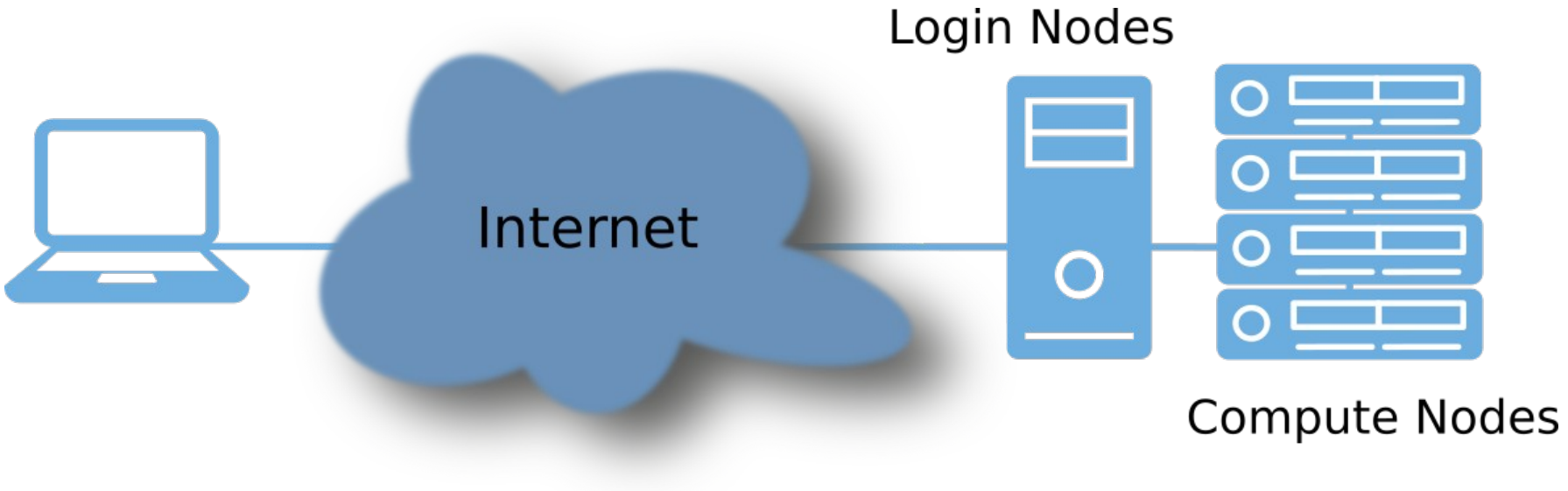
Introduction to LRZ HPC System

- Remote work via Secure Shell (SSH)
- Bash (command-line programming, tools)
- Environment Modules
- Slurm (Job Management, Resources and Scheduling)
- MPI (message passing user’s point of view)
- Git (version control system)

LRZ Cluster Overview



lxlogin?.lrz.de

SSH - Drills

1. SSH Keys + Connect
2. SSH Tunnel + VNC

lxlogin1.lrz.de, lxlogin2.lrz.de, lxlogin3.lrz.de, lxlogin4.lrz.de

Linux/Bash

- Linux Commands Overview
- Search/Find (files)
- Editors (for remote work)
- Bash (Basic, Advanced, GNU Bash Manual)
- Getting fast help/reminder (man, cmd-line)

Linux/Bash - Drills

1. Create a directory! Change to it! Check your current directory (present working directory)!
   a) Create a file! Enter some text to it!
   b) Remove/delete everything above (challenge: with one call)!
2. Change to HOME (several variants)!
3. Change to /lrz/sys/applications/OpenFOAM/OpenFOAM-v2112/OpenFOAM-v2112 (use <Tab> for auto-complete)!
   a) “Find“ files named “**`*bashrc*`**“ !
   b) “Grep“ for compiler (case-insensitive) in etc/bashrc!

