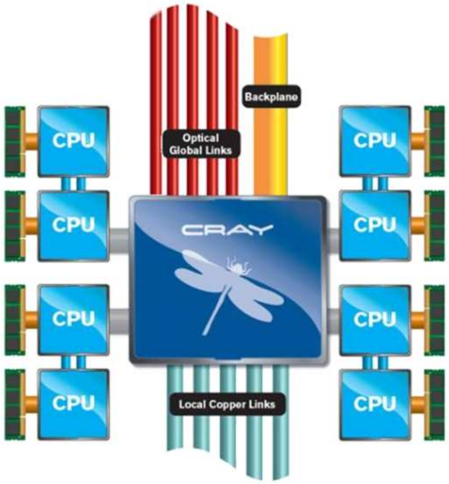
User Orientation on Cray XC40 SERC, IISc

Sudhakar Yerneni & Patricia Balle

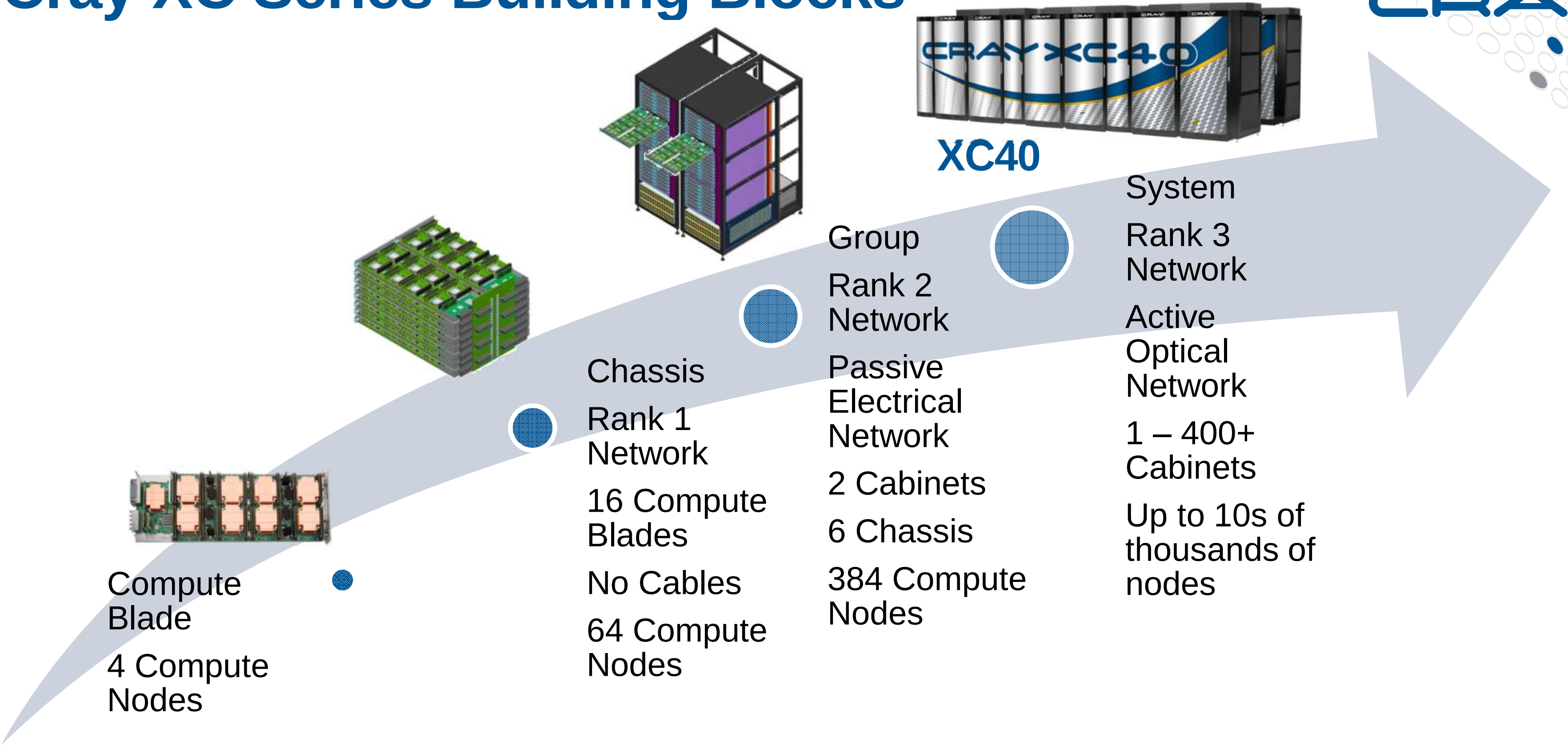
Agenda

- Introduction to Cray XC40 architecture.
- IISc's Cray system configuration
- Login & Filesystems
- Cray Programming environment
- Modules
- Compiling applications for the Cray XC
- Running applications on the Cray XC40
- Cray Scientific Libraries

