# Simple Linux Utility for Resource Management

# What is SLURM?

# SLURM

**Simple Linux Utility for Resource Management**

(SLURM)

- Open-source (GPL) project from Lawrence-Livermore National Laboratory

  www.llnl.gov/linux/slurm

- A light-weight, powerful infrastructure for managing a cluster of compute resources

# Where to get it:

http://www.schedmd.com/#repos

Slurm 2.4.4 stable release

Slurm 2.5.0-rc1 available ( release candidate 1)

- Slurm 2.4.4;
  - fault tolerant high-scalable cluster management and job sheduler
  - no kernel modifications and self-contained

  Key functions:
  - exclusive or non-exclusive access to resources  for users for some duration of time
  - it provides a framework for starting , executing and monitoring work  on a set of nodes
  - arbitrates contention for resources by managing a queue of pending work

# The Goals of SLURM

- A Cluster Resource Manager must:
  - Manage the state of the nodes (up/down, idle/in use)
  - Allocate nodes for (sometimes exclusive) use by users
  - Control job execution (start/run, monitor, and signal/cancel)
  - Manage contention by queuing up work if necessary

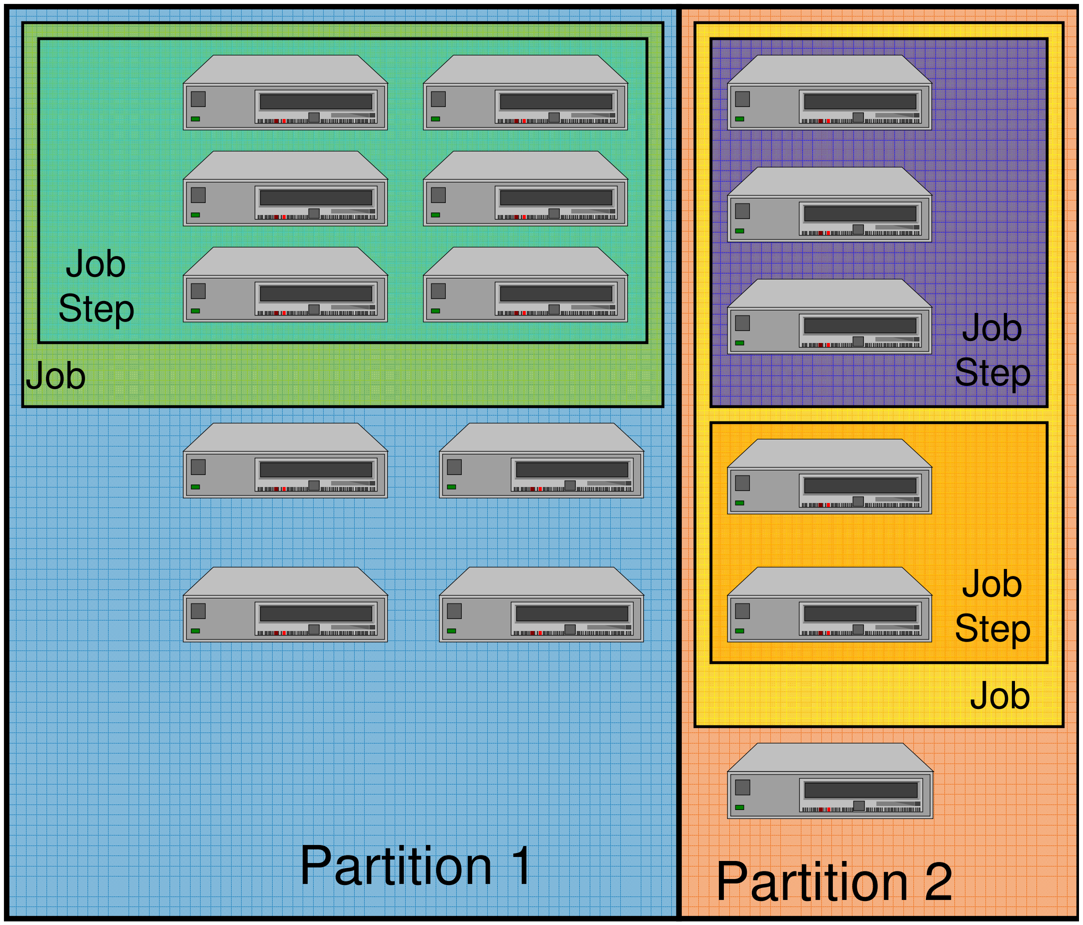
- Slurm :
  - Scalable to thousands of nodes
  - Portable to accommodate different OS, architecture, interconnect
  - Fault Tolerant for reliability within a cluster
  - Modular to work with other components ( HPC-LSF , … )
  - Secure
  - Simple to administer
  - One configuration file for all the nodes : slurm.conf

# The Non-Goals of SLURM

- NOT a comprehensive cluster administration or monitoring package
- NOT a sophisticated scheduling system
  - SLURM uses only one  builtin queue : FIFO
  - BUT supports plugins for
    - ACCOUNTING
    - ADVANCE RESERVATION
    - GANG SCHEDULING ( time sharing for parallel jobs)
    - BACKFILL SCHEDULING ( explained later)
    - TOPOLOGY OPTIMIZED resource selection

# SLURM Entities

- Nodes
- Partitions (group of nodes with similar characterstics)
- Jobs
- Job steps (set of tasks within a job)



