SUN GRID ENGINE ADVANCED ADMINISTRATION

Daniel Templeton
Staff Engineer, Sun Grid Engine
Sun Microsystems, Inc.

Objectives

- Teach you to fish
- Understand how Grid Engine works
- Hands-on experience
- Explore Grid Engine internals

Agenda

- Queues
- Resources
- Jobs
- Parallel Environments
- Resource Quotas
- The qmaster
- The scheduler
- Maintenance
- ARCo

About Me

- 4 years with Grid Engine team
  - 3.5 as software engineer
- 10 years with Sun
- DRMAA spec author
- Software & Java Ambassador
- I drink the Koolaid

About the Machines

- Grid Engine 6.1
- Linux
- Virtual 3-node cluster
- Execution hosts are diskless
  - Use /scratch to write data
- No execd on the master host
- Grid Engine is under /home/sge
  - Owned by sgeadmin

About this Class

- I want to tell you want you want to hear
  - Ask lots of questions
  - Feel free to interrupt
- I assume you:
  - Are a Grid Engine admin
  - Are not an expert Grid Engine admin
  - Able to use *vi*
- Learn by doing

Write down specific questions you hope to get answered

About the Exercises

- Lots of them
  - More fun than listening to me preach
- Not step by step
  - Assume you know what you're doing and/or can read
  - Leave out some details
  - Rely of what we've learned
    - Some rely on what we haven't learned
- General assumption is that you're operating as root
  - Or as manager/operator
- Assume you're starting from scratch unless noted

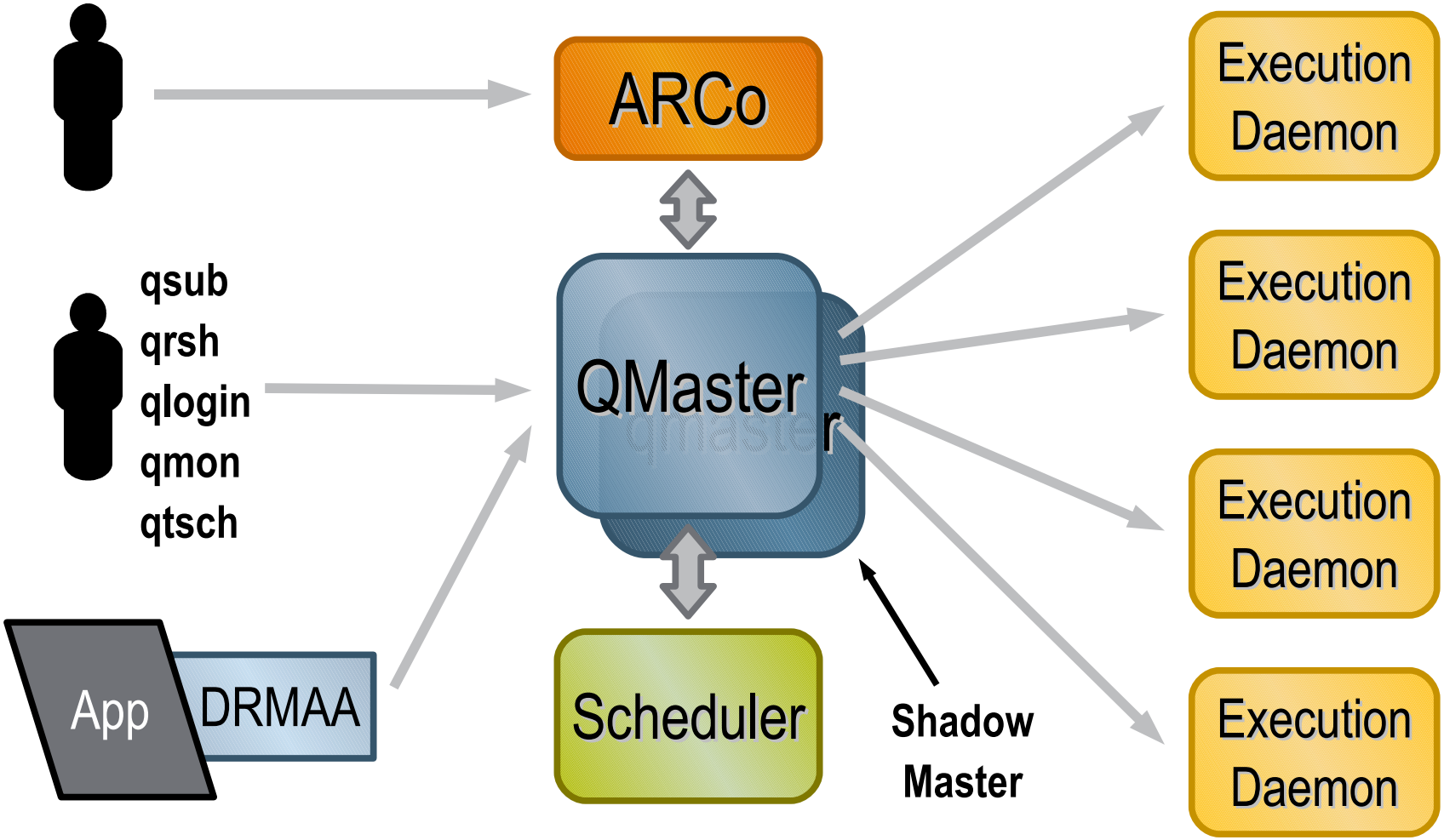
About You

Name, employer, job, Grid Engine experience

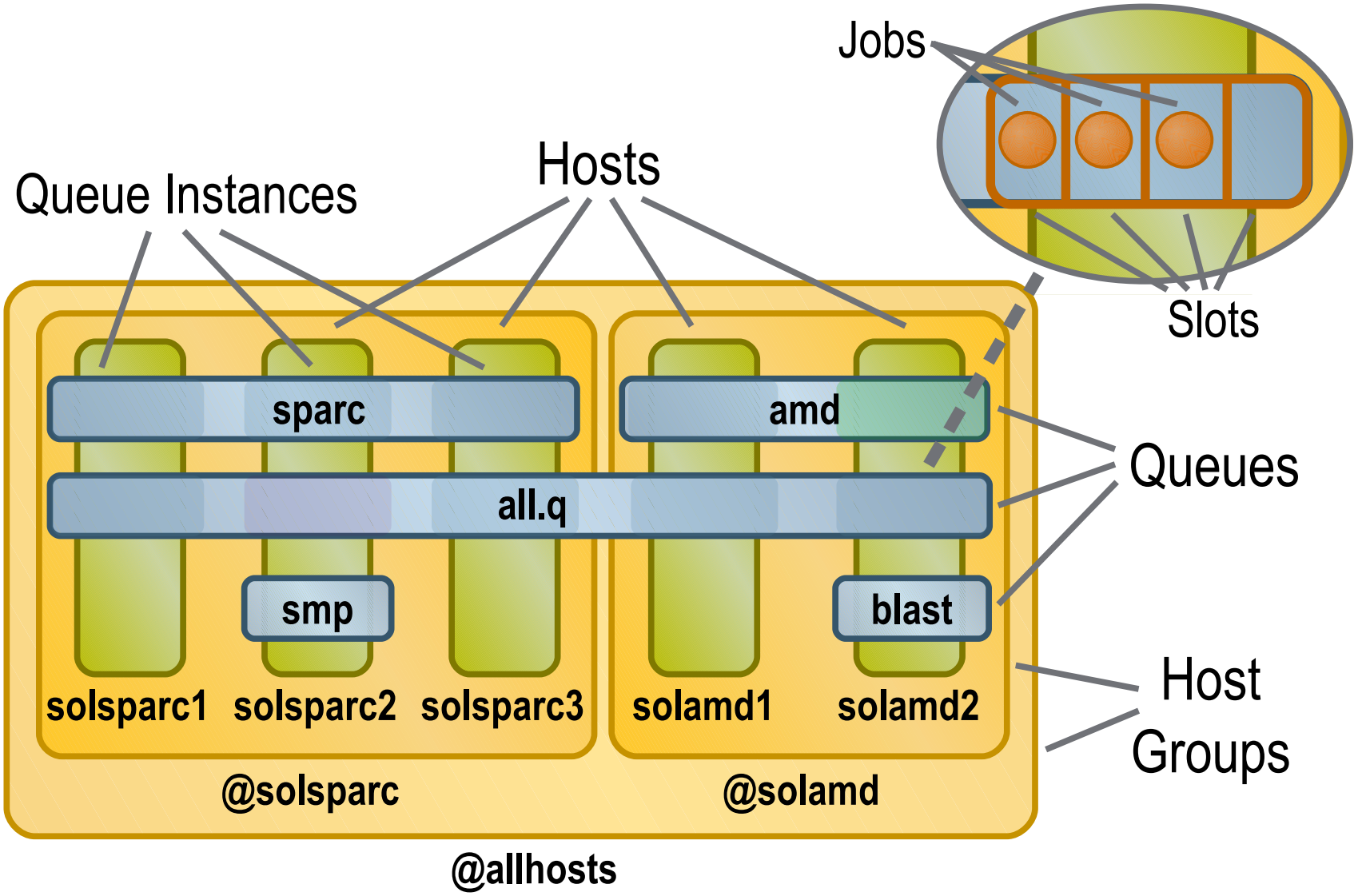
About Grid Engine

- Over 10,000 deployments
- Product: Sun Grid Engine
  - aka “N1 Grid Engine”
  - http://www.sun.com/gridware
  - Same source files as open source
- Open source project: Grid Engine
  - http://gridengine.sunsource.net
  - ~1 million LoC
- Policy, Configurability, & Community

N1 Grid Engine Components



Example Configuration



What Is a Queue?

- 6.x: “queue” == “cluster queue”
  - 5.x: “queue” == “queue instance”
- CQ_Type CULL
  - A list of attribute values
  - A list of queue instances
    - QI_Type CULL
      - A list of set attribute values
- A convenient handle to several queue instances
- Global queue settings
  - Queue instances can override

Grown-up Admin

- *qmon* is for sissies
- Real men use command-line tools
- Really manly men use *qconf -[admr]attr*
- Jack Bower uses *qconf -purge*
- We will do all our admin with *qconf*
  - We'll use scriptable commands wherever possible

Using qconf -[admr]attr

- Add, delete, modify, replace
- Primarily for list attributes
  - Work for non-list attrbutes, except delete
    - Add, modify, and replace → modify
- -[admr]attr obj_name attr_name value[=v] obj_id_lst
- Add host, host1, to hostgroup, @allhosts
  - qconf -aattr hostgroup hostlist host1 @allhosts
- Change np_load_avg to 2 in load_thresholds in all.q
  - qconf -mattr queue load_thresholds np_load_avg=2 all.q

Modify Versus Replace

- -mattr changes the value of a setting
  - > load_thresholds np_load_avg=1.75,mem_used=2G
  - > qconf -mattr queue load_thresholds np_load_avg=2 all.q
  - > load_thresholds np_load_avg=2,mem_used=2G
- -rattr replaces the entire list of settings
  - > load_thresholds np_load_avg=1.75,mem_used=2G
  - > qconf -rattr queue load_thresholds np_load_avg=2 all.q
  - > load_thresholds np_load_avg=2
- -mattr becomes -aattr if no values

Replace Versus Purge

- Replace is for list attributes
  - Not limited to queues
- Purge is for queue instances
  - Removes overridden queue instance settings
  - Only for queue *instances*
- Remove host-specific slots settings for *host1* in *all.q*
  - qconf -purge queue slots all.q@host1

Exercise: Man Up!

- Create a new PE called *dummy*
- Do all of the following without using qmon or -?q:
  1. Add *dummy* to *all.q*
  2. Remove *make* from *all.q*
  3. Make *make* the only PE for *all.q*
  4. Change the load_thresholds setting for *all.q* to np_load_avg=4
  5. Remove all slots settings from *all.q* for a single queue instance
  6. BONUS: Add a slots settings for *all.q* for a single queue instance

Solution: Man Up!

- qconf -sp make | awk '$1 == "pe_name" {print $1, "dummy"} $1 != "pe_name"' > /tmp/dummy; qconf -Ap /tmp/dummy; rm /tmp/dummy
- Do all of the following without using qmon or -[mM]q:
  1. qconf -aattr queue pe_list dummy all.q
  2. qconf -dattr queue pe_list make all.q
  3. qconf -rattr queue pe_list make all.q
  4. qconf -mattr queue load_thresholds np_load_avg=4 all.q
  5. qconf -purge queue slots all.q@host1
  6. BONUS: qconf -aattr queue slots '[host1=4]' all.q

Custom Signals

- By default
  - Suspend = SIGSUSP
  - Resume = SIGCONT
  - Terminate = SIGKILL
- suspend_method, resume_method, terminate_method
  - Signal name or number or absolute path to an executable
- notify
  - Send SIGUSR1/SIGUSR2 notify seconds before suspending/terminating a -notify job
  - Overridden by NOTIFY_SUSP & NOTIFY_KILL

Load Thresholds

- load_thresholds
  - list of resource=value pairs
  - relational op taken from complex
  - defaults to np_load_avg=1.75
- When *complex op value* is true
  - Stop accepting new jobs
  - Set queue into alarm state
- Used to prevent oversubscription

Suspend Thresholds

- suspend_thresholds
  - list of resource=value pairs
  - relational op taken from complex
  - defaults to NONE
- When *complex op value* is true
  - Suspend nsuspend jobs
  - Every suspend_interval, suspend nsuspend more
- When *complex op value* is false again
  - Resume nsuspend jobs
  - Every suspend_interval, resume nsuspend more
- Used to prevent resource hogging

Queue Priority

- priority
  - -20 to 20
    - Lower is higher
  - UNIX nice value
- Nothing to do with scheduling
- Nothing to do with *qsub -P*

Exercise: Priority Queues I

- Naïve approach to priority queues
- Create three queues, *low.q*, *regular.q*, and *high.q*
  - All on a single host
- Set slots to number of CPUs
  - Oversubscription handled by priority
- Set load_thresholds to NONE
  - Has to do with load adjustments...
- Set *high.q* priority to -20
- Set *low.q* priority to 20
- Submit *worker.sh 20* jobs to all three queues

Solution: Priority Queues I

- Did you remember to use *qsub -q*?
- You should see the *high.q* jobs finish first and the *low.q* jobs finish last
- Leaves scheduling up to the OS
- Oversubscription issues
- No disincentive to always using *high.q*
- We can do better

Queue Limits

- real time, CPU time, file size, core file size, heap size, stack size, virtual memory space size, total memory size
- Hard or soft
  - Behavior depends on limits
- See **queue_conf(5)** and **setrlimit(2)** man pages

Queue Run Time Limits

- h_rt → SIGKILL when exceeded
- s_rt → SIGUSR1 when exceeded
  - SIGKILL notify seconds later
- h_cpu → SIGKILL when exceeded
  - RACE: OS may send SIGXCPU first
- s_cpu → SIGXCPU when exceeded
  - Used with h_cpu to send a warning
- Specified in seconds

Exercise: Priority Queues II

- Set notify to 60 for *regular.q*
- Set a soft wall clock (real time) limit for *regular.q*
  - 24:00' = 86400”
- Set a soft CPU time limit for high.q
  - 9' = 540”
- Set a hard CPU time limit for high.q
  - 10' = 600”

