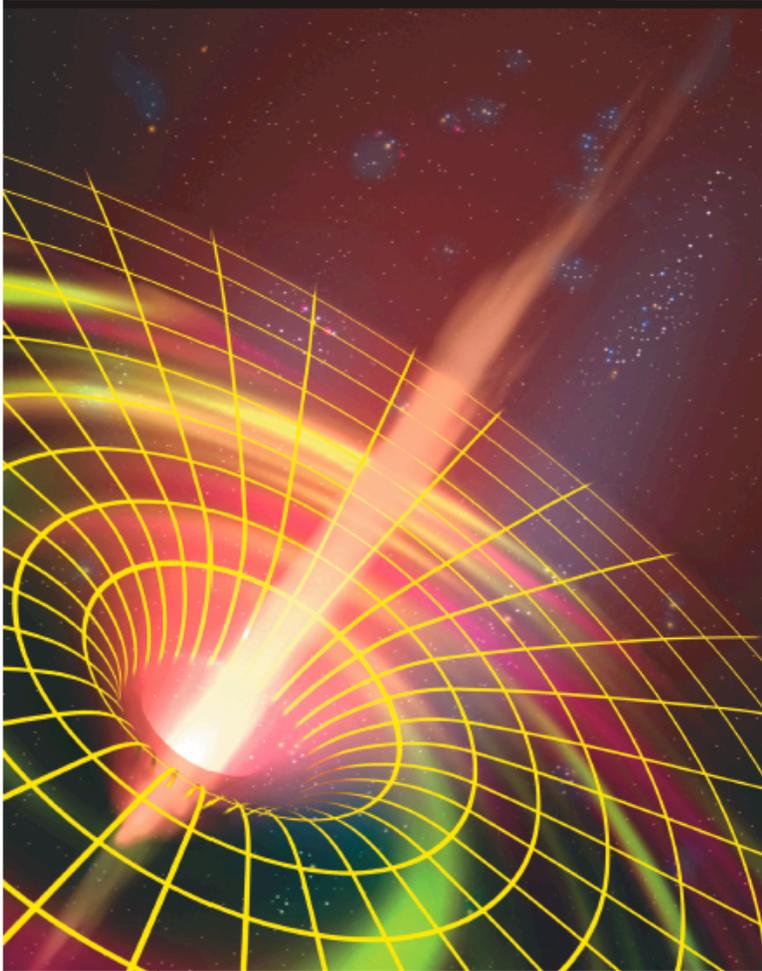


# *Parallel Scripting for Science Applications at the Petascale and Beyond*

Mike Wilde  
wilde@mcs.anl.gov

Computation Institute, University of Chicago  
and Argonne National Laboratory

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



# PARALLEL SCRIPTING FOR APPLICATIONS AT THE PETASCALE AND BEYOND

Michael Wilde, Ian Foster, Kamil Iskra, and Pete Beckman,  
*University of Chicago and Argonne National Laboratory*

Zhao Zhang, Allan Espinosa, Mihael Hategan, and Ben Clifford, *University of Chicago*

Ioan Raicu, *Northwestern University*

# Overview

- Parallel scripting as a model for large-scale computing
- An architecture for petascale parallel scripting
  - Swift
  - Collective data management
  - Resource provisioning and fast task dispatch
  - POSIX operating systems
- Application examples

# Swift is...

- A language for writing scripts that:
  - Process **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

# A simple Swift script

```
1 type imagefile; // Declare a "file" type.
2
3 app (imagefile output) flip (imagefile input) {
4 {
5     convert "-rotate" 180 @input @output ;
6 }
7
8 imagefile stars <"orion.2008.0117.jpg">;
9 imagefile flipped <"output.jpg">;
10
11 flipped = flip(stars);
```

# Parallelism via foreach { }

```
1 type imagefile; // Declare a "file" type.  
2  
3 (imagefile output) flip(imagefile input) {  
4   app {  
5     convert "-rotate" "180" @input @output;  
6   }  
7 }
```

```
8  
9 imagefile observations[ ] <simple_mapper; prefix="orion">;  
10 imagefile flipped[ ] <simple_mapper; prefix="orion-flipped">;
```

***Map inputs from local directory***

***Name outputs based on index***

```
11  
12  
13  
14 foreach obs,i in observations {  
15   flipped[i] = flip(obs);  
16 }
```

***Process all dataset members in parallel***

# Why script in Swift?

- Write scripts that are high-level, simpler, and location-independent: run anywhere
  - Higher level of abstraction makes a workflow script more portable than “ad-hoc” scripting
- Coordinate execution on *many* resources over *long time periods*
  - This is very complex to do manually – Swift automates it
- Enables restart of long running scripts
  - Swift tracks jobs in a parallel script completed

# 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** are **mapped** into this data model as arrays and structures
- 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

# *Application: 3<sup>o</sup> Protein structure prediction*

```
1. type Fasta;           // Primary protein sequence file in FASTA format
2. type SecSeq;         // Secodary structure file
3. type RamaMap;       // “Ramachandra” mapping info files
4. type RamaIndex;
5. type ProtGeo;       // PDB-format file – protein geometry: 3D atom coords
6. type SimLog;
7.
8. type Protein {       // Input file struct to protein simulator
9.     Fasta fasta;     // sequence to predict structure of
10.    SecSeq secseq;   // Initial secondary structure to use
11.    ProtGeo native; // 3D structure from experimental data when known
12.    RamaMap map;
13.    RamaIndex index;
14. }
15.
16. type PSimCf {       // Science configuration parameters to simulator
17.     float st;
18.     float tui;
19.     float coeff;
20. }
21.
22. type ProtSim {      // Output file struct from protein simulator
23.     ProtGeo pgeo;
24.     SimLog log;
25. }
```

# *Protein structure prediction*

```
1. app (ProtGeo pgeo) predict (Protein pseq)
2. {
3.   PSim @pseq.fasta @pgeo;
4. }
5.
6. (ProtGeo pg[ ]) doRound (Protein p, int n) {
7.   foreach sim in [0:n-1] {
8.     pg[sim] = predict(p);
9.   }
10. }
11.
12. Protein p <ext; exec="Pmap", id="1af7">;
13. ProtGeo structure[ ];
14. int nsim = 10000;
15. structure = doRound(p, nsim);
```

