Part 1: Cluster

High-Performance Computing

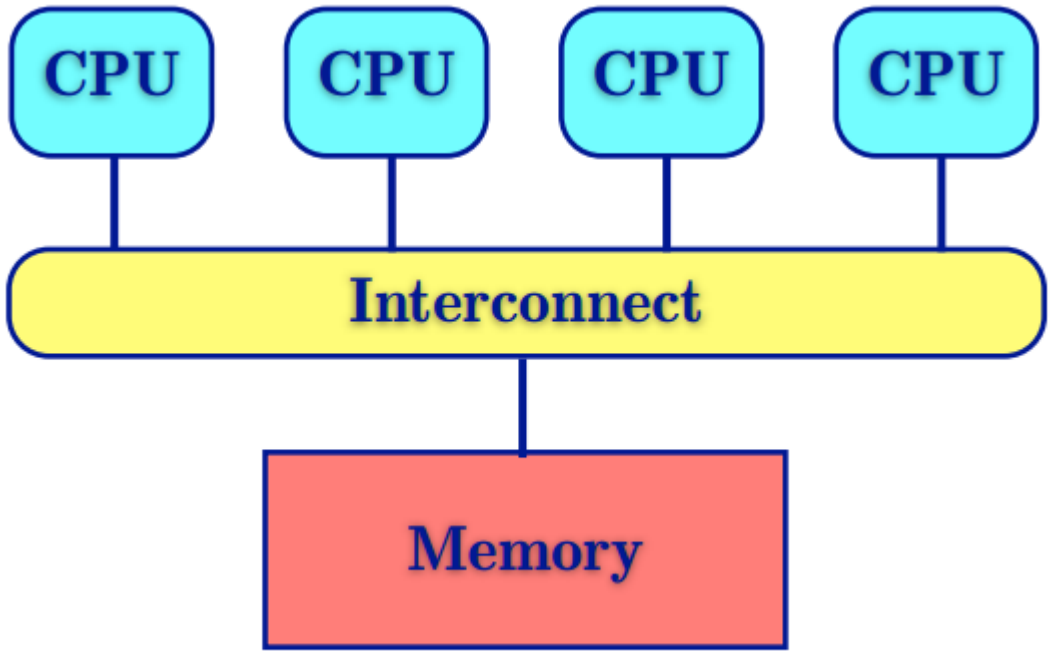
Summer 2021 at GIST

Tae-Hyuk (Ted) Ahn
Department of Computer Science
Program of Bioinformatics and Computational Biology
Saint Louis University

SAINT LOUIS
UNIVERSITY
EST. 1818

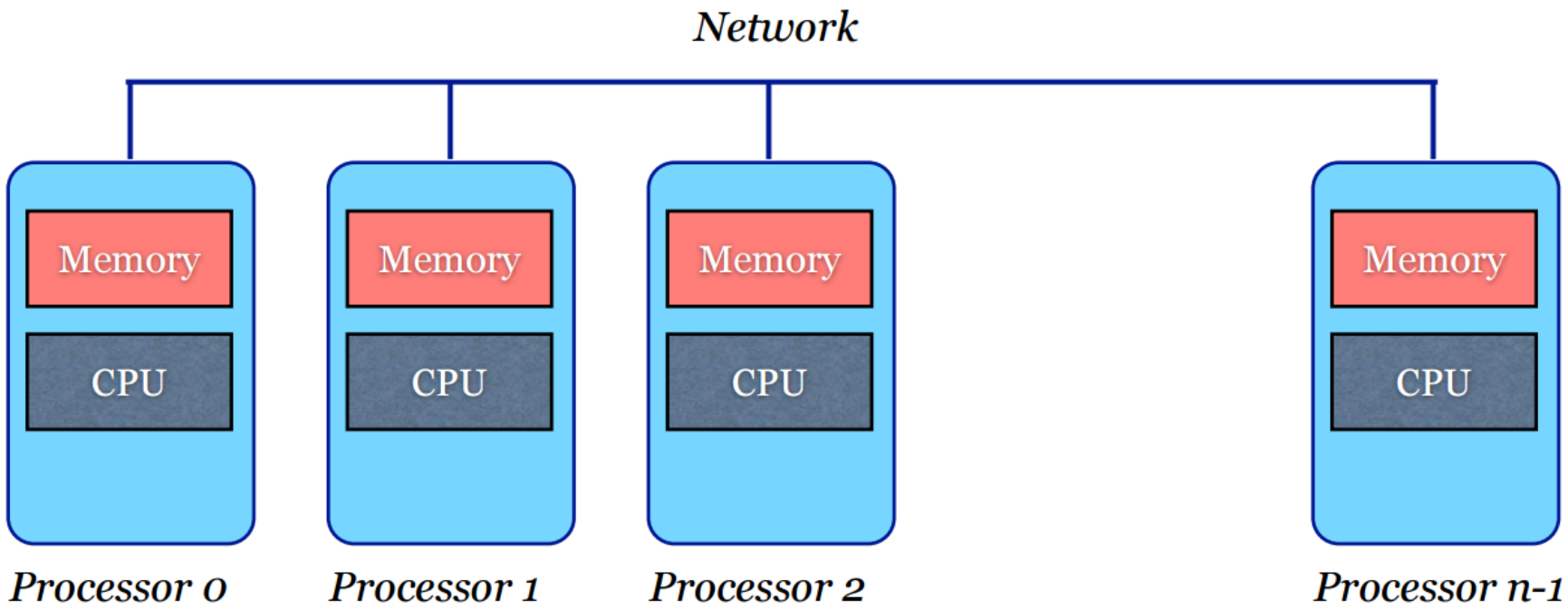
You learned OpenMP for Shared Memory System

- There is a single shared address space for all processors.
- All processors share the same view of memory.



Let us consider Distributed Memory System

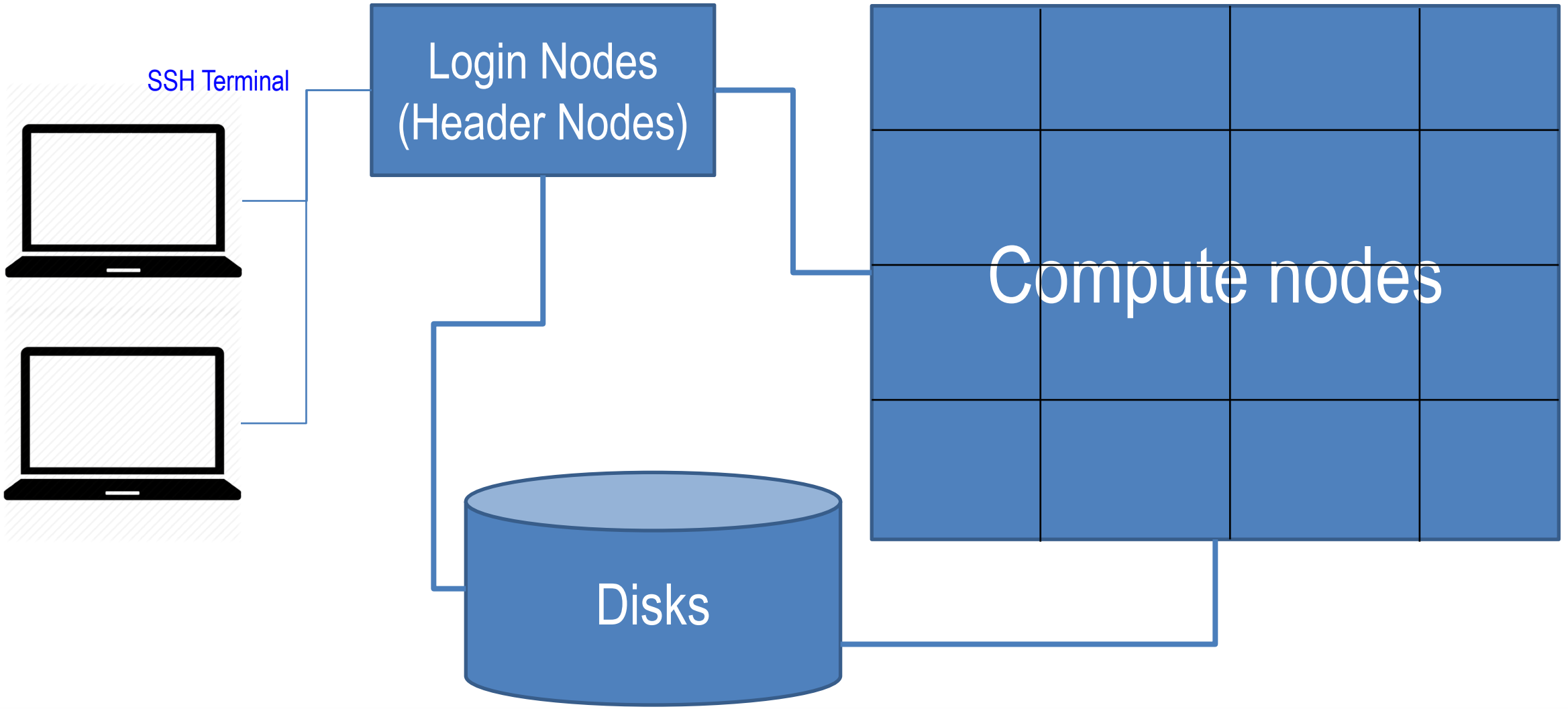
- Each processor has its own private memory.
- A network connects all the processors.



