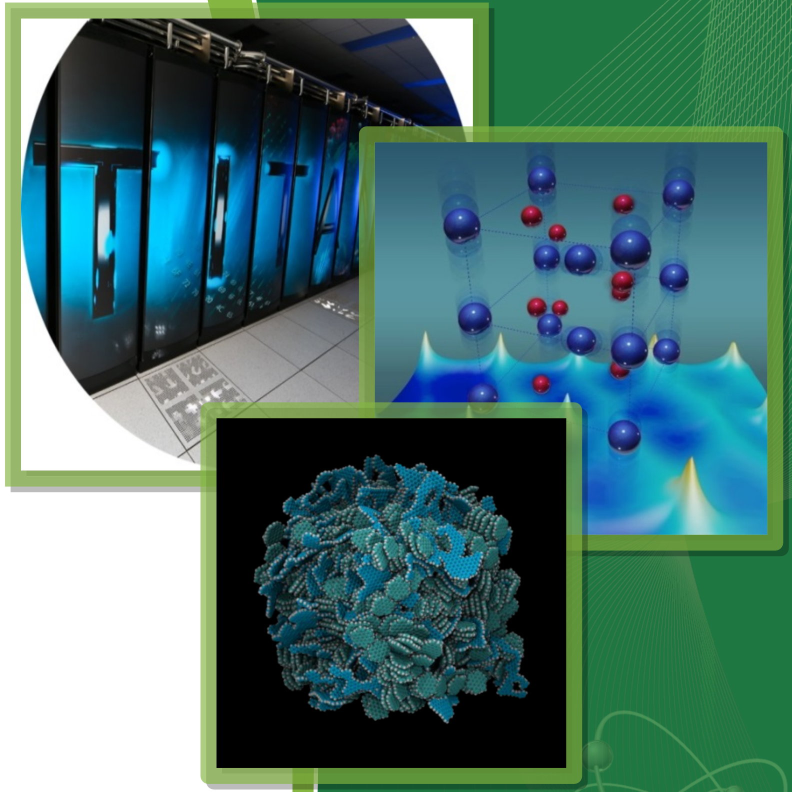
Support for Complex Hardware Topologies in MPI

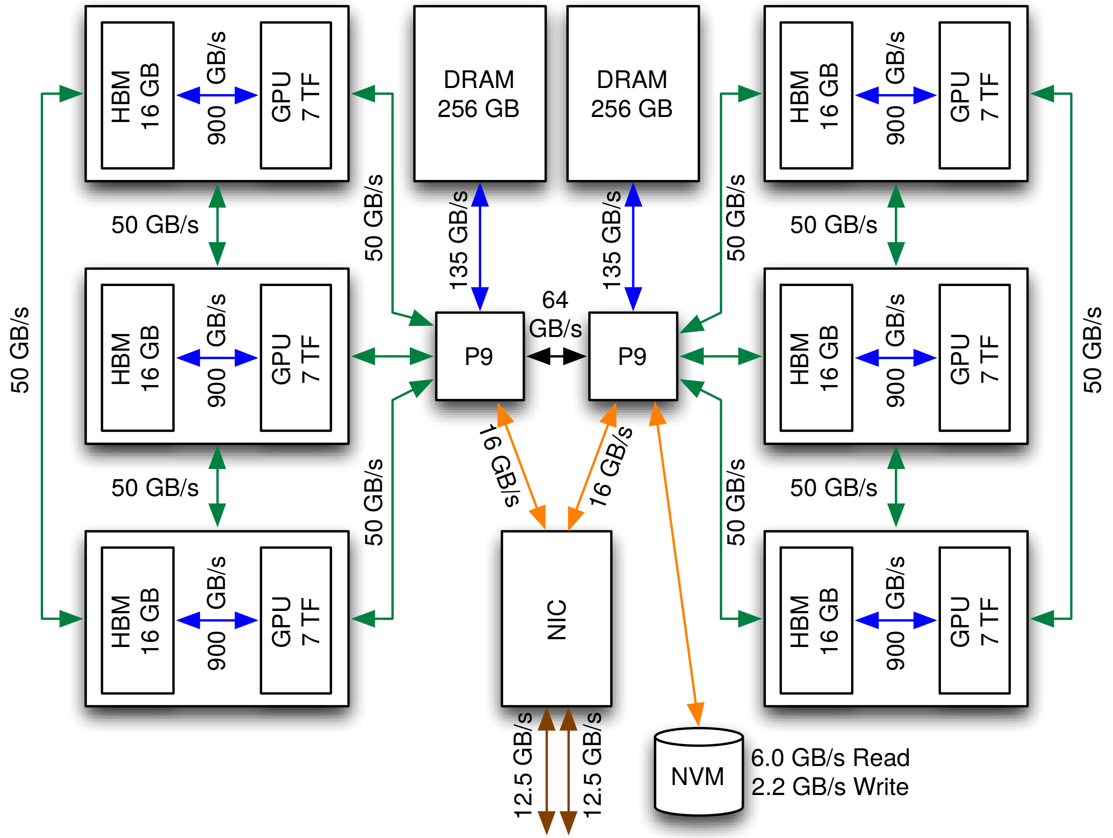
Geoffroy Vallee (ORNL)



