MPI Forum
2017/03/01
Portland, OR

Locality and Physical Topology Support in MPI

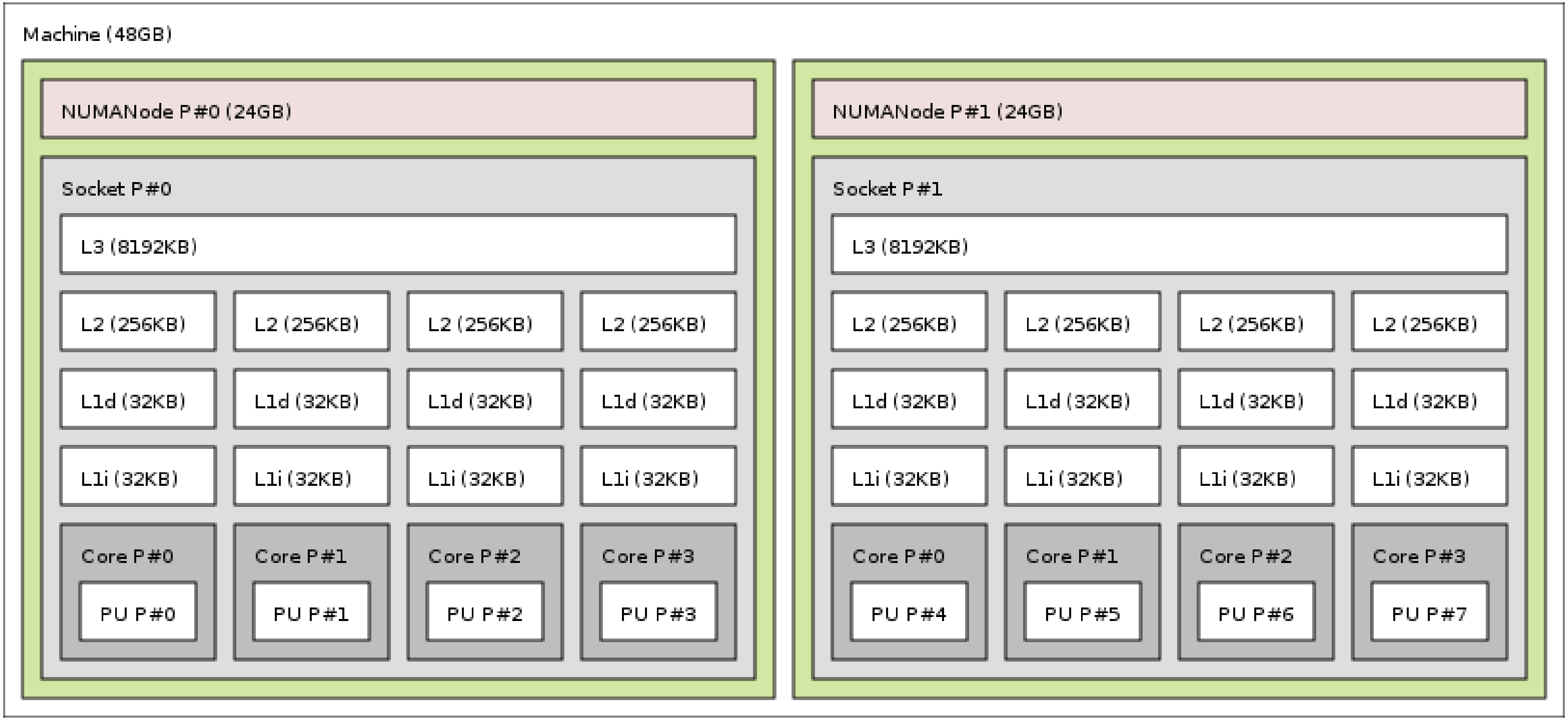
Guillaume Mercier – TADaaM Team – Inria Bordeaux Sud-Ouest

Machines are increasingly complex

- Multiple processors
- Multicore processors
- Simultaneous multithreading
- Shared caches
- NUMA effects

- We cannot expect users to understand all this ...
- … But we can help them to take advantage of this complexity
  - Often seen as a hierarchy of resources

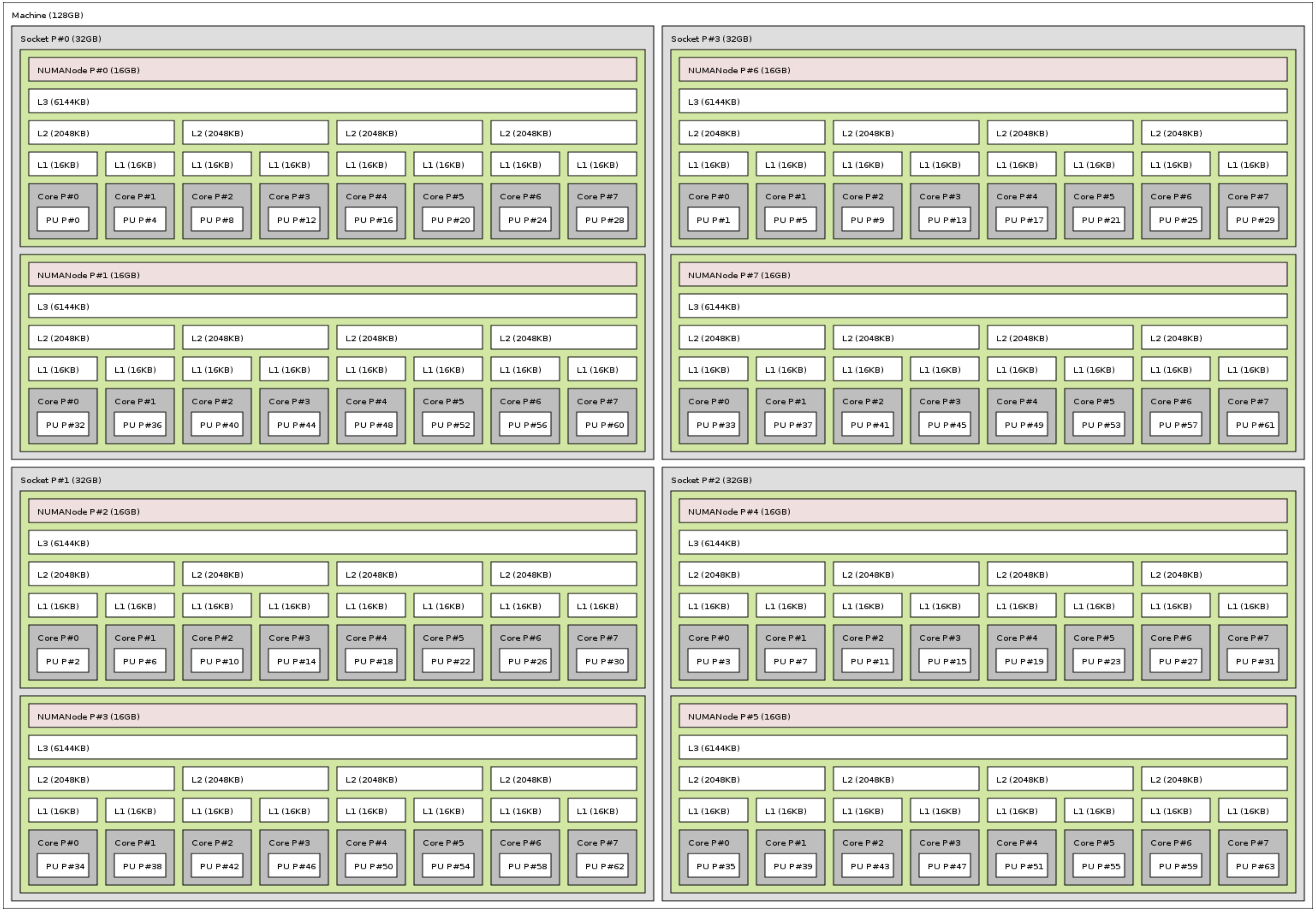
A random dual-processor quad-core machine



Machine (48GB)

- NUMANode P#0 (24GB)
  - Socket P#0
    - L3 (8192KB)
    - L2 (256KB) | L2 (256KB) | L2 (256KB) | L2 (256KB)
    - L1d (32KB) | L1d (32KB) | L1d (32KB) | L1d (32KB)
    - L1i (32KB) | L1i (32KB) | L1i (32KB) | L1i (32KB)
    - Core P#0: PU P#0 | Core P#1: PU P#1 | Core P#2: PU P#2 | Core P#3: PU P#3
- NUMANode P#1 (24GB)
  - Socket P#1
    - L3 (8192KB)
    - L2 (256KB) | L2 (256KB) | L2 (256KB) | L2 (256KB)
    - L1d (32KB) | L1d (32KB) | L1d (32KB) | L1d (32KB)
    - L1i (32KB) | L1i (32KB) | L1i (32KB) | L1i (32KB)
    - Core P#0: PU P#4 | Core P#1: PU P#5 | Core P#2: PU P#6 | Core P#3: PU P#7

A random larger machine



Machine (128GB)

Socket P#0 (32GB)

NUMANode P#0 (16GB)

L3 (6144KB)

L2 (2048KB) L2 (2048KB) L2 (2048KB) L2 (2048KB)

L1 (16KB) L1 (16KB) L1 (16KB) L1 (16KB) L1 (16KB) L1 (16KB) L1 (16KB) L1 (16KB)

Core P#0 PU P#0 | Core P#1 PU P#4 | Core P#2 PU P#8 | Core P#3 PU P#12 | Core P#4 PU P#16 | Core P#5 PU P#20 | Core P#6 PU P#24 | Core P#7 PU P#28

NUMANode P#1 (16GB)

L3 (6144KB)

L2 (2048KB) L2 (2048KB) L2 (2048KB) L2 (2048KB)

L1 (16KB) L1 (16KB) L1 (16KB) L1 (16KB) L1 (16KB) L1 (16KB) L1 (16KB) L1 (16KB)

Core P#0 PU P#32 | Core P#1 PU P#36 | Core P#2 PU P#40 | Core P#3 PU P#44 | Core P#4 PU P#48 | Core P#5 PU P#52 | Core P#6 PU P#56 | Core P#7 PU P#60

