Delivering easy-to-use frameworks to empower data-driven research

Dr. Rosa Filgueira, University of Edinburgh, EPCC

Background

- PhD Computer Science – Madrid/Carlos III
  - *Dynamic optimization techniques to enhance scalability and performance of MPI-based application*

- 5 years as a Postdoc – University of Edinburgh/DIR group
- 2 years as a Senior Data scientist at BGS

Currently – EPCC at the University of Edinburgh

- Data Architect activities across different domains/ Projects/
  - Scalability and performance of applications executed on HPC and Cloud resources
  - Scientific workflows, data-frameworks, containers and reproducibility tools, etc
  - Research activities

https://www.rosafilgueira.com/

epcc

Introduction

Big Data Sciences Era, Data Intensive Computing applications

Scientific fields →

Data-driven

Astronomy, Geosciences, Meteorology,, Bioinformatics

Common points

- Big complex data sets
- Need to be analysed
- Numerous software tools
- Data transformation and visualisation

Introduction

Big Data Sciences Era, Data Intensive Computing applications

Data analytic frameworks and Computing environments

Automatically handle large amount of data

Computational steps:

- Speed
- Scalability
- Flexibility
- Robustly

Delivering easy-to-use frameworks to empower data-driven research

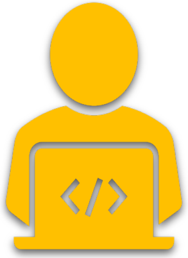
Seismology

Scientific Workflows:

dispel4py

Pegasus

CWL



Computing environments:

HPC Clusters

Cloud

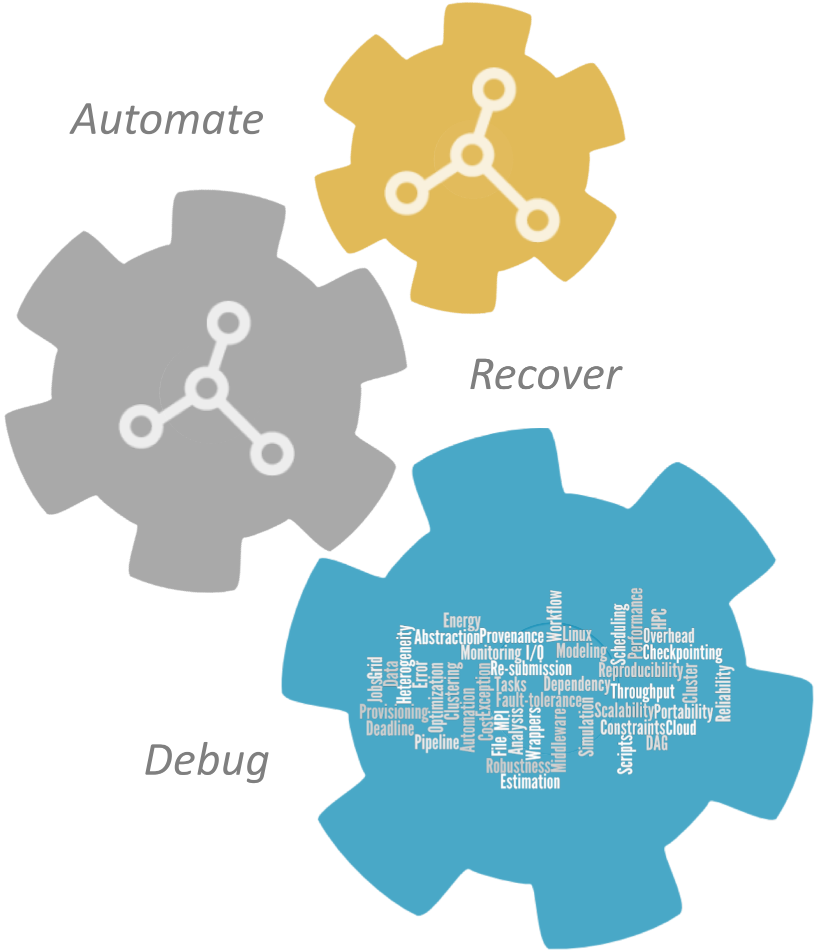
Why Scientific Workflows ?

General Features

Abstraction, scientists can focus on their research and not computation management

Easy composition and execution

Enables parallel, distributed computations



Different types

abstract vs. concrete

task-flow vs. data-flow

files vs. stream-based

Workflow Management Systems (WMS)

Provide tools to generate the scientific workflow

WORKFLOW MANAGEMENT SYSTEMS

Taverna
https://taverna.incubator.apache.org

Kepler
https://kepler-project.org

Makeflow
http://ccl.cse.nd.edu/software/makeflow

dispel4py
http://dispel4py.org

Nextflow
https://www.nextflow.io

KNIME
https://www.knime.org

VisTrails
http://vistrails.org

FireWorks
https://pythonhosted.org/FireWorks

Swift
http://swift-lang.org

Pegasus
http://pegasus.isi.edu

Asterism Framework

Easy to understood, platform-independent, open-source

Simplifies the development applications running across multiple heterogeneous

How ?

Combining the strengths of

Traditional WMS
Pegasus

Ewa Deelman, ISI-USC,US
Rafael Ferreira da Silva, ISI-USC, US

New stream-based data-flow systems
dispel4py

Rosa Filgueira, BGS-NERC, UK
Amrey Krause, EPCC-UoE, UK
Malcolm Atkinson, Informatics-UoE, UK

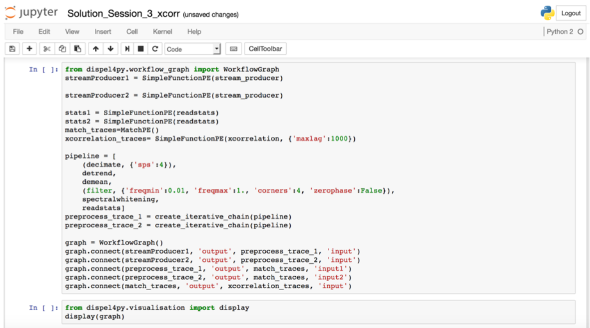
dispel4py parallel stream-based dataflow system

Automation

Automates pipeline executions
Concurrent, distributed computations
Stream-based model

Workflow Composition

Python Library
Groupings
Jupyter



High-level stream-based data pipeline
~ Apache Beam

Mapping

Sequential
Multiprocessing
MPI
Apache Storm
Apache Spark (Prototype)

Optimisation

Multiple streams
Avoids I/O operations

Key-features: Automatic parallelization/mappings, concurrent & stream-based

https://github.com/dispel4py/dispel4py

dispel4py parallel stream-based dataflow system

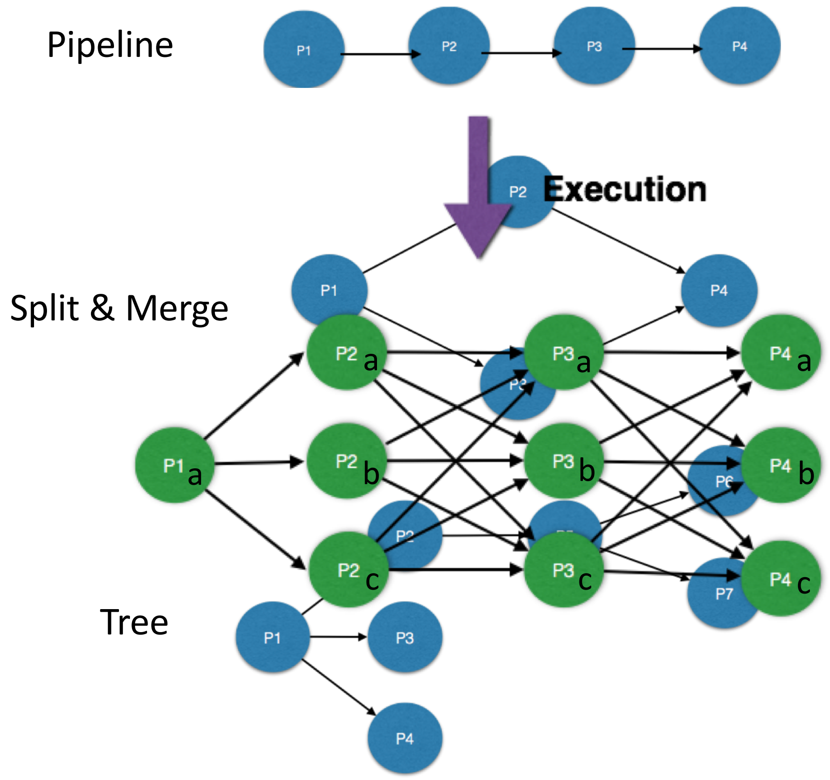
Graph

- Connections among PES
- Abstract workflow

Instance

- Executable copy of a PE that runs in a process
- Each PE is translated into one or more instances in run-time
- dispel4py does it for you
- Concrete workflow

+ Example of graphs



4 PEs & 10 processes

Pegasus workflow system

Automation

Automates pipeline executions
Parallel, distributed computations
Automatically executes data transfers
Heterogeneous resources
Task-oriented model
Application is seen as a black box

Debugging

Workflow execution and job performance metrics
Real-time monitoring, graphs, provenance

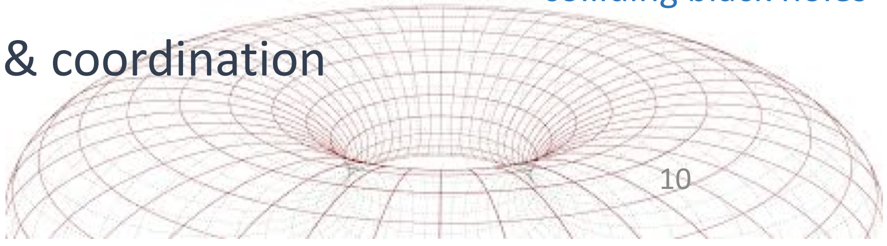
Recovery

Job failure detection
Checkpoint Files
Job Retry
Rescue DAGs

Optimisation

Job clustering
Data cleanup

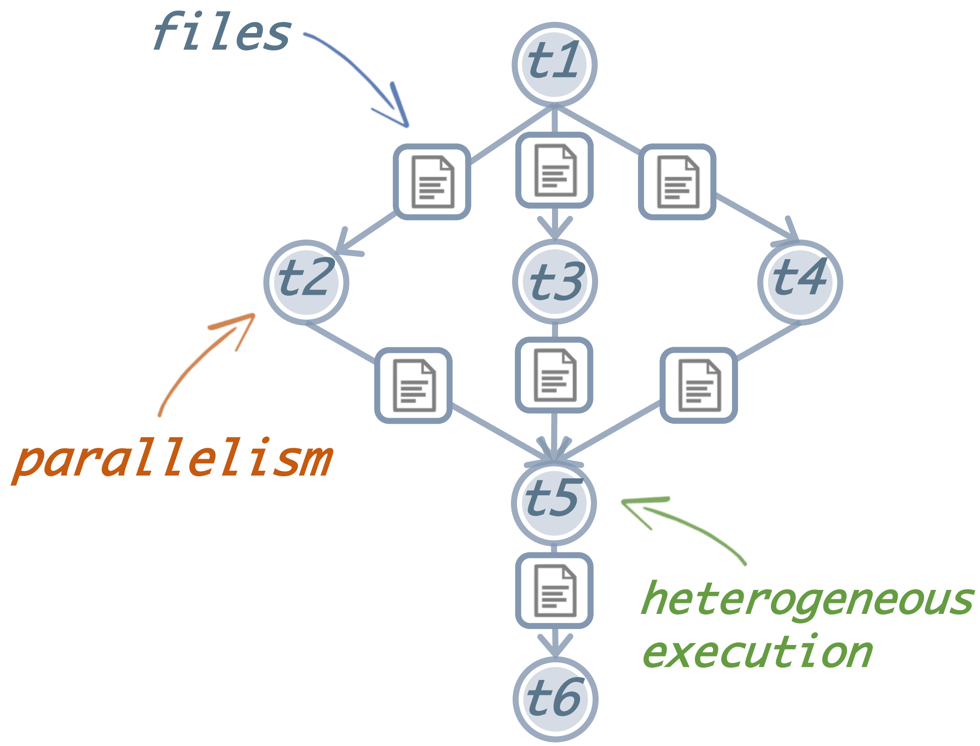
LIGO project:
First **detection of gravitational waves** from colliding black holes