Cray XC40 Architecture & Packaging



Cray XC Series Building Blocks



XC40

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

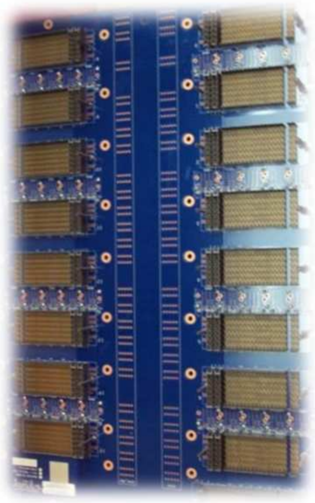
Active Optical Network

1 – 400+ Cabinets

Up to 10s of thousands of nodes

Cray XC Aries Network

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

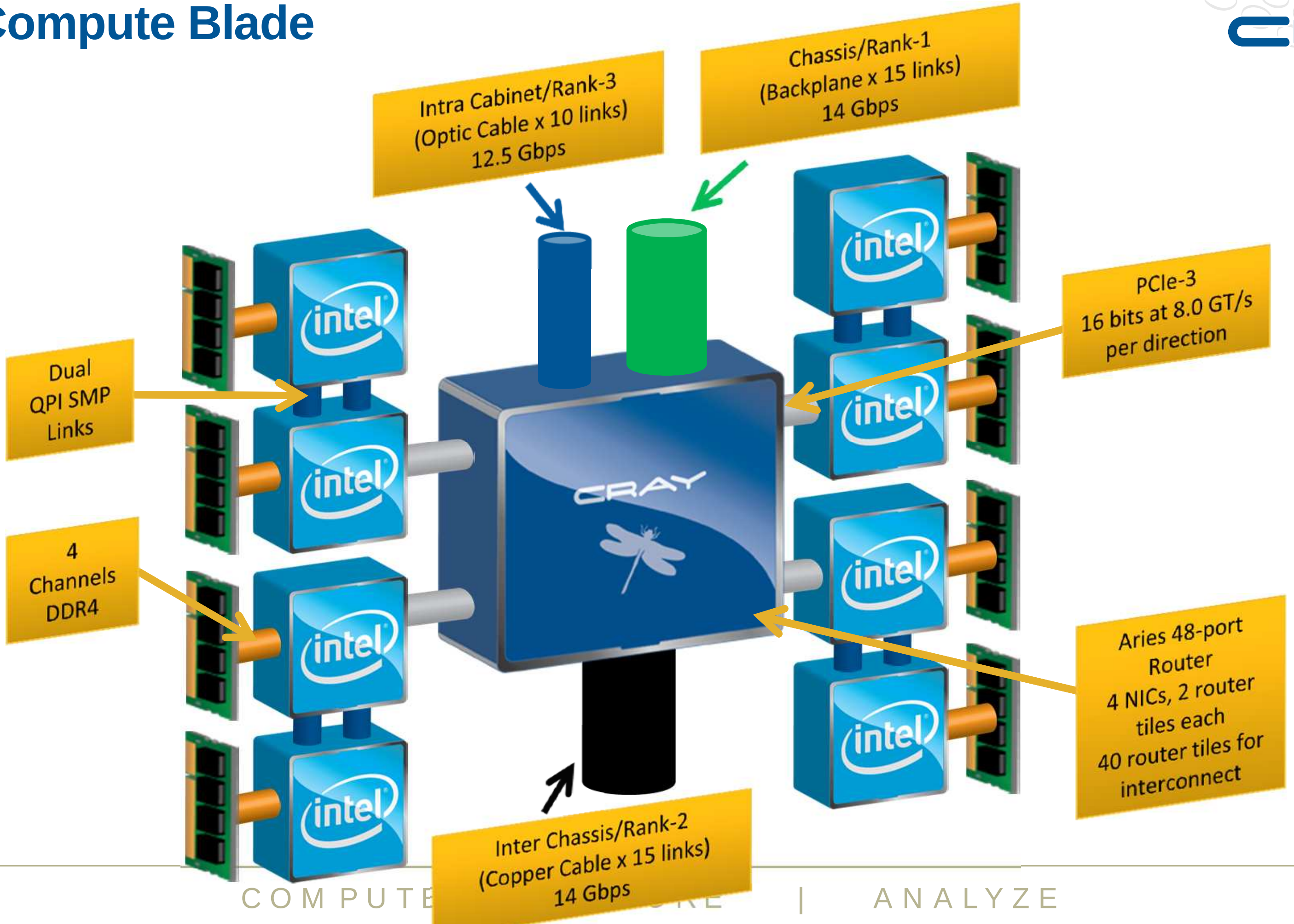


Rank-1
Backplane

Rank-2
Passive CU

Rank-3
Active Optics

XC40 Compute Blade



Intra Cabinet/Rank-3
(Optic Cable x 10 links)
12.5 Gbps

Chassis/Rank-1
(Backplane x 15 links)
14 Gbps

Dual
QPI SMP
Links

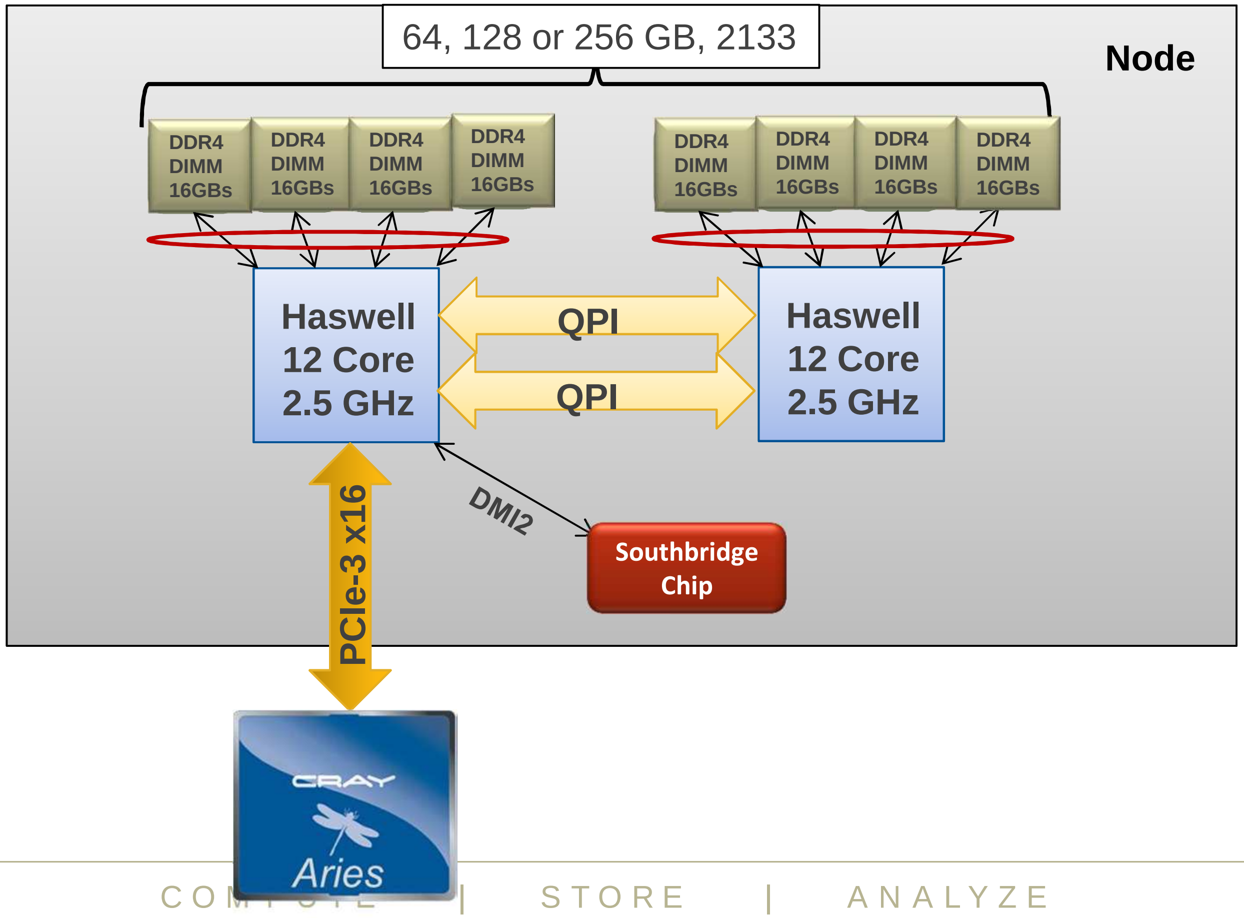
PCIe-3
16 bits at 8.0 GT/s
per direction

4
Channels
DDR4

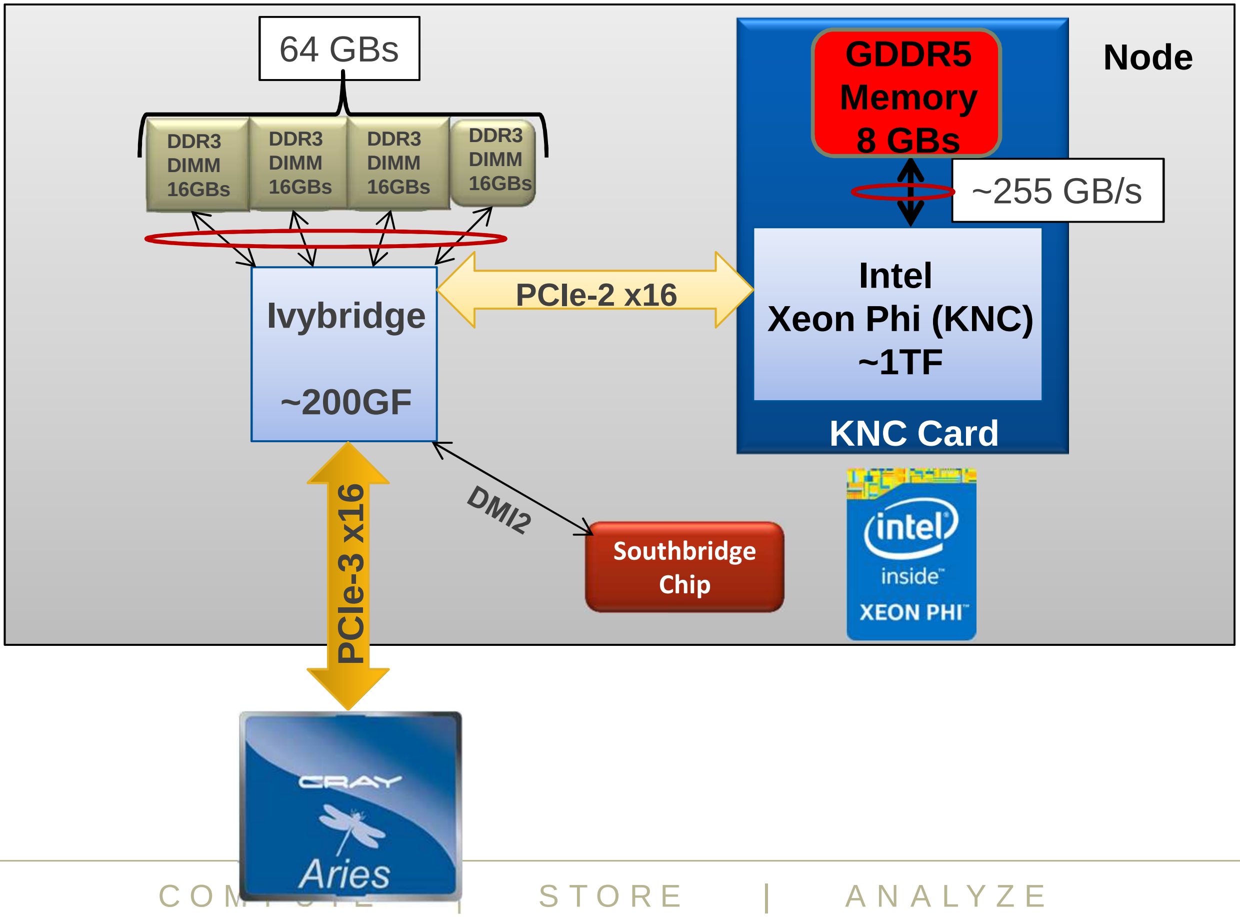
Aries 48-port
Router
4 NICs, 2 router
tiles each
40 router tiles for
interconnect

