

# PyLith

Brad Aagaard, Charles Williams, Matthew Knepley,  
Sue Kientz and Leif Strand



June 22, 2009

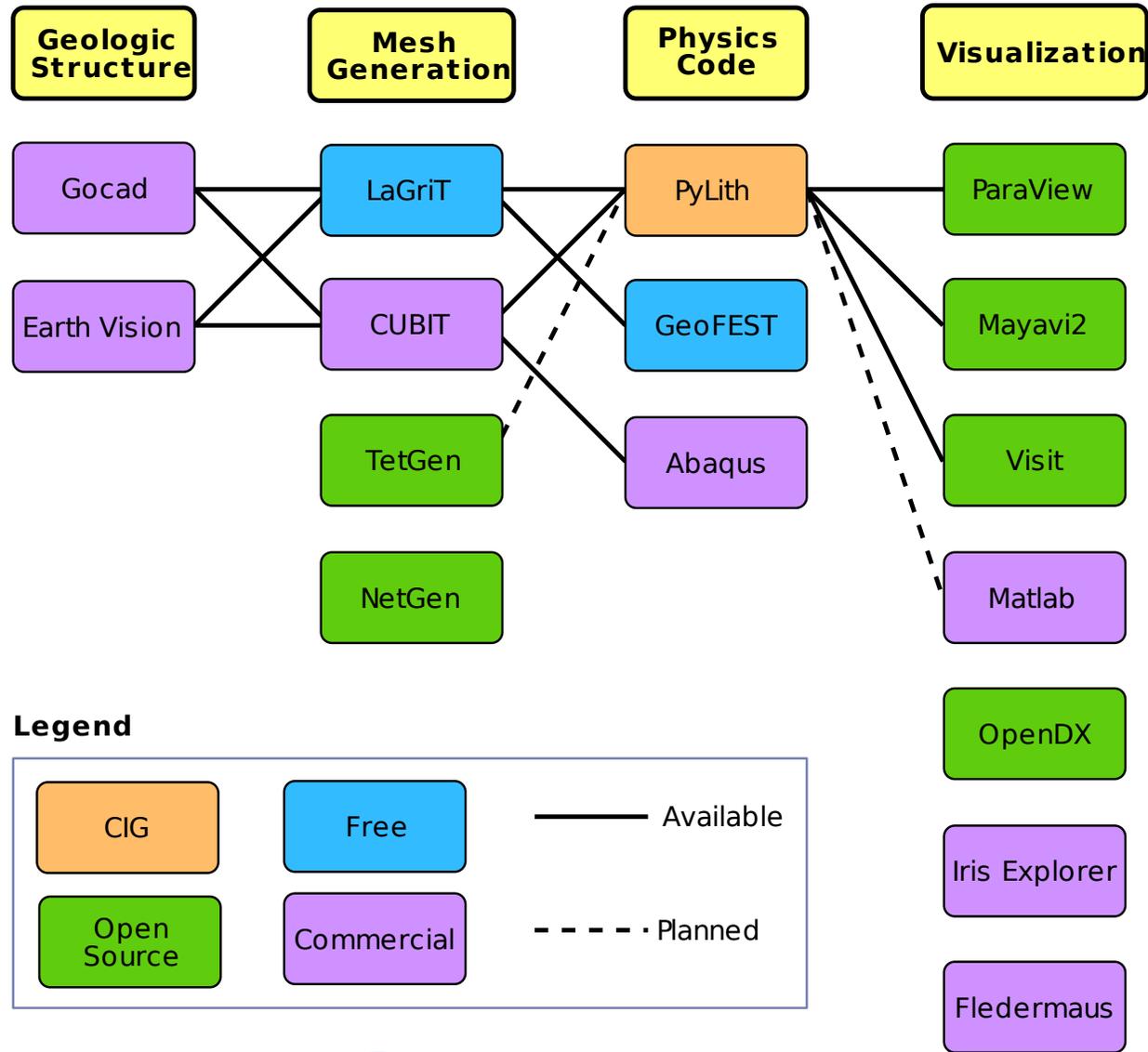
# PyLith

What is it good for?

