Parallel scripting with Swift for applications at the petascale and beyond

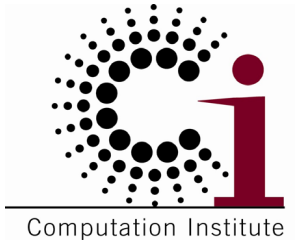
VecPar PEEPS Workshop

Berkeley, CA – June 22, 2010

Michael Wilde – wilde@mcs.anl.gov

Computation Institute, University of Chicago
and Argonne National Laboratory

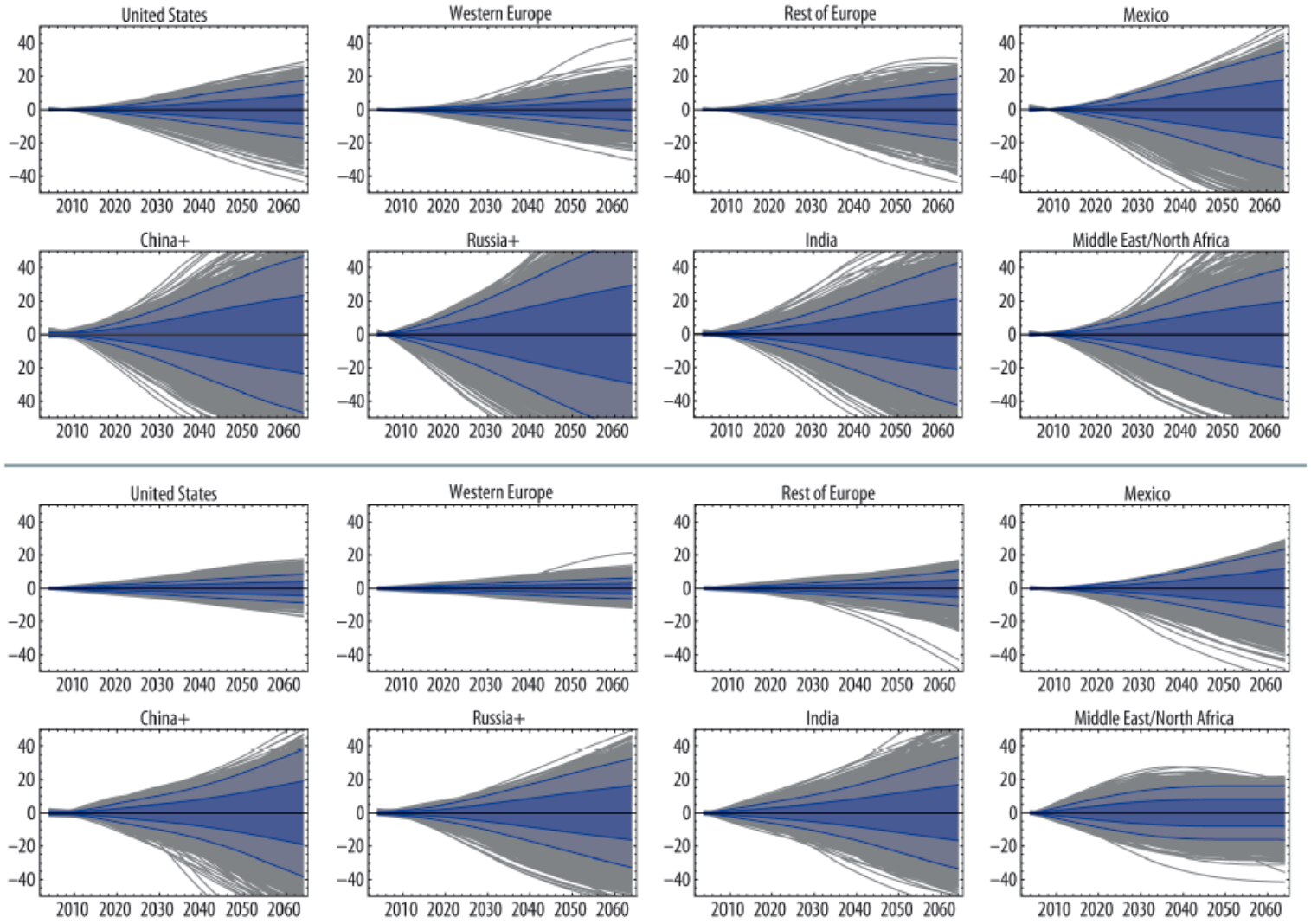
www.ci.uchicago.edu/swift



Problems addressed by Swift

- Many applications need loosely coupled scripting
- Swift harnesses parallel & distributed resources through a simple scripting language
- Productivity gains by enabling use of more powerful systems with less concern for the mechanics

Modeling uncertainty for CIM-EARTH

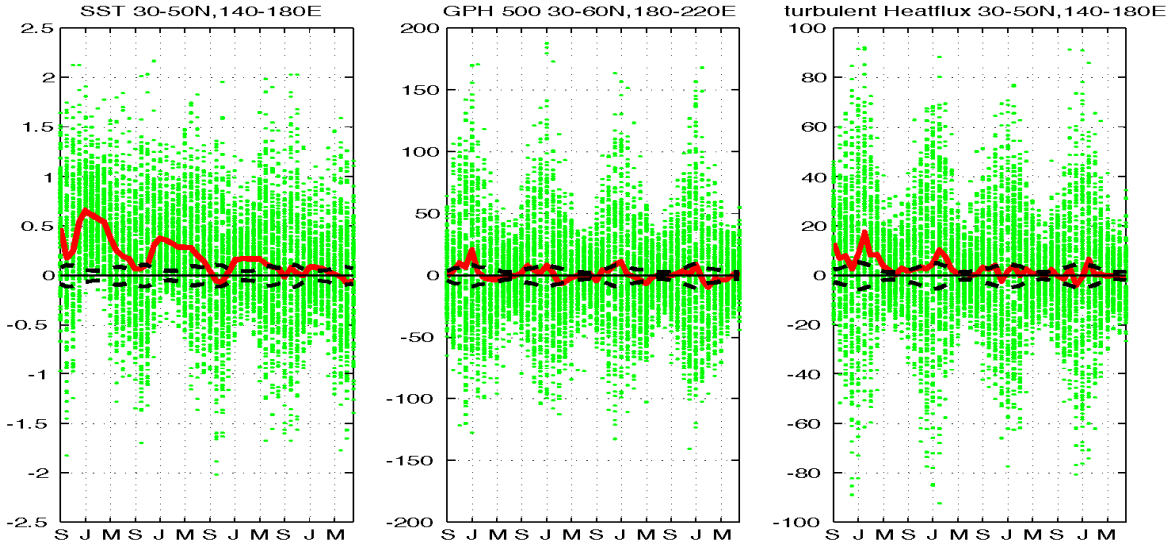


Parallel AMPL workflow by *Joshua Elliott, Meredith Franklin, Todd Munson, Allan Espinosa.*

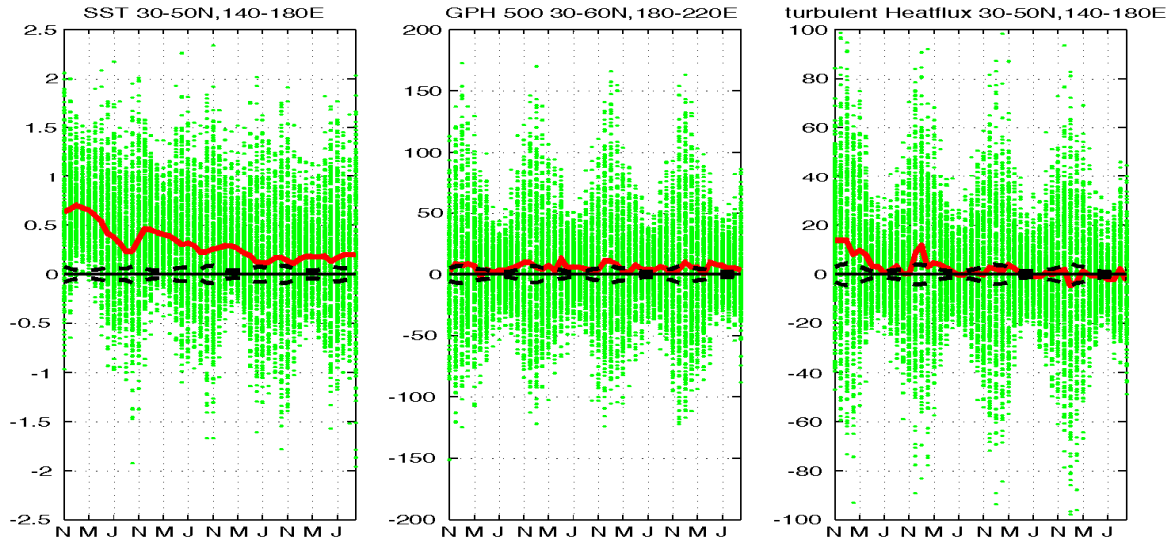
Figure 4. CIM-EARTH energy-economics parameter sweeps of 5,000 models exploring uncertainty in consumer (top) and industrial (bottom) electricity usage projections by region for the next five decades.

