

Before we start

Hands on Exercises Notes

<https://pegasus.isi.edu/tutorial/isi/>

System

workflow.isi.edu

On terminal login in via ssh

Training Accounts

Pick up from the instructor





U.S. DEPARTMENT OF
ENERGY



EScience 2019: Pegasus Scientific Workflows with Containers

Pegasus Workflow Management System

Karan Vahi
Mats Rynge



OUTLINE

Introduction *Scientific Workflows*
Pegasus Overview
Successful Stories

Pegasus Overview *Basic Concepts*
Features
System Architecture

Hands-on Tutorial *Submitting a Workflow*
Workflow Dashboard and Monitoring
Generating the Workflow

Understanding Pegasus Features *Information Catalogs*
Containers

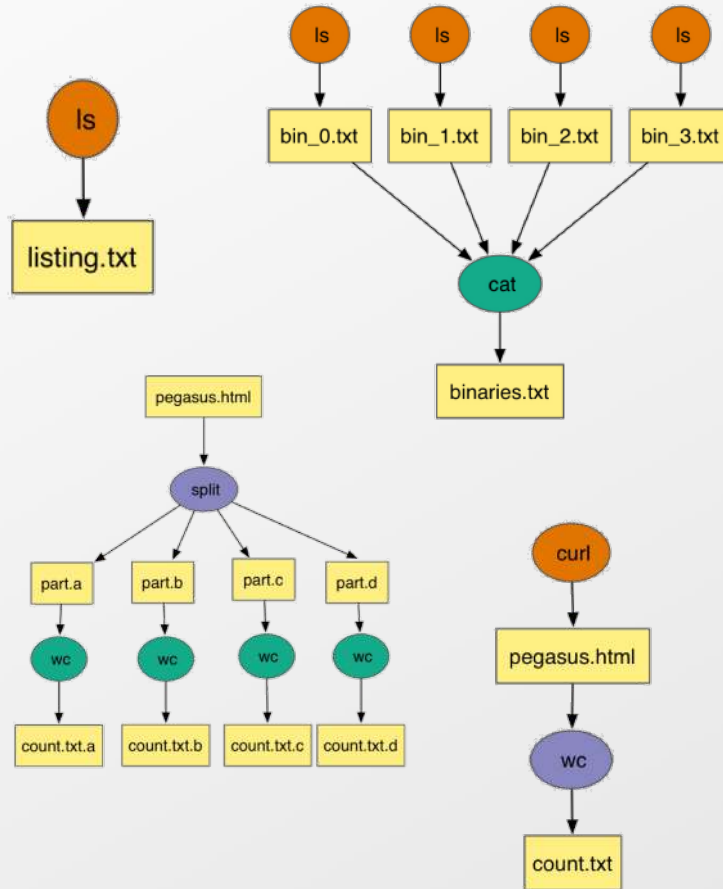
Hands-on Tutorial *Catalogs*
Workflows with Containers
Clustering
Fault-Tolerance

Other Features *Integrity Checking, Data Staging*
Jupyter Notebooks,
Metadata, Hierarchical Workflows, Data Reuse

<http://pegasus.isi.edu>



Compute Pipelines Building Blocks



Compute Pipelines

Allows scientists to connect different codes together and execute their analysis

Pipelines can be very simple (independent or parallel) jobs or complex represented as DAG's

Helps users to automate scale up

However, it is still up-to user to figure out

Data Management

How do you ship in the small/large amounts data required by your pipeline and protocols to use?

How best to leverage different infrastructure setups

OSG has no shared filesystem while XSEDE and your local campus cluster has one!

Debug and Monitor Computations

Correlate data across lots of log files

Need to know what host a job ran on and how it was invoked

Restructure Workflows for Improved Performance

Short running tasks? Data placement

Why Pegasus ?

Automates complex, multi-stage processing pipelines

Enables parallel, **distributed** computations

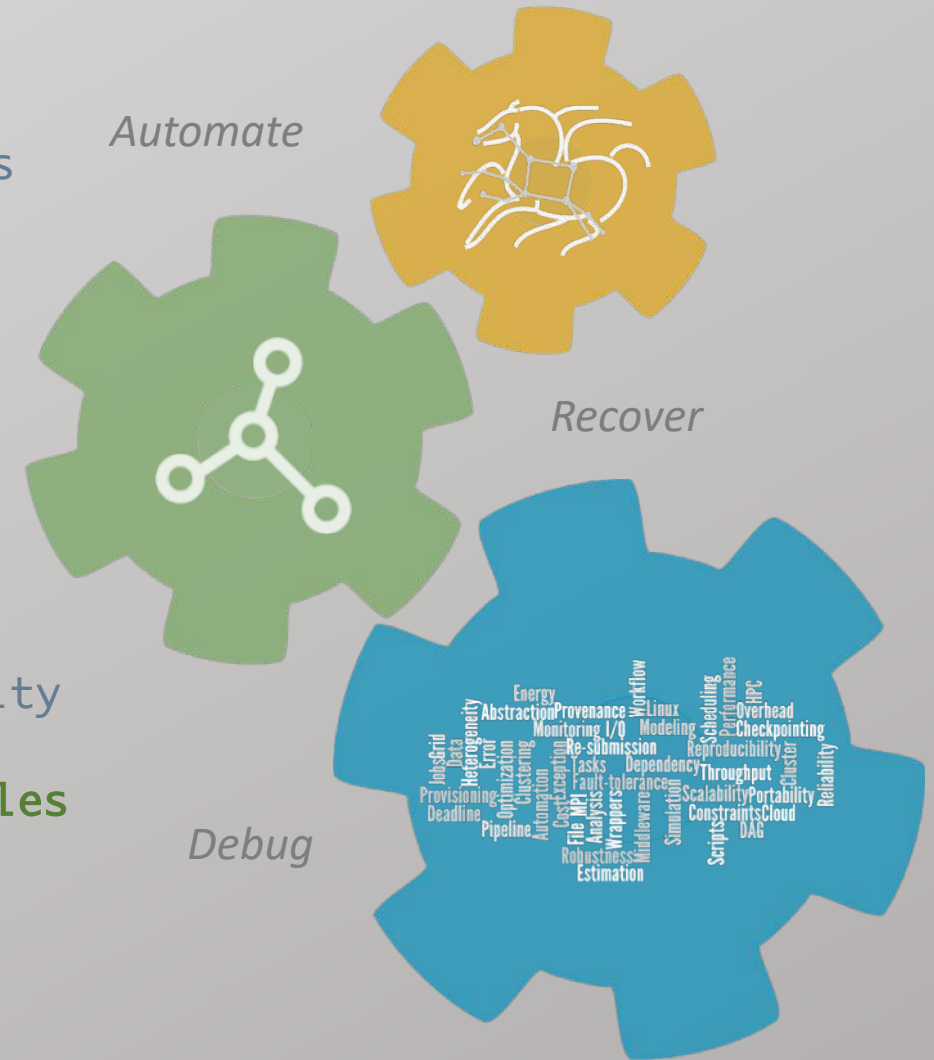
Automatically executes data transfers

Reusable, aids **reproducibility**

Records how data was produced (**provenance**)

Handles **failures** with to provide reliability

Keeps track of data and **files**

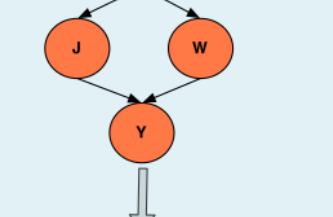


NSF funded project since 2001,
with close collaboration with
HTCondor team

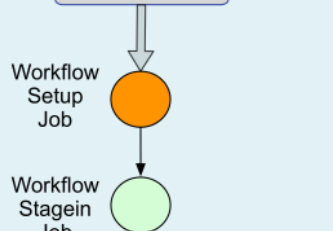
Some of the successful stories...

Data Flow for LIGO Pegasus Workflows in OSG

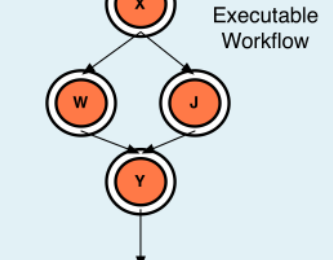
SUBMIT HOST Abstract Workflow



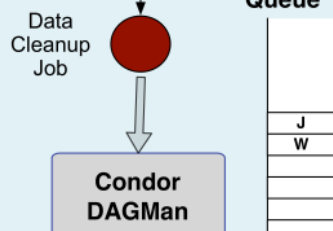
Pegasus Planner



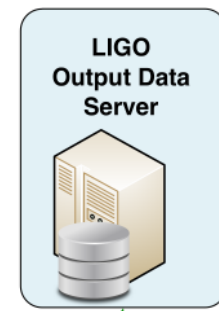
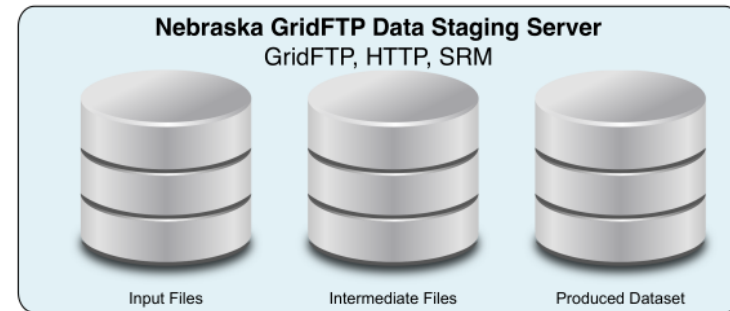
Executable Workflow



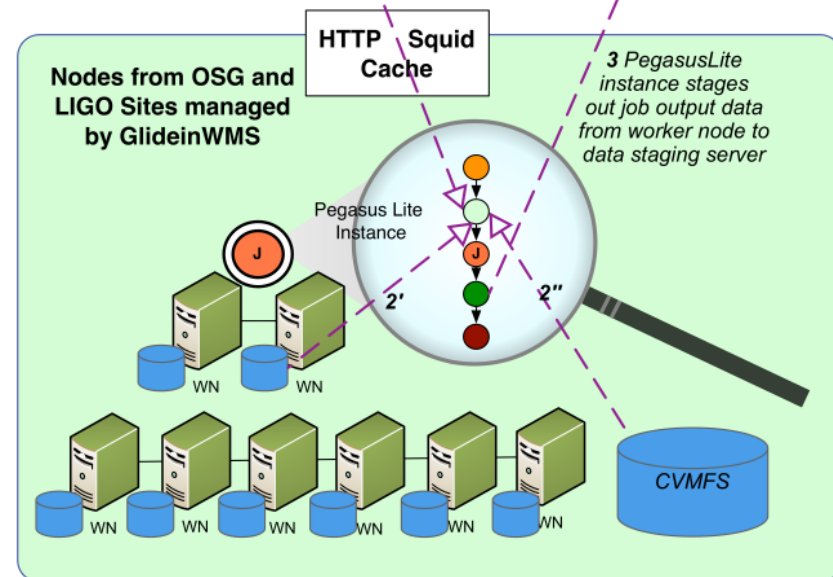
Condor Schedd Queue



Condor DAGMan



- 1 Workflow Stagein Job stages in the input data for workflow from user server
- 2 PegasusLite instance looks up input data on the compute node/ CVMFS. If not present, stage-in data from remote data staging server
- 3 PegasusLite instance stages out job output data from worker node to data staging server
- 4 Workflow Stageout Job stages produced data from data staging server to LIGO Output Data Server



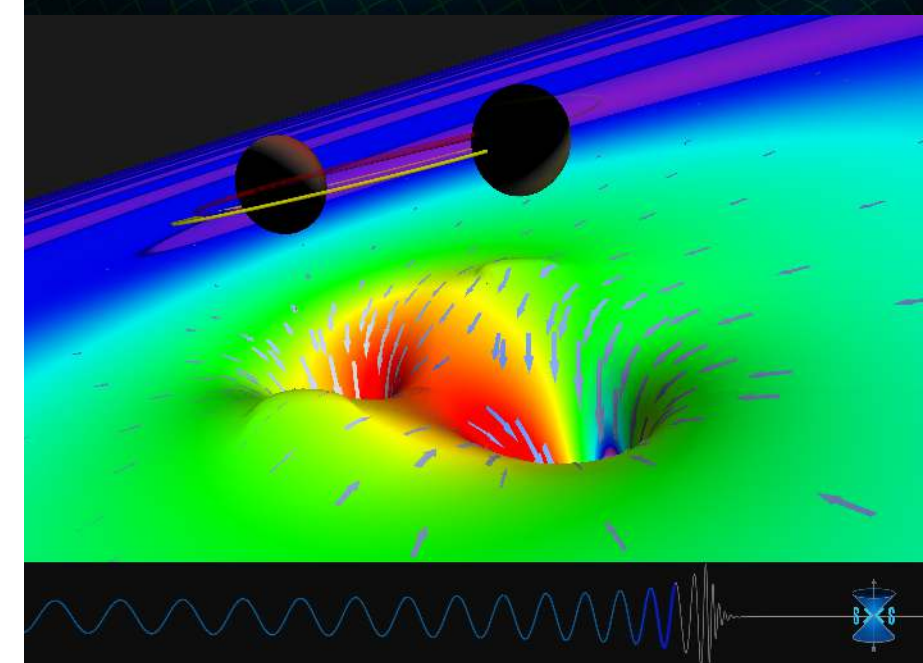
LEGEND

- Orange circle: Directory Setup Job
- Green circle: Data Stageout Job
- Circle with J: Pegasus Lite Compute Job
- Light green circle: Data Stagein Job
- Red circle: Directory Cleanup Job
- Server rack icon: Worker Node

Advanced LIGO – Laser Interferometer Gravitational Wave Observatory

60,000 compute tasks
Input Data: 5000 files (10GB total)
Output Data: 60,000 files (60GB total)

executed on LIGO Data Grid, Open Science Grid and XSEDE



Advanced LIGO PyCBC Workflow

One of the main pipelines to measure the statistical significance of data needed for discovery

