Parallel and Distributed Application Paradigms

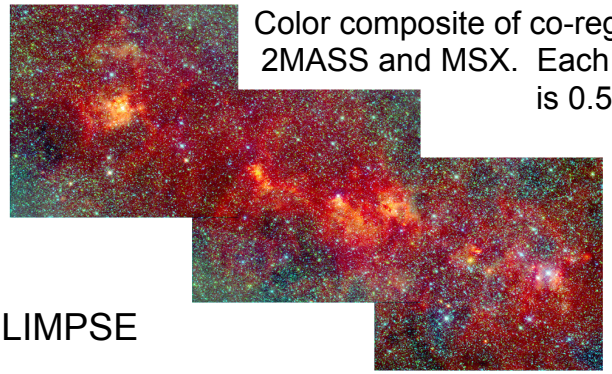
Daniel S. Katz
Senior Fellow, Computation Institute (University of Chicago & Argonne National Laboratory)
Affiliate Faculty, CCT (LSU)
Adjunct Associate Professor, ECE (LSU)

Montage (http://montage.ipac.caltech.edu/)

- An astronomical image mosaic service for the National Virtual Observatory (VAO)
- Originally developed by core team at JPL and Caltech (IPAC & CACR) – now at U. Chicago, too…; grid architecture developed in collaboration with ISI; further research with other institutions
- Delivers custom, science grade image mosaics
  - An image mosaic is a combination of many images containing individual pixel data so that they appear to be a single image from a single telescope or spacecraft
  - User specifies projection, coordinates, spatial sampling, mosaic size, image rotation
  - Preserve astrometry (to 0.1 pixels) & flux (to 0.1%)
- Modular, portable "toolbox" design
  - Loosely-coupled engines for image reprojection, background rectification, co-addition
    - Flexibility; e.g., custom background algorithm; use as a reprojection and co-registration engine
  - Each engine is an executable compiled from ANSI C

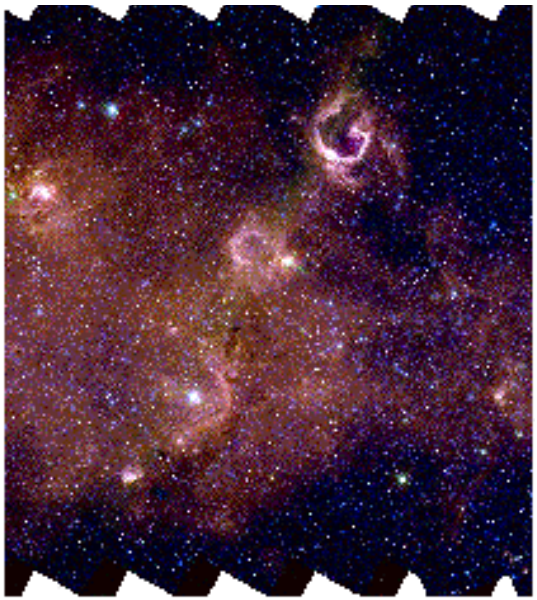
David Hockney *Pearblossom Highway* 1986

Montage use by Spitzer Legacy Teams

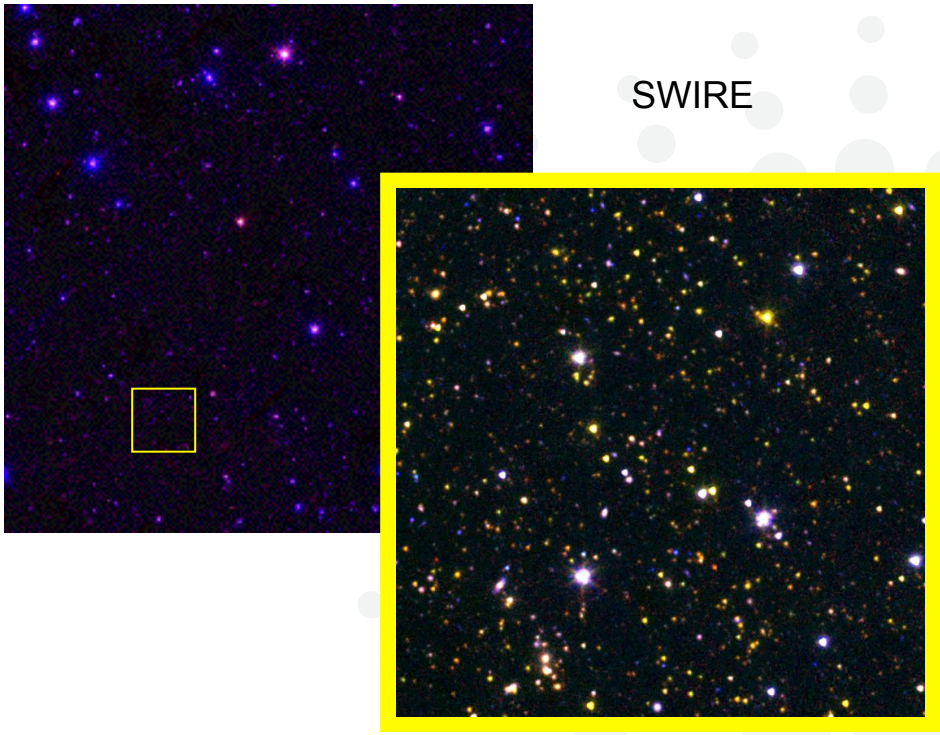


Color composite of co-registered 2MASS and MSX. Each square is 0.5° x 0.5°

GLIMPSE



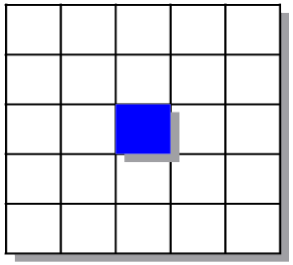
3-color GLIMPSE image mosaic over 1.1° x 0.8°



SWIRE

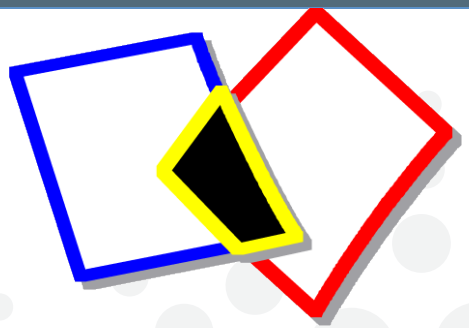
Right: Spitzer IRAC 3 channel mosaic (3.6μm in green, 4.5μm in red, and i-band optical in blue); high redshift non-stellar objects are visible in the full resolution view (yellow box).

Montage v1.7 Reprojection:
mProject module

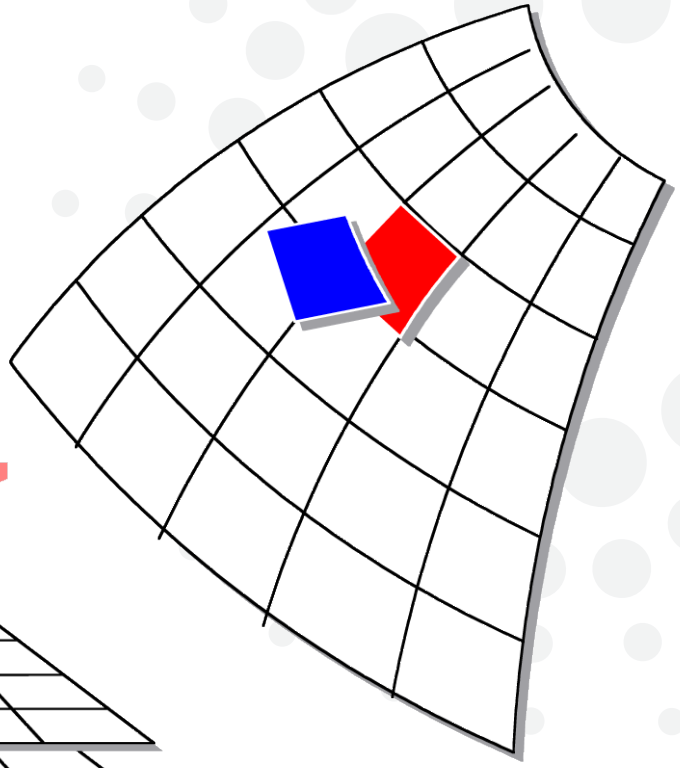


Arbitrary Input Image

Central to the algorithm is accurate calculation of the area of spherical polygon intersection between two pixels (assumes great circle segments are adequate between pixel vertices)



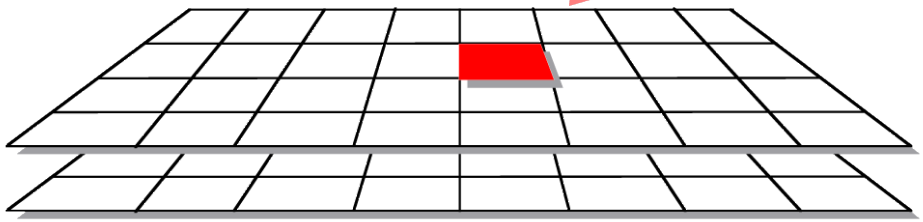
Input pixels projected on celestial sphere



