PorePy: A Python Simulation Tool for Fractured and Deformable Porous Media

Eirik Keilegavlen, Alessio Fumagalli, Runar Berge, Ivar Stefansson

Summary

- Meshing and discretization of dynamics in fractured rocks
- Built on discrete fracture-matrix (DFM) approach
- Automatic meshing of 2D and 3D fracture geometries
- Discretization schemes
    - Diffusion equation: TPFA, MPFA, mixed VEM
    - Elasticity and fracture deformation: MPSA
- Visualization with Paraview

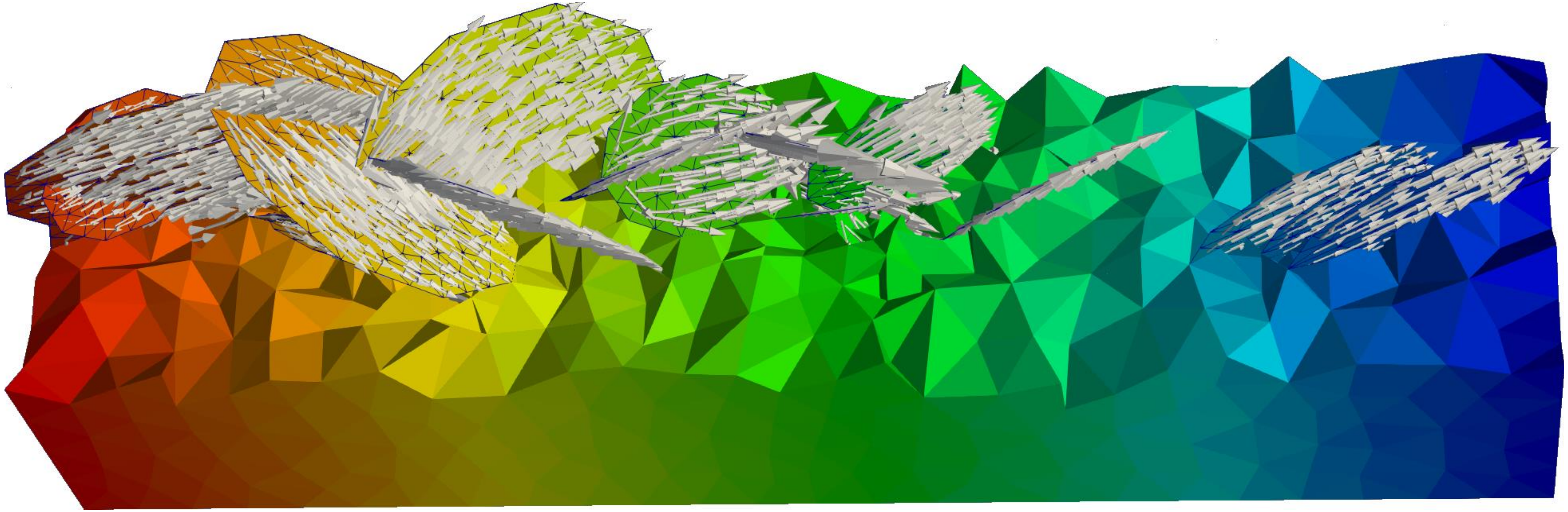
Motivation

Geothermal energy storage and extraction

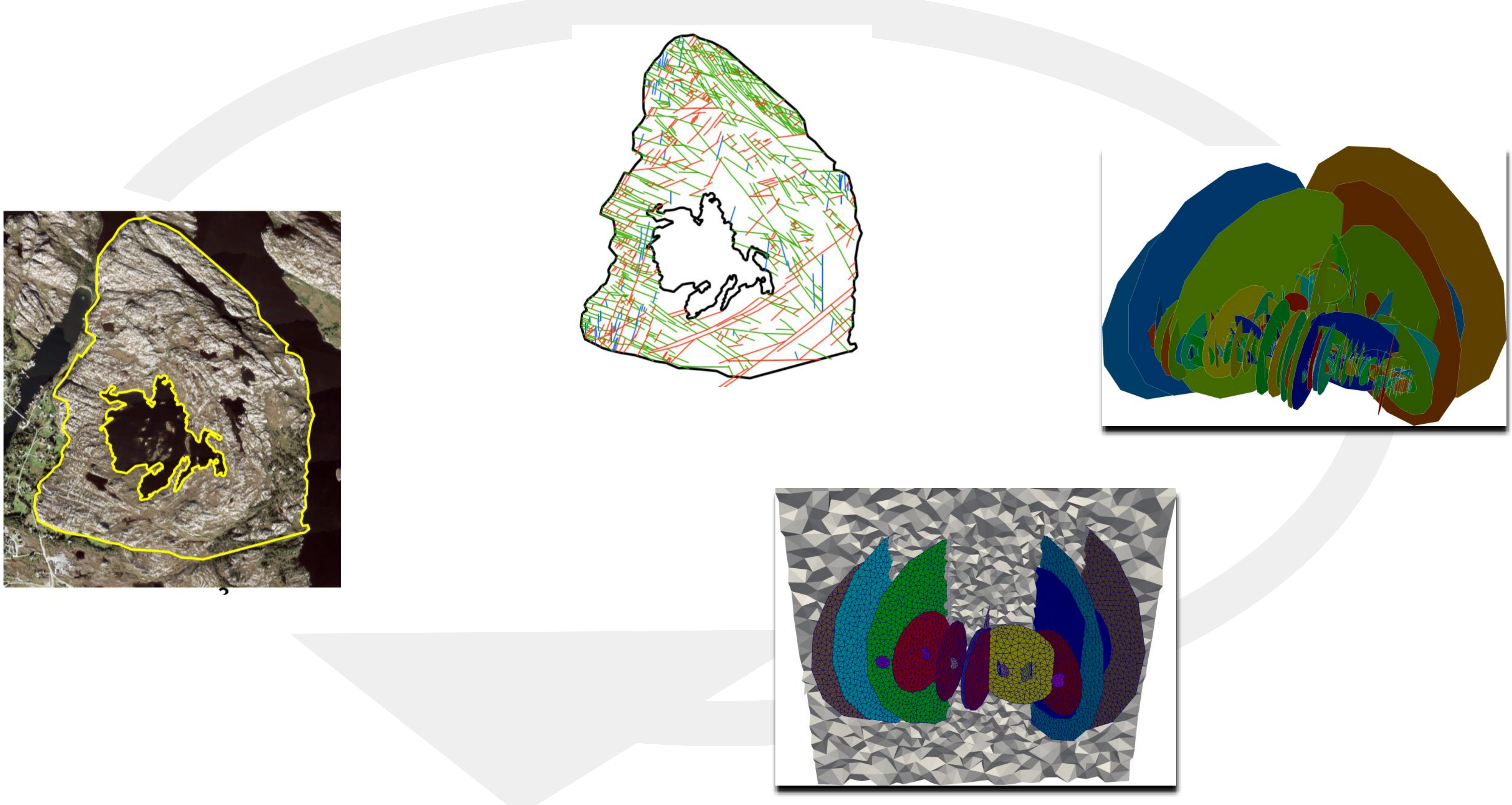
Simulation needs (dictated by ongoing projects):

- Flow and transport in fractured rocks
- Collaboration with geologists – (quasi)-realistic fracture networks
- Sliding of existing fractures – elasticity and fracture deformation