Cluster

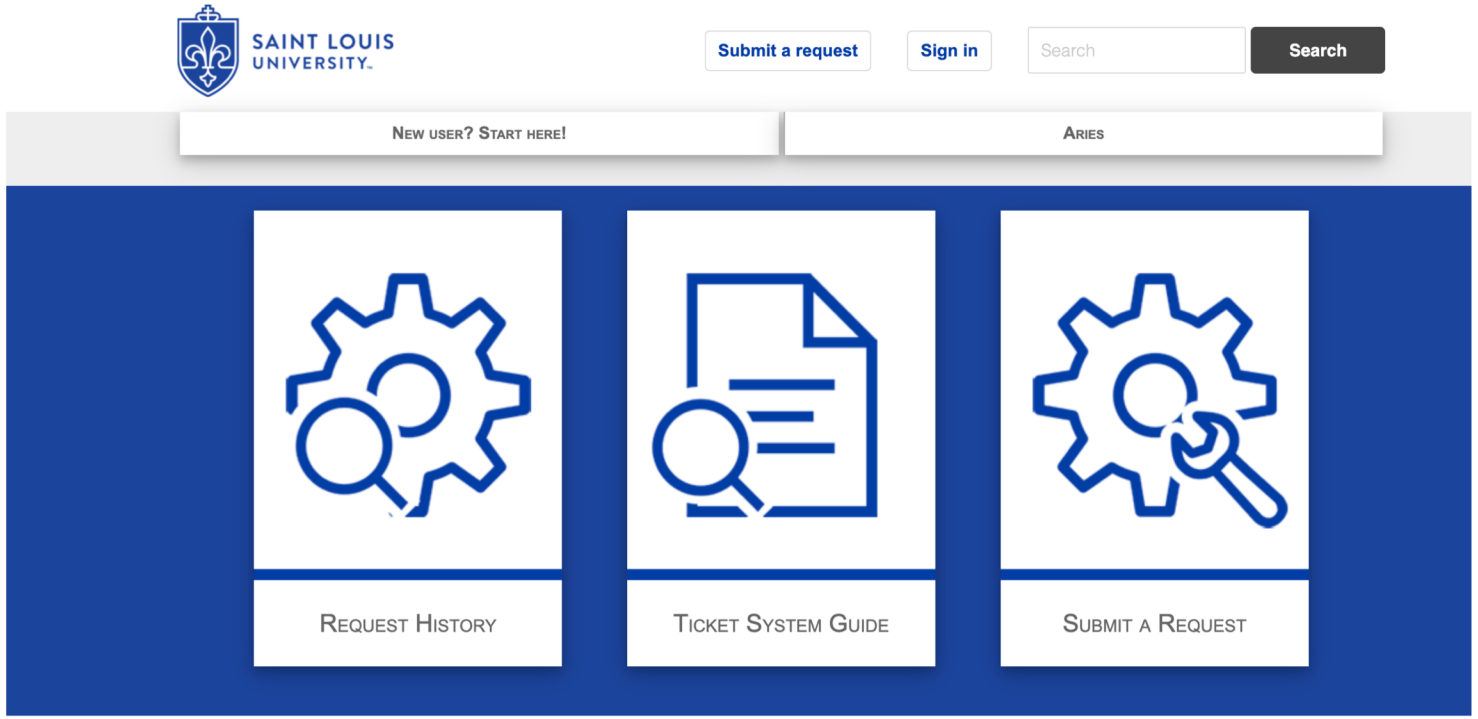
- A cluster needs:
  - Several computers, nodes, often in special cases for easy mounting in a rack
  - One or more networks (interconnects) to hook the nodes together
  - Software that allows the nodes to communicate with each other (e.g. MPI)
  - Software that reserves resources to individual users
- A cluster is: all of those components working together to form one big computer