Linux/Bash

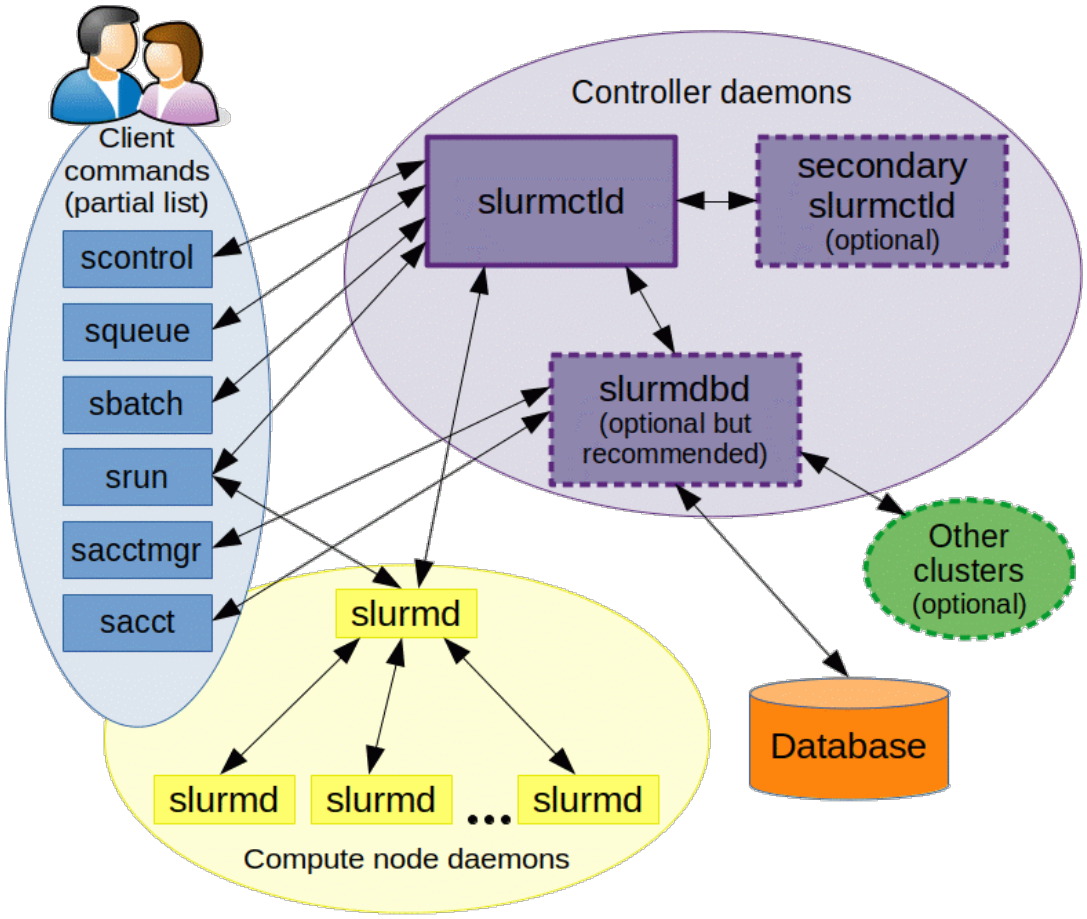
Bash Programming – Demo

- environment – **`env`**/**`printenv`**, **`export`**, **`unset`**
- aliases
- workflow – **`if`** (test), **`for`**
- bash functions – **`declare`**
- **`awk`**/**`sed`** (outline; + piping,I/O redirection)
- editors (**`vi`**, **`nano`**, … EDITOR env variable)
  (example: emacs + SSH remote)

Environment Modules - Drills

1. Look for module short help!
2. Check currently load modules!
3. Search for installed OpenFOAM packages! Load one if found!
   Check the environment! Unload and re-check the environment!
4. Add /lrz/sys/share/modules/extfiles/ to module search path!
5. Load/unload some module, and create a module collection!
   Check by logging out and in, and restore the collection!
   (Careful! Pitfalls ☹ ~/.module/<coll-name>)

Slurm


Client commands (partial list)
scontrol
squeue
sbatch
srun
sacctmgr
sacct
Controller daemons
slurmctld
secondary slurmctld (optional)
slurmdbd (optional but recommended)
Other clusters (optional)
Database
slurmd
slurmd
slurmd
slurmd
Compute node daemons

Slurm - Drills

1. Get an overview of the Clusters! (**`sinfo`**; Patience!)
2. Check for idle nodes on cm2_tiny/inter/serial cluster!
   a) Create an interactive session! (**`salloc`**; max 1 node, 5 min!!)
   b) Check status of your allocation! (**`squeue`**; --start; in different terminal)
   c) Cancel the job once it is running! (**`scancel`**)
   d) Check the jobs status! (**`sacct`**)
3. Run a in-situ non-interactive job with **`hostname`**! (**`srun`**)

