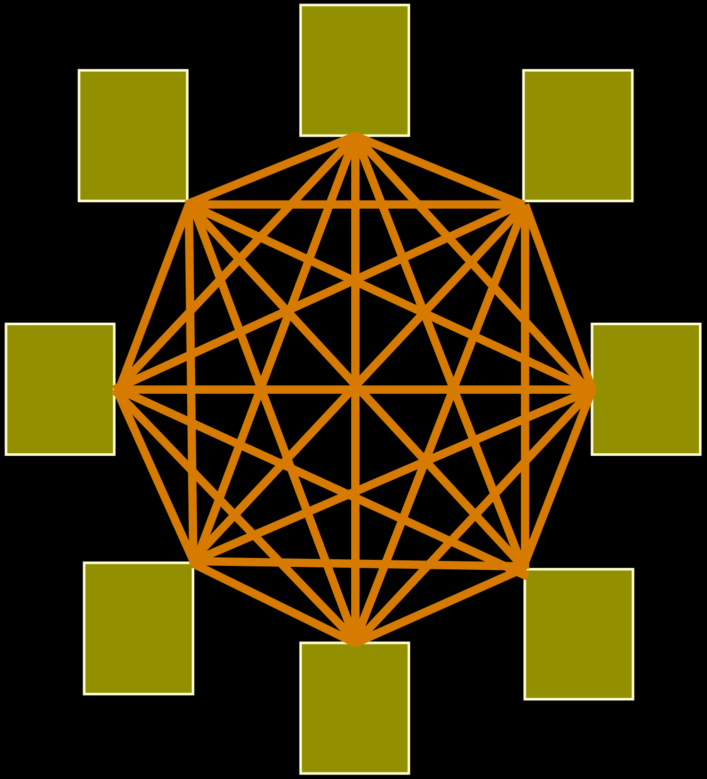
Grid Engine Administration

Overview

This module covers

- Grid Problem Types
- Distributed Resource Management
- Grid Engine 6 Variants
- How it works
- Grid Engine Scheduling
- Grid Engine 6 Architecture



Grid Problem Types

"Grid" Problem Types

Tightly Coupled	Embarrassingly Parallel
1 + 1 = X	1 + 1 = X
X + 2 = Y	2 + 2 = Y
Y + 3 = Z	3 + 3 = Z
N + Z = W	N + N = W

Traditional Parallelism

- ***Single process*** in which several parallel elements (threads) must communicate to solve ***a single problem***
- Parallel programming is complicated and far from automatic
- Users must explicitly use parallel programming tools and application environments
- Speedup on clusters ***not*** guaranteed
  - Depends on code, interconnect latency & the problem space

Traditional Parallelism

- Used to be a much more fractured space
  - PVM, LINDA, MPI, etc.
  - MPI rules present day (*some exceptions)
- Infinite possibilities for admin headaches
  - MPI flavor "A" using interconnect type "B" via integration method "C" using SGE CPU $allocation_rule="D"
  - *Ouch*.

Embarrassingly Parallel

- Also known as “Serial” or “Batch” Computing

- Large numbers of nearly identical jobs
  - Possibly only the input file changes

- Identical analysis on large pools of input data or queries

- Easily subdivided, mostly independent tasks

Embarrassingly Parallel

- Primary optimization focus
  - Often less focus on interconnect and latency issues
  - More focus on speed, size and scaling headroom of the storage infrastructure (or memory)
  - Performance tuning the grid scheduler (‘DRM’) and batch subsystem for high throughput
  - Data locality issues often important

Generic Use Case

Embarrassingly Parallel

1 + 1 = X

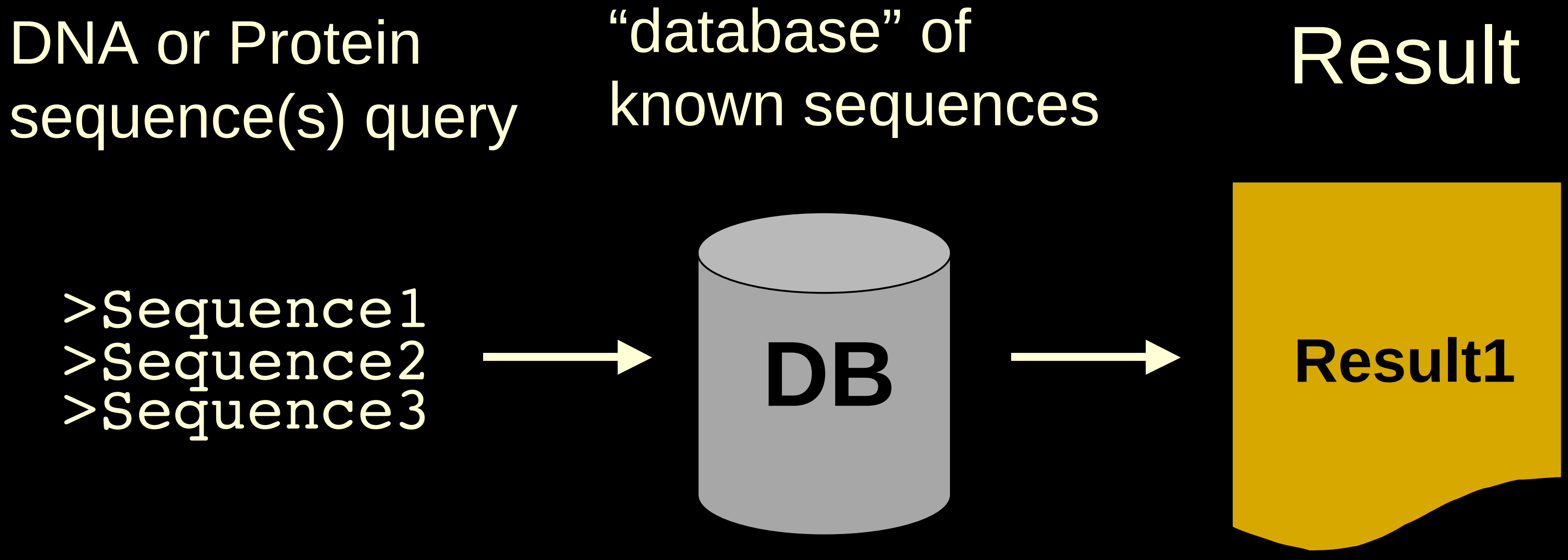
2 + 2 = Y

3 + 3 = Z

N + N = W

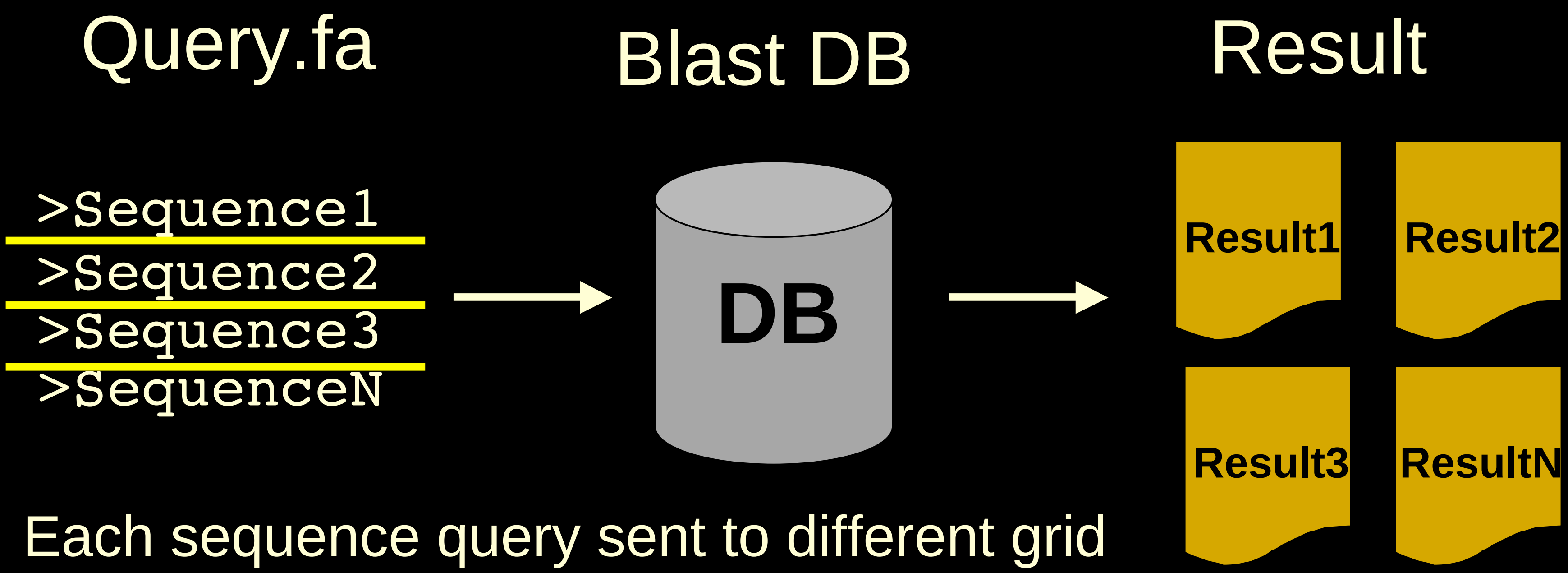
- Scientist has a problem
- She does not want or need to run ONE parallel application stretching across a 100 CPU cluster
- She wants to run a standalone program 100,000 or 1,000,000 times with slightly different input and output values

Cliche Life Science use case


DNA or Protein sequence(s) query
“database” of known sequences
Result
>Sequence1
>Sequence2
>Sequence3
DB
Result1

