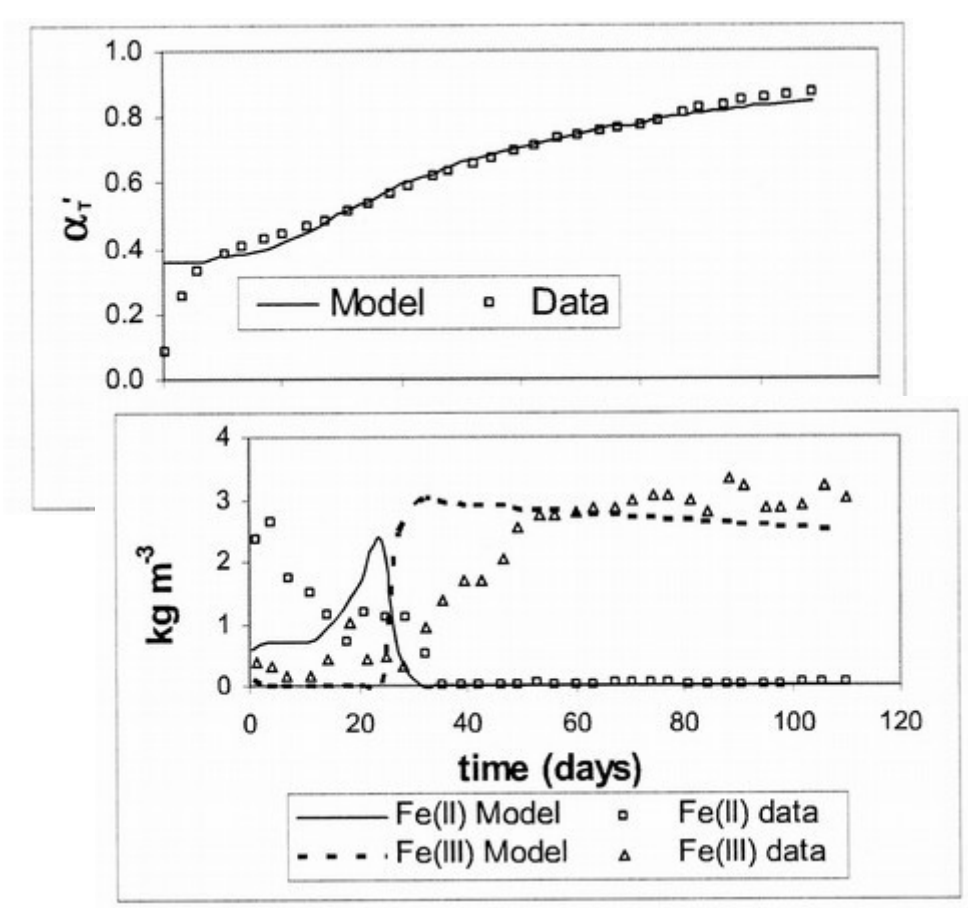
Python tools to help you manage your simulations

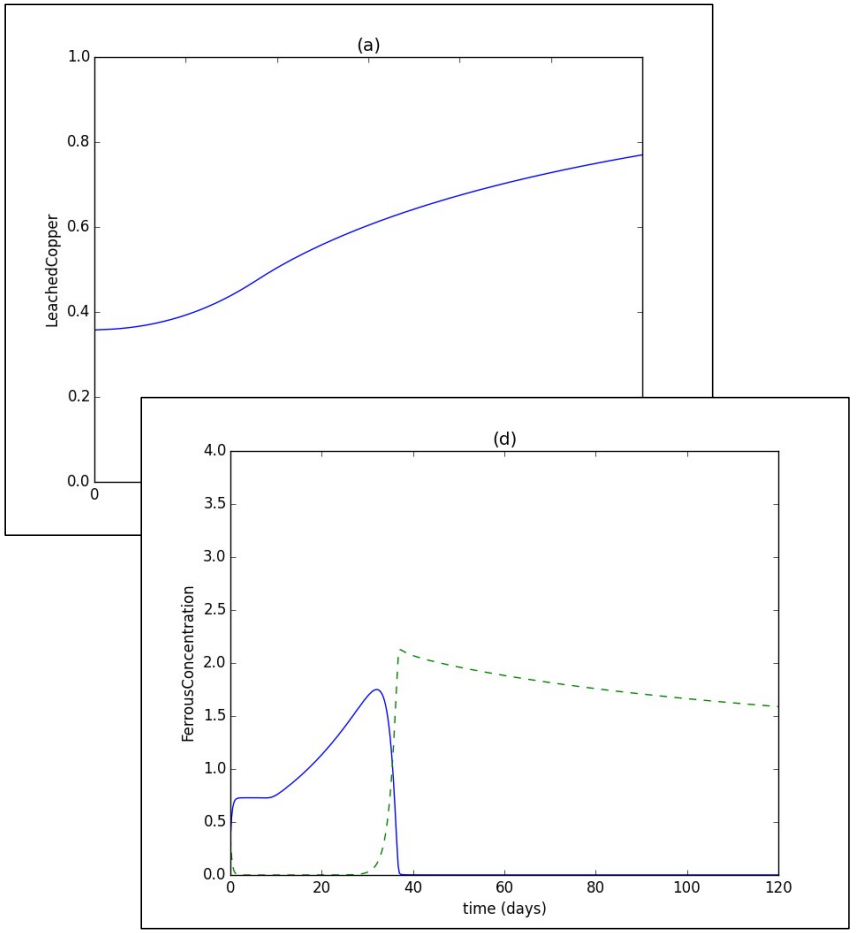
Outline

- Motivation
- Existing solutions
- Development
  - Main iterations
  - Dictionary features
- The Jinja2 Template Engine
- Example Application
- Conclusion

Motivation

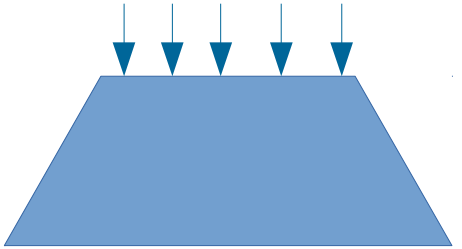
- Example 1: a “quick” simulation





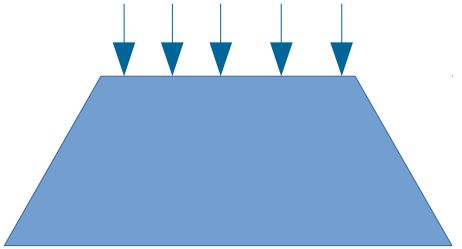
bioleach2d-2-slow.diml

-fast.diml



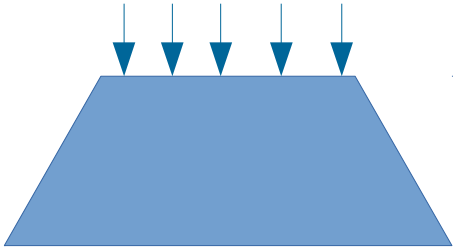
~~bioleach2d-2-slow.diml~~

~~-fast.diml~~



~~bioleach2d-2-slow.diml~~

~~-fast.diml~~



bioleach2d-2-slow-bc1.diml

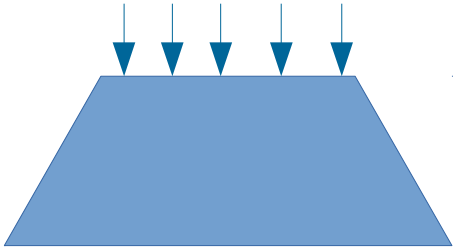
-bc2.diml

-fast-bc1.diml

-bc2.diml

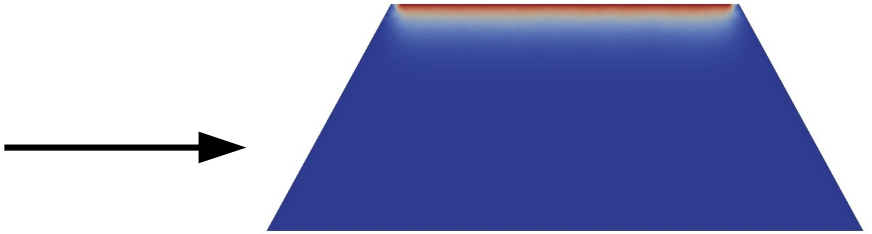
~~bioleach2d-2-slow.diml~~

~~-fast.diml~~



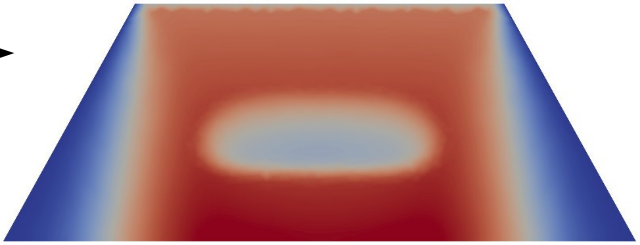
~~bioleach2d-2-slow-bc1.diml~~

-bc2.diml



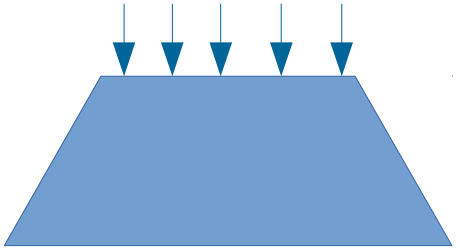
~~-fast-bc1.diml~~

-bc2.diml



~~bioleach2d-2-slow.diml~~

~~-fast.diml~~

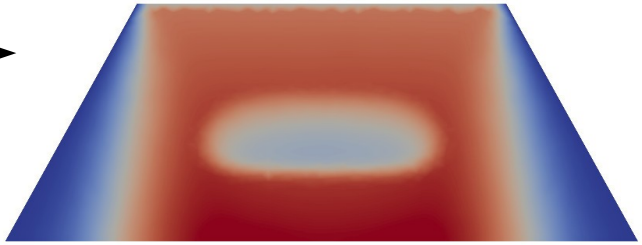


~~bioleach2d-2-slow-bc1.diml~~

-bc2.diml →

~~-fast-bc1.diml~~

-bc2.diml →



bioleach2d-2-slow-bc2-short.diml

-med.diml

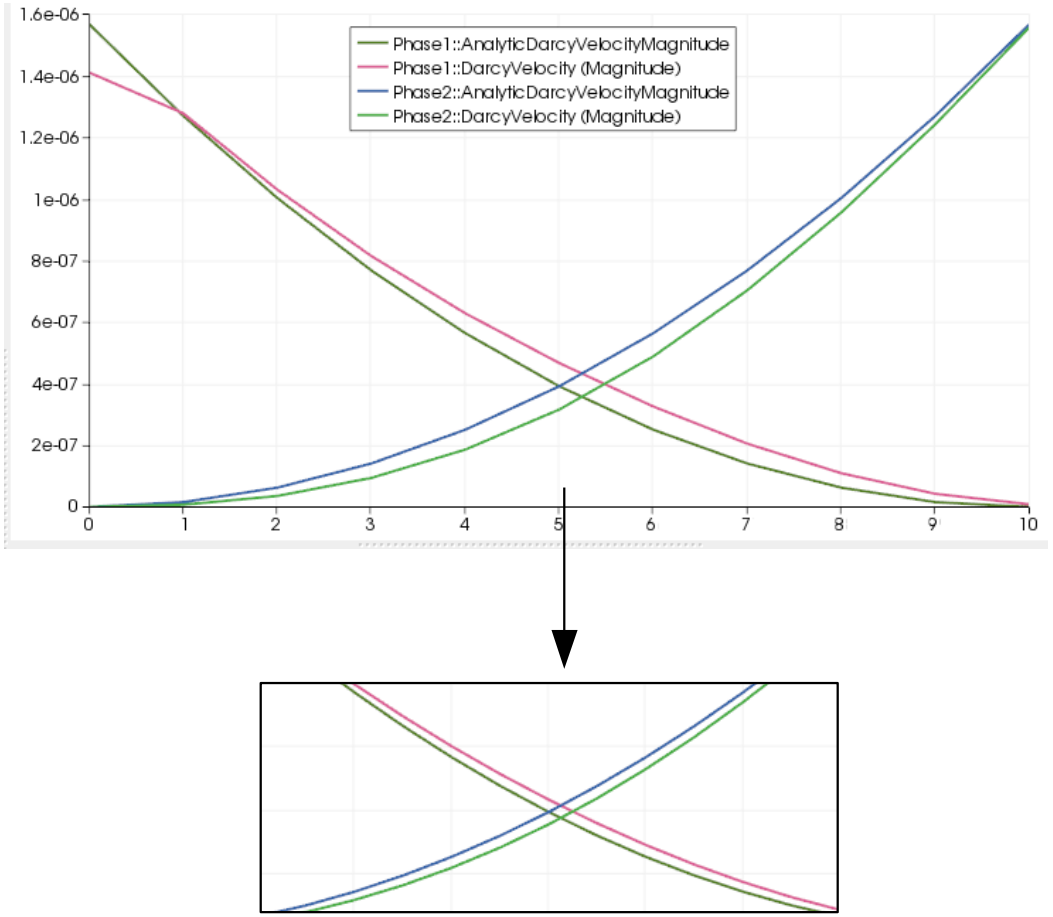
-long.diml

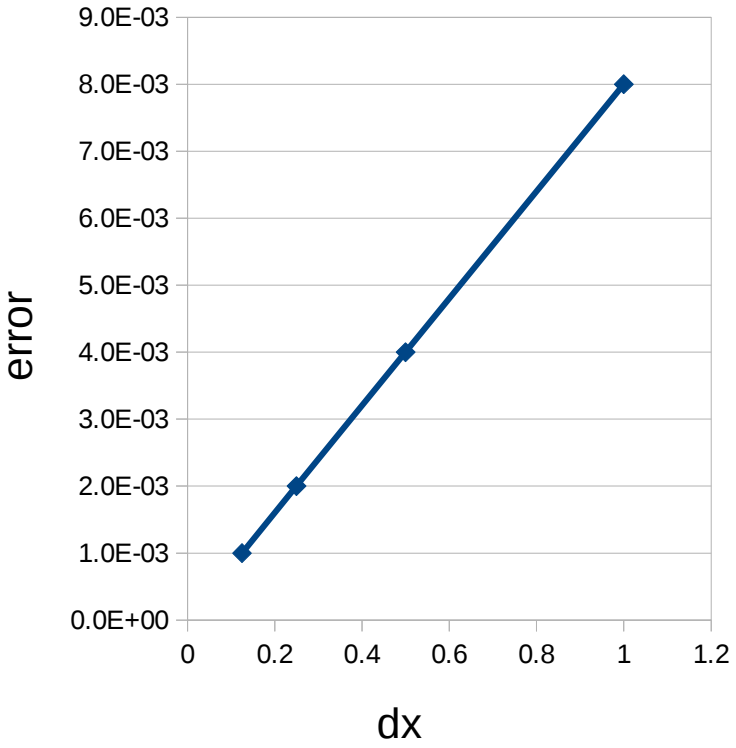
-5-slow-bc2-short.diml

-med.diml

-long.diml

- Example 2: convergence tests





…repeat for different domain dimensions, solver controls…

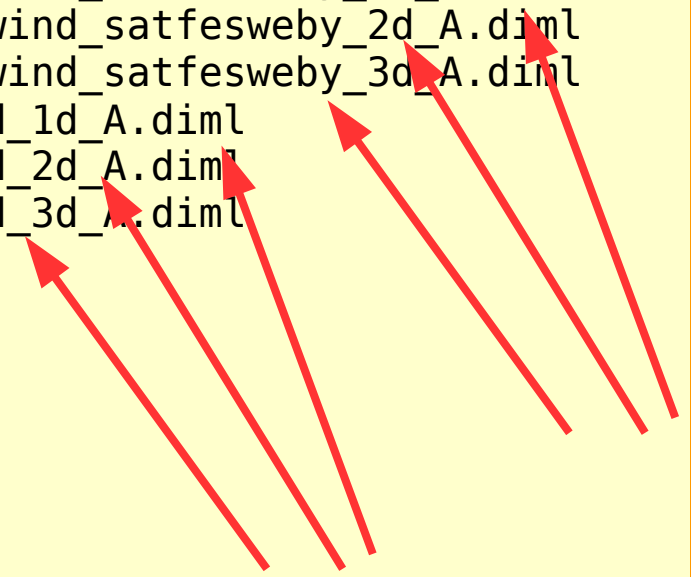
darcy_impes_p1_2phase_coreyrelperm_velBCinlet_strongpressoutlet_p1satdiag

```
Compare_Numerical_To_Analytic_1d.py
Linear_Interp_1D.py
Makefile
Run_Compare_Numerical_To_Analytic_1d_Multiple.py
darcy_impes_p1_2phase_..._p1satdiag.xml
darcy_impes_p1_2phase_..._p1satdiag_modrelpermupwind_satfesweby_1d_A.diml
darcy_impes_p1_2phase_..._p1satdiag_modrelpermupwind_satfesweby_2d_A.diml
darcy_impes_p1_2phase_..._p1satdiag_modrelpermupwind_satfesweby_3d_A.diml
darcy_impes_p1_2phase_..._p1satdiag_relpermupwind_1d_A.diml
darcy_impes_p1_2phase_..._p1satdiag_relpermupwind_2d_A.diml
darcy_impes_p1_2phase_..._p1satdiag_relpermupwind_3d_A.diml
mesh_data/
    cube_A.geo
    cube_A.msh
    cube_B.geo
    :
reference_solution/
    :
```

