Before we start

Hands on Exercises Notes	*https://pegasus.isi.edu/tutorial/chtc/*
System	*learn.chtc.wisc.edu*
Training Accounts	*Pick up from the instructor*

Pegasus Scientific Workflows with Containers

Pegasus Workflow Management System

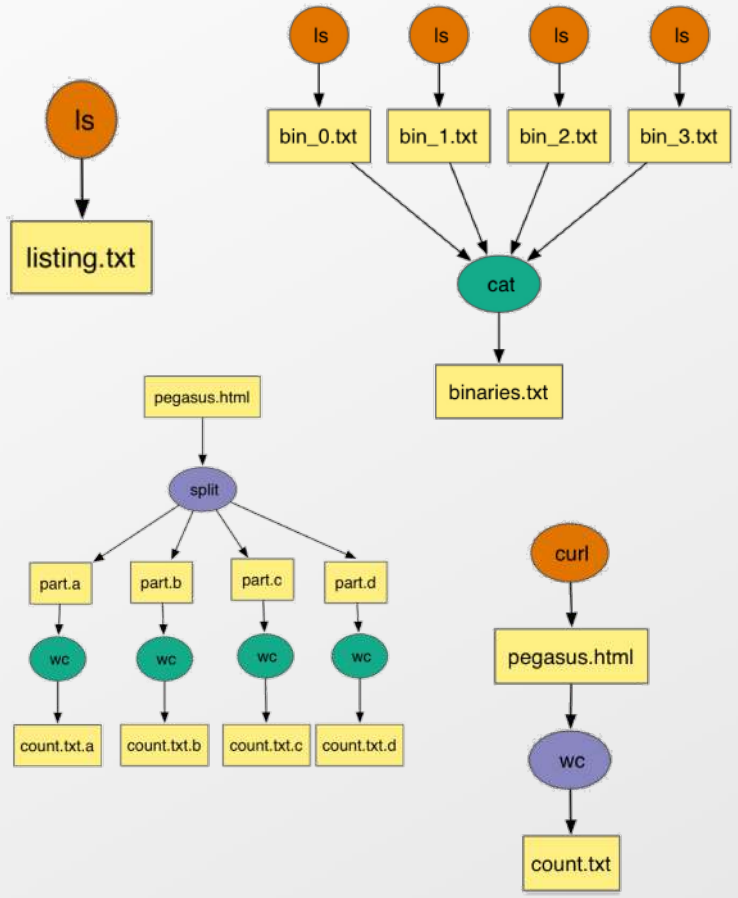
Karan Vahi

USC Viterbi
School of Engineering
Information Sciences Institute

http://pegasus.isi.edu

OUTLINE

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**

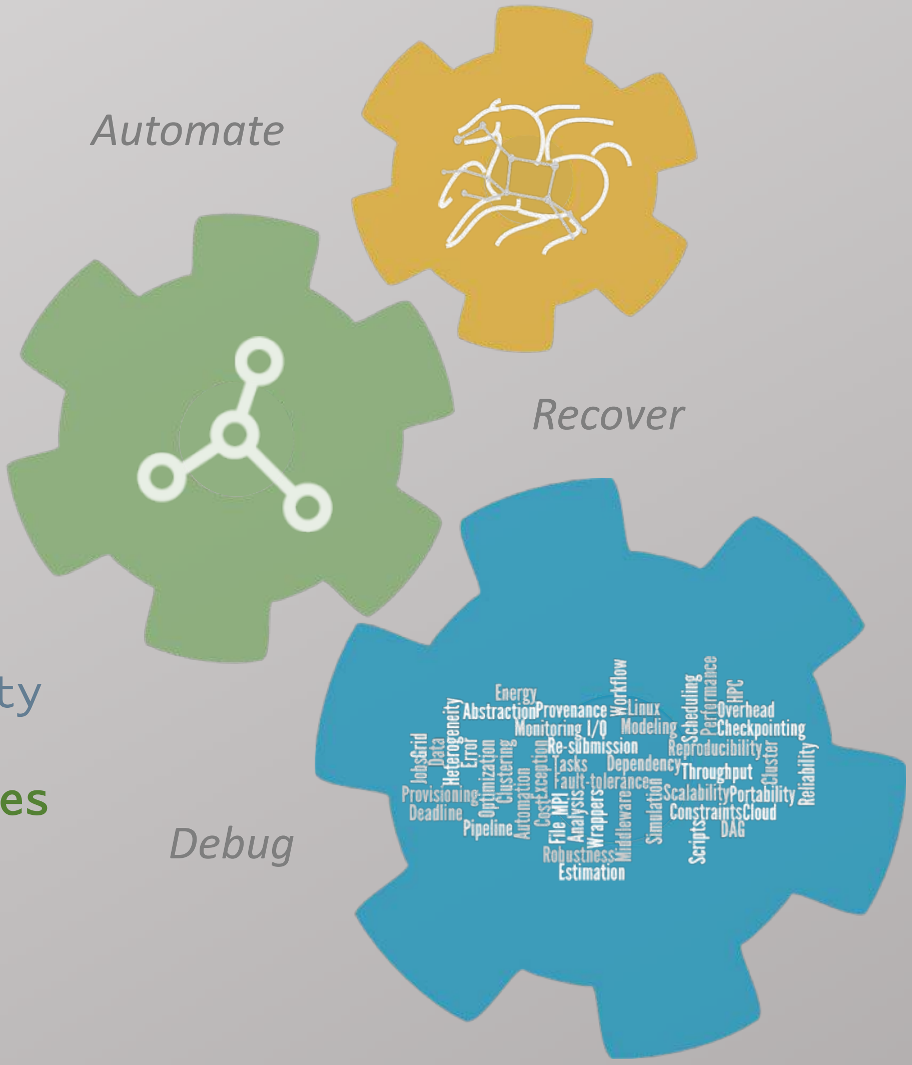
Automatically executes data transfers

Reusable, aids **reproducibility**

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

Handles **failures** with to provide reliability

Keeps track of data and **files**

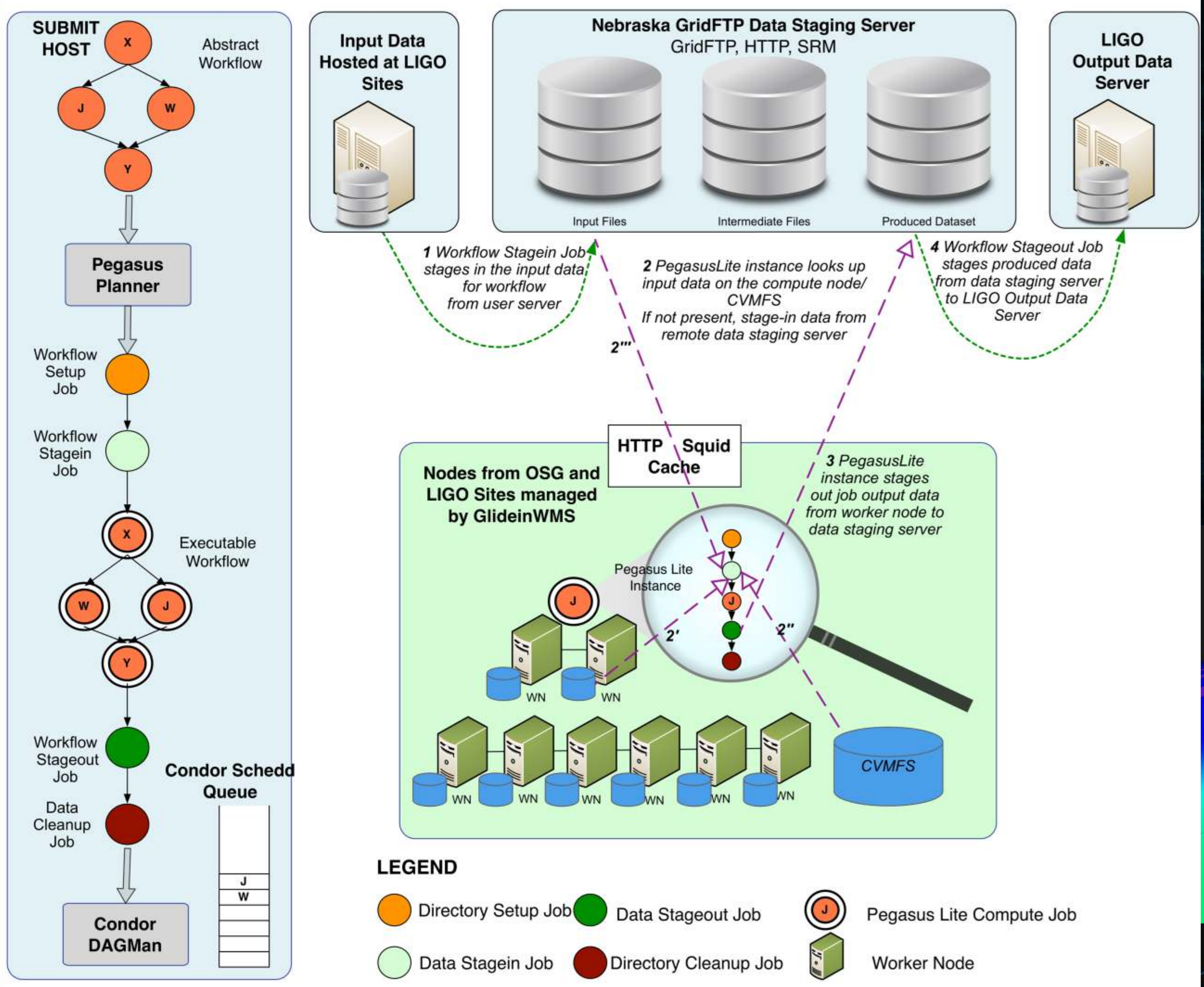


HTCondor High Throughput Computing

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

Some of the successful stories…

Data Flow for LIGO Pegasus Workflows in OSG



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

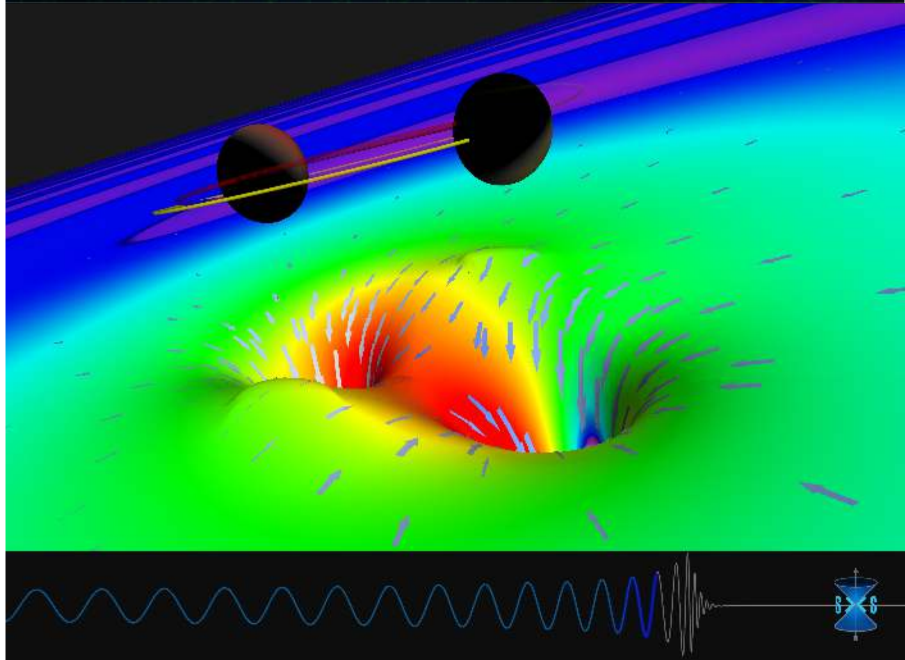
3 PegasusLite instance stages out job output data from worker node to data staging server