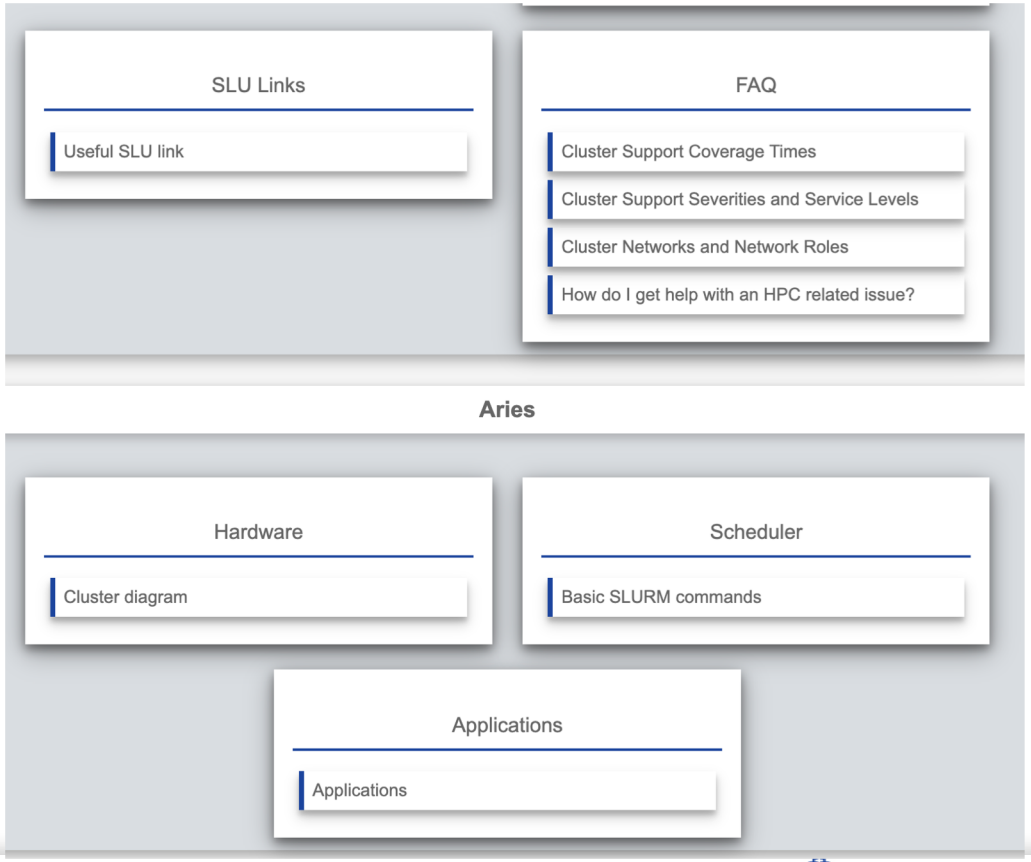
Typical Cluster System Layout

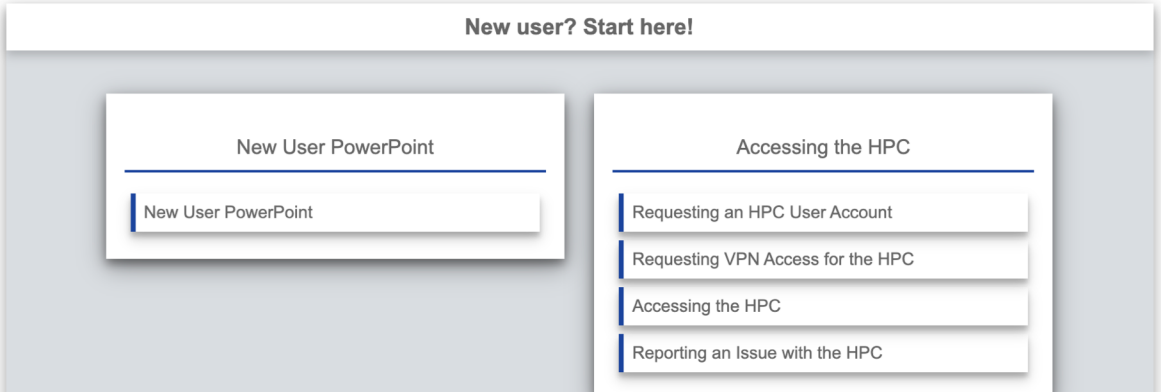


Let us explain cluster using a HPC system at SLU

- https://slu.zendesk.com/hc/en-us



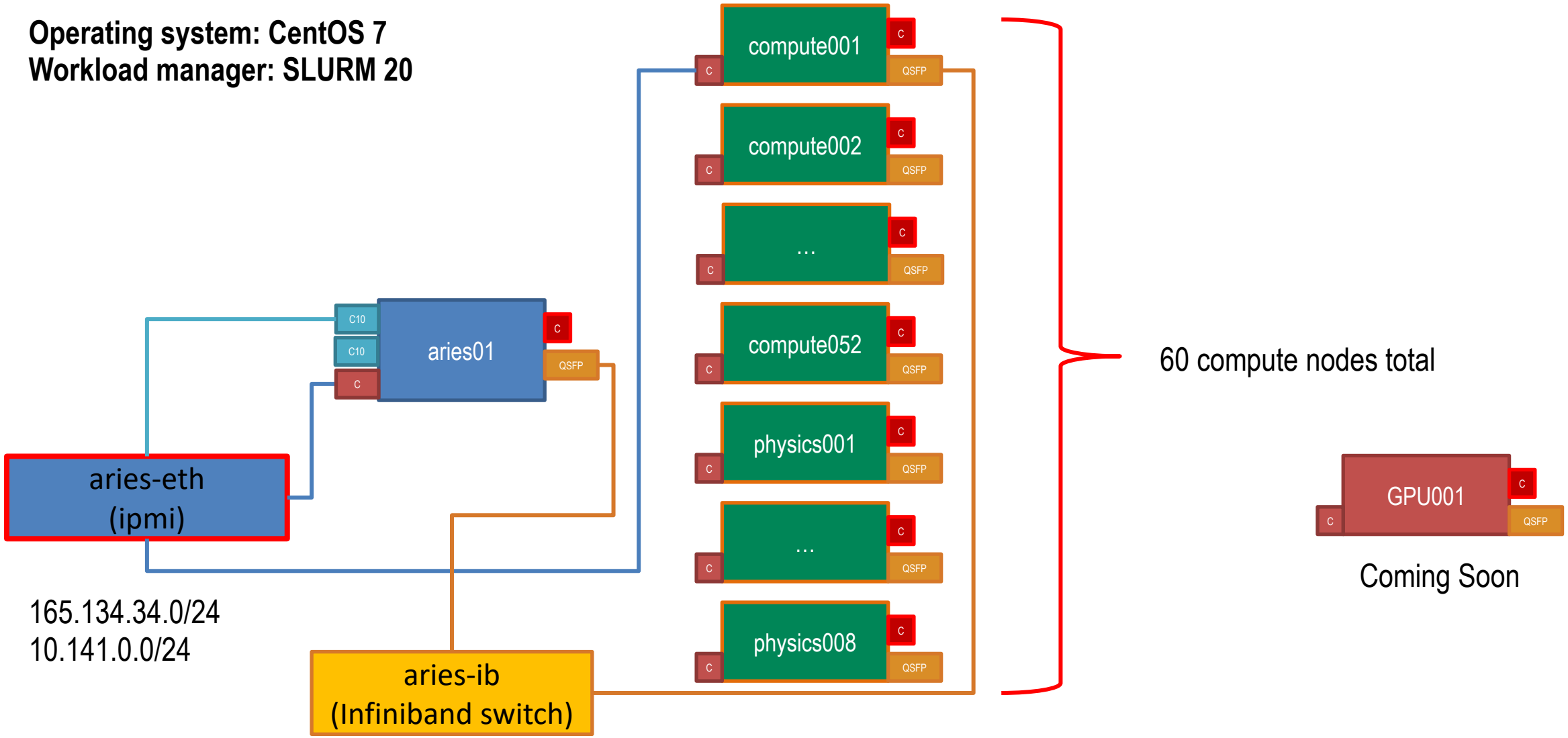




SAINT LOUIS UNIVERSITY

Aries Cluster – Overview

Operating system: CentOS 7
Workload manager: SLURM 20



165.134.34.0/24
10.141.0.0/24

CentOS

- https://www.centos.org/

- Compared to other OS (ubuntu, Fedora, and more), it is stable!

Logging into the SLU VPN

1. SLU ITS has adopted Okta for login

   Okta gives the option of using push for verification.

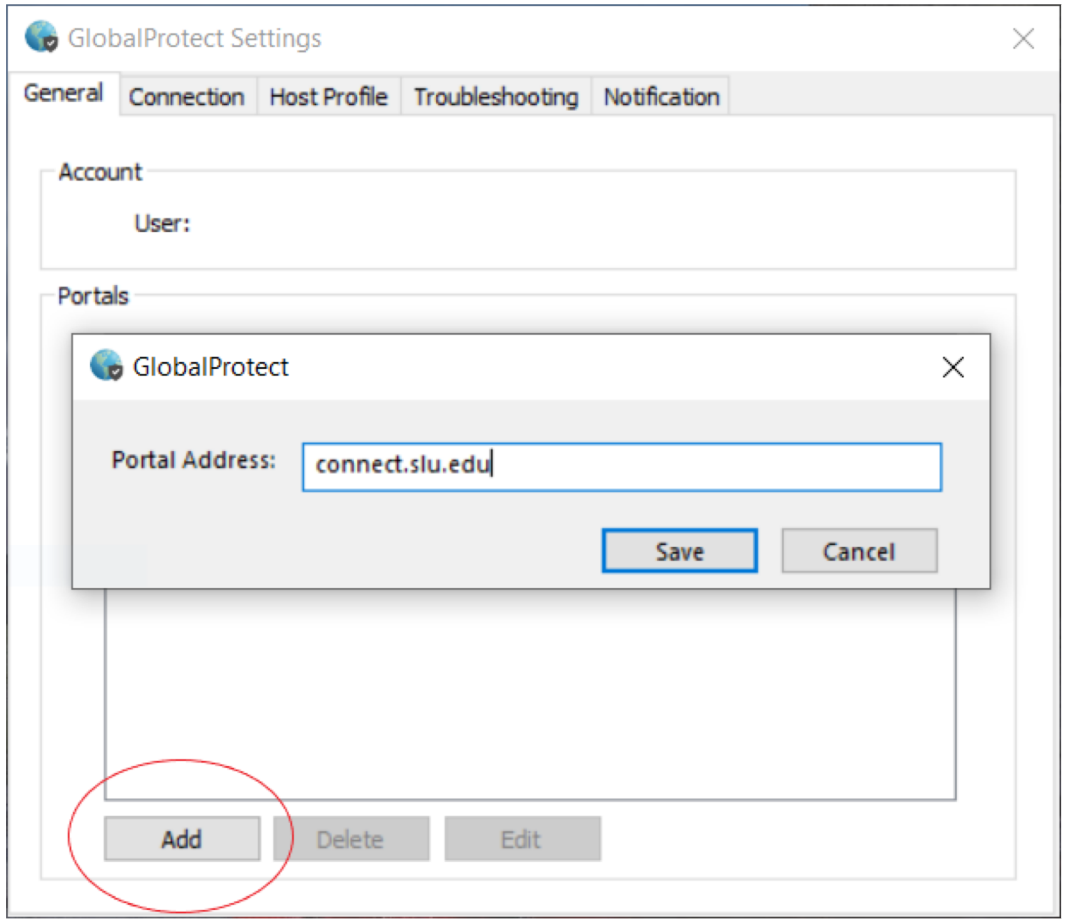
2. Download OpenConnect

   Follow this link for instructions

3. Configure OpenConnect

   Add “connect.slu.edu” to portals (see picture, right)

4. Sign in using VPN credentials



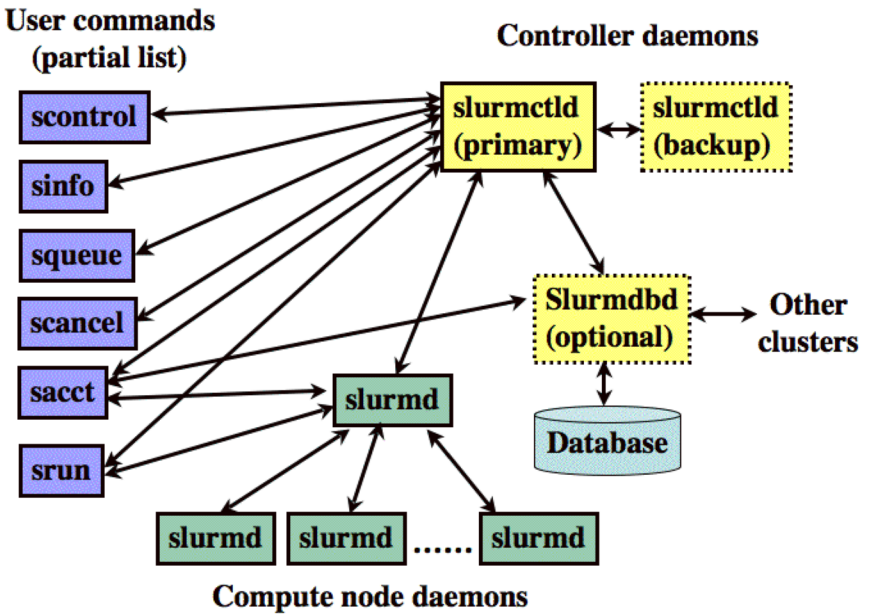
SLURM

- SLURM is the job scheduler. This allows jobs to be run on a node.
- How the jobs starts, runs, and completes is normally dictated by a script.
- More detailed information and tutorials can be found at SLURM information page.

https://slurm.schedmd.com/

Overview

Slurm is an open source, fault-tolerant, and highly scalable cluster management and job scheduling system for large and small Linux clusters. Slurm requires no kernel modifications for its operation and is relatively self-contained. As a cluster workload manager, Slurm has three key functions. First, it allocates exclusive and/or non-exclusive access to resources (compute nodes) to users for some duration of time so they can perform work. Second, it provides a framework for starting, executing, and monitoring work (normally a parallel job) on the set of allocated nodes. Finally, it arbitrates contention for resources by managing a queue of pending work. Optional plugins can be used for accounting, advanced reservation, gang scheduling (time sharing for parallel jobs), backfill scheduling, topology optimized resource selection, resource limits by user or bank account, and sophisticated multifactor job prioritization algorithms.



Basic Slurm Commands

Man pages exist for all Slurm daemons, commands, and API functions. The command option --help also provides a brief summary of options. Note that the command options are all case sensitive.

