Virtual Earthquake and seismology Research Community e-science environment in Europe
Project 283543 – FP7-INFRASTRUCTURES-2011-2 – www.verce.eu – info@verce.eu

dispel4py in detail

(dispel4py training)
day 3

3 July 2015, Liverpool

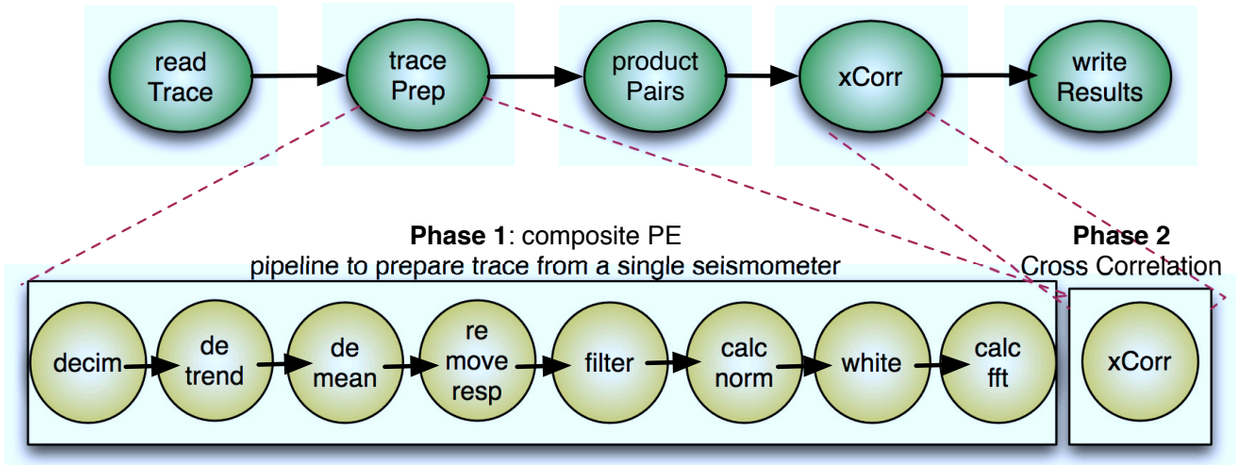
Outline

- Installation and links
- dispel4py workflows
    - Seismology
        - Cross-Correlation
        - Misfit
- Graph examples
- Composite PEs
- Chains
- Groupings
- Mappings to execution platforms

Installation & links

- This is all you need:
  - **pip install dispel4py**
- Web site: http://dispel4py.org/
- GitHub: https://github.com/dispel4py/dispel4py
- Documentation: http://dispel4py.org/documentation/

dispel4py workflows- Seismology- Cross Correlation



- **Phase 1- Preprocess**: Time series data (traces) from seismic stations are preprocessed in parallel
- **Phase 2: Cross-Correlation**: Pairs all of the stations and calculates the cross-correlation for each pair (complexity O(n2)).
- **Input data**: 1000 stations as input data (150 MB)
- **Output data**: 499,500 cross-correlations (39GB)

11th IEEE eScience 2015 paper

dispel4py: An Agile Framework for Data-Intensive eScience

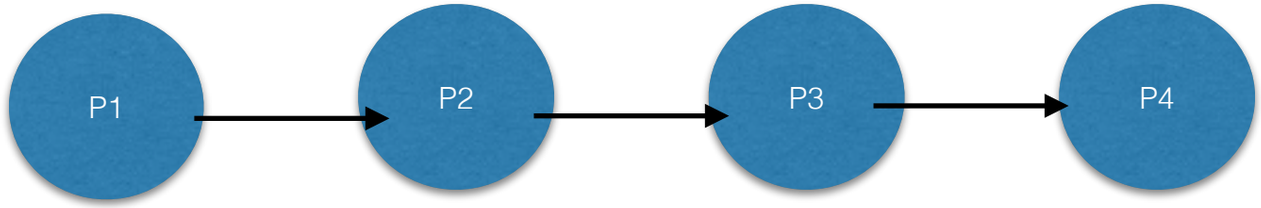
dispel4py workflows- Seismology- Misfit

Later …..

