# AMS 250: An Introduction to High Performance Computing

# Getting Started on NERSC Supercomputers

**Shawfeng Dong**

shaw@ucsc.edu

(831) 502-7743

Applied Mathematics & Statistics

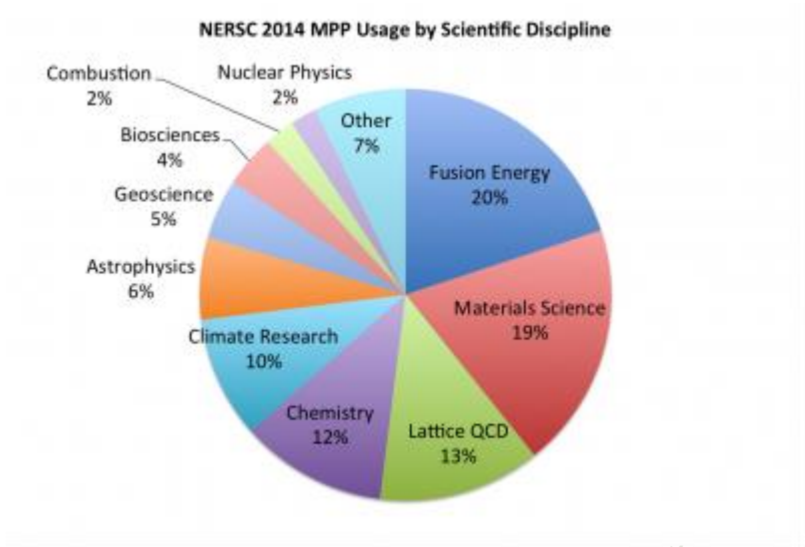
University of California, Santa Cruz

# Outline

- About NERSC
- Hardware Architecture of NERSC Supercomputers
    - Nodes
    - Interconnects
    - Storage
- Accessing NERSC Supercomputers
- Computing Environment
- Compiling Codes
- Running Jobs
- Visualization and Analysis

# About NERSC

- http://www.nersc.gov/about/
- National Energy Research Scientific Computing Center (**NERSC**) is the primary scientific computing facility for the Office of Science in the U.S. Department of Energy
- A division of Lawrence Berkeley National Laboratory
- One of the largest facilities in the world devoted to providing computational resources and expertise for basic scientific research
- More than 6,000 scientists use NERSC to perform basic scientific research across a wide range of disciplines



# NERSC Systems

http://www.nersc.gov/systems/

- Edison

  A Cray XC30, with a peak performance of more than 2 petaflops

- Cori

  A Cray XC40, NERSC’s newest supercomputer (NERSC-8)

- PDSF

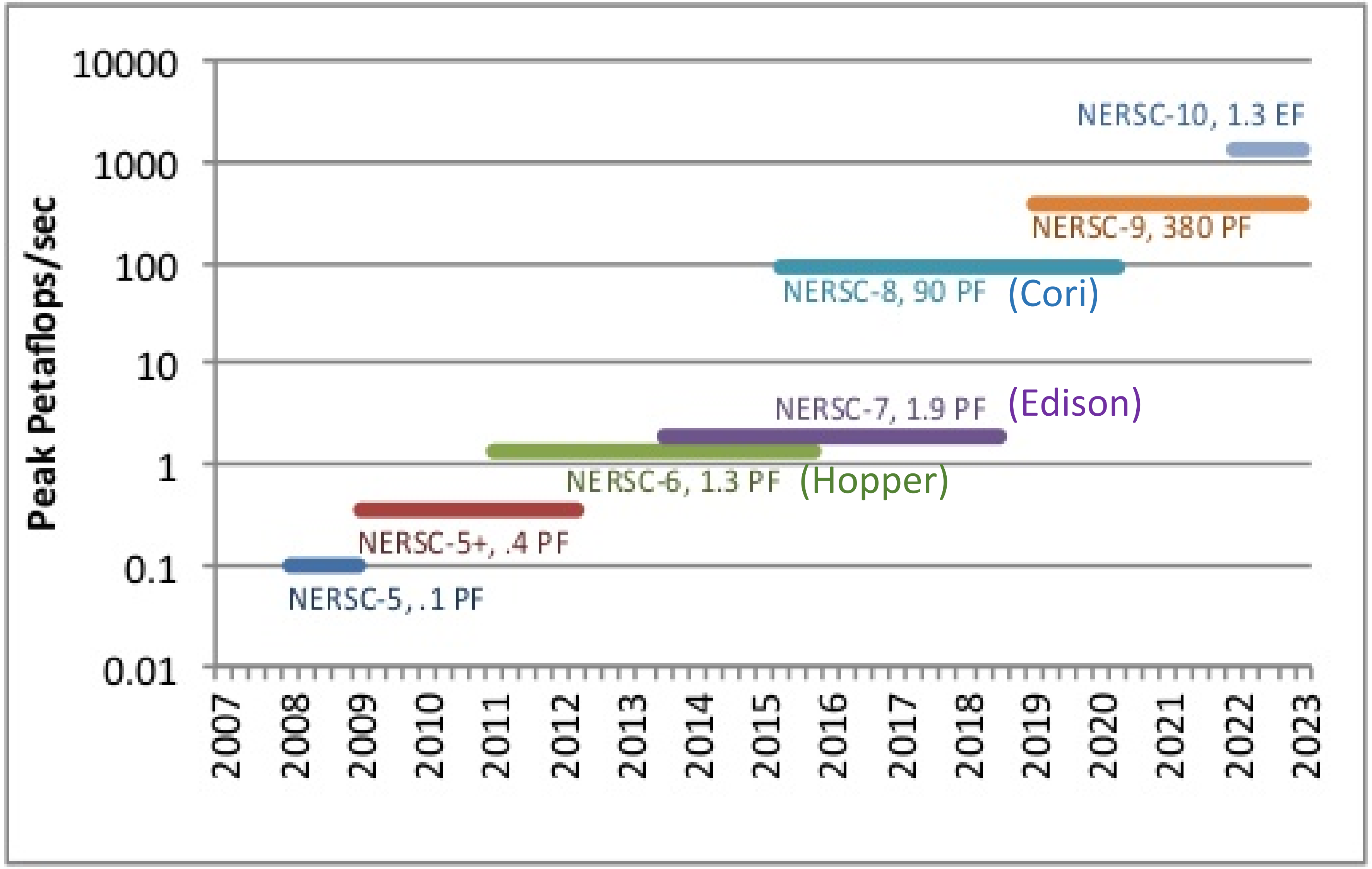
  A networked distributed computing environment used to meet the detector simulation and data analysis requirements of large scale Physics, High Energy Physics and Astrophysics and Nuclear Science investigations

- Genepool

  A cluster dedicated to the JGI's computing needs

- HPSS data archive

# NERSC Roadmap



# Edison System Overview

- Cray XC30 supercomputer (NERSC-7)
- 5,586 Compute Nodes, each with:
  - Two Intel “Ivy Bridge” Xeon E5-2695 v2 12-core CPU @ 2.4 GHz
  - 64 GB DDR3 1866 MHz memory
- 12 Login Nodes (quad-socket, quad-core) with 512GB memory each
- Cray Aries Interconnect with Dragonfly topology, 23.7 TB/s global bandwidth
- 7.56 PB of Lustre scratch storage, with an aggregate transfer rate of 168 GB/s

Total # of cores = 134,064

Aggregate memory = 357 TB

$R_{peak}$ = 2.569 PFLOPS

$R_{max}$ = 1.655 PFLOPS

Sustained application performance =

293 TFLOPS

Power = 3.7 MW

#18 in June 2014, #60 in November 2016

# Cori System Overview

- Cray XC40 supercomputer (NERSC-8)
- 2,388 Phase I Haswell Compute Nodes, each with:
  - Two Intel “Haswell” Xeon E5-2698v3 16-core CPU @ 2.3 GHz
  - 128 GB DDR4 2133 MHz memory
- 9,688 Phase II KNL Compute Nodes
- 12 Login Nodes (dual-socket, 16-core) with 512GB memory each
- Cray Aries Interconnect with Dragonfly topology, 5.625 TB/s global BW (Phase I), 45.0 TB/s global BW (Phase II)
- 30 PB of Lustre scratch storage, with an aggregate transfer rate of > 700 GB/s
- 1.8 PB of Burst Buffer

Total # of Phase I Haswell cores = 76,416

Total # of Phase II KNL cores = 658,784

Aggregate memory = 203 TB (Phase I),
1PB (Phase II)

$R_{peak}$ = 1.92 PFLOPS (I) + 29.1 PFLOPS (II)

$R_{max}$ = 14 PFLOPS

Power = 3.9 MW

#8 in November 2017

# UCSC Hyades Overview

- 180 Type I Compute Nodes, each with:
  - Two Intel “Sandy Bridge” Xeon E5-2650 8-core CPU @ 2.0 GHz
  - 64 GB 1600 MHz memory
- 8 Type II Compute Nodes, each with:
  - Two Intel “Sandy Bridge” Xeon E5-2650 8-core CPU @ 2.0 GHz
  - 64 GB 1600 MHz memory
  - One Nvidia K20 GPU accelerator
- 1 Login Node (quad-socket, 8-core) with 128GB memory
- 1 Analysis Node (quad-socket, 8-core) with 512GB memory
- QDR (40 Gb/s) InfiniBand Interconnect with a non-blocking fat tree topology, 4.32 Tb/s global bisection bandwidth
- 146 TB of Lustre scratch storage

Total # of x84-64 cores = 3,008

Aggregate memory = 12 TB

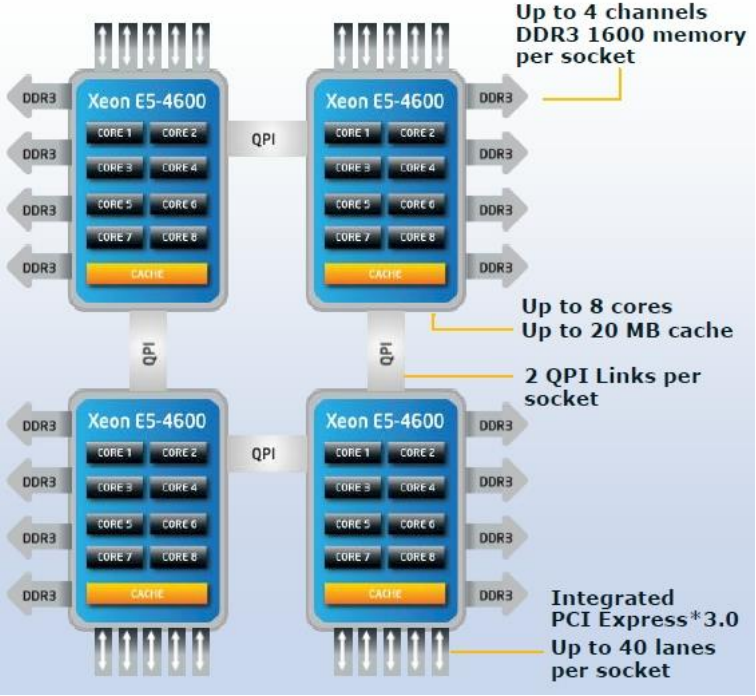
$R_{peak}$ = 58 TFLOPS

Power = 60 KW

Cost = $1M (2012)

# Edison Login Nodes

- http://www.nersc.gov/users/connecting-to-nersc/login-nodes-2/
- Login Nodes, aka Frontends or Master Nodes
  - When you ssh to Edison, you are connecting to a “Login Node”
  - 12 Login Nodes sit behind a load balancer
- Each Login Node is quad-socket server
  - 4x Intel “Sandy Bridge” Xeon 4-core E5-4603 processors @ 2.0 GHz
  - 512 GB memory
  - Runs standard SUSE Linux (Enterprise Server 11 SP3)
- Intended tasks:
  - editing files
  - compiling codes
  - submitting job scripts to the batch system to be run on the Compute Nodes
  - light test runs
- Public hostname: edison.nersc.gov



Up to 4 channels DDR3 1600 memory per socket

Up to 8 cores Up to 20 MB cache

2 QPI Links per socket

Integrated PCI Express*3.0 Up to 40 lanes per socket

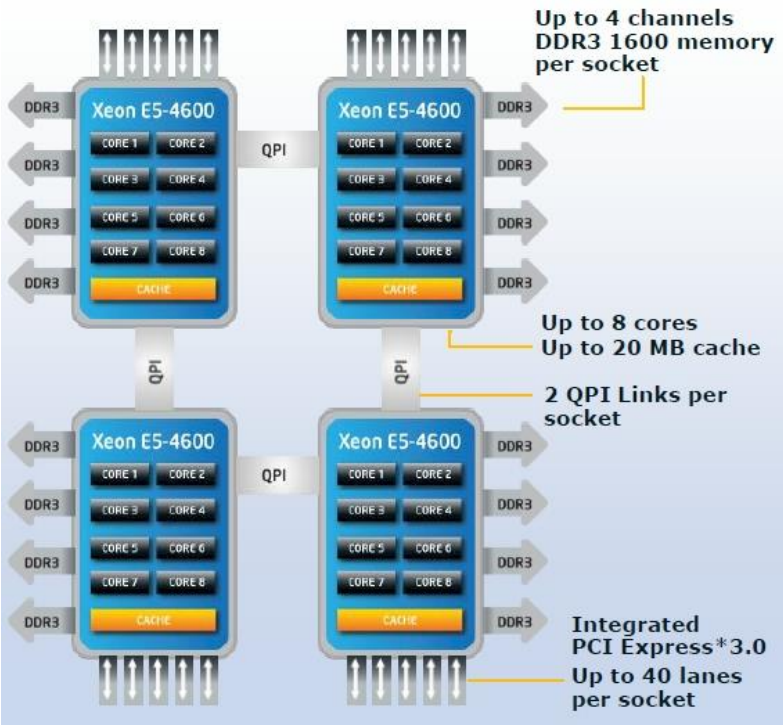
# Intel Xeon Processor E5-4603

- **Microarchitecture**: Sandy Bridge
- 4 cores / 8 threads
- **Process**: 32 nm
- **Frequency**: 2.0GHz
- **L1 cache**: 32 KB (code) + 32 KB (data) per core
- **L2 cache**: 256 KB per core
- **L3 cache**: 10 MB *shared*
- AVX (Advanced Vector Extensions)

  256-bit, 8 double-precision FLOP per cycle (why?)
- 2 QPI links @ 6.4 GT/s (3.2GHz)

  6.4 x 2 x 20 x 64/80 / 8 = 25.6 GB/s
- 42.656 GB/s memory bandwidth @DDR3-1066

  1.066 (GT/s) x 8 (Byte) x 4 (channels) = 34.112 GB/s



Up to 4 channels DDR3 1600 memory per socket

Up to 8 cores
Up to 20 MB cache

2 QPI Links per socket

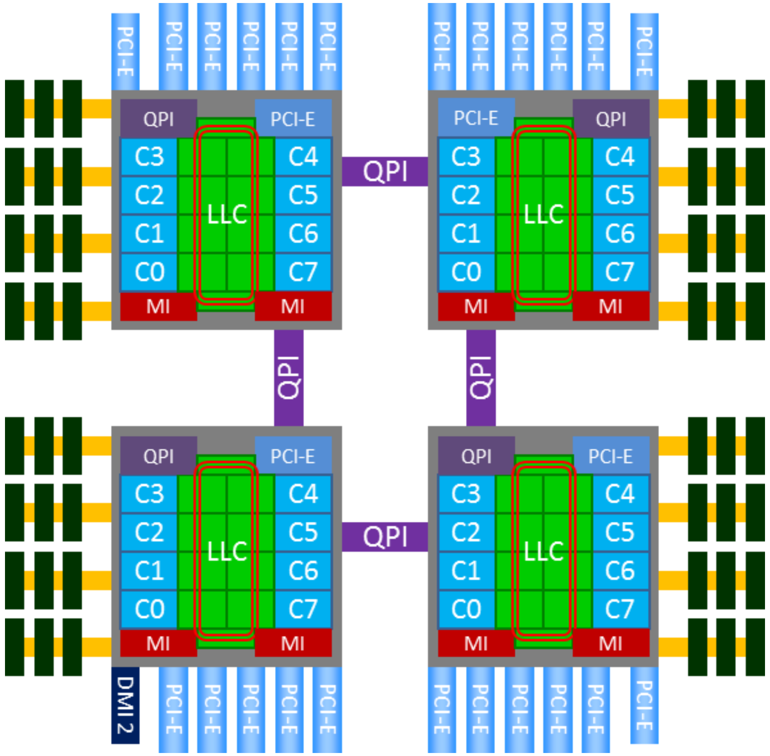
Integrated PCI Express*3.0
Up to 40 lanes per socket

http://www.cpu-upgrade.com/CPUs/Intel/Xeon/E5-4603.html

# 4-way Sandy Bridge System

- Intel QuickPath Interconnect (QPI)
  - point-to-point processor interconnect
  - two 20-lane links, one in each direction
  - unit of transfer is 80-bit “flit”, with 64 bits for data
  - bandwidth (Xeon E5-4603) = 
    6.4 GT/s * 2 (links) * 20 (lanes) * 64/80 / 8 (bits/byte)
- NUMA (non-uniform memory access)
  - Local memory bandwidth: 34.1 GB/s (Xeon E5-4603)
  - Remote memory bandwidth: 25.6 GB/s (Xeon E5-4603)
- Cache Coherence
  - 256-bit/cycle ring bus interconnect between on-die cores
  - QPI between processors
  - MESIF protocol

