# PyLith

Brad Aagaard, Charles Williams, Matthew Knepley
Sue Kientz and Leif Strand
Computational Infrastructure for Geodynamics

June 23, 2008

# Outline

- Introduction to PyLith
  - Motivation & development objective
  - What does PyLith do?
- PyLith Design
  - Architecture and programming languages
  - Development strategy
- Features
  - Current release
  - Planned releases
- Benchmark results
- Tutorial

# Motivation for Developing PyLith

- Available modeling codes
  - rarely solve the problem **you** want to solve
  - are often poorly documented
  - may not work correctly
- Current research demands larger, more complex simulations
- Want to avoid multiple, incompatible versions of the same code

# PyLith

## What is it good for?

- Quasi-static crustal deformation
  - Interseismic deformation
  - Post-seismic deformation
  - Volcano deformation
- Dynamic rupture and wave propagation
  - Kinematic (prescribed) earthquake ruptures
  - Strong ground motion modeling

# Features in PyLith 1.2

- Spatial dimensions: 1-D, 2-D, or 3-D
- 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)
- Boundary and interface conditions
  - Dirichlet (prescribed displacement and velocity) boundary conditions
  - Neumann (traction) boundary conditions
  - Absorbing boundary conditions
  - Kinematic (prescribed slip) fault interfaces w/multiple ruptures
  - Gravitational body forces

# Features in PyLith 1.2 (cont.)

- 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)

