PyMOR Usage

Usage Examples: Custom Pipelines and Custom Pipeline Steps

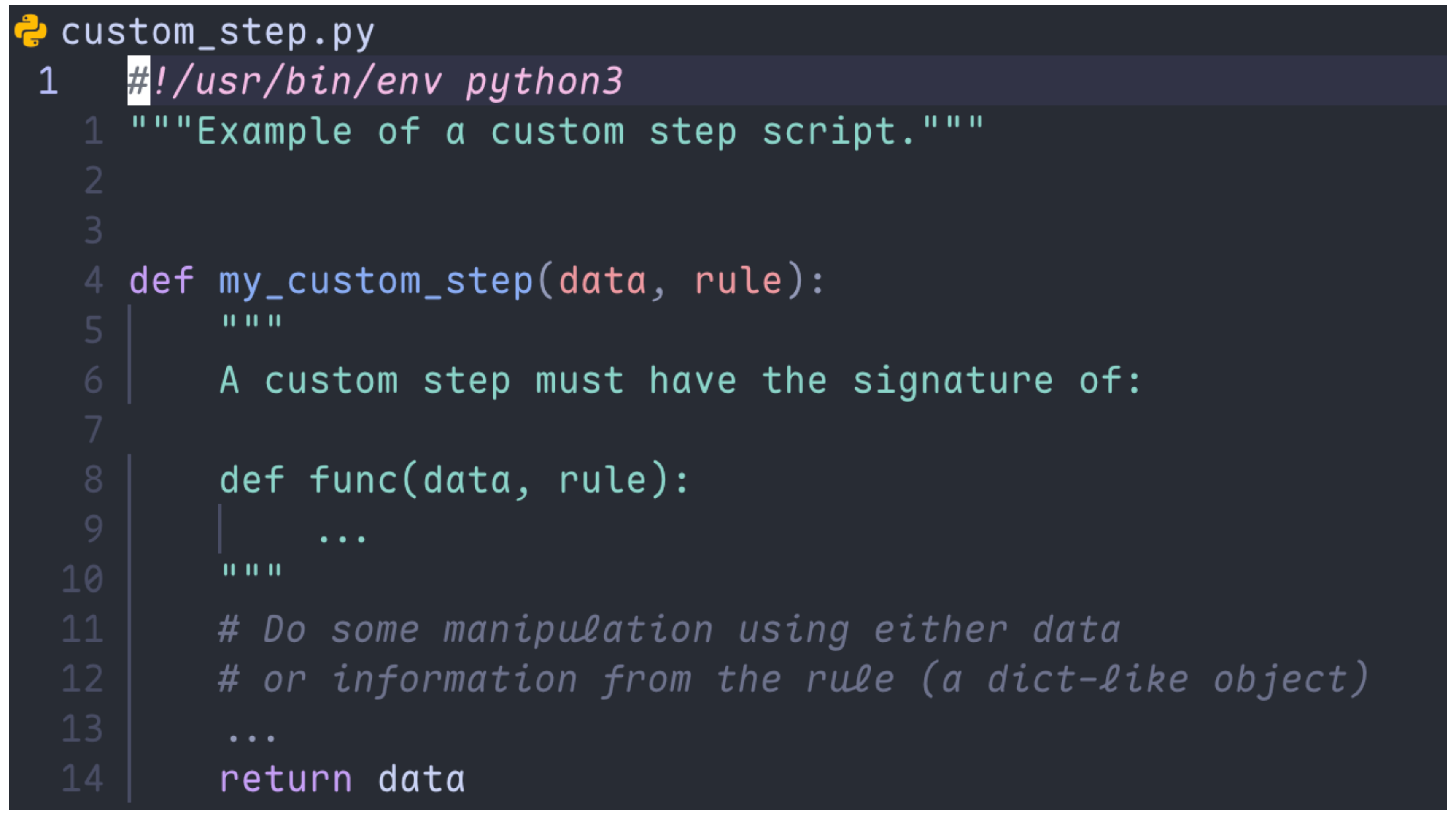
Paul Gierz

Motivation

- Most of the time, our DefaultPipeline should suit your needs for time averaging, unit conversion, and metadata. Sometimes you need more:
  - Mathematical manipulation of the data
  - Regridding
- Two Steps to a Solution:
  - Definition of on-the-fly pipeline with standard library steps
  - Inclusion of custom user logic

