Grid Compute Resources and Job Management

Open Science Grid

New Mexico Grid School – April 9, 2009

Marco Mambelli – University of Chicago
marco@hep.uchicago.edu

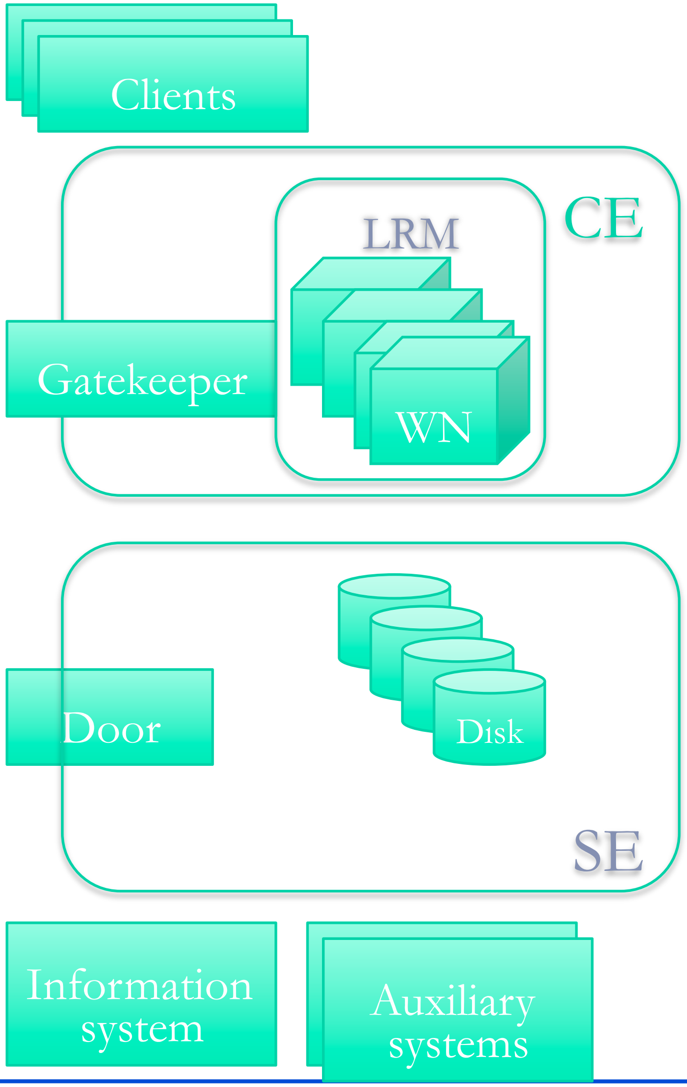
Outline

- Grid abstraction
- Clusters and Local Resource Managers
- GRAM, the Grid protocol
- Adding some Security
- Condor
- Something more with Condor
- DAGman
- OSG abstractions

Grid/OSG components

- Standard installation
- Set of services or resources containing data and providing processing power
- Computing elements (CE)
- Storage elements (SE)
- Information systems
- Clients

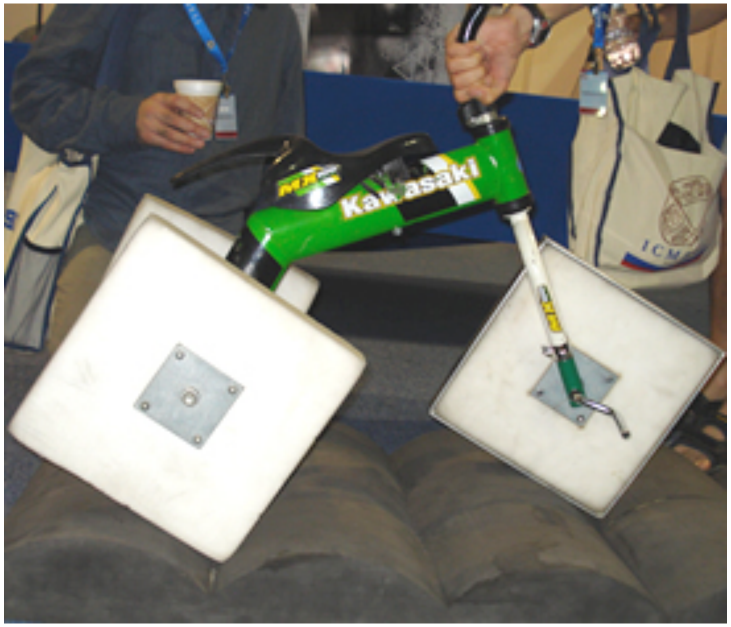
- How to interact?
  - validation and testing
  - active use



Grid as Abstraction

- Hides lower level components/fabrics
  - you can use it without knowing what is behind
- Provides leverage
  - do so much with so little
- Limit flexibility
- Suggest usage patterns
- Gain/loss

Do not reinvent the wheel!



Job management layers

- High level user tools (Panda, Swift, Pegasus, …)
- OSG abstractions
- DAGman
- Condor
- Adding some Security
- GRAM, the Grid protocol
- Clusters and Local Resource Managers

The power of leverage

Good use of the Grid Abstraction

- Leaks are likely
  - all abstraction “lie”
  - some leaks are desired
- Before you can abstract you must see the details
- Understanding the levels below you understand the abstraction
- To build components
  - must be fully aware of at least a couple of layers below
  - to EXCELL you must be VERY familiar with several layers above

Job management layers

- High level user tools (Panda, Swift, Pegasus, …)
- OSG abstractions
- DAGman
- Condor
- Adding some Security
- GRAM, the Grid protocol
- Clusters and Local Resource Managers

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
- *Example policy:*
  - Each cluster node can run one job.
  - If there are more jobs, then they must wait in a queue
- LRMs allow nodes in a cluster can be **reserved** for a specific person
- *Examples:* PBS, LSF, Condor

Job management layers

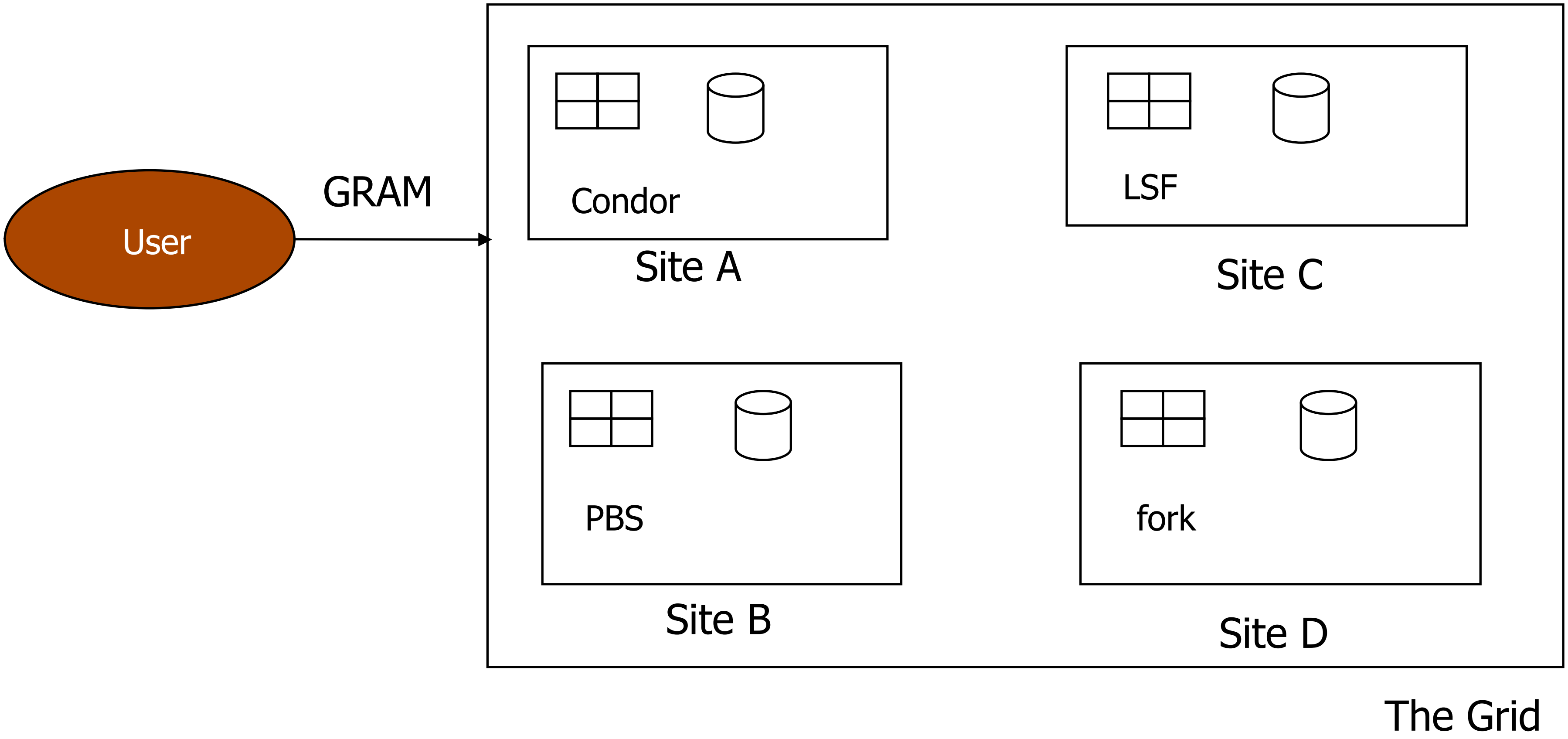
- High level user tools (Panda, Swift, Pegasus, …)
- OSG abstractions
- DAGman
- Condor
- Adding some Security
- GRAM, the Grid protocol
- Clusters and Local Resource Managers

