Pegasus Workflow Management System

Mats Rynge
rynge@isi.edu

George Papadimitriou
georgpap@isi.edu

Ewa Deelman
deelman@isi.edu

USC Viterbi
School of Engineering
Information Sciences Institute

https://pegasus.isi.edu

Outline

- Pegasus overview
- User Stories
- More Pegasus features
- Pegasus in OpenShift

Why Pegasus ?

Automates complex, multi-stage processing pipelines

Enables parallel, **distributed or remote computations**

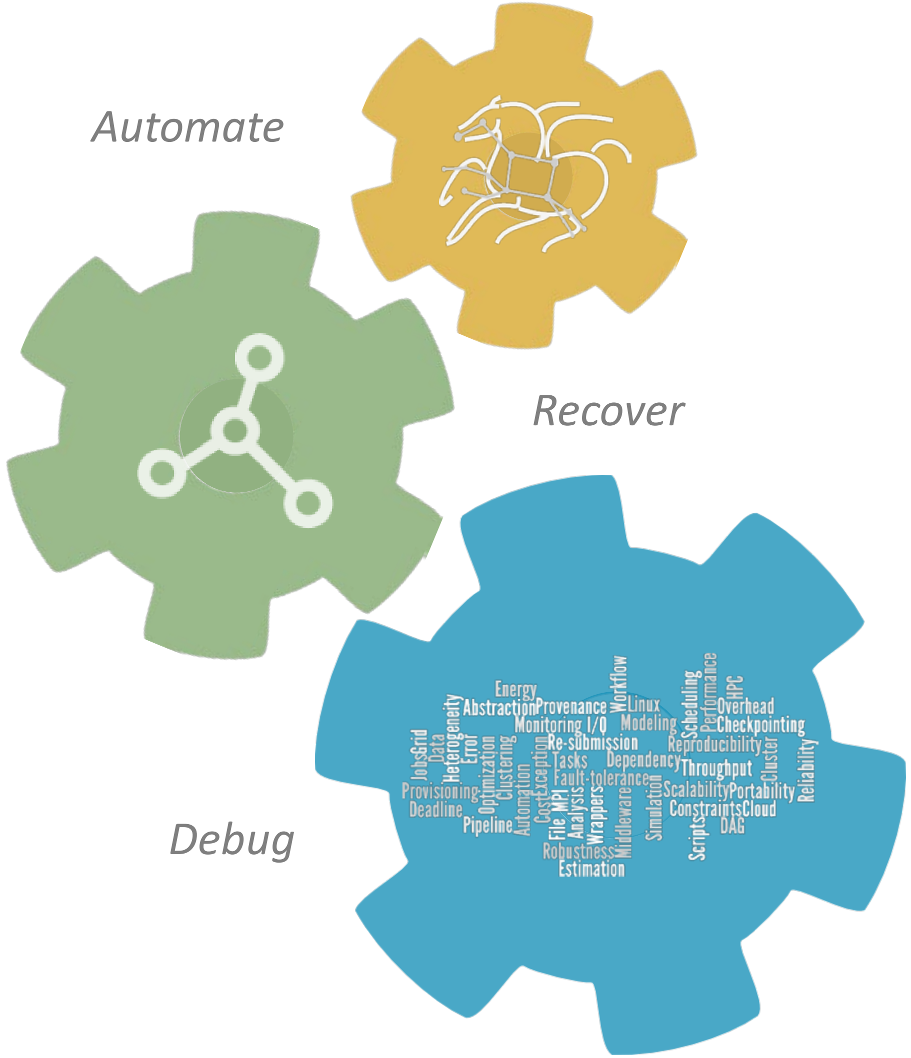
Automatically executes **data transfers**

Reusable, aids **reproducibility**

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

Handles **failures** with to provide reliability

Keeps track of data and **data integrity**



HTCondor
High Throughput Computing

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

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

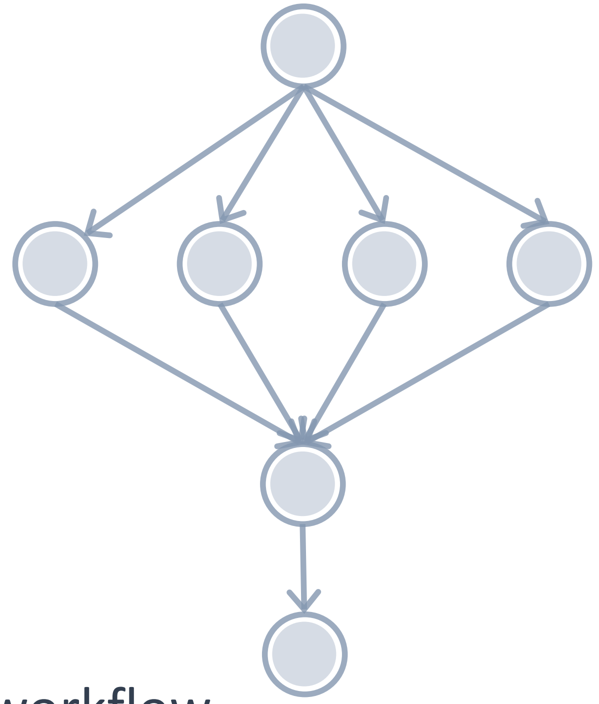
Workflows are DAGs (Directed Acyclic Graphs)

- 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

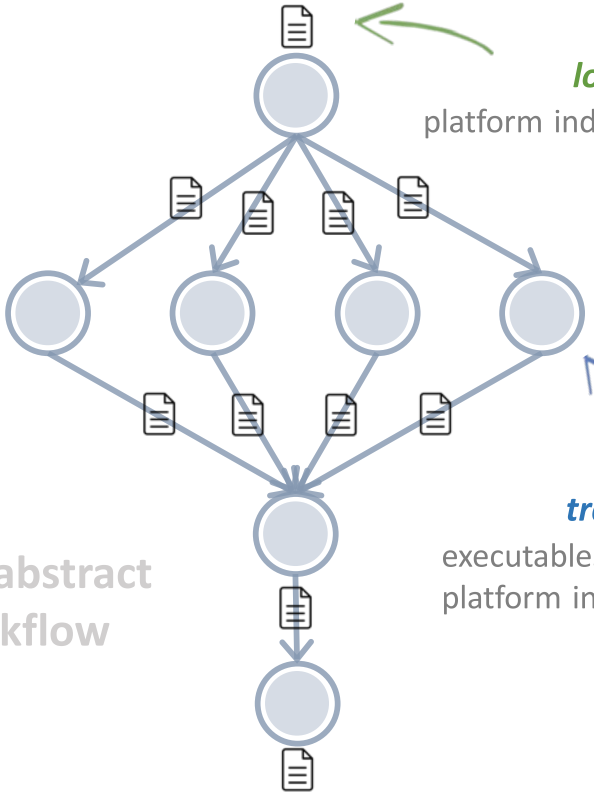


DAX

DAG in XML

Portable Description

Users do not worry about
low level execution details



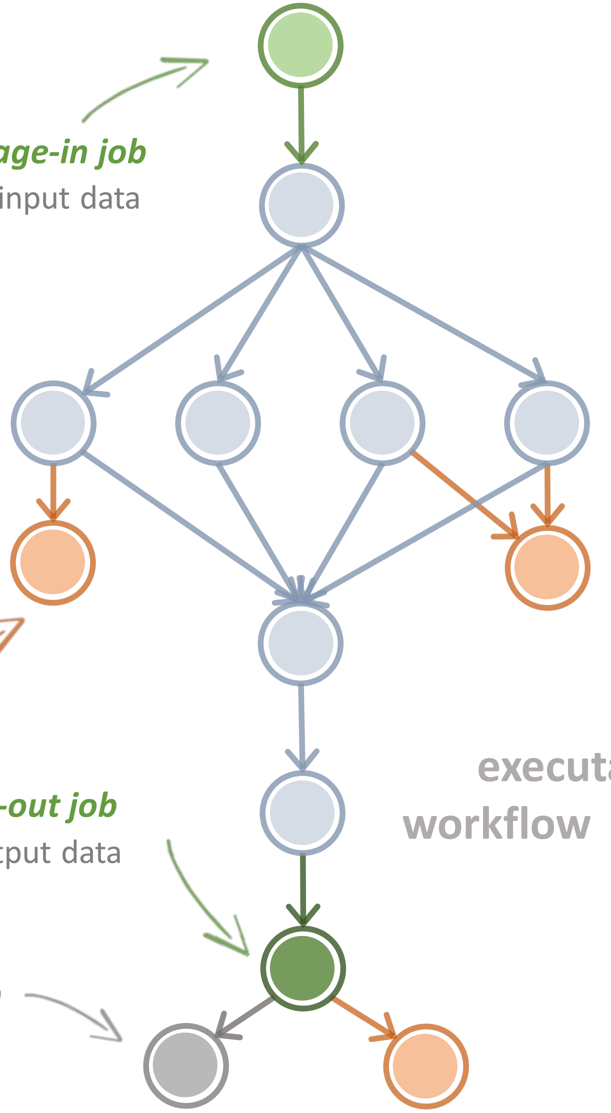
logical filename (LFN)
platform independent (abstraction)

transformation
executables (or programs)
platform independent

abstract
workflow

DAG

directed-acyclic graphs



stage-in job
Transfers the workflow input data

cleanup job
Removes unused data

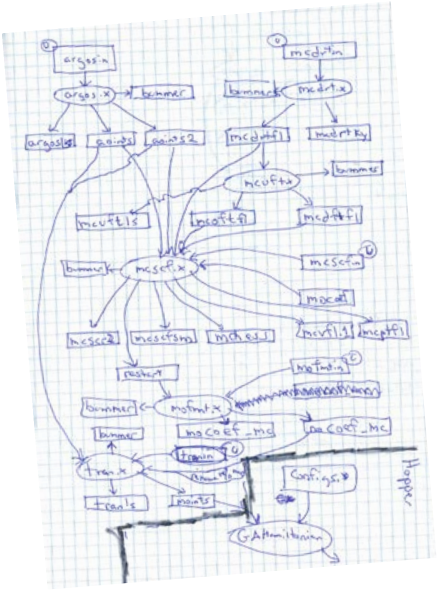
stage-out job
Transfers the workflow output data

