Argonne NATIONAL LABORATORY

Swift: implicitly parallel dataflow for cloud scripting and HPC programming

www.ci.uchicago.edu/swift

www.mcs.anl.gov/exm

Michael Wilde - wilde@mcs.anl.gov

U.S. DEPARTMENT OF ENERGY

Touchpoints for collaboration

- *Programming model:* implicitly parallel functional dataflow
  - *Simplify scientific, engineering, and analytic programming by abstracting **parallelism**, **location**, and **failure***
  - *Focus on helping people that want to **run** code, not write it*
- *A worker agent model that enables the model to run on diverse platforms*
  - *Multicores, clusters, grids, clouds, and supercomputers*
- *Flexible implementation with intermediate languages*
  - *Swift/K uses Karajan*
  - *Swift/T used ANTLR -> IR -> Turbine Intermediate Code (Tcl)*
- *A fully parallel evaluation model for extreme scaling*
  - *Leaf functions can be POSIX apps (fork/exec) or library functions*
  - *Data objects can be files or in-memory objects*
- *Multiple language bindings for the model*
  - *Native (C-like), Python ("PyDFlow"), R (Swift-R)*

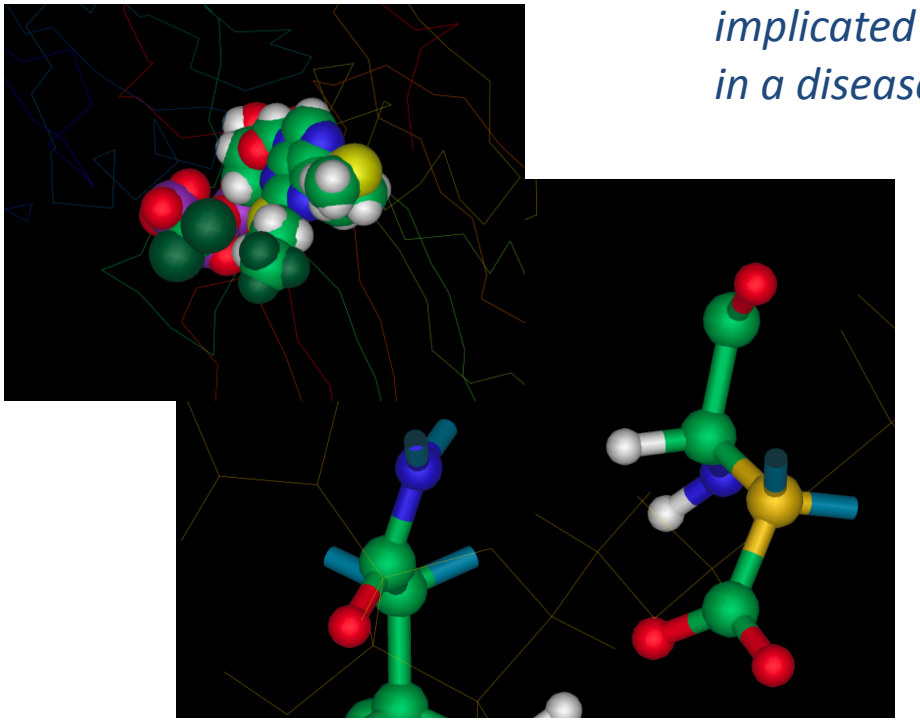
A collaboration of many people

- Core team
  - Justin Wozniak, MCS (lead researcher)
  - David Kelly (full time), Mihael Hategan (part time) – development and user engagement
  - Tim Armstrong, Zhao Zhang (students)
- PIs involved
  - Ian Foster, Rusty Lusk, Dan Katz, Matei Ripeanu, Todd Munson: co-PIs and senior staff on DOE X-Stack "ExM"
  - Ian is using Swift in NSF RDCEP and CIM-EARTH projects (J. Elliott)
  - Dan (now at NSF) was UChicago PI for NSF "ExTENCI", using Swift
  - Rob Jacob, PI of DOE ParVis (climate) using Swift (ALCF & NCAR)
  - Tobin Sosnick, Karl Freed (UChicago) Jinbo Xu (TTI) – protein science
  - Greg Voth – using Swift in Center for Multiscale Theory and Simulation (NSF CCI)
- Close collaborators
  - Mark Hereld, Tom Uram, Mike Papka: GPSI portal
  - Kate Keahey, Ravi Madduri, Raj Kettimuthu, Borja Sotamajor
  - Ketan Maheshwari, MCS Postdoc (ExTENCI and Beagle) – going to Cornell

When do you need Swift?

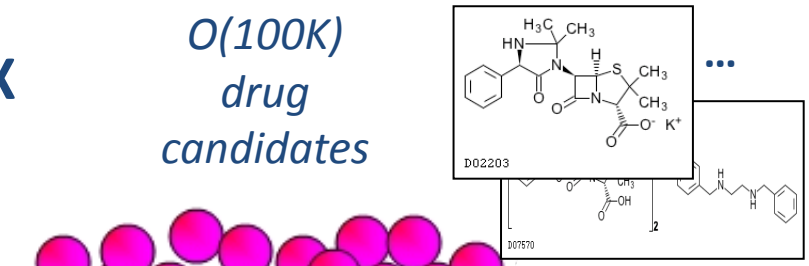
Typical application: protein-ligand docking for drug screening

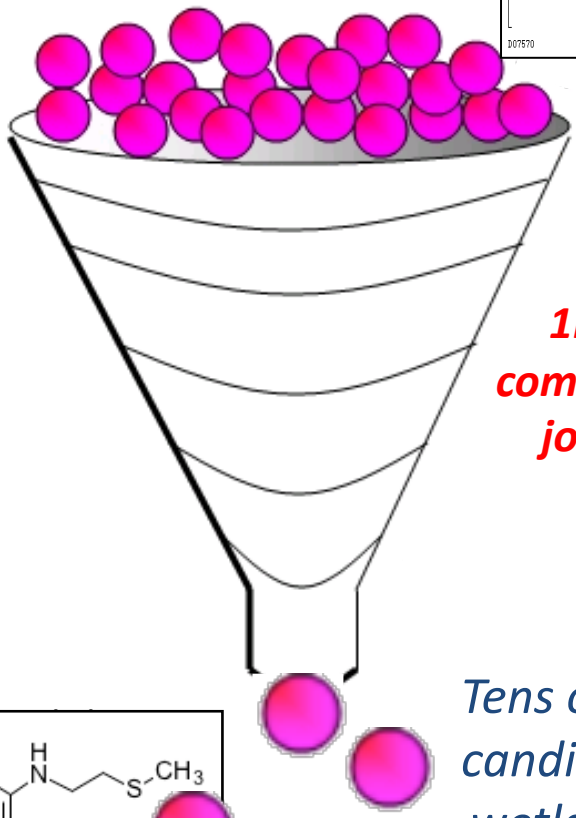
O(10) proteins implicated in a disease



X

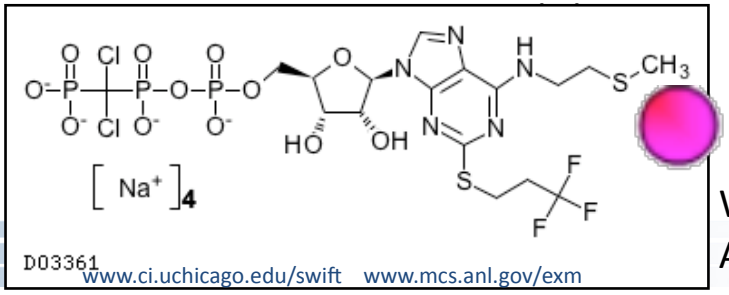
O(100K) drug candidates





1M compute jobs

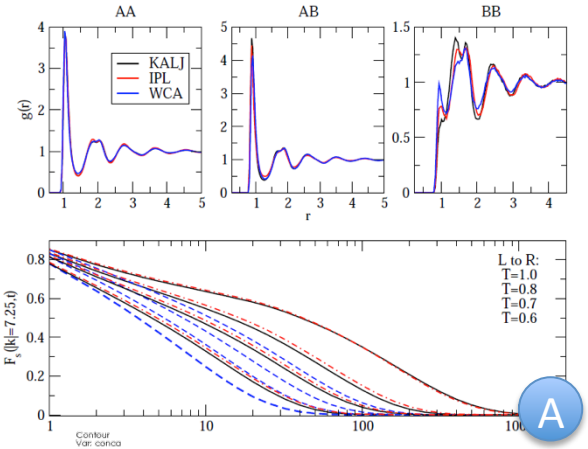
Tens of fruitful candidates for wetlab & APS

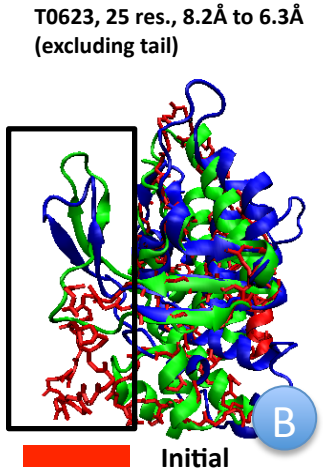


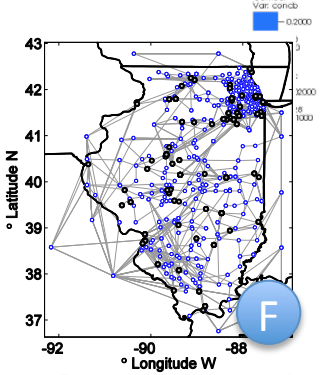
Work of M. Kubal, T.A.Binkowski, And B. Roux

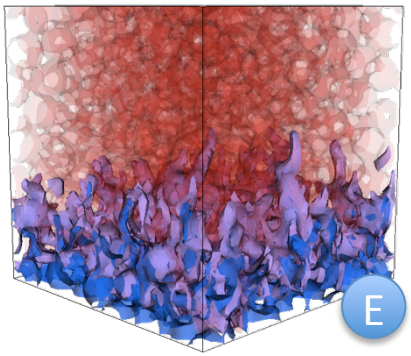
Numerous many-task applications

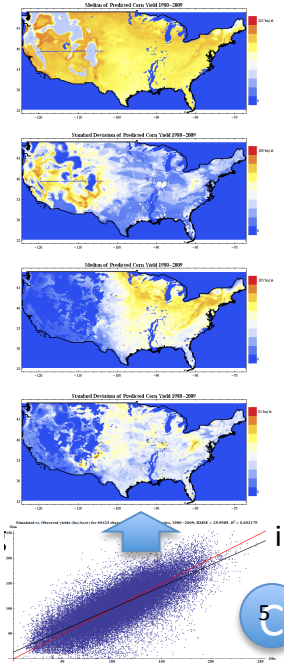
- A — Simulation of super-cooled glass materials
- B — Protein folding using homology-free approaches
- C — Climate model analysis and decision making in energy policy
- D — Simulation of RNA-protein interaction
- E — Multiscale subsurface flow modeling
- F — Modeling of power grid for OE applications
- > — A-E have published science results obtained using Swift