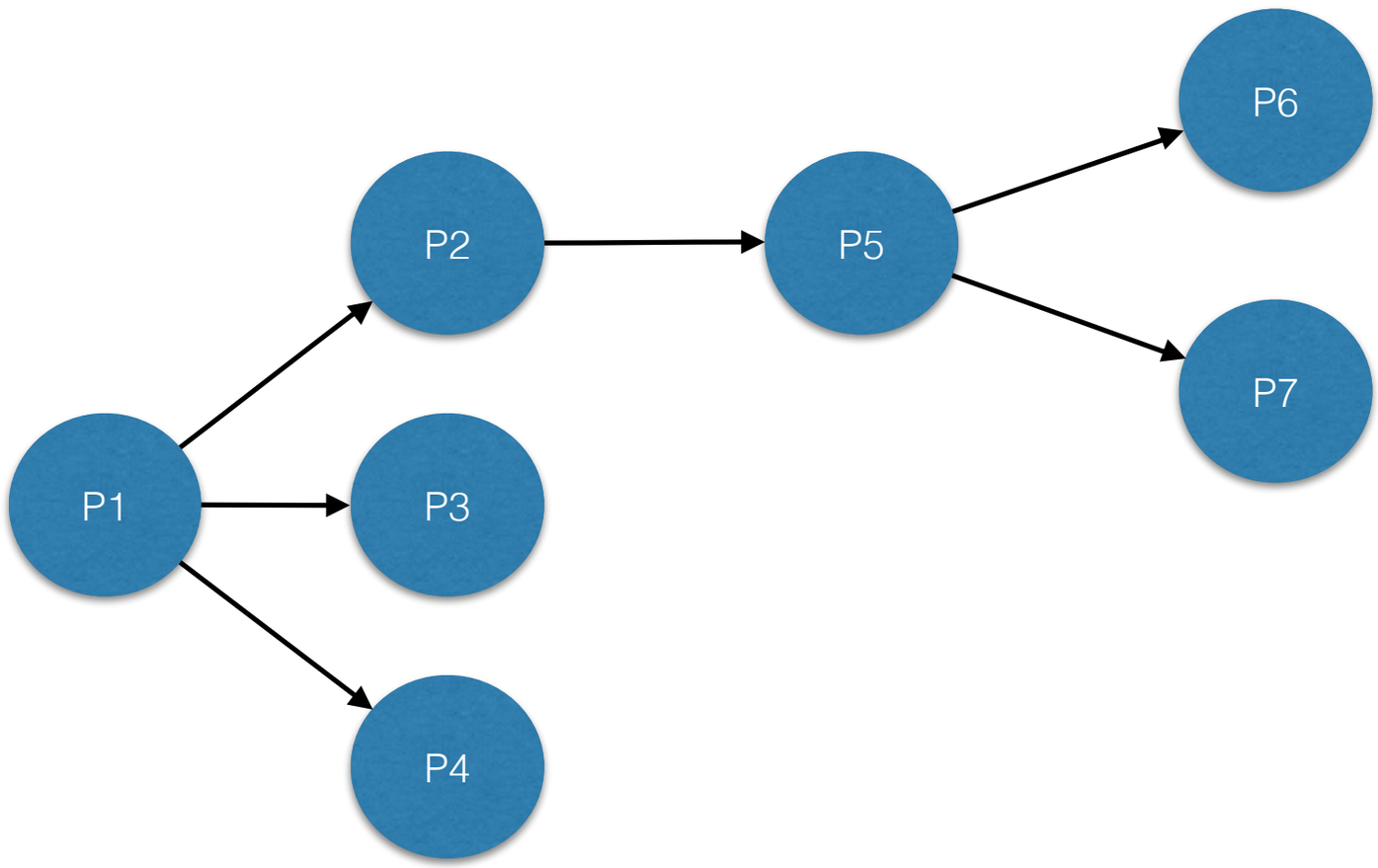
Inputs and outputs

- How many pieces of data does the PE combine?
  - Transformation: one input (e.g. normalisation)
  - Product or Join: two inputs (e.g. cross-correlation)
  - And many others!
- What is the rate of processing?
  - One output per input (transformation, filter)
  - Aggregation of data (e.g. stacking)