Output pixels projected on celestial sphere

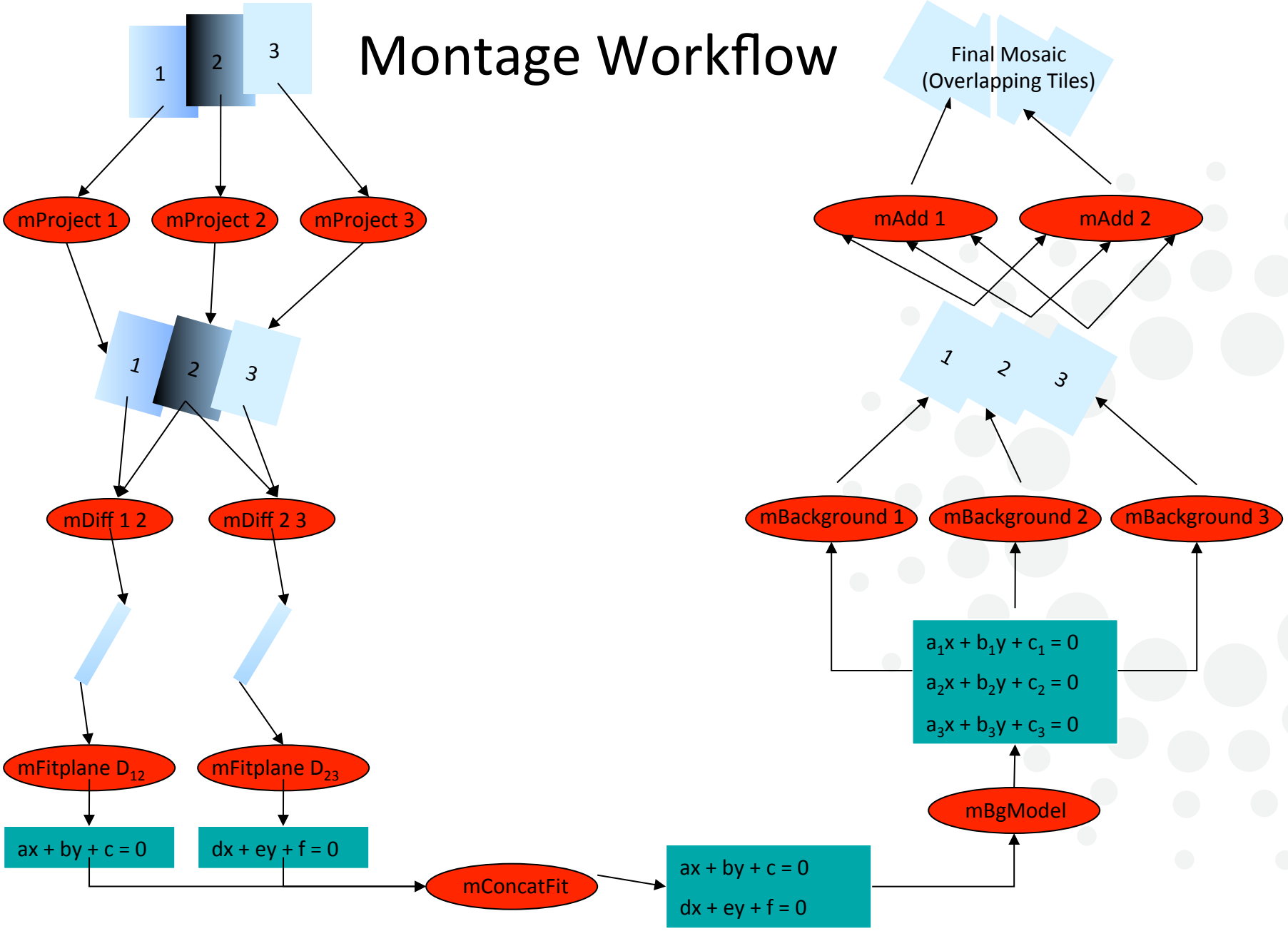
```
SIMPLE =                    T /
 BITPIX=                  -64 /
 NAXIS  =                    2 /
 NAXIS1=                 3000 /
 NAXIS2=                 3000 /
 CDELT1=      -   3.333333E-4 /
 CDELT2=      -   3.333333E-4 /
 CRPIX1=              1500.5 /
 CRPIX2=              1500.5 /
 CTYPE1='RA---TAN'
 CTYPE2='DEC--TAN'
 CRVAL1=           265.91334 /
 CRVAL2=           -29.35778 /
 CROTA2=                 0. /
END
```

FITS header defines output projection



Reprojected Image

Montage Workflow

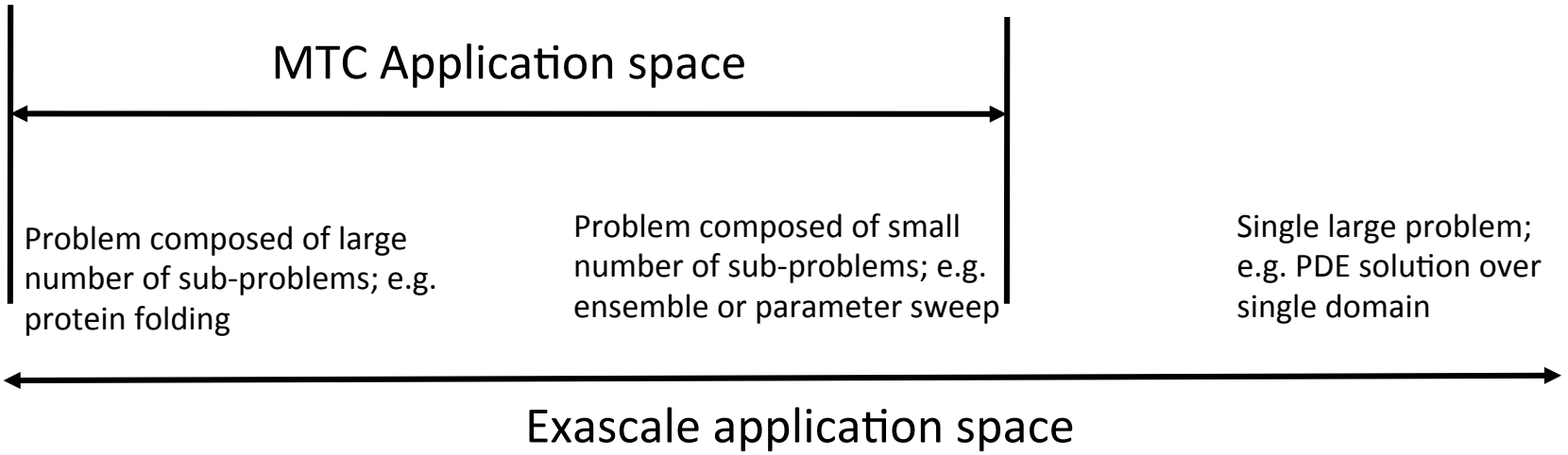


Montage Initial Version

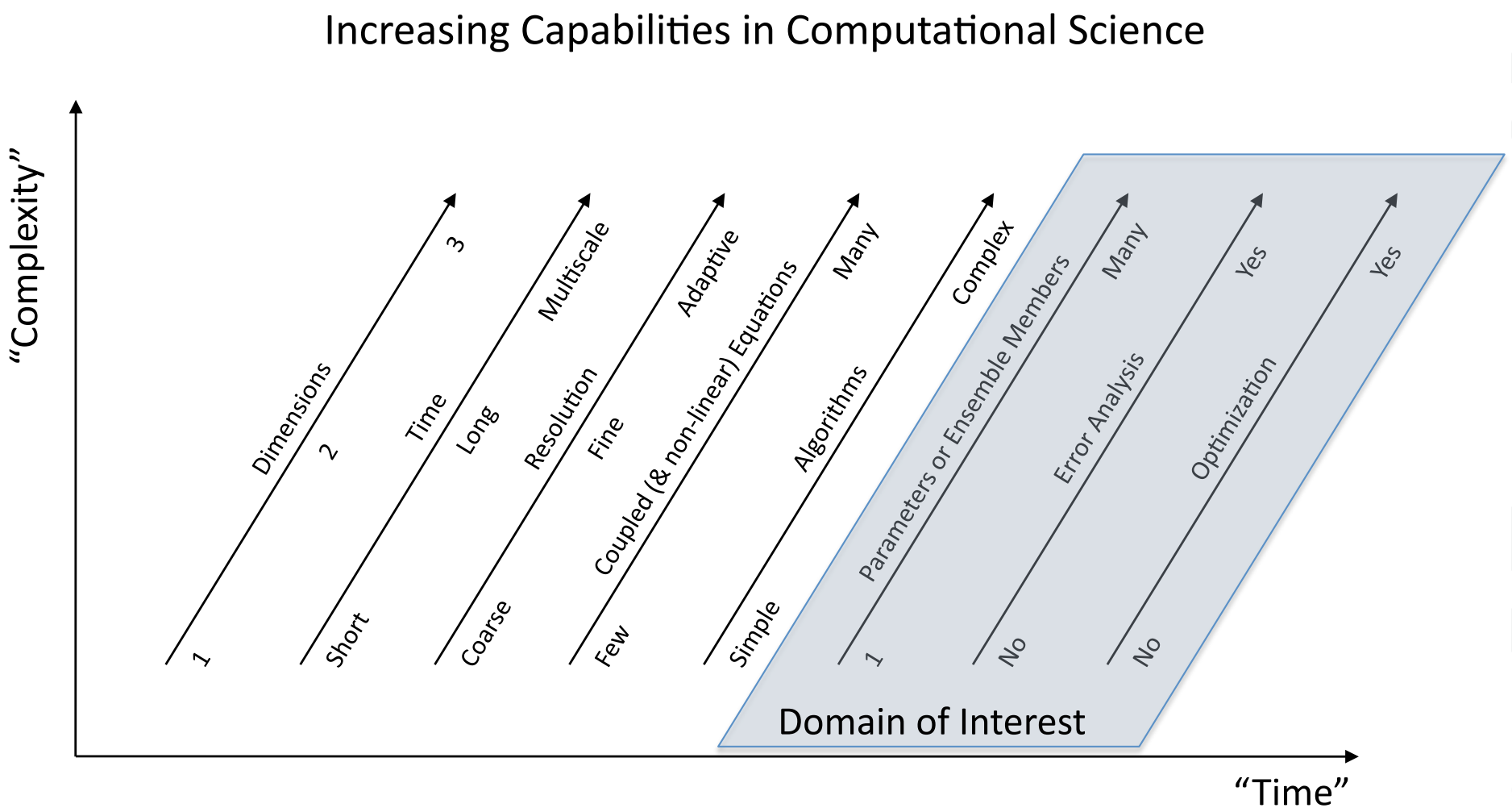
- Tested on a variety of architectures and problems
  - Good science results
- Decent performance on a single processor
- MPI version (enabled through preprocessor directives)
  - Mostly round-robin parallelization of tasks in a stage
    - Good scaling until problem/processor gets too small
  - Dynamic: each stage uses files from previous stage(s)
  - mAdd parallelization a bit more complicated
    - Scaling depends on shared I/O system performance