ORNL is managed by UT-Battelle
for the US Department of Energy

Background

- Current and upcoming hardware are becoming more complex
  - Various hardware links: PCIe, NV Link, Gen-Z, etc.
  - Various memory technologies: NVe, HBM, DDR, etc.
- Hardware architectures now based on graph representations (not trees)
  - On Summit, NV Link is already creating a loop between GPUs
  - Cannot be represented as a tree anymore



TF	42 TF (6x7 TF)
HBM	96 GB (6x16 GB)
DRAM	512 GB (2x16x16 GB)
NET	25 GB/s (2x12.5 GB/s)
MMsg/s	83

⟷ HBM/DRAM Bus (aggregate B/W)
⟷ NVLINK
⟷ X-Bus (SMP)
⟷ PCIe Gen4
⟷ EDR IB

HBM & DRAM speeds are aggregate (Read+Write).
All other speeds (X-Bus, NVLink, PCIe, IB) are bi-directional.

How can we expose such architectures to applications through MPI?

MPI – What already exists?

- MPI already supports using concepts:
  - Concept of topology
  - Concept of distributed graph
  - Groups of MPI processes
- Graph topologies
  - Provides a mean to organize MPI processes
  - Limitations
    - Virtual topology (*I tell MPI about the processes I know*)
    - Always create a communicator (can be expensive)
    - MPI_DIST_GRAPH_CREATE(comm_old, n, sources, degrees, destinations, weights, info, reorder, comm_dist_graph)

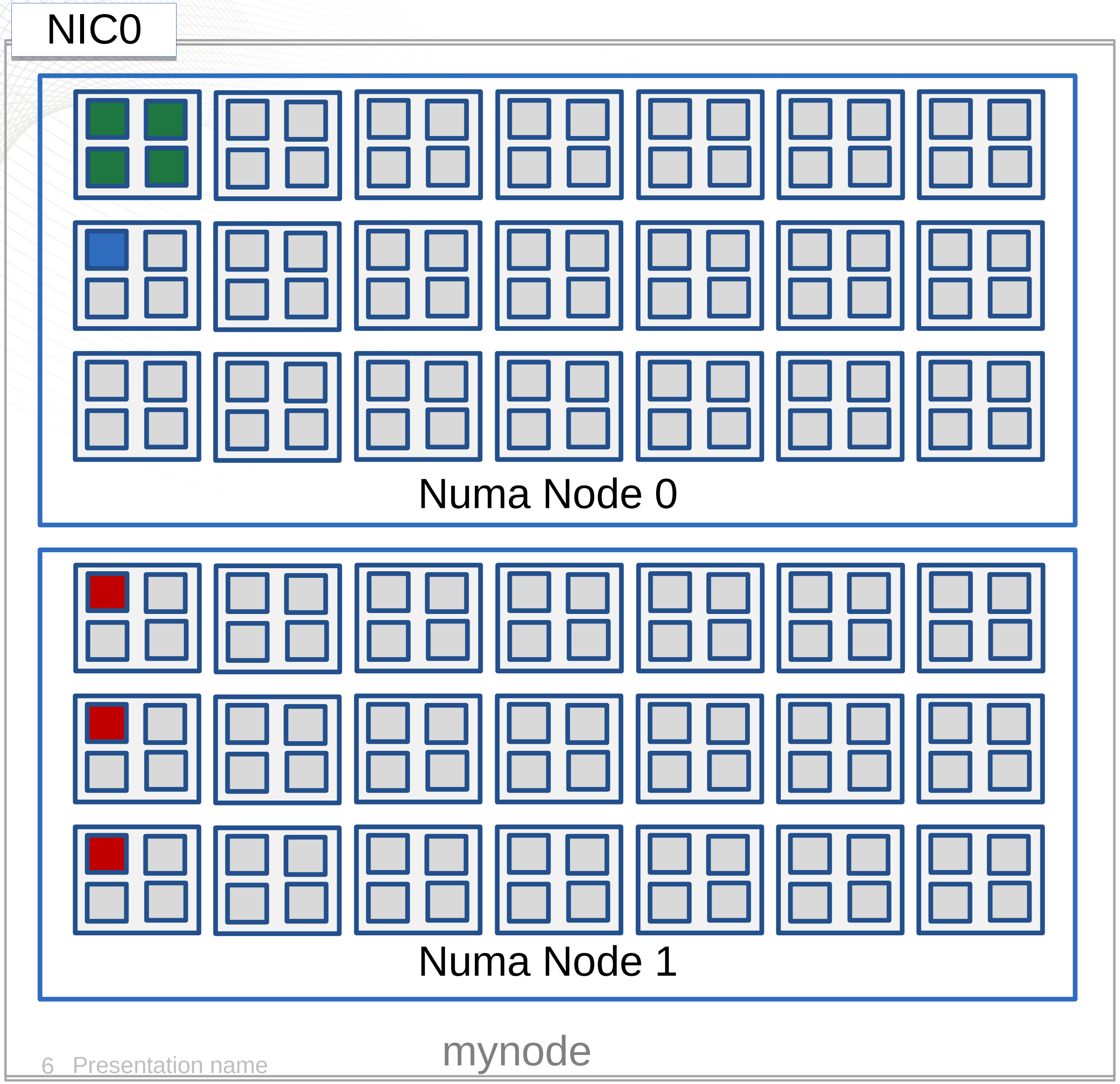
What would be beneficial?