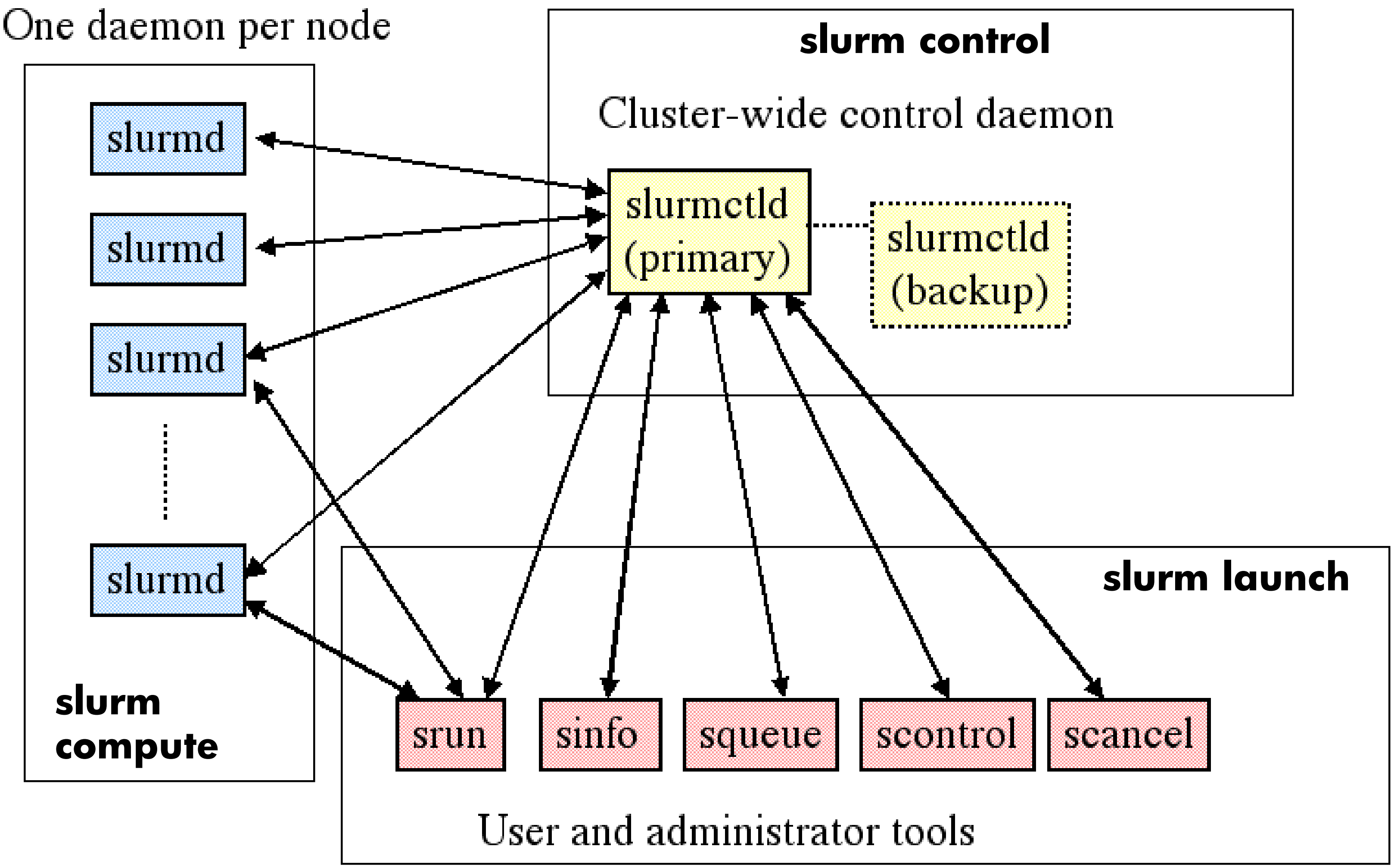
# Configuration files

- slurm.conf
  - Nodes Definition
  - Partition Definition
  - Scheduling Policies
  - Allocation Policies
  - Logging, Authentication, Accouting
- slurmdbd.conf
  - Type of persistent storage
  - Location of storage
- topology.conf
  - switch hierarchy

