DATA SCIENCE &
SCIENTIFIC COMPUTING

L07: multicore architecture

- Stefano Cozzini
- CNR-IOM and eXact lab srl

Goal of the day

- Get acquainted of basic brick on modern HPC system
- Learn about pro/cons of such architecture
- Start using tools to correctly exploit (almost) all the cores of the architecture

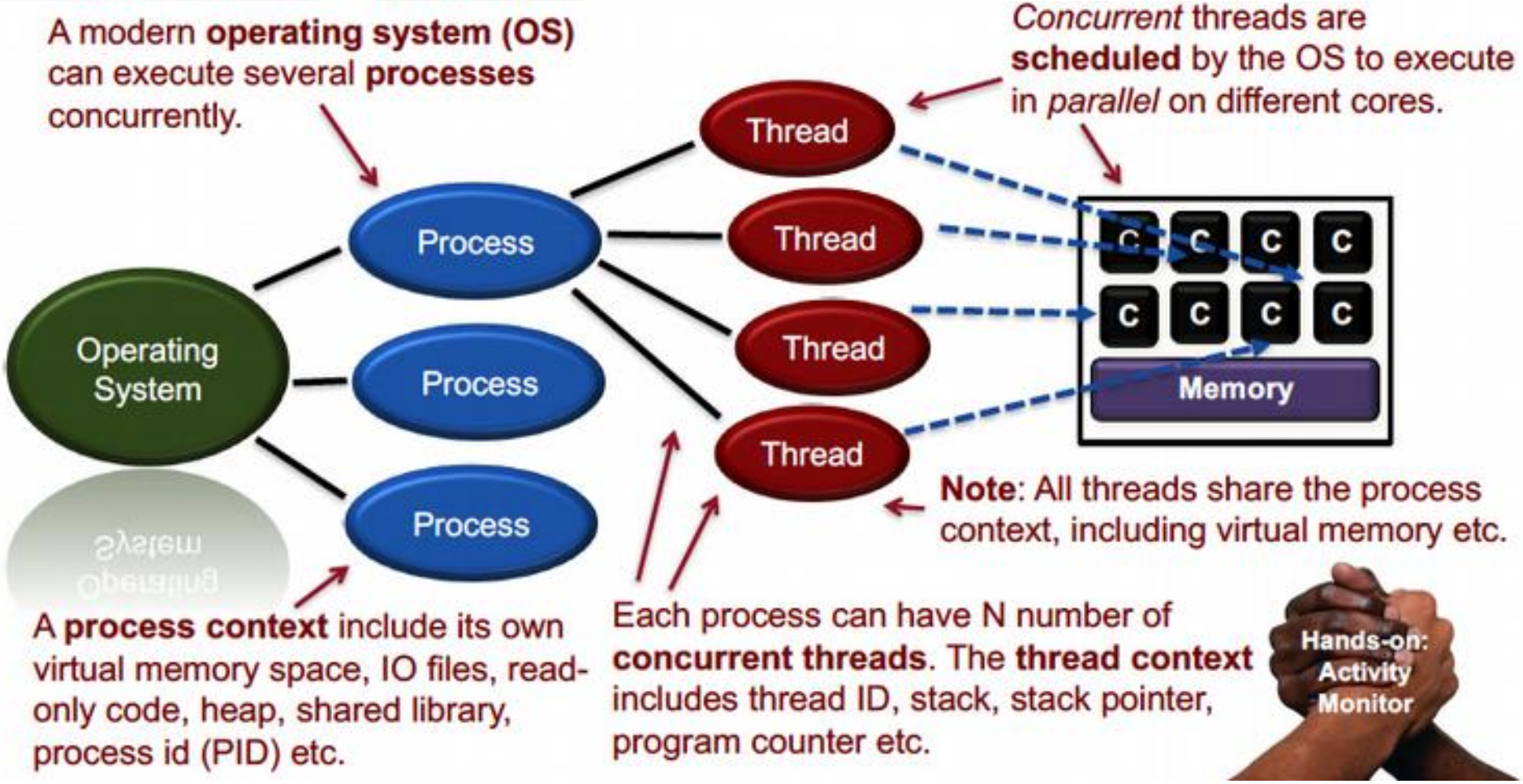
RECAP

- Modern nodes are multiprocessor (more than one CPUs) and each CPU is multicore
- RAM Memory is shared among all cores
- L1/L2 caches are private to cores
- L3 is shared within the same CPU
- The overall architecture is NUMA

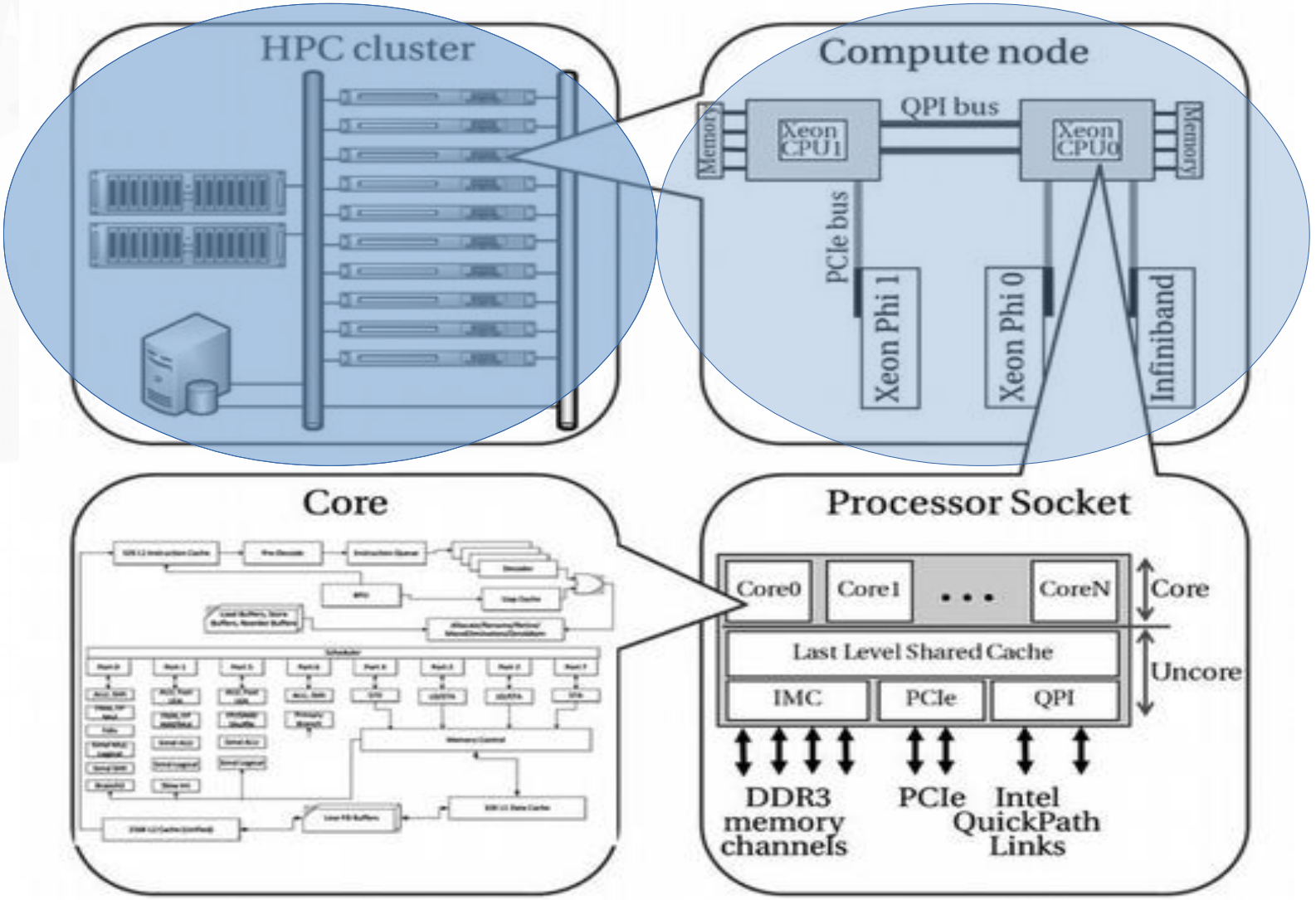
Motivation for multicores

- Exploits increased feature-size and density
- Increases functional units per chip (spatial efficiency)
- Limits energy consumption per operations
- Constrains growth in processor complexity

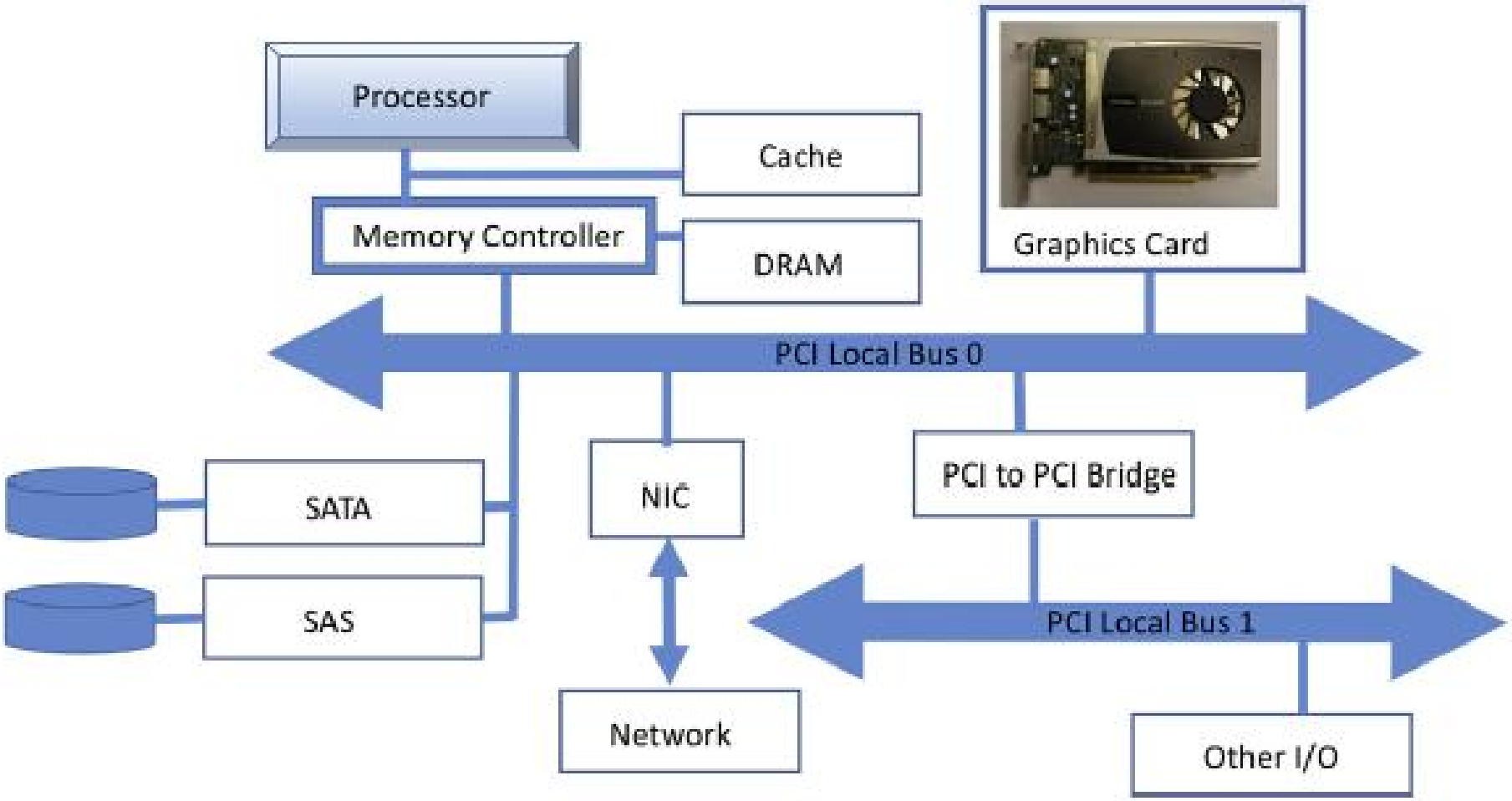
Processes, Threads and Cores



The building blocks of the HPC cluster



Buses within a computer



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

Communication interfaces within server

- Recent trends in I/O interfaces show that they are nearly matching state of the art network speeds