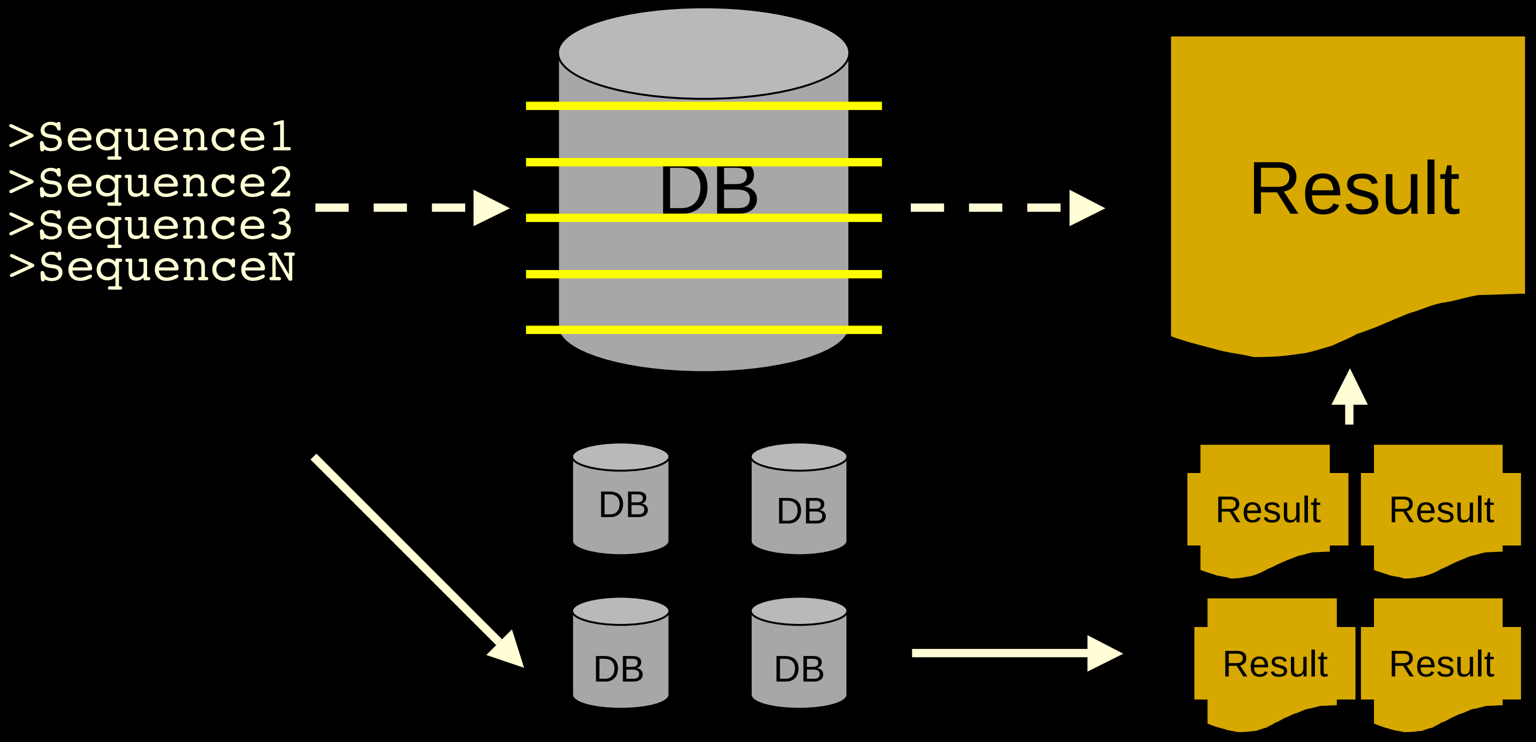
Comparing sequences of interest to known repositories in order to identify areas of *biologically significant* similarity

On a “grid” you can split large jobs up by query sequence



Each sequence query sent to different grid node for analysis. Each query is 100% independent of others.

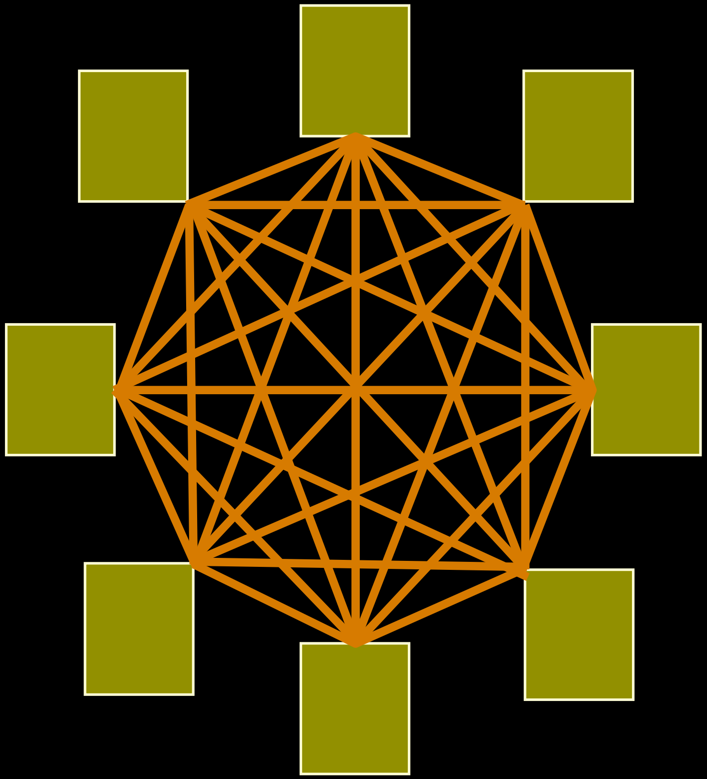
Can also split target databases & each query



MPIBLAST !

What's the point?

Supporting both job types …

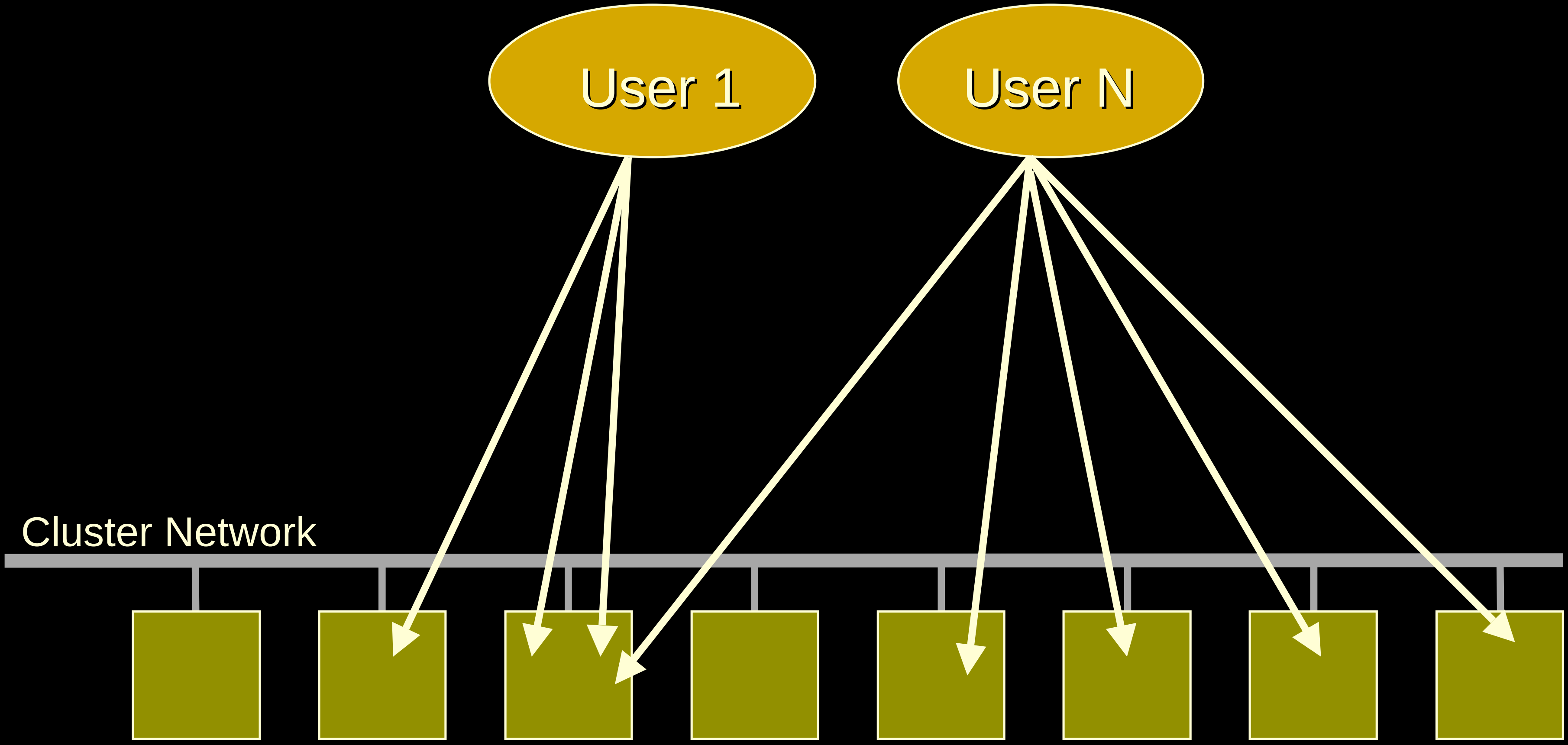


- Easy to support both types of workflows on a cluster or compute farm
- But …
  - Often require different optimization methods/focus
  - Policy based scheduling becomes a bit harder
  - Job starvation can be an issue

Node a01
00:E0:81:2A:B3:0D
Node a02
Node a03
00:E0:81:2A:F2:65
Node a04
00:E0:81:2A:F3:A5
Node a05
00:E0:81:2A:F3:17

Selling ‘DRM

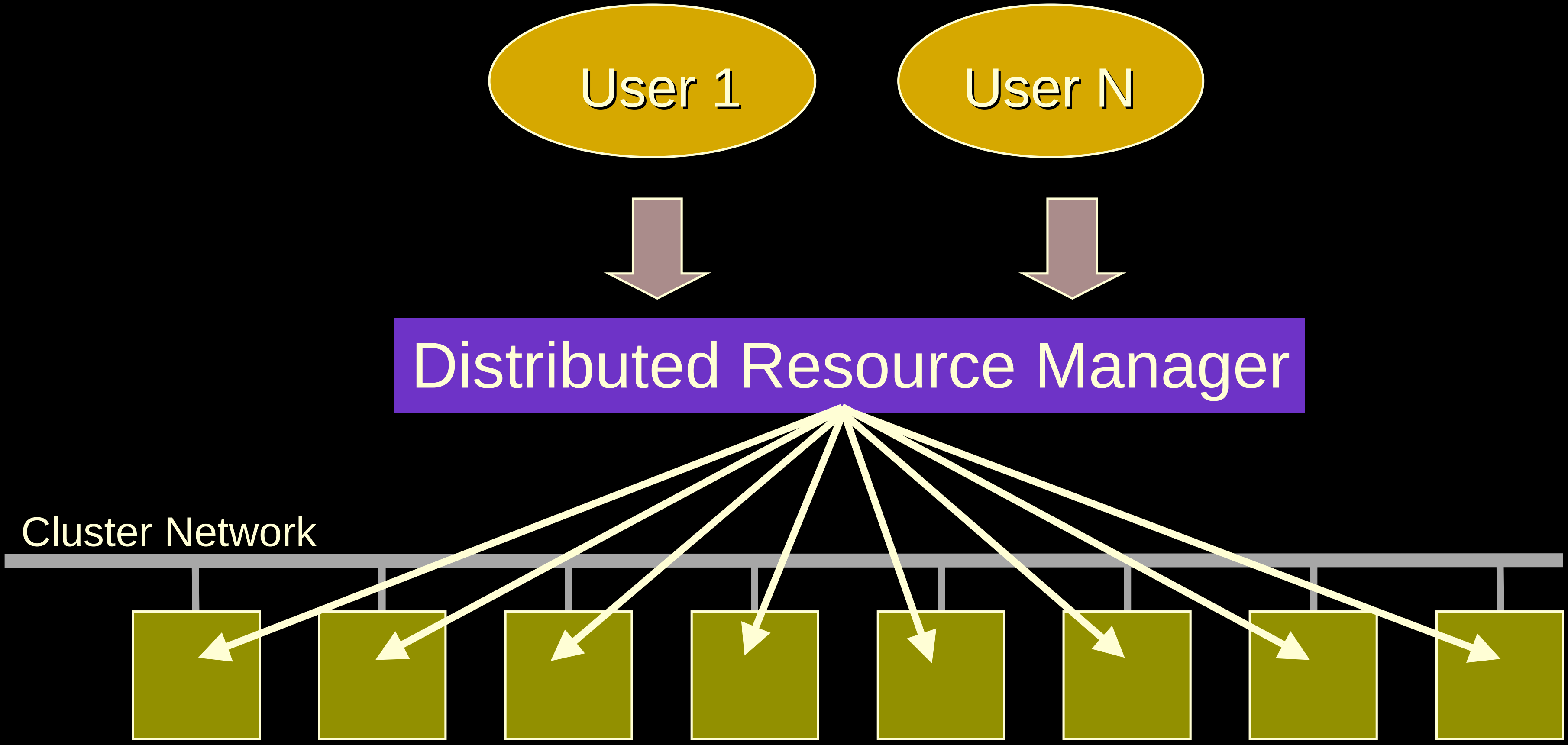
Old Way



Problems with “Old Way”

