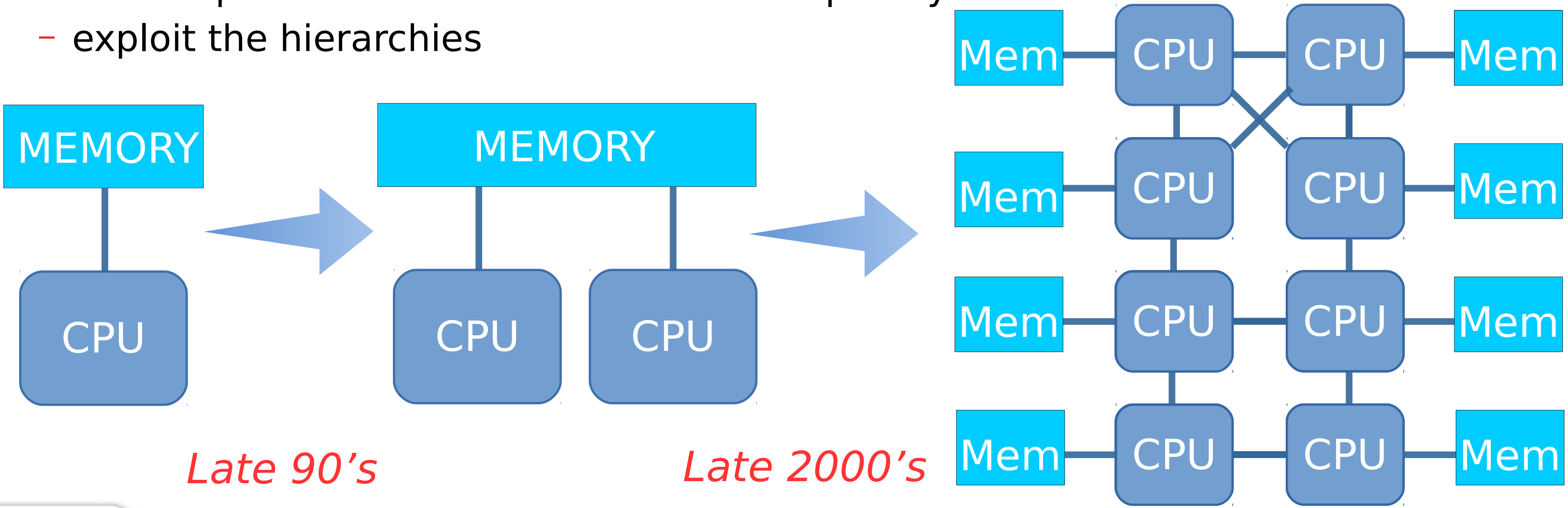
Hardware Topologies Management in Message-Passing Based Parallel Applications