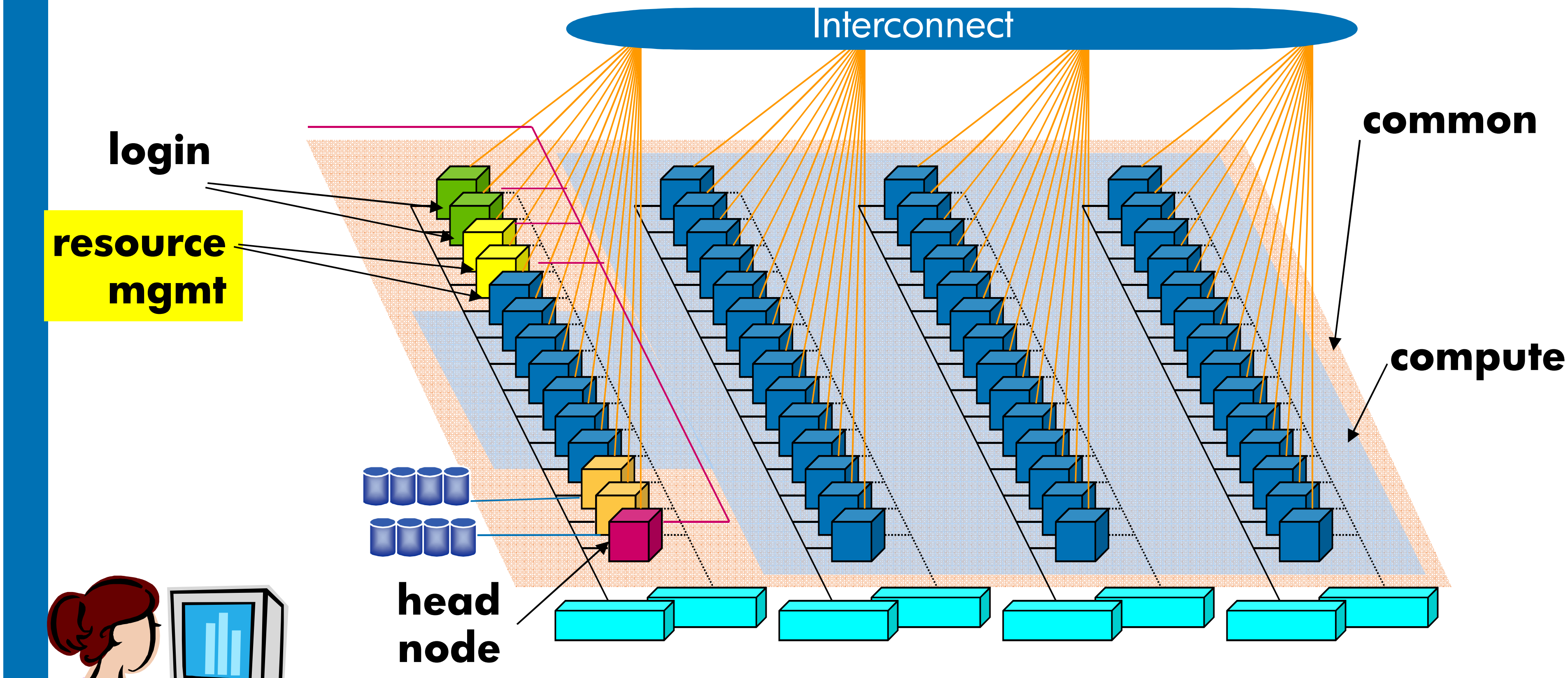
# SLURM Architecture

- Two daemons
  - slurmctld - controller, optional backup
  - slurmd - per node worker daemon
- Some user commands
  - sacct - display job accounting information
  - scancel - signal or cancel a job or job step
  - scontrol - administration tool, get/set configuration
  - sinfo - reports general system information
  - squeue - reports job and job step information
  - sview - graphical information viewer
  - srun - submit/initiate job or job step
  - sstat - show status of current running jobs

# SLURM Architecture



# Slurm control nodes



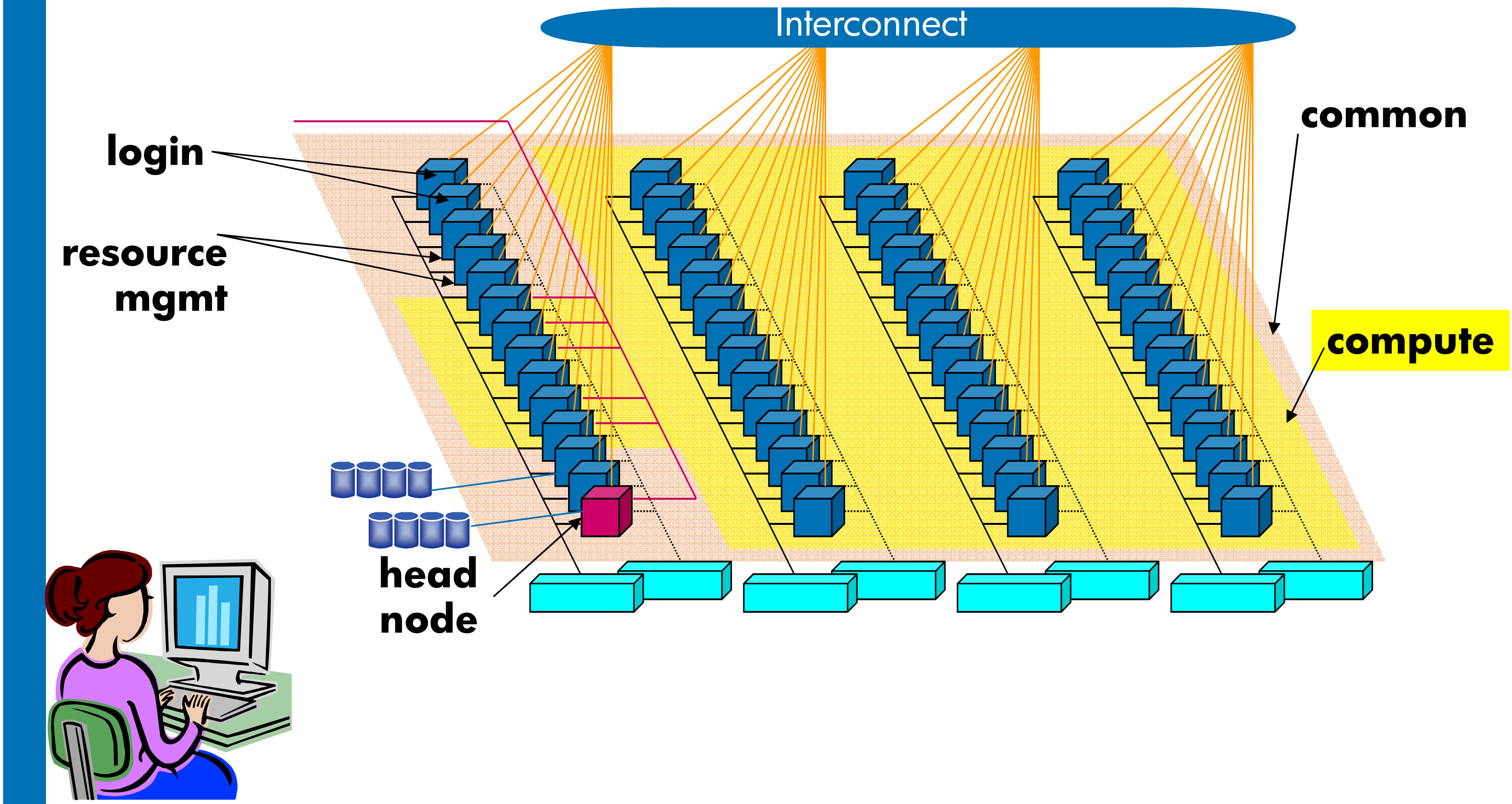
12/3/2012

# Slurmctld (control nodes)

- Orchestrates SLURM activities across entire cluster (with optional backup)

- Components
  - Job Manager - manages queue of pending jobs
  - Node Manager - node state information
  - Partition Manager - allocates nodes