- **sacct** is used to report job or job step accounting information about active or completed jobs. **#sacct --allocations**
- **salloc** is used to allocate resources for a job in real time. Typically this is used to allocate resources and spawn a shell. The shell is then used to execute srun commands to launch parallel tasks.
- **sattach** is used to attach standard input, output, and error plus signal capabilities to a currently running job or job step. One can attach to and detach from jobs multiple times.
- **sbatch** is used to submit a job script for later execution. The script will typically contain one or more srun commands to launch parallel tasks.
- **sbcast** is used to transfer a file from local disk to local disk on the nodes allocated to a job. This can be used to effectively use diskless compute nodes or provide improved performance relative to a shared file system.
- **scancel** is used to cancel a pending or running job or job step. It can also be used to send an arbitrary signal to all processes associated with a running job or job step.
- **scontrol** is the administrative tool used to view and/or modify Slurm state. Note that many scontrol commands can only be executed as user root.
- **sinfo** reports the state of partitions and nodes managed by Slurm. It has a wide variety of filtering, sorting, and formatting options. **# sinfo -N all**
- **squeue** reports the state of jobs or job steps. It has a wide variety of filtering, sorting, and formatting options. By default, it reports the running jobs in priority order and then the pending jobs in priority order. **# squeue -a**
- **srun** is used to submit a job for execution or initiate job steps in real time. srun has a wide variety of options to specify resource requirements, including: minimum and maximum node count, processor count, specific nodes to use or not use, and specific node characteristics (so much memory, disk space, certain required features, etc.). A job can contain multiple job steps executing sequentially or in parallel on independent or shared resources within the job's node allocation.
- **strigger** is used to set, get or view event triggers. Event triggers include things such as nodes going down or jobs approaching their time limit.
- **sview** is a graphical user interface to get and update state information for jobs, partitions, and nodes managed by Slurm.

How to check partition?

- First we determine what partitions exist on the system, what nodes they include, and general system state. This information is provided by the **sinfo** command.

```
[ahnt@apex csci4850]$ sinfo
 NODES PARTITION     STATE   CPUS    S:C:T    MEMORY NODELIST
     1 defq*         maint     24   2:12:1    128842 compute-009
     1 defq*         down*     20   2:10:2     64343 compute-023
     1 defq*         down*     20   2:10:2    128855 compute-028
     1 defq*        draine     24   2:12:1    515914 compute-013
     1 defq*        draine     20   2:10:2     64343 compute-024
     1 defq*        draine     20   2:10:2    257879 compute-030
     3 defq*        reserv     24   2:12:1     64330 compute-[006-008]
     1 defq*        alloca     20   2:10:2    515927 compute-027
     5 defq*          idle     24   2:12:1     64330 compute-[001-005]
     3 defq*          idle     24   2:12:1    128842 compute-[010-012]
     3 defq*          idle     24   2:12:1    515914 compute-[014-016]
    16 defq*          idle     20   2:10:2     64343 compute-[017-022,029,031-032],physics-[001-003,005-008]
     1 defq*          idle     20   2:10:2    128855 compute-025
     1 defq*          idle     20   2:10:2    257879 compute-026
     1 genomics     draine     24   2:12:1    515914 compute-013
     1 genomics     alloca     20   2:10:2    515927 compute-027
     3 genomics       idle     24   2:12:1    515914 compute-[014-016]
```

How to check jobs?

- Next we determine what jobs exist on the system using the squeue command.

```
[ahnt@apex csci4850]$ squeue
JOBID     PARTITION USER              NODELIST          TIME          ST  NAME
24848     genomics  robert.kousnetsov compute-027       17:49:40      R   runRScript
[ahnt@apex csci4850]$
```

More detailed info about nodes, partitions, and jobs

- The **scontrol** command can be used to report more detailed information about nodes, partitions, jobs, job steps, and configuration.
- The reservation (**csci4850**) is on three nodes (compute-00[6-8], 24 physical cores and 64G MEM each)

```
[ahnt@apex csci4850]$ scontrol show node compute-006
NodeName=compute-006 Arch=x86_64 CoresPerSocket=12
   CPUAlloc=0 CPUTot=24 CPULoad=1.01
   AvailableFeatures=(null)
   ActiveFeatures=(null)
   Gres=(null)
   NodeAddr=compute-006 NodeHostName=compute-006 Version=18.08
   OS=Linux 3.10.0-1062.1.1.el7.x86_64 #1 SMP Fri Sep 13 22:55:44 UTC 2019
   RealMemory=64330 AllocMem=0 FreeMem=487 Sockets=2 Boards=1
   State=RESERVED ThreadsPerCore=1 TmpDisk=2038 Weight=1 Owner=N/A MCS_label=N/A
   Partitions=defq
   BootTime=2020-01-10T08:29:12 SlurmdStartTime=2020-01-10T08:31:28
   CfgTRES=cpu=24,mem=64330M,billing=24
   AllocTRES=
   CapWatts=n/a
   CurrentWatts=0 LowestJoules=0 ConsumedJoules=0
   ExtSensorsJoules=n/s ExtSensorsWatts=0 ExtSensorsTemp=n/s
```

srun – run a parallel job on cluster

- It is possible to create a resource allocation and launch the tasks for a job step in a single command line using the **srun** command

```
[ahnt@apex csci4850]$ srun -N3 -l /bin/hostname
1: compute-011
0: compute-010
2: compute-012
```

- Can you try

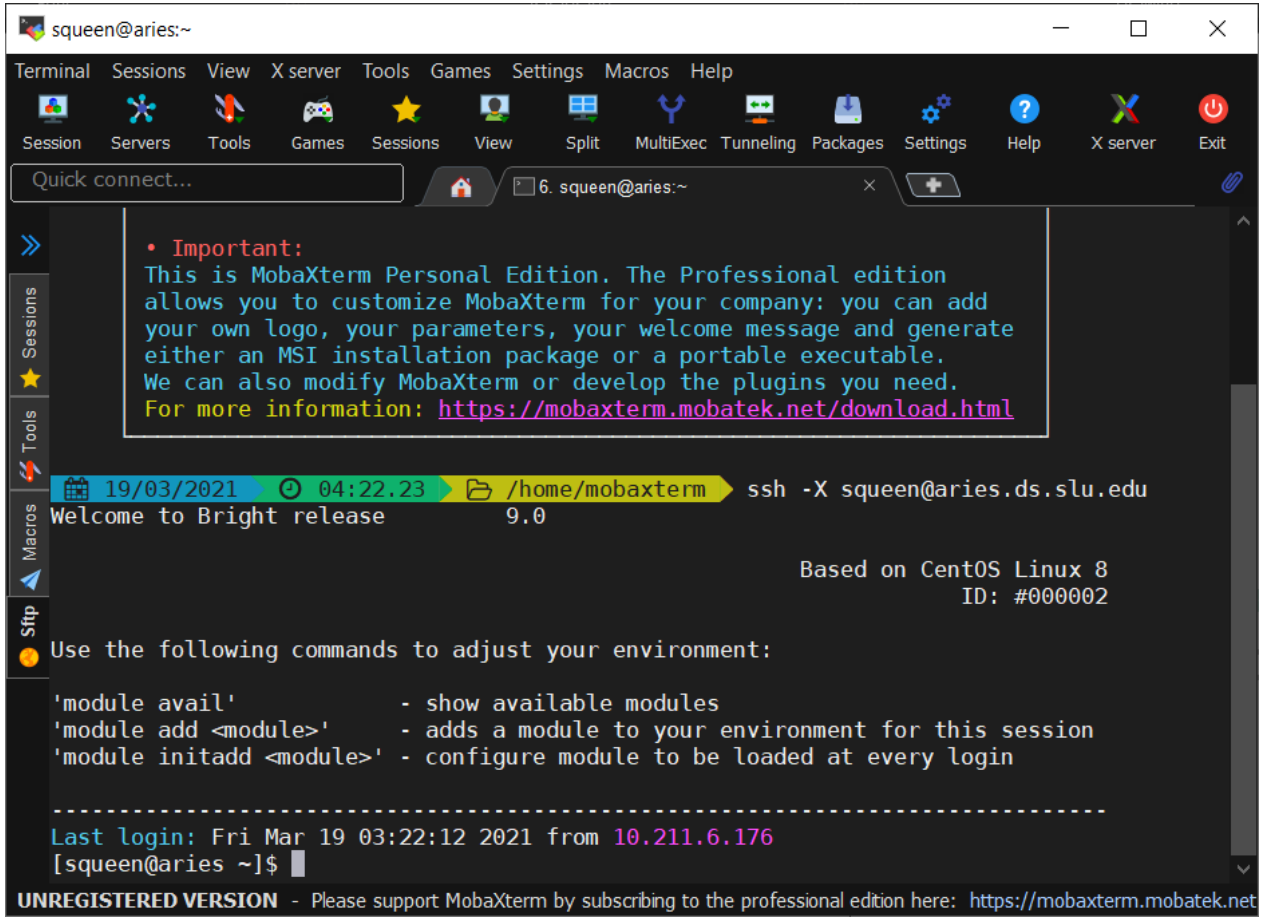
  $ srun -N3 --reservation=csci4850 -l /bin/hostname

Logging into the Cluster

- Open local terminal window
  - "terminal" on Linux and OSX
  - MobaXTerm or WSL on Windows

```
$ ssh -X user.name@165.134.34.113
```

  - You will be prompted for your password
  - When entering a password, "backspace" will not function



General usage guidelines

- Do not run jobs on the head node
  - *Don't load the head node!*
  - This can reduce the entire cluster's performance
- Manage file spaces accordingly
  - Different file spaces are meant for different functions (job I/O, archive, etc.)
  - Using the wrong file space can:
    - Cause you to run out of available space
    - Cause your jobs to run more slowly
  - More details in following slides

Directory organization

- Home space (/home)
  - Relatively small available space
  - used to store small files (job submission scripts)
- Project space (/data/projects)
  - used for archiving projects and files
  - Files should be compressed before being moved
- Archive space (/data/archive)
  - Includes files transferred from Apex /home and XFS2
- Scratch space (/scratch) **coming soon*

Transferring files using SCP

- "SCP" stands for "Secure copy"
- Local-to-remote:

```
scp local_file user@host:/path/to/file
```

```
$ scp Downloads/my_script.sh squeen@165.134.34.113:/home/squeen/
```

