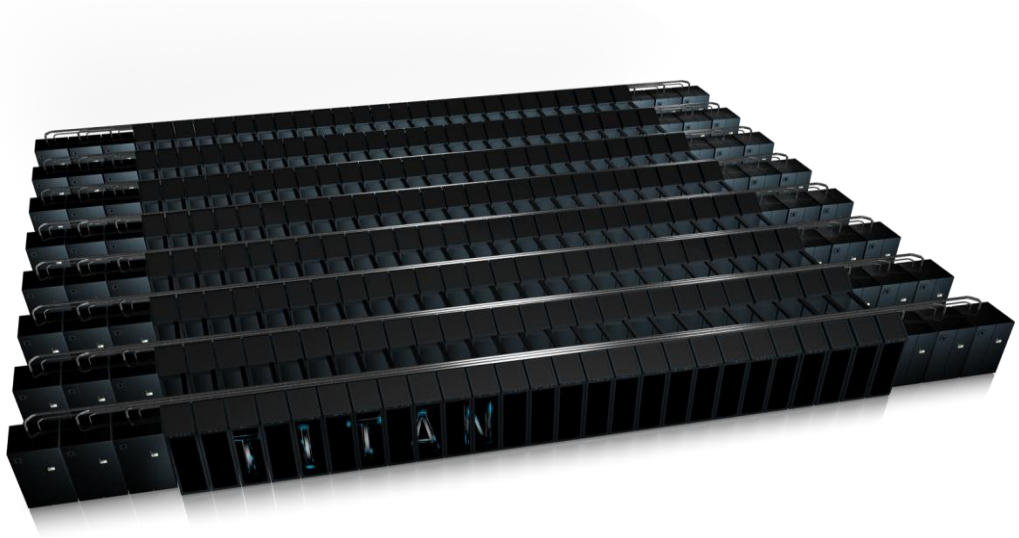
Titan Architecture

Jeff Larkin

OLCF
Oak Ridge Leadership Computing Facility



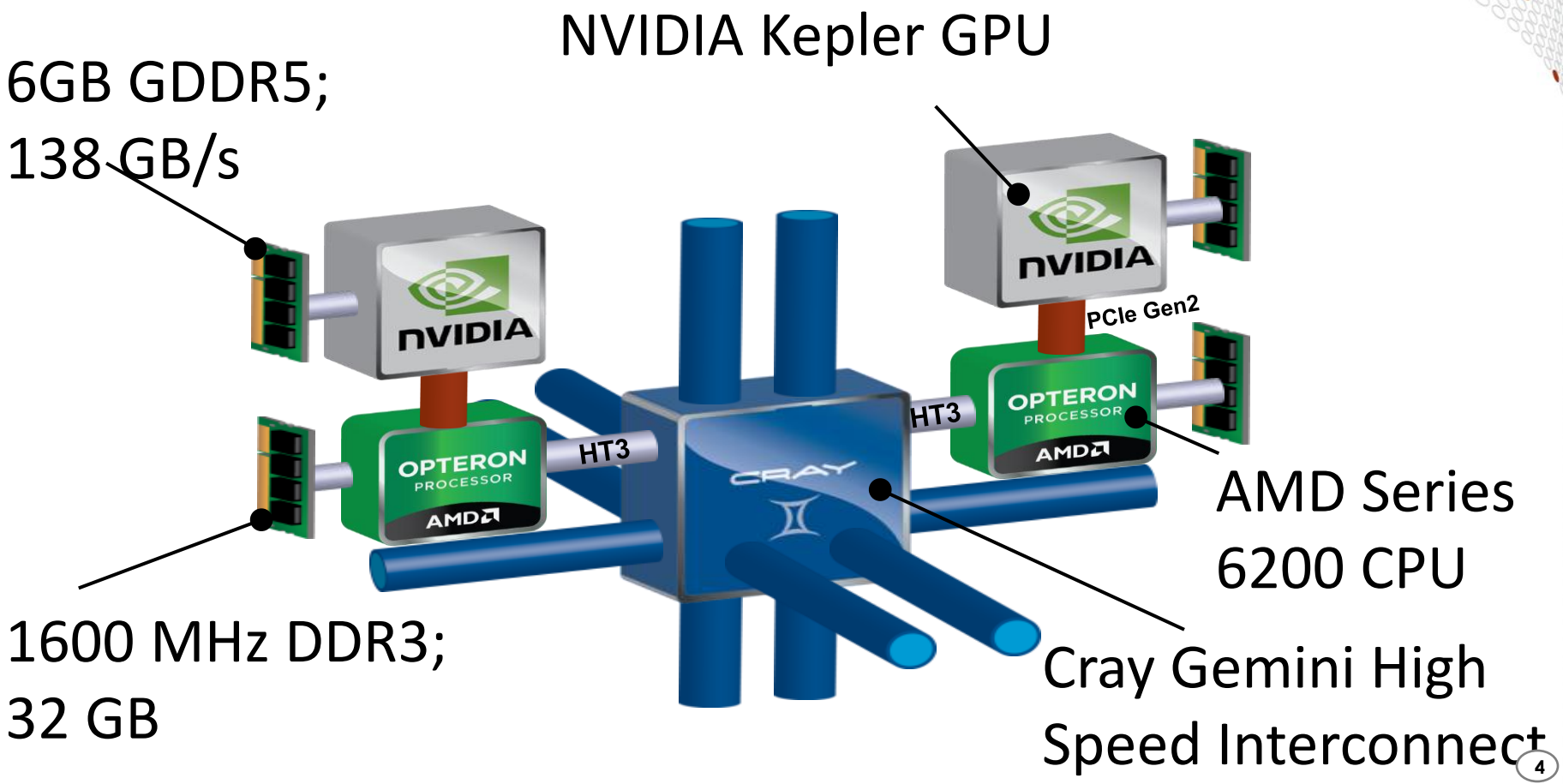
Titan Configuration

Name	Titan
Architecture	XK7
Processor	AMD Interlagos
Cabinets	200
Nodes	18,688
CPU Memory/Node	32 GB
GPU Memory/Node	6 GB
Interconnect	Gemini
GPUs	Nvidia Kepler

