ROME Workshop @ IPDPS Vancouver

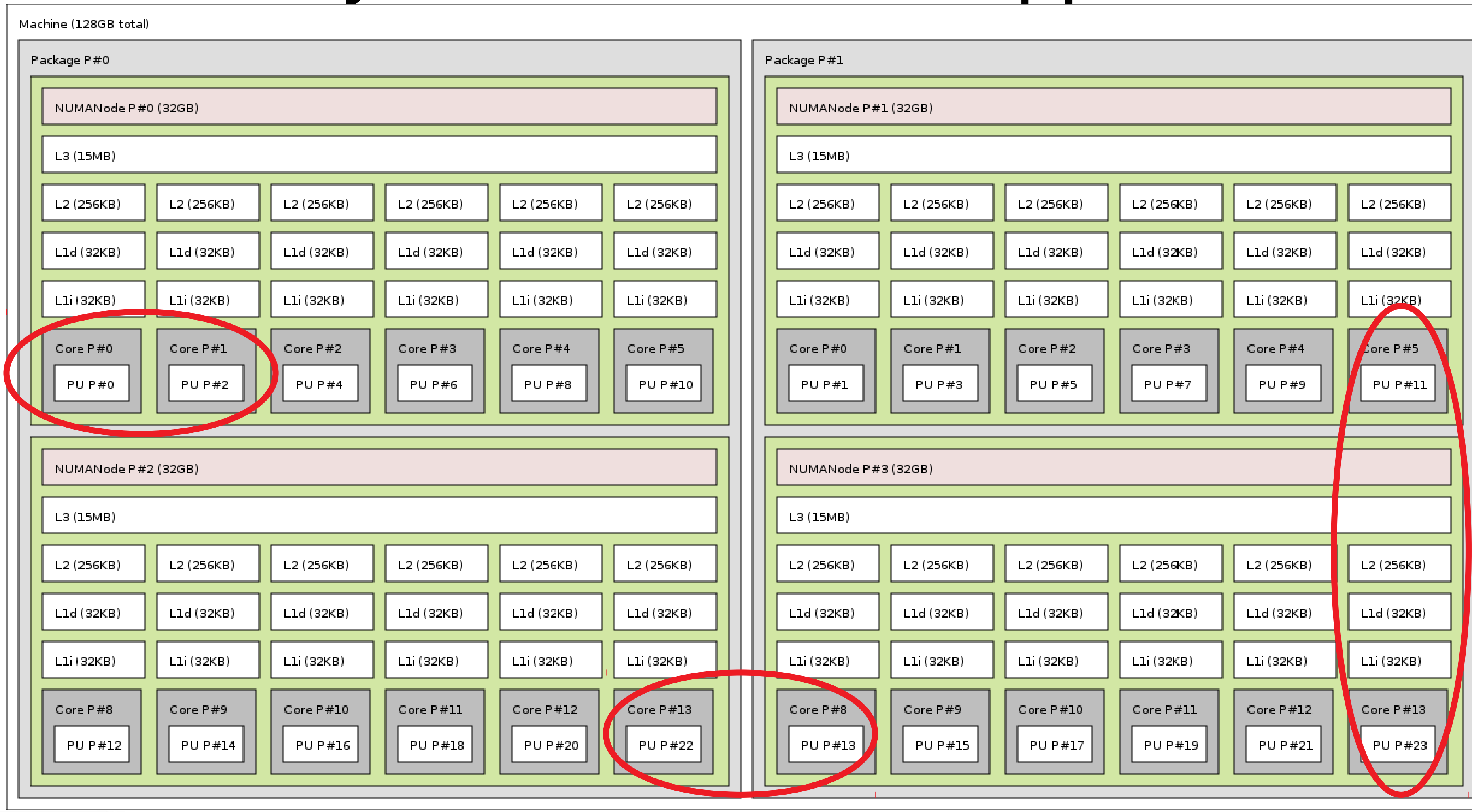
Memory Footprint of Locality Information On Many-Core Platforms