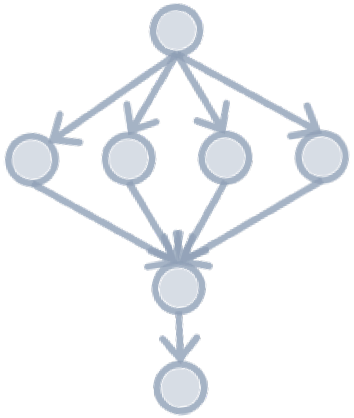
registration job
Registers the workflow output data

executable
workflow

Pegasus

Pegasus also provides tools to generate the abstract workflow…





```python
#!/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
<?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

So, what information does Pegasus need?

Transformation Catalog

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

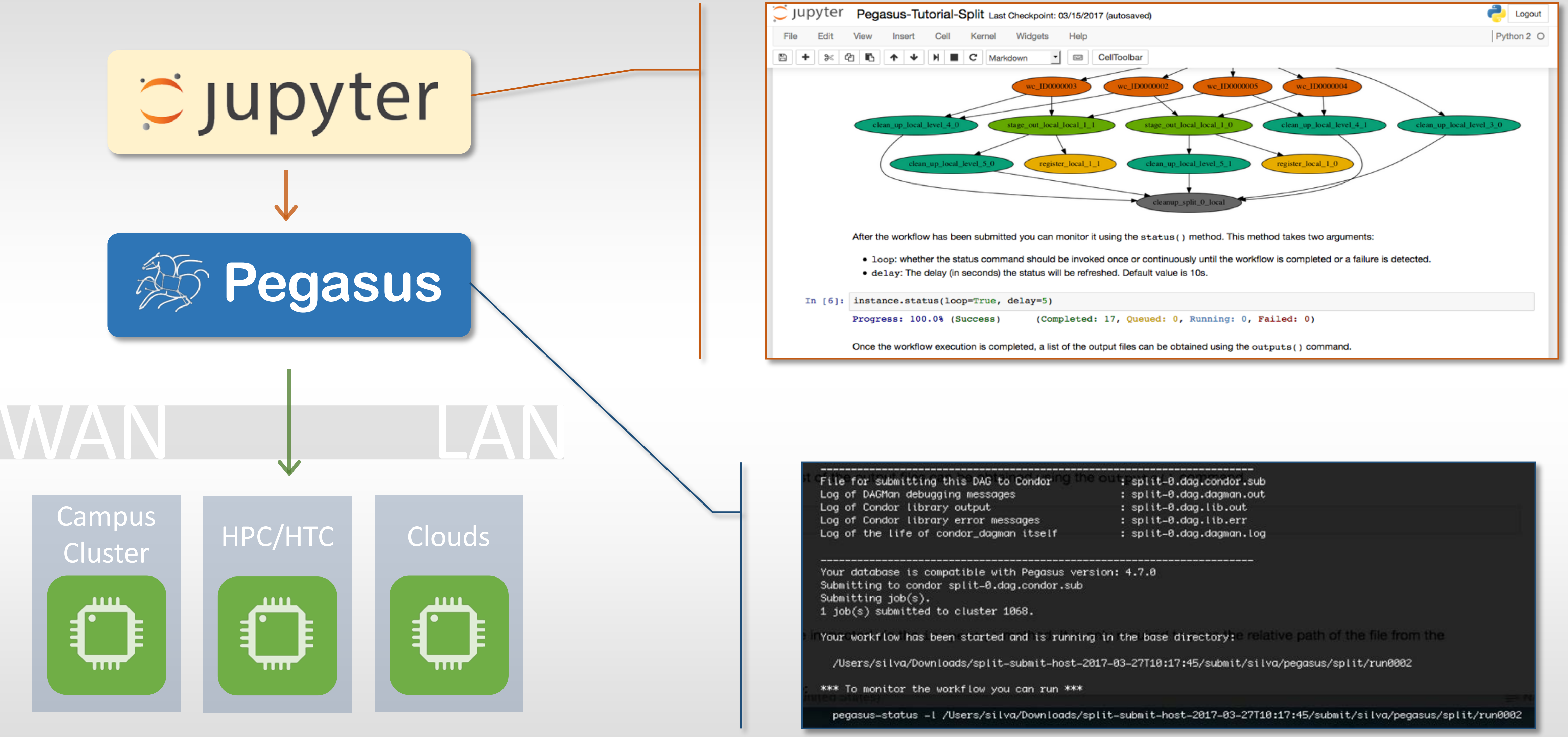
Site Catalog

describes the sites where the workflow jobs are to be executed

Replica Catalog

describes all of the input data stored on external servers

Running Pegasus workflows with Jupyter



>_ command-line...

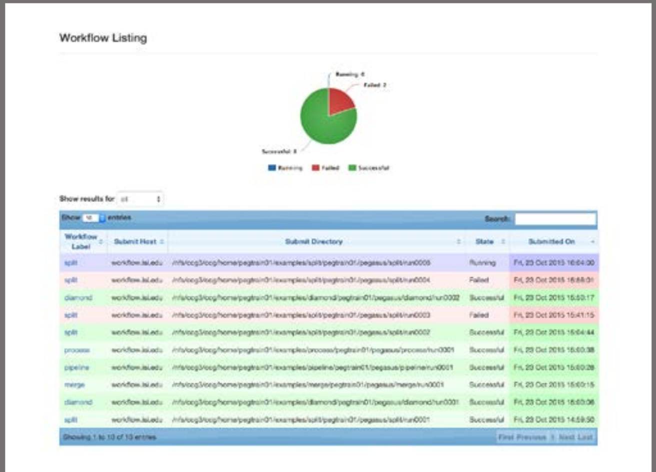
```
$ pegasus-status pegasus/examples/split/run0001
STAT IN_STATE JOB
Run 00:39 split-0 (/home/pegasus/examples/split/run0001)
Idle 00:03 └──split_ID0000001
Summary: 2 Condor jobs total (I:1 R:1)