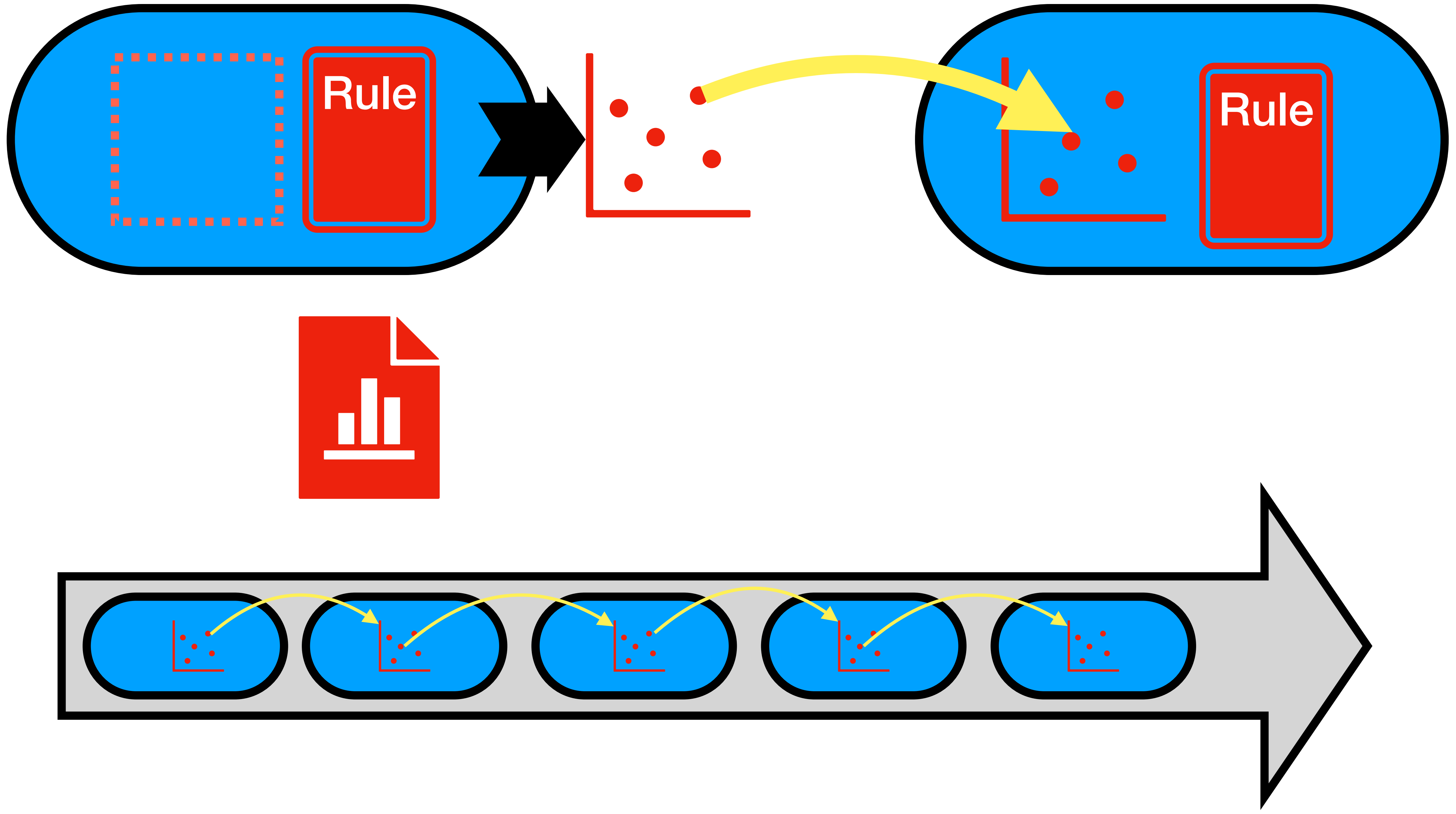
Theory: PyMOR's Pipelines

- `Pipeline` objects are composed of a series of steps, which are just Python functions with a special signature
- For each function, `data` is passed from one step to the next
- You can compose your own pipelines in your user YAML file.
- Adding steps from a script is easy to do.



custom_step.py

```python
#!/usr/bin/env python3
"""Example of a custom step script."""


def my_custom_step(data, rule):
    """
    A custom step must have the signature of:

    def func(data, rule):
        ...
    """
    # Do some manipulation using either data
    # or information from the rule (a dict-like object)
    ...
    return data
```