Fast Ocean Atmosphere Model (MPI)

160 ensemble members = 2.5 months to run



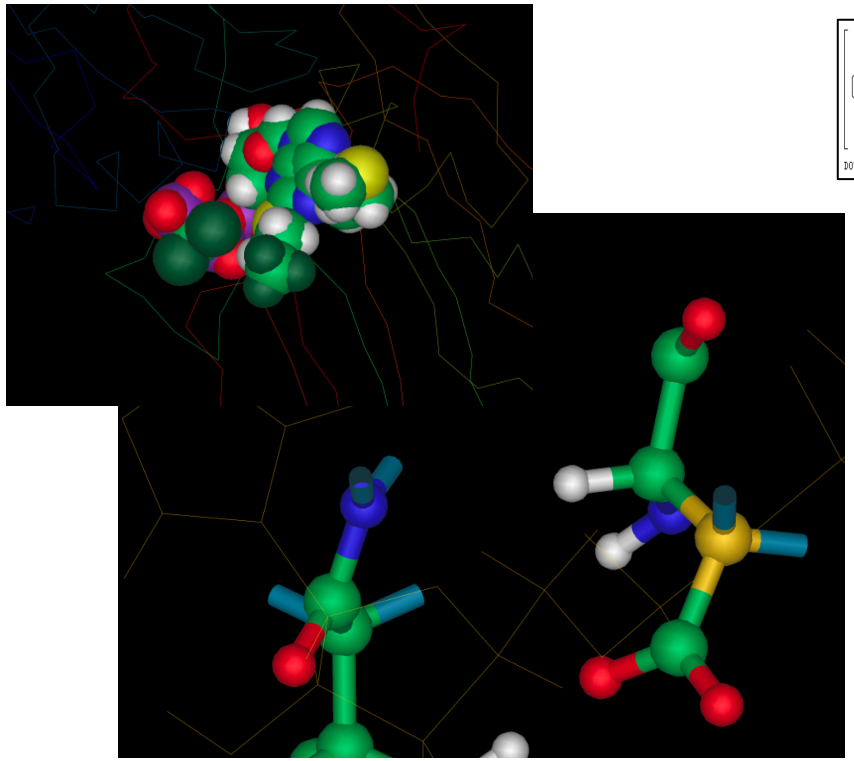
250 ensemble members = 4 days to run



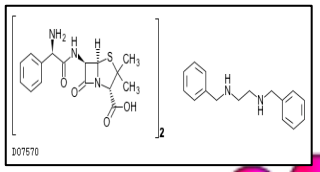
Green: each ensemble Red: ensemble mean

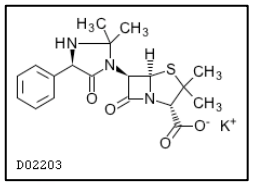
Work of
Veronica Nefedova
and Rob Jacob, Argonne