Inter Chassis/Rank-2
(Copper Cable x 15 links)
14 Gbps

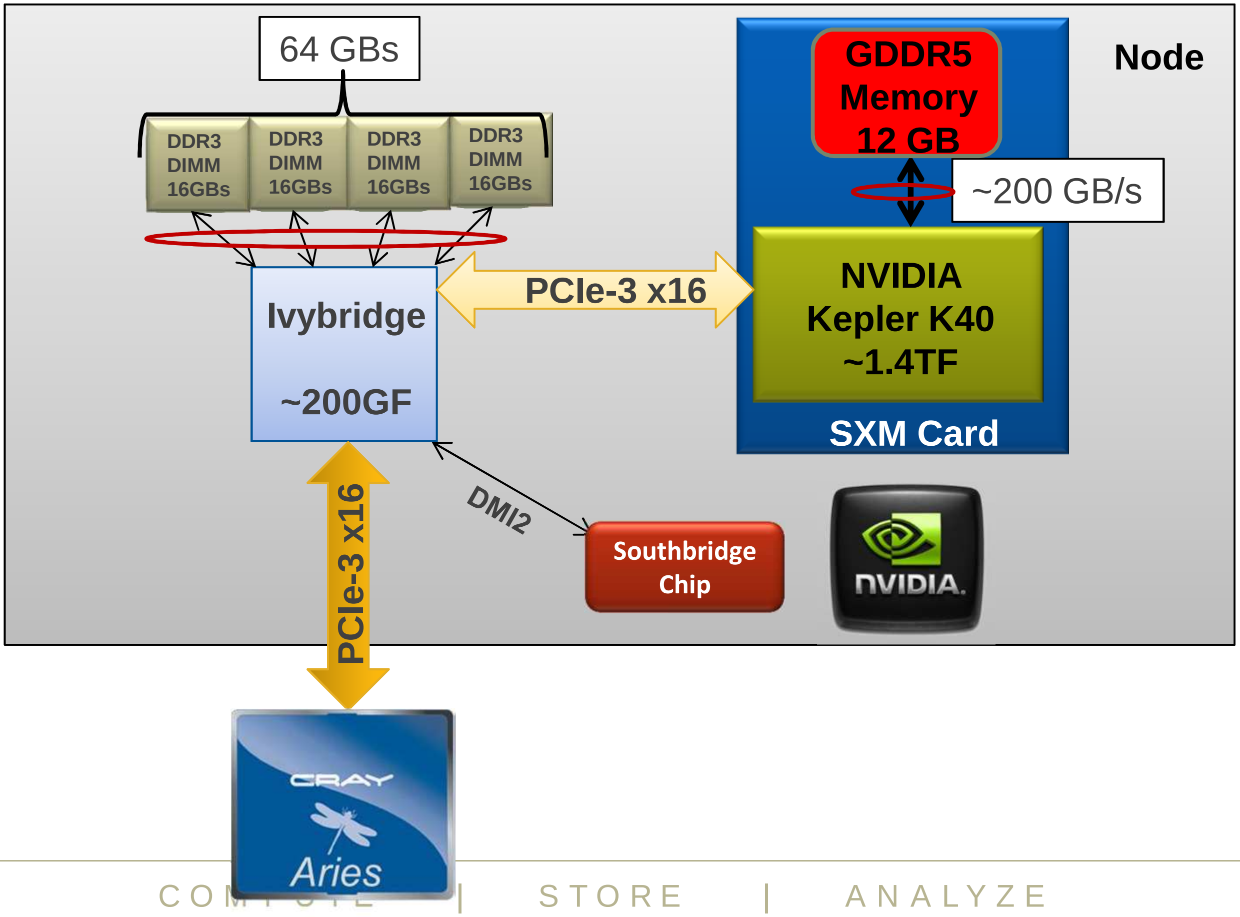
Cray XC40 Compute Node



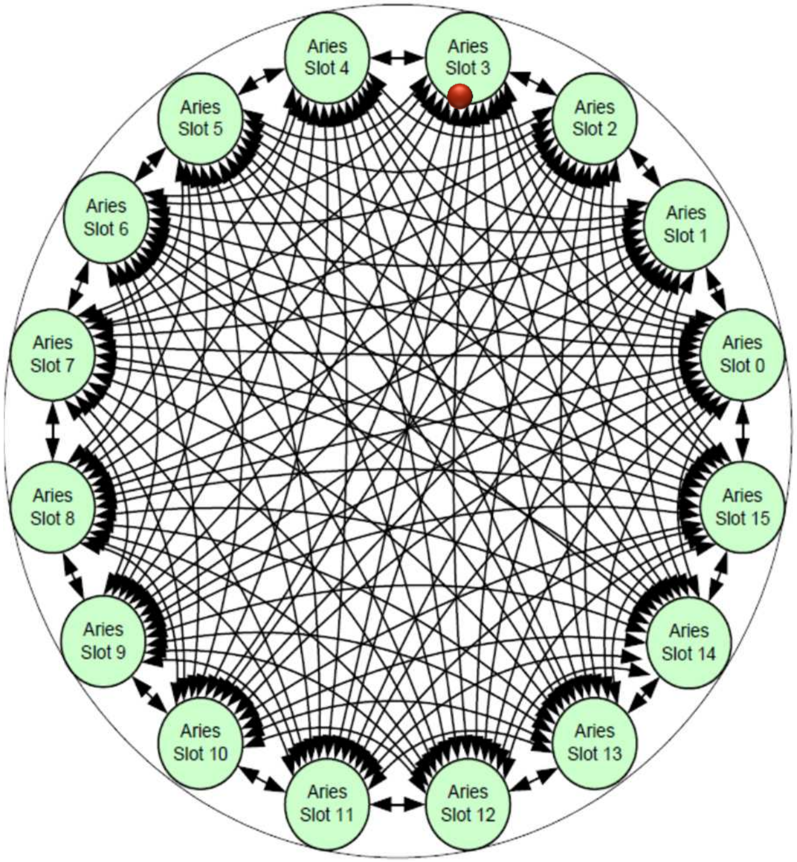
Cray XC Xeon Phi Node

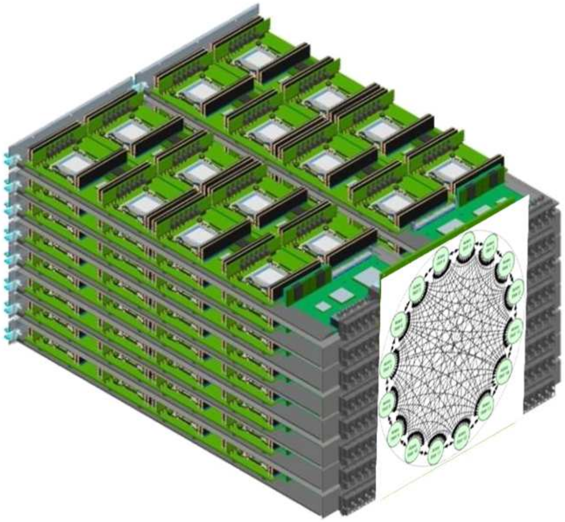


Cray XC Kepler Node



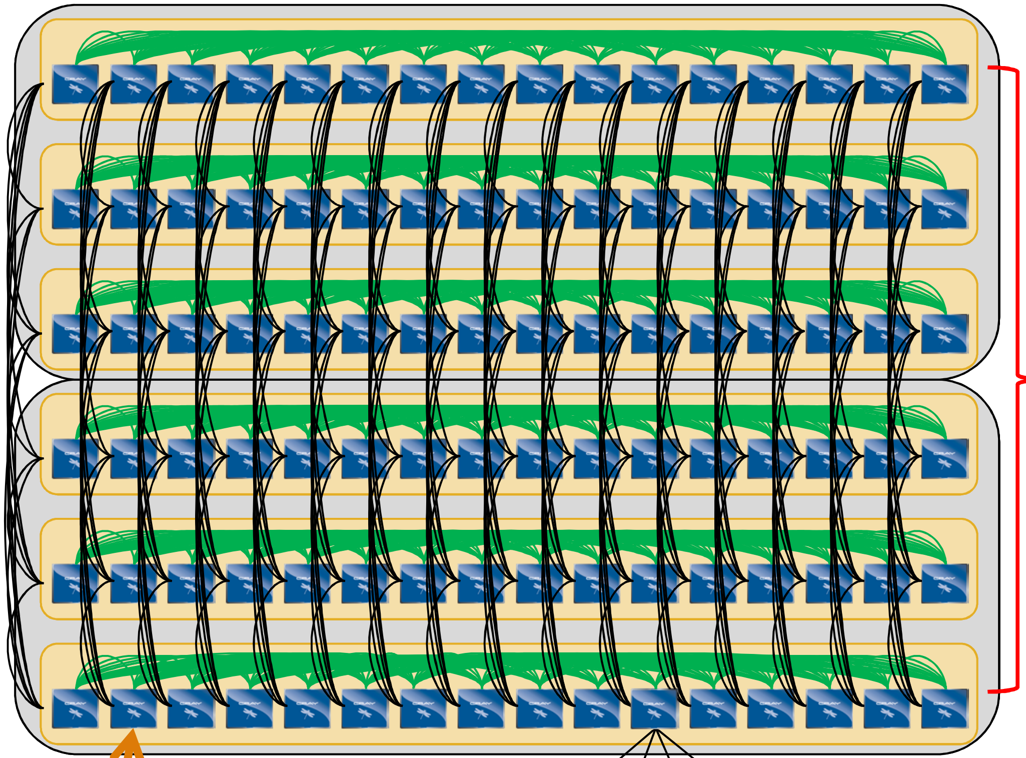
Cray XC Rank1 Network



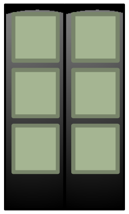


- Chassis with 16 compute blades
- 128 Sockets
- Inter-Aries communication over backplane
- Per-Packet adaptive Routing

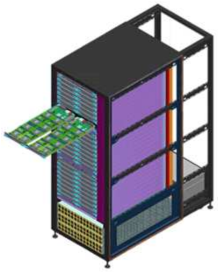
Cray XC Rank-2 Copper Network



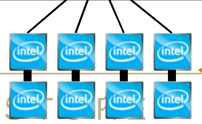
2 Cabinet Group
768 Sockets



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

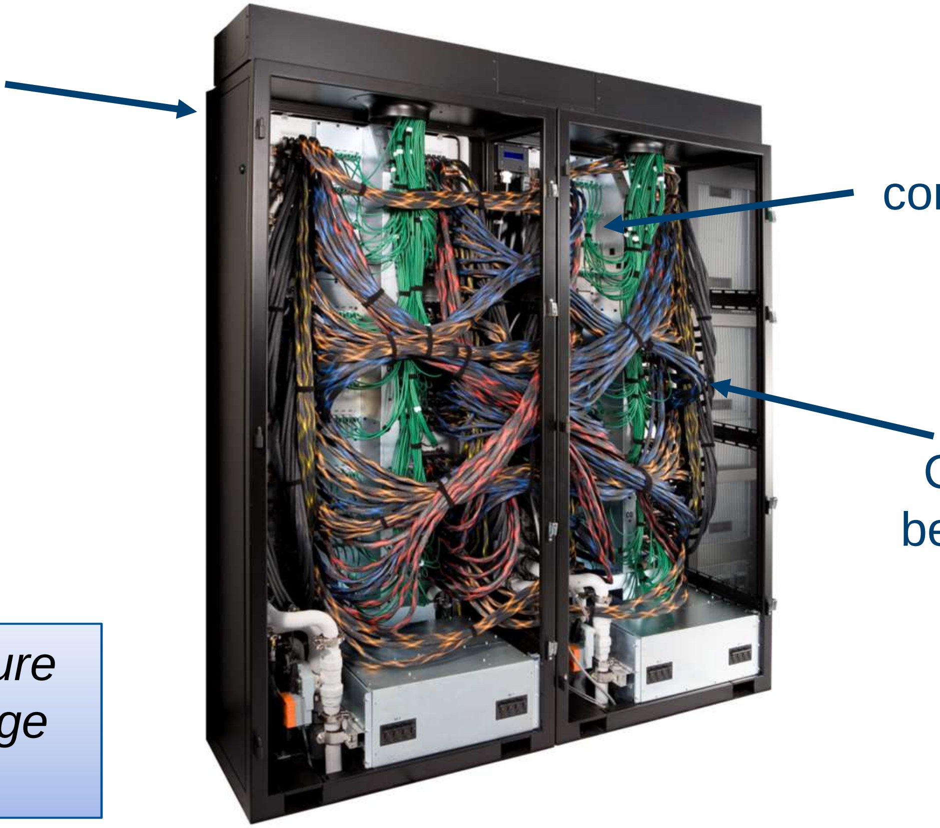


16 Aries connected by backplane “Green Network”



4 nodes connect to a single Aries

768 Sockets less than 1μs away



2-cabinet Group

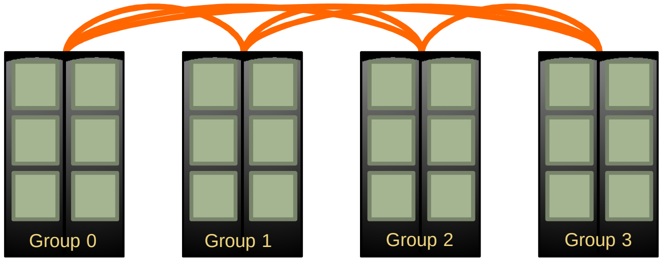
Backplane connections within chassis

Copper cables between chassis

This basic structure is repeated in large systems

Cray XC Network Overview – Rank-3 Network

- An all-to-all pattern is wired between the groups using optical cables
- 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



SERC, IISc system: 4-group system is interconnected with 6 optical "bundles".

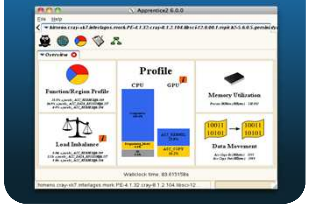
Full Support for Diverse Users

Cray provides great support across the full spectrum of HPC user types

Load & Go

Build & Go

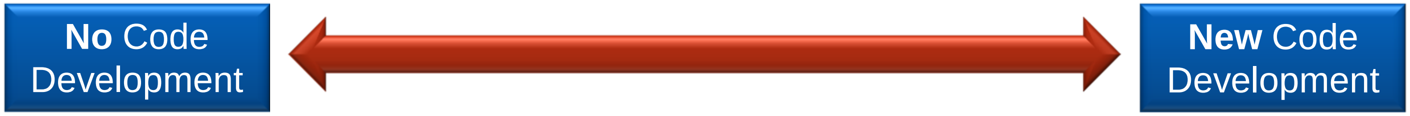
Tune & Go



Wallclock time: 83.615158s

homens.cray-xk7.interlagos.mork.PE-4.1.32.cray-8.1.2.104.libsci-12

Code & Go



No Code Development

New Code Development

CLE Can Adapt to Different Application Requirements

CLE

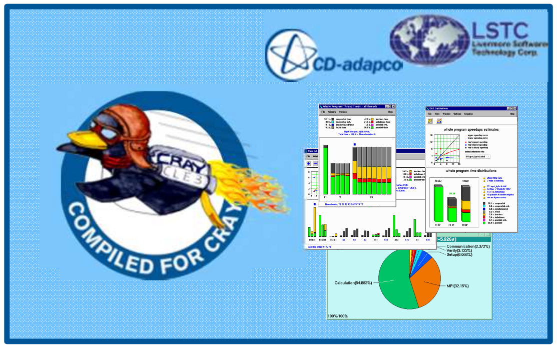
CRAY LINUX ENVIRONMENT

CCM *–Cluster Compatibility Mode*

- No compromise *compatibility*
- Fully standard x86/Linux
- Standardized Communication Layer
- Out-of-the-box ISV Installation
- ISV applications simply install and run

ESM *– Extreme Scalability Mode*

- No compromise *scalability*
- Low-Noise Kernel for scalability
- Native Comm. & Optimized MPI
- Application-specific performance tuning and scaling



CLE run mode is set by the user on a job-by-job basis to provide full flexibility

