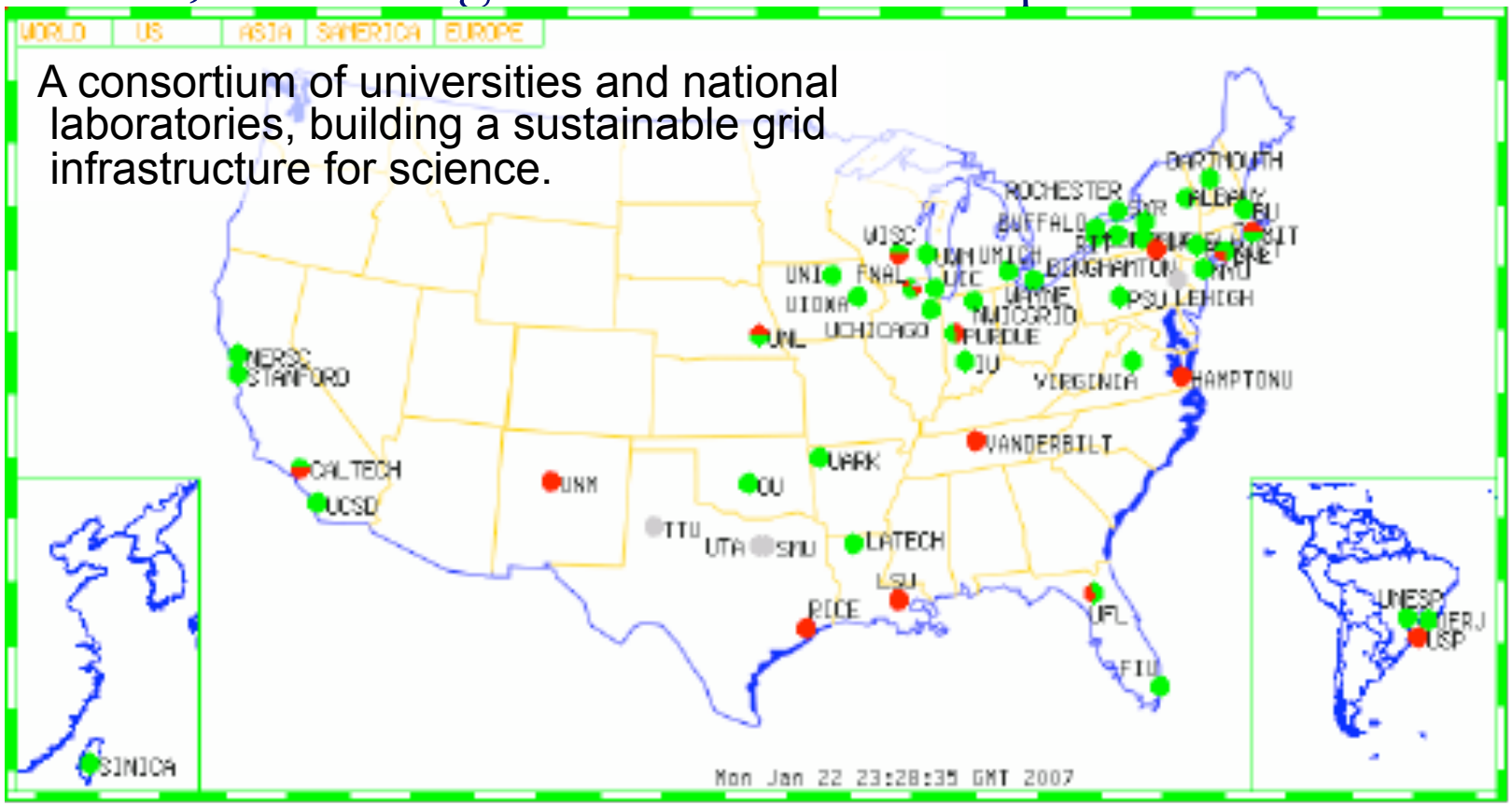
Introduction to Grid Computing Tutorial Outline

I. Motivation and Grid Architecture
II. Grid Examples from Life Sciences
III. Grid Security
IV. Job Management: Running Applications
V. Data Management
VI. Open Science Grid and TeraGrid
VII. Workflow on the Grid
VIII. Next steps: Learning more, getting started

Nationwide Grid Cyberinfrastructure

Public, general purpose grids for multiple disciplines

Open Science Grid (OSG) provides shared computing resources, benefiting a broad set of disciplines

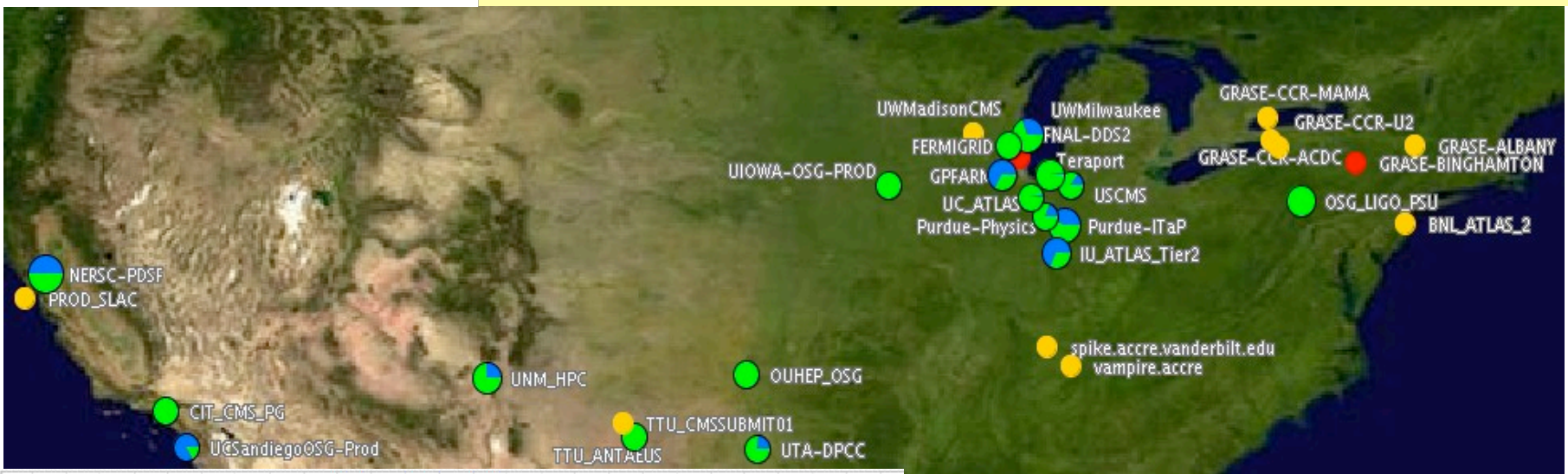


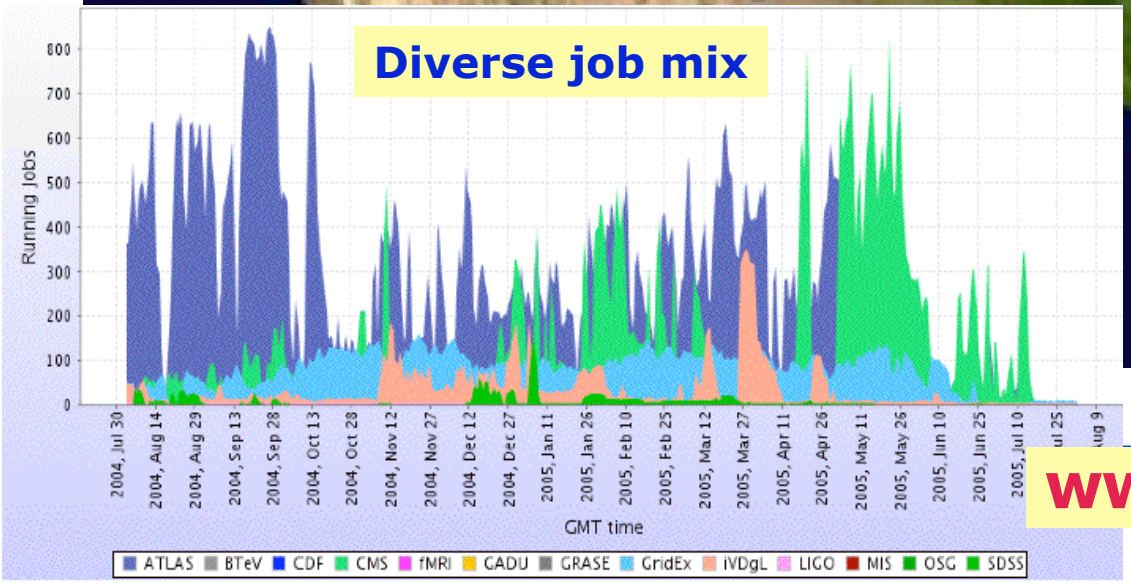
- OSG incorporates advanced networking and focuses on general services, operations, end-to-end performance
- Composed of a large number (>50 and growing) of shared computing facilities, or "sites"

http://www.opensciencegrid.org/

Open Science Grid

- 70 sites (20,000 CPUs) & growing
- 400 to >1000 concurrent jobs
- Many applications + CS experiments; includes long-running production operations

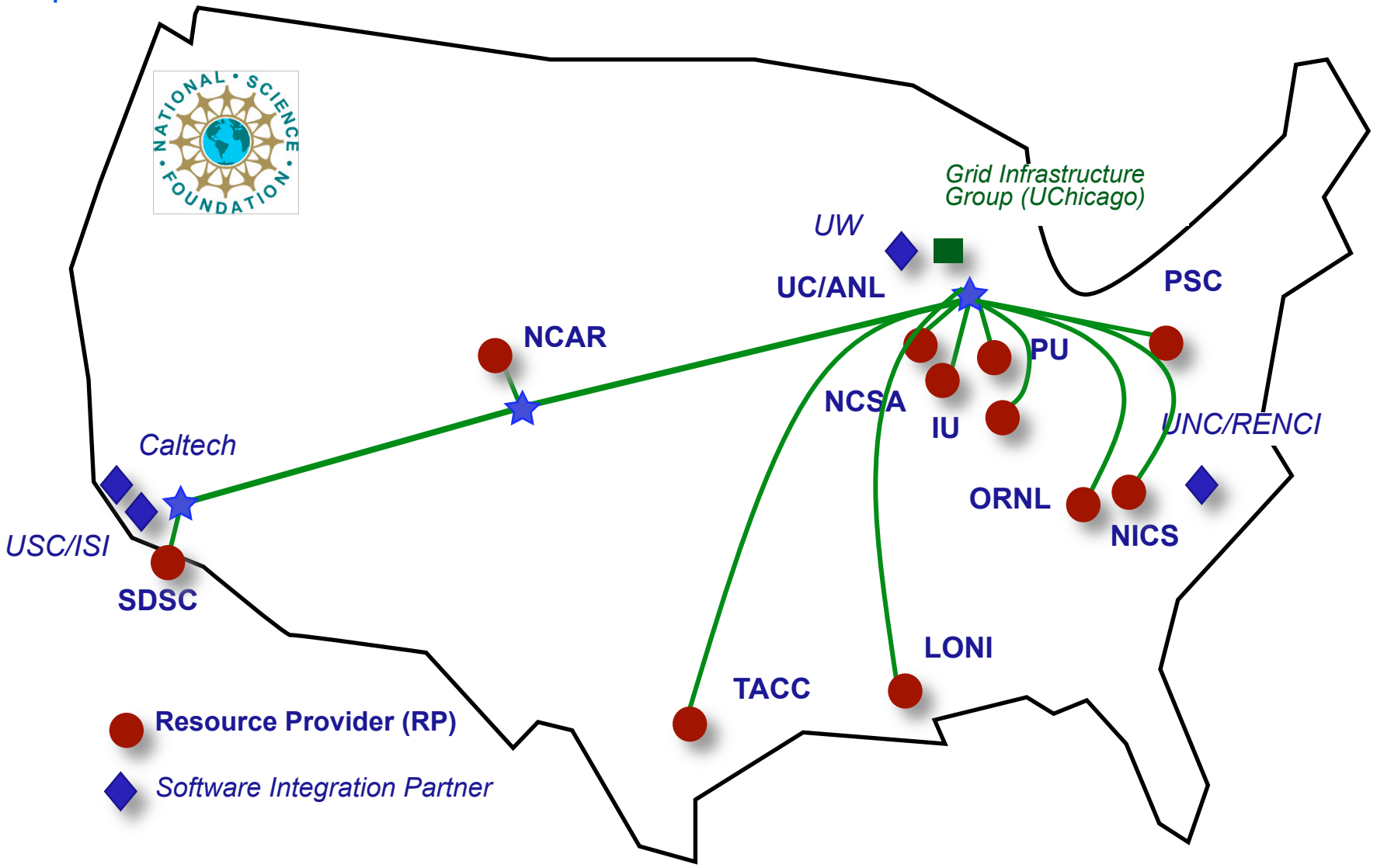




www.opensciencegrid.org

TeraGrid provides vast resources via a number of huge computing facilities.

11 TeraGrid Resource Providers



TeraGrid Resources and Services

- Computing - nearly a petaflop of computing power today and growing
  - 500 Tflop Ranger system at TACC
  - NICS (U Tenn) system to come on-line this year
  - Centralized help desk for all resource providers
- Remote visualization servers and software
- Data
  - Large-scale grid-enabled data storage facilities
  - Over 100 Scientific Data Collections
- Central allocations process
- Technical Support
  - Central point of contact for support of all systems
  - Advanced Support for TeraGrid Applications (ASTA)
  - Education and training events and resources
  - Over 20 Science Gateways

Requesting Resource Allocations

- TeraGrid resources are provided for free to academic researchers and educators
- Development Allocations Committee (DAC) for start-up accounts up to 30,000 hours of time. Requests processed in two weeks – for start-up and courses
- Medium Resource Allocations Committee (MRAC) for requests of up to 500,000 hours of time. Reviewed four times a year
- Large Resource Allocations Committee (LRAC) for requests of over 500,000 hours of time. Reviewed twice a year

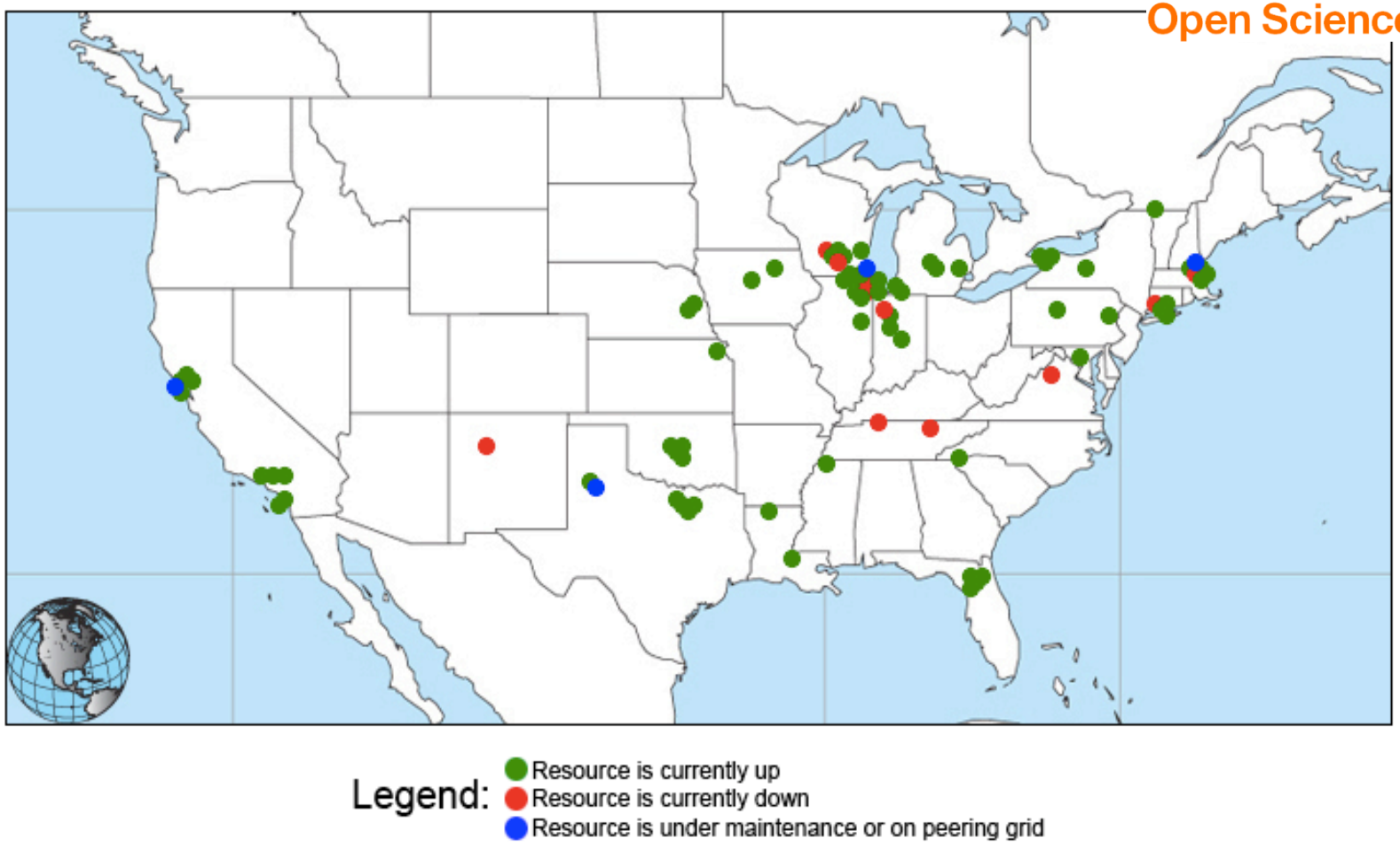
Snapshot of the OSG

- Research Participation
  - Majority from physics : Tevatron, LHC, STAR, LIGO.
  - Use by 10 other (smaller) research groups.
  - >10 software development groups contribute to commonly supported software stack (Virtual Data Toolkit).
- Core Project
  - 5 years funding (conditional on review) at total of $30M.
  - 35 FTE effort across 16 institutions
  - Full time Project Manager.
  - Leverages equivalent effort in contributions.
- Computational resources accessible at
  - 5 DOE Labs - BNL, Fermilab, NERSC, ORNL, SLAC.
  - 65 Universities.
  - 5 partner campus/regional grids.