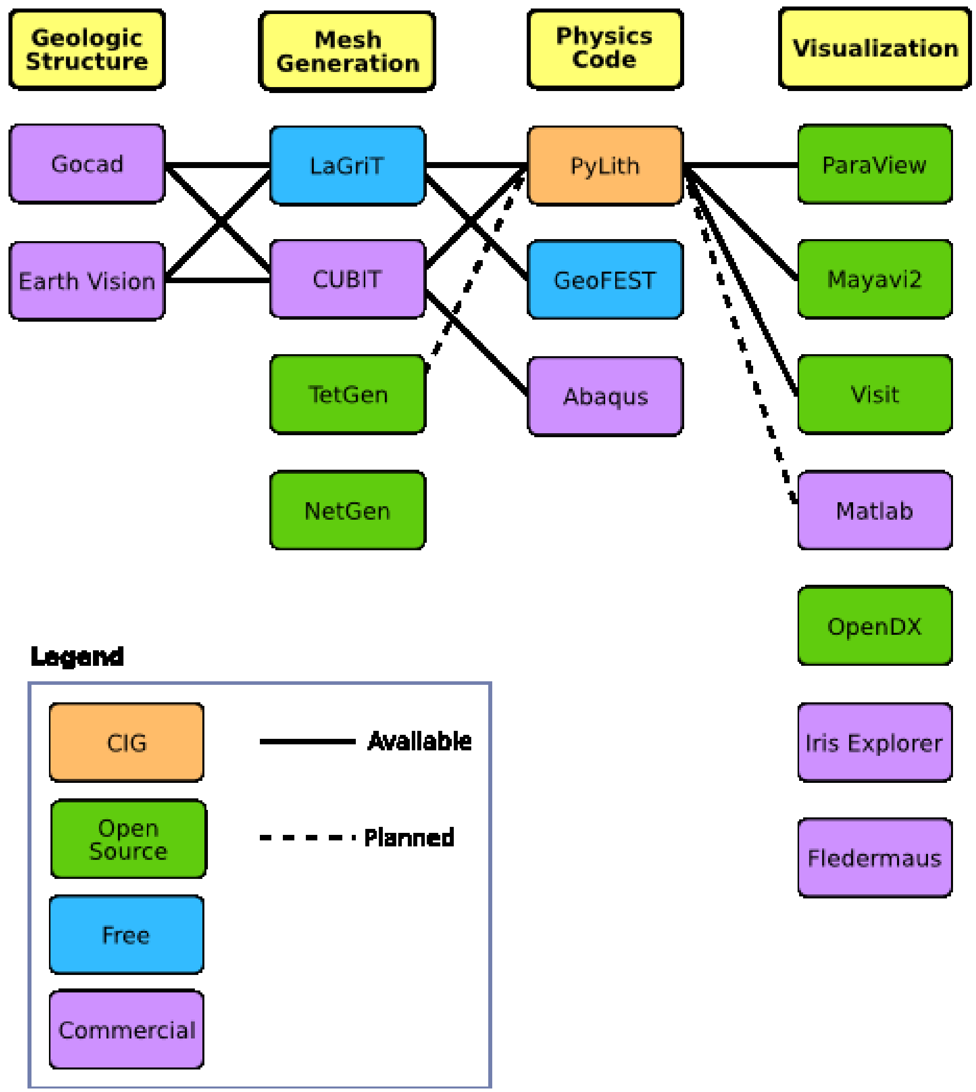
# PyLith Design Objective

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

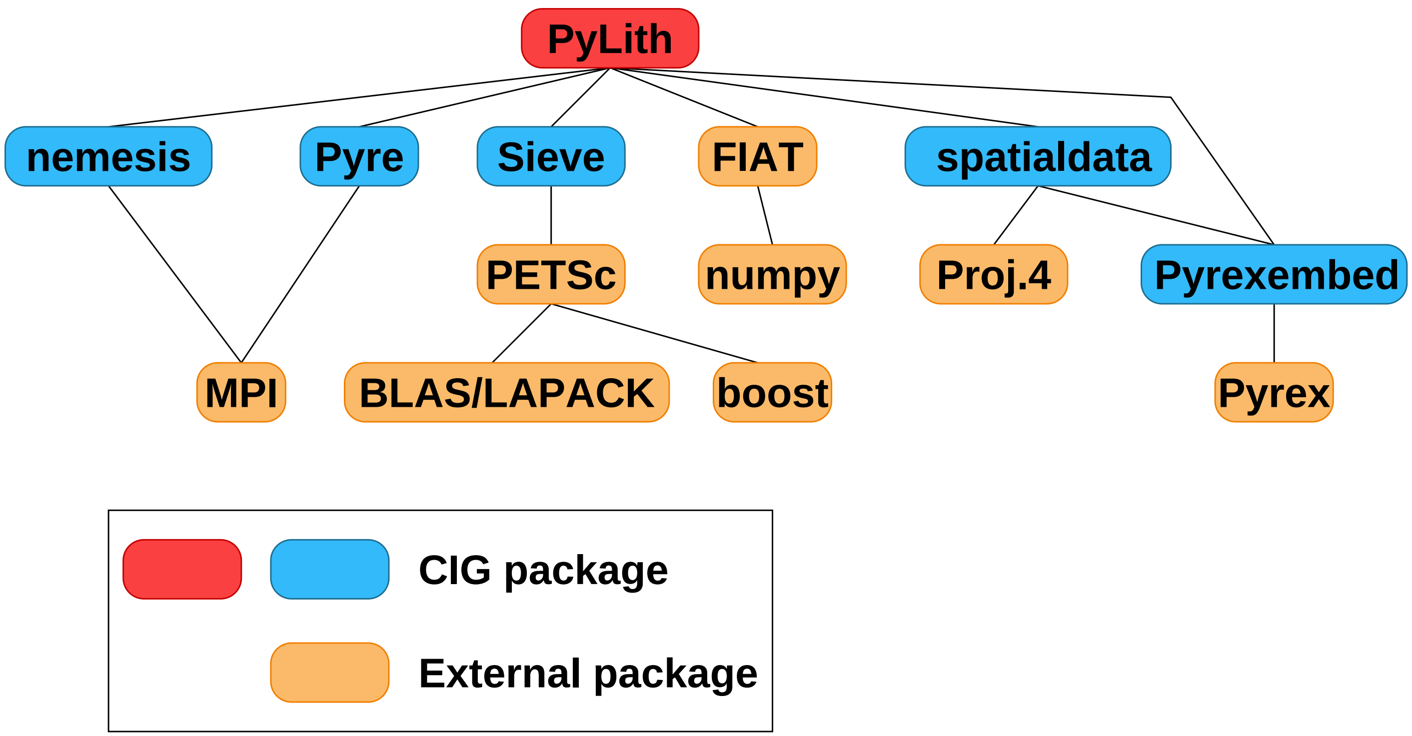
# PyLith

## Overview of workflow for typical research problem



# 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
  - Pyrex/pyrexembed generate C code for calling C++ from Python

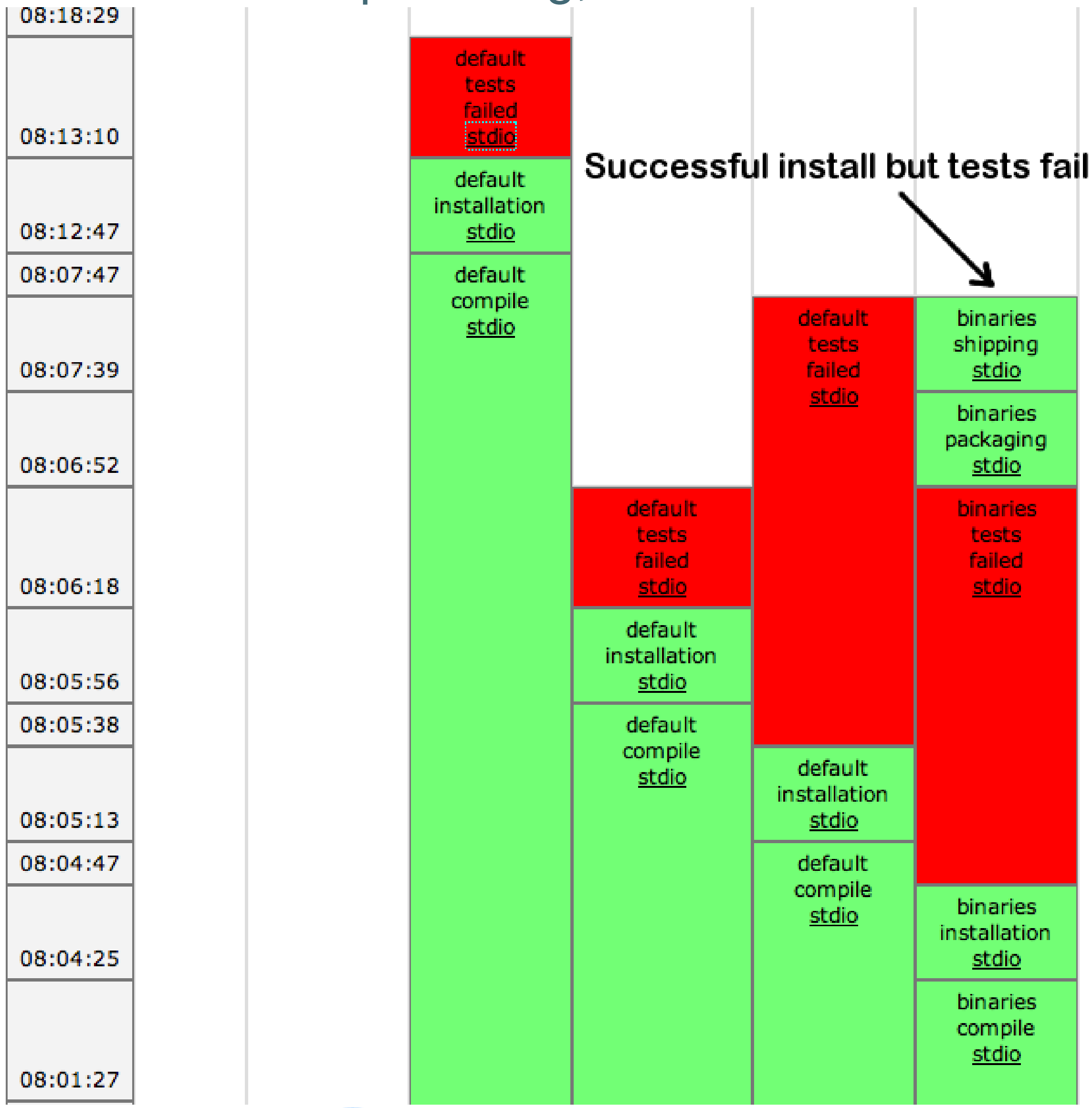
# PyLith Design

## Tests, tests, and more tests (>700 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

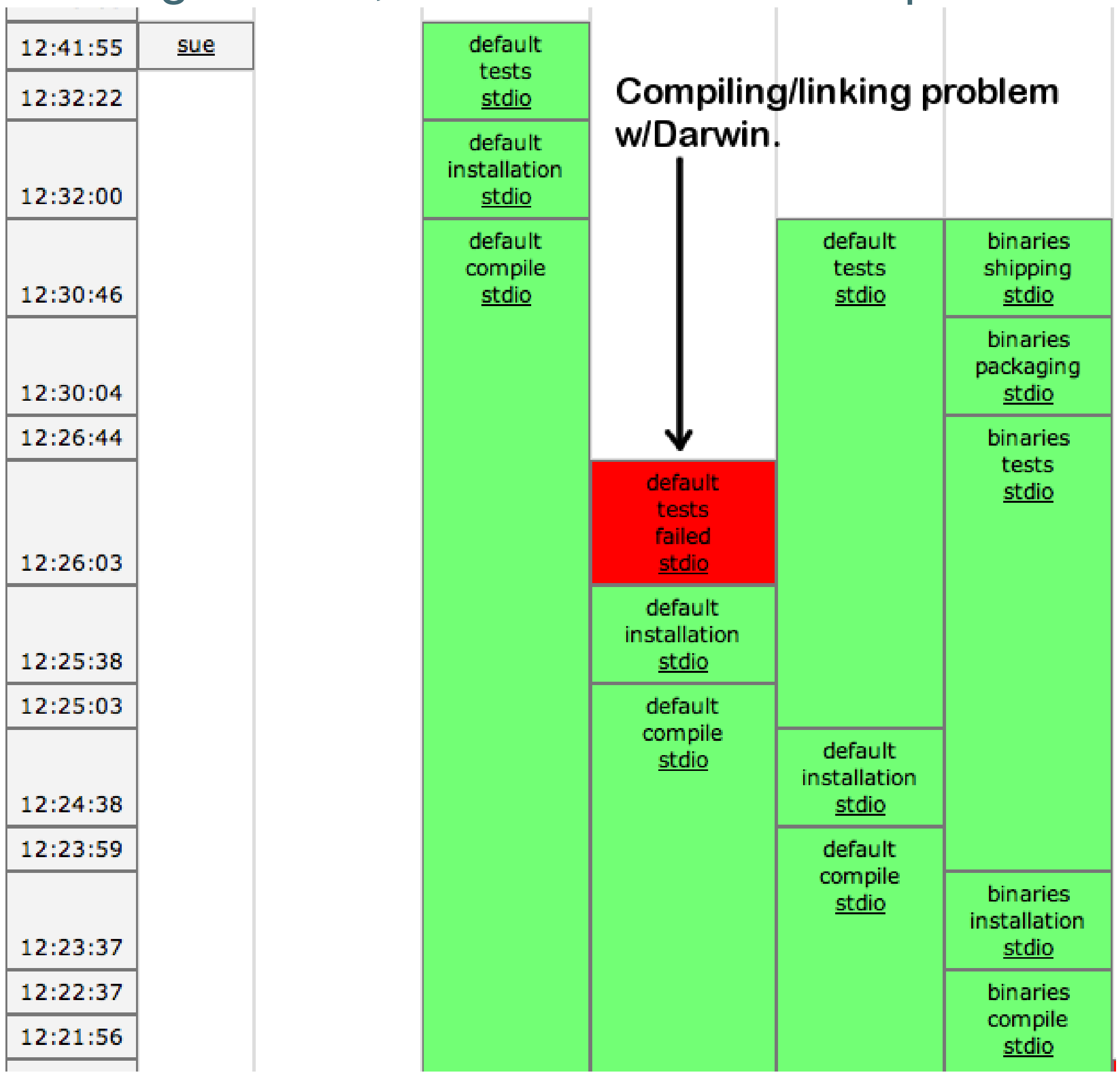
# Example of Automated Building and Testing