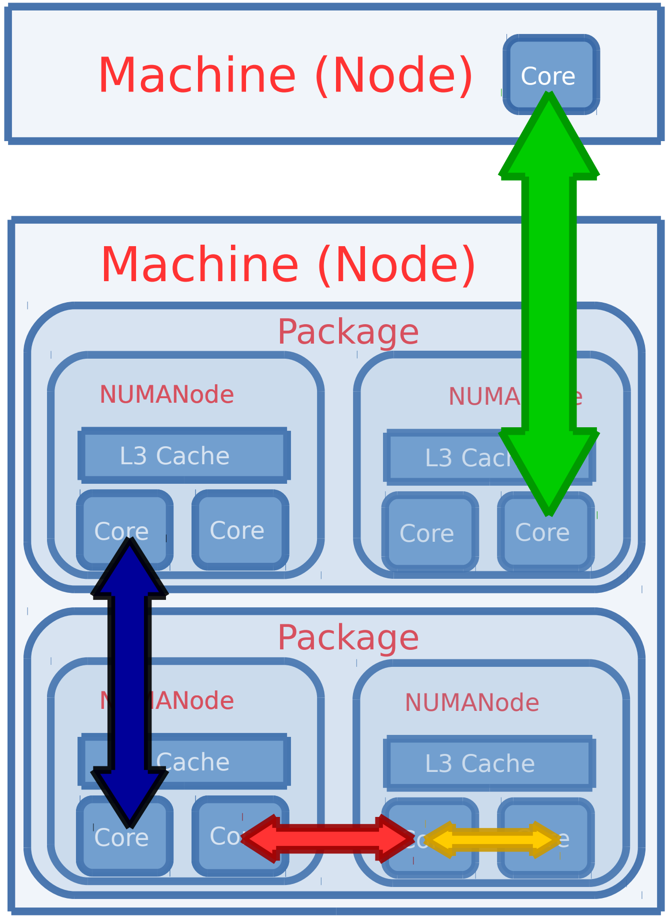
Guillaume Mercier

Parallel Computers are increasingly complex

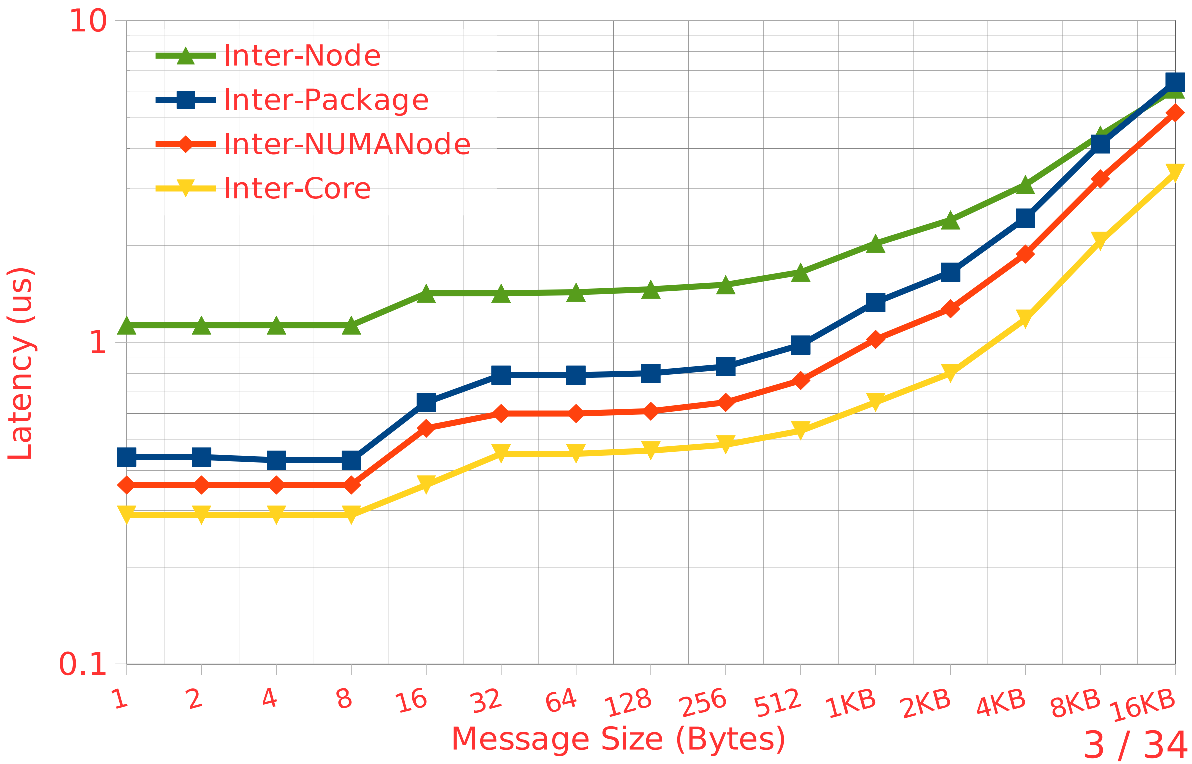
- Current trend in computing nodes
  - Increasing core count
  - Deeper memory hierarchies
- Regular users cannot understand/exploit all of this
- Some help is needed to deal with this complexity
  - exploit the hierarchies



The Performance Heterogeneity Issue



➔ Depending on process **location** on **cores**
- **Communication performance** can differ
- **Memory access costs** can change (NUMA effects)



Haswell E5-2680v3 + Infiniband QDR interconnect

Motivations

- Application developers need abstract features to:
  - Deal with hardware characteristics (Caches, Interconnect, Cores, NUMA nodes, etc.)
    - **Example: Selection of a set of hardware resources in an application**
  - Deal with existing low level tools
    - hwloc
    - numactl
    - Etc.
- Expected performance improvements
  - Improved locality
  - Improved communication performance

The Message Passing Interface and its programming model

- MPI is widely used for parallel applications since the mid-90’s
- Relies on **message exchanges** between processing entities (MPI processes)
- MPI programming model is **flat**
  - Any MPI process can communicate with any other MPI process
  - No routing scheme
  - No locality/hierarchy in the model originally
- MPI is **hardware-agnostic**
  - No assumptions made about the underlying hardware
  - Does not mean that it cannot be accessed at the application level

Hardware and Locality Management in the current (version 3.1) MPI Standard

- Some additions since `MPI_Get_processor_name`
- Currently available features:
  - Virtual topologies
  - Neighborhood Collective Communications
  - Shared Memory Support in MPI

MPI Virtual Topologies

- “**Software locality**”: characterize the application behavior (e.g., its communication pattern)
  - HW independent feature (interface-wise)
  - Virtual to physical mapping “outside of the scope of MPI”
- HW can be taken into account with **the reordering of processes**: possibility to match the HW topology to the virtual topology
  - **Implementation-dependent feature**
    - **Not standard**
    - No guarantee of behavior from one implementation to the other, or even from one version to the other
- Implementation issues
  - Virtual topologies poorly implemented
  - Chicken-and-egg problem

Neighborhood Collective Communications

- Leverage virtual topologies
  - A virtual topology is attached to a communicator
  - It represents the application’s **communication pattern**
- This communicator is an argument of the NCC function
  - Users can **define their own communication pattern** for the collective communication
    - **Better control** of how communications are scheduled
  - Refines the flat programming model of MPI
    - **Improves scalability and locality**

MPI and Shared Memory Support

- Until MPI 3.0, shmem support is **hidden** by the implementation
  - ➔ **Gain control** over the actual shared memory transfers
- **Hybrid programming** (e.g., MPI + X) is now commonplace
  - MPI + OpenMP (i.e. multithreading) is often used
    - Because of multicore nodes
    - Sensible approach (sometimes counter-intuitive)
  - **MPI + MPI is also possible!**
    - Message Passing for internode communications
    - MPI Shared Memory for intranode communications

MPI Shared Memory Functionalities

- Joint use of:
  (1) `MPI_Comm_split_type` with `MPI_COMM_TYPE_SHARED` split value
    - → **Process isolation** on computing nodes
  (2) `MPI_Win_allocate_shared`
- Result: **an explicit two-level hierarchy** in the application
  - No MPI overhead for intranode transfers
    - → **The MPI stack is bypassed!**
    - → **Use load/store operations instead of send/recv**
  - Performance discrepancies within a node not leveraged
    - → Use of an other programming model?

