高性能计算程序设计基础

任课教师：黄聃（Huang，Dan）

Based on the slides of Steve Leak

NERSC User Engagement Group

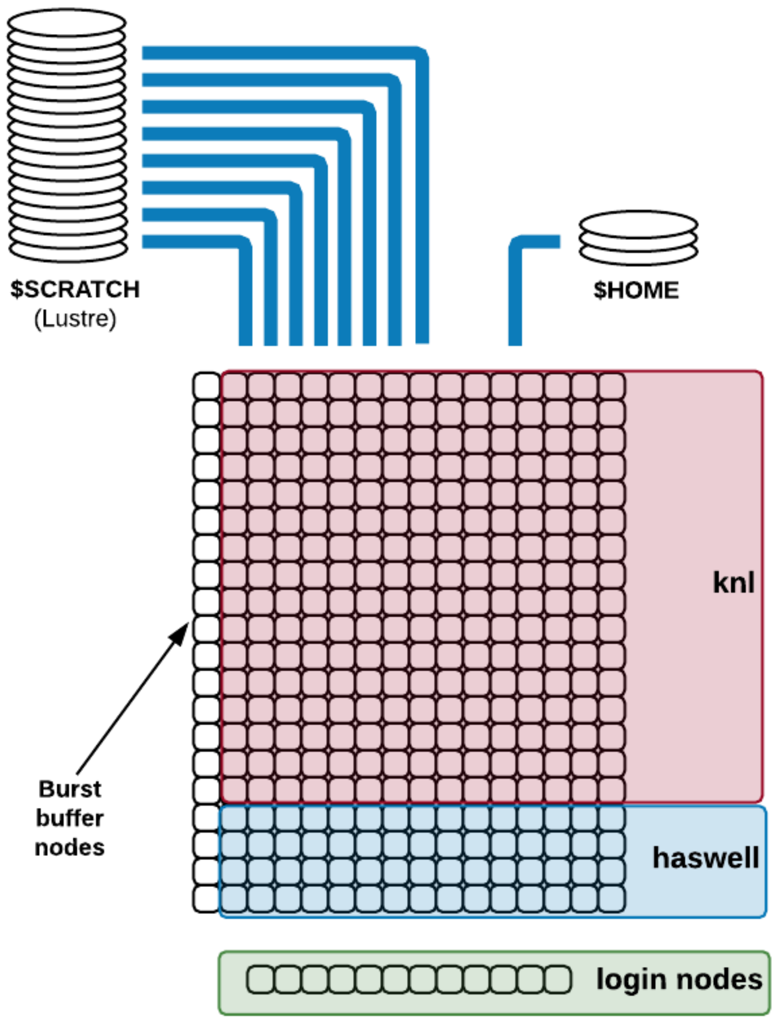
NERSC Cori

- 34 double-width cabinets
- 9,688 KNL + 2,388 Haswell nodes on Aries High-Speed Network
- 658,784 KNL cores + 76,416 Haswell cores
- Top500 #6 (June 2017)

Agenda

- Cori overview, logging in
- Run a simple job
- Building and running applications on Cori
  - Serial
  - Parallel (MPI)
  - Multithreaded (OpenMP)
- What affects performance?
  - Bottlenecks
  - Task placement and affinity
- Preparing for performance analysis

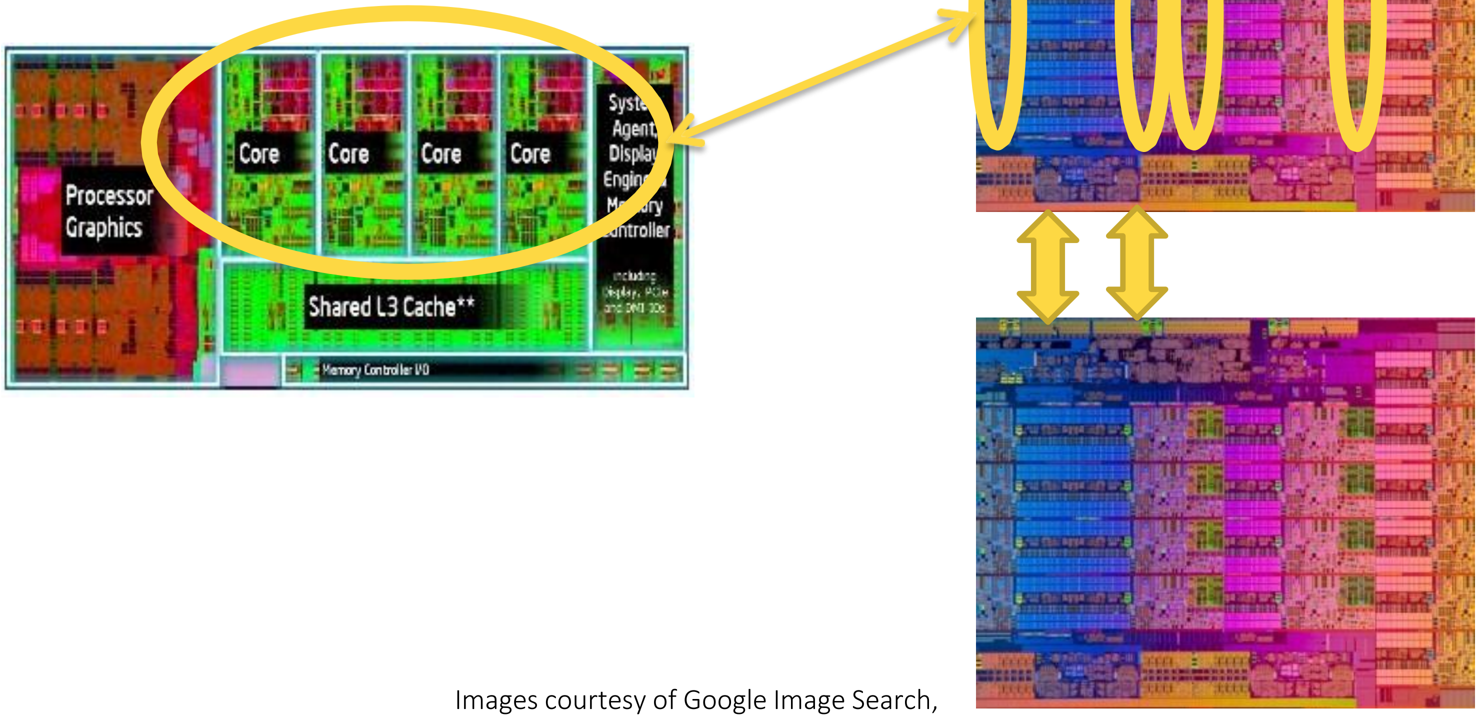
Cori Overview



- Login and compute nodes are distinct
- Large, fast, parallel
  - $SCRATCH filesystem for running jobs
- Smaller $HOME, configured for building code
- Burst buffer filesystem integrated, on high-speed network

What's so special about it?

- Your quad-core desktop CPU looks something like this:
- Compared to a Cori Haswell node:

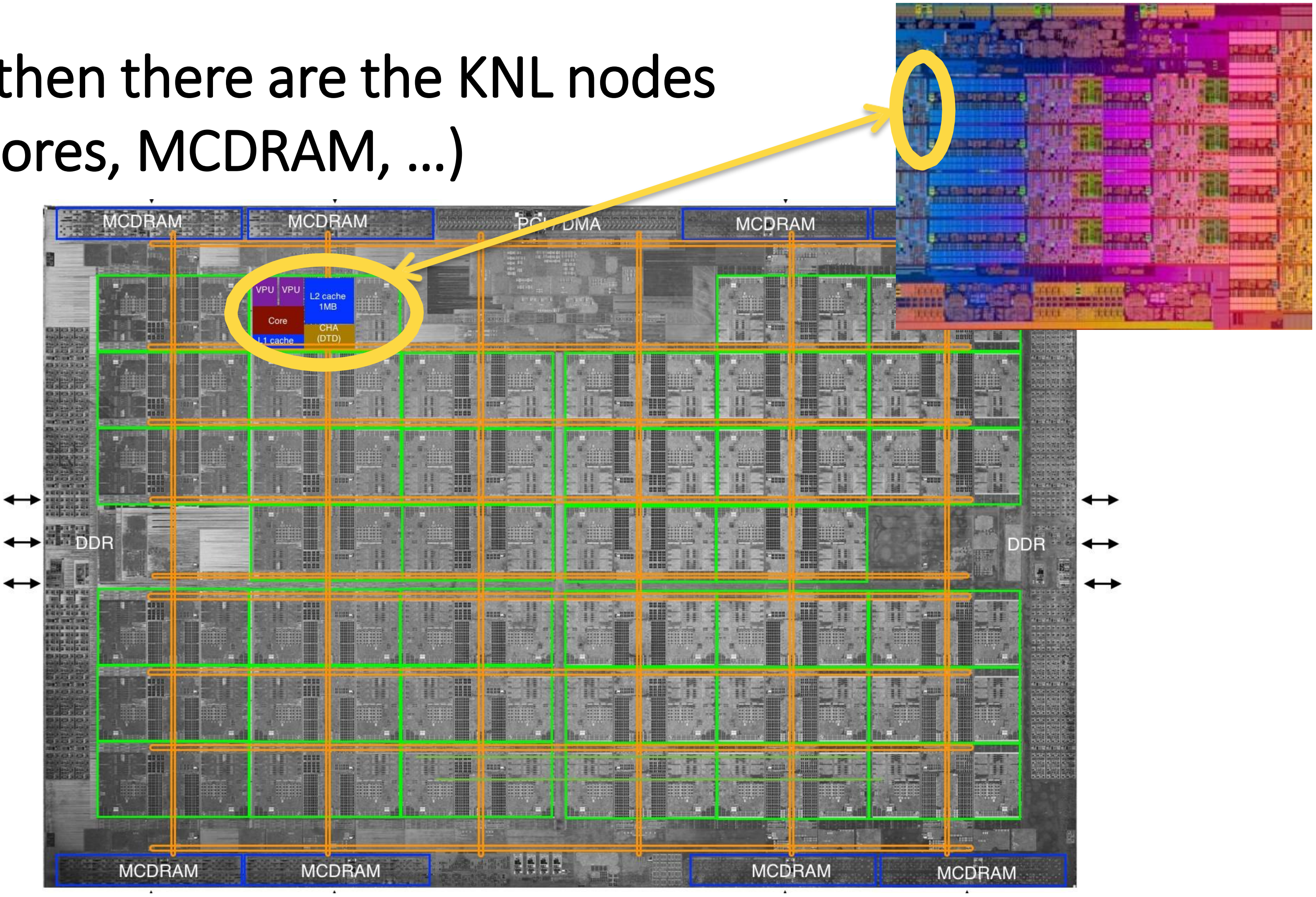

Processor Graphics
Core
Core
Core
Core
System Agent, Display Engine, Memory Controller
including Display, PCIe and DMI I/Os
Shared L3 Cache**
Memory Controller I/O

16 cores
x 2 sockets,
128GB RAM

Images courtesy of Google Image Search, Intel, EnterpriseTech.com, NextPlaQorm.com

What's so special about it?

- And then there are the KNL nodes (68 cores, MCDRAM, ...)



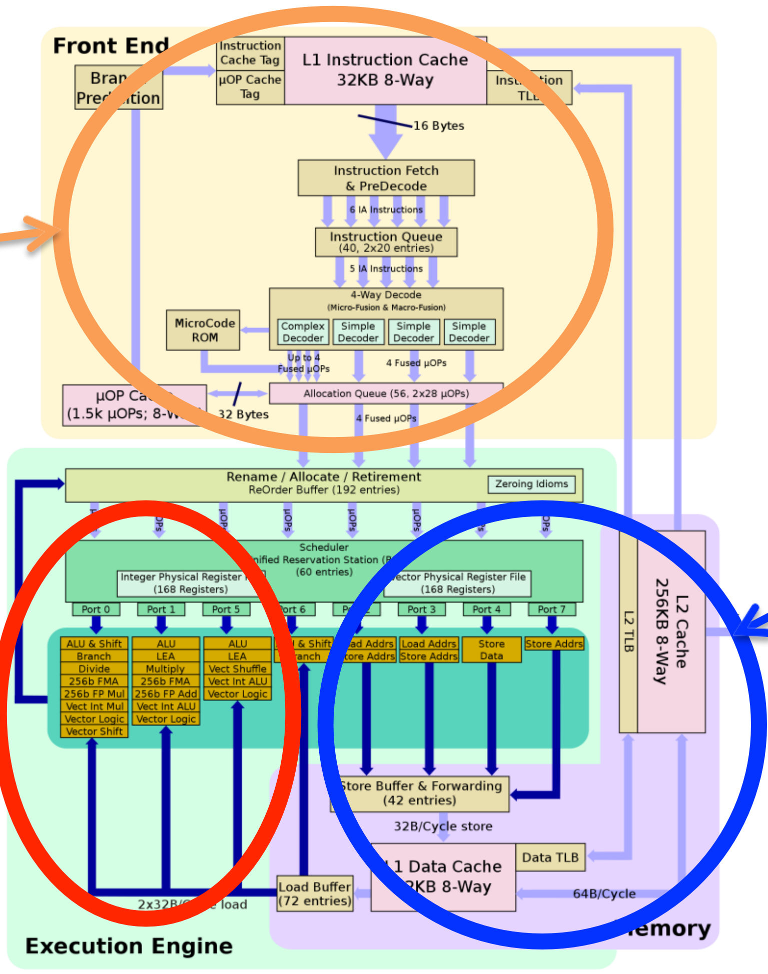
What’s so special about it?

