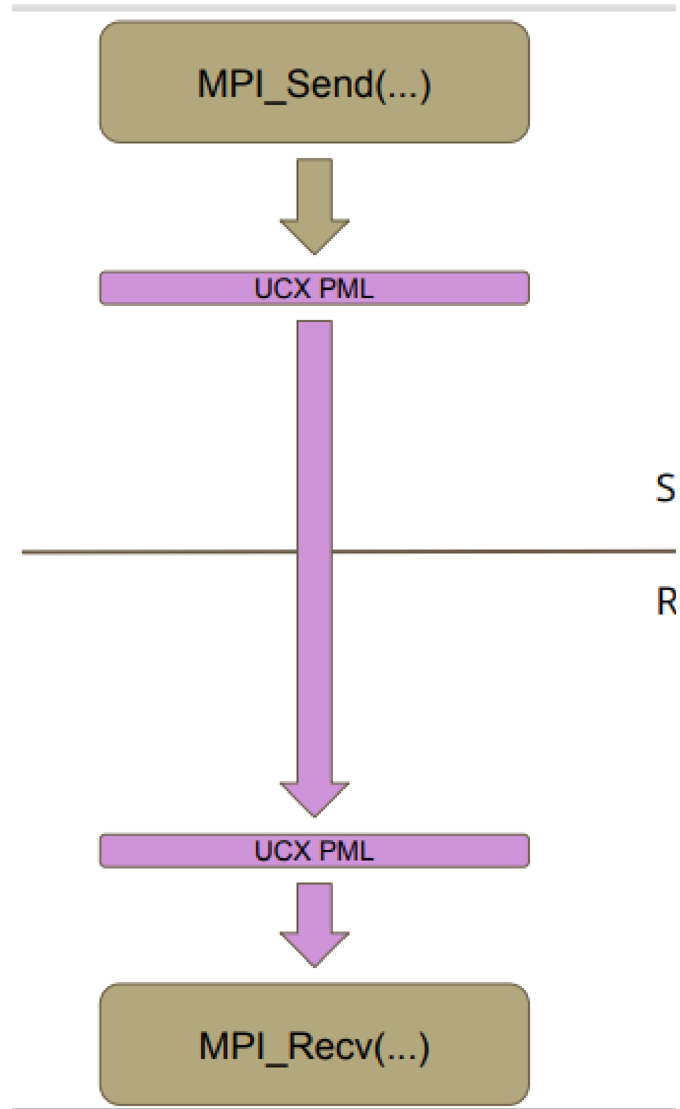


Available BTLs..

- ofi: Libfabric (OpenFabrics Interfaces)
- portals4: Portals-based networks (uncommon)
- self: Process-loopback communications
- ~~sm~~ vader: Shared memory
- smcuda: CUDA-aware shared memory
- tcp: TCP
- uct: UCX
- ugni: Cray uGNI (userspace Generic Network Interface)
- usnic: Cisco usNIC (userspace NIC)

ucx: Thin Interface to the UCX Library

- UCX is, itself, a multi-device, multi-rail transport library. It has its own engine, and therefore did not need another engine in Open MPI.
- Hence, the UCX community decided to write their own (very thin) PML and not use an existing Open MPI engine.
- NOTE: The diagram only shows the MPI code blocks (not the UCX library itself)



By default, which network gets used at run time?


```
mpirun -np 2 ./IMB-MPI1 PingPong
```

What should I do to use a different network stack ?

- Force the use of OB1 and BTLs:

```
mpirun --mca pml ob1 --mca btl [comma-delimited list]
```

- Force the use of CM and MTLs:

```
mpirun --mca pml ob1 --mca btl [comma-delimited list]
```

- Force the use of the UCX PML:

```
mpirun --mca pml ucx
```

Using OpenMPI library with ob1

- Tests to perform:

```
mpirun -np 2 --mca pml ob1 --report-bindings --map-by
node --mca btl tcp,self  ./IMB-MPI1 PingPong
```

```
mpirun -np 2 --mca pml ob1 --report-bindings --map-by
socket --mca btl tcp,self  ./IMB-MPI1 PingPong
```

```
mpirun -np 2 --mca pml ob1 --report-bindings --map-by
core --mca btl tcp,self  ./IMB-MPI1 PingPong
```

See tutorial in MPI directory

MPI Intel library

- Intel® MPI Library is a multi-fabric message-passing library that implements the open-source MPICH specification.
- Highly tuned on HPC clusters based on Intel® processors.
  - Achieve the best latency, bandwidth, and scalability through automatic tuning for the latest Intel® platforms.
- Fully integrated in the Intel Cluster edition with compilers, math libraries and performance tuner/analizer..