OSG Resources and Usage

- Accessible resources
  - > 43,000 cores,
  - 6 Petabytes disk cache
  - 10 Petabytes tape storage
  - 75 production sites (Grows by 10-20 per year)
- Usage
  - 15,000 CPU WallClock days/day
  - 1 petabyte data distributed/month.
  - 100,000 application jobs/day.
  - 20% cycles through resource sharing, opportunistic use.

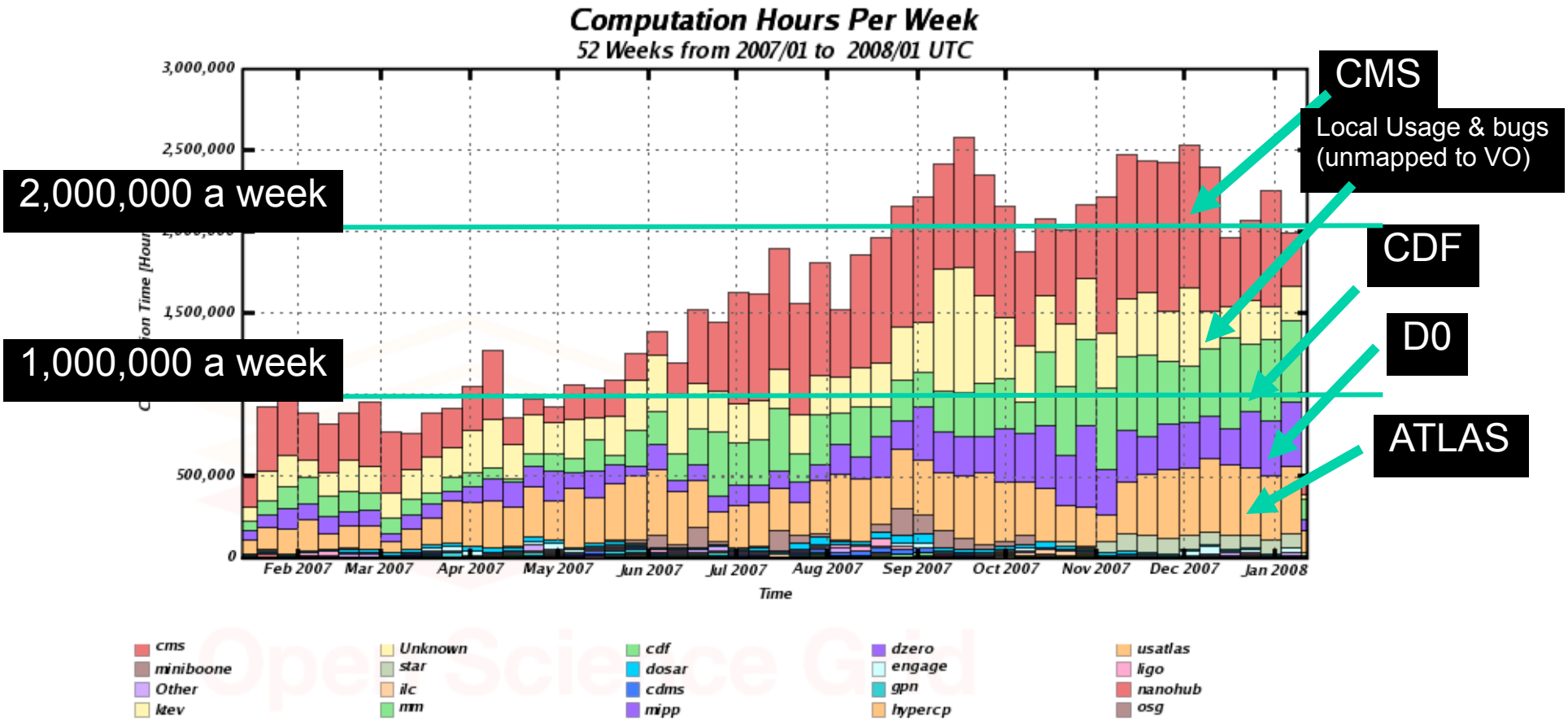
Map of OSG Sites



International sites: Brazil (3), Mexico (2), Tawain (2), UK (1)

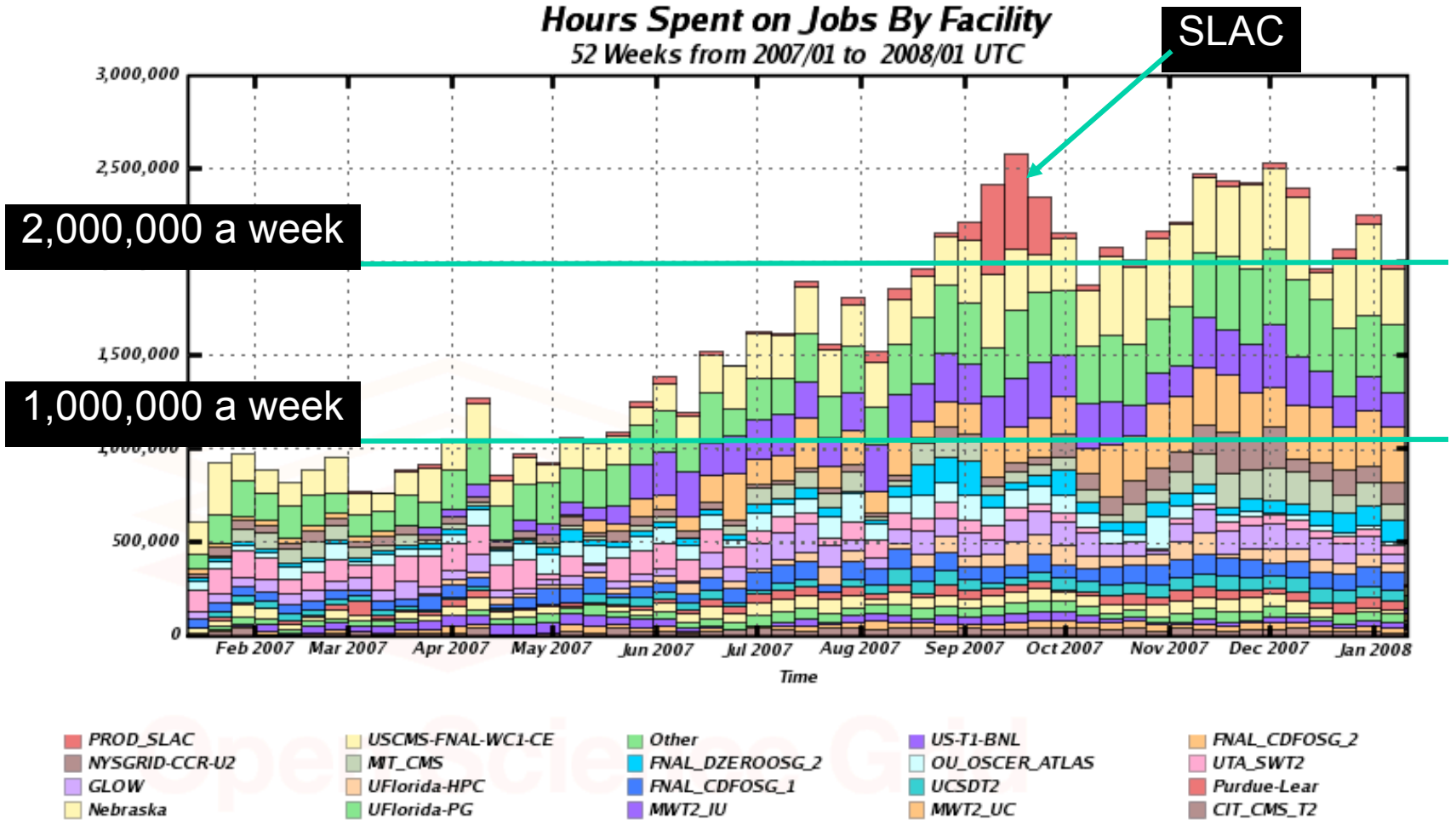
OSG Use by Community

Open Science Grid



OSG Use by Site

Open Science Grid



To efficiently use a Grid, you must locate and monitor its resources.

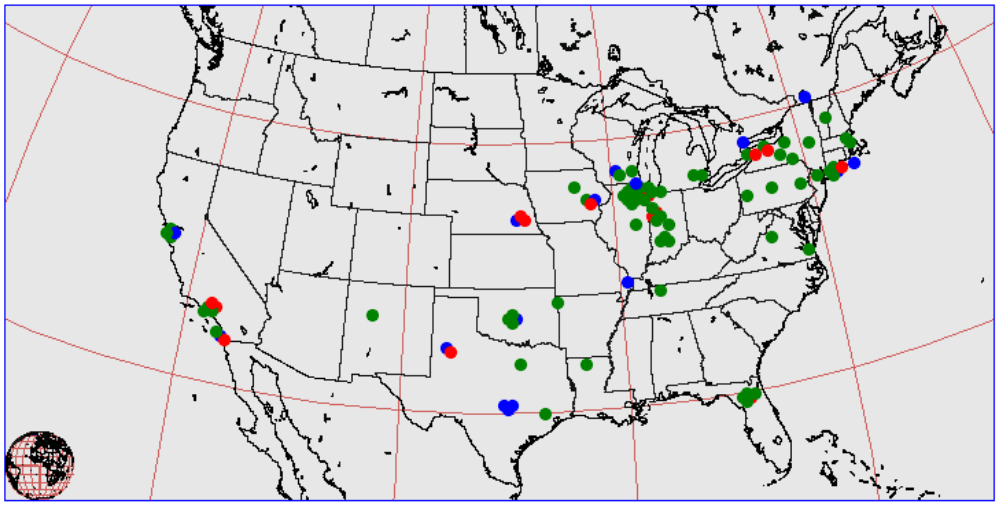
- Check the availability of different grid sites
- Discover different grid services
- Check the status of “jobs”
- Make better scheduling decisions with information maintained on the “health” of sites

OSG Resource Selection Service: VORS

Virtual Organization Resource Selector

All	OSG	TeraGrid	EGEE	OSG-ITB

Open Science Grid



Virtual Organization Selection

All	CDF	CMS	CompBioGrid	DES	DOSAR	DZero	Engage	Fermilab	fMRI	GADU
	geant4	GLOW	GPN	GRASE	GridChem	GridEx	GROW	i2u2	iVDGL	LIGO
mariachi	MIS	nanoHUB	NWICG	Ops	OSG	OSGEDU	SDSS	STAR	USATLAS	

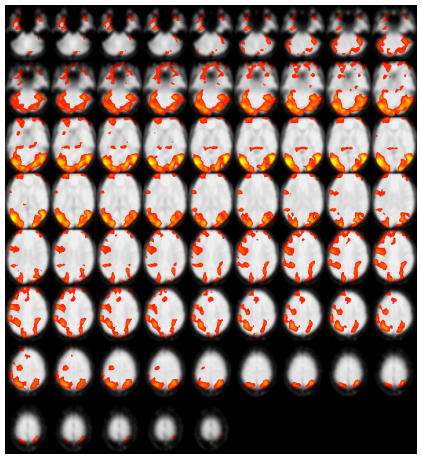
Resources

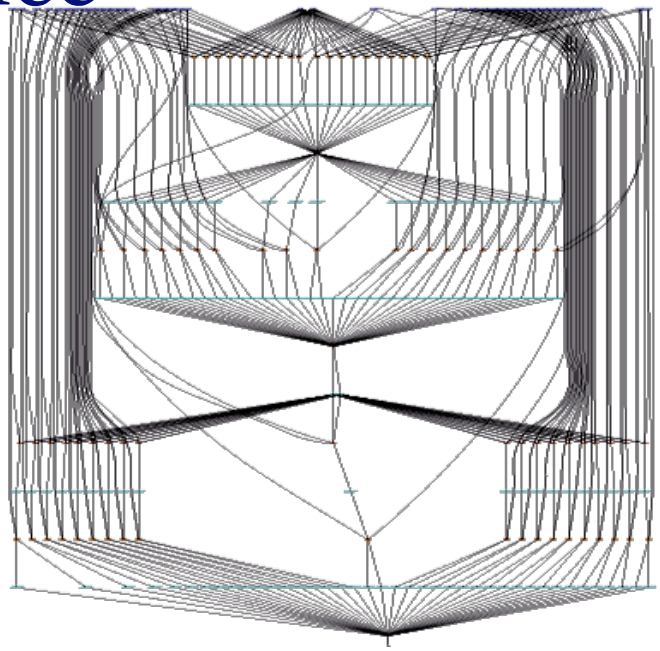
Name	Gatekeeper	Type	Grid	Status	Last Test Date
BNL_ATLAS_1	gridgk01.racf.bnl.gov:2119	compute	OSG	PASS	2006-12-08 14:57:13
BNL_ATLAS_2	gridgk02.racf.bnl.gov:2119	compute	OSG	PASS	2006-12-08 14:58:43
BU_ATLAS_Tier2	atlas.bu.edu:2119	compute	OSG	PASS	2006-12-08 15:00:44

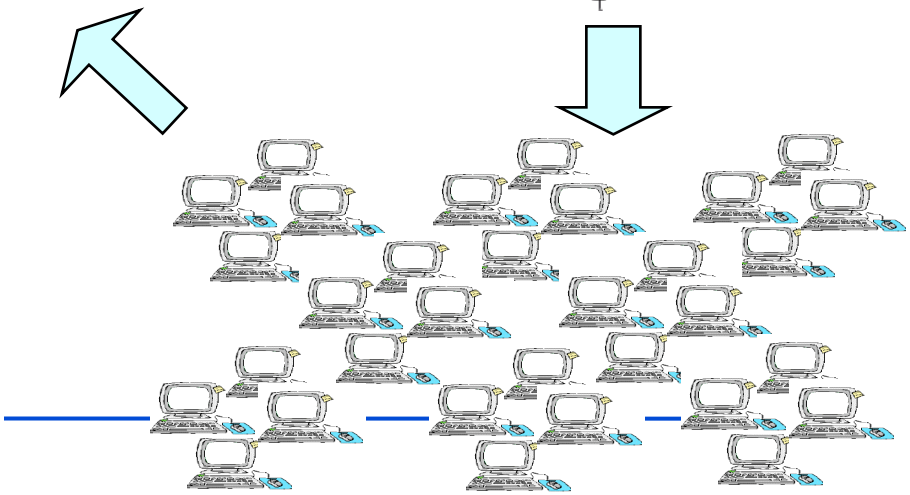
Introduction to Grid Computing Tutorial Outline

I. Motivation and Grid Architecture
II. Grid Examples from Life Sciences
III. Grid Security
IV. Job Management: Running Applications
V. Data Management
VI. Open Science Grid and TeraGrid
VII. Workflow on the Grid
VIII. Next steps: Learning more, getting started

Workflow Motivation: example from Neuroscience

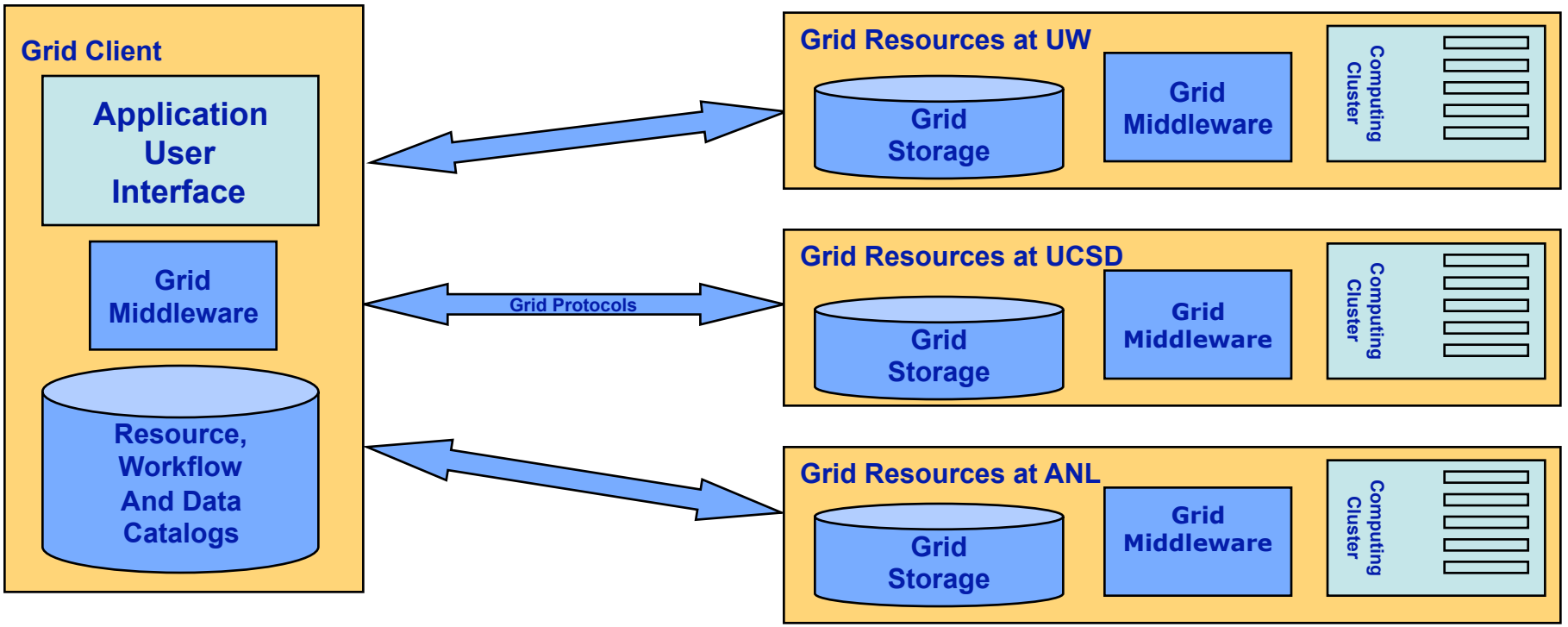






- Large fMRI datasets
  - 90,000 volumes / study
  - 100s of studies
- Wide range of analyses
  - Testing, production runs
  - Data mining
  - Ensemble, Parameter studies

