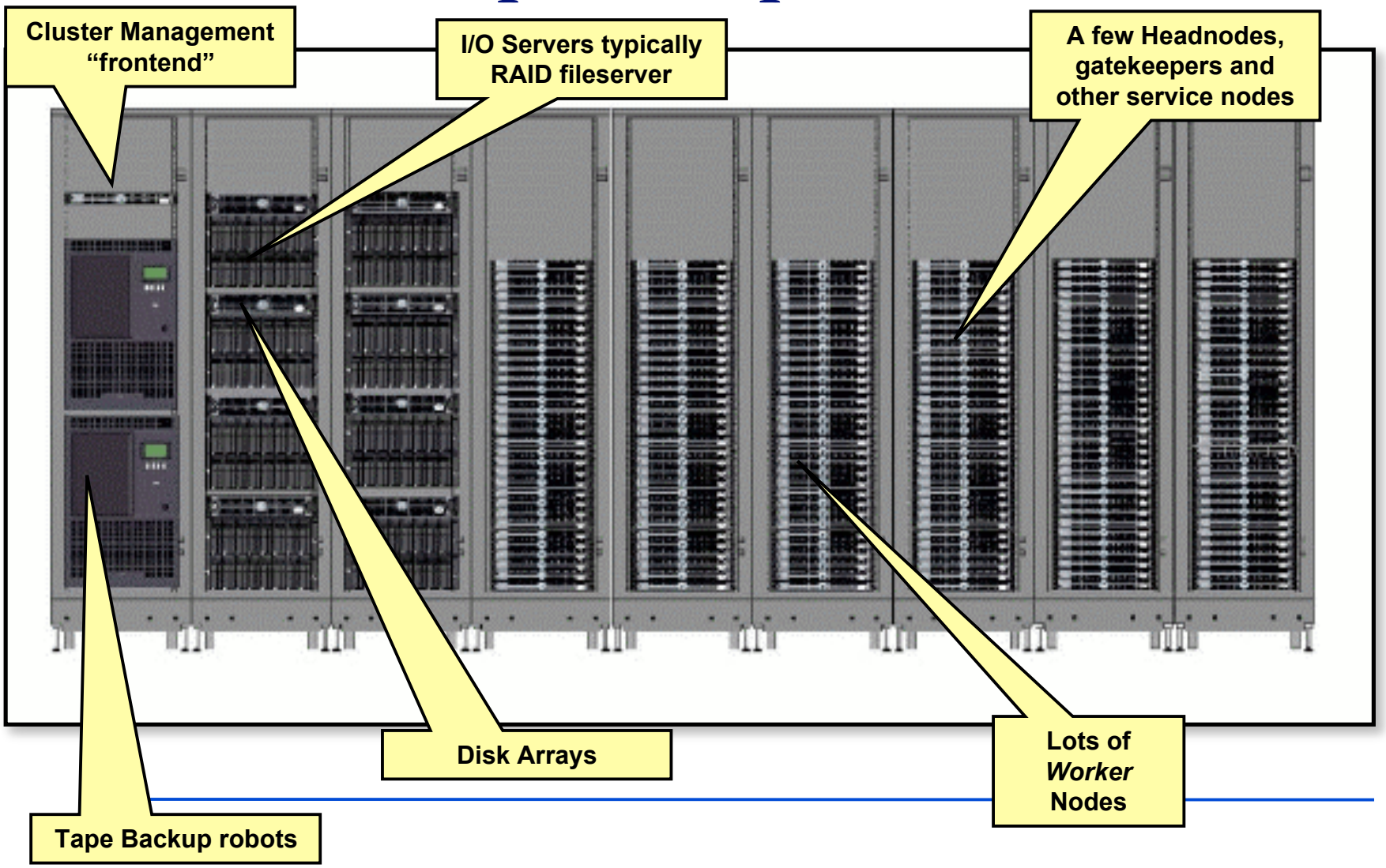
# Introduction to Grid Computing

New Users Training @ OSG All Hands Meeting

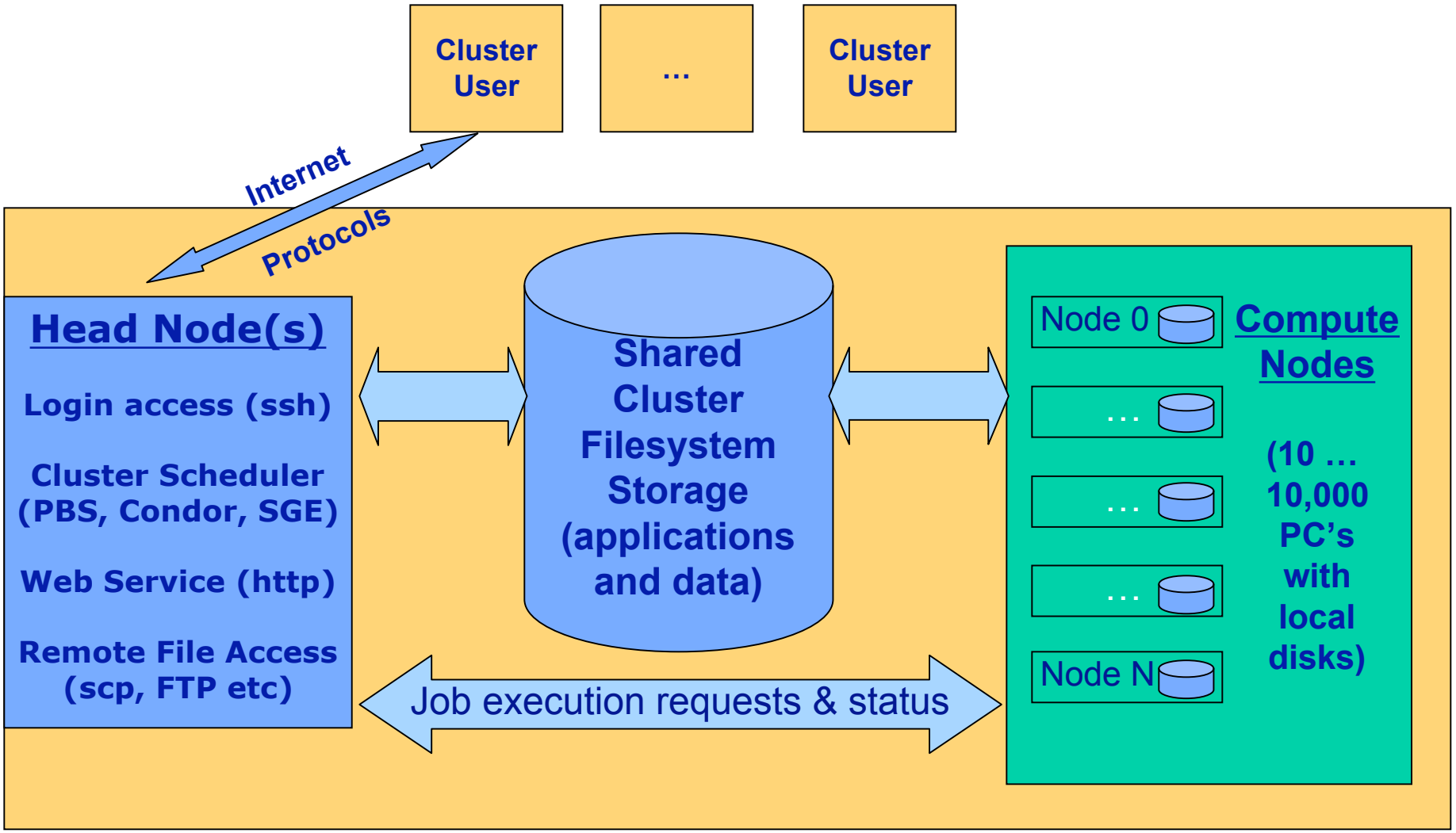
Alina Bejan - University of Chicago

March 6, 2008

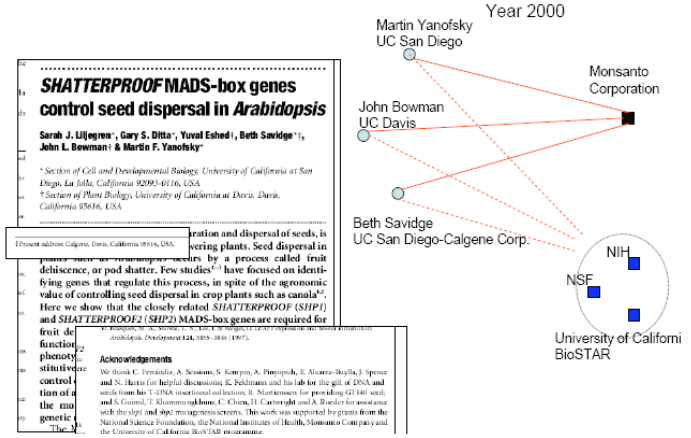
# Computing "Clusters" are today's Supercomputers