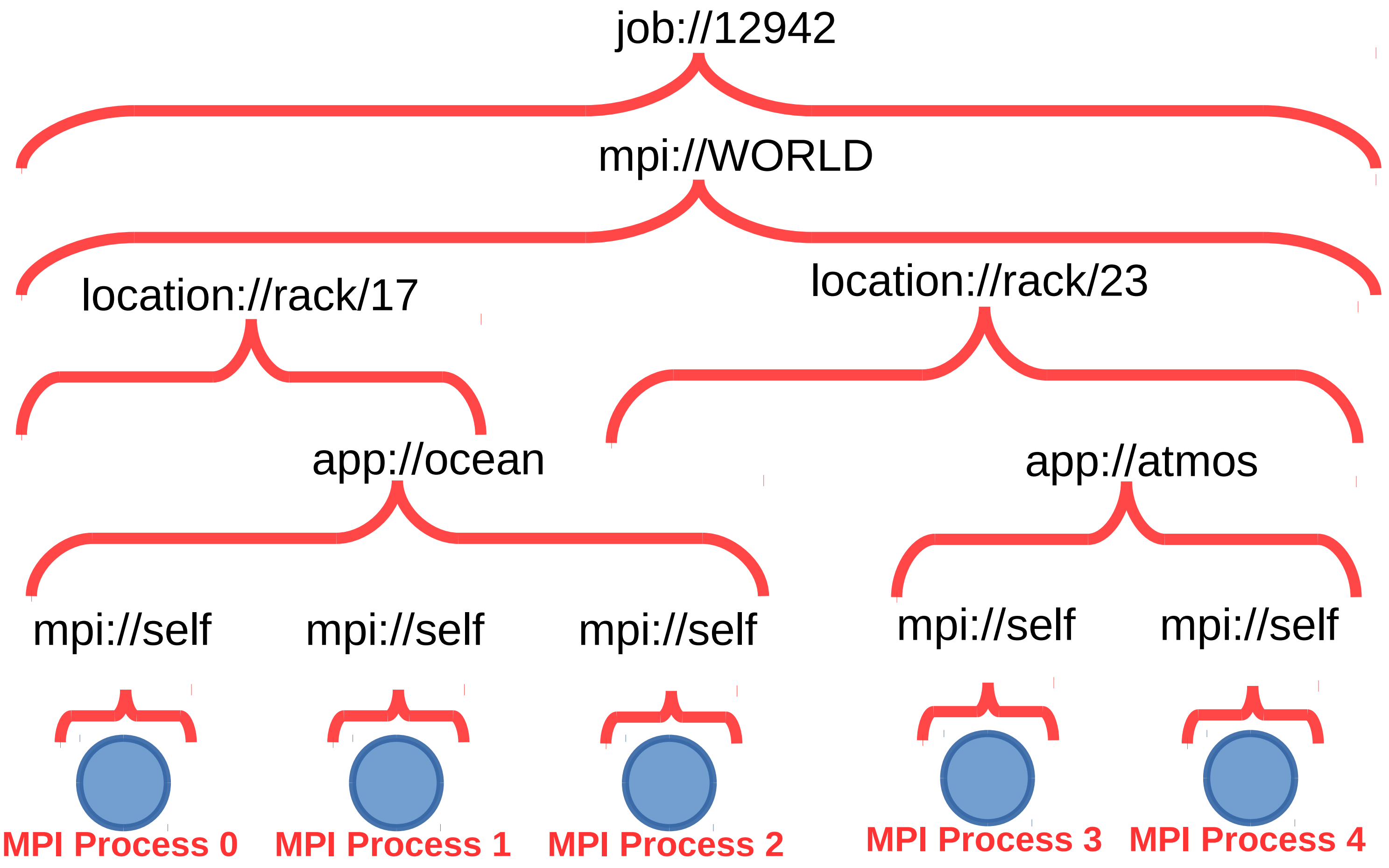
New features in the MPI Standard 4.0

- ~~Cartesian topologies optimized for hierarchical hardware~~
  - ~~Very few virtual topologies routines are optimized~~
  - Not passed in votes yet!
- MPI Sessions
- Hardware communicators