Many Task Computing (MTC)

- MTC applications: a set of tasks, usually linked by file system operations
  - Tasks are usually sequential or parallel executables
  - Set of tasks can be static or dynamic, homogeneous or heterogeneous
  - Files output by one program are used as input by others
- MTC applications can be described in many ways, including as in a compiled program, as a DAG, in a script, etc.
- MTC applications can include:
  - Bag of tasks, MapReduce, multi-stage workflow, iterative MapReduce, campaign, coupled components, ...



MTC domains


Increasing Capabilities in Computational Science
"Complexity"
"Time"
Dimensions
1
2
3
Time
Short
Long
Resolution
Coarse
Multiscale
Fine
Coupled (& non-linear) Equations
Few
Adaptive
Many
Algorithms
Simple
Complex
Parameters or Ensemble Members
1
Many
Error Analysis
No
Yes
Optimization
No
Yes
Domain of Interest

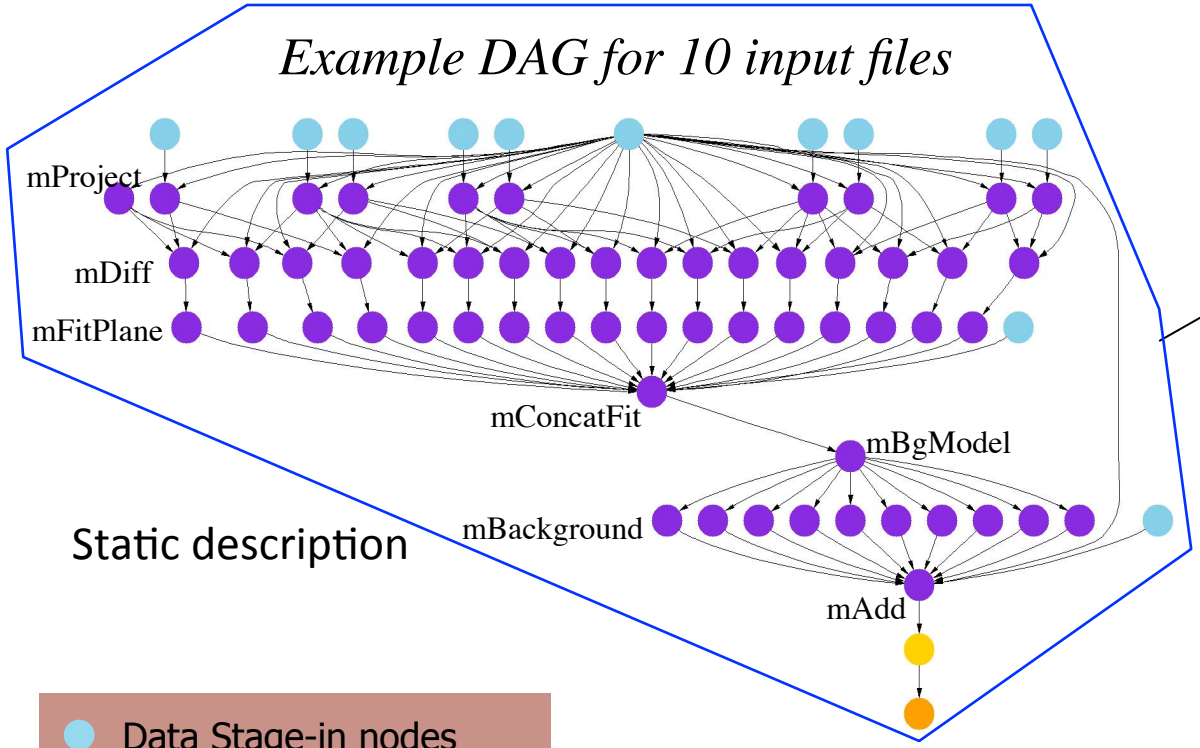
MTC Application Challenges

- Goal: Easily program and efficiently execute MTC applications on current and future parallel and distributed computers
- Needs:
  - Tools for describing MTC applications
  - Middleware for executing MTC applications
  - Future system characteristics to improve MTC applications

D. S. Katz, T. G. Armstrong, Z. Zhang, M. Wilde, and J. M. Wozniak, Many Task Computing and Blue Waters, Technical Report CI-TR-13-0911, Computation Institute, University of Chicago & Argonne National Laboratory, 2012. http://www.ci.uchicago.edu/research/papers/CI-TR-13-0911

S. Jha, D. S. Katz, M. Parashar, O. Rana, J. Weissman, "Critical Perspectives on Large-Scale Distributed Applications and Production Grids," Best paper at IEEE Grid 2009

Montage on the Grid Using Pegasus



Maps an abstract workflow to an executable form

Pegasus

http://pegasus.isi.edu/

Grid Information Systems

Information about available resources, data location

Condor DAGMan

Executes the workflow

MyProxy

Grid

User's grid credentials

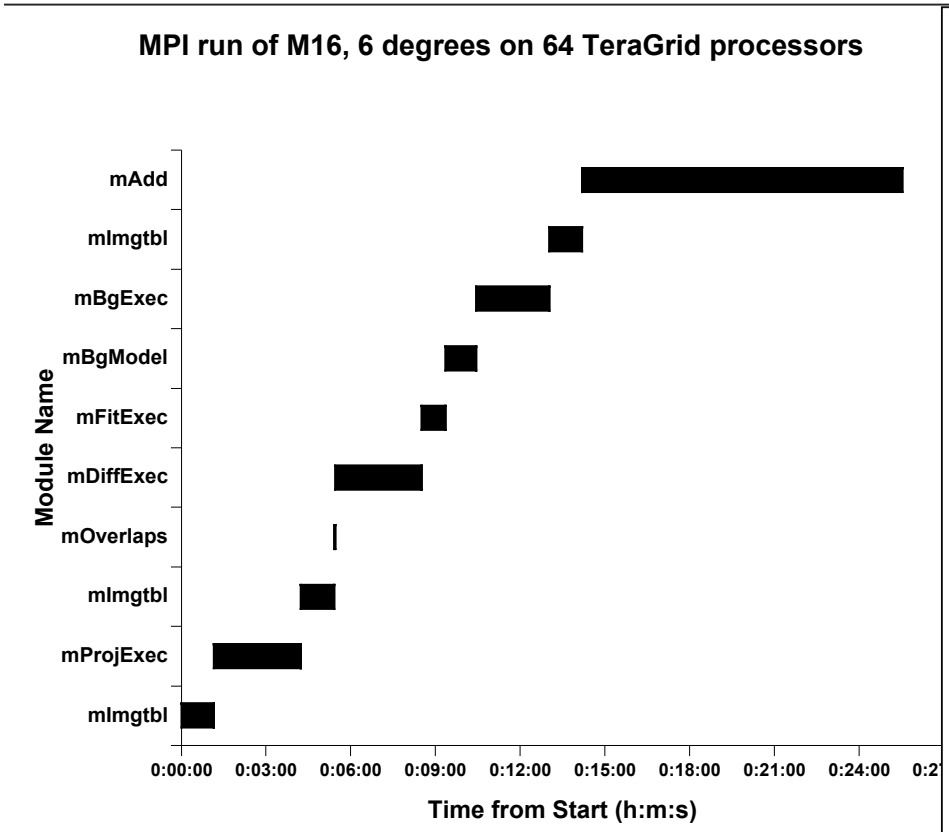
- Data Stage-in nodes
- Montage compute nodes
- Data stage-out nodes
- Registration nodes

