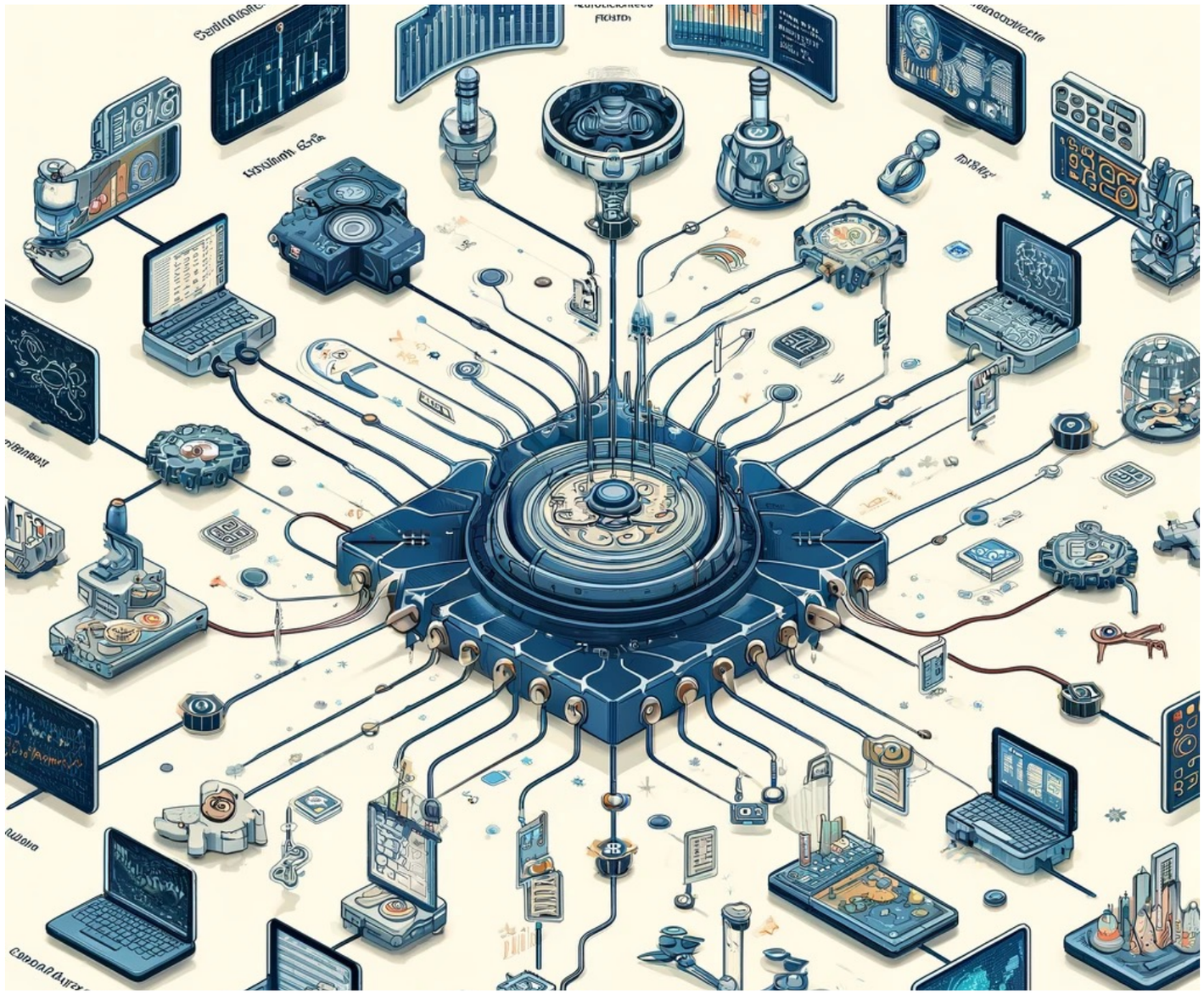
Exploring Scientific Workflows with CWL and dispel4py

Module 3.b

- Dr. Rosa Filgueira
- Lecturer at the School of Computer Science
- University of St Andrews
- rf208@st-andrews.ac.uk
- rosa.filgueira.vicente@gmail.com



Module 3.b – dispel4py Advanced Concepts

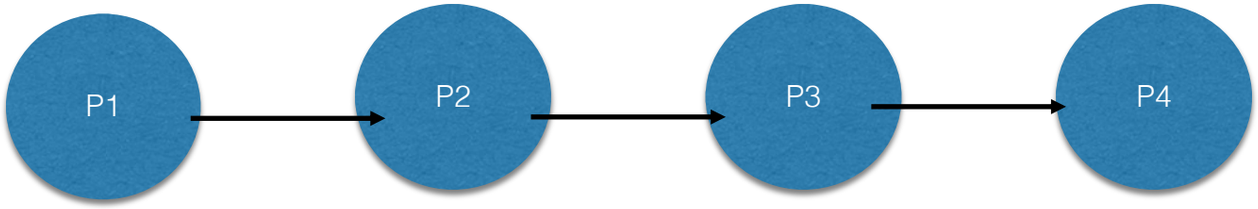
- Inputs and outputs
- Graph examples
- Composite PEs
- Chains
- Groupings
- Mappings to execution platforms
- Running graphs
- Useful dispel4py Information
- RA example
- Provenance

Inputs and outputs

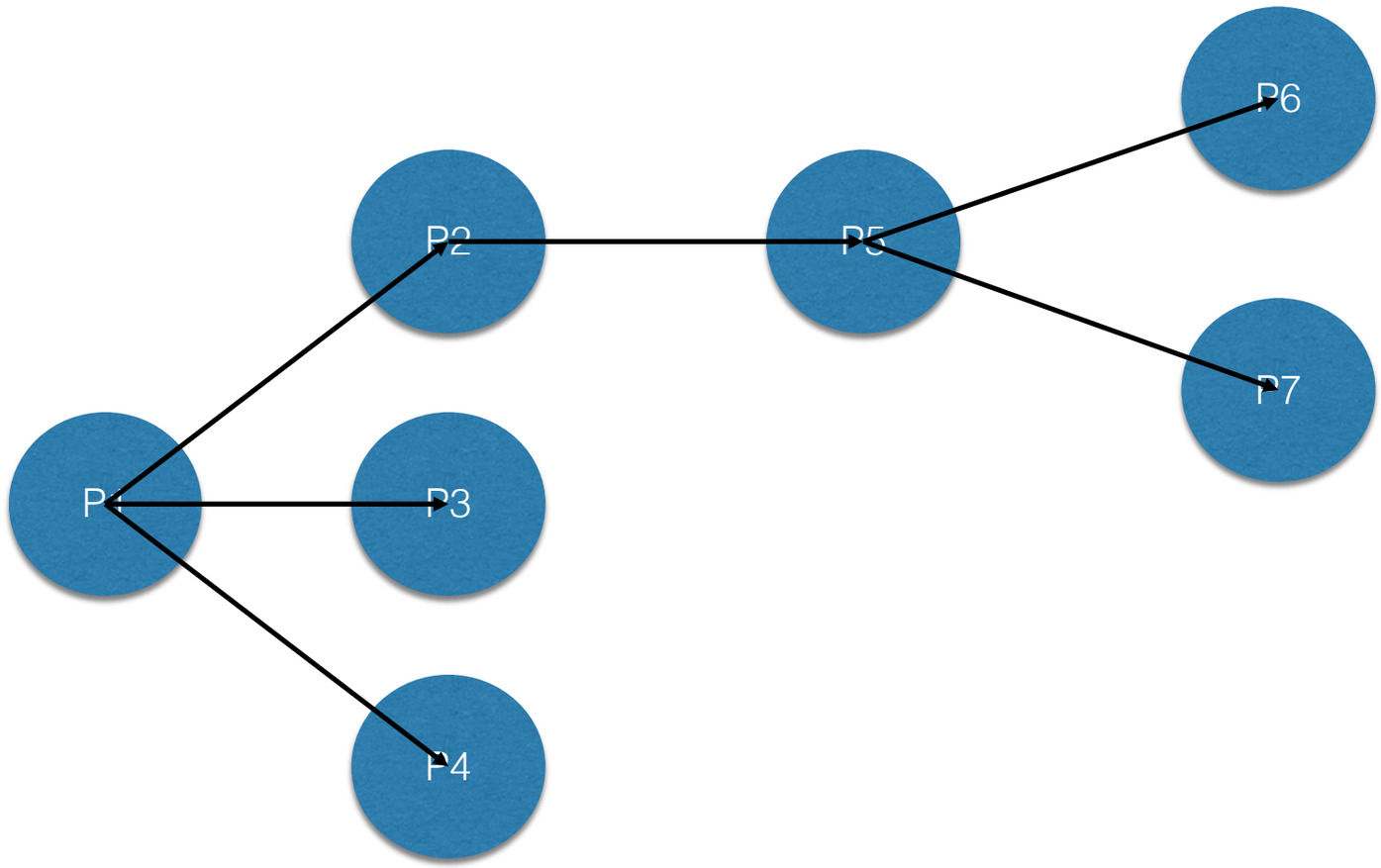
- How many pieces of data does the PE combine?
  - Transformation: one input (e.g. normalisation)
  - Product or Join: two inputs (e.g. cross-correlation)
  - And many others!

