Big Data Staging with MPI-IO for Interactive X-ray Science

Justin M Wozniak, Hemant Sharma, Timothy G. Armstrong, Michael Wilde, Jonathan D. Almer, Ian Foster

http://mcs.anl.gov/exm wozniak@mcs.anl.gov

Goal: Programmability for large scale analysis

- **Many-task computing**: Higher-level applications composed of many run-to-completion tasks: **input→compute→output**
  Message passing handled by our implementation details

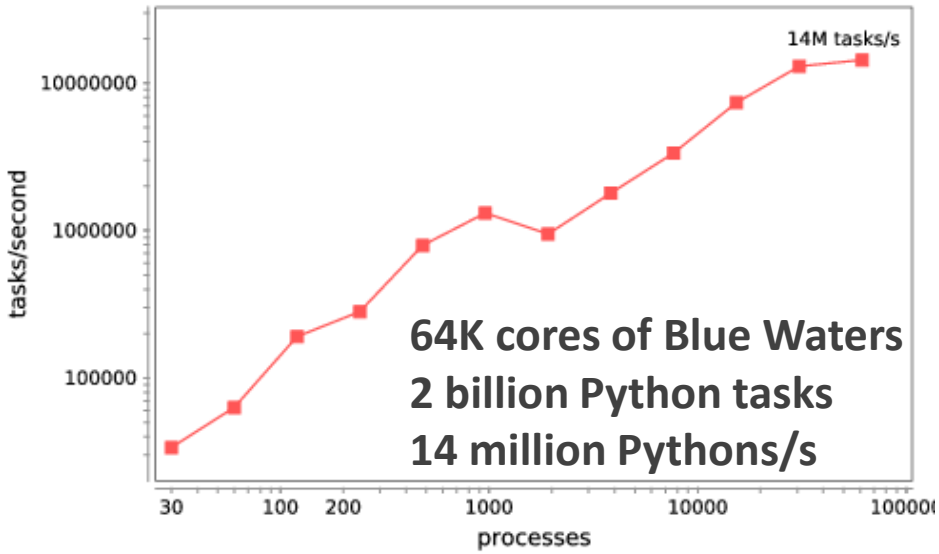
- **Programmability**
  - Large number of applications have this natural structure at upper levels: Parameter studies, ensembles, Monte Carlo, branch-and-bound, stochastic programming, UQ

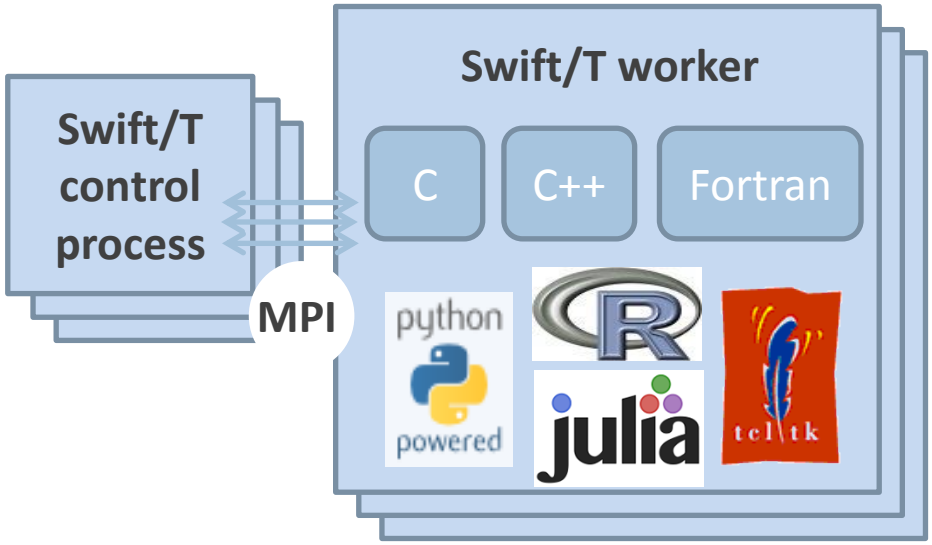
- **Data access optimizations**
  - Provide rich features for data-location-aware scheduling and **collective operations**

- **Experiment management**
  - Address workflow-scale issues: data transfer, application invocation, and metadata management

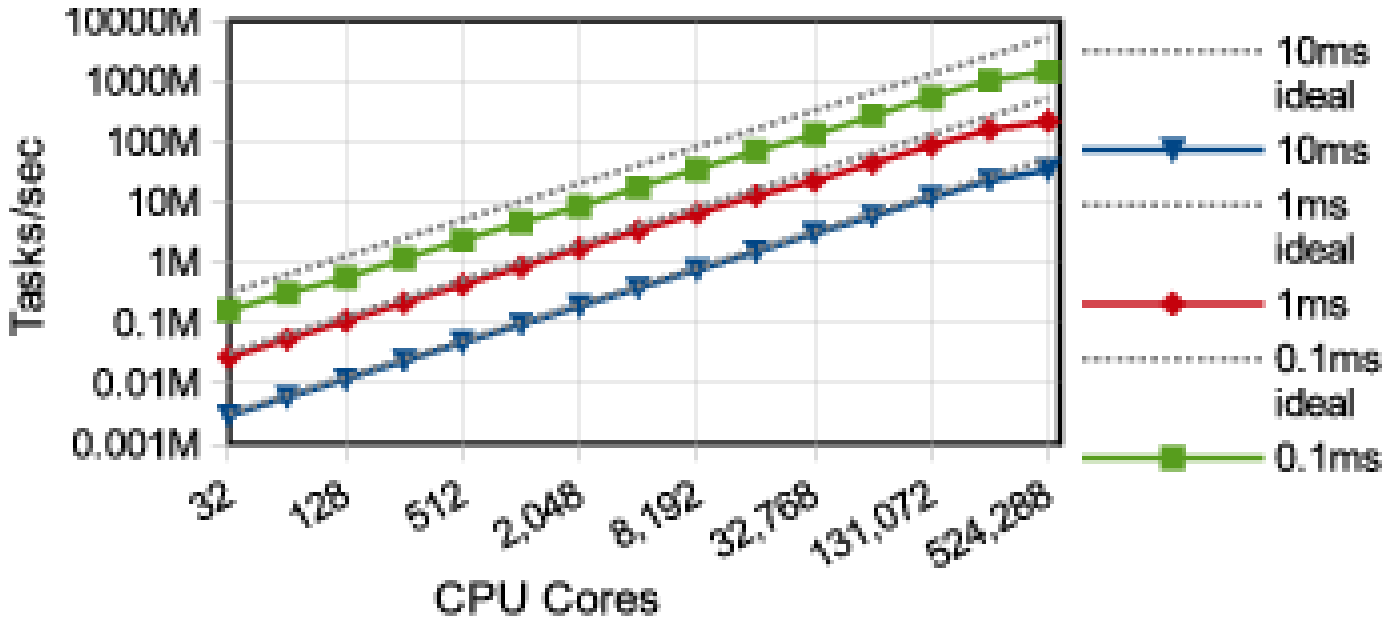
Swift/T: Enabling high-performance workflows

- Write site-independent scripts
- Automatic parallelization and data movement
- Run native code, script fragments as applications
- Rapidly subdivide large partitions for MPI jobs
- Move work to data locations





Basic scalability



- 1.5 billion tasks/s on 512K cores of Blue Waters, so far
- Armstrong et al. Compiler techniques for massively scalable implicit task parallelism. Proc. SC 2014.

Swift programming model: all progress driven by concurrent dataflow

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

- `F()` and `G()` implemented in native code
- `F()` and `G()` run in concurrently in different processes
- `r` is computed when they are both done

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

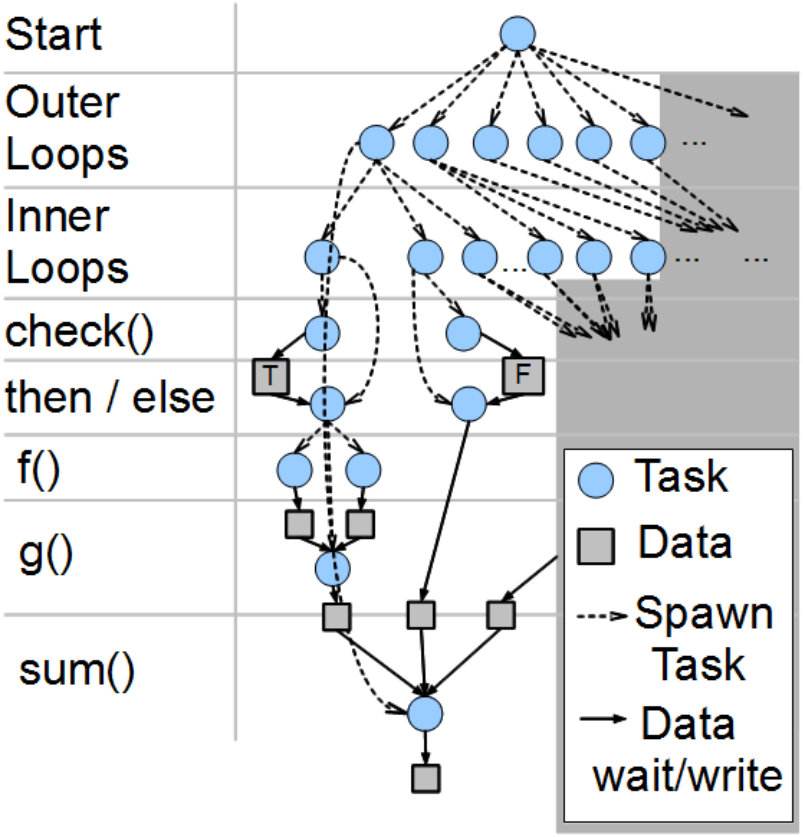
Characteristics of very large Swift programs

```
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]);
}
```