    5 states: Modified (M), Exclusive (E), Shared (S), Invalid (I) and Forward (F)

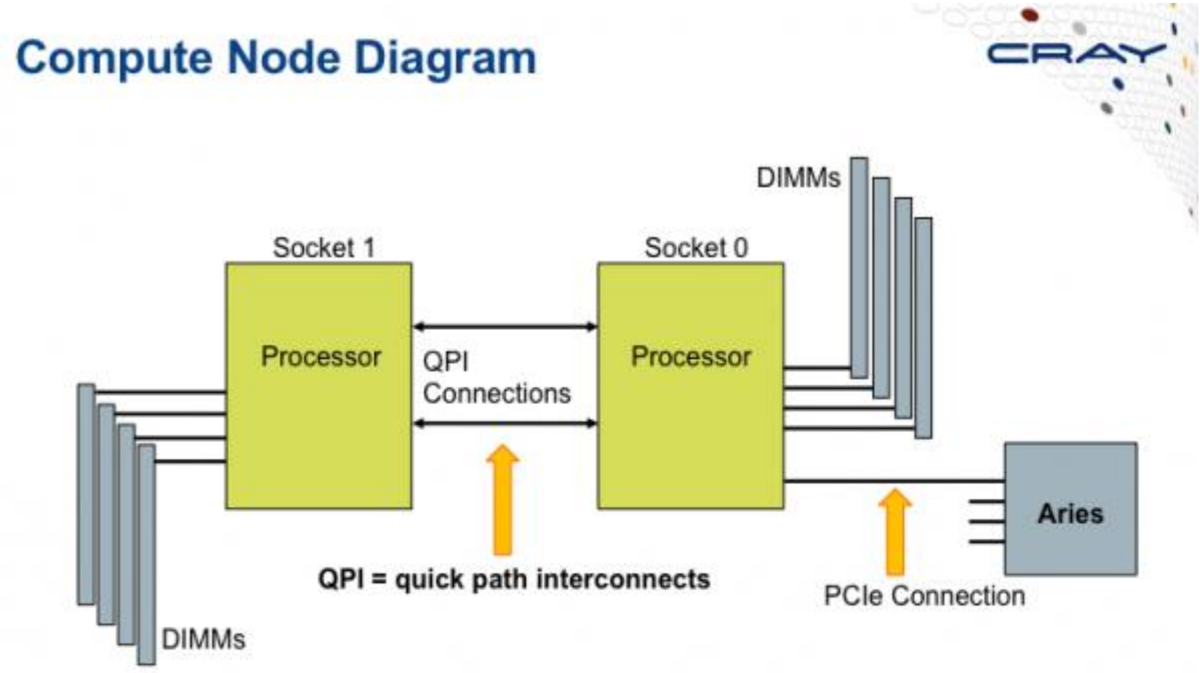


http://www.qdpma.com/systemarchitecture/systemarchitecture_sandybridge.html

# Edison Compute Nodes

- 5,586 Compute Nodes, each with:
  - Two Intel "Ivy Bridge" Xeon E5-2695 v2 12-core CPU @ 2.4 GHz
  - 64 GB DDR3 1866 MHz memory (four 8 GB DIMMs per socket)
  - An Aries Network Interface Controller (NIC)

    0.25 μs to 3.7 μs MPI latency,
    ~8GB/sec MPI bandwidth
- Compute nodes run a *lightweight* kernel and run-time environment based on SuSE Linux Enterprise Server (SLES) Linux distribution



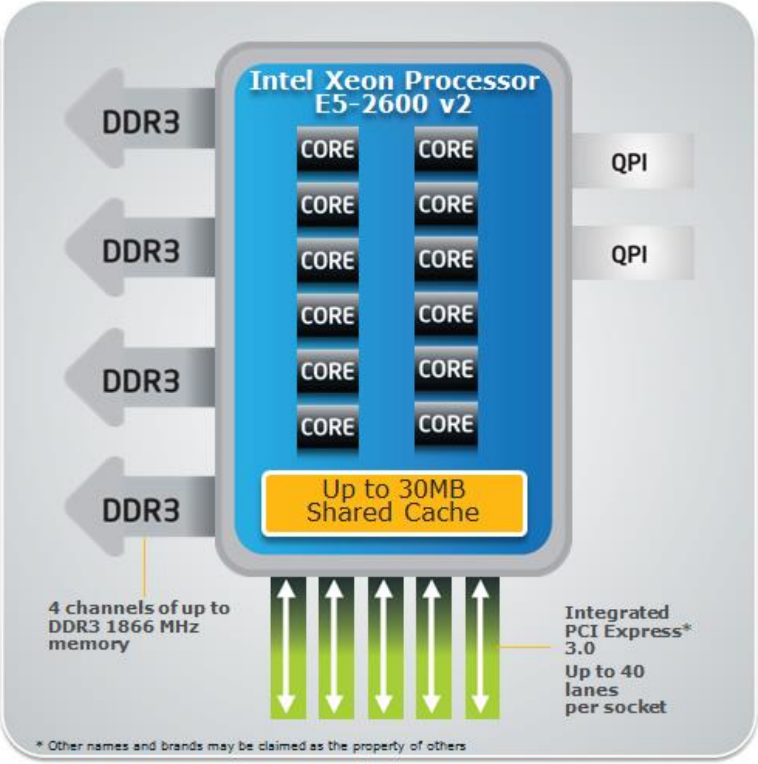
# Intel Xeon Processor E5-2695 v2

- **Microarchitecture**: Ivy Bridge
- 12 cores / 24 threads
- **Process**: 22 nm
- **Frequency**: 2.4GHz / 3.2GHz max Turbo
- **L1 cache**: 32 KB (code) + 32 KB (data) per core
- **L2 cache**: 256 KB per core
- **L3 cache**: 30 MB *shared*
- AVX (Advanced Vector Extensions)

  256-bit, **8** double-precision FLOP per cycle
- 2 QPI links @ 8.0 GT/s (4.0 GHz)

  8.0 x 2 x 20 x 64/80 / 8 = 32 GB/s
- 59.7 GB/s memory bandwidth @DDR3-1866

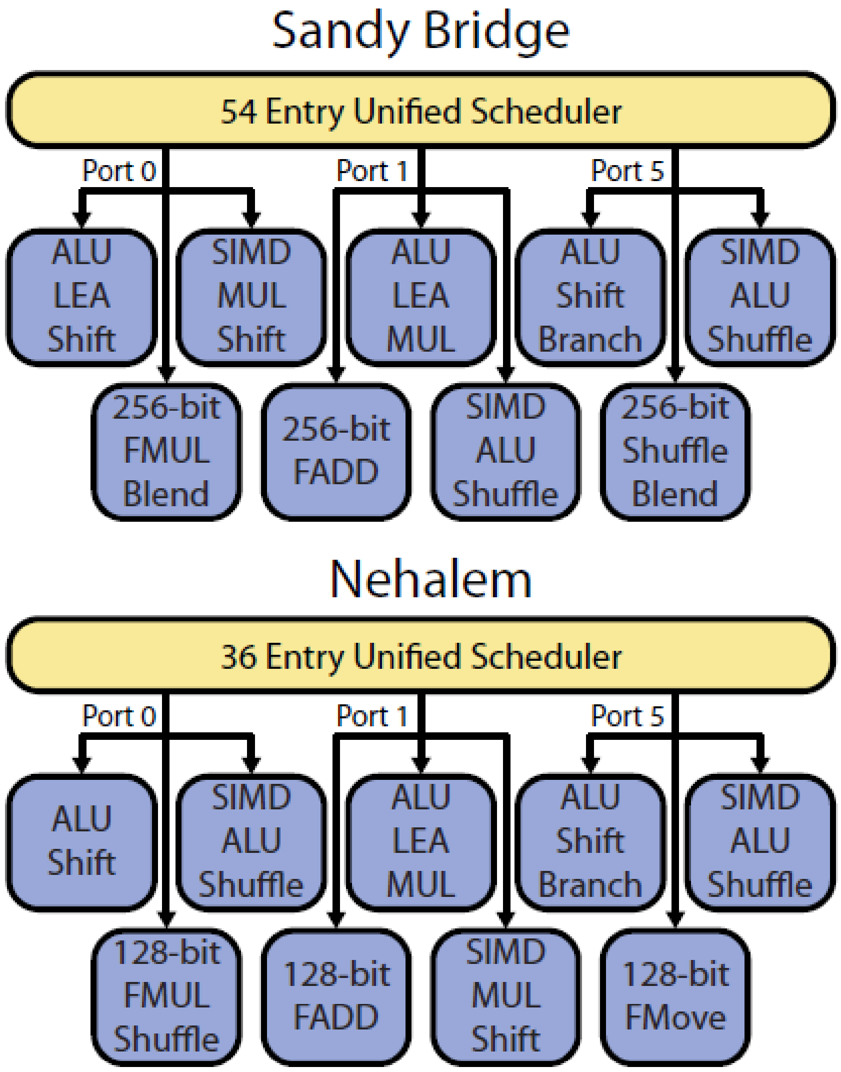
  1.866 (GT/s) x 8 (Byte) x 4 (channels) = 59.7 GB/s


Intel Xeon Processor E5-2600 v2
DDR3
DDR3
DDR3
DDR3
CORE
CORE
CORE
CORE
CORE
CORE
CORE
CORE
CORE
CORE
CORE
CORE
QPI
QPI
Up to 30MB Shared Cache
4 channels of up to DDR3 1866 MHz memory
Integrated PCI Express* 3.0
Up to 40 lanes per socket
* Other names and brands may be claimed as the property of others

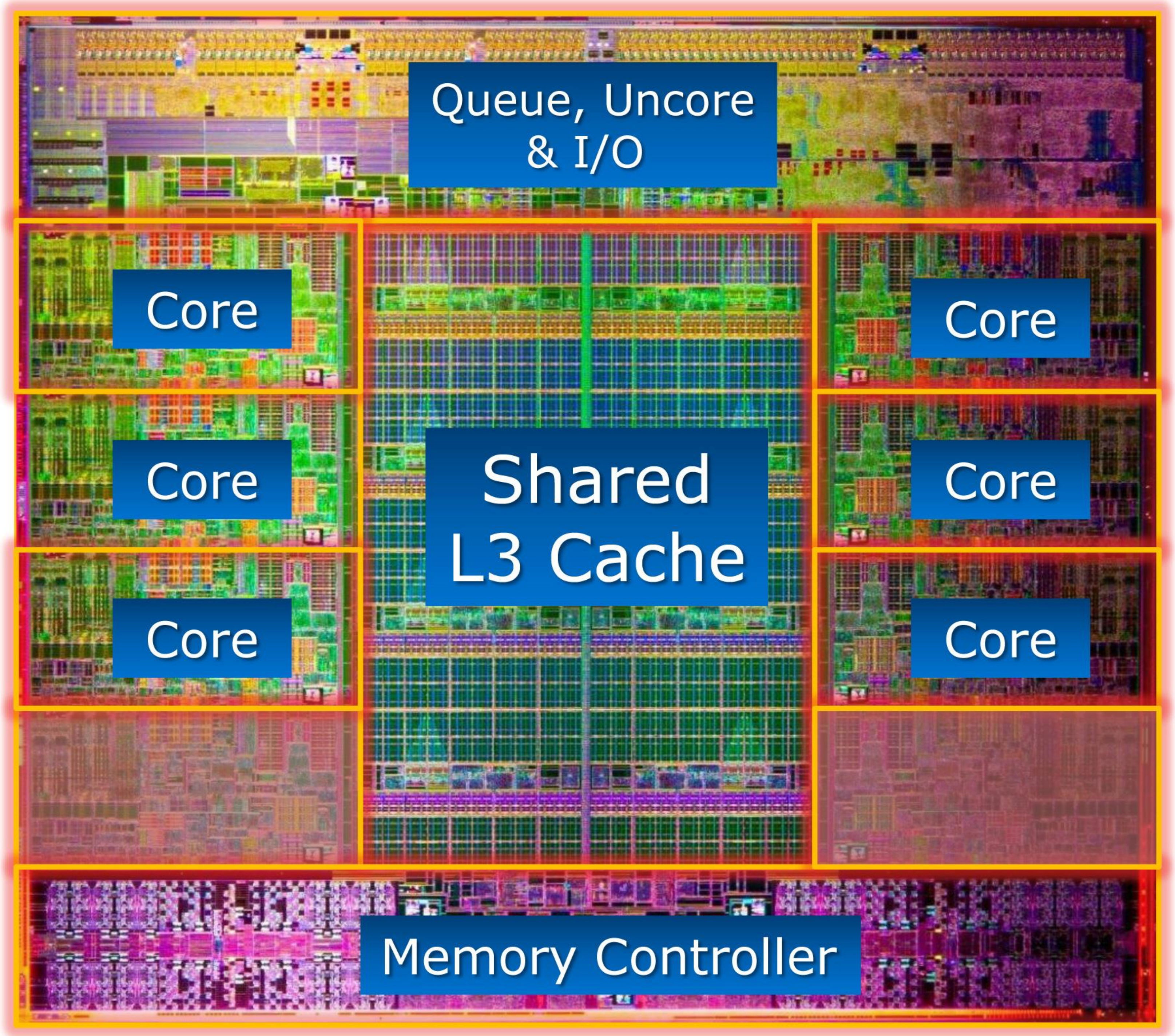
http://www.cpu-upgrade.com/CPUs/Intel/Xeon/E5-2695_v2.html

# Sandy Bridge Microarchitecture

- Instruction-Level Parallelism (ILP)
  - 3 integer ALU, 2 vector ALU and 2 AGU per core
  - 2 load/store operations per CPU cycle for each memory channel
  - 4 branch predictors
  - hyper-threading (2 logical core per physical core)
  - decoded micro-operation cache (uop cache)
  - 14- to 19- stage instruction pipeline, depending on the uop cache hit or miss
- Vector processing
  - 256-bit AVX (Advanced Vector Extensions)
  - can execute a 256-bit FP multiply, a 256-bit FP add and a 256-bit shuffle every cycle -> 8 DP FLOPS
- Ivy Bridge is the 22nm die shrink of Sandy Bridge (32nm)



http://www.realworldtech.com/sandy-bridge/



- 64-byte cache line size
- 32KB, 8-way associative L1 data cache per core
- 32KB, 4-way associative L1 instruction cache /core
- 256KB, 8-way associative L2 cache per core
- 30MB shared L3 cache (Ivy Bridge)