Brice Goglin

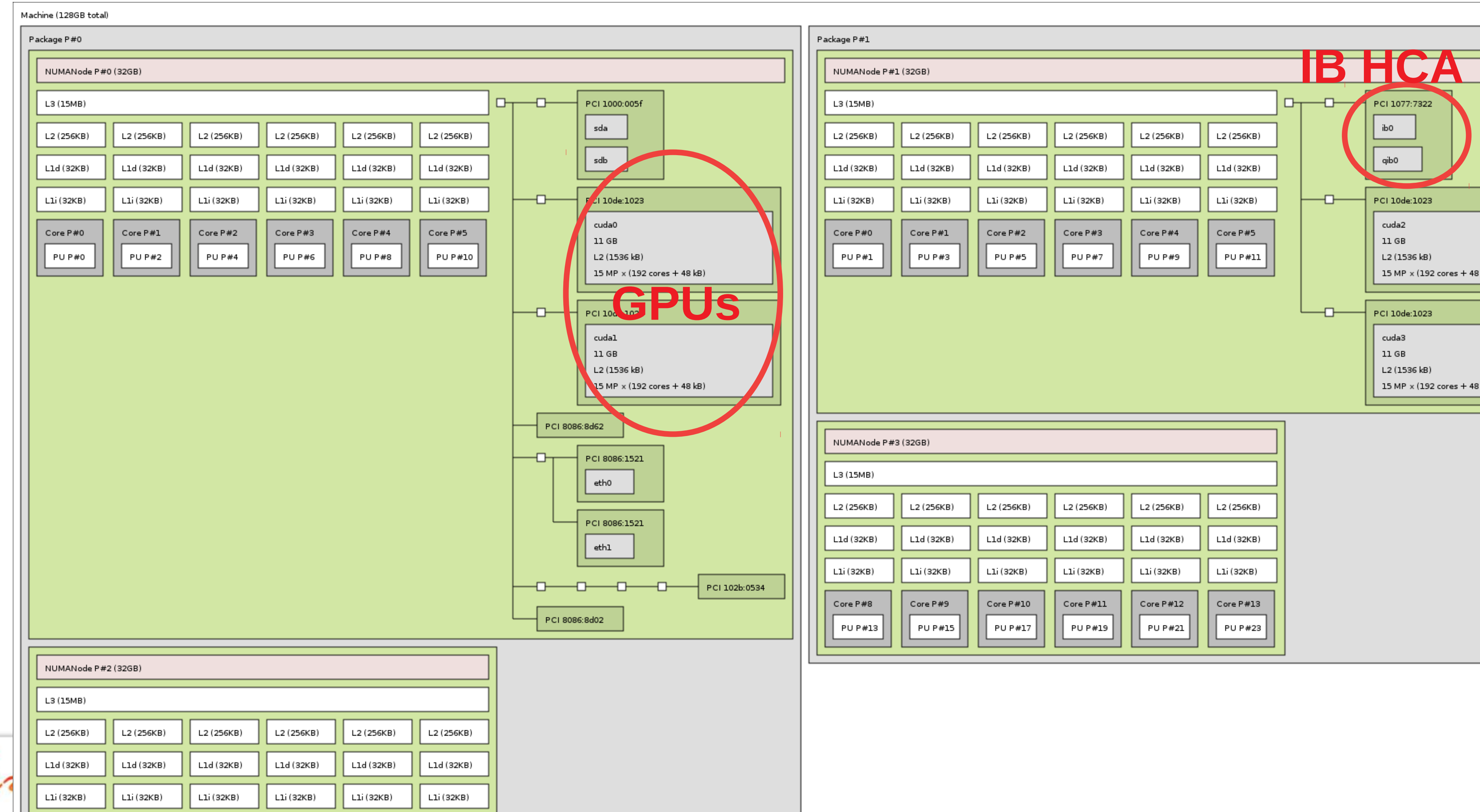
Inria Bordeaux – Sud-Ouest – France

2018/05/25

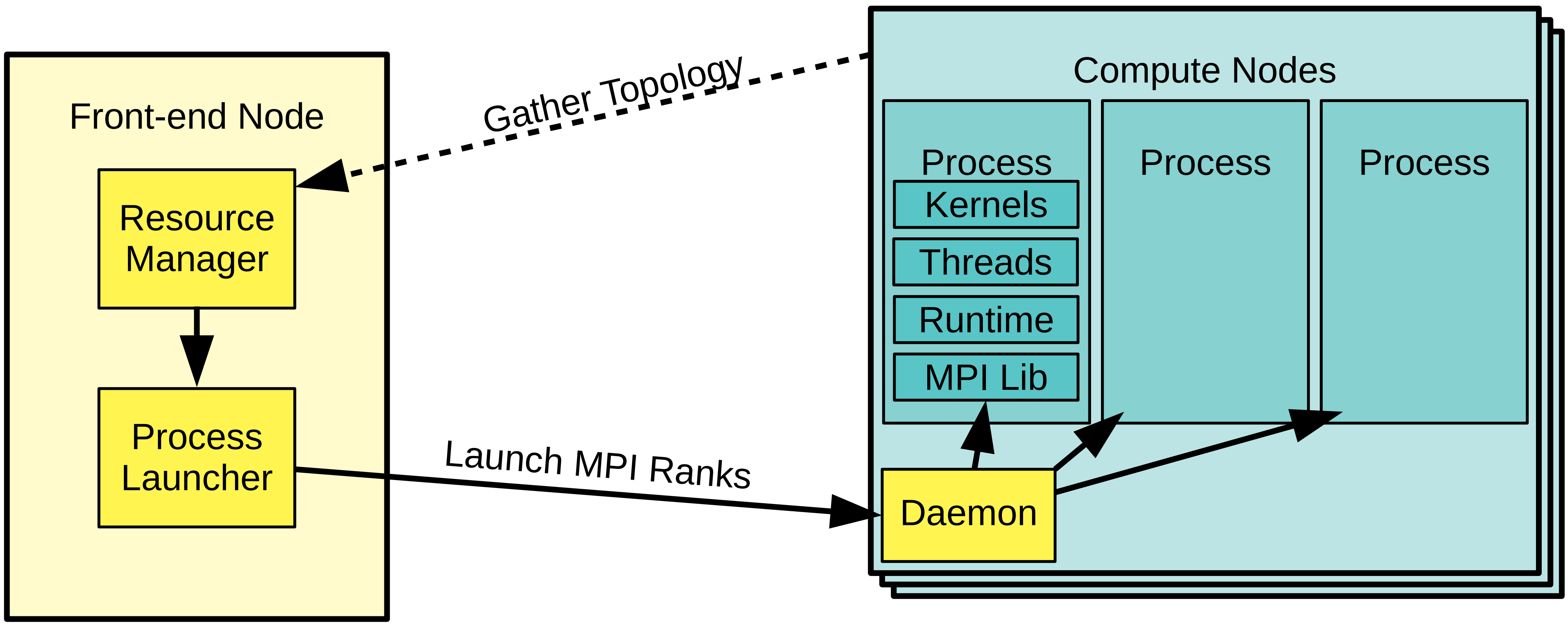
Locality Matters to HPC Applications



Locality Matters for I/O too



Who Needs Locality in the HPC Stack?



Memory in HPC Platforms

Top500 – 2017/11

Rank	*Name*	*Cores*	*Memory*	*GB per Core*
#1	Sunway TaihuLight	10 649 600	1.31 PB	0.12
#2	Tianhe-2	3 120 000	1 PB	0.32
#3	**Piz Daint**	**361 760**	**340 TB**	**1.06**
#4	Gyoukou	19 860 000	575 TB	0.028
#5	Titan	560 640	710 TB	1.27
#6	**Sequoia**	**1 572 864**	**1.5 PB**	**1**
#7	**Trinity**	**301 056 + 678 912**	**2 PB**	**2**
#8	**Cori**	**622 336**	**878 TB**	**1.41**
#9	**Oakforest-PACS**	**556 104**	**919 TB**	**1.65**
#10	**K-computer**	**705 024**	**1.4 PB**	**2**