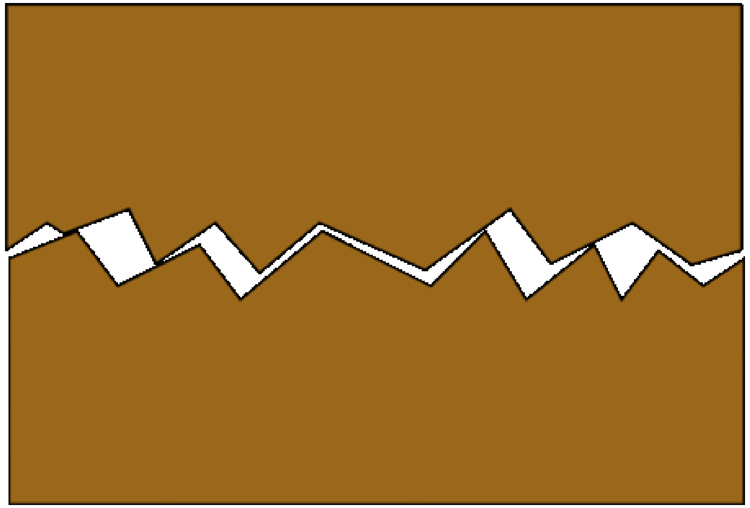
Coupled flow in matrix and fractures



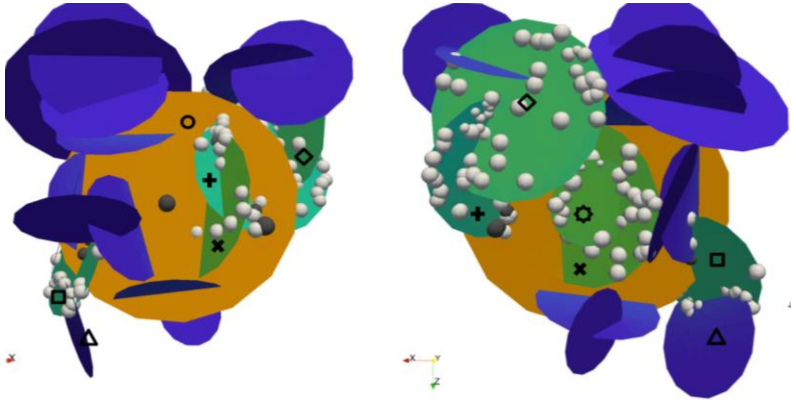
Collaboration with geologists



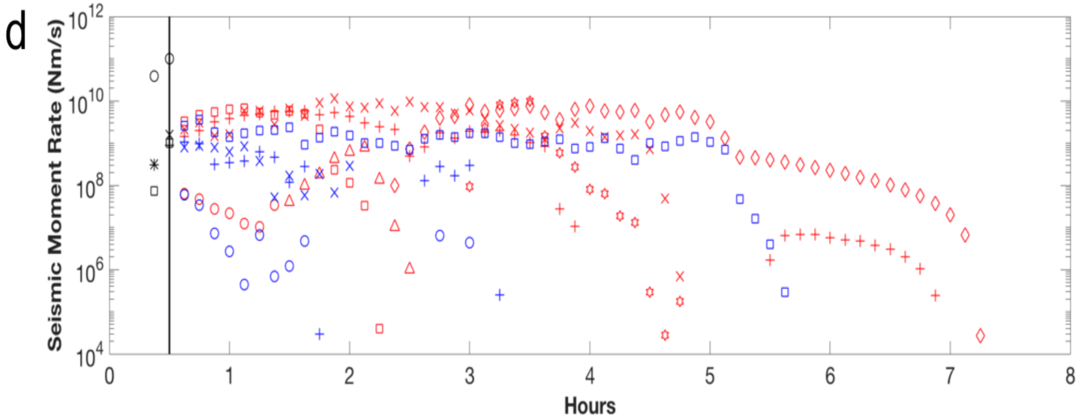
Permeability increase by hydroshearing



- Induced sliding along existing fractures.
- Results in permeability increase.
- Interaction between fluid flow and rock mechanics.



Permeability increases



Estimated induced seismicity

Typical research tasks

- Develop and test discretization schemes
- Numerics for multi-physics couplings
- Identify dominant physical processes

Conceptual numerical model

Coupling between dimensions

Flux law:

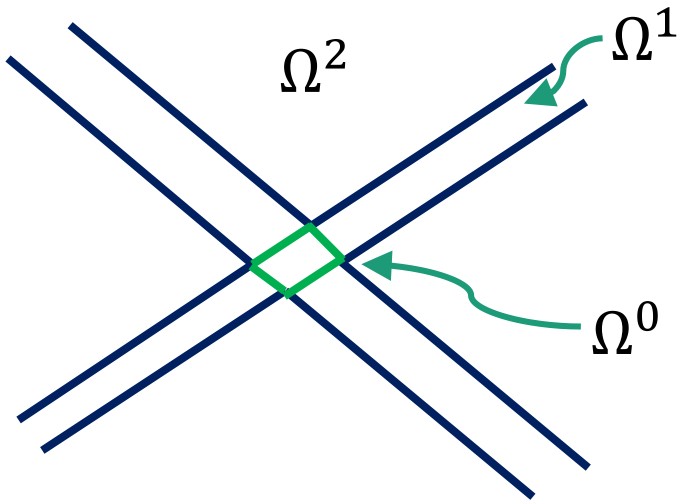
$$\boldsymbol{q}^d = -\boldsymbol{K}_T \nabla p^d, \qquad d = 1, 2, 3$$

Conservation:

$$\nabla \cdot \boldsymbol{q}^d = f^d, \qquad d = 1, 2, 3$$

External boundary conditions:

$$p = p^D \; on \; \partial\Omega_D^d$$
$$\boldsymbol{q} \cdot \boldsymbol{n} = q_N \; on \; \partial\Omega_N^d$$



Coupling between dimensions

Flux law:

$$\boldsymbol{q}^d = -\boldsymbol{K}_T \nabla p^d, \quad d = 1,2,3$$

Conservation:

$$\nabla \cdot \boldsymbol{q}^3 = f^3$$

$$\nabla \cdot \boldsymbol{q}^d = f^d + [\![\lambda^{d+1,d}]\!], d = 0,1,2$$

External boundary conditions:

$$p = p^D \; on \; \partial\Omega_D^d$$

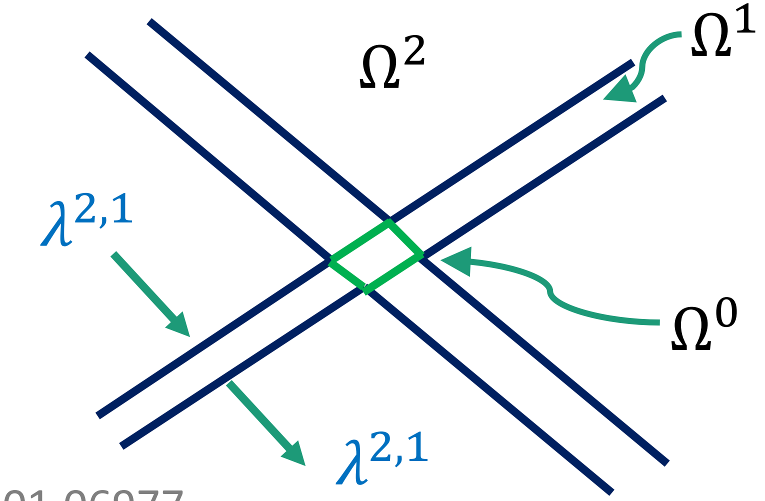
$$\boldsymbol{q} \cdot \boldsymbol{n} = q_N \; on \; \partial\Omega_N^d$$

Interface law:

$$\lambda^{d,d-1} = K_n\left(p^d - p^{d-1}\right)$$

Boundary condition on interface:

$$\boldsymbol{q}^d \cdot \boldsymbol{n} = \lambda^{d,d-1}$$



Boon, Nordbotten, Yotov: Robust discretization of flow in porous media. Arxiv: 1601.06977

Nordbotten, Boon: Modeling, structure and discretization of mixed-dimensional PDEs, Arxiv: 1705.06876

