

# *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 (reducing overhead in IO and communication systems) to enhance performance of MPI-based applications*
- 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 – *New data-driven frameworks to develop data-intensive applications that run in heterogenous systems*

# Introduction

## Big Data Sciences Era, Data Intensive Computing applications



Scientific fields →  
Data-driven

Astronomy, Geosciences,  
Meteorology, Bioinformatics



Common points

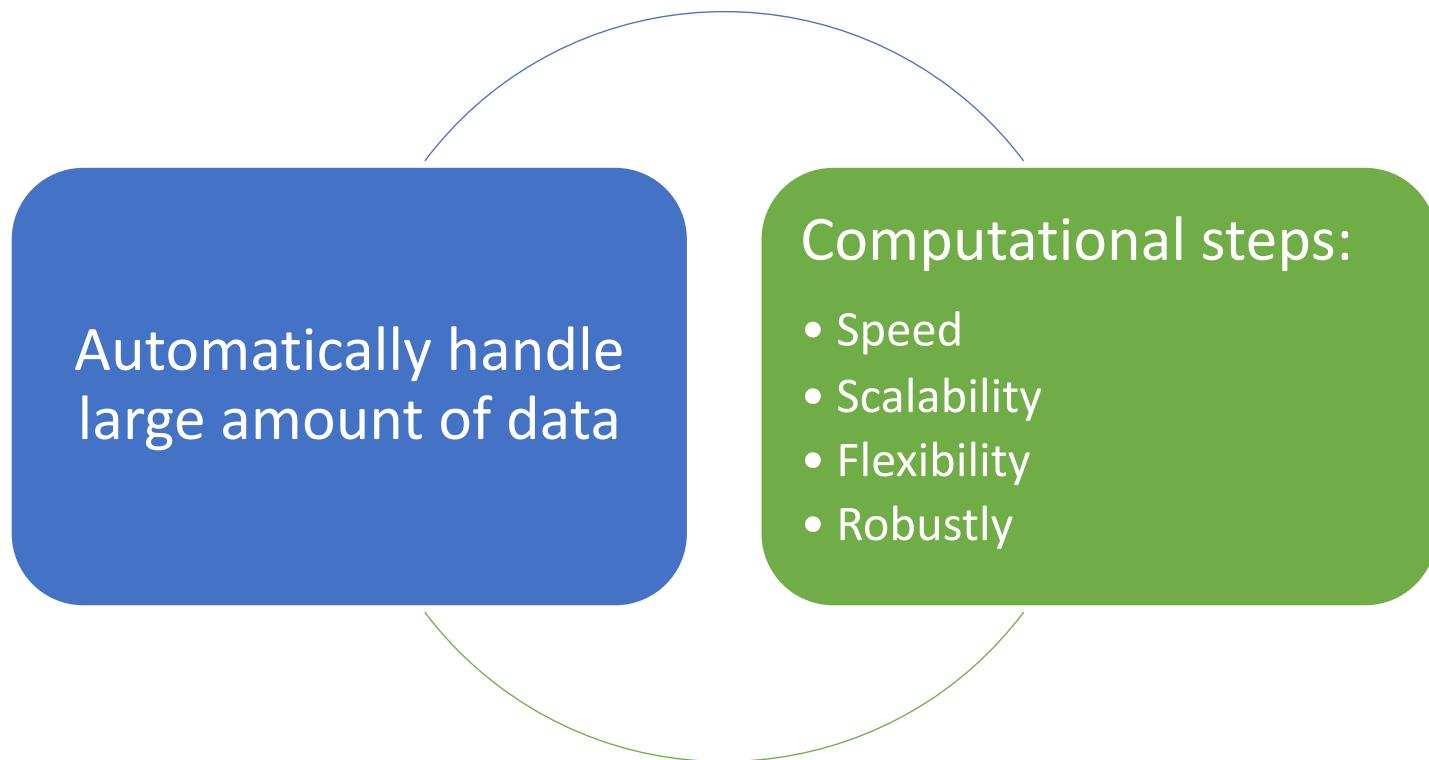
Big complex data sets  
Need to be analysed  
Numerous software tools  
Data transformation and visualisation

# Introduction

## Big Data Sciences Era, Data Intensive Computing applications

---

Data analytic frameworks and Computing environments

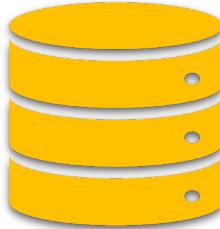


# Delivering easy-to-use frameworks to empower data-driven research

---



**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

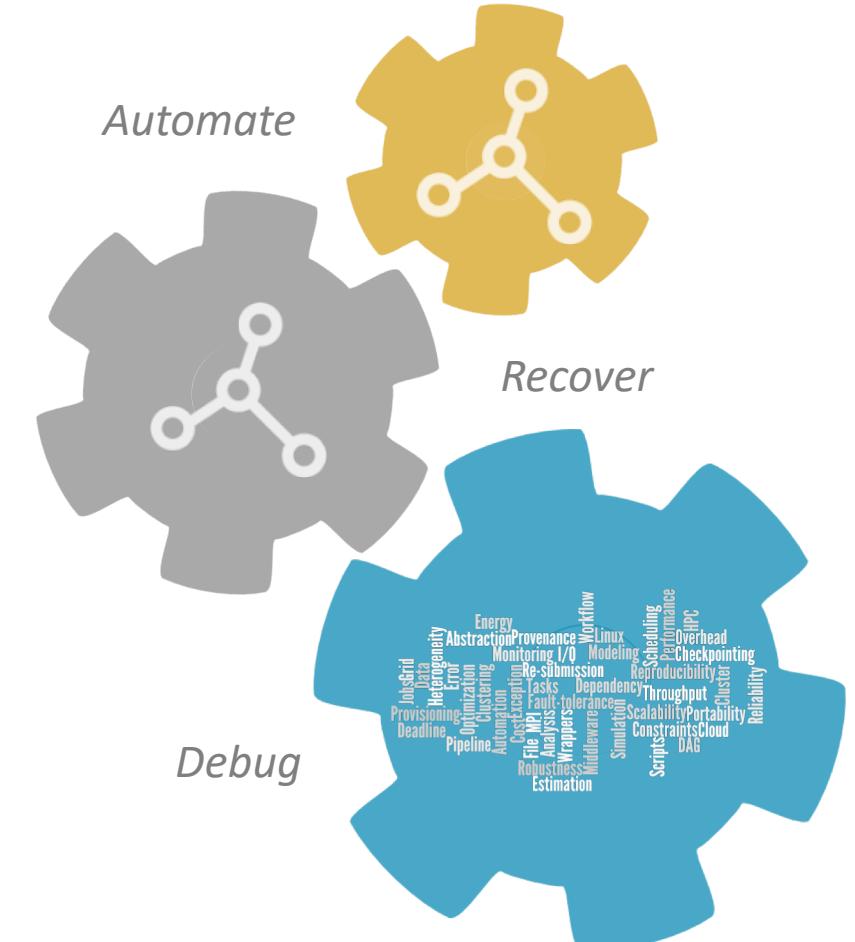
task-flow vs. data-flow

files vs. stream-based



## *Workflow Management Systems (WMS)*

Provide tools to generate the scientific workflow



# WORKFLOW MANAGEMENT SYSTEMS

---



**Taverna**

<https://taverna.incubator.apache.org>



**KNIME**

<https://www.knime.org>



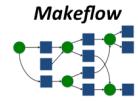
**Kepler**

<https://kepler-project.org>



**VisTrails**

<http://vistrails.org>



**Makeflow**

<http://ccl.cse.nd.edu/software/makeflow>



**FireWorks**

<https://pythonhosted.org/FireWorks>



**dispel4py**

<http://dispel4py.org>



**Swift**

<http://swift-lang.org>



**Pegasus**

<http://pegasus.isi.edu>



**Common Workflow Language**  
<https://www.commonwl.org>



# Asterism Framework

---

Easy to understand, platform-independent, open-source

Simplifies the development applications running across multiple heterogeneous

**How ?**

Combining the strengths of



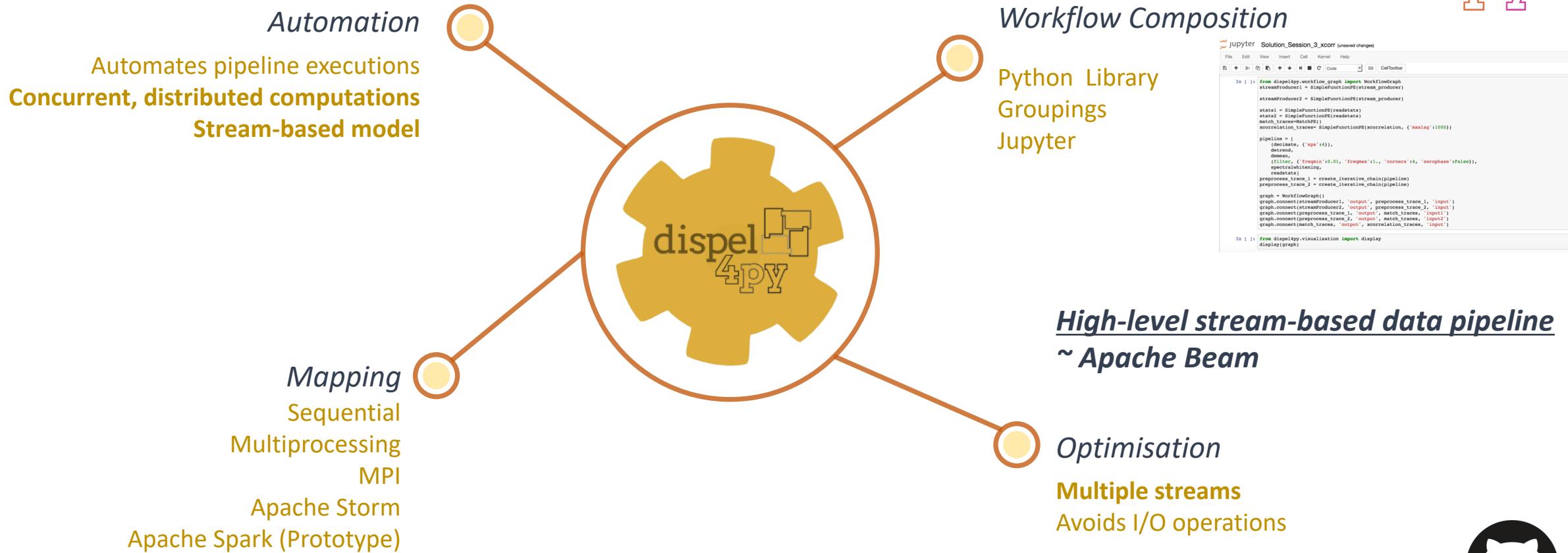
Traditional WMS  
Pegasus

New stream-based data-flow systems  
dispel4py

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

*Rosa Filgueira, EPCC-UoC, UK  
Amrey Krause, EPCC-UoE, UK  
Malcolm Atkinson, Informatics-UoE, UK*

