Pegasus

Automate, recover, and debug scientific computations.

Mats Rynge
rynge@isi.edu

USC Viterbi
School of Engineering
Information Sciences Institute

https://pegasus.isi.edu

Why Pegasus?

Automates complex, multi-stage processing pipelines

Enables parallel, distributed computations

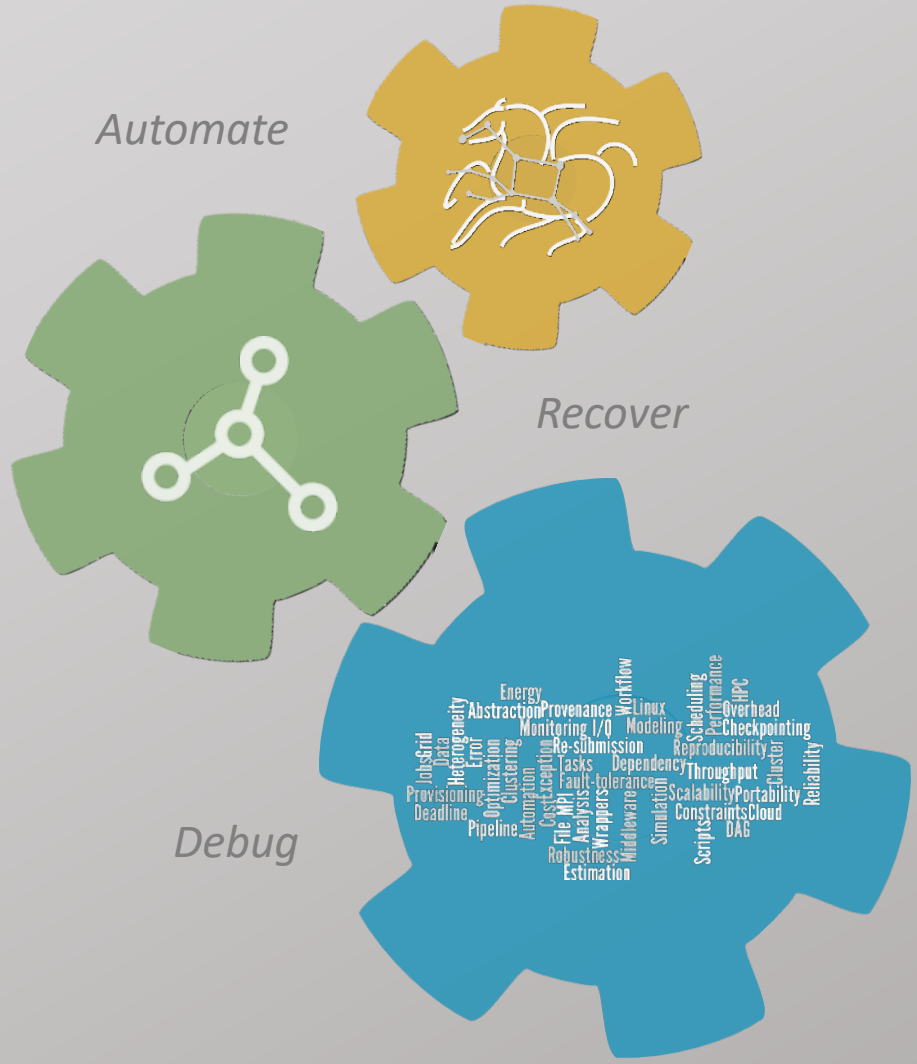
Automatically executes data transfers

Reusable, aids reproducibility

Records how data was produced (provenance)

Handles failures with to provide reliability

Keeps track of data and files

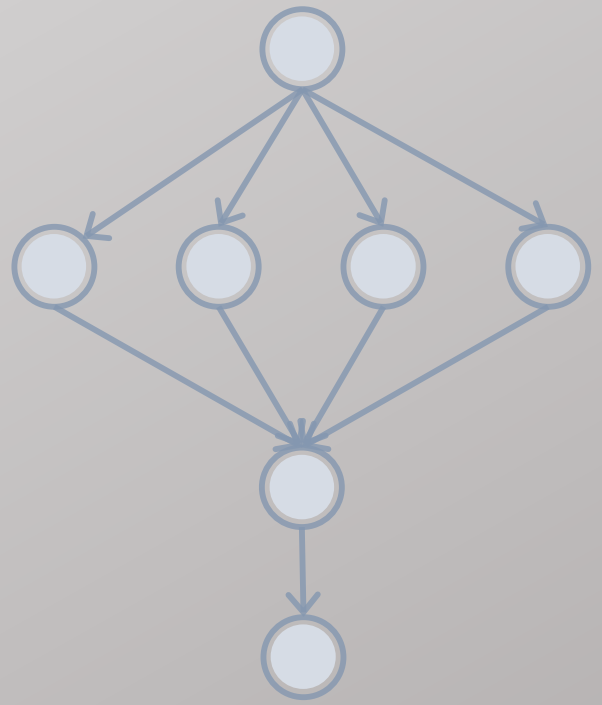


HTCondor
High Throughput Computing

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 (or hierarchical DAGs)

- Nodes: jobs, edges: dependencies
- No while loops, no conditional branches

Planning occurs ahead of execution

- (Except hierarchical workflows)

Planning converts an abstract workflow into a concrete, executable workflow

- Planner is like a compiler

Taking a closer look into a workflow…

abstract workflow

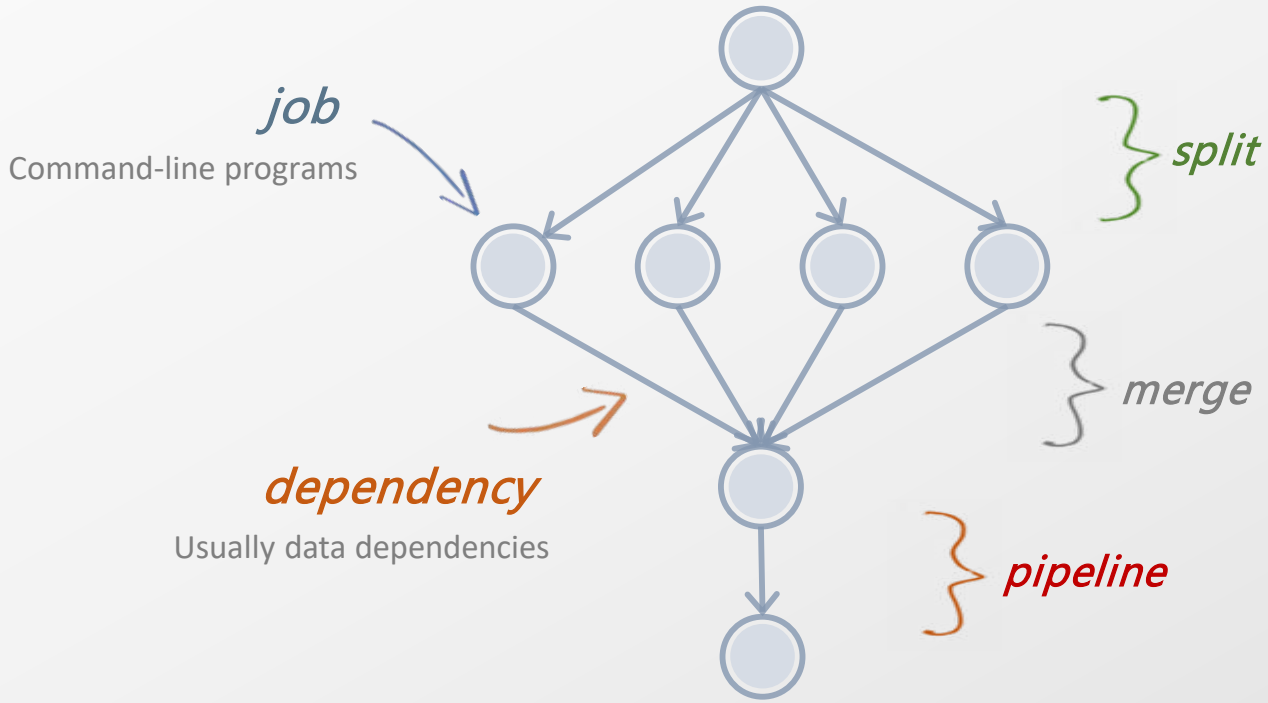
executable workflow

optimizations

storage constraints

DAG

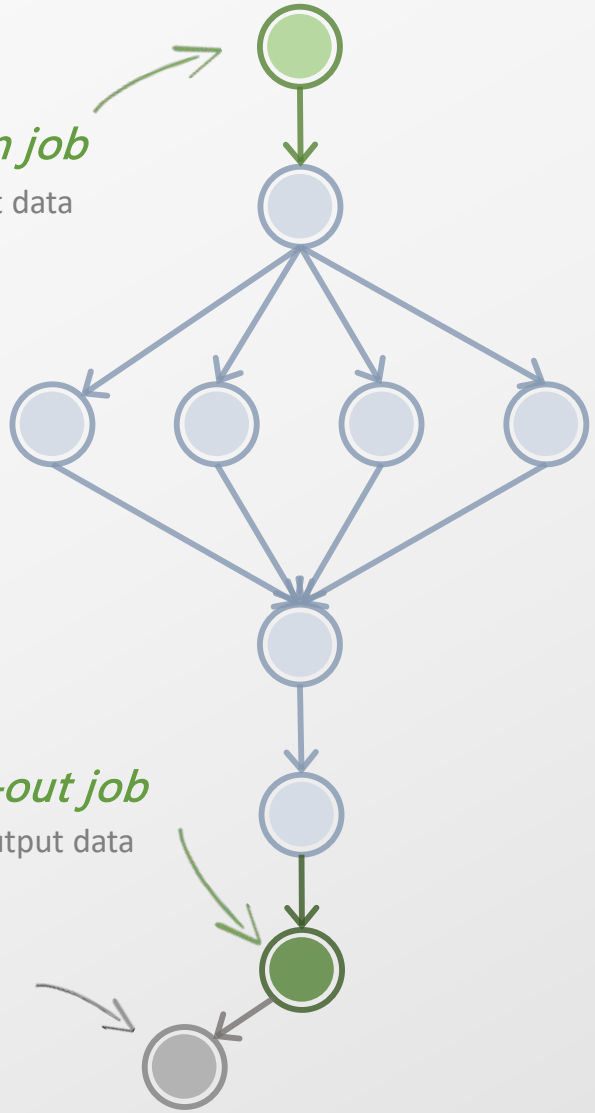
directed-acyclic graphs



DAX

DAG in XML

From the abstraction to execution!



