



Fast, Reliable, Loosely Coupled  
Parallel Computation

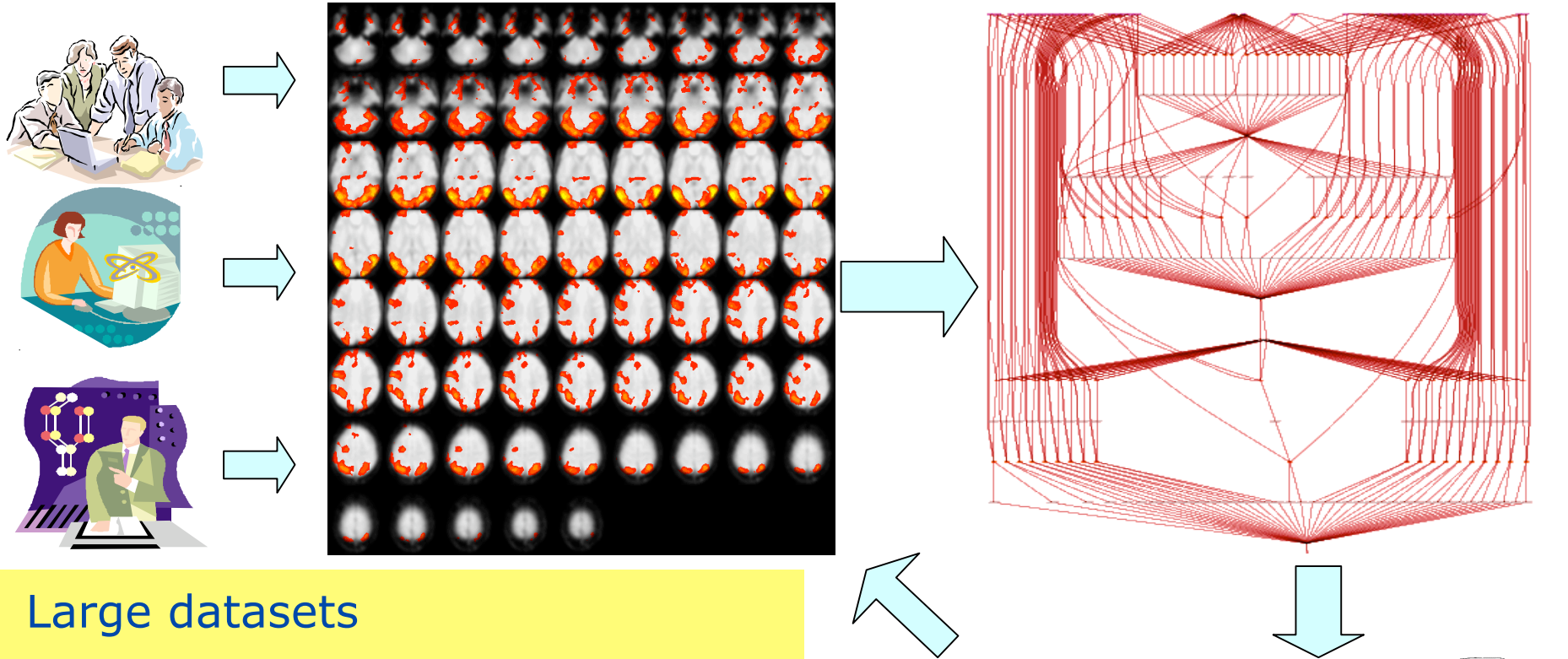
[www.ci.uchicago.edu/swift](http://www.ci.uchicago.edu/swift)

Joint work of **Ben Clifford, Ian Foster, Mihael Hategan, Veronika Nefedova, Ioan Raicu, Tibi Stef-Praun, Mike Wilde, Yong Zhao**

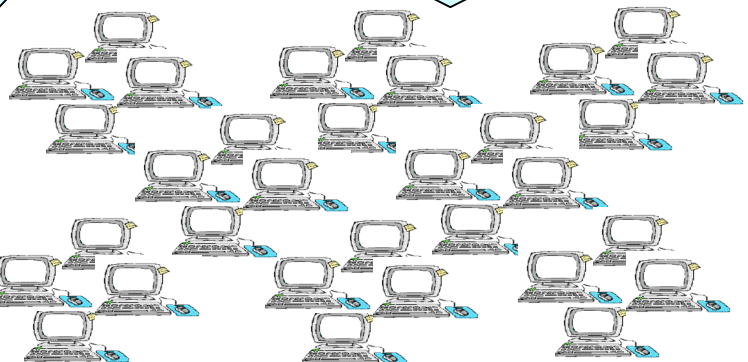
## Goals of using grids through scripting

- Provide an easy on-ramp to the grid
- Utilize massive resources with simple scripts
  - ◆ Leverage multiple grids like a workstation
- Empower script-writers to empower end users
- Future: Track and leverage provenance in the science process

# Case Study: Functional MRI (fMRI) Data Center



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



<http://www.fmridc.org> 3

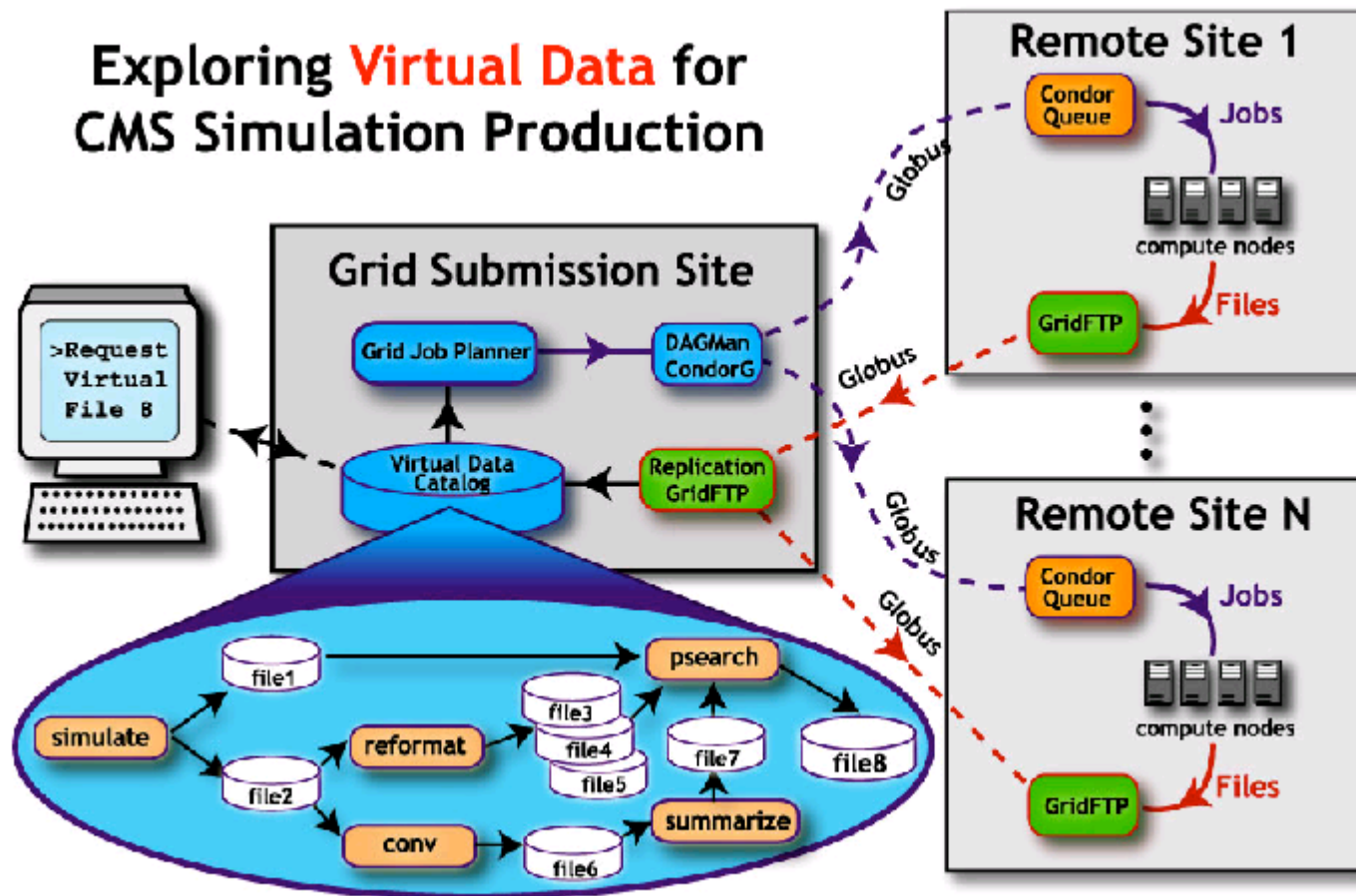
# Three Obstacles to Creating a Community Resource

- Accessing messy data
  - ◆ Idiosyncratic layouts & formats
  - ◆ Data integration a prerequisite to analysis
- Implementing complex computations
  - ◆ Expression, discovery, reuse of analyses
  - ◆ Hiding the complexity of distributed parallel systems
  - ◆ Scaling to large data, complex analyses
- Making analysis a community process
  - ◆ Collaboration on both data & programs
  - ◆ Provenance: tracking, query, application

# VDS – The Virtual Data System

- Introduced Virtual Data Language - VDL
  - ◆ A location-independent parallel language
- Several Planners
  - ◆ Pegasus: main production planner
  - ◆ Euryale: experimental “just in time” planner
  - ◆ GADU/GNARE – user application planner (D. Sulahke, Argonne)
- Provenance
  - ◆ Kickstart – app launcher and tracker
  - ◆ VDC – virtual data catalog

*Virtual Data* approach uses workflow to simplify grids by *abstracting* details.