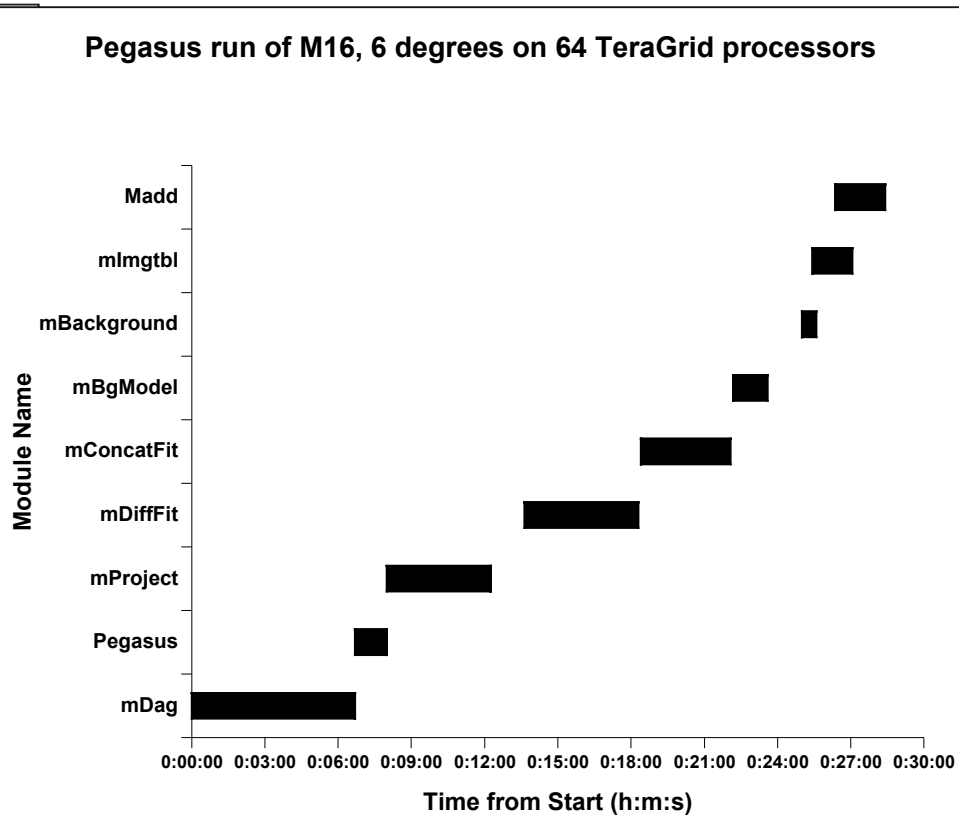
Montage Test Problem

- 2MASS data, 6x6 degree @ m101

Stage	# Tasks	# In	# Out	In (MB)	Out (MB)
mProject	1319	1319	2638	2800	5500
mImgtbl	1	1319	1	2800	0.81
mDiffFit	3883	7766	3883	31000	3900
mConcatFit	1	3883	1	3900	0.32
mBackground	1297	1297	1297	5200	3700

Montage Performance on Large Problem





D. S. Katz, G. B. Berriman, E. Deelman, J. Good, J. C. Jacob, C. Kesselman, A. C. Laity, T. A. Prince, G. Singh, and M.-H. Su, "A Comparison of Two Methods for Building Astronomical Image Mosaics on a Grid," Proceedings of the 7th Workshop on High Performance Scientific and Engineering Computing (HPSEC-05), 2005.

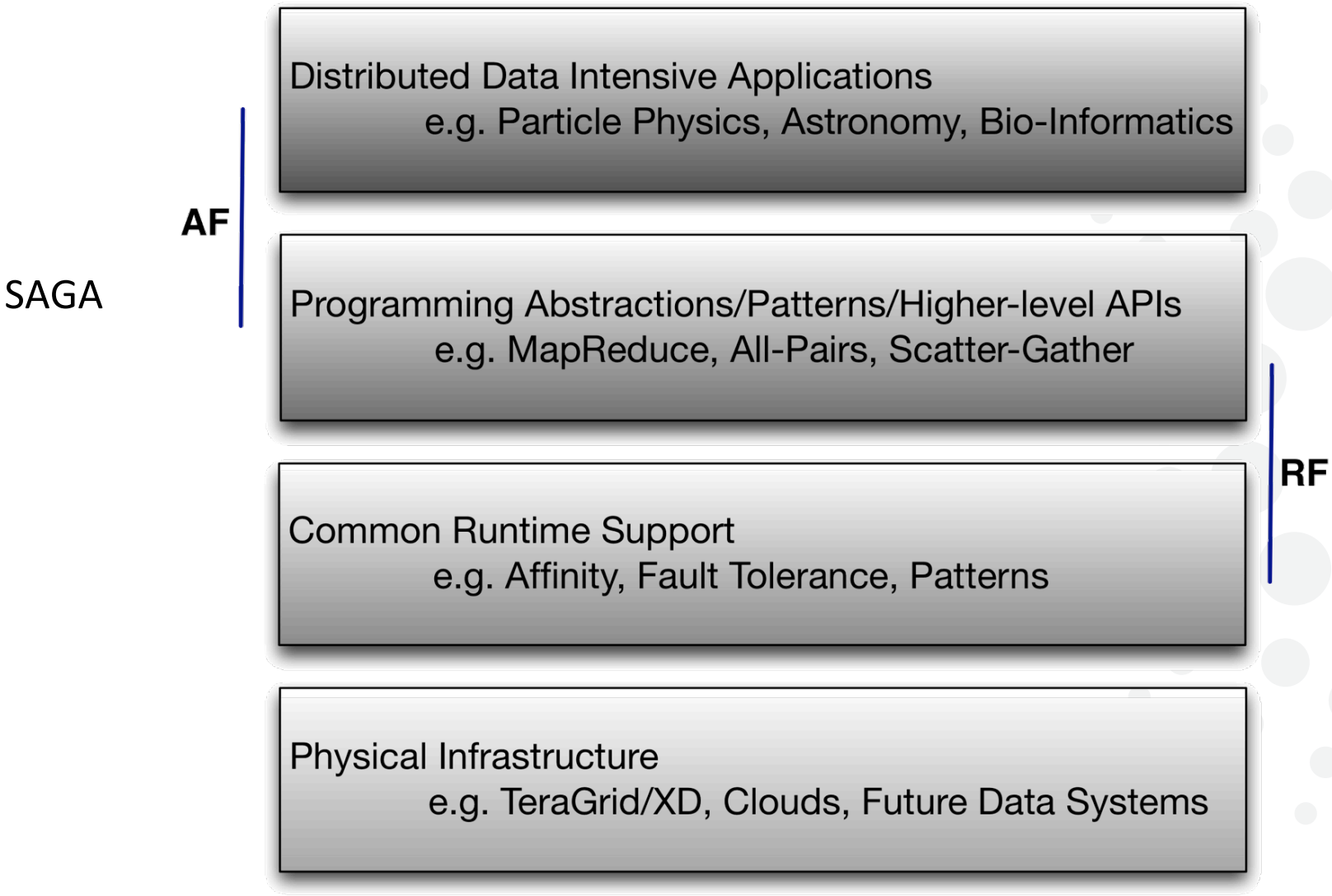
MPI vs. Pegasus Discussion

- Both MPI and Pegasus timings ignore time to start job (queuing delay)
  - MPI: queued job is shell script; Pegasus: queued job is Condor Glide-in
- Tasks are different
  - MPI: run by stages, each is a sequential or parallel job
  - Pegasus - mDag, Pegasus, then application: tasks are clustered by Pegasus/Condor
- Unused resources
  - MPI - trailing tasks in a stage
  - Pegasus - delays of up to 5 seconds from Condor/DAGman
  - Both - sequential job bottlenecks
- Accuracy
  - I/O dominates many tasks; in a multi-user environment, none of this is very precise
- Fault tolerance
  - Pegasus supports DAG restart from previous state, MPI must rollback to previous stage
- Resources
  - Pegasus can use parallel or distributed resources, MPI requires all nodes to have shared filesystem
- Performance
  - MPI finishes in 00:25:33, Pegasus finishes in 00:28:25
- Conclusion: probably use Pegasus in production

