PyLith

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

Computational Infrastructure for Geodynamics

June 14, 2010

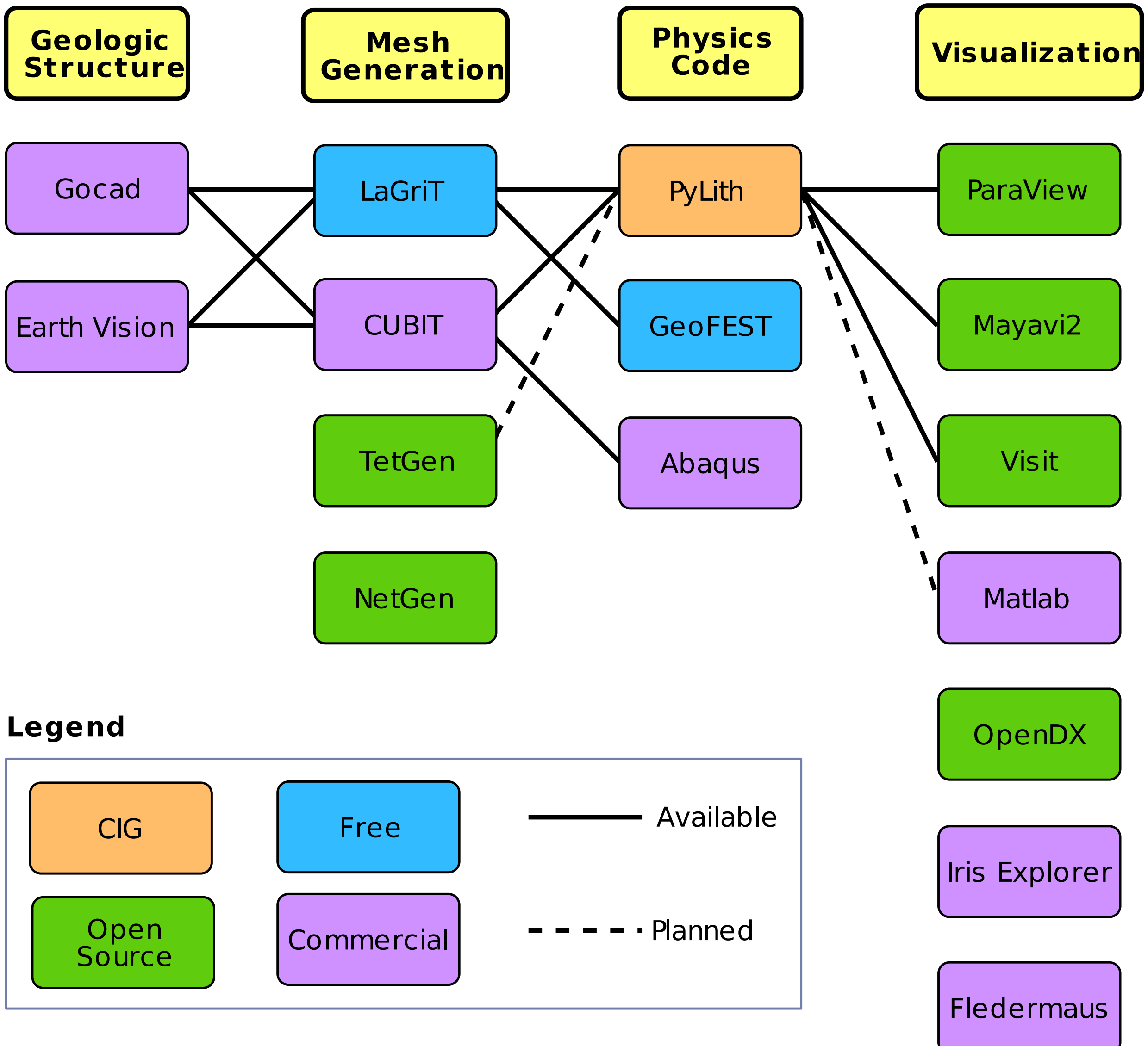
PyLith

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
  - Dynamic (spontaneous) earthquake ruptures
  - Local/regional ground-motion modeling

Crustal Deformation Modeling

Overview of workflow for typical research problem



Features in PyLith 1.5

Enhancements and new features in blue

- Time integration schemes and elasticity formulations
  - Implicit for quasi-static problems (neglect inertial terms)
    - Infinitesimal strains
    - Small strains
  - Explicit for dynamic problems
    - Infinitesimal strains with sparse system Jacobian
    - Infinitesimal strains with lumped system Jacobian
    - Small strains with sparse system Jacobian
- 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)
  - Linear Maxwell viscoelastic model (2-D)
  - Drucker-Prager elastoplastic model (3-D)

Features in PyLith 1.5 (cont.)