Test written to expose bug, buildbot shows tests fail



# Automated Building and Testing

## Bug is fixed, buildbot shows tests pass



# PyLith 1.x: Planned Releases

First add features present in Tecton and EqSim

- PyLith 1.3: anticipate release in late summer 2008
  - Initial stress state for constitutive models
  - Adaptive time stepping
- PyLith 1.4: anticipate release in Dec 2008
  - Fault constitutive behavior
  - Nonlinear bulk constitutive models
- PyLith 1.5: anticipate release in Jun 2009
  - Time dependent boundary conditions
  - Large deformations and finite strain
- PyLith 1.6: Automation of 4-D Green's functions
- PyLith 1.7: Coupling of quasi-static and dynamic simulations

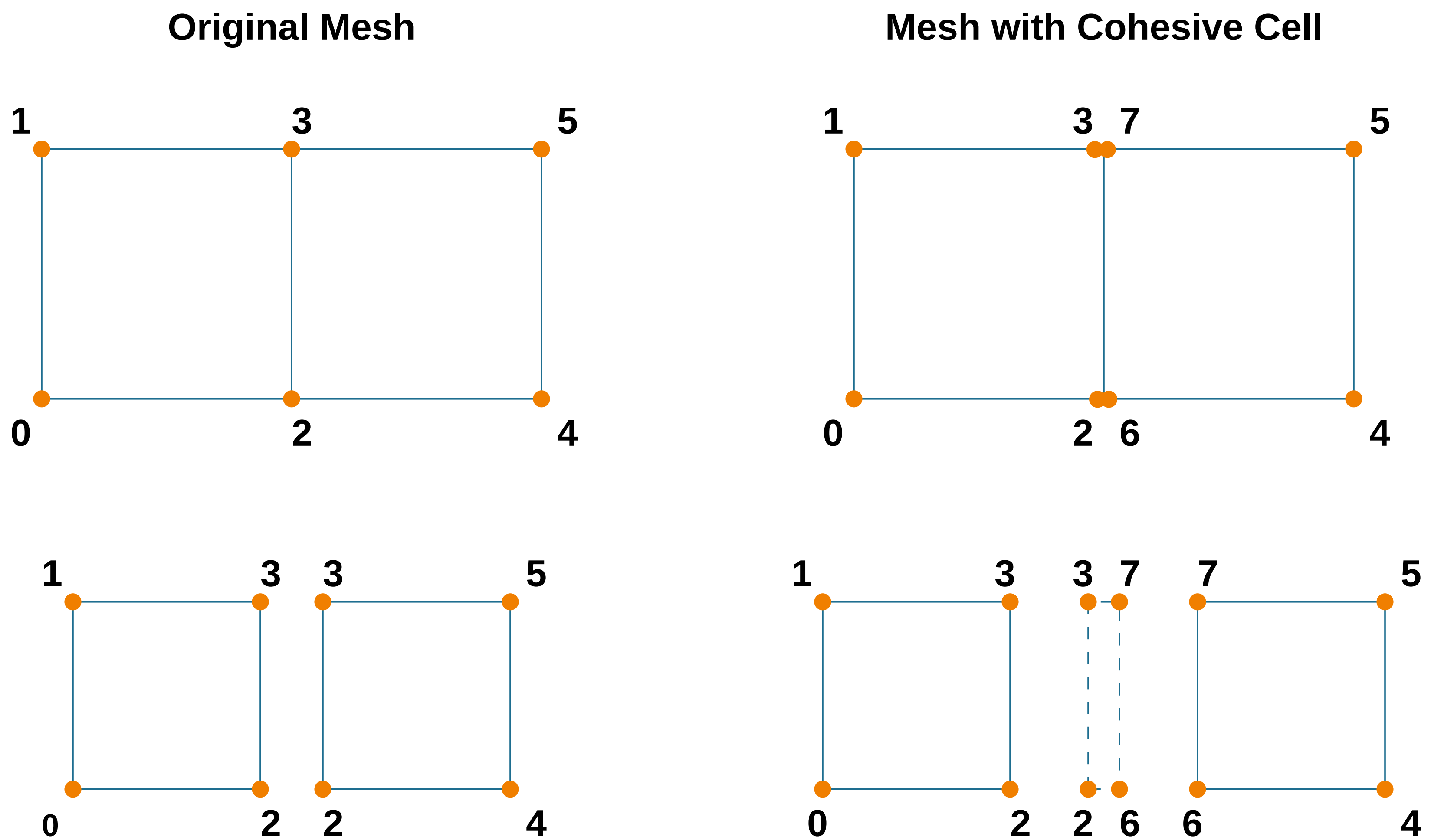
# Implementation: Finite-Element Data Structures