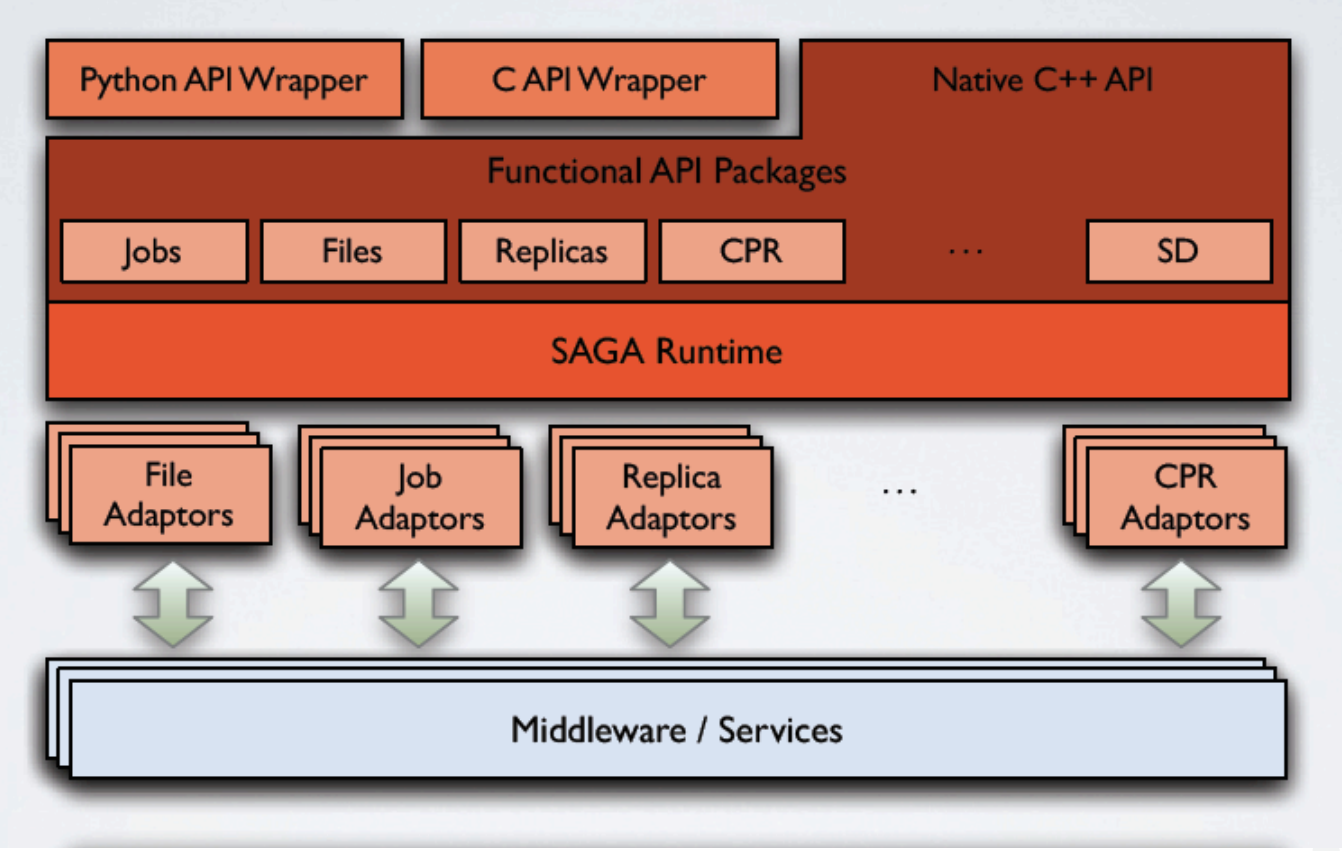
Remaining (Montage/MTC) Challenges

- Implementation and tools are not general
  - Development - could have been simpler
    - mDAG is not a simple code
    - Could have used Pegasus DAX API, but didn't seem any simpler
    - No way to make runtime decisions based on data
  - Deployment and Execution
    - Want to use other infrastructures, such as clouds
    - Want to make runtime decisions based on resources
      - Provide better fault tolerance than rescue DAG
    - Want to control resources (e.g., networks)

Frameworks as a solution?

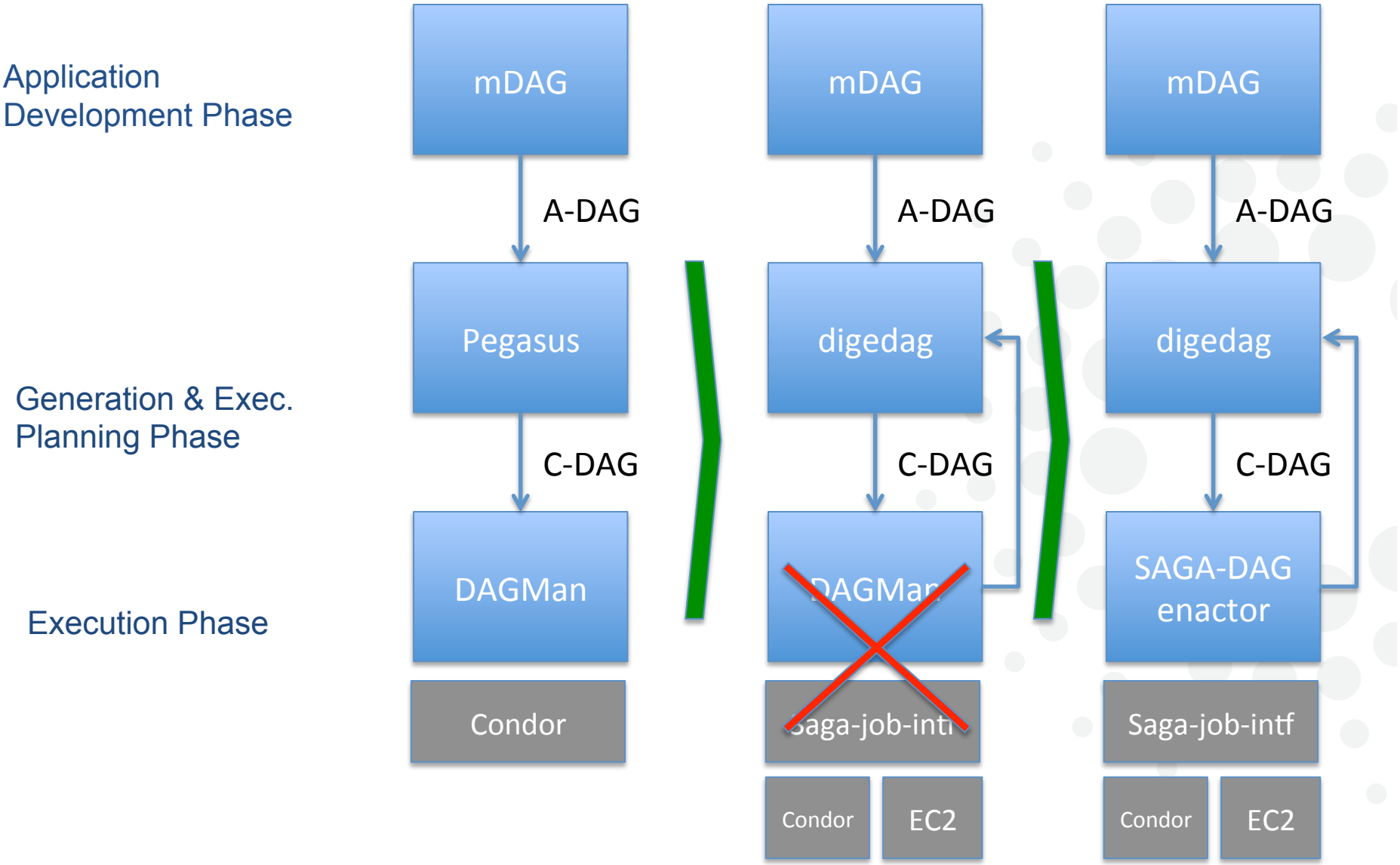


SAGA: In a thousand words..



http://saga-project.github.io/

DAG-based Workflow Applications: Extensibility Approach



SAGA-Montage Proof-of-concept

- Tests run
  - toy problem: m101 tutorial (0.2° x 0.2°)
  - Useful for trying things – functionality
- digedag used for planning
  - For this problem, takes about about 0.2 s – same as Pegasus
- Runs
  - Local submission using fork
  - Local submission using ssh/SAGA
  - Local submission using Condor/SAGA
    - Local submission using 2 of above 3 and 3 of above 3
  - Queen Bee submission using ssh/SAGA
  - EC2 submission using AWS/SAGA
    - Remote submission to Queen Bee and EC2 using both ssh/SAGA and AWS/SAGA
    - Local/remote submission to local, Queen Bee, and EC2 using fork, ssh/SAGA, and AWS/SAGA
- Unstudied(yet) issues:
  - Need better understanding of application performance
  - Tradeoffs between use of MPI components or sequential components?

A. Merzky, K. Stamou, S. Jha, D. S. Katz, "A Fresh Perspective on Developing and Executing DAG-Based Distributed Applications: A Case-Study of SAGA-based Montage," Proceedings of 5th IEEE International Conference on e-Science, 2009.

More MTC (Montage) Issues

- Why is an explicit DAG needed?
- User customization
  - mDAG builds a DAG for a standard workflow
  - What about a non-standard workflow?
    - Experimenting with different options
      - Build a set of plates with and without background rectification
    - Changing a module
      - mAdd uses mean for co-addition, could use median or count
  - Changing mDAG.c is not reasonable for most users
- A scripted version of Montage may be better in some cases, and may be more natural for some users

- Swift is designed to compose large parallel workflows, from serial or parallel application programs, to run fast and efficiently on a variety of platforms
  - A parallel scripting system for Grids and clusters for loosely-coupled applications - application and utility programs linked by exchanging files
  - Easy to write: simple high-level C-like functional language, allows small Swift scripts to do large-scale work
  - Easy to run: contains all services for running Grid workflow, in one Java application
    - Works on multicore workstations, HPC, Grids (interfaces to schedulers, Globus, ssh)
  - Fast: efficient, scalable and flexible execution
    - Swift/K scaling: O(1M) tasks
    - Collective data management being developed to optimize I/O
    - Swift/T in development with >10x performance improvements
- Used in neuroscience, proteomics, molecular dynamics, biochemistry, economics, statistics, and more
- http://www.ci.uchicago.edu/swift

M. Wilde, N. Hategan, J. M. Wozniak, B. Clifford, D. S. Katz, I. Foster, "Swift: A language for distributed parallel scripting," Parallel Computing, v.37(9), pp. 633-652, 2011.