- Inefficient use of available resources
- Most aggressive get most resources
- Business priorities not reflected in resource allocation
- Brittle
- “Consensus” management fails as users grow

A better way


User 1
User N
Distributed Resource Manager
Cluster Network

Distributed Resource Management

- The Grid Engine ('SGE') suite can be used to dynamically manage resource allocation and remote job execution.
- Users request resources and SGE transparently handles the process of assigning the work to the most appropriate machine(s).

A good DRM provides…

- Policy based allocation of distributed resources
- Batch queuing & scheduling
- Load balancing & remote job execution
- Detailed job accounting statistics
- Fine-grained user specifiable resources
- Suspend/resume/migrate jobs
- Tools for reporting Job/Host/Cluster status
- Job Arrays
- Integration & tight control of parallel jobs

DRM from User View

- Can greatly simplify life & improve workflows
- Users generally interact with a single “master” node. No stress about individual machines or resources
- User defines minimum resource requirements (if necessary) and the DRM subsystem handles the task of finding the best resources to complete the task

DRM from Manager View

- Holy Grail: Allow scientific or business priorities to dynamically influence allocation of computing resources
  - Prevents inequity caused by aggressive users who may “unfairly” monopolize systems and resources
- Maximize infrastructure utilization via transparent load balancing
  - Server utilization percentages can dramatically increase
- Help maximize job throughput via smart placement decisions
  - *NOAA.gov example: place parallel job on *smallest* suitable host*
    - Keeps the largest hosts “free” for the largest jobs …
- Offload CPU cycles from “expensive” resources to inexpensive commodity gear

A walk through Grid Engine history …

If you have used Grid Engine 5.x …

- "Enterprise Edition" type behavior is now the default
  - Mostly visible in the resource allocation policy mechanisms
- New feature: "Cluster queues"
  - "`qsub -q all.q ./my-job-script.sh`"
- New feature: "Queue instances"
  - "`qsub -q all.q@computeNode04 ./my-job.sh`"
- New feature: "Hostgroups"
  - "`qsub -q all.q@@BigMemoryMachines ./my-job.sh`"
  - "`qsub -q all.q@@AppleXserves ./my-job.sh`"
- You can now directly submit binaries
  - "`qsub -b y ./myapp.exe`"

If you have used SGE 6.0 …

- Now in 6.1:

- Biggest change:
  - Resource Quota Framework
    - Massive improvement in fine grained resource allocation
    - “firewall-like” rules for user/job/queue/resource quotas and limits

  - Qstat default output changes
    - Previous:
      - qstat -f -u ‘*’
    - Defaults to showing only your jobs …
      - qstat -f -u <you>
      - Easy to revert though …

If you have used SGE 6.1 …

- Now in 6.2:
- Biggest changes:
  - Result of internal scalability engineering efforts (largely for TACC …)
    - Qmaster is now threaded, supports ~64,000 core cluster
    - No more sge_schedd, it's now a thread within the qmaster
  - schedd_job_info disabled by default!
    - Boo!
  - Advance Reservation (AR) becomes mainline
    - Request resources be available at guaranteed time
  - New implementation for interactive job support
    - Removes dependency on RSH/SSH in some cases …
      - But … not if you need X11 forwarding :)
  - Project "Hedeby" becomes Service Domain Management
  - Array Job Interdependency
    - Very significant feature contributed by third party firm
  - Docs move into http://wikis.sun.com

SGE 6.2 prior to 6.2u2 …

- Biggest changes:

- Efforts geared towards new features/functionality
  - GUI Installer
  - MS Windows Vista exechost support
  - Job Submission Verifiers (**)
    - *Just like RQS was this is new enough that few 'best practices' are known yet …*
    - *Watch for a DanT whitepaper on this significant new feature*
  - Per-job Consumable Resources
  - Linux version switches to jmalloc (claim 20% memory footprint reduction)

SGE 6.2u3 (current release)

- Very significant software license changes
  - Back to the SGE 5.x days (sorta)
    - All code is open source and hosted at http://gridengine.sunsource.net
    - Only the courtesy binaries from sunsource.net are free
    - Courtesy binaries do not include EC2 Adapter or ‘sgeinspect’ utility
    - Use of the Sun.com binaries beyond 90 days requires a license
- New: ‘sgeinspect’ GUI monitor tool for SGE & SDM
- SDM:
  - New: Single JVM/host required now
  - New: Amazon EC2 & Power saving SDM modules
- SGE
  - Exclusive host scheduling (!)
  - Windows Vista display support for MS GUI applications

How Grid Engine 6 Works

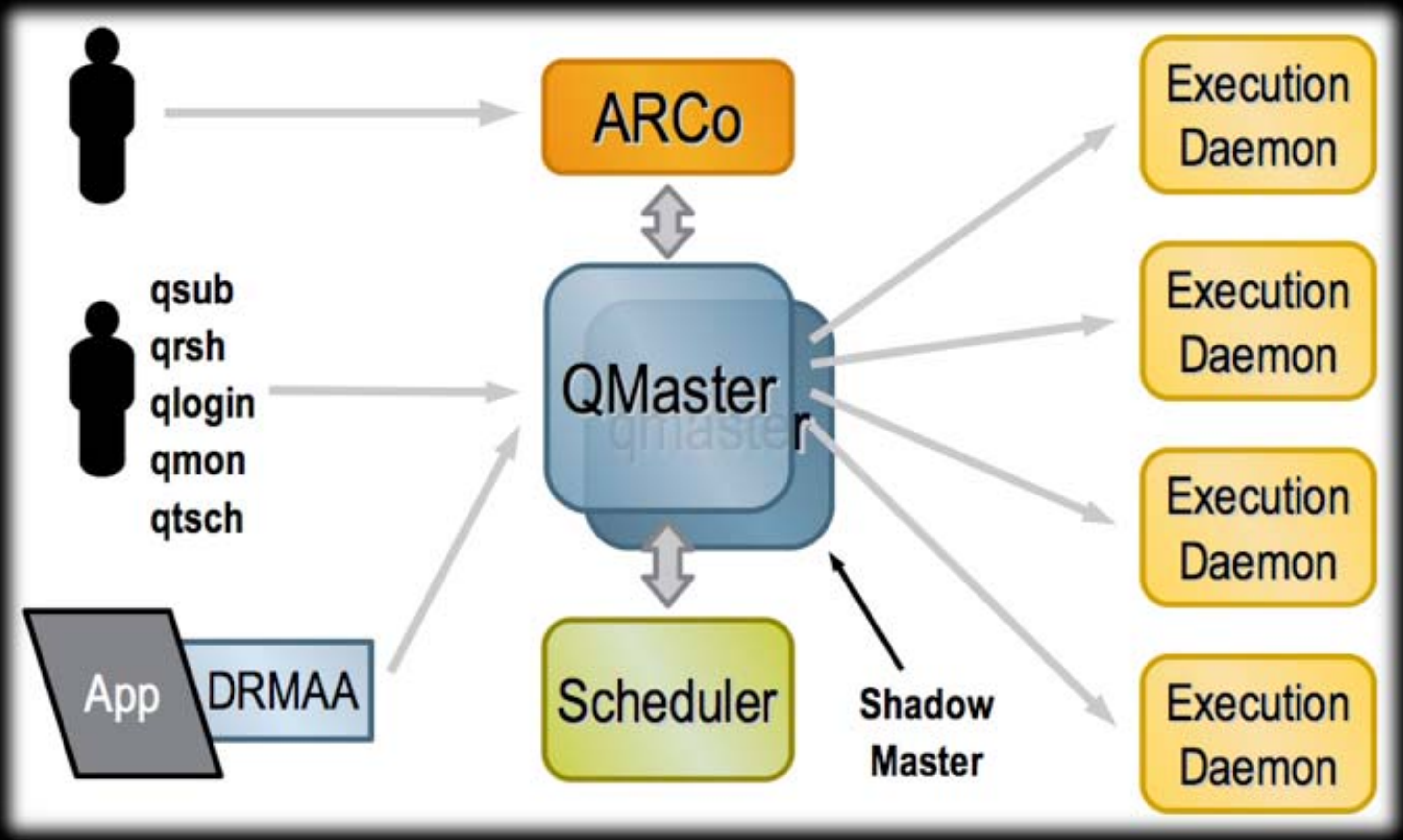
If you have used LSF/PBS/Torque …

- Features and functionality largely the same, but …
- Very significant conceptual changes
- Different approach to “queue”
- Different resource philosophy

For LSF/PBS/Torque Users …

- Grid Engine ‘queues’ are very different from ‘queues’ within LSF or PBS
  - A queue is not a pending area for jobs
  - Each cluster node will have one or more queues active on it
  - Users do not submit jobs to any particular queue.
  - Users describe the resources needed by the job and SGE works to find the best possible “queue instance”
  - Less common to see application-specific queues in SGE deployments

Key Terms & Concepts - I



Graphic credit: Dan Templeton

Key Grid Engine Terms

- Cluster
  - Collection of machines (‘hosts) where Grid Engine runs
- Master Host
  - Host that runs the ‘sge_qmaster’ process
  - Central to cluster operation, collects cluster state, status & load information
  - Controls policies, resource quotas and all scheduling decisions
  - All SGE clients communicate with this host
- sge_qmaster Daemon
  - Accepts incoming jobs from users.
  - Maintains tables about hosts, queues, jobs, system load, and user permissions.
  - Performs scheduling functions and requests actions from execution daemons on the appropriate execution hosts.
  - Decides which jobs are dispatched to which queues and how to reorder and reprioritize jobs to maintain share, priority, or deadline
- Shadow Master(s)
  - One or more systems that can take over for a failed master host

Key Grid Engine Terms

- Execution Host
  - A system that can run Grid Engine jobs
- sge_execd Daemon
  - Runs on all execution hosts in the cluster
  - Receives jobs from the master daemon and executes them locally
  - Responsible for the queue instances on its host and for the running of jobs within them
  - Periodically forwards information on host & job state/status/load to the master daemon.
- Scheduler
  - Makes all job dispatch and policy decisions
  - In SGE 6.1 and earlier versions a standalone daemon called ‘sge_schedd’
  - In SGE 6.2 and later the scheduler is just a thread within the sge_qmaster
- Submit Host
  - Any system with permission to submit jobs
- Admin Host
  - Any system with permission to run SGE administrative commands

Key Grid Engine Terms

- DRMAA
  - Software API for submitting and controlling jobs
  - “Distributed Resource Management Application API” (DRMAA)
  - http://www.drmaa.org
- ARCo
  - “Analysis & Reporting Console”
  - Web based accounting console
- SDM
  - “Service Domain Manager”
  - Module for implementing auto-scaling and service level agreements (“SLA”)
- RQS
  - “Resource Quota Syntax”
  - Powerful new resource allocation tool in SGE 6.1 and later
- JSV
  - “Job Submission Verifier”
  - Pre-test submitted jobs against custom conditions
  - Powerful new feature in SGE 6.2 and later

Huh?

