Streaming, Storing, and Sharing Big Data for Light Source Science

Justin M Wozniak <wozniak@mcs.anl.gov>

Kyle Chard, Ben Blaiszik, Michael Wilde, Ian Foster

Argonne National Laboratory

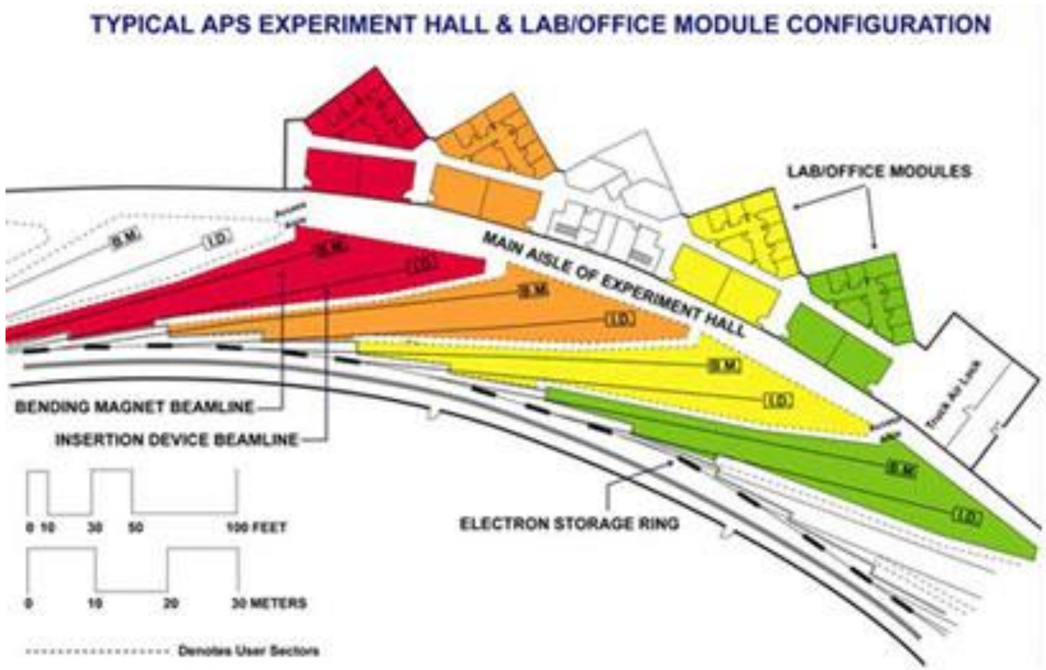
At STREAM 2015

Oct. 27, 2015


Chicago
Supercomputers
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
- Scattering detectors produce images containing experimental results



Distance from Top Light Sources to Top Supercomputer Centers

Light Source	Distance to Top10 Machine
SIRIUS, Brazil	> 5000Km, TACC, USA
BAP, China	2000Km, Tihane-2, China
MAX, Sweden	800Km, Jülich Germany
PETRA III, Germany	500Km, Jülich Germany
ESRF, France	400Km, Lugano, Switzerland
Spring 8, Japan	100Km, K-Machine, Kobe, Japan
APS, IL, USA	~1Km, ALCF & MCS*, ANL, USA

*ANL Computing Divisions
ALCF: Argonne Leadership Computing Facility
MCS: Mathematics & Computer Science

Proximity means we can closely couple computing in novel ways

Terabits/s in the near future

Petabits/s are possible

Goals and tools

TALK OVERVIEW

Goals

- **Automated data capture and analysis pipelines**
  To boost productivity during beamtime
- **Integration with high-performance computers**
  To integrate experiment and simulation
- **Effective use of large data sets**
  Maximize utility of high-resolution, high-frame-rate detectors and automation
- **High interactivity and programmability**
  Improve the overall scientific process

Tools

- **Swift**
  Workflow language with very high scalability
- **Globus Catalog**
  Annotation system for distributed data
- **Globus Transfer**
  Parallel data movement system
- **NeXpy/NXFS**
  GUI with connectivity to Catalog and Python remote object services

High performance workflows

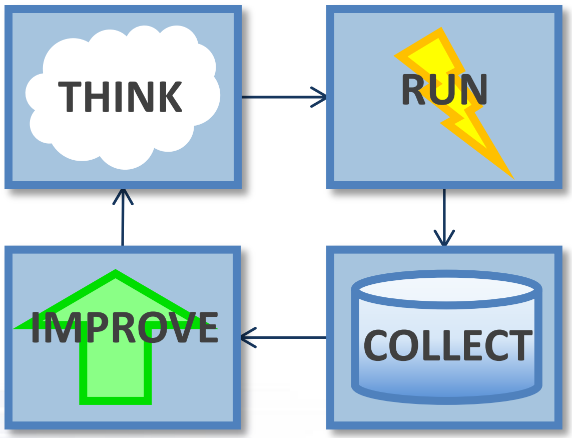
SWIFT

Goals of the Swift language

Swift was designed to handle many aspects of the computing campaign

- Ability to integrate **many application components** into a new workflow application
- Data structures for complex data organization
- Portability- separate site-specific configuration from application logic
- Logging, provenance, and plotting features

- **Today,** we will focus on running scripted applications on large streaming data sets



Swift programming model:
All progress driven by concurrent dataflow

```
(int r) myproc (int i, int j)
{
    int x = A(i);
    int y = B(j);
    r = x + y;
}
```

- `A()` and `B()` implemented in native code
- `A()` and `B()` 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

Swift programming model

- Data types

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

- Mapped data types

```
file image<"snapshot.jpg">;
```

- Structured data

```
image  A[]<array_mapper…>;
type protein {
   file pdb;
   file docking_pocket;
}
protein p<ext; exec=protein.map>;
```

- Conventional expressions

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

- Parallel loops

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

- Data flow

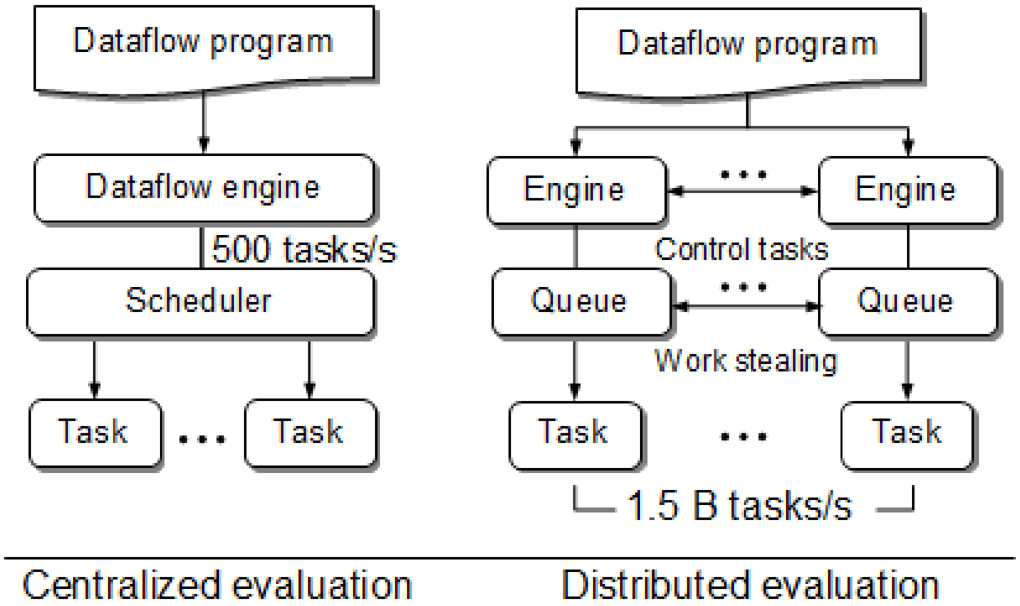
```
merge(analyze(B[0], B[1]),
        analyze(B[2], B[3]));
```

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

Swift/T: Distributed dataflow processing

Had this:
(Swift/K)

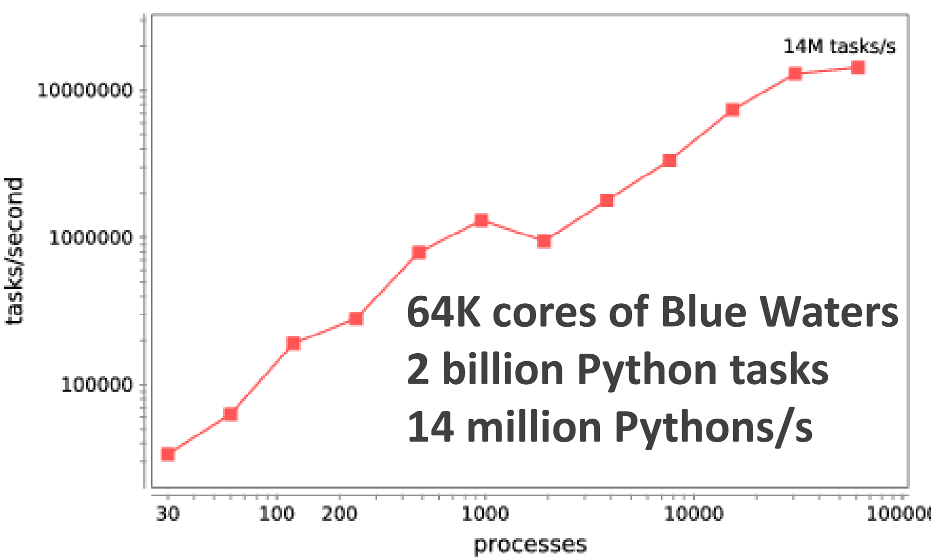
For extreme scale,
we need this:
(Swift/T)

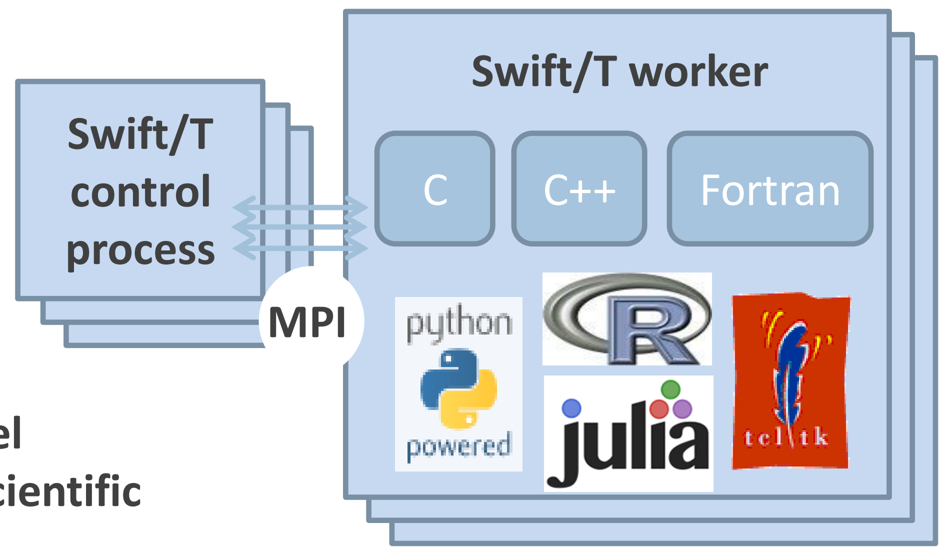


- Armstrong et al. **Compiler techniques for massively scalable implicit task parallelism.** Proc. SC 2014.
- Wozniak et al. **Swift/T: Scalable data flow programming for distributed-memory task-parallel applications .** Proc. CCGrid, 2013.