hwloc’s Modeling of Platforms

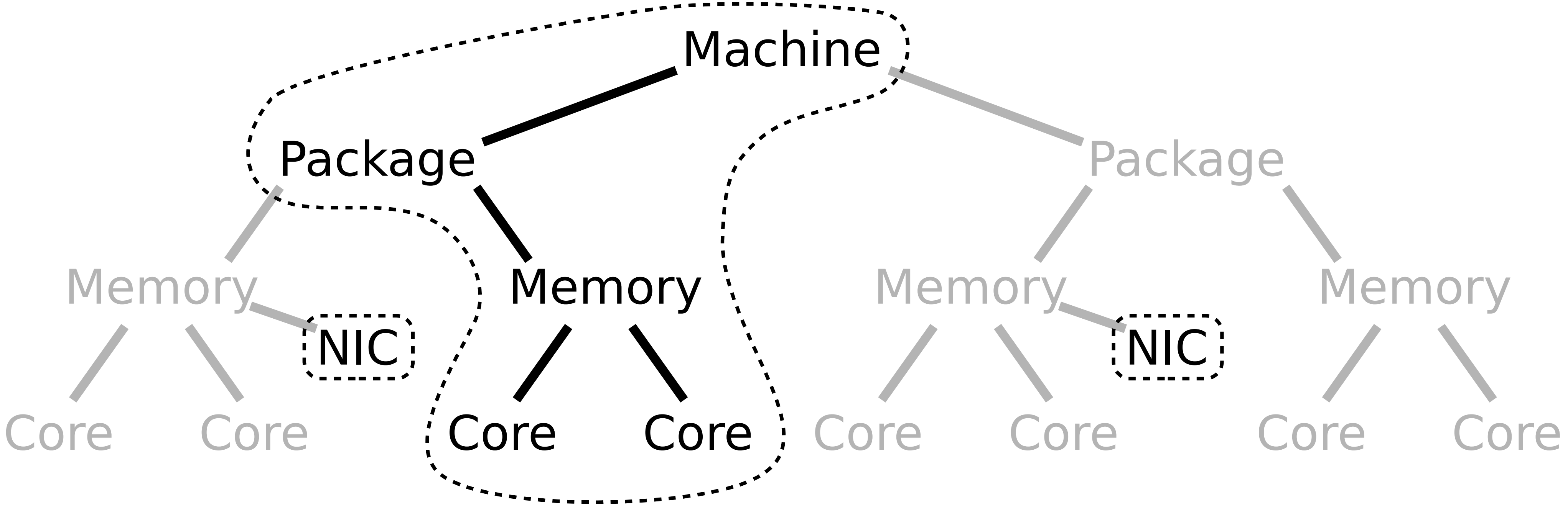
- Tree of hierarchical resource objects (hwloc_obj structure)
  - With many attributes
    - Location with respect to CPU and memory resources (bitmaps)
    - Indexes
    - Links to parent, children, siblings, cousins
    - Type-specific attributes
      - Amount of memory, kind of cache, etc.
    - Strings for custom attributes
      - CPU model, MAC address, name, PCI vendor, etc.
- A little bit of system-wide info
  - hwloc_topology structure