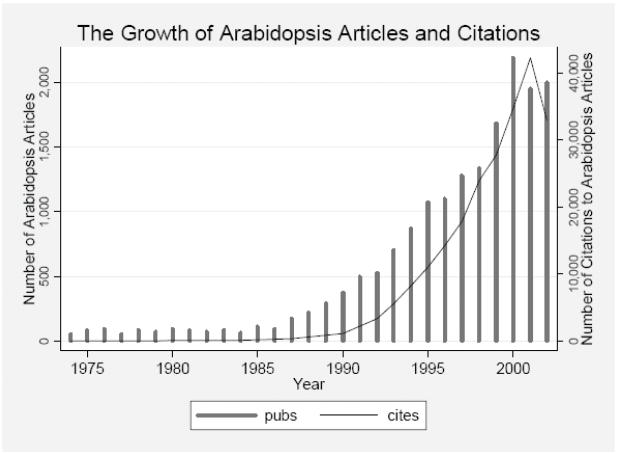


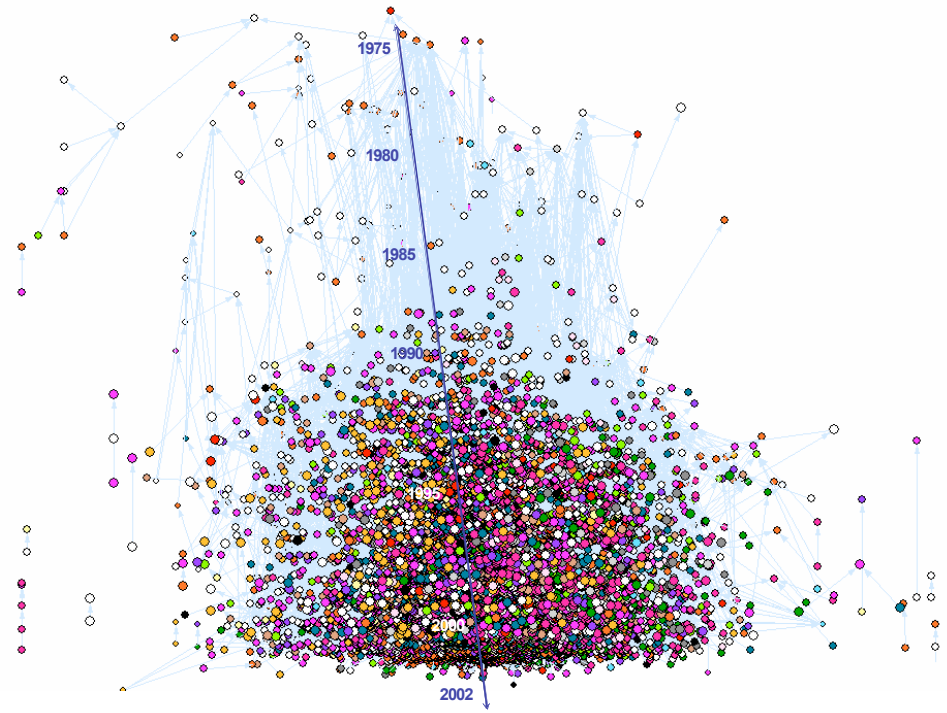
# Cluster Architecture



# Scaling up Science: Citation Network Analysis in Sociology



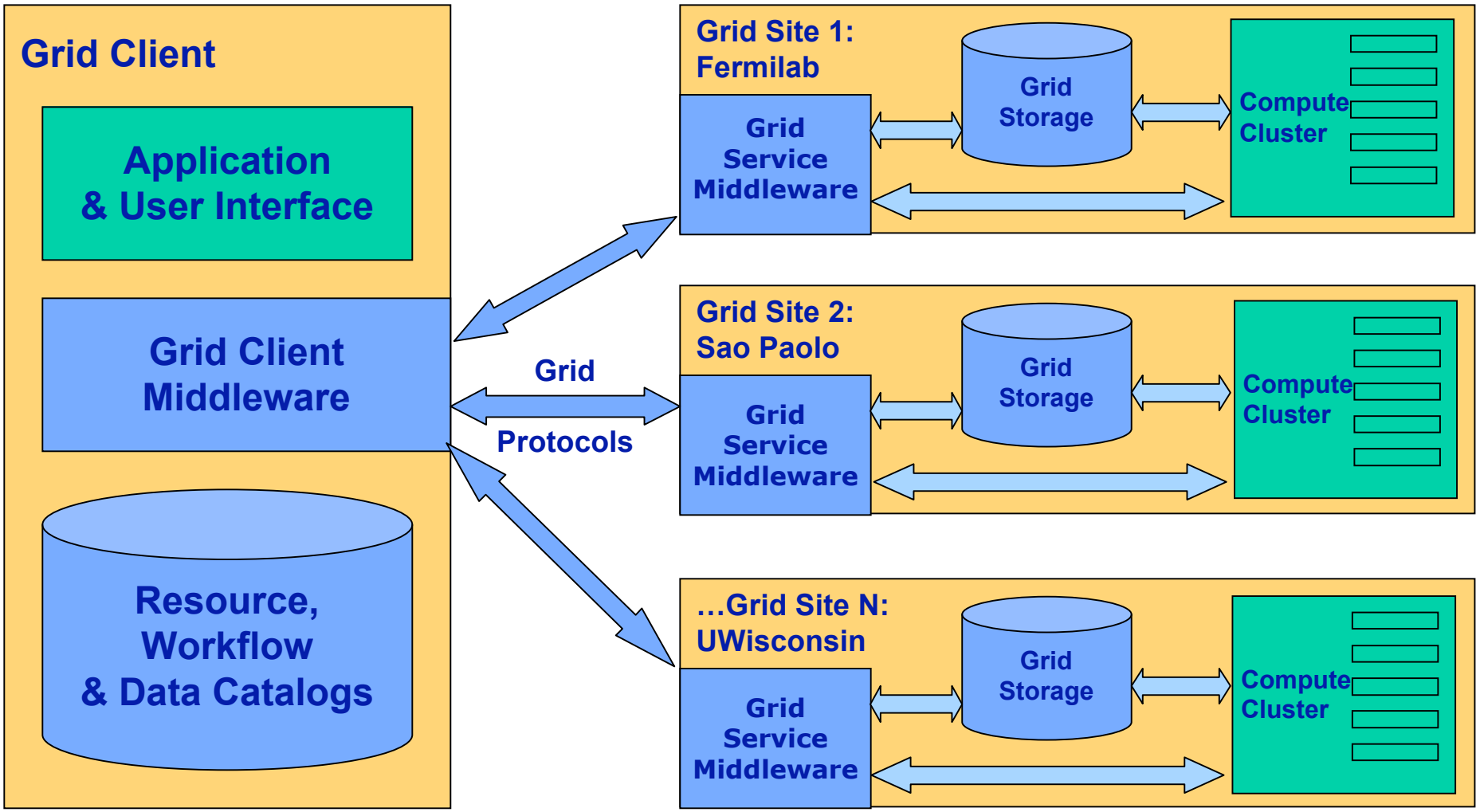

- miscelaneous
- transcription factors / morphogens
- receptors
- phosphosphorylation cascades
- organogenesis
- terpines, synthesis
- physical defense
- commercial disease resistance
- innate immunology
- nutrient metabolism and movement
- nutrient uptake
- genomics
- photosynthesis
- functional enzymatics
- protein isolation & characterization
- targeting / splicing
- tropisms

*Work of James Evans, University of Chicago, Department of Sociology*

# Scaling up the analysis

- Query and analysis of 25+ million citations
- Work started on desktop workstations
- Queries grew to month-long duration
- With data distributed across U of Chicago TeraPort **cluster**:
  - 50 (faster) CPUs gave 100 X speedup
  - Many more methods and hypotheses can be tested!
- Higher *throughput* and *capacity* enables *deeper analysis* and *broader community access*.

# Grids consist of distributed clusters



# HPC vs. HTC

HTC: many computing resources over long periods of time to accomplish a computational task.

HPC tasks are characterized as needing large of amounts of computing power for short periods of time, whereas HTC tasks also require large amounts of computing, but for much longer times (months and years, rather than hours and days).

Fine-grained calculations are better suited to **HPC**: big, monolithic supercomputers, or very tightly coupled computer clusters with lots of identical processors and an extremely fast, reliable network between the processors.

Embarrassingly parallel calculations are ideal for **HTC:** more loosely-coupled networks of computers where delays in getting results from one processor will not affect the work of the others.

# Grids represent a different approach

- Building HTC resource by joining clusters together in a ***grid***
- Example:

Open Science Grid

***The OSG***

Current Compute Resources:

- 70 Open Science Grid sites, ***but all very different***
- Connected via Inet2, NLR.... from 10 Gbps – 622 Mbps
- Compute Elements & Storage Elements
- Most are Linux clusters
- Most are shared
  - Campus grids
  - Local non-grid users
- More than 20,000 CPUs
  - A lot of opportunistic usage
  - Total computing capacity difficult to estimate
  - Same with Storage

# Initial Grid driver: High Energy Physics

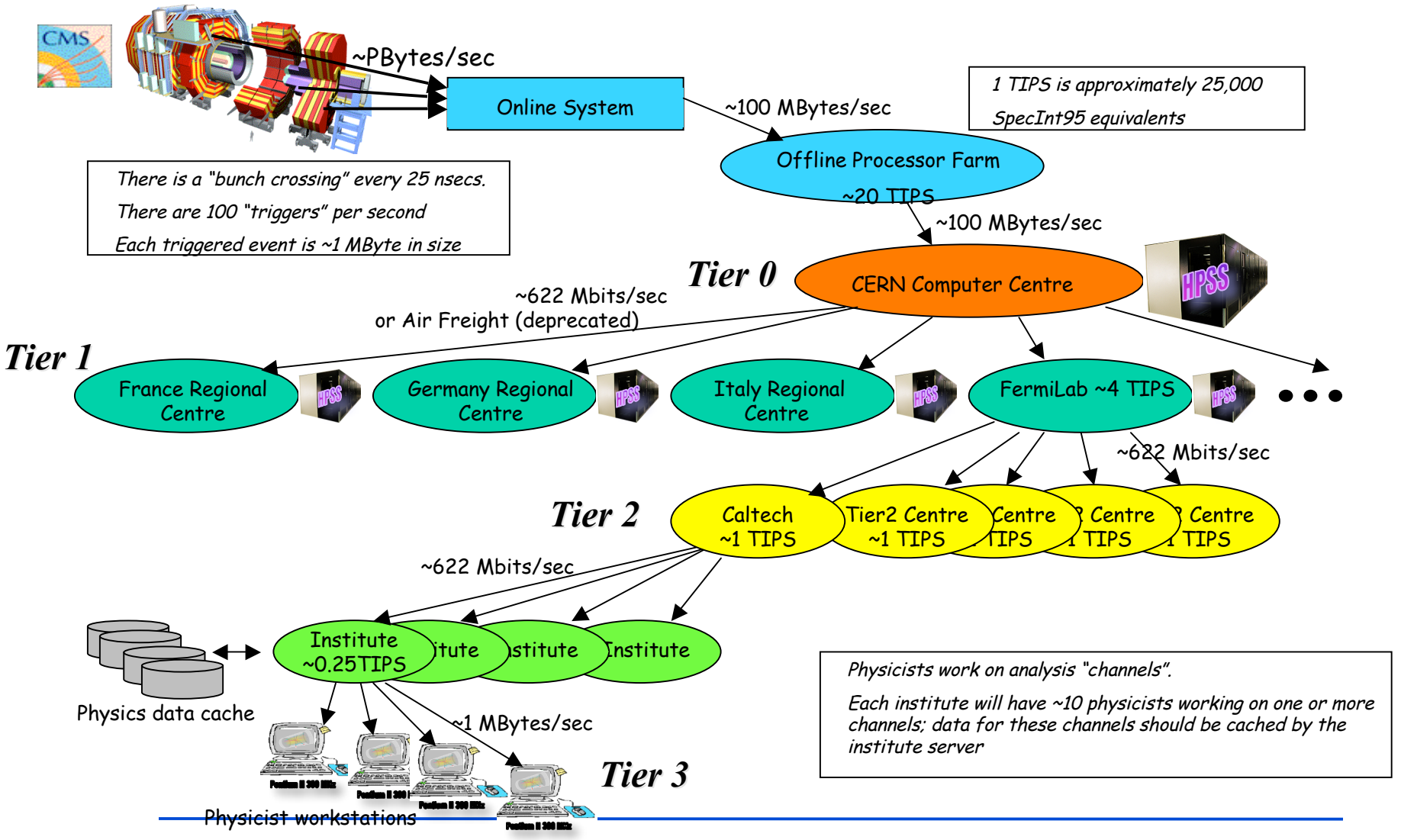
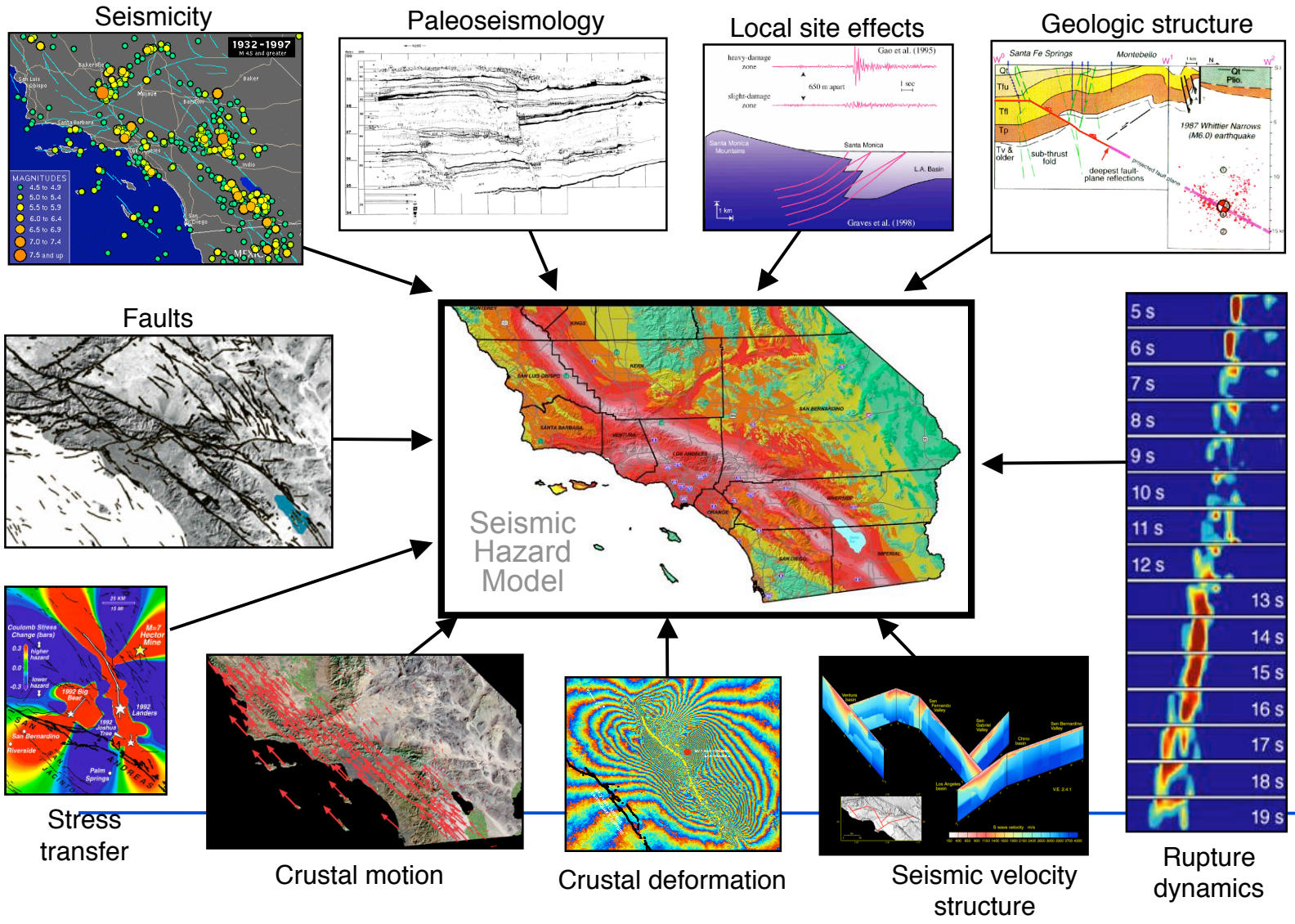


