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 in detail

(dispel4py training)

9-11 March 2015

Outline

- Graph examples
- Composite PEs
- Chains
- Groupings
- Mappings to execution platforms

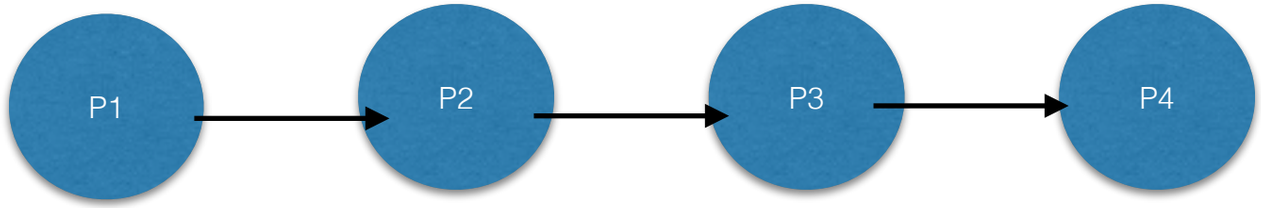
Writing PEs

- **Simple**: Usually a single unit of functionality without dependencies
  - Read file; apply normalisation; plot
- **Reusable**: To be recombined in many different applications
  - Apply a number of functions in any order
- **Shared**!

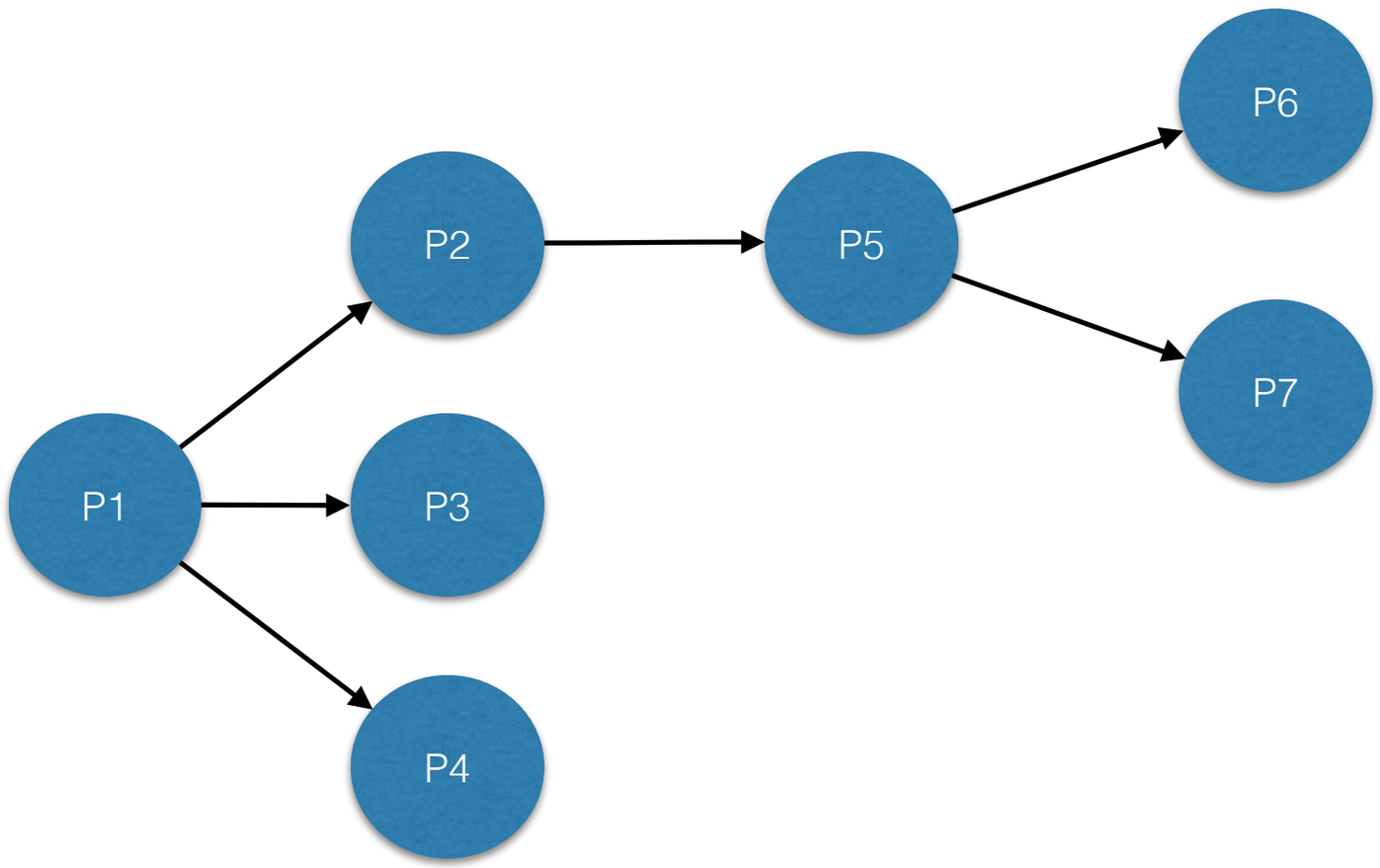
Inputs and outputs

- 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)