4 Workflow Stageout Job stages produced data from data staging server to LIGO Output Data Server

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

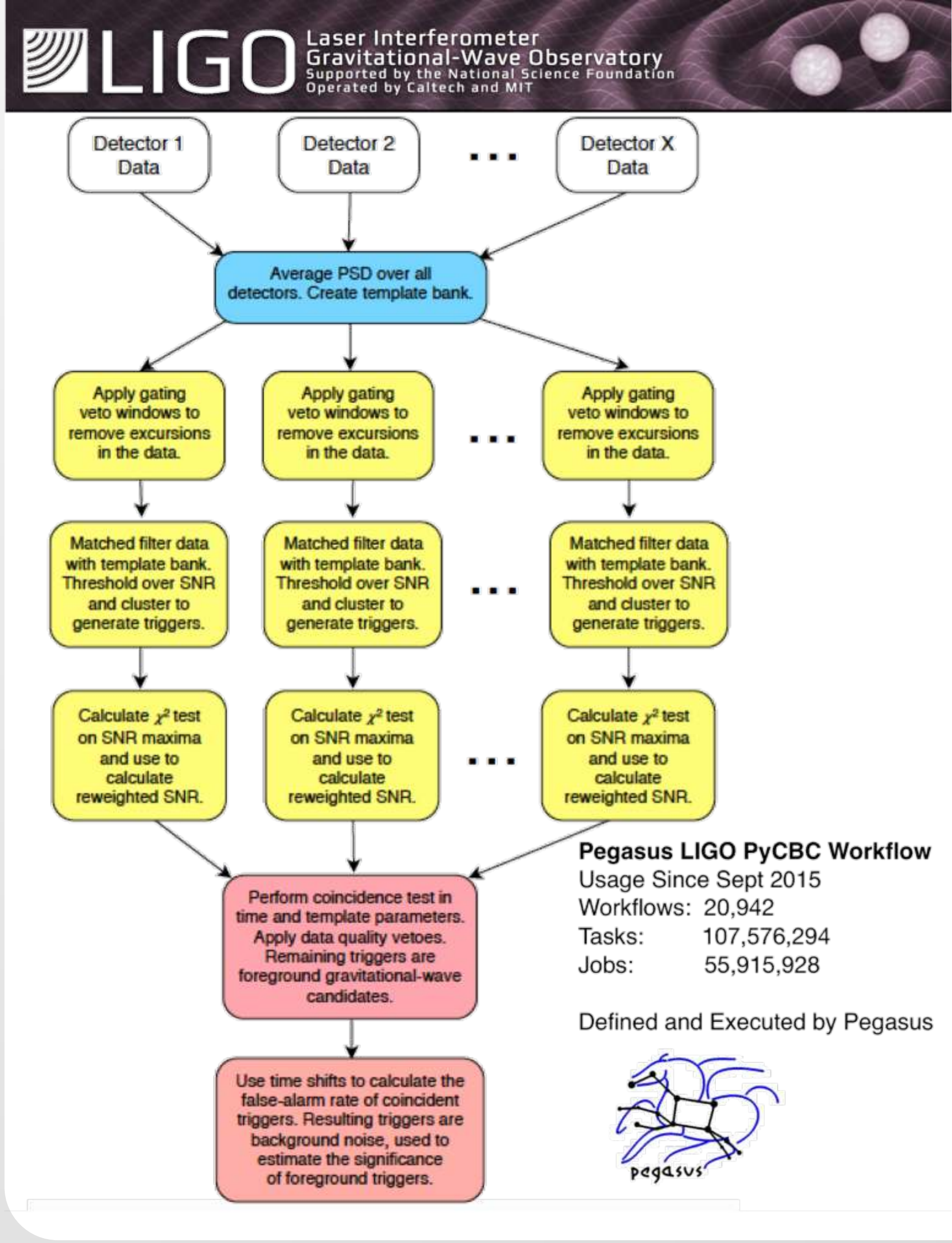
Contains **100's of thousands of jobs** and accesses on order of **terabytes of data**

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**


LIGO Laser Interferometer Gravitational-Wave Observatory
Supported by the National Science Foundation
Operated by Caltech and MIT
Detector 1 Data
Detector 2 Data
...
Detector X Data
Average PSD over all detectors. Create template bank.
Apply gating veto windows to remove excursions in the data.
Matched filter data with template bank. Threshold over SNR and cluster to generate triggers.
Calculate χ^2 test on SNR maxima and use to calculate reweighted SNR.
Perform coincidence test in time and template parameters. Apply data quality vetoes. Remaining triggers are foreground gravitational-wave candidates.
Use time shifts to calculate the false-alarm rate of coincident triggers. Resulting triggers are background noise, used to estimate the significance of foreground triggers.
pegasus

Pegasus LIGO PyCBC Workflow
Usage Since Sept 2015

Workflows:	20,942
Tasks:	107,576,294
Jobs:	55,915,928

Defined and Executed by Pegasus

PyCBC Papers: An improved pipeline to search for gravitational waves from compact binary coalescence. *Samantha Usman, Duncan Brown et al.*
The PyCBC search for gravitational waves from compact binary coalescence, *Samantha Usman et al* (https://arxiv.org/abs/1508.02357)
PyCBC Detection GW150914: First results from the search for binary black hole coalescence with Advanced LIGO. *B. P. Abbott et al.*

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

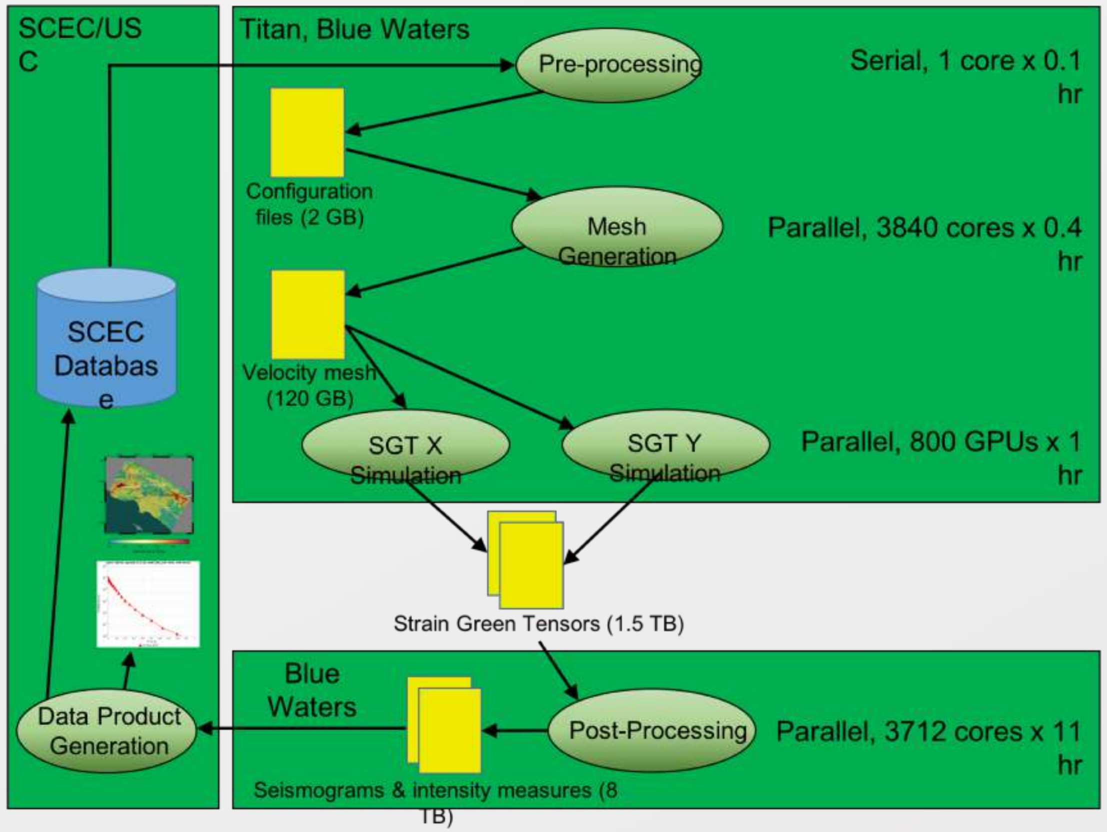
  200 GPUs for 1 hour

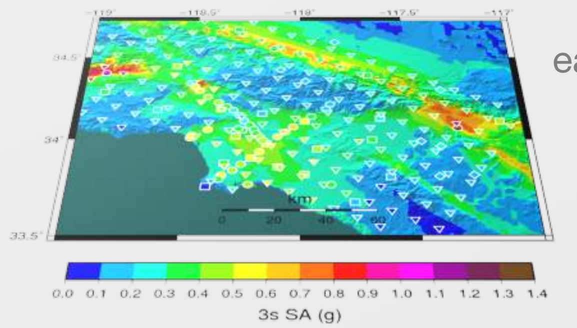
Titan:

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

Blue Waters:

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





286 sites, 4 models
each workflow has **420,000 tasks**

Pegasus

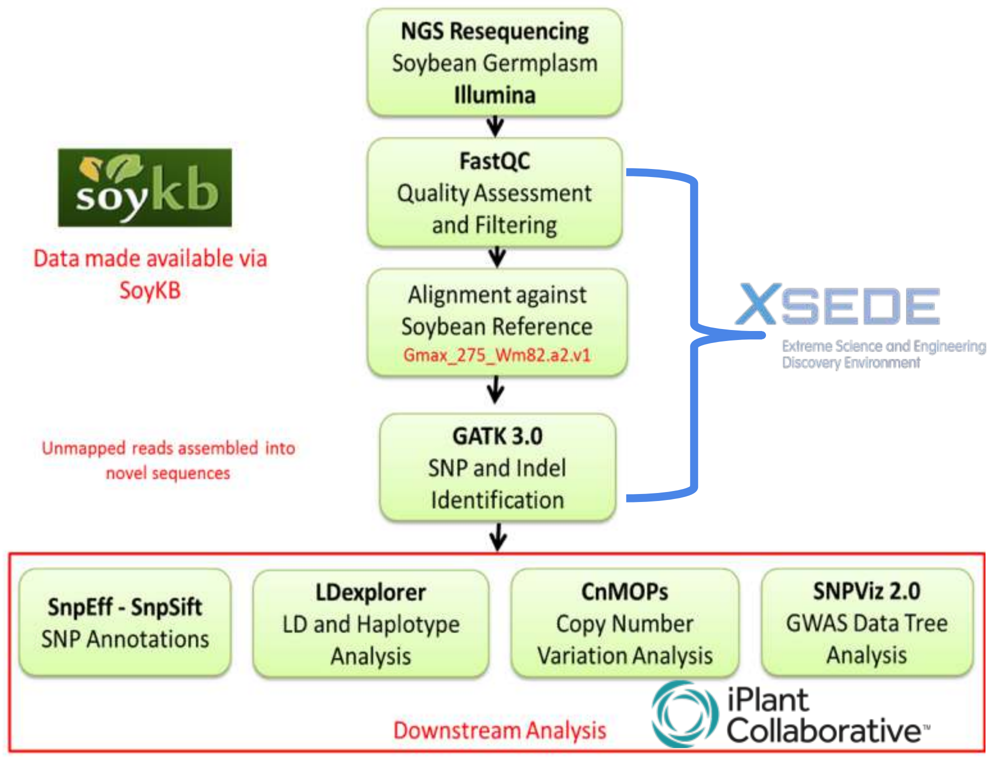
Soybean Workflow

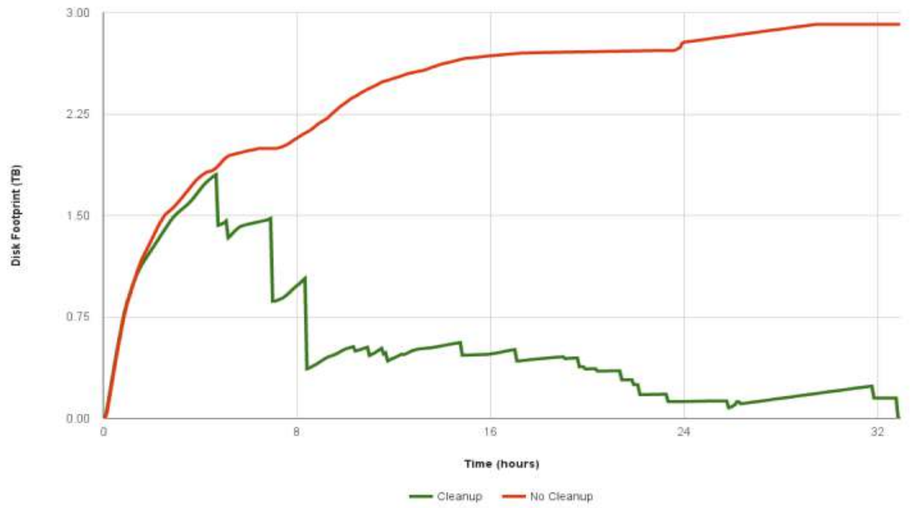
TACC Wrangler as Execution Environment

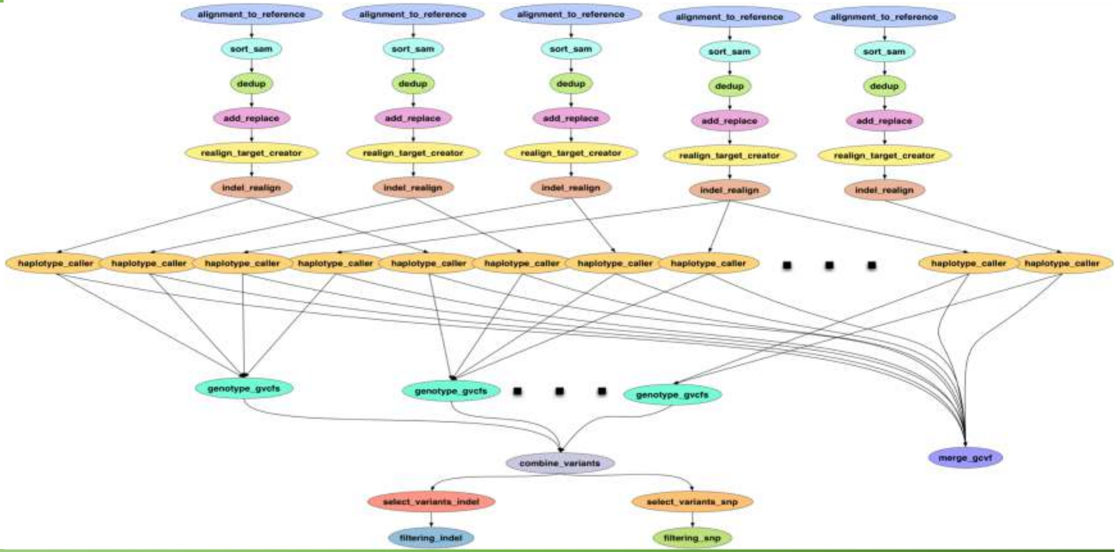
Flash Based Shared Storage

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)


NGS Resequencing
Soybean Germplasm
Illumina
FastQC
Quality Assessment and Filtering
Alignment against Soybean Reference
Gmax_275_Wm82.a2.v1
GATK 3.0
SNP and Indel Identification
soykb
Data made available via SoyKB
Unmapped reads assembled into novel sequences
XSEDE
Extreme Science and Engineering Discovery Environment
SnpEff - SnpSift
SNP Annotations
LDexplorer
LD and Haplotype Analysis
CnMOPs
Copy Number Variation Analysis
SNPViz 2.0
GWAS Data Tree Analysis
Downstream Analysis
iPlant Collaborative™





OUTLINE

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

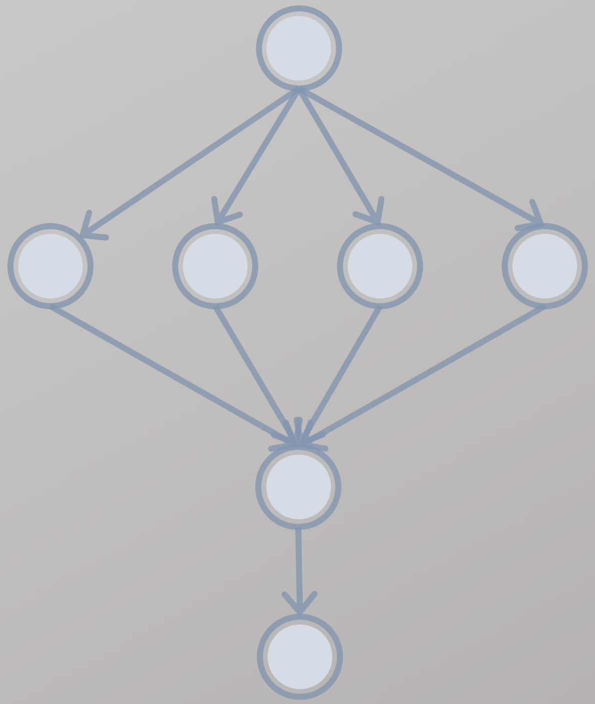
Workflows are DAGs

- 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

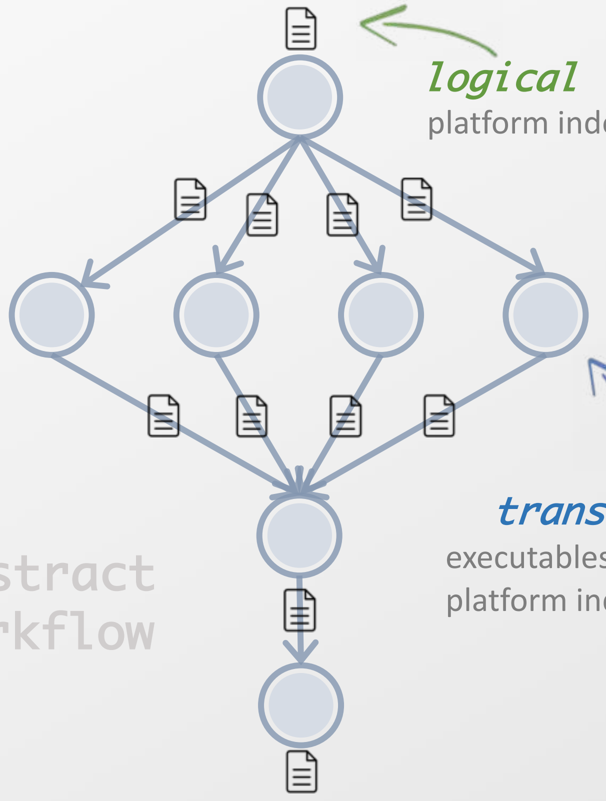
DAG in XML

DAG

directed-acyclic graphs

Portable Description

Users do not worry about
low level execution details

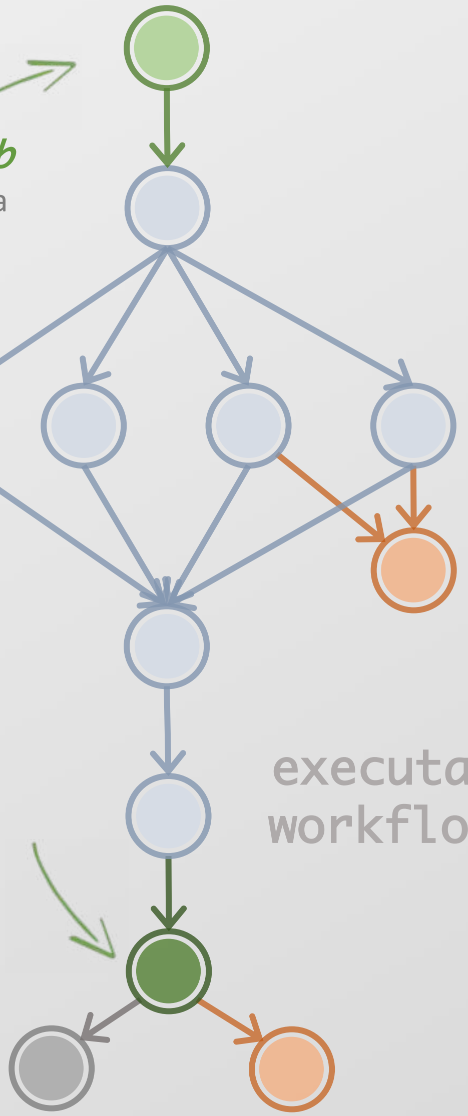


logical filename (LFN)

platform independent (abstraction)

