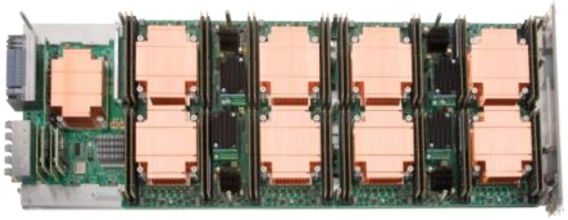
# Cray XC30 Architecture Overview



# Cray's recipe for a good supercomputer

- **Select best microprocessor**
  - Function of time
- **Surround it with a bandwidth-rich environment**
  - Interconnection network
  - Local memory
- **Scale the system**
  - Eliminate operating system interference (OS jitter)
  - Design in reliability and resiliency
  - Provide scalable system management
  - Provide scalable I/O
  - Provide scalable programming and performance tools
  - System service life

# Nodes: The building blocks

**The Cray XC30 is a Massively Parallel Processor (MPP) supercomputer design. It is therefore built from many thousands of individual nodes.**

**There are two basic types of nodes in any Cray XC30:**

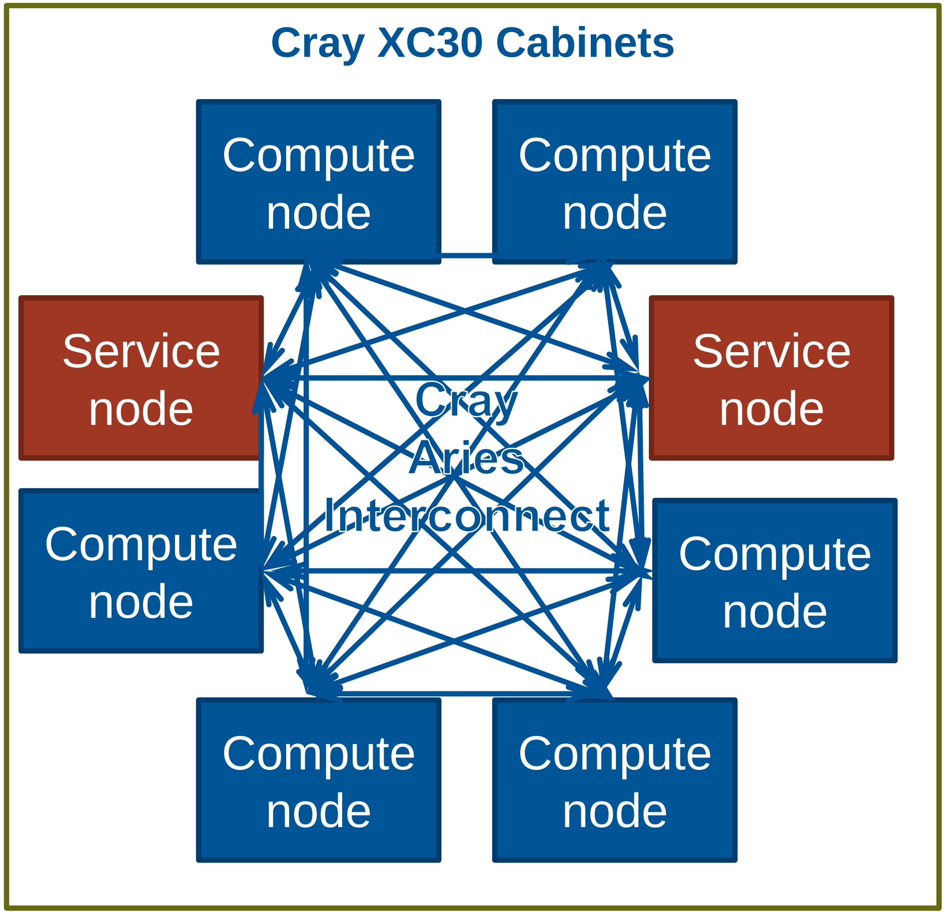
- **Compute nodes**
  - These only do user computation and are always referred to as "Compute nodes"
- **Service nodes**
  - These provide all the additional services required for the system to function, and are given additional names depending on their individual task:
    - Login nodes – allow users to log in and perform interactive tasks
    - PBS Mom nodes – run and managing PBS batch scripts
    - Service Database node (SDB) – holds system configuration information
    - LNET Routers - connect to the external filesystem.

**There are usually many more compute than service nodes**

# Connecting nodes together: Aries

**Obviously, to function as a single supercomputer, the individual nodes must have method to communicate with each other.**

**All nodes in the interconnected by the high speed, low latency Cray Aries Network.**



# Differences between Nodes

## Service Nodes

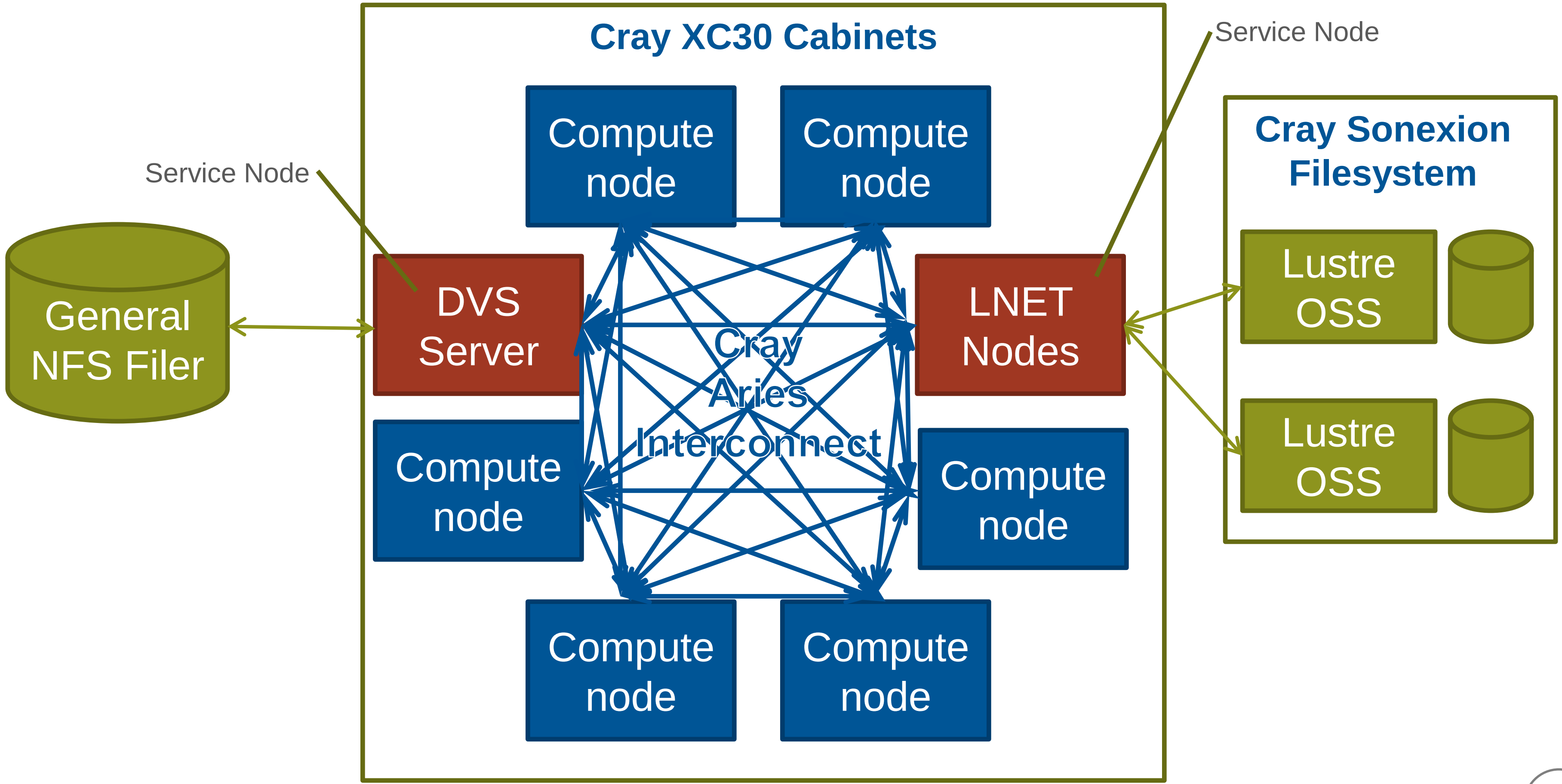
- This is the node you access when you first log in to the system.
- They run a full version of the CLE operating system (all libraries and tools available)
- They are used for editing files, compiling code, submitting jobs to the batch queue and other interactive tasks.
- They are shared resources that may be used concurrently by multiple users.
- There may be many service nodes in any Cray XC30 and can be used for various system services (login nodes, IO routers, daemon servers).

## Compute nodes

- These are the nodes on which production jobs are executed
- They run Compute Node Linux, a version of the OS optimised for running batch workloads
- They can only be accessed by submitting jobs through a batch management system (e.g. PBS Pro, Moab, SLURM)
- They are exclusive resources that may only be used by a single user.
- There are many more compute nodes in any Cray XC30 than login or service nodes.

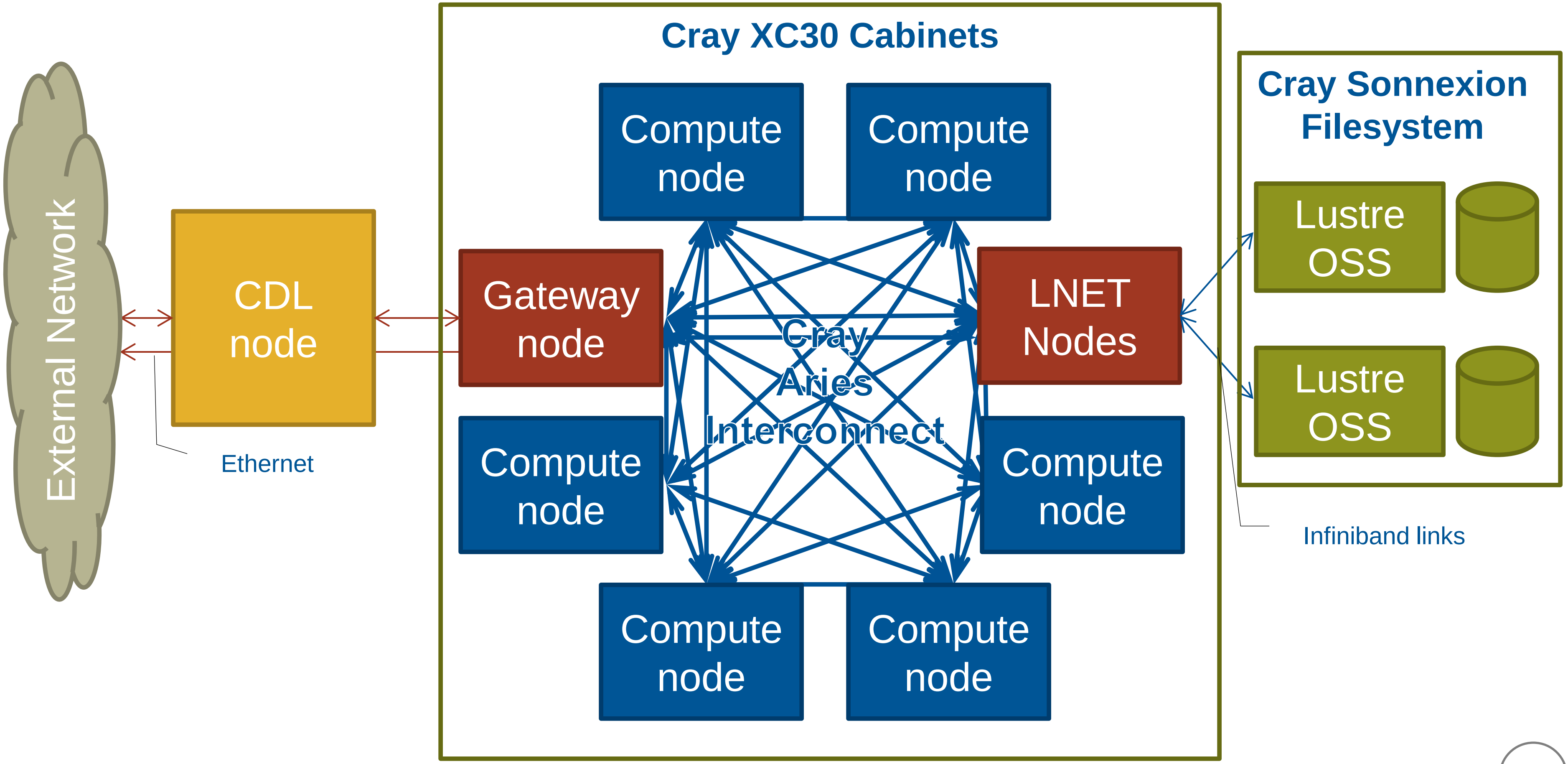
# Adding Storage

Neither compute nor service nodes have storage of their own. It must be connected via the service node's native Lustre Client or projected using the Cray Data Virtualization Service (DVS)