- Get all MPI processes running “near by” – *MPI tell me about the processes that share resources with me*
  - On the same L1, L2, L3, NUMA node, ...etc.
  - No implicit creation of communicators which may never be used
  - Example
    - MPI rank is sharing L1 with rank 42
    - MPI rank is sharing NUMA nodes with ranks 42, 44, 45, 80
    - MPI rank is sharing the compute node with ranks 42, 44, 45, 80, 101, 110, 122
- Try to leverage concepts from other proposals
- Can support future architectures

Proposal 1 – Abstracting the underlying architecture

- Rely on groups
- Abstract the actually hardware architecture

Proposal 1 – Conceptual example



- We assume MPI processes are numbered sequentially starting from 0
- Green processes get at least 3 groups
  - 0, 1, 2, 3 with URI “uri://core_0”
  - 0, 1, 2, 3, 4 with URI “uri://numa_0”
  - 0, 1, 2, 3, 4, 5, 6, 7 with URI “uri://mynode”
- Blue process gets 2 groups
  - 0, 1, 2, 3, 4 with URI “uri://numa_0”
  - 0, 1, 2, 3, 4, 5, 6, 7 with URI “uri://mynode”
- Red processes get 2 groups
  - 5, 6, 7 with URI ”uri://numa_1”
  - 0, 1, 2, 3, 4, 5, 6, 7 with URI “uri://mynode”

Proposal 1 – New MPI function

MPI_Hw_groups_get (MPI_Group groups[], uris, sizes)

Local operation that queries the runtime about group sharing resources

- OUT groups Array of groups representing MPI processes sharing hardware resources
- OUT uris Array of unique group identifiers, assigned by the runtime, representing the node type in hardware graph
- OUT sizes Number of groups

Note: uris are a similar concept than the one proposed in other proposals, but create a naming issue

Proposal 1 – Code example (1)

- Create communicators for all hardware levels involved in resource sharing

```
MPI_Hw_groups_get (&groups, &labels, &num_gps);
MPI_Comm *new_comms = malloc (num_gps * sizeof (MPI_Comm));
for (i = 0; i < num_gps; i++)
{
        MPI_Comm_create_from_group (groups[i], &new_comms[i]); /* Function from the session proposal */
}
```

Proposal 1 – Code example (2)

- Create a communicator for all MPI processes on the same NUMA node

```
MPI_Hw_groups_get (&groups, &labels, &num_gps);
MPI_Comm *new_comms = malloc (num_gps * sizeof (MPI_Comm));
for (i = 0; i < num_gps; i++)
{
        if (uricmp (uris[i], “uri://compute_node”) == 0)
                  MPI_Comm_create_from_group (groups[i], &new_comm);
}
```

Proposal 1 – Benefits/Limitation Analysis

- Pros
  - Guarantee separation between resource manager, runtime and the MPI standard
  - Local operation / No creation of global handles/objects
  - Leverage many concepts from the Session proposal
- Cons
  - Node centric
  - URI concept (from the Session proposal) is still unclear, lacking useful APIs
  - The underlying graph is not directly exposed to applications (possible but potentially require more extensive new APIs)
  - Does not expose the potential speed different for the link between MPI processes (PCIe versus NVLink)

Proposal 2 – Exposing the underlying architecture

- Rely on existing MPI functions
- Add a single new function that gets the distributed graph from group of MPI processes
- Require a single new function, ”reverse” of MPI_Dist_graph_create()

MPI_Group_get_hw_dist_graph (gp, in, src[], src_w[], out, dest[], dest_w[])

Returns data that represents the underlying hardware through a distributed graph for a given group.