Example Driver Code(s)

```
import {…}

MosaicData header <"header.hdr">;
Table images_tbl <"images.tbl">;
Image mosaic <"final/mosaic.fits">;
JPEG mosaic_jpg <"final/mosaic.jpg">;
Image projected_images[];
Table difference_tbl <"diffs.tbl">;
Table fits_images_tbl <"fits.tbl">;
Table rectification_tbl <"rectification.tbl">;
Table stat_tbl <"stats.tbl">;
Image difference_images[];
Image rectified_images[];


Image raw_image_files[] <filesys_mapper; location = "raw_dir", suffix = ".fits">;
projected_images = mProjectBatch( raw_image_files, header );
images_tbl = mImgtbl( projected_images );
difference_tbl = mOverlaps( images_tbl );
difference_images = mDiffBatch( difference_tbl, header );
fits_images_tbl = mFitBatch( difference_images, difference_tbl );
rectification_tbl = mBgModel( images_tbl, fits_images_tbl );
rectified_images = mBgBatch( projected_images, images_tbl, rectification_tbl );
mosaic = mAdd( rectified_imagesORprojected_images, images_tbl, header );
mosaic_jpg = mJPEG(mosaic );
```

Black code builds mosaic;
Red code needed for background rectification

Example Wrapper Code

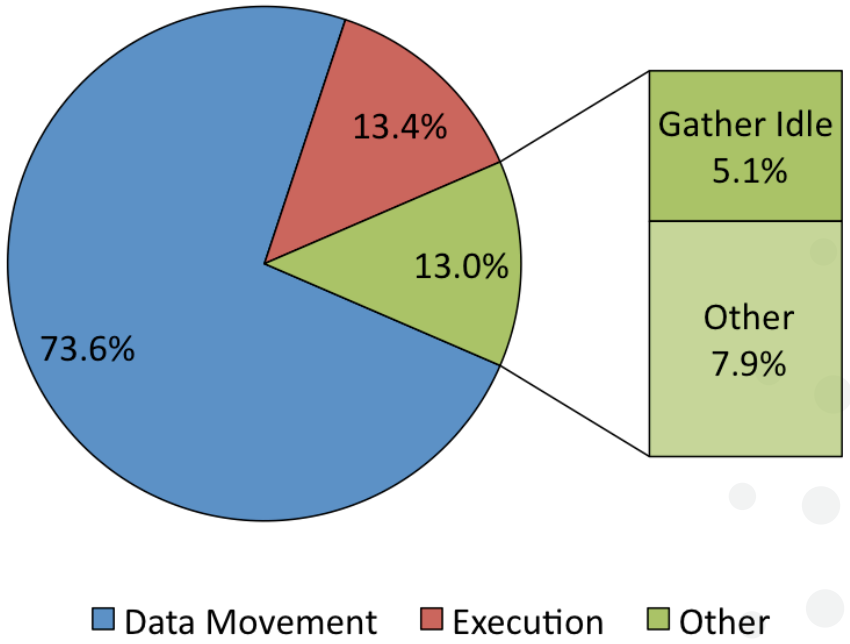
```
( Image proj_imgs[] ) mProjectBatch( Image raw_imgs[], MosaicData hdr )
{
    foreach img, i in raw_imgs
    {
        Image proj_img <regexp_mapper; source = @img, match = ".*\\/(.*)", transform =
    "proj_dir/proj_\\1">;
        proj_img = mProject( img, hdr );
        proj_imgs[ i ] = proj_img;
    }
}

app ( Image proj_img ) mProject( Image raw_img, MosaicData hdr )
{
    mProject "-X" @raw_img @proj_img @hdr;
}
```

Further Swift-Montage Work

- Better understand Montage performance
- Improve Swift
  - To improve Montage performance at the runtime level
  - Using AMFORA (was: AME & AMFS)

Z. Zhang, D. S. Katz, J. Wozniak, A. Espinosa, I Foster, "Design and Analysis of Data Management in Scalable Parallel Scripting," Proceedings of the International Conference for High Performance Computing, Networking, Storage and Analysis (SC12), 2012

Swift-Montage Performance

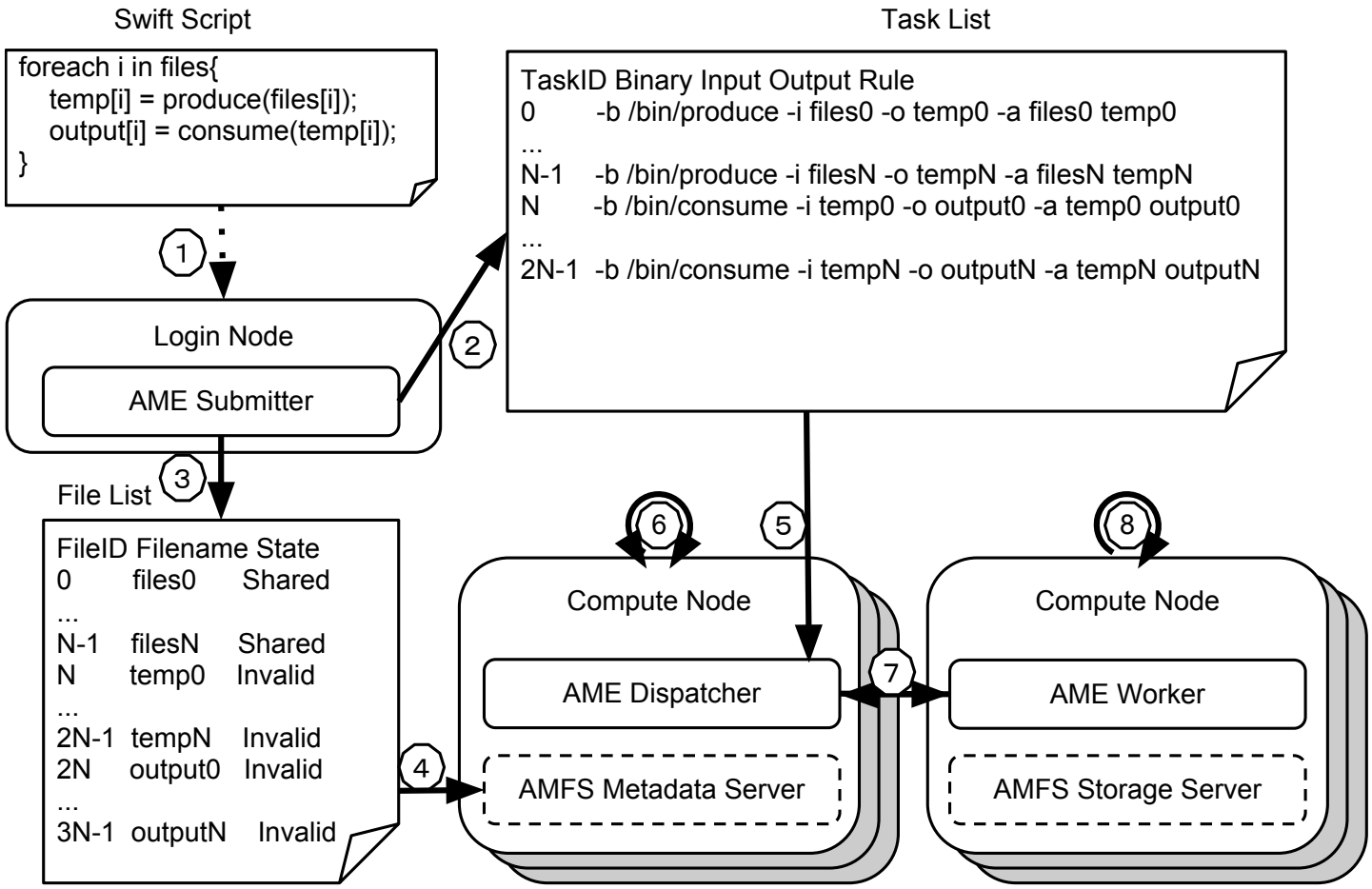


Execution time distribution of Montage test problem on 512 BG/P CPU cores

- Challenge: reduce non-execution time

AMFORA

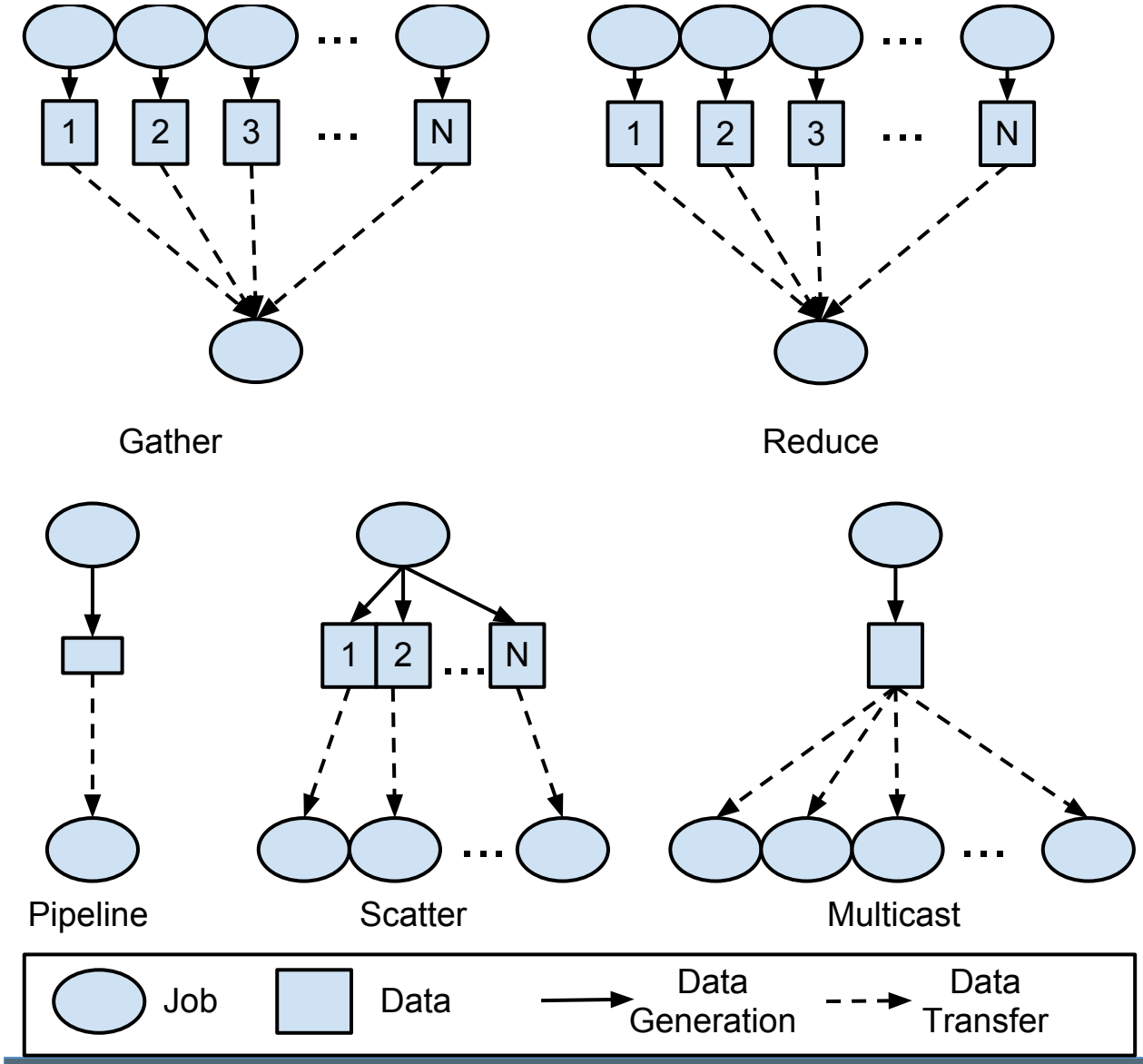
- AMFORA: an Any-scale MTC Engine and MTC runtime File System
  - https://github.com/zhaozhang/amfora