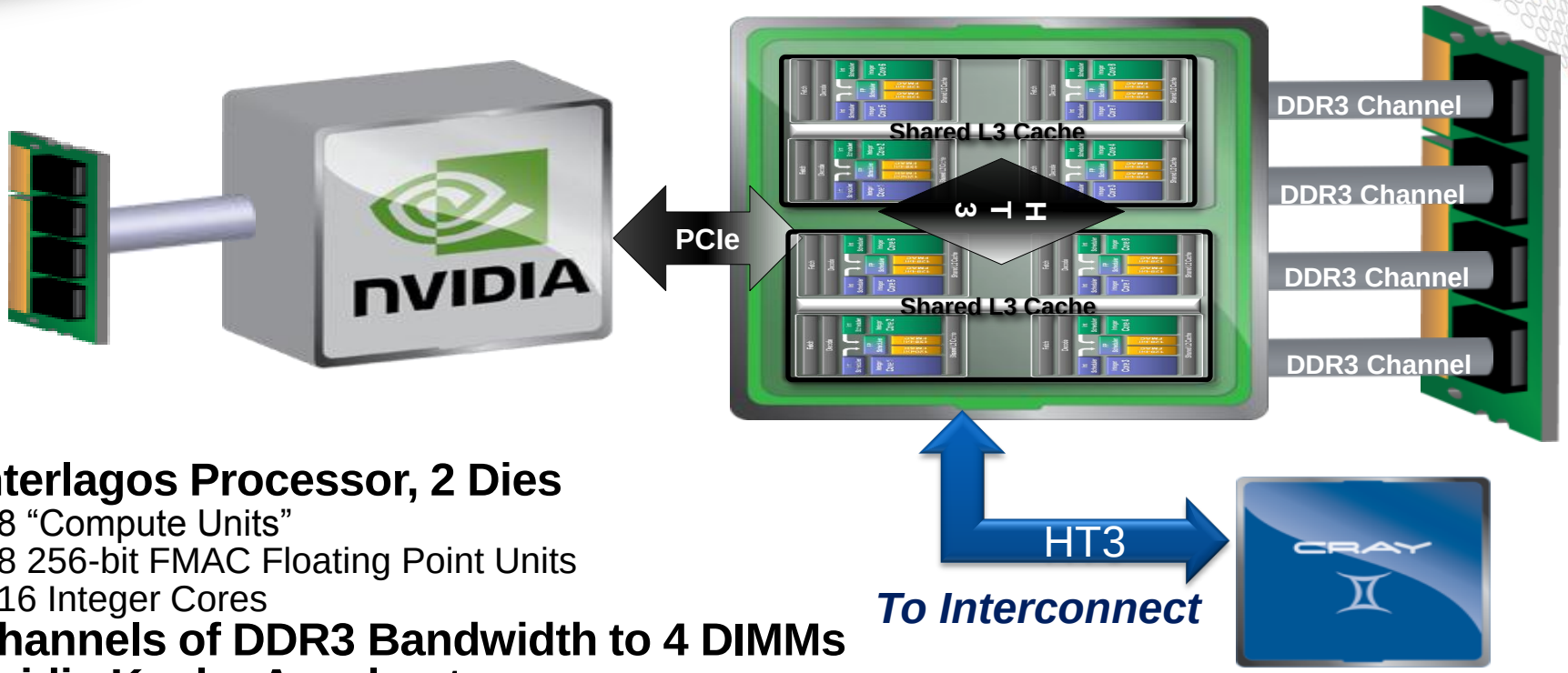
Cray XK7 Architecture

AMD Interlagos Processor
Cray Gemini Interconnect
Lustre Filesystem Basics
Nvidia Kepler Accelerator

Cray XK7 Architecture



XK7 Node Details



- **1 Interlagos Processor, 2 Dies**
  - 8 “Compute Units”
  - 8 256-bit FMAC Floating Point Units
  - 16 Integer Cores
- **4 Channels of DDR3 Bandwidth to 4 DIMMs**
- **1 Nvidia Kepler Accelerator**
  - Connected via PCIe Gen 2

AMD Interlagos Processor

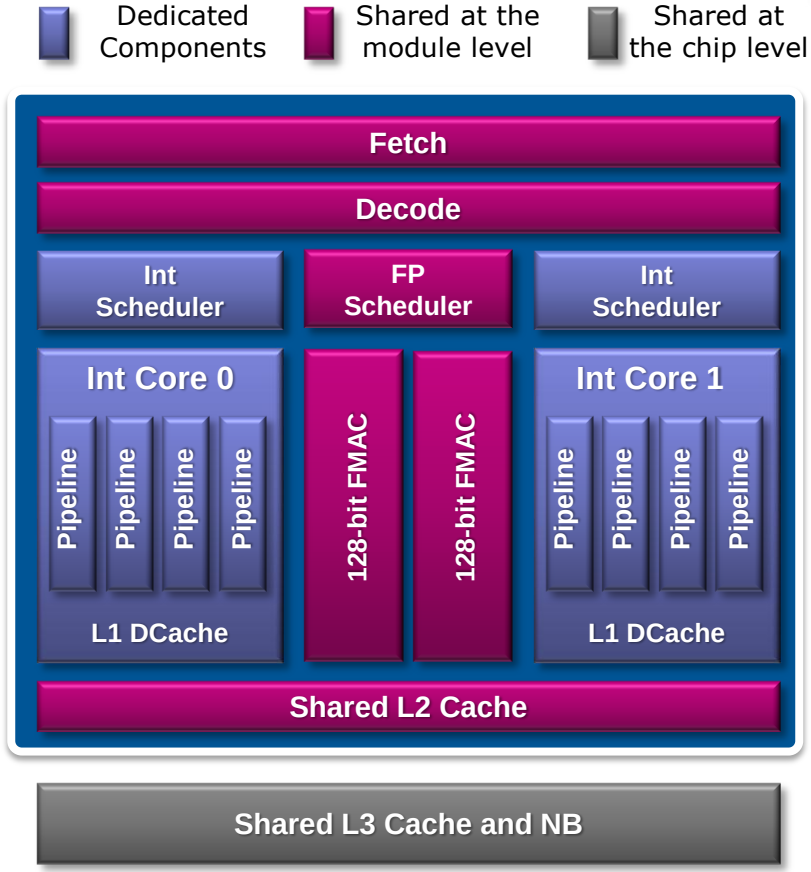
Interlagos Core Definition

- **In order to optimize the utilization of the shared and dedicated resources on the chip for different types of applications, modern x86 processors offer flexible options for running applications. As a result, the definition of a *core* has become ambiguous.**

- **Definition of a Core from Blue Waters proposal:**
  - Equivalent to an AMD “Interlagos” Compute Unit, which is an AMD Interlagos “Bulldozer module” consisting of: one instruction fetch/decode unit, one floating point scheduler with two FMAC execution units, two integer schedulers with multiple pipelines and L1 Dcache, and a L2 cache. This is sometimes also called a “Core Module.” A “core” = “compute unit” = “core module.”

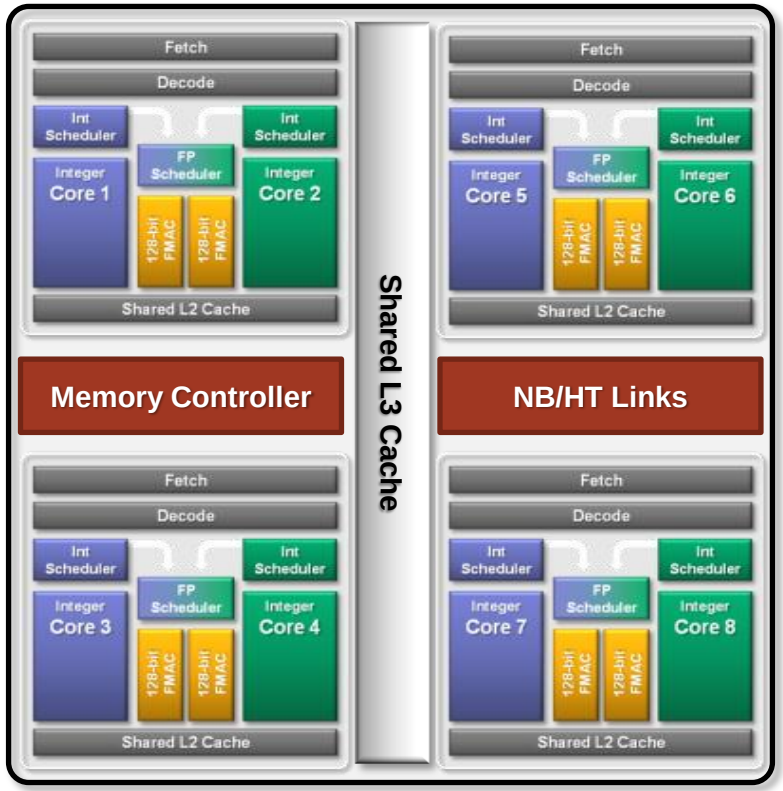
Interlagos Processor Architecture

- **Interlagos is composed of a number of "Bulldozer modules" or "Compute Unit"**
  - A compute unit has shared and dedicated components
    - There are two independent integer units; shared L2 cache, instruction fetch, Icache; and a *shared*, 256-bit Floating Point resource
  - A single Integer unit can make use of the entire Floating Point resource with 256-bit AVX instructions
    - Vector Length
      - 32 bit operands, VL = 8
      - 64 bit operands, VL = 4