- What is the rate of processing?
  - One output per input (transformation, filter)
  - Aggregation of data (e.g. stacking)

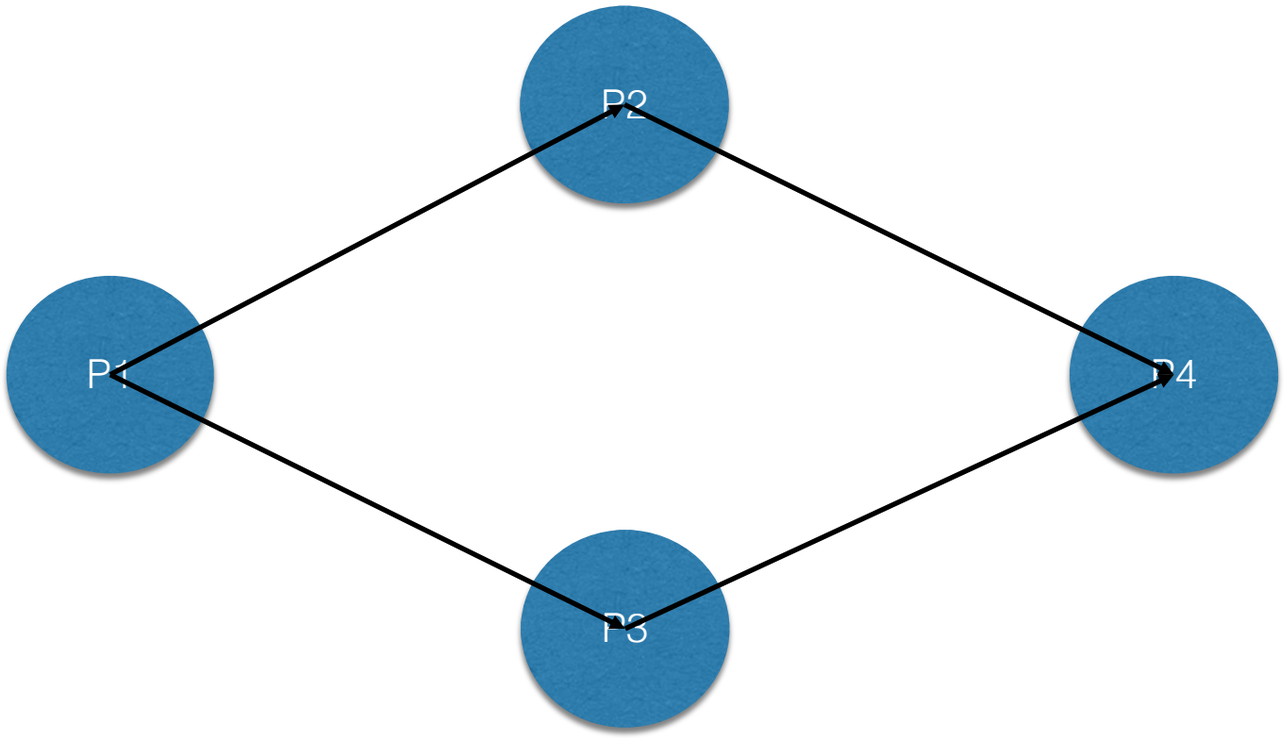
Pipeline



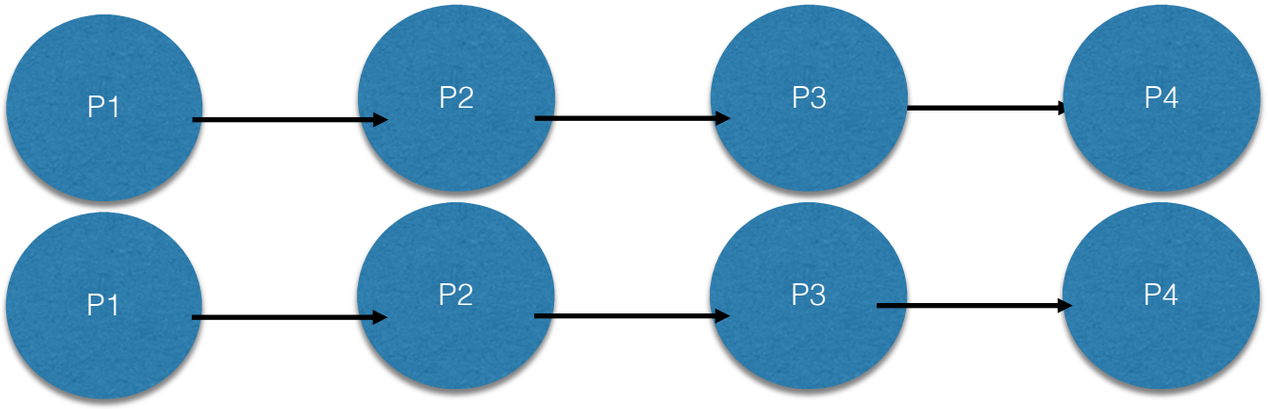
Tree



Split and Merge

