

Grid Compute Resources and Job Management



Job and compute resource management

- This module is about running jobs on remote compute resources

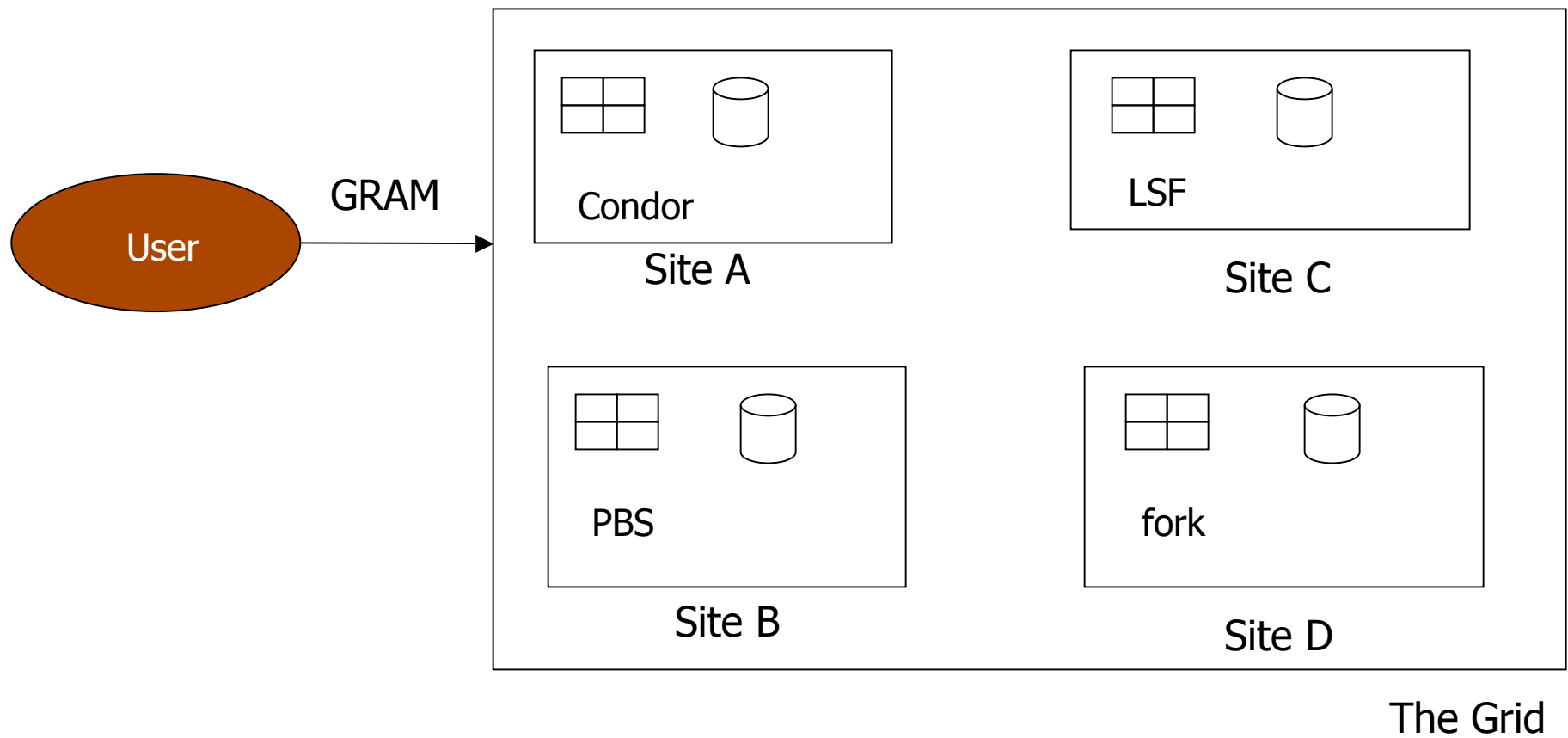
Job and resource management

- Compute resources have a local resource manager
 - This controls who is allowed to run jobs and how they run, on a resource
- GRAM
 - Helps us run a job on a remote resource
- Condor
 - Manages jobs

Local Resource Managers

- Local Resource Managers (LRMs) – software on a compute resource such a multi-node cluster.
- Control which jobs run, when they run and on which processor they run
- Example policies:
 - Each cluster node can run one job. If there are more jobs, then the other jobs must wait in a queue
 - Reservations – maybe some nodes in cluster reserved for a specific person
- eg. PBS, LSF, Condor

Job Management on a Grid



GRAM

- *Globus Resource Allocation Manager*
- Provides a standardised interface to submit jobs to different types of LRM
- Clients submit a job request to GRAM
- GRAM translates into something the LRM can understand
- Same job request can be used for many different kinds of LRM

GRAM

- Given a job specification:
 - ❑ Create an environment for a job
 - ❑ Stage files to and from the environment
 - ❑ Submit a job to a local resource manager
 - ❑ Monitor a job
 - ❑ Send notifications of the job state change
 - ❑ Stream a job's stdout/err during execution

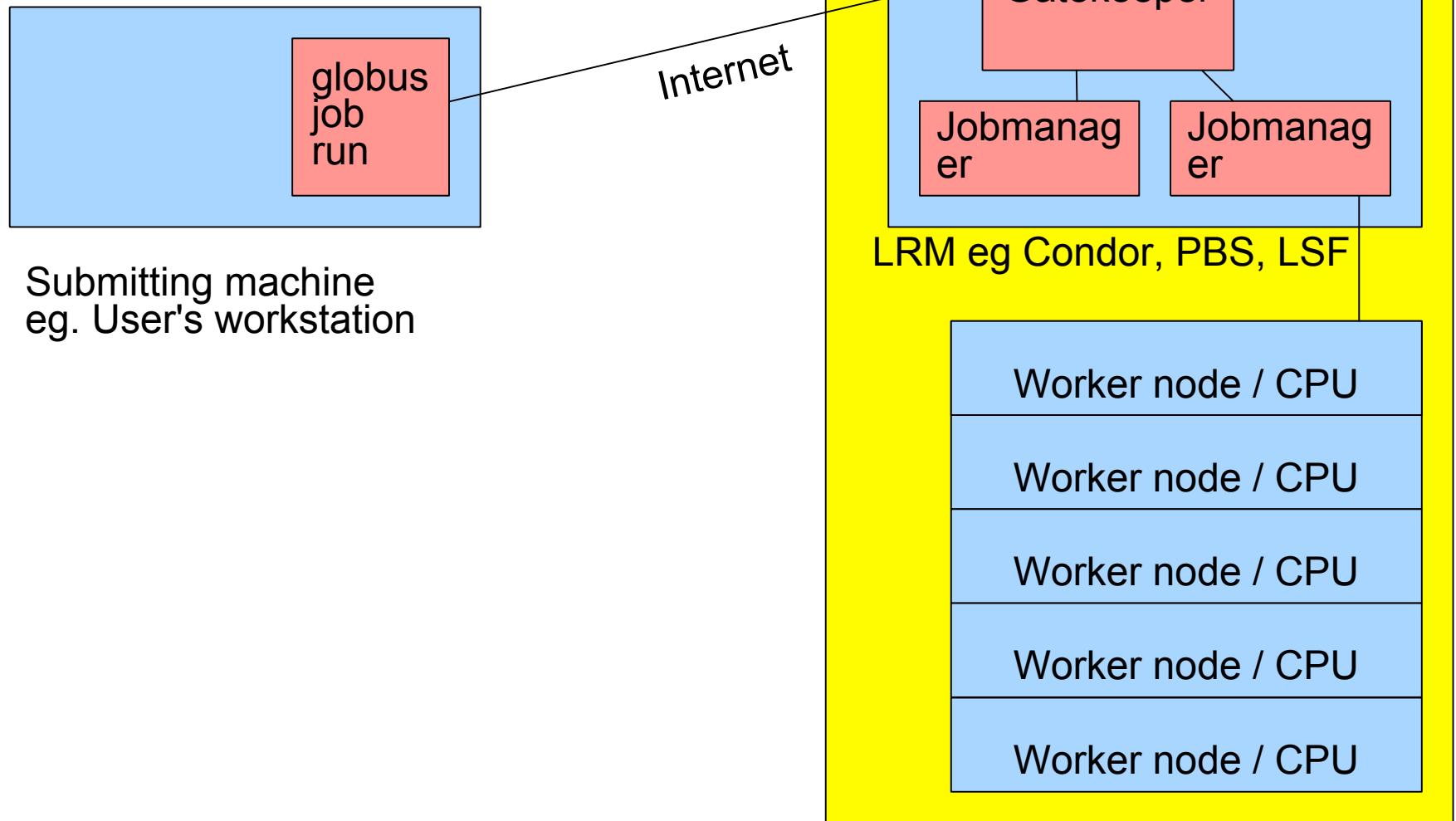
Two versions of GRAM

- There are two versions of GRAM
 - GRAM2
 - Own protocols
 - Older
 - More widely used
 - No longer actively developed
 - GRAM4
 - Web services
 - Newer
 - New features go into GRAM4
- In this module, will be using GRAM2

GRAM components

- Clients – eg. Globus-job-submit, globusrun
- Gatekeeper
 - Server
 - Accepts job submissions
 - Handles security
- Jobmanager
 - Knows how to send a job into the local resource manager
 - Different job managers for different LRMs

GRAM components



Submitting a job with GRAM

- Globus-job-run command
- `globus-job-run rookery.uchicago.edu /bin/hostname rook11`
- Run '/bin/hostname' on the resource `rookery.uchicago.edu`
- We don't care what LRM is used on 'rookery'. This command works with any LRM.