Building an Interlagos Processor

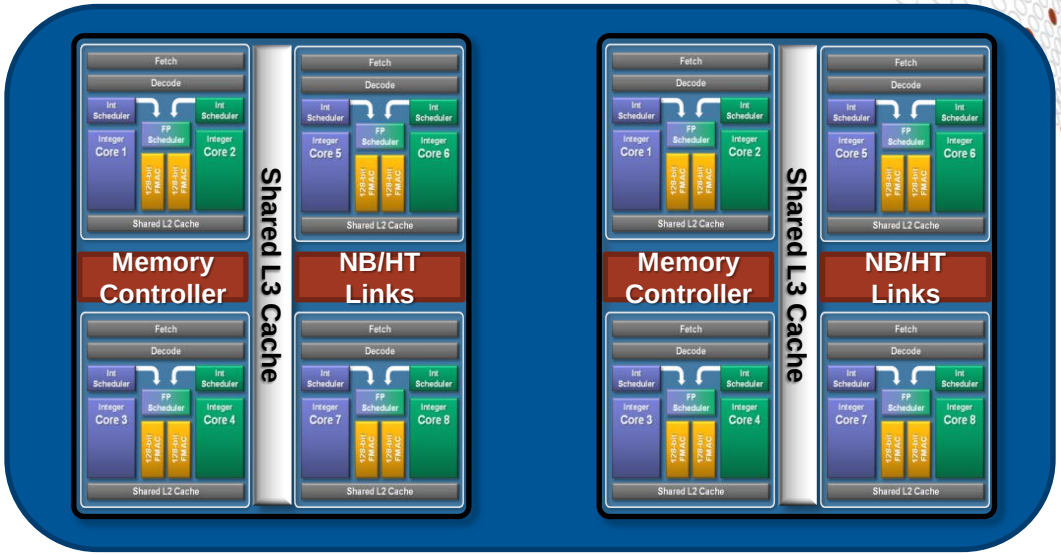
- **Each processor die is composed of 4 compute units**
  - The 4 compute units share a memory controller and 8MB L3 data cache
    - Each processor die is configured with two DDR3 memory channels and multiple HT3 links

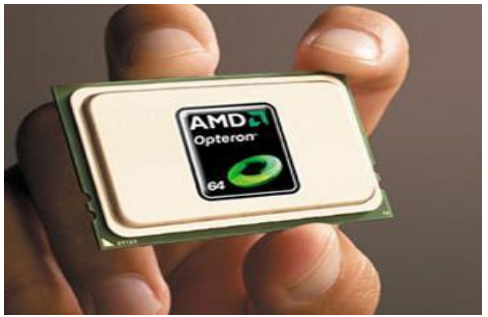


Interlagos Die Floorplan

Interlagos Processor

- **Two die are packaged on a multi-chip module to form an Interlagos processor**
  - Processor socket is called G34 and is compatible with Magny Cours
  - Package contains
    - 8 compute units
    - 16 MB L3 Cache
    - 4 DDR3 1333 or 1600 memory channels



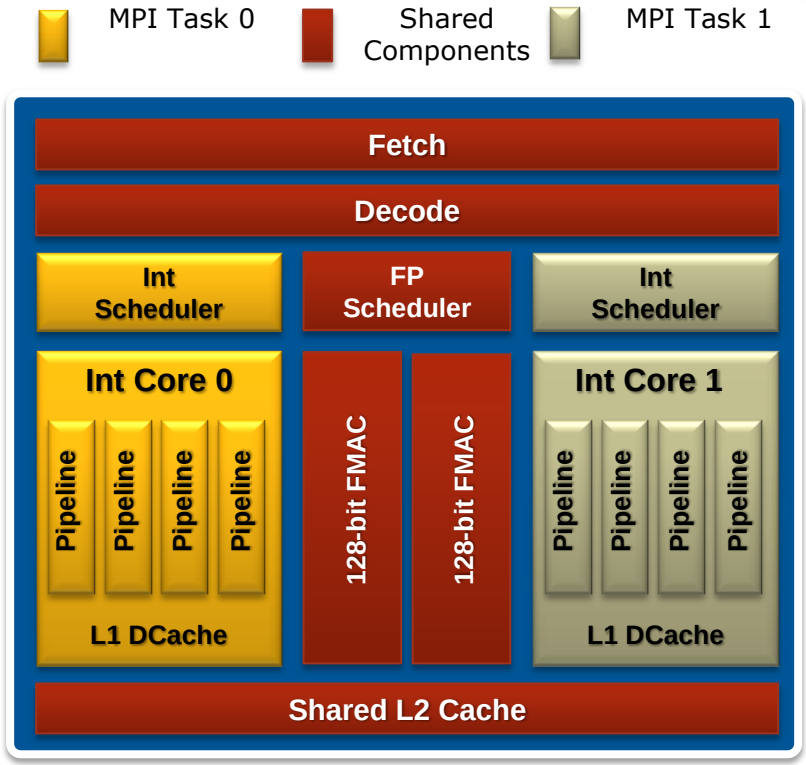


Interlagos Caches and Memory

- **L1 Cache**
  - 16 KB, 4-way predicted, parity protected
  - Write-through and inclusive with respect to L2
  - 4 cycle load to use latency
- **L2 Cache**
  - 2MB, Shared within core-module
  - 18-20 cycle load to use latency
- **L3 Cache**
  - 8 MB, non-inclusive victim cache (mostly exclusive)
    - Entries used by multiple core-modules will remain in cache
    - 1 to 2 MB used by probe filter (snoop bus)
    - 4 sub-caches, one close to each compute module
    - Minimum Load to latency of 55-60 cycles
- **Minimum latency to memory is 90-100 cycles**

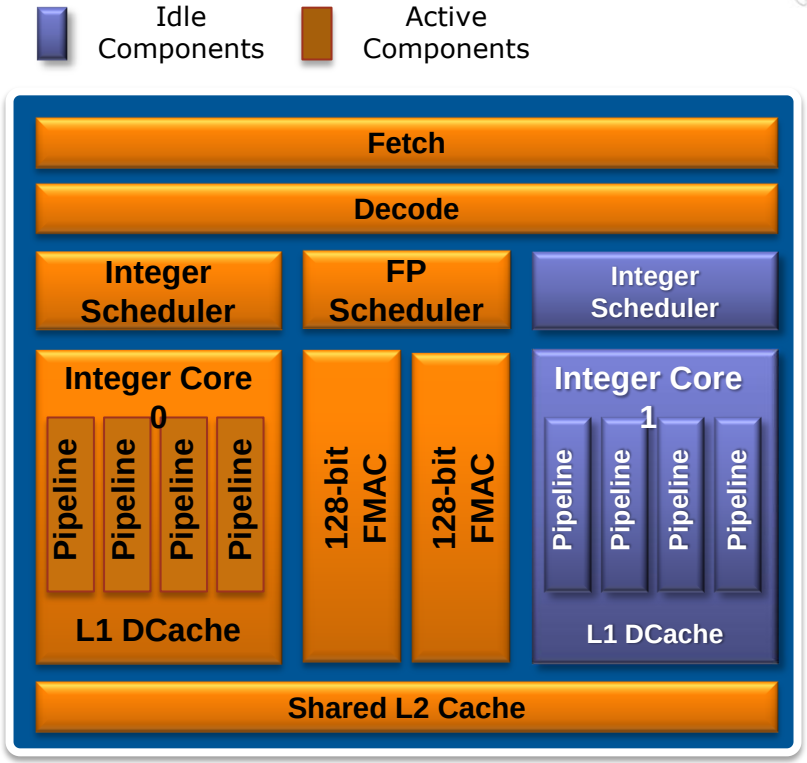
Two MPI Tasks on a Compute Unit ("Dual-Stream Mode")