AMD HyperTransport (HT)	2001 (v1.0), 2004 (v2.0) 2006 (v3.0), 2008 (v3.1)	102.4Gbps (v1.0), 179.2Gbps (v2.0) 332.8Gbps (v3.0), 409.6Gbps (v3.1) (32 lanes)
PCI-Express (PCIe) by Intel	2003 (Gen1), 2007 (Gen2), 2009 (Gen3 standard), 2017 (Gen4 standard)	Gen1: 4X (8Gbps), 8X (16Gbps), 16X (32Gbps) Gen2: 4X (16Gbps), 8X (32Gbps), 16X (64Gbps) Gen3: 4X (~32Gbps), 8X (~64Gbps), 16X (~128Gbps) Gen4: 4X (~64Gbps), 8X (~128Gbps), 16X (~256Gbps)
Intel QuickPath Interconnect (QPI)	2009	153.6-204.8Gbps (20 lanes)

PCI-express speed (from wikipedia)

PCI Express link performance[30][31]

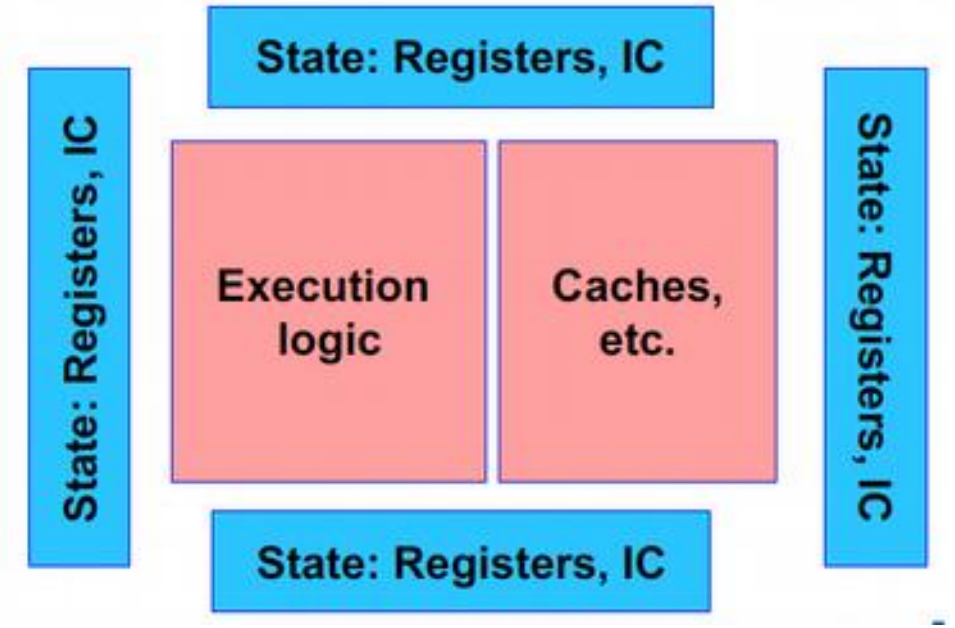
PCI Express version	Introduced	Line code	Transfer rate[i]	Throughput[i]				
				x1	x2	x4	x8	x16
1.0	2003	8b/10b	2.5 GT/s	250 MB/s	0.50 GB/s	1.0 GB/s	2.0 GB/s	4.0 GB/s
2.0	2007	8b/10b	5.0 GT/s	500 MB/s	1.0 GB/s	2.0 GB/s	4.0 GB/s	8.0 GB/s
3.0	2010	128b/130b	8.0 GT/s	984.6 MB/s	1.97 GB/s	3.94 GB/s	7.88 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

Recap: elements in a core

- Pipelining of arithmetic/functional units
  - Split complex instruction into several simple / fast steps (stages)
  - Execute different steps on instructions at the same time (in parallel)
  - after the pipeline is full one result at each cycle
- Superscalar processor:
  - Multiple units enable Instruction Level Parallelism (ILP)
  - 2-6 way superscalar : 2-6 instructions at each cycle
- SIMD extensions (AVX/SSE2)
  - Single Instruction Multiple data approach: 2/4/8 FP at each cycle
  - Pointer aliasing may prevent SIMD...
  - No SIMD approach on loop with dependencies: A(I)= A(I-1) *B(I)

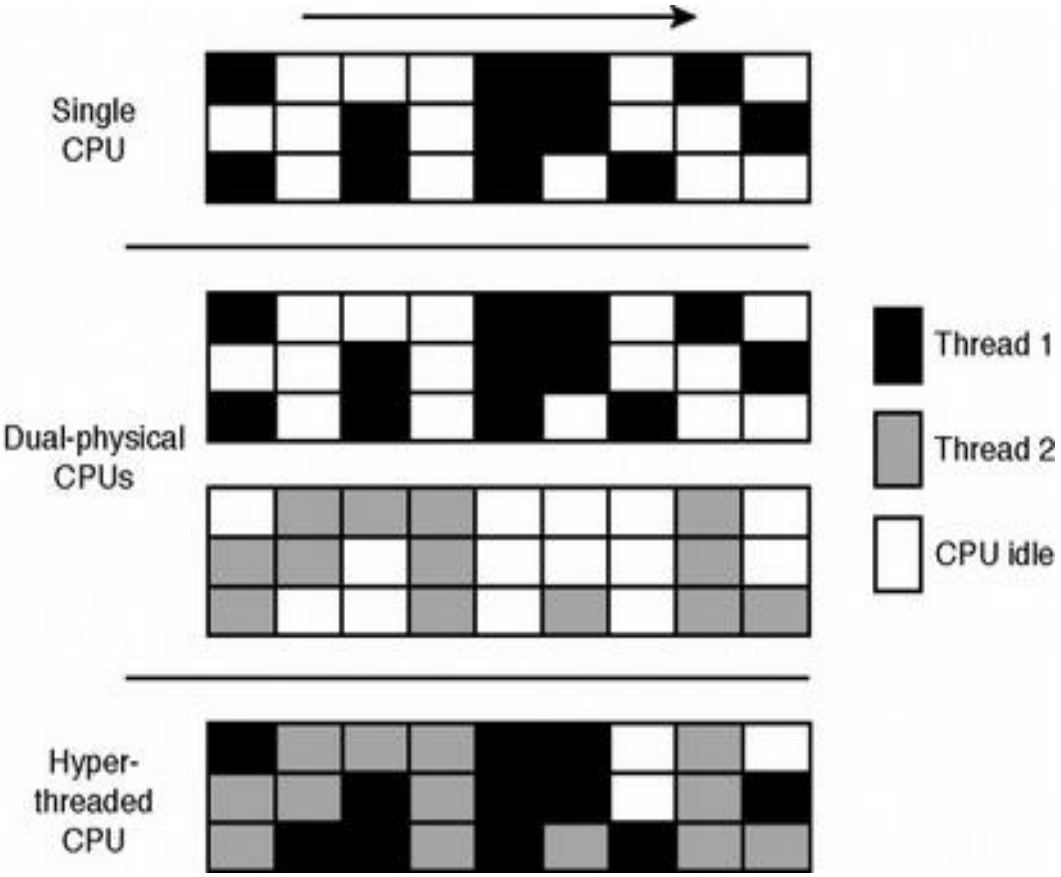
Definition of hardware core/thread

- Core
  - A complete ensemble of execution logic, and cache storage as well as register files plus instruction counter (IC) for executing a software process or thread
- Hardware thread (aka hyper-thread)
  - Addition of a set of register files plus IC



Hyper threading (HT)

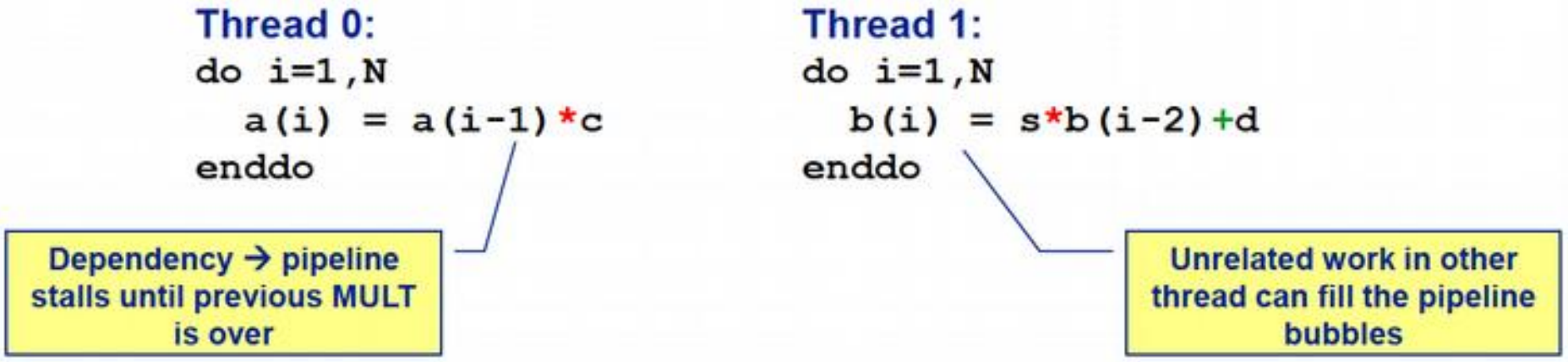
- Intel® Hyper-Threading Technology uses processor resources more efficiently, enabling multiple threads to run on each core.
- O.S. “sees” two cores and transparently try to execute two program on two different “cores”
- Generally bad for HPC ?