The client can describe the job with GRAM's Resource Specification Language (RSL)

■ Example:

```
& (executable = a.out)
(directory = /home/nobody )
(arguments = arg1 "arg 2")
```

■ Submit with:

```
globusrun -f spec.rsl -r
rookery.uchicago.edu
```

Use other programs to generate RSL

- RSL job descriptions can become very complicated
- We can use other programs to generate RSL for us
- Example: Condor-G – next section

Condor

- Globus-job-run submits jobs, but...
 - ❑ No job tracking: what happens when something goes wrong?
- Condor:
 - ❑ Many features, but in this module:
 - ❑ Condor-G for reliable job management

Condor can manage a large number of jobs

- Managing a large number of jobs
 - ❑ You specify the jobs in a file and submit them to Condor, which runs them all and keeps you notified on their progress
 - ❑ Mechanisms to help you manage huge numbers of jobs (1000's), all the data, etc.
 - ❑ Condor can handle inter-job dependencies (DAGMan)
 - ❑ Condor users can set job priorities
 - ❑ Condor administrators can set user priorities
- Can do this as:
 - ❑ a local resource manager on a compute resource
 - ❑ a grid client submitting to GRAM (Condor-G)

Condor can manage compute resource

- Dedicated Resources
 - Compute Clusters
- Non-dedicated Resources
 - Desktop workstations in offices and labs
 - Often idle 70% of time
- Condor acts as a Local Resource Manager



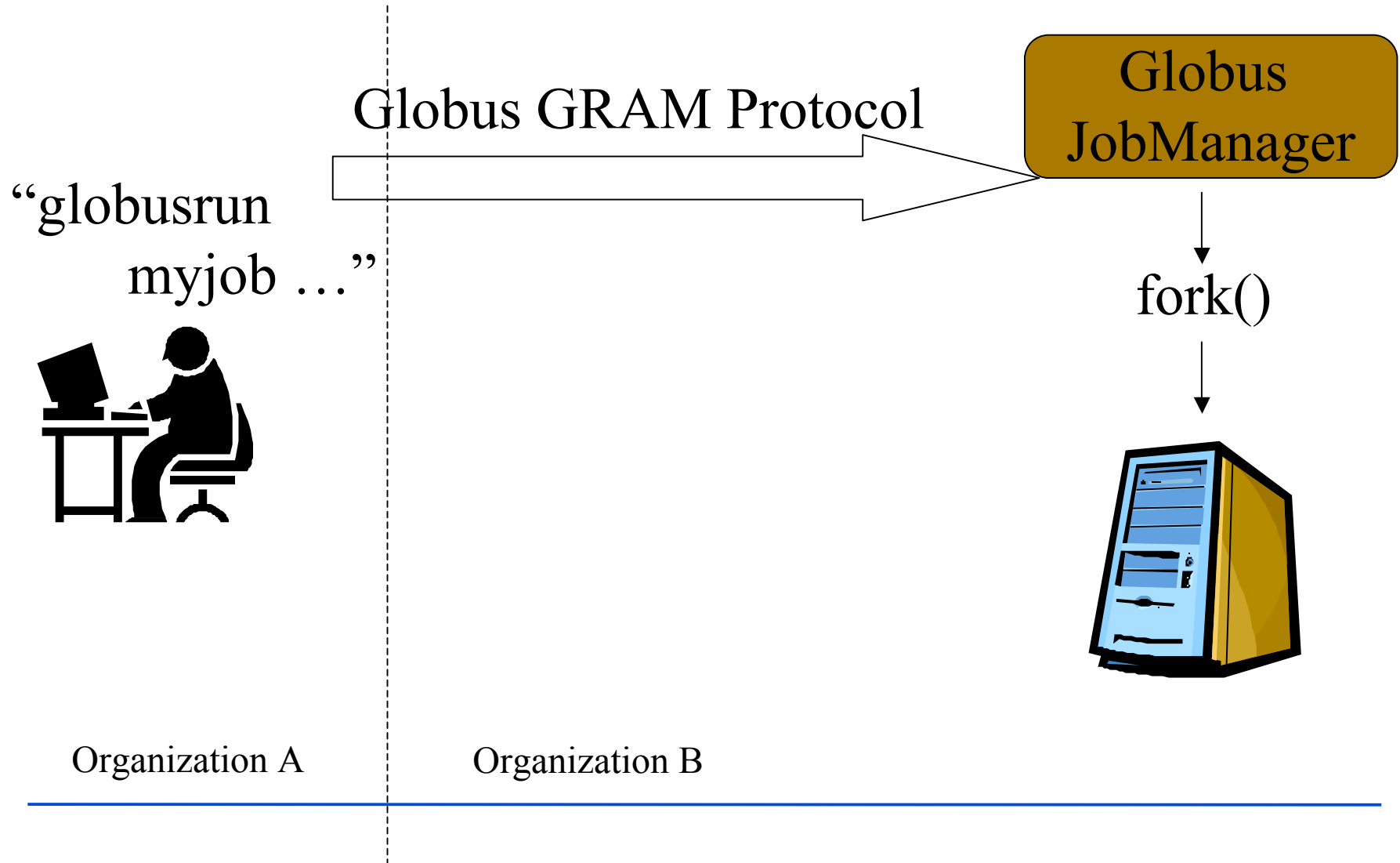
... and Condor Can Manage Grid jobs

- Condor-G is a specialization of Condor. It is also known as the “Grid universe”.
- Condor-G can submit jobs to Globus resources, just like globus-job-run.
- Condor-G benefits from Condor features, like a job queue.

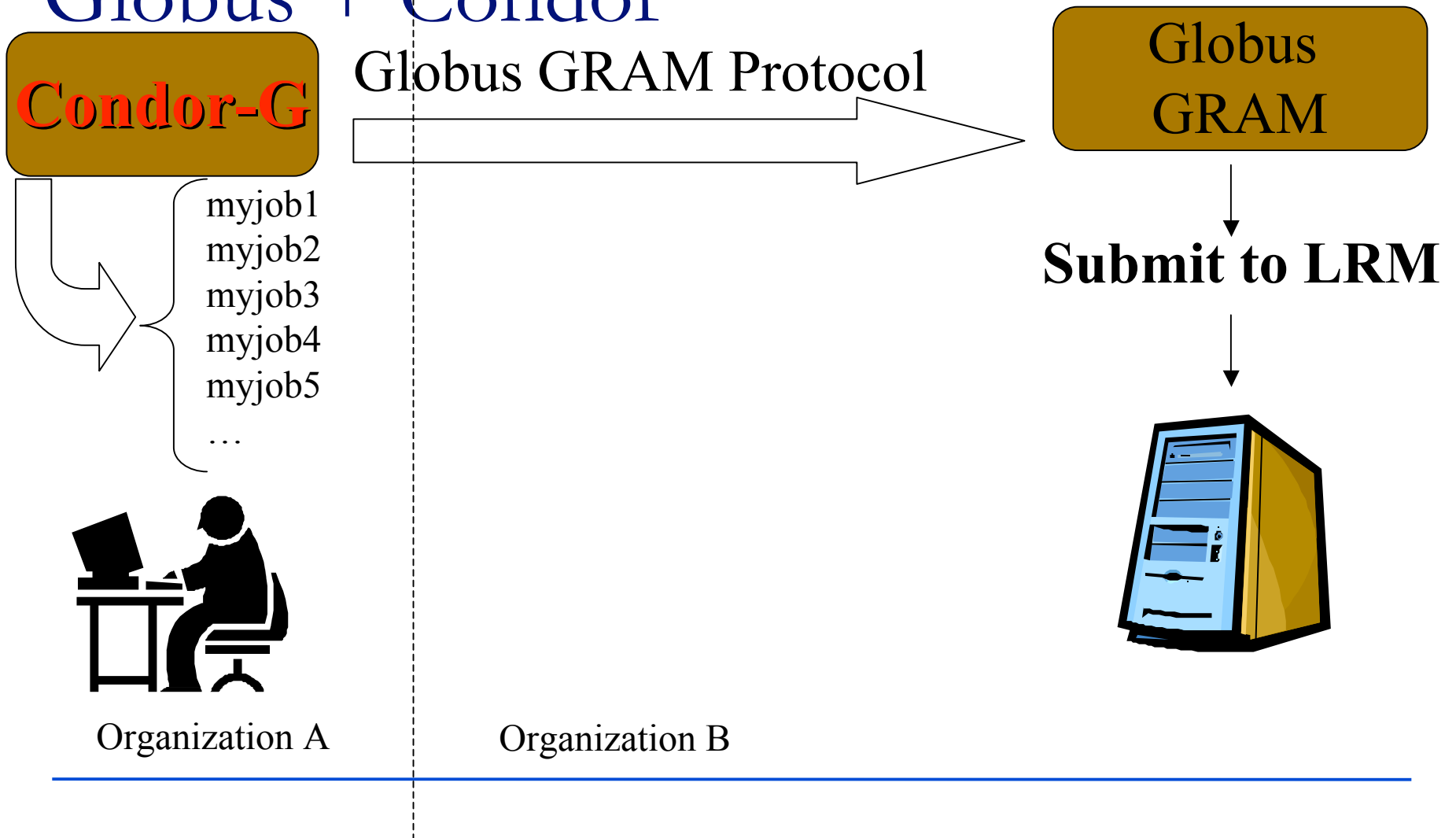
Some Grid Challenges

- Condor-G does whatever it takes to run your jobs, even if ...
 - ❑ The gatekeeper is temporarily unavailable
 - ❑ The job manager crashes
 - ❑ Your local machine crashes
 - ❑ The network goes down

Remote Resource Access: Globus



Remote Resource Access: Condor-G + Globus + Condor



Example Application ...

