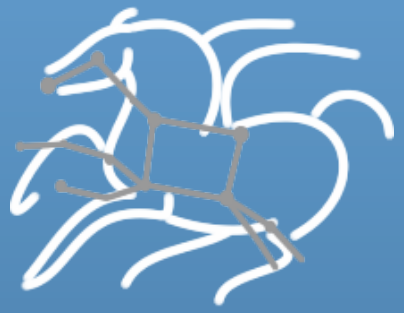


Introduction to Scientific Workflows and Pegasus

Karan Vahi

Science Automation Technologies Group

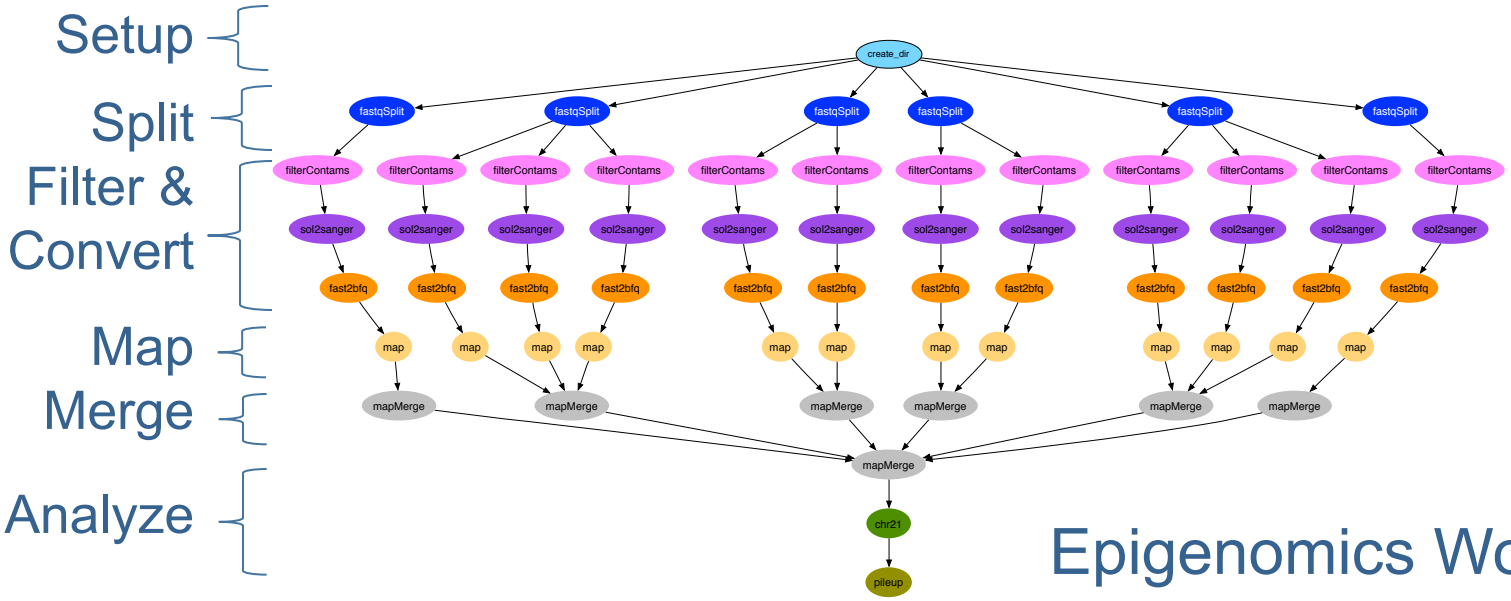
USC Information Sciences Institute

Outline

- Introduction to Scientific Workflows and Pegasus
- Running Workflows through Pegasus
  - Composition
  - Submission
  - Monitoring
  - Debugging
- Advanced Features
  - Data Cleanup
  - Data Reuse
  - Hierarchal Workflows
  - Job Clustering

Scientific Workflows

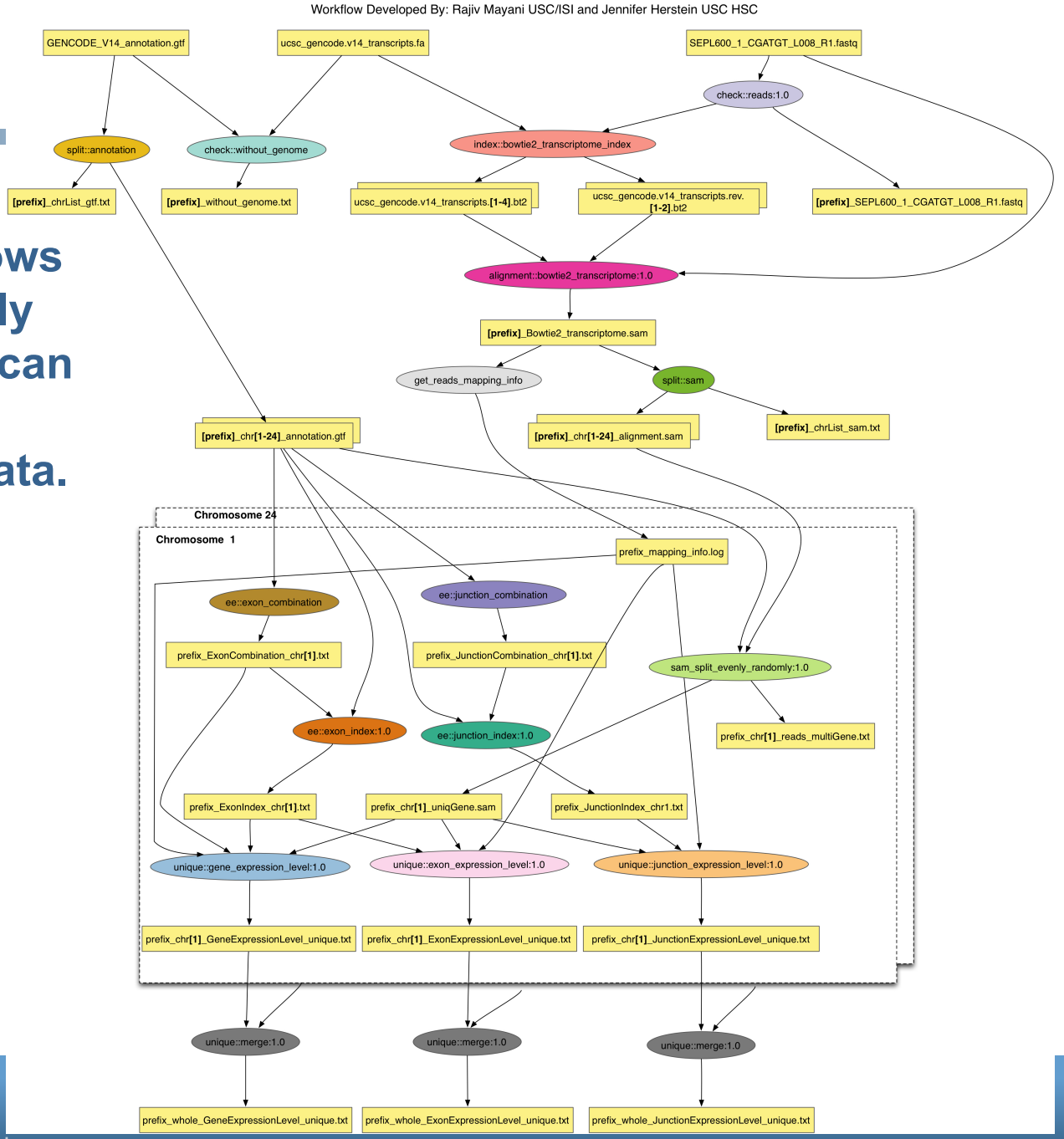
- Orchestrate complex, multi-stage scientific computations
- Often expressed as directed acyclic graphs (DAGs)
- Capture analysis pipelines for sharing and reuse
- Can execute in parallel on distributed resources



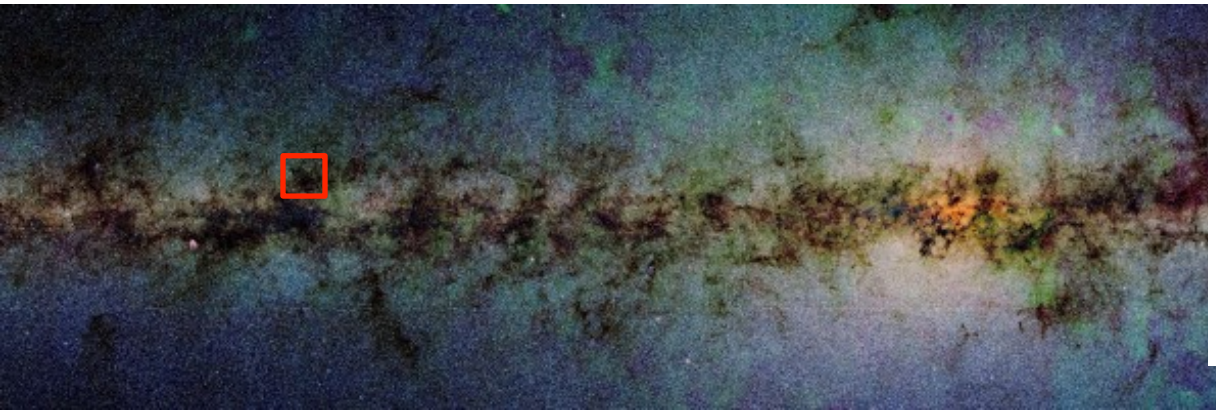
Epigenomics Workflow

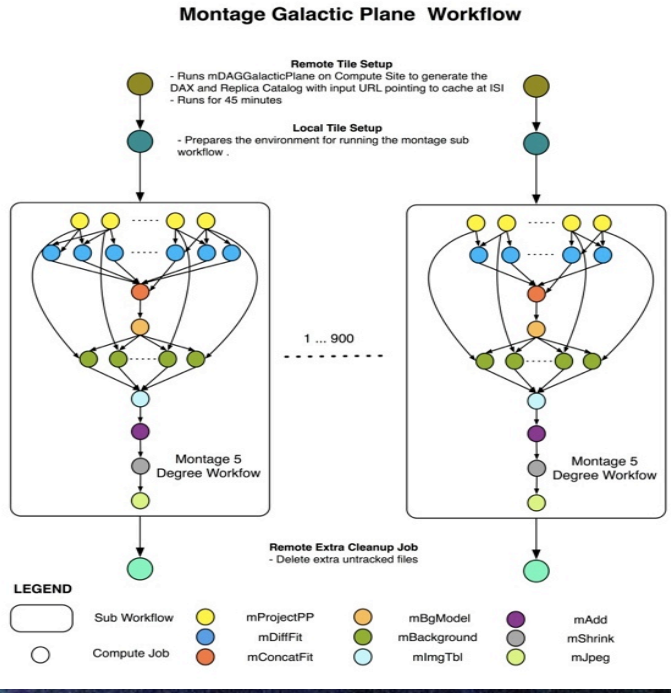
Workflows can be simple!