transformation

executables (or programs)
platform independent



stage-in job

Transfers the workflow input data

cleanup job

Removes unused data

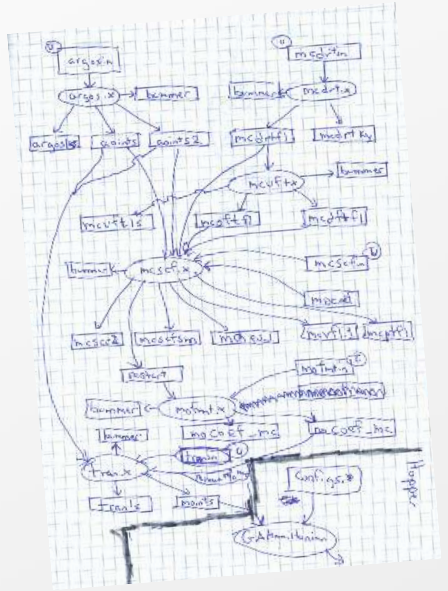
stage-out job

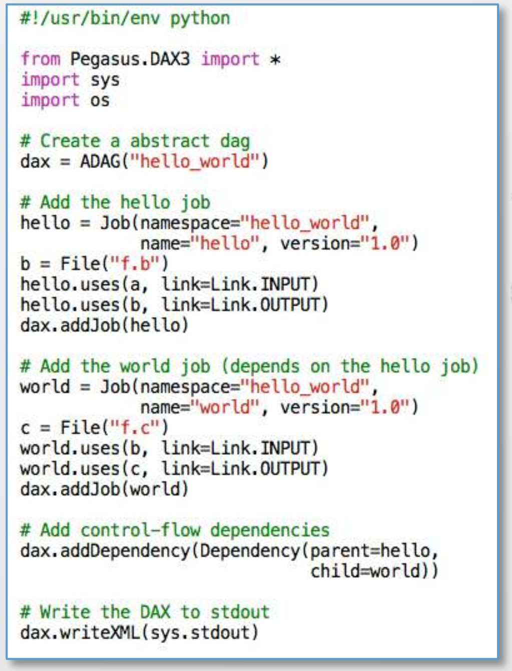
Transfers the workflow output data

registration job

Registers the workflow output data

Pegasus also provides tools to generate the abstract workflow





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

from Pegasus.DAX3 import *
import sys
import os

# Create a 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)
```

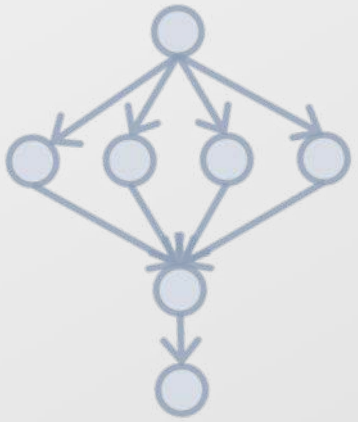
python™

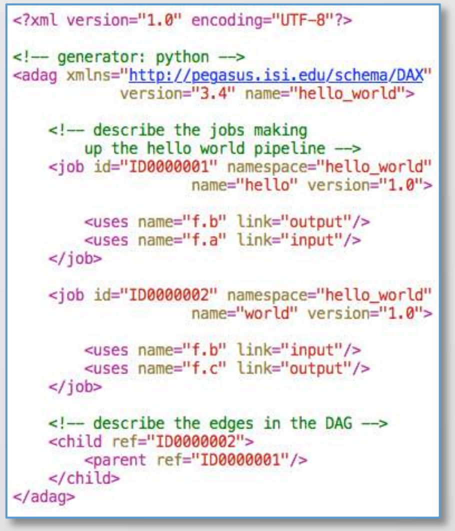
Java

R

perl

jupyter



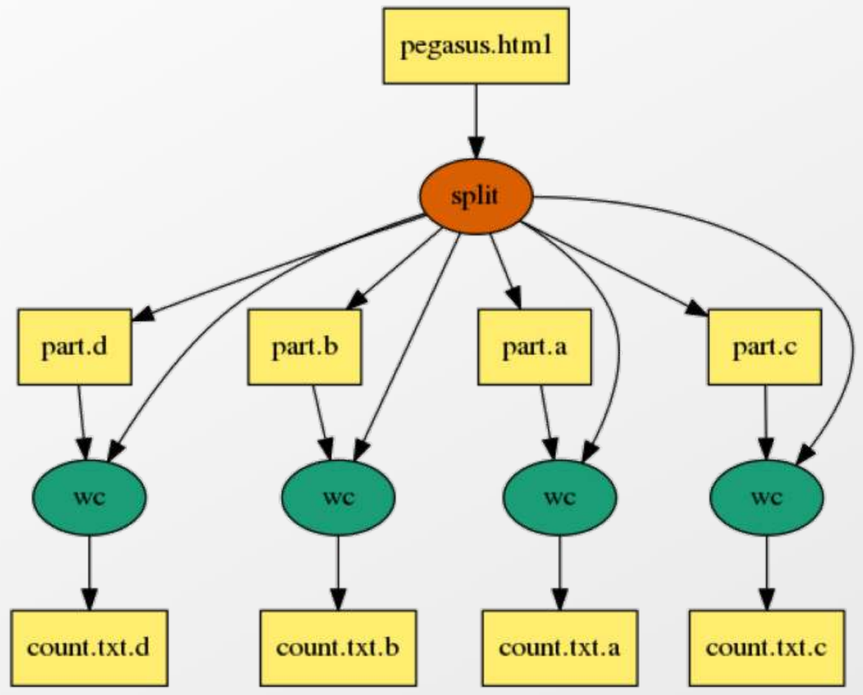


```
<?xml version="1.0" encoding="UTF-8"?>

<!-- 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>
```

DAX

DAG in XML

An example
Split Workflow

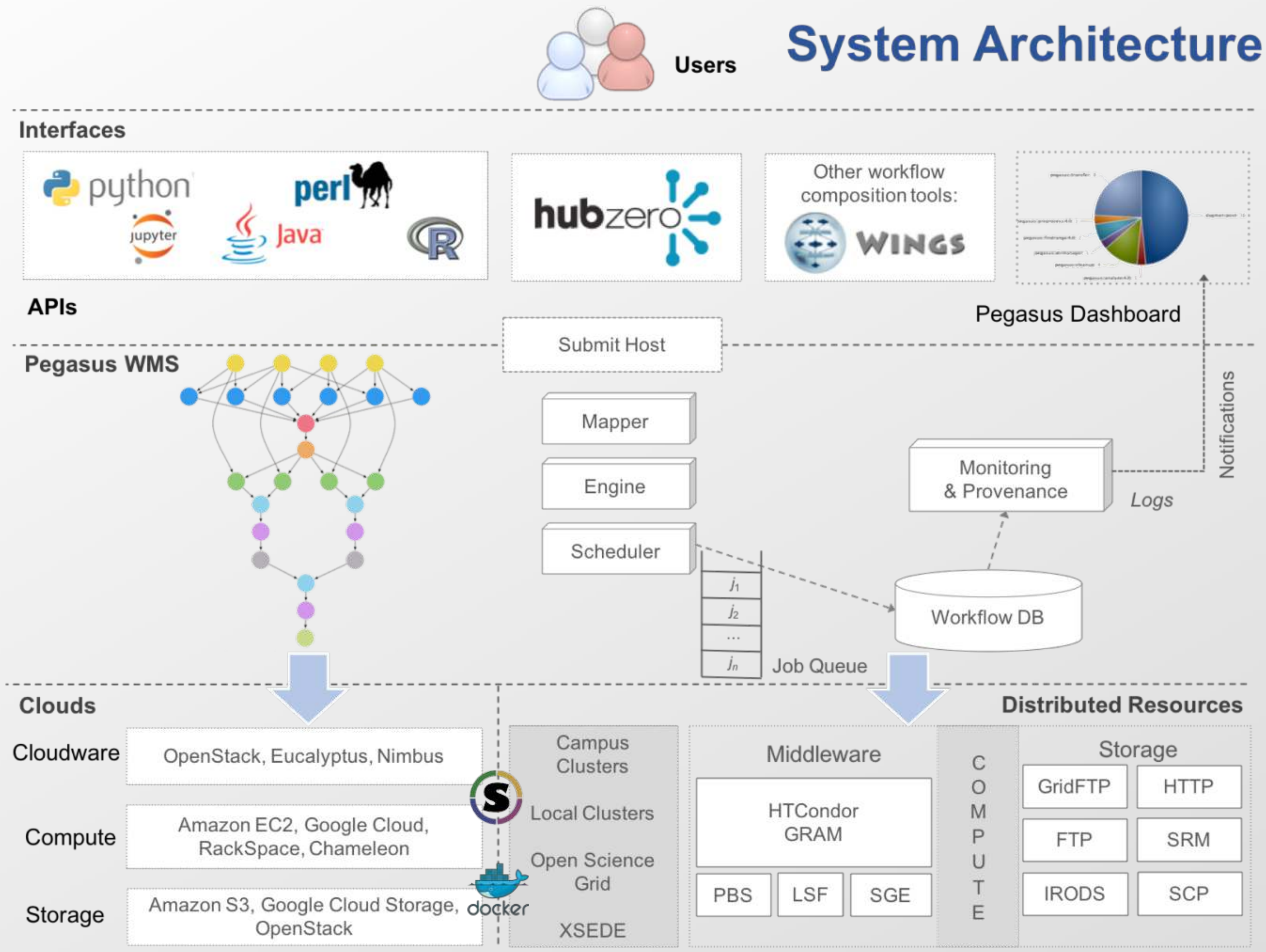


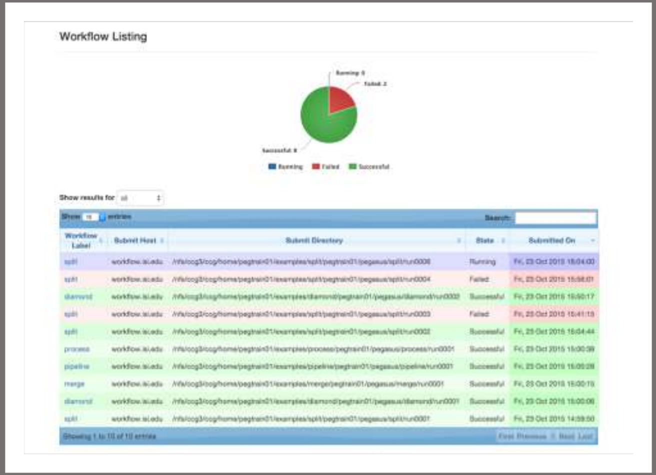
Visualization Tools:
pegasus-graphviz
pegasus-plots

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 parmeter 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()
```

System Architecture





Workflow Listing

Show results for all

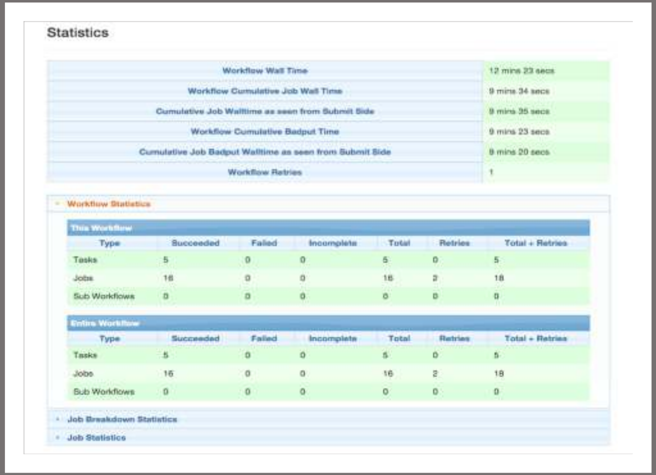
Show 10 entries Search:

Workflow Label	Submit Host	Submit Directory	State	Submitted On
split	workflow.isi.edu	/nfs/ccg3/ccg/home/pegtrain01/examples/split/pegtrain01/pegasus/split/run0006	Running	Fri, 23 Oct 2015 16:04:00
split	workflow.isi.edu	/nfs/ccg3/ccg/home/pegtrain01/examples/split/pegtrain01/pegasus/split/run0004	Failed	Fri, 23 Oct 2015 15:56:01
diamond	workflow.isi.edu	/nfs/ccg3/ccg/home/pegtrain01/examples/diamond/pegtrain01/pegasus/diamond/run0002	Successful	Fri, 23 Oct 2015 15:50:17
split	workflow.isi.edu	/nfs/ccg3/ccg/home/pegtrain01/examples/split/pegtrain01/pegasus/split/run0003	Failed	Fri, 23 Oct 2015 15:41:15
split	workflow.isi.edu	/nfs/ccg3/ccg/home/pegtrain01/examples/split/pegtrain01/pegasus/split/run0002	Successful	Fri, 23 Oct 2015 15:04:44
process	workflow.isi.edu	/nfs/ccg3/ccg/home/pegtrain01/examples/process/pegtrain01/pegasus/process/run0001	Successful	Fri, 23 Oct 2015 15:00:39
pipeline	workflow.isi.edu	/nfs/ccg3/ccg/home/pegtrain01/examples/pipeline/pegtrain01/pegasus/pipeline/run0001	Successful	Fri, 23 Oct 2015 15:00:28
merge	workflow.isi.edu	/nfs/ccg3/ccg/home/pegtrain01/examples/merge/pegtrain01/pegasus/merge/run0001	Successful	Fri, 23 Oct 2015 15:00:15
diamond	workflow.isi.edu	/nfs/ccg3/ccg/home/pegtrain01/examples/diamond/pegtrain01/pegasus/diamond/run0001	Successful	Fri, 23 Oct 2015 15:00:06
split	workflow.isi.edu	/nfs/ccg3/ccg/home/pegtrain01/examples/split/pegtrain01/pegasus/split/run0001	Successful	Fri, 23 Oct 2015 14:59:50

Showing 1 to 10 of 10 entries First Previous 1 Next Last

Pegasus dashboard

web interface for monitoring and debugging workflows



Statistics

Workflow Wall Time	12 mins 23 secs
Workflow Cumulative Job Wall Time	9 mins 34 secs
Cumulative Job Walltime as seen from Submit Side	9 mins 35 secs
Workflow Cumulative Badput Time	9 mins 23 secs
Cumulative Job Badput Walltime as seen from Submit Side	9 mins 20 secs
Workflow Retries	1

Workflow Statistics

This Workflow

Type	Succeeded	Failed	Incomplete	Total	Retries	Total + Retries
Tasks	5	0	0	5	0	5
Jobs	16	0	0	16	2	18
Sub Workflows	0	0	0	0	0	0

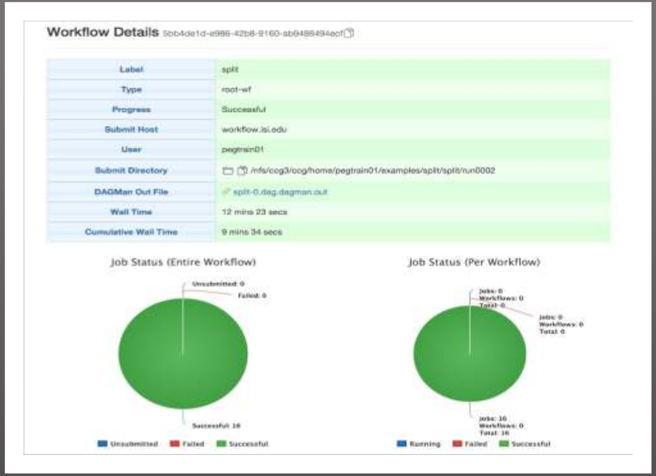
Entire Workflow

Type	Succeeded	Failed	Incomplete	Total	Retries	Total + Retries
Tasks	5	0	0	5	0	5
Jobs	16	0	0	16	2	18
Sub Workflows	0	0	0	0	0	0