# *Protein structure prediction*

```
1 (ProtSim psim[ ]) doRoundCf (Protein p, int n, PSimCf cf) {
2   foreach sim in [0:n-1] {
3     psim[sim] = predictCf(p, cf.st, cf.tui, cf.coeff );
4   }
5 }

6 (boolean converged) analyze( ProtSim prediction[ ], int r, int numRounds)
7 {
8   if( r == (numRounds-1) ) {
9     converged = true;
10  }
11  else {
12    converged = false;
13  }
14 }
```

# *Protein structure prediction*

```
1. ItFix( Protein p, int nsim, int maxr, float temp, float dt)
2. {
3.     ProtSim prediction[ ][ ];
4.     boolean converged[ ];
5.     PSimCf config;
6.
7.     config.st = temp;
8.     config.tui = dt;
9.     config.coeff = 0.1;
10.
11.     iterate r {
12.         prediction[r] =
13.             doRoundCf(p, nsim, config);
14.         converged[r] =
15.             analyze(prediction[r], r, maxr);
16.     } until ( converged[r] );
17. }
```

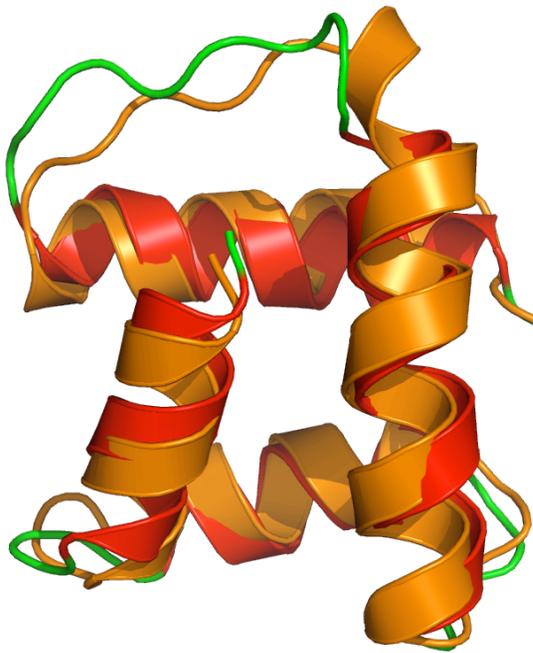
# *Protein structure prediction*

```
1. Sweep( )
2. {
3.   int nSim = 1000;
4.   int maxRounds = 3;
5.   Protein pSet[ ] <ext; exec="Protein.map">;
6.   float startTemp[ ] = [ 100.0, 200.0 ];
7.   float delT[ ] = [ 1.0, 1.5, 2.0, 5.0, 10.0 ];
8.   foreach p, pn in pSet {
9.     foreach t in startTemp {
10.      foreach d in delT {
11.        ItFix(p, nSim, maxRounds, t, d);
12.      }
13.    }
14.  }
15. }
16.
17. Sweep();
```

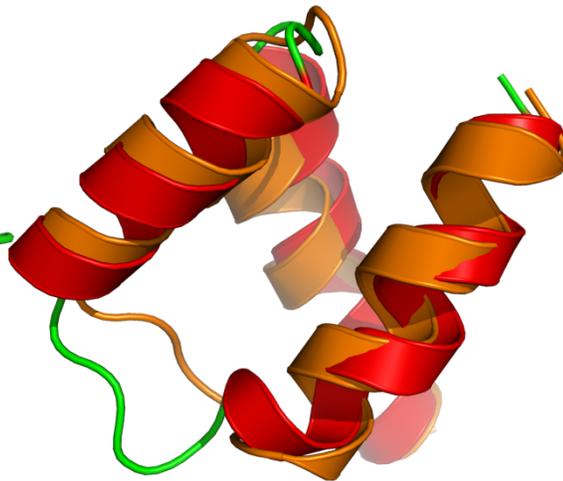
10 proteins x 1000 simulations x  
3 rounds x 2 temps x 5 deltas  
= 300K tasks

Protein	SEQ Len	Class	ST	TUI	Lowest RMSD (Å)	DeBartolo RMSD (Å)	Protein	Len	Class	ST	TUI	Lowest RMSD (Å)							
<b>1af7</b>	69	$\alpha$	15	25	3.77	2.5	<b>1dcj</b>	72	$\alpha/\beta$	15	25	8.75							
				50	3.60						50	9.11							
				10 0	3.77						10 0	7.22							
			25	25	3.20					25	25	8.34							
				50	3.78						50	7.69							
				10 0	3.01						10 0	8.94							
			<b>1r69</b>	61	$\alpha$					15	25	3.20	2.4	<b>1ubq</b>	73	$\alpha/\beta$	15	25	6.68
											50	4.09						50	7.05
											10 0	3.87						10 0	6.00
25	25	3.76				25	25	6.88											
	50	2.94					50	8.29											
	10 0	3.87					10 0	8.01											

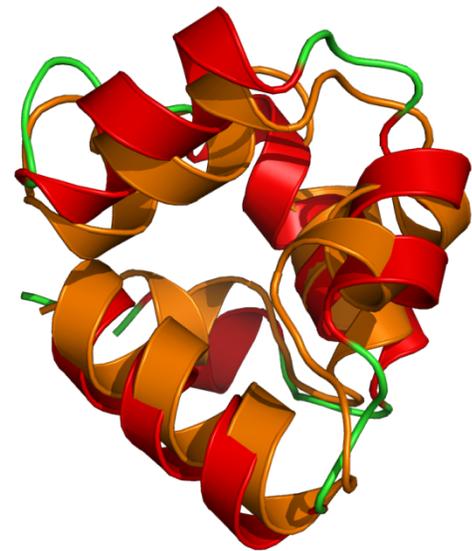
Protein	Length h	ST	TUI	Lowest RMSD (Å)	DeBartolo RMSD (Å)
<b>T1af7</b>	69	25	100	2.07	2.5
<b>T1b72</b>	50	25	100	1.41	1.6
<b>T1r69</b>	61	25	100	2.11	2.4



T1af7



T1b72



T1r69

## Display Swift Output:

[HOME](#)

[PROJECTS](#)

[GROUP  
INFORMATION](#)

[PUBLICATIONS](#)

[GALLERY](#)

[LINKS](#)