Key-features: Automatic data movement, cleanup, heterogeneous & coordination

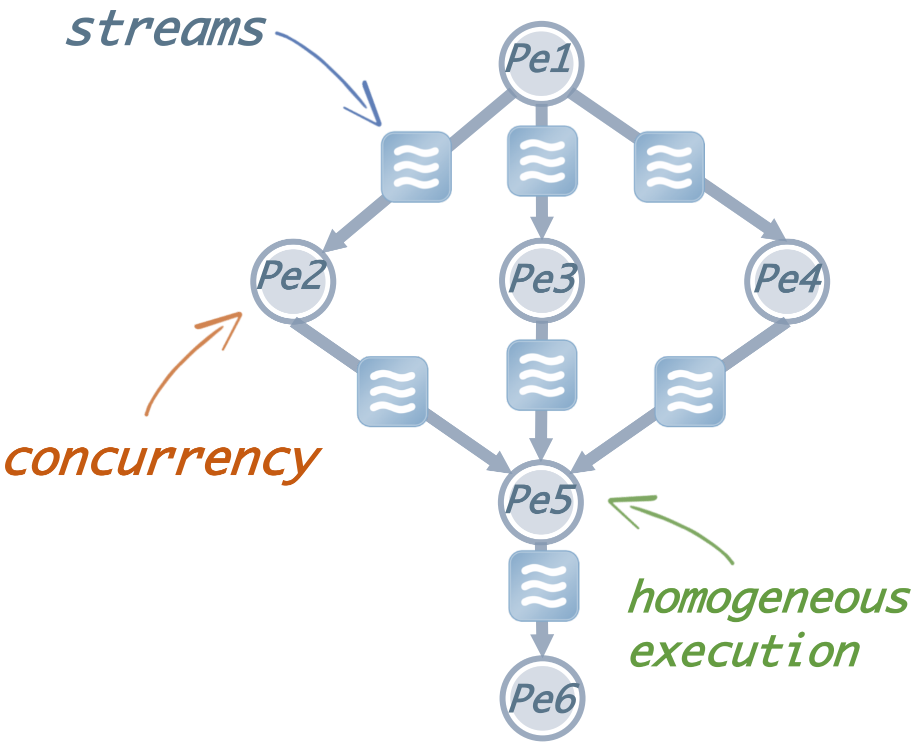
http://pegasus.isi.edu

Complementary systems



Task-flow

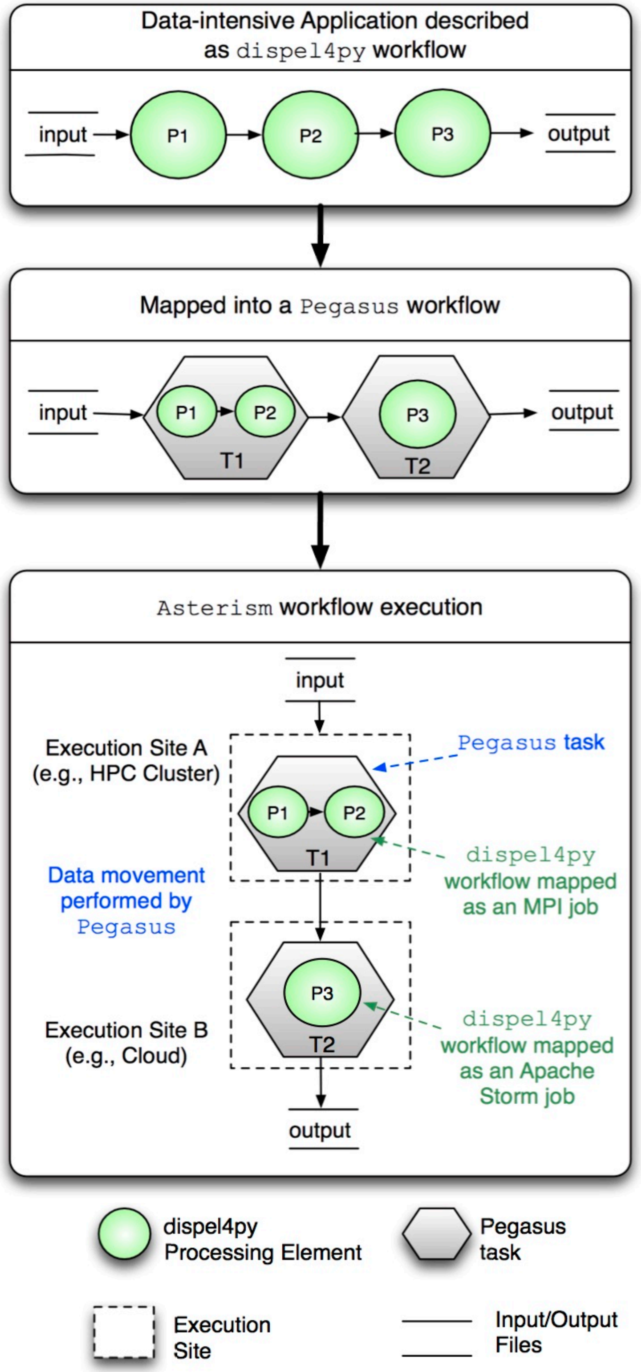
Task: Executable (black box)



Data-flow

Processing Elements: python objects

dispel4py + Pegasus = ASTERISM

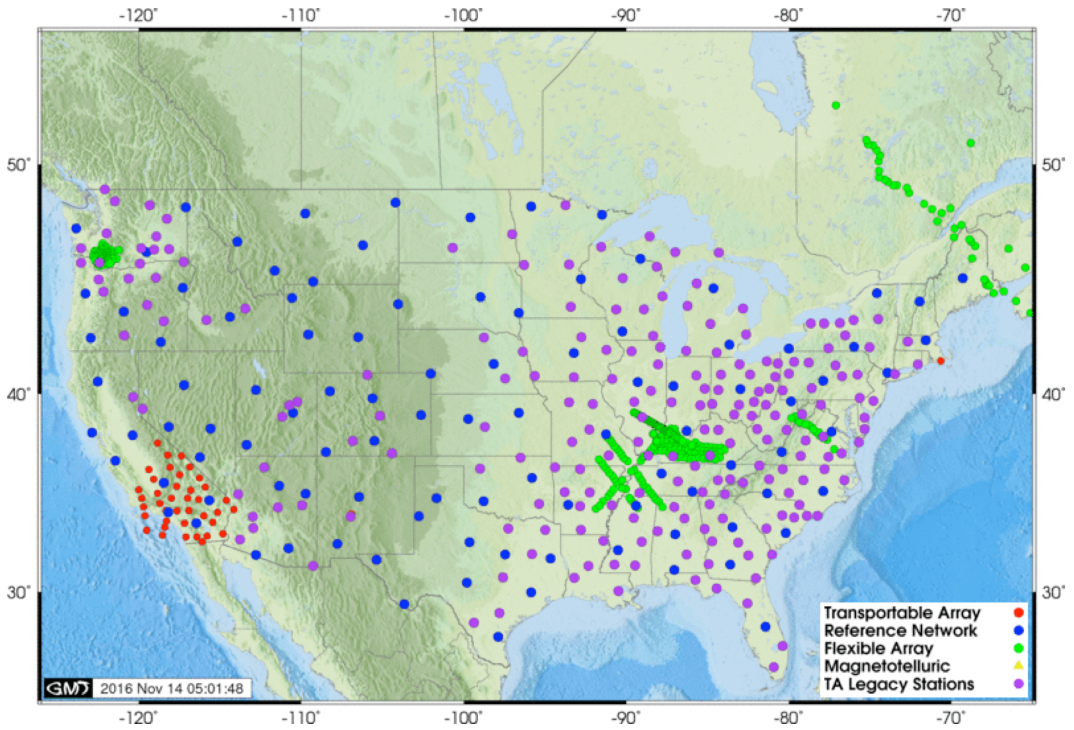


dispelpy to represent different parts of applications

Pegasus to distribute and execute each dispel4py workflow

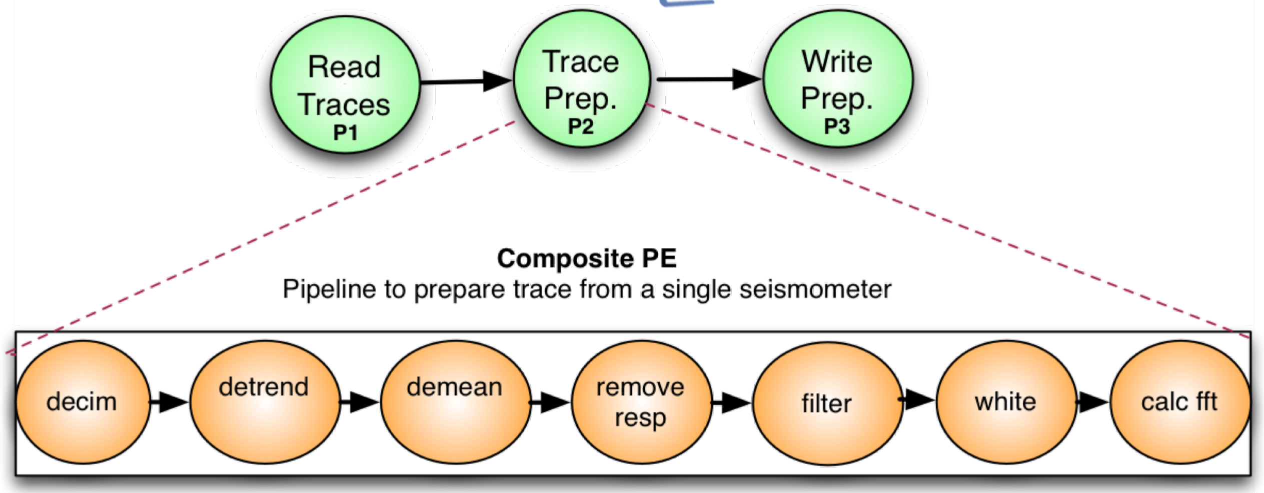
epcc

Seismic Ambient Noise Cross-Correlation



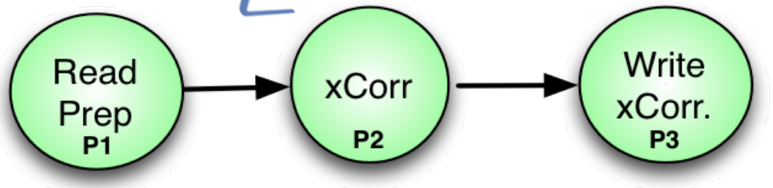
IRIS USA TA array – 1000 stations- 4999500 cross-correlations per hour