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

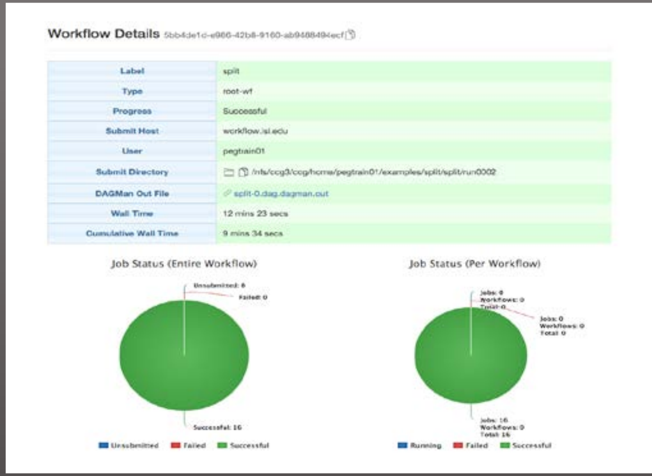


Pegasus

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

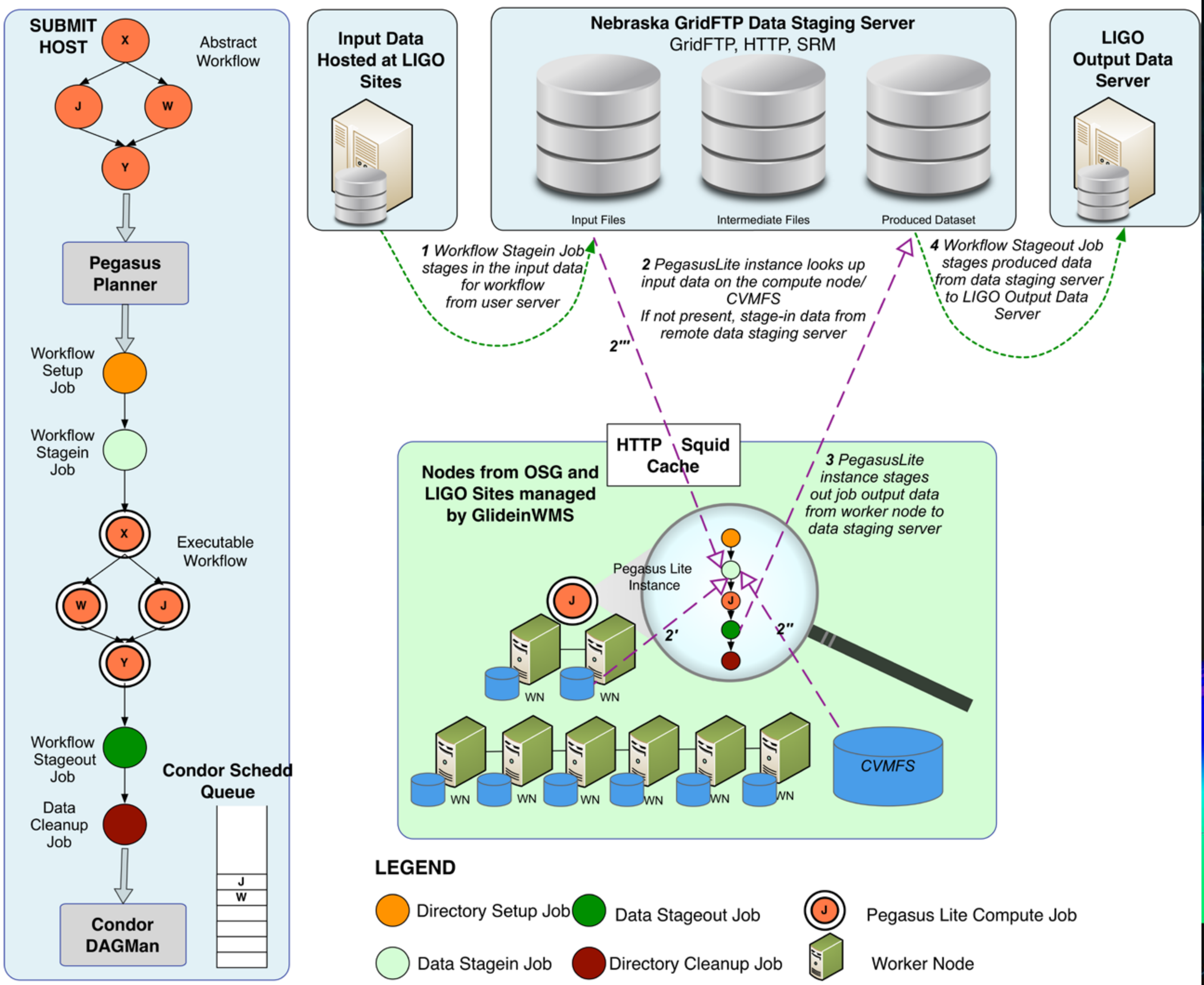
Debugging

Troubleshooting

RESTful API

User Stories

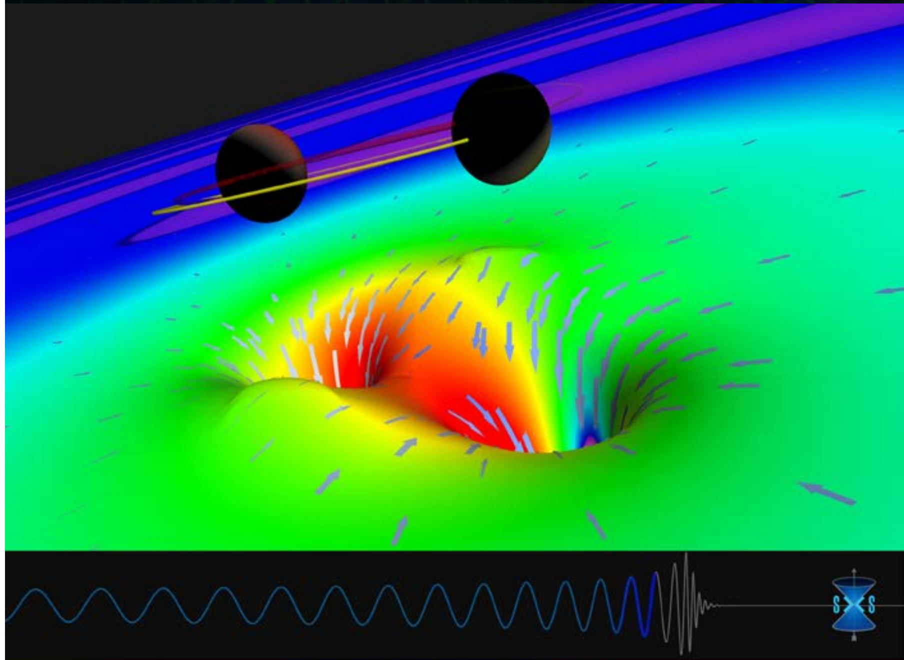
Data Flow for LIGO Pegasus Workflows in OSG



Advanced LIGO – Laser Interferometer Gravitational Wave Observatory

First gravitational wave detection:
21k Pegasus Workflows
107M tasks

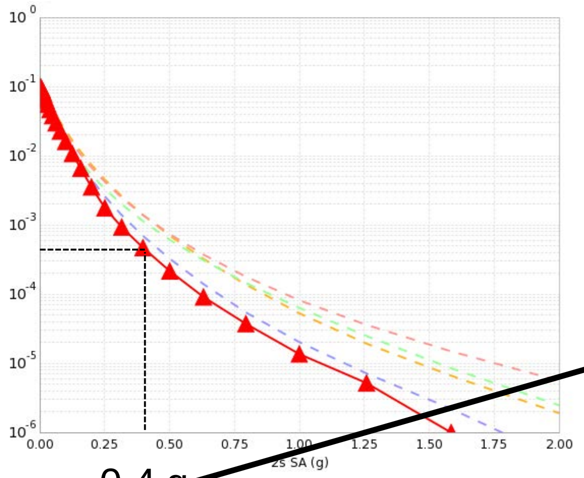
PyCBC Executed on LIGO Data Grid, Open Science Grid and XSEDE



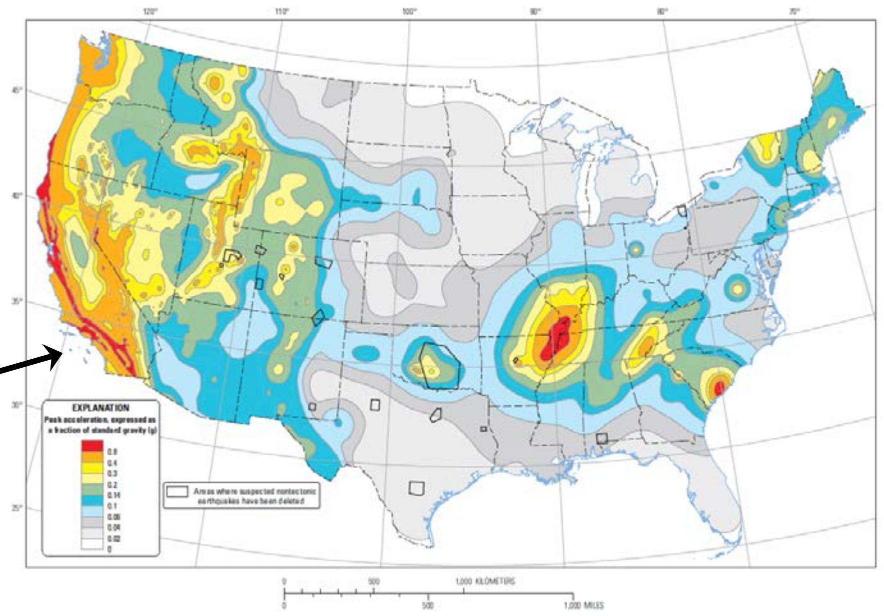
Probabilistic Seismic Hazard Analysis (PSHA)

- What will peak earthquake shaking be over the next 50 years?
- Useful information for:
  - Building engineers
  - Disaster planners
  - Insurance agencies

2% in 50 yrs



0.4 g

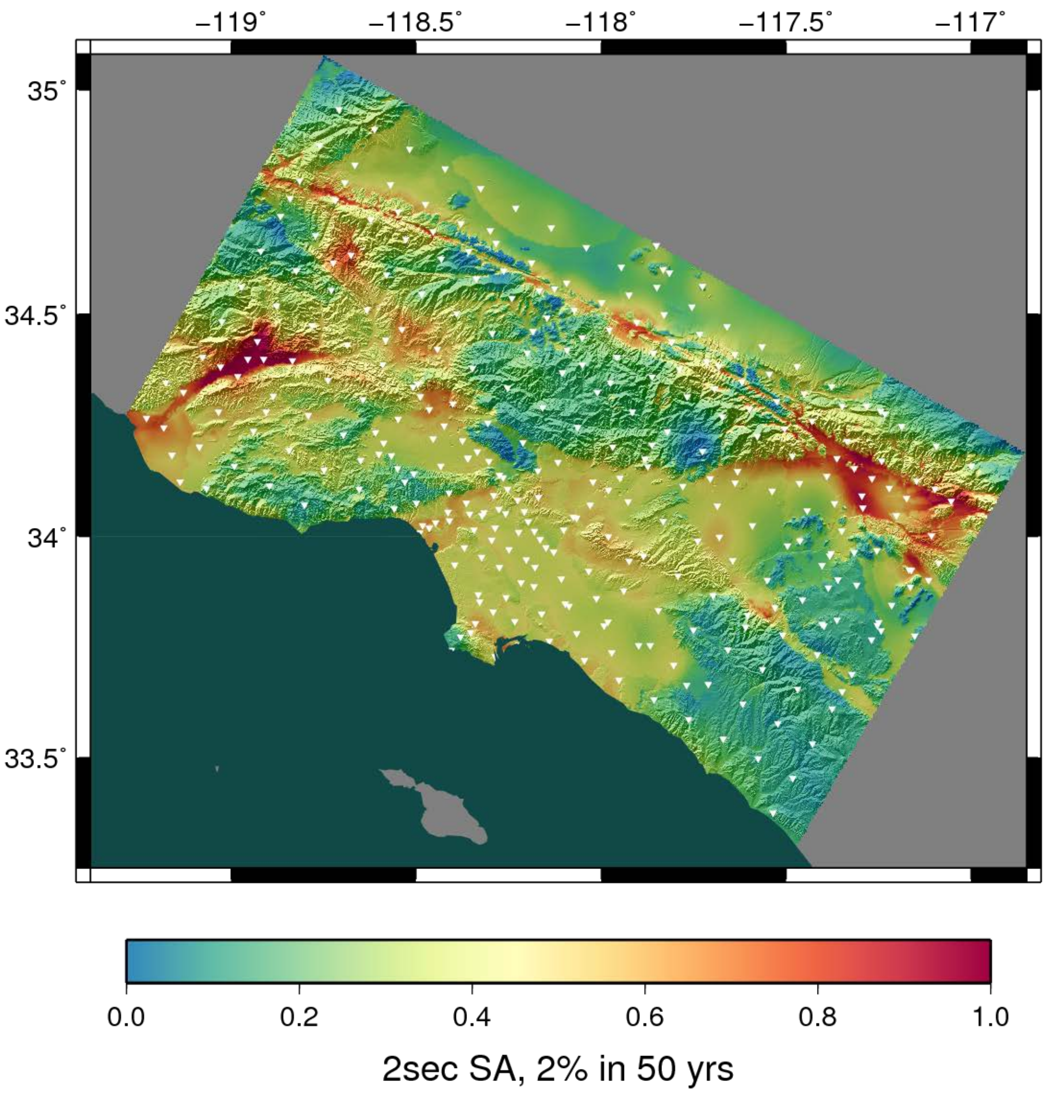


Two-percent probability of exceedance in 50 years map of peak ground acceleration

- PSHA performed by
  1. Assembling a list of earthquakes
  2. Determining how much shaking each event causes
  3. Combining the shaking levels with probabilities