stage-in job

Transfers the workflow input data

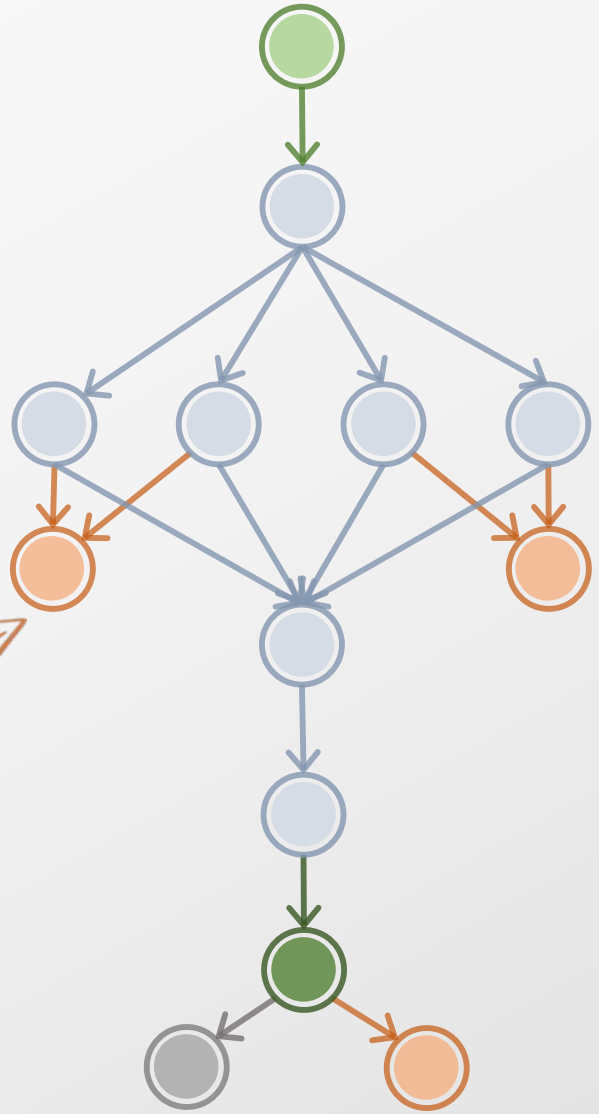
stage-out job

Transfers the workflow output data

registration job

Registers the workflow output data

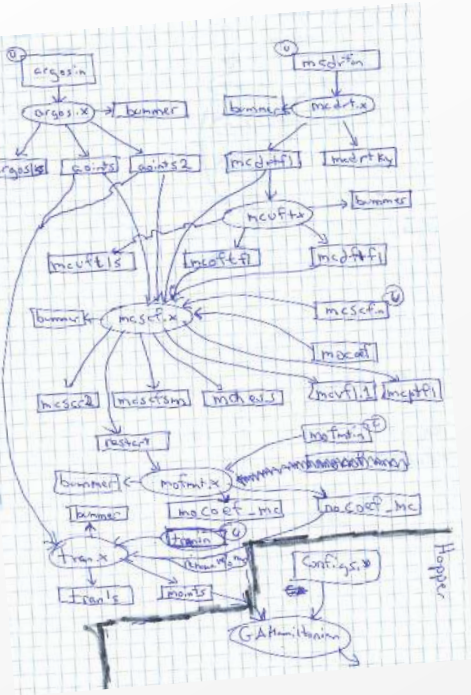
Optimizing storage usage…



cleanup job

Removes unused data

Pegasus also provides tools to generate the abstract workflow



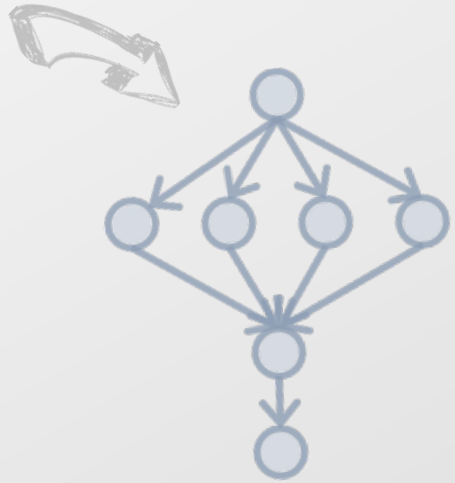
python

Java

perl

```
#!/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)
```

```
<?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

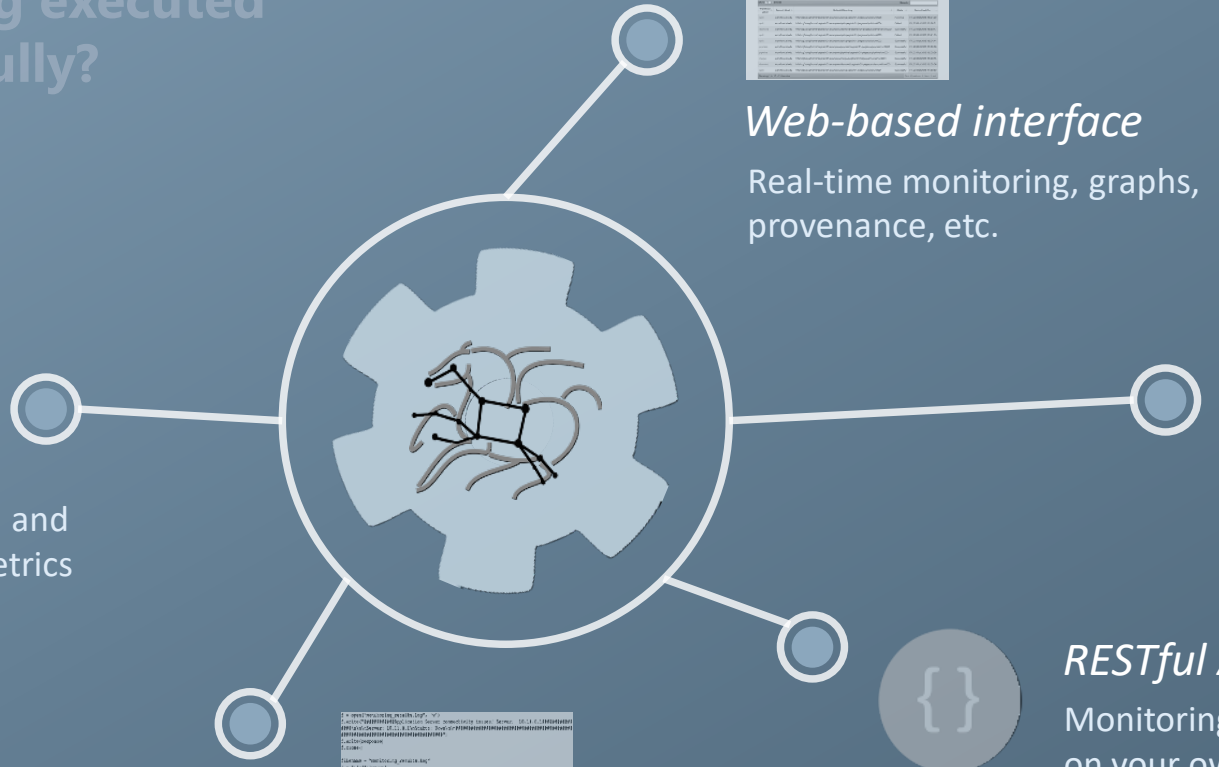
While you wait...

...or the execution is finished.

Does everything executed successfully?

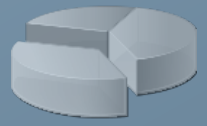
How my workflow behaves?

Past executions?



Web-based interface

Real-time monitoring, graphs, provenance, etc.



Statistics

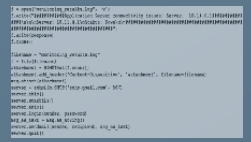
Workflow execution and job performance metrics

Debug

Set of debugging tools to unveil issues

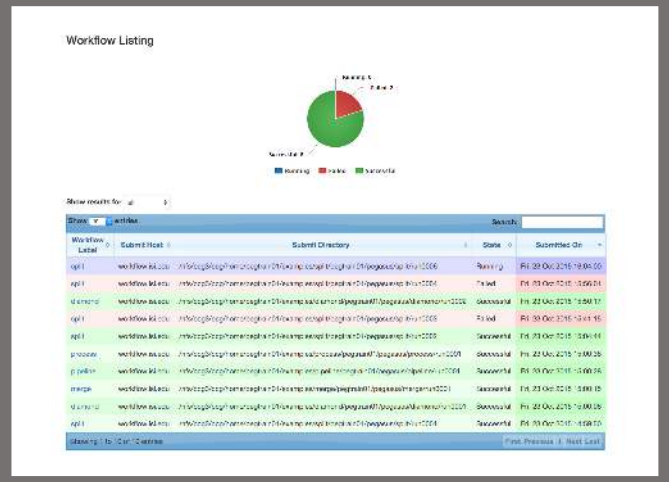
RESTful API

Monitoring and reporting information on your own application interface



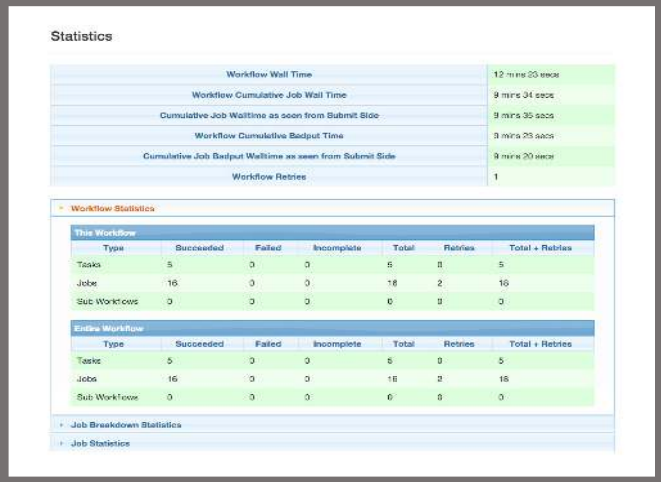
Command-line tools

Tools for monitor and debug workflows



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

But, if you prefer the 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%)
```

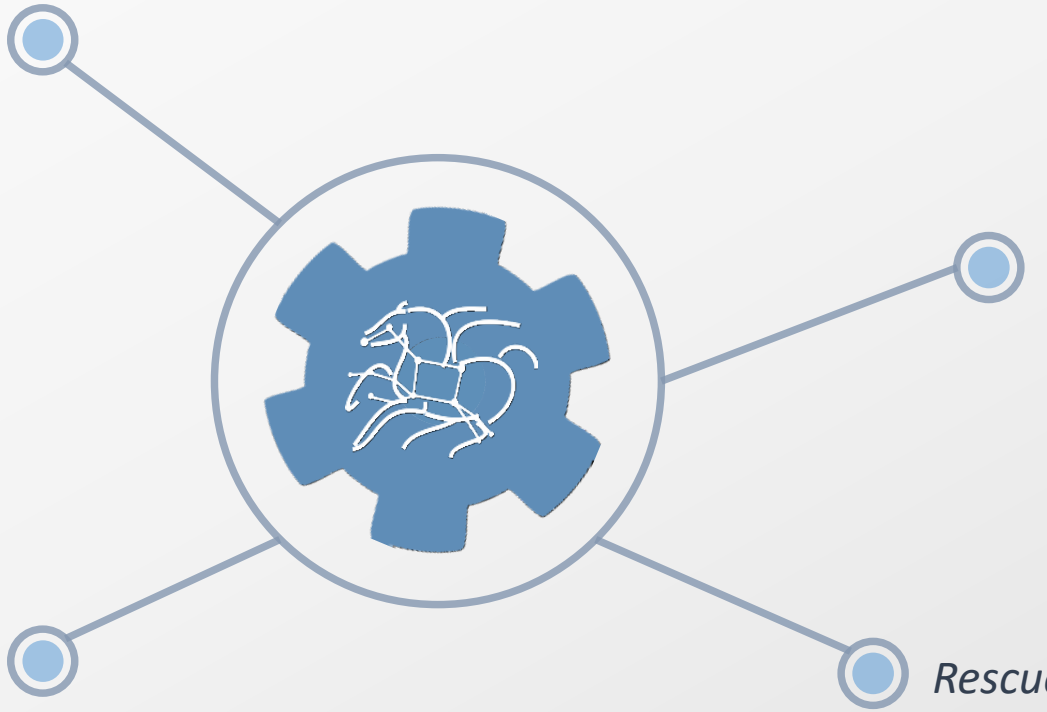
```
$ pegasus-statistics -s all pegasus/examples/split/run0001
------------------------------------------------------------------------------
Type           Succeeded Failed Incomplete Total Retries Total+Retries
Tasks              5       0         0        5     0         5
Jobs              17   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

…Pegasus provides a set of concise and powerful tools

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

pegasus-kickstart

() login02.osgconnect.net — Konsole

File Edit View Bookmarks Settings Help