Target environment: Cluster and Grids
(distributed sets of clusters)



Running a uniform middleware stack:

- Security to control access and protect communication (GSI)
- Directory to locate grid sites and services: (VORS, MDS)
- Uniform interface to computing sites (GRAM)
- Facility to maintain and schedule queues of work (Condor-G)
- Fast and secure data set mover (GridFTP, RFT)
- Directory to track where datasets live (RLS)

Grid Workflow Systems

- DAGman – part of the Condor system
  - Manages dependencies among large numbers of tasks
  - Uses Condor-G to provide scheduling, reliability, restart
- Pegasus
  - Automates the management of data
  - Generates DAGman workflows
- Swift
  - Provides a higher-level scripting language
  - Abstracts data with a typing and mapping model
- Many service-oriented workflow systems
  - BPEL, Scufl (MyGrid), Triana

DAGMan

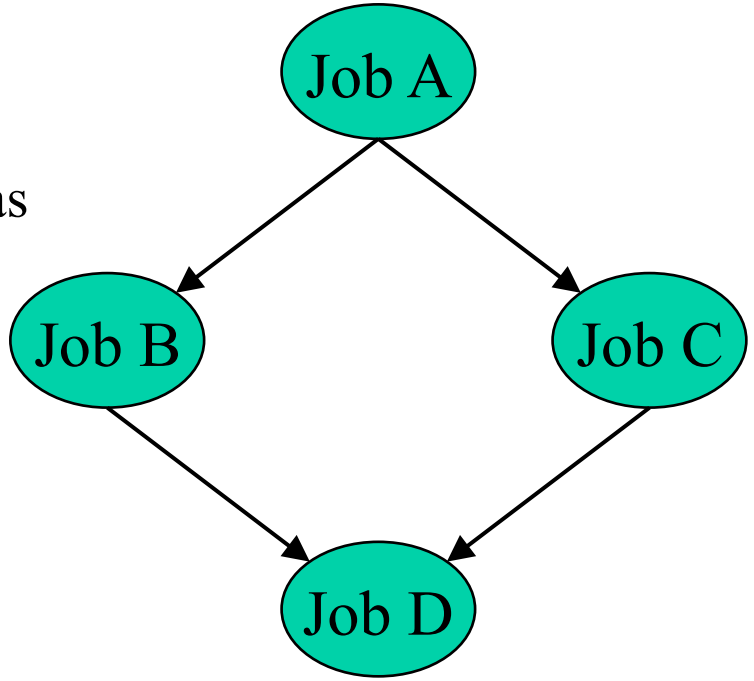
- **Directed Acyclic Graph Manager**

- 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?

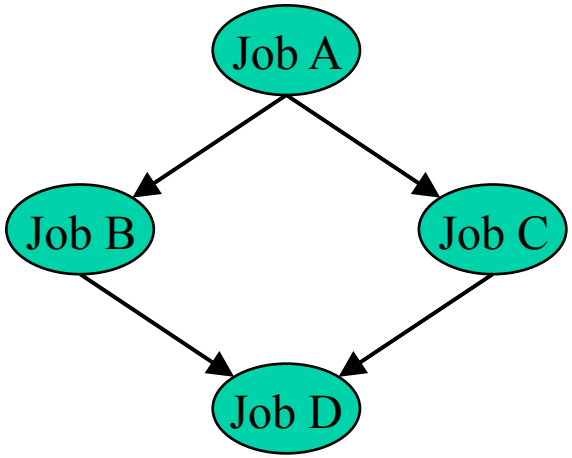
- A DAG is the data structure used by DAGMan to represent these dependencies.
- Each job is a "node" in the DAG.
- Each node can have any number of "parent" or "children" nodes – as long as there are no loops!



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

Managing Workflows with the Pegasus Workflow Management System

Ewa Deelman
USC Information Sciences Institute

A collaboration with Miron Livny and Kent Wenger, UW Madison
Funded by the NSF OCI SDCI project

USC VITERBI SCHOOL OF ENGINEERING

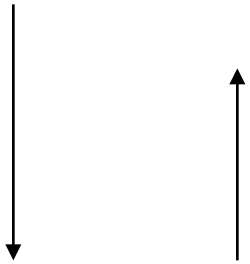
deelman@isi.edu http://pegasus.isi.edu

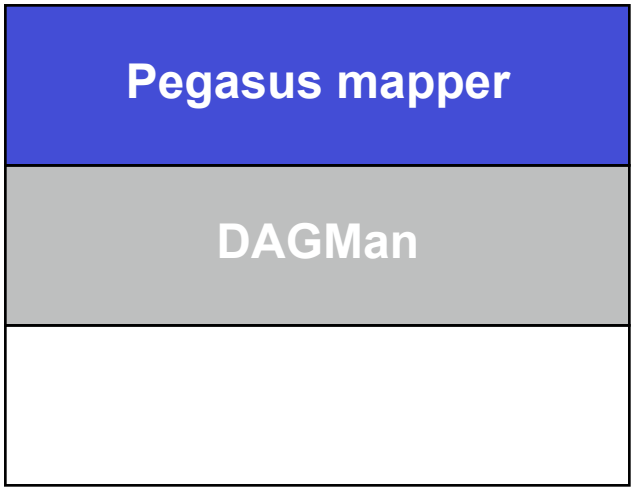
Pegasus: Planning for Execution in Grids

- Abstract Workflows - Pegasus input workflow description
  - workflow “high-level language”
  - only identifies the computations that a user wants to do
  - devoid of resource descriptions
  - devoid of data locations
- Pegasus
  - a workflow “compiler”
  - target language - DAGMan’s DAG and Condor submit files
  - transforms the workflow for performance and reliability
  - automatically locates physical locations for both workflow components and data
  - finds appropriate resources to execute the components
  - provides runtime provenance
- DAGMan
  - A workflow executor
  - Scalable and reliable execution of an executable workflow

Pegasus Workflow Management System

Abstract Workflow





Cyberinfrastructure: Local machine, cluster, Condor pool, OSG, TeraGrid

Pegasus DAX

```
<!-- part 1: list of all files used (may be empty) -->
  <filename file="f.input" link="input"/>
  <filename file="f.intermediate" link="input"/>
  <filename file="f.output" link="output"/>

<!-- part 2: definition of all jobs (at least one) -->
  <job id="ID000001" namespace="pegasus" name="preprocess" version="1.0" >
            <argument>-a top -T 6  -i <filename file="f.input"/>  -o <filename file="f.intermediate"/>
            </argument>
            <uses file="f.input" link="input" dontRegister="false" dontTransfer="false"/>
            <uses file="f.intermediate" link="output" dontRegister="true" dontTransfer="true"/>
  </job>
  <job id="ID000002" namespace="pegasus" name="analyze" version="1.0" >
            <argument>-a top -T 6  -i <filename file="f.intermediate"/>  -o <filename file="f.output"/>
            </argument>
            <uses file="f.input" link="input" dontRegister="false" dontTransfer="false"/>
            <uses file="f.intermediate" link="output" dontRegister="false" dontTransfer="false"/>
  </job>
```

- Resource-independent
- Portable across platforms

Comparing a DAX and a Condor DAG

DAX	DAG
Single XML File	One dag file and multiple submit files
Describes your workflow at a logical level	Describes your workflow in terms of physical files and paths
Site Independent	Site Specific
Captures just the computation that the user (you) want to do	Has auxillary jobs like data movement etc.

How to generate a DAX

- Write the XML directly
- Use the Pegasus Java API
- Use Wings for semantically rich workflow composition (http://www.isi.edu/ikcap/wings/)
- In the works python and perl APIs
- To come a Triana interface
- Prototype Kepler interface

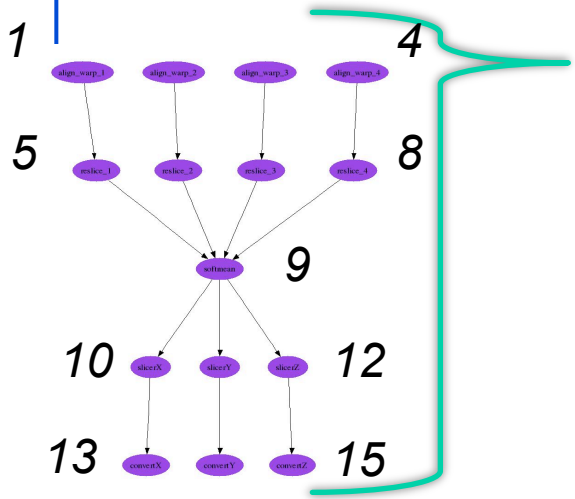
Basic Workflow Mapping

- Select where to run the computations
  - Change task nodes into nodes with executable descriptions
    - Execution location
    - Environment variables initializes
    - Appropriate command-line parameters set
- Select which data to access
  - Add stage-in nodes to move data to computations
  - Add stage-out nodes to transfer data out of remote sites to storage
  - Add data transfer nodes between computation nodes that execute on different resources

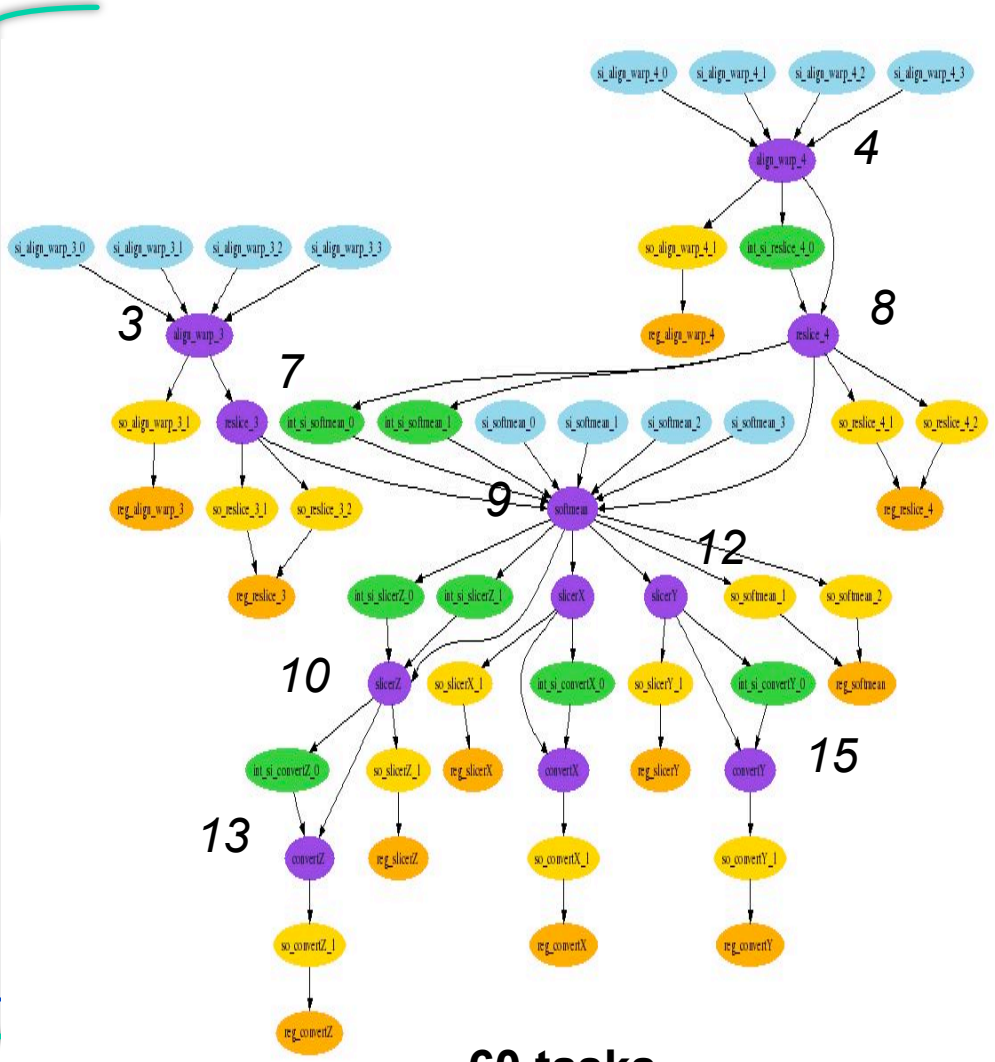
Basic Workflow Mapping

- Add nodes to create an execution directory on a remote site
- Add nodes that register the newly-created data products
- Add data cleanup nodes to remove data from remote sites when no longer needed
  - reduces workflow data footprint
- Provide provenance capture steps
  - Information about source of data, executables invoked, environment variables, parameters, machines used, performance

Pegasus Workflow Mapping



Original workflow: 15 compute nodes devoid of resource assignment



60 tasks

Resulting workflow mapped onto 3 Grid sites:

13 data stage-in nodes

11 compute nodes (4 reduced based on available intermediate data)

8 inter-site data transfers

14 data stage-out nodes to long-term storage

14 data registration nodes (data cataloging)

Catalogs used for discovery

- To execute on the a grid Pegasus needs to discover
  - Data (the input data that is required by the workflows)
  - Executables (Are there any application executables installed before hand)
  - Site Layout (What are the services running on an OSG site for example)

Discovery of Data

- Replica Catalog stores mappings between logical files and their target locations.
- Globus RLS
  - discover input files for the workflow
  - track data products created
  - data reuse
- Pegasus also interfaces with a variety of replica catalogs
  - File based Replica Catalog
    - useful for small datasets (like this tutorial)
    - cannot be shared across users.
  - Database based Replica Catalog
    - useful for medium sized datasets.
    - can be used across users.

How to: *A single client* ***rc-client*** *to interface with all type of replica catalogs*

Discovery of Site Layout

- Pegasus queries a site catalog to discover site layout
  - Installed job-managers for different types of schedulers
  - Installed GridFTP servers
  - Local Replica Catalogs where data residing in that site has to be catalogued
  - Site Wide Profiles like environment variables
  - Work and storage directories
- For the OSG, Pegasus interfaces with VORS (Virtual Organization Resource Selector) to generate a site catalog for OSG
- On the TG we can use MDS

How to: *A single client* ***pegasus-get-sites*** *to generate site catalog for OSG, Teragrid*

Discovery of Executables

- Transformation Catalog maps logical transformations to their physical locations
- Used to
  - discover application codes installed on the grid sites
  - discover statically compiled codes, that can be deployed at grid sites on demand

How to: *A single client* ***tc-client*** *to interface with all type of transformation catalogs*

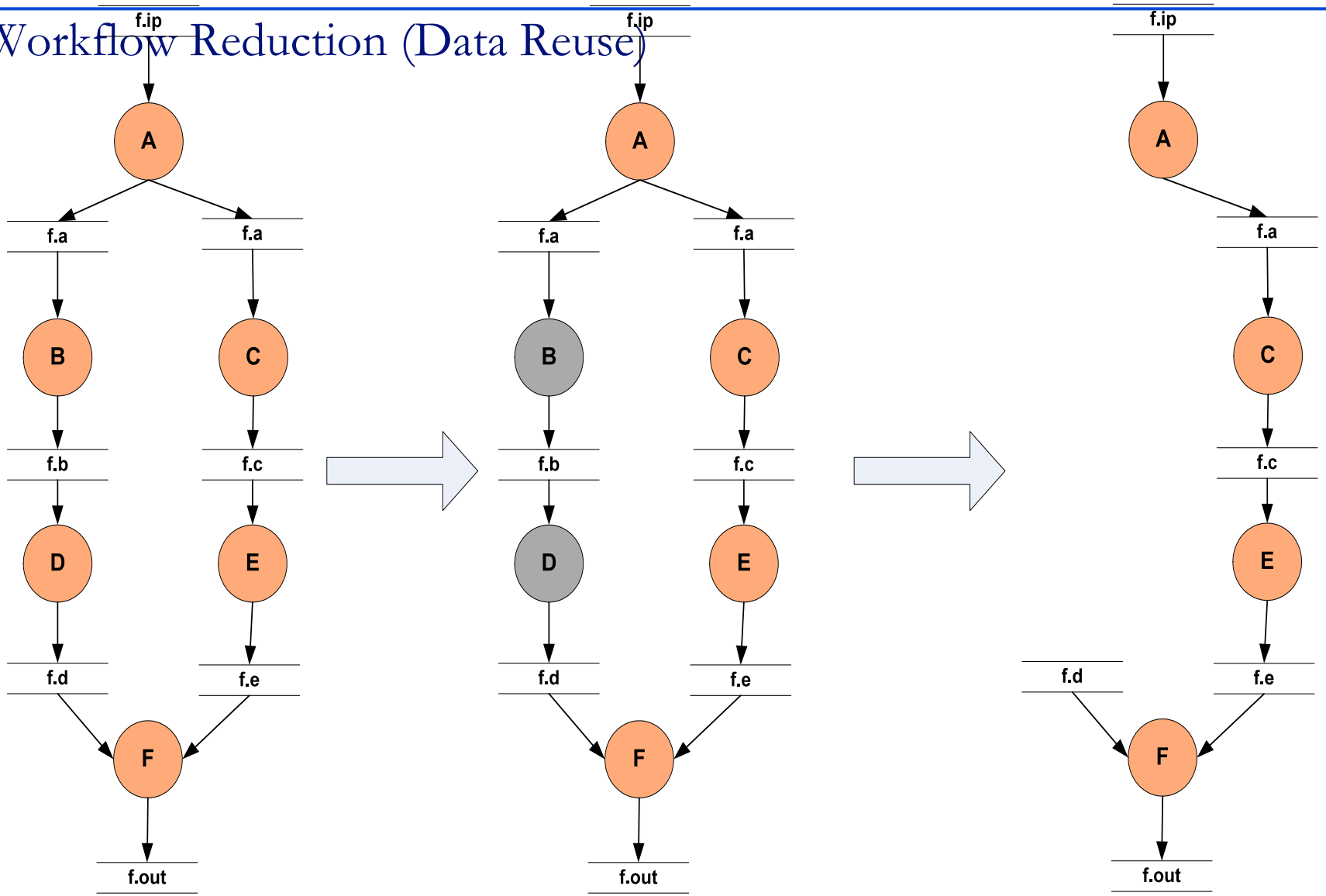
Simple Steps to run Pegasus

1. Specify your computation in terms of DAX
   - Write a simple DAX generator
   - Java based API provided with Pegasus
   - Details on http://pegasus.isi.edu/doc.php
2. Set up your catalogs
   - Use *pegasus-get-sites* to generate site catalog and transformation catalog for your environment
   - Record the locations of your input files in a replica client using *rc-client*
3. Plan your workflow
   - Use *pegasus-plan* to generate your executable workflow that is mapped onto the target resources
4. Submit your workflow
   - Use *pegasus-run* to submit your workflow
5. Monitor your workflow
   - Use *pegasus-status* to monitor the execution of your workflow