# Slurm compute nodes



# slurmd (compute nodes)

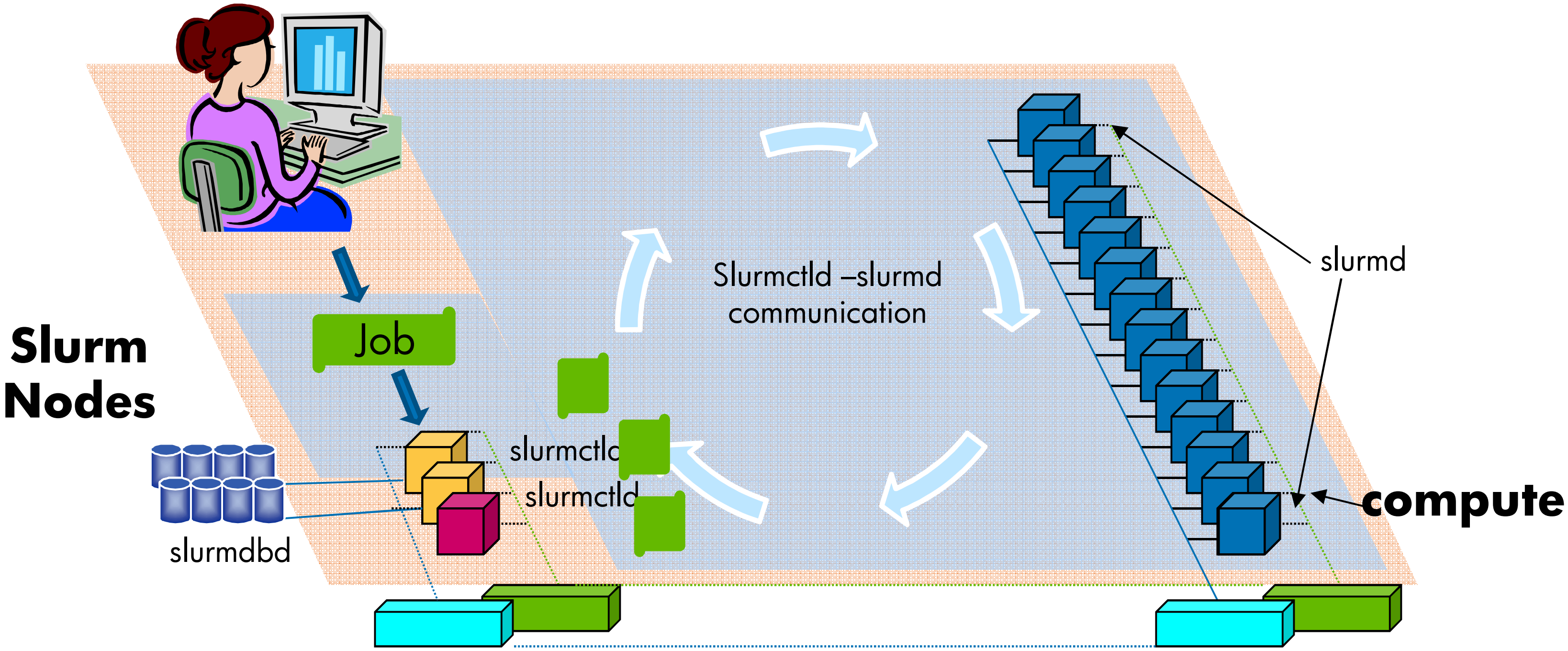
- Daemon running on each compute node
- Manages user jobs and job steps* within that node
- Components
  - Machine Status
  - Job Status
  - Remote Execution
  - Stream Copy (stdin, stdout, and stderr)
  - Job Control (signal)

* : slurmd spawns a ‘slurmstepd’ for each job step

# The way it works

User submits job « srun –N 3 –p Part app »

2.Slurmctld selects nodes based on inserted options



3.Slurmctld allocates CPUs from the nodes and distributes tasks

4.Slurmd remote execute the tasks and report the results

5. Nodes are released

# 2. Selection of nodes

- Options in slurm.conf
  - Nodename – defines a node and its resources, sockets, cores, threads,ec
  - PartitionName – defines a partition and its characteristics including the node set.

- srun/salloc/sbatch options:

  --partition, -P – specifies the partition from where to chose the nodes

  --nodelist - specifies the list of nodes from where the selection is made

  -N, --nodes – the number of nodes to be selected

- --sockets-per-node,--cores-per-socket, --threads-per-core – Select only the nodes with the specified characteristics

# 3a. Allocation of CPUs

- Options in slurm.conf
  - SelectType=select/linear = all the CPUs from one node are used and then one slurmctld passes to the next node.
  - SelectType=select/cons_res allows allocation of CPUs based on individual CPUs/cores.

- srun/salloc/sbatch options:

  --n, --ntasks = total tasks to be ran.

  --ntasks-per-node

# 3b. Distribution of Tasks on CPUs

- Options in slurm.conf
  - MaxTasksPerNode – the maximum number of tasks that one node can run.

- srun/salloc/sbatch options:

--m, --distrubution – controls the order in which the tasks are distributed to the nodes.

Distrubution:

block - The block distribution method will distribute tasks to a node such that consecutive tasks share a node.

cyclic - will distribute tasks to a node such that consecutive tasks are distributed over consecutive nodes (in a round-robin fashion).

# 3b. Distribution of Tasks on CPUs

## Block

srun —nodes=3 —ntasks=4
--distribution=block

| Node | cn01 | cn02 | cn03 |
|---|---|---|---|
| Allocated CPUS | 2 | 1 | 1 |
| Tasks per tasks ID | 0,1 | 2 | 3 |

```
srun  --nodes=3 --ntasks=4 --label --distribution=block  /bin/hostname
```

```
2: n301
0: n300
1: n300
3: n302
```

## Cyclic

srun —nodes=3 —ntasks=4
--distribution=cyclic

| Node | cn01 | cn02 | cn03 |
|---|---|---|---|
| Allocated CPUS | 2 | 1 | 1 |
| Tasks per tasks ID | 0,3 | 1 | 2 |

```
srun  --nodes=3 --ntasks=4 --label --distribution=cyclic  /bin/hostname
```

```
1: n301
2: n302
0: n300
3: n300
```