hwloc Memory Footprint on KNL

- Between 400 and 500 objects
  - 256 hwthreads (PUs), 64 caches per level, 64 cores
  - Between 1 and 8 NUMA nodes
  - Some I/O objects
- About 700kB total
- Some users complain
  - They use ***many*** processes per node
  - They want to keep that memory available for the application
    - Even if it’s about 0.1 percent of the available memory per core
  - Things will get worse in the future

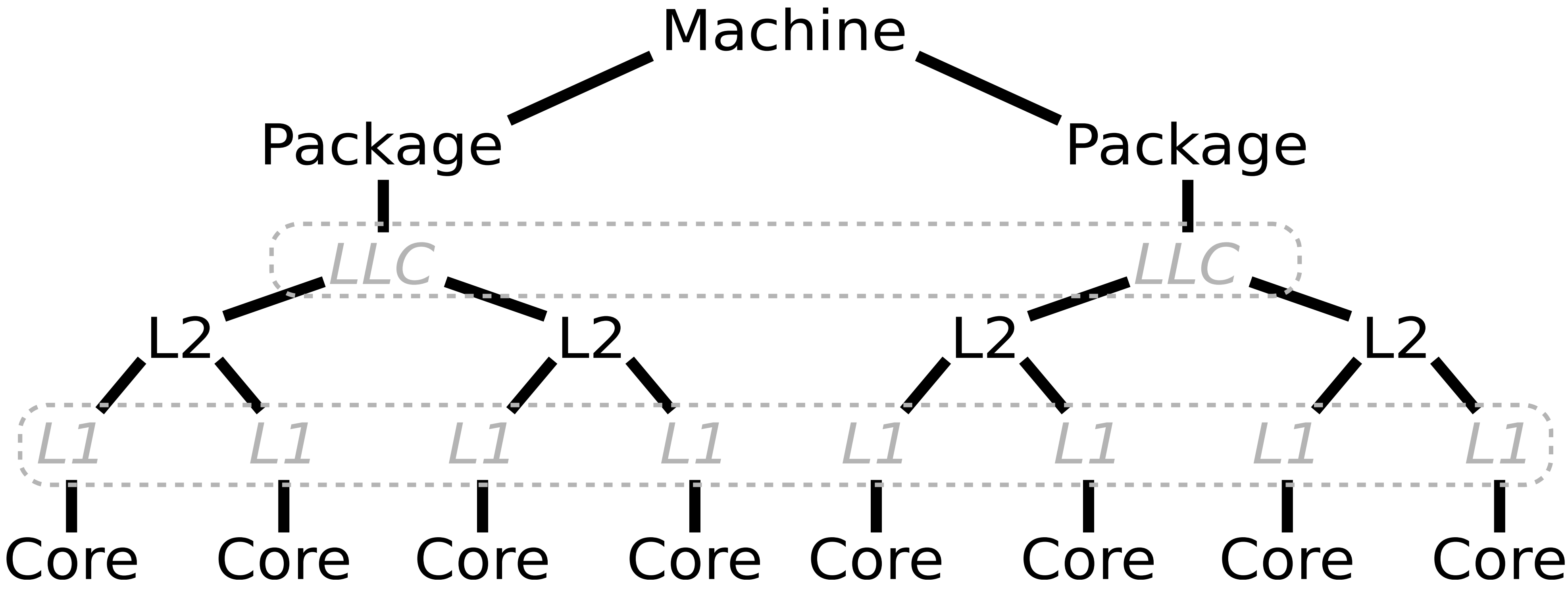
Horizontal Filtering of Available Resources