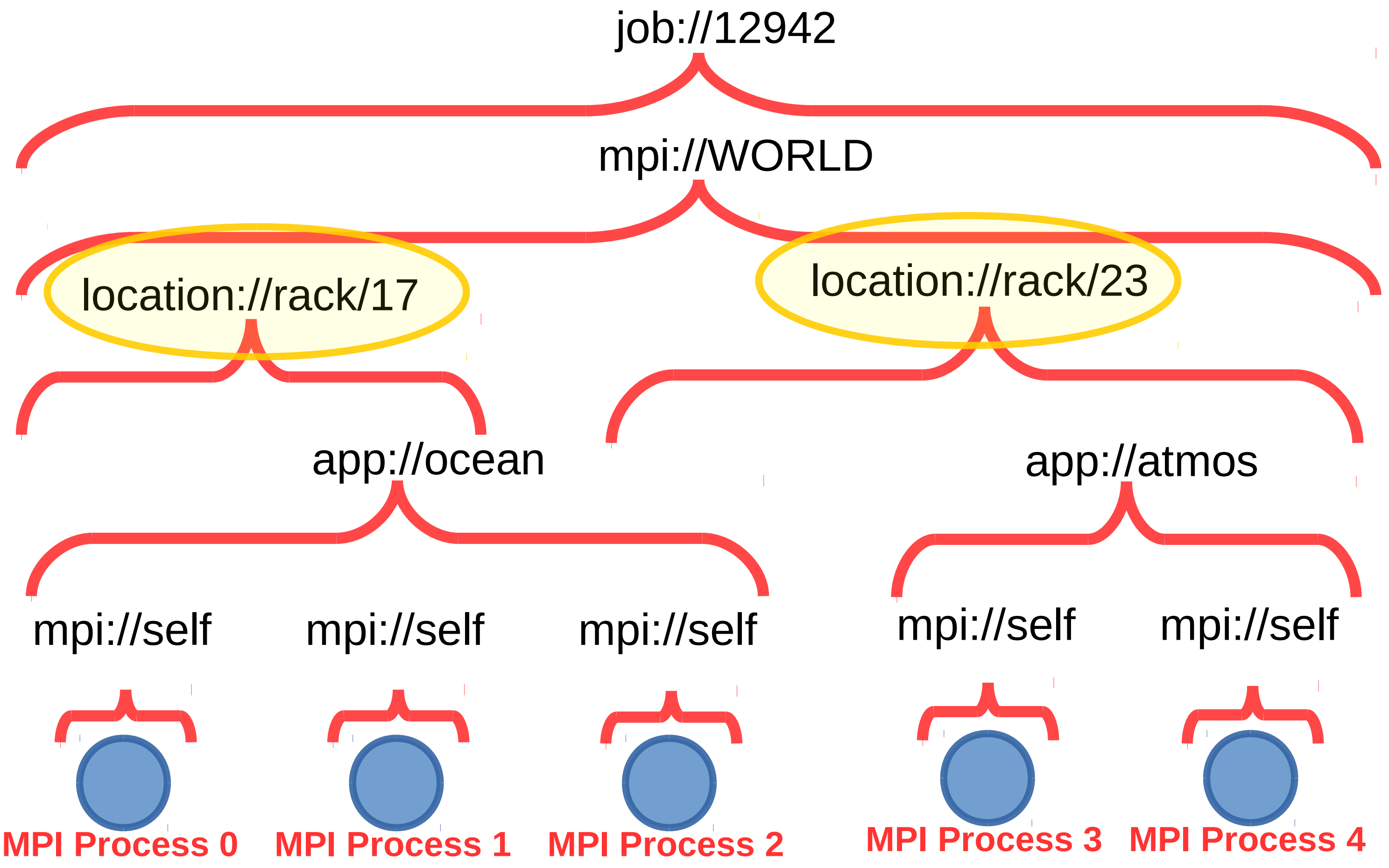
MPI Sessions

- Original goals
  - Solve composability issues
  - Solve multi init/finalize situations
- Principles
  - Process isolation in “sessions”
    - ➔ **No communications between sessions**
  - Based on **process sets**: psets → groups → communicators
    - Lighter objects than groups (and communicators)
  - Legacy “World Model” to ease the transition to “Sessions Model”
- Possibility to give “names” to sessions: URIs
  - Could extend `MPI_Get_processor_name` for hardware resources
  - **Naming scheme is problematic**