# dispel4py parallel stream-based dataflow system



A screenshot of a Jupyter Notebook interface. The title bar says "jupyter Solution\_Session\_3\_xcorr (unSaved changes)". The notebook shows two code cells:

```
In [1]: from dispel4py.workflow_graph import WorkflowGraph
streamProducer1 = SimpleFunctionP(stream_producer)
streamProducer2 = SimpleFunctionP(stream_producer)
statistic = SimpleFunctionP(readstats)
status = SimpleFunctionP(readstats)
xcorrFunctionP = XCorrelationFunctionP(xcorrelation, {'maxlag':1000})
pipeline = [
    (decimate, {'laga':14}),
    (detrend, {}),
    (filter, {'fregmin':10.0, 'fregmax':11., 'corner':4, 'zerophase':False}),
    (readstats),
    (status),
    (xcorrFunctionP),
    (preprocess_trace_1 = create_iterative_chain(pipeline)),
    (preprocess_trace_2 = create_iterative_chain(pipeline))
]
graph = WorkflowGraph()
graph.connect(streamProducer1, 'output', preprocess_trace_1, 'input')
graph.connect(streamProducer2, 'output', preprocess_trace_2, 'input')
graph.connect(preprocess_trace_1, 'output', match_traces, 'input1')
graph.connect(preprocess_trace_2, 'output', match_traces, 'input2')
graph.connect(match_traces, 'output', xcorrFunctionP, 'input')

In [2]: from dispel4py.visualization import display
display(graph)
```



**Key-features:** Automatic mappings to different engines, concurrent & stream-based

# dispel4py parallel stream-based dataflow system

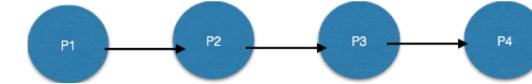


## Graph

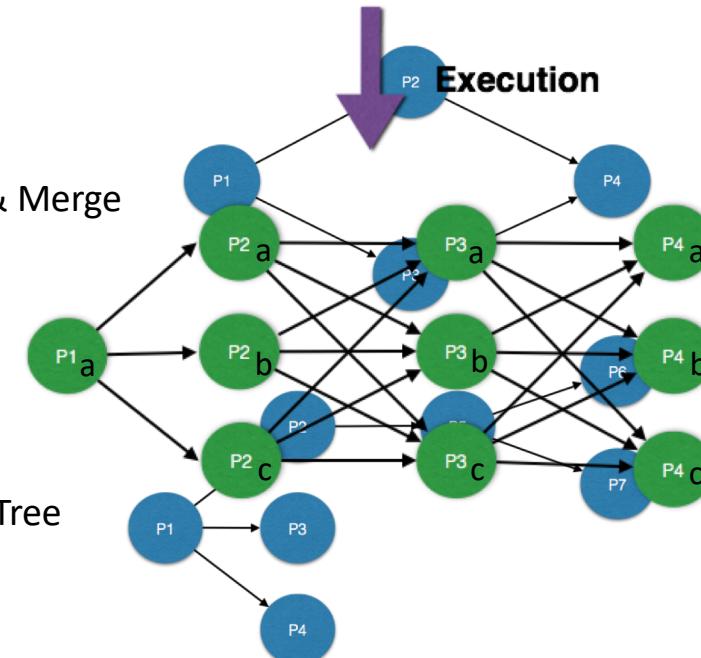
- Connections among PEs
- Abstract workflow

### + Example of graphs

#### Pipeline



#### Split & Merge



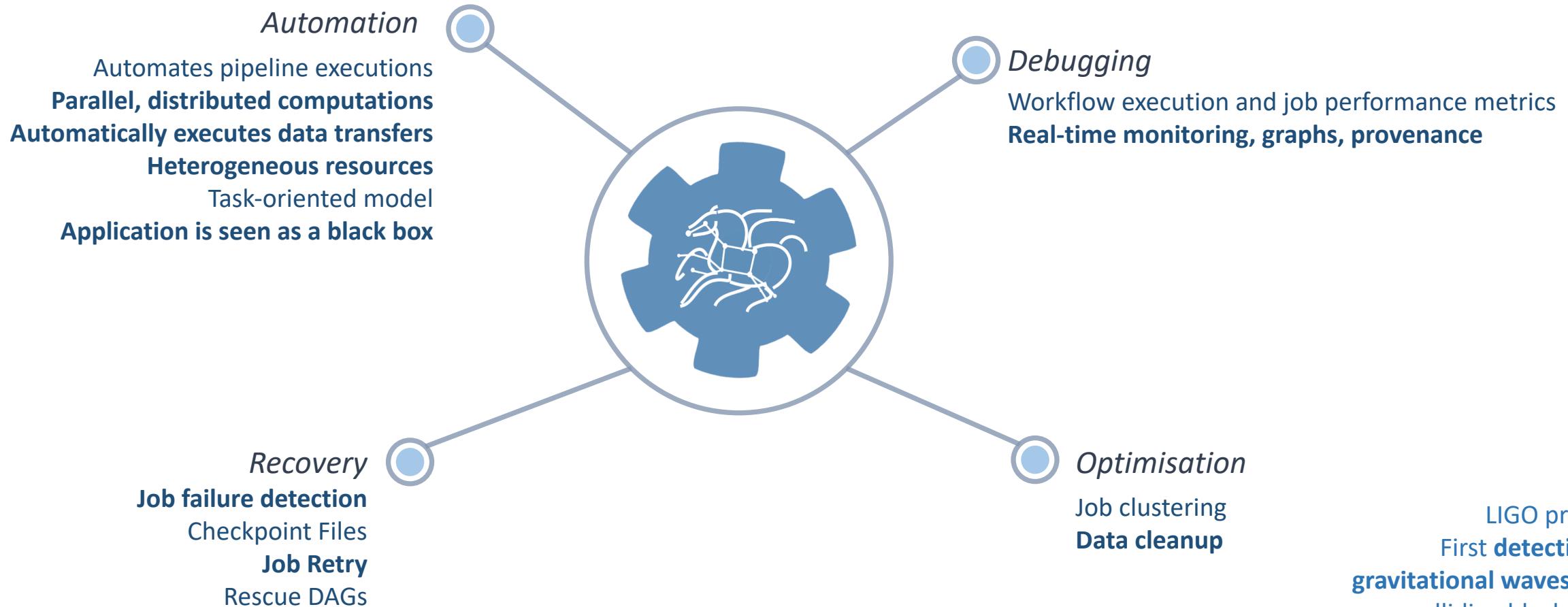
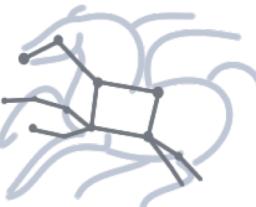
#### Tree

4 PEs & 10 processes



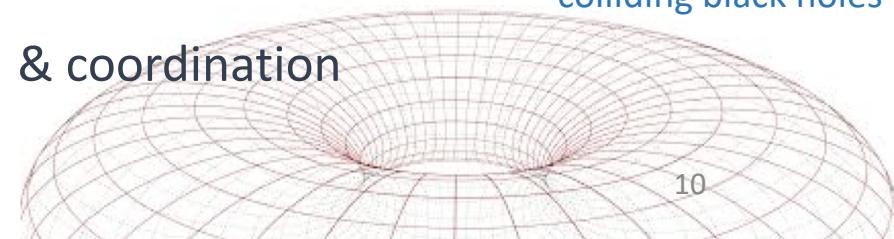
epcc

# Pegasus workflow system

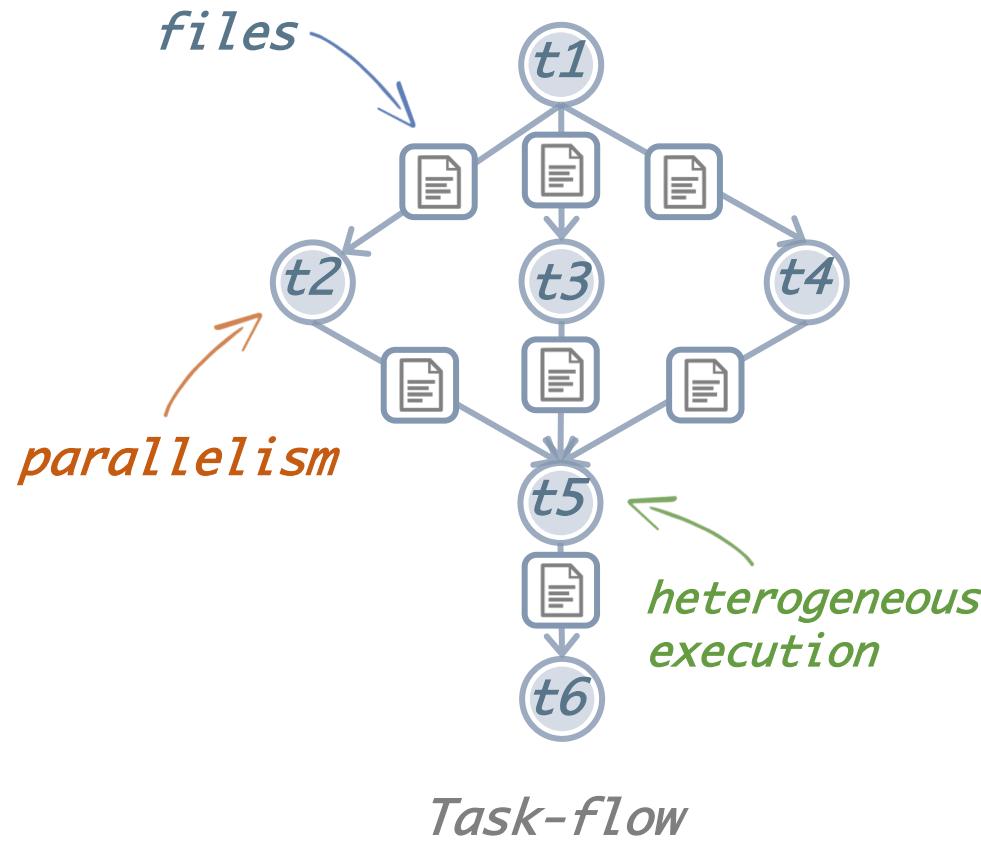


**Key-features:** Automatic data movement, cleanup, heterogeneous & coordination

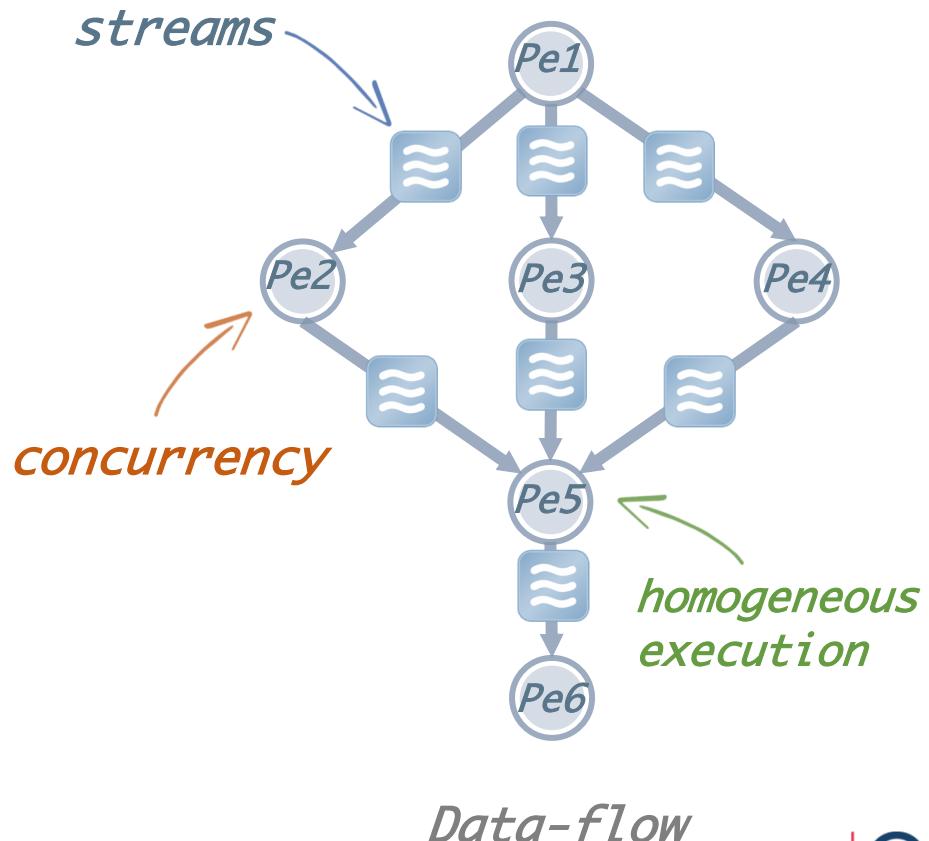
LIGO project:  
First detection of  
gravitational waves from  
colliding black holes