Slurm - Drills

4. Create Slurm job script and submit it (**`sbatch`**)
   a) Check its status! (**`squeue`**; --start)
   b) Cancel the job! (**`scancel`**) Resubmit!
   c) Check the jobs status after finished! (**`sacct`**)

LRZ Linux Cluster Doku: Parellel Jobs

Slurm Job Scripts

Minimum Slurm job script

```
#!/bin/bash
#SBATCH -J job_name                  # job's name
#SBATCH -o ./%x.%j.%N.out            # output file stdout/err
#SBATCH -D ./                        # work dir == submit dir
#SBATCH --clusters=cm2_tiny          # which cluster?
#SBATCH --partition=cm2_tiny         # which partition?
#SBATCH --mail-type=none             # others possible
#SBATCH --export=NONE                # mandatory!!!!
#SBATCH --nodes=1                    # resource (1 node)
#SBATCH --time=00:05:00              # resource (5 minutes)

module load slurm_setup              # mandatory @ LRZ

…                                    # your workflow
```

Slurm Job Scripts

Other relevant options

```
#SBATCH --ntasks=10                  # resource (how many ranks?)

#SBATCH --ntasks-per-node=28         # resource (ranks per node?)
#SBATCH --cpus-per-task=1            # resource (CPUs/rank?)

#SBATCH --mem=…                      # resource (memory/shared node)

#SBATCH --reservation=<resv-name>
```

Slurm Job Scripts

Slurm job script content example

```
#!/bin/bash
…                                       # SBATCH header

module load slurm_setup                 # mandatory @ LRZ

echo "Start: $(date "+%F %H:%M:%S")"
env | sort
echo "------------------------------------"
cat /proc/cpuinfo
echo "------------------------------------"
cat /proc/meminfo
echo "Stop:  $(date "+%F %H:%M:%S")"
```

Slurm Job Scripts

Workflow can be nearly anything:

- bash/python/R/… (at least as separate script)
- Modules can be loaded at will

- Test script as dummy (interactively – source; sbatch) before submitting for production!

MPI (How to use mpiexec)

Show case example (Intel MPI, OpenMPI):