Cray Programming Environment Distribution Focus on Performance and Productivity

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)
- intel
- PGI®

Tools

Environment setup

- Modules

Debuggers

- TotalView Technologies
- DDT
- lgdb

Debugging Support Tools

- Abnormal Termination Processing
- STAT

Performance Analysis

- CrayPat
- Cray Apprentice^2

Scoping Analysis

- Reveal

Optimized Scientific Libraries

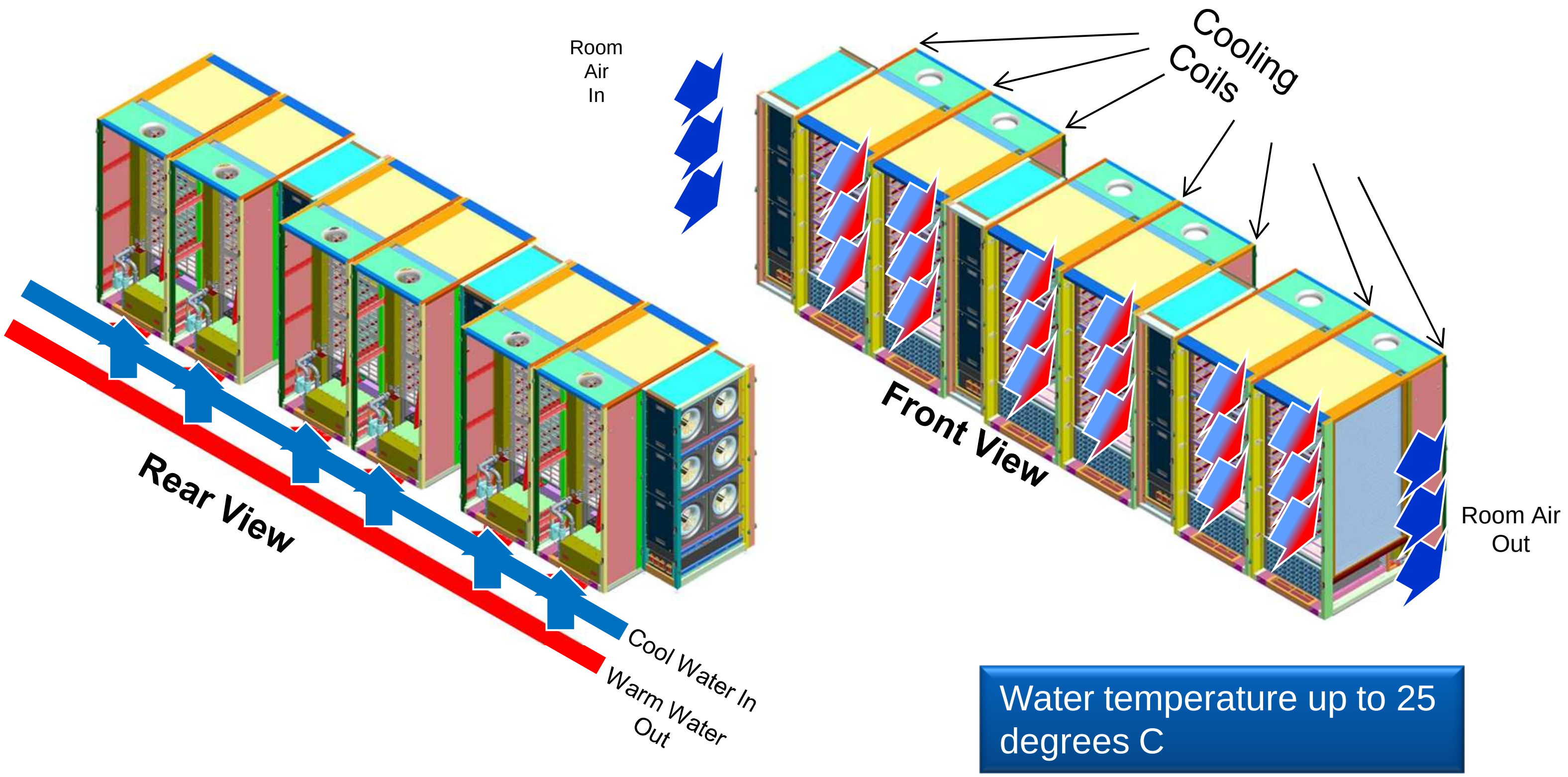
- LAPACK
- ScaLAPACK
- BLAS (libgoto)
- Iterative Refinement Toolkit
- Cray Adaptive FFTs (CRAFFT)
- FFTW
- Cray PETSc (with CASK)
- Cray Trilinos (with CASK)

I/O Libraries

- NetCDF
- HDF5

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

XC Series Transvers Cooling Advantage



Water temperature up to 25 degrees C

IISc system Configuration

System Configuration – Compute (H/W)

Cray XC40 System	System Configuration
Cray XC40 cabinets	8
Interconnect Network	Cray Aries with Dragonfly topology
Total Memory (TiB)	169.5
Total Peak Performance	1.45 PF
Compute Nodes	
Number of CPU Nodes	1376
CPU Type	Intel Haswell 12-Core 2.5 GHz
Memory Per Node / Total	128GB / 176 TB
Peak Performance Per Node / Total	960 GF / 1.32 PF
GPU	NVIDIA K40
Number of CPU-GPU Nodes	44
Xeon Phi / MIC	Intel Xeon-Phi
Number of CPU-Phi nodes	48
Service & IO nodes	56

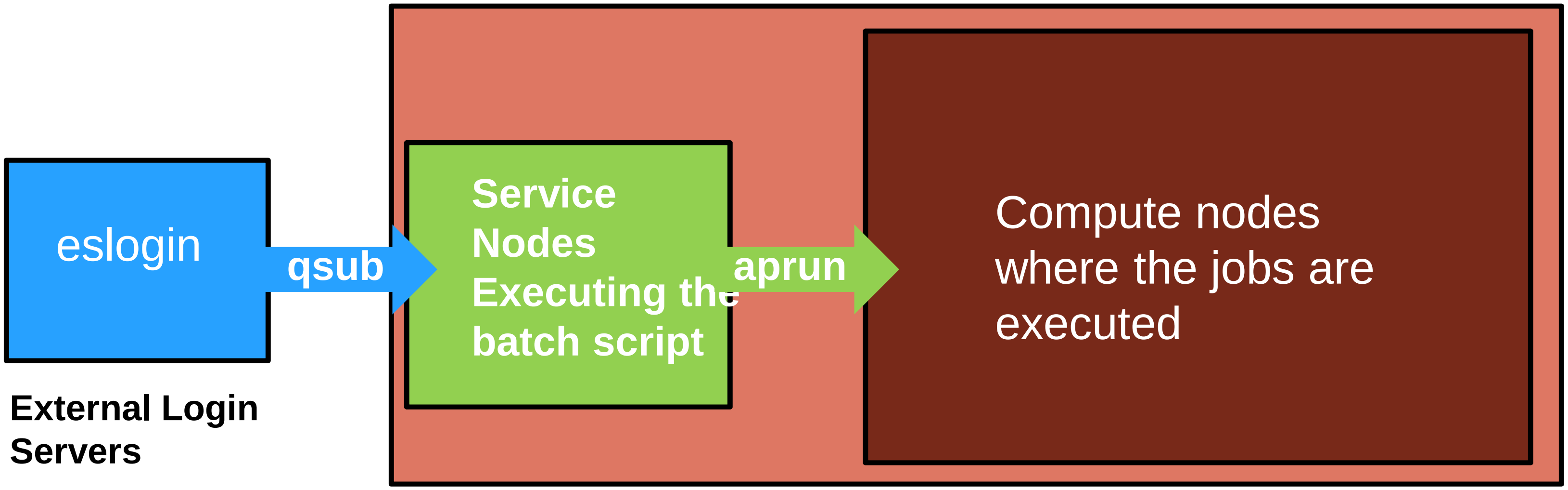
System Configuration – Storage (H/W)

Cray XC40 System	System Configuration
Storage cabinets	4
System Storage	Boot RAID
Storage Technology and File System Type	Cray Lustre File System, Lustre
Useable Storage Capacity	2 PB
Storage Array	2 x DDN SFA12000-40
OSS Disks - Density	960 x 3 TB NL SAS

System Configuration - Software

Software		License
OS	Cray Linux Environment	Yes
Compiler	Cray Compilation Environment (CCE)	30 seats
	GNU Complier	Yes
	Intel Composer XE for Linux	5 Seats
gdb and PAPI	gdb and PAPI	Yes
Programming Environment	Cray Developer Toolkit (CDT)	Yes
Profilers	Cray Performance and Analysis Toolkit (CPAT)	1 seat
Job schedler	Workload Manager: PBS Pro	Yes
Parallel debugger	Allinea DDT	2,048 processes

System Access


eslogin
qsub
Service Nodes Executing the batch script
aprun
Compute nodes where the jobs are executed
External Login Servers

Cray XC40 mainframe includes service nodes, network nodes and many compute nodes

How do I get to the system?

- Send a request to for new-user request – follow the institute procedure for this.
- Once your new account has been created, log in to:
  “xc40.serc.iisc.ernet.in”
- Your home area ➔ /home/*user_account* (UFS)
  - Limited size (around 1.4 TB)
- Lustre is mounted at “/mnt/lustre/”
  - Compile and execute your programs from this location
  - Separate directories have been created for each user
  - Quota system may kick in as per the institute procedure

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'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)

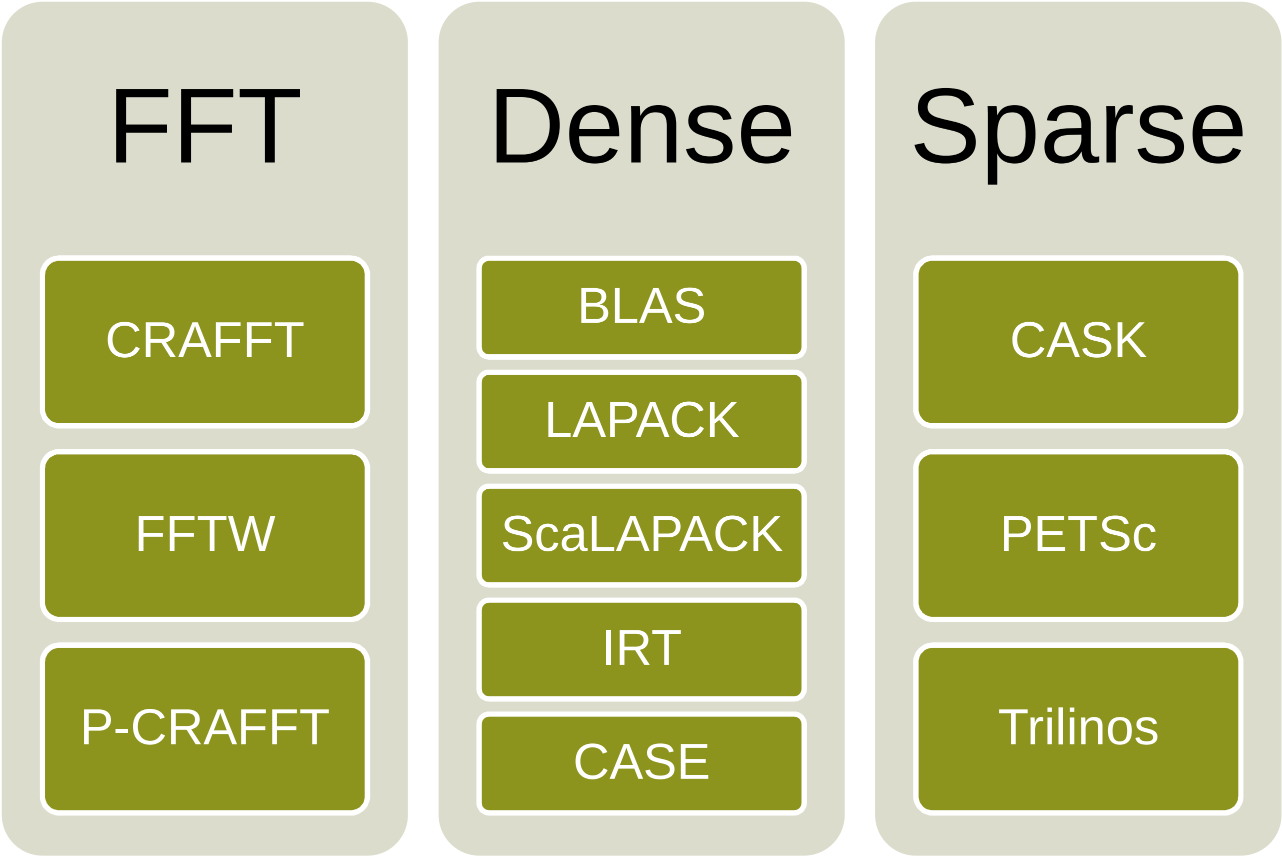
- **The default compiler on XC systems**
  - Specifically designed for HPC applications
  - Takes advantage of Cray's experience with automatic vectorization and and shared memory parallelization
- **Excellent standards support for multiple languages and programming models**
  - Fortran 2008 standards compliant
  - C++98/2003 compliant (C++11 schedule for v8.4)
  - OpenMP 3.1 compliant (OpenMP 4.0 coming soon)
  - OpenACC 2.0 compliant
- **Full integrated and optimised support for PGAS languages**
  - UPC 1.3 and Fortran 2008 coarray support
  - No preprocessor involved
  - Full debugger support (With Allinea DDT)
- **OpenMP and automatic multithreading fully integrated**
  - 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-3 support with the exception of
    - Dynamic process management (MPI_Comm_spawn)

