Slurm Workload Manager
Architecture, Configuration and Use

Morris Jette
jette@schedmd.com

SchedMD LLC
http://www.schedmd.com

Goals

- Learn the basics of SLURM's architecture, daemons and commands
- Learn how to use a basic set of commands
- Learn how to build, configure and install SLURM with a simple configuration
- This is only an introduction, but it should provide you a good start

Agenda

- Role of a resource manager and job scheduler
- SLURM design and architecture
- SLURM commands
- SLURM build and configuration
- SLURM scheduling plugins and development

Role of a Resource Manager

- The “glue” for a parallel computer to execute parallel jobs
- It should make a parallel computer as almost easy to use as a PC

On a PC.
Execute program “a.out”:

```
a.out
```

On a cluster.
Execute 8 copies of “a.out”:

```
srun -n8 a.out
```

- MPI would typically be used to manage communications within the parallel program

Role of a Resource Manager

- Allocate resources within a cluster
  - Nodes (typically a unique IP address)
    - NUMA boards
      - Sockets
        - Cores
          - Hyperthreads
      - Memory
    - Interconnect/switch resources
    - Generic resources (e.g. GPUs)
  - Licenses
- Launch and otherwise manage jobs

Can require extensive knowledge about the hardware and system software (e.g. to alter network routing or manage switch window)

Role of a Job Scheduler

- When there is more work than resources, the job scheduler manages queue(s) of work
  - Supports complex scheduling algorithms
    - Optimized for network topology, fair share scheduling, advanced reservations, preemption, gang scheduling (time-slicing jobs), etc.
  - Supports resource limits (by queue, user, group, etc.)

- Many batch systems provide both resource management and job scheduling within a single product (e.g. LSF) while others use distinct products (e.g. Torque resource manager and Moab job scheduler)

Agenda

- Role of a resource manager and job scheduler
- SLURM design and architecture
- SLURM commands
- SLURM build and configuration
- SLURM scheduling plugins and development

What is SLURM?

- **S**imple **L**inux **U**tility for **R**esource **M**anagement
- Development started in 2002 at Lawrence Livermore National Laboratory as a simple resource manager for Linux clusters
- Has evolved into a capable job scheduler through use of optional plugins
- About 500,000 lines of C code today.
- Supports AIX, Linux, Solaris, other Unix variants
- Used on many of the world's largest computers

SLURM Design Goals

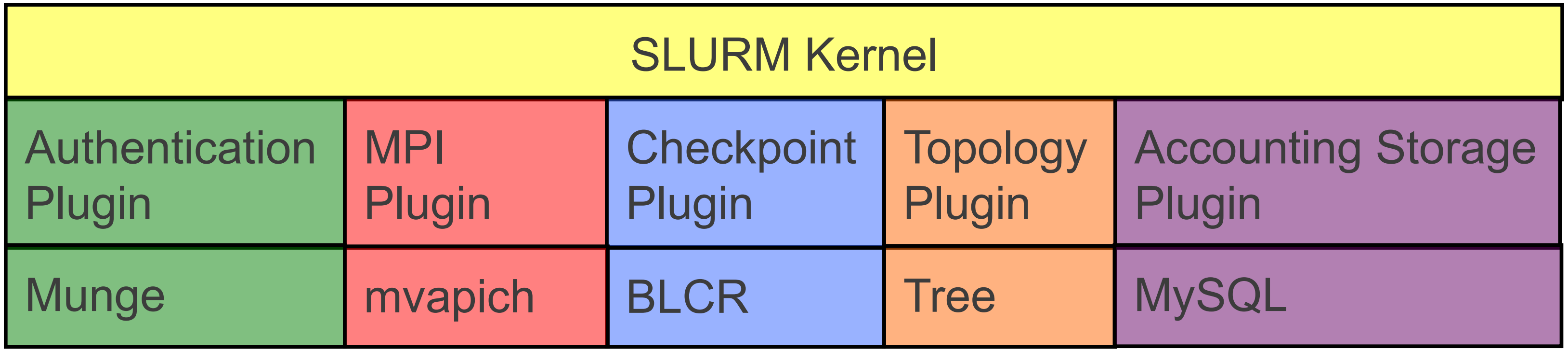
- Small and simple (depends upon configuration, used by Intel for their “cluster on a chip”)
- Highly scalable (managing 1.6 million core IBM BlueGene/Q, tested to 33 million cores using emulation)
- Fast (throughput up to 600 jobs per second and up to 1000 job submissions per second)
- Open source (GPL v2, active world-wide development)
- System administrator friendly
- Secure
- Fault-tolerant (no single point of failure)
- Portable

SLURM Portability

- No kernel modifications
- C language
- *Autoconf* configuration engine adapts to environment
- Provides skeleton of functionality with general-purpose plugin mechanism. System administrator can extensively customize installation using a building-block approach
- Various system-specific plugins available and more under development (e.g. *select/bluegene, select/cray)*

Plugins

- Dynamically linked objects loaded at run time based upon configuration file and/or user options
- 80 plugins of 20 different varieties currently available
    - Accounting storage: MySQL, PostgreSQL, text file
    - Network topology: 3D-torus, tree
    - MPI: OpenMPI, MPICH1, MVAPICH, MPICH2, etc.

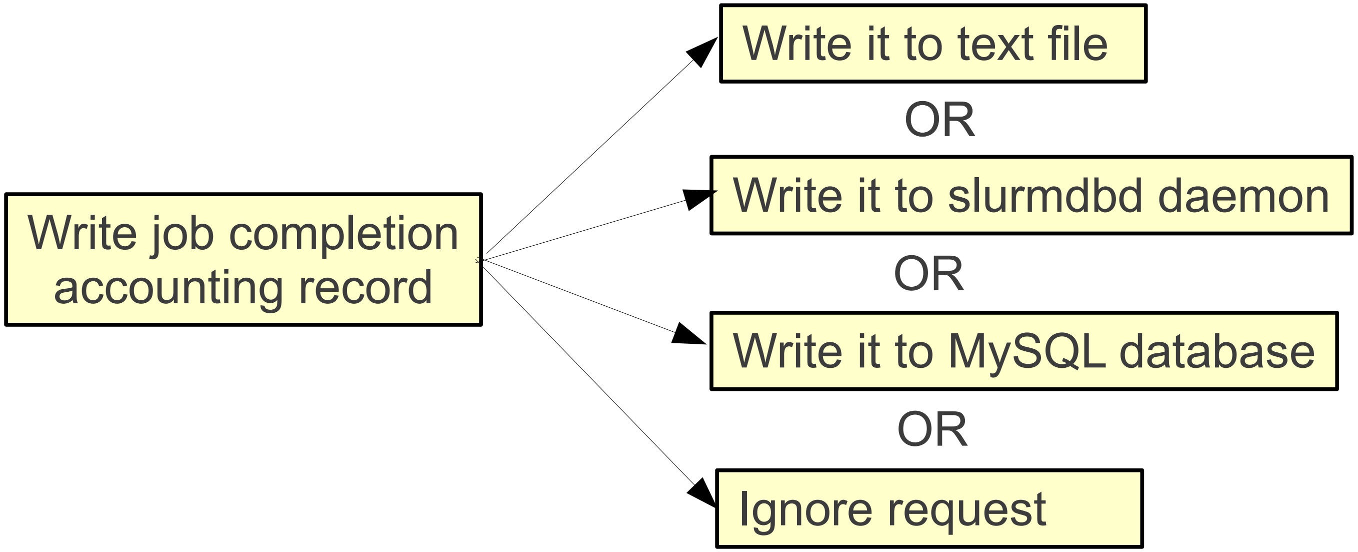


SLURM Kernel

Authentication Plugin	MPI Plugin	Checkpoint Plugin	Topology Plugin	Accounting Storage Plugin
Munge	mvapich	BLCR	Tree	MySQL

Plugin Design

- Plugins typically loaded when the daemon or command starts and persist indefinitely
- Provide a level of indirection to a configurable underlying function



Write job completion accounting record

Write it to text file

OR

Write it to slurmdbd daemon

OR

Write it to MySQL database

OR

Ignore request

Plugin Development

- Interfaces are all documented for custom development (e.g. GreenSpot for optimized use of green energy sources)
- Most plugins have several examples available
- Some plugins have a LUA script interface

Job Submit Plugin

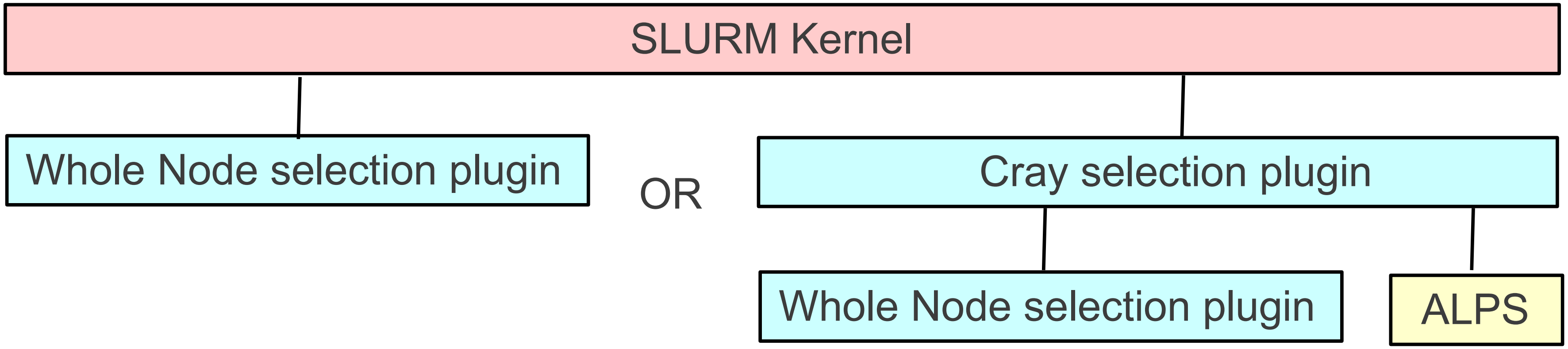
- Call for each job submission or modification
- Can be used to set default values or enforce limits using functionality outside of SLURM proper

Two functions need to be supplied:

```
int job_submit(struct job_descriptor *job_desc, uint32_t submit_uid);

int job_modify(struct job_descriptor *job_desc,
               struct job_record *job_ptr, uint32_t submit_uid);
```