- The goal is to support billion-way concurrency: $O(10^9)$
- Swift script logic will control trillions of variables and data dependent tasks
- Need to distribute Swift logic processing over the HPC compute system

Swift/T: 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]);
}
```



- Wozniak et al. Large-scale application composition via distributed-memory data flow processing. Proc. CCGrid 2013.

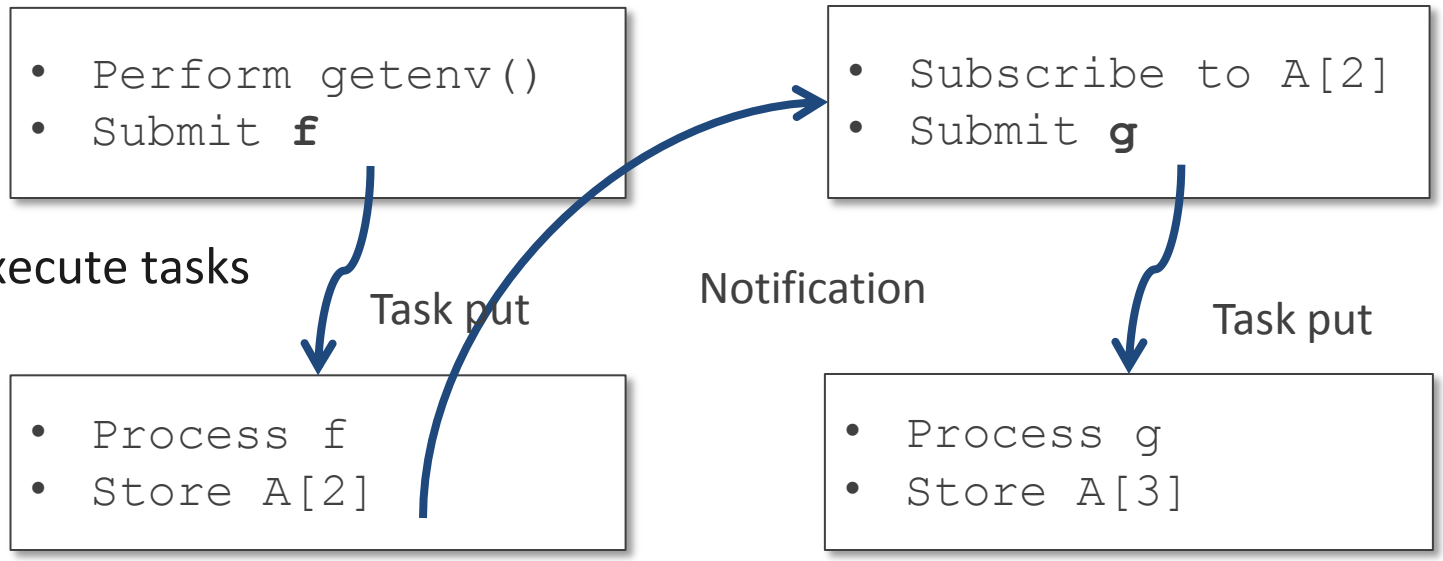
Example execution

- Code