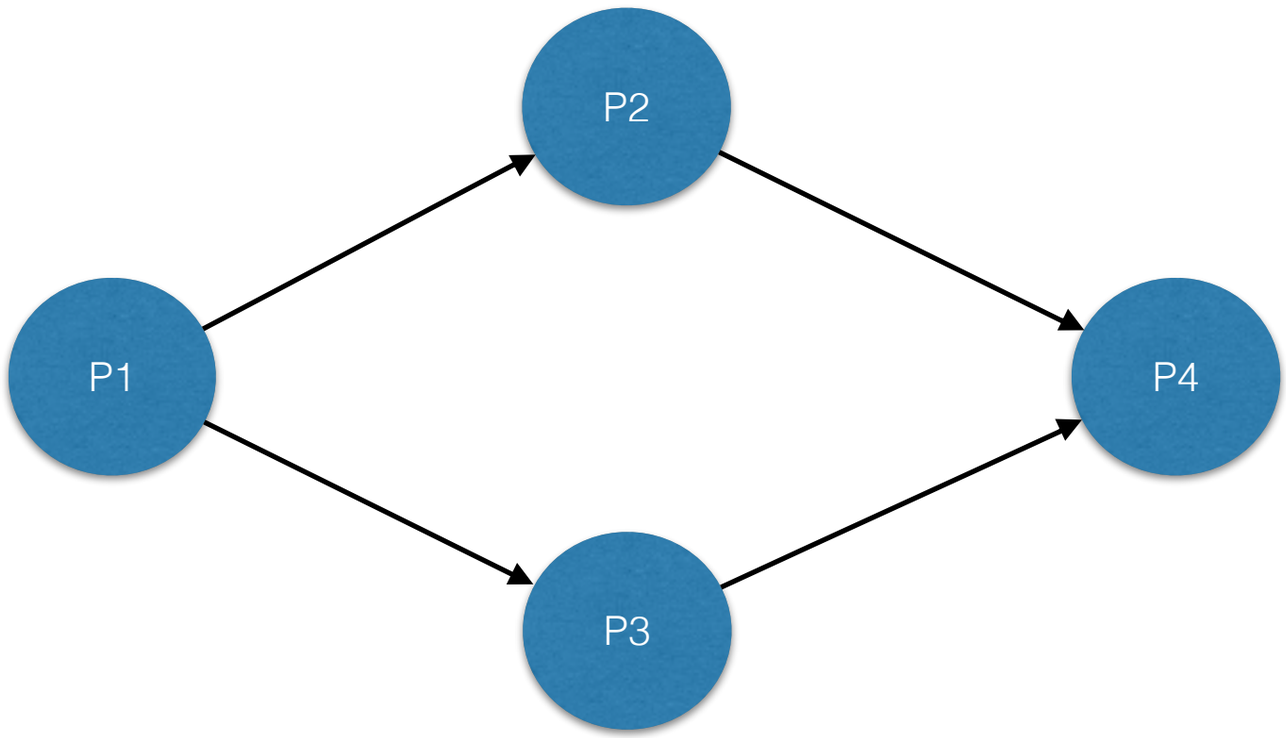
Pipeline



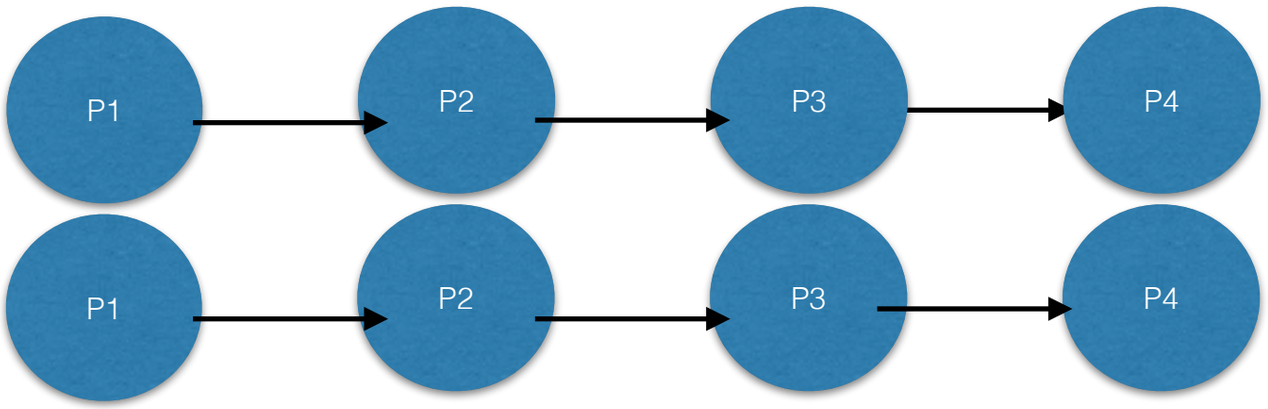
Tree



Split and Merge

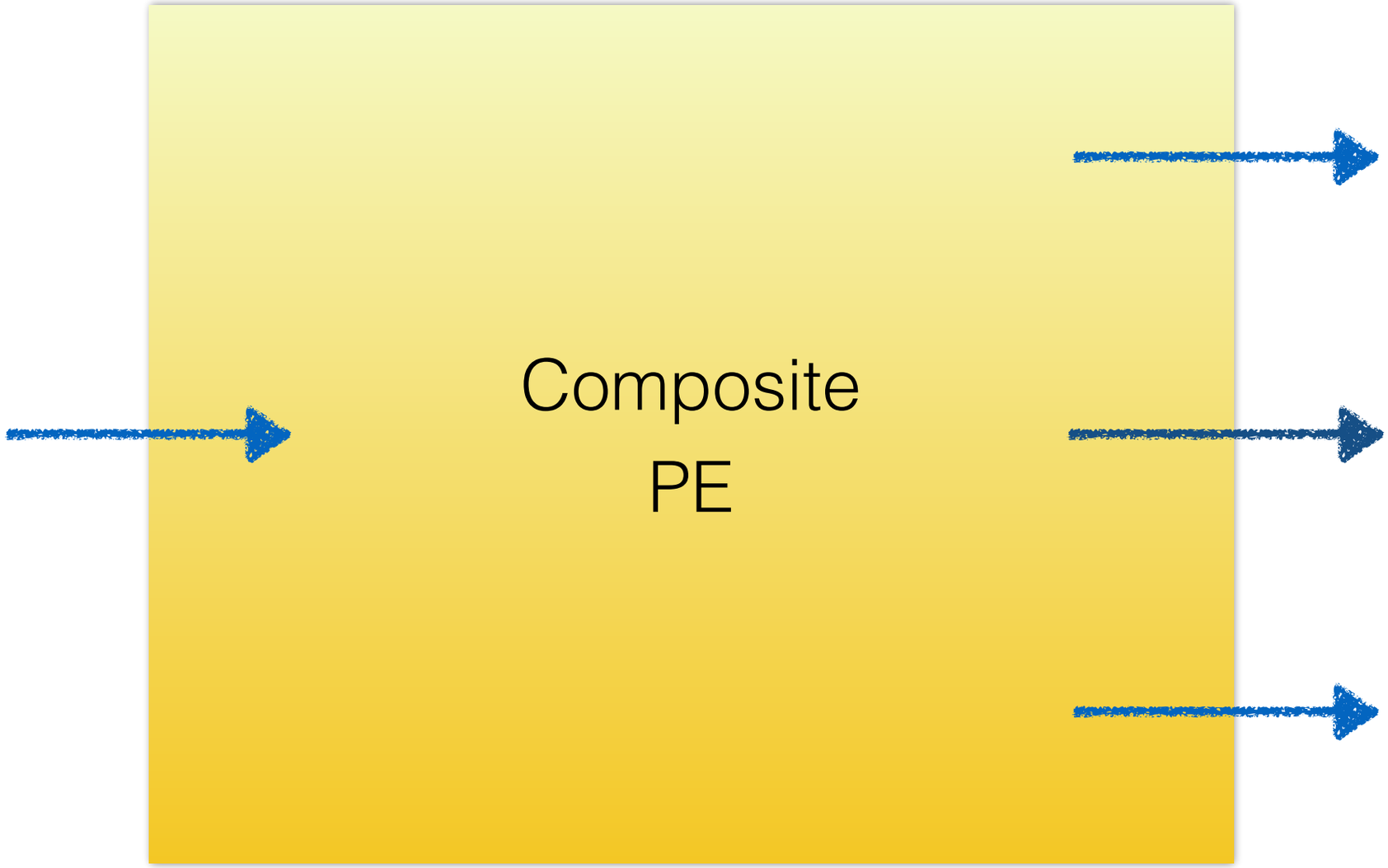


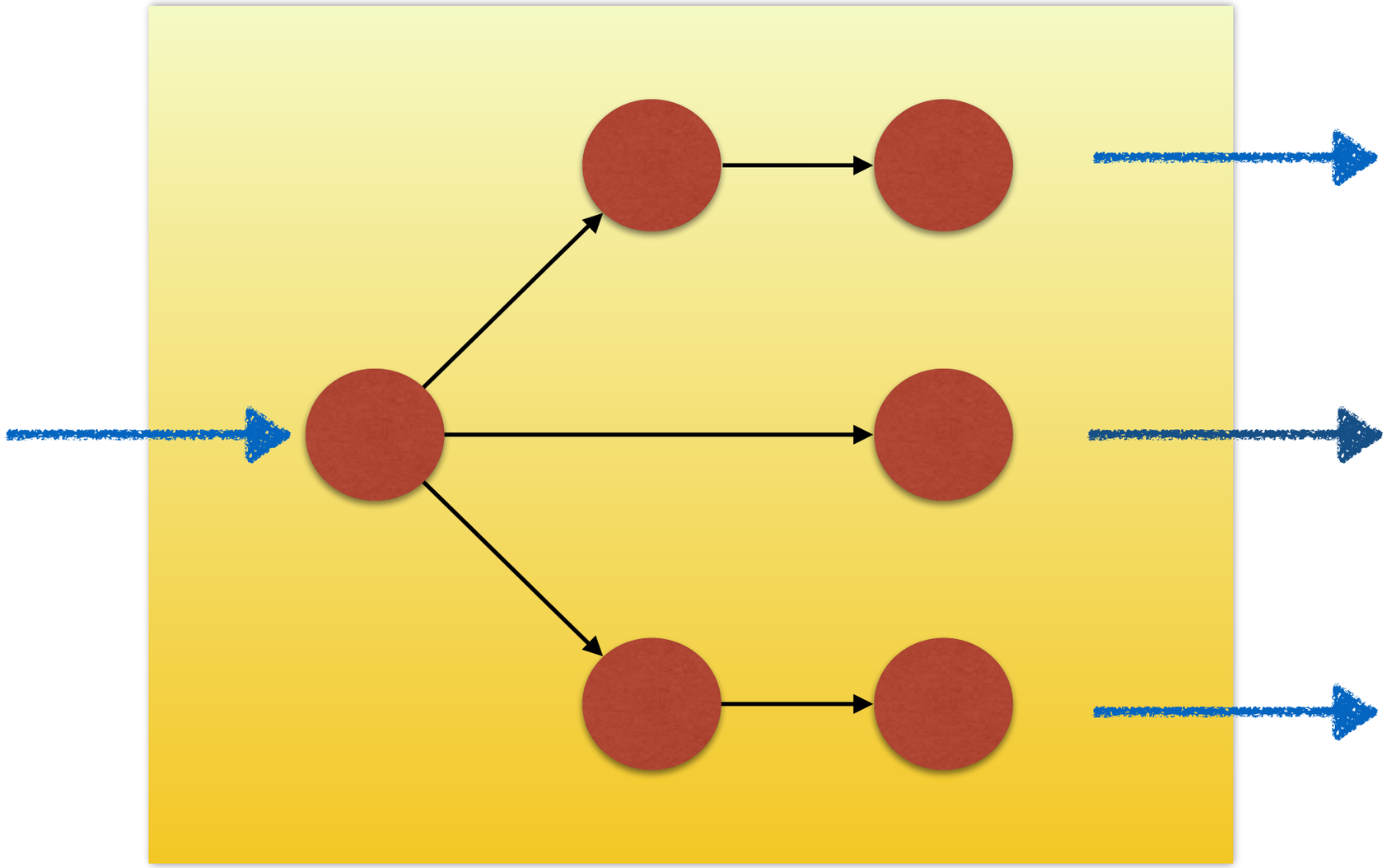
Unconnected

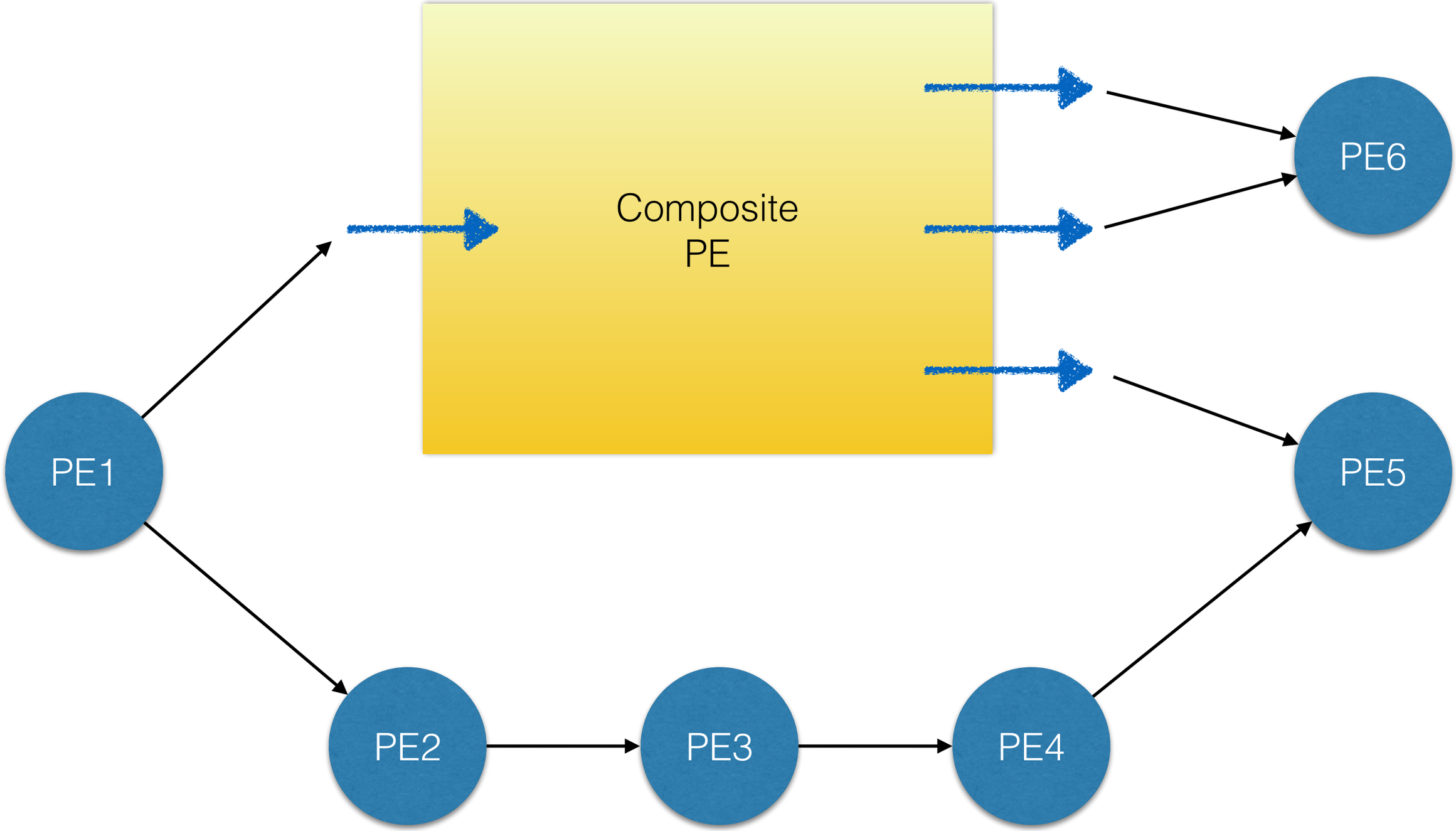


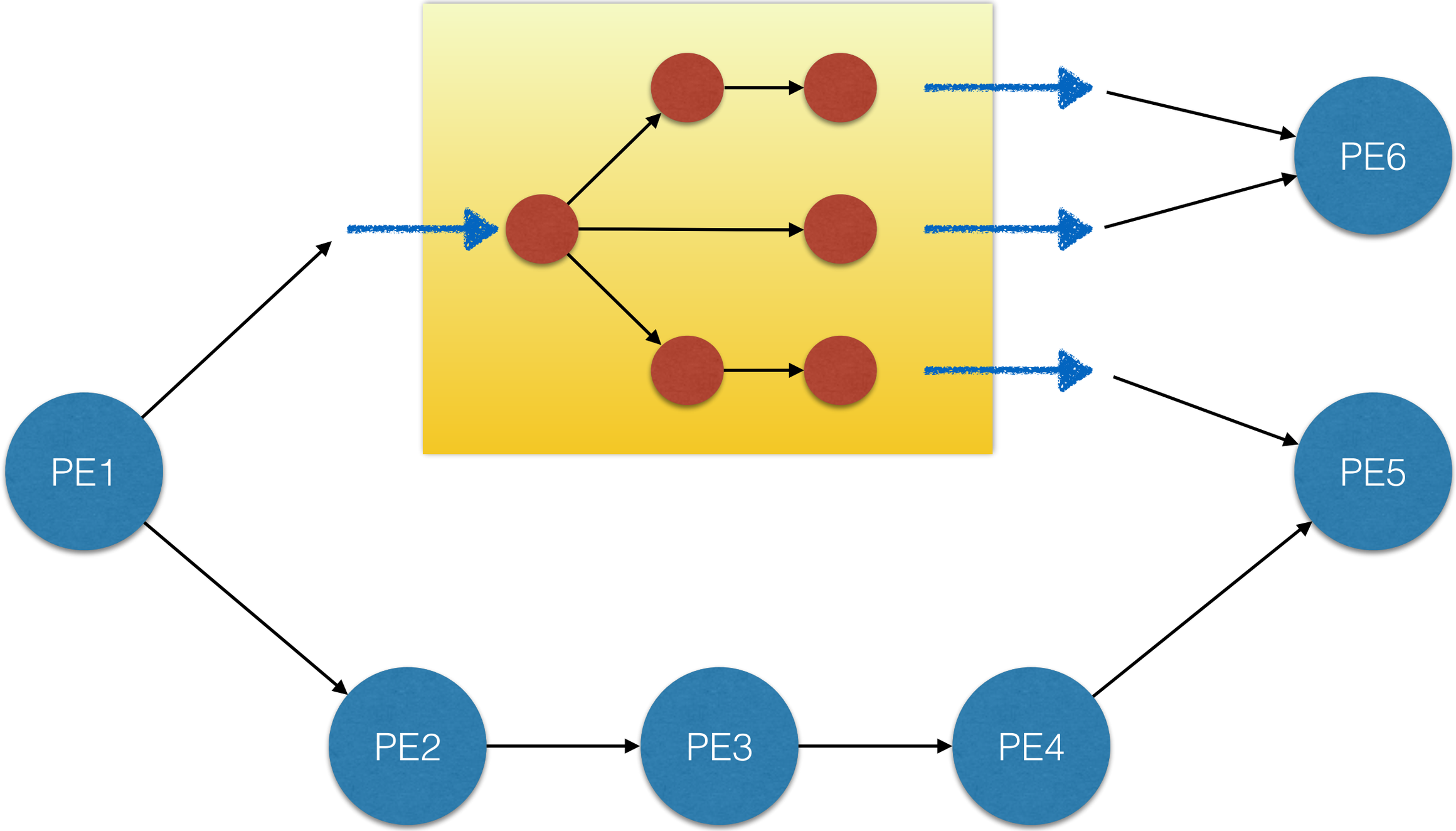
Composite PEs

- A composite PE is a nested graph