# Complementary systems

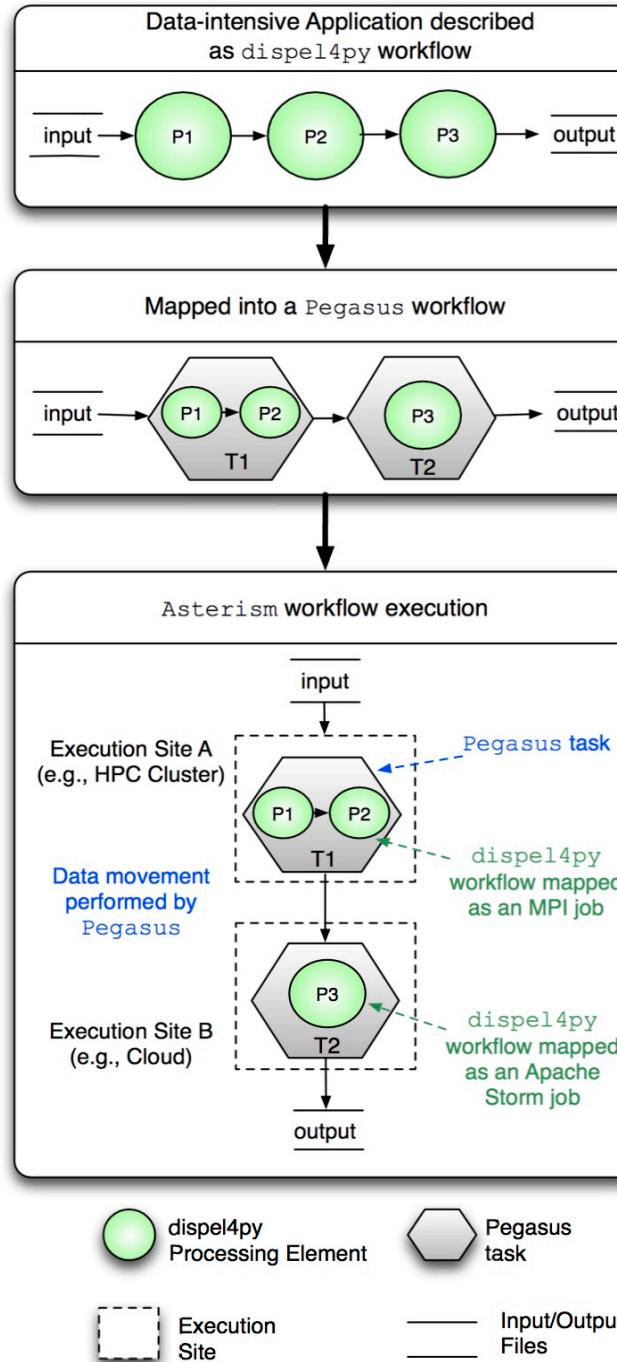


dispel  
4py





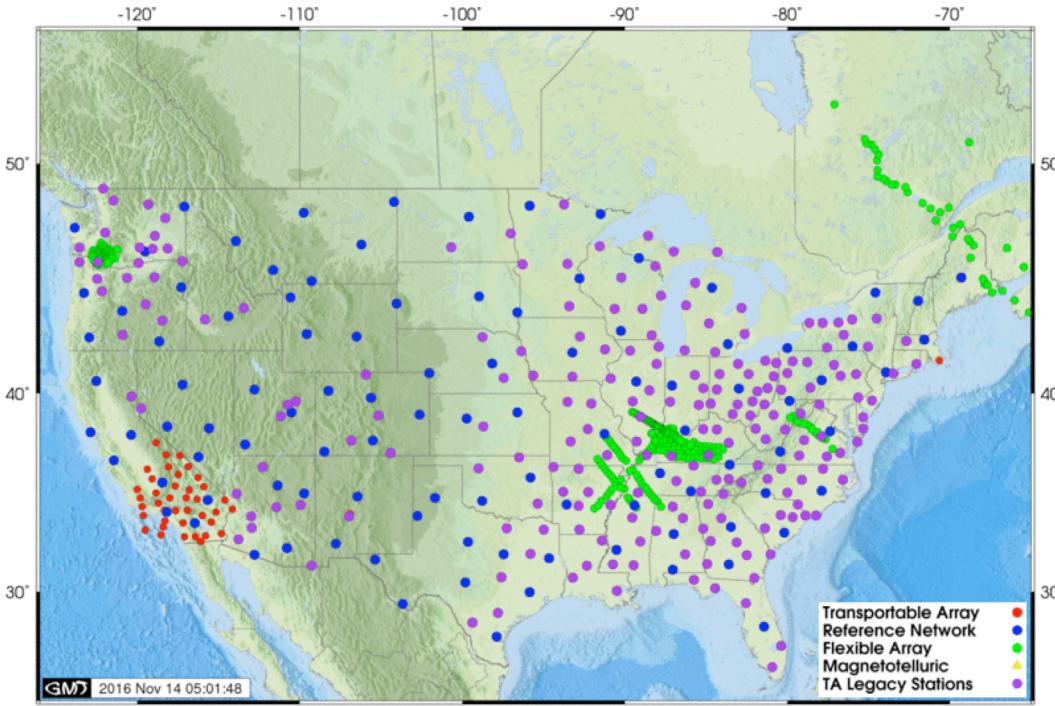
# ASTERISM



dispel4py to represent different parts of applications

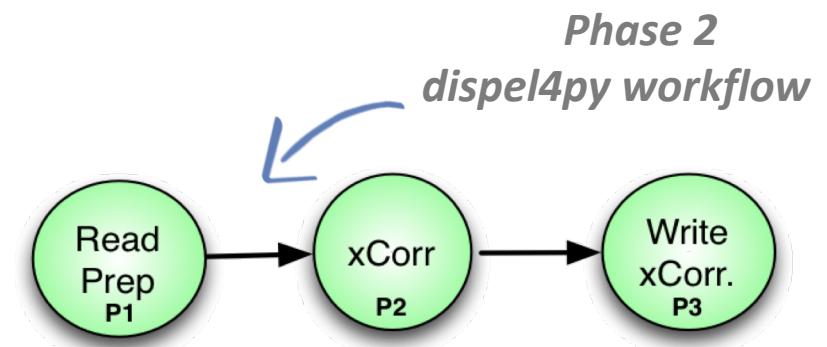
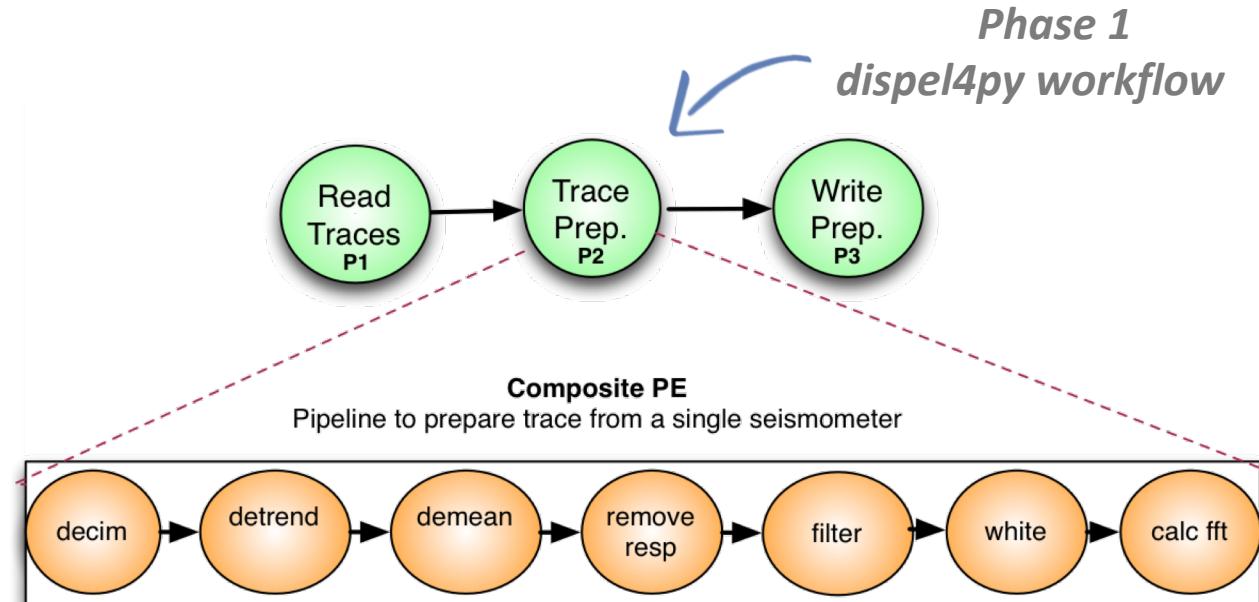
Pegasus to distribute and execute each dispel4py workflow

# Seismic Ambient Noise Cross-Correlation



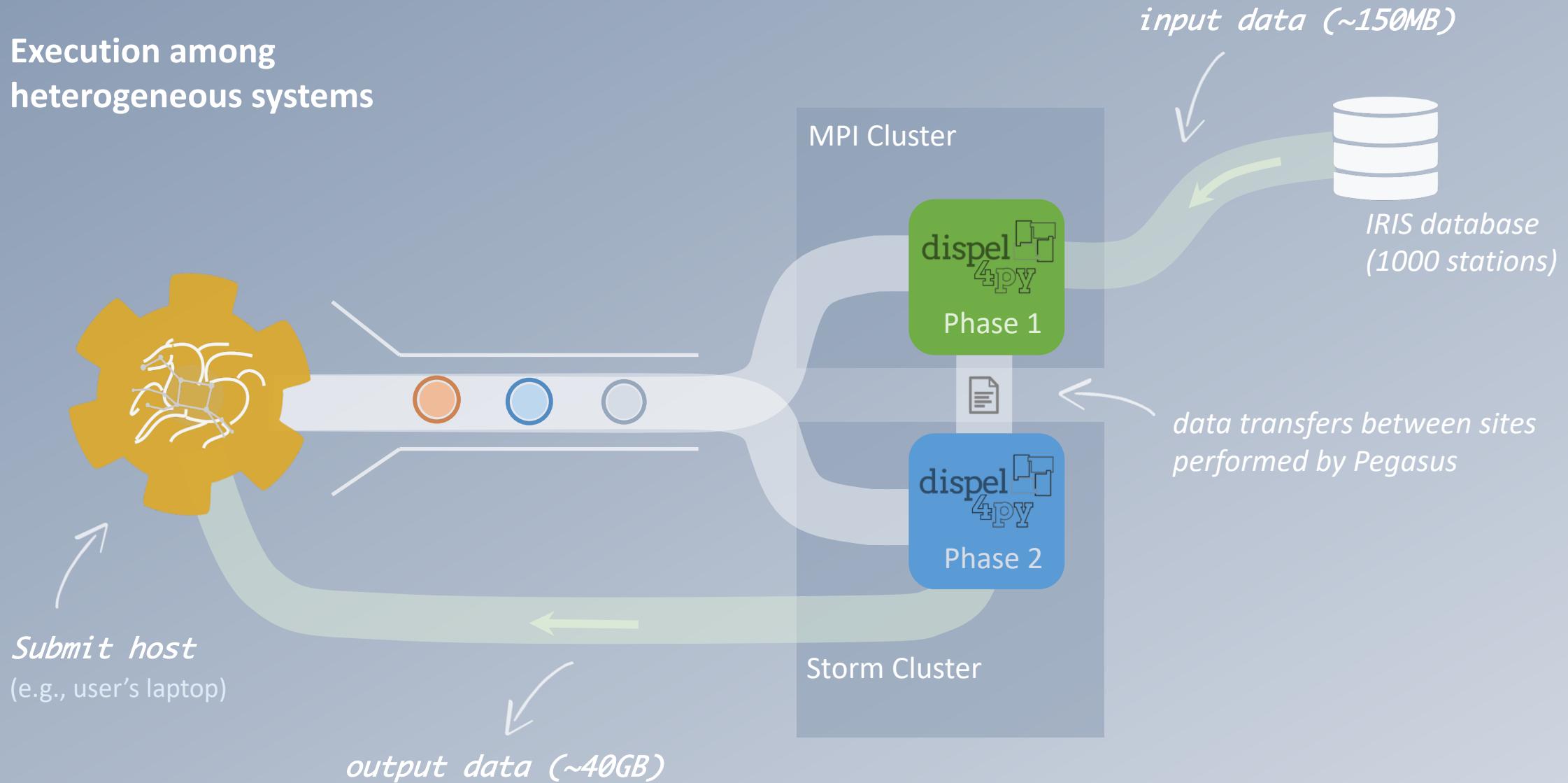
Preprocesses (Phase 1) and cross-correlates traces (Phase 2) from multiple seismic stations