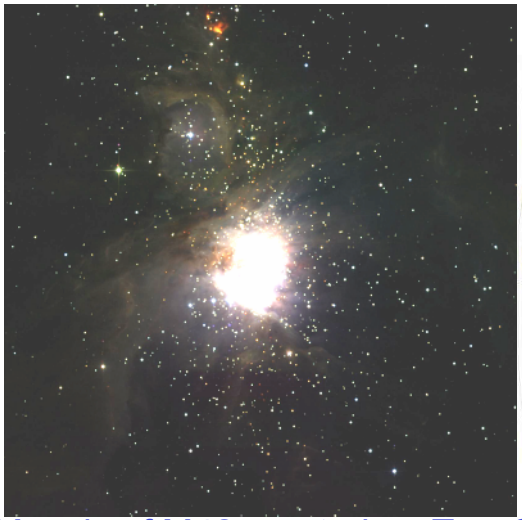

Image courtesy Harvey Newman, Caltech

# Mining Seismic data for hazard analysis (Southern Calif. Earthquake Center).

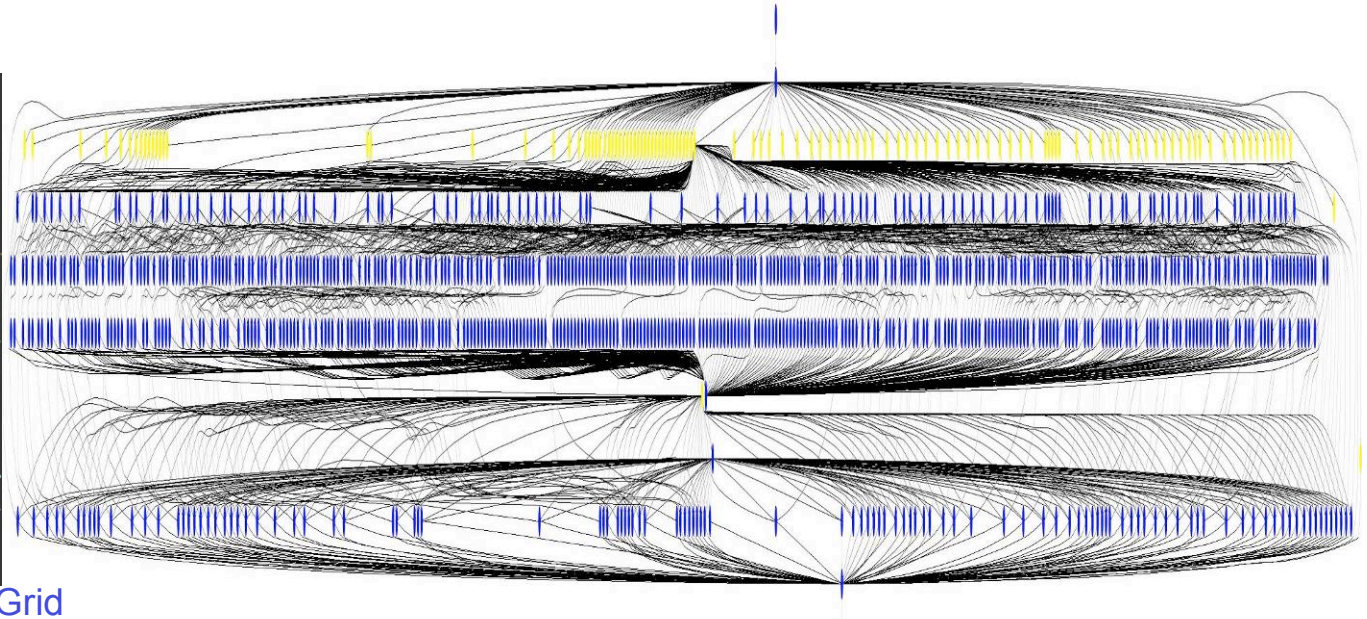


# Grids can process vast datasets.

- Many HEP and Astronomy experiments consist of:
  - Large datasets as inputs (find datasets)
  - "Transformations" which work on the input datasets (process)
  - The output datasets (store and publish)
- The emphasis is on the sharing of these large datasets
- *Workflows* of *independent* program can be *parallelized*.



Mosaic of M42 created on TeraGrid

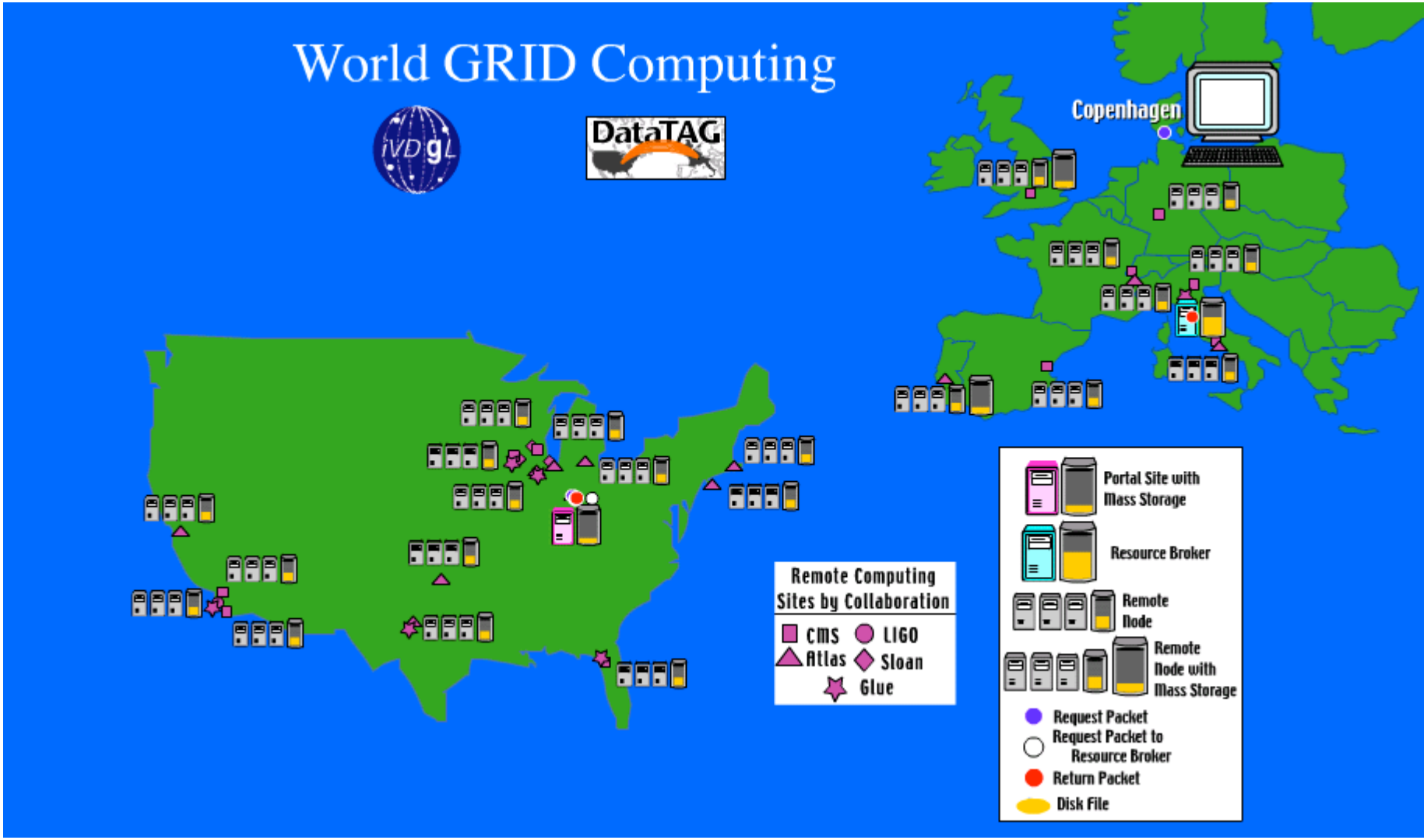


= Data Transfer

= Compute Job

Montage Workflow: ~1200 jobs, 7 levels
NVO, NASA, ISI/Pegasus - Deelman et al.

