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

The VERCE Science Gateway: Dispel4Py Down to the Basics

(dispel4py training)

9-11 March 2015

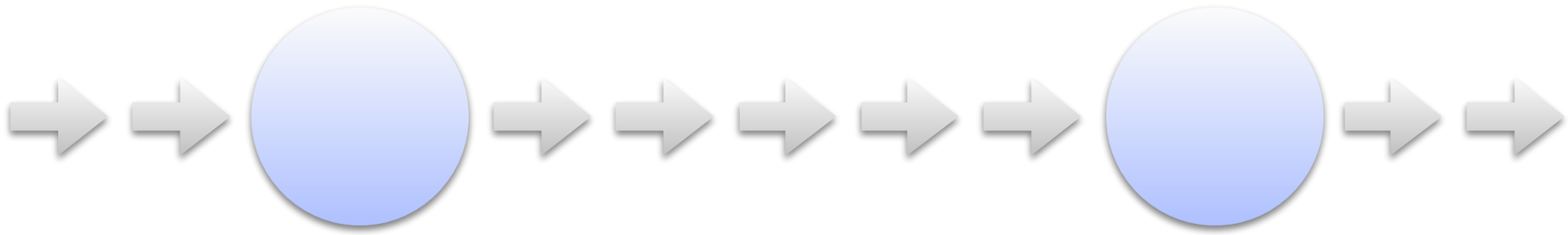
Outline

- What is dispel4py
- What is a stream
- What is a processing element (PE)
- What is a instance
- What is graph
- What I need for constructing a dispel4py workflow

What is dispel4py

- A library for developing **distributed data-intensive applications**
- Describes the **data-flow** and the **processing elements**
- The language is **Python**
- dispel4py **maps** to a number of enactment systems
- Applications **scale** automatically to exploit parallel processing, clusters, grids and clouds
- dispel4py is **dataflow-oriented** rather than control-oriented:
  - No specification of how data should be produced or consumed

What is a data stream



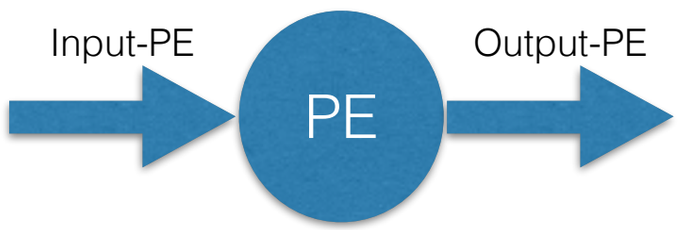
- A **stream** is a sequence of data elements made available over time

- It also can be defined as a flow of input or output data.

- It takes data from a source and delivers data to its destination.

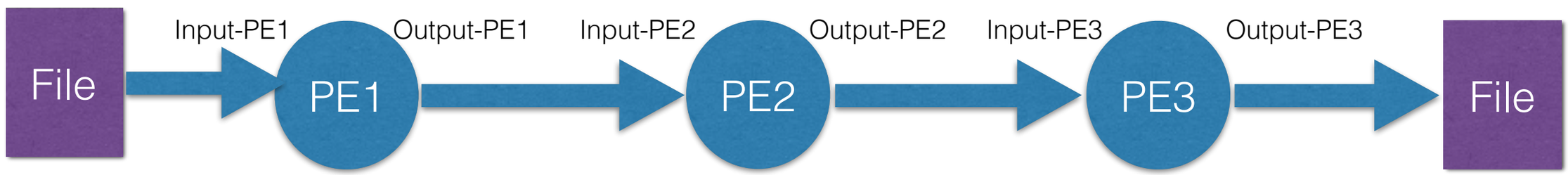
What is a processing element (PE)

- A dispel4py PE is a computational activity which encapsulates an algorithm, services and other data transformation processes.
- PEs represent the basic computational building blocks of any dispel4py workflow.
- Each PE has:
  - inputs & outputs
  - computational activity.
- PEs are connected by streams, and not by writing or reading files



What is a graph

- It defines the way throughout which PEs are connected and data can be streamed.
- A graph describes the topology of the data flow
- No limitations on the type of graphs



What I need for constructing a dispel4py workflow

- Users **only** have to **implement their PEs** (in python) and connect them as they wish:
- We need to learn how to implement PEs.
- We need to learn how to connect them.