Grid Engine does the following:

- Accept work requests (jobs) from users
- Puts jobs in a pending area
- Sends jobs from the pending area to the best available machine
- Manages the job while it runs
- Returns results, logs accounting data when the job is finished

Huh?

- What you need to know:
  - Don’t worry about queues or specific machines. All you need to do when submitting a job is describe the resources your job will need to run successfully.
  - Grid Engine will take care of the rest
  - The ‘default’ settings are good enough for most cases

Standard Job Types

- Batch
- Interactive
- Parallel
- Checkpoint

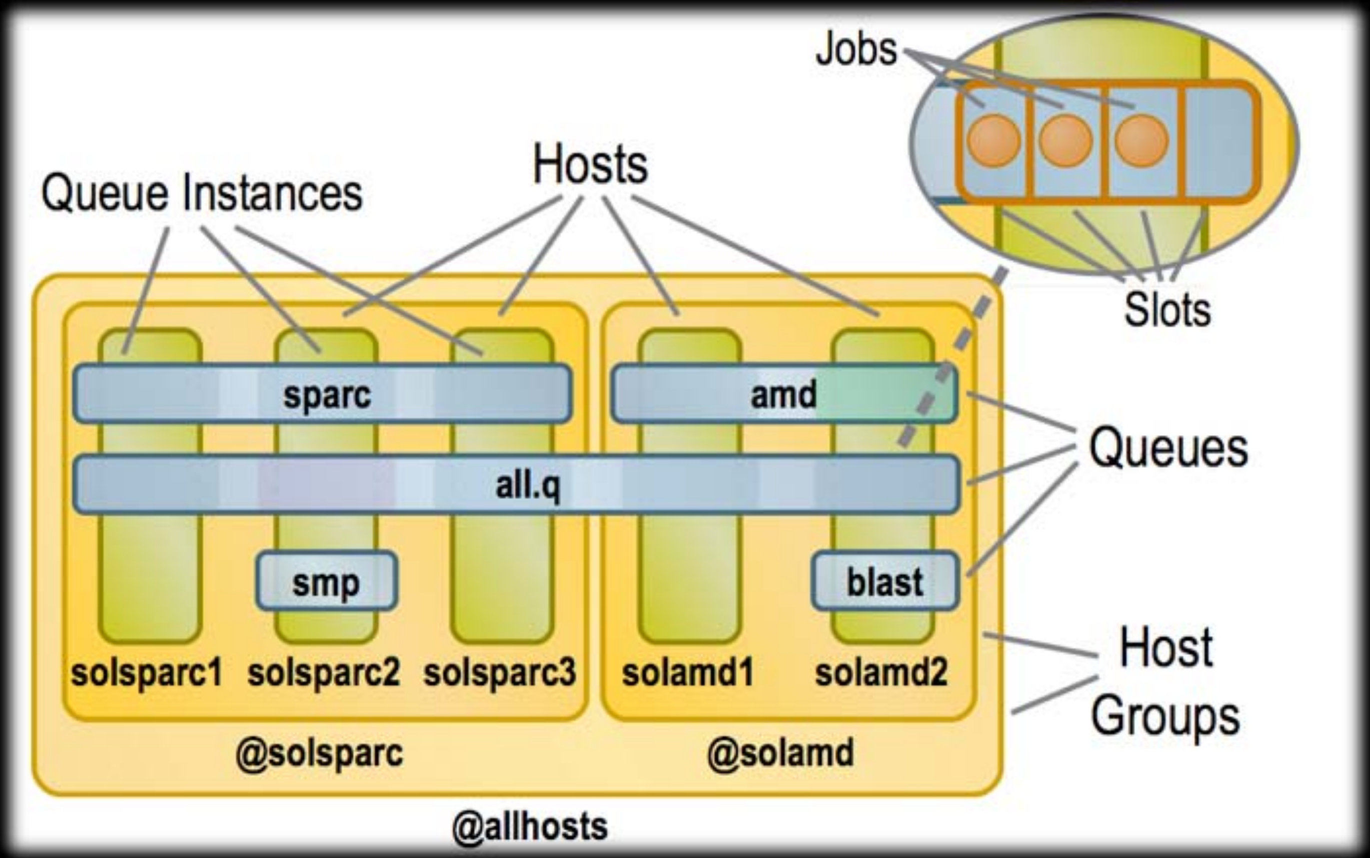
Queues

- A grid engine queue is:
  - "*A container for jobs that execute on a single host*"
- Configurable properties include
  - Job type
  - # Available CPUs
  - Resource usage thresholds / limits
  - Access control lists
    - User, Department, Group
  - Which custom environments are supported
    - Checkpointing
    - Parallel (PE
- Terminology
  - "cluster queue"
    - *all.q*
  - "queue instance"
    - *all.q@node004*

Slots

- Queues have “job slots”
- Slot count determines max number of concurrent running jobs in any queue
- Default queue config:
  - Slot count set equal to CPU count

Recap: Queues, slots & hosts

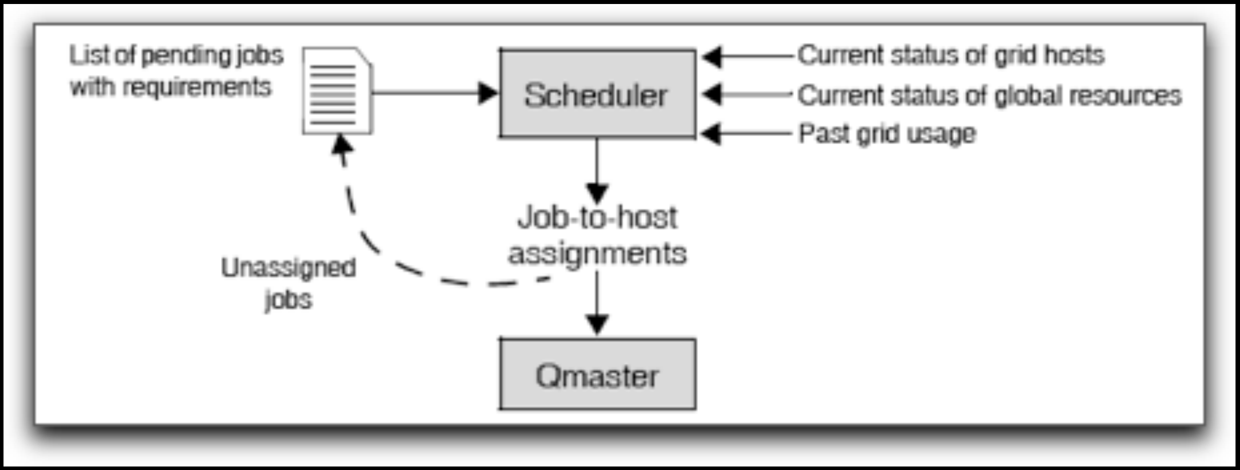


Graphic credit: Dan Templeton

Grid Engine Scheduling

The SGE scheduler is key …

- Decides which jobs run and *where* and *when* they get dispatched
- In most cases, an “endless loop” process referred to in the docs as a “scheduling run” or “scheduler interval”


List of pending jobs with requirements
Scheduler
Current status of grid hosts
Current status of global resources
Past grid usage
Job-to-host assignments
Unassigned jobs
Qmaster

Source: “Scheduler Policies for Job Prioritization in Sun N1GE”
http://www.sun.com/blueprints/1005/819-4325.html

In each scheduling run …

- Gather info about all jobs currently running
- Gather info from hosts via queue execd daemons
  - Load/utilization reports, slot info
- Recheck status of all global resources and consumables
- Remember some things about past usage

Source: "Scheduler Policies for Job Prioritization in Sun N1GE"
http://www.sun.com/blueprints/1005/819-4325.html

Then …

- Filter pending job list for those that CAN be started
  - Check hold status, dependency requirements, etc.
- Compute priority for all pending jobs
- Filter prioritized list for jobs that CAN be run
  - Availability of hard resource requests & hosts
- Dispatch into available job slots in order of priority
- Queue instance selection sort
  - Sort among available queue instances
  - Hard requests 1st, soft requests 2nd
  - If multiple queue instances are acceptable; subsort on load or seqno
- Dispatch to remote host
- Rinse, repeat …

Source: "Scheduler Policies for Job Prioritization in Sun N1GE"
http://www.sun.com/blueprints/1005/819-4325.html

How resources get allocated

Default scheduler behavior

- FIFO
  - “First in, First out”
  - Jobs executed in order received
  - Pending list also sorted by order received

Policy Based Resource Allocation

- Cluster admins can use all, some or none of the available high level usage policy implementations
- Any combination
- Any order
- Any relative weight

The 4 Key Policies in SGE 6

- Urgency
- Functional
- Share Tree
- Override

Urgency Policy

- Jobs sorted based on a calculated urgency value
- Urgency value dynamically derived from:
  - Job resource requirements
    - “Expensive” assets can be utilized first …
  - Approach of job initiation deadline
  - How long a job has been ‘pending’

