Grid Compute Resources and Job Management

Open Science Grid

Grid middleware - "glues" all pieces together

- Offers services that couple users with remote resources through resource brokers
- Remote process management
- Co-allocation of resources
- Storage access
- Information
- Security
- QoS

Terms:

- Globus
- GRAM
- Condor
- Condor-G

Local Resource Managers (LRM)

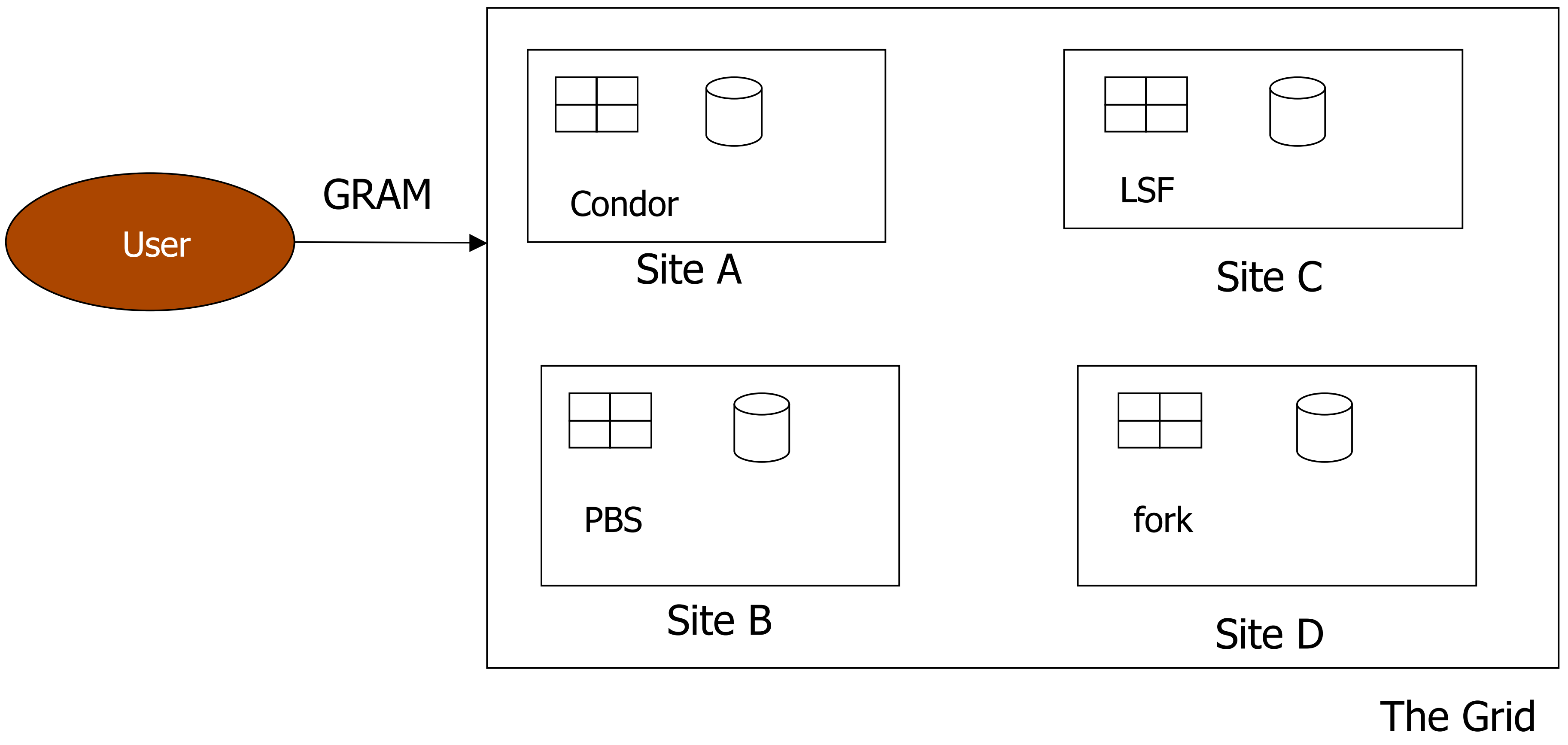
- Compute resources have a **local resource manager** (LRM) that controls:
  - Who is allowed to run jobs
  - How jobs run on a specific resource
  - Specifies the order and location of jobs
- *Example policy:*
  - Each cluster node can run one job.
  - If there are more jobs, then they must wait in a queue
- *Examples:* PBS, LSF, Condor