# VDL/VDS Limitations

- Missing language features
  - ◆ Data typing & data mapping
  - ◆ Iterators and control-flow constructs
- Run time complexity in VDS
  - ◆ State explosion for data-parallel applications
  - ◆ Computation status hard to provide
  - ◆ Debugging information complex & distributed
- Performance
  - ◆ Still many runtime bottlenecks

# The **Swift** Solution

- Accessing messy data

XDTM

- ◆ Idiosyncratic layouts & formats
- ◆ Data integration a prerequisite to analysis

- Implementing complex computations

SwiftScript

- ◆ Expression, discovery, reuse of analyses
- ◆ Hiding complexity of distributed systems
- ◆ Scaling to large data, complex analyses

Karajan  
+Falkon

- Making analysis a community process

- ◆ Collaboration on both data & programs
- ◆ Provenance: tracking, query, application

VDC

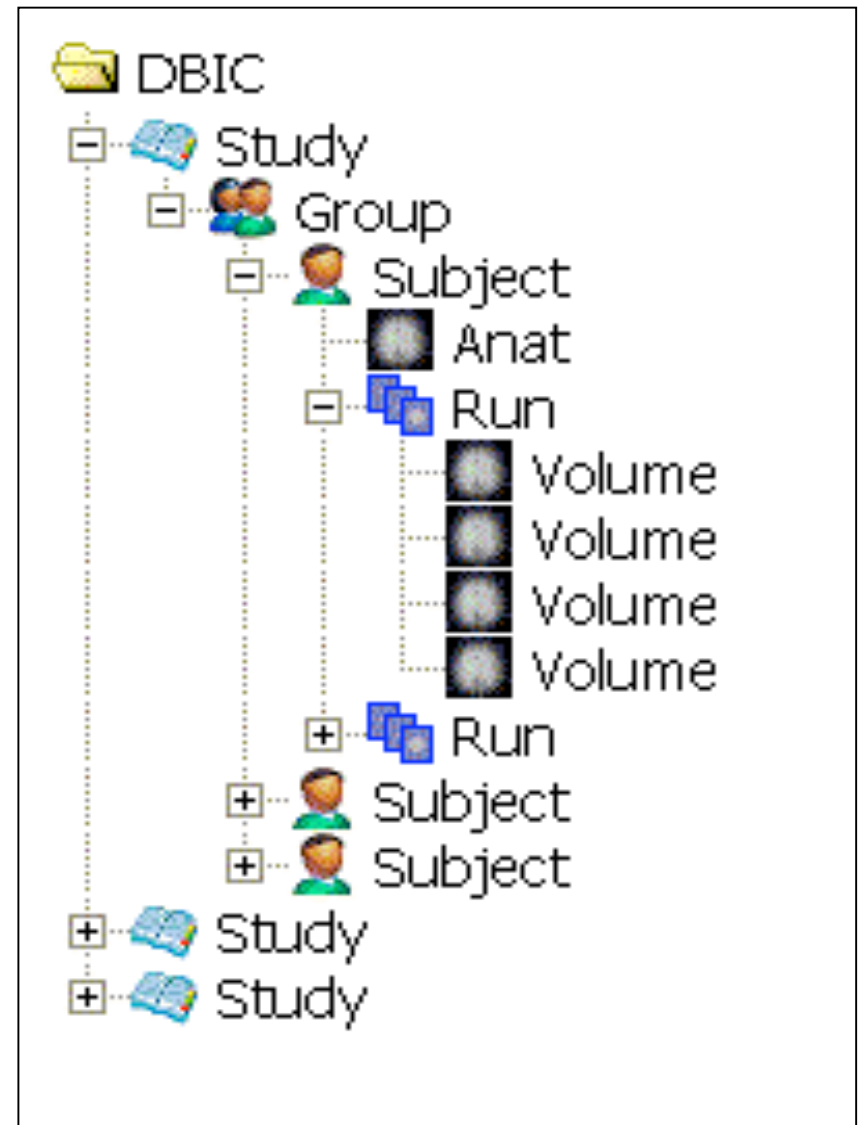
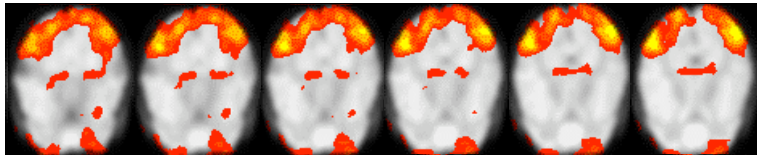


# Swift System

- 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
  - ◆ Lightweight threading, dynamic provisioning, Grid interfaces, pipelining, load balancing
- + Rigorous provenance tracking and query
  - ◆ Virtual data schema & automated recording
- **Improved usability and productivity**
  - ◆ Demonstrated in numerous applications

# 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 scale to 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

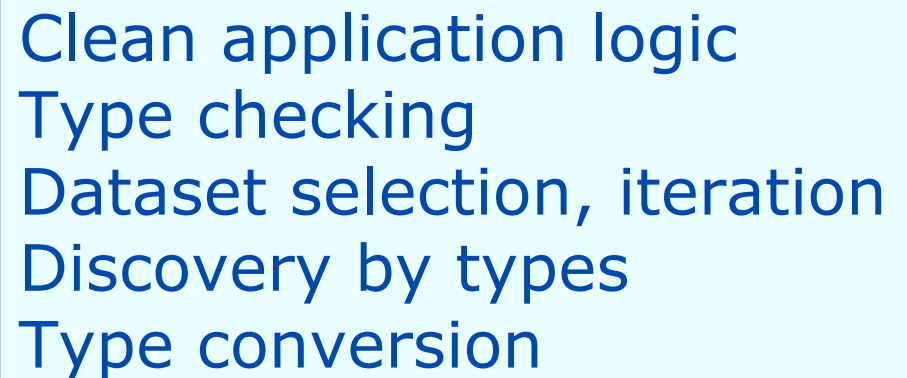
./group23/AA/04nov06aa/FUNCTIONAL:
-rw-r--r--  1 yongzh users   348 Nov  5 12:32 bold1_0001.hdr
-rw-r--r--  1 yongzh users 409600 Nov  5 12:32 bold1_0001.img
-rw-r--r--  1 yongzh users   348 Nov  5 12:32 bold1_0002.hdr
-rw-r--r--  1 yongzh users 409600 Nov  5 12:32 bold1_0002.img
-rw-r--r--  1 yongzh 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
```

## → XML Dataset Typing and Mapping (XDTM)

- Describe logical structure by **XML Schema**
  - ◆ Primitive scalar types: int, float, string, date, ...
  - ◆ Complex types (structs and arrays)
- Use **mapping descriptors** for mappings
  - ◆ How dataset elements are mapped to physical representations
  - ◆ External parameters (e. g. location)
- Use **XPath** for dataset selection
- Provide standard mapper implementations
  - ◆ String, File System, CSV File, etc.

# SwiftScript

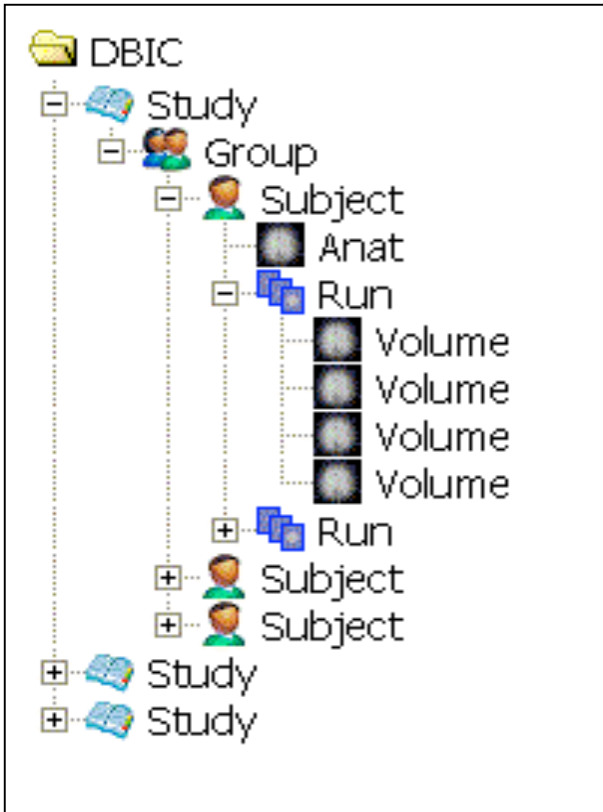
- Typed parallel programming notation
  - ◆ XDTM as data model and type system
  - ◆ Typed dataset and procedure definitions
- Scripting language
  - ◆ Implicit data parallelism
  - ◆ Program composition from procedures
  - ◆ Control constructs (foreach, if, while, ...)



Clean application logic  
Type checking  
Dataset selection, iteration  
Discovery by types  
Type conversion

**A Notation & System for Expressing and Executing Cleanly Typed Workflows on Messy Scientific Data [SIGMOD05]**

# fMRI Type Definitions in SwiftScript



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)  
{
```