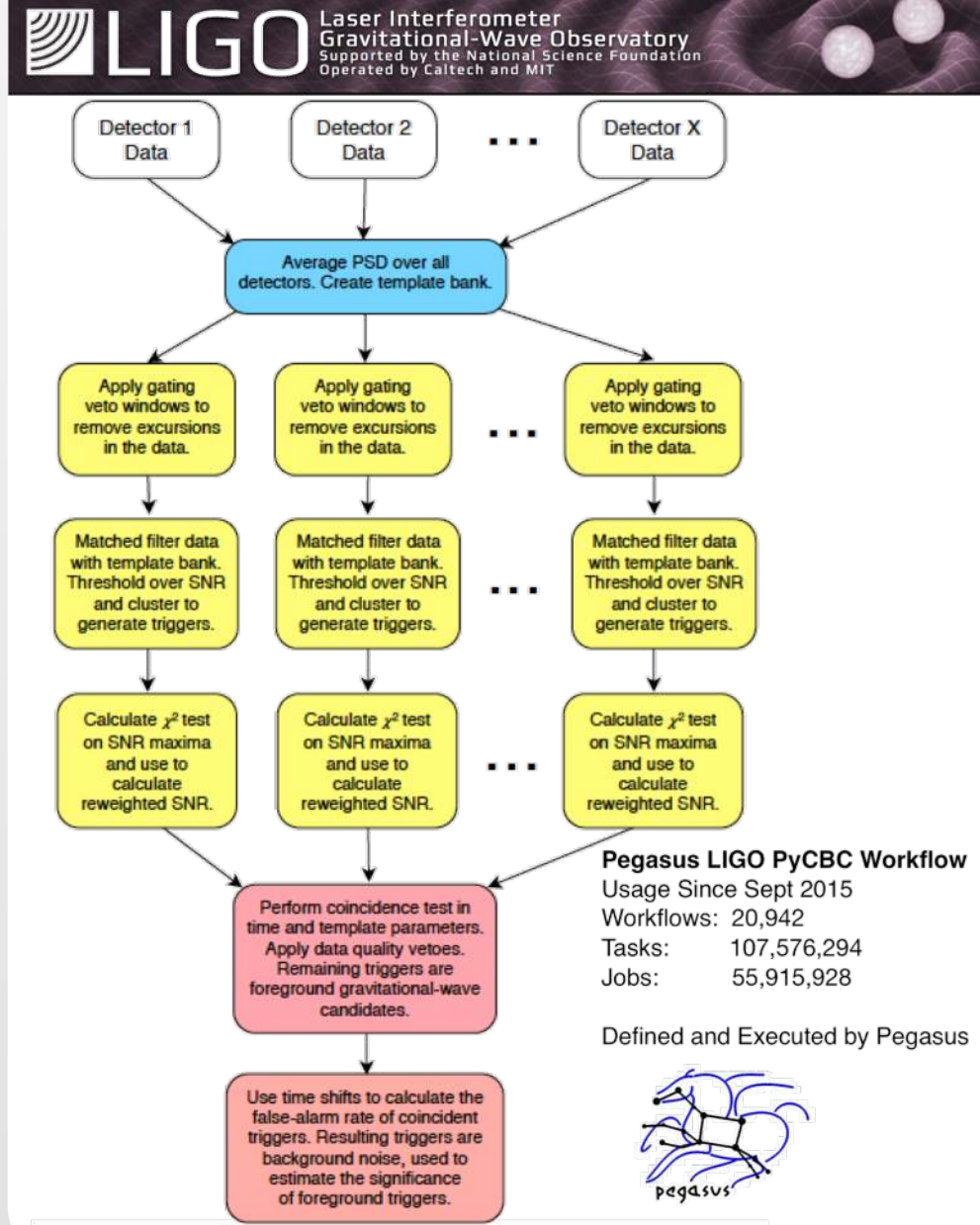
Contains **100's of thousands of jobs** and accesses on order of **terabytes of data**

Uses data from multiple detectors

For the detection, the pipeline was executed on Syracuse and Albert Einstein Institute Hannover

A single run of the binary black hole + binary neutron star search through the O1 data (about 3 calendar months of data with 50% duty cycle) requires a **workflow** with **194,364 jobs**

Generating the final O1 results with all the review required for the first discovery took about **20 million core hours**



Southern California Earthquake Center's CyberShake

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)

CPU jobs (Mesh generation, seismogram synthesis):

1,094,000 node-hours

GPU jobs: 439,000 node-hours

AWP-ODC finite-difference code

5 billion points per volume, 23000 timesteps

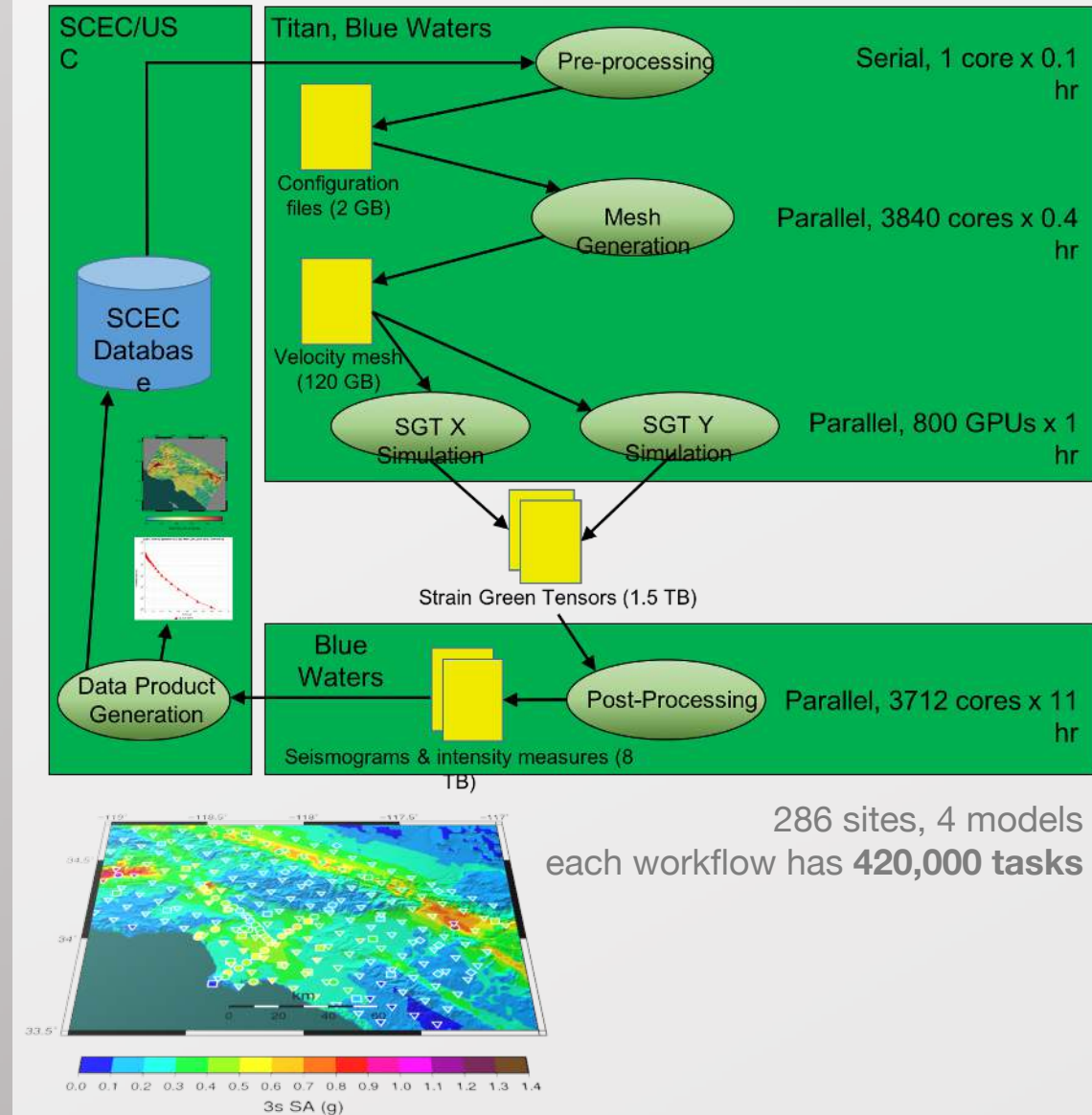
200 GPUs for 1 hour

Titan:

421,000 CPU node-hours, 110,000 GPU node-hours

Blue Waters:

673,000 CPU node-hours, 329,000 GPU node-hours



XENONnT - Dark Matter Search

Two workflows: Monte Carlo simulations, and the main processing pipeline.



Workflows execute across Open Science Grid (OSG) and European Grid Infrastructure (EGI)

Rucio for data management

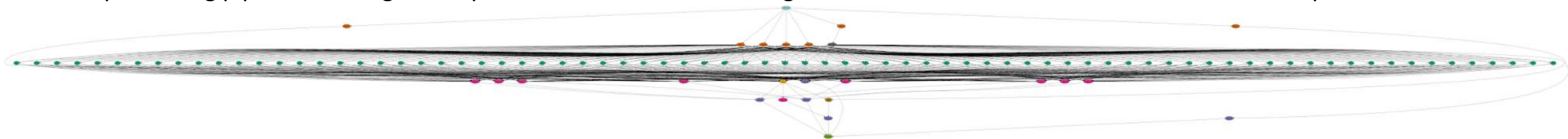
MongoDB instance to track science runs and data products.



Type	Succeeded	Failed	Incomplete	Total	Retries	Total+Retries
Tasks	4000	0	0	4000	267	4267
Jobs	4484	0	0	4484	267	4751
Sub-Workflows	0	0	0	0	0	0

Workflow wall time	: 5 hrs, 2 mins
Cumulative job wall time	: 136 days, 9 hrs
Cumulative job wall time as seen from submit side	: 141 days, 16 hrs
Cumulative job badput wall time	: 1 day, 2 hrs
Cumulative job badput wall time as seen from submit side	: 4 days, 20 hrs

Main processing pipeline is being developed for XENONnT - data taking will start at the end of 2019. Workflow in development:



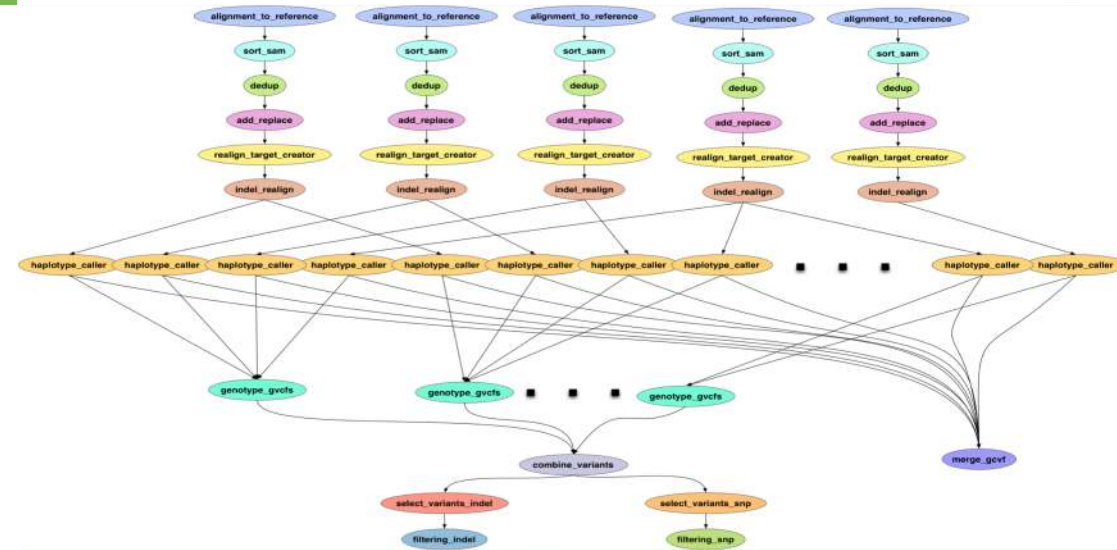
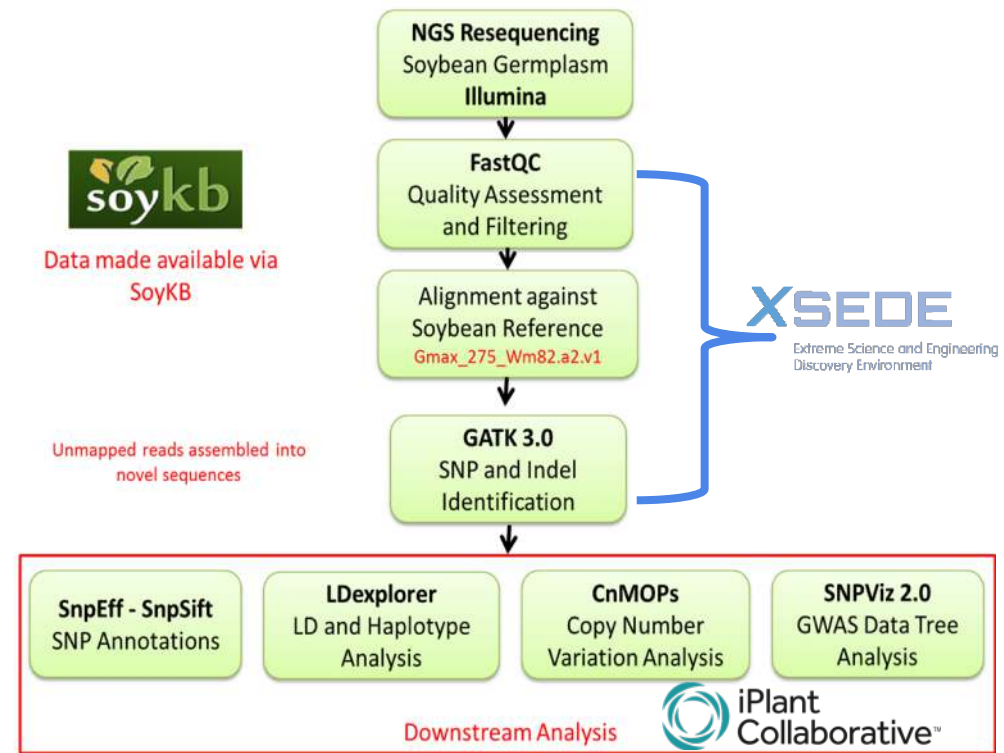
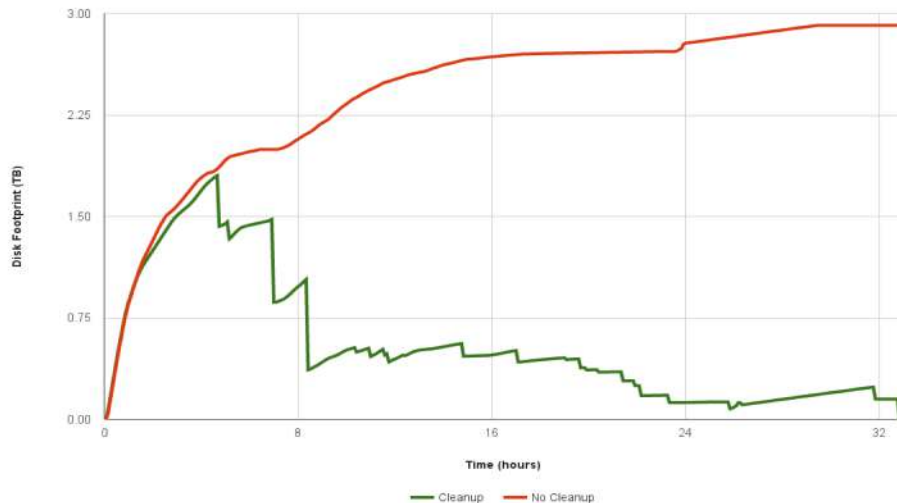
Soybean Workflow

TACC Wrangler as Execution Environment

Flash Based Shared Storage

Switched to glideins (pilot jobs) - Brings in remote compute nodes and joins them to the HTCondor pool on the submit host - Workflow runs at a finer granularity

Works well on Wrangler due to more cores and memory per node (48 cores, 128 GB RAM)



OUTLINE

Introduction *Scientific Workflows*
Pegasus Overview
Successful Stories

Pegasus Overview *Basic Concepts*
Features
System Architecture

Hands-on Tutorial *Submitting a Workflow*
Workflow Dashboard and Monitoring
Generating the Workflow

Understanding Pegasus Features *Information Catalogs*
Containers

Hands-on Tutorial *Catalogs*
Workflows with Containers
Clustering
Fault-Tolerance

Other Features *Integrity Checking, Data Staging*
Jupyter Notebooks
Metadata, Hierarchical Workflows, Data Reuse

<http://pegasus.isi.edu>

Basic concepts...