…repeat for different domain dimensions, solver controls…

darcy_impes_p1_2phase_coreyrelperm_velBCinlet_strongpressoutlet_p1satdiag

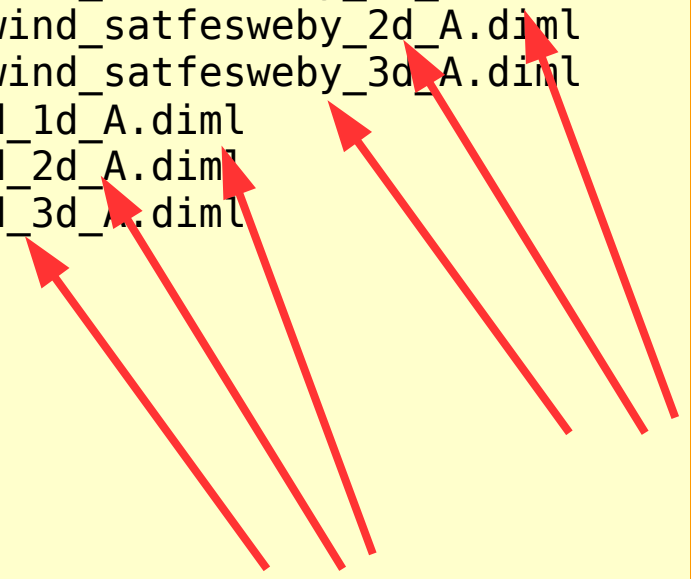
```
Compare_Numerical_To_Analytic_1d.py
Linear_Interp_1D.py
Makefile
Run_Compare_Numerical_To_Analytic_1d_Multiple.py
darcy_impes_p1_2phase_..._p1satdiag.xml
darcy_impes_p1_2phase_..._p1satdiag_modrelpermupwind_satfesweby_1d_A.diml
darcy_impes_p1_2phase_..._p1satdiag_modrelpermupwind_satfesweby_2d_A.diml
darcy_impes_p1_2phase_..._p1satdiag_modrelpermupwind_satfesweby_3d_A.diml
darcy_impes_p1_2phase_..._p1satdiag_relpermupwind_1d_A.diml
darcy_impes_p1_2phase_..._p1satdiag_relpermupwind_2d_A.diml
darcy_impes_p1_2phase_..._p1satdiag_relpermupwind_3d_A.diml
mesh_data/
    cube_A.geo
    cube_A.msh
    cube_B.geo
    :
reference_solution/
    :
```



…repeat for different domain dimensions, solver controls…

darcy_impes_p1_2phase_coreyrelperm_velBCinlet_strongpressoutlet_p1satdiag

```
Compare_Numerical_To_Analytic_1d.py
Linear_Interp_1D.py
Makefile
Run_Compare_Numerical_To_Analytic_1d_Multiple.py
darcy_impes_p1_2phase_..._p1satdiag.xml
darcy_impes_p1_2phase_..._p1satdiag_modrelpermupwind_satfesweby_1d_A.diml
darcy_impes_p1_2phase_..._p1satdiag_modrelpermupwind_satfesweby_2d_A.diml
darcy_impes_p1_2phase_..._p1satdiag_modrelpermupwind_satfesweby_3d_A.diml
darcy_impes_p1_2phase_..._p1satdiag_relpermupwind_1d_A.diml
darcy_impes_p1_2phase_..._p1satdiag_relpermupwind_2d_A.diml
darcy_impes_p1_2phase_..._p1satdiag_relpermupwind_3d_A.diml
mesh_data/
    cube_A.geo
    cube_A.msh
    cube_B.geo
    :
reference_solution/
    :
```



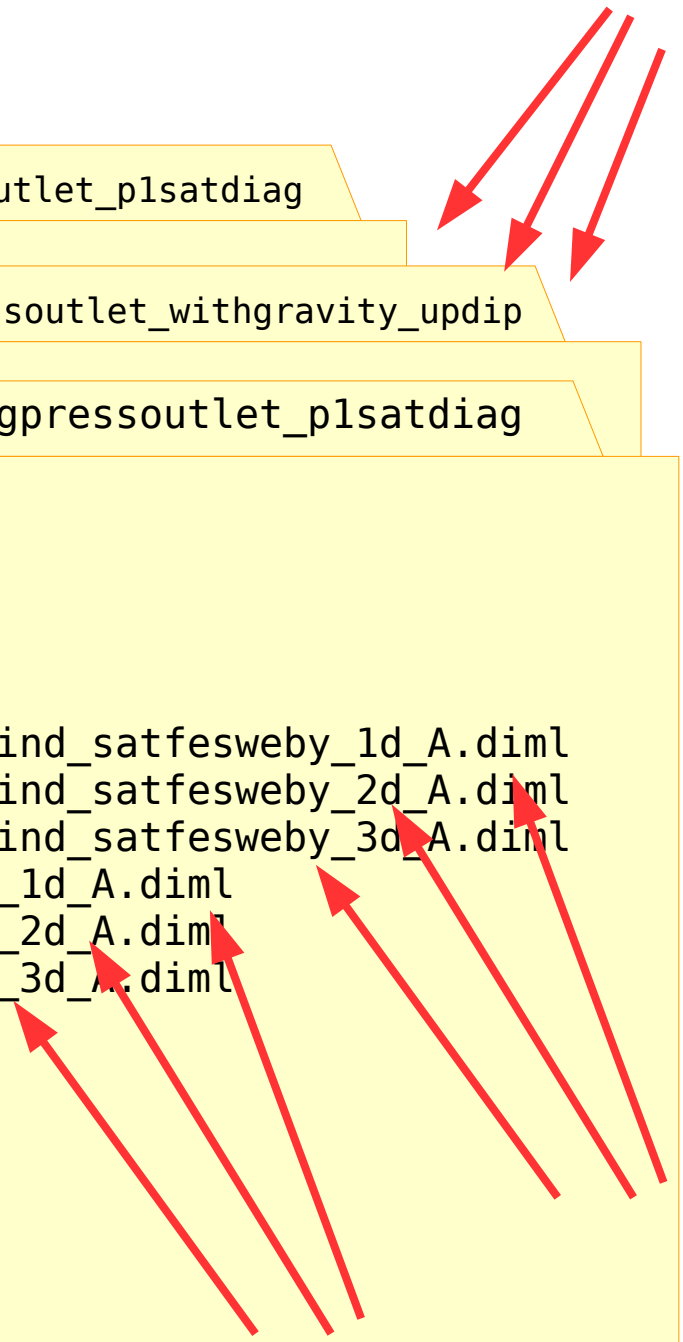
…model options…

darcy_impes_p1_2phase_quadraticrelperm_velBCinlet_strongpressoutlet_p1satdiag

darcy_impes_p1_2phase_quadraticrelperm_velBCinlet_strongpressoutlet_withgravity_updip

darcy_impes_p1_2phase_coreyrelperm_velBCinlet_strongpressoutlet_p1satdiag

```
Compare_Numerical_To_Analytic_1d.py
Linear_Interp_1D.py
Makefile
Run_Compare_Numerical_To_Analytic_1d_Multiple.py
darcy_impes_p1_2phase_..._p1satdiag.xml
darcy_impes_p1_2phase_..._p1satdiag_modrelpermupwind_satfesweby_1d_A.diml
darcy_impes_p1_2phase_..._p1satdiag_modrelpermupwind_satfesweby_2d_A.diml
darcy_impes_p1_2phase_..._p1satdiag_modrelpermupwind_satfesweby_3d_A.diml
darcy_impes_p1_2phase_..._p1satdiag_relpermupwind_1d_A.diml
darcy_impes_p1_2phase_..._p1satdiag_relpermupwind_2d_A.diml
darcy_impes_p1_2phase_..._p1satdiag_relpermupwind_3d_A.diml
mesh_data/
    cube_A.geo
    cube_A.msh
    cube_B.geo
    :
reference_solution/
    :
```



Merits

- Use of sed to reduce duplication

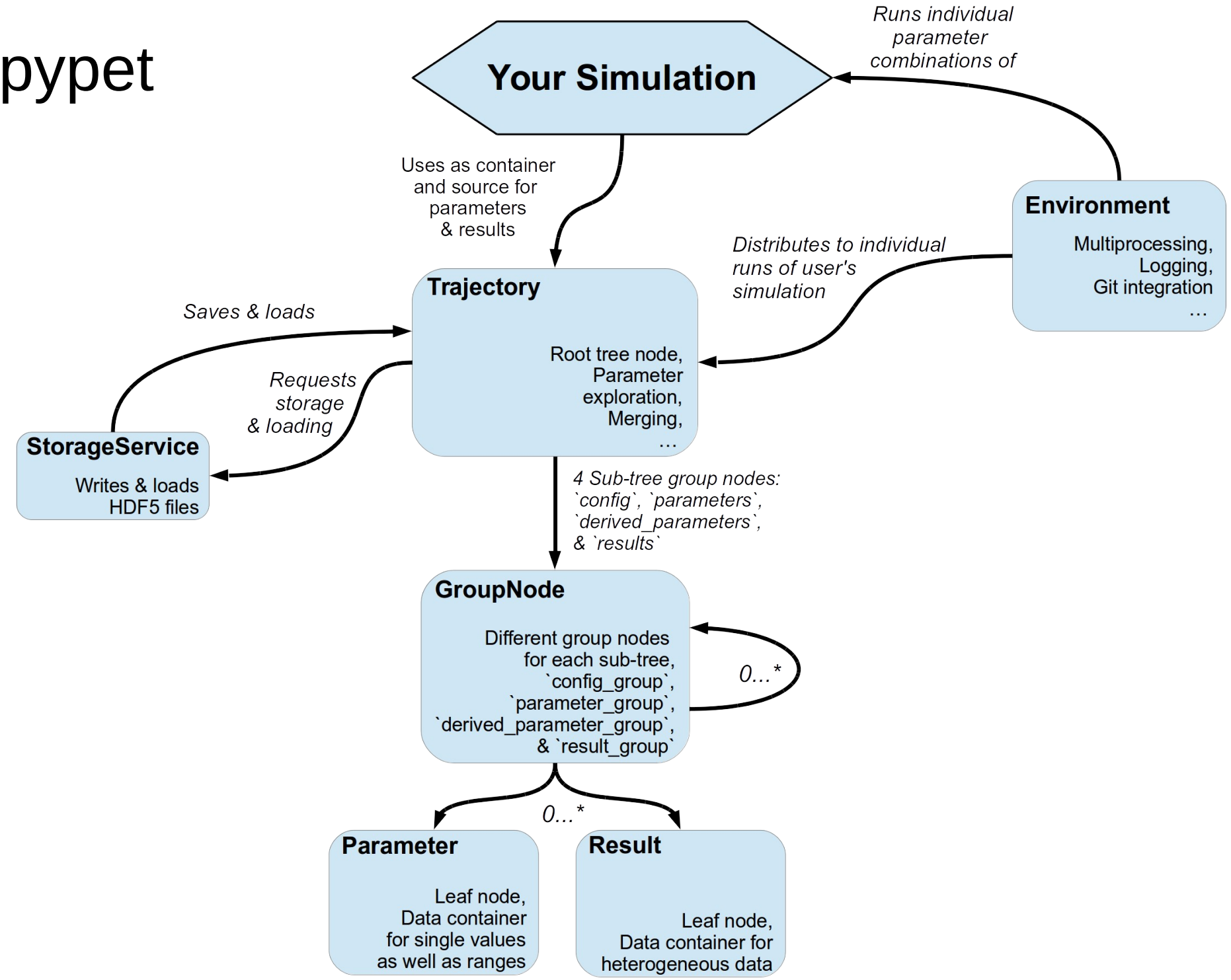
Problems

- Brittle; laborious to make changes
- Reliance on long filenames
- Lots of almost identical asserts
- Scales badly (amount of duplication ~ amount of test coverage)

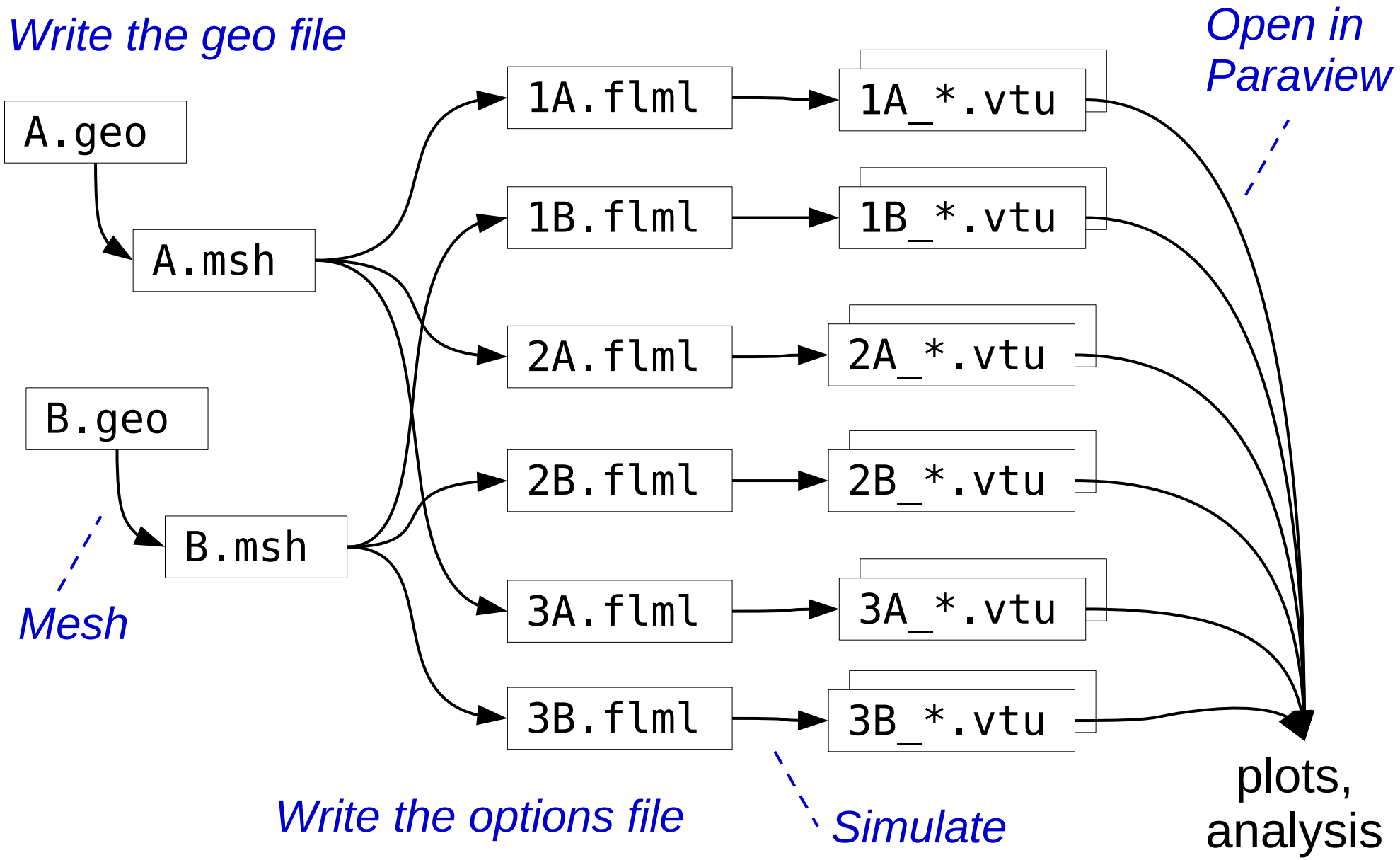
Existing Solutions

- create_param_sweep
  - Uses itertools.product to create full-factorial DOE
  - Manipulates a base options file via Spud bindings
- TreeDict / LazyRunner
  - Hierarchical Python container
  - System for caching results
- pypet...