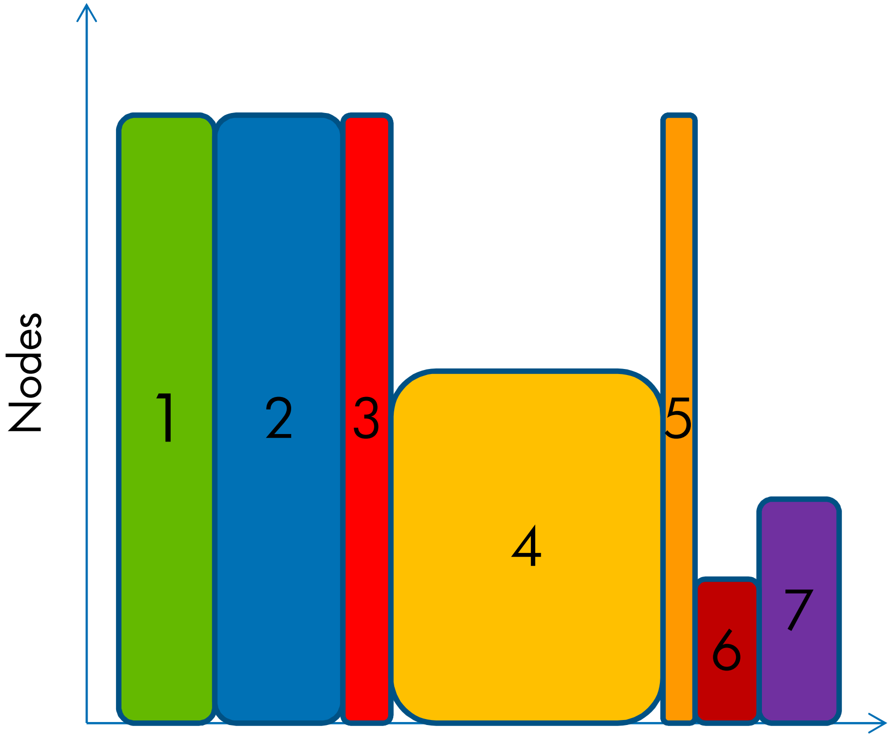
# Scheduling in Slurm

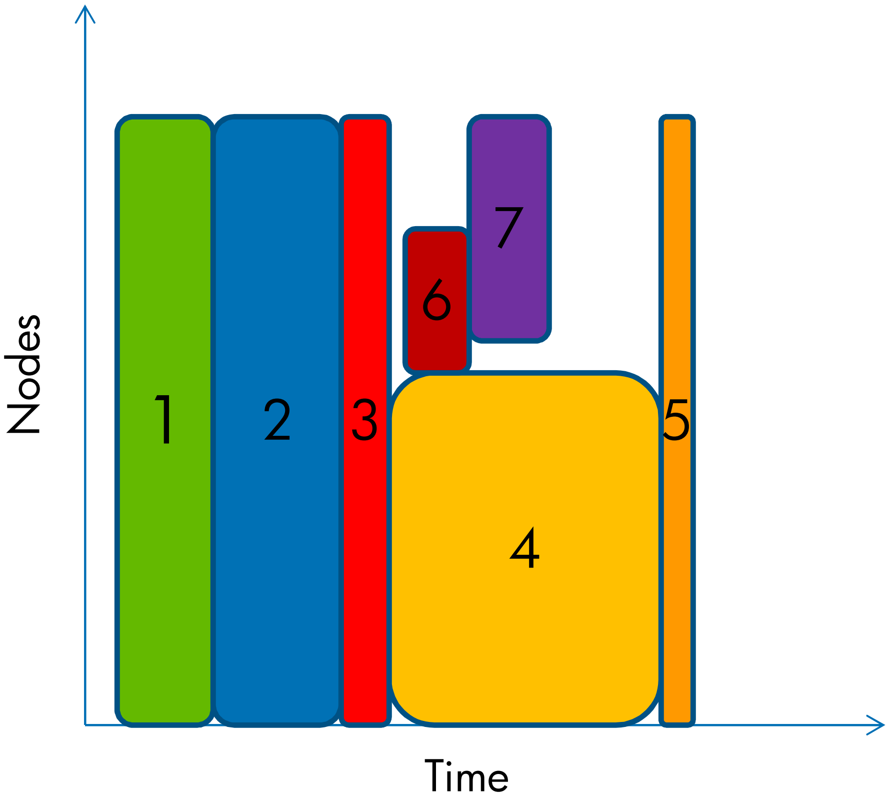
- Default Built in Scheduler: FIFO (First In First Out)

- Alternative:  Backfill
  - Increases the utilization of the cluster
  - Requires declaration of maximum time of execution of jobs ( --time when srun is used)
  - works as long as a job with higher priority is not delayed.

# Backfill

FIFO





BACKFILL

# Backfill example

```
srun -j C1 -N4 sleep 10
srun -j C2 -N1 –time=60 sleep 60
srun -j C3 -N4 –time=10 sleep 10
srun -j C4 -N2 –time=20 sleep 30
srun -j C5 -N3 –time=10 sleep 10
srun -j C6 –N1 –time=15 sleep 15
```

With Backfill

1. C1 Terminates
2. C2 Starts
3. C3 Pending, not enough nodes
4. C4 Backfills, limit less than C2
5. C5 Pending, can't backfill as not enough nodes
6. C6 Backfills, limit less than C2
7. C4 Terminates
8. C6 Terminates
9. C5 now backfills
10. C2 terminates
11. C3 waits for C5 to terminate.
12. C5's termnation still before C2's expected termination.