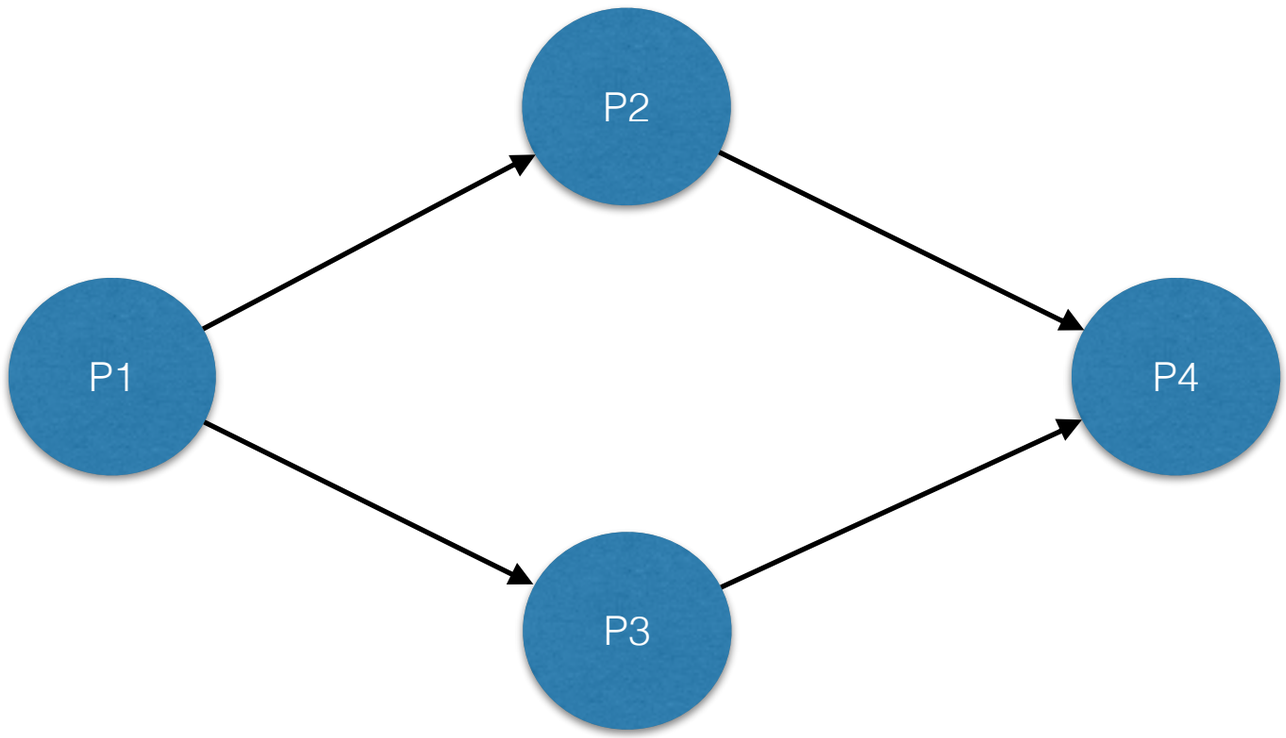
Pipeline



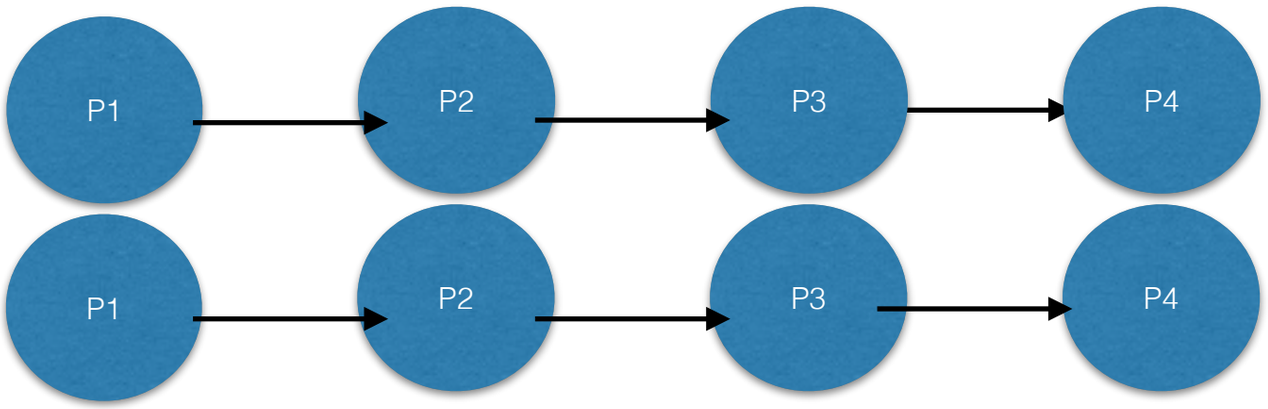
Tree



Split and Merge

