Swift Parallel Scripting: Workflows for Simulations and Data Analytics at Extreme Scale

Michael Wilde

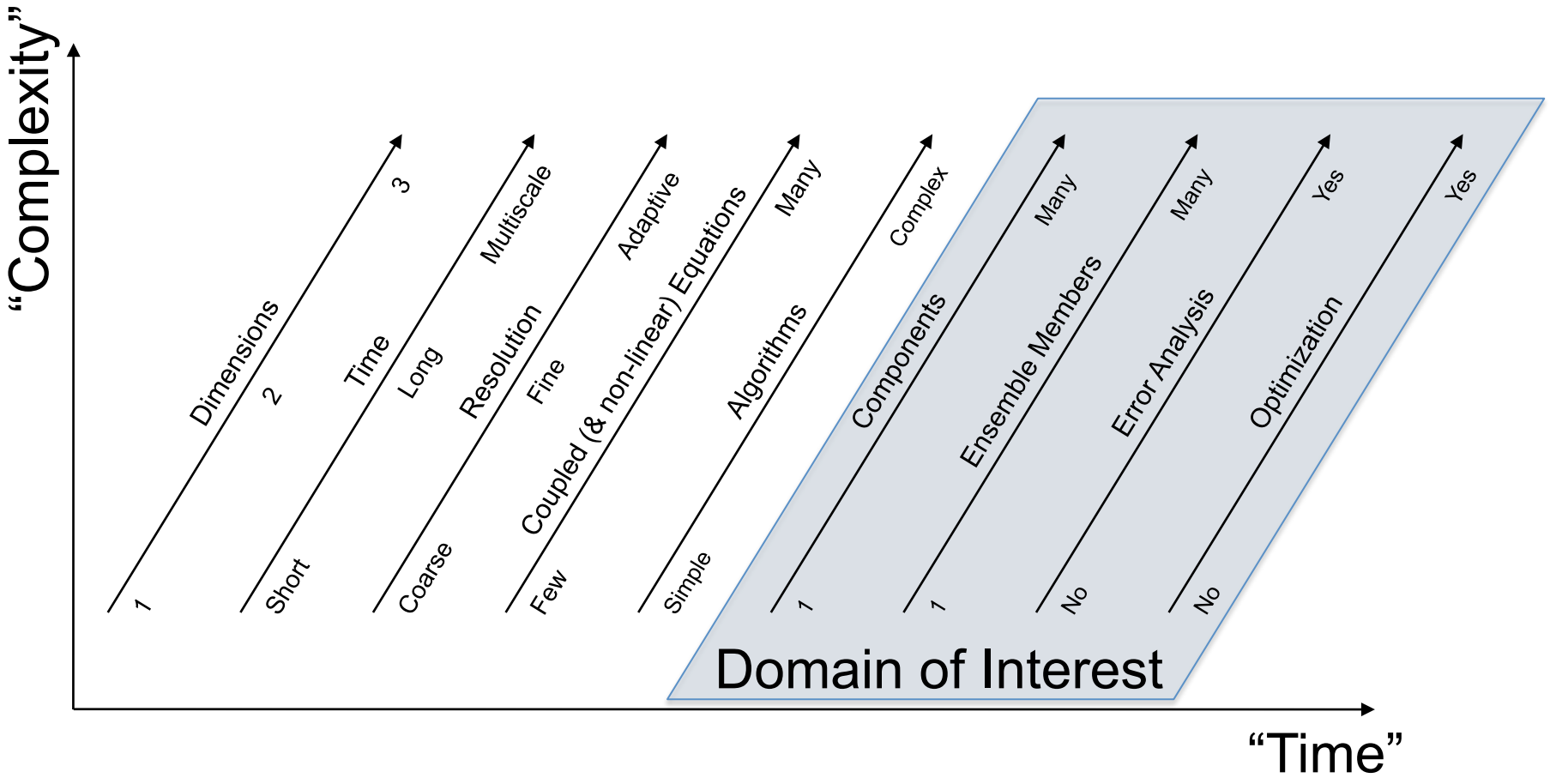
wilde@anl.gov

http://swift-lang.org

The Swift Team

- Timothy Armstrong, Yadu Nand Babuji, Ian Foster, Mihael Hategan, Daniel S. Katz, Ketan Maheshwari, Michael Wilde, Justin Wozniak, Yangxinye Yang
- 2015 REU Summer Collaborators: Jonathan Burge, Mermer Dupres, Basheer Subei, Jacob Taylor
- Contributions by Ben Clifford, Luiz Gadelha, Yong Zhao, Scott Krieder, Ioan Raicu, Tiberius Stef-Praun, Matthew Shaxted
- Our sincere thanks to the entire Swift user community

Increasing capabilities in computational science



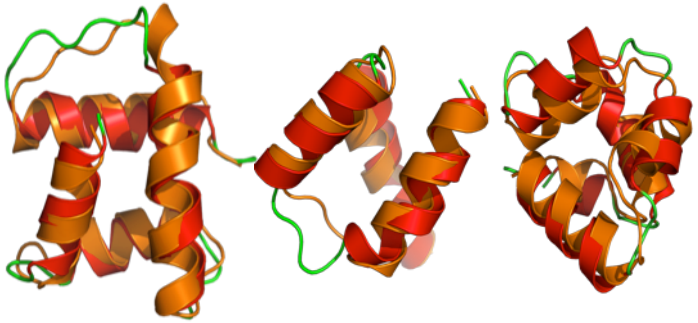
Workflow needs

- Application Drivers
  - Applications that are many-task in nature: parameters sweeps, UQ, inverse modeling, *and data-driven applications*
  - Analysis of capability application outputs
  - Analysis of stored or collected data
  - Increase productivity at major research instrumentation
  - Urgent computing
  - These applications are all *many-task* in nature
- Requirements
  - Usability and ease of workflow expression
  - Ability to leverage complex architecture of HPC and HTC systems (fabric, scheduler, hybrid node and programming models), individually and collectively
  - Ability to integrate high-performance data services and volumes
  - Make use of the system task rate capabilities from clusters to extreme-scale systems
- Approach
  - A programming model for *programming in the large*

When do you need HPC workflow?

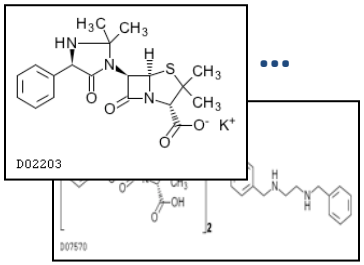
Example application: protein-ligand docking for drug screening

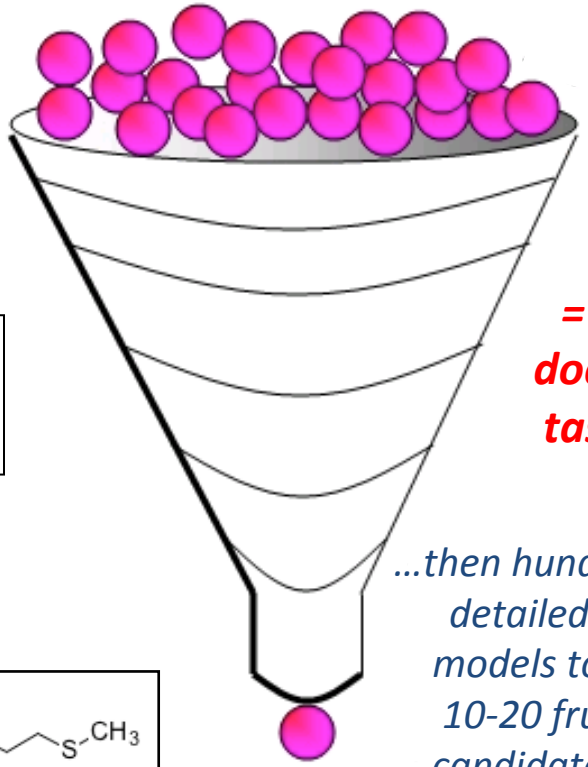
O(10) proteins implicated in a disease



X

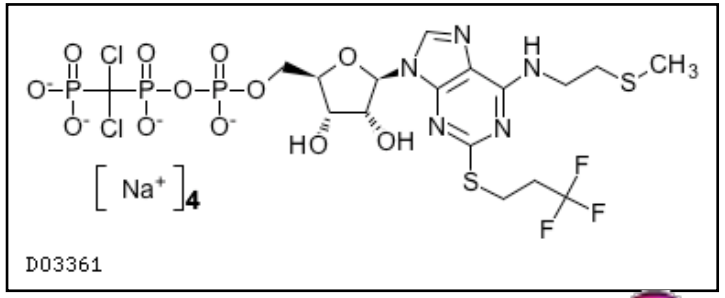
O(100K) drug candidates





= 1M docking tasks…

…then hundreds of detailed MD models to find 10-20 fruitful candidates for wetlab & APS crystallography



Expressing this many task workflow in Swift

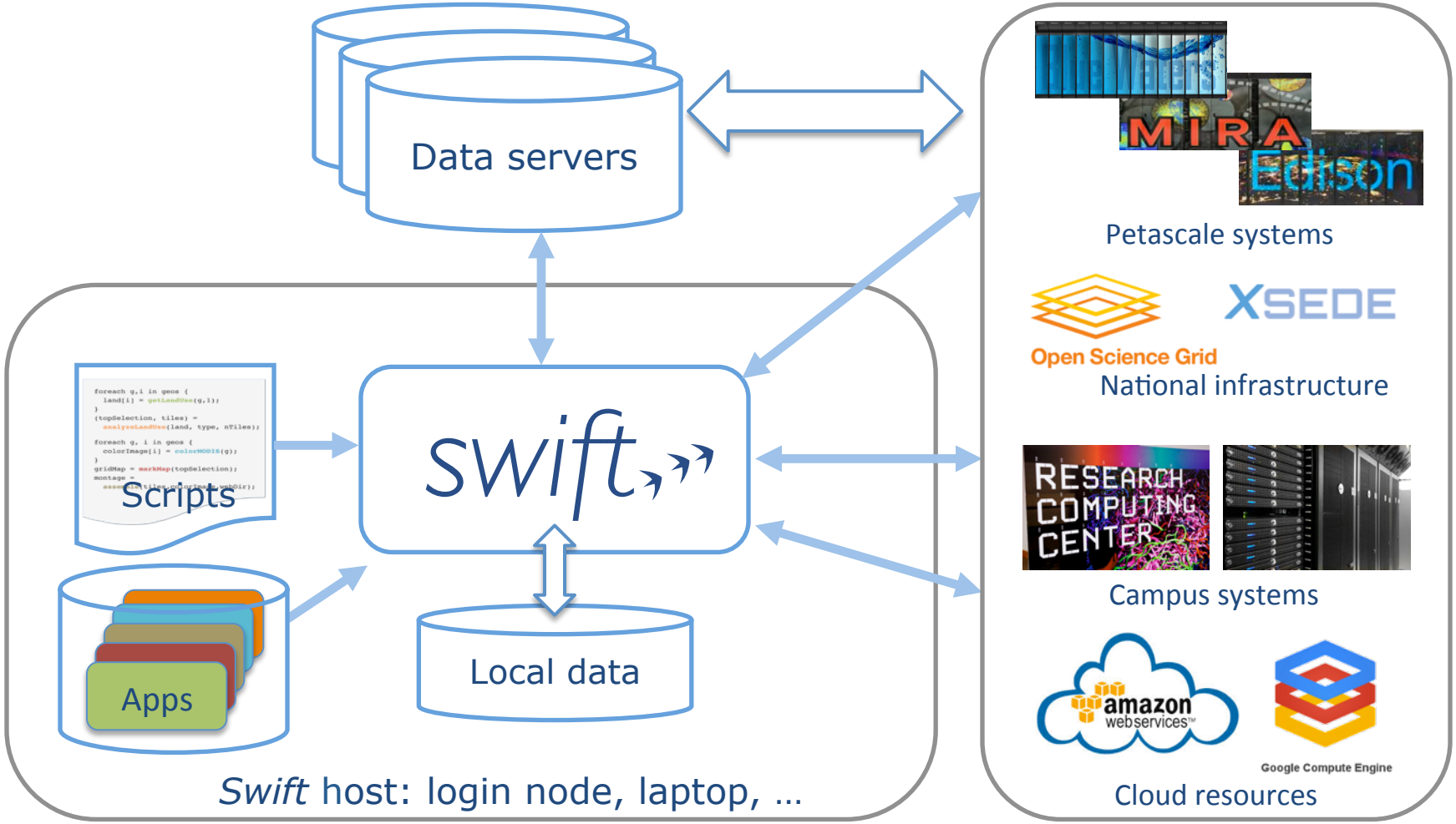
For protein docking workflow:

```
foreach p, i in proteins {
   foreach c, j in ligands {
      (structure[i,j], log[i,j]) =
       dock(p, c, minRad, maxRad);
}
scatter_plot = analyze(structure)
```

To run:

```
swift –site tukey,blues dock.swift
```

Swift enables execution of simulation campaigns across multiple HPC and cloud resources



The Swift runtime system has drivers and algorithms to efficiently support and aggregate diverse runtime environments

Comparison of parallel programming models

- MPI
  - Each process has a persistent state for the life of the application
  - Data is exchanged via messages
  - Program an app by “domain decomposition”
- Shared Memory
  - Tasks need not be persistent
  - Tasks initiated explicitly (eg pthreads) or implicitly (eg by OpenMP directives)
  - Challenge is to assure mutual exclusion and avoid deadlock
  - Limited to running within a single node, or use hybrid approach for multi-node
- Workflow
  - Tasks run to completion
  - Data is exchanged on task initiation and completion
  - Used to coordinate execution of multiple serial *or* parallel apps
- Hybrid
  - MPI and/or OpenMPI apps within a many-task workflow

Swift in a nutshell

- Data types

```
string s = "hello world";
int i = 4;
int A[];
```

- Mapped data types

```
type image;
image file1<"snapshot.jpg">;
```

- Mapped functions

```
app (file o) myapp(file f, int i)
{ mysim  "-s" i @f @o; }
```

- Conventional expressions

```
if (x == 3) {
    y = x+2;
    s = @strcat("y: ", y);
}
```

- Structured data

```
image  A[]<array_mapper…>;
```

- Loops

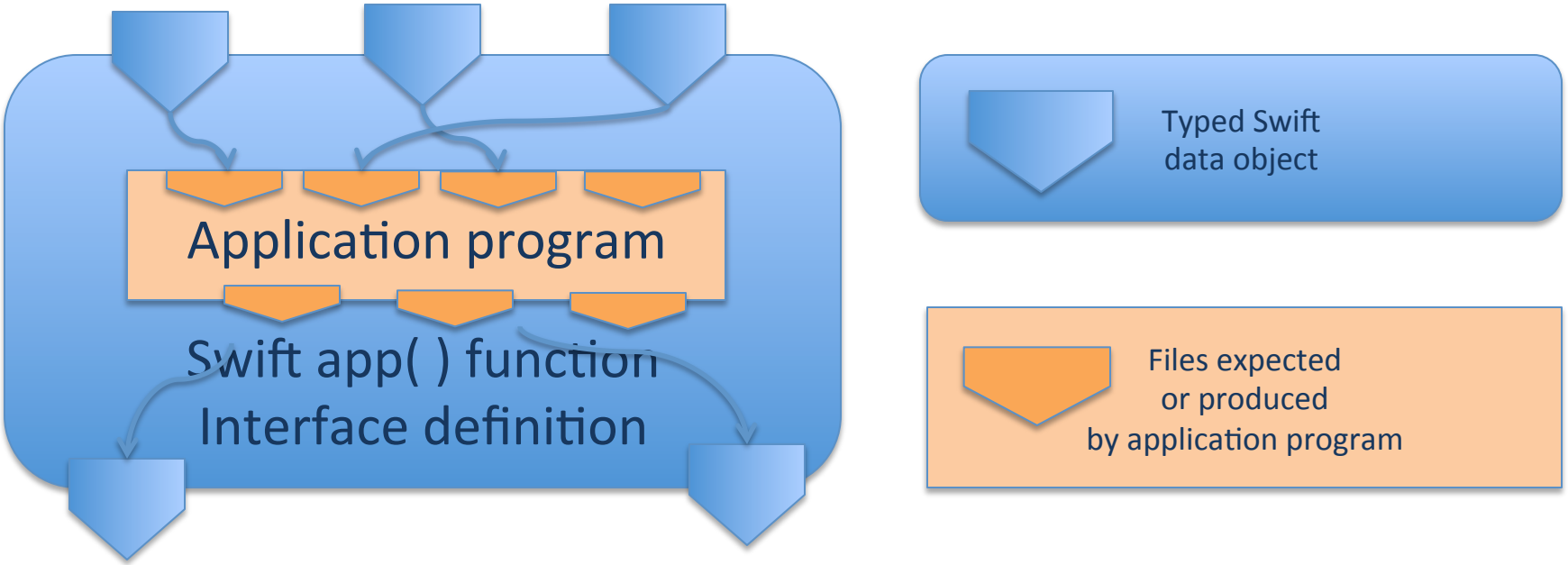
```
foreach f,i in A {
    B[i] = convert(A[i]);
}
```