# Grids Provide Global Resources To Enable e-Science



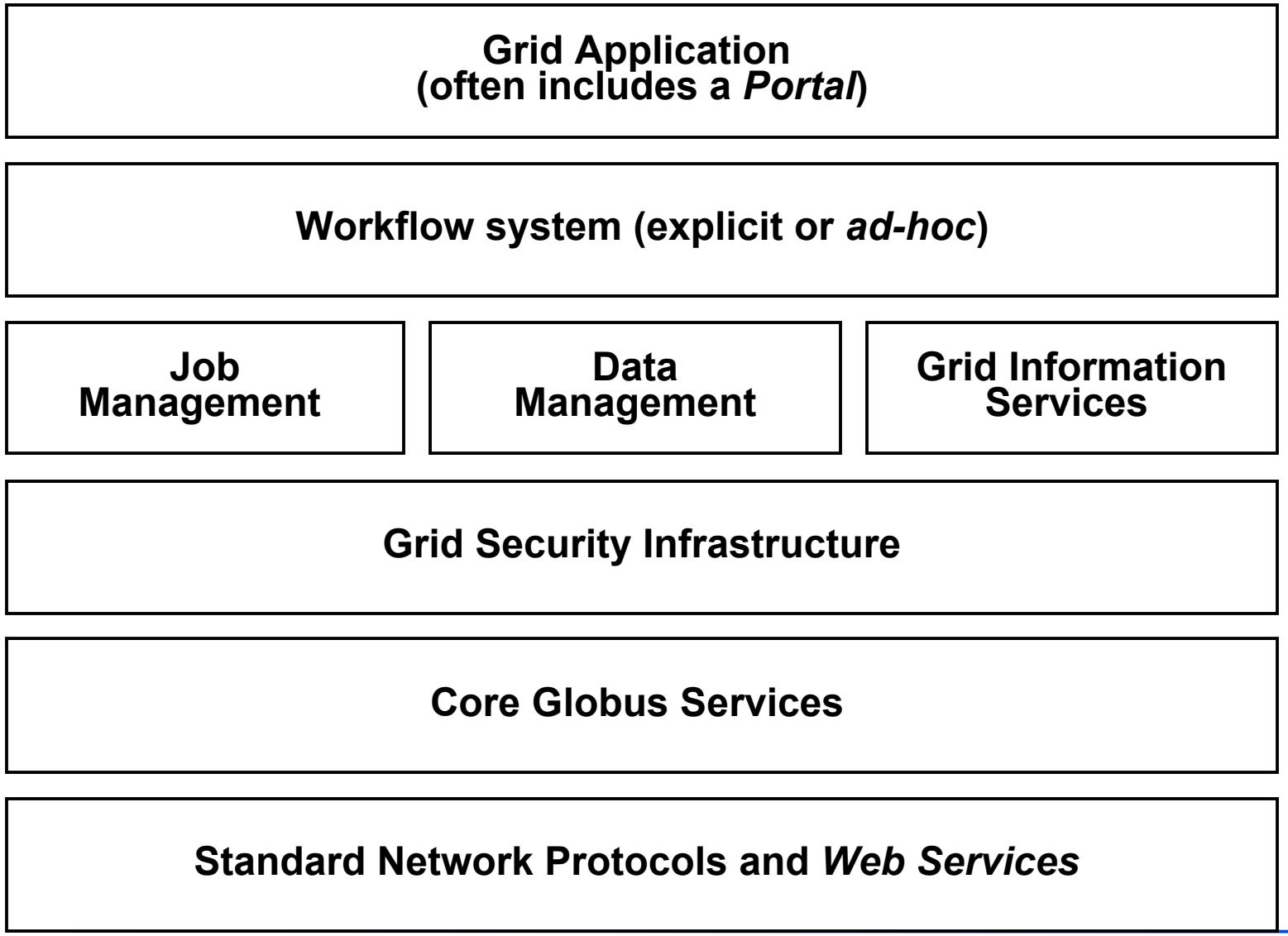
# Virtual Organizations

- Groups of organizations that use the Grid to share resources for ***specific purposes***
- Support a single community
- Deploy compatible technology and agree on working policies
  - Security policies - difficult
- Deploy different network accessible services:
  - Grid Information
  - Grid Resource Brokering
  - Grid Monitoring
  - Grid Accounting

# Ian Foster's Grid Checklist

- A Grid is a system that:
  - *Coordinates resources that are not subject to centralized control*
  - *Uses standard, open, general-purpose protocols and interfaces*
  - *Delivers non-trivial qualities of service*

# The Grid Middleware Stack *(and course modules)*



# Job and resource management

# How do we access the grid ?

- Command line with tools that you'll use
- Specialised applications
  - Ex: Write a program to process images that sends data to run on the grid as an inbuilt feature.
- Web portals
  - I2U2
  - SIDGrid

# Grid Middleware glues the grid together

- *A short, intuitive definition:*

  the software that glues together different clusters into a grid, taking into consideration the socio-political side of things (such as common policies on who can use what, how much, and what for)

# Grid middleware

- Job management
- Storage management
- Information Services
- Security

# **Globus** and **Condor** play key roles

- **Globus** Toolkit provides the ***base middleware***
  - Client tools which you can use from a command line
  - APIs (scripting languages, C, C++, Java, …) to build your own tools, or use direct from applications
  - Web service interfaces
  - Higher level tools built from these basic components, e.g. Reliable File Transfer (RFT)
- **Condor** provides both ***client & server scheduling***
  - In grids, Condor provides an agent to queue, schedule and manage work submission

# Globus Toolkit

- Developed at Uchicago/ANL and USC/ISI (Globus Alliance) - globus.org
- Open source
- Adopted by different scientific communities and industries
- Conceived as an open set of architectures, services and software libraries that support grids and grid applications
- Provides services in major areas of distributed systems:
  - Core services
  - Data management
  - Security

# Globus - core services

- A toolkit that can serve as a foundation to build a grid
- Authorization
- Message level security
- System level services (e.g., monitoring)
- Associated data management provides file services
  - GridFTP
  - RFT (Reliable File Transfer)
  - RLS (Replica Location Service)

# 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

# Local Resource Manager: a batch scheduler for running jobs on a computing cluster

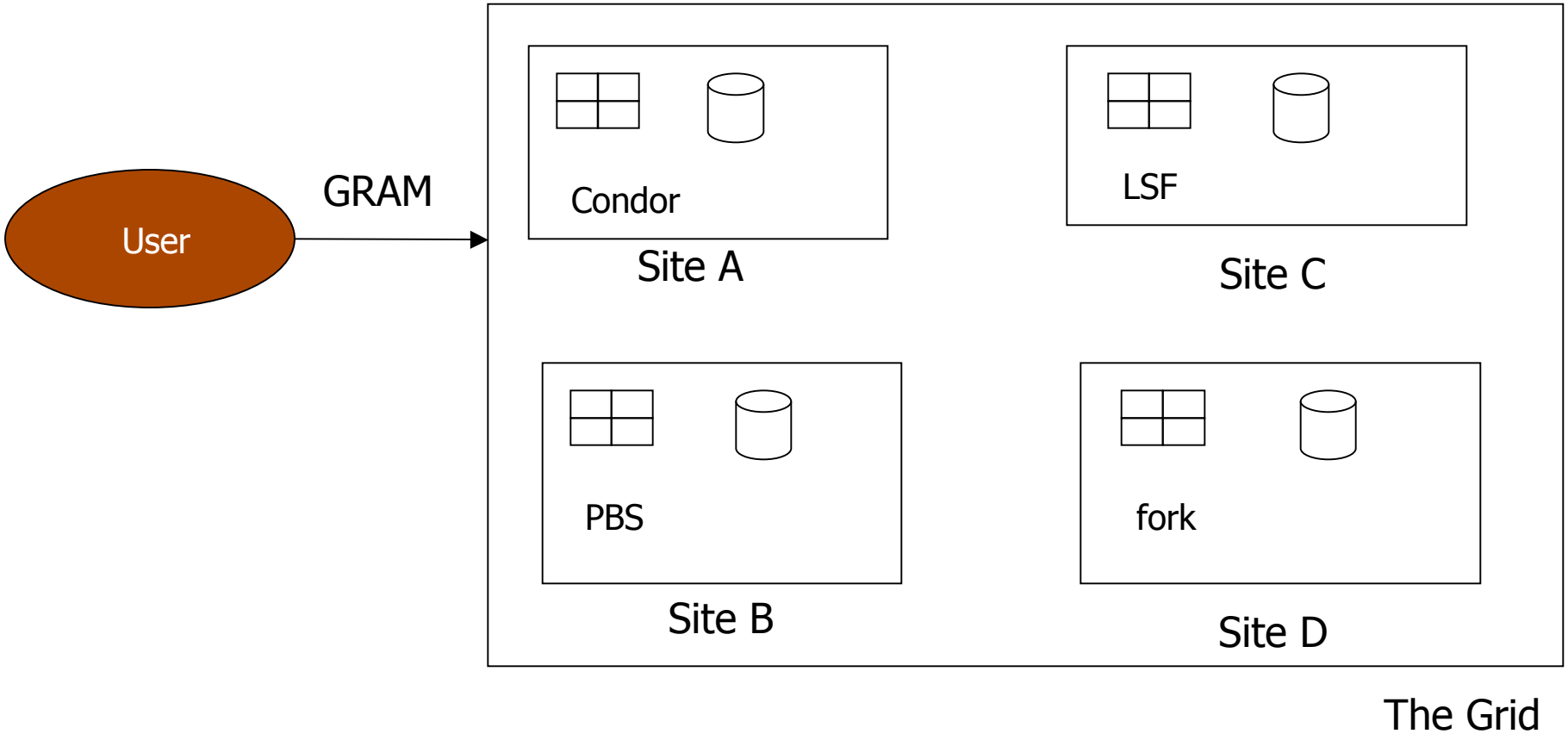
- Popular LRMs include:
  - PBS – Portable Batch System
  - LSF – Load Sharing Facility
  - SGE – Sun Grid Engine
  - *Condor – Originally for cycle scavenging, Condor has evolved into a comprehensive system for managing computing*
- LRMs execute on the cluster's *head node*
- Simplest LRM allows you to "fork" jobs quickly
  - Runs on the head node (*gatekeeper)* for fast utility functions
  - No queuing (but this is emerging to "throttle" heavy loads)
- In GRAM, each LRM is handled with a "job manager"