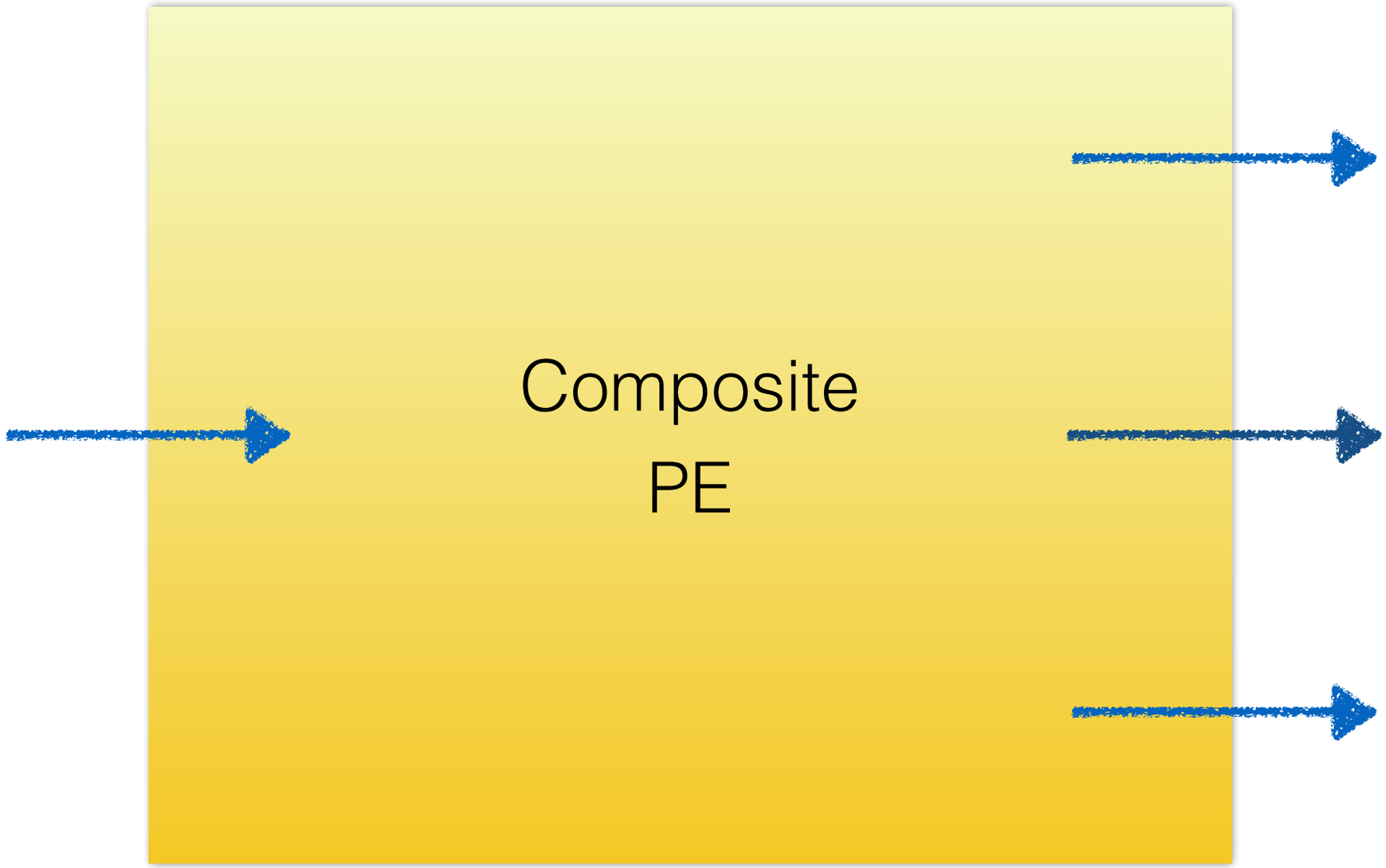


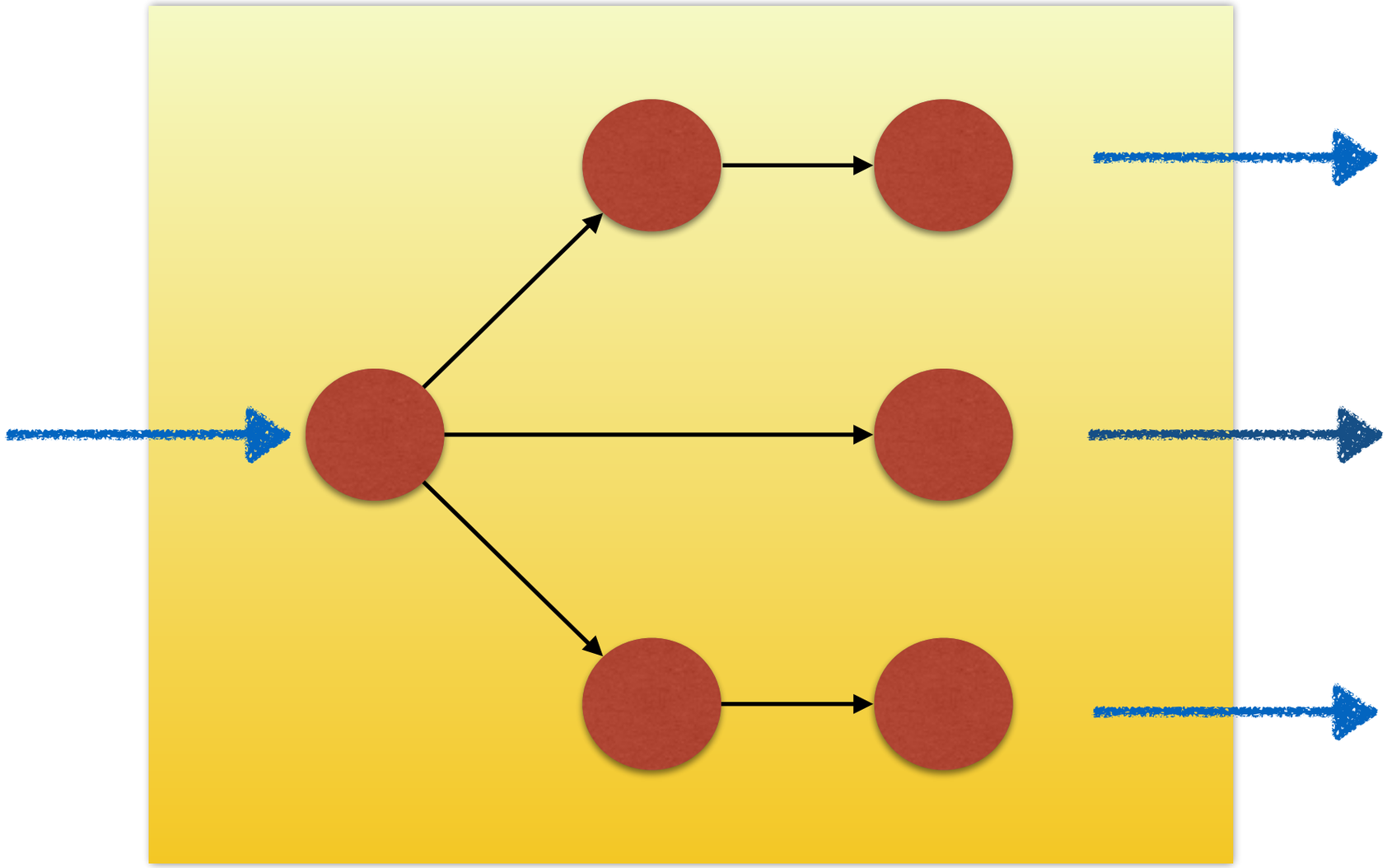
Unconnected

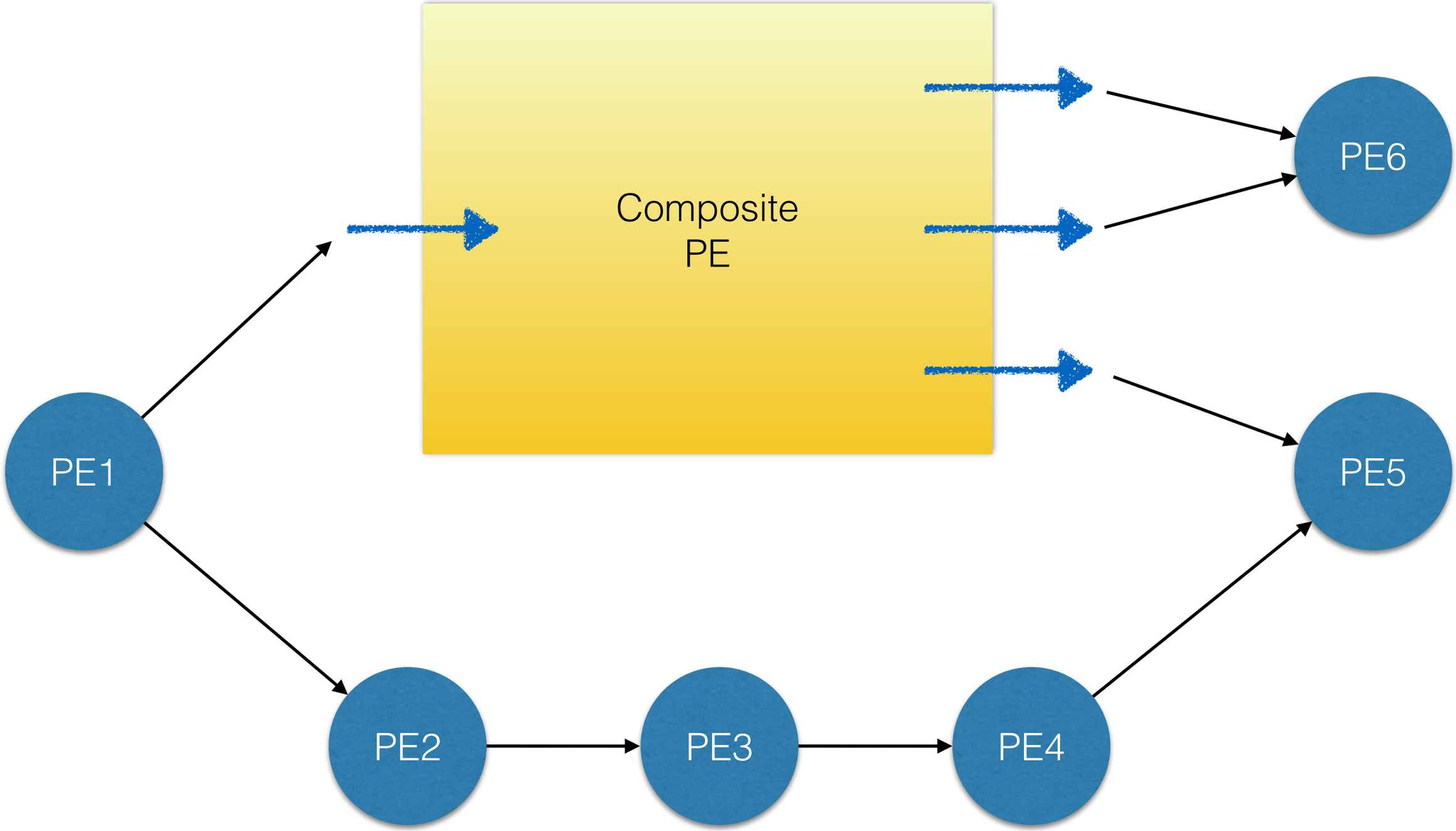