pypet


Your Simulation
Runs individual parameter combinations of
Uses as container and source for parameters & results
Environment
Multiprocessing, Logging, Git integration
…
Distributes to individual runs of user's simulation
Trajectory
Root tree node, Parameter exploration, Merging, …
Saves & loads
Requests storage & loading
StorageService
Writes & loads HDF5 files
4 Sub-tree group nodes: `config`, `parameters`, `derived_parameters`, & `results`
GroupNode
Different group nodes for each sub-tree, `config_group`, `parameter_group`, `derived_parameter_group`, & `result_group`
0...*
0...*
Parameter
Leaf node, Data container for single values as well as ranges
Result
Leaf node, Data container for heterogeneous data

Our Workflow



Development

Requirements:

- General-purpose
- Low-level
  - Minimal new classes and methods
  - Maximal reuse of existing Python idioms
- Automatic filename generation
- Parallelisable

Concept

- Replace this:

```
for element in iterable:
    function(element)
```

with this:

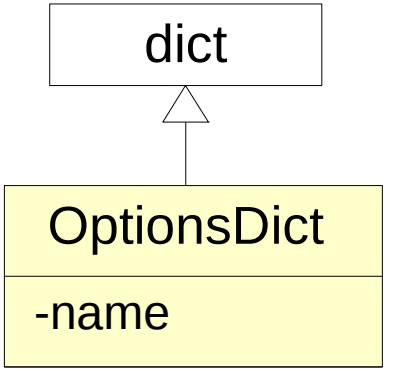
```
p = multiprocessing.Pool(nprocs)
p.map(function, iterable)
```

where

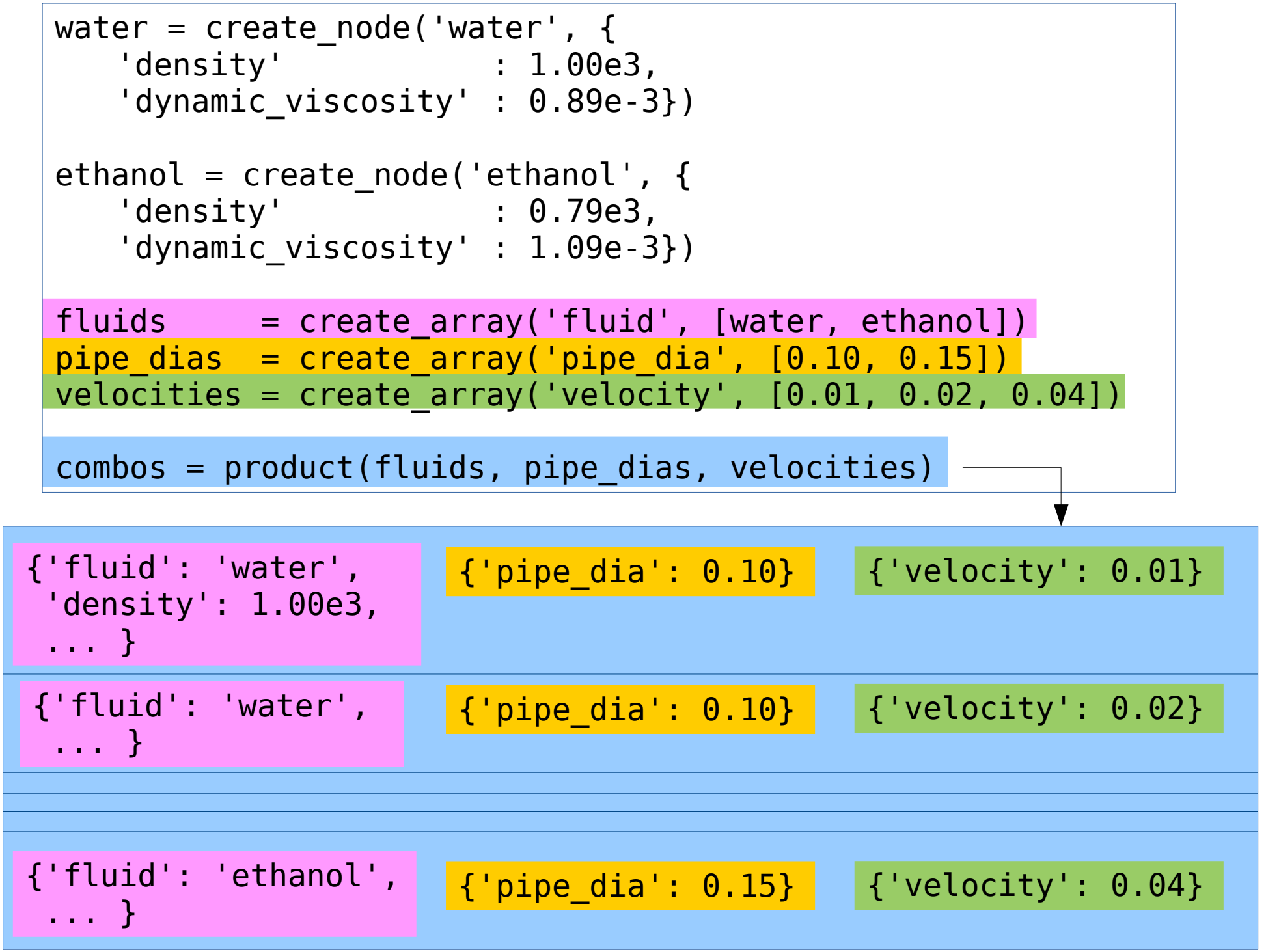
- `element` is a set of parameters
- `iterable` holds different sets of parameters
- `function` might expand a template, call Gmsh/Fluidity, scrape the results...

Iteration 1

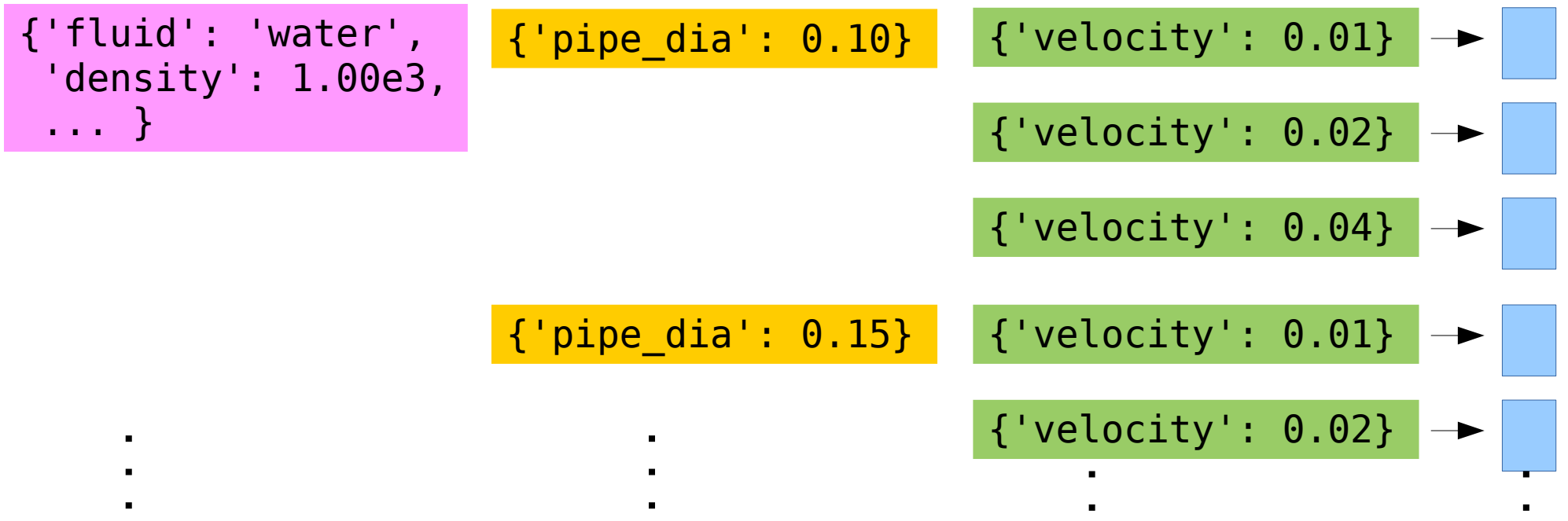
- Enhanced dictionary
- Use a list of dictionaries to represent parameter variations



- Use itertools to create combinations of parameters
- Use a merge function/decorator to
  - join names, forming an ID
  - turn a combination of dictionaries into a single dictionary


water = create_node('water', {
'density' : 1.00e3,
'dynamic_viscosity' : 0.89e-3})
ethanol = create_node('ethanol', {
'density' : 0.79e3,
'dynamic_viscosity' : 1.09e-3})
fluids = create_array('fluid', [water, ethanol])
pipe_dias = create_array('pipe_dia', [0.10, 0.15])
velocities = create_array('velocity', [0.01, 0.02, 0.04])
combos = product(fluids, pipe_dias, velocities)
{'fluid': 'water',
'density': 1.00e3,
... }
{'pipe_dia': 0.10}
{'velocity': 0.01}
{'fluid': 'water',
... }
{'pipe_dia': 0.10}
{'velocity': 0.02}
{'fluid': 'ethanol',
... }
{'pipe_dia': 0.15}
{'velocity': 0.04}

```
water = create_node('water', {
    'density'           : 1.00e3,
    'dynamic_viscosity' : 0.89e-3})

ethanol = create_node('ethanol', {
    'density'           : 0.79e3,
    'dynamic_viscosity' : 1.09e-3})

fluids     = create_array('fluid', [water, ethanol])
pipe_dias  = create_array('pipe_dia', [0.10, 0.15])
velocities = create_array('velocity', [0.01, 0.02, 0.04])

combos = product(fluids, pipe_dias, velocities)
```



```
for combo in combos:
    od = merge(combo)
    Re = od['density'] * od['velocity'] * \
       od['pipe_dia'] / od['dynamic_viscosity']
    print 'Test ID = {}, Reynolds number = {:.2e}'.\
        format(str(od), Re)
```

```
{'fluid': 'water', ... , 'pipe_dia': 0.10, 'velocity': 0.01}
{'fluid': 'water', ... , 'pipe_dia': 0.10, 'velocity': 0.01}
{'fluid': 'water', ... , 'pipe_dia': 0.10, 'velocity': 0.01}
                                        :
                                        :
{'fluid': 'ethanol', ... , 'pipe_dia': 0.15, 'velocity': 0.04}
```

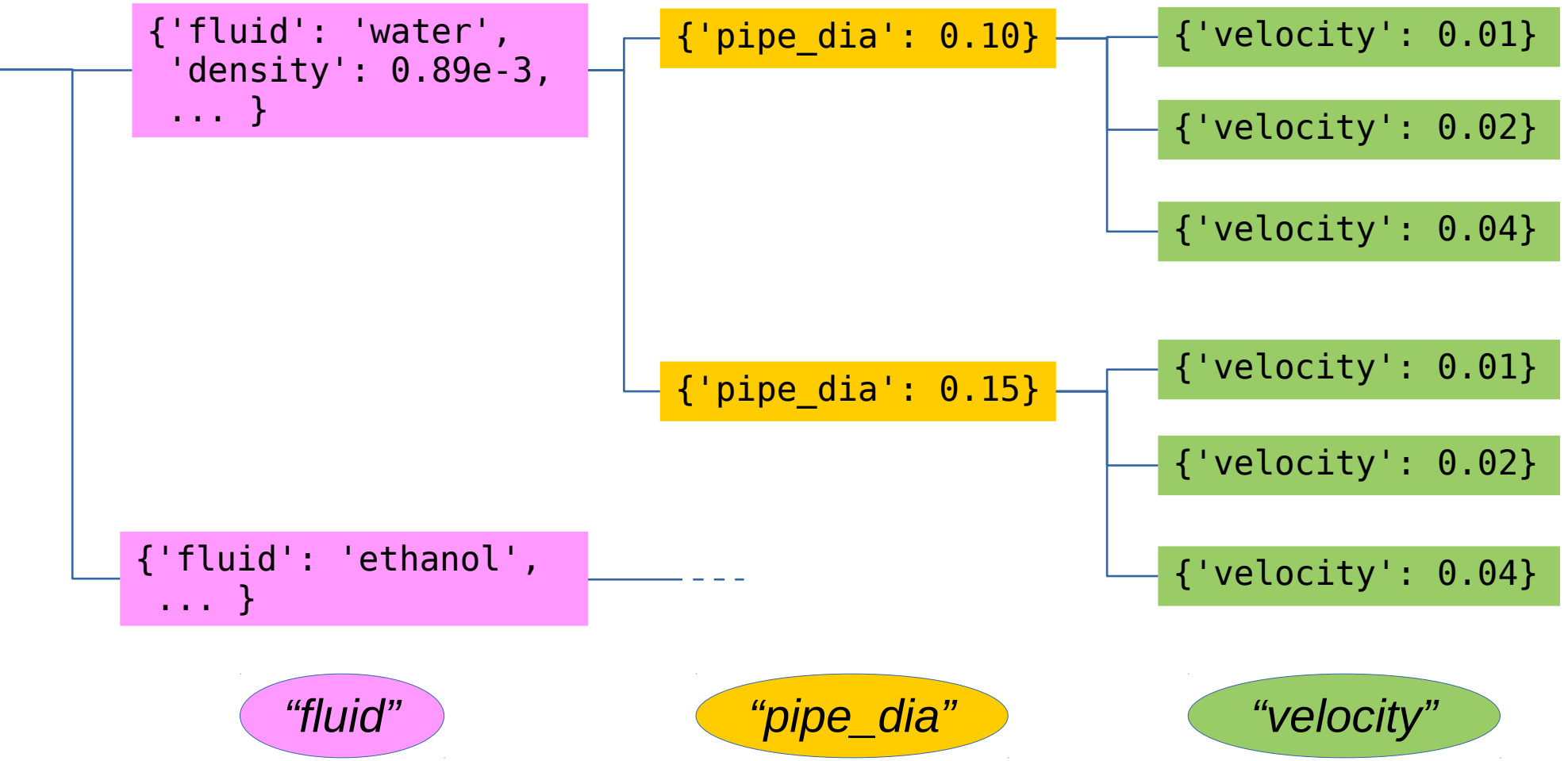
```
for combo in combos:
    od = merge(combo)
    Re = od['density'] * od['velocity'] * \
       od['pipe_dia'] / od['dynamic_viscosity']
    print 'Test ID = {}, Reynolds number = {:.2e}'.\
        format(str(od), Re)
```

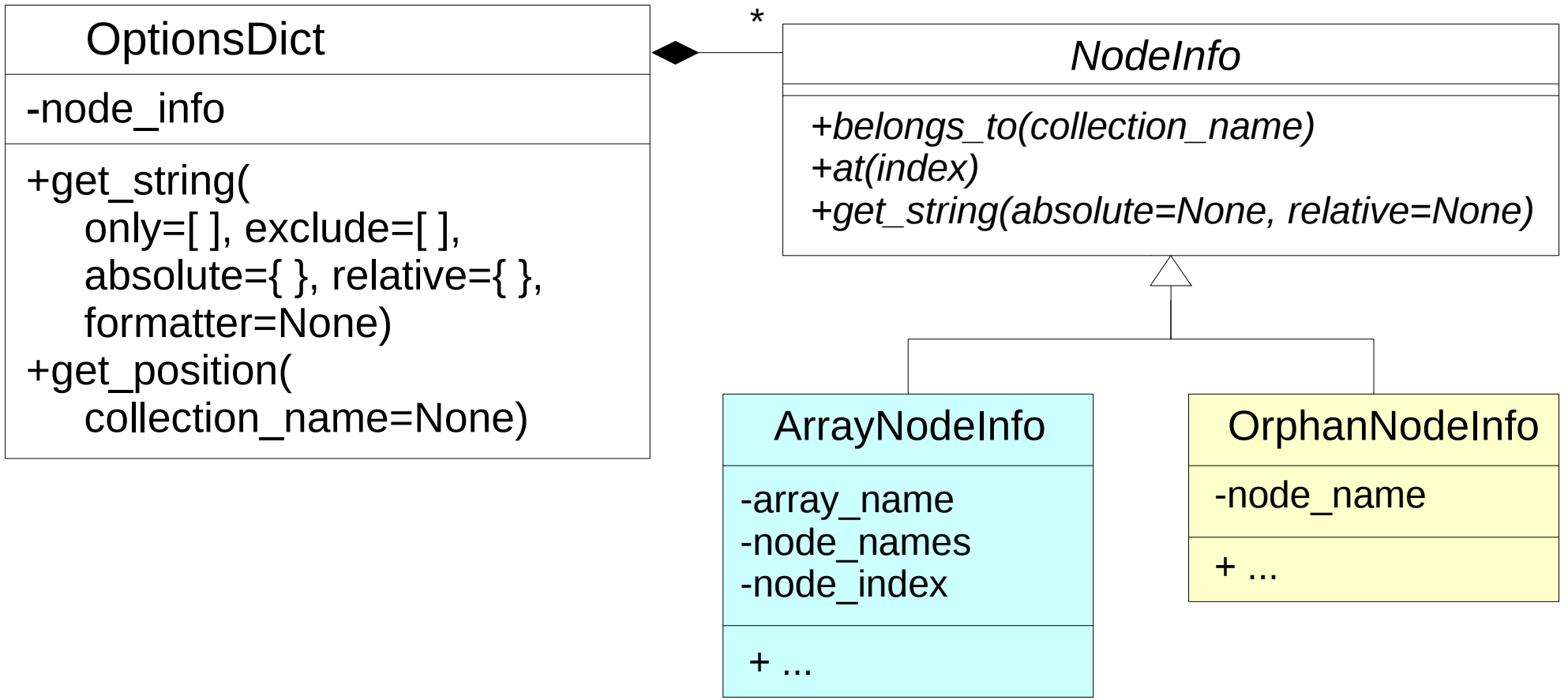
```
@merges_dicts
def calculate_Re(od):
    return od['density'] * od['velocity'] * \
       od['pipe_dia'] / od['dynamic_viscosity']

p = multiprocessing.Pool(4)
Reynolds_numbers = p.map(calculate_Re, combos)
```