# 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

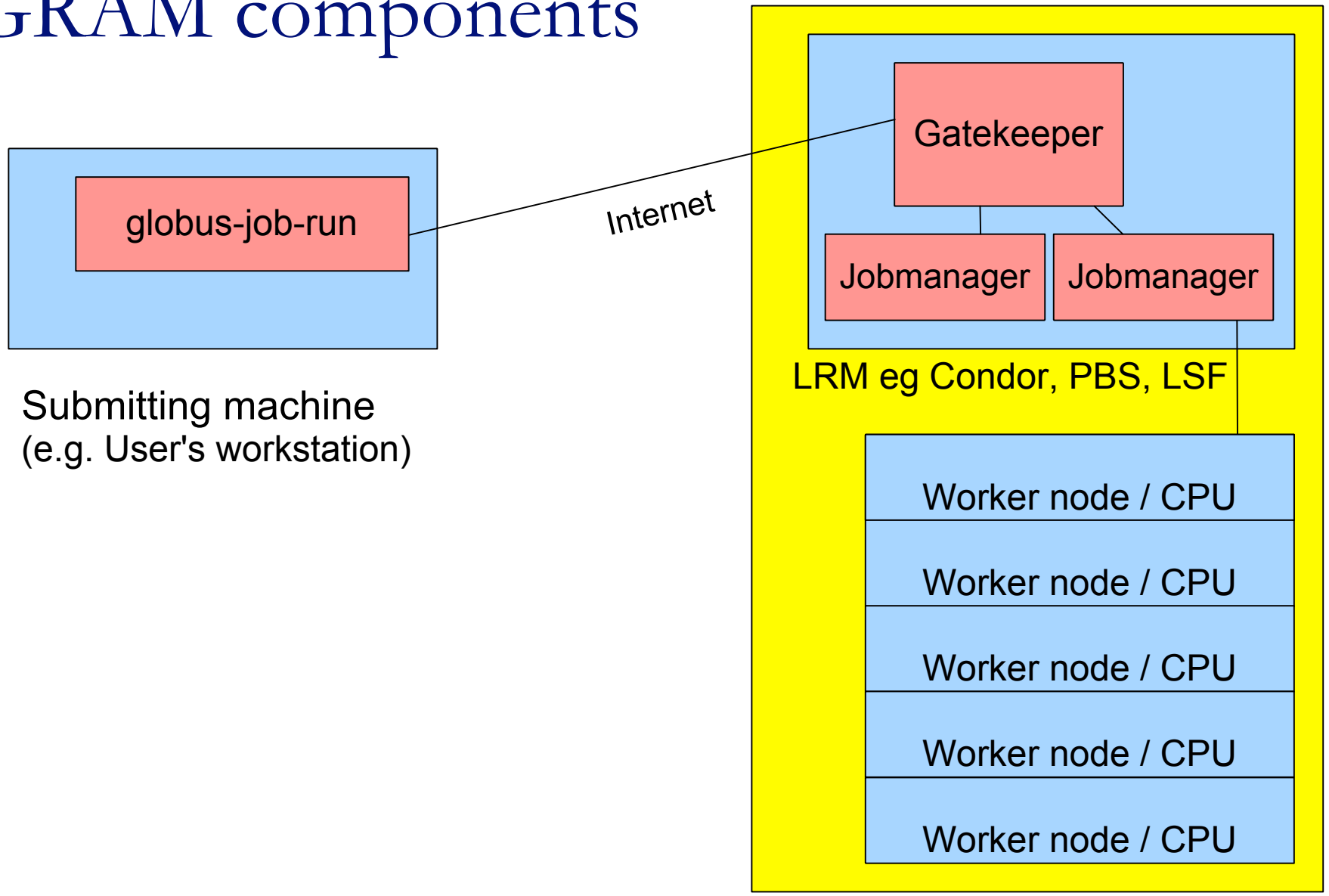
# Job Management on a Grid



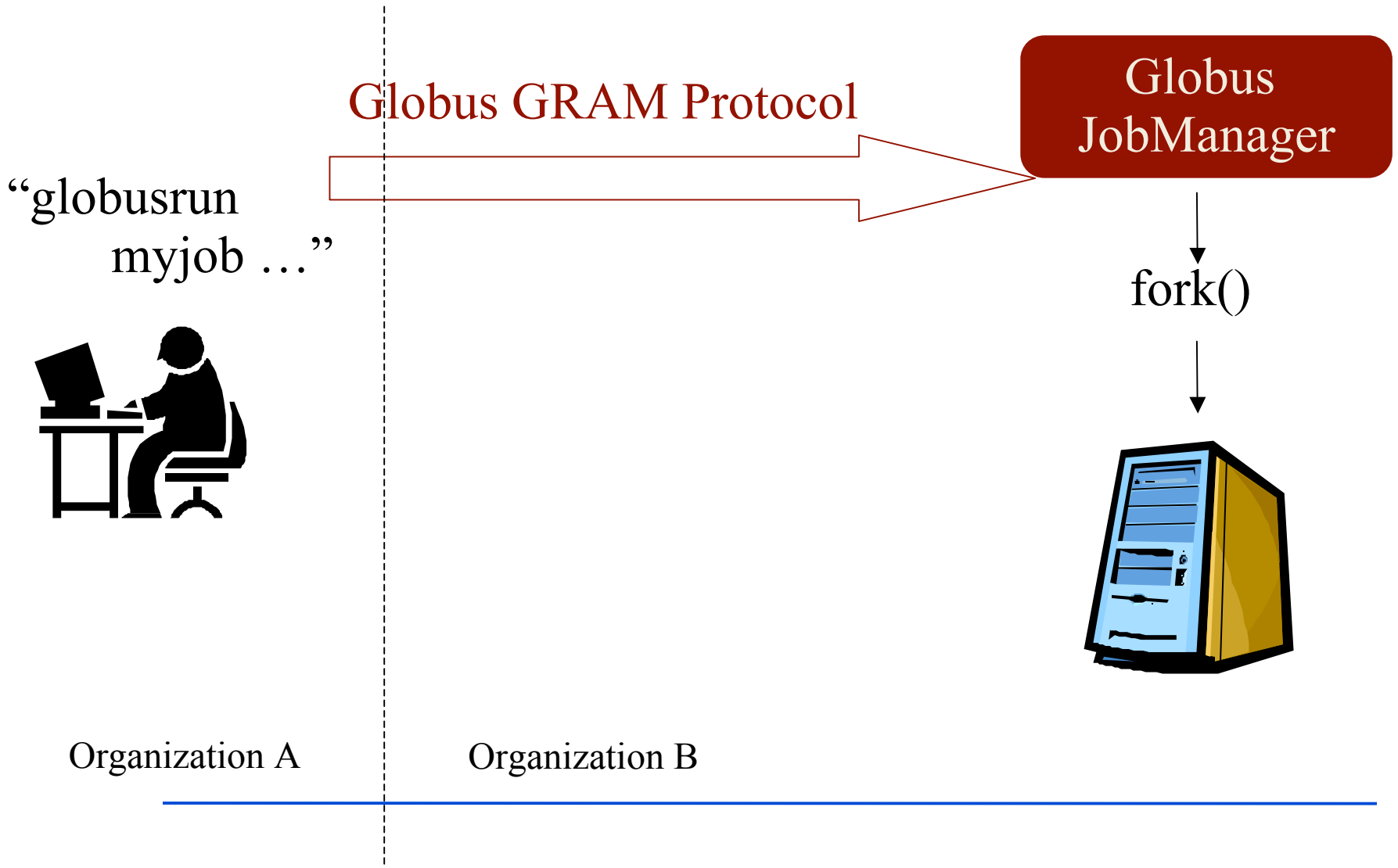
# GRAM's abilities

- 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



# Remote Resource Access: Globus



# Submitting a job with GRAM

- **globus-job-run** command

```
$ globus-job-run workshop1.ci.uchicago.edu
/bin/hostname
```

  - Run '/bin/hostname' on the resource workshop1.ci.uchicago.edu

- We don't care what LRM is used on ‘workshop1’ 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
workshop2.ci.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

- Condor is a specialized workload management system for compute-intensive jobs (aka batch system)
- is a software system that creates an HTC environment
  - Created at UW-Madison
  - 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)

# Condor manages your cluster

- Given a set of computers…
  - Dedicated
  - Opportunistic (Desktop computers)
- And given a set of jobs…
  - Can be a very large set of jobs
- Condor will run the jobs on the computers
  - Fault-tolerance (restart the jobs)
  - Priorities
  - Can be in a specific order
  - With more features than we can mention here…

# 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-G

- Condor has a neat feature: instead of submitting just to your local cluster, you can submit to an external grid.
  - The “G” in Condor-G
- Many grid types are supported:
  - Globus (old or new)
  - Nordugrid
  - CREAM
  - Amazon EC2
  - … others
- Condor-G is a marketing term
  - It’s a feature of Condor
  - It can be used without using Condor on your cluster

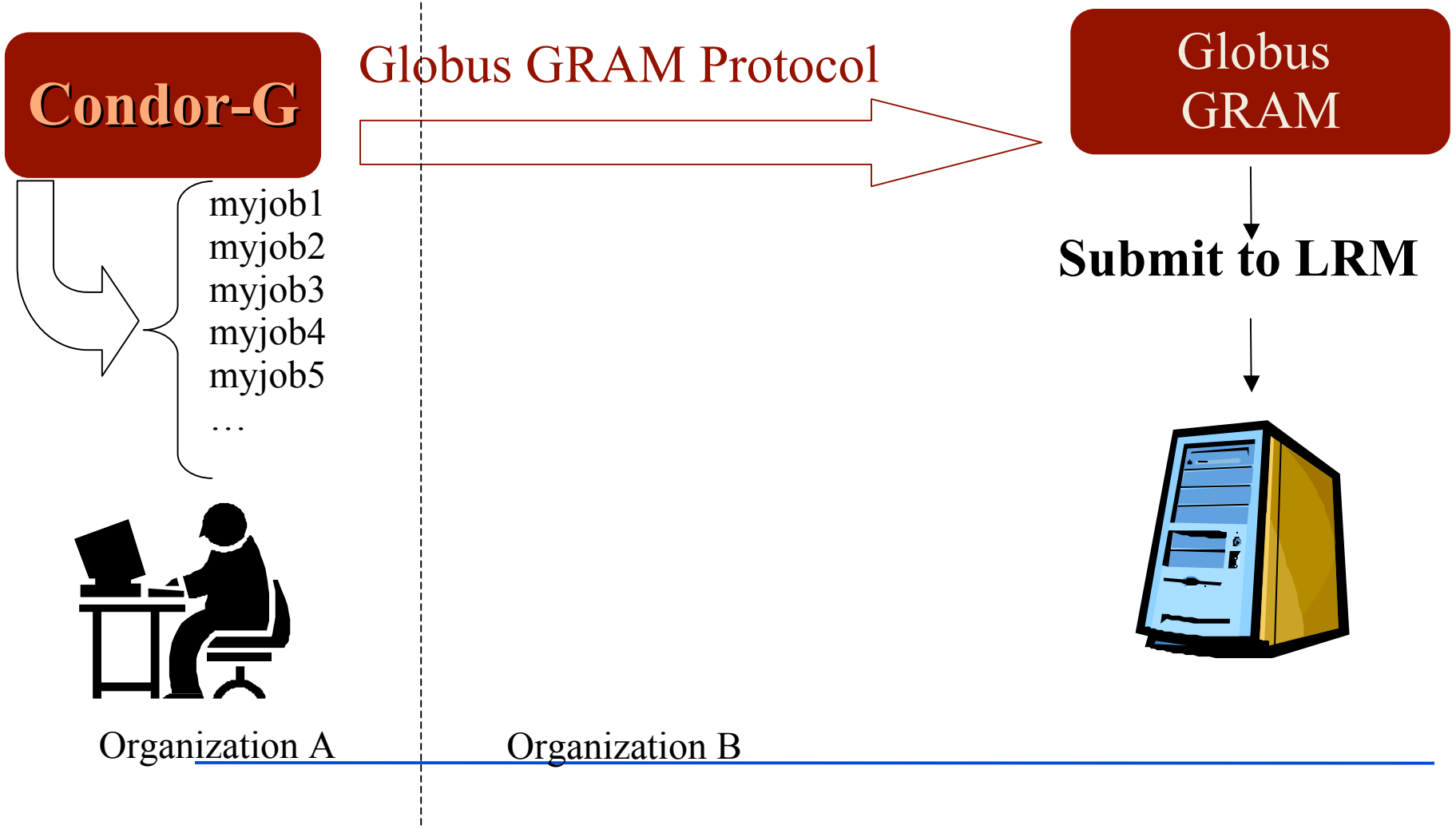
# Condor-G

- *Full-featured Task broker*

- Condor-G can manage thousands of jobs destined to run at distributed sites.
- It provides job monitoring, logging, notification, policy enforcement, fault tolerance, credential management, and it can handle complex job-interdependencies.
- allows the user to harness multi-domain resources as if they all belong to one personal domain.

- Condor-G is the job management part of Condor. Condor-G lets you submit jobs into a queue, have a log detailing the life cycle of your jobs, manage your input and output files, along with everything else you expect from a job queuing system

# Remote Resource Access: Condor-G + Globus + Condor



# Why bother with Condor-G?

- Globus has command-line tools, right?
- Condor-G provides extra features:
  - Access to multiple types of grid sites
  - A reliable queue for your jobs
  - Job throttling

# 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
  - **Grid**: run jobs on the grid
  - **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 grid***

# 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!
executable=/sw/national_grids/primetestarguments=143
output=results.output
error=results.error
log=results.log
notification=never
universe=grid
grid_resource=gt2
  workshop1.ci.uchicago.edu/jobmanager-forkqueue
```

# 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

# 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

  $\Rightarrow$ WORKFLOWS
- And also, we would like the workflow to be managed even in the face of failures

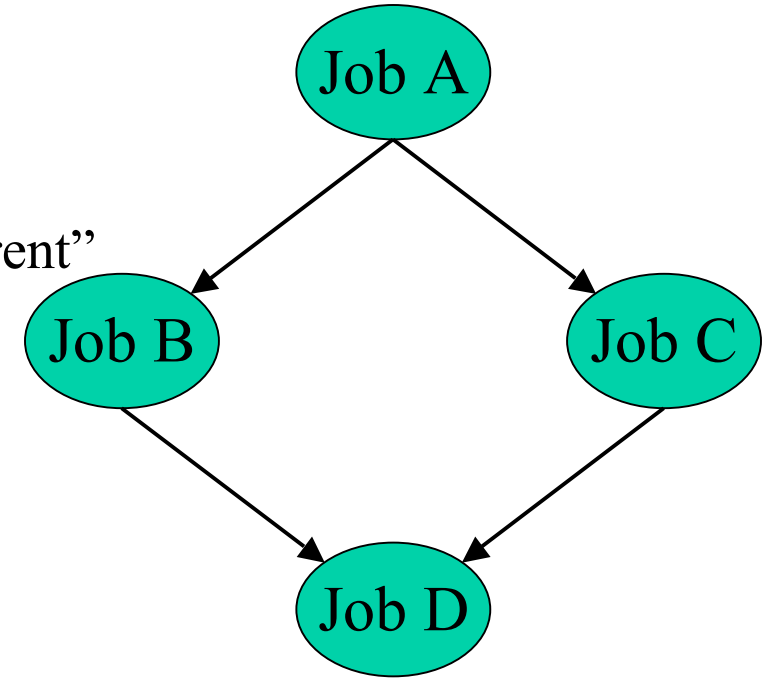
# DAGMan

- **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 –

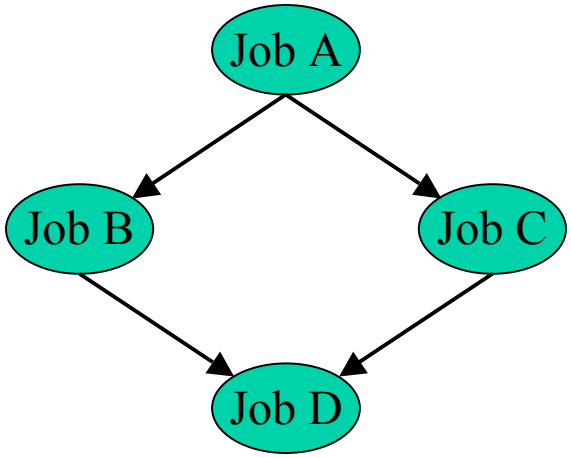
  as long as there are no loops!



# 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
```