General Improvements

- Base Case: Staging
  - Automatically move input files from GPFS to local RAM disk
  - Read/write to local RAM disk
  - Automatically move output files from local RAM disk to GPFS
  - Runs in 45% of the time as MPI implementation, which reads and writes files on GPFS
- Data cache
  - Automatically move input files from local RAM disk (where produced) to local RAM disk (where needed)
- Data aware scheduling
  - Launch task where input data is (on local RAM disk)

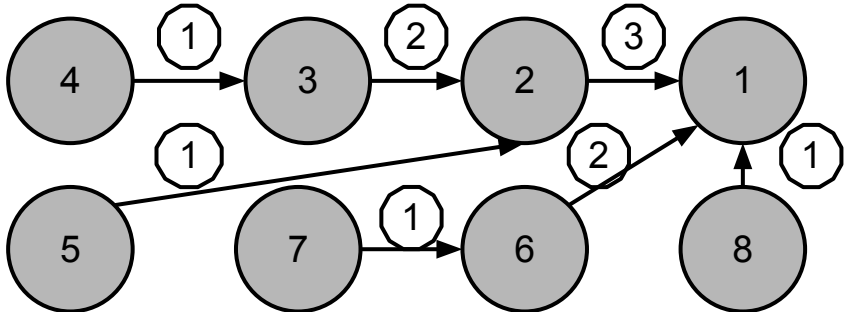
Dataflow Patterns



- Goals:
  - Identify at runtime
  - Apply optimizations

File Pattern Improvements

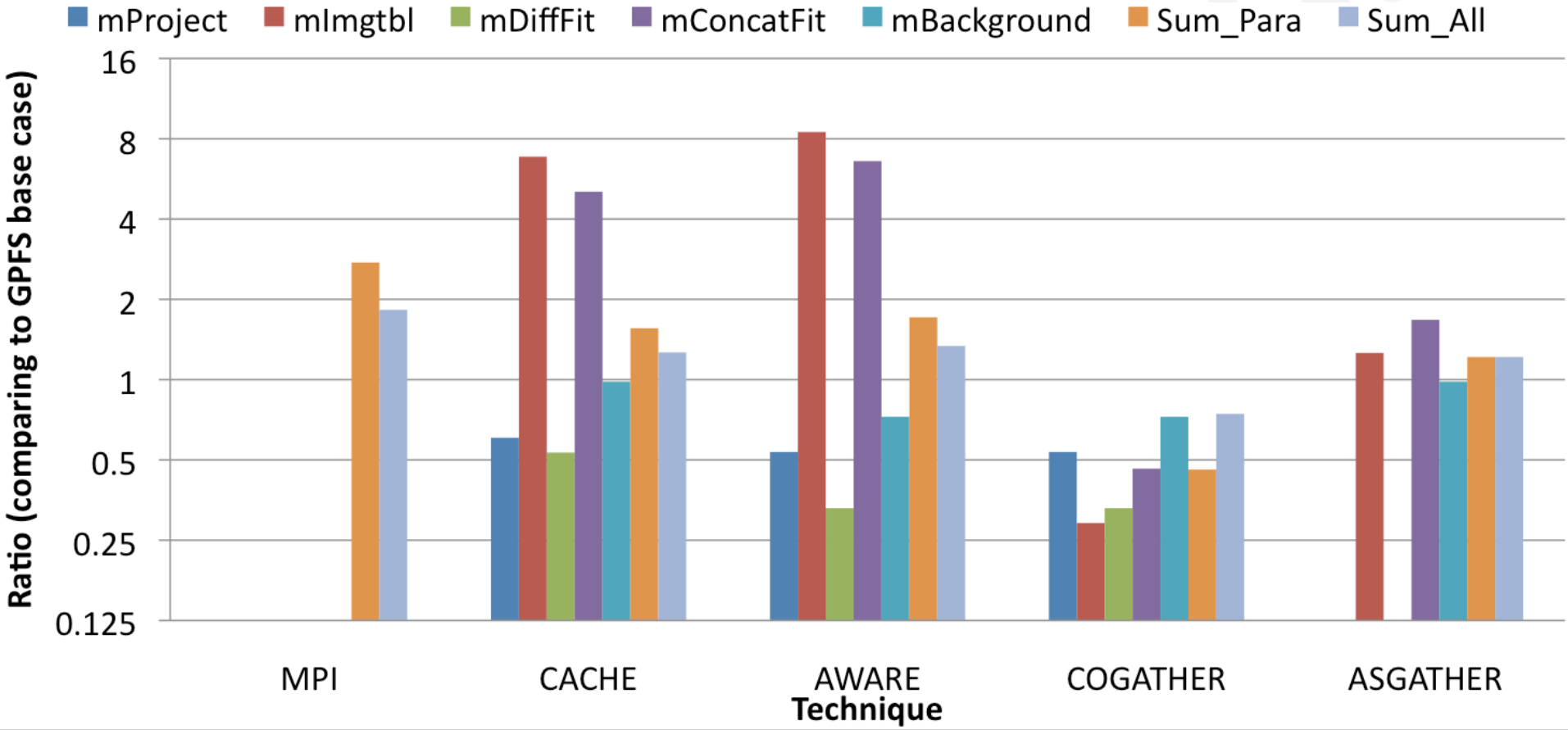
- Collective gather
  - Rather than moving files one at a time from sources to a single destination, create a minimum spanning tree and collectively move the files in $\log_2(n)$ stages



  - Works well when transfer time is dominated by network latency, or files are available at once
- Asynchronous gather
  - Worker that will run task requests all source files
  - If available, worker fetches the file; if not, worker asks for notification
  - Allows overlapping of computation and communication
  - Works well when transfer time is dominated by network bandwidth, or files are available gradually

Montage - AME & AMFS Performance

- Those bars that are less than 1 show improvements.
- GPFS base case refers to staging input/output data from/to GPFS
  - Already 45% improvement over MPI



Montage work by others

- C. Hoffa, G. Mehta, E. Deelman, T. Freeman, K. Keahey, B. Berriman, J. Good, "On the Use of Cloud Computing for Scientific Workflows," SWBES08: Challenging Issues in Workflow Applications, 2008
  - Ran Montage on virtual and physical machines, including a private cloud-like system
- Montage used as prototype application by teams involved in ASKALON, QoS-enabled GridFTP, SWIFT, SCALEA-G, VGrADS, etc.

Montage Summary

- Montage is a custom astronomical image mosaicking service that emphasizes astrometric and photometric accuracy
- Public release, available for download at the Montage website: http://montage.ipac.caltech.edu/
- MPI version of Montage:
  - Baseline performance
  - Requires a set of processors with a shared file system
- Pegasus/DAGman version of Montage:
  - Almost equivalent performance to MPI version for large problems
  - Built-in fault tolerance
  - Can use multiple sets of processors
- SAGA/digedag version of Montage:
  - Starts to address IDEAS: Interoperability, Distributed Scale-Out, Extensibility, Adaptivity, Simplicity
  - Have shown that this can be done, but haven't gotten too far yet
- Swift version of Montage
  - Flexible scripting, no explicit DAG
  - By modifying Swift runtime, can get very good parallel performance