Simulate the behavior of $F(x,y,z)$ for 20 values of x , 10 values of y and 3 values of z ($20*10*3 = 600$ combinations)

- ❑ F takes on the average 3 hours to compute on a “typical” workstation (total = 1800 hours)
- ❑ F requires a “moderate” (128MB) amount of memory
- ❑ F performs “moderate” I/O - (x,y,z) is 5 MB and $F(x,y,z)$ is 50 MB
- ❑ 600 jobs

Creating a Submit Description File

- A plain ASCII text file
- Tells Condor about your job:
 - Which executable, universe, input, output and error files to use, command-line arguments, environment variables, any special requirements or preferences (more on this later)
- Can describe many jobs at once (a “cluster”) each with different input, arguments, output, etc.

Simple Submit Description File

```
# Simple condor_submit input file
# (Lines beginning with # are comments)
# NOTE: the words on the left side are not
#       case sensitive, but filenames are!
Universe      = vanilla
Executable    = my_job
Queue
```

```
$ condor_submit myjob.sub
```

Other Condor commands

- `condor_q` – show status of job queue
- `condor_status` – show status of compute nodes
- `condor_rm` – remove a job
- `condor_hold` – hold a job temporarily
- `condor_release` – release a job from hold

Condor-G: Access non-Condor Grid resources



- middleware deployed across entire Grid
- remote access to computational resources
- dependable, robust data transfer



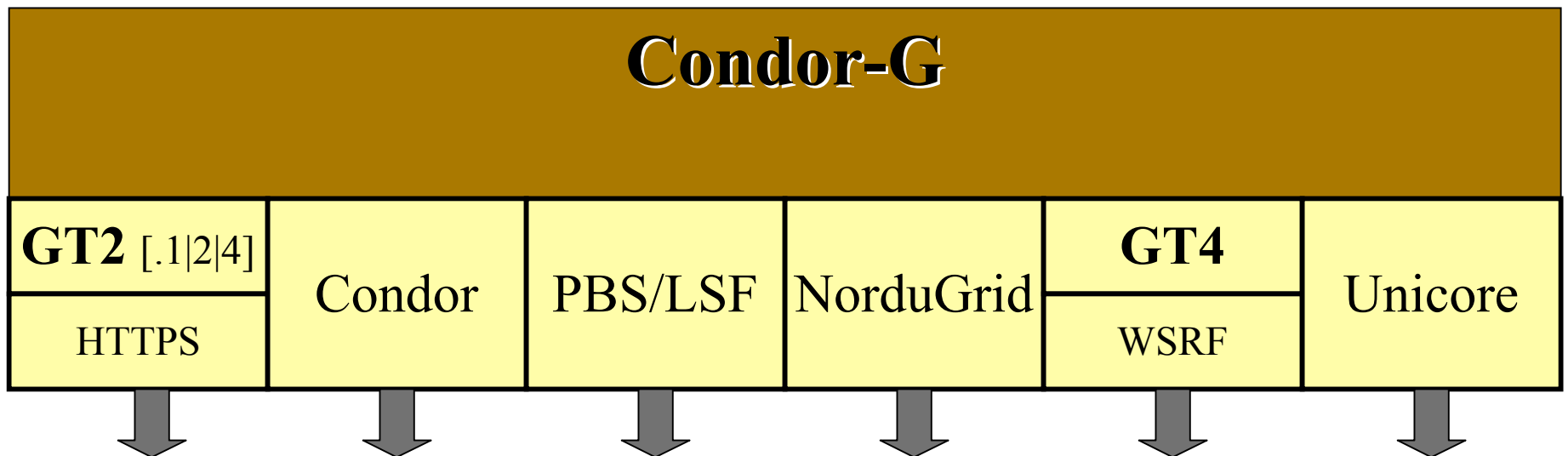
- job scheduling across multiple resources
- strong fault tolerance with checkpointing and migration
- layered over Globus as “personal batch system” for the Grid

Condor-G

Job Description (Job ClassAd)



Condor-G



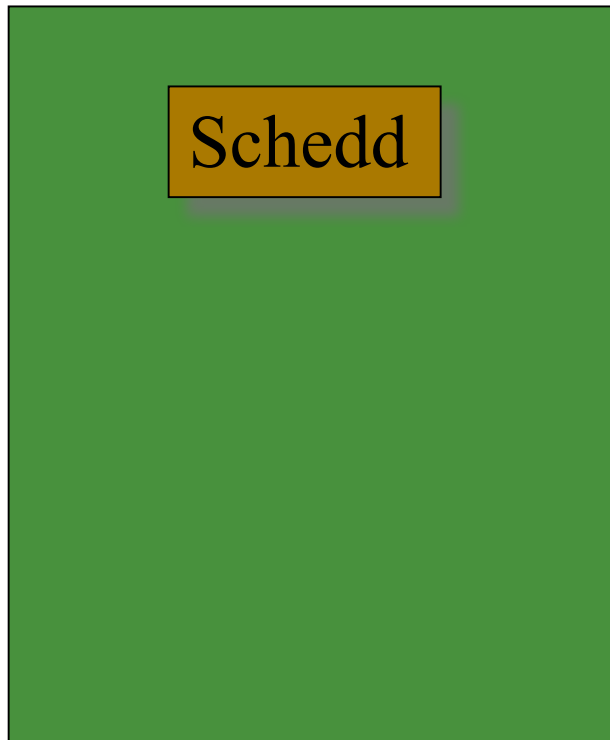
Submitting a GRAM Job

- In submit description file, specify:
 - Universe = grid
 - Grid_Resource = gt2 <gatekeeper host>
 - 'gt2' means GRAM2
 - Optional: Location of file containing your X509 proxy

```
universe      = grid
grid_resource = gt2 beak.cs.wisc.edu/jobmanager-pbs
executable    = progname
queue
```