Some workflows are structurally complex and can use large amounts of data.



Some workflows are large-scale and data-intensive





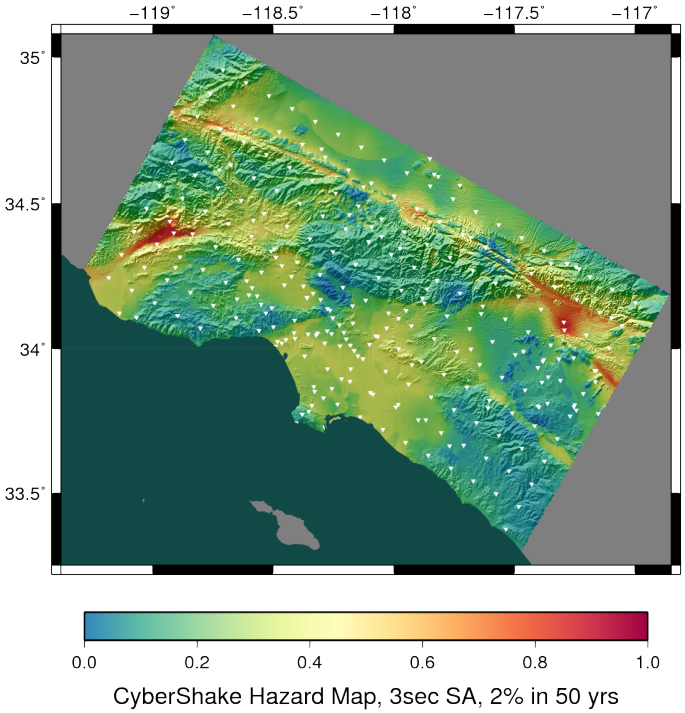
John Good (Caltech)

- **Montage Galactic Plane Workflow**
  - 18 million input images (~2.5 TB)
  - 900 output images (2.5 GB each, 2.4 TB total)
  - 10.5 million tasks (34,000 CPU hours)

× 17

- **Need to support hierarchical workflows and scale**

Some workflows couple large-scale simulations with data analysis



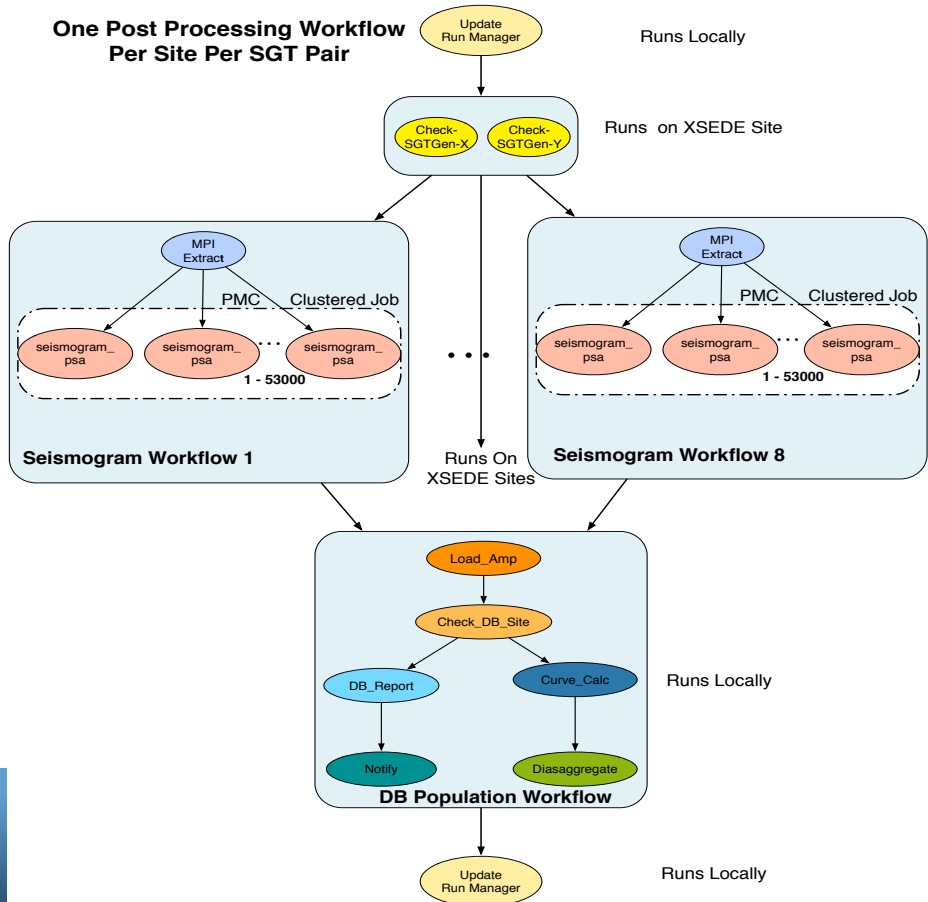
CyberShake Hazard Map, 3sec SA, 2% in 50 yrs

2014: 286 Sites, 4 models

- **Each site = one workflow**
- **Each workflow has 420,000 tasks in 21 jobs**

CyberShake PSHA Workflow

- Builders ask seismologists: "What will the peak ground motion be at my new building in the next 50 years?"
- Seismologists answer this question using Probabilistic Seismic Hazard Analysis (PSHA)



Why Scientific Workflows?

- Automate complex processing pipelines
- Support parallel, distributed computations
- Use existing codes, no rewrites
- Relatively simple to construct
- Reusable, aid reproducibility
- Can be shared with others
- Capture provenance of data

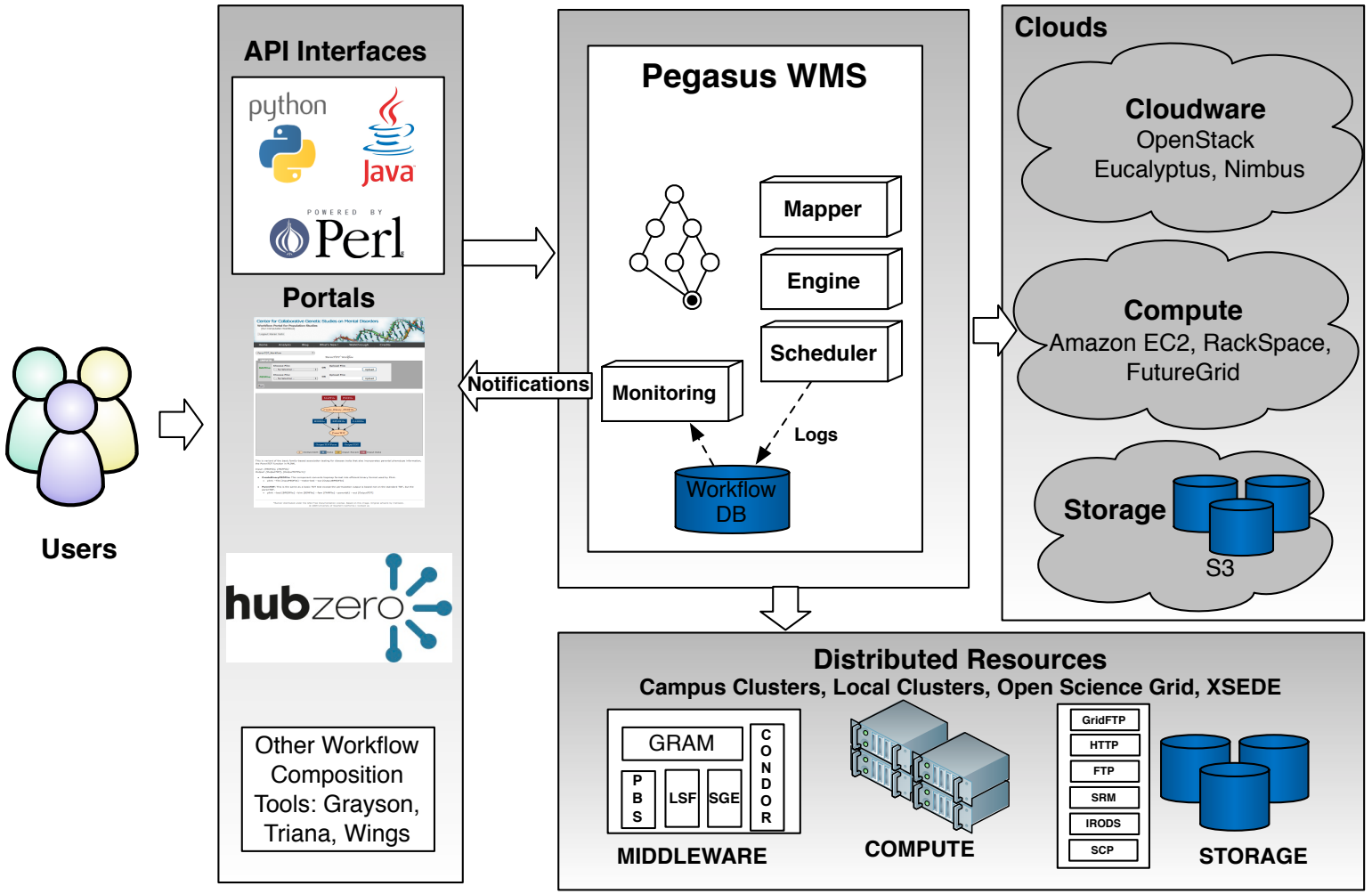
Scientific Workflow Challenges

- **Portability**
  - How can you run a pipeline on Amazon EC2 one day, and a PBS cluster the next?
- **Data Management**
  - How do you ship in the small/large amounts data required by your pipeline?
  - Different protocols for different sites: Can I use SRM? How about GridFTP? HTTP and Squid proxies?
  - Can I use Cloud based storage like S3 on EC2?
- **Debug and Monitor Computations.**
  - Users need automated tools to go through the log files
  - Need to correlate data across lots of log files
  - Need to know what host a job ran on and how it was invoked
- **Restructure Pipelines for Improved Performance**
  - Short running tasks?
  - Data placement?