## 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




**Exploded view of meshes**

# Kinematic (prescribed) slip earthquake ruptures

## Use Lagrange multipliers to specify slip

- System without cohesive cells

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

- System with cohesive cells

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

- System with cohesive cells & conditioning

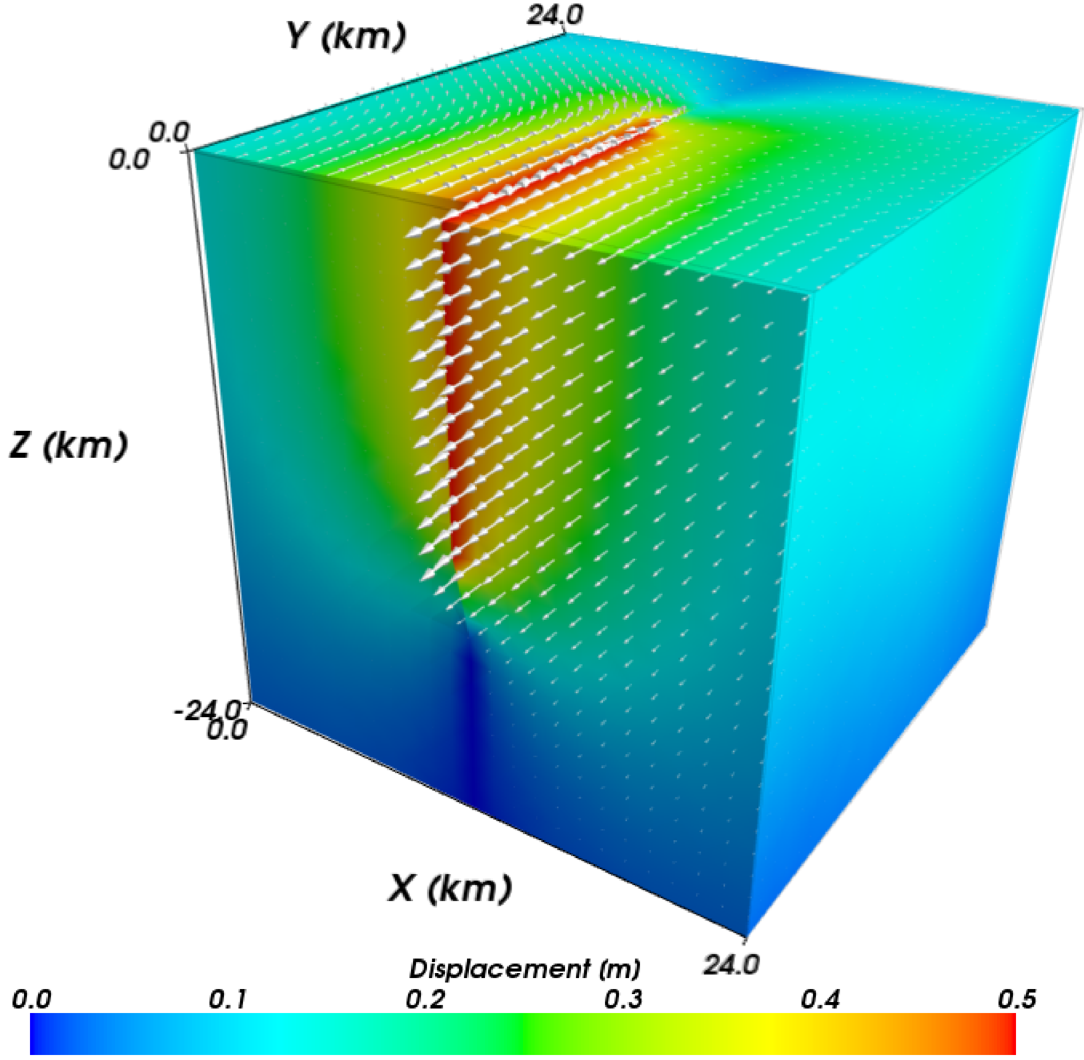
$$\begin{pmatrix} \underline{\mathsf{A}} & a\underline{\mathsf{C}}^T \\ \underline{\mathsf{C}} & 0 \end{pmatrix} \begin{pmatrix} \vec{u} \\ \frac{1}{a}\vec{L} \end{pmatrix} = \begin{pmatrix} \vec{b} \\ \vec{D} \end{pmatrix}$$

# 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
  - Conditioned matrix is non-symmetric
  - Mixes displacements and forces in solution