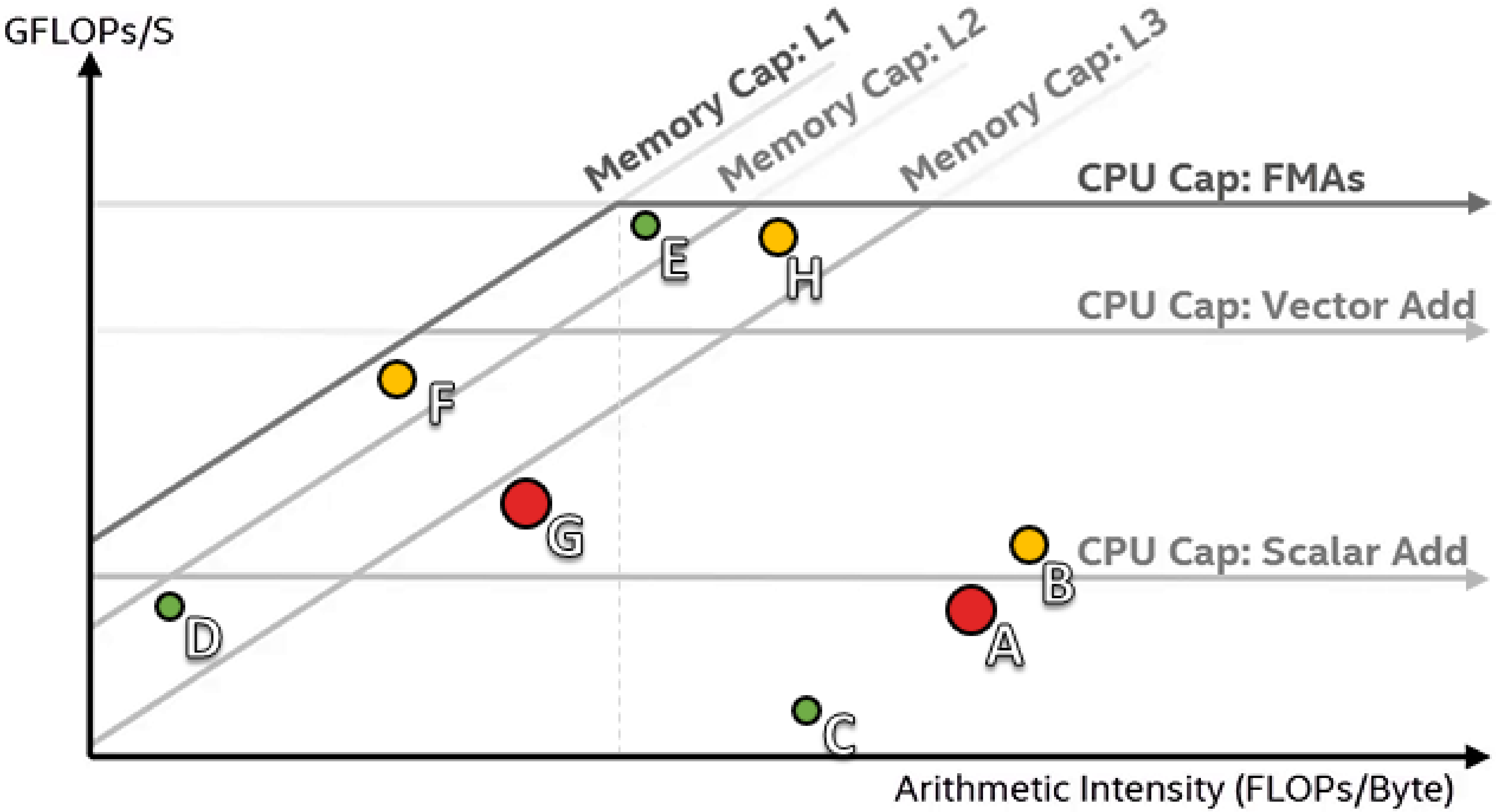
# Memory Hierarchy

| **Xeon E5-2650** | **Latency (clocks)** | **Bandwidth** | **Size (bytes)** |
|---|---|---|---|
| Registers | 1 | 8192 GB/s | ~100 |
| L1 Cache | 4 | 2048 GB/s | 32 K (data) + 32K (instruction) per core |
| L2 Cache | 11 | 744.7 GB/s | 256 K per core |
| L3 Cache | 25 | 327.7 GB/s | 20 M shared |
| Local Memory | 160 | 51.2 GB/s | 32 GB |
| QPI | 256 | 32.0 GB/s | |
| PCIe 3.0 x16 | 1024 | 16 GB/s | |
| InfiniBand QDR | 2048 | 4 GB/s | |
| SSD | 16384 | ~500 MB/s | |
| HDD | 81920 | ~100 MB/s | |

**Performance Tips**

- Hide latency
- Avoid data movement (increase *Arithmetic Intensity*)

# Roofline Model



https://software.intel.com/en-us/articles/intel-advisor-roofline

# Theoretical Peak Performance of Edison

Peak performance of Intel Xeon E5-2695 v2 processors

= 12 (cores) x 8 (AVX) x 2.4 (GHz) = 230.4 GFLOPS

Peak Performance of a compute node

= 2 (sockets) x 230.4 GFLOPS = 460.8 GFLOPS

Total Peak Performance of Edison

= 5586 (nodes) x 460.8 GFLOPS

= 2.574 PFLOPS

# Cori Login Nodes

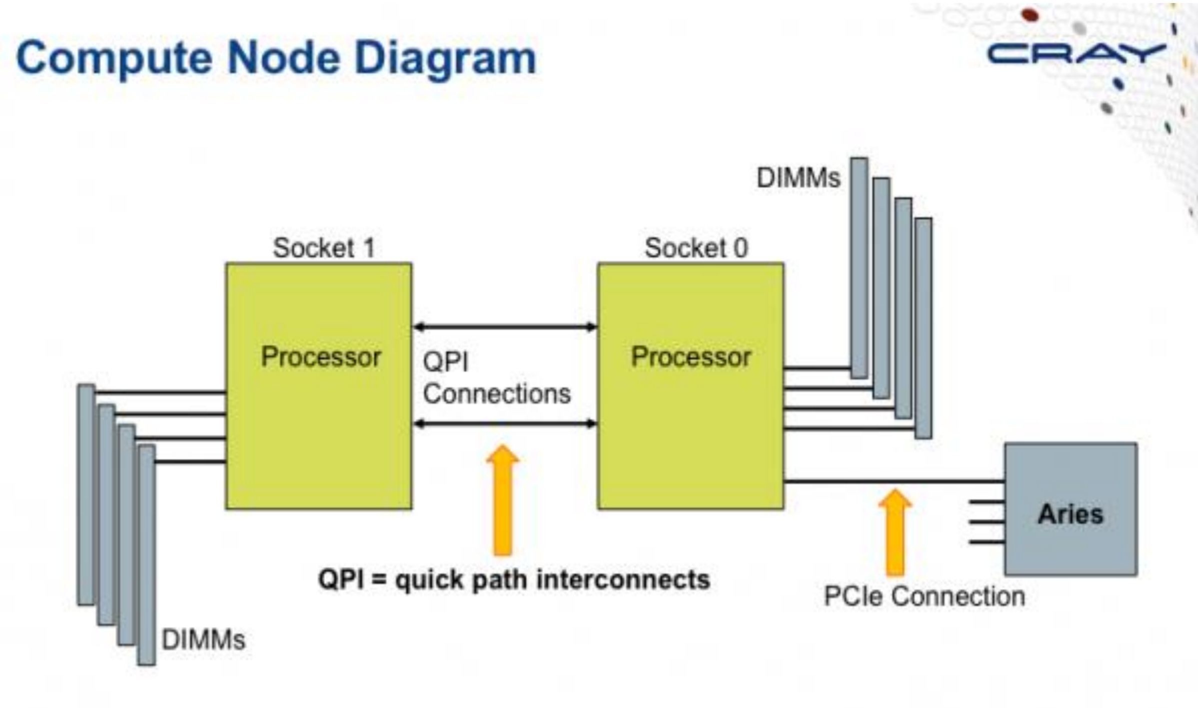
- http://www.nersc.gov/users/connecting-to-nersc/login-nodes-2/
- Login Nodes, aka Frontends or Master Nodes
  - When you ssh to Cori, you are connecting to a “Login Node”
  - 12 Login Nodes sit behind a load balancer
- Each Login Node is dual-socket server
  - 2x Intel “Haswell” Xeon 16-core E5-2698 v3 processors @ 2.3 GHz
  - 512 GB memory
  - Runs standard SUSE Linux (Enterprise Server 12)
- Intended tasks:
  - editing files
  - compiling codes
  - submitting job scripts to the batch system to be run on the Compute Nodes
  - light test runs
- Public hostname: cori.nersc.gov

# Cori Haswell Compute Nodes

- 2,388 Haswell Compute Nodes, each with:
  - Two Intel “Haswell” Xeon E5-2698 v3 16-core CPU @ 2.3 GHz
  - 128 GB DDR4 2133 MHz memory (four 16 GB DIMMs per socket)
  - An Aries Network Interface Controller (NIC)

    0.25 µs to 3.7 µs MPI latency,
    ~8GB/sec MPI bandwidth

- Compute nodes run a *lightweight* kernel and run-time environment based on SuSE Linux Enterprise Server (SLES) Linux distribution



Compute Node Diagram

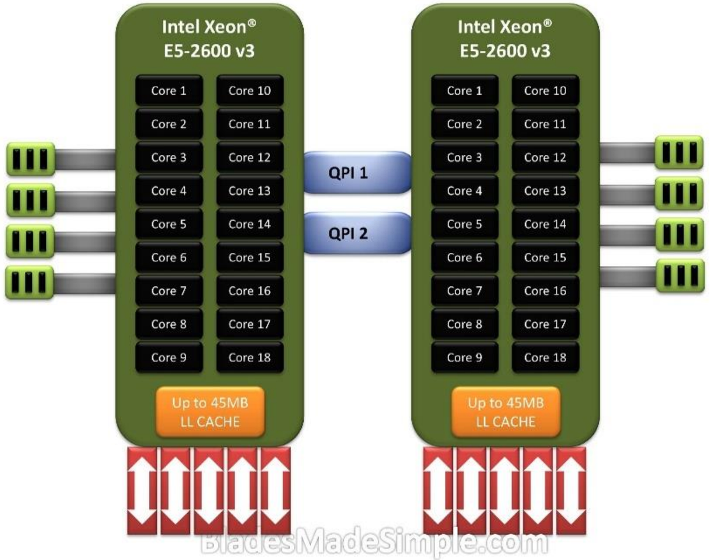
# Intel Xeon Processor E5-2698 v3

- **Microarchitecture**: Haswell
- 16 cores / 32 threads
- **Process**: 22 nm
- **Frequency**: 2.3GHz / 3.6GHz max Turbo
- **L1 cache**: 32 KB (code) + 32 KB (data) per core
- **L2 cache**: 256 KB per core
- **L3 cache**: 40 MB *shared*
- AVX2 (Advanced Vector Extensions 2)

  256-bit, **16** double-precision FLOP per cycle
- 2 QPI links @ 9.6 GT/s (4.8 GHz)

  9.6 x 2 x 20 x 64/80 / 8 = 38.4 GB/s
- 59.7 GB/s memory bandwidth @DDR3-2133

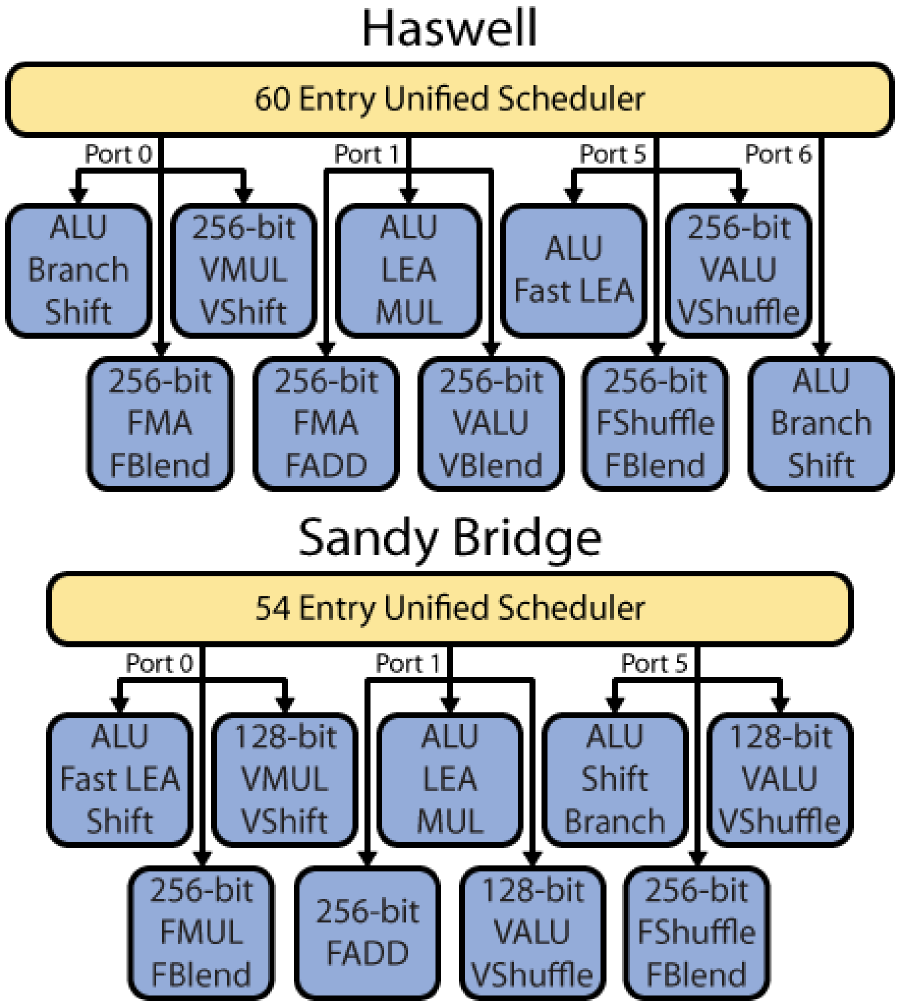
  2.133 (GT/s) x 8 (Byte) x 4 (channels) = 68.3 GB/s



http://www.cpu-upgrade.com/CPUs/Intel/Xeon/E5-2698_v3.html

# Haswell Microarchitecture

- Instruction-Level Parallelism (ILP)
  - 4 integer ALU, 2 vector ALU, 3 AGU and 2 branch execution units per core
  - 2 load/store operations per CPU cycle for each memory channel
  - 4 branch predictors
  - hyper-threading (2 logical core per physical core)
  - decoded micro-operation cache (uop cache)
  - 14- to 19- stage instruction pipeline, depending on the uop cache hit or miss
- AVX2 (Advanced Vector Extensions 2)
  - 256-bit Integer vectors
  - FMA: Fused Multiply-Add -> 8 DP FLOPS
  - 2 FP functional units -> 16 double-precision FLOPS