How to implement a PE

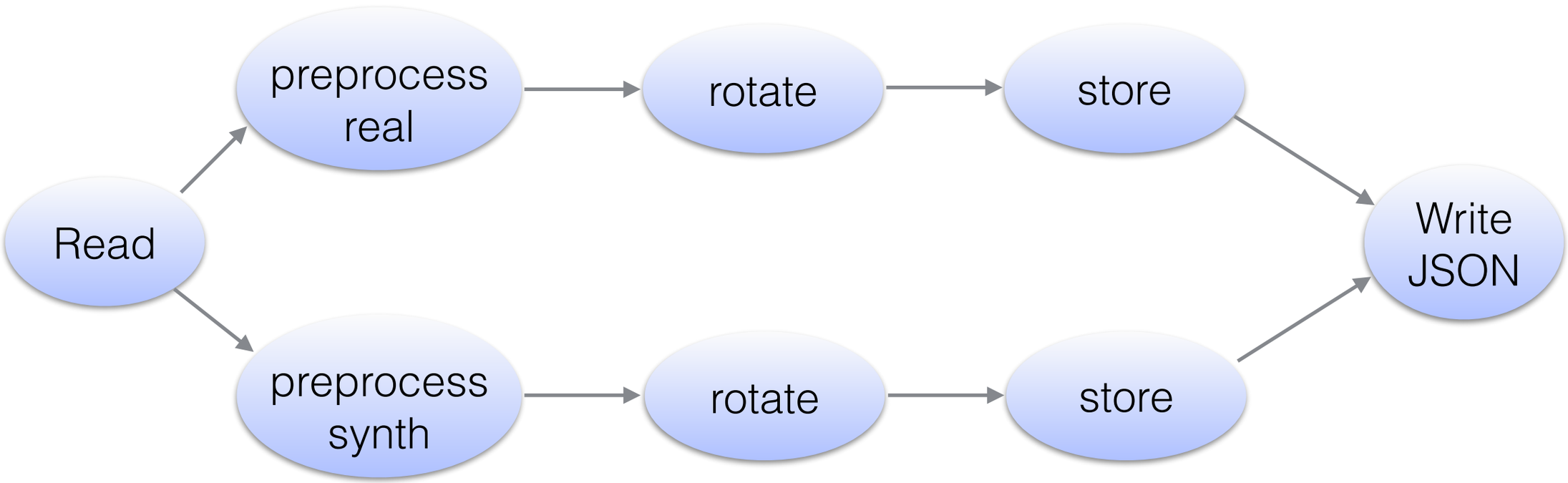
- Each PE must indicate:
  - Input & Output streams
  - Computational activity for processing data-blocks - implement the “_process” method.

Example:
Decimate Trace

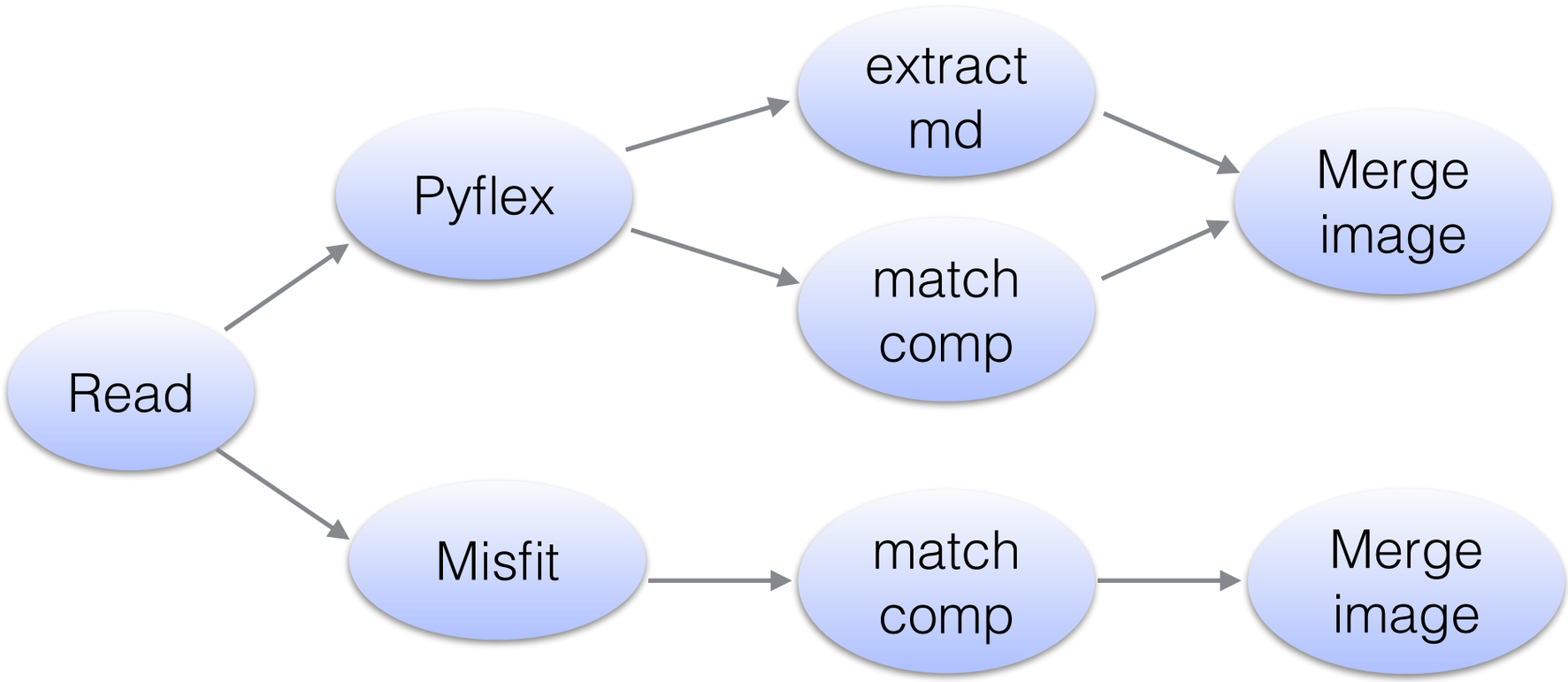
```
def decimate(data, sps):
    st = data[0]
    st.decimate(int(st[0].stats.sampling_rate/sps))
    return st
```