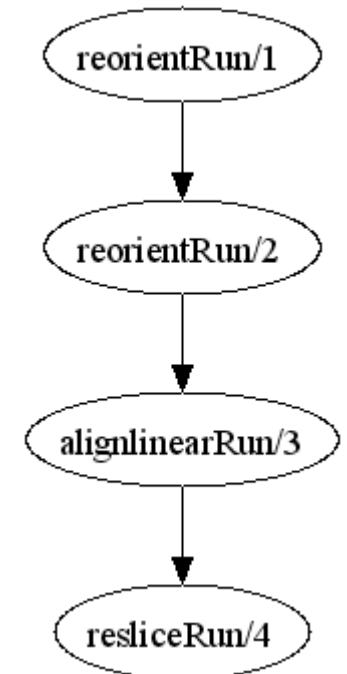
```
    Run yR = reorientRun( r , "y", "n" );  
    Run roR = reorientRun( yR , "x", "n" );  
    Volume std = roR.v[1];  
    AirVector roAirVec =  
        alignlinearRun(std, roR, 12, 1000, 1000, "81 3 3");  
    resliced = resliceRun( roR, roAirVec, "-o", "-k");
```

```
}
```

```
(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);  
    }  
}
```

```
}
```

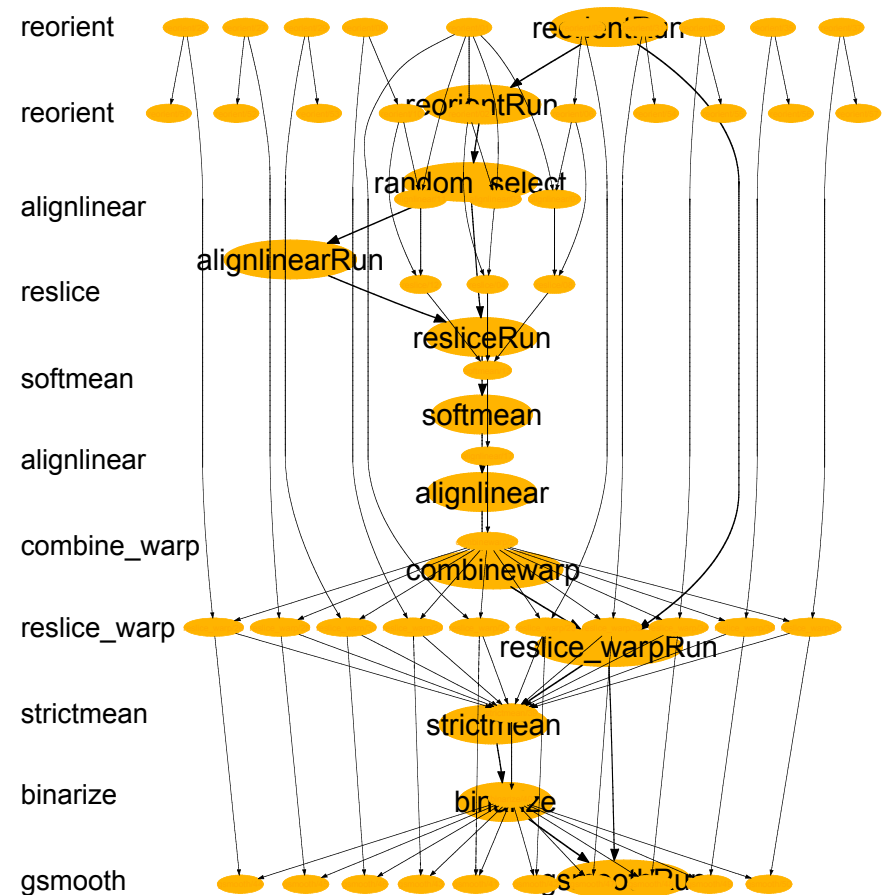


# SwiftScript Expressiveness

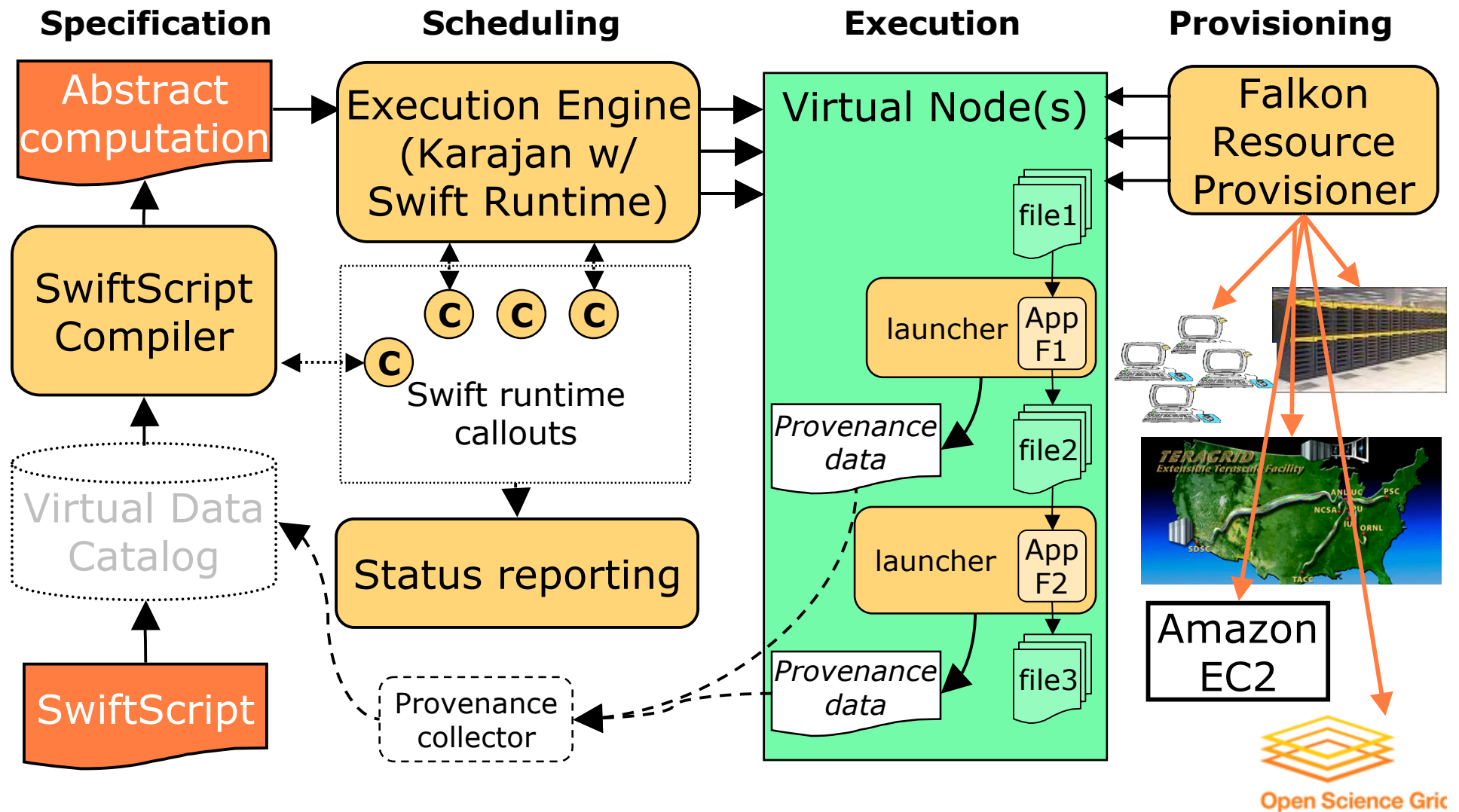
Lines of code with different workflow encodings