- But high-end CPUs don’t make a supercomputer
  - High speed interconnects between them
  - Lightweight compute node OS
  - Very large (28,000 TB) fast parallel filesystem
- …and a different usage model
  - Subset of nodes dedicated to a single task, run via batch system (no interactive GUI / desktop)

What's special about KNL?

- Different choice of compromise between die space allocated to different parts of the CPU

Instruction fetch-and-decode, hyperthreading, branching (important for eg compiling, GUI applications)



Arithmetic, vector and floating point units – the actual FLOPS

Memory access – keeps execution engine busy

What's special about KNL?

- Xeon (eg Haswell)
- Xeon Phi (KNL)

Large and fast - power hungry, hot

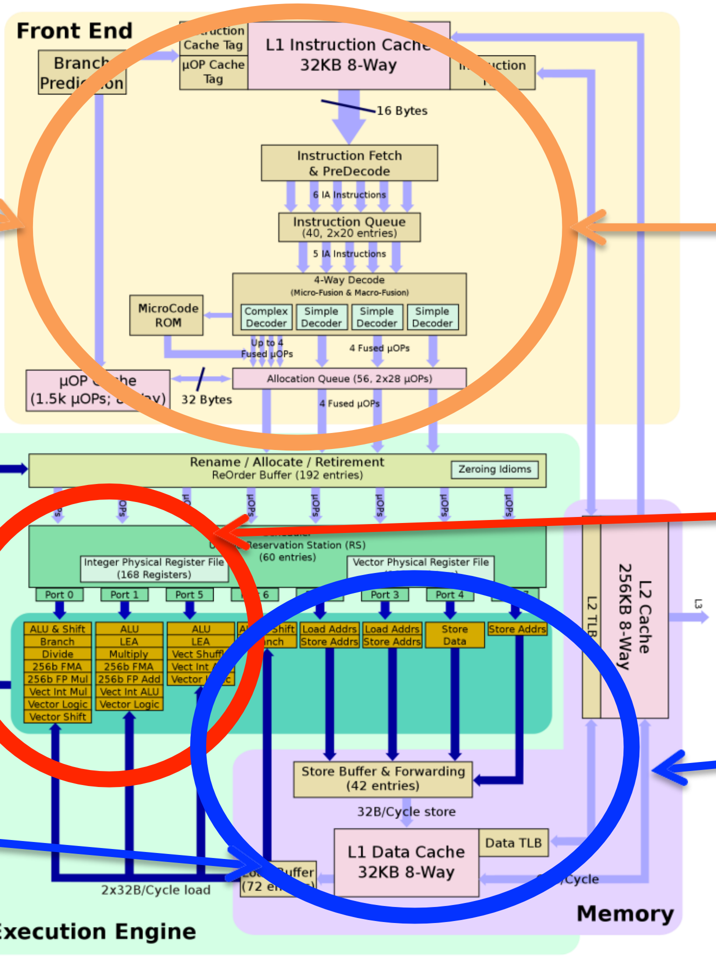
Small and slow, cool, efficient

Big wide vectors

Smaller space, shorter vectors

Large L2 cache, no L3 (but MCDRAM)

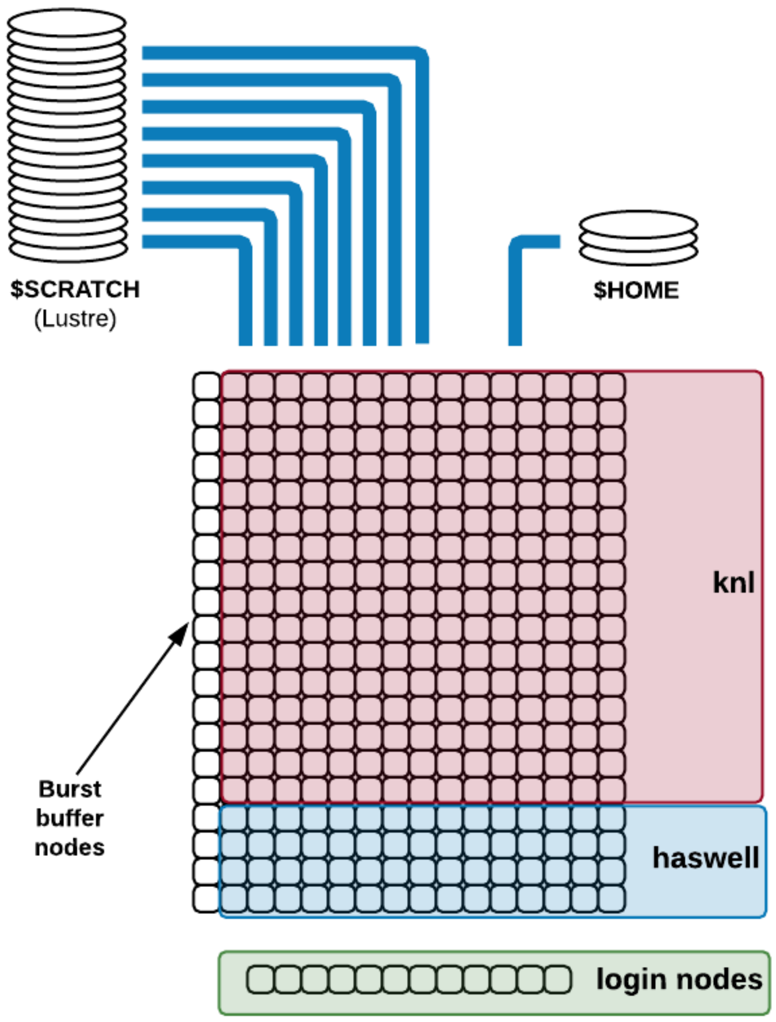
large L3 cache



Why?

- Exascale challenges – power and heat
  - CPU frequency plateaued ~15 years ago
  - Transistor density, feature size reaching fundamental limits
  - Power consumption and heat dissipation are now the key constraints for supercomputing
  - … we can't get there from here!
- KNL emphasizes vectorization and parallelism at lower power
  - Targets scientific computing

Cori Overview



- Login and compute nodes are distinct
- Large, fast, parallel
- $SCRATCH filesystem for running jobs
- Smaller $HOME, configured for building code
- Burst buffer filesystem integrated, on high-speed network

Connecting to Cori

- If you have a UNIX-like computer, you can directly contact NERSC with your built-in SSH client
  1. Open a new terminal
  2. % ssh –Y -l <training_acct_username> cori.nersc.gov

     -Y Enables trusted X11 forwarding.

- Many SSH clients exist for Windows
  – A very popular one is **putty**
    - hZp://www.putty.org/
  – Advanced users might prefer to use SSH directly within **mintty** (from Cygwin distribution)

X-forwarding

- Allows you to access GUI programs remotely

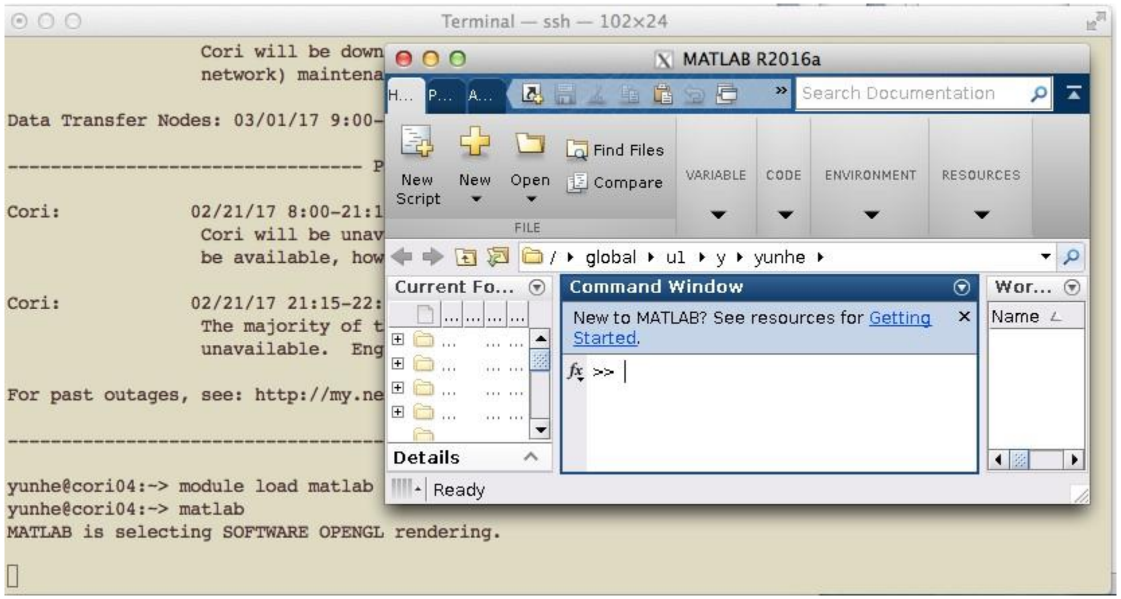
Example:

localhost% ssh –Y -l elvis cori.nersc.gov

…

e/elvis> module load matlab e/elvis> matlab

<MATLAB starts up>



```
Terminal — ssh — 102x24
                     Cori will be down
                     network) maintena

Data Transfer Nodes: 03/01/17 9:00-

---------------------------------- P

Cori:              02/21/17 8:00-21:1
                     Cori will be unav
                     be available, how

Cori:              02/21/17 21:15-22:
                     The majority of t
                     unavailable.  Eng

For past outages, see: http://my.ne

-----------------------------------

yunhe@cori04:~> module load matlab
yunhe@cori04:~> matlab
MATLAB is selecting SOFTWARE OPENGL rendering.
```

Example Session

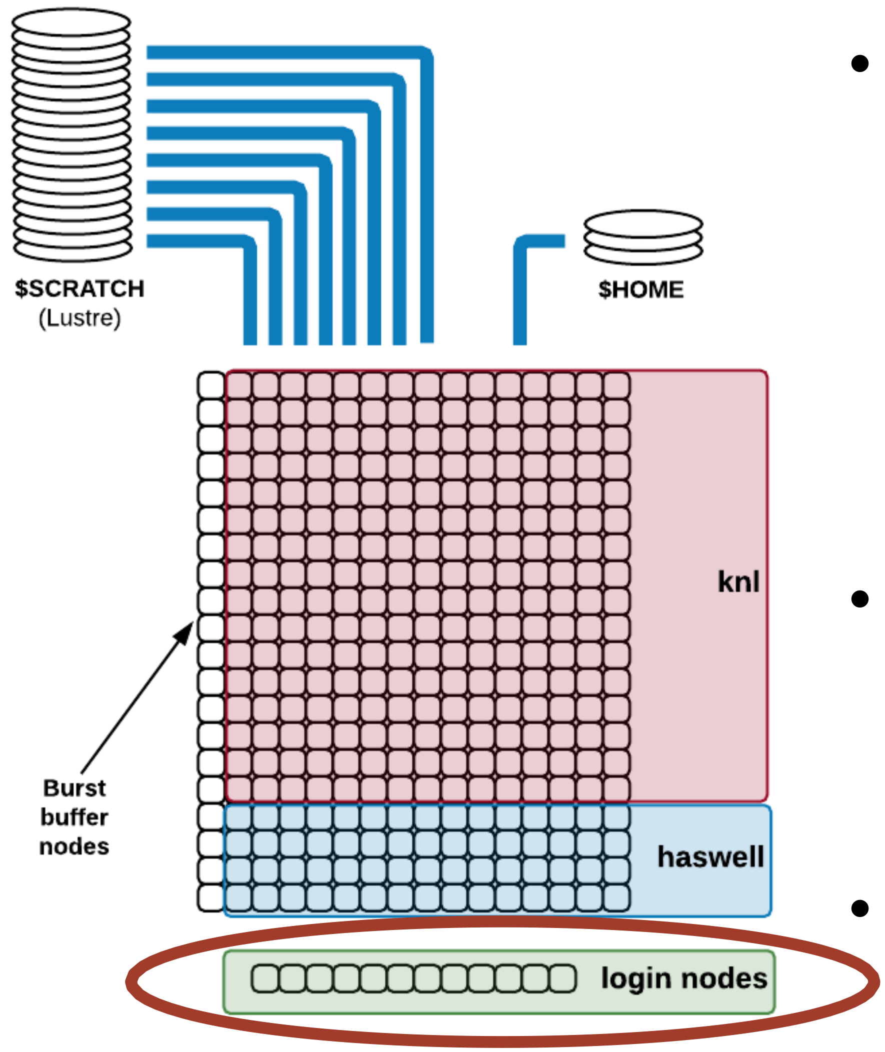
```
localhost:~elvis> ssh –Y -l <training_account_name> cori.nersc.gov

**************************************************************
*                                                            *
*                    NOTICE TO USERS                         *
*                    ---------------                         *
*                                                            *
*   Lawrence Berkeley National Laboratory operates this      *
*   computer system under contract to the U.S. Department of *
*   Energy.      This computer system is the property of the United *
*   States Government and is for authorized use only.   *Users *
*   (authorized or unauthorized) have no explicit or implicit *
*   expectation of privacy.*                                 *
*                                                            *
*   Any or all uses of this system and all files on this system  may be intercepted, *
*   monitored, recorded, copied, audited,  inspected, and disclosed to site, Department of *
*   Energy, and  law enforcement personnel, as well as authorized officials  of other *
*   agencies, both domestic and foreign.          *By using  this *
*   system, the user consents to such interception,  monitoring, recording, copying, *
*   auditing, inspection, and  disclosure at the discretion of authorized site or  Department *
*   of Energy personnel.*                                    *
*                                                            *
*                                                            *
*                                                            *
*   *Unauthorized or improper use of this system may result in  administrative disciplinary *
*   action and civil and criminal  penalties.  _By continuing to use this system you indicate *
*   your awareness of and consent to these terms and conditions  of use.     LOG OFF *
*   IMMEDIATELY if you do not agree to the  conditions stated in this warning._* *
*                                                            *
*                                                            *

**************************************************************

Password:      <enter your training account password here>
```

Prompt on local system

Notification of acceptable use.