Reference: pag 514 (paragraph 6.4)

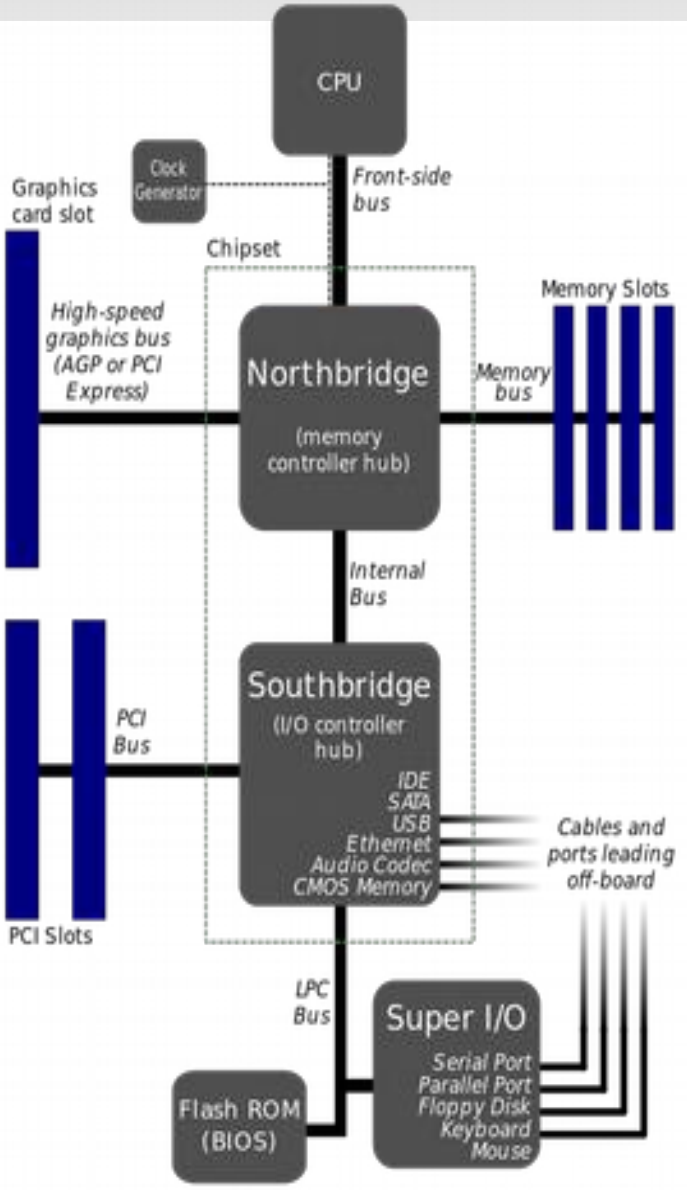
Does SMT/HT work for HPC kernel?

- SMT/HT introduce an additional topological layer inside the core
- All caches and PU are shared !
- Possible advantage: better pipeline throughput
  - Filling otherwise unused pipeline
  - Filling pipelines bubbles with other threads execution's instructions



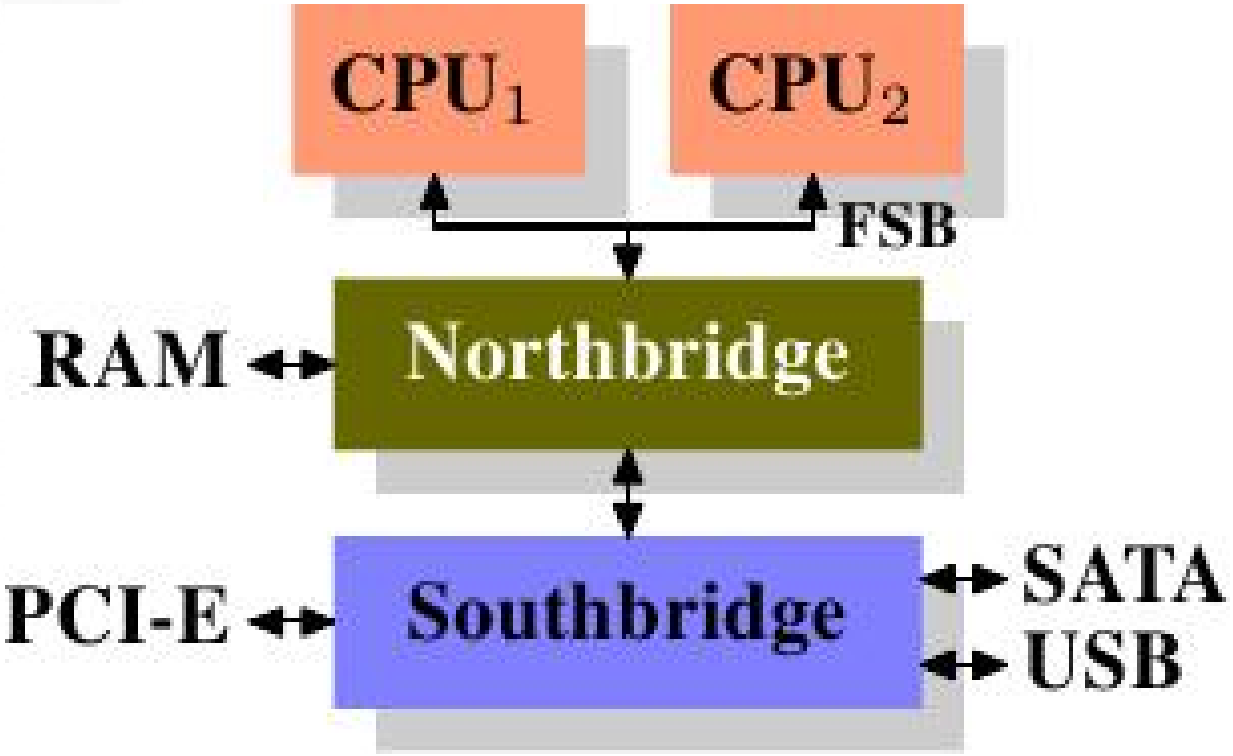
standard single cpu architecture

- All data communication from one CPU to another must travel over the same bus used to communicate with the Northbridge.
- All communication with RAM must pass through the Northbridge.
- Communication between a CPU and a device attached to the Southbridge is routed through the Northbridge.



standard SMP multisocket architecture

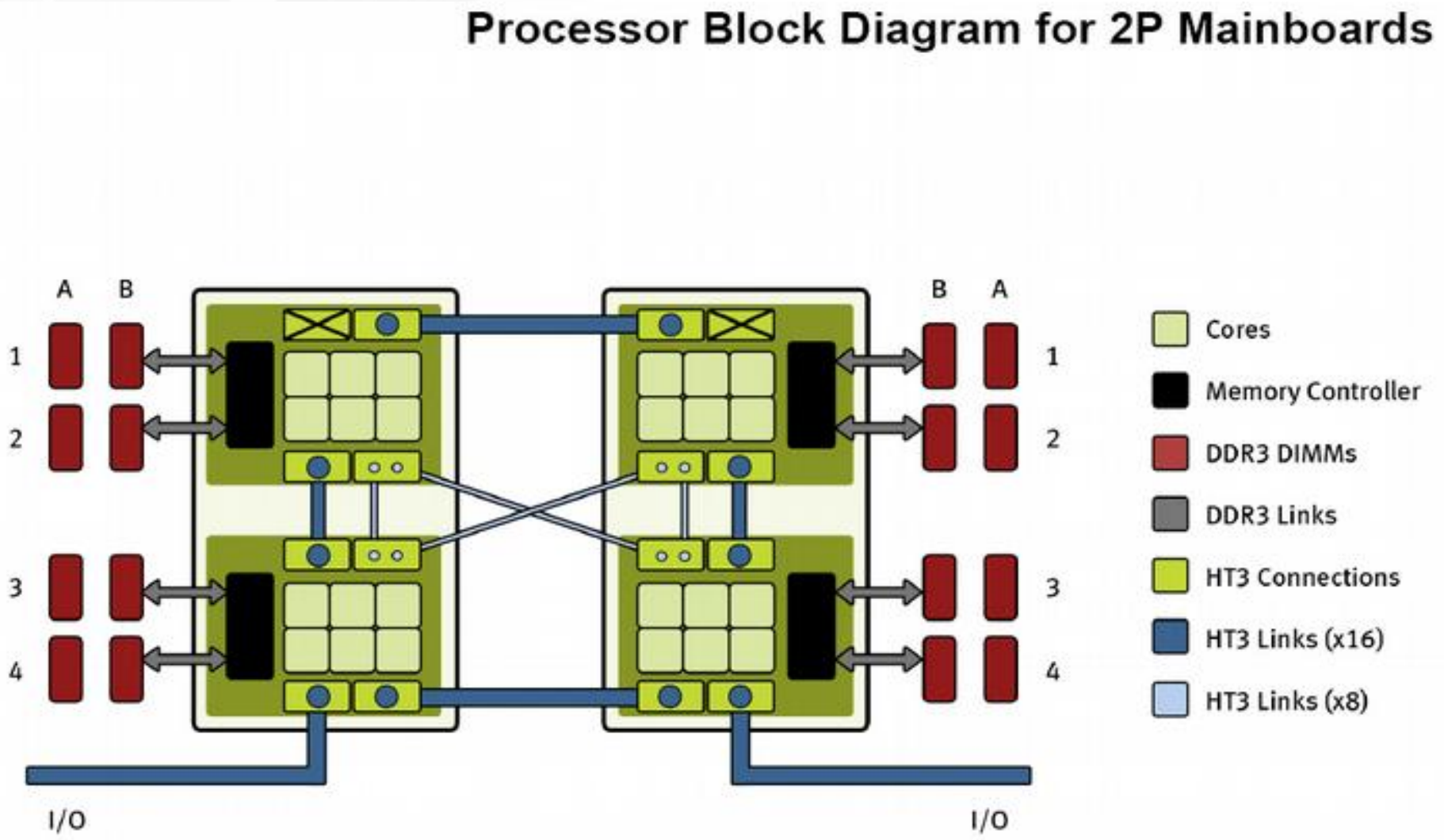
- Characteristics:
  - more than one CPU !
  - 64 bit address space