```
A[2] = f(getenv("N"));
A[3] = g(A[2]);
```

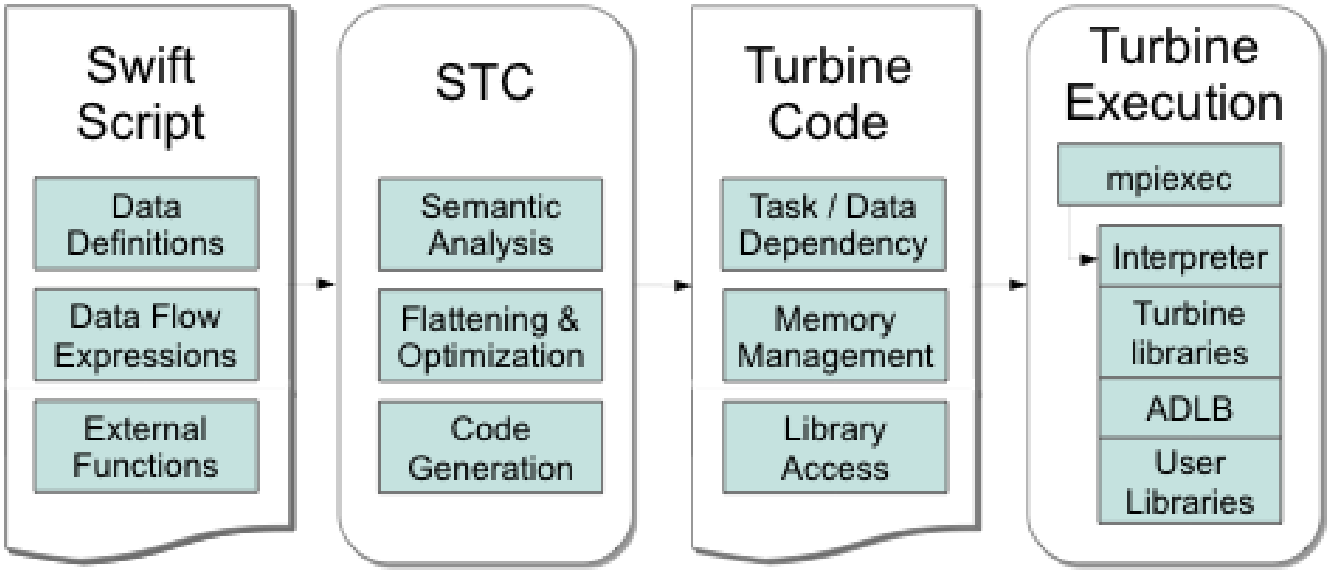
- Engines: evaluate dataflow operations

- Workers: execute tasks



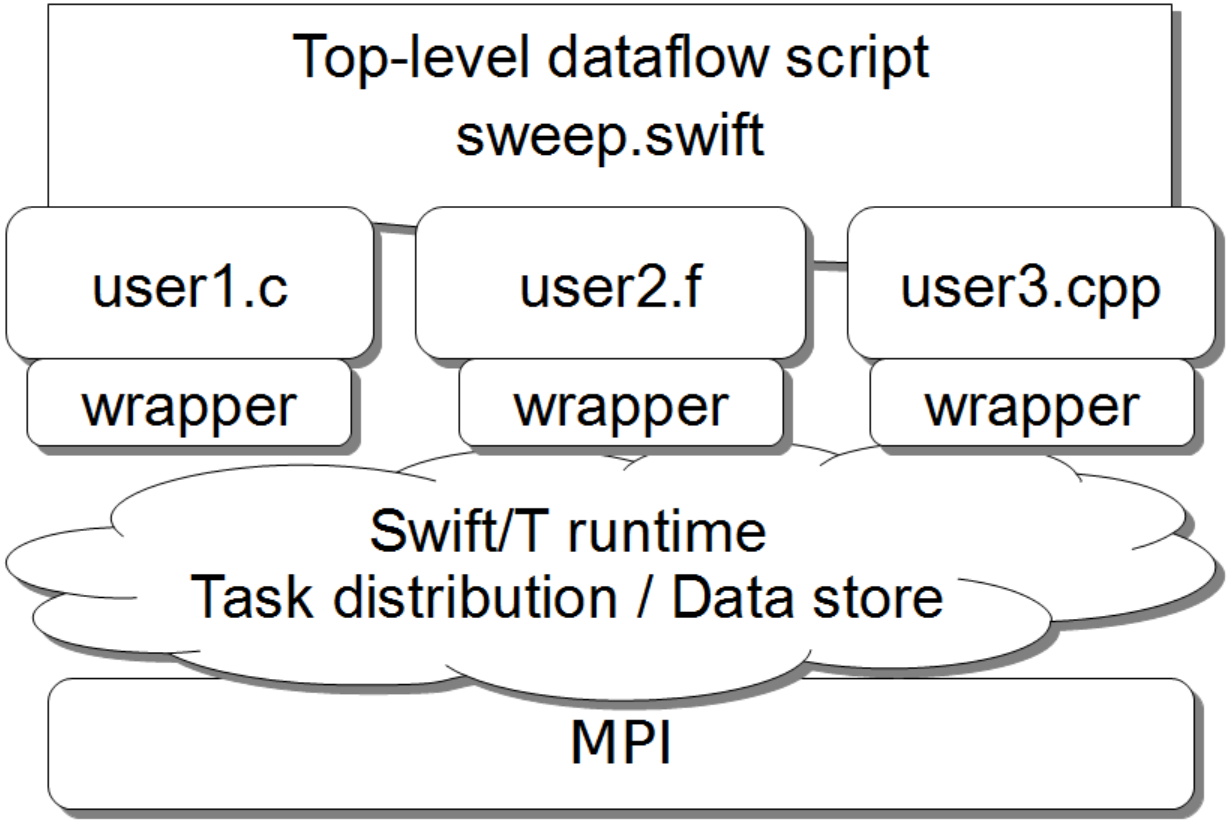
- Wozniak et al. Turbine: A distributed-memory dataflow engine for high performance many-task applications. Fundamenta Informaticae 128(3), 2013

STC: The Swift-Turbine Compiler



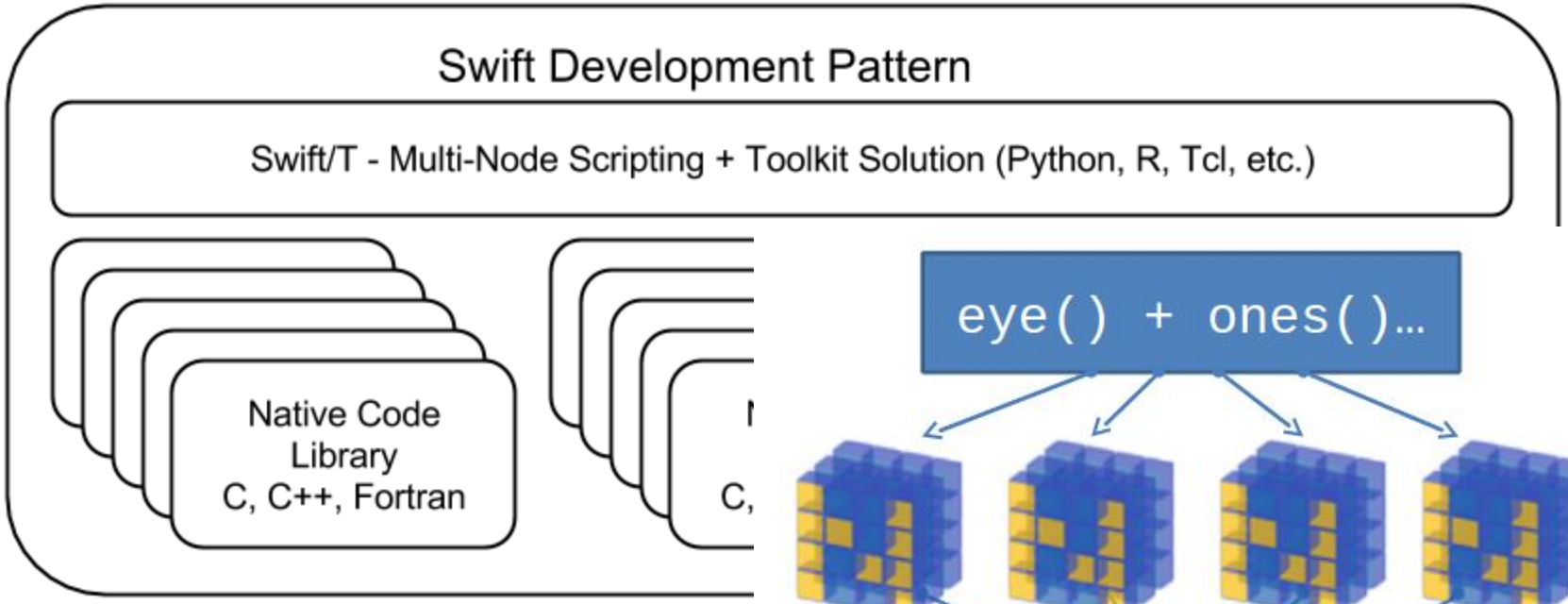
- STC translates high-level Swift expressions into low-level Turbine operations:
  - Create/Store/Retrieve typed data
  - Manage arrays
  - Manage data-dependent tasks

Support calls to native libraries



- Including MPI libraries

Support calls to embedded interpreters

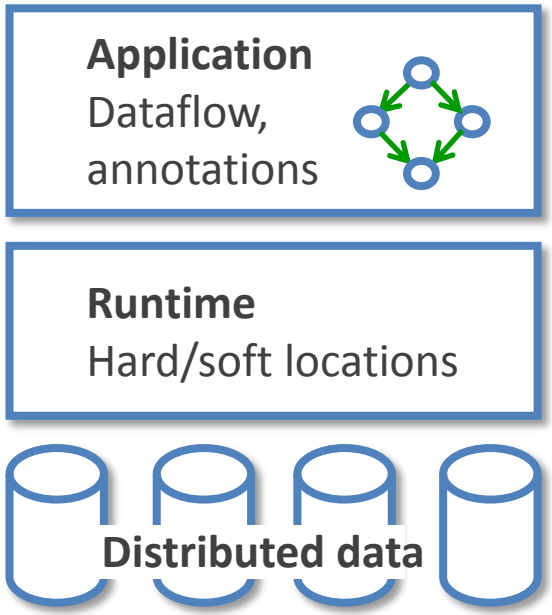


We have plugins for Python, R, Tcl, Julia, and QtScript

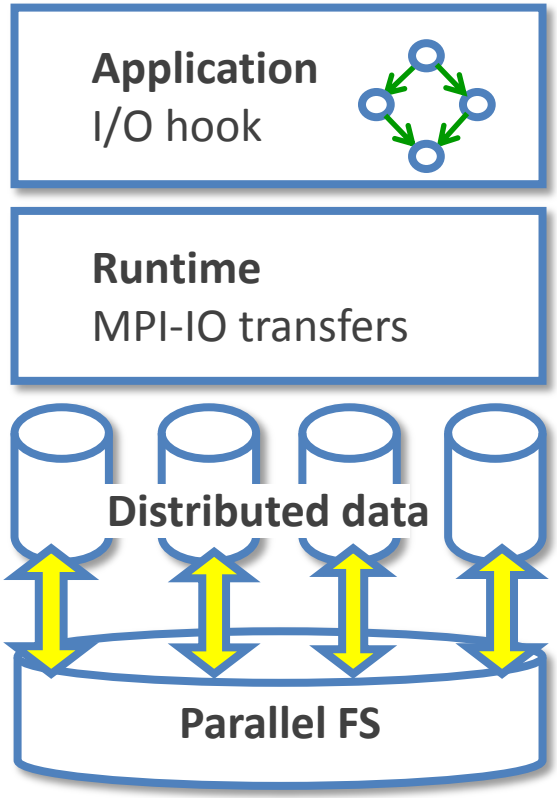
- Wozniak et al. Toward computational experiment management via multi-language applications. Proc. ASCR SWP4XS, 2014.

Features for Big Data analysis

- **Location-aware scheduling**
  User and runtime coordinate data/task locations



- **Collective I/O**
  User and runtime coordinate data/task locations

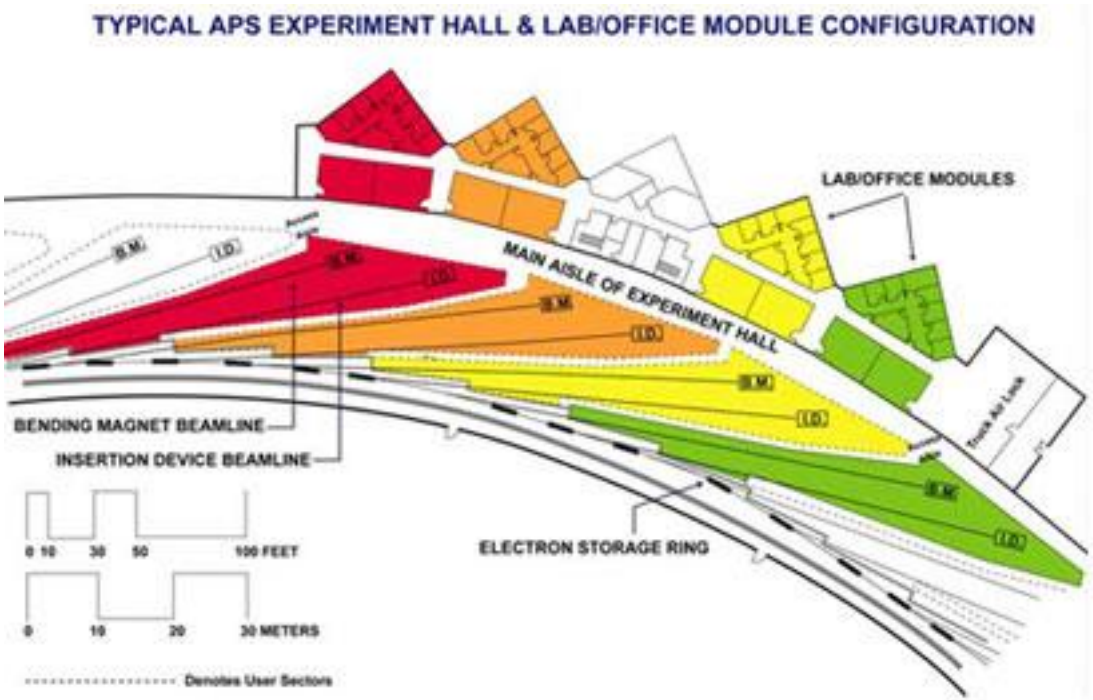


- F. Duro et al. Exploiting data locality in Swift/T workflows using Hercules . Proc. NESUS Workshop, 2014.


Advanced Photon Source (APS)

Advanced Photon Source (APS)

- Moves electrons at electrons at >99.999999% of the speed of light.
- Magnets bend electron trajectories, producing x-rays, highly focused onto a small area
- X-rays strike targets in 35 different laboratories – each a lead-lined, radiation-proof experiment station



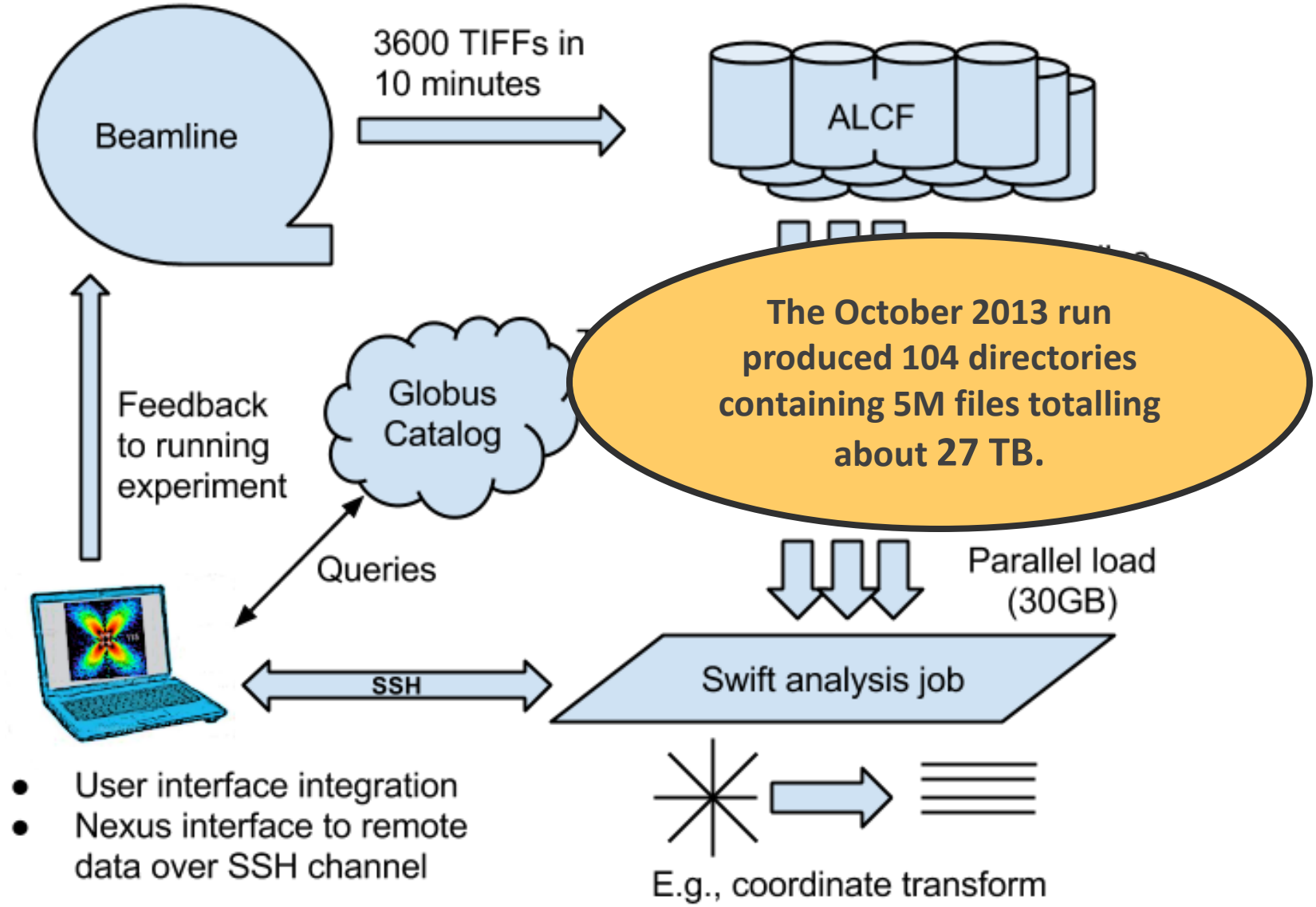
Data management for the energy sciences

- "Despite the central role of **digital data** in Dept. of Energy (DOE) research, the methods used to manage these data and to support the information and **collaboration processes** that underpin DOE research are often **surprisingly primitive**…"

  - DOE Workshop Report on Scientific Collaborations (2011)

- Our goals:
  - Modify the operating systems of APS stations to allow real-time streaming to a novel data storage/analysis platform.
  - Converting data from the standard detector formats (usually TIFF) to HDF5 and adding metadata and provenance, based on the NeXus data format.
  - Rewrite analysis operations to work in a massively parallel environment.
  - Scale up simulation codes that complement analysis.

Data ingest/analysis/archive



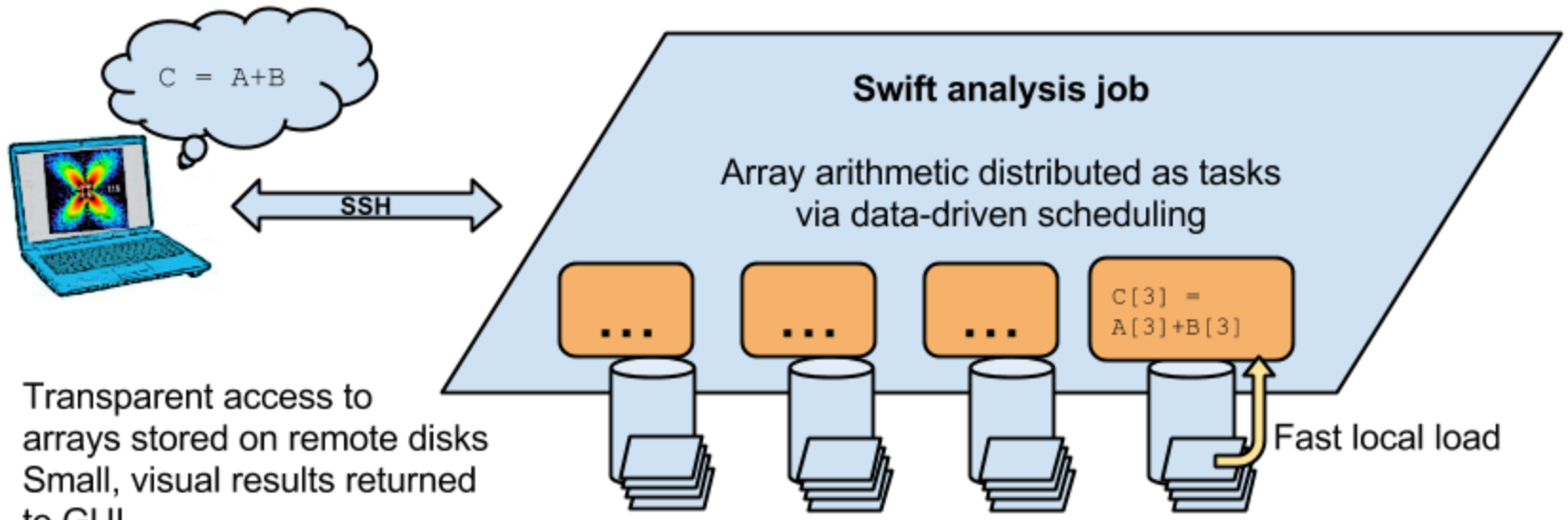
The October 2013 run produced 104 directories containing 5M files totalling about 27 TB.

- User interface integration
- Nexus interface to remote data over SSH channel

PADS: Petascale Active Data Store

- 23 higher-end nodes for data-intensive computing, repurposed for this work (installed in 2009)
  - Each node has 12-way RAID for very fast local disk operations
- Previously, difficult to use as "Active Data Store"
  - Difficult to access specific nodes through PBS scheduler
  - No catalog (where is my data?)
  - *No way to organize/access Data Store!*
- Solution: Swift/T
  - Organizes distributed data using Swift data structures and mappers
  - Leaves data on nodes for later access
  - Allows for targeted tasks (can send work to node with data chunk)
  - Integrates with Globus Catalog for metadata, provenance, archive...
  - Combining unscheduled resource access with high performance data rates will allow for **real-time beamline data analysis, accelerating progress for materials science efforts**

Interactive analysis powered by scalable storage



- Transparent access to arrays stored on remote disks
- Small, visual results returned to GUI
- Bulk data **stays on PADS**

- Replace GUI analysis internals with operations on remote data

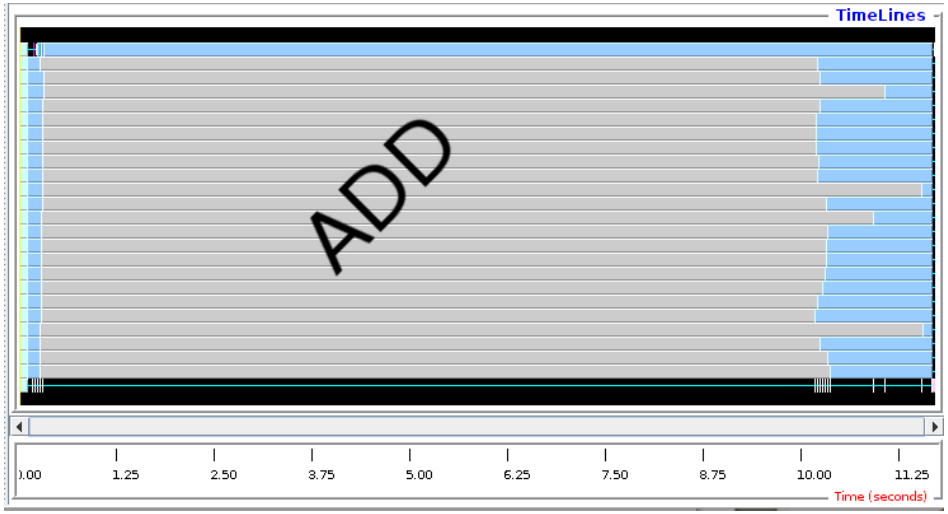
Example Remote Numpy Operations

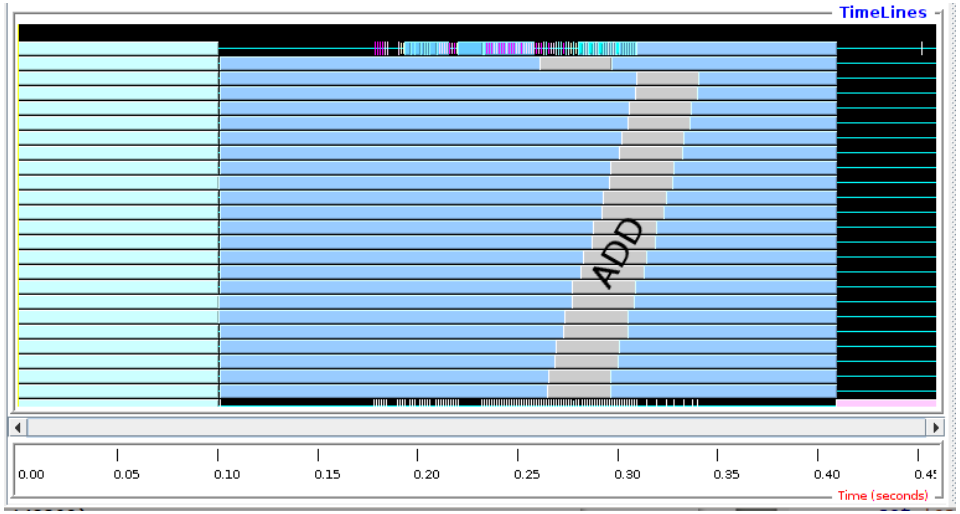
```
with NXFileRemote("tukey.alcf.anl.gov",
                  "sample123.nexus") as nxfr:

    # Step through NeXus metadata:
    # Obtain the top-level entry:
    f1 = nxfr["/entry"]
    # Obtain the data entry:
    f2 = f1["data"]
    # Obtain the 3D bulk data variable:
    v = nxfr["/entry/data/v"]
    # Obtain a slice of the variable (a plane)
    v = f[0,0]
    # Obtain a single element in the variable:
    v = f[0,0,0]
    # Do all of this in one stroke:
    v = nxfr["/entry/data/v"][0,0,0]
```

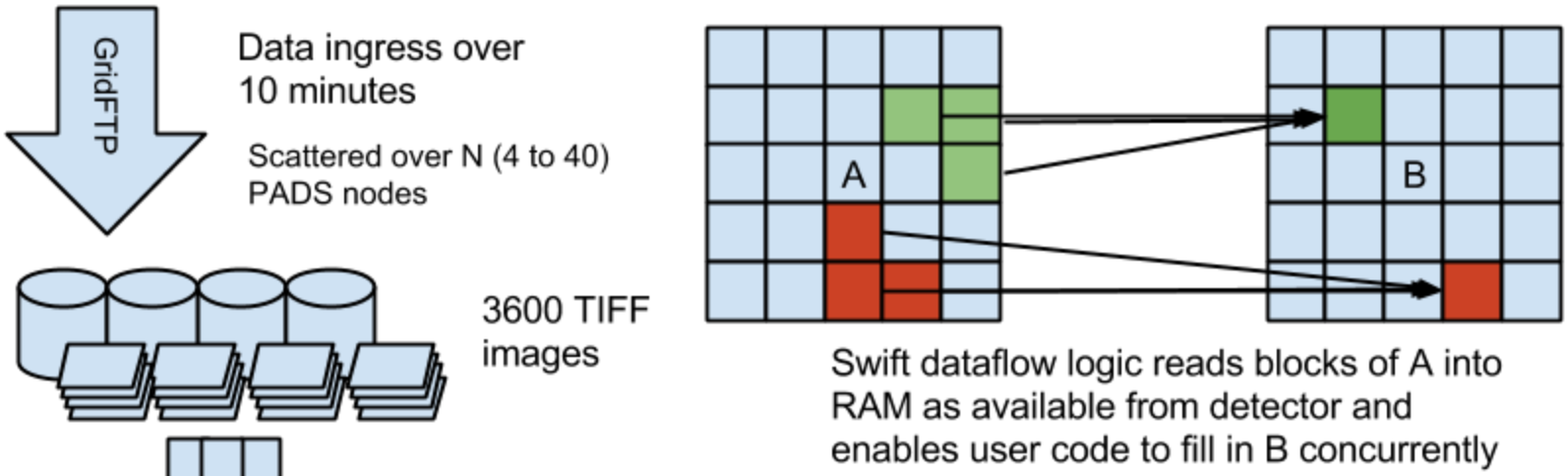
Remote matrix arithmetic: Initial results

- Initial run shows performance issue: addition took too long
- Swift profiling isolated issue: convert addition routine from script to C function: obtained 10,000 X speedup
- Swift/T integrates with MPE/Jumpshot and other MPI-based performance analysis techniques





Crystal Coordinate Transformation Workflow

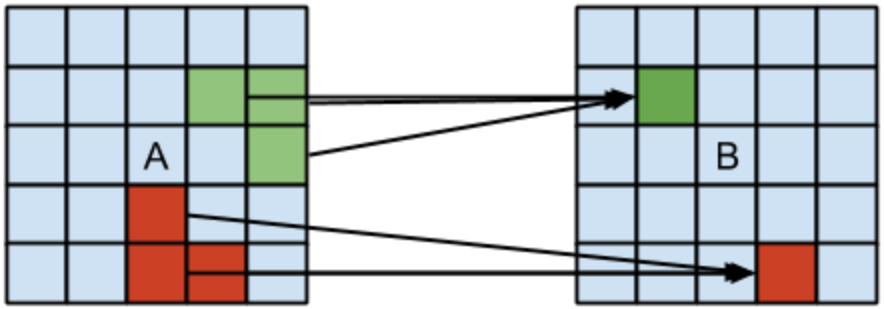


Swift/T execution:

1. Concurrent read of all TIFFs into Swift memory (blobs) as 3D input matrix
2. For each output cell, requisite input cells are retrieved from Swift memory
3. Output cells are concurrently computed and written to GPFS for visualization

CCTW: Swift/T application (C++)