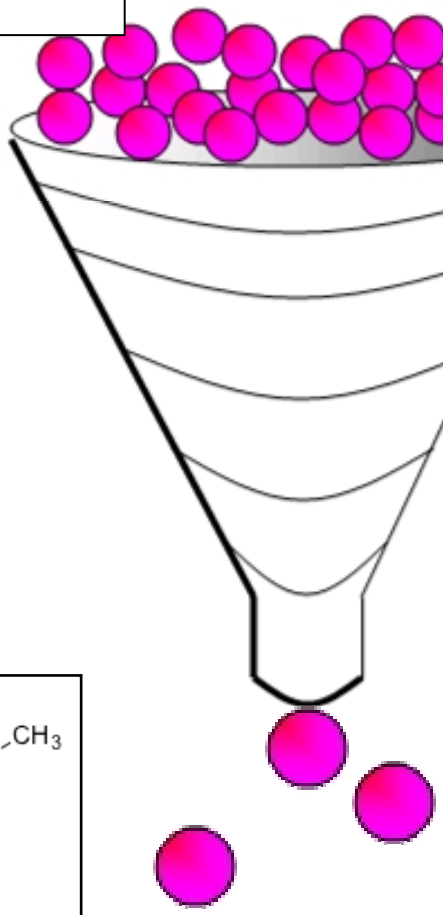
Problem: Drug screening at APS

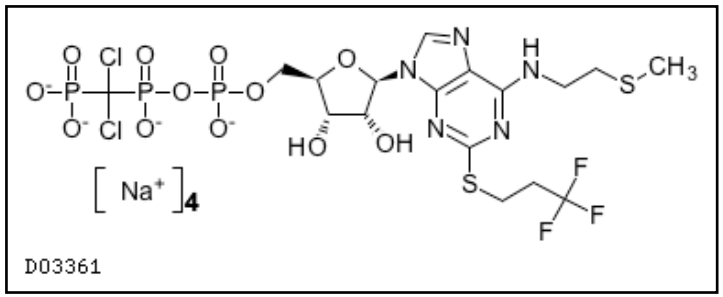


O(Millions) of drug candidates

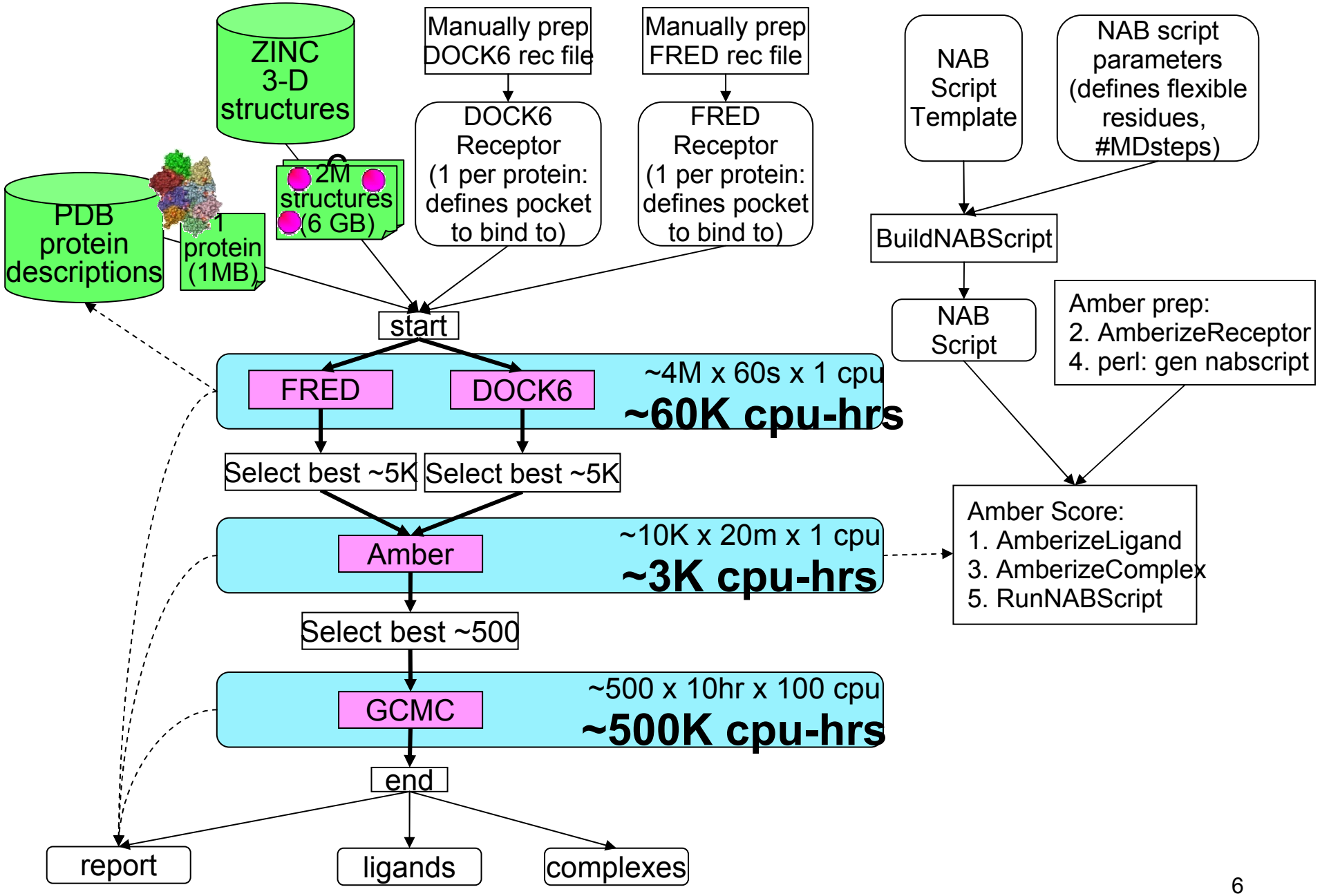






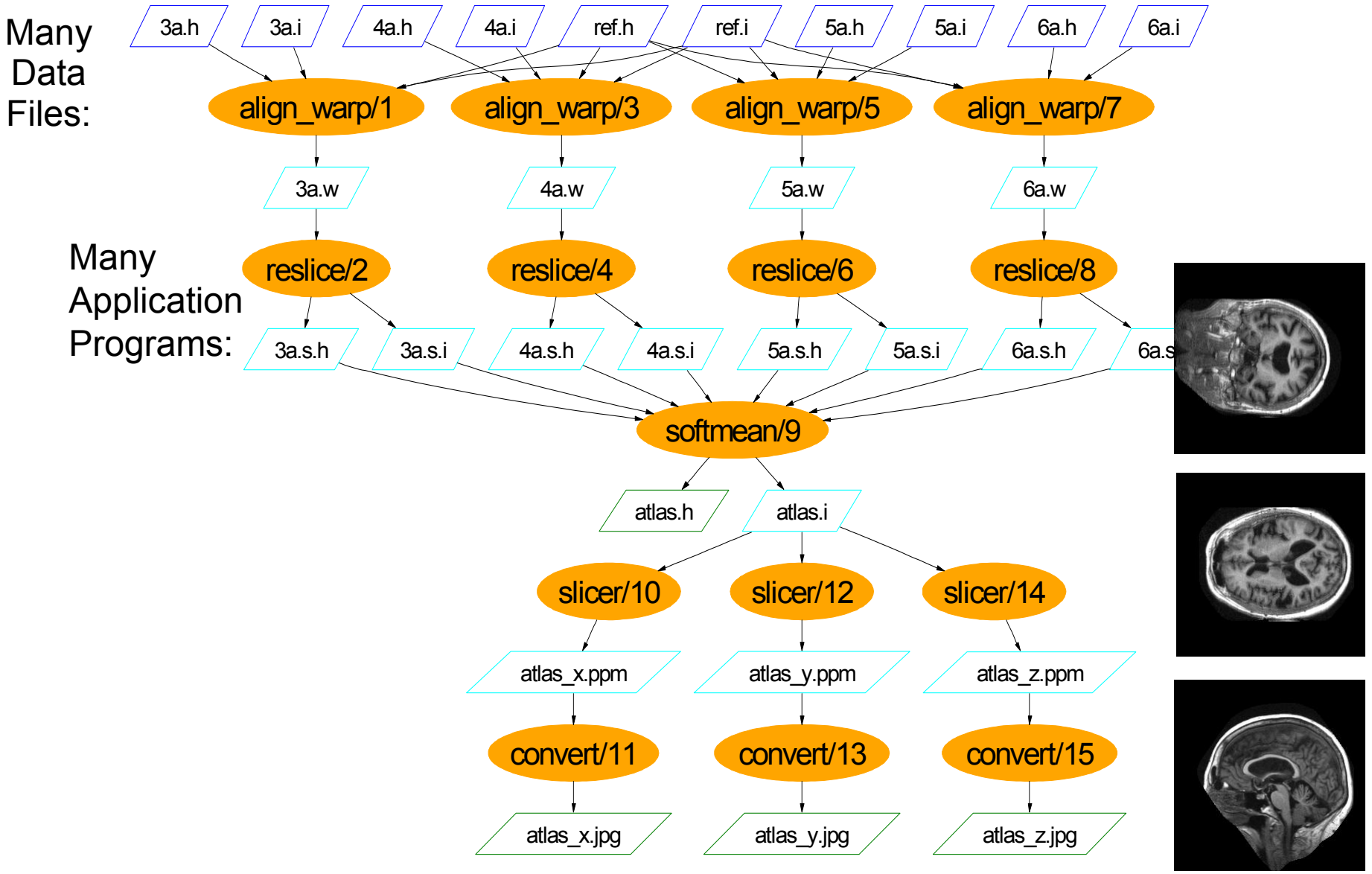


O(tens) of fruitful candidates for wetlab & APS



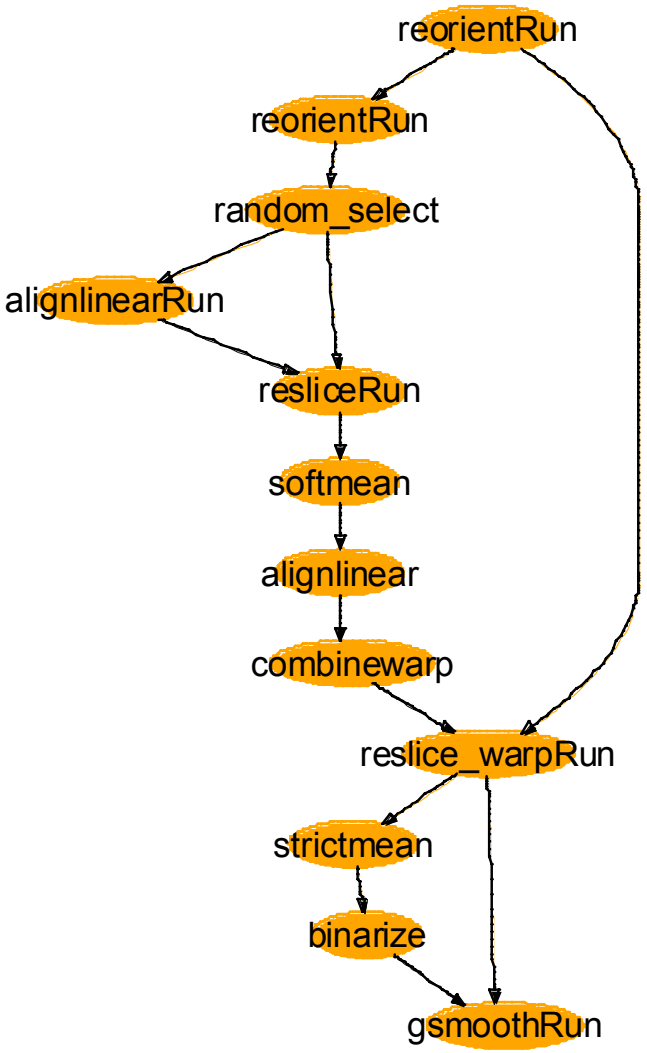
Work of Andrew Binkoaski and Michael Kubal

Problem: preprocessing and analysis of neuroscience experiments

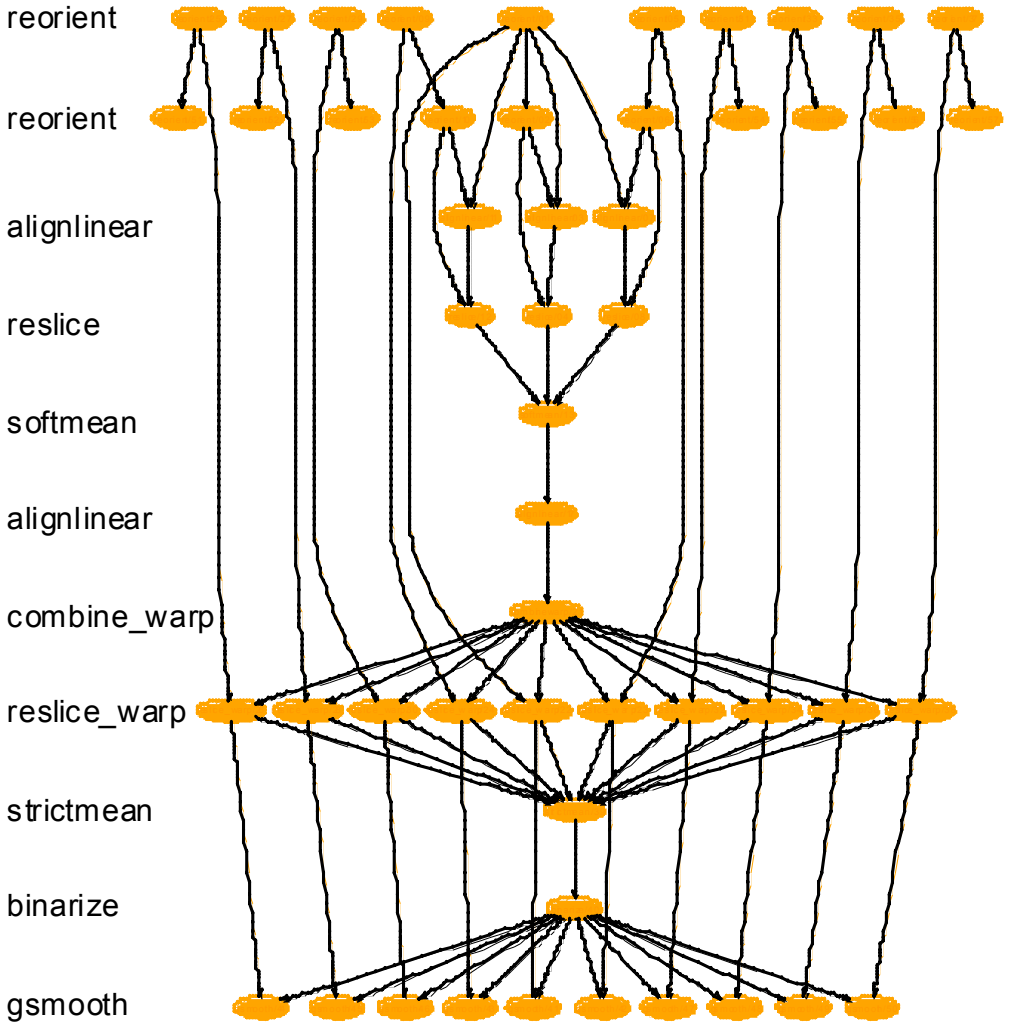


Automated image registration for spatial normalization

AIRSN workflow:



AIRSN workflow expanded:



Swift programs

- A Swift script is a set of **functions**
  - Atomic functions wrap & invoke application programs (on parallel compute nodes)
  - Composite functions invoke other functions (run in Swift engine)
- 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
- Variables are **single assignment**
- Expressions and statements are executed in **data-flow** dependency order and concurrency
- Members of datasets can be processed in **parallel**
- **Provenance** is gathered as scripts execute

A simple Swift script

To run the Image Magick app “convert”:

```
1       convert -rotate 180 $in $out

2   type imagefile { } // Declare a “file” type.
3
4   app (imagefile output) rotate (imagefile input) {
5   {
6      convert "-rotate" 180 @input @output ;
7   }
8
9   imagefile image      <"m101.2010.0601.jpg">;
10  imagefile newimage <"output.jpg">;
11
12  newimage = rotate(image);
```

Execution is driven by data flow

```
1  (int result) myproc (int input)
2  {
3      j = f(input);
4      k = g(input);
5      result = j + k;
6  }
```

7 j=f() and k=g() are computed in parallel.