```
<?xml version="1.0" encoding="UTF-8"?>
<invocation xmlns="http://pegasus.isi.edu/schema/invocation" xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance" xsi:schemaLocation="http://pegasus.isi.edu/schema/invocation http://pegasus.isi.edu/schema/iv-2.3.xsd" version="2.3" start="2016-11-28T14:27:48.909-06:00" duration="11200.691" transformation="job-wrapper.sh" derivation="ID0013214" resource="condorpool" wf-label="particleshower" wf-stamp="2016-11-22T21:14:13-06:00" interface="eth0" hostaddr="131.225.208.240" hostname="fnpc4593.fnal.gov" pid="1725084" uid="12740" user="osg" gid="9652" group="osg" umask="0022">
  <mainjob start="2016-11-28T14:27:49.007-06:00" duration="11200.593" pid="1725089">
    <usage utime="10921.591" stime="30.304" maxrss="395820" minflt="128741" majflt="18" nswap="0" inblock="85776" outblock="1717424" msgsnd="0" msgrcv="0" nsignals="0" nvcsw="7676" nivcsw="185495"/>
    <status raw="0"><regular exitcode="0"/></status>
    <statcall error="0">
      <file name="/storage/local/data1/condor/execute/dir_1227464/glide_bSxwfe/execute/dir_1724937/pegasus.XRZ1p3/job-wrapper.sh">23212F62696E2F626173680A0A736574</file>
      <statinfo mode="0100755" size="1305" inode="16648869" nlink="1" blksize="4096" blocks="8" mtime="2016-11-28T12:10:53-06:00" atime="2016-11-28T14:27:48-06:00" ctime="2016-11-28T14:27:48-06:00" uid="12740" user="osg" gid="9652" group="osg"/>
    </statcall>
    <argument-vector>
      <arg nr="1">100</arg>
      <arg nr="2">0</arg>
      <arg nr="3">gamma</arg>
      <arg nr="4">62</arg>
      <arg nr="5">VERITAS</arg>
      <arg nr="6">corsika.tar.gz</arg>
      <arg nr="7">corsika75000Linux_QGSII_urqmd</arg>
      <arg nr="8">13213</arg>
    </argument-vector>
  </mainjob>
  <jobids condor="547839.0"/>
  <cwd>/storage/local/data1/condor/execute/dir_1227464/glide_bSxwfe/execute/dir_1724937/pegasus.XRZ1p3</cwd>
  <usage utime="0.013" stime="0.085" maxrss="828" minflt="2448" majflt="0" nswap="0" inblock="0" outblock="0" msgsnd="0" msgrcv="0" nsignals="0" nvcsw="1" nivcsw="12"/>
  <machine page-size="4096">
    <stamp>2016-11-28T14:27:48.909-06:00</stamp>