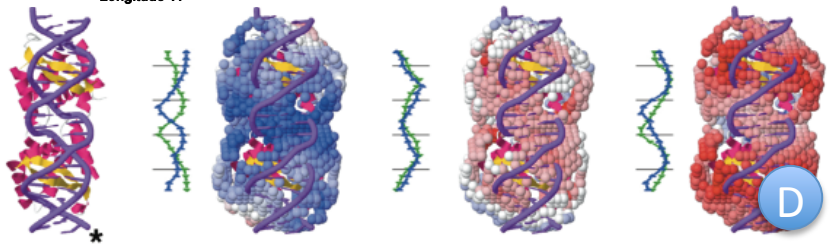




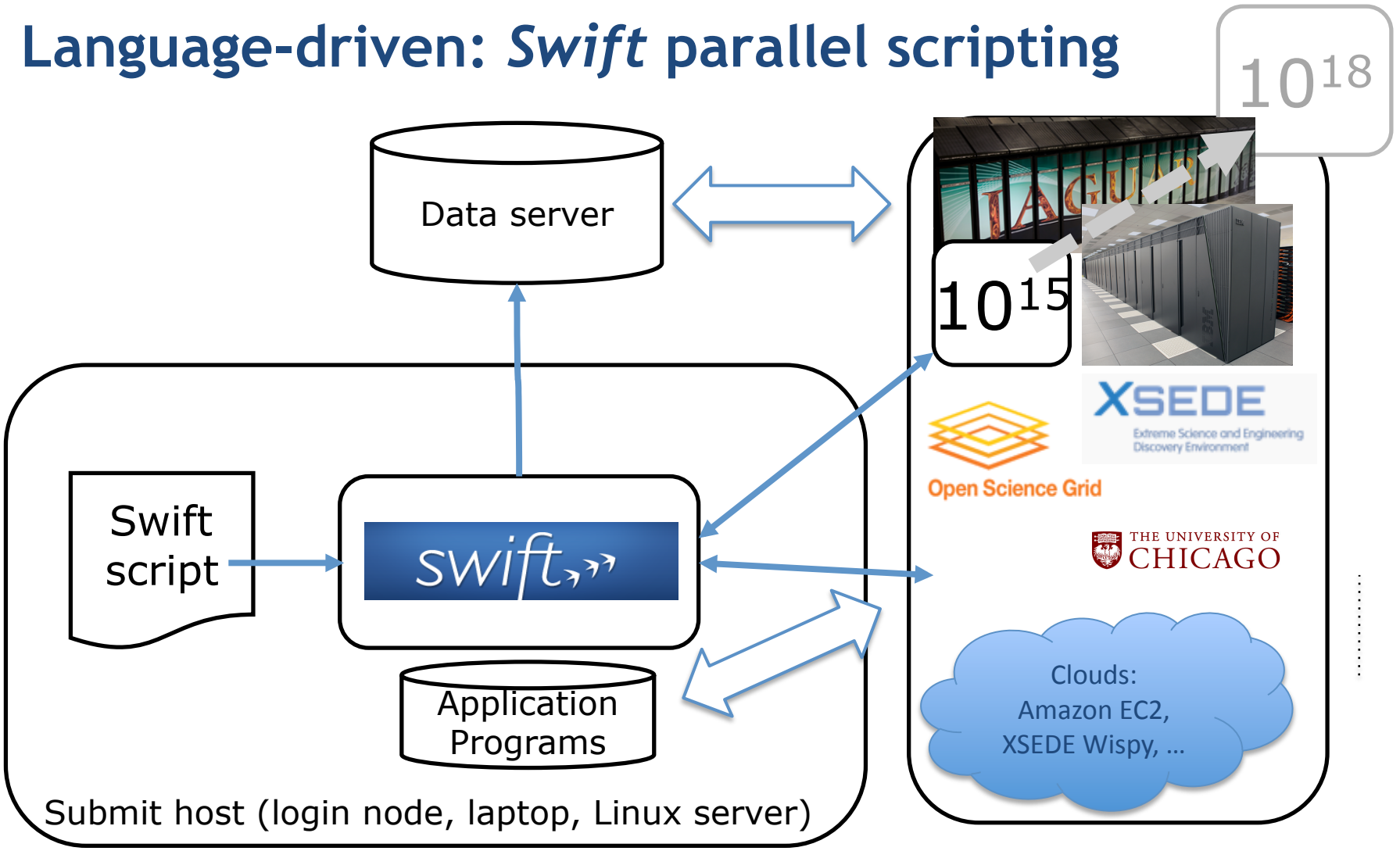








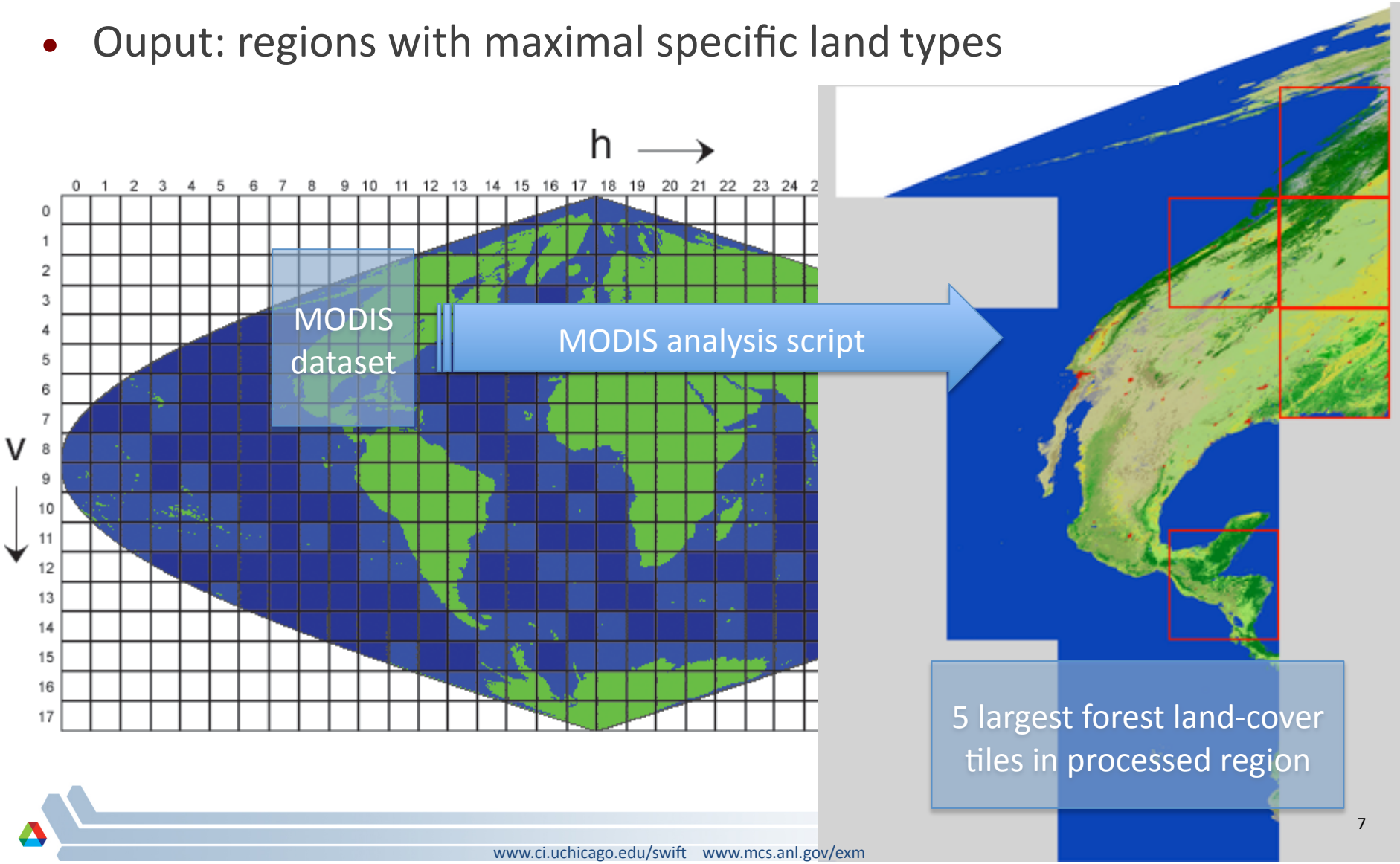
Language-driven: *Swift* parallel scripting



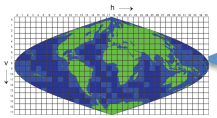
Swift runs parallel scripts on a broad range of parallel computing resources.

Example - MODIS satellite image processing

- Input: tiles of earth land cover (forest, ice, water, urban, etc)
  - Ouput: regions with maximal specific land types



Goal: Run MODIS processing pipeline in cloud



getLandUse x 317

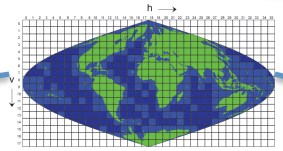
colorMODIS

analyzeLandUse

markMap

assemble

MODIS script is automatically run in parallel:



getLandUse x 317

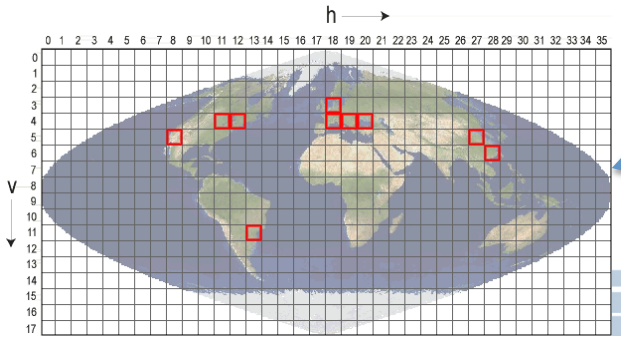
colorMODIS x 317

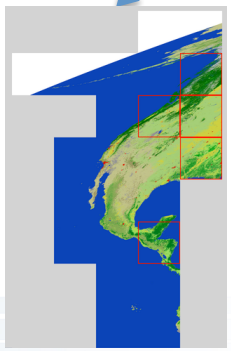
analyzeLandUse

assemble

Each loop level can process tens to thousands of image files.

markMap

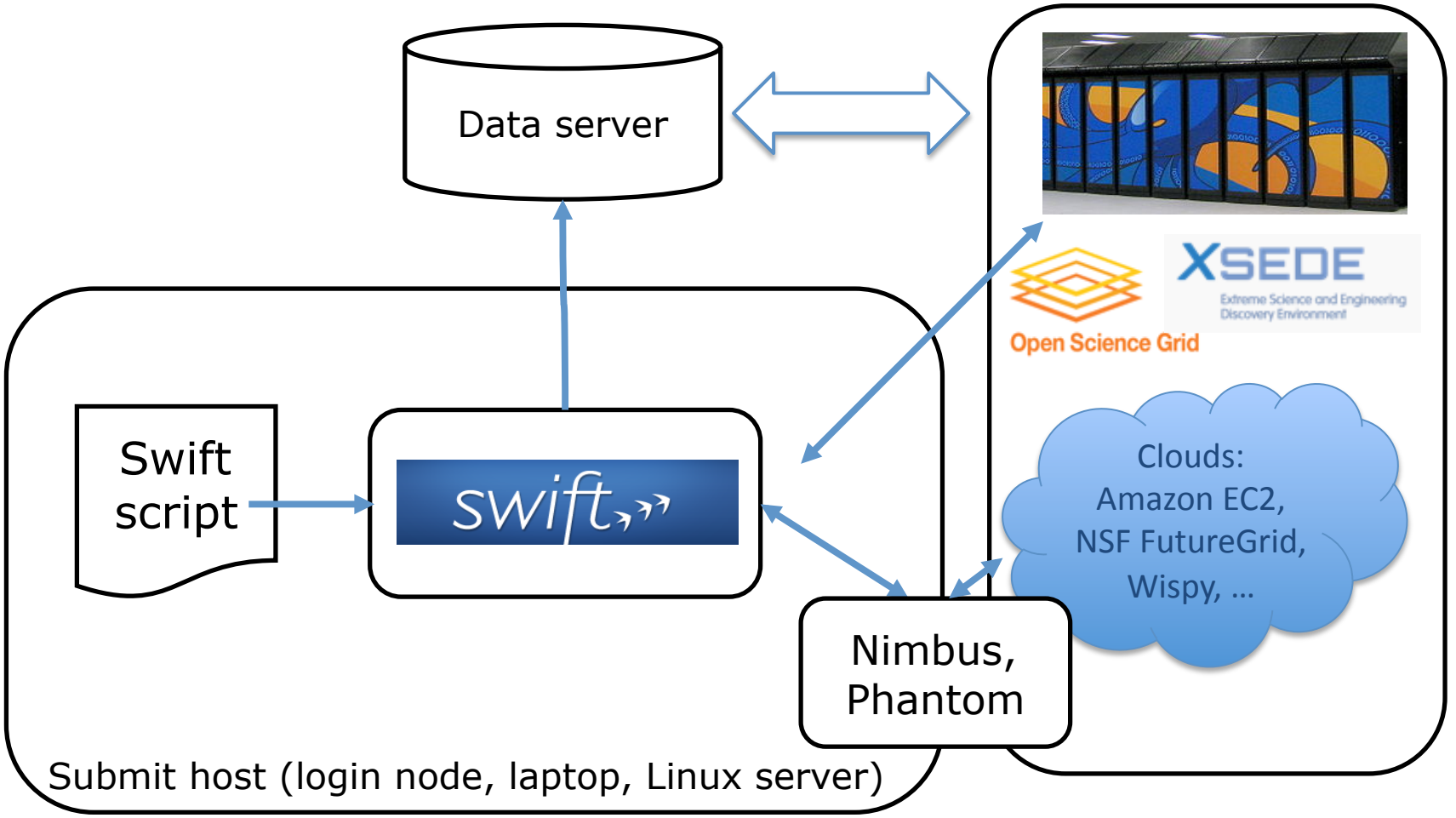




MODIS script in Swift: main data flow

```
foreach g,i in geos {
    land[i] = getLandUse(g,1);
}
(topSelected, selectedTiles) =
    analyzeLandUse(land, landType, nSelect);

foreach g, i in geos {
  colorImage[i] = colorMODIS(g);
}
gridMap = markMap(topSelected);
montage =
   assemble(selectedTiles,colorImage,webDir);
```

Parallel distributed scripting for clouds



Swift runs parallel scripts on cloud resources provisioned by Nimbus's *Phantom* service.