- **An MPI task is pinned to each integer unit**
  - Each integer unit has exclusive access to an integer scheduler, integer pipelines and L1 Dcache
  - The 256-bit FP unit, instruction fetch, and the L2 Cache are shared between the two integer units
    - 256-bit AVX instructions are dynamically executed as two 128-bit instructions if the 2nd FP unit is busy
- **When to use**
  - Code is highly scalable to a large number of MPI ranks
  - Code can run with a 2GB per task memory footprint
  - Code is not well vectorized



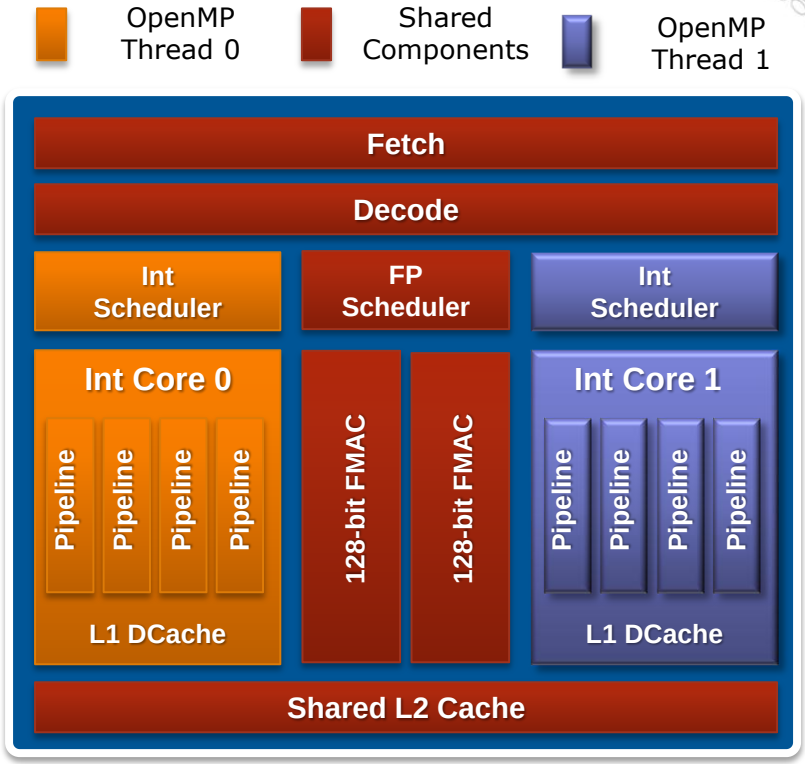
One MPI Task on a Compute Unit ("Single Stream Mode")

- **Only one integer unit is used per compute unit**
  - This unit has exclusive access to the 256-bit FP unit and is capable of 8 FP results per clock cycle
  - The unit has twice the memory capacity and memory bandwidth in this mode
  - The L2 cache is effectively twice as large
  - The peak of the chip is not reduced
- **When to use**
  - Code is highly vectorized and makes use of AVX instructions
  - Code benefits from higher per task memory size and bandwidth



One MPI Task per compute unit with Two OpenMP Threads ("Dual-Stream Mode")

- **An MPI task is pinned to a compute unit**
- **OpenMP is used to run a thread on each integer unit**
  - Each OpenMP thread has exclusive access to an integer scheduler, integer pipelines and L1 Dcache
  - The 256-bit FP unit and the L2 Cache is shared between the two threads
  - 256-bit AVX instructions are dynamically executed as two 128-bit instructions if the 2nd FP unit is busy
- **When to use**
  - Code needs a large amount of memory per MPI rank
  - Code has OpenMP parallelism at each MPI rank



AVX (Advanced Vector Extensions)

- **Max Vector length doubled to 256 bit**
- **Much cleaner instruction set**
  - Result register is unique from the source registers
  - Old SSE instruction set always destroyed a source register
- **Floating point multiple-accumulate**
  - A(1:4) = B(1:4)*C(1:4) + D(1:4) ! Now one instruction
- **Both AMD and Intel now have AVX**

- **Vectors are becoming more important, not less**

Running in Dual-Stream mode

- Dual-Stream mode is the current default mode. General use does not require any options. CPU affinity is set automatically by ALPS.
- Use the `aprun -d` option to set the number of OpenMP threads per process. If OpenMP is not used, no `-d` option is required. The `aprun -N` option is used to specify the number of MPI processes to assign per compute node or -S to specify the number of MPI processes per Interlagos die. These options are generally only needed in the case of OpenMP programs or programs needed more memory per process.

Running in Single-Stream mode

- Single-Stream mode is specified through the `-j` aprun option. Specifying `-j 1` tells aprun to place 1 process or thread on each compute unit.
- When OpenMP threads are used, the `-d` option must be used to specify how many threads will be spawned per MPI process. See the `aprun(1)` man page for more details. The `aprun -N` option may be used to specify the number of MPI processes to assign per compute node or `-S` to specify the number of processes per Interlagos die. Also, the environment variable `$OMP_NUM_THREADS` needs to be set to the correct number of threads per process.

- For example, the following spawns 4 MPI processes, each with 8 threads, using 1 thread per compute unit.

```
OMP_NUM_THREADS=8  aprun -n 4 -d 8 -j 1 ./a.out
```

aprun Examples (XK7)