Job Breakdown Statistics

Job Statistics

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



Workflow Details 5bb4de1d-e986-42b8-9160-ab948494acf

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

Real-time Monitoring

Reporting

Debugging

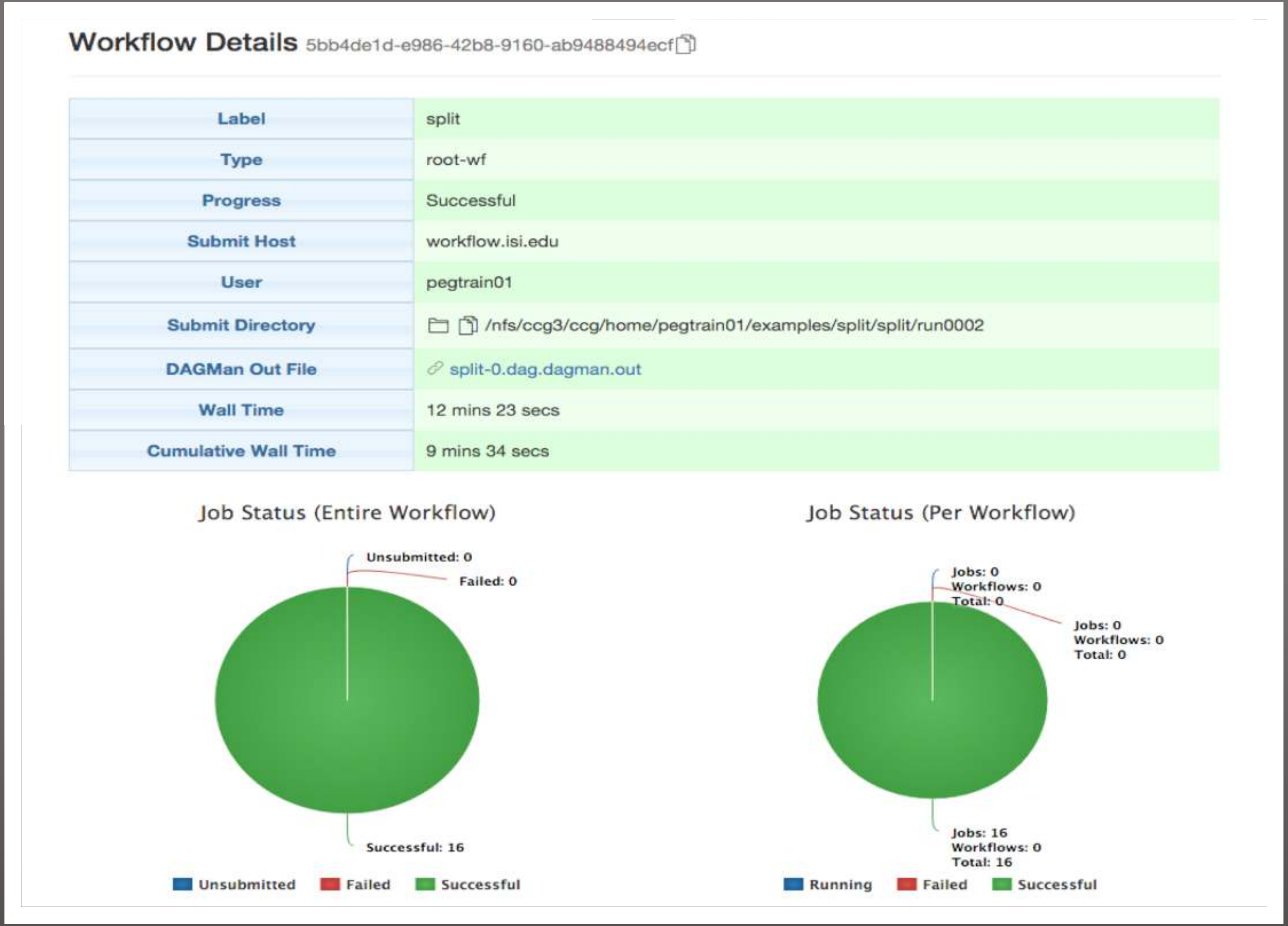
Troubleshooting

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

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

Hands-on Pegasus Tutorial…

Hands On Tutorial

- SSH to our training machine
  - Login with your user's tutorial login and password
  - `ssh trainXX@learn.chtc.wisc.edu`

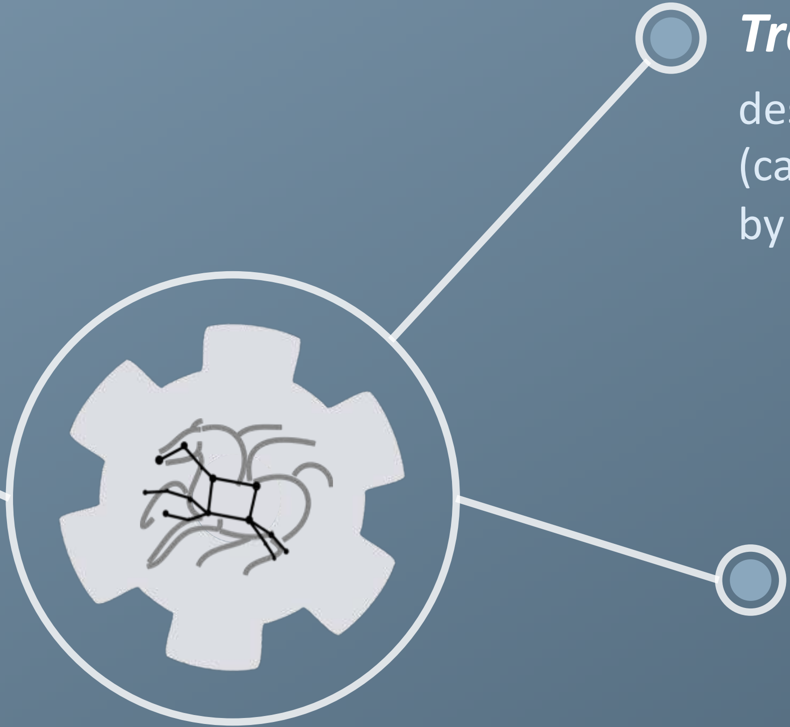
- Open exercise notes in your browser
  - https://pegasus.isi.edu/tutorial/chtc/tutorial.php

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	*Workflows with Containers* *Clustering* *Fault-Tolerance*
Other Features	*Data Staging* *Jupyter Notebooks* *Metadata, Hierarchal Workflows, Data Reuse*

Understanding Pegasus features…

So, what information does Pegasus need?



Transformation Catalog

describes all of the executables (called “transformations”) used by the workflow

Site Catalog

describes the sites where the workflow jobs are to be executed

Replica Catalog

describes all of the input data stored on external servers

How does Pegasus decide where to execute?

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 were 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?