http://www.realworldtech.com/haswell-cpu/

# Theoretical Peak Performance of Cori Phase I

Peak performance of Intel Xeon E5-2698 v3 processor

= 16 (cores) x 16 (AVX2) x 2.3 (GHz) = 588.8 GFLOPS

Peak Performance of a Haswell compute node

= 2 (sockets) x 588.8 GFLOPS = 1177.6 GFLOPS

= 1.2 TFLOPS

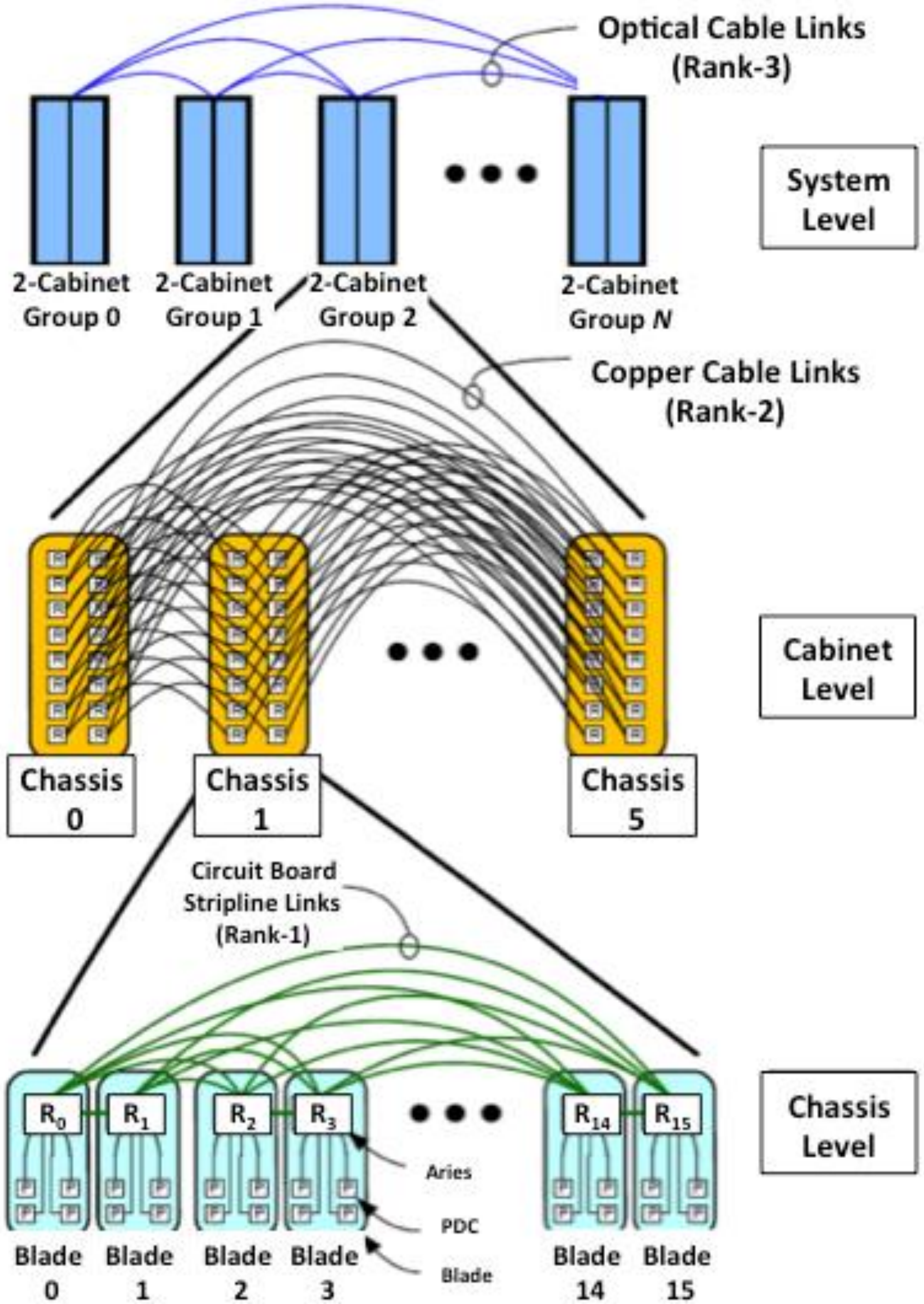
Total Peak Performance of Edison

= 2388 (nodes) x 1.2 TFLOPS

= 2.9 PFLOPS

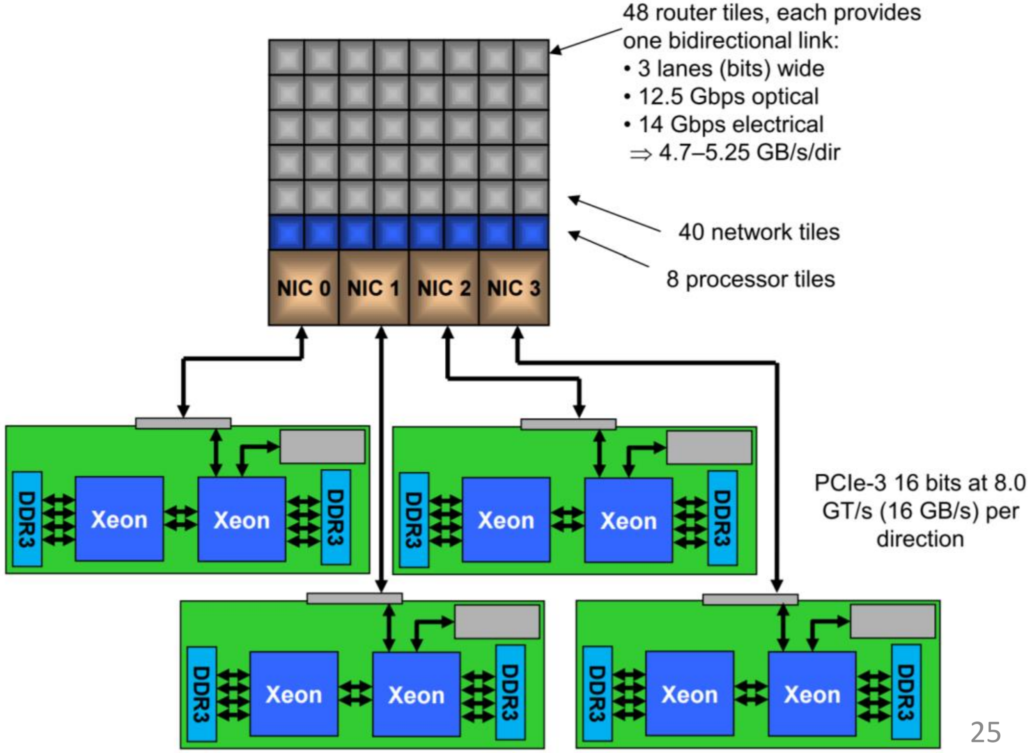
# Cray Aries Interconnect with Dragonfly Topology

- Cray Cascade: a Scalable HPC System based on a Dragonfly Network
- Aries interconnect ASIC
- 3-dimensional all-to-all topology
  - Rank 1 – Green Dimension – 16 Aries within a chassis
  - Rank 2 – Black Dimension – from each Aries within a chassis to its 5 peers in each of the other chassis within a group
  - Rank 3 – Blue Dimension – connecting Dragonfly groups together



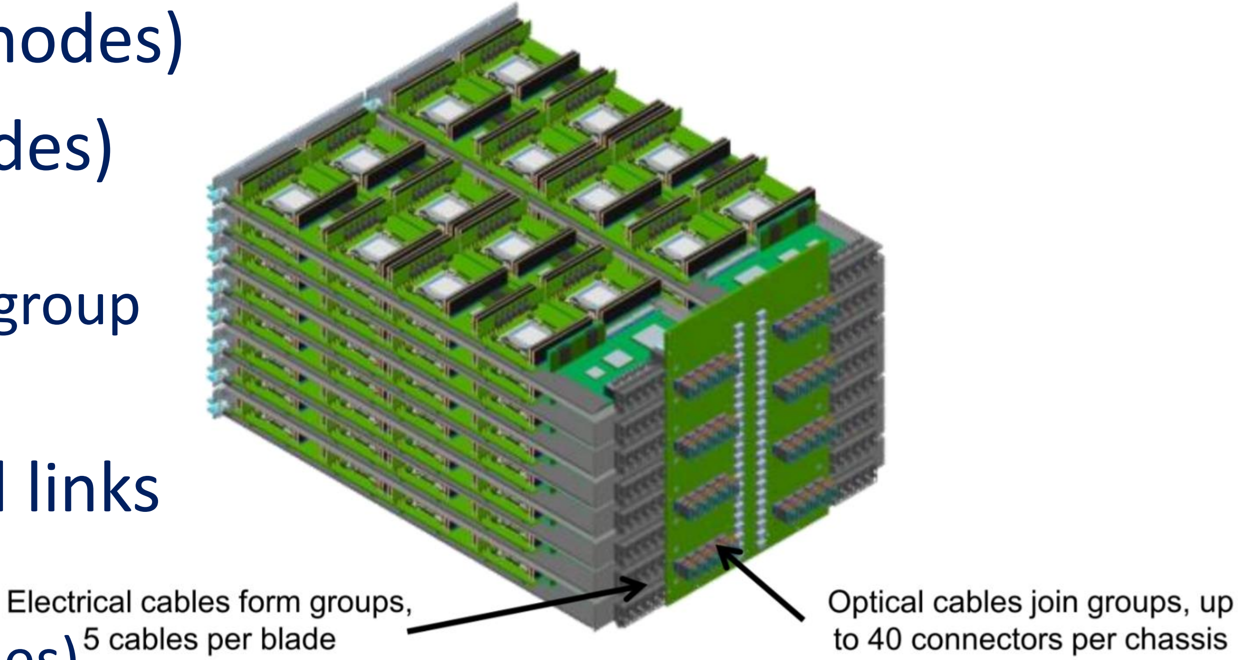
# Aries ASIC

- A Cray Cascade blade consists of 4 dual-socket nodes, and a single Aries ASIC
- Aries ASIC combines:
  - 4 Network Interface Controllers (NICs), each to a node via PCIe-3 x16 host interface (16 GB/s/direction)
  - 48-port high radix router



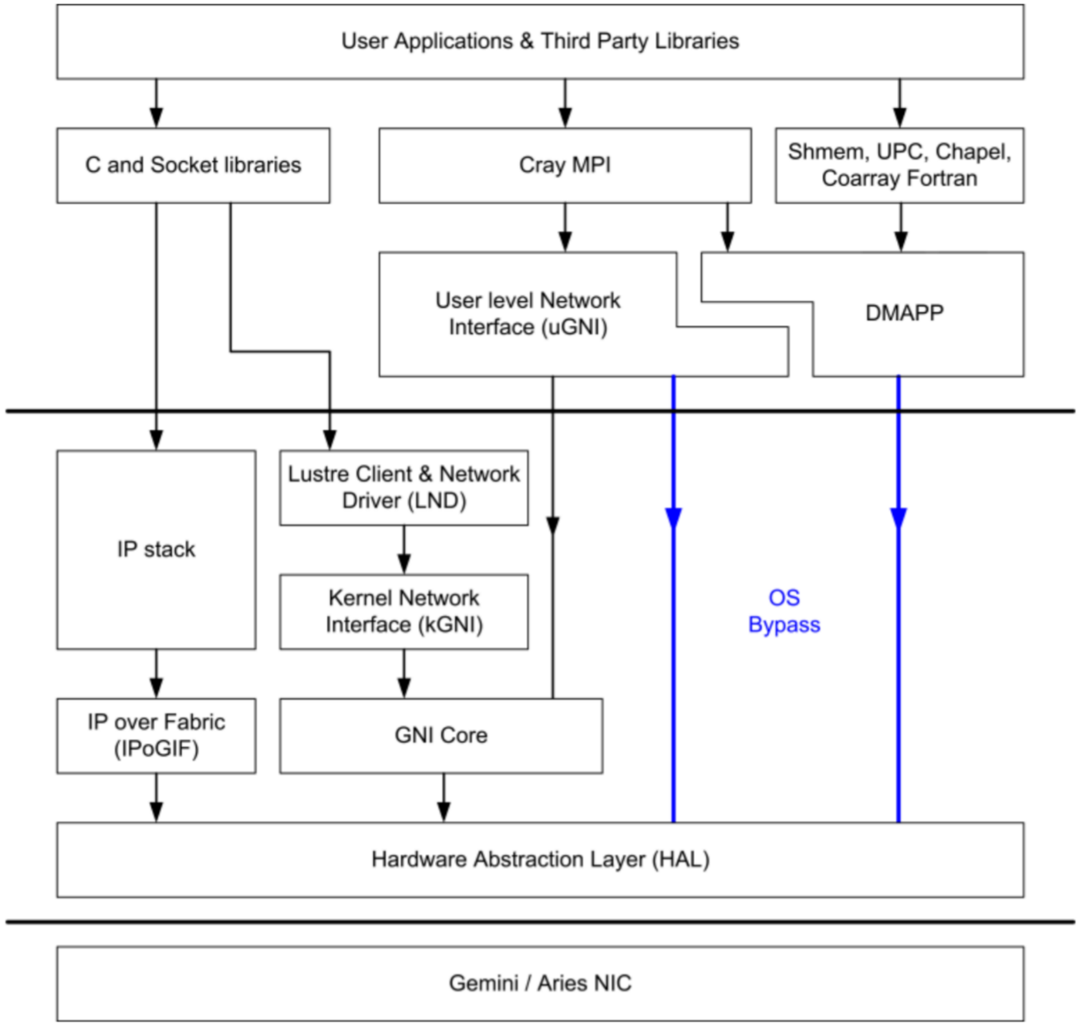
# Node Organization

- Each compute **blade** has 4 **nodes**, and an Aries ASIC
- Each **chassis** has 16 compute blades (64 nodes)
  - The chassis backplane provides all-to-all connectivity between the blades
  - Rank 1 – Green Dimension
- Each **cabinet** has 3 chassis (192 nodes)
- 2 cabinets form a **group** (384 nodes)
  - Each blade provides 5 electrical connectors linking to peers in the group
  - Rank 2 – Black Dimension
- Groups are connected via optical links
  - Rank 3 – Blue Dimension
  - Maximum 241 groups (92,544 nodes)


Electrical cables form groups, 5 cables per blade
Optical cables join groups, up to 40 connectors per chassis

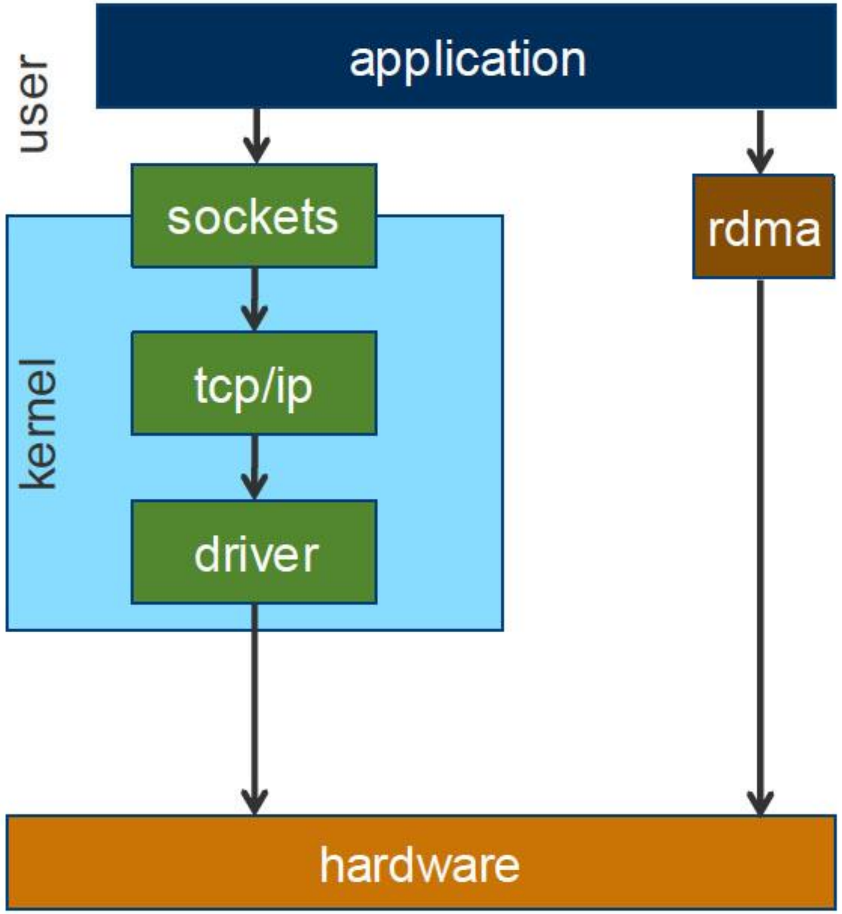
# Cray Network Software Stack

- Cray Cascade supports
  - both **kernel communication**, through a Linux device driver
  - and **direct user space communication**, where the driver is used to establish communication domains and handle errors, but is bypassed for data transfer.
- Parallel applications
  - MPI
  - SHMEM
  - PGAS, such as CAF, UPC, or Chapel