- Elasticity problems where geometry does not change significantly
- Quasi-static crustal deformation
  - Strain accumulation associated with interseismic deformation
  - Post-seismic relaxation of the crust
  - Volcanic deformation associated with magma chambers and/or dikes
- Dynamic rupture and wave propagation
  - Kinematic (prescribed) earthquake ruptures
  - Local/regional ground-motion modeling

# Crustal Deformation Modeling

Overview of workflow for typical research problem



# Features in PyLith 1.4

Enhancements and new features in blue

- Time integration schemes
  - Implicit time stepping for quasi-static problems
  - Explicit time stepping for dynamic problems
- Bulk constitutive models
  - Elastic model (1-D, 2-D, and 3-D)
  - Linear and Generalized Maxwell viscoelastic models (3-D)
  - Power-law viscoelastic model (3-D)
- Boundary and interface conditions
  - Time-dependent Dirichlet boundary conditions
  - Time-dependent Neumann (traction) boundary conditions
  - Absorbing boundary conditions
  - Kinematic (prescribed slip) fault interfaces w/multiple ruptures
  - Time-dependent point forces
  - Gravitational body forces

# Features in PyLith 1.4 (cont.)

Enhancements and new features in blue

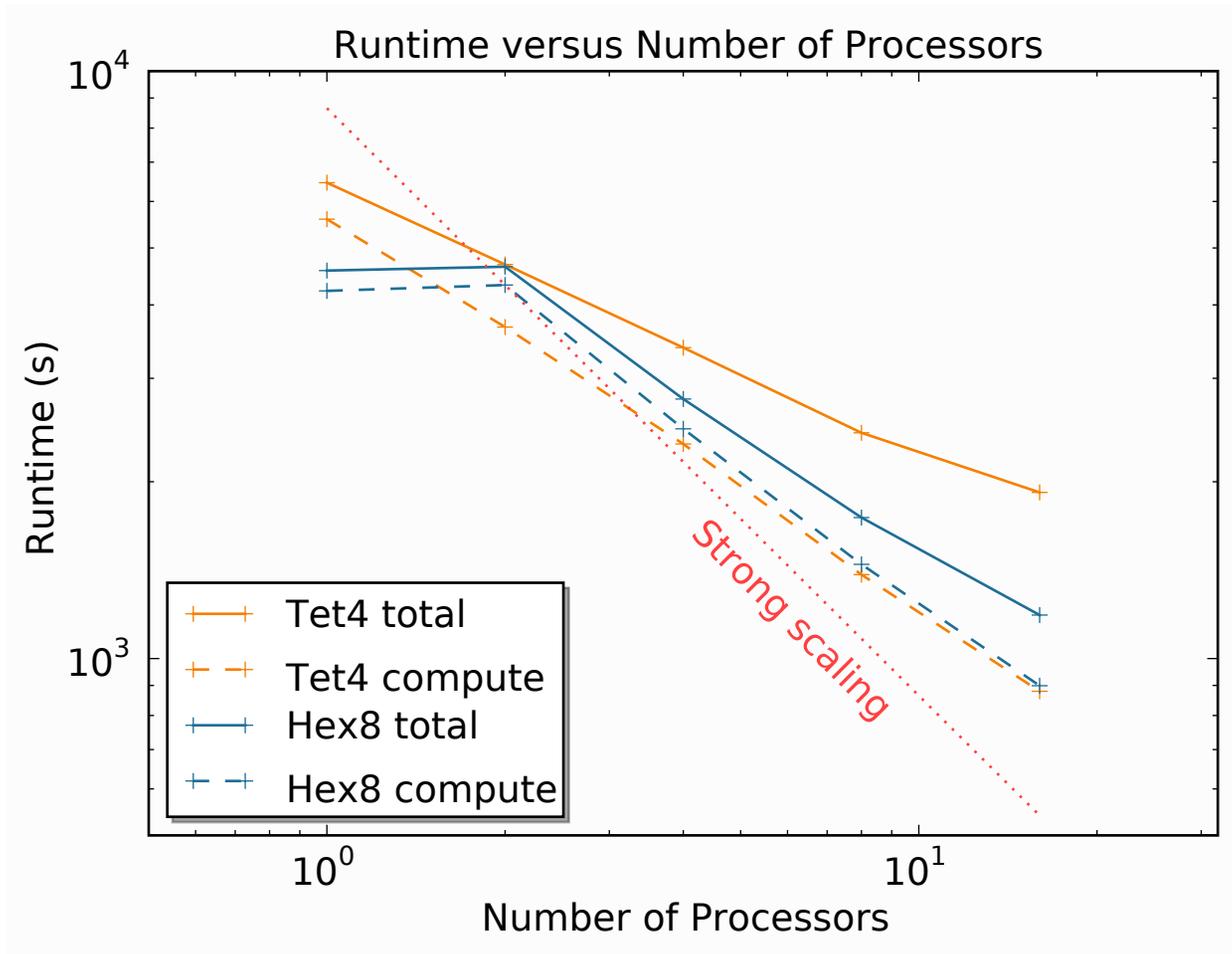
- Automatic and user-controlled time stepping
- Ability to specify initial stress state
- Importing meshes
  - LaGriT: GMV/Pset
  - CUBIT: Exodus II
  - ASCII: PyLith mesh ASCII format (intended for toy problems only)
- Output: VTK files
  - Solution over volume
  - Solution over surface boundary
  - State variables (e.g., stress and strain) for each material
  - Fault information (e.g., slip and tractions)
- Automatic conversion of units for all parameters