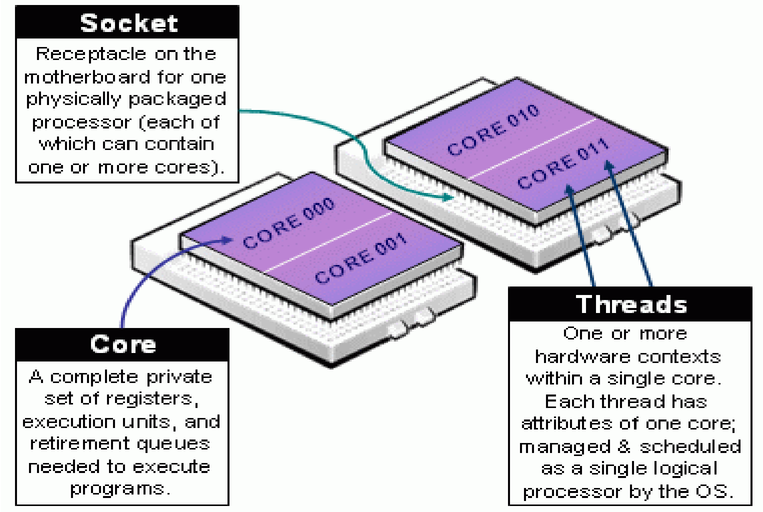
Resource Selection Plugin

- Whole node allocations (select/linear)
- Socket/Core/Hyperthread allocation (select/cons_res)
- IBM BlueGene - Interfaces to IBM's BlueGene APIs
- Cray – Interfaces with Cray's APIs (BASIL) then uses SLURM's whole node allocation plugin



SLURM Entities

- Jobs: Resource allocation requests
- Job steps: Set of (typically parallel) tasks
- Partitions: Job queues with limits and access controls
- Nodes
    - NUMA boards
        - Sockets
            - Cores
                - Hyperthreads
        - Memory
        - Generic Resources (e.g. GPUs)



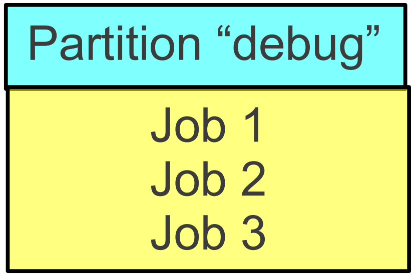
Socket
Receptacle on the motherboard for one physically packaged processor (each of which can contain one or more cores).

Core
A complete private set of registers, execution units, and retirement queues needed to execute programs.

Threads
One or more hardware contexts within a single core. Each thread has attributes of one core; managed & scheduled as a single logical processor by the OS.

SLURM Entities Example

- Users submit jobs to a partition (queue)

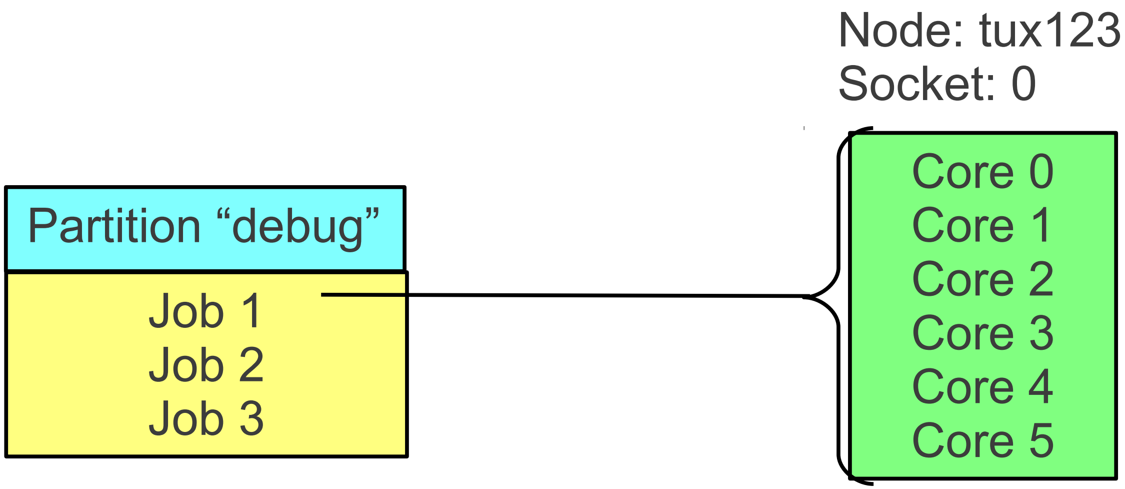


Partition “debug”

Job 1
Job 2
Job 3

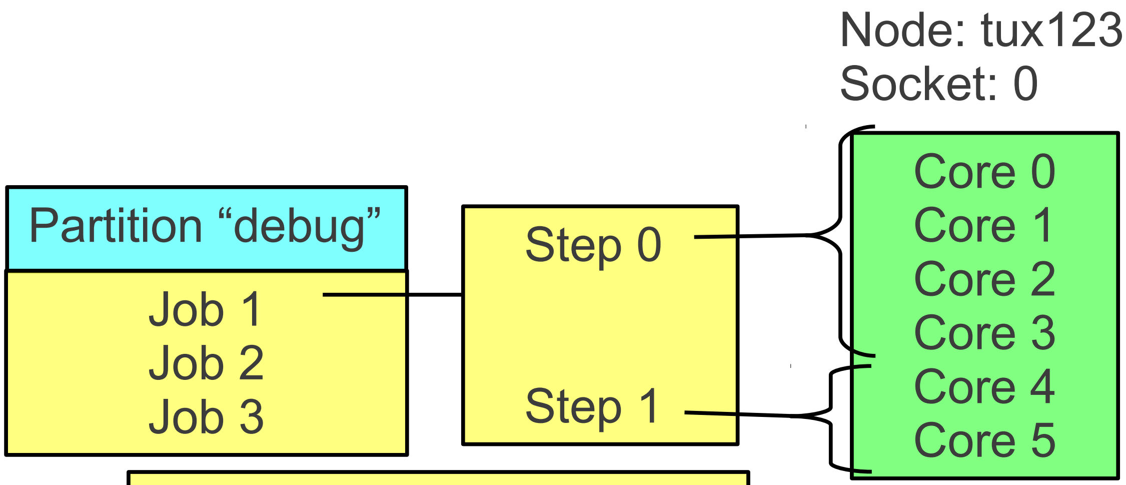
SLURM Entities Example

- Jobs are allocated resources



SLURM Entities Example

- Jobs spawn steps, which are allocated resources from within the job's allocation

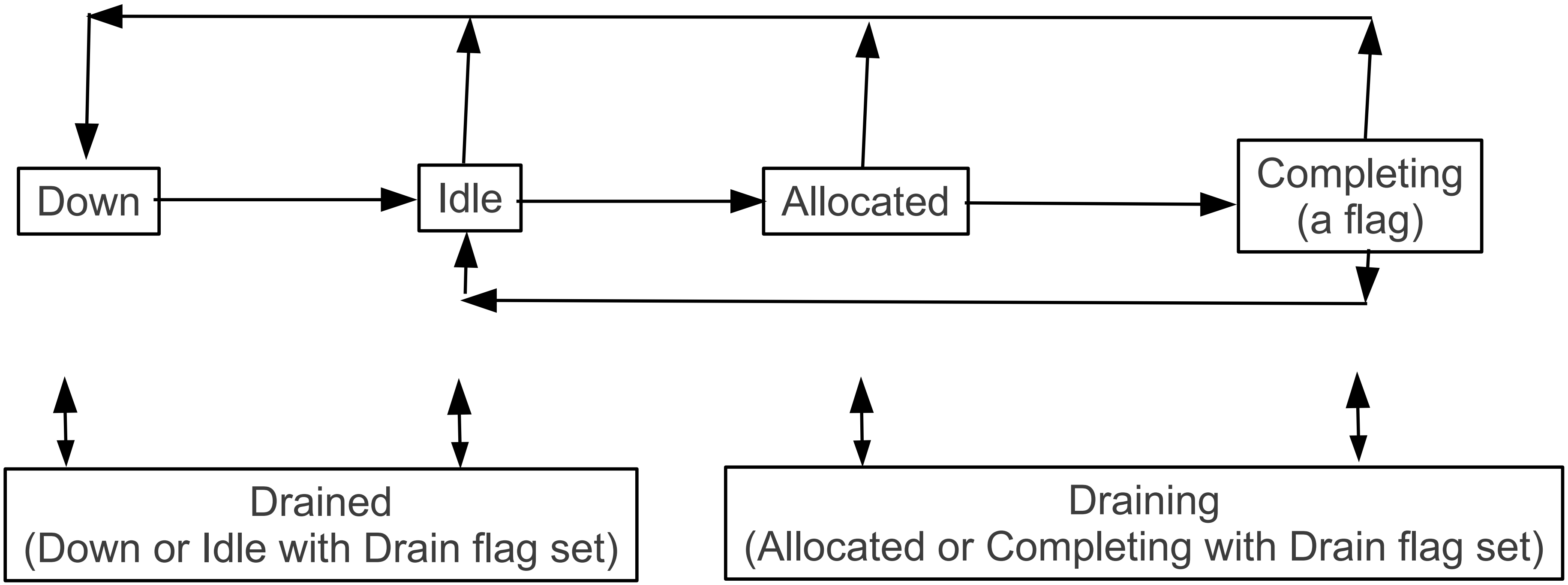


```
#!/bin/bash
srun -n4 –exclusive a.out &
srun -n2 –exclusive a.out &
wait
```

Node State Information

- NUMA boards, Sockets, Cores, Threads
- CPUs (can treat each core or each thread as a CPU for scheduling purposes)
- Memory size
- Temporary disk space
- Features (arbitrary string, e.g. OS version)
- Weight (scheduling priority, can favor least capable node that satisfies job requirement)
- Boot time
- CPU Load
- State (e.g. drain, down, etc.)
  - Reason, time and user ID (e.g. “Bad PDU [operator@12:40:10T12/20/2011]”)

Node States



Down

Idle

Allocated

Completing
(a flag)

Drained
(Down or Idle with Drain flag set)

Draining
(Allocated or Completing with Drain flag set)

```
> scontrol update NodeName=X State=[drain | resume] Reason=X
```