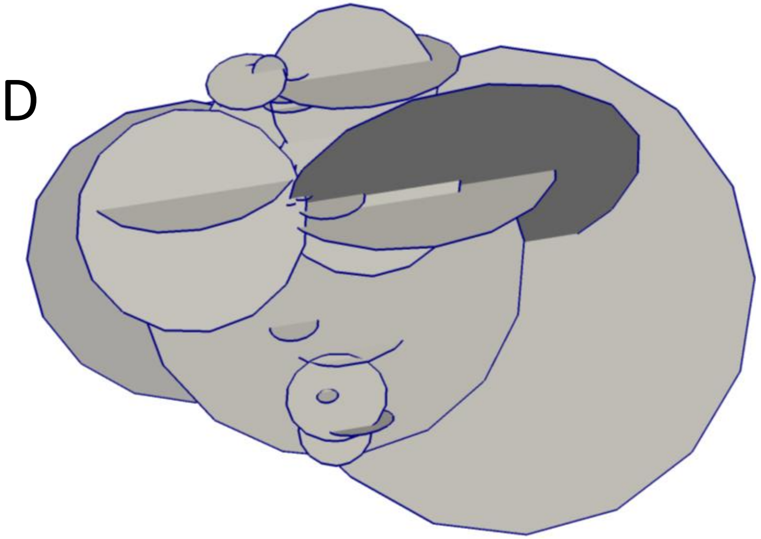
Implementation needs

1. Mixed-dimensional mesh
2. Discretization within each dimension
3. Coupling conditions

Meshing in fractured domains

Meshing of fractured domains

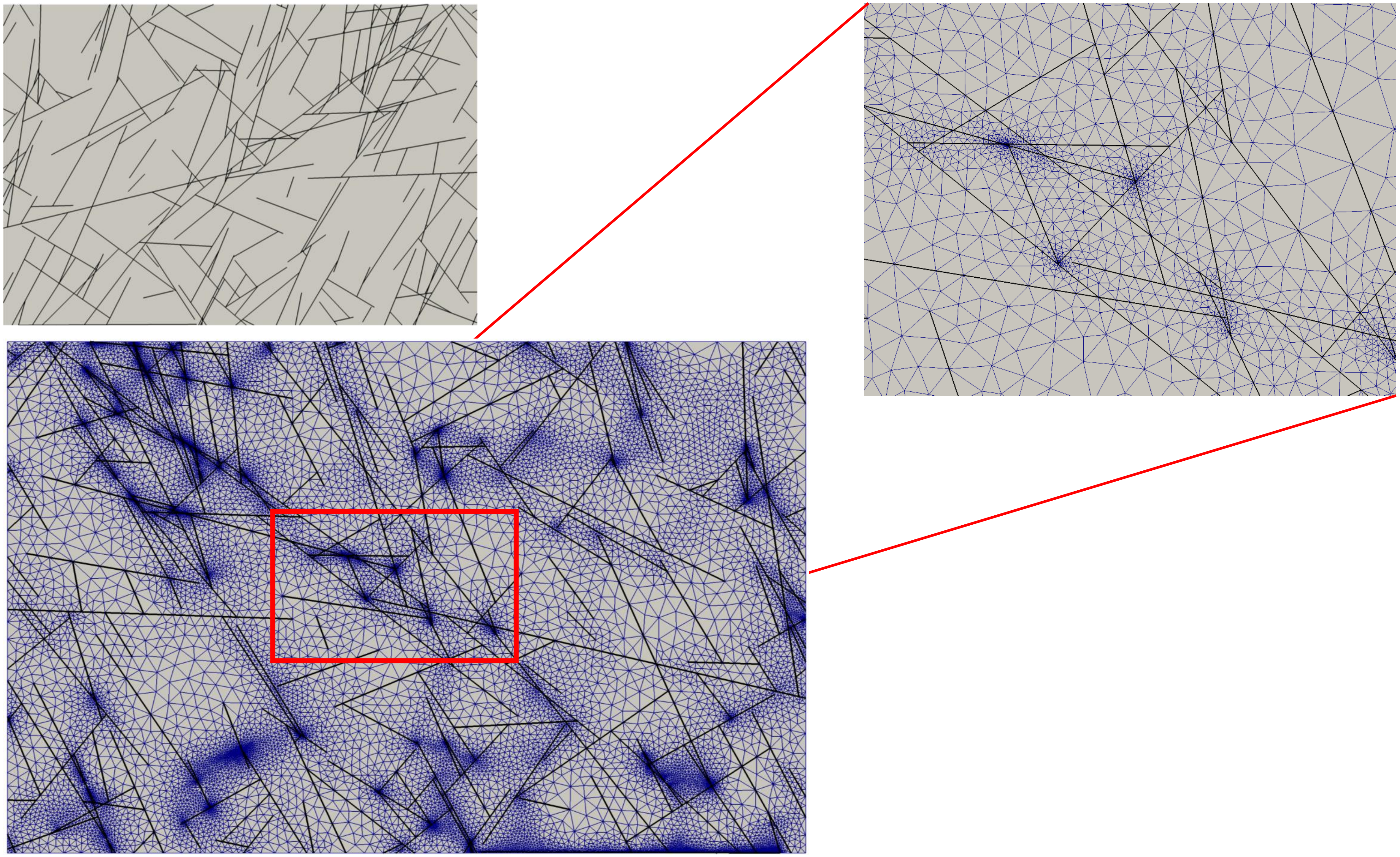
- Fractures represented as constraints for meshing algorithm
- Complex fracture networks: Mesh size dictated by geometry, rather than accuracy needs
- (Non-commercial) meshing software tends to require non-intersecting constraints
  - Preprocessing required – computational geometry
  - Well established in 2D, considerably more difficult in 3D

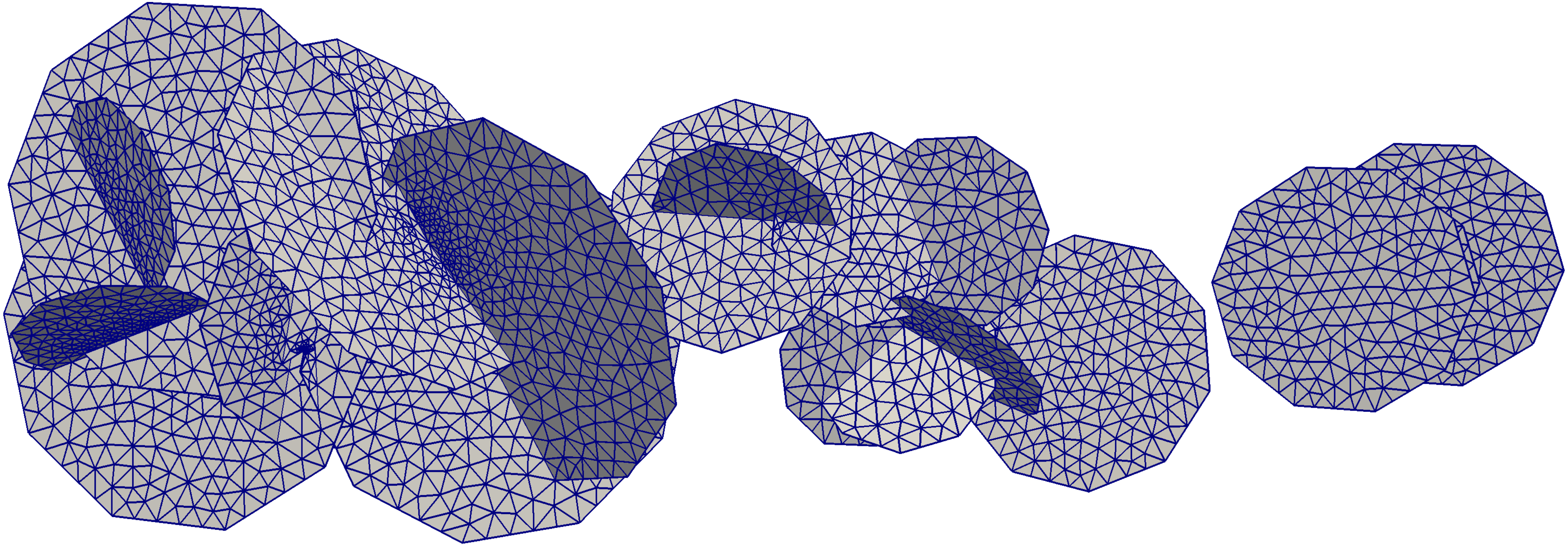


Meshing of fractured domains in PorePy

- Automatic handling of intersections
  - Quite stable in 2D
  - Workable, but far from perfect, in 3D
- Actual meshing by Gmsh
- Automatic mesh size tuning based on fracture geometry and (two) user parameters
  - 2D: Quite mature
  - 3D: Is improving
- No silver bullets: Small angles and close objects give bad meshes

Geuzaine, C. and Remacle, J.-F: Gmsh: A 3-D finite element mesh generator with built-in pre- and post-processing facilities. IJNME, 2009