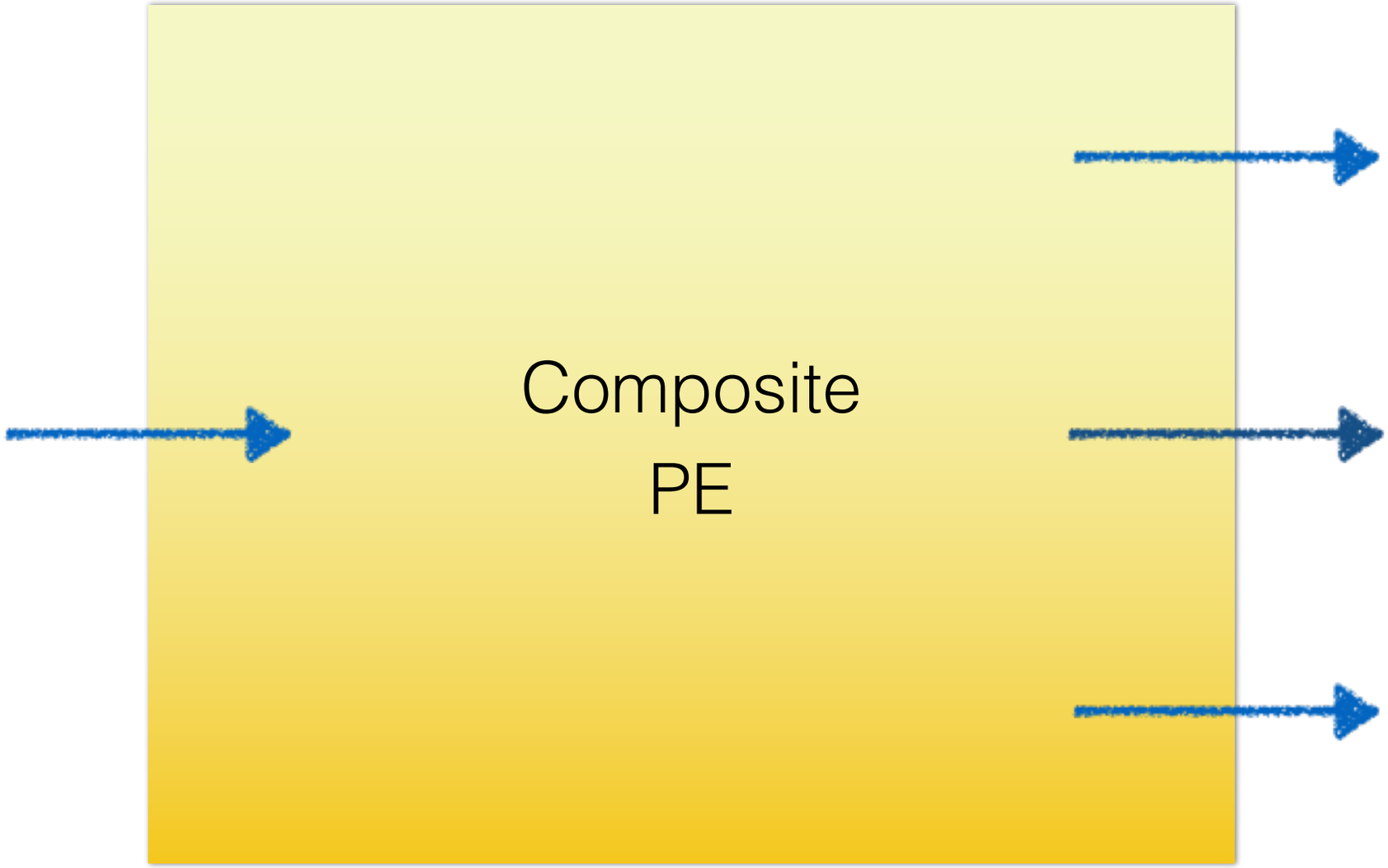


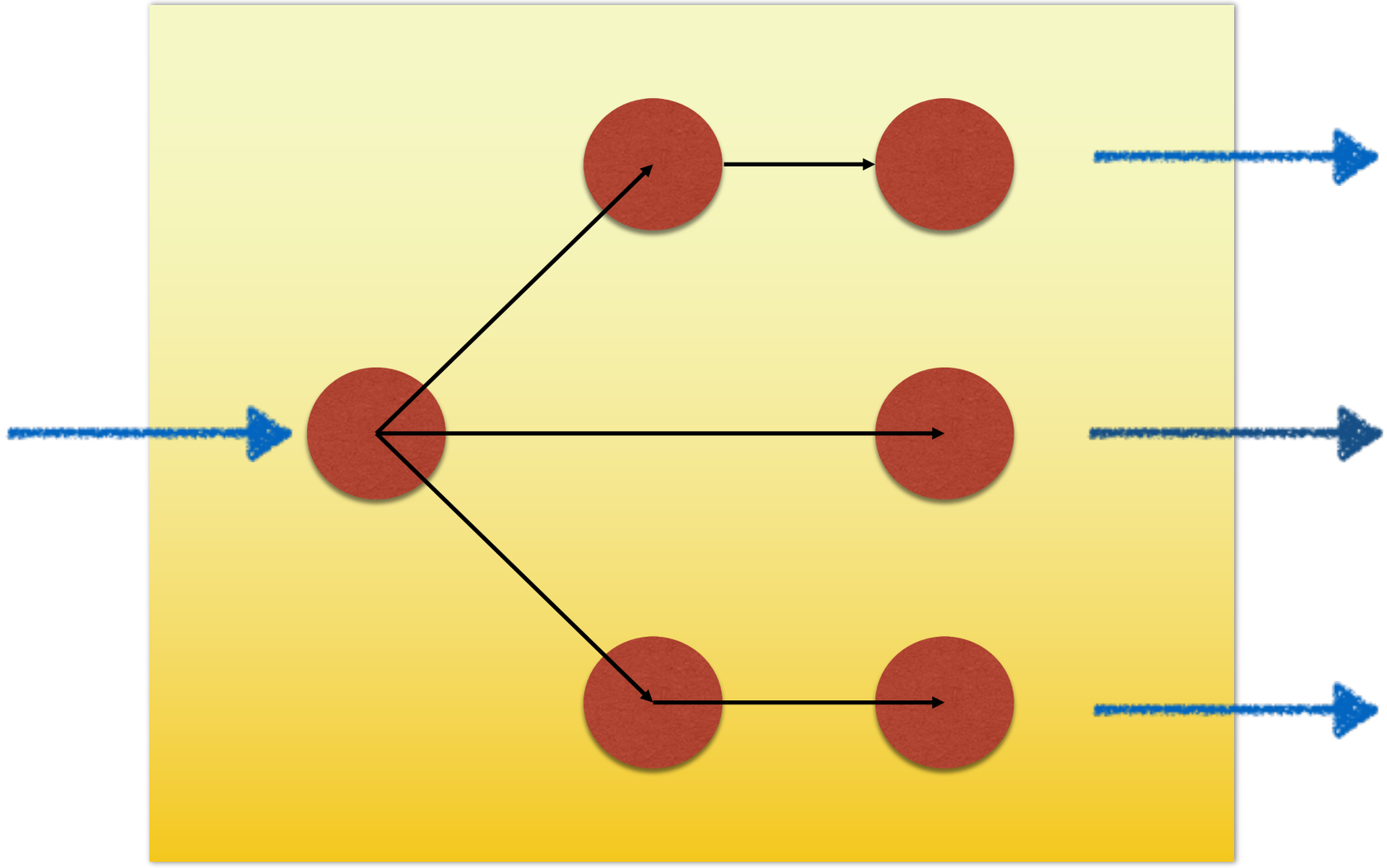
Unconnected

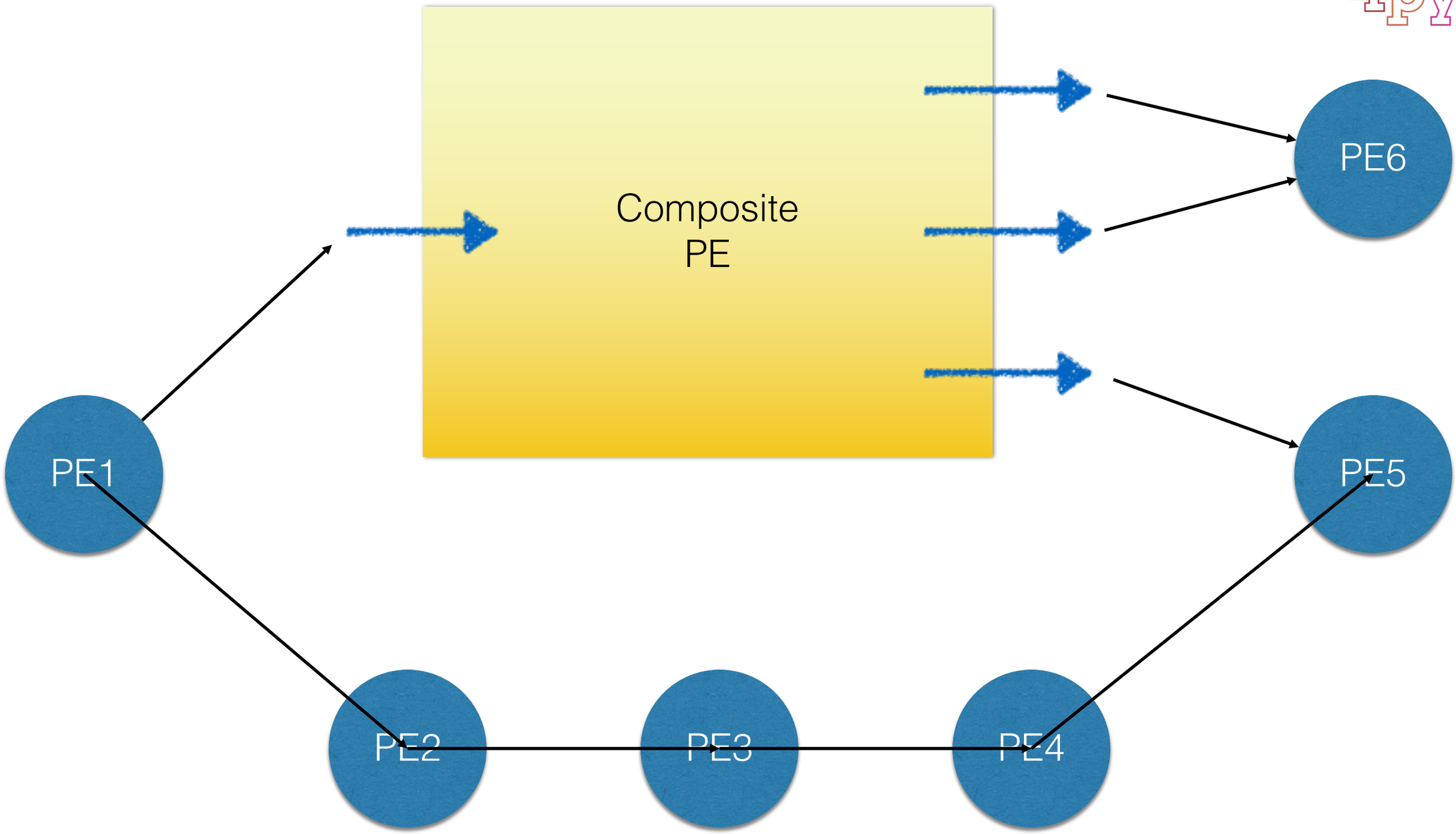


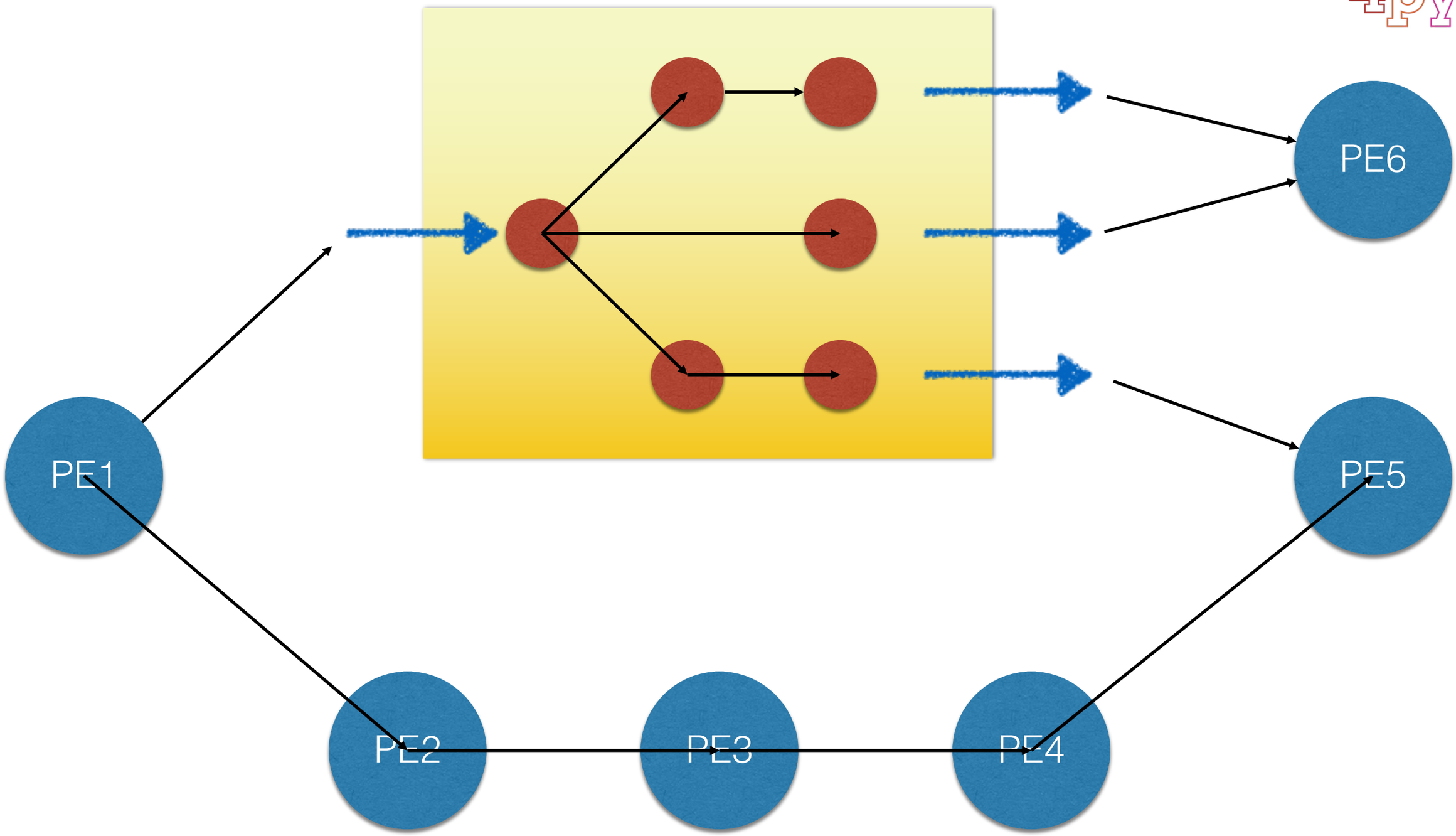
Composite PEs

- 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:

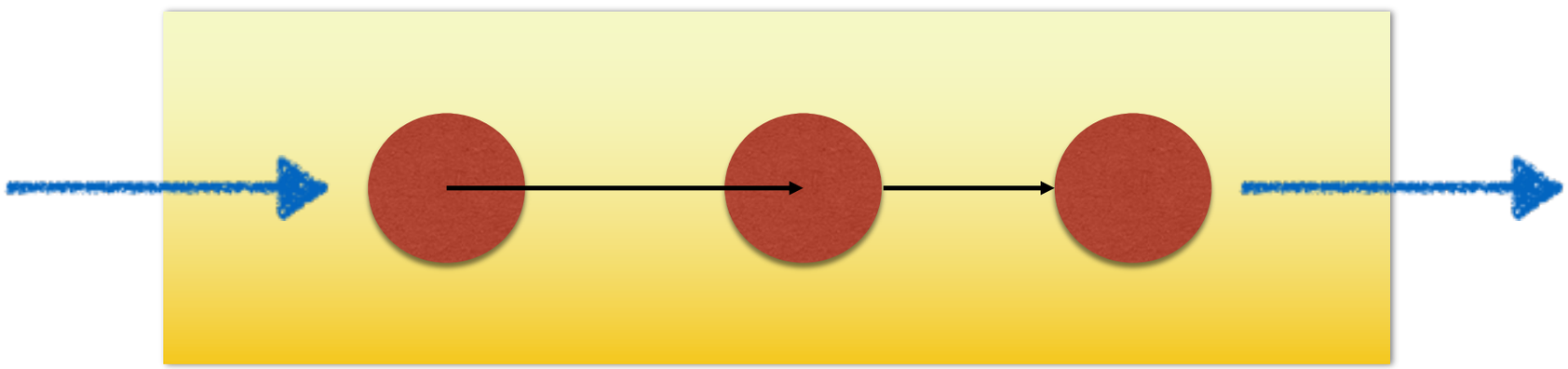
1. Only implement the processing functions (process one, return one)

2. Create a list that contains the functions in the order that they are to be applied

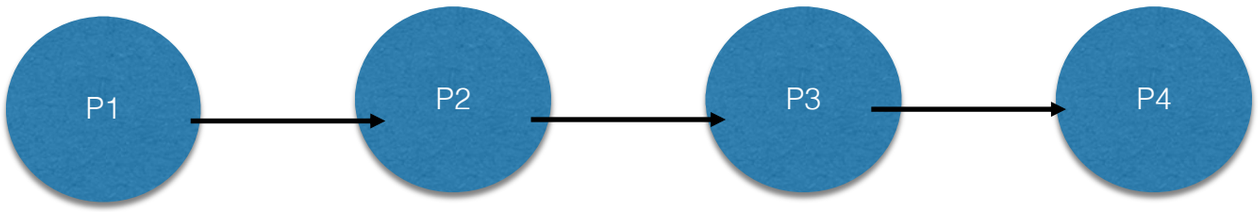
3. Pass this list to the utility and receive a composite PE in return

```
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
```

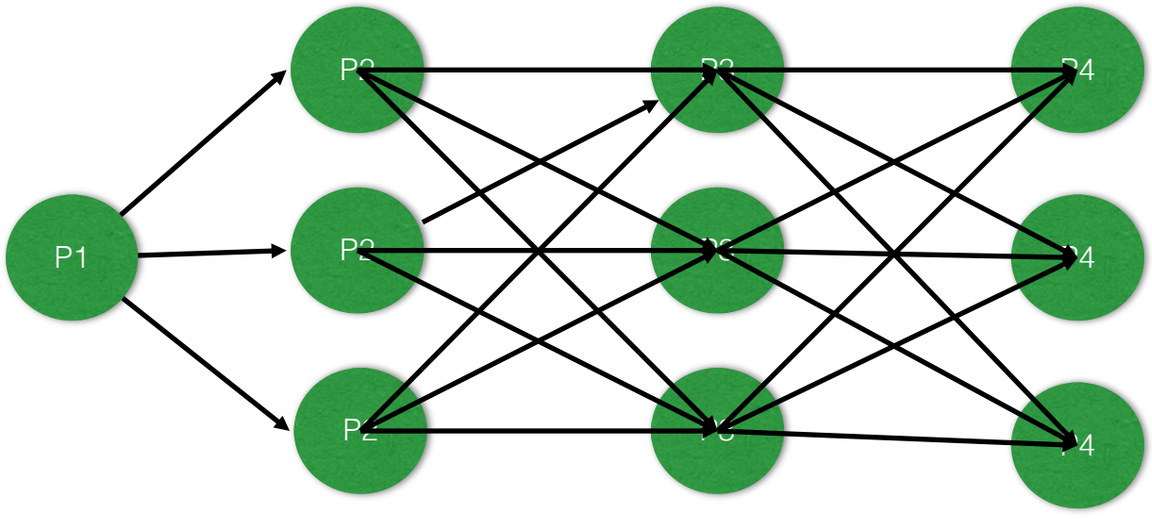
```
compositePE = create_iterative_chain([(decimate, {'sps':4}), detrend, demean])
```