From SMP to NUMA

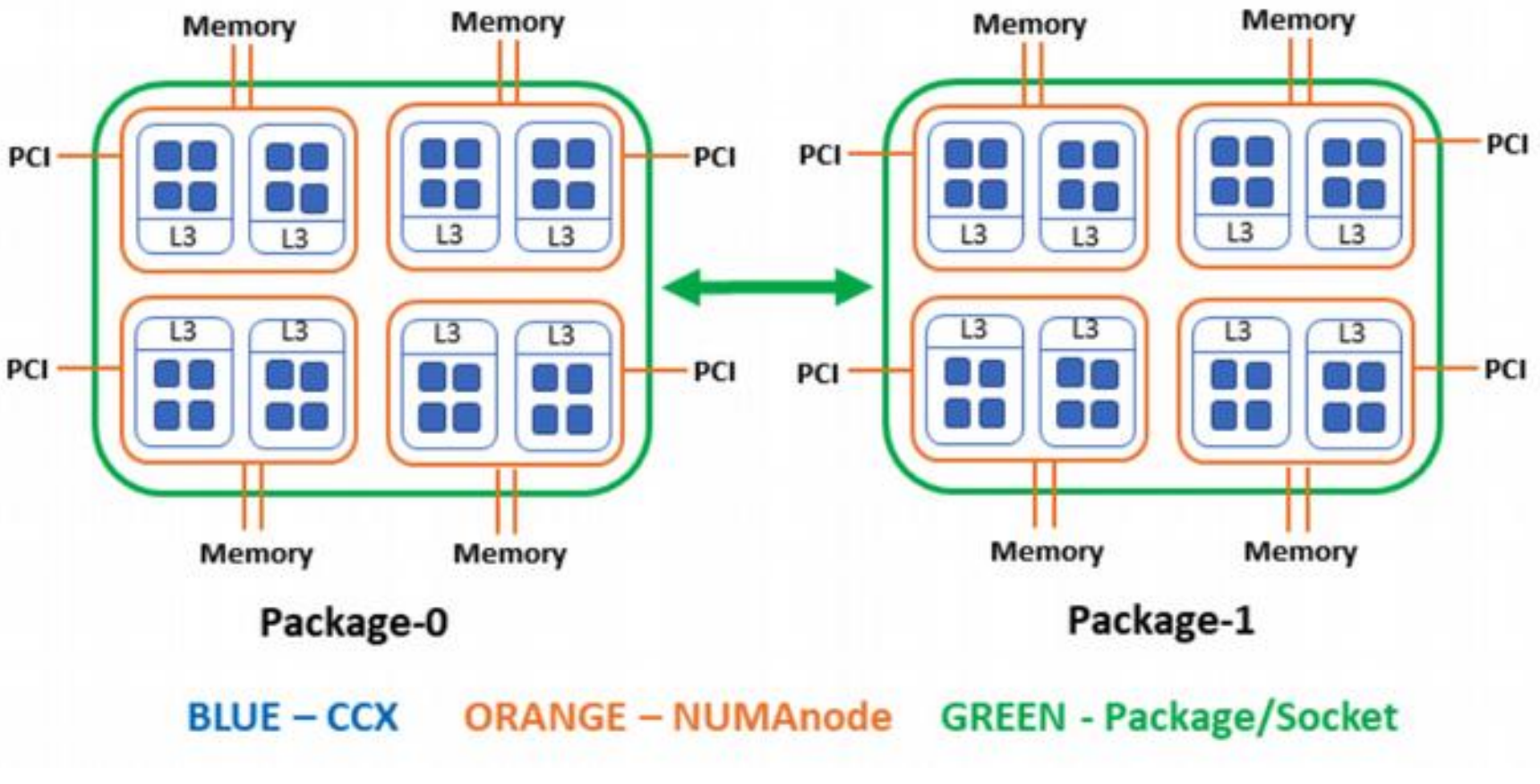
- FSB became rapidly a bottleneck: all the CPUs accessing memory throught it
- SMP (UMA) approach no longer possible
- First NUMA architecture:
  - Hypertransport technology by AMD (2005)
- Intel came much later
  - Quick Path Interconnect (2009)
  - Ultra Path Interconnect (2016 on)

Opteron 6xxx AMD CPU (elcid -2013)



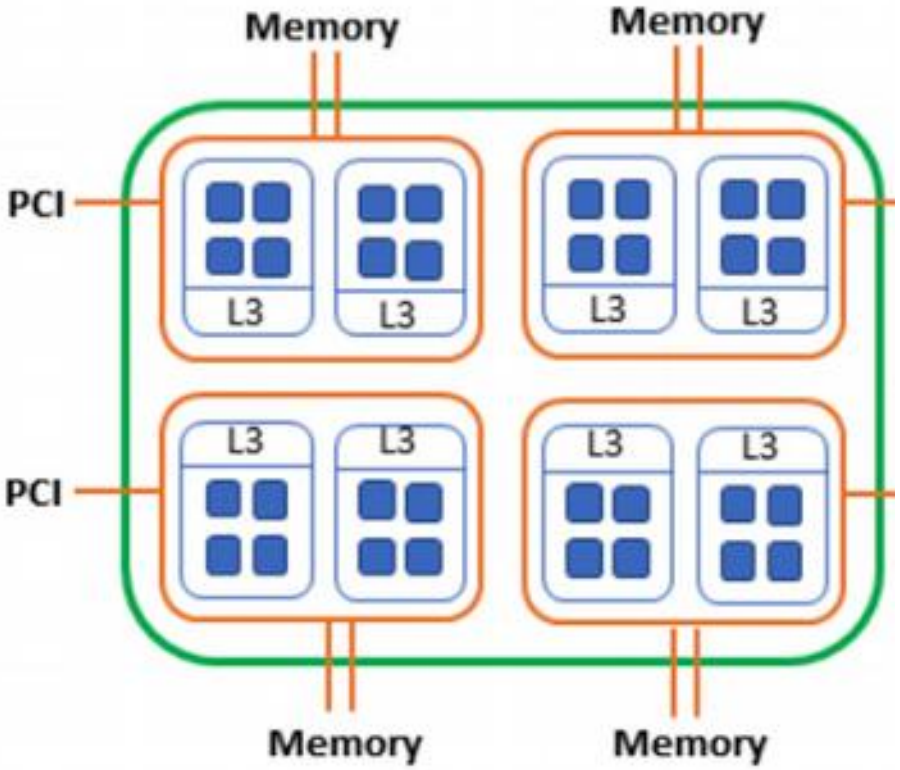
AMD Dual socket (2018)

- EPYC family
  - Each socket comprises 4 Numa nodes (aka Zeppelin)

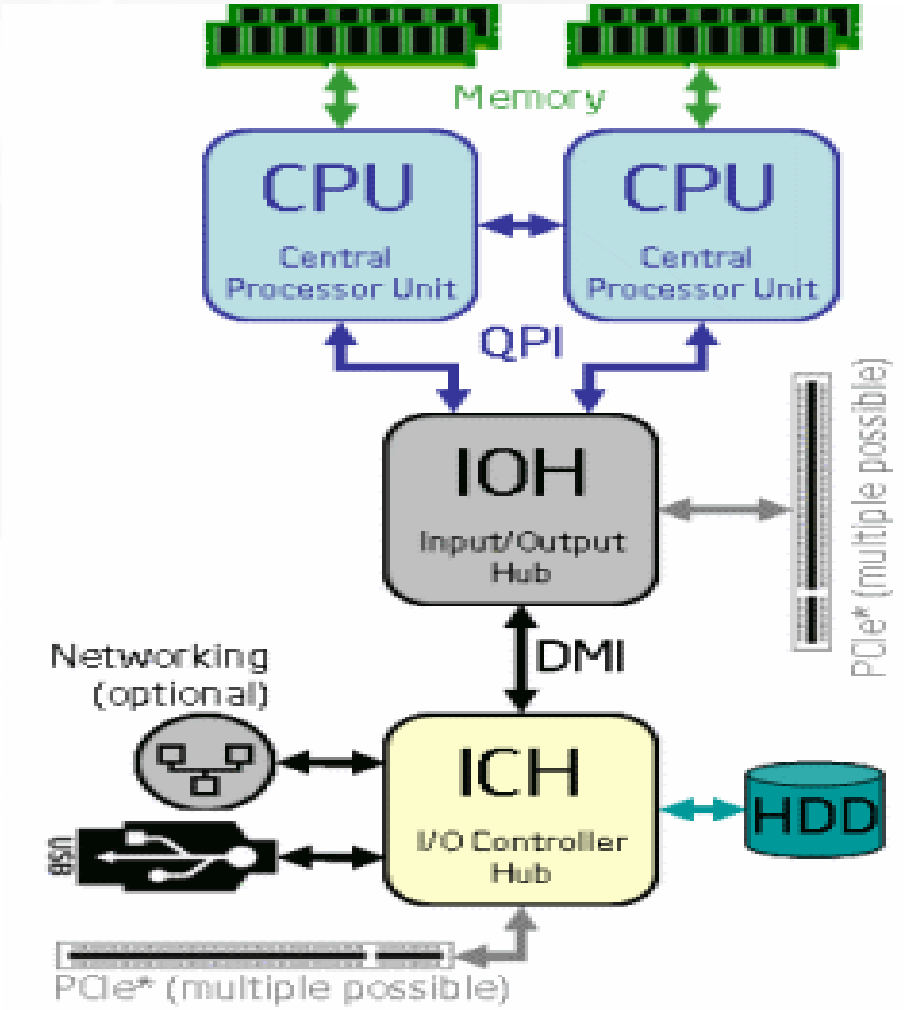


AMD Dual socket (2018)

- Each Zeppelin
  - provides 2 memory channels, i.e. 8 memory channels per socket
  - provides PCIe gen3 slots (each OEM individu decides how to present/consume these resources)
  - Is connected to the remaining 3 Zeppelins wi the socket via the internal Global Memory Interconnect, or "Infinity Fabric". This enable each NUMA node to access the memory and capabilities associated with its counterparts within the socket and between sockets
  - Within each Zeppelin there are 2 Compute Complexes (CCX). Each CCX has
    - its own L3 cache. Each core has its own L2 and L1i and L1d caches
    - up to 4 cores per CCX, i.e. 32 cores per socket.



Xeon Family: Nehalem (20090 introduces NUMA

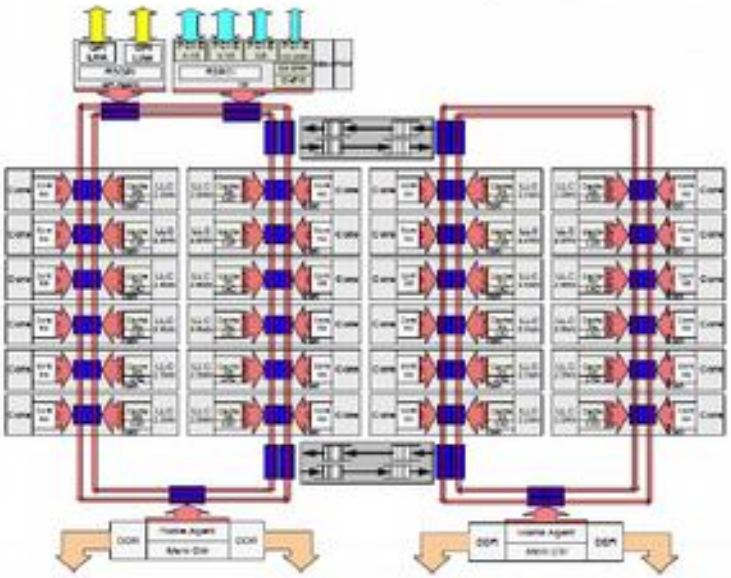


- First NUMA architecture by INTEL
- QPI among CPUs to play the role of hyper-transport in AMD
- Released April 2009