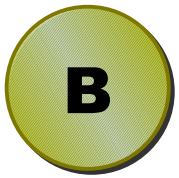
Solution: Priority Queues II

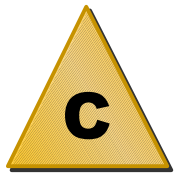
- qconf -rattr queue notify 60 regular.q
- qconf -rattr queue s_rt 86400 regular.q
- qconf -rattr queue s_cpu 540 high.q
- qconf -rattr queue h_cpu 600 high.q
- Users now encouraged to use *low.q*
- We can still do better

Grid Engine Resources

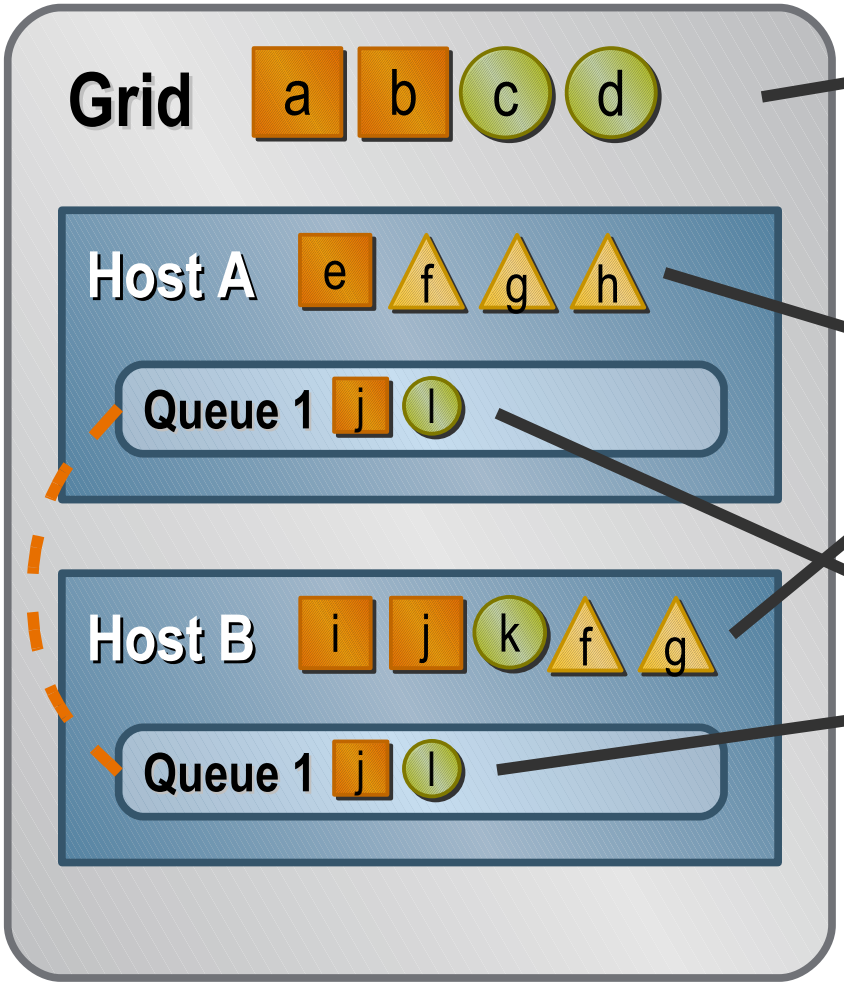
Three types of resources

- Static resources
  - Strings
  - Numbers
  - Booleans
- Countable resources
  - Licenses
  - Memory
  - Storage
  - etc.
- Measured resources
  - System load
  - Idle time
  - etc.





Resource Configuration



- Global resources
  - Apply to all queues and hosts
- Host resources
  - Apply to all queues on host
- Queue resources
  - Apply to this queue on all hosts

Sharing Resources

- Move them one level up
  - > Host resources are shared by queues
  - > Global resources are shared by hosts
    - – And by queues
- Resource usage totaled across level
  - > mem_total is sum from all jobs in all queues on host
- slots queue attribute is really a resource
  - > Make slots a host resource
  - > Sets maximum number of slots for that host
  - > Prevents oversubscription by multiple queues

Scaling Resources

- All machines aren't created equal
- load_scaling host config attribute
  - resource=factor
- Bad example:
  - 2-node grid, 1 8-way, 1-1way
  - 1-way load scaling: load_avg=8
  - np_load_avg

Exercise: Priority Queues III

- Add slots as a host resource for the host
  - Equal to number of CPUs
- Submit a bunch of *worker.sh 20* jobs to the queues
  - Be sure to submit to a specific queue

Solution: Priority Queues III

- qconf -aattr exechost complex_values slots=4 host
- Solved oversubscription problem
- Created a new problem
  - Jobs are scheduled FIFO in our config
  - Priority is handled by OS after scheduling
  - Run-time priority is overridden by non-priority scheduling
- Could slightly oversubscribe
  - Better, but not good
- We can do better yet

Resource Priorities

- Resources can be assigned a priority
  - Called resource urgency
- Jobs inherit resource urgency
  - Multiple resources are summed
  - Numerical resources x number requested
    - x number of slots for parallel jobs
- Part of urgency policy
  - Deadline time
  - Wait time

Exercise: Priority Queues IV

- Create a new resource called *high_priority*
  - Requestable (non-consumable) boolean
  - Urgency = 100
- Add *high_priority* to *high.q*
- Create a new resource called *low_priority*
  - Requestable (non-consumable) boolean
  - Urgency = -100
- Add *low_priority* to *low.q*
- Show urgency information for all jobs

Solution: Priority Queues IV

- echo “high_priority hp BOOL == YES NO FALSE 100” >> /tmp/MC
- echo “low_priority lp BOOL == YES NO FALSE -100” >> /tmp/MC
- qconf -Mc /tmp/MC; rm /tmp/MC
- qconf -aattr queue complex_values hp=TRUE high.q
- qconf -aattr queue complex_values lp=TRUE low.q
- qsub -l hp $SGE_ROOT/examples/jobs/worker.sh
- qsub $SGE_ROOT/examples/jobs/worker.sh
- qsub -l lp $SGE_ROOT/examples/jobs/worker.sh
- qstat -urg

Solution: Priority Queues IV

- Jobs now scheduled in priority order
- No oversubscription
- Still have problems
  - Regular jobs could end up in *high.q* or *low.q*
  - Non-premptive ⟶ priority inversions

Requestable Versus Forced

- Requestable complex
  - Can be requested by job, e.g. *-l rp[=true]*
  - requestable: YES
- Forced complex
  - Must be requested by job
  - requestable: FORCED
- Queue or host with forced complex
  - Only jobs requesting that resource
- Prevents misuse of resources

Exercise: Priority Queues V

- Make *high_priority* and *low_priority* forced
- Submit some regular jobs
  - Confirm that they only go to *regular.q*
- What reason does the scheduler give for not scheduling the pending jobs?

Solution: Priority Queues V

- qconf -mc
  - Sometimes interactive is easier
- qstat -j <job_num>
  - Scheduler messages at the bottom
- Pretty reasonable solution
  - Still not preemptive
  - Why not oversubscribe low priority jobs?
    - Essentially background tasks

Subordinate Queues

- subordinate_list queue attribute
  - List of queue=value pairs
  - Defaults to NONE
- When this queue has *value* or more jobs, suspend the subordinate queue
  - Suspends all jobs in subordinate queue
- When this queue has fewer than *value* jobs, resume the subordinate queue
- If *value* is not given, *value* = slots

Exercise: Priority Queues VI

- Delete the slots complex from the host
- Make *regular.q* and *low.q* subordinate to *high.q*
- Submit some jobs
  - What happens when a high priority job is scheduled?

Solution: Priority Queues VI

- qconf -dattr exechost complex_values slots=4 host
- qconf -rattr queue subordinate_list regular.q=1 high.q
- qconf -aattr queue subordinate_list low.q=1 high.q
- Job running in *high.q*, suspends *low.q* and *regular.q*
  - Also called an “express queue”
  - Could remove the priority for *high.q*
- *low.q* and *regular.q* are decoupled
  - *low.q* is now for oversubscription
- QED

Consumables

- Consumable complex
  - Decremented by each requesting job
    - By amount requested
  - consumable: YES
  - default is amount to decrement for non-requesting jobs
- Represents fixed resources that can be consumed
  - When 0, no more jobs scheduled there
    - Unless default = 0
  - Incremented when jobs finish
    - By amount requested or default
- slots is a consumable

Exercise: License To Ill

- Create a consumable called *license1*
  - Requestable, consumable int
- Create a consumable called *license2*
  - Requestable, consumable int, default=1
- Add *license1* to the global configuration
- Create a new queue called *sfw.q*
- Add *license2* to *sfw.q*

Solution: License To Ill

- echo “license1 l1 INT <= YES YES 0 0” >> /tmp/MC
- echo “license2 l2 INT <= YES YES 1 0” >> /tmp/MC
- qconf -Mc /tmp/MC; rm /tmp/MC
- qconf -aattr exechost complex_values l1=10 global
- qconf -aq sfw.q
- qconf -aattr queue complex_values l2=4 sfw.q
- Jobs using license1 can run anywhere
- Jobs using license2 run in *sfw.q*
  - If forced, can't request with -q

Load Sensors

- Custom complex monitors
- Any executable
  - Script, binary, Java application, etc
- Simple input/output contract
  - \n → output complex values
    - begin\n
    - host:name:value\n
    - end\n
  - quit\n → exit.

Simplest Load Sensor

```
#!/bin/sh
myhost=`uname -n`
while [ 1 ]; do
      # wait for input
      read input
      result=$?
      if [ $input = quit ]; then
            exit 0
      fi
      echo begin
      echo "$myhost:complex:3"
      echo end
done
exit 0
```

Configuring Load Sensors

- load_sensor host config attribute
  - Comma-delimited list
  - Absolute execution paths
- Can be global
  - Run on every host
- Will be restarted
  - If it dies
  - If executable is modified

Load Sensor Scope

- Unrelated to complex scope
- Host load sensor can report a host complex
- Global load sensor can report a host complex
- Host load sensor can report a global complex
- Global load sensor shouldn't report a global complex
  - > “Global” for load sensor means “runs on each host”
  - > Host reports will conflict with each other

Exercise: Use the Force

- Create a new complex called *logins*
  - Non-requestable, non-consumable int
- Create a new complex called *unique_logins*
  - Non-requestable, non-consumable int
- Create a load sensor for both complexes
- Add the load sensor to the global host config
  - Allow a minute for the setting to propogate
- View the complex's status

Solution: Use the Force

- echo “logins al INT <= NO NO 0 0” >> /tmp/MC
- echo “unique_logins ul INT <= NO NO 0 0” >> /tmp/MC
- qconf -Mc /tmp/MC; rm /tmp/MC
- See utf.pl for my load sensor
- qconf -mconf
- qhost -F al,ul
- Complexes could be used for scheduling decisions
  - We'll talk about that later...

Queue Sorting

- seq_no queue attribute
  - Order for qstat output
  - Used by scheduler
- By default, scheduler breaks “load” tie with seq_no
  - Lets you favor queues
    - Faster
    - Cheaper
    - Not yours...
- Can be reversed
  - Useful to create fill-up order

Job Reruns

- If a host crashes, non-checkpointing jobs are lost
  - > Fail when host restarts
- If the rerun queue attribute is TRUE
  - > Failed jobs will be restarted
  - > As soon as execd cleans up from failed jobs
- Default is FALSE
- Jobs can override with *qsub -r y|n*

Transfer Queues

- Concept, not a feature
- Several pieces
  - “Transfer” queue
  - Load sensor to monitor remote site
  - Starter method to send jobs to remote site
  - Terminate/suspend/resume method for remote site
- Howto's on gridengine.sunsource.net
  - Transfer-queue Over Globus (TOG)

Exercise: Early Warning Signs

- You want to protect the *sfw.q* such that only certain users can submit jobs there
- Because this is a policy change and you're a nice person, you want to offer a friendly denial message

Exercise: Early Warning Signs

- First, limit access to *sfw.q*
  - Create an ACL called *sfw_users*
    - qconf -au username sfw_users
  - Add *sfw_users* to *sfw.q*
    - qconf -aattr queue user_lists sfw_users sfw.q
- Next, give denied job somewhere to go
  - Create a new queue called *denied.q*
    - qconf -sq denied.q
  - Add sfw_users to *denied.q* as an XACL
    - qconf -aattr queue xuser_lists sfw_users denied.q
  - Add license2 to *denied.q*
    - qconf -aattr queue complex_values l2=999 denied.q

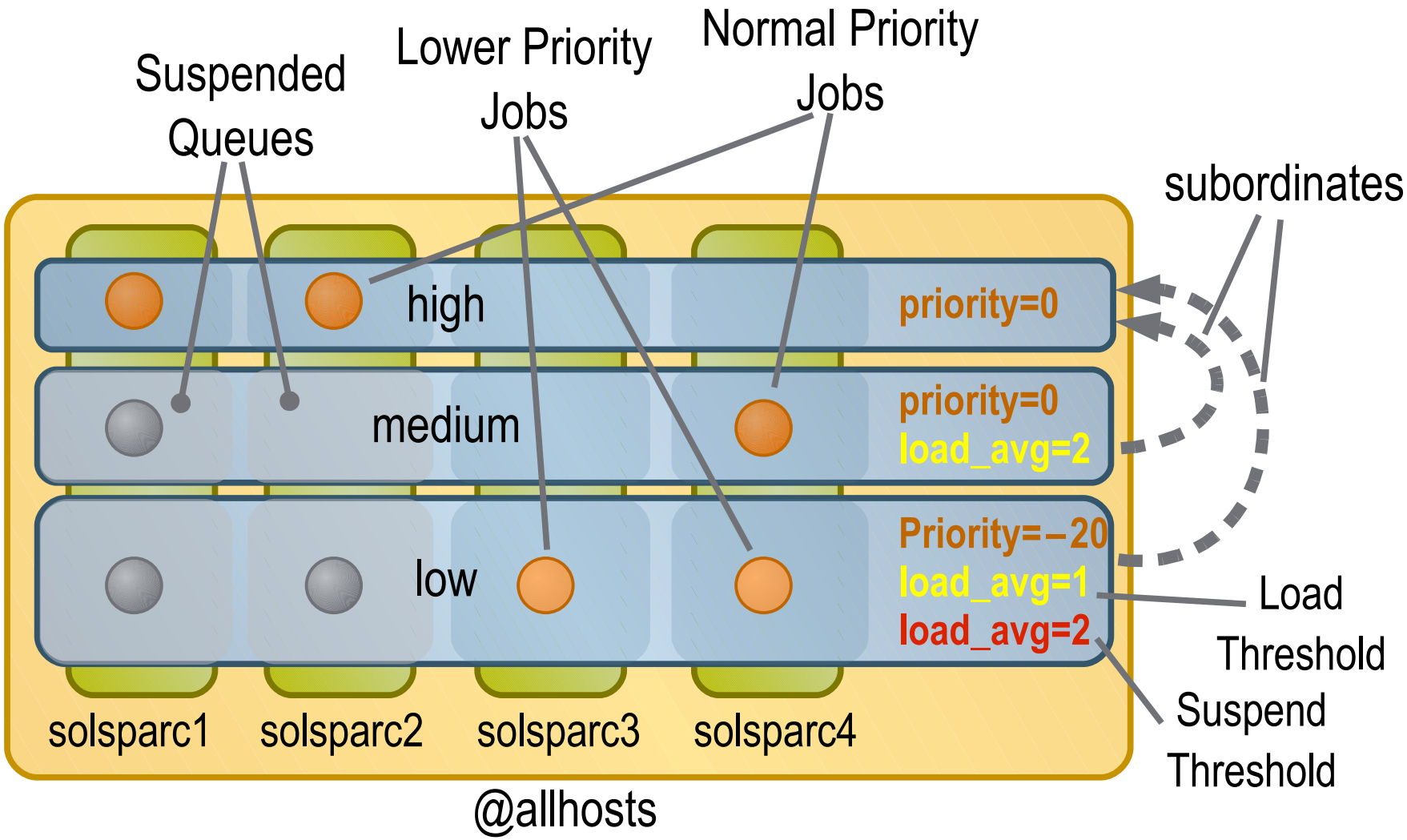
Exercise: Early Warning Signs

- Finally, set a starter method for *denied.q*
  - > Write a starter script that output a warning:

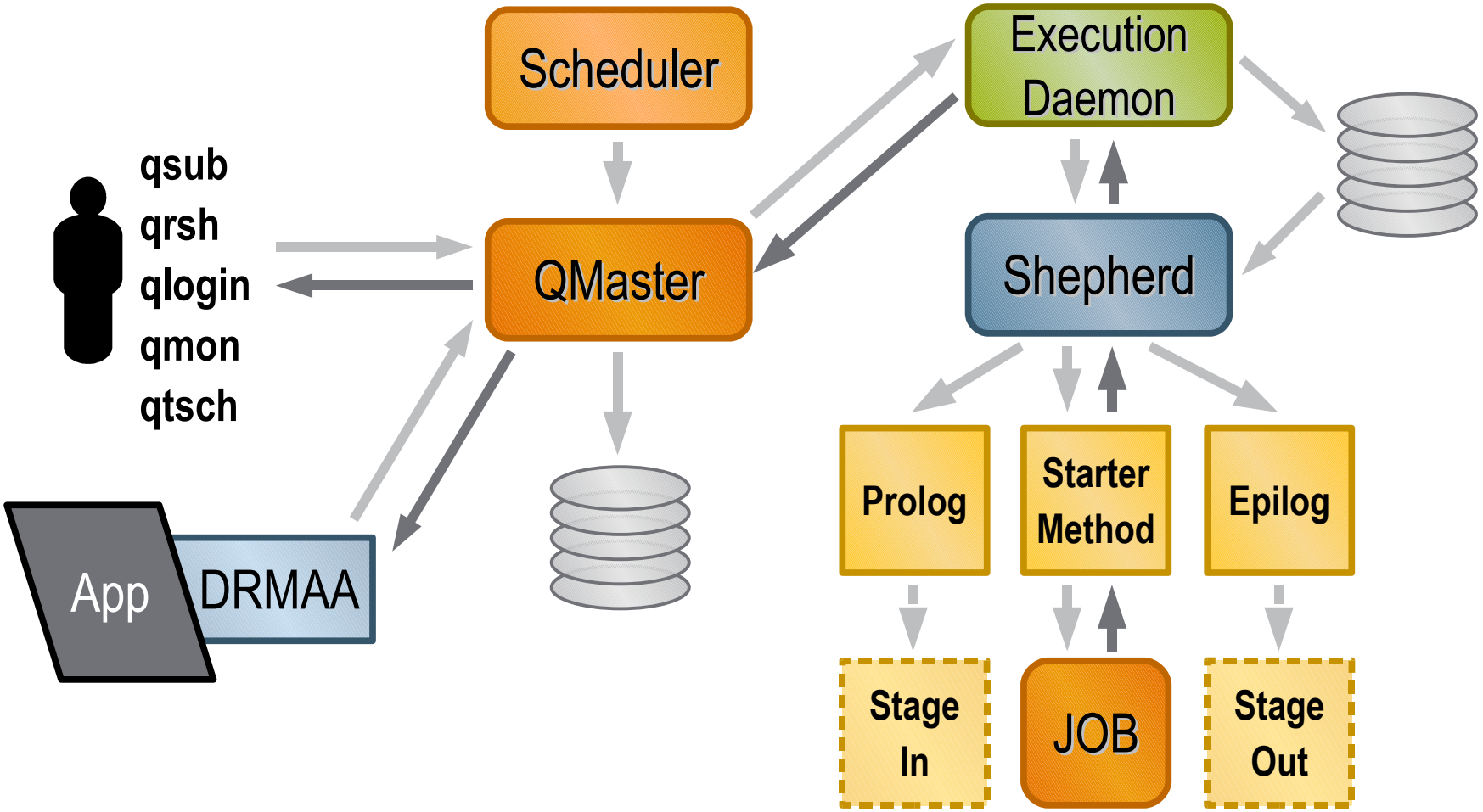
```
#!/bin/sh

echo Access denied to the sfw queue
exit 100
```

  - > Set the starter script for *denied.q*
    - – qconf -rattr queue starter_method `pwd`/ews.sh denied.q
- Try it out

Priority Queue Example



Starting a Non-parallel Job



What is a Job?

- JB_Type CULL
  - > Job description
  - > Task list
    - – Always at least one task
    - – JAT_Type CULL
      - – Task description
- Created by submitter
- Stored by qmaster
- Passed to execd

Binary Versus Script

- Two important differences
  - Script sends the entire file; binary sends a path
  - Script interpolates embedded options; binary doesn't
- Does not affect job environment
- Script by default with CLI tools
- Binary by default with DRMAA
- 150k job script, 1000 submits
  - 30 seconds as script
  - 20 seconds as binary

Shell Versus No Shell

- Exec shell -c job or exec job directly
  - > unix_behavior or posix_compliant
  - > Overridden by script_from_stdin
- Does not escape the starter method
- Changes binary error behavior
  - > Shell: exit status=1
  - > No shell: job set to error state
  - > *Script: won't submit*
- No Script: environment comes directly from execd

Exercise: Three Out of Four Ain't Bad

- Run toofab.sh
- Note that 3 out of four of the job tasks fail
- Run it again. Notice a pattern?
- T-shirt for the first person to tell me what's going wrong and why

Inherited Job Environment

- execd → shepherd → shell → job
  - shepherd overwrites environment with submit settings
  - shell overwrites environment with user settings
- Options you care about get set explicitly
  - Options you don't care about get inherited
  - Can lead to strange errors
- INHERIT_ENV execd parameter
  - Defaults to TRUE
  - Should always be set to FALSE

Inherited Shared Library Path

- Many jobs need $SGE_ROOT/lib/$ARCH
  - Default assumption: inherited from execd env
- What happens when INHERIT_ENV=FALSE?
  - $SGE_ROOT/lib/$ARCH isn't in shared lib path
- SET_LIB_PATH execd param
  - Defaults to FALSE
  - Should be set to TRUE
    - If INHERIT_ENV=FALSE
    - Grid Engine 6.0
    - Grid Engine 6.1 not on Solaris or Linux, unless DRMAA

Shared Library Path in 6.1

- Solaris and Linux hosts
- Shared library is not set by settings.[c]sh
- Set through RUNPATH
  - */opt/SUNWspro/bin/cc ... -R $SGE_ROOT/lib/$ARCH ...*
- Not for DRMAA Java™ language binding
  - Must set shared lib path to include $SGE_ROOT/lib/$ARCH
- Maybe for DRMAA C language binding
  - Apps should compile in a run path
  - Most will need $SGE_ROOT/lib/$ARCH in the lib path

Solution: Three Out of Four Ain't Bad

- “Whoever” started three of the execd's had USER_DEBUG=true in his env
- Set INHERIT_ENV=FALSE
- Now the process is missing libdrmaa
- Set SET_LIB_PATH=TRUE

Default Settings

- Default job submission settings
  - $SGE_ROOT/$SGE_CELL/common/sge_request
  - $HOME/.sge_request
  - $PWD/.sge_request
- Default qstat settings
  - $SGE_ROOT/$SGE_CELL/common/sge_qstat
  - $HOME/.sge_qstat
- Overridden by runtime parameters

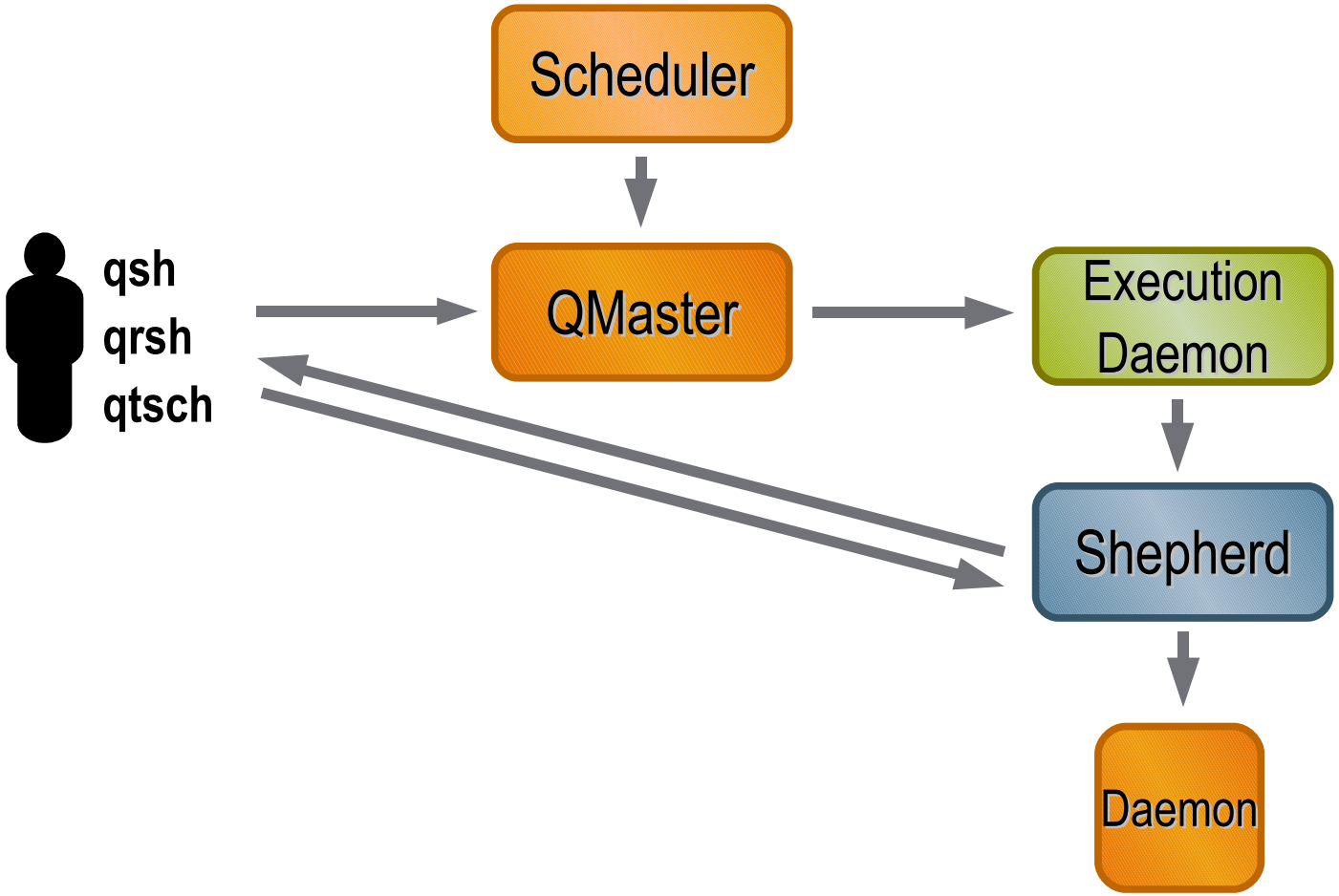
qtcsh and DRMAA Job Category

- qtcsh
  - tcsh with built-in job submission
  - Commands in qtask file automatically “qrsh'ed”
    - $SGE_ROOT/$SGE_CELL/common/qtask
    - $HOME/.qtask
- DRMAA job category
  - Uses qtask file to translate category to options
- qtask Format
  - <command> <options>
  - e.g. mozilla -now y -o /dev/null -j y -b y -shell y
  - !cmd in global file cannot be overridden

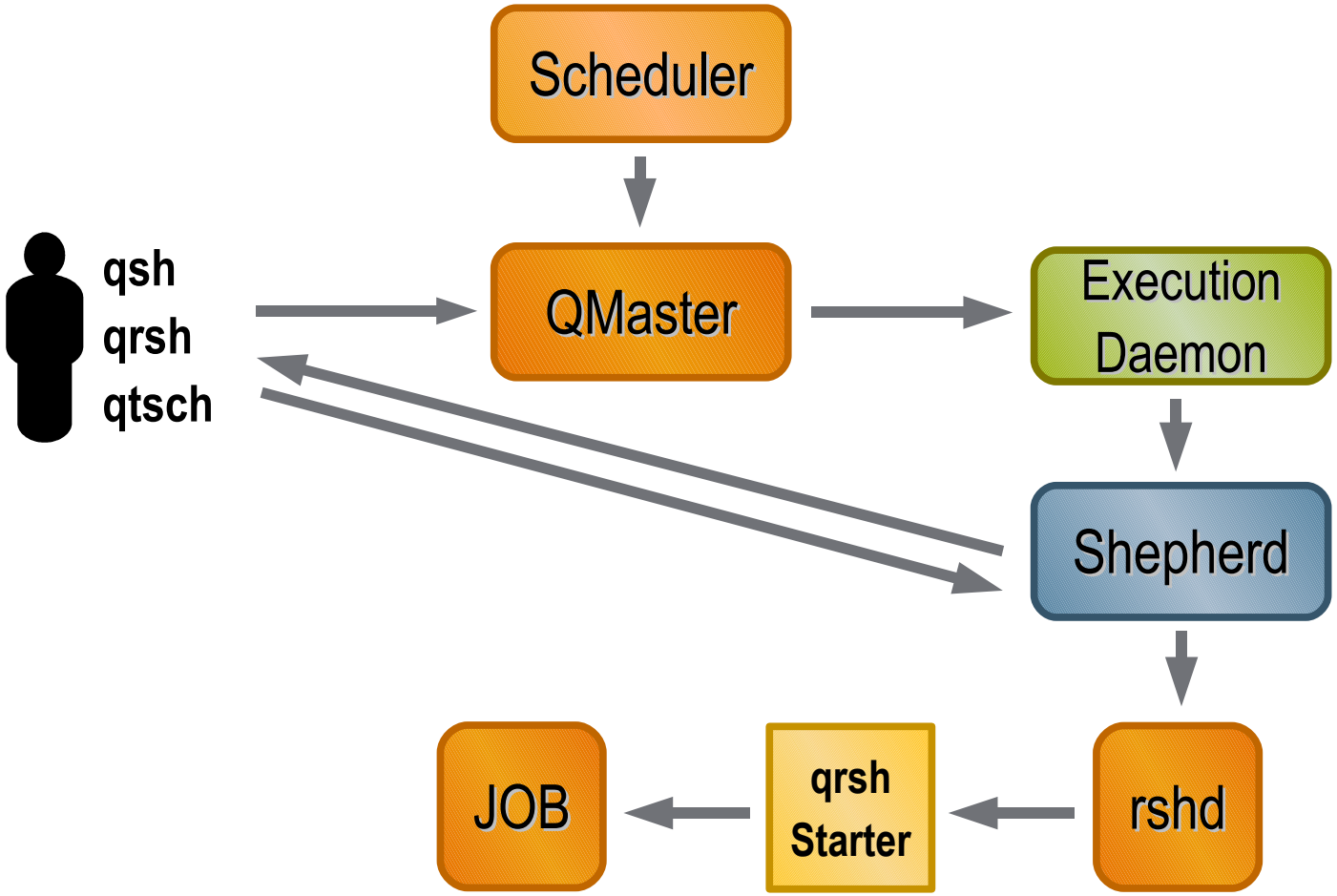
Interactive Jobs

- qsh, qlogin, qrsh, qtcsh
  - *qsub -now y*
  - qsh xhosts back an xterm
- Only run in qtype INTERACTIVE queue
  - Default is BATCH INTERACTIVE
- Use custom rlogin, rsh, rshd
  - Required for control and accounting
  - $SGE_ROOT/utilbin/$ARCH

Starting an Interactive Login



Starting an Interactive Job



Exercise: the Real Slim Shady

- Run trrs <userid> as root
- Confirm that *trss.sh* is running as root
  - > ps -fp `cat /tmp/<job_id>.1.all.q/output`
  - > **Should** be running as userid
- (Pretend trss is a daemon that run jobs by proxy)
- How do we fix it?