GRAM
Globus Resource Allocation Manager

- **GRAM** = provides a <u>standardised interface</u> to submit jobs to LRMs.
- Clients submit a job request to GRAM
- GRAM translates into something a(ny) LRM can understand

  …. Same job request can be used for many different kinds of LRM

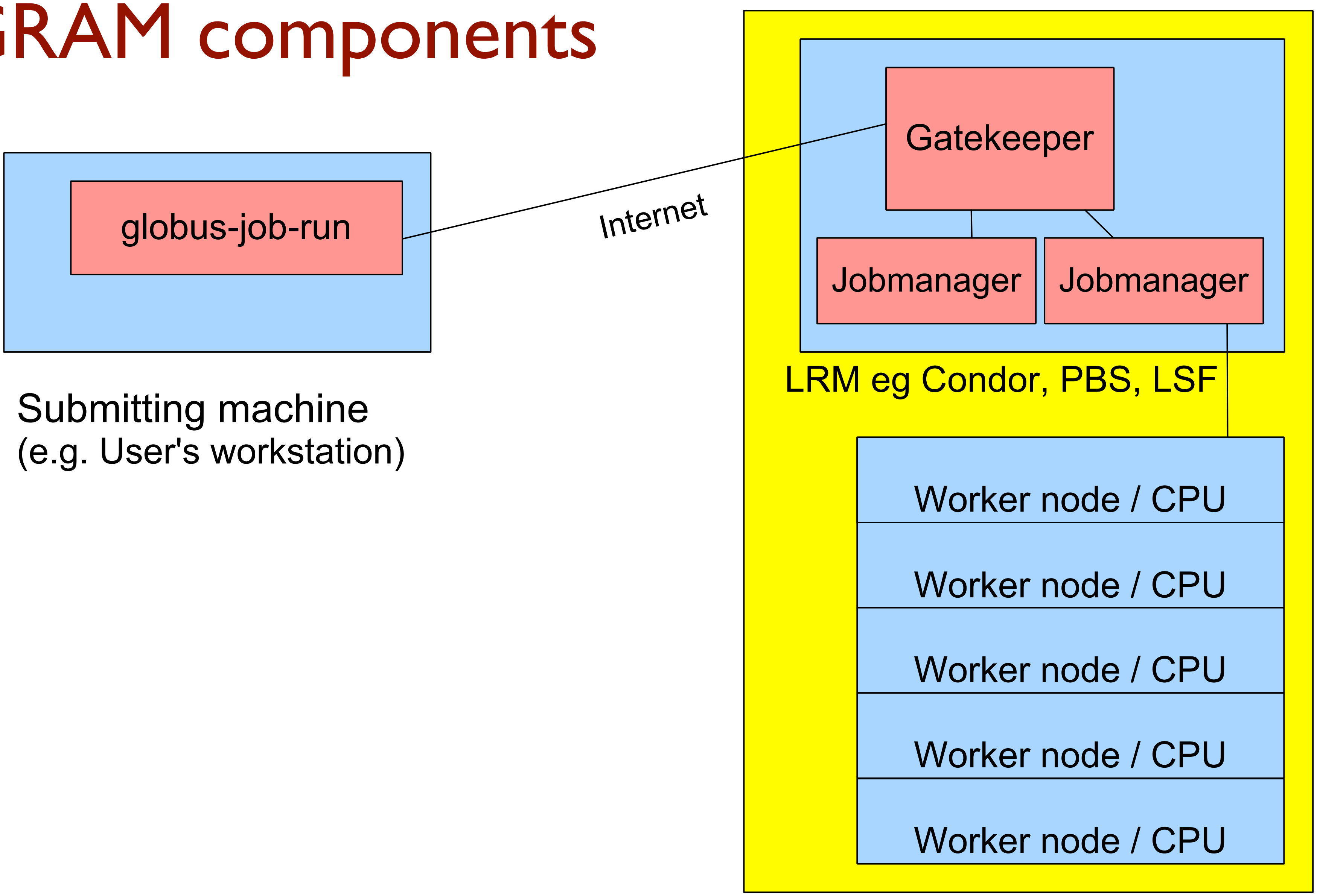
Job Management on a Grid



GRAM's abilities

- Given a job specification:
  - Creates an environment for the job
  - Stages files to and from the environment
  - Submits a job to a local resource manager
  - Monitors a job
  - Sends notifications of the job state change
  - Streams a job's stdout/err during execution