Phase 1 dispel4py workflow



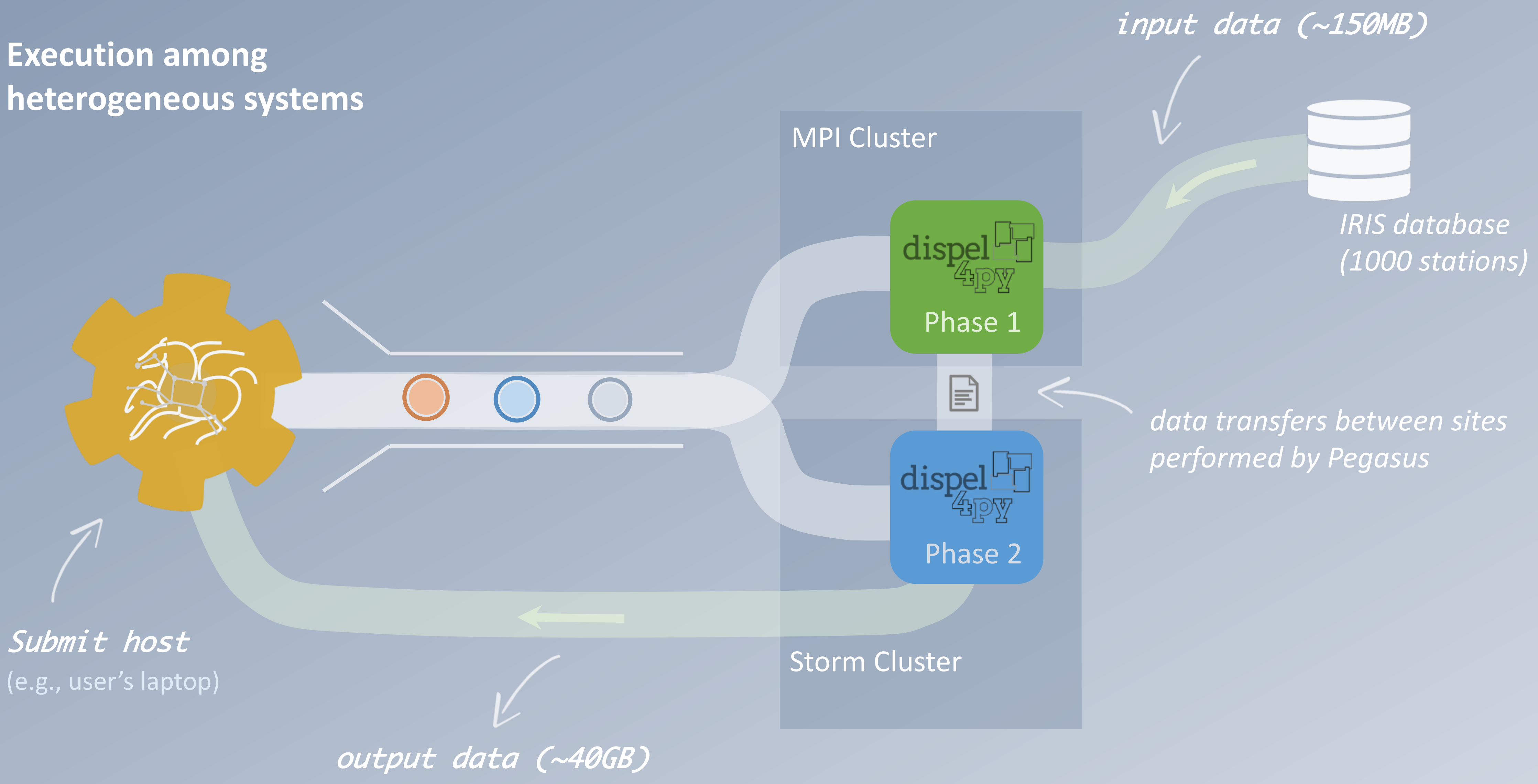
Preprocesses (Phase 1) and cross-correlates traces (Phase 2) from multiple seismic stations

Phase 2 dispel4py workflow



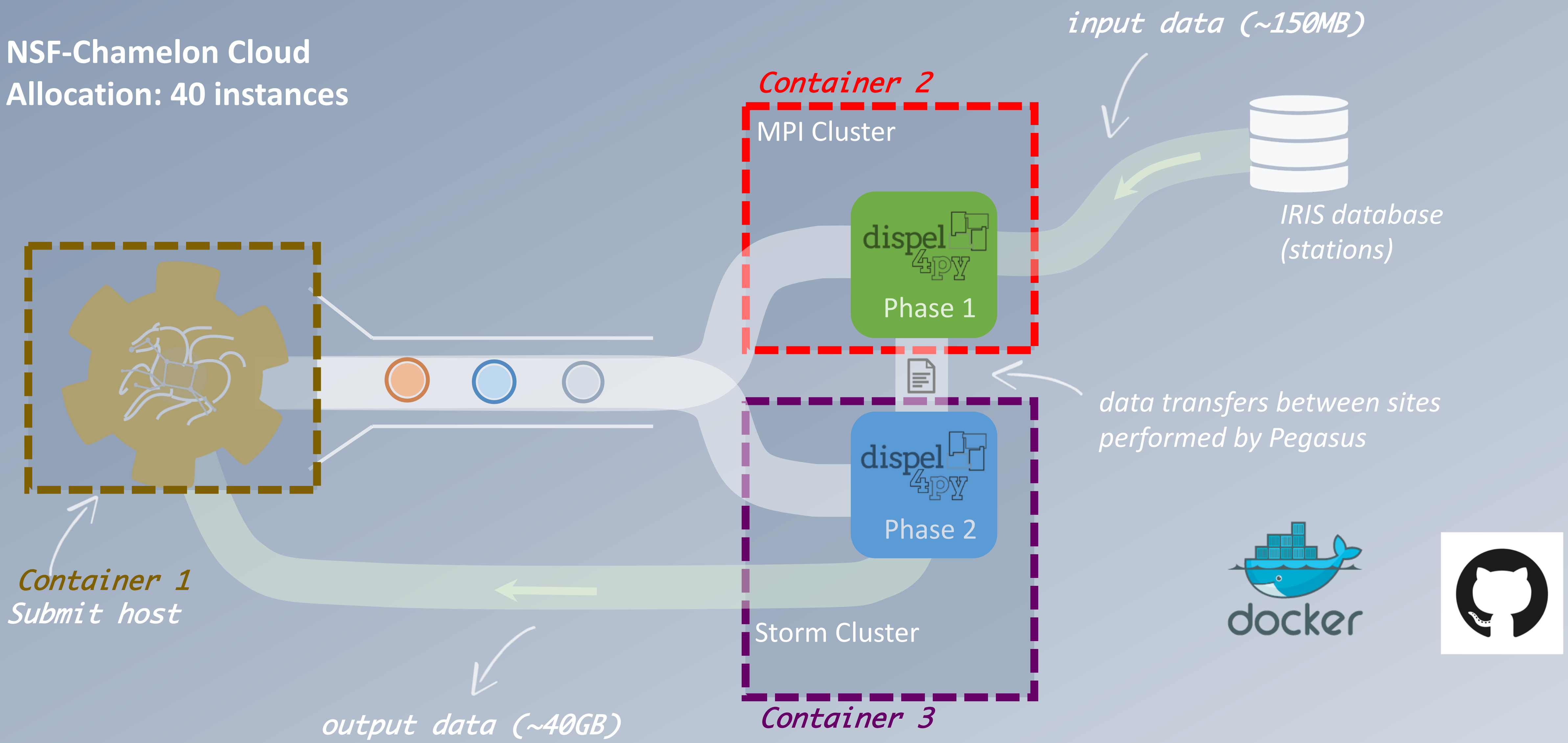
VERCE project both phases on the same HPC resource

Evaluation- Seismic Ambient Noise Cross-Correlation



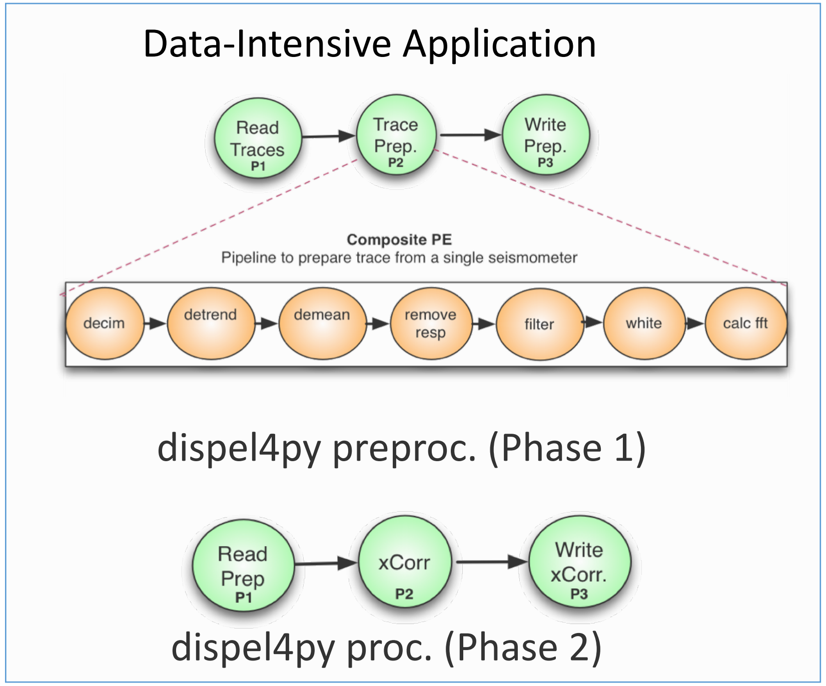
Evaluation- Seismic Ambient Noise Cross-Correlation