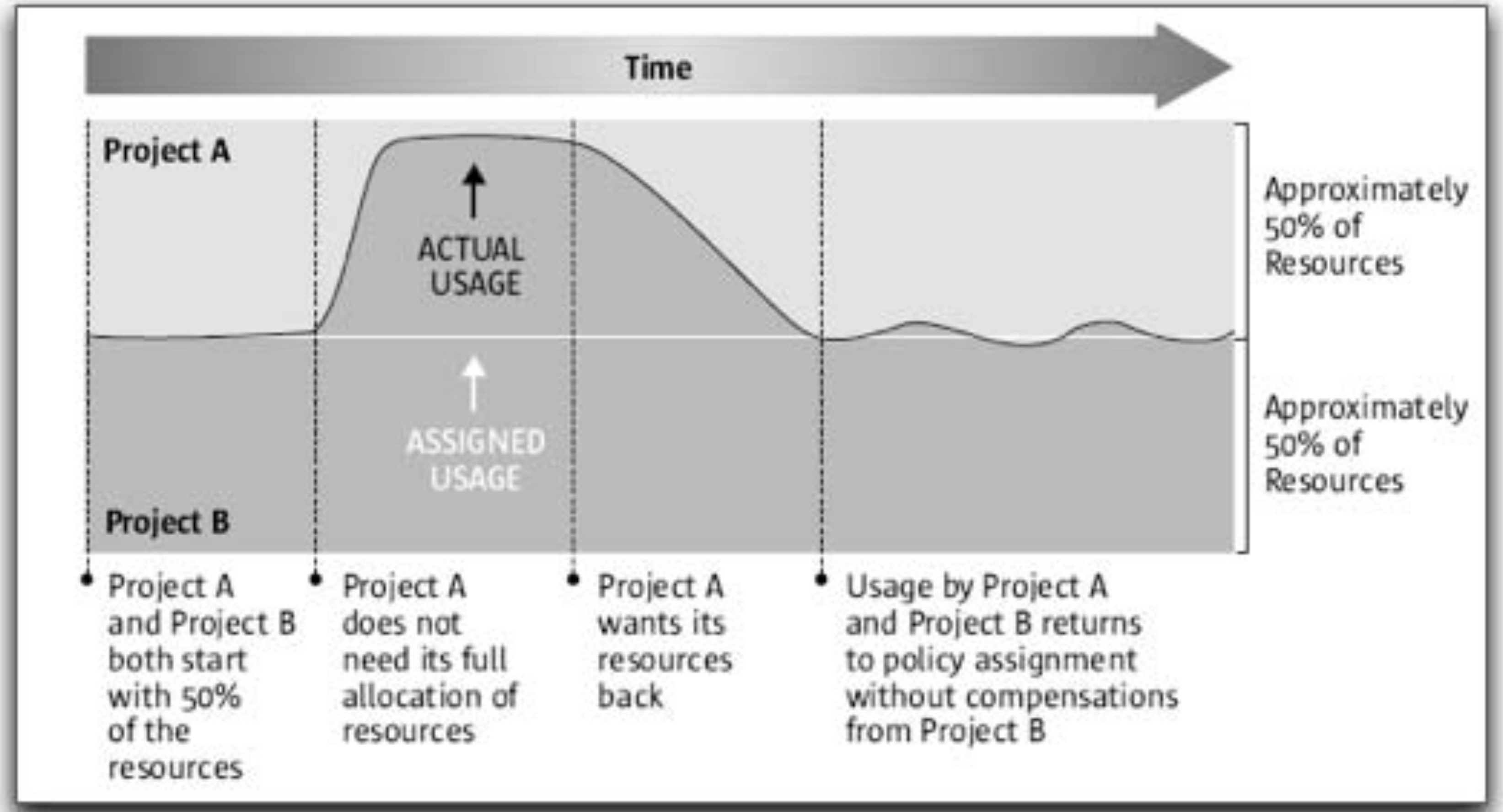
Functional Policy

- Strict division of available resources
  - “Group A gets 50%, Group B gets 50%”
  - Can also do “fair share”
- Can involve
  - User / User Group affiliation
  - Department
  - Project

Share Tree Policy

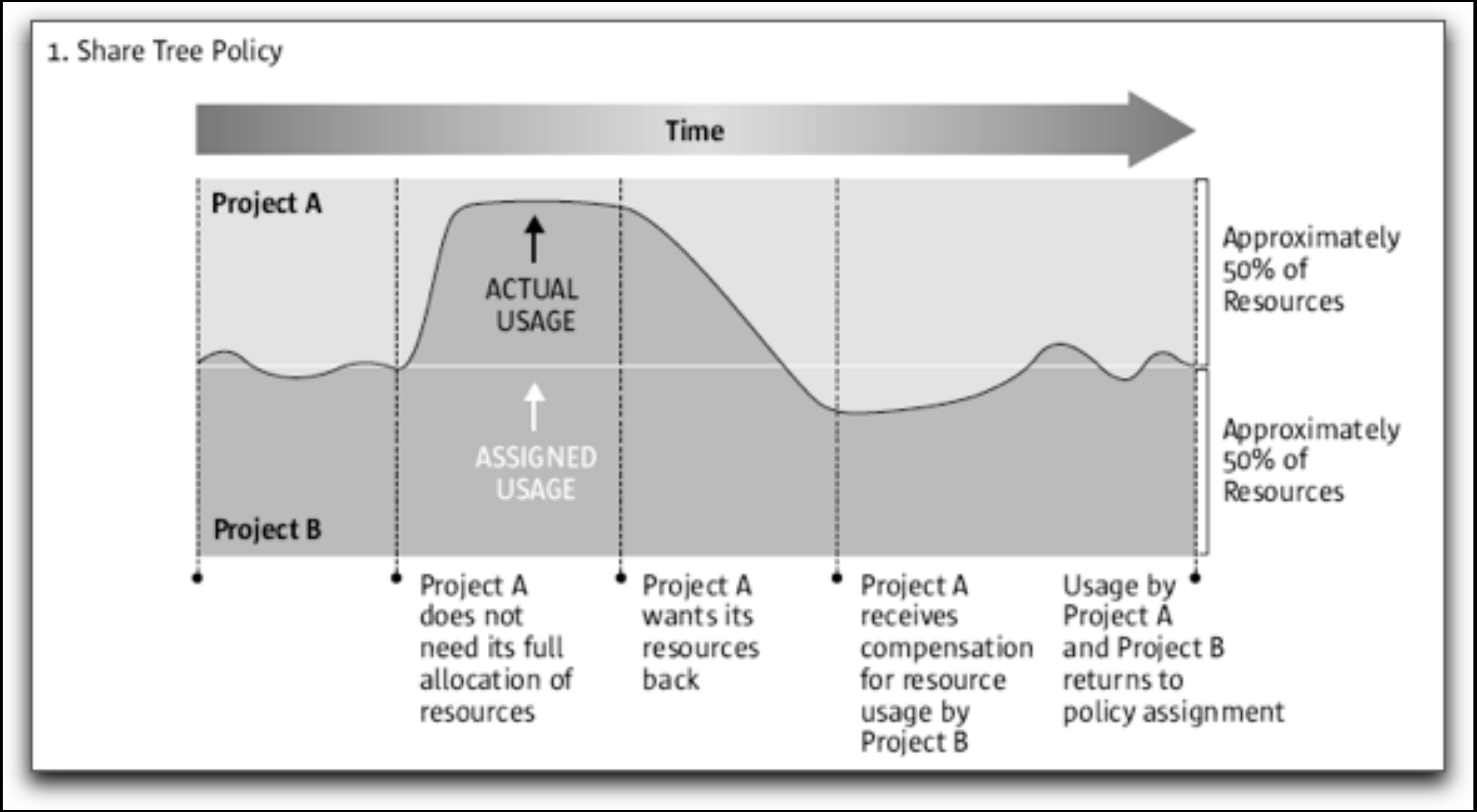
- More flexible than Functional Policy
- Past and current usage considered when making scheduling decisions
- Scheduler works to satisfy service level & entitlement policies *as averaged over time*

Functional Policy Behavior


Time
Project A
ACTUAL USAGE
ASSIGNED USAGE
Project B
Approximately 50% of Resources
Approximately 50% of Resources
Project A and Project B both start with 50% of the resources
Project A does not need its full allocation of resources
Project A wants its resources back
Usage by Project A and Project B returns to policy assignment without compensations from Project B

Source: http://www.sun.com/products-n-solutions/edu/whitepapers/pdf/web_services_for_HPC.pdf

Sharetree Policy Behavior


1. Share Tree Policy
Time
Project A
ACTUAL USAGE
ASSIGNED USAGE
Project B
Approximately 50% of Resources
Approximately 50% of Resources
Project A does not need its full allocation of resources
Project A wants its resources back
Project A receives compensation for resource usage by Project B
Usage by Project A and Project B returns to policy assignment

Source: http://www.sun.com/products-n-solutions/edu/whitepapers/pdf/web_services_for_HPC.pdf

Tickets

Grid Engine Tickets

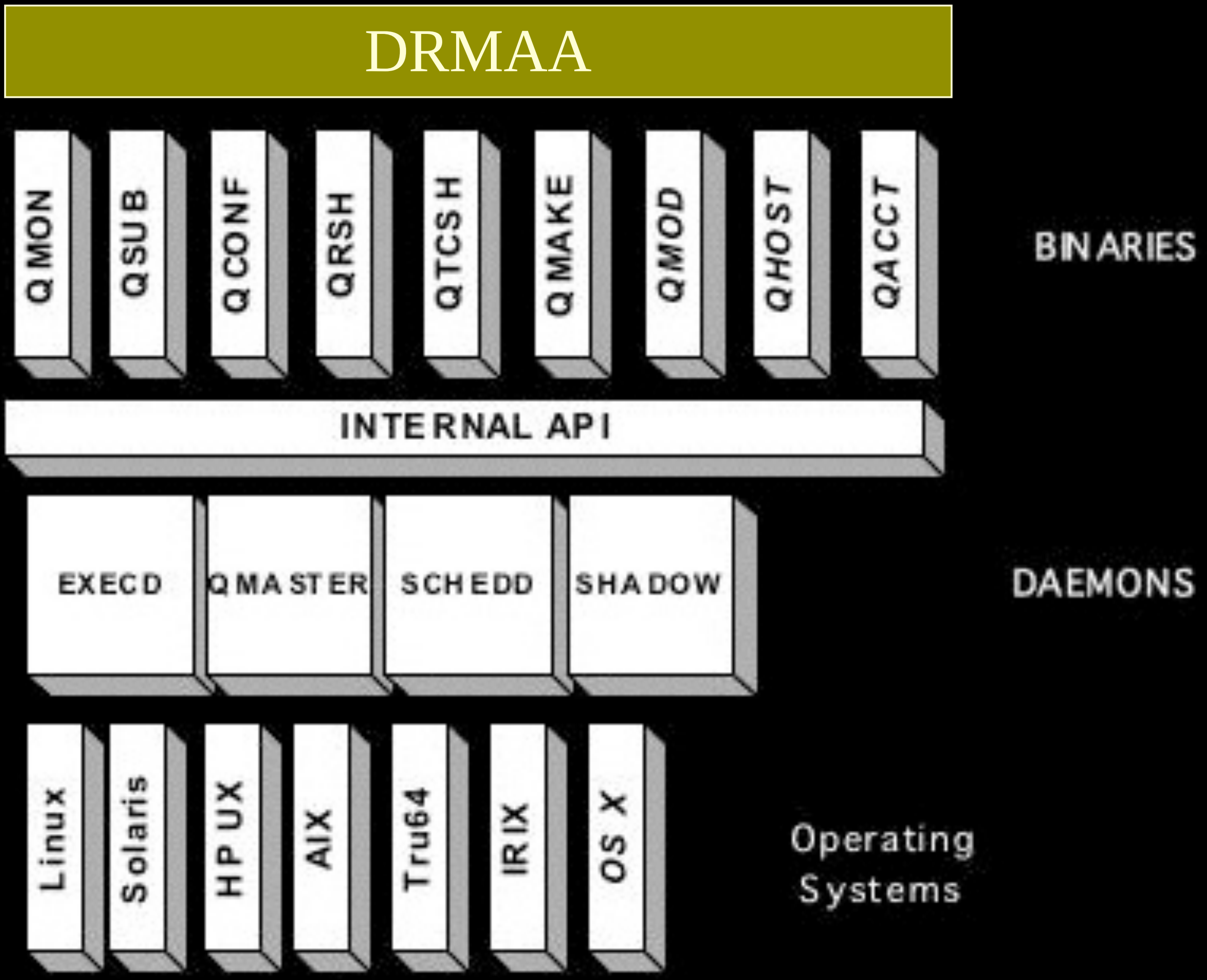
- All policies are defined using “tickets”
- Jobs get tickets from all the various policies
- Jobs with more tickets are more important
- Administrator controls the total number of tickets in the system
  - # of tickets assigned to each policy determines how “important” each of the different available policies are
  - To disable a policy within scheduler, assign zero tickets to it