Password prompt

After logging in…



- On a login node
  - cori01, cori02, …
  - **Shared by many users**
  - Not necessarily the same one each time!
  - But same access to filesystems
- **No direct access to compute nodes**
  - Only via batch system (**salloc**, **sbatch**)
- **Haswell (Xeon) architecture**

Agenda

- Cori overview, logging in
- Run a simple job
- Building and running applications on Cori
  - Serial
  - Parallel (MPI)
  - Multithreaded (OpenMP)
- What affects performance?
  - Bottlenecks
  - Task placement and affinity
- Preparing for performance analysis

Running jobs – key points

- HPC work is via batch system
  - Dedicated subset of compute resources
  - Login nodes are shared resource for building code, editing scripts, etc. Use batch jobs for real work
- Key commands:
  - sbatch / salloc - submit a job
  - srun - start an (optionally MPI) application within a job
  - sqs - check the queue for my job status
- For today, we have a reservation
  - #SBATCH –reservation=csgrrain

https://docs.nersc.gov/jobs/

Running jobs – key points

- HPC work is via batch system
  - Dedicated subset of compute resources
  - Login nodes are shared resource for building code, editing scripts, etc. Use batch jobs for real work
- Key commands:
  - sbatch / salloc - submit a job
  - srun - start an (optionally MPI) application within a job
  - sqs - check the queue for my job status
- For today, we have a reservation
  - #SBATCH –reservation=csgrrain

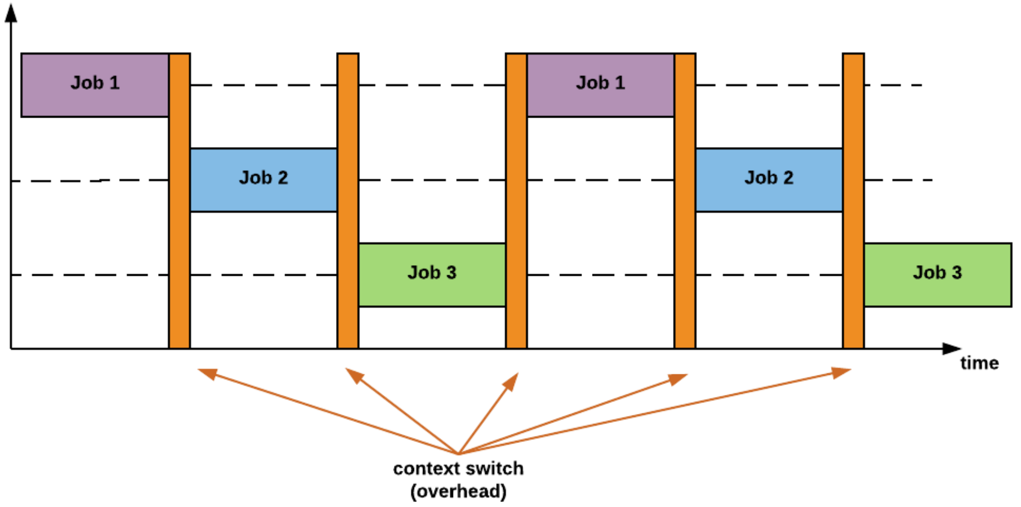
All of this is on the web!

https://docs.nersc.gov/jobs/

How jobs work

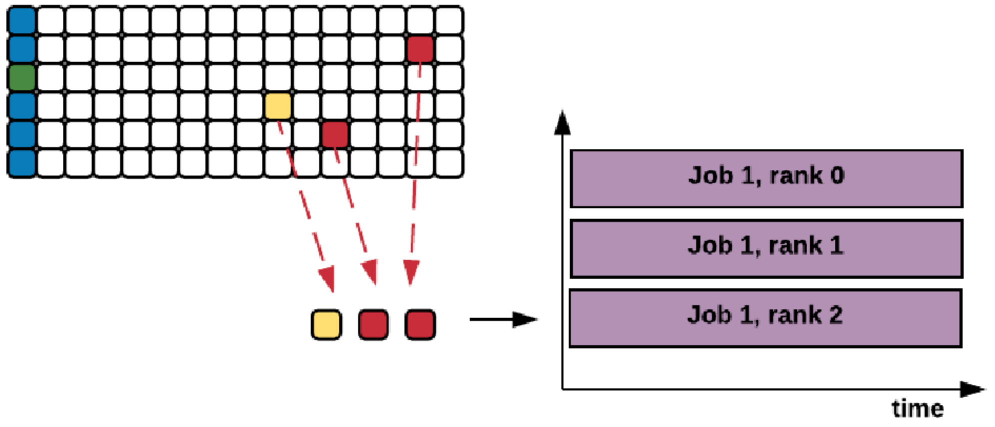
Desktop / login node

- Timeslicing
  - core shared by multiple tasks
  - Works when the computer is mostly waiting for you



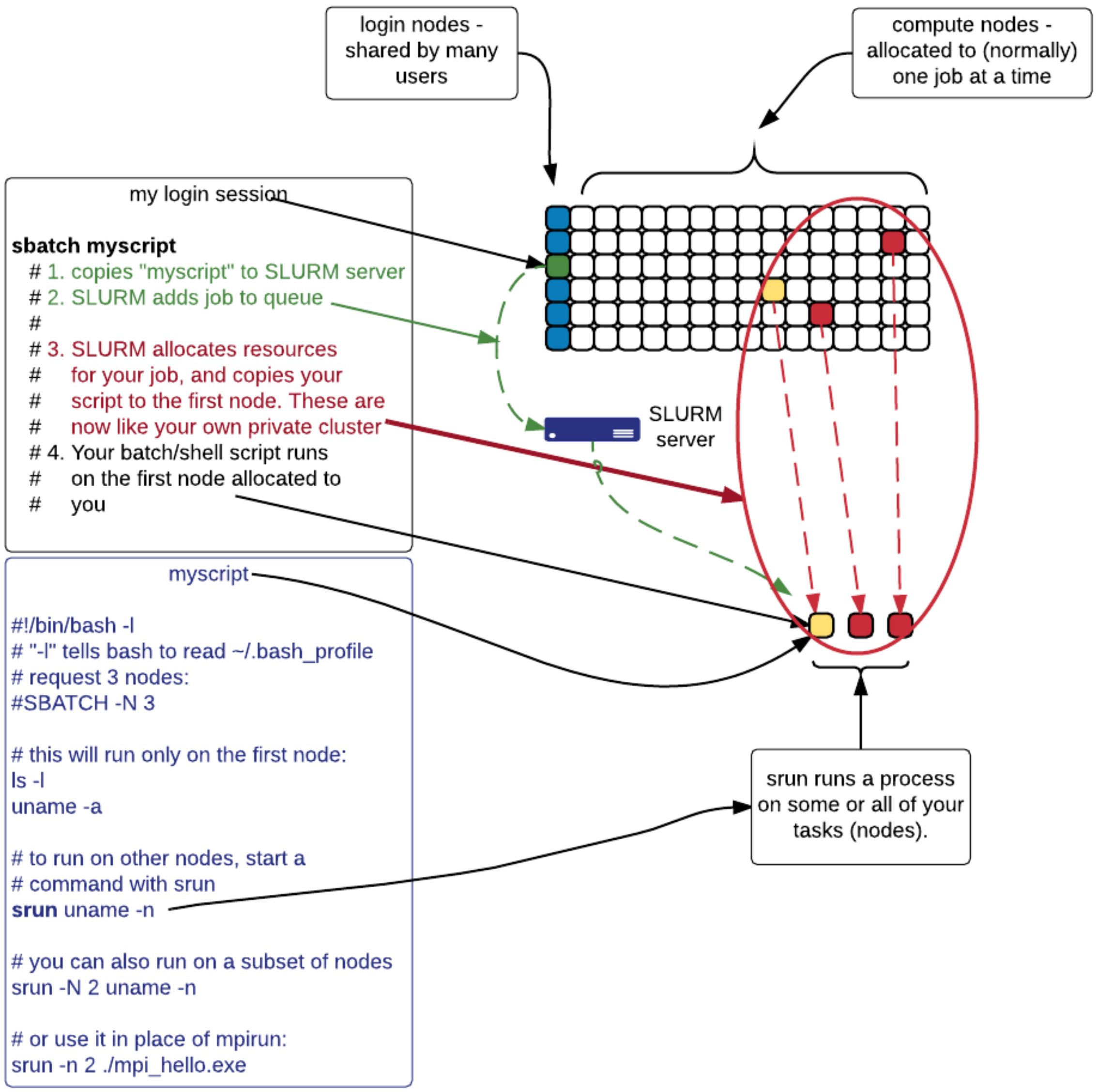
HPC

- You are waiting for the computer
- Subset of pooled resources dedicated to one job



How jobs work

- Start on login node
  - shared by many users, not for computational work
- Access compute nodes with sbatch or salloc
- Batch script
  - Copied to queue
  - Has directives for SLURM, and shell commands to perform on first compute node
- Access your other allocated nodes with srun
- stdout, stderr saved to file
  - (when running in batch mode)



login nodes - shared by many users
compute nodes - allocated to (normally) one job at a time
SLURM server
srun runs a process on some or all of your tasks (nodes).
my login session
myscript

my login session

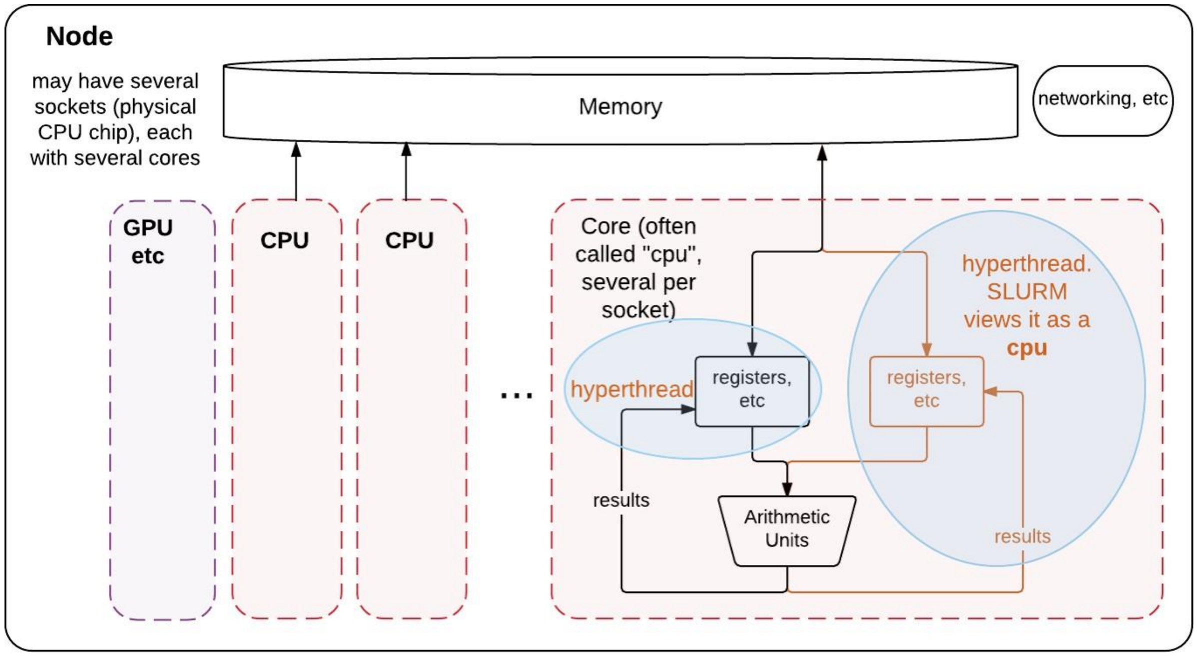
```
sbatch myscript
  # 1. copies "myscript" to SLURM server
  # 2. SLURM adds job to queue
  #
  # 3. SLURM allocates resources
  #    for your job, and copies your
  #    script to the first node. These are
  #    now like your own private cluster
  # 4. Your batch/shell script runs
  #    on the first node allocated to
  #    you
```

myscript

```
#!/bin/bash -l
# "-l" tells bash to read ~/.bash_profile
# request 3 nodes:
#SBATCH -N 3

# this will run only on the first node:
ls -l
uname -a

# to run on other nodes, start a
# command with srun
srun uname -n

# you can also run on a subset of nodes
srun -N 2 uname -n

# or use it in place of mpirun:
srun -n 2 ./mpi_hello.exe
```