8 This parallelism is **automatic**, based on futures;
9 Works recursively down the scripts's call graph.

Parallelism via foreach { }

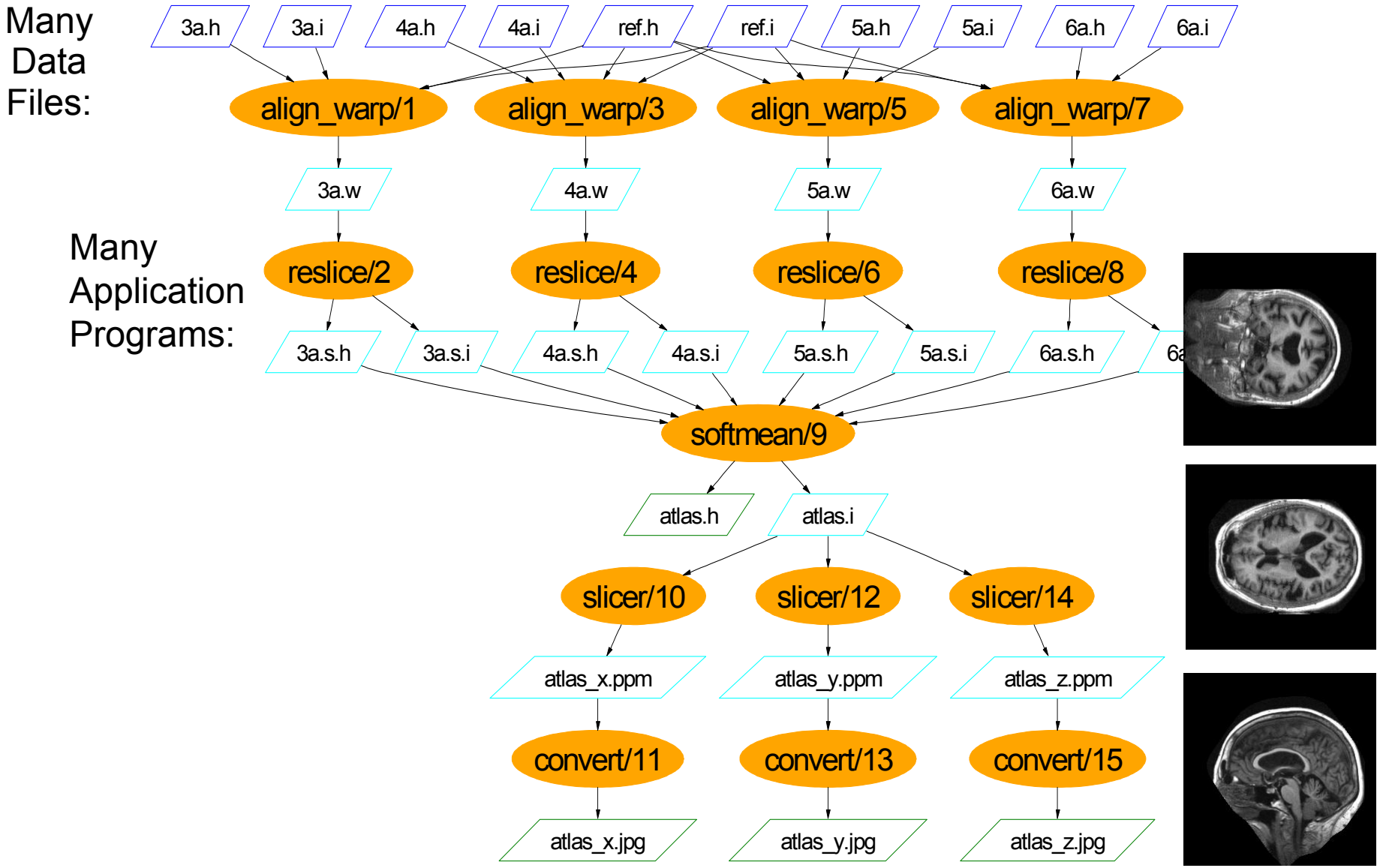
```
1  type imagefile;  // Declare a “file” type.
2
3  app (imagefile output) rotate (imagefile input) {
4        convert "-rotate" "180" @input @output;
5  }
6
7  imagefile observations[ ]
8  imagefile flipped[ ]
9
10
11
```

Map inputs from local directory

Name outputs based on index

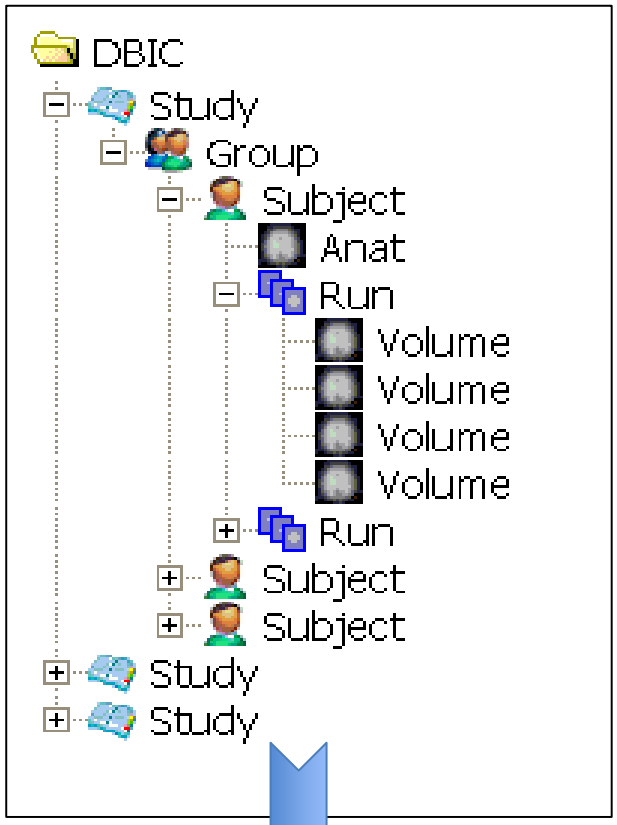
Process all dataset members in parallel

Many domains process structured datasets



Swift Data Mapping

On-Disk Data Layout

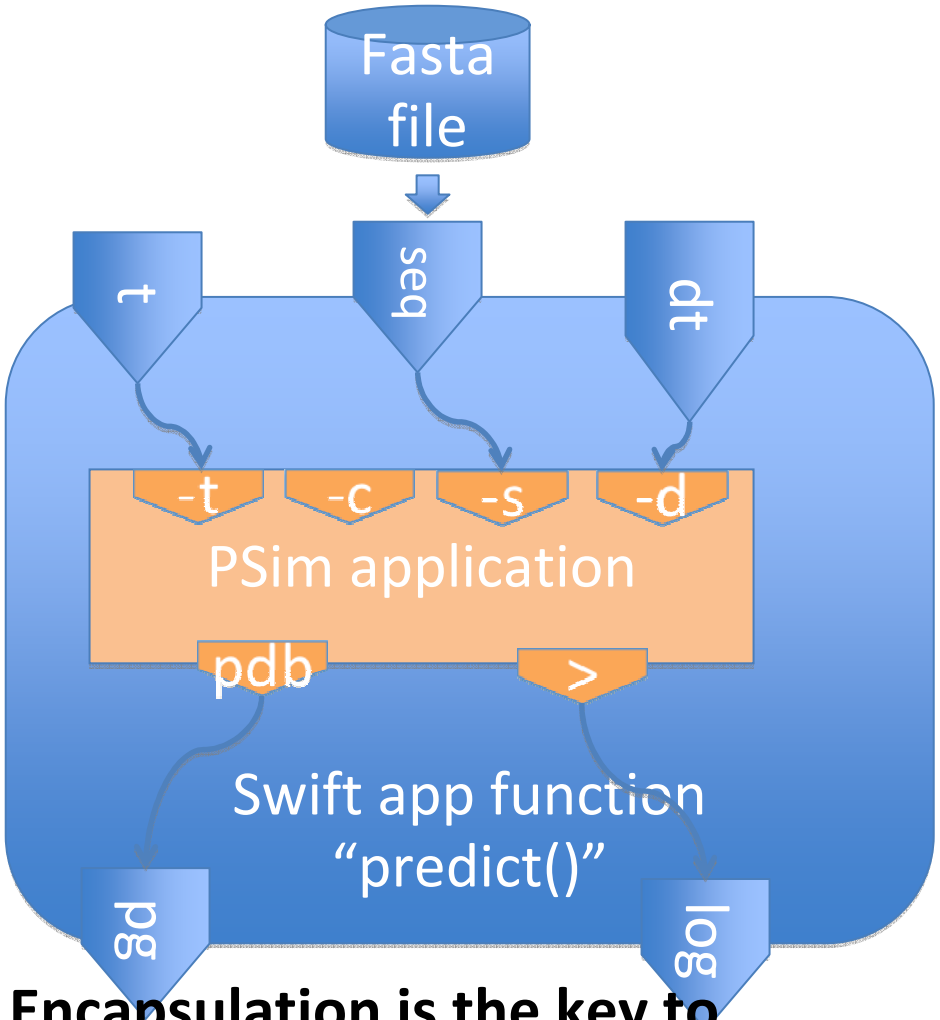


Mapping function or script

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

type Group {
        Subject s[ ];
}

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

type Run {
        Volume v[ ];
}

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

Swift's in-memory data model

Application: Protein structure prediction



To run:

```
psim –s 1ubq.fas –pdb p          \
        –temp 100.0 –inc 25.0 >log
```