How It Works

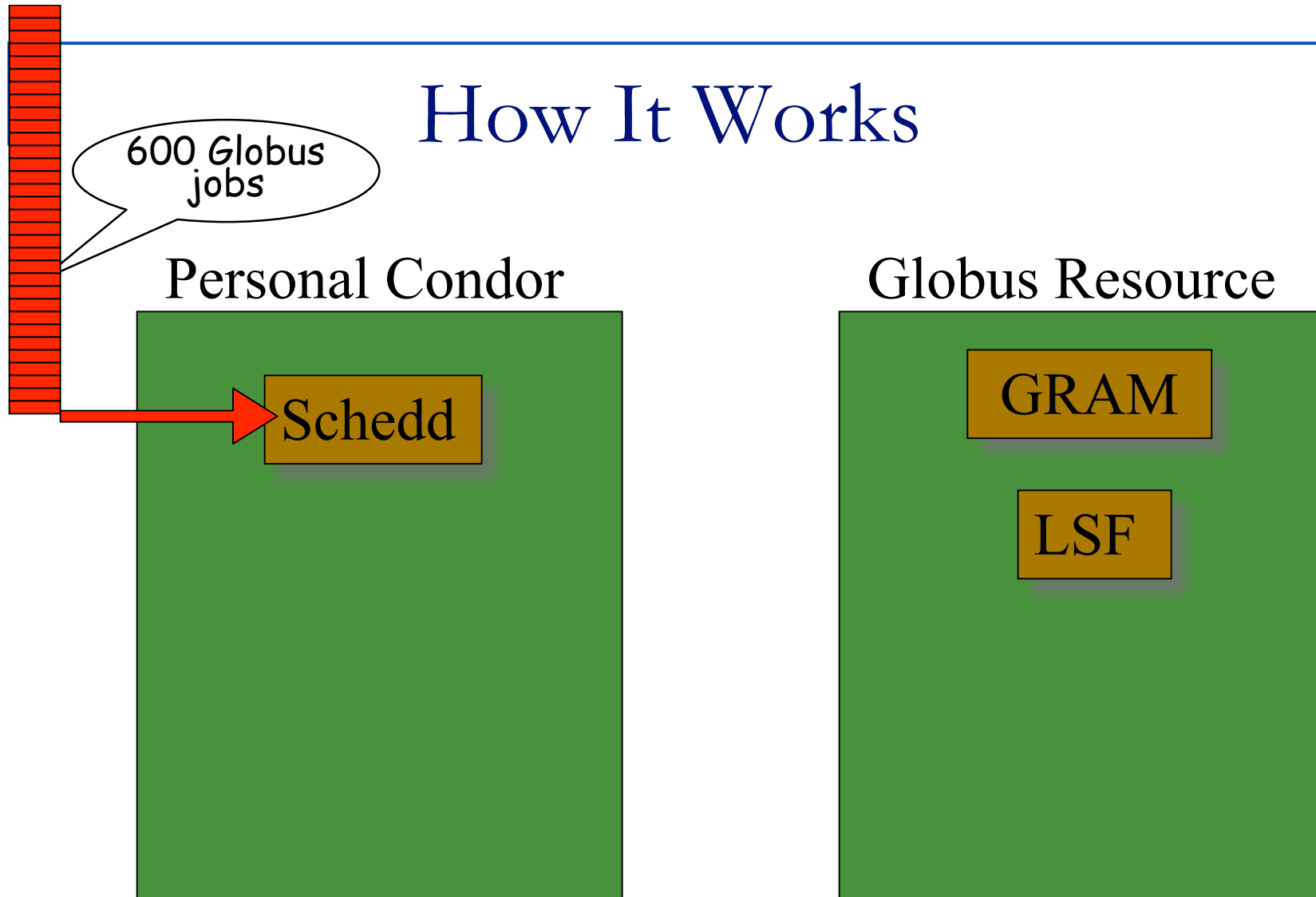
Personal Condor



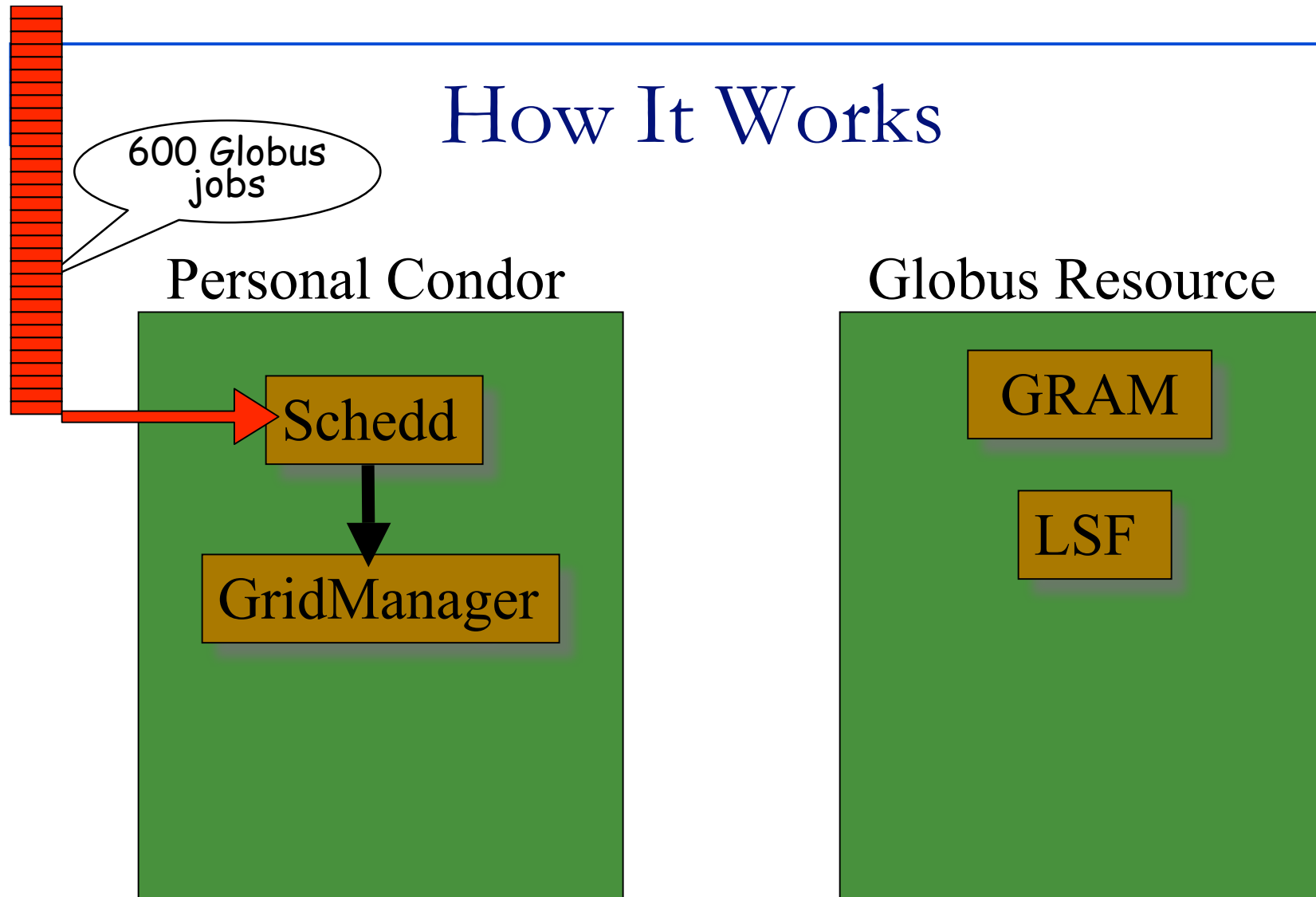
Globus Resource



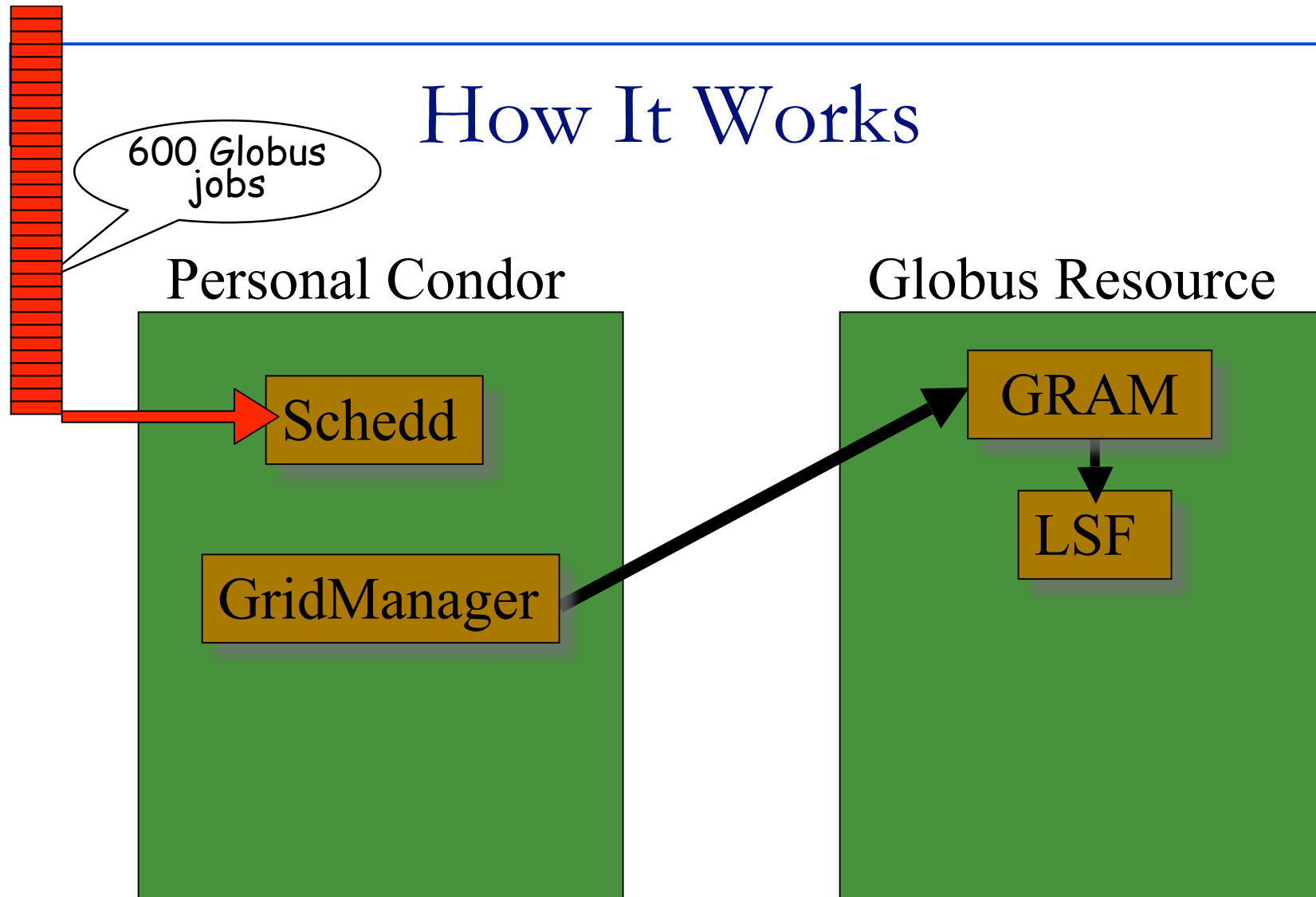
How It Works



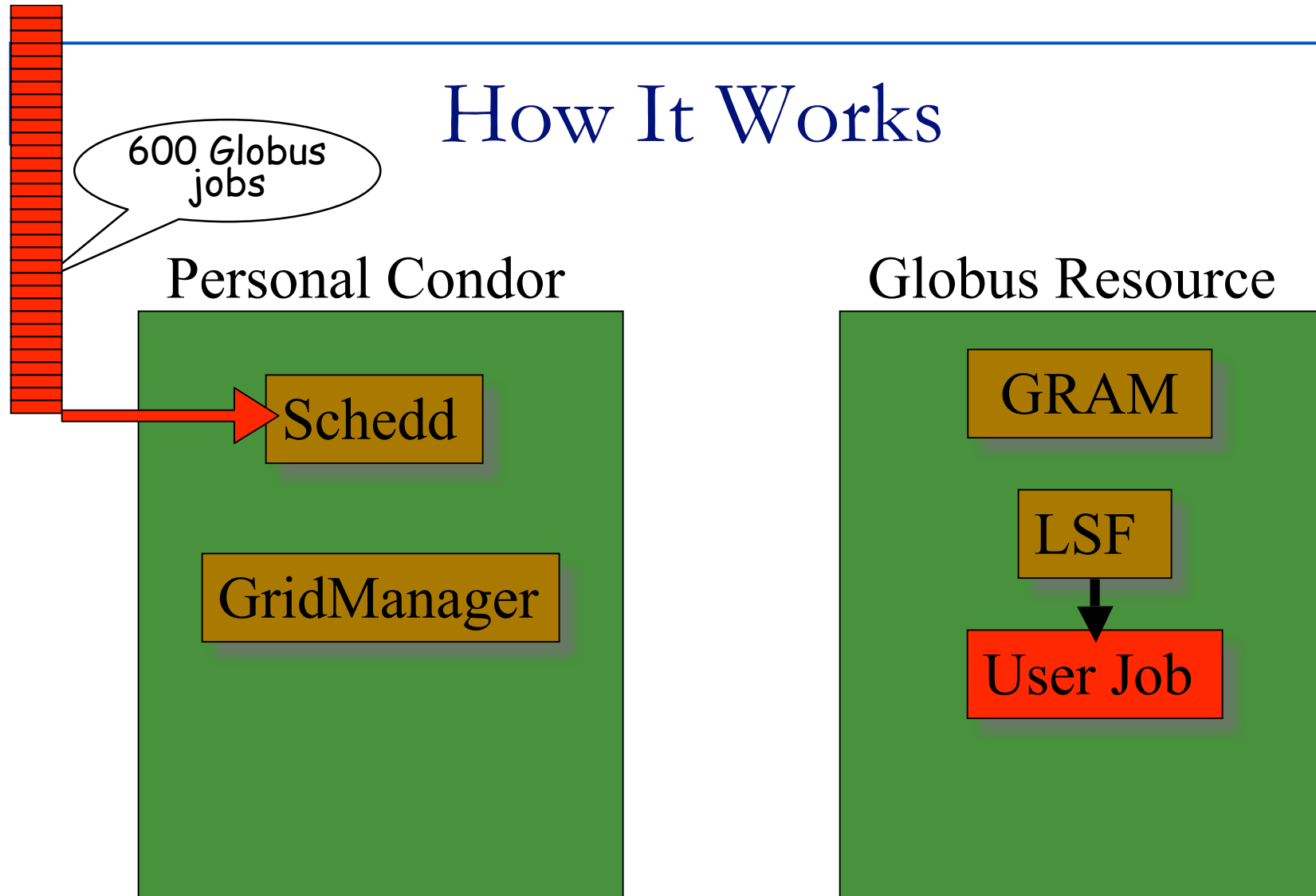
How It Works



How It Works



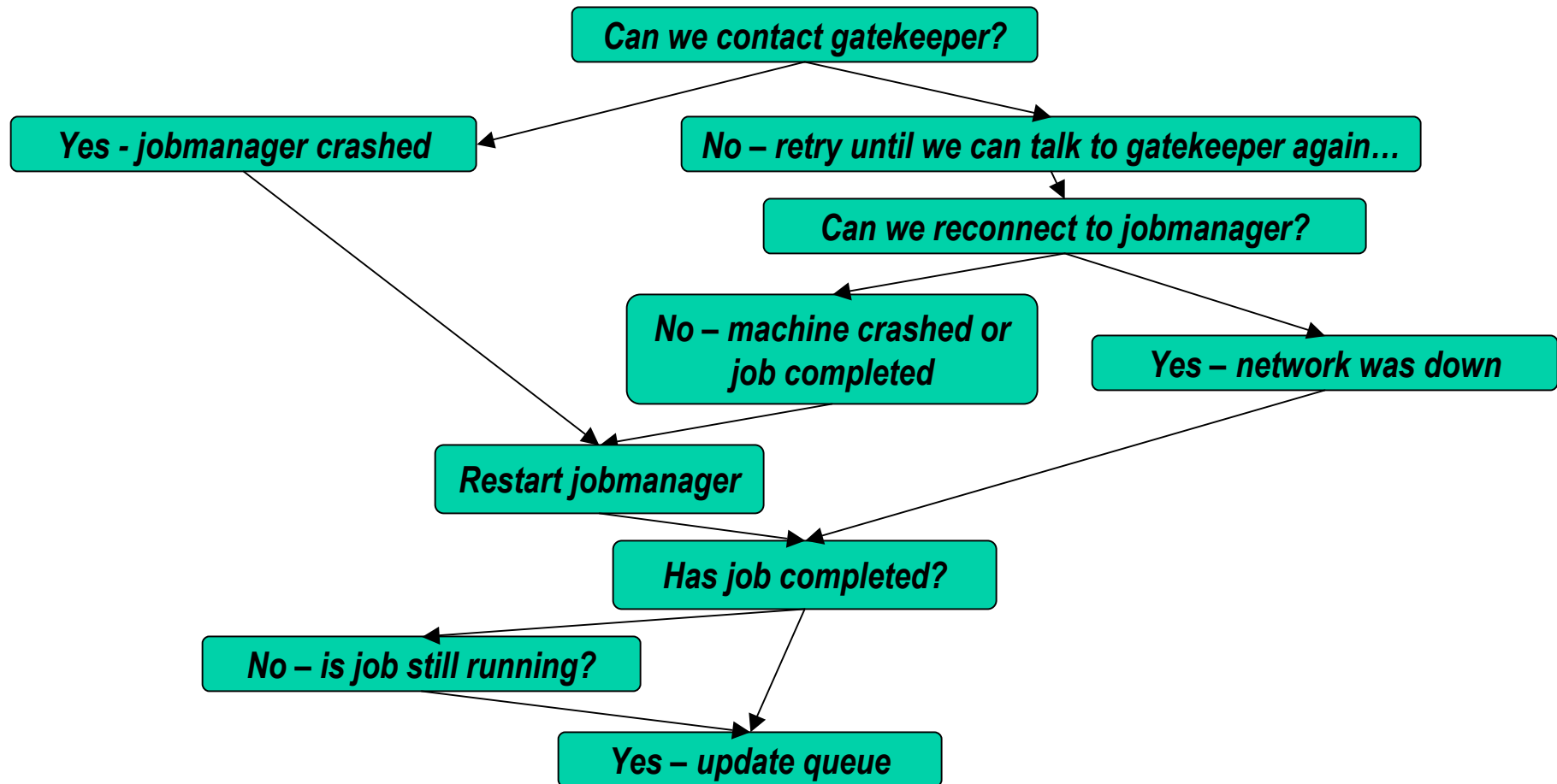
How It Works



Grid Universe Concerns

- What about Fault Tolerance?
 - Local Crashes
 - What if the submit machine goes down?
 - Network Outages
 - What if the connection to the remote Globus jobmanager is lost?
 - Remote Crashes
 - What if the remote Globus jobmanager crashes?
 - What if the remote machine goes down?
 - Condor-G's persistent job queue lets it recover from all of these failures
 - If a JobManager fails to respond...
-

Globus Universe Fault-Tolerance: Lost Contact with Remote Jobmanager



Back to our submit file...