Enhancements and new features in blue

- 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
  - Dynamic (friction) fault interfaces
  - Time-dependent point forces
  - Gravitational body forces
- Fault constitutive models
  - Static friction
  - Linear slip-weakening
  - Dieterich-Ruina rate and state friction w/ageing law

Features in PyLith 1.5 (cont.)

Enhancements and new features in blue

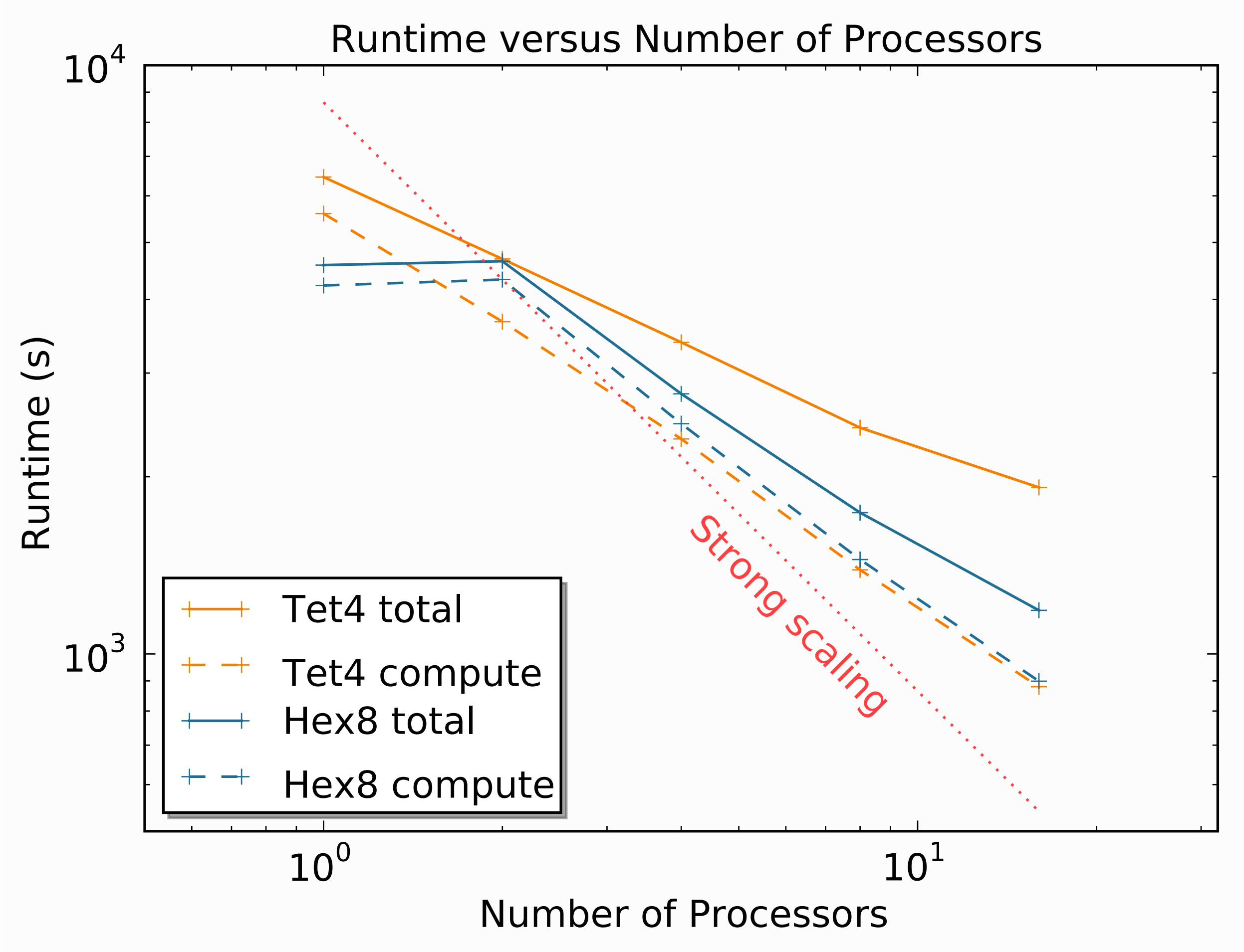
- Automatic and user-controlled time stepping
- Ability to specify initial stress state
- Importing meshes
  - LaGriT: GMV/Pset
  - CUBIT: Exodus II
  - ASCII: PyLith mesh ASCII format (intended for toy problems only)
- Output: VTK files
  - Solution over volume
  - Solution over surface boundary
  - 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 1.5: Under-the-hood Improvements

- Additional cleanup of C++ code
- Optimization of several modules
  - Mesh distribution among processors
  - Integration of elasticity terms
- Ability to use algebraic multigrid preconditioners