Pegasus
Workflow Management System (est. 2001)

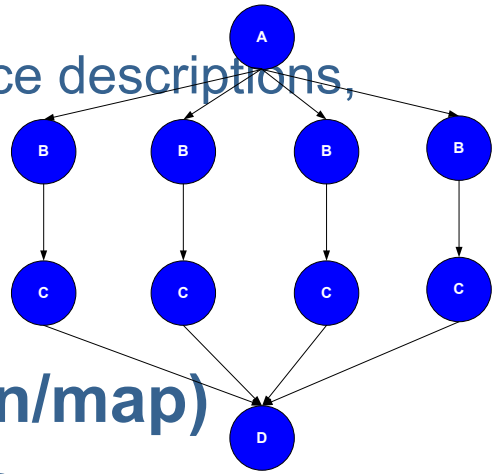
- A collaboration between USC and the Condor Team at UW Madison (includes DAGMan)
- Maps a resource-independent "abstract" workflow onto resources and executes the "executable" workflow
- Used by a number of applications in a variety of domains
- Provides reliability—can retry computations from the point of failure
- Provides scalability—can handle large data and many computations (kbytes-TB of data, 1-10^6 tasks)
- Infers data transfers, restructures workflows for performance
- Automatically captures provenance information
- Can run on resources distributed among institutions, laptop, campus cluster, Grid, Cloud

Pegasus WMS Environment



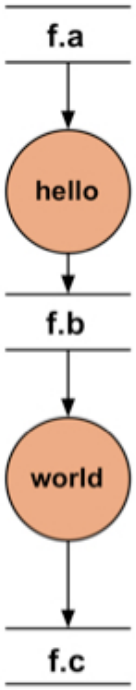
Pegasus Workflow Management System

- **Abstract Workflows - Pegasus input workflow description**
  - Workflow “high-level language”
  - Only identifies the computation, devoid of resource descriptions, devoid of data locations
  - File Aware



- **Pegasus is a workflow “compiler” (plan/map)**
  - Target is DAGMan DAGs and Condor submit files
  - Transforms the workflow for performance and reliability
  - Automatically locates physical locations for both workflow components and data
  - Collects runtime provenance

DAX – XML format to describe Abstract Workflows



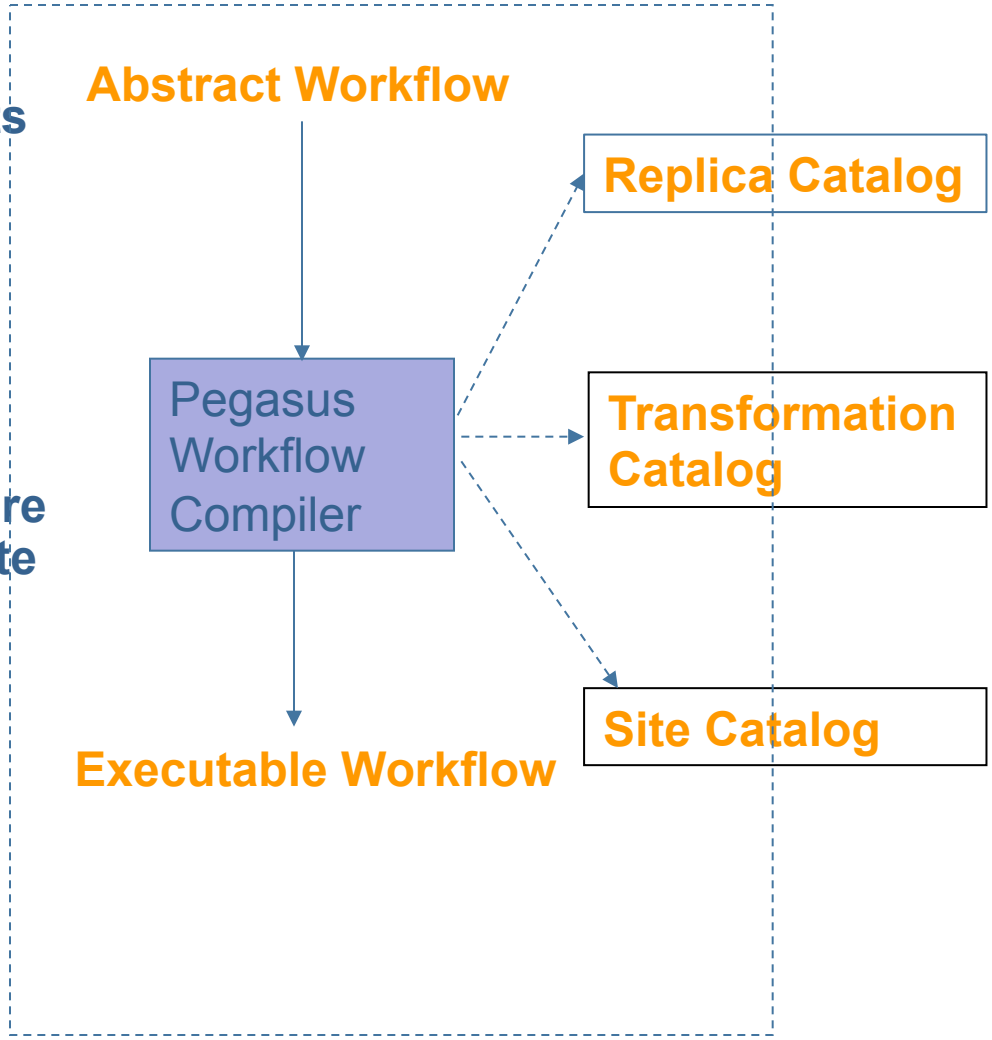
Abstract Workflow

```
<?xml version="1.0" encoding="UTF-8"?>
<adag version="3.4" name="hello-world" index="0" count="1">

<!-- Section: Job's, DAX's or Dag's - Defines a JOB or DAX or
DAG (Atleast 1 required) -->

   <job id="j1" namespace="pegasus" name="hello" version="4.0">
      <argument>-a hello -T 60 -i  <file name="f.a"/>
        -o <file name="f.b"/>
    </argument>
    <uses name="f.a" link="input" transfer="true"
register="true"/>
     <uses name="f.b" link="output" transfer="false"
register="false"/>
   </job>

   <job id="j2" namespace="pegasus" name="world" version="4.0">
      <argument>-a world -T 60 -i  <file name="f.b"/>
            -o  <file name="f.c"/>
      </argument>
    <uses name="f.b" link="input" transfer="true"
register="true"/>
     <uses name="f.c" link="output" transfer="false"
register="false"/>
   </job>

<!-- Section: Dependencies - Parent Child relationships (can be
empty) -->

   <child ref="j2">
      <parent ref="j1"/>
   </child>

</adag>
```

USC Viterbi School of Engineering

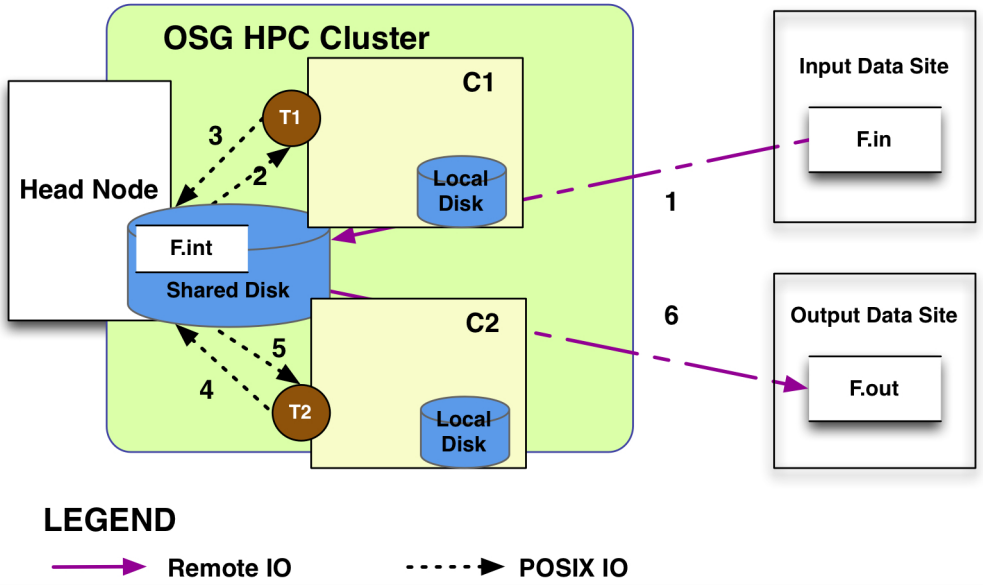
Abstract to Executable Workflow Mapping - Discovery

- **Data**
  - Where do the input datasets reside?
- **Executables**
  - Where are the executables installed ?
  - Do binaries exist somewhere that can be staged to remote grid sites?
- **Site Layout**
  - What does a execution site look like?



How does Pegasus view a compute resource as?

- For Pegasus a compute resource or a site is associated with the following
  - An entry point or a scheduler contact to submit jobs to e.g PBS/LSF/Condor
  - File servers to stage data to the cluster
  - Different types of directories on the site
    - Shared-scratch - shared across all the worker nodes in the site
    - Local – a directory/filesystem local to the node where a job executes
  - Site wide information like environment variables to be set when a job is run.