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

*AIRSN workflow:expanded:*



# Swift Architecture



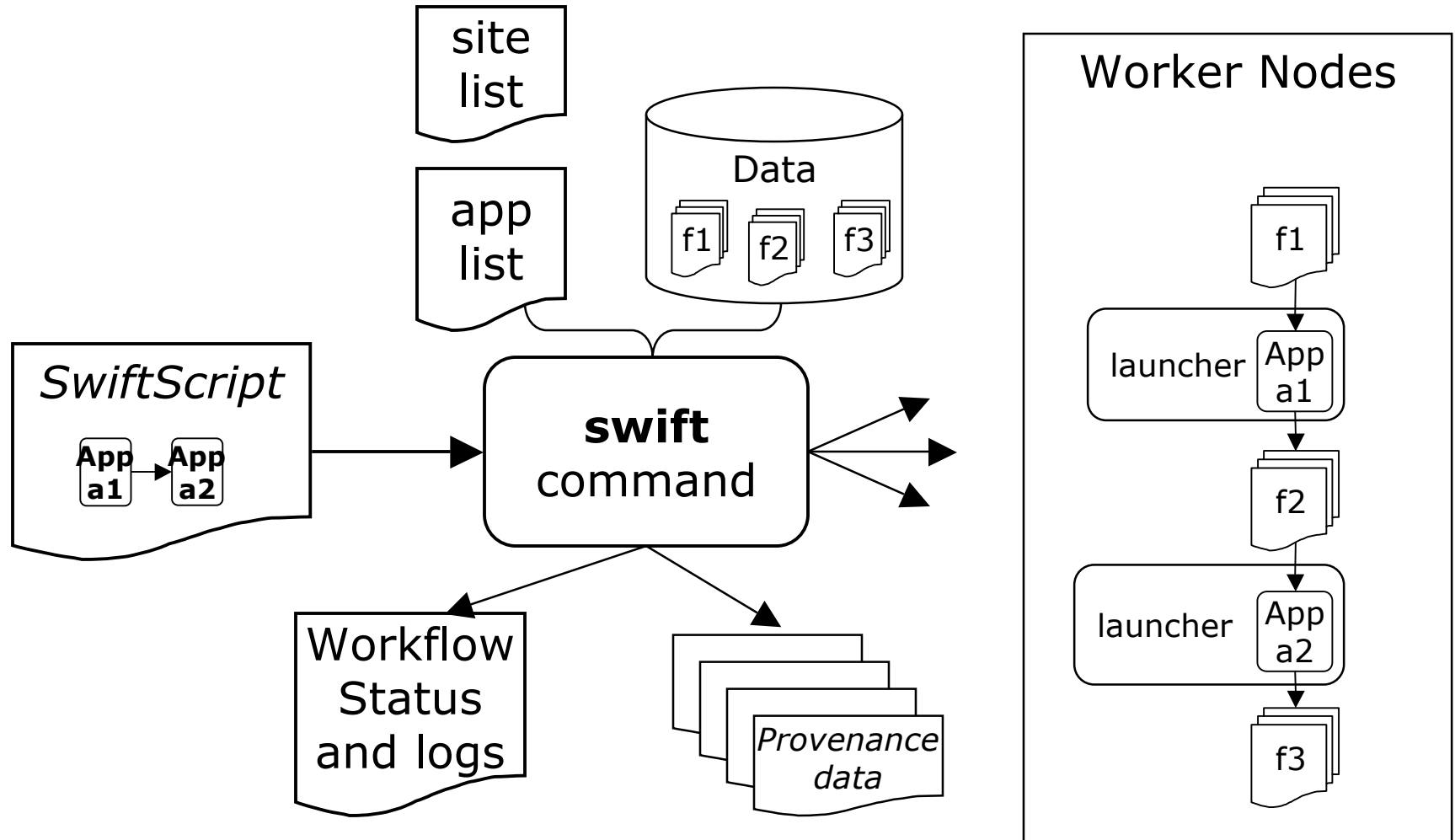
# Swift uses **Karajan** Workflow Engine

- Fast, scalable lightweight threading model
- Suitable constructs for control flow
- Flexible task dependency model
  - ◆ “Futures” enable pipelining
- Flexible provider model allows for use of different run time environments
  - ◆ Job execution and data transfer
  - ◆ Flow controlled to avoid resource overload
- Workflow client runs from a Java container

# Swift Runtime System

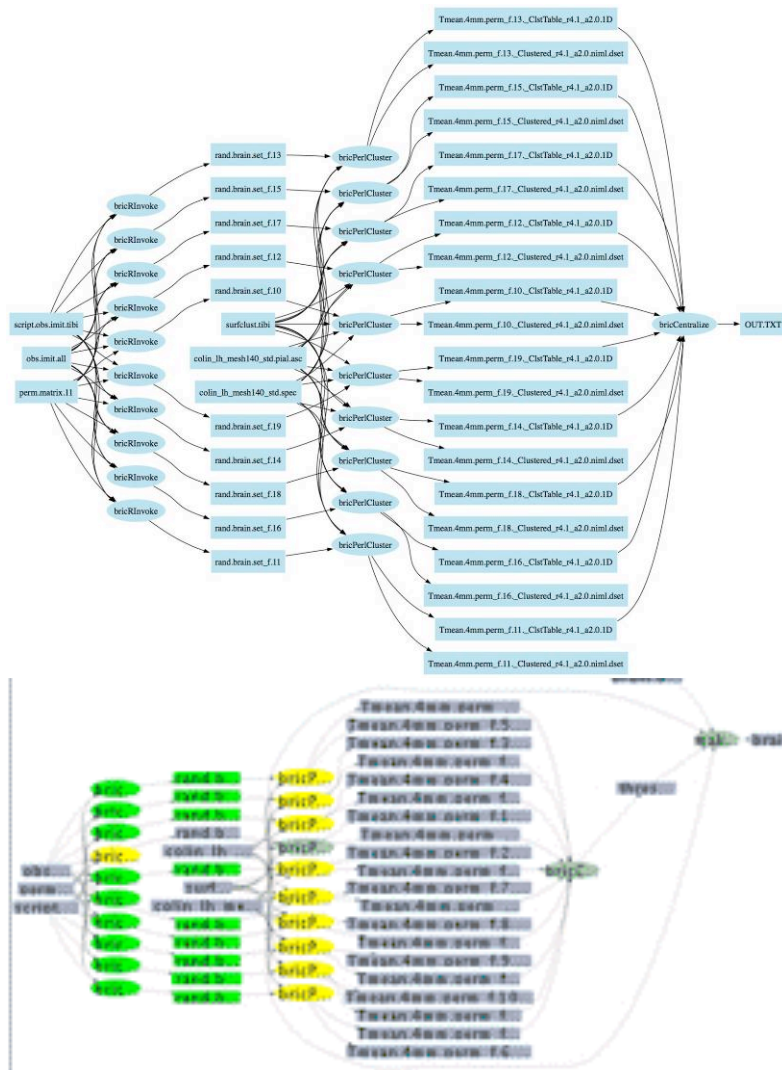
- Runtime system for SwiftScript
  - ◆ Translate programs into task graphs
  - ◆ Schedule, monitor, execute task graphs on local clusters and/or distributed Grid resources
  - ◆ Annotate data products with provenance metadata
- Grid scheduling and optimization
  - ◆ Lightweight execution engine: **Karajan**
  - ◆ **Falkon**: lightweight dispatch, dynamic provisioning
  - ◆ Grid execution: site selection, data movement
  - ◆ Caching, pipelining, clustering, load balancing
  - ◆ Fault tolerance, exception handling

# Using Swift



# Application example: ACTIVAL: Neural activation validation

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 brainMeasurements{}
type precomputedPermutations{}
type brainClusterTable {}
type brainDatasets{ brainDataset b[]; }
type brainClusters{ brainClusterTable c[]; }

type fullBrainData {}
type fullBrainSpecs {}
type brainDataset {}
```

### // 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 likelihoods