NSF-Chamelon Cloud
Allocation: 40 instances



https://github.com/dispel4py/pegasus_dispel4py

Reminder



Container 2

MPI Cluster
Phase 1

Container 1

Submit
Site

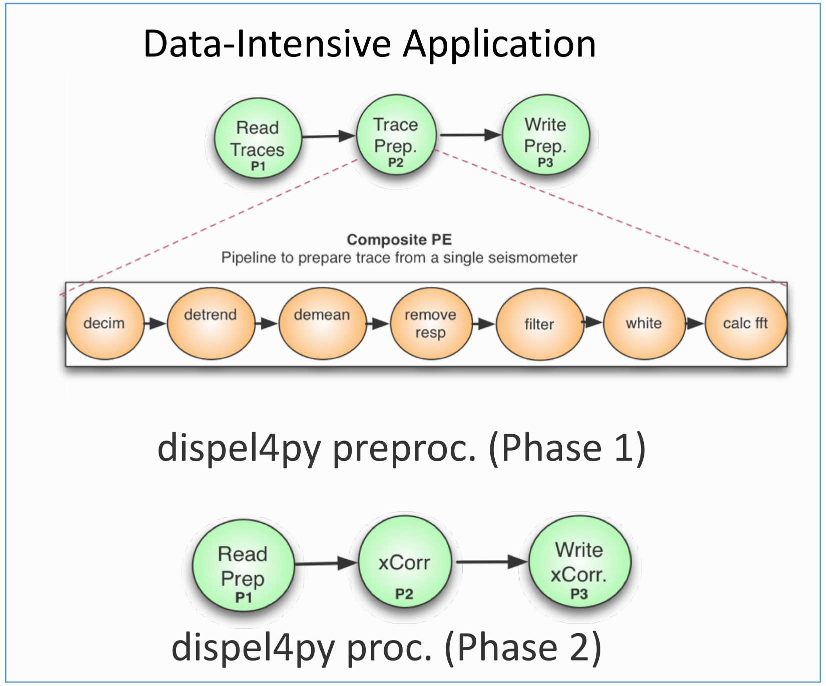
Storm Cluster
Phase2

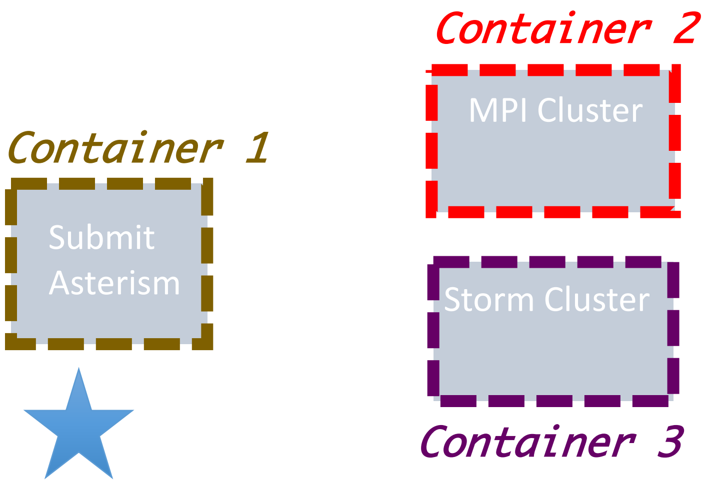
Container 3

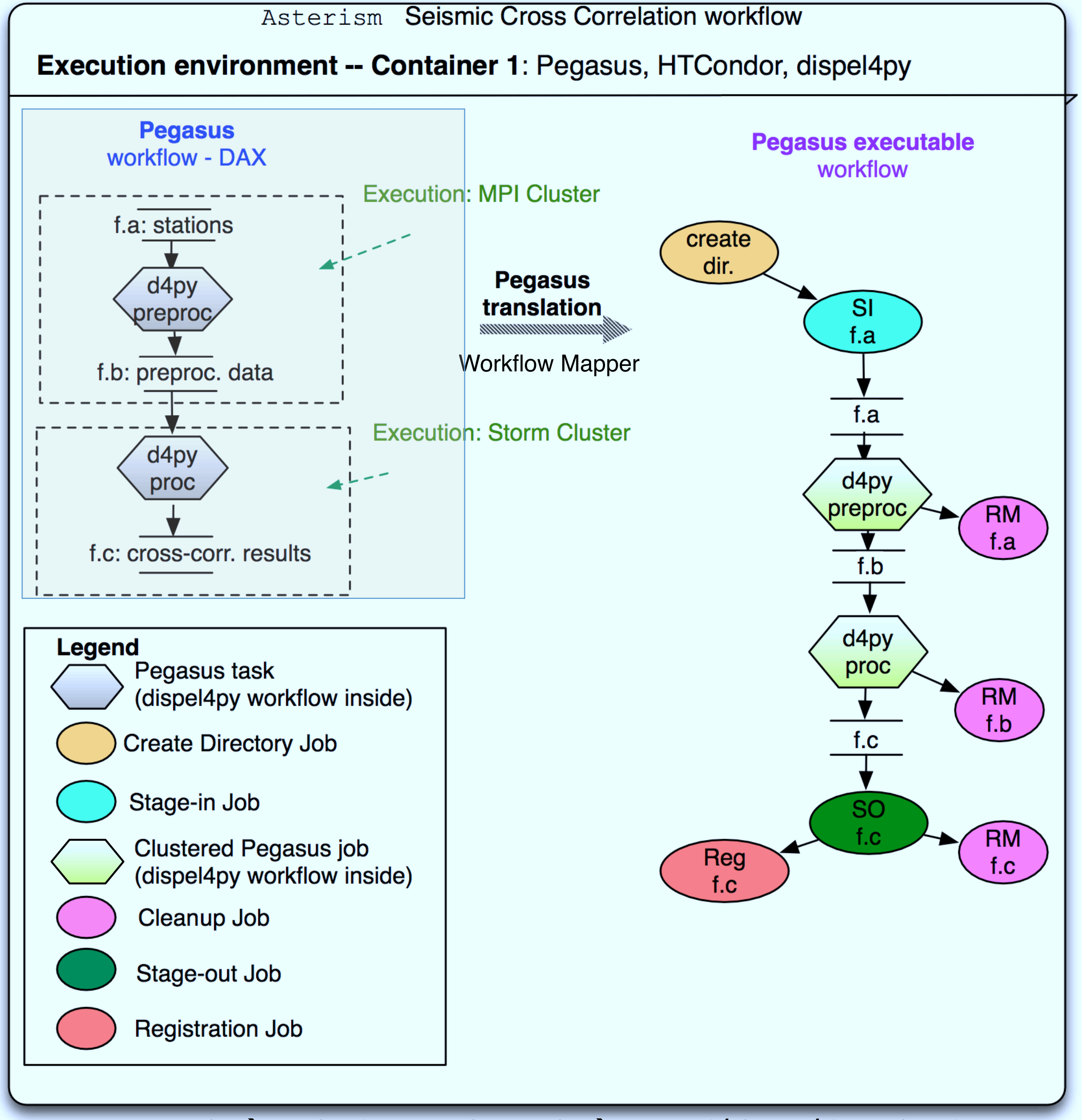
NSF-
Chamelon
cloud
1 node –
40 cores

Dockerfiles to configure containers **images** → Stored in our **GitHub** → Linked to **DockerHub** → stored/share/download images

Reminder





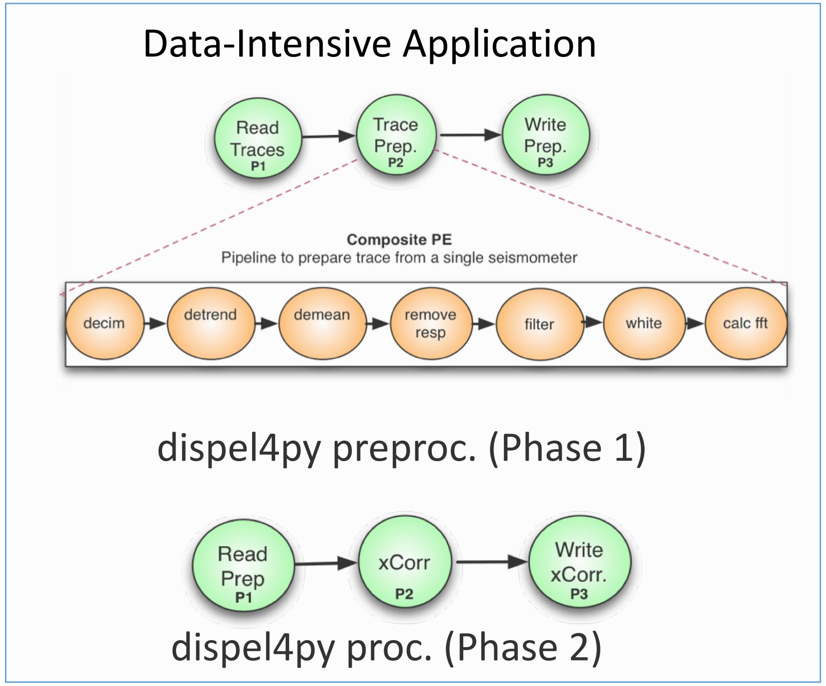


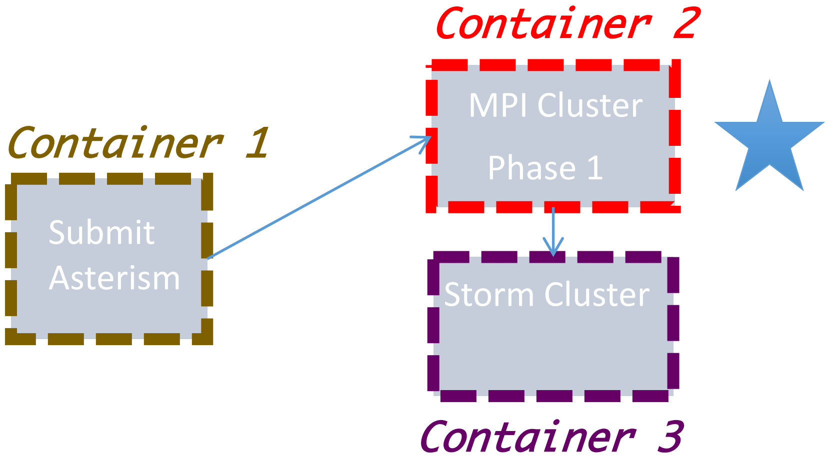
1 instance as Container 1

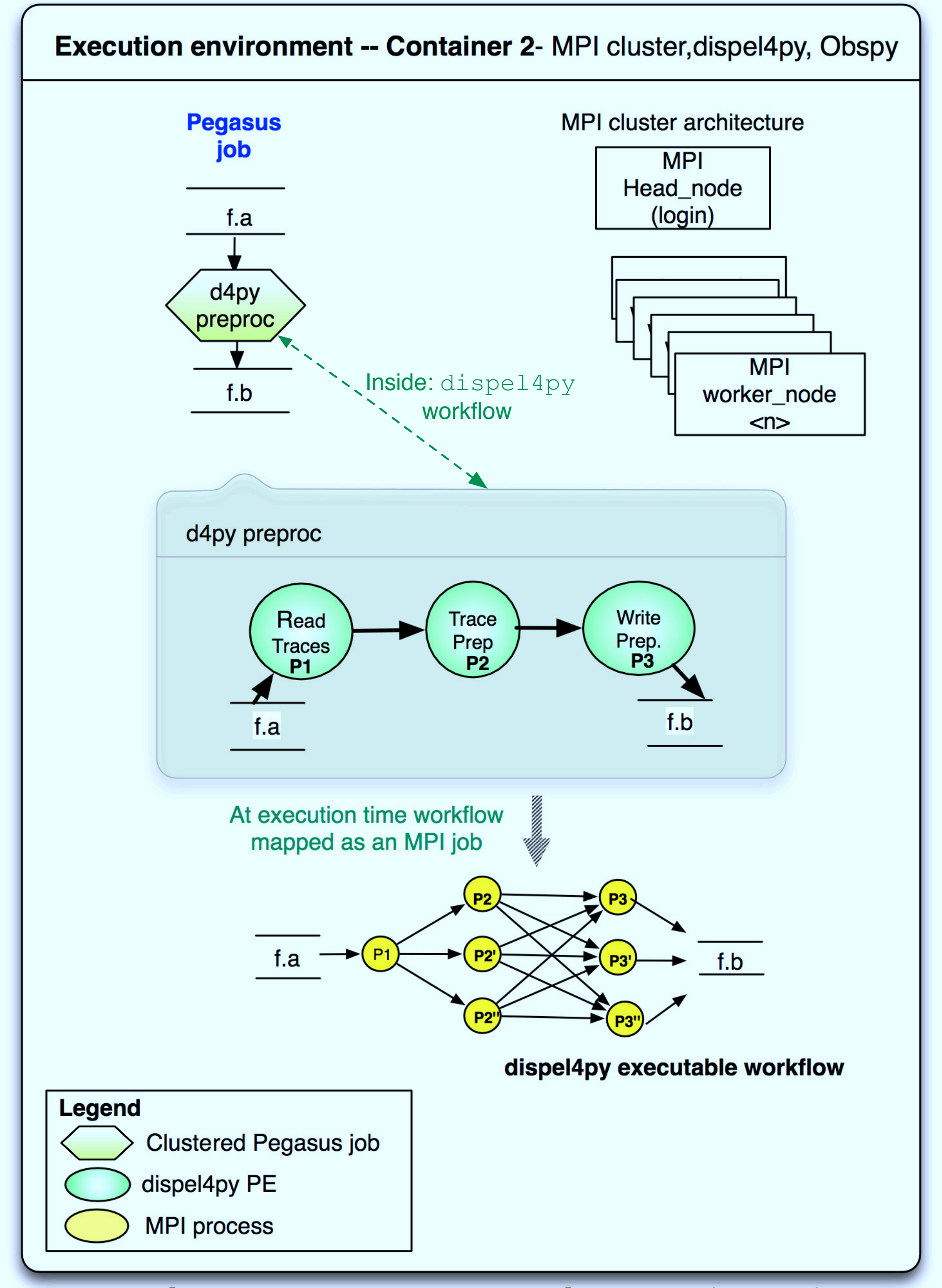
NSF-Chamelon cloud
1 node – 40 cores

Dockerfiles to configure containers **images** → Stored in our **GitHub** → Linked to **DockerHub** → stored/share/download images