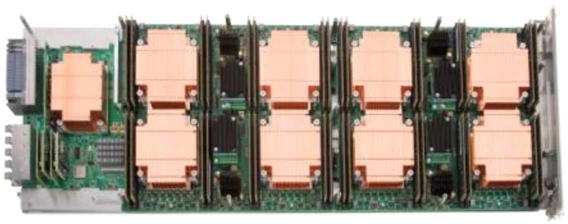


# Interacting with the system

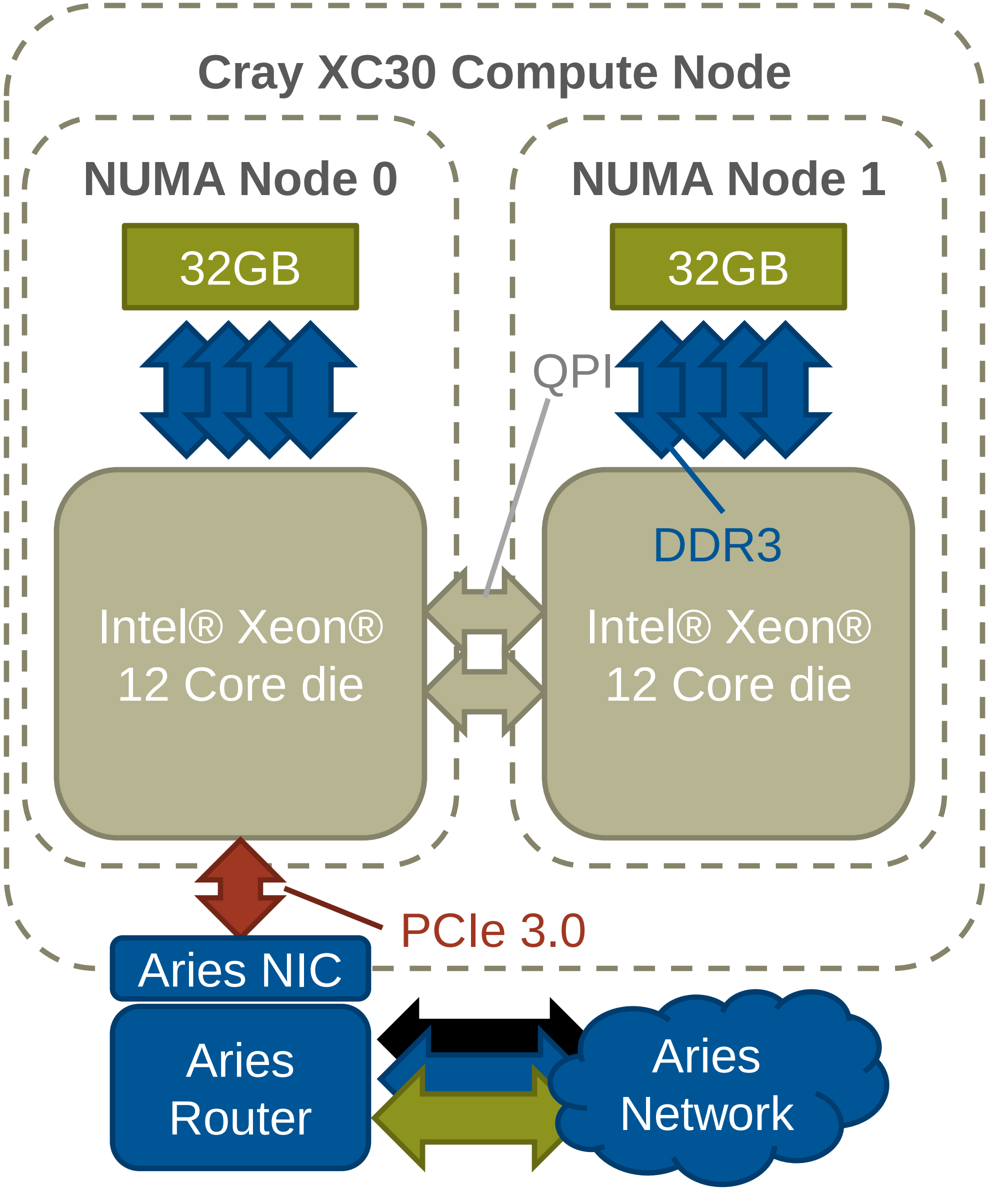
**Users do not log directly into the system. Instead they run commands via an Cray Development Login servers. This server will relay commands and information via a service node referred to as a “Gateway node”**



# Cray XC30 Compute Node Architecture



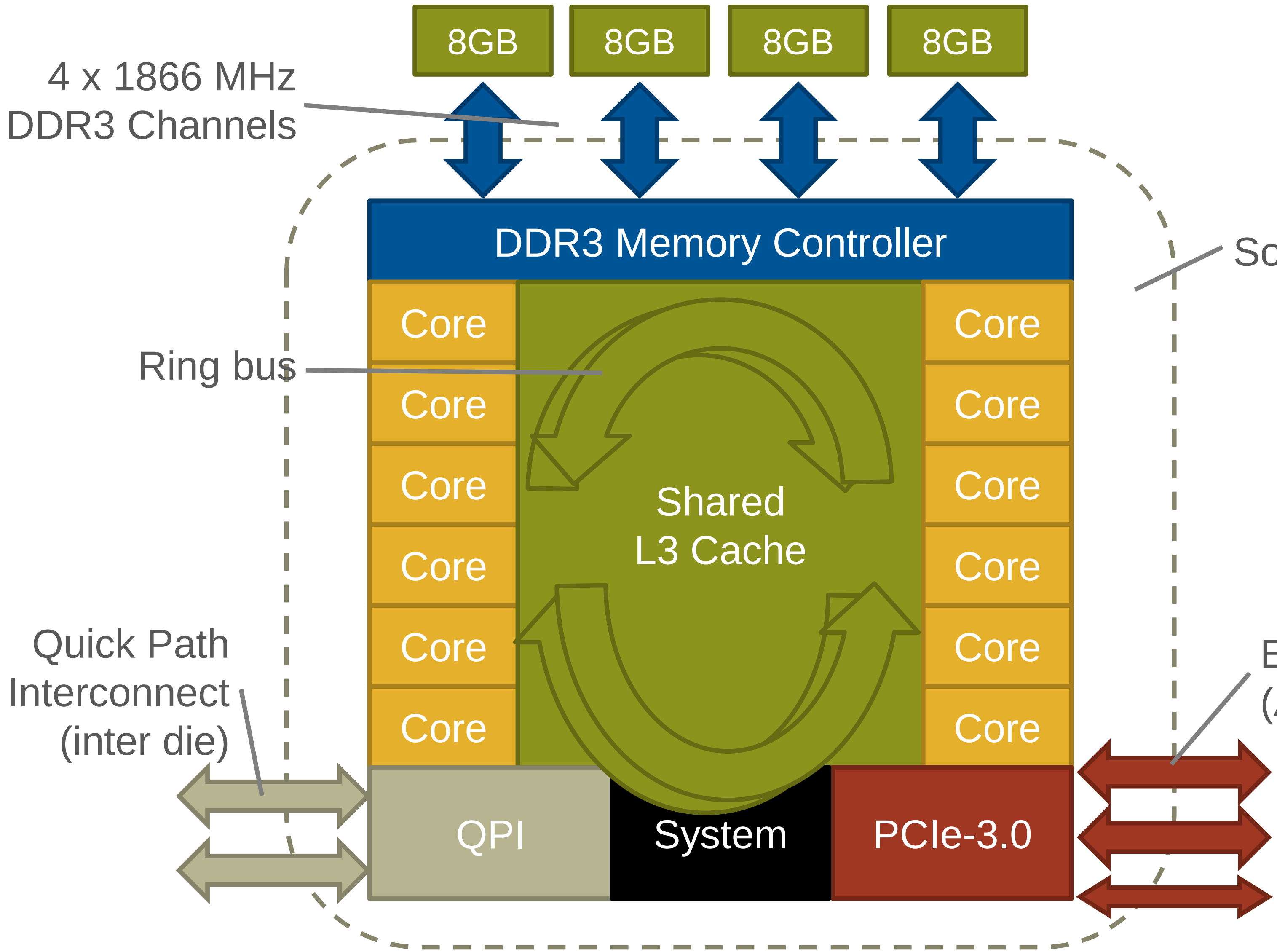
# Cray XC30 Intel® Xeon® Compute Node



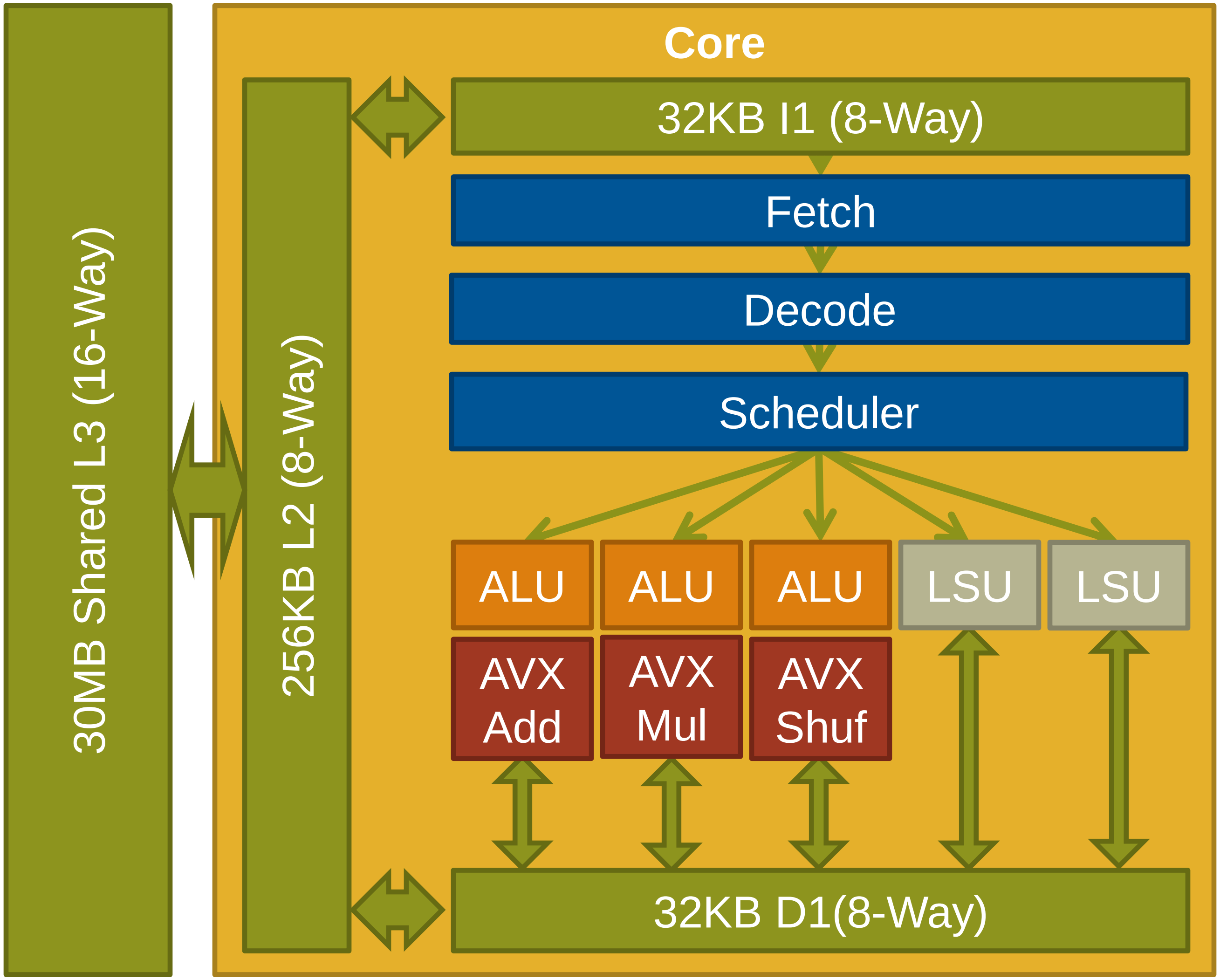

**The XC30 Compute node features:**

- **2 x Intel® Xeon® Sockets/die**
  - 12 core Ivybridge
  - QPI interconnect
  - Forms 2 NUMA nodes
- **8 x 1833MHz DDR3**
  - 8 GB per Channel
  - 64 GB total
- **1 x Aries NIC**
  - Connects to shared Aries router and wider network
  - PCI-e 3.0