PyLith 1.5 Performance

PyLith 1.5 is up to ∼??% faster than PyLith 1.4



Computational Infrastructure for Geodynamics

PyLith 1.x: Planned Releases

Current productivity is about 2 feature releases per year

- PyLith 1.6: anticipate release in late 2010 or early 2011
  - Additional fault constitutive models
  - Additional optimization
- 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 and adjoint methods
  - Easier discretization (ability to use structured meshes, meshless methods)

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

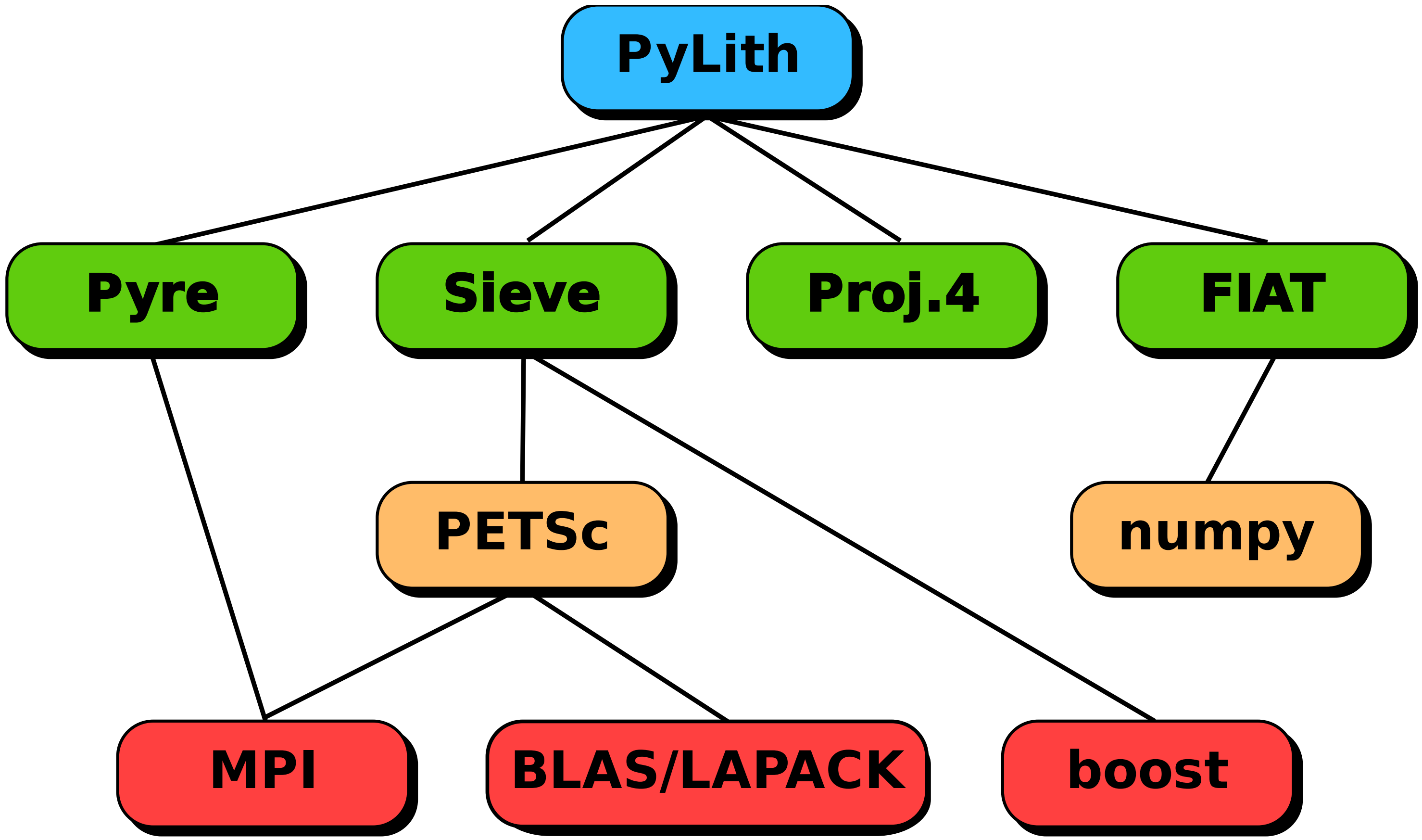
PyLith is a Community Code

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

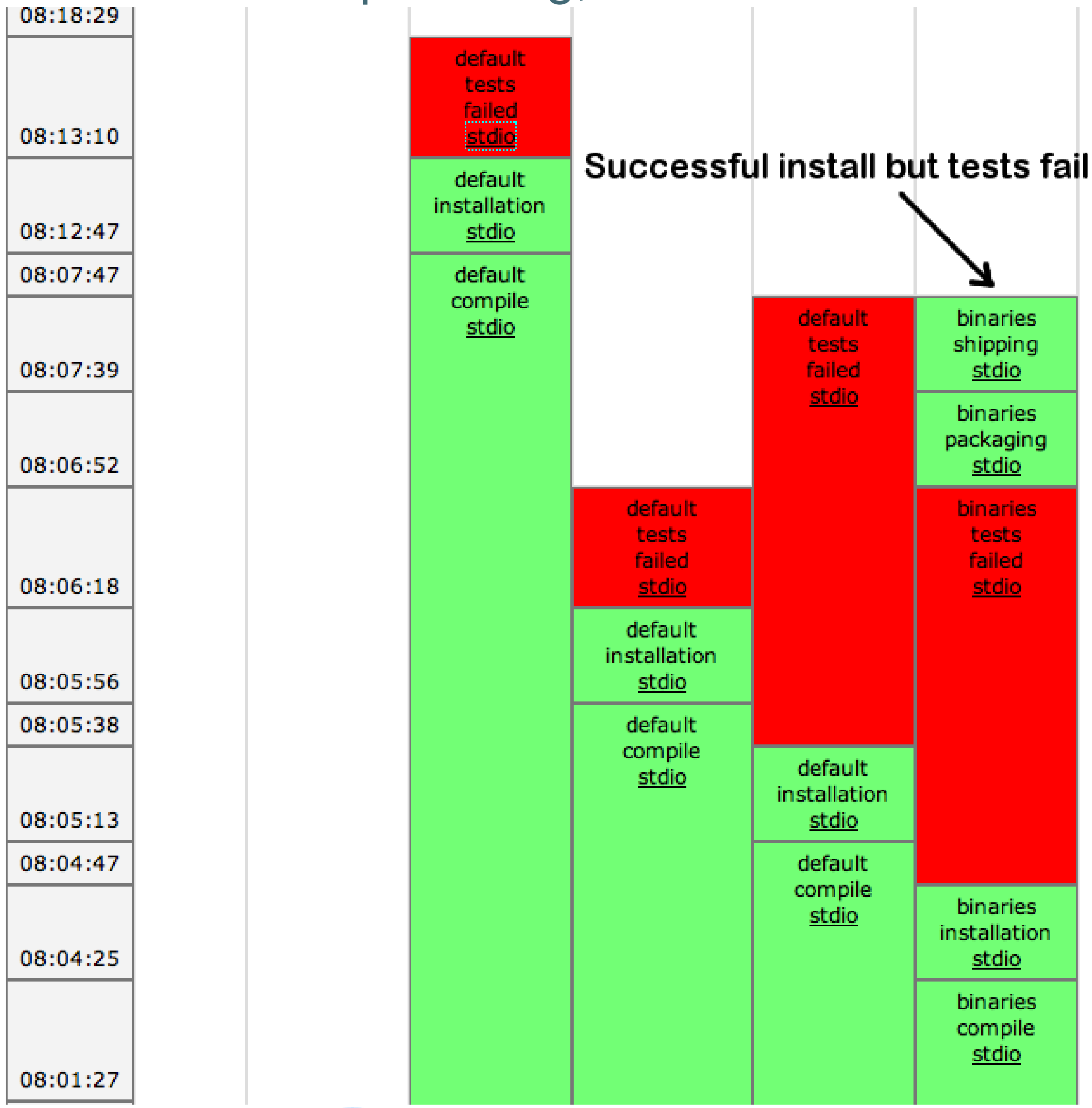
PyLith Design

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

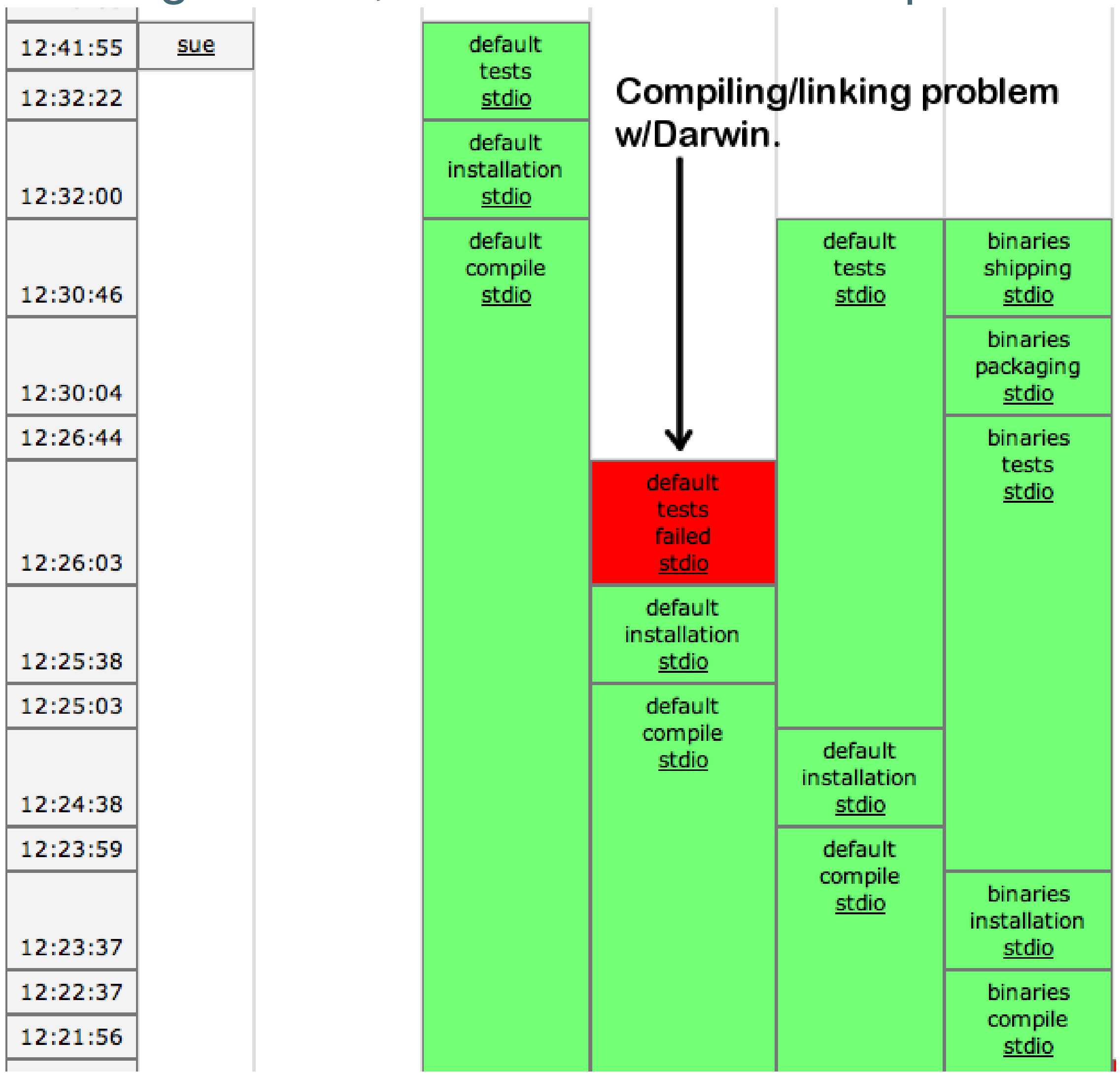
Test written to expose bug, buildbot shows tests fail