GRAM
Globus Resource Allocation Manager

- **GRAM** = provides a standardised interface 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



Two versions of GRAM

- There are two versions of GRAM
    - GT2
        - Own protocols
        - Older
        - More widely used
        - No longer actively developed
    - GT4
        - Web services
        - Newer
        - New features go into GRAM4

- In this module, will be using GT2

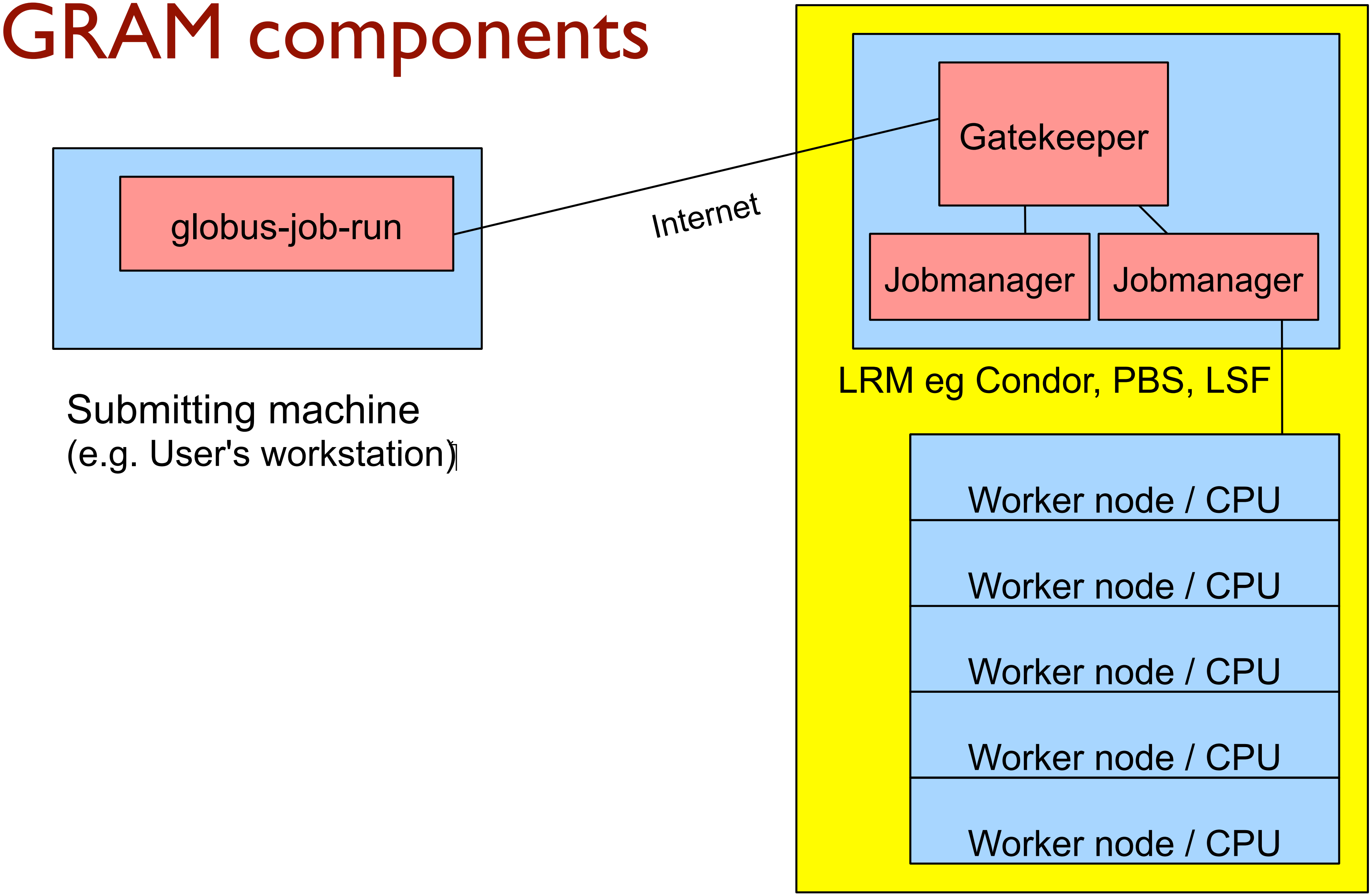
GRAM's abilities

- 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

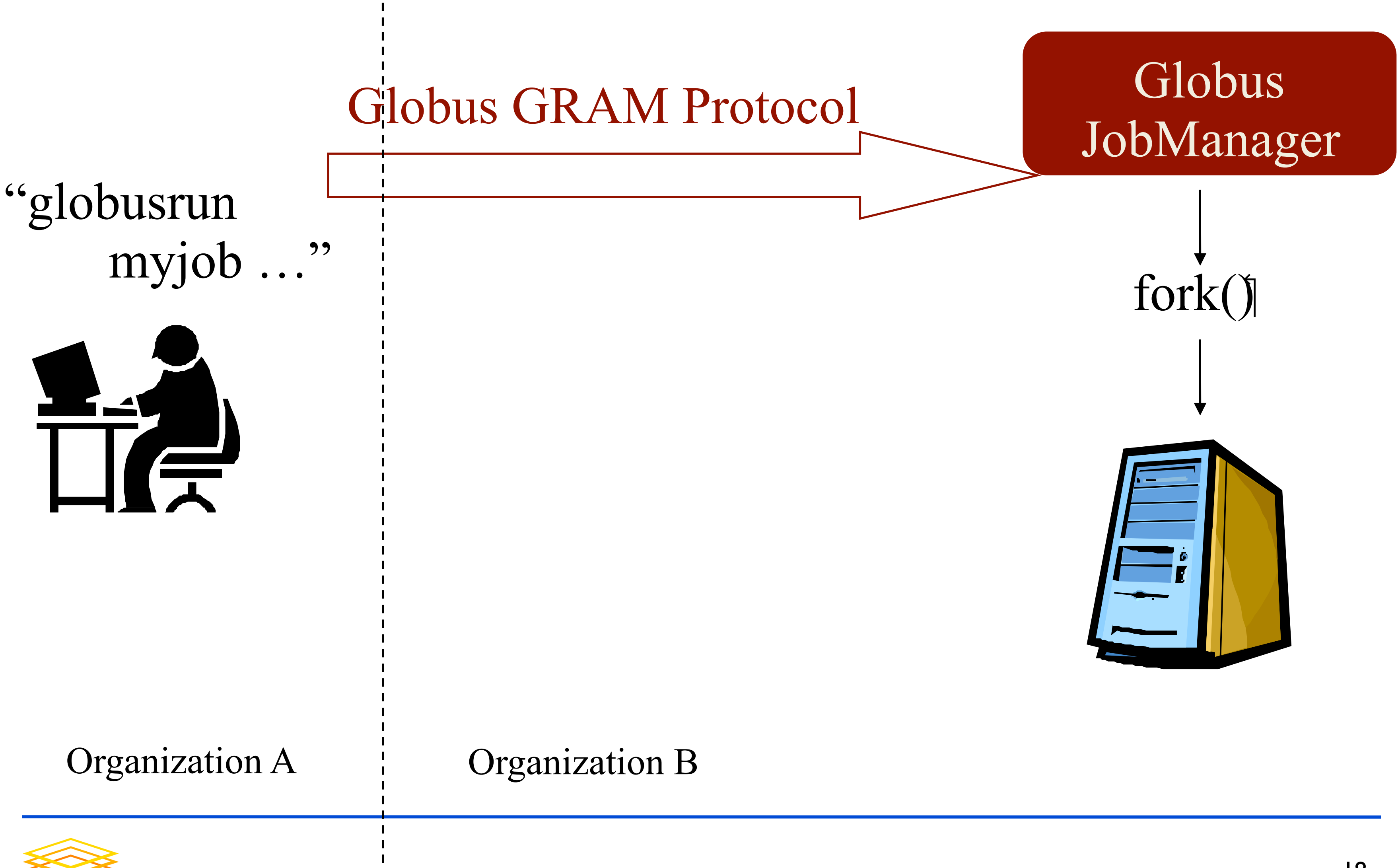
GRAM components

- Clients –
  - eg. globus-job-submit, globus-run
- 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



Remote Resource Access: Globus



“globusrun
myjob …”

Globus GRAM Protocol

Globus JobManager

fork()

Organization A

Organization B

Job management layers

- High level user tools (Panda, Swift, Pegasus, …)
- OSG abstractions
- DAGman
- Condor
- Adding some Security
- GRAM, the Grid protocol
- Clusters and Local Resource Managers

Security on the Grid

- GRAM is using **GSI** (Grid Security Infrastructure) to **authenticate** users and servers/services using x509 certificates
- An extended certificate identifies the users, their roles and the groups they belong to and it is the basis for the **authorization**
- Mechanisms are in place to guarantee **integrity** of the messages
- Messages can be encrypted to add **privacy**

Certificates and Proxy (certificates)

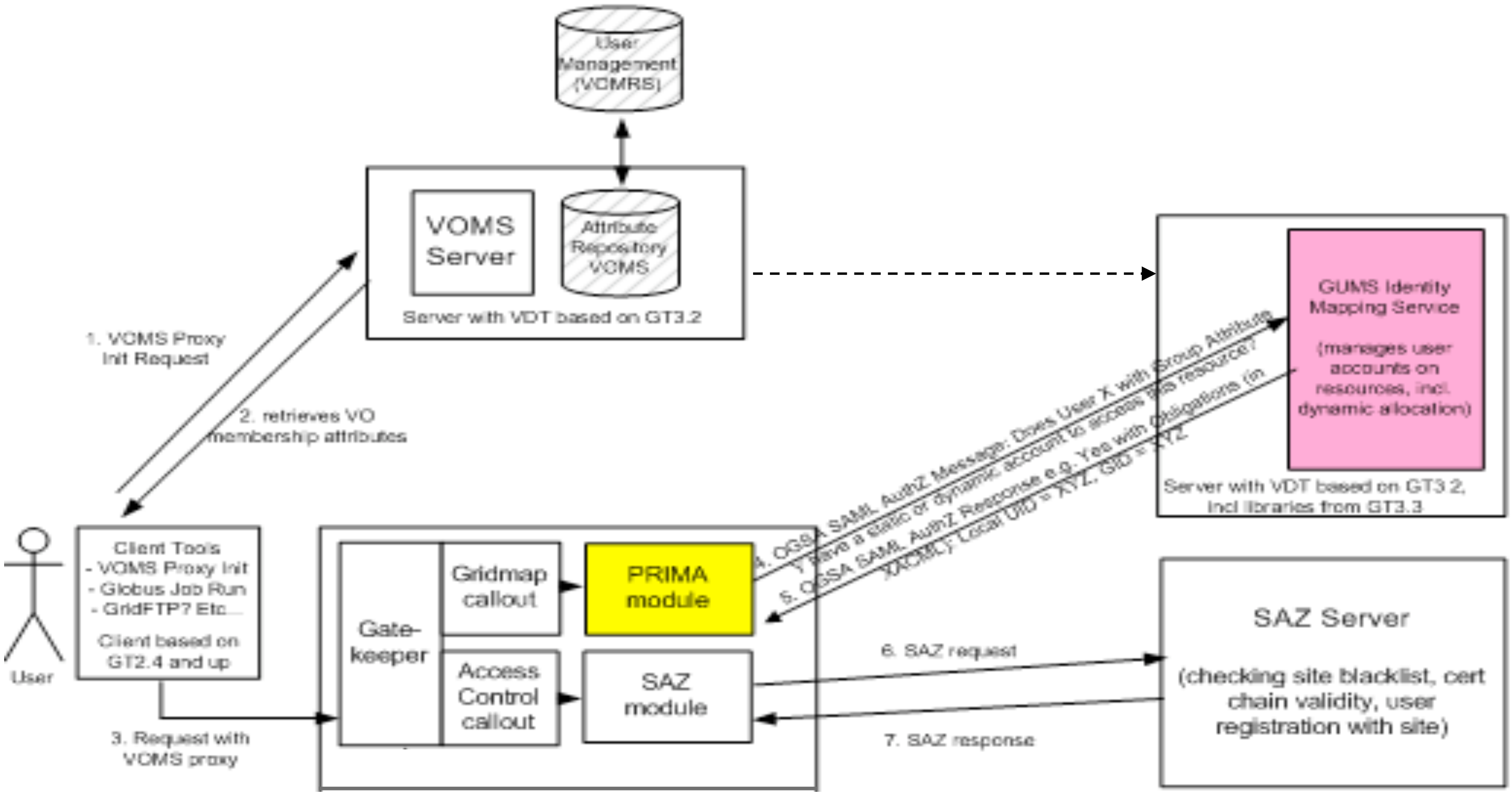
- Certificate content
  - public part
  - private key
- Proxy
  - temporary delegation
- Extended attributes (VOMS)
  - groups
  - roles

```
Certificate:    Data:        Version: 3 (0x2)        Serial
     Number: 923 (0x39b)        Signature Algorithm:
     md5WithRSAEncryption        Issuer: C=US, O=SDSC, OU=SDSC-
     CA, CN=Certificate Authority/UID=certman        Validity
     Not Before: Jun 22 00:46:02 2006 GMT            Not After :
     Jul  2 00:46:02 2006 GMT         Subject: C=US, O=SDSC,