```
bag<blob> M[];
foreach i in [1:n] {
    blob b1= cctw_input("pznpt.nxs");
    blob b2[];
    int outputId[];
    (outputId, b2) = cctw_transform(i, b1);
    foreach b, j in b2 {
        int slot = outputId[j];
        M[slot] += b;
    }}
    foreach g in M {
        blob b = cctw_merge(g);
        cctw_write(b);
    }}
```

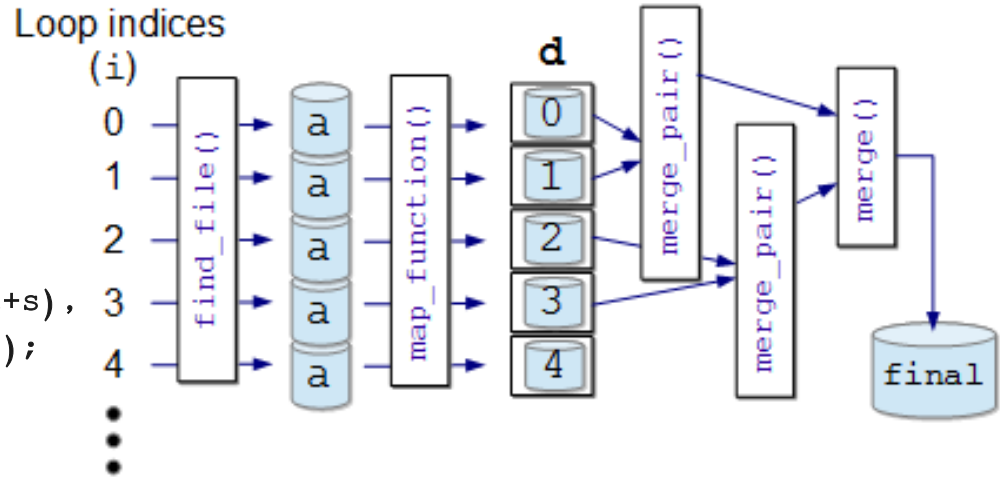


Abstract, extensible MapReduce in Swift

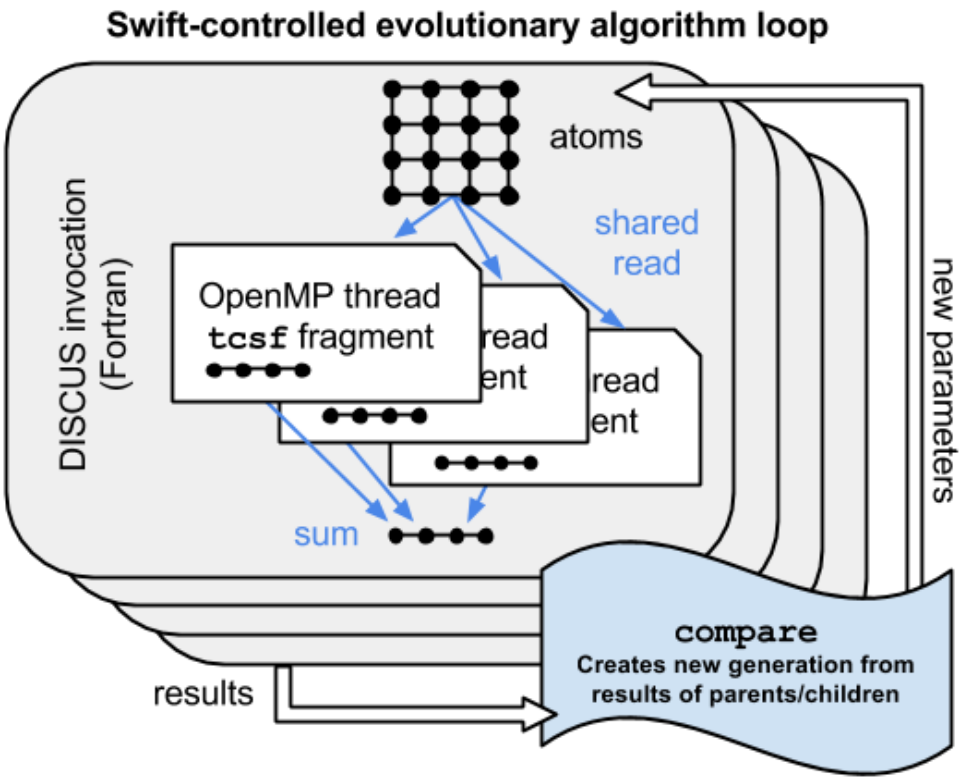
```
main {
  file d[];
  int N = string2int(argv("N"));
  // Map phase
  foreach i in [0:N-1] {
    file a = find_file(i);
    d[i] = map_function(a);
  }
  // Reduce phase
  file final <"final.data"> = merge(d, 0, tasks-1);
}