```
(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">;
```

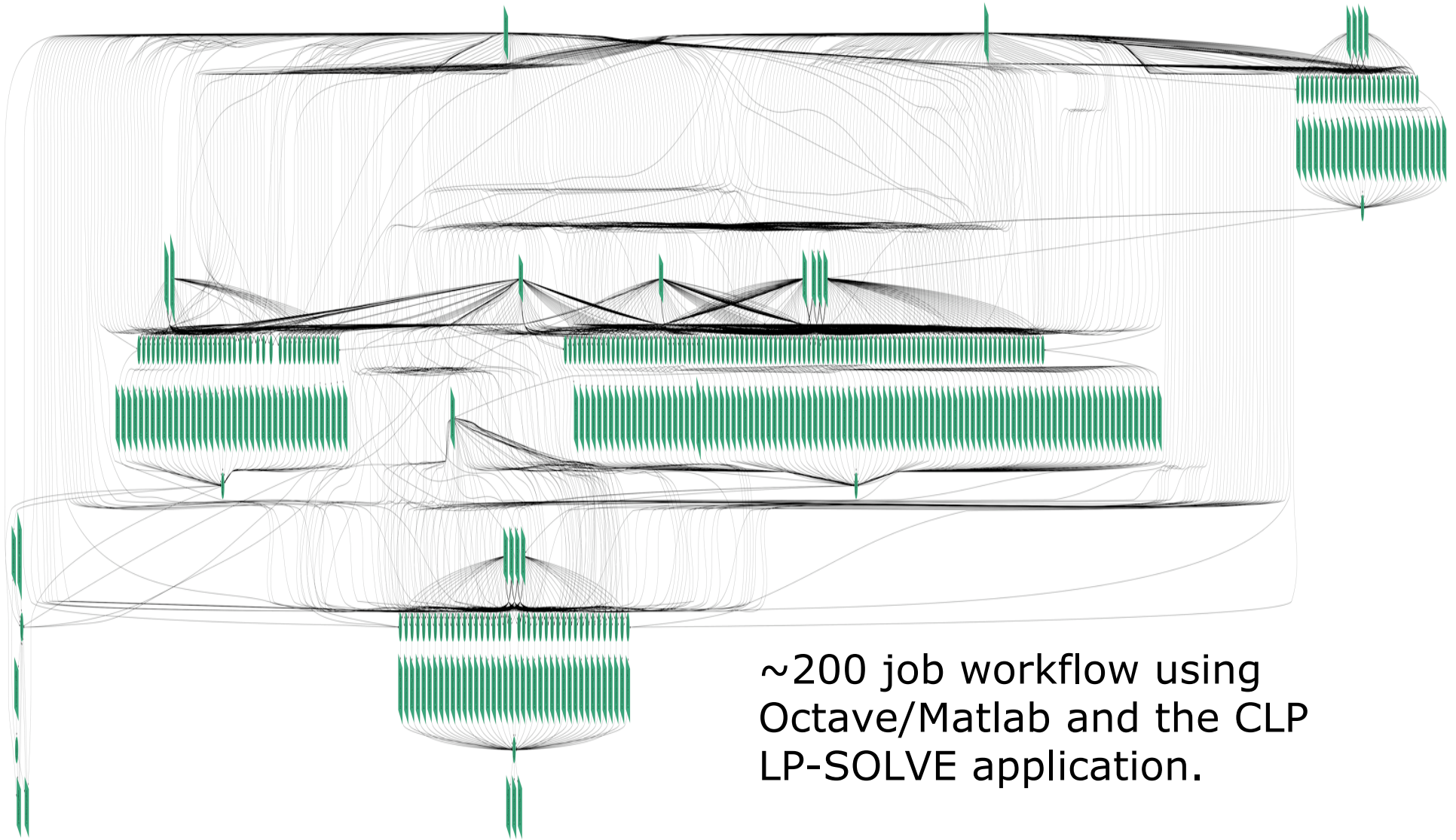
## // Main program – executes the entire workflow

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

clusterThresholdsTable = bricCentralize (randCluster.c);

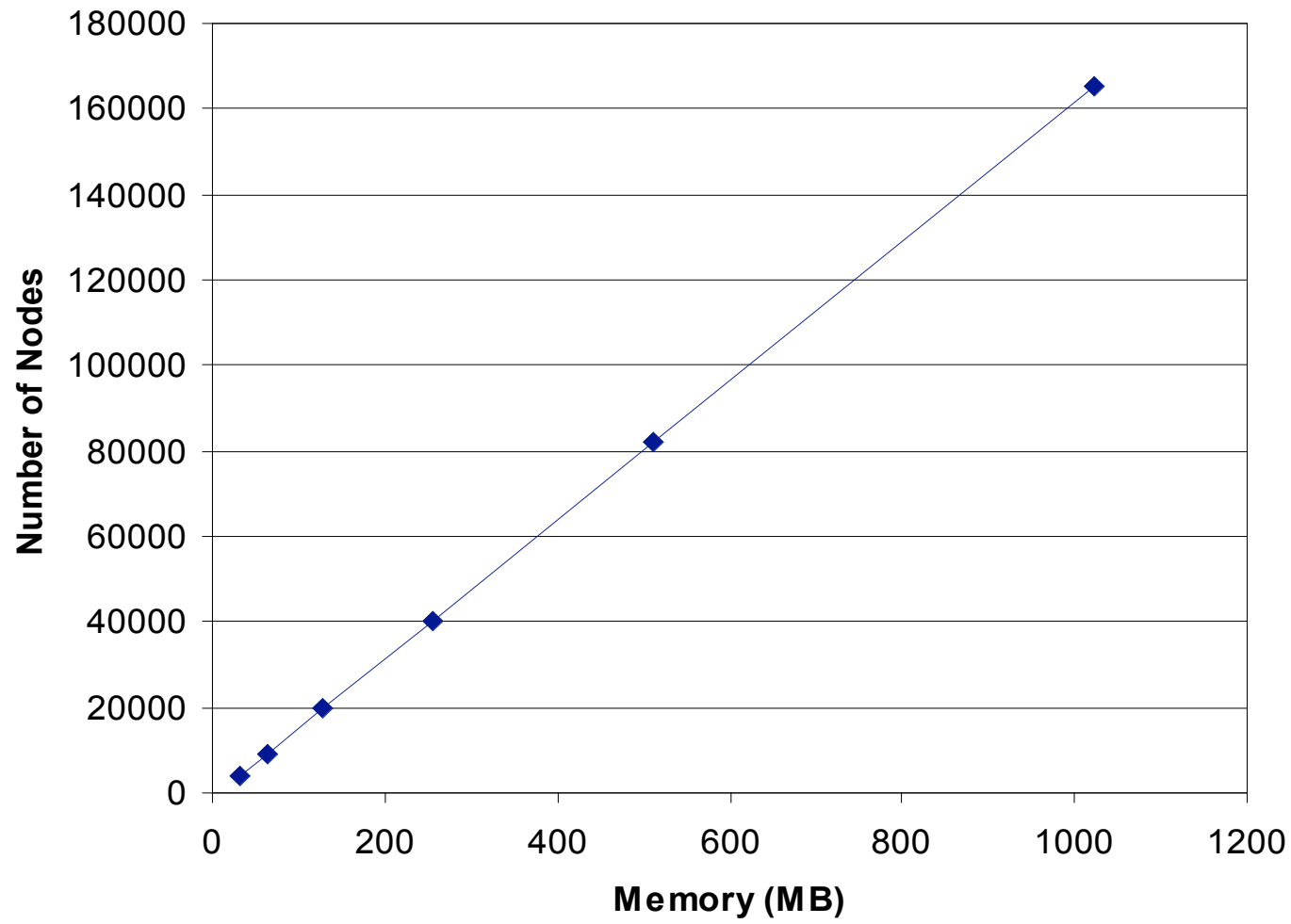
brainResult = makebrain(origBrain,clusterThresholdsTable,brainFile,specFile);
```

# Swift Application: Economics “moral hazard” problem

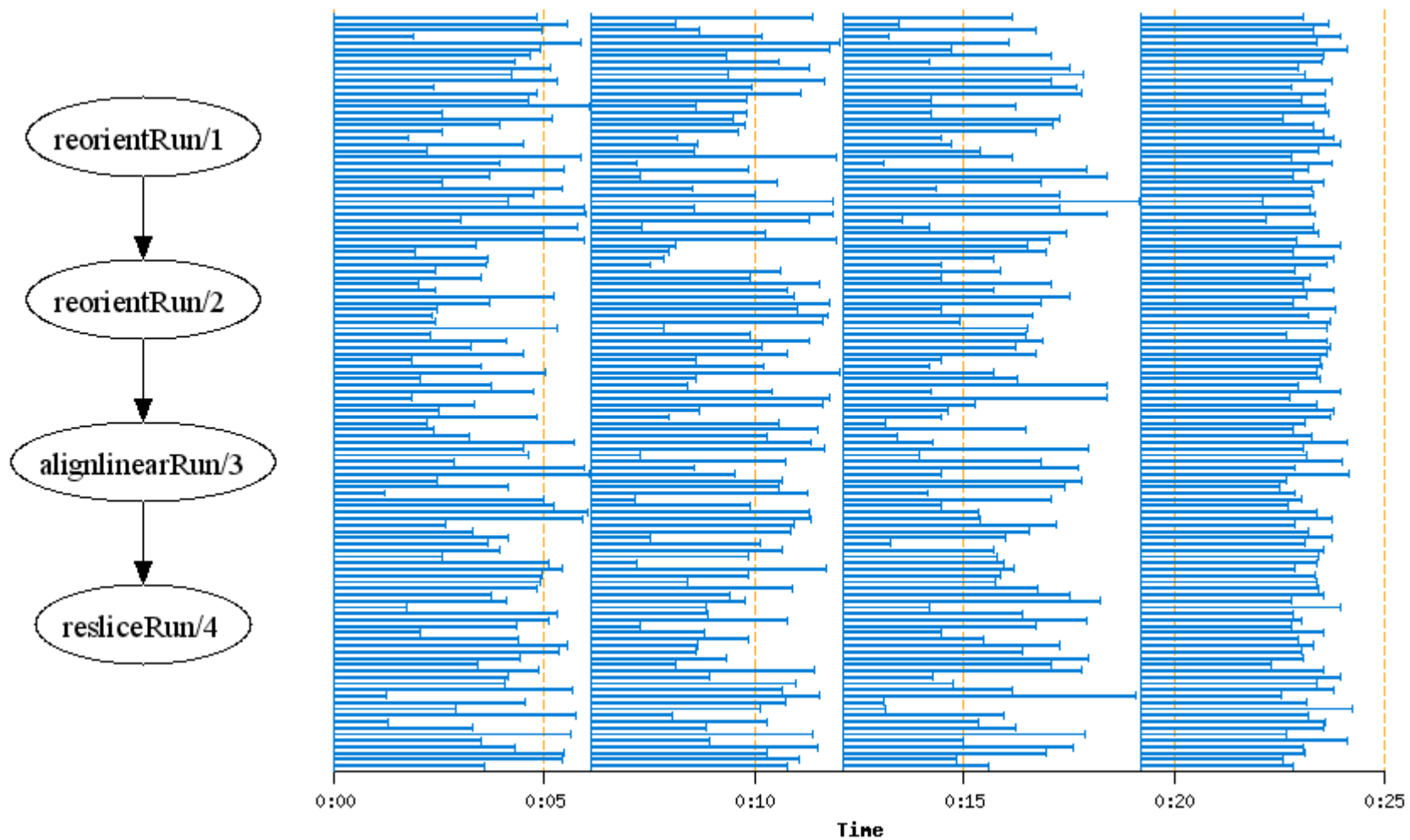


~200 job workflow using  
Octave/Matlab and the CLP  
LP-SOLVE application.