- Remote-to-local:

```
scp user@host:/path/to/file /local/path/to/file
```

```
$ scp squeen@165.134.34.113:/data/projects/squeen/project_output.gz Downloads/
```

Rsync

- A more robust alternative to SCP for large transfers
  - Allows transfers to stop and resume
  - Handles symlinks effectively
- Local-to-remote:

```
rsync -avzh local_file user@host:/path/to/file
```

```
$ rsync -avzh my_script.sh squeen@165.134.34.113:/home/squeen/
```

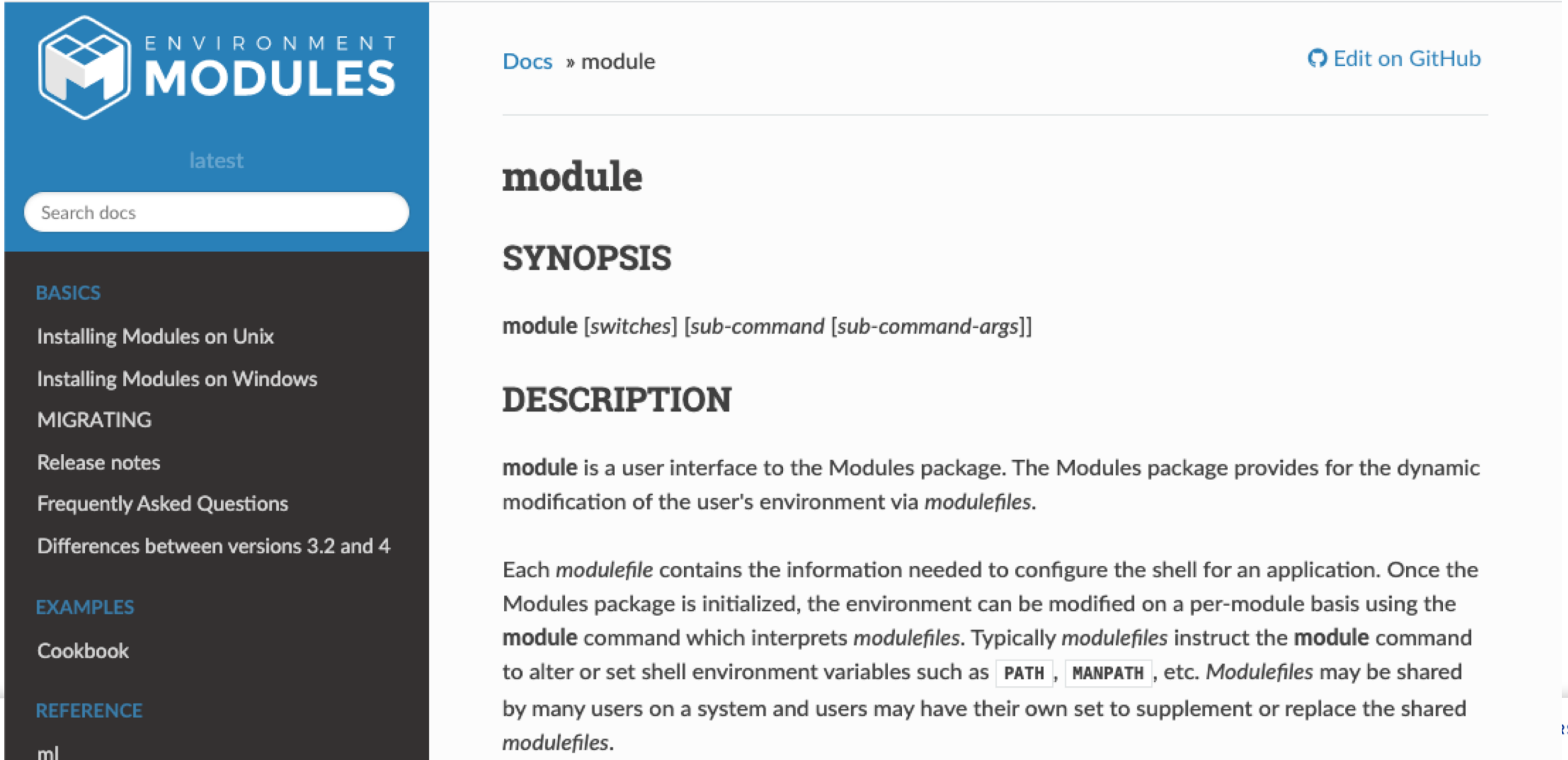
- Remote-to-local:

```
rsync -avzh user@host:/path/to/file /local/path/to/file
```

```
$ rsync -avzh squeen@165.134.34.113:/home/squeen/project_output.gz Downloads/
```

module

- https://modules.readthedocs.io/en/latest/module.html#synopsis



Available software

- $ module list

```
[ahnt@apex lab]$ module list
Currently Loaded Modulefiles:
 1) gcc/8.2.0   2) conda/3.6   3) python/3.6.1   4) slurm/18.08.8
```

- $ module avail

```
[ahnt@apex lab]$ module avail
------------------------------------------------------------ /cm/local/modulefiles ------------------------------------------------------------
cluster-tools-dell/8.2  cm-cloud-copy/8.2  cm-setup/8.2  cmsh   dot            gcc/8.2.0       lua/5.3.5   module-info  openldap  python36  shared
cluster-tools/8.2       cm-scale/8.2       cmd           cmsub  freeipmi/1.6.2  ipmitool/1.8.18  module-git  null         python2   python37

------------------------------------------------------------ /cm/shared/modulefiles ------------------------------------------------------------
abaqus/6.14-1                   default-environment                  lammps/lammps              openmpi/gcc/64/1.10.7          xplor-nih/2.48
abaqus/2020                     fastx/0.0.14                         lammps/lammps-20190605     openmpi/gcc/64/2.1.0
amber/amber16                   fftw2/openmpi/gcc/64/double/2.1.5    lammps/lammps_61118        orca/3.0.3
amber/amber18                   fftw2/openmpi/gcc/64/float/2.1.5     lapack/gcc/64/3.8.0        orca/4.0.0.2
amber/current                   fftw3/openmpi/gcc/64/3.3.8           matlab/R2016a              orca/4.1.1
blacs/openmpi/gcc/64/1.1patch03 freesurfer/5.3.0                     matlab/R2016b              python/3.6.1
blas/gcc/64/3.8.0               gaussian/current                     matlab/R2017a              r/mro-3.3
bonnie++/1.97.3                 gaussian/g09                         matlab/R2018b              raxml/raxml
bwa/0.7.15                      gdb/8.2                              matlab/R2019a              samtools/1.4.1
cellranger-atac/1.2.0           globalarrays/openmpi/gcc/64/5.7      mpich/ge/gcc/64/3.3        sas/9.4
cellranger/3.1.0                gurobi/811                           multimech/multimech        scalapack/openmpi/gcc/64/2.0.2
cellranger/5.0.1                hdf5/1.10.1                          mvapich/gcc/64/1.2rc1      schrodinger/2018-1
cellranger/current              hdf5_18/1.8.20                       mvapich/open64/64/1.2rc1   schrodinger/2019-1
cm-pmix3/3.1.4                  hpl/2.2                              mvapich2/gcc/64/2.2b       sctetra/13
conda/2.7                       hwloc/1.11.11                        mvapich2/gcc/64/2.3        sdag/current
conda/3.6                       ICMPro/icm-3.9.1b                    mvapich2/open64/64/2.2b    sge/2011.11p1
conda/3.7.2                     intel-tbb-oss/ia32/2019_20191006oss  netcdf/gcc/64/4.6.1        slurm/18.08.8
conda/kirkpatrick               intel-tbb-oss/intel64/2019_20191006oss netperf/2.7.0            stata/16
conda/lebeau                    iozone/3_482                         openblas/dynamic(default)  waring/matlab_toolboxes
conda/microsoft_r_3.4.3_rstudio iperf/3.6                            openblas/dynamic/0.2.20    xplor-nih/2.44
[ahnt@apex lab]$
```

Useful commands

- Loading modules

```
$ module avail
$ module load <modulename>
$ module unload <modulename>
```

- Viewing SLURM information

```
$ squeue
```

  - Shows detailed partition information

```
$ sacct
```

  - Shows job state

- Checking file contents

```
$ cat <filename>
```

  - Outputs entire file content to screen

```
$ less <filename>
```

  - Allows scrolling through file contents, searching file, jumping to specific lines

Other useful commands

- tar
  - Used to group several files into a single file object
  - Often used with gzip to compress entire project folders
- gzip
  - Used to compress files to reduce space used and speed up file transfer
- find
  - Used to locate files based on name or other characteristics like age
- grep
  - Used to parse files for lines containing specific text
- history
  - Used to retrieve previously-used commands

Available software

- Anaconda3
- Blacs 1.1p3
- Blas 3.8.0
- Cellranger 6.0.2
- Cradle CFD 2021
- Ctffind 4.1.8
- Eman2 2.9
- Fftw3 3.3.8
- Gromacs2021
- Openblas 0.3.7
- Relion 3.1.1

- You can check this on the cluster with the following command:
  - **module avail**

Submitting a job: SLURM queue

- The SLURM queue allows jobs to:
  - Request several nodes' worth of resources
  - Run unsupervised for long amounts of time
- To submit a batch job, use the following command:

```
$ sbatch job_script.sh
```

- Batch script requires a header containing SBATCH parameters
  - Example script on next slide

Submitting a job: example batch script

```
#!/bin/bash
#SBATCH --job-name=test_job             # Job name
#SBATCH --ntasks=1                  # Run on a single CPU
#SBATCH --mem=1gb                       # Job memory request
#SBATCH --time=00:01:00                 # Time limit hrs:min:sec
#SBATCH --output=test_job_%j.log # Standard output and error log

Module load python
/home/squeen/python_test.py
```

Aries SLURM environment

- Currently only one queue available to users
  - More planned later
- Job requests can be resource-based and/or node-based
  - Several jobs requiring a single core can run on one node
  - Cannot exceed a single node's available memory
- Fair share policy enabled to balance user requests
  - Accounts for several factors when determining job priority
    - Age, size, queue, etc.