Executing graphs

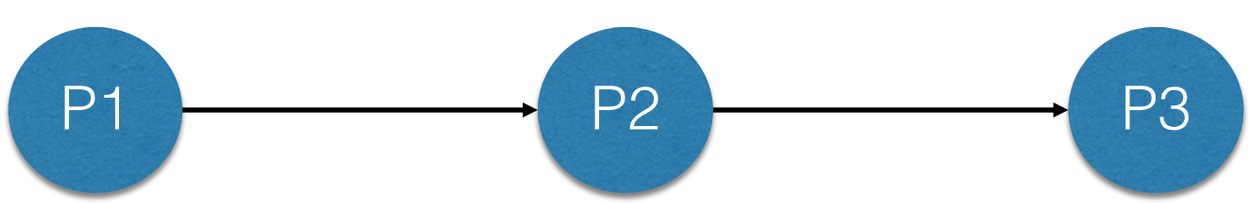


Execution

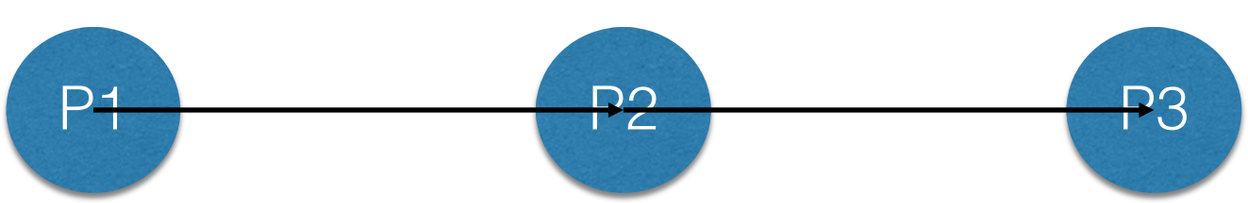


PE Instance

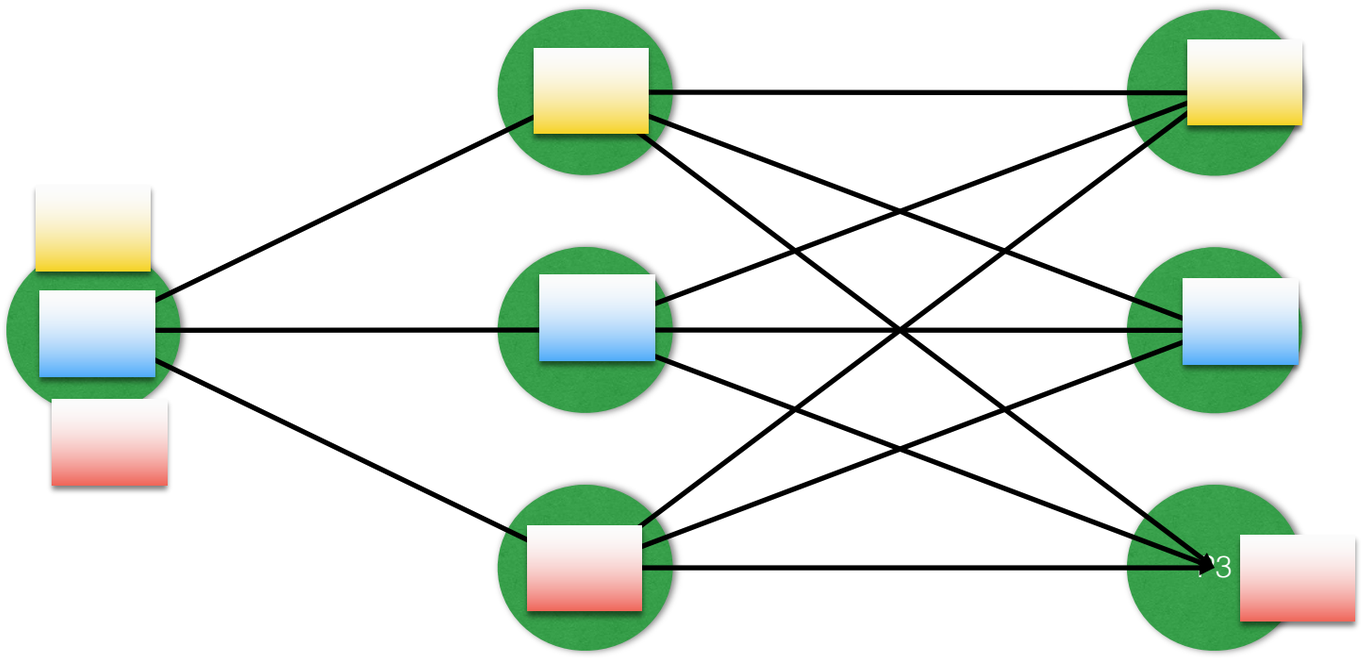
Executing graphs



Executing graphs

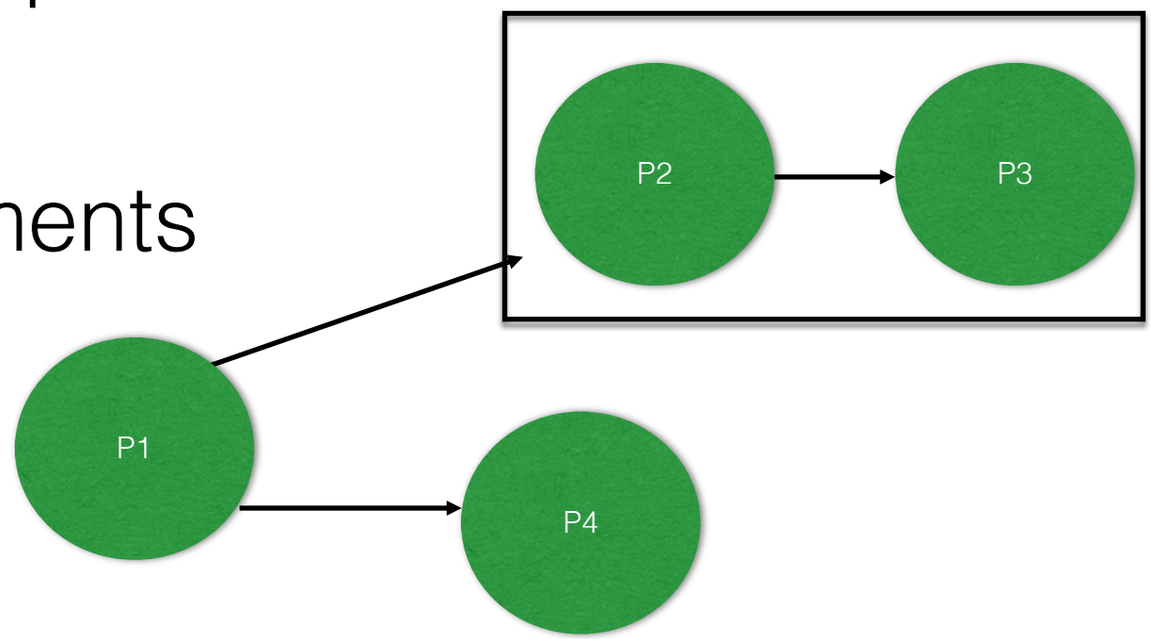


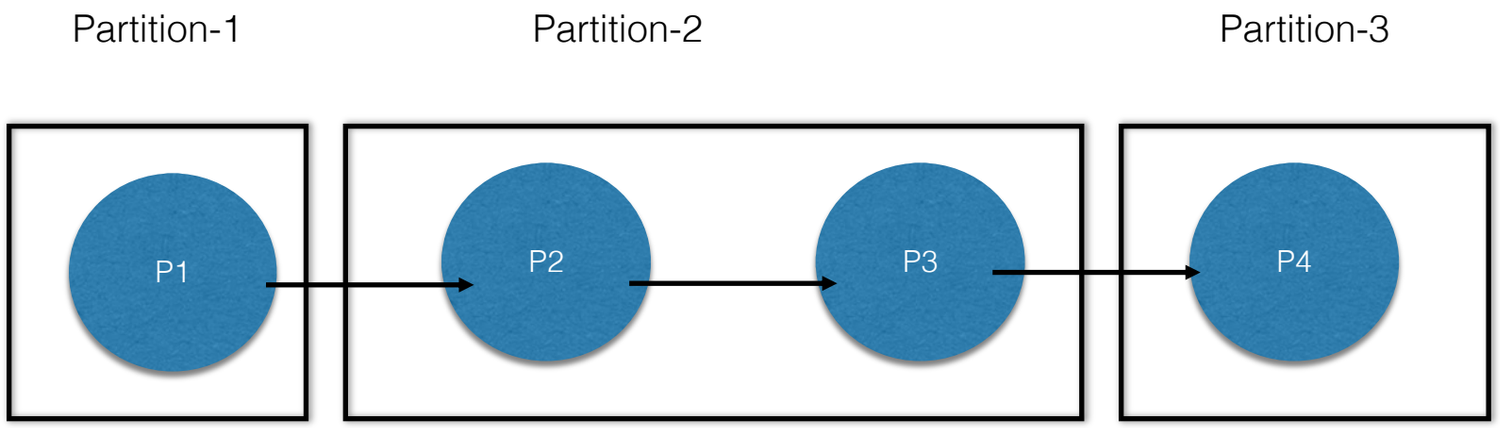
Executing graphs



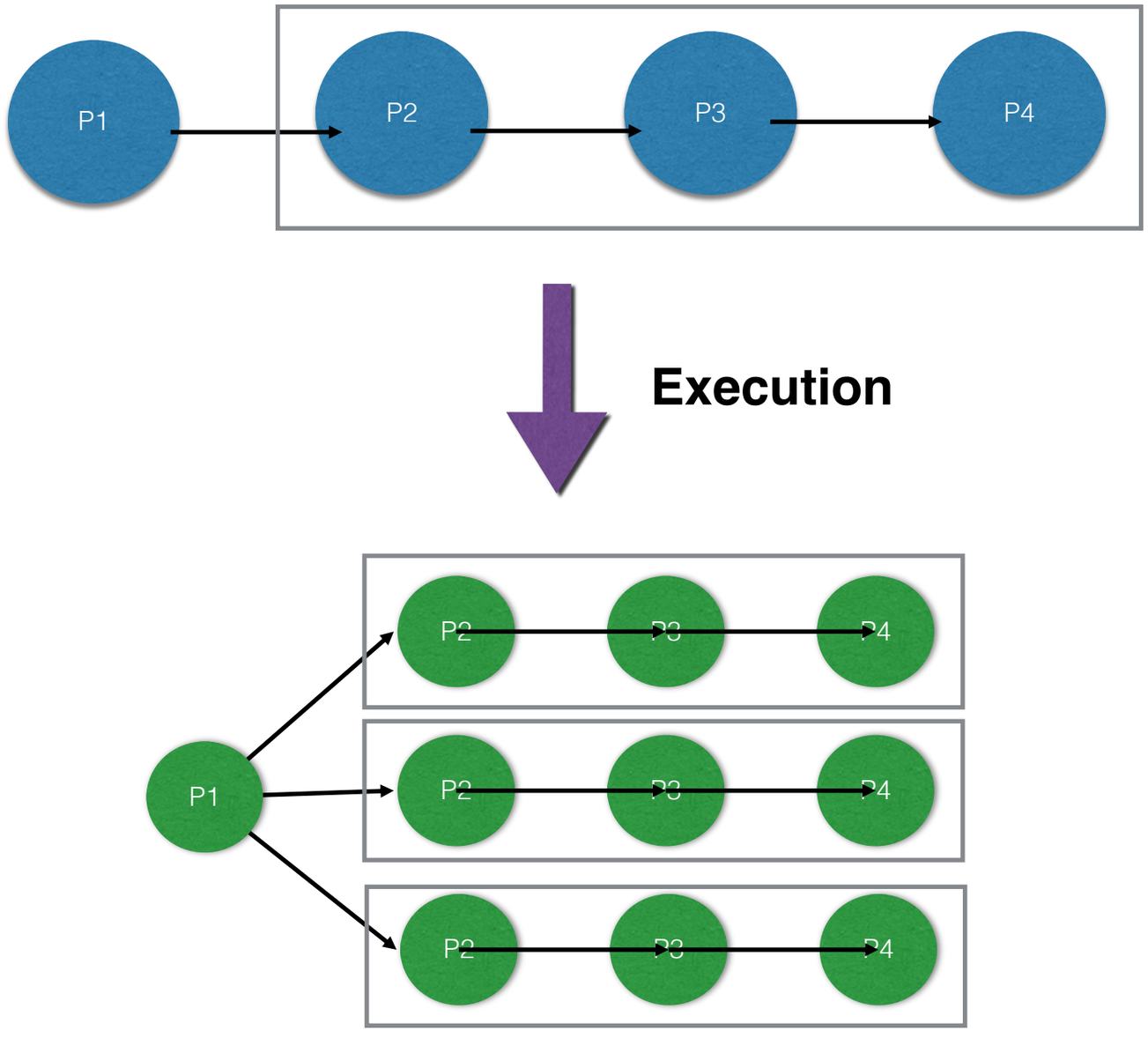
Partitions

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





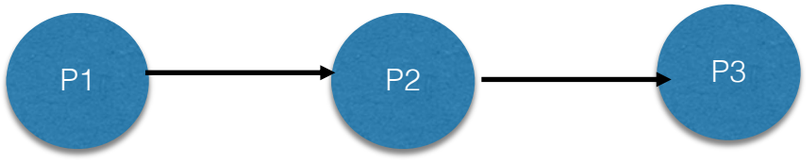
Partitioned Pipeline

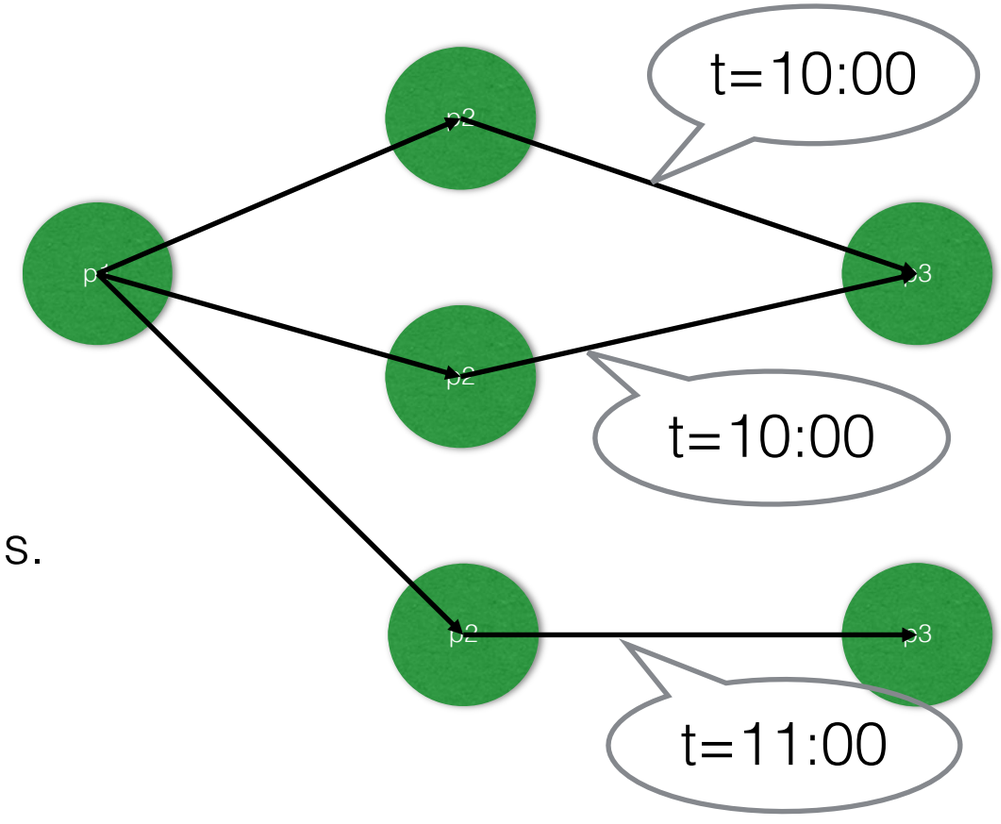


Groupings

“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





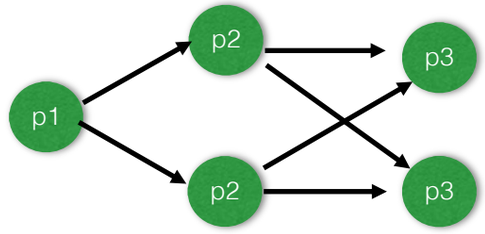
Note: You will need a GenericPE for using the groupings.