Reminder







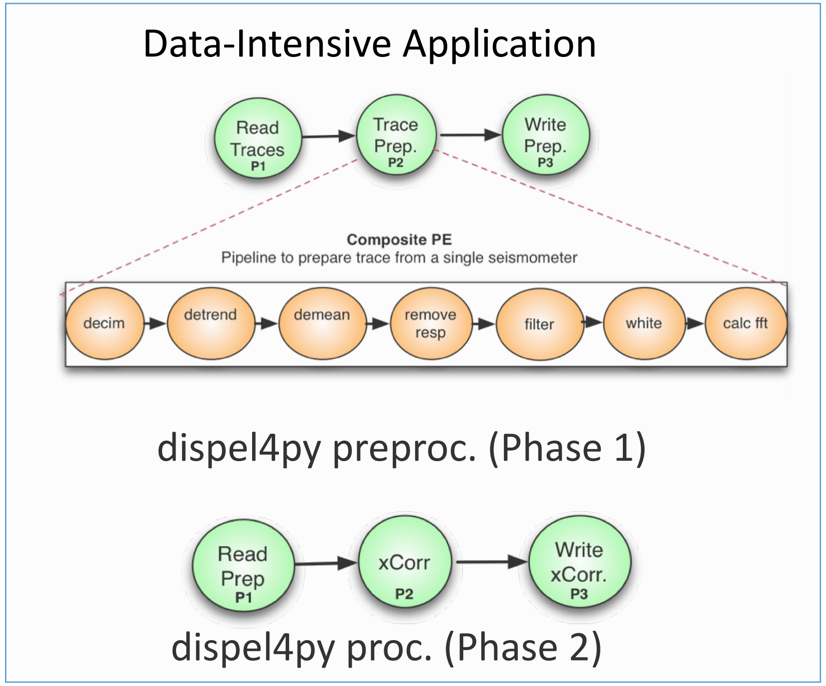
1 instance as Container 2 (MPI head node)

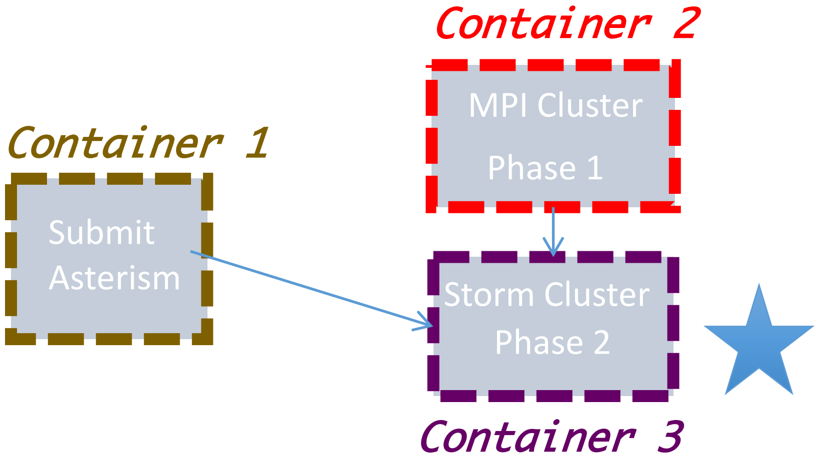
16 instances as Container 2 (MPI workers)

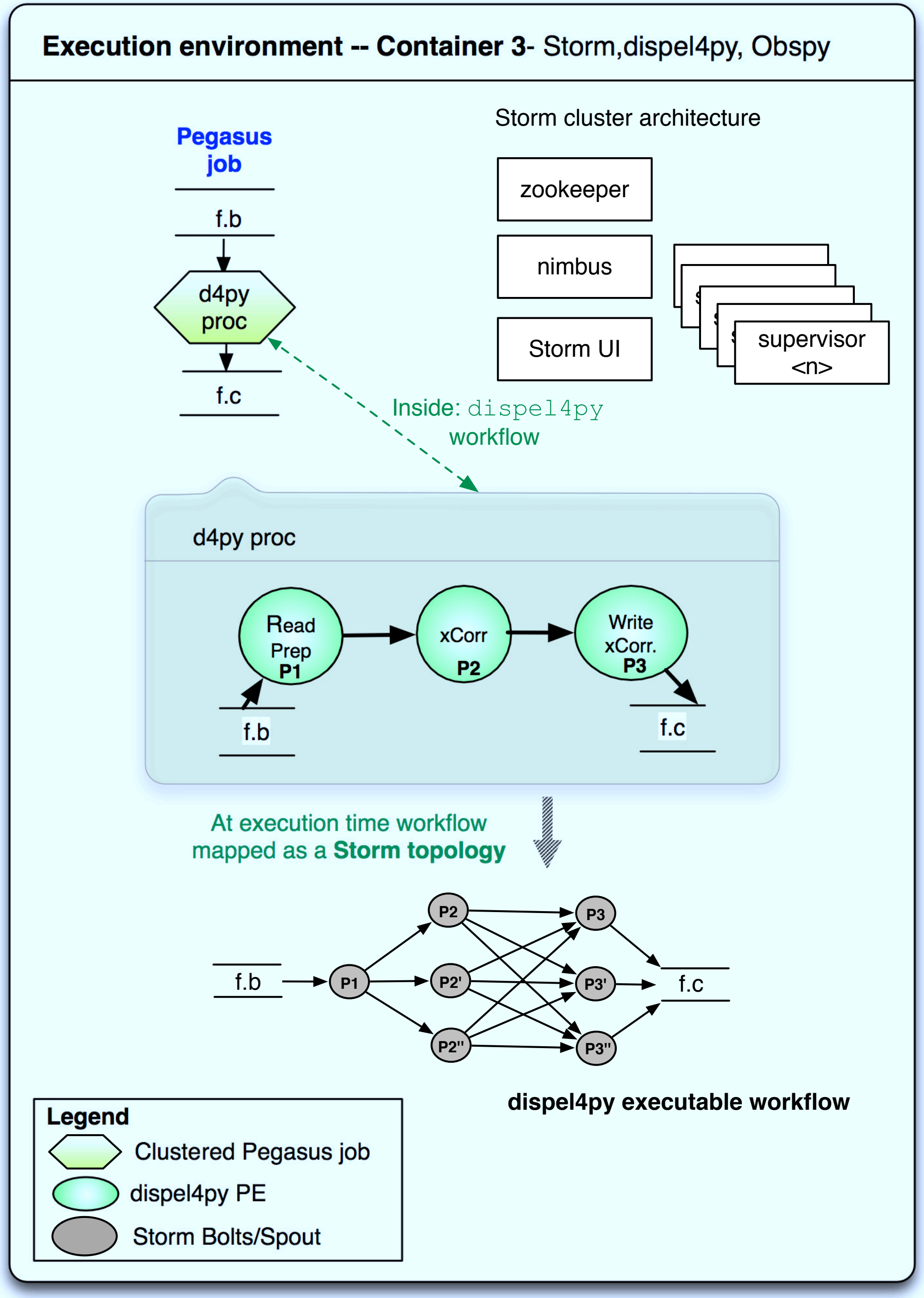
NSF-Chamelon cloud
1 node – 40 cores

Dockerfiles to configure containers **images** → Stored in our **GitHub** → Linked to **DockerHub** → stored/share/download images

Reminder







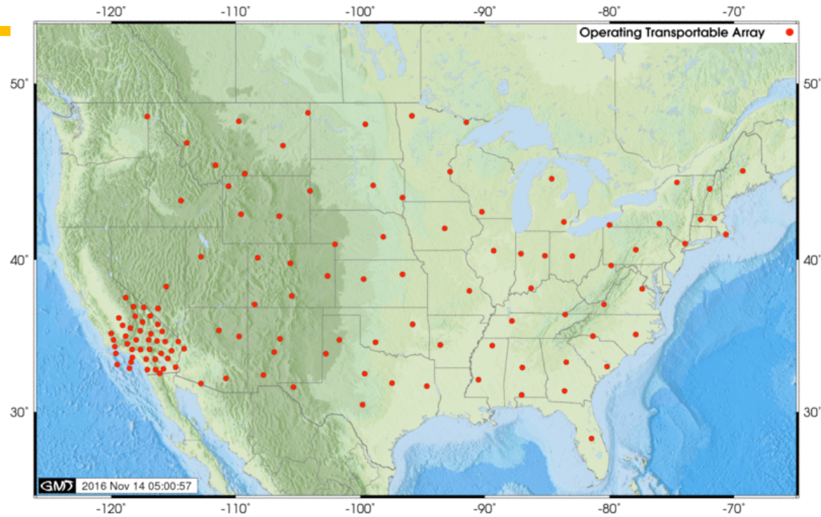
3 instances as Container 3 (zookeper, nimbus, Storm UI)

16 instances as Container 3 (Supervisors)

NSF-Chamelon cloud
1 node – 40 cores

Dockerfiles to configure containers **images** → Stored in our **GitHub** → Linked to **DockerHub** → stored/share/download images

Asterism Evaluations



Experiment 1: Data from IRIS services (394 stations)

Time
Phase 1 – 8 minutes
Phase 2 – 2 hours
Moving data < 1 minute

Data size
Input data 150MB
Output data 39GB

Experiment 2: Workflow for 3 days requesting data every 2 hours

NSF-Chamelon cloud
1 node – 40 cores

Scope of this work

Executing & paralyzing automatically data-intensive applications

in heterogeneous systems with different enactment engines

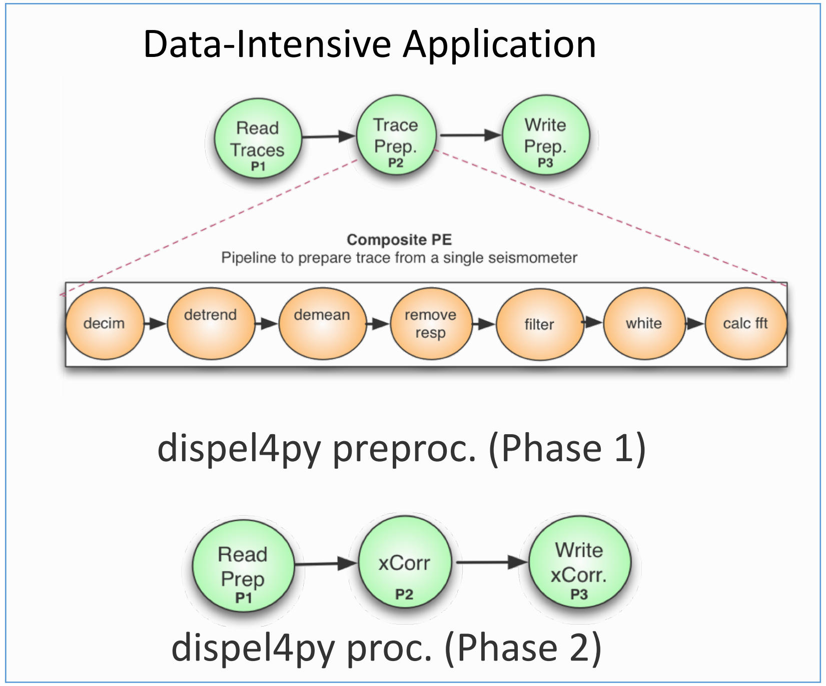
Maximizing performance
- both phases in the MPI cluster
- increasing the number of Storm Supervisors

epcc

dispel4py Performance Evaluations

Computing Resources	Terracorrelator	SuperMuc	Amazon EC2	EDIM1
Type	Shared-memory	Cluster	Cloud	Cloud
Enactment Systems	MPI, multi	MPI, multi	MPI, Storm, multi	MPI, Storm, multi
Nodes	1	16	18	14
Cores per Node	32	16	2	4
Total Cores	32	256	36	14
Memory	2TB	32GB	4GB	3GB

dispel4py Performance Evaluations



Both workflows (Phase 1 and Phase 2) executed in the same computing resource and the same mapping.