- Looks like a PE but contains other PEs
- Hides the complexity of an underlying process
- When creating a graph, a composite PE is treated like any other PE


Composite
PE




Composite
PE
PE1
PE2
PE3
PE4
PE5
PE6


PE1
PE2
PE3
PE4
PE5
PE6

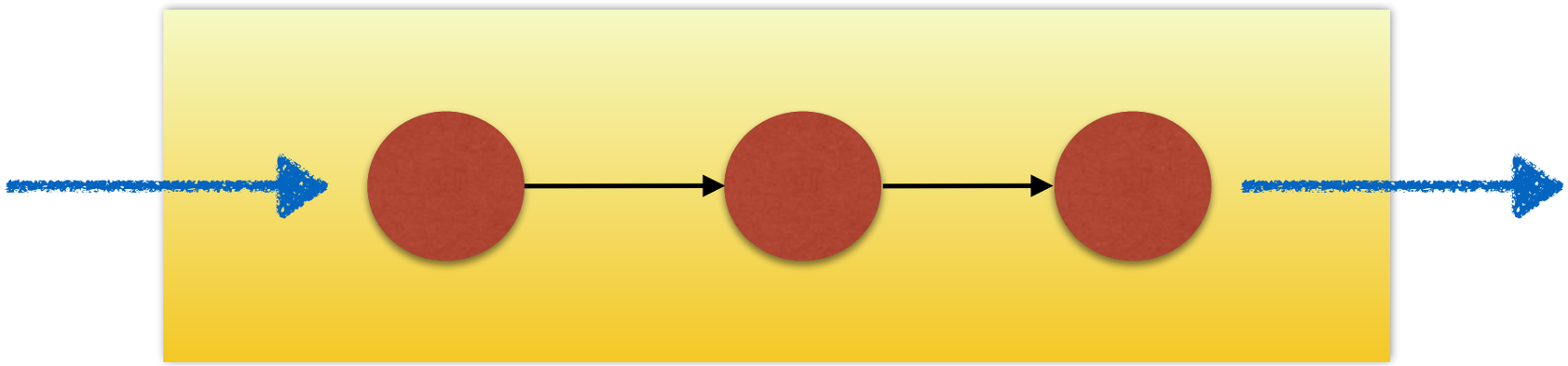
Why composite PEs?

dispel4py provides a utility to create a composite PE with a chain of processing PEs:

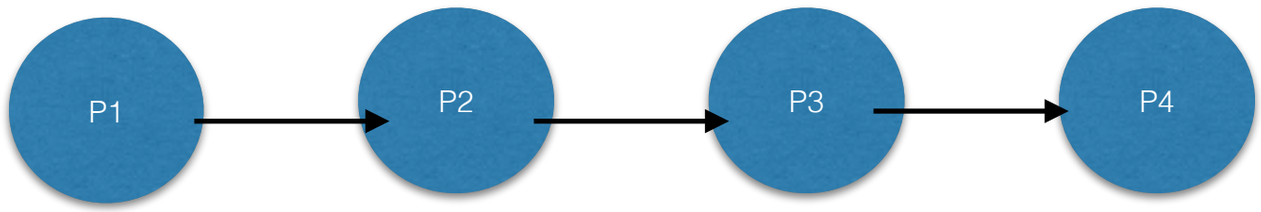
1. Only implement the processing functions (process one, return one)
2. Create a list that contains the functions in the order that they are to be applied
3. Pass this list to the utility and receive a composite PE in return

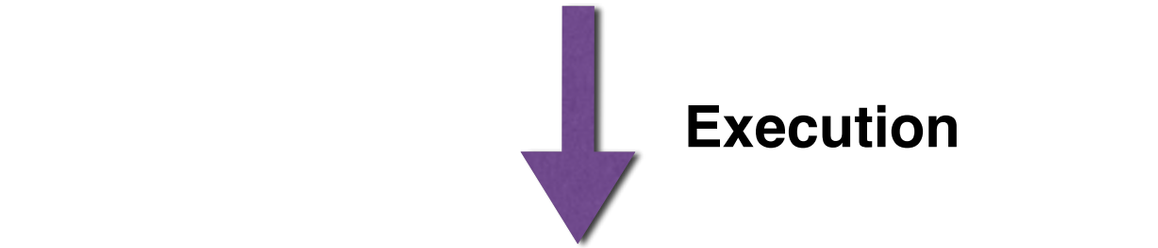
```
def decimate(data, sps):