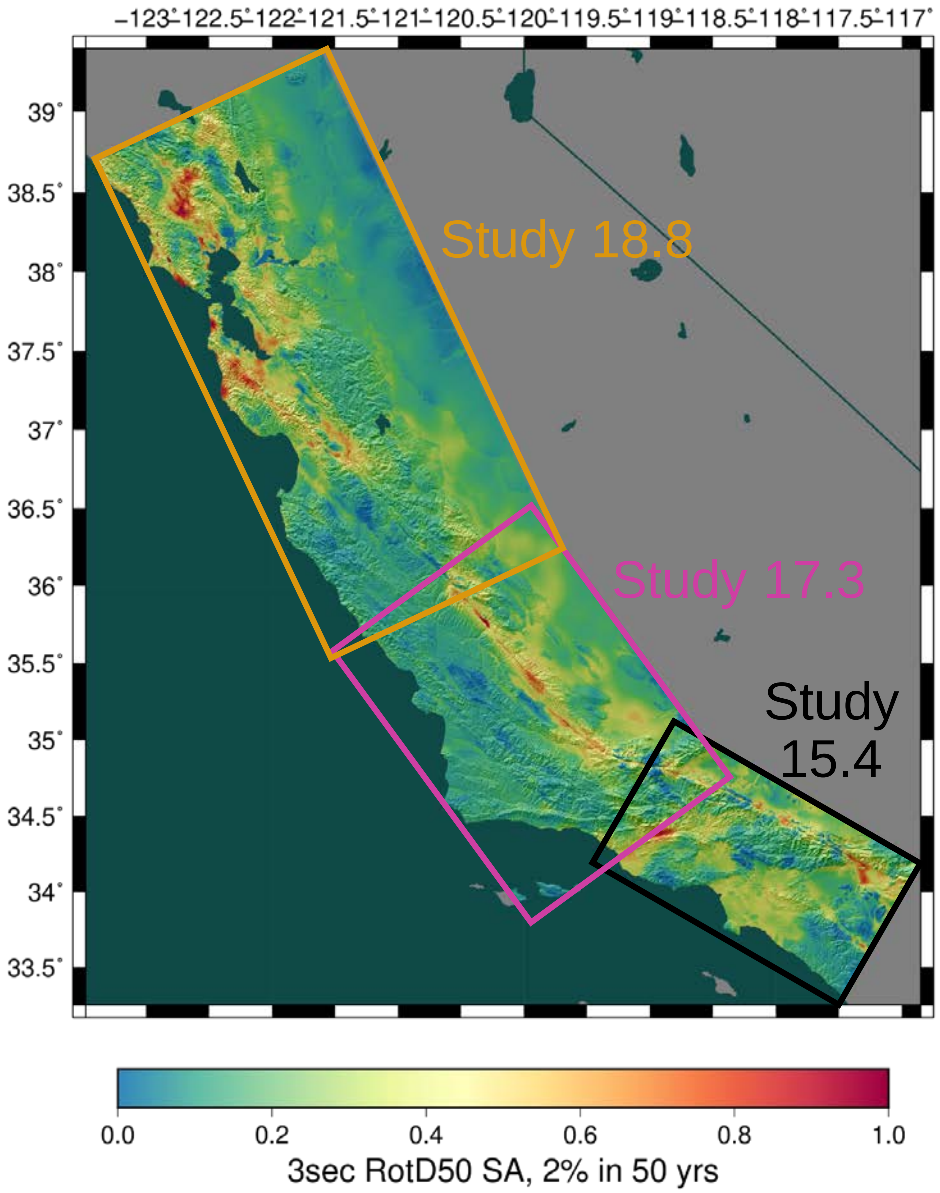
SCEC CyberShake Project

- 3D physics-based platform for PSHA
- For each site of interest:
  - Determine nearby (<200 km) earthquakes
  - Add variability to earthquakes
  - Simulate each of 500,000 earthquakes
  - Determine maximum shaking from each
  - Combine with probabilities to produce curve
- Repeat process for multiple locations
- Continual improvement since 2007



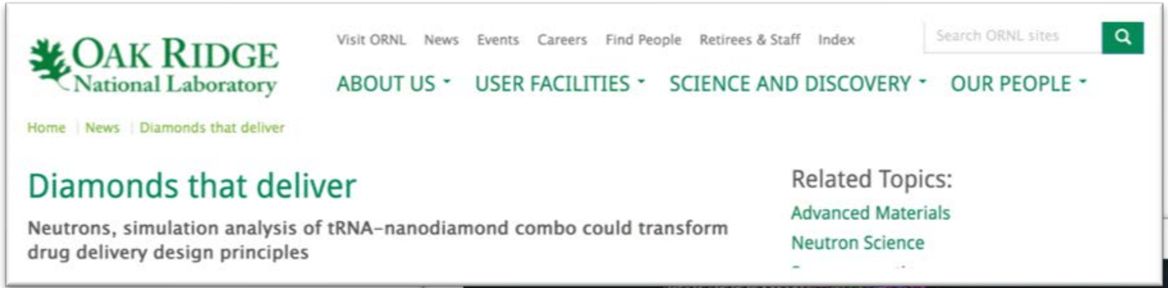
CyberShake Study 18.8 Metrics

- Study conducted over 128 days
- Consumed 6.2 million node-hours (120M core-hours/13,650 core-years)
  - Averaged 2,018 nodes / 38,850 cores
  - Max of 16,219 nodes / 279,984 cores
- Ran 21,220 jobs at USC, 10,308 at Blue Waters, and 7,757 jobs at Titan
- 1.2 PB of data generated
  - 157 TB of data automatically transferred
  - 14.4 TB of final data products staged to USC HPC
- Simulated 203 million seismograms
  - 30.4 billion shaking values



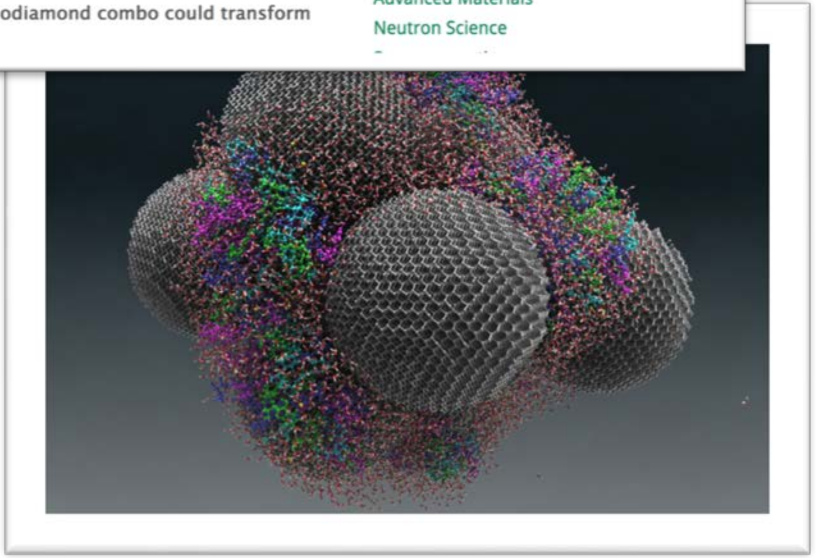
Impact on DOE Science

Enabled cutting-edge domain science (e.g., drug delivery) through collaboration with scientists at the DoE **Spallation Neutron Source (SNS)** facility



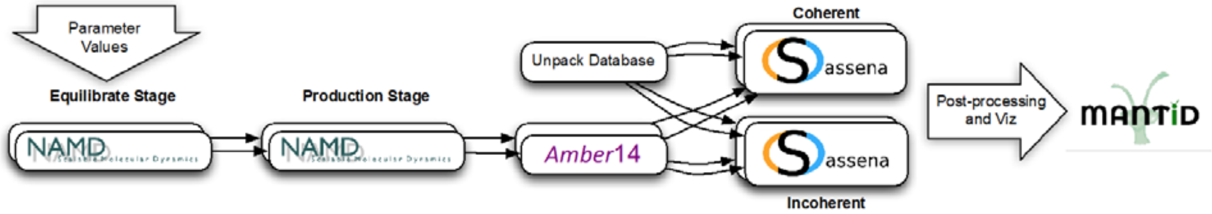
A Pegasus workflow was developed that confirmed that ***nanodiamonds*** can enhance the dynamics of tRNA

- It compared SNS neutron scattering data with MD simulations by calculating the epsilon that best matches experimental data



Water is seen as small red and white molecules on large nanodiamond spheres. The colored tRNA can be seen on the nanodiamond surface. (Image Credit: Michael Mattheson, OLCF, ORNL)

Ran on a Cray XE6 at NERSC using 400,000 CPU hours, and generated 3TB of data.