Example of Cloud programming: Nimbus-Phantom-Swift on FutureGrid

- User provisions 5 nodes with Phantom
  - Phantom starts 5 VMs
  - Swift worker agents in VMs contact Swift coaster service to request work
- Start Swift application script “MODIS”
  - Swift places application jobs on free workers
  - Workers pull input data, run app, push output data
- 3 nodes fail and shut down
  - Jobs in progress fail, Swift retries
- User can add more nodes with phantom
  - User asks Phantom to increase node allocation to 12
  - Swift worker agents register, pick up new workers, runs more in parallel
- Workload completes
  - Science results are available on output data server
  - Worker infrastructure is available for new workloads

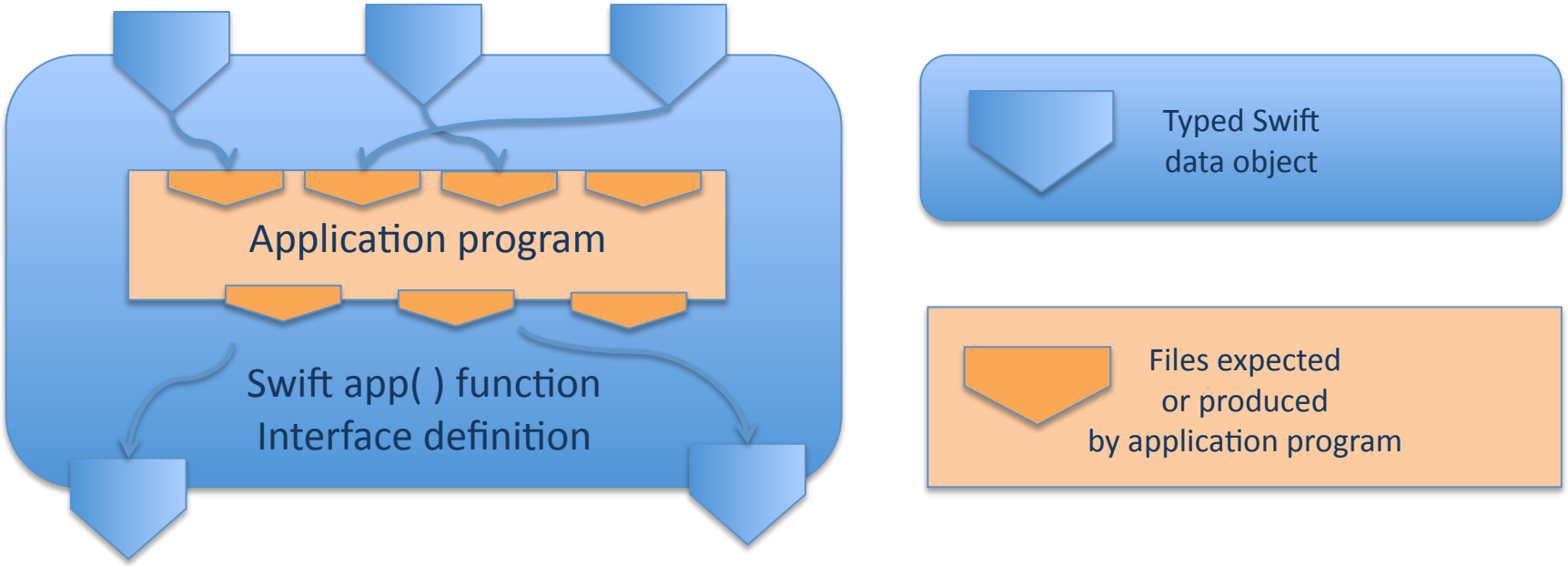
Programming model: all execution driven by parallel data flow

```
(int r) myproc (int i)
{
    j = f(i);
    k = g(i);
    r = j + k;
}
```

- f() and g() are computed in parallel
- myproc() returns r when they are done

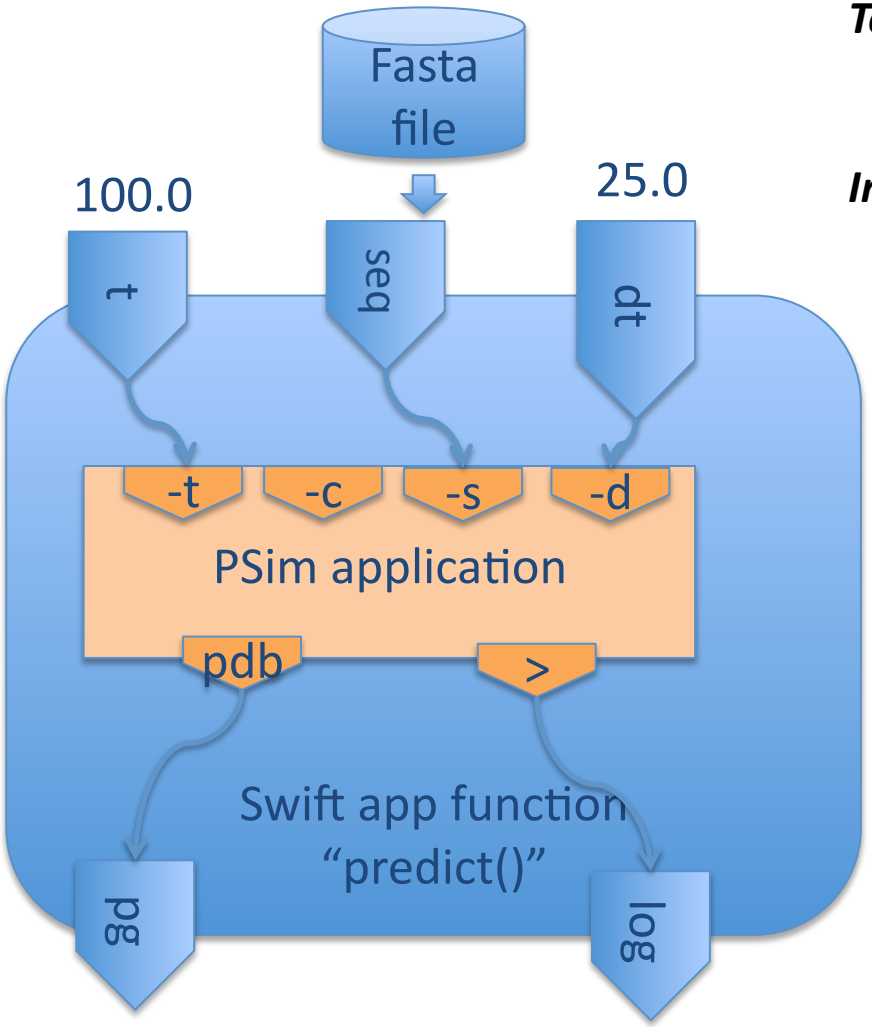
- This parallelism is *automatic*
- Works recursively throughout the program's call graph

Encapsulation enables distributed parallelism



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

app( ) functions specify cmd line argument passing



To run:

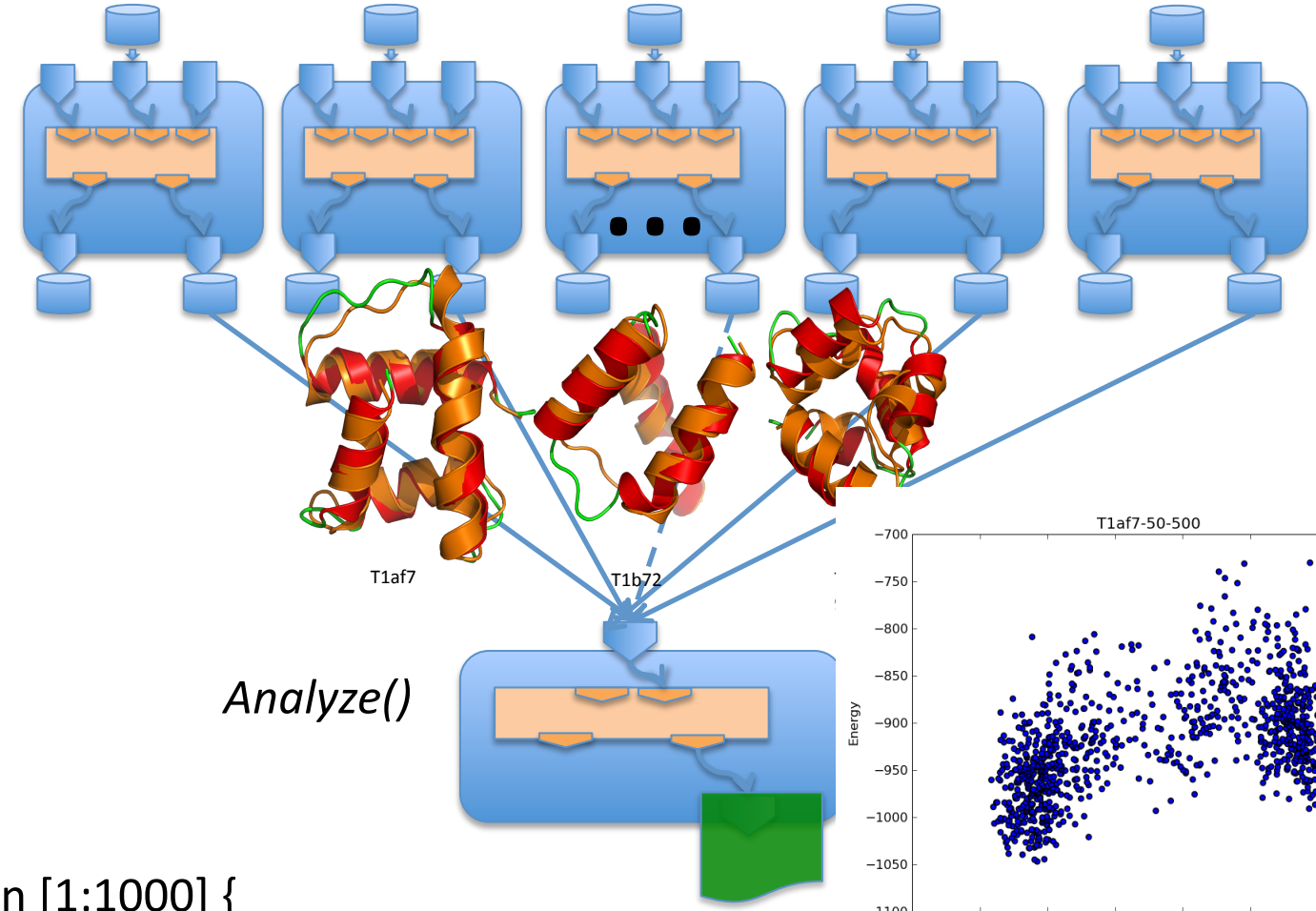
psim -s 1ubq.fas -pdb p -t 100.0 -d 25.0 >log

In Swift code:

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

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

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

Large scale parallelization with simple loops

1000 Runs of the "predict" application



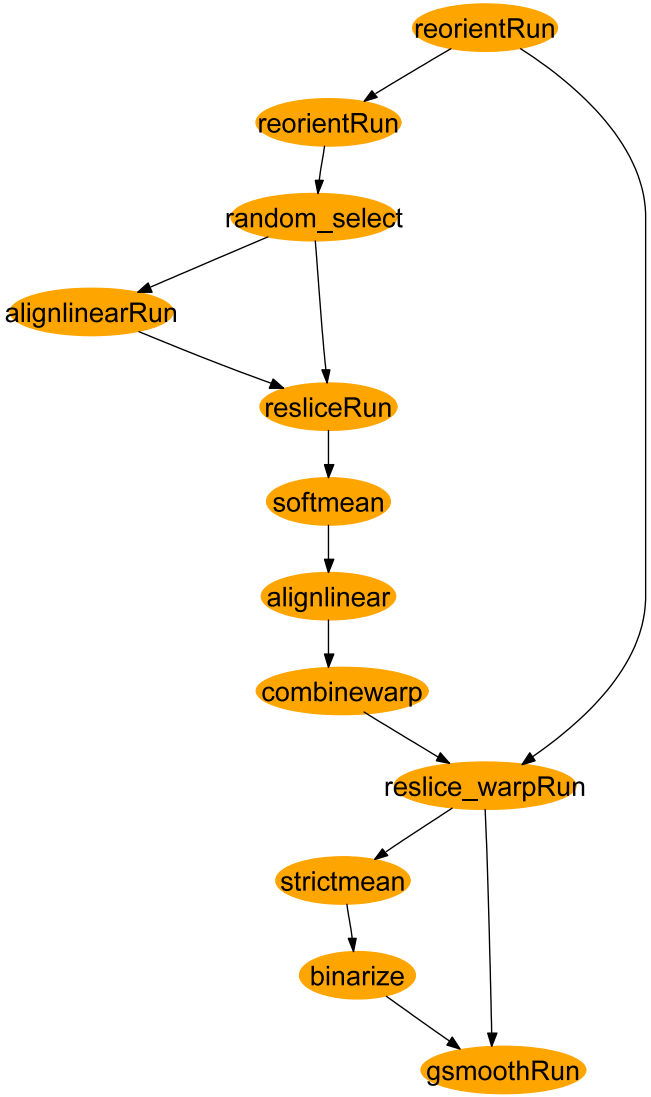

```
foreach sim in [1:1000] {
   (structure[sim], log[sim]) = predict(p, 100., 25.);
}
result = analyze(structure)
```