VERCE project both phases on the same HPC resource.  
It delivers a productive e-Science environment for seismology research



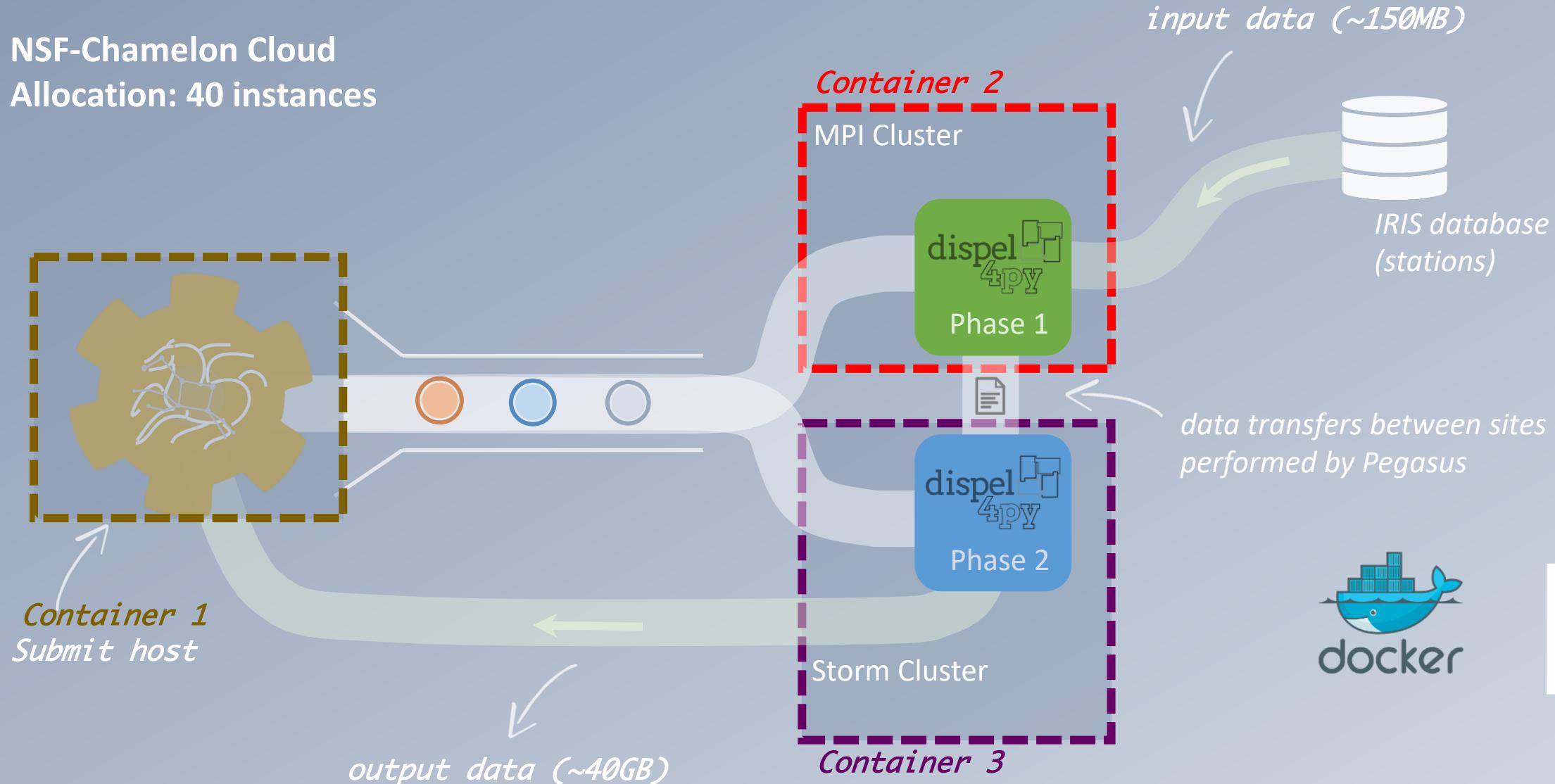
# Evaluation- Seismic Ambient Noise Cross-Correlation

Execution among  
heterogeneous systems



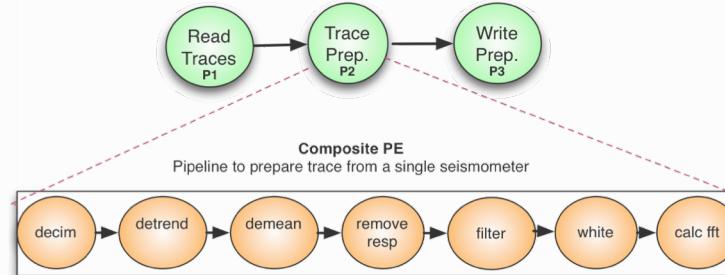
# Evaluation- Seismic Ambient Noise Cross-Correlation

NSF-Chameleon Cloud  
Allocation: 40 instances

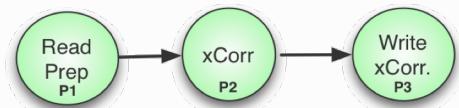


## Reminder

### Data-Intensive Application



dispel4py preproc. (Phase 1)



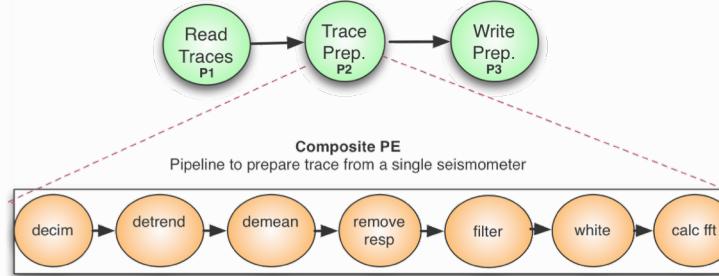
dispel4py proc. (Phase 2)



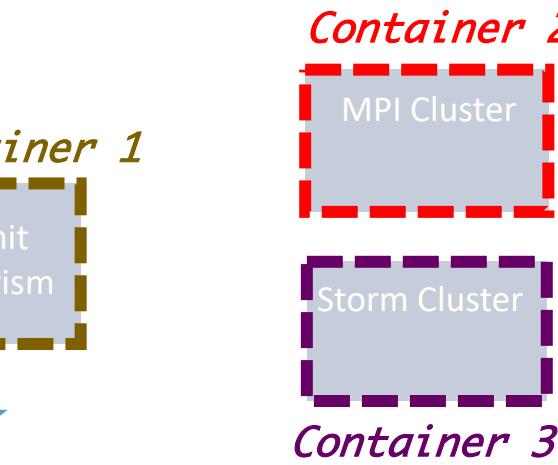
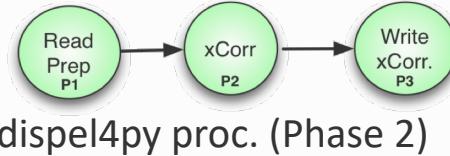
Dockerfiles to configure containers **images** → Stored in our **GitHub** → Linked to **DockerHub** → stored/share/download images

## Reminder

### Data-Intensive Application

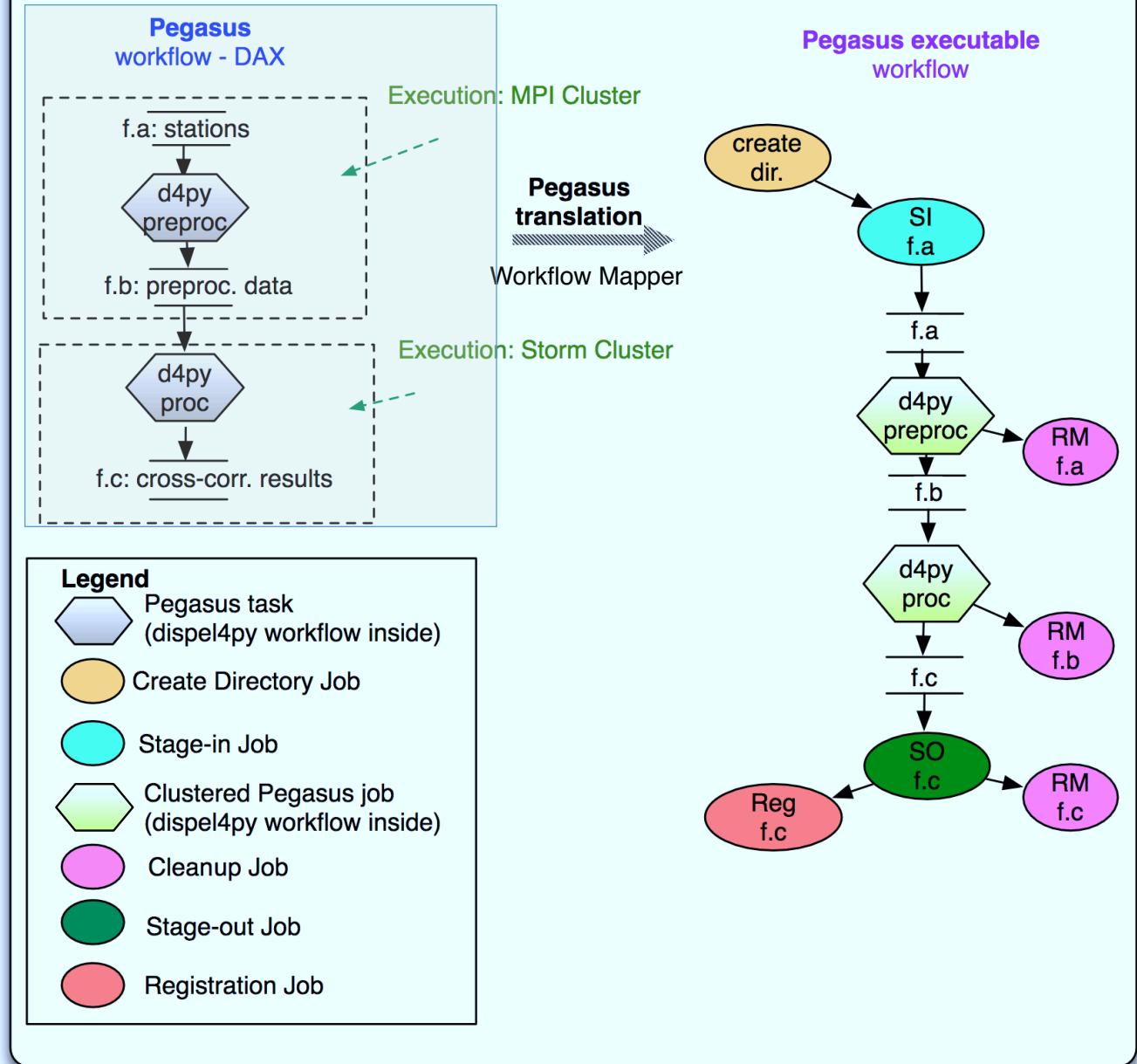


dispel4py preproc. (Phase 1)