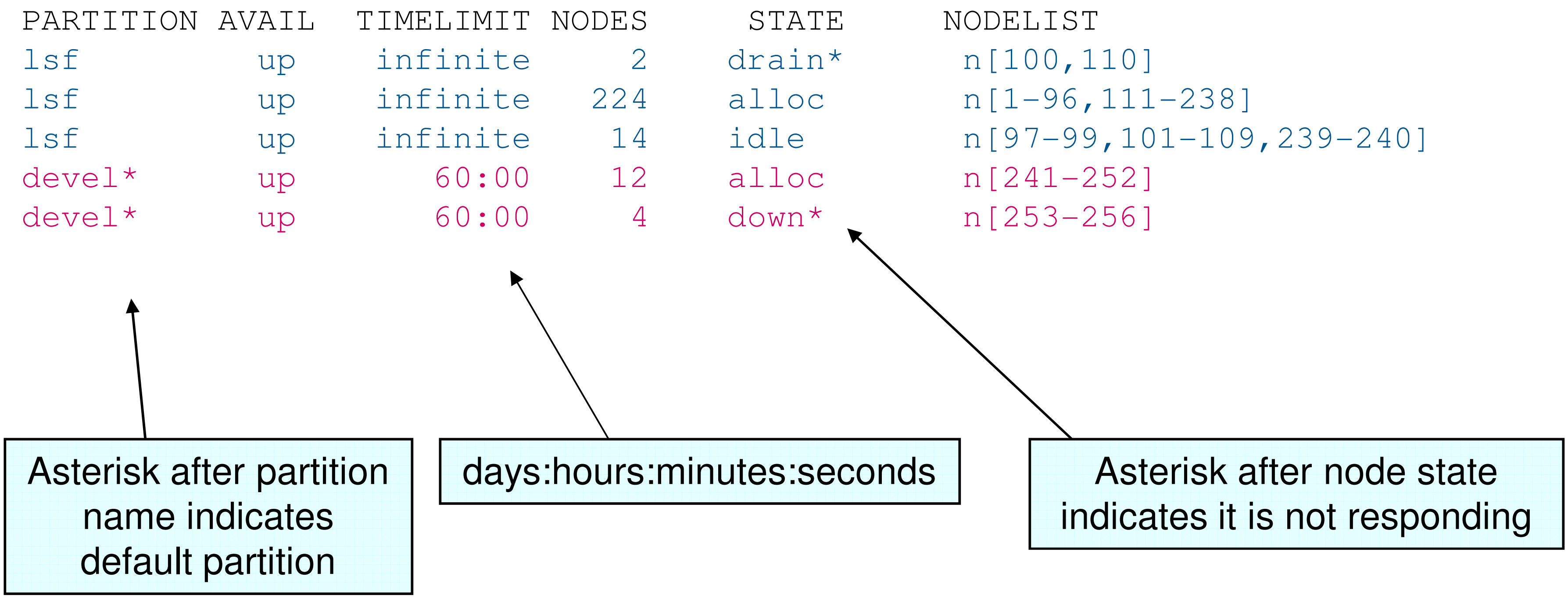
| Node | | | | | | | | | | | | | |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| 0 | C1 | C2 | C2 | C2 | C2 | C2 | C2 | C3 | C4 | C4 | C4 | C5 | |
| 1 | C1 | | | | | | | C3 | C4 | C4 | C4 | C5 | |
| 2 | C1 | | | | | | | C3 | | | | C5 | |
| 3 | C1 | | | | | | | C3 | | | | C6 | C6 |
| Time | 0:10 | 0:20 | 0:30 | 0:40 | 0:50 | 1:00 | 1:10 | 1:20 | 1:30 | 1:40 | 1:50 | 2:00 | 2:05 |

| Node | | | | | | | | |
|---|---|---|---|---|---|---|---|---|
| 0 | C1 | C2 | C2 | C2 | C2 | C2 | C2 | C3 |
| 1 | C1 | C4 | C4 | C4 | | | C5 | C3 |
| 2 | C1 | C4 | C4 | C4 | | | C5 | C3 |
| 3 | C1 | C6 | C6 | | | | C5 | C3 |
| Time | 0:10 | 0:20 | 0:30 | 0:40 | 0:50 | 1:00 | 1:10 | 1:20 |

# sinfo

- Displays node and partition information

[user@n260 ~]$ **sinfo**



```
PARTITION AVAIL  TIMELIMIT NODES     STATE      NODELIST
lsf          up    infinite     2    drain*      n[100,110]
lsf          up    infinite   224    alloc       n[1-96,111-238]
lsf          up    infinite    14    idle        n[97-99,101-109,239-240]
devel*       up       60:00    12    alloc       n[241-252]
devel*       up       60:00     4    down*       n[253-256]
```

Asterisk after partition name indicates default partition

days:hours:minutes:seconds

Asterisk after node state indicates it is not responding

# More sinfo

- Options permit you to filter, sort, and output information in almost any way desired

```
[root@n194 ~]# sinfo --long -node --exact

Mon Jan 23 15:54:35 2006
NODELIST      NODES PARTITION STATE CPUS MEMORY TMP_DISK WEIGHT FEATURES REASON
n[1-8,10-16]    15       lsf    idle    2    7633        1     1    himem,g1   none
n[9,17-64]      49       lsf    idle    2    3696        1     1    lomem,g1   none
n[65-192]      128       lsf    idle    4    7970        1     1    himem,g2   none
```

- Note the assigned Features, CPUs, and Memory
  - (DL145) g2 nodes have 2 dual-core processors

# squeue

- Displays job and job step information

```
[user@n260 ~]$ squeue
JOBID   PARTITION  NAME  USER  ST    TIME   NODES NODELIST
16306    lsf     xc1@37  brian  R   4:03:53  128  n[111-238]
16721    devel    fall   cheryl R     20:07    8  n[241-248]
```

```
[user@n260 ~]$ squeue -s
STEPID    PARTITION USER      TIME     NODELIST
16306.0   lsf       brian   4:05:54      n111
16306.1   lsf       brian   4:05:53      n[111-238]
16721.0   devel     cheryl    22:07      n[241-248]
16721.1   devel     cheryl    22:06      n[241-248]
16721.2   devel     cheryl    22:05      n[241-248]
```

# squeue - Job Step Example

```
[user@n260 ~]$ squeue -s
STEPID  PARTITION    USER      TIME  NODELIST
16000.0              lsf     alice   6:48:04  n1
16000.1              lsf     alice   6:48:03  n[1-96]
16306.0              lsf     brian   4:05:54  n111
16306.1              lsf     brian   4:05:53  n[111-238]
16721.0          devel    cheryl       22:07  n[241-248]
16721.1          devel    cheryl       22:06  n[241-248]
16721.2          devel    cheryl       22:05  n[241-248]
16745.0          devel     david        8:40  n[249-252]
```

Job 16721 has three active steps

days:hours:minutes:seconds

# squeue – Job status

CA CANCELLED Job was explicitly cancelled by the user or system administrator.