Predictive Modeling and Diagnostic Monitoring of Extreme Science Workflows

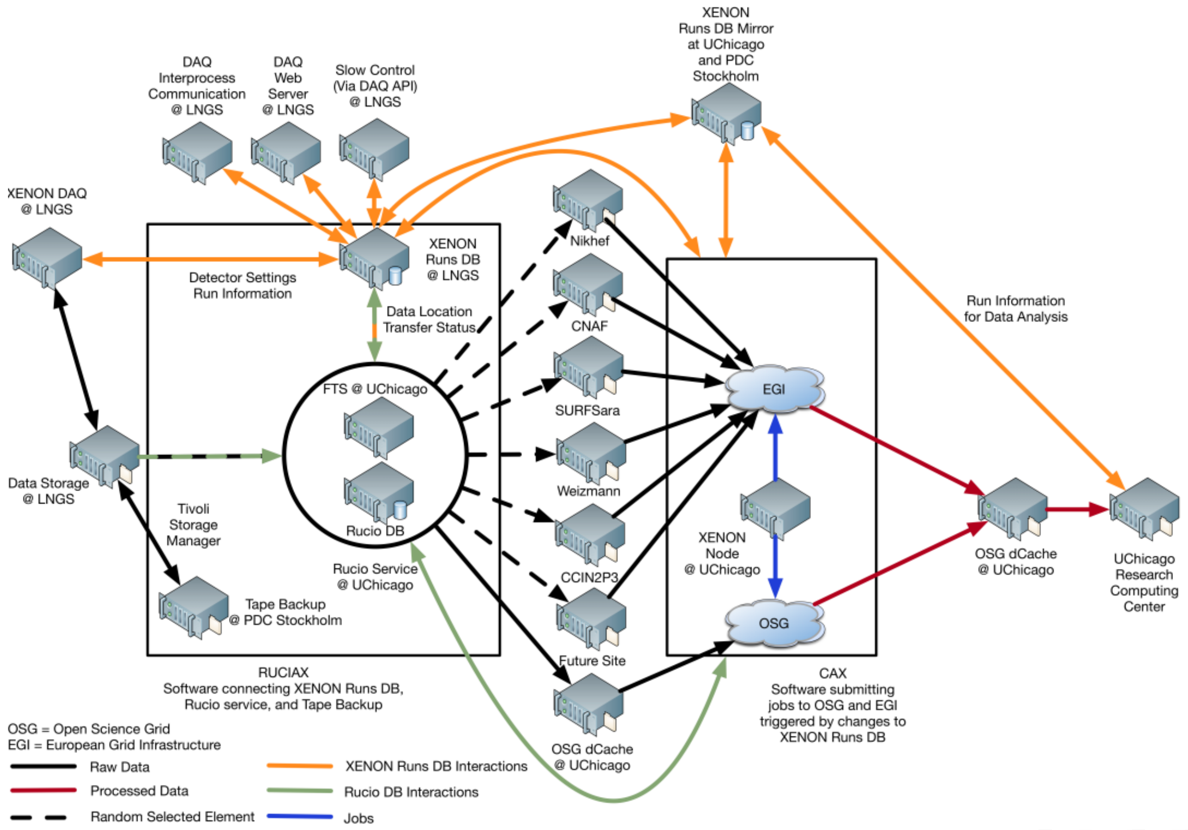
PANORAMA

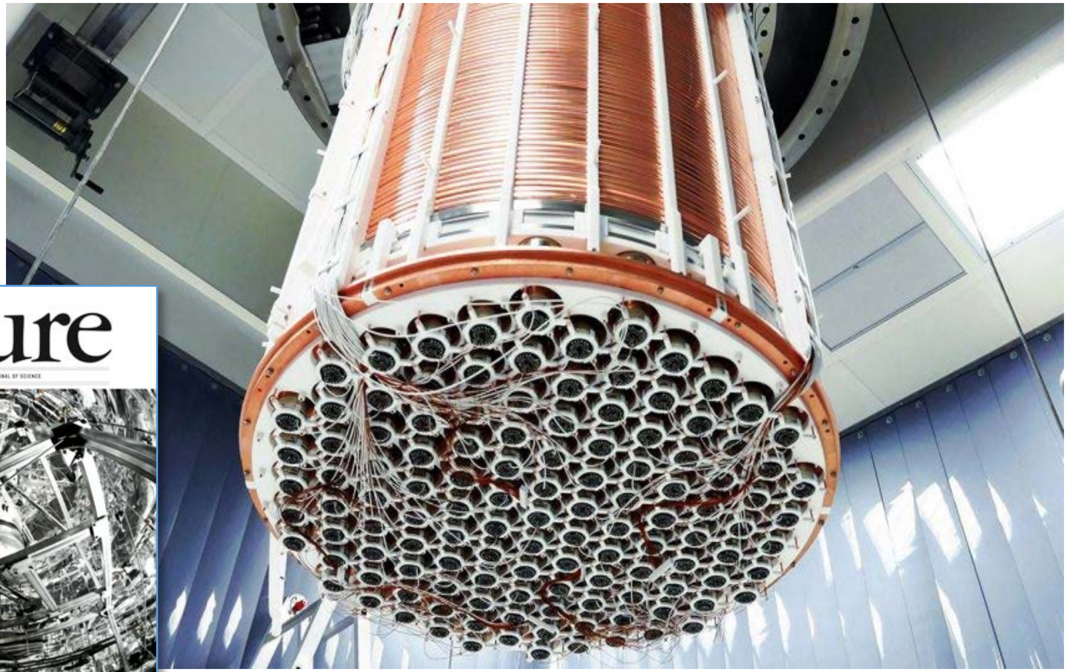
An automated analysis workflow for optimization of force-field parameters using neutron scattering data*. V. E. Lynch, J. M. Borreguero, D. Bhowmik, P. Ganesh, B. G. Sumpter, T. E. Proffen, M. Goswami, Journal of Computational Physics, July 2017.*