Site Catalog

- Stores details about each target execution/storage site
  - Job submission endpoints (GRAM URL, etc.)
  - Paths to storage/scratch directories
  - Data transfer services (GridFTP servers, etc.)
  - Paths to credentials (X509 proxy, ssh key, etc.)
  - Site-level configuration (environment variables, etc.)
  - “local” site is special—refers to submit host

```
<!-- Example site catalog -->
<sitecatalog>
  <site handle="example" arch="x86_64" os="LINUX">
    <grid type="gt5" contact="example.isi.edu/jobmanager-fork” jobtype="auxillary"/>
    <grid type="gt5" contact="example.isi.edu/jobmanager-pbs” jobtype="compute"/>
    <directory type="shared-scratch" path="/scratch">
      <file-server operation="all" url="gsiftp://example.isi.edu/scratch"/>
    </directory>
    <profile namespace="env" key="GLOBUS_LOCATION">/usr/local/globus</profile>
    <profile namespace=“pegasus” key=“style”>globus</profile>
    <profile namespace=“pegasus” key=“X509_USER_PROXY”>/tmp/x509_u40001</profile>
  </site>
</sitecatalog>
```


Transformation Catalog

- **Maps transformations to executables on each site**
  - Physical path or URL of executable and dependent data/ configuration files
  - Executable characteristics (OS, architecture, glibc, etc.)
  - Job-level configuration (e.g. environment variables, profiles)

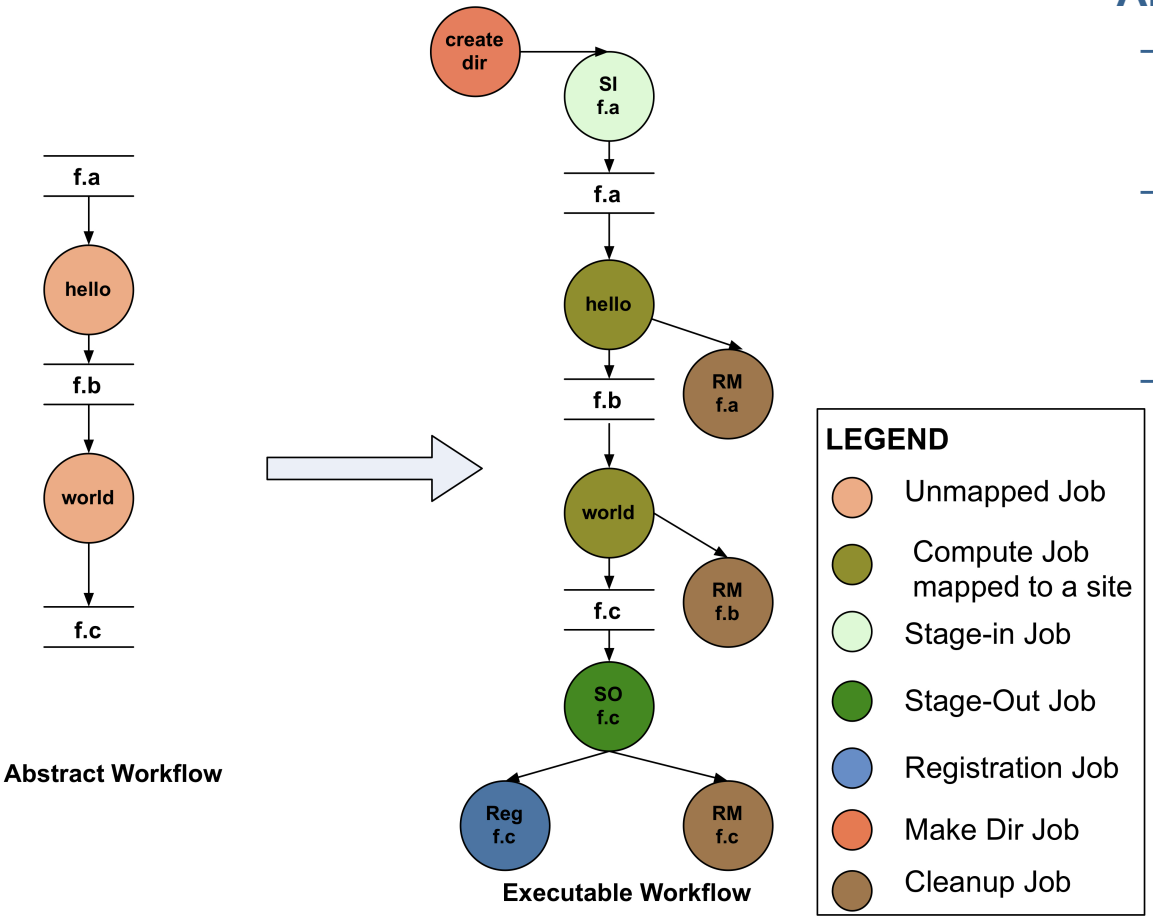
```
# Example transformation catalog
tr example::date {

  profile env ”TZ" ”America/Los_Angeles”

  site example {
    pfn "/bin/date"
    os "linux"
    arch "x86_64"
    type "INSTALLED"
  }
}
```


Replica Catalog

- Maps logical files to physical files
  - LFN (name) to PFN (path or URL)
  - Mappings annotated with metadata (e.g. site/pool, size, etc.)
- Enables Pegasus to choose “best” replica (replica selection phase of planner)
- Where Pegasus registers workflow output locations
- Support file-based or DB-based RC (also callout)

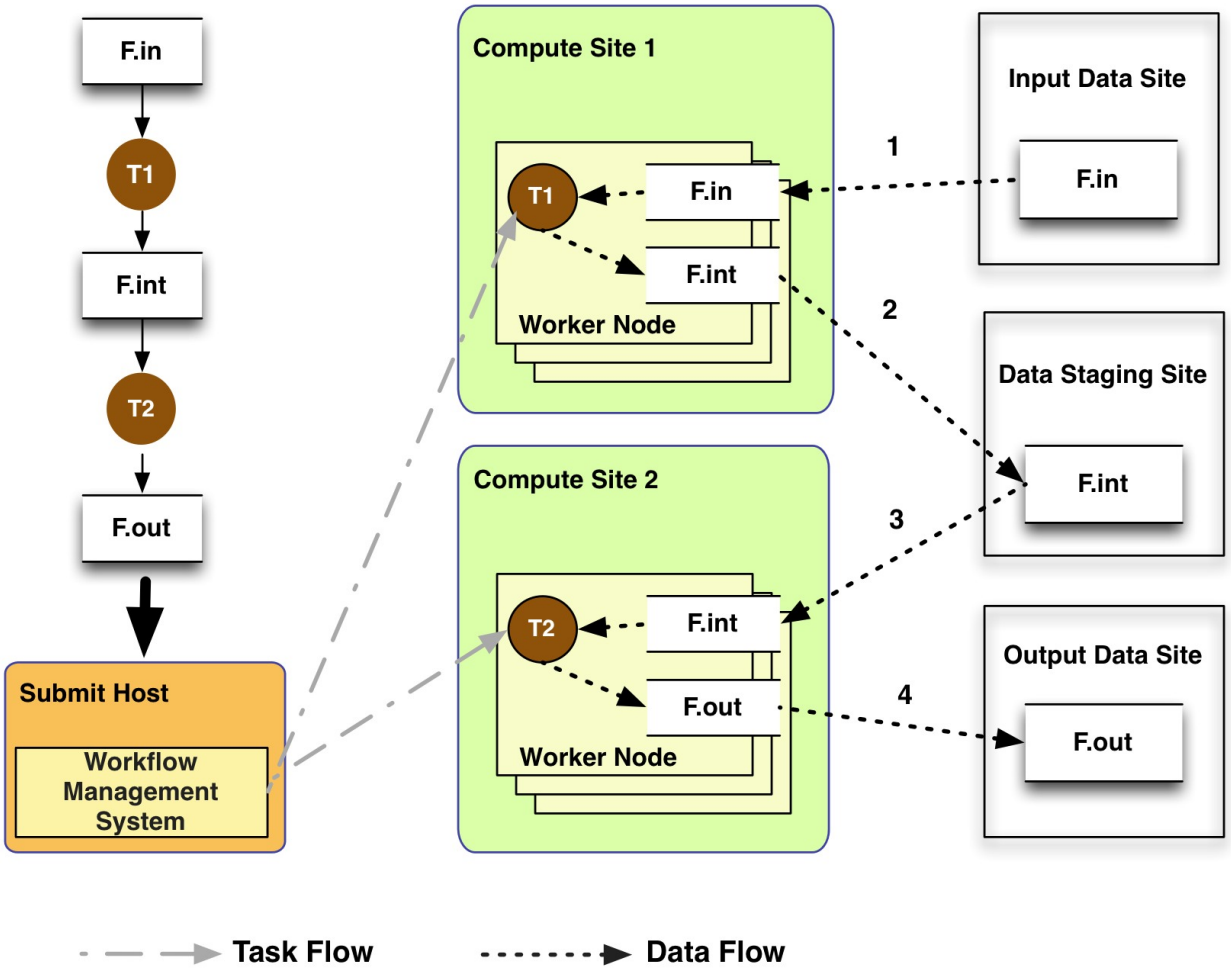
```
# Example replica catalog
f.1    gsiftp://example.isi.edu/inputs/f.1 pool=”example”
f.1    file:///inputs/f.1                  site=”example”
f.2    file:///inputs/f.2                  site=“example”
f.2    file:///inputs/f.2                  site=“local”
```


Abstract to Executable Workflow Mapping



- **Abstraction provides**
  - **Ease of Use (do not need to worry about low-level execution details)**
  - **Portability (can use the same workflow description to run on a number of resources and/or across them)**
  - **Gives opportunities for optimization and fault tolerance**
    - **automatically restructure the workflow**
    - **automatically provide fault recovery (retry, choose different resource)**

General Workflow Execution Model



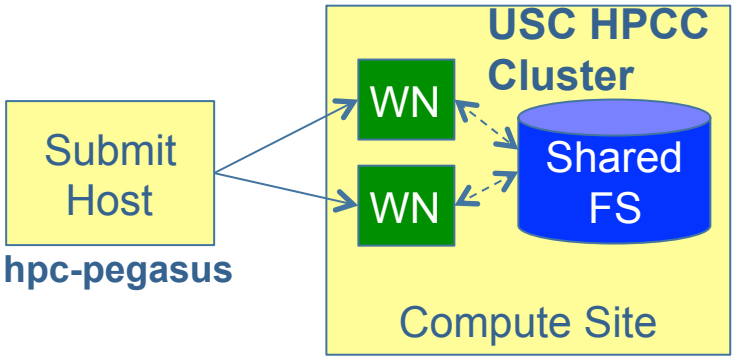
- Most of the tasks in scientific workflow applications require POSIX file semantics
  - Each task in the workflow opens one or more input files
  - Read or write a portion of it and then close the file.
- Data Staging Site can be the shared filesystem on the compute cluster!

- Input Data Site, Compute Site and Output Data Sites can be co-located
  - Example: Input data is already present on the compute site.

Supported Data Staging Approaches - I

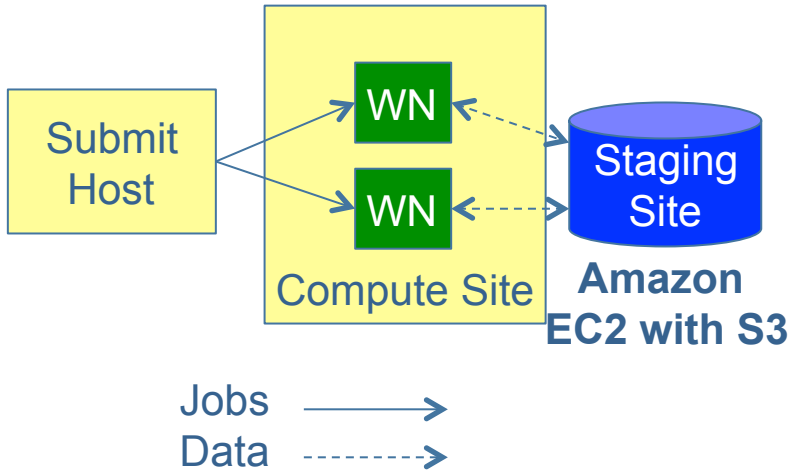
Shared Filesystem setup (typical of XSEDE and HPC sites)

- Worker nodes and the head node have a shared filesystem, usually a parallel filesystem with great I/O characteristics
- Can leverage symlinking against existing datasets
- Staging site is the shared-fs.