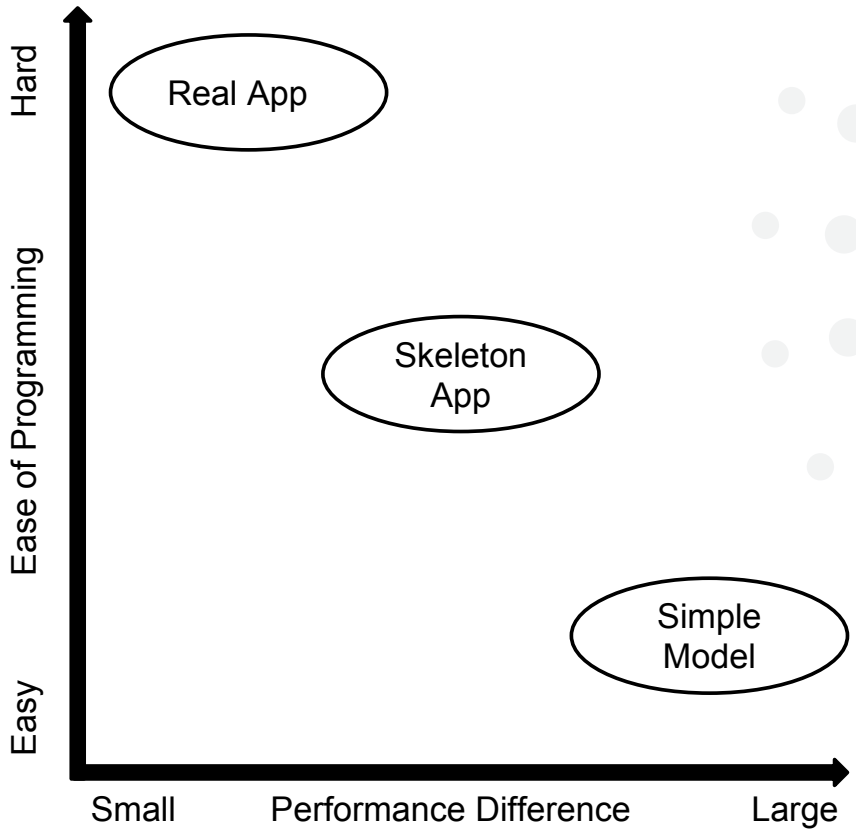
Application Skeletons

- Goal: Represent important applications
- With only the information needed to run the application in a distributed context
- But without requiring building the actual application, or finding data files, ...
- Compactly, e.g., few parameters for Bag of Tasks
  - Input data size (could be generalized as a distribution)
  - Output data size (could be generalized as a distribution, or a function of input data size)
  - Computing time (could be generalized as a distribution, or a function of input data size)
  - Number of tasks

Z. Zhang, D. S. Katz, "Application Skeletons: Encapsulating MTC Application Task Computation and I/O," Best paper of 6th Workshop on Many-Task Computing on Clouds, Grids, and Supercomputers (MTAGS), co-located with SC 2013.

Challenge

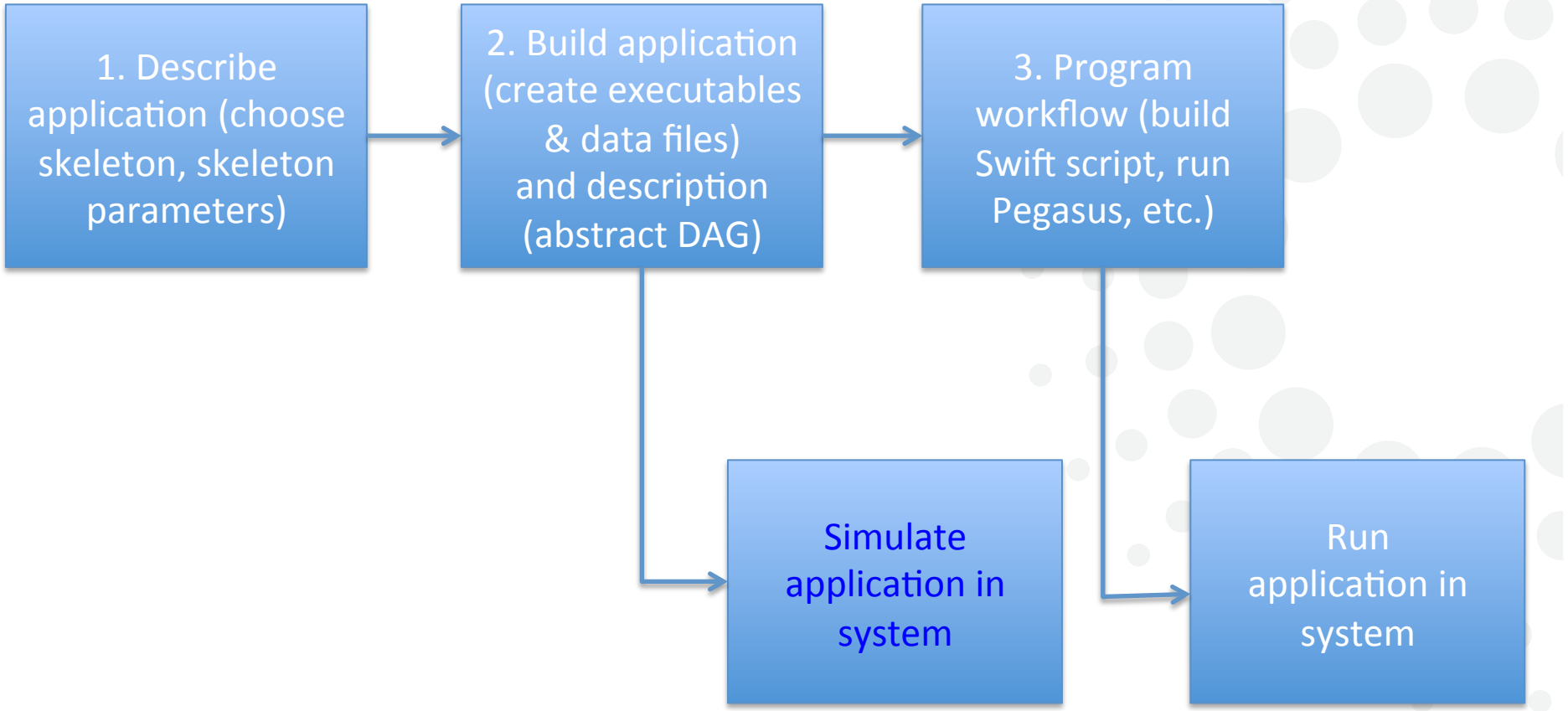
- Balance the easy of programming and usage with the performance gap between Skeleton applications and real applications



Types of Applications that can be Represented by Skeletons

- Bag of Tasks
  - Set of independent tasks
  - Represents MG-RAST, TIP, science gateway, ...
- MapReduce (distributed)
  - Set of map tasks, then reduce tasks
  - Represents HEP histograms, genomics/metagenomics, ...
- Iterative MapReduce
  - Represents clustering, linear algebra
- Campaign
  - Similar to iterative bag of tasks, but tasks can change from one iteration to the next
  - Represents Kalman filtering
- Multi-stage Workflow (most general for now)
  - Represents Montage, ...
- Concurrent tasks
  - Represents fusion simulation, climate simulation (2nd phase)

Role and Use of Application Skeletons



Skeleton Montage vs. Real Montage

	mProject	mImgtbl	mOverlaps	mDiffFit	mConcatFit	mBgModel	mBackground	mAdd	Total
Montage	282.3	139.7	10.2	426.7	60.1	288.0	107.9	788.8	2103.7
Skeleton	281.8	136.8	10.0	412.5	59.2	288.1	106.2	781.8	2076.4
Error	-0.2%	-2.1%	-0.2%	-3.3%	-1.5%	0.03%	-1.6%	-0.9%	-1.3%

- Similar results for BLAST, CyberShake

Z. Zhang, D. S. Katz, "Using Application Skeletons to Improve eScience Infrastructure," submitted to IEEE eScience 2014.

Skeleton Questions/TODOs

- Explore other methods for specifying skeleton parameters
- Define compute time in resource independent manner
- Add to details of tasks
  - For tasks that are internally parallel: number of internal components, internal communication requirements (to be used for mapping the task to appropriate resources)
- Turn skeletons into a 'open source' project – more contributors, more users

Further Work

- Goal: Use MTC paradigm to develop parallel and distributed data-intensive scientific applications to utilize a broad range of systems, with the flexibility and performance that scientific applications demand, using heritage (legacy) software
- Possible solutions: Frameworks (e.g. SAGA), MTC runtime (e.g. Swift, AMFORA)
- Application challenges
  - What are the application components?
  - How are they coupled?
  - How is functionality expressed/exposed?
  - How is coordination handled?
  - How to support layering, ordering, encapsulations of components
- Framework/runtime challenges
  - Coordinate data & computing; use task placement; and optimize I/O for best performance
  - Support range of architectures and fault-tolerance
  - Support runtime (dynamic) scheduling (including networks), including use of information systems
  - Avoid duplicating things that work – e.g., Pegasus's planner
- System challenges
  - Tradeoff of costs & rewards: balance user & system utility (time to solution vs. system utilization)
  - Impact future systems through knowledge gained
- Tool challenges
  - Application skeletons support framework/runtime & systems research
  - What other tools are needed?

Credits

- Montage (funded by NASA ESTO): Bruce Berriman, John Good, Joseph C. Jacob, Daniel S. Katz, Anastasia Laity, Nathaniel Anagnostou, Attila Bergou, Roy Williams, Thomas Prince
- Grid Montage: Ewa Deelman, Carl Kesselman, Gurmeet Singh, Mei-Hui Su
- DPA Theme (funded by UK e-Science Institute): Shantenu Jha, Daniel S. Katz, Manish Parashar, Omer Rana, Jon Weissman
- SAGA: SAGA Team, http://saga-project.org/
- SAGA Montage: Andre Merzky, Katerina Stamou, Shantenu Jha, Daniel S. Katz
- Swift: Swift Team, http://www.ci.uchicago.edu/swift
- Swift Montage: Jon Monette, Zhao Zhang, Daniel S. Katz, Michael Wilde
- AMFORA (AME/AMFS): Zhao Zhang, part of (DOE-funded) ExM Project, led by Michael Wilde, Matei Ripeanu, Daniel S. Katz
- Application Skeletons: Zhao Zhang, part of (DOE-funded) AIMES Project, led by Shantenu Jha, Jon Weissman, Daniel S. Katz