Pegasus

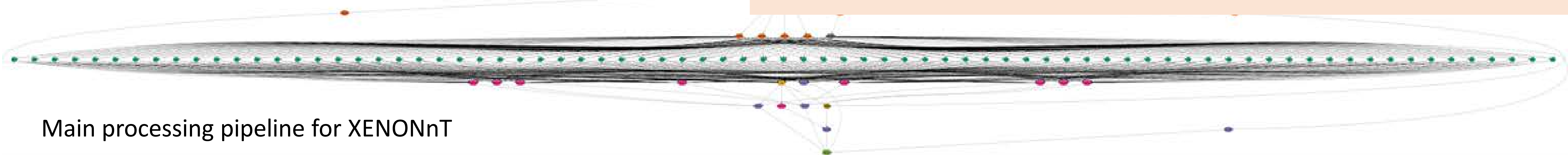
XENONnT - Dark Matter Search

Detector at Laboratori Nazionali del Gran Sass (LNGS) in Italy. Data is distributed world-wide with Rucio. Workflows execute across Open Science Grid (OSG) and European Grid Infrastructure (EGI)






```
------------------------------------------------------------------------------
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 for XENONnT

More Pegasus features…

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

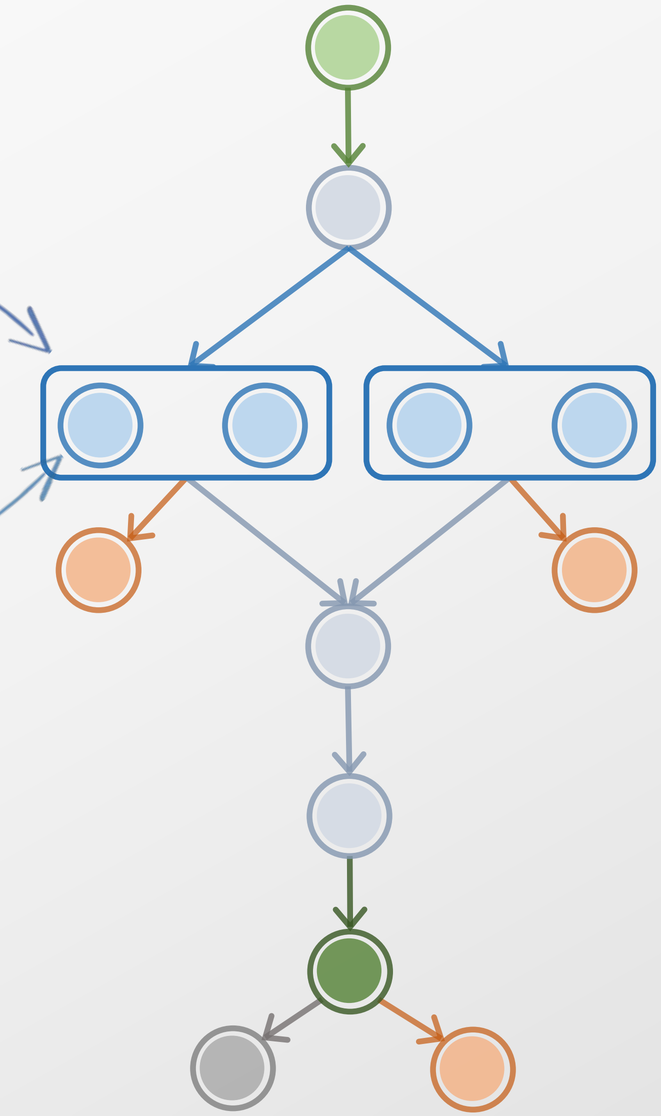
Performance, why not improve it?

clustered job

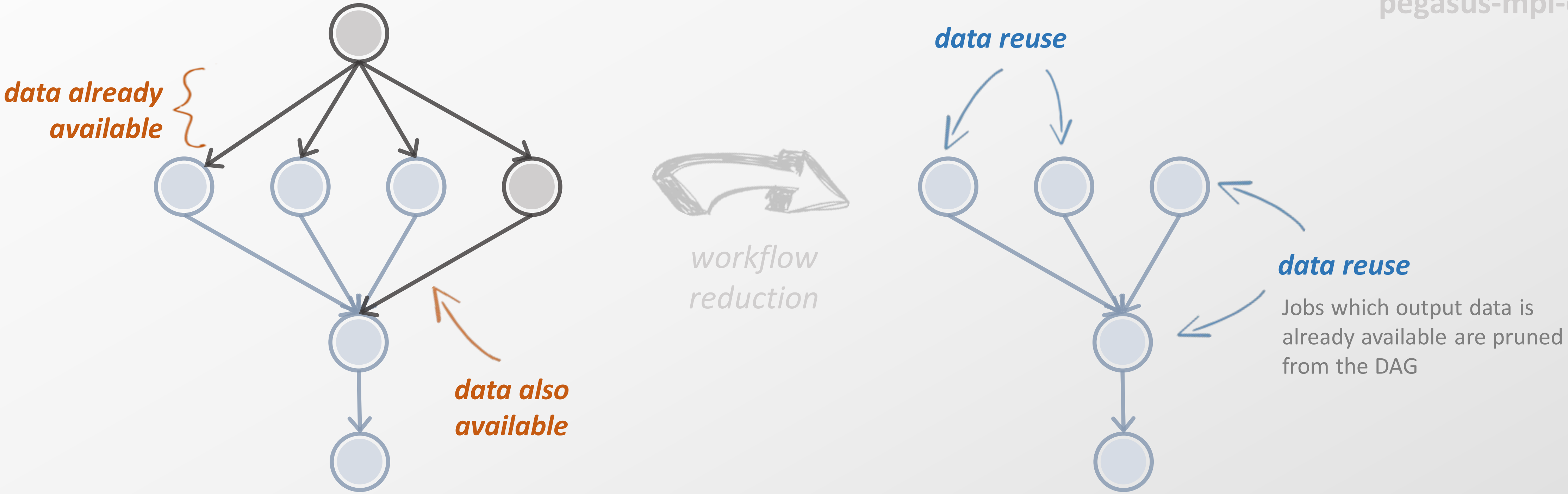
Groups small jobs together
to improve performance

task

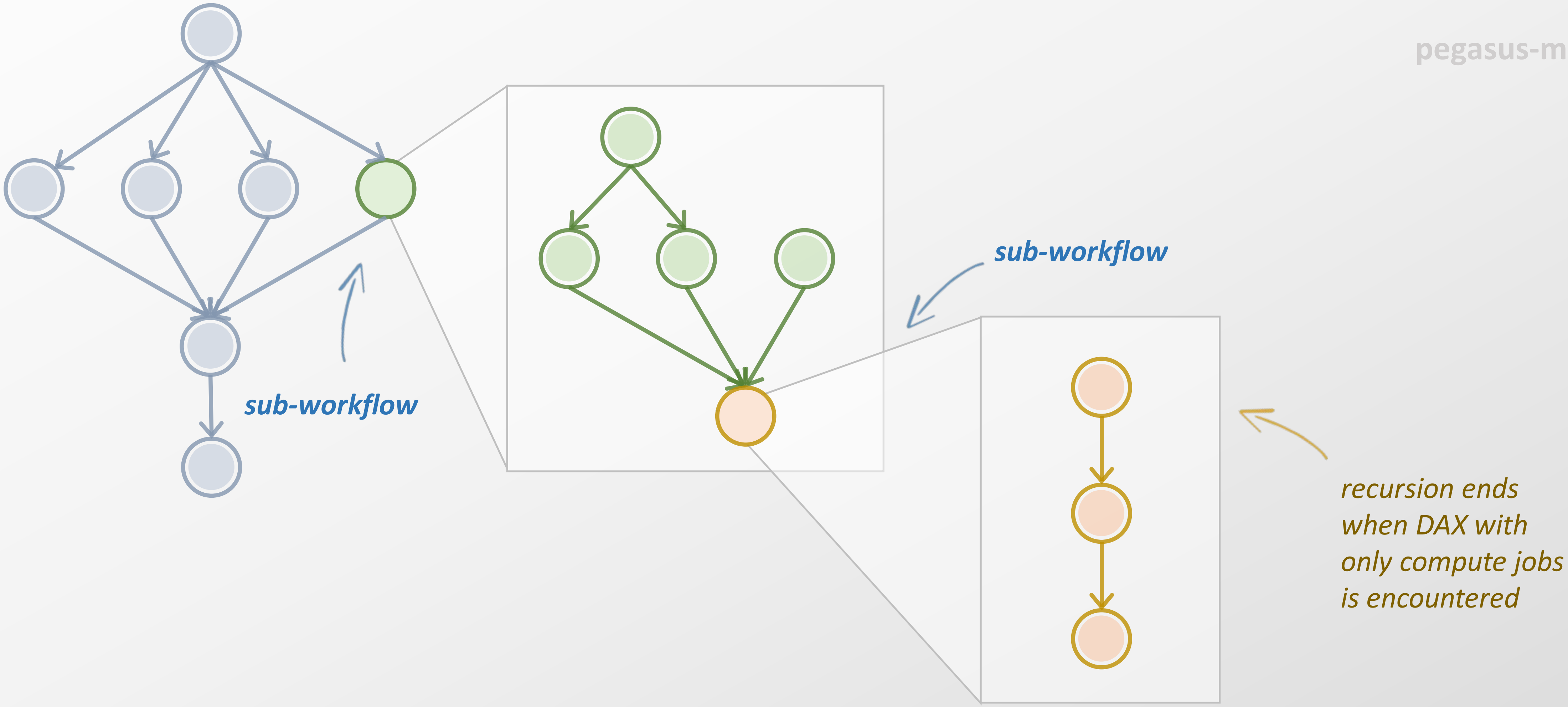
small granularity



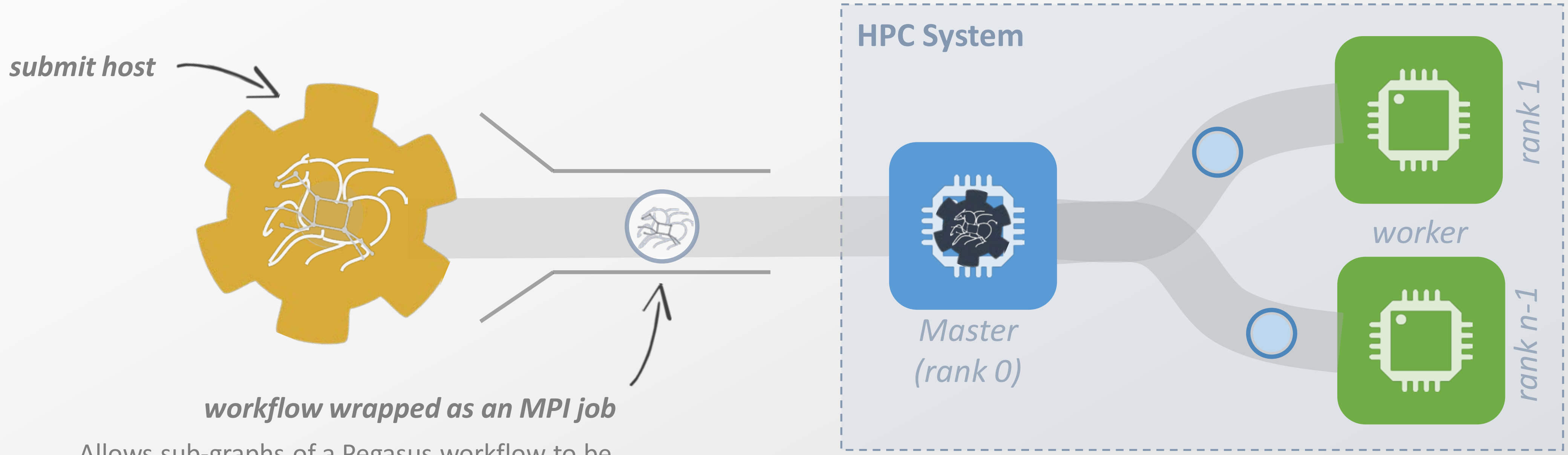
What about **data reuse**?



Pegasus also handles **large-scale workflows**



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

Pegasus in OpenShift

panorama360

USC Viterbi
School of Engineering
Department of Computer Science

- GitHub: https://github.com/Panorama360
- Website: https://panorama360.github.io

George Papadimitriou
Computer Science PhD Student
University of Southern California

email: georgpap@isi.edu

https://panorama360.github.io/

Acknowledgements

Special thanks to the OLCF people that helped us make this deployment happen !

Jason Kincl
kincljc@ornl.gov

Valentine Anantharaj
anantharajvg@ornl.gov

Jack Wells
wellsjc@ornl.gov

This work was funded by DOE contract number DESC0012636, ``Panorama---Predictive Modeling and Diagnostic Monitoring of Extreme Science Workflows'', and U.S. Department of Energy, Office of Science under contract DE-AC02-06CH11357.

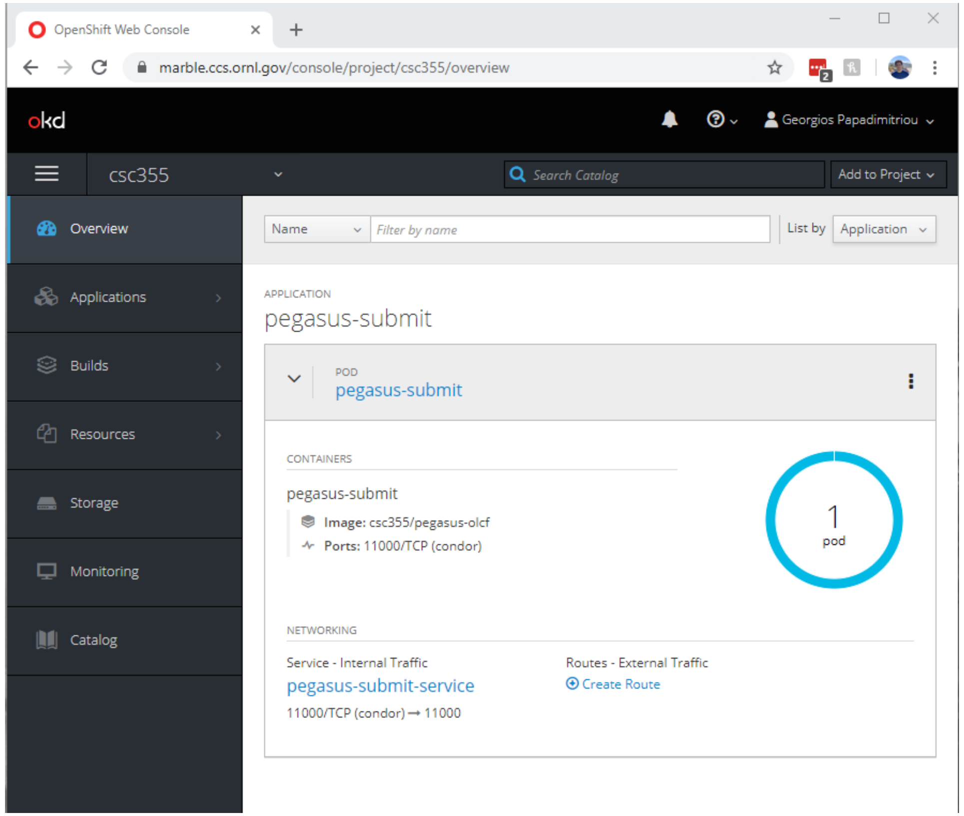
This work used resources of the Oak Ridge Leadership Computing Facility, which is a DOE Office of Science User Facility supported under Contract DE-AC05-00OR22725.

Kubernetes: Why it can be useful in HPC

- Running services on login nodes can be cumbersome (build from scratch, compile all dependences etc.) and sometimes prohibited by the system administrators.
- Maintaining an application/service up to day is easier
- **Assist workflow execution**
  - Create submission environments
  - Handle data movement and job submissions
  - Automation and Reproducibility
- **Create collaborative web portals**
  - Jupyter Notebooks
  - Workflow Design (e.g. Wings)
- **Streaming Data**
  - Consuming
  - Publishing

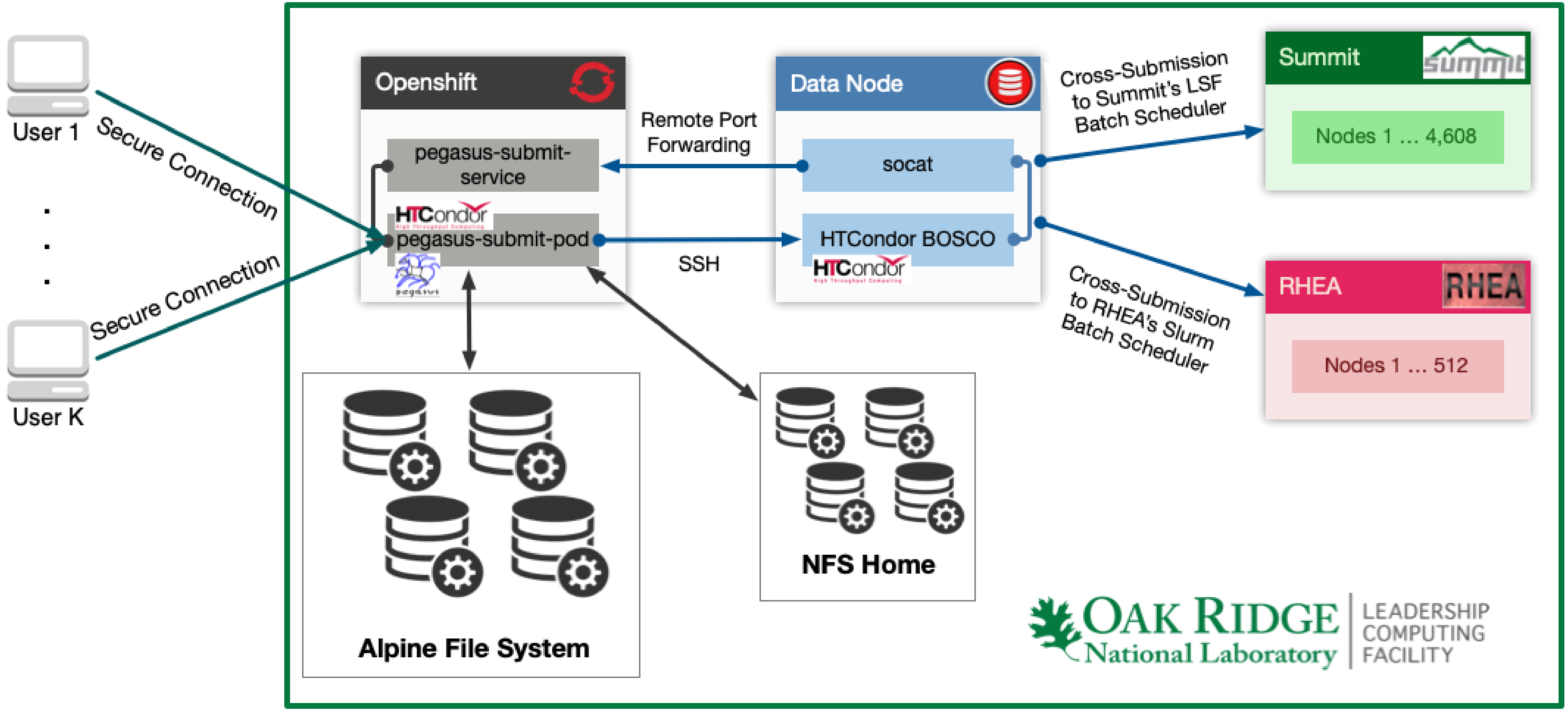
Kubernetes (OpenShift) at OLCF

- OLCF has deployed OpenShift, a distribution of Kubernetes developed by RedHat
- OpenShift provides a command line and a web interface to manage your Kubernetes objects (pods, deployments, services, storage etc.)
- OLCF's deployment has automation mechanisms that allow users to submit jobs to the batch system and access the shared file systems (NFS, GPFS)
- All containers run as an automation user that is tied to a project



Reference:
https://www.olcf.ornl.gov/wp-content/uploads/2017/11/2018UM-Day3-Kincl.pdf

Kubernetes (OpenShift) at OLCF: Pegasus Deployment



Kubernetes at OLCF: Pegasus Deployment - Advantages

- Pegasus workflow **environments** at OLCF have been **simplified**.
- Using the Kubernetes cluster at OLCF, we can deploy Pegasus submit nodes as services, within a few seconds.
- The deployment uses HTCondor's BOSCO SSH style submissions on the DTNs and achieves submissions to the SLURM and LSF batch schedulers.
- This approach allows a single workflow to be configured to use **all** of OLCF's resources. E.g. Execute transfers on the DTNs, run simulations and heavy processing on Summit and then do lightweight post processing steps on RHEA.

How to Deploy: Prerequisites

- Pegasus Kubernetes Templates for OLCF:
  - https://github.com/pegasus-isi/pegasus-olcf-kubernetes
- OpenShift's Origin Client:
  - https://github.com/openshift/origin/releases
- A working RSA Token to access OLCF's systems
- An automation user for OLCF's systems
- Allocation on OLCF's OpenShift Cluster (https://marble.ccs.ornl.gov)

How to Deploy: Pegasus - Kubernetes Templates

- **bootstrap.sh** Generates customized Dockerfile and Kubernetes pod and service specifications for your deployment.

- **Specs/pegasus-submit-build.yml** Contains Kubernetes build specification for the pegasus-olcf image.

- **Specs/pegasus-submit-service.yml** Contains Kubernetes service specification that can be used to spawn a Nodeport service that exposes the HTCondor Gridmanager Service running in your submit pod, to outside world.

- **Specs/pegasus-submit-pod.yml** Contains Kubernetes pod specification that can be used to spawn a pegasus/condor pod that has access to Summits's GPFS filesystem and its batch scheduler.

How to Deploy: Customize Templates

In **bootstrap.sh** update the section "ENV Variables For User and Group" with your automation user's name, id, group name, group id and the Gridmanager Service Port, which must be in **the range 30000-32767**.

Replace the highlighted text:

- **USER:** with the username of your automation user (eg. csc001_auser)
- **USER_ID:** with the user id of your automation user (eg. 20001)
- **USER_GROUP:** with the project name your automation user belongs to (eg. csc001)
- **USER_GROUP_ID:** with the project group id your automation user belongs to (eg. 10001)
- **GRIDMANAGER_SERVICE_PORT:** with the Kubernetes Nodeport port number the Gridmanager Service should use (eg. 32752)