Simplified Priority Formula

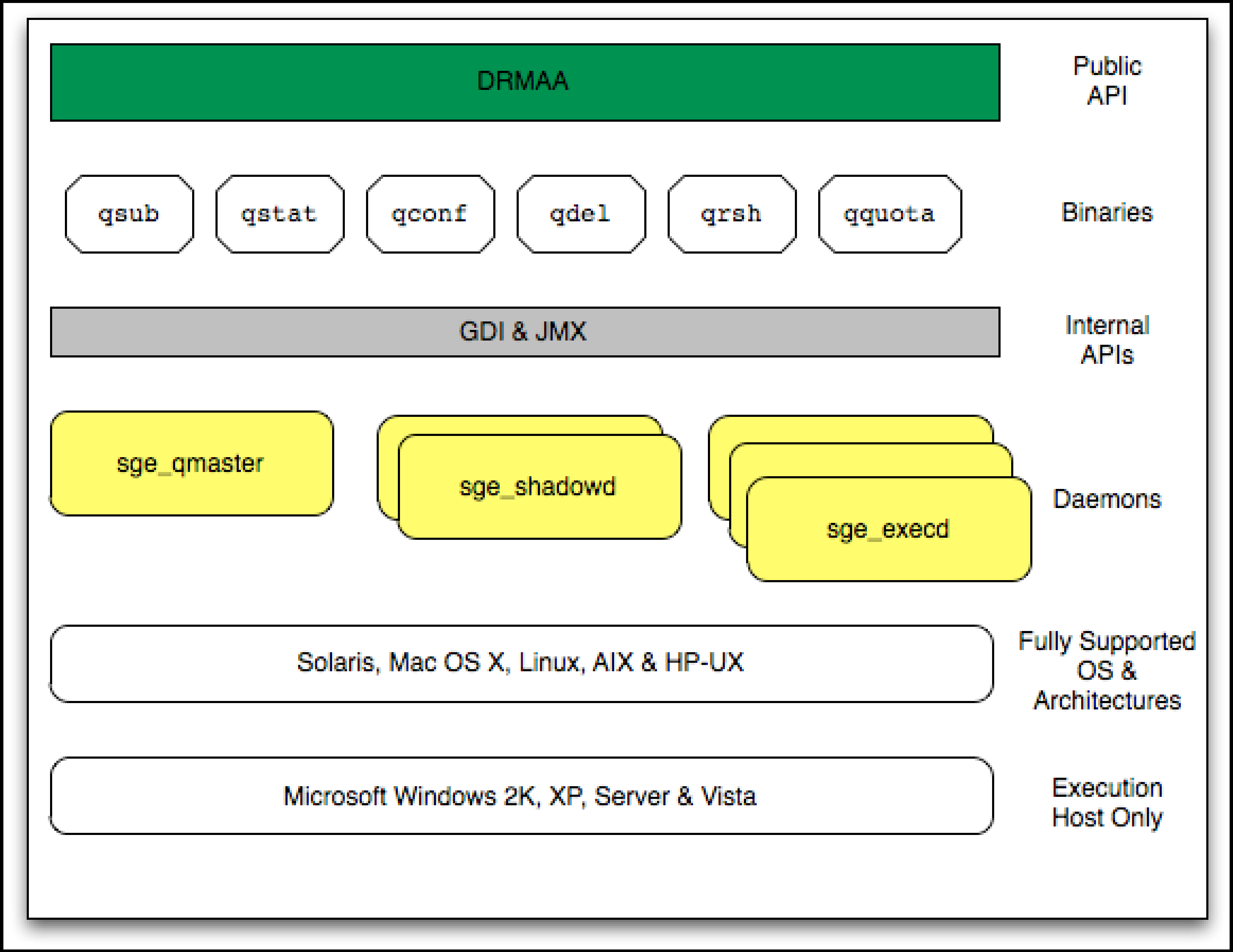
```
(weight_urgency  * normalized_urgency_val) +
(weight_ticket   * normalized_ticket_val)  +
(weight_priority * normalized_priority_val)
 = job_priority
```

Grid Engine Architecture

SGE Architecture Stack (old)



SGE 6 Architecture Stack



Fully Supported Platforms

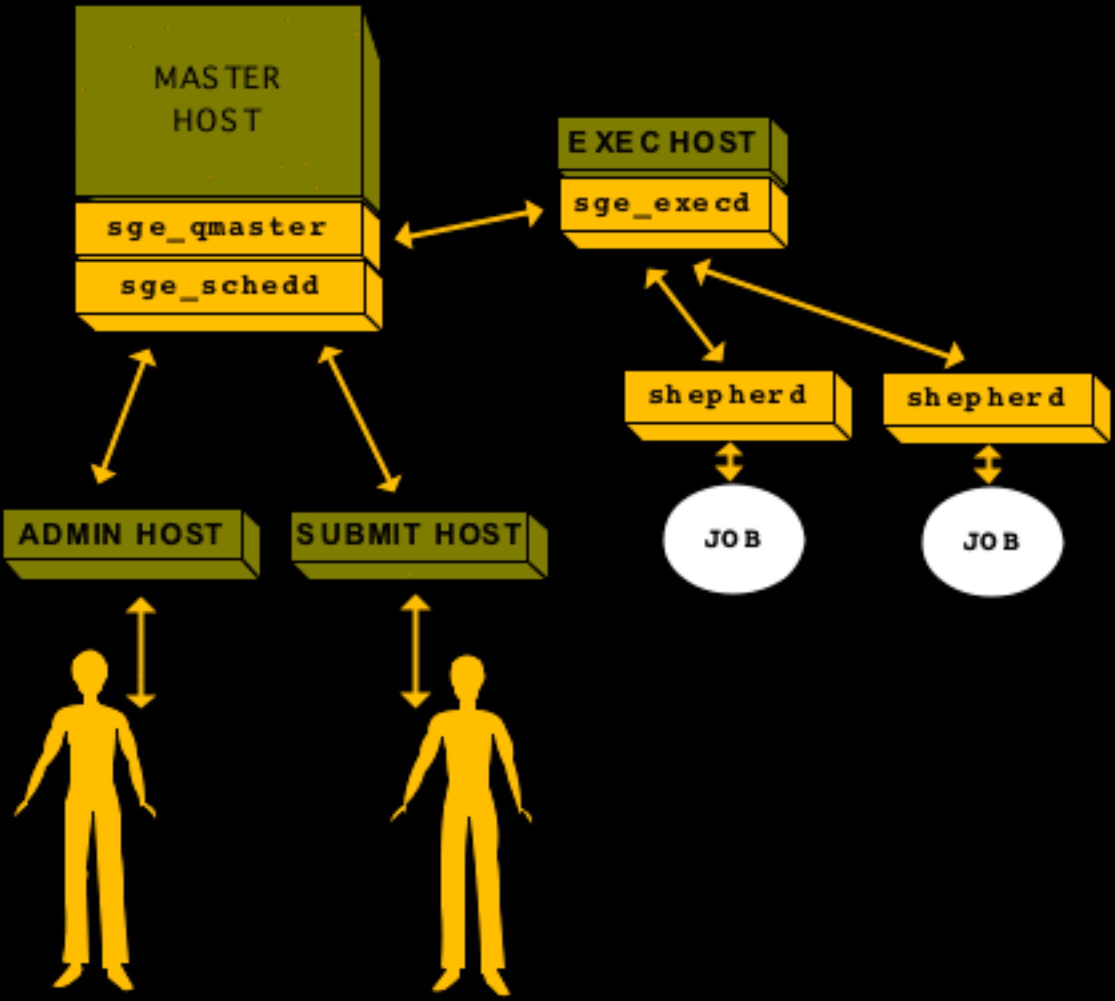
- Solaris 10, 9, and 8 Operating Systems (SPARC Platform Edition)
- Solaris 10 and 9 Operating Systems (x86 Platform Edition)
- Solaris 10 Operating System (x64 Platform Edition)
- Apple Mac OS X 10.4 (Tiger), PPC platform
- Apple Mac OS X 10.4 (Tiger), x86 platform
- Apple Mac OS X 10.5 (Leopard), x86 platform
- Hewlett Packard HP-UX 11.00 or higher, 32 bit
- Hewlett Packard HP-UX 11.00 or higher, 64 bit (including IA64)
- IBM AIX 5.1, 5.3
- Linux x86, kernel 2.4, 2.6, glibc >= 2.3.2
- Linux x64, kernel 2.4, 2.6, glibc >= 2.3.2
- Linux IA64, kernel 2.4, 2.6, glibc >= 2.3.2

Partially Supported Platforms

- Submit, Admin & Execution Hosts Only:
    - *(No DRMAA)*
    - Microsoft Windows Server 2003
    - Windows XP Professional SP1
    - Windows 2000 Server SP3
    - Windows 2000 Professional SP3
    - Microsoft Windows Server 2003 R2
    - Windows Server 2008
    - Windows Vista Enterprise
    - Windows Vista Ultimate

Host Types & SGE Daemons

- Exec hosts run 'sge_execd'
- Master runs 'sge_qmaster' *'sge_schedd' ***
- Master optionally can run 'sge_execd'



sge_qmaster daemon

- Controls overall behavior of the cluster
  - Dan says: “The ears, not the brain!”
- Multi-threaded since SGE 6.0
- Holds most current state about jobs, queues and other SGE objects
- Interoperates with sge_schedd for scheduling activities
- Answers requests from clients
- Delivers dispatched jobs to proper sge_execd daemons

sge_schedd daemon *or thread***

- Dedicated scheduling daemon
- Currently single threaded
  - *Remember: In SGE 6.2 scheduler operates as a thread within the qmaster*
- Matches pending jobs to available resources and queues
- During startup:
  - Contacts sge_qmaster; gets full state info
- After startup:
  - Gets new state changes from qmaster
  - Updates internal state information
  - Performs a scheduling run
  - Sends job ‘order list’ back to qmaster
  - Rinse, repeat …

sge_execd daemon

- 1 daemon per execution host (compute node)
- Controls all running jobs
  - Including suspend/resume & reprioritization
- Collects job information
  - Resource consumption, exit code, etc.
- Collects & reports host information to qmaster
  - System load, free memory, etc.