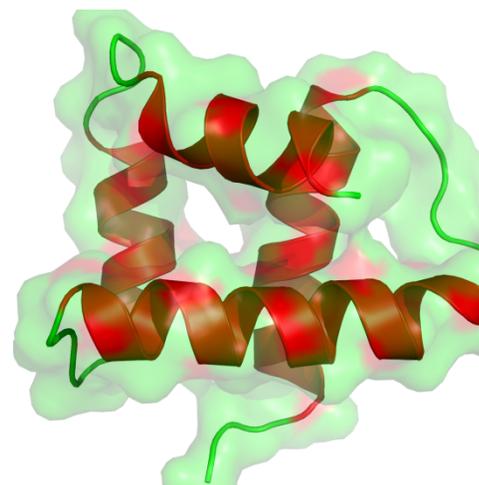
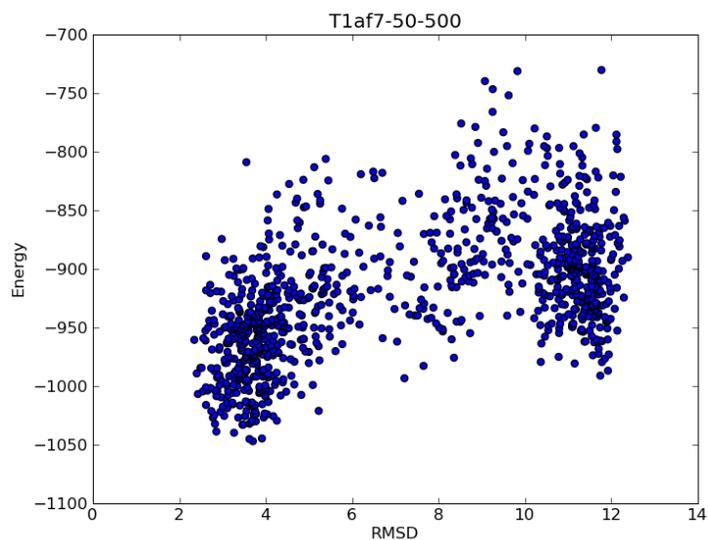
Choose run:

1093outdir.1

key	AverageRunTime	LowestRMSD	LowestRMSDEnergy	PredictionEnergy	PredictionRMSD	TotalRuns
T1af7-50-500	50995	2.34647	-960.561	-1047.0	3.70194	985
T1b72-50-500	36777	2.10575	-469.571	-568.818	3.20041	1012
T1dej-50-500	12617	6.30753	-2584.0	-5591.67	8.47168	995
T1di2-50-500	13484	3.4034	-5281.69	-8442.19	5.78596	1005
T1mky-50-500	12129	5.61484	-3885.61	-4729.66	8.55371	1004
T1r69-50-500	34001	2.23661	-592.601	-664.585	7.72925	998
T1tif-50-500	9050	4.1065	-4994.31	-6344.37	9.34496	1012
T1ubq-50-500	14857	4.84337	-4645.08	-7223.23	9.56997	991