     OU=SDSC, CN=Account Train99/UID=train99        Subject
     Public Key Info:            Public Key Algorithm:
     rsaEncryption           RSA Public Key: (2048 bit)
     Modulus (2048 bit):
     00:af:93:40:80:ce:14:68:d6:6c:67:89:45:0c:3e:
     30:98:38:35:c9:bd:b5:08:00:17:4c:e1:fb:38:50:
     bd:97:f5:41:92:e7:6e:c4:6f:dc:ad:52:2c:e0:2a:
     54:83:79:45:fb:5d:e2:f5:a5:cf:42:94:45:98:22:
     d9:5b:81:93:e2:46:5f:e0:7f:71:5f:2d:b0:4a:82:
     21:7d:f2:41:f7:b6:33:eb:59:93:f1:71:e3:79:ea:
     c0:1b:5e:07:c6:d5:c2:67:41:56:73:d8:1f:a3:fb:
     32:4b:f5:96:9f:65:f5:0a:f0:28:d5:90:d6:b0:dc:
     4b:29:85:aa:8b:b7:d5:c0:f3:45:28:f9:af:80:7a:
     88:40:40:21:60:ea:14:cd:8a:8e:53:40:67:c5:47:
     51:bc:95:76:1e:90:b0:ee:ee:41:5a:ec:d4:4c:3c:
     ea:eb:2f:f1:55:82:d8:b2:36:d9:92:88:bd:b6:93:
     eb:46:69:3b:3a:e2:15:54:82:c0:30:4b:a9:54:3c:
     af:52:4e:a5:71:40:a1:58:21:2e:ab:6d:c4:7c:59:
     5d:68:b6:95:80:0e:12:91:51:90:0e:38:84:3f:de:
     07:99:43:86:a1:0f:70:01:2f:3c:bf:e3:47:b2:16:
     67:eb:00:6b:c4:7d:d8:e5:39:77:ac:29:cc:76:94:
     2b:d3                Exponent: 65537 (0x10001)       X509v3
     extensions:            X509v3 Basic Constraints:
     CA:FALSE            Netscape Cert Type:                 SSL
     Client, S/MIME, Object Signing            Netscape Comment:
     OpenSSL Generated Certificate            Netscape CA
     Revocation Url:                 http://www.sdsc.edu/CA/
     SDSC_CRL.pem            X509v3 Subject Key Identifier:
     E1:E3:C9:6E:A6:CF:2C:FC:D3:B7:51:F6:03:66:98:C5:18:71:60:F8
     X509v3 Authority Key Identifier:
     keyid:BF:A3:87:2C:F6:0D:74:BD:48:6C:0E:27:BF:01:E4:F2:4F:
     46:BA:27                DirName:/C=US/O=SDSC/OU=SDSC-CA/
     CN=Certificate Authority/UID=certman                serial:
     00    Signature Algorithm: md5WithRSAEncryption
     93:b2:78:07:d9:72:e2:71:d7:66:83:0c:d3:97:0c:9e:24:33:
     4e:e3:48:28:9c:44:7e:31:13:70:cc:f8:4a:5d:bc:64:84:3e:
     aa:fa:da:86:3f:5e:f8:4a:72:a1:59:57:5a:89:49:5a:2d:c9:
     09:5c:a5:69:6e:65:f7:85:8b:07:57:f1:6a:cb:6e:e5:00:17:
     …
```