# Intel® Xeon® Ivybridge 12-core socket/die



# Intel Xeon Ivybridge Core Structure

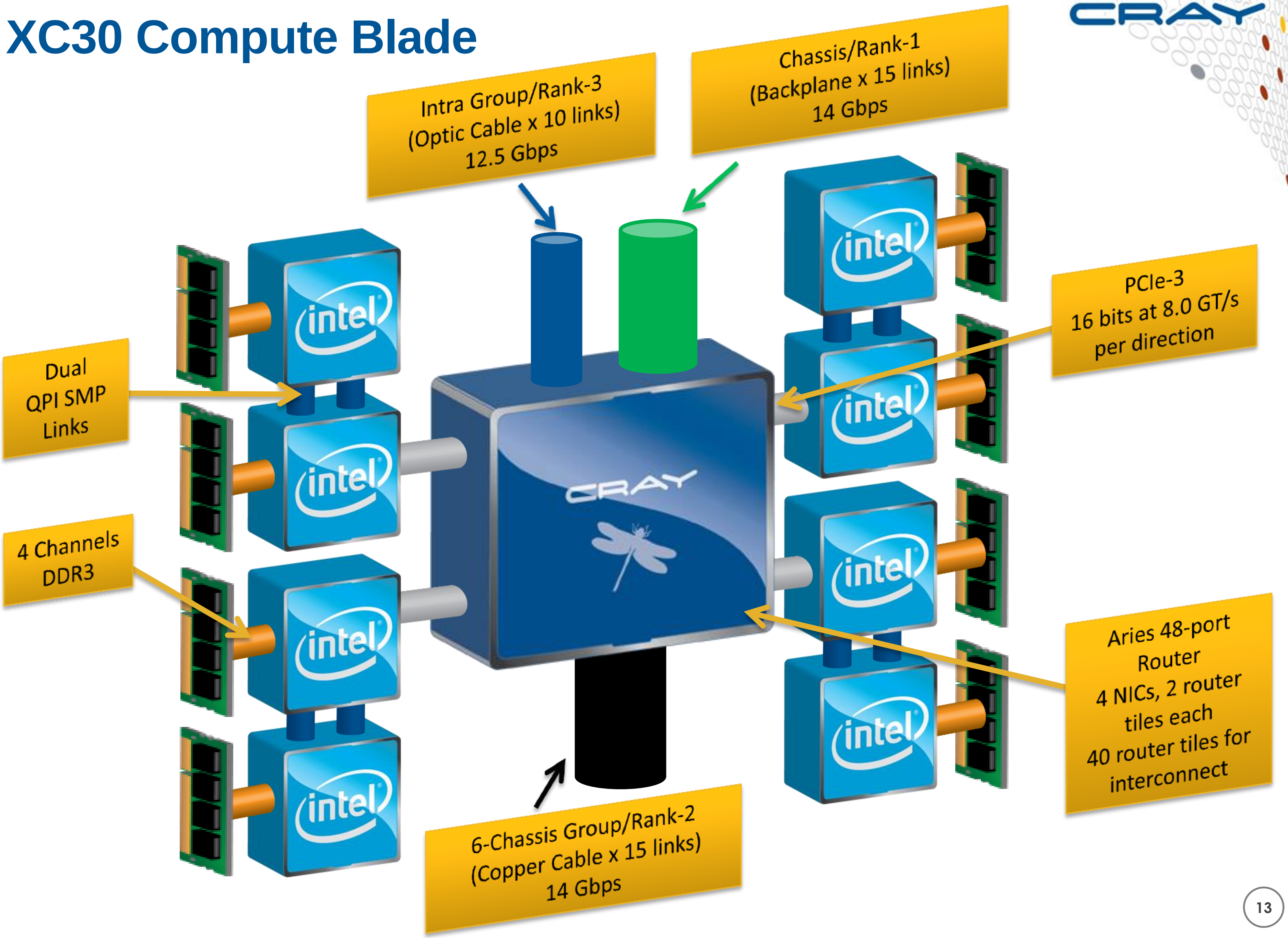



- Manufactured on a 22nm Process
- 256 bit AVX Instructions (4 double precision floating point)
  - 1 x Add
  - 1 x Multiply
  - 1 x Other
- 2 Hardware threads (Hyperthreads)
- Peak DP FP per node 8FLOPS/clock

# Interlagos/Ivybridge Comparison

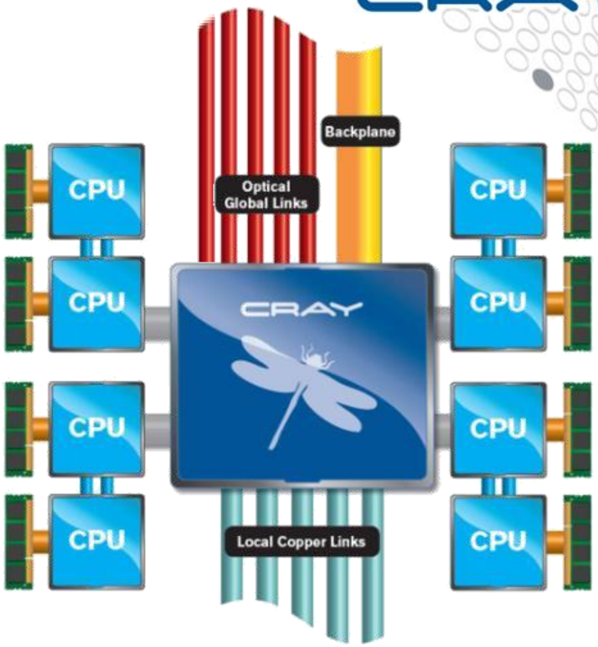
| | AMD Opteron “Interlagos” | Intel Xeon “Ivybridge” |
|---|---|---|
| Base Clock Speed | 2.3 GHz | 2.7 GHz |
| Cores per die | 6 | 12 |
| Dies per node | 4 | 2 |
| *Each cores has:* | | |
| User threads | 1 | 2 |
| Function group | 1 SSE (vector) | 1 AVX (vector) |
| bits wide | 128 bits wide | 256 bits wide |
| functional units | 1 add and 1 multiply | 1 add and 1 multiply |
| Cache: L1 | 32KB | 32KB |
| Cache: L2 | 512KB | 256KB |
| L3 Cache (per die) | 6 MB | 30 MB |
| Total Cache per core | 1.5 MB | 2.75 MB |
| *Cache BW Per core (GB/s)* | | |
| L1/L2/L3 | 35 / 3.2 / 3.2 | 100 / 40 / 23 |
| Stream TRIAD BW/node | 52 Gbytes/s | 100 Gbytes/s |
| Peak DP FLOPs per core | 4 flops/clk | 8 flops/clk |
| Peak DP FLOPs per node | 294 GFlops | 518 GFlops |
| Main memory latency | 110ns | 82ns |

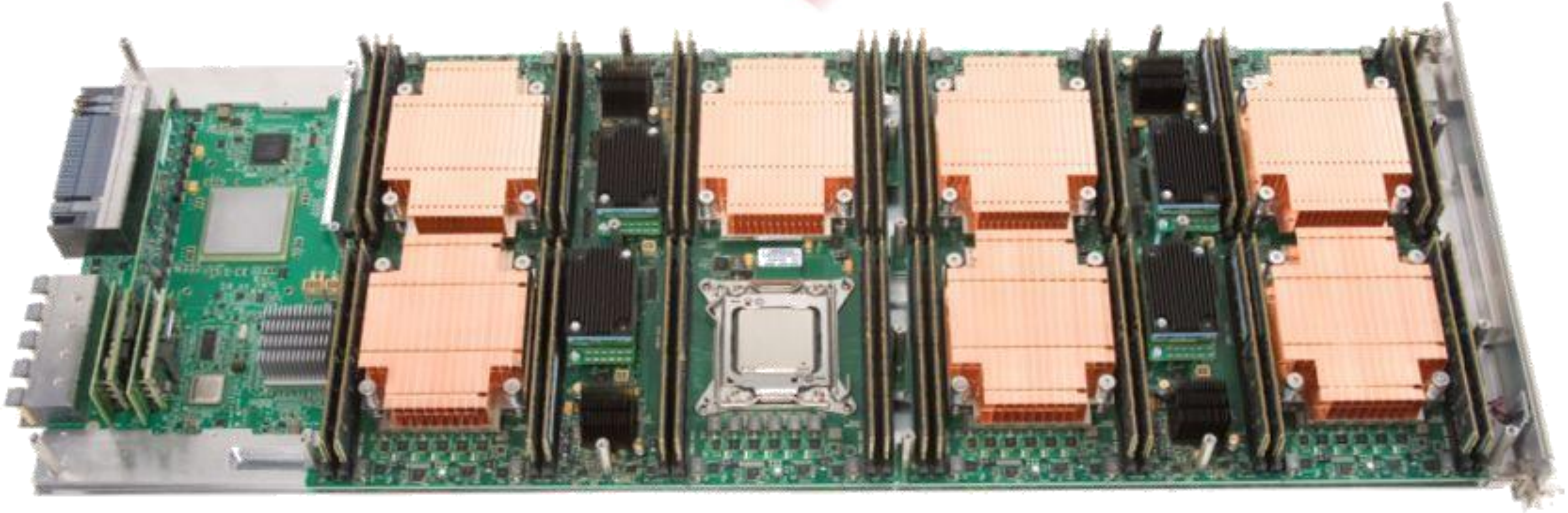
# XC30 Compute Blade



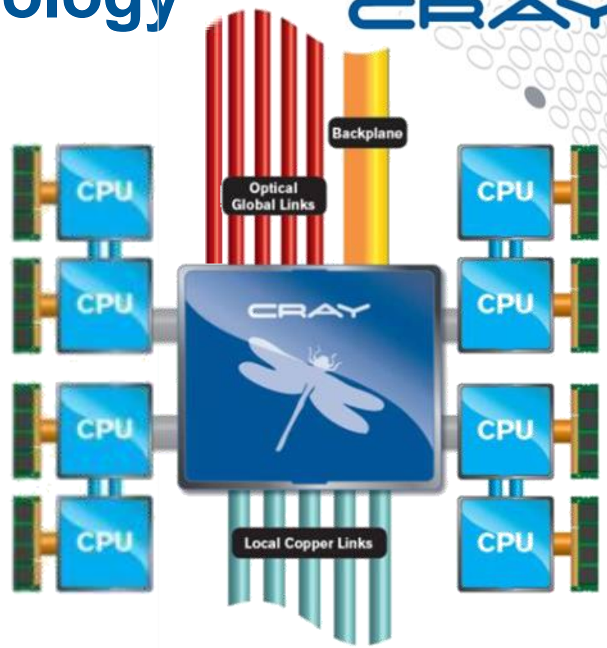
# Cray XC30 Fully Populated Compute Blade