Computational Infrastructure for Geodynamics

Automated Building and Testing

Bug is fixed, buildbot shows tests pass



Computational Infrastructure for Geodynamics

Implementation: Finite-Element Data Structures

Use Sieve for storage and manipulating mesh information

- PyLith makes only a few MPI calls

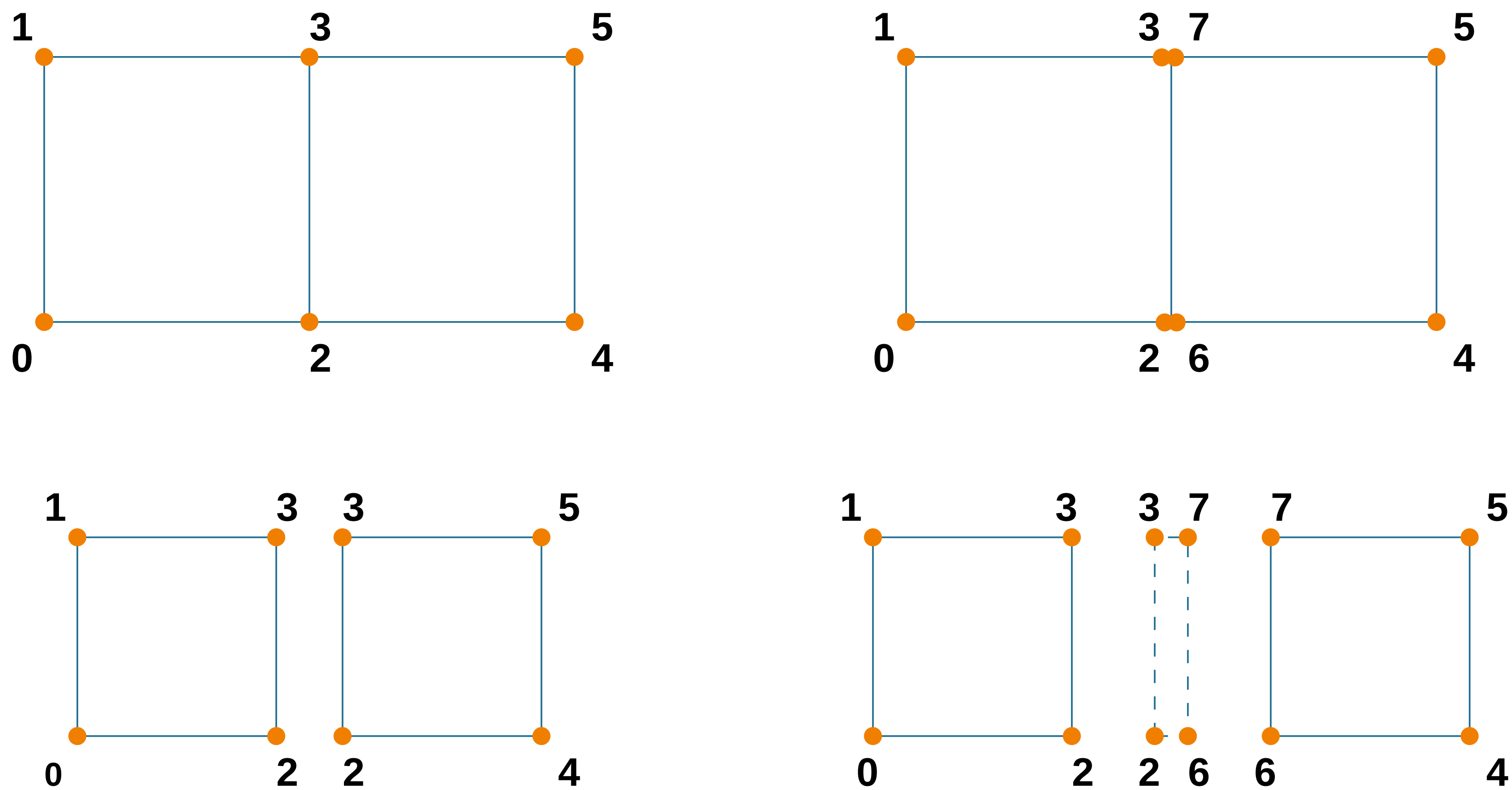
- Data structures are independent of basis functions and reference cells
  - Same code for many cell shapes and types
  - Physics implementation limits code, not data structures