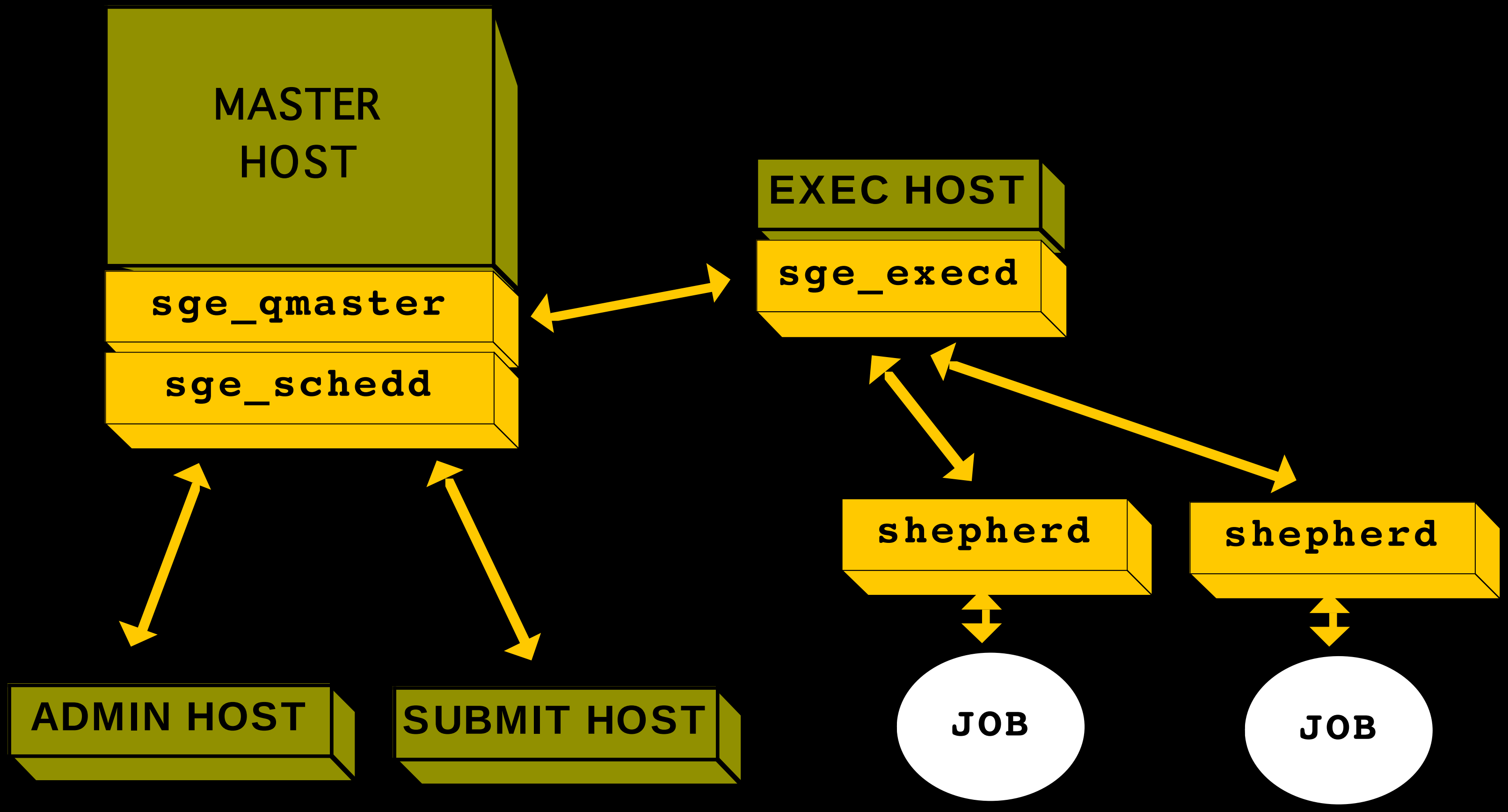
sge_execd daemon

- During startup:
  - Tries to connect/register with sge_qmaster
    - If failure, loop, log & retry
  - Looks for old jobs
    - If found, establishes process control again
  - Cleanup finished jobs, report to qmaster
- After startup:
  - Loop
    - Receive request
    - Process request
    - Report to qmaster

shepherd process

- Invoked by sge_execd to handle a single job
- Sets up resources before jobs begin
  - Prolog script, Parallel environment startup, etc.
- Uses ‘setuid’ behavior to *become the user*
- Starts the job as a child process
- Controls the job
  - Suspend/resume/terminate signals
  - Handles checkpointing
- Cleanup after job termination
  - Epilog scripts, Parallel environment shutdown, etc.