Key Pegasus Concepts

Pegasus WMS == Pegasus planner (mapper) + DAGMan workflow engine + HTCondor scheduler/broker

Pegasus maps workflows to infrastructure

DAGMan manages dependencies and reliability

HTCondor is used as a broker to interface with different schedulers

Workflows are DAGs

Nodes: jobs, edges: dependencies

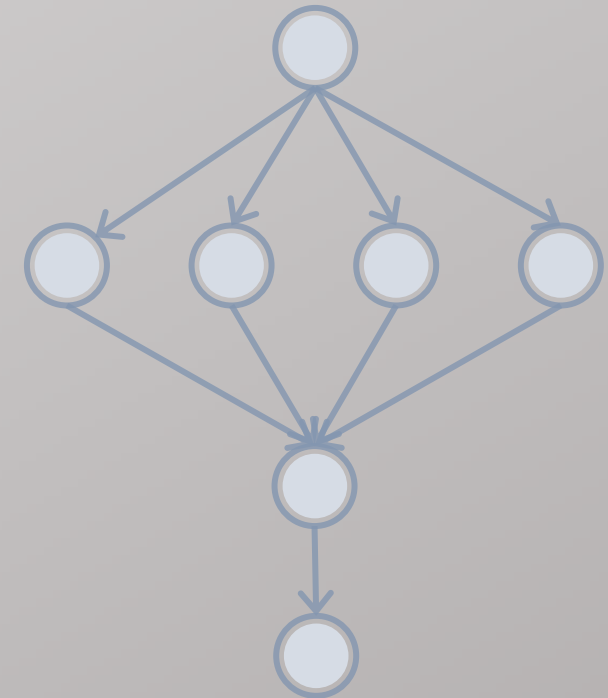
No while loops, no conditional branches

Jobs are standalone executables

Planning occurs ahead of execution

Planning converts an abstract workflow into a concrete, executable workflow

Planner is like a compiler

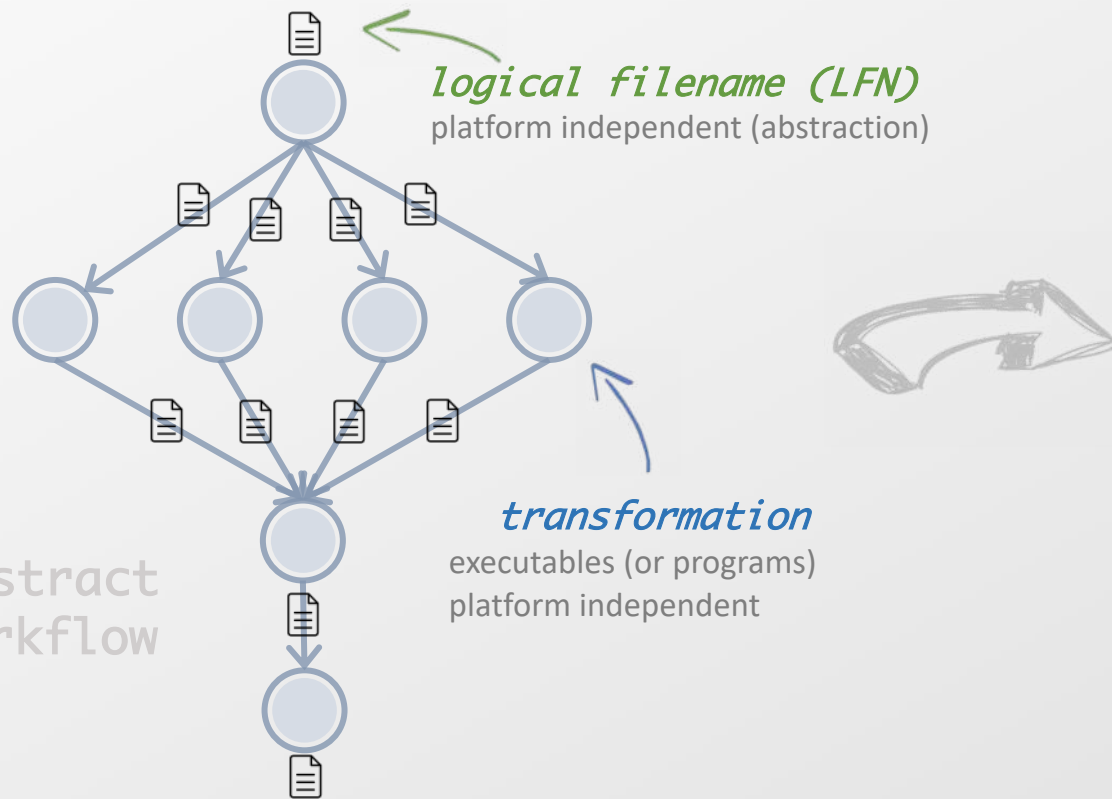


DAX

DAG in XML

Portable Description

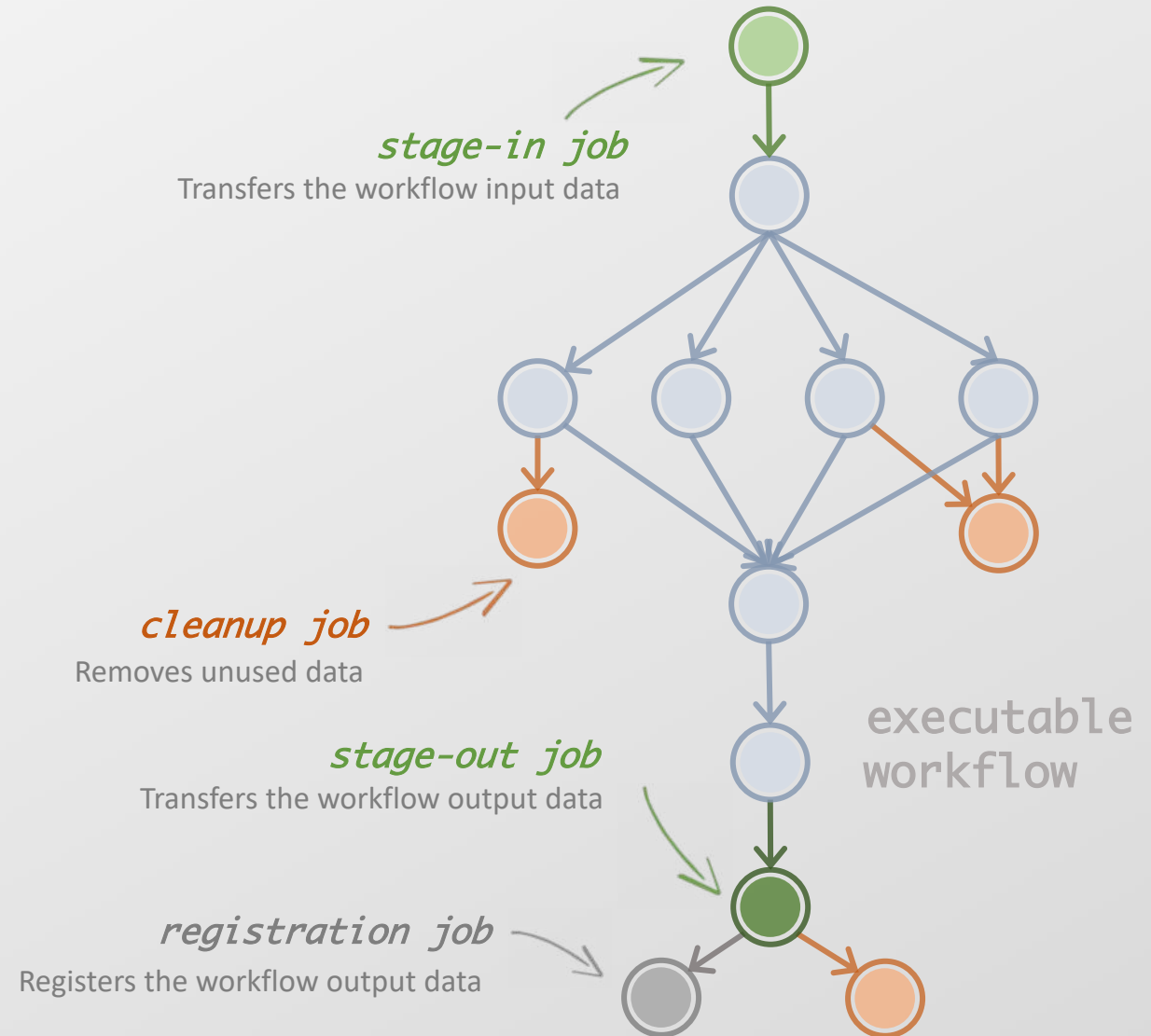
Users do not worry about
low level execution details



abstract
workflow

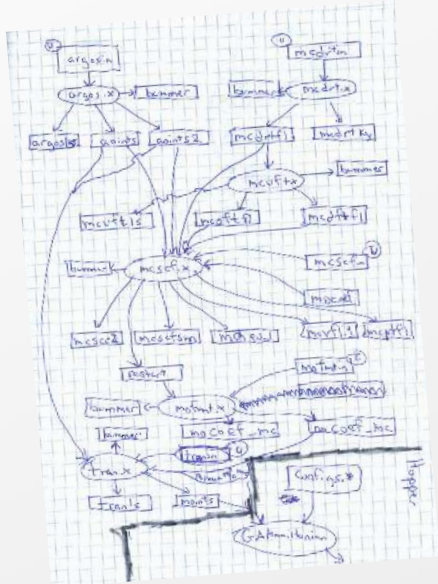
DAG

directed-acyclic graphs



executable
workflow

Pegasus also provides tools to generate the abstract workflow



```
#!/usr/bin/env python

from Pegasus.DAX3 import *
import sys
import os

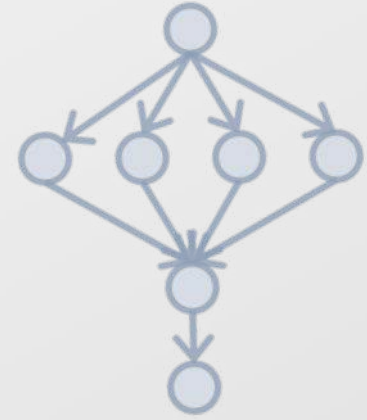
# Create an abstract dag
dax = ADAG("hello_world")

# Add the hello job
hello = Job(namespace="hello_world",
             name="hello", version="1.0")
b = File("f.b")
hello.uses(a, link=Link.INPUT)
hello.uses(b, link=Link.OUTPUT)
dax.addJob(hello)

# Add the world job (depends on the hello job)
world = Job(namespace="hello_world",
             name="world", version="1.0")
c = File("f.c")
world.uses(b, link=Link.INPUT)
world.uses(c, link=Link.OUTPUT)
dax.addJob(world)

# Add control-flow dependencies
dax.addDependency(Dependency(parent=hello,
                              child=world))

# Write the DAX to stdout
dax.writeXML(sys.stdout)
```



```
<?xml version="1.0" encoding="UTF-8"?>
<!-- generator: python -->
<adag xmlns="http://pegasus.isi.edu/schema/DAX"
      version="3.4" name="hello_world">

  <!-- describe the jobs making
  up the hello world pipeline -->
  <job id="ID0000001" namespace="hello_world"
       name="hello" version="1.0">

    <uses name="f.b" link="output"/>
    <uses name="f.a" link="input"/>
  </job>

  <job id="ID0000002" namespace="hello_world"
       name="world" version="1.0">

    <uses name="f.b" link="input"/>
    <uses name="f.c" link="output"/>
  </job>

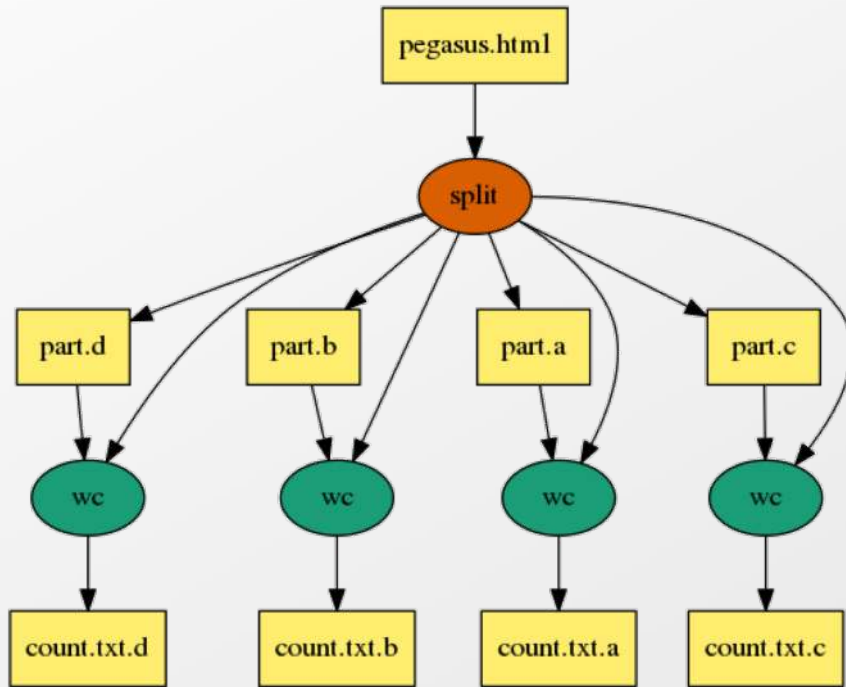
  <!-- describe the edges in the DAG -->
  <child ref="ID0000002">
    <parent ref="ID0000001"/>
  </child>
</adag>
```