MPI Sessions Example: URIs



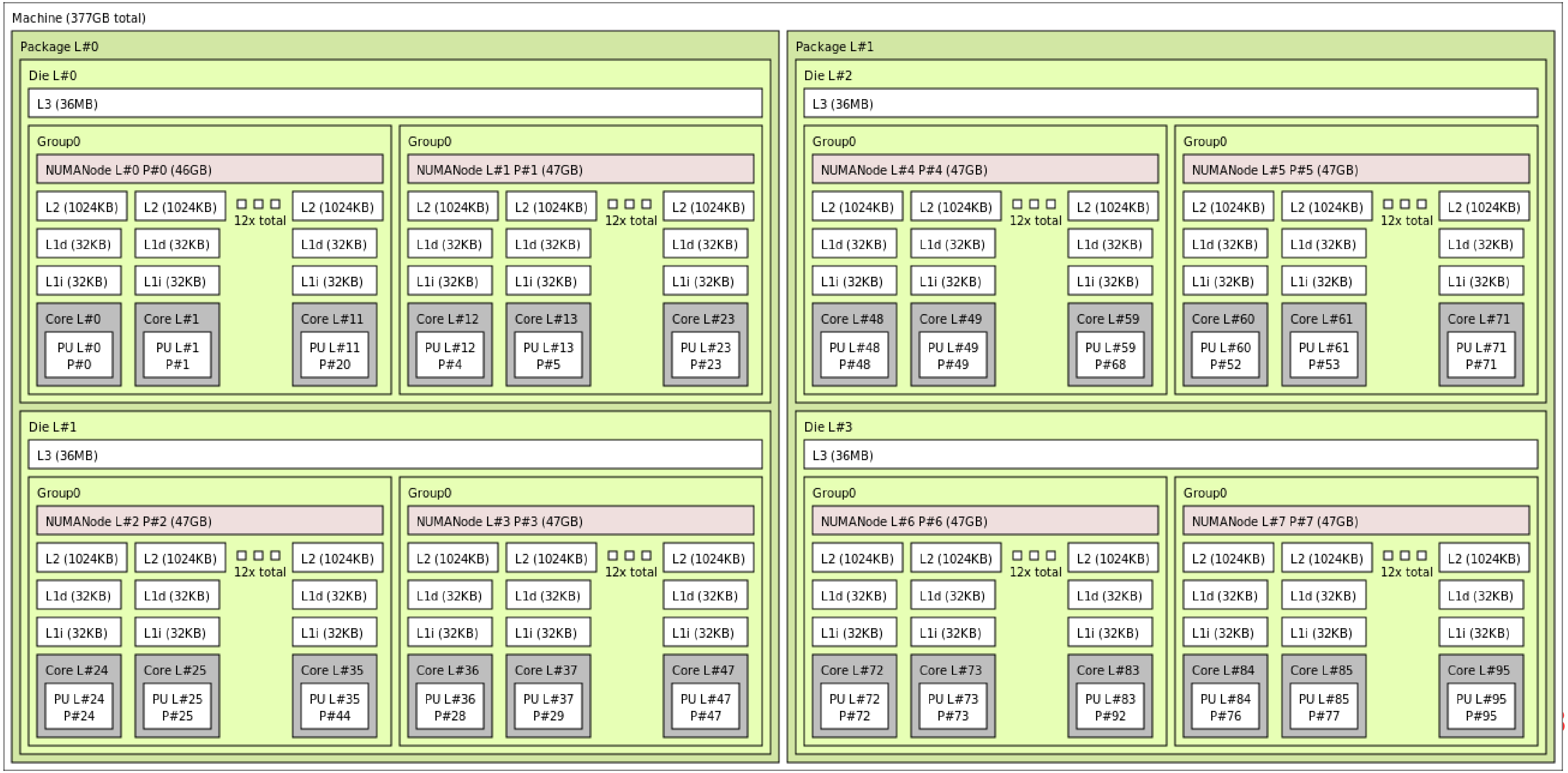
MPI Sessions Example: URIs



Hardware Communicators Purpose

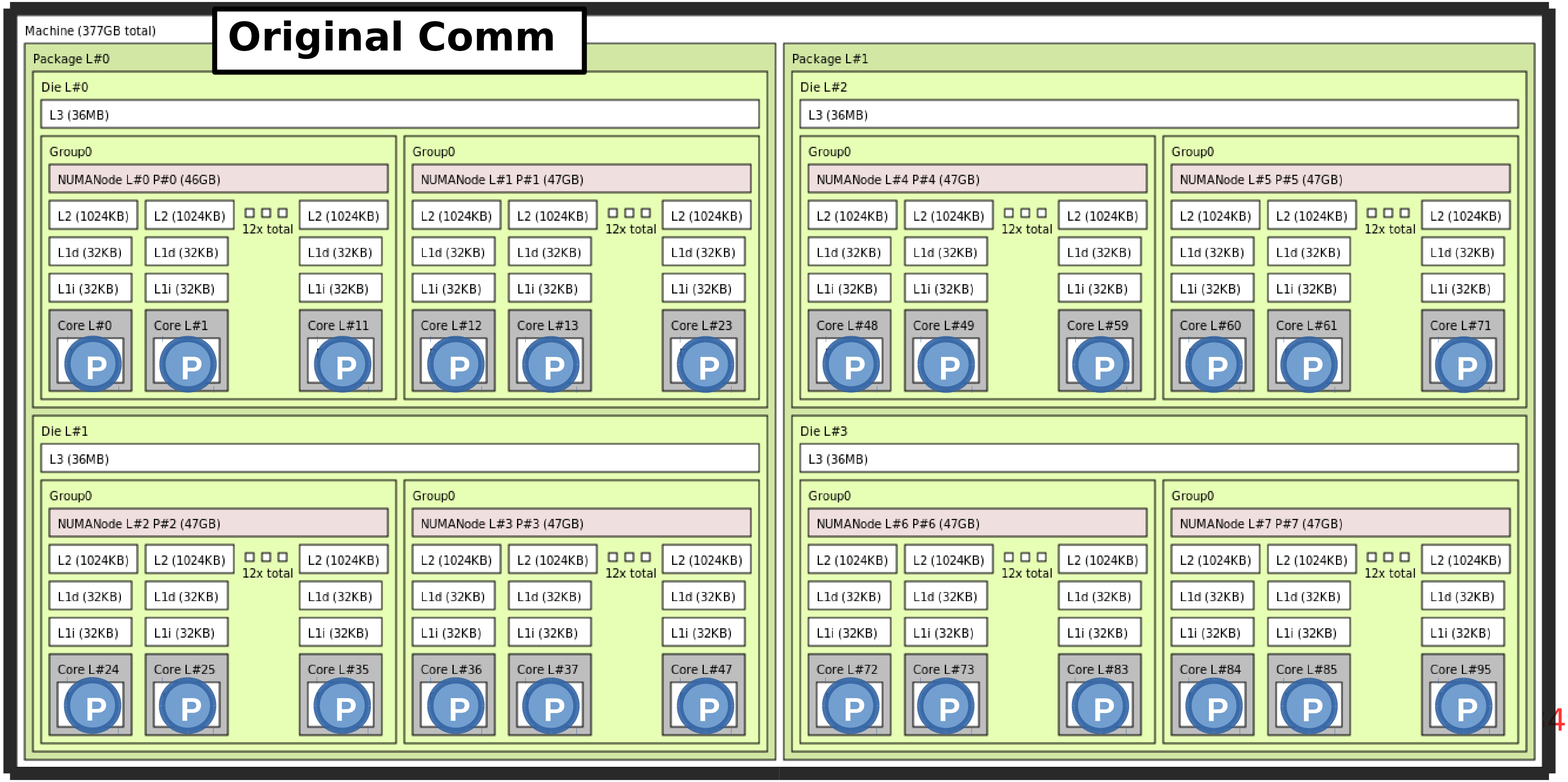
→ **Create communicators that convey locality/the sharing of resources between MPI processes**