# Benchmarking PyLith

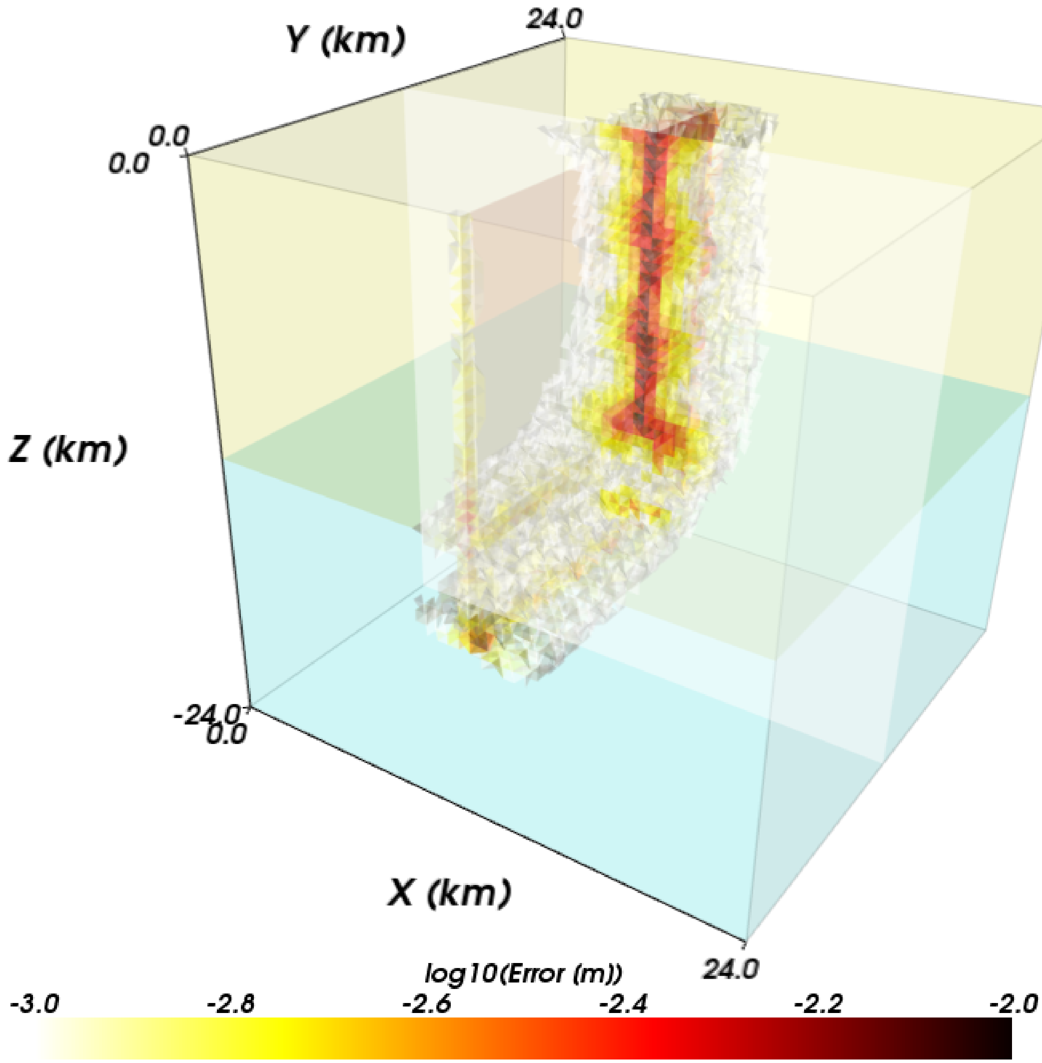
## Elastic solution for strike-slip benchmark