Swift/T: Enabling high-performance workflows

- Write site-independent scripts
- Automatic parallelization and data movement
- Run native code, script fragments as applications

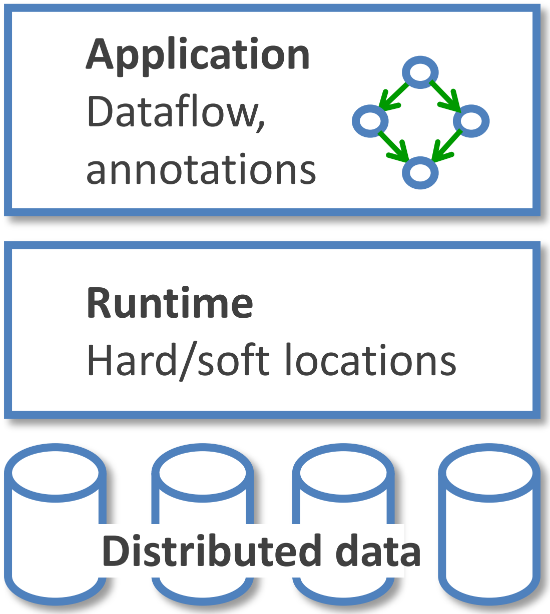




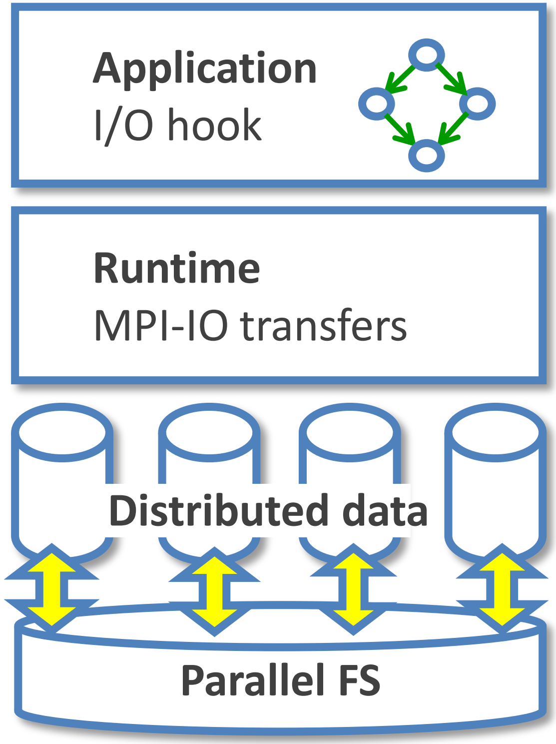
- Wozniak et al. **Interlanguage parallel scripting for distributed-memory scientific computing.** Proc. WORKS 2015.

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



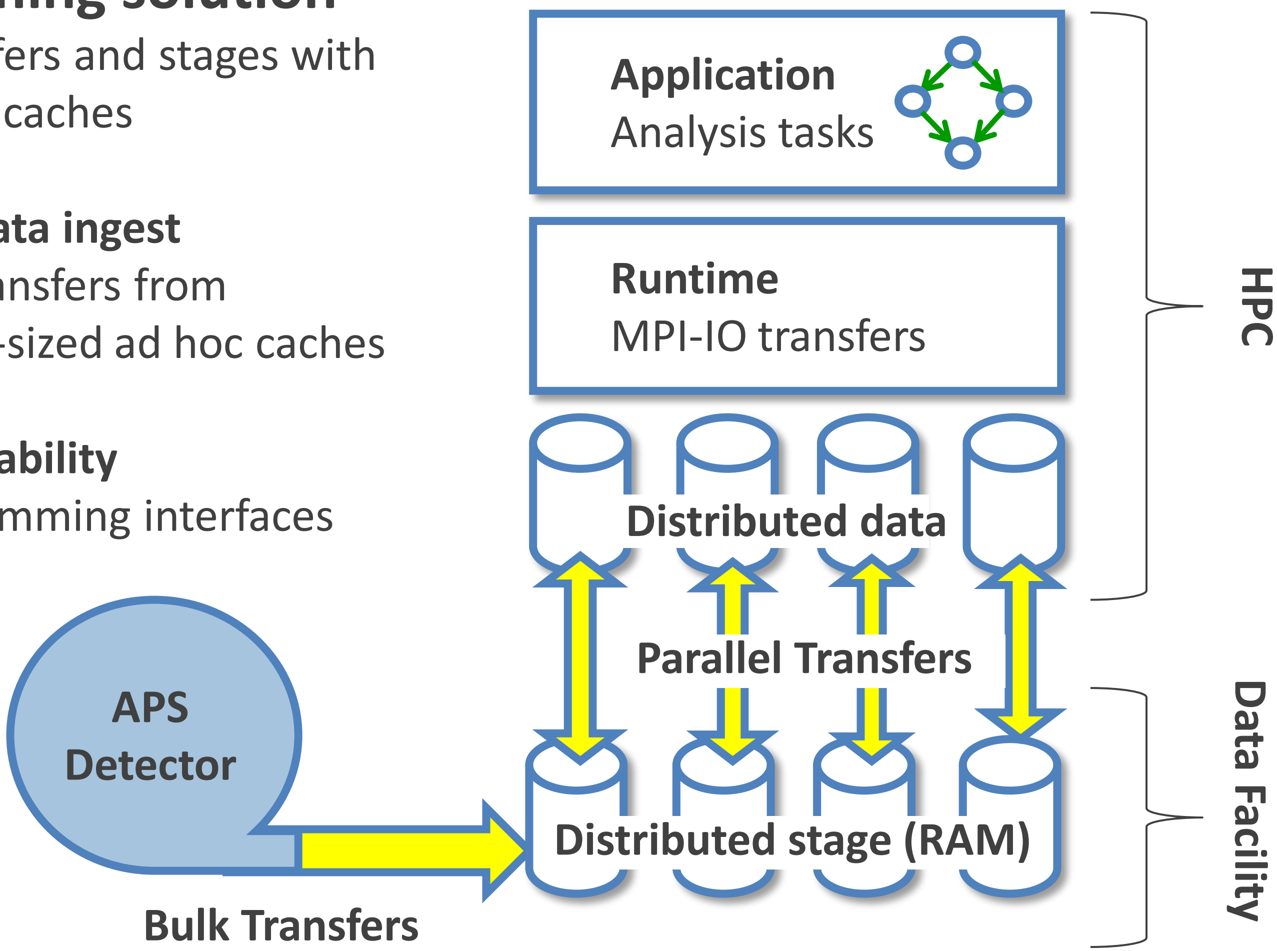
- F. Duro et al. **Exploiting data locality in Swift/T workflows using Hercules.** Proc. NESUS Workshop, 2014.
- Wozniak et al. **Big data staging with MPI-IO for interactive X-ray science**. Proc. Big Data Computing, 2014.

Next steps for streaming analysis

- **Integrated streaming solution**
  Combine parallel transfers and stages with distributed in-memory caches

- **Parallel, hierarchical data ingest**
  Implement fast bulk transfers from experiment to variably-sized ad hoc caches

- **Retain high programmability**
  Provide familiar programming interfaces

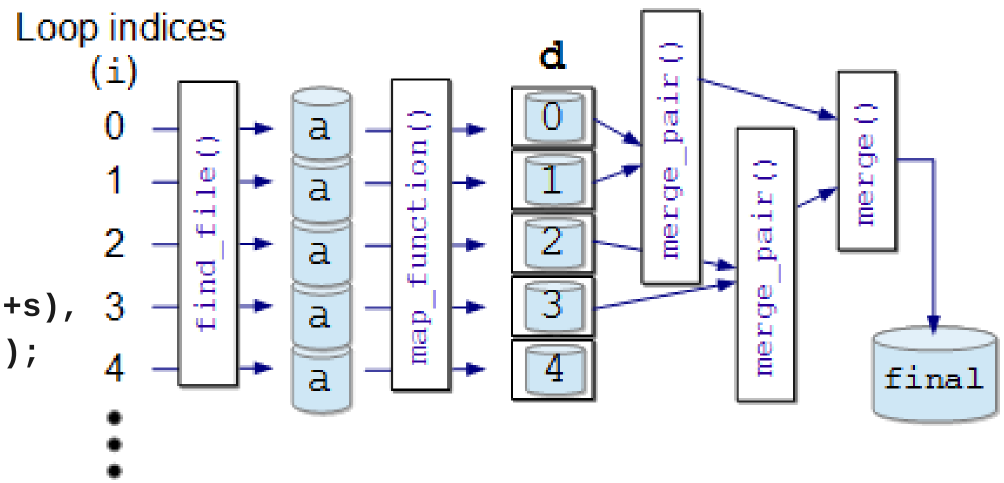


Abstract, extensible MapReduce in Swift

```
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);
}
```

- User needs to implement `map_function()` and `merge()`
- These may be implemented in native code, Python, etc.
- Could add **annotations**
- Could add additional custom **application logic**

```
(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));
  }}
```



Hercules/Swift

- Want to run arbitrary workflows over distributed filesystems that expose data locations: **Hercules** is based on **Memcached**
  - Data analytics, post-processing
  - Exceed the generality of MapReduce without losing data optimizations
- Can *optionally* send a Swift task to a particular location with simple syntax:

```
foreach i in [0:N-1] {
  location L = locationFromRank(i);
  @location=L f(i);
}
```

- Can obtain ranks from hostnames :

```
int rank = hostmapOneWorkerRank("my.host.edu");
```

- Can now specify location constraints:

```
location L = location(rank, HARD|SOFT, RANK|NODE);
```

- Much more to be done here!