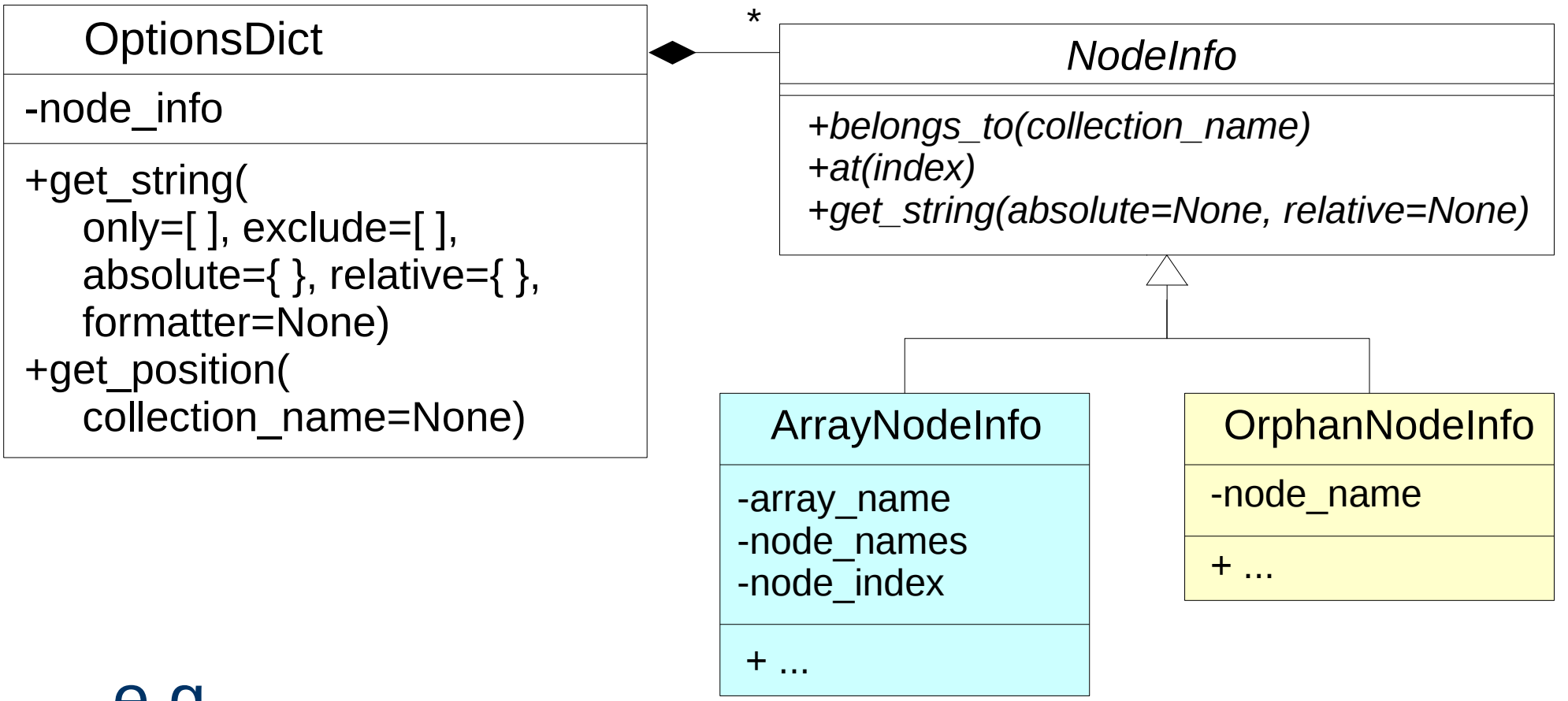
```
{'fluid': 'water', ... , 'pipe_dia': 0.10, 'velocity': 0.01}
{'fluid': 'water', ... , 'pipe_dia': 0.10, 'velocity': 0.01}
{'fluid': 'water', ... , 'pipe_dia': 0.10, 'velocity': 0.01}
                                   :
                                   :
{'fluid': 'ethanol', ... , 'pipe_dia': 0.15, 'velocity': 0.04}
```

Iteration 2

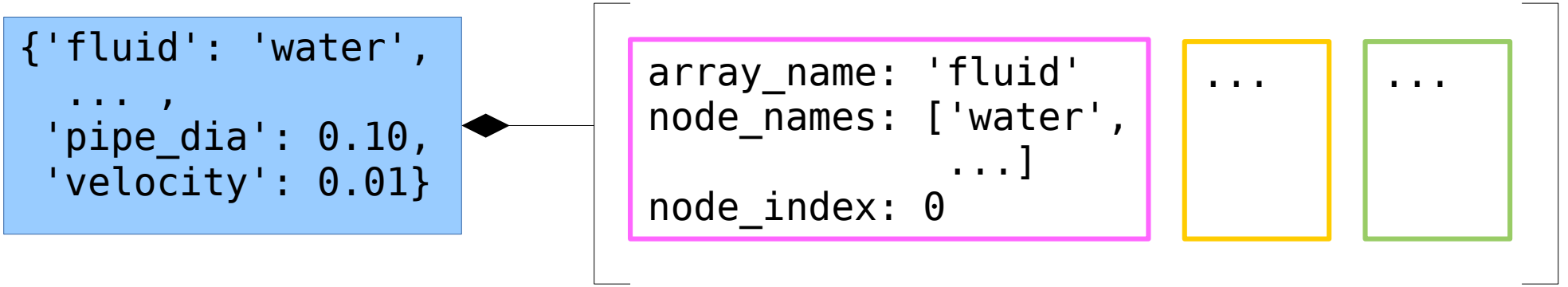
- Encapsulate tree information to manage IDs




OptionsDict
-node_info
+get_string(
only=[ ], exclude=[ ],
absolute={ }, relative={ },
formatter=None)
+get_position(
collection_name=None)
*
NodeInfo
+belongs_to(collection_name)
+at(index)
+get_string(absolute=None, relative=None)
ArrayNodeInfo
-array_name
-node_names
-node_index
+ ...
OrphanNodeInfo
-node_name
+ ...

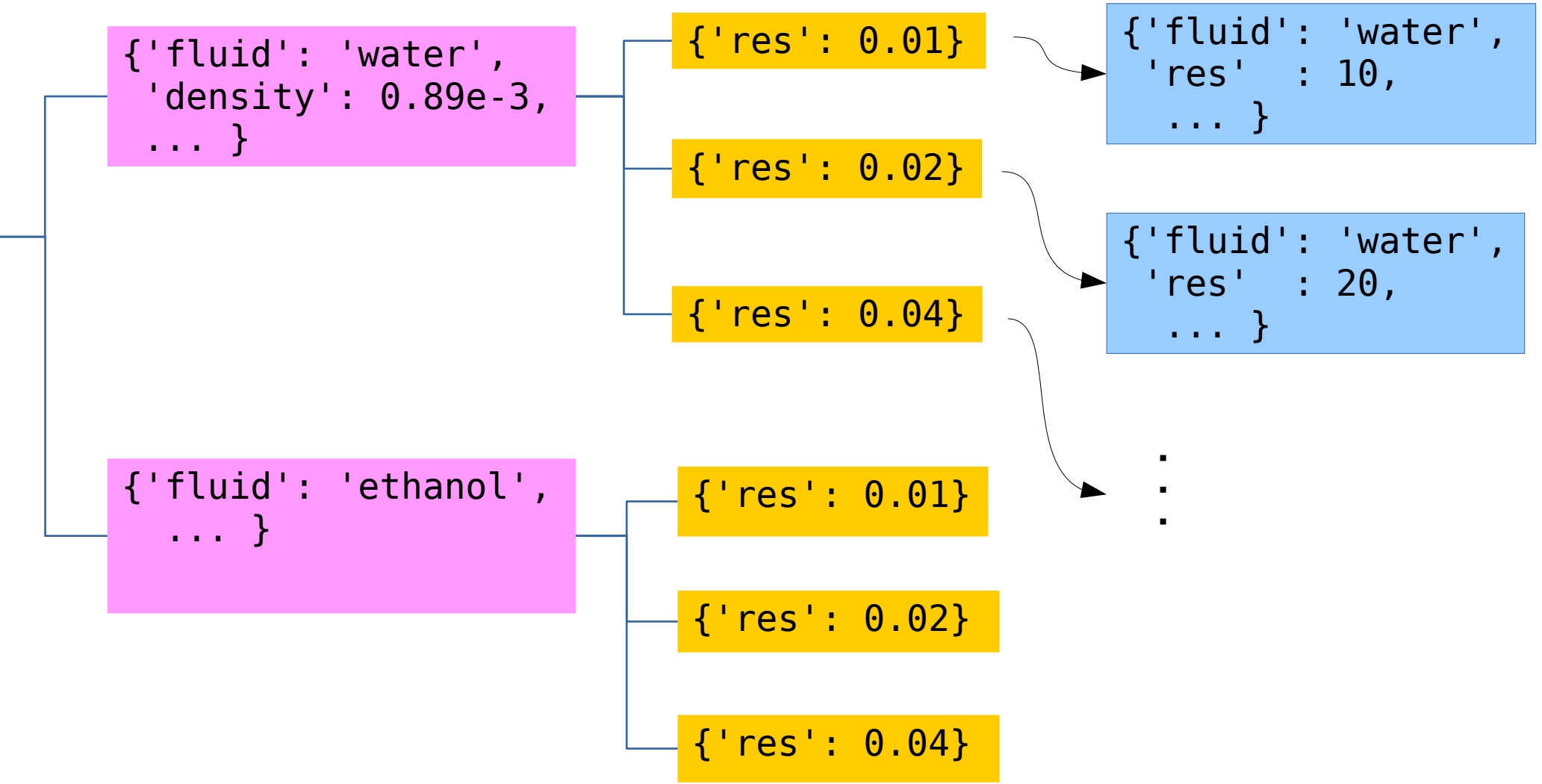

OptionsDict
-node_info
+get_string(
only=[ ], exclude=[ ],
absolute={ }, relative={ },
formatter=None)
+get_position(
collection_name=None)
*
NodeInfo
+belongs_to(collection_name)
+at(index)
+get_string(absolute=None, relative=None)
ArrayNodeInfo
-array_name
-node_names
-node_index
+ ...
OrphanNodeInfo
-node_name
+ ...

e.g.

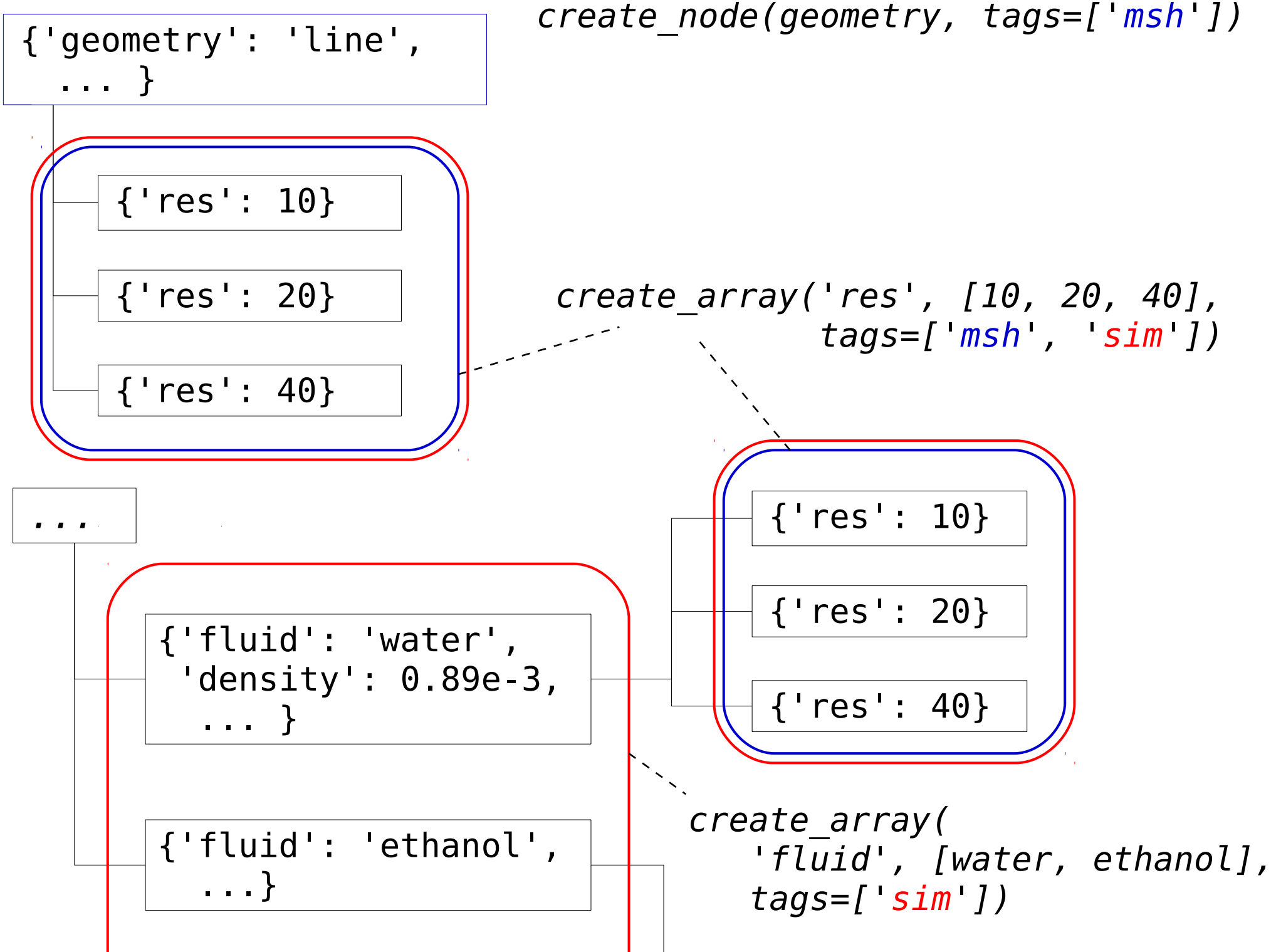

{'fluid': 'water',
... ,
'pipe_dia': 0.10,
'velocity': 0.01}
array_name: 'fluid'
node_names: ['water',
...]
node_index: 0
...
...