- No-OpenMP, 16 MPI processes per node

```
<default>
```

- No-OpenMP, 8 MPI processes per node

```
-j 1
```

- OpenMP, 2 MPI processes, 8 threads per process

```
-d 8
```

- OpenMP, 2 MPI processes, 4 threads per process

```
-d 4 -j 1
```

- OpenMP, 1 MPI process, 16 threads

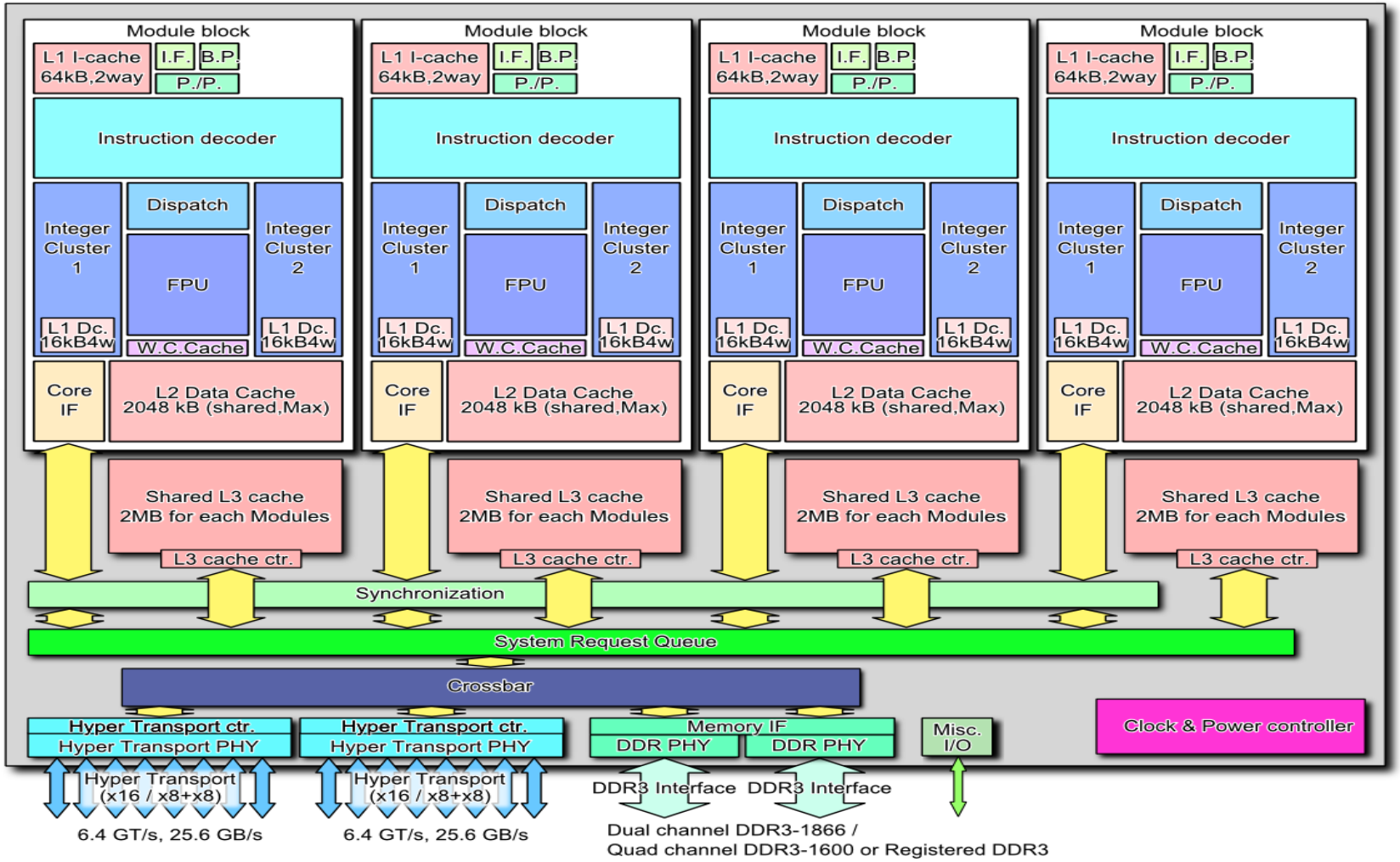
```
-d 16
```

- OpenMP, 1 MPI process, 8 threads

```
-d 8 -j 1
```

NUMA Considerations

- An XK7 compute node with 1 Interlagos processors has 2 NUMA memory domains, each with 4 Bulldozer Modules. Access to memory located in a remote NUMA domain is slower than access to local memory. Bandwidth is lower, and latency is higher.
- OpenMP performance is usually better when all threads in a process execute in the same NUMA domain. For the Dual-Stream case, 8 CPUs share a NUMA domain, while in Single-Stream mode 4 CPUs share a NUMA domain. Using a larger number of OpenMP threads per MPI process than these values may result in lower performance due to cross-domain memory access.
- When running 1 process with threads over both NUMA domains, it's critical to initialize (not just allocate) memory from the thread that will use it in order to avoid NUMA side effects.


Module block
L1 I-cache 64kB,2way
I.F.
B.P.
P./P.
Instruction decoder
Integer Cluster 1
Dispatch
FPU
Integer Cluster 2
L1 Dc. 16kB4w
W.C.Cache
Core IF
L2 Data Cache 2048 kB (shared,Max)
Shared L3 cache 2MB for each Modules
L3 cache ctr.
Synchronization
System Request Queue
Crossbar
Hyper Transport ctr.
Hyper Transport PHY
Memory IF
DDR PHY
Misc. I/O
Clock & Power controller
Hyper Transport (x16 / x8+x8)
6.4 GT/s, 25.6 GB/s
DDR3 Interface
Dual channel DDR3-1866 /
Quad channel DDR3-1600 or Registered DDR3

Cray Gemini Interconnect

Cray Network Evolution

SeaStar

- Built for scalability to 250K+ cores
- Very effective routing and low contention switch

Gemini

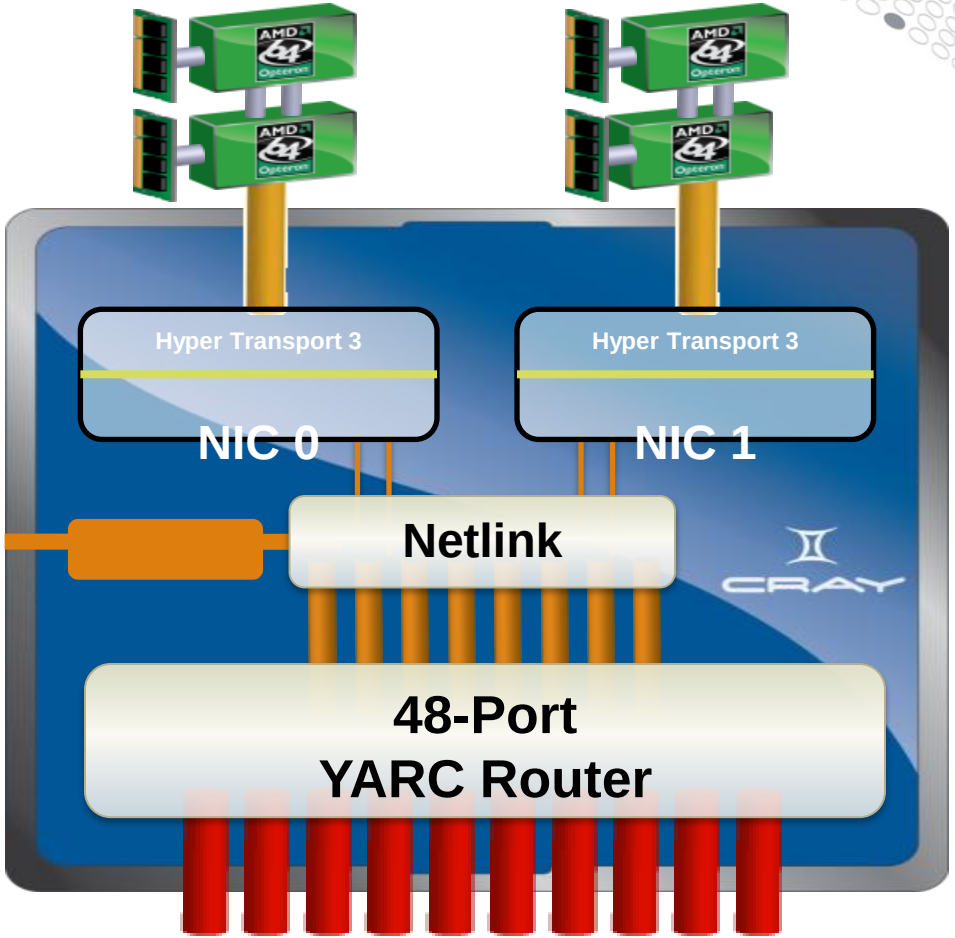
- 100x improvement in message throughput
- 3x improvement in latency
- PGAS Support, Global Address Space
- Scalability to 1M+ cores

Aries

- Cray “Cascade” Systems
- Funded through DARPA program
- Details not yet publically available