### Asterism Seismic Cross Correlation workflow

#### Execution environment -- Container 1: Pegasus, HTCondor, dispel4py



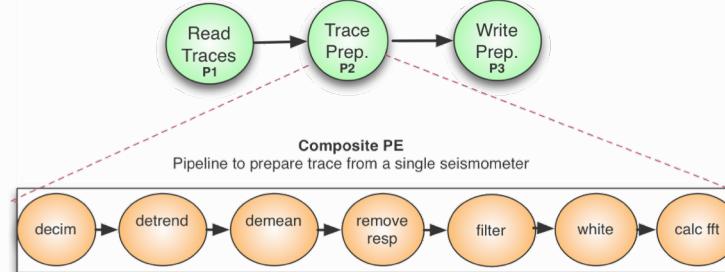
1 instance as  
Container 1

NSF-  
Chameleon  
cloud  
1 node –  
40 cores

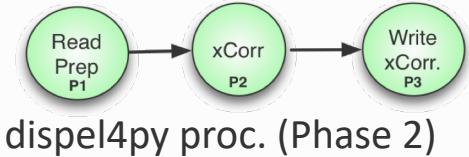


## Reminder

### Data-Intensive Application



dispel4py preproc. (Phase 1)



dispel4py proc. (Phase 2)

*Container 2*

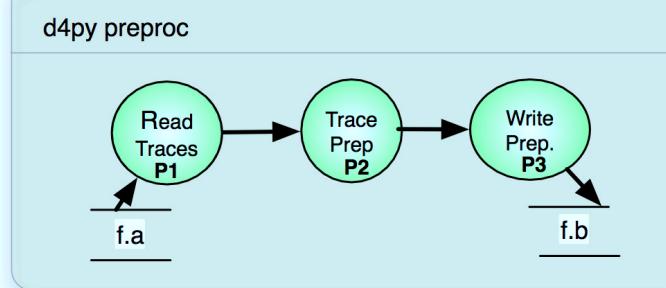
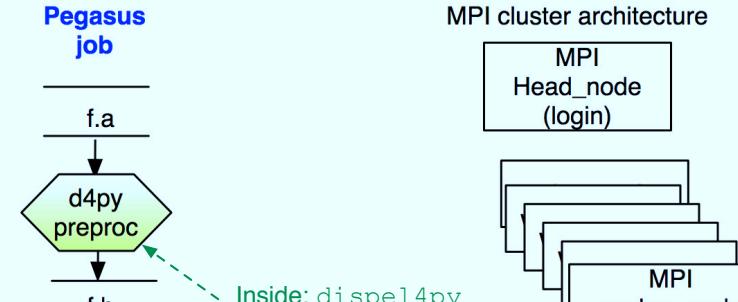


*Container 1*

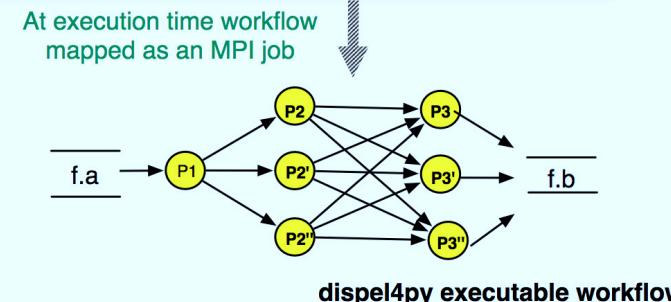


*Container 3*

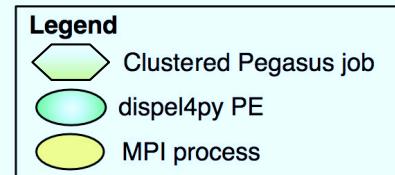
### Execution environment -- Container 2- MPI cluster, dispel4py, Obspy



At execution time workflow mapped as an MPI job



dispel4py executable workflow



1 instance as  
Container 2  
(MPI head node)

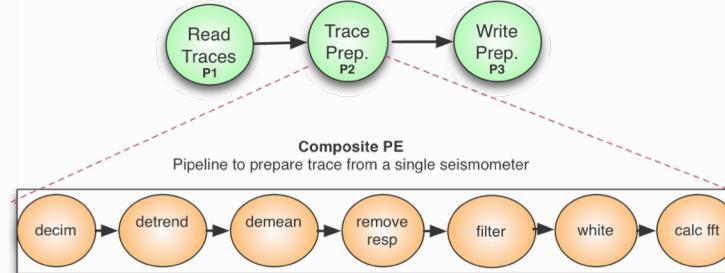
16 instances as  
Container 2  
(MPI workers)

NSF-  
Chameleon  
cloud  
1 node –  
40 cores

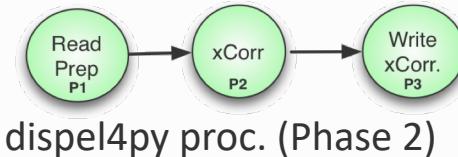


## Reminder

### Data-Intensive Application



dispel4py preproc. (Phase 1)



dispel4py proc. (Phase 2)

*Container 2*

MPI Cluster

Phase 1

Storm Cluster

Phase 2

*Container 3*

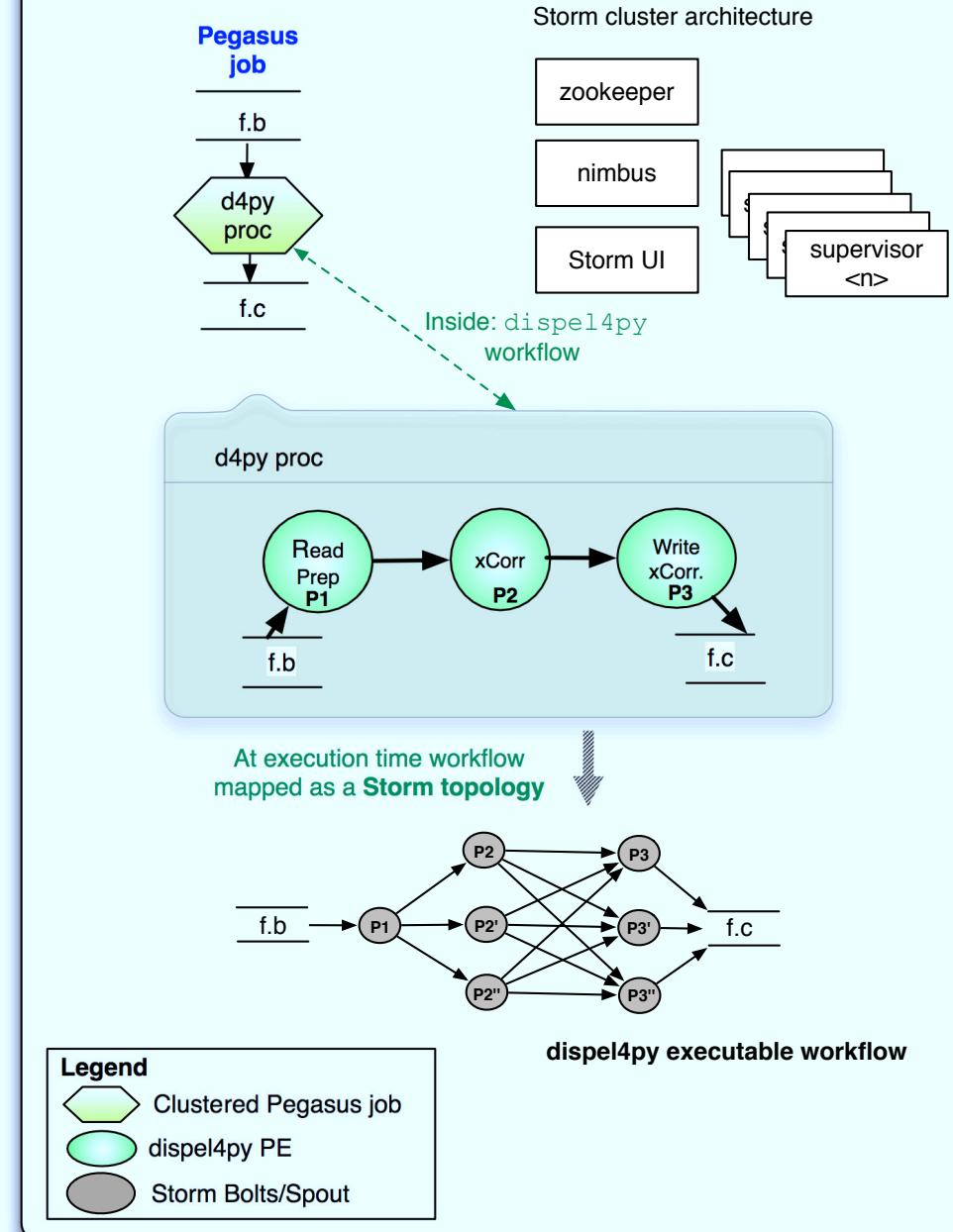
*Container 1*

Submit Asterism

**3 instances as Container 3 (zookeeper, nimbus, Storm UI)**

**16 instances as Container 3 (Supervisors)**

### Execution environment -- Container 3- Storm, dispel4py, Obspy



NSF-Chameleon cloud  
1 node – 40 cores



# Asterism Evaluations

Experiment 1: Data from IRIS services (394 stations)

## Time

Phase 1 – 8 minutes

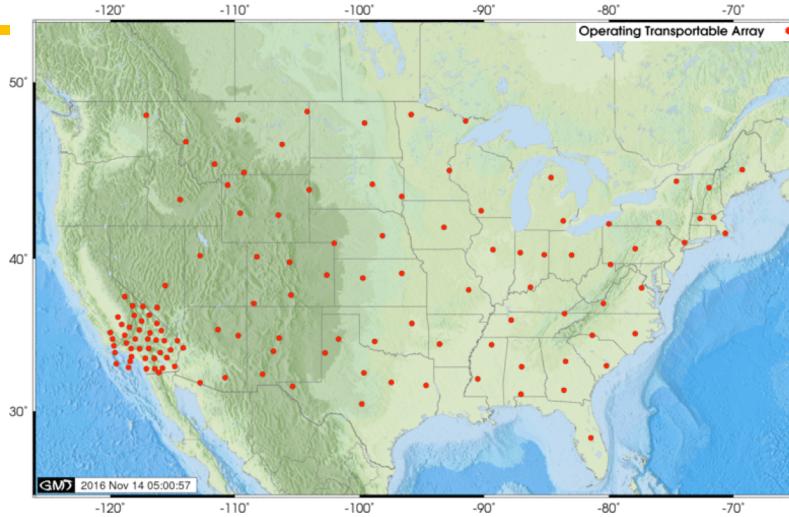
Phase 2 – 2 hours

Moving data < 1 minute

## Data size

Input data 150MB

Output data 39GB



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

## Scope of this work

Executing & paralyzing automatically data-intensive applications

in heterogeneous systems with different enactment engines

NSF-  
Chameleon  
cloud  
1 node –  
40 cores

## Maximizing performance

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

# dispel4py Performance Evaluations

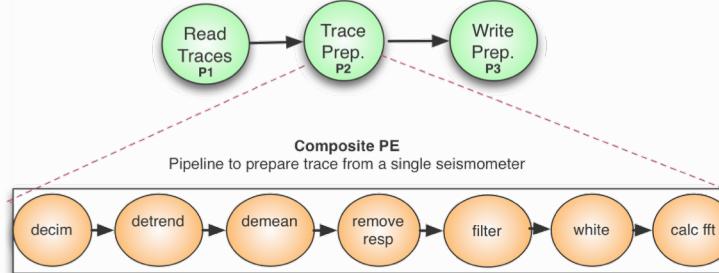


Computing Resources	Terracorrelator	SuperMuc	Amazon EC2	EDIM1	Processor and Network Speed are the lowest
Type	Shared-memory	Cluster	Cloud	Cloud	
Enactment Systems	MPI, multi	MPI, multi	MPI, Storm, multi	MPI, Storm, multi	
Nodes	1	16	18	14	
Cores per Node	32	16	2	4	
Total Cores	32	256	36	14	
Memory	2TB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