Virtual Organization Management System

- User management delegated to VO (Virtual Organization)
  - flexible
  - scalable
- Introduction of groups and roles
- Dynamic calls

VOMS Architecture


User Management (VOMRS)
VOMS Server
Attribute Repository VOMS
Server with VDT based on GT3.2
1. VOMS Proxy Init Request
2. retrieves VO membership attributes
GUMS Identity Mapping Service
(manages user accounts on resources, incl. dynamic allocation)
Server with VDT based on GT3.2, incl libraries from GT3.3
4. OGSA SAML AuthZ Message: Does User X with Group Attribute Y have a static or dynamic account to access this resource?
5. OGSA SAML AuthZ Response e.g. Yes with Obligations (in XACML): Local UID = XYZ, GID = XYZ
User
Client Tools
- VOMS Proxy Init
- Globus Job Run
- GridFTP? Etc...
Client based on GT2.4 and up
Gate-keeper
Gridmap callout
PRIMA module
Access Control callout
SAZ module
3. Request with VOMS proxy
6. SAZ request
7. SAZ response
SAZ Server
(checking site blacklist, cert chain validity, user registration with site)

Open Science Grid

Submitting a job with GRAM

- **globus-job-run** command

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

  - ❑ 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

Job management layers

- High level user tools (Panda, Swift, Pegasus, …)
- OSG abstractions
- DAGman
- Condor
- Adding some Security
- GRAM, the Grid protocol
- Clusters and Local Resource Managers

Condor

- is a software system that creates an HTC environment
  - Created at UW-Madison
- Condor is a specialized workload management system for compute-intensive jobs.
  - Detects machine availability
  - Harnesses available resources
  - Uses remote system calls to send R/W operations over the network
  - 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 - features

- Checkpoint & migration
- Remote system calls
  - Able to transfer data files and executables across machines
- Job ordering
- *Job requirements and preferences can be specified via powerful expressions*

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)

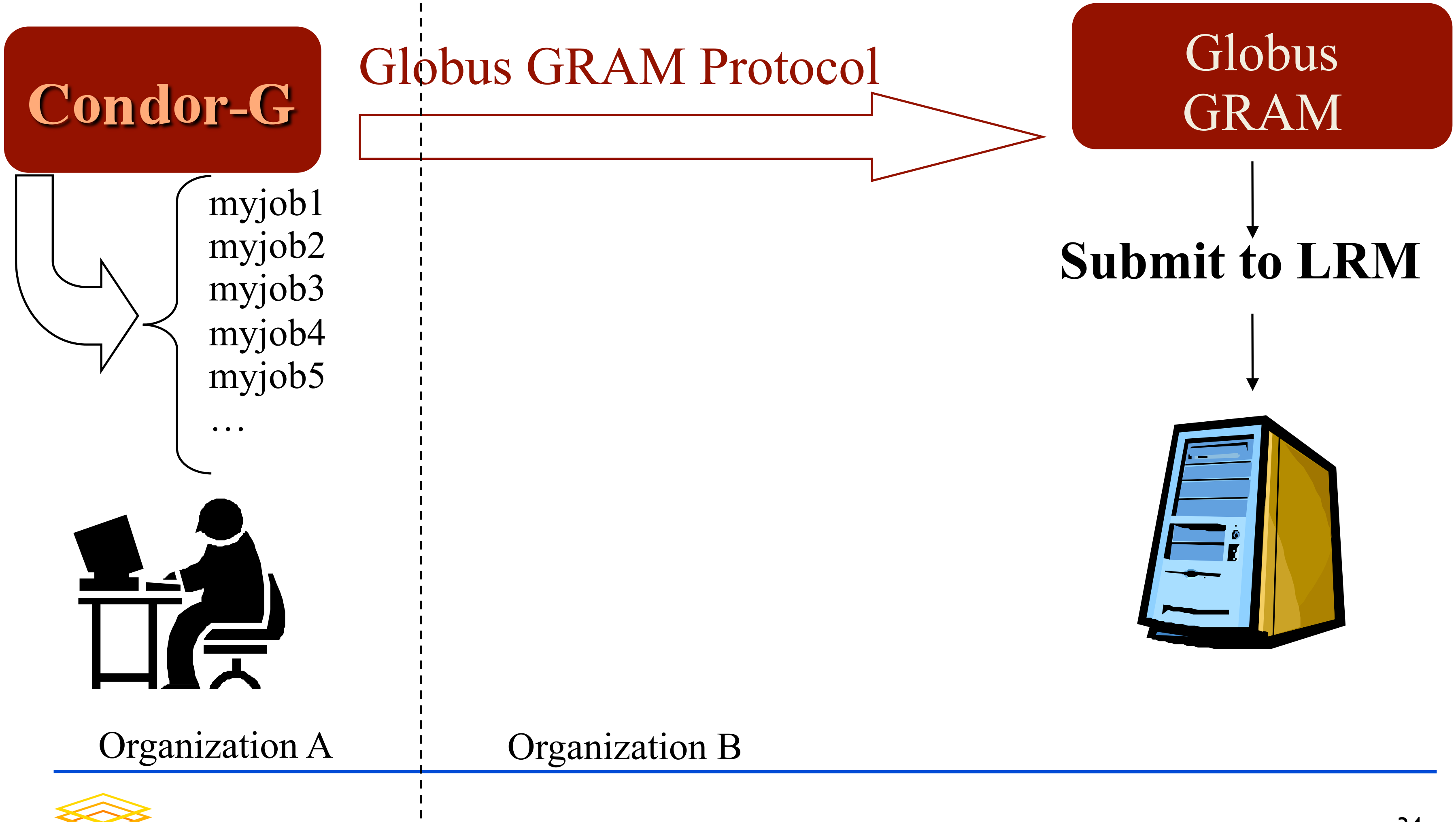
Condor-G

- 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.