Computational Infrastructure for Geodynamics

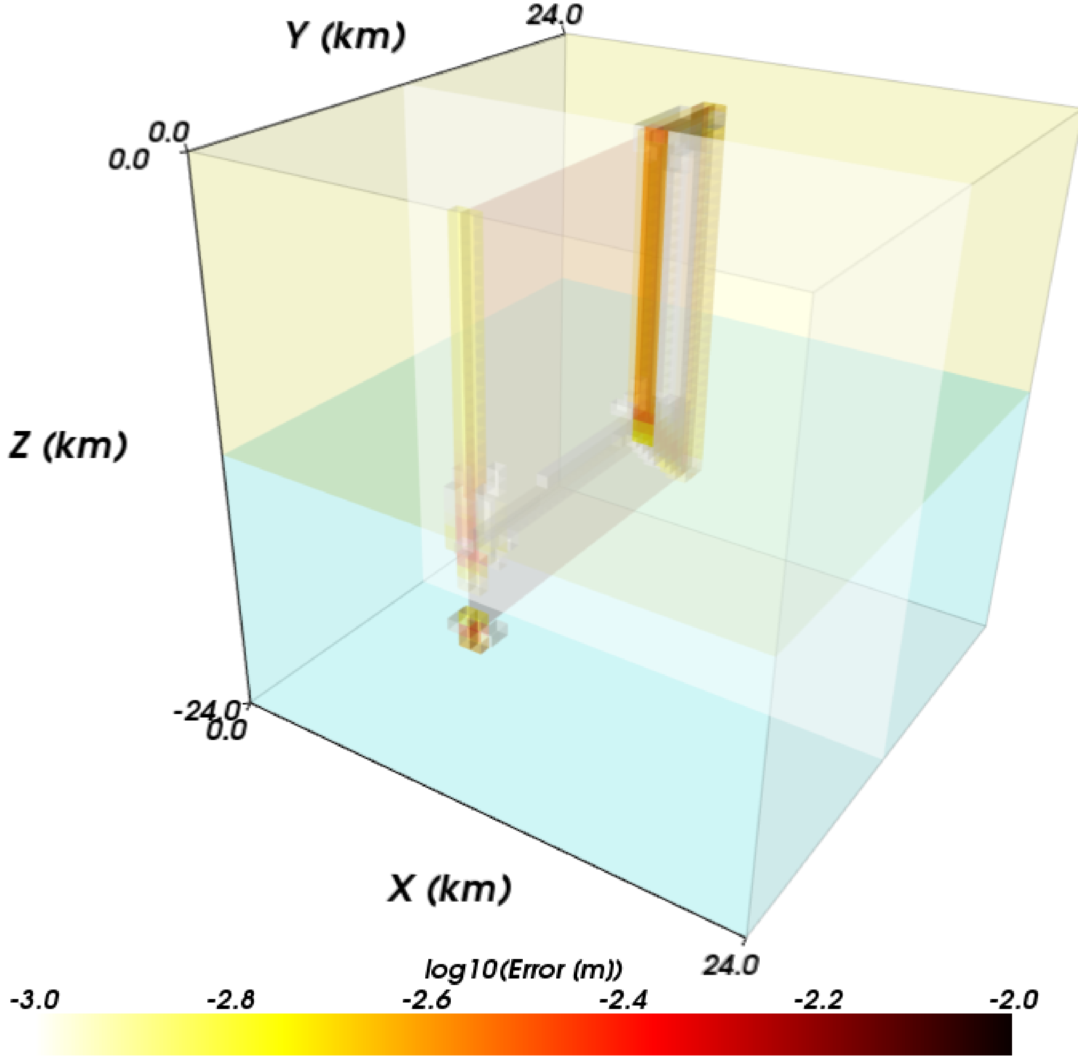
# Tet4 500m Mesh: Local Error

Error largest around edges of fault



# Hex8 500m Mesh: Local Error

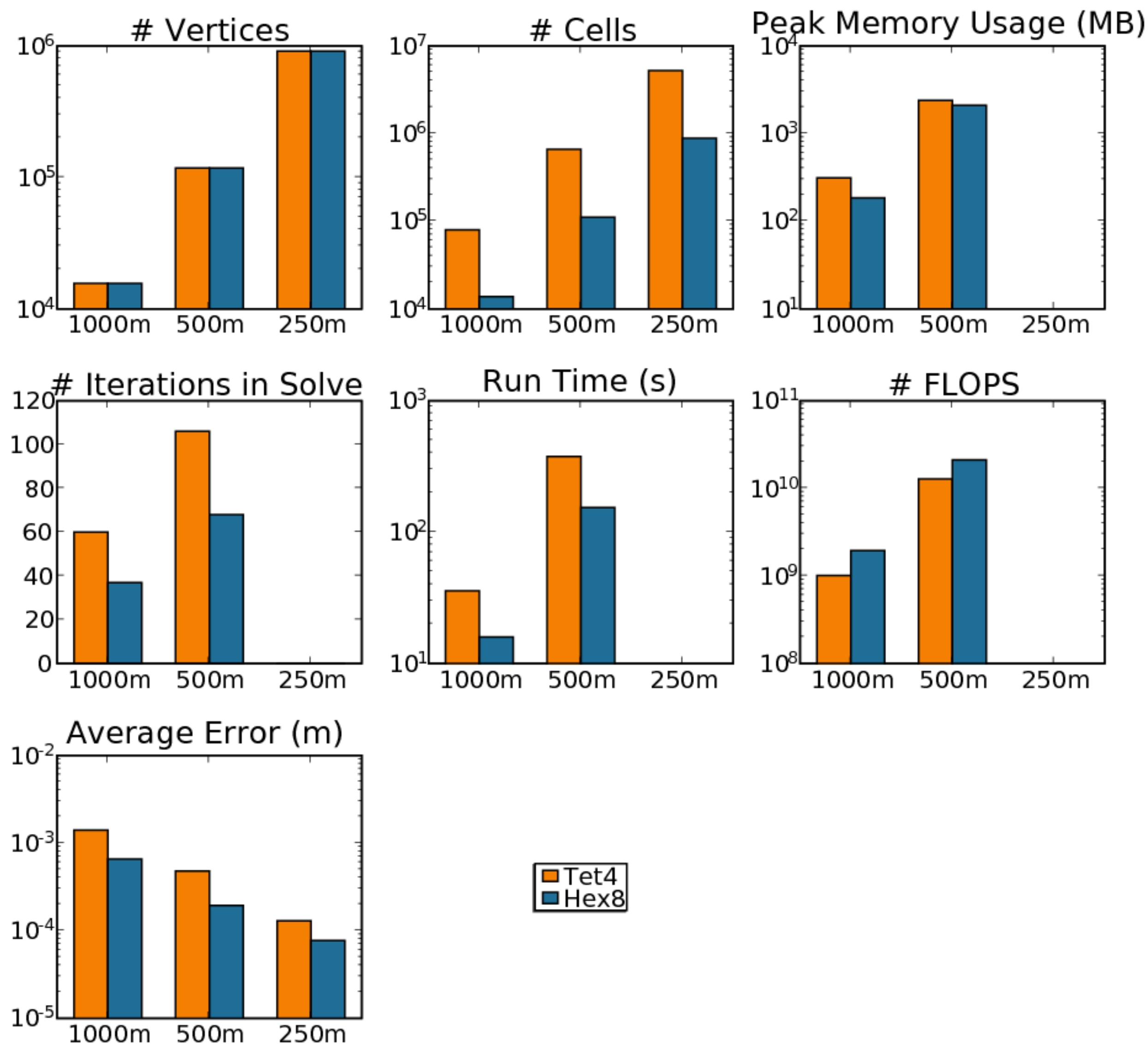
Error concentrated around change in slip gradient