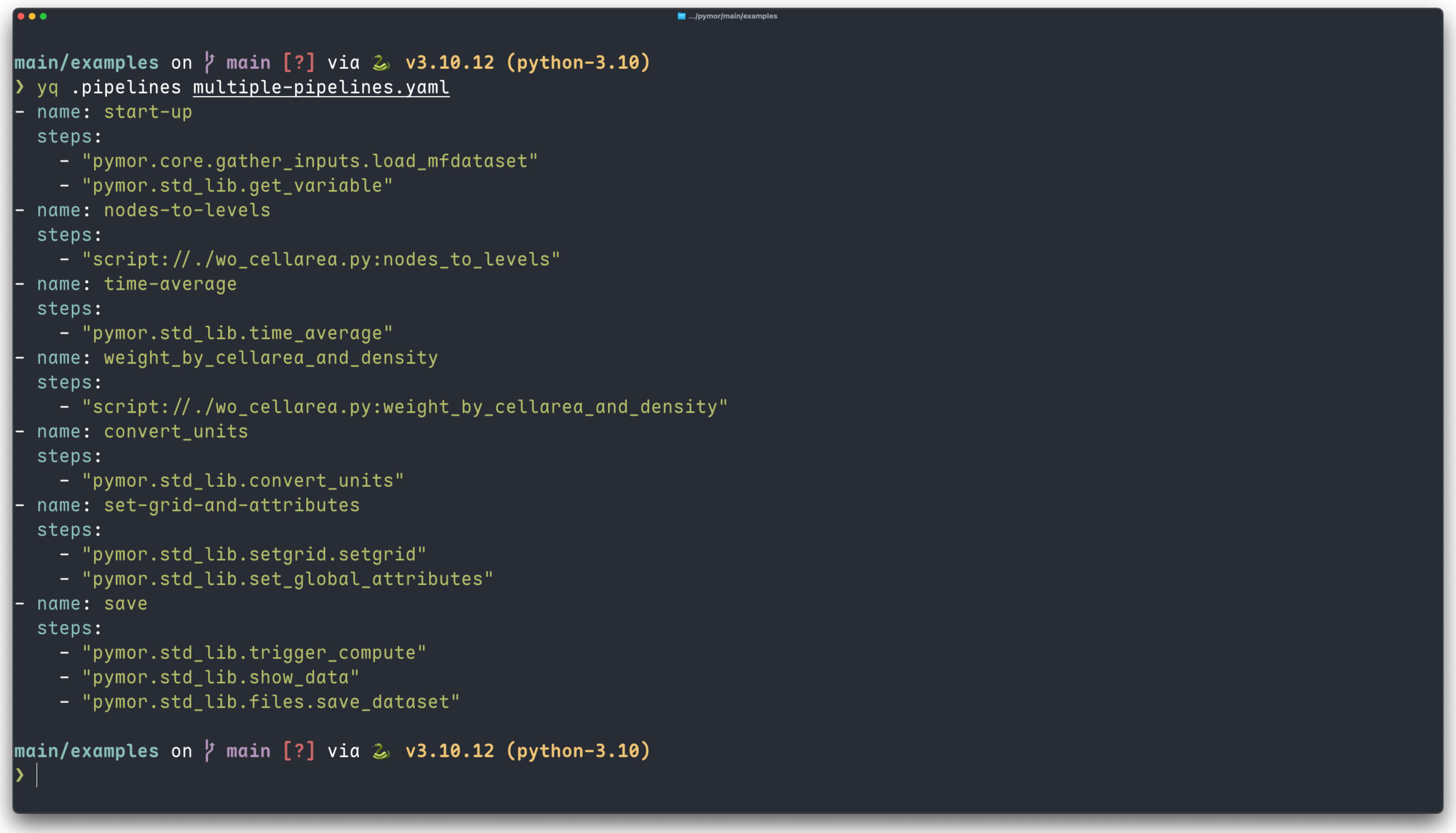
Theory: PyMOR’s Pipelines

- `Pipeline` objects are composed of a series of steps, which are just Python