DAG in XML



An example Split Workflow



Visualization Tools:
pegasus-graphviz
pegasus-plots

https://pegasus.isi.edu/documentation/tutorial_submitting_wf.php

```
#!/usr/bin/env python
```

```
import os, pwd, sys, time
from Pegasus.DAX3 import *
```

```
# Create an abstract dag
dax = ADAG("split")
```

```
webpage = File("pegasus.html")
```

```
# the split job that splits the webpage into smaller chunks
split = Job("split")
split.addArguments("-l", "100", "-a", "1", webpage, "part.")
split.uses(webpage, link=Link.INPUT)
# associate the label with the job. all jobs with same label
# are run with PMC when doing job clustering
split.addProfile( Profile("pegasus", "label", "p1"))
dax.addJob(split)
```

```
# we do a parameter sweep on the first 4 chunks created
for c in "abcd":
    part = File("part.%s" % c)
    split.uses(part, link=Link.OUTPUT, transfer=False, register=False)
    count = File("count.txt.%s" % c)
    wc = Job("wc")
    wc.addProfile( Profile("pegasus", "label", "p1"))
    wc.addArguments("-l", part)
    wc.setStdout(count)
    wc.uses(part, link=Link.INPUT)
    wc.uses(count, link=Link.OUTPUT, transfer=True, register=True)
    dax.addJob(wc)
```

```
#adding dependency
dax.depends(wc, split)
```

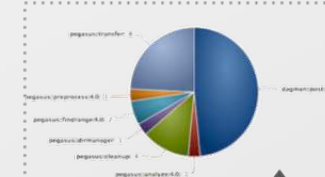
```
f = open("split.dax", "w")
dax.writeXML(f)
f.close()
```



Users

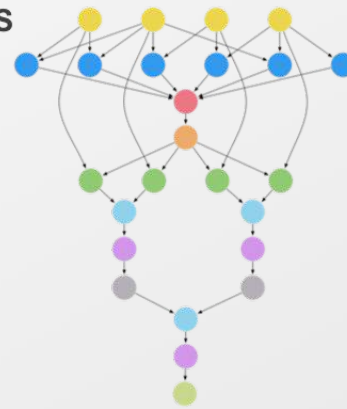
System Architecture

Interfaces



APIs

Pegasus WMS



Submit Host

Mapper

Engine

Scheduler

Pegasus Dashboard

Monitoring
& Provenance

Logs

Notifications

Workflow DB

Job Queue

Clouds

Cloudware

OpenStack, Eucalyptus, Nimbus

Compute

Amazon EC2, Google Cloud,
RackSpace, Chameleon

Storage

Amazon S3, Google Cloud Storage,
OpenStack



Campus
Clusters

Local Clusters

Open Science
Grid

XSEDE

Middleware

HTCondor
GRAM

PBS

LSF

SGE

C
O
M
P
U
T
E

Storage

GridFTP

HTTP

FTP

SRM

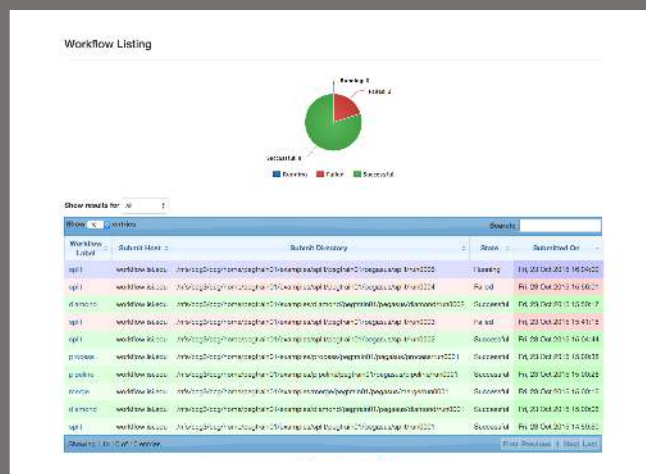
IRODS

SCP



Pegasus

<http://pegasus.isi.edu>



Pegasus dashboard

web interface for monitoring
and debugging workflows



Real-time monitoring of workflow executions. It shows the status of the workflows and jobs, job characteristics, statistics and performance metrics. Provenance data is stored into a relational database.



- Real-time Monitoring
- Reporting
- Debugging
- Troubleshooting
- RESTful API





Pegasus dashboard

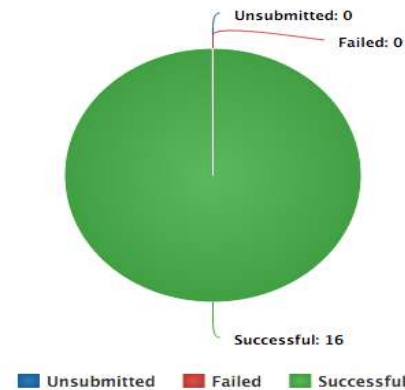
web interface for monitoring
and debugging workflows

Real-time monitoring of
workflow executions. It shows
the status of the workflows and
jobs, job characteristics, statistics
and performance metrics.
Provenance data is stored into a
relational database.

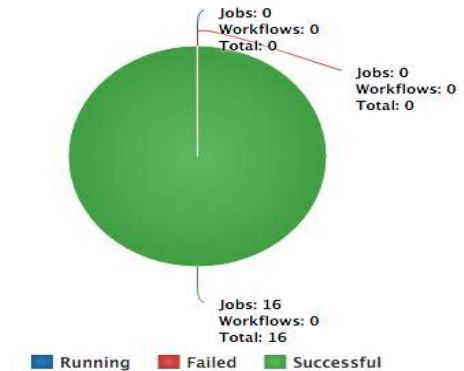
Workflow Details 5bb4de1d-e986-42b8-9160-ab9488494ecf

Label	split
Type	root-wf
Progress	Successful
Submit Host	workflow.isi.edu
User	pegtrain01
Submit Directory	/nfs/ccg3/ccg/home/pegtrain01/examples/split/split/run0002
DAGMan Out File	split-0.dag.dagman.out
Wall Time	12 mins 23 secs
Cumulative Wall Time	9 mins 34 secs

Job Status (Entire Workflow)



Job Status (Per Workflow)





command-line...

```
$ pegasus-status pegasus/examples/split/run0001
STAT IN_STATE JOB
Run 00:39 split-0 (/home/pegasus/examples/split/run0001)
Idle 00:03 └─split_ID0000001
Summary: 2 Condor jobs total (I:1 R:1)

UNRDY READY PRE IN_Q POST DONE FAIL %DONE STATE DAGNAME
14      0      0      1      0      2      0      11.8 Running *split-0.dag
```

```
$ pegasus-analyzer pegasus/examples/split/run0001
pegasus-analyzer: initializing...
```

```
*****Summary*****

Total jobs : 7 (100.00%)
# jobs succeeded : 7 (100.00%)
# jobs failed : 0 (0.00%)
# jobs unsubmitted : 0 (0.00%)
```

```
$ pegasus-statistics -s all pegasus/examples/split/run0001
-----
Type          Succeeded Failed Incomplete Total Retries Total+Retries
Tasks          5         0         0         5         0         5
Jobs           17         0         0        17         0        17
Sub-Workflows   0         0         0         0         0         0
-----
```

```
Workflow wall time : 2 mins, 6 secs
Workflow cumulative job wall time : 38 secs
Cumulative job wall time as seen from submit side : 42 secs
Workflow cumulative job badput wall time :
Cumulative job badput wall time as seen from submit side :
```

Provenance data can be
summarized
pegasus-statistics

or used for debugging
pegasus-analyzer



OUTLINE

Introduction *Scientific Workflows*
Pegasus Overview
Successful Stories

Pegasus Overview *Basic Concepts*
Features
System Architecture

Hands-on Tutorial *Submitting a Workflow*
Workflow Dashboard and Monitoring
Generating the Workflow

Understanding Pegasus Features *Information Catalogs*
Containers

Hands-on Tutorial *Catalogs*
Workflows with Containers
Clustering
Fault-Tolerance

Other Features *Integrity Checking, Data Staging*
Jupyter Notebooks
Metadata, Hierarchical Workflows, Data Reuse

<http://pegasus.isi.edu>

Hands-on Pegasus Tutorial...

Hands On Tutorial

- SSH to our training machine
 - Login with your user's tutorial login and password
 - ssh **pegtrainXX**@workflow.isi.edu
- Open exercise notes in your browser
 - <https://pegasus.isi.edu/tutorial/isi/tutorial.php>

OUTLINE

Introduction *Scientific Workflows*
Pegasus Overview
Successful Stories

Pegasus Overview *Basic Concepts*
Features
System Architecture

Hands-on Tutorial *Submitting a Workflow*
Workflow Dashboard and Monitoring
Generating the Workflow

Understanding Pegasus Features *Information Catalogs*
Containers

Hands-on Tutorial *Catalogs*
Workflows with Containers
Clustering
Fault-Tolerance

Other Features *Integrity Checking, Data Staging*
Jupyter Notebooks
Metadata, Hierarchical Workflows, Data Reuse

<http://pegasus.isi.edu>



Understanding Pegasus features...

So, what information does Pegasus need?



How does Pegasus decide where to execute?

site catalog

transformation catalog

replica catalog

site description

describes the compute resources

scratch

tells where temporary data is stored

storage

tells where output data is stored

profiles

key-pair values associated per job level

```
<!-- The local site contains information about the submit host -->
<!-- The arch and os keywords are used to match binaries in the -->
<!-- transformation catalog -->
<site handle="local" arch="x86_64" os="LINUX">

  <!-- These are the paths on the submit host where Pegasus stores data -->
  <!-- Scratch is where temporary files go -->
  <directory type="shared-scratch" path="/home/tutorial/run">
    <file-server operation="all" url="file:///home/tutorial/run"/>
  </directory>

  <!-- Storage is where pegasus stores output files -->
  <directory type="local-storage" path="/home/tutorial/outputs">
    <file-server operation="all" url="file:///home/tutorial/outputs"/>
  </directory>

  <!-- This profile tells Pegasus where to find the user's private key -->
  <!-- for SCP transfers -->
  <profile namespace="env" key="SSH_PRIVATE_KEY">
    /home/tutorial/.ssh/id_rsa
  </profile>

</site>
```



How does it know where the executables are or which ones to use?

site catalog

transformation catalog

replica catalog

executables description

list of executables locations per site

physical executables

mapped from logical transformations

transformation type

whether it is installed or
available to stage

```
...  
# This is the transformation catalog. It lists information about  
# each of the executables that are used by the workflow.  
  
tr ls {  
  site PegasusVM {  
    pfn "/bin/ls"  
    arch "x86_64"  
    os "linux"  
    type "INSTALLED"  
  }  
}  
...
```

What if data is not local to the submit host?

site catalog

transformation catalog

replica catalog

```
# This is the replica catalog. It lists information about each of the
# input files used by the workflow. You can use this to specify locations to
# input files present on external servers.

# The format is:
# LFN PFN site="SITE"

f.a    file:///home/tutorial/examples/diamond/input/f.a    site="local"
```

logical filename

abstract data name

physical filename

data physical location on site
different transfer protocols
can be used (e.g., scp, http,
ftp, gridFTP, etc.)

site name

in which site the file is available

Replica catalog

multiple sources

site catalog
transformation catalog
replica catalog

pegasus.conf

```
# Add Replica selection options so that it will try URLs first, then
# XrootD for OSG, then gridftp, then anything else
pegasus.selector.replica=Regex
pegasus.selector.replica.regex.rank.1=file:///cvmfs/*.
pegasus.selector.replica.regex.rank.2=file://*.
pegasus.selector.replica.regex.rank.3=root://*.
pegasus.selector.replica.regex.rank.4=gridftp://*.
pegasus.selector.replica.regex.rank.5=.\*
```

rc.data

```
# This is the replica catalog. It lists information about each of the
# input files used by the workflow. You can use this to specify locations
# to input files present on external servers.

# The format is:
# LFN PFN site="SITE"

f.a    file:///cvmfs/oasis.opensciencegrid.org/diamond/input/f.a    site="cvmfs"
f.a    file:///local-storage/diamond/input/f.a    site="prestaged"
f.a    gridftp://storage.mysite/edu/examples/diamond/input/f.a    site="storage"
```

Pegasus Container Support

- Support for
 - Docker
 - Singularity – Widely supported on OSG
- Users can refer to **containers** in the **Transformation Catalog** with their executable preinstalled.
- Users can **refer** to a **container** they want to **use**. However, they let **Pegasus** stage their executable to the node.
 - Useful if you want to use a site recommended/standard container image.
 - Users are using generic image with executable staging.
- **Future Plans**
 - Users can **specify an image buildfile** for their jobs.
 - *Pegasus will build the Docker image as separate jobs in the executable workflow, export them at tar file and ship them around (planned for 4.8.X)*



Pegasus: Data Management

- Treat containers as input data dependency
 - Needs to be staged to compute node if not present
- Users can refer to container images as
 - Docker Hub or Singularity Library URL's
 - Docker Image **exported as a TAR file** and available at a server , just like any other input dataset.
- If an image is specified to be residing in a hub
 - The **image is pulled down** as a tar file as part of **data stage-in** jobs in the workflow
 - The **exported** tar file is then **shipped** with the workflow and made available to the jobs
 - **Motivation: Avoid** hitting Docker Hub/Singularity Library **repeatedly** for large workflows
- Symlink against a container image if available on shared filesystem
 - For e.g. CVMFS hosted images on Open Science Grid

OUTLINE

Introduction	<i>Scientific Workflows Pegasus Overview Successful Stories</i>
Pegasus Overview	<i>Basic Concepts Features System Architecture</i>
Hands-on Tutorial	<i>Submitting a Workflow Workflow Dashboard and Monitoring Generating the Workflow</i>
Understanding Pegasus Features	<i>Information Catalogs Containers</i>
Hands-on Tutorial	<i>Catalogs Workflows with Containers Clustering Fault-Tolerance</i>
Other Features	<i>Integrity Checking, Data Staging Jupyter Notebooks Metadata, Hierarchical Workflows, Data Reuse</i>



Hands-on Pegasus Tutorial...