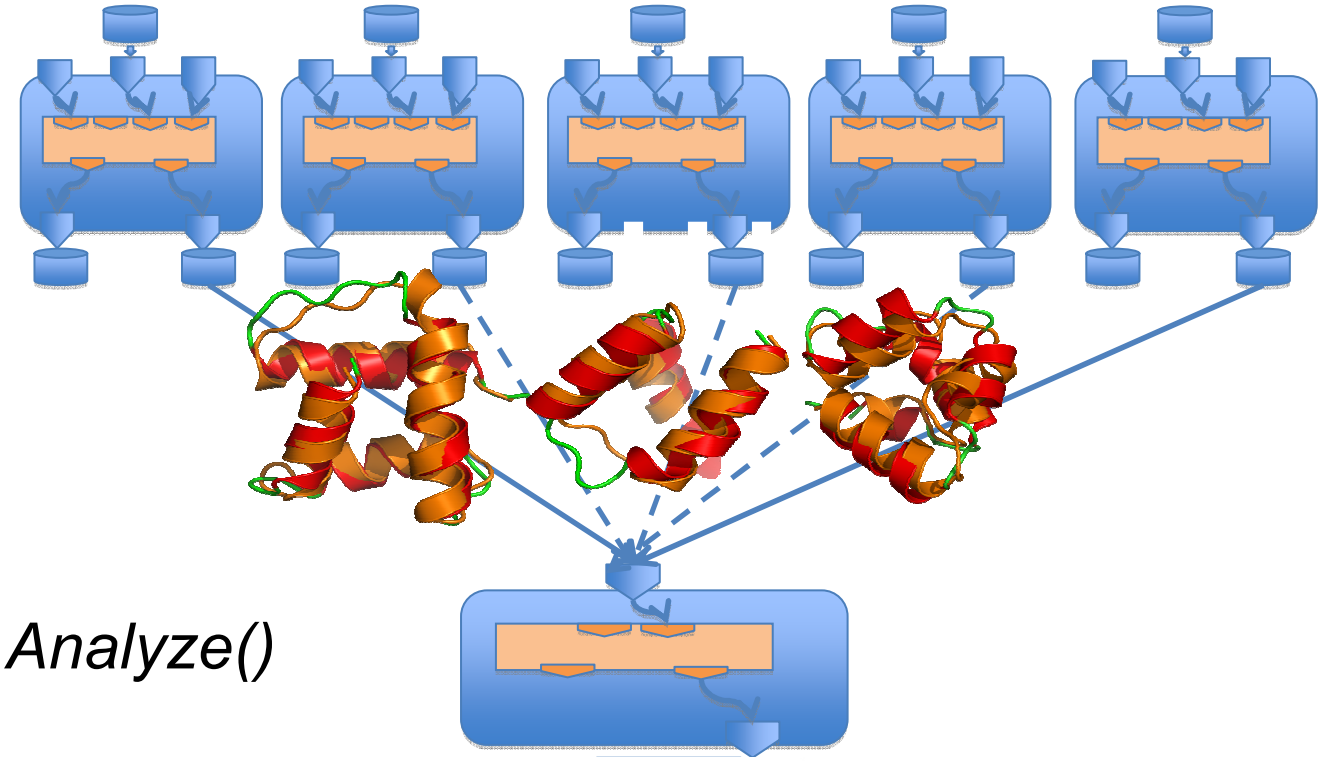
In Swift code:

```
app (PDB pg, File log) predict (Protein seq,
                                Float temp, Float dt)
{
  psim "-s" @pseq.fasta "-pdb" @pg
          "–temp" temp ”-inc" dt;
}

Protein p <ext; exec="Pmap", id="1ubq">;
ProtGeo structure;
TextFile log;

(structure, log) = predict(p, 100., 25.);
```

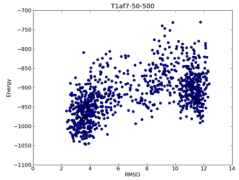
Encapsulation is the key to transparent distribution, parallelization, and provenance

Parallelism via foreach { }

1000 predict() calls



Analyze()



```
foreach sim in [1:1000] {
    (structure[sim], log[sim]) = predict(p, 100., 25.);
}
result = analyze(structure)
```

Application: 3D 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 = test_convergence(prediction);
13   }
14  }
```

Protein structure prediction

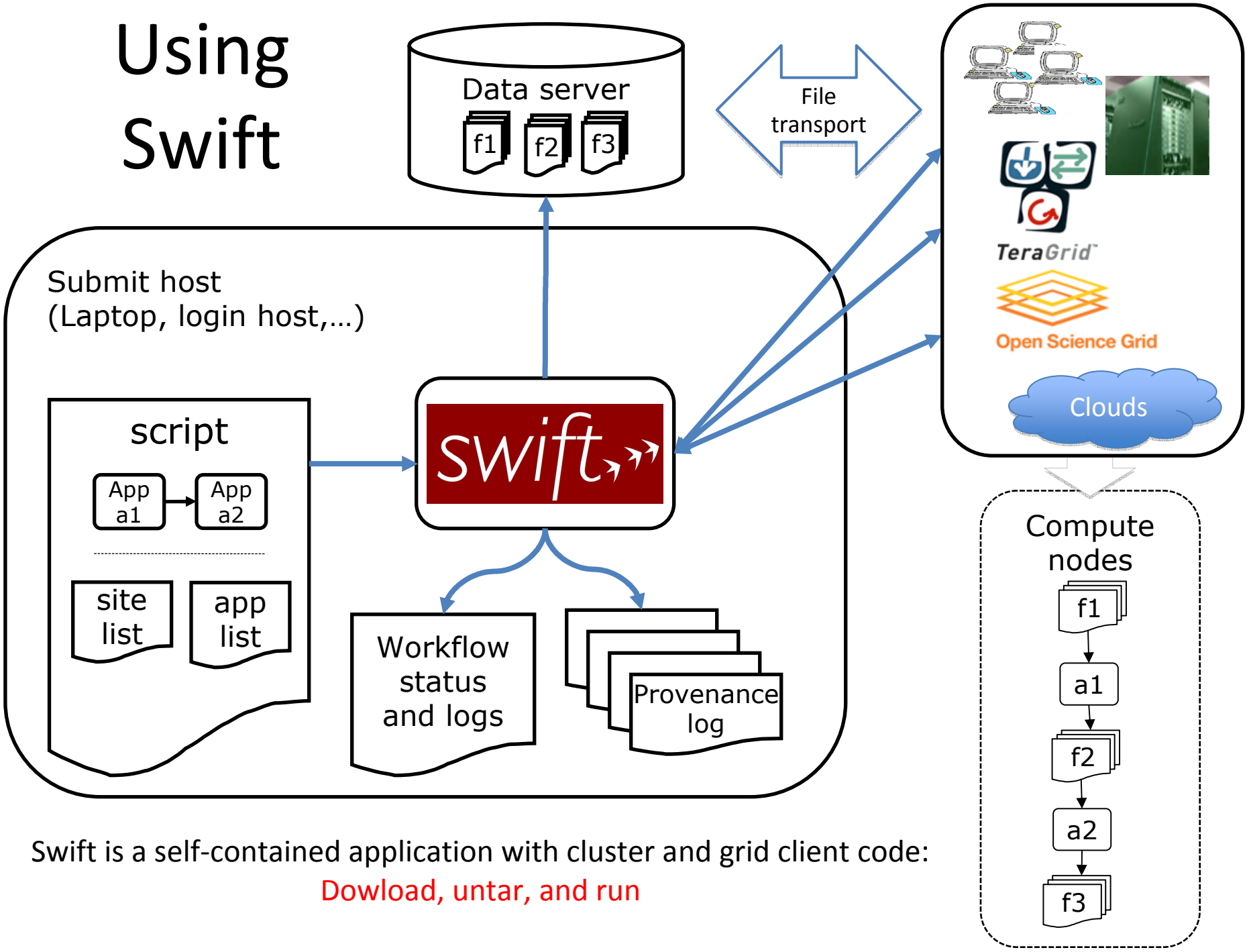
```
1.  ItFix( Protein p, int nsim, int maxr, float temp, float dt)
2.  {
3.    ProtSim prediction[ ][ ];
4.    boolean converged[ ];
5.    PSimCf config;
6.
7.    config.st = temp;
8.    config.tui = dt;
9.    config.coeff = 0.1;
10.
11.   iterate r {
12.     prediction[r] =
13.       doRoundCf(p, nsim,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