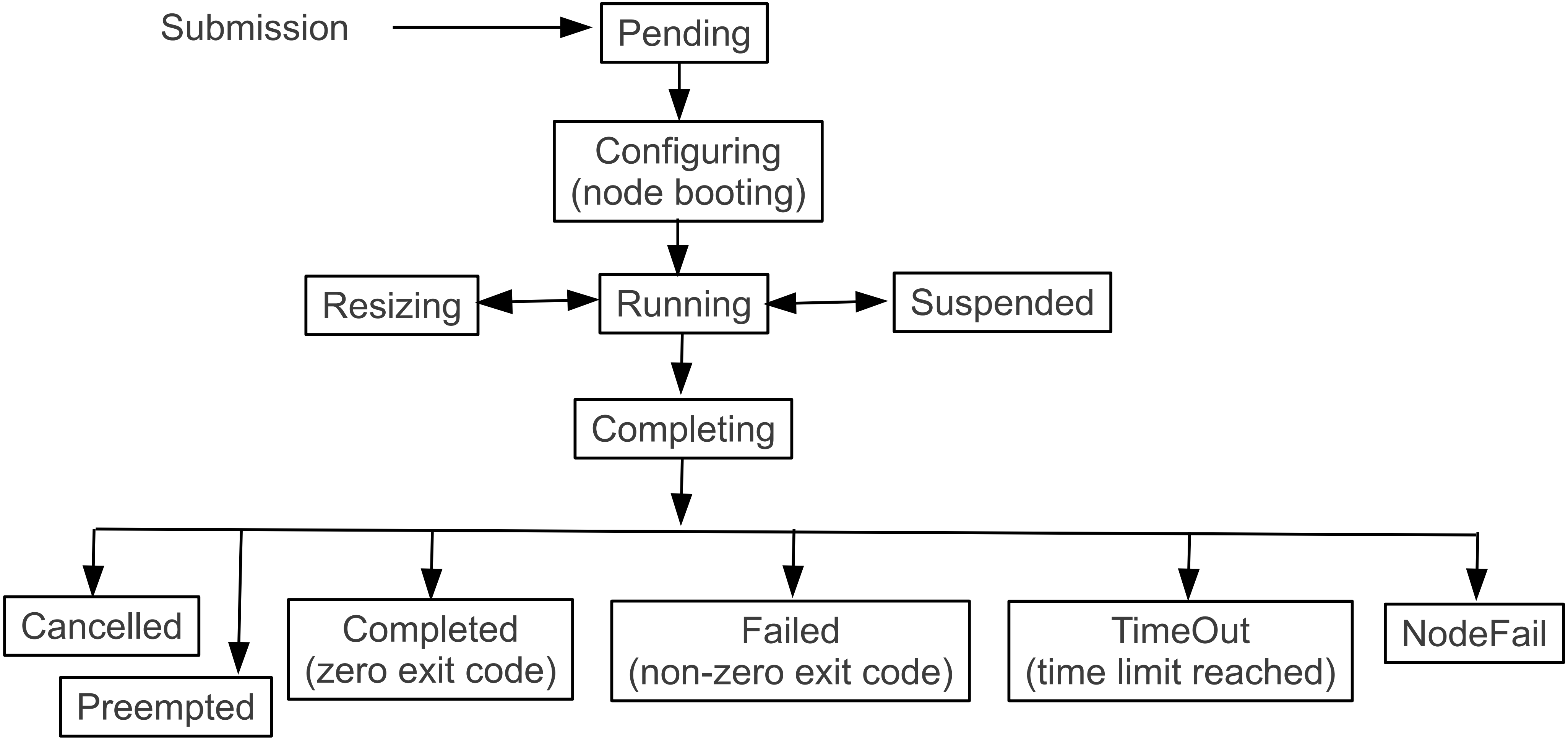
Queue/Partition State Information

- Associated with specific set of nodes
  - Nodes can be in more than one partition
- Job size and time limits (e.g. small size and time limits for some partition and larger limits for others)
- Access control list (by Linux group)
- Preemption rules
- State information (e.g. drain)
- Over-subscription and gang scheduling rules

Job State Information

- ID (a number)
- Name
- Time limit (minimum and/or maximum)
- Size specification (minimum and/or maximum; nodes, CPUs, sockets, cores, and/or threads)
- Specific node names to include or exclude in allocation
- Node features required in allocation
- Dependency
- Account name
- Quality Of Service (QOS)
- State (Pending, Running, Suspended, Cancelled, Failed, etc.)

Job States



Submission

Pending

Configuring (node booting)

Resizing

Running

Suspended

Completing

Cancelled

Preempted

Completed (zero exit code)

Failed (non-zero exit code)

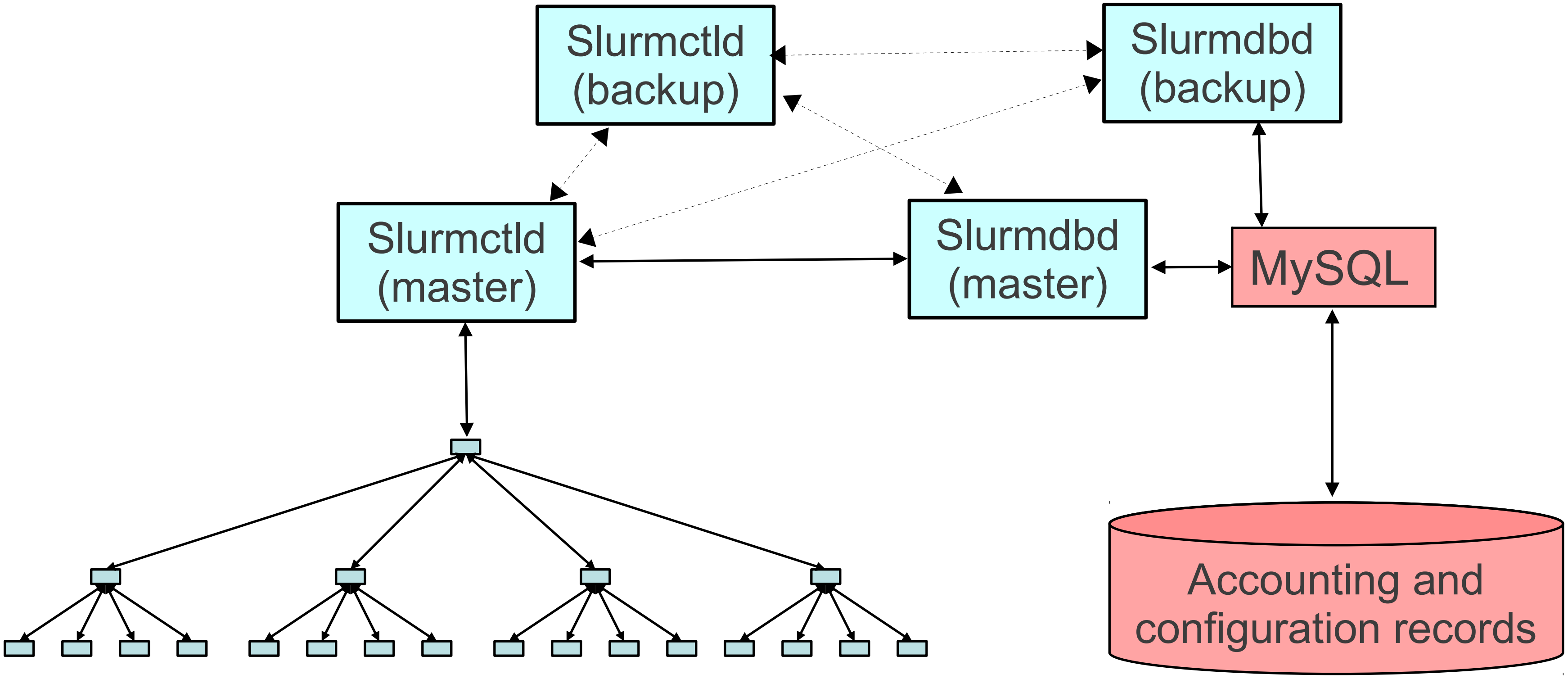
TimeOut (time limit reached)

NodeFail

Step State Information

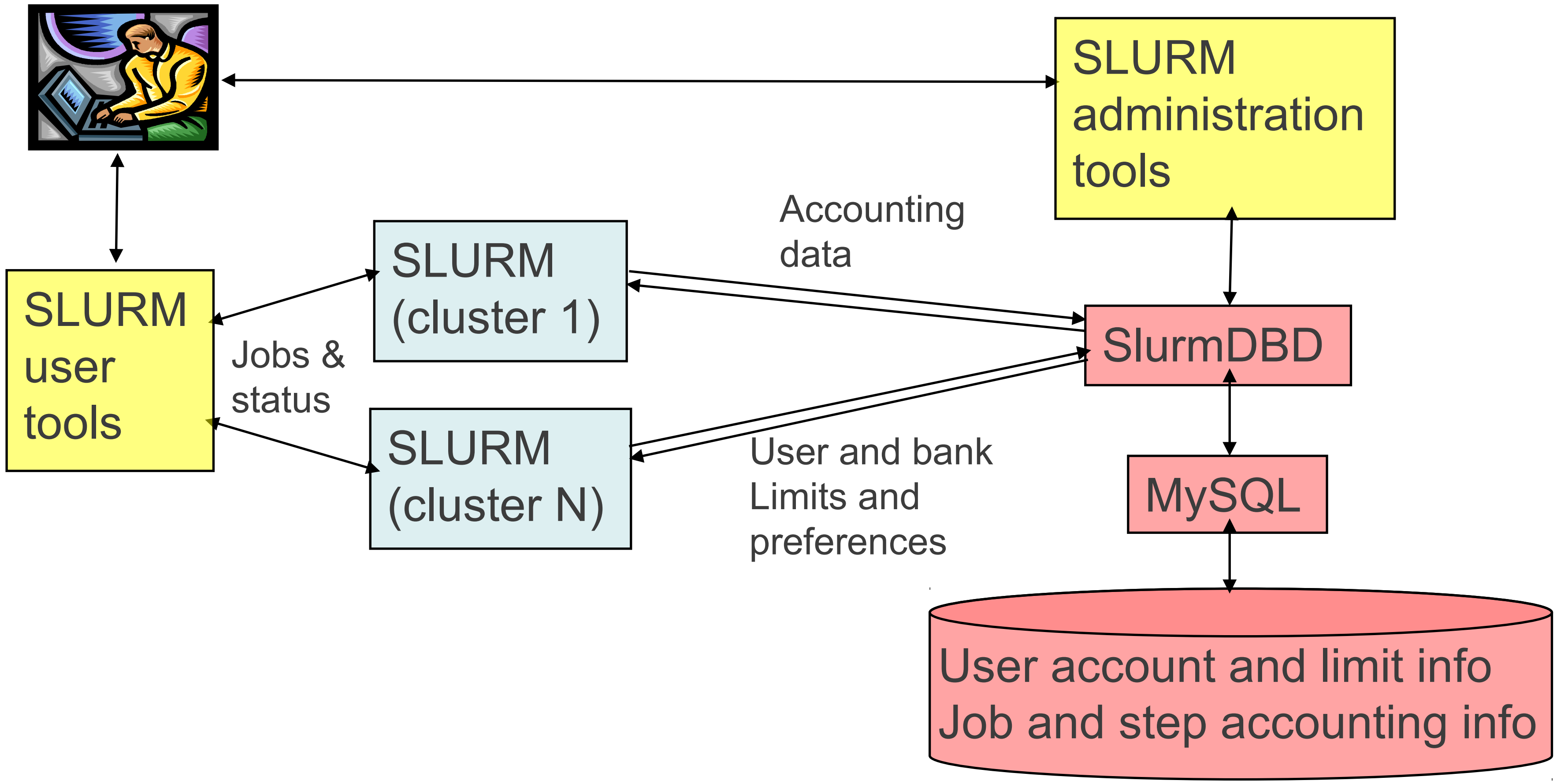
- ID (a number): <jobid>.<stepid>
- Name
- Time limit (maximum)
- Size specification (minimum and/or maximum; nodes, CPUs, sockets, cores, and/or threads)
- Specific node names to include or exclude in allocation
- Node features required in allocation