- 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

Implementation: Fault Interfaces

Use Lagrange multipliers to specify slip/tractions

- System without cohesive cells

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

- System with cohesive cells

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

- Kinematic slip: specify $\vec{d}(t)$.
- Dynamic slip: fault constitutive model places bounds on $\vec{l}(t)$

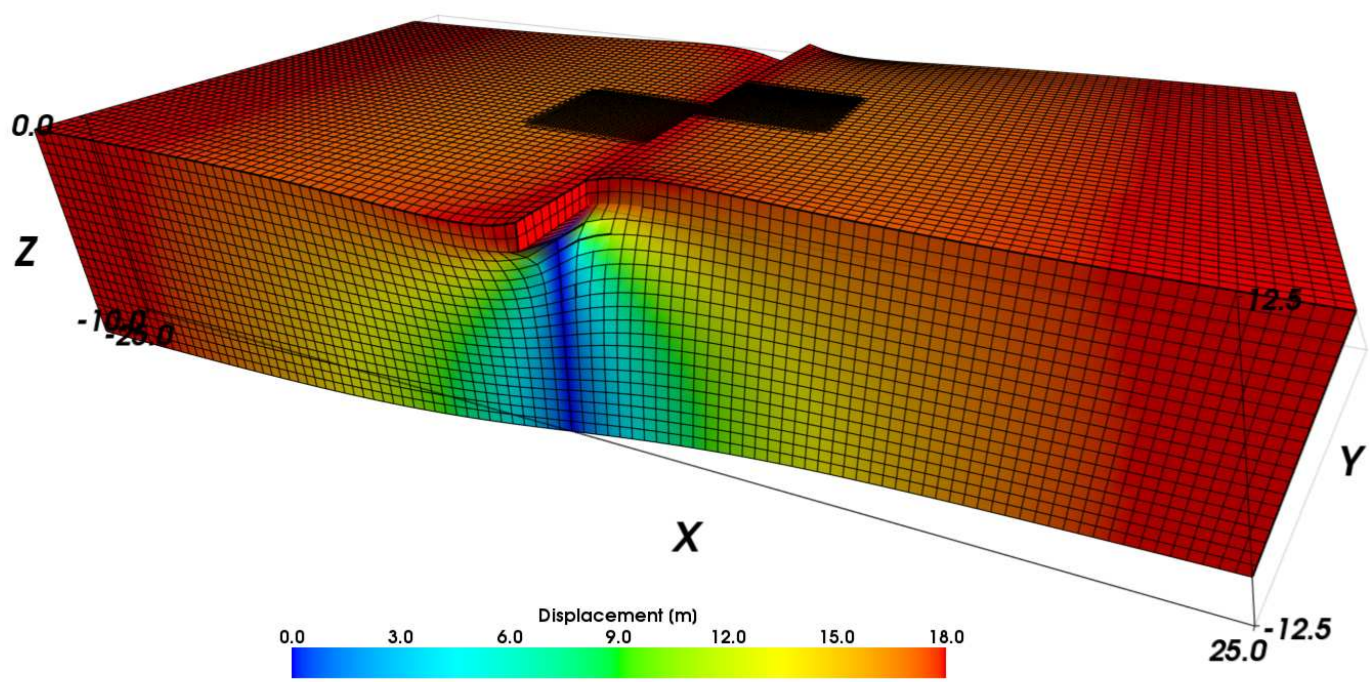
Implementing Fault Slip with Lagrange multipliers