Interactive jobs

- For running I/O or computationally-intensive but short jobs
  - Compressing files
  - Modifying large files or large numbers of files
  - Interactive/GUI-based jobs

- Example: Interactive job on one node with X11 enabled:

```
$ srun -N1 --pty --x11 bash
```

Accessing the GUI

- Notice: This slide assumes you have X11 forwarding enabled
  - See slide 4
- Submit an interactive job to a node

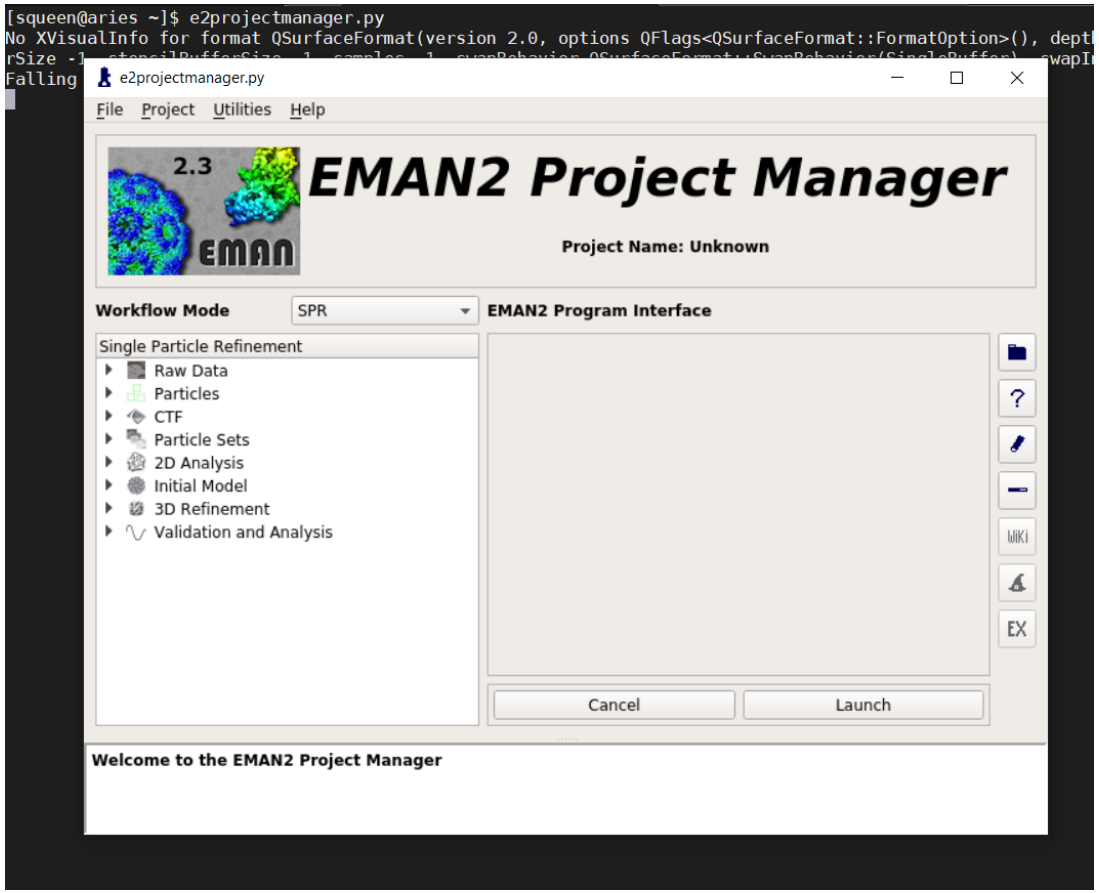
```
$ srun -N1 --pty --x11 bash
```

- Load the relevant module

```
$ module load eman2
```

- Run the command

```
$ e2projectmanager.py
```



Part 2: Intro to MPI

High-Performance Computing

Summer 2021 at GIST

Tae-Hyuk (Ted) Ahn
Department of Computer Science
Program of Bioinformatics and Computational Biology
Saint Louis University

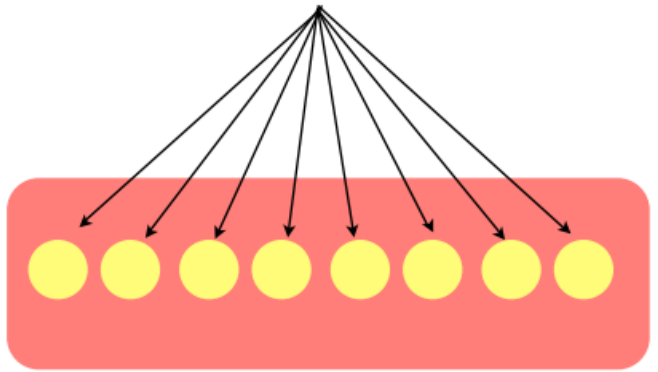
SAINT LOUIS
UNIVERSITY
EST. 1818

OpenMP Recap: Core Parallelism

- One thread per core.
- Shared-memory.
- Synchronization between cores:
  - Basic operations: fork, join.
  - Bulk-synchronous parallel (BSP).

OpenMP

```
#pragma omp parallel for
for(i=0;i<N;i++)
    x[i] += y[i]
```

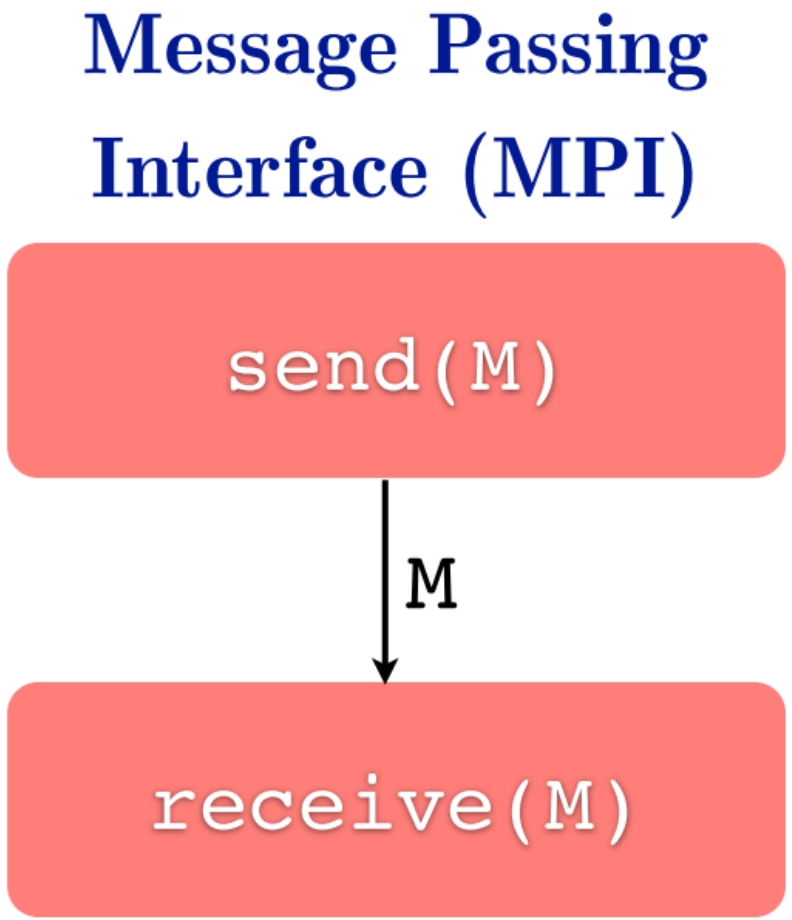


What is MPI?