- Data flow

```
analyze(B[0], B[1]);
analyze(B[2], B[3]);
```

Swift: A language for distributed parallel scripting, J. Parallel Computing, 2011

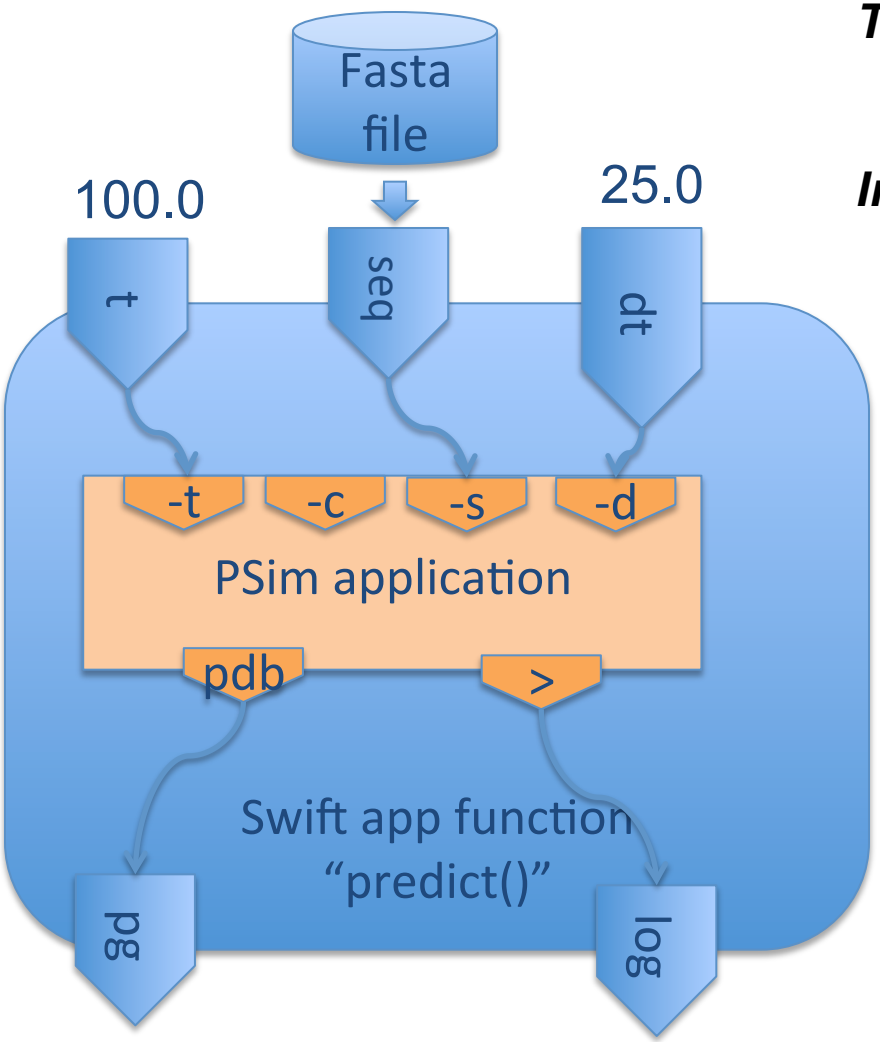
Encapsulation enables distributed parallelism



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

Critical in a world of scientific, engineering, technical and analytical applications

app( ) functions specify command line arg passing



To run:

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

In Swift code:

```
app (PDB pg, Text 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;
Text log;
```

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

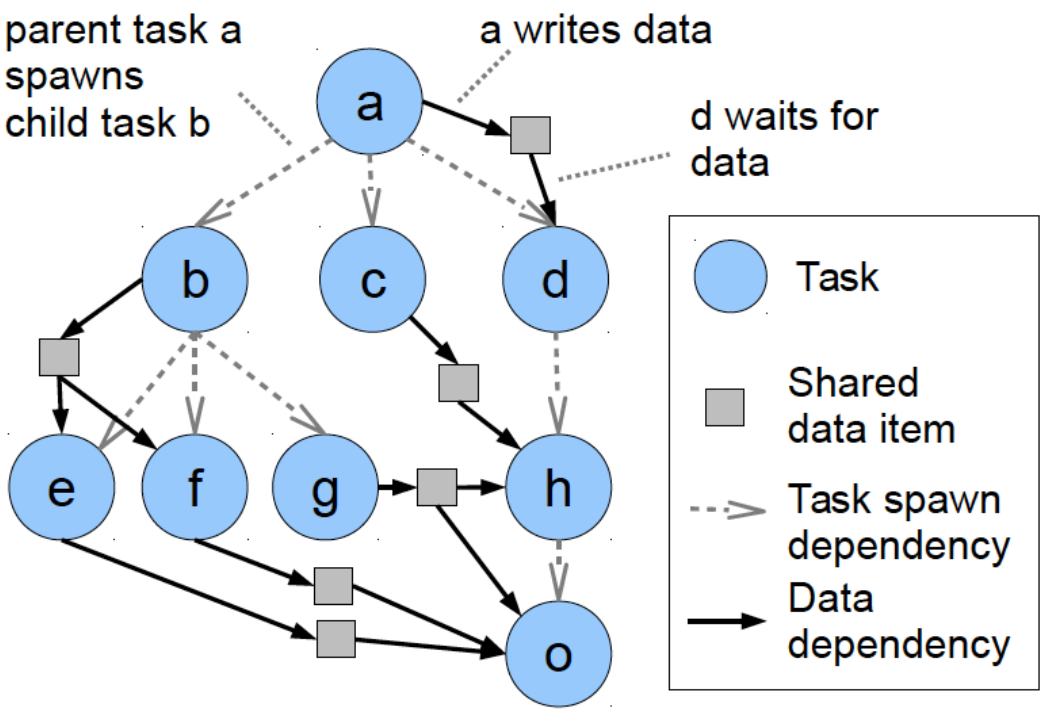
Implicitly parallel

- Swift is an implicitly parallel functional programming language for clusters, grids, clouds and supercomputers
- All expressions evaluate when their data inputs are "ready"

```
(int r) myproc (int i)
{
    int f = F(i);
    int g = G(i);
    r = f + g;
}
```

- F() and G() are computed in parallel
  - Can be Swift functions, or leaf tasks (executables or scripts in shell, python, R, Octave, MATLAB, ...)
- r computed when they are done
- This parallelism is *automatic*
- Works recursively throughout the program's call graph

Pervasive parallel data flow



Functional composition in *Swift* – *enables powerful parallel loops*

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

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

Data-intensive example: Processing MODIS land-use data

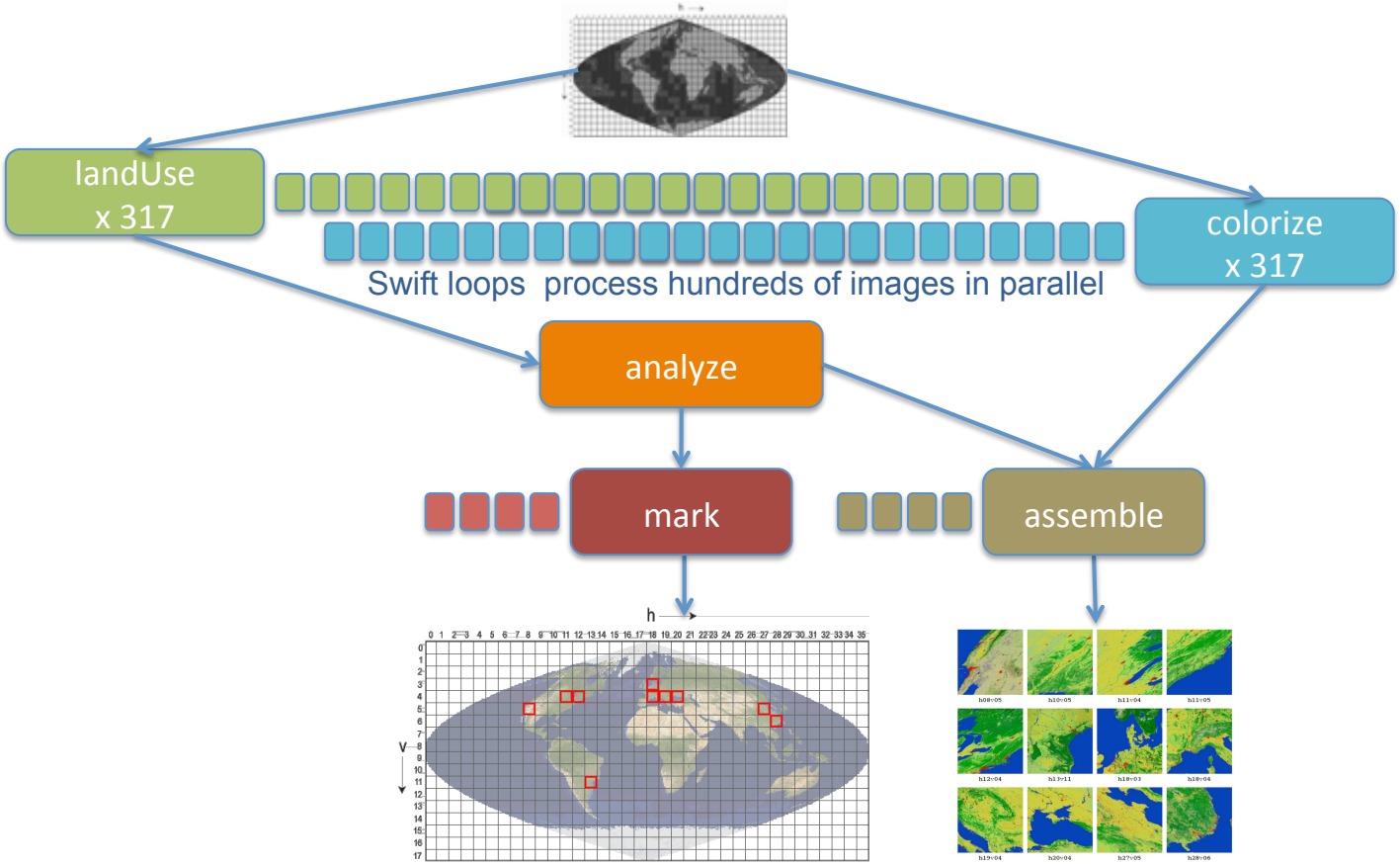


Image processing pipeline for land-use data from the MODIS satellite instrument…

Processing MODIS land-use data

```
foreach raw,i in rawFiles {
    land[i] = landUse(raw,1);
    colorFiles[i] = colorize(raw);
}
(topTiles, topFiles, topColors) =
    analyze(land, landType, nSelect);

gridMap = mark(topTiles);
montage =
  assemble(topFiles,colorFiles,webDir);
```

Example of Swift's implicit parallelism: Processing MODIS land-use data

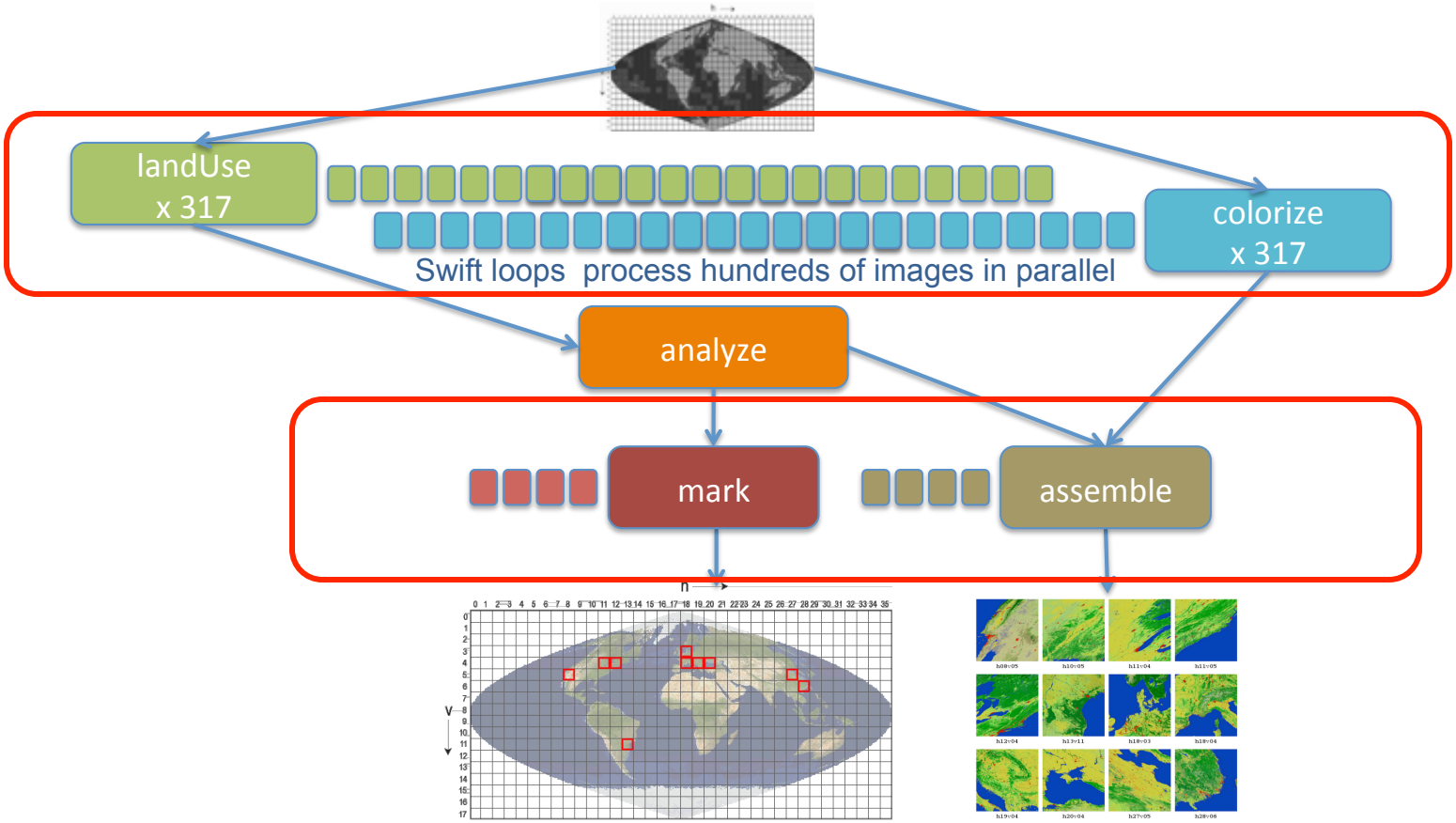
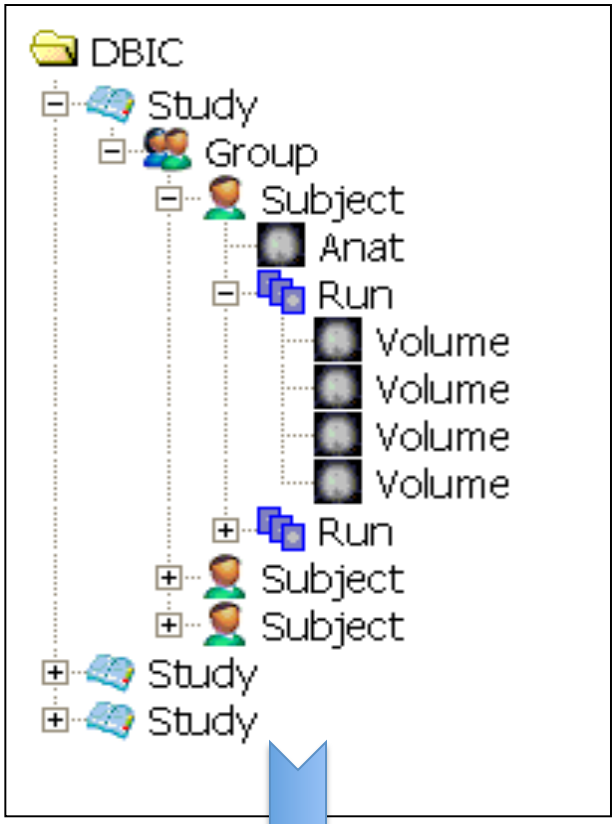