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?

```
# 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

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)

Data Management for Containers

- Users can refer to container images as
  - Docker or Singularity Hub URL’s
  - Docker Image exported as a TAR file and available at a server , just like any other input dataset.

- We want to avoid hitting Docker/Singularity Hub repeatedly for large workflows
  - Extend pegasus-transfer to pull image from Docker Hub and then export it as tar file, that can be shipped around in the workflow.

- Ensure pegasus worker package gets installed at runtime inside the user container.

OUTLINE

Hands-on Pegasus Tutorial…

OUTLINE

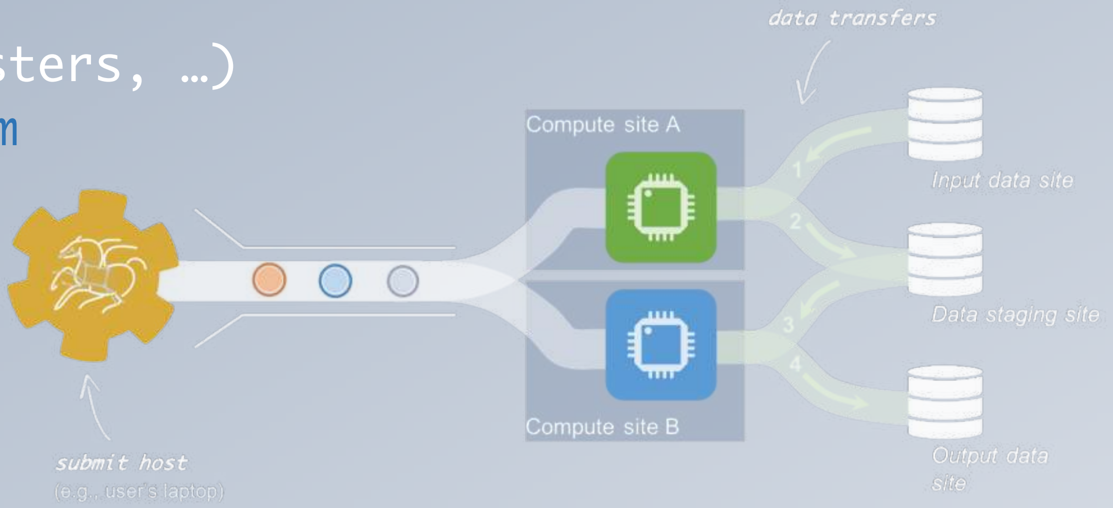
A few more features…

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

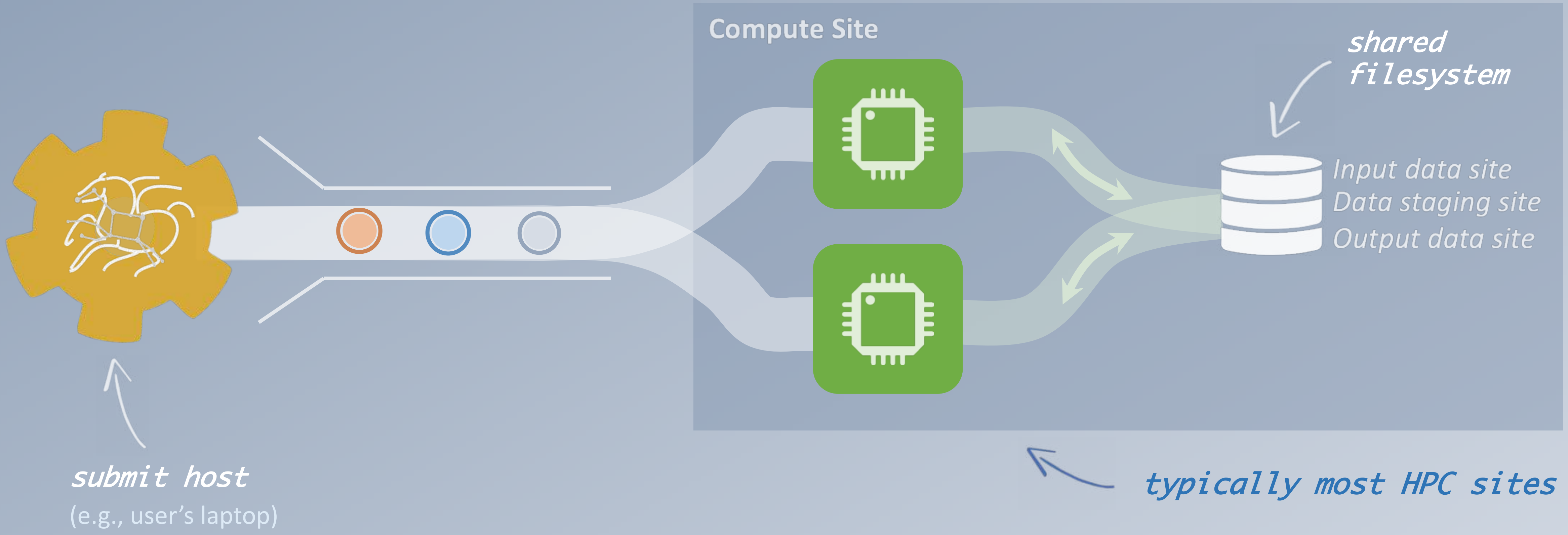
Non-shared File System (clouds, OSG, …)
- Worker nodes do not share a file system
- Data is pulled / pushed from a staging site, possibly not co-located with the computation

Shared File System (HPC sites, XSEDE, Campus clusters, …)
- I/O is directly against the shared file system



High Performance Computing

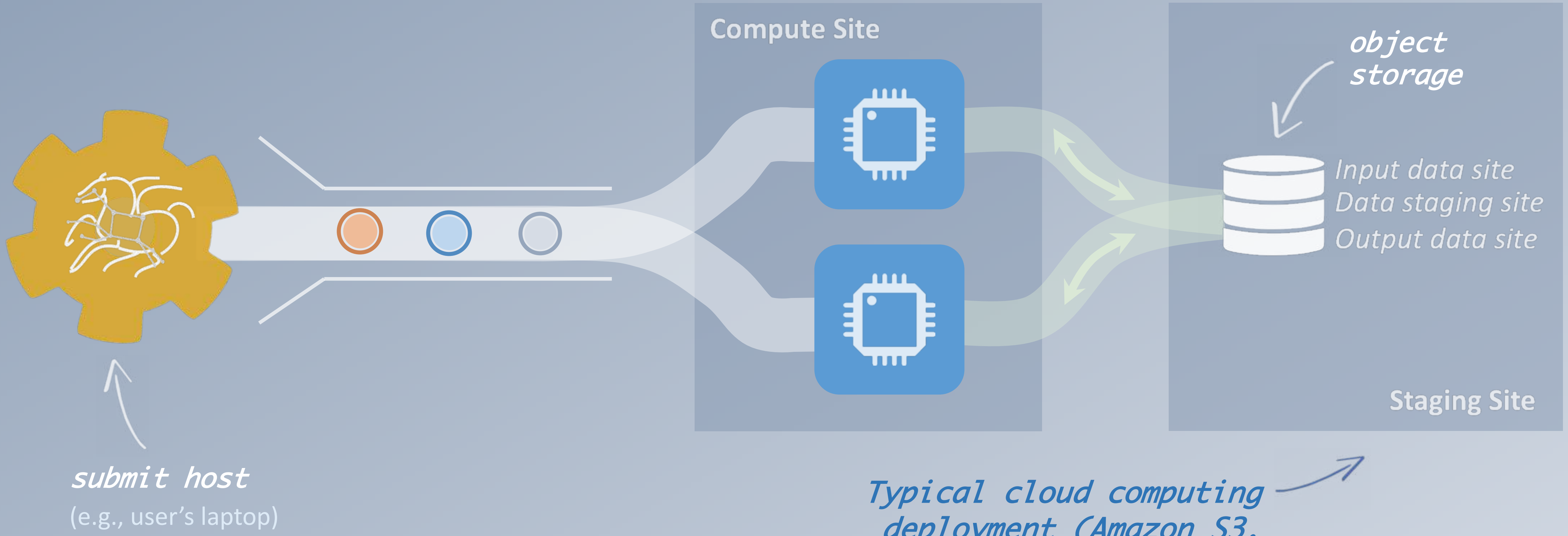
There are several possible configurations…



typically most HPC sites

Cloud Computing

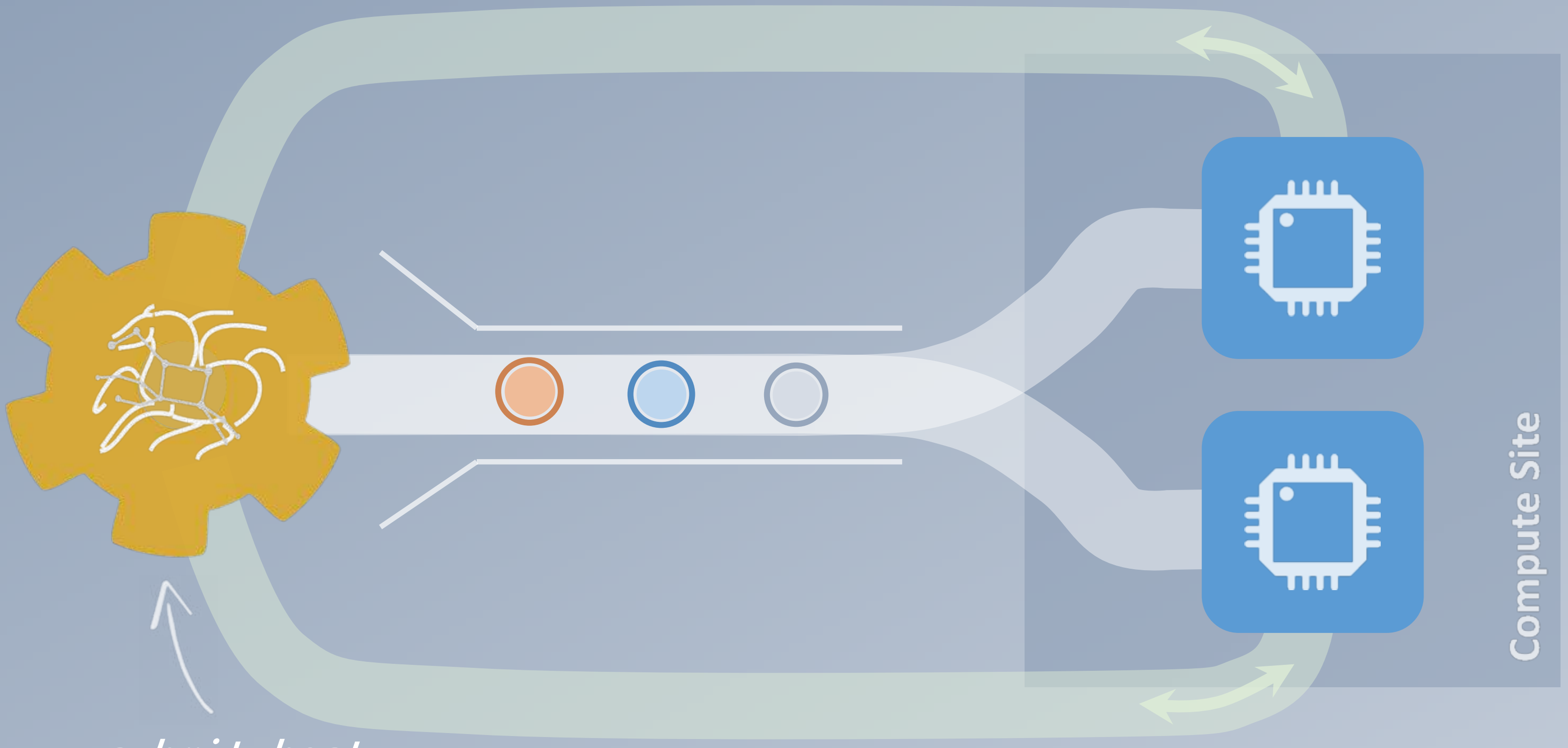
high-scalable object storages



Typical cloud computing deployment (Amazon S3, Google Storage)

Grid Computing

local data management



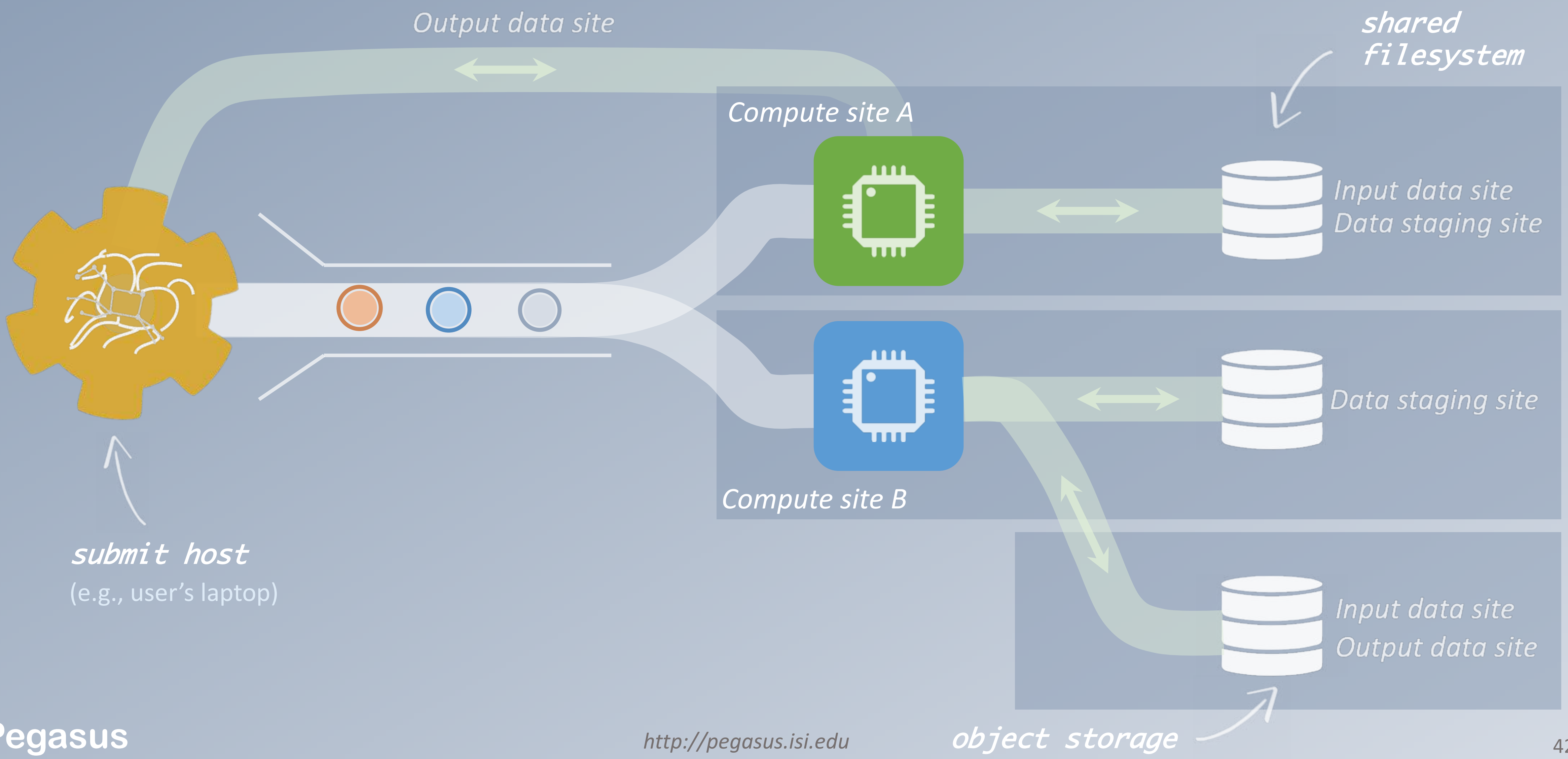
Typical OSG sites

Open Science Grid

submit host

(e.g., user's laptop)

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

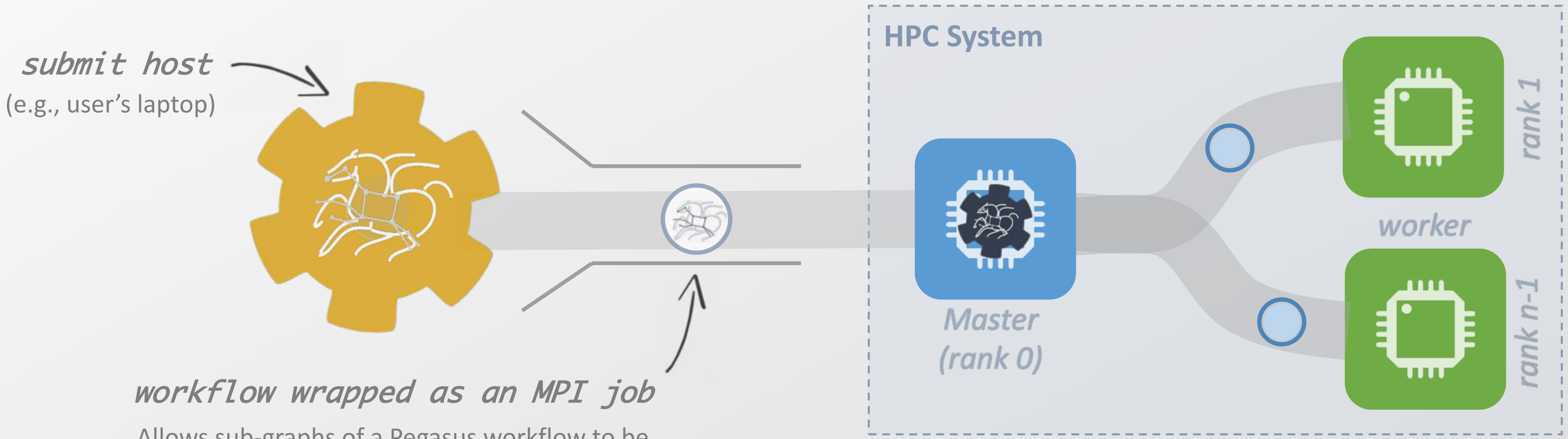
Automatic retries

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

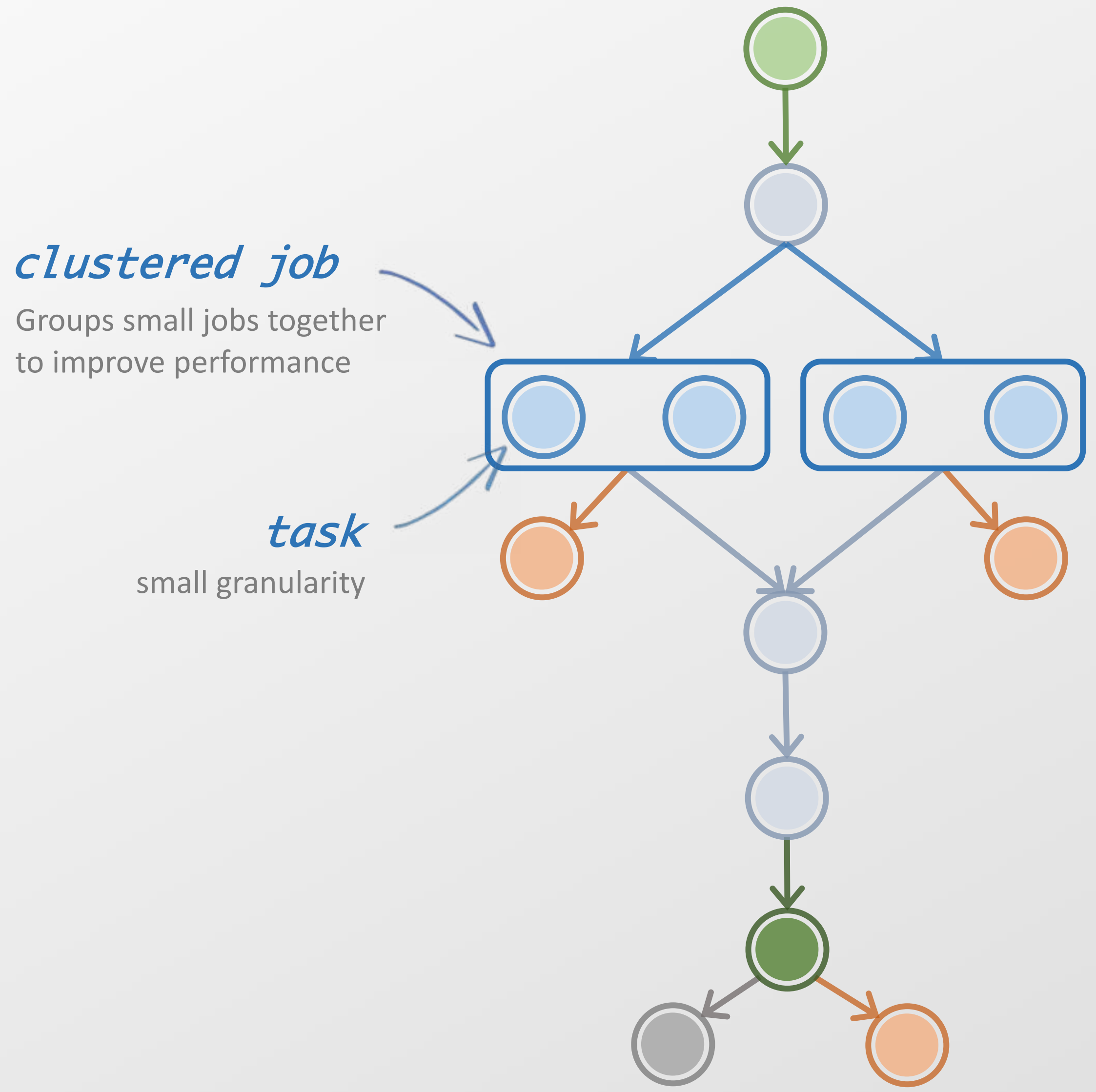
Running fine-grained workflows on HPC systems…



workflow 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?



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

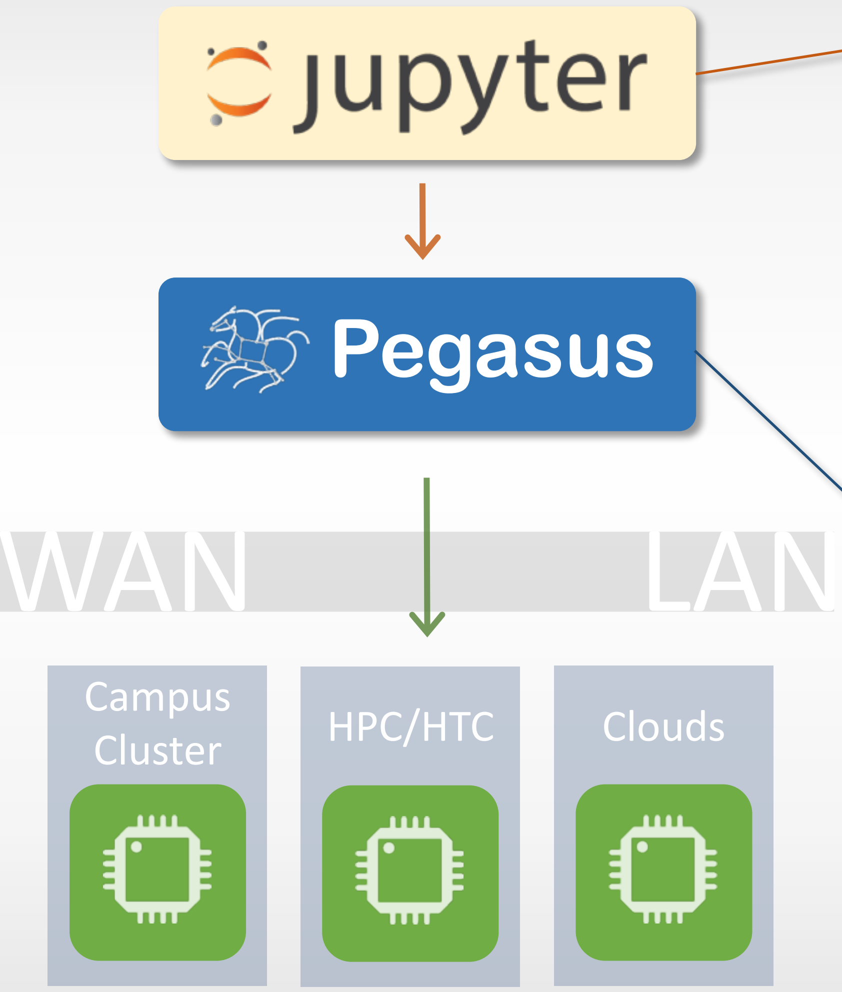
Checkpoint Files

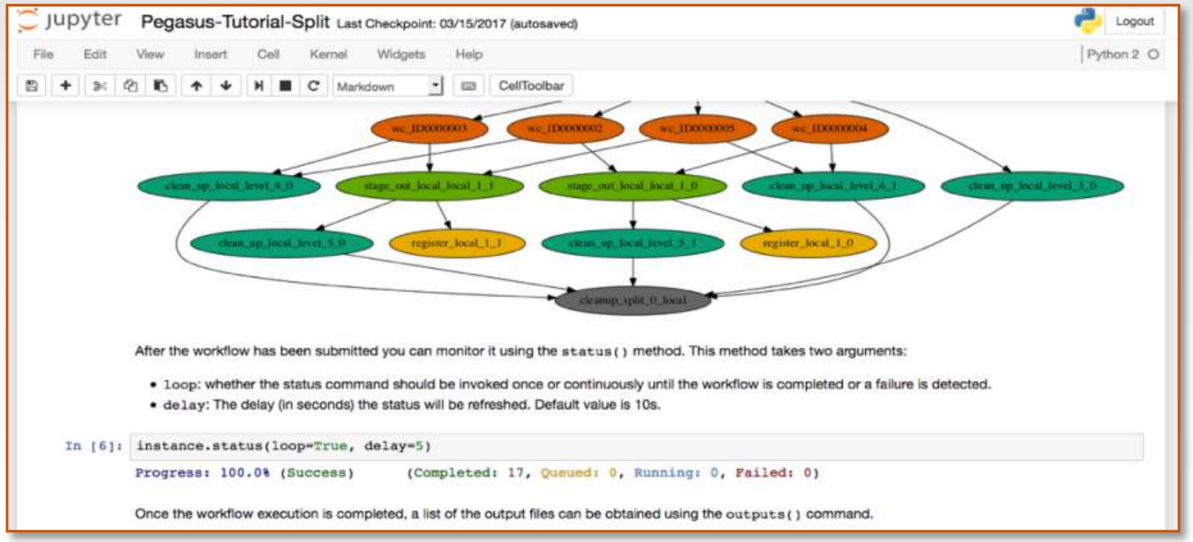
job generates checkpoint files
staging of checkpoint files is
automatic on restarts

Rescue DAGs

workflow can be restarted from checkpoint file
recover from failures with minimal loss

Running Pegasus workflows with Jupyter



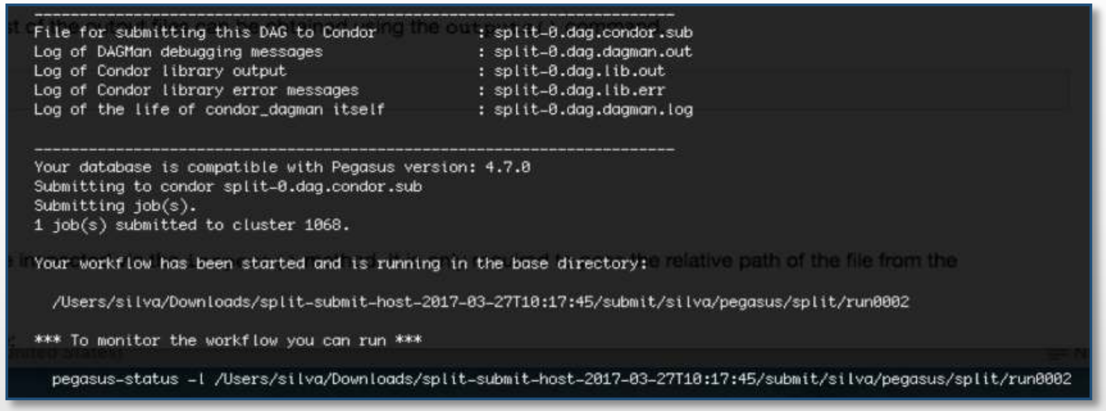

jupyter Pegasus-Tutorial-Split Last Checkpoint: 03/15/2017 (autosaved)
Logout
File Edit View Insert Cell Kernel Widgets Help
Python 2
Markdown
CellToolbar

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:

  /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