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: Condor-G + Globus + Condor



Condor-G

Globus GRAM Protocol

Globus GRAM

myjob1
myjob2
myjob3
myjob4
myjob5
…

Submit to LRM

Organization A

Organization B

Condor-G: Access non-Condor Grid resources

the globus project
www.globus.org

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

Condor
High Throughput Computing

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

1. Choose a Universe

- There are many choices
  - ❑ **Vanilla**: any old job
  - ❑ **Standard**: checkpointing & remote I/O
  - ❑ **Java**: better for Java jobs
  - ❑ **MPI**: Run parallel MPI jobs
  - ❑ Virtual Machine: Run a virtual machine as job
  - ❑ …
- *For now, we'll just consider vanilla*

2. Make your job batch-ready

- Must be able to run in the background:
  - no interactive input, windows, GUI, etc.
- Condor is designed to run jobs as a batch system, with pre-defined inputs for jobs
- Can still use `STDIN`, `STDOUT`, and `STDERR` (the keyboard and the screen), but files are used for these instead of the actual devices
- Organize data files

3. Create a Submit Description File

- A plain ASCII text file
- Condor does not care about file extensions

- Tells Condor about your job:
  - ❑ Which executable to run and where to find it
  - ❑ Which universe
  - ❑ Location of input, output and error files
  - ❑ Command-line arguments, if any
  - ❑ Environment variables
  - ❑ Any special requirements or preferences

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   = vanilla
Executable = analysis
Log        = my_job.log
Queue
```

4. 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
# (Lines beginning with # are comments)
# NOTE: the words on the left side are not
#       case sensitive, but filenames are!
Universe   = vanilla
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
```

Details

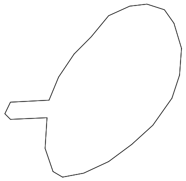
- Lots of options available in the submit file
- Commands to
  - watch the queue,
  - the state of your pool,
  - and lots more
- You'll see much of this in the hands-on exercises.

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

Job management layers

- High level user tools (Panda, Swift, Pegasus, …)
- OSG abstractions
- DAGman
- Condor
- Adding some Security
- GRAM, the Grid protocol
- Clusters and Local Resource Managers

DAGMan

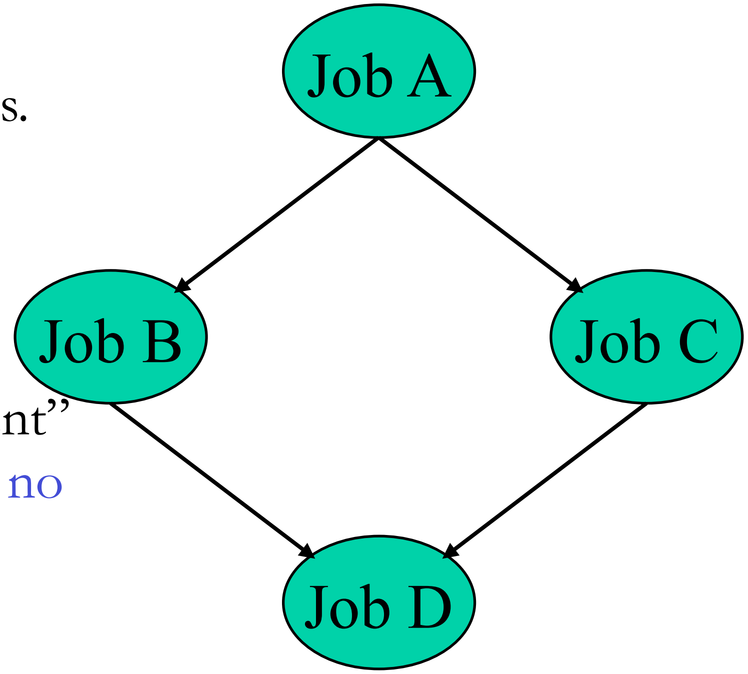
- **Directed Acyclic Graph Manager**

- 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?

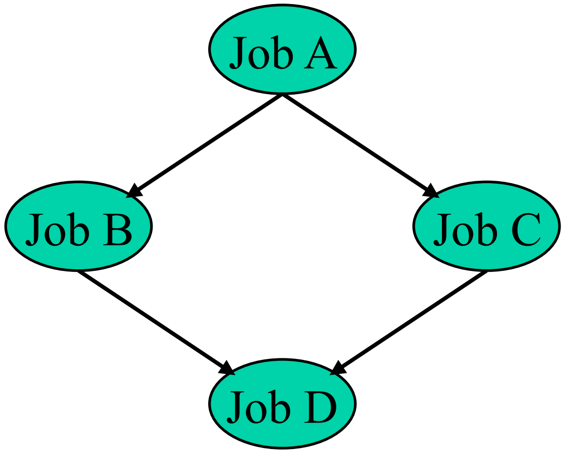
- A DAG is the data structure used by DAGMan to represent these dependencies.
- Each job is a "node" in the DAG.
- Each node can have any number of "parent" or "children" nodes – as long as there are no loops!