# Lightweight Threading - Scalability

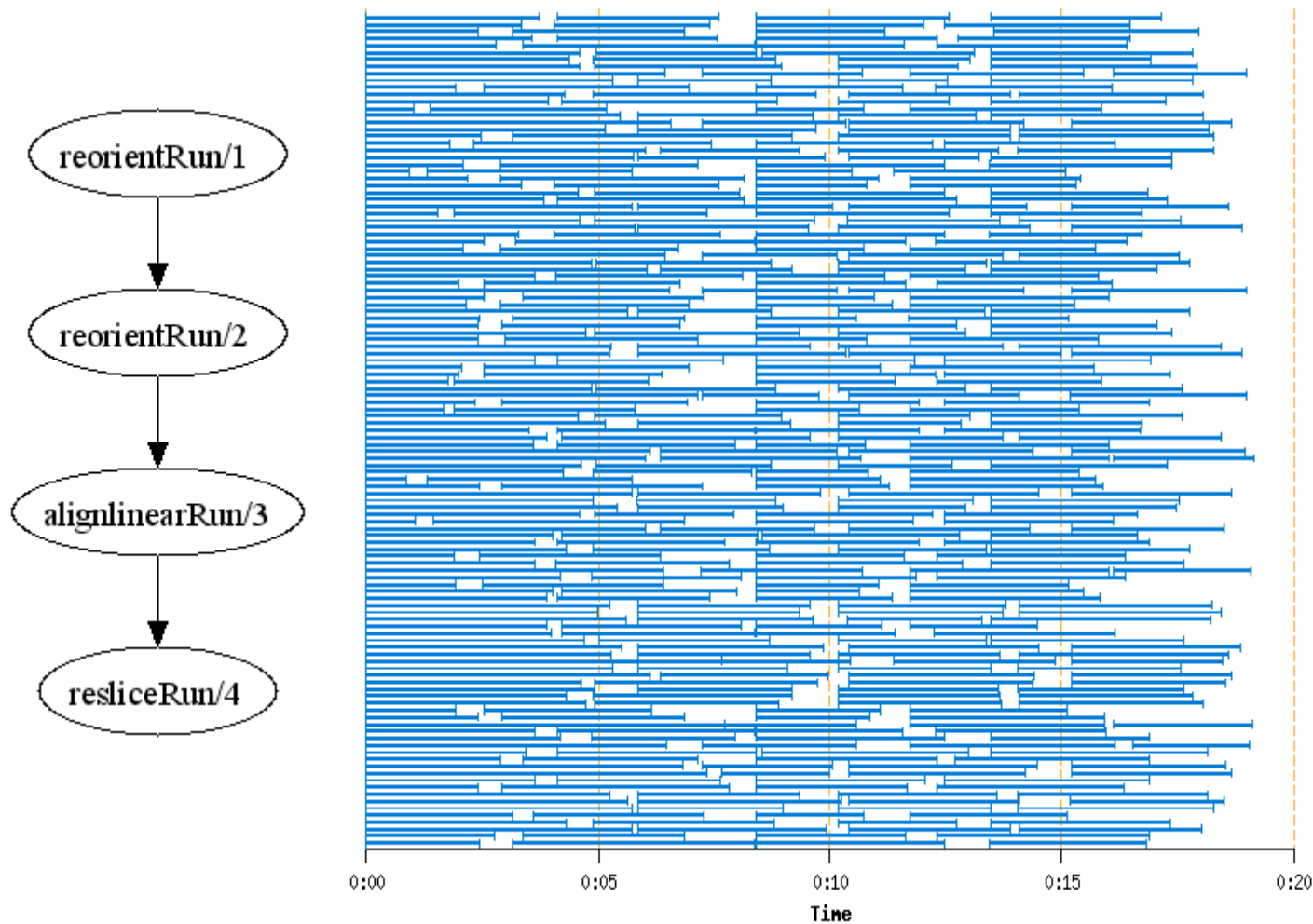


# Karajan Futures Enable Pipelining



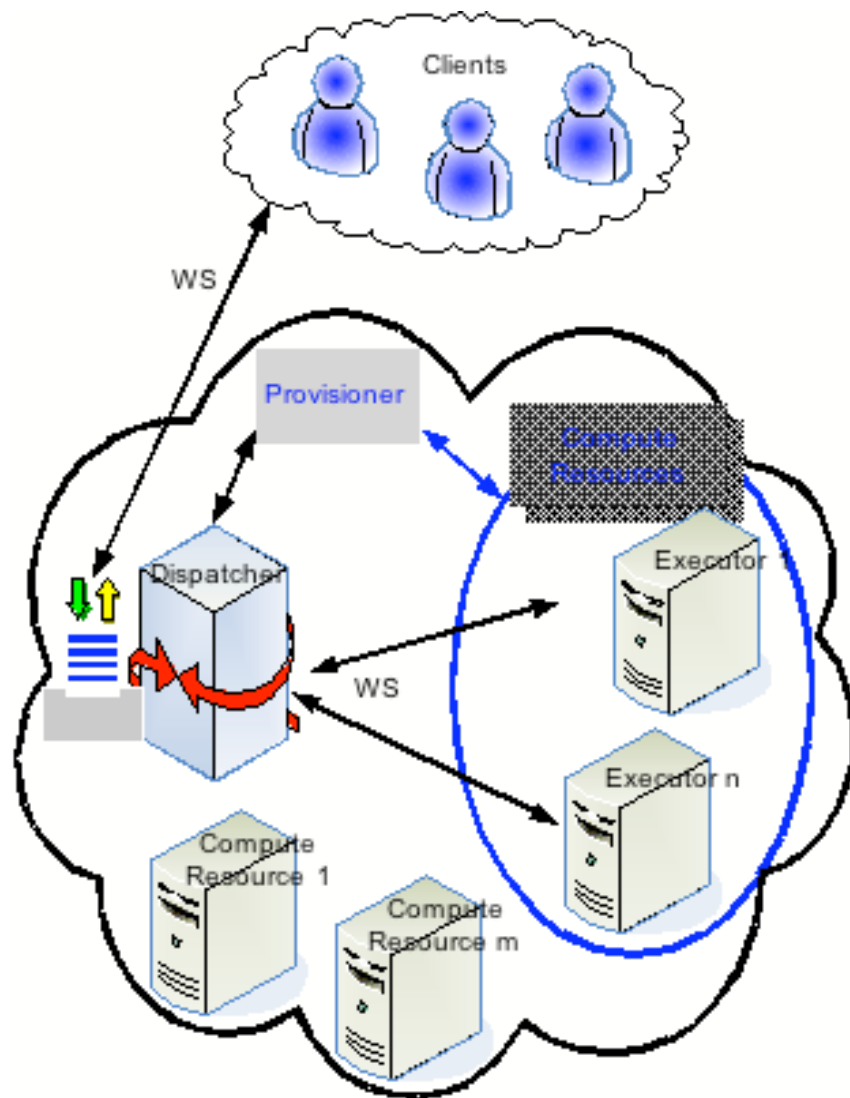
(Dispatch is performed here via GRAM+PBS)