GRAM components



Submitting a job with GRAM

- **globus-job-run** command

```
$ globus-job-run grid07.uchicago.edu /bin/hostname
```

  - Run '/bin/hostname' on the resource grid07.uchicago.edu

- We don't care what LRM is used on ‘grid07’.
- This command works with any LRM.

Condor

- Condor is a specialized workload management system for compute-intensive jobs.

- is a software system that creates an HTC environment (created at UW-Madison)
  - Detects machine availability
  - Harnesses available resources
  - Provides powerful resource management by ***matching*** resource owners with consumers (broker)

How Condor works

Condor provides:

- a job queueing mechanism
- scheduling policy
- priority scheme
- resource monitoring, and
- resource management.

Users **submit** their serial or parallel jobs to Condor,

Condor places them into a **queue**,

… chooses **when** and **where** to run the jobs based upon a policy,

… carefully **monitors** their progress, and

… ultimately **informs** the user upon completion.

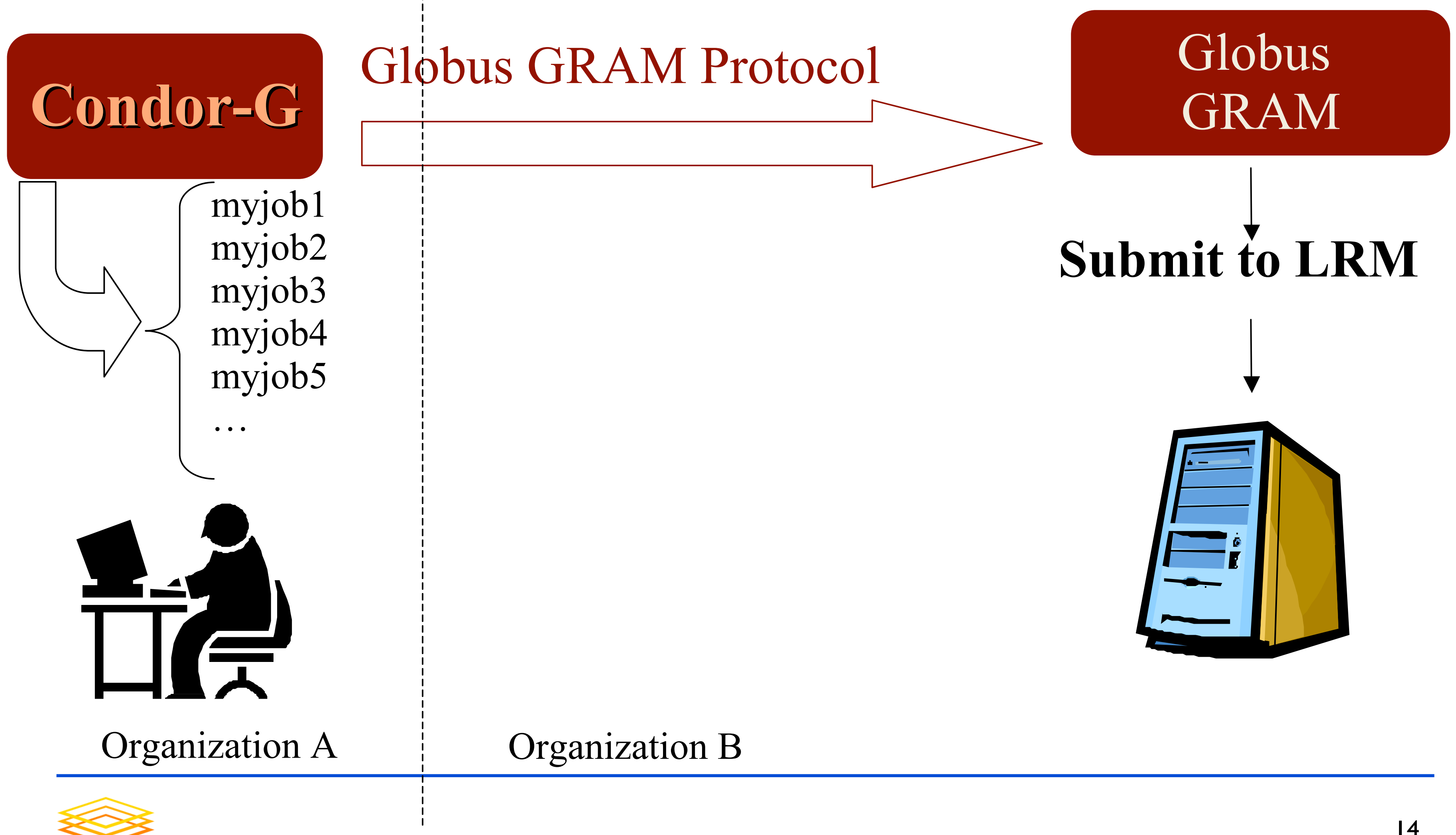
Condor lets you manage a large number of jobs.