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

Submitting a DAG

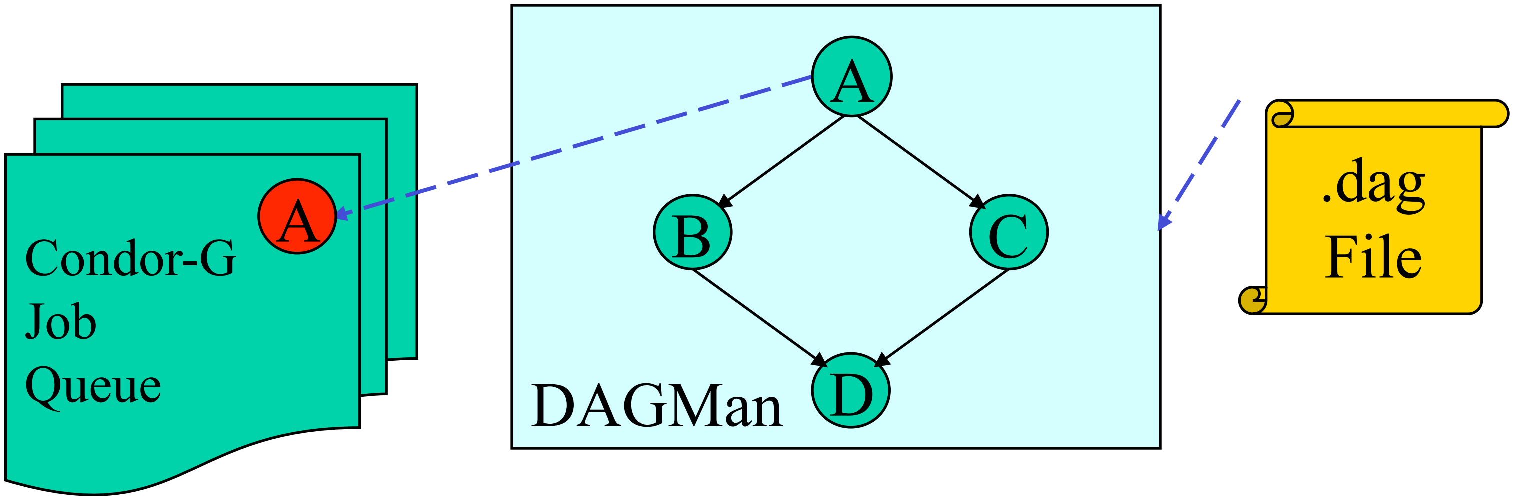
- To start your DAG, just run ***condor_submit_dag*** with your .dag file, and Condor will start a personal DAGMan daemon which to begin running your jobs:

```
% 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.

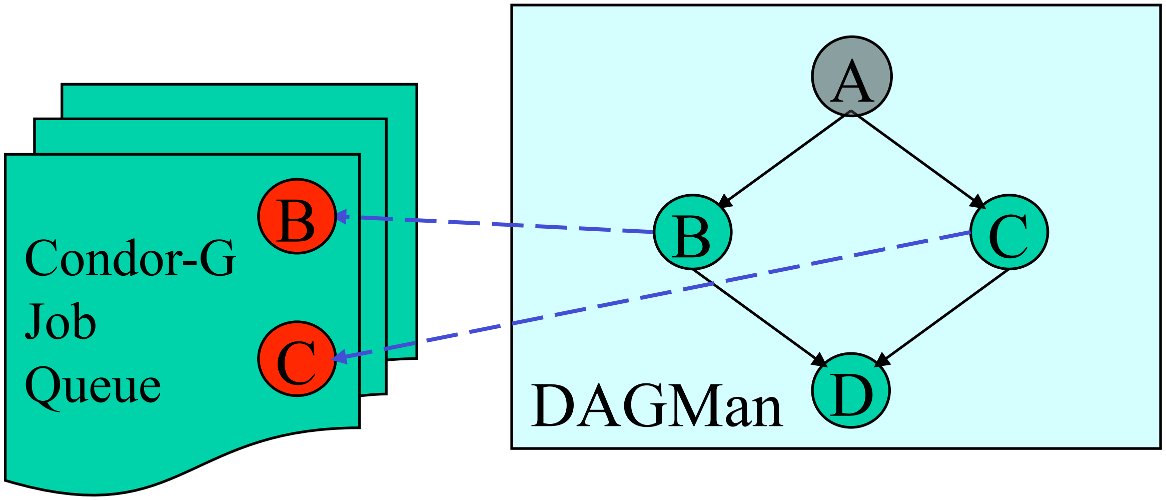
Running a DAG

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



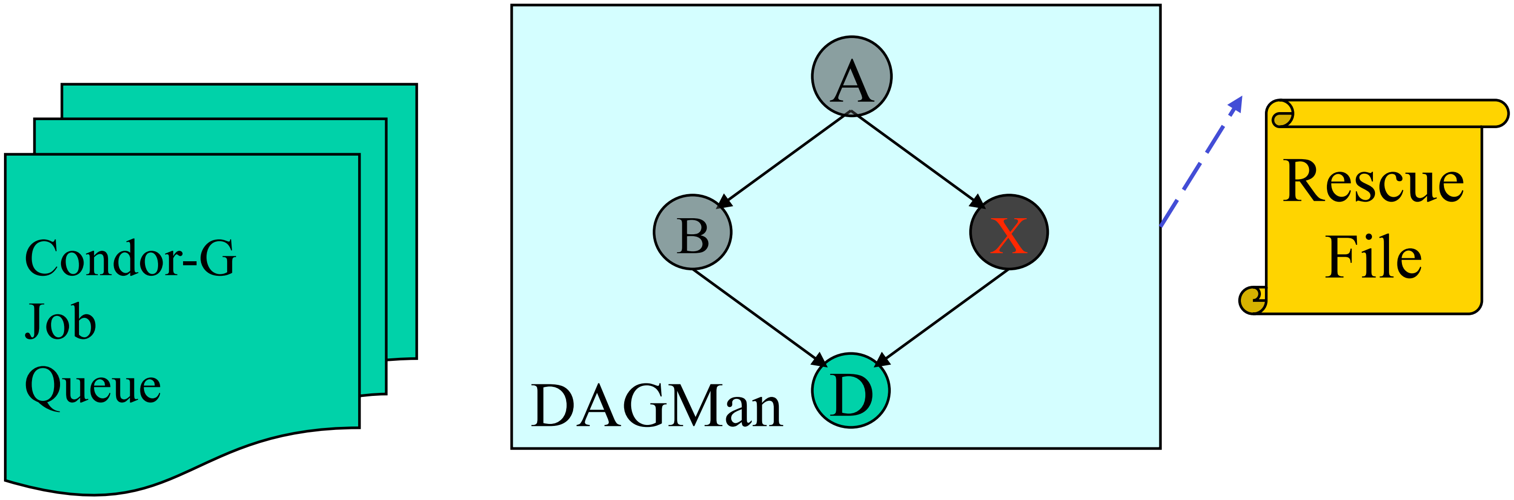
Running a DAG (cont'd)

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



Running a DAG (cont'd)

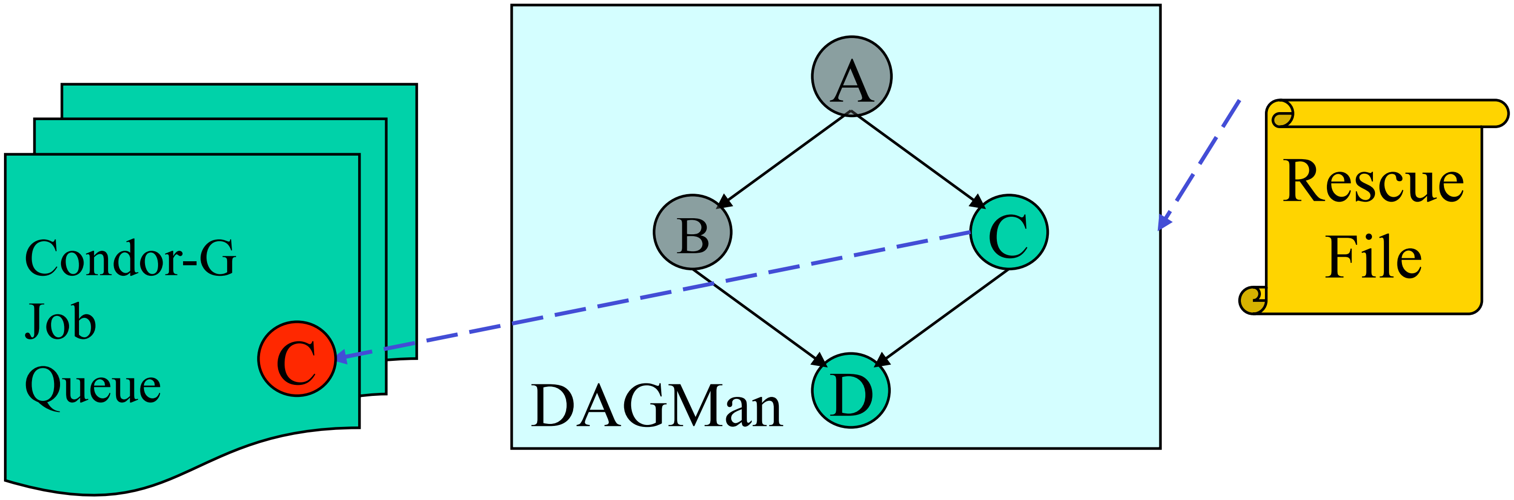
- In case of a job failure, DAGMan continues until it can no longer make progress, and then creates a *"rescue" file* with the current state of the DAG.



Open Science Grid

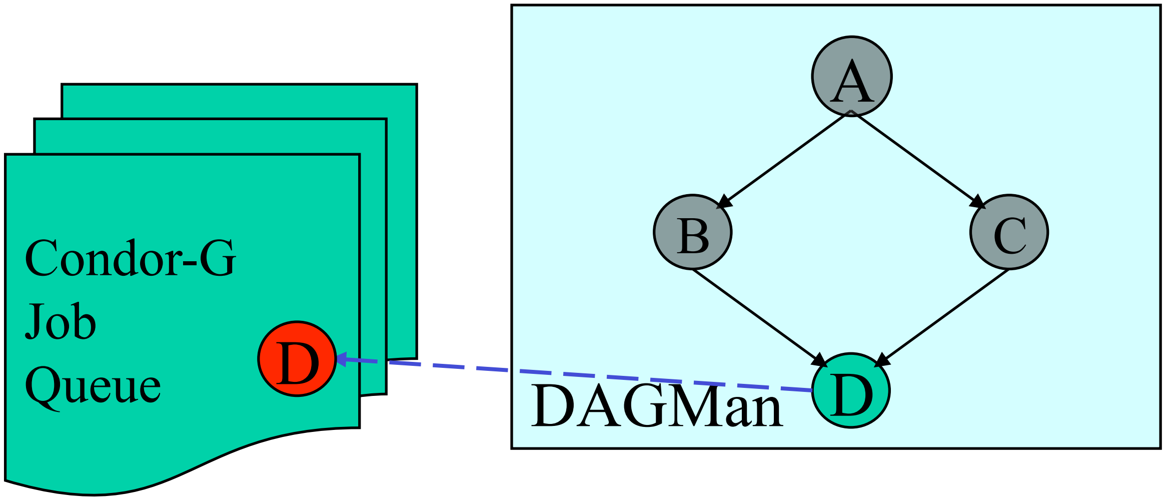
Recovering a DAG
-- fault tolerance

- 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)

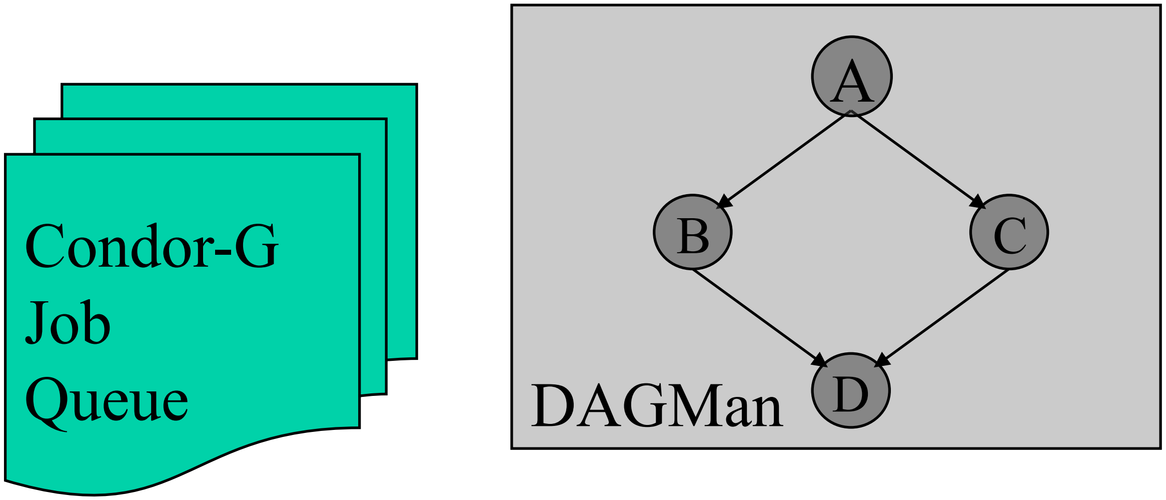
- Once that job completes, DAGMan will continue the DAG as if the failure never happened.



Open Science Grid

Finishing a DAG

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



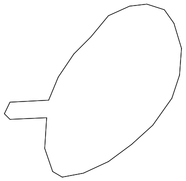
We have seen how Condor:

… monitors submitted jobs and reports progress

… implements your policy on the execution order of the jobs

… keeps a log of your job activities

Long jobs: if my jobs run for weeks …



- What happens to my job when
  - ❑ a machine is shut down
  - ❑ there is a network outage, or