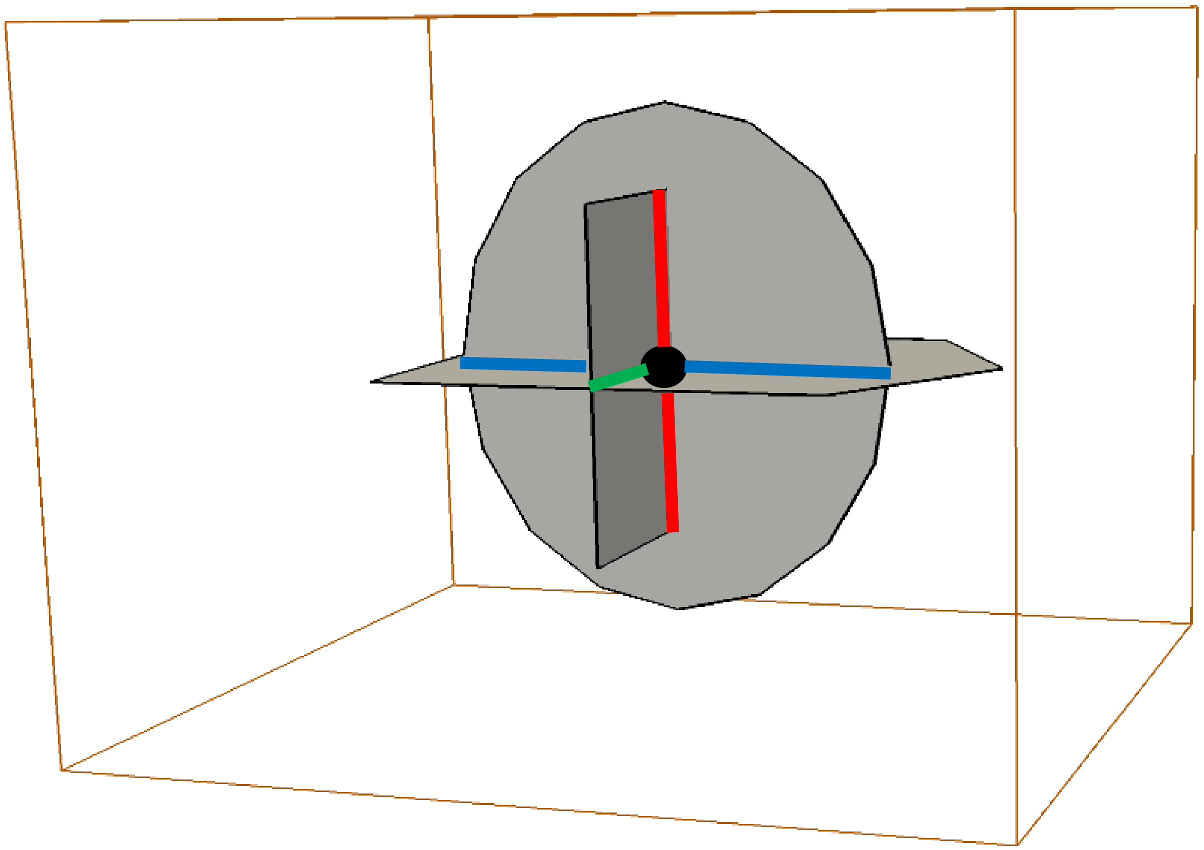


3d mesh left out for clarity

Mixed-dimensional meshes

Data structures and implementation

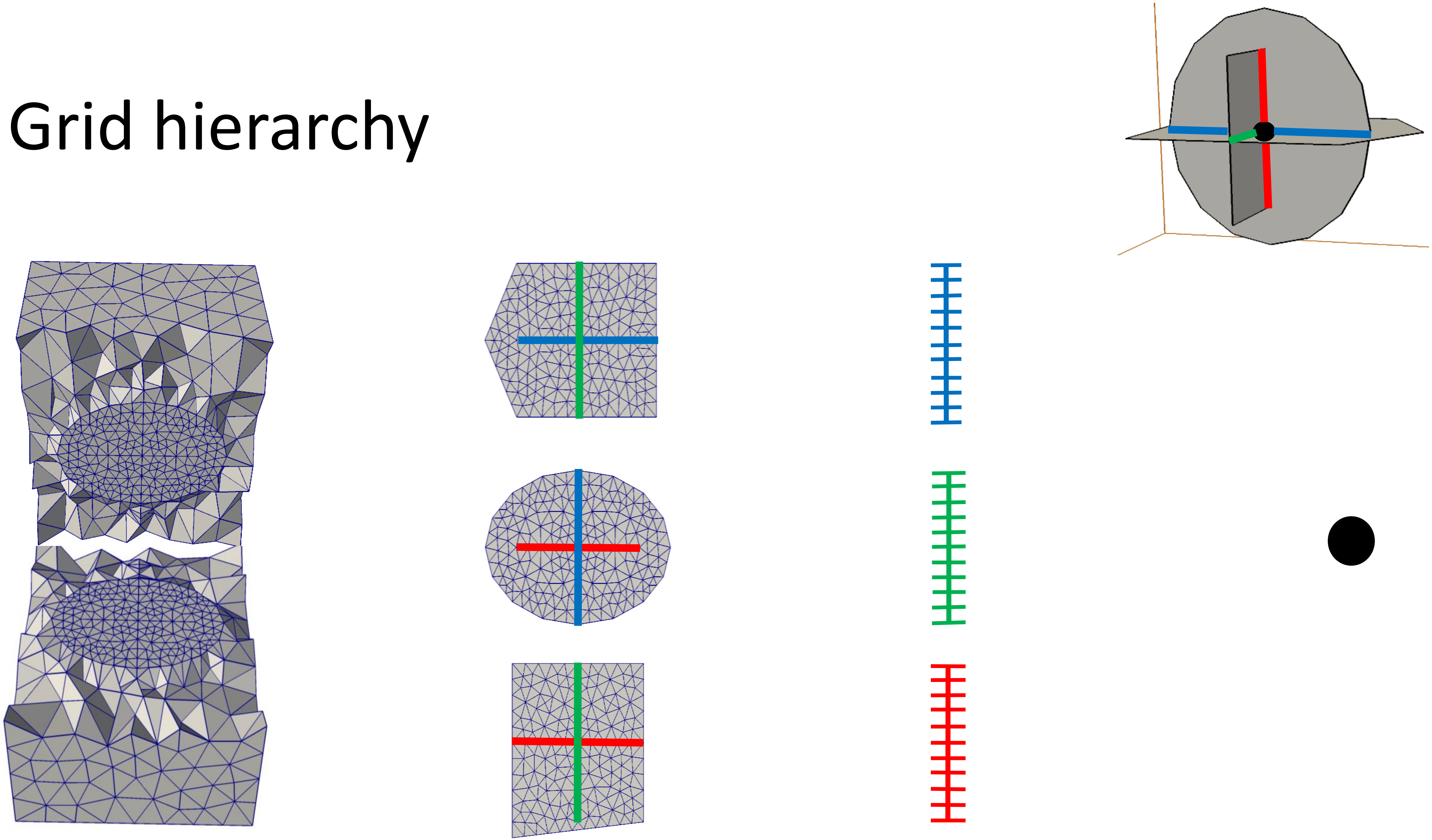
Mixed-dimensional grid hierarchy



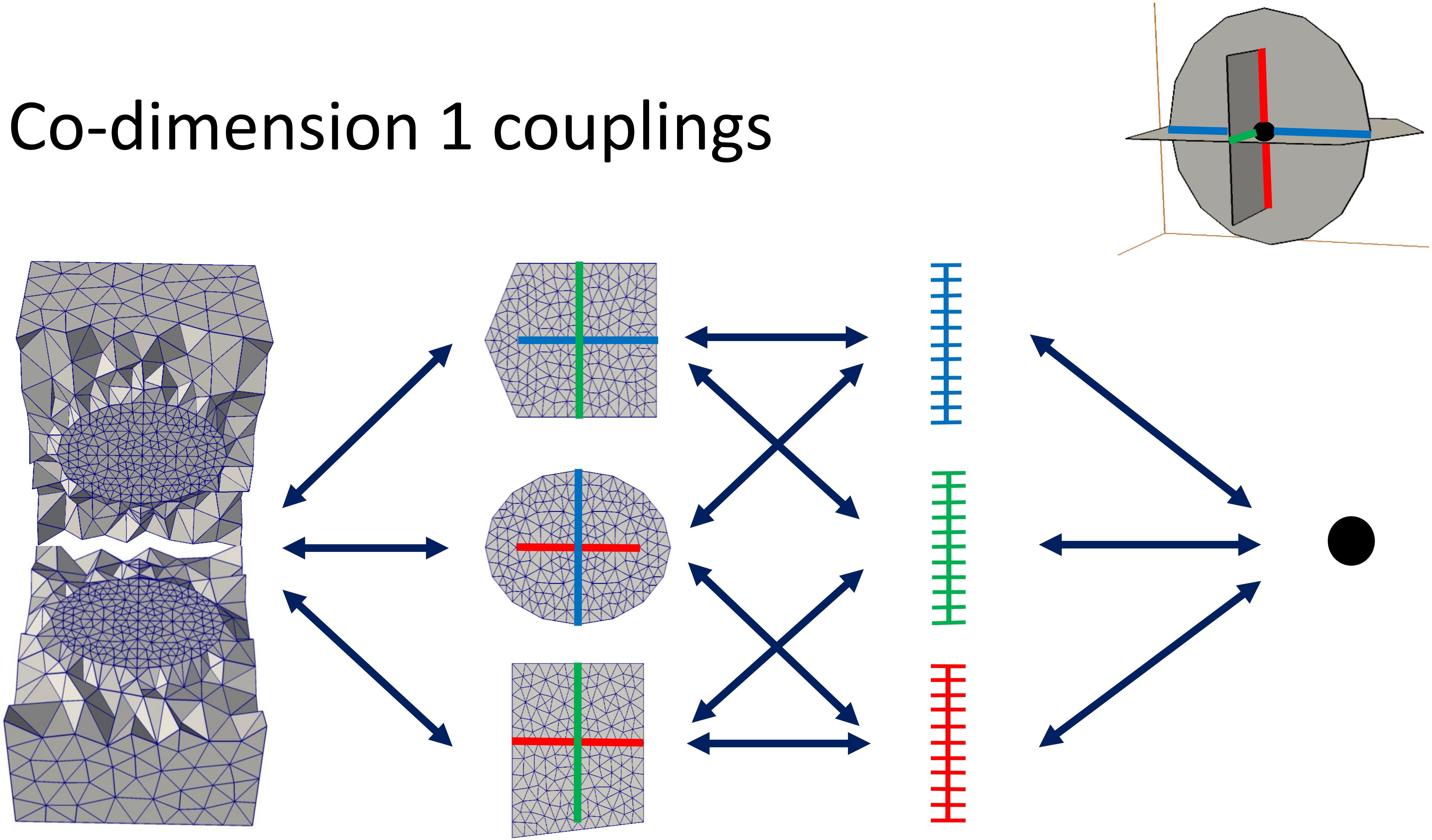
3 intersecting planes embedded in 3D domain:

- 3 2D objects
- 3 1D intersection lines (colored)
- 1 0D intersection of intersections