    <uname system="linux" nodename="fnpc4593.fnal.gov" release="2.6.32-642.6.2.el6.x86_64" machine="x86_64">#1 SMP Tue Oct 25 15:06:33 CDT 2016</uname>
    <linux>
      <ram total="65319608" free="1071948" shared="0" buffer="148224"/>
      <swap total="8388604" free="7741364"/>
      <boot idle="45893257.760">2016-11-09T16:40:54.260-06:00</boot>
      <cpu count="32" speed="2000" vendor="AuthenticAMD">AMD Opteron(tm) Processor 6128</cpu>
      <load min1="26.35" min5="27.70" min15="24.33"/>
      <procs total="881" running="23" sleeping="854" waiting="3" zombie="1" vmsize="65009304" rss="14780272"/>
      <task total="1273" running="24" sleeping="1243" waiting="5" zombie="1"/>
    </linux>
```

5,5 Top

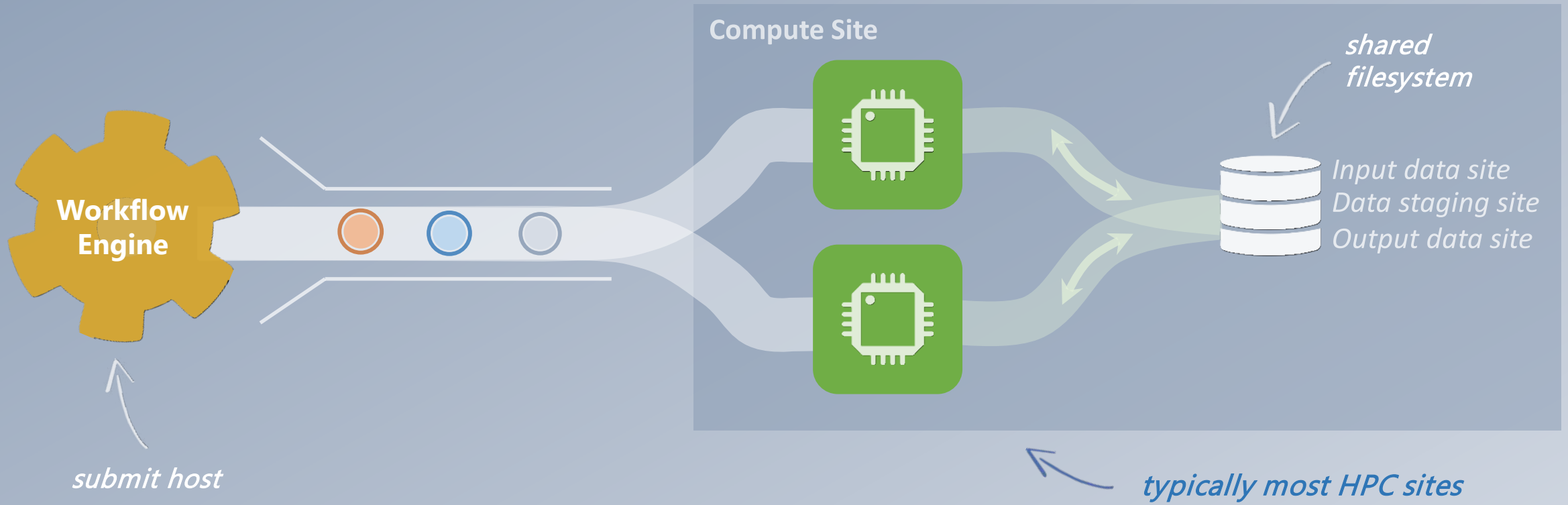
() login02.osgconnect.net

Data Staging Configurations

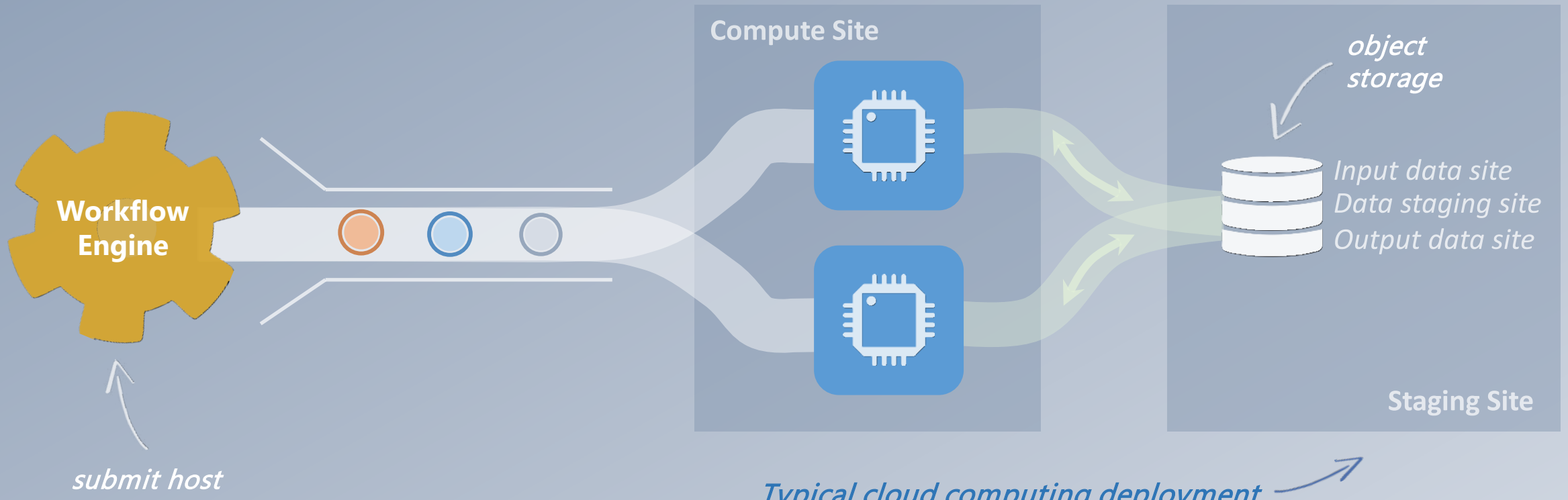
- Condor I/O (HTCondor pools, OSG, …)
  - Worker nodes do not share a file system
  - Data is pulled from / pushed to the submit host via HTCondor file transfers
  - Staging site is the submit host
- Non-shared File System (clouds, OSG, …)
  - Worker nodes do not share a file system
  - Data is pulled / 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...

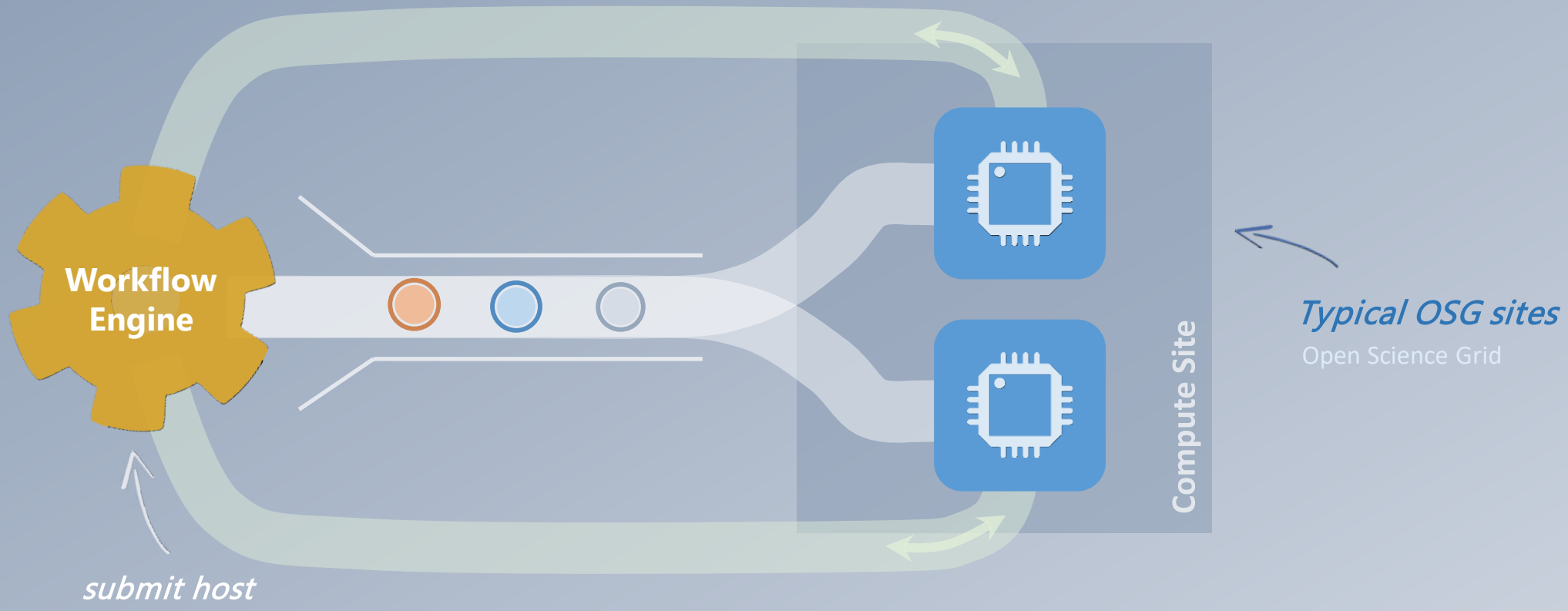


High-scalable object storages

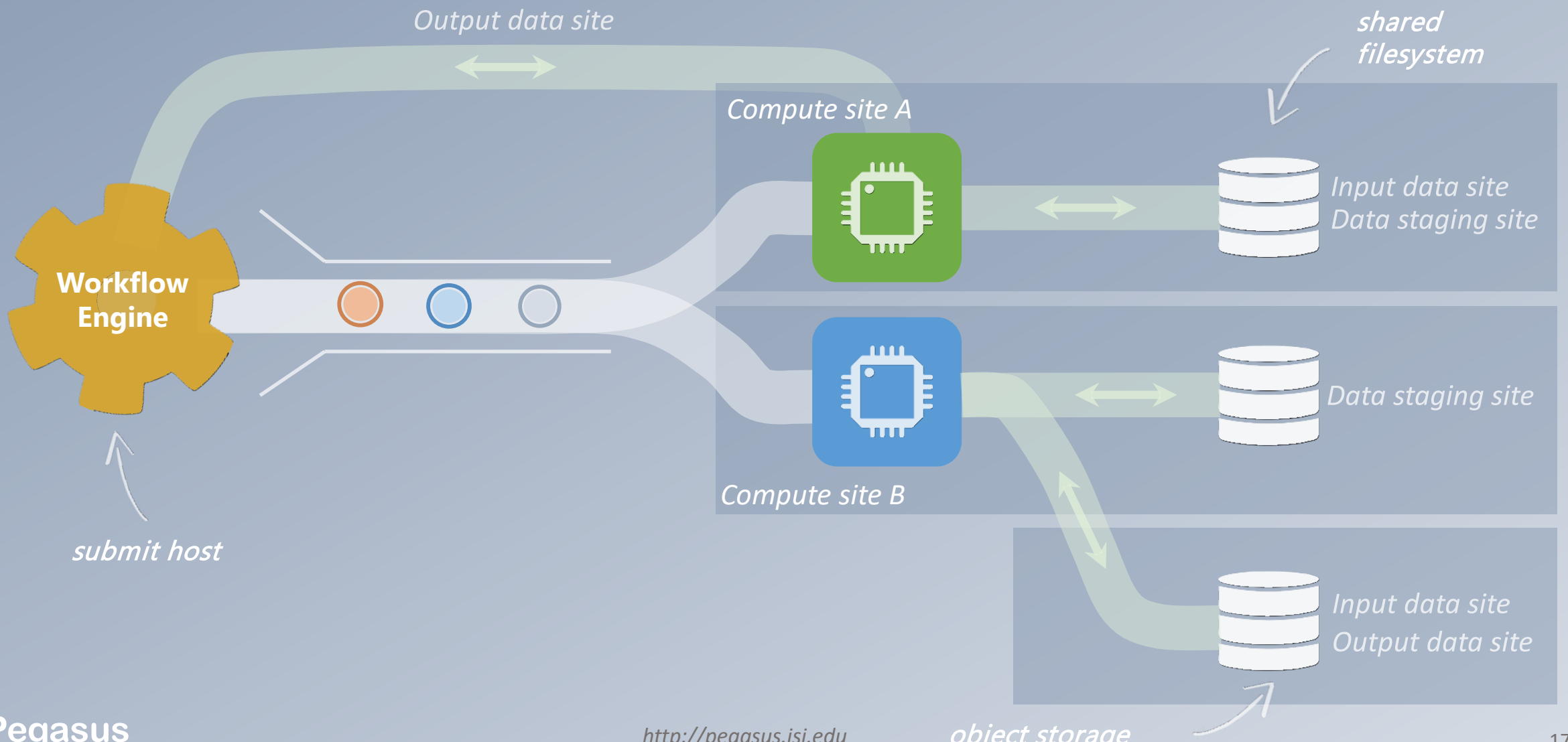


Typical cloud computing deployment (Amazon S3, Google Storage)

local data management



And yes… you can mix everything!



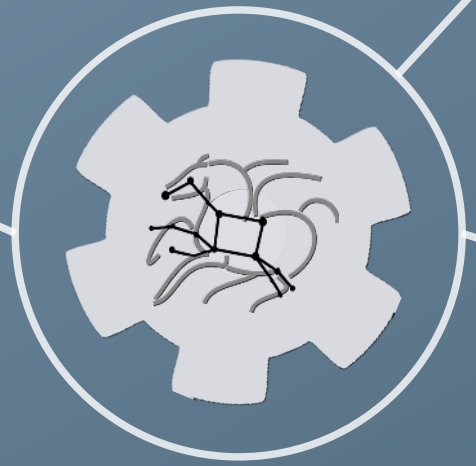
pegasus-transfer

- Pegasus' internal data transfer tool
- Supports many 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
- Checkpoint and restart transfers
- Credential management
  - Uses the appropriate credential for each site and each protocol (even 3rd party transfers)

Protocols

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

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 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 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   http://storage.mysite.edu/examples/diamond/input/f.a   site="storage"
```