Non-shared filesystem setup with staging site (typical of OSG and EC 2)

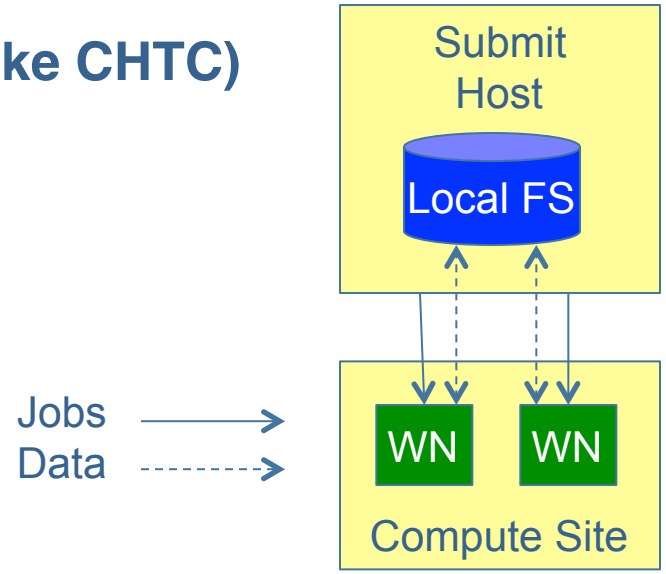
- Worker nodes don't share a filesystem.
- Data is pulled from / pushed to the existing storage element.
- A separate staging site such as S3.



Supported Data Staging Approaches - II

Condor IO (Typical of large Condor Pools like CHTC)

- Worker nodes don't share a filesystem
- Symlink against datasets available locally
- Data is pulled from / pushed to the submit host via Condor file transfers
- Staging site is the submit host.



Supported Transfer Protocols

- HTTP
- SCP
- GridFTP
- IRODS
- S3
- Condor File IO
- File Copy

Using Pegasus allows you to move from one deployment to another without changing the workflow description!

Simple Steps to Run Pegasus

1. **Specify your computation in terms of DAX**
   - Write a simple DAX generator
   - Python, Java , Perl based API provided with Pegasus
2. **Set up your catalogs**
   - Replica catalog, transformation catalog and site catalog.
3. **Plan and Submit your workflow**
   - Use *pegasus-plan* to generate your executable workflow that is mapped onto the target resources and submits it for execution
4. **Monitor and Analyze your workflow**
   - Use *pegasus-status* | *pegasus-analyzer* to monitor the execution of your workflow
5. **Workflow Statistics**
   - Run pegasus-statistics to generate statistics about your workflow run.

Different Directories used by Pegasus

1. **Submit Directory**
   - The directory where pegasus-plan generates the executable workflow i.e HTCondor DAGMan and job submit files.
   - Specified by ***--dir*** option to pegasus-plan
2. **Input Directory**
   - Mostly input file locations are catalogued in the Replica Catalog.
   - However, if inputs are on the submit host, then you can pass ***–input-dir*** option to pegasus-plan
3. **Scratch Directory**
   - Workflow specific directory created on the staging site by the **create-dir** job. This is where all the workflow inputs and outputs are gathered.
   - The base directory specified in the site catalog entry in ***sites.xml*** file.
4. **Output Directory**
   - The output directory where the outputs of the workflow appear.
   - Specified in the output site entry in the ***sites.xml*** file.
   - Can also be optionally specified by ***–output-dir*** option to pegasus-plan

Workflow Monitoring - Stampede

- **Leverage Stampede Monitoring framework with DB backend**
  - Populates data at runtime. A background daemon monitors the logs files and populates information about the workflow to a database
  - Stores workflow structure, and runtime stats for each task.

- **Tools for querying the monitoring framework**
  - **pegasus-status**
    - Status of the workflow
  - **pegasus-statistics**
    - Detailed statistics about your finished workflow

```
------------------------------------------------------------------------------
Type           Succeeded Failed  Incomplete  Total     Retries   Total+Retries
Tasks          135002    0       0           135002    0         135002
Jobs           4529      0       0           4529      0         4529
Sub-Workflows  2         0       0           2         0         2
------------------------------------------------------------------------------

Workflow wall time                                : 13 hrs, 2 mins, (46973 secs)
Workflow cumulative job wall time                 : 384 days, 5 hrs, (33195705 secs)
Cumulative job walltime as seen from submit side : 384 days, 18 hrs, (33243709 secs)
```


Workflow Debugging Through Pegasus

- **After a workflow has completed, we can run pegasus-analyzer to analyze the workflow and provide a summary of the run**

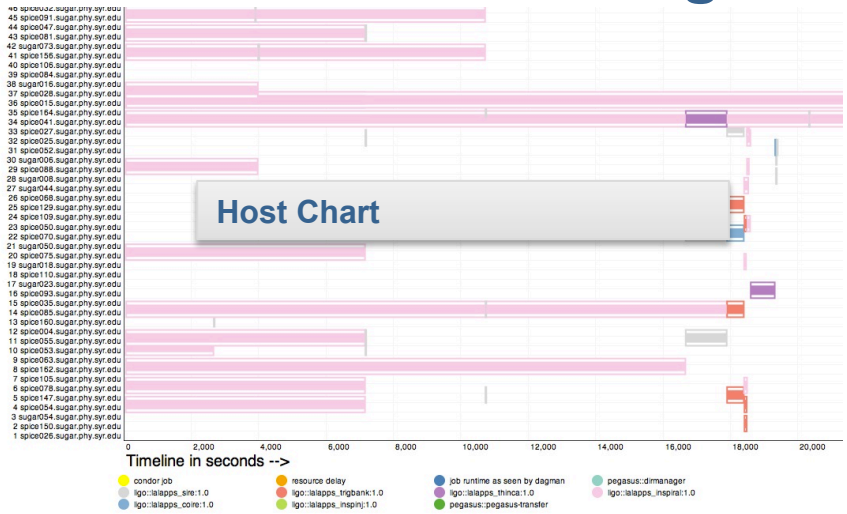
- **pegasus-analyzer's output contains**
  - **a brief summary section**
    - showing how many jobs have succeeded
    - and how many have failed.
  - **For each failed job**
    - showing its last known state
    - exitcode
    - working directory
    - the location of its submit, output, and error files.
    - any stdout and stderr from the job.

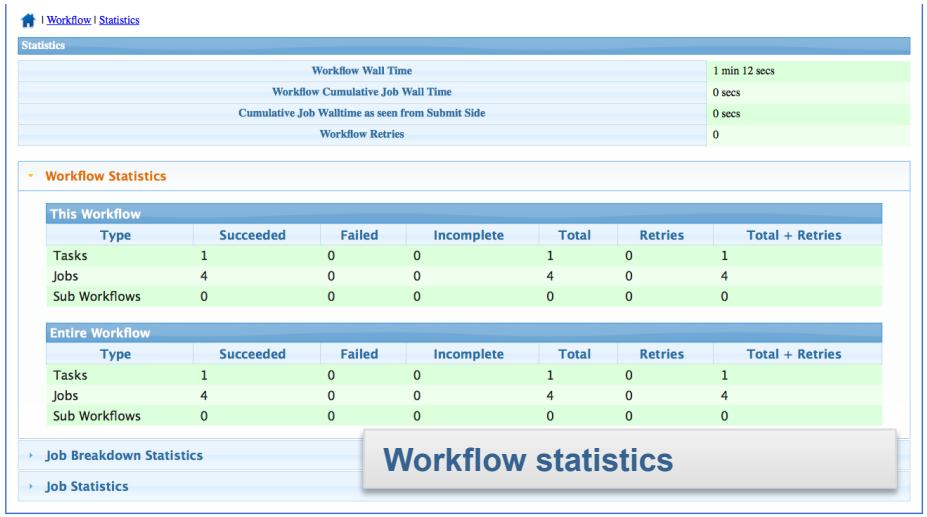
Alleviates the need for searching through large DAGMan and Condor logs!

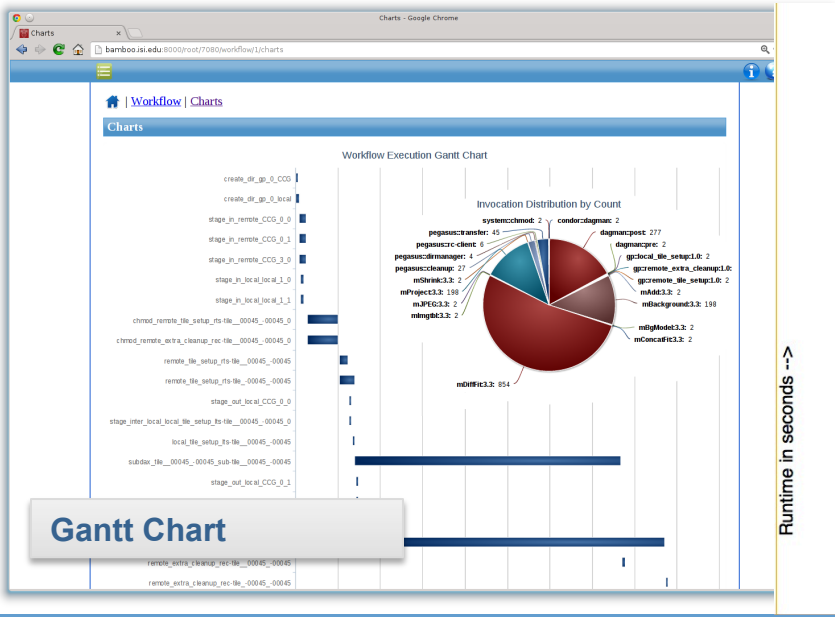
Workflow Monitoring Dashboard: pegasus-dashboard