Socket P#3 (32GB)

NUMANode P#6 (16GB)

L3 (6144KB)

L2 (2048KB) L2 (2048KB) L2 (2048KB) L2 (2048KB)

L1 (16KB) L1 (16KB) L1 (16KB) L1 (16KB) L1 (16KB) L1 (16KB) L1 (16KB) L1 (16KB)

Core P#0 PU P#1 | Core P#1 PU P#5 | Core P#2 PU P#9 | Core P#3 PU P#13 | Core P#4 PU P#17 | Core P#5 PU P#21 | Core P#6 PU P#25 | Core P#7 PU P#29

NUMANode P#7 (16GB)

L3 (6144KB)

L2 (2048KB) L2 (2048KB) L2 (2048KB) L2 (2048KB)

L1 (16KB) L1 (16KB) L1 (16KB) L1 (16KB) L1 (16KB) L1 (16KB) L1 (16KB) L1 (16KB)

Core P#0 PU P#33 | Core P#1 PU P#37 | Core P#2 PU P#41 | Core P#3 PU P#45 | Core P#4 PU P#49 | Core P#5 PU P#53 | Core P#6 PU P#57 | Core P#7 PU P#61

Socket P#1 (32GB)

NUMANode P#2 (16GB)

L3 (6144KB)

L2 (2048KB) L2 (2048KB) L2 (2048KB) L2 (2048KB)

L1 (16KB) L1 (16KB) L1 (16KB) L1 (16KB) L1 (16KB) L1 (16KB) L1 (16KB) L1 (16KB)

Core P#0 PU P#2 | Core P#1 PU P#6 | Core P#2 PU P#10 | Core P#3 PU P#14 | Core P#4 PU P#18 | Core P#5 PU P#22 | Core P#6 PU P#26 | Core P#7 PU P#30

NUMANode P#3 (16GB)

L3 (6144KB)

L2 (2048KB) L2 (2048KB) L2 (2048KB) L2 (2048KB)

L1 (16KB) L1 (16KB) L1 (16KB) L1 (16KB) L1 (16KB) L1 (16KB) L1 (16KB) L1 (16KB)

Core P#0 PU P#34 | Core P#1 PU P#38 | Core P#2 PU P#42 | Core P#3 PU P#46 | Core P#4 PU P#50 | Core P#5 PU P#54 | Core P#6 PU P#58 | Core P#7 PU P#62

Socket P#2 (32GB)

NUMANode P#4 (16GB)

L3 (6144KB)

L2 (2048KB) L2 (2048KB) L2 (2048KB) L2 (2048KB)

L1 (16KB) L1 (16KB) L1 (16KB) L1 (16KB) L1 (16KB) L1 (16KB) L1 (16KB) L1 (16KB)

Core P#0 PU P#3 | Core P#1 PU P#7 | Core P#2 PU P#11 | Core P#3 PU P#15 | Core P#4 PU P#19 | Core P#5 PU P#23 | Core P#6 PU P#27 | Core P#7 PU P#31

NUMANode P#5 (16GB)

L3 (6144KB)

L2 (2048KB) L2 (2048KB) L2 (2048KB) L2 (2048KB)

L1 (16KB) L1 (16KB) L1 (16KB) L1 (16KB) L1 (16KB) L1 (16KB) L1 (16KB) L1 (16KB)

Core P#0 PU P#35 | Core P#1 PU P#39 | Core P#2 PU P#43 | Core P#3 PU P#47 | Core P#4 PU P#51 | Core P#5 PU P#55 | Core P#6 PU P#59 | Core P#7 PU P#63

Current support of HW topology in MPI

- MPI is hardware-agnostic
  - And should remain so (I.e no assumptions about the HW)
  - Doesn’t prevent from accessing the HW topology from MPI directly
- Virtual topologies
  - Machine-independent
  - Virtual to physical mapping “outside of the scope of MPI”
- No standard behaviour (implementation-dependent)
  - Reordering
    - MPI_Dist_graph_create, MPI_Graph_map, etc.
    - Side-effect of the function, implementation-dependent
  - Process Managers mapping and binding options
- MPI Sessions?

Motivation

- Application developpers need abstract features to:
  - Deal with hardware characteristics (Caches, Interconnect, Cores, NUMA nodes, etc.)
  - Deal with low level tools (Hwloc, Lib_NUMA, Etc.)
- Expected performance improvements
  - Improved locality
  - Improved communication performance

Basic Idea

- Use available abstractions in MPI: communicators
  - Well-known concept/object in MPI programming
  - “Natural” fit for our purpose:
    - Group MPI processes in communicators for each meaningful level in the hierarchy of the physical topology
    - Usable for collective communications
- Rather expand than add new features
  - Leverage existing mechanisms and abstractions

Communicator creation functions

- MPI_Comm_create
- MPI_Comm_dup and friends
  - idup
  - with_info
- MPI_Comm_split
- MPI_Comm_split_type
  - MPI_COMM_TYPE_SHARED
  - MPI_COMM_TYPE_ADDRESS_SPACE (issue #31)
  - Implementations can also define their own values

Proposal

- Add a new predefined value for the split_type arg
  - e.g MPI_COMM_TYPE_PHYSICAL_TOPOLOGY
  - Or any suitable (meaningful) name
- Property of the newly created communicator(s)
  - All newly created communicators should be a strict subset of the input communicator
    - MPI_Comm_compare(oldcomm,newcomm) yields MPI_UNEQUAL
    - **To ensure we don't create several useless communicators in case of physical levels redundancy**

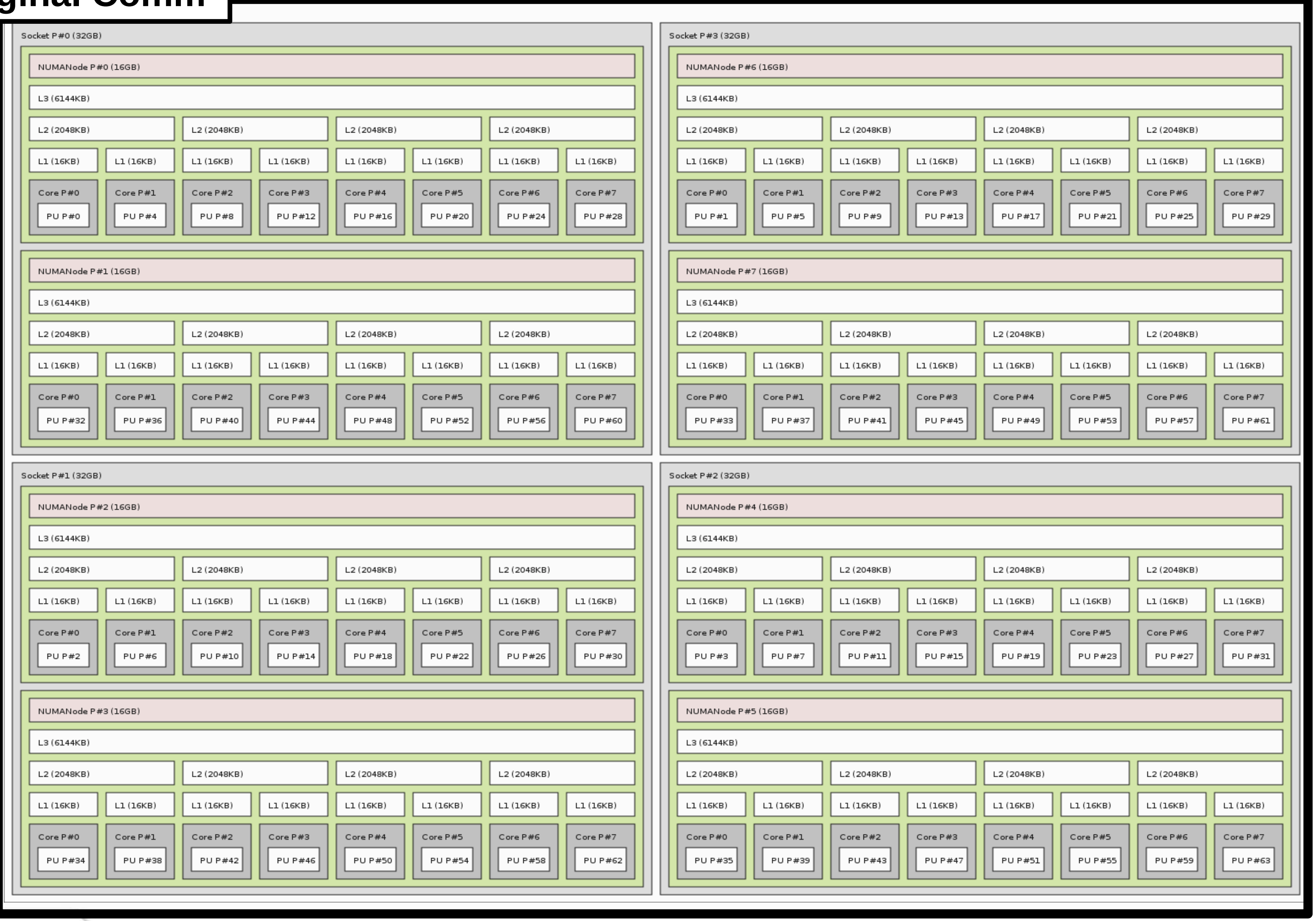
Practical Use

- Recursively split an input communicator until the bottom of the hierarchy is reached (MPI_COMM_NULL)
  - **Independent of the hierarchy depth**
  - **No (fixed) names for communicators**
- Example:

```
#define NLEVELS 16
int in_rank;
MPI_Comm out_comm[NLEVELS];
in_comm = MPI_COMM_WORLD;
idx = 0;
while(in_comm != MPI_COMM_NULL){
    MPI_Comm_rank(in_comm,&in_rank);
    MPI_Comm_split_type(in_comm,
                                     MPI_COMM_TYPE_PHYSICAL_TOPOLOGY,
                                     in_rank,MPI_INFO_NULL,
                                     &out_comm[idx]);
   in_comm = out_comm[idx];
   assert(++idx < NLEVELS);
 }
```