Nested parallel prediction loops in *Swift*

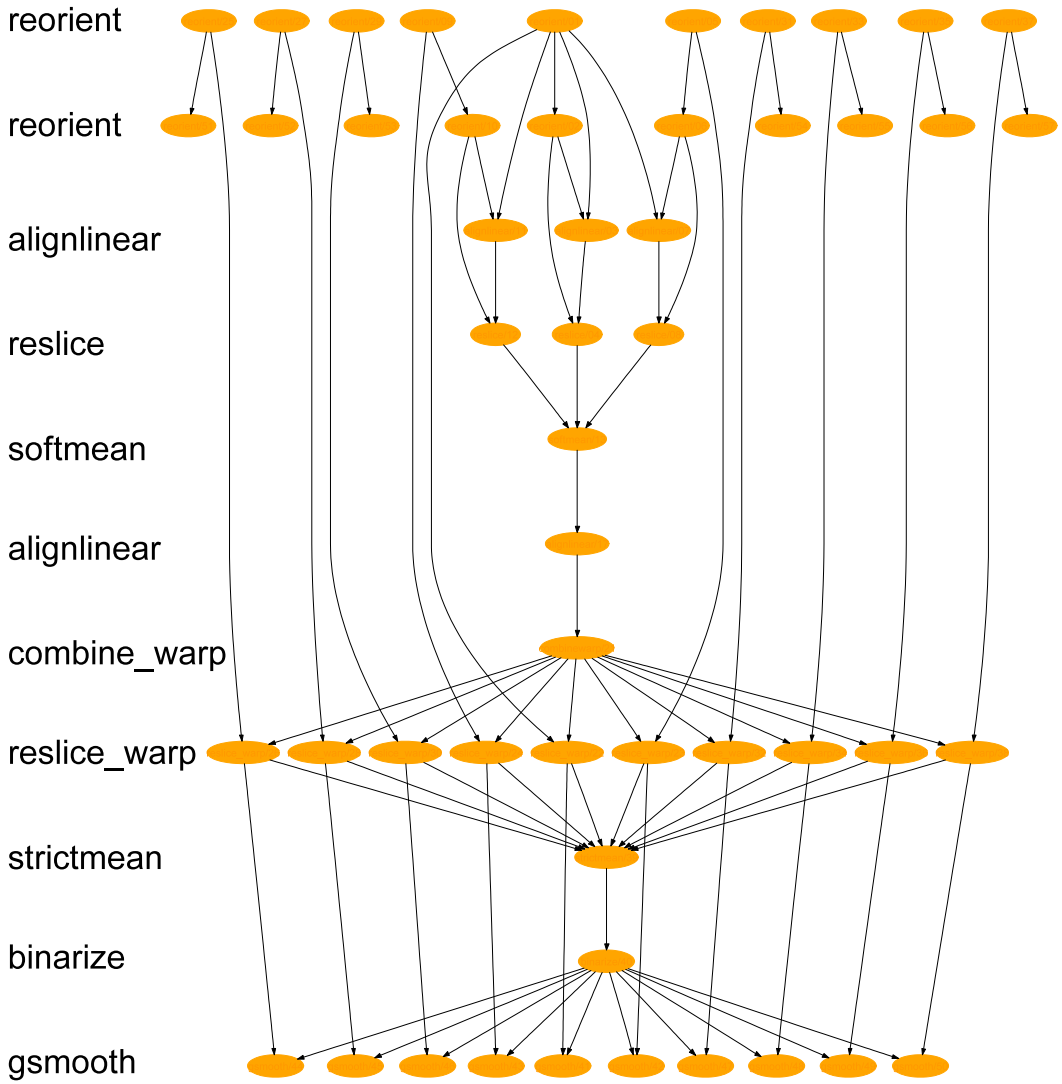
```
1.  Sweep( )
2.  {
3.    int nSim = 1000;
4.    int maxRounds = 3;
5.    Protein pSet[ ] <ext; exec="Protein.map">;
6.    float startTemp[ ] = [ 100.0, 200.0 ];
7.    float delT[ ] = [ 1.0, 1.5, 2.0, 5.0, 10.0 ];
8.    foreach p, pn in pSet {
9.      foreach t in startTemp {
10.       foreach d in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

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

Spatial normalization of functional run



Dataset-level workflow



Expanded (10 volume) workflow

Complex scripts can be well-structured

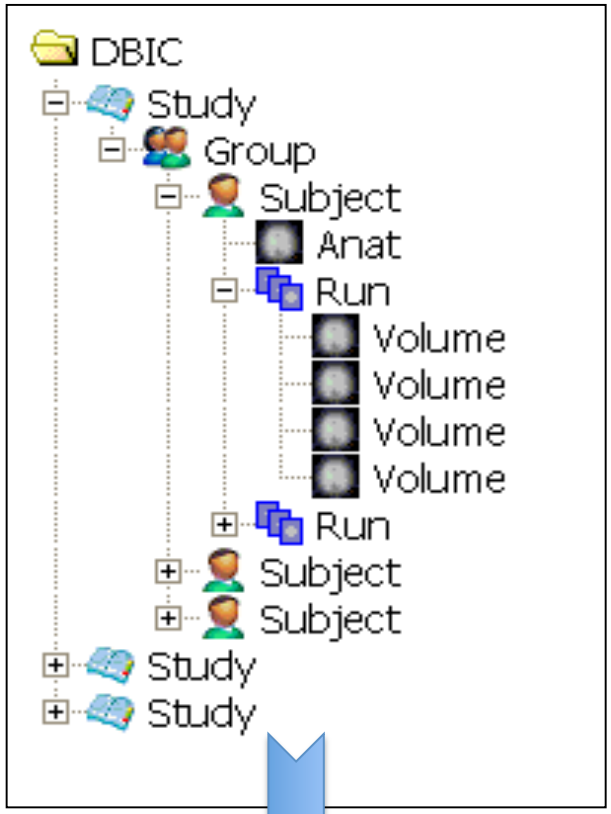
programming in the large: fMRI spatial normalization script example

```
(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);
        }
}
```

Dataset mapping example: fMRI datasets

On-Disk Data Layout

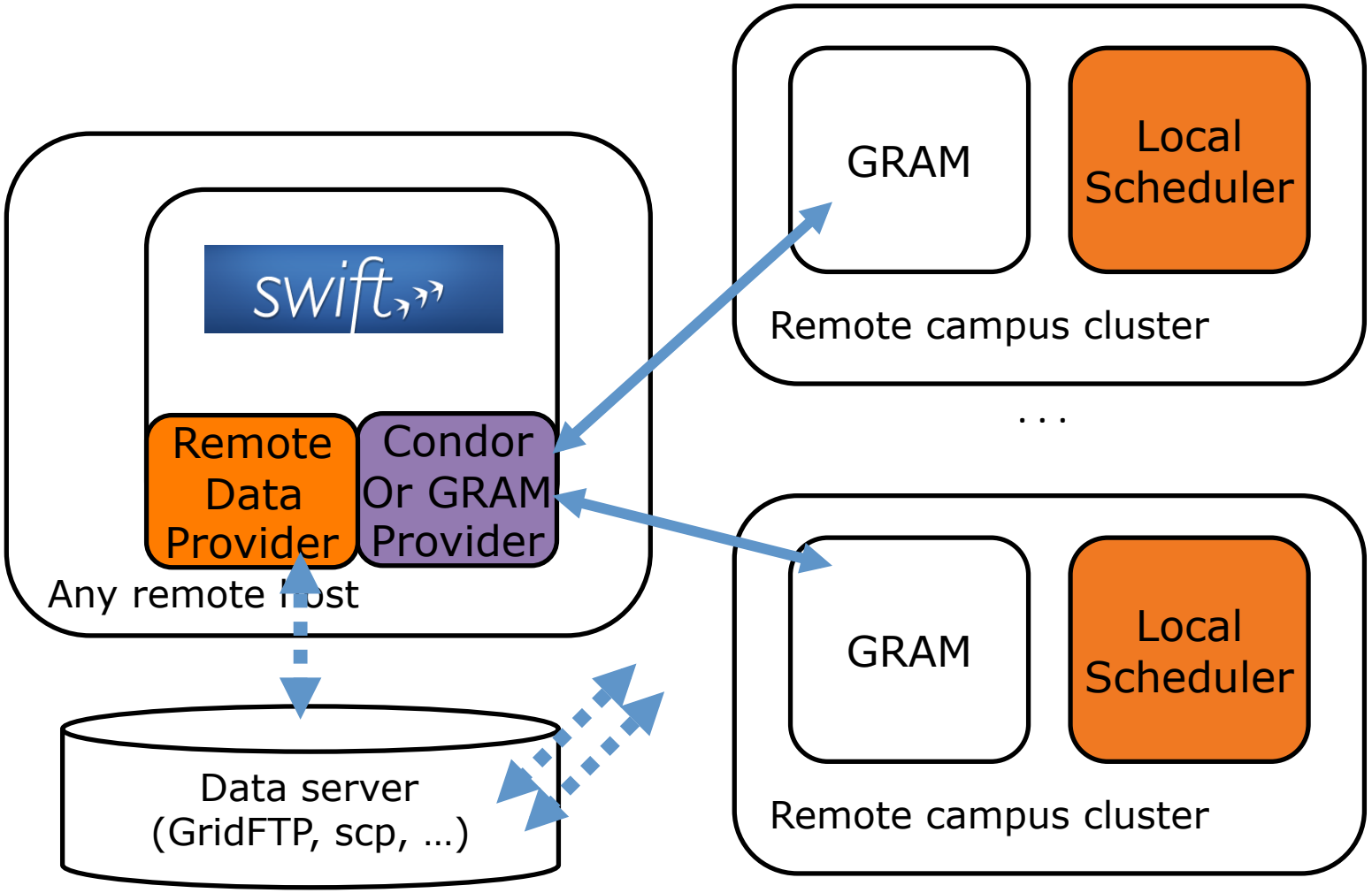


Mapping function or script