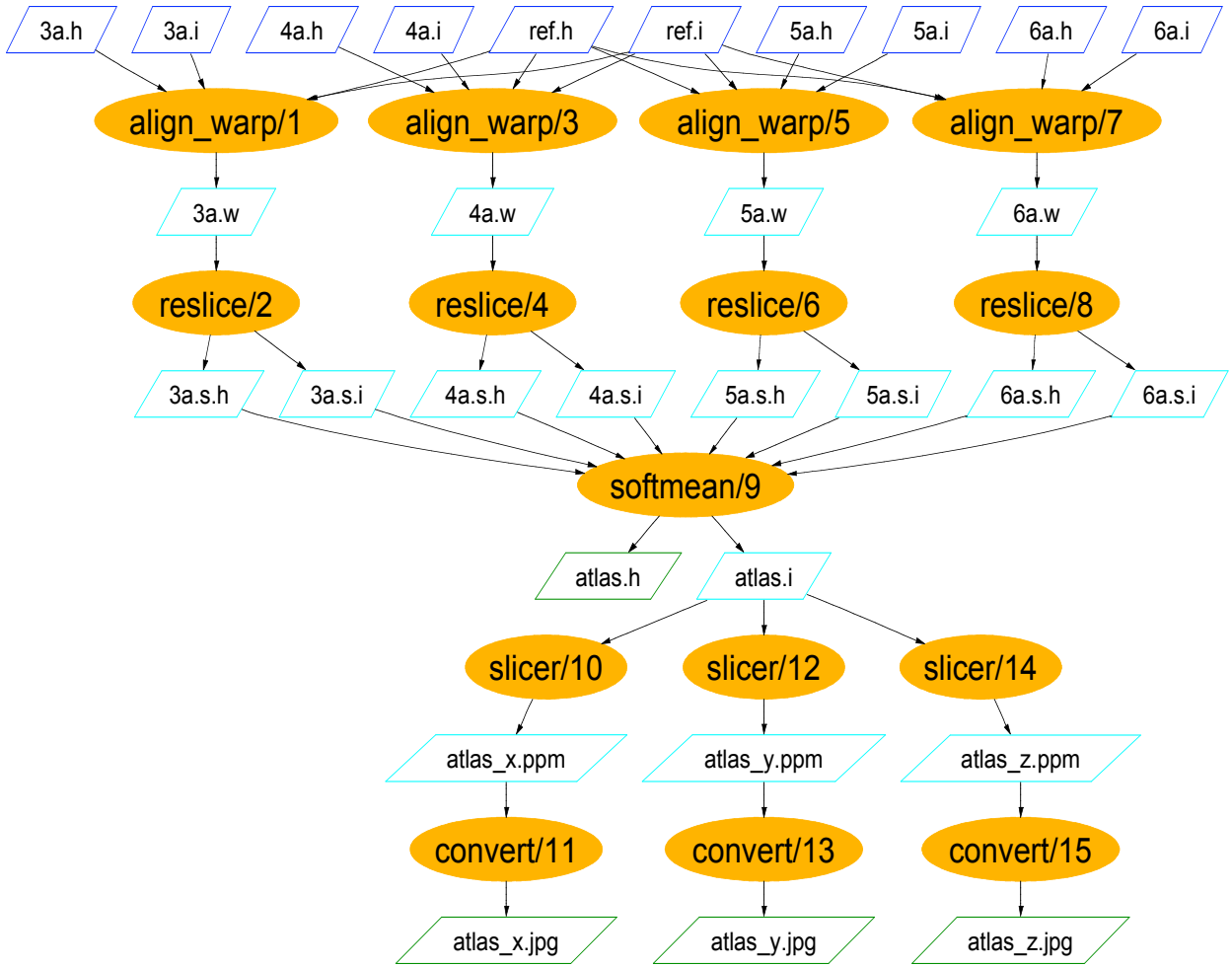
- condor_submit_dag submits a Scheduler Universe Job with DAGMan as the executable.
- Thus the DAGMan daemon itself runs as a Condor job, so you don't have to baby-sit it.

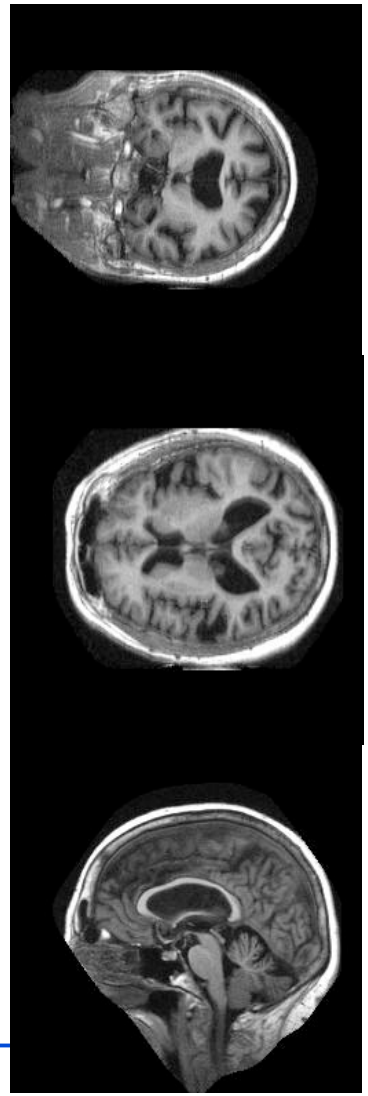
# Why care about DAGMan?

- Submit and forget
  - You can submit a large workflow and let DAGMan run it
- Fault tolerance
  - DAGMan can help recover from many common failures
  - Example: You can retry jobs automatically if they fail

# Grid Workflow

# A typical workflow pattern in image analysis runs many filtering apps (fMRI - functional magnetic resonance imaging)



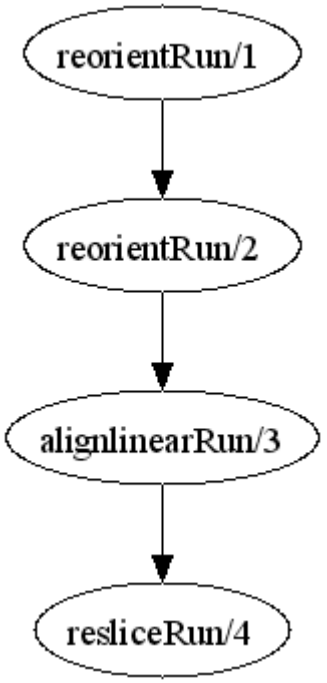
***Workflow courtesy James Dobson, Dartmouth Brain Imaging Center***

# Workflow management systems

- Orchestration of *many* resources over long time periods
  - Very complex to do manually - workflow automates this effort
- Enables restart of long running scripts
- Write scripts in a manner that's location-independent: run anywhere
  - Higher level of abstraction gives increased portability of the workflow script (over ad-hoc scripting)

# fMRI Example Workflow

```
(Run resliced) reslice_wf ( Run r)
{
    Run yR = reorientRun( r , "y", "n" );
    Run roR  = reorientRun( yR , "x", "n" );
    Volume std = roR.v[1];
    AirVector roAirVec =
        alignlinearRun(std, roR, 12, 1000, 1000, "81 3 3");
    resliced = resliceRun( roR, roAirVec, "-o", "-k");
}
```




```
(Run or) reorientRun (Run ir, string direction, string overwrite)
{
      foreach Volume iv, i in ir.v {
            or.v[i] = reorient (iv, direction, overwrite);
      }
}
```

# 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.

# Data Management

\

# High-performance tools needed to solve several data problems.

- The huge raw volume of data:
  - Storing it
  - Moving it
  - Measured in terabytes, petabytes, and further …
- The huge number of filenames:
  - $10^{12}$ filenames is expected soon
  - Collection of $10^{12}$ of anything is a lot to handle efficiently
- How to find the data

# Data Questions on the Grid

- Where are the files I want?
- How to move data/files to where I want?

# Data management services provide the mechanisms to find, move and share data

- GridFTP
  - Fast, Flexible, Secure, Ubiquitous data transport
  - Often embedded in higher level services
- RFT
  - Reliable file transfer service using GridFTP
- Replica Location Service (RLS)
  - Tracks multiple copies of data for speed and reliability
- Storage Resource Manager (SRM)
  - Manages storage space allocation, aggregation, and transfer
- Metadata management services are evolving

# Data Management

- Moving data - scenarios:
  - Store long term in appropriate places (e.g., tape silos)
  - Move input to where your job is running
  - Move output data from where your job ran to where you need it (eg. your workstation, long term storage)

# GridFTP

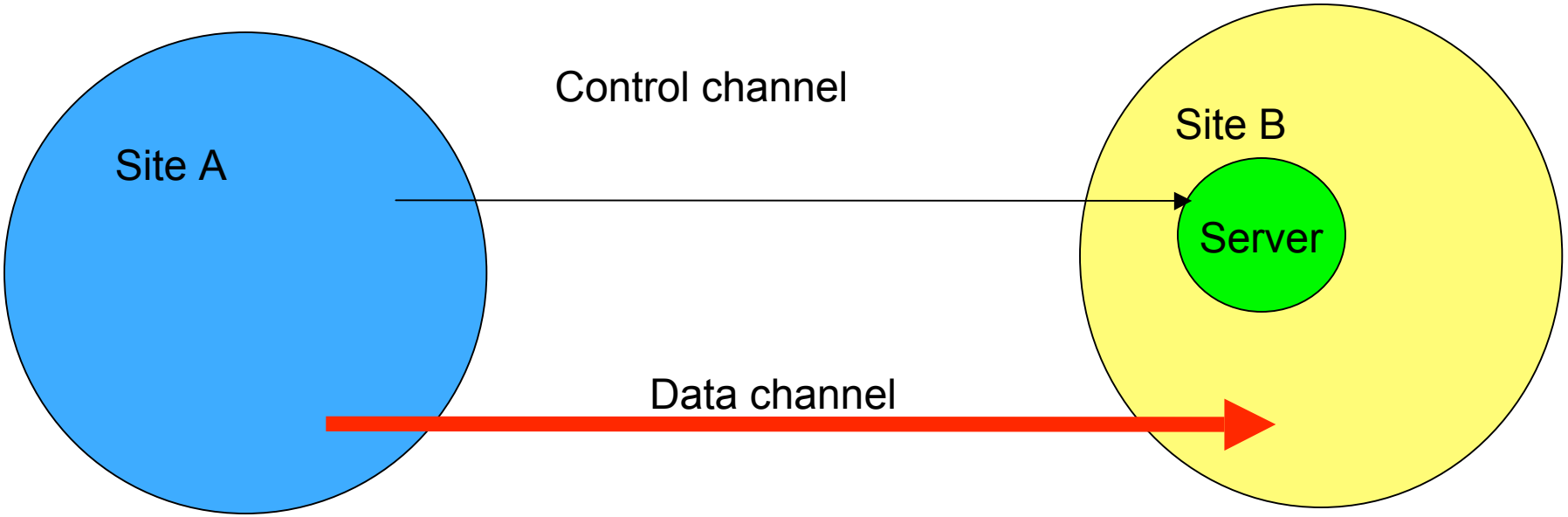
- high performance, secure, and reliable data transfer protocol based on the standard FTP
  - http://www.ogf.org/documents/GFD.20.pdf
- Extensions include
  - Strong authentication, encryption via Globus GSI
  - Multiple data channels for parallel transfers
  - Third-party transfers
  - Tunable network & I/O parameters
  - Authenticated reusable channels
  - Server side processing, command pipelining