Annotation system for distributed scientific data

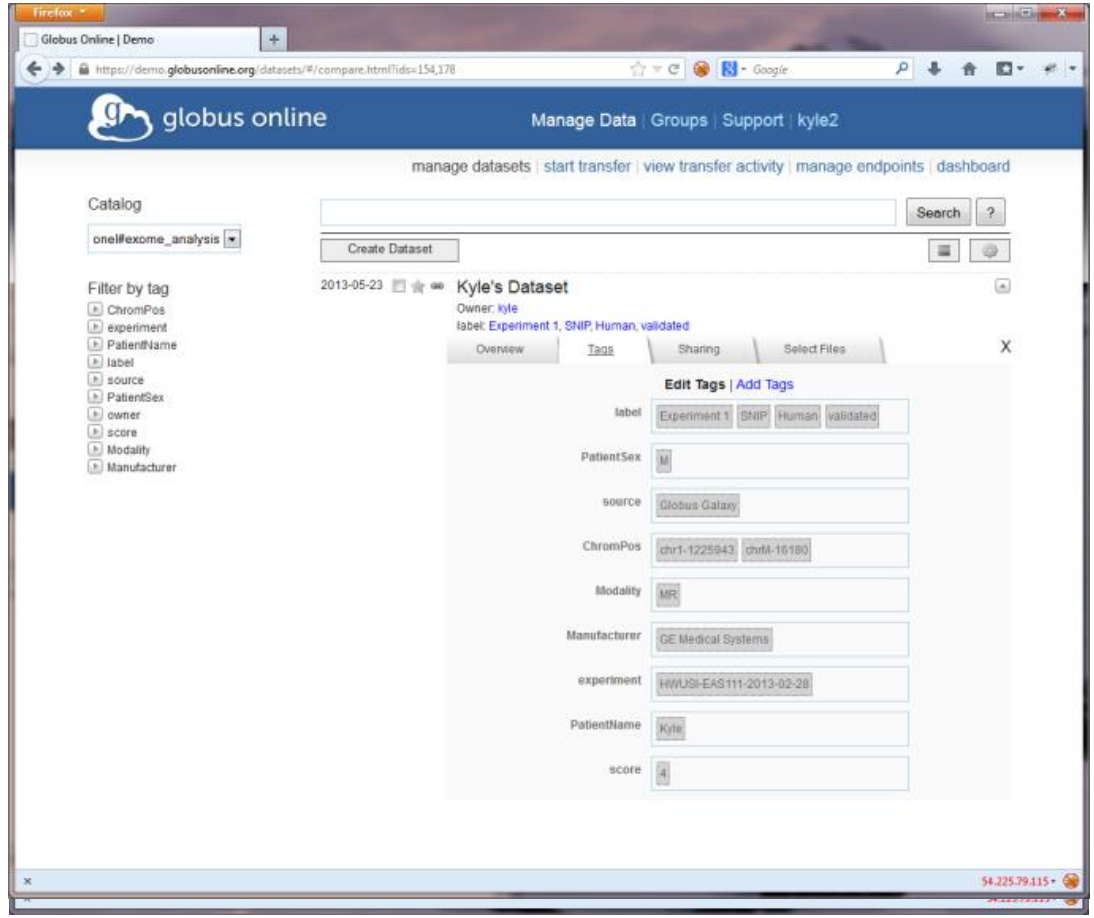
GLOBUS CATALOG

Catalog Goals

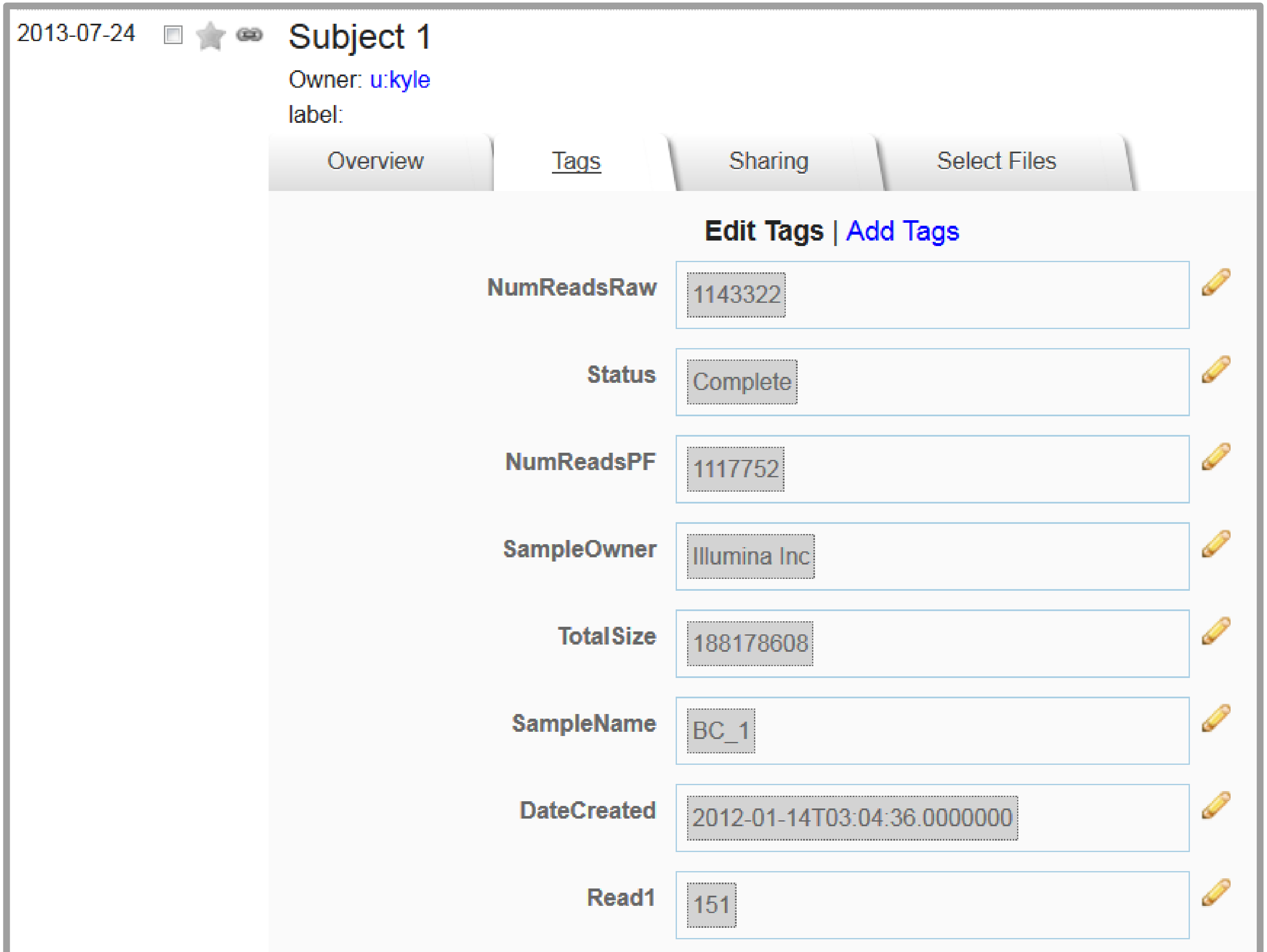
- **Group** data based on use, not location
  - Logical grouping to organize, reorganize, search, and describe usage
- **Annotate** with characteristics that reflect content …
  - Capture as much existing information as possible
  - Share datasets for collaboration- user access control
- **Operate** on datasets as units
- Research data lifecycle is **continuous and iterative**:
  - Metadata is created (automatically and manually) throughout
  - Data provenance and linkage between raw and derived data
- Most often:
  - Data is grouped and acted on collectively
    - Views (slices) may change depending on activity
  - Data and metadata changes over time
  - Access permissions are important (and also change)

Catalog Data Model

- **Catalog:** a hosted resource that enables the grouping of related datasets
- **Dataset:** a virtual collection of (schema-less) metadata and distributed data elements
- **Annotation:** a piece of metadata that exists within the context of a dataset or data member
  - Specified as key-value pairs
- **Member:** a specific data item (file, directory) associated with a dataset