Only part of the platform is available to each job



Vertical Filtering of Useful Resources

Some levels aren't useful



Possible Ways to Manage Multiple Clients

- Native Discovery
  - Expensive, should be performed as rarely as possible (PDP’17)
- XML exchange
  - Much faster
  - Still instantiates multiple topologies in memory
- Centralizing in a server
  - Single instance
  - Requires to redirect process queries to the server
    - Slower, API change?
- Shared memory

Shared Memory, obviously but ...

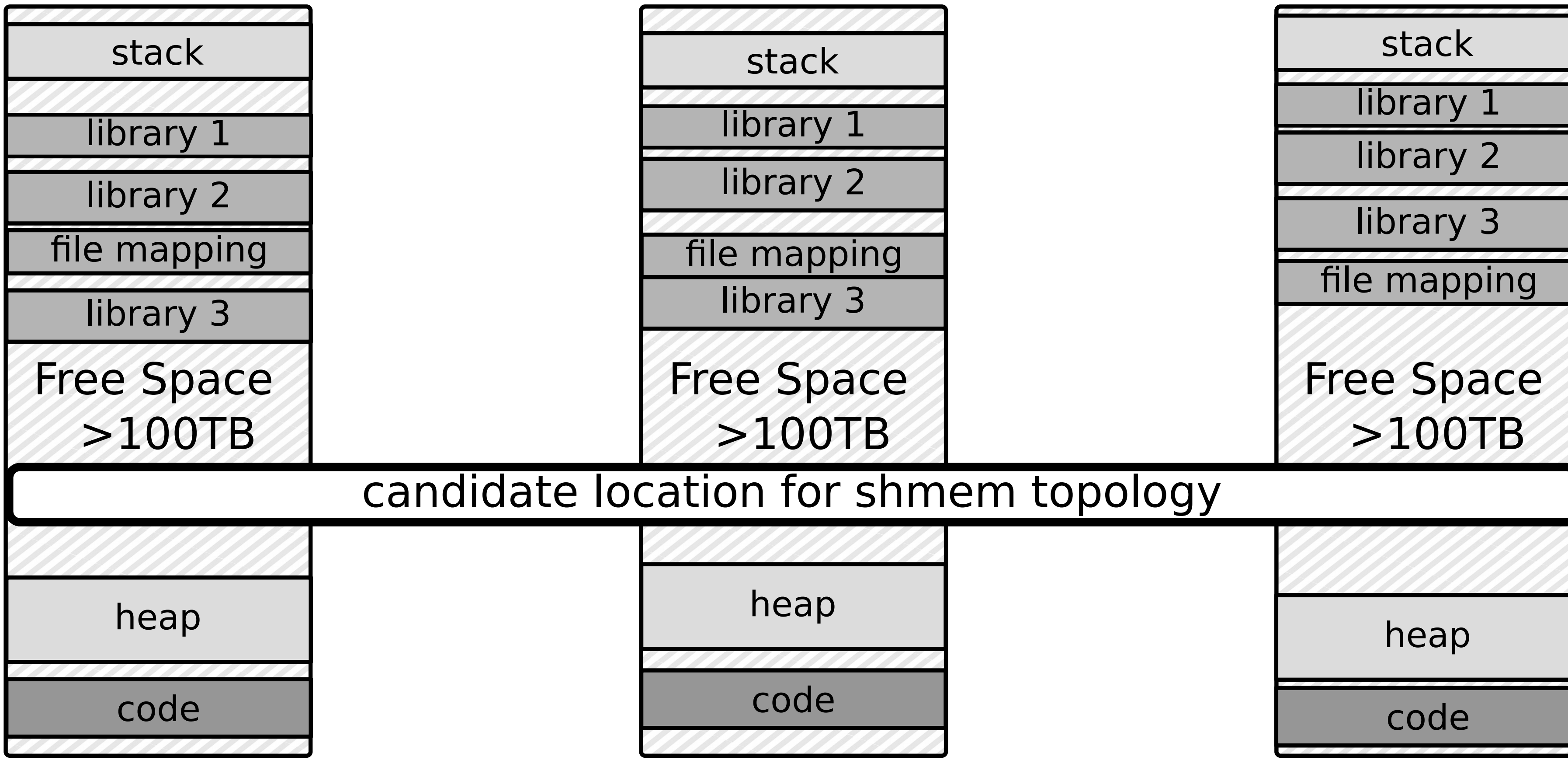
- hwloc was designed in 2009
  - Many objects attributes, many ways to traverse the topology
    - We decided we didn’t want so maaaaaany accessor functions to manipulate these attributes and pointers
- Many users are tied to the existing API
  - Pointers must remain valid, even if mapped in another process
    - Means all processes must map at the ***same*** virtual address
  - Or we would have to replace the entire existing API