OUTLINE

Introduction *Scientific Workflows*
Pegasus Overview
Successful Stories

Pegasus Overview *Basic Concepts*
Features
System Architecture

Hands-on Tutorial *Submitting a Workflow*
Workflow Dashboard and Monitoring
Generating the Workflow

Understanding Pegasus Features *Information Catalogs*
Containers

Hands-on Tutorial *Catalogs*
Workflows with Containers
Clustering
Fault-Tolerance

Other Features *Integrity Checking, Data Staging*
Jupyter Notebooks
Metadata, Hierarchical Workflows, Data Reuse



A few more features...

Challenges to Scientific Data Integrity

Modern IT systems are not perfect - errors creep in.

At modern “Big Data” sizes we are starting to see checksums breaking down.

Plus there is the threat of intentional changes: malicious attackers, insider threats, etc.

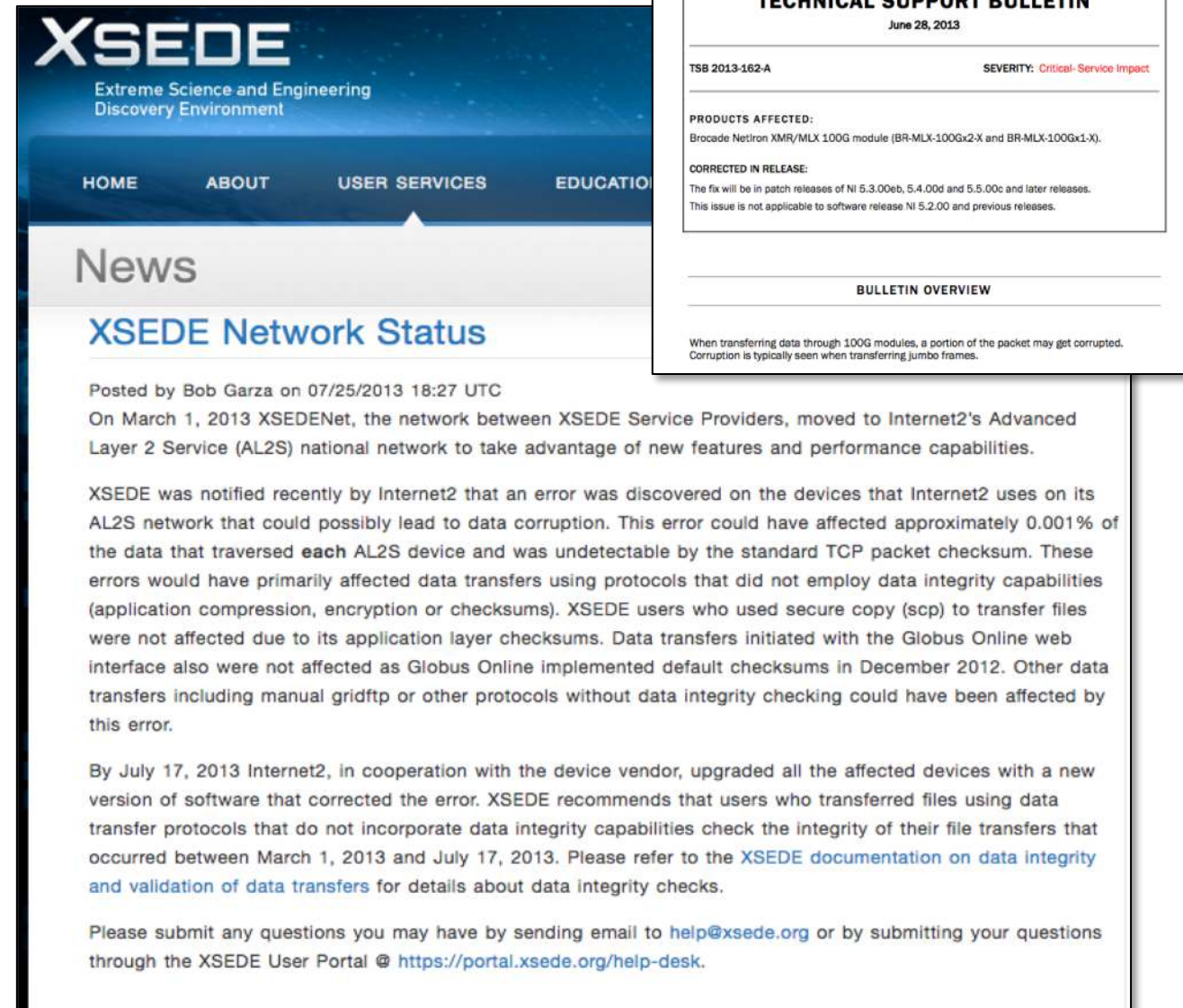
User Perception: “Am I not already protected? I have heard about TCP checksums, encrypted transfers, checksum validation, RAID and erasure coding – is that not enough?”

Motivation: Network Corruption

Network router software
inadvertently corrupts TCP **data**
and/or checksum!

XSEDE and Internet2 example
from 2013.

Second similar case in 2017:
University of Chicago network
upgrade caused data corruption
for the FreeSurfer/Fsurf project.



XSEDE
Extreme Science and Engineering
Discovery Environment

HOME ABOUT USER SERVICES EDUCATION

News

XSEDE Network Status

Posted by Bob Garza on 07/25/2013 18:27 UTC

On March 1, 2013 XSEDENet, the network between XSEDE Service Providers, moved to Internet2's Advanced Layer 2 Service (AL2S) national network to take advantage of new features and performance capabilities.

XSEDE was notified recently by Internet2 that an error was discovered on the devices that Internet2 uses on its AL2S network that could possibly lead to data corruption. This error could have affected approximately 0.001% of the data that traversed **each** AL2S device and was undetectable by the standard TCP packet checksum. These errors would have primarily affected data transfers using protocols that did not employ data integrity capabilities (application compression, encryption or checksums). XSEDE users who used secure copy (scp) to transfer files were not affected due to its application layer checksums. Data transfers initiated with the Globus Online web interface also were not affected as Globus Online implemented default checksums in December 2012. Other data transfers including manual gridftp or other protocols without data integrity checking could have been affected by this error.

By July 17, 2013 Internet2, in cooperation with the device vendor, upgraded all the affected devices with a new version of software that corrected the error. XSEDE recommends that users who transferred files using data transfer protocols that do not incorporate data integrity capabilities check the integrity of their file transfers that occurred between March 1, 2013 and July 17, 2013. Please refer to the [XSEDE documentation on data integrity and validation of data transfers](#) for details about data integrity checks.

Please submit any questions you may have by sending email to help@xsede.org or by submitting your questions through the XSEDE User Portal @ <https://portal.xsede.org/help-desk>.

BROCADE

TECHNICAL SUPPORT BULLETIN

June 28, 2013

TSB 2013-162-A **SEVERITY: Critical-Service Impact**

PRODUCTS AFFECTED:
Brocade Netron XMR/MLX 100G module (BR-MLX-100Gx2-X and BR-MLX-100Gx1-X).

CORRECTED IN RELEASE:
The fix will be in patch releases of NI 5.3.00eb, 5.4.00d and 5.5.00c and later releases.
This issue is not applicable to software release NI 5.2.00 and previous releases.

BULLETIN OVERVIEW

When transferring data through 100G modules, a portion of the packet may get corrupted. Corruption is typically seen when transferring jumbo frames.

Motivation: Software failures

Bug in StashCache data transfer software would occasionally cause silent failure (failed but returned zero).

Failures in the final staging out of data were not detected.

The workflow management system, believing workflow was complete, cleaned up. With the final data being incomplete and all intermediary data lost, ten CPU-years of computing came to naught.

How is this an data integrity issue? The workflow system should have verified that the data at the storage system after the transfer, is the expected data.



Pegasus 4.9.0 Released

on OCTOBER 31, 2018

We are pleased to announce release of Pegasus 4.9.0. Pegasus 4.9.0 is a major release of Pegasus. Highlights of new features: Integrity Checking – Pegasus now performs integrity checking on files in a workflow for non shared filesystem deployments. More details can be found in the documentation at https://pegasus.isi.edu/documentation/integrity_checking.php ... [Read More](#)

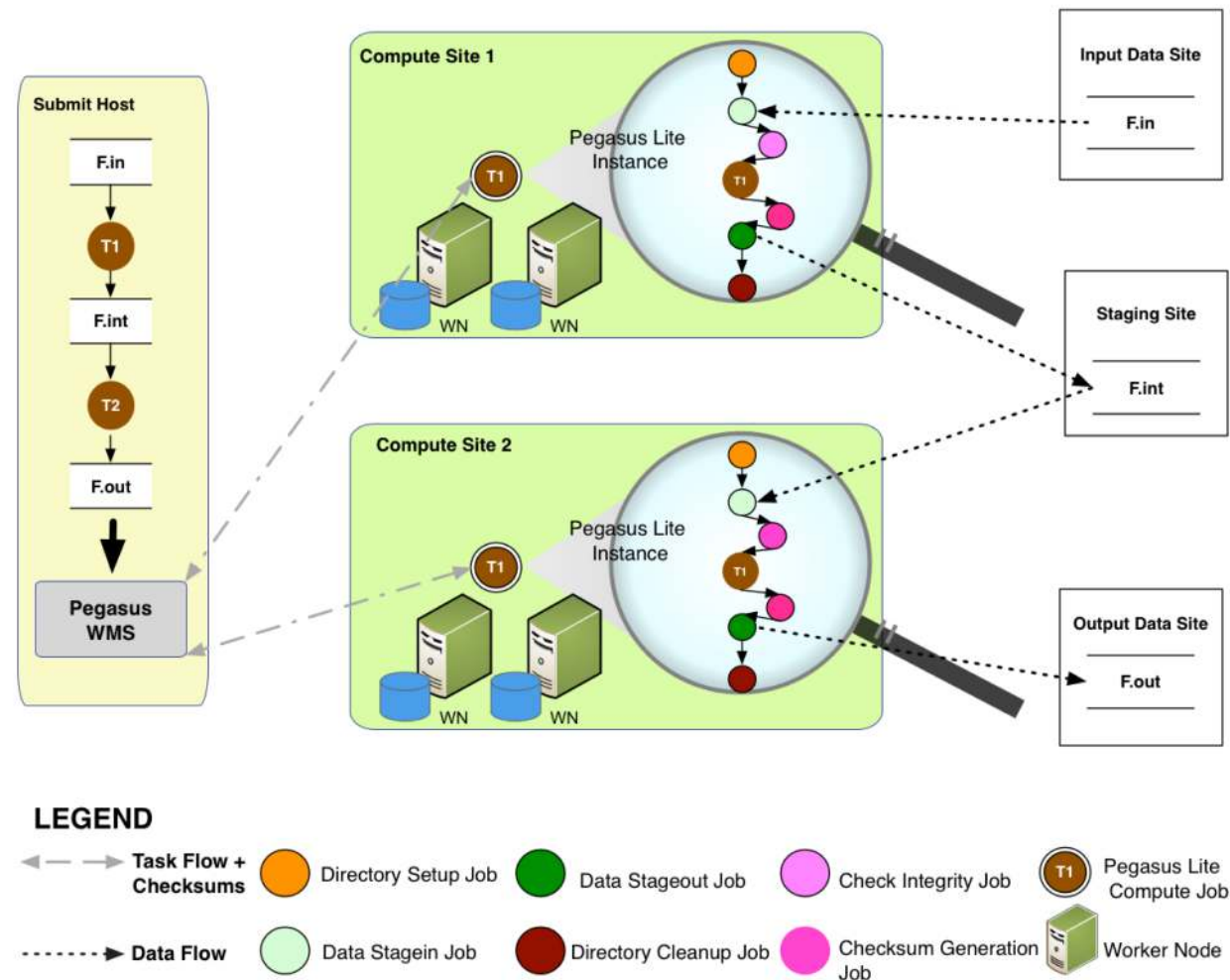
Integrity validation is on by default since the Pegasus 4.9.0 release (Oct 31st, 2018). Users who upgrade will automatically get the protection, but can opt out.

Sharing of detailed monitoring data with the Pegasus team is off by default. Users can opt-in. (We will come back to this at the end of the talk)

Automatic Integrity Checking in Pegasus

Pegasus performs integrity checksums on input files right before a job starts on the remote node.

- For raw inputs, checksums specified in the input replica catalog along with file locations
- All intermediate and output files checksums are generated and tracked within the system.
- Support for sha256 checksums



Job failure is triggered if checksums fail

Data Staging Configurations

HTCondor I/O (HTCondor pools, OSG, ...)