Cluster Architecture
Typical Linux Cluster



Slurmd daemons on compute nodes
(Note hierarchical communications with configurable fanout)

Enterprise Architecture



SLURM administration tools

Accounting data

SLURM (cluster 1)

SLURM user tools

Jobs & status

SlurmDBD

SLURM (cluster N)

User and bank Limits and preferences

MySQL

User account and limit info
Job and step accounting info

Daemons

- **slurmctld** – Central controller (typically one per cluster)
  - Optional backup with automatic fail over
  - Monitors state of resources
  - Manages job queues
  - Allocates resources
- **slurmd** – Compute node daemon (typically one per compute node, on Cray and IBM Bluegene systems, one or more on front-end nodes)
  - Launches and manages tasks
  - Small and very light-weight (low memory and CPU use)
  - Quiescent after launch except for optional accounting
  - Supports hierarchical communications with configurable fanout
- **slurmdbd** – database daemon (typically one per enterprise)
  - Collects accounting information
  - Uploads configuration information (limits, fair-share, etc.)
  - Optional backup with automatic fail over

Daemon Command Line Options

- -c Clear previous state, purge all job, step, partition state
- -D Run in the foreground, logs are written to stdout
- -v Verbose error messages, each “v” roughly doubles volume of messages

Typical debug mode command lines

```
> slurmctld -Dcvvvv
> slurmd -Dcvvv
```

Compute Node Configuration

- Execute *slurmd* with *-C* option to print the node's current configuration and exit
- This information can be used as input to the SLURM configuration file

```
> slurmd -C
NodeName=jette CPUs=6 Sockets=1 CoresPerSocket=6 ThreadsPerCore=1
    RealMemory=8000 TmpDisk=930837
```

slurmstepd
(SLURM daemon to shepherd a job step)

- One slurmstepd per job step
- Spawned by slurmd at job step initiation
- Manages a job step and processes its I/O
- Only persists while the job step is active

Logs

- Each daemon writes its own logfile
  - See configuration parameters *SlurmctldLogFile* and *SlurmdLogFile* in *slurm.conf* and LogFile in *slurmdbd.conf*
    - *SlurmdLogFile* name can include “%n” which is replaced by the node's name (e.g. “*SlurmdLogFile=slurmd.%n.log*”)
  - *DebugLevel* configuration parameters control how verbose the logging is
  - Important messages go to *syslog*
  - The daemon's log file can be much more verbose

Logs

- Detailed logging can also be generated on 17 specific sub-systems using the *DebugFlags* configuration parameter
  - Backfill, CPU_Bind, Gang, Gres, Priority, Reservation, Steps, Triggers, etc.
- *DebugFlags* and *DebugLevel* can be reset in real time using the *scontrol* command
  - *scontrol setdebugflags +priority* (Add *DebugFlag* of "*priority*")
  - *scontrol setdebug debug*

Agenda

- Role of a resource manager and job scheduler
- SLURM design and architecture
- SLURM commands
- SLURM build and configuration
- SLURM scheduling plugins and development

Commands: General Information

- Man pages available for all commands, daemons and configuration files
- --help option prints brief description of all options
- --usage option prints a list of the options
- Commands can be run on any node in the cluster
- Any failure results in a non-zero exit code
- APIs make new tool development easy
  - Man pages available for all APIs

Commands: General Information

- Almost all options have two formats
  - A single letter option (e.g. “-p debug” for partition debug)
  - A verbose option (e.g. “--partition=debug”)
- Time formats are days-hours:minutes:seconds
- Almost all commands support verbose logging with “-v” option, use more v’s for more verbosity, -vvvv
- Many environment variables can be used to establish site-specific and/or user-specific defaults
  - For example “SQUEUE_STATES=all” for the squeue command to display jobs in any state, including COMPLETED or CANCELLED

SLURM Commands: Job/step Allocation

- **sbatch** – Submit script for later execution (batch mode)
- **salloc** – Create job allocation and start a shell to use it (interactive mode)
- **srun** – Create a job allocation (if needed) and launch a job step (typically an MPI job)
- **sattach** – Connect stdin/out/err for an existing job or job step

Job and Step Allocation Examples

Submit sequence of three batch jobs

```
> sbatch –ntasks=1 –time=10 pre_process.bash
Submitted batch job 45001
> sbatch –ntasks=128 –time=60 --depend=45001 do_work.bash
Submitted batch job 45002
> sbatch –ntasks=1 –time=30 --depend=45002 post_process.bash
Submitted batch job 45003
```

Job and Step Allocation Examples

Create allocation for 2 tasks then launch “hostname” on the allocation, label output with the task ID

```
> srun –ntasks=2 –label hostname
0: tux123
1: tux123
```

As above, but allocate the job two whole nodes

```
> srun –nnodes=2 --exclusive –label hostname
0: tux123
1: tux124
```

Job and Step Allocation Examples

Create allocation for 4 tasks and 10 minutes for bash shell, then launch some tasks

```
> salloc –ntasks=4 –time=10 bash
salloc: Granted job allocation 45000
> env | grep SLURM
SLURM_JOBID=45000
SLURM_NPROCS=4
SLURM_JOB_NODELIST=tux[123-124]
…
> hostname
tux_login
> srun –label hostname
0: tux123
1: tux123
2: tux124
3: tux124
> exit  (terminate bash shell)
```

Different Executables by Task ID

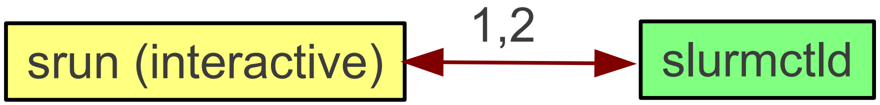
- Different programs may be launched by task ID with different arguments
- Use “--multi-prog option and specify configuration file instead of executable program
- Configuration file lists task IDs, executable programs, and arguments (“%t” mapped to task ID and “%o” mapped to offset within task ID range)

```
> cat master.conf
#TaskID  Program              Arguments
0          /usr/me/master
1-4       /usr/me/slave      --rank=%o

> srun –ntasks=5 –multi-prog master.conf
```

MPI Support

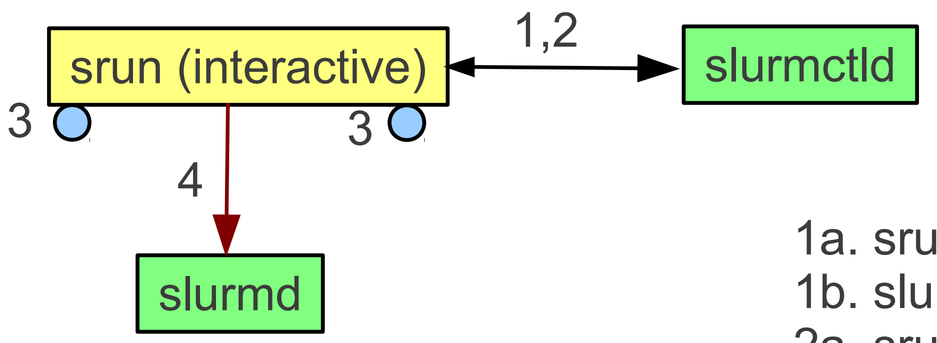
- Many different MPI implementations are supported:
  - MPICH1, MPICH2, MVAPICH, OpenMPI, etc.
- Many use srun to launch the tasks directly
- Some use “mpirun” or another tool within an existing SLURM allocation (they reference SLURM environment variables to determine what resources are allocated to the job)
- Details are online: *http://www.schedmd.com/slurmdocs/mpi_guide.html*

Linux Job Launch Sequence



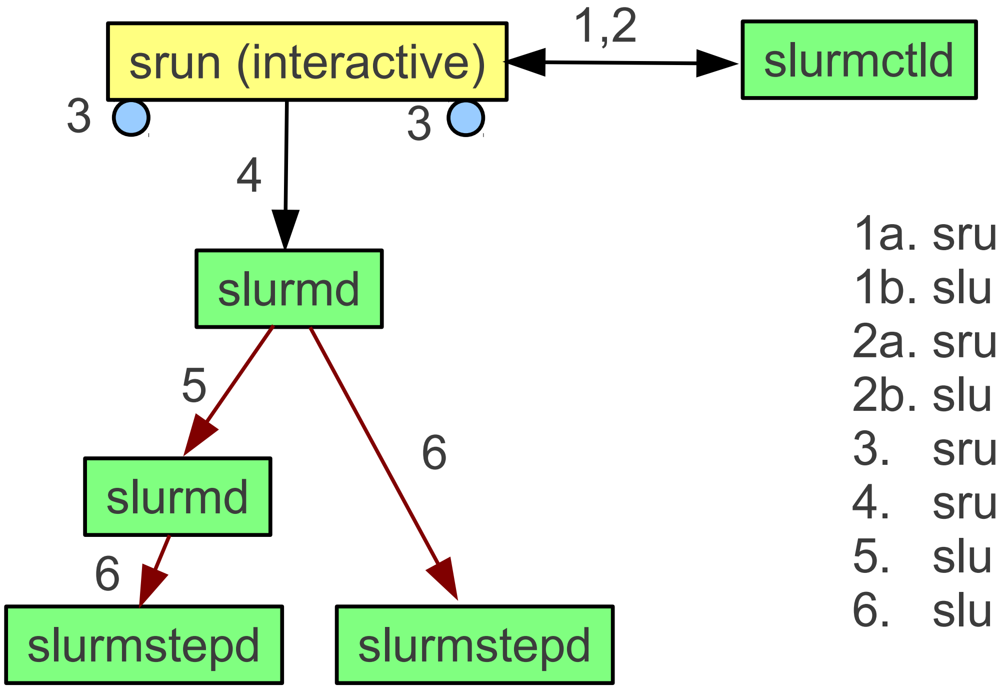
1a. srun sends job allocation request to slurmctld
1b. slurmctld grant allocation and returns details
2a. srun sends step create request to slurmctld
2b. slurmctld responds with step credential