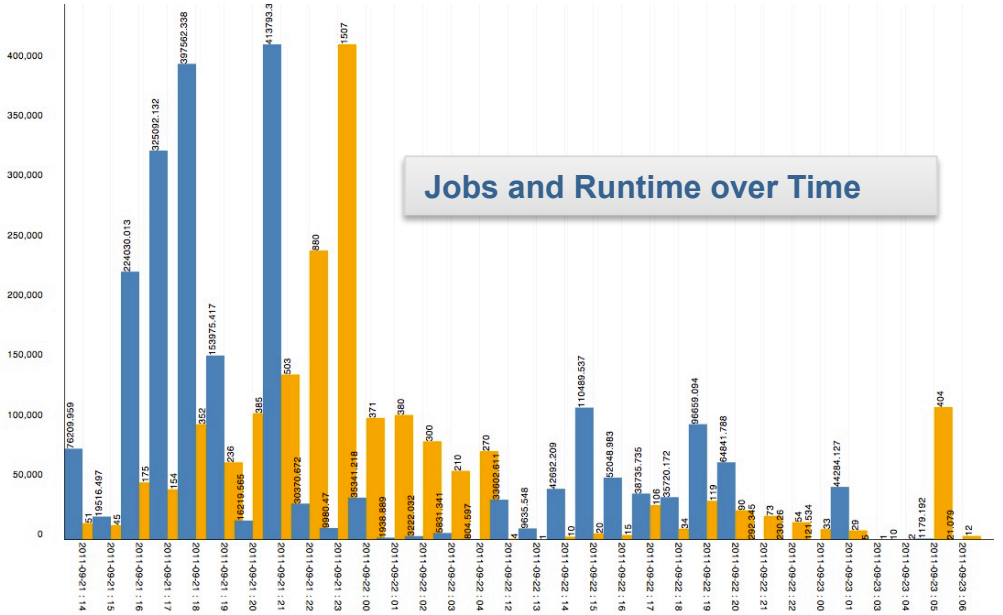
- **A python based online workflow dashboard**
  - Uses the FLASK framework
  - Beta version released in 4.2
  - Queries the STAMPEDE database
- **Lists all the user workflows on the home page and are color coded.**
  - Green indicates a successful workflow,
  - Red indicates a failed workflow
  - Blue indicates a running workflow
- **Explore Workflow and Troubleshoot (Workflow Page)**
  - Has identifying metadata about the workflow
  - Tabbed interface to
    - List of sub workflows
    - Failed jobs
    - Running jobs
    - Successful jobs.

Workflow Monitoring Dashboard

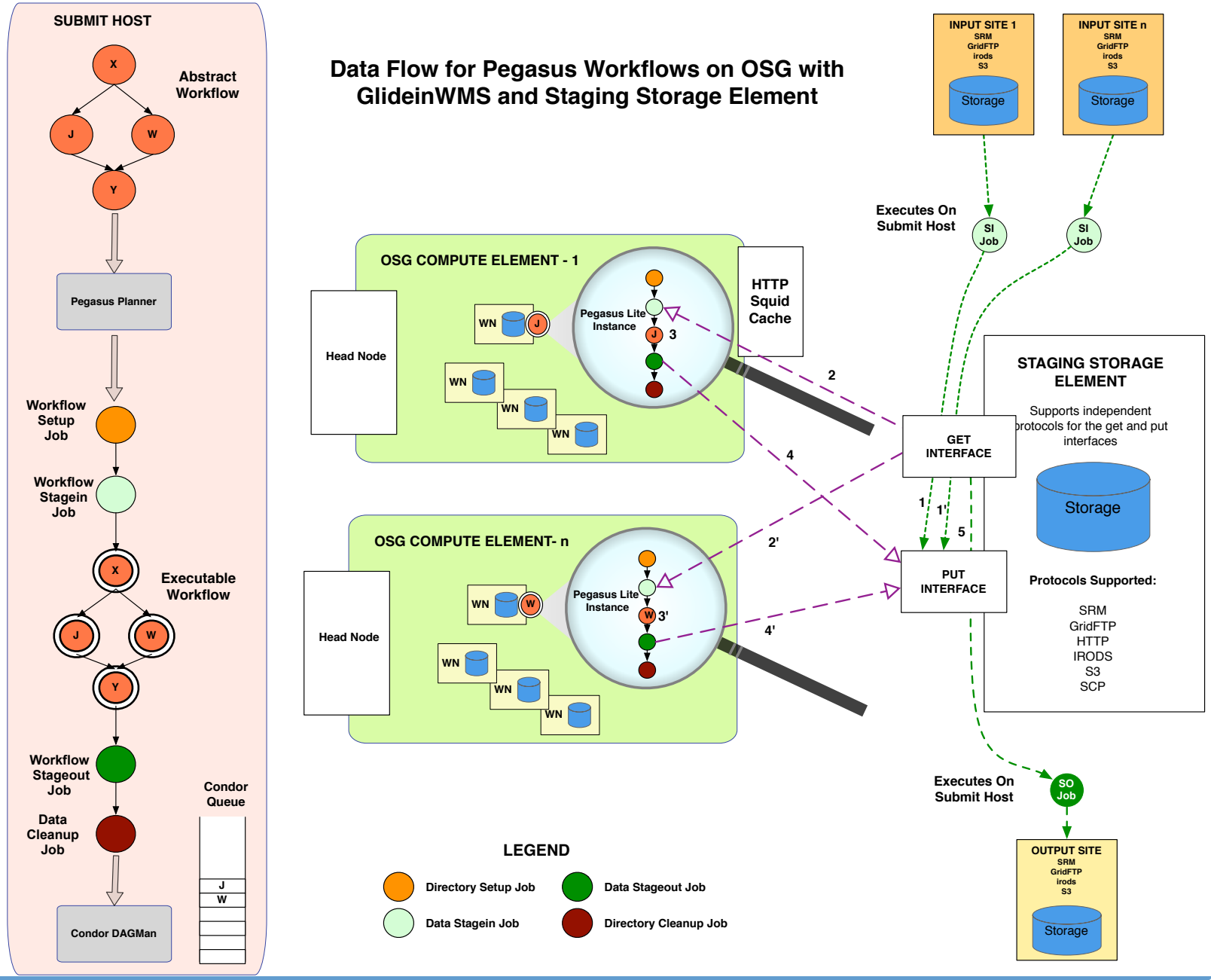




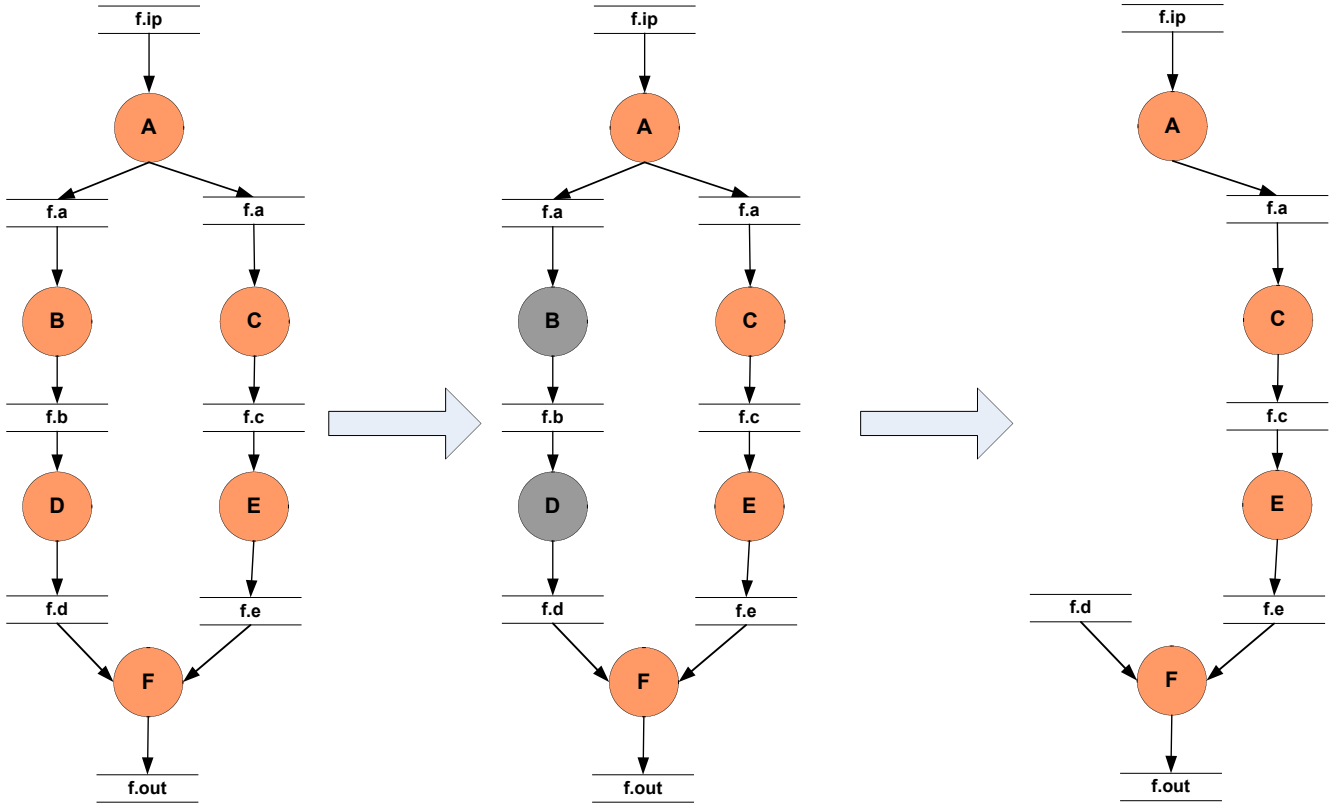




Data Flow for Pegasus Workflows on OSG with GlideinWMS and Staging Storage Element



Workflow Reduction (Data Reuse)



Abstract Workflow

File f.d exists somewhere.
Reuse it.
Mark Jobs D and B to delete

Delete Job D and Job B

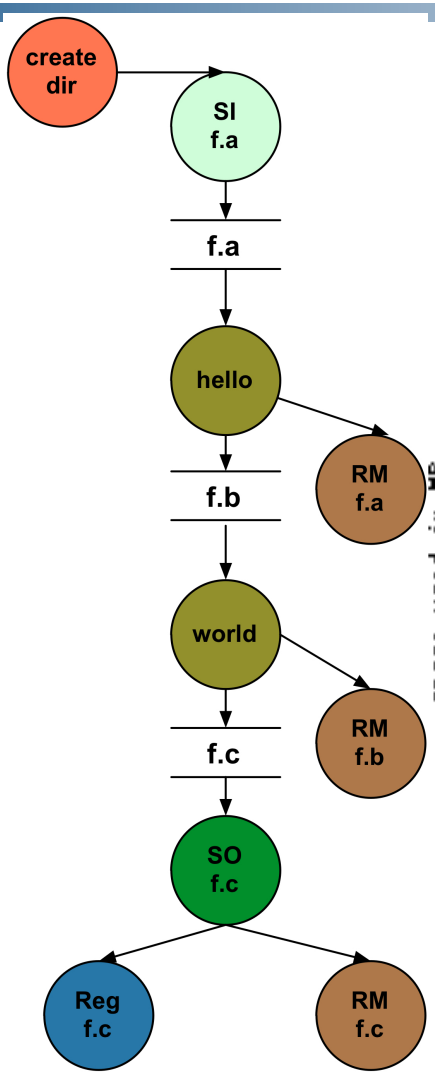
Useful when you have done a part of computation and then realize the need to change the structure. Re-plan instead of submitting rescue DAG!