The Virtual Address Space is mostly empty

- 128TB of VA on current x86 platforms
  - 64PB on next-generation (Intel la57 extension)
- Similar values on ARM64 and Power
- The available per-core physical memory is ***MUCH*** lower (GB)

- Trinity/KNL (96GB/node)
  - 99.925% of VM free if one process per node
  - 99.9988% if one process per core
- Summit/P9 (512GB/node) 99.2% and 99.981% respectively

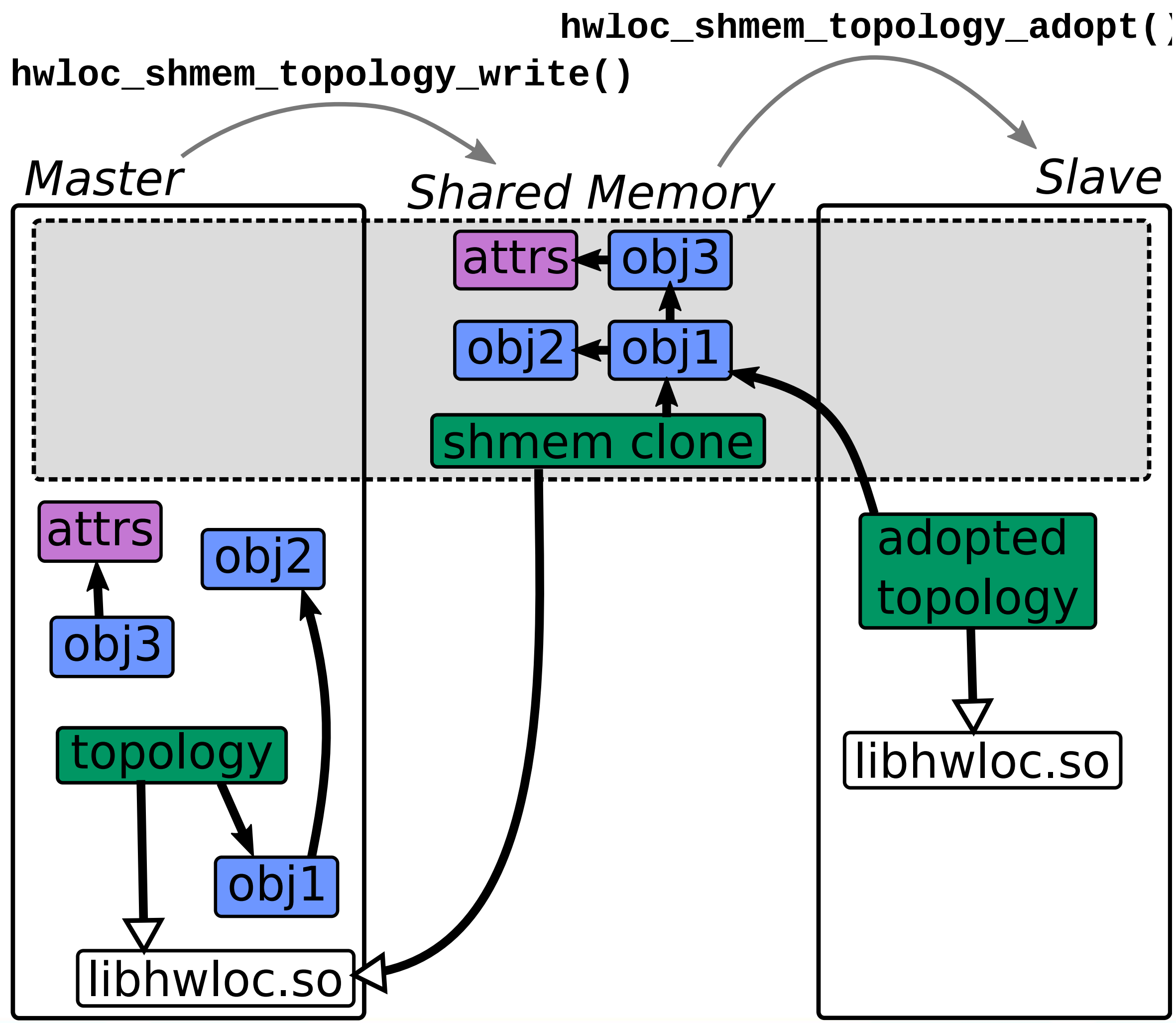
Virtual Address Space Layout on Linux



Implementation in Open MPI

- One ORTE daemon per node
  - Finds the largest hole in its own virtual address space
    - Doesn’t know what other processes will look like
  - Allocates a shared memory region there
  - Stores the hwloc topology in it
- MPI ranks map that shared region
  - Use the hwloc topology contained there
  - If mapping failed (e.g. virtual address range not available)
    - Fall back to XML as usual

hwloc shmem topology



Experimentation Platforms

- KNL64 = 430 hwloc objects
  - Intel Xeon Phi 7230 (64 cores, 1.3GHz)
  - SNC-4, Flat
- NUMA96 = 405 hwloc objects
  - 4x Intel Xeon E7-8890v4 (24 cores each, 2.2GHz)
  - Cluster-on-Die, no Hyper-threading
- Normal24 = 97 objects
  - 2x Intel Xeon E5-2680v3 (12 cores each, 2.5GHz)
  - Cluster-on-Die, no Hyper-threading

Memory Footprint per MPI rank

- ORTE hello instrumented with mallinfo

	Native Discovery	XML	Shared-Memory	No topology
KNL64	2.21MiB	2.35MiB	1.614MiB	1.613MiB
NUMA96	1.82MiB	1.94MiB	1.230MiB	1.229MiB
Normal24	1.74MiB	1.78MiB	1.535MiB	1.534MiB