`/lrz/sys/tools/placement_test_2021/bin/`

Placement and pinning matter in HPC

Intel MPI/MPICH - Drills

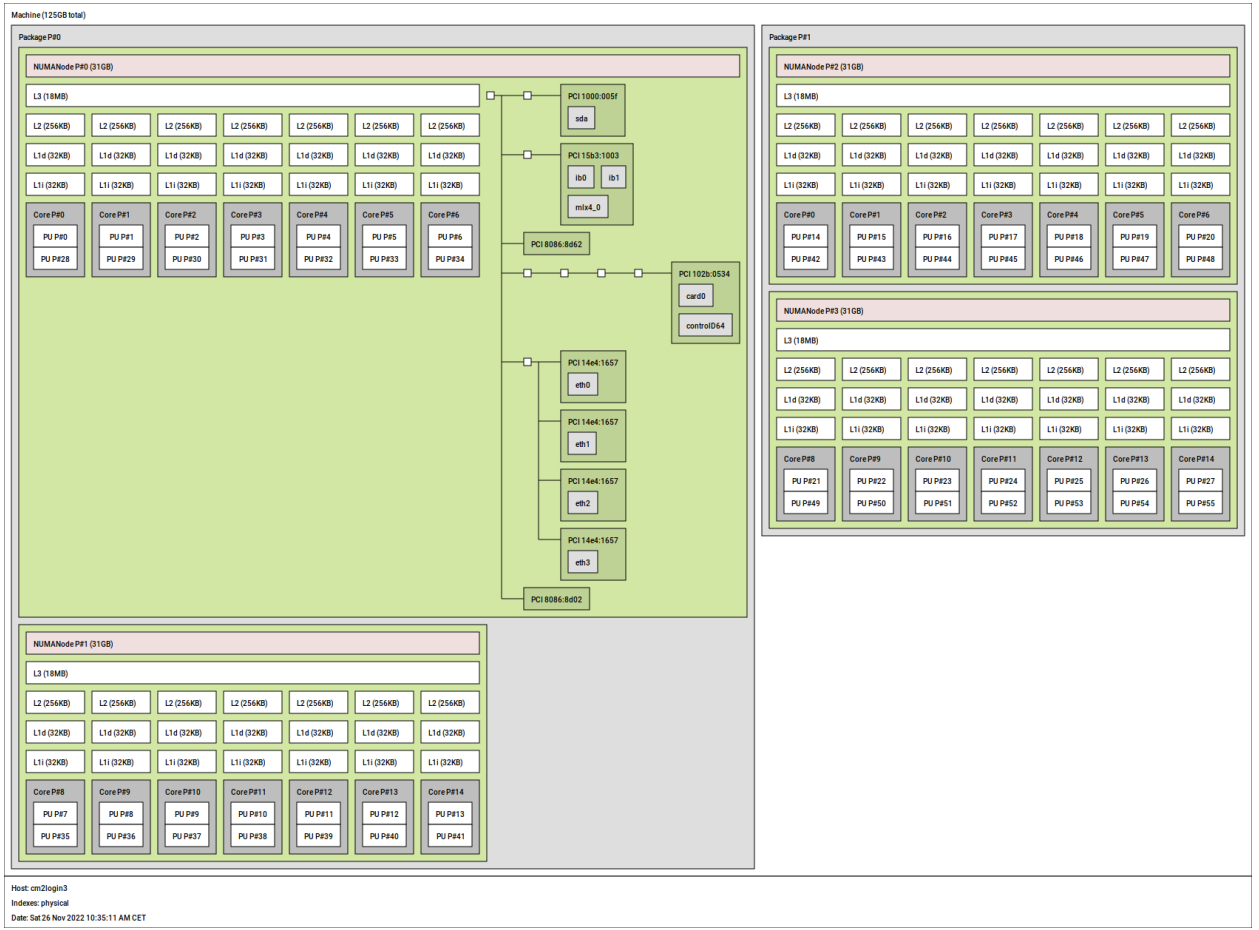
1. Execute **`hostname`** 4 times (login node)!
2. Execute **`hostname`** 4 times – 2 per node (localhost,cm2loginX)!
3. Same as 2. But this time with
   `placement-test.intel_impi -t 1`!
4. Repeat, but with debug information!
5. Placement test again with **`-t 28`**, **`mpiexec`** with **`-ppn 1`**
   (**`-t 14`**, **`-ppn 2`**), (**`-t 7`**, **`-ppn 4`**),
   (**`-t 1`**, **`-ppn 56`**), (**`-t 2`**, **`-ppn 28`**)

Intel MPI - Drills

If ***hwloc*** is installed:

`lstopo topo.png`
or
`lstopo topo.pdf`

(**`lstopo -h`** for
available output formats)



Open MPI - Drills

Repeat with OpenMPI! (Docu) (Better use a **`salloc`** allocation)

Warning! Placement/pinning of ranks/OpenMP threads more difficult!!

Use **`placement-test.gcc_ompi`** with **`openmpi/4.1.2-gcc11`** module loaded.

Example:

```
OMP_PLACES=cores mpiexec -n 4 --map-by ppr:1:numa --bind-to numa \
/lrz/sys/tools/placement_test_2021/bin/placement-test.gcc_ompi -t 7
```

or

```
OMP_NUM_THREADS=7 OMP_PLACES=cores mpiexec -n 4 \
  --map-by ppr:1:numa --bind-to numa             \
  /lrz/sys/tools/placement_test_2021/bin/placement-test.gcc_ompi
```

GIT (Version Control)

Intro Example:

- **`git init`**
- **`git status`**
- **`git add`**
- **`git commit`**
- **`git clone`**
- **`git log`**
- **`git checkout`**
- **`git config`**
- **`git tag`**
- **`git pull`**
- **`git push`**
- **`…`**

GIT - Drills

1. Check **`git --help`**!
2. Create a directory, **`cd`** into it!
3. Initialize git versioning for that directory! (**`init`**)
4. Create files, add them to the versioning, add/change files content, and make commits! (**`status`**, **`add`**, **`commit`**, **`log`**)
5. Exclude some new file from versioning (**`.gitignore`**)!
6. Clone that directory! (**`clone`**)
7. Go to an earlier commit stage! (**`checkout`**)