# PyLith 1.4: Under-the-hood Improvements

- General cleanup of C++ code
- Pyrex/pyrexembed replaced by SWIG
  - Greatly simplifies creating Python bindings for C++ objects
  - SWIG generated files included in source distribution
  - User-defined spatial databases and bulk constitutive models
- Automatic nondimensionalization of problem
  - User supplies pressure, time, and length scale of problem
  - All parameters nondimensionalized appropriately
  - Eliminates need to condition terms in sparse matrix
  - Restores symmetry of sparse matrix (reduces memory use)
- Integration with PETSc Scalable Nonlinear Equations Solvers
  - Displacement increment formulation for implicit and dynamic time-stepping

# PyLith 1.4 Performance

PyLith 1.4 is  $\sim 5\%$  faster and uses  $\sim 50\%$  less memory than PyLith 1.3



# PyLith 1.x: Planned Releases

Current productivity is about 2 feature releases per year

- PyLith 1.5: anticipate release in late 2009 or early 2010
  - Fault constitutive behavior with several widely used friction models
  - Ability to specify initial strain and state variables
- PyLith 1.6: Large deformations and finite strain
- PyLith 1.7: Automation of 4-D Green's functions
- PyLith 1.8: Coupling of quasi-static and dynamic simulations
- Long-term objectives
  - Adaptive mesh refinement
  - Easier meshing (better meshers or ability to use structured meshes)

# PyLith Design Objective

Want a code developed for and by the community

- Modular
  - Users can swap modules to run the problem of interest
- Scalable
  - Code runs on one to a thousand processors efficiently
- Extensible
  - Expert users can add functionality to solve their problem without polluting main code