The Shell Game

- shell_start_mode attribute
  - How to determine which shell to use to start the job
- unix_behavior
  - Act like a shell, i.e. look at the #! line
- posix_compliant
  - Always use the shell attribute
- script_from_stdin
  - While still root, read in script
  - Feed script to shell via stdin
  - Uses the shell attribute

Who's Your Daddy?

- Jobs get started by:

	Script	Binary
unix_behavior	Shell named by #! line of script	Shell named by queue's shell attribute
posix_compliant	Shell named by queue's shell attribute	Shell named by queue's shell attribute
script_from_stdin	Shell named by queue's shell attribute	Shell named by queue's shell attribute*

* script_from_stdin is ignored

Unless, Of Course...

- Shell can be overridden by *-S <shell>*
  - Command argument
  - Embedded option – very common
- But only if posix_compliant or script_from_stdin

Enter the Starter Method

- Overrides shell_start_mode
- Arbitrary script used to start the job
- Simplest form:

```
#!/bin/sh
$*
```

- Runs as job owner

Solution: the Real Slim Shady

- Create a custom starter method
  - Start job with su $2 -c $1
  - $0 is the starter method script
- trss.sh tries to use the tmp dir
  - Belongs to job owner → root
- Change the tmpdir's ownership in the starter
  - chown -R $2 $TMPDIR
  - Could also use $TMP

Exercise: Musical Environments

- Make sure shell_start_mode is posix_compliant
  > qconf -rattr queue shell_start_mode posix_compliant all.q
- Run me.sh
- Note that is fails
- Run me.**c**sh
- Why does *me.csh* fail as a job, but not as a command?

Specifying Login Shells

- Login shells execute more command files
  - Example: csh
    - As login shell
      1. /etc/.login
      2. $HOME/.cshrc
      3. $HOME/.login
    - Not as login shell
      1. $HOME/.cshrc
- Job started from command shell if
  - shell_start_mode is posix_compliant or script_from_stdin
  - Job is not binary
  - Shell is in login_shells host config attribute

Solution: Musical Environments

- $HOME/.login sets the $LOGIN_SHELL env var
- csh is missing from login_shells list
  - .login isn't getting executed
- Add csh to login_shells list
  - Wait about 30 seconds for the config to propagate

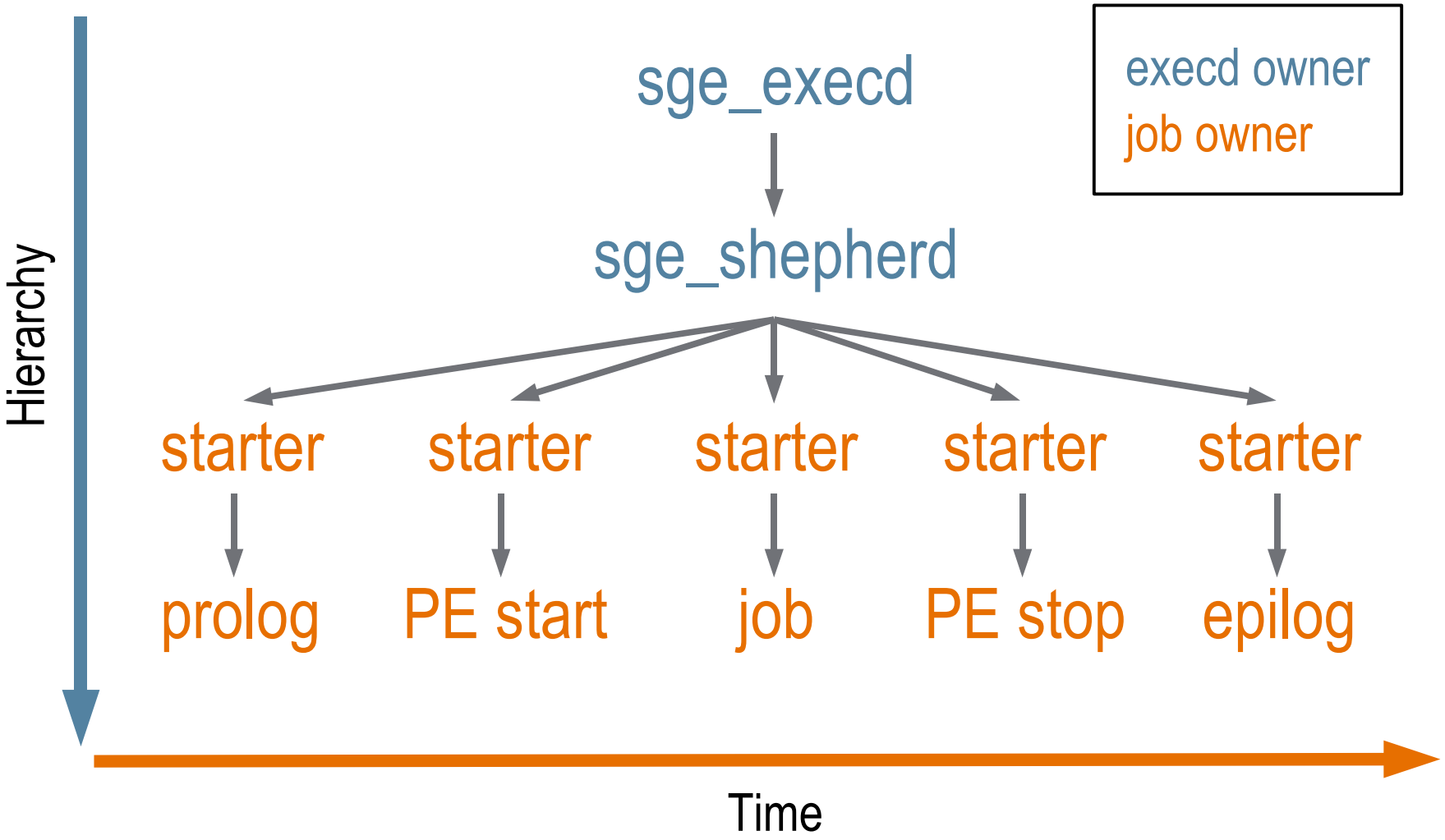
Magic Exit Codes

- Exit code 99
  - Reschedule this job
  - Used by a job to say that it doesn't like where it's running
  - Ignore with FORBID_RESCHEDULE execd param
- Exit code 100
  - Put this job in error state
  - Used by a job to indicate things are foobar
  - Ignore with FORBID_APPERROR exec param

Prolog and Epilog

- prolog and epilog
- Uses same starter rules as job
  - > Including starter_method
  - > Except shell_start_mode is always unix_behavior
- Gets same env as job
- Started by shepherd as job owner
- Queue overrides host conf overrides global conf
  - > Unless queue setting is NONE
- Runs before/after PE startup/shutdown

Process Hierarchy



File Staging

- Delegated file staging
  - Only file staging Sun Grid Engine “provides”
  - Mechanism to pass DRMAA file staging info to prolog and epilog
- Do-it yourself
  - Need to discover file paths
    - Environment variables
    - Temp file
- Prolog stages input in
- Epilog stages output and error out and deletes input

Exercise: Fruits of Your Labor

- Create a new queue called *staged*
- Write a prolog and epilog
  - Use $FOYL_INPUT and $FOYL_OUTPUT
  - Stage to $CWD
- Add prolog and epilog to queue
- Run *foyl.sh*

Solution: Fruits of Your Labor

- prolog:

```
#!/bin/sh

if [ "$FOYL_INPUT" != "" -a \
     "$FOYL_OUTPUT" != "" ]; then
   cp $FOYL_INPUT ./foyl.input
else
   exit 100
fi
```

Solution: Fruits of Your Labor

- epilog:

```
#!/bin/sh

if [ "$FOYL_INPUT" != "" -a \
     "$FOYL_OUTPUT" != "" ]; then
   cp ./foyl.output $FOYL_OUTPUT
   rm ./foyl.*
else
   exit 100
fi
```

Prolog/Epilog Exit Codes

- 99 and 100 have same meaning as for jobs
- 0 is success
- Anything else is failure
  - **Queue** is put in error state!
  - Prologs & epilogs shouldn't “fail” lightly

Migration and Checkpointing

- Checkpointing environments
  - > User-level checkpointing
  - > Kernel-level checkpointing
  - > External to Grid Engine
    - – Initiated by configured commands
- Checkpointing jobs can be migrated
  - > Execution daemon goes missing
  - > Host overloaded
  - > Job or queue suspension
- qsub -ckpt checkpoint

The Magic of qalter

- qalter lets you change job parameters
  - After submission
  - Before being scheduled
- Supports most qsub parameters
- Common usage pattern:

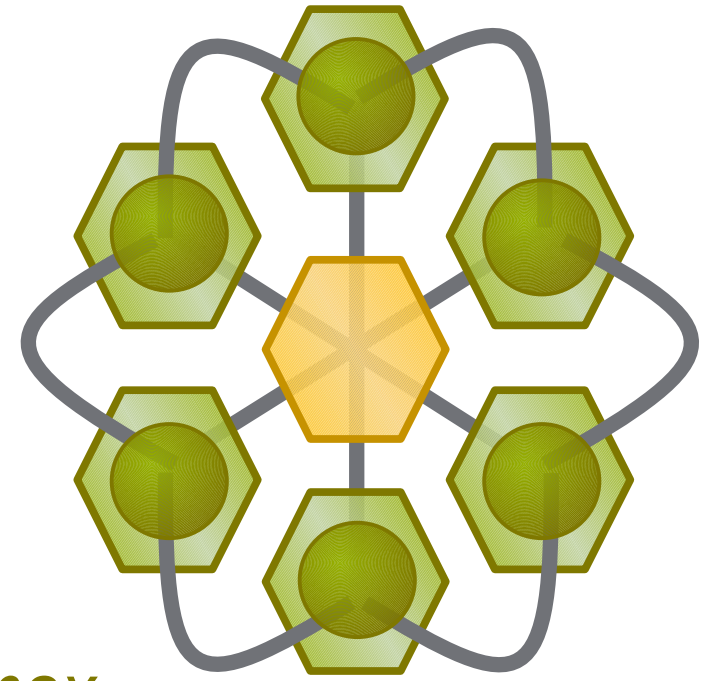
```
% qsub -h -N job1 ...
...
% qsub -h -N jobN ...
% qalter -h n "*"
```

Job Workflows

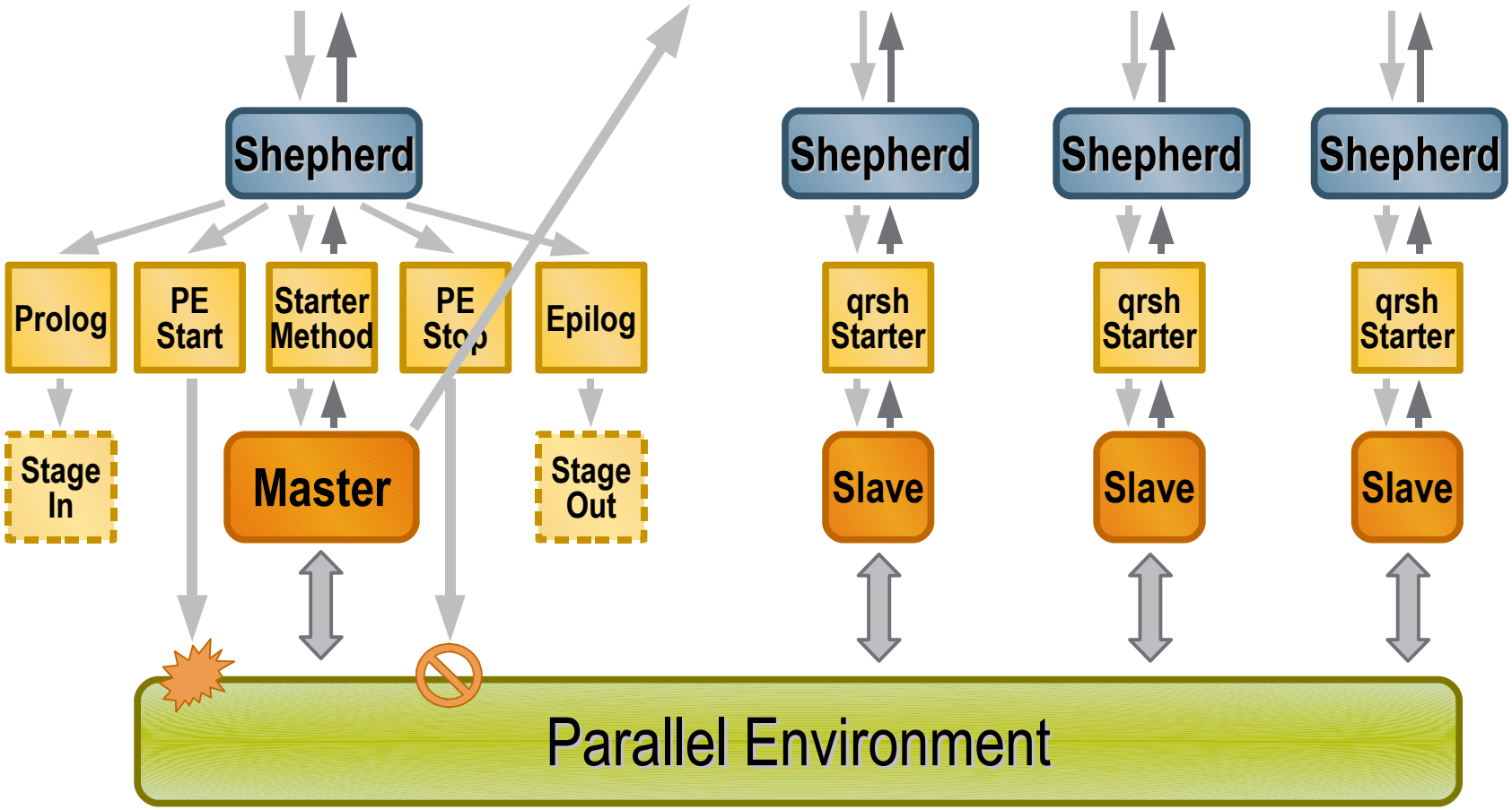
- Best answer is DRMAA
  - Allows for complex branching and intelligent decisions
  - C, Java, Perl, Python, Ruby
- Poor man's workflow
  - *qsub -hold_jid*
  - Job is held until listed jobs are no longer queued
  - Killed or failed are OK, too