Example

Original Comm



Socket P#0 (32GB)

NUMANode P#0 (16GB)
L3 (6144KB)
L2 (2048KB) L2 (2048KB) L2 (2048KB) L2 (2048KB)
L1 (16KB) L1 (16KB) L1 (16KB) L1 (16KB) L1 (16KB) L1 (16KB) L1 (16KB) L1 (16KB)
Core P#0 Core P#1 Core P#2 Core P#3 Core P#4 Core P#5 Core P#6 Core P#7
PU P#0 PU P#4 PU P#8 PU P#12 PU P#16 PU P#20 PU P#24 PU P#28

NUMANode P#1 (16GB)
L3 (6144KB)
L2 (2048KB) L2 (2048KB) L2 (2048KB) L2 (2048KB)
L1 (16KB) L1 (16KB) L1 (16KB) L1 (16KB) L1 (16KB) L1 (16KB) L1 (16KB) L1 (16KB)
Core P#0 Core P#1 Core P#2 Core P#3 Core P#4 Core P#5 Core P#6 Core P#7
PU P#32 PU P#36 PU P#40 PU P#44 PU P#48 PU P#52 PU P#56 PU P#60

Socket P#3 (32GB)

NUMANode P#6 (16GB)
L3 (6144KB)
L2 (2048KB) L2 (2048KB) L2 (2048KB) L2 (2048KB)
L1 (16KB) L1 (16KB) L1 (16KB) L1 (16KB) L1 (16KB) L1 (16KB) L1 (16KB) L1 (16KB)
Core P#0 Core P#1 Core P#2 Core P#3 Core P#4 Core P#5 Core P#6 Core P#7
PU P#1 PU P#5 PU P#9 PU P#13 PU P#17 PU P#21 PU P#25 PU P#29

NUMANode P#7 (16GB)
L3 (6144KB)
L2 (2048KB) L2 (2048KB) L2 (2048KB) L2 (2048KB)
L1 (16KB) L1 (16KB) L1 (16KB) L1 (16KB) L1 (16KB) L1 (16KB) L1 (16KB) L1 (16KB)
Core P#0 Core P#1 Core P#2 Core P#3 Core P#4 Core P#5 Core P#6 Core P#7
PU P#33 PU P#37 PU P#41 PU P#45 PU P#49 PU P#53 PU P#57 PU P#61

Socket P#1 (32GB)

NUMANode P#2 (16GB)
L3 (6144KB)
L2 (2048KB) L2 (2048KB) L2 (2048KB) L2 (2048KB)
L1 (16KB) L1 (16KB) L1 (16KB) L1 (16KB) L1 (16KB) L1 (16KB) L1 (16KB) L1 (16KB)
Core P#0 Core P#1 Core P#2 Core P#3 Core P#4 Core P#5 Core P#6 Core P#7
PU P#2 PU P#6 PU P#10 PU P#14 PU P#18 PU P#22 PU P#26 PU P#30

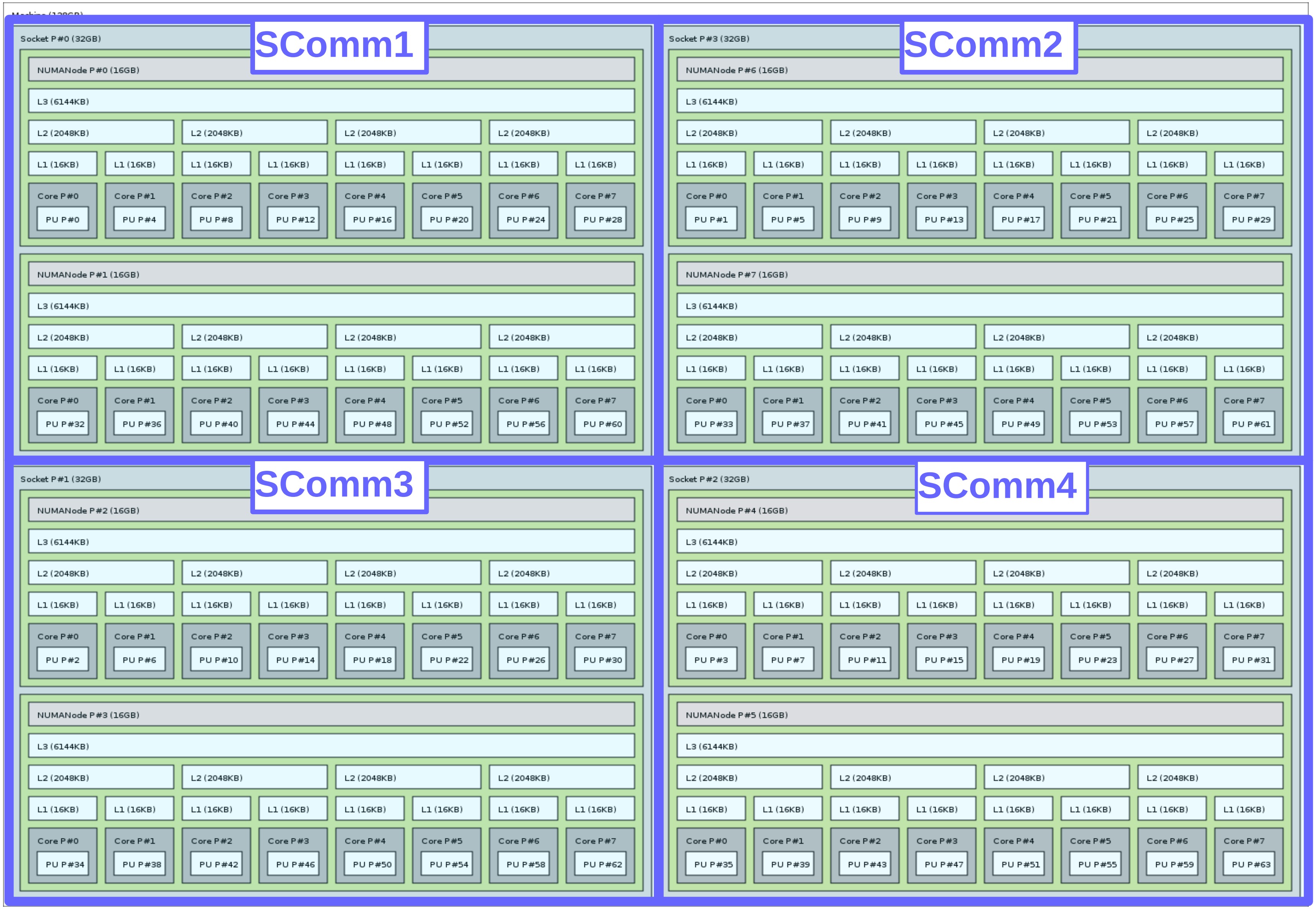
NUMANode P#3 (16GB)
L3 (6144KB)
L2 (2048KB) L2 (2048KB) L2 (2048KB) L2 (2048KB)
L1 (16KB) L1 (16KB) L1 (16KB) L1 (16KB) L1 (16KB) L1 (16KB) L1 (16KB) L1 (16KB)
Core P#0 Core P#1 Core P#2 Core P#3 Core P#4 Core P#5 Core P#6 Core P#7
PU P#34 PU P#38 PU P#42 PU P#46 PU P#50 PU P#54 PU P#58 PU P#62

Socket P#2 (32GB)

NUMANode P#4 (16GB)
L3 (6144KB)
L2 (2048KB) L2 (2048KB) L2 (2048KB) L2 (2048KB)
L1 (16KB) L1 (16KB) L1 (16KB) L1 (16KB) L1 (16KB) L1 (16KB) L1 (16KB) L1 (16KB)
Core P#0 Core P#1 Core P#2 Core P#3 Core P#4 Core P#5 Core P#6 Core P#7
PU P#3 PU P#7 PU P#11 PU P#15 PU P#19 PU P#23 PU P#27 PU P#31