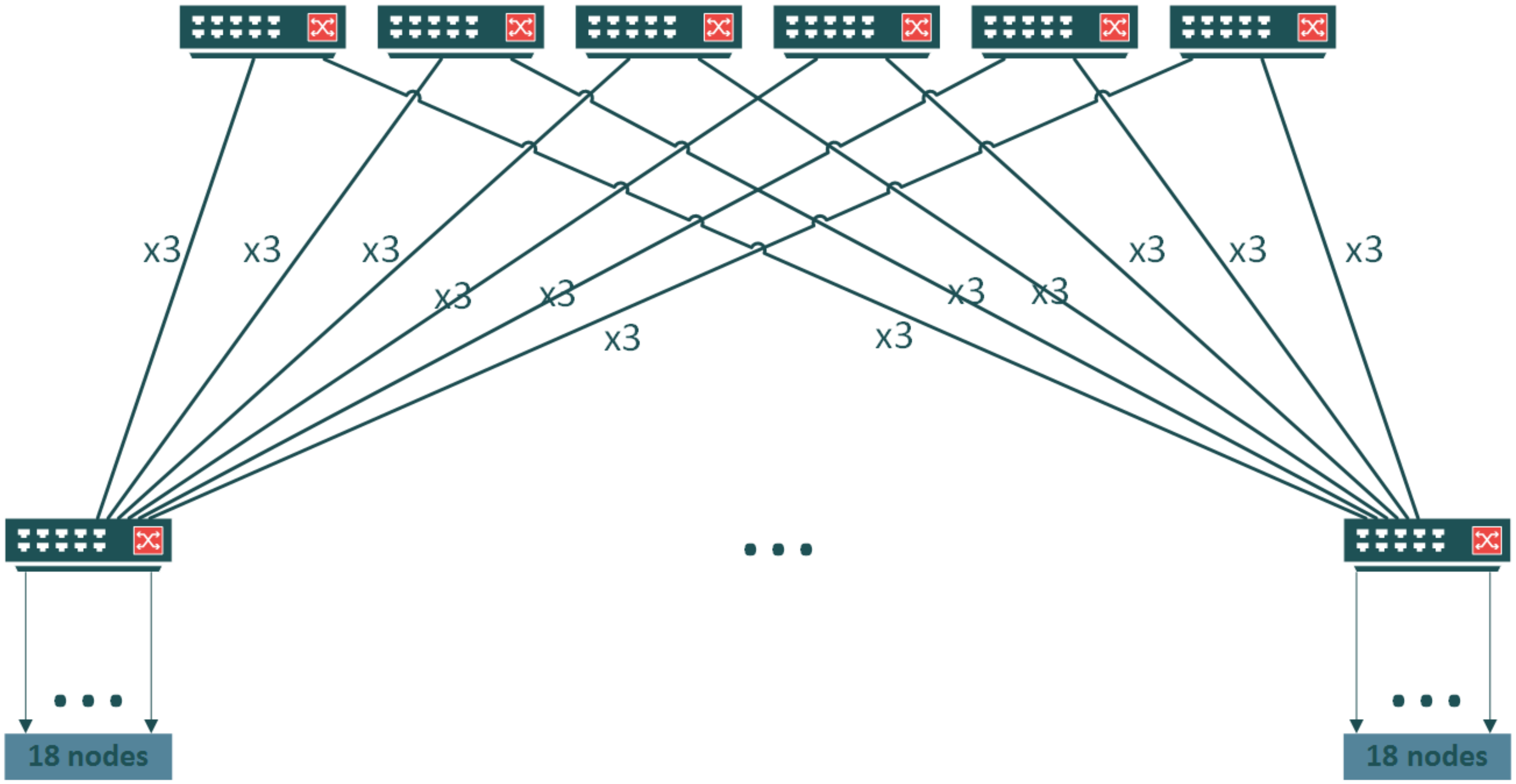

User Applications & Third Party Libraries
C and Socket libraries
Cray MPI
Shmem, UPC, Chapel, Coarray Fortran
User level Network Interface (uGNI)
DMAPP
IP stack
Lustre Client & Network Driver (LND)
Kernel Network Interface (kGNI)
OS Bypass
IP over Fabric (IPoGIF)
GNI Core
Hardware Abstraction Layer (HAL)
Gemini / Aries NIC

# Hyades InfiniBand Interconnect

- **RDMA** – Remote Direct Memory Access
- Kernel bypass & low CPU overhead
- Mellanox ConnectX-2 VPI QDR 4x InfiniBand HCA (host channel adapter)
  - Link Bandwidth: 40Gbps signaling rate & 32Gbps data rate (8/10 encoding)
  - Link Latency: 1.3 microseconds
- Used for:
  - Message Passing (MPI)
  - Lustre Network (LNET)
  - IPoIB (IP over InfiniBand) – subnet 10.8.0.0/16
- InfiniBand fabric for Hyades
  - 18x 36-port Mellanox IS5024 QDR IB switches
  - 1:1 non-blocking fat-tree topology



# 1:1 non-blocking fat-tree topology



6 core switches
(each with 36 ports)

12 leaf switches
(each with 36 ports)

216 nodes

bisection bandwidth = 108 x 40 Gb/s = 4320 Gb/s

# NERSC File Systems

On each machine you have access to at least three different file systems (http://www.nersc.gov/users/storage-and-file-systems/file-systems/)

- **Home**: Permanent, relatively small storage for data like source code, shell scripts, etc. that you want to keep. This file system is *not tuned for high performance* from parallel jobs. Referenced by the environment variable **$HOME**.
- **Scratch**: Large, *high-performance* file system. Place your large data files in this file system for capacity and capability computing. Data is routinely purged, so you must save important files elsewhere (like HPSS). Referenced by the environmental variable **$SCRATCH**.
- **Project**: Large, permanent, *medium-performance* file system. Project directories are intended for sharing data within a group of researchers.
- **Burst Buffer**: temporary, *flexible high-performance* SSD file system that sits within the High Speed Network (HSN) on *Cori*. Accessible only from compute nodes, the Burst Buffer provides per-job (or short-term) storage for I/O-limited codes.

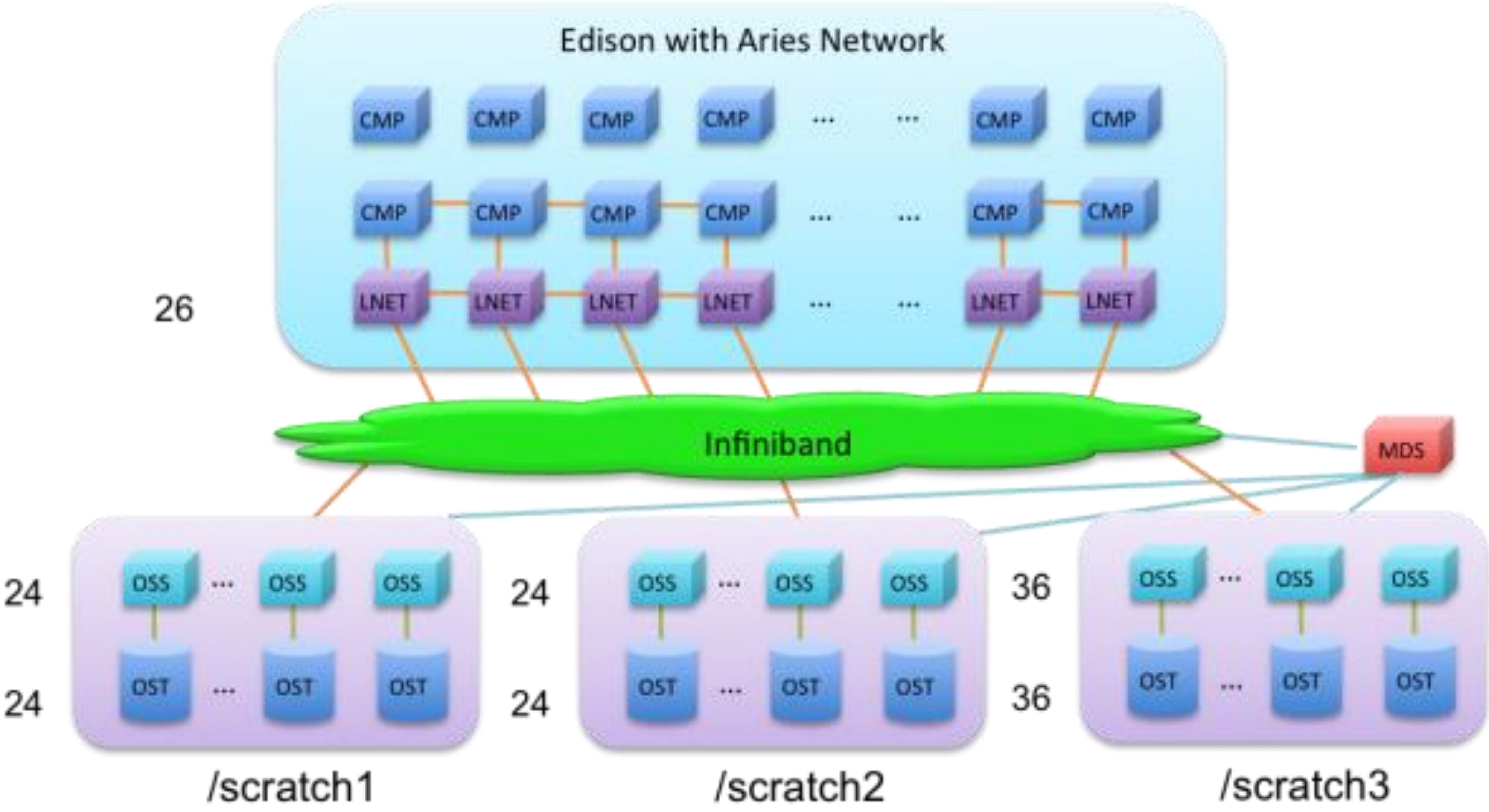
# Summary of File System Policies

| File System | Path | Type | Peak Performance | Default Quota | Backups | Purge Policy |
|---|---|---|---|---|---|---|
| Global homes | $HOME | GPFS | Not For IO Jobs | 40 GB<br>1,000,000 Inodes | Yes | Not purged |
| Global project | /project/projectdirs/*projectname* | GPFS | 130GB/Second | 1 TB<br>1,000,000 Inodes | Yes if ≤ 5 TB quota. No if quota is > 5 TB. | Not purged |
| Edison local scratch | $SCRATCH | Lustre | 168GB/Second (across 3 systems) | 10 TB<br>5,000,000 Inodes | No | Files not accessed for 8 weeks are deleted |
| Cori local scratch | $SCRATCH | Lustre | 700GB/Second | 20 TB<br>10,000,000 Inodes | No | Files not accessed for 12 weeks are deleted |
| Cori Burst Buffer | $DW_JOB_STRIPED,<br>$DW_PERSISTENT_STRIPED_*XXX* | DataWarp | 1.7 TB/s, 28M IOP/s | none | No | Data is deleted at the end of every job, or at the end of the lifetime of the persistent reservation |

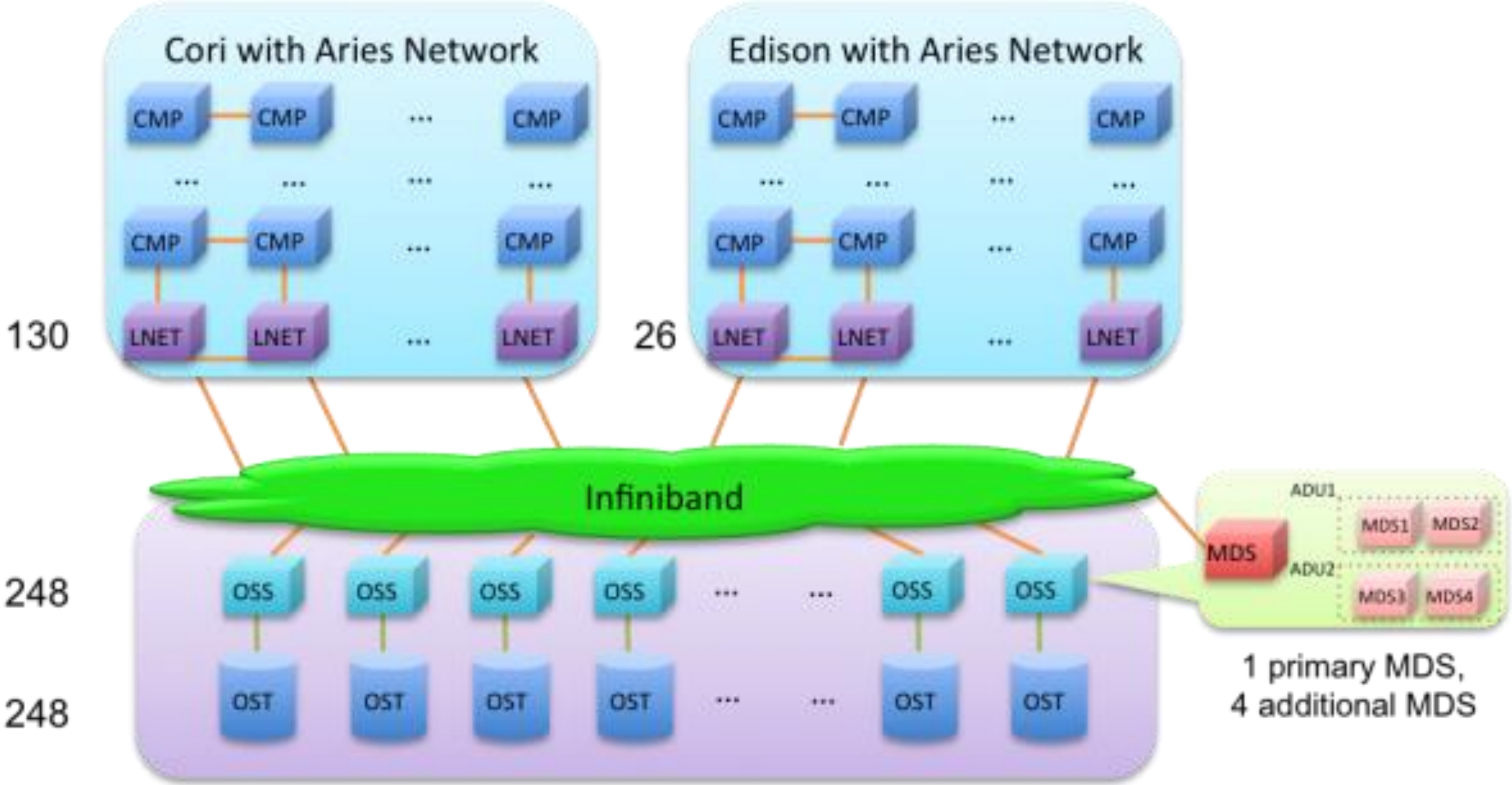
# Lustre File System

- http://lustre.org/
- Lustre is a high-performance parallel file system, used by more than 60 of the top 100 fastest supercomputers in the world
- A Lustre file system has three major functional units:
  - One or more **metadata servers** (**MDS**es) that has one or more **metadata targets** (**MDT**s) per Lustre file system that stores namespace metadata, such as filenames, directories, access permissions, and file layout.
  - One or more **object storage servers** (**OSS**s) that store file data on one or more **object storage targets** (**OST**s). The capacity of a Lustre file system is the sum of the capacities provided by the OSTs.
  - **Lustre Network** (LNET) layer can use several types of network interconnects, including native InfiniBand verbs, TCP/IP on Ethernet, etc.
  - Clients that access and use the data. Lustre presents all clients with a unified namespace for all of the files and data in the file system, using standard POSIX semantics, and allows concurrent and coherent read and write access to the files in the file system.

# Edison Scratch



# Cori Scratch



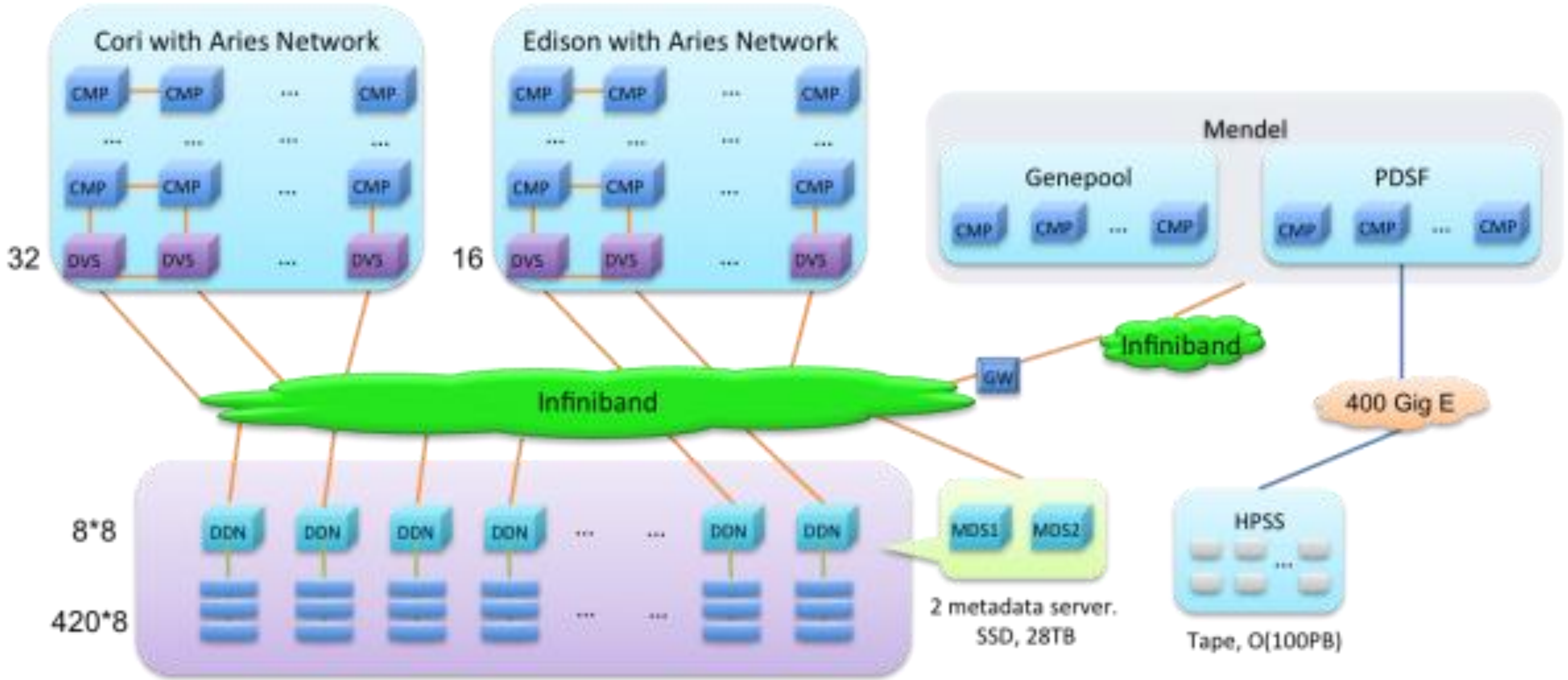
1 primary MDS,
4 additional MDS

Each OSS controls one OST. The Infiniband connects the MDS, ADUs and OSSs to the LNET routers on the Cray XC System. The OSTs are configured with GridRAID, similar to RAID6, (8+2), but can restore failure 3.5 times faster than traditional RAID6. Each OST consists of 41 disks, and can deliver 240TB capacity.