- A simple example of a function you would implement as a PE
- Each execution of the function processes a trace from the input and produces a trace as output
- This function can be applied to a data stream, processing one data block at a time (*iterative*)

Misfit Preprocessing



Misfit


extract
md
Pyflex
Merge
image
match
comp
Read
Misfit
match
comp
Merge
image

Misfit calculation

http://tinyurl.com/dispel4py-training-march2015

Types of PEs

Type	Inputs	Outputs	When to use it
GenericPE	*n* inputs	*n* outputs	many inputs and/or many outputs
IterativePE	1 input named 'input'	1 output named 'output'	processing one data block and producing one in each iteration
Consumer PE	1 input named 'input'	no output	only implement _process method
Simple FunctionPE	1 input named 'input'	1 output named 'output'	only implement _process method
create_iterative_chain	1 input named 'input'	1 output named 'output'	pipeline of functions processing sequentially; creates a composite PE

SimpleFunctionPE

```
def stream_producer(data):
    filename = data
    st = read(filename)
    return st
#For using this function as a PE we need to use 'SimpleFunctionPE' before defining the graph:

streamProducer = SimpleFunctionPE(stream_producer)
```

This function reads a file that contains seismological traces and returns it as an obspy stream.

What we have learnt:

- Only implement the processing function
- The easiest but the most restrictive way
- The **function cannot store state between calls**; for example you can't implement SUM or AVG with it
- 1 input called 'input', 1 output called 'output'.

IterativePE example

```
class StreamProducer(IterativePE):
    def __init__(self):
        IterativePE.__init__(self)
    def _process(self, data):
       # this PE consumes one input
        self.log(data)
        filename = data
        st = read(filename)
        return st
```

This PE also reads a file (‘input’) that contains seismological traces and returns one output (‘output’): obspy stream.

What we have learnt:

- We don’t need to specify the input and output
- The parameter to the _process method is a tuple
- _process returns the value that is written to the output stream

GenericPE example

```
# this PE consumes data in the format [url] - a list with one element
from obspy.core import read
from dispel4py.core import GenericPE
class StreamAndStatsProducer(GenericPE):
    def __init__(self):
        GenericPE.__init__(self)
        self._add_input('input')
        self._add_output('output_stream')
        self._add_output('output_stats')
    def process(self, inputs):
        data = inputs['input']
        filename = data
        st = read(filename)
        # This PE returns two outputs: the output stream and the trace stadistics (metadata).
        return {'output_st': stream, 'output_stats': st[0].stats}
```

This PE reads a file (input) that contains seismological traces and returns two outputs: obspy stream (output_stream) and metadata (output_stats).