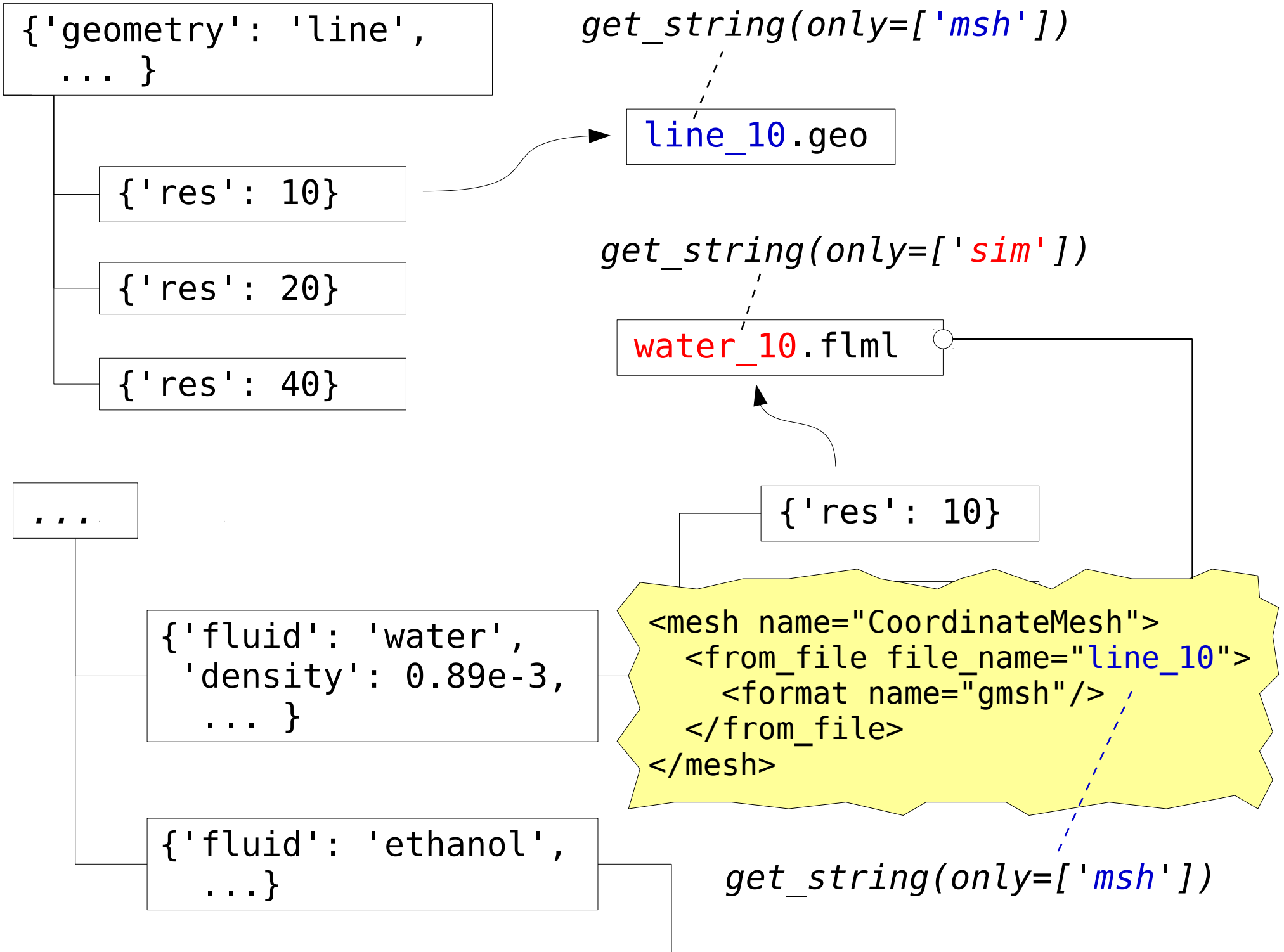
- Example:

```
map( StudyConvergence(), combos )
```



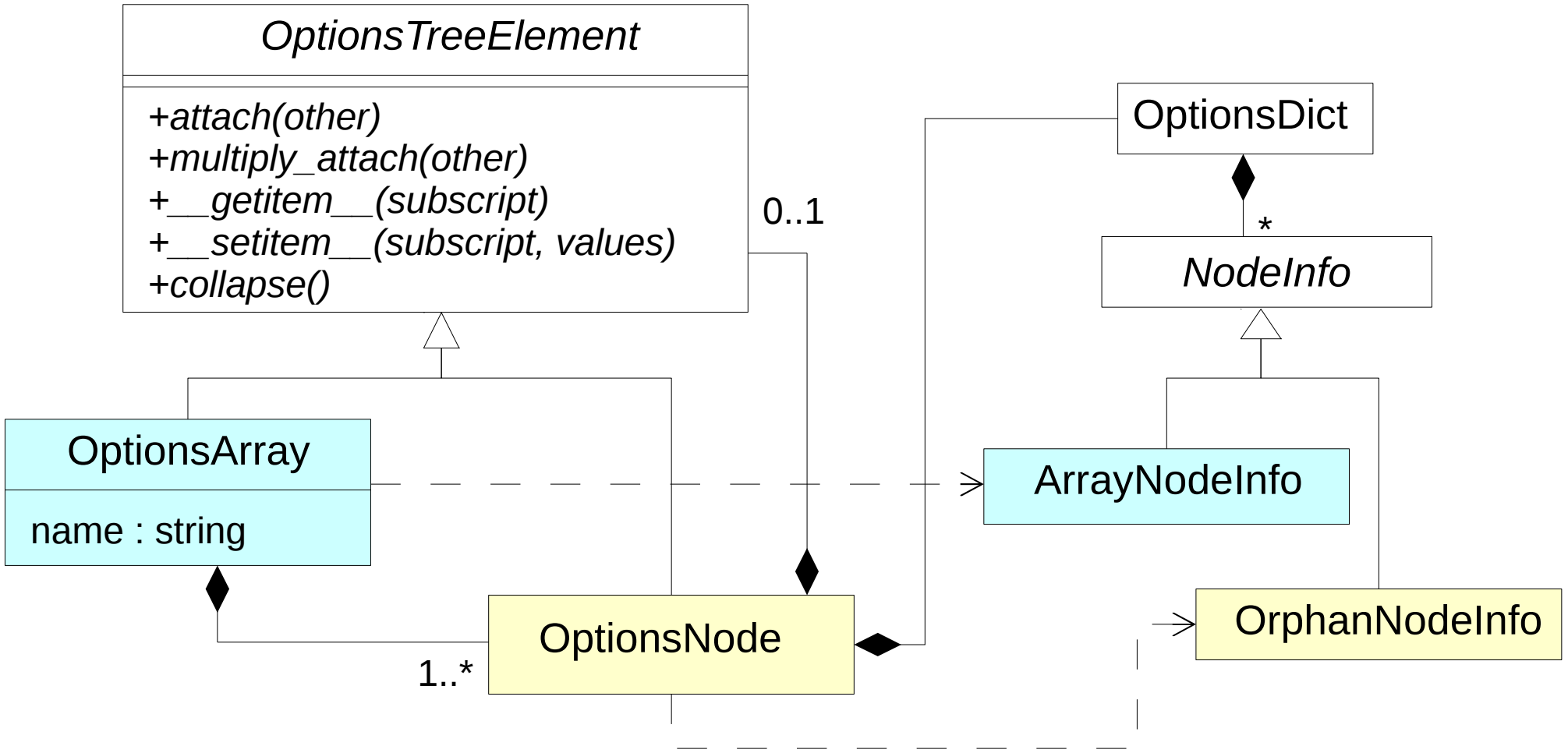
```
class StudyConvergence:
    def __init__(self):
        self.resolutions = {}; self.errors = {}; self.rates = {}

    @merges_dicts
    def __call__(self, options_dict):
        current_id = options.get_string(only=['sim'])
        current_res = float(options['res'])
        current_err = self.get_error_norm(options)

        self.resolutions[current_id] = current_res
        self.errors[current_id] = current_err
        try:
            previous_id = options.get_string(only=['sim'],
                                             relative={'res': -1})
            previous_res = self.resolutions[previous_id]
            previous_err = self.errors[previous_id]
            self.rates[current_id] = \
                 -log(current_err/previous_err) / \
                 log(current_res/previous_res)
        except:
            self.rates[current_id] = nan   # on the first mesh

    def get_error_norm(self, options):
        ...
```


{'geometry': 'line',
... }
create_node(geometry, tags=['msh'])
{'res': 10}
{'res': 20}
{'res': 40}
create_array('res', [10, 20, 40],
tags=['msh', 'sim'])
...
{'res': 10}
{'res': 20}
{'res': 40}
{'fluid': 'water',
'density': 0.89e-3,
... }
{'fluid': 'ethanol',
...}
create_array(
'fluid', [water, ethanol],
tags=['sim'])


{'geometry': 'line',
... }
{'res': 10}
{'res': 20}
{'res': 40}
get_string(only=['msh'])
line_10.geo
get_string(only=['sim'])
water_10.flml
{'res': 10}
...
{'fluid': 'water',
'density': 0.89e-3,
... }
{'fluid': 'ethanol',
...}
<mesh name="CoordinateMesh">
<from_file file_name="line_10">
<format name="gmsh"/>
</from_file>
</mesh>
get_string(only=['msh'])

Iteration 3

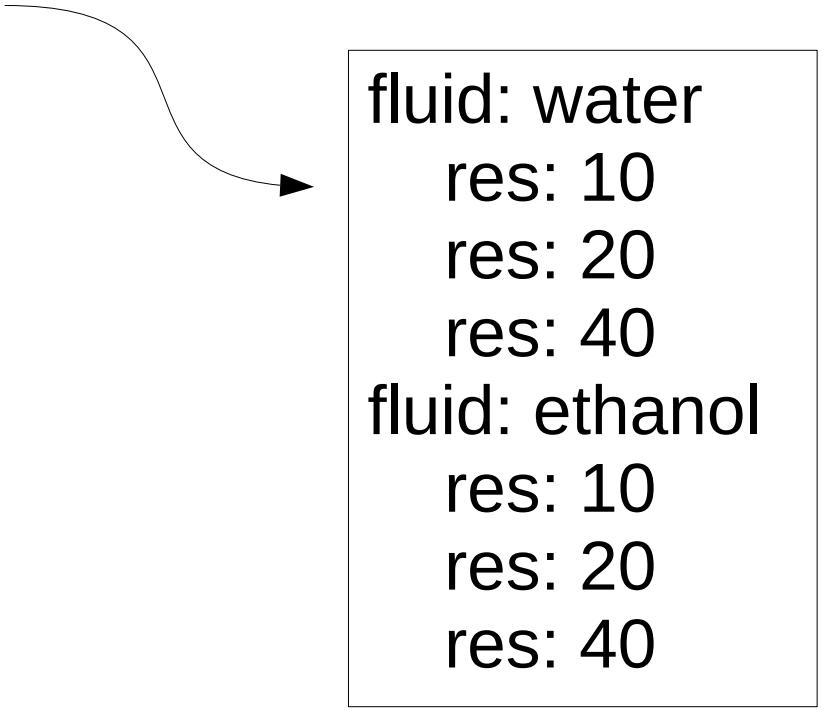
- Introduce composite; add “collapse” method



Operator overloading

- Multiplication

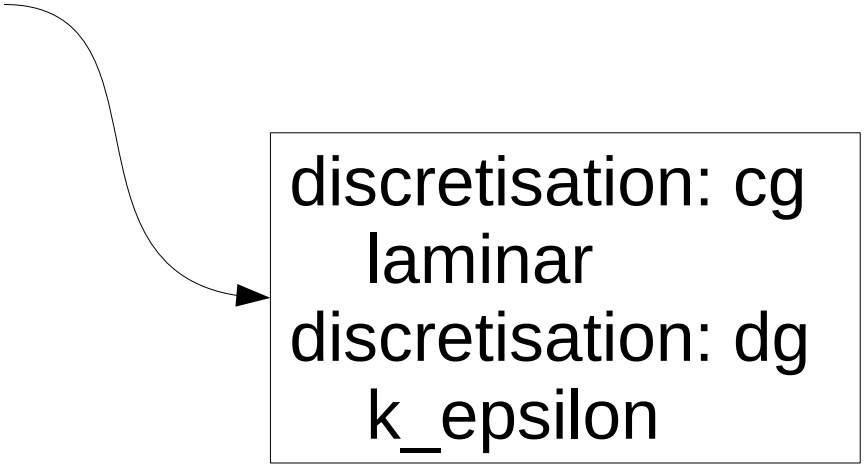
```
fluids = OptionsArray('fluid', [water, ethanol])
resolutions = OptionsArray('res', [10, 20, 40])
fluids * resolutions
```



Operator overloading

- Addition

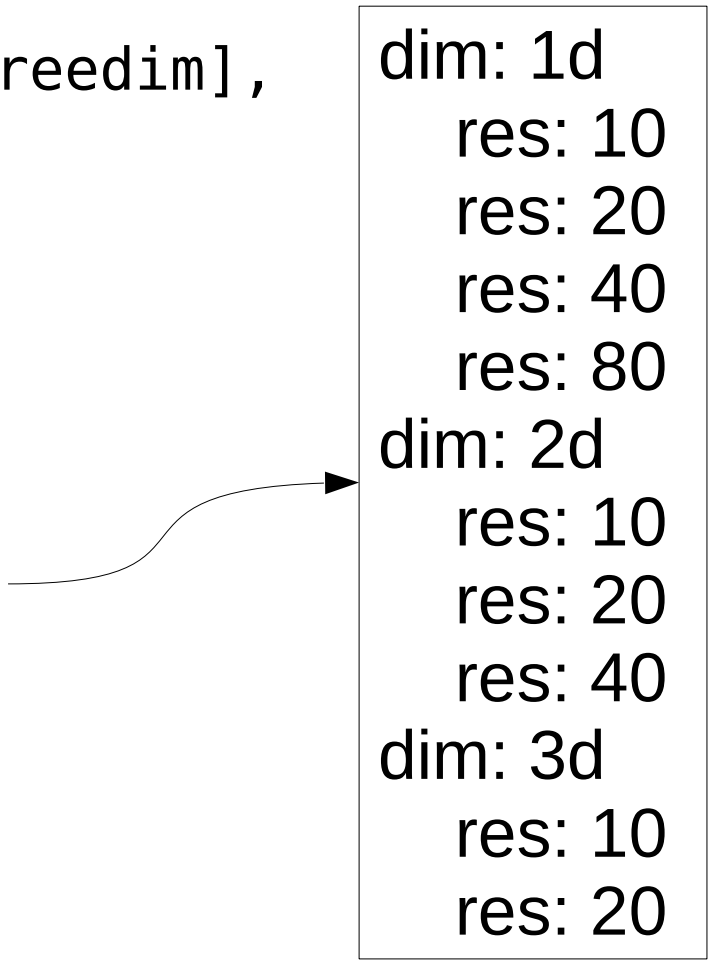
```
discs = OptionsArray('discretisation', [cg, dg])

turbmodels = OptionsArray('turbulence_model',
                          [laminar, k_epsilon])
discs + turbmodels
```