This page last updated on Sun, 12 Apr 2009 22:37:50

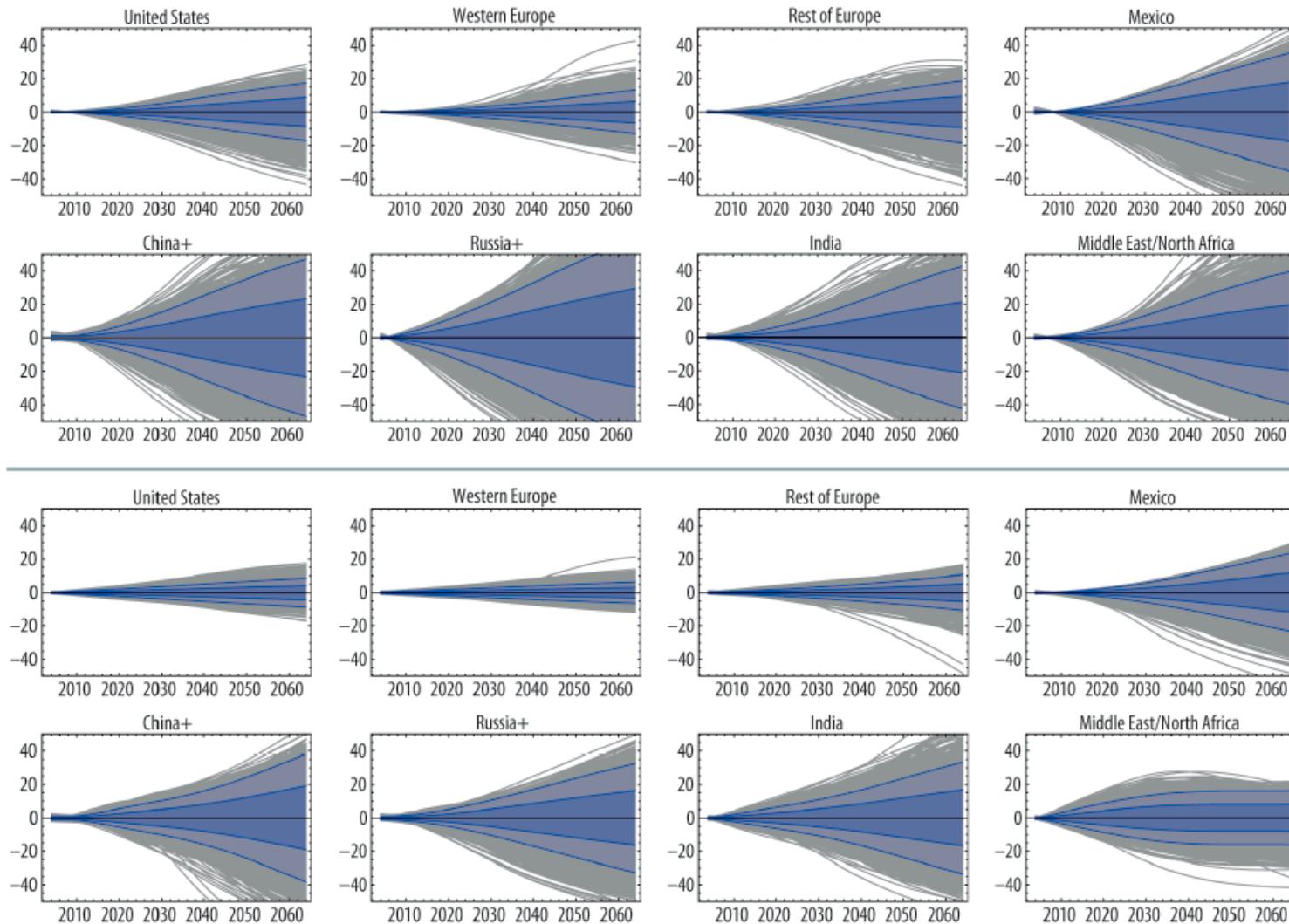


**Table 1. Example parallel scripting applications.**

Field	Description	Characteristics	Status
Astronomy	Creation of montages from many digital images	Many 1-core tasks, much communication, complex dependencies	Experimental
Astronomy	Stacking of cutouts from digital sky surveys	Many 1-core tasks, much communication	Experimental
Biochemistry*	Analysis of mass-spectrometer data for post-translational protein modifications	10,000-100 million jobs for proteomic searches using custom serial codes	In development
Biochemistry*	Protein structure prediction using iterative fixing algorithm; exploring other biomolecular interactions	Hundreds to thousands of 1- to 1,000-core simulations and data analysis	Operational
Biochemistry*	Identification of drug targets via computational docking/screening	Up to 1 million 1-core docking operations	Operational
Bioinformatics*	Metagenome modeling	Thousands of 1-core integer programming problems	In development
Business economics	Mining of large text corpora to study media bias	Analysis and comparison of over 70 million text files of news articles	In development
Climate science	Ensemble climate model runs and analysis of output data	Tens to hundreds of 100- to 1,000-core simulations	Experimental
Economics*	Generation of response surfaces for various economic models	1,000 to 1 million 1-core runs (10,000 typical), then data analysis	Operational
Neuroscience*	Analysis of functional MRI datasets	Comparison of images; connectivity analysis with structural equation modeling, 100,000+ tasks	Operational
Radiology	Training of computer-aided diagnosis algorithms	Comparison of images; many tasks, much communication	In development
Radiology	Image processing and brain mapping for neuro-surgical planning research	Execution of MPI application in parallel	In development

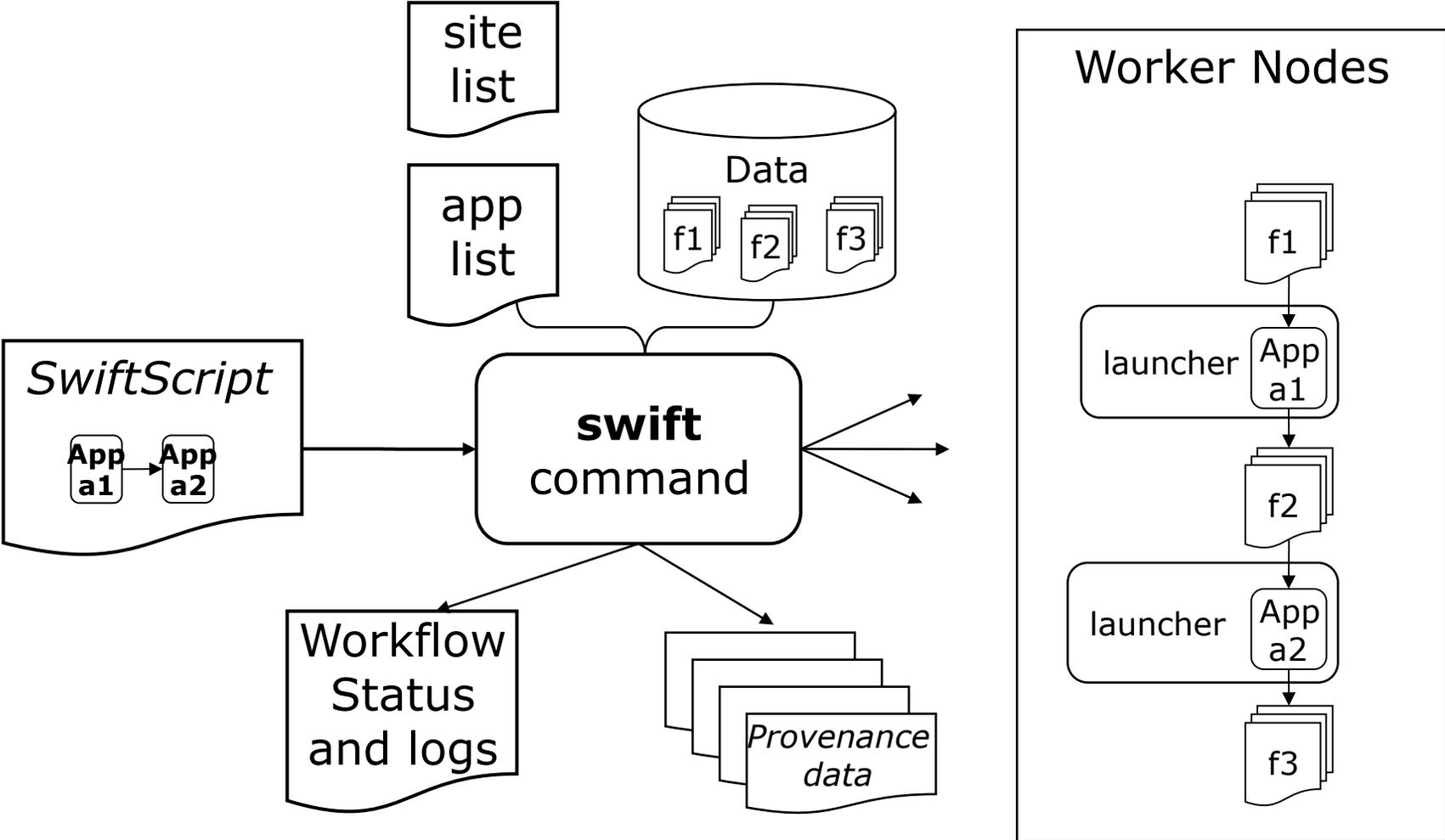
\*Note: Asterisks indicate applications being run on Argonne National Laboratory's Blue Gene/P (Intrepid) and/or the TeraGrid Sun Constellation at the University of Texas at Austin (Ranger).