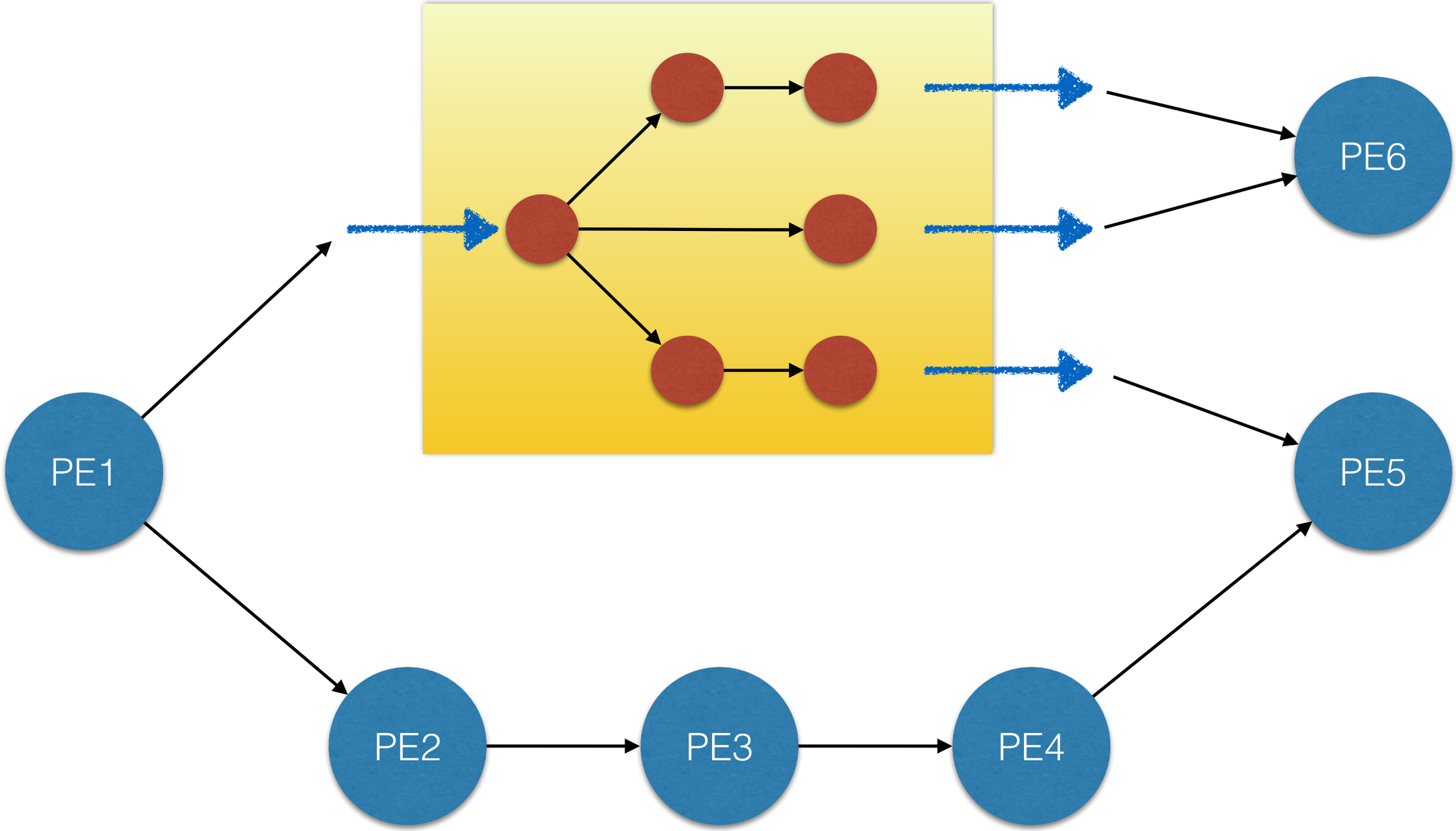
Composite PEs

- 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?