Cray Gemini

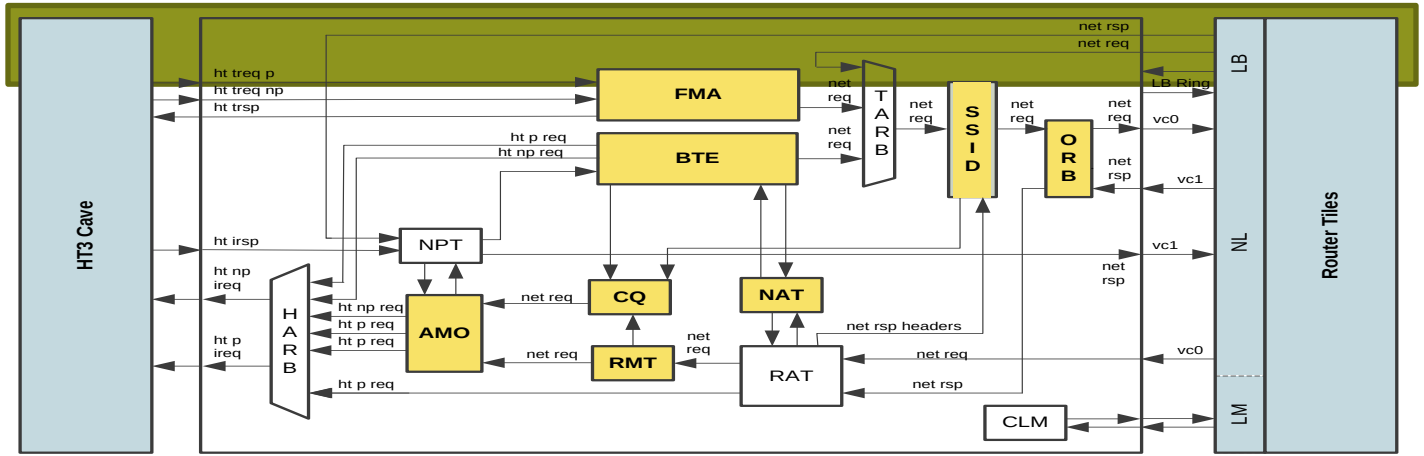
- **3D Torus network**
- **Supports 2 Nodes per ASIC**
- **168 GB/sec routing capacity**
- **Scales to over 100,000 network endpoints**
  - Link Level Reliability and Adaptive Routing
  - Advanced Resiliency Features
- **Provides global address space**
- **Advanced NIC designed to efficiently support**
  - MPI
    - Millions of messages/second
  - One-sided MPI
  - UPC, FORTRAN 2008 with coarrays, shmem
  - Global Atomics



Gemini Advanced Features

- **Globally addressable memory provides efficient support for UPC, Co-array FORTRAN, Shmem and Global Arrays**
  - Cray Programming Environment will target this capability directly
- **Pipelined global loads and stores**
  - Allows for fast irregular communication patterns
- **Atomic memory operations**
  - Provides fast synchronization needed for one-sided communication models

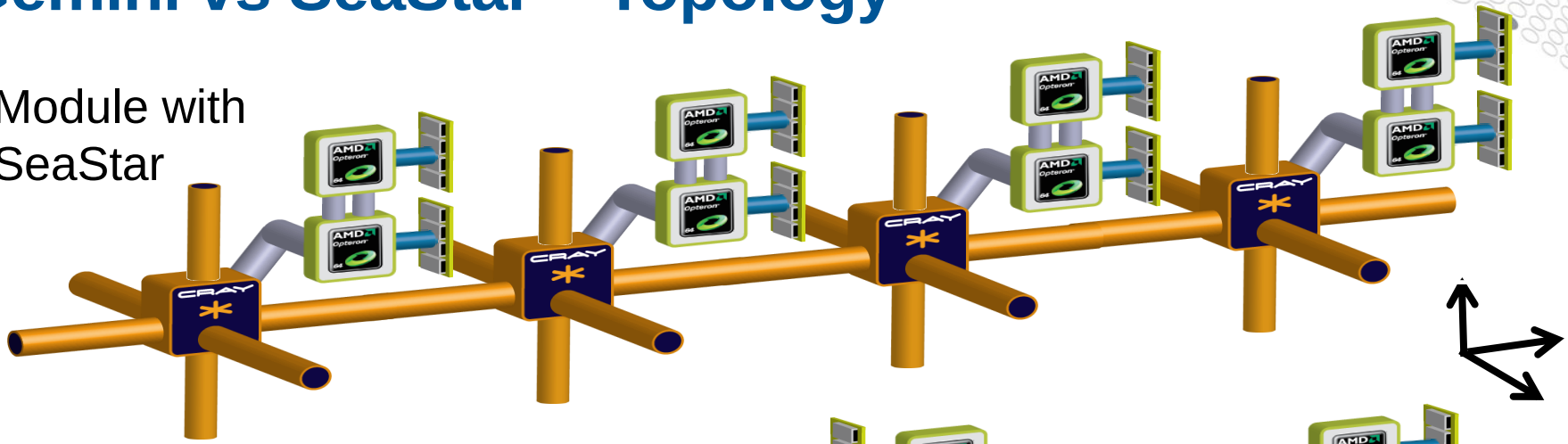
Gemini NIC block diagram

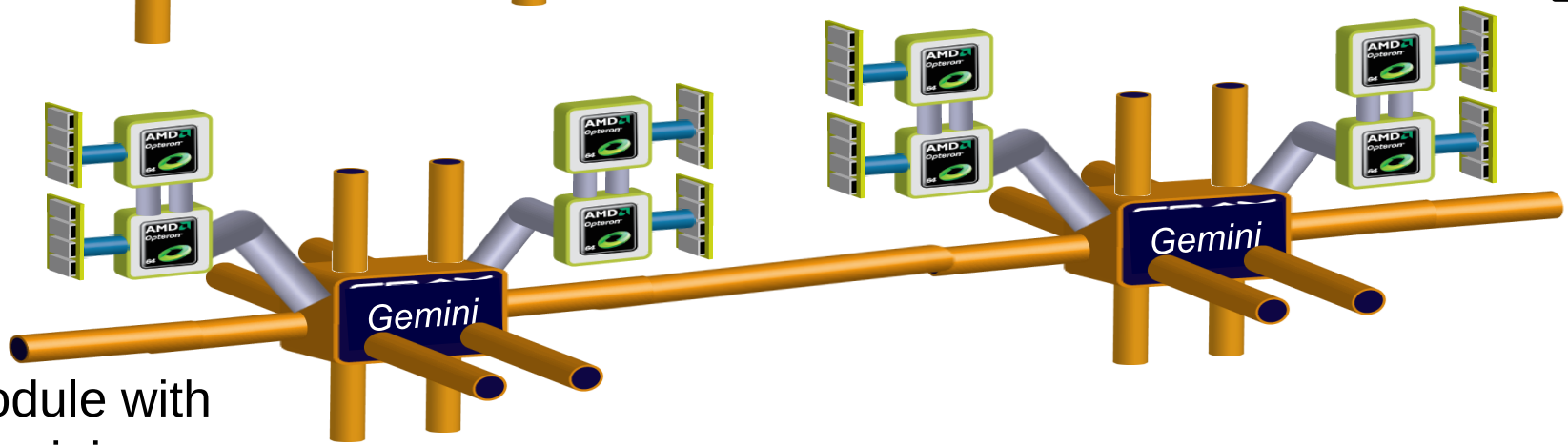


- **FMA (Fast Memory Access)**
  - Mechanism for most MPI transfers
  - Supports tens of millions of MPI requests per second
- **BTE (Block Transfer Engine)**
  - Supports *asynchronous* block transfers between local and remote memory, in either direction
  - For use for large MPI transfers that happen in the background

Gemini vs SeaStar – Topology

Module with SeaStar





Module with Gemini

A Question About the Torus…

It looks like for each x,y,z coordinate, there are two node numbers associated. Is there some reason for this? Is each node number actually indicating 8-cores rather than 16?