```
#!/usr/bin/env bash

#### ENV Variables For Packages ####
PEGASUS_VERSION="pegasus-4.9.3dev"
PEGASUS_VERSION_NUM="4.9.3dev"
BOSCO_VERSION_NUM="1.2.12"

#### ENV Variables For User and Group ####
USER=""
USER_ID=""
USER_GROUP=""
USER_GROUP_ID=""
GRIDMANAGER_SERVICE_PORT=""
GRIDMANAGER_SERVICE_ADDRESS="${USER_GROUP}.m


#### Don't edit this part ####

```

Execute Script:

```
$ bash bootstrap.sh
```

Pegasus in OpenShift: Status

G. Papadimitriou, K. Vahi, J. Kincl, V. Anantharaj, E. Deelman, and J. Wells, "Workflow Submit Nodes as a Service on Leadership Class Systems," in *Proceedings of the Practice and Experience in Advanced Research Computing*, New York, NY, USA, 2020. (Funding Acknowledgments: DOE DESC0012636)

Might seem complicated, but only 6 easy steps:

https://pegasus.isi.edu/tutorial/summit/

Pegasus est. 2001

Automate, recover, and debug scientific computations.

Get Started

Pegasus Website

https://pegasus.isi.edu

Users Mailing List

pegasus-users@isi.edu

Support

pegasus-support@isi.edu

Pegasus Online Office Hours

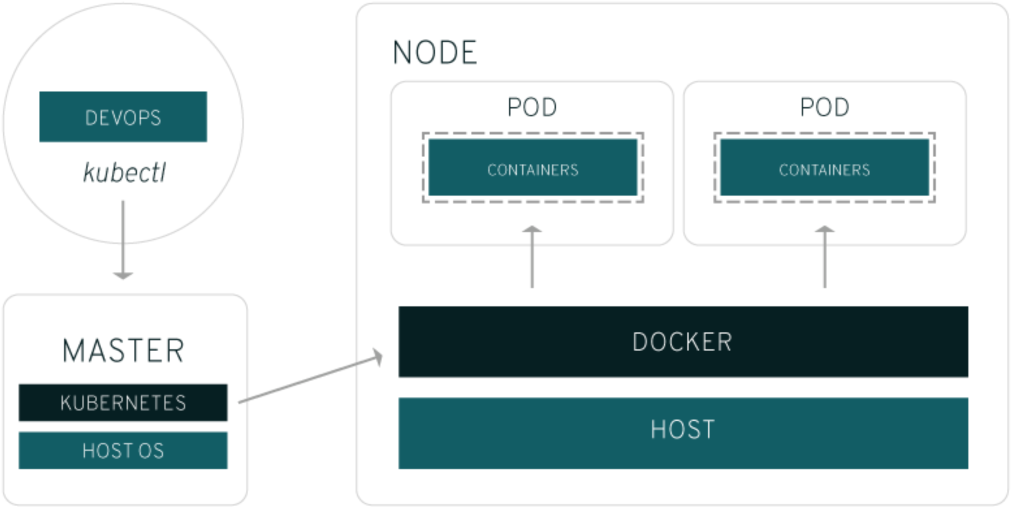
https://pegasus.isi.edu/blog/online-pegasus-office-hours/

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

Extra Slides

Kubernetes: Brief Overview

- **Kubernetes** is an open-source platform for running and coordinating containerized application across a cluster of machines.
- It can be useful for:
  - Orchestrating containers across multiple hosts
  - Control and automate deployments
  - Scale containerized applications on the fly
  - And more…



Reference:
https://www.redhat.com/en/topics/containers/what-is-kubernetes

- **Key objects** in the Kubernetes architecture are:
  - Master: Controls Kubernetes nodes – assign tasks
  - Node: Perform the assigned tasks
  - Pod: A group of one or more containers deployed on a single node
  - Replication Controller: Controls how many copies of a pod should be running
  - Service: Allow pods to be reached from the outside world
  - Kubelet: Runs on the nodes and starts the defined containers

Kubernetes: Configuring Objects

- Within Kubernetes, specification files describe the applications, services and objects being deployed

- Specification files can be written in YAML and JSON formats and can be used to
  - Deploy Pods
  - Create and mount volumes
  - Expose services etc.

pods/resource/memory-request-limit.yaml