  - ❑ another job with higher priority preempts it?

- Do I lose all of those hours or days of computation time??
- What happens when they get pre-empted?
- *How can I add fault tolerance to my jobs?*

Condor's Standard Universe to the rescue!

- Condor can support various combinations of features/ environments in different "Universes"
- Different Universes provide different functionalities to your job:
  - ❑ Vanilla: Run any serial job
  - ❑ Scheduler: Plug in a scheduler
  - ❑ Standard: Support for *transparent process checkpoint and restart*

provides two important services to your job:
process checkpoint
remote system calls.

Process Checkpointing

- Condor's process checkpointing mechanism ***saves the entire state of a process into a checkpoint file***
  - Memory, CPU, I/O, etc.
- The process can then be *restarted* from the point it left off
- Typically no changes to your job's source code needed—however, your job must be relinked with Condor's Standard Universe support library

Job management layers

- High level user tools (Panda, Swift, Pegasus, …)
- OSG abstractions
- DAGman
- Condor
- Adding some Security
- GRAM, the Grid protocol
- Clusters and Local Resource Managers

OSG & job submissions

- OSG sites present interfaces allowing remotely submitted jobs to be accepted, queued and executed locally.
- OSG supports the Condor-G job submission client which interfaces to either the pre-web service or web services GRAM Globus interface at the executing site.