functions with a special signature

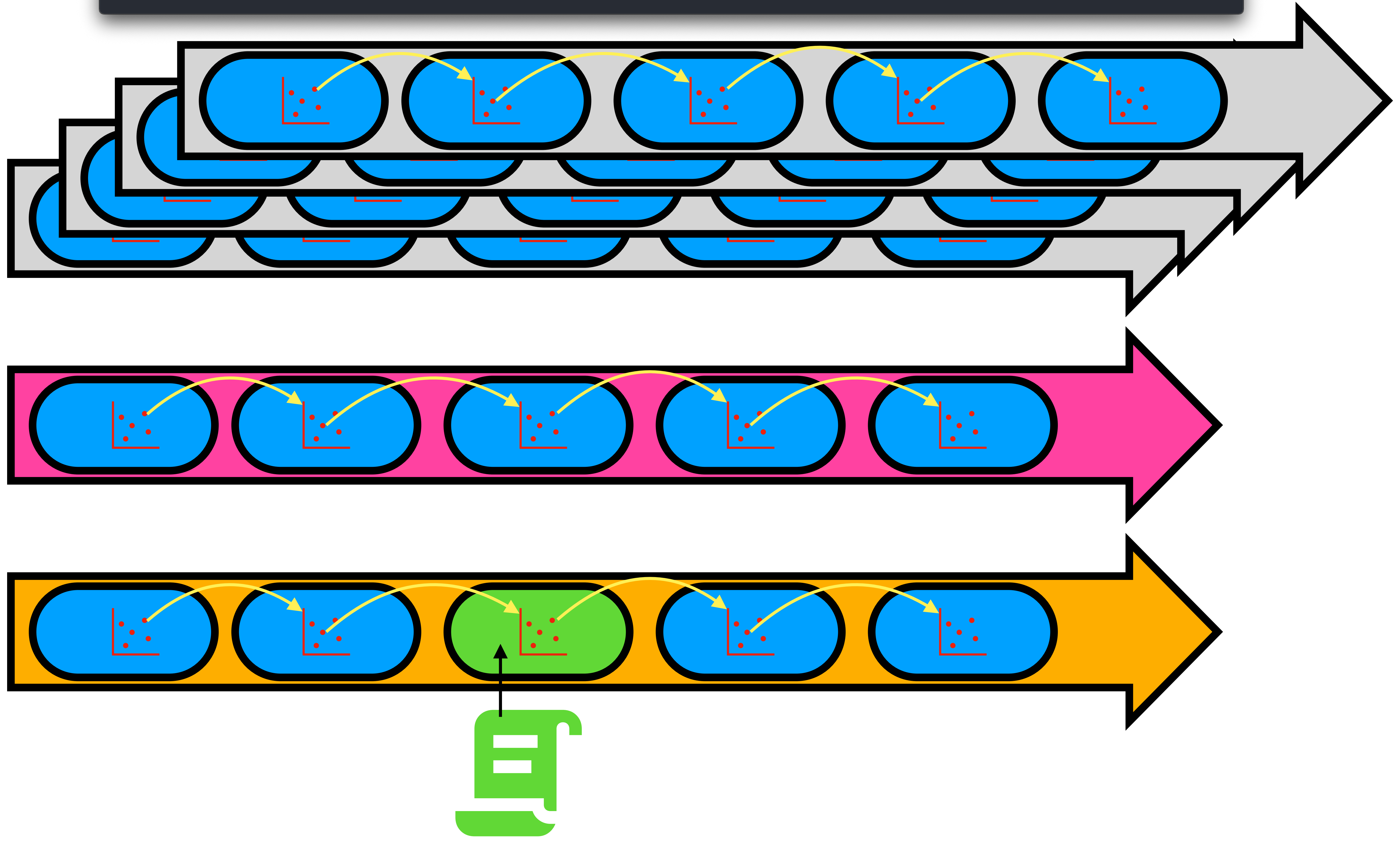
- For each function, `data` is passed from one step to the next