Using Intel MPI

- Load module
- Check wrapper:

```
[cozzini@ct1pg-gnode001 src_c-intel]$ mpicc -v
mpigcc for the Intel(R) MPI Library 2019 Update 9 for Linux*
Copyright 2003-2020, Intel Corporation.
Using built-in specs.
COLLECT_GCC=gcc
…
Thread model: posix
gcc version 4.8.5 20150623 (Red Hat 4.8.5-39) (GCC)
[cozzini@ct1pg-gnode001 src_c-intel]$ mpiicc -v
mpiicc for the Intel(R) MPI Library 2019 Update 9 for Linux*
Copyright 2003-2020, Intel Corporation.
icc version 19.1.3.304 (gcc version 4.8.5 compatibility)
```

Using Intel MPI

- Run benchmark:

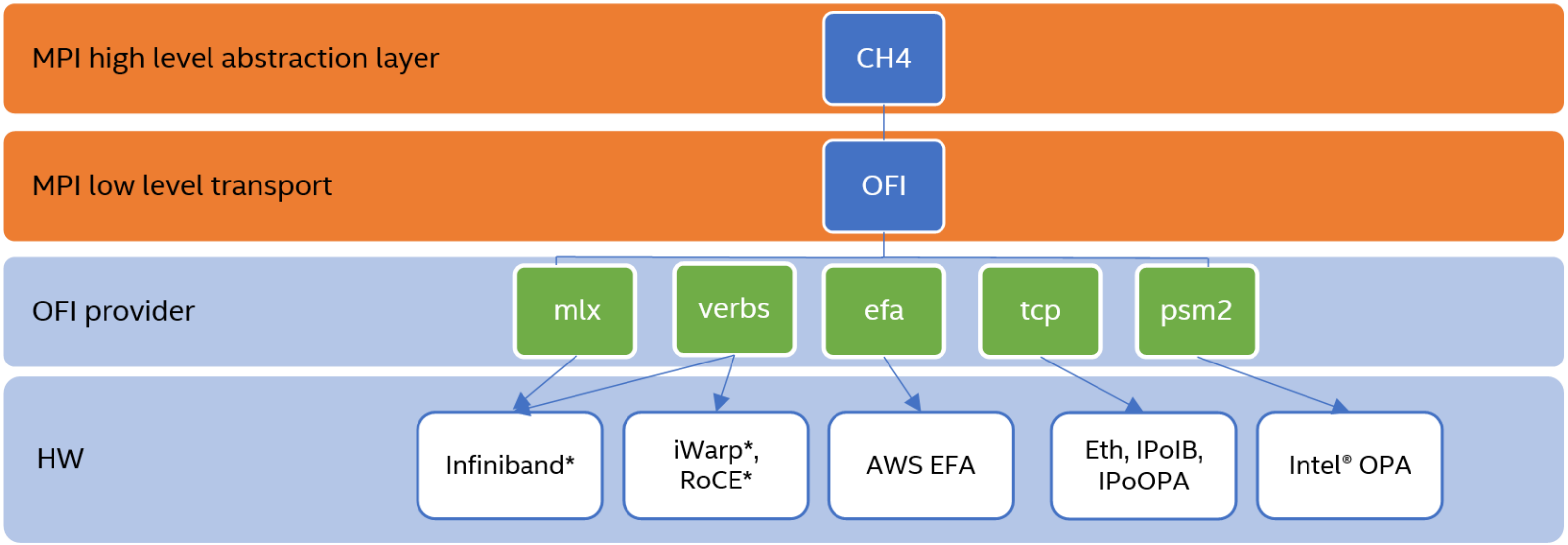
```
cozzini@ct1pg-gnode00 ]$ which mpirun
/opt/area/shared/programs/x86_64/intel/parallel_studio_xe_2020_update4_clus
ter_edition/compilers_and_libraries_2020/linux/mpi/intel64/bin/mpirun

[cozzini@ct1pg-gnode001]$ mpirun -np 2 ./IMB-MPI1 PingPong -msglog 4
 ….
#-----------------------------------------------------
# Benchmarking PingPong
# #processes = 2
#-----------------------------------------------------
       #bytes #repetitions      t[usec]   Mbytes/sec
            0         1000         0.47         0.00
            1         1000         0.47         2.12
            2         1000         0.47         4.26
            4         1000         0.47         8.52
            8         1000         0.47        17.06
           16         1000         0.47        34.10

# All processes entering MPI_Finalize
```

Tuning Intel MPI

- Intel MPI Library software stack:



- The Intel® MPI Library will attempt to select the fastest available fabric by default,

Tuning Intel MPI

- Pinning MPI process on specific processors:

I_MPI_PIN_PROCESSOR_LIST generates a custom process to processor map with one of the three alternative syntax available:

<proclist>

<procset>][:map=<map>]