Worker nodes do not share a file system

Data is pulled from / pushed to the submit host via HTCondor file transfers

Staging site is the submit host

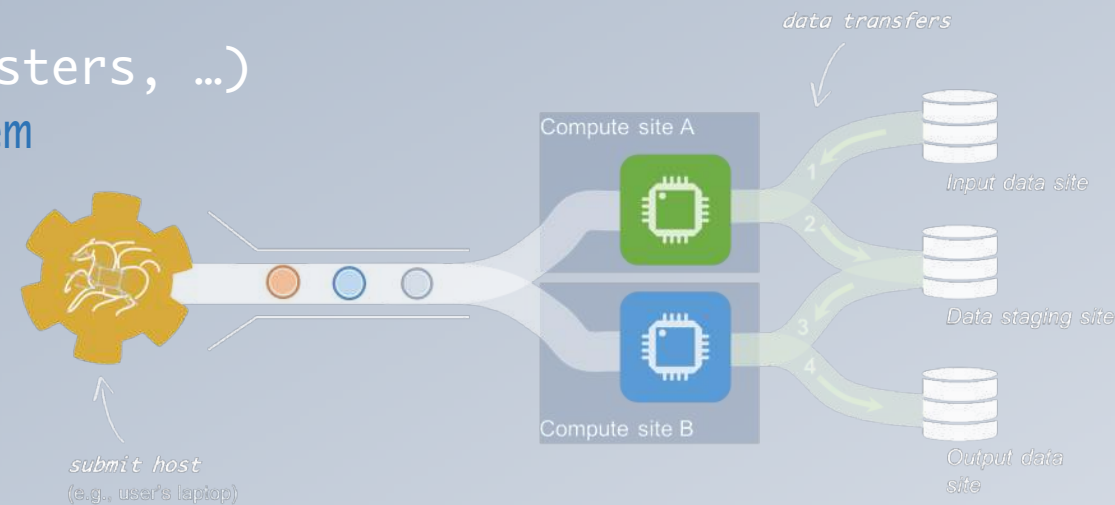
Non-shared File System (clouds, OSG, ...)

Worker nodes do not share a file system

Data is pulled / pushed from a staging site, possibly not co-located with the computation

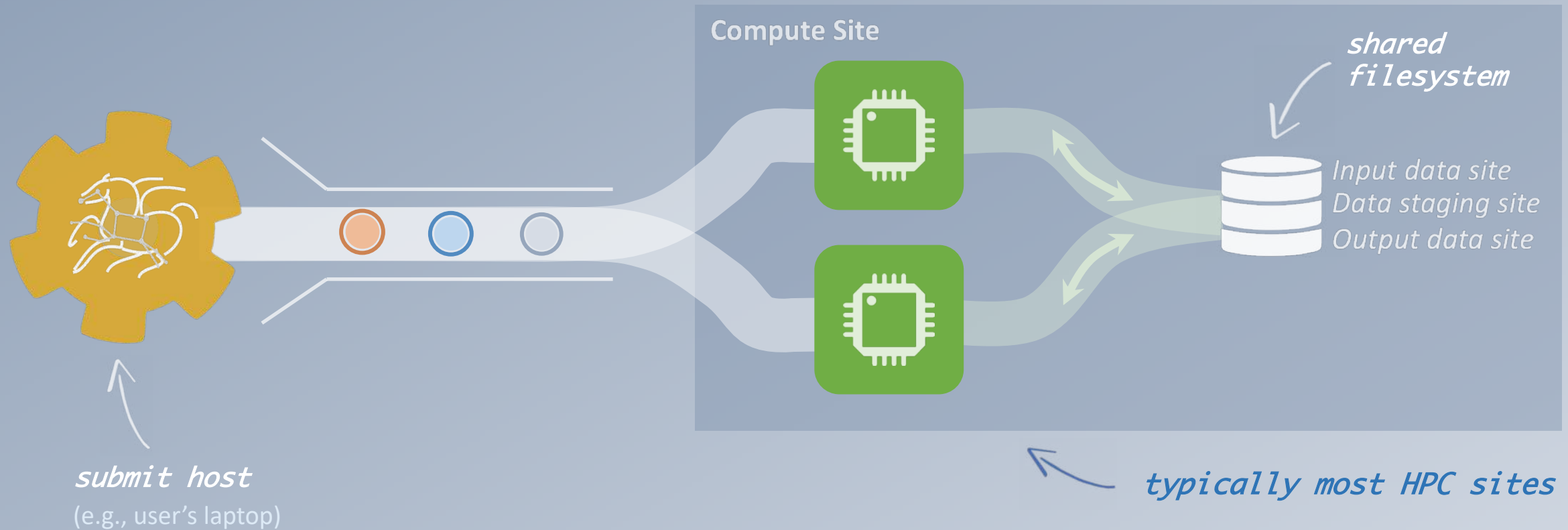
Shared File System (HPC sites, XSEDE, Campus clusters, ...)

I/O is directly against the shared file system

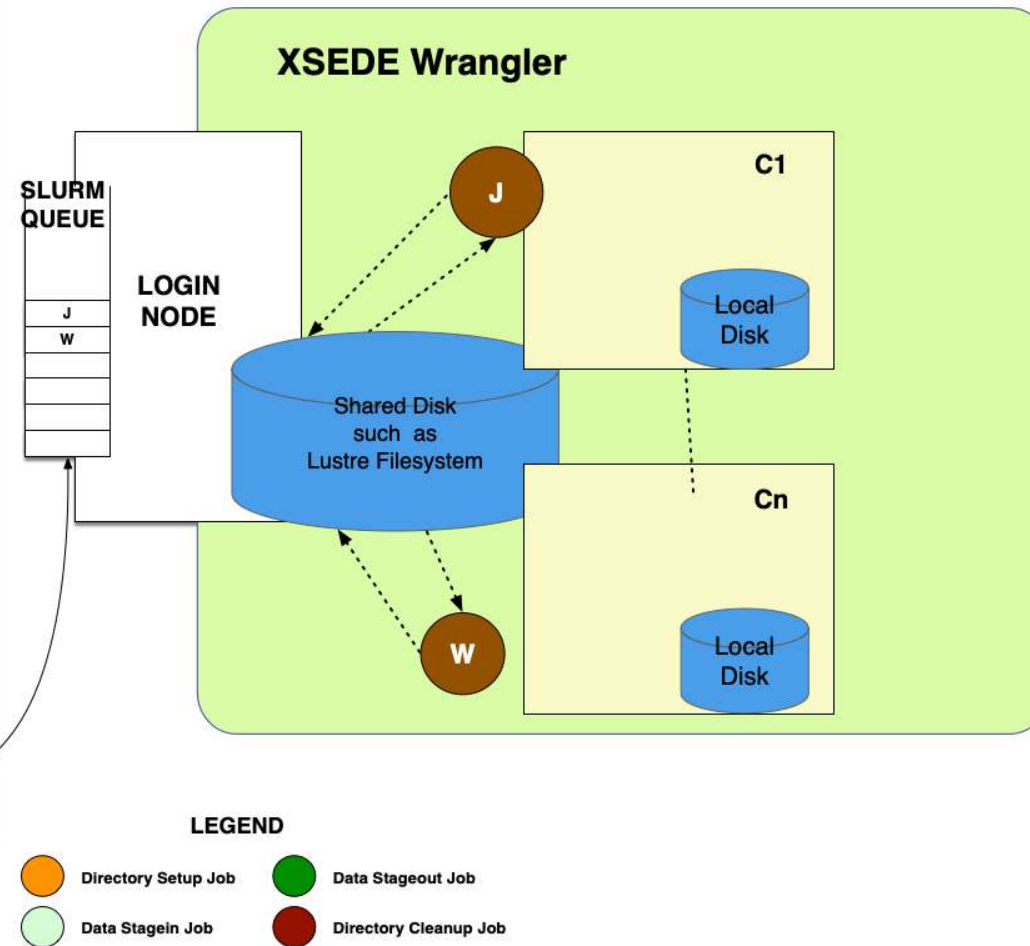
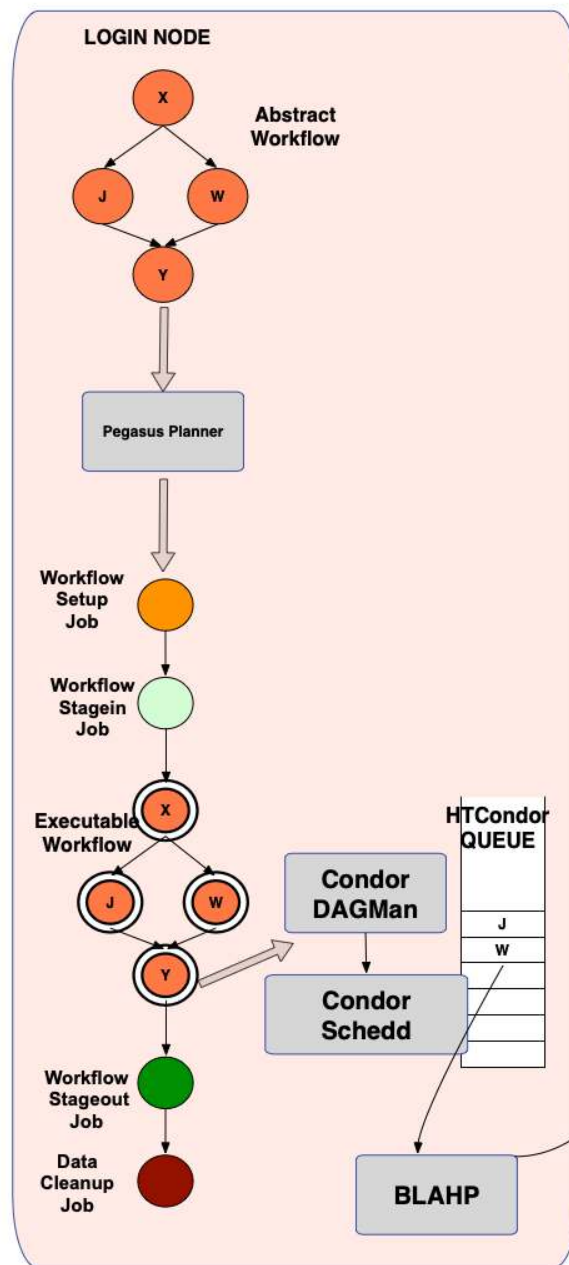


High Performance Computing

There are several possible configurations...