Image processing pipeline for land-use data from the MODIS satellite instrument…

Dataset mapping example: deep fMRI directory tree

On-Disk Data Layout

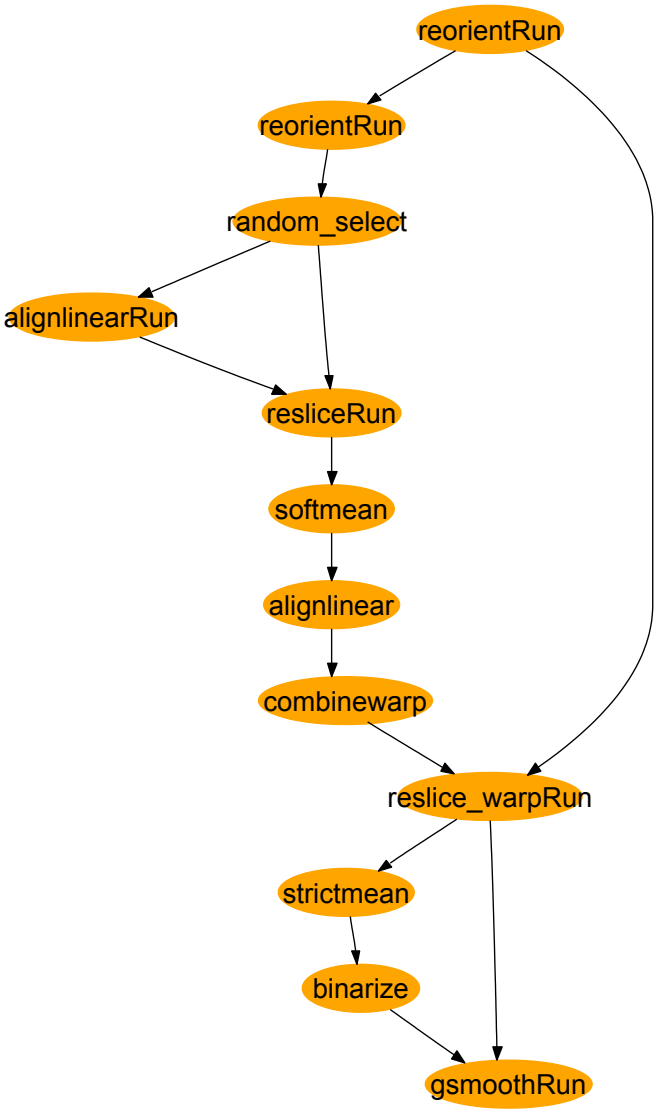


Mapping function or script

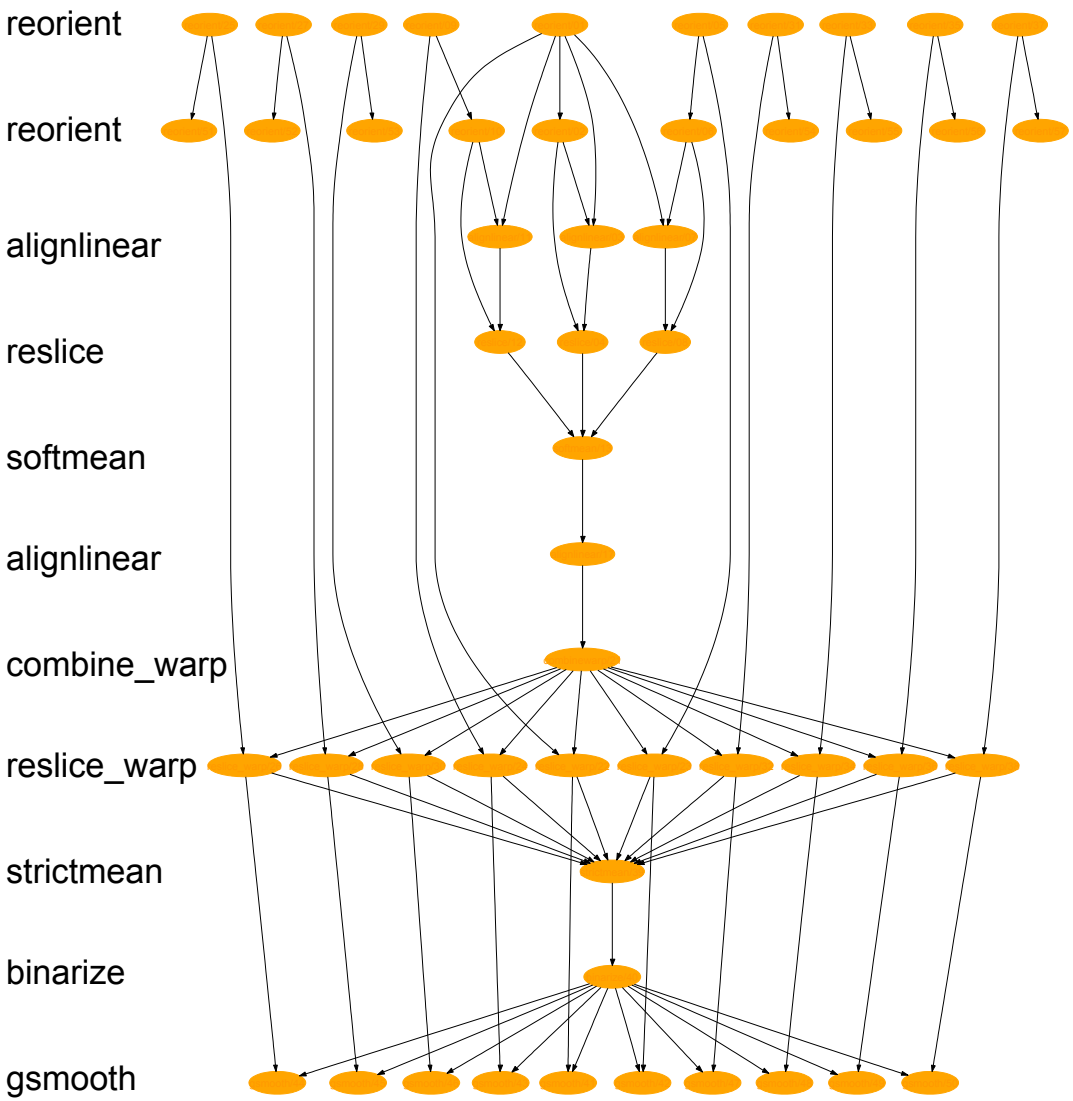
```
type Study {
        Group g[ ];
}
type Group {
        Subject s[ ];
}
type Subject {
        Volume anat;
        Run run[ ];
}
type Run {
        Volume v[ ];
}
type Volume {
        Image img;
        Header hdr;
}
```

Swift's in-memory data model

Spatial normalization of functional MRI runs



Dataset-level workflow



Expanded (10 volume) workflow

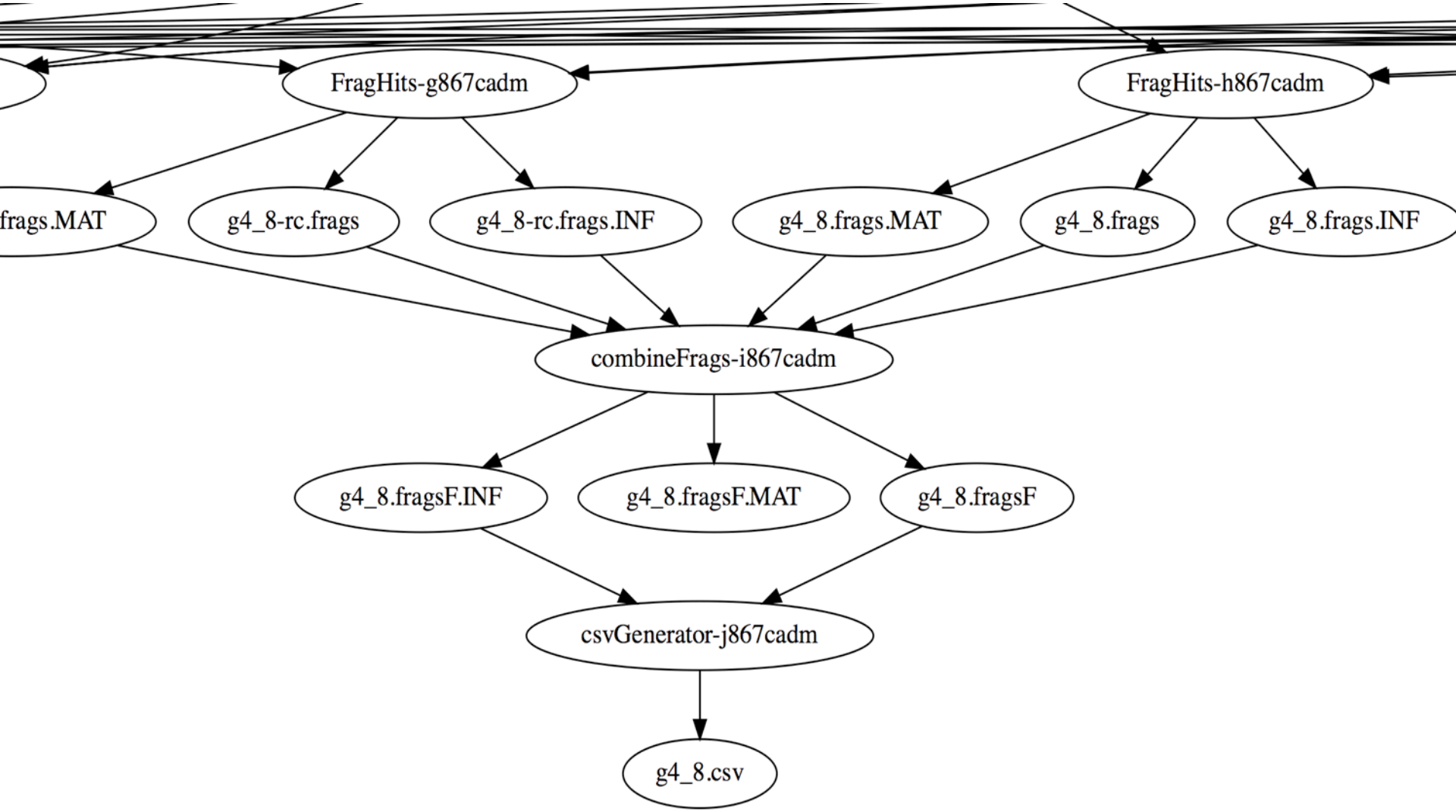
Complex scripts can be well-structured

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);
        }
}
```

Provenance graph from Swift log



Work of Luiz Gadelha and Maria Luiza Mondelli, LNCC

Provenance data from log into SQL DB

	app_exec_id	real_secs	percent_cpu	max_rss	fs_writes
1	allvsall-run011-2683821486:w2hd-t667cadm	147.96	94	1984	542968
2	allvsall-run011-2683821486:w2hd-s667cadm	145.48	96	1984	542968
3	allvsall-run011-2683821486:w2hd-n667cadm	127.52	99	1984	359376
4	allvsall-run011-2683821486:w2hd-r667cadm	127.09	99	1984	367880
5	allvsall-run011-2683821486:w2hd-o667cadm	126.13	99	1984	367880
6	allvsall-run011-2683821486:w2hd-m667cadm	126.04	99	1984	367784
7	allvsall-run011-2683821486:w2hd-q667cadm	125.82	99	1968	367792
8	allvsall-run011-2683821486:w2hd-p667cadm	123.11	99	1984	359376
9	allvsall-run011-2683821486:w2hd-j667cadm	114.17	98	1984	301560
10	allvsall-run011-2683821486:w2hd-l667cadm	113.0	99	1984	301560
11	allvsall-run011-2683821486:w2hd-i667cadm	100.34	99	2000	233912
12	allvsall-run011-2683821486:w2hd-g667cadm	99.82	98	1984	233912
13	allvsall-run011-2683821486:w2hd-e667cadm	83.14	98	1968	175488
14	allvsall-run011-2683821486:w2hd-8667cadm	78.87	98	1968	175480
15	allvsall-run011-2683821486:w2hd-7667cadm	76.56	98	1984	161960
16	allvsall-run011-2683821486:w2hd-a667cadm	75.2	98	1984	161960
17	allvsall-run011-2683821486:w2hd-4667cadm	74.29	98	1984	153936
18	allvsall-run011-2683821486:w2hd-d667cadm	74.03	98	1984	153936
19	allvsall-run011-2683821486:hits-ra67cadm	36.72	69	1564624	777904
20	allvsall-run011-2683821486:hits-ec67cadm	33.54	82	1564640	1000608
21	allvsall-run011-2683821486:hits-rc67cadm	32.71	74	1564640	1085016

Work of Luiz Gadelha and Maria Luiza Mondelli, LNCC

Use SQL to mine insight from provenance