Optimizations during Mapping

- Node clustering for fine-grained computations
  - Can obtain significant performance benefits for some applications (in Montage ~80%, SCEC ~50%)
- Data reuse in case intermediate data products are available
  - Performance and reliability advantages—workflow-level checkpointing
- Data cleanup nodes can reduce workflow data footprint
  - by ~50% for Montage, applications such as LIGO need restructuring
- Workflow partitioning to adapt to changes in the environment
  - Map and execute small portions of the workflow at a time

Workflow Reduction (Data Reuse)



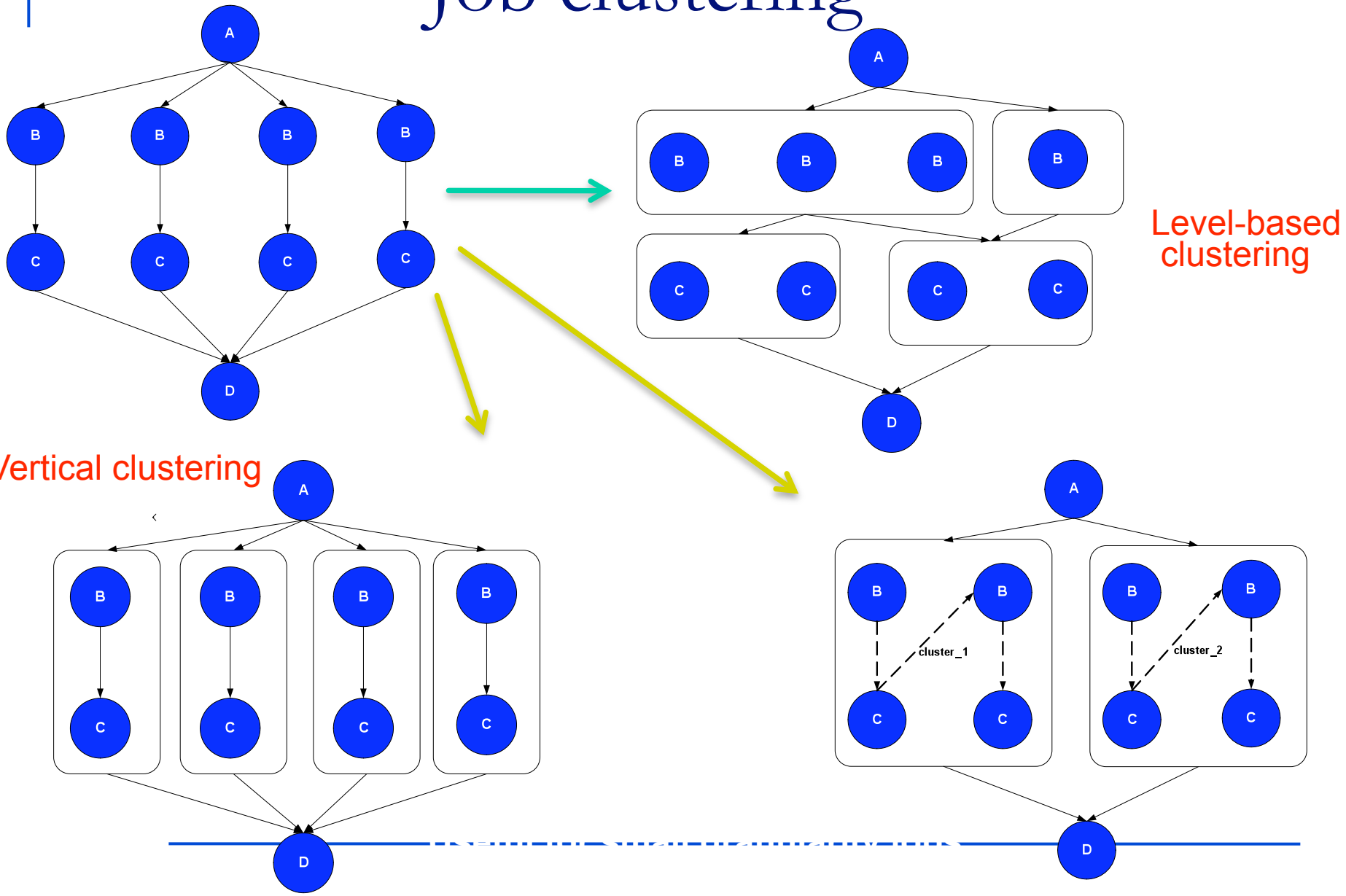
Abstract Workflow

File f.d exists somewhere.
Reuse it.
Mark Jobs D and B to delete

Delete Job D and Job B

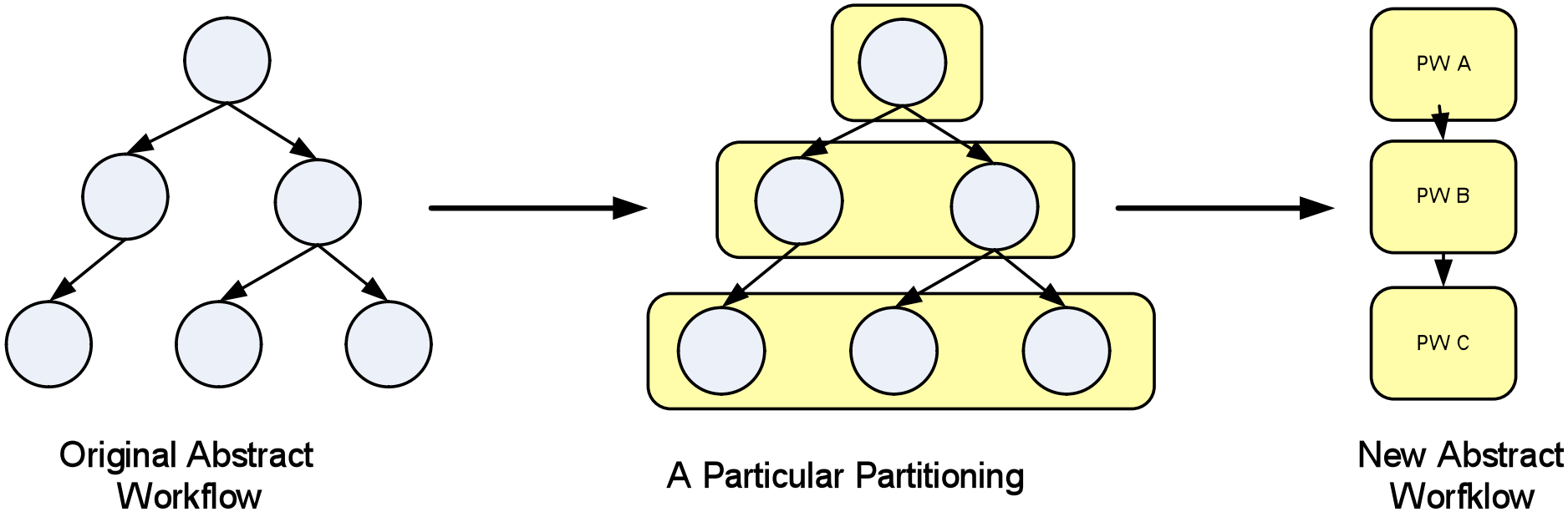
How to: *To trigger workflow reduction the files need to be cataloged in replica catalog at runtime. The registration flags for these files need to be set in the DAX*

Job clustering



How to: *To turn job clustering on, pass* ***--cluster*** *to* ***pegasus-plan***

Managing execution environment changes through partitioning



How to: *1) Partition the workflow into smaller partitions at runtime using* ***partitiondax*** *tool.*

2) Pass the partitioned dax to ***pegasus-plan*** *using the* ***--pdax*** *option.*

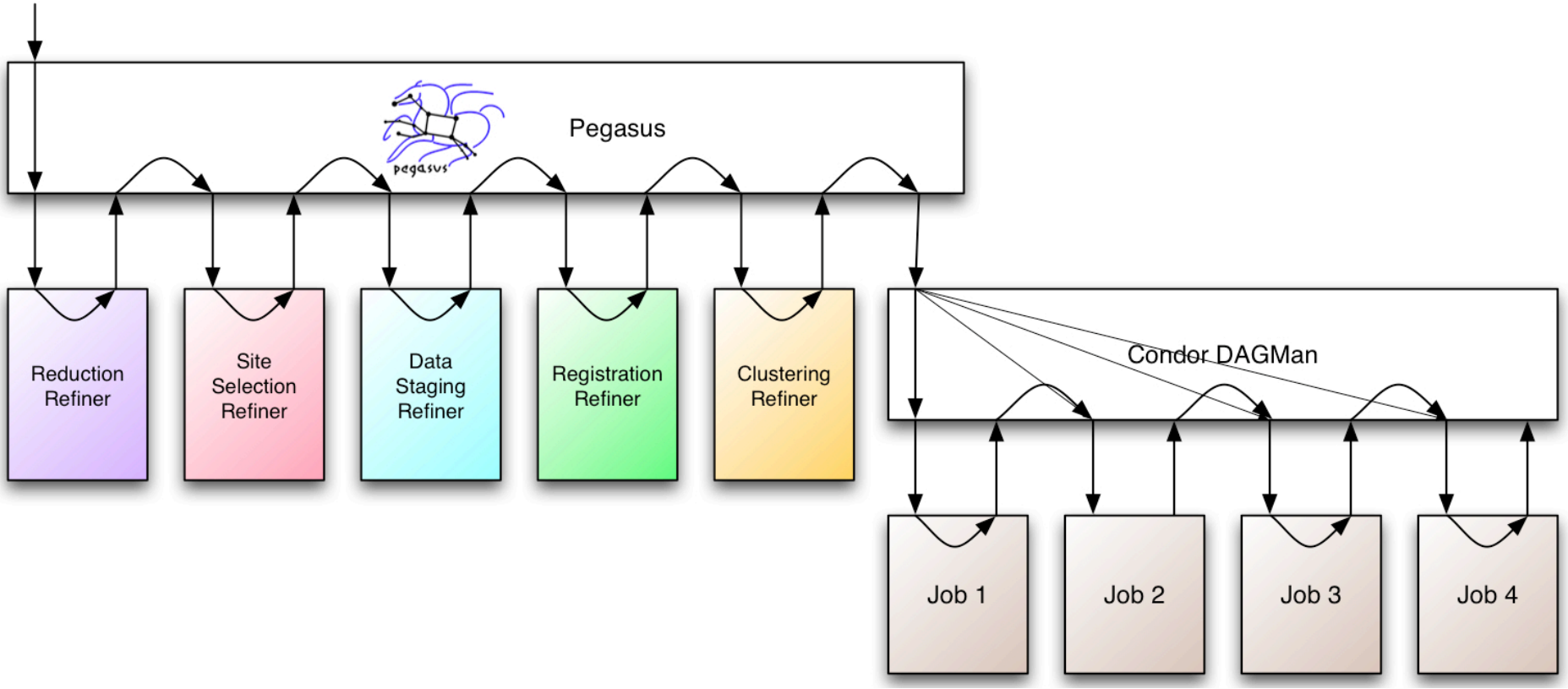
Paper: ***"Pegasus: a Framework for Mapping Complex Scientific Workflows onto Distributed Systems"****, E. Deelman, et al.* ***Scientific Programming Journal, Volume 13, Number 3, 2005***

Reliability Features of Pegasus and DAGMan

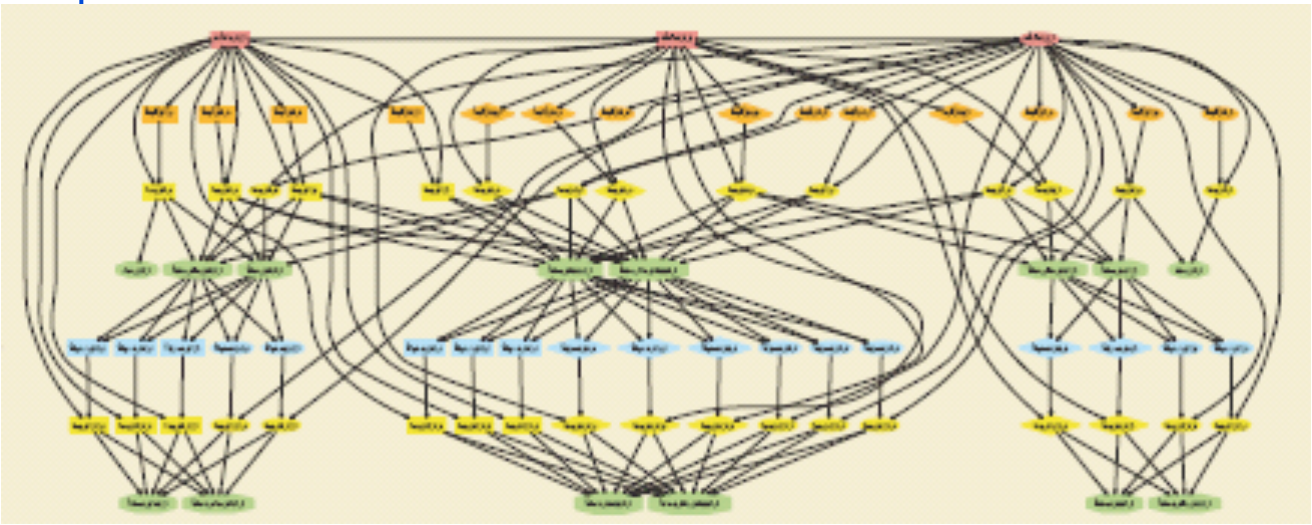
- Provides workflow-level checkpointing through data re-use
- Allows for automatic re-tries of
  - task execution
  - overall workflow execution
  - workflow mapping
- Tries alternative data sources for staging data
- Provides a rescue-DAG when all else fails
- Clustering techniques can reduce some of failures
  - Reduces load on CI services

Provenance tracking

- Uses the VDS provenance tracking catalog to record information about the execution of a single task
- Integrated with the PASOA provenance system to keep track of the entire workflow mapping and execution

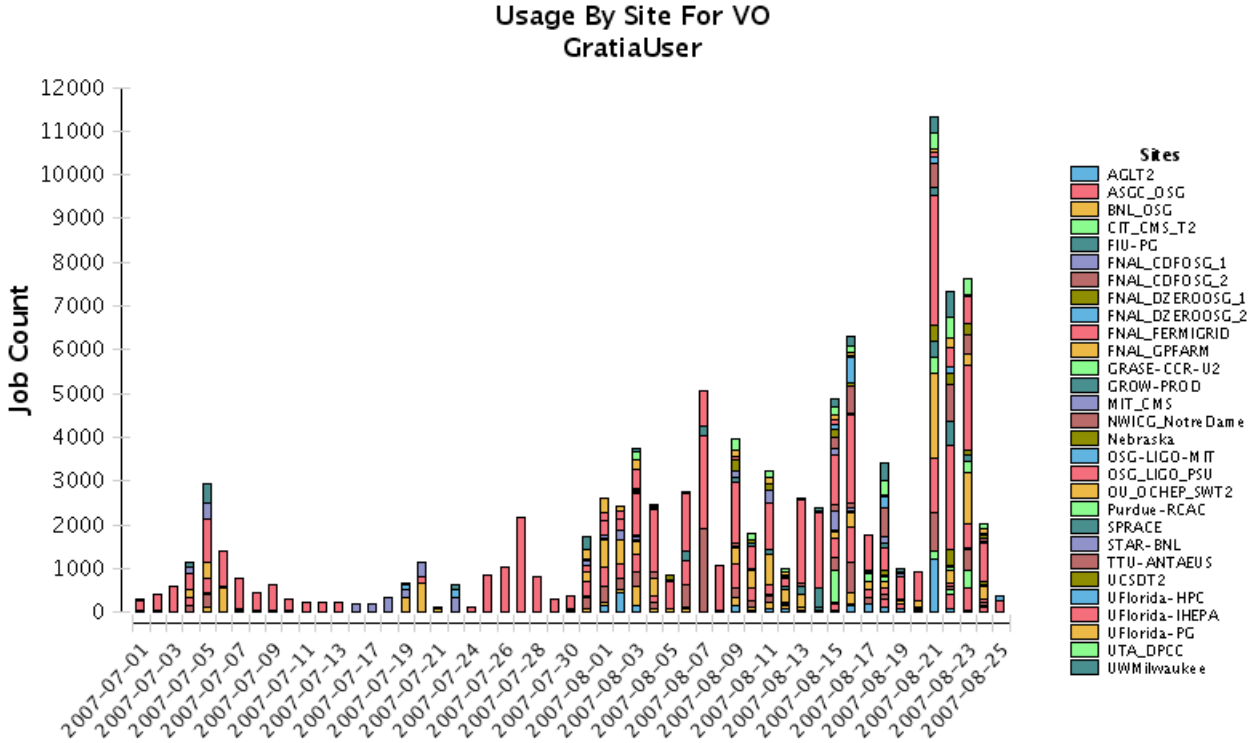


Pegasus Applications-LIGO

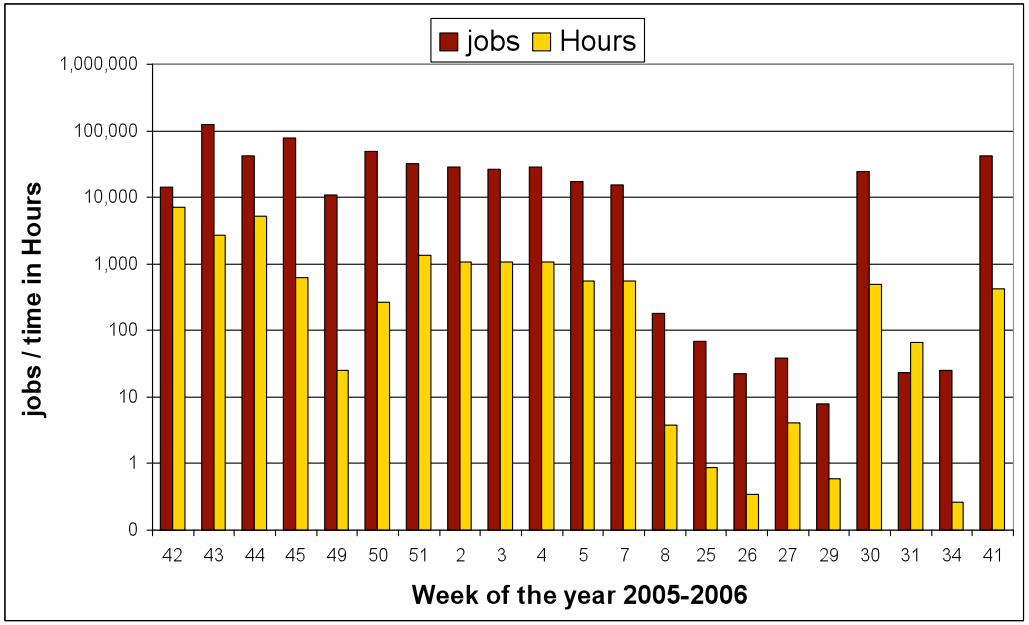


Support for LIGO on Open Science Grid LIGO Workflows: 185,000 nodes, 466,000 edges 10 TB of input data, 1 TB of output data.