    st = data[0]
    st.decimate(int(st[0].stats.sampling_rate/sps))
    return st
def detrend(data):
    st = data[0]
    st.detrend('simple')
    return st
def demean(data):
    st = data[0]
    st.detrend('demean')
    return st
```

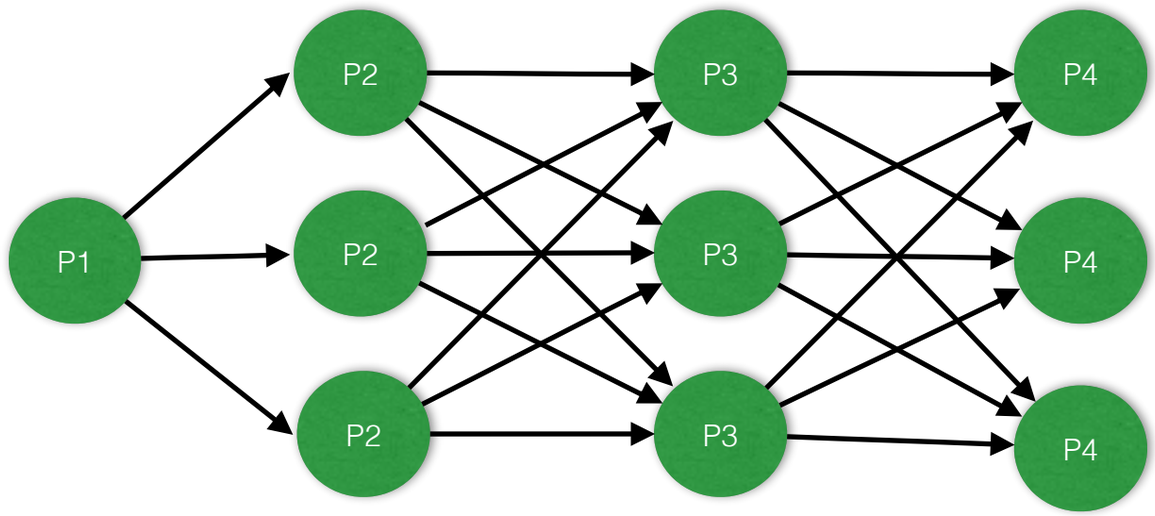
```
compositePE = create_iterative_chain([(decimate, {'sps':4}),
detrend, demean])
```