- **M P I = Message Passing Interface**
- A *message-passing library specification*
  - extended message-passing model
  - not a language or compiler specification
  - not a specific implementation or product
- For parallel computers, clusters, and heterogeneous networks
- Full-featured
- Designed to provide access to advanced parallel hardware for end users, library writers, and tool developers

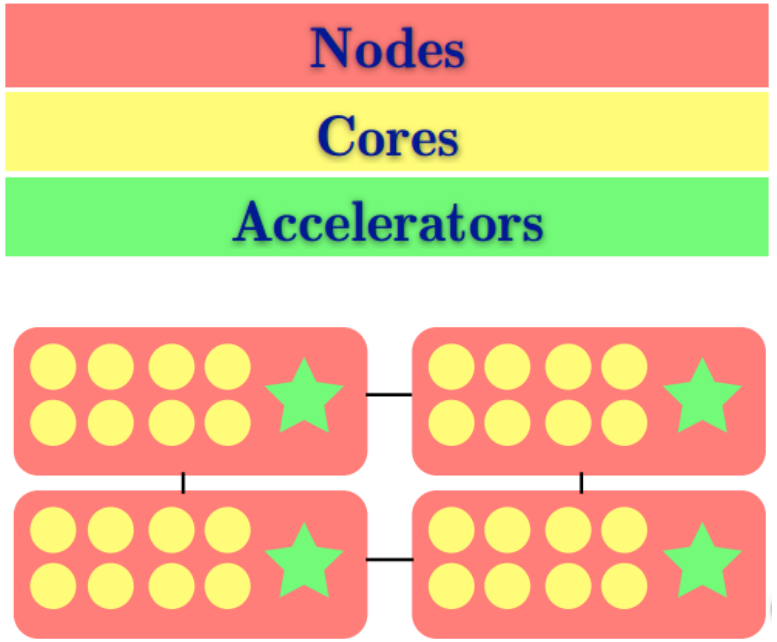
Node Parallelism

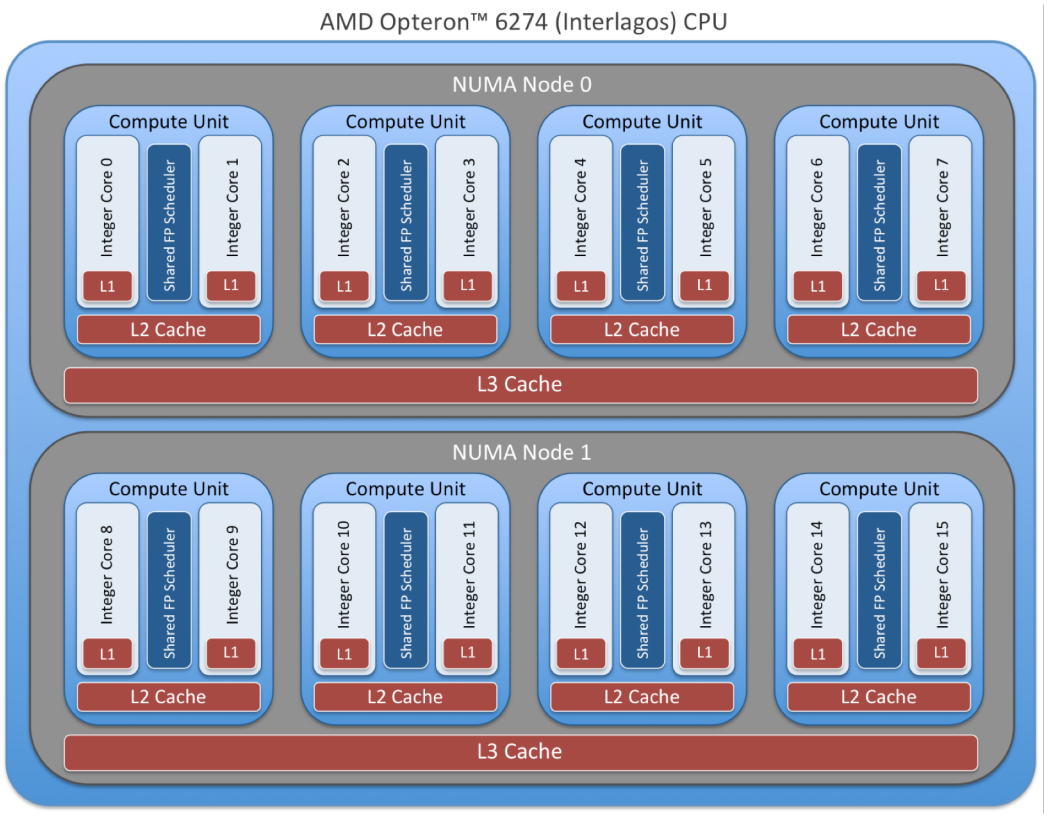
- One process per node.
- Private memory.
- Communication between nodes:
  - Message-passing.
  - Basic operations: send, receive.
  - Collective operations.



Keep in Mind the Current Trend of System

- 3-tier parallelism, hybrid architecture.





Message Passing Interface (MPI)

- Standard for operations in message passing.
- Led by MPI Forum (academia & industry).
  - MPI-1 standard (1994)
  - MPI-2 standard (1997)
  - MPI-3 standard (2012)
- Implementations:
  - Open-source: MPICH, OpenMPI, MVAPICH, mpi4py, Intel MPI
  - Proprietary: Cray, IBM.
  - Languages: C/C++, Fortran, Python, Perl, Ruby, R, Java, CL, Haskell.
- There is no automatic sequential-equivalent to an MPI program.

Message-Passing Paradigm

- A parallel program is decomposed into processes, called ranks.
- Each rank holds a portion of the program's data into its private memory.
- Communication among ranks is made explicit through messages.
- Channels honor first-in-first-out (FIFO) ordering.

- Message passing is for communication among processes, which have separate address spaces (vs share addr.)

Single-Program Multiple-Data (SPMD)

- All processes run the same program, each accesses a different portion of data.
- All processes are launched simultaneously.
- Communication:
  - Point-to-point messages.
  - Collective communication operations.
- Inter-process communication consists of
  - synchronization
  - Interaction: movement of data from one process's address space to another's;
- SPMD (VS SIMD, MIMD)
  - processors run a personal copy of a program

Features of Message Passing

- *Simplicity*: the basics of the paradigm are traditional communication operations.
- *Generality*: can be implemented on most parallel architectures.
- *Performance*: the implementation can match the underlying hardware.
- *Scalability*: the same program can be deployed on larger systems.

Important Features of MPI

- Communicators encapsulate communication spaces for library safety
- Datatypes reduce copying costs and permit heterogeneity
- Multiple communication modes allow precise buffer management
- Extensive collective operations for scalable global communication
- Process topologies permit efficient process placement, user views of process layout
- Good interface encourages portable tools

MPI is not very difficult

- Many parallel programs can be written using just these six functions, only two of which are non-trivial:
  - **`MPI_INIT`**
  - **`MPI_FINALIZE`**
  - **`MPI_COMM_SIZE`**
  - **`MPI_COMM_RANK`**
  - **`MPI_SEND`**
  - **`MPI_RECV`**

OpenMPI

- https://www.open-mpi.org/

Open MPI:
Open Source High Performance Computing

Home | Support | FAQ | Search

About
- Presentations
- Open MPI Team
- FAQ
- Videos
- Performance

Open MPI Software
- Download
- Documentation
- Source Code Access
- Bug Tracking
- Regression Testing
- Version Information

Sub-Projects
- Hardware Locality
- Network Locality
- MPI Testing Tool
- Open MPI User Docs
- Open Tool for Parameter Optimization

Community
- Mailing Lists
- Getting Help/Support
- Contribute
- Contact
- License

A High Performance Message Passing Library

The Open MPI Project is an open source Message Passing Interface implementation that is developed and maintained by a consortium of academic, research, and industry partners. Open MPI is therefore able to combine the expertise, technologies, and resources from all across the High Performance Computing community in order to build the best MPI library available. Open MPI offers advantages for system and software vendors, application developers and computer science researchers.

Features implemented or in short-term development for Open MPI include:

- Full MPI-3.1 standards conformance
- Thread safety and concurrency
- Dynamic process spawning
- Network and process fault tolerance
- Support network heterogeneity
- Single library supports all networks
- Run-time instrumentation
- Many job schedulers supported
- Many OS's supported (32 and 64 bit)
- Production quality software
- High performance on all platforms
- Portable and maintainable
- Tunable by installers and end-users
- Component-based design, documented APIs
- Active, responsive mailing list
- Open source license based on the BSD license

Open MPI is developed in a true open source fashion by a consortium of research, academic, and industry partners. The Open MPI Team page has a comprehensive listing of all contributors and active members.

See the FAQ page for more technical information

Join the mailing lists

Open MPI v4.1.0 released

Bug fix release

> Read more

Open MPI v4.0.5 released

Bug fix release

> Read more

hwloc 2.4.0

Major release

> Read more

Open MPI is an Associated Project of the Software in the Public Interest non-profit organization

Contact the Open MPI webmaster

Page last modified: 22-Feb-2021
©2004-2021 The Open MPI Project

OpenMPI

- **General Information:**
  - Open MPI is a thread-safe, open source MPI implementation developed and supported by a consortium of academic, research, and industry partners.
  - Available on our GIST VMs.
- **Compiling: mpic++ (for C++), mpicc (for C), mpif90 (for Fortran)**
- **Running:**
  - Be sure to load the same Open MPI module that you used to build your executable. If you are running a batch job, you will need to load the module in your batch script.
  - Launching an Open MPI job can be done using the following commands. For example, to run a 12 process MPI job:
    - **$ mpirun -np 12 a.out**
    - **$ mpiexec -np 12 a.out**
- **Documentation:**
  - Open MPI home page: http://www.open-mpi.org/