Parallel Jobs

- Parallelized distributed applications
  - > Multiple collaborating jobs acting as one
  - > Shared memory
  - > Distributed memory
  - > External to N1 Grid Engine
- Parallel environments
  - > MPI, PVM, OpenMP, etc.
  - > Associated with queues
  - > Loose or tight integration
- qsub -pe *paralell_env min-max*



Starting a Parallel Job



Loose Integration

- qmaster generates list of nodes for slave tasks
  - Blocks off slots for slave tasks
- Master task starts slave tasks
  - Not via rsh or ssh
  - Usually an agent
- qmaster only has accounting for master task
- Needs custom terminate method
  - Otherwise slave tasks don't get killed

Tight Integration

- qmaster generates list of nodes for slave tasks
  - Blocks off slots for slave tasks
  - Notifies slave tasks hosts
- Master task starts slave tasks
  - Via *qrsh -inherit*
  - Or via rsh or ssh
    - Translated into *qrsh -inherit*
- qrsh -inherit bypasses scheduler
  - Runs *rsh|ssh host qrsh_starter job*
- Job runs as child of a shepherd
  - Full accounting and deletion support

Exercise: Alien Autopsy

- Open and examine
  - $SGE_ROOT/pvm/README
  - $SGE_ROOT/pvm/startpvm.sh
  - $SGE_ROOT/pvm/stoppvm.sh
  - $SGE_ROOT/pvm/pvm.sh
- Compare to
  - $SGE_ROOT/mpi/README
  - $SGE_ROOT/mpi/startmpi.sh
  - $SGE_ROOT/mpi/stopmpi.sh
  - $SGE_ROOT/mpi/rsh
  - $SGE_ROOT/mpi/mpi.sh

Solution: Alien Autopsy

- PVM integration is loose
  - startpvm.sh starts a pvmd on each slave node
  - pvm.sh calls spmd to start tasks under slave daemons
    - Internal protocol
  - stoppvm.sh stops the slave node daemons
- MPI integration is tight
  - startmpi.sh just creates the machine file
  - mpi.sh call mpirun to start tasks on slave nodes
    - Uses rsh wrapper script
  - stopmpi.sh removes the machine file
- How would you make the PVM integration tight?

Solution Solution: Alien Autopsy

- To make the PVM integration tight
  - startpvm.sh should start daemons with qrsh -inherit
- http://gridengine.sunsource.net/howto/pvm-integration/pvm-integration.html

Exercise: Parallel Universe

- Create a new PE integration for permi
- permi/permireg name hostfile
  - Starts master registry where all slaves will connect
  - Takes name of job as a param
    - Used by slaves to find the right master
  - Takes hostfile in Grid Engine format
  - Runs until you kill it
- permi/permirun name hostfile jarfile master
  - Starts slaves remotely via simple ssh
    - ssh hostname command args
  - Takes hostfile in Grid Engine format

Solution: Parallel Universe

- PE Starter:

```
#!/bin/sh

cp $2 $TMPDIR/hostfile
$SGE_ROOT/permi/permireg $1 $2
  >$TMPDIR/results 2>$TMPDIR/error &
ps -jp $! | tail -1 | awk '{print $2}' >
  $TMPDIR/pid

ln -s $SGE_ROOT/permi/pu_ssh $TMPDIR/ssh

# Give the server time to start
sleep 3
```

Solution: Parallel Universe

- ssh Wrapper:

```
#!/bin/sh

. $SGE_ROOT/$SGE_CELL/common/settings.sh

qrsh -inherit $*
```

Solution: Parallel Universe

- PE Shutdown:

```
#!/bin/sh

pid=`cat $TMPDIR/pid`; kill -9 -$pid
rm $TMPDIR/hostfile
rm $TMPDIR/pid
rm $TMPDIR/ssh

echo; echo RESULTS
echo ------------------------------------
cat $TMPDIR/results; rm $TMPDIR/results
echo ERRORS
echo ------------------------------------
cat $TMPDIR/error; rm $TMPDIR/error
```

Job Life-cycle Event Hooks

- Run command on event
  - Prolog – set up job environment
  - Start PE – set up parallel environment
  - Start job – start job in new shell
  - Suspend job – send suspend signal
  - Resume job – send resume signal
  - Checkpoint job – send signal or run command
  - Terminate job – send terminate signal
  - Stop PE – shut down parallel environment
  - Epilog – clean up job environment

Resource Quotas

- New with 6.1
- Before
  - Limit resource usage globally
- Now
  - Limit resource usage
    - By user, user group, queue, host, host group, project, PE
      - Individually
      - As a group
    - Or globally

Resource Quota Configuration

- Based on firewall configuration
  - Multiple rule sets
    - With multiple rules
- Take first matching rule from each rule set
- Take strictest rule set
- Rules can contain
  - Wildcard – *
  - Logical not – !
  - “Quoting” – {}
    - Treat as “per member” instead of as a group

Resource Quota Example I

- The total number of running jobs from power users should not total more than 40

```
{
   name         power_limit
   description  Limit all power users
   enabled      TRUE
   limit        users @power to slots=40
}
```

Resource Quota Example II

- No power user should have more than 10 running jobs

```
{
   name         power_limit
   description  Limit all power users
   enabled      TRUE
   limit        users {@power} to slots=10
}
```

Resource Quota Example III

- The total number of running jobs from power users should not total more than 40, and everyone else is limited to 5 running jobs each

```
{
   name         power_limit
   description  Limit all power users
   enabled      TRUE
   limit        users @power to slots=40
   limit        users {*} to slots=5
}
```

Resource Quota Example IV

- The total number of jobs without projects must be less than 10

```
{
   name         power_limit
   description  Limit all power users
   enabled      TRUE
   limit        projects !* to slots=10
}
```

Exercise: Taking Liberties

- Create the following rule:

```
{
    name            taking_liberties
    description     Fail to limit licenses
    enabled         TRUE
    limit           users * to slots=10
    limit           users * to license1=4
    limit           users * to license2=2
}
```

- Set up the license1=10 and license2=10 resources
- Submit 10 jobs that need both licenses
  - What happens? Why? Fix it.

Solution: Taking Liberties

- The first rule always matches, so the others are never evaluated
- To fix it, use a compound limit:

```
{
    name            taking_liberties
    description     Limit licenses
    enabled         TRUE
    limit           users * to \
                    slots=10,license1=4,license2=2
}
```

- Could also fix with three different rule sets

Exercise: Playing By the Rules

- Configure the following business rules:
  - There should never be more than 100 active jobs in the system
  - No user should have more than 10 active jobs, except for users working on project Blackbox, who are allowed to have 20 running jobs each, but no more than 60 active jobs total
  - There are 10 software licenses available, but no single user may user more than 2 at a time, except for users in the Development department, who are not limited in their license usage

Solution: Playing By the Rules

- Set max_jobs to 100 for global conf
- Set complex_values to license=10
- Requires three rule sets
- Set #1
  - > limit users {*} projects Blackbox to slots=20
  - > limit users {*} to slots=10
- Set #2
  - > limit projects Blackbox to slots=60
- Set #3
  - > limit users {!@Development} to license=2

Resource Quota Planning

- Stack rules top to bottom
  - Subset of the rule below it
  - Completely disjoint with the rule below it
- Spread rules horizontally
  - Whenever subsetting rule doesn't apply
  - Especially when filter targets are identical
- Combine rules when filters are identical
- Be care full with “per element” versus “total”
- Start with each rule in a separate rule set
  - Combine until it works

The qmaster

- sge_qmaster
- The mouth and ears, not the brain
- Manages the grid “database”
  - A giant state machine
- GDI – Grid Database Interface
  - Versioned protocol used to talk to qmaster
  - Synchronous or asynchronous
- Multi-threaded since 6.0
  - Hundreds of thousands of concurrent jobs
    - Officially claim 5

Exercise: Meet the Bootstrap

- cat $SGE_ROOT/$SGE_CELL/common/bootstrap
- What's in there?

Solution: Meet the Bootstrap

- admin_user – EUID for the daemons
- default_domain – Used for hostname resolution
- ignore_fqdn – Used for hostname resolution
- spooling_method – How to spool
- spooling_lib – Name of the spooling library
- spooling_params – Data for the spooling library
- binary_path – Where to find Grid Engine binaries
- qmaster_spool_dir – Where to spool
- security_mode – Security setting

qmaster Spooling

- Everything gets spooled
  - Failure recovery
- Three spooling methods
  - Berkeley Database
    - Local filesystem
      - Not NFSv3-friendly
      - Fast
    - Remote server
      - Single point of failure
      - Not as fast as local
  - Classic
    - Flatfile
    - Slow but simple

Spooling Locations

- “By default”
  - $SGE_ROOT/$SGE_CELL/spool/qmaster
    - Set in $SGE_ROOT/$SGE_CELL/common/bootstrap
    - Miscellaneous data files
    - Classic: All objects and state
  - $SGE_ROOT/$SGE_CELL/spool/spooldb
    - Set in $SGE_ROOT/$SGE_CELL/common/bootstrap
    - BDB: All objects and state
  - $SGE_ROOT/$SGE_CELL/spool/<hostname>
    - execd_spool_dir host config parameter
    - Temporary job storage
    - Miscellaneous data files

Security In Grid Engine

- Default install is not secure
  - Fundamental distributed application issues
- CSP install
  - Certificates to verify identify
  - All communications are sent over SSL
- Interactive jobs
  - Insecure even in CSP mode
  - Replace rsh/rlogin/telnet/rshd/telnetd with ssh/sshd
    - rsh_command, rlogin_command, qlogin_command: ssh
    - rsh_daemon, rlogin_daemon, qlogin_daemon: sshd -i
    - Loose control and accounting

Exercise: Secure Line

- Configure grid to use ssh for qrsh login
- qrsh
- ptree $$
- qrsh `pwd`/sl.sh
- Wait for job to end
- Configure grid to use ssh for qrsh command
- qrsh `pwd`/sl.sh
- Wait for job to end
- Compare the accounting records

Solution: Secure Line

- sl.sh spawns an escaped process
  - Creates a process tree in a new process group
  - Kills the head of the process tree
  - Leaves remaining processes unattached
- Grid Engine's rshd knows how Grid Engine tracks job processes
  - Used to find escaped processes
- sshd doesn't
  - Escaped processes get away
- Tightly integrated sshd available in open source

What Is a GID Range???

- Common installation question
- Every jobs gets an additional job id
  - Attached to job and all child processes
  - Used to track wayward job processes
- ENABLE_ADDGRP_KILL execd_params
  - Uses the additional group id for killing jobs
- gid_range is the range for the additional GIDs
  - Host config – applies per host
  - Can't have more jobs than additional GIDs

Exercise: the Great Escape

- qsub `pwd`/sl.sh
- Check if the *work* process is still running
- Add ENABLE_ADDGRP_KILL=TRUE to execd_params
- qsub `pwd`/sl.sh
- Check if the *work* process is still running
- Compare the accounting records

Solution: the Great Escape

- Without ENABLE_ADDGRP_KILL=TRUE, the worker process doesn't get killed
- With ENABLE_ADDGRP_KILL=TRUE, the worker process is killed when the job ends
- Regardless of the setting, CPU time is 0
  - Job ended before *work*

High Availability With Grid Engine

- Shadow daemon
  - Multiple can be active
  - Needs access to qmaster spool dir – heartbeat
- NFS server is single point of failure
- If NFSv3, shadow daemon needs BDB server
  - Single point of failure
  - In addition to NFS server
- Sun Cluster
  - NFS server
  - qmaster

Exercise: Who Shot the Sheriff?

- Run two shadow daemons
  - export $SGE_CHECK_INTERVAL=10
  - export $SGE_GET_ACTIVE_INTERVAL=30
  - export $SGE_DELAY_TIME=60
  - sge_shadowd
- Have to be on different hosts
- Create the .../common/shadow_masters file
  - master
shadow1
shadow2
- Kill -9 the qmaster

Solution: Who Shot the Sheriff?

- After SGE_CHECK_INTERVAL + SGE_GET_ACTIVE_INTERVAL a new master is started
- Second shadow daemon waits another SGE_DELAY_TIME seconds and then gives up

Communication With Execds

- Execds report load status periodically
  - load_report_time – defaults to 40 seconds
  - Balance between load and information
    - Reporting too often can overload the qmaster
    - Reporting too seldom can cause bad scheduling decisions
- Execd goes into *unknown* state after max_unheard
  - Defaults to 5 minutes
  - Jobs on unknown execd remain in last known state
    - Rescheduled after reschedule_unknown
    - If crash, rescheduled on restart
    - Dependent on rerunnable, checkpointing, PE

Managing Users

- Use OS facilities for auth & auth
- User object is for policy
- Annoying to keep both in sync
- Clever trick
  - enforce_user
    - TRUE – a user object is required to submit jobs
    - AUTO – a user object is automatically created
  - auto_user_oticket, auto_user_fshare, auto_user_default_project, auto_user_delete_time
    - Set default user object field values

The Scheduler

- Single-threaded
  - Will become a thread in the qmaster
- Periodically requests job data and load reports
- Scheduler algorithm is pluggable
  - Compile time
- Generates orders for qmaster

Three Configurations

- Choice of configurations at install time:

	MAX	HIGH	NORMAL
job_load_adjustments	NONE	NONE	np_load_avg=0.50
load_adjustments_decay_time	0:0:0	0:0:0	0:7:30
schedd_job_info	FALSE	FALSE	TRUE
schedule_interval	0:2:0	0:0:15	0:0:15
flush_submit_second	4	0	0
flush_finish_second	4	0	0
report_pjob_tickets	FALSE	TRUE	TRUE

Load Adjustments

- Load is usually measured by np_load_avg
  - Grows slowly
- Some jobs ramp up slowly
- Artificial load to prevent overloading
  - job_load_adjustments added for every job
  - Drops to 0 over load_adjustments_decay_time
- Seldom useful
  - If you use Normal scheduler config, set to NONE

Scheduler Runs

- Triggered by job events from qmaster
- Job events sent every schedule_interval
- If flush_submit_sec is non-zero
  - > flush_submit_sec seconds after job submit
  - > Delay to prevent excessive communication
- If flush_finish_sec is non-zero
  - > flush_finish_sec seconds after job end
  - > Delay to prevent excessive communication

Conservation of Information

- sched_job_info
  - If FALSE, scheduler messages are not reported
  - Only turn off in extreme situations
- report_pjob_tickets
  - If FALSE, jobs in qmon/qstat sorted by submit order
  - Turn off if performance matters
- Reduce network traffic & qmaster load