```
# 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 = Instance(dax)
```

creating an instance of the DAX

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

running 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

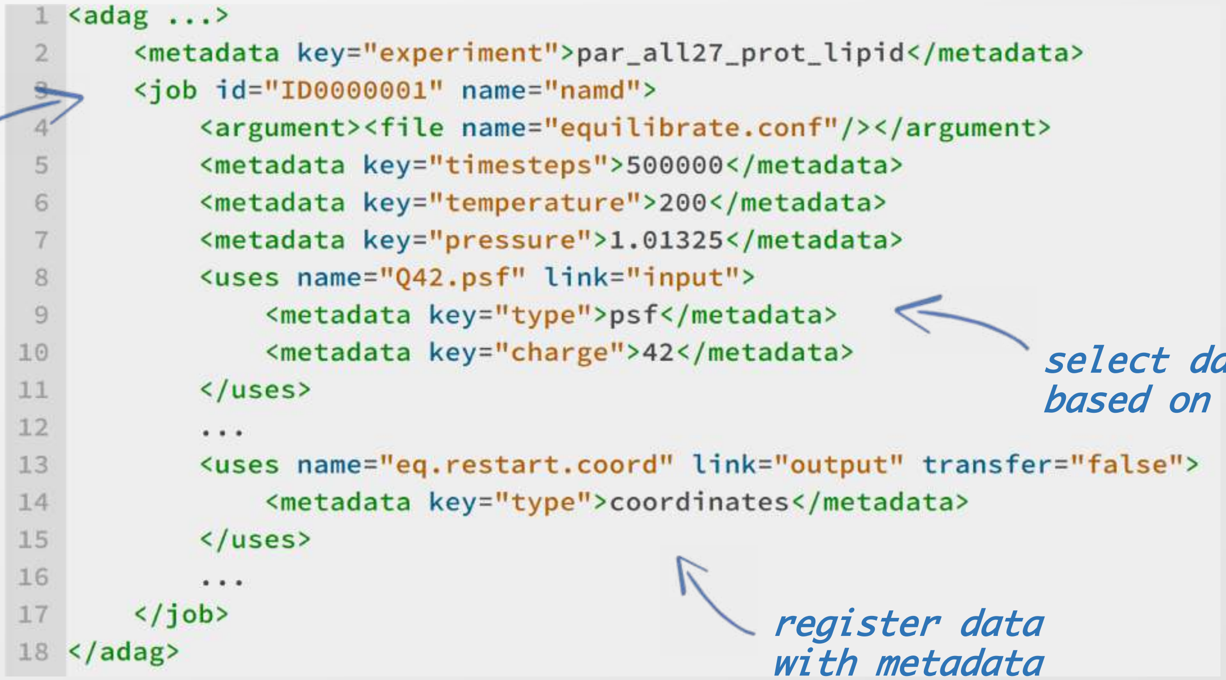
Output files get tagged with metadata on registration in the workflow database

Static and runtime metadata

Static: application parameters

Runtime: performance metrics

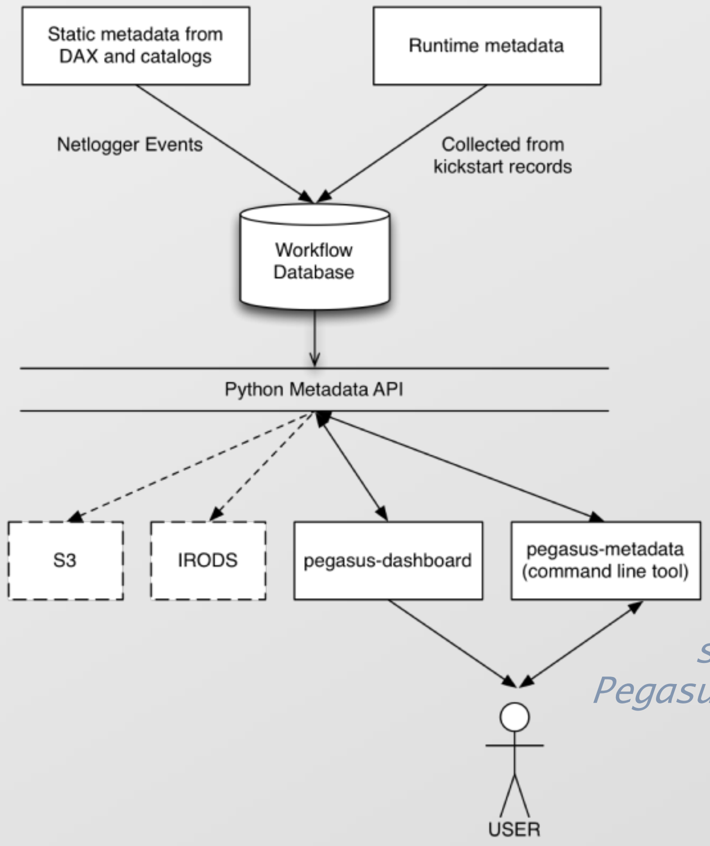
workflow, job, file