Grid hierarchy

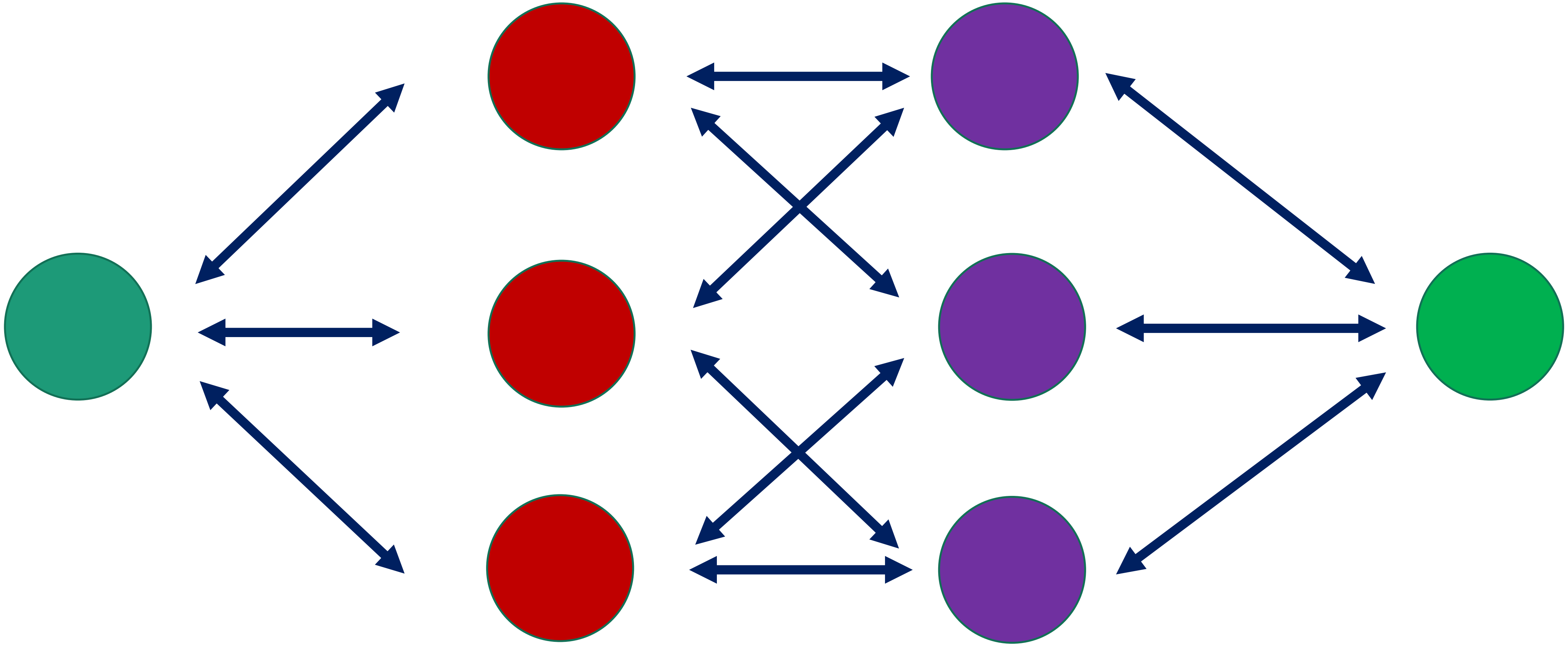


Co-dimension 1 couplings



Graph representation

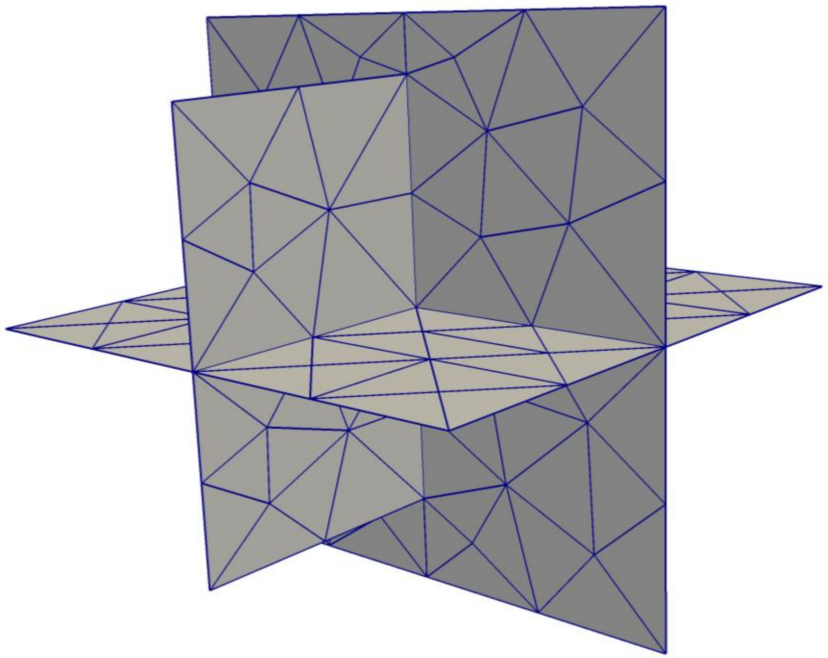
Each node represent a simulation domain
Edges gives rise to boundary conditions / sources



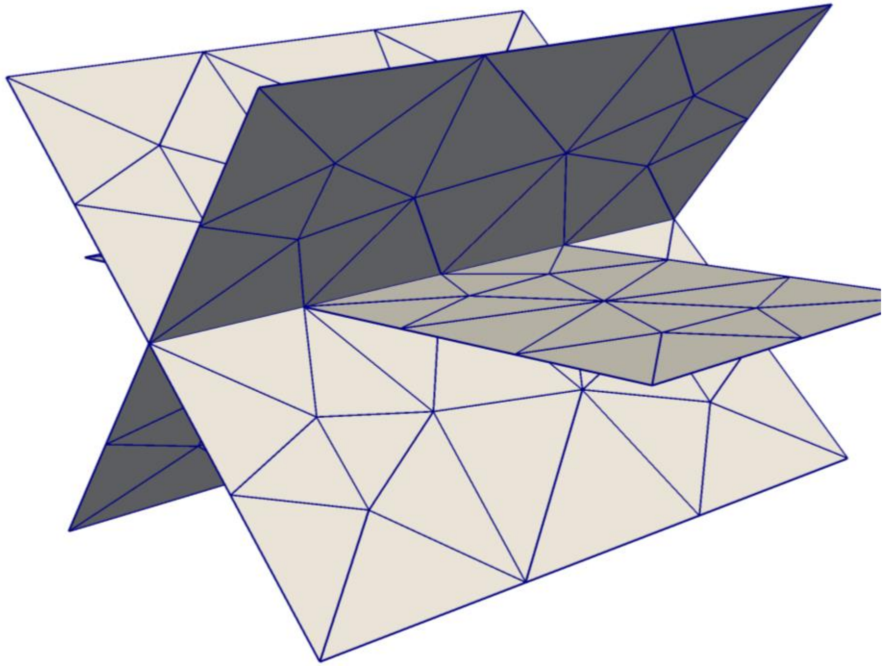
Mixed-dimensional grid in PorePy

```
# Define lower-dimensional objects
f_1 = Fracture([[1, 0, -1], [1.2, 0, -1], [1, 0, 1], [-1, 0, 1]])
f_2 = …

# Construct graph for mixed-dimensional grid, and data storage
# Includes processing of geometry
mesh = meshing.simplex_grid([f_1, f_2, …])

# Loop over nodes
for g, data in mesh:
    data[‘conductivity’] = … # Assign mono-dimensional data

# Loop over edges:
for e, data in mesh.edges():
    data[‘normal_perm’] = … # Assign inter-dimensional data
```