- **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
- CRAFFT
- FFTW
- P-CRAFFT

Dense
- BLAS
- LAPACK
- ScaLAPACK
- IRT
- CASE

Sparse
- CASK
- PETSc
- Trilinos

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
      - Allows 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 mechanisms
    - DDT and gdb at SERC

An introduction to modules

How to use modules to control the environment

Environment Setup

- The Cray XC system uses the GNU “modules” framework in the user environment 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 added automatically to the Programming Environment as a new version, while earlier versions are retained to support legacy applications
- System admins 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 admins can install site-specific modules available to many users
- Modules are very flexible and powerful and allow the user to dynamically manage their programming environment

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 from SERC system

```
 $ module list
Currently Loaded Modulefiles:
  1) modules/3.2.6.7                          12) Base-opts/1.0.2-1.0502.53325.1.2.ari
  2) nodestat/2.2-1.0502.53712.3.109.ari      13) craype-network-aries
  3) sdb/1.0-1.0502.55976.5.27.ari            14) craype/2.2.1
  4) alps/5.2.1-2.0502.9072.13.1.ari          15) cce/8.3.5
  5) lustre-cray_ari_s/2.5_3.0.101_0.31.1_1.0502.8394.10.1-1.0502.17871.10.3
                                              16) cray-libsci/13.0.1
  6) udreg/2.3.2-1.0502.9275.1.12.ari         17) pmi/5.0.6-1.0000.10439.140.2.ari
  7) ugni/5.0-1.0502.9685.4.24.ari            18) rca/1.0.0-2.0502.53711.3.127.ari
  8) gni-headers/3.0-1.0502.9684.5.2.ari      19) atp/1.7.5
  9) dmapp/7.0.1-1.0502.9501.5.219.ari        20) PrgEnv-cray/5.2.40
 10) xpmem/0.1-2.0502.55507.3.2.ari           21) nodehealth/5.1-1.0502.56494.9.2.ari
 11) hss-llm/7.2.0                            22) pbs/12.2.402.142964
```

(13 Jan: Note to Cray admin – cray-mpich module is missing!)

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.**

```
1006 $ module avail gcc

------------------------------------------- /opt/modulefiles -----------------
gcc/4.8.1          gcc/4.9.1(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 individual module 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/8.3.3  cce/8.2.6`
- **Unload module**
  - `module unload cce`
- **Display module release notes**
  - `module help cce`
- **Show summary of module environment changes**
  - `module show cce`

Targeting different node types

- **Compiling for the CPU nodes:**
  - module load craype-haswell

  (module craype-ivybridge will give an executable that works on the Haswell nodes but won't enable the Haswell-specific instructions)

- **Compiling for the GPU nodes:**
  - module load craype-accel-nvidia35
  - "module display craype-accel-nvidia35" tells you that this module also loads cudatoolkit and cray-libsci-acc

- **Compiling for Phi nodes (offload mode):**
  - module load craype-ivybridge (since the host nodes are Ivybridge)

- **Compiling for the Phi nodes (native mode):**
  - module load craype-intel-knc

Compiling applications for the Cray XC

Compiler Driver Wrappers

- **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 (cont.)

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++`

- **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.

Which compiler do I use?

- **All compilers provide Fortran, C, C++ and OpenMP support**

- **If your site offers you a choice, experiment with the various compilers**
  - Mixing binaries created by different compilers may cause issues

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

About the `–I`, `–L` and `–l` flags

- **For libraries and include files being triggered 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	

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
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

Running applications on the Cray XC40

First Steps...

How applications run on a Cray XC

- **Most Cray XC40s are batch systems.**
  - Users submit batch job scripts to a scheduler from a login node (e.g. PBS for SERC) for execution at some point in the future.
  - Each job requires resources and a prediction of how long it will run.
  - The scheduler (running on an external server) chooses which jobs to run and allocates appropriate resources
  - The batch system will then execute the user's job script on a different login or batch "MOM" node.
  - The scheduler monitors the job and kills any that overrun their runtime prediction.

- **User job scripts typically contain two types of statements.**
  1. Serial commands that are executed by the MOM node, e.g.
     - quick setup and post processing commands such as `rm`, `cd`, `mkdir`, etc.
  2. Parallel executables that run on compute nodes.
     - Launched using the **`aprun`** command.

The Two types of Cray XC Nodes

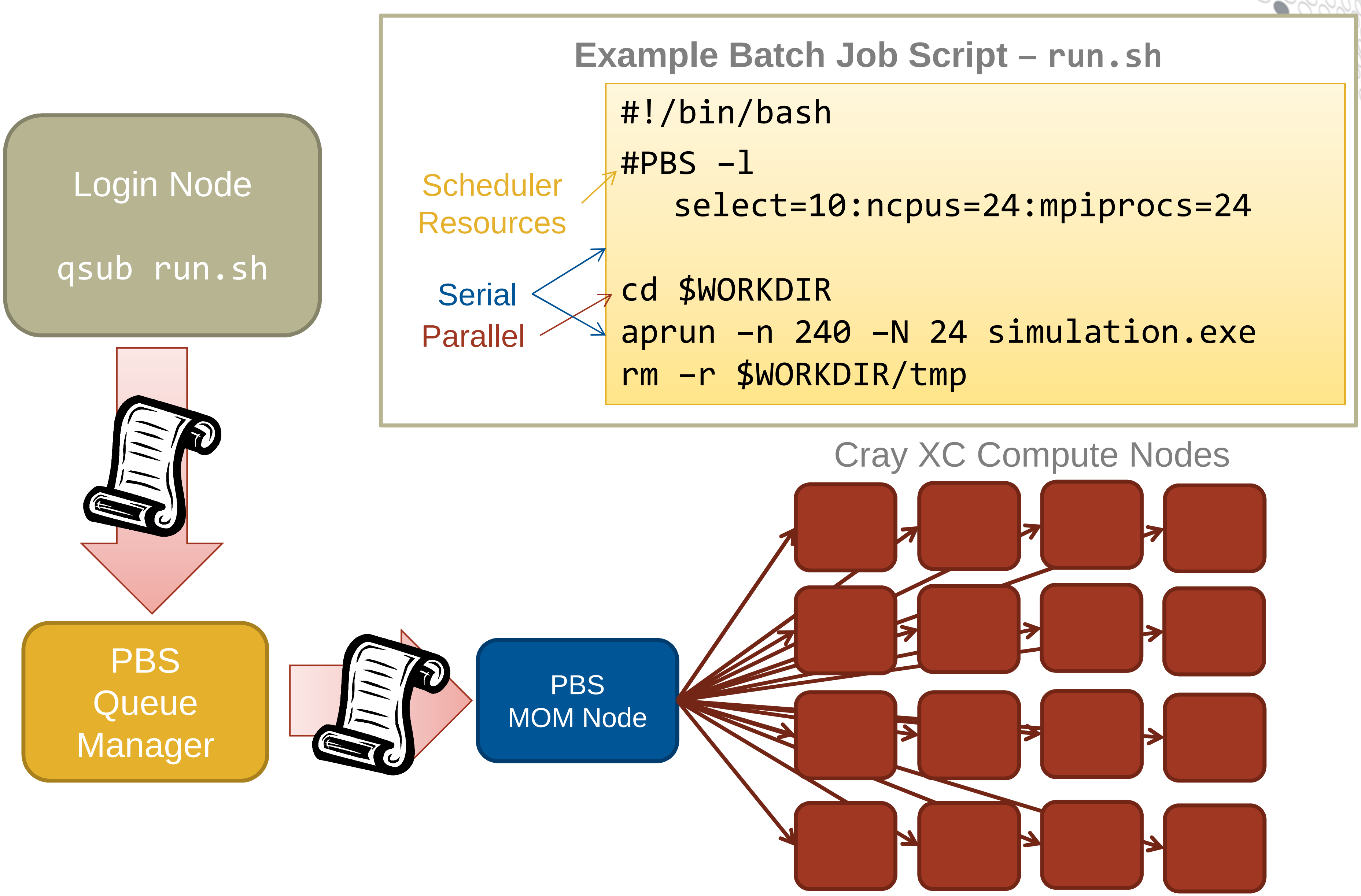
Login or service nodes

- This is the node you access when you first log in to the system.
- Runs a full version of the CLE operating system (all libraries and tools available)
- Used for editing files, compiling code, submitting jobs to the batch queue and other interactive tasks.
- Shared resources that may be used concurrently by multiple users.
- There may be many login nodes in any Cray XC40 and can be used for various system services (IO routers, daemon servers).
- They can be either connected to the Cray Aries network (internal login nodes) or proxies (external or esLogin nodes).

Compute nodes

- The nodes on which production jobs are executed
- Optimized for running batch jobs using Compute Node Linux (CNL)
- Can only be accessed by submitting jobs through ALPS
- Exclusive resources that may only be used by a single user.
- There are many more compute nodes in any Cray XC40 than login or service nodes.
- Always directly connected to the Cray Aries.

Running a batch job



Example Batch Job Script – `run.sh`