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

Other groupings

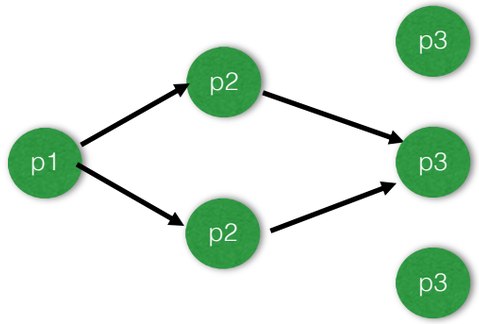
One-To-All

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

Note: Now we have also REDIS and Dynamic Mappings

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

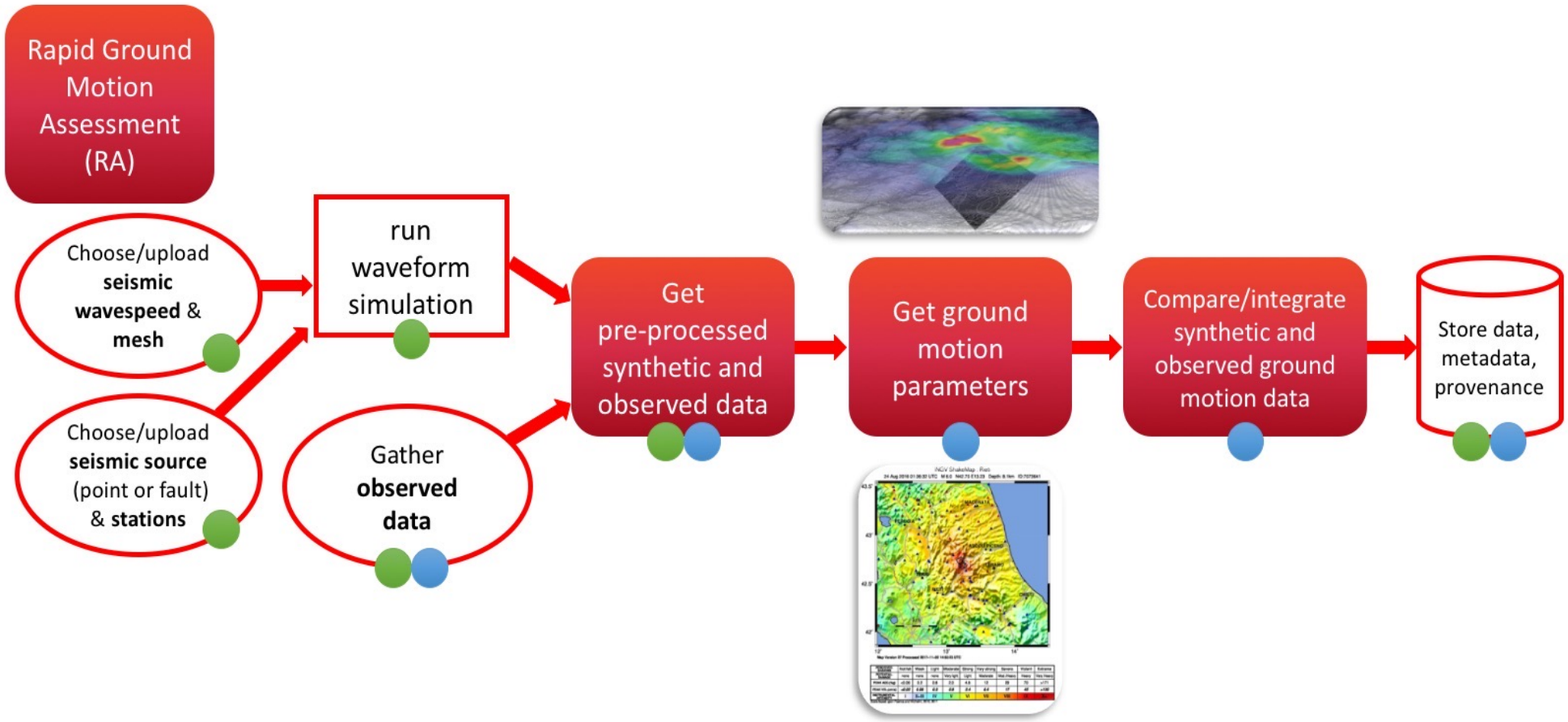
Useful dispel4py information

- 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

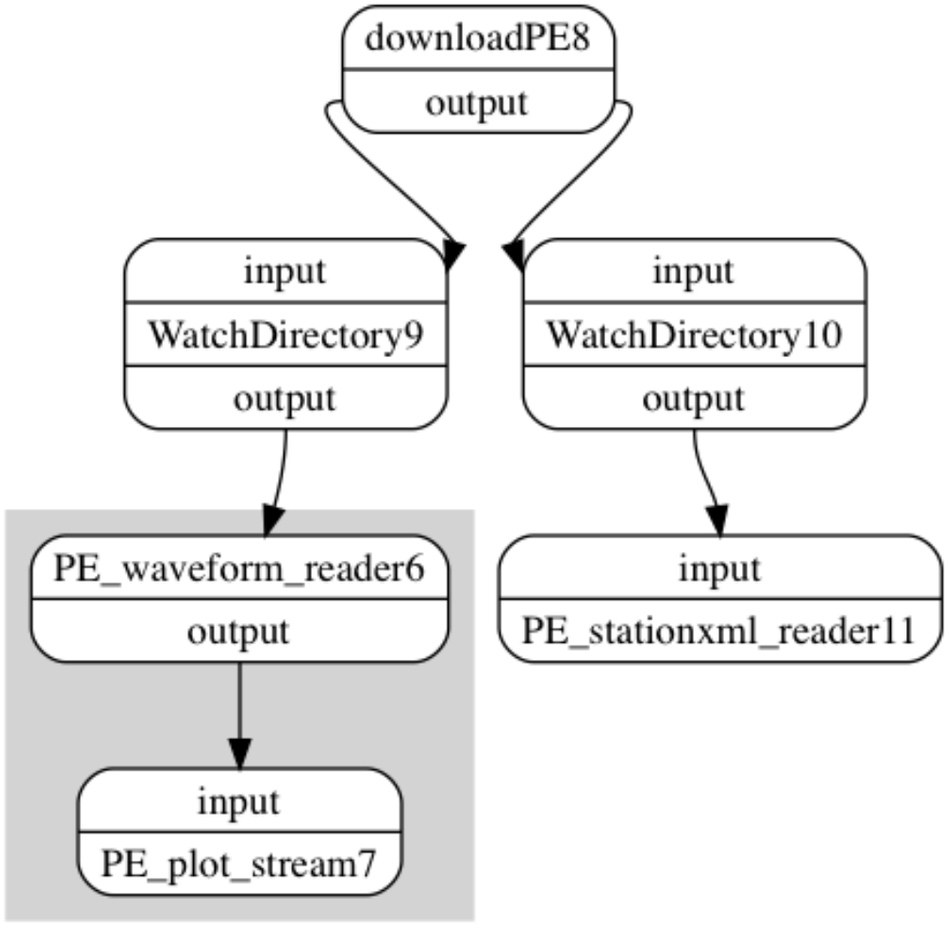
Useful dispel4py information

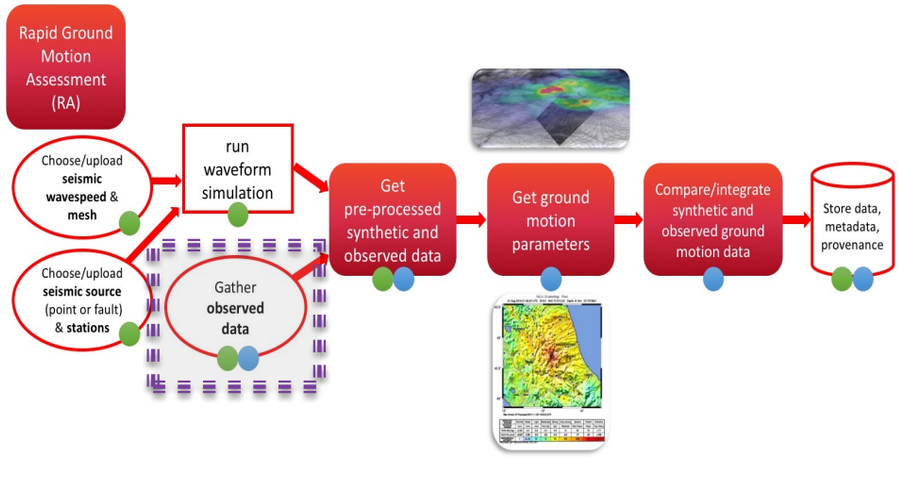
- inputs to the workflow:
  - dispel4py simple workflow.py -d ‘{“PE NAME CLASS" : [{"input" : “Hello World”}]}’
  - dispel4py simple workflow.py -d ‘{"PE NAME CLASS" : [{"input" : "coordinates.txt"}]}’
  - dispel4py simple workflow.py -f coordinates.txt