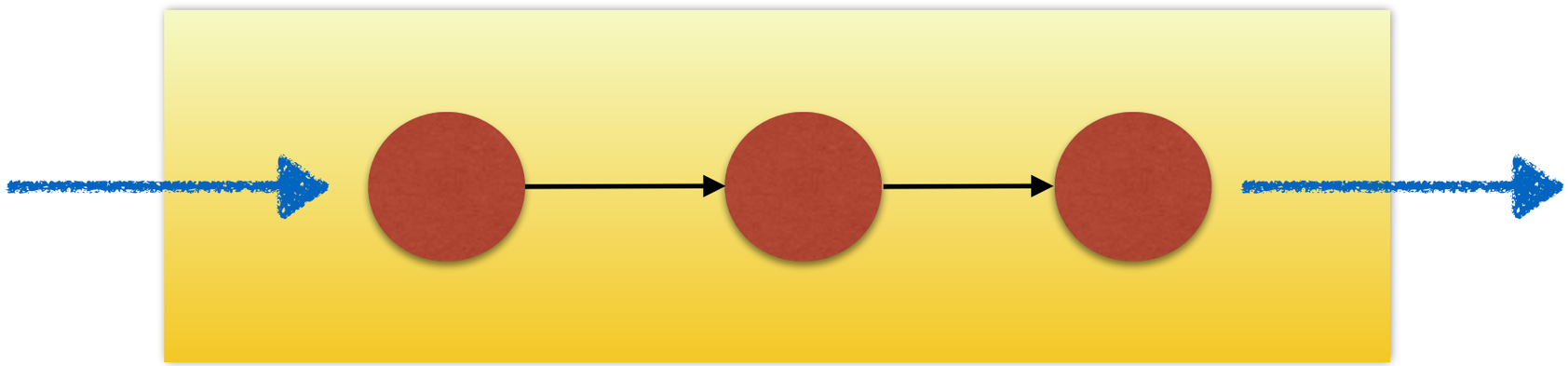
Dispel4Py provides a utility to create a composite PE with a chain of processing PEs:

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

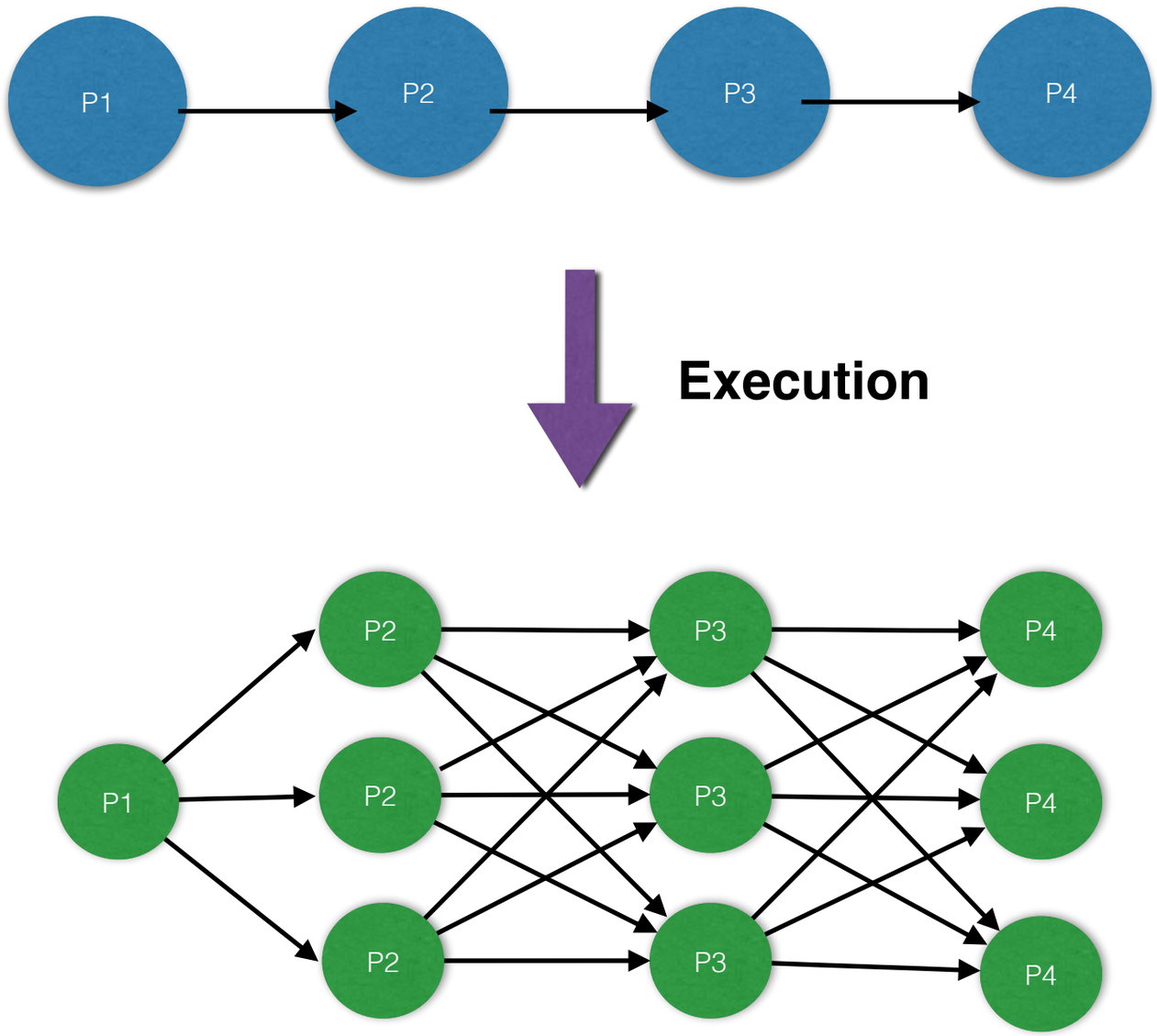
```
def detrend(st):
    return …
def demean(st):
    return …
```

```
pipeline = [ detrend, demean, … ]

compositePE = create_iterative_chain(pipeline)
```