Nodes, cores, CPUs, threads, tasks - some definitions

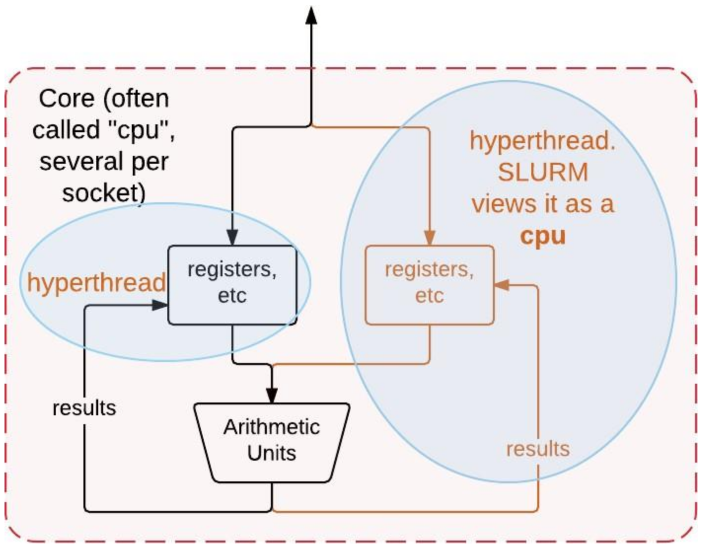
- *Node* is the basic unit of allocation at NERSC
  - Think "one host" or "one server"
  - Single memory space, multiple CPU cores (24 or 32 or 68 ...
    - And a core might support hyperthreading

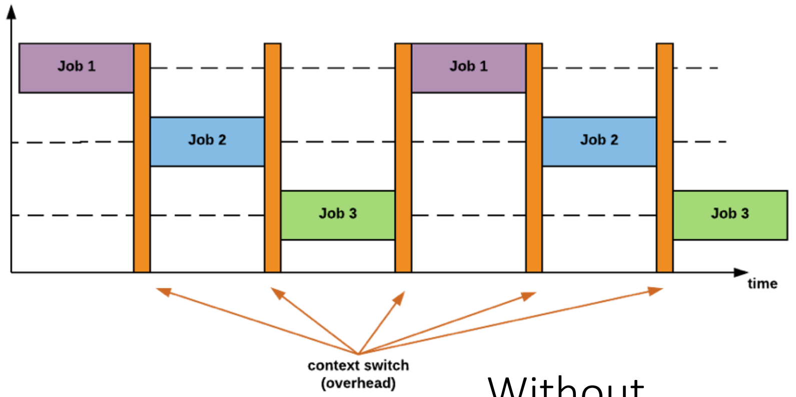

Node
may have several sockets (physical CPU chip), each with several cores
Memory
networking, etc
GPU etc
CPU
CPU
...
Core (often called "cpu", several per socket)
hyperthread
registers, etc
results
Arithmetic Units
hyperthread. SLURM views it as a cpu
registers, etc
results

Nodes, cores, CPUs, threads, tasks - some definitions

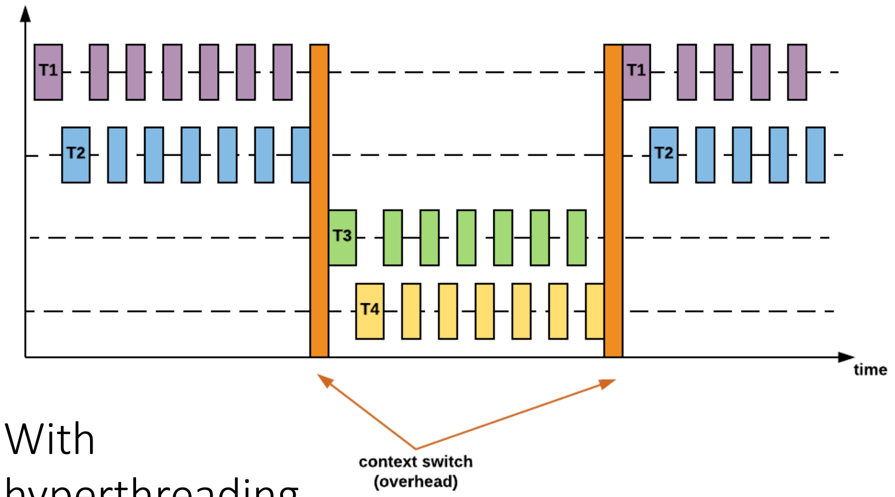
Hyperthreading

- Fast timeslicing
  - Good when arithmetic units frequently wait on memory
- Core holds state of 2 (4 on KNL) processes, they share arithmetic units
- **SLURM views each hyperthread as a *CPU***
- But most HPC jobs perform best when not sharing a core!
- Usually best to reserve 2 (or 4) CPUs / core


Core (often called "cpu", several per socket)
hyperthread
registers, etc
hyperthread. SLURM views it as a **cpu**
registers, etc
results
Arithmetic Units
results



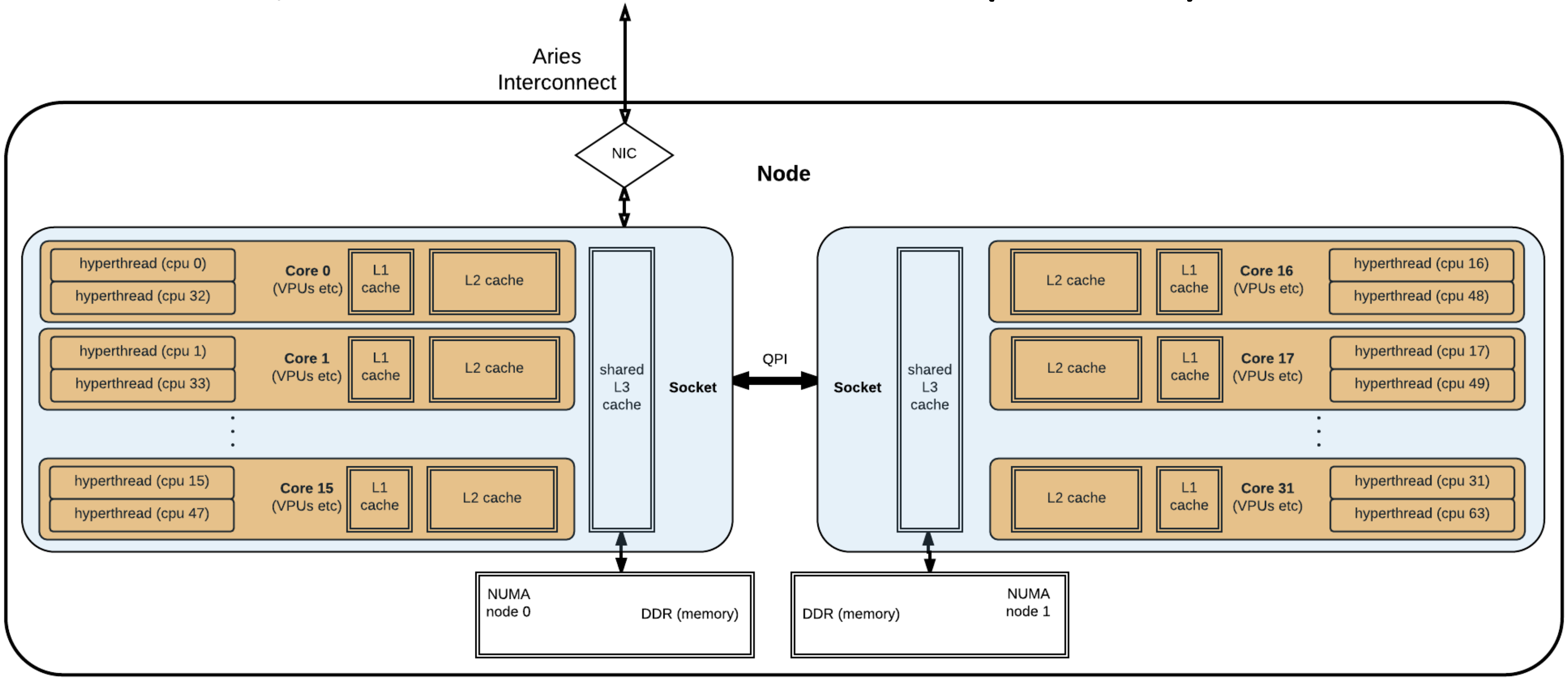
Without hyperthreading



With hyperthreading

Slurm tasks

- First, a block diagram of how hyperthreads, cores, cache, and sockets relate within a (haswell) node

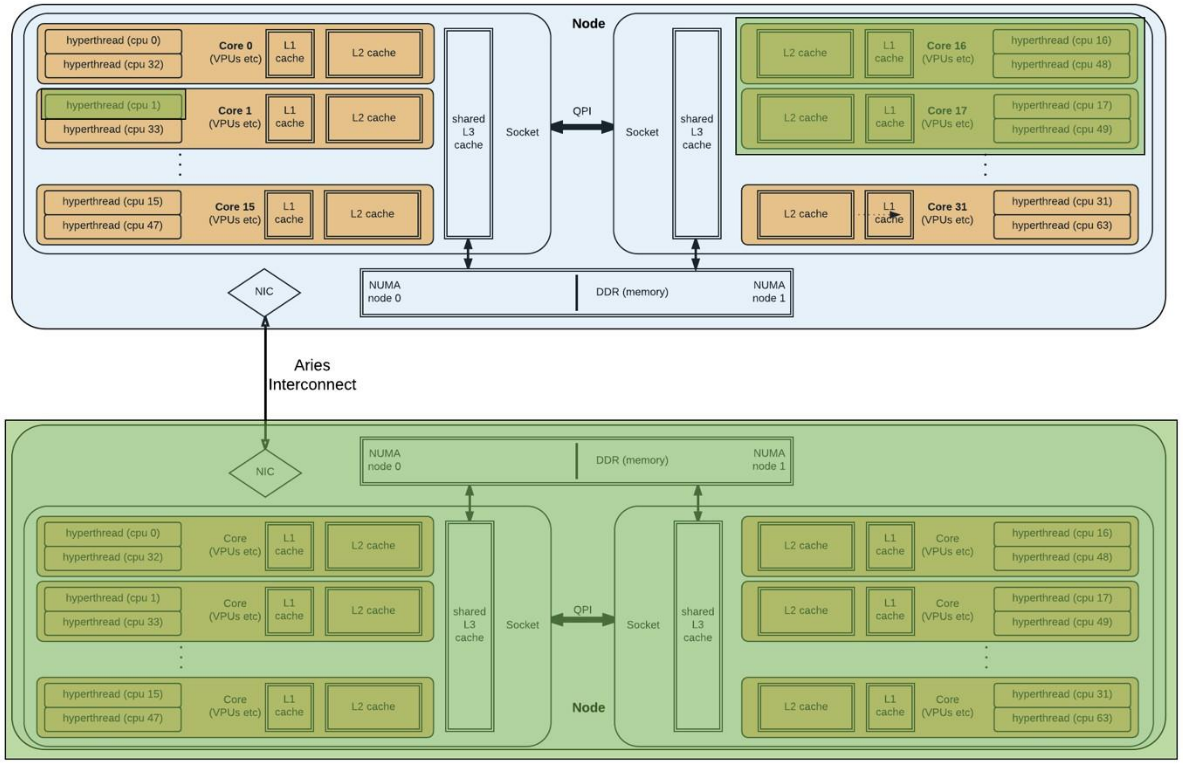


Slurm tasks

- A SLURM *task* is a reservation of CPUs and memory, up to one full node
  - A *job* has many tasks
  - 1 task typically corresponds to 1 MPI rank

srun -n <*ntasks*> ..

- Eg: 3 possible tasks on 2 nodes



So what must I request for my job?