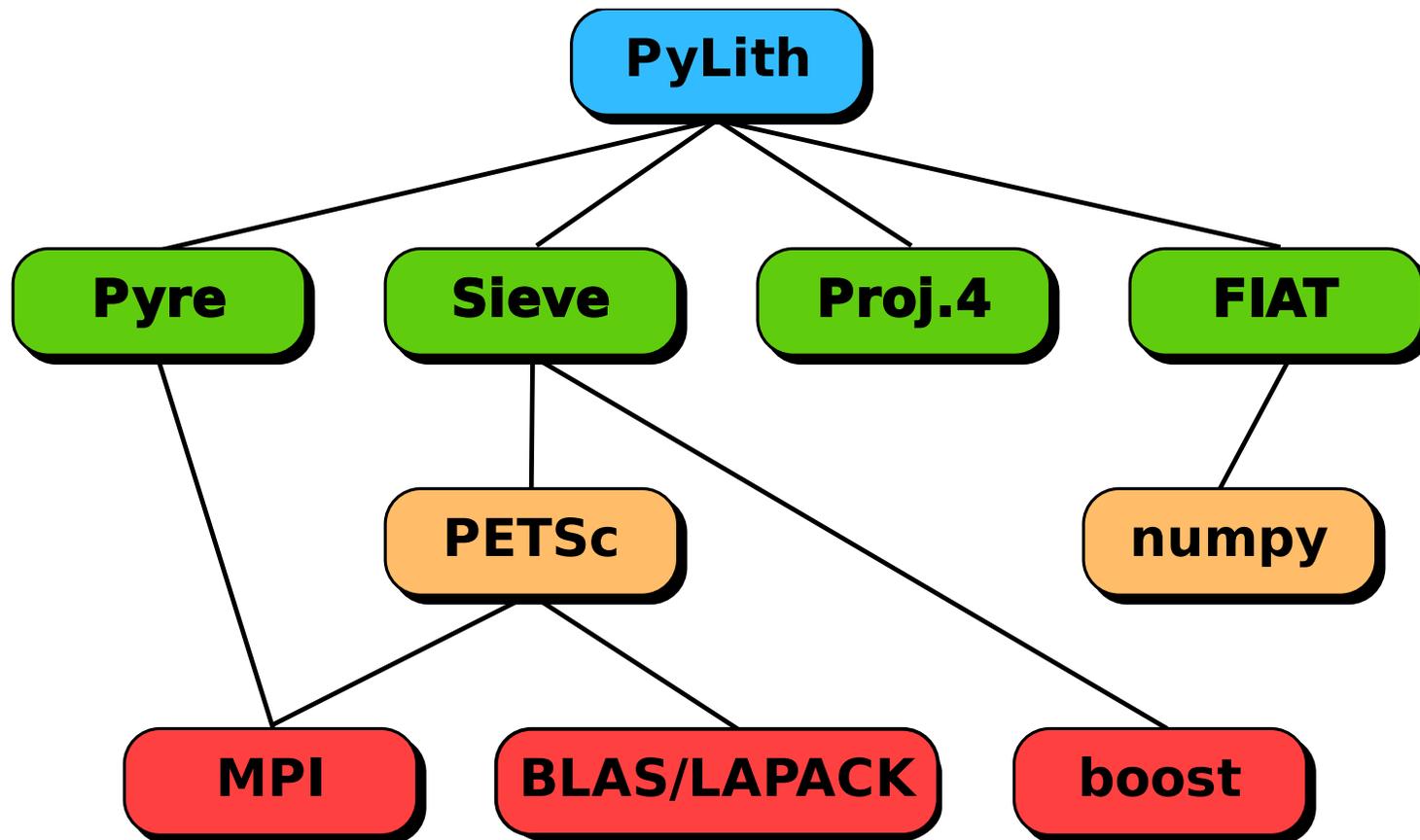
# PyLith is a Community Code

Success of code depends on community participation

- End-users (anyone who uses the code)
  - Help define and prioritize features that should be added
  - Report bugs/problems and suggest improvements
- Expert users
  - Help test alpha versions of releases
  - Run benchmarks and report results
  - Contribute meshing and visualization examples to documentation
  - Add features following template (e.g., constitutive models)
- Developer
  - Define development strategy
  - Implement new features and tests
  - Write documentation

# PyLith Design: Focus on Geodynamics

Leverage packages developed by computational scientists



# PyLith Design: Code Architecture

Flexible and modular with good performance

- Top-level code written in Python
  - Expressive, high-level, object-oriented language
  - Dynamic typing allows adding additional modules at runtime
  - Convenient scripting
- Low-level code written in C++
  - Compiled (fast execution), object oriented language
- Bindings to glue Python & C++ together
  - SWIG generates code for calling C++ functions from Python