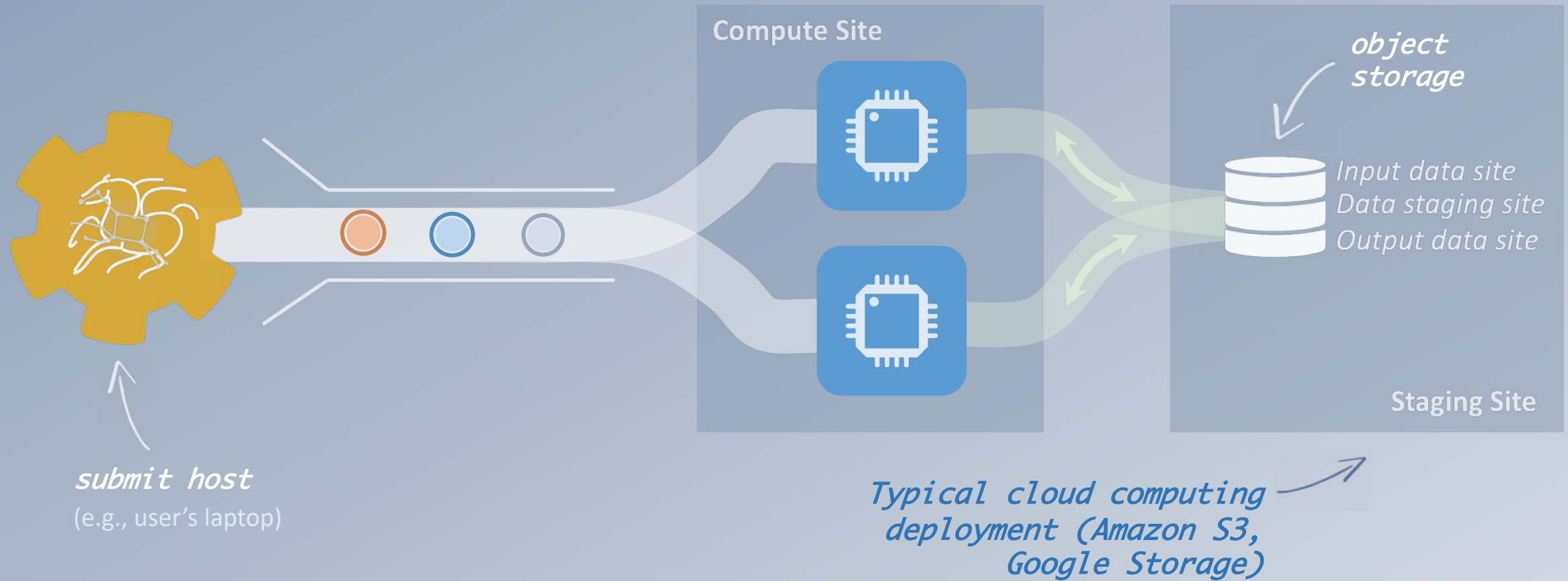


Using Shared FileSystem for Data Access



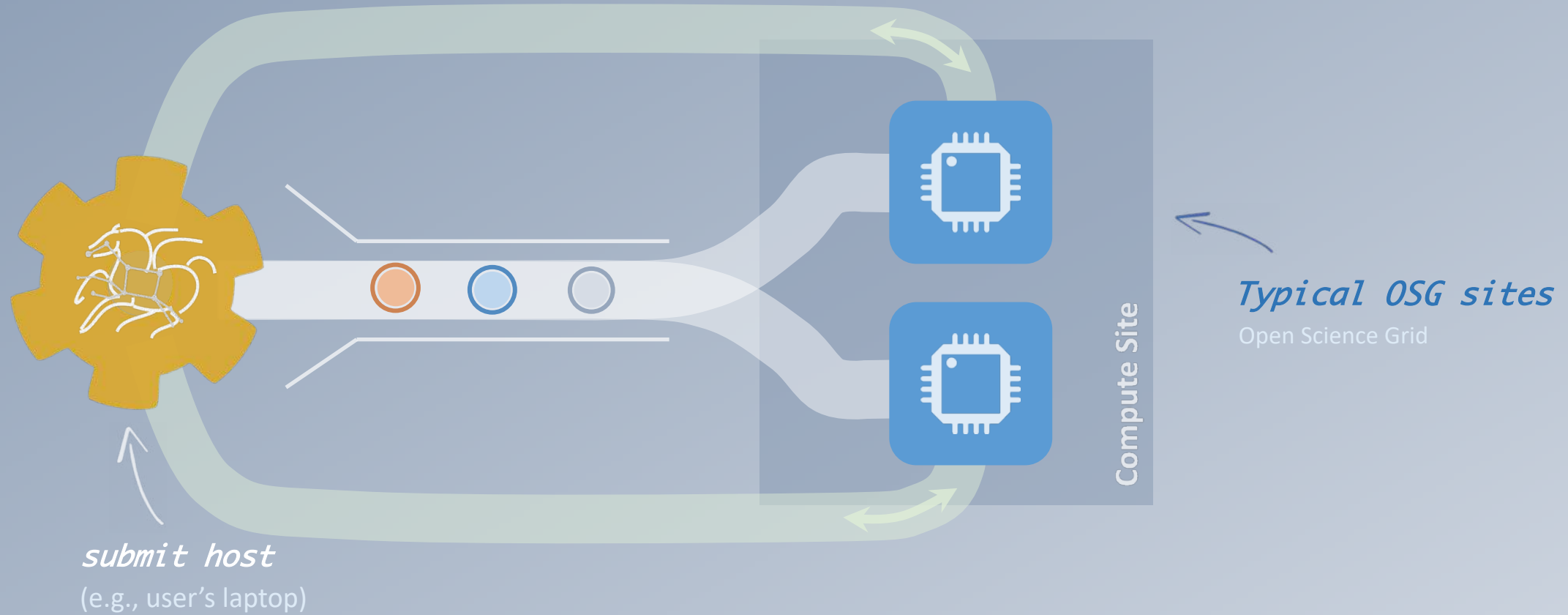
Cloud Computing

high-scalable object storages

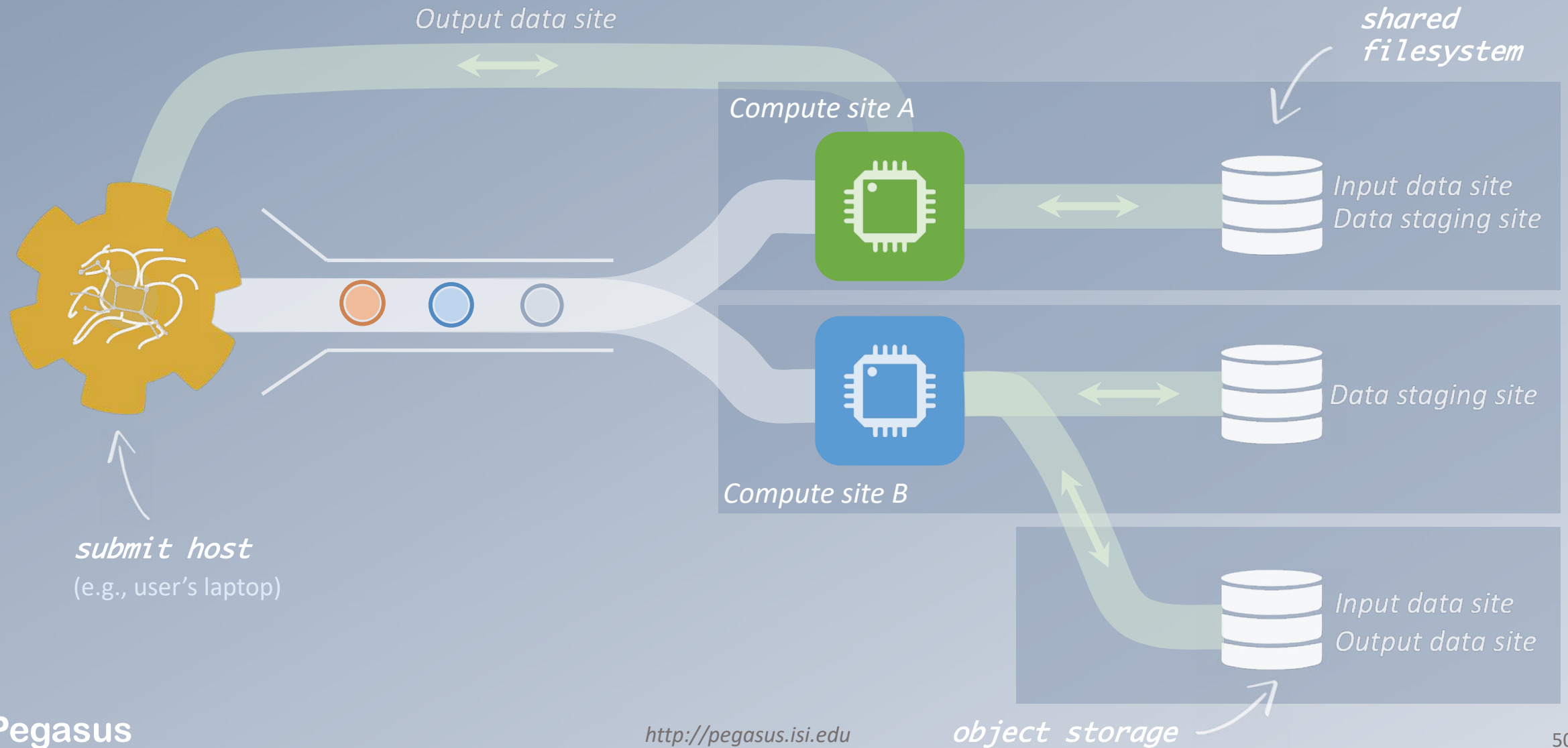


Grid Computing

local data management



And yes... you can mix everything!



pegasus-transfer

Pegasus' internal data transfer tool with support for a number of different protocols

Directory creation, file removal

If protocol supports, used for cleanup

Two stage transfers

e.g., GridFTP to S3 = GridFTP to local file, local file to S3

Parallel transfers

Automatic retries

Credential management

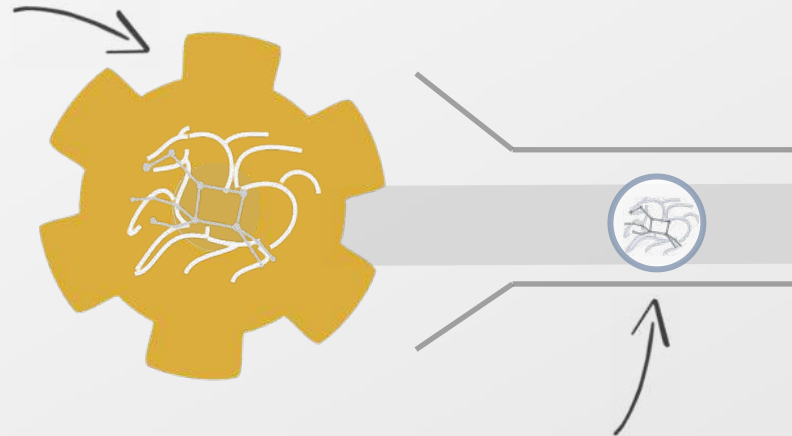
Uses the appropriate credential for each site and each protocol (even 3rd party transfers)

HTTP
SCP
GridFTP
Globus
Online
iRods
Amazon S3
Google
Storage
SRM
FDT
stashcp
cp
ln -s

Running fine-grained workflows on HPC systems...

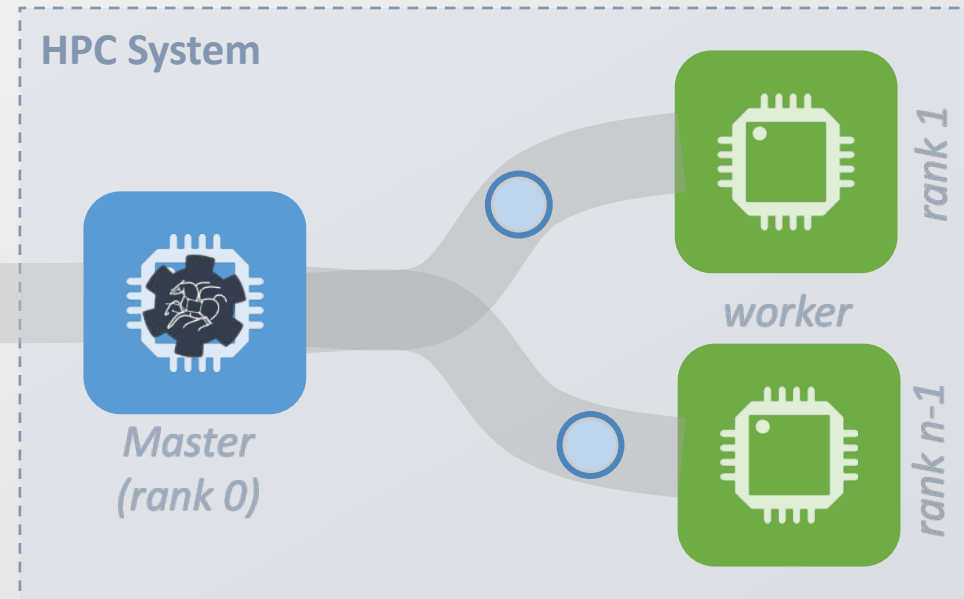
workflow restructuring
workflow reduction
hierarchical workflows
pegasus-mpi-cluster

submit host
(e.g., user's laptop)



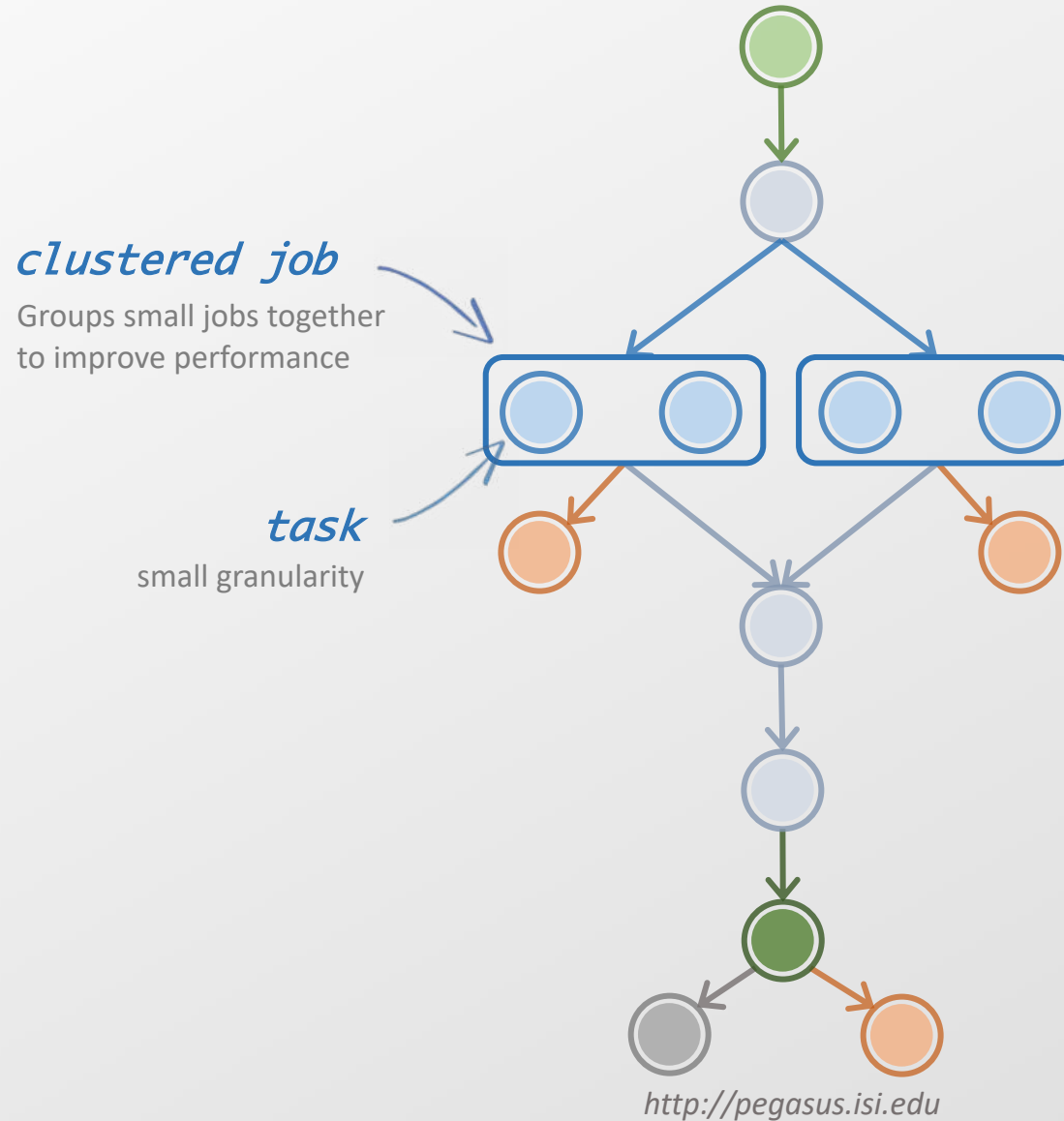
workflow wrapped as an MPI job

Allows sub-graphs of a Pegasus workflow to be submitted as monolithic jobs to remote resources



Performance, why not improve it?

workflow restructuring
workflow reduction
hierarchical workflows
pegasus-mpi-cluster



And if a job fails?