Scheduling Basis

- queue_sort_order
  - > load
    - – Soft requests, load_formula, seq_no
  - > seqno
    - – Soft requests, seq_no, load_formula
- load_formula
  - > $c_0 + c_1v_1 + c_2v_2 \ldots + c_nv_n$
    - – c – constant
    - – v – complex variable
  - > Default: np_load_avg

Exercise: Fill 'Em Up

- Create two queues called *primary* and *secondary*
- Set up the grid such that
  - Jobs aren't submitted to *secondary* unless *primary* is full
  - Hosts are chosen normally
    - Host fill order isn't important

Solution: Fill 'Em Up

- qconf -aq primary
- qconf -aq secondary
- qconf -rattr queue seqno 1 secondary
- Leave queue_sort_order as load and load_formula as np_load_avg
  - Schedule first by load → np_load_avg
    - Selects host
  - Queues on same host have same load
    - Schedule second by seq_no

Scheduler Policies

- Intended to be a “steering wheel”
- Three classes:
  - Entitlement Policy (aka “ticket policies”)
    - Share Tree Policy
    - Functional Ticket Policy
    - Override Tickets
  - Urgency Policy
    - Deadline time
    - Wait time
    - Resources
  - Custom Policy (aka “priority”)

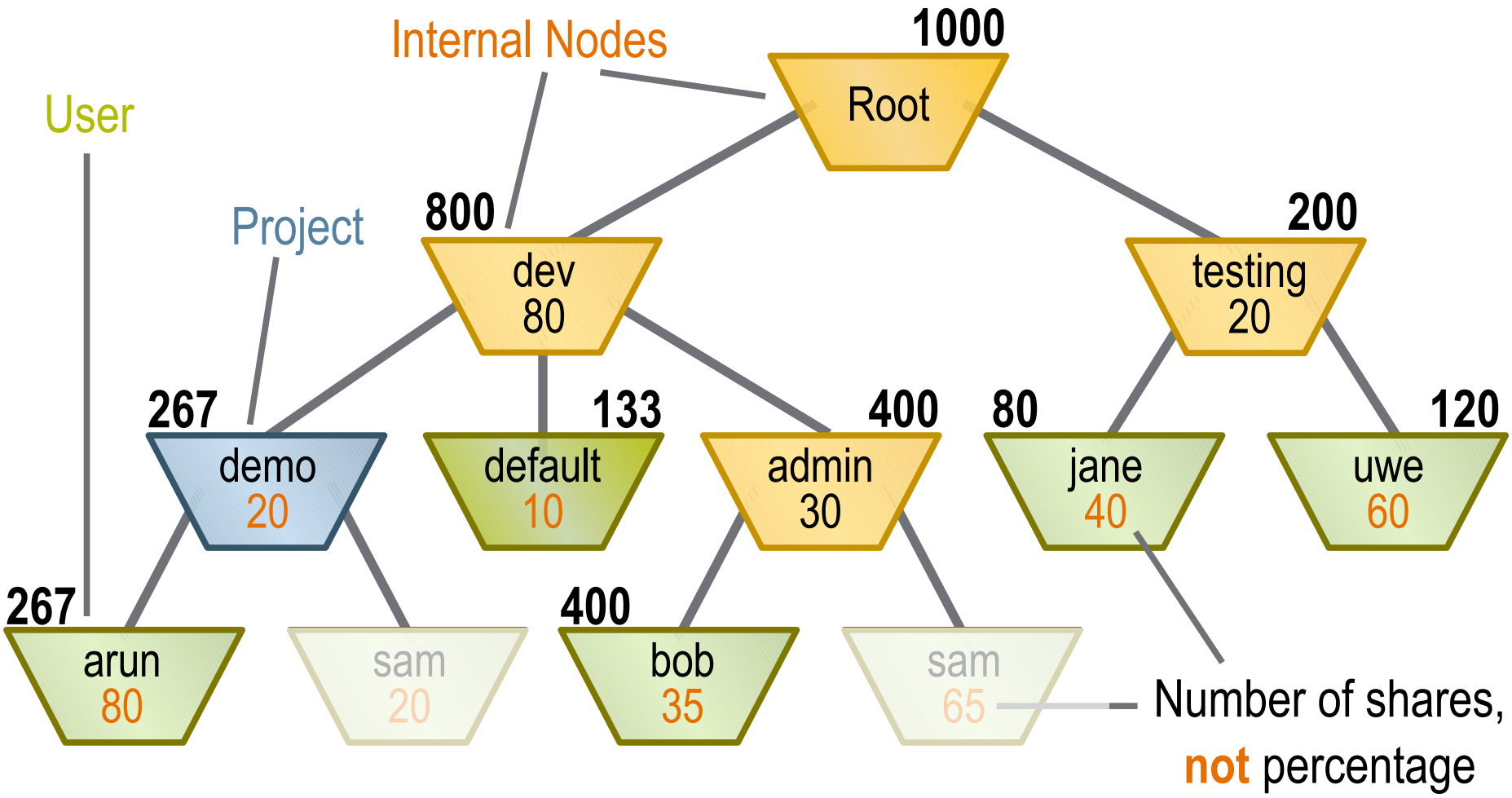
Share Tree Policy

- Start with *n* tickets
- Divide the tickets according to a policy tree
  - Each node divides its tickets among its children
  - Only accounts for active users
- Jobs sorted according to ticket count
- Has a memory
  - User who gets more now gets less later

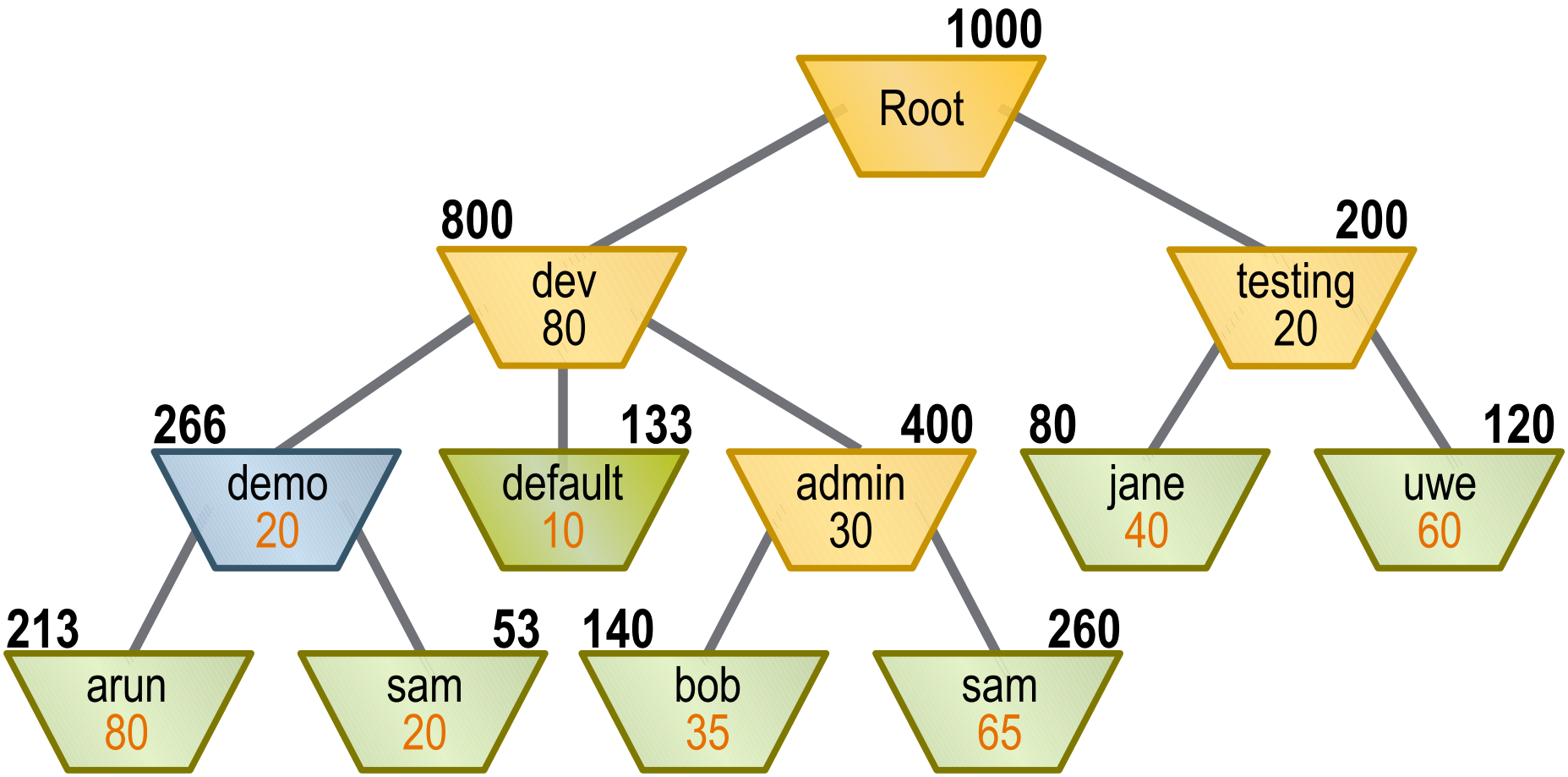
Share Tree Rules

- Users must be leaf nodes
- Leaf nodes must be project nodes or user nodes
- Project nodes cannot have project sub-nodes
- Non-leaf nodes can be:
  - User nodes
  - Project nodes
  - Arbitrary aggregation nodes
- Each user can appear only once in a project sub-tree or outside of all project sub-trees
  - Applies to special user, default

Share Tree Example



Share Tree Example



Exercise: Share And Share Alike I

- Implement the share tree policy from the example
  - Add the required users
    - Linux (useradd) and Grid Engine (qconf -auser)
- Disable/delete all queues except all.q
- Submit one non-project job as each user
- Submit one project job as arun and sam
- Check the tickets
- Delete sam's jobs
- Check the tickets

Solution: Share And Share Alike I

- The share tree is one of the few things for which I user qmon
- qstat -u * -f -ext
- After submitting the first round jobs, the ticket count should match the second example
  - arun's non-project job goes to the default node
- After deleting sam's jobs, the ticket count should match the first example

Exercise: Share And Share Alike II

- Submit a second project job as arun
- Check the tickets
- Submit a second non-project job as arun
- Check the tickets
- Submit a non-project job as another user
  - You might need to add a new user
- Check the tickets

Solution: Share And Share Alike II

- arun's second project job divides his share of the project shares
- arun's second non-project job divides his share of the default node
- The non-project job from the other user also goes to the default node
  - Gets a full share
- Shares are divided among a user's jobs
  - Default node shares are not divided among users

Exercise: Share And Share Alike III

- Submit a project job as jane
- Check the tickets
- Delete all the jobs in the system
- Submit a project array job for arun with enough tasks to fill all available job slots
- Submit a project job for arun
- Submit a non-project job for jane and uwe
- Check the tickets

Solution: Share And Share Alike III

- jane's project job gets no tickets
  - No default node under project
- qdel -u * *
- arun is using much more than his share
  - Should normally get more tickets than jane and uwe combines
  - Overage penalty reduces his tickets

Share Tree Tuning

- usage_weight_list
  - Determines what “resources” means
  - cpu=1.0,mem=0.0,io=0.0
- halftime
  - Time in hours to decay influence by half
  - 0: never
- halflife_decay_list
  - Same, but usage-specific
  - -1: immediate
  - cpu=168:mem=0;io=-1

More Share Tree Tuning

- compensation_factor
  - Multiplier for compensation limit
  - 2 means no compensation greater than 2x
- weight_tickets_share
  - Number of share tree tickets to share
    - Relevant only in comparison to other policies
  - Defaults to 0, i.e. no share tree tickets
- Don't try to predict ticket counts
  - Relative, not absolute
  - Steering wheel

Functional Ticket Policy

- Start with *n* tickets
- Divide the tickets into four categories
  - Users, departments, projects, jobs
- Divide tickets in each category among jobs in each category
- Sum ticket count from each category for each job
- Jobs sorted according to ticket count
- No memory

Additional Details

- Tickets from missing categories are shared with present categories
- By default, all categories weighted equally
- Job ticket shares calculated from category fshares
  - fshares is relative

Functional Ticket Example

- Assume 1000 tickets available

Job #1
job share=50
user dant: fshares=100
dept eng: fshares=86

(1000 / 4) * (50 / 50) = 250
(1000 / 4) * (100 / 250) = 100
(1000 / 4) * (86 / 86) / 2 = 125
475

Job #2
project blackbox: fshares=20
user andy: fshares=150
dept eng: fshares=86

(1000 / 4) * (20 / 20) = 250
(1000 / 4) * (150 / 250) = 150
(1000 / 4) * (86 / 86) / 2 = 125
525

Functional Ticket Example

- Assume 1000 tickets available

Job #1
job share=50
user dant: fshares=100
dept eng: fshares=86

(1000 / 4) * (50 / 100) / 2 = 125
(1000 / 4) * (100 / 250) / 2 = 50
(1000 / 4) * (86 / 86) / 3 = 83

258 x 2

Job #2
project blackbox: fshares=20
user andy: fshares=150
dept eng: fshares=86

(1000 / 4) * (20 / 20) = 250
(1000 / 4) * (150 / 250) = 150
(1000 / 4) * (86 / 86) / 3 = 83

483

Exercise: Visualization

- 1000 total tickets, 25% per category
- Project 1: fshares=100
- Project 2: fshares=80
- Department: fshares=75
  - Arun: fshares=80
    - 2 jobs in Project 1
  - Uwe: fshare=40
    - 1 job in Project 1, 2 jobs in Project 2, 1 job in no project
  - Jane: fshares=70
    - 1 job in no project
- What will the functional tickets be?

Solution: Visualization

- No job shares, so other three get 33% each
- Arun = 179
  - (1000 / 3) * ((80 / 190 / 2) + (75 / 75 / 7) + (100 / 180 / 3))
- Jane = 170
  - (1000 / 3) * ((70 / 190) + (75 / 75 / 7))
- Uwe = p1=126, p2=139, np=65
  - (1000 / 3) * ((40 / 190 / 4) + (75 / 75 / 7) + (100 / 180 / 3))
  - (1000 / 3) * ((40 / 190 / 4) + (75 / 75 / 7) + (80 / 180 / 2))
  - (1000 / 3) * ((40 / 190 / 4) + (75 / 75 / 7))

Exercise: Actualization

- 1000 total tickets, 25% per category
- Project 1: fshares=100
- Project 2: fshares=80
- Department: fshares=75
  - Arun: fshares=80
    - 2 jobs in Project 1
  - Uwe: fshare=40
    - 1 job in Project 1, 2 jobs in Project 2, 1 job in no project
  - Jane: fshares=70
    - 1 job in no project
- Check the tickets

Solution: Actualization

- You should see the same ticket totals that you calculated in the previous exercise
- Arun = 179
- Jane = 170
- Uwe = p1=126, p2=139, np=65

Function Ticket Tuning

- weight_tickets_functional
  - Total number of tickets to be divided, default to 0
- weight_user, weight_project, weight_department, weight_job
  - Category shares
  - Must sum to 1.0
- max_functional_jobs_to_schedule
  - Ticket calculations take time
    - The more jobs, the more time
  - Caps the number of jobs considered per scheduler run
  - Default is 200

More Function Ticket Tuning

- share_functional_shares
  - > TRUE
    - Default
    - Job count dilutes ticket share
    - share / sum of shares in category / job count
  - > FALSE
    - Job count doesn't affect tickets
    - Every job gets its the category's full share
    - Priority users can hog the grid
    - share / sum of share of jobs in category

Exercise: Revisualization

- Set share_functional_shares=FALSE
- 1000 total tickets, 25% per category
- Project 1: fshares=100, Project 2: fshares=80
- Department: fshares=75
  - Arun: fshares=80
    - 2 jobs in Project 1
  - Uwe: fshare=40
    - 1 job in Project 1, 2 jobs in Project 2, 1 job in no project
  - Jane: fshares=70
    - 1 job in no project
- What will the functional tickets be?