# Karajan Futures Enable Pipelining



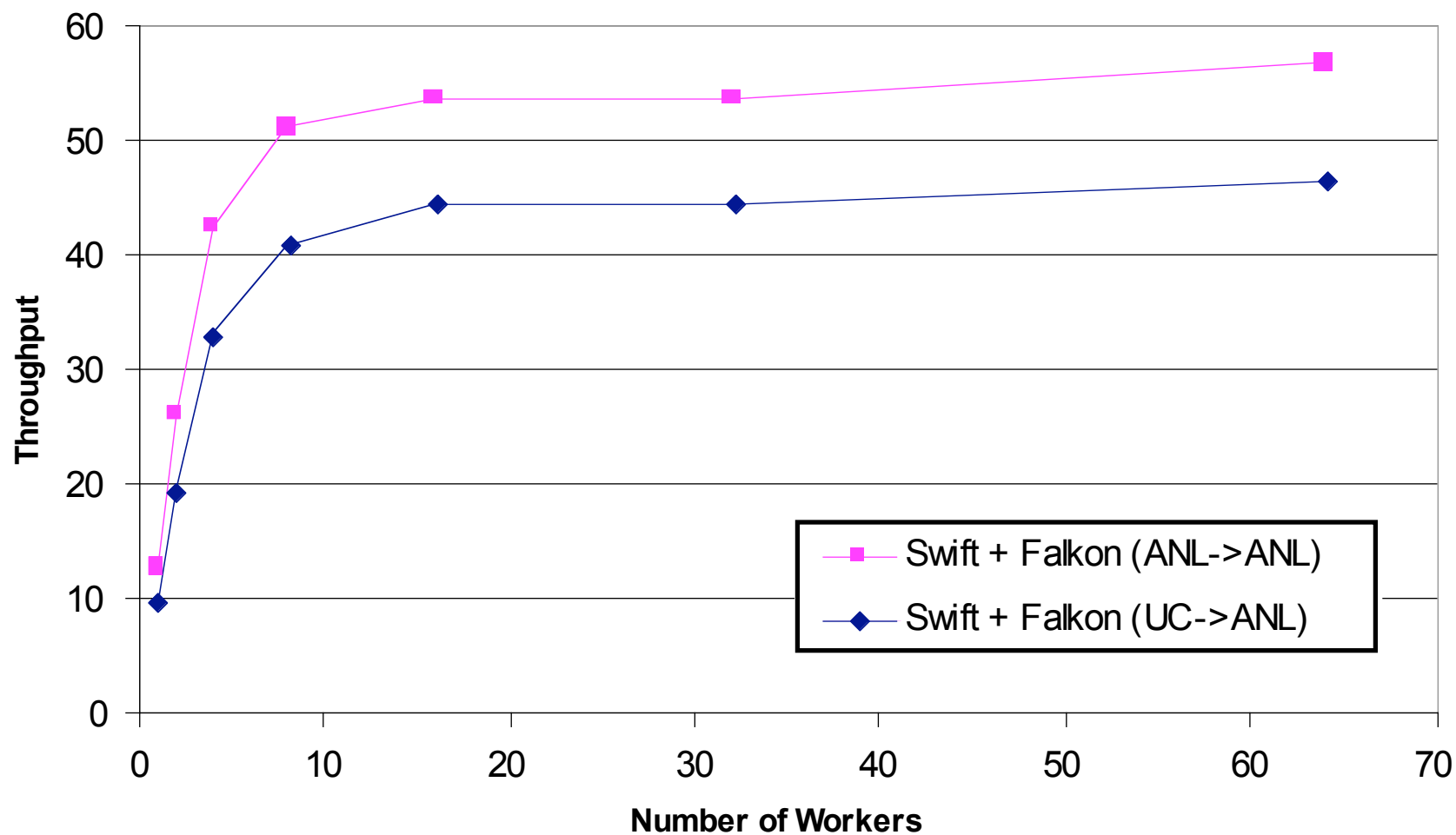
(Dispatch is performed here via GRAM+PBS)

# Swift Can Use **Falkon** Lightweight Execution Service

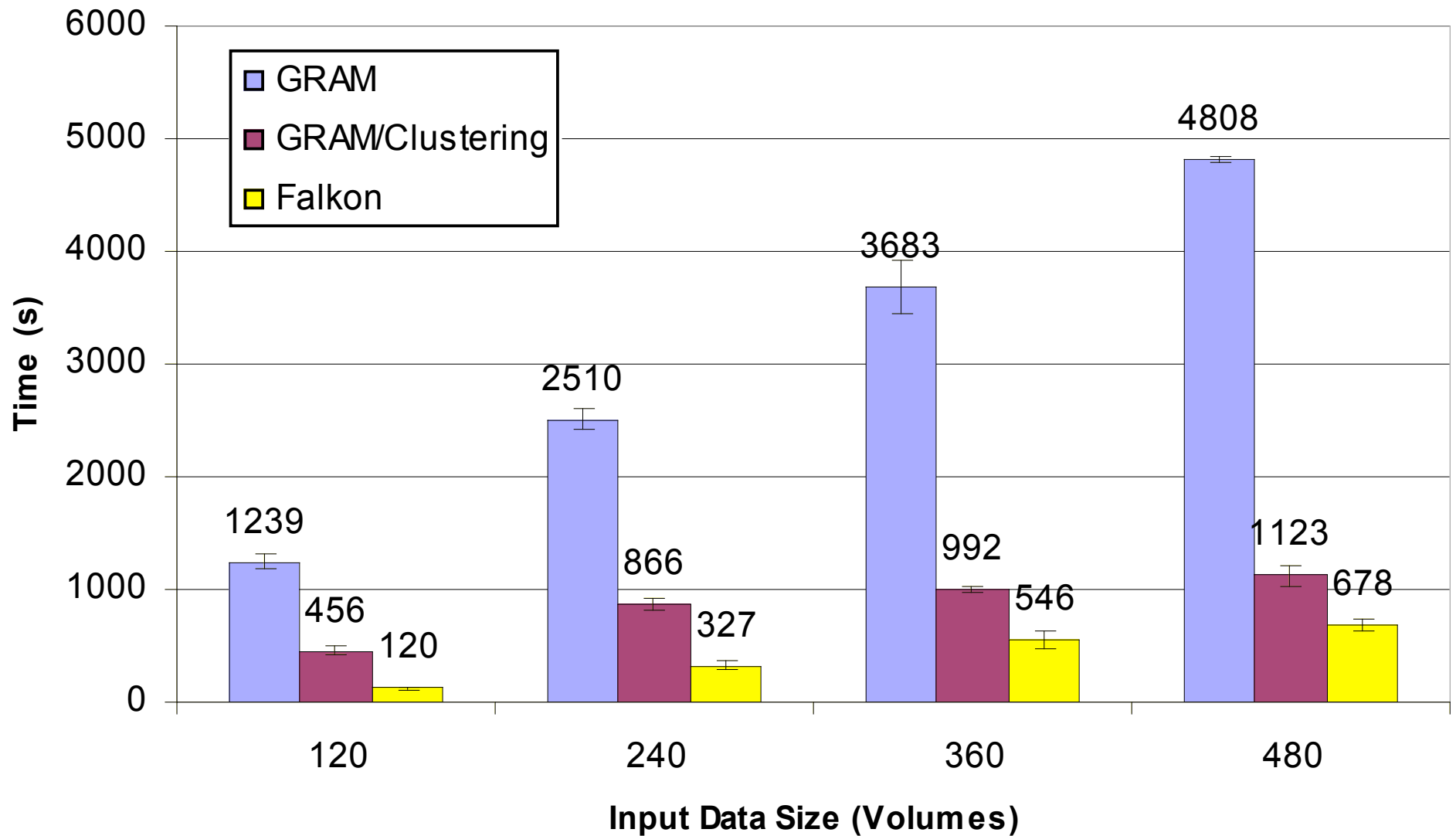


- **Falkon dynamic provisioner:**
  - ◆ Monitors **demand** (incoming user requests)
  - ◆ Manages **supply**: selects resources; creates executors (via Globus GRAM+LRM)
  - ◆ Various decision strategies for acquisition and release
- **Falkon executor**
  - ◆ Streamlined task dispatch
  - ◆ 440 tasks/sec max
- Dispatch to other executors also supported—e.g., GRAM