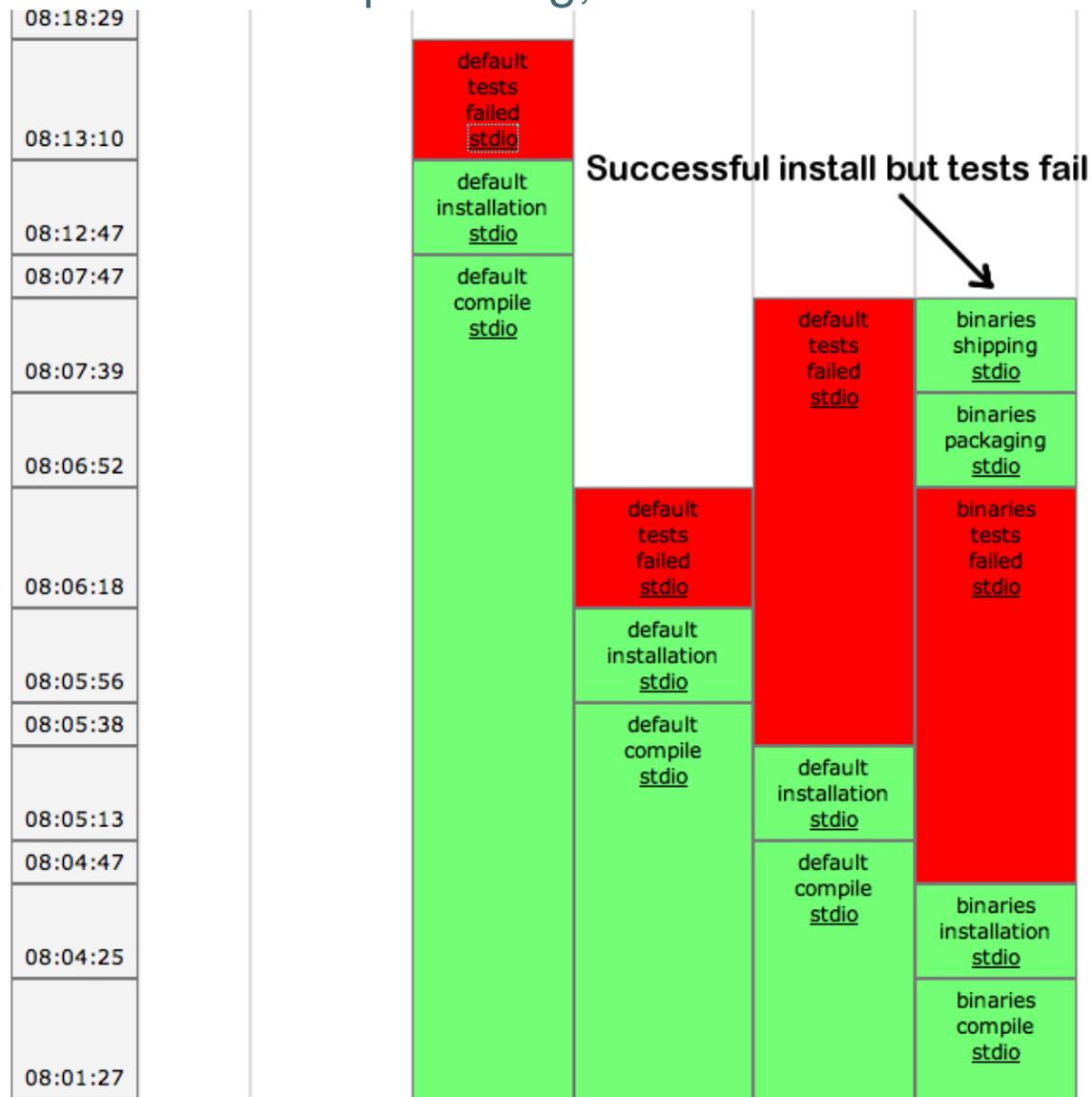
# PyLith Design

Tests, tests, and more tests (>1100 in all)

- Create tests for nearly every function during development
  - Remove most bugs during initial implementation
  - Isolate and expose bugs at origin
- Create new tests to expose bugs reported
  - Prevent bugs from reoccurring
- Rerun tests whenever code is changed
  - Allows optimization of performance with quality control
  - Code continually improves