# CIM-EARTH: Modeling uncertainty

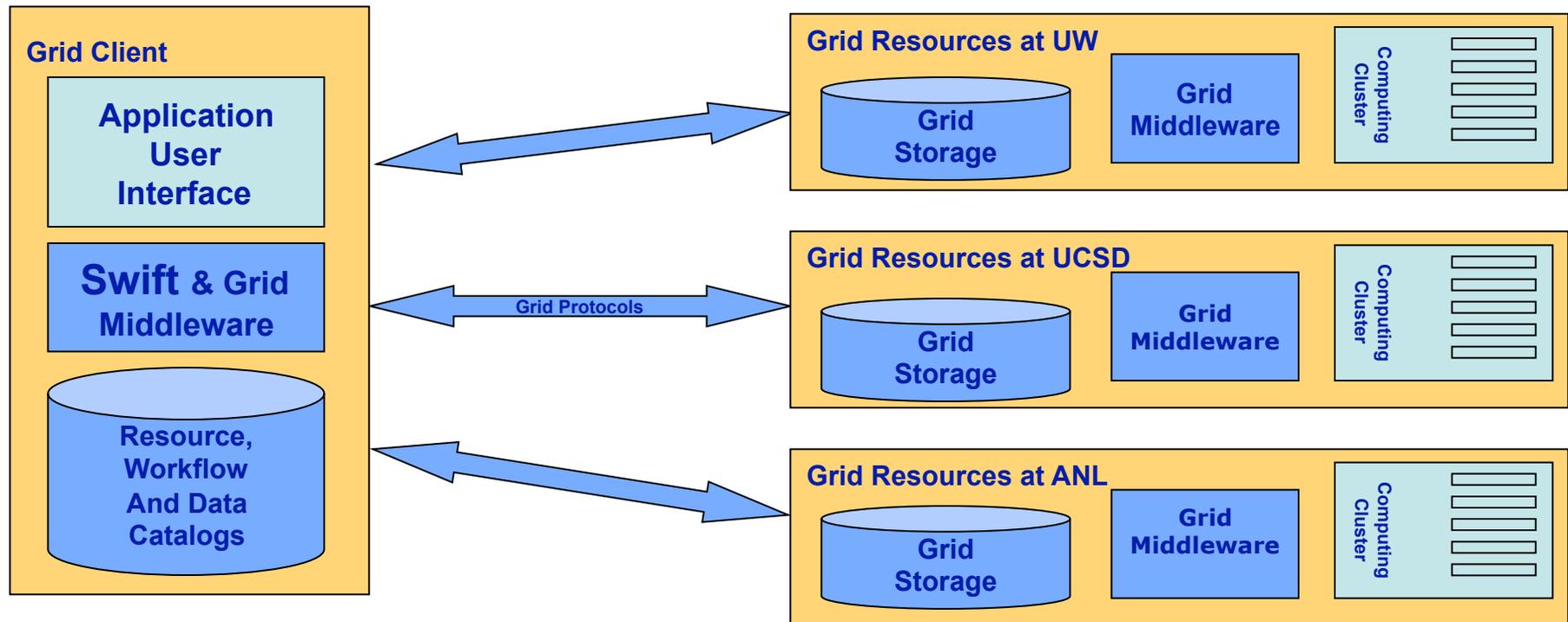


**Figure 4. CIM-EARTH energy-economics parameter sweeps of 5,000 models exploring uncertainty in consumer (top) and industrial (bottom) electricity usage projections by region for the next five decades.**