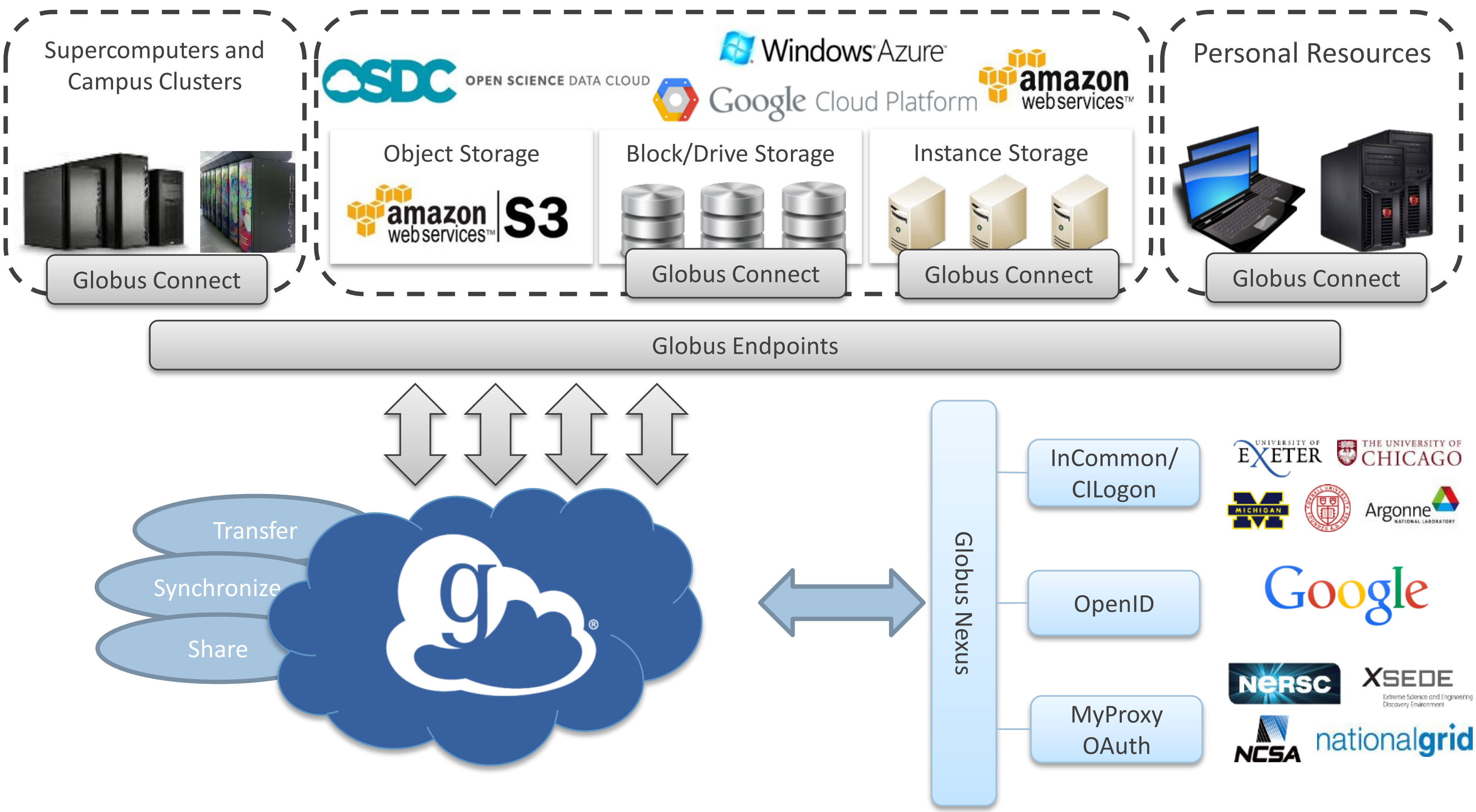
Web interface for annotations



High-speed wide area data transfers

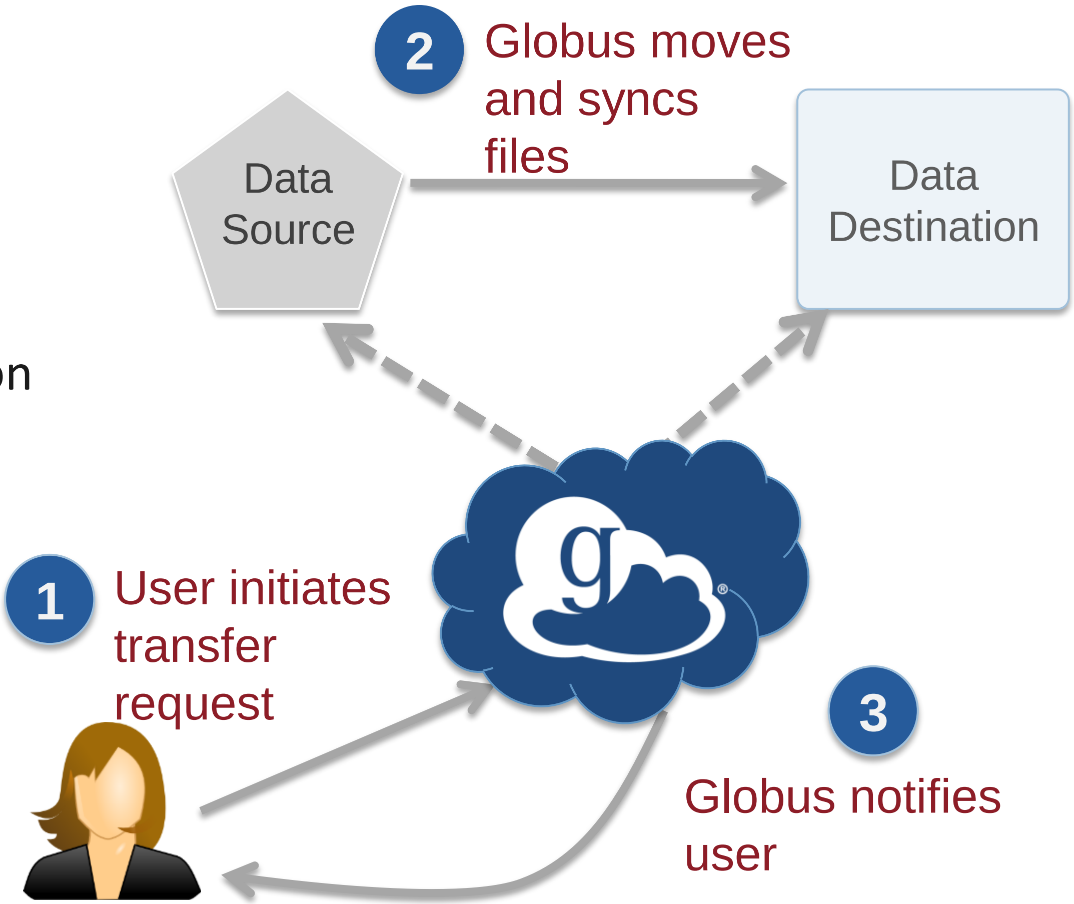
GLOBUS TRANSFER

Globus Transfer

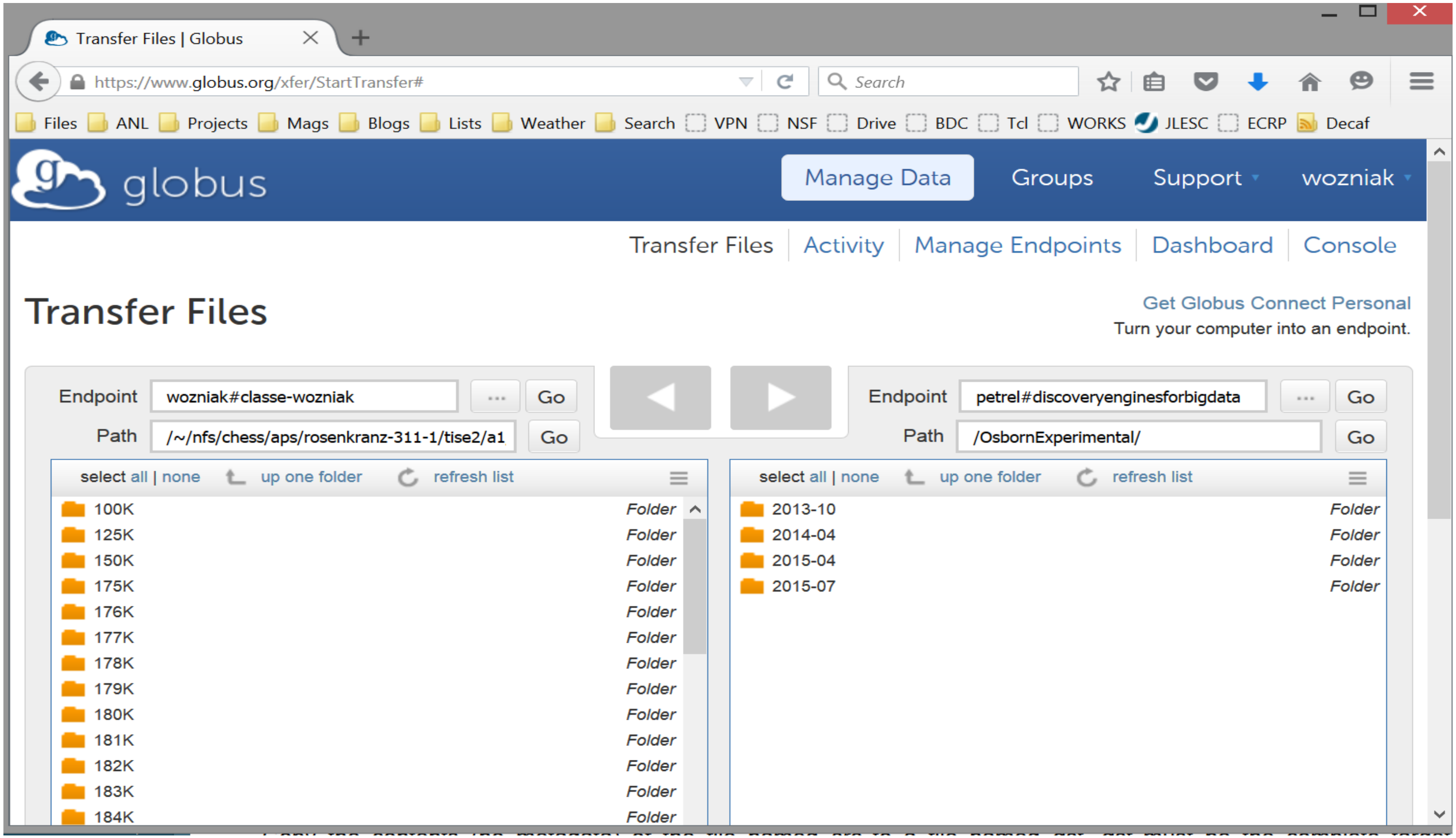


Globus Transfer

- Reliable, secure, high-performance ***file transfer*** and ***synchronization***
- "Fire-and-forget" transfers
- Automatic fault recovery
- Seamless security integration
- **10x faster than SCP**



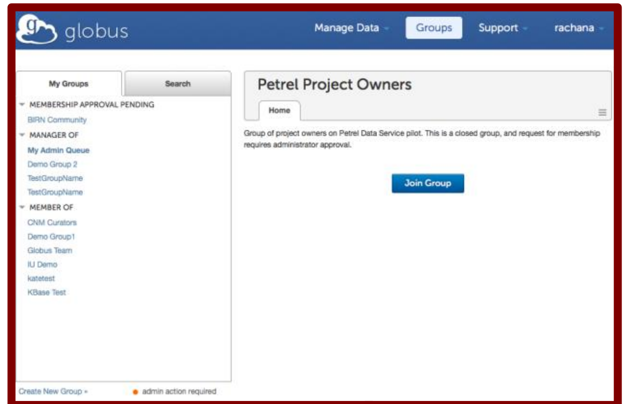
Globus Transfer: CHESS to ALCF

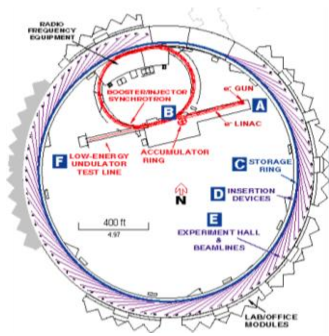


- K. Dedrick. **Argonne group sets record for largest X-ray dataset ever at CHESS.** News at CHESS, Oct. 2015.

The Petrel research data service

- High-speed, high-capacity data store
- Seamless integration with data fabric
- Project-focused, self-managed

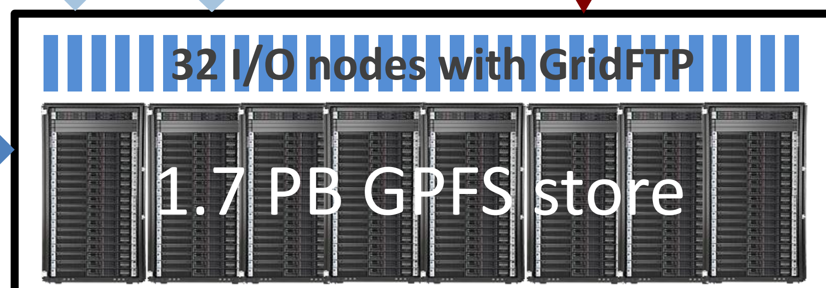




100 TB allocations
User managed access

Other sites, facilities, colleagues

3 GB/s network



Rapid and remote structured data visualization

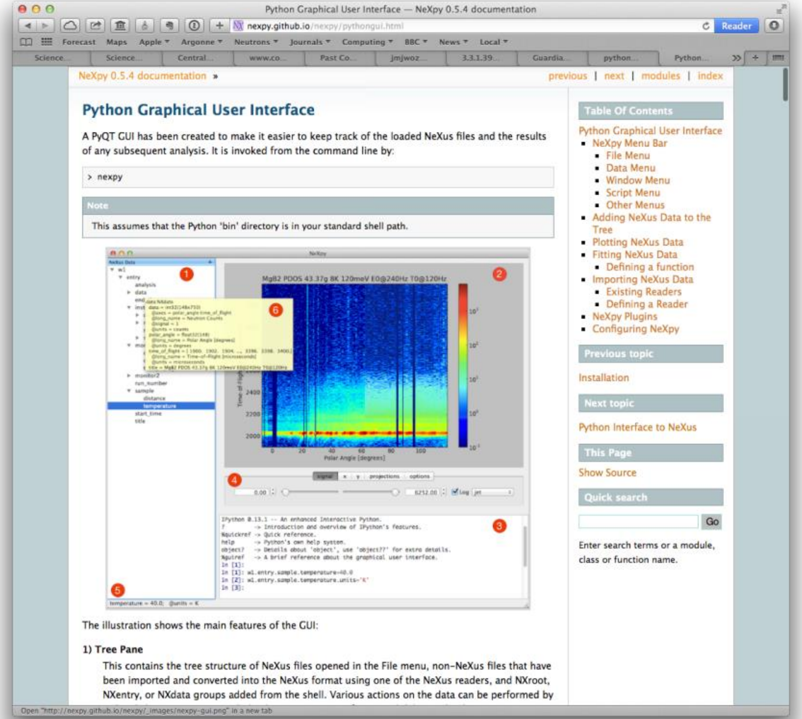
NEXPY / NXFS

NeXpy: A Python Toolbox for Big Data

- A toolbox for manipulating and visualizing arbitrary NeXus data of any size
- A scripting engine for GUI applications
- A portal to Globus Catalog
- A demonstration of the value of combining:
  - a flexible data model
  - a powerful scripting language