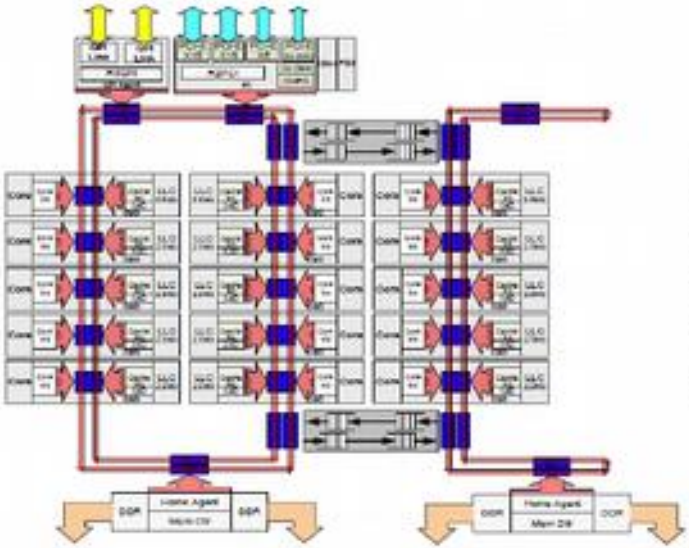
Brodwell layout (2016)

Broadwell EP die configurations

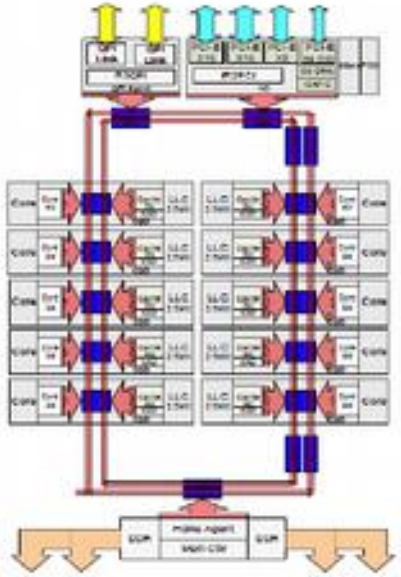
Up to 22 Core (HCC)



Up to 14 Core (MCC)



Up to 8 Core (LCC)



Chop	Columns	Home Agents	Cores	Power (W)
HCC	4	2	12-22	105-145
MCC	3	2	8-14	85-120
LCC	2	1	6-8	85

Should we know all this ?

It is important to understand this hierarchy. Doing so will aid will the developer in optimizing their application layout so as to choose the correct locality of memory and possibly L3 cache.

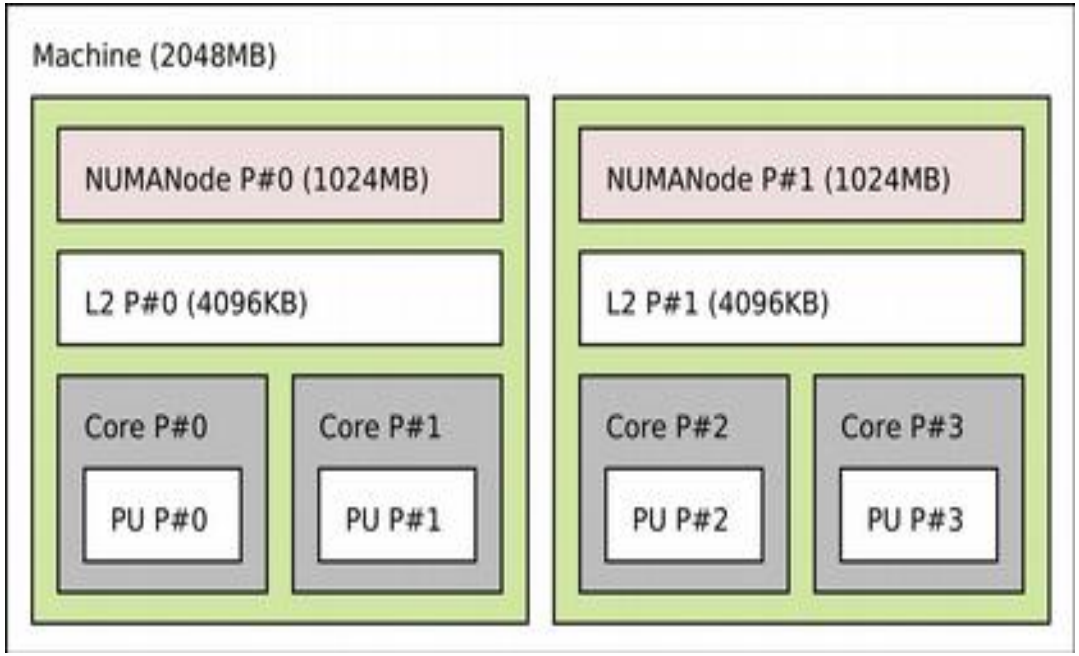
- Example:
  - given the choice would you launch a 4-core application within a single CCX or would you thread it across all 4 NUMA nodes within the socket? This is application-dependent and would be necessary to investigate.
  - For network connectivity: how do you allocate MPI processes ?

Our dilemma

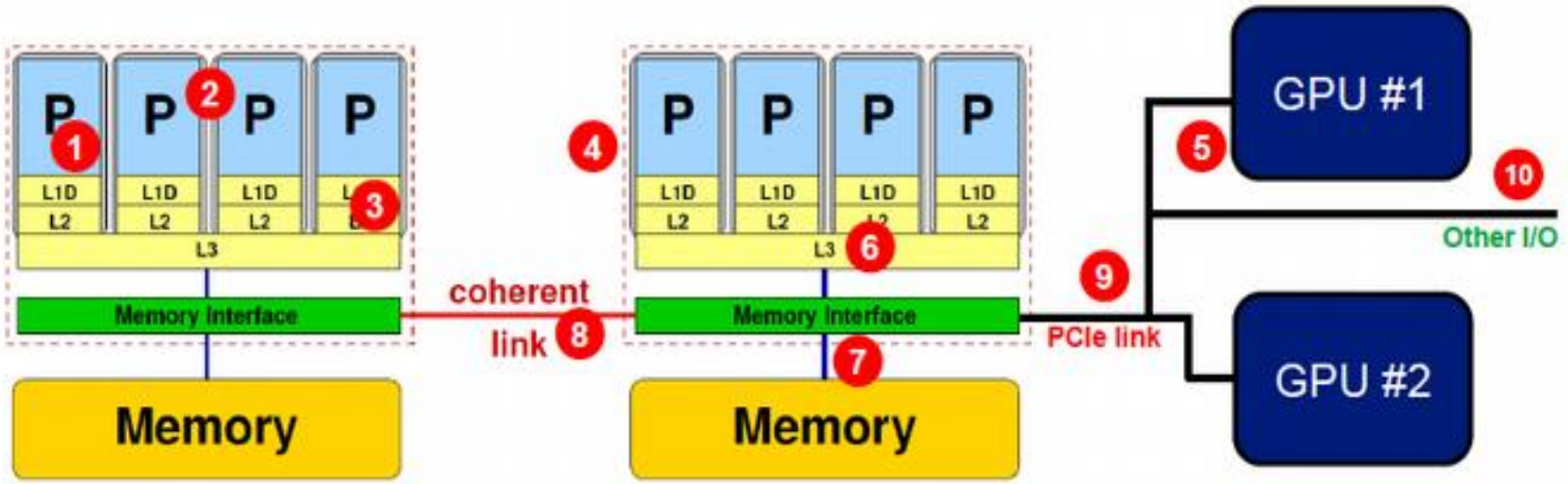
- Use cores 0 & 1 to share cache and improve synchronization cost?
- Use cores 0 & 2 to maximize memory bandwidth?
- How to choose portability?

Depends on

- the application structure
- machine structure



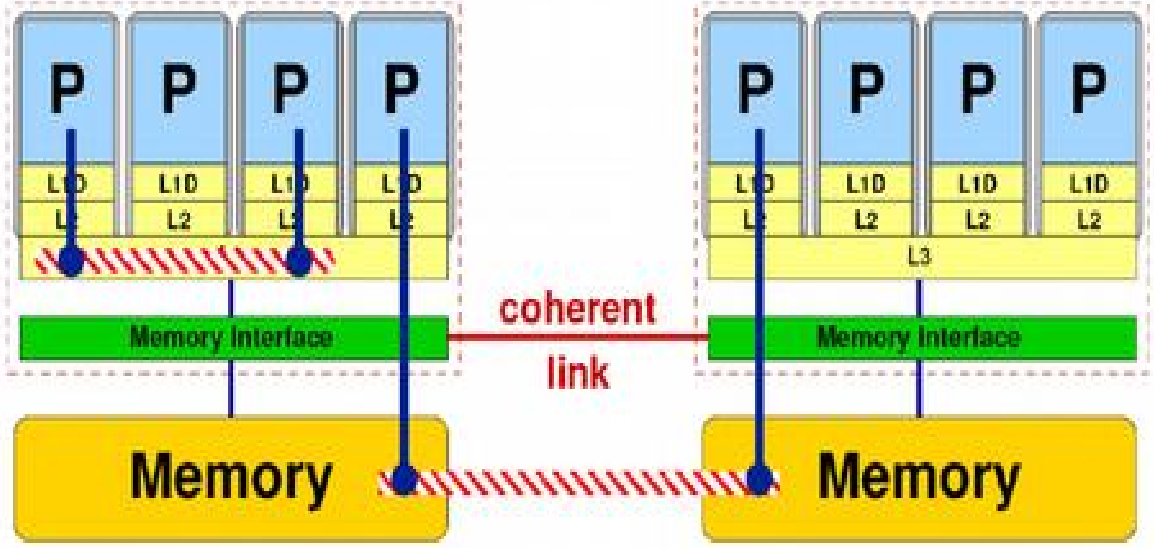
Parallellism and shared resource on shared-memory node

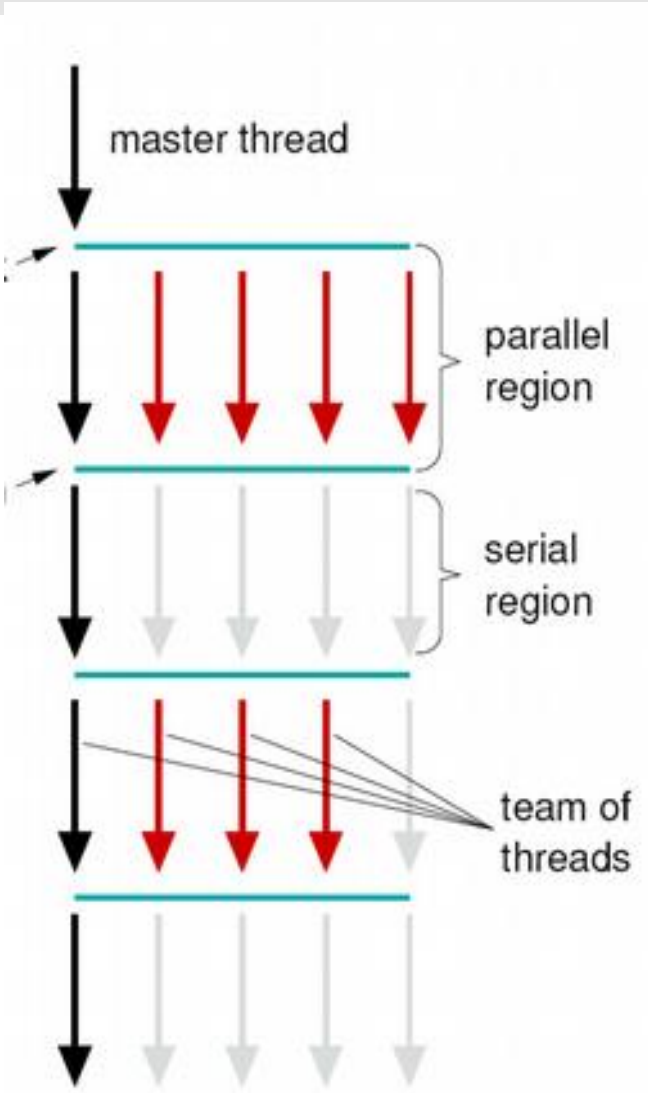


- Parallel resources
  - Execution/SIMD units (1)
  - Cores (2)
  - Inner cache levels (3)
  - Socket/ccNuma domains (4)
  - Multiple accelerator (5)
- Shared resources
  - Outer cache level per socket (6)
  - Memory bus per socket(7)
  - Intersocket link (8)
  - PCI-bus(es) (9)
  - Other I/O resources (10)