- Specify the jobs in a file and submit them to Condor
- Condor runs them and keeps you notified on their progress
  - Mechanisms to help you manage huge numbers of jobs (1000's), all the data, etc.
  - Handles inter-job dependencies (DAGMan)
- Users can set Condor's job priorities
- Condor administrators can set user priorities
- Can do this as:
  - Local resource manager (LRM) on a compute resource
  - Grid client submitting to GRAM (as Condor-G)

Condor-G

- is the job management part of Condor.
  - *Hint:* Install Condor-G to submit to resources accessible through a Globus interface.
- Condor-G does not ***create*** a grid service.
- It only deals with ***using*** remote grid services.

Remote Resource Access: Condor-G + Globus + Condor



Open Science Grid

Four Steps to Run a Job with Condor

- These choices tell Condor
  - **how**
  - **when**
  - **where** to run the job,
  - and describe exactly **what** you want to run.

- Choose a Universe for your job
- Make your job batch-ready
- Create a *submit description* file
- Run *condor_submit*

Simple Submit Description File

```
# myjob.submit 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   = grid
Executable = analysis
Log        = my_job.log
Queue
```

Open Science Grid

Run condor_submit

- You give *condor_submit* the name of the submit file you have created:

```
condor_submit my_job.submit
```

- *condor_submit* parses the submit file

Another Submit Description File

```
# Example condor_submit input file

Universe    = grid
Executable = /home/wright/condor/my_job.condor
Input       = my_job.stdin
Output      = my_job.stdout
Error       = my_job.stderr
Arguments   = -arg1 -arg2
InitialDir = /home/wright/condor/run_1
Queue
```

Open Science Grid

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

Submitting more complex jobs

- express dependencies between jobs
  ⇒ WORKFLOWS
- And also, we would like the workflow to be managed even in the face of failures

Want other Scheduling possibilities? Use the Scheduler Universe

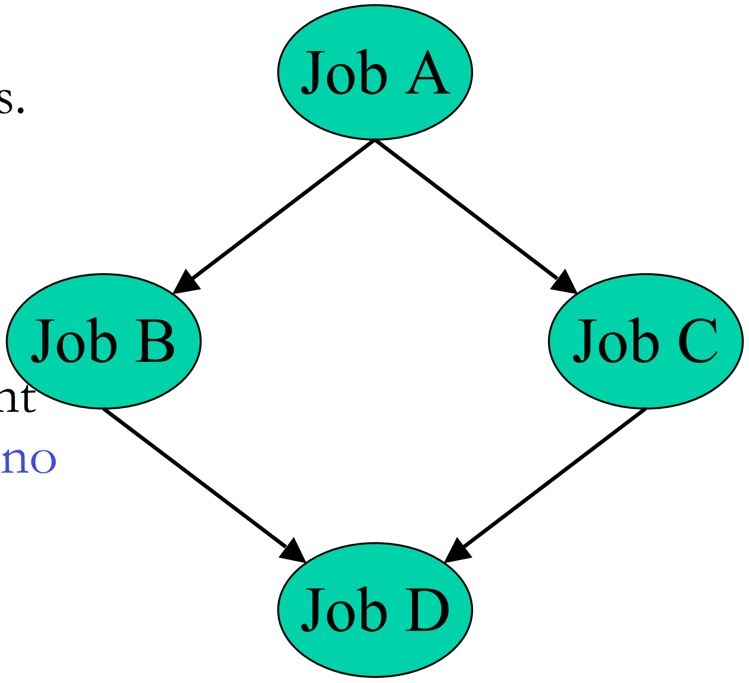
- In addition to VANILLA, another job universe is the *Scheduler Universe*.
- Scheduler Universe jobs run on the submitting machine and serve as a meta-scheduler.
- **Condor's Scheduler Universe lets you set up and manage job workflows.**
- DAGMan meta-scheduler included
  - DAGMan manages these jobs