```yaml
apiVersion: v1
kind: Pod
metadata:
  name: memory-demo
  namespace: mem-example
spec:
  containers:
  - name: memory-demo-ctr
    image: polinux/stress
    resources:
      limits:
        memory: "200Mi"
      requests:
        memory: "100Mi"
    command: ["stress"]
    args: ["--vm", "1", "--vm-bytes", "150M", "--vm-hang", "1"]
```

Reference:
https://kubernetes.io/docs/tasks/configure-pod-container/

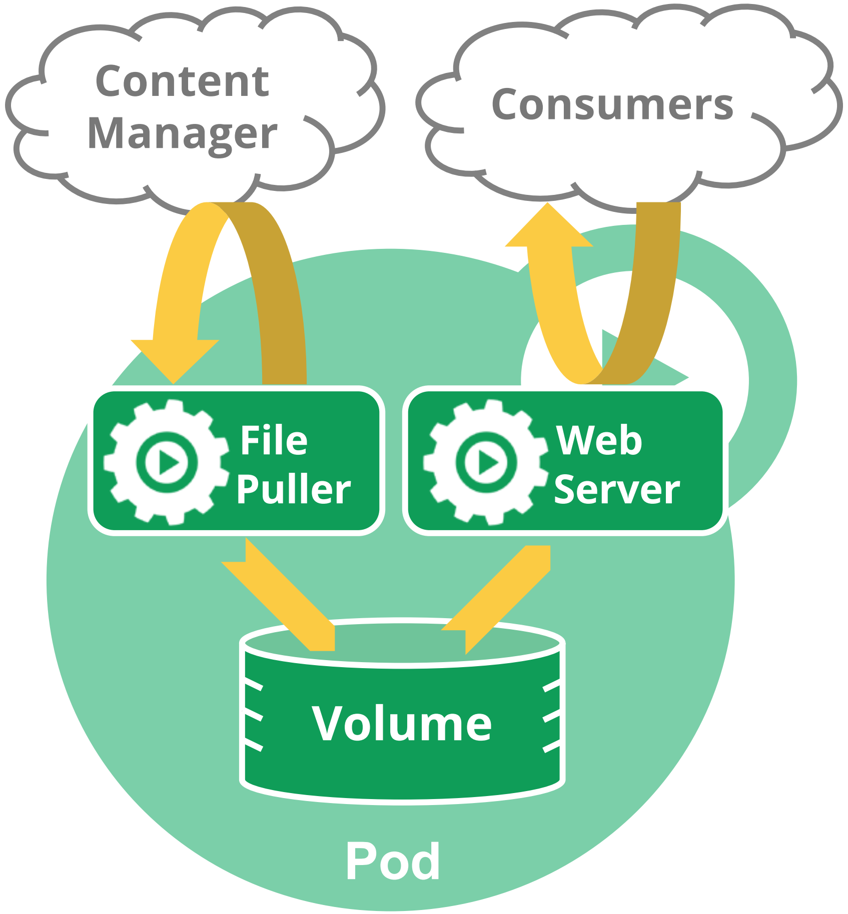
Kubernetes: Pods

- A **Pod** is the basic execution unit of a Kubernetes application
- Pods represent processes running on the cluster
- One can have one or multiple containers running within a Pod.

- **Networking:** Each Pod is assigned a unique IP address within the cluster

- **Storage:** A Pod can specify a set of shared storage Volumes. Volumes persist data and allow Pods to maintain state between restarts.

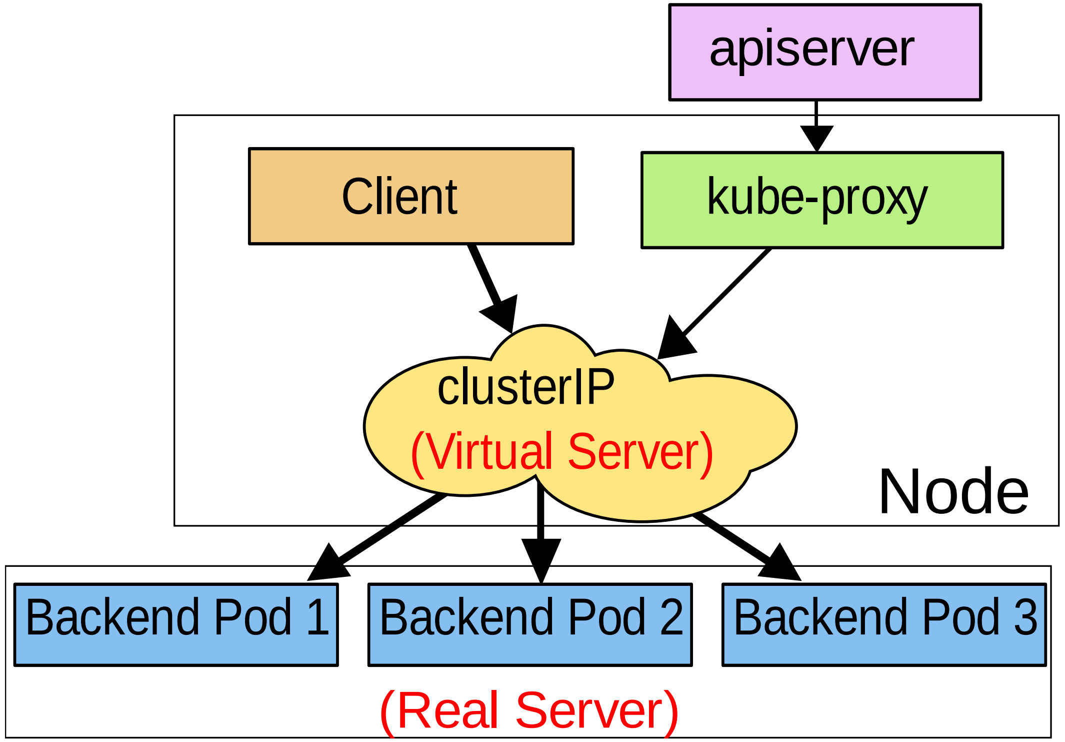
- **Lifecycle:** A Pod starts running on its assigned cluster-node until the container(s) exit or it is removed for some other reason (e.g. user deletes it).



References:
https://kubernetes.io/docs/concepts/workloads/pods/pod-overview/
https://kubernetes.io/docs/concepts/workloads/pods/pod/
https://kubernetes.io/docs/concepts/workloads/pods/pod-lifecycle/
https://kubernetes.io/docs/concepts/storage/volumes/

Kubernetes: Services

- A **Service** provides an abstract way to expose an application running on a set of Pods as network service to the rest of the world
- Since Pods are ephemeral, services allow users to access the backend applications via a common way
- Service types are:
  - ClusterIP: Exposes the service on a cluster-internal IP
  - NodePort: Exposes the service on each Node's IP at a static port
  - LoadBalancer: Exposes the service externally and loadbalances it
  - ExternalName: Maps the service to a name, returns a CNAME record



Reference:
https://kubernetes.io/docs/concepts/services-networking/service/

How to Deploy

We will follow the tutorial: https://pegasus.isi.edu/tutorial/summit/tutorial_setup.php

How to Deploy: Useful Origin Client Commands

- oc login: acquires an access token, authenticate against a cluster
- oc status: returns/prints the status of your deployments
- oc describe: shows details of a specific resource
- oc create: creates a Kubernetes resource from specification
- oc start-build: initiates the creation of a container image
- oc logs: returns/prints the Kubernetes log for a resource
- oc exec: executes a command in a container
- oc delete: deletes a resource

How to Deploy: Acquire an Access Token (Step 1)

```
$ oc login -u YOUR_USERNAME https://marble.ccs.ornl.gov/

Username: olcf_user
Password:
Login successful.

You have one project on this server: "csc001"

Using project "csc001".
```

How to Deploy: Build the Container Image (Step 2)

Create a new build and build the image:

1

```
$ oc create -f Specs/pegasus-submit-build.yml
buildconfig.build.openshift.io/pegasus-olcf created
```

2

```
$ oc start-build pegasus-olcf --from-file=Docker/Dockerfile
Uploading file "Docker/Dockerfile" as binary input for the build ...

Uploading finished
build.build.openshift.io/pegasus-olcf-1 started
```

How to Deploy: Build the Container Image (Step 2)

Trace the progress of the build:

```
$ oc logs -f build/pegasus-olcf-1

...
Step 30/30 : LABEL "io.openshift.build.name" "pegasus-olcf-1" "io.openshift.l
 ---> Using cache
 ---> ed0f4341ff43
Successfully built ed0f4341ff43
Pushing image docker-registry.default.svc:5000/cscXXX/pegasus-olcf:latest ..
Pushed 2/14 layers, 14% complete
Pushed 3/14 layers, 21% complete
Pushed 4/14 layers, 29% complete
Pushed 5/14 layers, 36% complete
Pushed 6/14 layers, 43% complete
Pushed 7/14 layers, 50% complete
Pushed 8/14 layers, 57% complete
Pushed 9/14 layers, 64% complete
Pushed 10/14 layers, 71% complete
Pushed 11/14 layers, 79% complete
Pushed 12/14 layers, 86% complete
Pushed 13/14 layers, 93% complete
Pushed 14/14 layers, 100% complete
Push successful
```

How to Deploy: Start the Kubernetes Service (Step 3)

Start a Kubernetes Service that will expose your pod's services:

```
$ oc create -f Specs/pegasus-submit-service.yml
service/pegasus-submit-service created
```

Note: In case this step fails, go back to the bootstrap.sh change the service port number and execute it again.
Proceed from this step, <u>there is no need to rebuild the container.</u>

How to Deploy: Start the Pegasus Pod (Step 4)

Start a Kubernetes Pod with Pegasus and HTCondor:

```
$ oc create -f Specs/pegasus-submit-pod.yml

pod/pegasus-submit created
```

Logon to the Pod:

```
$  oc exec -it pegasus-submit /bin/bash
[csc001_auser@pegasus-submit /]$
```

How to Deploy: Configuring for Batch Submissions (Step 5)

If this is the first time you bringing up the Pegasus container in Kubernetes we need to configure it for batch submissions.

In the shell you got on the previous step execute:

```
$ bash /opt/remote_bosco_setup.sh
```

Note: This script installs some additional files needed to operate on OLCF, and prepares the environment on the DTNs, by installing BOSCO.

How to Deploy: Check the status of the deployment

If all goes well you should see something similar to this in your terminal:

```
$oc status
In project cscXXX on server https://marble.ccs.ornl.gov:443

svc/pegasus-submit-service (all nodes):32753 -> 11000
  pod/pegasus-submit runs docker-registry.default.svc:5000/cscXXX/pegasus-olcf:latest

bc/pegasus-olcf docker builds Dockerfile on istag/centos:centos7
  -> istag/pegasus-olcf:latest
  build #1 succeeded 15 minutes ago


1 info identified, use 'oc status --suggest' to see details.
```

How to Deploy: Deleting the Pod and the Service

Deleting the Pod:

```
$  oc delete pod pegasus-submit
```

Deleting the Service:

```
$ oc delete svc pegasus-submit-service
```

Deleting the container image:

```
$  oc delete bc pegasus-olcf
```