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"
```

A few more features...

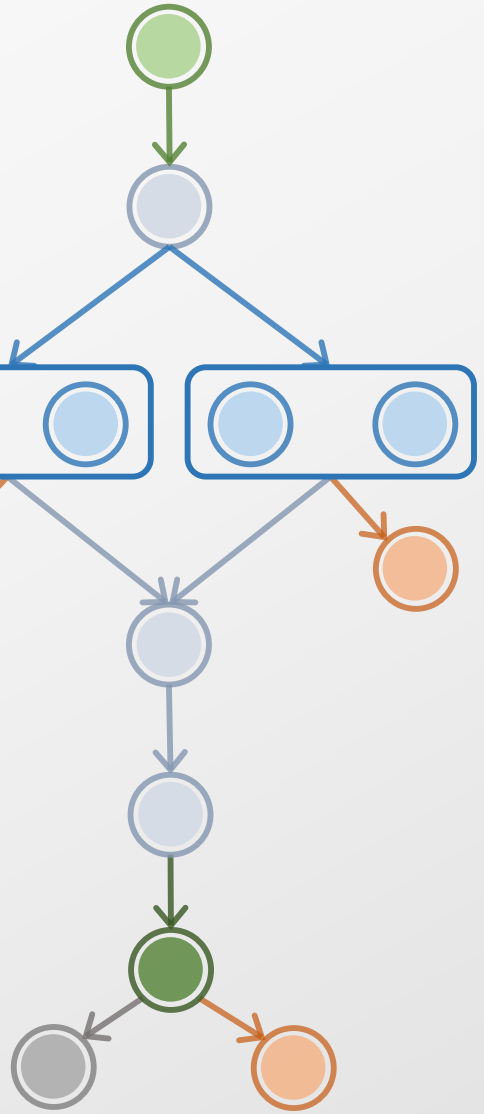
Performance, why not improve it?

clustered job

Groups small jobs together to improve performance

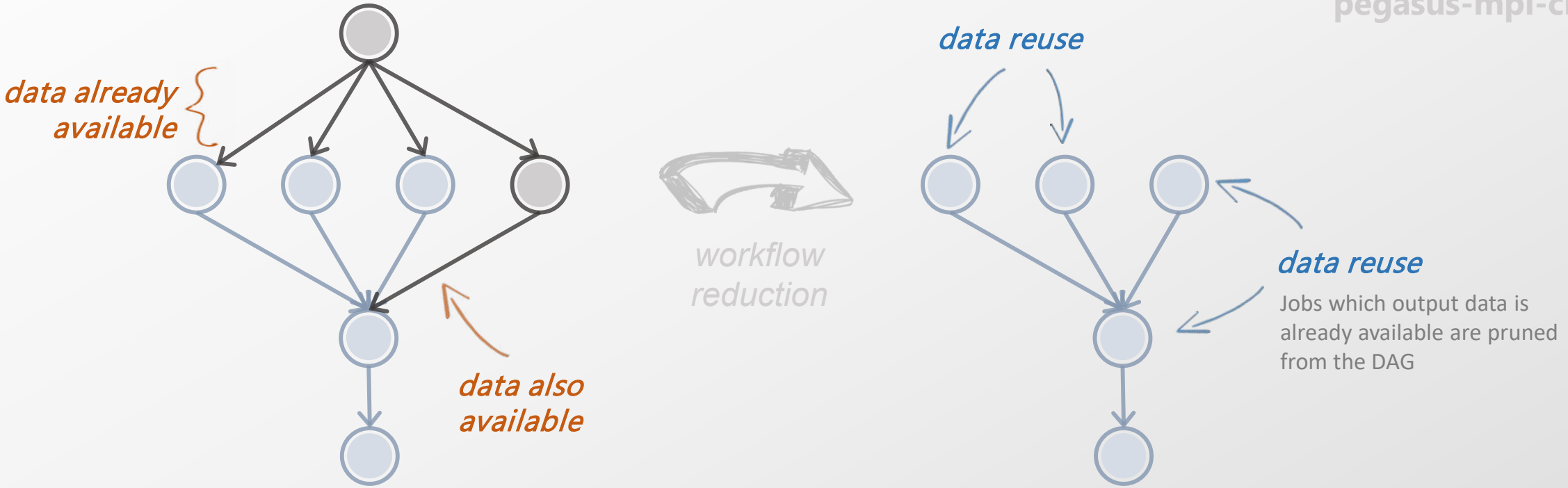
task

small granularity

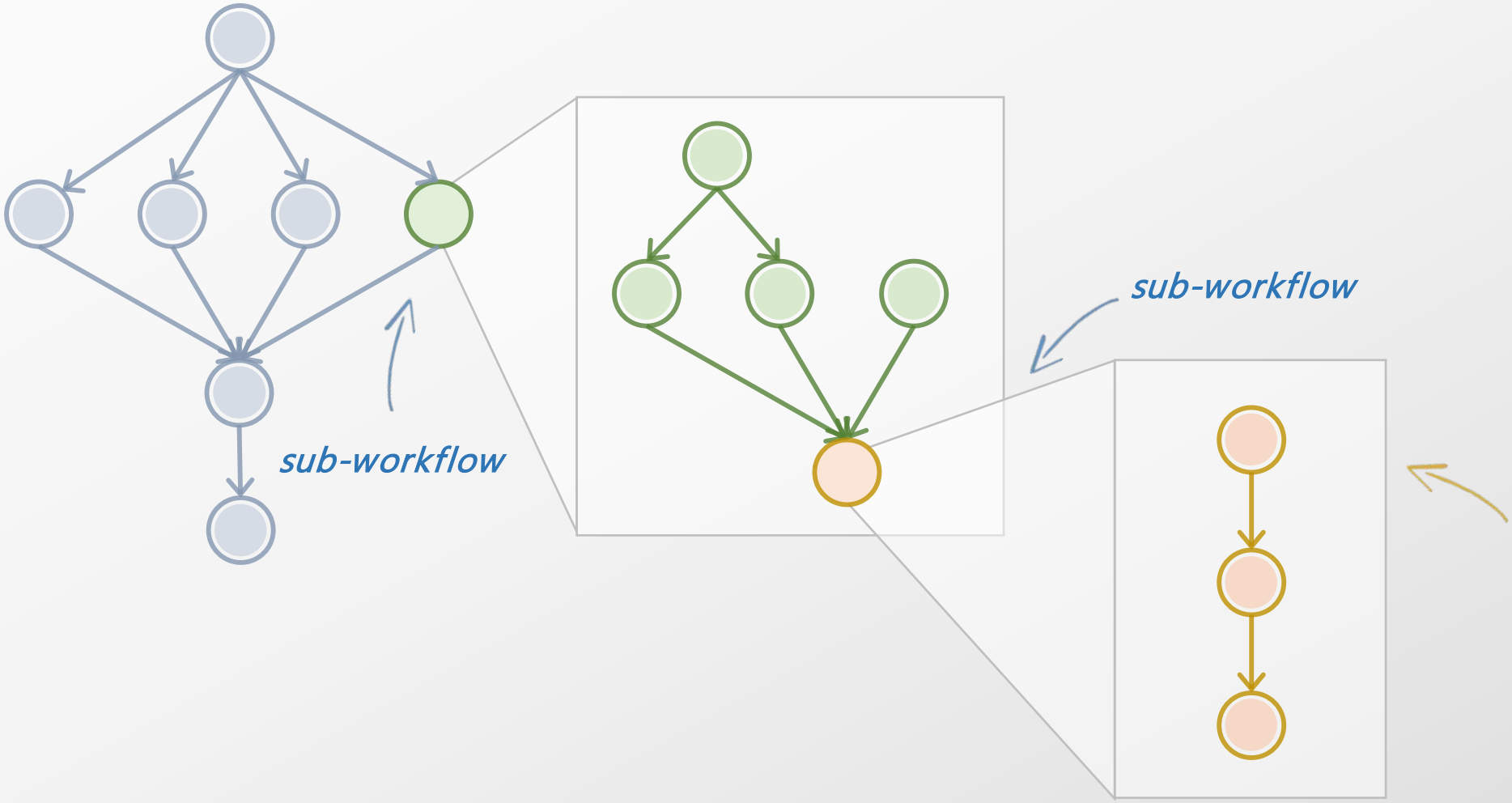


http://pegasus.isi.edu

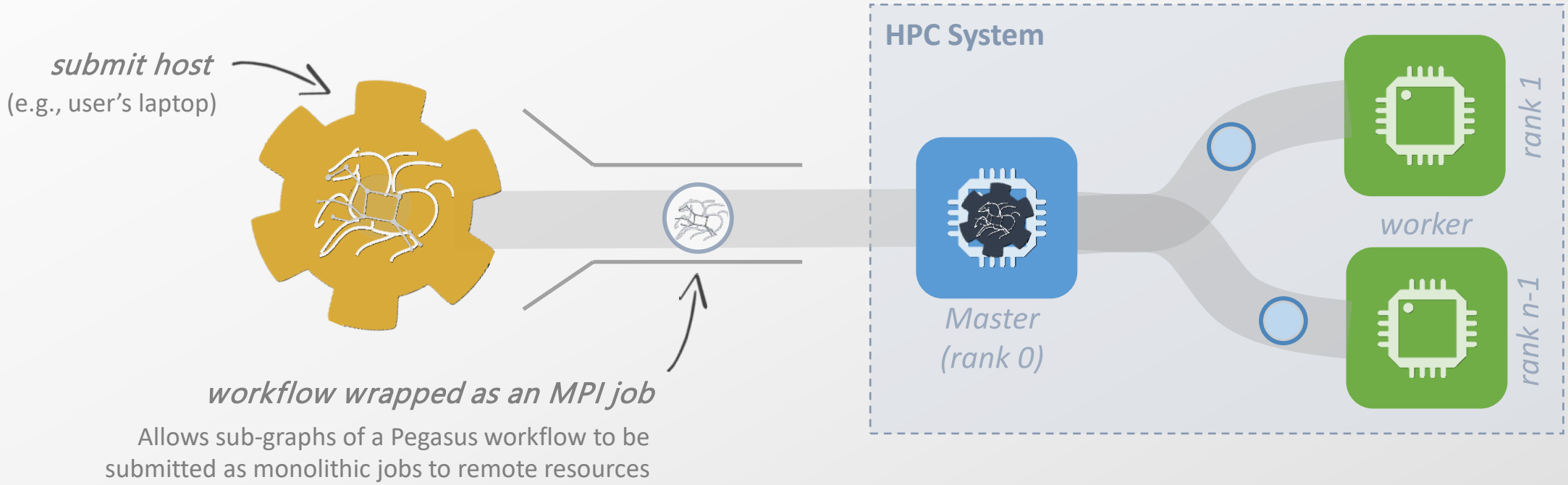
What about **data reuse**?



Pegasus also handles **large-scale workflows**



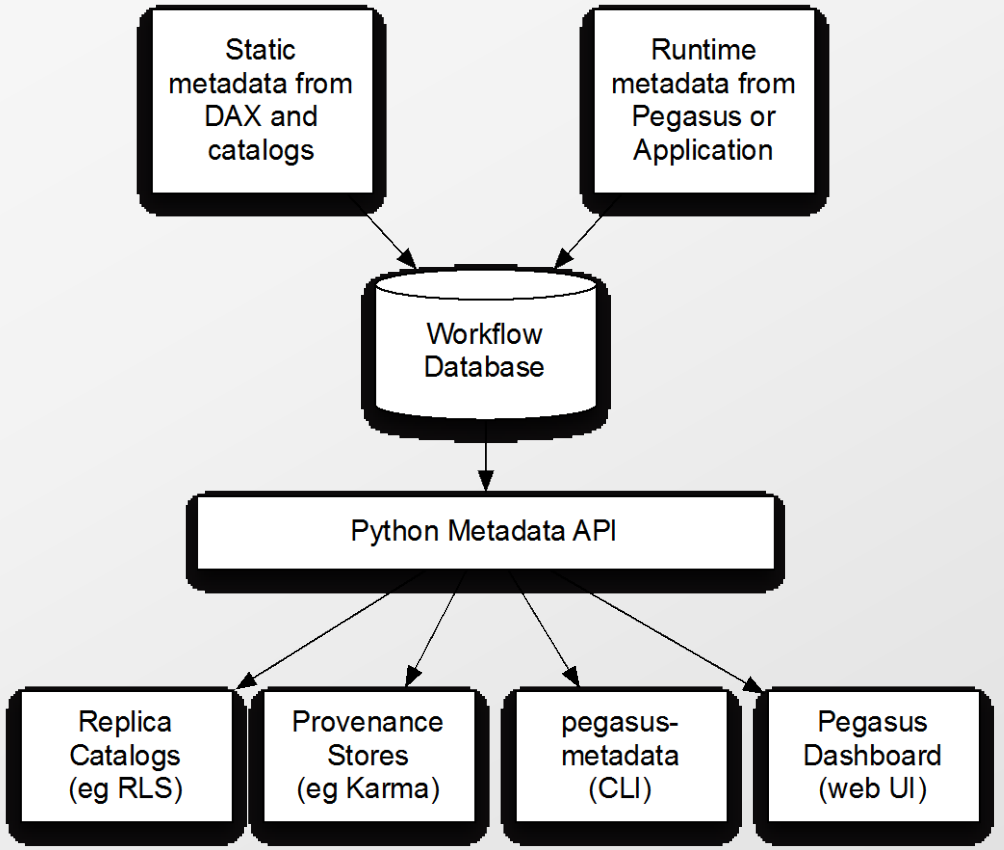
Running **fine-grained** workflows on HPC systems…



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

Metadata

- Can associate arbitrary key-value pairs with workflows, jobs, and files
- Replica selection
  - Input files are selected based on metadata attributes
- Data registration
  - Output files get tagged with metadata on registration
- Static and runtime metadata
  - Static: application parameters
  - Runtime: performance metrics



New in Pegasus 4.6, added to support users who want to select data based on attributes rather than names (e.g. LIGO)

DAX Metadata Example

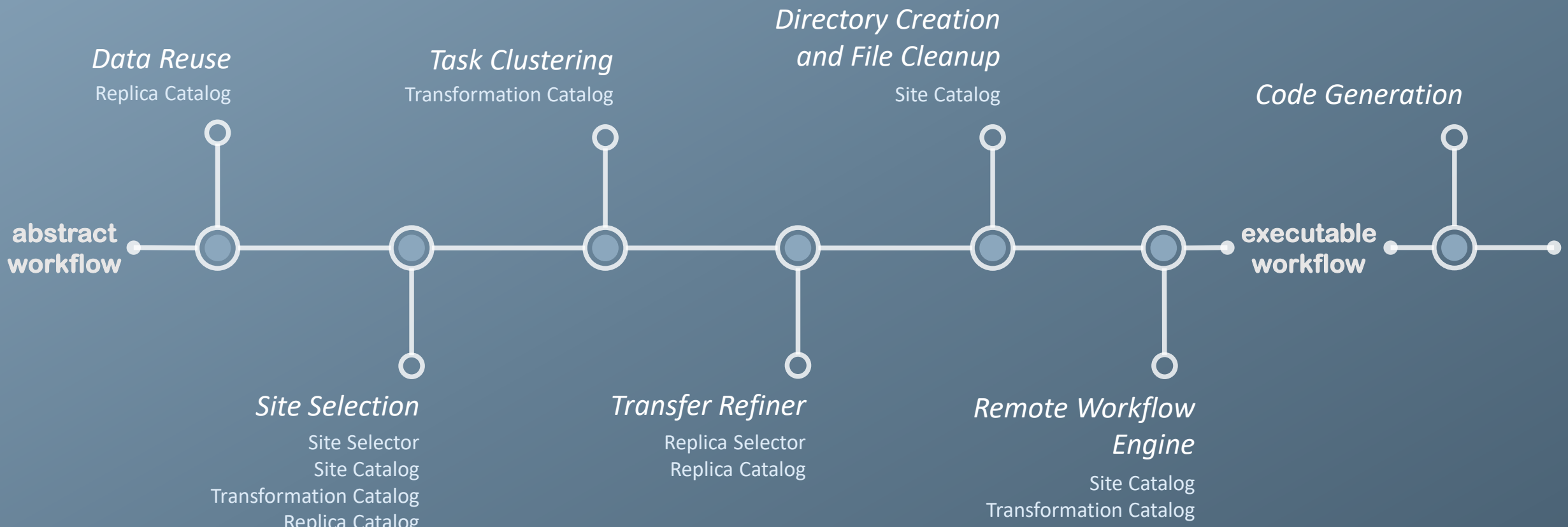
```
1  <adag ...>
       <metadata key="experiment">par_all27_prot_lipid</metadata>
       <job id="ID0000001" name="namd">
           <argument><file name="equilibrate.conf"/></argument>
5          <metadata key="timesteps">500000</metadata>
6          <metadata key="temperature">200</metadata>
7          <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