DAGMan

- **<u>D</u>irected <u>A</u>cyclic <u>G</u>raph <u>Man</u>ager**
- DAGMan allows you to specify the *dependencies* between your Condor 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?

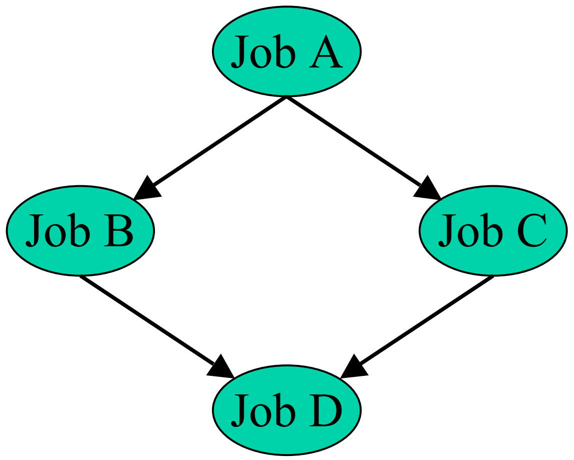
- A DAG is the data structure used by DAGMan to represent these dependencies.
- Each job is a "node" in the DAG.
- Each node can have any number of "parent" or "children" nodes – as long as there are no loops!



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 job specified by its accompanying Condor submit file