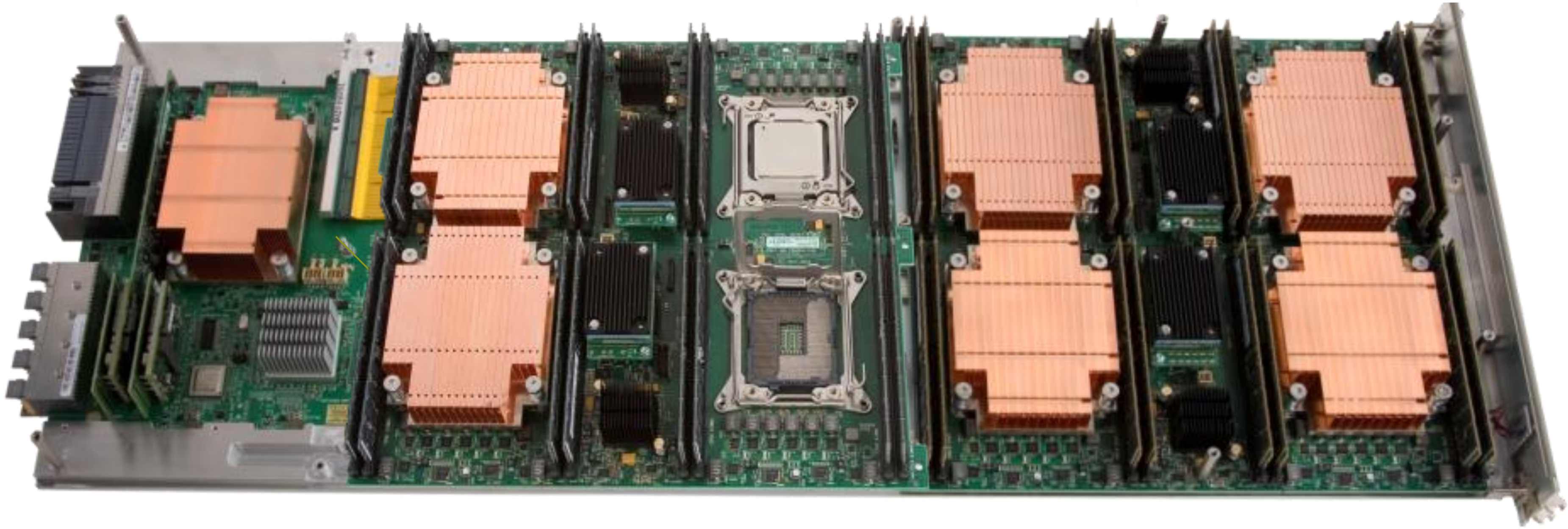
| SPECIFICATIONS | |
|---|---|
| Module power: | 2014 Watts |
| PDC max. power: | 900 Watt |
| Air flow req.: | 275 cfm |
| Size: | 2.125 in x 12.95 in x 33.5 in |
| Weight: | <40 lbm |



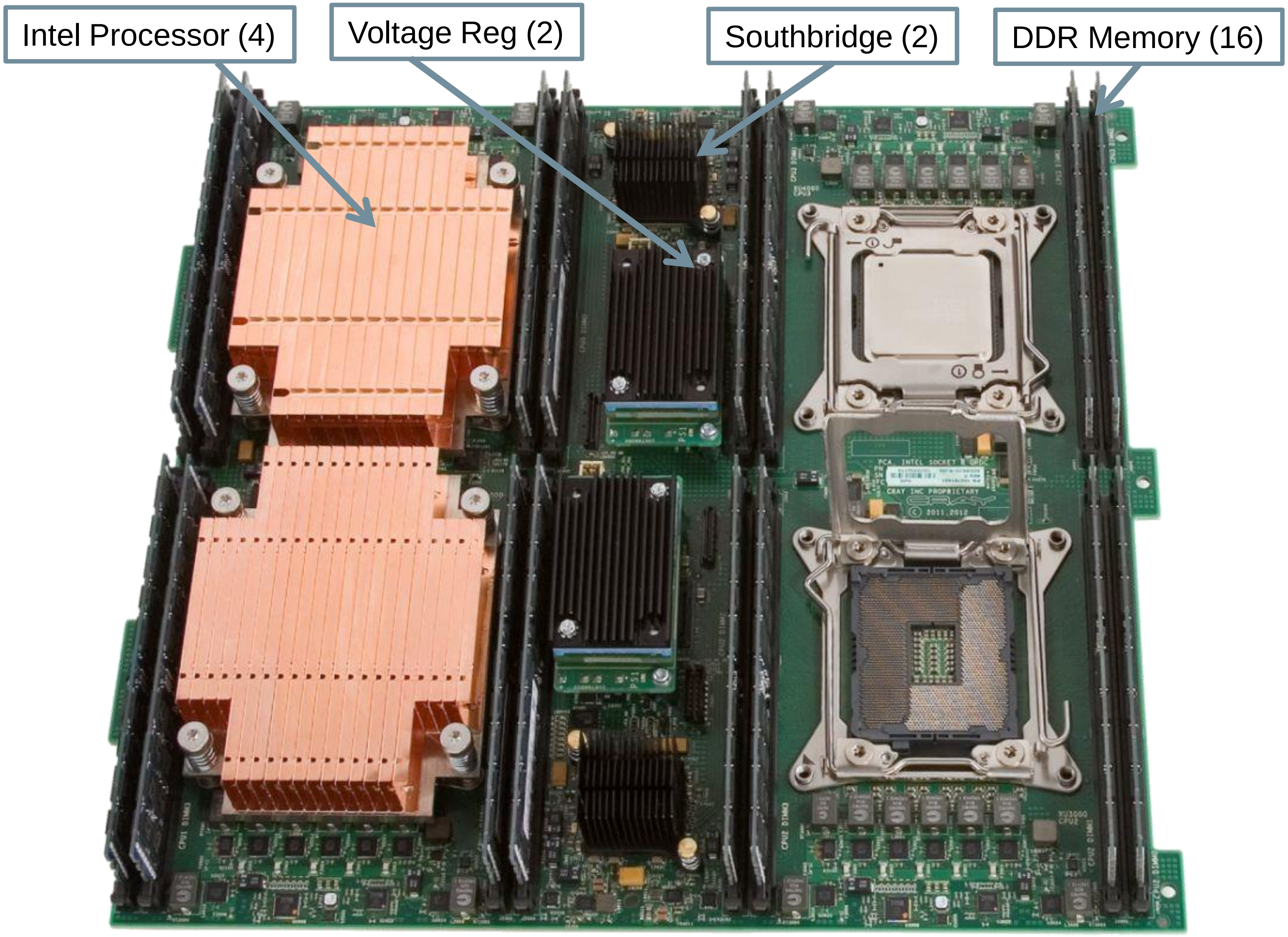



# PDC's are Upgradeable to New Technology



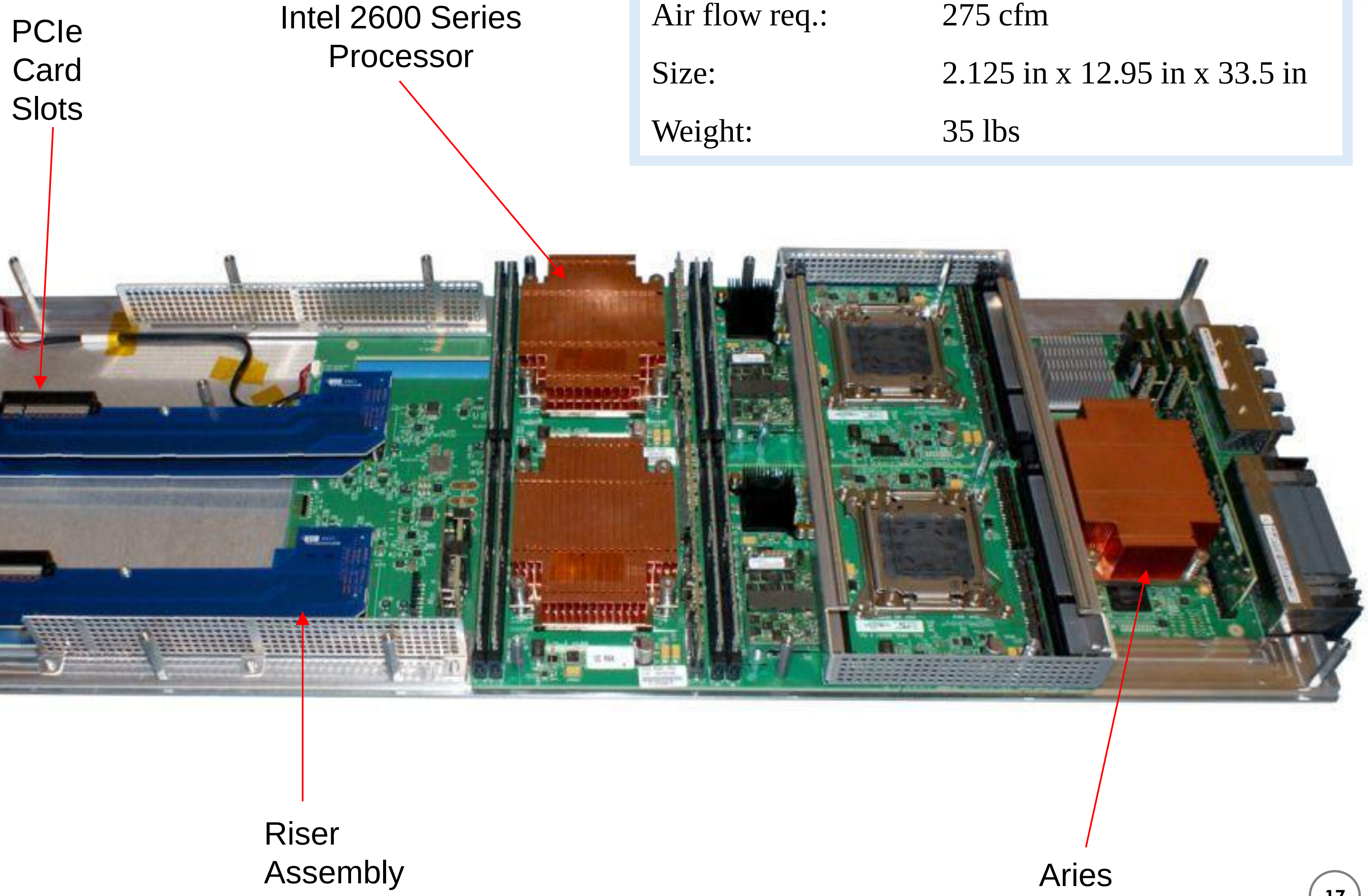
# Cray XC30 Quad Processor Daughter Card



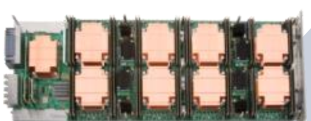
# Cray XC Service Node

**SPECIFICATIONS**

| | |
|---|---|
| Module power: | 1650 Watts |
| PDC max. power: | 225 Watt |
| Air flow req.: | 275 cfm |
| Size: | 2.125 in x 12.95 in x 33.5 in |
| Weight: | 35 lbs |