1000 stations
Input 150MB
Output 39GB

Mode	Terracorrelator (32 cores/Node)	SuperMuc (256 cores 16 cores/node)	Amazon (36 cores 2 cores/node)	EDIM1 (14 cores 4 cores/node)
MPI	1501.32 (~25minutes)	1093.16 (~**19minutes**)	16862.73 (~5hours)	38656.94 (~11 hours)
multi	1332 .20 (~23minutes)			
Storm			27898.89 (~8 hours)	120077.123 (~33 hours)

Asterism: Easy-to-use system to empower data-driven research

New framework that automatically

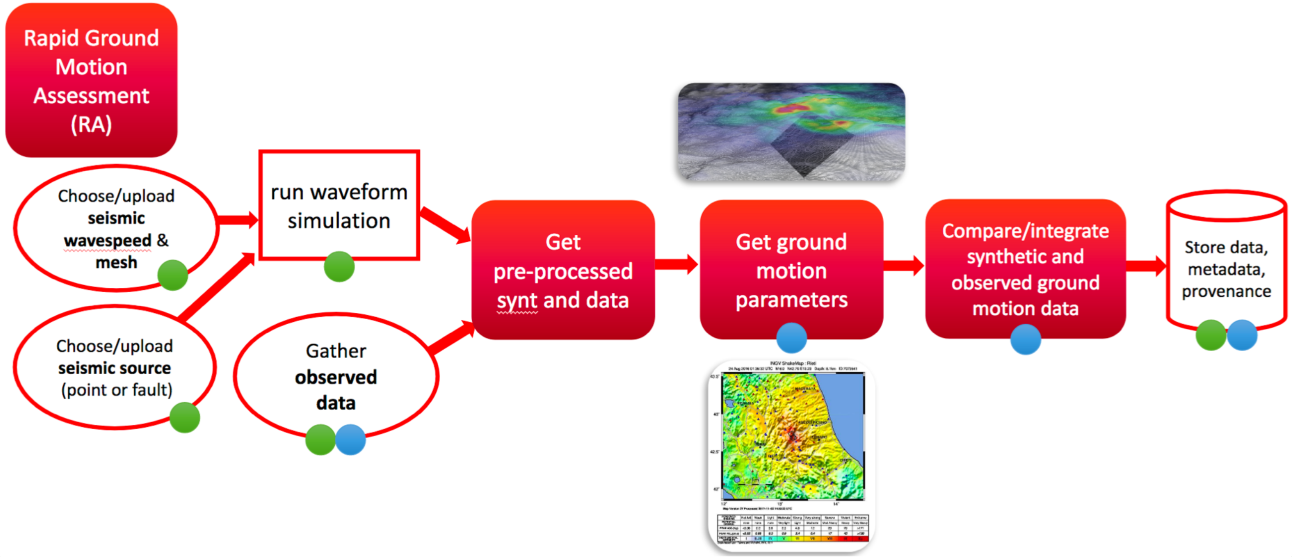
- manage the entire workflow, monitor its execution
- handle data transfers between different platforms
- map to different enactment engines at runtime

DARE (Delivering Agile Research Excellence on European e-Infrastructures)

Provide scientific communities with tools/frameworks/APIs for data-driven experiments, and rapid prototyping

European RIs, initially of EPOS, on Earth science, and IS/ENES2, on climate.

Rapid Ground Motion Assessment (RA) - EPOS



Aims:

Model the strong ground motion after large earthquakes

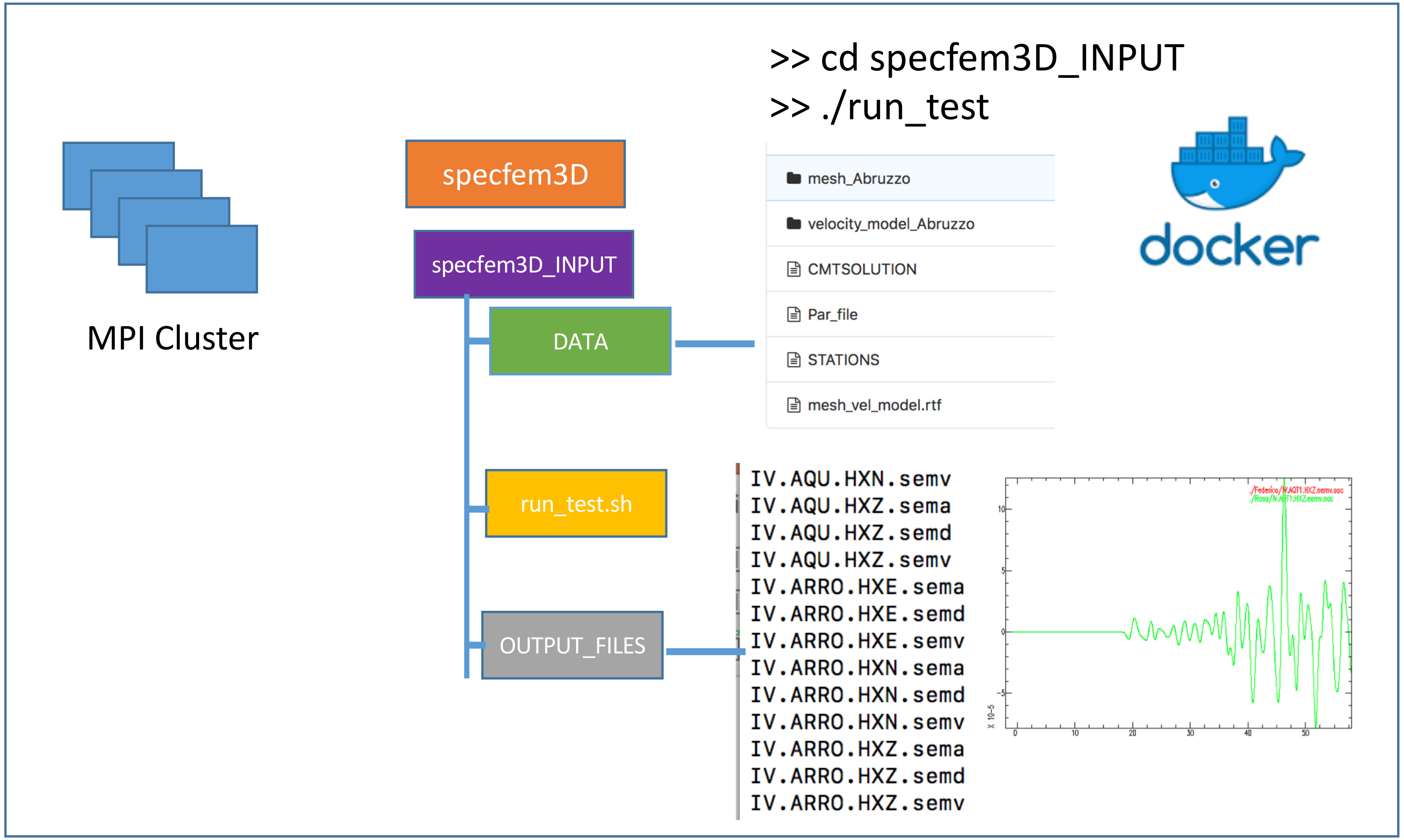
Rapid assessment of earthquakes' impact, and emergency response

Steps:

Build dispel4py workflows to test them locally – using small dataset.

Scale them up:
-Run the workflows using HPC/Cloud
** Using dispel4py parallel mappings

Waveform simulation: Specfem3D + MPI cluster



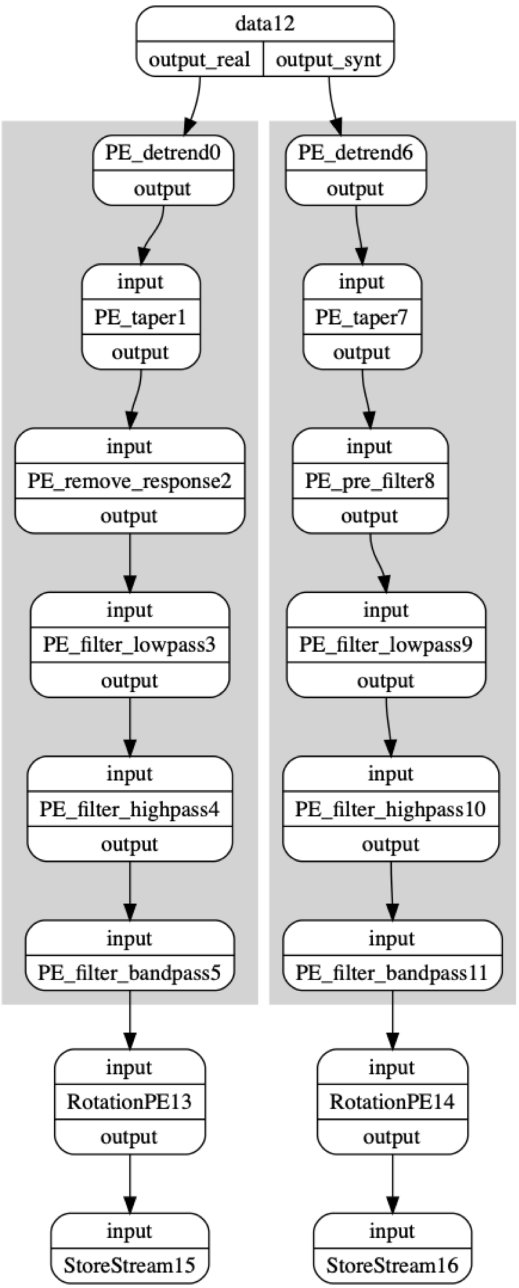
Test Case: RA

run waveform simulation

- Virtual Machine
  - (Bare-metal Instance)
  - 24 Cores
- Config script to install docker, pip & python.