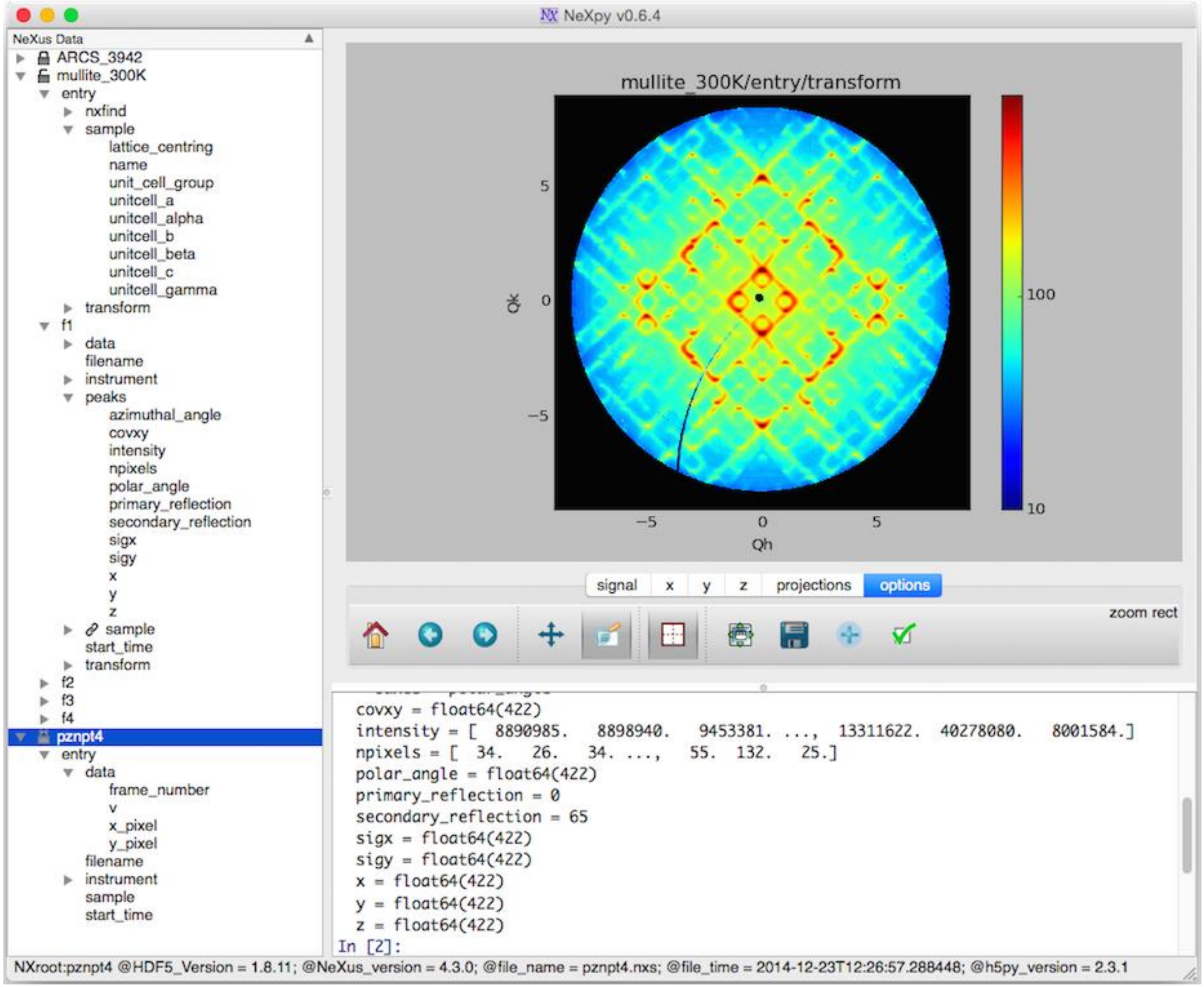
http://nexpy.github.io/nexpy

$ pip install nexpy



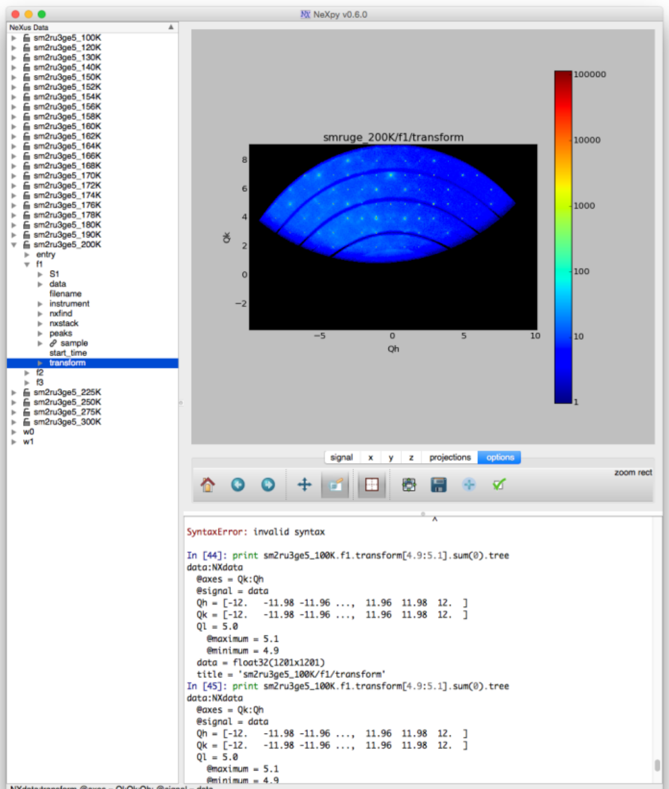
NeXpy = NeXus + NumPy

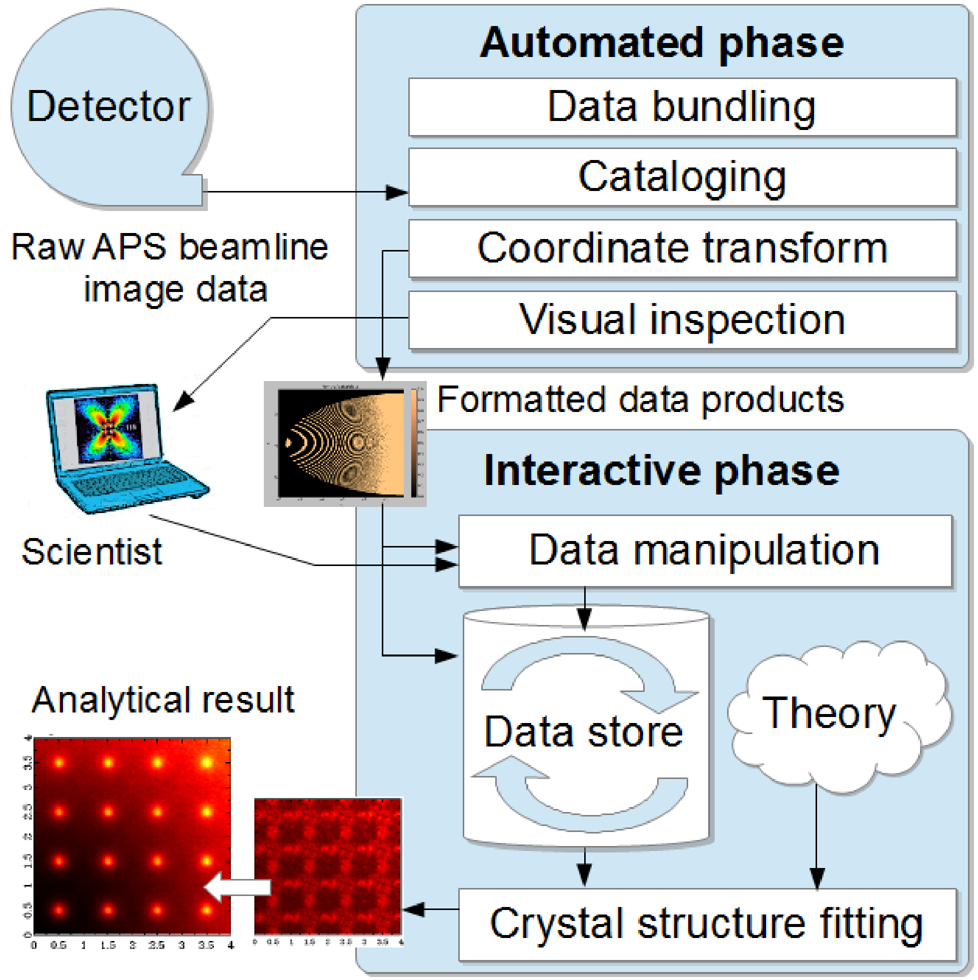
Mullite


NeXpy v0.6.4
NeXus Data
ARCS_3942
mullite_300K
entry
nxfind
sample
lattice_centring
name
unit_cell_group
unitcell_a
unitcell_alpha
unitcell_b
unitcell_beta
unitcell_c
unitcell_gamma
transform
f1
data
filename
instrument
peaks
azimuthal_angle
covxy
intensity
npixels
polar_angle
primary_reflection
secondary_reflection
sigx
sigy
x
y
z
sample
start_time
transform
f2
f3
f4
pznpt4
entry
data
frame_number
v
x_pixel
y_pixel
filename
instrument
sample
start_time
mullite_300K/entry/transform
Qk
Qh
100
10
signal
x
y
z
projections
options
zoom rect
covxy = float64(422)
intensity = [8890985. 8898940. 9453381. ..., 13311622. 40278080. 8001584.]
npixels = [34. 26. 34. ..., 55. 132. 25.]
polar_angle = float64(422)
primary_reflection = 0
secondary_reflection = 65
sigx = float64(422)
sigy = float64(422)
x = float64(422)
y = float64(422)
z = float64(422)
In [2]:
NXroot:pznpt4 @HDF5_Version = 1.8.11; @NeXus_version = 4.3.0; @file_name = pznpt4.nxs; @file_time = 2014-12-23T12:26:57.288448; @h5py_version = 2.3.1