```
1 select app__name, real_secs, fs_writes
2 from app_exec natural join resource_usage;
```

Total rows loaded: 520

	app__name	real_secs	fs_writes
1	sortHits	15.08	2054096
2	sortHits	12.99	1995280
3	sortHits	10.07	1856472
4	sortHits	10.41	1796360
5	hits	32.71	1085016

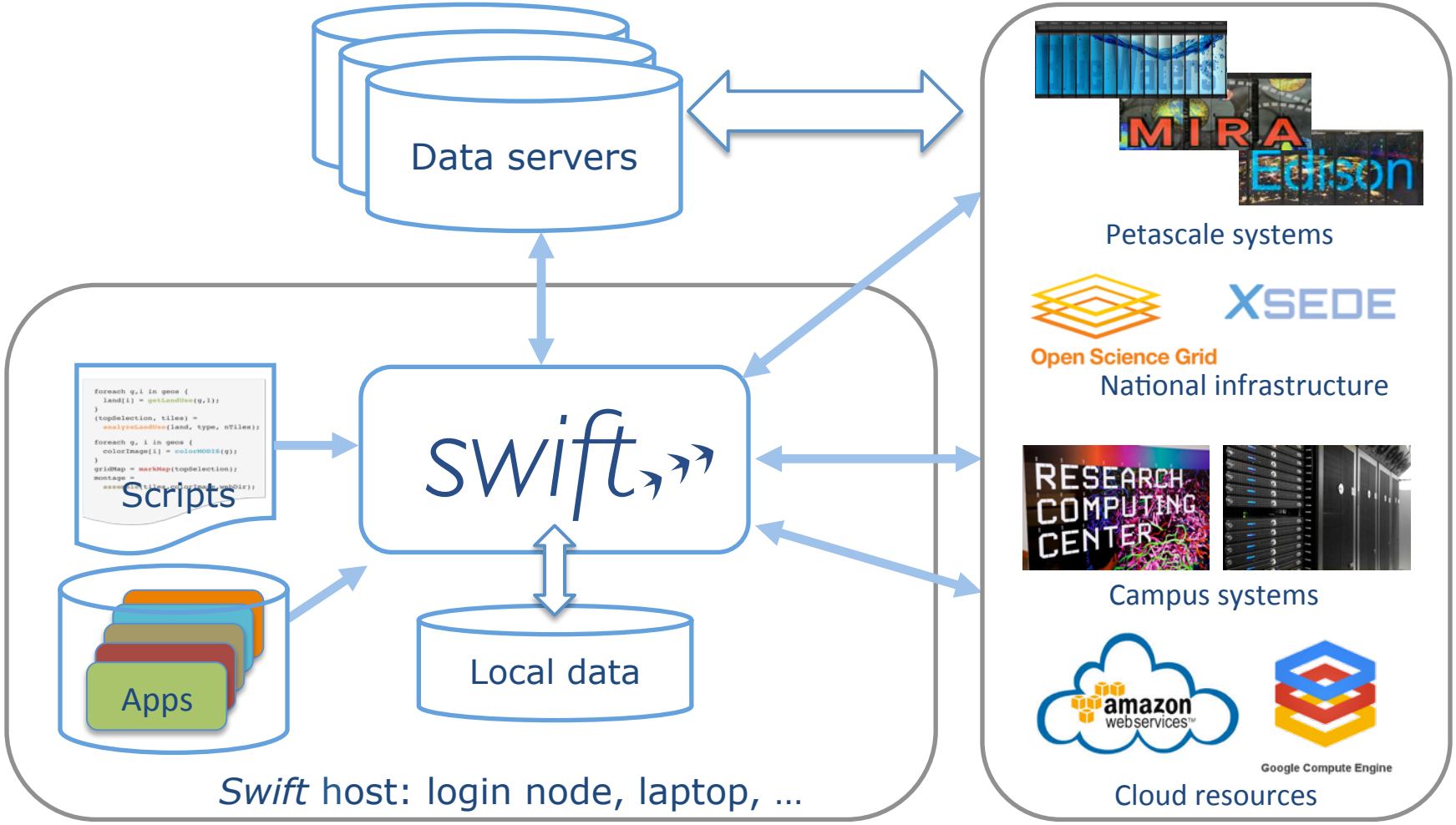
Work of Luiz Gadelha and Maria Luiza Mondelli, LNCC

Domain-specific metadata on workflow results

	filename	key	value
1	g0_1.fragsF.INF	seqx_length	3886916
2	g0_1.fragsF.INF	seqy_length	4847600
3	g0_1.fragsF.INF	min_fragment_length	100
4	g0_1.fragsF.INF	min_identity	65
5	g0_1.fragsF.INF	tot_hits_seeds	493943
6	g0_1.fragsF.INF	tot_hits_seeds_used	492470
7	g0_1.fragsF.INF	total_fragments	288

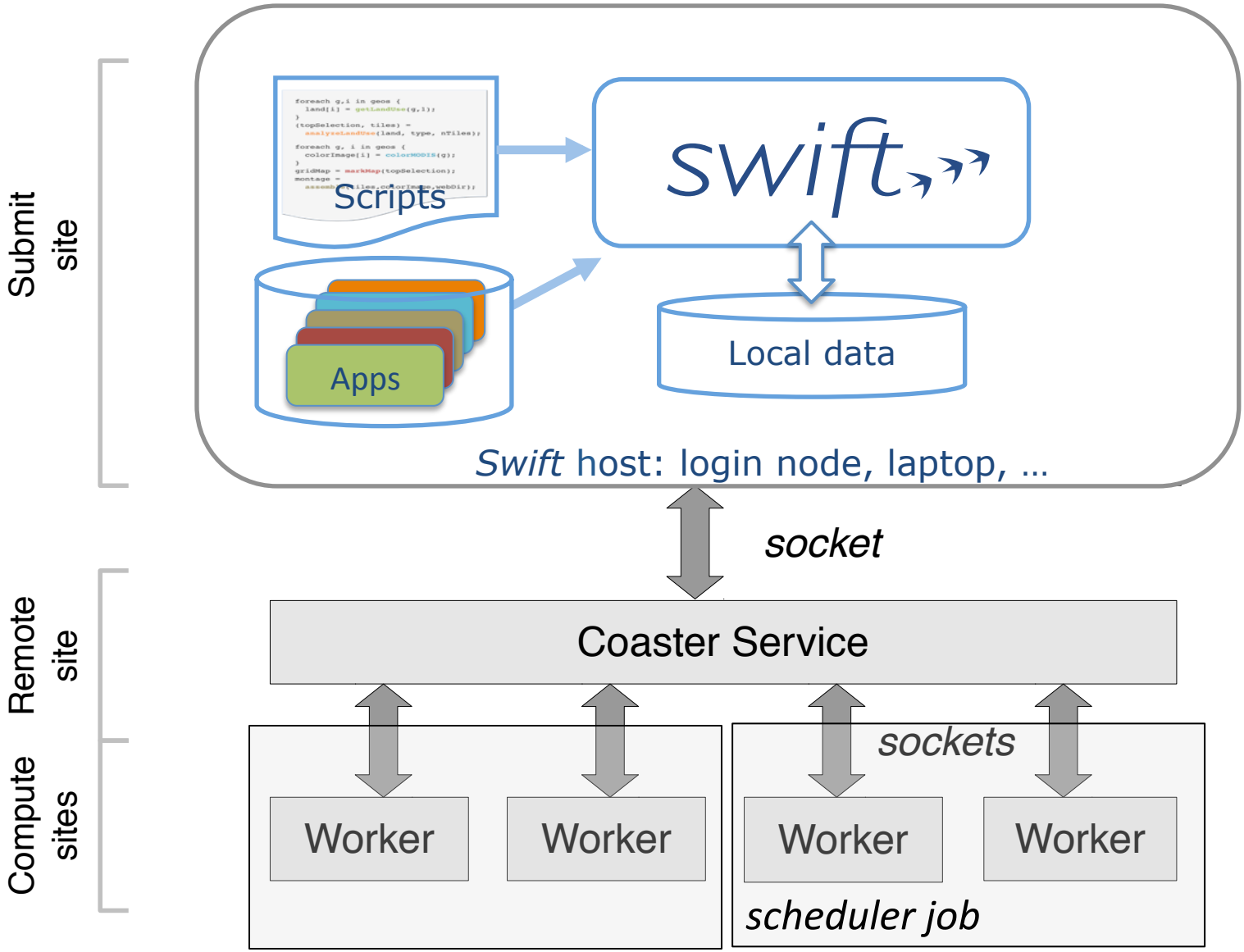
Work of Luiz Gadelha and Maria Luiza Mondelli, LNCC

Swift's distributed architecture is based on a client/worker mechanism (internally named "coasters")



The Swift runtime system has drivers and algorithms to efficiently support and aggregate diverse runtime environments

Worker architecture handles diverse environments



Summary of Swift main benefits

Makes parallelism more transparent

Implicitly parallel functional dataflow programming

Makes computing location more transparent

Runs your script on multiple distributed sites and diverse computing resources (desktop to petascale)

Makes basic failure recovery transparent

Retries/relocates failing tasks

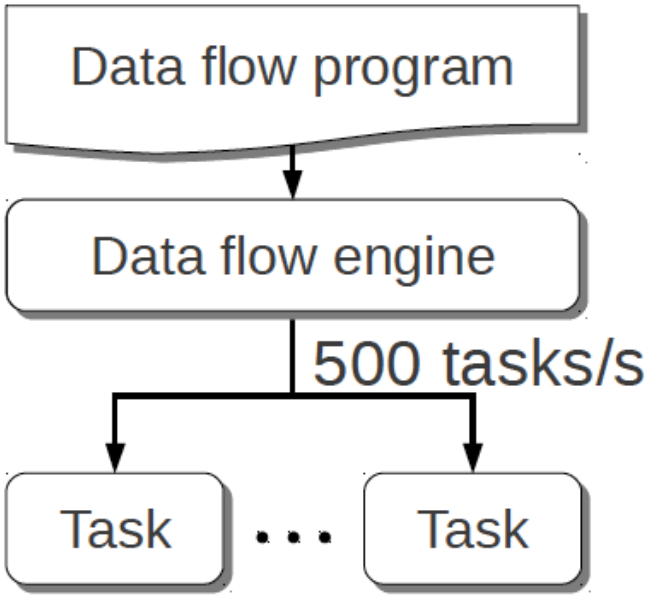
Can restart failing runs from point of failure

Enables provenance capture

Tasks have recordable inputs and outputs

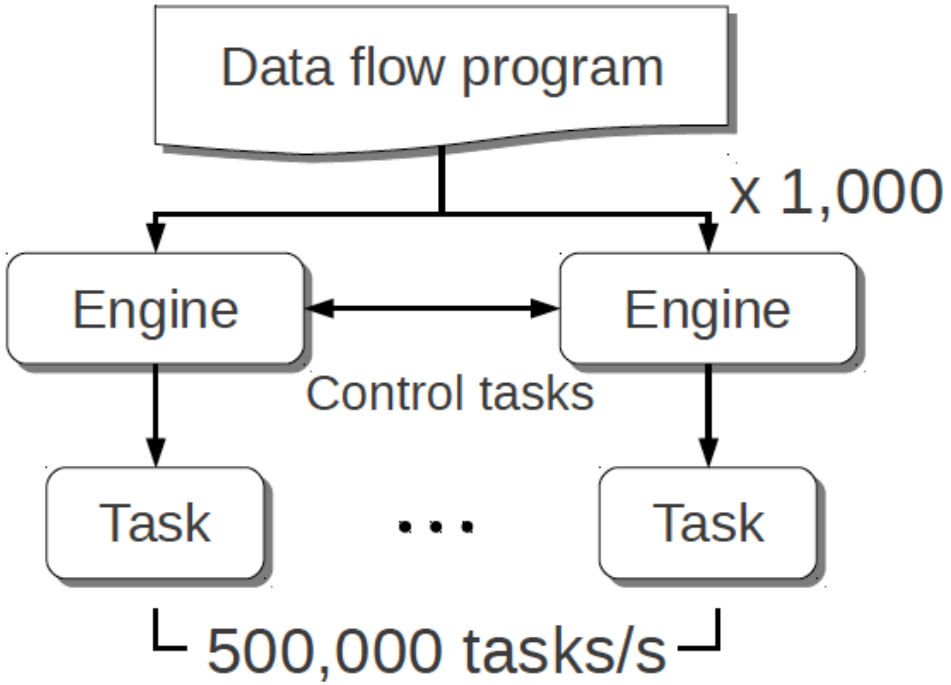
BUT: Centralized evaluation can be a bottleneck at extreme scales

Had this (Swift/K):



Centralized evaluation

For extreme scale, we need this (Swift/T):

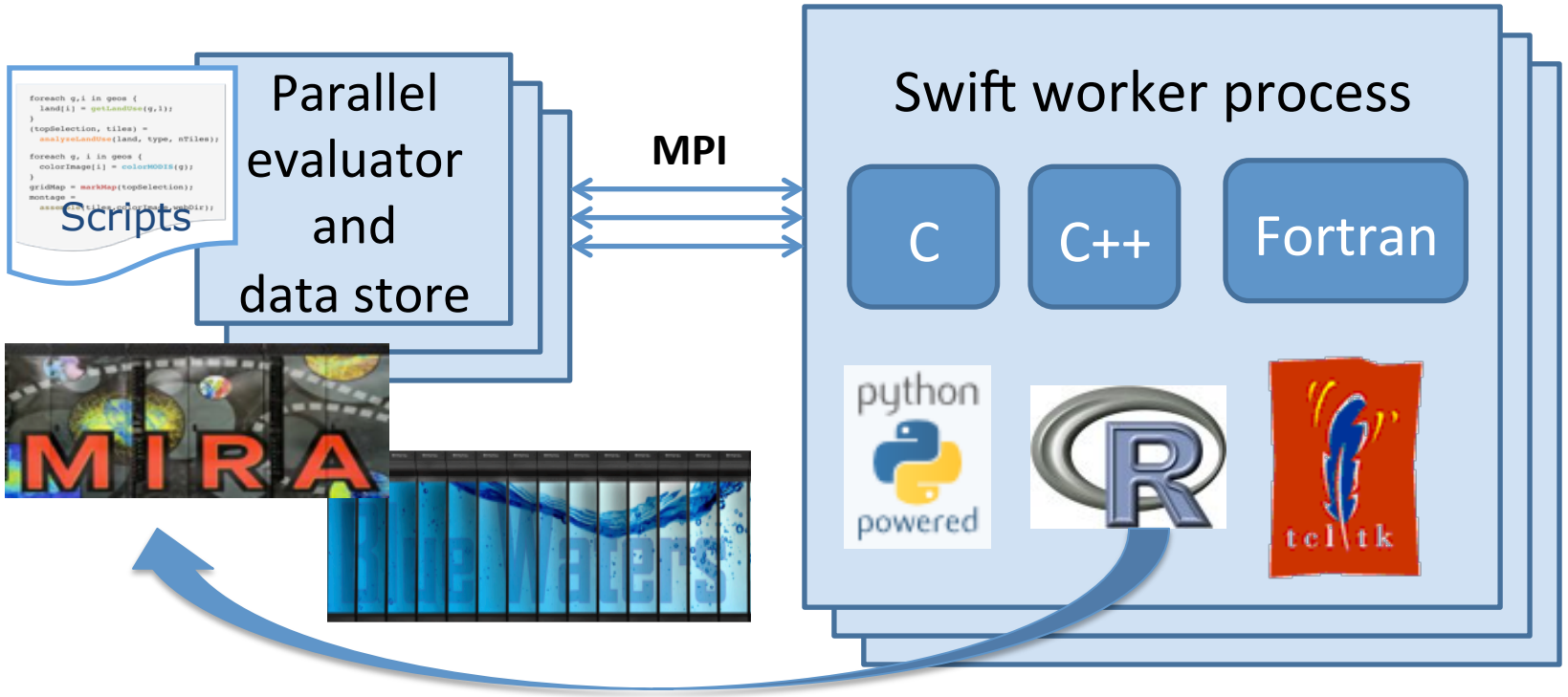


Distributed evaluation

Two Swift implementations

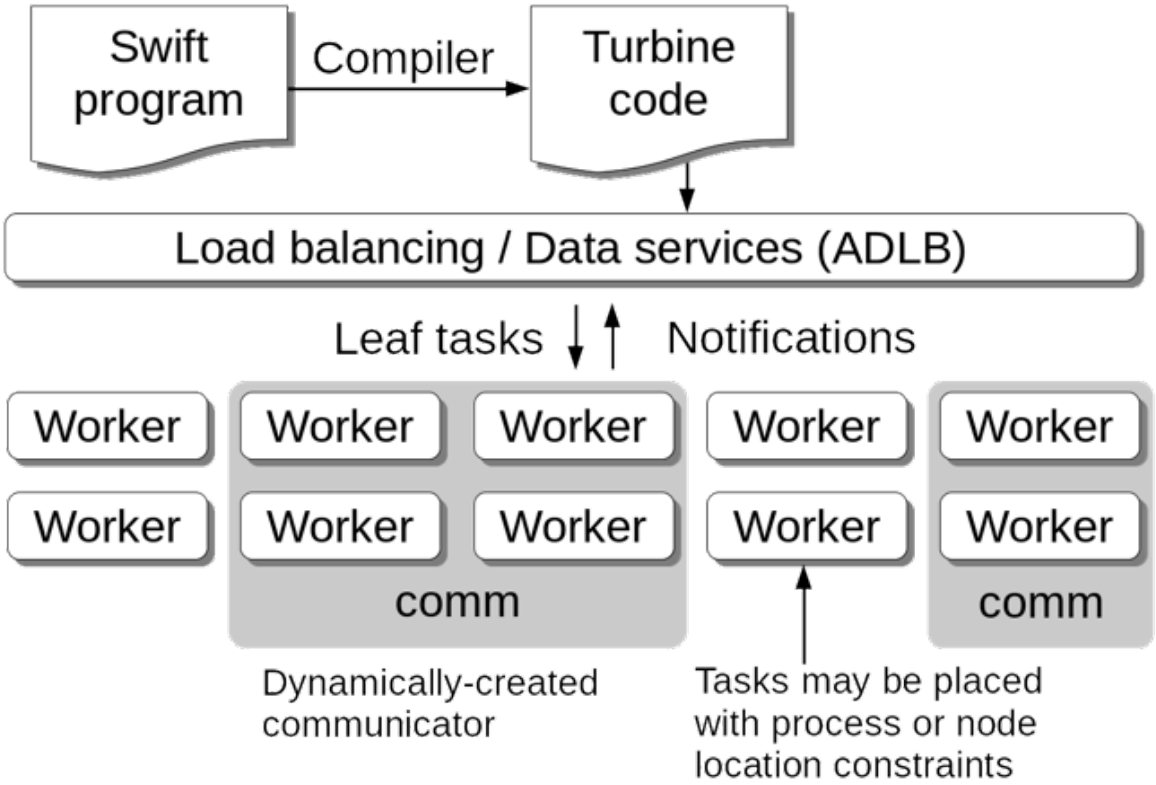
- Swift/K: Classic, Java Swift with "Karajan" engine
  - Portable, mature (2006)
  - Runs script from single node
  - Installs anywhere, instantly: just untar and run
  - Runs any POSIX app – serial or parallel
  - 500-1000 tasks/second
  - *Use for irregular workloads and flexible MPI app invocation*
- Swift/T: HPC Swift (in C) with "Turbine" engine
  - Faster, newer (2011)
  - Runs script and apps from multinode MPI program
  - Runs anywhere that MPI runs
  - Runs both POSIX apps and and library functions (C/C++, Fortran, Python, R, Julia)
  - 1.5 billion tasks/s on 512K Blue Waters cores
  - *Use for fine-grained tasking, in-memory workflow and single-core MPI apps*

Swift/T: productive extreme-scale scripting



- Script-like programming with “leaf” tasks
  - In-memory function calls in C++, Fortran, Python, R, … passing in-memory objects
  - More expressive than master-worker for “programming in the large”
  - Leaf tasks can be MPI programs, etc. Can be separate processes if OS permits.
- Distributed, *scalable* runtime manages tasks, load balancing, data movement
- User function calls to external code run on thousands of worker nodes

Parallel tasks in Swift/T

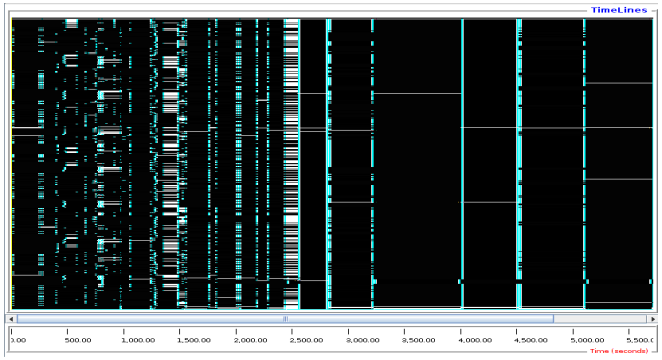


- Swift expression: `z = @par=32 f(x,y);`
- ADLB server finds 8 available workers
  - Workers receive ranks from ADLB server
  - Performs `comm = MPI_Comm_create_group()`
- Workers perform `f(x,y)` communicating on `comm`

LAMMPS parallel tasks