```
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

Swift can send work to multiple sites

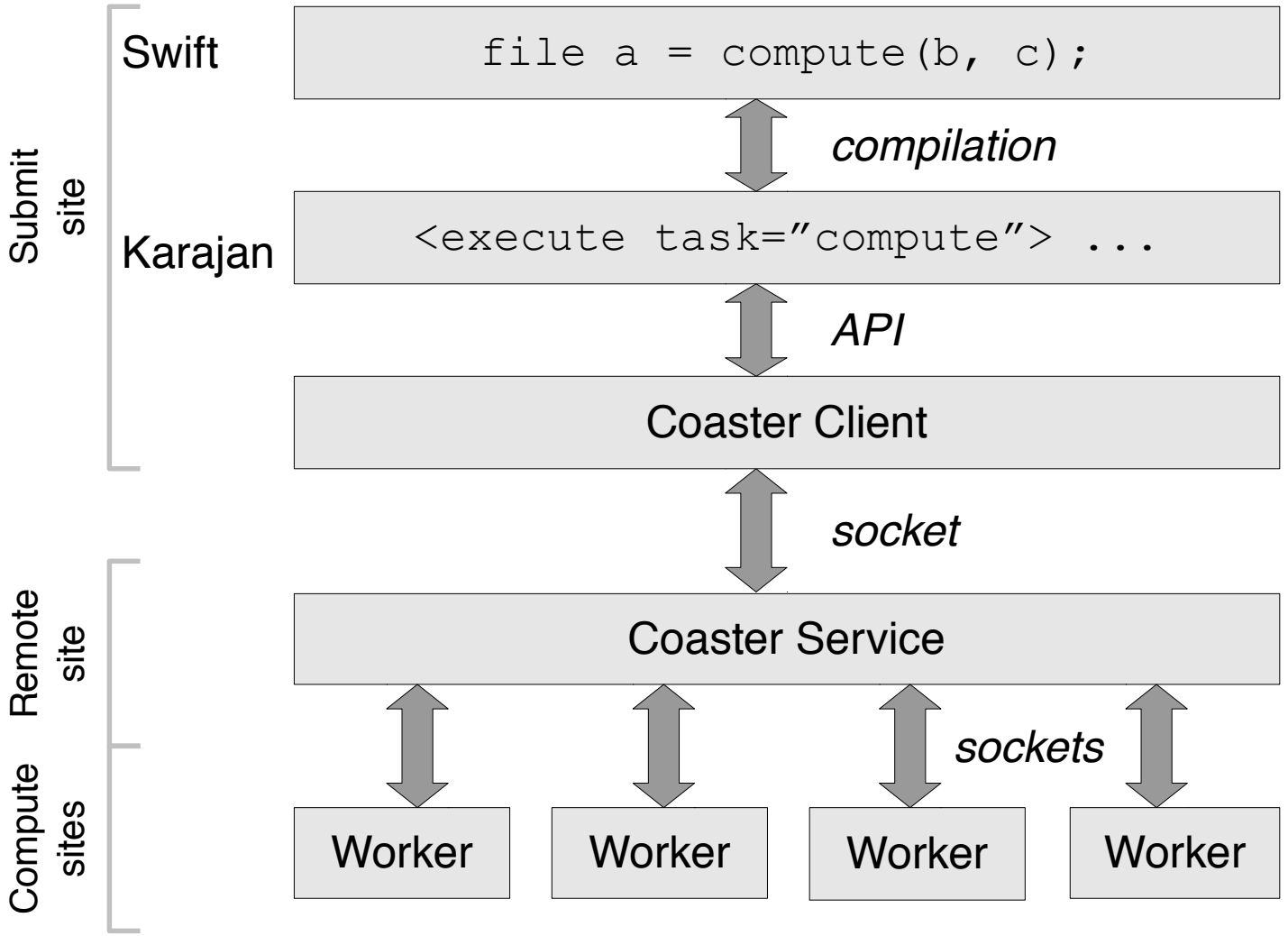


Simple campus Swift usage: running locally on a cluster login host

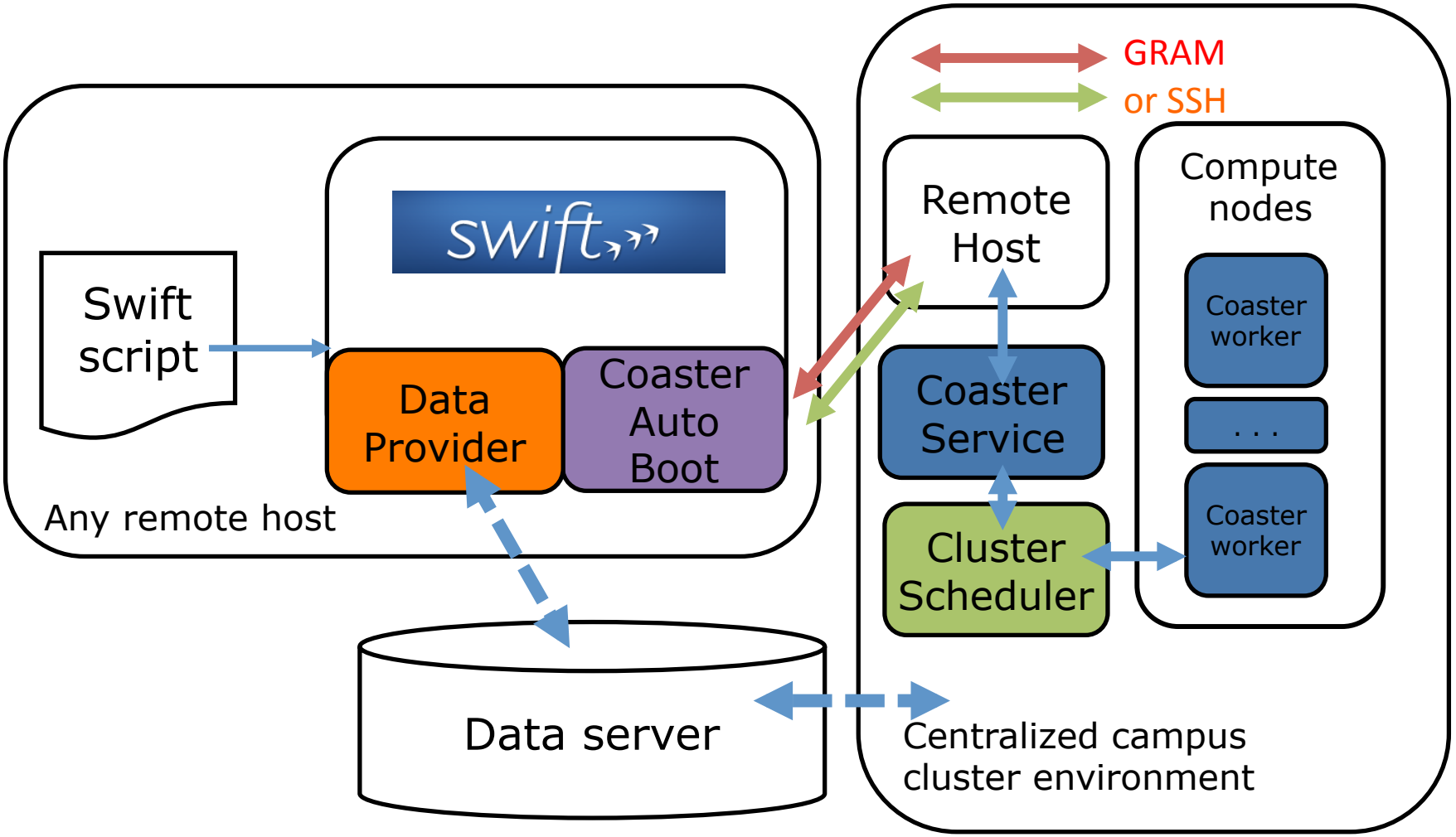
Flexible worker-node agents for execution and data transport

- Main role is to efficiently run Swift tasks on allocated compute nodes, local and remote
- Handles short tasks efficiently
- Runs over Grid infrastructure: Condor, GRAM
- Also runs with few infrastructure dependencies
- Can optionally perform data staging
- Provisioning can be automatic or external (manually launched or externally scripted)

Coaster architecture handles diverse environments

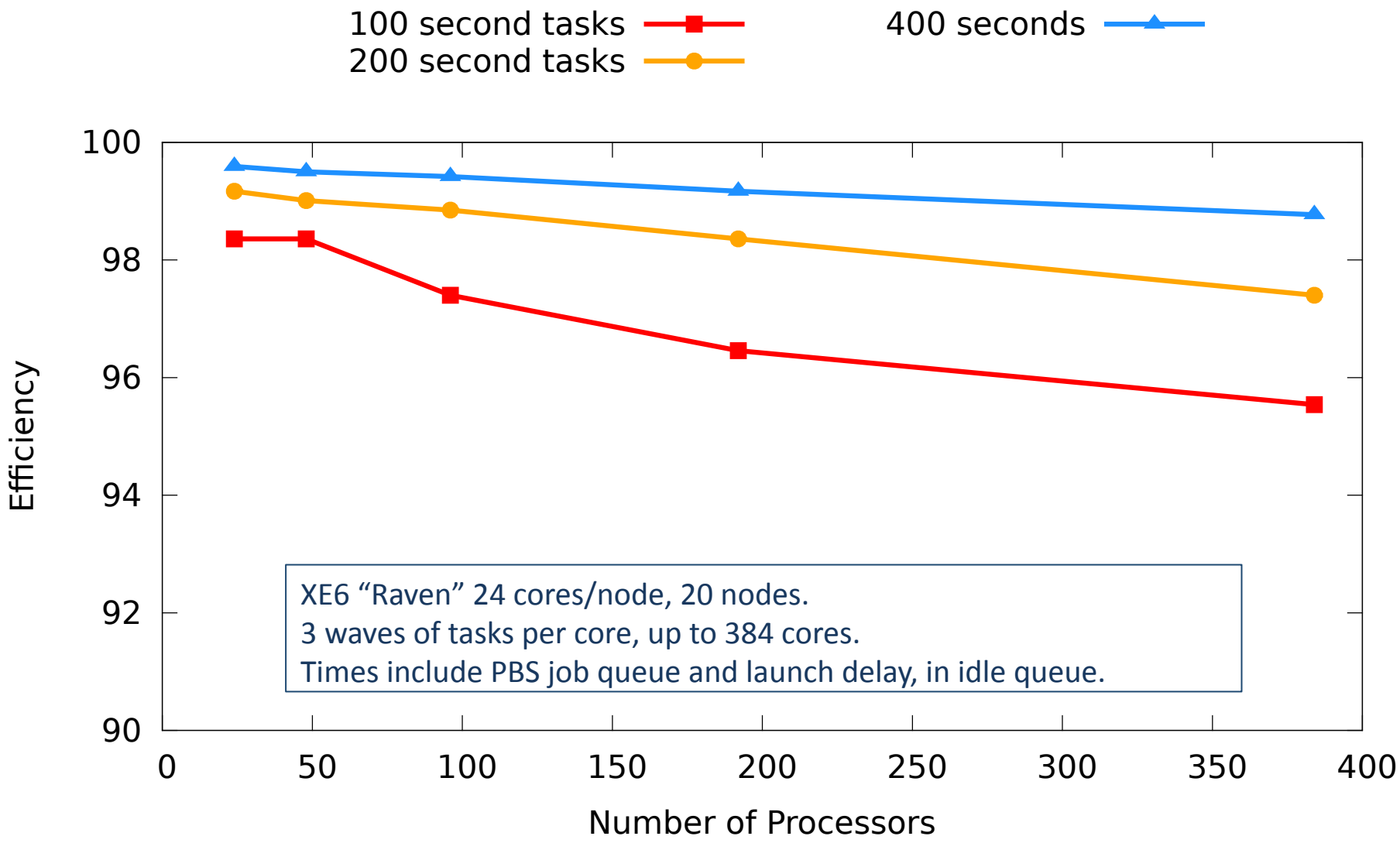


Coasters deployment can be automatic or manual

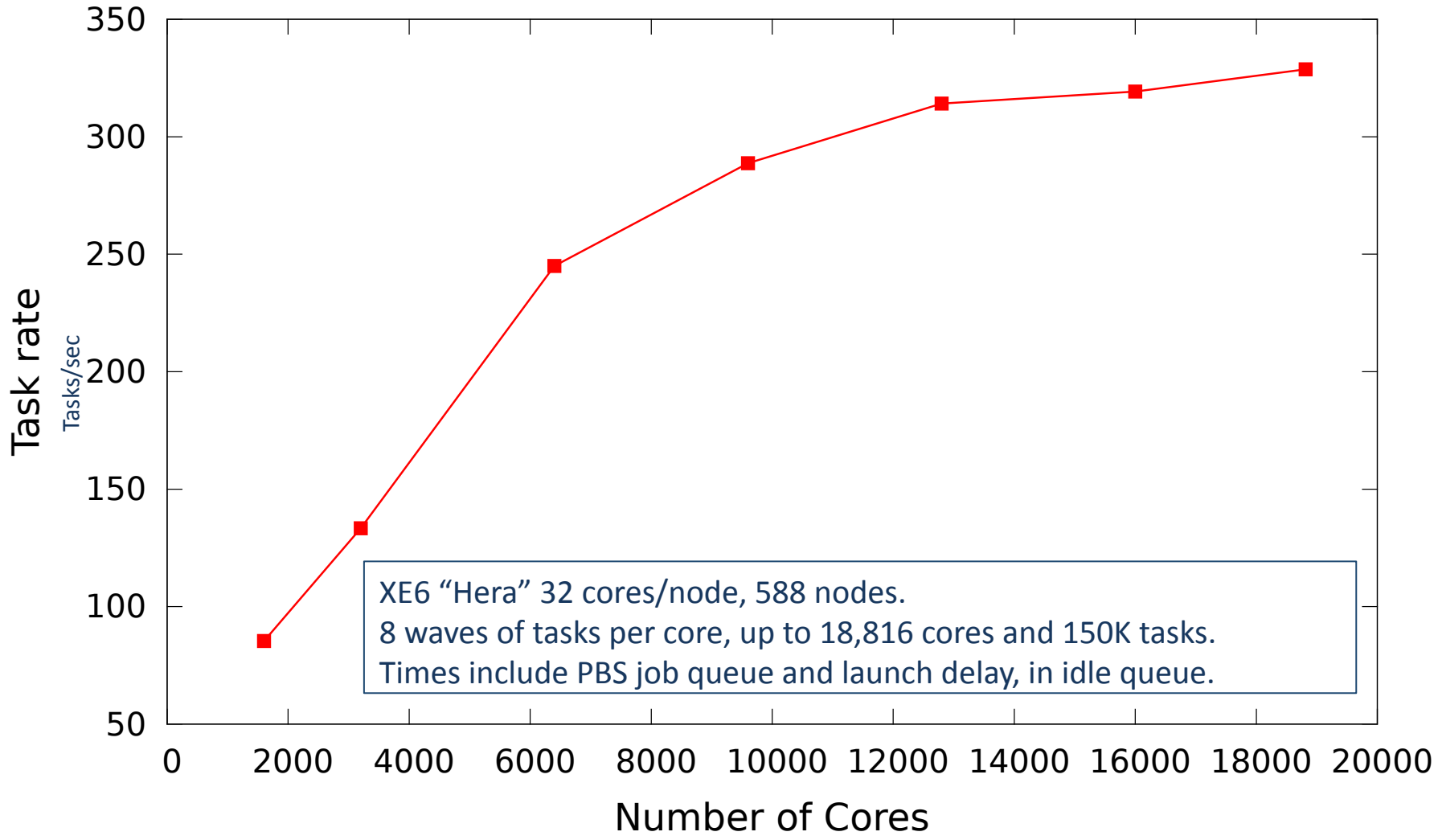


Swift can deploy coasters automatically, or user can deploy manually or with a script. Swift provides scripts for clouds, ad-hoc clusters, etc.