# Cray XC30 System Building Blocks



Compute Blade

4 Compute Nodes

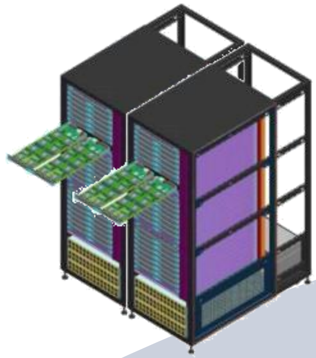
Chassis

Rank 1 Network

16 Compute Blades

No Cables

64 Compute Nodes



Group

Rank 2 Network

Passive Electrical Network

2 Cabinets

6 Chassis

384 Compute Nodes


System

Rank 3 Network

Active Optical Network

Hundreds of Cabinets

Up to 10s of thousands of nodes

# ARCHER's Nodes

**ARCHER hardware on site today has the following:**

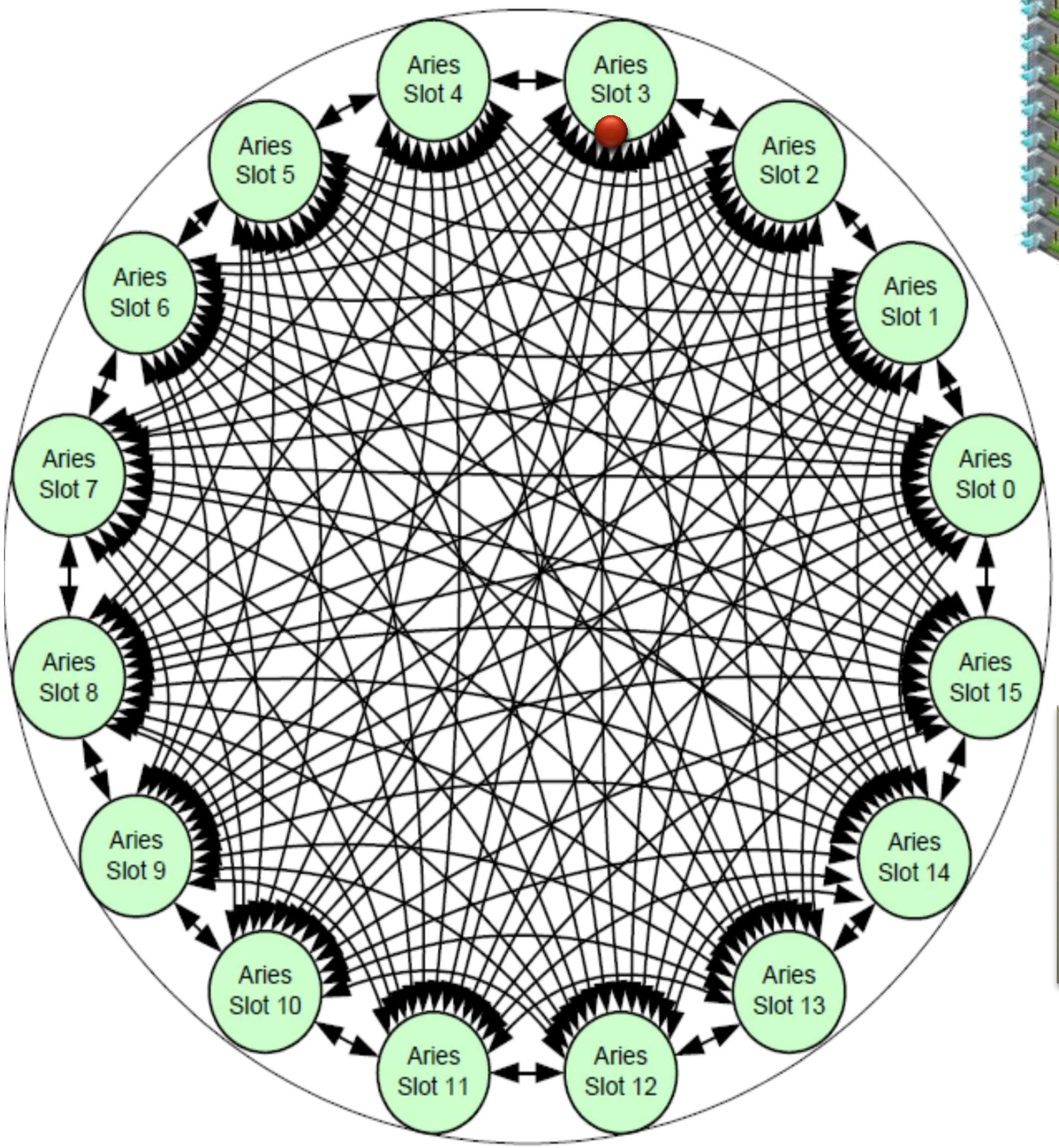
- **16 Cabinets = 8 Groups**
- **3008 Compute Nodes**
  - Dual socket 12 core Intel® Xeon Ivybridge @2.7GHz
    - 2632 x 64 GB 1866MHz Memory
    - 376 x128GB 1866MHz Memory (1 group)
- **32 Service Nodes**
- **8 Cray Development Logins**
  - 256 GB Memory available
- **2 Pre/Post Processing Servers**
  - 1TB Memory per server
- **20 Sonexion SSUs**
  - 160 Lustre Object Storage Targets (distributed over multiple filesystems)
  - 4.34 PB of storage (distributed over multiple filesystems)

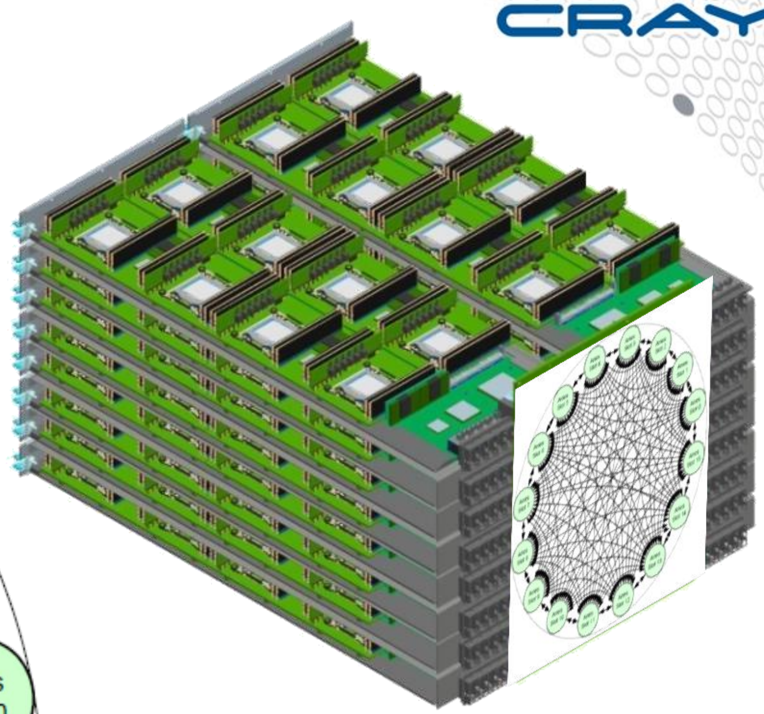
# Cray XC30 Dragonfly Topology + Aries

# Cray Aries Features

- **Scalability to > 500,000 X86 Cores**
  - Cray users run large jobs – 20-50% of system size is common
  - Many examples of 50K-250K MPI tasks per job
  - Optimized collectives MPI_Allreduce in particular
- **Optimized short transfer mechanism (FMA)**
  - Provides global access to memory, used by MPI and PGAS
  - High issue rate for small transfers: 8-64 byte put/get and amo in particular
- **HPC optimized network**
  - Small packet size 64-bytes
  - Router bandwidth >> injection bandwidth
  - Adaptive Routing & Dragonfly topology
- **Connectionless design**
  - Doesn’t depend on a connection cache for performance
  - Limits the memory required per node
- **Fault tolerant design**
  - Link level retry on error
  - Adaptive routing around failed links
  - Network reconfigures automatically (and quickly) if a component fails
  - End to end CRC check with automatic software retry in MPI

# Cray XC30 Rank1 Network

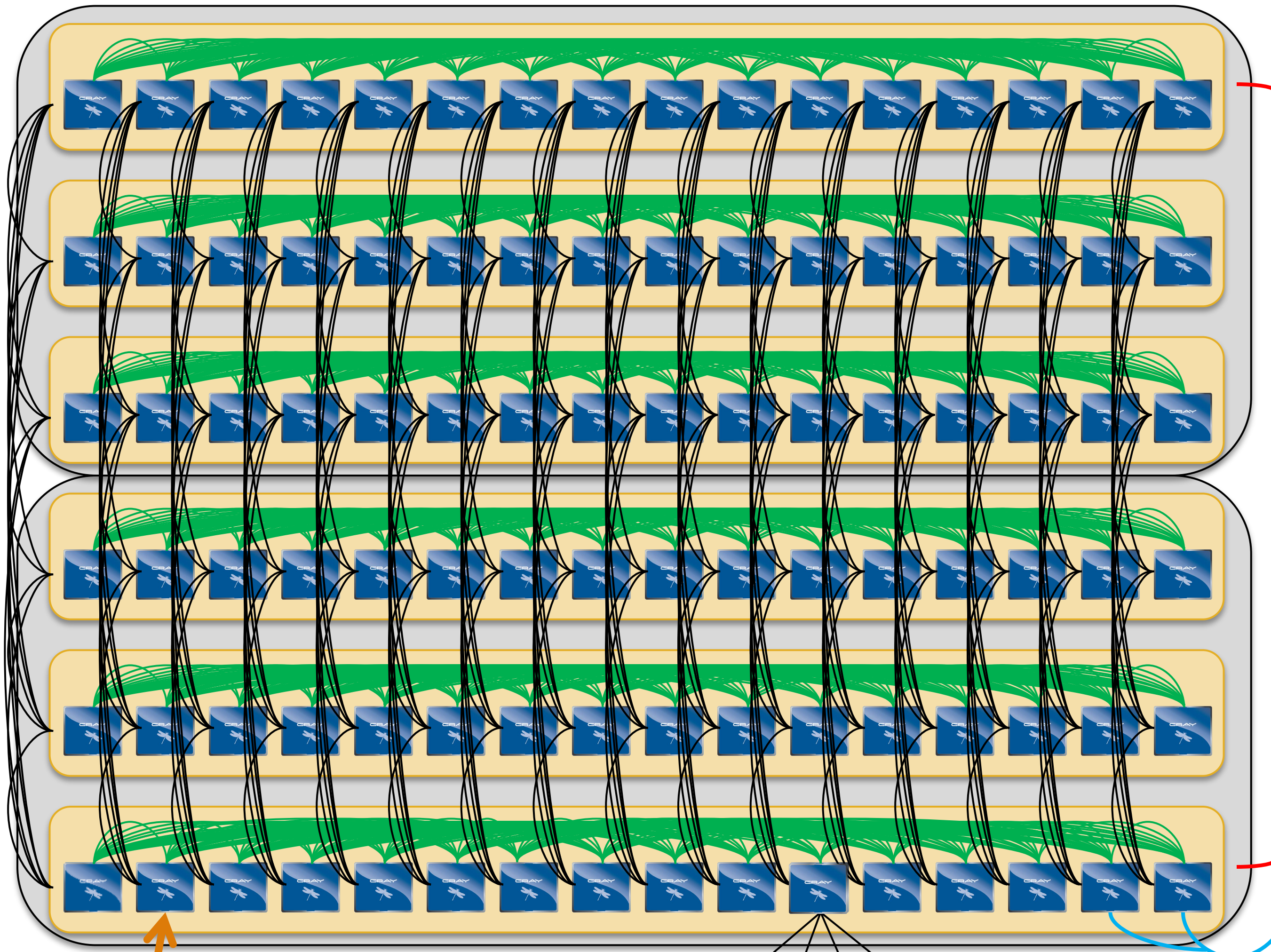


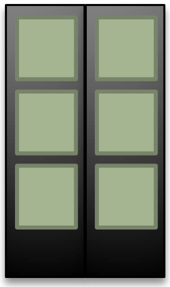



- Chassis with 16 compute blades
- 128 Sockets
- Inter-Aries communication over backplane
- Per-Packet adaptive Routing

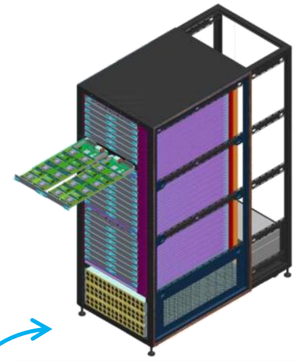
# Cray XC30 Rank-2 Copper Network

**2 Cabinet Group**
**768 Sockets**



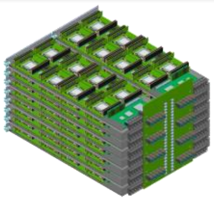


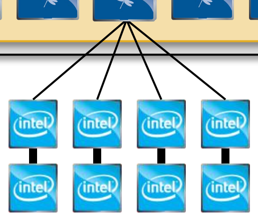
**6 backplanes connected with copper cables in a 2-cabinet group: “Black Network”**