Parallel programming model : multithreaded

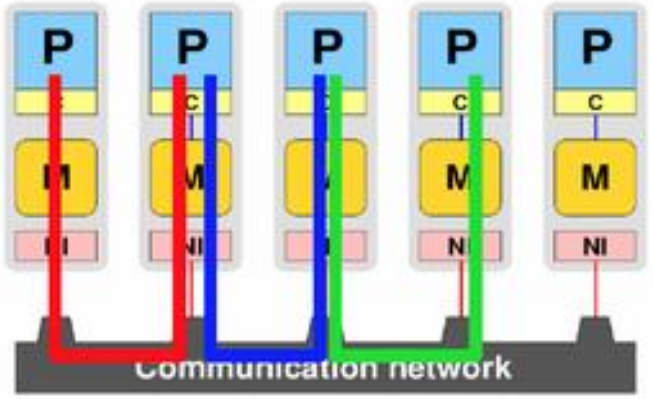
- Machine structure is invisible to user
  - Very simple programming model
  - Threading SW (OpenMP, pthreads, TBB,…) should know about the details
- Performance issues
  - Synchronization overhead
  - Memory access
  - Node/system topology

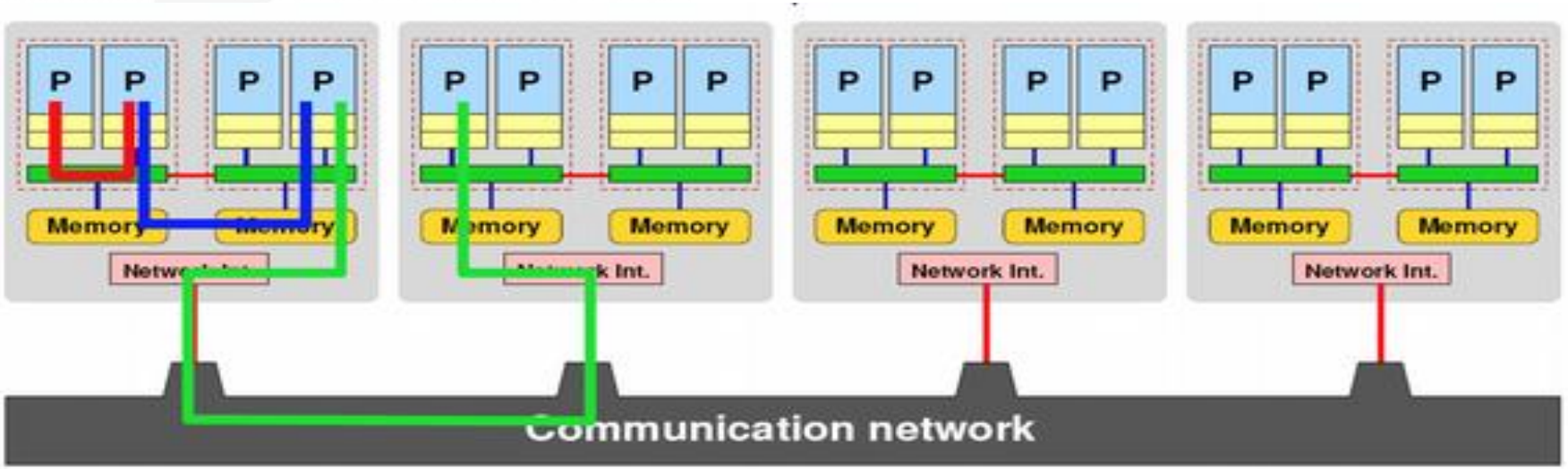




Parallel programming model : MPI

- Machine structure is invisible to user
  - Very simple programming model
  - MPI “knows what to do”!?
- Performance issues
  - Intranode vs. internode MPI
  - Node/system topology





Challenges for multicores

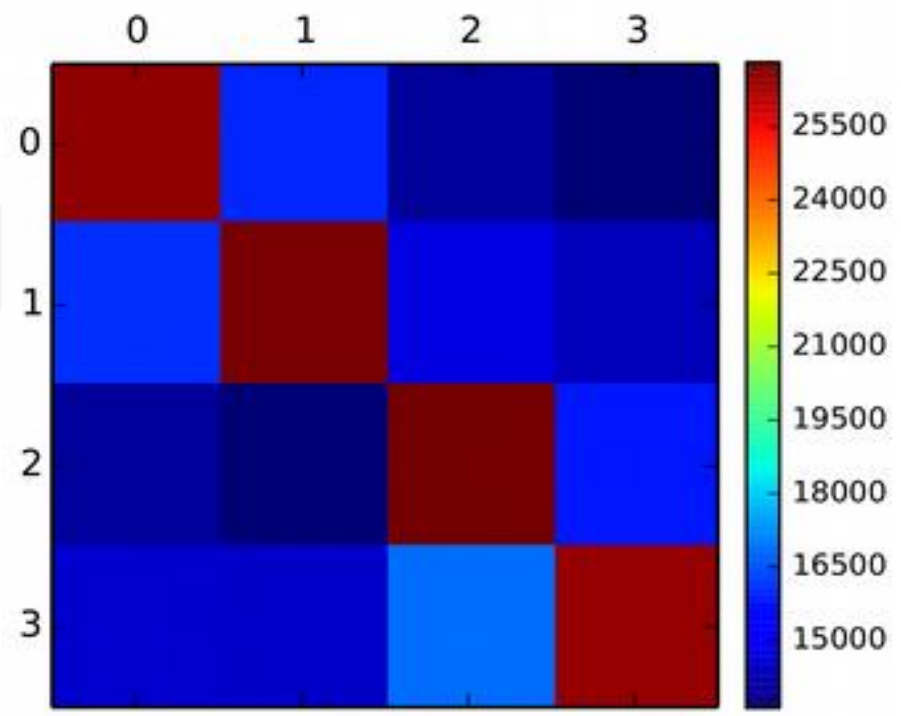
- Relies on effective exploitation of multiple-thread parallelism
  - Need for parallel computing model and parallel programming model
- Aggravates memory wall problem
  - Memory bandwidth
    - Way to get data out of memory banks
    - Way to get data into multi-core processor array
    - Memory latency
  - Cache sharing

Thread Affinity and Data Locality

- Affinity
  - Process Affinity: bind processes (MPI tasks, etc.) to CPUs
  - Thread Affinity: further binding threads to CPUs that are allocated to their parent process
- Data Locality
  - Memory Locality: allocate memory as close as possible to the core on which the task that requested the memory is running
  - Cache Locality: use data in cache as much as possible
- Correct process, thread and memory affinity is the basis for getting optimal performance.

An example: Sandybridge node

- CcNuma map: bandwidth for remote access
  - Run 10 threads per ccNUMA domain
  - Place memory in different domain



Thread/Processor Placement

How to keep threads on a particular core, so data is readily available when needed ?

- Many possible ways:
  - Numactl
  - Likwid-pin
  - hwloc
  - OpenMP (OMP_PLACES/ OMP_PROC_BIND/KMP_AFFINITY)

Numactl

- numactl is a command of the operating system providing much focused on the NUMA features of a system.
- numactl understands which processors form a NUMA node and how threads/processors need to be grouped together

Numactl on Ulysses:

```
[cozzini@cn02-03 ~]$ numactl --show
policy: default
preferred node: current
physcpubind: 0 1 2 3 4 5 6 7 8 9 10 11 12 13 14 15 16 17 18 19
cpubind: 0 1
nodebind: 0 1
membind: 0 1
```

```
[cozzini@cn02-03 ~]$ numactl --hardware
available: 2 nodes (0-1)
node 0 cpus: 0 1 2 3 4 5 6 7 8 9
node 0 size: 20451 MB
node 0 free: 18675 MB
node 1 cpus: 10 11 12 13 14 15 16 17 18 19
node 1 size: 20480 MB
node 1 free: 19538 MB
node distances:
node   0   1
  0:  10  11
  1:  11  10
```

Thread placement - numactl

Numactl

--membind <n>: place pages on NUMA node <n>

--cpunodebind <n>: pin threads to node <n>

--interleave <nodes>: put the pages round-robin on <nodes>

Example:

```
numactl --cpunodebind=0 --membind=0,1 ./a.out
```

This puts memory on nodes 0 an 1, but threads only on node 0.

Numactl as simple ccNuma locality tool:

- Numactl can influence the way a code maps the memory pages
  - `Numactl –membind=<nodes> a.out # map page on # <node>`
  - `--preferred=<nodes> a.out # map pages on # <node> and the # when it is full # on others -- interleaved= <nodes> a.out`
- Example:

ccNUMA map scan

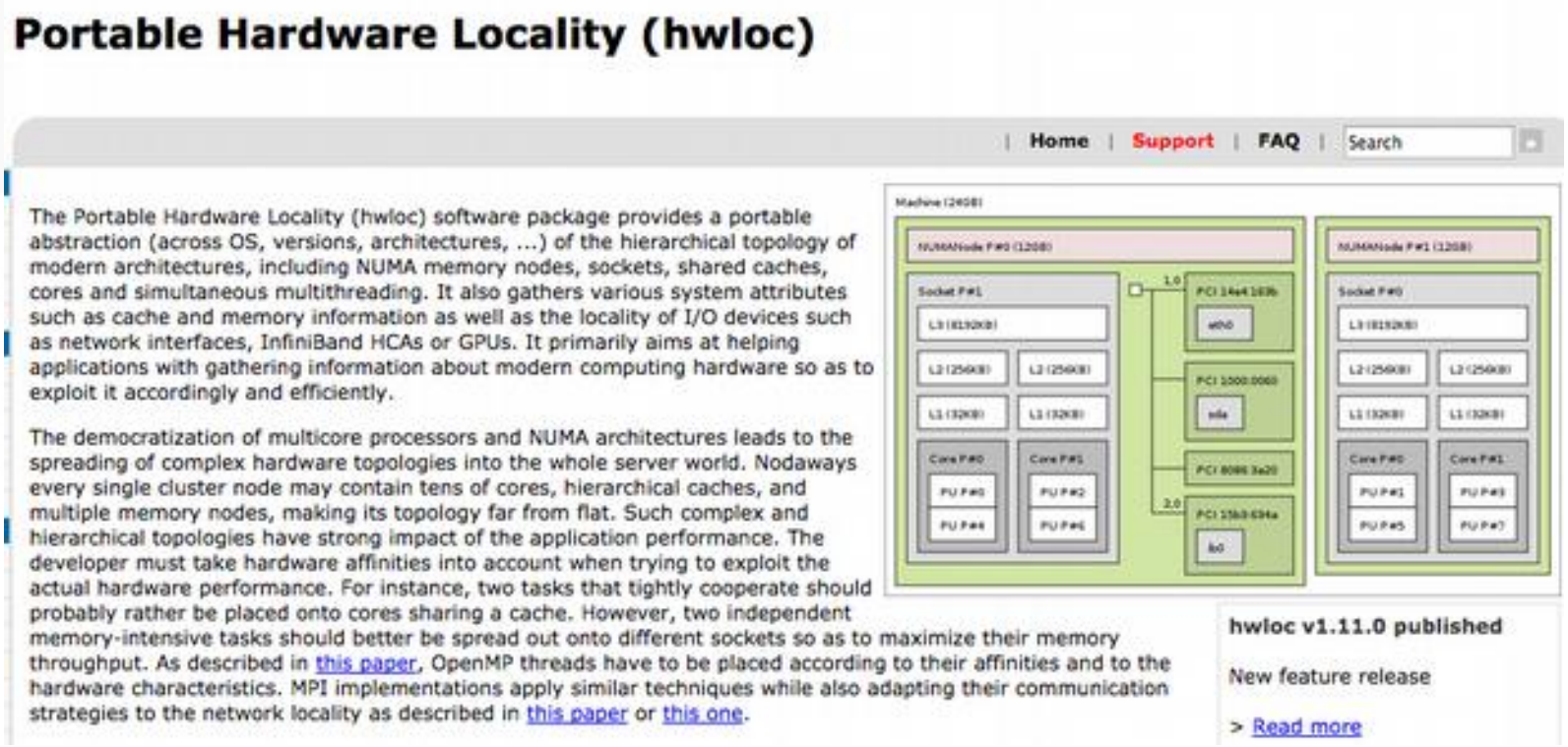
```
for m in `seq 0 3`; do
  for c in `seq 0 3`; do
    env OMP_NUM_THREADS=8 \
        numactl --membind=$m --cpunodebind=$c ./stream
  enddo
enddo
```

hwloc

Portable Hardware Locality

Portable topology information

Portable binding toolset



hwloc

- Two parts
  - - Set of command line tools (lstopo, hwloc-bind, calc, etc.)
  - - C API + library, Perl and Python bindings
- Portable: Linux, Solaris, AIX, HP-UX, FreeBSD, Darwin, Windows
- BSD-3 license
- Used by a lot of projects: most MPI, runtimes, batch scheds, ...

http://www.open-mpi.org/projects/hwloc/

Let us using hwloc

`lstopo` – Displaying topology information

`hwloc-distances` – show object distances

Notably NUMA distances:

```
$ ./utils/hwloc-distances
```

Binding processes and memory

```
hwloc-bind – bind process
```

Bind a new process to a given set of CPUs:

```
$ hwloc-bind socket:1 -- mycommand
```

Bind an existing process:

```
$ hwloc-bind --pid 1234 socket:1
```

Bind memory:

```
$ hwloc-bind --membind node:1 --cpubind
node:1.socket:0  ./a.out
```

Distribute memory:

```
$ hwloc-bind --membind --mempolicy interleave all
-- mycommand
```

Evaluating memory performance

- Peak performance
- Stream benchmark

Main Memory bandwidth

Throughput = memory bus frequency * bits per cycle * bus width

Memory clock != CPU clock

In bits !! divide by 8 for GB/s

Examples:

- Intel Core i7 DDR3: 1.333 * 2 * 64 = 21 GB/s
- NVIDIA GTX 580 GDDR5: 1.002 * 4 * 384 = 192 GB/s

Memory bandwidths

- On-chip memory can be orders of magnitude faster
  - Registers, shared memory, caches, …
- Other memories: depends on the interconnect
  - Intel's technology: QPI/UPI(Quick Path Interconnect)
    - 25.6 GB/s
  - AMD's technology: HT3 (Hyper Transport 3)
    - 19.2 GB/s
  - Accelerators: PCI-e 2.0
    - 8/16 GB/s

How fast are memories ?

- Synchronous dynamic random-access memory (SDRAM)
- Double Data Rate (DDR) with ECC
- DDR ->DDR2->DDR3-->DDR4

DDR SDRAM Standard	Internal rate (MHz)	Bus clock (MHz)	Prefetch	Data rate (MT/s)	Transfer rate (GB/s)	Voltage (V)
SDRAM	100-166	100-166	1n	100-166	0.8-1.3	3.3
DDR	133-200	133-200	2n	266-400	2.1-3.2	2.5/2.6
DDR2	133-200	266-400	4n	533-800	4.2-6.4	1.8
DDR3	133-200	533-800	8n	1066-1600	8.5-14.9	1.35/1.5
DDR4	133-200	1066-1600	8n	2133-3200	17-21.3	1.2

Memory Technologies

- SRAM (Static Random Access Memory)
  - Simple integrated circuit with one single access port
  - On Chip memory (caches)
  - Access times :0.5- 2.5 nanosec
  - Cost (2012): $500-1000 GiB
- DRAM
  - Memory stored in capacitors – need to be refreshed
  - One transistor per bit – much cheaper than SRAM
  - SDRAM (synchronous DRAM). Uses clocks. Transfer data in bursts.
  - DDR (Double Data Rate) SDRAM. Transfer data both on rising and falling clock edge.
  - Access time: 50-70nanosec,
  - Cost (2012): $10-$20 GiB

Source: Patterson and Hennessy, 2012

Recent architecture: latency..

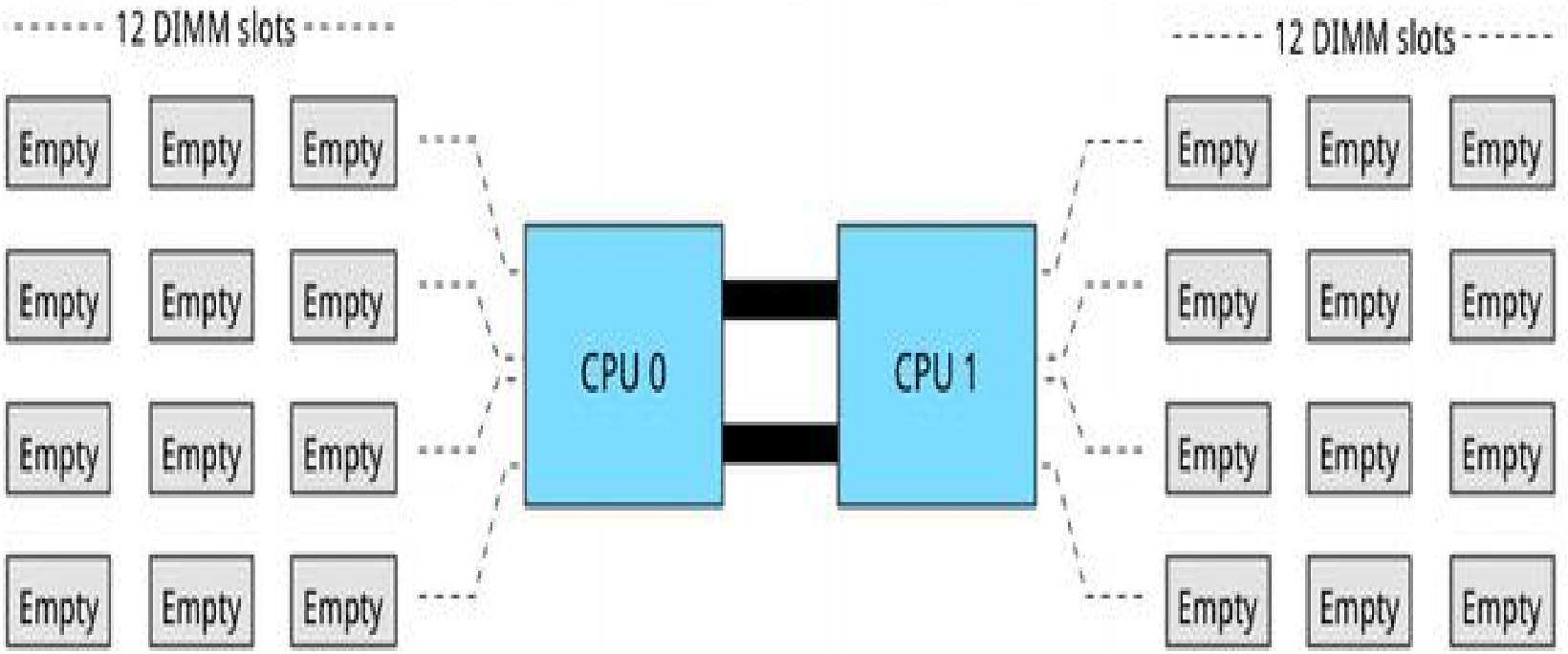
The numbers we looked at were "Random load latency stride=16 Bytes" (LMBench).

Mem Hierarchy	IBM POWER8	Intel Broadwell Xeon E5-2640v4 DDR4-2133	Intel Broadwell Xeon E5-2699v4 DDR4-2400
L1 Cache (cycles)	3	4	4
L2 Cache (cycles)	13	12-15	12-15
L3 Cache 4-8 MB(cycles)	27-28 (8 ns)	49-50	50
16 MB (ns)	55 ns	26 ns	21 ns
32-64 MB (ns)	55-57 ns	75-92 ns	80-96 ns
Memory 96-128 MB (ns)	67-74 ns	90-91 ns	96 ns
Memory 384-512 MB (ns)	89-91 ns	91-93 ns	95 ns

How is memory organized ?