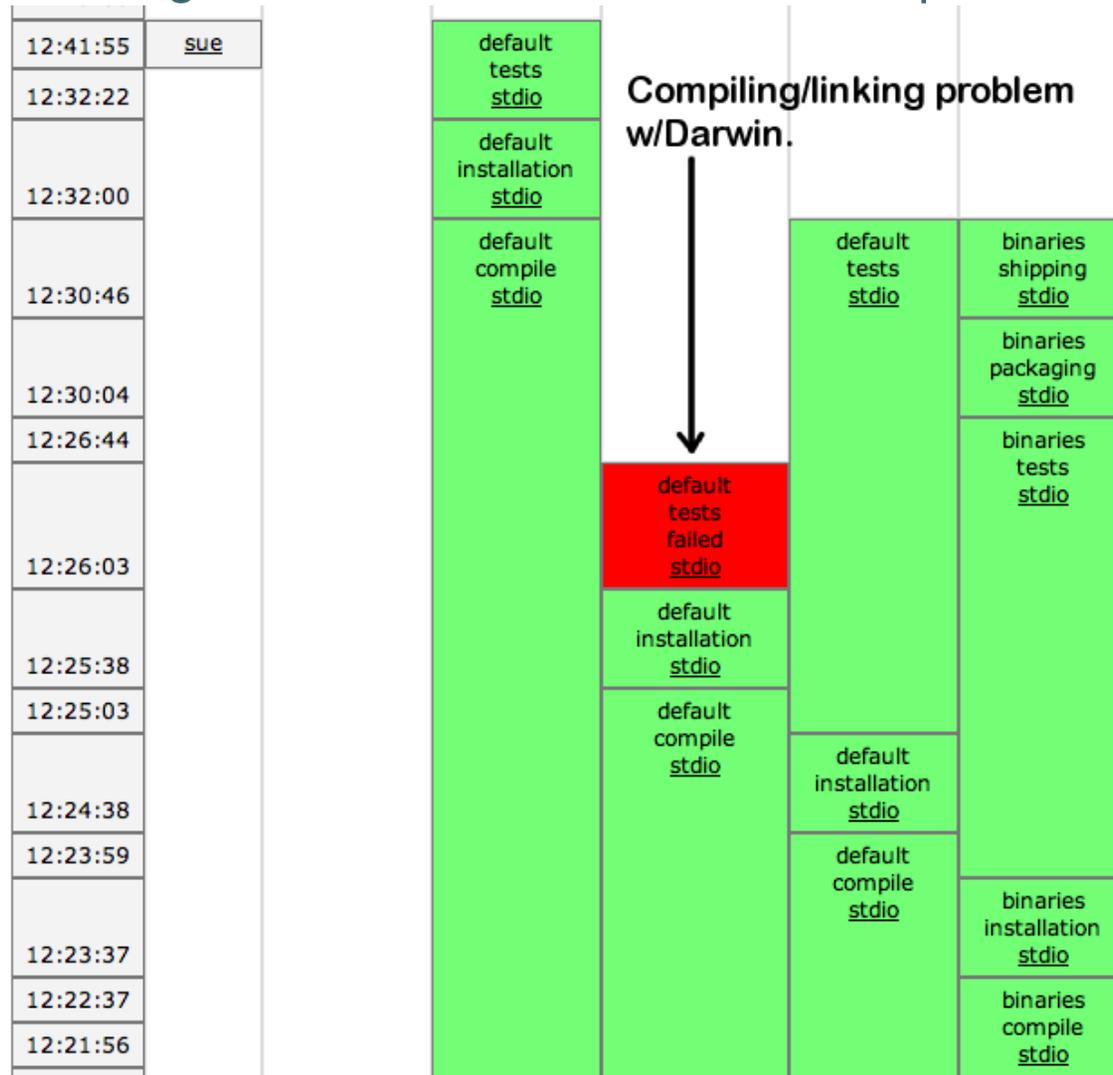
# Example of Automated Building and Testing