Open Science Grid

Submitting a DAG

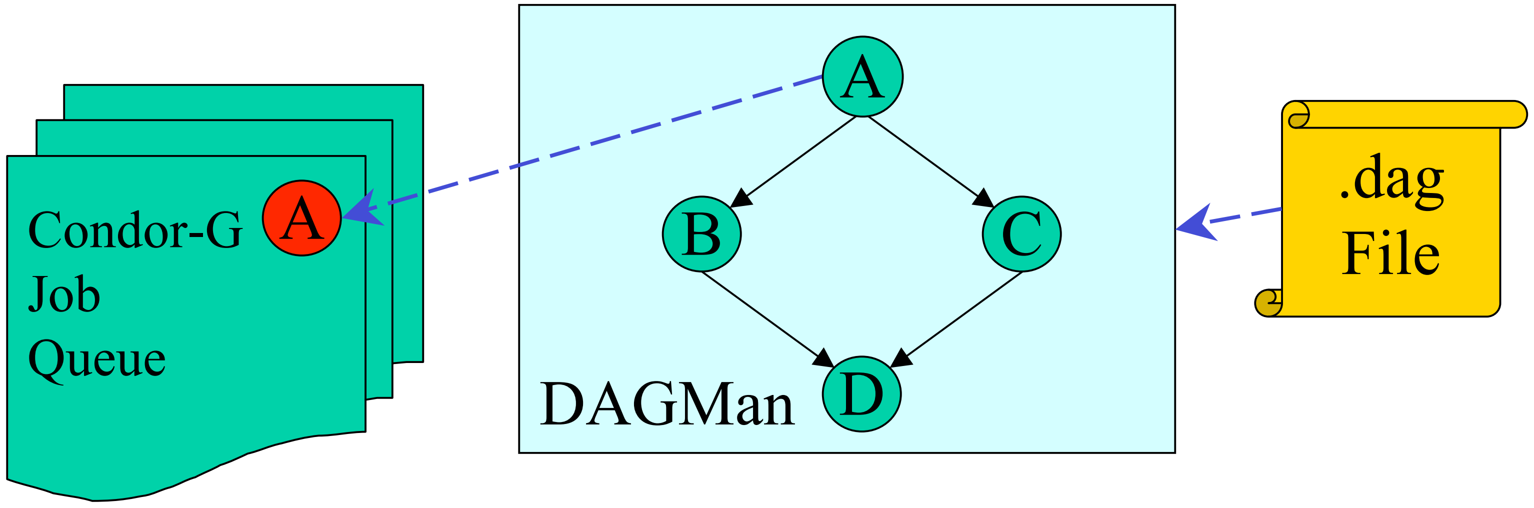
- To start your DAG, just run ***condor_submit_dag*** with your .dag file, and Condor will start a personal DAGMan daemon which to begin running your jobs:

```
% 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 job, so you don't have to baby-sit it.

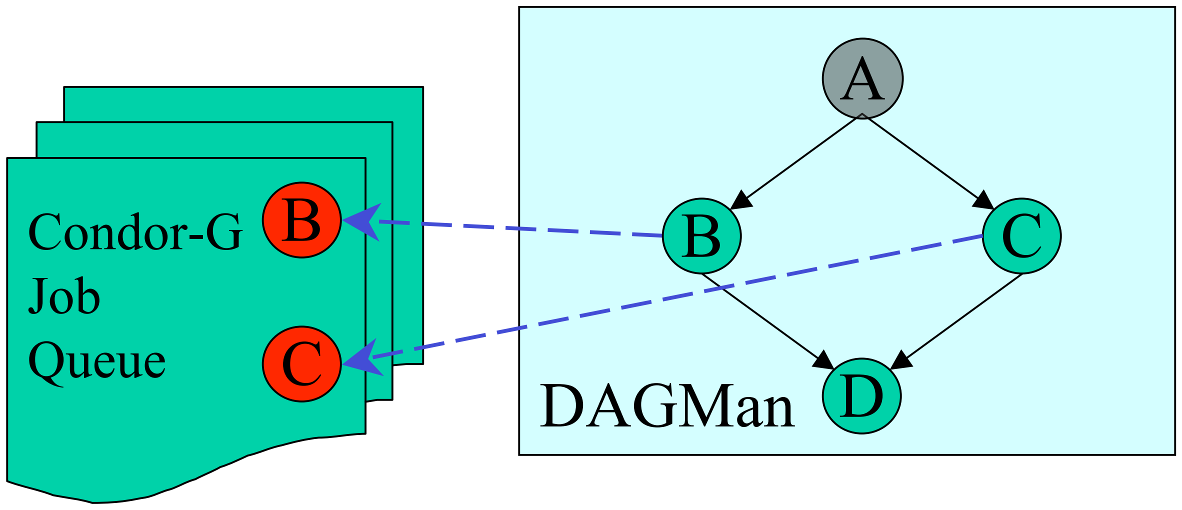
Running a DAG

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



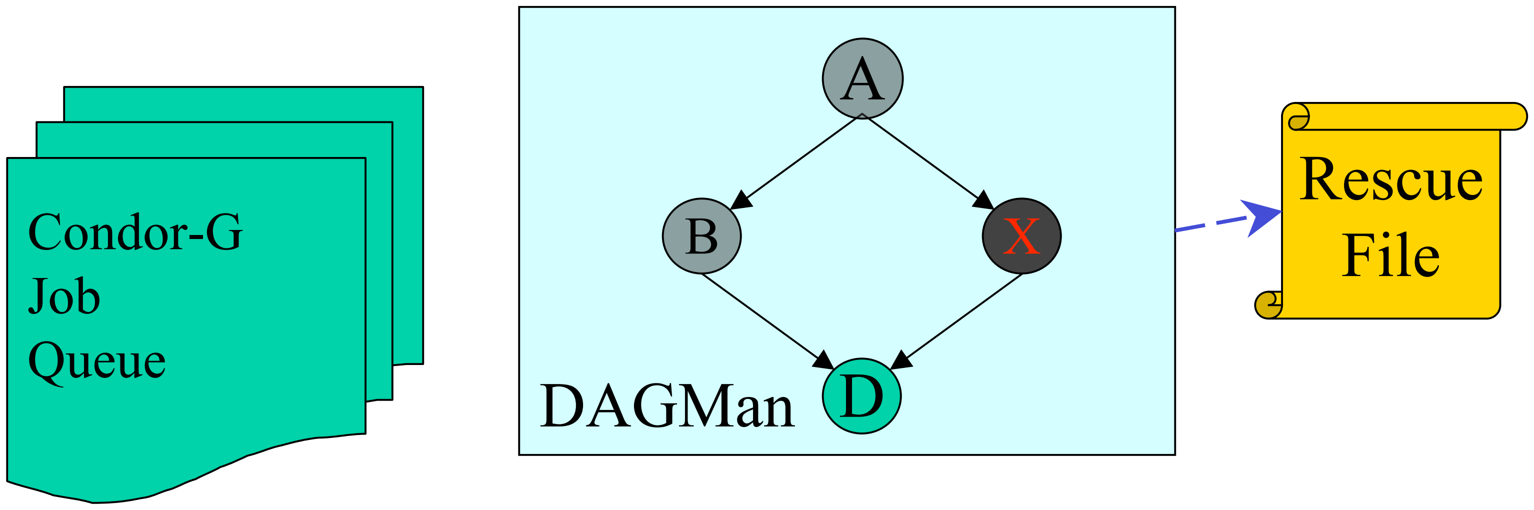
Running a DAG (cont'd)