```
#!/bin/bash
#PBS -l
   select=10:ncpus=24:mpiprocs=24

cd $WORKDIR
aprun -n 240 -N 24 simulation.exe
rm -r $WORKDIR/tmp
```

Scheduling a batch application with PBS

- module load pbs (should be loaded by default)
- The number of required nodes can be specified in the job header
- The job is submitted by the `qsub` command
- At the end of the exection, output and error files are returned to submission directory
- You can check the status of jobs with: `qstat`
- You can delete a job with**:** **qdel**

Other PBS options

#PBS –j oe

- Combine stderr and stdout into one file

#PBS –mppwidth=N **DEPRECATED!**

- Start N PEs (tasks)

#PBS –mppnppn=M **DEPRECATED!**

- Start M tasks per node. Total Nodes used are N/M (+1 if mod(N,M)!=0)

#PBS –mppdepth=D **DEPRECATED!**

- Number of threads per task. Most used together with OMP_NUM_THREADS

#PBS –l select=10:ncpus=24:mpiprocs=24

- NEW way to reserve 10 nodes with 24 CPUs and 24 ranks per node

#PBS –V

- Export current environment to PBS job

#PBS –W depend=afterany:JOBID

- Don't start PBS job until after job JOBID has finished

(Note: #PBS –lnodes=4:ppn=24 is also deprecated)

Quick Old/New Syntax Comparison

select = number of compute nodes
ncpus = number of CPUs per compute node
mpiprocs = number of MPI processes per compute node

Example: 8 nodes with 24 CPUS each

Select Syntax	MPP Syntax
-l select=8:ncpus=24:mpiprocs=8	-l mppwidth=64,mppnppn=8,mppdepth=3
-l select=8:ncpus=24,place=scatter	-l mppwidth=8,mppnppn=1,mppdepth=2

- See “man pbs_resources” or the PBS Pro Users Guide for more details
- Note: important to set OMP_NUM_THREADS in run script explicitly otherwise the default sets it to the value of ncpus
- Look out for the placement policy setting (need place=scatter to avoid packing onto minimum node count)

PBS Queues on SERC System

Can select batch queue for a particular node type (CPU, GPU or PHI)

- qsub –q cpu_nodes myjob

```
crayadm@clogin72:~> qstat -q

server: sdb

Queue            Memory CPU Time Walltime Node   Run   Que   Lm
State
---------------- ------ -------- -------- ---- ----- ----- ----  -----
workq              --      --       --      --     1     0   --   D R
ccm_queue             --      --       --      --     0     0   --   E R
cpu_nodes            --      --       --      --    11     1   --   E R
gpu_nodes            --      --       --      --     0     0   --   E R
phi_nodes           --      --       --      --     0     0   --   E R
                                              ----- -----
                                                 12     1
```

Simple Job Scripts (old and new syntax)

Using mpp* with PBS

!!!! Deprecated !!!!

```
#!/bin/sh
#PBS -j oe
#PBS -l mppwidth=4
#PBS -l mppdepth=6
#PBS -l mppnppn=4
#PBS -l walltime=00:30:00
$PBS –q cpu_queue
# Change to the directory where job was submitted
cd $PBS_O_WORKDIR
export OMP_NUM_THREADS=6
aprun –n 4 –N  4 –d 6 –j 1 program [options]
```

Using select and ncpus with PBS Pro

```
#!/bin/sh
#PBS -j oe
#PBS -l select=1:ncpus=24:mpiprocs=4
#PBS -l walltime=00:30:00
#PBS –q cpu_queue
cd $PBS_O_WORKDIR
export OMP_NUM_THREADS=6
aprun –n 4 –N 4 –d 6 –j 1 program [options]
```

Batch Job Submission

- qsub returns a Job ID of `574.sdb`
- Use `qstat -a` to show the status of job 574
- Default filename for `stdout` is `runjob.o574`
  - Script contains `-oe` option so only one output file
- For an interactive session use `-I`

```
% qsub runjob
574.sdb
% ls -l *574
-rw-------  1 rns hwpt 460 Nov 21 12:01 runjob.o574
%
```

```
% qsub -I -l select=1:ncpus=24,walltime=00:30:00
qsub: waiting for job 148535.sdb to start
qsub: job 148535.sdb ready
%
```

Batch Job Status

```
% > qstat -a

sdb:
                                                            Req'd  Req'd
Elap
Job ID     Username Queue    Jobname    SessID NDS TSK Memory Time  S Time
--------- -------- -------- ---------- ------ --- --- ------ ----- - -----
573.sdb   crayadm  workq    phi          7495  42 504    --  24:00 R 00:38
575.sdb   crayadm  workq    runit        7858   8   8    --  24:00 R 00:00



E - Job is exiting after having run (if stays in E too long use qdel –
Wforce)
H - Job is held
Q - job is queued, eligible to run or routed
R - job is running
T - job is being moved to new location
W - job is waiting for its execution time (-a option) to
    be reached
S - job is suspended
```

Running an application on the Cray XC
ALPS + aprun

- **ALPS : Application Level Placement Scheduler**
- **aprun is the ALPS application launcher**
  - It **must** be used to run application on the XC compute nodes: interactively or in a batch job
  - If `aprun` is not used, the application is launched on the MOM node (and will most likely fail).
  - `aprun` launches groups of Processing Elements (PEs) on the compute nodes
    (PE == (MPI RANK || Coarray Image || UPC Thread || ..))
  - `aprun` man page contains several useful examples
  - The 3 most important parameters to set are:

Description	Option
Total Number of PEs used by the application	-n
Number of PEs per compute node	-N
Number of threads per PE (More precise, the “stride” between 2 PEs on a node)	-d

ALPS Basics

Terminology

- **Node**
  - All resources managed by a single CNL instance
  - Once you have reserved a node, nobody else can use it
- **Processing Element (PE)**
  - PEs are instances of the executable
- **NUMA node**
  - On a Cray XC40 system a numa node is a multi-core socket (so two numa nodes/sockets per physical node)

Important Note: Applications must be run from the lustre file system (/mnt/lustre/) – compute nodes are unable to read the $HOME filesystem!

ALPS aprun Basic Job Control

- **Width (`aprun -n`)**
  - Number of PEs to launch (the default value is 1)
    - A PE is an instance of a binary copied to a node
  - Determines the number of MPI Ranks an application uses
- **Node List (`aprun -L`)**
  - A user-supplied list of candidate nodes to constrain placement
    - List must be specified in an increasing range
    - Could use “cnselect” output here
- **To run in Multiple Program Multiple Data (MPMD) mode, a colon (:) separates programs**
    - There must be a space on both sides of the colon
    - Example: aprun –n 960 –N24 ./atm : -n240 –N24 ./ocean : -n8 –N1 ./cpl

ALPS Multicore Control

PEs Per Node / PPN (aprun -N):

- Number of PEs per CNL instancei
- When specified:
  - Places specified number of PEs per node
  - A sufficient number of cores must exist on each node
- When unspecified:
  - Allows ALPS to pack PEs tightly
  - Behavior dependent upon application and system resources
- ALPS assigns the same number of PEs to all nodes, regardless of whether PPN is specified. (This is required for distributed memory (DM).)

ALPS Multicore Control (cont.)

Depth (`aprun -d`)

- **Specifies number of threads per PE**
  - The meaning of this option depends on the programming model. Specify this option when you use OpenMP code.
  - Compute nodes must have at least `depth` cores

- **ALPS reserves (width * depth) processor cores for the use of the application**

- **ALPS invokes width (`-n`) instances of the binary**
  - Application spawns (d - 1) threads per PE
  - Running two threads on a core is not a good idea

ALPS Multisocket Control

`aprun`	Description
`-S #`	Defines the number of PEs per NUMA node
`-sl #`	Defines the NUMA node to use: 0 or 1
`-sn #`	Defines the number of NUMA nodes to use: 1 or 2
`-ss`	Specifies strict memory containment per NUMA node
`-cc`	Defines a CPU list for binding, can also use keywords: *cpu* (default) binds a PE to a CPU in the NUMA node *numa_node* binds a PE to the CPUs within a NUMA node, any threads created by the PE are bound to the same NUMA node
`-cp`	Defines a file name to use for CPU binding
`-j #`	Specifies how many CPUs to use per compute unit for an ALPS job -j 0 - use all CPUs (or –j2) -j 1 - use only one (default on XC30)

ALPS Memory Control

Memory allocation is controlled with the -m option

- The option is in Megabytes per PE
- Don't use this option when first trying a new program – it is generally used only when necessary.

- -m size
- -m *size*h
  - Requests *size* of huge pages to be allocated to each processing element. All nodes use as much huge page memory as they are able to allocate and 4 KB pages afterward.
- -m *size*hs
  - Requires *size* of huge pages to be allocated to each processing element. If aprun cannot satisfy the request, it issues an error message and the application terminates.

Hyperthreads

- **Intel Hyper-Threading is a method of improving the throughput of a CPU by allowing two independent program threads to share the execution resources of one CPU**
  - When one thread stalls the processor can execute ready instructions from a second thread instead of sitting idle
  - Typically much less than 2x with two hyperthreads
- **With aprun, hyper-threading is controlled with the switch -j**
  - -j 1 = no hyper-threading (a node is treated to contain 24 cores)
  - -j 2 or –j 0 = hyper-threading enabled (a node is treated to contain 48 cores)
  - Default at SERC is –j1 (I think)
- **It may improve or degrade the performance of your application**
  - Try it, if it does not help, turn it off

Running applications on the Cray XC40: Some basic examples

Assume an XC40 with Haswell nodes (24 real cores per node)

- **Pure MPI application using 48 ranks (2 nodes)**

```
$ aprun –n 48 –j1 ./a.out
```

- **Same number of ranks, but spread to four nodes**

```
$ aprun -j1 -n 48 -N 12 -S 6 ./a.out
```

  - Can be used to increase the amount of memory available for each PE

- **To launch a hybrid MPI+OpenMP application**
  - 1024 ranks in total, 1 CPU per Compute Unit
  - Use 4 PEs per node and 6 threads per PE -> 256 nodes

```
$ export OMP_NUM_THREADS=6
$ aprun -n 1024 -N 4 -d $OMP_NUM_THREADS ./a.out
```

- **Launch the same hybrid application with 2 CPUs per CU**
  - 1024 ranks in total, 2 CPUs per Compute Unit

```
$ export OMP_NUM_THREADS=12
$ aprun -n 1024 -N 4 -d $OMP_NUM_THREADS -j 2 ./a.out
```

Matching PBS and aprun

It is important to reserve enough resources through PBS to satisfy the aprun command in your run script. If you don't, aprun will fail on launch with a message about lack of resources.

For instance, suppose the aprun command is

```
aprun -j1 -n 48 -N 12 -S 6 ./a.out
```

We want 4 nodes, with 12 ranks on each. Therefore, the PBS request should be

```
qsub –lselect=4:mpiprocs=12:ncpus=24
```

Selecting Nodes using cnselect

- Can select the nodes to submit jobs to via “cnselect”.
- Convenient MySQL interface to attributes table
- Returns a list of compute nodes based on user-specified criteria
- Must be run on a login node

- Select the CPU nodes (48 “cores” per node including hyperthreads):

```
> cnselect -e numcores.eq.48
      8-27,29-63,88-95,120-123,125-127,132-191,200-219,221-255,260-319,324-383,392-
      411,413-447,452-511,516-575,580-603,605-639,644-703,708-795,797-987,989-1179,1181-
      1371,1376-1415,1440-1447,1472-1535
```

- Select the GPU nodes:

```
> cnselect -e name.eq.Tesla_K40s
                    68-87,96-119
```

- Select the Phi nodes:

```
> cnselect -e name.eq.Xeon_Phi
        1416-1439,1448-1471
```

- There are many other attributes can be applied to cnselect (see man page). The output from cnselect can be passed to “aprun –L”

Watching a launched job on the Cray XC

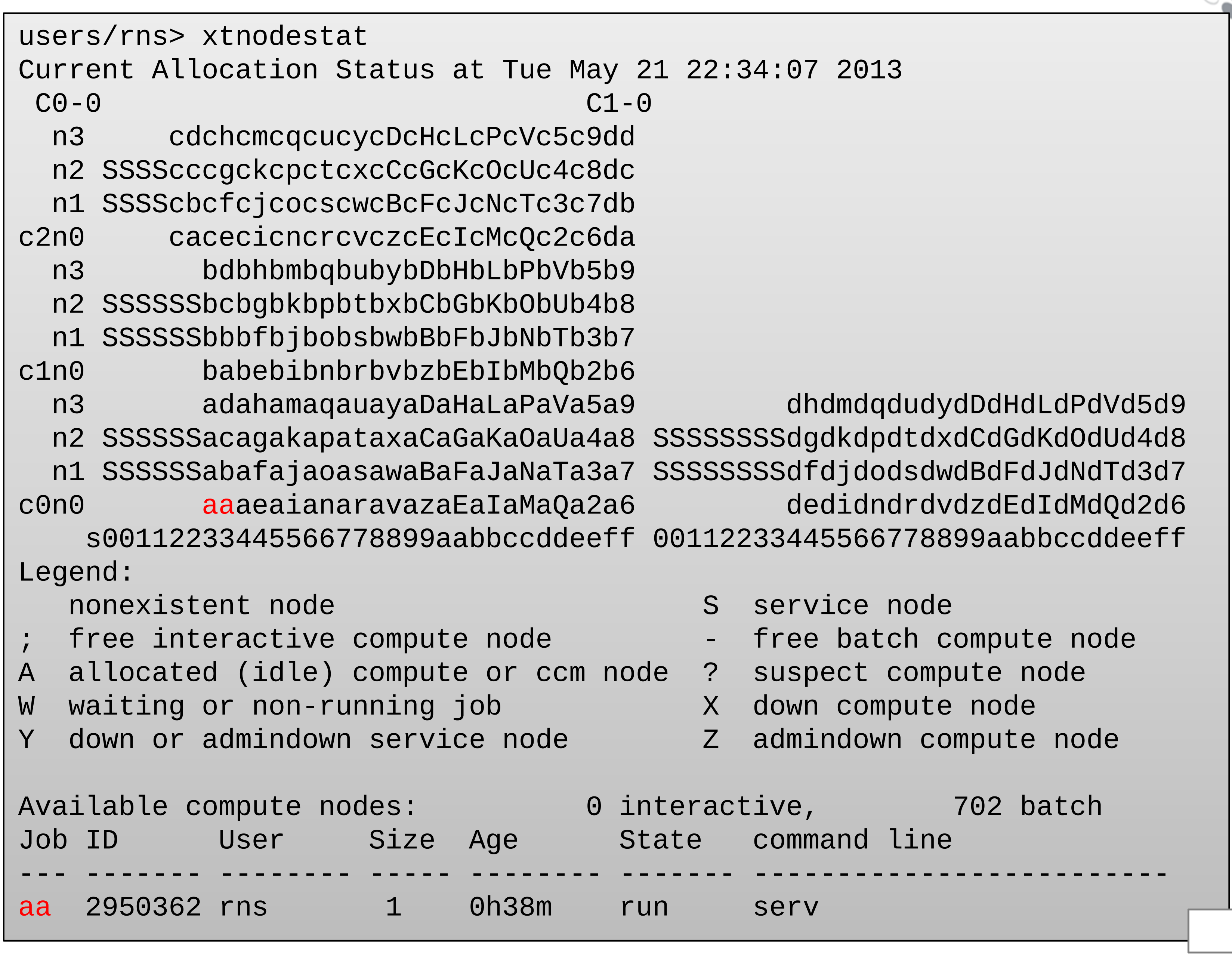
- **`xtnodestat`**
  - Shows how XC nodes are allocated and corresponding `aprun` commands

- **`apstat`**
  - Shows `aprun` processes status (apps, nodes, reservations etc)
  - `apstat` overview
  - `apstat -a[ apid ]` info about all the applications or a specific one
  - `apstat -n` info about the status of the nodes

- **Batch `qstat` command**
  - shows batch jobs

- **To kill a running job**
  - apkill [-signal] apid
  - -signal: specify name or integer of signal (default = SIGTERM = 15)

xtnodestat Example (not from SERC system)



```
users/rns> xtnodestat
Current Allocation Status at Tue May 21 22:34:07 2013
 C0-0                             C1-0
  n3     cdchcmcqcucycDcHcLcPcVc5c9dd
  n2 SSSScccgckcpctcxcCcGcKcOcUc4c8dc
  n1 SSSScbcfcjcocscwcBcFcJcNcTc3c7db
c2n0     cacecicncrcvczcEcIcMcQc2c6da
  n3       bdbhbmbqbubybDbHbLbPbVb5b9
  n2 SSSSSSbcbgbkbpbtbxbCbGbKbObUb4b8
  n1 SSSSSSbbbfbjbobsbwbBbFbJbNbTb3b7
c1n0       babebibnbrbvbzbEbIbMbQb2b6
  n3       adahamaqauayaDaHaLaPaVa5a9         dhdmdqdudydDdHdLdPdVd5d9
  n2 SSSSSSacagakapataxaCaGaKaOaUa4a8 SSSSSSSSdgdkdpdtdxdCdGdKdOdUd4d8
  n1 SSSSSSabafajaoasawaBaFaJaNaTa3a7 SSSSSSSSdfdjdodsdwdBdFdJdNdTd3d7
c0n0       aaaeaianaravazaEaIaMaQa2a6         dedidndrdvdzdEdIdMdQd2d6
    s00112233445566778899aabbccddeeff 00112233445566778899aabbccddeeff
Legend:
   nonexistent node                      S  service node
;  free interactive compute node         -  free batch compute node
A  allocated (idle) compute or ccm node  ?  suspect compute node
W  waiting or non-running job            X  down compute node
Y  down or admindown service node        Z  admindown compute node

Available compute nodes:          0 interactive,         702 batch
Job ID      User     Size  Age      State   command line
--- ------- -------- ----- -------- ------- -------------------------
aa  2950362 rns        1   0h38m    run     serv
```

Application Status (apstat)

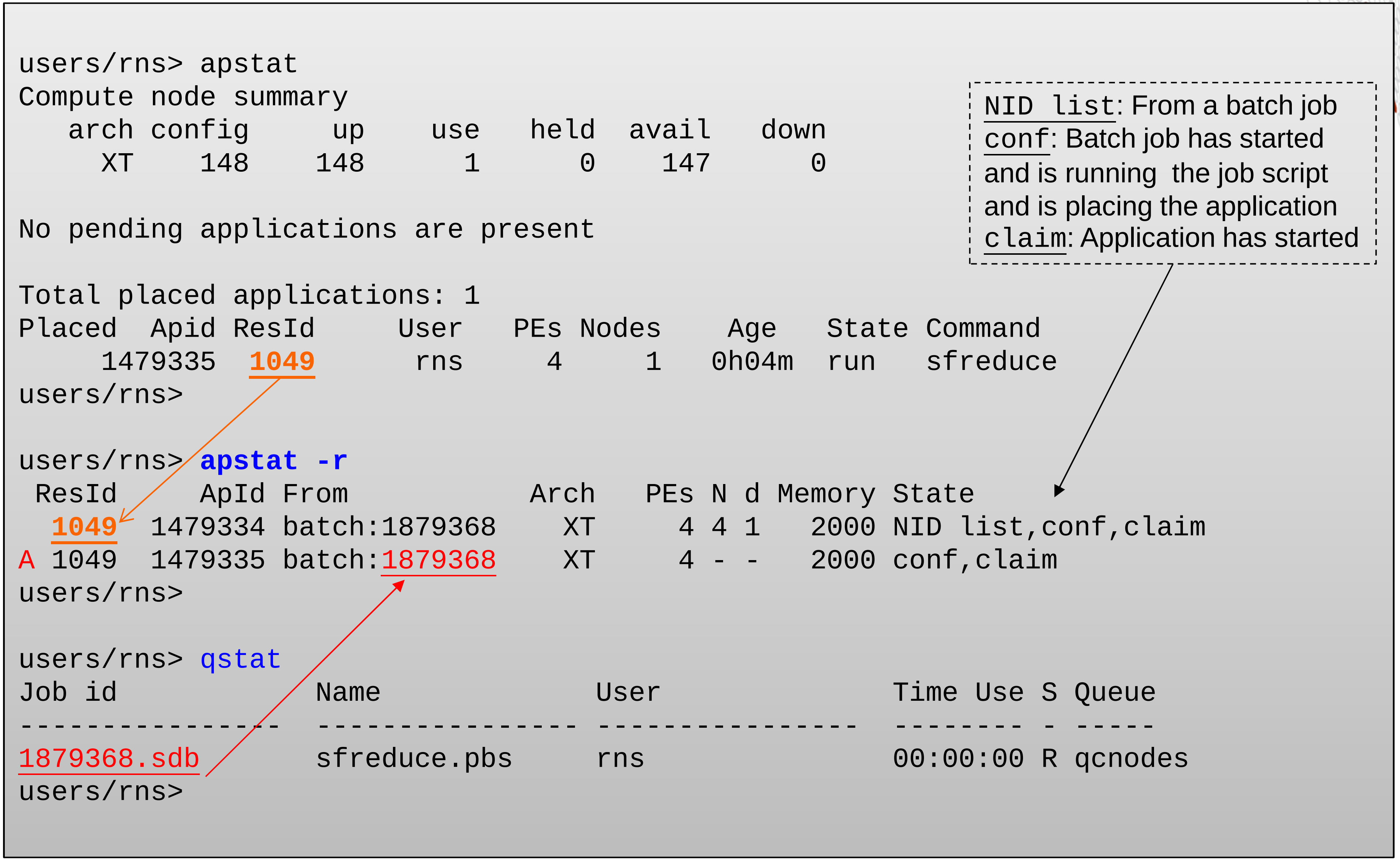
```
%  apstat
Compute node summary
    arch config     up   resv    use  avail   down
      XT   1464    767     95     95    672    697

No pending applications are present

Total placed applications: 8
  Apid  ResId     User PEs Nodes   Age State       Command
265385 272459  nisaadi  48     2 4h57m   run          vasp
265388 272460 sectanuj 408    17 4h57m   run l1_16K_4_16_w
265389 272461 sectanuj 408    17 4h57m   run l1_shared_16K
265394 272464  nisaadi  24     1 4h55m   run          vasp
265398 272465  nisaadi  72     3 4h55m   run          vasp
265400 272467 sectanuj 408    17 4h55m   run l1_shared_16K
265403 272468 sectanuj 410    18 4h55m   run  l1_16K_8_way
265408 272470 phyprtek 480    20 4h55m   run         pluto