# Swift parallel scripting architecture



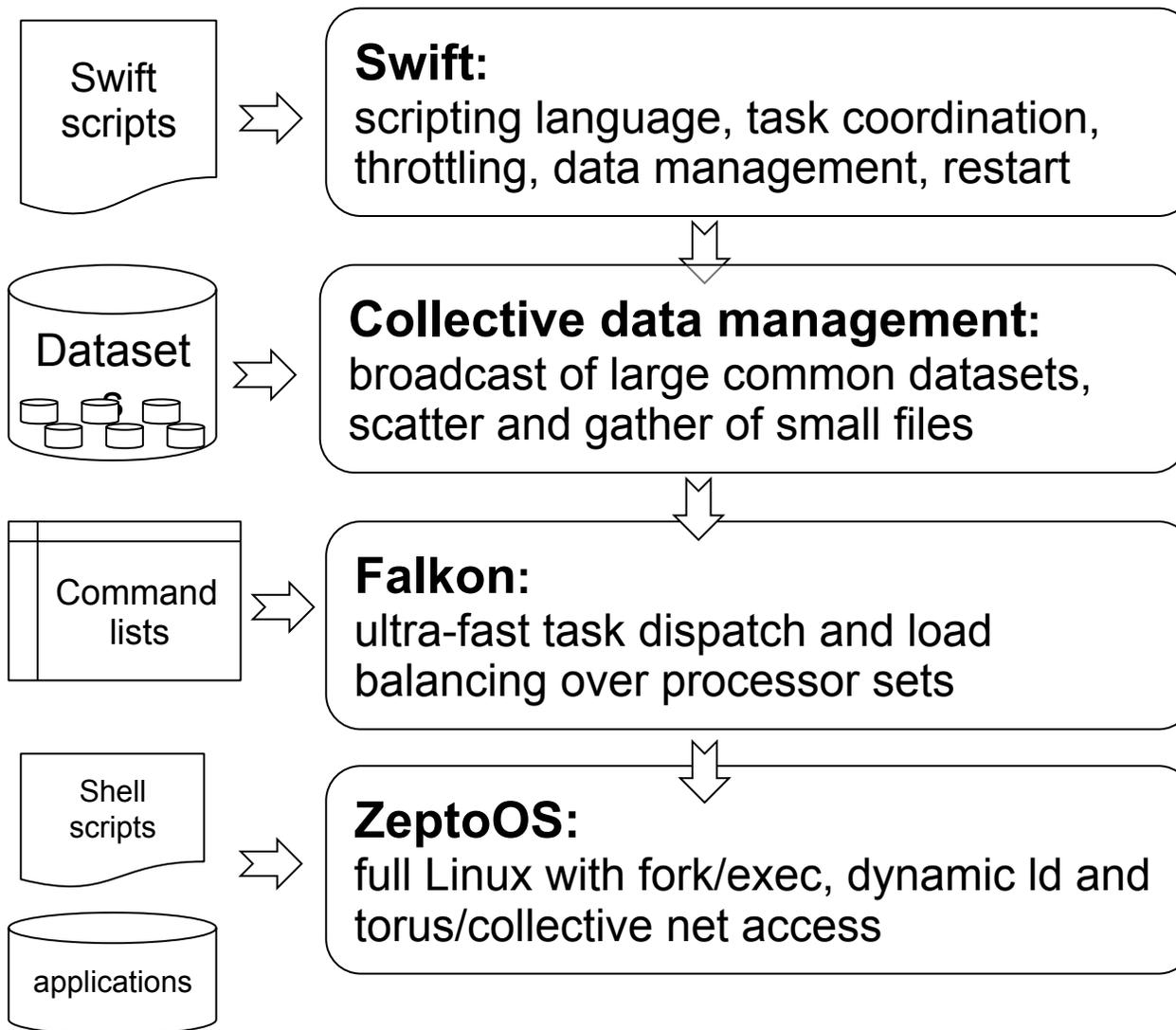
# Environment for Grid scripting



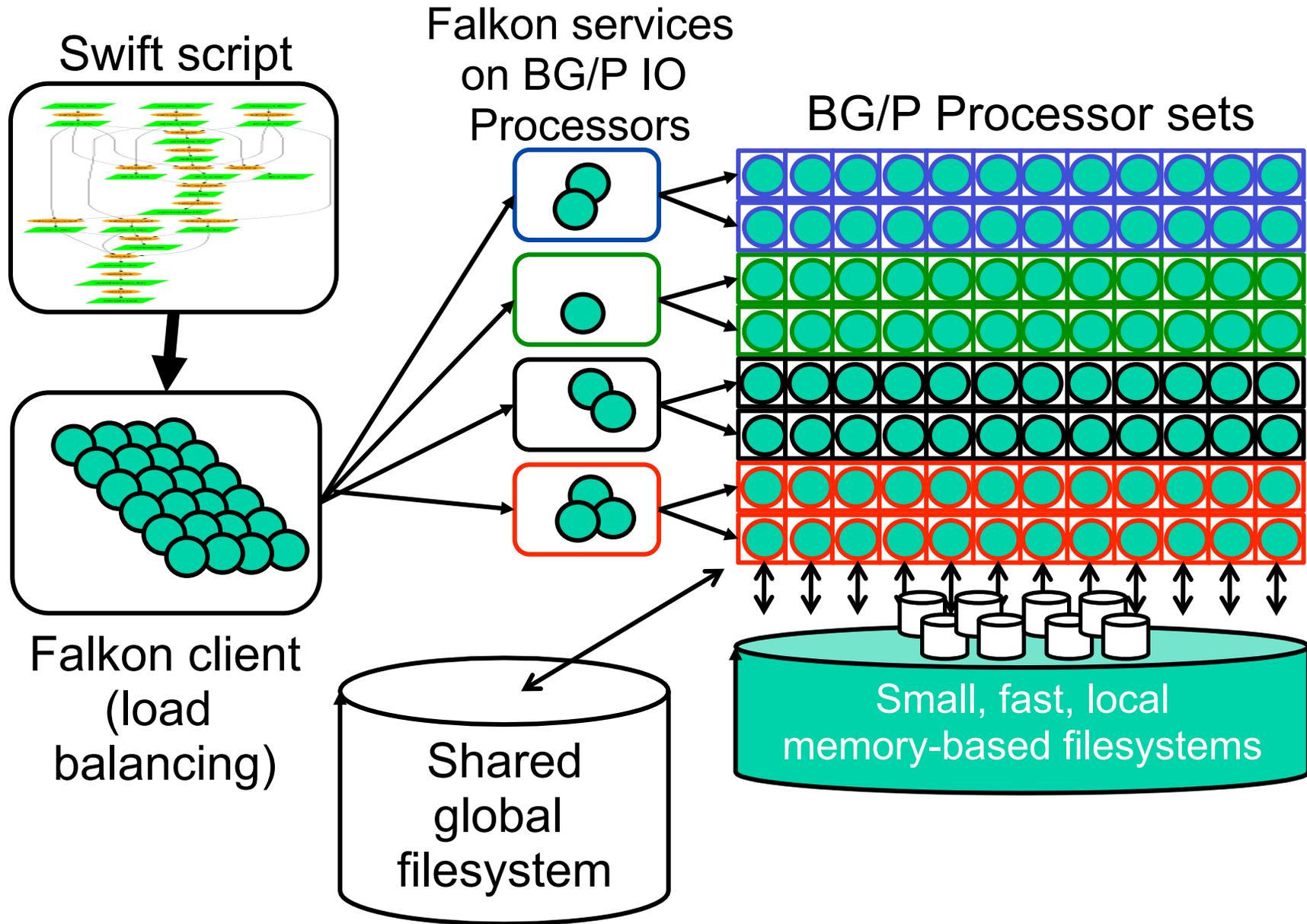
Swift runs on the grid client or “submit host”

- Sends jobs to one or more grid sites using GRAM and Condor-G
- Sends files to and from grid sites using GridFTP
- Directory to locate grid sites and services: (ReSS)
- Can also run on local hosts, or directly on a local cluster
- Can overlay a faster scheduling mechanism (Coasters, Falkon)