LIGO Collaborators: Kent Blackburn, Duncan Brown, Britta Daubert, Scott Koranda, Stephen Fairhurst, and others



SCEC (Southern California Earthquake Center)



SCEC CyberShake workflows run using Pegasus-WMS on the TeraGrid and USC resources

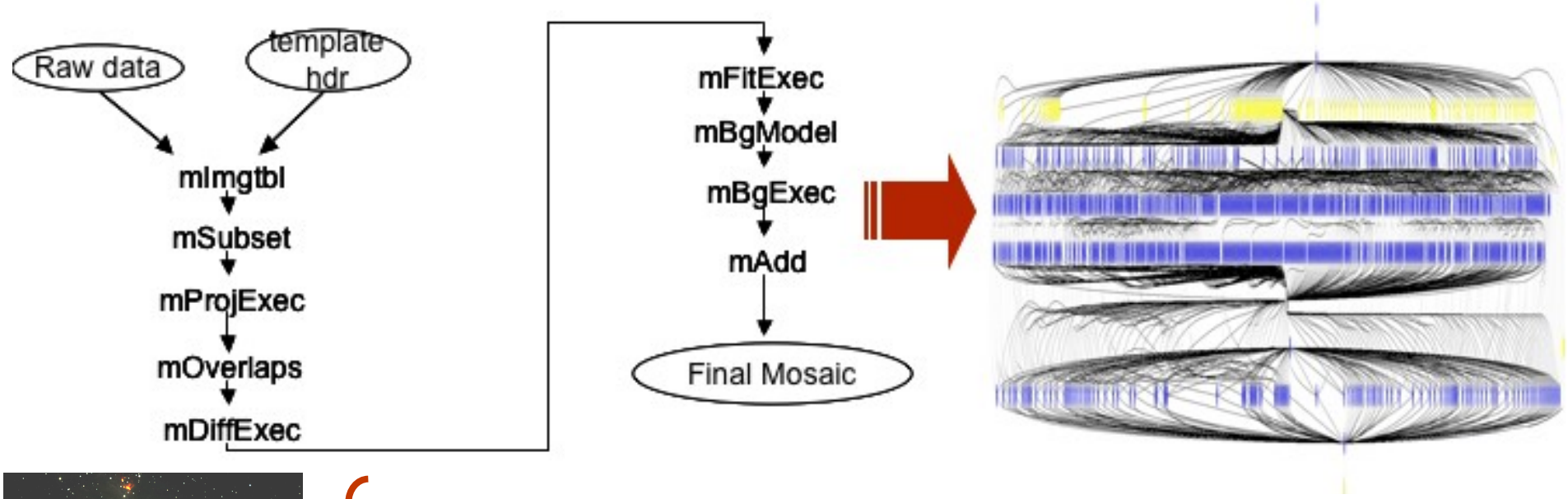
Cumulatively, the workflows consisted of over half a million tasks and used over 2.5 CPU Years.

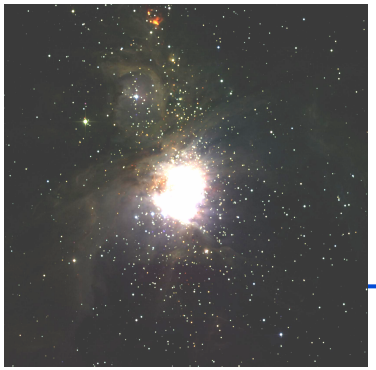
The largest CyberShake workflow contained on the order of 100,000 nodes and accessed 10TB of data

SCEC Collaborators: Scott Callahan, Robert Graves, Gideon Juve, Philip Maechling, David Meyers, David Okaya, Mona Wong-Barnum

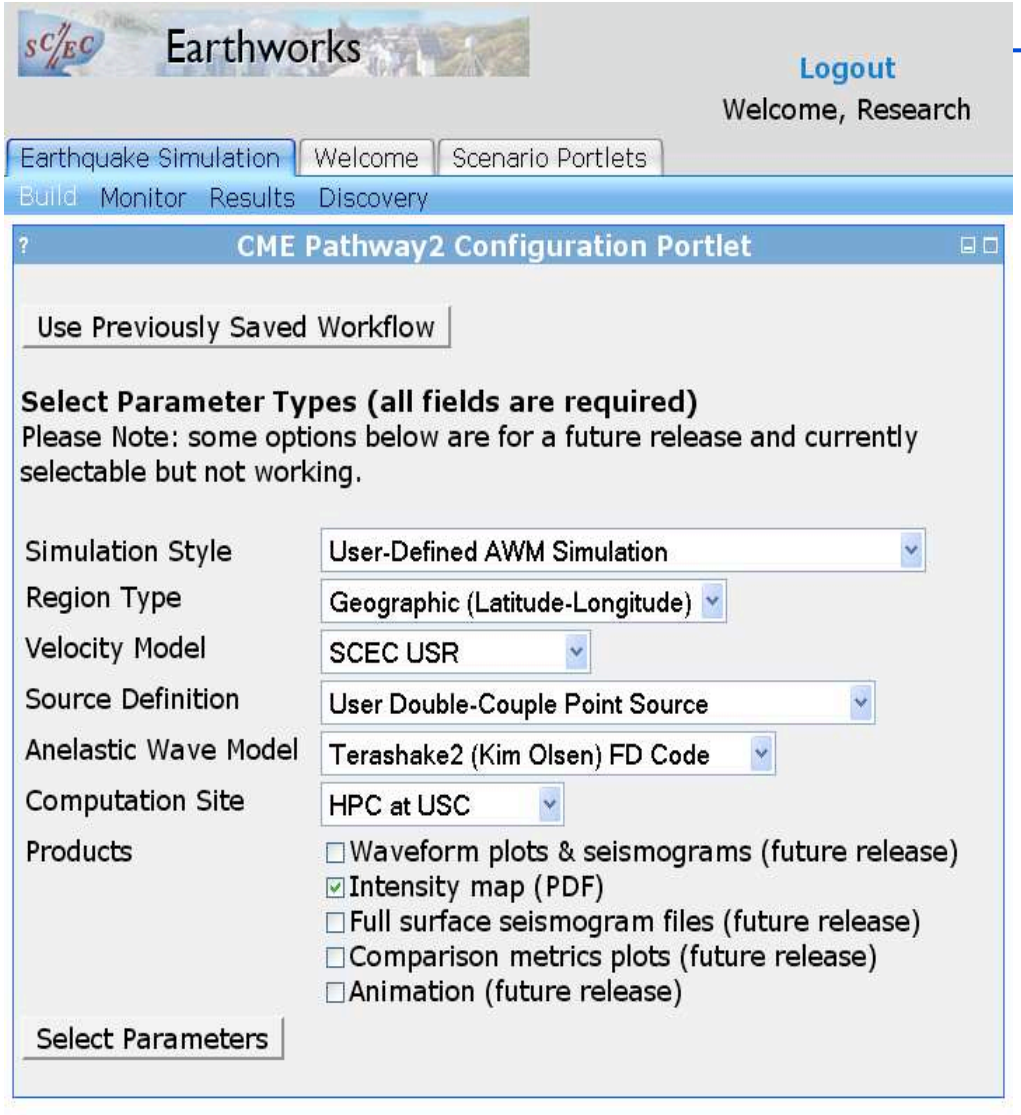
National Virtual Observatory and Montage

NVO's Montage mosaic application: Transformed a single-processor code into a workflow and parallelized computations to process larger-scale images

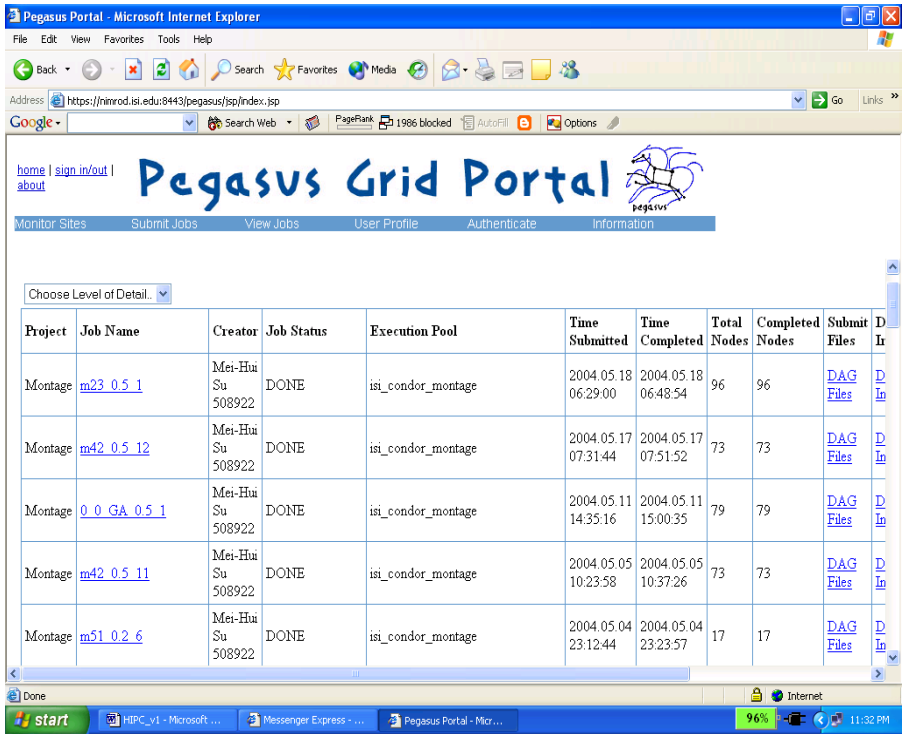




- Montage is a collaboration between IPAC, JPL and CACR



Portal Interfaces for Pegasus workflows



September 8, 2007

Gridsphere-based portal for workflow monitoring

Ensemble Manager

- Ensemble = a set of workflows
- Command-line interfaces to submit, start, monitor ensembles and their elements
  - The state of the workflows and ensembles is stored in a DB
  - Priorities can be given to workflows and ensembles
- Future work
  - Kill
  - Suspend
  - Restart
  - Web-based interface

What does Pegasus do for an application?

- Provides a Grid-aware workflow management tool
  - Interfaces with the Replica Location Service to discover data
  - Does replica selection to select replica.
  - Manages data transfer by interfacing to various transfer services like RFT, Stork and clients like globus-url-copy.
  - No need to stage-in data before hand. We do it within the workflow as and when it is required.
  - Reduced Storage footprint. Data is also cleaned as the workflow progresses.
  - Improves successful application execution
  - Improves application performance
  - Data Reuse
    - Avoids duplicate computations
    - Can reuse data that has been generated earlier.

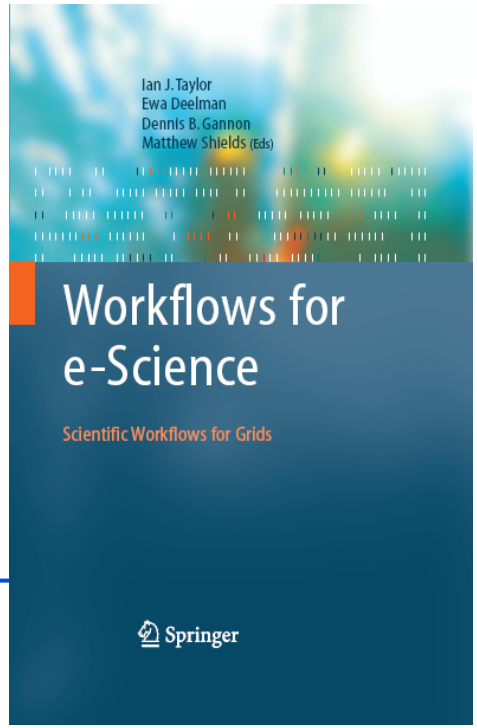
Relevant Links

- Pegasus: http://pegasus.isi.edu
  - Distributed as part of VDT
  - Standalone version in VDT 1.7 and later
  - Can be downloaded directly from
    - http://pegasus.isi.edu/code.php

- Interested in trying out Pegasus
  - Do the tutorial
    - http://pegasus.isi.edu/tutorial/tg07/index.html
    - Send email to pegasus@isi.edu,

      to do tutorial on ISI cluster.
  - Quickstart Guide
    - Available at http://pegasus.isi.edu/doc.php
    - More detailed documentation appearing soon.

- Support lists
  - pegasus-support@mailman.isi.edu



Introduction to Swift

Parallel scripting for distributed systems

Mike Wilde
wilde@mcs.anl.gov

Ben Clifford
benc@ci.uchicago.edu

Computation Institute, University of Chicago
and Argonne National Laboratory

www.ci.uchicago.edu/swift

Why script in Swift?

- 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)

Swift is…

- A language for writing scripts that:
  - process and produce large collections of persistent data
  - with large and/or complex sequences of application programs
  - on diverse distributed systems
  - with a high degree of parallelism
  - persisting over long periods of time
  - surviving infrastructure failures
  - and tracking the provenance of execution

Swift programs

- A Swift script is a set of **functions**
  - Atomic functions wrap & invoke application programs
  - Composite functions invoke other functions
- Data is **typed** as composable arrays and structures of files and simple scalar types (int, float, string)
- Collections of **persistent file structures** (datasets) are **mapped** into this data model
- Members of datasets can be processed in **parallel**
- Statements in a procedure are executed in **data-flow** dependency order and concurrency
- Variables are **single assignment**
- **Provenance** is gathered as scripts execute

A simple Swift script

```
type imagefile;

(imagefile output) flip(imagefile input) {
  app {
    convert "-rotate" "180" @input @output;
  }
}

imagefile stars <"orion.2008.0117.jpg">;
imagefile flipped <"output.jpg">;

flipped = flip(stars);
```

Parallelism via foreach { }

```
type imagefile;

(imagefile output) flip(imagefile input) {
  app {
    convert "-rotate" "180" @input @output;
  }
}

imagefile observations[ ] <simple_mapper; prefix="orion">;
imagefile flipped[ ]      <simple_mapper; prefix="orion-flipped">;
```

Name outputs based on inputs

```
foreach obs,i in observations {
  flipped[i] = flip(obs);
}
```

Process all dataset members in parallel

A Swift data mining example

```
type pcapfile;                  // packet data capture - input file type
type angleout;                  // “angle” data mining output
type anglecenter;               // geospatial centroid output

(angleout ofile, anglecenter cfile) angle4 (pcapfile ifile)
{
  app { angle4.sh --input @ifile --output @ofile --coords @cfile; }
// interface to shell script
}

pcapfile infile <"anl2-1182-dump.1.980.pcap">;    // maps real file

angleout      outdata <"data.out">;
anglecenter outcenter <"data.center">;

(outdata, outcenter) = angle4(infile);
```

Parallelism and name mapping

```
type pcapfile;
type angleout;
type anglecenter;

(angleout ofile, anglecenter cfile) angle4 (pcapfile ifile)
{
  app { angle4.sh --input @ifile --output @ofile --coords @cfile; }
}

pcapfile pcapfiles[]<filesys_mapper; prefix="pc", suffix=".pcap">;

angleout    of[] <structured_regexp_mapper;
                  source=pcapfiles,match="pc(.*)\.pcap",
                  transform="_output/of/of\1.angle">;

anglecenter cf[] <structured_regexp_mapper;
                  source=pcapfiles,match="pc(.*)\.pcap",
                  transform="_output/cf/cf\1.center">;
```