- 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)

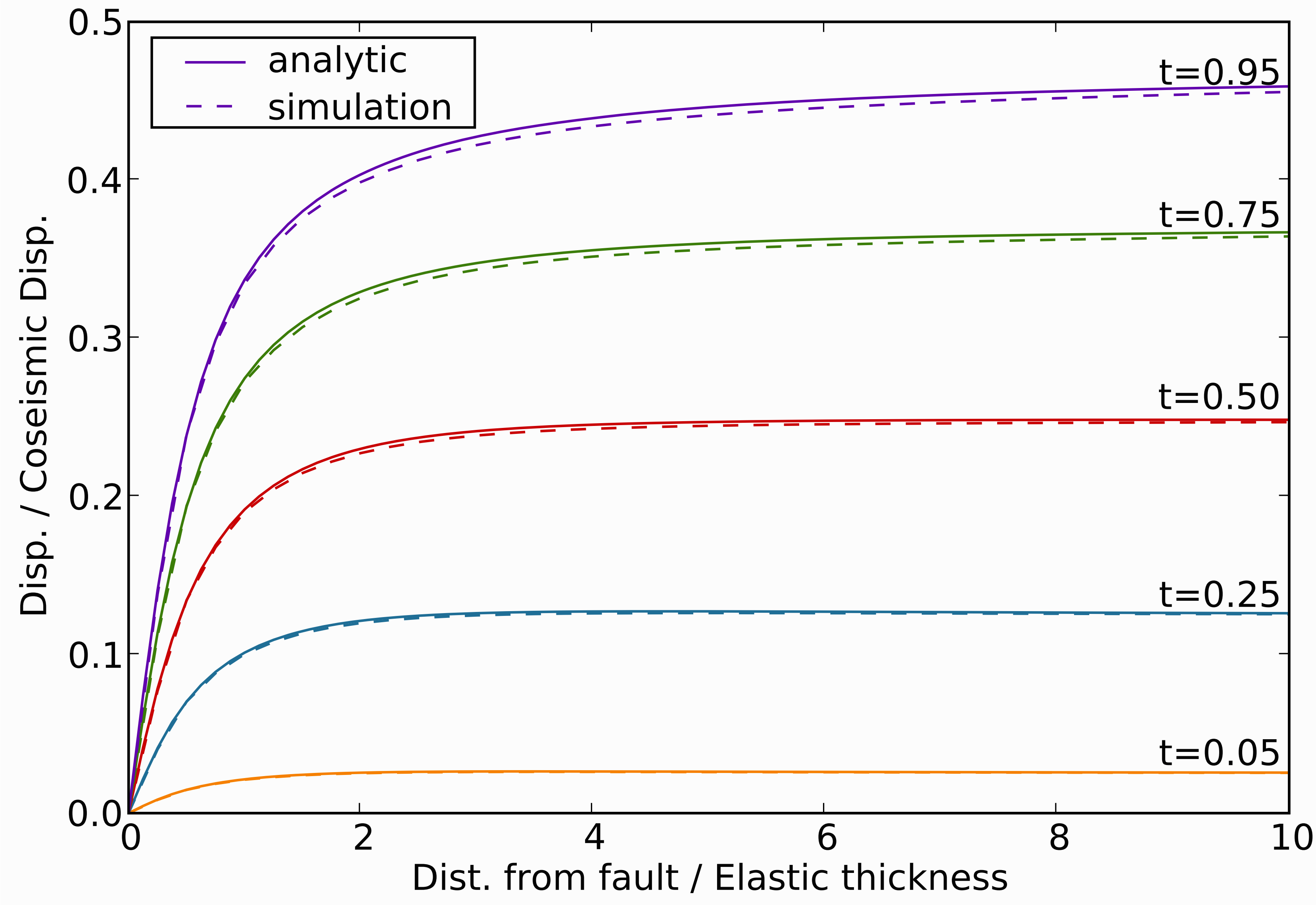
- Repeated rupture on a vertical, strike-slip fault
- Elastic layer over a linear Maxwell viscoelastic half-space
- Steady creep over bottom half of the elastic layer