What the batch system needs to know:

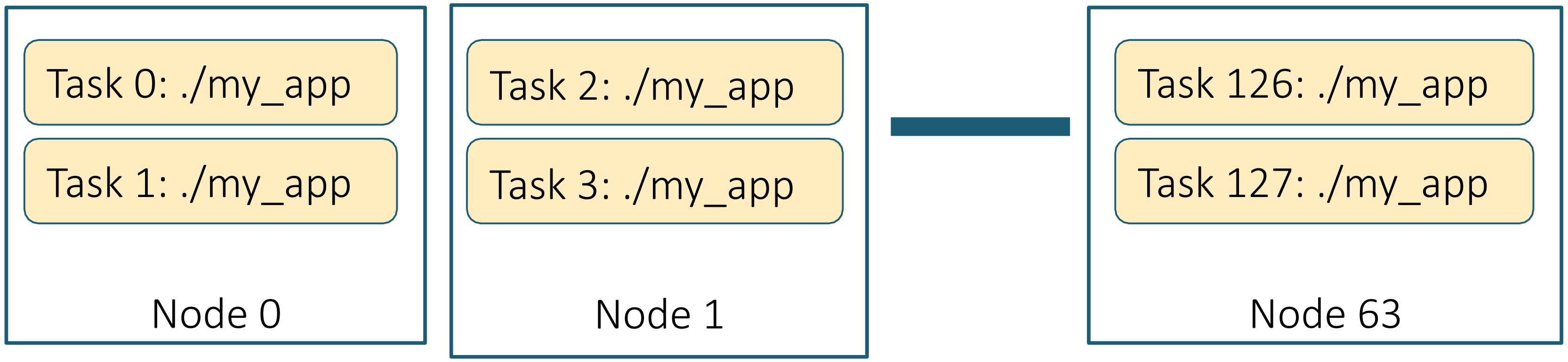
- How many nodes (or CPUs or tasks) does this job need?
- For how long does it need them?
  - Wallclock time limit

NERSC-specific extras:

- What type of CPU? (-C …)
  - KNL or Xeon (haswell/ivybridge)?
- Which filesystems will this job use? (-L …)
  - Usually SCRATCH

Requesting nodes or tasks

```
#SBATCH -N 64             # request 64 nodes
srun -N 32 ./my_app       # start ./my_app on 32 of them  # (default: 1
                          per node)
srun -n 128 ./my_app      # start 128 instances of ./my_app,
                          # across my 64 nodes (default is  # to evenly
                          distribute them in
                          # block fashion)
```



One MPI rank generally corresponds to one SLURM Task

Requesting time

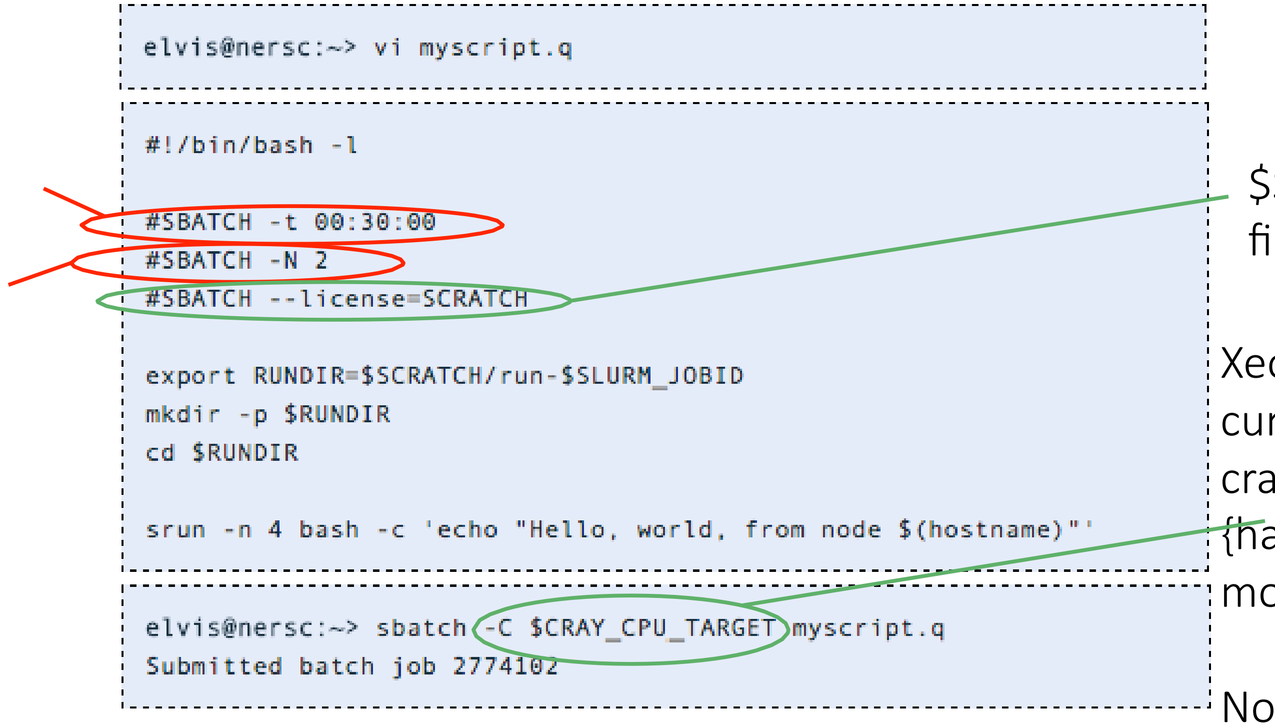
```
#SBATCH -t 30           # 30 minutes
#SBATCH -t 30:00        # 30 minutes
#SBATCH -t 1:00:00      # 1 hour
#SBATCH -t 1-0          # 1 day
#SBATCH -t 1-12         # 1.5 days
```

- Wallclock time, ie real elapsed time
- Aper this much time, SLURM can kill this job

Hands-on exercise: My first job

A SLURM job script has two sections:

1. Directives telling SLURM what you would like it to do with this job
2. The script itself - shell commands to run on the first compute node



```
elvis@nersc:~> vi myscript.q
```

```
#!/bin/bash -l

#SBATCH -t 00:30:00
#SBATCH -N 2
#SBATCH --license=SCRATCH

export RUNDIR=$SCRATCH/run-$SLURM_JOBID
mkdir -p $RUNDIR
cd $RUNDIR

srun -n 4 bash -c 'echo "Hello, world, from node $(hostname)"'
```

```
elvis@nersc:~> sbatch -C $CRAY_CPU_TARGET myscript.q
Submitted batch job 2774102
```

For how long?

How many nodes?

$SCRATCH filesystem

Xeon nodes on current cluster (set by craype-{haswell,ivybridge} module)

Note: cannot use env vars in directives - but directives have

Hands-on exercise: My first job

A SLURM job script has two sections:

1. Directives telling SLURM what you would like it to do with this job
2. The script itself - shell commands to run on the first compute node

```
elvis@nersc:~> vi myscript.q
```

Make starting environment like my login environment

Run from $SCRATCH

Start 4 tasks across my nodes

```
#!/bin/bash -l

#SBATCH -t 00:30:00
#SBATCH -N 2
#SBATCH --license=SCRATCH

export RUNDIR=$SCRATCH/run-$SLURM_JOBID
mkdir -p $RUNDIR
cd $RUNDIR

srun -n 4 bash -c 'echo "Hello, world, from node $(hostname)"'
```

```
elvis@nersc:~> sbatch -C $CRAY_CPU_TARGET myscript.q
Submitted batch job 2774102
```

“sbatch” submits a job script

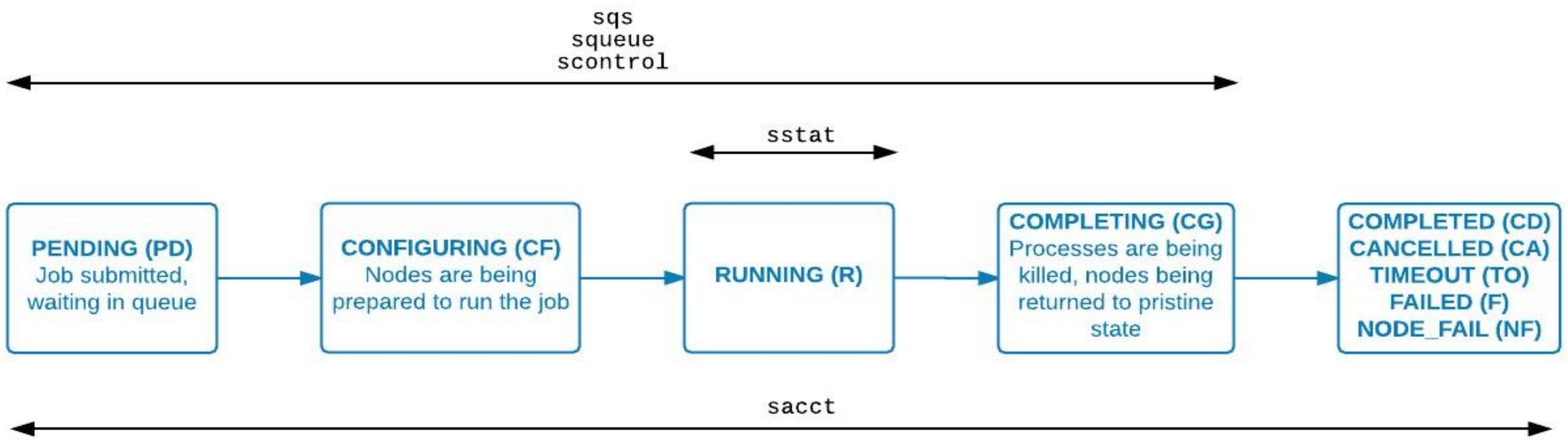
Running jobs – key points

- HPC work is via batch system
  - Dedicated subset of compute resources
  - Login nodes are shared resource for building code, editing scripts, etc. Use batch jobs for real work
- Key commands:
  - sbatch / salloc - submit a job
  - srun - start an (optionally MPI) application within a job
  - sqs - check the queue for my job status
- Don’t forget we have a reservation
  - #SBATCH –reservation=csgrrain

https://docs.nersc.gov/jobs/

Where is my job?

```
elvis@nersc:~> sqs
JOBID      ST REASON USER      NAME                 NODES USED REQUESTED ...
2774102    R  Prolog elvis myscript.q 2                   0:00 30:00

              ...              SUBMIT                     PARTITION RANK_P RANK_BF
                               2016-11-18T11:24:20 debug            N/A    N/A

elvis@nersc:~> ls -lt  total 11280
-rw-r----- 1 elvis elvis 132 Nov 18 11:24 slurm-2774102.out
-rw-r----- 1 elvis elvis 208 Nov 18 11:24 myscript.q
```


sqs
squeue
scontrol
sstat
PENDING (PD) Job submitted, waiting in queue
CONFIGURING (CF) Nodes are being prepared to run the job
RUNNING (R)
COMPLETING (CG) Processes are being killed, nodes being returned to pristine state
COMPLETED (CD)
CANCELLED (CA)
TIMEOUT (TO)
FAILED (F)
NODE_FAIL (NF)
sacct

Agenda

- Cori overview, logging in
- Run a simple job
- Building and running applications on Cori
  - Serial
  - Parallel (MPI)
  - Multithreaded (OpenMP)
- What affects performance?
  - Bottlenecks
  - Task placement and affinity
- Preparing for performance analysis

Cray compiler wrappers, ftn, cc and CC

- Building code optimally for Cori requires a complex set of compiler option and libraries
  - eg, static linking by default (important for performance at scale)
- Compiler wrappers ftn, cc and CC manage this complexity for you
  - Using environment variables set by the modules you have loaded
- Also provide MPI (so eg mpicc is not required)

Wait .. “environment modules” ?

- Sopware on Cori (and most HPC systems) is managed with “environment modules”
- Why?
  - Cori is a shared resource
  - Different people need different combinations of sorware, at different versions, with different dependencies (and for different jobs)
- Loading and unloading a module updates environment variables (eg $PATH, $LD_LIBRARY_PATH) to make a package available

Module Commands

module load <modulename>

– Add the module from your environment

module unload <modulename>

– Remove the module from your environment

module swap <module1> <module2>

– Unload one module and replace it with another

```
% module swap intel        intel/16.0.3.210
```

(replace current default to a specific version)

module list

– See what modules you have loaded right now

module show <modulename>

– See what the module actually does

module display <modulename>

– Show module details

module help <modulename>

– Get more information about the software

Key modules for compiling

- PrgEnv-intel / PrgEnv-cray / PrgEnv-gnu
  - Which underlying compiler the wrappers should invoke
- craype-haswell / craype-mic-knl
  - Remember, login nodes are haswell, but we are building for KNL!