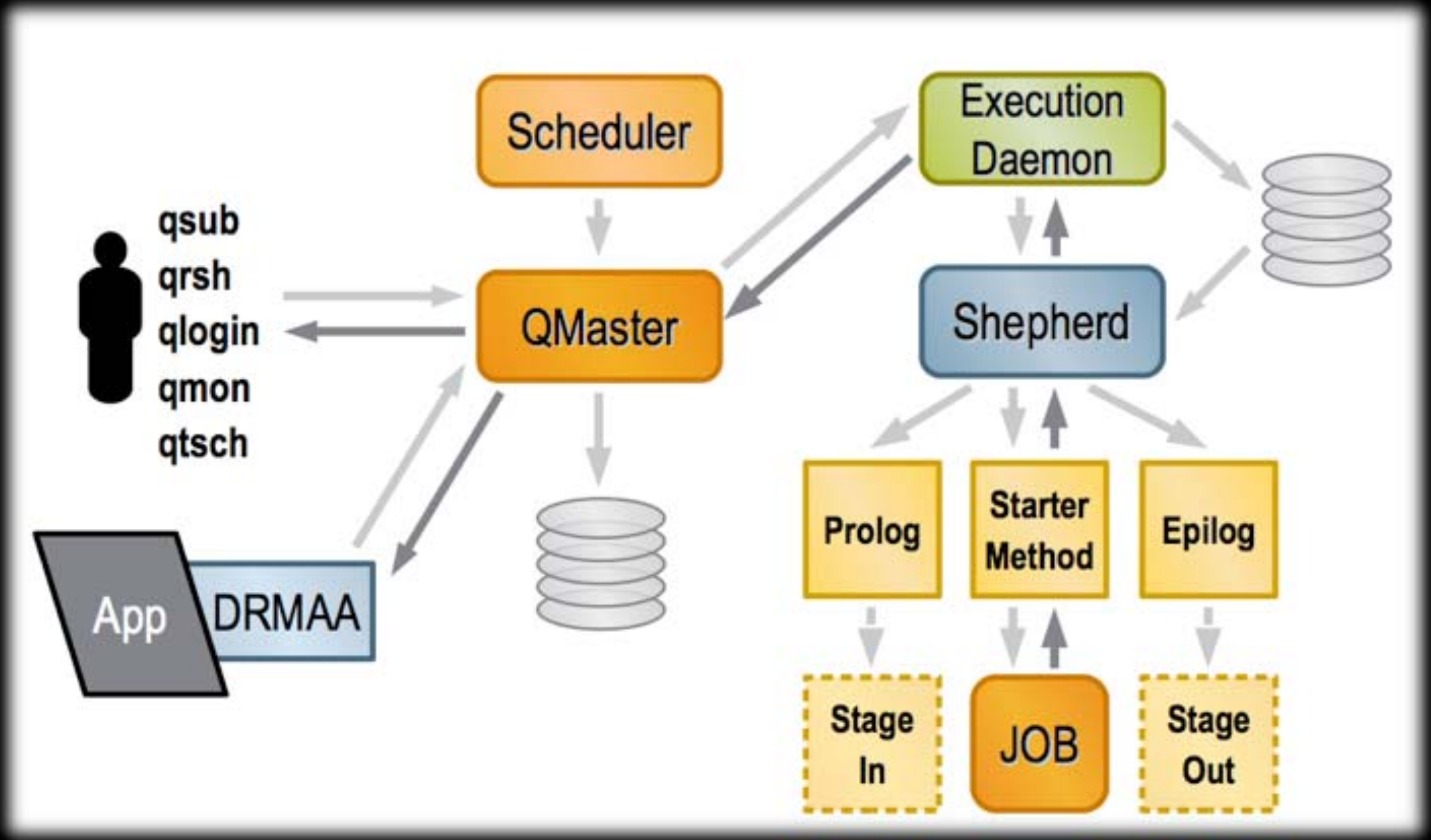
Putting it all together



Step by step

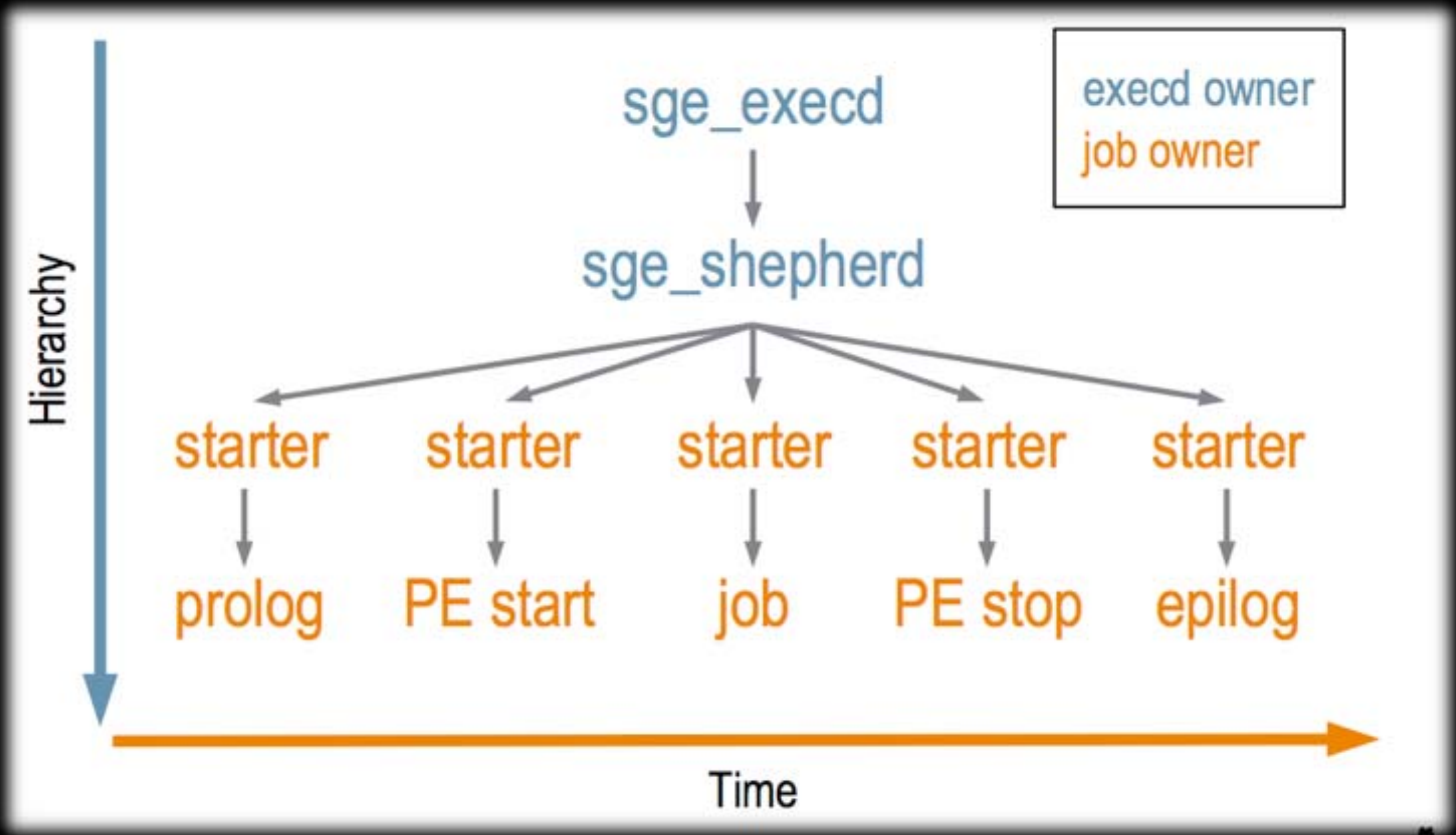
1. Job submitted from submit host or via DRMAA API
2. User identity, cwd, environment preserved
3. Master host receives job request, places entry in database and notifies the scheduler
4. Scheduler attempts to dispatch job to best possible queue. If successful, queue is returned to the `qmaster` daemon
5. Qmaster sends job to the proper `sge_execd`
6. `sge_execd` spawns a sge_shepherd process; suid() pivot
7. Shepherd sets up job, including paths, cwd and environment variables recorded at submission time. The shepherd starts the job, monitors resource usage data and exit status. Shepherd also performs post-job cleanup tasks.
8. Exit status & accounting data passed back to `sge_execd` and `sge_qmaster`

Step by step (in pictures)



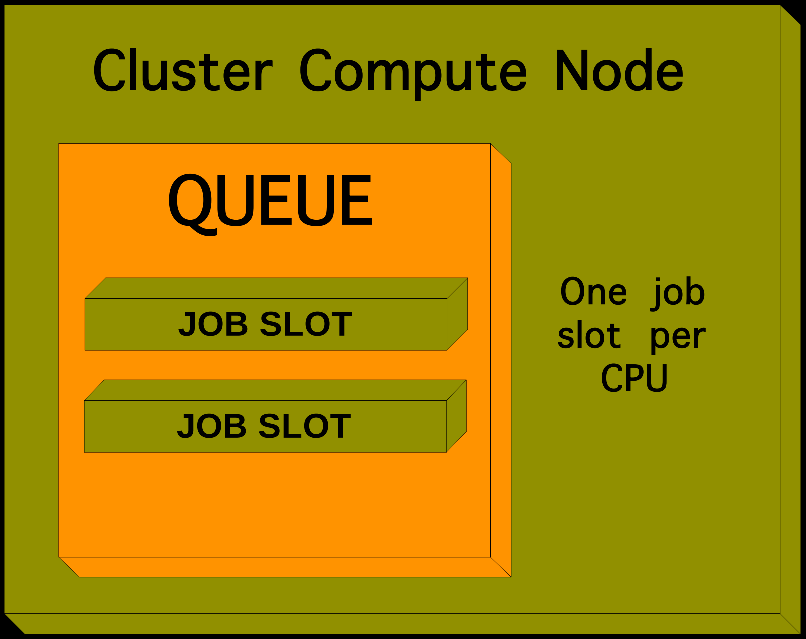
Graphic credit: Dan Templeton

Process Hierarchy (thanks Dan!)



Cliché/Default Configuration

- 1 queue instance per compute server.
- Job slots == number of CPUs in machine
- Items such as Flexlm license attributes can be attached as global consumable ‘complexes’



Grid Engine Scheduling II

Scheduling

- Jobs get scheduled based on:
  - Job priority / importance
  - Cluster and individual host load
  - Job resource request requirements
- Without active policies, behavior is FIFO

Scheduling tasks you can influence

- Dynamic / Automatic scheduling
  - Via: Functional, Sharetree, Override policies
- Queue Sorting
  - Via: Influence order in which queues are filled
  - Load vs. seqno vs. Hybrid
- Job Sorting
  - Via: Adjusting the mechanisms responsible for sorting pending jobs into dispatch order *(Urgency,wait_waiting_time, override, etc.)*
- Reservation & Backfilling
  - Via: backfill short jobs into blocks reserved for large jobs

Automatic/Dynamic Methods

- Standard “policy based scheduling” mechanisms
  - Functional
  - Share Tree
  - Override
- Controlled by “tickets”
  - Pro: Any policy, any order, any weight
  - Cons: Complex setups can become opaque and difficult to troubleshoot

Queue Sorting

- Load reporting
  - Lots of control over the specific parameters used by scheduler to compare load status among hosts
- Load scaling
  - Possible to override/alter scaling parameters used to normalize values between machines of different “speeds”
- Load adjustment
  - Influence artificial load spikes & decay times to prevent host oversubscription
- SeqNo Sorting
  - Sort viable nodes by strict integer based sequence ranks
  - “Big Hammer” method

Job Sorting

- Ticket based priority
  - Pending jobs are sorted by ticket value
- Urgency based priority
  - Gain additional entitlement from one or more of the urgency sub-policies
- POSIX Priority
  - Site specific priority values -1023 to 1024
- max_u_jobs etc.

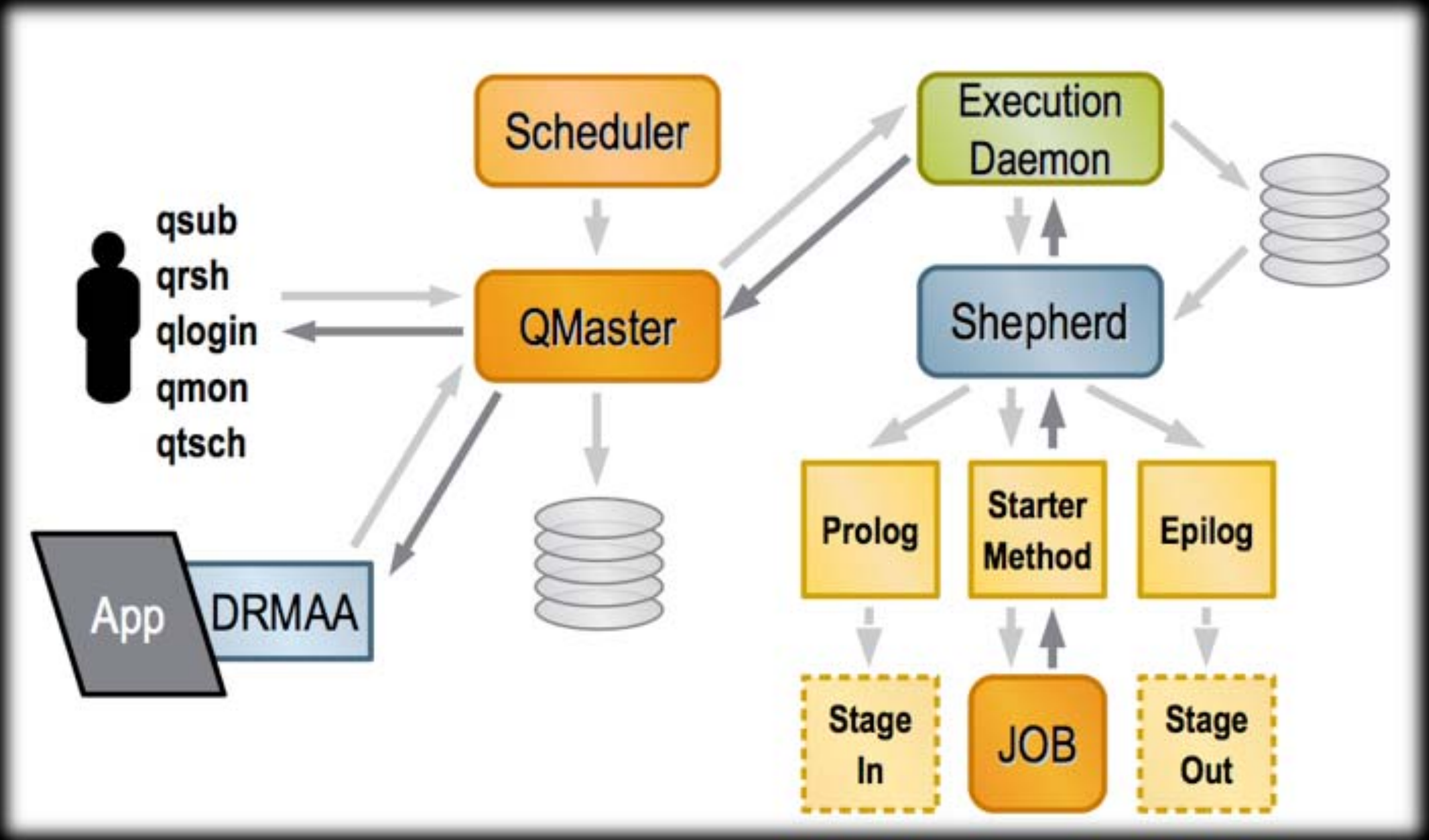
It all comes down to this …

- On an idle cluster everyone gets whatever they need
- On a contested cluster the scheduler will work to enforce configured policies
- How this is done:
  - Manipulating the order of jobs waiting in the pending list

Power users can also ...

- Use the POSIX Priority policy to rank the importance of pending jobs
- Huh?
  - You can tell Grid Engine which among your pending jobs are most important.
  - Assign within range [*-1023 to +1024*]**
    - *Only managers allowed to use values 0-1024*
  - Used (if needed) to further sort/prioritize within your “immediate”, “normal” or “background” jobs

End; Overview



Graphic credit: Dan Templeton