Test written to expose bug, buildbot shows tests fail



# Automated Building and Testing

Bug is fixed, buildbot shows tests pass



# Implementation: Finite-Element Data Structures

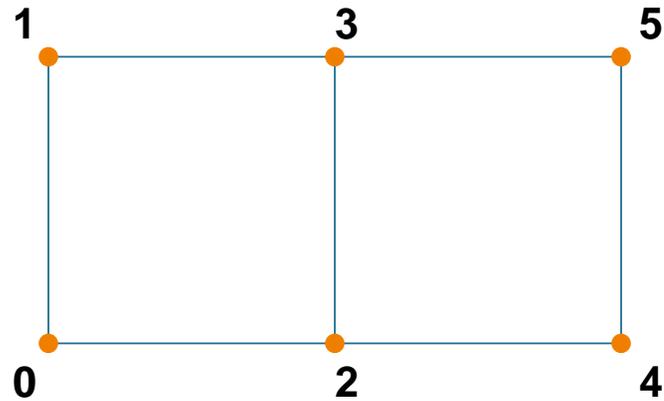
Use Sieve for storage and manipulating mesh information

- PyLith makes only a few MPI calls
- Data structures are independent of basis functions and reference cells
  - Same code for many cell shapes and types
  - Physics implementation limits code, not data structures
- Sieve routines force adhering to finite-element formulation
  - Do not have access to underlying storage
  - Manipulations must be done using Sieve interface
  - Only valid finite-element manipulation is allowed

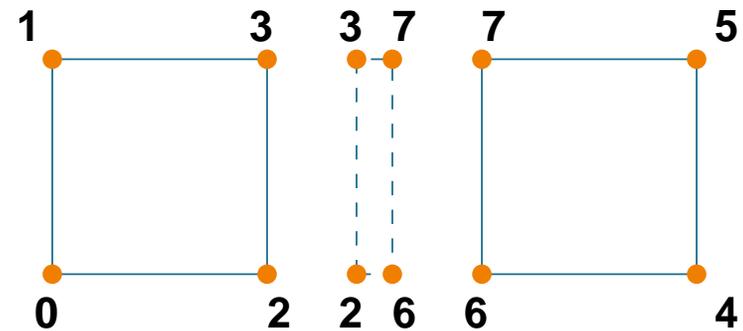
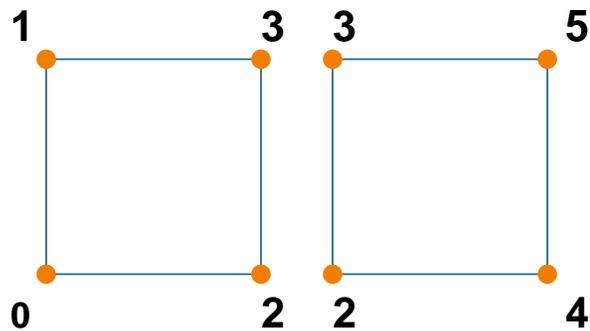
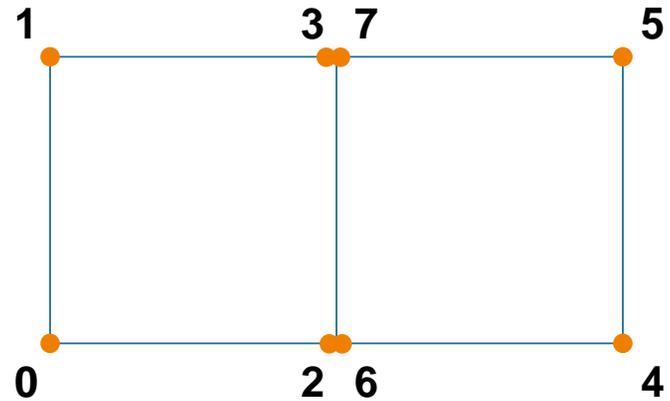
# Implementation: Fault Interfaces