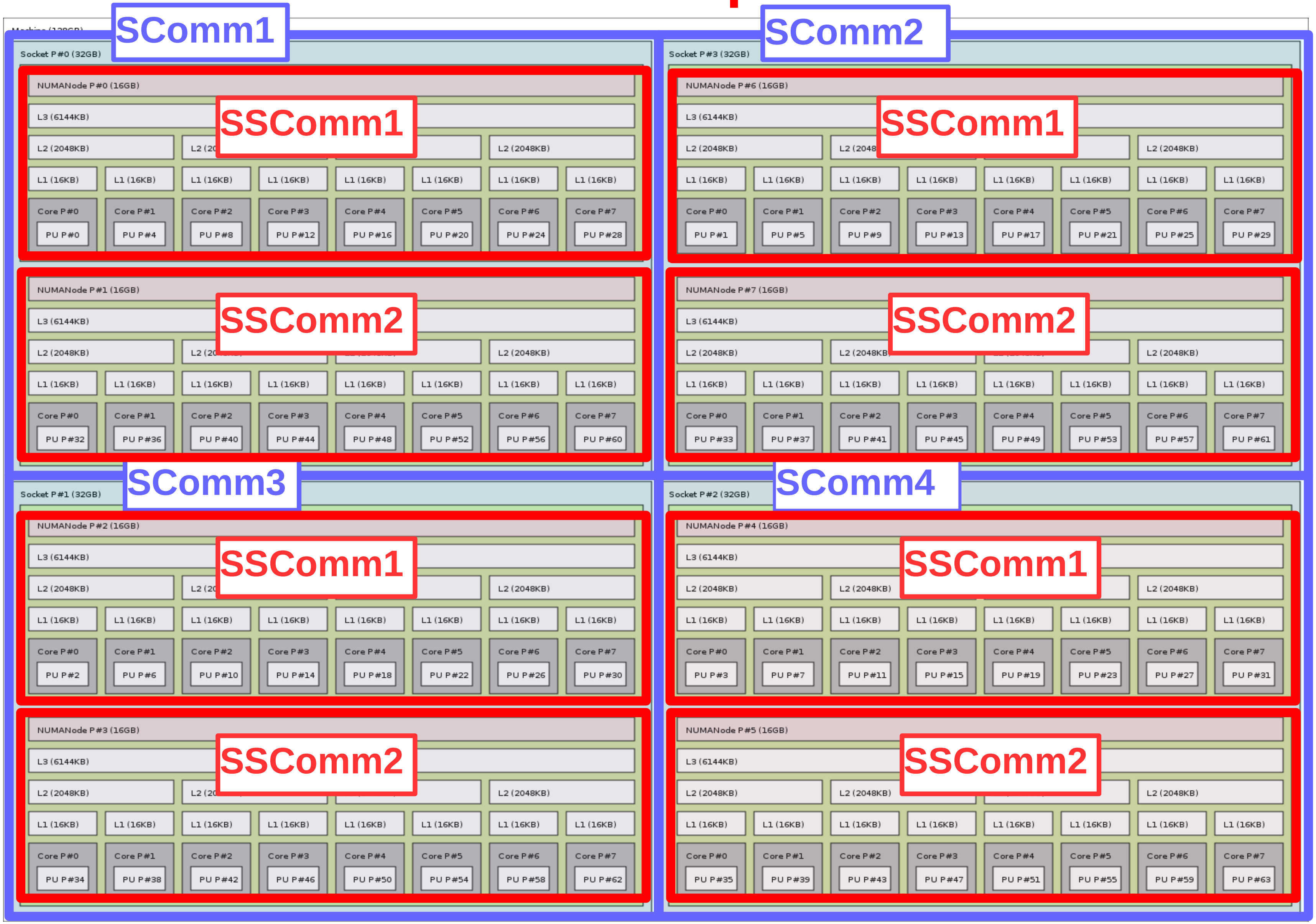
NUMANode P#5 (16GB)
L3 (6144KB)
L2 (2048KB) L2 (2048KB) L2 (2048KB) L2 (2048KB)
L1 (16KB) L1 (16KB) L1 (16KB) L1 (16KB) L1 (16KB) L1 (16KB) L1 (16KB) L1 (16KB)
Core P#0 Core P#1 Core P#2 Core P#3 Core P#4 Core P#5 Core P#6 Core P#7
PU P#35 PU P#39 PU P#43 PU P#47 PU P#51 PU P#55 PU P#59 PU P#63

Example



Machine (128GB)
SComm1
Socket P#0 (32GB)
NUMANode P#0 (16GB)
L3 (6144KB)
L2 (2048KB) L2 (2048KB) L2 (2048KB) L2 (2048KB)
L1 (16KB) L1 (16KB) L1 (16KB) L1 (16KB) L1 (16KB) L1 (16KB) L1 (16KB) L1 (16KB)
Core P#0 Core P#1 Core P#2 Core P#3 Core P#4 Core P#5 Core P#6 Core P#7
PU P#0 PU P#4 PU P#8 PU P#12 PU P#16 PU P#20 PU P#24 PU P#28
NUMANode P#1 (16GB)
L3 (6144KB)
L2 (2048KB) L2 (2048KB) L2 (2048KB) L2 (2048KB)
L1 (16KB) L1 (16KB) L1 (16KB) L1 (16KB) L1 (16KB) L1 (16KB) L1 (16KB) L1 (16KB)
Core P#0 Core P#1 Core P#2 Core P#3 Core P#4 Core P#5 Core P#6 Core P#7
PU P#32 PU P#36 PU P#40 PU P#44 PU P#48 PU P#52 PU P#56 PU P#60
SComm2
Socket P#3 (32GB)
NUMANode P#6 (16GB)
L3 (6144KB)
L2 (2048KB) L2 (2048KB) L2 (2048KB) L2 (2048KB)
L1 (16KB) L1 (16KB) L1 (16KB) L1 (16KB) L1 (16KB) L1 (16KB) L1 (16KB) L1 (16KB)
Core P#0 Core P#1 Core P#2 Core P#3 Core P#4 Core P#5 Core P#6 Core P#7
PU P#1 PU P#5 PU P#9 PU P#13 PU P#17 PU P#21 PU P#25 PU P#29
NUMANode P#7 (16GB)
L3 (6144KB)
L2 (2048KB) L2 (2048KB) L2 (2048KB) L2 (2048KB)
L1 (16KB) L1 (16KB) L1 (16KB) L1 (16KB) L1 (16KB) L1 (16KB) L1 (16KB) L1 (16KB)
Core P#0 Core P#1 Core P#2 Core P#3 Core P#4 Core P#5 Core P#6 Core P#7
PU P#33 PU P#37 PU P#41 PU P#45 PU P#49 PU P#53 PU P#57 PU P#61
SComm3
Socket P#1 (32GB)
NUMANode P#2 (16GB)
L3 (6144KB)
L2 (2048KB) L2 (2048KB) L2 (2048KB) L2 (2048KB)
L1 (16KB) L1 (16KB) L1 (16KB) L1 (16KB) L1 (16KB) L1 (16KB) L1 (16KB) L1 (16KB)
Core P#0 Core P#1 Core P#2 Core P#3 Core P#4 Core P#5 Core P#6 Core P#7
PU P#2 PU P#6 PU P#10 PU P#14 PU P#18 PU P#22 PU P#26 PU P#30
NUMANode P#3 (16GB)
L3 (6144KB)
L2 (2048KB) L2 (2048KB) L2 (2048KB) L2 (2048KB)
L1 (16KB) L1 (16KB) L1 (16KB) L1 (16KB) L1 (16KB) L1 (16KB) L1 (16KB) L1 (16KB)
Core P#0 Core P#1 Core P#2 Core P#3 Core P#4 Core P#5 Core P#6 Core P#7
PU P#34 PU P#38 PU P#42 PU P#46 PU P#50 PU P#54 PU P#58 PU P#62
SComm4
Socket P#2 (32GB)
NUMANode P#4 (16GB)
L3 (6144KB)
L2 (2048KB) L2 (2048KB) L2 (2048KB) L2 (2048KB)
L1 (16KB) L1 (16KB) L1 (16KB) L1 (16KB) L1 (16KB) L1 (16KB) L1 (16KB) L1 (16KB)
Core P#0 Core P#1 Core P#2 Core P#3 Core P#4 Core P#5 Core P#6 Core P#7
PU P#3 PU P#7 PU P#11 PU P#15 PU P#19 PU P#23 PU P#27 PU P#31
NUMANode P#5 (16GB)
L3 (6144KB)
L2 (2048KB) L2 (2048KB) L2 (2048KB) L2 (2048KB)
L1 (16KB) L1 (16KB) L1 (16KB) L1 (16KB) L1 (16KB) L1 (16KB) L1 (16KB) L1 (16KB)
Core P#0 Core P#1 Core P#2 Core P#3 Core P#4 Core P#5 Core P#6 Core P#7
PU P#35 PU P#39 PU P#43 PU P#47 PU P#51 PU P#55 PU P#59 PU P#63

Example



SComm1

Socket P#0 (32GB)

SSComm1

NUMANode P#0 (16GB)

L3 (6144KB)

L2 (2048KB) | L2 (20 | L2 (2048KB)

L1 (16KB) | L1 (16KB) | L1 (16KB) | L1 (16KB) | L1 (16KB) | L1 (16KB) | L1 (16KB) | L1 (16KB)

Core P#0 PU P#0 | Core P#1 PU P#4 | Core P#2 PU P#8 | Core P#3 PU P#12 | Core P#4 PU P#16 | Core P#5 PU P#20 | Core P#6 PU P#24 | Core P#7 PU P#28

SSComm2

NUMANode P#1 (16GB)

L3 (6144KB)

L2 (2048KB) | L2 (20 | L2 (2048KB)

L1 (16KB) | L1 (16KB) | L1 (16KB) | L1 (16KB) | L1 (16KB) | L1 (16KB) | L1 (16KB) | L1 (16KB)

Core P#0 PU P#32 | Core P#1 PU P#36 | Core P#2 PU P#40 | Core P#3 PU P#44 | Core P#4 PU P#48 | Core P#5 PU P#52 | Core P#6 PU P#56 | Core P#7 PU P#60

SComm2

Socket P#3 (32GB)

SSComm1

NUMANode P#6 (16GB)

L3 (6144KB)

L2 (2048KB) | L2 (2048 | L2 (2048KB)

L1 (16KB) | L1 (16KB) | L1 (16KB) | L1 (16KB) | L1 (16KB) | L1 (16KB) | L1 (16KB) | L1 (16KB)

Core P#0 PU P#1 | Core P#1 PU P#5 | Core P#2 PU P#9 | Core P#3 PU P#13 | Core P#4 PU P#17 | Core P#5 PU P#21 | Core P#6 PU P#25 | Core P#7 PU P#29

SSComm2

NUMANode P#7 (16GB)

L3 (6144KB)

L2 (2048KB) | L2 (2048KB | L2 (2048KB)

L1 (16KB) | L1 (16KB) | L1 (16KB) | L1 (16KB) | L1 (16KB) | L1 (16KB) | L1 (16KB) | L1 (16KB)

Core P#0 PU P#33 | Core P#1 PU P#37 | Core P#2 PU P#41 | Core P#3 PU P#45 | Core P#4 PU P#49 | Core P#5 PU P#53 | Core P#6 PU P#57 | Core P#7 PU P#61

SComm3

Socket P#1 (32GB)

SSComm1

NUMANode P#2 (16GB)

L3 (6144KB)

L2 (2048KB) | L2 (20 | L2 (2048KB)

L1 (16KB) | L1 (16KB) | L1 (16KB) | L1 (16KB) | L1 (16KB) | L1 (16KB) | L1 (16KB) | L1 (16KB)

Core P#0 PU P#2 | Core P#1 PU P#6 | Core P#2 PU P#10 | Core P#3 PU P#14 | Core P#4 PU P#18 | Core P#5 PU P#22 | Core P#6 PU P#26 | Core P#7 PU P#30

SSComm2

NUMANode P#3 (16GB)

L3 (6144KB)

L2 (2048KB) | L2 (20 | L2 (2048KB)