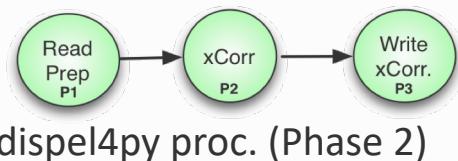
# dispel4py Performance Evaluations



## Data-Intensive Application



## dispel4py preproc. (Phase 1)



1000 stations  
Input 150MB  
Output 39GB

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

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

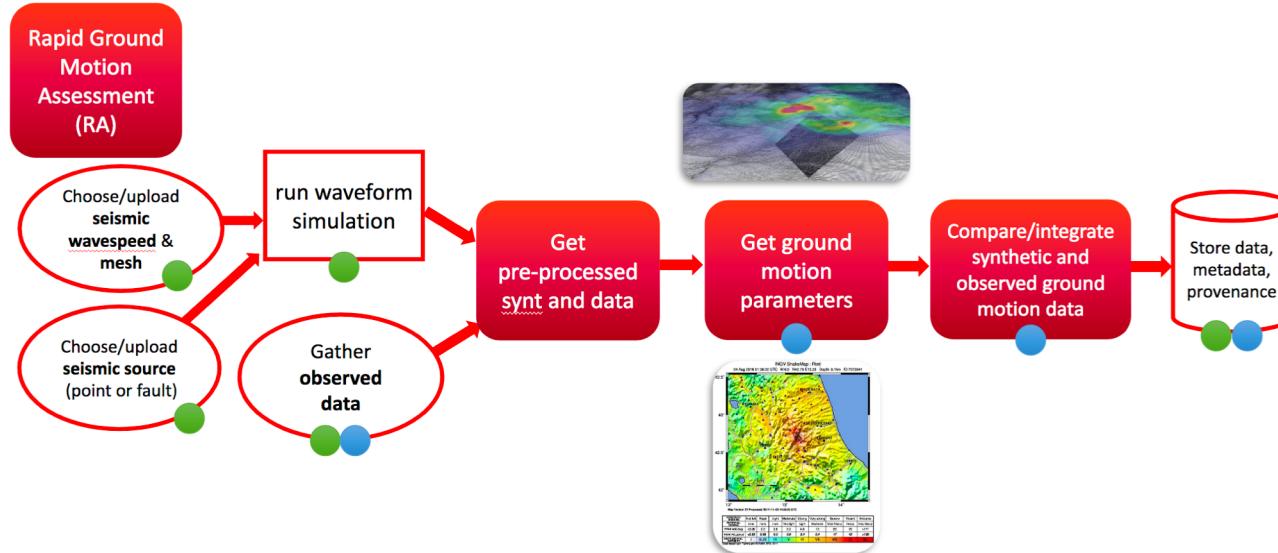
map to different enactment engines at runtime

## 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.



## Aims:

Quickly analyse earthquakes

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.

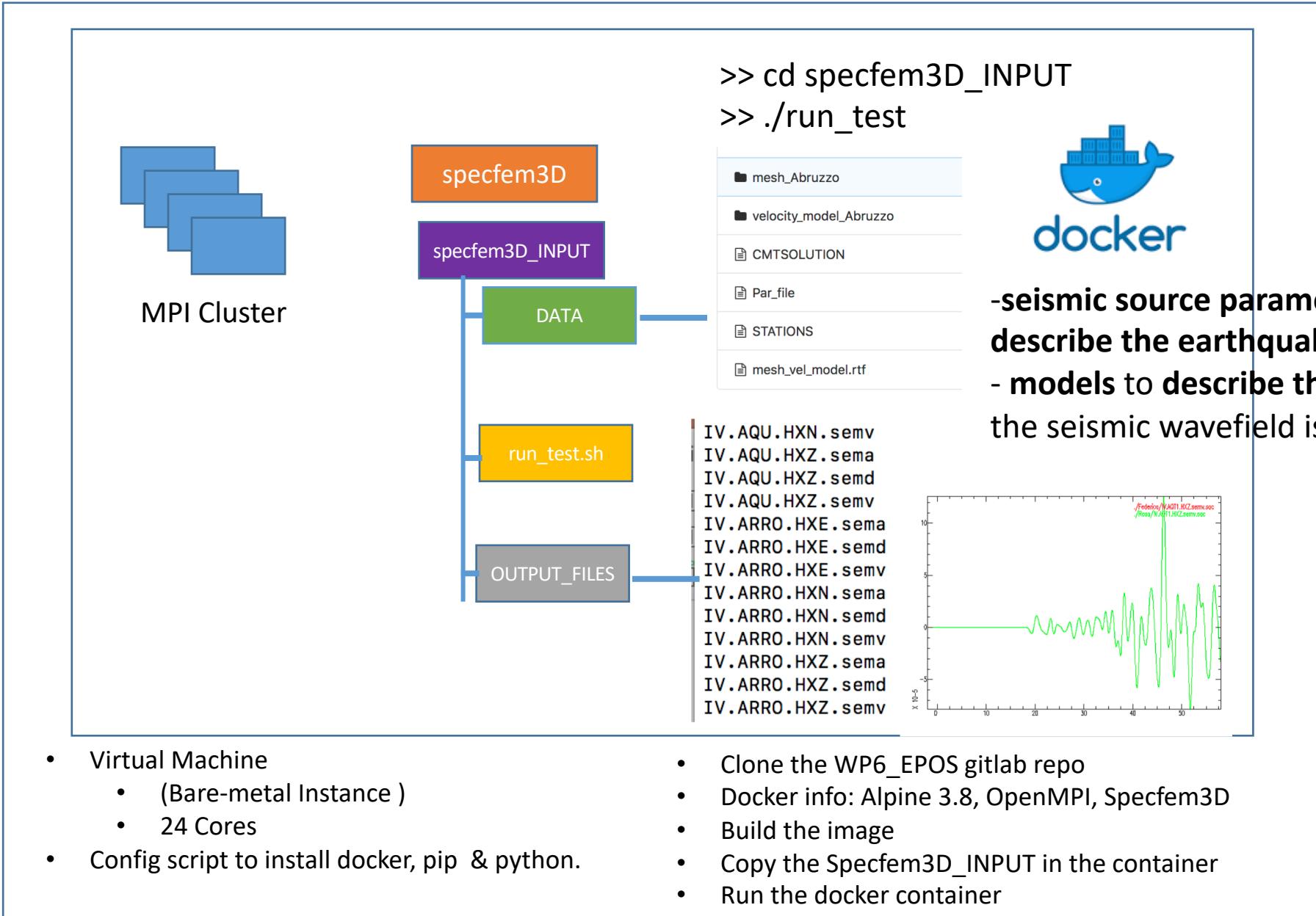
Scale them up:

- Run the workflows using HPC/Cloud
  - \*\* Using dispel4py parallel mappings

# Seismic Waveform Simulation: Specfem3D + MPI cluster

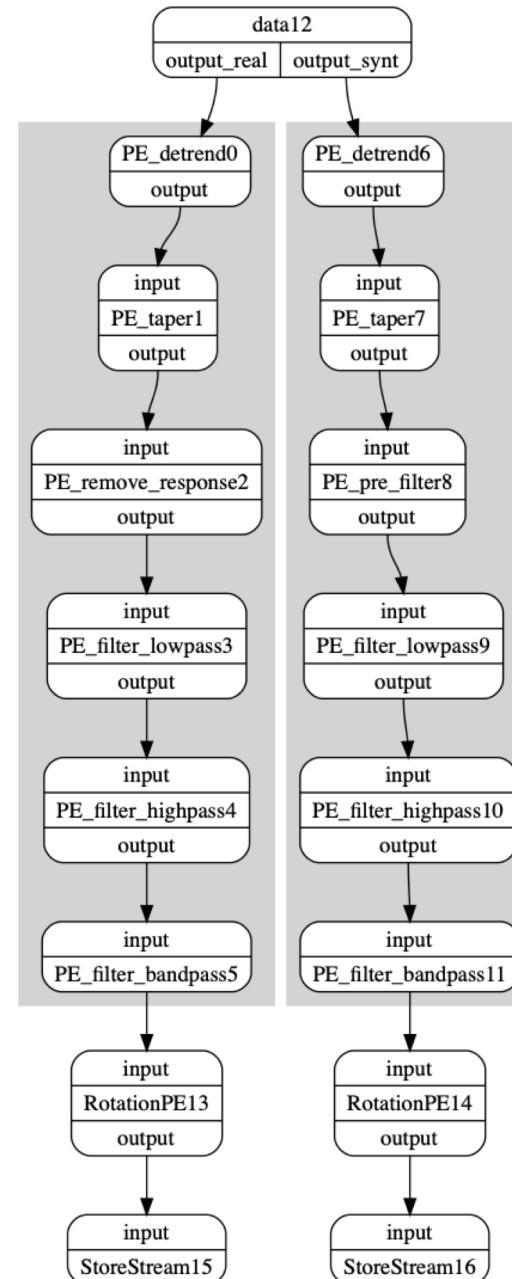
Test Case: RA

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



Similar preprocessing steps  
in synthetic and observed data

Test Case: RA

Get  
pre-processed  
synt and data

# Get Pre-Preprocessed Synthetic and Data



run\_preprocess\_misfit.sh 175 Bytes

```
1 #!/bin/bash
2
3 export PYTHONPATH=$PYTHONPATH
4 export MISFIT_PREP_CONFIG="pr
5 echo $MISFIT_PREP_CONFIG
6 dispel4py simple create_misfi
7
```

data

- IV.ARRO..EHE.mseed
- IV.ARRO..HN.mseed
- IV.ARRO..HZ.mseed

synth

- IV.ARRO.HXE.sema
- IV.ARRO.HXE.semfd
- IV.ARRO.HXE.semfv
- IV.ARRO.HXN.sema
- IV.ARRO.HXN.semfd
- IV.ARRO.HXN.semfv
- IV.ARRO.HXZ.sema
- IV.ARRO.HXZ.semfd
- IV.ARRO.HXZ.semfv

output

misfit\_input.json 848 Bytes

```
1 {
2     "data": [
3         {
4             "input": {
5                 "data_dir": "/Users/rosafilgueira/EPCC/DARE/WP6/test/MISFIT_RA/misfit_data/data",
6                 "synt_dir": "/Users/rosafilgueira/EPCC/DARE/WP6/test/MISFIT_RA/misfit_data/synth",
7                 "events": "/Users/rosafilgueira/EPCC/DARE/WP6/test/MISFIT_RA/misfit_data/events_simulation_CI_CI_test_0_1507128030823",
8                 "event_id": "smi:webservices.ingv.it/fdsnws/event/1/query?eventId=1744261",
9                 "stations_dir": "/Users/rosafilgueira/EPCC/DARE/WP6/test/MISFIT_RA/misfit_data/stations",
10                "output_dir" : "/Users/rosafilgueira/EPCC/DARE/WP6/test/MISFIT_RA/misfit_data/output",
11                "network": [
12                    "IV"
13                ],
14                "station": [
15                    "ARRO"
16                ]
17            }
18        }
19    ]
20}
```