Use cohesive cells to control fault behavior

Original Mesh



Mesh with Cohesive Cell



Exploded view of meshes

# Kinematic (prescribed) slip earthquake ruptures

Use Lagrange multipliers to specify slip

- System without cohesive cells

$$\underline{A}\vec{u} = \vec{b}$$

- System with cohesive cells

$$\begin{pmatrix} \underline{A} & \underline{C}^T \\ \underline{C} & 0 \end{pmatrix} \begin{pmatrix} \vec{u} \\ \vec{L} \end{pmatrix} = \begin{pmatrix} \vec{b} \\ \vec{D} \end{pmatrix}$$

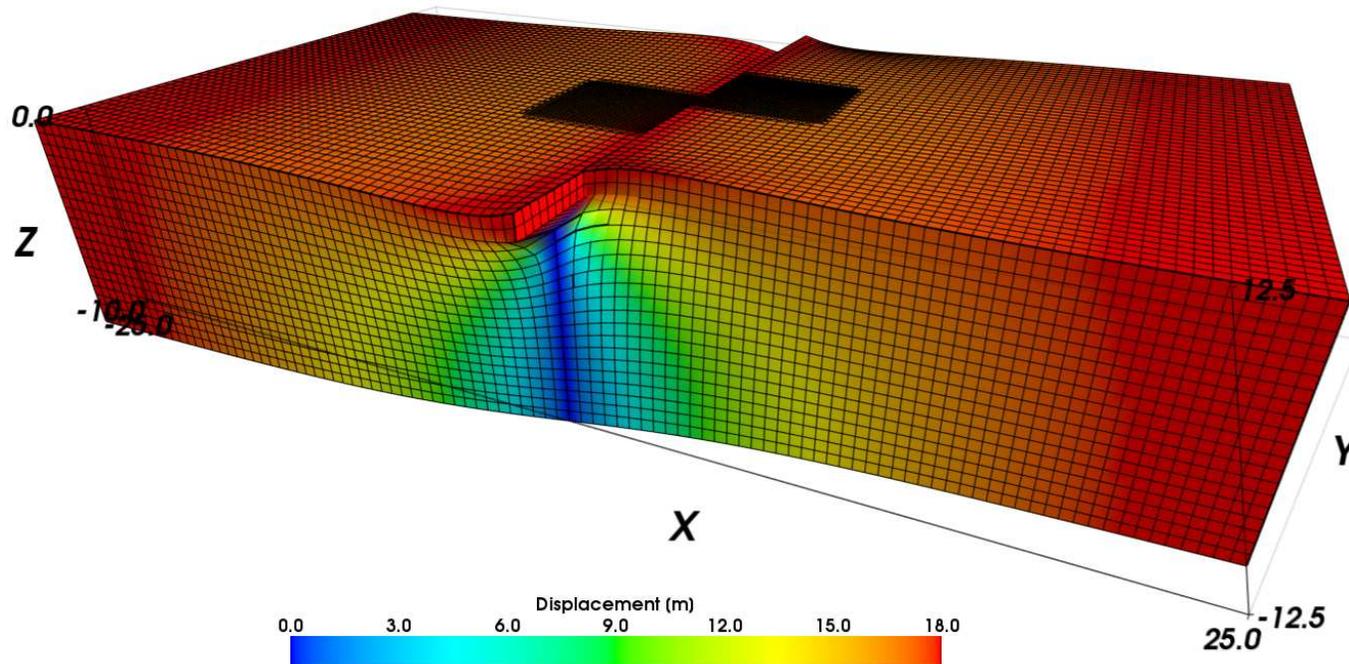
# Implementing Fault Slip with Lagrange multipliers

- Advantages
  - Fault implementation is local to cohesive cell
  - Solution includes forces generating slip (Lagrange multipliers)
  - Retains block structure of matrix (same number of DOF per vertex)
  - Offsets in mesh mimic slip on natural faults
- Disadvantages
  - Creates a saddle point problem (slower convergence)
  - Mixes displacements and forces in solution

# Benchmarking PyLith

Analytical solution from Savage and Prescott (1978)

- Repeated rupture on a vertical, strike-slip fault
- Elastic layer over a linear Maxwell viscoelastic half-space
- Steady creep over bottom half of the elastic layer



# Benchmarking PyLith

Simulation closely matches analytical solution during 10th eq cycle