Executing graphs

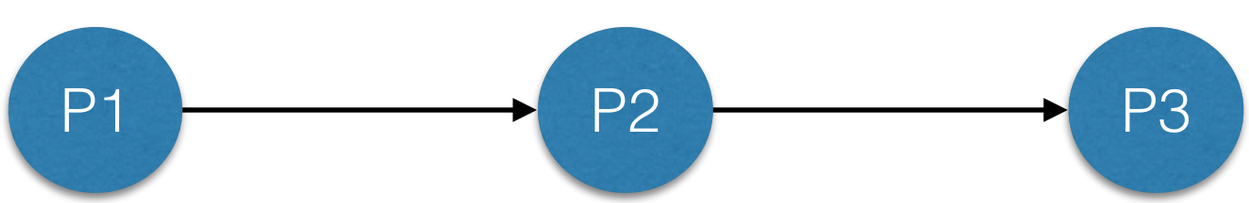




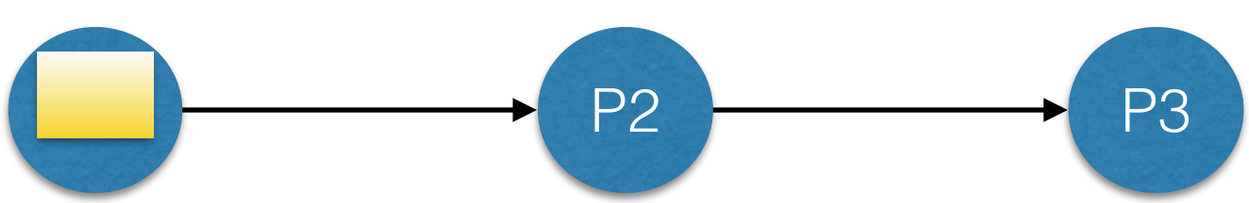


PE Instance

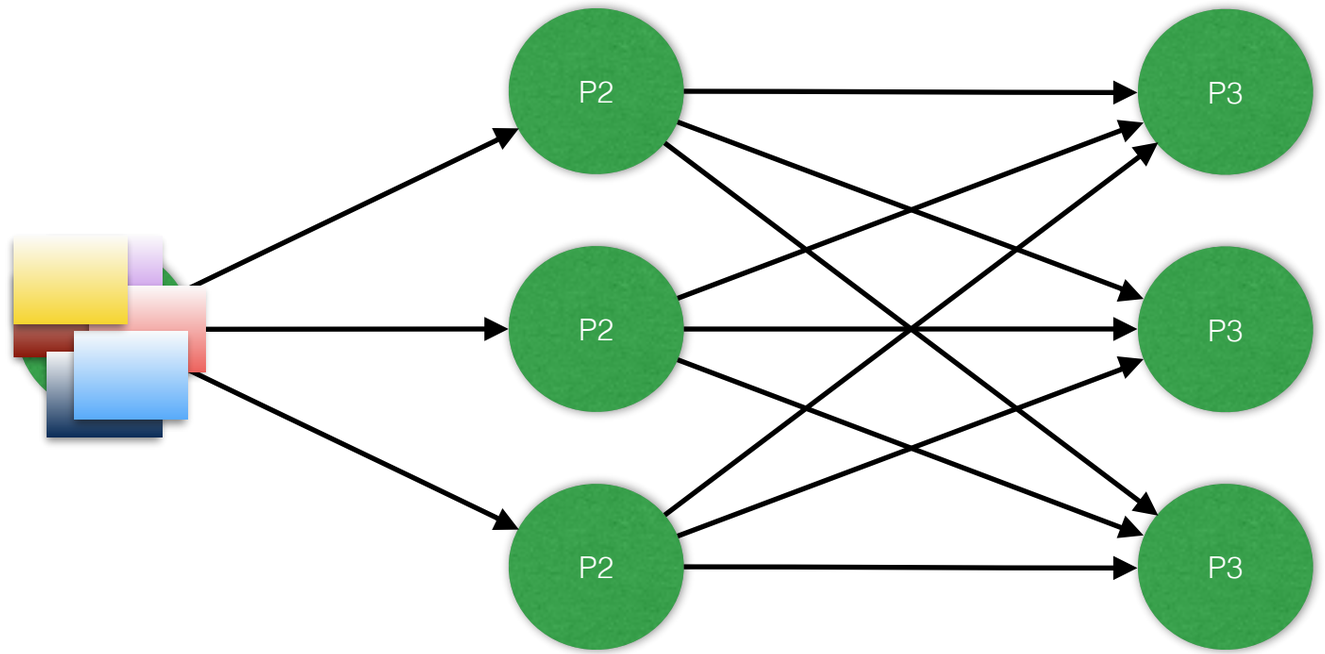
Executing graphs



Executing graphs

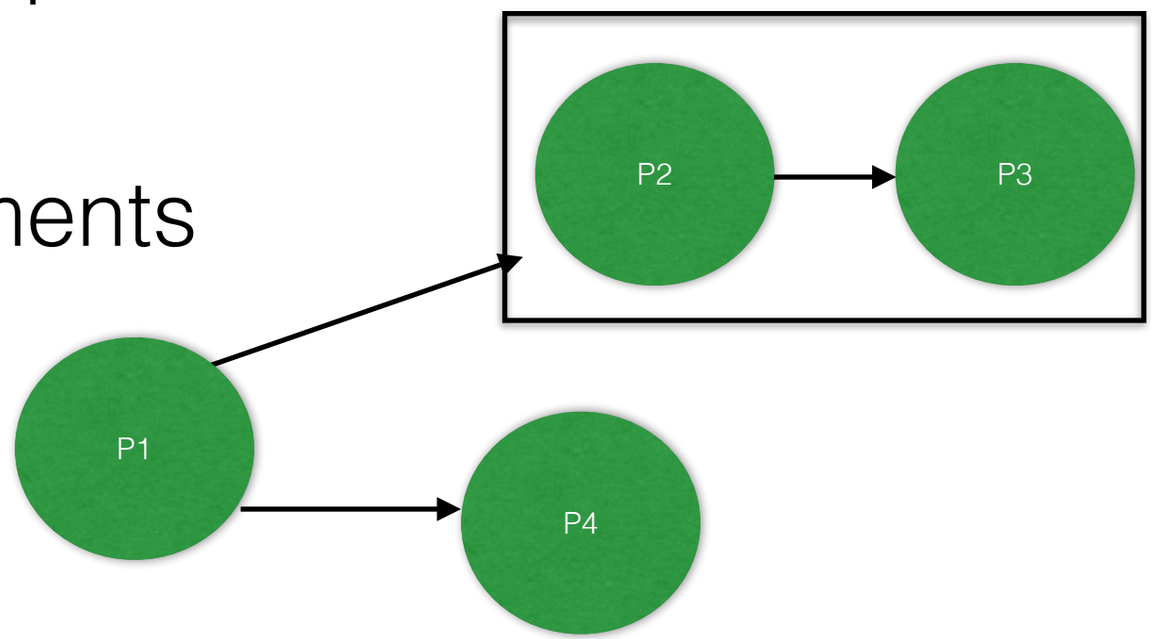


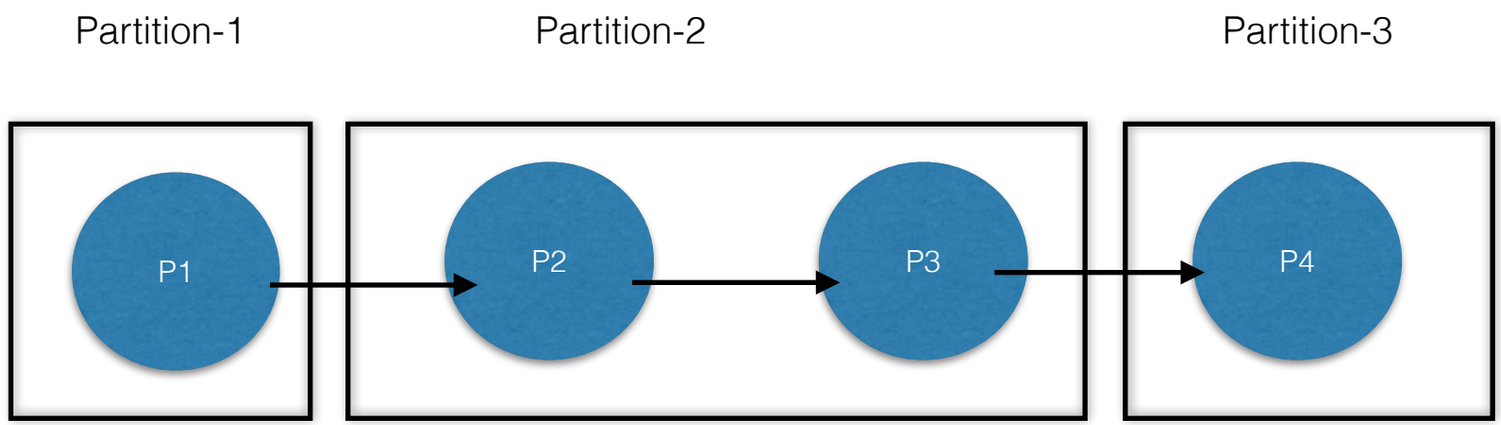
Executing graphs



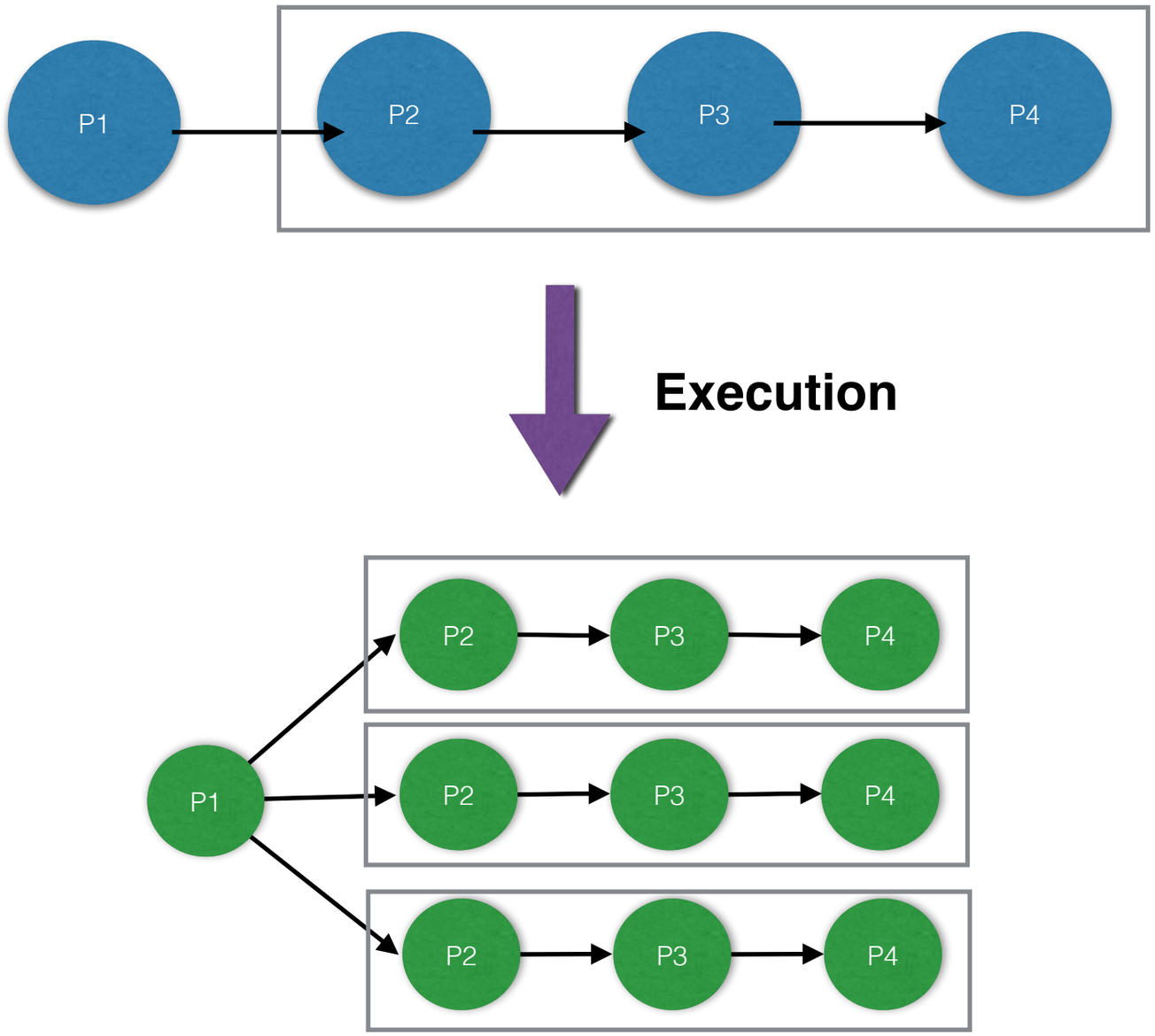
Partitions

- Run several PEs in a single process
- User defined
- Applies to parallel environments