<procset>][:[grain=<grain>][,shift=<shift>][,preoffset =<preoffset>][,postoffset=<postoffset>]]

Some examples:

- Run on the two contiguous processors:

```
[cozzini@ct1pg-gnode001 src_c-intel]$ mpirun -np 2 -ppn=2  -env I_MPI_DEBUG 5 -genv I_MPI_PIN_PROCESSOR_LIST
0,1 ./IMB-MPI1 PingPong -msglog 4
[0] MPI startup(): Intel(R) MPI Library, Version 2019 Update 9  Build 20200923 (id: abd58e492)
[0] MPI startup(): Copyright (C) 2003-2020 Intel Corporation.  All rights reserved.
[0] MPI startup(): library kind: release
[0] MPI startup(): libfabric version: 1.10.1-impi
[0] MPI startup(): libfabric provider: mlx
[0] MPI startup(): Rank    Pid      Node name       Pin cpu
[0] MPI startup(): 0       32018    ct1pg-gnode001  0
[0] MPI startup(): 1       32019    ct1pg-gnode001  1
[0] MPI startup():
I_MPI_ROOT=/opt/area/shared/programs/x86_64/intel/parallel_studio_xe_2020_update4_cluster_edition//compilers
_and_libraries_2020/linux/mpi
[0] MPI startup(): I_MPI_MPIRUN=mpirun
[0] MPI startup(): I_MPI_HYDRA_RMK=pbs
[0] MPI startup(): I_MPI_HYDRA_TOPOLIB=hwloc
[0] MPI startup(): I_MPI_PIN_PROCESSOR_LIST=0,1
[0] MPI startup(): I_MPI_INTERNAL_MEM_POLICY=default
[0] MPI startup(): I_MPI_DEBUG=5

#----------------------------------------------------
# Benchmarking PingPong
# #processes = 2
#----------------------------------------------------
       #bytes #repetitions      t[usec]   Mbytes/sec
            0         1000         0.47         0.00
            1         1000         0.47         2.11
            2         1000         0.47         4.22
            4         1000         0.47         8.44
            8         1000         0.47        16.87
           16         1000         0.47        33.90
```

Some examples:

- Run on the on the same socket:

```
[cozzini@ct1pg-gnode001 src_c-intel]$ mpirun -np 2  -env I_MPI_DEBUG 5 -genv I_MPI_PIN_PROCESSOR_LIST 0,2
./IMB-MPI1 PingPong
[0] MPI startup(): Intel(R) MPI Library, Version 2019 Update 9  Build 20200923 (id: abd58e492)
[0] MPI startup(): Copyright (C) 2003-2020 Intel Corporation.  All rights reserved.
[0] MPI startup(): library kind: release
[0] MPI startup(): libfabric version: 1.10.1-impi
[0] MPI startup(): libfabric provider: mlx
[0] MPI startup(): Rank    Pid      Node name       Pin cpu
[0] MPI startup(): 0       32479    ct1pg-gnode001  0
[0] MPI startup(): 1       32480    ct1pg-gnode001  2
[0] MPI startup():
I_MPI_ROOT=/opt/area/shared/programs/x86_64/intel/parallel_studio_xe_2020_update4_cluster_edition//compilers
_and_libraries_2020/linux/mpi
[0] MPI startup(): I_MPI_MPIRUN=mpirun
[0] MPI startup(): I_MPI_HYDRA_RMK=pbs
[0] MPI startup(): I_MPI_HYDRA_TOPOLIB=hwloc
[0] MPI startup(): I_MPI_PIN_PROCESSOR_LIST=0,2
[0] MPI startup(): I_MPI_INTERNAL_MEM_POLICY=default
[0] MPI startup(): I_MPI_DEBUG=5
…..
# Benchmarking PingPong
# #processes = 2
#----------------------------------------------------
       #bytes #repetitions      t[usec]   Mbytes/sec
            0         1000         0.28         0.00
            1         1000         0.29         3.47
            2         1000         0.29         6.91
            4         1000         0.29        13.84
            8         1000         0.29        27.84
```

Comparing performance

- Left to the readers… ☺

- A starting point for intel:

  [Tuning the Intel® MPI Library: Basic Techniques]

Final considerations 1

- Why latency is so important ?
- According to Amdahl's law:
  - a high-performance parallel system tends to be bottlenecked by its slowest sequential process
- in all but the most embarrassingly parallel supercomputer workloads, the slowest sequential process is often the latency of message transmission across the network

Final considerations 2

- In general the compute/communication ratio in a parallel program remains fairly constant.
- So as the computational power increases the network speed must also be increased.
- We are living in a multi-core world: MPI processes sharing the same network device !
- Contention for the interconnect device can have a significant impact on performance.