(file o) merge(file d[], int start, int stop) {
  if (stop-start == 1) {
    // Base case: merge pair
    o = merge_pair(d[start], d[stop]);
  } else {
    // Merge pair of recursive calls
    n = stop-start;
    s = n % 2;
    o = merge_pair(merge(d, start,     start+s),
                   merge(d, start+s+1, stop));
  }}
```

- User needs to implement `map_function()` and `merge()`
- These may be implemented in native code, Python, etc.

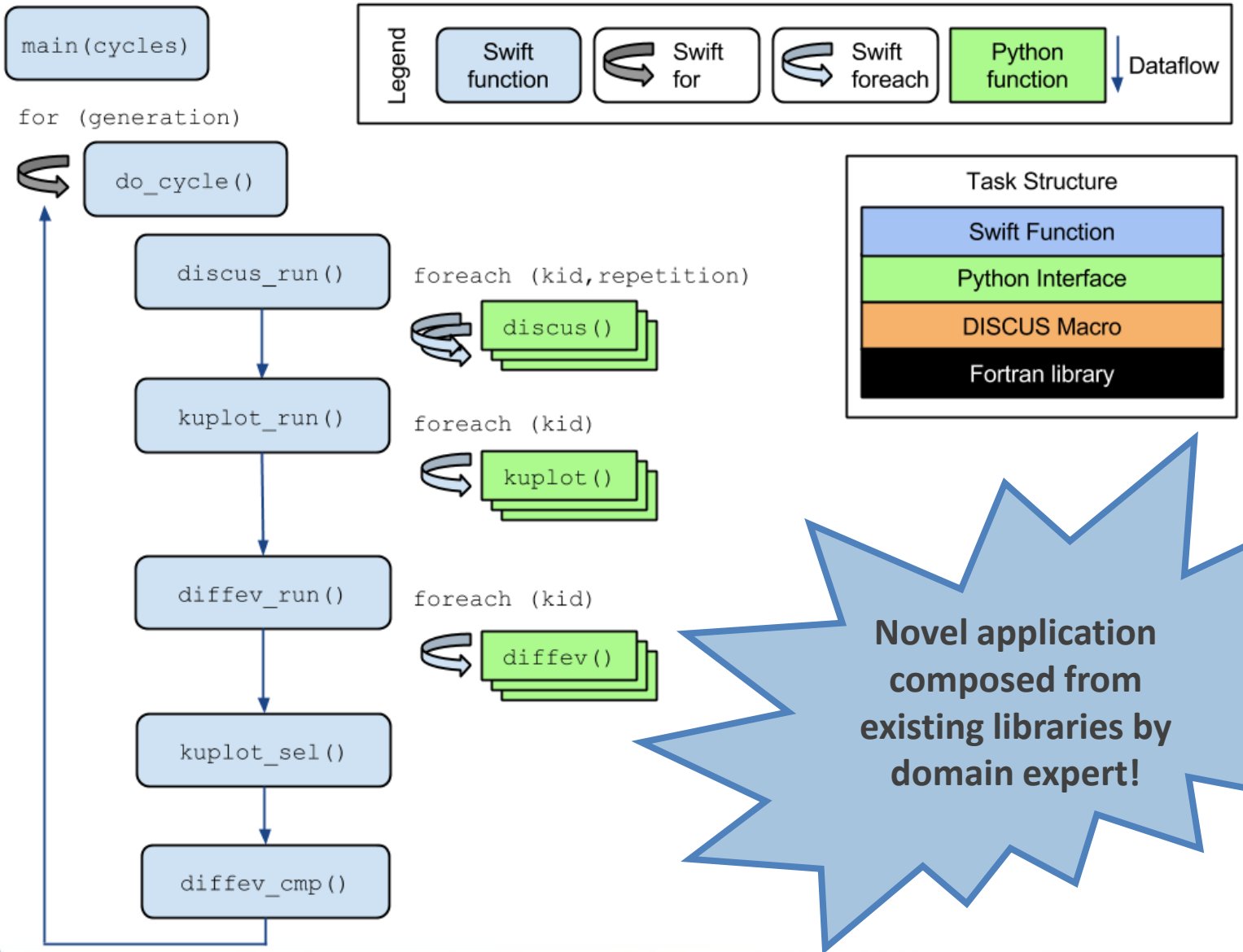


DIFFEV: Scaling crystal diffraction simulation



- Determines crystal configuration that produced given scattering image through simulation and evolutionary algorithm
- Swift/T calls DISCUS via Python interfaces

DIFFEV: Genetic algorithm via dataflow



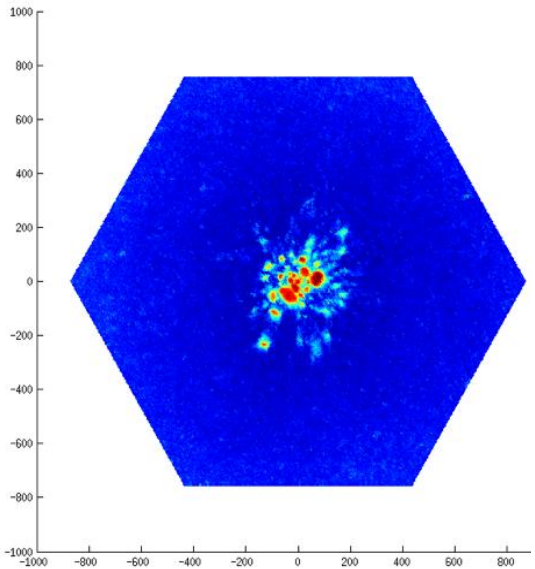
Novel application composed from existing libraries by domain expert!

Real-time beamline analysis

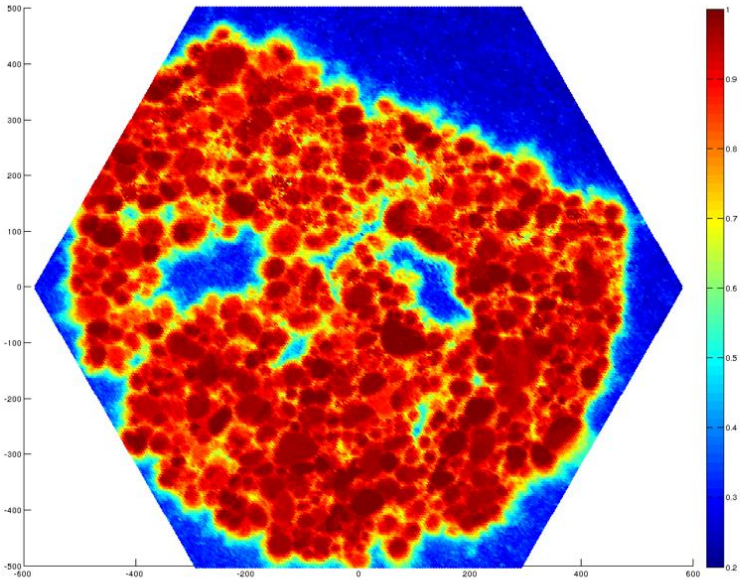
- Goal: Transfer data from APS to HPC while experiment is running
- Use ALCF *Mira*, an IBM Blue Gene/Q: 786,432 cores @ 10 PF

- Challenges
  - Data transfer (15 TB / week or more)
  - Co-scheduling HPC time with beam time
  - Rapidly scaling existing prototypical analysis codes to ~100K cores
  - Staging experimental data (577 MB) from GPFS to the compute nodes
- Can use 22 M CPU hours / week!

High-Energy Diffraction Microscopy



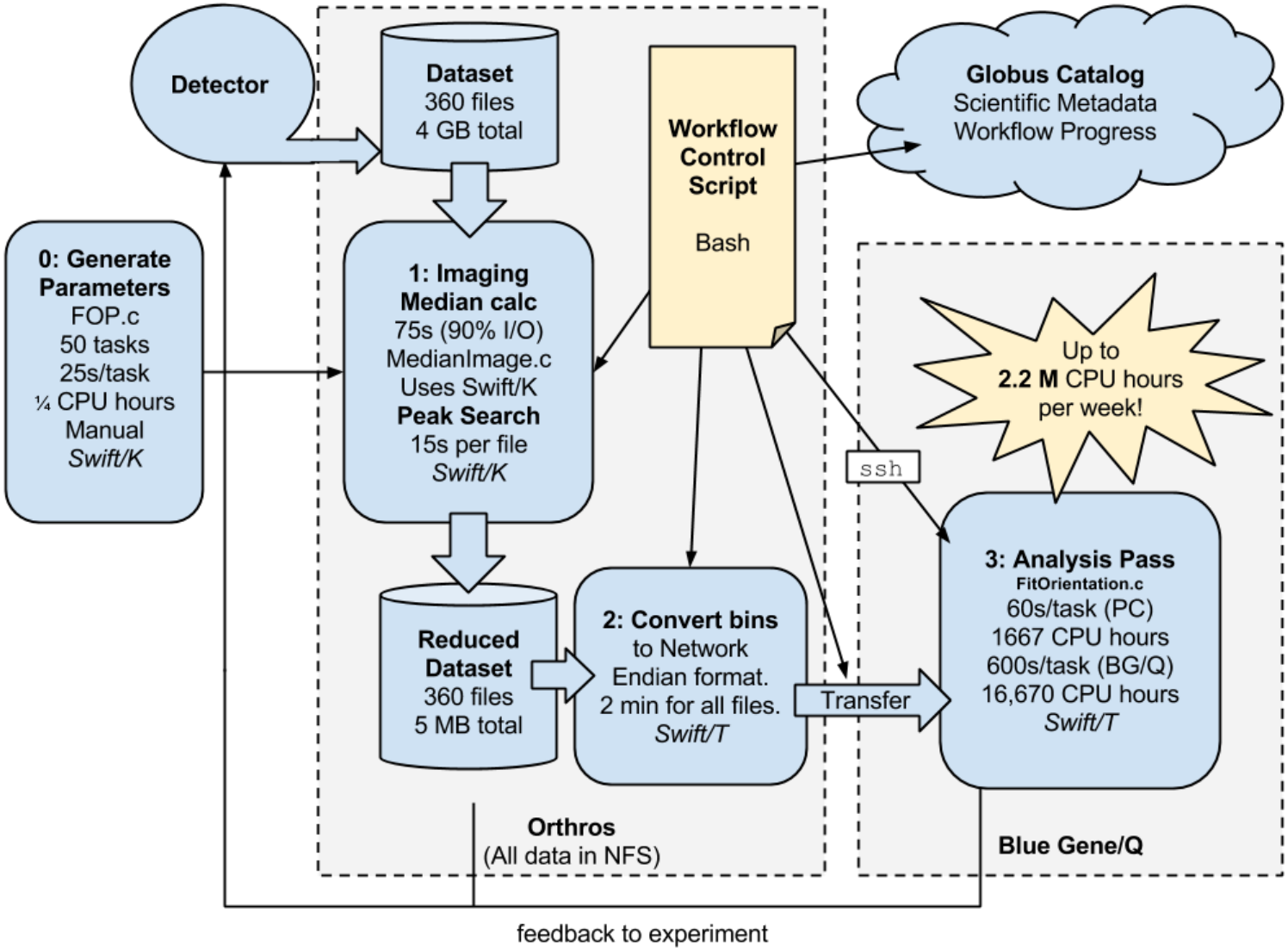
October 2013: Without Swift



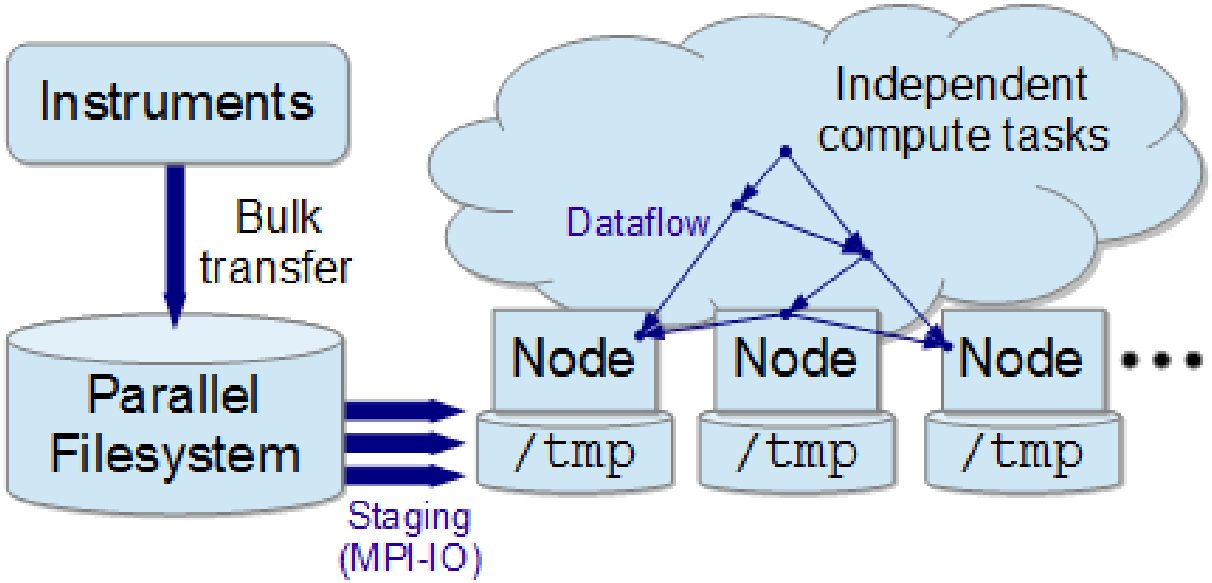
April 2014: With Swift

- Near-field high-energy diffraction microscopy discovers metal grain shapes and structures
- The experimental results are greatly improved with the application of Swift-based cluster computing (**RED** indicates higher confidence in results)

NF-HEDM

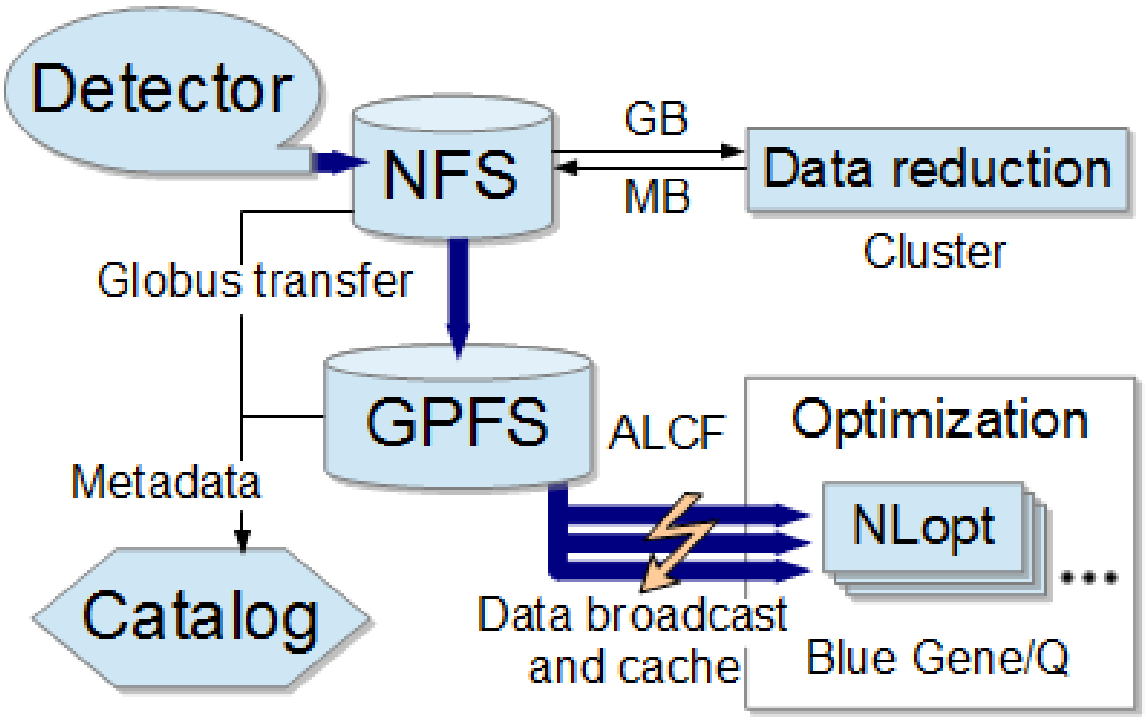

Detector
Dataset
360 files
4 GB total
Workflow
Control
Script
Bash
Globus Catalog
Scientific Metadata
Workflow Progress
0: Generate
Parameters
FOP.c
50 tasks
25s/task
¼ CPU hours
Manual
Swift/K
1: Imaging
Median calc
75s (90% I/O)
MedianImage.c
Uses Swift/K
Peak Search
15s per file
Swift/K
Up to
2.2 M CPU hours
per week!
ssh
Reduced
Dataset
360 files
5 MB total
2: Convert bins
to Network
Endian format.
2 min for all files.
Swift/T
Transfer
3: Analysis Pass
FitOrientation.c
60s/task (PC)
1667 CPU hours
600s/task (BG/Q)
16,670 CPU hours
Swift/T
Orthros
(All data in NFS)
Blue Gene/Q
feedback to experiment

Task-based HPC



- Existing C code (NFHEDM) assembled into scalable HPC program with Swift/T
- Problem: Each task must consume ~500 MB of experimental data – each task does uncoordinated I/O

Intended use of broadcast operation



- Grain orientation optimization workflow runs on BG/Q once data is there
- Each task needs to read all input from a given dataset
- Desire to use MPI-IO before running tasks

NF-HEDM application Swift code

- **Swift dataflow**