No applications or reservations are being cleaned up
```

Reservation status (apstat –r)



```
users/rns> apstat
Compute node summary
   arch config      up     use    held  avail    down
     XT     148    148       1       0    147       0

No pending applications are present

Total placed applications: 1
Placed  Apid ResId     User   PEs Nodes    Age   State Command
     1479335  1049      rns     4     1   0h04m  run   sfreduce
users/rns>

users/rns> apstat -r
 ResId     ApId From             Arch   PEs N d Memory State
  1049  1479334 batch:1879368     XT     4 4 1   2000 NID list,conf,claim
A 1049  1479335 batch:1879368     XT     4 - -   2000 conf,claim
users/rns>

users/rns> qstat
Job id                 Name             User              Time Use S Queue
----------------  ---------------- ----------------  -------- - -----
1879368.sdb            sfreduce.pbs     rns               00:00:00 R qcnodes
users/rns>
```

`NID list`: From a batch job
`conf`: Batch job has started and is running the job script and is placing the application
`claim`: Application has started

Self-Guided XC40 Demo

- Guided set of commands and scripts to demonstrate use of various aspects of Cray XC40 system
- Simple to follow and provides a great introduction to working with PBS and ALPS to run jobs on CPU, GPU and Phi nodes
- Find it at /mnt/lustre/crayadm/BM/XC40_Demo (or wherever your friendly support guy moves it to)
- Few bugs to be shaken out – hopefully all are noted.

- Many thanks to Dave Strenski

- Please contact Tricia (pballe@cray.com) if anything doesn't work!

Cray Scientific Libraries

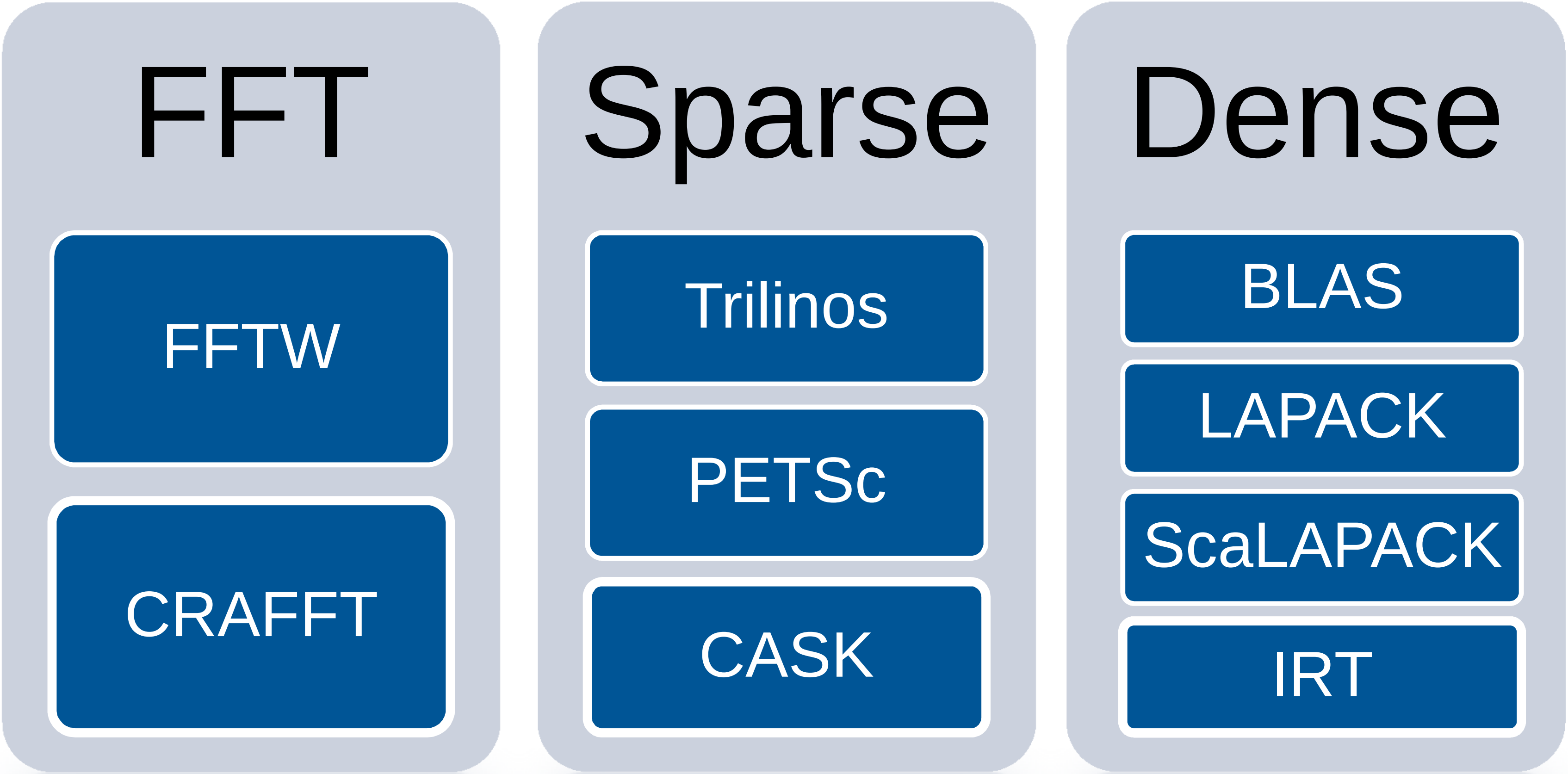
Overview

What are libraries for?

- Building blocks for writing scientific applications
- Historically – allowed the first forms of code re-use
- Later – became ways of running optimized code
- These days the complexity of the hardware is very high
- Cray PE insulates the user from that complexity
    - Cray module environment
    - CCE
    - Performance tools
    - Tuned MPI libraries (+PGAS)
    - Optimized Scientific libraries

Cray Scientific Libraries are designed to provide the maximum possible performance from Cray systems with minimum effort.

Scientific libraries on XC – functional view



FFT

- FFTW
- CRAFFT

Sparse

- Trilinos
- PETSc
- CASK

Dense

- BLAS
- LAPACK
- ScaLAPACK
- IRT

What makes Cray libraries special

1. **Node performance**
   - Highly tuned routines at the low-level (ex. BLAS)
2. **Network performance**
   - Optimized for network performance
   - Overlap between communication and computation
   - Use the best available low-level mechanism
   - Use adaptive parallel algorithms
3. **Highly adaptive software**
   - Use auto-tuning and adaptation to give the user the known best (or very good) codes at runtime
4. **Productivity features**
   - Simple interfaces into complex software

LibSci usage

- **LibSci**
  - The drivers should do it all for you - no need to explicitly link
  - For threads, set OMP_NUM_THREADS
    - Threading is used within LibSci
    - If you call within a parallel region, single thread used
- **FFTW**
  - `module load fftw` (there are also wisdom files available)
- **PETSc**
  - `module load petsc` (or module load petsc-complex)
  - Use as you would your normal PETSc build
- **Trilinos**
  - `module load trilinos`
- **Cray Adaptive Sparse Kernels (CASK)**
  - You get optimizations for free

Check you got the right library!

- Add options to the linker to make sure you have the correct library loaded.
- `-Wl` adds a command to the linker from the driver
- You can ask for the linker to tell you where an object was resolved from using the –y option.
  - E.g. `–Wl, -ydgemm_`

```
.//main.o: reference to dgemm_
/opt/xt-libsci/11.0.05.2/cray/73/mc12/lib/libsci_cray_mp.a(dgemm.o):
definition  of dgemm_
```

Note: do not explicitly link “-lsci”. This will not be found from libsci 11+ and means a single core library for 10.x.

Threading

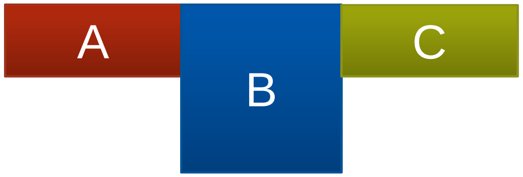
- **LibSci is compatible with OpenMP**
  - Control the number of threads to be used in your program using **OMP_NUM_THREADS**
  - e.g., in job script `export OMP_NUM_THREADS=16`
  - Then run with `aprun -n1 -d16`
- **What behavior you get from the library depends on your code**
  1. No threading in code
     - The BLAS call will use OMP_NUM_THREADS threads
  2. Threaded code, outside parallel regions
     - The BLAS call will use OMP_NUM_THREADS threads
  3. Threaded code, inside parallel regions
     - The BLAS call will use a single thread

Threaded LAPACK

- Threaded LAPACK works exactly the same as threaded BLAS
- Anywhere LAPACK uses BLAS, those BLAS can be threaded
- Some LAPACK routines are threaded at the higher level
- No special instructions

Performance Focus and Autotuning

- **Some components of the library are performance critical**
  - For example BLAS and specifically GEMM
- **It is a significant challenge to get best performance across a range of architectures and problem sizes and thread counts**



- **Cray has an autotuning framework to address this:**
  - It uses a general GEMM framework
  - Offline tuning runs are done for a wide range of problem sizes
  - CPU and GPU targets
  - Knowledge gained from offline runs incorporated into the runtime library.

Tuning requests

- **CrayBLAS is an auto-tuned library**
  - Generally, excellent performance is possible for all shapes and sizes
- **However, the adaptive CrayBLAS can be improved by tuning for exact sizes and shapes**
- **Send your specific tuning requirements to crayblas@cray.com**
  - Send the routine name and the list of calling sequences

ScaLAPACK and IRT

- **ScaLAPACK in LibSci is optimized for Aries interconnect**
  - New collective communication procedures are added
  - Default topologies are changed to use the new optimizations
  - Much better strong scaling
- **It also benefits from the optimizations in CrayBLAS**
- **Iterative Refinement Toolkit (IRT) can provide further improvements**
  - Uses mixed precision
  - For some targets (CPU vector instructions and GPUs) single-precision can be much faster
  - Used for serial and parallel LU, Cholesky and QR
  - Either set IRT_USE_SOLVERS to 1
    or use the advanced API.

Third-party libraries

- **The modules cray-trilinos and cray-petsc / cray-petsc-complex contain the popular Trilinos and PETSc packages**
  - These will automatically employ the Cray Adaptive Sparse Kernels
- **The module cray-tpsl contains ready builds of some other quite common libraries and solvers:**
  - MUMPS, ParMetis, SuperLU, SuperLU_DIST, Hypre, Scotch, Sundials
  - These are for your convenience (i.e. no need to build the library yourself) but do not feature Cray-specific modifications

Intel MKL

- **The Intel Math Kernel Libraries (MKL) is an alternative to LibSci**
  - Features tuned performance for Intel CPUs as well
- **Linking quite complicated, but the Intel MKL Link Line Advisor can tell you what to add to your link line**
  - http://software.intel.com/sites/products/mkl/
- **Using MKL together with the Intel compilers (PrgEnv-intel) is usually straightforward**

Summary

- **Do not re-invent the wheel but use scientific libraries wherever you can!**
- **All the most widely used library families and frameworks readily available as XC optimized versions**
  - And if the cornerstone library of your application is still missing, let us know about it!
- **Make sure you use the optimized version provided by the system instead of a reference implementation**

- **... and give us feedback!**