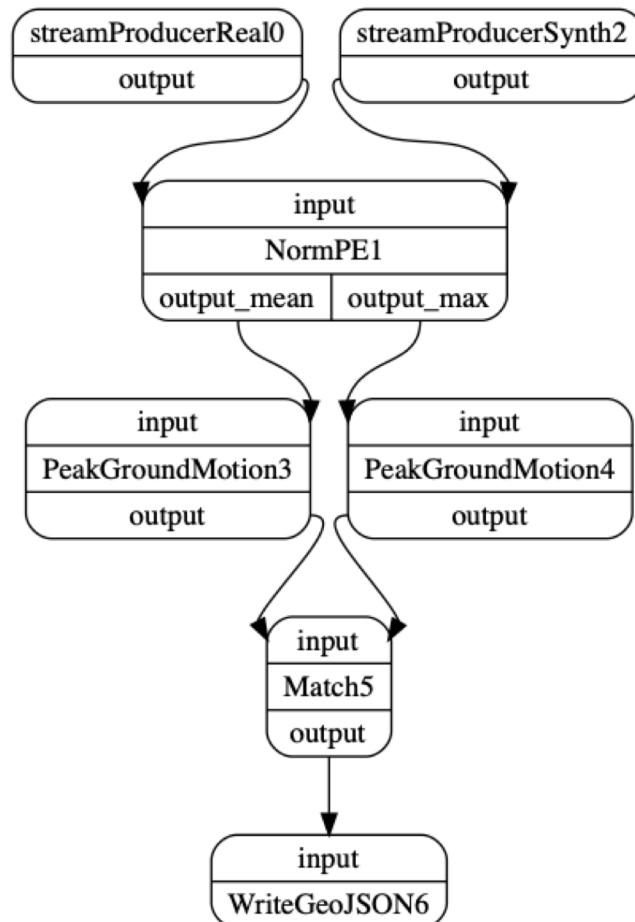
IV.ARRO.HXZ.synth  
IV.ARRO.HXR.synth  
IV.ARRO.HXT.synth  
IV.ARRO.EHZ.data  
IV.ARRO.EHR.data  
IV.ARRO.EHT.data

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

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



# Get and Compare Ground Motion Parameters



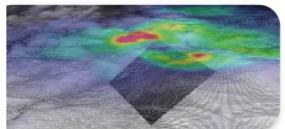
**Ground motion parameters:**

Peak ground values of displacement, velocity and acceleration.

Two types of normalisation – Mean & Max

Two set of PGM outputs – Max & Mean

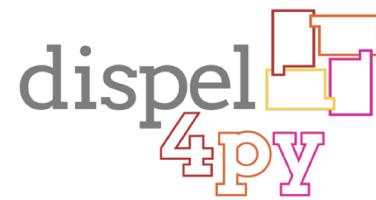
Test Case: RA



Get ground motion parameters

Compare/integrate synthetic and observed ground motion data

# Get and Compare Ground Motion Parameters



run\_RA.sh 210 Bytes

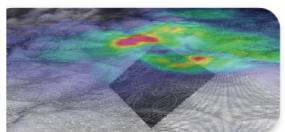
Lock Edit Web IDE Replace Delete

```
1 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

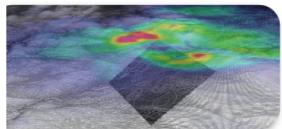
# Get and Compare Ground Motion Parameters



Peak ground values of  
Displacement (PGD),  
Velocity (PGV)  
Acceleration (PGA)

output

Test Case: RA

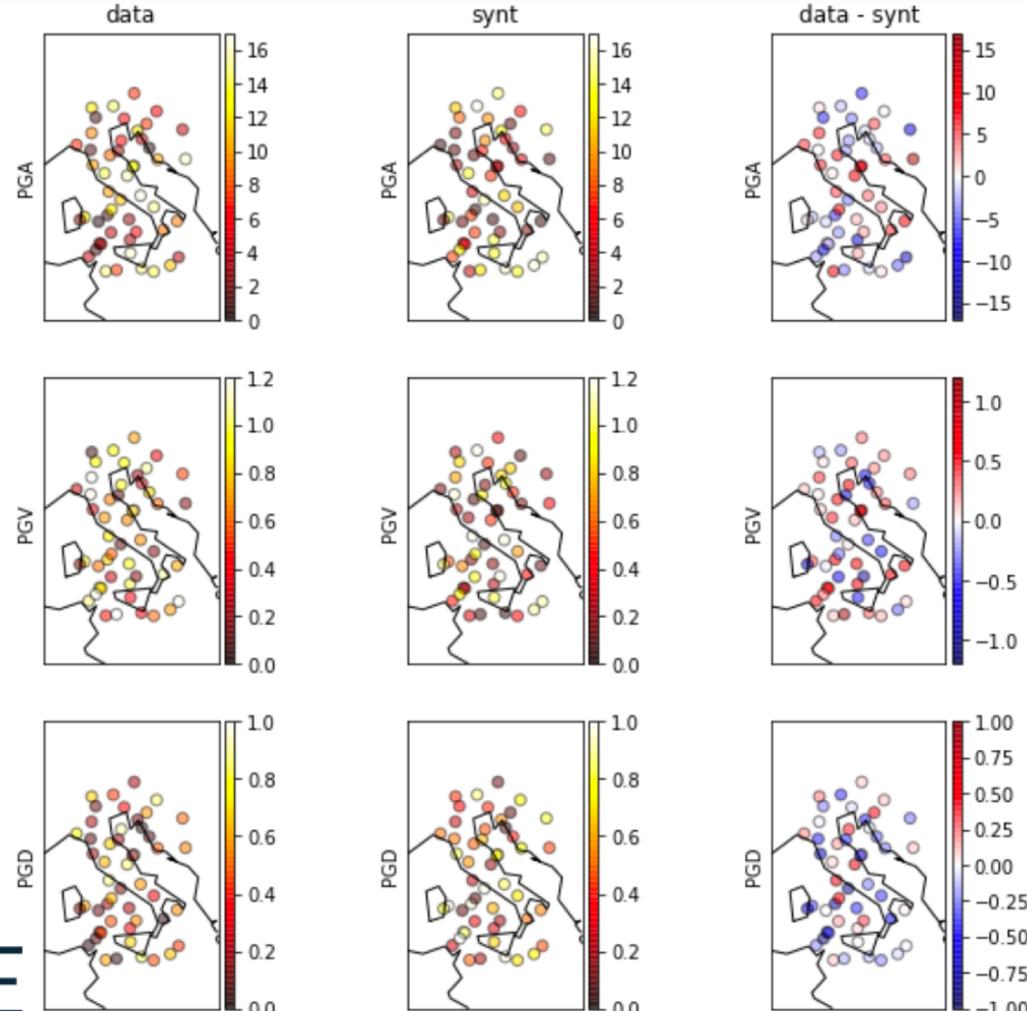
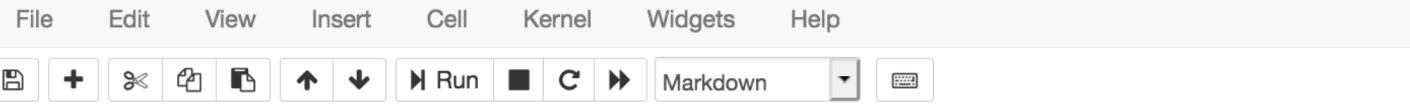


Get ground  
motion  
parameters

Compare/integrate  
synthetic and  
observed ground  
motion data

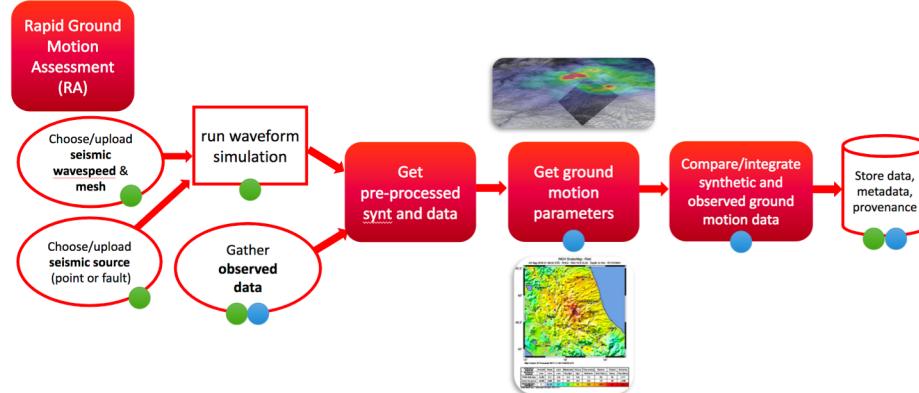
type:	"Feature"	type:	"Feature"
▼ properties:		▼ properties:	
station:	"ARRO"	station:	"ARRO"
▼ data:		▼ data:	
PGD:	0.00003058970011915515	PGD:	0.00003058970011915515
PGV:	0.00015387362683992627	PGV:	0.00015387362683992627
PGA:	0.0008035731527687157	PGA:	0.0008035731527687157
p_norm:	"max"	p_norm:	"mean"
PSA_0.3Hz:	0.000171880777011629	PSA_0.3Hz:	0.000171880777011629
PSA_1.0Hz:	0.0020140831084336057	PSA_1.0Hz:	0.0020140831084336057
PSA_3.0Hz:	0.0009812436987400703	PSA_3.0Hz:	0.0009812436987400703
▼ synt:		▼ synt:	
PGD:	0.000038899272805005096	PGD:	0.000038899272805005096
PGV:	0.00011736045694475592	PGV:	0.00011736045694475592
PGA:	0.00035493665398226155	PGA:	0.00035493665398226155
p_norm:	"max"	p_norm:	"mean"
PSA_0.3Hz:	0.0003942182089108563	PSA_0.3Hz:	0.0003942182089108563
PSA_1.0Hz:	0.0007176179285863899	PSA_1.0Hz:	0.0007176179285863899
PSA_3.0Hz:	0.0003707452407479844	PSA_3.0Hz:	0.0003707452407479844
▼ difference:		▼ difference:	
PGD:	-0.000008309572685849946	PGD:	-0.000008309572685849946
PGV:	0.000036513169895170355	PGV:	0.000036513169895170355
PGA:	0.0004486364987864541	PGA:	0.0004486364987864541
PSA_0.3Hz:	-0.0002223374318992273	PSA_0.3Hz:	-0.0002223374318992273
PSA_1.0Hz:	0.0012964651798472158	PSA_1.0Hz:	0.0012964651798472158
PSA_3.0Hz:	0.0006104984579920859	PSA_3.0Hz:	0.0006104984579920859
▼ relative_difference:		▼ relative_difference:	
PGD:	-0.2716460983102781	PGD:	-0.2716460983102781
PGV:	0.23729322980834633	PGV:	0.23729322980834633
PGA:	0.5583020005592205	PGA:	0.5583020005592205
PSA_0.3Hz:	-1.2935561251517065	PSA_0.3Hz:	-1.2935561251517065
PSA_1.0Hz:	0.6436999418834826	PSA_1.0Hz:	0.6436999418834826
PSA_3.0Hz:	0.6221680289778919	PSA_3.0Hz:	0.6221680289778919
▼ geometry:		▼ geometry:	
type:	"Point"	type:	"Point"
coordinates:	[]	coordinates:	[]

# RA Maps



Waveform propagation snapshots and maps of ground motion parameters are fundamental for a visual representation of the earthquake

# Rapid Ground Motion Assessment (RA) - EPOS



dispel4py +

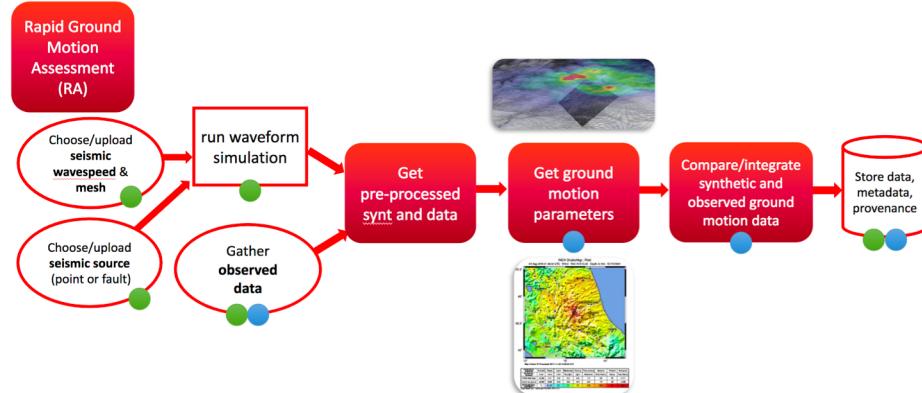


→ semantics and description

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 +



→ 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

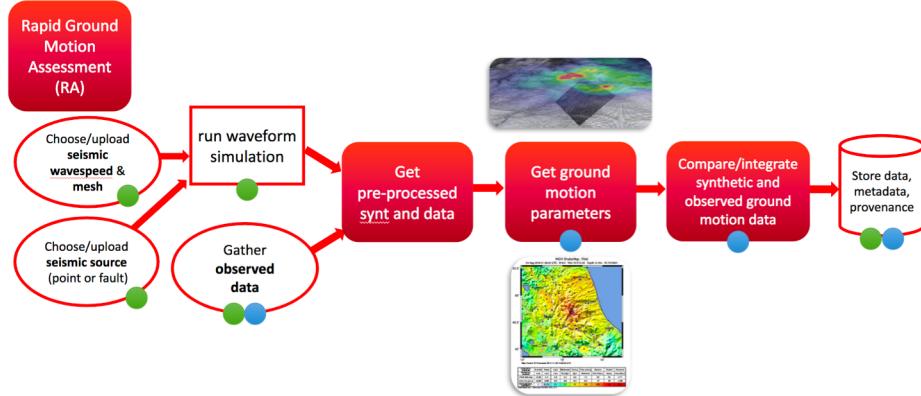
```
1 script:
2   class: File
3   path: /Users/rosofilgueira/EPCC/DARE/WP6/test/MISFIT_RA/dispel4py_RA.pgm_story.py
4
5
6 input: '{"streamProducerReal": [ {"input": "/Users/rosofilgueira/EPCC/DARE/WP6/test/MI
```

dispel4py-RA-pgm\_story.cwl 437 Bytes