Rapid Ground Motion Assessment



RA – Download observed data

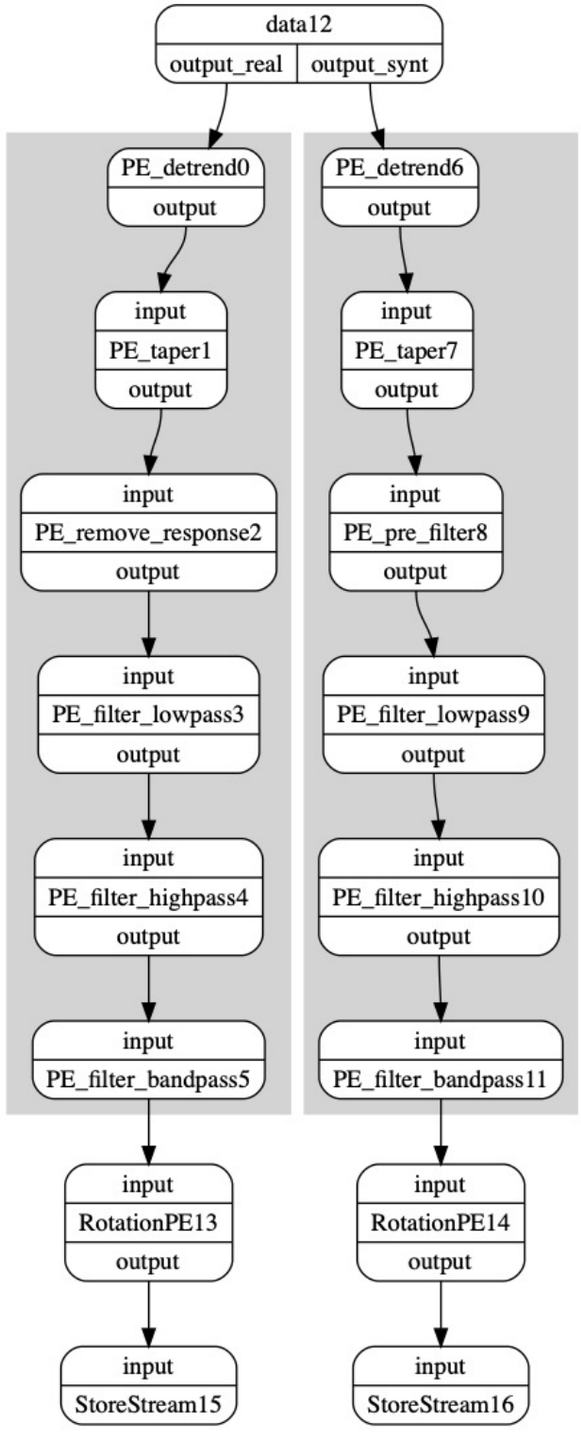


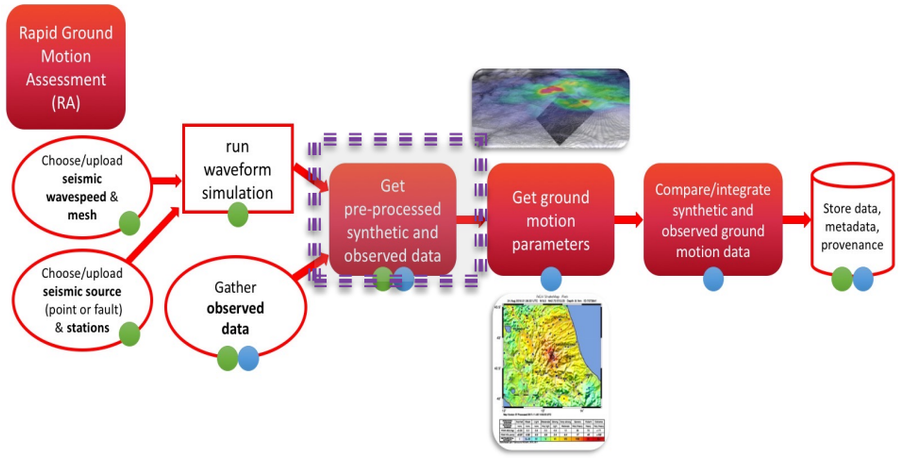


The workflow downloads real waveforms corresponding to the same earthquake.

```
downloadPE:
  0:
    input:
      minimum_interstation_distance_in_m:   100
      channel_priorities:
        0:                                  "BH[E,N,Z]"
        1:                                  "EH[E,N,Z]"
      location_priorities:
        0:                                  ""
        1:                                  "00"
        2:                                  "10"
      mseed_path:                           "./data"
      stationxml_path:                      "./stations"
      RECORD_LENGTH_IN_MINUTES:             1
      ORIGIN_TIME:                          "2013-02-16T21:16:09.29"
      minlatitude:                          41.10007459633125
      maxlatitude:                          42.89777970948071
      minlongitude:                         12.041644551237324
      maxlongitude:                         14.439665626790928
```

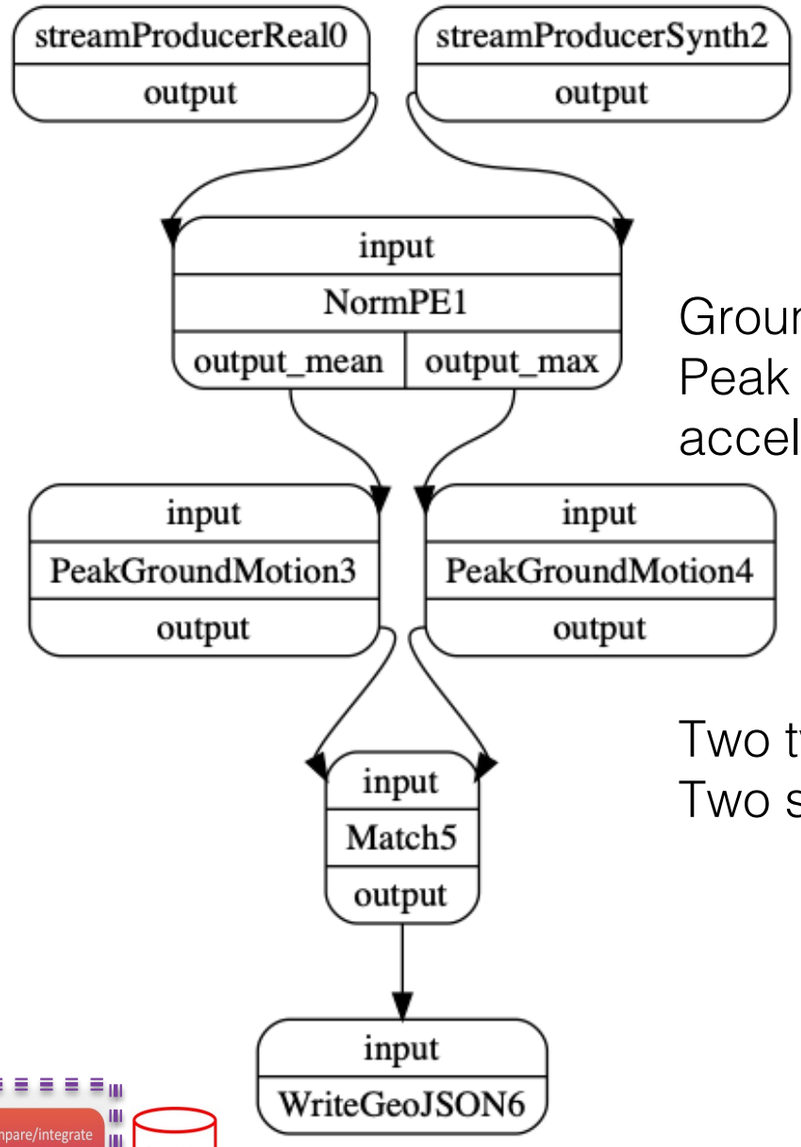
RA – Preprocessing observed and synthetic





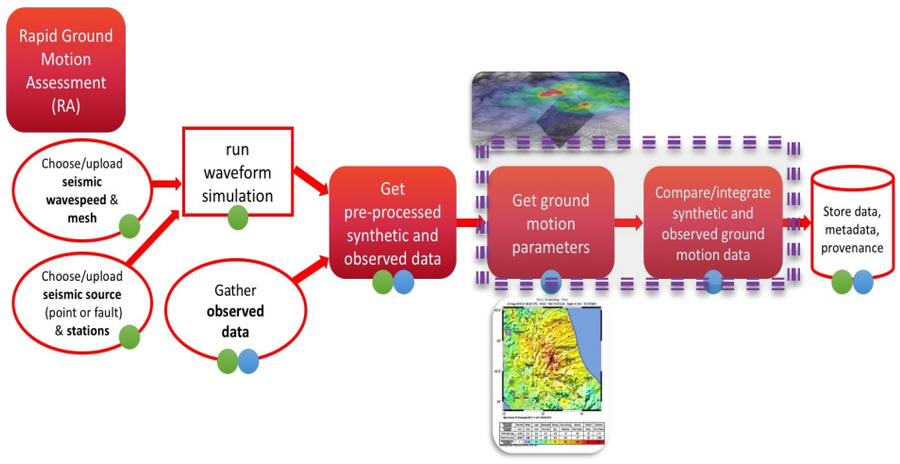
Similar preprocessing steps in synthetic and observed data

RA – Ground motion parameters



Ground motion parameters:
Peak ground values of displacement, velocity and acceleration.