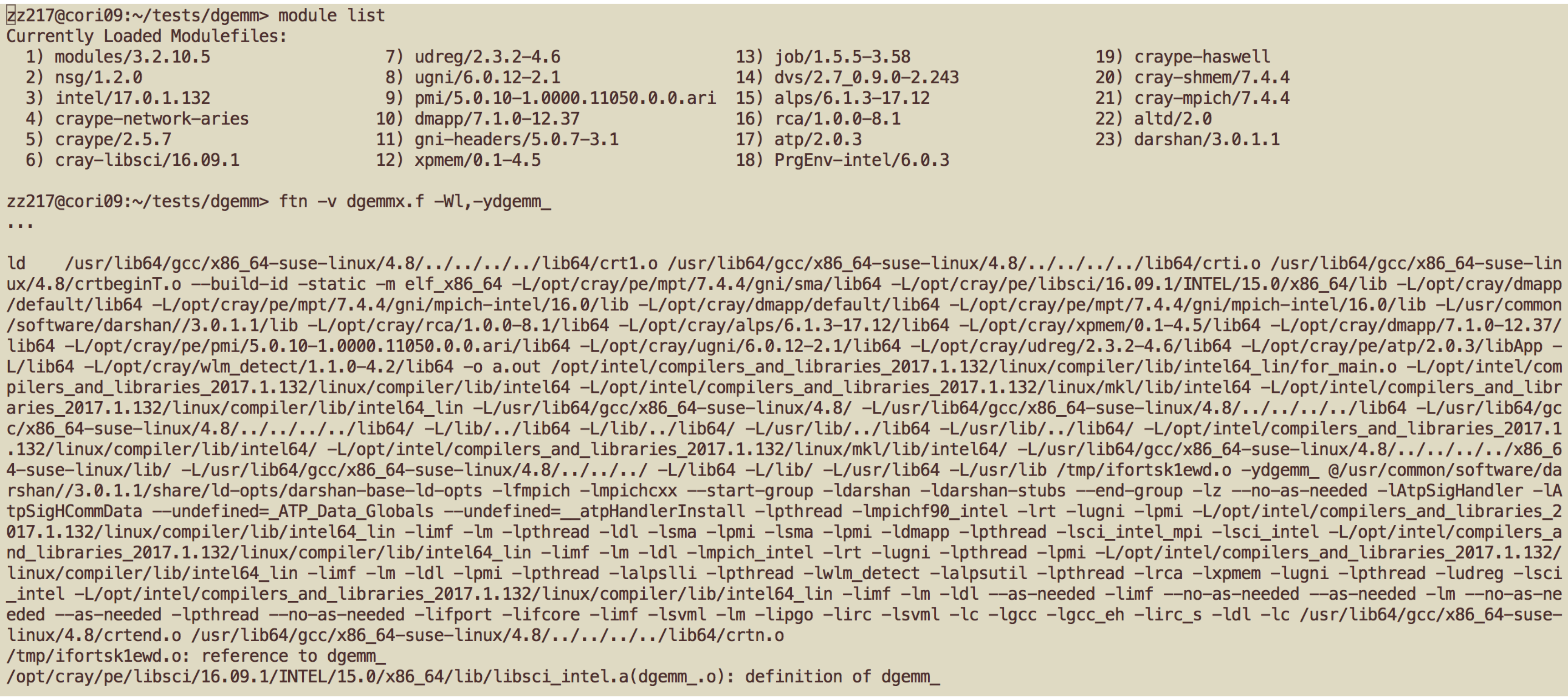
```
module swap craype-haswell          craype-mic-knl
```

  - Wrappers manage cross-compiling

Important for today!

What do compiler wrappers link by default?

- Depending on the modules loaded, MPI, LAPACK/BLAS/ScaLAPACK libraries, and more



```
zz217@cori09:~/tests/dgemm> module list
Currently Loaded Modulefiles:
  1) modules/3.2.10.5              7) udreg/2.3.2-4.6                  13) job/1.5.5-3.58                    19) craype-haswell
  2) nsg/1.2.0                     8) ugni/6.0.12-2.1                  14) dvs/2.7_0.9.0-2.243               20) cray-shmem/7.4.4
  3) intel/17.0.1.132              9) pmi/5.0.10-1.0000.11050.0.0.ari  15) alps/6.1.3-17.12                  21) cray-mpich/7.4.4
  4) craype-network-aries         10) dmapp/7.1.0-12.37                16) rca/1.0.0-8.1                     22) altd/2.0
  5) craype/2.5.7                 11) gni-headers/5.0.7-3.1            17) atp/2.0.3                         23) darshan/3.0.1.1
  6) cray-libsci/16.09.1          12) xpmem/0.1-4.5                    18) PrgEnv-intel/6.0.3

zz217@cori09:~/tests/dgemm> ftn -v dgemmx.f -Wl,-ydgemm_
...

ld    /usr/lib64/gcc/x86_64-suse-linux/4.8/../../../../lib64/crt1.o /usr/lib64/gcc/x86_64-suse-linux/4.8/../../../../lib64/crti.o /usr/lib64/gcc/x86_64-suse-lin
ux/4.8/crtbeginT.o --build-id -static -m elf_x86_64 -L/opt/cray/pe/mpt/7.4.4/gni/sma/lib64 -L/opt/cray/pe/libsci/16.09.1/INTEL/15.0/x86_64/lib -L/opt/cray/dmapp
/default/lib64 -L/opt/cray/pe/mpt/7.4.4/gni/mpich-intel/16.0/lib -L/opt/cray/dmapp/default/lib64 -L/opt/cray/pe/mpt/7.4.4/gni/mpich-intel/16.0/lib -L/usr/common
/software/darshan//3.0.1.1/lib -L/opt/cray/rca/1.0.0-8.1/lib64 -L/opt/cray/alps/6.1.3-17.12/lib64 -L/opt/cray/xpmem/0.1-4.5/lib64 -L/opt/cray/dmapp/7.1.0-12.37/
lib64 -L/opt/cray/pe/pmi/5.0.10-1.0000.11050.0.0.ari/lib64 -L/opt/cray/ugni/6.0.12-2.1/lib64 -L/opt/cray/udreg/2.3.2-4.6/lib64 -L/opt/cray/pe/atp/2.0.3/libApp -
L/lib64 -L/opt/cray/wlm_detect/1.1.0-4.2/lib64 -o a.out /opt/intel/compilers_and_libraries_2017.1.132/linux/compiler/lib/intel64_lin/for_main.o -L/opt/intel/com
pilers_and_libraries_2017.1.132/linux/compiler/lib/intel64 -L/opt/intel/compilers_and_libraries_2017.1.132/linux/mkl/lib/intel64 -L/opt/intel/compilers_and_libr
aries_2017.1.132/linux/compiler/lib/intel64_lin -L/usr/lib64/gcc/x86_64-suse-linux/4.8/ -L/usr/lib64/gcc/x86_64-suse-linux/4.8/../../../../lib64 -L/usr/lib64/gc
c/x86_64-suse-linux/4.8/../../../../lib64/ -L/lib/../lib64 -L/lib/../lib64/ -L/usr/lib/../lib64 -L/usr/lib/../lib64/ -L/opt/intel/compilers_and_libraries_2017.1
.132/linux/compiler/lib/intel64/ -L/opt/intel/compilers_and_libraries_2017.1.132/linux/mkl/lib/intel64/ -L/usr/lib64/gcc/x86_64-suse-linux/4.8/../../../../x86_6
4-suse-linux/lib/ -L/usr/lib64/gcc/x86_64-suse-linux/4.8/../../../ -L/lib64 -L/lib/ -L/usr/lib64 -L/usr/lib /tmp/ifortsk1ewd.o -ydgemm_ @/usr/common/software/da
rshan//3.0.1.1/share/ld-opts/darshan-base-ld-opts -lfmpich -lmpichcxx --start-group -ldarshan -ldarshan-stubs --end-group -lz --no-as-needed -lAtpSigHandler -lA
tpSigHCommData --undefined=_ATP_Data_Globals --undefined=__atpHandlerInstall -lpthread -lmpichf90_intel -lrt -lugni -lpmi -L/opt/intel/compilers_and_libraries_2
017.1.132/linux/compiler/lib/intel64_lin -limf -lm -lpthread -ldl -lsma -lpmi -lsma -lpmi -ldmapp -lpthread -lsci_intel_mpi -lsci_intel -L/opt/intel/compilers_a
nd_libraries_2017.1.132/linux/compiler/lib/intel64_lin -limf -lm -ldl -lmpich_intel -lrt -lugni -lpthread -lpmi -L/opt/intel/compilers_and_libraries_2017.1.132/
linux/compiler/lib/intel64_lin -limf -lm -ldl -lpmi -lpthread -lalpslli -lpthread -lwlm_detect -lalpsutil -lpthread -lrca -lxpmem -lugni -lpthread -ludreg -lsci
_intel -L/opt/intel/compilers_and_libraries_2017.1.132/linux/compiler/lib/intel64_lin -limf -lm -ldl --as-needed -limf --no-as-needed --as-needed -lm --no-as-ne
eded --as-needed -lpthread --no-as-needed -lifport -lifcore -limf -lsvml -lm -lipgo -lirc -lsvml -lc -lgcc -lgcc_eh -lirc_s -ldl -lc /usr/lib64/gcc/x86_64-suse-
linux/4.8/crtend.o /usr/lib64/gcc/x86_64-suse-linux/4.8/../../../../lib64/crtn.o
/tmp/ifortsk1ewd.o: reference to dgemm_
/opt/cray/pe/libsci/16.09.1/INTEL/15.0/x86_64/lib/libsci_intel.a(dgemm_.o): definition of dgemm_
```

Compiling code

- Very similar to regular Linux, but using CC / cc / ftn
- Do this bit once:

```
module swap craype-haswell              craype-mic-knl
```

- Then:

```
cc    –c hack-a-kernel.c
cc    –o hack-a-kernel.ex             hack-a-kernel.o
```

- Note that the module looks aper CPU target!

Compiling parallel code

- Compiler wrappers give you MPI "for free"

```
CC –c hello-mpi.c++
CC –o hello-mpi.ex                hello-mpi.o
```

- (Cray MPICH – optimized for Aries HSN)

- OpenMP: with PrgEnv-intel (NERSC default):

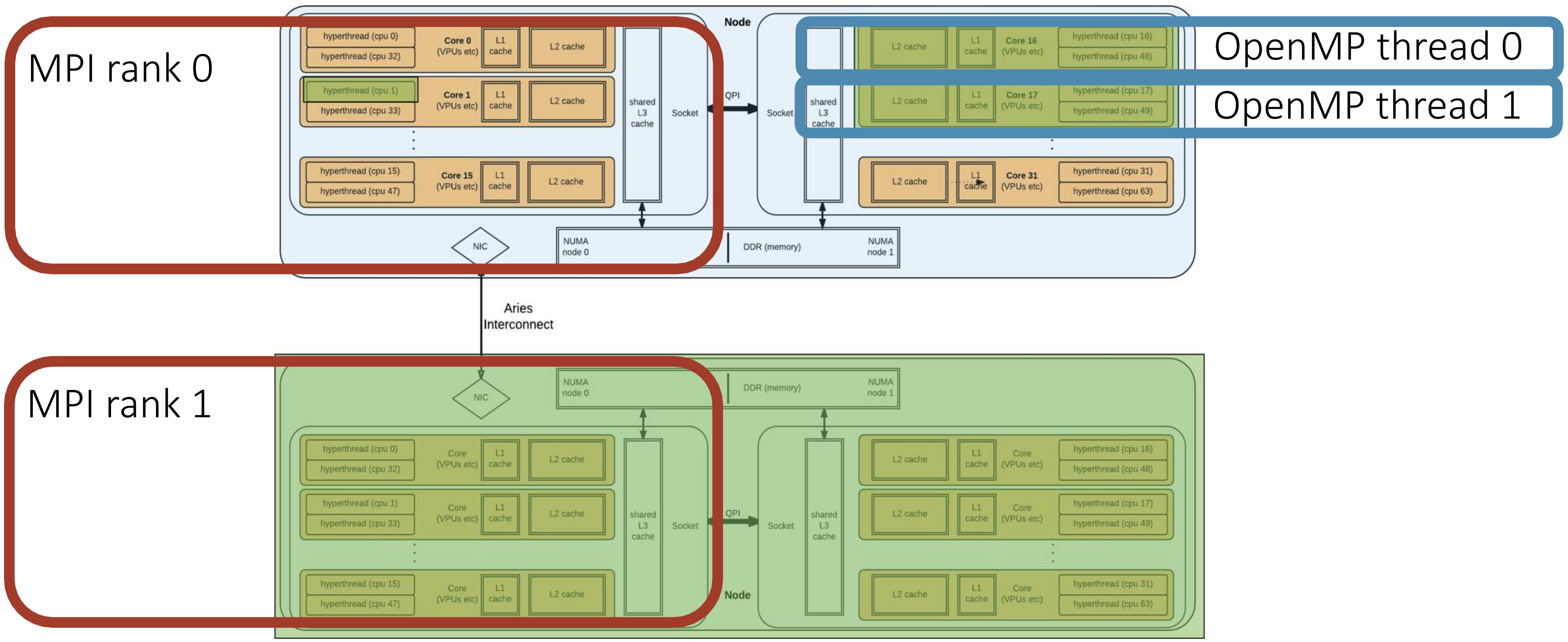
```
cc –qopenmp –c hello-omp.c
cc –qopenmp –o hello-omp.ex hello-omp.o
```

MPI vs OpenMP

- MPI provides explicit communication between separate processes
  - Optionally on separate nodes – ie packets over a network
  - Most parallel development in last 2 decades has used this approach
- OpenMP provides work-sharing and synchronization between threads in a single process
  - Threads share the same memory image
  - To make the most of a KNL node, most applications will need to use OpenMP

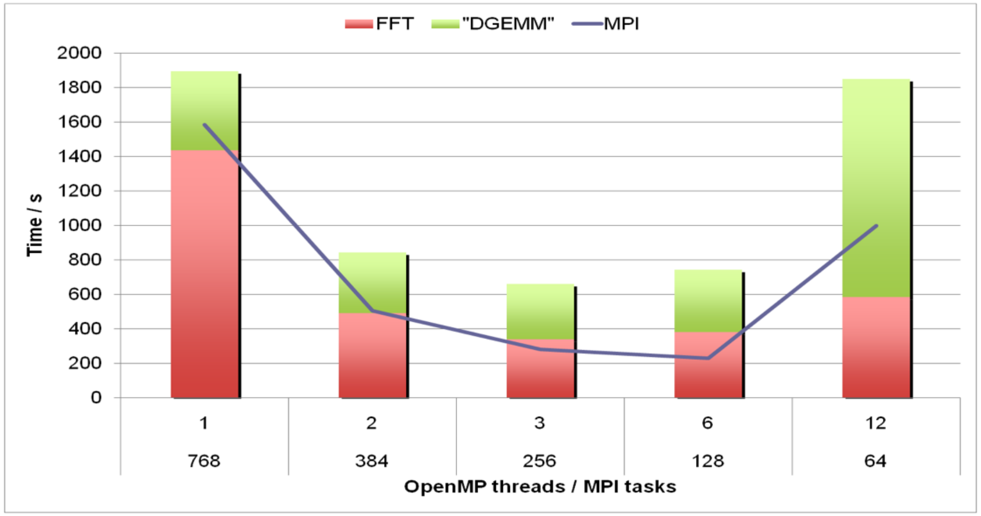
MPI vs OpenMP

- An MPI application can have processes on more than one node
- An OpenMP application exists entirely within 1 node



Why do we suddenly need OpenMP?