Name outputs based on inputs

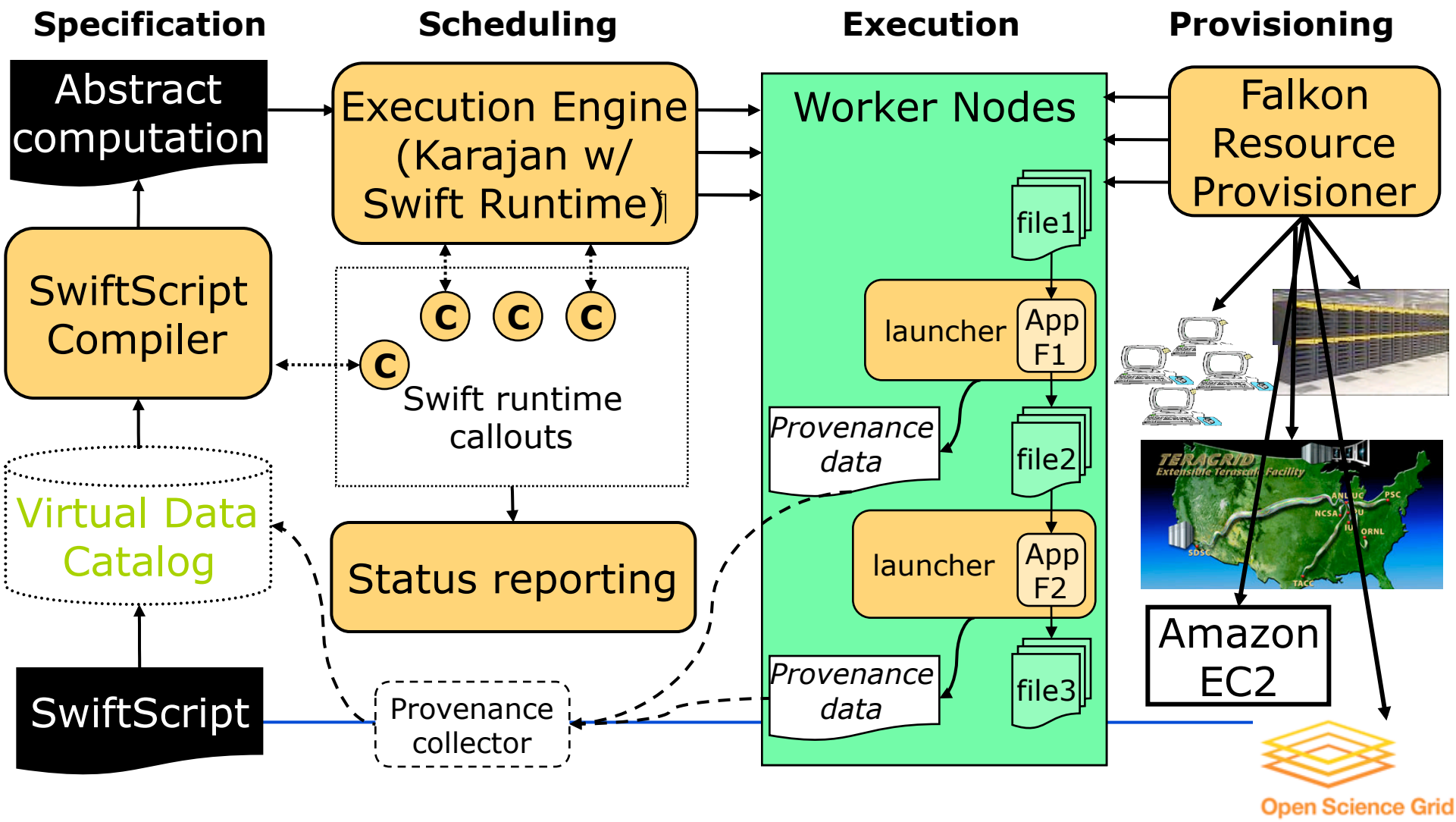
```
foreach pf,i in pcapfiles {
  (of[i],cf[i]) = angle4(pf);
}
```

Iterate over dataset members in parallel

The Swift Scripting Model

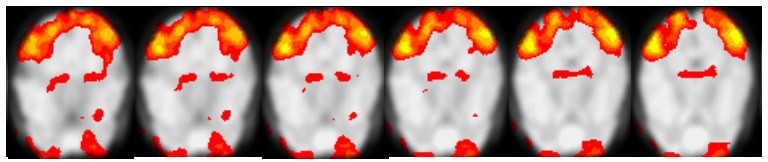
- Program in high-level, functional model
- Swift hides issues of location, mechanism and data representation
- Basic active elements are functions that encapsulate application tools and run jobs
- Typed data model: structures and arrays of files and scalar types
- Variables are single assignment
- Control structures perform conditional, iterative and *parallel* operations

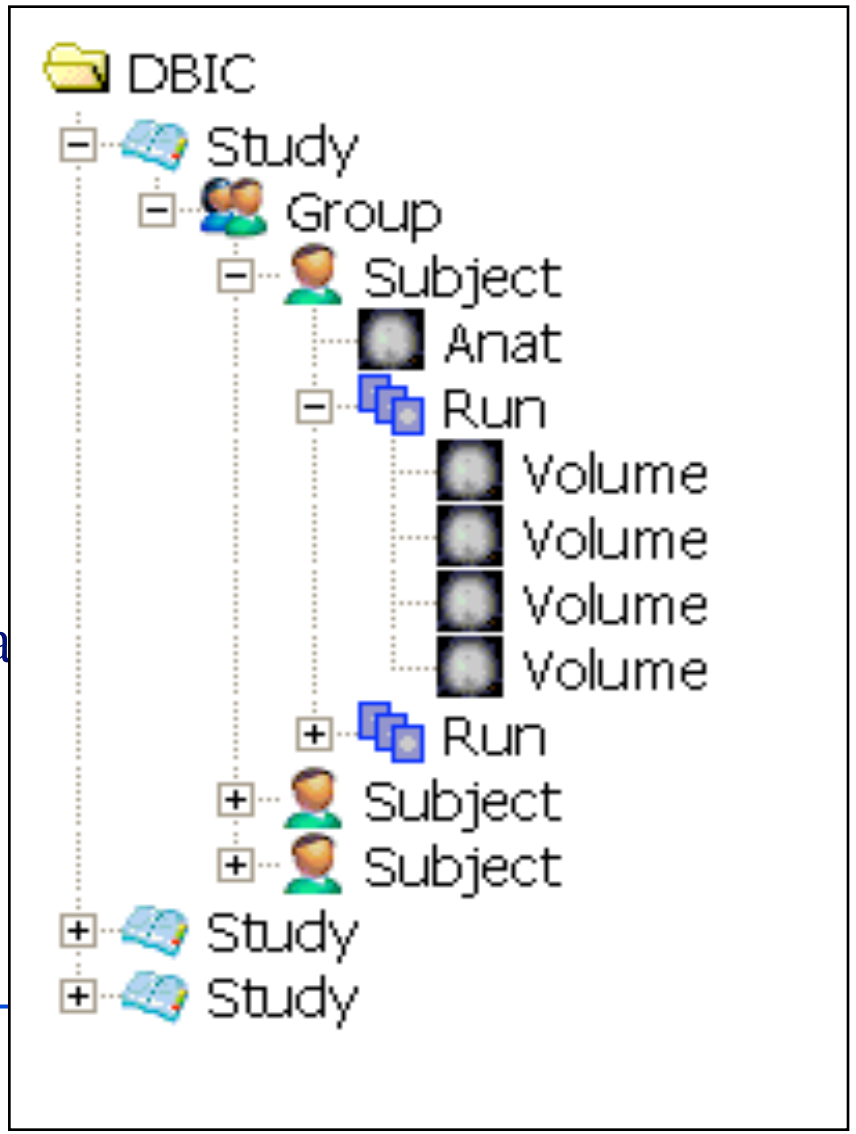
Swift Architecture



The Messy Data Problem (1)

- Scientific data is often logically structured
  - E.g., hierarchical structure
  - Common to map functions over dataset members
  - Nested map operations can sca millions of objects





The Messy Data Problem (2)

- Heterogeneous storage format & access protocols
  - Same dataset can be stored in text file, spreadsheet, database, …
  - Access via filesystem, DBMS, HTTP, WebDAV, …
- Metadata encoded in directory and file names
- Hinders program development, composition, execution

./group23
drwxr-xr-x 4 yongzh users 2048 Nov 12 14:15 **AA**
drwxr-xr-x 4 yongzh users 2048 Nov 11 21:13 **CH**
drwxr-xr-x 4 yongzh users 2048 Nov 11 16:32 **EC**

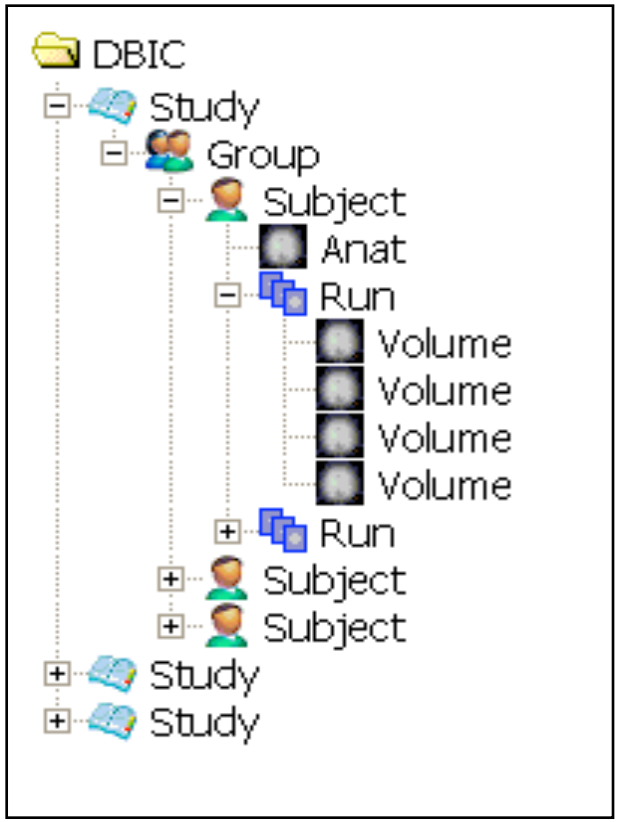
./group23/AA:
drwxr-xr-x 5 yongzh users 2048 Nov 5 12:41 **04nov06aa**
drwxr-xr-x 4 yongzh users 2048 Dec 6 12:24 **11nov06aa**

. /group23/AA/04nov06aa:
drwxr-xr-x 2 yongzh users 2048 Nov 5 12:52 **ANATOMY**
drwxr-xr-x 2 yongzh users 49152 Dec 5 11:40 **FUNCTIONAL**

. /group23/AA/04nov06aa/ANATOMY:
-rw-r--r-- 1 yongzh users 348 Nov 5 12:29 **coplanar.hdr**
-rw-r--r-- 1 yongzh users 16777216 Nov 5 12:29 **coplanar.img**

. /group23/AA/04nov06aa/FUNCTIONAL:
-rw-r--r-- 1 yongzh users 348 Nov 5 12:32 **bold1_0001.hdr**
-rw-r--r-- 1 yongzh users 409600 Nov 5 12:32 **bold1_0001.img**
-rw-r--r-- 1 yongzh users 348 Nov 5 12:32 **bold1_0002.hdr**
-rw-r--r-- 1 yongzh users 409600 Nov 5 12:32 **bold1_0002.img**
-rw-r--r-- 1 yongzh users 496 Nov 15 20:44 **bold1_0002.mat**
-rw-r--r-- 1 yongzh users 348 Nov 5 12:32 **bold1_0003.hdr**
-rw-r--r-- 1 yongzh users 409600 Nov 5 12:32 **bold1_0003.img**

Example: fMRI Type Definitions



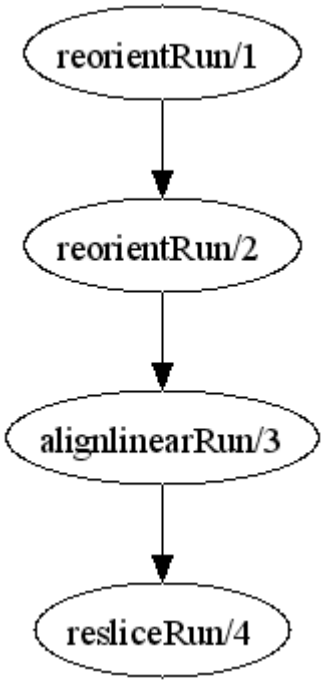
Simplified version of fMRI AIRSN Program (Spatial Normalization)

```
type Study {
        Group g[ ];
}

type Group {
        Subject s[ ];
}

type Subject {
        Volume anat;
        Run run[ ];
}

type Run {
        Volume v[ ];
}

type Volume {
        Image img;
        Header hdr;
}
```

```
type Image {};

type Header {};

type Warp {};

type Air {};

type AirVec {
        Air a[ ];
}

type NormAnat {
        Volume anat;
        Warp aWarp;
        Volume nHires;
}
```

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);
      }
}
```

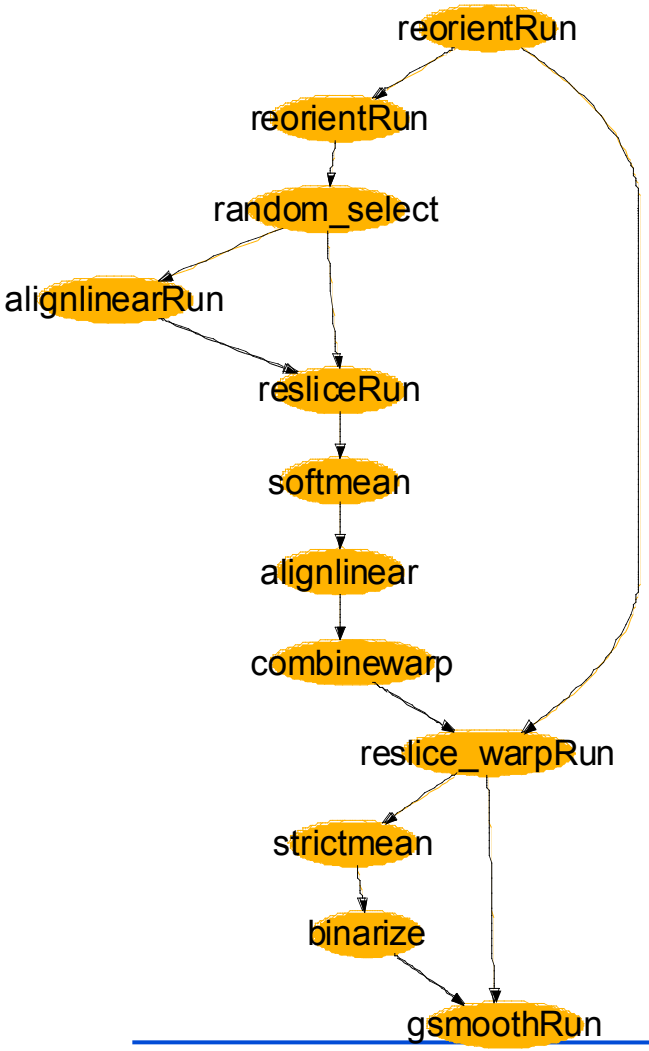
AIRSN Program Definition