Hardware Communicators Example

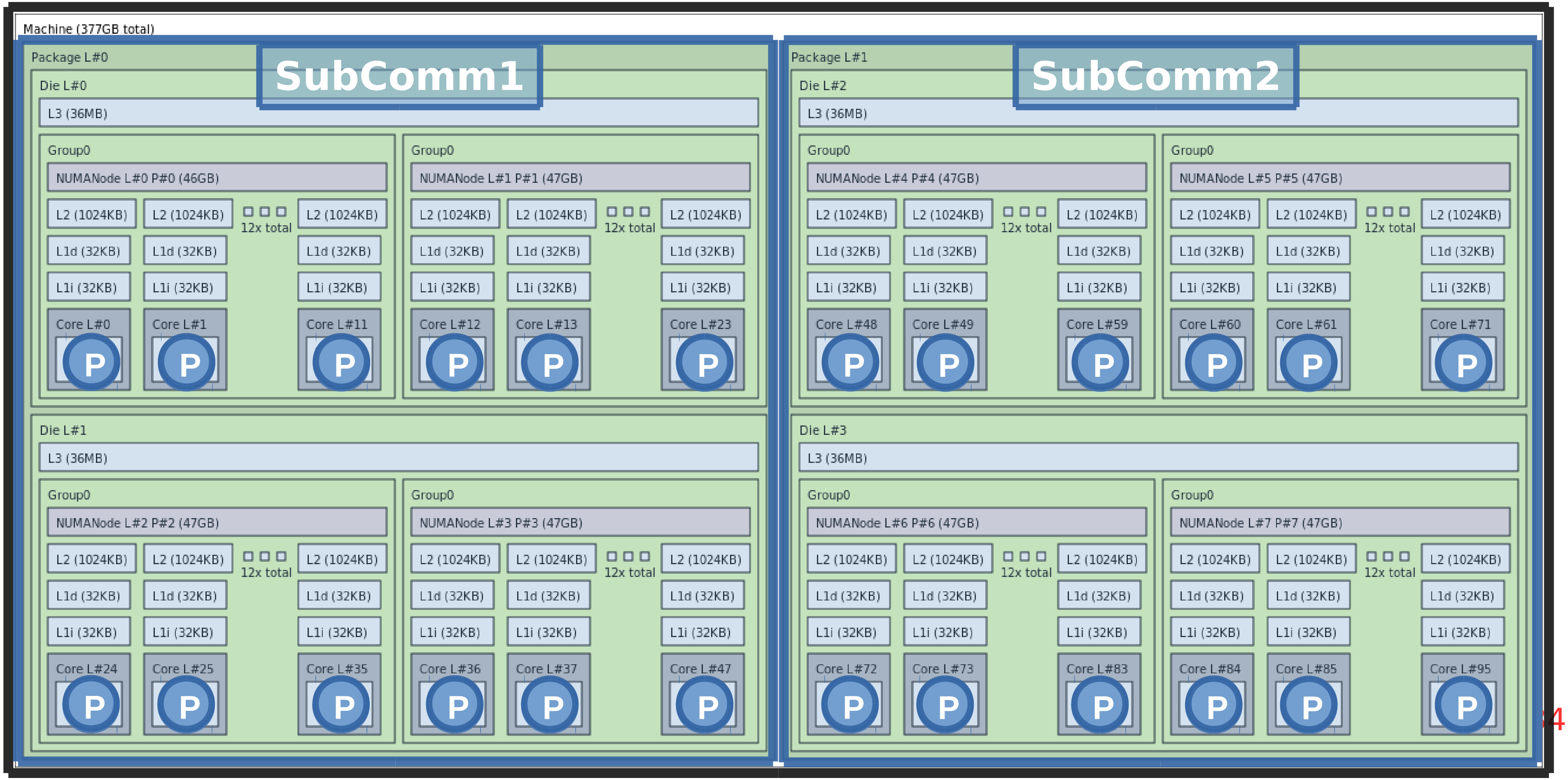


Hardware Communicators Example

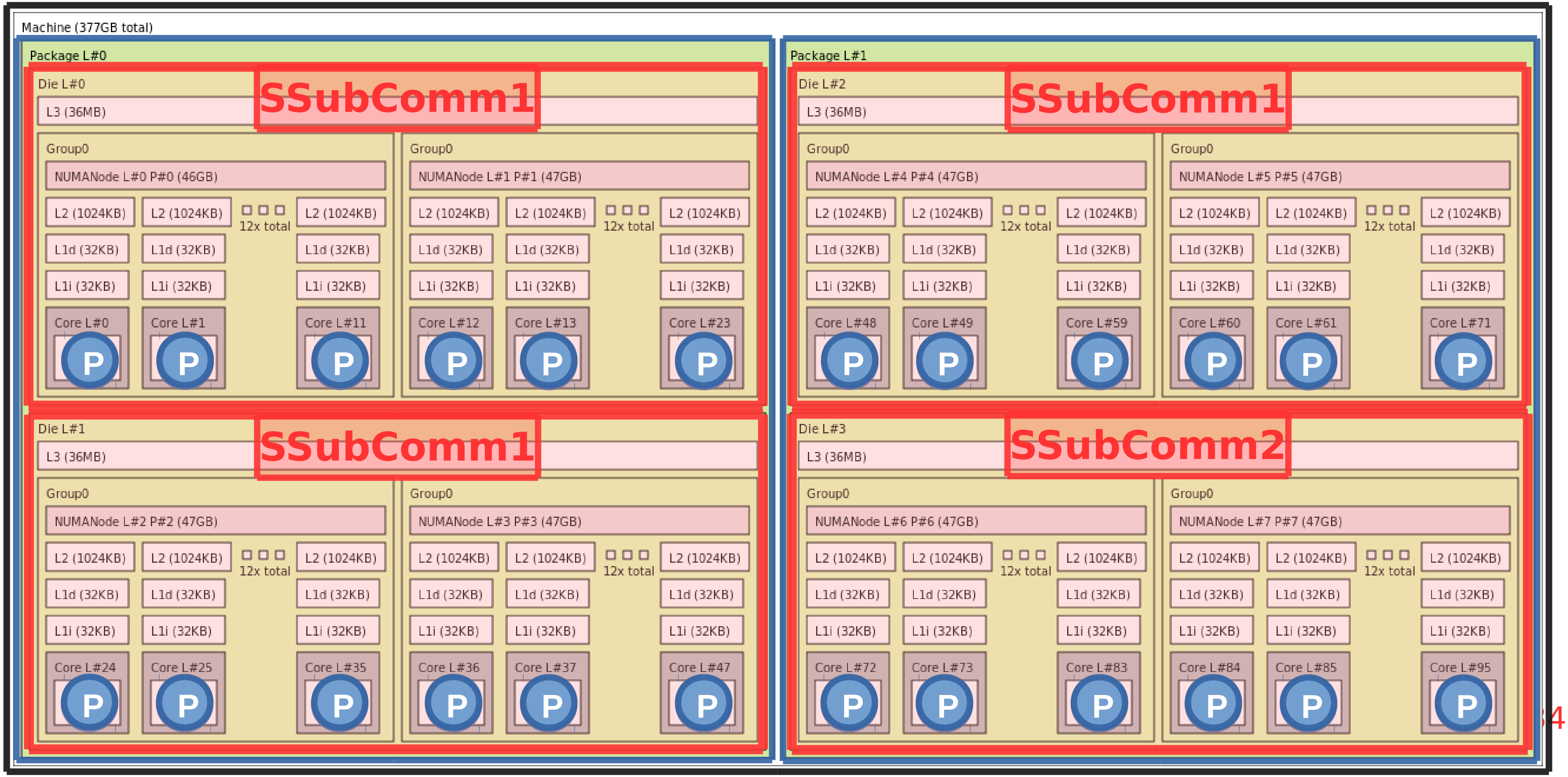
Hardware Communicators Example



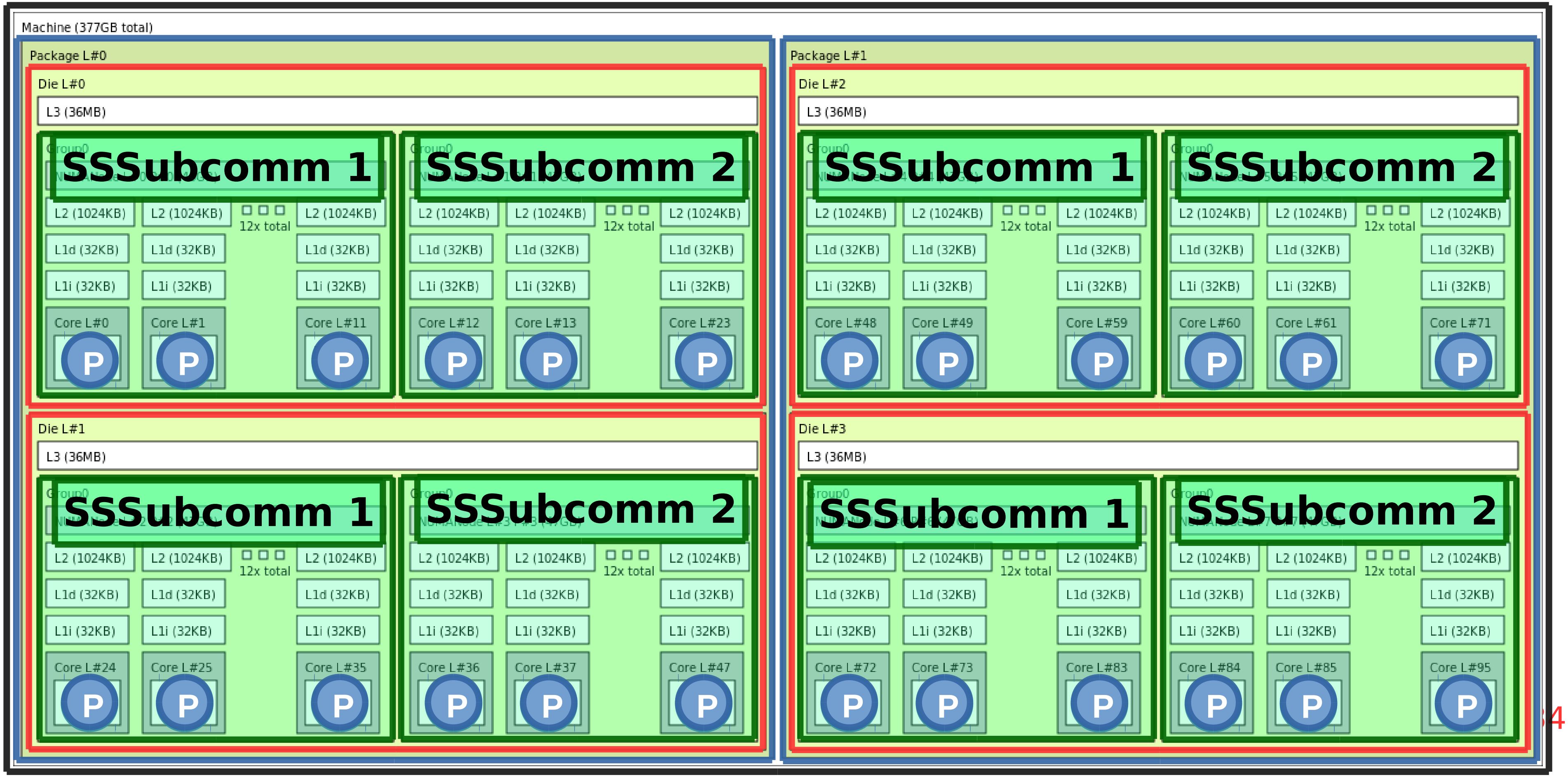
Hardware Communicators Example



Hardware Communicators Example



Hardware Communicators Example



Hardware Communicators Example

Hardware Communicators Proposal

- **Extension of an already existing** communicator creation operation:

```
MPI_Comm_split_type(MPI_Comm comm     /* IN */,
                    int split_type    /* IN */,
                    int key           /* IN */,
                    MPI_Info info     /* INOUT */,
                    MPI_Comm newcomm  /* OUT */);
```

Hardware Communicators Proposal

- **Extension of an already existing** communicator creation operation:

```
MPI_Comm_split_type(MPI_Comm comm     /* IN */,
                    int split_type    /* IN */,
                    int key           /* IN */,
                    MPI_Info info     /* INOUT */,
                    MPI_Comm newcomm  /* OUT */);
```

- With:
  - Two new **`split_type`** values:
    - **`MPI_COMM_TYPE_HW_UNGUIDED`**: split at the next level in the platform
      - Valid (i.e., non `COMM_NULL`) `new` comms are strict subsets of the old comm
    - **`MPI_COMM_TYPE_HW_GUIDED`**: split for a specified resource type
  - One new info key: `mpi_hw_resource_type`
    - To constrain the splitting operation (guided mode)
    - To query the type of resource represented by newcomm (unguided mode)

Hardware Communicators Proposal

- **Extension of an already existing** communicator creation operation:

```
MPI_Comm_split_type(MPI_Comm comm     /* IN */,