Node	X	Y	Z
0	0	0	0
1	0	0	0
2	0	0	1
3	0	0	1
4	0	0	2
5	0	0	2

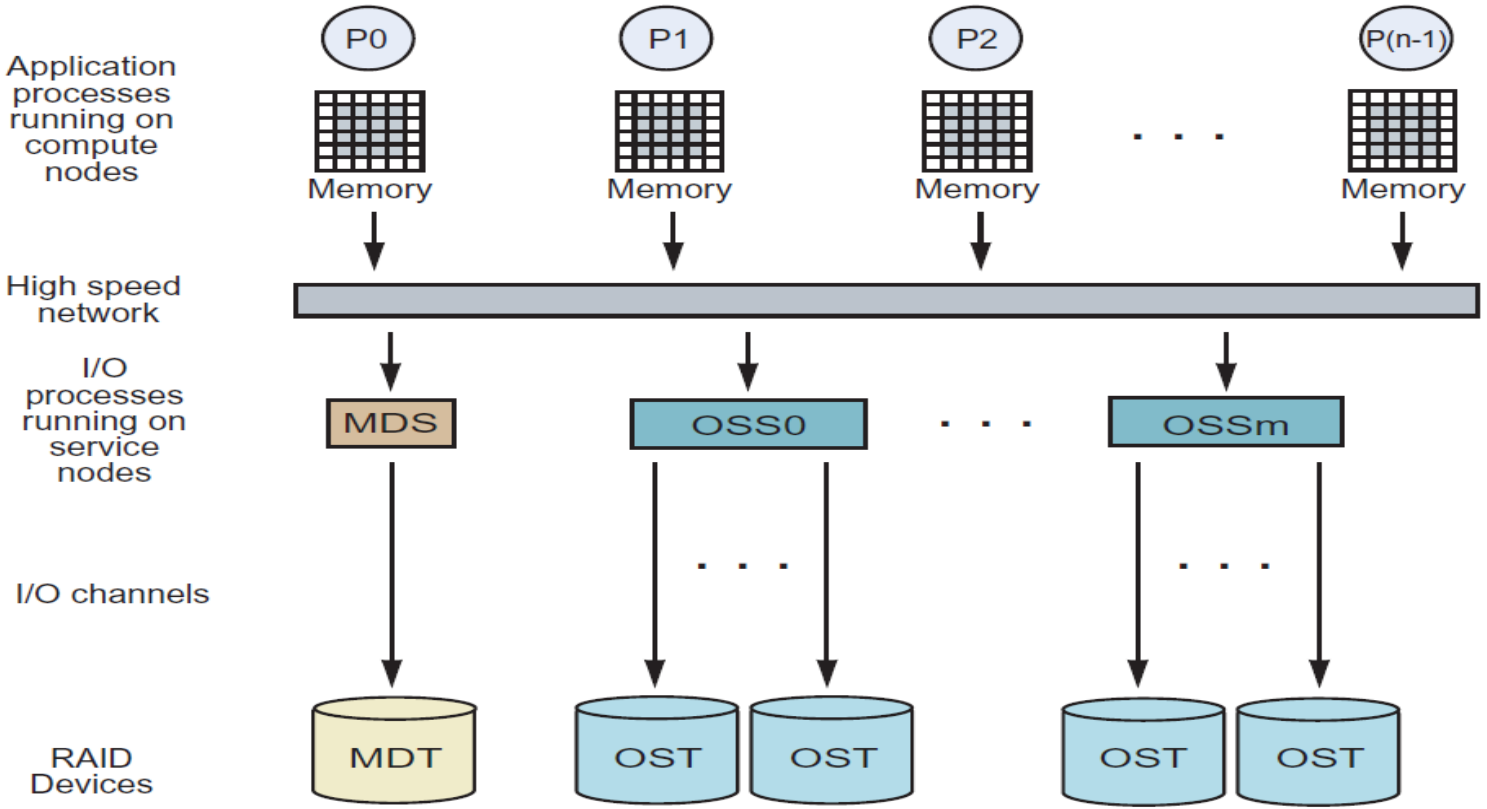
- Unlike the XT-line of systems, where each node had an idividual SeaStar, a Gemini services 2 compute nodes.
- So, 2 compute nodes will have the same dimensions in the torus in an XE or XK system.

Lustre Filesystem Basics

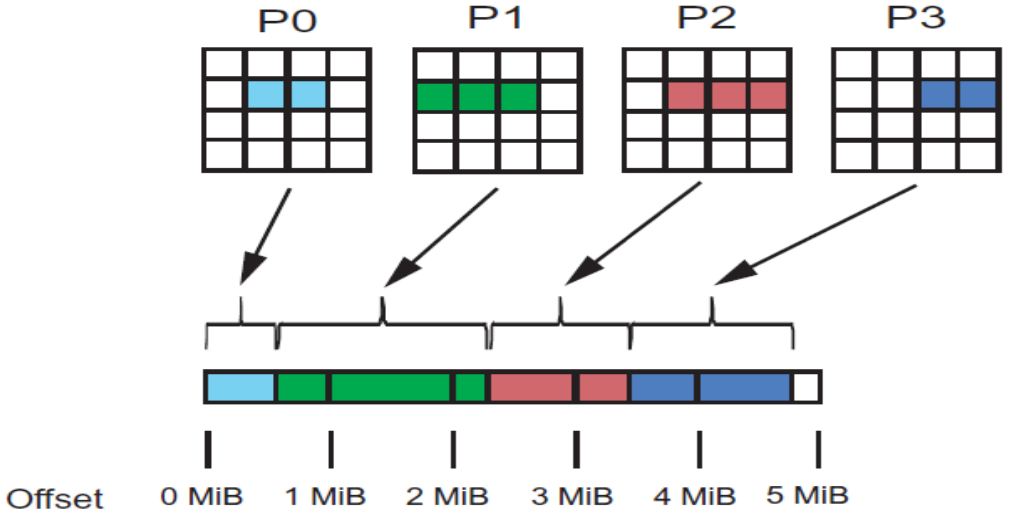
Key Lustre Terms

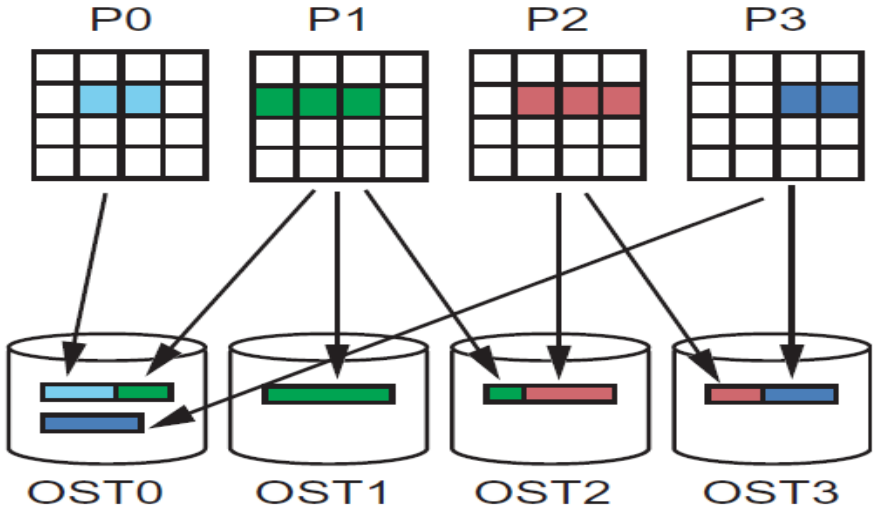
Term	Meaning	Purpose
MDS	Metadata Server	Manages all file metadata for filesystem. 1 per FS
OST	Object Storage Target	The basic “chunk” of data written to disk. Max 160 per file.
OSS	Object Storage Server	Communicates with disks, manages 1 or more OSTs. 1 or more per FS
Stripe Size	Size of chunks.	Controls the size of file chunks stored to OSTs. Can’t be changed once file is written.
Stripe Count	Number of OSTs used per file.	Controls parallelism of file. Can’t be changed once file is writte.