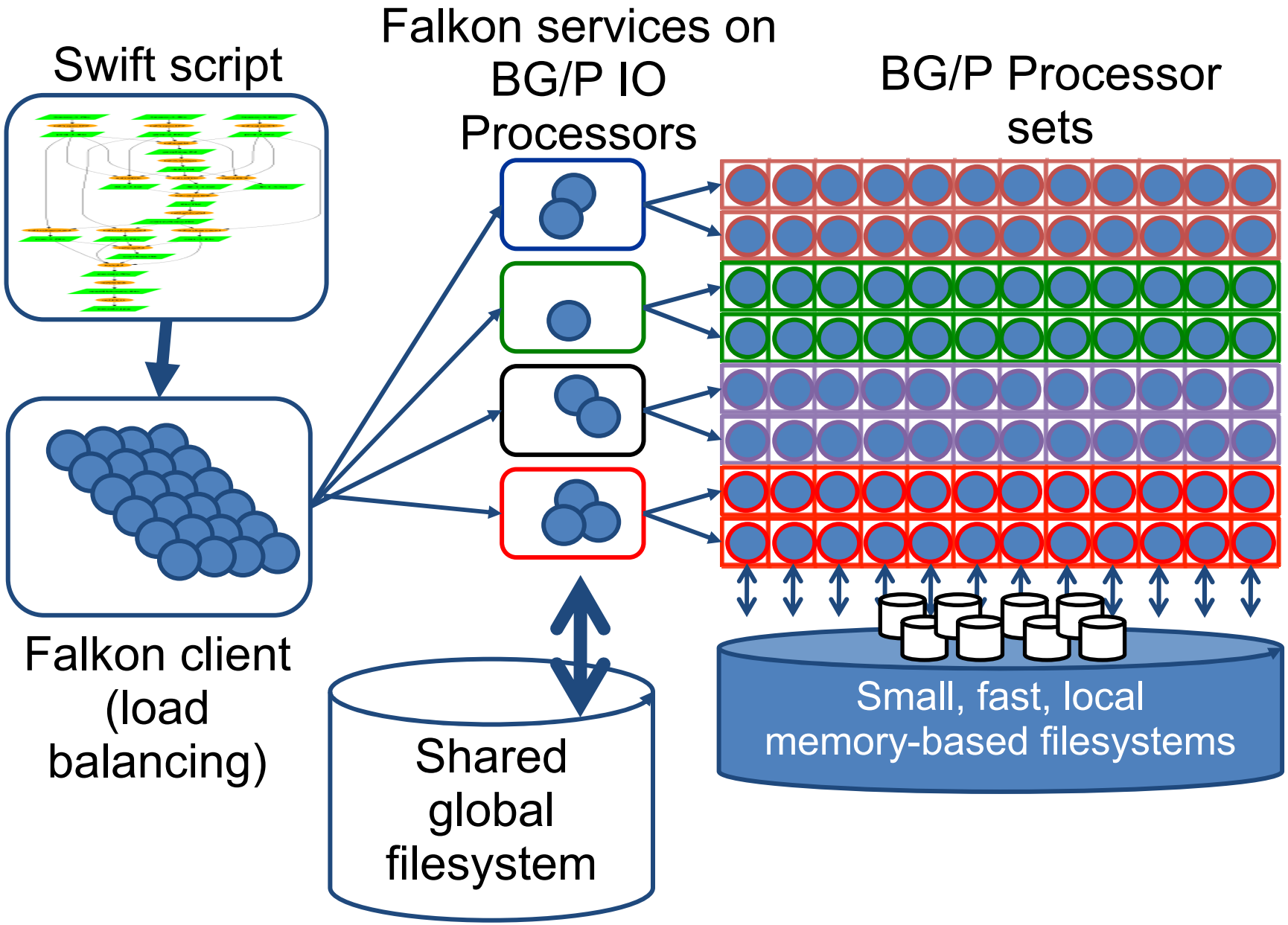
Using Swift



Swift is a self-contained application with cluster and grid client code:

Dowload, untar, and run

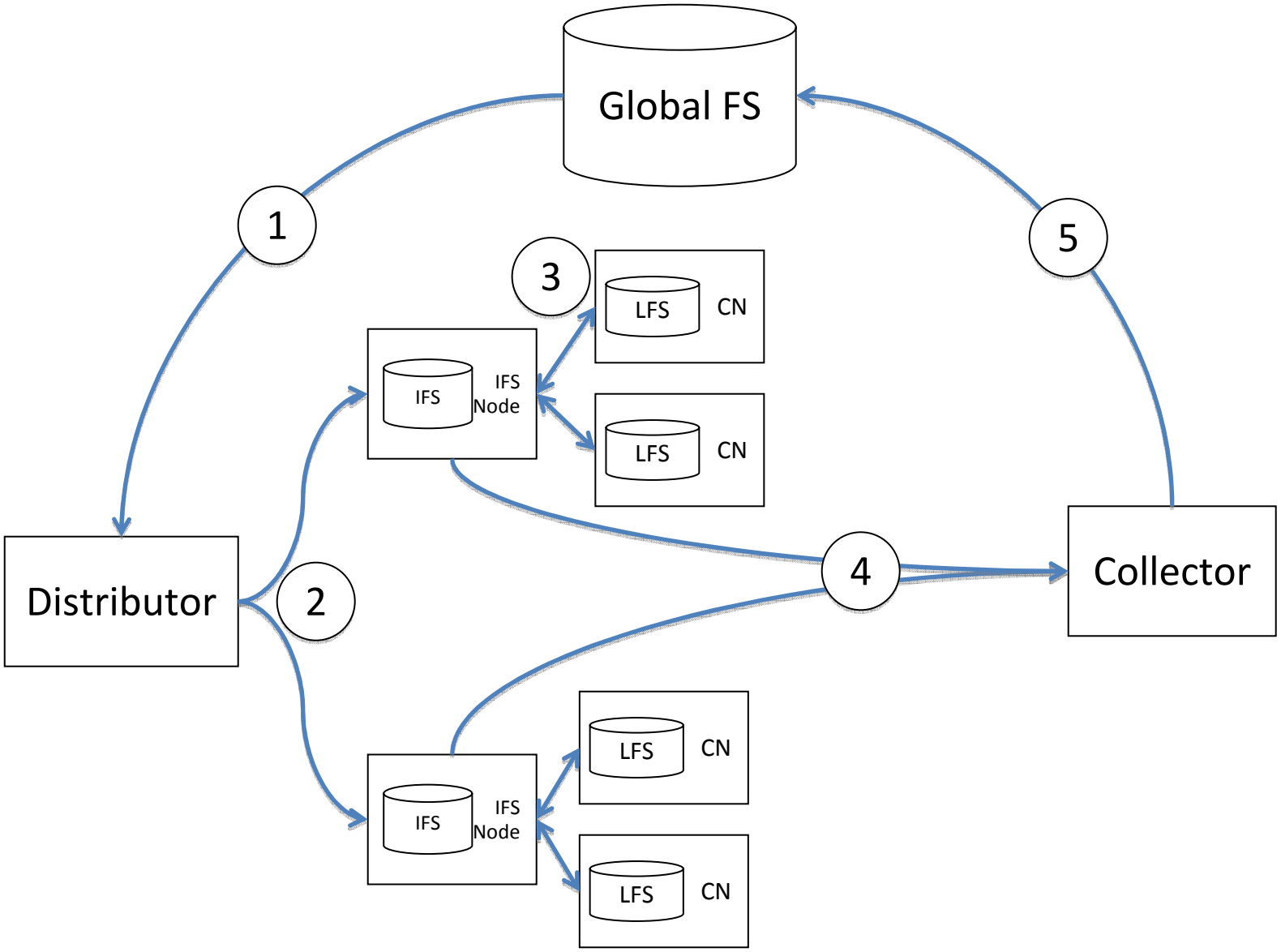
Architecture for petascale scripting



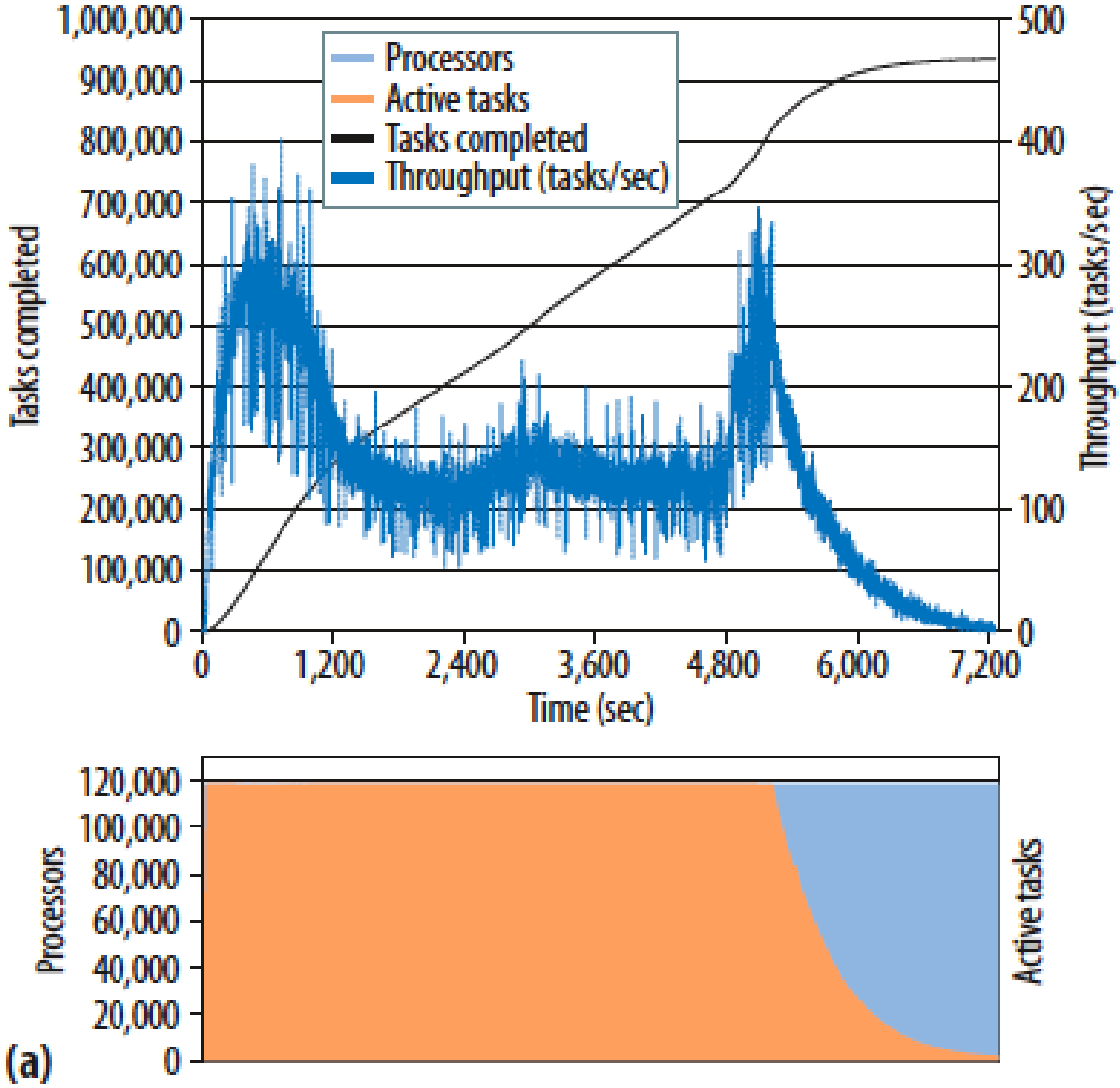
Collective data management is critical for petascale

- 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 phase: paradigm and architecures

Collective data management

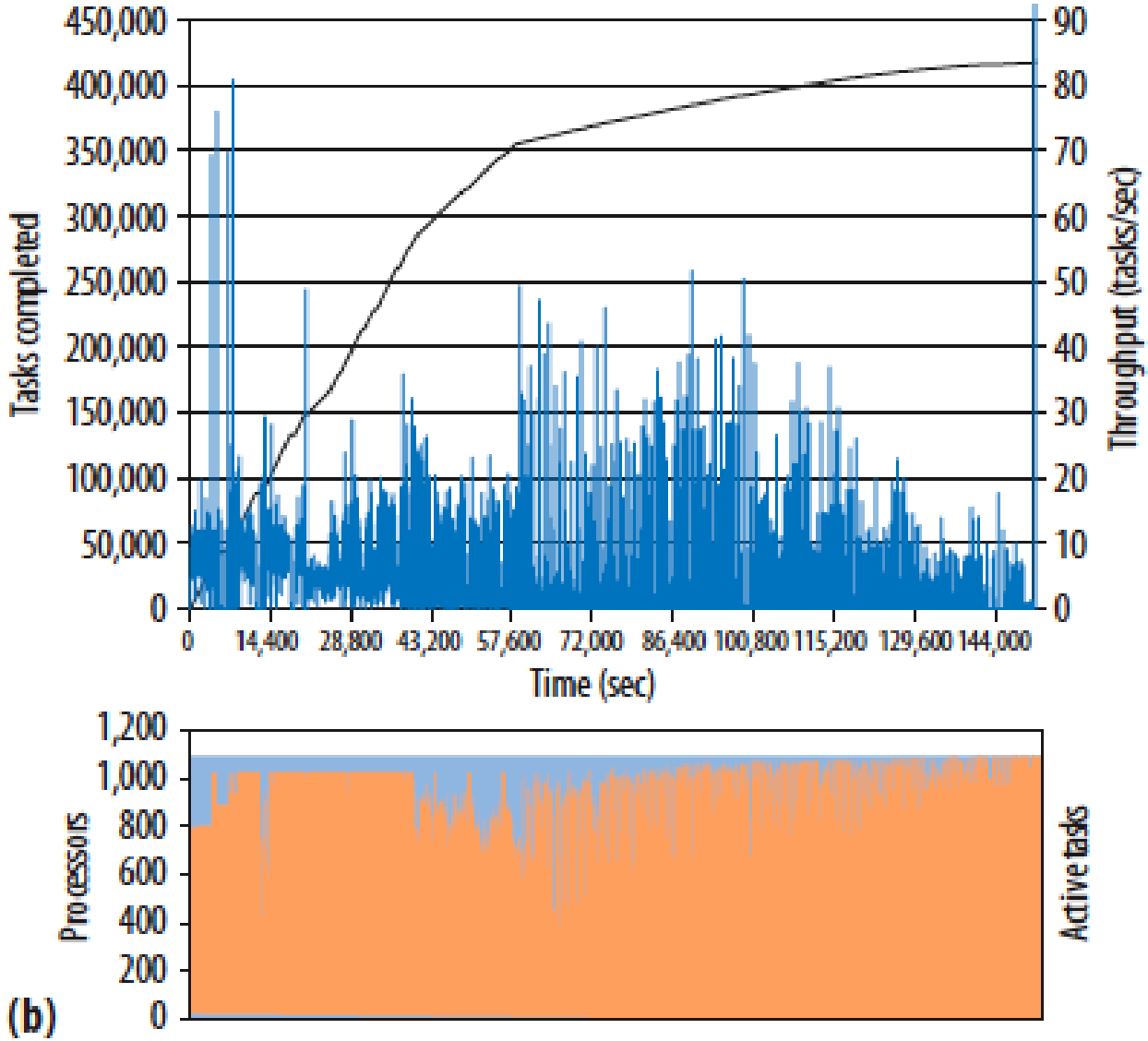


Performance: Molecular dynamics on BG/P



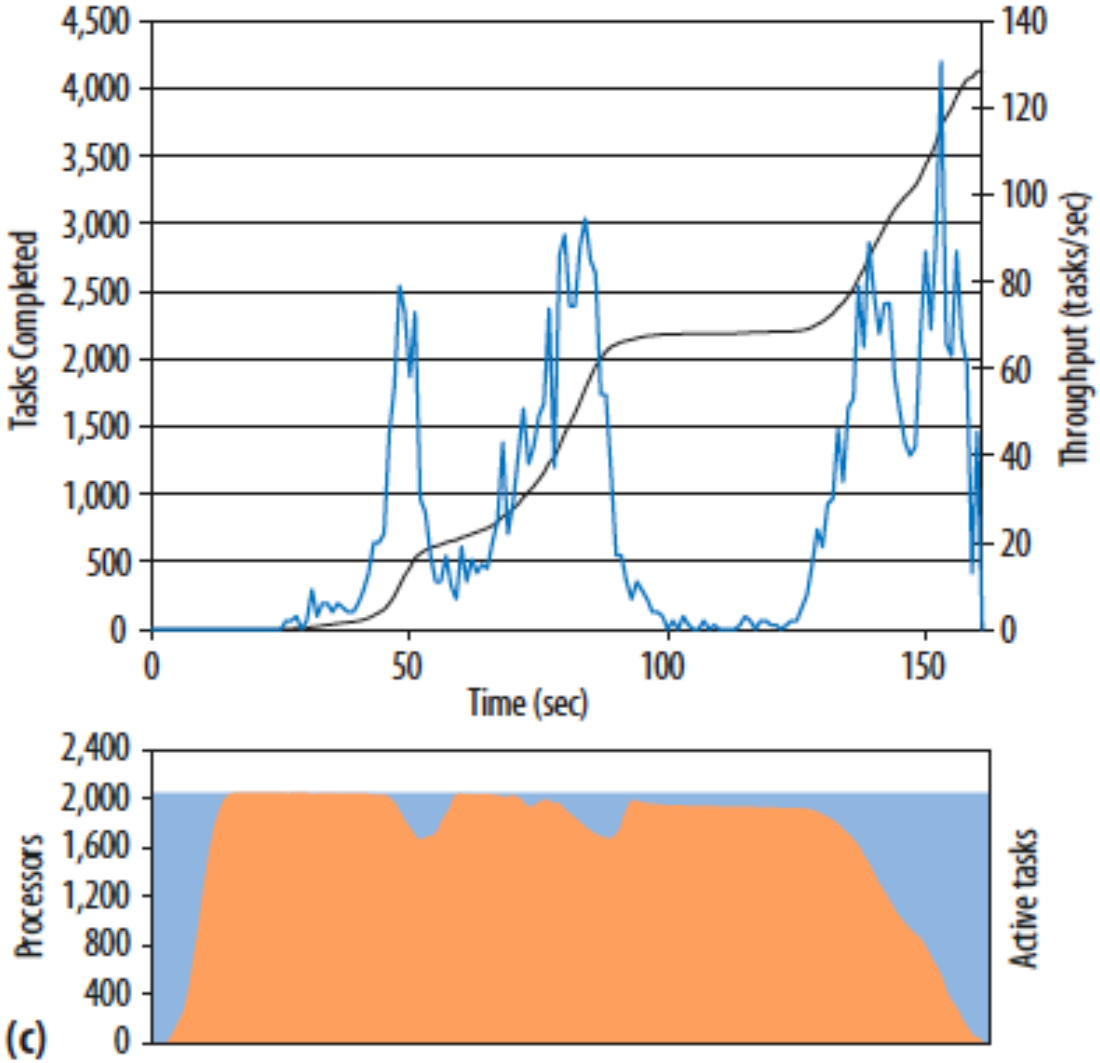
935,803 DOCK jobs with Falkon on BG/P in 2 hours