Solution: Revisualization

- No job shares, so other three get 33% each
- Arun = 188
  - (1000 / 3) * ((80 / 390) + (75 / 75 / 7) + (100 / 460))
- Jane = 107
  - (1000 / 3) * ((70 / 390) + (75 / 75 / 7))
- Uwe = p1=123, p2=109, np=51
  - (1000 / 3) * ((40 / 390) + (75 / 75 / 7) + (100 / 460))
  - (1000 / 3) * ((40 / 390) + (75 / 75 / 7) + (80 / 460))
  - (1000 / 3) * ((40 / 390) + (75 / 75 / 7))

Override Ticket Policy

- Used to make temporary changes
- Assign extra tickets
  - User, project, department or job
  - Arbitrary ticket number
- share_override_tickets
  - Whether job count dilutes override tickets
  - Defaults to TRUE

Running Versus Pending

- Tickets for running jobs
  - Follow the scheme we've talked about so far
  - Used for reprioritization
- Tickets for pending jobs
  - Have an extra wrinkle
    - Job submission order
  - Have a couple of extra wrinkles...
  - Used for scheduling

Reprioritization

- Update nice values to match ticket policy
- reprioritize
  - Master switch
  - Default is FALSE
- reprioritize_interval
  - How often to update nice values
  - 0:0:0 is off
  - Default is 0:0:0

The Wrinkles

- Job submission order is important
  - If two jobs are equal, they must run FIFO
- Each job's tickets divided by number of jobs before
  - Per category for shared functional tickets
- Obscenely complicated
  - Don't try to predict ticket amounts
  - Remember it's a steering wheel!

Exercise: Pending Investigation

- Disable your queues
- 1000 total tickets, 25% per category
- Project 1: fshares=100, Project 2: fshares=80
- Department: fshares=75
  - Arun: fshares=80
    - 2 jobs in Project 1
  - Uwe: fshare=40
    - 1 job in Project 1, 2 jobs in Project 2, 1 job in no project
  - Jane: fshares=70
    - 1 job in no project
- Check the tickets, enable the queue, check again

Solution: Pending Investigation

- With the queue disabled, you see the pending ticket counts
- Notice that they're very different from what you saw before
- Notice the geometric tendency
- After you enable the queue, you should see the same running tickets count that you saw previously

The Silver Lining

- Three different ticket policies
  - Order is important for pending tickets
  - Each policy gets the order from the previous
    - First policy gets job submission order
- policy_hierarchy
  - Controls policy order
  - OFS by default
  - OS – sort first by override tickets, then by share tree
    - Ignore function ticket policy
  - O ignores order
    - Usually goes first

Exercise: Connecting the Dots I

- Disable your queues
- Set weight_tickets_share to 10000
- Set weight_tickets_functional to 40000
- Create (reuse) a project with fshare of 100
- Set your user fshare to 0
- Create a share tree with your user as only node
  - shares of 100
- Submit 2 non-project jobs, then 4 project jobs
- Check the tickets

Solution: Connecting the Dots I

- Nothing unusual
  - The two non-project jobs got the fewest tickets
  - No override
  - Functional gives tickets in submission order
  - Share tree gives tickets to project jobs in functional order
- Notice the pretty geometric series in the tickets
- On to part two...

Exercise: Connecting the Dots II

- Set the policy_hierarchy to OSF
- Check the tickets
- Set 1 override ticket for the last project job
- Check the tickets
- Set 50000 override tickets for the first non-project job
- Check the tickets

Solution: Connecting the Dots II

- The order changed!
  - No override
  - Share tree gives tickets in job submission order
  - Functional gives projects jobs tickets in share tree order
- The order changed again!
  - Override puts last project job first
  - Share tree gives tickets in job override order
  - Functional gives projects jobs tickets in share tree order
- The order changed again!
  - The first non-project job's override tickets overwhelm the other policies

Urgency Policies

- Deadline time = $w_{deadline} / (t_{deadline} - t)$
  - Increases as deadline approaches
  - Only users in *deadlineusers* can submit deadline jobs
  - Default weight_deadline = 3600000.0
    - When $t_{deadline} = t$, deadline = $w_{deadline}$
- Wait time = $w_{wait} * (t - t_{submit})$
  - Increases the longer a job waits
  - Guarantees that jobs will eventually run
  - Default weight_waiting_time = 0.0
- Urgency = deadline + wait_time + resource

Exercise: Time Is On Your Side I

- Disable/delete all queues but one queue instance
- Set that queue instance's slots to 1
- Submit 2 jobs
- Check the urgency
- Set the wait time weight to 10
- Check the urgency again
- Check the urgency again

Solution: Time Is On Your Side I

- qmod -d “*”
- qmod -e all.q@host
- qstat -urg
- Notice that the wtcontr keeps going up

Exercise: Time Is On Your Side II

- Add yourself to the *deadlineusers* user list
- Submit a deadline job for ~2 minutes from now
- Check the urgency
- Check again every few seconds
  - Until the deadline passes

Solution: Time Is On Your Side II

- qconf -au user deadlineusers
- qsub -dl 06051300 ...
- Notice that the dlcontr keeps going up
  - Exponentially
- Once the deadline has been crossed, dlcontr is 3600000

Custom (Priority) Policy

- POSIX priority: -1023 to 1024
  - Bigger is higher
  - Non-privileged users can only set negative
- Set at submit time
  - qalter'ed while pending
- Override setting for admin
- Could be used to implement custom prioritization

Putting It All Together

- Final job priority
  - > $w_{ticket} * p_{ticket} + w_{urgency} * p_{urgency} + w_{priority} * p_{priority}$
  - > Default
    - weight_ticket: 0.01
    - weight_urgency: 0.1
    - weight_priority: 1.0
- Generally acceptable settings
  - > Can turn off all but priority somewhere else
  - > Priority should stay largest
  - > Might want to swap tickets and urgency

Exercises: the Plan Comes Together

- Disable your queues
- Set weight_tickets_share to 10000
- Set weight_tickets_functional to 40000
- Set your user fshare to 0
- Create (reuse) a project with fshare of 100
- Create a share tree with your user: shares = 100
- Submit 2 non-project, 2 project, 2 deadline jobs, and 2 jobs with -p 1000
- Check the priority, adjust weights, check again, ...

Solution: the Plan Comes Together

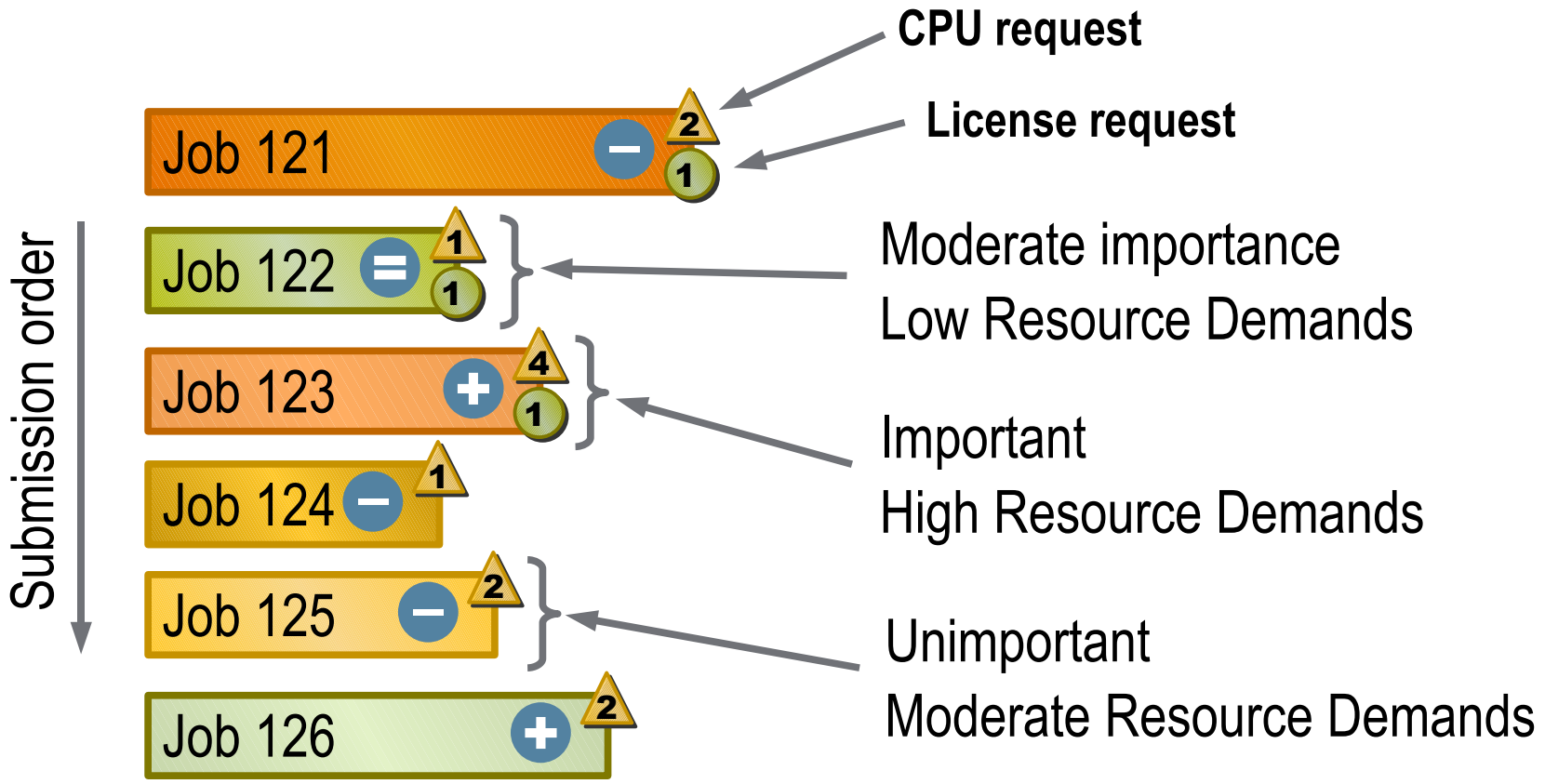
- qstat -ext -urg -pri
  - Make sure your terminal is **really** wide
- Notice that by qaltering the POSIX priority, you can override all the other policies
  - Unless you've changed the weight_priority...
- Notice that you can control job priority by changing the weighting factors

Resource Reservation

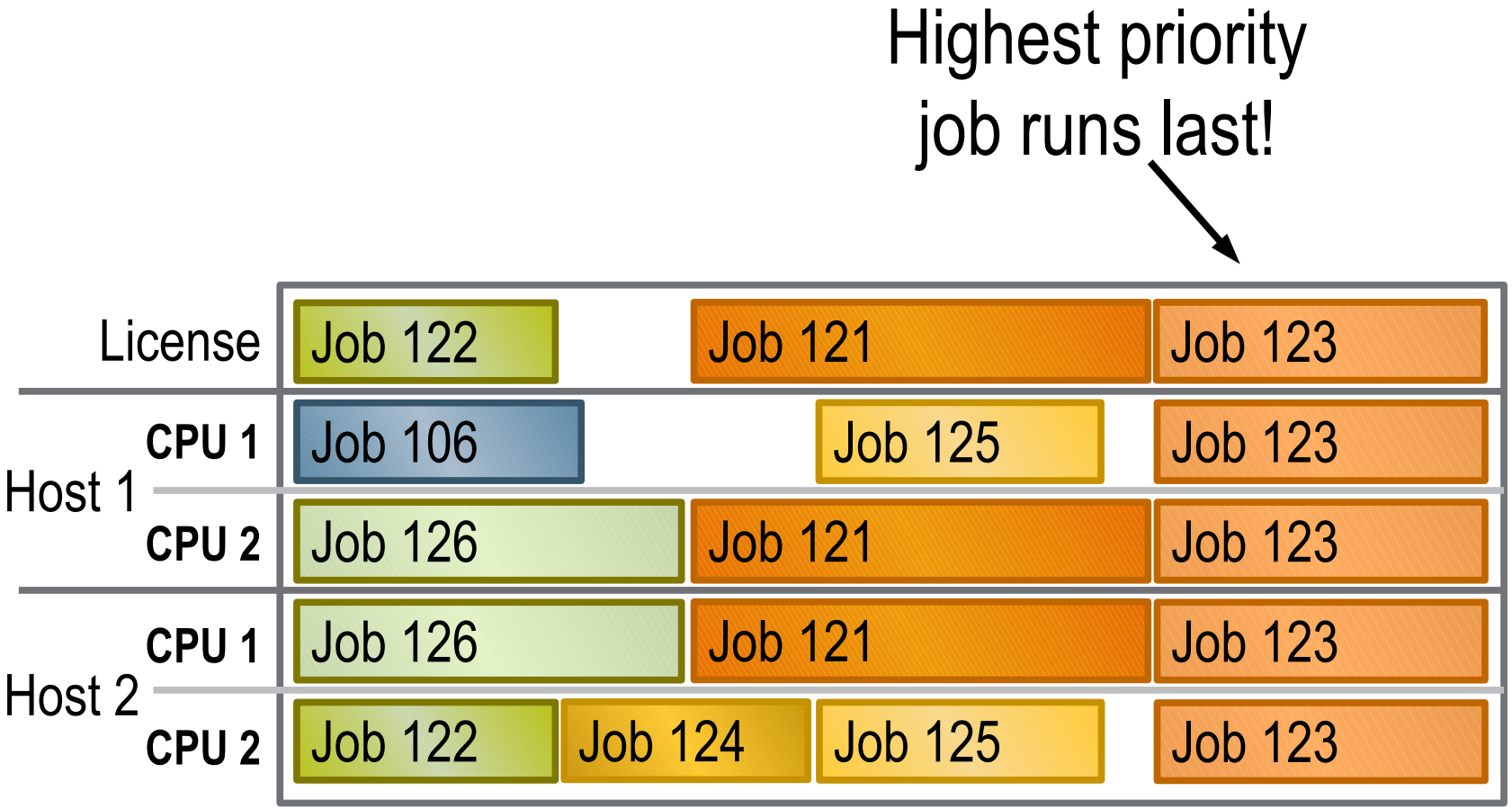
- Not advance reservation – Coming soon!
- “Big” jobs can get starved out by smaller jobs
  - Priority inversion
  - Wait time urgency is one solution
- Resource Reservation
  - Allows a job to gather resources
  - Runs when all the resources are available
- Backfilling
  - Makes sure remaining resources are used
  - Fills gaps with smaller jobs