Computational Infrastructure for Geodynamics

Benchmarking PyLith

Simulation closely matches analytical solution during 10th eq cycle



Computational Infrastructure for Geodynamics

Benchmarking PyLith

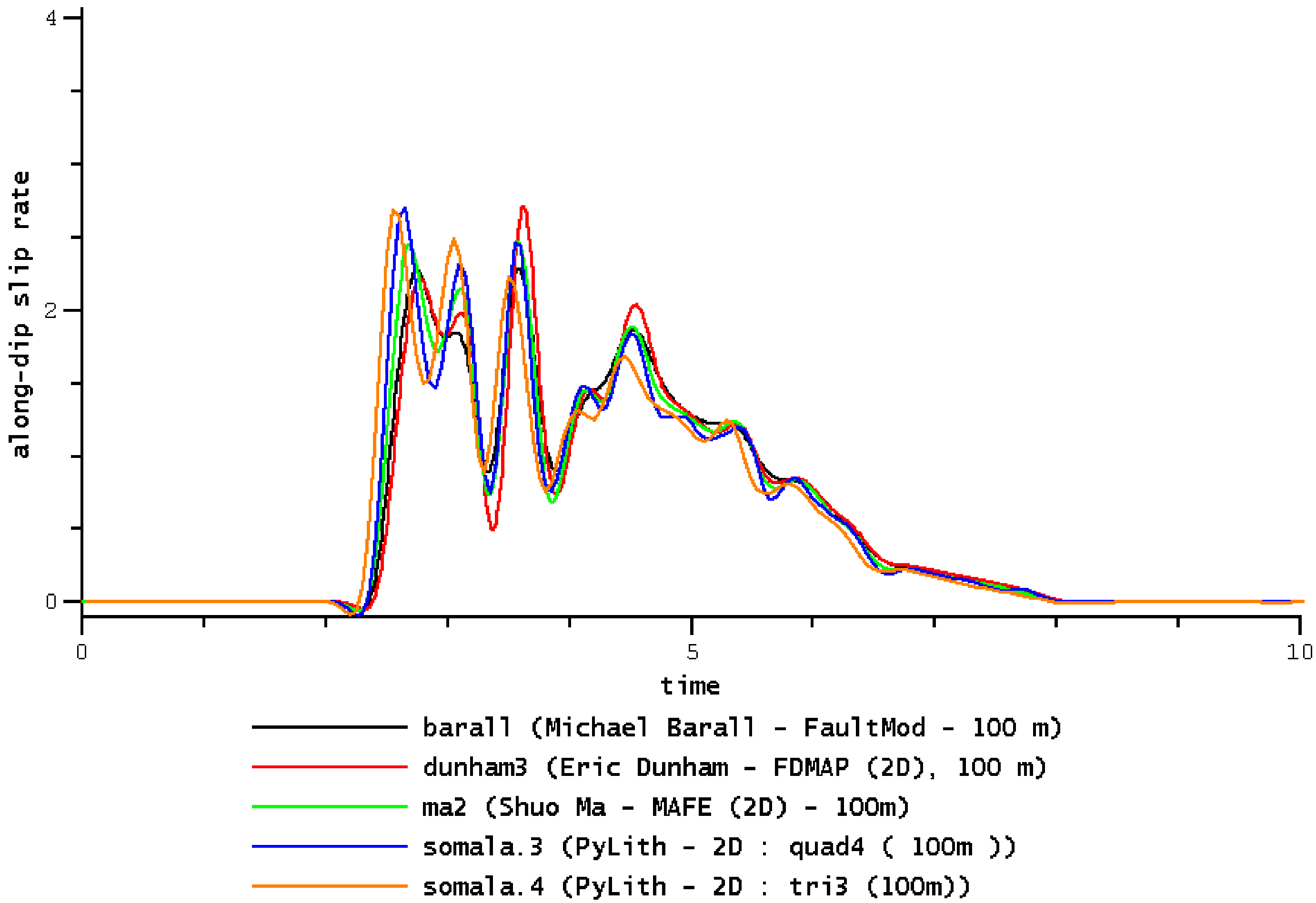
SCEC Dynamic Rupture Verification Benchmarks

- Rupture on a 60 degree normal fault
- Normal tractions equal to overburden pressure
- Shear tractions proportional to normal tractions
- Linear slip-weakening friction

ADD FIGURE HERE

Benchmarking PyLith

Close agreement with other dynamic rupture codes



Computational Infrastructure for Geodynamics