```
foreach i in [0:20] {
  t = 300+i;
  sed_command = sprintf("s/_TEMPERATURE_/%i/g", t);
  lammps_file_name = sprintf("input-%i.inp", t);
  lammps_args = "-i " + lammps_file_name;
  file lammps_input<lammps_file_name> =
    sed(filter, sed_command) =>
    @par=8 lammps(lammps_args);
}
```

- LAMMPS provides a convenient C++ API
- Easily used by Swift/T parallel tasks



Tasks with varying sizes packed into big MPI run
Black: Compute **Blue:** Message **White:** Idle

Swift/T-specific features

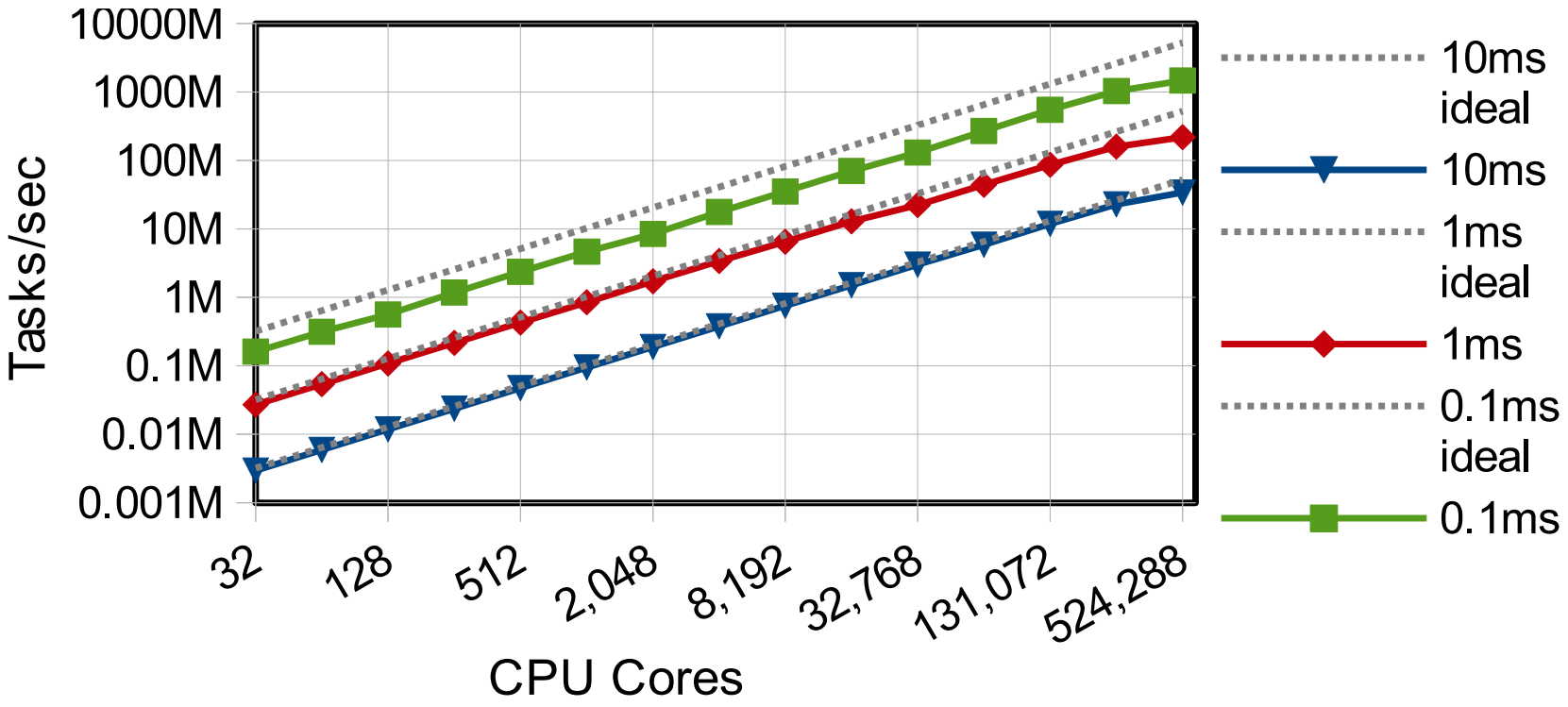
- Task locality: Ability to send a task to a process
  - Allows for big data –type applications
  - Allows for stateful objects to remain resident in the workflow
  - ```
    location L = find_data(D);
    int y = @location=L f(D, x);
    ```
- Data broadcast
- Task priorities: Ability to set task priority
  - Useful for tweaking load balancing
- Updateable variables
  - Allow data to be modified after its initial write
  - Consumer tasks may receive original or updated values when they emerge from the work queue

Wozniak et al. Language features for scalable distributed-memory dataflow computing. Proc. Dataflow Execution Models at PACT, 2014.


    ```

Swift/T: scaling of trivial foreach { } loop

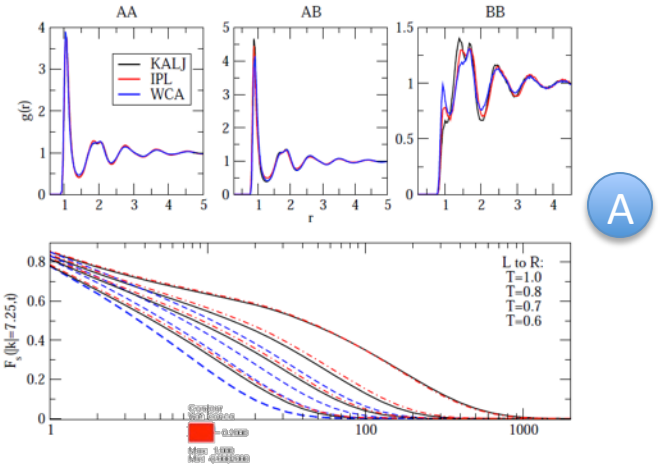
100 microsecond to 10 millisecond tasks on up to 512K integer cores of Blue Waters

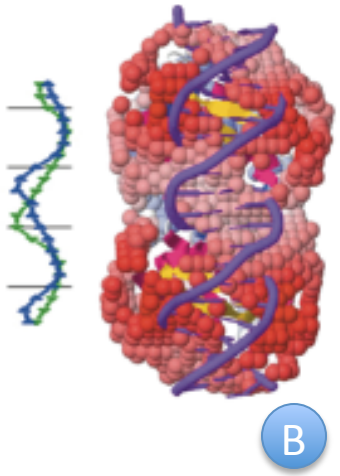


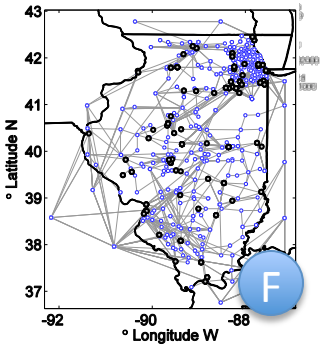
Large-scale applications using Swift

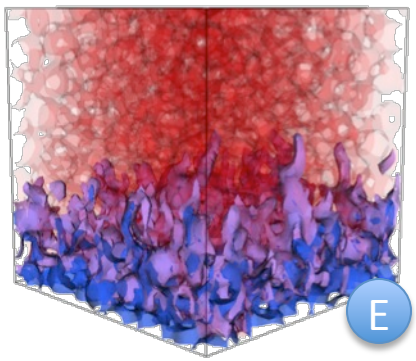
- **A** Simulation of super-cooled glass materials
- **B** Protein and biomolecule structure and interaction
- **C** Climate model analysis and decision making for global food production & supply
- **D** Materials science at the Advanced Photon Source
- **E** Multiscale subsurface flow modeling
- **F** Modeling of power grid for OE applications

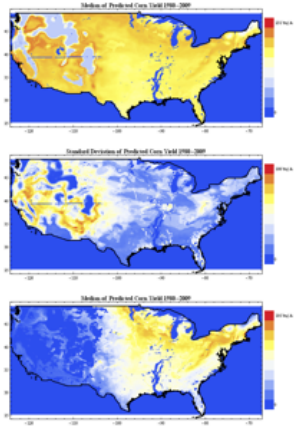
All have published science results obtained using Swift

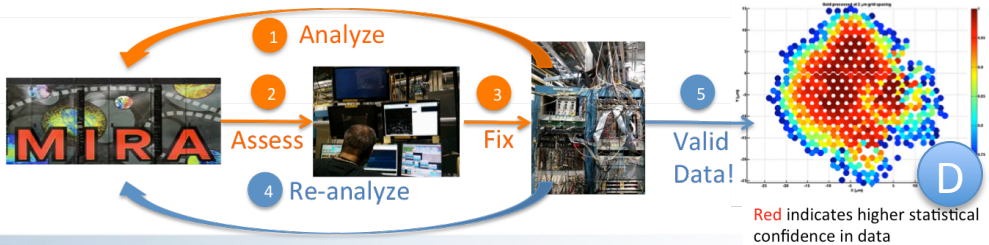




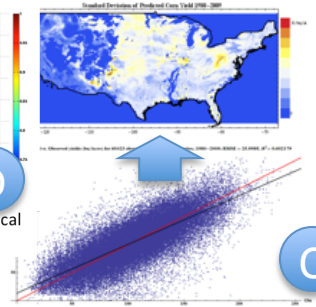








Red indicates higher statistical confidence in data

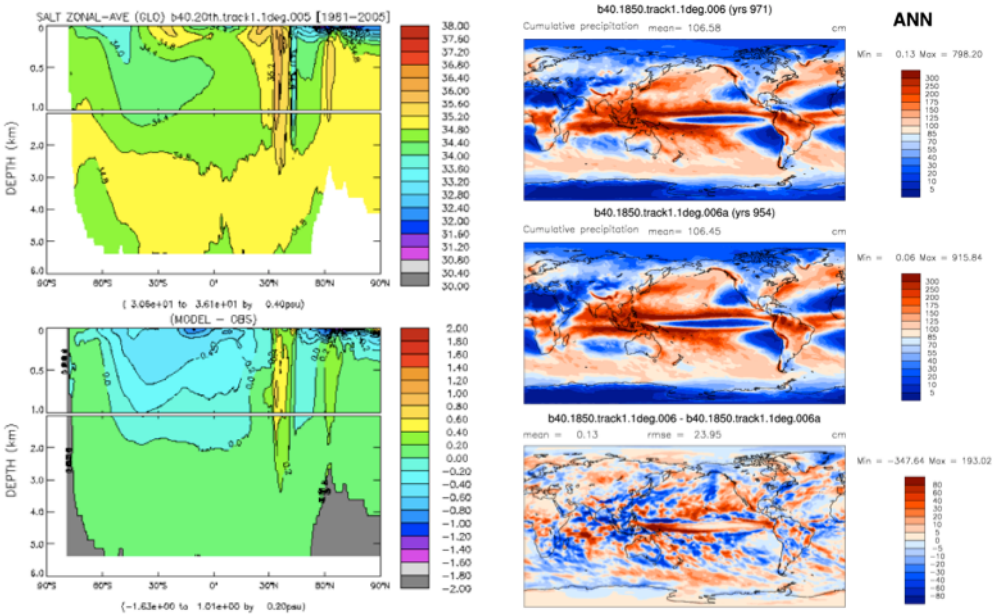


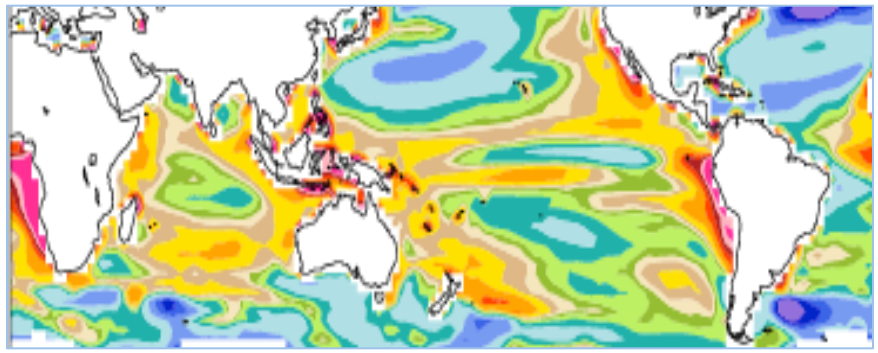
Benefit of implicit pervasive parallelism: Analysis & visualization of high-resolution climate models

powered by *Swift*

parVis

- Diagnostic scripts for each climate model (ocean, atmospehere, land, ice) were expressed in complex shell scripts
- Recoded in Swift, the CESM community has benefited from significant speedups and more modular scripts





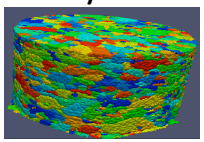
Work of: J Dennis, M Woitasek, S Mickelson, R Jacob, M Vertenstein

Boosting Light Source Productivity with *Swift* ALCF Data Analysis

H Sharma, J Almer (APS); J Wozniak, M Wilde, I Foster (MCS)

Impact and Approach

- HEDM imaging and analysis shows granular material structure, non-destructively



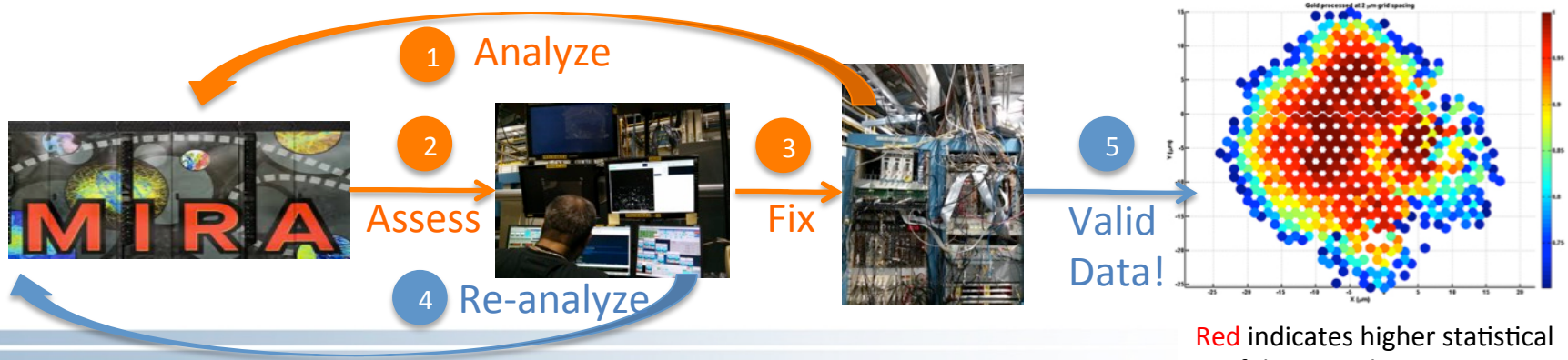
- APS Sector 1 scientists use Mira to process data from live HEDM experiments, providing real-time feedback to correct or improve in-progress experiments
- Scientists working with *Discovery Engines* LDRD developed new *Swift* analysis workflows to process APS data from Sectors 1, 6, and 11