IN gp Group from which we want to the distributed graph representing the underlying hardware.

OUT in Size of src and src_w

OUT src MPI processes for which the calling process is a destination

OUT src_w Weights of the edges into the calling process (hardware relative distance)

OUT out Size of dest and dest_w

OUT dest MPI processes for which the calling process is a source

OUT dest_w Weights of the edges from the calling process (hardware relative distance)

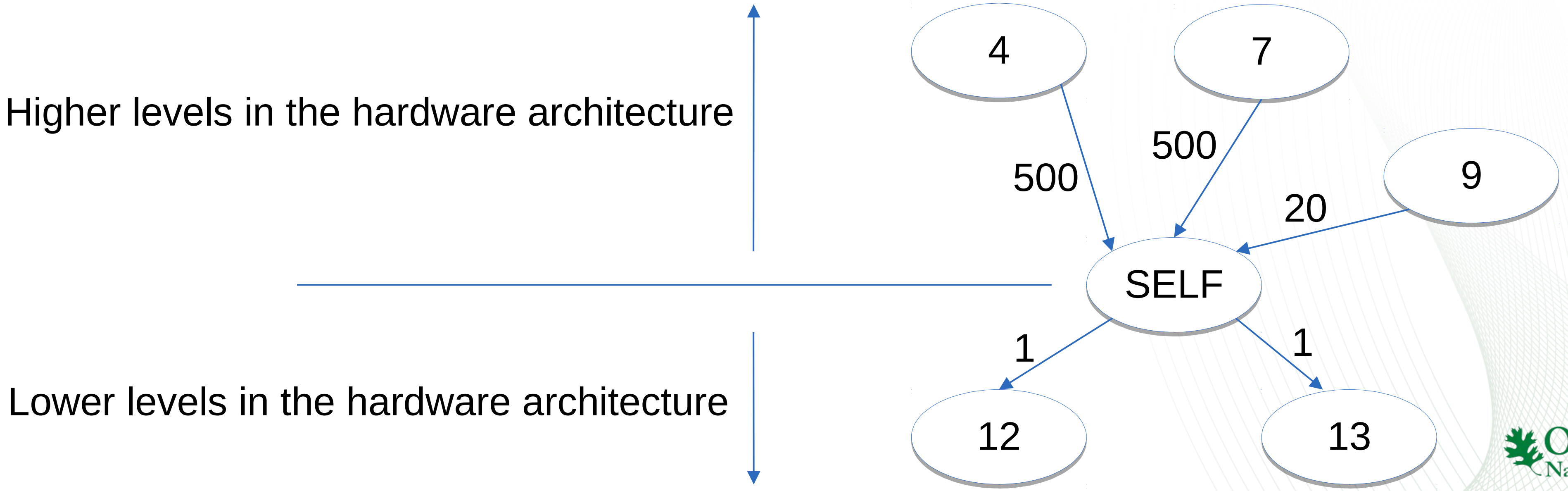
Illustration

- Get local representation of all local MPI processes

```
/* Get the MPI processes that are running on the compute node */
MPI_Comm_split_type (MPI_COMM_WORLD, MPI_COMM_TYPE_SHARED, 0, NULL, &my_local_comm);

/* Get group of local MPI processes */
MPI_Comm_group (my_local_comm, &my_local_gp);

/* Get the distributed graph from the group */
MPI_Group_get_hw_dist_graph (my_local_gp, &in, &src[], &src_w[], &out, &dest[], &dest_w);
```


Higher levels in the hardware architecture
Lower levels in the hardware architecture
4
7
9
500
500
20
SELF
1
1
12
13

Proposal 2 – Benefits/Limitation Analysis

- Pros
  - Guarantee separation between resource manager, runtime and the MPI standard
  - Can be used to expose more than local resources (network neighbors)
  - Expose the underlying hardware
  - Scalable
  - No naming issue (URIs)
- Cons
  - MPI does not provide a type for graphs so it handles a graphs through arrays
  - Like with MPI_Dist_graph, we only provide a distributed (but scalable) representation of the hardware