# Swift Throughput via Falkon



# Swift Application Performance: fMRI Task Graph

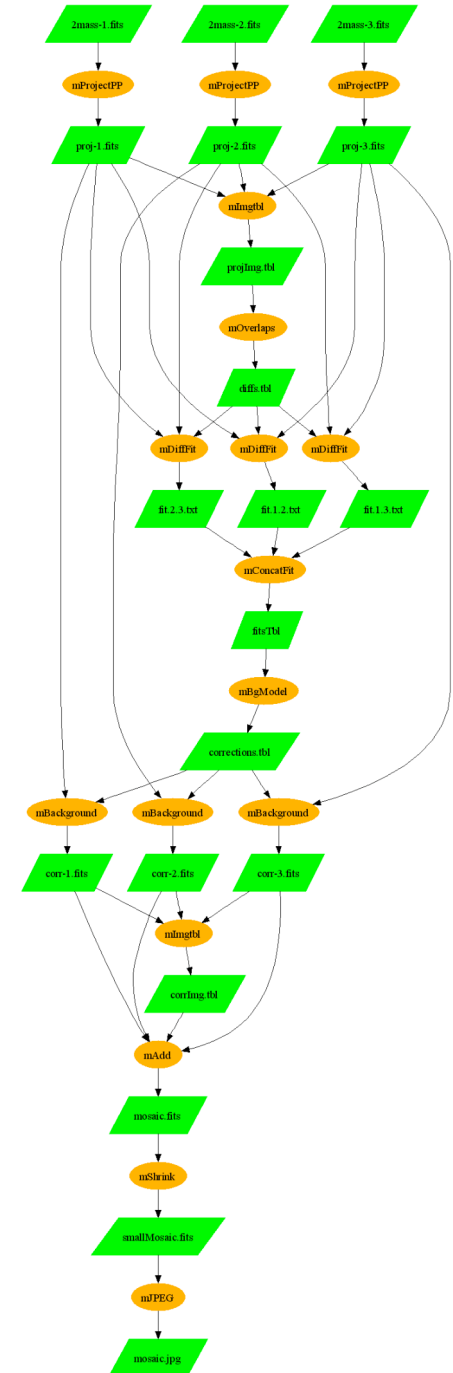
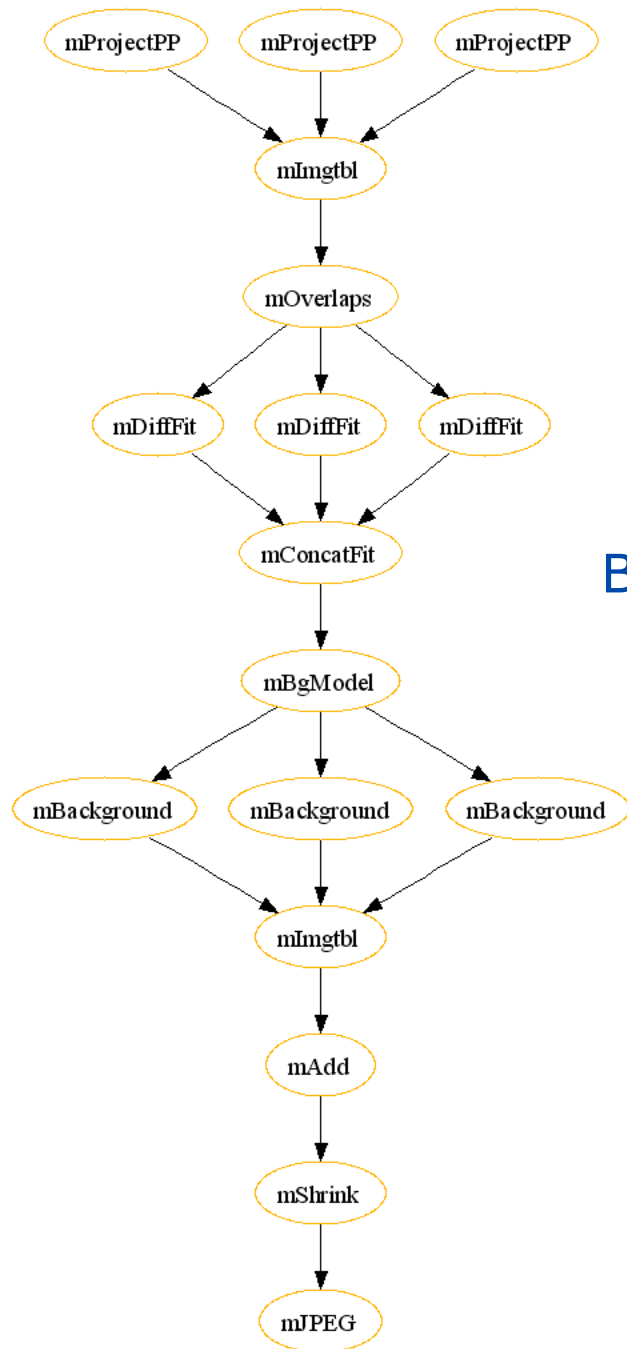




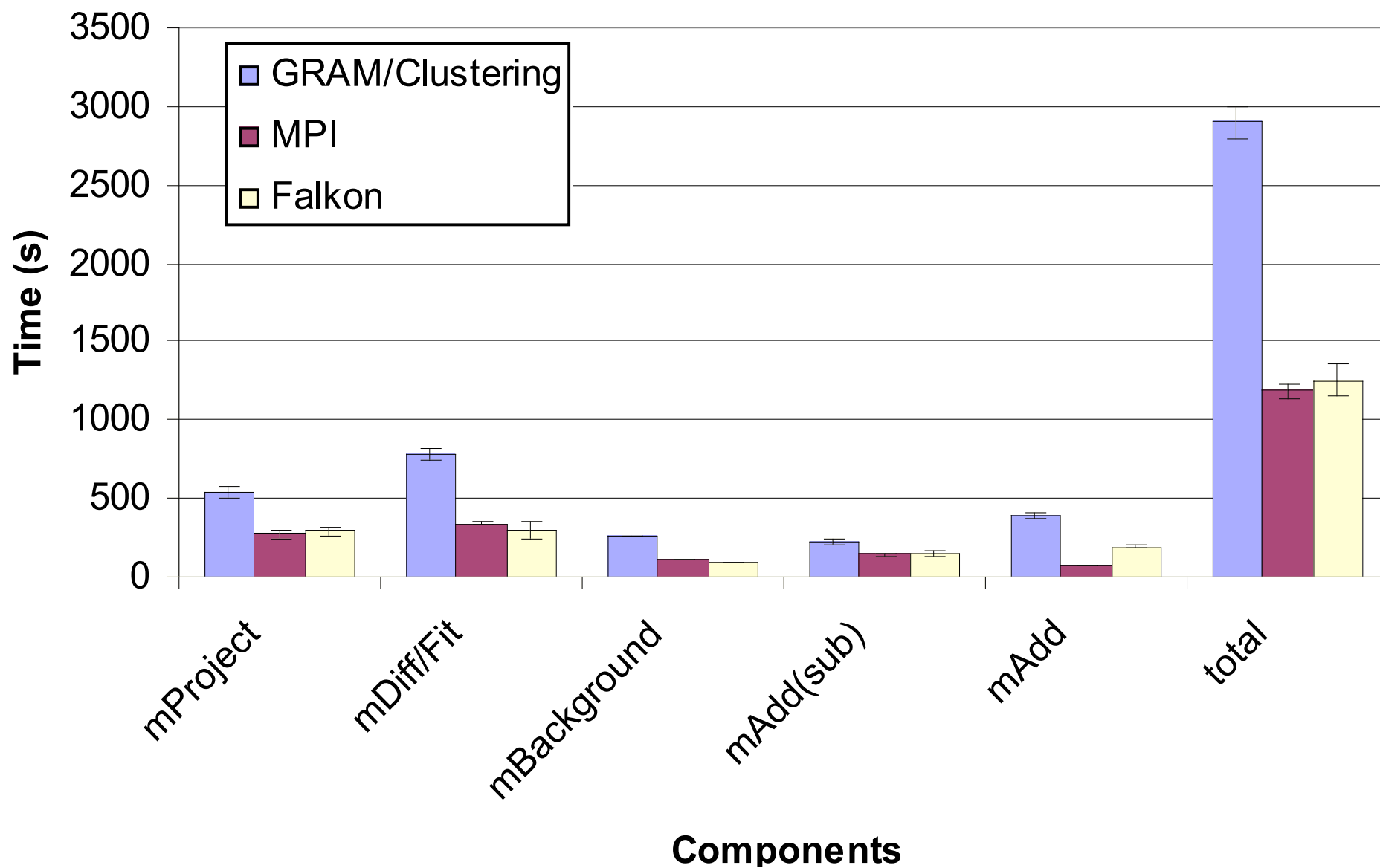
# Swift Application



B. Berriman, J. Good (Caltech)  
J. Jacob, D. Katz (JPL)



# Montage



# Other Swift Applications Include ...

Application	#Jobs/computation	Levels
ATLAS* HEP Event Simulation	500K	1
fMRI AIRSN Image Processing	100s	12
FOAM* Ocean/Atmosphere Model	2000 (250 8-CPU jobs)	3
GADU* Genomics: (14M seq. analyzed)	40K	4
fMRI Aphasia Study	500	4
NVO/NASA Montage	1000s	16
QuarkNet/I2U2*+ Physics Science Education	10s	3-6
RadCAD: Radiology Classifier Training	1000s	5
SIDGrid: EEG Wavelet Proc, Gaze Analysis, ...	100s	20
SDSS* Coadd, Cluster Search	40K, 500K	2, 8

\* Using predecessor **Virtual Data System** (VDS)

+ Collaborative science learning & education: 18 experiments,  
51 universities/labs, 500+ schools, 100,000 students

# Future Work

- XDTM
  - ◆ Support for services as well as applications
  - ◆ Greater abstraction in mappers; databases
- SwiftScript
  - ◆ Exceptions
  - ◆ Event-driven dispatch & execution
- Falcon
  - ◆ Scale to more resources; data caching
  - ◆ Support for service workloads
- VDC
  - ◆ Integration into Swift; collaboration support
  - ◆ Experiments at scale

# Acknowledgements

- Swift effort is supported by NSF (I2U2, iVDGL), NIH, UChicago/Argonne Computation Institute
- Swift team
  - ◆ Ben Clifford, Ian Foster, Mihael Hategan, Veronika Nefedova, Ioan Raicu, Mike Wilde, Yong Zhao
- Java CoG Kit
  - ◆ Mihael Hategan, Gregor Von Laszewski, and many collaborators
- User contributed workflows and application use
  - ◆ I2U2, ASCI Flash, U.Chicago Molecular Dynamics, U.Chicago Radiology, Human Neuroscience Lab

# 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
  - ◆ Virtual data schema & automated recording
- **Improved usability and productivity**
  - ◆ Demonstrated in numerous applications

<http://www.ci.uchicago.edu/swift>

Thank You!

Questions???



# Fast, Reliable, Loosely Coupled Parallel Computation

For more info:

[www.ci.uchicago.edu/swift](http://www.ci.uchicago.edu/swift)

[swift-user@ci.uchicago.edu](mailto:swift-user@ci.uchicago.edu)

Computation Institute  
University of Chicago

Joint work of **Ben Clifford, Ian Foster, Mihael Hategan, Veronika Nefedova, Ioan Raicu, Tibi Stef-Praun, Mike Wilde, Yong Zhao**