```
(Run snr) functional ( Run r, NormAnat a,
                       Air shrink ) {
    Run yroRun = reorientRun( r , "y" );
    Run roRun = reorientRun( yroRun , "x" );
    Volume std = roRun[0];
    Run rndr = random_select( roRun, 0.1 );
    AirVector rndAirVec = align_linearRun( rndr, std, 12, 1000, 1000, "81 3 3" );
    Run reslicedRndr = resliceRun( rndr, rndAirVec, "o", "k" );
    Volume meanRand = softmean( reslicedRndr, "y", "null" );
    Air mnQAAir = alignlinear( a.nHires, meanRand, 6, 1000, 4, "81 3 3" );
    Warp boldNormWarp = combinewarp( shrink, a.aWarp, mnQAAir );
    Run nr = reslice_warp_run( boldNormWarp, roRun );
    Volume meanAll = strictmean( nr, "y", "null" )
    Volume boldMask = binarize( meanAll, "y" );
    snr = gsmoothRun( nr, boldMask, "6 6 6" );
}
```

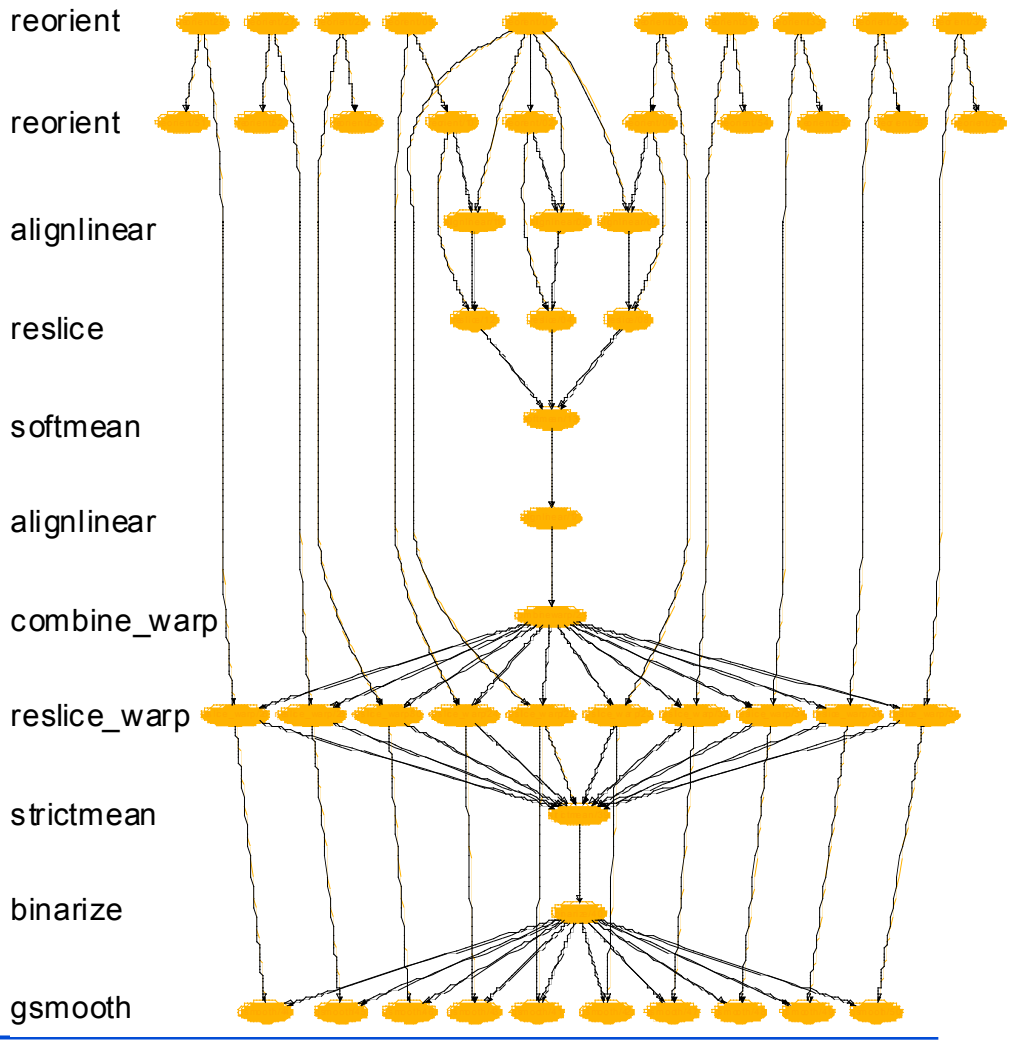
```
(Run or) reorientRun (Run ir,
                      string direction) {
    foreach Volume iv, i in ir.v {
        or.v[i] = reorient(iv, direction);
    }
}
```

Automated image registration for spatial normalization

AIRSN workflow:



AIRSN workflow expanded:



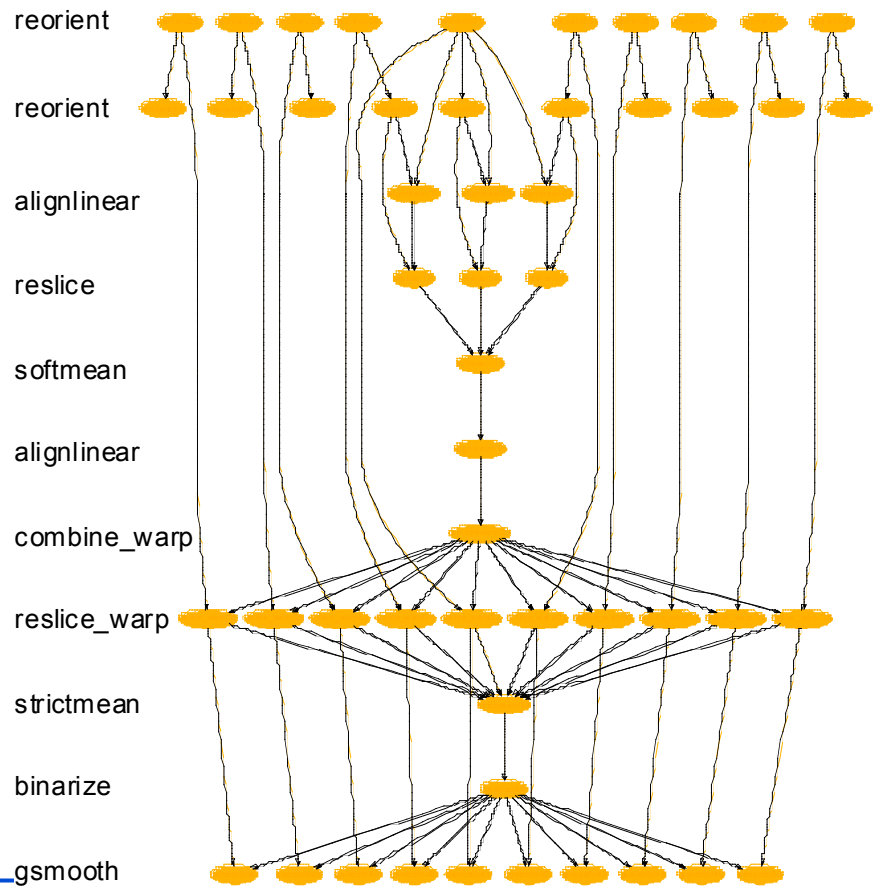
Collaboration with James Dobson, Dartmouth [SIGMOD Record Sep05]

SwiftScript Expressiveness

Lines of code with different workflow encodings

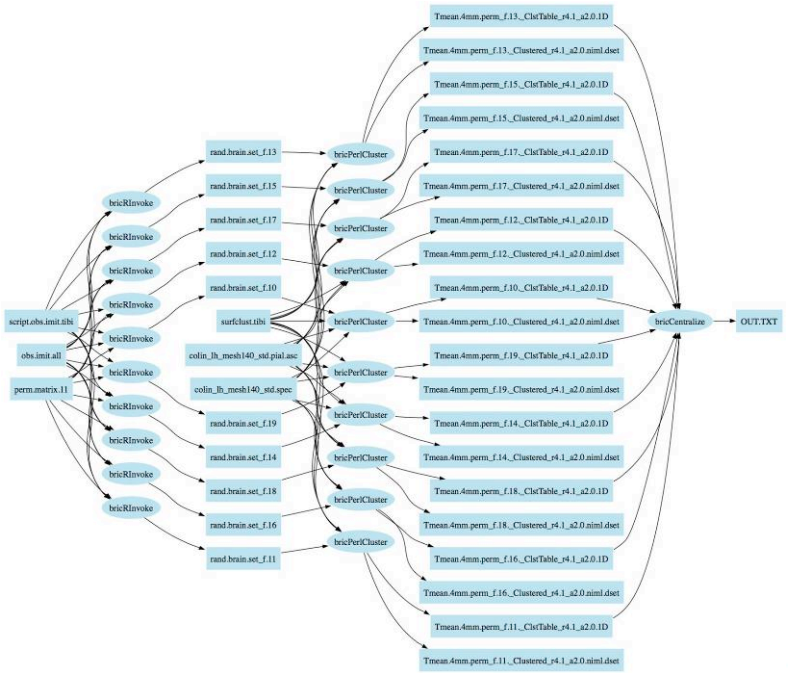
fMRI Workflow	*Shell Script*	*VDL*	*Swift*
ATLAS1	**49**	**72**	**6**
ATLAS2	**97**	**135**	**10**
FILM1	**63**	**134**	**17**
FEAT	**84**	**191**	**13**
AIRSN	**215**	**~400**	**34**

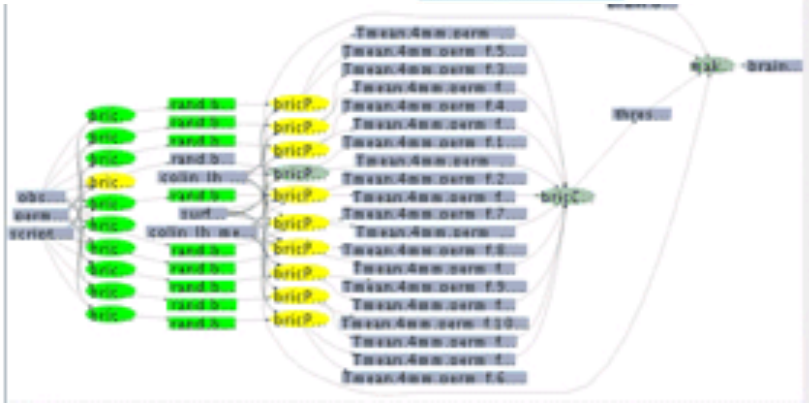
AIRSN workflow:expanded:



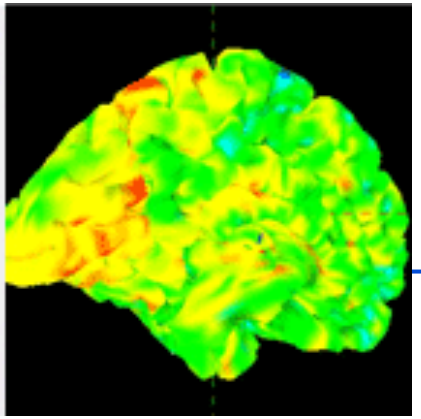
Collaboration with James Dobson, Dartmouth [SIGMOD Record Sep05]

Application example: ACTIVAL: Neural activation validation





The ACTIVAL Swift script identifies clusters of neural activity not likely to be active by random chance: switch labels of the conditions for one or more participants; calculate the delta values in each voxel, re-calculate the reliability of delta in each voxel, and evaluate clusters found. If the clusters in data are greater than the majority of the clusters found in the permutations, then the null hypothesis is refuted indicating that clusters of activity found in our experiment are not likely to be found by chance.



Work by S. Small and U. Hasson, UChicago.

SwiftScript Workflow ACTIVAL – Data types and utilities

```
type script {}                          type fullBrainData {}
type brainMeasurements{}                type fullBrainSpecs {}
type precomputedPermutations{}          type brainDataset {}
type brainClusterTable {}
type brainDatasets{ brainDataset b[]; }
type brainClusters{ brainClusterTable c[]; }
```

// Procedure to run "R" statistical package

```
(brainDataset t) bricRInvoke (script permutationScript, int iterationNo,
    brainMeasurements dataAll, precomputedPermutations dataPerm) {
      app { bricRInvoke @filename(permutationScript) iterationNo
                        @filename(dataAll) @filename(dataPerm); }
}
```

// Procedure to run AFNI Clustering tool

```
(brainClusterTable v, brainDataset t) bricCluster (script clusterScript,
  int iterationNo, brainDataset randBrain, fullBrainData brainFile,
  fullBrainSpecs specFile) {
      app { bricPerlCluster @filename(clusterScript) iterationNo
                        @filename(randBrain) @filename(brainFile)
                        @filename(specFile); }
}
```

// Procedure to merge results based on statistical likelhoods

```
(brainClusterTable t) bricCentralize ( brainClusterTable bc[]) {
     app { bricCentralize @filenames(bc); }
}
```

ACTIVAL Workflow – Dataset iteration procedures

// Procedure to iterate over the data collection

```
(brainClusters randCluster, brainDatasets dsetReturn) brain_cluster
   (fullBrainData brainFile, fullBrainSpecs specFile)
{
   int sequence[]=[1:2000];

   brainMeasurements          dataAll<fixed_mapper; file="obs.imit.all">;
   precomputedPermutations dataPerm<fixed_mapper; file="perm.matrix.11">;
   script                          randScript<fixed_mapper; file="script.obs.imit.tibi">;
   script                          clusterScript<fixed_mapper; file="surfclust.tibi">;
   brainDatasets                randBrains<simple_mapper; prefix="rand.brain.set">;

   foreach int i in sequence {
      randBrains.b[i] = bricRInvoke(randScript,i,dataAll,dataPerm);
      brainDataset rBrain = randBrains.b[i] ;
      (randCluster.c[i],dsetReturn.b[i]) =
         bricCluster(clusterScript,i,rBrain, brainFile,specFile);
   }
}
```

ACTIVAL Workflow – Main Workflow Program

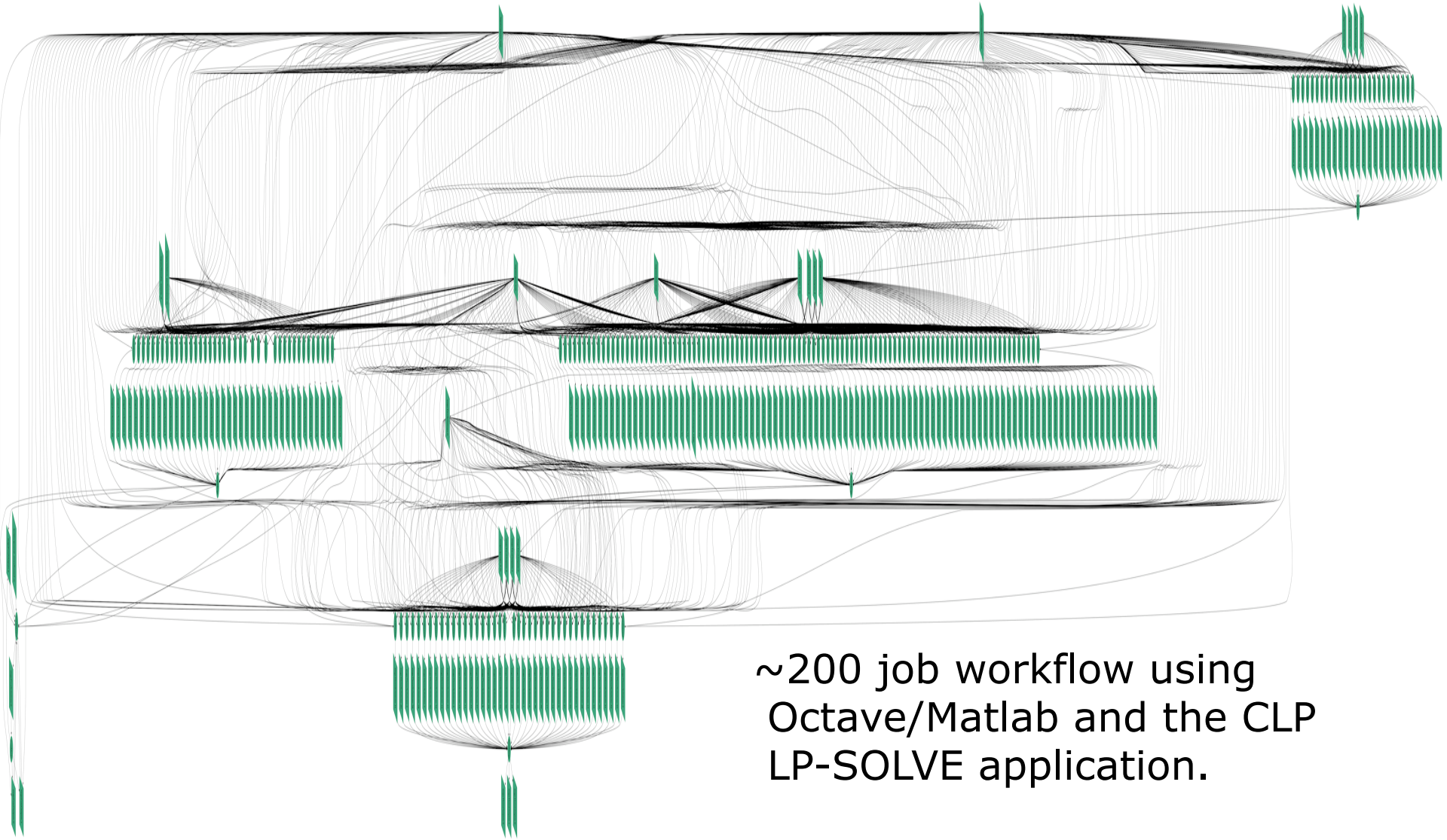
// Declare datasets

```
fullBrainData       brainFile<fixed_mapper; file="colin_lh_mesh140_std.pial.asc">;
fullBrainSpecs      specFile<fixed_mapper; file="colin_lh_mesh140_std.spec">;

brainDatasets       randBrain<simple_mapper; prefix="rand.brain.set">;
brainClusters       randCluster<simple_mapper; prefix="Tmean.4mm.perm",
                        suffix="_ClstTable_r4.1_a2.0.1D">;
brainDatasets       dsetReturn<simple_mapper; prefix="Tmean.4mm.perm",
                        suffix="_Clustered_r4.1_a2.0.niml.dset">;
brainClusterTable   clusterThresholdsTable<fixed_mapper; file="thresholds.table">;
brainDataset        brainResult<fixed_mapper; file="brain.final.dset">;
brainDataset        origBrain<fixed_mapper; file="brain.permutation.1">;
```

// Main program – executes the entire workflow