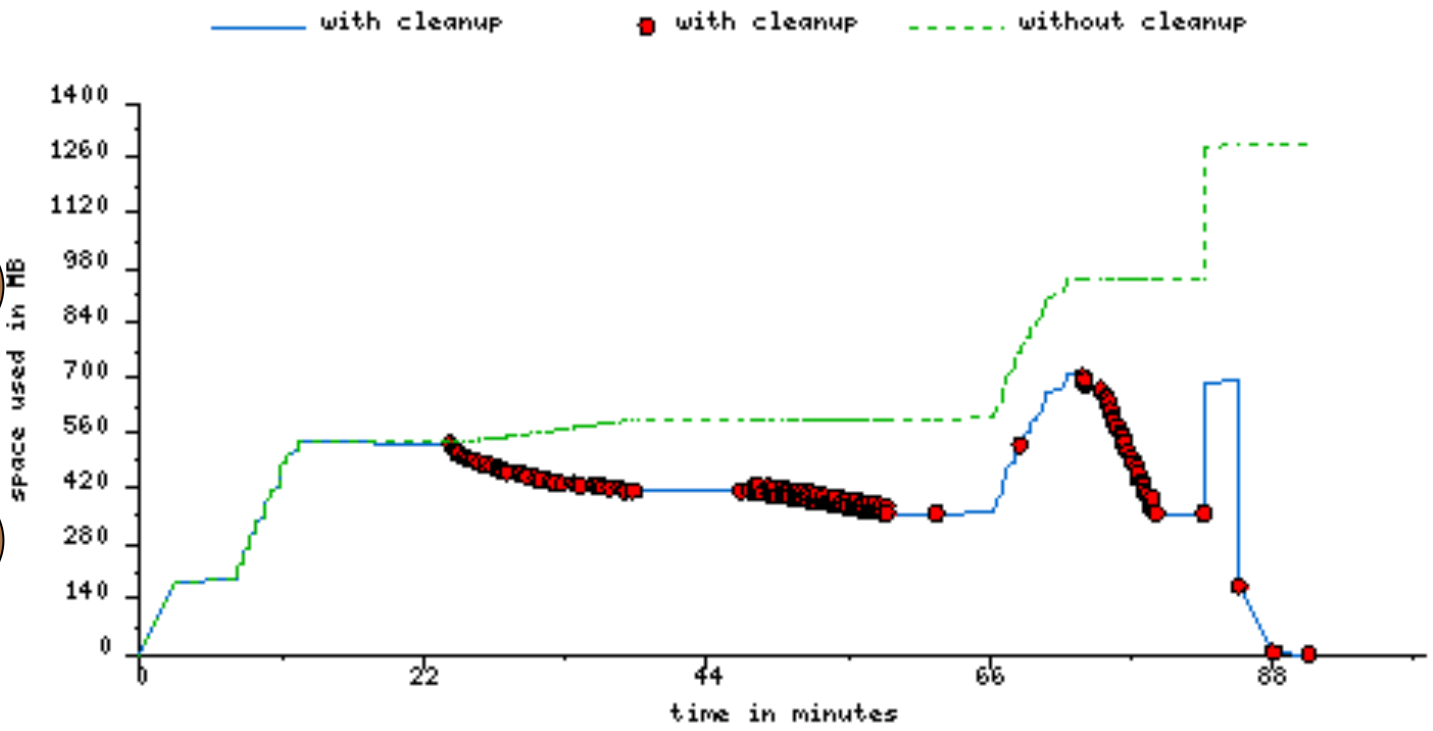
Data cleanup

- **Problem: Running out of disk space during workflow execution**
- **Why does it occur**
  - Workflows could bring in huge amounts of data
  - Data is generated during workflow execution
  - Users don't worry about cleaning up after they are done
- **Solution**
  - **Do cleanup after workflows finish**
    - Does not work as the scratch may get filled much before during execution
  - **Interleave cleanup automatically during workflow execution.**
    - Requires an analysis of the workflow to determine, when a file is no longer required
  - **Cluster the cleanup jobs by level for large workflows**

Real Life Example: Used by a UCLA genomics researcher to delete TB's of data automatically for long running workflows!!

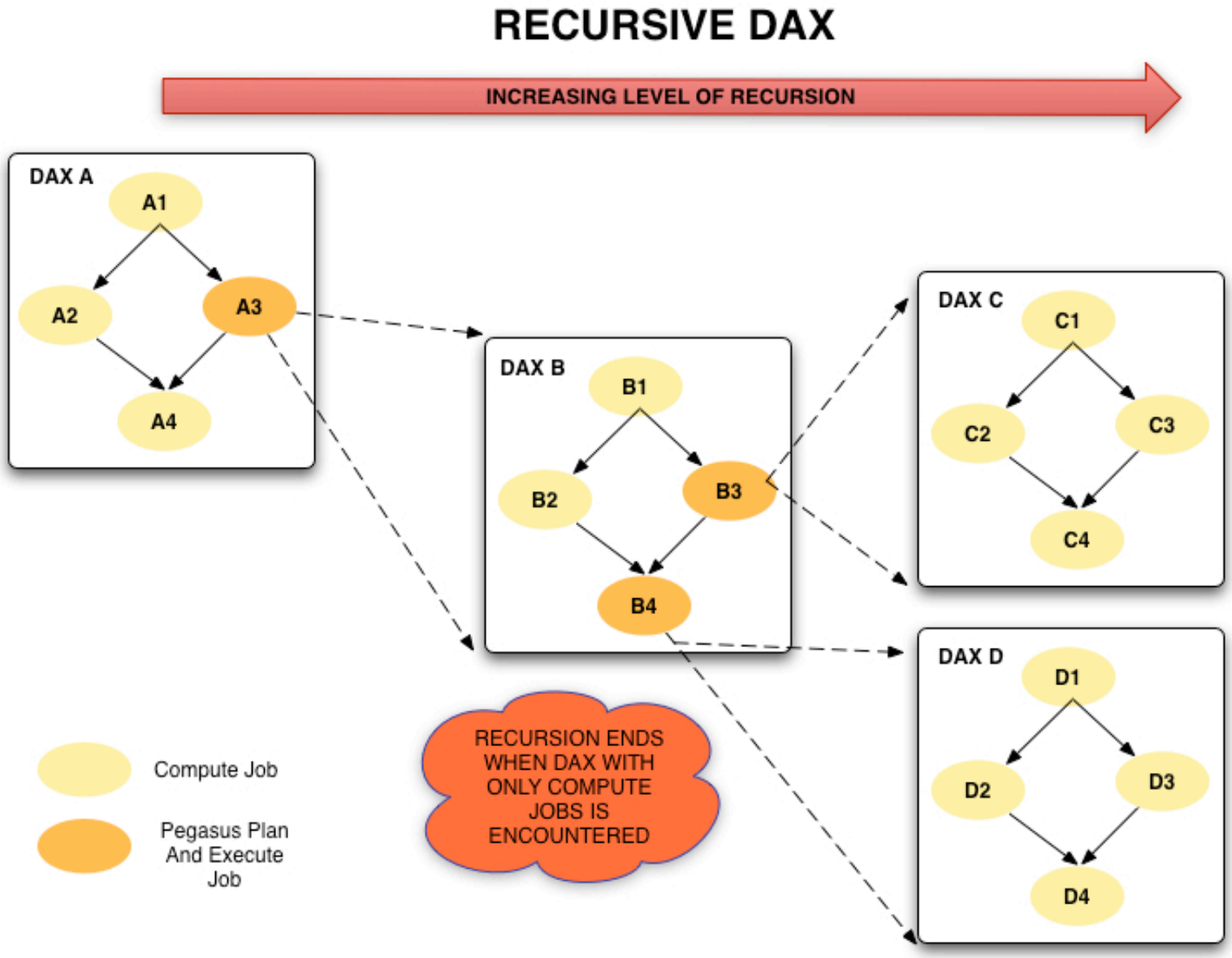
Data cleanup (cont)





Montage 1 degree workflow run with cleanup

Hierarchical Workflows



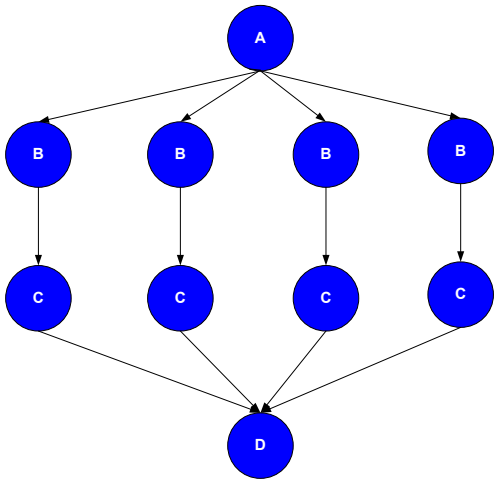
Example Hierarchical Workflow

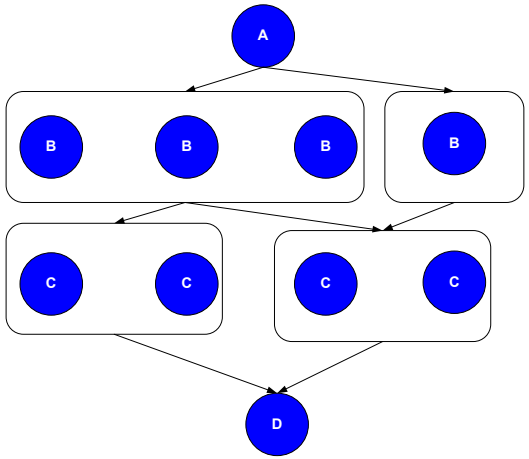
- <dax> element behaves like <job>
  - Arguments are for pegasus-plan (most are inherited)
- Planner is invoked when DAX job is ready to run

```
<?xml version="1.0" encoding="UTF-8"?>
<adag version="3.4" name="multi-level">
    <job id="ID0000001" namespace="example" name="sleep">
        <argument>5</argument>
    </job>
    <dax id="ID0000002" file="sub.dax">
        <argument>--output-site local</argument>
    </dax>
    <job id="ID0000003" namespace="example" name="sleep">
        <argument>5</argument>
    </job>
    <child ref="ID0000002">
        <parent ref="ID0000001"/>
    </child>
    <child ref="ID0000003">
        <parent ref="ID0000002"/>
    </child>
</adag>
```