Operator overloading

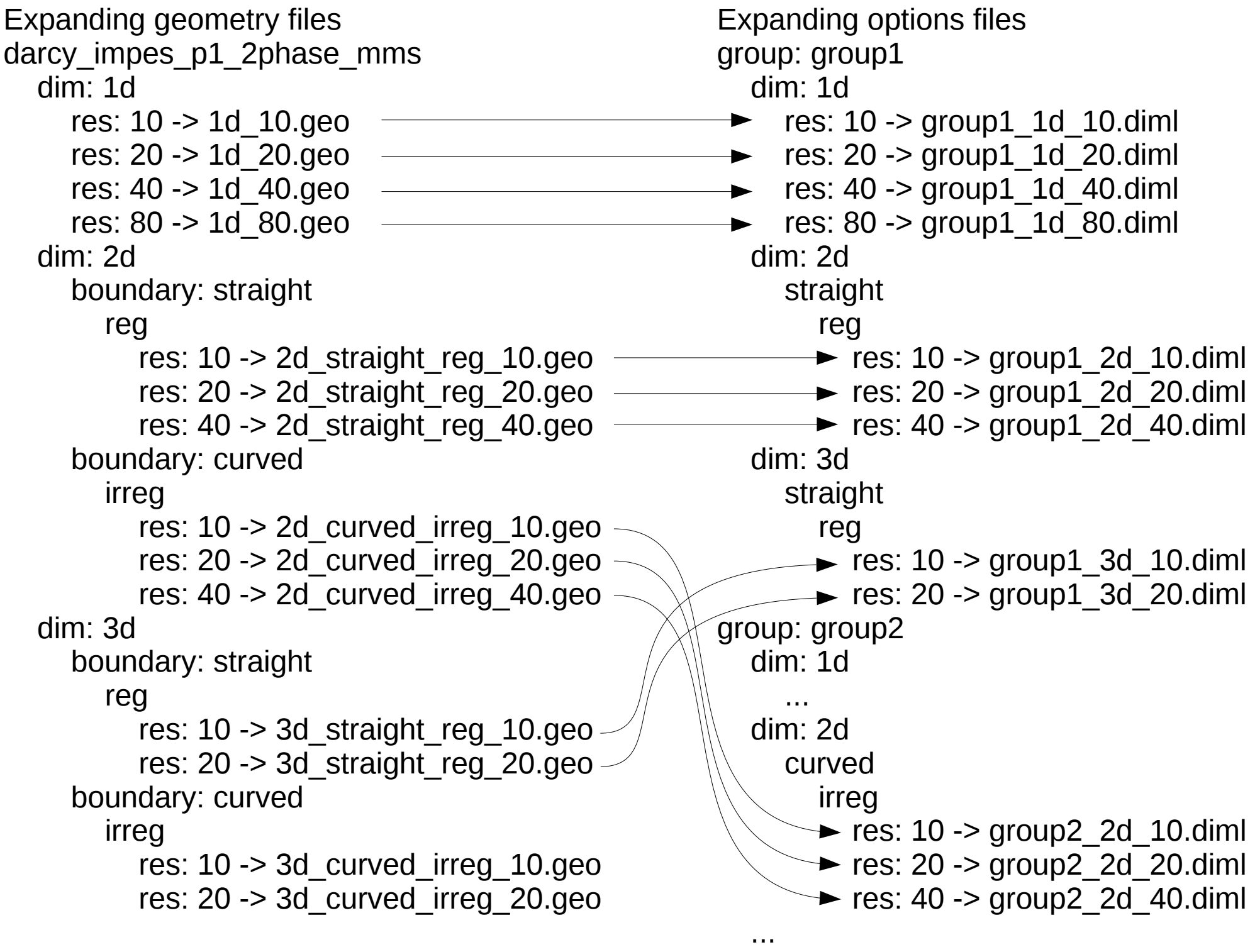
- Node setting/getting

```
dims = OptionsArray(
    'dim', [onedim, twodim, threedim],
    names=['1d', '2d', '3d'])
resolutions = OptionsArray(
    'res', [10, 20, 40, 80])

dims[0] *= resolutions[0:4]
dims[1] *= resolutions[0:3]
dims[2] *= resolutions[0:2]
```



```
Expanding geometry files
darcy_impes_p1_2phase_mms
    dim: 1d
        res: 10 -> 1d_10.geo
        res: 20 -> 1d_20.geo
        res: 40 -> 1d_40.geo
        res: 80 -> 1d_80.geo
    dim: 2d
        boundary: straight
            reg
                res: 10 -> 2d_straight_reg_10.geo
                res: 20 -> 2d_straight_reg_20.geo
                res: 40 -> 2d_straight_reg_40.geo
        boundary: curved
            irreg
                res: 10 -> 2d_curved_irreg_10.geo
                res: 20 -> 2d_curved_irreg_20.geo
                res: 40 -> 2d_curved_irreg_40.geo
    dim: 3d
        boundary: straight
            reg
                res: 10 -> 3d_straight_reg_10.geo
                res: 20 -> 3d_straight_reg_20.geo
        boundary: curved
            irreg
                res: 10 -> 3d_curved_irreg_10.geo
                res: 20 -> 3d_curved_irreg_20.geo
```

```
Expanding options files
group: group1
    dim: 1d
        res: 10 -> group1_1d_10.diml
        res: 20 -> group1_1d_20.diml
        res: 40 -> group1_1d_40.diml
        res: 80 -> group1_1d_80.diml
    dim: 2d
        straight
            reg
                res: 10 -> group1_2d_10.diml
                res: 20 -> group1_2d_20.diml
                res: 40 -> group1_2d_40.diml
    dim: 3d
        straight
            reg
                res: 10 -> group1_3d_10.diml
                res: 20 -> group1_3d_20.diml
group: group2
    dim: 1d
        ...
    dim: 2d
        curved
            irreg
                res: 10 -> group2_2d_10.diml
                res: 20 -> group2_2d_20.diml
                res: 40 -> group2_2d_40.diml
    ...
```


Expanding geometry files
darcy_impes_p1_2phase_mms
dim: 1d
res: 10 -> 1d_10.geo
res: 20 -> 1d_20.geo
res: 40 -> 1d_40.geo
res: 80 -> 1d_80.geo
dim: 2d
boundary: straight
reg
res: 10 -> 2d_straight_reg_10.geo
res: 20 -> 2d_straight_reg_20.geo
res: 40 -> 2d_straight_reg_40.geo
boundary: curved
irreg
res: 10 -> 2d_curved_irreg_10.geo
res: 20 -> 2d_curved_irreg_20.geo
res: 40 -> 2d_curved_irreg_40.geo
dim: 3d
boundary: straight
reg
res: 10 -> 3d_straight_reg_10.geo
res: 20 -> 3d_straight_reg_20.geo
boundary: curved
irreg
res: 10 -> 3d_curved_irreg_10.geo
res: 20 -> 3d_curved_irreg_20.geo
Expanding options files
group: group1
dim: 1d
res: 10 -> group1_1d_10.diml
res: 20 -> group1_1d_20.diml
res: 40 -> group1_1d_40.diml
res: 80 -> group1_1d_80.diml
dim: 2d
straight
reg
res: 10 -> group1_2d_10.diml
res: 20 -> group1_2d_20.diml
res: 40 -> group1_2d_40.diml
dim: 3d
straight
reg
res: 10 -> group1_3d_10.diml
res: 20 -> group1_3d_20.diml
group: group2
dim: 1d
...
dim: 2d
curved
irreg
res: 10 -> group2_2d_10.diml
res: 20 -> group2_2d_20.diml
res: 40 -> group2_2d_40.diml
...

Dictionary Features

```
class fluid:
    def kinematic_viscosity(self):
        return self.dynamic_viscosity / self.density
```

```
class water(fluid):
    density = 1.00e3
    dynamic_viscosity = 0.89e-3
```

```
class ethanol(fluid):
    density = 0.79e3
    dynamic_viscosity = 1.09e-3
```

Dictionary Features

- Class-based initialisation
- Cleaner attribute-style (dot) syntax
- Dependent items (but need to wrap functions)

```
class simulation_options:
    reference_timestep_number = 10
    preconditioner = 'mg'
    adaptive_timestepping = False

    def time_step(self):
        "Maintain a constant Courant number"
        scale_factor = float(self.reference_mesh_res) /
            self.mesh_res
        return scale_factor * self.finish_time /
            self.reference_timestep_number
```

Content Transformations

- Unfortunately, when doing this:

```
p = multiprocessing.Pool(nprocs)
p.map(function, iterable)
```

This happens:

cPickle.PicklingError: Can't pickle <type 'function'>: attribute lookup __builtin__.function failed

because Python's serialization module chokes on localised functions.

Content Transformations

- Use `transform_items(`**`function`**`, recursive=True)`

  where **`function`** can be:

  - `unlink`
  - `Check(`**`test`**`)`
  - `Remove(`**`test`**`)`

  and **`test`** can be:

  - `missing_dependencies`
  - `unpicklable`
- Can add these as hooks to tree elements, or in calls to front-end functions

The Jinja2 Template Engine

```
  <physical_parameters>
    {% if gravity_magnitude %}
    <gravity>
      <magnitude>
        <real_value rank="0">{{ gravity_magnitude }}</real_value>
      </magnitude>
      <vector_field name="GravityDirection" rank="1">
        <prescribed>
          <mesh name="ElementWiseMesh"/>
          <value name="WholeMesh">
            <constant>
              <real_value shape="{{ dim_number }}" dim1="dim"
rank="1">{{ gravity_direction|join(' ') }}</real_value>
            </constant>
          </value>
        </prescribed>
      </vector_field>
    </gravity>
    {% endif %}
    ....
```

The Jinja2 Template Engine

- “For” loops - iterate over subcomponents
  - fields of interest
  - variables in diagnostic algorithm blocks
  - assert statements
- Macros
- Template inheritance
- Custom filters

Example: MMS-based tests

```
class darcy_impes_2phase_mms:
   domain_extents = (1.0, 1.2, 0.8)
   finish_time = 1.0

   def saturations(self):
      return (1 - self.saturation2,
                self.saturation2)

   def saturation_sources(self):
      results = []
      t = Symbol('t')
      # loop over phases
      for i in range(2):
         phi = self.porosity
         S = self.saturations[i]
         u = self.darcy_velocities[i]
         results.append(
            diff(phi*S, t) + div(u))
      return results
```

```
class three_dim:
```

```
class two_dim:
   geometry = "rectangle"
```

```
class one_dim:
   geometry = "line"
   wall_ids = ()

   def saturation2(self):
      """Invented saturation
         profile"""
      x = Symbol('x')
      t = Symbol('t')
      L = self.domain_extents[0]
      T = self.finish_time
      return exp(-x/L)/(1 + t/T)

   def pressure1(self):
      ...
```

Example: MMS-based tests

```
class darcy_impes_2phase_mms:
   domain_extents = (1.0, 1.2, 0.8)
   finish_time = 1.0

   def saturations(self):
      return (1 - self.saturation2,
                self.saturation2)

   def saturation_sources(self):
      results = []
      t = Symbol('t')
      # loop over phases
      for i in range(2):
         phi = self.porosity
         S = self.saturations[i]
         u = self.darcy_velocities[i]
         results.append(
            diff(phi*S, t) + div(u))
      return results
```

```
class three_dim:
```

```
class two_dim:
   geometry = "rectangle"
```

```
class one_dim:
   geometry = "line"
   wall_ids = ()

   def saturation2(self):
      """Invented saturation
         profile"""
      x = Symbol('x')
      t = Symbol('t')
      L = self.domain_extents[0]
      T = self.finish_time
      return exp(-x/L)/(1 + t/T)

   def pressure1(self):
      ...
```

```
{% extends "darcy_impes_base.diml.template" %}

{% block saturation1_source %}
<scalar_field name="Source" rank="0">
  <prescribed>
    <mesh name="SaturationSourceMesh"/>
    <value name="WholeMesh">
      <python>
        <string_value type="code" lines="20" language="python">
def val(X, t):
  from numpy import sqrt, pi, cos, sin, exp
  return {{ saturation_sources[0]|format_sympy }}
</string_value>
      </python>
    </value>
    <stat>
      <include_cv_stats/>
    </stat>
  </prescribed>
</scalar_field>
{% endblock %}

{% block saturation2_source %}
...
```

darcy_impes_p1_2phase_mms.diml.template

Example Application

- examples/top_hat
  - base.py
  - postproc.py
  - runner.py
  - runner.sh
  - templates/
    - base.flml.template
    - cg.flml.template
    - cv.flml.template
    - dg.flml.template
    - line.geo.template

Example Application

- examples/top_hat
  - base.py
  - postproc.py
  - runner.py
  - runner.sh
  - templates/
    - base.flml.template
    - cg.flml.template
    - cv.flml.template
    - dg.flml.template
    - line.geo.template