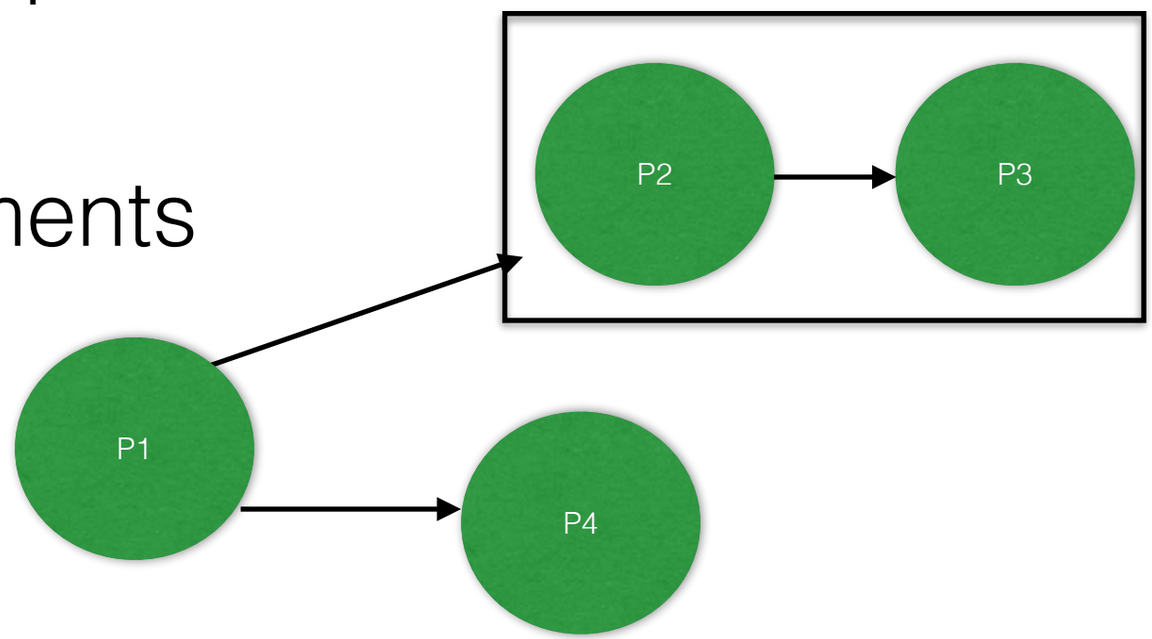


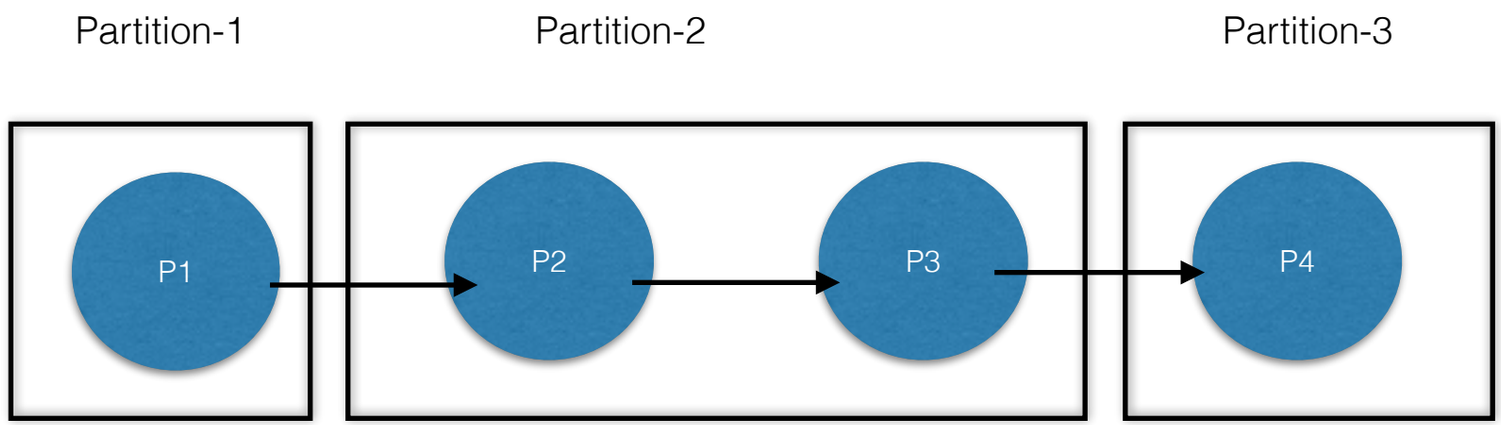
Executing graphs



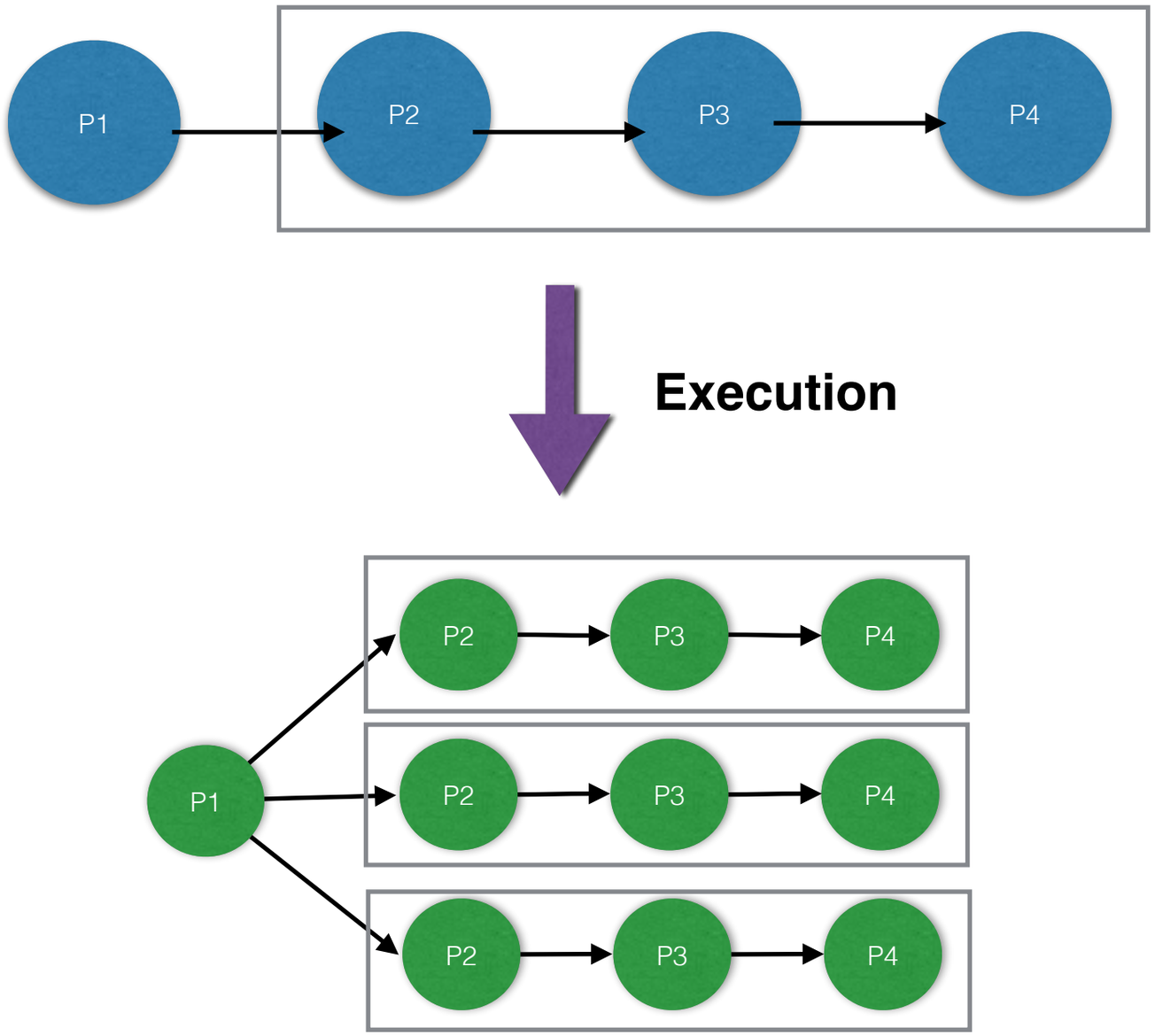
Partitions

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





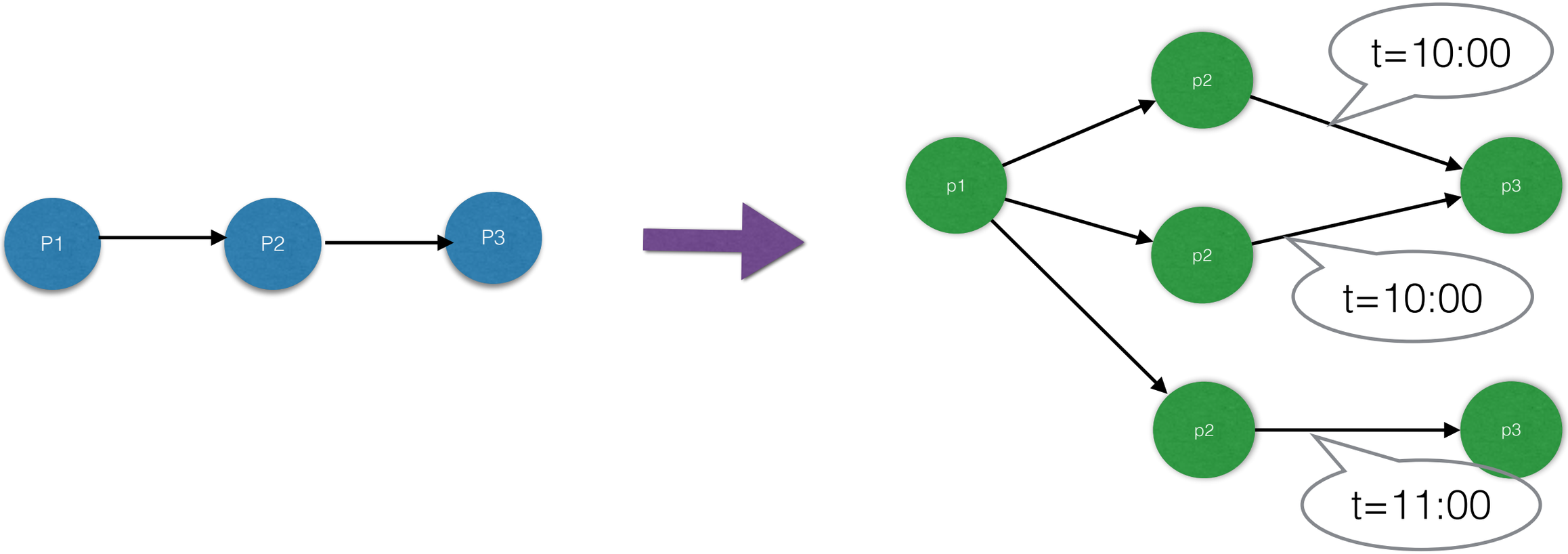
Partitioned Pipeline



Groupings

“Grouping by” a feature (MapReduce)

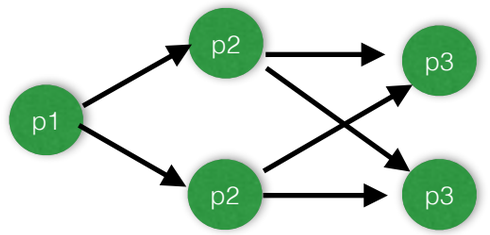
All data items that satisfy the same feature are guaranteed to be delivered to the same **instance** of a PE



Other groupings

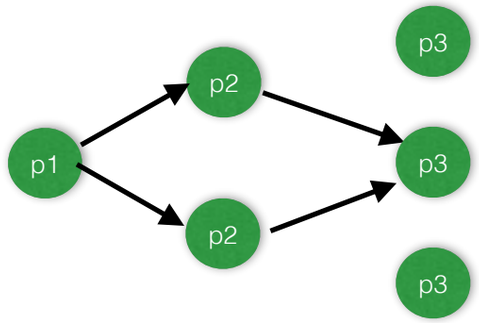
One-To-All

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

STORM mapping
>> python storm_submission.py <name_dispy_graph> <-m mode: [remote, local, create]>
E.g : dispel4py storm dispel4py.examples.graph_testing.pipeline_test -m remote