Lustre File System Basics



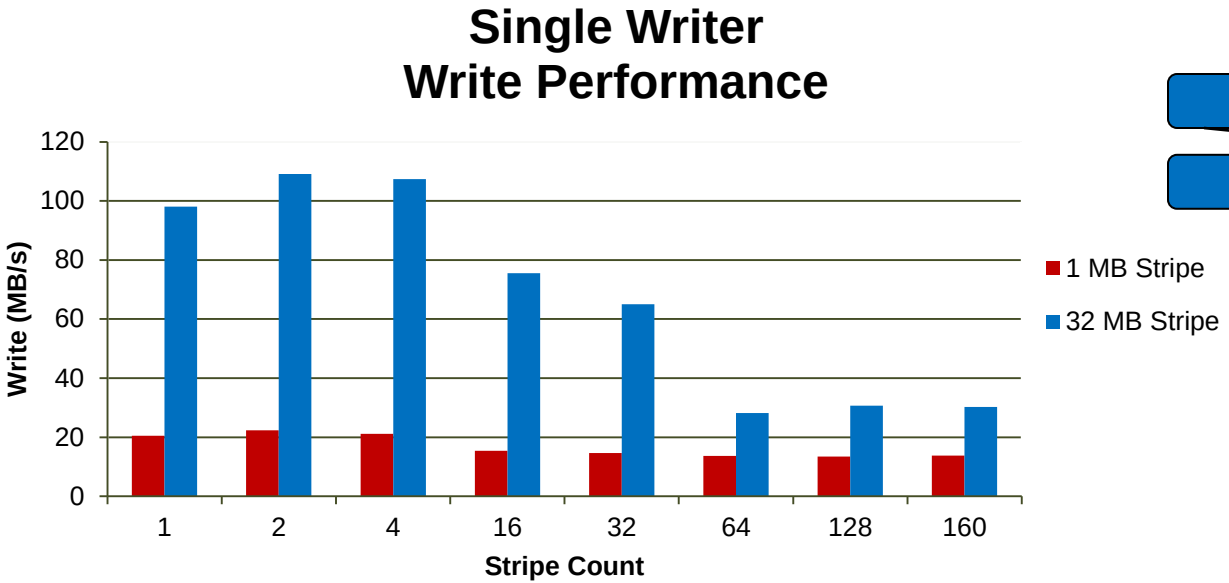
File Striping: Physical and Logical Views

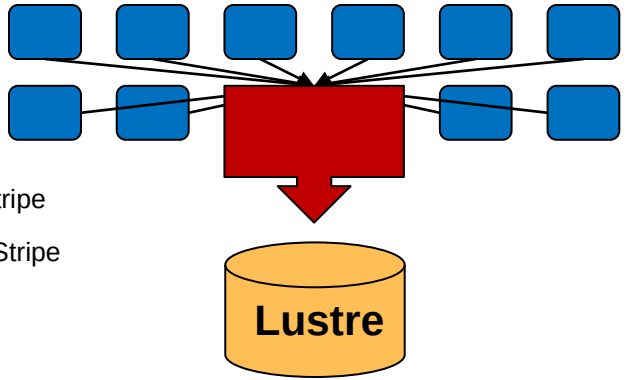




Single writer performance and Lustre

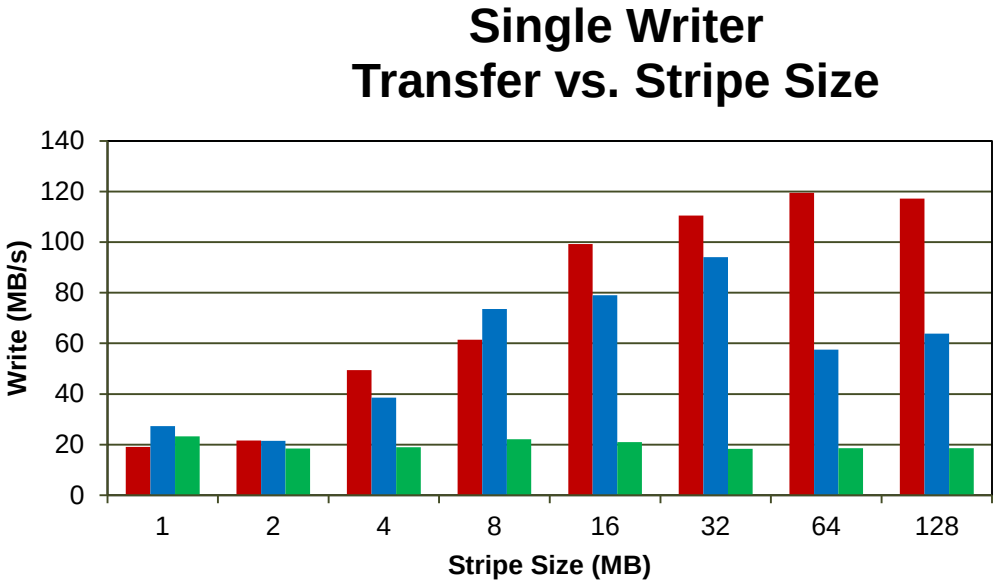
- **32 MB per OST (32 MB – 5 GB) and 32 MB Transfer Size**
  - Unable to take advantage of file system parallelism
  - Access to multiple disks adds overhead which hurts performance

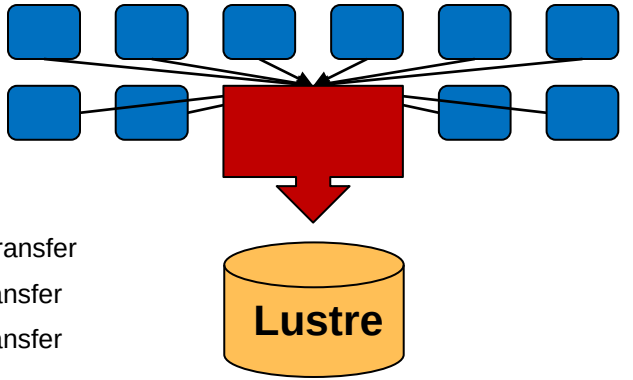




Stripe size and I/O Operation size

- **Single OST, 256 MB File Size**
  - Performance can be limited by the process (transfer size) or file system (stripe size)





Lustre: Important Information

- **Use the lfs command, libLUT, or MPIIO hints to adjust your stripe count and possibly size**
  - lfs setstripe -c -1 -s 4M <file or directory> (160 OSTs, 4MB stripe)
  - lfs setstripe -c 1 -s 16M <file or directory> (1 OST, 16M stripe)
  - export MPICH_MPIIO_HINTS='*: striping_factor=160'
- **Files inherit striping information from the parent directory, this cannot be changed once the file is written**
  - Set the striping before copying in files

NVIDIA KEPLER ACCELERATOR

Tesla K20: Same Power, 3x DP Perf of Fermi

Product Name	M2090	K20X
Peak Single Precision Peak SGEMM	1.3 Tflops 0.87 TF	3.95 TF 3.35 TF
Peak Double Precision **Peak DGEMM**	**0.66 Tflops** **0.43 TF**	**1.32 TF** **1.12 TF**
Memory size	6 GB	6 GB
Memory BW (ECC off)	177.6 GB/s	250 GB/s
New CUDA Features	-	GPUDirect w/ RDMA, Hyper-Q, Dynamic Parallelism
ECC Features	DRAM, Caches & Reg Files	DRAM, Caches & Reg Files
# CUDA Cores	512	2688
Total Board Power	225W	225W

3x Double Precision

Hyper-Q, Dynamic Parallelism

CFD, FEA, Finance, Physics