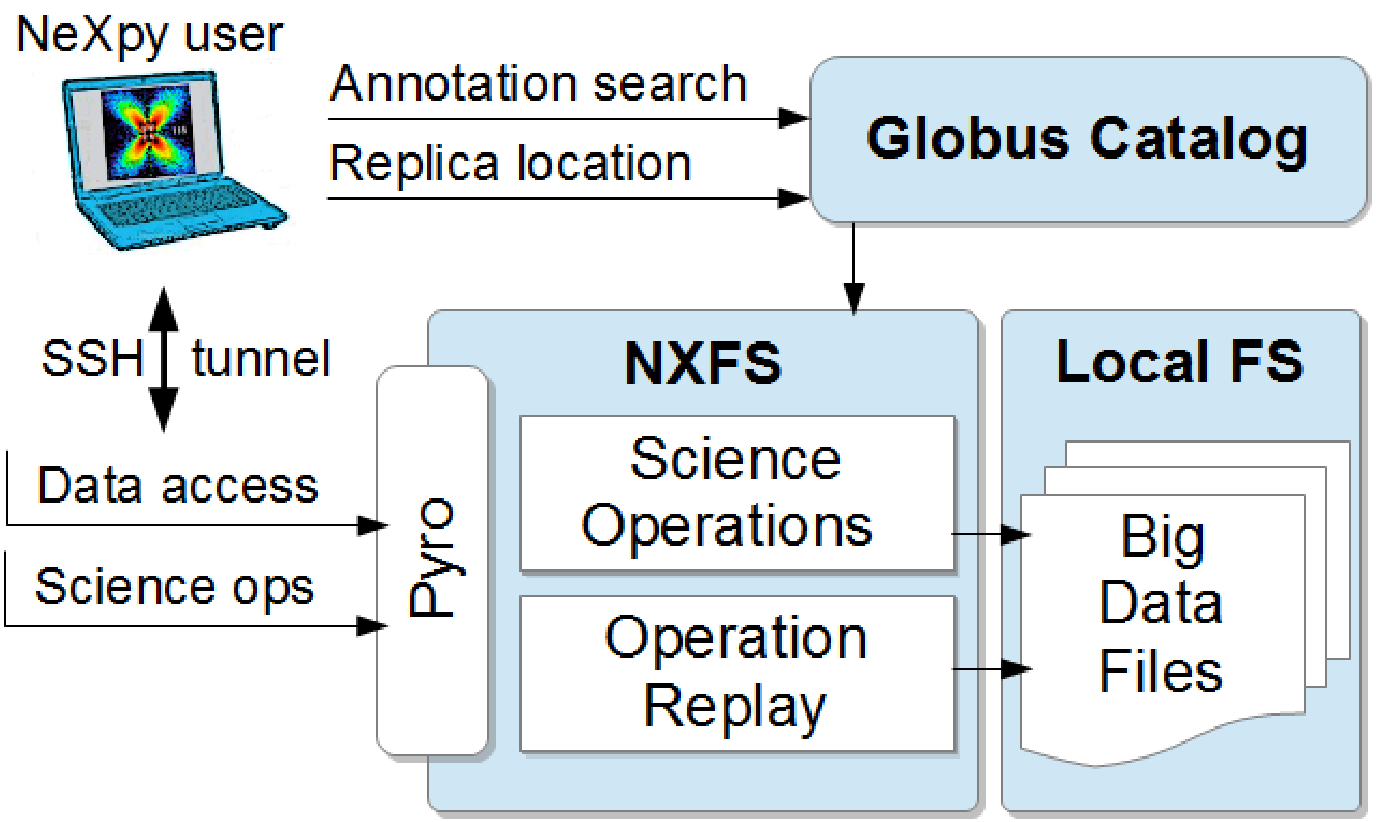
NeXpy in the Pipeline

- Use of NeXpy throughout the analysis pipeline





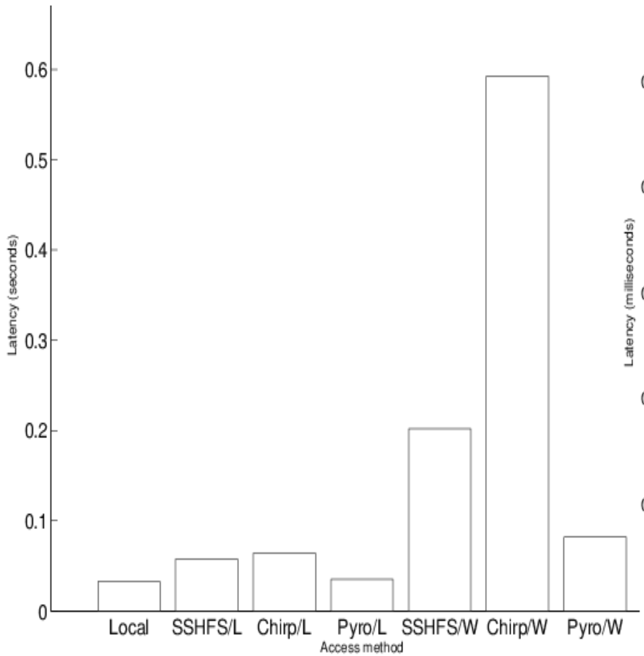
The NeXus File Service (NXFS)

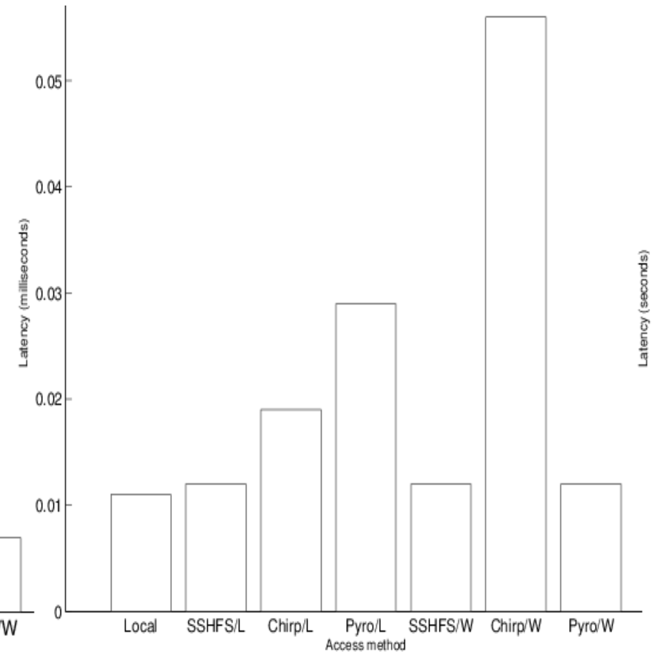


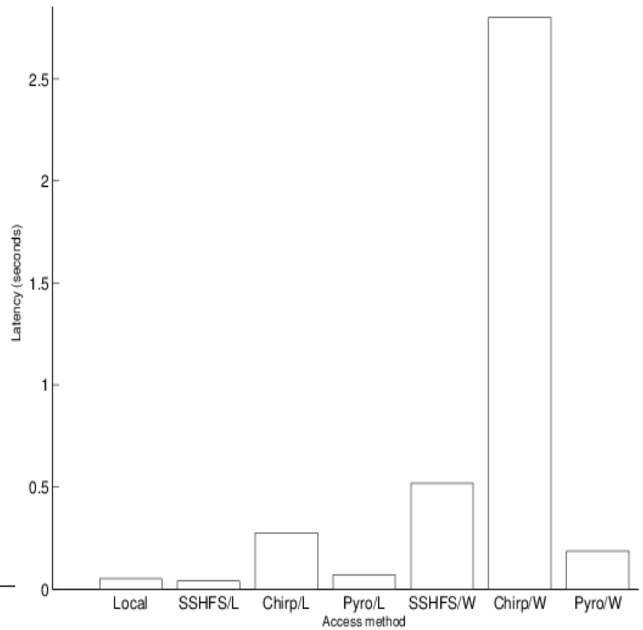
- Wozniak et al. **Big data remote access interfaces for light source science**. Proc. Big Data Computing, 2015.

NXFS Performance

- Faster than application-agnostic remote filesystem technologies
  - Compared Pyro to Chirp and SSHFS from inside ANL (**L**) and AWS EC2 (**W**)
- Plus ability to invoke remote methods!







- File open (10^{-1}s)
- Metadata read (10^{-2}s)
- Pixel read (1s)

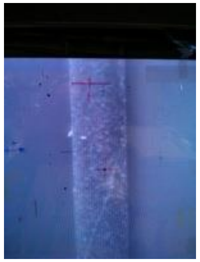
Operation and Time Scale

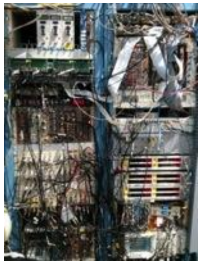
Near Field – High Energy Diffraction Microscopy

Collaboration with APS Sector 1: Jon Almer, Hemant Sharma, et al.

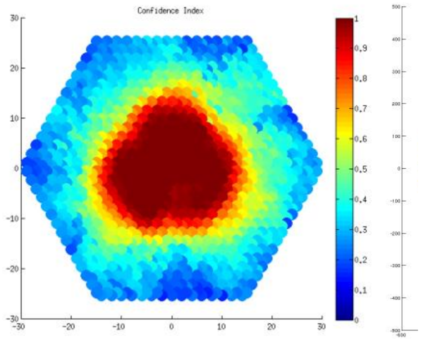
CASE STUDY: NF-HEDM

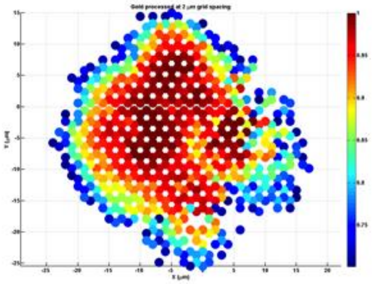
Determining the crystal structure of metals non-destructively