# Architecture for petascale scripting



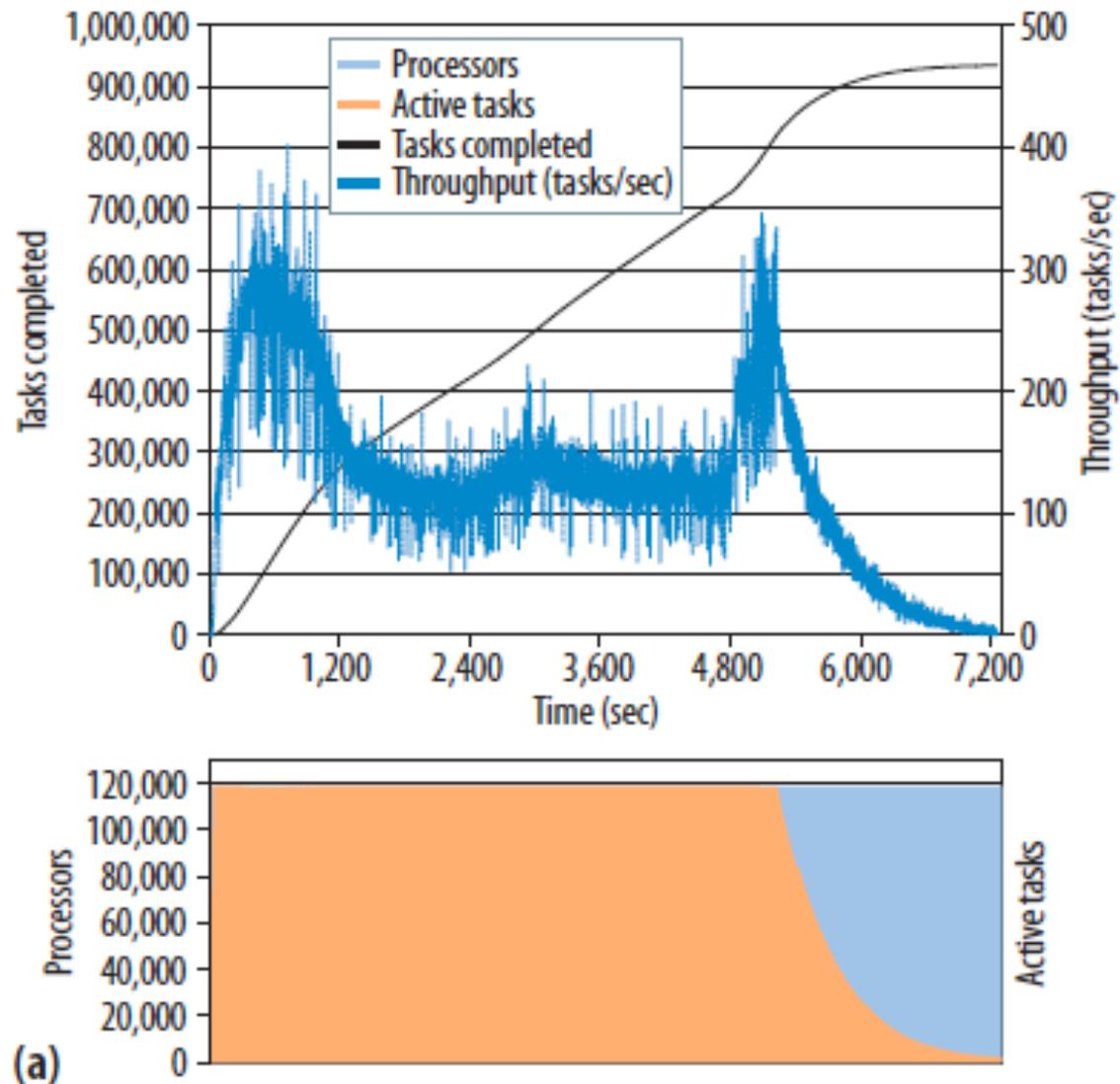
# Architecture for petascale scripting



# Collective data management is critical for petascale

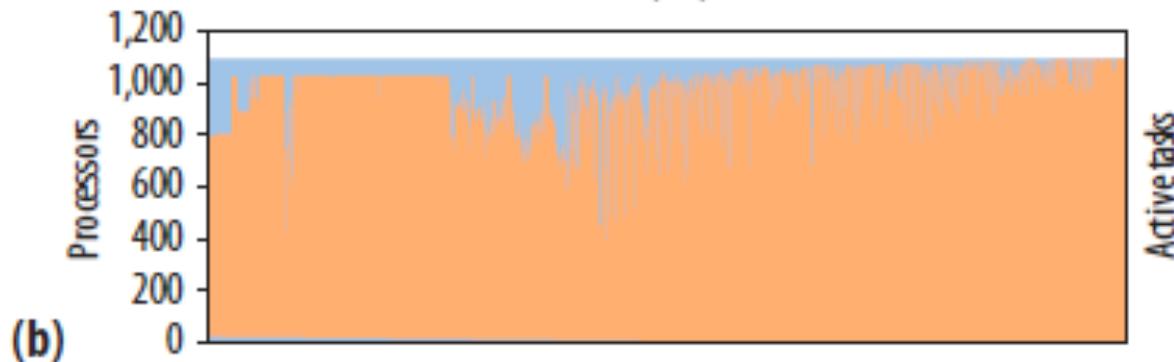
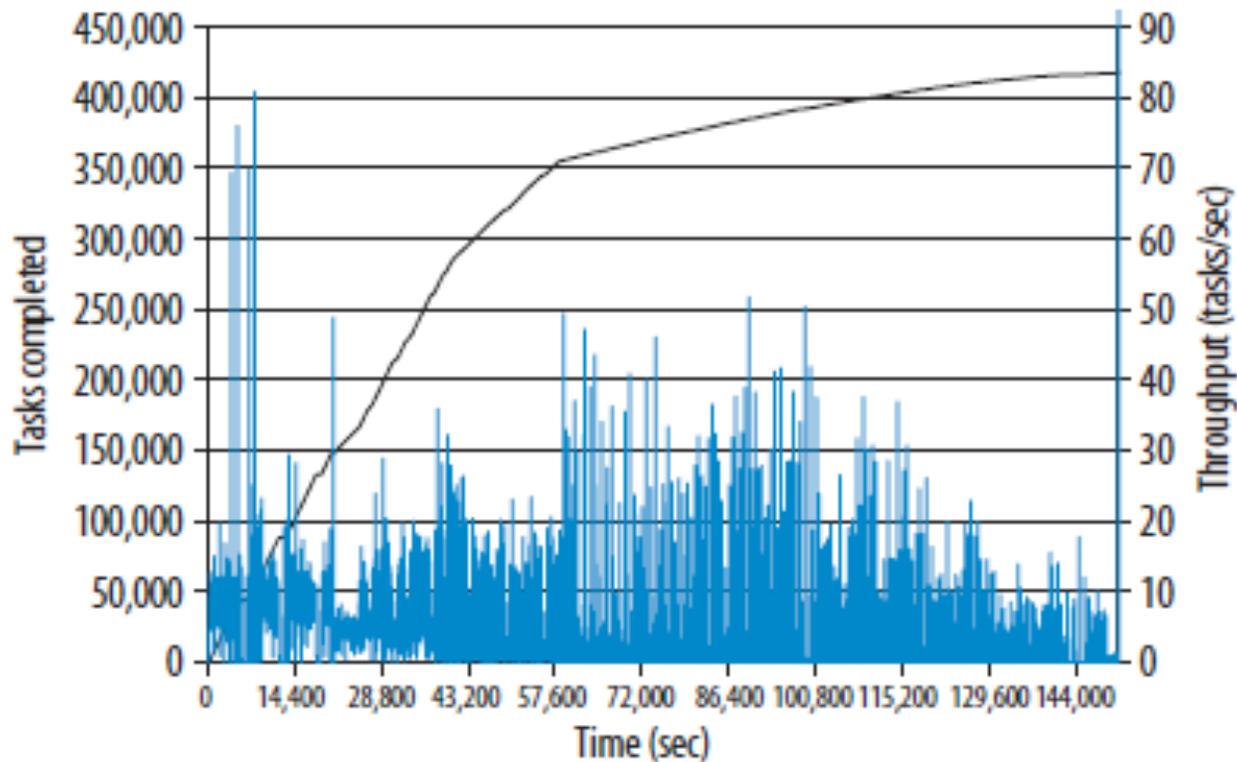
- Applies “scatter/gather” concepts at the file management level
- Seeks to avoid contention, maximize parallelism and use petascale interconnects
  - Broadcast common files to compute nodes
  - Place per-task data on local (RAM) FS
  - Gather output into larger sets (time/space)
  - Aggregate small local FS’s into large striped FS
- Still in research – topic of new EAGER grant