```
(randCluster, dsetReturn) = brain_cluster(brainFile, specFile);

clusterThresholdsTable = bricCentralize (randCluster.c);

brainResult = makebrain(origBrain,clusterThresholdsTable,brainFile,specFile);
```

Swift Application: Economics "moral hazard" problem



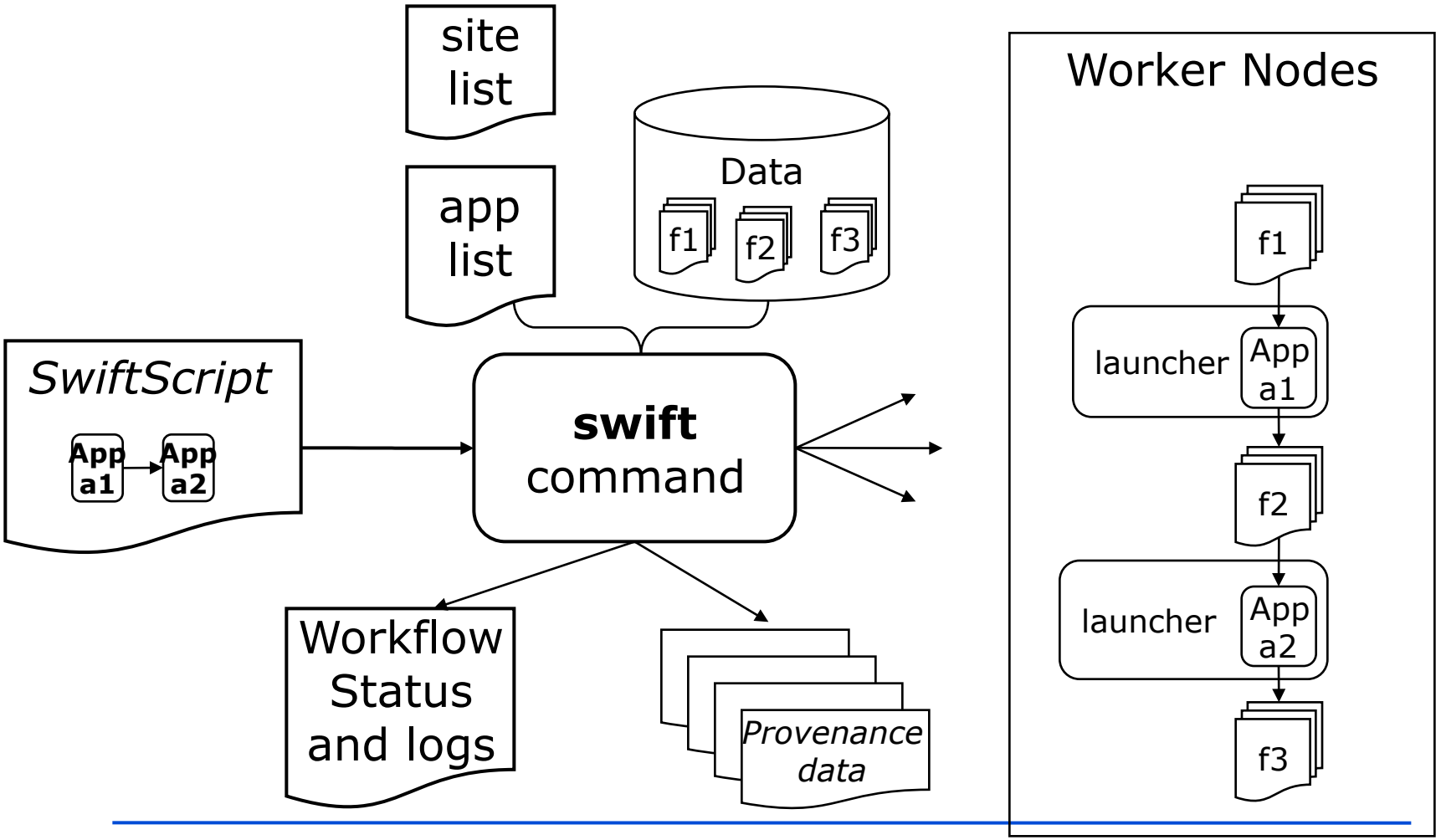
~200 job workflow using Octave/Matlab and the CLP LP-SOLVE application.

Work by Tibi Stef-Praun, CI, with Robert Townsend & Gabriel Madiera, UChicago Economics

Running swift

- Fully contained Java grid client
- Can test on a local machine
- Can run on a PBS cluster
- Runs on multiple clusters over Grid interfaces

Using Swift


site list
app list
Data
f1
f2
f3
SwiftScript
App a1
App a2
swift command
Workflow Status and logs
Provenance data
Worker Nodes
f1
launcher
App a1
f2
launcher
App a2
f3

The Variable model

- Single assignment:
  - Can only assign a value to a var once
  - This makes data flow semantics much cleaner to specify, understand and implement
- Variables are scalars or references to composite objects
- Variables are typed
- File typed variables are “mapped” to files

Data Flow Model

- This is what makes it possible to be location independent
- Computations proceed when data is ready (often not in source-code order)
- User specifies DATA dependencies, doesn't worry about sequencing of operations
- Exposes maximal parallelism

Swift statements

- Var declarations
  - Can be mapped
- Type declarations
- Assignment statements
  - Assignments are type-checked
- Control-flow statements
  - if, foreach, iterate
- Function declarations

Passing scripts as data

- When running scripting languages, target language interpreter can be the executable
- Powerful technique for running scripts in:
  - sh, bash
  - Perl, Python, Tcl
  - R, Octave
- These are often pre-installed at known places
  - No application installation needed
- Need to deal with library modules manually

Assessing your analysis tool performance

- Job usage records tell where when and how things ran:

V runtime cputime

angle4-szlfhtji-kickstart.xml 2007-11-08T23:03:53.733-06:00 0 0 1177.024 1732.503 4.528 ia64 tg-c007.uc.teragrid.org

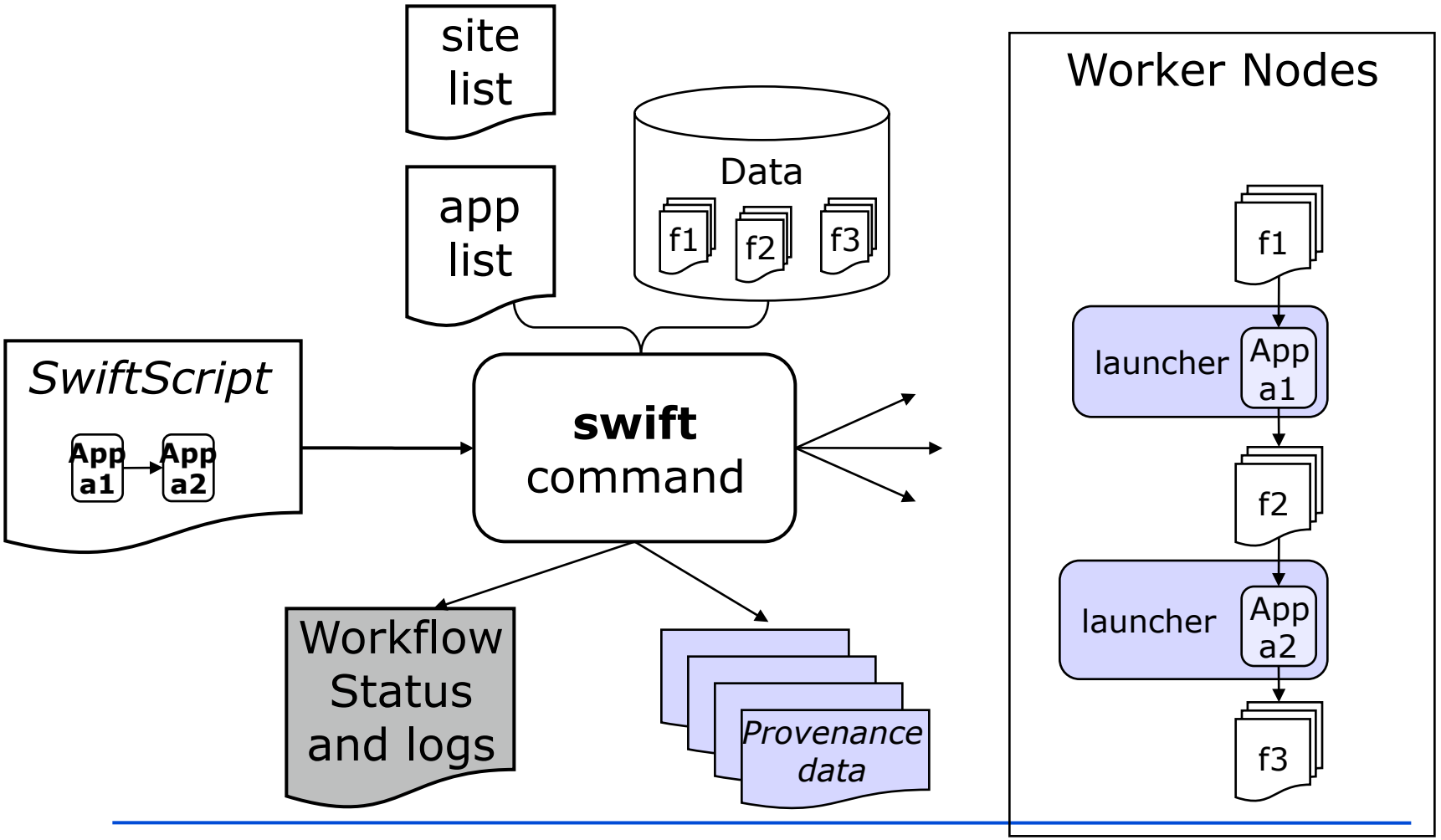
angle4-hvlfhtji-kickstart.xml 2007-11-08T23:00:53.395-06:00 0 0 1017.651 1536.020 4.283 ia64 tg-c034.uc.teragrid.org

angle4-oimfhtji-kickstart.xml 2007-11-08T23:30:06.839-06:00 0 0 868.372 1250.523 3.049 ia64 tg-c015.uc.teragrid.org

angle4-u9mfhtji-kickstart.xml 2007-11-08T23:15:55.949-06:00 0 0 817.826 898.716 5.474 ia64 tg-c035.uc.teragrid.org

- Analysis tools display this visually

Performance recording



Data Management

- Directories and management model
  - local dir, storage dir, work dir
  - caching within workflow
  - reuse of files on restart
- Makes unique names for: jobs, files, wf
- Can leave data on a site
  - For now, in Swift you need to track it
  - In Pegasus (and VDS) this is done automatically

Mappers and Vars

- Vars can be “file valued”
- Many useful mappers built-in, written in Java to the Mapper interface
- “Ext”ernal mapper can be easily written as an external script in any language

Mapping outputs based on input names

```
type pcapfile;
type angleout;
type anglecenter;

(angleout ofile, anglecenter cfile) angle4 (pcapfile ifile)
{
  app { angle4 @ifile @ofile @cfile; }
}

pcapfile pcapfiles[]<filesys_mapper; prefix="pc", suffix=".pcap">;

angleout    of[] <structured_regexp_mapper;
                  source=pcapfiles,match="pc(.*)\.pcap",
                  transform="_output/of/of\1.angle">;
anglecenter cf[] <structured_regexp_mapper;
                  source=pcapfiles,match="pc(.*)\.pcap",
                  transform="_output/cf/cf\1.center">;

foreach pf,i in pcapfiles {
  (of[i],cf[i]) = angle4(pf);
}
```

Name outputs based on inputs

Parallelism for processing datasets

```
type pcapfile;
type angleout;
type anglecenter;

(angleout ofile, anglecenter cfile) angle4 (pcapfile ifile)
{
  app { angle4.sh --input @ifile --output @ofile --coords @cfile; }
}

pcapfile pcapfiles[]<filesys_mapper; prefix="pc", suffix=".pcap">;

angleout    of[] <structured_regexp_mapper;
                  source=pcapfiles,match="pc(.*)\.pcap",
                  transform="_output/of/of\1.angle">;

anglecenter cf[] <structured_regexp_mapper;
                  source=pcapfiles,match="pc(.*)\.pcap",
                  transform="_output/cf/cf\1.center">;
```

Name outputs based on inputs

```
foreach pf,i in pcapfiles {
  (of[i],cf[i]) = angle4(pf);
}
```

Iterate over dataset members in parallel

Coding your own “external” mapper

```
awk <angle-spool-1-2 '
BEGIN {
  server="gsiftp://tp-osg.ci.uchicago.edu//disks/ci-gpfs/angle";
 }
{   printf "[%d] %s/%s\n", i++, server, $0    }’

$ cat angle-spool-1-2
spool_1/anl2-1182294000-dump.1.167.pcap.gz
spool_1/anl2-1182295800-dump.1.170.pcap.gz
spool_1/anl2-1182296400-dump.1.171.pcap.gz
spool_1/anl2-1182297600-dump.1.173.pcap.gz
…
$ ./map1 | head
[0] gsiftp://tp-osg.ci.uchicago.edu//disks/ci-gpfs/angle/spool_1/anl2-1182294000-dump.
      1.167.pcap.gz
[1] gsiftp://tp-osg.ci.uchicago.edu//disks/ci-gpfs/angle/spool_1/anl2-1182295800-dump.
      1.170.pcap.gz
[2] gsiftp://tp-osg.ci.uchicago.edu//disks/ci-gpfs/angle/spool_1/anl2-1182296400-dump.
      1.171.pcap.gz
      …
```

Site selection and throttling

- Avoid overloading target infrastructure
- Base resource choice on current conditions and real response for *you*
- Balance this with space availability
- Things are getting more automated.

Clustering and Provisioning

- Can cluster jobs together to reduce grid overhead for small jobs
- Can use a provisioner
- Can use a provider to go straight to a cluster

Testing and debugging techniques

- Debugging
  - Trace and print statements
  - Put logging into your wrapper
  - Capture stdout/error in returned files
  - Capture glimpses of runtime environment
  - Kickstart data helps understand what happened at runtime
  - Reading/filtering swift client log files
  - Check what sites are doing with local tools - condor_q, qstat
- Log reduction tools tell you how your workflow behaved

Other Workflow Style Issues

- Expose or hide parameters
- One atomic, many variants
- Expose or hide program structure
- Driving a parameter sweep with readdata() - reads a csv file into struct[].
- Swift is not a data manipulation language - use scripting tools for that

Swift: Getting Started

- www.ci.uchicago.edu/swift
  - Documentation -> tutorials
- Get CI accounts
  - https://www.ci.uchicago.edu/accounts/
    - Request: workstation, gridlab, teraport
- Get a DOEGrids Grid Certificate
  - http://www.doegrids.org/pages/cert-request.html
    - Virtual organization: OSG / OSGEDU
    - Sponsor: Mike Wilde, wilde@mcs.anl.gov, 630-252-7497
- Develop your Swift code and test locally, then:
  - On PBS / TeraPort
  - On OSG: OSGEDU
- Use simple scripts (Perl, Python) as your test apps

Planned Enhancements

- Additional data management models
- Integration of provenance tracking
- Improved logging for troubleshooting
- Improved compilation and tool integration (especially with scripting tools and SDEs)
- Improved abstraction and descriptive capability in mappers
- Continual performance measurement and speed improvement

Swift: Summary

- Clean separation of logical/physical concerns
  - XDTM specification of logical data structures

+ Concise specification of parallel programs
  - SwiftScript, with iteration, etc.

+ Efficient execution (on distributed resources)
  - **Karajan**
    +Falkon:
    Grid interface, lightweight dispatch, pipelining, clustering, provisioning

+ Rigorous provenance tracking and query
  - Records provenance data of each job executed

→ **Improved usability and productivity**
  - Demonstrated in numerous applications

Acknowledgments

- Swift effort is supported in part by NSF grants OCI-721939 and PHY-636265, NIH DC08638, and the UChicago/Argonne Computation Institute
- The Swift team:
  - Ben Clifford, Ian Foster, Mihael Hategan, Veronika Nefedova, Ioan Raicu, Tibi Stef-Praun, Mike Wilde, Zhao Zhang, Yong Zhao
- Java CoG Kit used by Swift developed by:
  - Mihael Hategan, Gregor Von Laszewski, and many collaborators
- User contributed workflows and application use
  - I2U2, U.Chicago Molecular Dynamics, U.Chicago Radiology and Human Neuroscience Lab, Dartmouth Brain Imaging Center

References - Workflow

- Taylor, I.J., Deelman, E., Gannon, D.B. and Shields, M. eds. Workflows for e-Science, Springer, 2007
- SIGMOD Record Sep. 2005 Special Section on Scientific Workflows, http://www.sigmod.org/sigmod/record/issues/0509/index.html
- Zhao Y., Hategan, M., Clifford, B., Foster, I., vonLaszewski, G., Raicu, I., Stef-Praun, T. and Wilde, M Swift: Fast, Reliable, Loosely Coupled Parallel Computation IEEE International Workshop on Scientific Workflows 2007
- Stef-Praun, T., Clifford, B., Foster, I., Hasson, U., Hategan, M., Small, S., Wilde, M and Zhao,Y. Accelerating Medical Research using the Swift Workflow System Health Grid 2007
- Stef-Praun, T., Madeira, G., Foster, I., and Townsend, R. Accelerating solution of a moral hazard problem with Swift e-Social Science 2007
- Zhao, Y., Wilde, M. and Foster, I. Virtual Data Language: A Typed Workflow Notation for Diversely Structured Scientific Data. Taylor, I.J., Deelman, E., Gannon, D.B. and Shields, M. eds. Workflows for eScience, Springer, 2007, 258-278.
- Zhao, Y., Dobson, J., Foster, I., Moreau, L. and Wilde, M. A Notation and System for Expressing and Executing Cleanly Typed Workflows on Messy Scientific Data. SIGMOD Record 34 (3), 37-43
- Vöckler, J.-S., Mehta, G., Zhao, Y., Deelman, E. and Wilde, M., Kickstarting Remote Applications. 2nd International Workshop on Grid Computing Environments, 2006.
- Raicu, I., Zhao Y., Dumitrescu, C., Foster, I. and Wilde, M Falkon: a Fast and Light-weight tasK executiON framework Supercomputing Conference 2007

Additional Information

- www.ci.uchicago.edu/swift
  - Quick Start Guide:
    - http://www.ci.uchicago.edu/swift/guides/quickstartguide.php
  - User Guide:
    - http://www.ci.uchicago.edu/swift/guides/userguide.php
  - Introductory Swift Tutorials:
    - http://www.ci.uchicago.edu/swift/docs/index.php

Introduction to Grid Computing Tutorial Outline

I. Motivation and Grid Architecture
II. Grid Examples from Life Sciences
III. Grid Security
IV. Job Management: Running Applications
V. Data Management
VI. Open Science Grid and TeraGrid
VII. Workflow on the Grid
VIII. Next steps: Learning more, getting started

Conclusion: Why Grids?

- New approaches to inquiry based on
  - Deep analysis of huge quantities of data
  - Interdisciplinary collaboration
  - Large-scale simulation and analysis
  - Smart instrumentation
  - ***Dynamically assemble the resources to tackle a new scale of problem***
- Enabled by access to resources & services without regard for location & other barriers

Grids:
Because Science needs community …

- Teams organized around common goals
  - People, resource, software, data, instruments…
- With diverse membership & capabilities
  - Expertise in multiple areas required
- And geographic and political distribution
  - No location/organization possesses all required skills and resources
- Must adapt as a function of the situation
  - Adjust membership, reallocate responsibilities, renegotiate resources

What's next ?

- Take the self-paced course:

  opensciencegrid.org/OnlineGridCourse

- Contact us:

  eot@opensciencegrid.org

- Info for all aspects of Grid usage is at OSG Web site:

  opensciencegrid.org

Acknowledgements

(OSG, Globus, Condor teams and associates)

Gabrielle Allen, LSU CCT

Rachana Ananthakrishnan, ANL/Globus

Ben Clifford, UChicago CI

Jaime Frey, UWisconsin/Condor

Alain Roy, UWisconsin/Condor

Alex Sim, BNL