```
main {
  parameterFile = argv("p");
  microstructureFile = argv("m");
  start = toint(argp(1));
  end   = toint(argp(2));
  foreach row in [start:end] {
    FitOrientation(parameterFile, row,
                   microstructureFile);
  }
}
```

- **FitOrientation()** is linked to a C function
- Each task reads the same data
- Output is inserted into the microstructure file

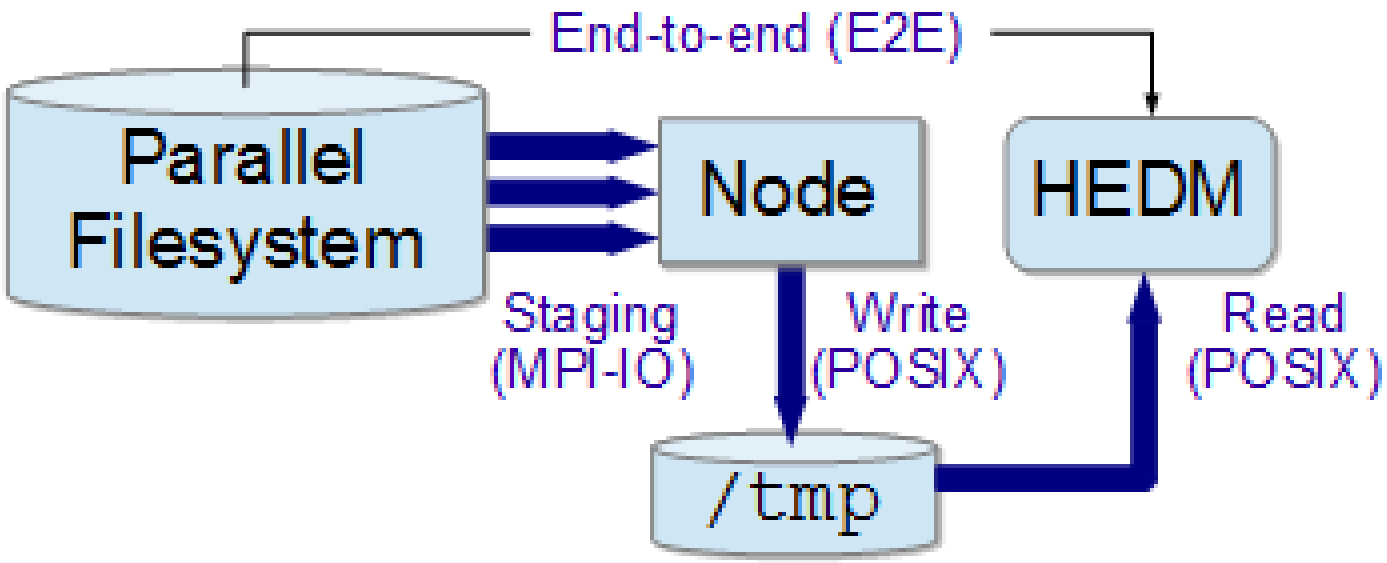
- **Swift I/O hook specification**