Job Failure Detection

detects non-zero exit code

output parsing for success or failure message

exceeded timeout

do not produced expected output files

Job Retry

helps with transient failures

set number of retries per job and run

Checkpoint Files

job generates checkpoint files

staging of checkpoint files is

automatic on restarts

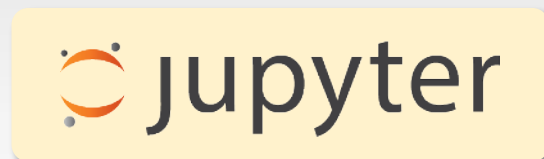
Rescue DAGs

workflow can be restarted from checkpoint file

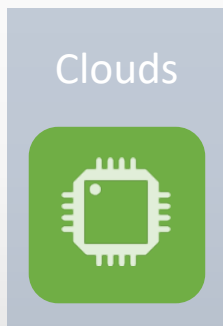
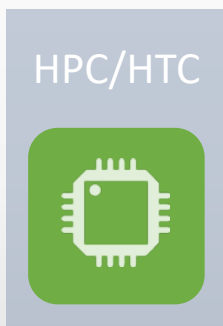
recover from failures with minimal loss



Running Pegasus workflows with Jupyter



WAN LAN



Jupyter Pegasus-Tutorial-Split Last Checkpoint: 03/15/2017 (autosaved)

File Edit View Insert Cell Kernel Widgets Help Python 2

After the workflow has been submitted you can monitor it using the `status()` method. This method takes two arguments:

- `loop`: whether the status command should be invoked once or continuously until the workflow is completed or a failure is detected.
- `delay`: The delay (in seconds) the status will be refreshed. Default value is 10s.

```
In [6]: instance.status(loop=True, delay=5)
```

Progress: 100.0% (Success) (Completed: 17, Queued: 0, Running: 0, Failed: 0)

Once the workflow execution is completed, a list of the output files can be obtained using the `outputs()` command.

```
File for submitting this DAG to Condor: split-0.dag.condor.sub
Log of DAGMan debugging messages: split-0.dag.dagman.out
Log of Condor library output: split-0.dag.lib.out
Log of Condor library error messages: split-0.dag.lib.err
Log of the life of condor_dagman itself: split-0.dag.dagman.log

Your database is compatible with Pegasus version: 4.7.0
Submitting to condor split-0.dag.condor.sub
Submitting job(s).
1 job(s) submitted to cluster 1068.

Your workflow has been started and is running in the base directory: a relative path of the file from the
/Users/silva/Downloads/split-submit-host-2017-03-27T10:17:45/submit/silva/pegasus/split/run0002

*** To monitor the workflow you can run ***

pegasus-status -l /Users/silva/Downloads/split-submit-host-2017-03-27T10:17:45/submit/silva/pegasus/split/run0002
```

Pegasus-Jupyter Python API

```
from Pegasus.jupyter.instance import *
```

importing the API

```
instance = Instance(dax)
```

*creating an instance
of the DAX*

```
instance.run(site='condorpool')
```

running a workflow

```
# Create an abstract dag
```

```
dax = ADAG("split")
```

```
# the split job that splits the webpage into smaller chunks
```

```
split = Job("split")
```

```
split.addArguments("-l", "100", "-a", "1", webpage, "part.")
```

```
split.uses(webpage, link=Link.INPUT)
```

```
# associate the label with the job. All jobs with same label
```

```
# are run with PMC when doing job clustering
```

```
split.addProfile( Profile("pegasus", "label", "p1"))
```

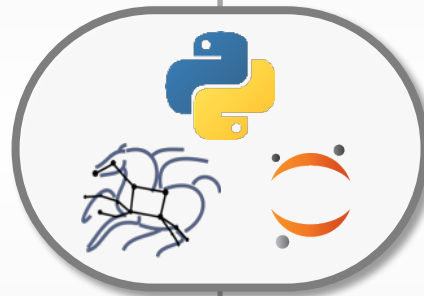
```
dax.addJob(split)
```

*using the Pegasus DAX3 API
to write a workflow*

```
instance.status(loop=True, delay=5)
```

monitoring a workflow execution

```
Progress: 100.0% (Success) (Completed: 17, Queued: 0, Running: 0, Failed: 0)
```



Metadata

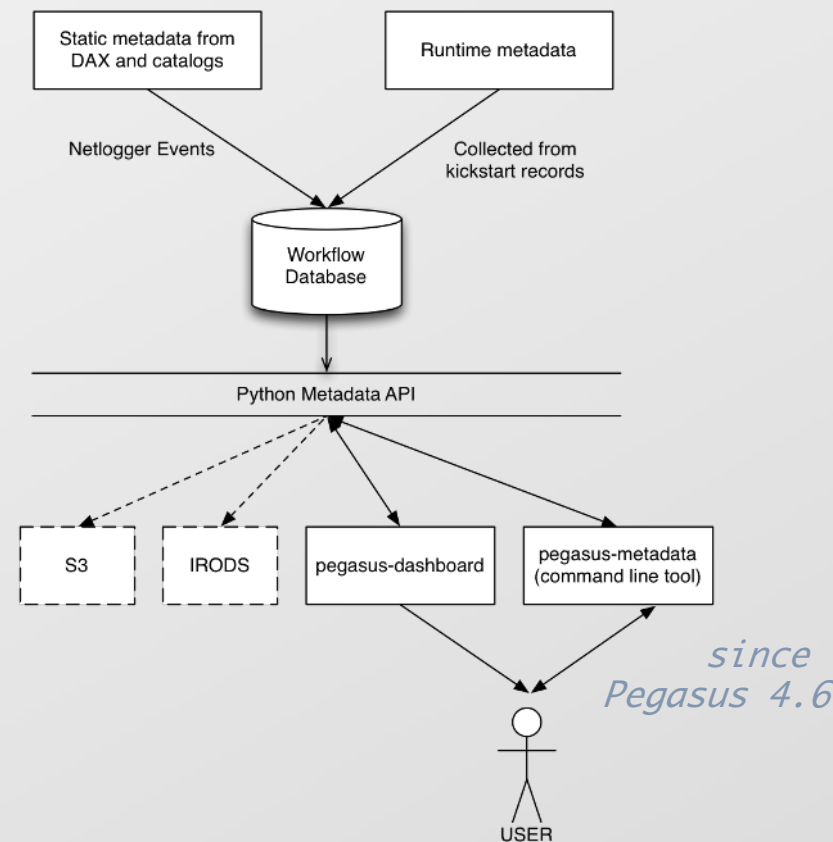
Can associate arbitrary key-value pairs with workflows, jobs, and files

Data registration

Output files get tagged with metadata on registration in the workflow database

Static and runtime metadata

*Static: application parameters
Runtime: performance metrics*



```
1 <adag ...>
2   <metadata key="experiment">par_all27_prot_lipid</metadata>
3   <job id="ID0000001" name="namd">
4     <argument><file name="equilibrate.conf"/></argument>
5     <metadata key="timesteps">500000</metadata>
6     <metadata key="temperature">200</metadata>
7     <metadata key="pressure">1.01325</metadata>
8     <uses name="Q42.psf" link="input">
9       <metadata key="type">psf</metadata>
10      <metadata key="charge">42</metadata>
11    </uses>
12    ...
13    <uses name="eq.restart.coord" link="output" transfer="false">
14      <metadata key="type">coordinates</metadata>
15    </uses>
16    ...
17  </job>
18 </adag>
```

*workflow,
job, file*

*select data
based on metadata*

*register data
with metadata*

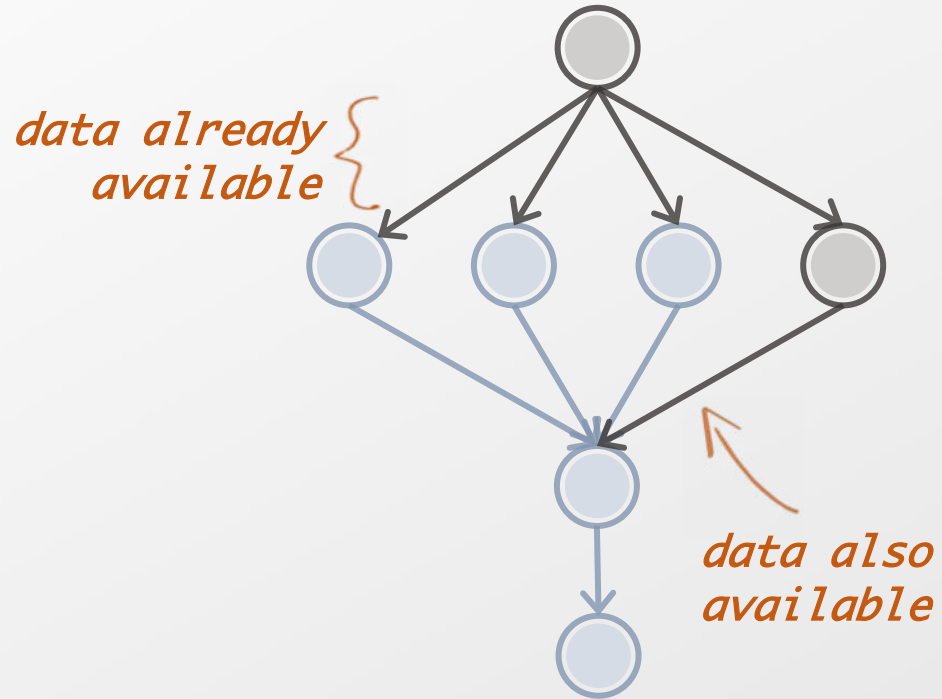
What about data reuse?

workflow restructuring

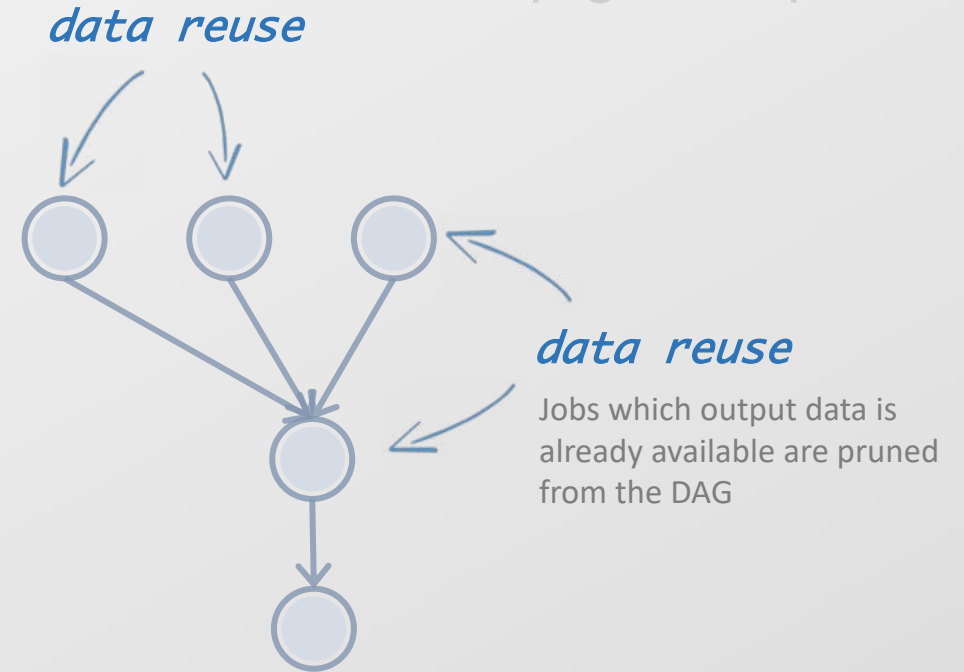
workflow reduction

hierarchical workflows

pegasus-mpi-cluster

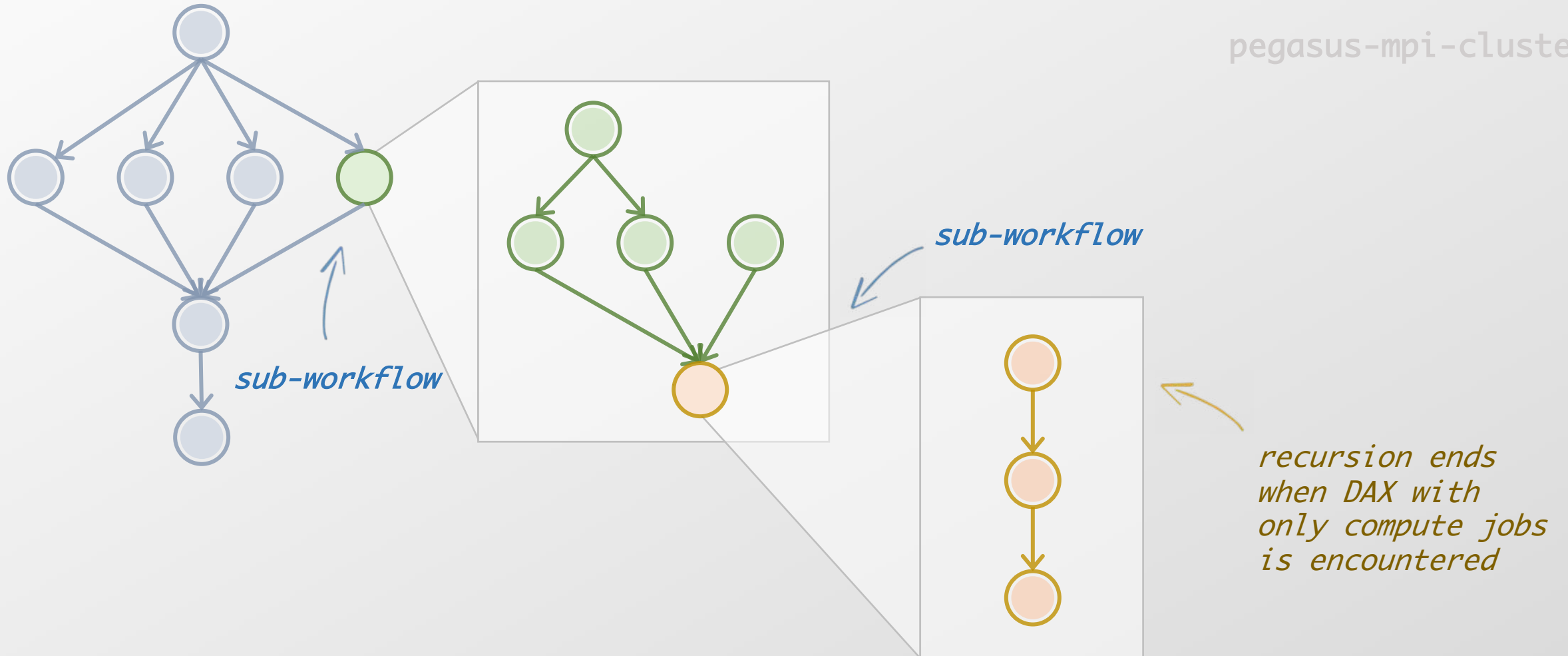


workflow reduction



Pegasus also handles large-scale workflows

workflow restructuring
workflow reduction
hierarchical workflows
pegasus-mpi-cluster



Job Submissions

Local

Submit Machine

Personal HTCondor

*Local Campus Cluster accessible via
Submit Machine **

HTCondor via Glite

*** Both Glite and BOSCO build on HTCondor BLAHP
Support.*

Supported schedulers

PBS SGE SLURM MOAB

Remote

*BOSCO + SSH***

*Each node in executable workflow
submitted via SSH connection to
remote cluster*

*BOSCO based Glideins***

SSH based submission of Glideins

PyGlidein

ICE Cube Glidein service

OSG using glideinWMS

CREAMCE

Uses CondorG

Globus GRAM

Uses CondorG





Pegasus est. 2001

Automate, recover, and debug scientific computations.

Get Started

Pegasus Website

<http://pegasus.isi.edu>

Users Mailing List

pegasus-users@isi.edu

Support

pegasus-support@isi.edu

Pegasus Online Office Hours

<https://pegasus.isi.edu/blog/online-pegasus-office-hours/>

Bi-monthly basis on second Friday of the month, where we address user questions and also apprise the community of new developments