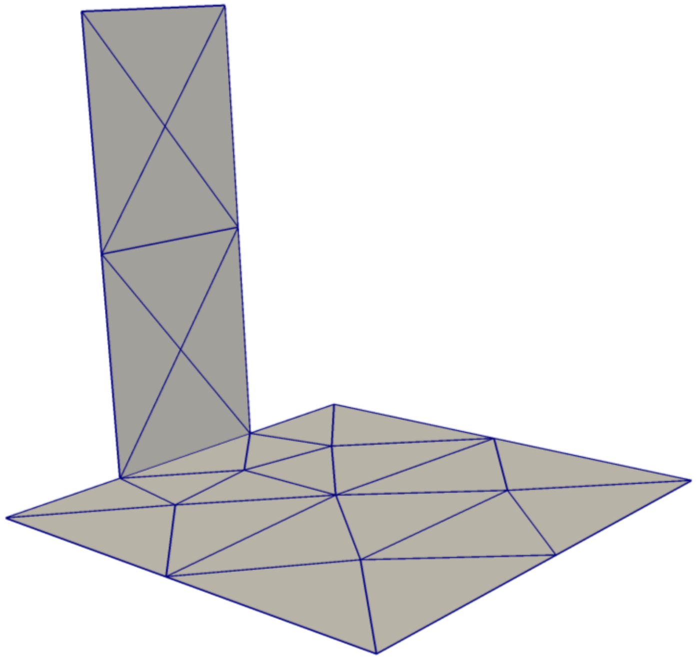
Possible intersection configurations



Intersecting segments

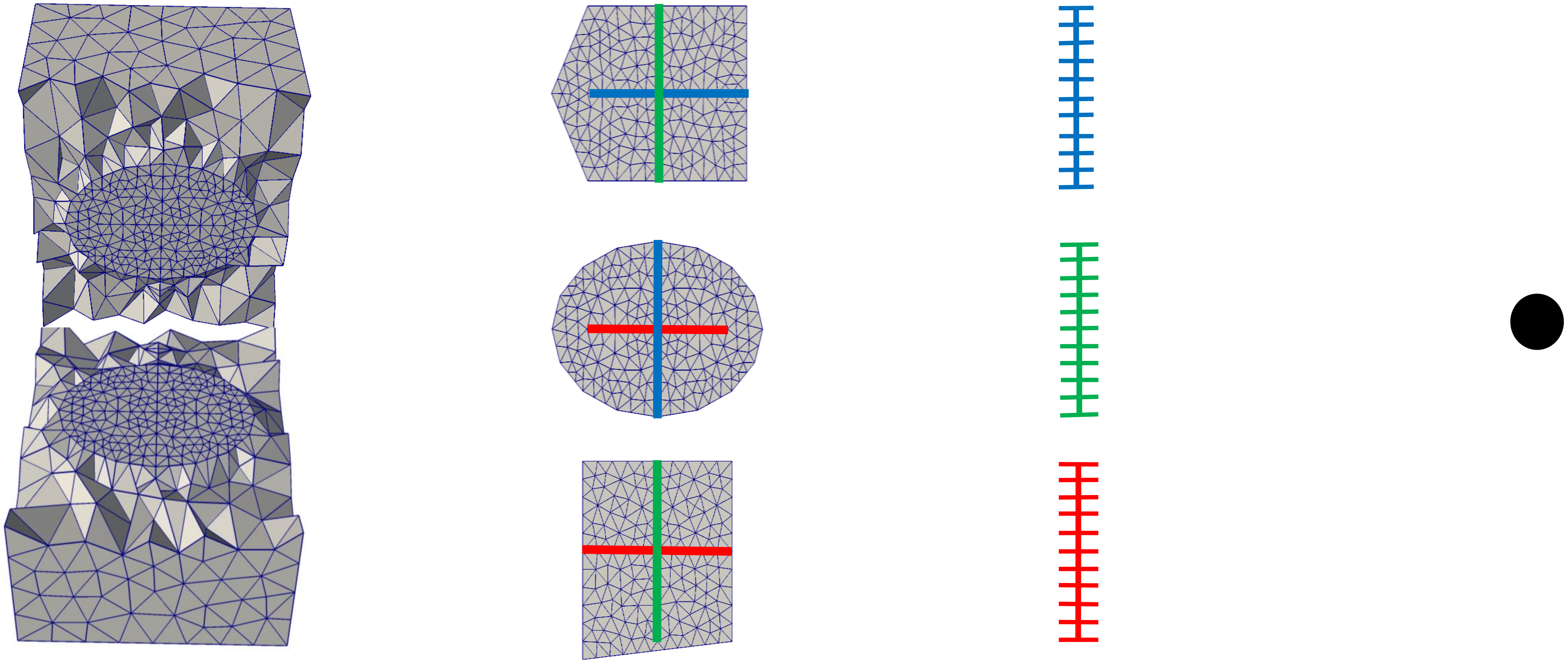


Partially shared segment



L-intersection

Individual grids: Data structure (and implementation) is to a large part a Python translation / extension of corresponding concepts in the Matlab Reservoir Simulation Toolbox.



Discretization of mixed-dimensional problems

Currently implemented methods

- Diffusion equation (mixed-dimensional):
  - TPFA, MPFA, Virtual Element Method (mixed form)
- Advection-diffusion equation (mixed-dimensional):
  - Advection: Upstream weighting
  - Diffusion: TPFA, MPFA
  - Various time stepping schemes
- Linear elasticity (mono-dimensional):
  - Multi-point stress approximations (MPSA)
  - Coupling to fracture deformation models
- Poro-elasticity (mono-dimensional):
  - Coupling of MPFA and MPSA

Discretization of pressure equation

```
# Discretization on individual grids
mono_discr = TPFA()

# Corresponding discretization of inter-dimensional couplings
coupling_discr = TfpaCoupling(mono_discr)

# Combined discretization
combined_discr = Coupler(mono_discr, coupling_discr)

# Loop over all grids and edges, discretize, assemble
A, b = solver_coupler.matrix_rhs(mesh)

# Linear solver
pressure_flux = scipy.sparse.linalg.solve(A, b)
```

Implement new numerical scheme?

1. Discretization on individual grids
2. Handle Neumann boundary conditions
3. Handle source terms

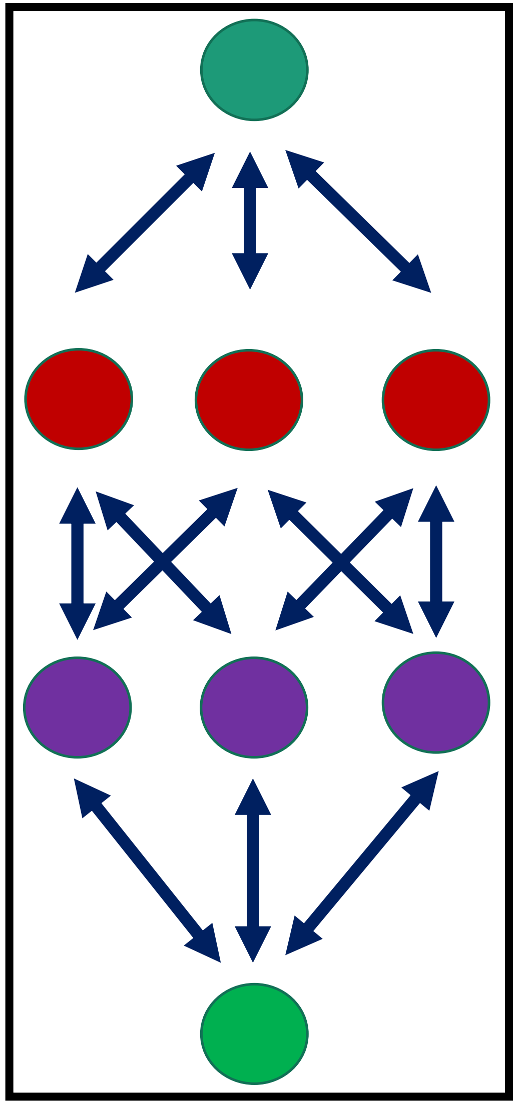
Discretization of pressure equation

```
# Discretization on individual grids
mono_discr = DualVem()
# mono_discr = TPFA()
```