# Basic Definitions

- **Control Channel**
  - TCP link over which **commands** and **responses** flow
  - Low bandwidth; encrypted and integrity protected by default
- **Data Channel**
  - Communication link(s) over which the actual **data** of interest flows
  - High Bandwidth; authenticated by default; encryption and integrity protection optional

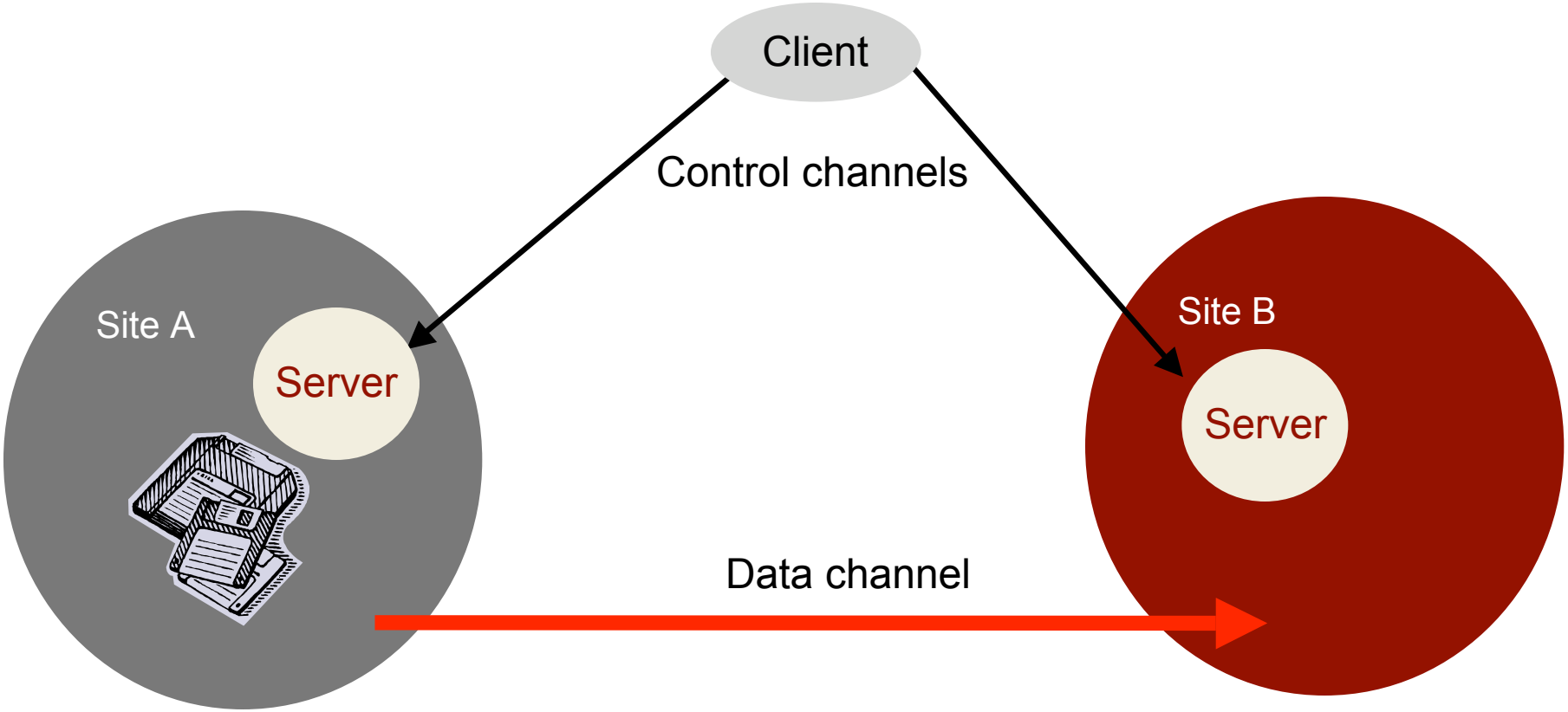
# A file transfer with GridFTP

- Control channel can go either way
  - Depends on which end is client, which end is server
- Data channel is still in same direction



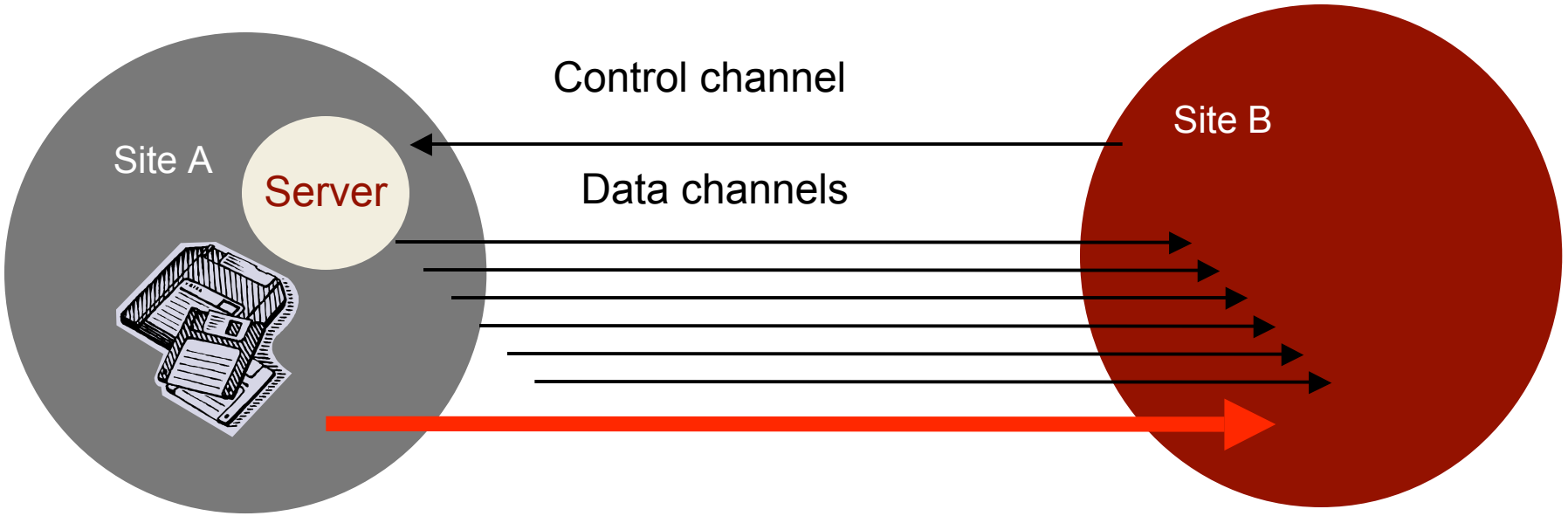
# Third party transfer

- File transfer without data flowing through the client.
- Controller can be separate from src/dest
- Useful for moving data from storage to compute



# Going fast – parallel streams

- Use several data channels



# GridFTP usage

- **globus-url-copy**

- Convensions on URL formats:

  - `file:///home/YOURLOGIN/dataex/largefile`
    - a file called **largefile** on the local file system, in directory **/home/YOURLOGIN/dataex/**

  - `gsiftp://osg-edu.cs.wisc.edu/scratch/YOURLOGIN/`
    - a directory accessible via gsiftp on the host called `osg-edu.cs.wisc.edu` in directory **/scratch/YOURLOGIN**.

# GridFTP examples

- **globus-url-copy**
  **file:///home/YOURLOGIN/dataex/myfile**
  **gsiftp://osg-edu.cs.wisc.edu/nfs/osgedu/YOURLOGIN/ex1**

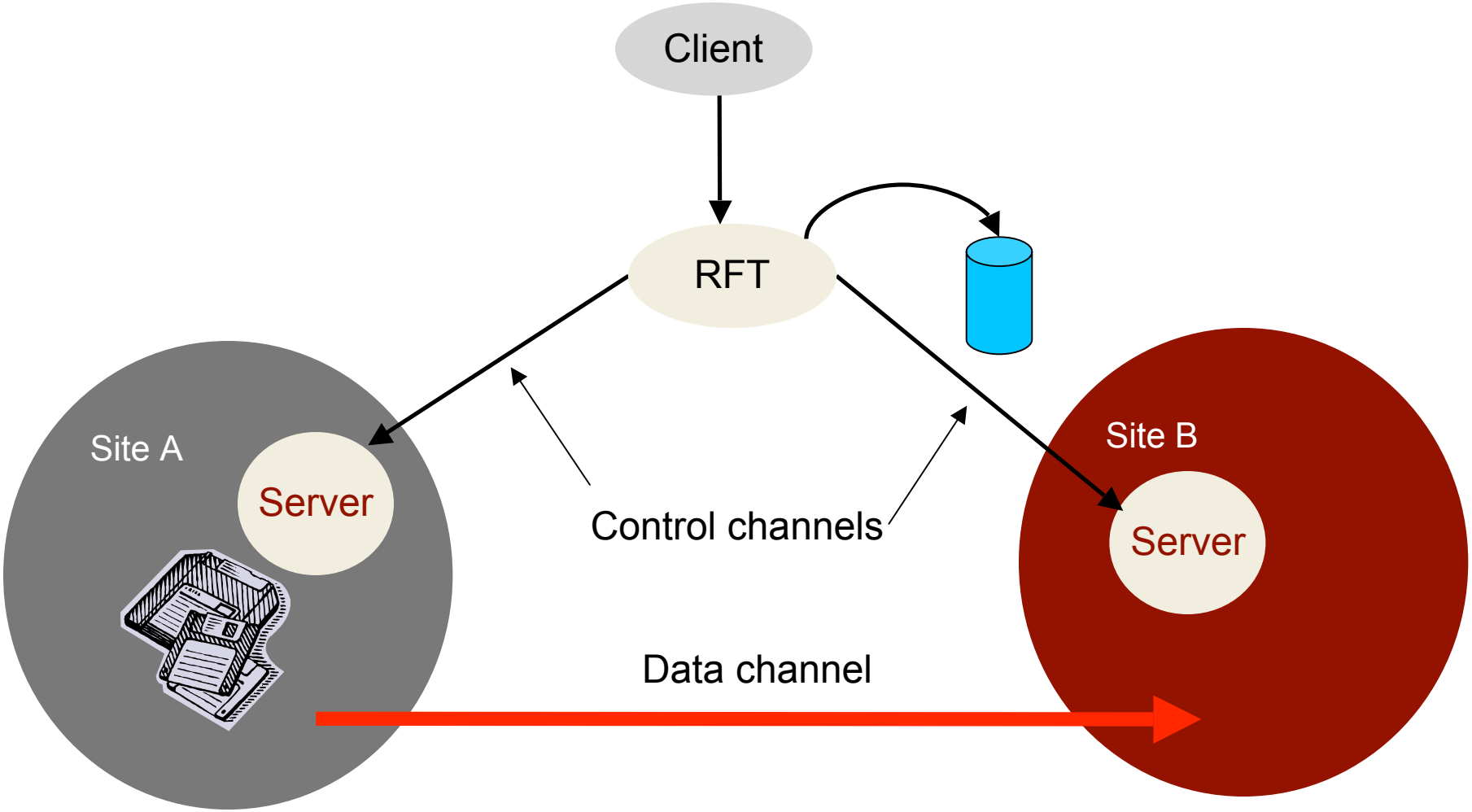
- **globus-url-copy**

**gsiftp://osg-edu.cs.wisc.edu/nfs/osgedu/YOURLOGIN/ex2**
**gsiftp://tp-osg.ci.uchicago.edu/YOURLOGIN/ex3**

# RFT = Reliable file transfer

- protocol that provides reliability and fault tolerance for file transfers
  - Part of the Globus Toolkit
- RFT acts as a client to GridFTP, providing management of a large number of transfer jobs (same as Condor to GRAM)
- RFT can
  - keep track of the state of each job
  - run several transfers at once
  - deal with connection failure, network failure, failure of any of the servers involved.