- Clone the WP6_EPOS gitlab repo
- Docker info: Alpine 3.8, OpenMPI, Specfem3D
- Build the image
- Copy the Specfem3D_INPUT in the container
- Run the docker container

Get Pre-Preprocessed Synthetic and Data



Test Case: RA

Get pre-processed synt and data

Get Pre-Preprocessed Synthetic and Data

run_preprocess_misfit.sh 175 Bytes

```
#!/bin/bash

export PYTHONPATH=$PYTHONPATH
export MISFIT_PREP_CONFIG="pr
echo $MISFIT_PREP_CONFIG
dispel4py simple create_misfi
```

misfit_input.json 848 Bytes

Edit | Web IDE


```
{
    "data": [
        {
            "input": {
                "data_dir": "/Users/rosafilgueira/EPCC/DARE/WP6/test/MISFIT_RA/misfit_data/data",
                "synt_dir": "/Users/rosafilgueira/EPCC/DARE/WP6/test/MISFIT_RA/misfit_data/synth",
                "events": "/Users/rosafilgueira/EPCC/DARE/WP6/test/MISFIT_RA/misfit_data/events_simulation_CI_CI_test_0_1507128030823",
                "event_id": "smi:webservices.ingv.it/fdsnws/event/1/query?eventId=1744261",
                "stations_dir": "/Users/rosafilgueira/EPCC/DARE/WP6/test/MISFIT_RA/misfit_data/stations",
                "output_dir" : "/Users/rosafilgueira/EPCC/DARE/WP6/test/MISFIT_RA/misfit_data/output",
                "network": [
                    "IV"
                ],
                "station": [
                    "ARRO"
                ]
            }
        }
    ]
}
```


data

- IV.ARRO..EHE.mseed
- IV.ARRO..EHN.mseed
- IV.ARRO..EHZ.mseed

synth

```
IV.ARRO.HXE.sema
IV.ARRO.HXE.semd
IV.ARRO.HXE.semv
IV.ARRO.HXN.sema
IV.ARRO.HXN.semd
IV.ARRO.HXN.semv
IV.ARRO.HXZ.sema
IV.ARRO.HXZ.semd
IV.ARRO.HXZ.semv
```

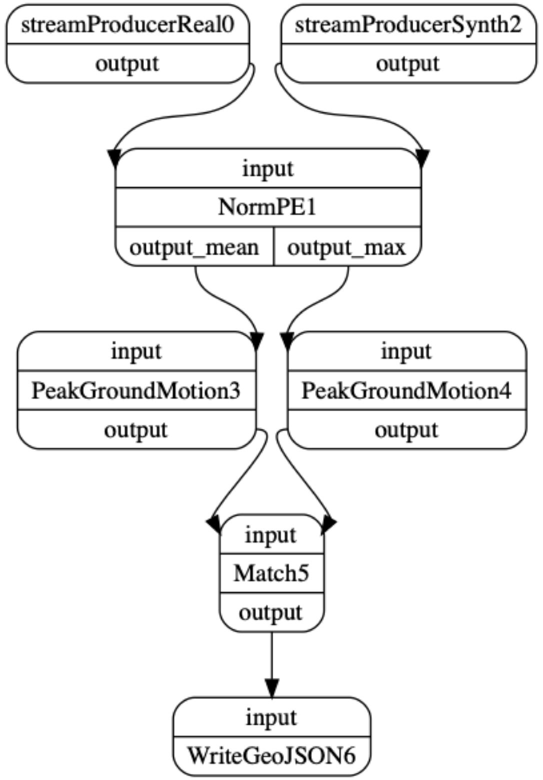
Note: Those are the synth seismograms generated in the previous step (slide)

output

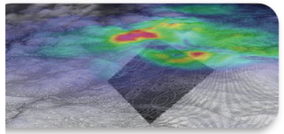
```
IV.ARRO.HXZ.synth
IV.ARRO.HXR.synth
IV.ARRO.HXT.synth
IV.ARRO.EHZ.data
IV.ARRO.EHR.data
IV.ARRO.EHT.data
```

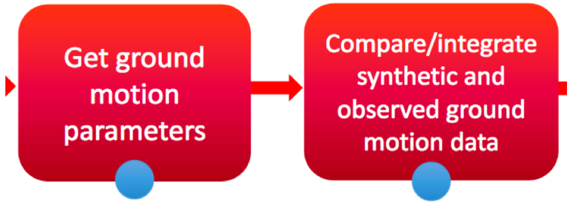
Pre-preprocessed synthetic and observed data
(Underline files are used in the next step (slide))

Get and Compare Ground Motion Parameters



Test Case: RA





Get and Compare Ground Motion Parameters

run_RA.sh 210 Bytes

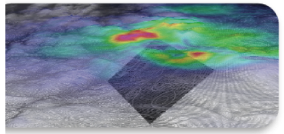
Lock | Edit | Web IDE | Replace | Delete

```
dispel4py simple dispel4py_RA.pgm_story.py -d '{"streamProducerReal": [ {"input": "./misfit_data/output/IV.ARRO.EHR.data"} ], "streamProducerSynth":
```

output

```
ARRO_max.json    ARRO_mean.json
```

Test Case: RA



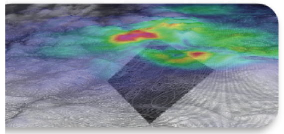
Get ground motion parameters → Compare/integrate synthetic and observed ground motion data

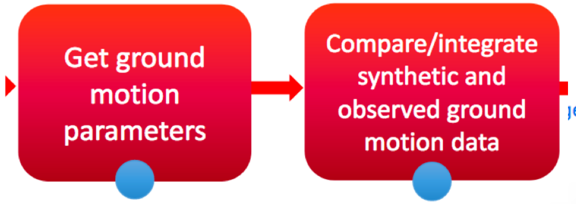
Get and Compare Ground Motion Parameters

type: "Feature"
properties:
  station: "ARRO"
  data:
    PGD: 0.000030589700119155l5
    PGV: 0.00015387362683992627
    PGA: 0.0008035731527687157
    p_norm: "max"
    PSA_0.3Hz: 0.000171880777011629
    PSA_1.0Hz: 0.002014083108433605 7
    PSA_3.0Hz: 0.0009812436987400703
  synt:
    PGD: 0.000038899272805005096
    PGV: 0.00011736045694475592
    PGA: 0.0003549366539822615 5
    p_norm: "max"
    PSA_0.3Hz: 0.0003942182089108563
    PSA_1.0Hz: 0.0007176179285863899
    PSA_3.0Hz: 0.0003707452407479844
  difference:
    PGD: -0.000008309572685849946
    PGV: 0.000036513169895170355
    PGA: 0.0004486364987864541
    PSA_0.3Hz: -0.0002223374318992273
    PSA_1.0Hz: 0.0012964651798472158
    PSA_3.0Hz: 0.0006104984579920859
  elative_difference:
    PGD: -0.2716460983102781
    PGV: 0.23729322980834633
    PGA: 0.5583020005592205
    PSA_0.3Hz: -1.2935561251517065
    PSA_1.0Hz: 0.6436999418834826
    PSA_3.0Hz: 0.6221680289778919
eometry:
  type: "Point"
  coordinates: []

output

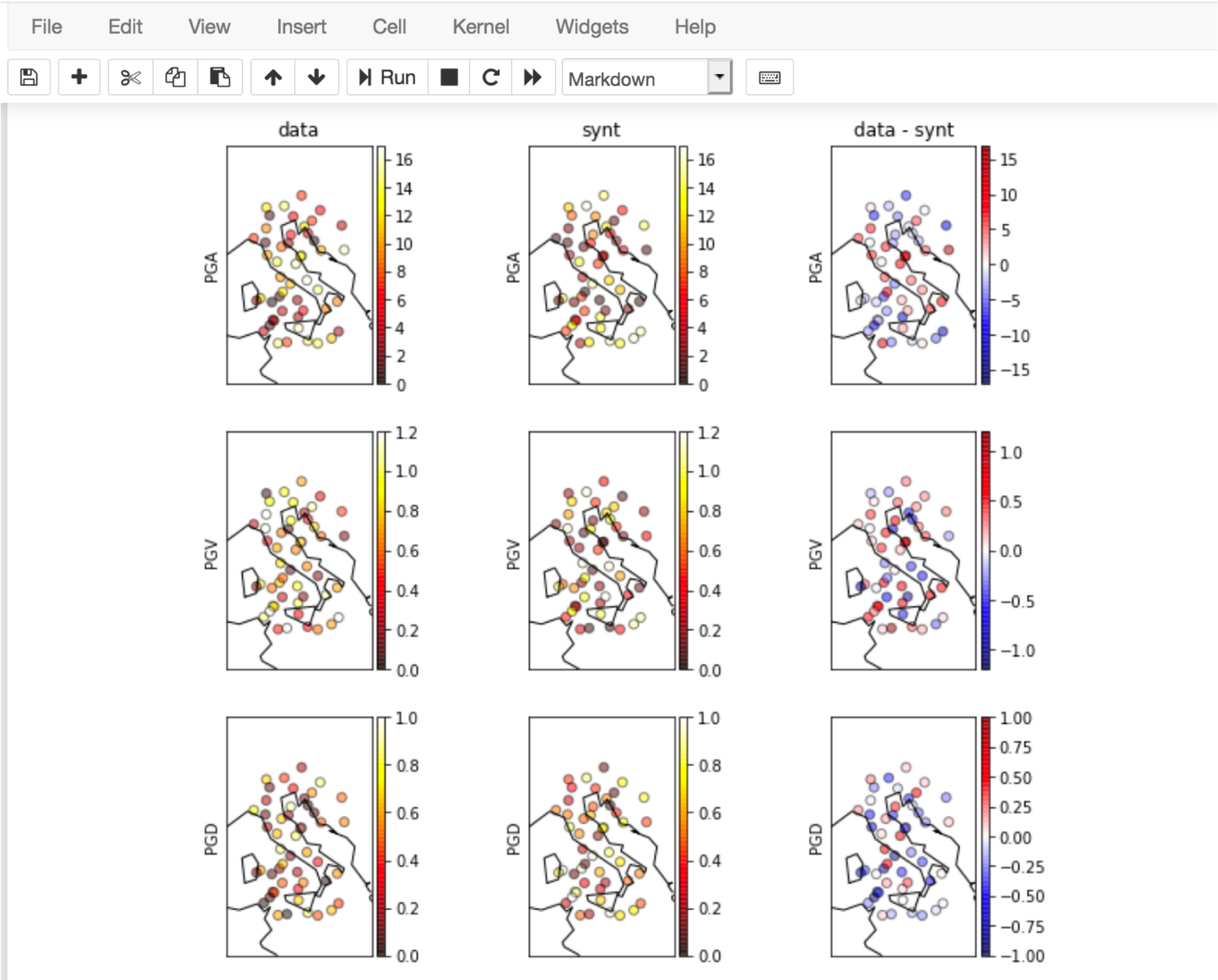
Test Case: RA



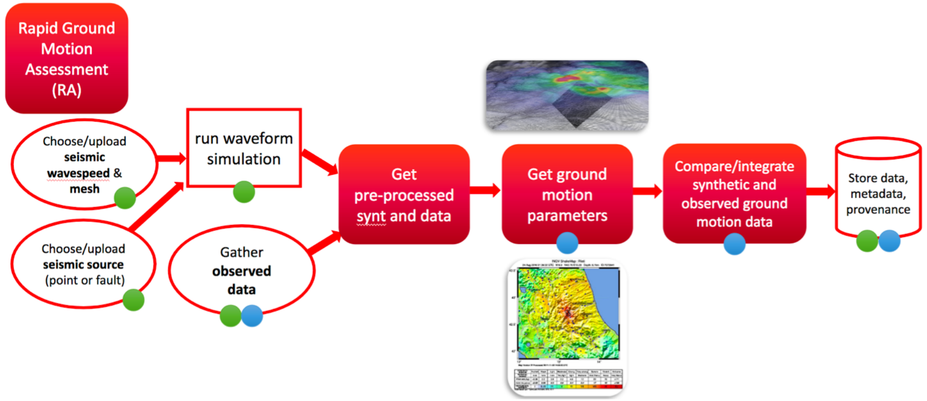


type: "Feature"
properties:
  station: "ARRO"
  data:
    PGD: 0.000030589700119155l5
    PGV: 0.00015387362683992627
    PGA: 0.0008035731527687157
    p_norm: "mean"
    PSA_0.3Hz: 0.000171880777011629
    PSA_1.0Hz: 0.0020140831084336057
    PSA_3.0Hz: 0.0009812436987400703
  synt:
    PGD: 0.000038899272805005096
    PGV: 0.00011736045694475592
    PGA: 0.00035493665398226155
    p_norm: "mean"
    PSA_0.3Hz: 0.0003942182089108563
    PSA_1.0Hz: 0.0007176179285863899
    PSA_3.0Hz: 0.0003707452407479844
  difference:
    PGD: -0.000008309572685849946
    PGV: 0.000036513169895170355
    PGA: 0.0004486364987864541
    PSA_0.3Hz: -0.0002223374318992273
    PSA_1.0Hz: 0.0012964651798472158
    PSA_3.0Hz: 0.0006104984579920859
  relative_difference:
    PGD: -0.2716460983102781
    PGV: 0.23729322980834633
    PGA: 0.5583020005592205
    PSA_0.3Hz: -1.2935561251517065
    PSA_1.0Hz: 0.6436999418834826
    PSA_3.0Hz: 0.6221680289778919
  geometry:
    type: "Point"
    coordinates: []