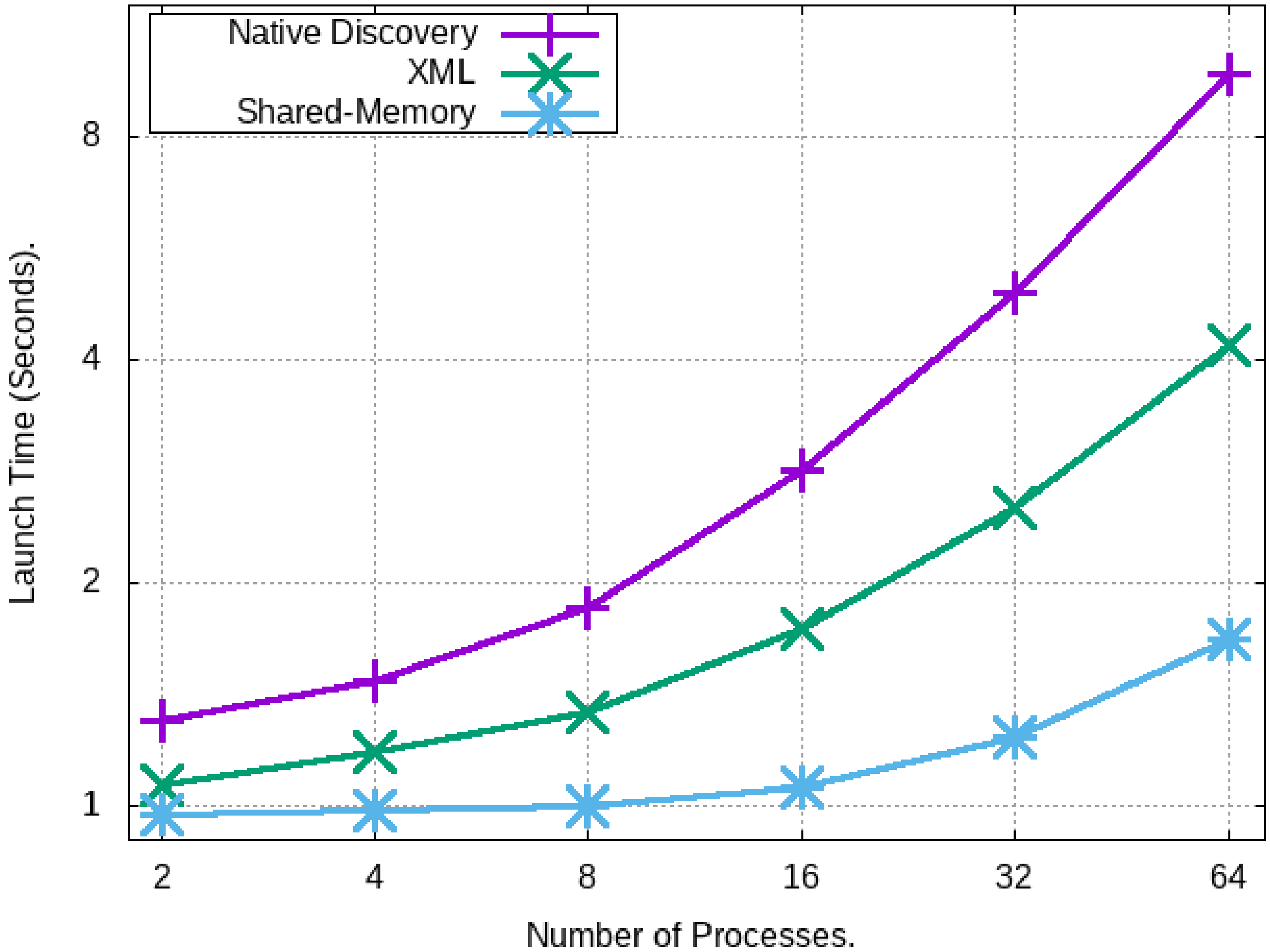
Memory Saving per Node

		Saving per Node
KNL64	1 process	0
	2 processes	500kiB
	64 processes (1 per core)	31MiB
	256 processes (1 per hwthread)	127MiB
NUMA96	1 process	0
	2 processes	590kiB
	96 processes (1 per core)	56MiB
Normal24	1 process	0
	2 processes	200kiB
	24 processes (1 per core)	4.6MiB

Launch Time

	Native Discovery	XML	Shared-Memory	Speedup vs XML
KNL64 – 64 procs	9.69s	4.16s	1.68s	x 2.48
KNL64 – 256 procs	47.20s	18.45s	7.02s	x 2.63
NUMA96 – 96 procs	7.29s	1.17s	0.56s	x 2.10
Normal24 – 24 procs	0.84s	0.53s	0.47s	x 1.13

Launch Time – KNL64



Conclusion

- Many components of the HPC stack use topology information
  - And many processes per node
- Must share topology information to reduce memory footprint
  - Already needed on many-core platforms

Contribution

- hwloc may now place topology in shared-memory
- We designed a way to use the same virtual address in all processes
  - Required to maintain compatibility with old hwloc API
- Available in Open MPI 4.0 and hwloc 2.0
- Reduces footprint to a single topology per node
- Reduces launch time significantly

Future Work

- Share topology between jobs with different sets of allocated resources on same node
- Extend to other process managers
  - Slurm’s srun, etc.
- Propagate shared topology information to all layers inside each process
  - Cooperation between MPI, OpenMP, etc.

Thank you for your attention

Brice.Goglin@inria.fr

informatics mathematics