What we have learnt:

- We can add several outputs with different names
- The process method gets values from the input streams
- The process method returns both streams

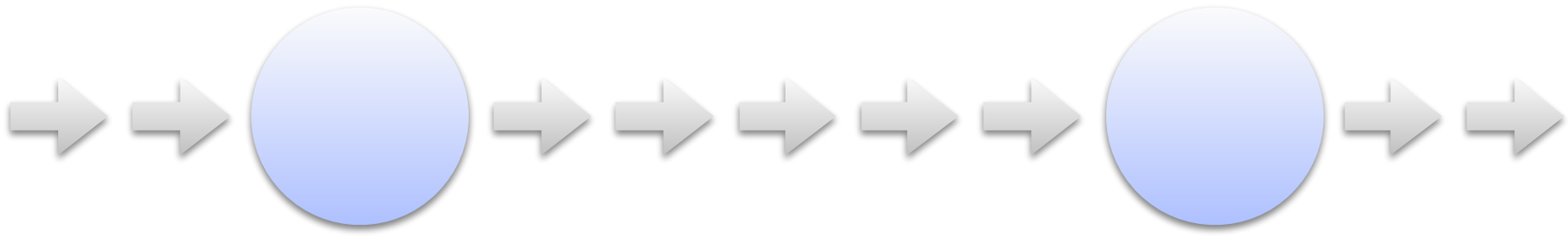
create_iterative_chain

```
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
# For using this function as a PE we need to use ‘creative_iterative_chain’ before defining the graph.
preprocess_trace = create_iterative_chain([(decimate, {'sps':4}), detrend, demean])
```

What we have learnt:

- We can create a *composite* PE which processes several function in a sequence
- Creates a pipeline of SimpleFunctionPEs
- It’s the easiest way to create a pipeline but the most restrictive
- 1 input called ‘input’, 1 output called ‘output’.

How to connect PEs
What does it mean



- PEs process a small amount of data at a time
- No data is stored, data is only read once
- PEs may store a small amount of state (e.g. stacking)

How to connect PEs: Create a PE object

- Create a PE

```
prod = StreamProducer()
```

- Create a function wrapped in a simple PE

```
prod = SimpleFunctionPE(stream_producer)
```

- Create a composite PE with a pipeline

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

How to connect PEs: Create a graph

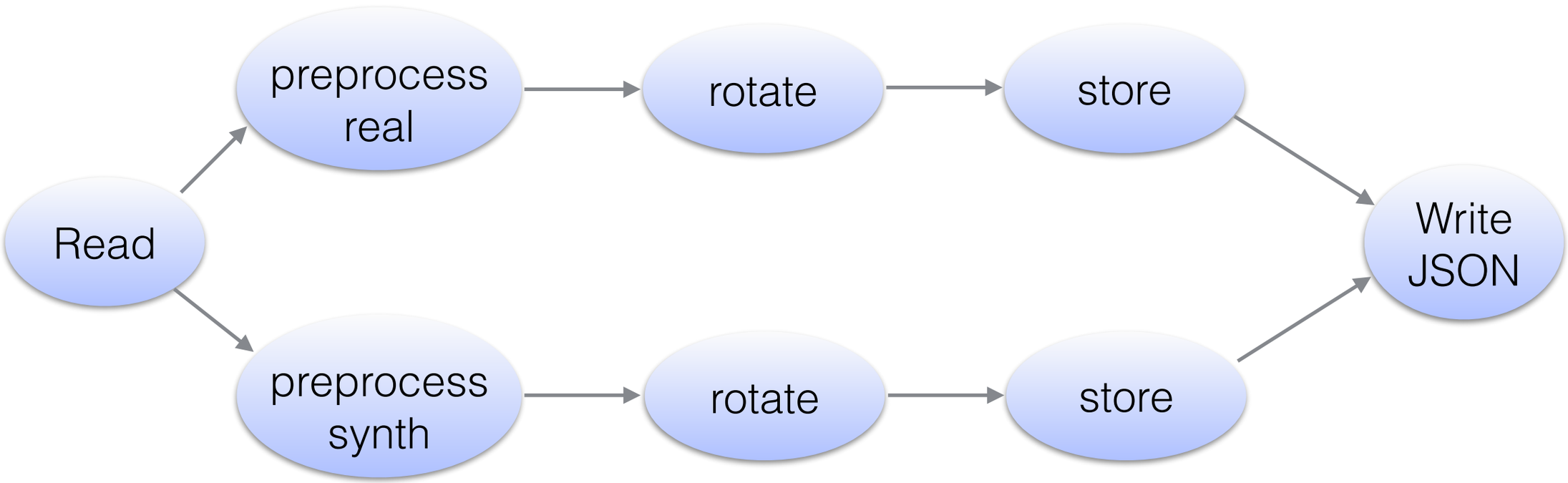
- **Create the PEs**

```
prod = SimpleFunctionPE(stream_producer)
preprocess_trace =
      create_iterative_chain([(decimate, {'sps':4}), detrend, demean])
```