Linux Job Launch Sequence



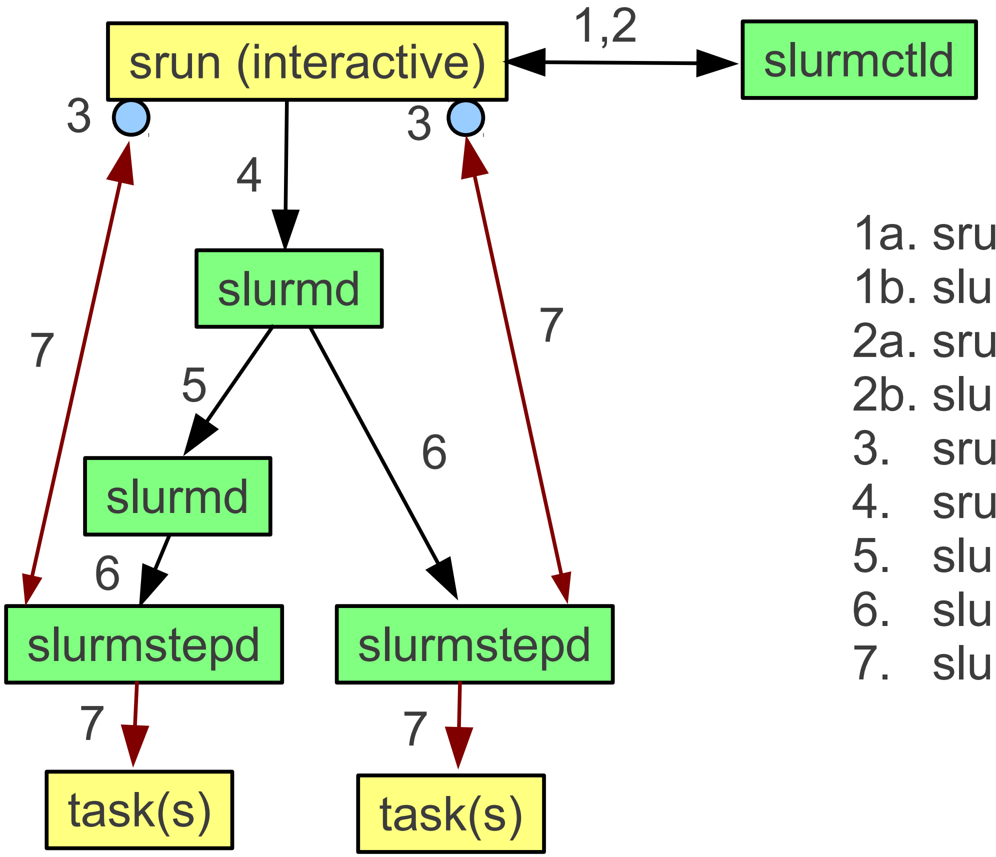
1a. srun sends job allocation request to slurmctld
1b. slurmctld grant allocation and returns details
2a. srun sends step create reqeust to slurmctld
2b. slurmctld responds with step credential
3. srun opens sockets for I/O
4. srun forwards credential with task info to slurmd

Linux Job Launch Sequence



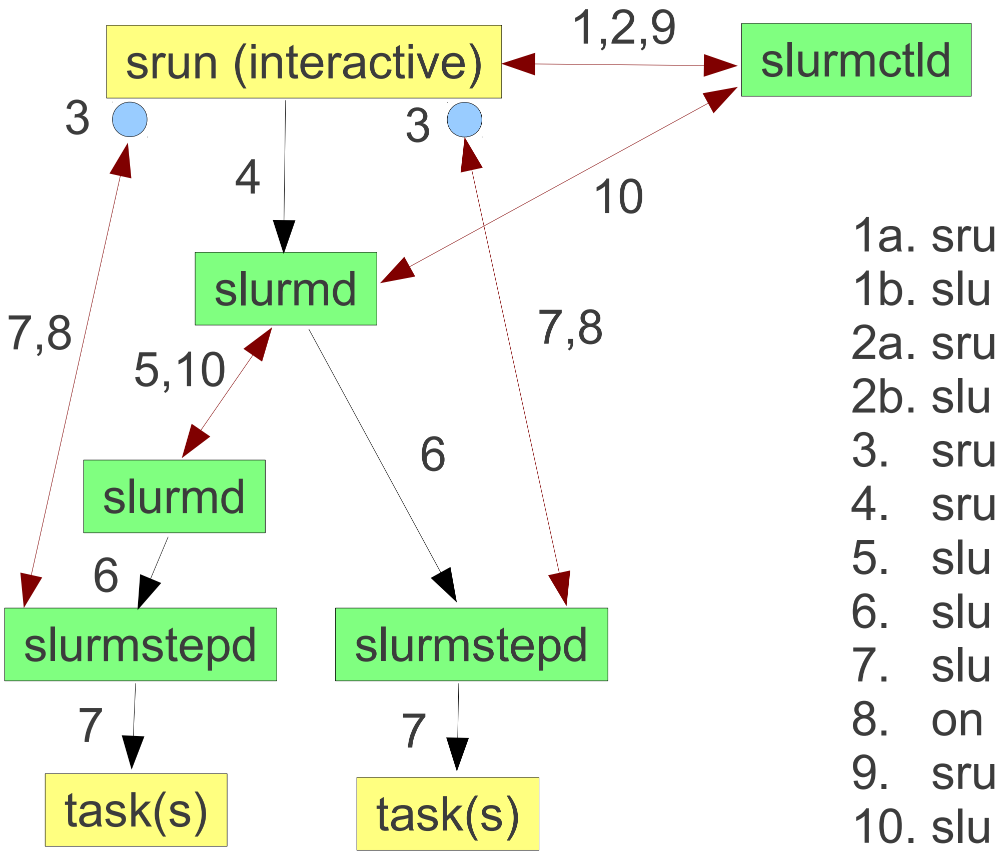
1a. srun sends job allocation request to slurmctld
1b. slurmctld grant allocation and returns details
2a. srun sends step create request to slurmctld
2b. slurmctld responds with step credential
3. srun opens sockets for I/O
4. srun forwards credential with task info to slurmd
5. slurmd forward request as needed (per fanout)
6. slurmd forks/execs slurmstepd

Linux Job Launch Sequence



1a. srun sends job allocation request to slurmctld
1b. slurmctld grant allocation and returns details
2a. srun sends step create request to slurmctld
2b. slurmctld responds with step credential
3. srun opens sockets for I/O
4. srun forwards credential with task info to slurmd
5. slurmd forward request as needed (per fanout)
6. slurmd forks/execs slurmstepd
7. slurmstepd connects I/O to run & launches tasks

Linux Job Launch Sequence



1a. srun sends job allocation request to slurmctld
1b. slurmctld grant allocation and returns details
2a. srun sends step create request to slurmctld
2b. slurmctld responds with step credential
3. srun opens sockets for I/O
4. srun forwards credential with task info to slurmd
5. slurmd forward request as needed (per fanout)
6. slurmd forks/execs slurmstepd
7. slurmstepd connects I/O to run & launches tasks
8. on task termination, slurmstepd notifies srun
9. srun notifies slurmctld of job termination
10. slurmctld verifies termination of all processes via slurmd and releases resources for next job

SLURM Commands: System Information

- **sinfo** – Report system status (nodes, queues, etc.)
- **squeue** – Report job and job step status
- **smap** – Report system, job or step status with topology (curses-based GUI), less functionality than sview
- **sview** – Report and/or update system, job, step, partition or reservation status with topology (GTK-based GUI)
- **scontrol** – Administrator tool to view and/or update system, job, step, partition or reservation status

sinfo Command

- Reports status of nodes or partitions
  - Partition-oriented format is the default
- Almost complete control over filtering, sorting and output format is available