L1 (16KB) | L1 (16KB) | L1 (16KB) | L1 (16KB) | L1 (16KB) | L1 (16KB) | L1 (16KB) | L1 (16KB)

Core P#0 PU P#34 | Core P#1 PU P#38 | Core P#2 PU P#42 | Core P#3 PU P#46 | Core P#4 PU P#50 | Core P#5 PU P#54 | Core P#6 PU P#58 | Core P#7 PU P#62

SComm4

Socket P#2 (32GB)

SSComm1

NUMANode P#4 (16GB)

L3 (6144KB)

L2 (2048KB) | L2 (2048KB) | L2 (2048KB)

L1 (16KB) | L1 (16KB) | L1 (16KB) | L1 (16KB) | L1 (16KB) | L1 (16KB) | L1 (16KB) | L1 (16KB)

Core P#0 PU P#3 | Core P#1 PU P#7 | Core P#2 PU P#11 | Core P#3 PU P#15 | Core P#4 PU P#19 | Core P#5 PU P#23 | Core P#6 PU P#27 | Core P#7 PU P#31

SSComm2

NUMANode P#5 (16GB)

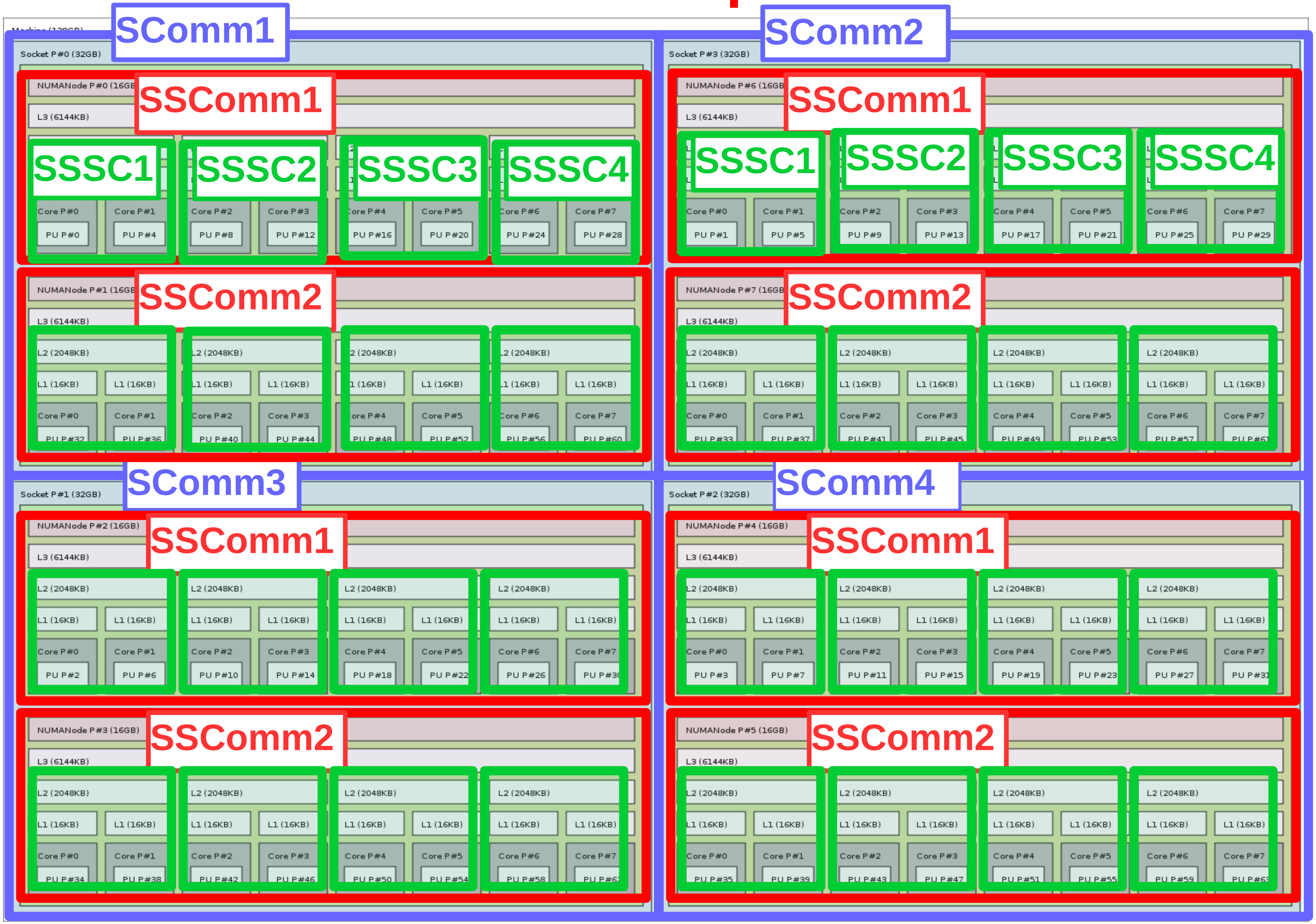
L3 (6144KB)

L2 (2048KB) | L2 (2048KB) | L2 (2048KB)

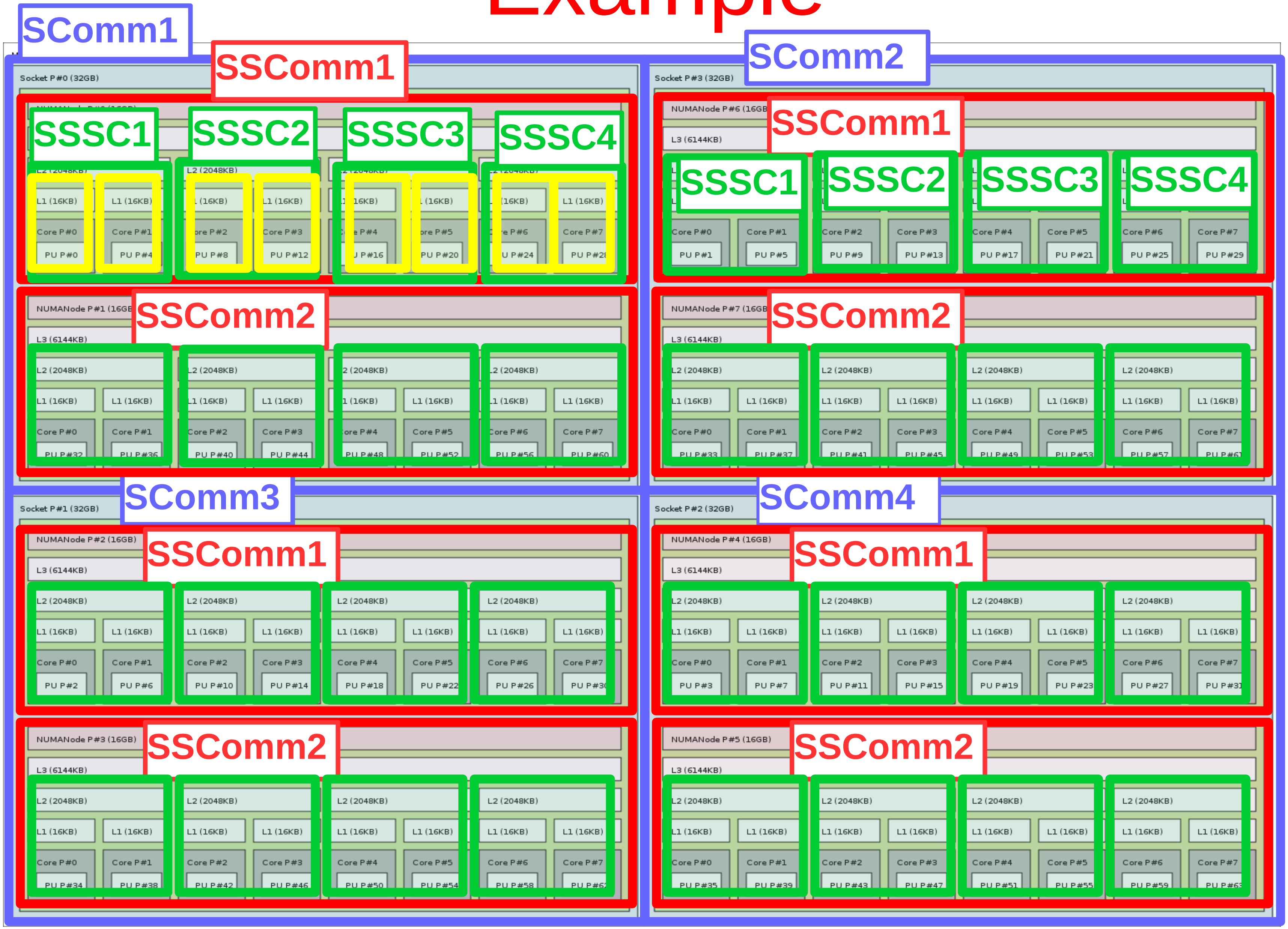
L1 (16KB) | L1 (16KB) | L1 (16KB) | L1 (16KB) | L1 (16KB) | L1 (16KB) | L1 (16KB) | L1 (16KB)

Core P#0 PU P#35 | Core P#1 PU P#39 | Core P#2 PU P#43 | Core P#3 PU P#47 | Core P#4 PU P#51 | Core P#5 PU P#55 | Core P#6 PU P#59 | Core P#7 PU P#63