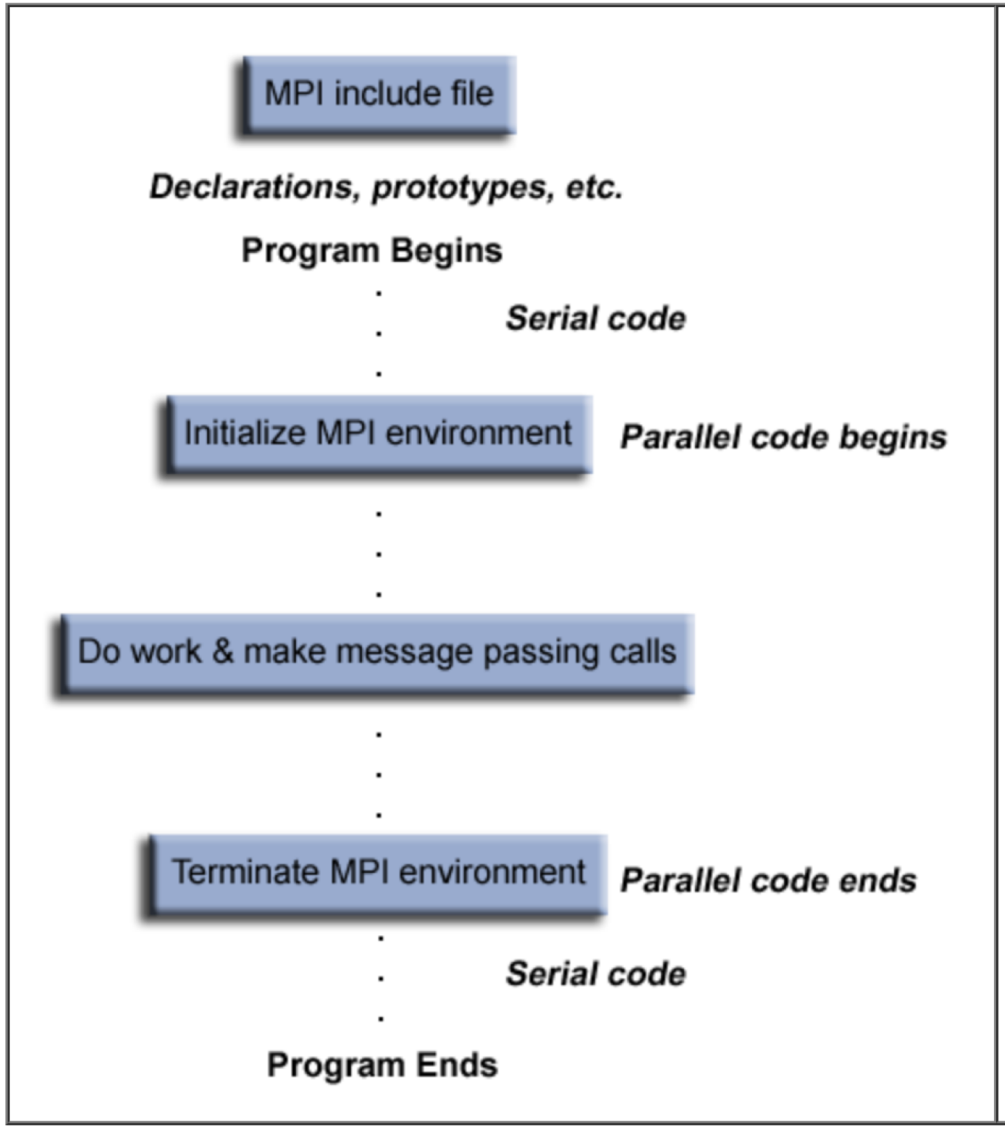
MPI Distributions

- There are numerous MPI distributions available!
- Check: MPI standard page.
- More information from the developers of each MPI distribution can be found here:
  - MVAPICH2
  - OpenMPI
  - MPICH
  - Intel MPI
- In our labs, we will use OpenMPI.

MPICH vs OpenMPI or MVAPICH2

- Advances in compiler technology and MPI libraries are starting to allow building single executables optimized for multiple CPU architectures and running over multiple networks.
- Recommendations? Not easy!
  - https://stackoverflow.com/questions/2427399/mpich-vs-openmpi
  - https://www.chpc.utah.edu/documentation/software/mpilibraries.php
  - https://userpages.umbc.edu/~gobbert/papers/BlasbergGobbertHPCF20087.pdf

General MPI Program Structure




```c
#include "mpi.h"
#include <stdio.h>
#include <stdlib.h>

int main (int argc, char *argv[])
{
int numtasks, rank, dest, source, rc, count, tag=1;
char inmsg, outmsg='x';
MPI_Status Stat;

MPI_Init(&argc,&argv);
MPI_Comm_size(MPI_COMM_WORLD, &numtasks);
MPI_Comm_rank(MPI_COMM_WORLD, &rank);

if (rank == 0) {
  dest = 1;
  source = 1;
  rc = MPI_Send(&outmsg, 1, MPI_CHAR, dest, tag, MPI_COMM_WORLD);
  rc = MPI_Recv(&inmsg, 1, MPI_CHAR, source, tag, MPI_COMM_WORLD, &Stat);
  }

else if (rank == 1) {
  dest = 0;
  source = 0;
  rc = MPI_Recv(&inmsg, 1, MPI_CHAR, source, tag, MPI_COMM_WORLD, &Stat);
  rc = MPI_Send(&outmsg, 1, MPI_CHAR, dest, tag, MPI_COMM_WORLD);
  }

MPI_Finalize();
}
```

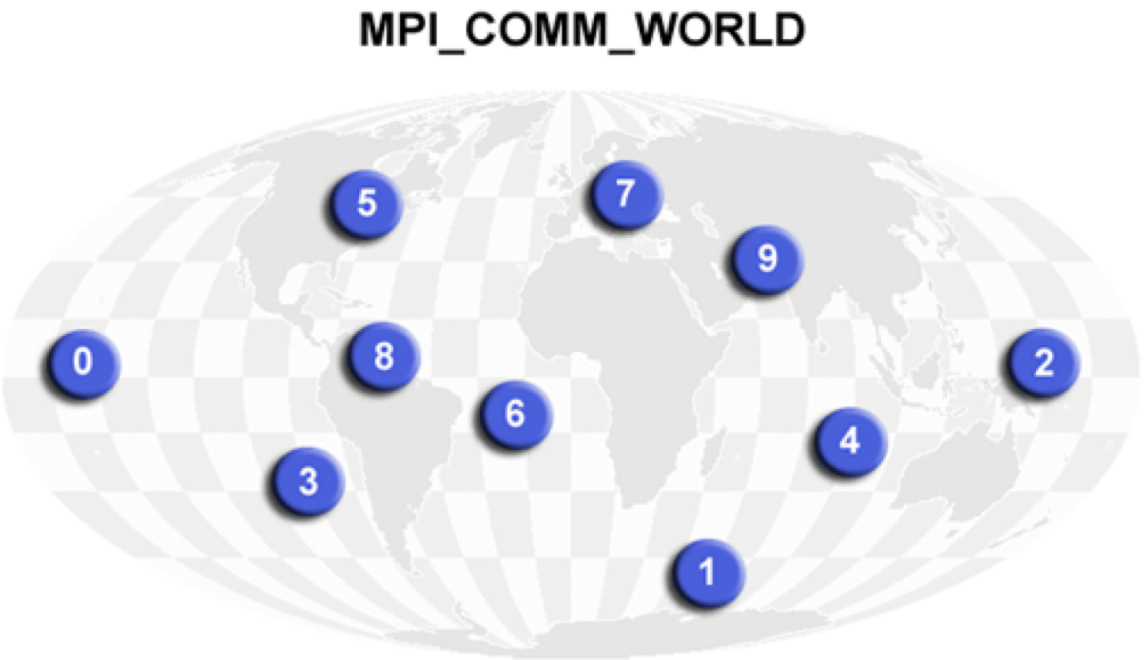
Header File

- Required for all programs that make MPI library calls

C include file	Fortran include file
`#include "mpi.h"`	`include 'mpif.h'`

Basic concepts: communicators

- MPI uses objects called communicators and groups to define which collection of processes may communicate with each other.
- Most MPI routines require you to specify a communicator as an argument.
- Communicators and groups will be covered in more detail later.
- For now, simply use **MPI_COMM_WORLD (MPI::COMM_WORLD)** whenever a communicator is required - it is the predefined communicator that includes all of your MPI processes.



helloworld_mpi.cpp

- Get connect to our VM machine.
- Open an editor and write this helloworld_mpi.cpp code.

helloworld_mpi.cpp

```cpp
#include <iostream>
#include <mpi.h>        // MPI header file
#define MASTER 0

using namespace std;

int main(int argc, char **argv) {

  // initialize for MPI (should come before any other calls to MPI routines)
  MPI_Init(&argc, &argv);   // C style
  // MPI::Init(argc, argv);    // C++ style

  // get number of processes
  int nprocs;
  MPI_Comm_size(MPI_COMM_WORLD, &nprocs);   // C style
  // nprocs = MPI::COMM_WORLD.Get_size();      // C++ style

  // get the rank = this process's number (ranges from 0 to nprocs - 1)
  int rank;
  MPI_Comm_rank(MPI_COMM_WORLD, &rank);     // C style
  // rank = MPI::COMM_WORLD.Get_rank();        // C++ style

  // print a greeting
  cout << "Hello from process " << rank << " of " << nprocs << endl;

  // clean up for MPI
  MPI_Finalize();   // C style
  // MPI::Finalize();  // C++ style

  return 0;
}
```

helloworld_mpi.cpp

```
ai@ubuntu-20-04:~/Lab/MPI$ g++ helloworld_mpi.cpp -o helloworld_mpi
helloworld_mpi.cpp:2:10: fatal error: mpi.h: No such file or directory
    2 | #include <mpi.h>  // MPI header file
      |          ^~~~~~~
compilation terminated.
ai@ubuntu-20-04:~/Lab/MPI$ mpic++ helloworld_mpi.cpp -o helloworld_mpi
ai@ubuntu-20-04:~/Lab/MPI$
ai@ubuntu-20-04:~/Lab/MPI$ ./helloworld_mpi
Hello from process 0 of 1
ai@ubuntu-20-04:~/Lab/MPI$
ai@ubuntu-20-04:~/Lab/MPI$ mpirun -np 4 ./helloworld_mpi
Hello from process 0 of 4
Hello from process 1 of 4
Hello from process 2 of 4
Hello from process 3 of 4
```

helloworld_mpi.cpp

- Test C syntax on CPP code.
- Test mpiexec.
- Test –np using different numbers.