Accomplishments

- Mira analyzes experiment in 10 mins vs. 5.2 hours on APS cluster: > 30X improvement
- Scaling up to ~ 128K cores (driven by data features)
- **Cable flaw was found and fixed at start of experiment,** saving an entire multi-day experiment and valuable user time and APS beam time.
- **In press:** *High-Energy Synchrotron X-ray Techniques for Studying Irradiated Materials*, J-S Park et al, J. Mat. Res.
- *Big data staging with MPI-IO for interactive X-ray science*, J Wozniak et al, Big Data Conference, Dec 2014

ALCF Contributions

- Design, develop, support, and trial user engagement to make *Swift* workflow solution on ALCF systems a reliable, secure and supported production service
- Creation and support of the Petrel data server
- Reserved resources on Mira for APS HEDM experiment at Sector 1-ID beamline (8/10/2014 and future sessions in APS 2015 Run 1)



Red indicates higher statistical confidence in data

Conclusion: parallel workflow scripting is practical, productive, *and necessary*, at a broad range of scales

- Swift programming model demonstrated feasible and scalable on XSEDE, Blue Waters, OSG, DOE systems
- Applied to numerous MTC and HPC application domains
  - attractive for data-intensive applications
  - and several hybrid programming models
- Proven productivity enhancement in materials, genomics, biochem, earth systems science, …
- Deep integration of workflow in progress at XSEDE, ALCF

Workflow through implicitly parallel dataflow is productive for applications and systems at many scales, including on highest-end system

What's next?

- Programmability
  - New patterns ala Van Der Aalst et al (workflowpatterns.org)
- Fine grained dataflow – programming in the smaller?
  - Run leaf tasks on accelerators (CUDA GPUs, Intel Phi)
  - How low/fast can we drive this model?
- PowerFlow
  - Applies dataflow semantics to manage and reduce energy usage
- Extreme-scale reliability
- Embed Swift semantics in Python, R, Java, shell, make
  - Can we make Swift "invisible"? Should we?
- Swift-Reduce
  - Learning from map-reduce
  - Integration with map-reduce

GeMTC: GPU-enabled Many-Task Computing

Motivation: Support for MTC on all accelerators!

Goals:

1) MTC support 2) Programmability

3) Efficiency 4) MPMD on SIMD

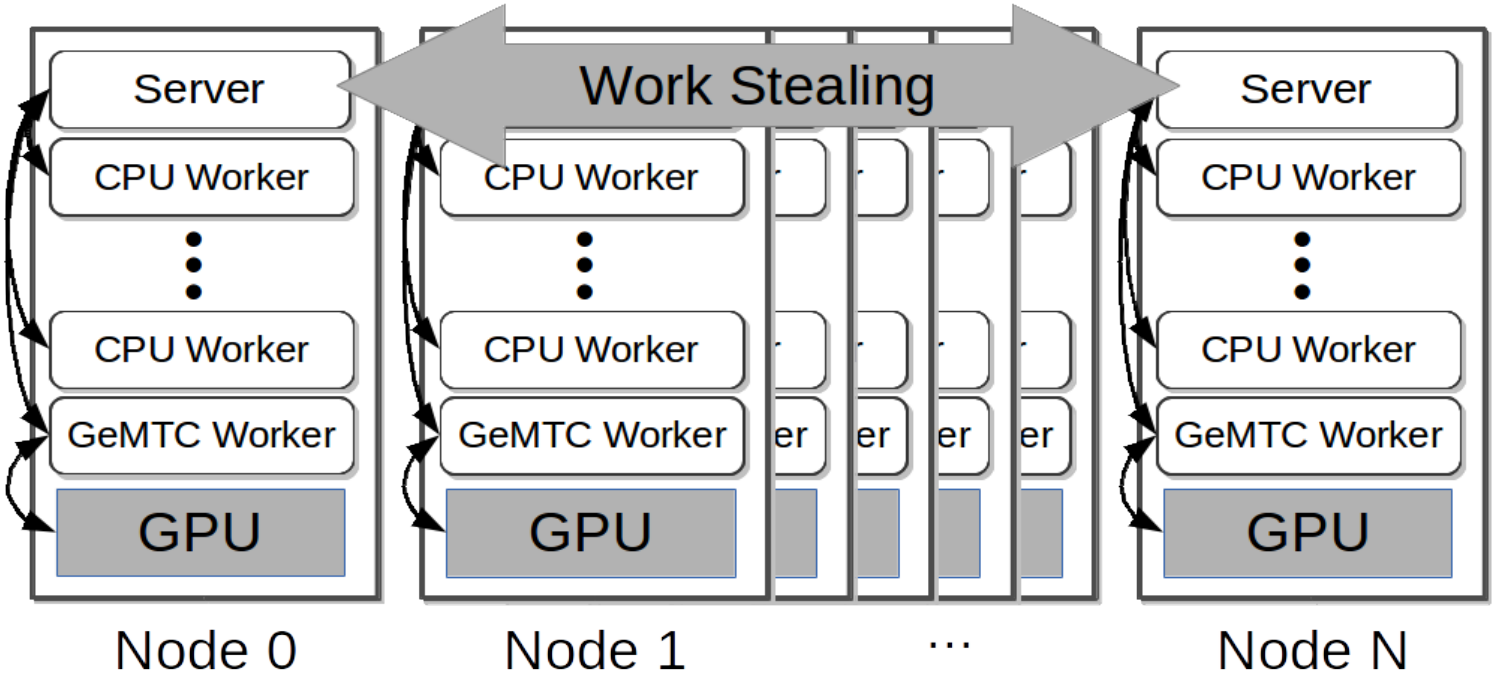
5) Increase concurrency to warp level

Approach:

Design & implement GeMTC middleware:

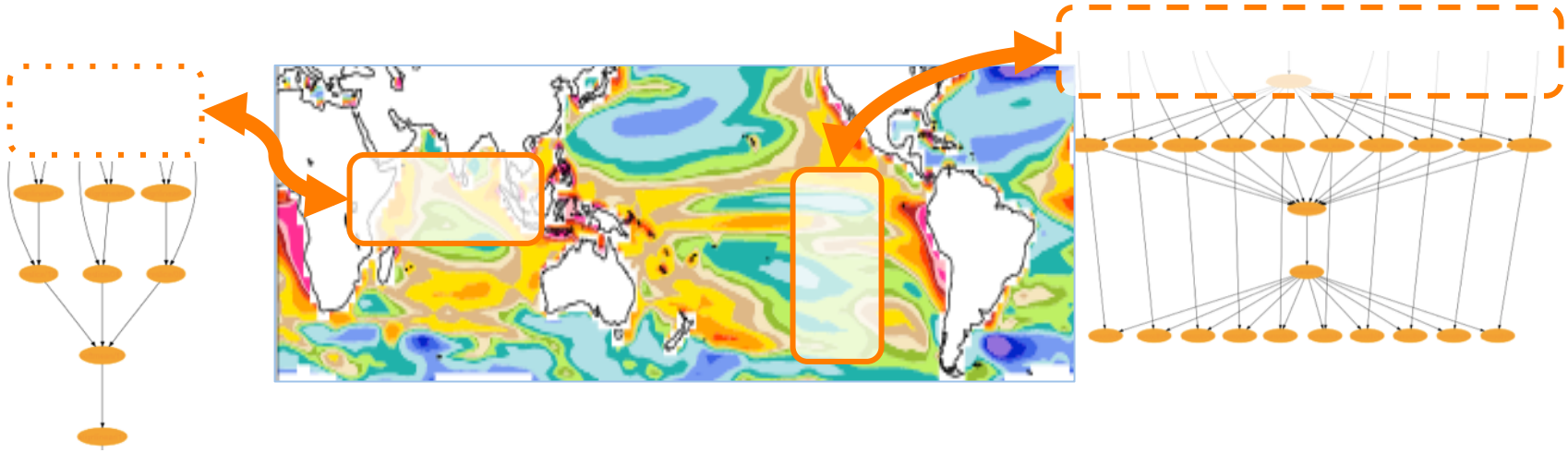
1) Manages GPU 2) Spread host/device

3) Workflow system integration (Swift/T)



Further research directions

- Deeply in-situ processing for extreme-scale analytics
- Shell-like Read-Evaluate-Print Loop ala iPython
- Debugging of extreme-scale workflows



Deeply in-situ analytics of a climate simulation

The Swift Team

- Timothy Armstrong, Yadu Nand Babuji, Ian Foster, Mihael Hategan, Daniel S. Katz, Ketan Maheshwari, Michael Wilde, Justin Wozniak, Yangxinye Yang
- 2015 REU Summer Collaborators: Jonathan Burge, Mermer Dupres, Basheer Subei, Jacob Taylor
- Contributions by Zhao Zhang, Ben Clifford, Luiz Gadelha, Yong Zhao, Scott Krieder, Ioan Raicu, Tiberius Stef-Praun, Matthew Shaxted
- Sincere thanks to the entire Swift user community

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

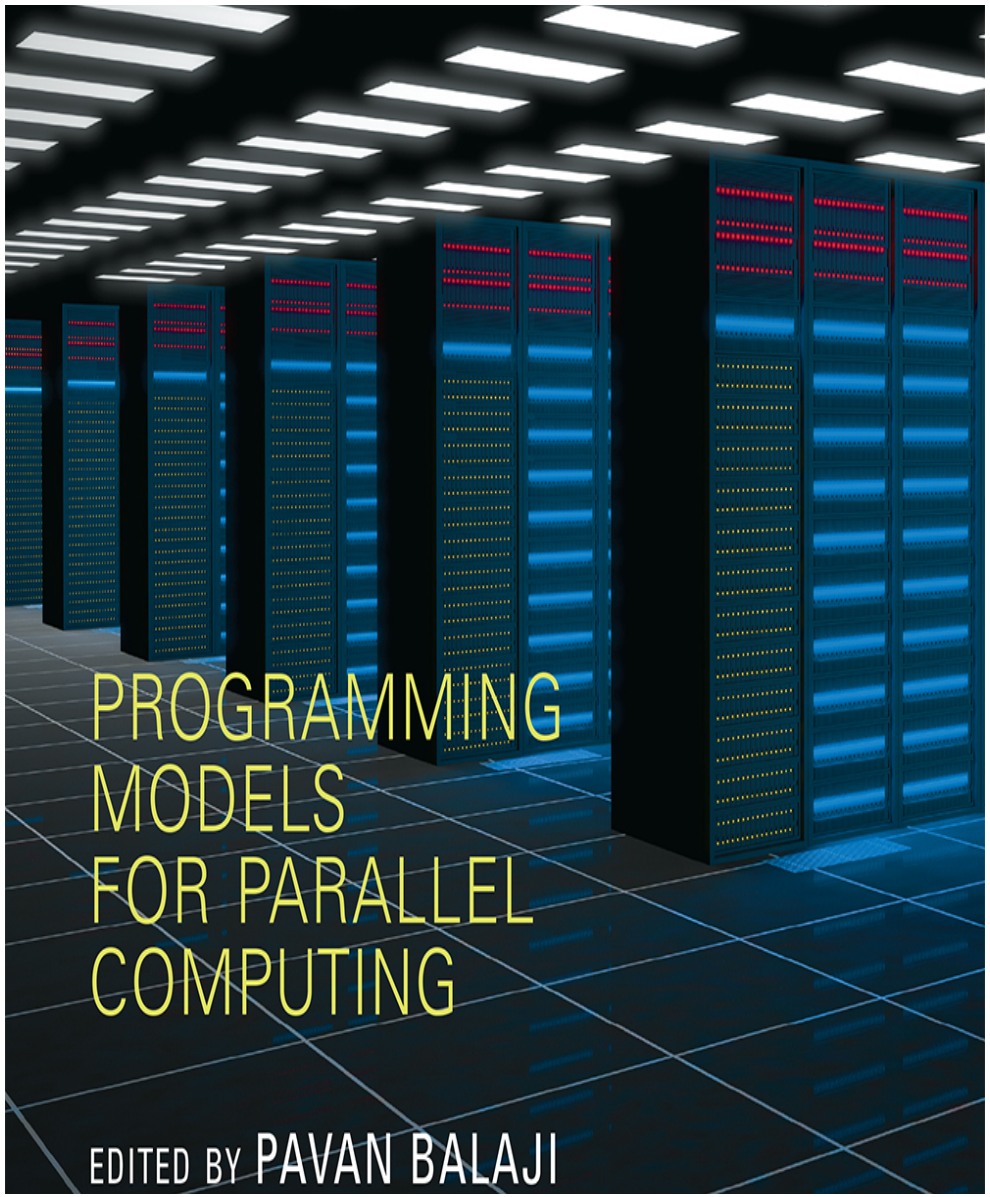
Scientists, engineers, and statisticians must execute domain-specific application programs many times on large collections of file-based data. This activity requires complex orchestration and data management as data is passed to, from, and among application invocations. Distributed and parallel computing 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.

New Book on Programming Models

Editor: Pavan Balaji

Chapter Contributions:

- ***MPI:*** W. Gropp and R. Thakur
- ***GASNet:*** P. Hargrove
- ***OpenSHMEM:*** J. Kuehn and S. Poole
- ***UPC:*** K. Yelick and Y. Zheng
- ***GA:*** S. Krishnamoorthy, J. Daily, A. Vishnu, and B. Palmer
- ***Chapel:*** B. Chamberlain
- ***Charm++:*** L. Kale, N. Jain, and J. Lifflander
- ***ADLB:*** E. Lusk, R. Butler, and S. Pieper
- ***Scioto:*** J. Dinan
- ***Swift:*** T. Armstrong, J. M. Wozniak, M. Wilde, and I. Foster
- ***CnC:*** K. Knobe, M. Burke, and F. Schlimbach
- ***OpenMP:*** B. Chapman, D. Eachempati, and S. Chandrasekaran
- ***Cilk Plus:*** A. Robison and C. Leiserson
- ***Intel TBB:*** A. Kukanov
- ***CUDA:*** W. Hwu and D. Kirk
- ***OpenCL:*** T. Mattson



Pre-order at https://mitpress.mit.edu/models
Discount code: MBALAJI30
(valid till 12/31/2015)

Join the Swift Community: explore, use, engage

swift

downloads documentation case studies papers support

A simple tool for fast, easy scripting on big machines.

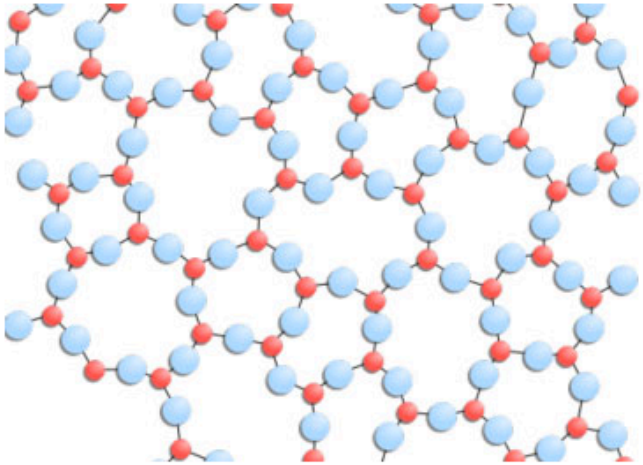
SWIFT NEWS:

⇒ Oct. 5: LATEST SWIFT PROJECT REPORT details over 20 scientific engagements

⇒ Nov 6: Swift chapter in new PARALLEL PROGRAMMING MODELS TEXT available at SC15

⇒ Swift at the SC15 Supercomputing Conference

⇒ Preview of new Swift Language Reference



Modeling the molecular structure of glass materials using theoretical chemistry methods, on the Open Science Grid and UChicago's Beagle Cray supercomputer.

read more

Try Swift online

Try Swift from your browser

Get Started

Run tutorial examples today

Download Now

Current: 0.96.2, 2015/08/05

Join

Join the Swift community

Try Swift/T

Our MPI-based version.