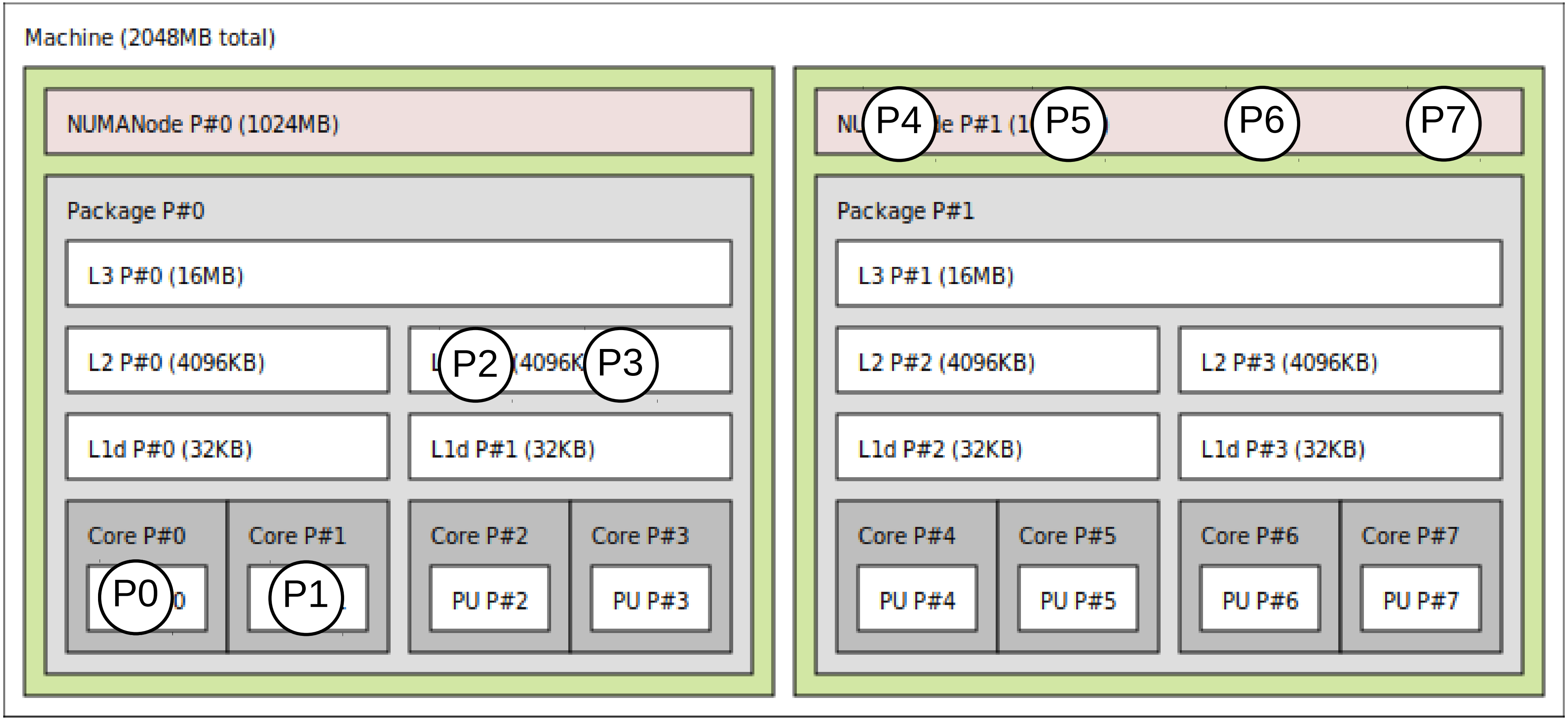
Example


Machine (128GB)
SComm1
Socket P#0 (32GB)
NUMANode P#0 (16GB)
SSComm1
L3 (6144KB)
SSSC1
SSSC2
SSSC3
SSSC4
Core P#0 Core P#1 Core P#2 Core P#3 Core P#4 Core P#5 Core P#6 Core P#7
PU P#0 PU P#4 PU P#8 PU P#12 PU P#16 PU P#20 PU P#24 PU P#28
NUMANode P#1 (16GB)
SSComm2
L3 (6144KB)
L2 (2048KB)
L1 (16KB)
PU P#32 PU P#36 PU P#40 PU P#44 PU P#48 PU P#52 PU P#56 PU P#60
SComm2
Socket P#3 (32GB)
NUMANode P#6 (16GB)
SSComm1
SSSC1
SSSC2
SSSC3
SSSC4
PU P#1 PU P#5 PU P#9 PU P#13 PU P#17 PU P#21 PU P#25 PU P#29
NUMANode P#7 (16GB)
SSComm2
PU P#33 PU P#37 PU P#41 PU P#45 PU P#49 PU P#53 PU P#57 PU P#61
SComm3
Socket P#1 (32GB)
NUMANode P#2 (16GB)
SSComm1
PU P#2 PU P#6 PU P#10 PU P#14 PU P#18 PU P#22 PU P#26 PU P#30
NUMANode P#3 (16GB)
SSComm2
PU P#34 PU P#38 PU P#42 PU P#46 PU P#50 PU P#54 PU P#58 PU P#62
SComm4
Socket P#2 (32GB)
NUMANode P#4 (16GB)
SSComm1
PU P#3 PU P#7 PU P#11 PU P#15 PU P#19 PU P#23 PU P#27 PU P#31
NUMANode P#5 (16GB)
SSComm2
PU P#35 PU P#39 PU P#43 PU P#47 PU P#51 PU P#55 PU P#59 PU P#63

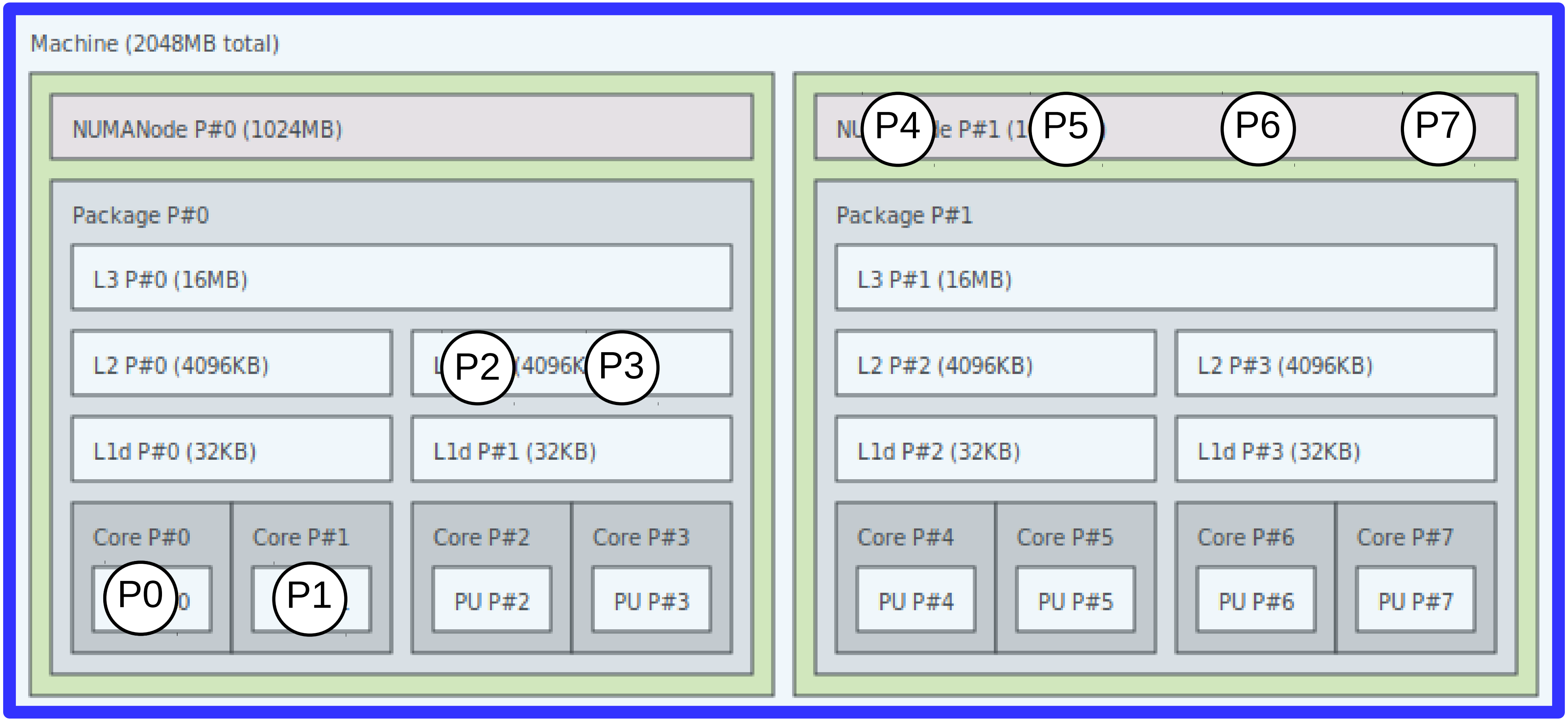
Example


SComm1
SSComm1
SSSC1
SSSC2
SSSC3
SSSC4
SSComm2
SComm2
SSComm1
SSSC1
SSSC2
SSSC3
SSSC4
SSComm2
SComm3
SSComm1
SSComm2
SComm4
SSComm1
SSComm2
Socket P#0 (32GB)
Socket P#3 (32GB)
Socket P#1 (32GB)
Socket P#2 (32GB)
NUMANode P#1 (16GB
NUMANode P#6 (16GB
NUMANode P#7 (16GB
NUMANode P#2 (16GB)
NUMANode P#3 (16GB)
NUMANode P#4 (16GB)
NUMANode P#5 (16GB)
L3 (6144KB)
L2 (2048KB)
L1 (16KB)
Core P#0
Core P#1
Core P#2
Core P#3
Core P#4
Core P#5
Core P#6
Core P#7
PU P#0
PU P#4
PU P#8
PU P#12
PU P#1
PU P#5
PU P#9
PU P#13
PU P#17
PU P#21
PU P#25
PU P#29
PU P#32
PU P#36
PU P#40
PU P#44
PU P#33
PU P#37
PU P#41
PU P#45
PU P#49
PU P#53
PU P#57
PU P#2
PU P#6
PU P#10
PU P#14
PU P#18
PU P#22
PU P#26
PU P#3
PU P#7
PU P#11
PU P#15
PU P#19
PU P#23
PU P#27
PU P#34
PU P#38
PU P#42
PU P#46
PU P#50
PU P#54
PU P#58
PU P#35
PU P#39
PU P#43
PU P#47
PU P#51
PU P#55
PU P#59