```
broadcast to /tmp files {
  ~/dataset-1/*.cfg
}

broadcast to /tmp/bulk files {
    ~/dataset-1/bulk/file1.index
    ~/dataset-1/bulk/file2.index
    ~/dataset-1/bulk/*.bin
}
```

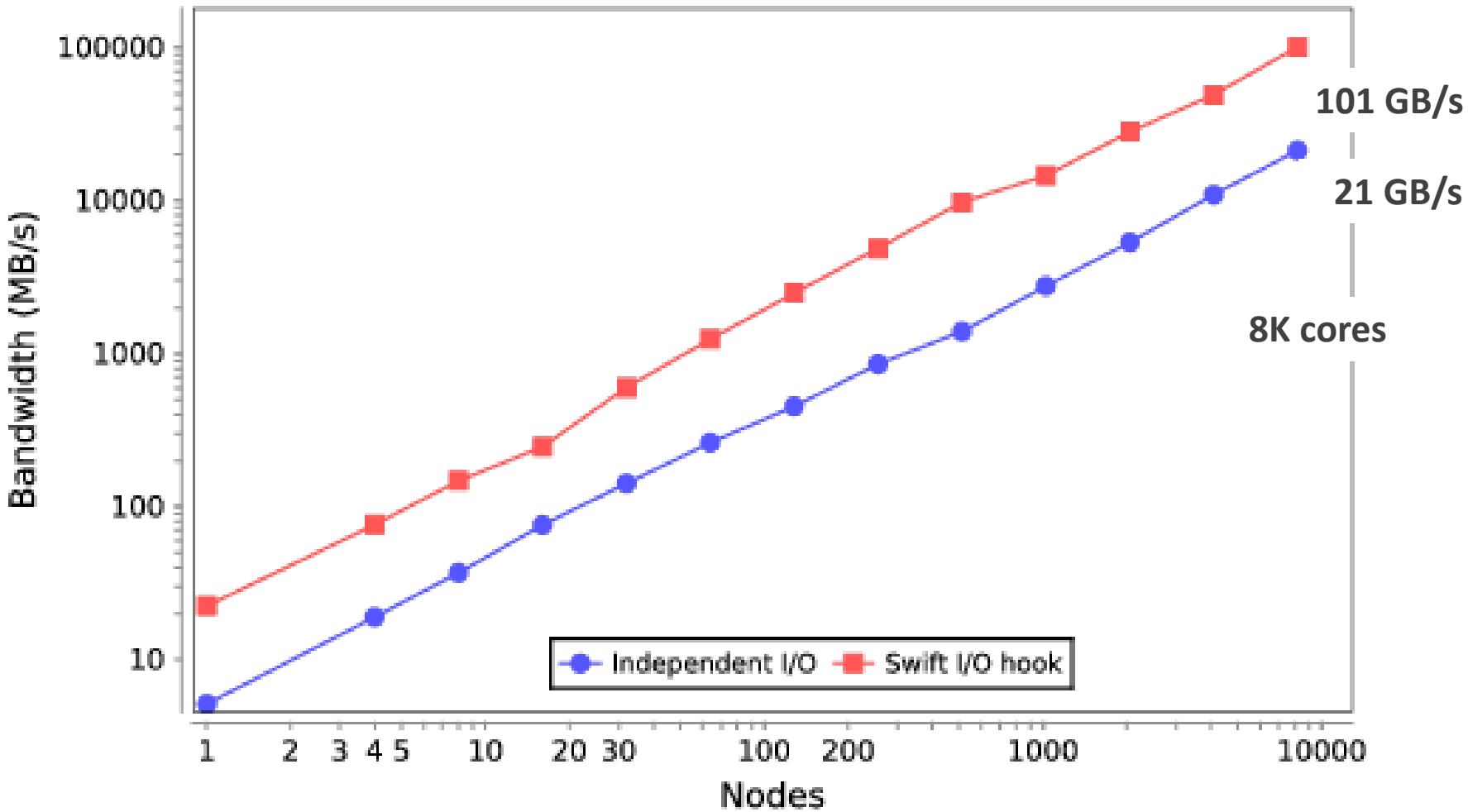
- Executed by Swift at startup

Big Data Staging with MPI-IO

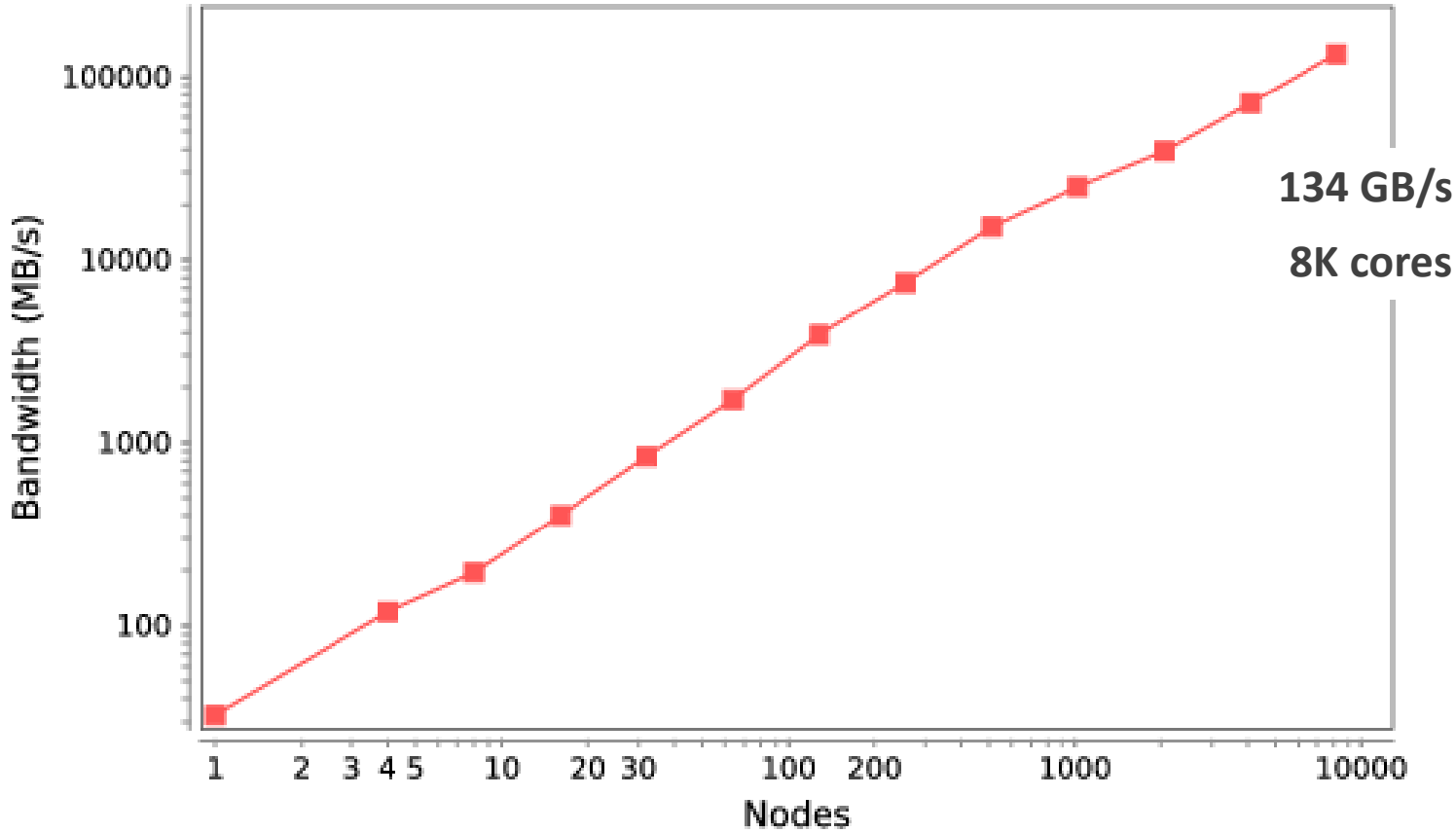


- Solution: Broadcast experimental data on HPC system with MPI-IO
- Tasks consume data normally from node-local storage

Scalability result: End-to-end



Scalability result: Stage+Write



- This plot breaks I/O hook into 1) stage+write and 2) read phases
- Read phase is node-local: consistently 10.8 ±0.1 s

Big Data Staging: Conclusions

- **Blue Gene/Q can be used for big data problems and a many-task programming model**
  - Just broadcast the data to compute nodes first with MPI-IO
- **The Swift I/O hook enables efficient I/O in a many-task model**
  - Reduces I/O time by factor of 4.7!
- **Connecting HPC to a real-time experiment saved an experiment by detecting a loose cable**
- **Code is now being reused by about 5 different groups**
  - Now must accommodate extra users on HPC resources!

Summary

- **Swift:** High-level scripting for outermost programming constructs
- Described features for **big data computing** on clusters and supercomputers

- Thanks to the organizers
- Thanks to the Swift team
- Thanks to application collaborators

- **Questions?**