Swift efficiency on Cray XE6 test system “raven”



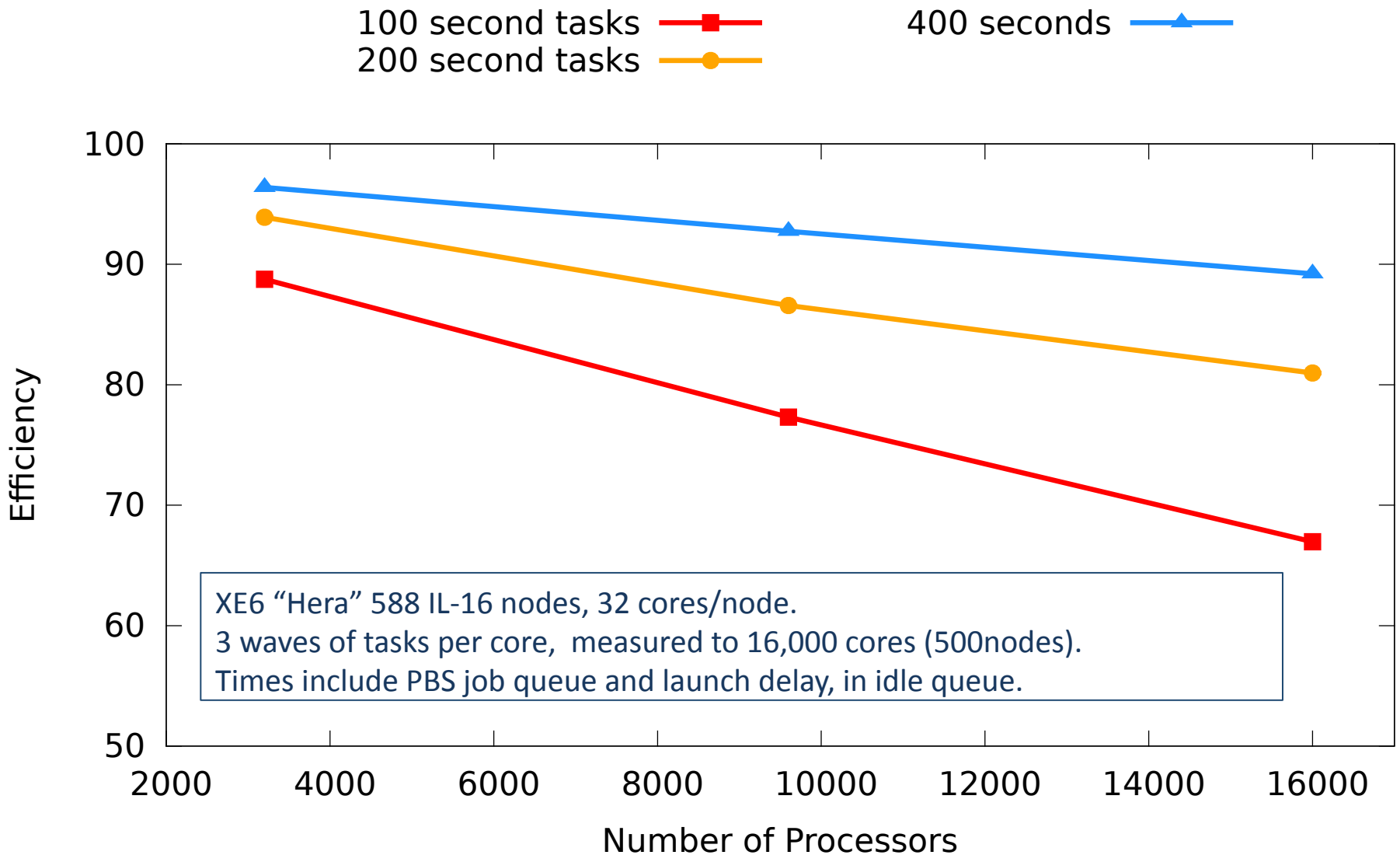
Swift task rates on Cray XE6 test system "hera"



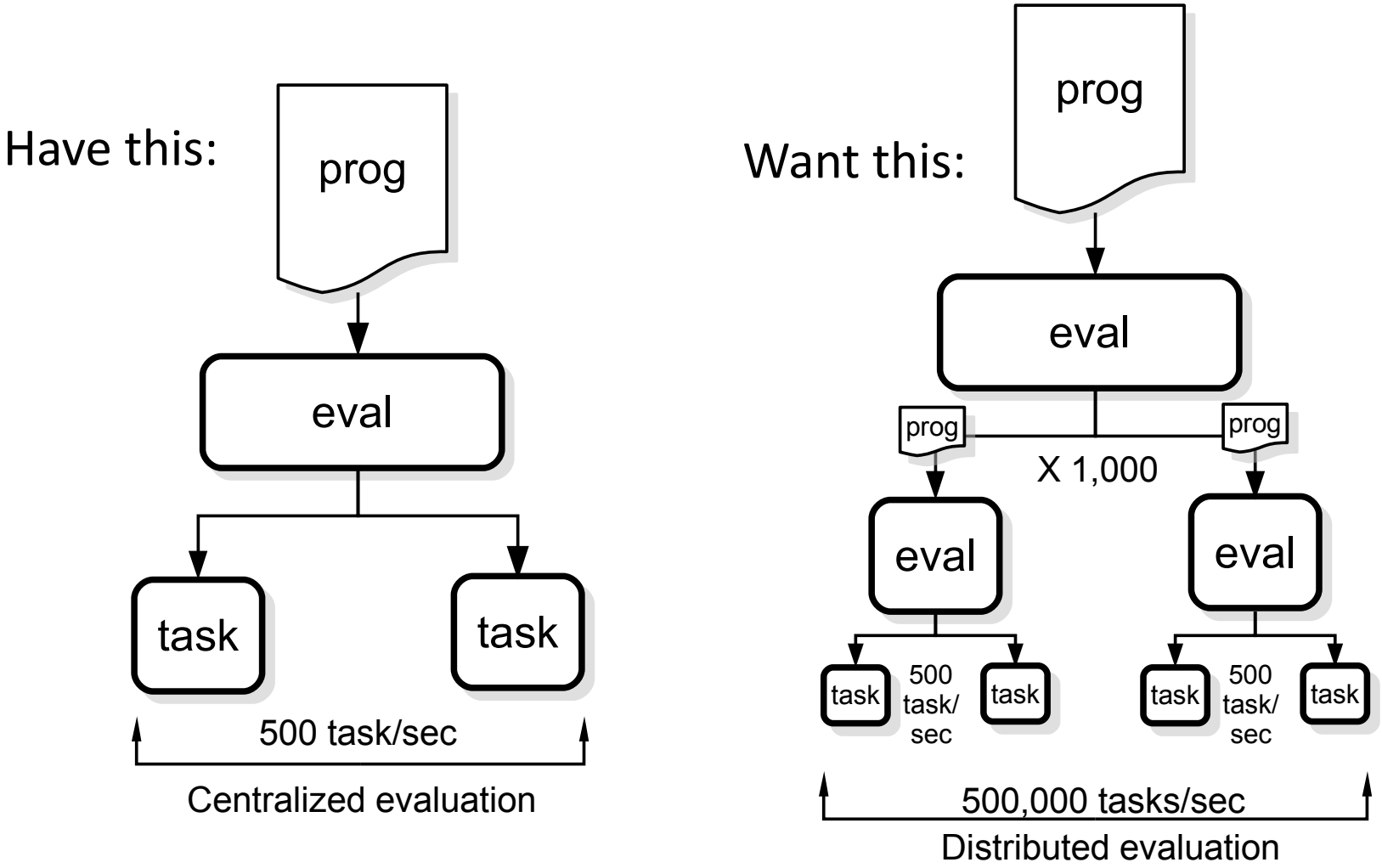
Performance study for DSSAT job profile

Synthetic test of DSSAT application workload, 48,000 200 sec tasks, 16,000 cores of Cray XK6 Hera, 32 cores/node (2x IL-16) .

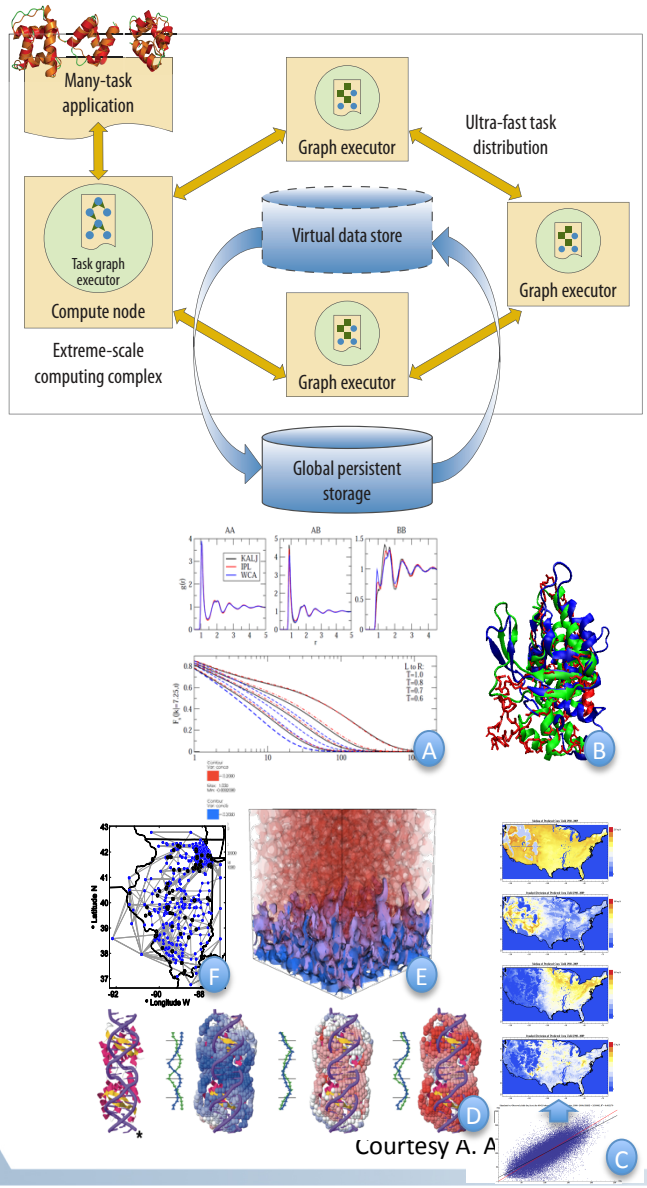
Swift efficiency on Cray XE6 test system "hera"



Centralized evaluation can be a bottleneck



ExM: extreme-scaling for many-task computing



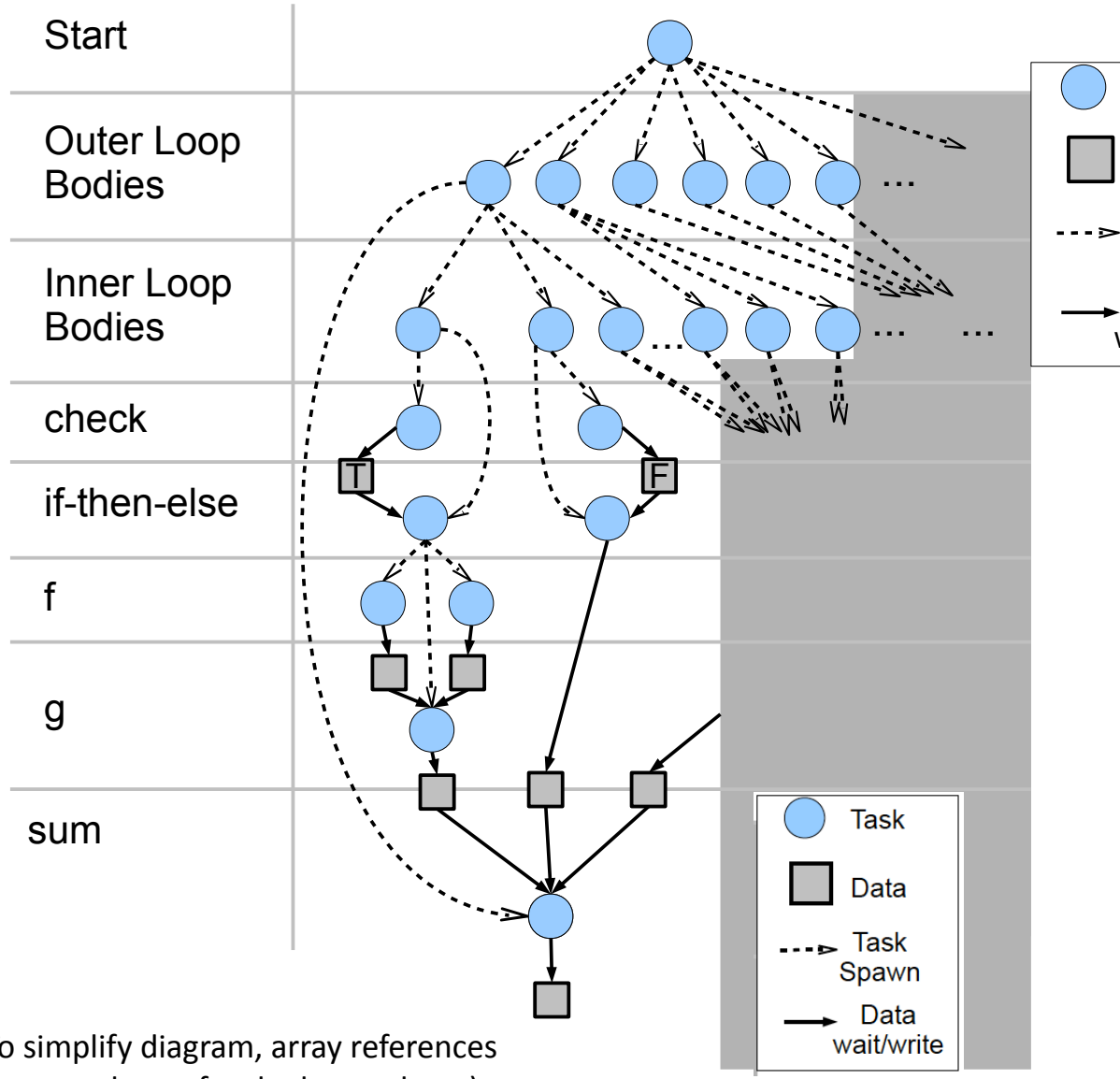
- Implicitly-parallel functional dataflow programming for upper-level application logic
- Drivers: inverse problems, branch-and-bound, stochastic programming, UQ, ensembles
- Enablers: scalable parallel evaluation, dynamic load balancing, in-RAM datasets
- Benefits: programmability, fault-recovery;
  possibly, power savings
- Results: new scalable Swift implementation, 25K tasks/sec, 128K-core parallel loop scaling; datastore and MTC publications