```
1 #!/usr/bin/env cwl-runner
2
3 cwlVersion: v1.0
4 class: CommandLineTool
5 baseCommand: [dispel4py, simple]
6 requirements:
7   EnvVarRequirement:
8     envDef:
9       PYTHONPATH: ${inputs.script.dirname}
10
11 inputs:
12   - id: script
13     type: File
14     inputBinding:
15       position: 1
16   - id: input
17     type: string
18     inputBinding:
19       prefix: -d
20       position: 2
21
22 outputs:
23   output:
24     type:
25       type: array
26       items: File
27     outputBinding:
28       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*

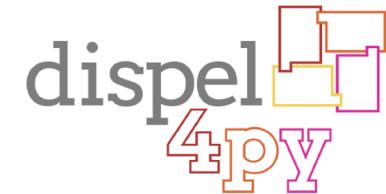
---

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 ()

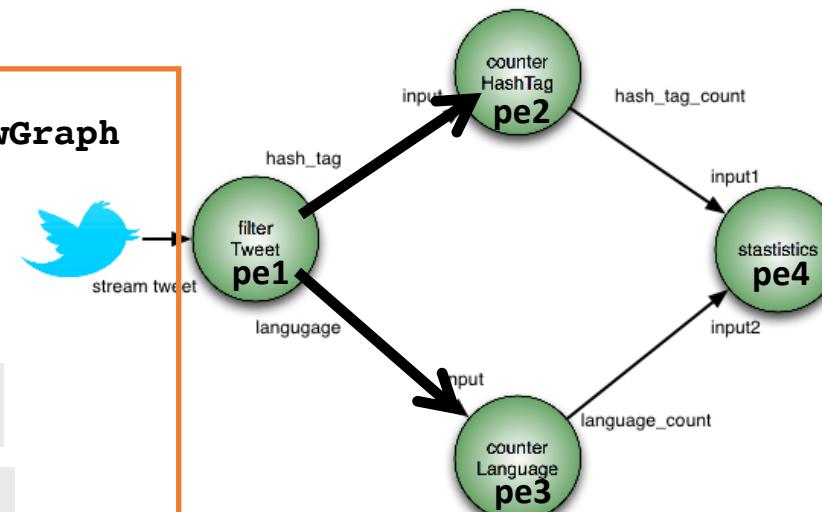
graph = WorkflowGraph ( )

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



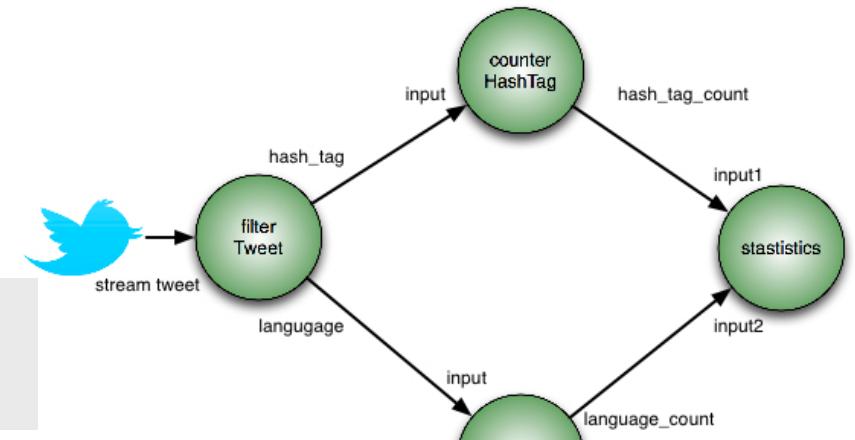
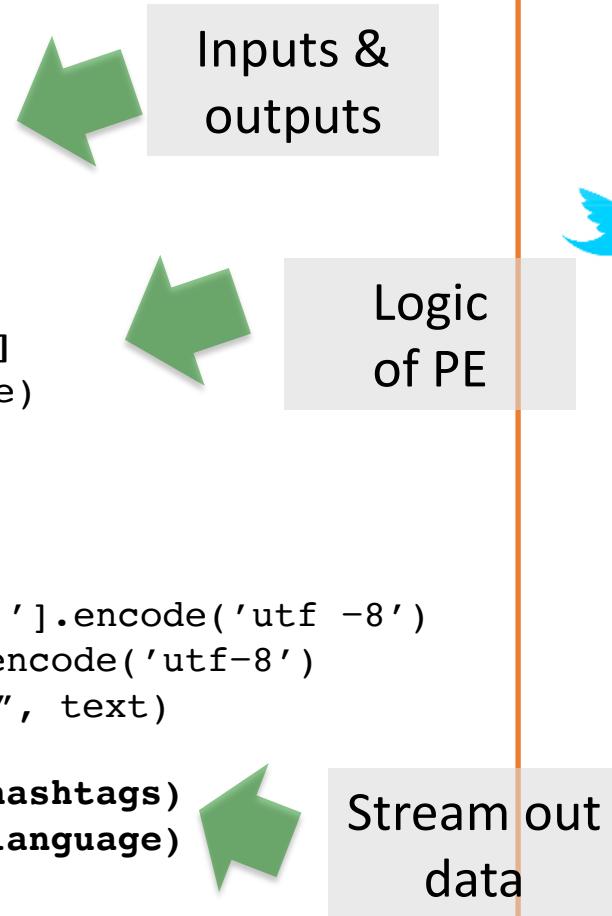
|epcc

# 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)

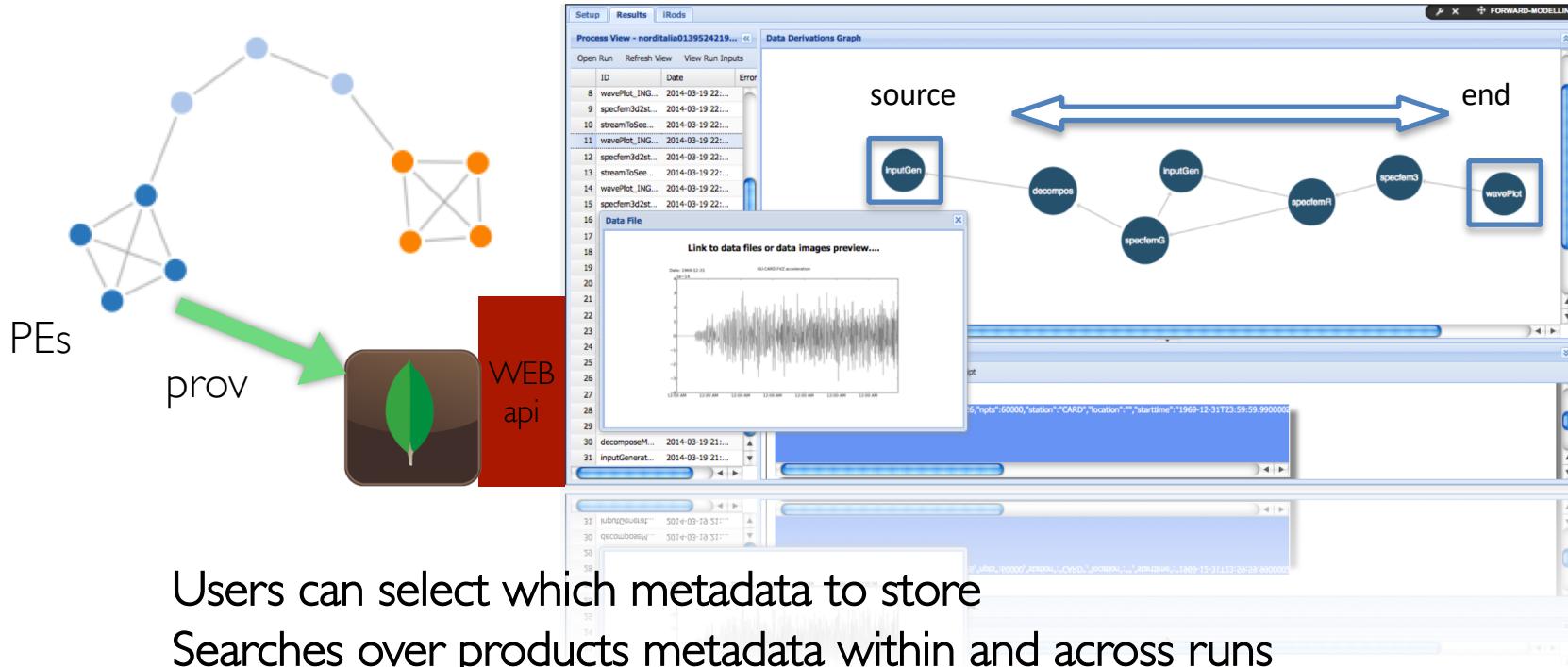
        self.write('hash tag', hashtags)
        self.write('language', language)
```



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.