                    int split_type    /* IN */,
                    int key           /* IN */,
                    MPI_Info info     /* INOUT */,
                    MPI_Comm newcomm  /* OUT */);
```

- With:
  - Two new `split_type` values:
    - **`MPI_COMM_TYPE_HW_UNGUIDED`**: split at the next level in the platform
      - Valid (i.e., non `COMM_NULL`) `new` comms are strict subsets of the old comm
    - **`MPI_COMM_TYPE_HW_GUIDED`**: split for a specified resource type
  - One new info key: **`mpi_hw_resource_type`**
    - To constrain the splitting operation (guided mode)
    - To query the type of resource represented by `newcomm` (unguided mode)

Unguided Mode Example

```
#define MAX_NUM_LEVELS 32

MPI_Comm hwcomm[MAX_NUM_LEVELS];
int rank, level_num = 0;

hwcomm[level_num] = MPI_COMM_WORLD;

while((hwcomm[level_num] != MPI_COMM_NULL) &&
      (level_num < MAX_NUM_LEVELS-1))
{
  MPI_Comm_rank(hwcomm[level_num],&rank);
  MPI_Comm_split_type(hwcomm[level_num],
                      MPI_COMM_TYPE_HW_UNGUIDED,
                      rank,
                      MPI_INFO_NULL,
                      &hwcomm[level_num+1]);
  level_num++;
}
```

Splitting operation

Recursive Splitting of MPI_COMM_WORLD

Guided Mode Example

Splitting `MPI_COMM_WORLD` into NUMANode subcommunicators:

```
MPI_Info info;
int rank;
MPI_Comm hwcomm;

MPI_Comm_rank(MPI_COMM_WORLD,&rank);

MPI_Info_create(&info);
MPI_Info_set(info,"mpi_hw_resource_type","NUMANode");

MPI_Comm_split_type(MPI_COMM_WORLD,
                    MPI_COMM_TYPE_HW_GUIDED,
                    rank,
                    info,
                    &hwcomm);
/* Use hwcomm now */
```