Goal: programmability for extreme scale

- Focus is “many-task” computing: higher-level applications composed of many run-to-completion tasks (input-process-output vs. message passing)
- Why is it relevant to DOE extreme-scale computing?
  - Programmability
    - Increasing number of applications have this natural structure: material by design, inverse problems, stochastic programming, branch-and-bound problems, UQ.
    - Coupling extreme-scale applications to preprocessing, analysis, and visualization
  - Resilience
    - The functional programming model provides a modular hierarchy for re-execution of failing units of an application
  - Power management
    - Yet to prove: Graph structure of application upper levels may enable functional units to be quiesced.
- Challenges
  - Data locality and load balancing!

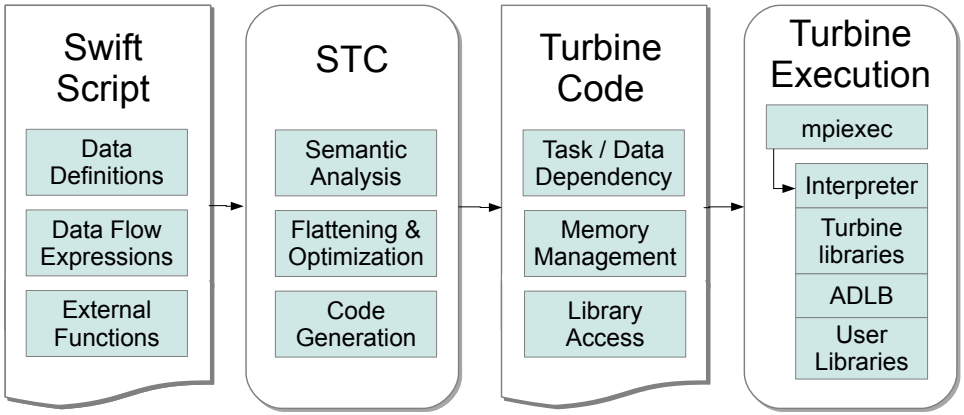
Fully parallel evaluation of complex scripts

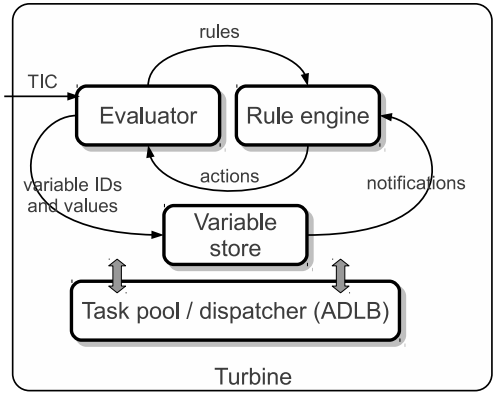
```
int X = 100, Y = 100;
int A[][];
int B[];
foreach x in [0:X-1] {
  foreach y in [0:Y-1] {
    if (check(x, y)) {
      A[x][y] = g(f(x), f(y));
    } else {
      A[x][y] = 0;
    }
  }
  B[x] = sum(A[x]);
}
```

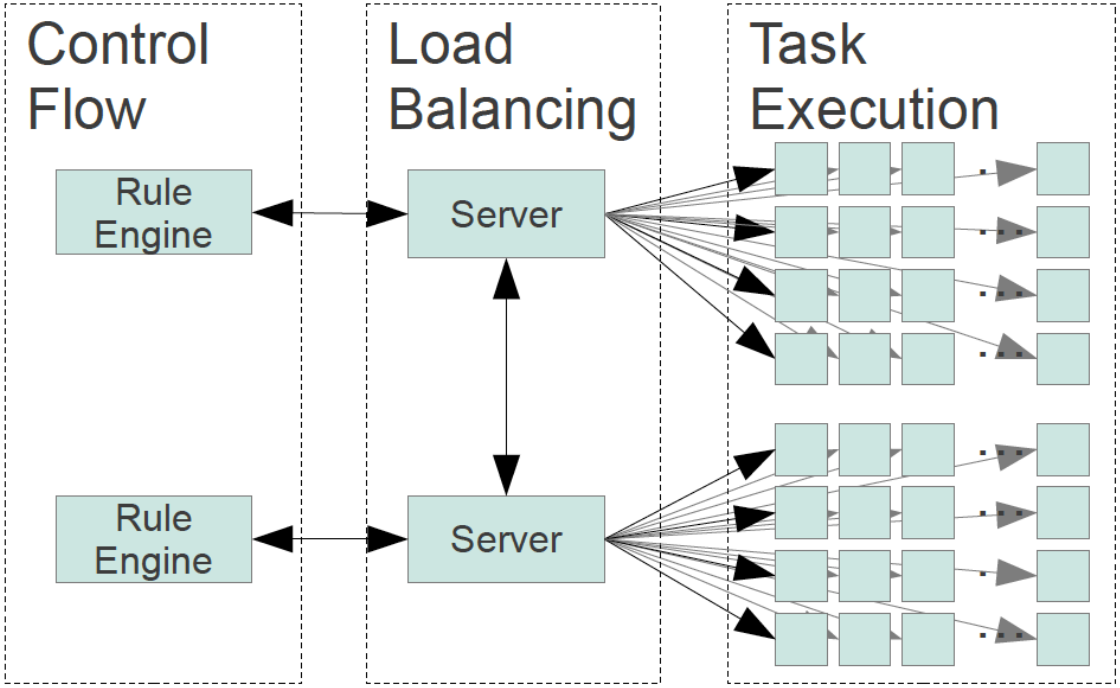


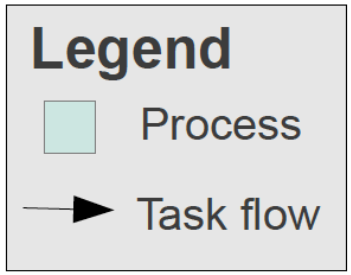
(To simplify diagram, array references are not shown for the loops above)

Parallel evaluation of Swift/T in ExM





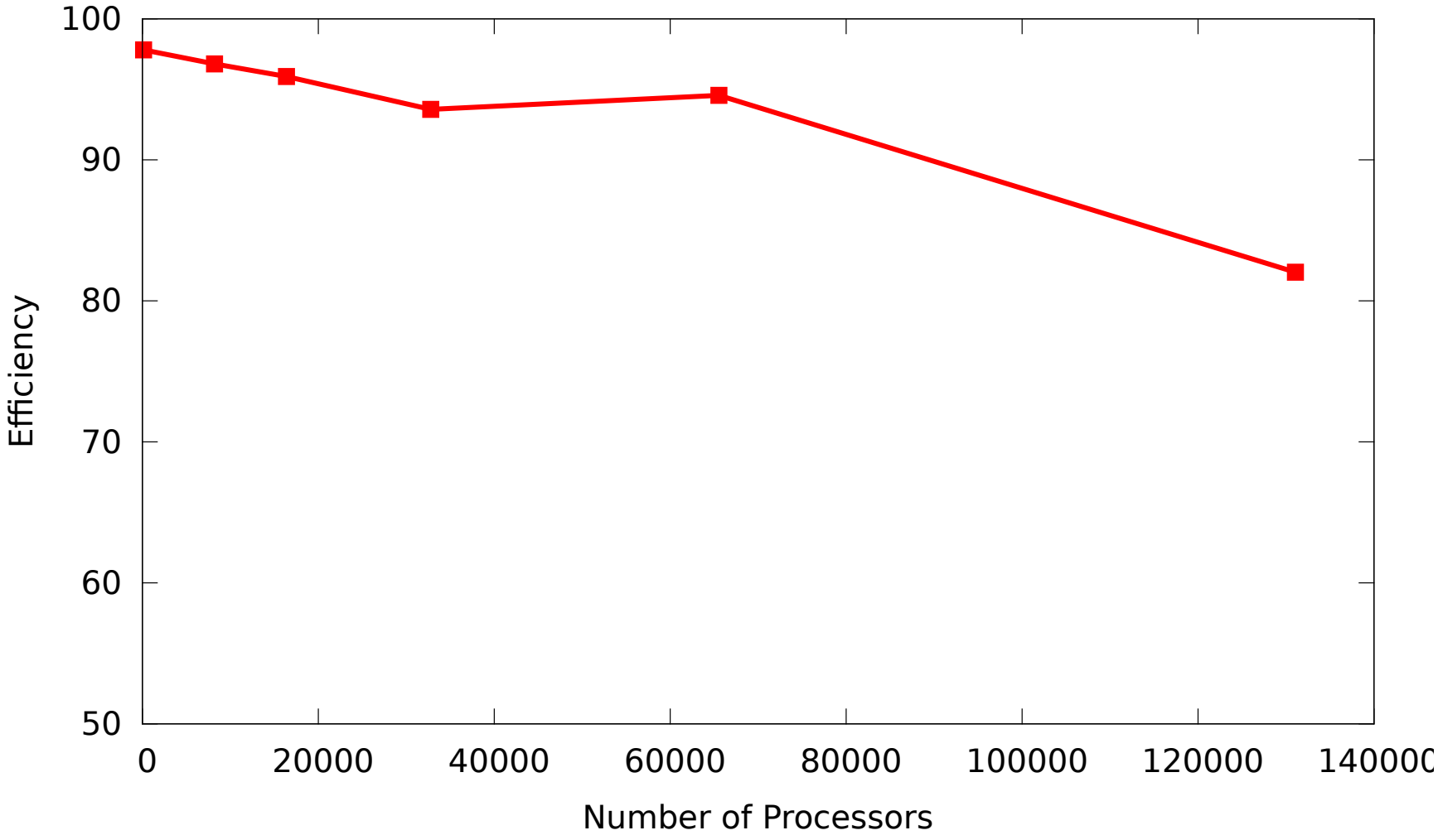




What are the challenges in applying the many-task model at extreme scale?

- Dealing with high task dispatch rates: depends on task granularity but is commonly an obstacle
  - Load balancing
  - Global and scoped access to script variables: increasing locality
- Handling tasks that are themselves parallel programs
  - We focus on OpenMP, MPI, and Global Arrays
- Data management for intermediate results
  - Object based abstractions
  - POSIX-based abstractions
  - Interaction between data management and task placement

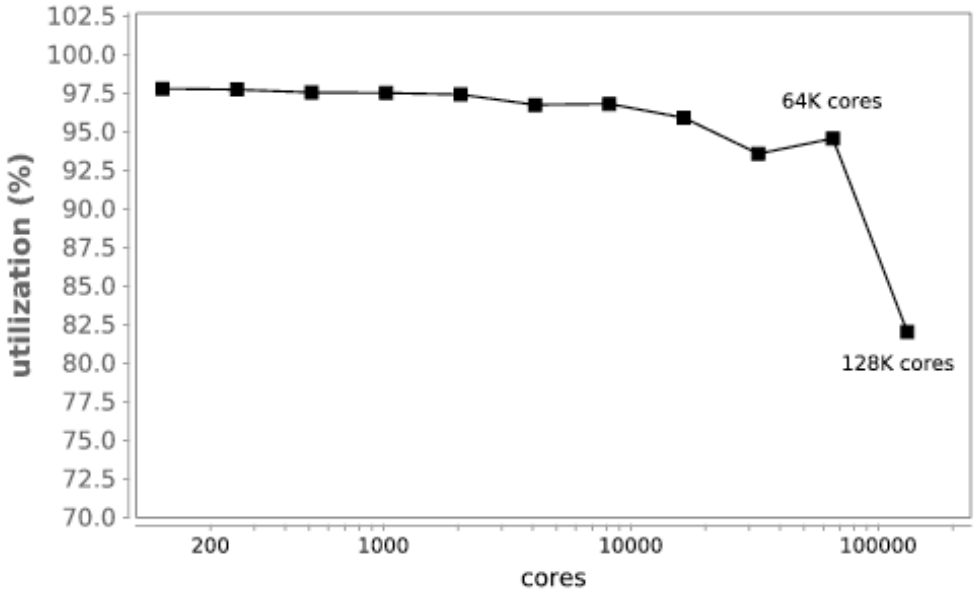
Swift-ExM efficiency - to 128K cores



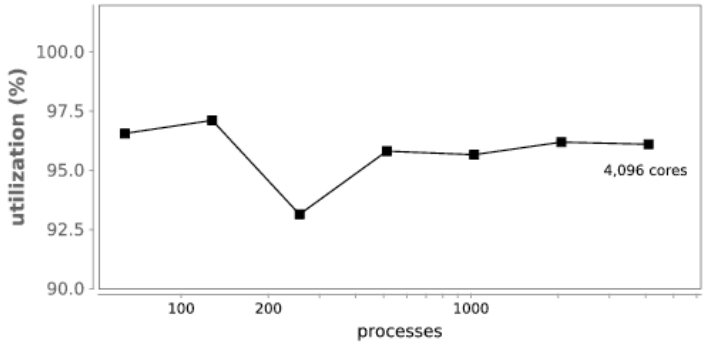
Prototype Swift-ExM on BG/P Intrepid, 32,768 nodes, 131,072 cores.
100 second tasks (processes = #cores)