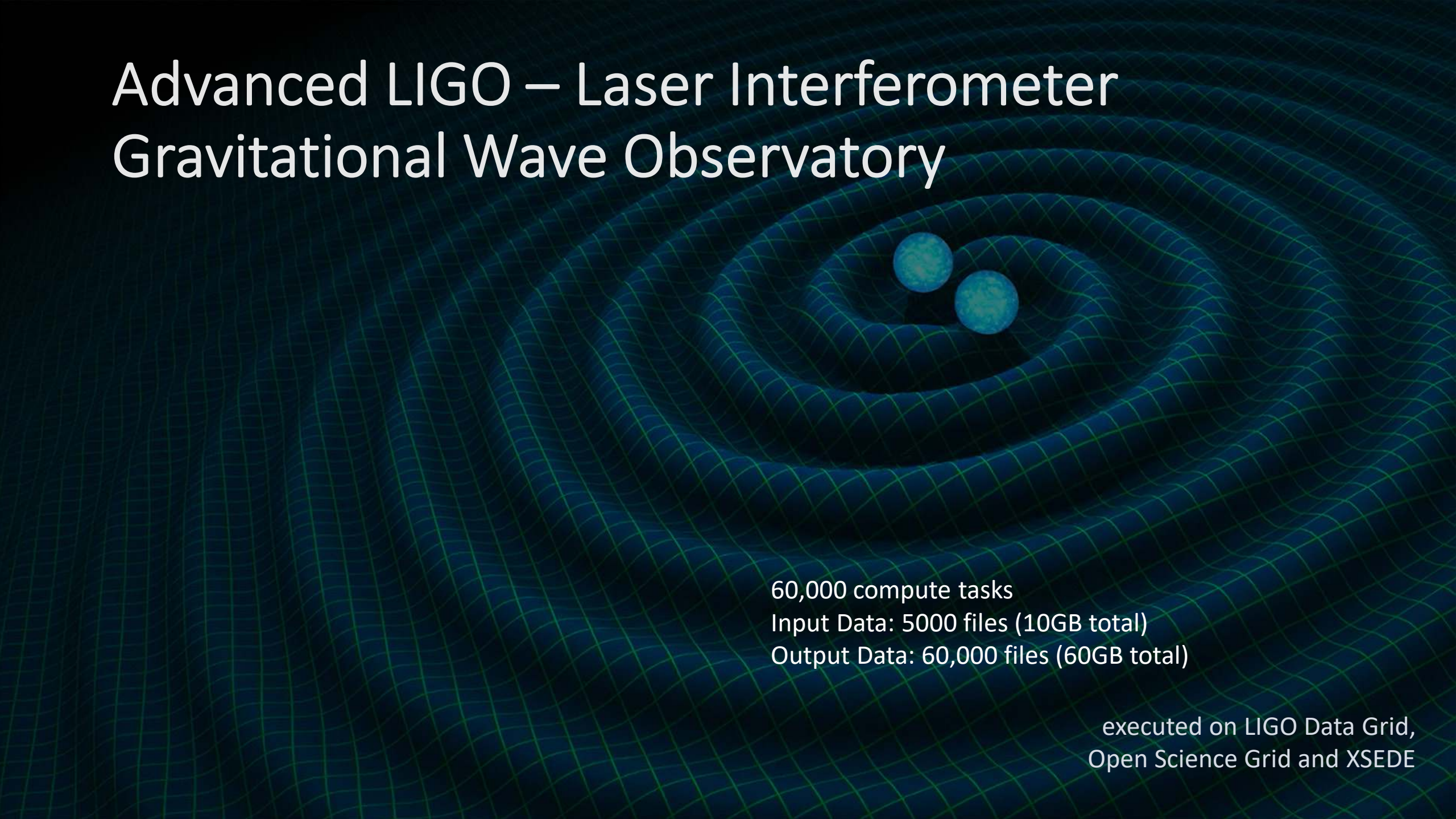
Select data based on metadata

Register data with metadata

Pegasus' flow at a glance



Advanced LIGO – Laser Interferometer Gravitational Wave Observatory

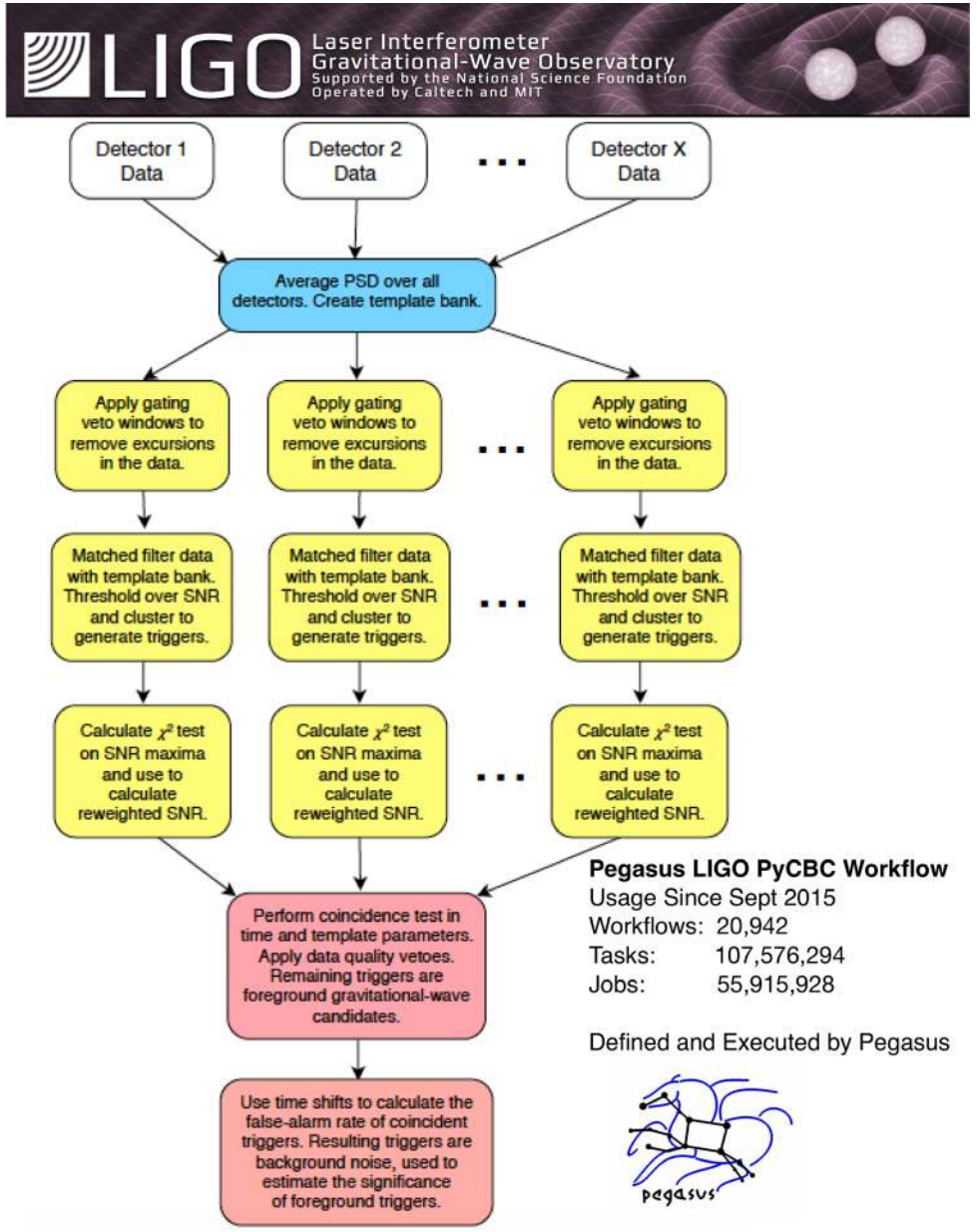


60,000 compute tasks
Input Data: 5000 files (10GB total)
Output Data: 60,000 files (60GB total)

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



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.*

Benefits to LIGO provided by Pegasus- Expanded Computing Horizons

- No longer limited to a single execution resource
  - Non Pegasus LIGO pipelines can often only run on LIGO clusters
  - Input is replicated out of band , in a rigid directory layout.
  - Rely on the shared filesystem to access data.

- Pegasus made it possible to leverage Non LDG Computing Resources

  - Open Science Grid
    - Dynamic – Best Effort Resource with no shared filesystem available

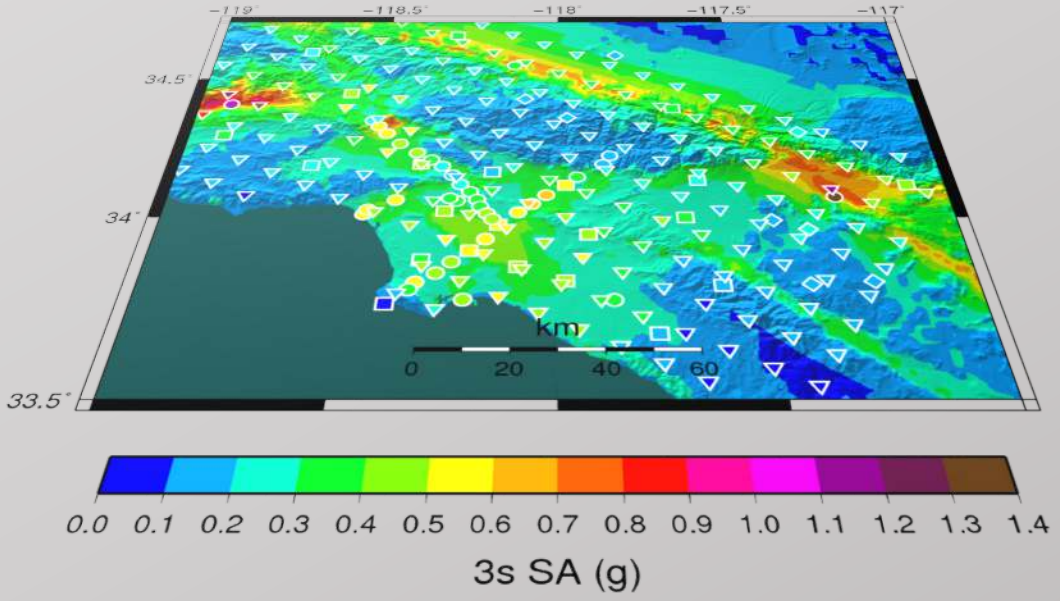
  - Large NSF Supercomputing Clusters XSEDE
    - No HTCondor
    - Geared for Large MPI jobs, not thousands of single node jobs
    - LIGO tried to setup XSEDE cluster as a LDG site but mismatch in setup.
    - Pegasus enabled LIGO to use XSEDE without changes at LIGO or at XSEDE

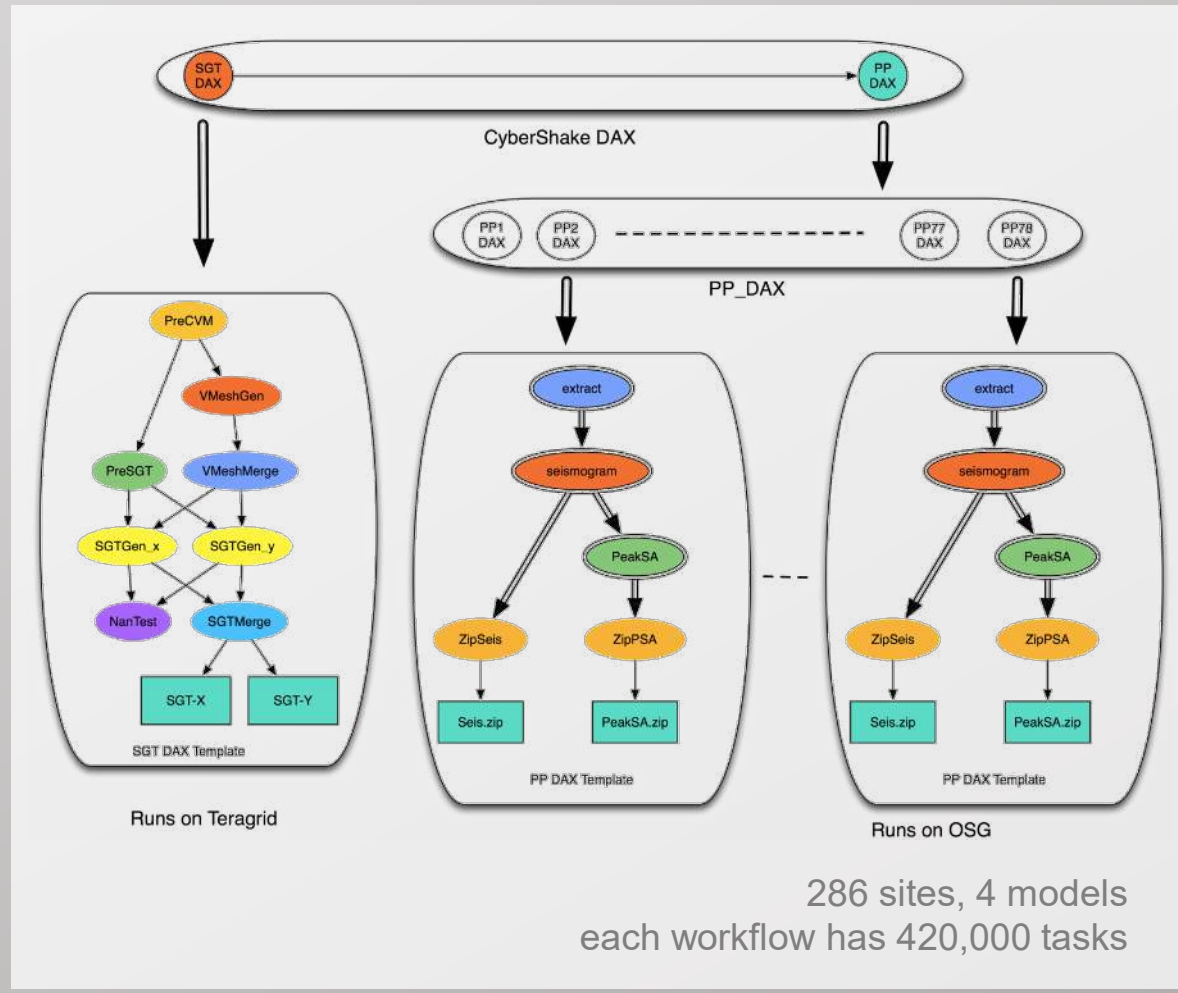
  - VIRGO Resources in Europe
    - Clusters with no shared filesystem and different storage management infrastructure than LDG
    - No HTCondor