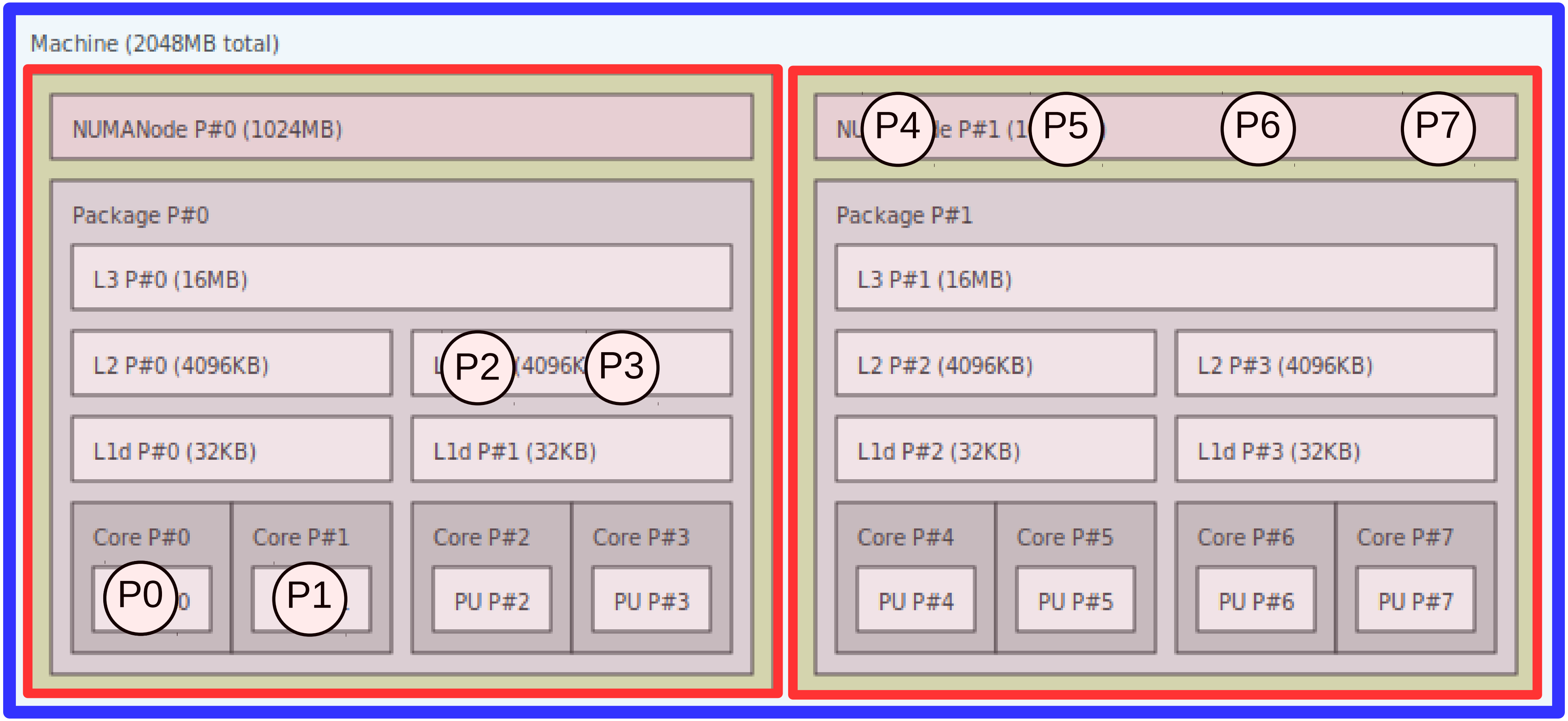
Another example



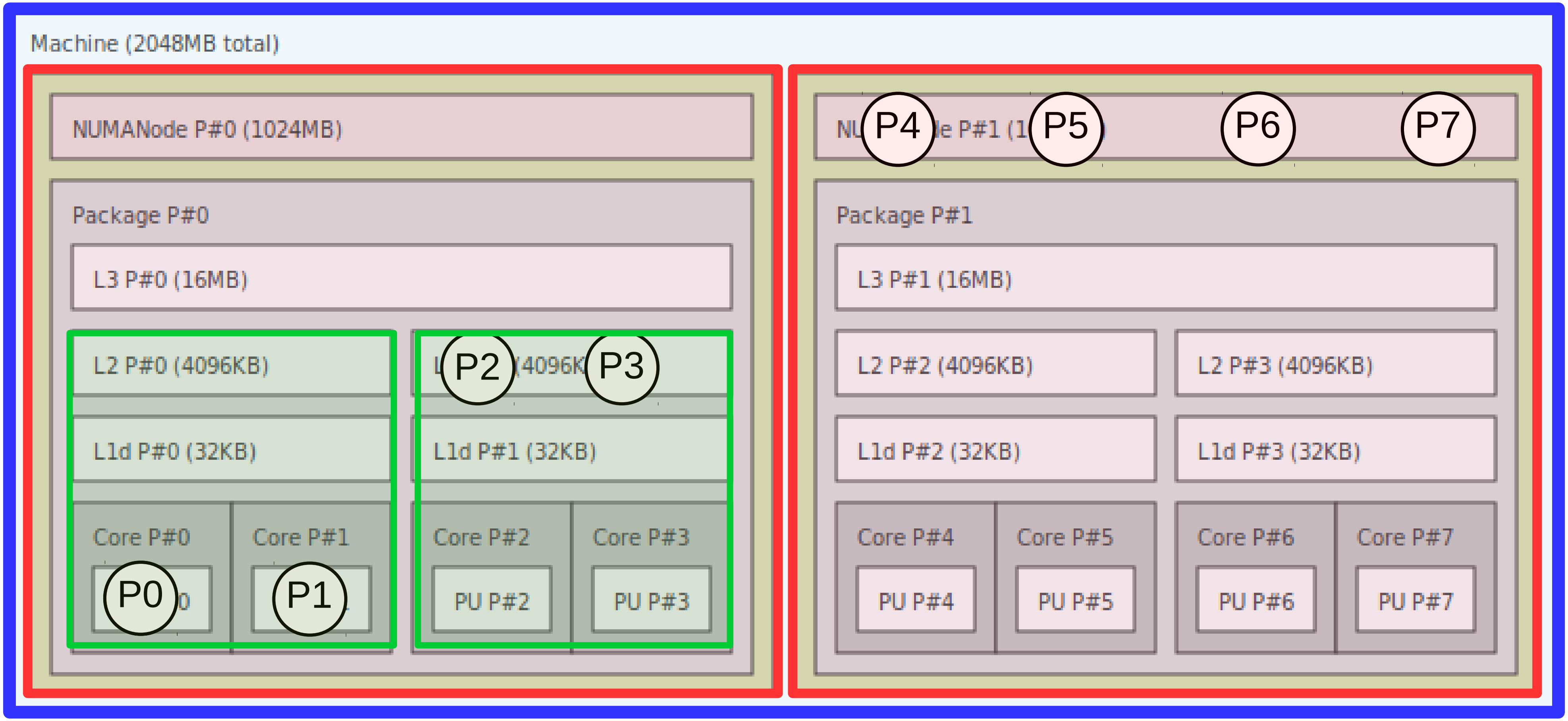
Another example



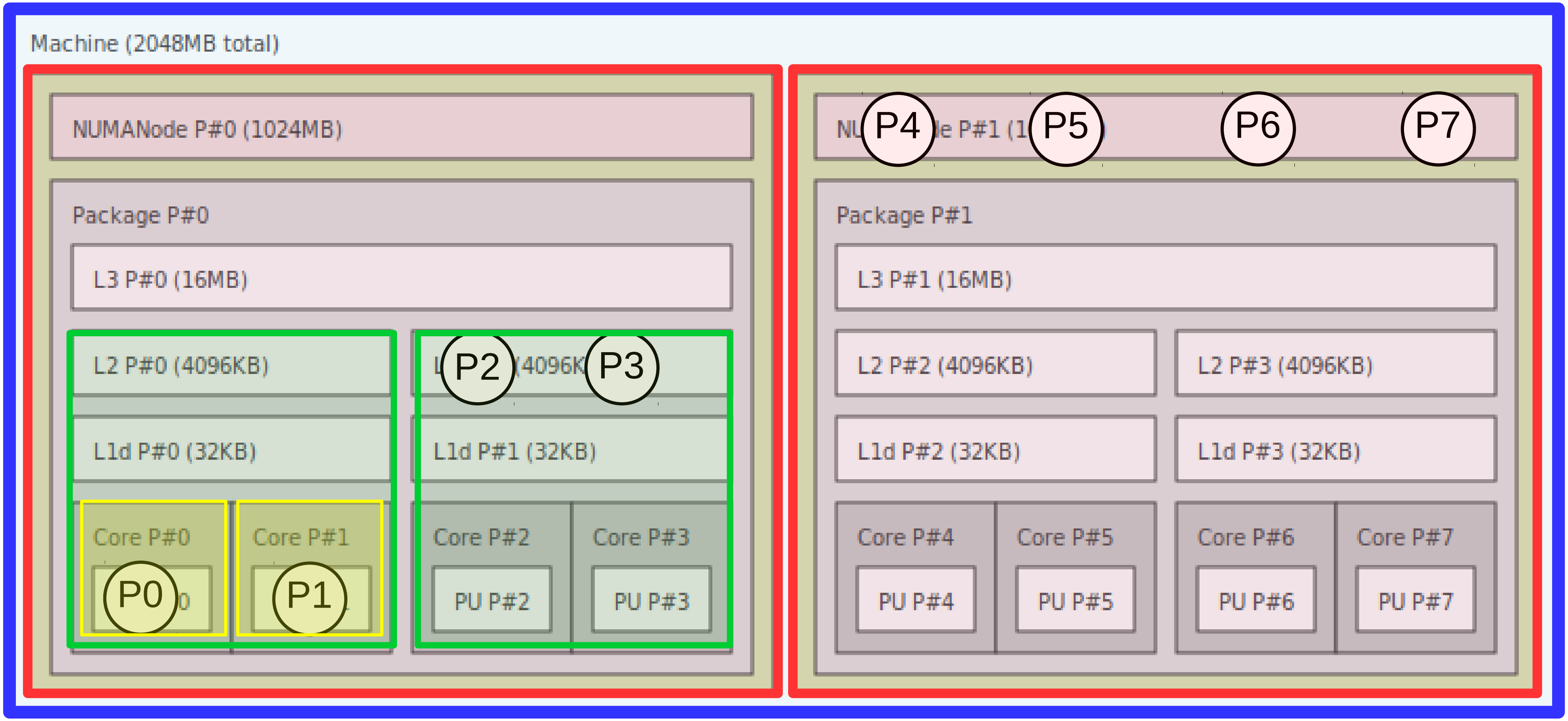
Another example



Another example



Another example



Other routines

- Query routine:

```
int MPI_Get_min_hierarchy(MPI_Comm comm,int size, int *ranks, MPI_Info info);
```

  - Returns the deepest level in the hierarchy encompassing the ranks
  - Result is a string (key: MPI_HIER_LEVEL)
  - In practice, an Hwloc name (e.g : L1,L2,Package)
  - Really useful?