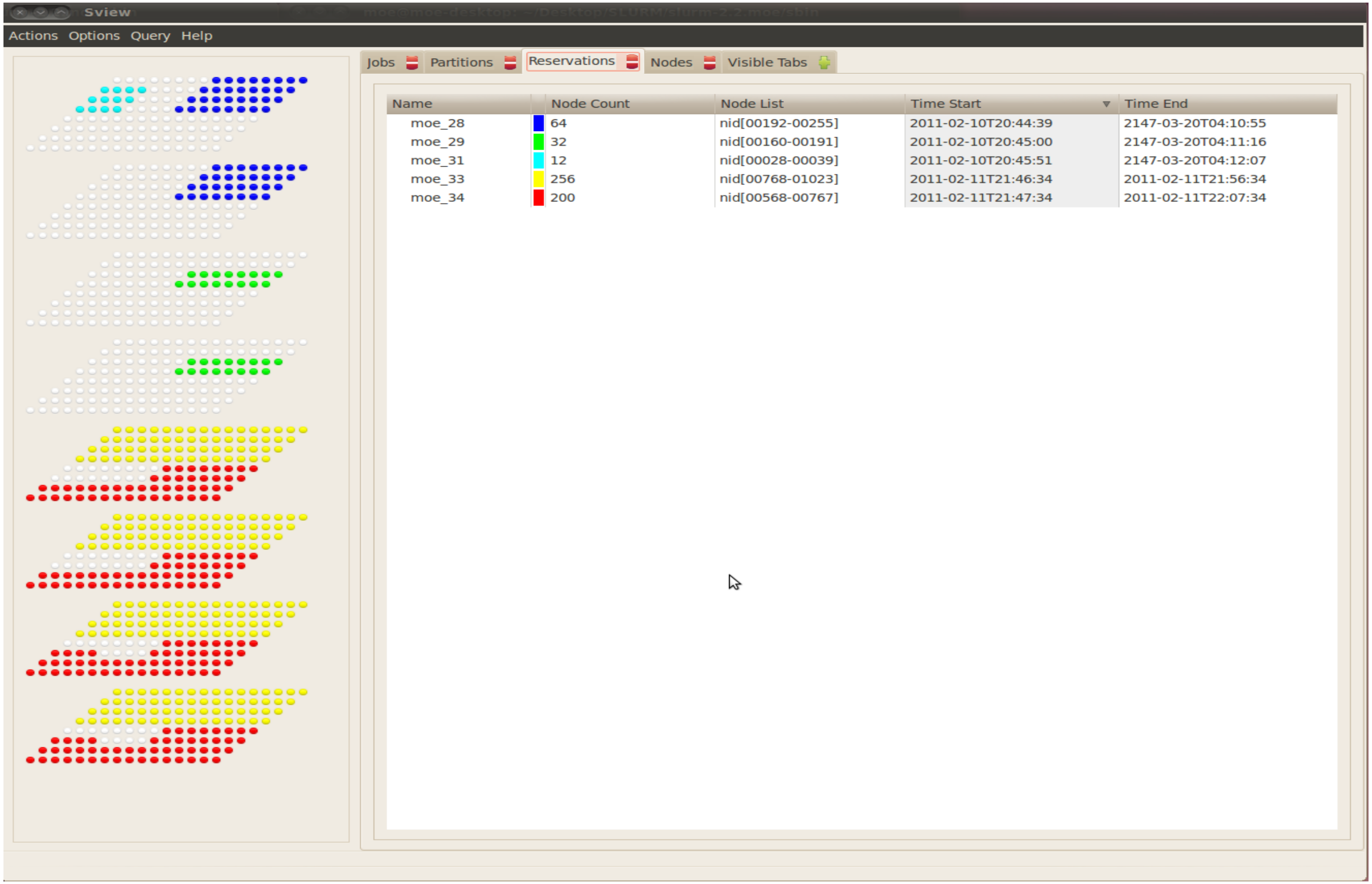
```
> sinfo --Node (report status in node-oriented form)
NODELIST     NODES  PARTITION  STATE
tux[000-099]      100    batch        idle
tux[100-127]        28    debug        idle

> sinfo -p debug (report status of nodes in partition “debug”)
PARTITION AVAIL TIMELIMIT NODES NODELIST
debug              up        60:00         28 tux[100-127]

> sinfo -i60   (report status every 60 seconds)
```

squeue Command

- Reports status of jobs and/or steps in slurmctld daemon's records (recent job's only, older information available in accounting records only)
- Almost complete control over filtering, sorting and output format is available

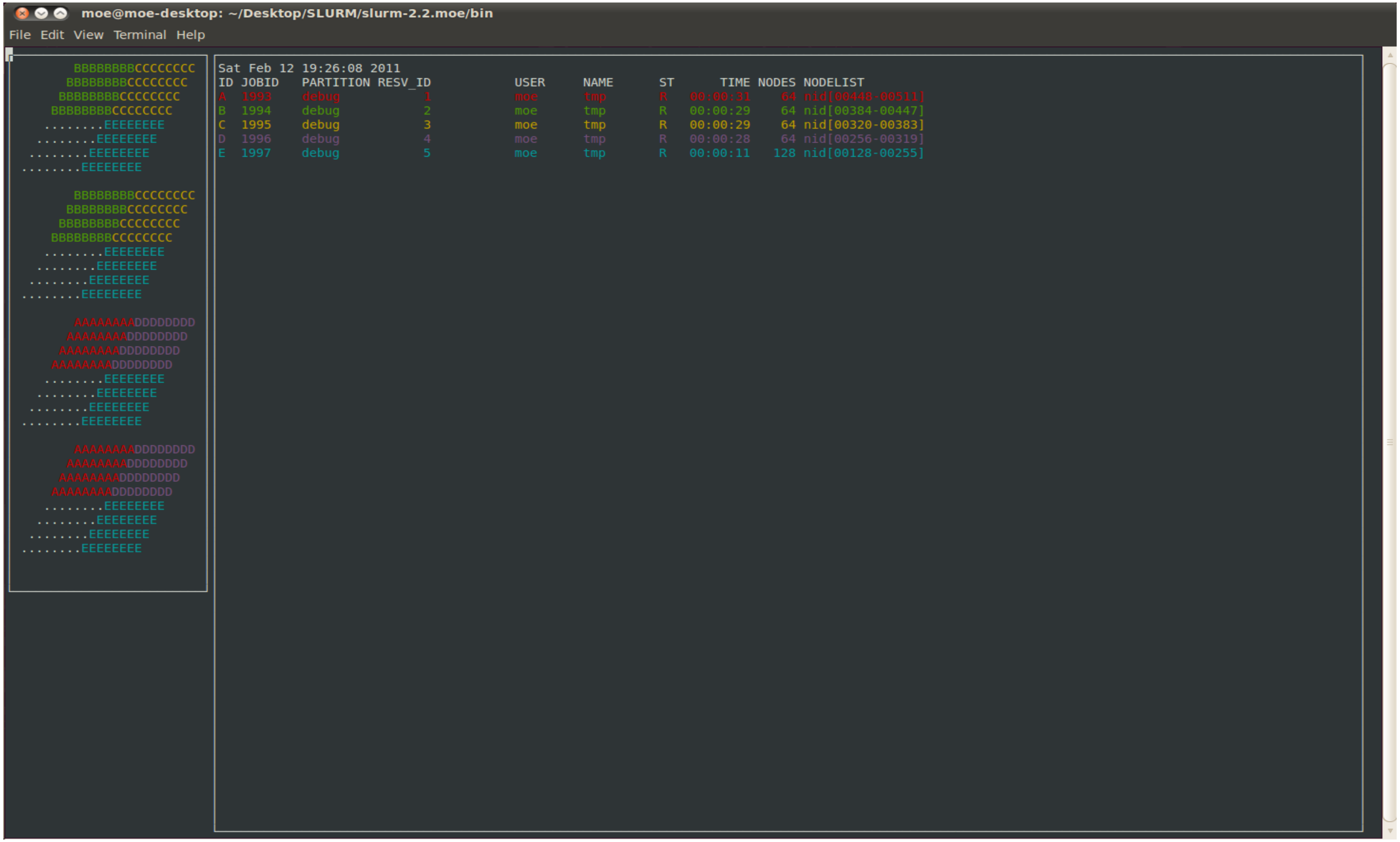
```
> squeue -u alec -t all (report jobs for user “alec” in any state)
JOBID PARTITION  NAME  USER  ST  TIME  NODES NODELIST(REASON)
45124  debug         a.out   alec    CD   0:12          1 tux123

> squeue -s -p debug  (report steps in partition “debug”);
STEPID PARTITION NAME USER TIME NODELIST
45144.0 debug         a.out    moe   12:18 tux[100-115]

> squeue -i60   (report currently active jobs every 60 seconds)
```

sview on Cray (3-D torus)



Name		Node Count	Node List	Time Start	Time End
moe_28	🟦	64	nid[00192-00255]	2011-02-10T20:44:39	2147-03-20T04:10:55
moe_29	🟩	32	nid[00160-00191]	2011-02-10T20:45:00	2147-03-20T04:11:16
moe_31	■	12	nid[00028-00039]	2011-02-10T20:45:51	2147-03-20T04:12:07
moe_33	🟨	256	nid[00768-01023]	2011-02-11T21:46:34	2011-02-11T21:56:34
moe_34	🟥	200	nid[00568-00767]	2011-02-11T21:47:34	2011-02-11T22:07:34

smap on Cray (3-D torus)



moe@moe-desktop: ~/Desktop/SLURM/slurm-2.2.moe/bin
File Edit View Terminal Help

```
       BBBBBBBBCCCCCCCC
      BBBBBBBBCCCCCCCC
     BBBBBBBBCCCCCCCC
    BBBBBBBBCCCCCCCC
   ........EEEEEEEE
  ........EEEEEEEE
 ........EEEEEEEE
........EEEEEEEE

       BBBBBBBBCCCCCCCC
      BBBBBBBBCCCCCCCC
     BBBBBBBBCCCCCCCC
    BBBBBBBBCCCCCCCC
   ........EEEEEEEE
  ........EEEEEEEE
 ........EEEEEEEE
........EEEEEEEE

       AAAAAAAADDDDDDDD
      AAAAAAAADDDDDDDD
     AAAAAAAADDDDDDDD
    AAAAAAAADDDDDDDD
   ........EEEEEEEE
  ........EEEEEEEE
 ........EEEEEEEE
........EEEEEEEE

       AAAAAAAADDDDDDDD
      AAAAAAAADDDDDDDD
     AAAAAAAADDDDDDDD
    AAAAAAAADDDDDDDD
   ........EEEEEEEE
  ........EEEEEEEE
 ........EEEEEEEE
........EEEEEEEE
```

```
Sat Feb 12 19:26:08 2011
ID JOBID   PARTITION RESV_ID            USER      NAME      ST       TIME NODES NODELIST
A  1993    debug           1            moe       tmp       R    00:00:31    64 nid[00448-00511]
B  1994    debug           2            moe       tmp       R    00:00:29    64 nid[00384-00447]
C  1995    debug           3            moe       tmp       R    00:00:29    64 nid[00320-00383]
D  1996    debug           4            moe       tmp       R    00:00:28    64 nid[00256-00319]
E  1997    debug           5            moe       tmp       R    00:00:11   128 nid[00128-00255]
```

scontrol Command

- Designed for system administrator use
- Shows all available fields, but no filtering, sorting or formatting options
- Many fields can be modified

```
> scontrol show partition
PartitionName=debug
   AllocNodes=ALL AllowGroups=ALL Default=YES
   DefaultTime=NONE DisableRootJobs=NO GraceTime=0 Hidden=NO
   MaxNodes=UNLIMITED MaxTime=UNLIMITED MinNodes=1
   Nodes=tux[000-031]
   Priority=1 RootOnly=NO Shared=NO PreemptMode=OFF State=UP
   TotalCPUs=64 TotalNodes=32 DefMemPerNode=512
MaxMemPerNode=1024
> scontrol update PartitionName=debug MaxTime=60
```

SLURM Commands: Accounting

- **sacct** – Report accounting information by individual job and job step
- **sstat** – Report accounting information about currently running jobs and job steps (more detailed than sacct)
- **sreport** – Report resources usage by cluster, partition, user, account, etc.