Southern California Earthquake Center's CyberShake

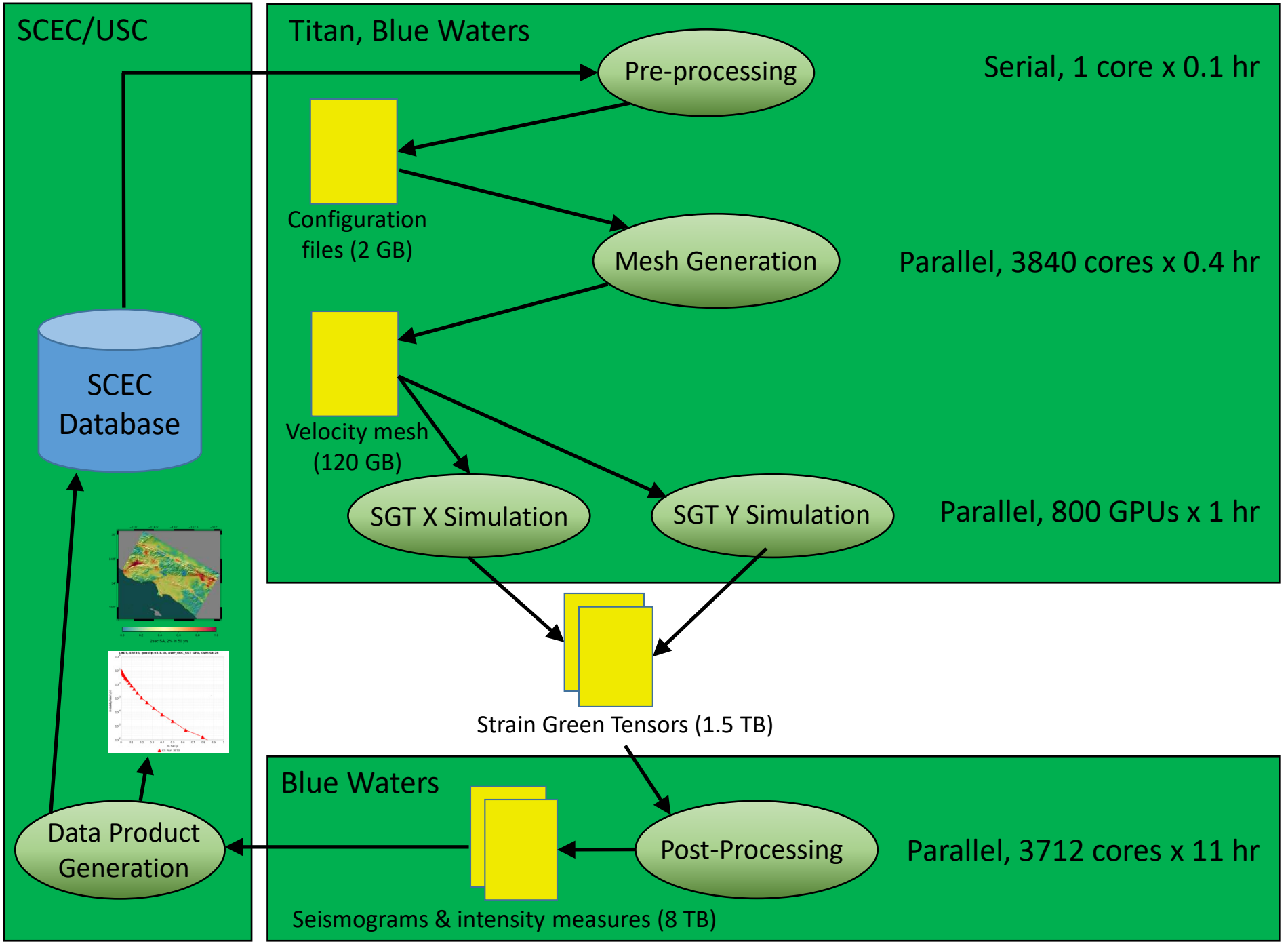
Builders ask seismologists: What will the peak ground motion be at my new building in the next 50 years?

Seismologists answer this question using Probabilistic Seismic Hazard Analysis (PSHA)



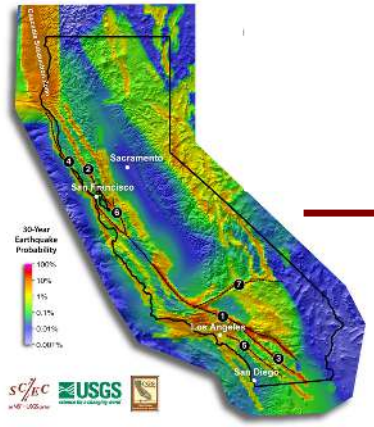


286 sites, 4 models
each workflow has 420,000 tasks


SCEC/USC
Titan, Blue Waters
Pre-processing
Serial, 1 core x 0.1 hr
Configuration files (2 GB)
Mesh Generation
Parallel, 3840 cores x 0.4 hr
SCEC Database
Velocity mesh (120 GB)
SGT X Simulation
SGT Y Simulation
Parallel, 800 GPUs x 1 hr
Strain Green Tensors (1.5 TB)
Blue Waters
Post-Processing
Parallel, 3712 cores x 11 hr
Data Product Generation
Seismograms & intensity measures (8 TB)

CyberShake Data Flow

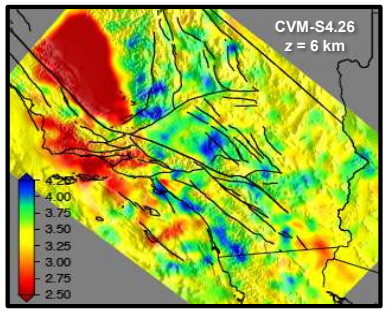
Most recent CyberShake study took 5 weeks of real time and used 38 million core-hours



Uniform California Earthquake Rupture Forecast

UCVM

Mesh generation
1 job per site
MPI, 3840 cores



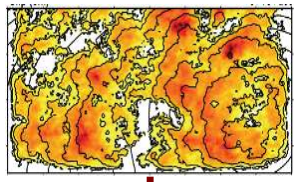
Community Velocity Model

AWP-ODC

SGT computation
2 jobs per site
MPI, 200-800 GPUs

NCSA *Blue Waters*
OLCF *Titan*

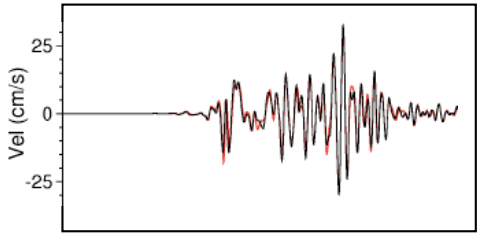
Graves-Pitarka kinematic rupture simulator