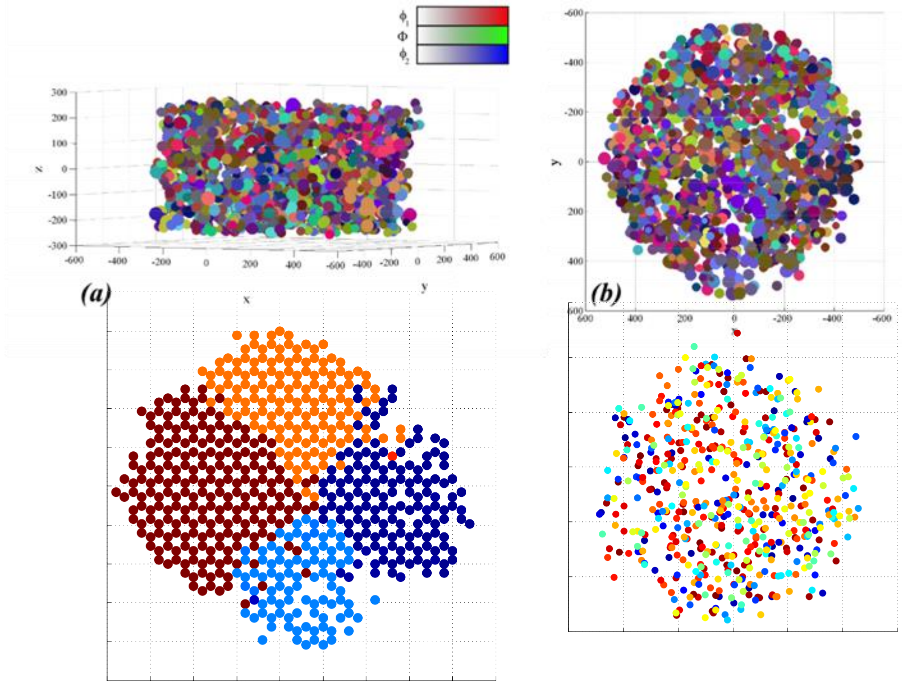




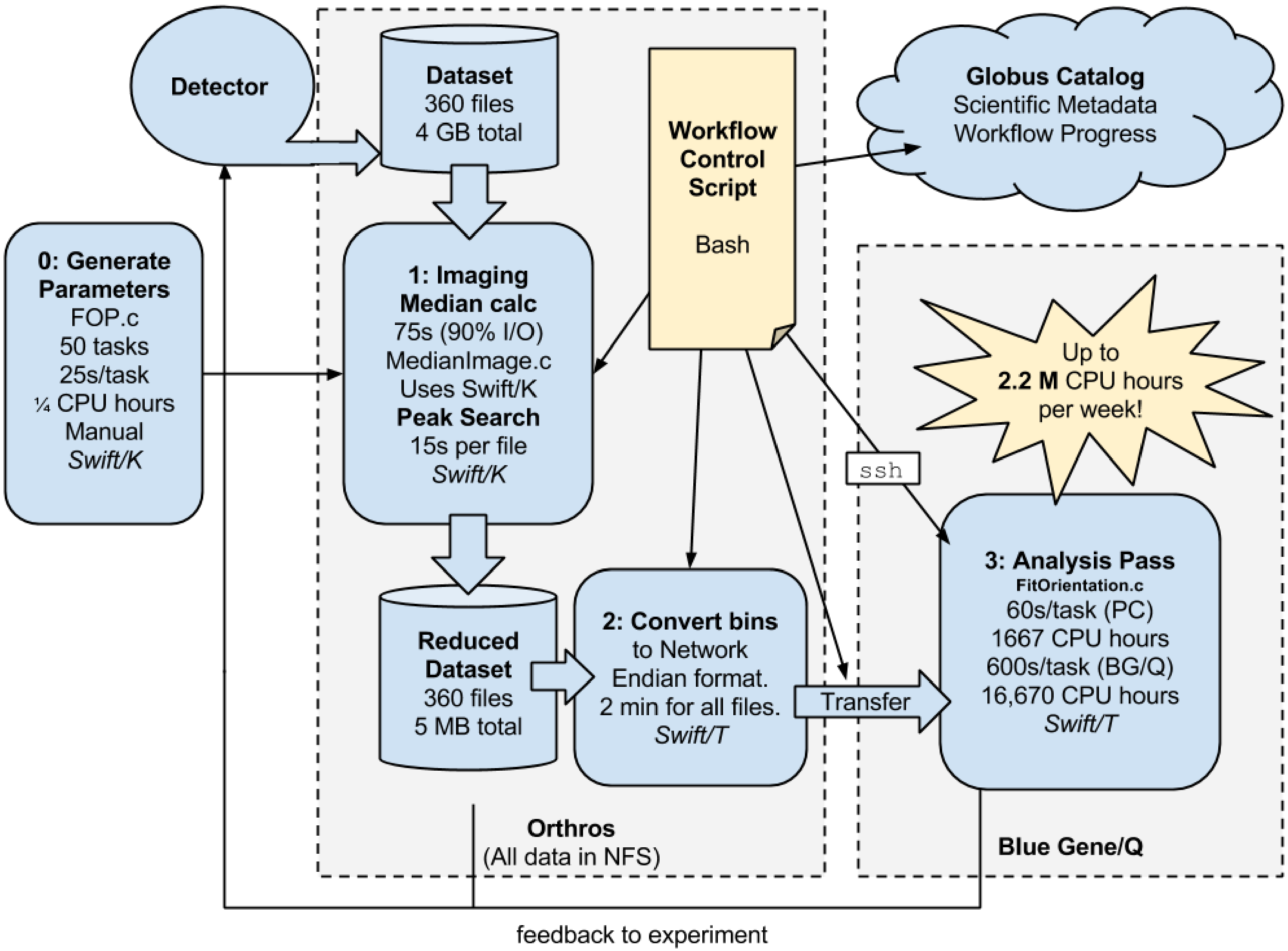
Gold calibrant wire





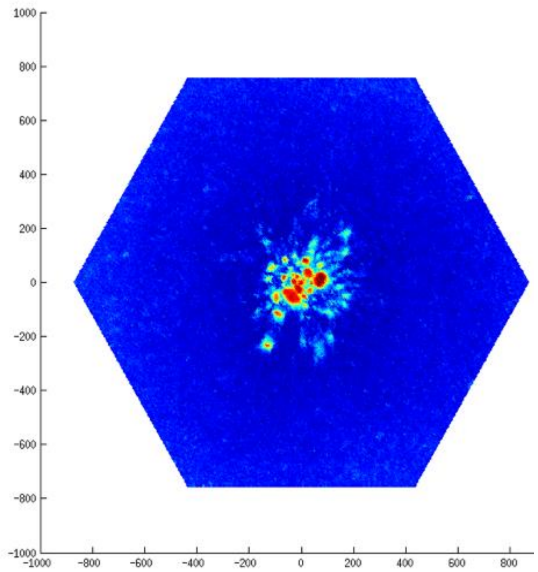


NF-HEDM

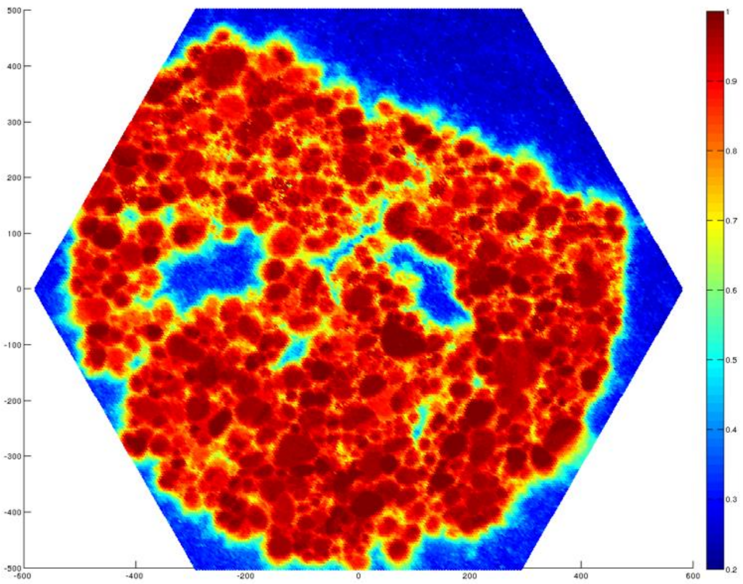


Detector
Dataset
360 files
4 GB total
Workflow Control Script
Bash
Globus Catalog
Scientific Metadata
Workflow Progress
0: Generate Parameters
FOP.c
50 tasks
25s/task
¼ CPU hours
Manual
Swift/K
1: Imaging Median calc
75s (90% I/O)
MedianImage.c
Uses Swift/K
Peak Search
15s per file
Swift/K
Up to
2.2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

High-Energy Diffraction Microscopy



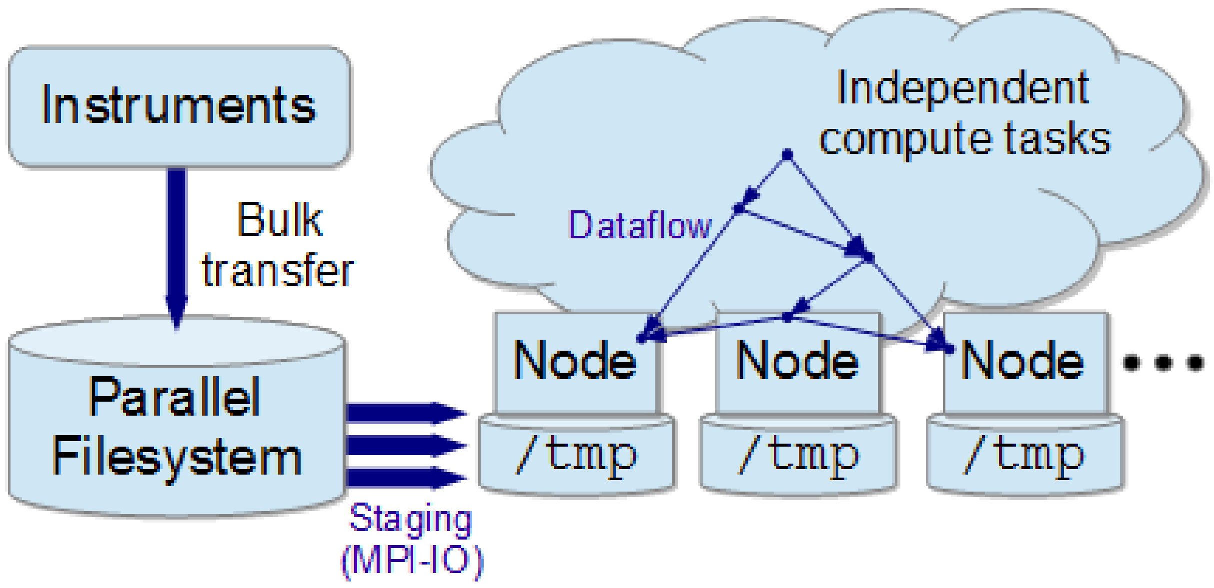
October 2013: Without Swift



April 2014: With Swift

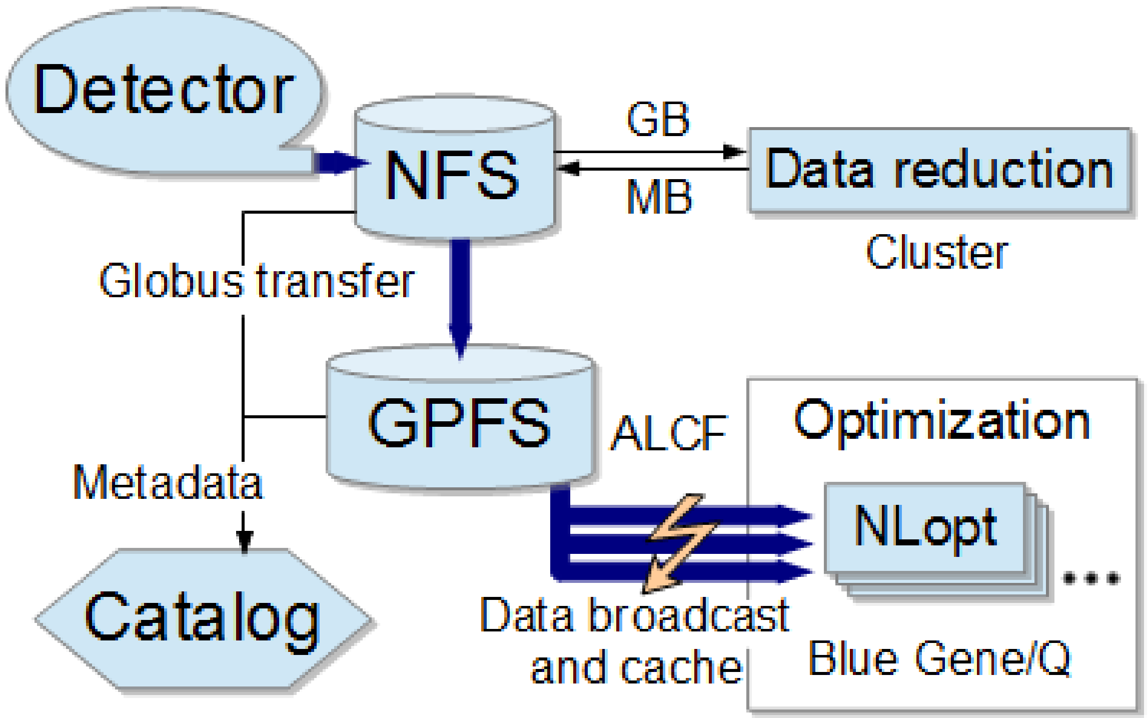
- Near-field high-energy diffraction microscopy discovers metal grain shapes and structures
- The experimental results are greatly improved with the application of Swift-based cluster computing (**RED** indicates higher confidence in results)