sacct Command

- Reports accounting information for jobs and steps
- Many filtering and output format options
- Uses accounting file or database (which may not exist depending upon SLURM configuration)

```
> sacct -u joseph (report accounting information for user “joseph”)
> sacct -p debug  (report accounting information for partition “debug”)
```

SLURM Commands: Scheduling

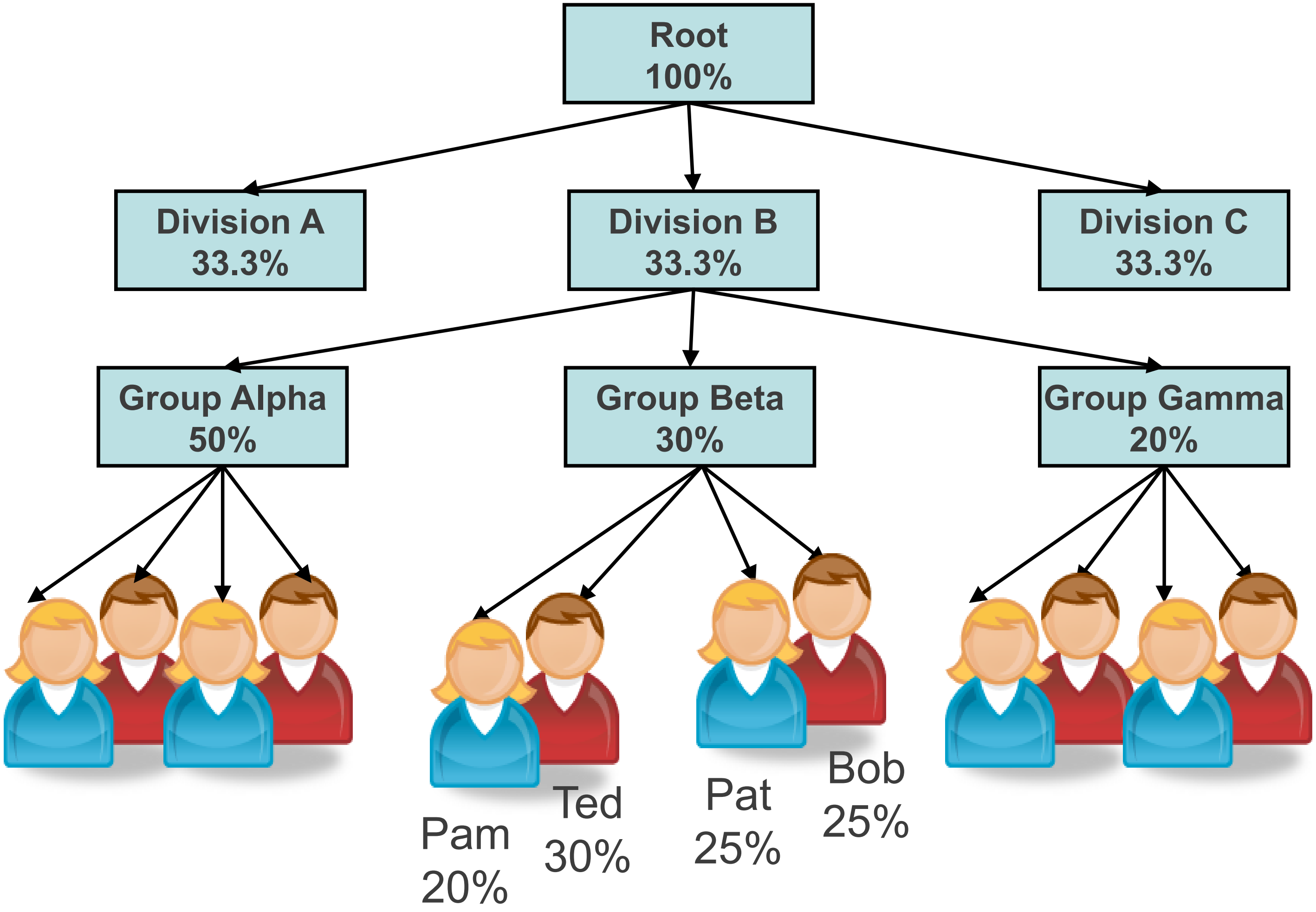
- **sacctmgr** – Database management tool
  - Add/delete clusters, accounts, users, etc.
  - Get/set resource limits, fair-share allocations, etc.
- **sprio** – View factors comprising a job's priority
- **sshare** – View current hierarchical fair-share information
- **sdiag** – View statistics about scheduling module operations (execution time, queue length, etc.) New in SLURM version 2.4

Database Use

- Accounting information written to a database plus
  - Information pushed out live to scheduler daemons
  - Quality of Service (QOS) definitions
  - Fair-share resource allocations
  - Many limits (max job count, max job size, etc)
  - Based upon hierarchical banks
    - Limits by user AND by banks
  - “Bank Coordinators” can directly alter limits and shares for sub-tree

> *“All I can say is wow – this is the most flexible, useful scheduling tool I’ve ever run across.”*
> Adam Todorski, Rensselaer Polytechnic Institute

Hierarchical bank example



SchedMD LLC
http://www.schedmd.com

SLURM Commands: Other

- **scancel** – Signal/cancel jobs or job steps
- **sbcast** – Transfer file to a compute nodes allocated to a job (uses hierarchical communications)
- **srun_cr** – Wrapper to srun for support of Berkeley checkpoint/restart
- **strigger** – Event trigger management tools

scancel Command

- Cancel a running or pending job or step
- Can send arbitrary signal to all processes on all nodes associated with a job or step
- Has filtering options (state, user, partition, etc.)
- Has interactive (verify) mode

```
> scancel 45001.1 (cancel job step 45001.1)
> scancel 45002   (cancel job 45002)
> scancel –user=alec –state=pending (cancel all pending jobs from user “alec”)
```

sbcast Command

- Copy a file to local disk on allocated nodes
  - Execute command after a resource allocation is made
- Data transferred using hierarchical slurmd daemons communications
- May be faster than shared file system

```
> salloc -N100 bash
salloc: Granted job allocaiton 45201
> sbcast --force my_data /tmp/moe/my_data  (overwrite old files)
> srun a.out
> exit  (terminate spawned “bash” shell)
```

strigger command

- SLURM can run an arbitrary script when certain events occur
  - Node goes DOWN
  - Daemon stops or restarts
  - Job close to time limit
  - Many others
- strigger command can be used to create, destroy or list event triggers

Agenda

- Role of a resource manager and job scheduler
- SLURM design and architecture
- SLURM commands
- SLURM build and configuration
- SLURM scheduling plugins and development

Select Distribution

- Download a tar-ball
  - http://www.schedmd.com/#repos
- New minor release about every 9 months
  - 2.4.x June 2012
  - 2.5.x December 2012
- Micro releases with bug fixes about once each month
- Very latest code base available from github
  - “git clone git://github.com/SchedMD/slurm.git”

Install Needed Infrastructure

- Munge (authentication)
  - RPMs: munge, munge-devel, munge-libs
  - Need to create some directories and configure keys
- MySQL (job accounting, limits or QOS)
  - RPMs: mysql-client, mysql-server, libmysqlclient-dev

Build and Install RPMs

- Build and install the relevant RPMs
  - rpmbuild -ta slurm-2.4.1.tar.bz2
  - rpm –install <the rpm files>
- NOTE: Some RPMs are infrastructure dependent
  - slurm-auth-authd*rpm   Authd authentication
  - slurm-bluegene*rpm   IBM BlueGene systems only

SLURM RPMs

Slurm	Commands, daemons
Slurm-dev	Header files and libraries
Slurm-perlapi	Perl API interface to SLURM
Slurm-auth-none	Trivial authentication plugin (avoid)
Slurm-auth-authd	Authd authentication (avoid)
Slurm-auth-munge	Munge authentication (recommended)
Slurm-bluegene	IBM BlueGene support (IBM Bluegene only)
Slurm-slurmdb-direct	Direct database access tool (avoid)
Slurm-slurmdbd	Database daemon and plugins
Slurm-sql	Database plugins
Slurm-plugins	Most plugins

SLURM RPMs

Slurm-torque	Torque/PBS command wrappers (optional)
Slurm-srun2aprun	srun command wrapper for aprun (Cray only)
Slurm-sjobexit	Job exit code management tool (optional)
Slurm-aix	AIX systems only (avoid)
Slurm-percs	IBM PERCS systems plugins (PERCS only)
Slurm-proctrack-sgi-job	SGI job container plugin (recommended)
Slurm-lua	LUA bindings (recommended)
Slurm-sjstat	job stats tool (avoid)
Slurm-pam-slurm	PAM module for restricting node access (optional)
Slurm-blcr	Berkely Lab Checkpoint Restart plugin (optional)

Build and Install RPMs

- Build and install the relevant RPMs
  - rpmbuild -ta slurm-2.4.1.tar.bz2
  - rpm –install <the rpm files>
- NOTE: Some RPMs are infrastructure dependent
  - slurm-auth-authd*rpm   Authd authentication
  - slurm-bluegene*rpm   IBM BlueGene systems only
  - slurm-switch-elan*rpm   Quadrics Elan switch

Build and Install without RPMs

- Uncompress and unpack the tar-ball (or get git repository)
- Create “build” directory and enter it
- Execute “configure <options>”
  - Typical options:

--enable-debug	perform additional debugging
--prefix=<dir>	installation directory
--sysconfdir=<dir>	configuration file directory
--with-munge=<dir>	Munge installation directory
--with-srun2aprun	Build srun wrapper to aprun command

- Run “make” to build the executable
- Run “make install” to install the executable, header, and library files
- Individual commands can be edited, built and instaled quickly