Potentially useful other things

- For data distribution, one might want:
  - Subcommunicators number
  - Subcommunicators ranks
- This can be stored in an info objet attached to the subcommunicator
  - MPI_Comm_set_info
  - MPI_Comm_get_info
- Really needed?
  - See Roots comms

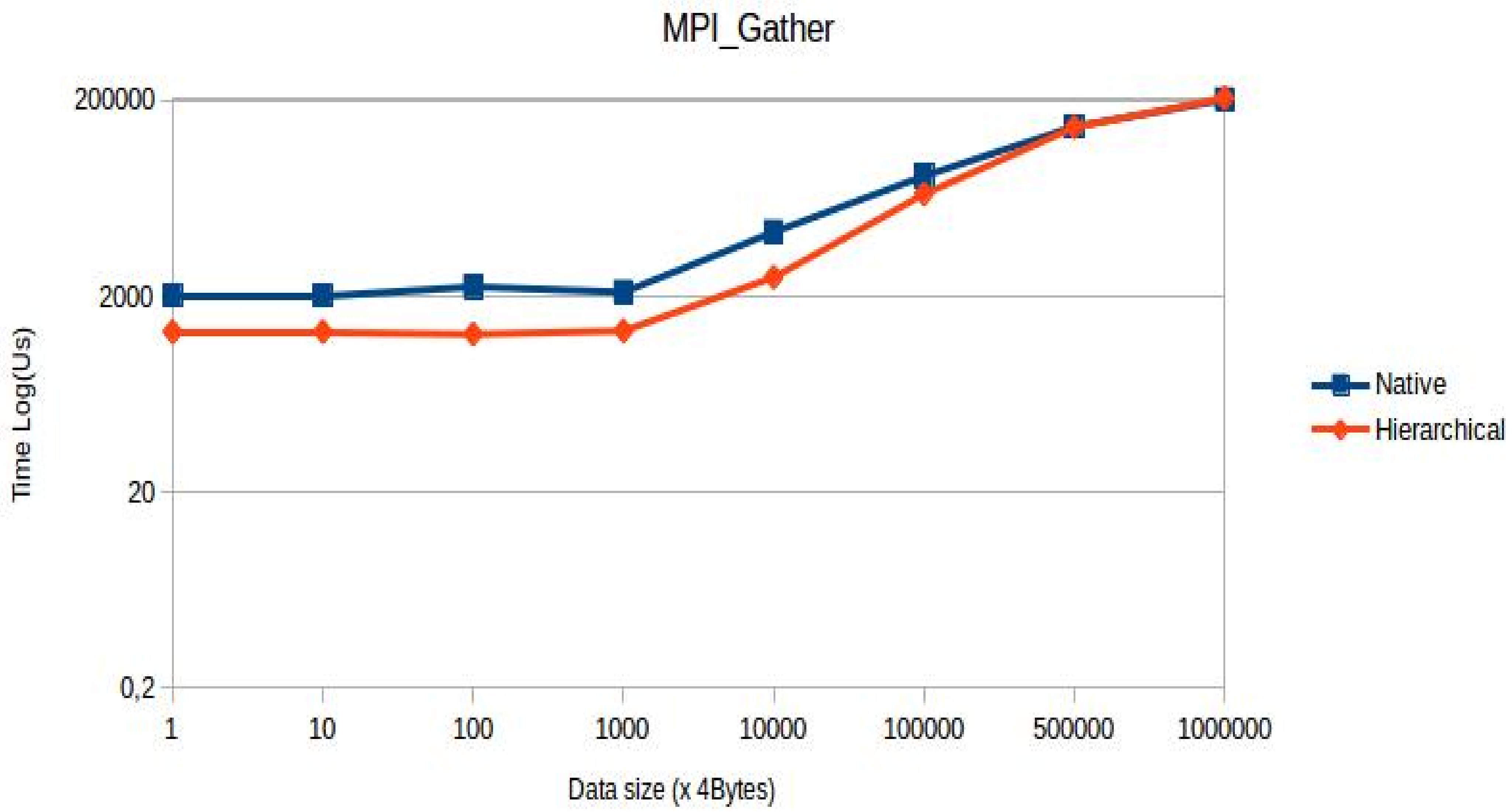
Roots Communicators

- In practice, the subcommunicators are not enough to write applications
  - Need to access easily the roots of each subcommunicator
  - Hierarchy of Roots subcommunicators
    - The root of a communicator should also be the root of a newly created communicator in the hierarchy
    - Easy with the right key in Comm_split
- Lastovtesky papers
  - Hierarchical decomposition of collectives can improve performance
  - Topology-oblivious implementations (so far)

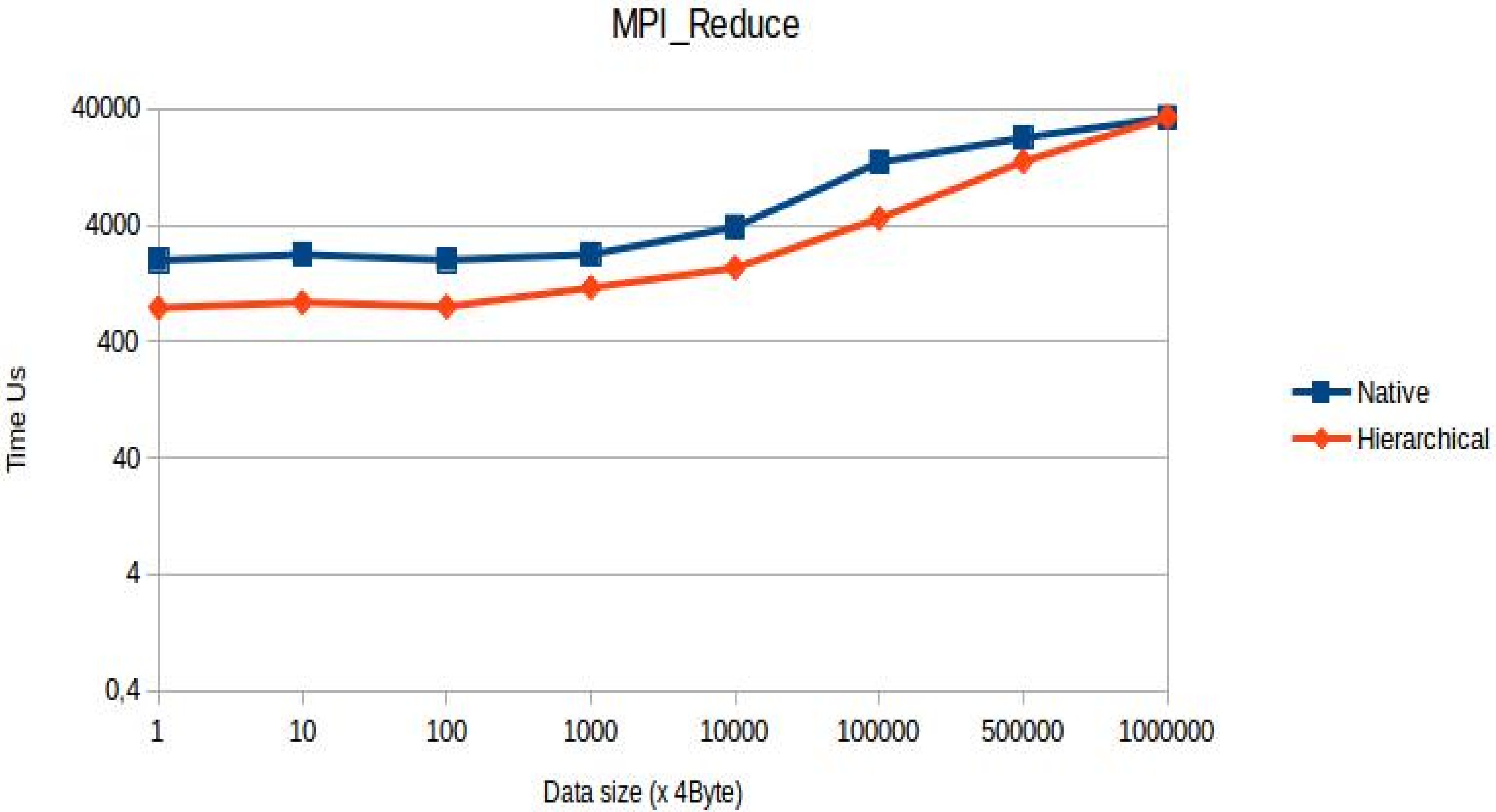
Implementation

- Prototype available
- Implemented as an external library
- Hwloc-based implementation
  - Does not (yet) address network topology
  - But Netloc should help :)
- Use of MPI_Comm_split
- Roots Communicators creation
  - Easy with hwloc: use the logical_index of the parent level as the color for the split
  - Possible with MPI calls, but less efficient (involves more collectives)

Preliminary Results



Preliminary Results



Acknowledgements

- Brice Goglin
- Emmanuel Jeannot
- Farouk Mansouri
- Work funded by ELCI Project :

  http://elciproject.fr/