CD COMPLETED Job has terminated all processes on all nodes.

CF CONFIGURING Job has been allocated resources, but are waiting for them to become ready for use

CG COMPLETING Job is in the process of completing. Some processes on some nodes may still be active.

F FAILED Job terminated with non-zero exit code or other failure condition.

NF NODE_FAIL Job terminated due to failure of one or more allocated nodes.

PD PENDING Job is awaiting resource allocation.

PR PREEMPTED Job terminated due to preemption.

R RUNNING Job currently has an allocation.

S SUSPENDED Job has an allocation, but execution has been suspended.

TO TIMEOUT Job terminated upon reaching its time limit.

# srun

- User command to initiate jobs and job steps
  - Run jobs interactively
  - Allocate resources
  - Submit batch jobs
  - Attach to currently running job
  - Launch a set of parallel tasks (job step)

- Options to specify resource requirements
  - Partition, processor count, node count, minimum memory per node, minimum processor count per node, specific nodes to use or avoid, node can be shared, etc.

# srun - Interactive Example

Run a job interactively (waits for execution).
Create a four task (and implicitly four processor) resource allocation (job) in the 'devel' partition and execute the program */bin/hostname* in it labeling the output.
The job's resource allocation is automatically released upon termination of all tasks.

```
[user@n16 ~]$ srun -n 4 -p devel -l hostname
0: n8
1: n8
2: n9
3: n9
```

NOTE: Most SLURM command options have both a long form and a single letter equivalent. The alternate form of the above command is
**srun --ntasks=4 --partition=devel --label /bin/hostname**

# srun resource request options

| Short form | Long form | Description |
|---|---|---|
| -n | --ntasks | processors count |
| -N | --nodes | Node count |
| -c | --cpus-per-task | count of CPUs required per task |
| -r | --relative | start allocation on specified node within allocation |
| -w | --nodelist | include (at least) the listed nodes |
| -x | --exclude | exclude listed node(s) |
| -C | --constraint | Constrain allocation based on given feature list |
|  | --contiguous=yes/no | contiguous set of nodes |

# srun – Interactive MPI Example

Run an mpi  job interactively (waits for execution).
Create a four task (and implicitly four processor) resource allocation (job) in the default partition on 4 nodes and execute the mpi job “hello_world” in it.
The job’s resource allocation is automatically released upon termination of all tasks.

```
[user@n16 ~]$ mpirun -srun -N 4 –p devel ./hello_world
Hello world! I'm rank 0 of 4 on n8
Hello world! I'm rank 1 of 4 on n9
Hello world! I'm rank 2 of 4 on n10
Hello world! I'm rank 3 of 4 on n11
```

# srun - Allocation Example (1)

Create a four task (and implicitly four processor) resource allocation (job) in the ‘devel’ partition and spawn a shell to use it.
Launch two job steps (sequentially) to use the job’s allocation.
The job’s resource allocation is automatically released upon termination of the shell.

```
[user@n16 ~]$ srun -n4 -p devel –A
[user@n16 ~]$ squeue
  JOBID PARTITION     NAME     USER  ST       TIME  NODES NODELIST
     14          devel               user    R        1:01            2 n[8-9]
```

Job 14 has been created

```
[user@n16 ~]$ srun hostname
n8
n8
n9
n9
```

Job step maintains job’s four tasks

```
[user@n16 ~]$ srun -n 2 hostname
n8
n9
```

Job step explicitly specifies two tasks

# srun - Allocation Example (2)

Note that a new shell is spawned when running srun -A

```
[user@n16 ~]$ printenv | grep SL
[user@n16 ~]$
[user@n16 ~]$ srun -A -n4
[user@n16 ~]$ printenv | grep SL
SLURM_NODELIST=n[1-2]
SLURM_NNODES=2
SLURM_JOBID=27
SLURM_TASKS_PER_NODE=2(x2)
SLURM_NPROCS=4
[user@n16 ~]$ exit
exit
[user@n16 ~]$ printenv | grep SL
[user@n16 ~]$
```

No  SLURM variables defined

Variables defined in the new shell

Back in the old shell

# srun - Batch Example

Submit a batch job that executes different job steps on different nodes simultaneously

```
[user@n16 ~]$ ls
script
[user@n16 ~]$  cat ./script
#!/bin/bash
srun -r 0 -n 1 sleep 10&
srun -r 1 -n 3 sleep 10&
[user@n16 ~]$ sbatch script
srun: jobid 36 submitted
[user@n16 ~]$ squeue -s
STEPID    PARTITION    USER     TIME NODELIST
36.0                 test      user     0:04 n8
36.1                 test      user     0:01 n[9-11]

[user@n16 ~]$ ls

script  slurm-36.out
```

Step 0

Step 1

Output goes to slurm-`job id`.out

# scancel

- Send specified signal to a job and/or job step.
- By default, sends SIGKILL to terminate job.
- Filters can be used to specify user, program name, partition, job state, etc.

Cancel job id 12345

```
[user@n16 ~]$ scancel 12345
```

Cancel all jobs belonging to user *brian* with interaction

```
[root@n16 root]# scancel --interactive --user=brian
Cancel job id=13601 name=summer partition=pdebug [y/n]? y
Cancel job id=13777 name=NewJob partition=pdebug [y/n]? n
```

# sacct

- View accounting data after job completes.

- SLURM job accounting data is stored in a file
  - /hptc_cluster/slurm/job/jobacct.log
- The ‘sacct’ command accesses and parses this file and displays the data requested by the user.
- Access to viewing the job accounting data is controlled by the permissions on the data file. To let all users view the data:
- # **chmod 644 /hptc_cluster/slurm/job/jobacct.log**

# sacct

```
[user@n16 ~]$ sacct -j 3301
Jobstep    Jobname    Partition  Nprocs Status          Error
------------ ------------- ------------ --------- ----------------- -------
3301         sigspin      lsf                   2 COMPLETED       0

[user@n16 ~]$ sacct --dump -j 3301
3301 lsf 20050602042616 1117686376 - - JOB_START 1 16 500 500 sigspin 0 0
4 n16

3301 lsf 20050602042616 1117686376 - - JOB_STEP 1 38 0 20050602042728
CD 0 2 4 72 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0

3301 lsf 20050602042616 1117686376 - - JOB_TERMINATED 1 38 42
20050602042658 CD 0 0 4 42 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0
```

- See the sacct man page for details on data fields.