```
<adag ...>
    <metadata key="experiment">par_all27_prot_lipid</metadata>
    <job id="ID0000001" name="namd">
        <argument><file name="equilibrate.conf"/></argument>
        <metadata key="timesteps">500000</metadata>
        <metadata key="temperature">200</metadata>
        <metadata key="pressure">1.01325</metadata>
        <uses name="Q42.psf" link="input">
            <metadata key="type">psf</metadata>
            <metadata key="charge">42</metadata>
        </uses>
        ...
        <uses name="eq.restart.coord" link="output" transfer="false">
            <metadata key="type">coordinates</metadata>
        </uses>
        ...
    </job>
</adag>
```

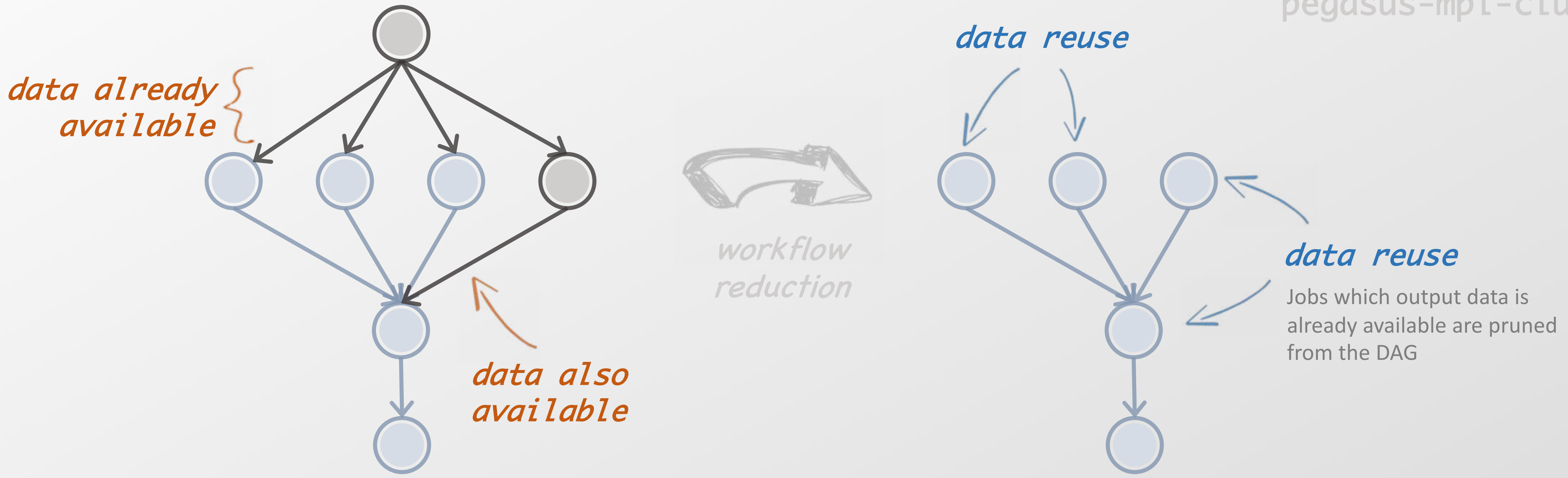
select data based on metadata

register data with metadata


Static metadata from DAX and catalogs
Runtime metadata
Netlogger Events
Collected from kickstart records
Workflow Database
Python Metadata API
S3
IRODS
pegasus-dashboard
pegasus-metadata (command line tool)
USER

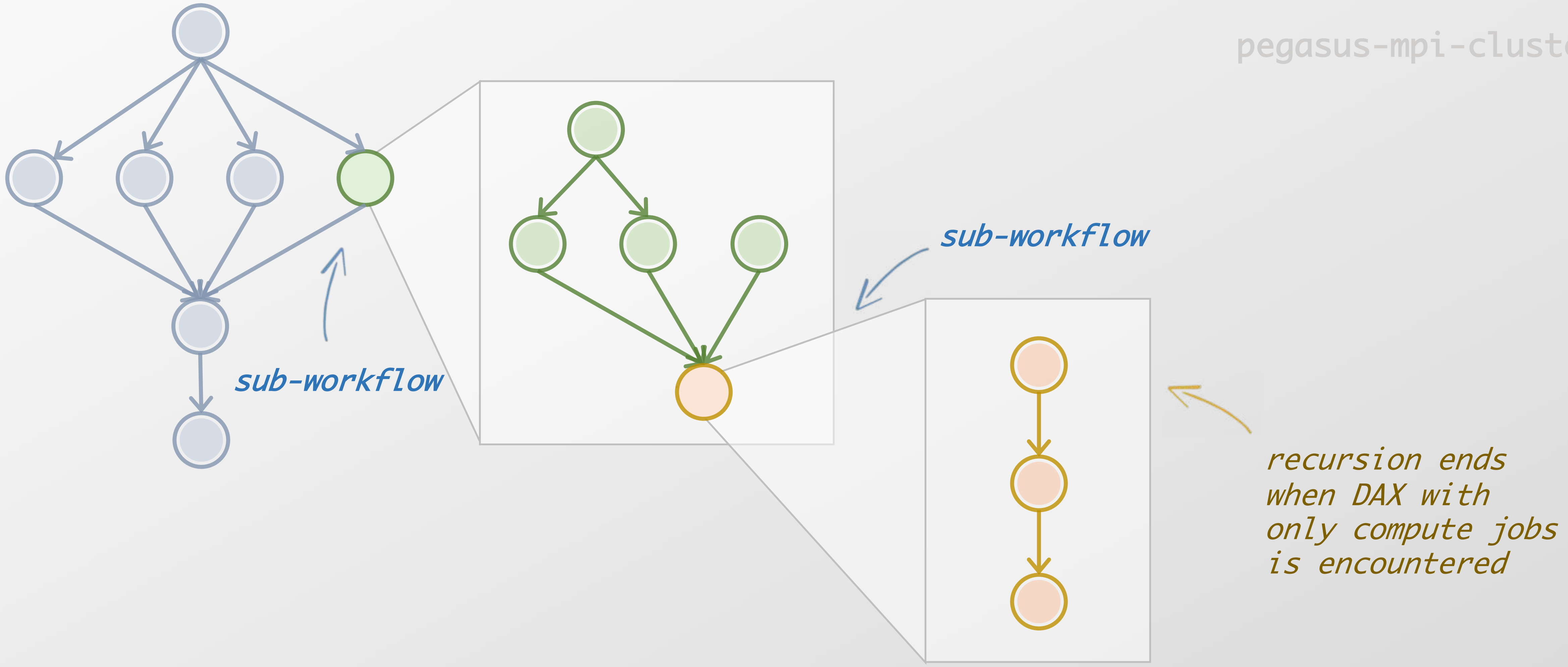
since Pegasus 4.6

What about data reuse?



Jobs which output data is already available are pruned from the DAG

Pegasus also handles **large-scale workflows**


sub-workflow
sub-workflow
recursion ends when DAX with only compute jobs is encountered

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