Classes representing global options

A plotting functor

Front-end

Initial simulation options

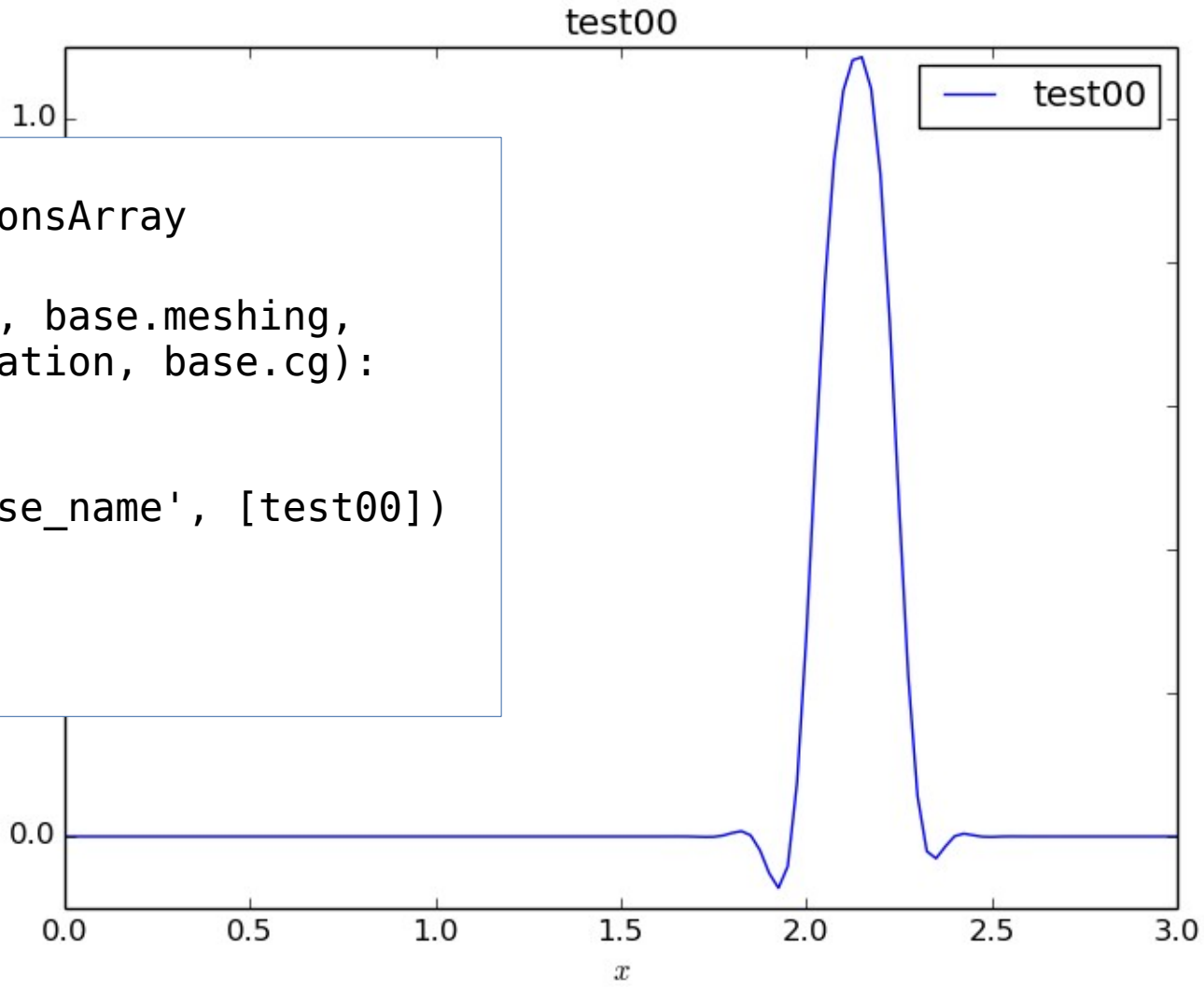
Extended sim. options

Geometry template

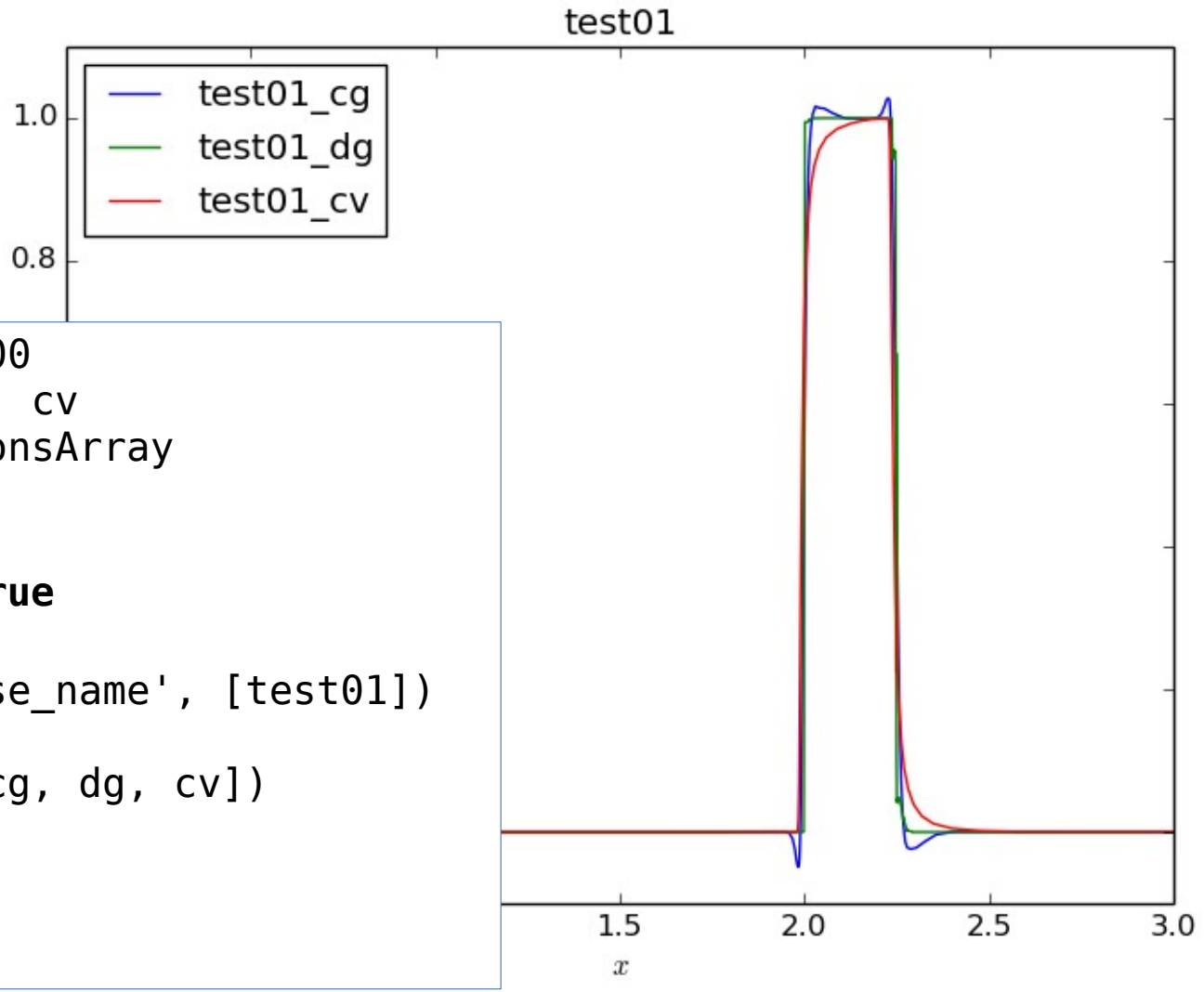
```
class Plot(SerialFunctor):
    def preamble(self, options):
        plt.clf()

    def __call__(self, options):
        stem = '{}/{}'.format(options.case_name, options.simulation_name)
        # get the last dump file
        n = 0
        while os.path.isfile(get_filename(options, n + 1)):
            n += 1
        # load results and plot
        vtu_obj = vtktools.vtu(get_filename(options, n))
        x = vtu_obj.GetLocations()[:,0]
        f = vtu_obj.GetScalarField('Tracer')
        x, f = monotonic(x, f)
        plt.plot(x, f, label=options.simulation_name)
        self.print_end(options.simulation_name, options)

    def postamble(self, options):
        "This gets called after iteration"
        plt.axis([0.0, 3.0, -0.1, 1.1])
        plt.xlabel('$x$')
        plt.ylabel('Tracer')
        plt.title(options.case_name)
        plt.legend(loc='best')
        if show_plots:
            plt.show()
        if save_plots:
            plt.savefig('{}.png'.format(options.case_name))
```

test00

```
import base
from opiter import OptionsArray

class test00(base.admin, base.meshing,
             base.simulation, base.cg):
    pass

root = OptionsArray('case_name', [test00])

mesh_tree = root
sim_tree = root
```

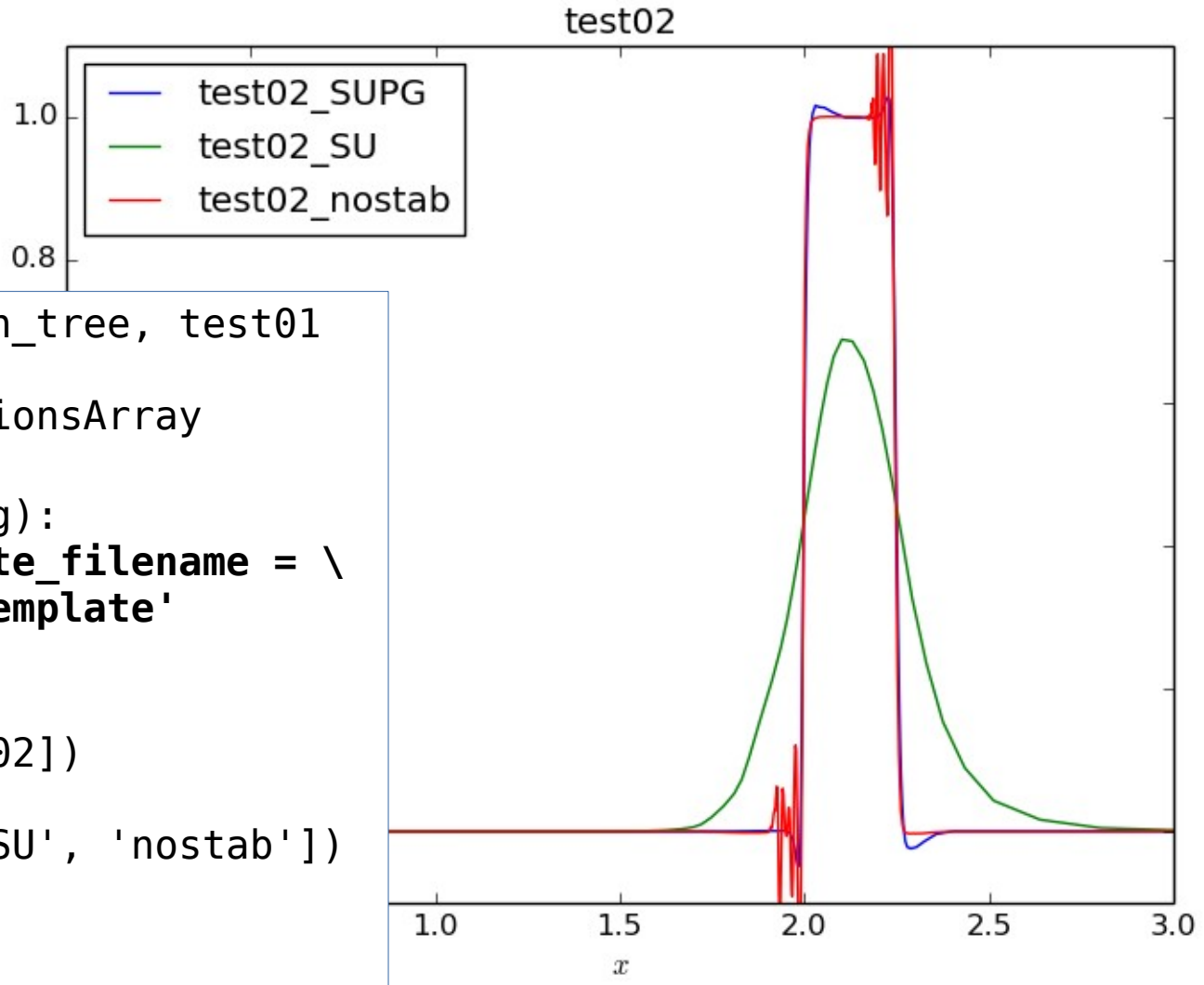


test01



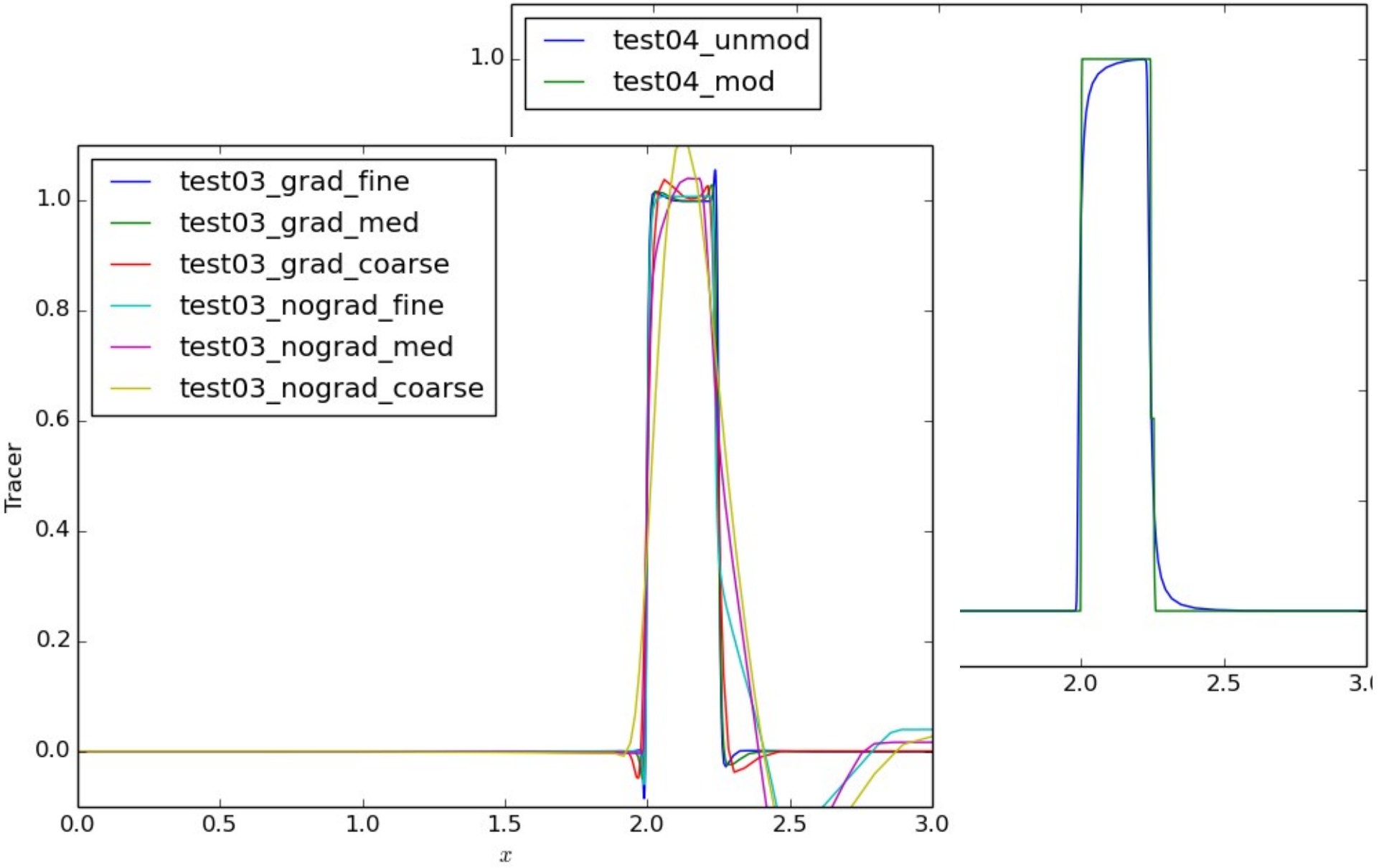

```
from test00 import test00
from base import cg, dg, cv
from opiter import OptionsArray

class test01(test00):
    mesh_adaptivity = True

root = OptionsArray('case_name', [test01])
discr = OptionsArray(
    'discretisation', [cg, dg, cv])

mesh_tree = root
sim_tree = root * discr
```

test02



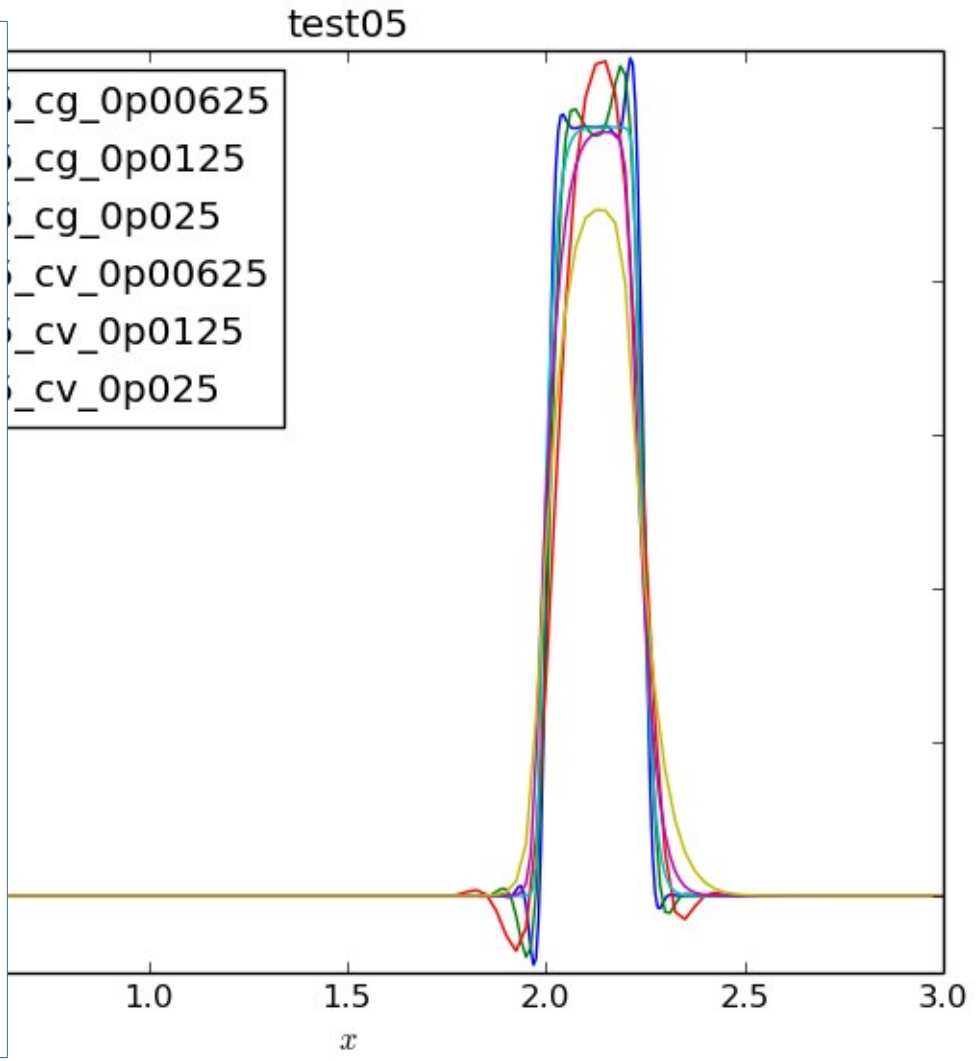

```
from test01 import mesh_tree, test01
from base import cg
from opiter import OptionsArray

class test02(test01, cg):
    sim_options_template_filename = \
        'test02.flml.template'

root = OptionsArray(
    'case_name', [test02])
stab = OptionsArray(
    'stab', ['SUPG', 'SU', 'nostab'])

mesh_tree = root
sim_tree = root * stab
```

etc.


test04_unmod
test04_mod
test03_grad_fine
test03_grad_med
test03_grad_coarse
test03_nograd_fine
test03_nograd_med
test03_nograd_coarse
Tracer
x
0.0
0.5
1.0
1.5
2.0
2.5
3.0
0.2
0.4
0.6
0.8