Computational Infrastructure for Geodynamics

# Strike-Slip Benchmark: Performance Summary

## Hex8 cells outperform tet4 cells




Computational Infrastructure for Geodynamics

# Running PyLith

## Ingredients

- Simulation parameters
- Finite-element mesh
  - Mesh exported from LaGriT
  - Mesh exported from CUBIT
  - Mesh constructed by hand (PyLith mesh ASCII format)
- Spatial databases for boundary and fault conditions
  - Simple ASCII files specify spatial variation of parameters
  - Independent of discretization scheme and size

# Useful Tips/Tricks

- Command line arguments
  - `--help`
  - `--help-components`
  - `--help-properties`
  - `--petsc.start_in_debugger` (run in xterm)
  - `--nodes=N` (to run on N processors on local machine)
- PyLith User Manual
- CIG Short-Term Tectonics mailing list
  - `cig-short@geodynamics.org`
- CIG bug tracking system
  - `http://www.geodynamics.org/roundup`

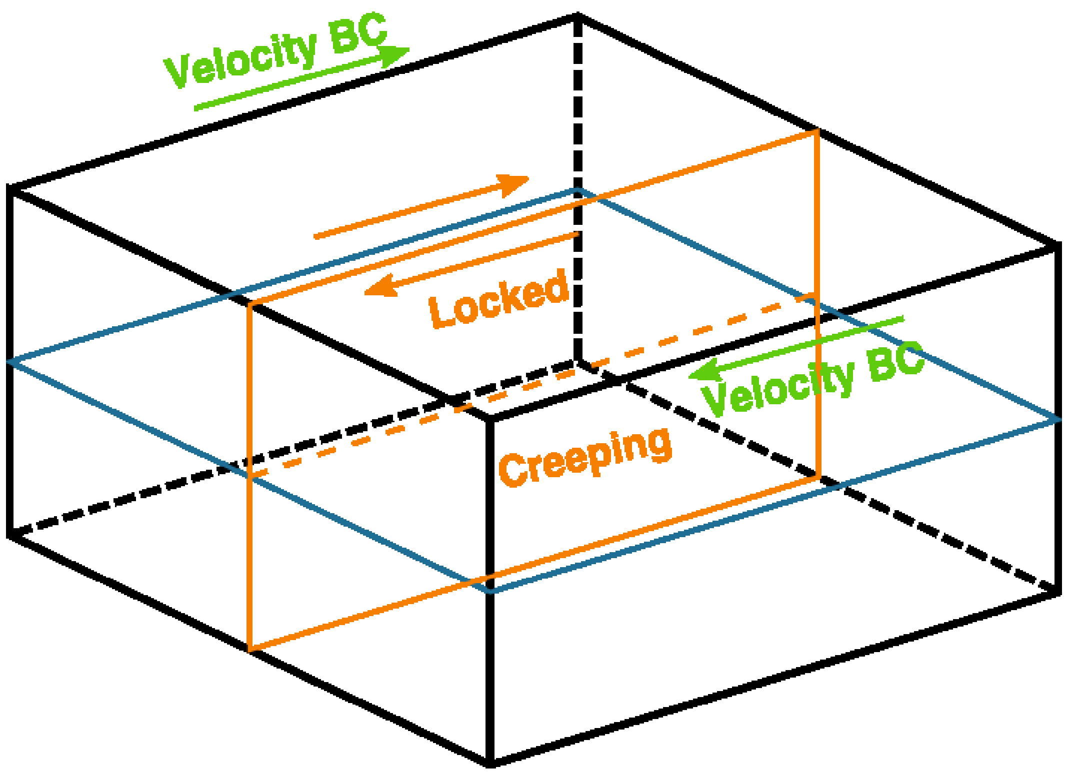
# PyLith is a Community Code

## 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

# Example: Slip on a Vertical Strike-Slip Fault

examples/3d/hex8



# Workflow for Example

1. Generate finite-element mesh using CUBIT (hex8 cells)

2. Create `.cfg` file with simulation parameters

3. Create spatial database files with parameters for boundary conditions and faults

4. Run PyLith

5. Visualize results with ParaView