- **Multi-level parallelism**
  - At very large scale, the overheads of MPI (or any parallel approach) become excessively costly
  - Combining (nesting) parallel approaches allows us to operate each at lower scale
    - Sweet spot for best overall efficiency
  - MPI -> OpenMP -> Vectorization



Why do we suddenly need OpenMP?

- **Memory-per-core is trending downwards**
  - Cori Haswell: 128GB for 32 cores
  - Cori KNL: 96GB for 68 cores (16GB MCDRAM for 68 cores)
  - Parallelism within same memory footprint is necessary

Agenda

- Cori overview, logging in
- Run a simple job
- Building and running applications on Cori
  - Serial
  - Parallel (MPI)
  - Multithreaded (OpenMP)
- What affects performance?
  - Bottlenecks
  - Task placement and affinity
- Preparing for performance analysis

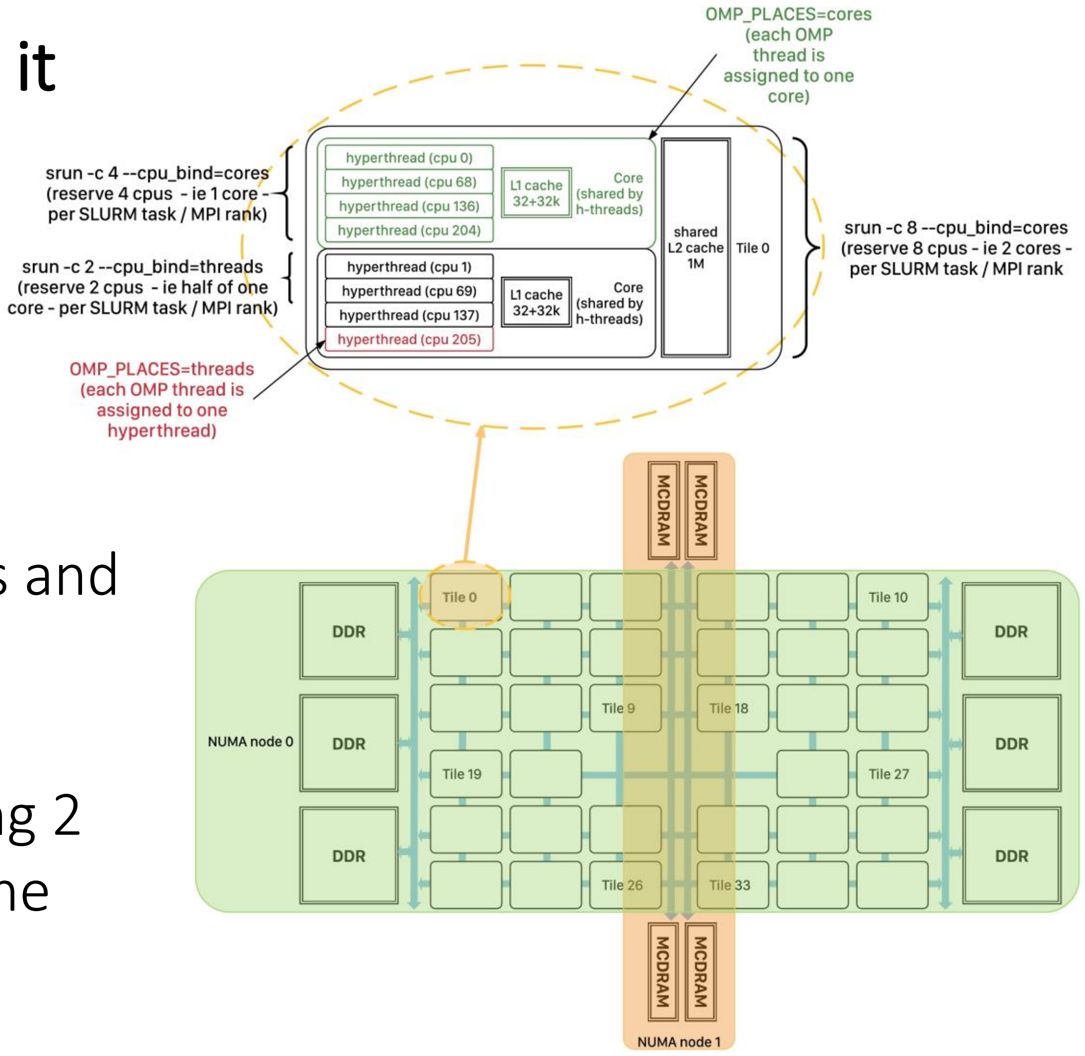
Performance bottlenecks

- Any point at which some component of the system or application is stalled, waiting on some other component, is a bottleneck

- Eg waiting for MPI call to complete
  - Load imbalance or communication overhead is a bottleneck
- BUT for today we are interested in KNL specifics
  - Bottlenecks within the node

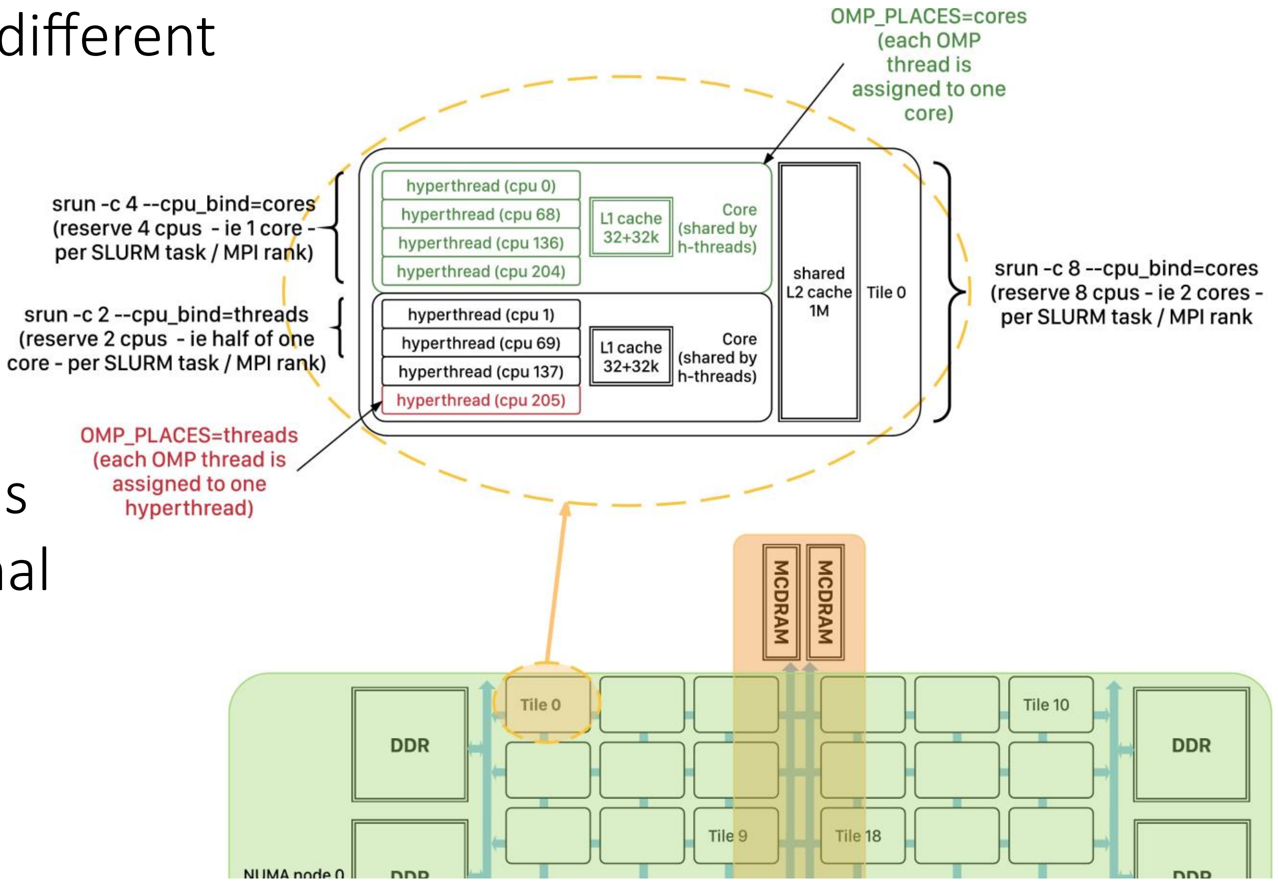
Bottlenecks within the node – Affinity issues

- Firstly: Am I running it right?
  - Cori KNL has 68 cores per node, arranged on a mesh of 34 tiles
  - Each tile has 2 cores and a shared L2 cache
  - Each core has 4 hyperthreads sharing 2 VPUs and an L1 cache


OMP_PLACES=cores (each OMP thread is assigned to one core)
srun -c 4 --cpu_bind=cores (reserve 4 cpus - ie 1 core - per SLURM task / MPI rank)
hyperthread (cpu 0)
hyperthread (cpu 68)
hyperthread (cpu 136)
hyperthread (cpu 204)
L1 cache 32+32k
Core (shared by h-threads)
shared L2 cache 1M
Tile 0
srun -c 8 --cpu_bind=cores (reserve 8 cpus - ie 2 cores - per SLURM task / MPI rank
srun -c 2 --cpu_bind=threads (reserve 2 cpus - ie half of one core - per SLURM task / MPI rank)
hyperthread (cpu 1)
hyperthread (cpu 69)
hyperthread (cpu 137)
hyperthread (cpu 205)
L1 cache 32+32k
Core (shared by h-threads)
OMP_PLACES=threads (each OMP thread is assigned to one hyperthread)
MCDRAM
MCDRAM
Tile 0
Tile 10
DDR
DDR
Tile 9
Tile 18
NUMA node 0
DDR
DDR
Tile 19
Tile 27
DDR
DDR
Tile 26
Tile 33
MCDRAM
MCDRAM
NUMA node 1

Bottlenecks within the node – Affinity issues

- Where are my threads?
  - Is each using a different core at least?
- Linux does not always choose best placement
  - Use srun options to ensure optimal thread/process placement

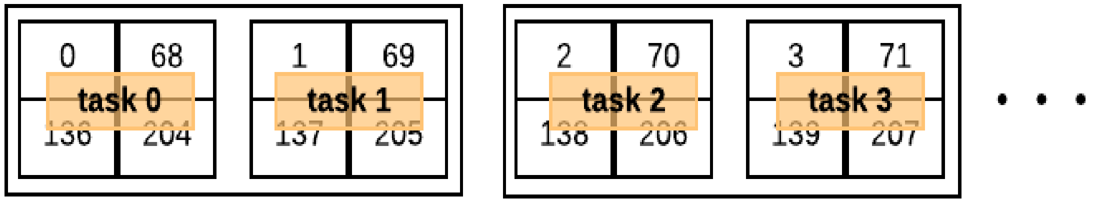

OMP_PLACES=cores (each OMP thread is assigned to one core)
srun -c 4 --cpu_bind=cores (reserve 4 cpus - ie 1 core - per SLURM task / MPI rank)
srun -c 2 --cpu_bind=threads (reserve 2 cpus - ie half of one core - per SLURM task / MPI rank)
hyperthread (cpu 0)
hyperthread (cpu 68)
hyperthread (cpu 136)
hyperthread (cpu 204)
L1 cache 32+32k
Core (shared by h-threads)
hyperthread (cpu 1)
hyperthread (cpu 69)
hyperthread (cpu 137)
hyperthread (cpu 205)
L1 cache 32+32k
Core (shared by h-threads)
shared L2 cache 1M
Tile 0
srun -c 8 --cpu_bind=cores (reserve 8 cpus - ie 2 cores - per SLURM task / MPI rank
OMP_PLACES=threads (each OMP thread is assigned to one hyperthread)
MCDRAM
MCDRAM
DDR
DDR
Tile 0
Tile 10
Tile 9
Tile 18
NUMA node 0
DDR
DDR