**Active optical cables interconnect groups “Blue Network”**

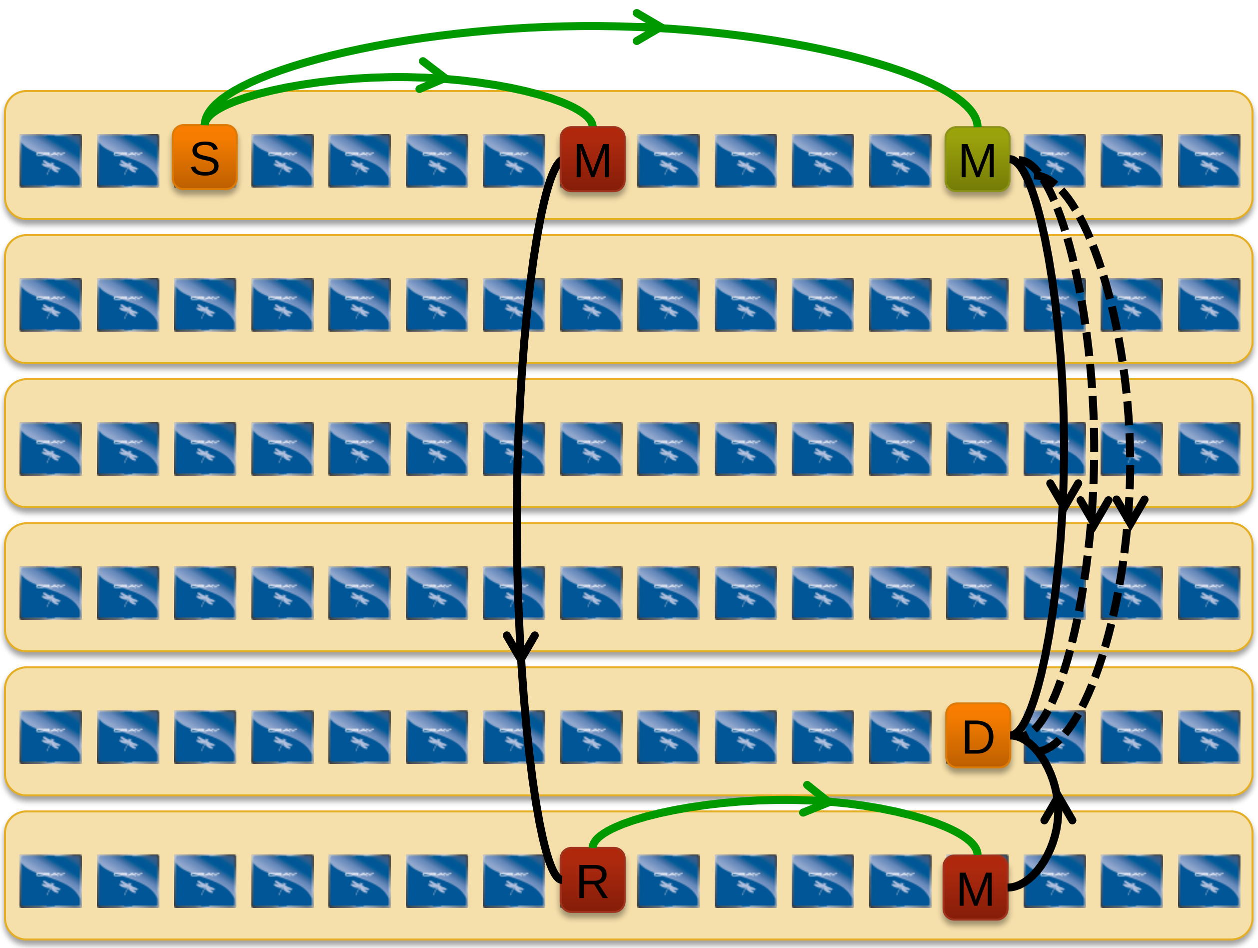
**16 Aries connected by backplane “Green Network”**





**4 nodes connect to a single Aries**

# Cray XC30 Routing




Minimal routes between any two nodes in a group are just two hops

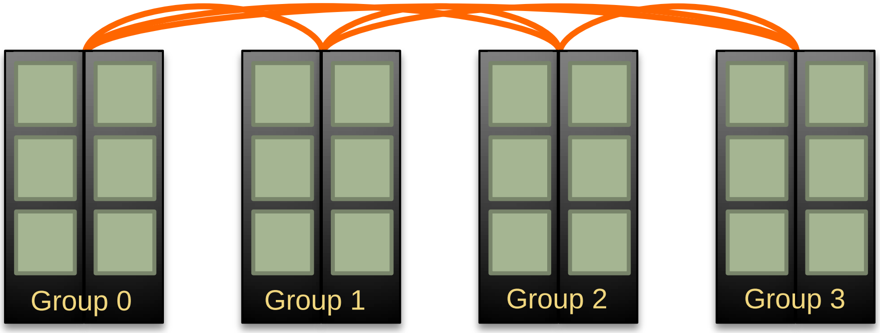
Non-minimal route requires up to four hops.

*With adaptive routing we select between minimal and non-minimal paths based on load*

*The Cray XC30 Class-2 Group has sufficient bandwidth to support full injection rate for all 384 nodes with non-minimal routing*

# Cray XC30 Network Overview – Rank-3 Network

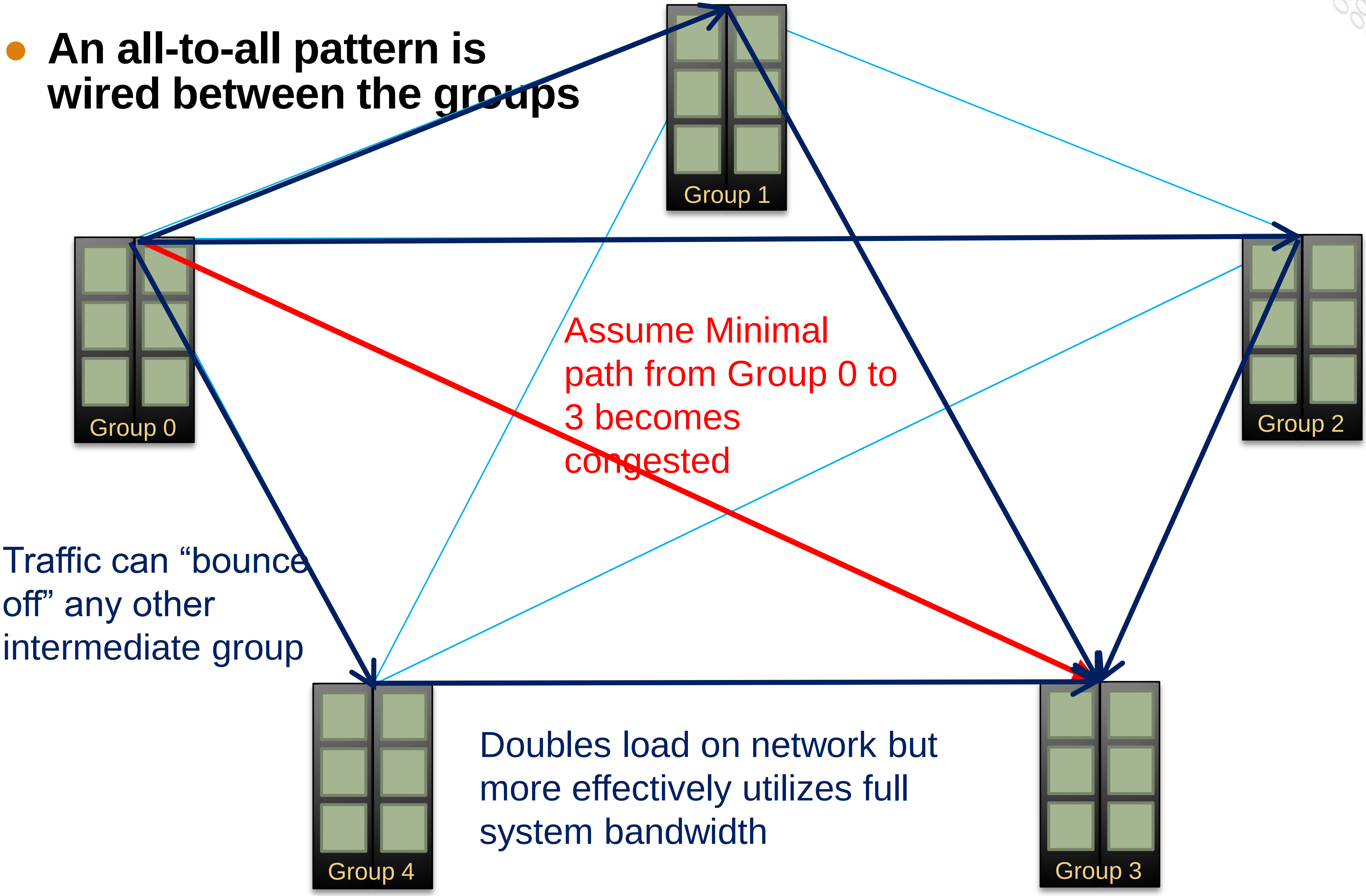
- An all-to-all pattern is wired between the groups using optical cables (blue network)
- Up to 240 ports are available per 2-cabinet group
- The global bandwidth can be tuned by varying the number of optical cables in the group-to-group connections




*Example: An 4-group system is interconnected with 6 optical "bundles". The "bundles" can be configured between 20 and 80 cables wide*

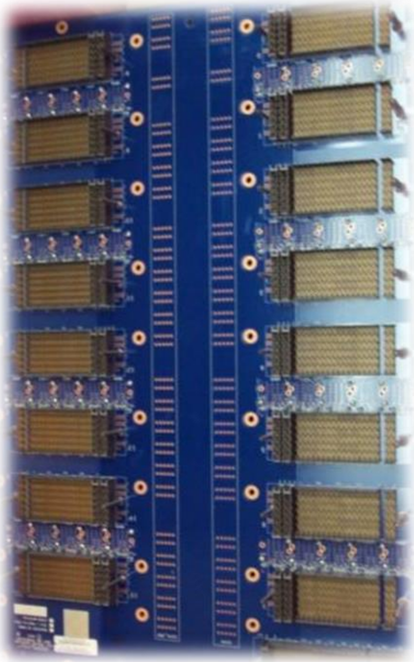
# Adaptive Routing over the Blue Network

- **An all-to-all pattern is wired between the groups**



# Cray XC30 Network

- The Cray XC30 system is built around the idea of optimizing interconnect bandwidth and associated cost at every level



**Rank-1**
**PC Board**

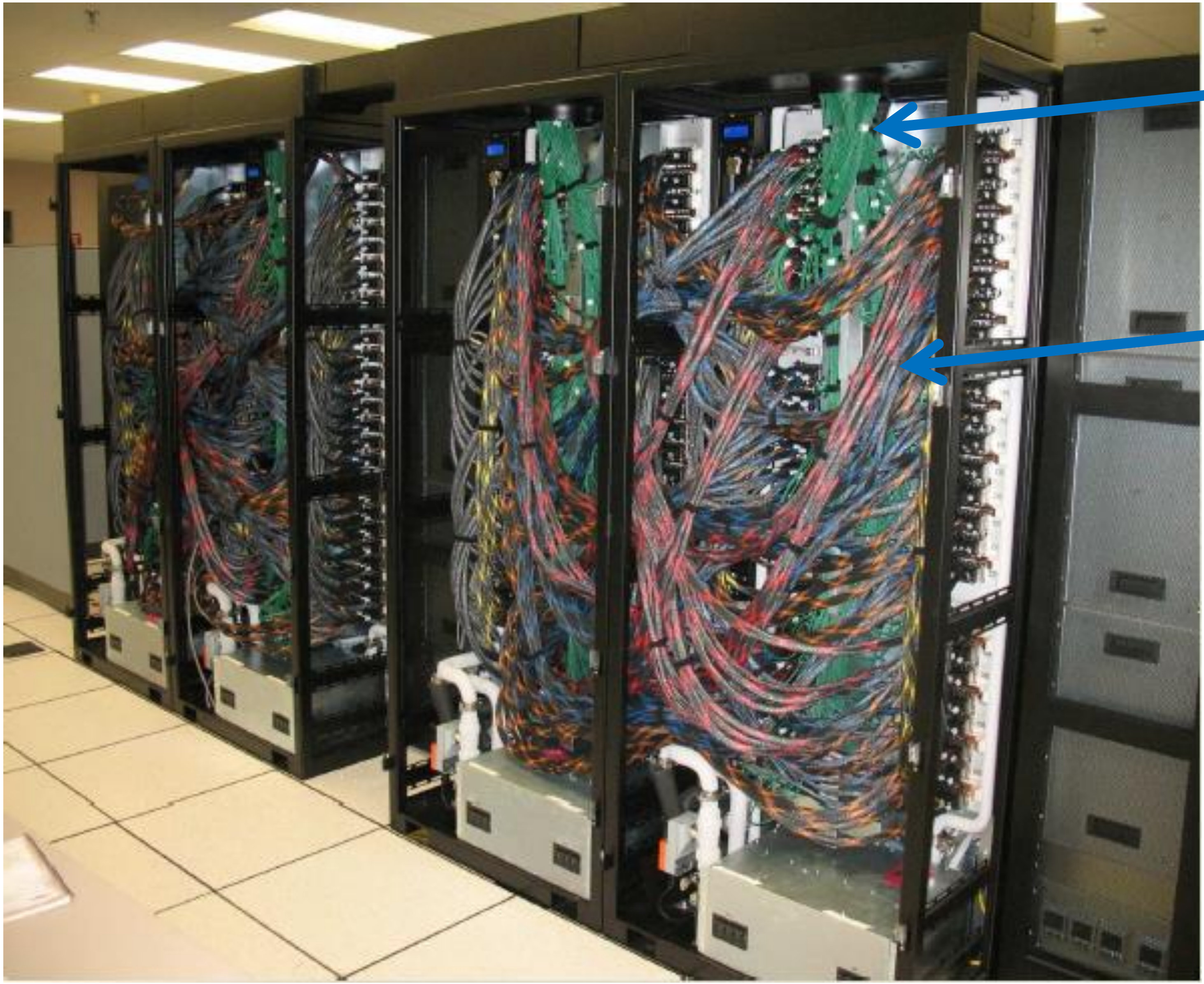
**Rank-2**
**Passive CU**

**Rank-3**
**Active Optics**

# Cray XC30 Rank-2 Cabling

- Cray XC30 two-cabinet group
  - 768 Sockets
  - 96 Aries Chips
- All copper and backplane signals running at 14 Gbps