Resource Reservation Example

Pending Jobs List:

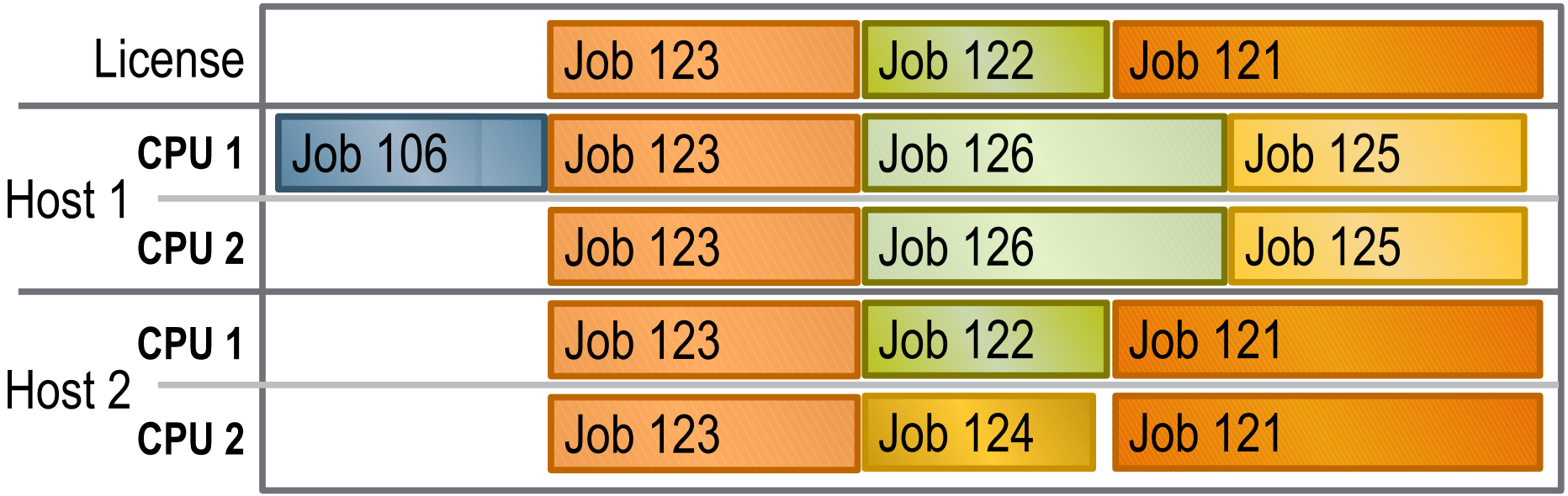


Without Resource Reservation



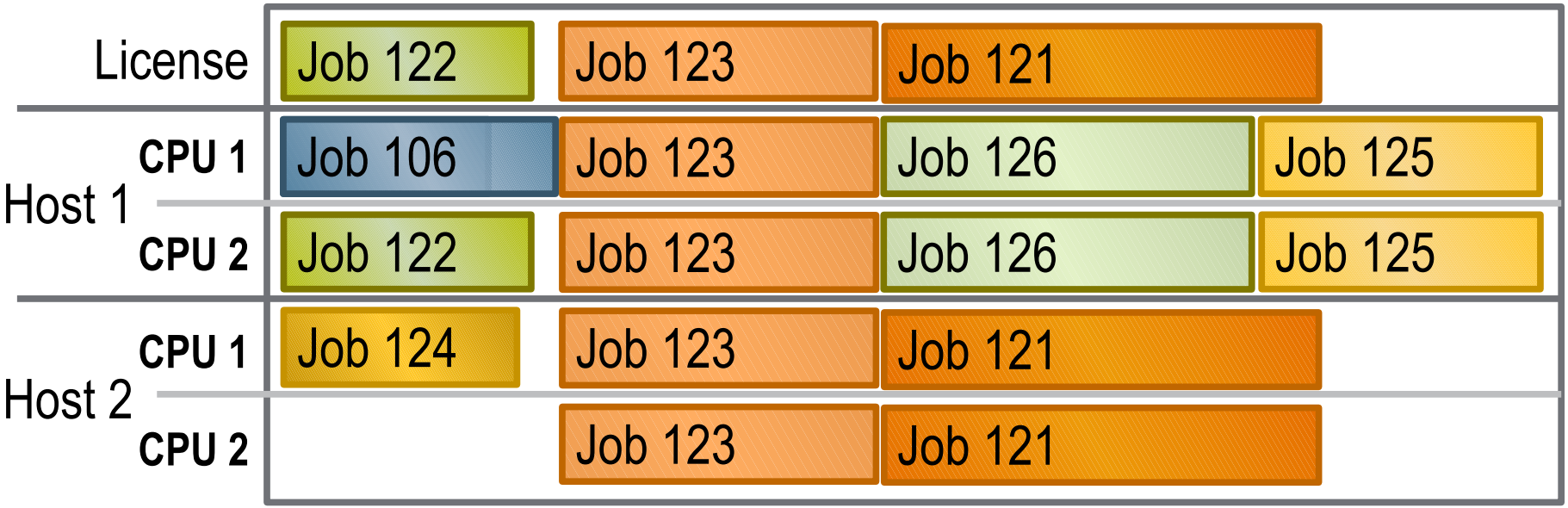
With Resource Reservation

Right job order,
but less efficient!



Resource Reservation + Backfilling

Best trade-off between job order and efficiency



Exercise: Dinner For Seventeen I

- Disable all queue instances but one
  - Set the slots to 4
- Make sure weight_waiting_time is 0
- Set max_reservation to 10
- Run dfs.sh four times
- Run a high (1000) priority PE job (4 slots)
- What happens?
- Run a higher (1024) priority PE job (4 slots) with reservation
- Now what happens?

Solution: Dinner For Seventeen I

- qsub -p 1000 -pe make 4 .../sleeper.sh 10
- The PE job, never runs, even though it's always at the top of the pending job list
- qsub -p 1024 -pe make 4 -R y .../sleeper.sh 10
- This PE job gathers resources until it has enough to run
  - May have to run two – looks like a bug...
- Why the higher priority?
  - Only the highest priority job can gather resoures

Exercise: Dinner For Seventeen II

- Let the PE job(s) end
  - Or delete it
- Check the start times of the DFS jobs
  - Use qmod -s/-us to space them out a little if needed
- Run dfs_short.sh a few times
- Check the priorities
- Run a high-priority PE job with reservation (4 slots)
- What happens?

Solution: Dinner For Seventeen II

- qmod -sj jodid
- qmod -usj jobid
- dfs_short.sh; dfs_short.sh; dfs_short.sh
- qstat -pri
- The DFS_Short jobs are low priority
- qsub -p 1024 -pe make 4 -R y .../sleeper.sh 10
- As the PE job gathers resource, it will fill in the space with the short jobs where possible

Exercise: Dinner For Seventeen III

- Kill the PE job and the DFS_Short jobs
- Set the default_duration to 0:0:5
- Run a few low-priority sleeper jobs
- Run a high-priority PE job with reservation (4 slots)
- What happens?

Solution: Dinner For Seventeen III

- qdel DFS_Short Sleeper
- qsub -p -100 -N Liar .../sleeper.sh 30
- qsub -p 1024 -pe make 4 -R y .../sleeper.sh 10
- The default duration says the sleeper jobs will only run 5 seconds
- The sleeper jobs really run 60 seconds
- After they're back-filled in, they prevent the PE job from running until they finish

Backfilling

- Depends on knowing the length of the jobs
  - Jobs can specify length
    - Hard or soft run time resource
  - default_duration
    - A suggestion, not a limit
- Reservation is held up by jobs that run over
  - Set default duration above the average run time
  - Submit with run time to be backfill eligible
- Interesting research being done with virtual machines...

Grid Engine Log Files

- $SGE_CELL/spool/master/messages
  - Qmaster
- $SGE_CELL/spool/master/schedd/messages
  - Scheduler
- $SGE_CELL/spool/<host>/messages
  - Execution daemon
- Check them periodically for errors
- Control information level
  - loglevel – log_error, log_warning, log_info

Scheduler Messages

- qconf -tsm
  - Dumps data into $SGE_ROOT/$SGE_CELL/common/schedd_runlog
- Similar to qstat -j scheduler messages
  - Useful if schedd_job_info is turned off

Accounting Data

- Two accounting files
  - $SGE_ROOT/$SGE_CELL/common/accounting
    - Predefined accounting info
    - On by default
    - Made available through qacct
  - $SGE_ROOT/$SGE_CELL/common/reporting
    - Configurable accounting info
    - Much more data than the accounting file
    - Off by default
    - Used by ARCo

Reporting Parameters

- reporting_parameters
  - accounting
    - Enable/disable writing of accounting file
  - reporting
    - Enable/disable writing of reporting file
  - flush_time
    - Configure interval between flushing of reporting file
  - accounting_flush_time
    - Configure interval between flushing of accounting file
    - 0:0:0 means that accounting data is not buffered
    - Defaults to flush_time

Troubleshooting With Debug Output

- Log files are nice, but...
  - Not terribly informative
- $SGE_ROOT/util/dl.[c]sh
  - Source the file
  - Run dl <n>
  - Start daemon or run command
- Helps to understand the source base...
  - Still useful even if you don't

Using Debug Levels

- dl.[c]sh script has predefined levels:
  1. Top = Info
  2. Top = Trace + Info
  3. Top + CULL + GDI = Info
  4. Top + CULL + GDI = Trace + Info
  5. Top + GUI + GDI = Info
- Higher number != more info
- Translated into $SGE_DEBUG_LEVEL
- Also sets $SGE_ND
  - > Causes daemons not to daemonize

Exercise: De Bugs, Boss!

- source $SGE_ROOT/util/dl.sh
- Set the debug level to 1
- echo $SGE_DEBUG_LEVEL
- Submit a job with some options
- Set the debug level to 4
- echo $SGE_DEBUG_LEVEL
- Do the submission again
  - Notice the difference?
- Set the debug level back to 0

Solution: De Bugs, Boss!

- Debug level 1 just shows general debug information
  - SGE_DEBUG_LEVEL = 2 0 0 0 0 0 0 0
    - 2 = Info
- Debug level 3 shows debug info and function tracing for general, CULL, and GDI layers
  - SGE_DEBUG_LEVEL = 3 3 0 0 0 0 3 0
    - 1 = Trace
- <Line #> <PID> <Thread ID> -->|<-- <Message>
  - --> = function enter
  - <-- = function exit

Exercise: De Bugs, Master!

- Stop the qmaster
- Set $SGE_DEBUG_LEVEL to 2 0 0 0 0 0 0 0
- Start the qmaster
  - What happens?
- Stop the qmaster
- Set $SGE_ND to true
- Start the qmaster
- CRTL-C the qmaster
- Browse through the output

Solution: De Bugs, Master!

- qconf -km
- export SGE_DEUBG_LEVEL=”2 0 0 0 0 0 0 0”
- sge_qmaster
- You only get output until it daemonizes
- qconf -km
- export SGE_ND=true
- sge_qmaster
- Now you get the full output
  - Notice how the qmaster spools its data before exiting

Debugging the Shepherd

- The shepherd is started by the execd
  - Can't turn (or see) debugging output
- KEEP_ACTIVE execd_params
  - Does not delete job from active jobs directory
- Browse through config files
- Turn on debugging and run shepherd by hand

Exercise: Night Of the Living Job

- Turn on KEEP_ACTIVE
- Submit a job
- cd to .../$SGE_CELL/spool/host/active_jobs/jobid
- Look at the trace, config and environment files
- Run the shepherd

Exercise: Night Of the Living Job

- environment and config are written by execd
- trace is written by the shepherd as it runs
- config contains the entire execution context
  - Shepherd reads it and applies it before forking job
- environment contains the env vars to set
- trace contains the shepherd's output
- Running the shepherd produces the same output as in trace
  - Useful when modifying config

Troubleshooting Communications

- qping host port qmaster|execd 1
  - Gives status message for qmaster or execd
  - Traces messaging packets
- Loops every second (-i to set interval)
  - Gives simple status by default
  - -f gives full status message
- -info gives full status message and exits
- -dump
  - Traces communications
  - Run as root from same machine

Exercise: Machine That Goes QPing

- Run qping against the qmaster
- Run qping -info against the qmaster
- Run qping -info against an execd
- Run qping -dump against the qmaster

Solution: Machine That Goes Qping

- Simple output just tells you the master is alive and since how long
- Full output provides details
  - Messages in buffers is most interesting
  - Info and Monitor can be interesting
  - Only qmaster provides monitoring
- Message tracing provides tons of info
- See the qping(1) man page for interpretation

Qmaster Monitoring

- Qmaster thread monitoring is disabled by default
- MONITOR_TIME qmaster_params
  - How often monitoring info is dumped
  - Default is 0:0:0 = not at all
- LOG_MONITOR_MESSAGE qmaster_params
  - TRUE
    - Monitoring data is available via qping and written to messages
    - Default
  - FALSE
    - Monitoring data is only available via qping

Troubleshooting With Qevent

- Not in the courtesy binaries or product
  - Must build from source
- Mostly intended for use with the test suite
- One interesting option
  - *-trigger event script*
    - Event is either JB_END or JB_TASK_END
    - Runs script each time event occurs
    - Script gets three arguments
      - Event
      - Job id
      - Task id

Q & A

Useful Resources

- http://gridengine.info
- http://gridengine.sunsource.net/howtos
- http://bioteam.net/dag/
- http://blogs.sun.com/templedf
- http://docs.sun.com/app/docs/coll/1017.4
- http://gridengine.sunsource.net/nonav/source/browse/~checkout~/gridengine/doc/htmlman/manuals.html?content-type=text/html
- Sign up to the mailing lists!

Wrap Up

- That's all, folks!
- I hope it was educational
- Please make sure you fill out (and hand in!) a survey
- Remember, grid is good

SUN GRID ENGINE ADVANCED ADMINISTRATION

Daniel Templeton
dan.templeton@sun.com

maintenance

- *log files*
  - *loglevel*
- *dl*
  - *SGE_DEBUG*
  - *SGE_ND*
- *qping*
  - Use to confirm load report time > max unheard
- *qevent*
- *qconf -tsm*
- *accounting file*
- *KEEP_ACTIVE*

ARCo

- Internals
- installation
- reporting file
  - configuring
- running queries
- creating queries

4. Template – Text Slide with Two Line Title and Subtitle

This is a Subtitle with Initial Caps Each Major Word

- To insert a Subtitle on other slides, copy and paste the Subtitle text block.
- On a slide with a two line Title and Subtitle (as shown here) move the bullet text block down to make room for the Subtitle. You can also duplicate this slide and replace its content.

9. Template – Quote

"Revolutionary solutions come from the meeting of many different minds."

Jane R. Doe
Vice President, Engineering
XYZ Corporation


Sun
microsystems


Sun
microsystems


Sun
microsystems


ULTRASPARC™


ULTRASPARC™


ULTRASPARC™


Java™


Java™


Java™


solaris™


solaris™


solaris™

opensolaris™

opensolaris™

opensolaris™

StorageTek™

StorageTek™

StorageTek™

Positive Reading

Reversed Out