Performance: SEM for fMRI on Constellation



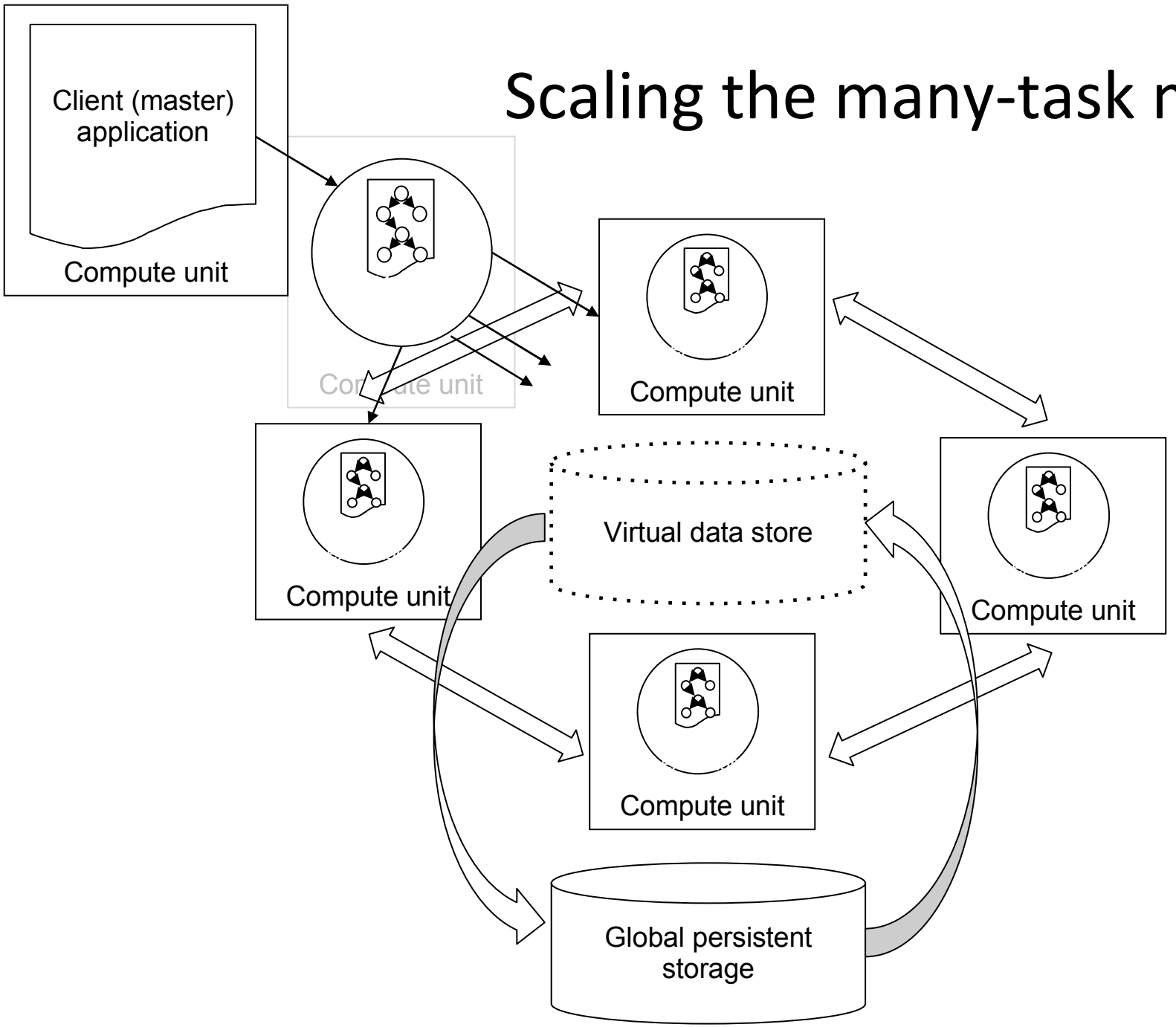
418K SEM tasks with Swift/Coasters on Ranger in 41 hours

Performance: Proteomics on BG/P



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

Scaling the many-task model



Scaling many-task computing

- ADLB: tasks can be lightweight functions
  - Retains RPC model of input-process-output
  - Fast, distributed, asynchronous load balancing
- Multi-level task manager
  - Must scale to massive computing complexes
- Transparent distributed management of local storage
  - Leverage local filesystems (RAM), aggregate, make access more transparent through DHT methods

Conclusion: Motivation for Swift

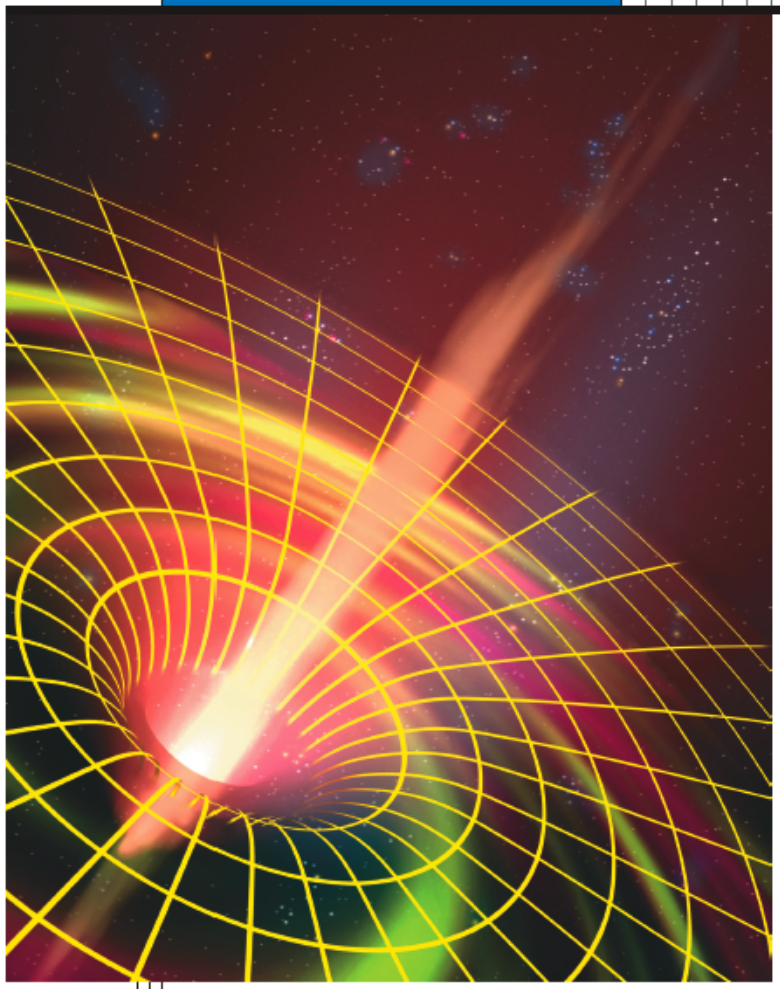
- Enhance scientific productivity
  - Location – and paradigm – independence: Same scripts run on workstations, clusters, clouds, grids, and petascale supercomputers
  - Automation of dataflow, resource selection and error recovery
- Enable and motivate collaboration
  - Community libraries of techniques, protocols, methods
  - Designed for recording the provenance of all data produced to facilitate scientific processes

- **Swift is a parallel scripting system for Grids and clusters**
  - for loosely-coupled applications - application and utility programs linked by exchanging files
- **Swift is easy to write**: simple high-level C-like functional language
  - *Small Swift scripts can do large-scale work*
- **Swift is easy to run**: contains all services for running Grid workflow - in one Java application
  - *Untar and run – acts as a self-contained Grid client*
- **Swift is fast**: Karajan provides Swift a powerful, efficient, scalable and flexible execution engine.
  - *Scaling close to 1M tasks – .5M in live science work, and growing*
- Swift **usage** is growing:
  - *applications in neuroscience, proteomics, molecular dynamics, biochemistry, economics, statistics, and more.*

To learn more and try Swift…

- 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

COVER FEATURE



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*

IEEE COMPUTER, Nov 2009

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 (present and former):
  - 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/Econ: Joshua Elliott, Meredith Franklin, Todd Muson, Tib Stef-Praun
  - PTMap: Yingming Zhao, Yue Chen