- Job managers at the backend of the GRAM gatekeeper support job execution by local Condor, LSF, PBS, or SGE batch systems.

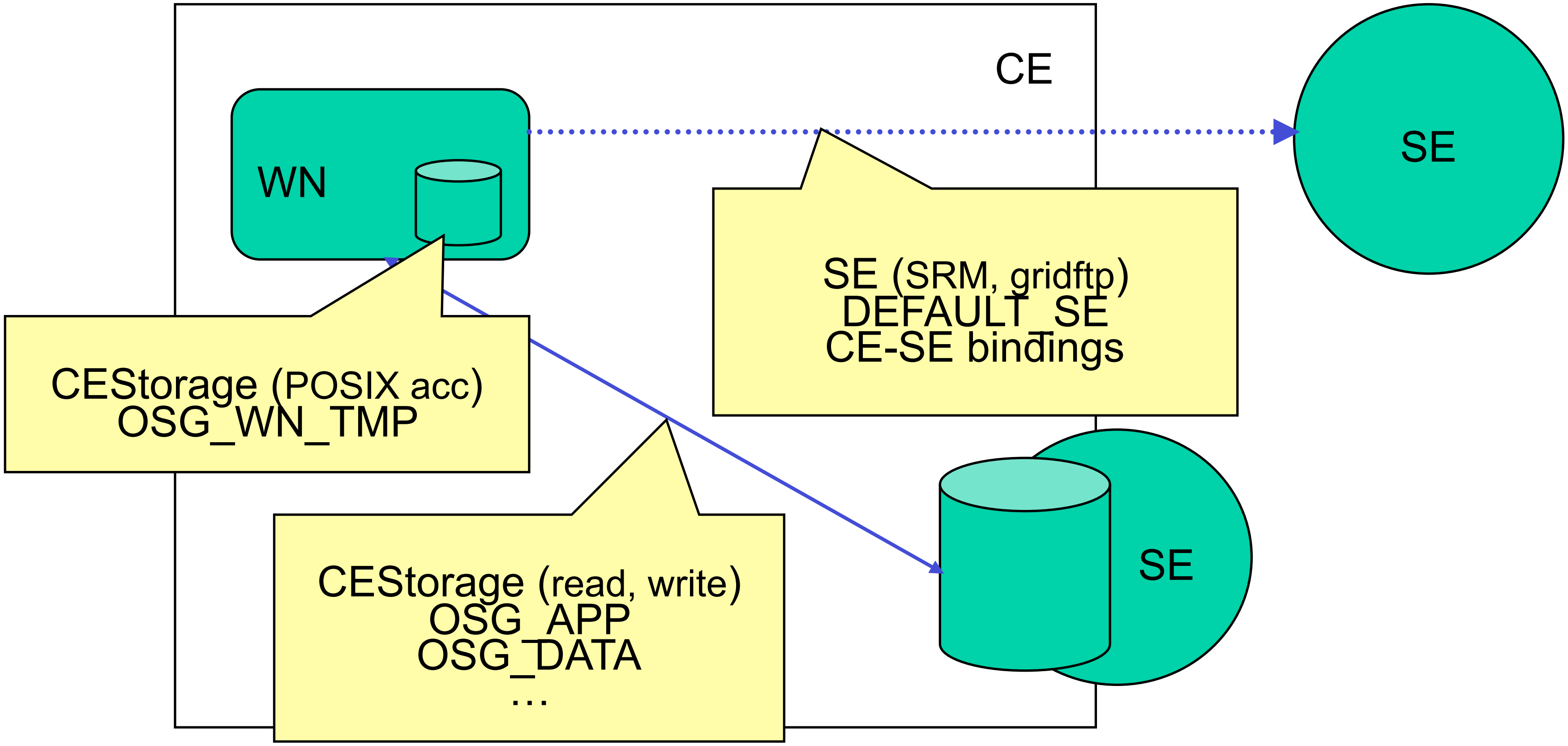
OSG and Environment

- Information systems to discover and describe resources
- Guidelines on how to execute jobs
- Guidelines on how to use available disk spaces
- Jobs find an uniform standard environment
    - OSG_AAA variables

Everything has a place

- OSG_APP
  - directory to install applications
- OSG_GRID
  - directory to find grid clients (OSG:WN)
- OSG_WN_TMP
  - local working directory
- OSG_DATA
  - shared directory
- OSG_SITE_READ, OSG_SITE_WRITE
  - optimized for efficient read/write
- DEFAULT_SE
  - storage element close to this computing element

CE View

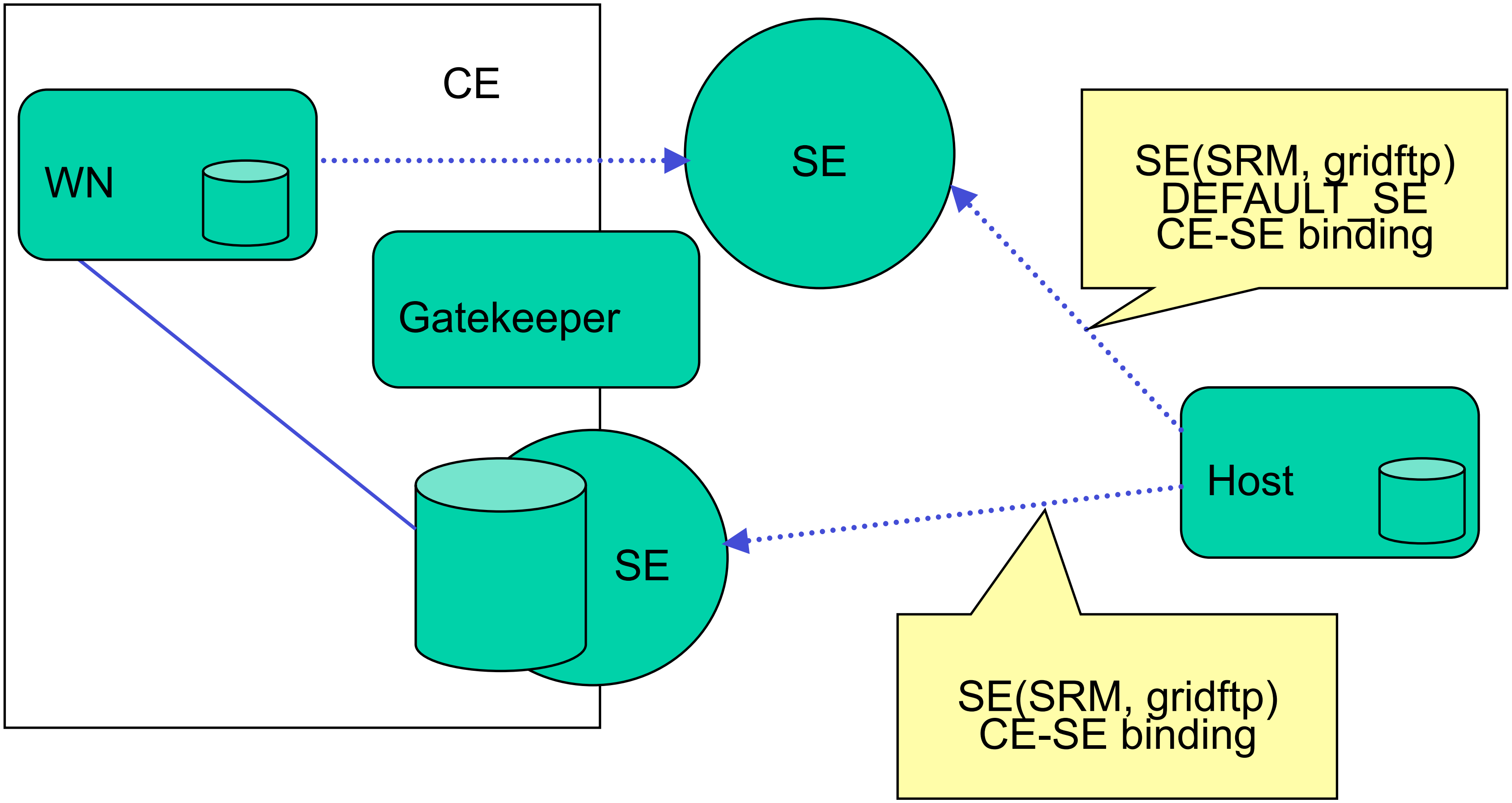


SE (SRM, gridftp)
DEFAULT_SE
CE-SE bindings

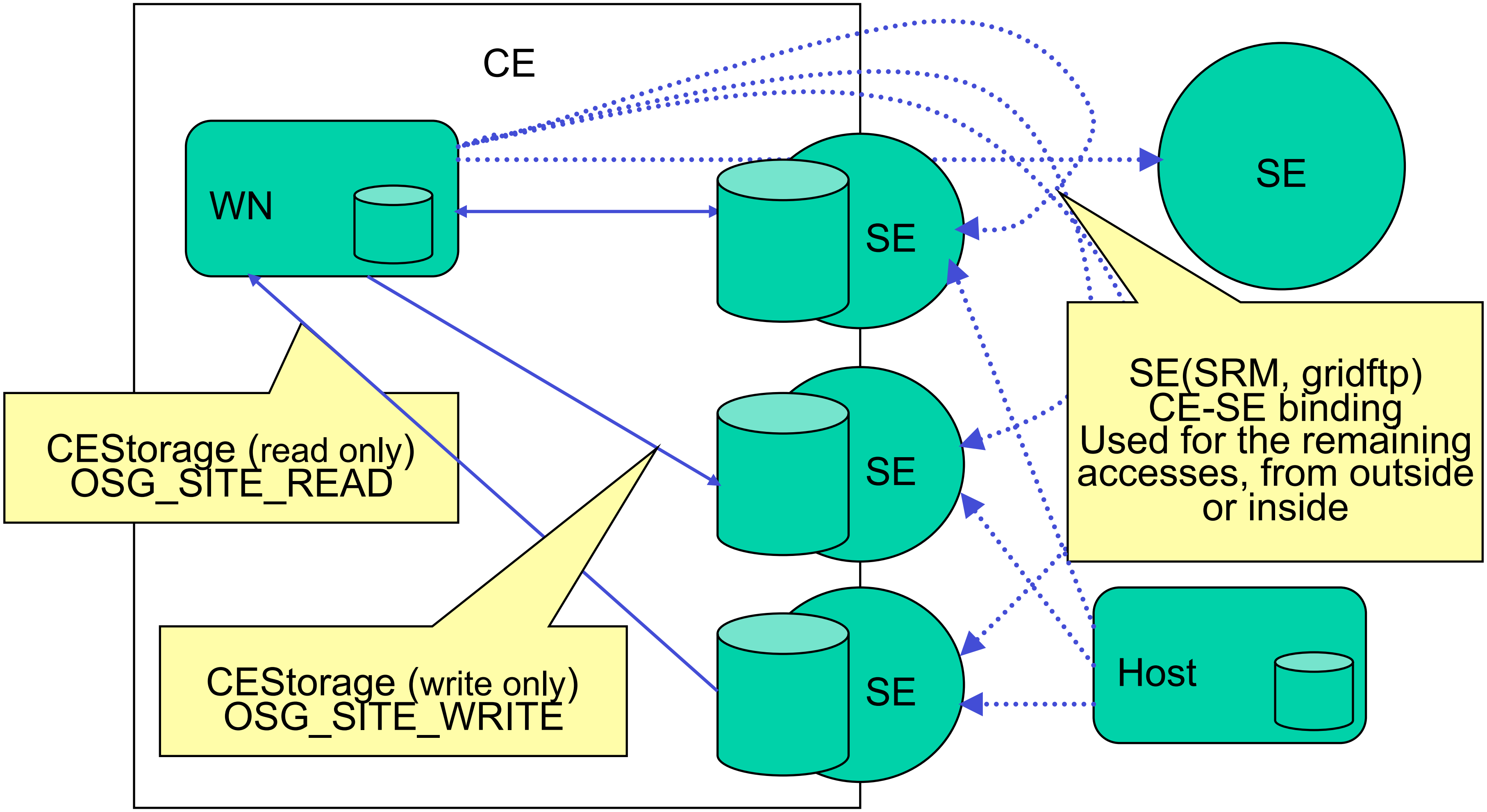
CEStorage (POSIX acc)
OSG_WN_TMP

CEStorage (read, write)
OSG_APP
OSG_DATA
…

Outside View



CE View (OSG_WRITE, OSG_READ)



CEStorage (read only)
OSG_SITE_READ

CEStorage (write only)
OSG_SITE_WRITE

SE(SRM, gridftp)
CE-SE binding
Used for the remaining accesses, from outside or inside

Open Science Grid

Acknowledgments:

This presentation based on: Grid Resources and Job Management

Open Science Grid

Jaime Frey and Becky Gietzel

Condor Project

U. Wisconsin-Madison

Job management layers

- High level user tools (Panda, Swift, Pegasus, …)
- OSG abstractions
- DAGman
- Condor
- Adding some Security
- GRAM, the Grid protocol
- Clusters and Local Resource Managers