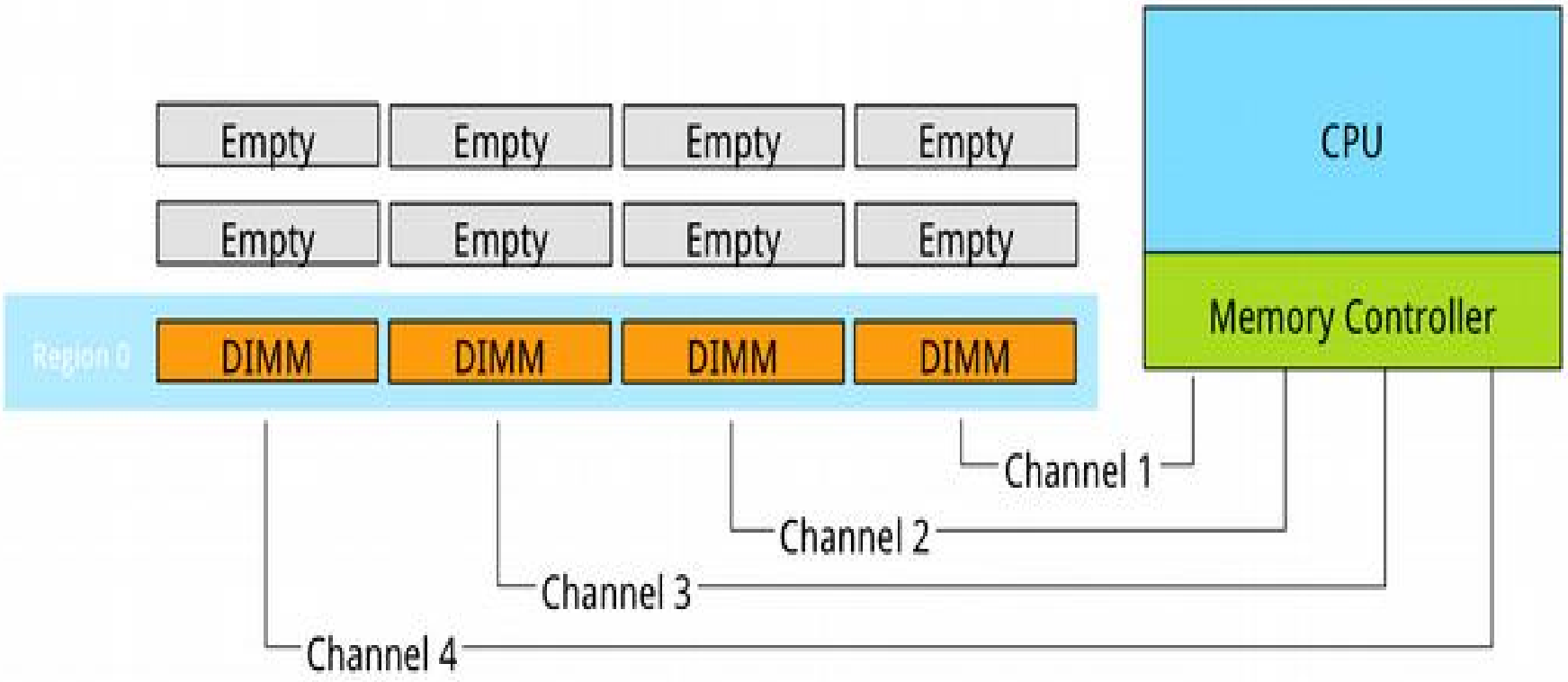
```
[root@b11 ~]# dmidecode --type memory | grep
BRANCH
Bank Locator: BRANCH 0 CHANNEL 0 DIMM 0
Bank Locator: BRANCH 0 CHANNEL 0 DIMM 1
Bank Locator: BRANCH 0 CHANNEL 0 DIMM 2
Bank Locator: BRANCH 0 CHANNEL 1 DIMM 0
Bank Locator: BRANCH 0 CHANNEL 1 DIMM 1
Bank Locator: BRANCH 0 CHANNEL 1 DIMM 2
Bank Locator: BRANCH 0 CHANNEL 2 DIMM 0
Bank Locator: BRANCH 0 CHANNEL 2 DIMM 1
Bank Locator: BRANCH 0 CHANNEL 2 DIMM 2
Bank Locator: BRANCH 0 CHANNEL 3 DIMM 0
Bank Locator: BRANCH 0 CHANNEL 3 DIMM 1
Bank Locator: BRANCH 0 CHANNEL 3 DIMM 2
...
```

The official layout for IvyBridge



The data layout on C3HPC nodes

E5 v2 processor



The data layout on Skylake architecture

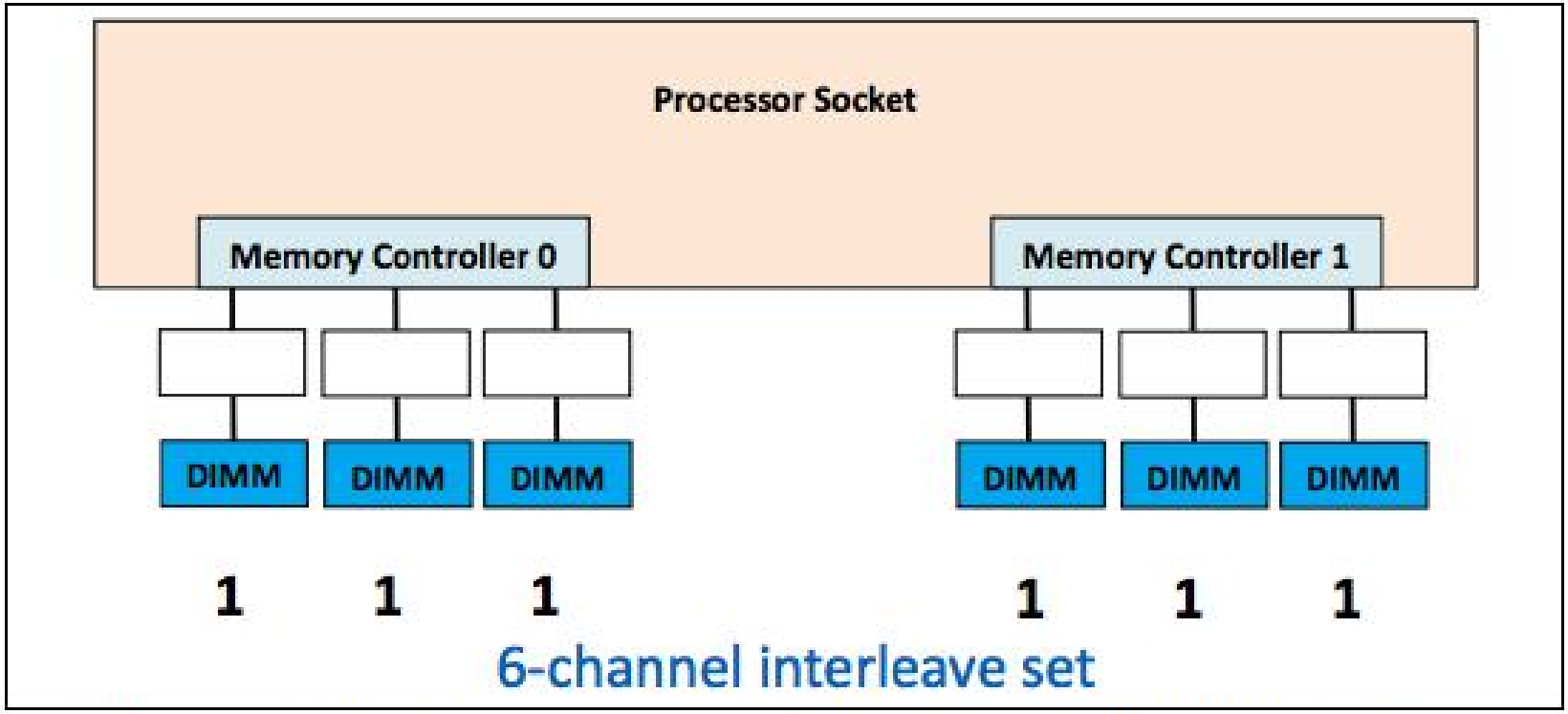


Figure 11 1:1:1,1:1:1 memory configuration (STREAM Triad relative memory bandwidth = 97%)

Memory layout

```
# dmidecode --type memory | grep "Locator" | grep -v Bank
   Locator: A1
   Locator: A2
   Locator: A3
   Locator: A4
   Locator: A5
   Locator: A6
   Locator: A7
   Locator: A8
   Locator: A9
   Locator: A10
   Locator: A11
   Locator: A12
   Locator: B1
   Locator: B2
   Locator: B3
   Locator: B4
   Locator: B5
   Locator: B6
   Locator: B7
   Locator: B8
   Locator: B9
   Locator: B10
   Locator: B11
   Locator: B12
```

```
Handle 0x1000, DMI type 16, 23 bytes
Physical Memory Array
    Location: System Board Or Motherboard
    Use: System Memory
    Error Correction Type: Multi-bit ECC
    Maximum Capacity: 7680 GB
    Error Information Handle: Not Provided
    Number Of Devices: 24

Handle 0x1100, DMI type 17, 84 bytes
Memory Device
    Array Handle: 0x1000
    Error Information Handle: Not Provided
    Total Width: 72 bits
    Data Width: 64 bits
    Size: 64 GB
    Form Factor: DIMM
    Set: 1
    Locator: A1
    Bank Locator: Not Specified
    Type: DDR4
    Type Detail: Synchronous LRDIMM
    Speed: 2666 MT/s
    Manufacturer: 00CE00B300CE
    Serial Number: 246FBB42
    Asset Tag: 03185051
    Part Number: M386A8K40BM2-CTD
    Rank: 4
    Configured Clock Speed: 2666 MT/s
    Minimum Voltage: 1.2 V
    Maximum Voltage: 1.2 V
    Configured Voltage: 1.2 V
```

STREAM benchmark

- Created in 1991
- Intended to be a oversimplified representation of low-computing intensity and long vector operations
- Widely used, more 1100 results in the database
- Hosted at www.cs.virginia.edu/stream
- Compilation and execution trivial

STREAM benchmark

- Four kernels, separately timed:

```
Copy:  C[i] = A[i];                 16 Bytes
Scale: B[i] = scalar * C[i];        16 Bytes
Add:   C[i] = A[i] + B[i];          24 Bytes
Triad  A[i] = B[i] + scalar*C[i];   24 Bytes
```

- N chosen to make each array >> cache size
- Repeated several times, first iteration ignored
- Min/Max/Avg reported and the best time used to compute Bandwidth

Stream at work:

```
-------------------------------------------------------------
STREAM version $Revision: 5.10 $
-------------------------------------------------------------
This system uses 8 bytes per array element.
-------------------------------------------------------------
Array size = 120000000 (elements), Offset = 0 (elements)
Memory per array = 915.5 MiB (= 0.9 GiB).
Total memory required = 2746.6 MiB (= 2.7 GiB).
Each kernel will be executed 20 times.
 The *best* time for each kernel (excluding the first iteration)
 will be used to compute the reported bandwidth.
-------------------------------------------------------------
Number of Threads requested = 40
Number of Threads counted = 40
-------------------------------------------------------------
Your clock granularity/precision appears to be 1 microseconds.
Each test below will take on the order of 101043 microseconds.
   (= 101043 clock ticks)
Increase the size of the arrays if this shows that
you are not getting at least 20 clock ticks per test.
-------------------------------------------------------------
WARNING -- The above is only a rough guideline.
For best results, please be sure you know the
precision of your system timer.
-------------------------------------------------------------
Function    Best Rate MB/s  Avg time     Min time     Max time
Copy:           17007.8     0.116517     0.112889     0.123725
Scale:          17055.9     0.120043     0.112571     0.130133
Add:            20137.6     0.148766     0.143016     0.159369
Triad:          20329.5     0.147540     0.141666     0.152015
-------------------------------------------------------------
Solution Validates: avg error less than 1.000000e-13 on all three arrays
-------------------------------------------------------------
```

STREAM results on Ulysses