# Performance: Molecular dynamics on BG/P



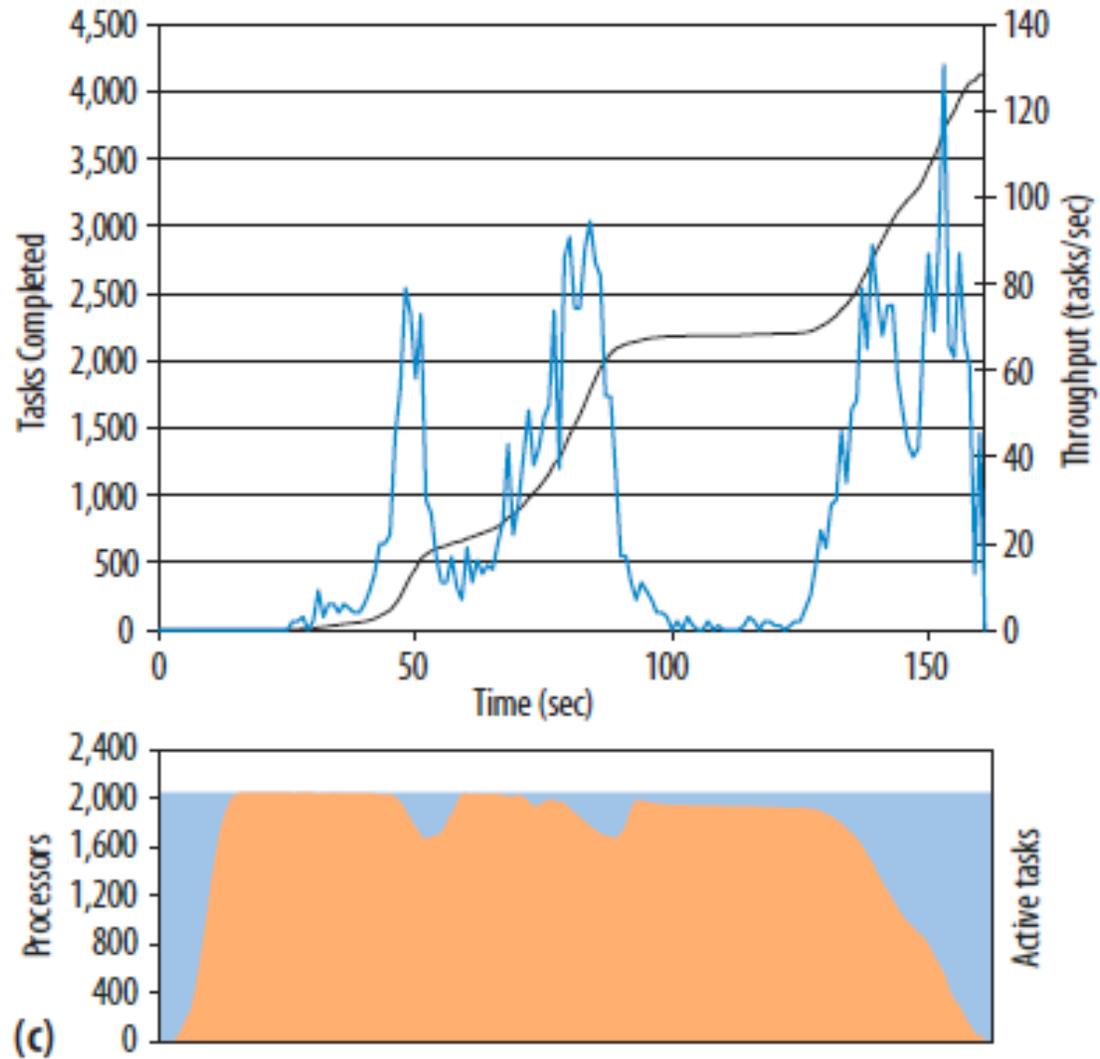
935,803 DOCK jobs with Falkon on BG/P in 2 hours

# Performance: SEM for fMRI on Constellation



418K SEM tasks with Swift/Coasters on Ranger in 41 hours

# Performance: Proteomics on BG/P



4,127 PTMap jobs with Swift/Falkon on BG/P in 3 minutes

# Summary

- Clean separation of logical/physical concerns
  - Mapper-based specification of logical data structures
- + Concise specification of parallel programs
  - Simple scripting language with iteration, etc.
- + Efficient execution
  - On distributed and petascale resources
  - **Karajan+Falkon/Coasters**: 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

# To learn more...

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

# Acknowledgments

- Swift effort is supported in part by NSF grants OCI-721939, OCI-0944332, and PHY-636265, NIH DC08638, and the UChicago/Argonne Computation Institute
- The Swift team:
  - Ben Clifford, Allan Espinosa, Ian Foster, Mihael Hategan, Ioan Raicu, Sarah Kenny, Mike Wilde, Justin Wozniak, Zhao Zhang, Yong Zhao
- Java CoG Kit used by Swift developed by:
  - Mihael Hategan, Gregor Von Laszewski, and many collaborators
- Falkon software
  - developed by Ioan Raicu and Zhao Zhang
- ZeptoOS
  - Kamil Iskra, Kazutomo Yoshii, and Pete Beckman
- Scientific application collaborators and users
  - U. Chicago Open Protein Simulator Group (Karl Freed, Tobin Sosnick, Glen Hocky, Joe Debartolo, Aashish Adhikari)
  - U.Chicago Radiology and Human Neuroscience Lab, (Dr. S. Small)
  - SEE/CIM-EARTH: Joshua Elliott, Meredith Franklin, Todd Muson
  - PTMap: Yingming Zhao, Yue Chen