RA Maps


File
Edit
View
Insert
Cell
Kernel
Widgets
Help
Run
Markdown
data
synt
data - synt
PGA
PGV
PGD

Rapid Ground Motion Assessment (RA) - EPOS

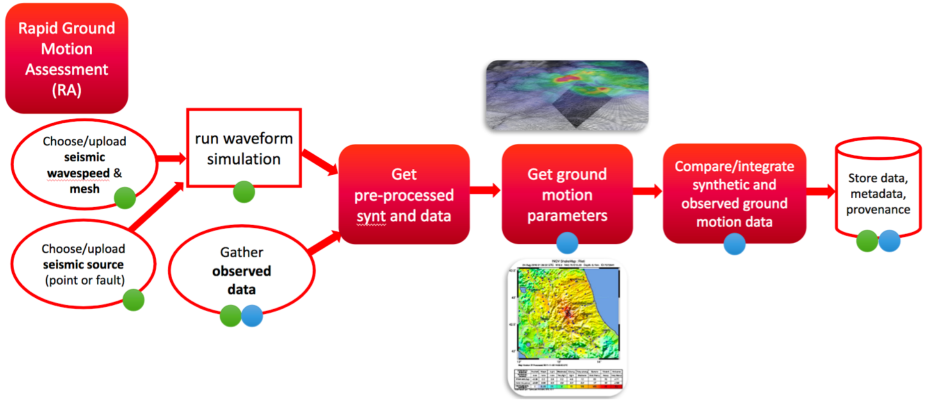


dispel4py + COMMON WORKFLOW LANGUAGE → semantics and description

CWL is a specification for describing the data and execution model of workflows/command tools

Provides a consistent way to connect programs

Rapid Ground Motion Assessment (RA) - EPOS



dispel4py + COMMON WORKFLOW LANGUAGE → semantics and description

CWL is a specification for describing the data and execution model of workflows/command tools

Provides a consistent way to connect programs

dispel4py-RA-pgm_story-job.yml 372 Bytes

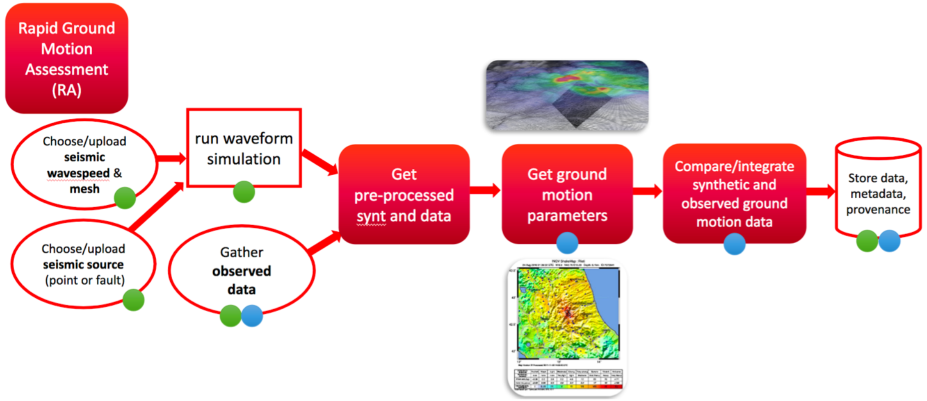
```
script:
  class: File
  path: /Users/rosafilgueira/EPCC/DARE/WP6/test/MISFIT_RA/dispel4py_RA.pgm_story.py


input: '{"streamProducerReal": [ {"input": "/Users/rosafilgueira/EPCC/DARE/WP6/test/MI
```

dispel4py-RA-pgm_story.cwl 437 Bytes

```
#!/usr/bin/env cwl-runner

cwlVersion: v1.0
class: CommandLineTool
baseCommand: [dispel4py, simple]
requirements:
  EnvVarRequirement:
    envDef:
      PYTHONPATH: $(inputs.script.dirname)

inputs:
- id: script
  type: File
  inputBinding:
    position: 1
- id: input
  type: string
  inputBinding:
    prefix: -d
    position: 2

outputs:
  output:
    type:
      type: array
      items: File
    outputBinding:
      glob: "*.json"
```

```
cwl-runner dispel4py-RA-pgm_story.cwl dispel4py-RA-pgm_story-job.yml
```

Rapid Ground Motion Assessment (RA) - EPOS



Once tested all the workflows locally (seq. map),
They can be scaled up automatically → Next step.

Provenance data is also collected in runtime (optional)

Metadata, Input and Output files --> registry/catalog

Future development:

Optimization of workflows → dynamic deployment
→ mapping selection

Delivering easy-to-use frameworks to empower data-driven research

Dr. Rosa Filgueira, University of Edinburgh, EPCC

Extra Slides

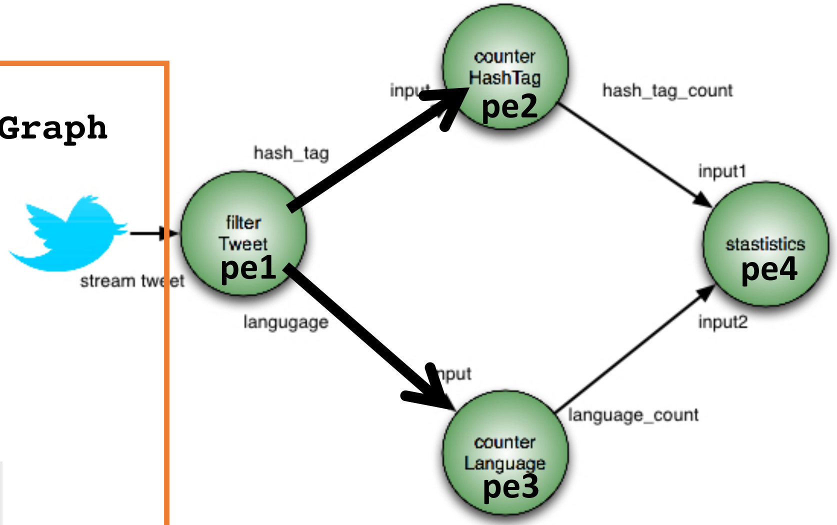
dispel4py parallel stream-based dataflow system

```
from dispel4py.workflow_graph import WorkflowGraph

pe1 = filterTweet ()
pe2 = counterHashTag ()
pe3 = counterLanguage ()
pe4 = statistics ()

graph = WorkflowGraph ( )

graph.connect(pe1,'hash_tag',pe2,'input')
graph.connect(pe1,'language',pe3,'input')
graph.connect(pe2,'hash_tag_count',pe4,'input1')
graph.connect(pe3,'language_count',pe4,'input2')
```

PEs objects

Graph

Connections



Users only have to implement:

- PEs
- Connections

https://github.com/dispel4py/dispel4py

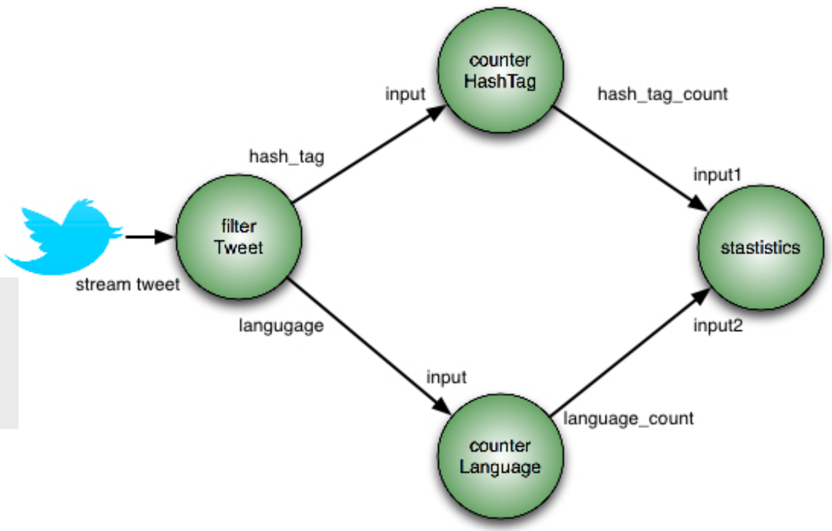
dispelp4y basic concepts–Example of a PE

```
Class filterTweet(GenericPE):
def __init__( self ):
    GenericPE.init (self)
    self.add_output('hash_tags ')
    self.add_output('language')

def process ( self , inputs ):
        twitterDataFile= inputs['input ']
        tweet file = open(twitterDataFile)
        for line in tweet file :
        tweet = json.loads( line )
                language = ' '
                hashtags =[]
                language = tweet[u'lang '].encode('utf -8')
                text = tweet[u'text '].encode('utf-8')
            hashtags=re.findall(r"#(\w+)", text)

                self.write('hash tag', hashtags)
                self.write('language', language)
```

Inputs & outputs

Logic of PE

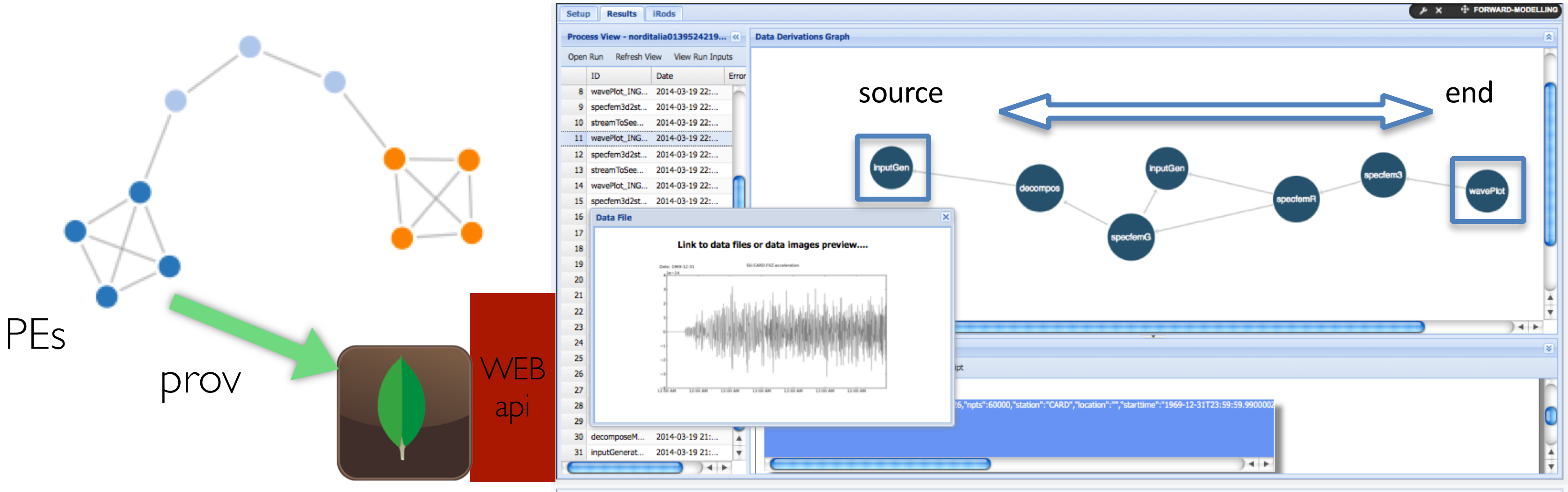
Stream out data



Users only have to implement:

- PEs
- Connections

dispel4py advance concepts – Provenance



Users can select which metadata to store
Searches over products metadata within and across runs
Data download and preview
Capturing of Errors for Diagnostic purposes
Data Fabric: Multi directional navigations across data dependencies
W3C PROV-DM as reference model.