- Many options can go into the submit description file.

```
universe      = grid
grid_resource = gt2 beak.cs.wisc.edu/jobmanager-pbs
executable    = progname
log = some-file-name.txt
queue
```

A Job's story: The “User Log” file

- A UserLog must be specified in your submit file:
 - Log = filename
- You get a log entry for everything that happens to your job:
 - When it was submitted to Condor-G, when it was submitted to the remote Globus jobmanager, when it starts executing, completes, if there are any problems, etc.
- Very useful! Highly recommended!

Sample Condor User Log

000 (8135.000.000) 05/25 19:10:03 Job submitted from host: <128.105.146.14:1816>

...

001 (8135.000.000) 05/25 19:12:17 Job executing on host: <128.105.165.131:1026>

...

005 (8135.000.000) 05/25 19:13:06 Job terminated.

(1) Normal termination (return value 0)

Usr 0 00:00:37, Sys 0 00:00:00 - Run Remote Usage

Usr 0 00:00:00, Sys 0 00:00:05 - Run Local Usage

Usr 0 00:00:37, Sys 0 00:00:00 - Total Remote Usage

Usr 0 00:00:00, Sys 0 00:00:05 - Total Local Usage

9624 - Run Bytes Sent By Job

7146159 - Run Bytes Received By Job

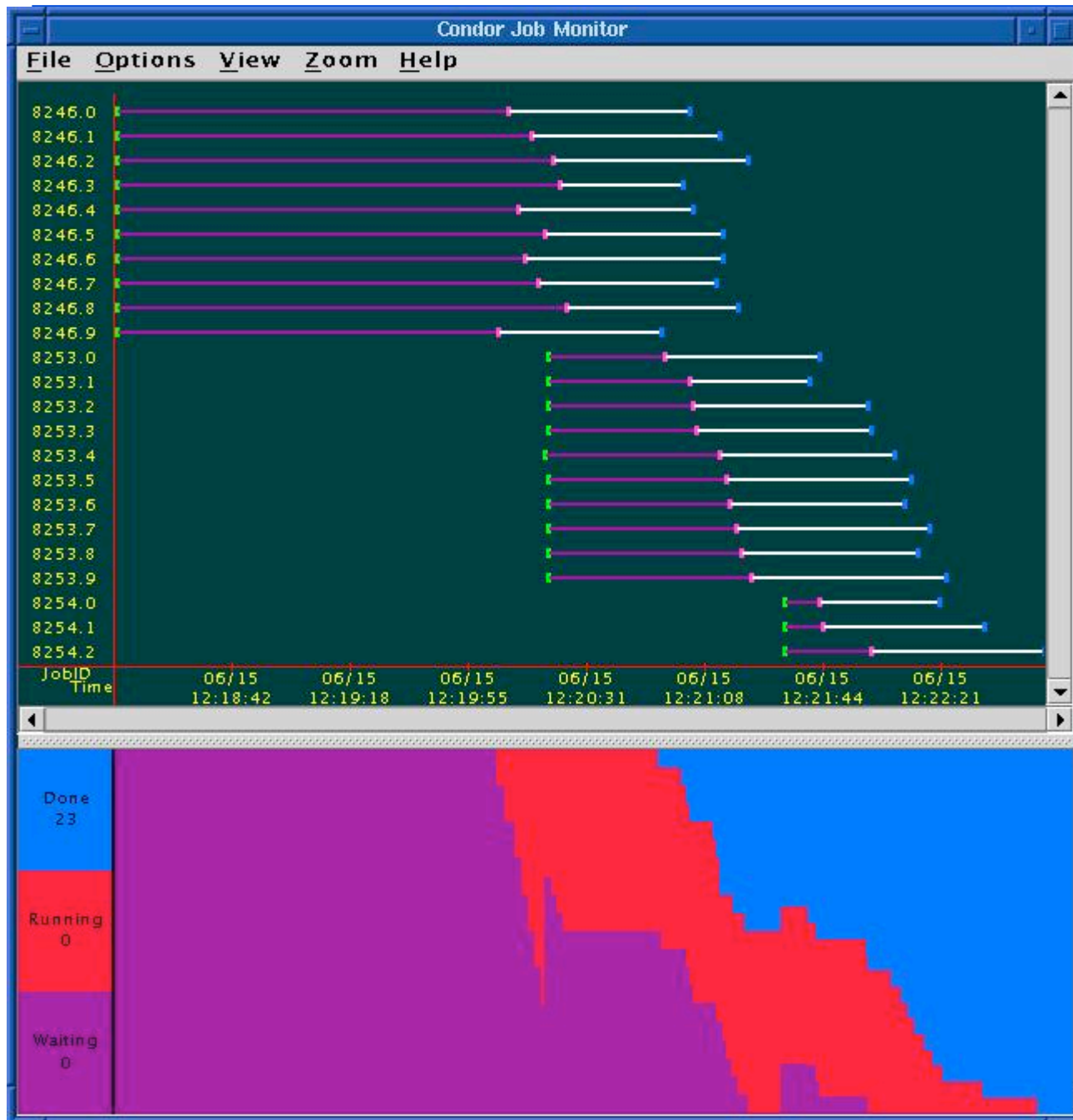
9624 - Total Bytes Sent By Job

7146159 - Total Bytes Received By Job

...

Uses for the User Log

- Easily read by human or machine
 - C++ library and Perl Module for parsing UserLogs is available
- Event triggers for meta-schedulers
 - Like DAGMan...
- Visualizations of job progress
 - Condor-G JobMonitor Viewer



Condor-G
JobMonitor
Screenshot

Want other Scheduling possibilities?

Use the Scheduler Universe

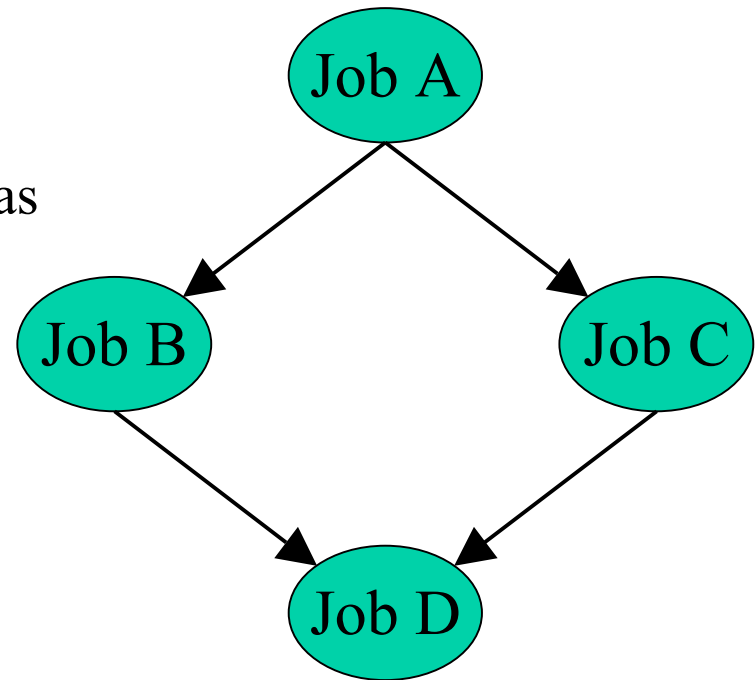
- In addition to Globus, another job universe is the *Scheduler Universe*.
- Scheduler Universe jobs run on the submitting machine.
- Can serve as a meta-scheduler.
- DAGMan meta-scheduler included

DAGMan

- **Directed Acyclic Graph Manager**
- DAGMan allows you to specify the *dependencies* between your Condor-G jobs, so it can *manage* them automatically for you.
- (e.g., “Don’t run job “B” until job “A” has completed successfully.”)

What is a DAG?

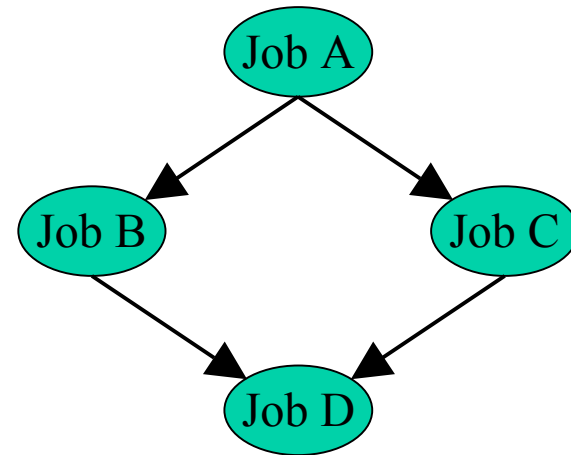
- A DAG is the **data structure** used by DAGMan to represent these dependencies.
- Each job is a **“node”** in the DAG.
- Each node can have any number of “parent” or “children” nodes – as long as there are **no loops**!