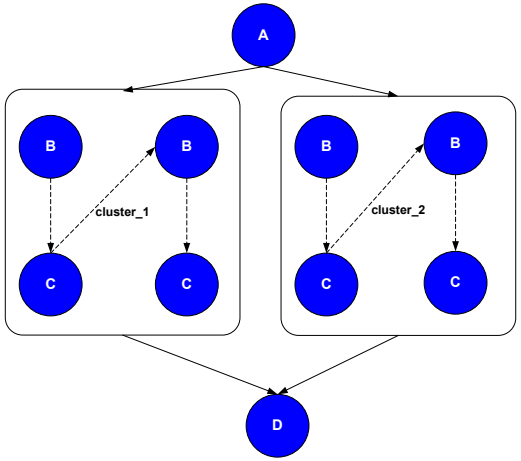

Workflow Restructuring to improve application performance

- **Cluster small running jobs together to achieve better performance**
- **Why?**
  - Each job has scheduling overhead – need to make this overhead worthwhile
  - Ideally users should run a job on the grid that takes at least 10/30/60/? minutes to execute
  - Clustered tasks can reuse common input data – less data transfers





Horizontal clustering



Label-based clustering

Pegasus-MPI-Cluster

- A master/worker task scheduler for running fine-grained workflows on batch systems
- Runs as an MPI job
  - Uses MPI to implement master/worker protocol
- Works on most HPC systems
  - Requires: MPI, a shared file system, and fork()
- Allows sub-graphs of a Pegasus workflow to be submitted as monolithic jobs to remote resources

PMC Features

- **Fault Tolerance**
  - Retries at the task level (master resends task to another worker)
  - Retries at the workflow level (using a transaction log to record progress)
- **Resource-aware scheduling**
  - Many HPC machines have low memory/core
  - PMC can allocate memory and cores to a task, and force other slots on the same node to be idle
- **I/O Forwarding**
  - Small tasks == small I/O == poor performance
  - PMC reads data off of pipes from worker and forwards it using MPI messages to a central I/O process, which collects the data and writes it to disk
  - Writes are not interleaved, no locking required for synchronization

What Does Pegasus provide an Application - I

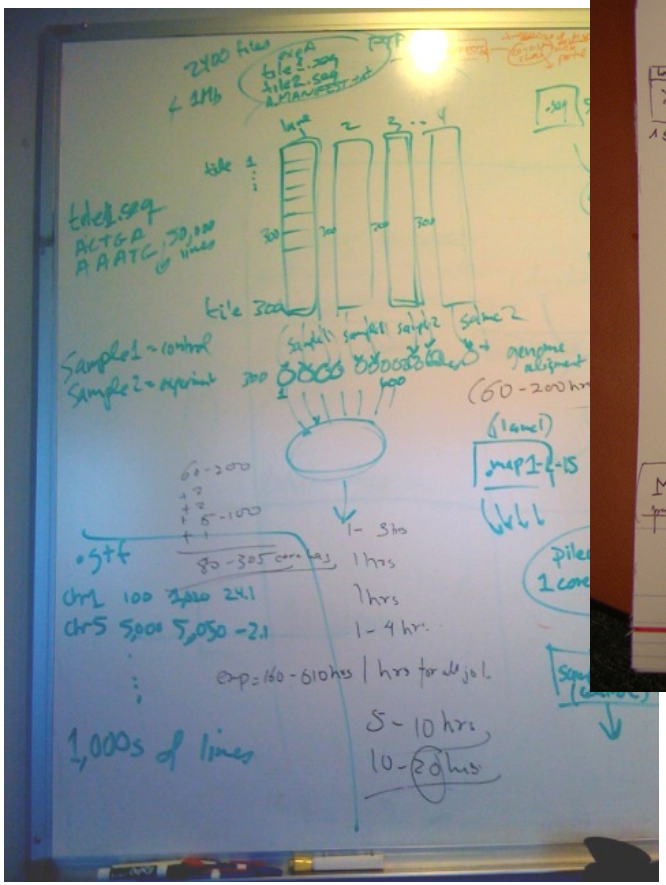
- **Portability / Reuse**
  - User created workflows can easily be mapped to and run in different environments without alteration.
- **Data Management**
  - Pegasus handles replica selection, data transfers and output registrations in data catalogs. These tasks are added to a workflow as auxiliary jobs by the Pegasus planner.
- **Performance**
  - The Pegasus mapper can reorder, group, and prioritize tasks in order to increase the overall workflow performance.

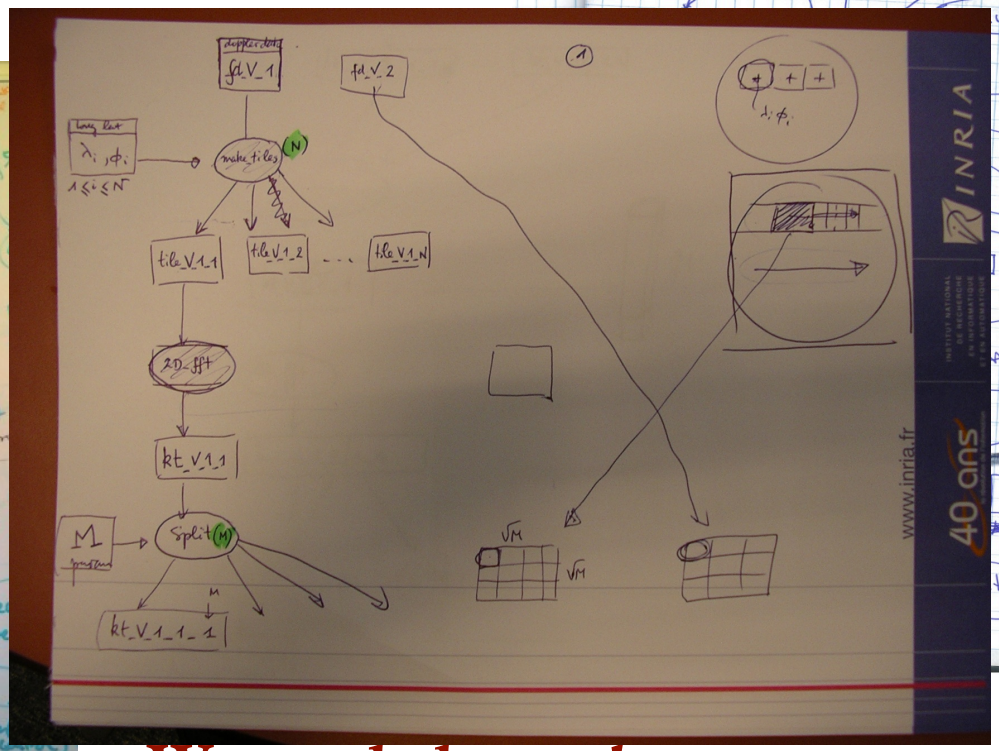
What Does Pegasus provide an Application - II

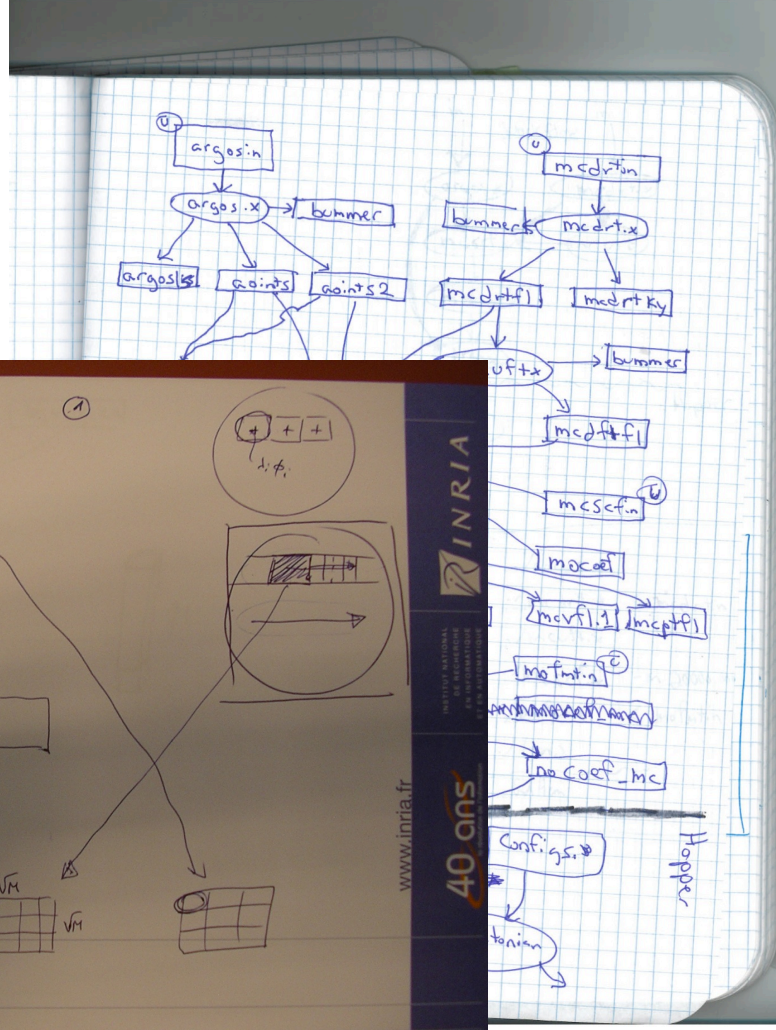
- **Provenance**
  - Provenance data is collected in a database, and the data can be summaries with tools such as pegasus-statistics, pegasus-plots, or directly with SQL queries.
- **Reliability and Debugging Tools**
  - Jobs and data transfers are automatically retried in case of failures. Debugging tools such as pegasus-analyzer helps the user to debug the workflow in case of non-recoverable failures.
- **Scalability**
  - Hierarchal workflows
  - Scale to hundreds of thousands of nodes in a workflow.

If you get stuck…

And you can draw….







We can help you!

More Information

- **Pegasus Website:**
  - http://pegasus.isi.edu
- **Tutorial:**
  - http://pegasus.isi.edu/wms/docs/latest/tutorial.php
- **Documentation:**
  - http://pegasus.isi.edu/documentation
- **Email addresses:**
  - Pegasus users list (public): pegasus-users@isi.edu
  - Pegasus support (private): pegasus-support@isi.edu