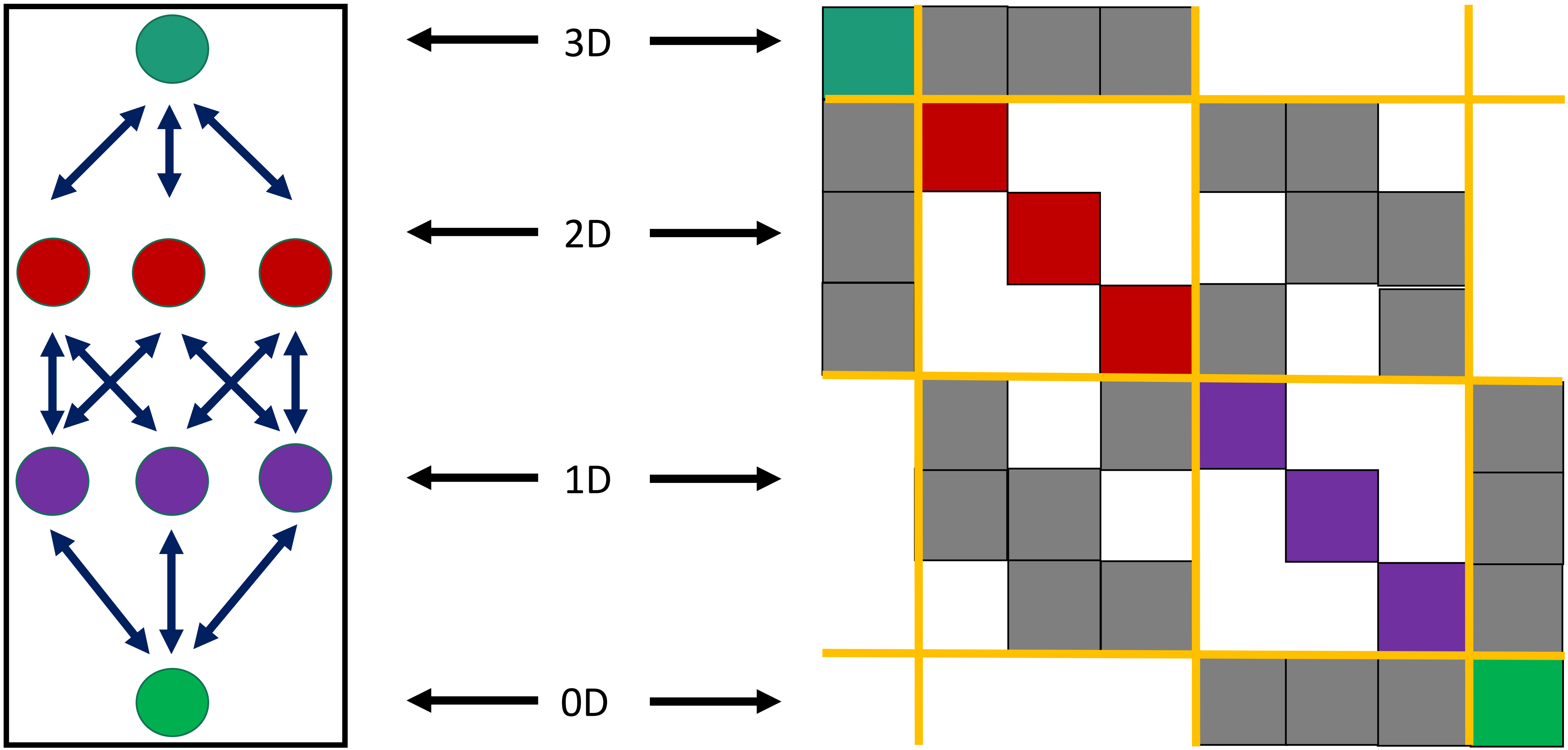
More user-friendly wrappers for problem statements, parameter assignment and discretization / solver is currently being developed.

```
# Combined discretization
combined_discr = Coupler(mono_discr, coupling_discr)

# Loop over all grids and edges, discretize, assemble
A, b = solver_coupler.matrix_rhs(mesh)

# Linear solver
pressure_flux = scipy.sparse.linalg.solve(A, b)
```

Linear system structure



Linear system structure



Example simulations

1. Coupled flow and transport
2. Hydroshearing / low-pressure stimulation

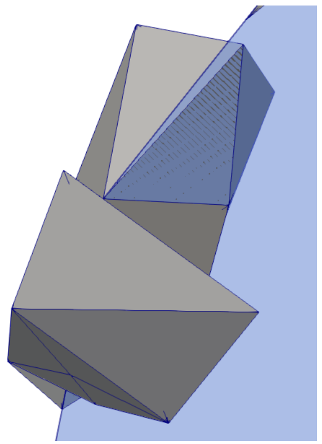
Application: Advection-diffusion

Setup:
Flow from bottom to top
~20 fractures

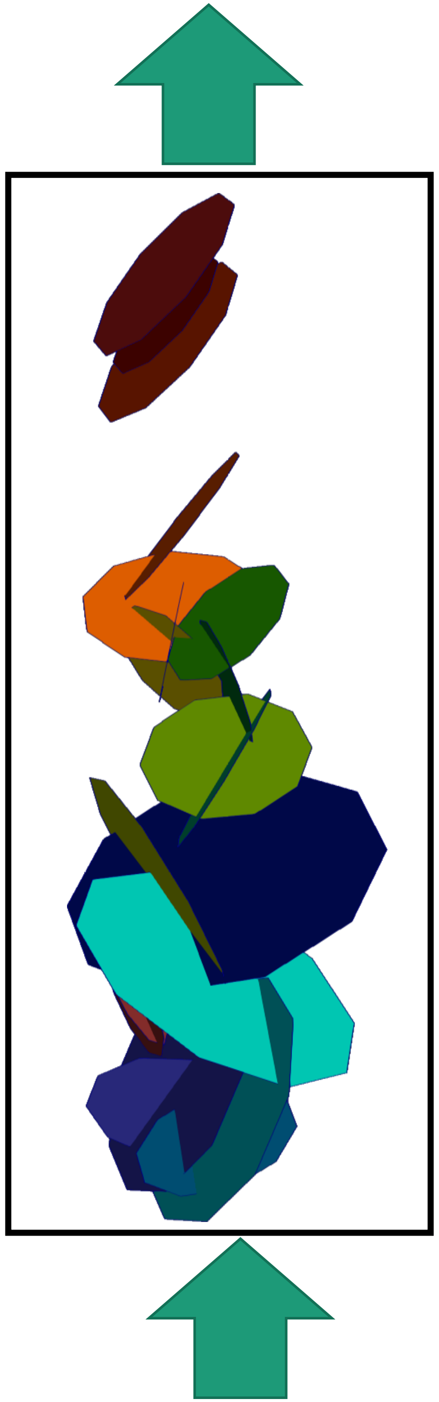
Modeling:
1. Flow field from elliptic pressure equation
2. Concentration by advection-diffusion equation

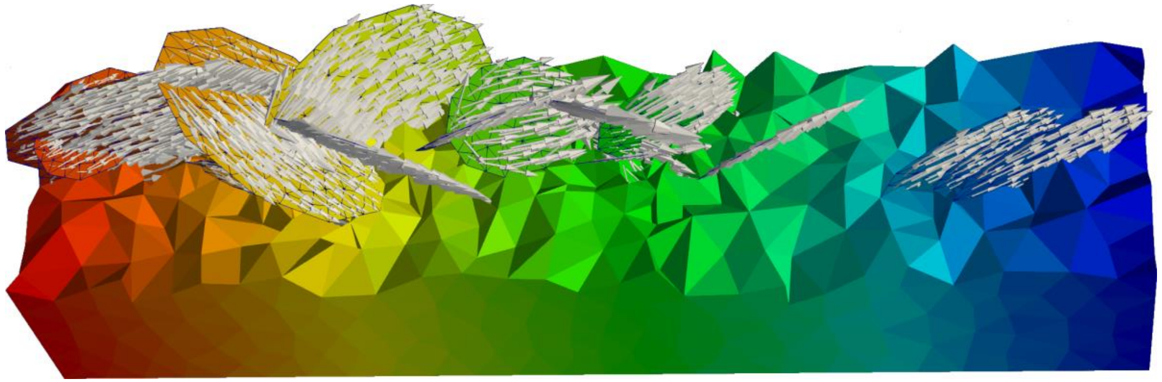
Numerics:
~8000 cells, coarsened from simplex grid
Cells of dimensions {0, 1, 2, 3}