Seismogram Synthesis

Post-processing
~500,000 jobs per site
MPI master/worker, 3712 cores

NCSA *Blue Waters*
OLCF *Titan*

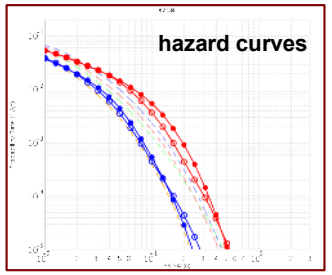


500,000 Seismograms
75M intensity measures

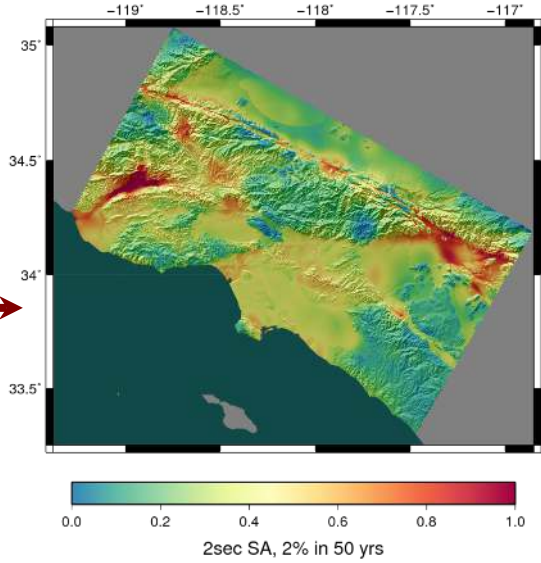
12 TB
data transfer

Data Product Generation

Populate DB, construct queries
6 jobs per site



CyberShake Hazard Map

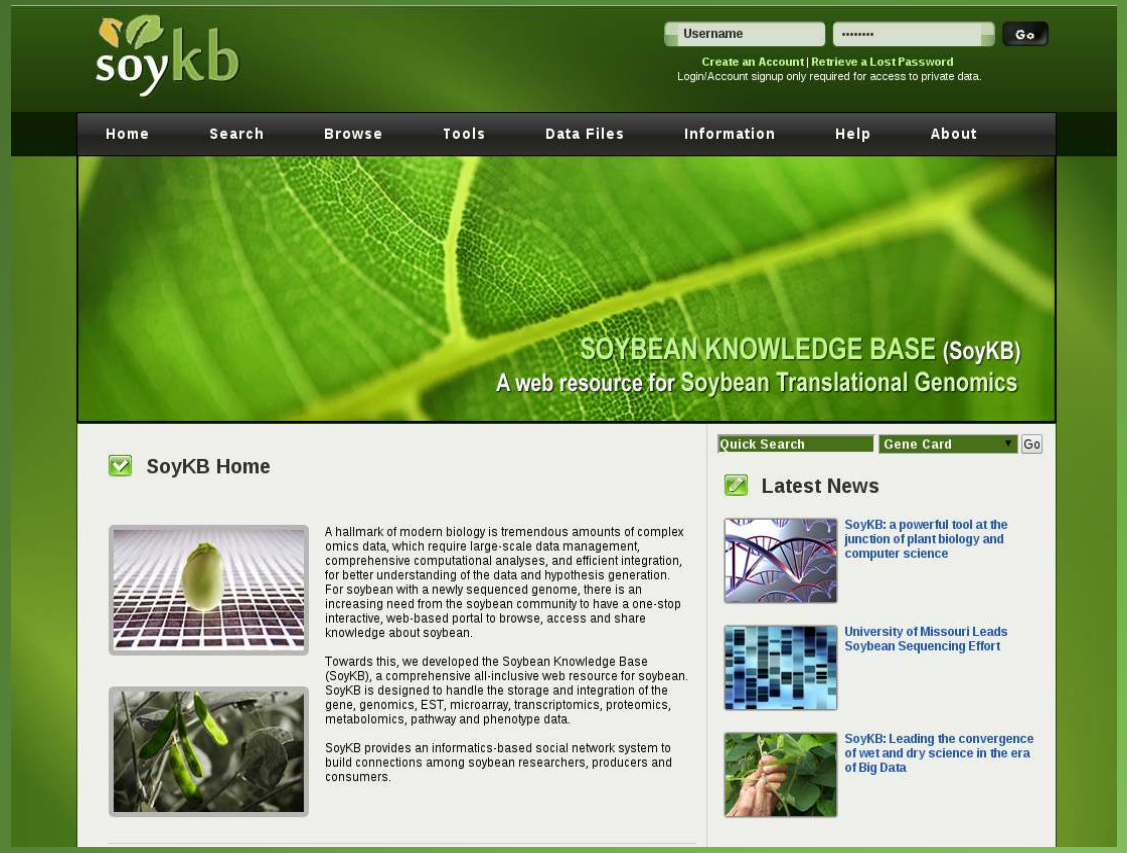


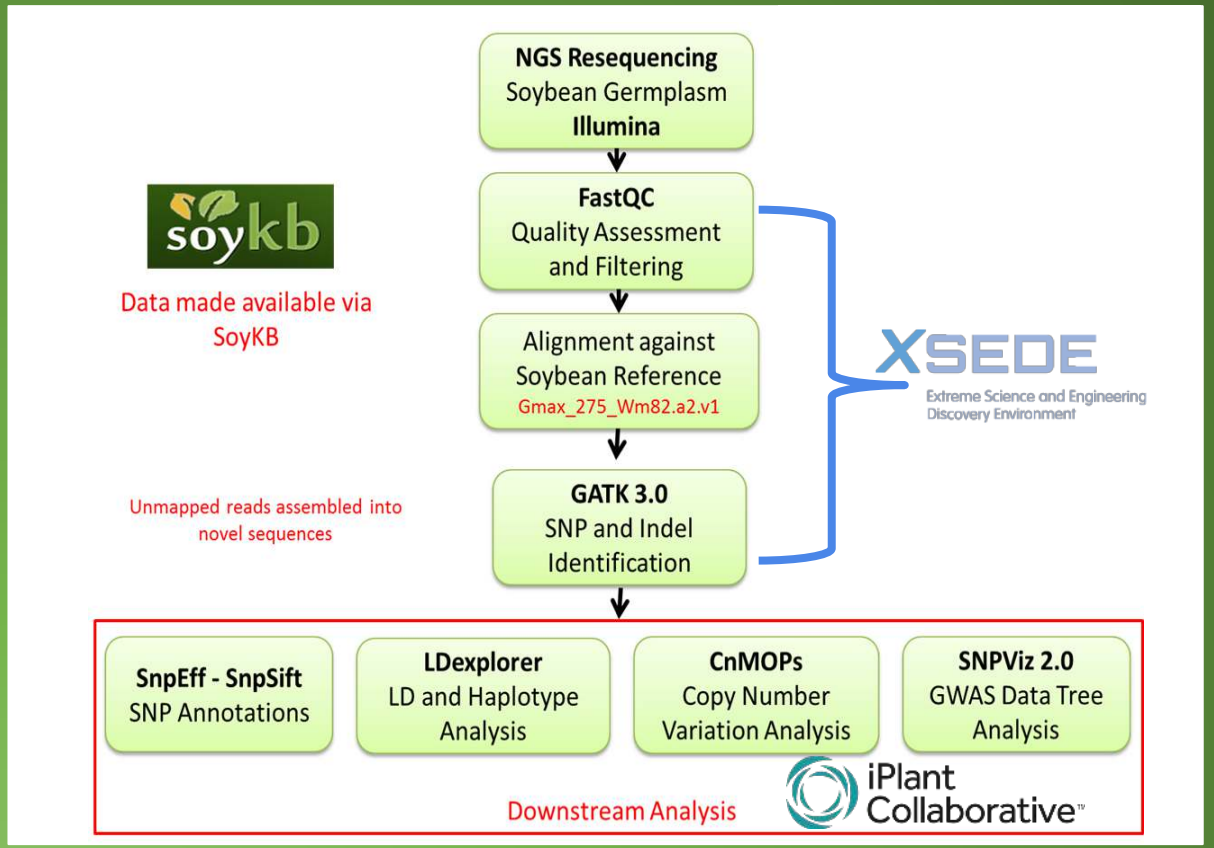
Central California CyberShake Run

- 438 sites x 2 velocity models (3D tomographic, 1D average)
- NCSA Blue Waters (75%) and OLCF Titan (25%)
- 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

CyberShake Study 15.4

- 336 sites
- NCSA Blue Waters (69%) and OLCF Titan (31%)
- CPU jobs (Mesh generation, seismogram synthesis): 590,000 node-hours
- GPU jobs: 795,000 node-hours
  - AWP-ODC finite-difference code
  - 10 billion points per volume, 40000 timesteps
  - 800 GPUs for 1 hour
- Titan:
  - 428,000 GPU node-hours
- Blue Waters:
  - 590,000 CPU node-hours, 366,000 GPU node-hours

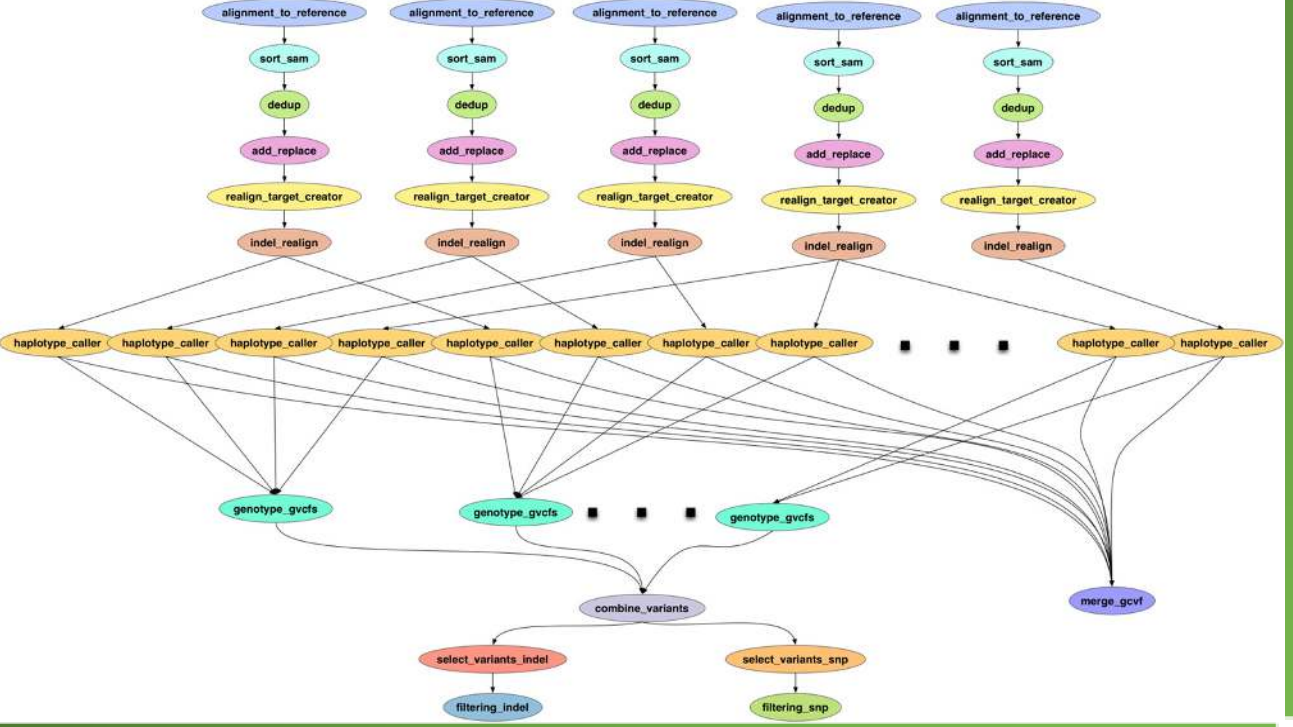




http://soykb.org

XSEDE Allocation
PI: Dong Xu
Trupti Joshi, Saad Kahn, Yang Liu, Juexin Wang, Badu Valliyodan, Jiaojiao Wang

https://github.com/pegasus-isi/Soybean-Workflow



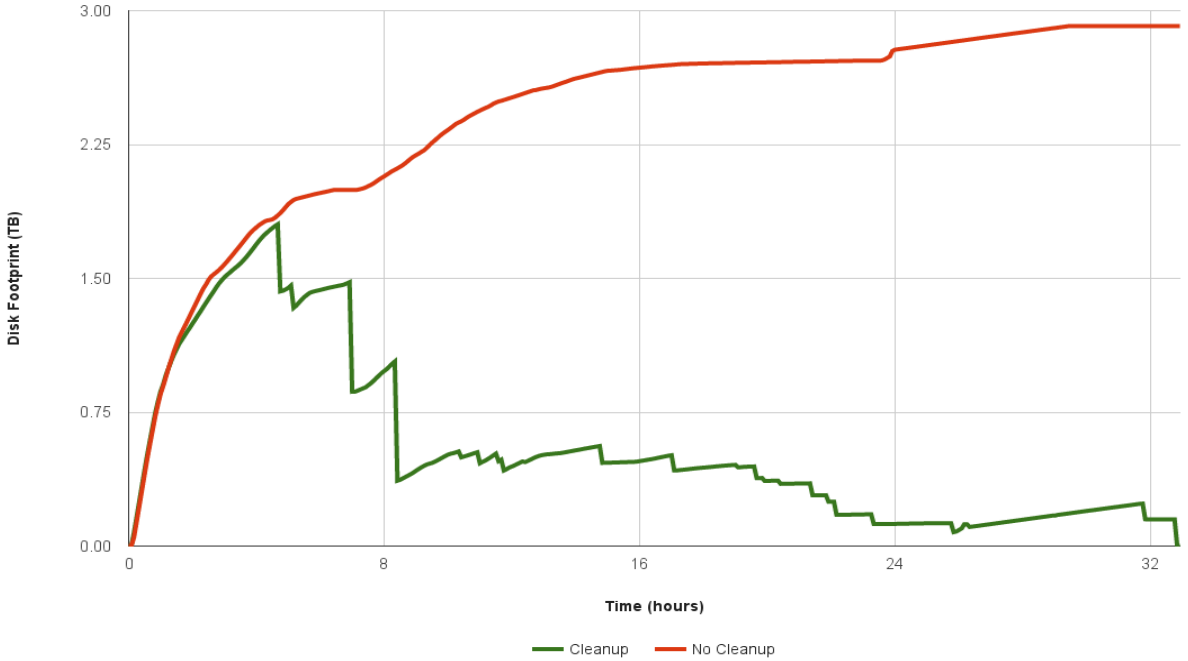
Task	Base Code	Cores (Threads)	Memory (GB)
Alignment_to_reference	BWA	7	8
Sort_sam	Picard	1	21
Dedup	Picard	1	21
Add_replace	Picard	1	21
Realign_target_creator	GATK	15	10
Indel_realign	GATK	1	10
Haplotype_caller	GATK	1	3
Genotype_gvcfs	GATK	1	10
Merge_gvcf	GATK	10	20
Combine_variants	GATK	1	10
Select_variants	GATK	14	10
Filtering	GATK	1	10

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 in 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)



Pegasus

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

HipChat

Pegasus est. 2001

Automate, recover, and debug scientific computations.

Thank You

Questions?

Mats Rynge
rynge@isi.edu

USC Viterbi
School of Engineering
Information Sciences Institute

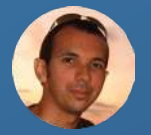
Meet our team


Ewa Deelman


Karan Vahi


Mats Rynge


Rajiv Mayani


Rafael Ferreira da Silva

NSF U.S. DEPARTMENT OF ENERGY NIH