Learning Swift

swift downloads documentation

Documentation

Swift Quickstart

This guide describes the steps needed to download, install, configure, and run the b for Swift.

- Quickstart Guide

Swift Tutorials

This hands-on tutorial gives a quick (~20 minute) taste of Swift. It shows you how to workflows on computation resources such as :

- Ad-hoc nodes
- Beagle (UChicago)
- Blues (LCRC)
- Edison (NERSC)
- Elastic Cloud Compute (Amazon)
- Midway (UChicago)
- Open Science Grid
- Stampede (XSEDE/TACC)
- Swan (Cray MPN)

For a complete list go here.

Swift User Guide

The User Guide provides more detailed reference documentation and background ir swift. It is assumed that the reader is already familiar with the material in the Quickst Tutorial documents.

- Latest (0.96) [html] [pdf]
- Interim (0.95) [html] [pdf]
- Trunk [html] [pdf]

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

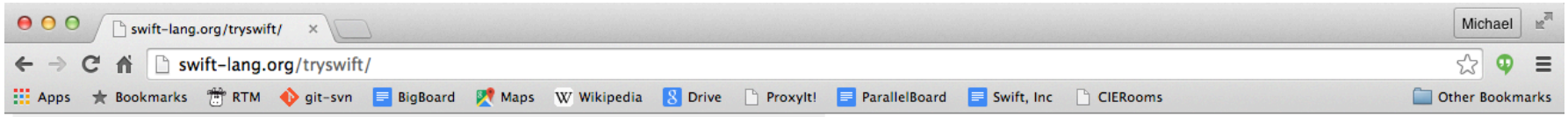
Keywords:
Swift
Parallel programming
Scripting
Dataflow

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.

Try Swift from your browser:

http://swift-lang.org/tryswift




```
35 # Dynamically generated bias for simulation ensemble
36 file seedfile<"output/seed.dat">;
37 seedfile = genseed_app(1);
38
39 int seedval = readData(seedfile);
40 tracef("Generated seed=%i\n", seedval);
41
42 file sims[]; # Array of files to hold each simulation output
43
44 foreach i in [0:nsim-1] {
45   file biasfile <single_file_mapper;
46                  file=strcat("output/bias_",i,".dat")>;
47   file simout   <single_file_mapper;
48                  file=strcat("output/sim_",i,".out")>;
49   file simlog   <single_file_mapper;
50                  file=strcat("output/sim_",i,".log")>;
51   biasfile = genbias_app(1000, 20);
52   (simout,simlog) = simulation_app(steps, range, biasfile,
53                                    1000000, values);
54   sims[i] = simout;
55 }
56
57 file stats_out<"output/average.out">;
```

Execute | Reset | File outputs | < | Multi-stage workflows | >

```
Swift run starting at 05:02:07
05:02:10

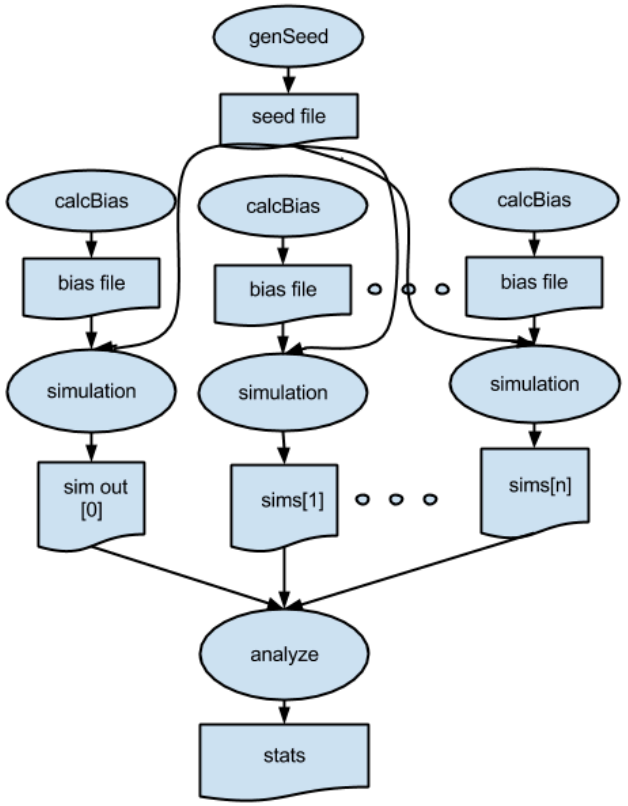
*** Script parameters: nsim=10 range=100 num values=10

Generated seed=66127
05:02:12  Ready:7  Active:3  Done:11
05:02:13  Ready:6  Active:1  Done:14
05:02:14  Ready:4  Active:3  Done:14
05:02:15  Ready:1  Active:3  Done:17
05:02:16  Active:1  Done:20
Final status: 05:02:16  Done:22

Generated 22 file(s)
Swift completed
```

Multi-stage workflows

This example expands the workflow pattern of the previous example by adding additional stages to the workflow. Here, we generate a dynamic seed value that will be used by all of the simulations, and for each simulation, we run an pre-processing application to generate a unique "bias file". This pattern is shown below, followed by the Swift script.



Appendix:

Examples of Swift Science Applications (~ Sep 2014)

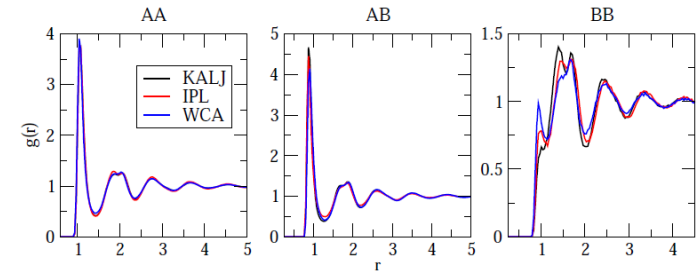
Glass Structure Modeling
powered by *Swift*

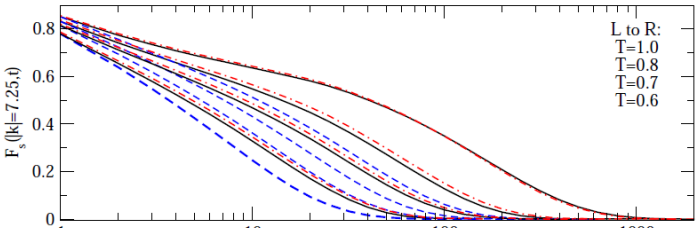
This project models of aspects of glass structure at a theoretical chemistry level. (Hocky/Reichman)

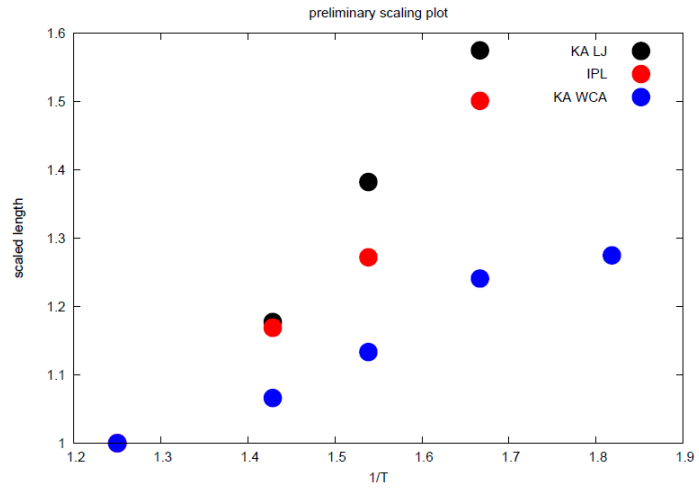
Recent studies of the glass transition in model systems have focused on calculating from theory or simulation what is known as the “mosaic length”. This project evaluated a new “cavity method” for measuring this length scale. Correlation functions are calculated at the interior of cavities of varying sizes and averaged over many independent simulations to determine a thermodynamic length. Using Swift on Beagle, Hocky investigated whether this thermodynamic length causes variations among seemingly identical systems. ~1M Beagle CPU hours were used.

Results: Three simple models of glassy behavior were studied. All appear the same (top, abc) but only two of which have particles relaxing at the same rate for the same temperature (top, d). This would imply that the glass structure does not dictate the dynamics. A new computational technique was used to extract a length scale on which the liquid is ordered in an otherwise undetectable way. Results (bottom) showed that using this length we can distinguish the two systems which have the same dynamics as separate from the third which has faster dynamics than the other two.

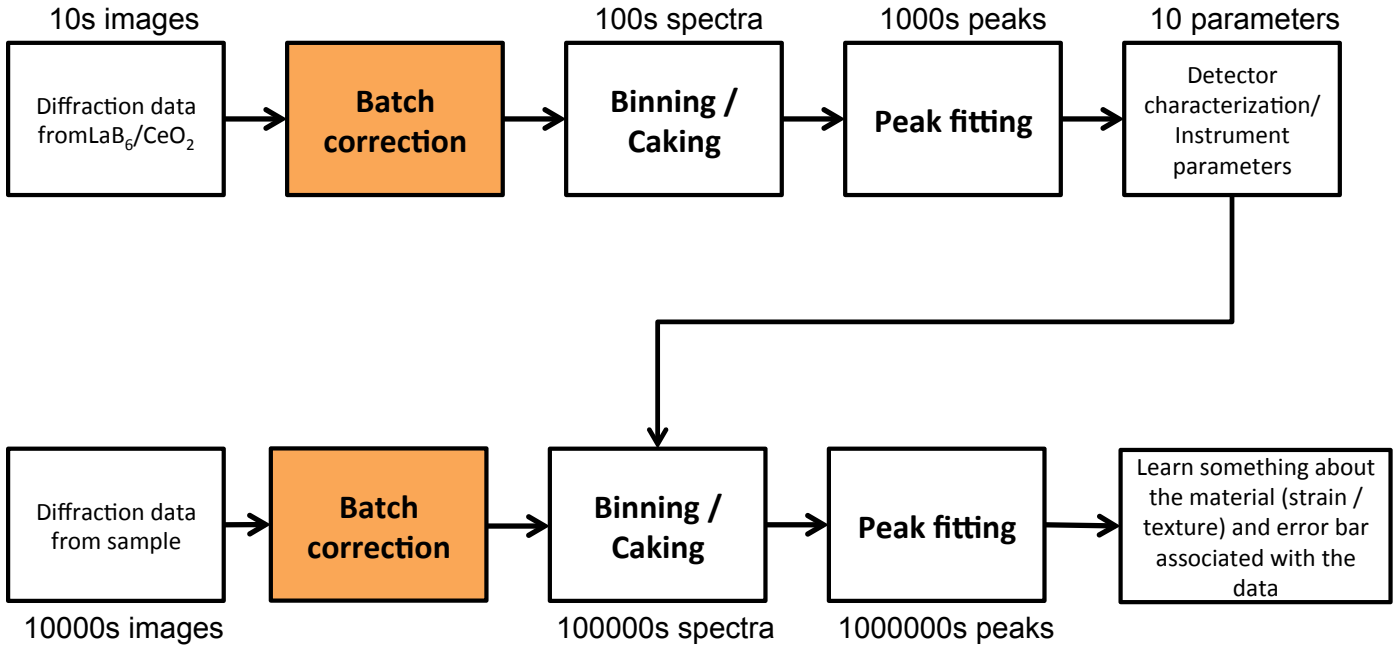
Published in Physical Review Letters B.







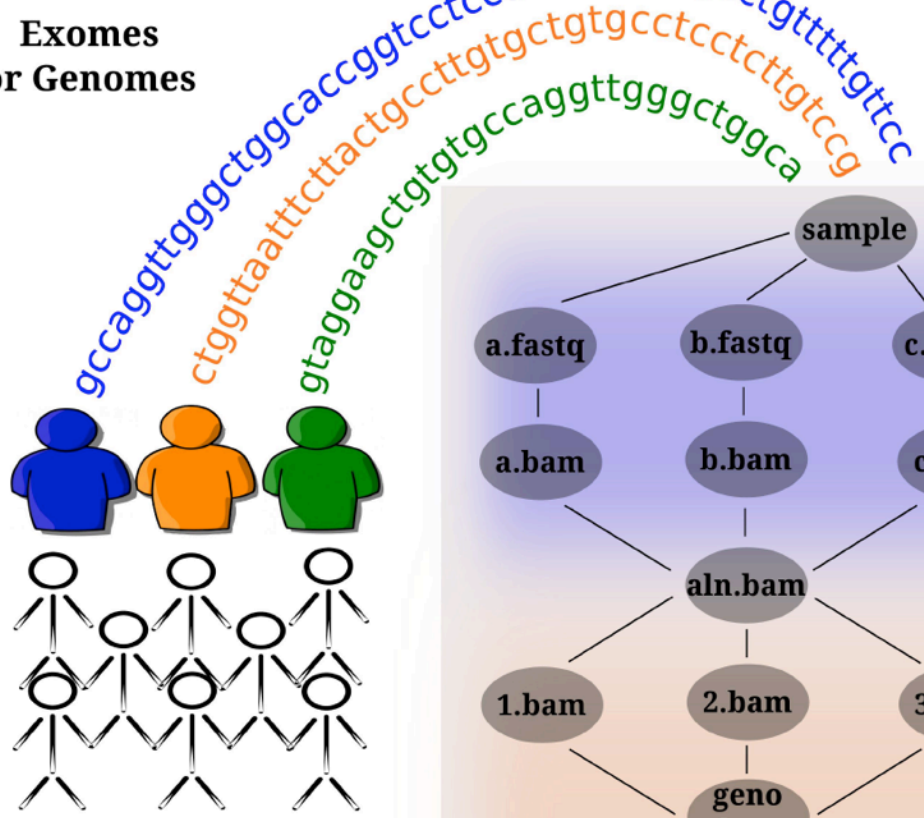
Powder diffraction experiment analysis workflow: making a notable difference to APS users!