# GPFS

- The **General Parallel File System** (**GPFS**) is a high-performance clustered file system developed by IBM
- GPFS provides *concurrent* high-speed file access to applications executing on multiple nodes of clusters. GPFS provides higher IO performance by striping blocks of data from individual files over multiple disks, and reading and writing these blocks in parallel
- Features of GPFS:
  - Distributed metadata
  - Efficient indexing of directory entries for very large directories
  - Distributed locking,  which allows for full Posix filesystem semantics, including locking for exclusive file access.
  - Partition Aware
  - Filesystem maintenance can be performed online

# Global Project



2 metadata server. SSD, 28TB

Tape, O(100PB)

8 DDN (SF 12KE), 8 VMs and 420 disks (RAID6) per DDN. Peak aggregate performance is 128GB/s (16GB/s x 8). Total capacity is about 6.2 PB

# Accessing NERSC Supercomputers

You must use ssh protocol!

Edison Logins Nodes (edison.nersc.gov):

```
ssh -l username edison.nersc.gov
ssh username@edison.nersc.gov
```

Cori Login Nodes (cori.nersc.gov):

```
ssh -l username cori.nersc.gov
ssh username@cori.nersc.gov
```

# SSH clients

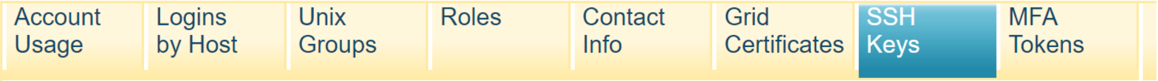
- On Unix-like system, including OS X and Linux, OpenSSH server and client are commonly present
- On Windows
  - You can install a Unix-like environment, such as
    - Cygwin (https://www.cygwin.com/)
    - MinGW/MSYS (http://www.mingw.org/)
    - Git for Windows (https://git-for-windows.github.io/)
    - Windows Subsystem for Linux (https://msdn.microsoft.com/en-us/commandline/wsl/install_guide)
  - Or native SSH clients, such as
    - PuTTY (http://www.chiark.greenend.org.uk/~sgtatham/putty/download.html)
    - WinSCP (https://winscp.net/eng/download.php)

# Using SSH Key for Authentication

Generate a pair of SSH keys on your client computer (Unix-like):

```
cd $HOME/.ssh
ssh-keygen –b 2048 –t rsa –f nersc
```

Add the public key (*nersc.pub*) to your account via NERSC Information Management system ([https://nim.nersc.gov/](https://nim.nersc.gov/))



Then you can log in in with your private key (*nersc*):

```
ssh -i ~/.ssh/nersc -l username edison.nersc.gov
ssh -i ~/.ssh/nersc -l username cori.nersc.gov
```

[https://pleiades.ucsc.edu/hyades/SSH](https://pleiades.ucsc.edu/hyades/SSH)

# Using SSH Key for Authentication (cont'd)

Add the following to *$HOME/.ssh/config* on your computer

```
host edison
    HostName edison.nersc.gov
    User username
    IdentityFile ~/.ssh/nersc
host cori
    HostName cori.nersc.gov
    User username
    IdentityFile ~/.ssh/nersc
```

Then a lot of keystrokes will be saved:

```
ssh edison
ssh cori
```

# GUI (X Windows) Applications

http://www.nersc.gov/users/connecting-to-nersc/using-x-windows/

X Windows allows you to display remote GUI applications on your local computer screen. 2 ways to run remote GUI applications:

1. X forwarding using SSH
   - An X server is running on your local computer
   - X forwarding is enabled in SSH ("-X" or "-Y" option for OpenSSH client)
2. NX: NX can greatly improve the performance of the X Windows, to the point that it can be usable over a slow link
   - Download NX Player: http://www.nersc.gov/users/connecting-to-nersc/using-nx/download-tested-nx-player/
   - Download NX configuration file: http://www.nersc.gov/users/connecting-to-nersc/using-nx/nxconfig-2/

# NERSC User Environment

http://www.nersc.gov/users/software/nersc-user-environment/

- Home Directories
  - All NERSC systems use global home directories, regardless of the platform.
  - You should refer to your home directory using the environment variable **$HOME**.
  - Optimized for small files. **Do not run jobs in your home directory.**
- Shells
  - NERSC fully supports bash, csh, and tcsh as login shells.
  - The default shell on Edison is **csh**; however the default shell on Cori is **bash**!
  - You can change your default shell using NIM (https://nim.nersc.gov/)
- Dotfiles
  - The "standard" dotfiles are symbolic links to read-only files.
  - For each standard dotfile, there is a user-writable ".ext" file.

# Customization of User Environment

- By modifying “.ext” files
  - **C shell**: .login.ext & .cshrc.ext
  - **Bash**: .bash_profile.ext & .bashrc.ext
- Use the environment variable $NERSC_HOST to make platform-specific customization. For example, if your shell is csh, to change the default programming environment on Edison (.cshrc.ext):

```
if ($NERSC_HOST == "edison") then
  module swap PrgEnv-intel PrgEnv-gnu
endif
```

# Edison File Storage

http://www.nersc.gov/users/computational-systems/edison/file-storage-and-i-o/

- Home
  - $HOME
  - GPFS file system shared with all NERSC systems, not for IO intensive applications
- Local Scratch
  - $SCRATCH
  - Lustre file systems, Edison only, for running production applications
  - Peak performance: 168GB/sec
- Project
  - /project/projectdirs/m2744
  - GPFS global file system mounted on all NERSC systems
- Global Scratch
  - $CSCRATCH
  - Lustre file system shared by Cori and Edison, for running production applications
  - Peak performance: >700GB/sec

# Cori File Storage

http://www.nersc.gov/users/computational-systems/cori/file-storage-and-i-o/

- Home
  - ➢$HOME
  - ➢GPFS file system shared with all NERSC systems, not for IO intensive applications
- Local Scratch
  - ➢$SCRATCH
  - ➢Lustre file system, also mounted on Edison as $CSCRATCH
  - ➢For running production applications and I/O intensive jobs
  - ➢Peak performance: >700GB/sec
- Project
  - ➢/project/projectdirs/m2744
  - ➢GPFS global file system mounted on all NERSC systems
  - ➢Peak performance: 40GB/s

# Modules Software Environment

http://www.nersc.gov/users/software/nersc-user-environment/modules/

NERSC uses the *module* utility to manage nearly all software.

Advantages:

1. NERSC can provide many different software and many different versions of the same software (including a default version as well as several older and newer versions)
2. Users can easily switch to different software or version without having to explicitly specify different paths.

The Modules utility consists of 2 parts:

1. The *module* command interface
2. The *modulefiles* on which *module* operates

# Modulefile

- Written in TCL (Tool Command Language)
- Begins with the magic cookie *#%Module*
- Sets or adds environmental variables, like PATH, LD_LIBRARY_PATH, etc.
- Hides the notion of different type of shells (sh, bash, csh, tcsh, ksh, zsh, etc.)
- You can write your own, private *modulefiles*
- http://modules.sourceforge.net/man/modulefile.html

# A Sample Modulefile

```
#%Module1.0

proc ModulesHelp { } {
    global version
    puts stderr "\n\tp_HDF5_impi_intel Version $version"
}

module-whatis   "Set up environment for Parallel HDF5 compiled
with Intel MPI"

# for Tcl script use only
set  version  1.8.10-patch1
set  root     /pfs/sw/parallel/impi_intel/hdf5-${version}

prepend-path  PATH          $root/bin
prepend-path  LIBRARY_PATH  $root/lib
prepend-path  INCLUDE       $root/include

setenv  I_MPI_EXTRA_FILESYSTEM       on
setenv  I_MPI_EXTRA_FILESYSTEM_LIST  lustre

conflict hdf5
```

# Module Command Interface

To learn the usage of each sub-command:

```
module help
```

To list the loaded modules:

```
module list
```

To list all available modulefiles in the current $MODULEPSATH:

```
module avail
```

# Module Command Interface (cont'd)

To list all available modulefiles whose names start with, e.g., *PrgEnv*:

```
module avail PrgEnv
```

```
--------------------- /opt/cray/modulefiles -----------------------
PrgEnv-cray/5.0.41           PrgEnv-gnu/5.2.25
PrgEnv-cray/5.1.18           PrgEnv-gnu/5.2.40
PrgEnv-cray/5.1.29           PrgEnv-gnu/5.2.56(default)
PrgEnv-cray/5.2.25           PrgEnv-intel/5.0.41
PrgEnv-cray/5.2.40           PrgEnv-intel/5.1.18
PrgEnv-cray/5.2.56(default)  PrgEnv-intel/5.1.29
PrgEnv-gnu/5.0.41            PrgEnv-intel/5.2.25
PrgEnv-gnu/5.1.18            PrgEnv-intel/5.2.40
PrgEnv-gnu/5.1.29            PrgEnv-intel/5.2.56(default)
```

# Module Command Interface (cont’d)

To list all available modulefiles whose names start with, e.g., *python*:

```
module avail python
```

```
------------------------------- /opt/cray/modulefiles --------------------------------
python3/3.3.2

-------------------------- /usr/common/software/modulefiles --------------------------
python/2.7                 python/3.4                 python_base/2.7.3
python/2.7-anaconda        python/3.4-anaconda        python_base/2.7.5
python/2.7.3               python/3.5-anaconda        python_base/2.7.9(default)
python/2.7.5               python-libs/2.7.3
python/2.7.9(default)      python-libs/2.7.9(default)
```

# Module Command Interface (cont’d)

To print the module-specific help information for, e.g., the module python/3.5-anaconda:

```
module help python/3.5-anaconda
```

To display the information about a module (full path of the modulefile and environment changes)

```
module show python/3.5-anaconda
module display python/3.5-anaconda
```

To load, e.g., the module python/2.7.9 into the shell environment:

```
module load python/2.7.9
```

Or simply (since it is the *default*):

```
module load python
```

To switch the loaded, e.g., module python/2.7.9 with, e.g., module python/3.5-anaconda:

```
module swap python/2.7.9 python/3.5-anaconda
module switch python/2.7.9 python/3.5-anaconda
```

# Module Command Interface (cont'd)

To remove a loaded module from the shell environment:

```
module rm python/3.5-anaconda
module unload python/3.5-anaconda
```

What the heck do you intend to do with this command?

```
module swap python python
```

To load you own private module (with absolute path):

```
module load $HOME/modulefiles/python-3.6.1
```

Further reading:
http://modules.sourceforge.net/man/module.html

# Default Environment

- A few modules are loaded by default, including *PrgEnv-intel*:

```
shawdong@cori07:~> module list
```

- You can modify your environment so that certain modules are loaded whenever you log in. Put your changes in one of the following files, depending on your shell:

  .cshrc.ext (C shell) or .bashrc.ext (Bash)

- For example, if your shell is **bash**, to change the default programming environment on Cori (.bashrc.ext):

```
if [ $NERSC_HOST == "cori" ]; then
  module swap PrgEnv-intel PrgEnv-cray
fi
```

# Stages of Compilation

We’ve all done this before:

```
$ gcc hello.c
$ ls
a.out  hello.c
$ ./a.out
Hello, world!
```

```
#include <stdio.h>

int main()
{
  printf("Hello, world!\n");
  return 0;
}
```

What happens behind the scene:

```
$ gcc -save-temps hello.c
$ ls
a.out  hello.c hello.i  hello.o  hello.s
```

4 stages of compilation:

preprocessing -> compilation -> assembling -> linking

http://althing.cs.dartmouth.edu/local/www.acm.uiuc.edu/sigmil/RevEng/ch02.html

# 4 Stages of Compilation

1. **Preprocessing**, performing the following tasks:
   - Macro substitution
   - Stripping comments off
   - Expansion of the included files

   ```
   $ gcc -E hello.c -o hello.i
   ```

   or:

   ```
   $ cpp hello.c -o hello.i
   ```

2. **Compilation**, or *Parsing and Translation* stage, turning source code into assembly (*hello.s*):

   ```
   $ gcc -S hello.c
   ```

# 4 Stages of Compilation (cont'd)

3. **Assembling**, translating assembly code to object file (*hello.o*):

```
$ gcc -c hello.s
```

or:

```
$ as hello.s
```

We often combine stages:

```
$ gcc -c hello.c
$ gcc -c hello.i
```

4. **Linking**, which links against libraries and generates an executable file (*a.out* by default):

```
$ gcc hello.o
```

What it does behind the scene:

```
$ gcc -v hello.o
...
 /usr/libexec/gcc/x86_64-redhat-linux/4.4.7/collect2 ...
```

# Quiz: which stages do those options belong to?

This is typically how we compile an MPI program:

```
icc –DNDEBUG -O2 -mp1 mpi_hostname.c -o mpi_hostname.x
-I/opt/intel/impi/4.1.3.045/intel64/include
-L/opt/intel/impi/4.1.3.045/intel64/lib
-Xlinker --enable-new-dtags -Xlinker -rpath
-Xlinker /opt/intel/impi/4.1.3.045/intel64/lib
-Xlinker -rpath -Xlinker /opt/intel/mpi-rt/4.1 -lmpigf
-lmpi -lmpigi -ldl -lrt -lpthread
```

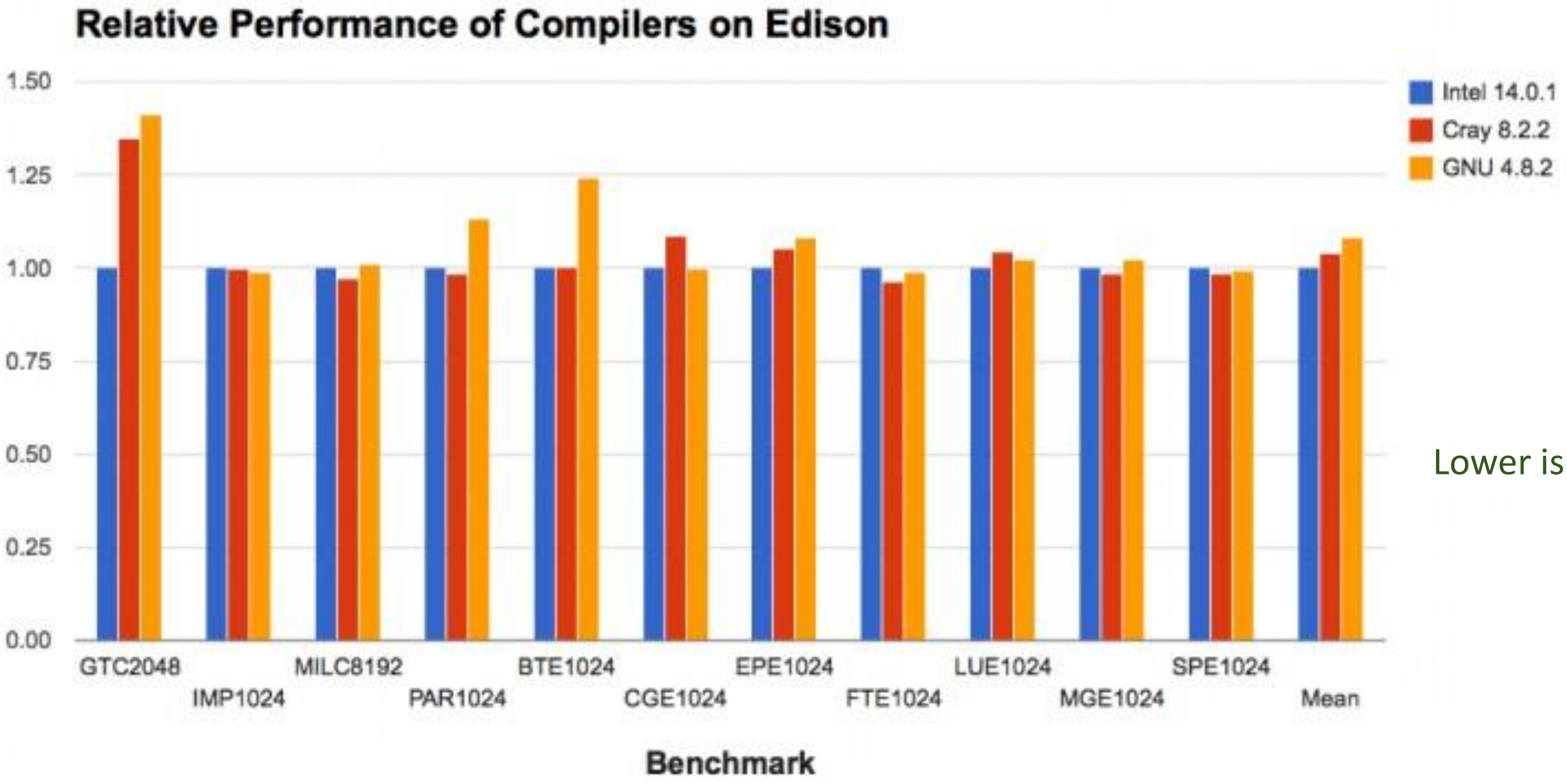
Which stages do those options belong to?