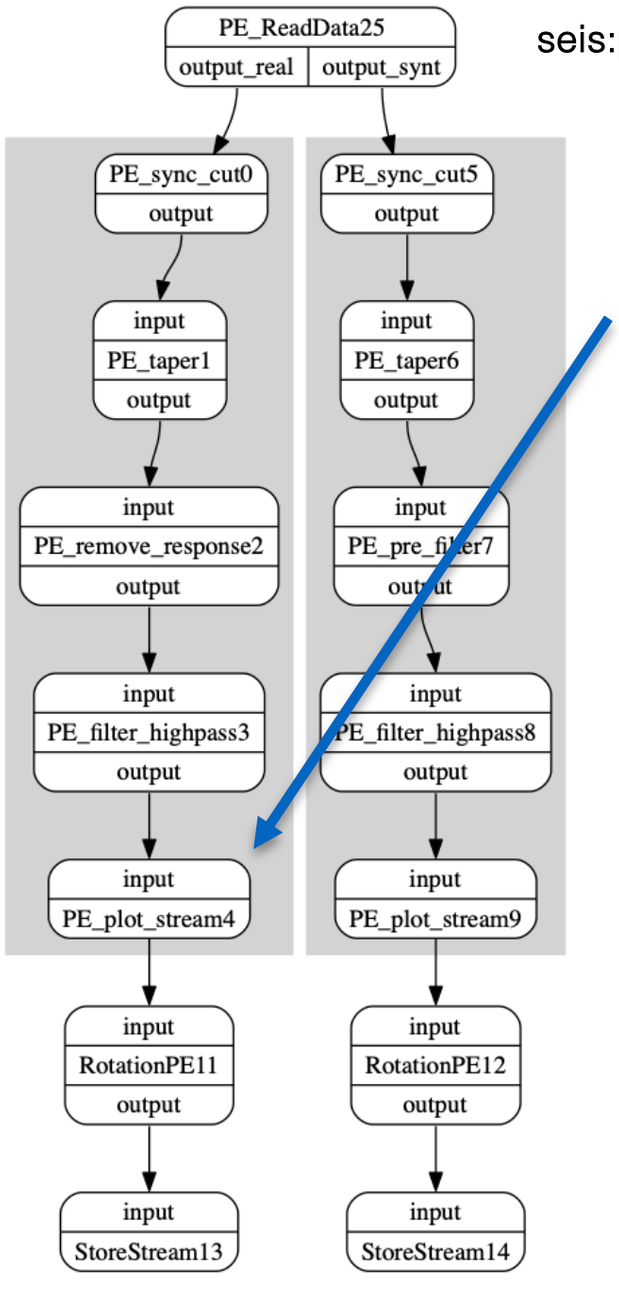
Two types of normalisation – Mean & Max
Two set of PGM outputs



Provenance

Inline metadata injection

Waveform Preprocessing



seis:preprocessing

pipeline JSON Description (eg. from file)

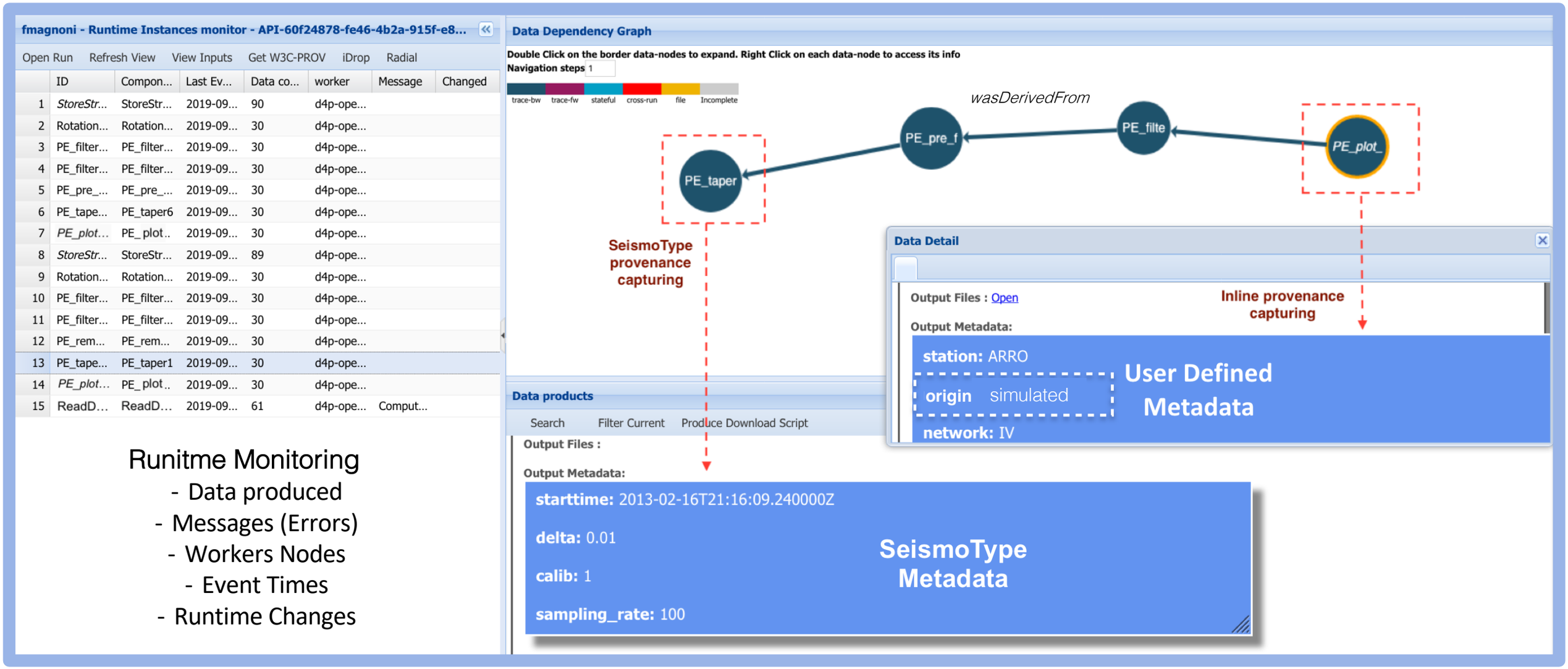
Manual Extensions

Functions encoded in Python
User Defined Metadata injection into Lineage traces

```
def plot_stream(stream, output_dir, tag):
    stats = stream[0].stats
    filename = "%s.%s.%s.%s.png" %
        (stats['network'], stats['station'],
         stats['channel'], tag)

    path = os.environ['STAGED_DATA'] + '/' + output_dir
    dest = os.path.join(path, filename)
    stream.plot(outfile=dest)

    prov = {'location': "file://" + socket.gethostname()+"/"+dest,
            'format': 'image/png',
            'metadata': {'origin': tag}}

    return {'_d4p_prov': prov, '_d4p_data': stream}
```

User Defined Metadata

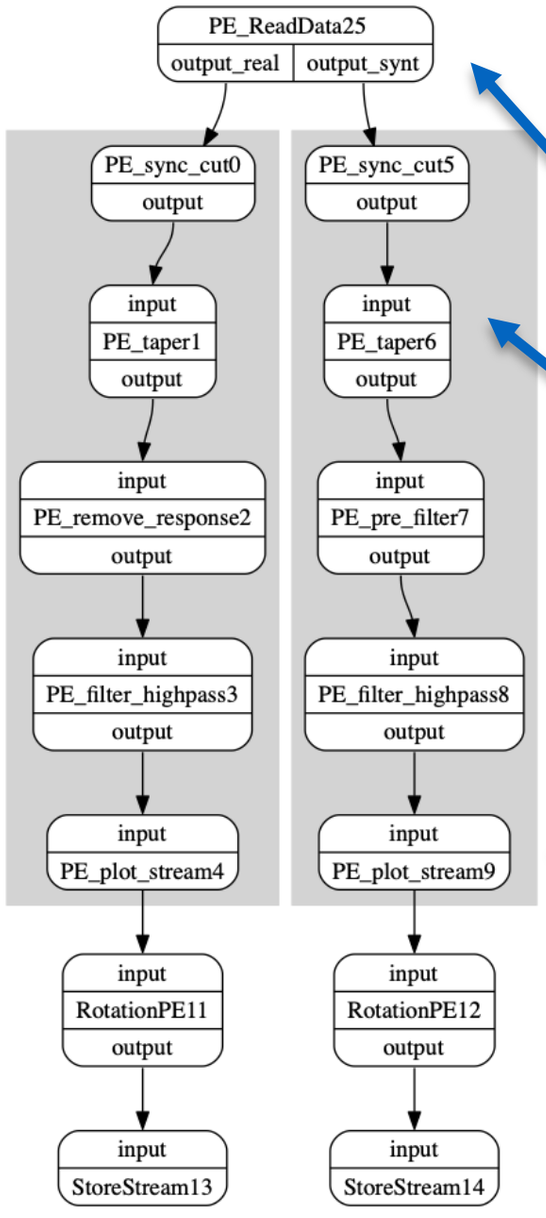
Monitor, search and analyse results through lineage



S-ProvFlow:https://gitlab.com/project-dare/s-Pro

Workflow Contextualisation

Seismic Data Preprocessing



Configuration Profile in JSON with Provenance Types

```
{
'provone:User': "aspinuso",
's-prov:description' : "provdemo",
's-prov:workflowName': "waveform preprocessing pipeline",
's-prov:workflowType': "seis:preprocessing",
's-prov:WFExecutionInputs' : [{...}],
's-prov:save-mode'    : 'service',
's-prov:WFExecutionInputs':  [{...}],
# defines the Provenance Types and Provenance Clusters for the Workflow's Components
's-prov:componentsType' :
        {'s-prov:componentsType' :
              {'PE_ReadData':       { 's-prov:type':['SeismoType'],
                                      's-prov:prov-cluster': "seis:DataHandler",

'PE_taper':           { 's-prov:type':['SeismoType'],
                        's-prov:prov-cluster': "seis:Processor",

'PE_remove_response':{  's-prov:type':['SeismoType'],
                        's-prov:prov-cluster': 'seis:Processor'},

'PE_plot_stream':     {'s-prov:type'

'
```

Semantic Tagging

starttime: 2013-02-16T21:16:09.240000Z

delta: 0.01

calib: 1

sampling_rate: 100

ProvenanceType for Metadata Contextualisation

Spinuso, A, Atkinson, M & Magnoni, *Active provenance for Data-Intensive workflows: engaging users and developers.*

IEEE eScience 2019 proceedings. Bridging from Concepts to Data and Computation for eScience (BC2DC'19) Workshop