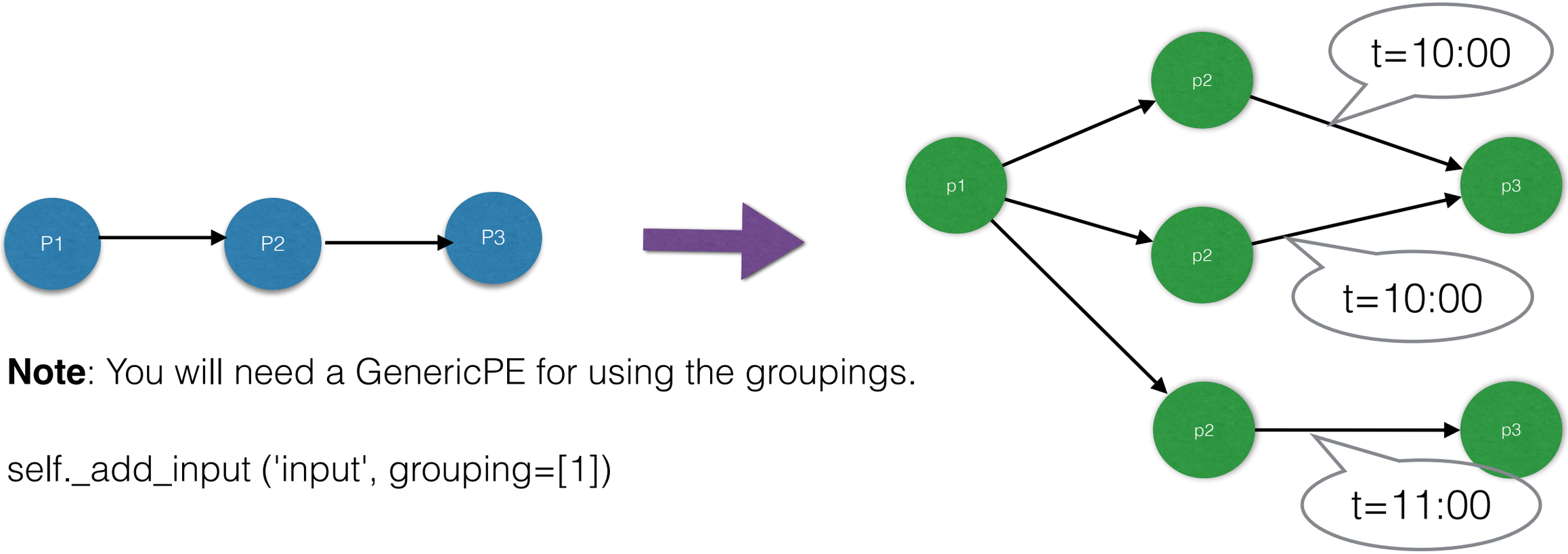
Partitioned Pipeline



Groupings

“Grouping by” a feature (MapReduce)

All data items that satisfy the same feature are guaranteed to be delivered to the same **instance** of a PE



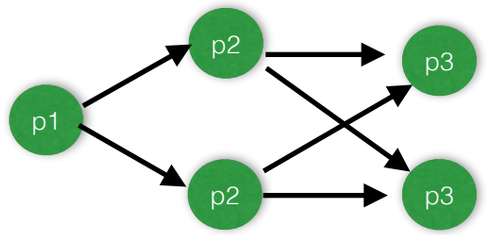
Note: You will need a GenericPE for using the groupings.

self._add_input ('input', grouping=[1])

Other groupings

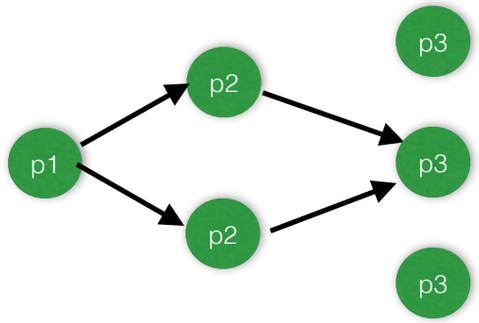
One-To-All

P3 - grouping “all”:
P2 instances send copies of their output data to **all** the connected instances



Global

P3 - grouping “global”:
All the instances of P2 send all the data to **one** instance of P3



Mappings

Simple process

- Sequential mapping for local testing

Multi process

- Parallelism based on processes, using Python's multiprocessing library
- Shared memory

MPI

- Distributed Memory, message-passing parallel programming model
- Practical, portable, efficient, flexible and stable
- Many HPC centres support it, and it has been widely used in the HPC community

STORM:

- Distributed Real-Time computation System
- Fault-tolerant and scalable

Running graphs

Sequential mapping
>> dispel4py simple <name_dispy_graph> <-f input_file in JSON format>
E.g: dispel4py simple dispel4py.examples.graph_testing.pipeline_test

Multi-process mapping
>> dispel4py multi <name_dispy_graph> -n <number mpi_processes> <-f input_file in JSON format> <-s>
E.g : dispel4py multi dispel4py.examples.graph_testing.pipeline_test -n 6

MPI mapping
>> mpiexec -n <number mpi_processes> dispel4py mpi <name_dispy_graph> <-f input_file in JSON format> <-s>
E.g : mpiexec -n 6 dispel4py mpi dispel4py.examples.graph_testing.pipeline_test

Useful dispel4py information

- dispel4py commands:
    - -h, --help >> show this help message and exit
    - -a attribute, --attr attribute >> name of graph variable in the module
    - -f inputfile, --file inputfile >> file containing input dataset in JSON format
    - -d inputdata, --data inputdata >> input dataset in JSON format
    - -i iterations, --iter iterations >> number of iterations

Useful dispel4py information

- inputs to the workflow:
  - dispel4py simple workflow.py -d ‘{“PE NAME CLASS" : [{"input" : “Hello World World”}]}’
  - dispel4py simple workflow.py -d ‘{"PE NAME CLASS" : [{"input" : "coordinates.txt"}]}’
  - dispel4py simple workflow.py -f coordinates.txt

Thanks to

* **Malcolm Atkinson**
* **Alessandro Spinuso**

Questions

* **Amy Krause -> a.krause@epcc.ed.ac.uk**
* **Rosa Filgueira -> rosa.filgueira@ed.ac.uk**
* **Malcolm Atkinson -> malcolm.atkinson@ed.ac.uk**
* **Alessandro Spinuso -> Alessandro.Spinuso@knmi.nl**