Guided Mode Example

Splitting MPI_COMM_WORLD into NUMANode subcommunicators:

```
MPI_Info info;
int rank;
MPI_Comm hwcomm;

MPI_Comm_rank(MPI_COMM_WORLD,&rank);

MPI_Info_create(&info);
MPI_Info_set(info,"mpi_hw_resource_type","NUMANode");

MPI_Comm_split_type(MPI_COMM_WORLD,
                    MPI_COMM_TYPE_HW_GUIDED,
                    rank,
                    info,
                    &hwcomm);
/* Use hwcomm now */
```

Info object set-up

Guided Mode Example

Splitting `MPI_COMM_WORLD` into NUMANode subcommunicators:

Implementation dependent name

Info object set-up

```
MPI_Info info;
int rank;
MPI_Comm hwcomm;

MPI_Comm_rank(MPI_COMM_WORLD,&rank);

MPI_Info_create(&info);
MPI_Info_set(info,"mpi_hw_resource_type","NUMANode");

MPI_Comm_split_type(MPI_COMM_WORLD,
                    MPI_COMM_TYPE_HW_GUIDED,
                    rank,
                    info,
                    &hwcomm);
/* Use hwcomm now */
```

Guided Mode Example

Splitting `MPI_COMM_WORLD` into NUMANode subcommunicators:

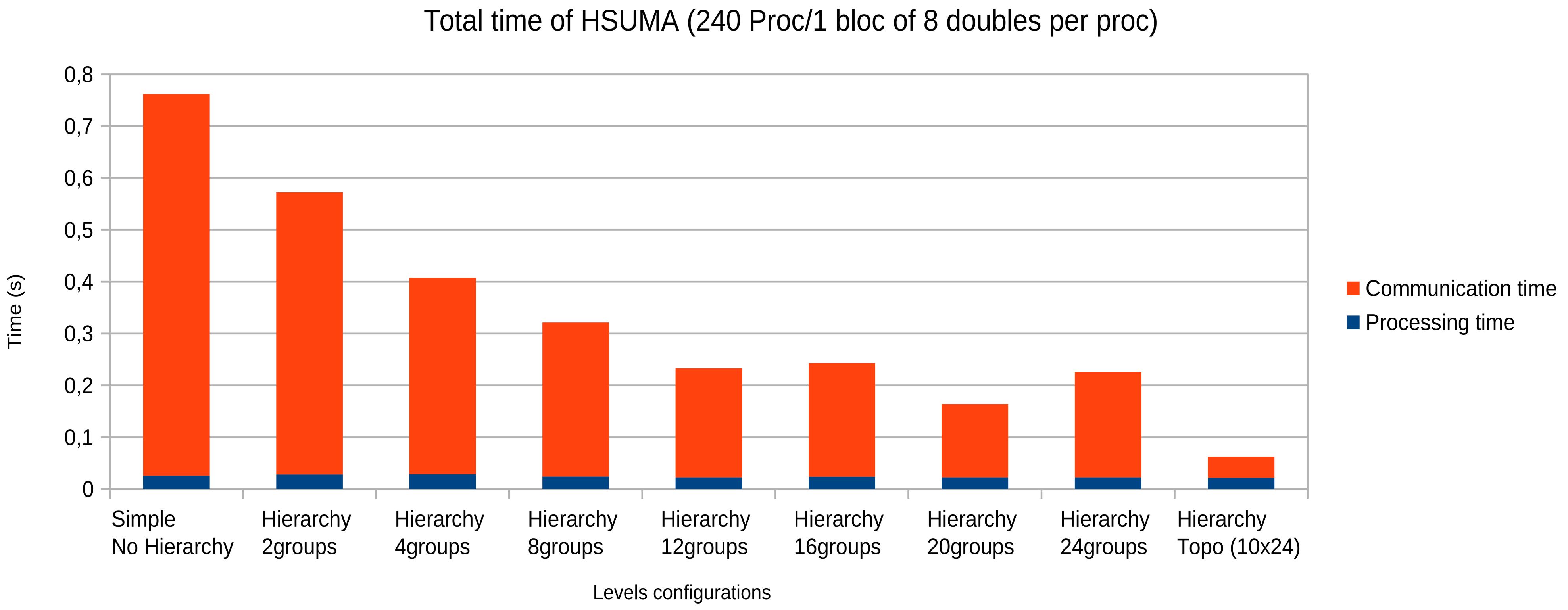
```
MPI_Info info;
int rank;
MPI_Comm hwcomm;

MPI_Comm_rank(MPI_COMM_WORLD,&rank);

MPI_Info_create(&info);
MPI_Info_set(info,"mpi_hw_resource_type","NUMANode");
MPI_Comm_split_type(MPI_COMM_WORLD,
                    MPI_COMM_TYPE_HW_GUIDED,
                    rank,
                    info,
                    &hwcomm);
/* Use hwcomm now */
```

Implementation dependent name

Info object set-up

Splitting operation

Results: Hierarchical Matrix Multiplication



MPI 4.X: Hardware Information Query Proposal

- **Resource types are implementation dependent**
  - Query function to the retrieve the types recognized by the MPI implementation:

```
MPI_Get_hw_resource_types(MPI_Info info /* OUT */);
```

- User can also check if the **resource type is supported** by the MPI implementation with:

```
MPI_Get_hw_resource_status(char *res_type /* IN */,
                           int result     /* OUT */);
```

On the Standardization Front...

- Continue current efforts in MPI for 4.X
- **Interact** more with other HPC communities (e.g., OpenMP)
- **Process Mapping/Binding is a mess!**
  - Discussions started in the Hardware Topologies Working Group of the MPI Forum
  - CEA initiative to set up a **dedicated working group**, outside of the MPI Forum
    - Target: the whole HPC ecosystem, not just MPI libraries
    - Goal: identify **necessary components** and their **interactions** in the HPC ecosystem

That's all Folks!