Process (task) affinity

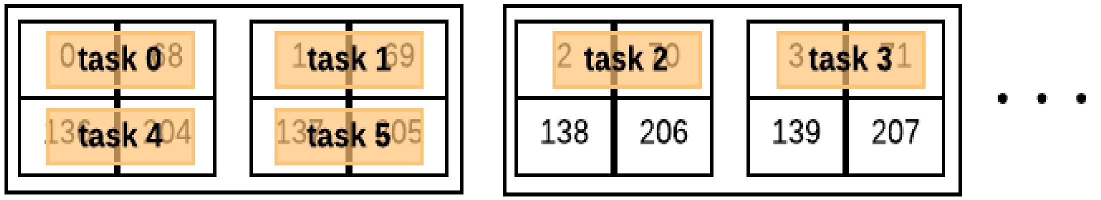
Solution: use --cpu_bind:

```
srun –n 64 -c 4 --cpu_bind=verbose,cores                    ./my_exec
srun -n 128 -c 2 --cpu_bind=verbose,threads                 ./my_exec
```

- Controls what a task (MPI rank) is bound to
  - If no more than 1 MPI rank per core: --cpu_bind=cores

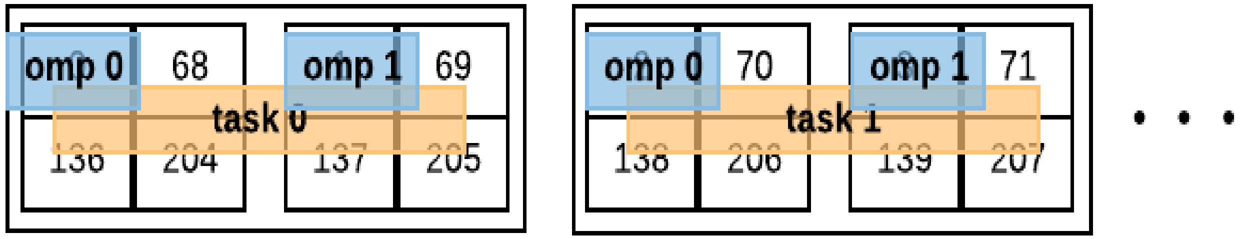


  - If more than 1 MPI rank per core: --cpu_bind=threads



Thread affinity (OpenMP)

```
export OMP_NUM_THREADS=2  export
OMP_PROC_BIND=spread  export                # or close
OMP_PLACES=cores                            # or threads, or sockets
srun -n        32 -c 8 --cpu_bind=verbose,cores              ./my_exec
```



...If using hyperthreads, use OMP_PLACES=threads

Memory affinity

Linux default behavior is to allocate to closest NUMA-node, if possible

Not always optimal:

- KNL nodes: DDR is “closer” than MCDRAM

```
#SBATCH -C knl,quad,flat  export
OMP_NUM_THREADS=4
srun -n16 -c16 --cpu_bind=cores --mem_bind=map_mem:1 ./a.out
```

- NUMA node 1 is MCDRAM in quad,flat mode
- “Mandatory” mapping: if using >16GB, malloc will fail

NOTE: today’s reservation is for “cache-mode” nodes, so MCDRAM is invisible

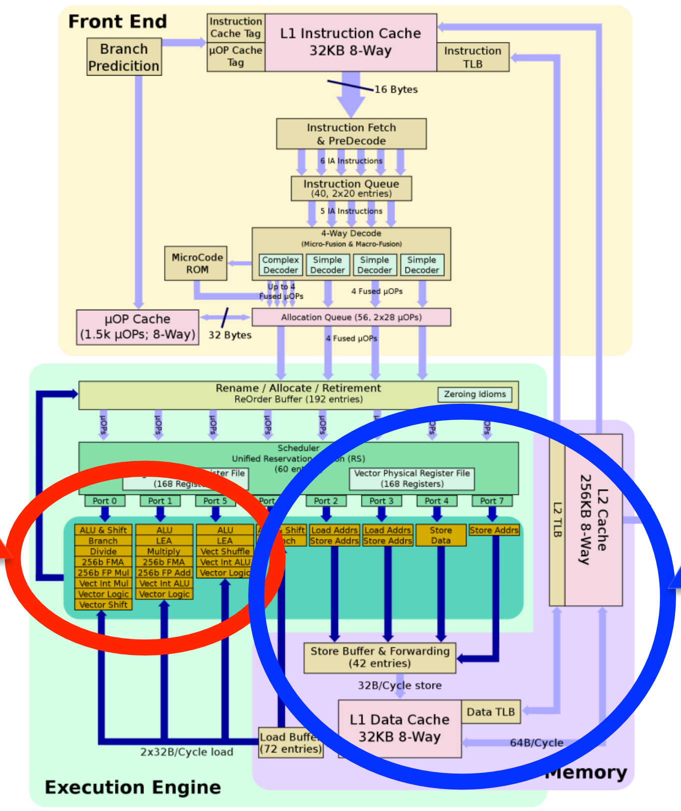
Bottlenecks within the node – Affinity issues

OMP_PROC_BIND=true OMP_PLACES=cores (threads) srun -c 4 --cpu_bind=cores …

Bottlenecks within the node

- Performance tends to be dominated by:

Doing arithmetic operations (# FLOPS/cycle)

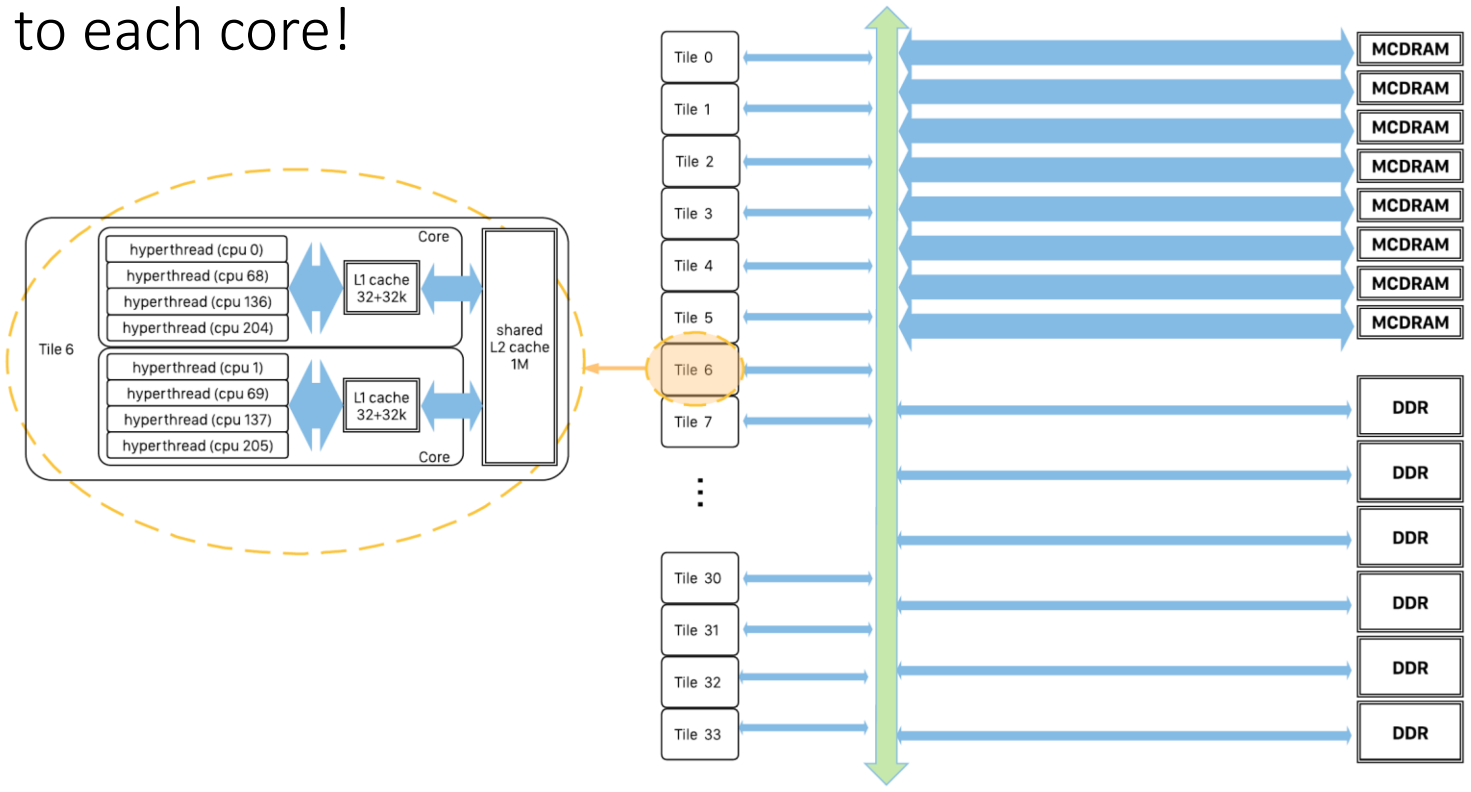


Moving data between arithmetic units and memory

- Bottlenecks are usually due to one of these parts waiting on the other

Bottlenecks within the node

- Bandwidth!
  - MCDRAM is very high bandwidth (~450 GB/s)
  - .. But aggregate bandwidth is not the same as bandwidth to each core!

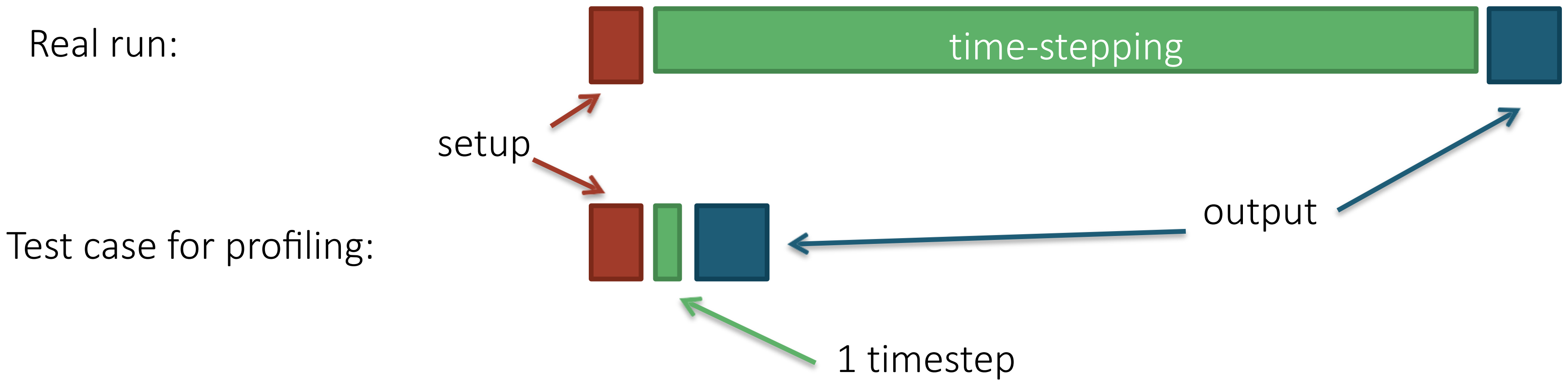


Agenda

- Cori overview, logging in
- Run a simple job
- Building and running applications on Cori
  - Serial
  - Parallel (MPI)
  - Multithreaded (OpenMP)
- What affects performance?
  - Bottlenecks
  - Task placement and affinity
- Preparing for performance analysis

Profiling prerequisite

- A small, short, but representative test case for your application
  - Profiling tends to be costly
    - Runtime overhead
    - Size of data collected
  - **BUT**: must cover same paths through code as “real” example (or at least, the differences must be understood)
    - What could go wrong with this test case?



…But my application *just is* big!

- Start with a low-overhead profiling method
  - Eg sampling-based (gprof, TAU, CrayPat, …)
  - Identify hotspots
- Only profile part of a run
  - Some tools (eg Vtune) allow you to start the run “paused” and “resume collection” via an API call
- Only profile 1 MPI rank
  - Via srun options, eg run Vtune on one rank, not others (beyond today’s scope)
- Luckily, our hack-a-kernel is already small!

Performance Analysis Gotchas

- Many tools require dynamically-linked executable
  - Including Vtune
  - NERSC/Cray: applications are statically linked by default, must use “-dynamic” at compile time
- “Uncore” (eg memory related) performance counters usually require special permissions (eg kernel module)
  - NERSC/Slurm supports this with #SBATCH –perf=vtune directive

Summary and Recap

- Now you can:
  - Log in
  - Build an application
  - Run an application, and be aware of how and where it is running
  - Prepare an application for performance analysis with Vtune
- Q & A?

NERSC