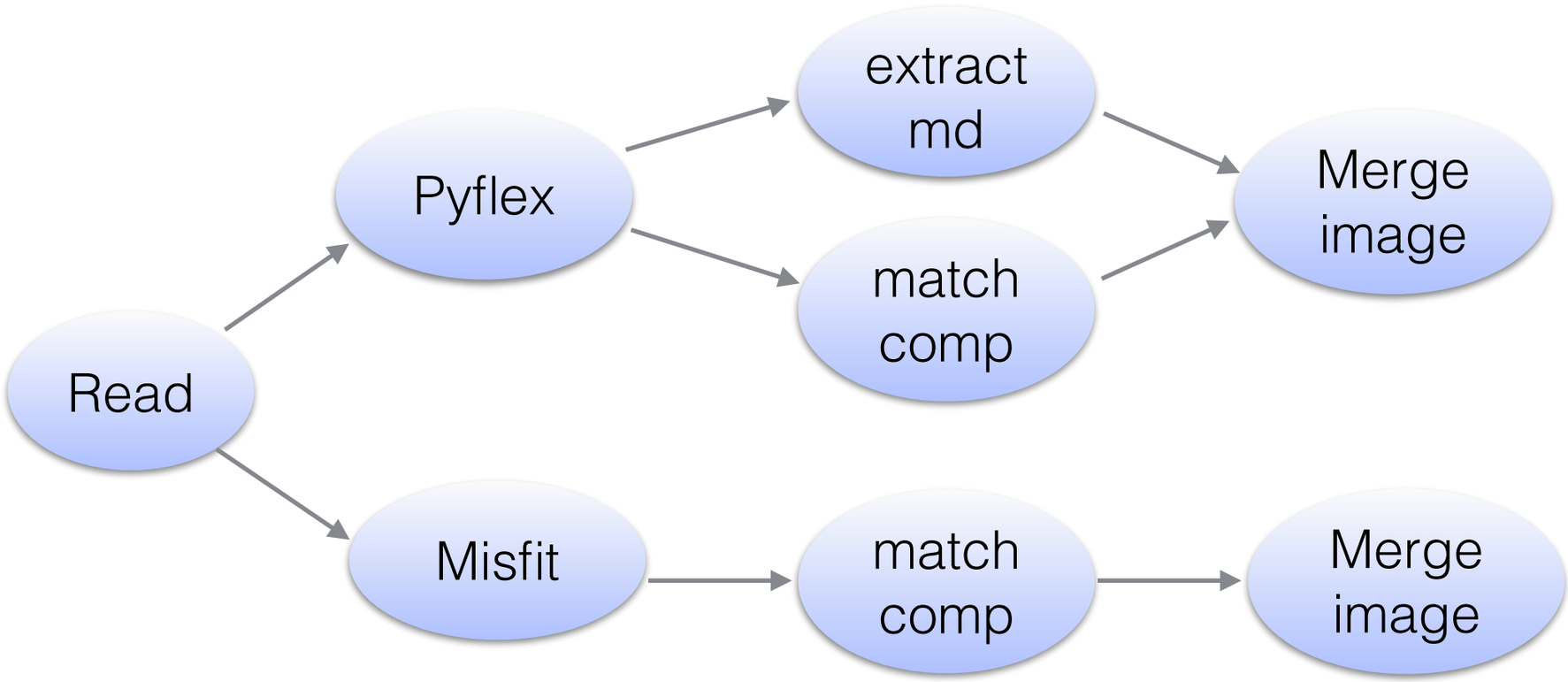
- **Create the graph and connect the PEs**

```
graph = WorkflowGraph()
graph.connect(prod, ‘output_stream', preprocess_trace, 'input')
```

Misfit Preprocessing



Misfit


extract
md
Pyflex
Merge
image
match
comp
Read
Misfit
match
comp
Merge
image

Misfit calculation

http://tinyurl.com/dispel4py-training-march2015