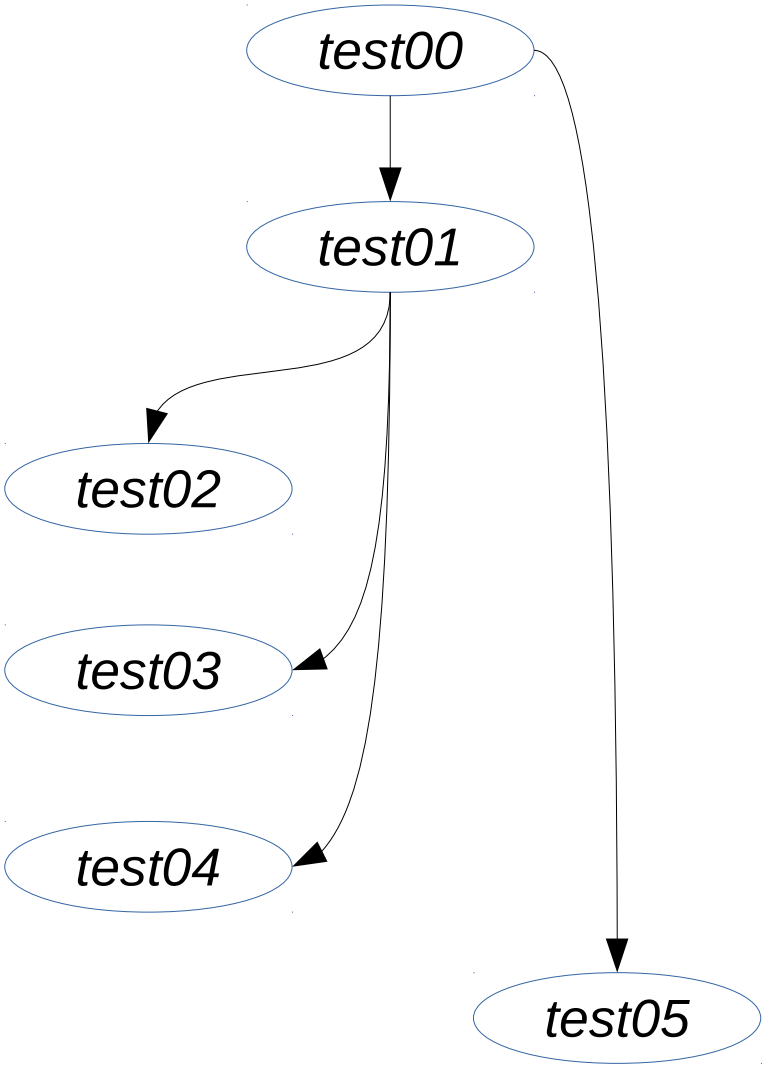
varying both mesh and simulation

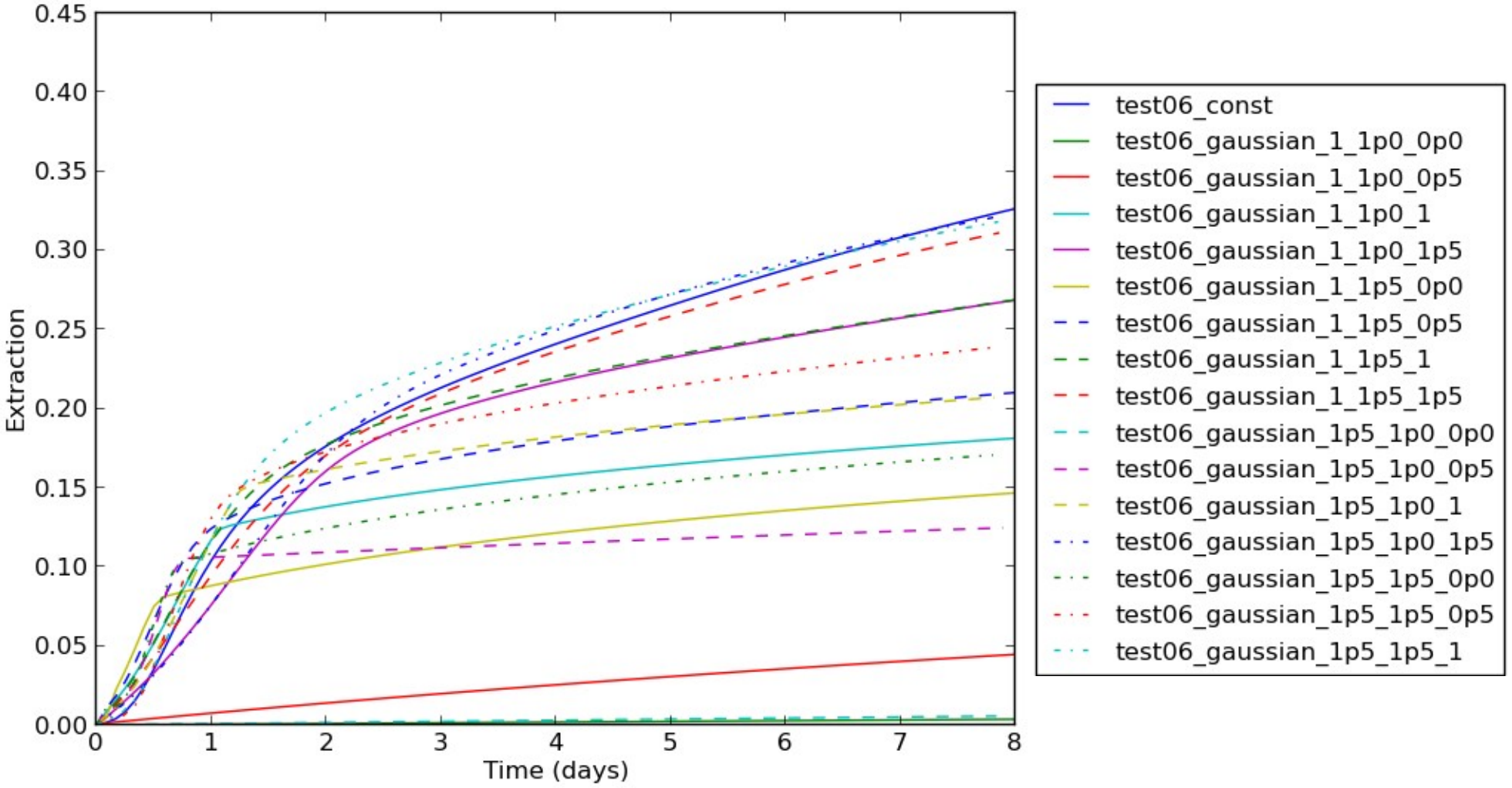
```
from test00 import test00
from base import cg, dg, cv
from opiter import OptionsArray

class test05(test00):
    pass

root = OptionsArray(
    'case_name', [test05])
discr = OptionsArray(
    'discretisation', [cg, cv])

dx = OptionsArray(
    'dx', [0.00625, 0.0125, 0.025],
    name_format=lambda dx: \
        str(dx).replace('.', 'p'),
    tags=['mesh'])

mesh_tree = root * dx
sim_tree = root * discr * dx
```

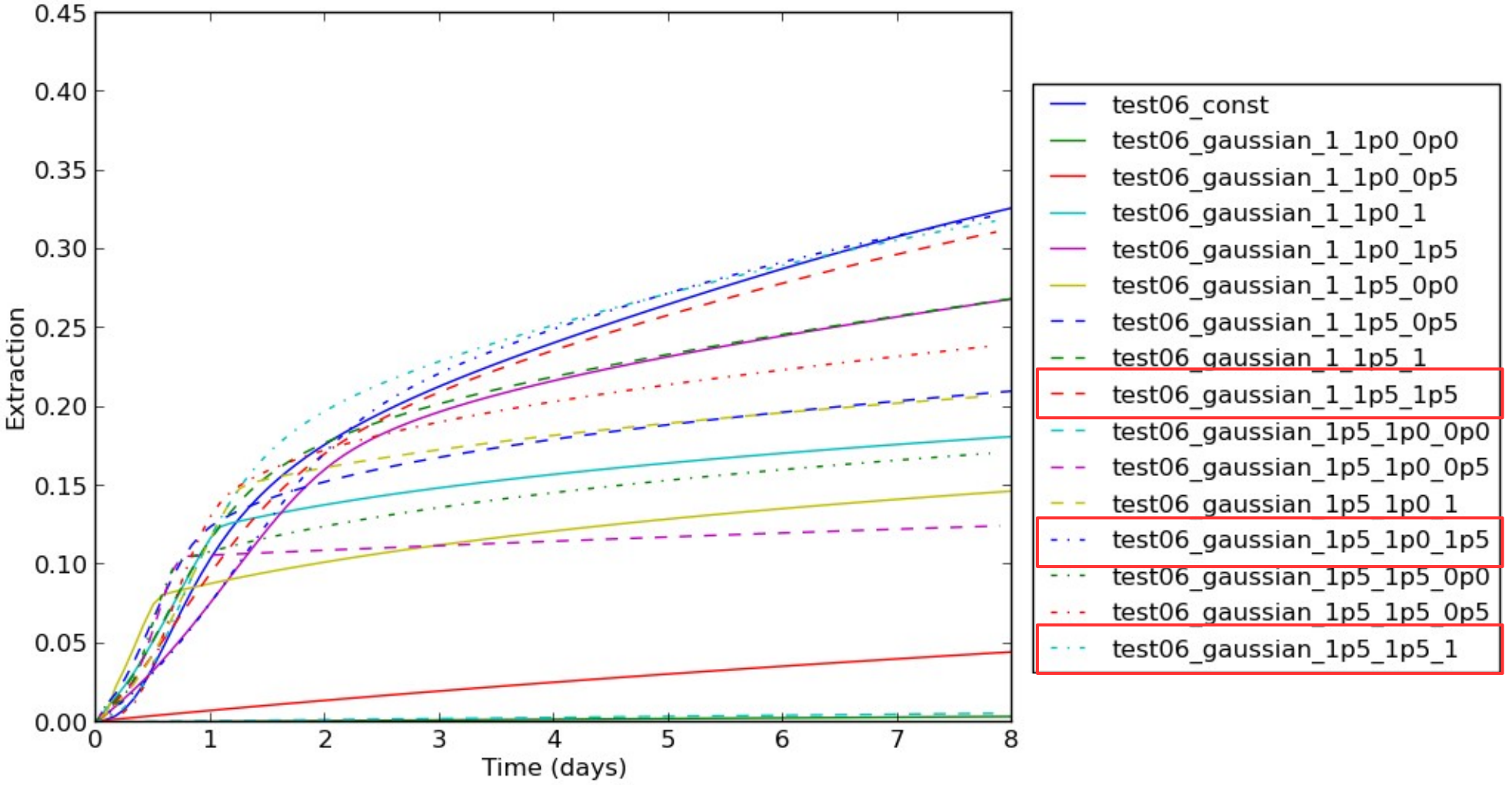


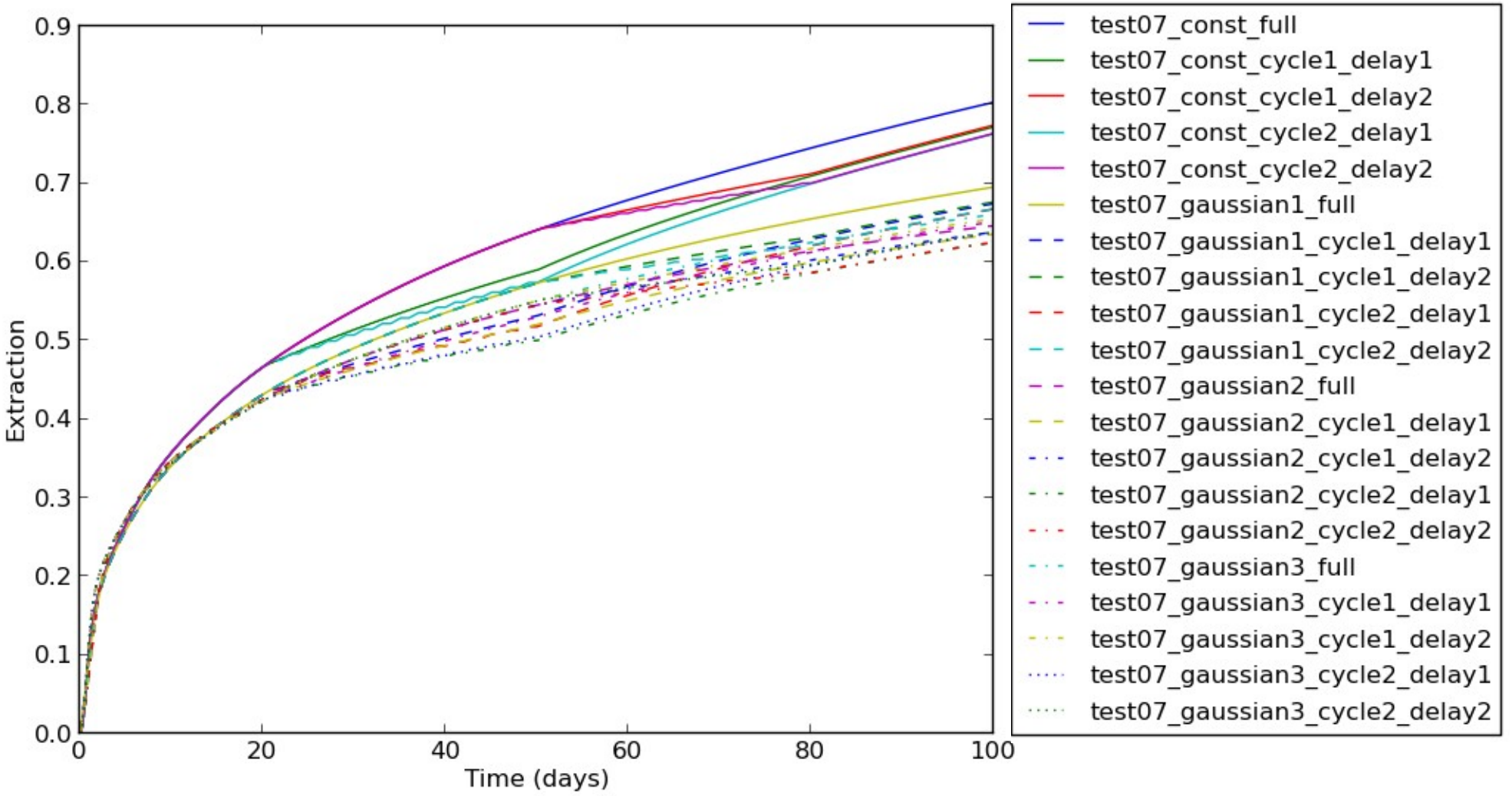
The General Idea

1. Set up basic options, templates and any custom functors
2. Write a test file whose options may inherit from tests gone before
3. Run, plot, analyse
4. Decide what to vary next; insert blocks into the base template if necessary
5. Repeat steps (2)–(4)




0.45
0.40
0.35
0.30
0.25
0.20
0.15
0.10
0.05
0.00
Extraction
0
1
2
3
4
5
6
7
8
Time (days)
test06_const
test06_gaussian_1_1p0_0p0
test06_gaussian_1_1p0_0p5
test06_gaussian_1_1p0_1
test06_gaussian_1_1p0_1p5
test06_gaussian_1_1p5_0p0
test06_gaussian_1_1p5_0p5
test06_gaussian_1_1p5_1
test06_gaussian_1_1p5_1p5
test06_gaussian_1p5_1p0_0p0
test06_gaussian_1p5_1p0_0p5
test06_gaussian_1p5_1p0_1
test06_gaussian_1p5_1p0_1p5
test06_gaussian_1p5_1p5_0p0
test06_gaussian_1p5_1p5_0p5
test06_gaussian_1p5_1p5_1


0.45
0.40
0.35
0.30
0.25
0.20
0.15
0.10
0.05
0.00
Extraction
0
1
2
3
4
5
6
7
8
Time (days)
test06_const
test06_gaussian_1_1p0_0p0
test06_gaussian_1_1p0_0p5
test06_gaussian_1_1p0_1
test06_gaussian_1_1p0_1p5
test06_gaussian_1_1p5_0p0
test06_gaussian_1_1p5_0p5
test06_gaussian_1_1p5_1
test06_gaussian_1_1p5_1p5
test06_gaussian_1p5_1p0_0p0
test06_gaussian_1p5_1p0_0p5
test06_gaussian_1p5_1p0_1
test06_gaussian_1p5_1p0_1p5
test06_gaussian_1p5_1p5_0p0
test06_gaussian_1p5_1p5_0p5
test06_gaussian_1p5_1p5_1


Extraction
Time (days)
0.0
0.1
0.2
0.3
0.4
0.5
0.6
0.7
0.8
0.9
0
20
40
60
80
100
test07_const_full
test07_const_cycle1_delay1
test07_const_cycle1_delay2
test07_const_cycle2_delay1
test07_const_cycle2_delay2
test07_gaussian1_full
test07_gaussian1_cycle1_delay1
test07_gaussian1_cycle1_delay2
test07_gaussian1_cycle2_delay1
test07_gaussian1_cycle2_delay2
test07_gaussian2_full
test07_gaussian2_cycle1_delay1
test07_gaussian2_cycle1_delay2
test07_gaussian2_cycle2_delay1
test07_gaussian2_cycle2_delay2
test07_gaussian3_full
test07_gaussian3_cycle1_delay1
test07_gaussian3_cycle1_delay2
test07_gaussian3_cycle2_delay1
test07_gaussian3_cycle2_delay2

Conclusion

- Package name: **opiter** (“Options Iteration”)
- Licence: Lesser GPL v2.1
- Getting it:

  git clone https://github.com/rjferrier/opiter

  cd opiter

  sudo python setup.cfg install
- Companion repository: opiter-fluidity
- Queries, feedback, ideas welcome

Disclaimers

- Does not protect against user errors
- Defers to low-level exception handling

Further work

- More complete documentation
- More examples
- Higher-level exception handling
- Code review and clean up

Acknowledgements

- Stephen Neethling
- Liping Cai