# scontrol

- **Administrative tool** to set and get configuration information
- Used during gconfig to create initial SLURM configuration file

- To look at the SLURM configuration

```
# scontrol show config

# scontrol ping
Slurmctld(primary/backup) at n11/n12 are UP/UP
```

- To drain a node

```
# scontrol update nodename=n1 state=drain reason=‘. . .’
```

- To instruct all daemons to re-read the slurm configuration file

```
# scontrol reconfig
```

# scontrol

- Can be useful to **users** who want to see full state information without fancy filtering or formatting

```
[root@n16 root]# scontrol show partition pdebug
PartitionName=pdebug TotalNodes=64 TotalCPUs=128 RootOnly=NO
   Default=NO Shared=NO State=UP MaxTime=30
   MinNodes=1 MaxNodes=UNLIMITED AllowGroups=(null)
   Nodes=xc[40-103] NodeIndecies=0,63,-1

[root@n16 root]# scontrol show job 70573
JobId=70573 UserId=david(789) Name=winter JobState=RUNNING
   Priority=4294895192 Partition=pdebug BatchFlag=0
   AllocNode:Sid=mcr39:4277 TimeLimit=30
   StartTime=02/03-14:00:49 EndTime=02/03-14:30:49
   NodeList=xc[64-79] NodeListIndecies=64,79,-1
   ....
```

# What knowledge of the allocation is available to my jobs?

- SLURM establishes several environment variables for each job:

```
SLURM_JOBID=56
SLURM_NODELIST=n[17-36]
SLURM_NPROCS=40
SLURM_NNODES=20
SLURM_NODEID=0    (unique per node)
SLURM_PROCID=0    (unique per process)
SLURM_DISTRIBUTION=block
SLURM_CPUS_ON_NODE=2
SLURM_TASKS_PER_NODE=2
```

# SLURM installation on XC

- Most everything is installed and configured in **/opt/hptc/slurm**:
  - Commands, daemons, libraries, doc, manpages, header files, job credential keys

- Shared files on /hptc_cluster/slurm/…
  - **/hptc_cluster/slurm/etc/slurm.conf**
  - The slurm job log file (job/slurm.job.log)
  - The slurmctld state files (state/{job,node,part}_state)

- The rest is configured locally in **/var/slurm**:
  - The daemon log files (log/{slurmctld.log,slurmd.log}
  - The slurmd state file (state/cred_state)
  - The daemon 'pid' files to indicate run status (run/*.pid)

# The slurm.conf Configuration File

- Located at **/hptc_cluster/slurm/etc/slurm.conf**
- During installation:
  - the SLURM primary and backup daemon nodes are selected from the set of nodes with the resource_management role
    - If only one resource_management node, then no backup!
    - Installer can select specific nodes for master and backup daemons from among this set of nodes during cluster_config
  - If a Quadrics ELAN card is detected on the head node, SLURM ELAN switch support is enabled.
    - This can also be enabled/disabled manually

# The slurm.conf Configuration File

- All nodes with the compute role are configured to run slurmd daemons
  - Nodes are displayed but not configurable during cluster_config
  - Node CPU count and memory value are configured during spconfig
  - SLURM supports other configuration options per node
    - Special custom features; weighted scheduling priority

  - A single ‘lsf’ partition is created containing all nodes
    - customized for use by LSF-HPC for XC
    - other partition configurations possible

# SLURM Job Accounting Support

- SLURM job accounting is enabled in the slurm.conf configuration file:

```
[root@n16 root]# scontrol show config | grep -i Acct
JobAcctLoc        = /hptc_cluster/slurm/job/jobacct.log
JobAcctParameters = Frequency=30
JobAcctType       = jobacct/log
```

- Set appropriate permissions on the account file if you want non-root users to view the data:

```
[root@n16 root]# chmod 644
/hptc_cluster/slurm/job/jobacct.log
```

# MUNGE

- SLURM on XC is configured to use MUNGE to authenticate communication between SLURM components running on remote nodes
- Consists of a daemon, a library, and a couple of commands
- Must be running on every node where SLURM daemons are running AND on nodes where SLURM commands may be executed
  - The MUNGE service is associated with the common role and configured automatically
- Very sensitive to unsynchronized time within XC !

# SLURM-based user access to nodes

- SLURM requires user authentication on every node for users submitting jobs
  - As a consequence, the compute nodes are freely accessible to all users

- To control unrestricted access to compute nodes, a pam_slurm module is available
  - When enabled on the compute nodes, PAM authentication will only allow users access to the node if they currently have it allocated in SLURM
  - Provides users with the ability to login and check on their jobs while preventing other users from stealing CPU cycles
  - Supports applications with ssh-type launch mechanisms (LINDA apps, etc.)

# SLURM Exercises

- Execute ‘**scontrol ping**’. What is the state of the master SLURM daemon? Is there a backup controller configured?
- Execute ‘**sinfo**’. How many compute nodes are ready for use? Use the sinfo options to view the number of processors per node.
- Review the slurm.conf file. Configure a default partition that is accessible by non-root users. Configure an ‘ateam’ feature on half of the compute nodes, and a ‘bteam’ feature on the other half.
- Execute ‘**srun –n X hostname**’. Use ‘srun’ options to execute the ‘hostname’ command only once on the ‘ateam’ nodes.
- Execute ‘**srun –n X sleep 60**’ (choose an appropriate X). In another terminal, execute ‘**sinfo**’ and ‘**squeue**’ to view the job. On which node(s) is your job running? Use ‘**scancel**’ to kill the job.
- Execute ‘**srun –n1 printenv | grep SLURM**’. How many SLURM environment variables are added to the user environment?

Questions?

hp

invent