Swift/T: Scaling performance

- Swift/T synthetic app: simple task loop
  - Scales to capacity of ADLB

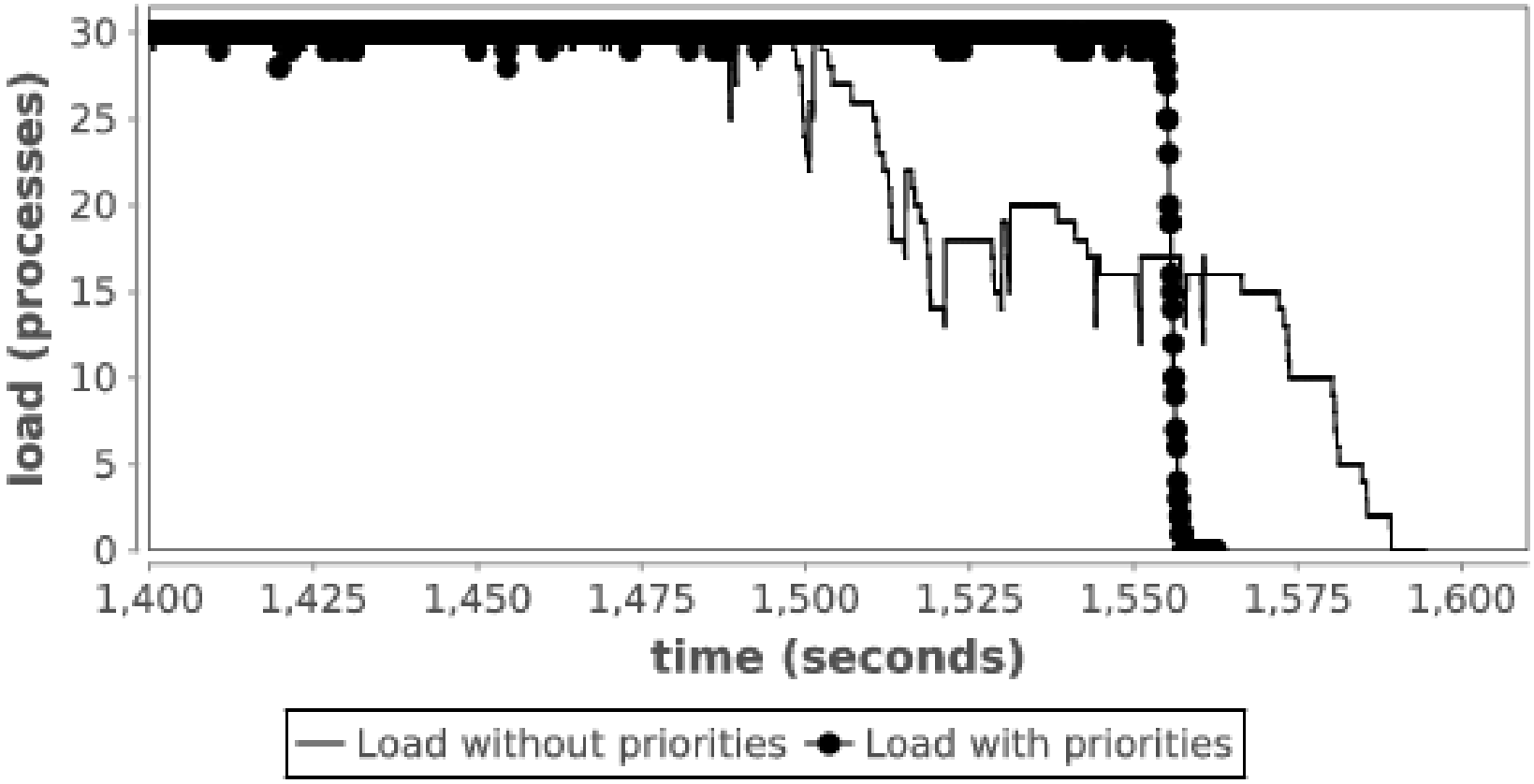


- SciSolSim: collaboration graph modeling
  - ~400 lines Swift
  - C++ graph model; simulated annealing
  - Scales well to 4K cores in initial tests (further tests awaiting app debugging)



Swift/T: performance optimizations

- Variable-sized tasks produce trailing tasks:
  addressed by exposing task priorities at language level



Related work

- Runtime systems
  - ParalleX
  - Swarm
- Libraries
  - DAGuE, FLAME
  - TBB, CnC
  - Map-Reduce
- Languages
  - Cilk, Dryad
  - Habanero languages
- Other load balancers
  - Sciotto, TASCEL
- Early foundational work
  - Futures (LISP), Strand, PCN, J Machine, Linda (coordination language), Sisal (early dataflow language), Fresh Breeze

Plans and opportunities

- Mosastore-Swift/T integration at large scale
- Integration of AME heuristics into Swift/T and Mosastore
- Fault tolerance design and evaluation
- Structured datasets in RAM: HDF5, NetCDT
- Vendors: Support Cray customers' evaluation of Swift
- Unified Swift implementation based on Swift/T (multi-HPC, cloud)
- Large-scale evaluation of MosaStore/Swift integration
- Power awareness and optimization: locality of computation; tail tuning; data locality improvement

Vision and next steps

- *Broad, community-wide adoption and support*
- *Refine into a highly usable, well documented package*
- *Promote through multiple channels: web, collaborations, endorsements*
- *Language bindings to increase user adoption*
- *Unified simpler implementation through ExM deliverables*
- *Integration into GO and other portals (Galaxy, GPSI, …)*
- *DPI – a universal interface for file-based and memory-based data passing?*
- *Make the community self-sustaining*
- *Shift more focus to the computer science and deeper (and perhaps larger) engagements*

Computer science topics and challenges

- *Programming model:* implicitly parallel functional dataflow
  - *Hybrid programming models: Swift over MPI, OpenMP, PGAS*
  - *Applicability to Big Data processing models: MapReduce, Dremel, Sawzall, …*
- *Diverse language bindings for the programming model*
  - *Libraries, reflection, and source-to-source translation (e.g., ROSE)*
- *Performance and scaling*
  - *Load balancing – working with ADLB*
  - *Targeting Mira, Blue Waters, beyond*
- *Data management and scheduling*
  - *Accurate models and intelligent routing/placement of jobs and data*
- *Data transport: high performance parallel streams and wide pipes*
  - *within and between large-scale systems*
- *Data models*
  - *Fresh Breeze concept: new memory model, massive multithreading (XMT)*
- *Energy and resilience*
  - *The Swift programming model offers a wealth of opportunities for research in these fields.*
- *Provenance – expression, representation, query and usability*
- *User interface design for parallel distributed programming environments*

- Swift is a parallel scripting system for grids, clouds 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: uses efficient, scalable and flexible "Karajan" 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.
- Try Swift! www.ci.uchicago.edu/swift and www.mcs.anl.gov/exm

Parallel Computing 37 (2011) 633–652

Contents lists available at ScienceDirect

Parallel Computing

journal homepage: www.elsevier.com/locate/parco

Swift: A language for distributed parallel scripting

Michael Wilde [a,b,*], Mihael Hategan [a], Justin M. Wozniak [b], Ben Clifford [d], Daniel S. Katz [a], Ian Foster [a,b,c]

[a] Computation Institute, University of Chicago and Argonne National Laboratory, United States
[b] Mathematics and Computer Science Division, Argonne National Laboratory, United States
[c] Department of Computer Science, University of Chicago, United States
[d] Department of Astronomy and Astrophysics, University of Chicago, United States

ARTICLE INFO

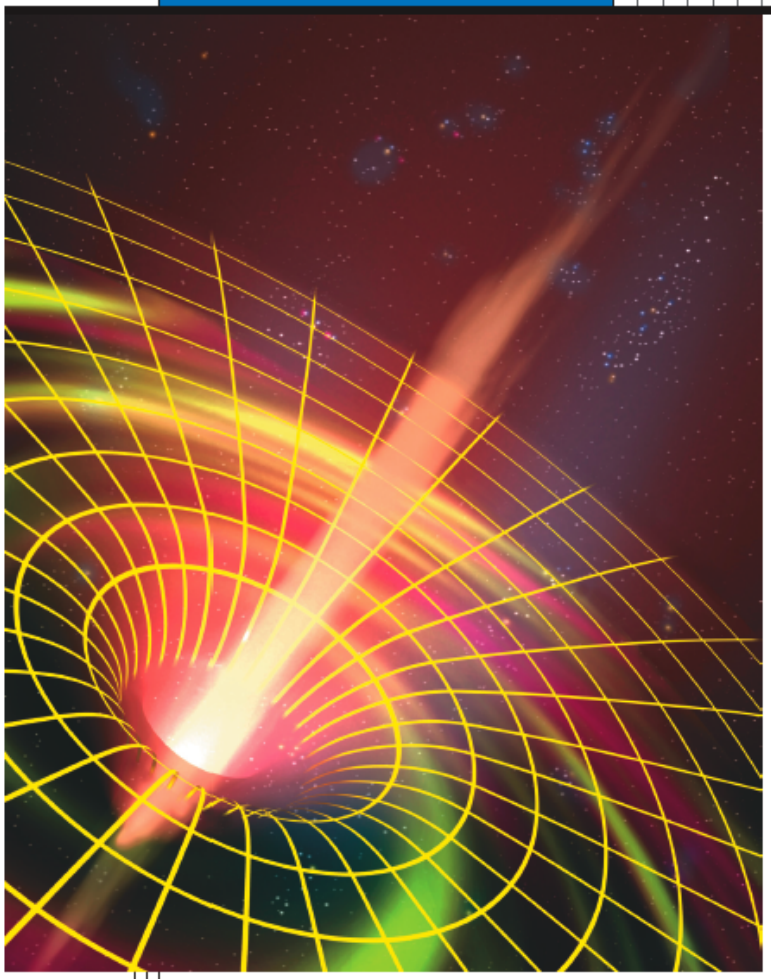
Article history:
Available online 12 July 2011

Keywords:
Swift
Parallel programming
Scripting
Dataflow

ABSTRACT

Scientists, engineers, and statisticians must execute domain-specific application programs many times on large collections of file-based data. This activity requires complex orchestration and data management as data is passed to, from, and among application invocations. Distributed and parallel computing resources can accelerate such processing, but their use further increases programming complexity. The Swift parallel scripting language reduces these complexities by making file system structures accessible via language constructs and by allowing ordinary application programs to be composed into powerful parallel scripts that can efficiently utilize parallel and distributed resources. We present Swift's implicitly parallel and deterministic programming model, which applies external applications to file collections using a functional style that abstracts and simplifies distributed parallel execution.

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*

Acknowledgments

- Swift is supported in part by NSF grants OCI-1148443 and PHY-636265, NIH DC08638, and the UChicago SCI Program
- ExM is supported by the DOE Office of Science, ASCR Division
- Structure prediction supported in part by NIH
- The Swift team:
  - Mihael Hategan, Justin Wozniak, Ketan Maheshwari, Ben Clifford, David Kelly, Jon Monette, Allan Espinosa, Ian Foster, Dan Katz, Ioan Raicu, Sarah Kenny, Mike Wilde, Zhao Zhang, Yong Zhao
- GPSI Science portal:
  - Mark Hereld, Tom Uram, Wenjun Wu, Mike Papka
- Java CoG Kit used by Swift developed by:
  - Mihael Hategan, Gregor Von Laszewski, and many collaborators
- ZeptoOS
  - Kamil Iskra, Kazutomo Yoshii, and Pete Beckman
- Scientific application collaborators and usage described in this talk:
  - 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
  - ParVis and FOAM: Rob Jacob, Sheri Mickelson (Argonne); John Dennis, Matthew Woitaszek (NCAR)
  - PTMap: Yingming Zhao, Yue Chen