Defining a DAG

- A DAG is defined by a *.dag file*, listing each of its nodes and their dependencies:

```
# diamond.dag
Job A a.sub
Job B b.sub
Job C c.sub
Job D d.sub
Parent A Child B C
Parent B C Child D
```



- each node will run the Condor-G job specified by its accompanying *Condor submit file*

Submitting a DAG

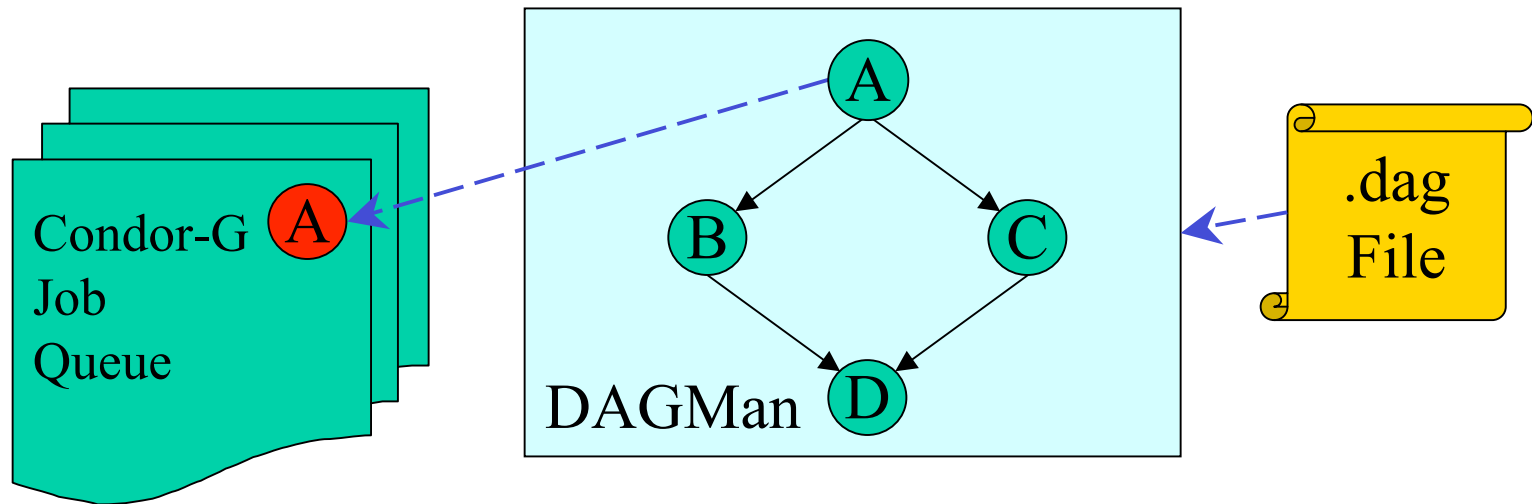
- To start your DAG, just run **condor_submit_dag** with your .dag file, and Condor will start a personal DAGMan daemon which to begin running your jobs:

```
% condor_submit_dag diamond.dag
```

- `condor_submit_dag` submits a Scheduler Universe Job with DAGMan as the executable.
- Thus the DAGMan daemon itself runs as a Condor-G scheduler universe job, so you don't have to baby-sit it.

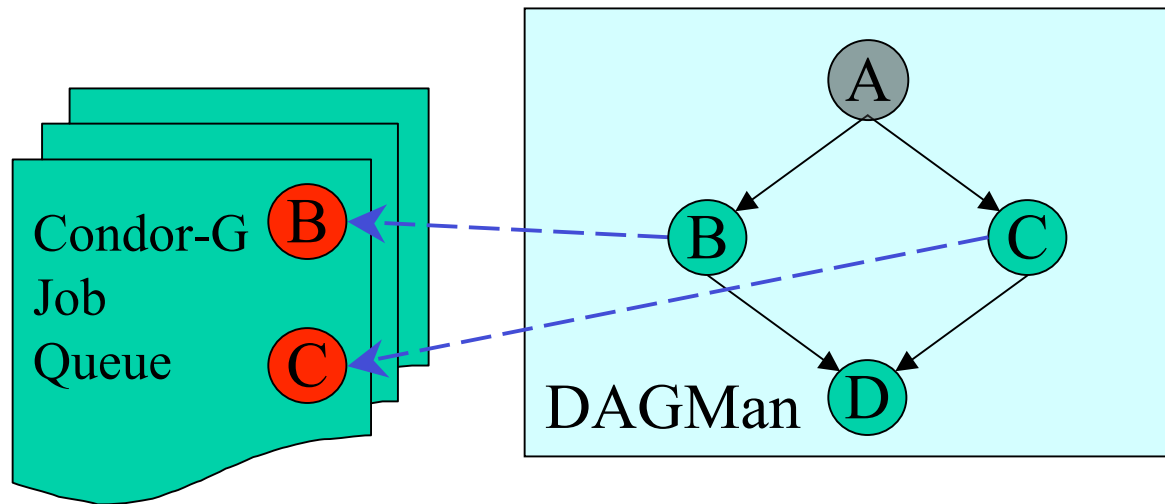
Running a DAG

- DAGMan acts as a “meta-scheduler”, managing the submission of your jobs to Condor-G based on the DAG dependencies.



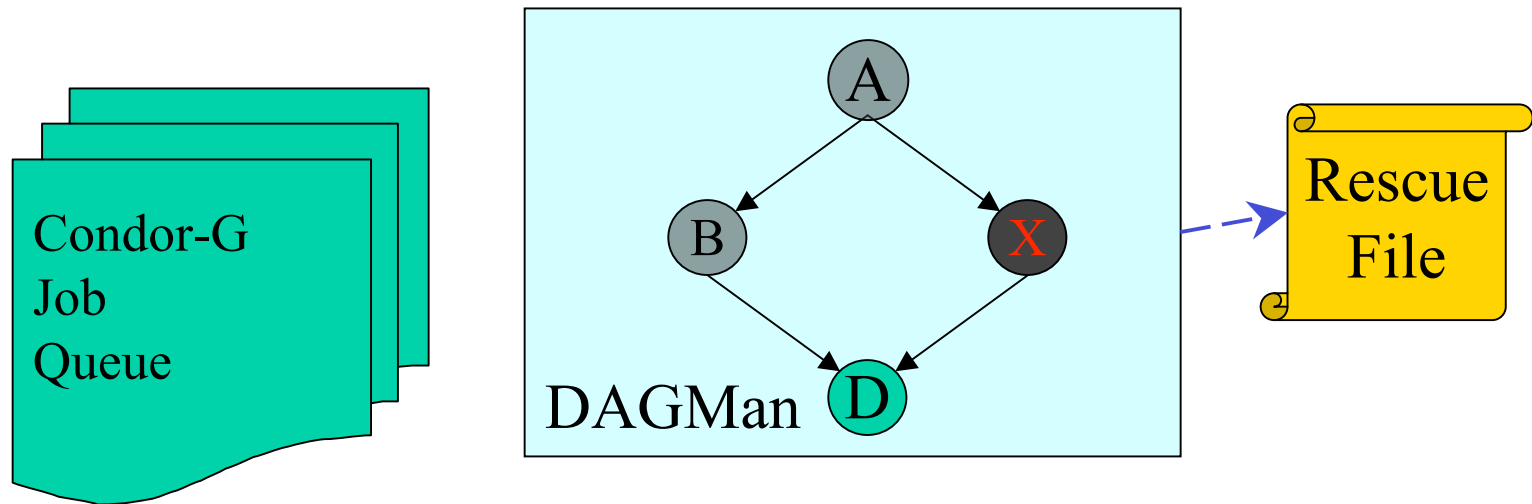
Running a DAG (cont'd)

- DAGMan holds & submits jobs to the Condor-G queue at the appropriate times.



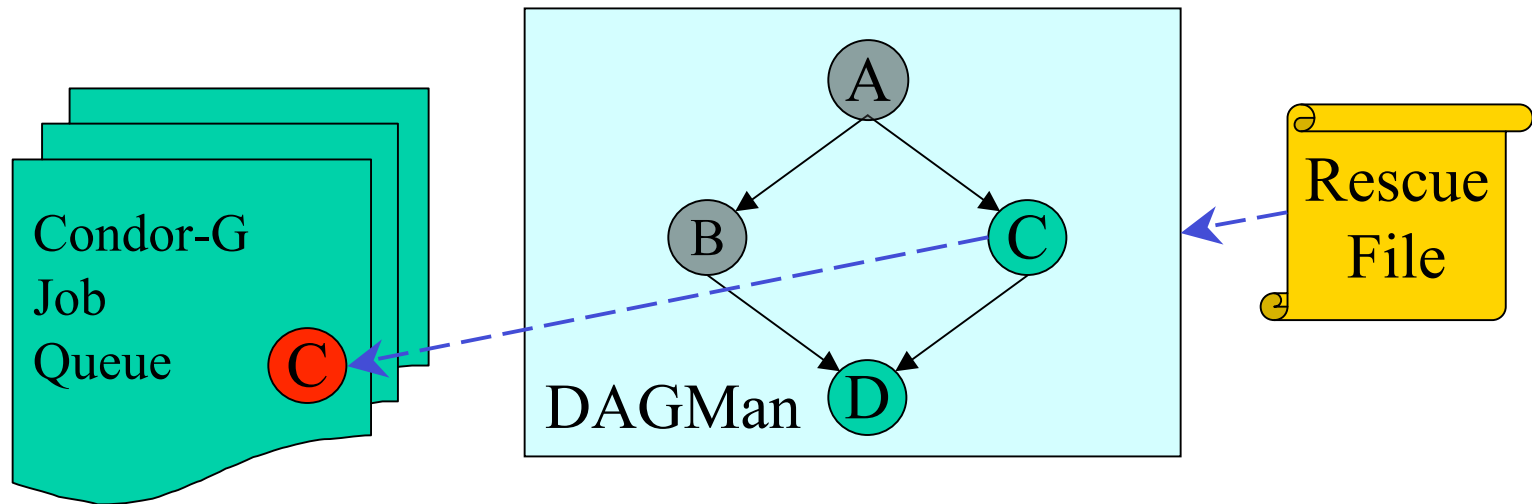
Running a DAG (cont'd)

- In case of a job failure, DAGMan continues until it can no longer make progress, and then creates a *“rescue” file* with the current state of the DAG.