# Copper & Optical Cabling



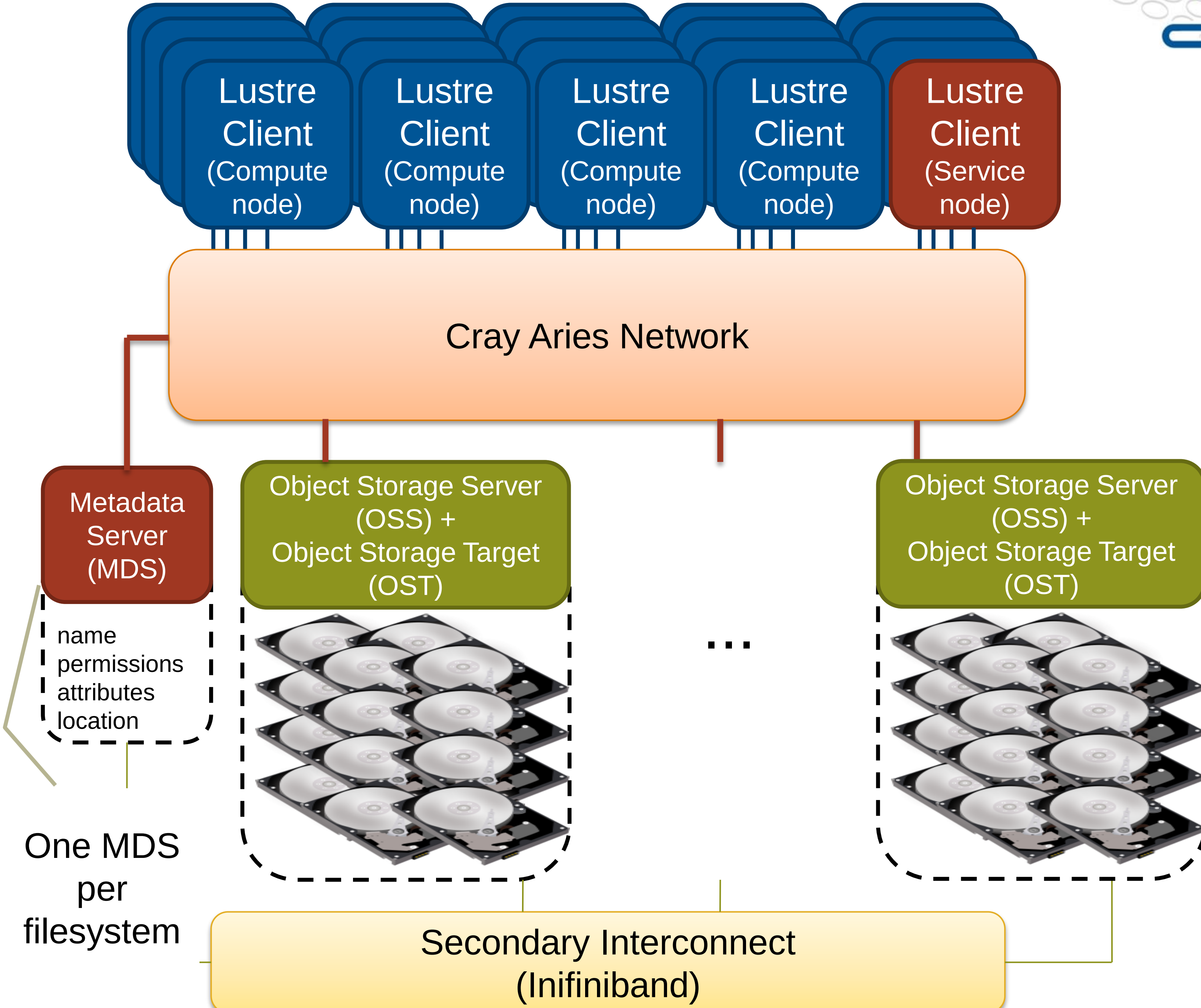
Optical Connections

Copper Connections

# Why is the Dragonfly topology a good idea?

- **Scalability**
  - Topology scales to very large systems
- **Performance**
  - More than just a case of clever wiring, this topology leverages state-of-the-art adaptive routing that Cray developed with Stanford University
  - Smoothly mixes small and large messages eliminating need for a 2nd network for I/O traffic
- **Simplicity**
  - Implemented *without* external switches
  - No HBAs or separate NICs and Routers
- **Price/Performance**
  - Dragonfly maximizes the use of backplanes and passive copper components
  - Dragonfly minimizes the use of active optical components

# Storage


Lustre Client (Compute node)
Lustre Client (Compute node)
Lustre Client (Compute node)
Lustre Client (Compute node)
Lustre Client (Service node)
Cray Aries Network
Metadata Server (MDS)
name
permissions
attributes
location
Object Storage Server (OSS) + Object Storage Target (OST)
...
Object Storage Server (OSS) + Object Storage Target (OST)
One MDS per filesystem
Secondary Interconnect (Inifiniband)

# Sonexion: Only Three Components

## MMU: *Metadata Management Unit*

1

**Fully integrated metadata module**

- Lustre Metadata software
- Metadata disk storage
- Dual redundant management servers
- Metadata storage target RAID

## SSU: *Scalable Storage Unit*

2

**Fully integrated storage module**

- Storage controller, Lustre server
- Disk controller, RAID engine
- High speed storage
- Provides both capacity and performance

3


**Fully prepared rack**

- Prewired for InfiniBand, Ethernet and power
- Ready for instant expansion

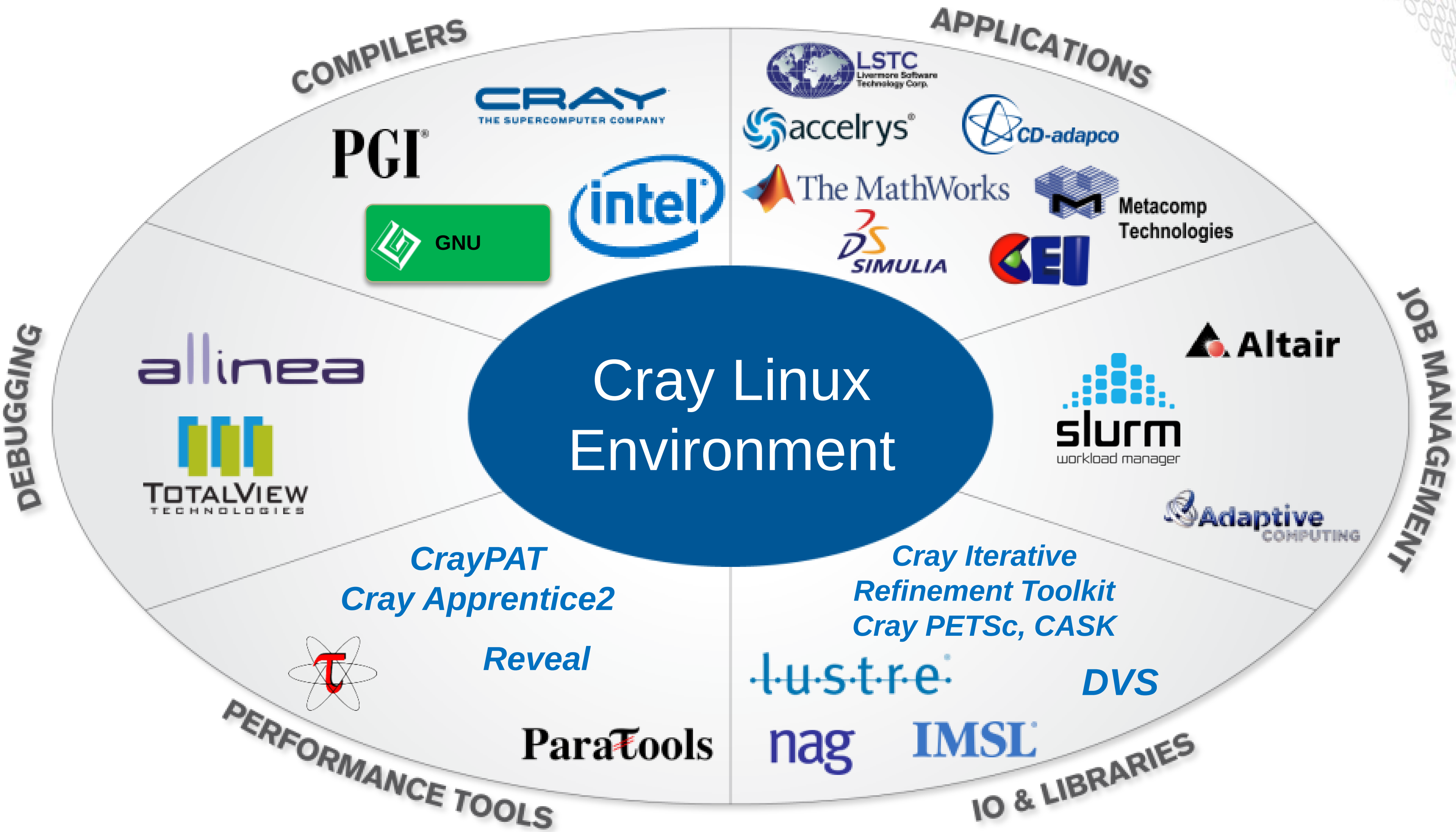
# The Cray Programming Environment

## Building software for the Cray XC30

# Vision

- **Cray systems are designed to be High Productivity as well as High Performance Computers**
- **The Cray Programming Environment (PE) provides a simple consistent interface to users and developers.**
  - Focus on improving scalability and reducing complexity
- **The default Programming Environment provides:**
  - the highest levels of application performance
  - a rich variety of commonly used tools and libraries
  - a consistent interface to multiple compilers and libraries
  - an increased automation of routine tasks
- **Cray continues to develop and refine the PE**
  - Frequent communication and feedback to/from users
  - Strong collaborations with third-party developers

# Cray Software Ecosystem



# Cray's Supported Programming Environment

## Programming Languages

- Fortran
- C
- C++
- Python

## Programming models

- Distributed Memory (Cray MPT)
  - MPI
  - SHMEM
- Shared Memory
  - OpenMP 3.0
  - OpenACC
- PGAS & Global View
  - UPC (CCE)
  - CAF (CCE)
  - Chapel

## Compilers

- Cray Compiling Environment (CCE)
- GNU
- 3rd Party Compilers
  - Intel Composer
  - PGI