- `–DNDEBUG -I/opt/intel/impi/4.1.3.045/intel64/include:` preprocessing
- `-O2 -mp1:` compilation
- `-L/opt/intel/impi/4.1.3.045/intel64/lib –Xlinker ...:` linking

# Compilers on NERSC Systems

http://www.nersc.gov/users/software/compilers/

- Intel Compilers
  - Module *PrgEnv-intel*, default on Edison and Cori
  - The Intel compiler suite offers C, C++ and Fortran compilers with optimization features and multithreading capabilities, highly optimized performance libraries, and error-checking, security, and profiling tools.
- Cray Compilers
  - Module *PrgEnv-cray*
  - Fortran, C, C++, UPC (Unified Parallel C) and CAF (Coarray Fortran) compilers
- GCC (GNU Compiler Collection)
  - Module *PrgEnv-gnu*
- LLVM/Clang



Lower is better

# Compiling Codes on NERSC Systems

http://www.nersc.gov/users/computational-systems/cori/programming/compiling-codes-on-cori/

Basic Examples:

- For **Fortran** source code, use ftn

```
ftn -o example.x example.f90
```

- For **C** source code, use cc

```
cc -o example.x example.c
```

- For **C++** source code use CC

```
CC -o example.x example.C
```

Here **ftn**, **cc** & **CC** are wrappers of underlying compilers (Intel, Cray or GNU). For example, **cc** is wrapper of **icc** (Intel C compiler)

# Compiler Wrappers

```
edison05 ams250/codes> cc -craype-verbose -o hello.x hello.c

icc -mavx -static -D__CRAYXC ... -o hello.x hello.c
-I/opt/cray/mpt/7.4.1/gni/sma/include ...
-L/opt/cray/mpt/7.4.1/gni/sma/lib64 ...
-Wl,-u,MPI_Init,-u,MPI_Wtime,-u,__wrap_H5Fcreate,...
-lpthread -lsma -lpmi -ldmapp -lpthread -lsci_intel_mpi ...
-Wl,--as-needed,-lpthread,--no-as-needed
```

```
edison05 ams250/codes> man ftn
NAME
       ftn - Invokes the Fortran compiler in the currently loaded programming
       environment

SYNOPSIS
       ftn [-default64] [ Cray_options | PGI_options| GCC_options | Intel_options ]
        files [-craype-verbose][-dynamic][-help][-shared][-static]
```

# Static Linking

By default, executables are **staticly** linked on Cray systems (because compute nodes only run a lightweight kernel and run-time environment).

```
edison05 ams250/codes> ldd hello.x
        not a dynamic executable
```

What would be the output on **Hyades**?

By contrast, executables are by default **dynamically** linked in modern Linux. Only the *names* of sharable libraries are placed in the executable image. Actual linking with the library routines does not occur until the image is run, when both the executable and the libraries are loaded in memory.

# Compilers on Hyades

- Intel Compilers (default and recommended)
  - User and Reference Guide for the Intel C++ Compiler 15.0: https://software.intel.com/en-us/compiler_15.0_ug_c
  - User and Reference Guide for the Intel Fortran Compiler 15.0: https://software.intel.com/en-us/compiler_15.0_ug_f
- PGI Compilers
  - PGI documentation: https://www.pgroup.com/resources/docs.htm
  - PGI Compiler User's Guide: https://www.pgroup.com/doc/pgiug.pdf
  - PGI Compiler Reference Manual: https://www.pgroup.com/doc/pgiref.pdf
- GCC (GNU Compiler Collection)
  - GCC documentation: https://gcc.gnu.org/onlinedocs/
  - GNU Fortran Compiler: https://gcc.gnu.org/onlinedocs/gfortran/

# Compilers on Hyades (cont'd)

| | **Intel** | **PGI** | **GNU** |
|---|---|---|---|
| C Compiler | icc | pgcc | gcc |
| C++ Compiler | icpc | pgCC<br>pgcpp<br>pgc++ (GNU-compatible) | g++ |
| Fortran Compiler | ifort | pgfortran<br>pgf77 (Fortan 77)<br>pgf90 (Fortan 90/95)<br>pgf95 (Fortan 90/95) | gfortran |

1. The Intel and PGI C compilers are mostly object compatible (ABI compatible) with gcc, except for some OpenMP contructs;
2. C++ compilers are generally not ABI compatible; but Intel C++ is mostly compatible with g++;
3. Fortran compilers are generally not ABI compatible. Even different versions of gfortran may not be ABI compatible.

# Basic Levels of Optimization for Intel Compilers

| Level | Description |
|---|---|
| -O0 | Fast compilation, full debugging support; equivalent to -g |
| -O1<br>-O2<br>(*default*) | Low to moderate optimization, partial debugging support:<br>• instruction rescheduling<br>• copy propagation<br>• software pipelining<br>• common subexpression elimination<br>• prefetching, loop transformations |
| -O3 | Aggressive optimization - compile time/space intensive and/or marginal effectiveness; may change code semantics and results (sometimes even breaks code!):<br>• enables -O2<br>• more aggressive prefetching, loop transformations |

# Some Important Options for Intel Compilers

| Option | Description |
|---|---|
| -c | For compilation of source file only. |
| -fast | Maximizes speed across the entire program (always use with -no-ipo) |
| -xHost | Generates instructions for the highest instruction set available on the compilation host processor. |
| -xAVX | Optimizes for Intel processors that support AVX (Advanced Vector Extensions) instructions. |
| -xCORE-AVX2 | Optimizes for Intel processors that support AVX2 instructions (Haswell processors). |
| -g | Debugging information, generates symbol table. |
| -mp | Maintain floating point precision (disables some optimizations). |
| -mp1 | Improve floating-point precision (speed impact is less than -mp). |
| -ip | Enable single-file interprocedural (IP) optimizations (within files). |
| -ipo | Enable multi-file IP optimizations (between files). |
| -prefetch | Enables data prefetching (requires -O3). |
| -qopenmp | Enable the parallelizer to generate multi-threaded code based on the OpenMP directives. |

# NERSC Recommendations

- Use default level of optimization for most codes, i.e. *no* optimization arguments to Intel compilers
- Use the "-qopt-report=3 -qopt-report-phase=vec" compiler option to get vectorization report. You can add "-qopt-report-file=*filename*" to send the output to a file
- Always use -no-ipo if you use -fast

Further readings:

http://www.nersc.gov/users/computational-systems/edison/performance-and-optimization/

http://www.nersc.gov/users/computational-systems/cori/application-porting-and-performance/

# Running Jobs on NERSC Systems

- A **job** is the parallel execution of a program using multiple program tasks and/or threads on multiple processors
- A job executes on one or more **compute nodes** dedicated to that job
- Before a job begins execution, a **resource manager** and **batch scheduler** reserve nodes for the job based on instructions given to the scheduler by the user
- When the nodes are ready, a **job launcher** distributes the executable code to the nodes allocated to the job, then starts and manages execution of the code on multiple nodes in a coordinated fashion
- Edison & Cori use **SLURM** as their batch system/resource manager and job launcher

# SLURM

A Brief Summary of SLURM Commands:

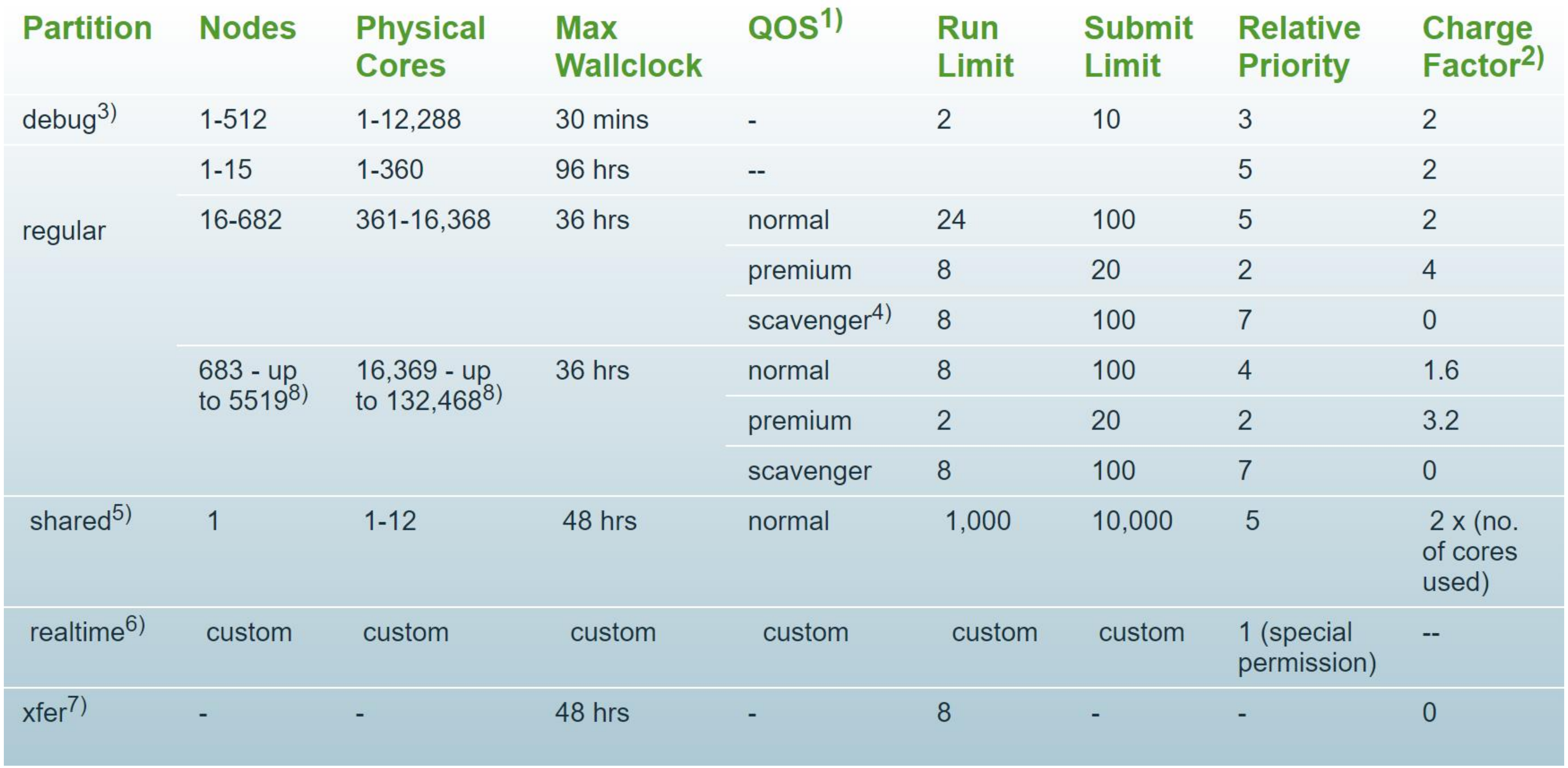
- Batch jobs are submitted with **sbatch**
- Interactive job sessions are requested through **salloc**
- The command to launch a job is **srun**
- Nodes info and cluster status may be requested with **sinfo**
- Job control and monitoring are performed by **scontrol** and **squeue**
- NERSC provides a custom queue monitor **sqs**
- Job and job steps accounting data can be accessed with **sacct**
- Useful environment variables are **$SLURM_NODELIST** and **$SLURM_JOBID**

➢ http://www.nersc.gov/users/computational-systems/cori/running-jobs/slurm-at-nersc-overview/

➢ A quick two page summary of SLURM Commands

# Edison Queues and Scheduling Policies

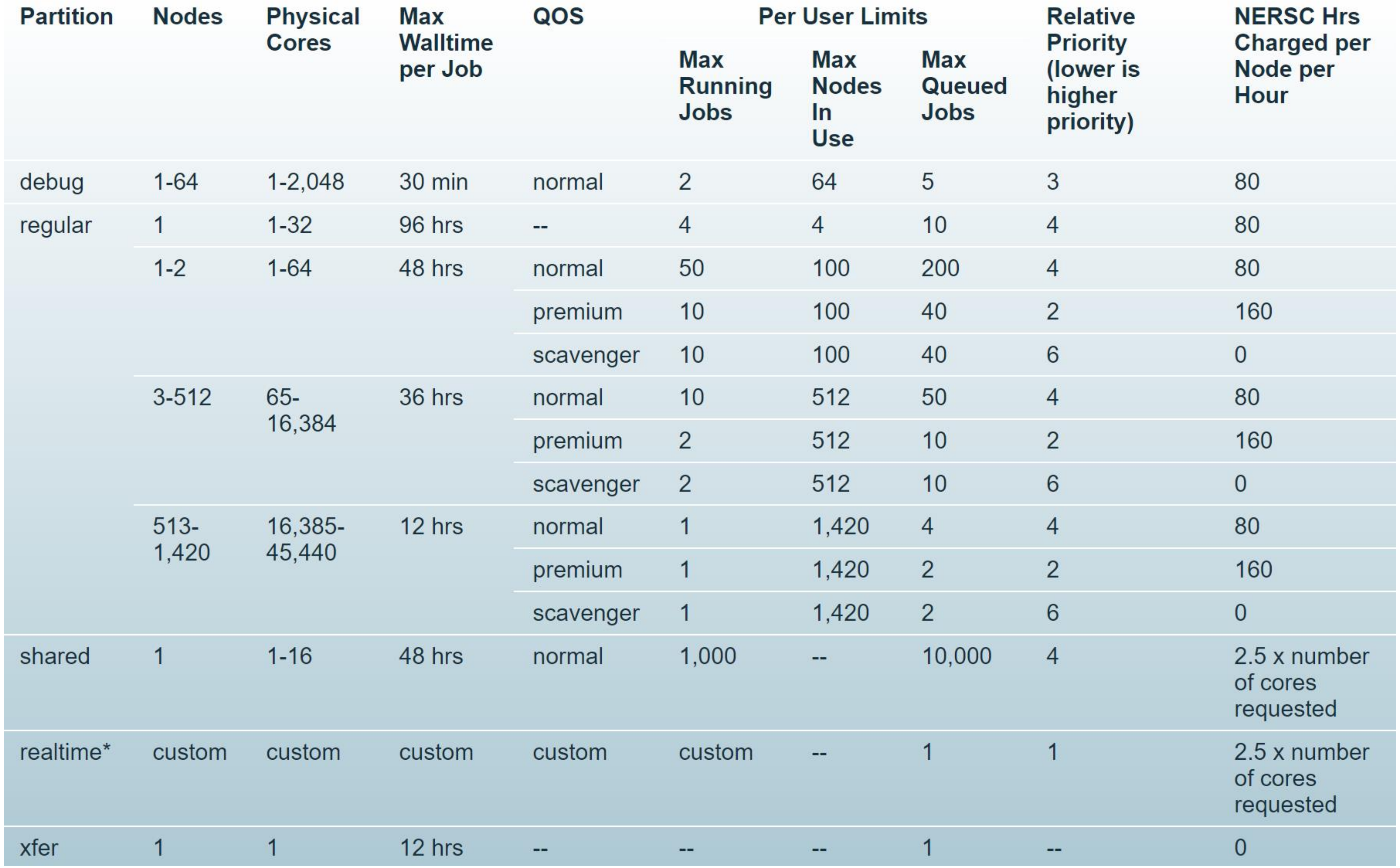
http://www.nersc.gov/users/computational-systems/edison/running-jobs/queues-and-policies/



| Partition | Nodes | Physical Cores | Max Wallclock | QOS[1] | Run Limit | Submit Limit | Relative Priority | Charge Factor[2] |
|---|---|---|---|---|---|---|---|---|
| debug[3] | 1-512 | 1-12,288 | 30 mins | - | 2 | 10 | 3 | 2 |
| regular | 1-15 | 1-360 | 96 hrs | -- | | | 5 | 2 |
| | 16-682 | 361-16,368 | 36 hrs | normal | 24 | 100 | 5 | 2 |
| | | | | premium | 8 | 20 | 2 | 4 |
| | | | | scavenger[4] | 8 | 100 | 7 | 0 |
| | 683 - up to 5519[8] | 16,369 - up to 132,468[8] | 36 hrs | normal | 8 | 100 | 4 | 1.6 |
| | | | | premium | 2 | 20 | 2 | 3.2 |
| | | | | scavenger | 8 | 100 | 7 | 0 |
| shared[5] | 1 | 1-12 | 48 hrs | normal | 1,000 | 10,000 | 5 | 2 x (no. of cores used) |
| realtime[6] | custom | custom | custom | custom | custom | custom | 1 (special permission) | -- |
| xfer[7] | - | - | 48 hrs | - | 8 | - | - | 0 |