# RFT


Client
RFT
Site A
Server
Control channels
Site B
Server
Data channel

# RFT example

- **Use the rft command with a .xfr file**
- **cp /soft/globus-4.0.3-r1/share/globus_wsrf_rft_client/transfer.xfr rft.xfr**
- **Edit rft.xfr to match your needs**
- **rft -h terminable.ci.uchicago.edu -f ./rft.xfr**

# Grids replicate data files for faster access

- Effective use of the grid resources – more parallelism
- Each *logical* file can have multiple *physical* copies
- Avoids single points of failure
- Manual or automatic replication
  - Automatic replication considers the demand for a file, transfer bandwidth, etc.

# File catalogues tell you where the data is

- File Catalog Services
  - Replica Location Service (RLS)

- Requirements
  - Abstract the logical file name (LFN) for a physical file
  - maintain the mappings between the LFNs and the PFNs *(physical file names)*
  - Maintain the location information of a file

# RLS -Replica Location Service

- RLS
  - component of the data grid architecture (Globus component)
  - It provides access to mapping information from logical names to physical names of items

  Goal:

  **reduce access latency, improve data locality, improve robustness, scalability and performance for distributed applications**

- RLS produces replica catalogs (LRCs), which represent mappings between logical and physical files scattered across the storage system.
  - For better performance, the LRC can be indexed.

# RLS -Replica Location Service

- RLS maps logical filenames to physical filenames.
- Logical Filenames (LFN)
  - Names a file with interesting data in it
  - Doesn't refer to location (which host, or where in a host)
- Physical Filenames (PFN)
  - Refers to a file on some filesystem somewhere
  - Often use `gsiftp://` URLs to specify
- Two RLS catalogs:
  - Local Replica Catalog (LRC) and
  - Replica Location Index (RLI)

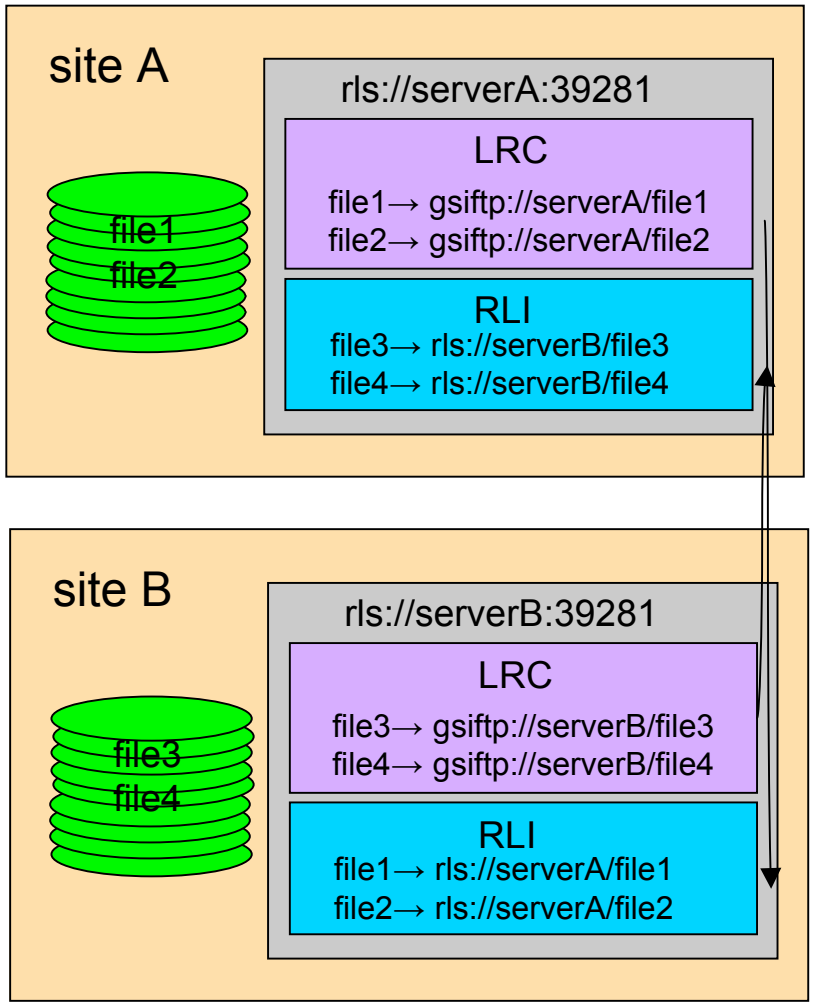
# Local Replica Catalog (LRC)

- stores mappings from LFNs to PFNs.
- Interaction:
  - *Q:* Where can I get filename ‘experiment_result_1’?
  - *A:* You can get it from gsiftp://gridlab1.ci.uchicago.edu/home/benc/r.txt
- Undesirable to have one of these for whole grid
  - Lots of data
  - Single point of failure

# Replica Location Index (RLI)

- stores mappings from LFNs to LRCs.
- Interaction:
  - *Q:* Who can tell me about filename ‘experiment_result_1’.
  - *A:* You can get more info from the LRC at gridlab1
  - (Then go to ask that LRC for more info)
- Failure of one RLI or LRC doesn’t break everything
- RLI stores reduced set of information, so can cope with many more mappings

# Globus RLS


site A
rls://serverA:39281
LRC
file1→ gsiftp://serverA/file1
file2→ gsiftp://serverA/file2
RLI
file3→ rls://serverB/file3
file4→ rls://serverB/file4
file1
file2
site B
rls://serverB:39281
LRC
file3→ gsiftp://serverB/file3
file4→ gsiftp://serverB/file4
RLI
file1→ rls://serverA/file1
file2→ rls://serverA/file2
file3
file4

# Globus RLS

- Quick Review
  - LFN → logical filename (think of as simple filename)
  - PFN → physical filename (think of as a URL)
  - LRC → your local catalog of maps from LFNs to PFNs
    - H-R-792845521-16.gwf → gsiftp://dataserver.phys.uwm.edu/LIGO/H-R-792845521-16.gwf
  - RLI → your local catalog of maps from LFNs to LRCs
    - H-R-792845521-16.gwf → LRCs at MIT, PSU, Caltech, and UW-M
  - LRCs inform RLIs about mappings known
- Can query for files is a 2-step process: find files on your Grid by
  - querying RLI(s) to get LRC(s)
  - then query LRC(s) to get URL(s)

# RLS command line tools

- **globus-rls-admin**
  - administrative tasks
    - ping server
    - connect RLIs and LRCs together
- **globus-rls-cli**
  - end user tasks
    - query LRC and RLI
    - add mappings to LRC

# Globus RLS: Client Perspective

Two ways for clients to interact with RLS Server

- **globus-rls-cli** simple command-line tool
  - query
  - create new mappings

# Globus-rls-cli - query

Simple query to LRC to find a PFN for LFN

- Note more then one PFN may be returned

```
$ globus-rls-cli query lrc lfn some-file.jpg rls://dataserver:39281

 some-file.jpg : file://localhost/netdata/s001/S1/R/H/714023808-714029599/some-file.jpg
 some-file.jpg : file://medusa-slave001.medusa.phys.uwm.edu/data/S1/R/H/714023808-
    714029599/some-file.jpg
 some-file.jpg :
    gsiftp://dataserver.phys.uwm.edu:15000/data/gsiftp_root/cluster_storage/data/s001/S1
    /R/H/714023808-714029599/some-file.jpg
```

- Server and client sane if LFN not found

```
$ globus-rls-cli query lrc lfn foo rls://dataserver
LFN doesn't exist: foo
$ echo $?
1
```

# Globus-rls-cli - wildcards

## Wildcard searches of LRC supported

- probably a good idea to quote LFN wildcard expression

```
$ globus-rls-cli query wildcard lrc lfn H-R-7140242*-16.gwf
    rls://dataserver:39281
  H-R-714024208-16.gwf:
    gsiftp://dataserver.phys.uwm.edu:15000/data/gsiftp_root/cluster_storage/data/s
    001/S1/R/H/714023808-714029599/H-R-714024208-16.gwf
  H-R-714024224-16.gwf:
    gsiftp://dataserver.phys.uwm.edu:15000/data/gsiftp_root/cluster_storage/data/s
    001/S1/R/H/714023808-714029599/H-R-714024224-16.gwf
```

# Globus-rls-cli - bulk

Bulk queries also supported

- obtain PFNs for more then one LFN at a time

```
$ globus-rls-cli bulk query lrc lfn H-R-714024224-16.gwf
   H-R-714024320-16.gwf rls://dataserver
   H-R-714024320-16.gwf:
    gsiftp://dataserver.phys.uwm.edu:15000/data/gsiftp_root/
    cluster_storage/data/s001/S1/R/H/714023808-714029599/H-
    R-714024320-16.gwf
   H-R-714024224-16.gwf:
    gsiftp://dataserver.phys.uwm.edu:15000/data/gsiftp_root/
    cluster_storage/data/s001/S1/R/H/714023808-714029599/H-
    R-714024224-16.gwf
```

# Globus-rls-cli - RLI query

Simple query to RLI to locate a LFN -> LRC mapping

- then query that LRC for the PFN

```
$ globus-rls-cli query rli lfn example-file.gwf rls://dataserver

  example-file.gwf: rls://ldas-cit.ligo.caltech.edu:39281

$ globus-rls-cli query lrc lfn example-file.gwf rls://ldas-
   cit.ligo.caltech.edu:39281

  example-file: gsiftp://ldas-
   cit.ligo.caltech.edu:15000/archive/S1/L0/LHO/H-R-7140/H-R-
   714024224-16.gwf
```

# Globus-rls-cli - RLI wildcard and bulk queries

- Bulk queries to RLI also supported

```
$ globus-rls-cli bulk query rli lfn H-R-714024224-16.gwf H-R-714024320-
   16.gwf rls://dataserver
  H-R-714024320-16.gwf: rls://ldas-cit.ligo.caltech.edu:39281
  H-R-714024224-16.gwf: rls://ldas-cit.ligo.caltech.edu:39281
```

- Wildcard queries to RLI may not be supported!
  - no wildcards when using Bloom filter updates

```
$ globus-rls-cli query wildcard rli lfn "H-R-7140242*-16.gwf"
   rls://dataserver
Operation is unsupported: Wildcard searches with Bloom filters
```

# Globus-rls-cli - create, add, delete

Create new LFN → PFN mappings

- use **`create`** to create 1st mapping for a LFN

```
$ globus-rls-cli create file1 gsiftp://dataserver/file1
   rls://dataserver
```

- use **`add`** to add more mappings for a LFN

```
$ globus-rls-cli add file1 file://dataserver/file1
   rls://dataserver
```

- use **`delete`** to remove a mapping for a LFN
  - when last mapping is deleted for a LFN the LFN is also deleted
  - cannot have LFN in LRC without a mapping

```
$ globus-rls-cli delete file1 file://file1 rls://dataserver
```

# Related Work - SRMs

- Storage Resource Manager (SRM)
  - Equivalent of a job scheduler for storage;  allocates space, makes sure it doesn't get swapped out before you are done (pinning); handles staging to and from tape
  - http://sdm.lbl.gov/indexproj.php?ProjectID=SRM
  - Examples:
    - dCache
      - provide a system for storing and retrieving huge amounts of data, distributed among a large number of heterogenous server nodes, under a single virtual filesystem tree with a variety of standard access methods.
      - http://www.dcache.org/
    - BeSTMan (Berkeley Storage Manager)
      - http://datagrid.lbl.gov/bestman/

# OSG & Data management

- OSG relies on GridFTP protocol for the raw transport of the data using Globus GridFTP in all cases except where interfaces to storage management systems (rather than file systems) dictate individual implementations.

- OSG supports the SRM interface to storage resources to enable management of space and data transfers to prevent unexpected errors due to running out of space, to prevent overload of the GridFTP services, and to provide capabilities for pre-staging, pinning and retention of the data files.
- OSG currently provides reference implementations of two storage systems the BeStMan and dCache