Flow: Mixed virtual element method
Transport: Finite volume

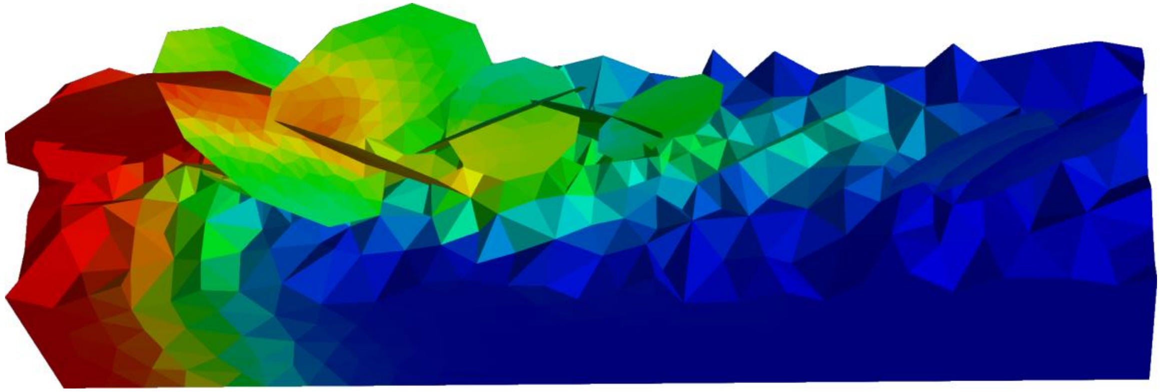


Single cell hugging a fracture

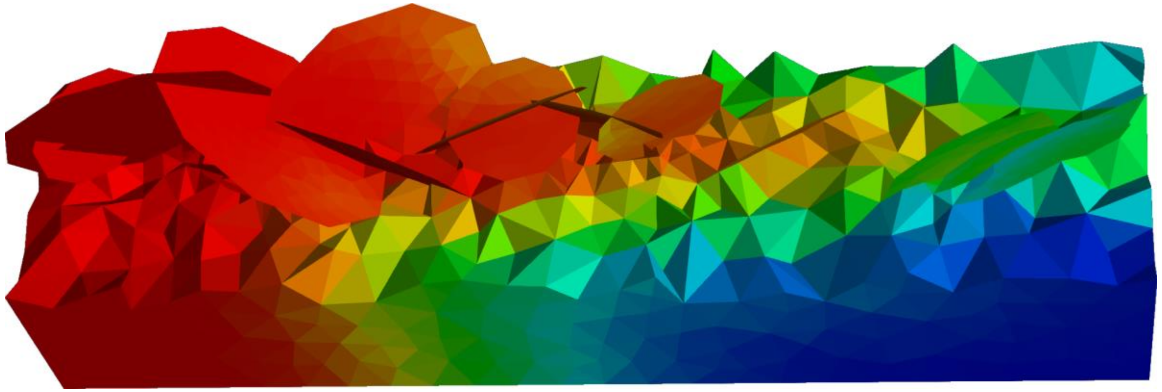




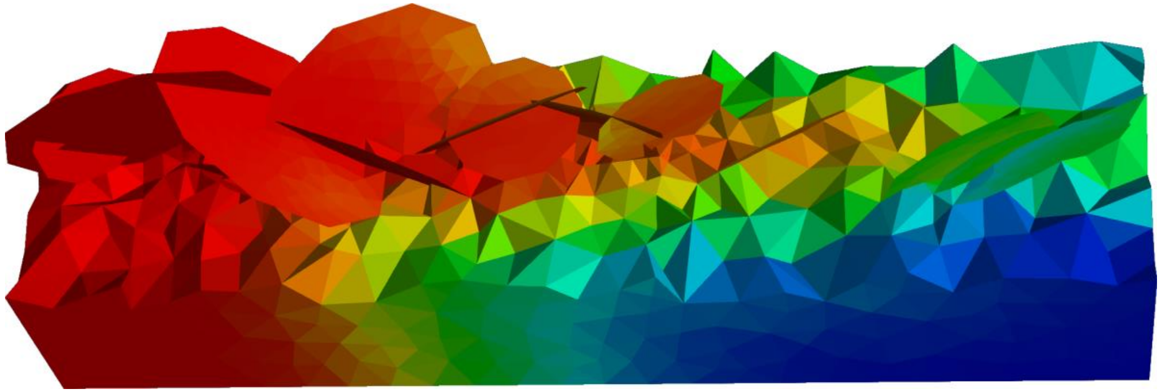
Pressure and fluxes



Tracer concentration (different time steps)

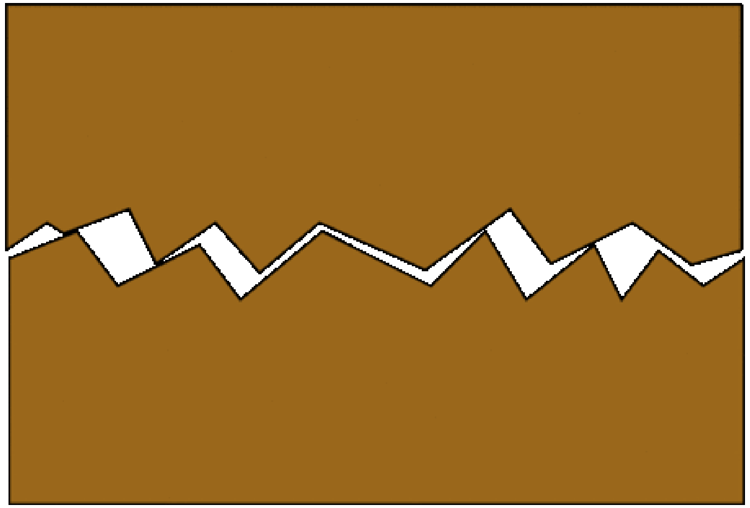


Reference concentration obtained on simplex grid



Application: Stimulation of geothermal reservoirs

- Physical process: Fracture slip due to interaction between fracture fluid pressure and in situ stress field
- Result: Increased fracture width, increased permeability
- Key variables: Stress on fracture surfaces, fluid pressure in fracture

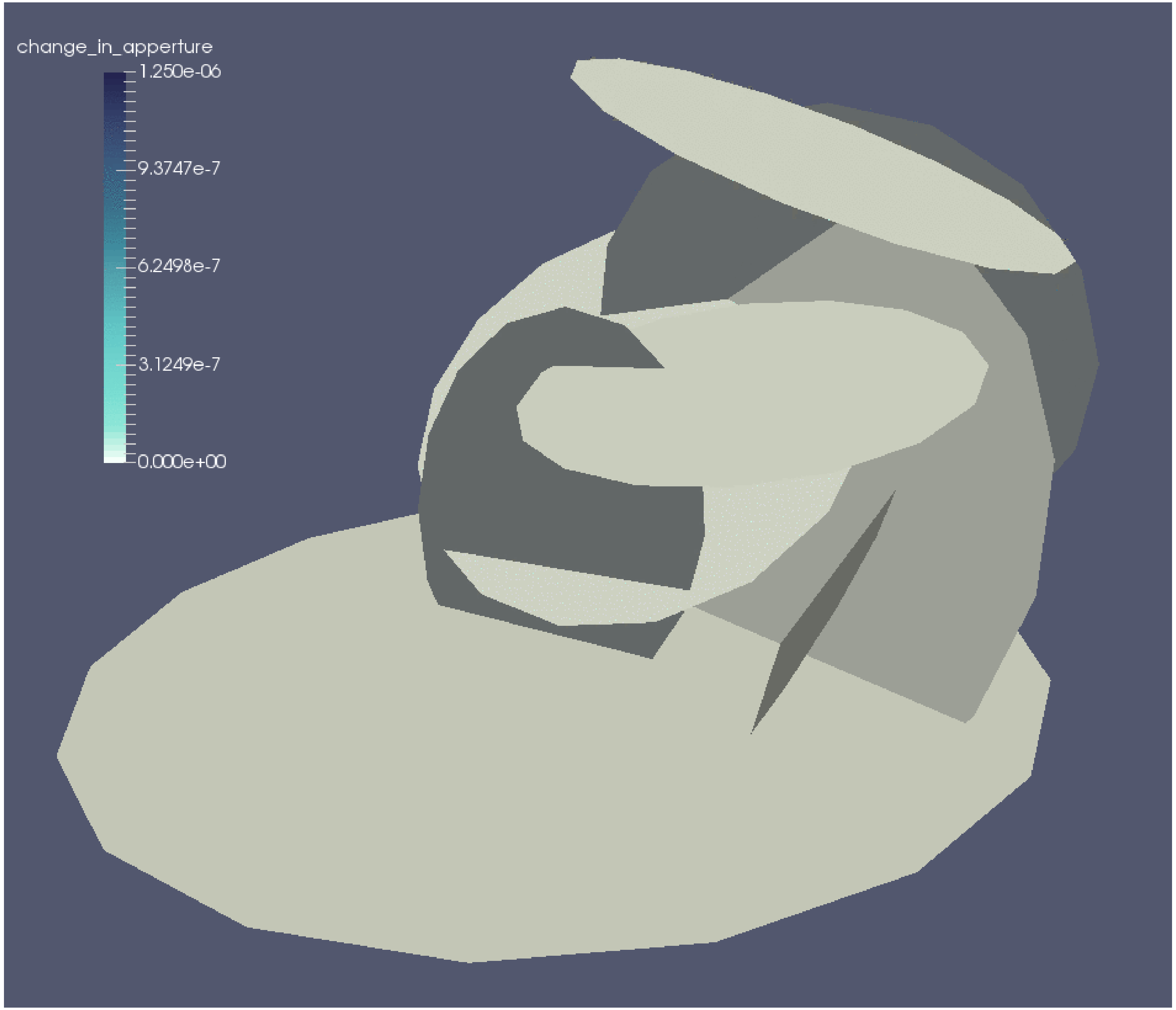


Setup:
Fluid injection, followed by fluid migration in fracture network (and surroundings).

Modeling:
Coupling of flow, elasticity and fracture deformation.

Numerics:
Flow: Finite volume method (TPFA)
Elasticity: Finite volume method (MPSA)

Mixed-dimensional approach for fluid flow only



The road ahead

Likely improvements in the coming monhts

Stronger focus on thermal effects

Multi-physics couplings

- Pressure-temperature couplings
- Thermo-elasticity

Numerical considerations:

- Linear solvers
- Coupling strength

Stability and performance

Current weak points:

1. The code is purely sequential – limited capacity for large-scale networks
   - Likely solution: Use suitable software framework (dune?) as backend
2. Meshing algorithm in 3d is only semi-stable
   - Gradual progress expected
   - Long term goal (dream?): Automatic meshing of (more or less) stochastic networks

Features (likely) still missing in 1-2 years

- Multiphase flow
- Focus on optimal performance
- …

Access

- GPL licence
- Code hosted on GitHub
- Installation: pip install porepy (+ some more)
  - Detailed instructions on GitHub repository
  - Installation from source recommended
- Getting started:
  - Tutorials (jupyter notebooks)
  - Examples (including examples from papers / preprints)

www.github.com/pmgbergen/porepy