# Cori Phase I Queues and Policies

http://www.nersc.gov/users/computational-systems/cori/running-jobs/queues-and-policies/



| Partition | Nodes | Physical Cores | Max Walltime per Job | QOS | Per User Limits | | | Relative Priority (lower is higher priority) | NERSC Hrs Charged per Node per Hour |
|---|---|---|---|---|---|---|---|---|---|
| | | | | | Max Running Jobs | Max Nodes In Use | Max Queued Jobs | | |
| debug | 1-64 | 1-2,048 | 30 min | normal | 2 | 64 | 5 | 3 | 80 |
| regular | 1 | 1-32 | 96 hrs | -- | 4 | 4 | 10 | 4 | 80 |
| | 1-2 | 1-64 | 48 hrs | normal | 50 | 100 | 200 | 4 | 80 |
| | | | | premium | 10 | 100 | 40 | 2 | 160 |
| | | | | scavenger | 10 | 100 | 40 | 6 | 0 |
| | 3-512 | 65-16,384 | 36 hrs | normal | 10 | 512 | 50 | 4 | 80 |
| | | | | premium | 2 | 512 | 10 | 2 | 160 |
| | | | | scavenger | 2 | 512 | 10 | 6 | 0 |
| | 513-1,420 | 16,385-45,440 | 12 hrs | normal | 1 | 1,420 | 4 | 4 | 80 |
| | | | | premium | 1 | 1,420 | 2 | 2 | 160 |
| | | | | scavenger | 1 | 1,420 | 2 | 6 | 0 |
| shared | 1 | 1-16 | 48 hrs | normal | 1,000 | -- | 10,000 | 4 | 2.5 x number of cores requested |
| realtime* | custom | custom | custom | custom | custom | -- | 1 | 1 | 2.5 x number of cores requested |
| xfer | 1 | 1 | 12 hrs | -- | -- | -- | 1 | -- | 0 |

# Interactive Jobs

**Edison**: http://www.nersc.gov/users/computational-systems/edison/running-jobs/interactive-jobs/

**Cori**: http://www.nersc.gov/users/computational-systems/cori/running-jobs/interactive-jobs/

Use the **salloc** command to request an interactive job. For example, on Edison, to request 2 nodes using the *debug* partition, and a *license* to run on the $SCRATCH file system:

```
salloc -N 2 -p debug -L SCRATCH
```

Once the compute nodes are allocated, **salloc** will return, you will land on the head compute node, and will be in your *work directory* where the salloc command was executed.

From the shell prompt, you can start your program on the compute nodes using the **srun** command, the parallel job launcher.

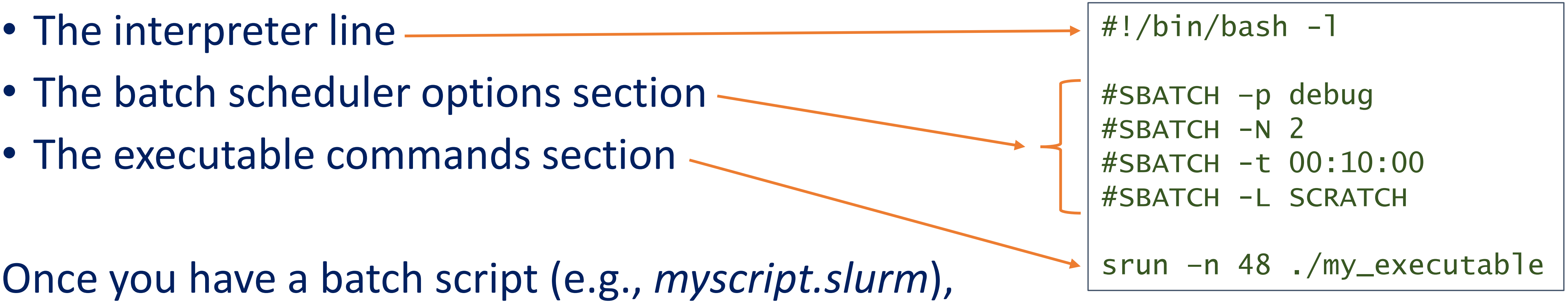
```
srun –n 48 ./my_executable
```

# Batch Jobs

**Edison:** http://www.nersc.gov/users/computational-systems/edison/running-jobs/batch-jobs/

**Cori:** http://www.nersc.gov/users/computational-systems/cori/running-jobs/batch-jobs/

A **batch job** runs non-interactively under the control of a **batch script**. **Batch scripts** are submitted to the batch system, where they are queued awaiting free resources. A batch script is simply a shell script which contains:



- The interpreter line
- The batch scheduler options section
- The executable commands section

```
#!/bin/bash -l

#SBATCH -p debug
#SBATCH -N 2
#SBATCH -t 00:10:00
#SBATCH -L SCRATCH

srun -n 48 ./my_executable
```

Once you have a batch script (e.g., *myscript.slurm*),

you submit it to the batch system using the **sbatch** command.

```
sbatch myscript.slurm
```

# Example Batch Scripts

- Edison:

  http://www.nersc.gov/users/computational-systems/edison/running-jobs/example-batch-scripts/

- Cori:

  http://www.nersc.gov/users/computational-systems/cori/running-jobs/example-batch-scripts/

# Running Jobs on Hyades

- **Torque** + **Maui** are the batch scheduler on Hyades.
- **Torque**, base on *PBS*, is the resource manager.
- Torque utilities
  - ➢**qsub**, submitting job
  - ➢**qstat**, monitoring status of jobs
  - ➢**qdel**, terminating jobs prior to completion
- **Maui** is the job scheduler.
- Maui utilities
  - ➢**showq**, listing both active and idle jobs
  - ➢**showbf**, showing what resources are immediately available (backfill)
  - ➢**checkjob**, viewing details of a job in the queue

# Queues on Hyades

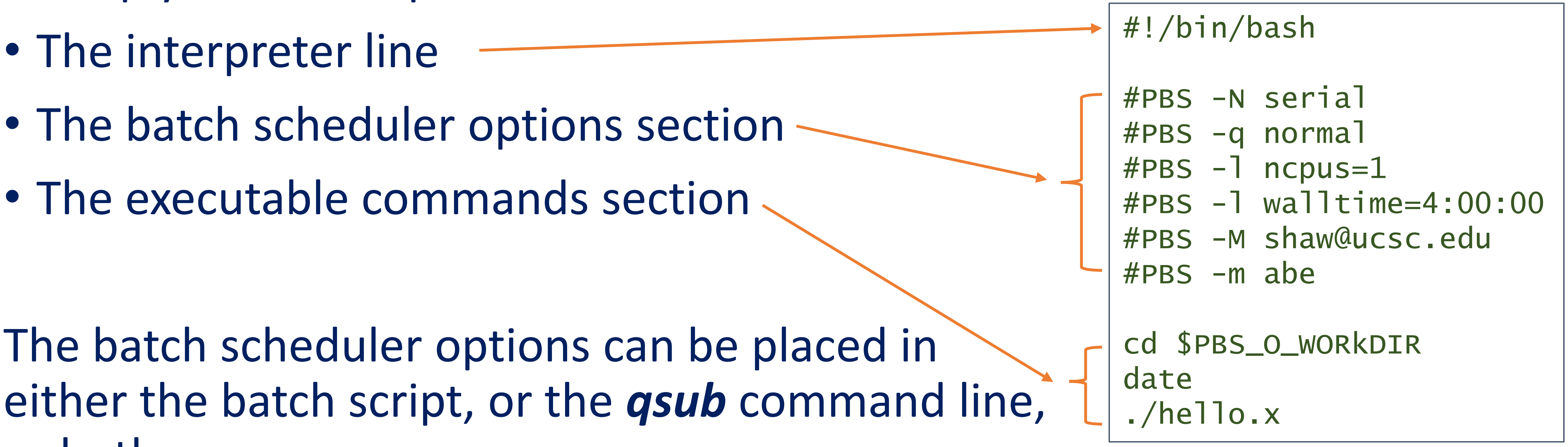
| Queue | Total # of nodes | Resource per node | Max Walltime | qsub options |
|---|---|---|---|---|
| normal | 150 | 16 cores<br>(hyper-threading disabled) | 2 days | -l nodes=*n*:ppn=16 -q normal |
| hyper | 30 | 32 cores<br>(hyper-threading enabled) | 4 days | -l nodes=*n*:ppn=32 -q hyper |
| gpu | 8 | 16 cores and 1 GPU | 10 days | -l nodes=*n*:ppn=16 -q gpu |

Notes:

1. *n* is the number of nodes you request for your job.
2. The default queue is **normal**. Your job will be submitted to the **normal** queue if no queue name is specified.
3. Run `showbf -p all` to find out what resources are immediately available.

# Batch Scripts on Hyades

**Batch scripts**, or job submission scripts, are the mechanism by which a user submits and configures a job for eventual execution. A batch script is simply a shell script which contains:



- The interpreter line
- The batch scheduler options section
- The executable commands section

```
#!/bin/bash

#PBS -N serial
#PBS -q normal
#PBS -l ncpus=1
#PBS -l walltime=4:00:00
#PBS -M shaw@ucsc.edu
#PBS -m abe

cd $PBS_O_WORkDIR
date
./hello.x
```

The batch scheduler options can be placed in either the batch script, or the ***qsub*** command line, or both.

Run “`man qsub`” to learn the options.

# Interactive Batch Job on Hyades

You can start an interactive batch job by calling **qsub** with the "`-I`" option:

```
[dong@hyades dong]$ pwd
/pfs/dong
[dong@hyades dong]$ qsub -q gpu -I -l nodes=2:ppn=16,walltime=1:00:00
qsub: waiting for job 37955.hyades.ucsc.edu to start
qsub: job 37955.hyades.ucsc.edu ready

[dong@gpu-7 ~]$ pwd
/home/dong
[dong@gpu-7 ~]$ printenv | grep PBS
[dong@gpu-7 ~]$ echo $PBS_O_WORKDIR
/pfs/dong
[dong@gpu-7 ~]$ cat $PBS_NODEFILE
```

# Sample Batch Script for Serial Jobs on Hyades

Batch script *serial.pbs:*

```
#!/bin/bash

#PBS -N serial
#PBS -q normal
#PBS -l ncpus=1
#PBS -l walltime=4:00:00
#PBS -M shaw@ucsc.edu
#PBS -m abe

cd $PBS_O_WORkDIR
./hello.x
```

Comments:

```
### your favorite shell

### job name
### job queue
### request only 1 core
### and 4 hours walltime
### ask Torque to send emails
### when jobs aborts, starts and ends

### go to the directory where you submit the job
### run your serial executable
```

To submit the job:

```
qsub serial.pbs
```

Which is equivalent to:

```
Qsub -N serial -q normal -l ncpus=1 \
–l walltime=4:00:00 -M shaw@ucsc.edu \
-m abe serial2.pbs
```

*serial2.pbs:*

```
#!/bin/bash

cd $PBS_O_WORkDIR
./hello.x
```

# Sample script for embarrassingly parallel jobs on Hyades

For example, there is a serial program (*jobarray_hello.x*) that takes an integer argument:

```
./jobarray_hello.x 23
Hello master, I am slave no. 23 running on hyades.ucsc.edu!
```

Let’s assume that we need to run the following instances:

```
./jobarray_hello.x 101
./jobarray_hello.x 102
...
./jobarray_hello.x 164
```

Instead of submitting 64 serial jobs, we can submit only one job array:

```
qsub jobarray.pbs
```

Batch script *jobarray.pbs:*

```
#!/bin/bash

#PBS -N jobarray
#PBS -q normal
#PBS -l ncpus=1
#PBS -l walltime=4:00:00
#PBS -t 101-164
#PBS -M shaw@ucsc.edu
#PBS -m abe

cd $PBS_O_WORKDIR
./jobarray_hello.x $PBS_ARRAYID
```

# Sample Batch Script for OpenMP Jobs on Hyades

Batch script *omp.pbs:*

```
#!/bin/bash

#PBS -N omp
#PBS -q normal
#PBS -l nodes=1:ppn=16
#PBS -l walltime=4:00:00
#PBS -M shaw@ucsc.edu
#PBS -m abe

export OMP_NUM_THREADS=16
cd $PBS_O_WORkDIR
./omp_hello.x
```

Comments:

```
### your favorite shell

### job name
### job queue
### request 1 node (16 cores)
### and 4 hours walltime
### ask Torque to send emails
### when jobs aborts, starts and ends

### set the maximum no. of OpenMP threads to 16
### go to the directory where you submit the job
### run your OpenMP executable
```

To submit the job:

```
qsub omp.pbs
```

# Sample Batch Script for MPI Jobs on Hyades

Batch script *impi.pbs:*

```
#!/bin/bash

#PBS -N impi
#PBS -q normal
#PBS -l nodes=4:ppn=16
#PBS -l walltime=4:00:00
#PBS -M shaw@ucsc.edu
#PBS -m abe
#PBS -j oe

cd $PBS_O_WORkDIR
mpirun -genv I_MPI_FABRICS shm:ofa -n 64 ./mpi_hello.x
```

Comments:

```
### your favorite shell

### job name
### job queue
### request 4 node (16 cores per node)
### and 4 hours walltime
### ask Torque to send emails
### when jobs aborts, starts and ends
### merge standard error with standard output

### go to the directory where you submit the job
### run your MPI executable
```

To submit the job:

```
qsub impi.pbs
```

# Data & Analytics

http://www.nersc.gov/users/data-analytics/

NERSC offers a variety of services to support data-centric workloads:

- Data Management and I/O optimization
- Data Analytics, including:
  - Machine Learning and Deep Learning
  - Spark
- Data Visualization, including:
  - ParaView
  - VisIt
  - NCAR Graphics
- Data Transfer

# Further Readings

1. Intel 64 and IA-32 Architectures Software Developer Manuals: http://www.intel.com/content/www/us/en/processors/architectures-software-developer-manuals.html
2. Introduction to InfiniBand for End Users: http://www.mellanox.com/pdf/whitepapers/Intro_to_IB_for_End_Users.pdf
3. RDMA Aware Networks Programming User Manual: http://www.mellanox.com/related-docs/prod_software/RDMA_Aware_Programming_user_manual.pdf
4. Quick Reference Guide to Optimization with Intel C++ and Fortran Compilers: https://software.intel.com/sites/default/files/managed/12/f1/Quick-Reference-Card-Intel-Compilers-v16.pdf