- You can compose your own pipelines in your user YAML file.

- Adding steps from a script is easy to do.



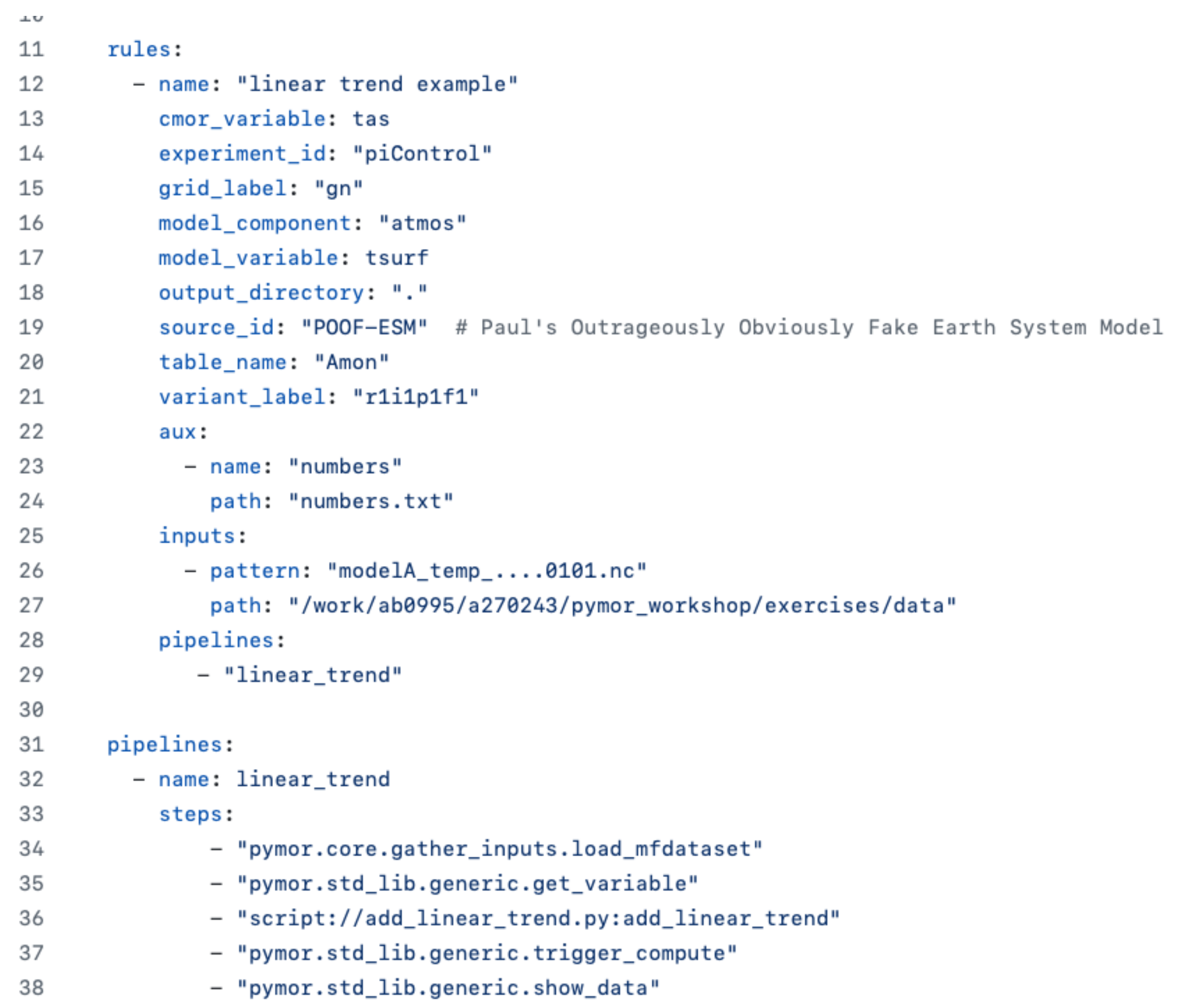
```
main/examples on ᚴ main [?] via 🐍 v3.10.12 (python-3.10)
❯ yq .pipelines multiple-pipelines.yaml
- name: start-up
  steps:
    - "pymor.core.gather_inputs.load_mfdataset"
    - "pymor.std_lib.get_variable"
- name: nodes-to-levels
  steps:
    - "script://./wo_cellarea.py:nodes_to_levels"
- name: time-average
  steps:
    - "pymor.std_lib.time_average"
- name: weight_by_cellarea_and_density
  steps:
    - "script://./wo_cellarea.py:weight_by_cellarea_and_density"
- name: convert_units
  steps:
    - "pymor.std_lib.convert_units"
- name: set-grid-and-attributes
  steps:
    - "pymor.std_lib.setgrid.setgrid"
    - "pymor.std_lib.set_global_attributes"
- name: save
  steps:
    - "pymor.std_lib.trigger_compute"
    - "pymor.std_lib.show_data"
    - "pymor.std_lib.files.save_dataset"

main/examples on ᚴ main [?] via 🐍 v3.10.12 (python-3.10)
❯ |
```