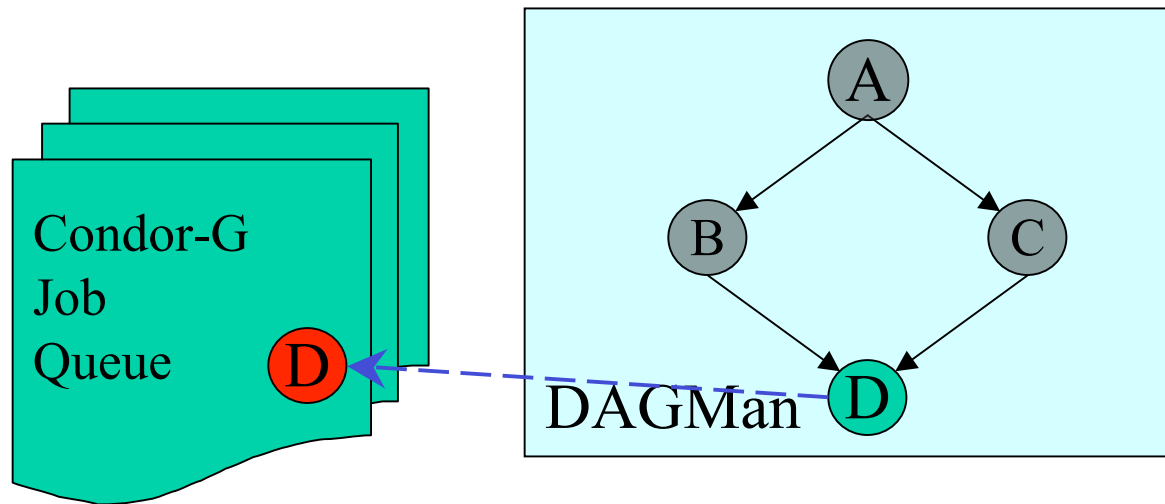
Recovering a DAG

- Once the failed job is ready to be re-run, the rescue file can be used to restore the prior state of the DAG.



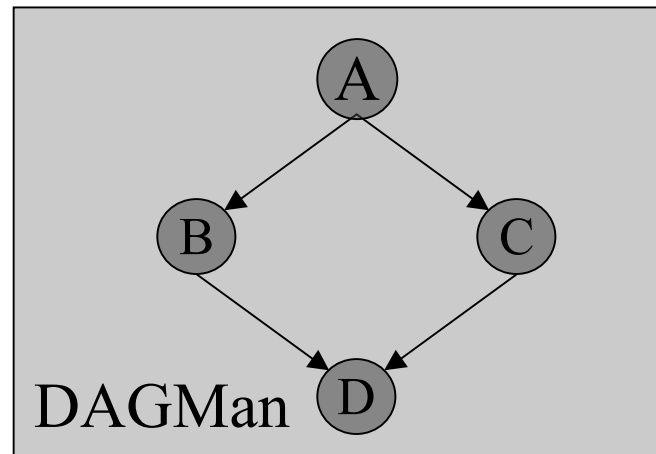
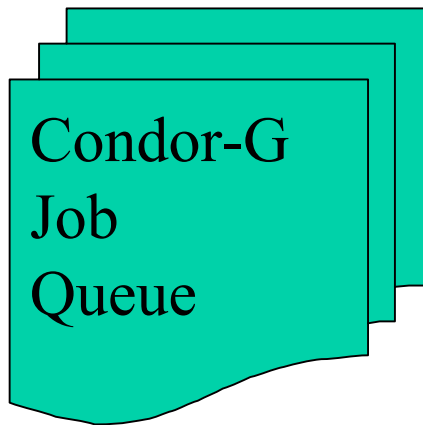
Recovering a DAG (cont'd)

- Once that job completes, DAGMan will continue the DAG as if the failure never happened.



Finishing a DAG

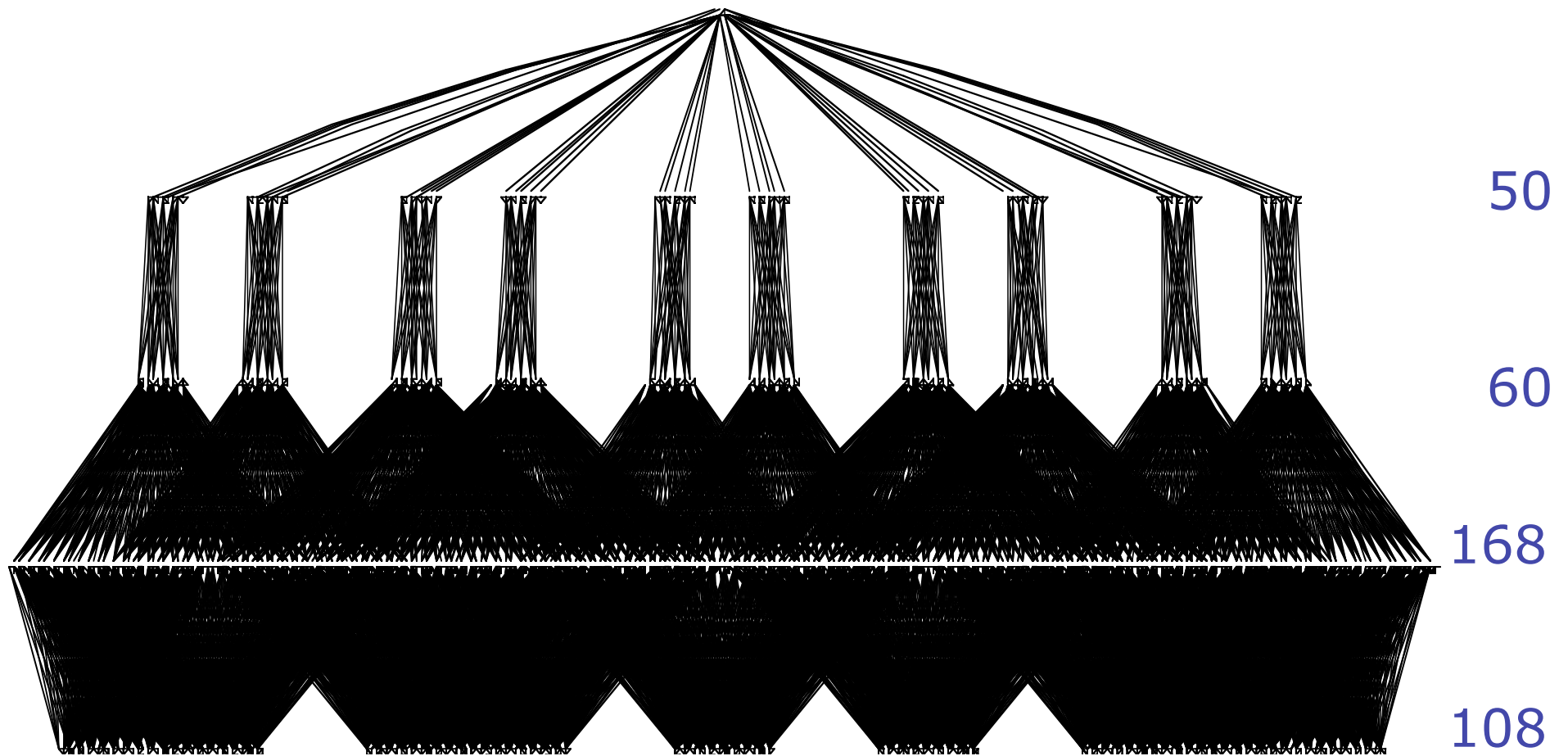
- Once the DAG is complete, the DAGMan job itself is finished, and exits.



Additional DAGMan Features

- Provides other handy features for job management...
 - ❑ nodes can have **PRE** & **POST** scripts
 - ❑ failed nodes can be automatically re-tried a configurable number of times
 - ❑ job submission can be “throttled”
 - ❑ reliable data placement

Here is a real-world workflow:
744 Files, 387 Nodes



This presentation based on: Grid Resources and Job Management



Jaime Frey
Condor Project,
University of Wisconsin-Madison
jfrey@cs.wisc.edu