## Tools

**Environment setup**

- Modules

**Debuggers**

- TotalView Technologies
- Allinea (DDT)
- lgdb

**Debugging Support Tools**

- Abnormal Termination Processing
- STAT

**Performance Analysis**

- CrayPat
- Cray Apprentice²

**Scoping Analysis**

- Reveal

## Optimized Scientific Libraries

- LAPACK
- ScaLAPACK
- BLAS (libgoto)
- Iterative Refinement Toolkit
- Cray Adaptive FFTs (CRAFFT)
- FFTW
- Cray PETSc (with CASK)
- Cray Trilinos (with CASK)

## I/O Libraries

- NetCDF
- HDF5

Cray developed
Licensed ISV SW
3rd party packaging
Cray added value to 3rd party

# The Cray Compilation Environment (CCE)

- **The default compiler on XE and XC systems**
  - Specifically designed for HPC applications
  - Takes advantage of Cray's experience with automatic vectorization and and shared memory paralleization

- **Excellent standards support for multiple languages and programming models**
  - Fortran 2008 standards compliant
  - C++98/2003 compliant (working on C++11)
  - OpenMP 3.1 compliant, working on OpenMP 4.0
  - OpenACC 1.0 compliant (working on OpenACC 2.0)

- **Full integrated and optimised support for PGAS languages**
  - UPC 1.2 and Fortran 2008 coarray support
  - No preprocessor involved
  - Full debugger support (With Allinea DDT)

- **OpenMP and automatic multithreading fully integrated**
  - Share the same runtime and resource pool
  - Aggressive loop restructuring and scalar optimization done in the presence of OpenMP
  - Consistent interface for managing OpenMP and automatic multithreading

# Cray MPI & SHMEM

- **Cray MPI**
  - Implementation based on MPICH2 from ANL
  - Includes many improved algorithms and tweaks for Cray hardware
    - Improved algorithms for many collectives
    - Asynchronous progress engine allows overlap of computation and comms
    - Customizable collective buffering when using MPI-IO
    - Optimized Remote Memory Access (one-sided) fully supported including passive RMA
  - Full MPI-2 support with the exception of
    - Dynamic process management (MPI_Comm_spawn)
  - MPI-3 support coming soon

- **Cray SHMEM**
  - Fully optimized Cray SHMEM library supported
    - Fully compliant with OpenSHMEM v1.0
    - Cray XC implementation close to the T3E model

# Cray Scientific Libraries

| FFT | Dense | Sparse |
|---|---|---|
| CRAFFT | BLAS | CASK |
| FFTW | LAPACK | PETSc |
| P-CRAFFT | ScaLAPACK | Trilinos |
| | IRT | |
| | CASE | |

**IRT – Iterative Refinement Toolkit**
**CASK – Cray Adaptive Sparse Kernels**
**CRAFFT – Cray Adaptive FFT**
**CASE – Cray Adaptive Simplified Eigensolver**

# Cray Performance Analysis Tools (PAT)

- **From performance measurement to performance analysis**
- **Assist the user with application performance analysis and optimization**
  - Help user identify important and meaningful information from potentially massive data sets
  - Help user identify problem areas instead of just reporting data
  - Bring optimization knowledge to a wider set of users
- **Focus on ease of use and intuitive user interfaces**
  - Automatic program instrumentation
  - Automatic analysis
- **Target scalability issues in all areas of tool development**

# Debuggers on Cray Systems

- **Systems with hundreds of thousands of threads of execution need a new debugging paradigm**
  - Innovative techniques for productivity and scalability
    - Scalable Solutions based on MRNet from University of Wisconsin
    - STAT - Stack Trace Analysis Tool
      - Scalable generation of a single, merged, stack backtrace tree
      - running at 216K back-end processes
    - ATP - Abnormal Termination Processing
      - Scalable analysis of a sick application, delivering a STAT tree and a minimal, comprehensive, core file set.
    - Fast Track Debugging
      - Debugging optimized applications
      - Added to Allinea's DDT 2.6 (June 2010)
    - Comparative debugging
      - A data-centric paradigm instead of the traditional control-centric paradigm
      - Collaboration with Monash University and University of Wisconsin for scalability
  - Support for traditional debugging mechanism
    - TotalView, DDT, and gdb

# Controlling the environment with modules

# Modules

- **The Cray Programming Environment uses the GNU “modules” framework to support multiple software versions and to create integrated software packages**
  - As new versions of the supported software and associated man pages become available, they are installed and added to the Programming Environment as a new version, while earlier versions are retained to support legacy applications
  - System administrators will set the default version of an application, or you can choose another version by using modules system commands
  - Users can create their own modules, or administrators can install site specific modules available to many users.

# Viewing the current module state

- Each login session has its own module state which can be modified by loading, swapping or unloading the available modules.
- This state affects the functioning of the compiler wrappers and in some cases runtime of applications.
- A standard, default set of modules is always loaded at login for all users.
- Current state can be viewed by running:

```
$> module list
```

# Default modules example

```
tedwards@swan:~> module list
Currently Loaded Modulefiles:
  1) modules/3.2.6.7
  2) nodestat/2.2-1.0500.41375.1.85.ari
  3) sdb/1.0-1.0500.43793.6.11.ari
  4) alps/5.0.3-2.0500.8095.1.1.ari
  5) MySQL/5.0.64-1.0000.7096.23.1
  6) lustre-cray_ari_s/2.3_3.0.58_0.6.6.1_1.0500.7272.12.1-1.0500.44935.7.1
  7) udreg/2.3.2-1.0500.6756.2.10.ari
  8) ugni/5.0-1.0500.0.3.306.ari
  9) gni-headers/3.0-1.0500.7161.11.4.ari
 10) dmapp/6.0.1-1.0500.7263.9.31.ari
 11) xpmem/0.1-2.0500.41356.1.11.ari
 12) hss-llm/7.0.0
 13) Base-opts/1.0.2-1.0500.41324.1.5.ari
 14) craype-network-aries
 15) craype/1.06.05
 16) cce/8.2.0.181
...
```

# Viewing available modules

- **There may be many hundreds of possible modules available to users.**
  - Beyond the pre-loaded defaults there are many additional packages provided by Cray
  - Sites may choose to install their own versions.
- **Users can see all the modules that can be loaded using the command:**
  - `module avail`
- **Searches can be narrowed by passing the first few characters of the desired module, e.g.**

```
tedwards@swan:~> module avail gc


------------------------------- /opt/modulefiles ----------------------------
gcc/4.6.1            gcc/4.7.2            gcc/4.8.0
gcc/4.6.3            gcc/4.7.3            gcc/4.8.1(default)
```

# Further refining available modules

- **`avail [avail-options] [path...]`**
  - List all available modulefiles in the current `MODULEPATH`
- **Useful options for filtering**
  - `-U, --usermodules`
    - List all modulefiles of interest to a typical user
  - `-D, --defaultversions`
    - List only default versions of modulefiles with multiple available versions
  - `-P, --prgenvmodules`
    - List all PrgEnv modulefiles
  - `-T, --toolmodules`
    - List all tool modulefiles
  - `-L, --librarymodules`
    - List all library modulefiles
  - `% module avail <product>`
    - List all <product> versions available

# Modifying the default environment

- **Loading, swapping or unloading modules:**
  - The default version of any inidividual modules can be loaded by name
    - e.g.: `module load perftools`
  - A specific version can be specified after the forward slash.
    - e.g.: `module load perftools/6.1.0`
  - Modules can be swapped out in place
    - e.g.: `module swap intel intel/13.1.1.163`
  - Or removed entirely
    - e.g.: `module unload perftools`
- **Modules will automatically change values of variables like `PATH`, `MANPATH`, `LM_LICENSE_FILE`... etc**
  - Modules also provide a simple mechanism for updating certain environment variables, such as `PATH`, `MANPATH`, and `LD_LIBRARY_PATH`
  - In general, you should make use of the modules system rather than embedding specific directory paths into your startup files, makefiles, and scripts

# Summary of Useful module commands

- **Which modules are available?**
  - `module avail`, `module avail cce`
- **Which modules are currently loaded?**
  - `module list`
- **Load software**
  - `module load perftools`
- **Change programming environment**
  - `module swap PrgEnv-cray PrgEnv-gnu`
- **Change software version**
  - `module swap cce/8.0.2  cce/7.4.4`
- **Unload module**
  - `module unload cce`
- **Display module release notes**
  - `module help cce`
- **Show summary of module environment changes**
  - `module show cce`

# Compiling applications for the Cray XC

# Compiler Driver Wrappers (1)

- **All applications that will run in parallel on the Cray XC should be compiled with the standard language wrappers.**

  **The compiler drivers for each language are:**
  - `cc – wrapper around the C compiler`
  - `CC – wrapper around the C++ compiler`
  - `ftn – wrapper around the Fortran compiler`

- **These scripts will choose the required compiler version, target architecture options, scientific libraries and their include files automatically from the module environment.**

- **Use them exactly like you would the original compiler, e.g. To compile `prog1.f90` run**

```
ftn -c prog1.f90
```

# Compiler Driver Wrappers (2)

- **The scripts choose which compiler to use from the `PrgEnv` module loaded**

| PrgEnv | Description | Real Compilers |
|---|---|---|
| PrgEnv-cray | Cray Compilation Environment | crayftn, craycc, crayCC |
| PrgEnv-intel | Intel Composer Suite | ifort, icc, icpc |
| PrgEnv-gnu | GNU Compiler Collection | gfortran, gcc, g++ |
| ~~PrgEnv-pgi~~ | ~~Portland Group Compilers~~ | ~~pgf90, pgcc, pgCC~~ |

- **Use module swap to change `PrgEnv`, e.g.**
  - `module swap PrgEnv-cray PrgEnv-intel`
- **`PrgEnv-cray` is loaded by default at login. This may differ on other Cray systems.**
  - use `module list` to check what is currently loaded
- **The Cray MPI module is loaded by default (`cray-mpich`).**
  - To support SHMEM load the `cray-shmem` module.

# Compiler Versions

- **There are usually multiple versions of each compiler available to users.**
  - The most recent version is usually the default and will be loaded when swapping `PrgEnvs`.
  - To change the version of the compiler in use, swap the Compiler Module. e.g. `module swap cce cce/8.1.6`

| PrgEnv | Compiler Module |
|---|---|
| `PrgEnv-cray` | `cce` |
| `PrgEnv-intel` | `intel` |
| `PrgEnv-gnu` | `gcc` |
| ~~`PrgEnv-pgi`~~ | ~~`pgi`~~ |

# About the `–I`, `–L` and `–l` flags

- **For libraries and include files covered by module files, you should NOT add anything to your Makefile**
  - No additional MPI flags are needed (included by wrappers)
  - You do not need to add any `-I`, `-l` or `–L` flags for the Cray provided libraries

- **If your Makefile needs an input for `–L` to work correctly, try using '.'**

- **If you really, really need a specific path, try checking 'module show X' for some environment variables**

# OpenMP

- **OpenMP is support by all of the `PrgEnvs`.**
  - CCE (`PrgEnv-cray`) recognizes and interprets OpenMP directives by default. If you have OpenMP directives in your application but do not wish to use them, disable OpenMP recognition with `–hnoomp`.

| PrgEnv | Enable OpenMP | Disable OpenMP |
|---|---|---|
| `PrgEnv-cray` | `-homp` | `-hnoomp` |
| `PrgEnv-intel` | `-openmp` | |
| `PrgEnv-gnu` | `-fopenmp` | |
| ~~`PrgEnv-pgi`~~ | ~~`-mp`~~ | |

# Compiler man Pages

- **For more information on individual compilers**

| PrgEnv | C | C++ | Fortran |
|---|---|---|---|
| `PrgEnv-cray` | `man craycc` | `man crayCC` | `man crayftn` |
| `PrgEnv-intel` | `man icc` | `man icpc` | `man ifort` |
| `PrgEnv-gnu` | `man gcc` | `man g++` | `man gfortran` |
| ~~`PrgEnv-pgi`~~ | ~~`man pgcc`~~ | ~~`man pgCC`~~ | ~~`man pgf90`~~ |
| Wrappers | `man cc` | `man CC` | `man ftn` |

- **To verify that you are using the correct version of a compiler, use:**
  - -V option on a cc, CC, or ftn command with PGI, Intel and Cray
  - --version option on a cc, CC, or ftn command with GNU