Big picture: Task-based HPC on Big Data



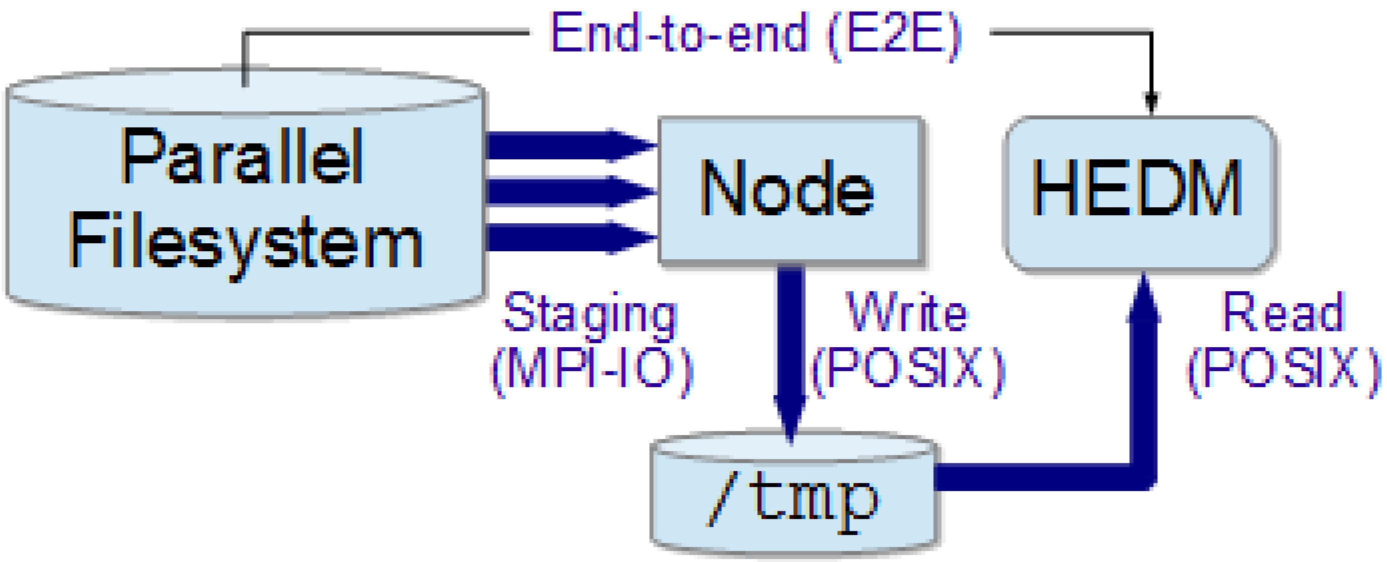
- Existing C code assembled into scalable HPC program with Swift/T
- Problem: Each task must consumes ~500 MB of experimental data
- Runs on the Blue Gene/Q
- Relevant to Big Data – HPC convergence
- Could use Swift/T data locality annotations for high-level, data location-aware programming

Intended use of broadcast operation



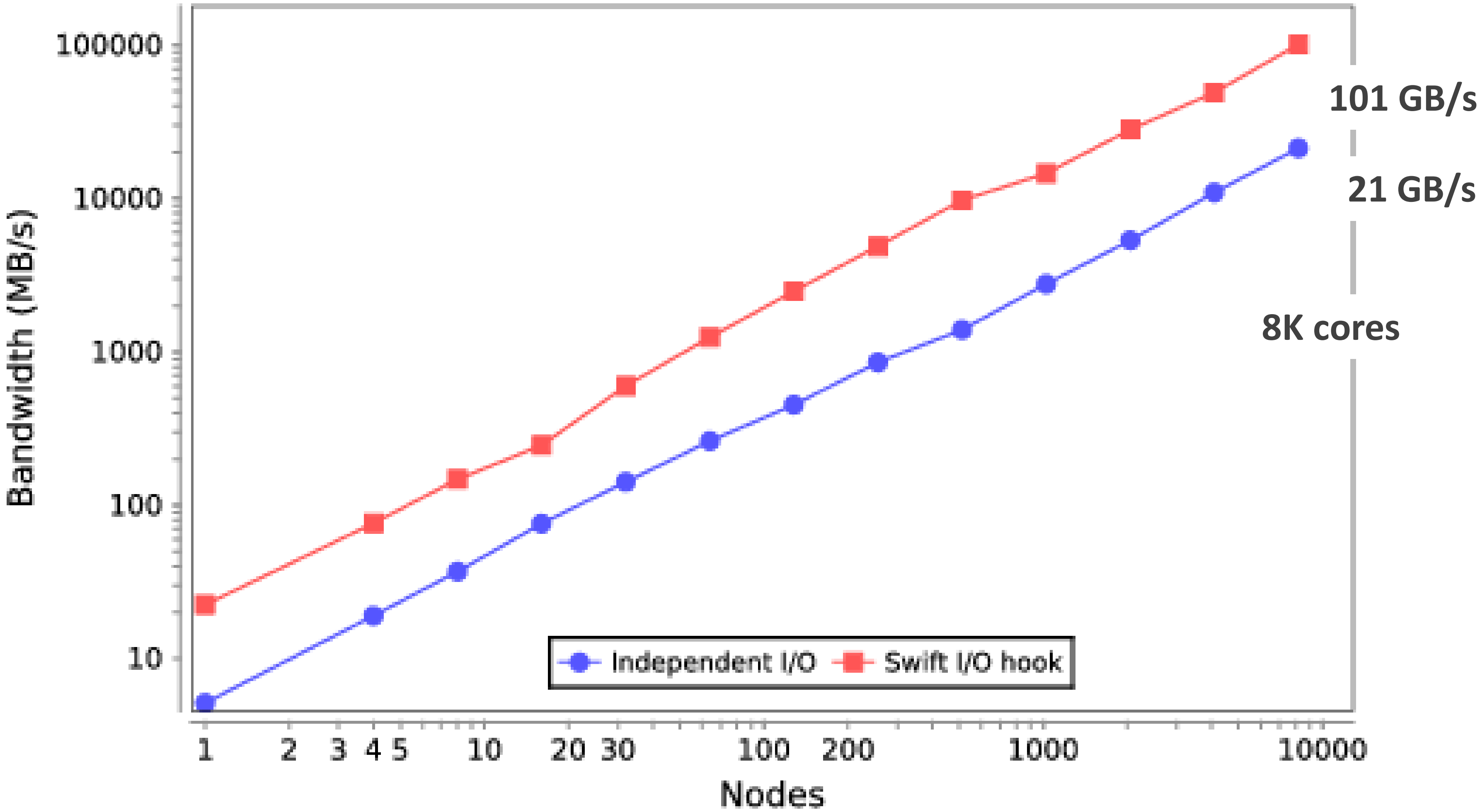
- Grain orientation optimization workflow runs on BG/Q once data is there
- Each task needs to read all input from a given dataset
- Desire to use MPI-IO before running tasks

Big Data Staging with MPI-IO

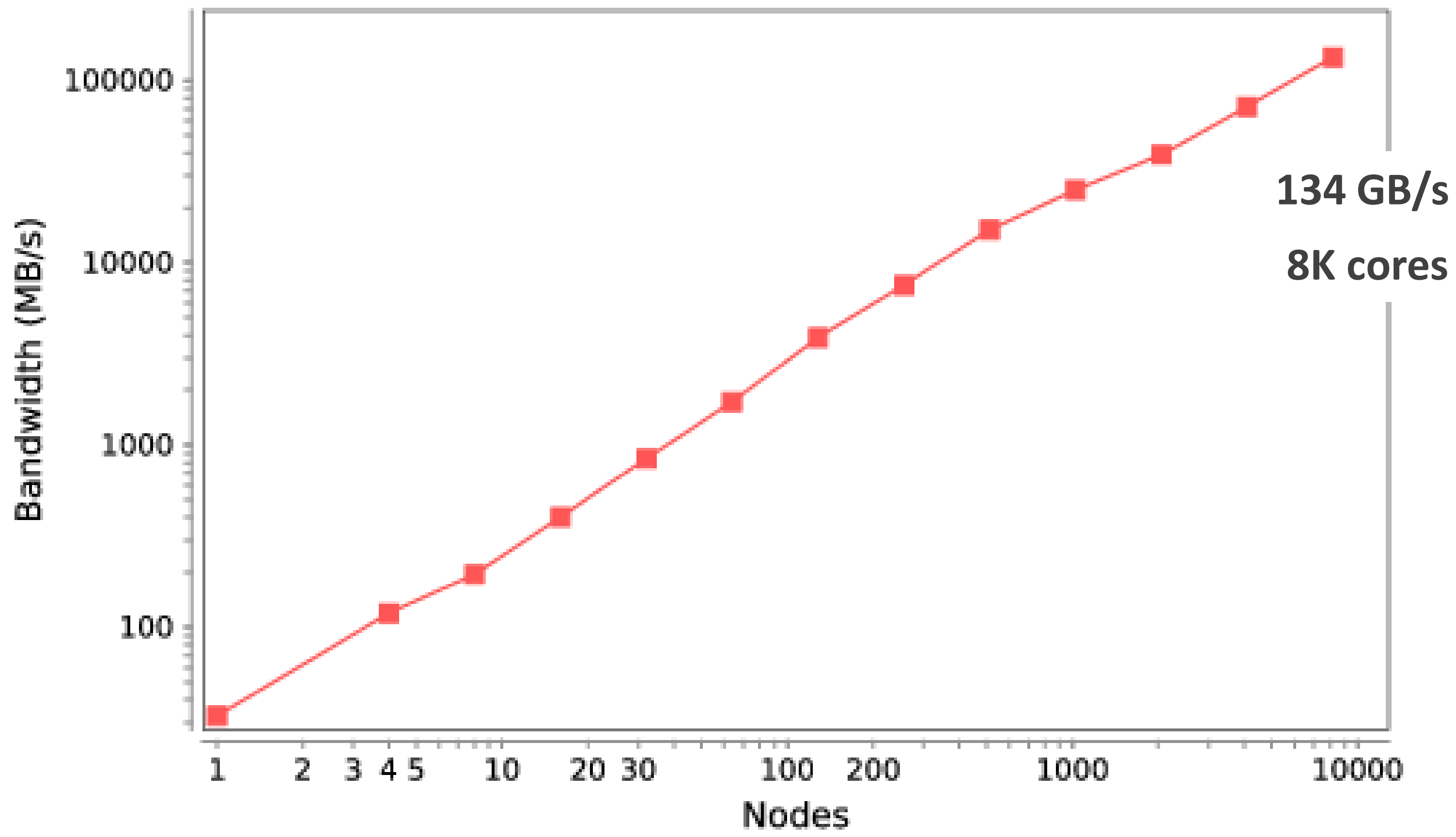


- Solution: Broadcast experimental data on HPC system with MPI-IO
- Tasks consume data normally from node-local storage

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

NF-HEDM: 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

- Described Big Data + HPC application: X-ray crystallography
- Described four relevant tools:
  - Swift
  - Globus Transfer
  - Globus Catalog
  - NeXpy/NXFS
- Described path forward, integrating tools for streaming workflows

- Thanks to the organizers
- Thanks to our application collaborators

- **Questions?**

- **http://swift-lang.org**