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



Running a DAG (cont'd)

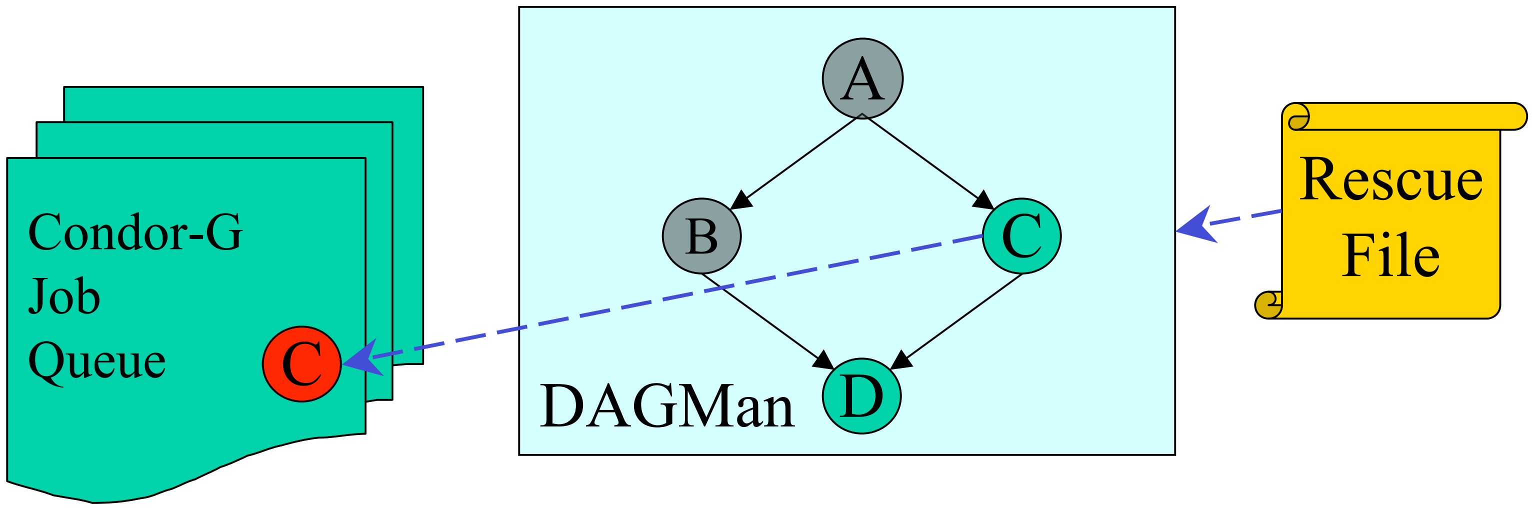
- In case of a job failure, DAGMan continues until it can no longer make progress, and then creates a *"rescue" file* with the current state of the DAG.



Open Science Grid

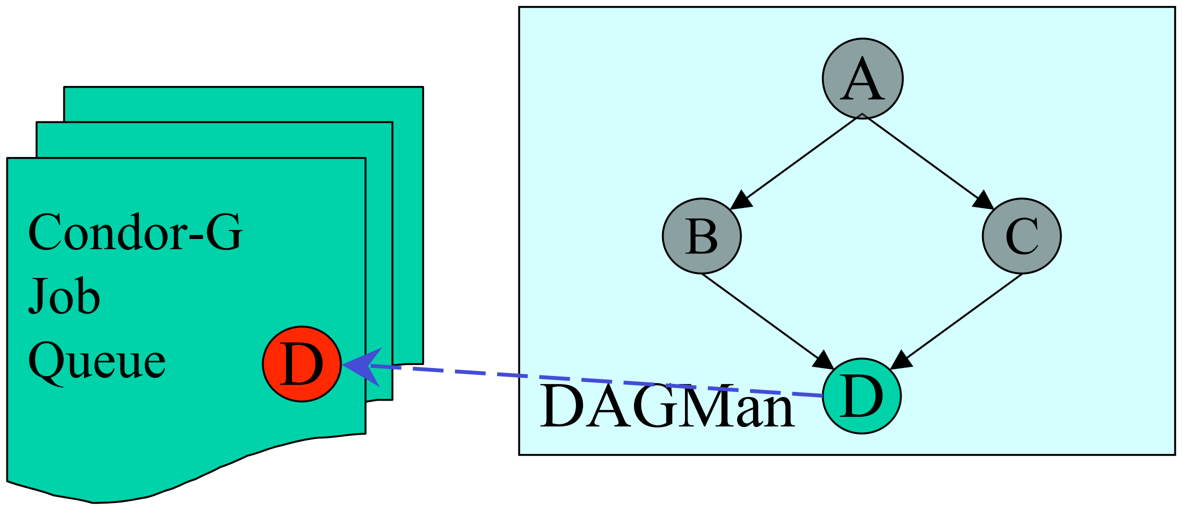
Recovering a DAG -- fault tolerance

- 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)

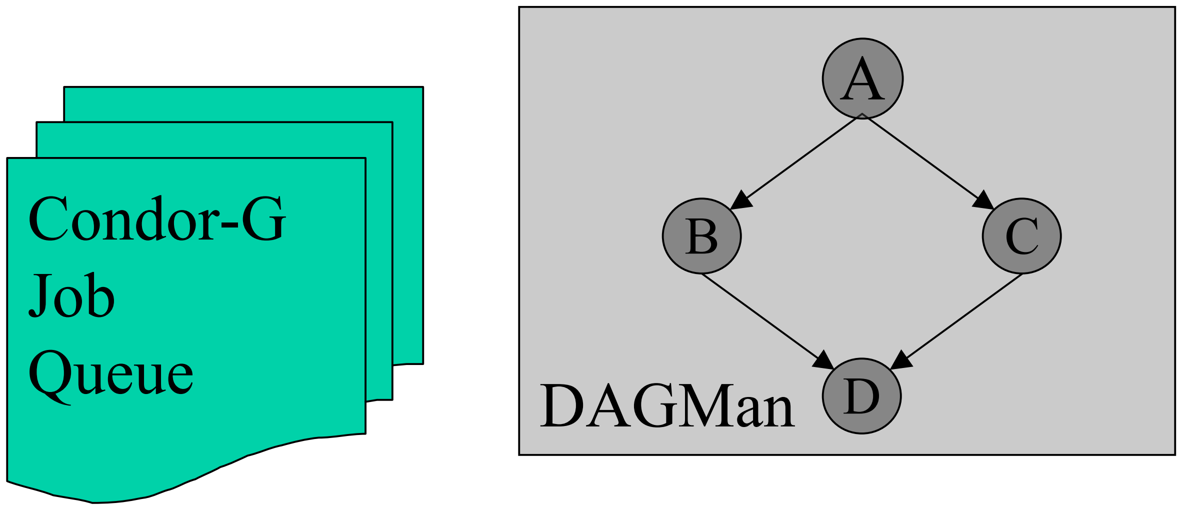
- Once that job completes, DAGMan will continue the DAG as if the failure never happened.



Open Science Grid

Finishing a DAG

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



We have seen how Condor:

… monitors submitted jobs and reports progress

… implements your policy on the execution order of the jobs

… keeps a log of your job activities

…. Now go to the Lab part ….

Acknowledgments:
This presentation based on:
Grid Resources and Job Management

Open Science Grid

Jaime Frey and Becky Gietzel

Condor Project

U. Wisconsin-Madison