User Config Settings

Custom Pipeline

- `pipelines.[N]` can be a list of dictionaries, so long as you have `name` and `steps`
- Refer to `pipelines` in each `rules.[N].pipelines`



```yaml
rules:
  - name: "linear trend example"
    cmor_variable: tas
    experiment_id: "piControl"
    grid_label: "gn"
    model_component: "atmos"
    model_variable: tsurf
    output_directory: "."
    source_id: "POOF-ESM"  # Paul's Outrageously Obviously Fake Earth System Model
    table_name: "Amon"
    variant_label: "r1i1p1f1"
    aux:
      - name: "numbers"
        path: "numbers.txt"
    inputs:
      - pattern: "modelA_temp_....0101.nc"
        path: "/work/ab0995/a270243/pymor_workshop/exercises/data"
    pipelines:
       - "linear_trend"

pipelines:
  - name: linear_trend
    steps:
        - "pymor.core.gather_inputs.load_mfdataset"
        - "pymor.std_lib.generic.get_variable"
        - "script://add_linear_trend.py:add_linear_trend"
        - "pymor.std_lib.generic.trigger_compute"
        - "pymor.std_lib.generic.show_data"
```

User Defined Steps

- Need to start with a `script://` tag.
- You can provide a relative or full file path and function name, separated with a colon.
- Remember: this function runs in the same environment as `pymor`! You have access to the same Python libraries as the main program. Should you need something else, you need to install this.

Exercises

1. Split a pipeline into multiple sub pipelines
2. Write a custom step script