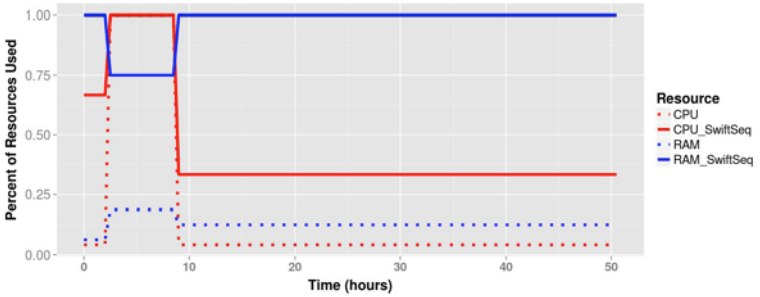


- Background-removal step extracted into separate step for Powder Diffraction beamline (Sector 1)
- Used 210+ times by 30 users to process 50TB (90% of PD data at Sector 1) in the past 6 months
- Enables Sector 1 users to test data quality at beam time, and to leave APS with all their data, ready to analyze

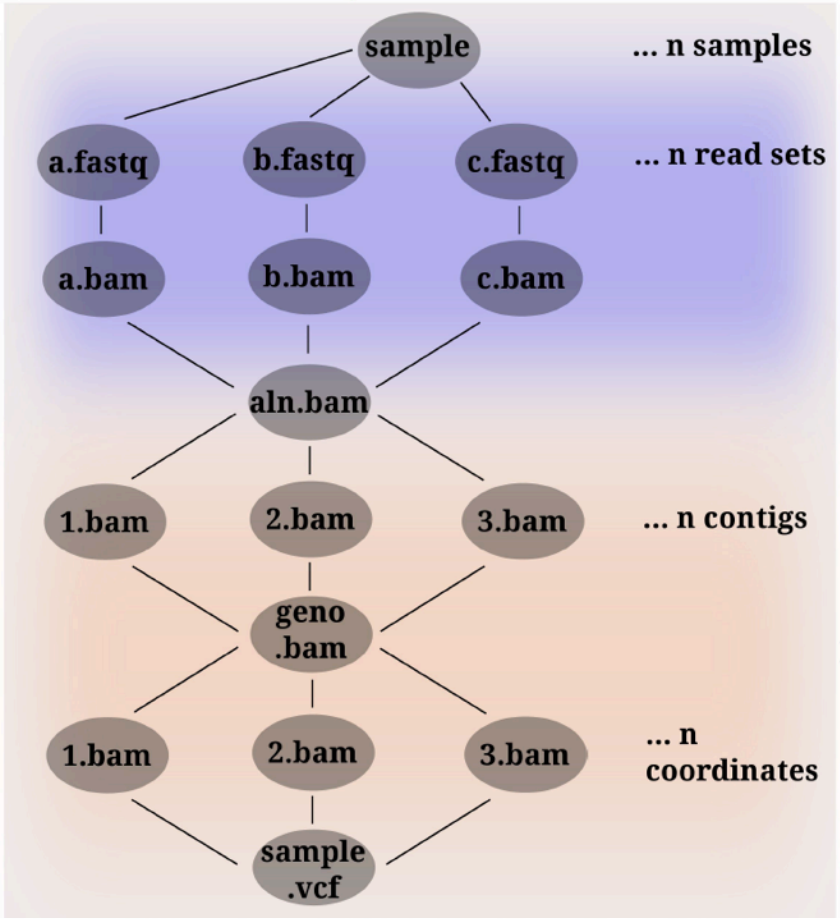
SwiftSeq

Fast parallel annotation of next-generation sequence data powered by *Swift*





Efficiently and dynamically balances RAM and CPU on 5K-15K cores



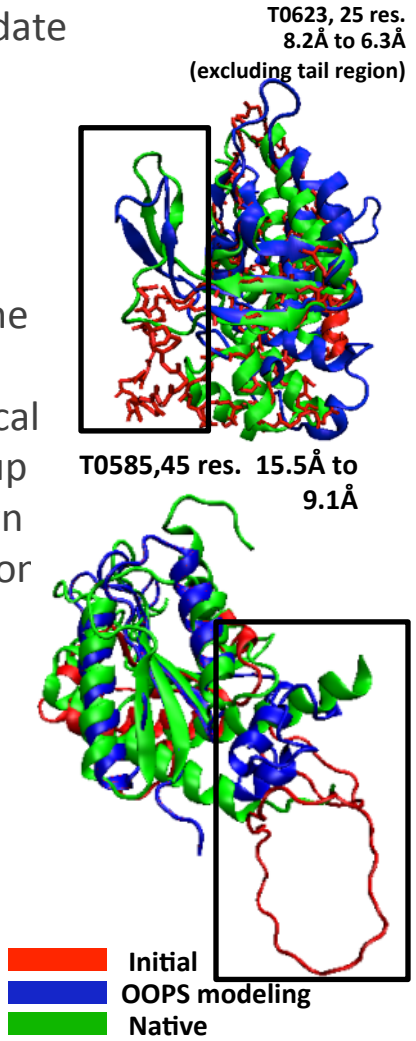
Work of Jason Pitt and Kevin White, UChicago IGSB

Protein structure prediction

powered by *Swift*

The laboratories of Karl Freed and Tobin Sosnick use Beagle to develop and validate methods to predict protein structure using homology-free approaches.

In this lab, Aashish Adhikari (now UC Berkeley) has developed new structure prediction techniques based on Monte Carlo simulated annealing which employ novel, compact molecular representations and innovative “moves” of the protein backbone to achieve accurate prediction with far less computation then previous methods. One of the applications of the method involves rebuilding local regions in protein structures, called “loop modeling”, a problem which the group tackled with considerable success in the CASP protein folding tournament(shown right).They are now testing further algorithmic innovations using the computatior power of Beagle.



Results: The group developed a new iterative algorithm for predicting protein structure and folding pathway starting only from the amino acid sequence.

Protein loop modeling. Courtesy A. Adhikari

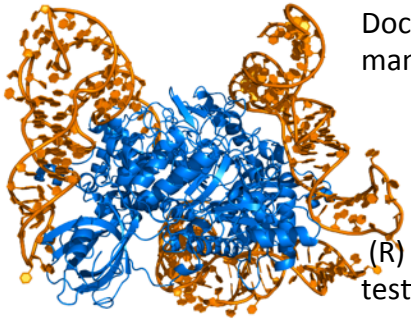
Protein-RNA interaction modeling

powered by *Swift*

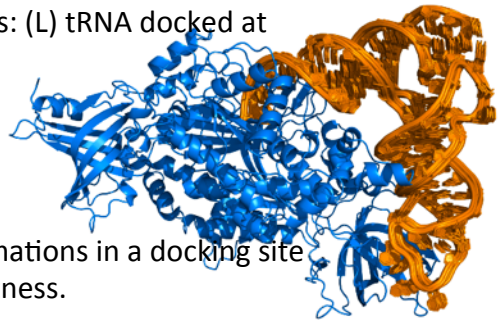
M. Parisien (with T. Sosnick, T. Pan, and K. Freed) developed a novel algorithm for the prediction of the RNA-protein interactome, on the UCHicago Beagle Cray XE6, powered by Swift.

Non-coding RNAs often function in cells through specific interactions with their protein partners. Experiments alone cannot provide a complete picture of the RNA-protein interactome. To complement experimental methods, computational approaches are highly desirable. No existing method, however, can provide genome-wide predictions of docked RNA-protein complexes.
The application of computational predictions, together with experimental methods, provides a more complete understanding on cellular networks and function of RNPs. The approach makes use of a rigid-body docking algorithm and a scoring function custom- tailored for protein-tRNA interactions. Using Swift, Beagle screened ~300 proteins per day on 1920 cores.

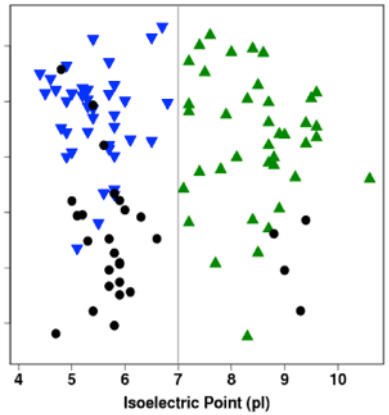
Results: the scoring function can identify the native docking conformation in large sets of decoys (100,000) for many known protein-tRNA complexes (4TRA shown here). (left) Scores for true positive complexes (.)(N=28) are compared to true negative ones of low (▼)(N=40) and high (▲) (N=40) isoelectric points. (right) Because the density curve of the true positives, which have pI < 7, has minimal overlap with the curve of the low pI true negatives (blue area), the scoring function has the specificity to identify tRNA-binding proteins.

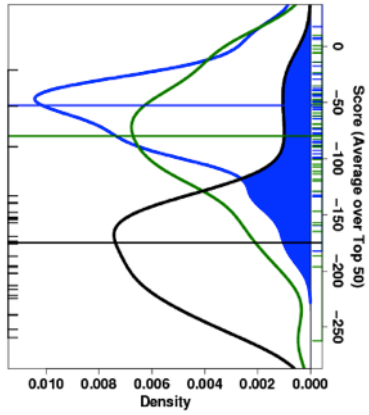


Docked complexes: (L) tRNA docked at many positions.



(R) Many conformations in a docking site testing site robustness.





Systematic prediction and validation of RNA-protein interactome. Parisien M, Sosnick TR, Pan T. Kyoto, June 12-19, 2011, RNA Society.

Protein-RNA interaction. Courtesy M. Parisien

RDCEP

Center for Robust Decision Making on Climate and Energy Policy
powered by *Swift*

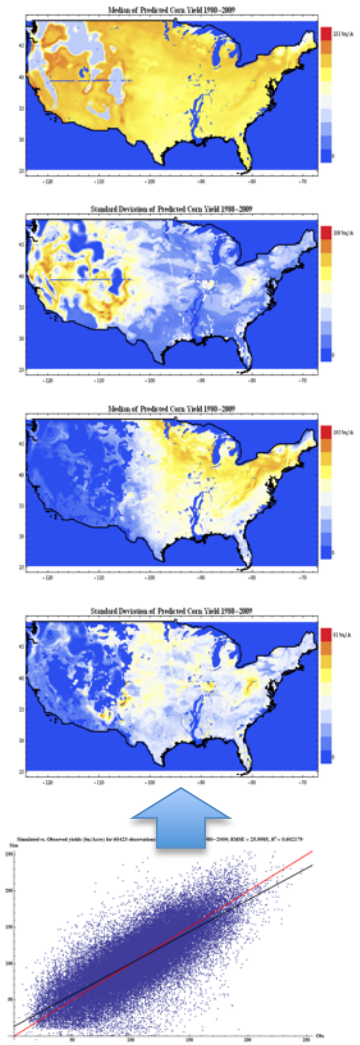
The RDCEP project operates a large-scale integrated modeling framework for decision makers in climate and energy policy, powered by Swift.
(Ian Foster, Joshua Elliott, et al)

UChicago's Midway research cluster is used to study land use, land cover, and the impacts of climate change on agriculture and the global food supply. Using a DSSAT 4.0 ("decision support system for agrotechnology transfer") crop systems model, a parallel simulation framework was implemented using *Swift*. Simulation campaigns measure, e.g., yield and climate impact for a single crop (maize) across the conterminous USA with daily weather data and climate model output for 120 years (1981-2100) and 16 different configurations fertilizer, irrigation, and cultivar.

Top two maps: maize yields across the USA with intensive nitrogen application and full irrigation

Bottom two maps show results with no irrigation.

Each map is a RDCEP model run of ~120,000 DSSAT invocations.



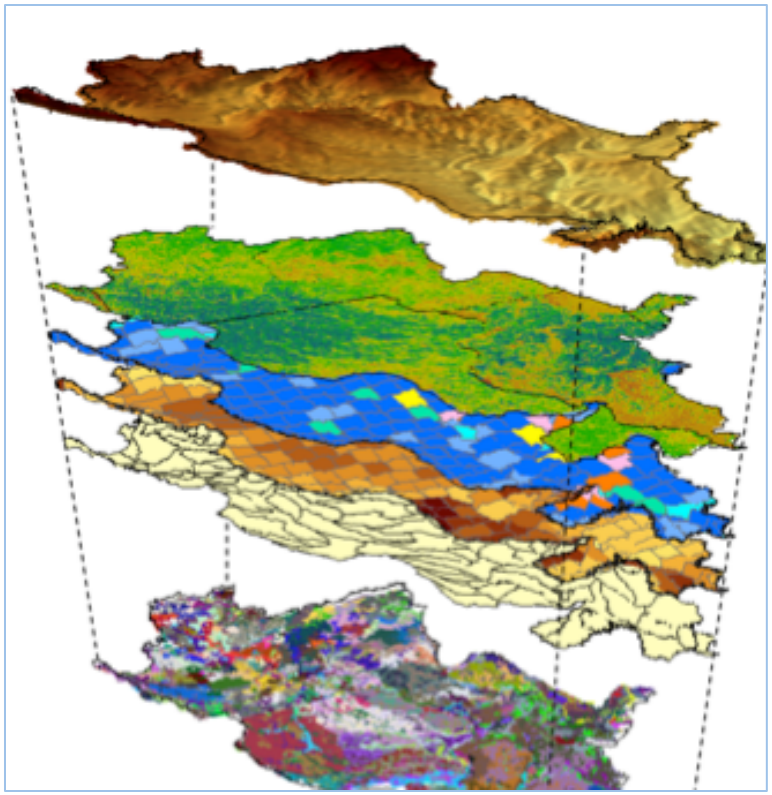
DSSAT models of corn yield.
Courtesy J. Elliott and K. Maheshwari

Modeling climate impact on hydrology
powered by *Swift*

Projecting biofuel production impact on hydrology (E. Yan)

This project studies the impact of global temperature increase on the Upper Mississippi River Basin on water and plant productivity. It is in the process of combining future climate data obtained from a statistically downscaled global circulation model (GCM) into the Upper Mississippi River Basin model. The results from these models will be used in the proposed study to evaluate the relative performance of the proposed coupling of climate and hydrology models.

Results of this research demonstrate that plausible changes in temperature and precipitation caused by increases in atmospheric greenhouse gas concentrations could have major impacts on both the timing and magnitude of runoff, soil moisture, water quality, water availability, and crop yield (including energy crops) in important agricultural areas.



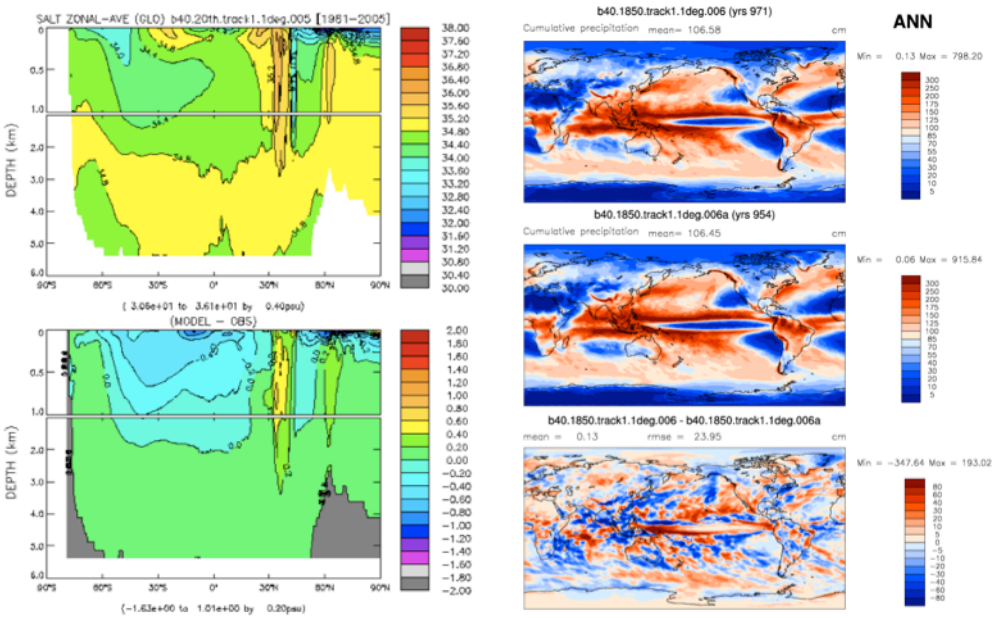
Visualization of multiple layers of SWAT hydrology model. Courtesy E. Yan.

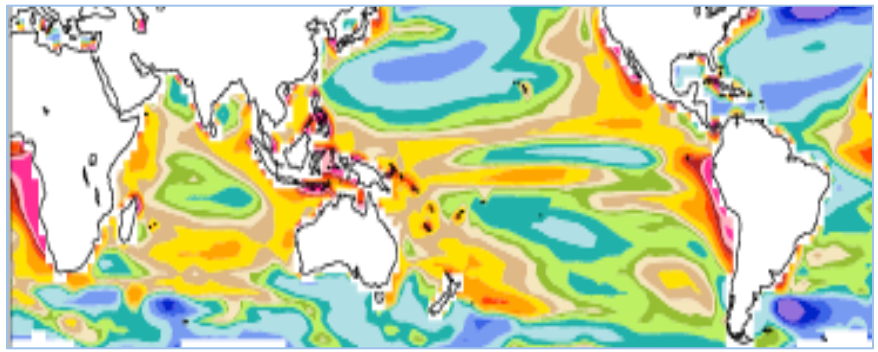
Benefit of implicit pervasive parallelism: Analysis & visualization of high-resolution climate models

powered by *Swift*

parVis

- Diagnostic scripts for each climate model (ocean, atmospehere, land, ice) were expressed in complex shell scripts
- Recoded in Swift, the CESM community has benefited from significant speedups and more modular scripts





Work of: J Dennis, M Woitasek, S Mickelson, R Jacob, M Vertenstein

Swift gratefully acknowledges support from:

U.S. DEPARTMENT OF ENERGY

THE UNIVERSITY OF CHICAGO

Argonne NATIONAL LABORATORY

http://swift-lang.org