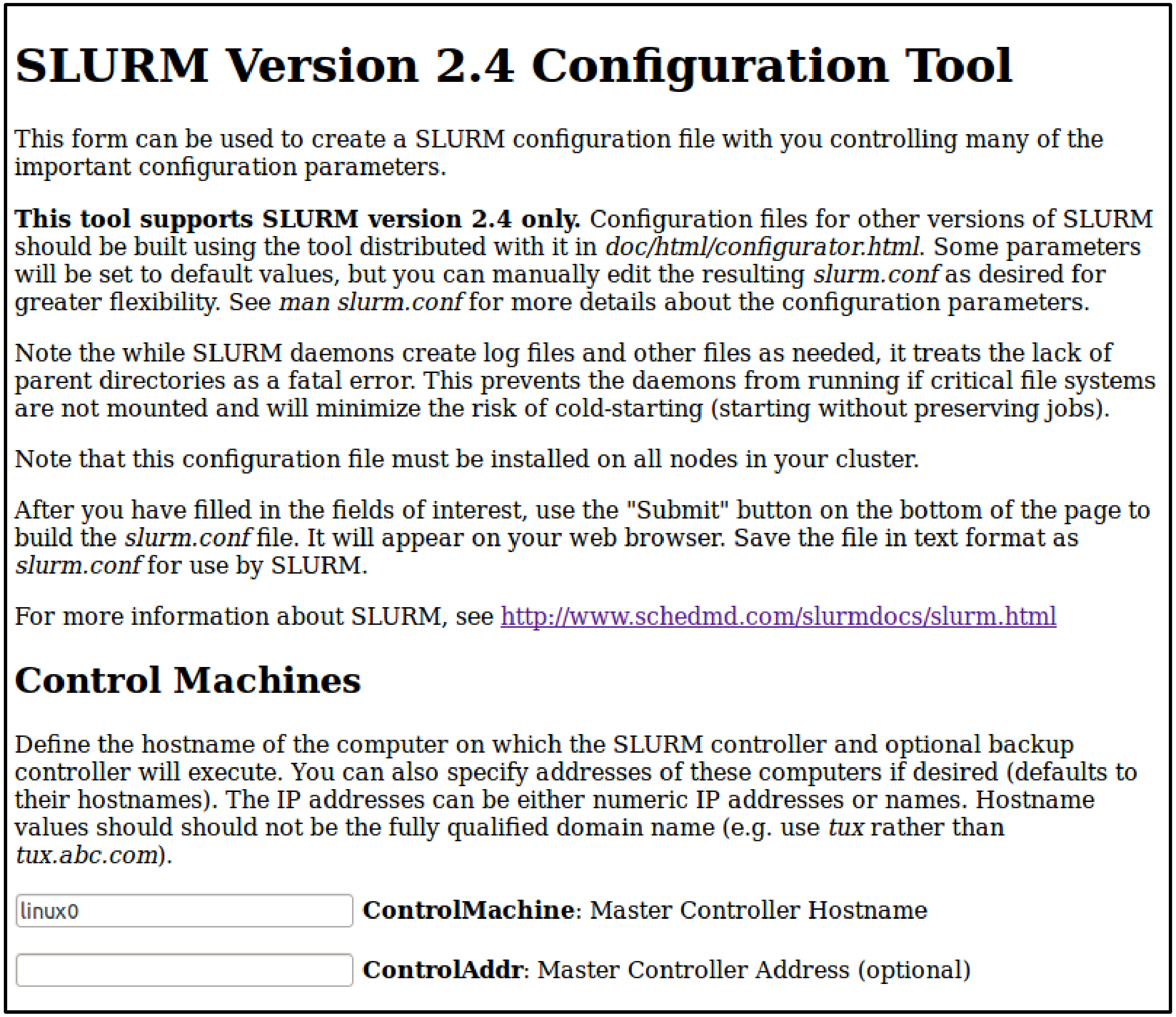
Build and Install Example

```
> cd /tmp/slurm
(get the slurm tar-ball)
> bunzip2 slurm-2.4.1.tar.bz2
> tar -xf slurm-2.4.1.tar
> ls
slurm-2.4.1
> mkdir build
> cd build
> /tmp/slurm/slurm-2.4.1/configure –enable-debug –prefix=/tmp/slurm/install
(identifies header files, libraries, etc.)
> make -j
(builds the SLURM executable and library files)
> make install
(install the files in /tmp/slurm/install)
```

Configuration

- Configuration *slurm.conf* file required on all compute and service nodes
- Most configuration parameters have usable defaults
- At least the nodes in the cluster and grouping into a partition is required
- Web-based tools available to build SLURM configuration file: doc/html/configurator.html and doc/html/configurator.easy.html
  - Open with web browser
  - Set values as appropriate
  - Select the “Submit” button on the bottom
  - Save resulting file in “sysconfdir” location
- See “man slurm.conf” for more help

Configuration Tool



SLURM Version 2.4 Configuration Tool

This form can be used to create a SLURM configuration file with you controlling many of the important configuration parameters.

This tool supports SLURM version 2.4 only. Configuration files for other versions of SLURM should be built using the tool distributed with it in *doc/html/configurator.html*. Some parameters will be set to default values, but you can manually edit the resulting *slurm.conf* as desired for greater flexibility. See *man slurm.conf* for more details about the configuration parameters.

Note the while SLURM daemons create log files and other files as needed, it treats the lack of parent directories as a fatal error. This prevents the daemons from running if critical file systems are not mounted and will minimize the risk of cold-starting (starting without preserving jobs).

Note that this configuration file must be installed on all nodes in your cluster.

After you have filled in the fields of interest, use the "Submit" button on the bottom of the page to build the *slurm.conf* file. It will appear on your web browser. Save the file in text format as *slurm.conf* for use by SLURM.

For more information about SLURM, see http://www.schedmd.com/slurmdocs/slurm.html

Control Machines

Define the hostname of the computer on which the SLURM controller and optional backup controller will execute. You can also specify addresses of these computers if desired (defaults to their hostnames). The IP addresses can be either numeric IP addresses or names. Hostname values should should not be the fully qualified domain name (e.g. use *tux* rather than *tux.abc.com*).

linux0 **ControlMachine**: Master Controller Hostname

ControlAddr: Master Controller Address (optional)

http://www.schedmd.com

Example slurm.conf file

```
# slurm.conf file generated by configurator.html.
# Put this file on all nodes of your cluster.
# See the slurm.conf man page for more information.
#
ControlMachine=linux0
#ControlAddr=
#BackupController=
#BackupAddr=
#
AuthType=auth/munge
CacheGroups=0
#CheckpointType=checkpoint/none
CryptoType=crypto/munge
… CONTENT REMOVED HERE ...
#
FrontEndNodes= State=UNKNOWN
NodeName=nid[00000-01234] CPUs=1 State=UNKNOWN
PartitionName=debug Nodes=nid[00000-01234] Default=YES State=UP
```

Authentication

- MUNGE is SLURM's default authentication and digital signature mechanism
  - http://code.google.com/p/munge/
- Each node in cluster must be configured with a MUNGE key and have the daemons running
- MUNGE generated credential includes
  - User ID
  - Group ID
  - Time stamp
  - Whatever else it is asked to sign and/or encrypt
    - Names of nodes allocated to a job/step
    - Specific CPUs on each node allocated to a job/step, etc.

Authentication

- If desired, multiple Munge daemons can be configured with different keys
  - One key for use within a cluster
  - A second key for communications between clusters

Test Suite

- SLURM includes an extensive test suite that can be used to validate proper operation
  - Includes over 300 test programs
  - Executes thousands of jobs
  - Executes tens of thousands of steps
- Change directory to “testsuite/expect”
- Create file “globals.local” with installation specific information
- Execute individual tests or run “regression” for all tests

Test Suite

- SLURM includes an extensive test suite that can be used to validate proper operation
  - Includes over 300 test programs
  - Executes thousands of jobs
  - Executes tens of thousands of steps
- Change directory to “testsuite/expect”
- Create file “globals.local” with installation specific information
- Execute individual tests or run “regression” for all tests

Test Suite Example

```
> cat globals.local
set slurm_dir  "/home/moe/Desktop/SLURM/install.linux"
set build_dir  "/home/moe/Desktop/SLURM/build.linux"
set src_dir    "/home/moe/Desktop/SLURM/slurm.git"
> regression >qa.tux.jun5
Completions: 315
Failures:        1
Time (sec):      3650
```

- Search output file “qa.tux.jun5” above for “FAILURE” to investigate test failures

System Emulation

- SLURM can easily be configured to emulate various system architectures or system sizes
  - Emulate a Cray or IBM BlueGene/Q on a laptop
    - Underlying database interactions are simulated
  - Emulate 64 cores per node when there are only 4 cores
  - Emulate a 128 node cluster using 4 nodes
    - Run 32 *slurmd* daemons on each compute node
  - Good to test resource allocation logic
  - Not so good to run MPI applications

Agenda

- Role of a resource manager and job scheduler
- SLURM design and architecture
- SLURM commands
- SLURM build and configuration
- SLURM scheduling plugins and development

Scheduling Plugins - Priority

- Prioritizes pending job allocations - Sets initial job priority and can reset priorities (e.g. based upon how over- or under-served a user is)
  - Priority/basic – FIFO scheduling (first-in first-out)
  - Priority/multifactor – Sets priority based upon job age, queue/partition, size, QOS and fair-share
  - Priority/multifactor2 – Ticket-based variant of priority/multifactor

Scheduling Plugins - Select

- Determines which resources are allocated to a job
  - Select/bluegene – Interfaces with IBM Bluegene system software to status system, status jobs, reconfigure network, boot nodes, etc.
  - Select/cray – Interfaces with Cray ALPS software to status system, status jobs, allocated resources, etc.
  - Select/linear – Allocates whole nodes to jobs, optimized for network topology
  - Select/cons_res – Allocates hyperthreads, cores, sockets, boards or nodes to jobs, optimized for network topology
  - Select/serial – Variation of select/cons_res optimized for serial (single CPU) jobs

Scheduling Plugins - Preempt

- Determines which jobs can preempt other jobs
  - Preempt/none – No job preemption
  - Preempt/QOS – Based upon job's QOS (Quality Of Service)
  - Preempt/partition_prio – Based upon job partition/queue priorities

Scheduling Plugins - Sched

- Controls job scheduling order for variation of simple priority order
  - Sched/builtin – FIFO (does nothing)
  - Sched/backfill – Conservative backfill, starts jobs out of order if doing so will not delay the expected initiation time of any higher priority job, spawns thread to periodically compute expected job flow and attempt backfill scheduling, many configuration parameters
  - Sched/wiki – Interface to Maui Scheduler
  - Sched/wiki2 – Interface to Moab Scheduler

Development

- New development should generally be done in plugins rather than the SLURM kernel
- Coordinate new development with SchedMD
  - Design review, code review, joint development
- Follow Linux kernel coding style
  - http://www.schedmd.com/slurmdocs/coding_style.pdf
- Keep documentation current
- Release new work under GPL v2

What's Next

- This just covers the basics of SLURM administration and use
- Lots more documentation available online and in the man pages
  - Help available via email: slurm-dev@schedmd.com
  - Bugzilla available at: http:bugs.schedmd.com
  - Problem reports should include
    - SLURM version
    - Configuration details (output of *“scontrol show config”* or *slurm.conf* file contents)
    - Logs, scripts, etc.