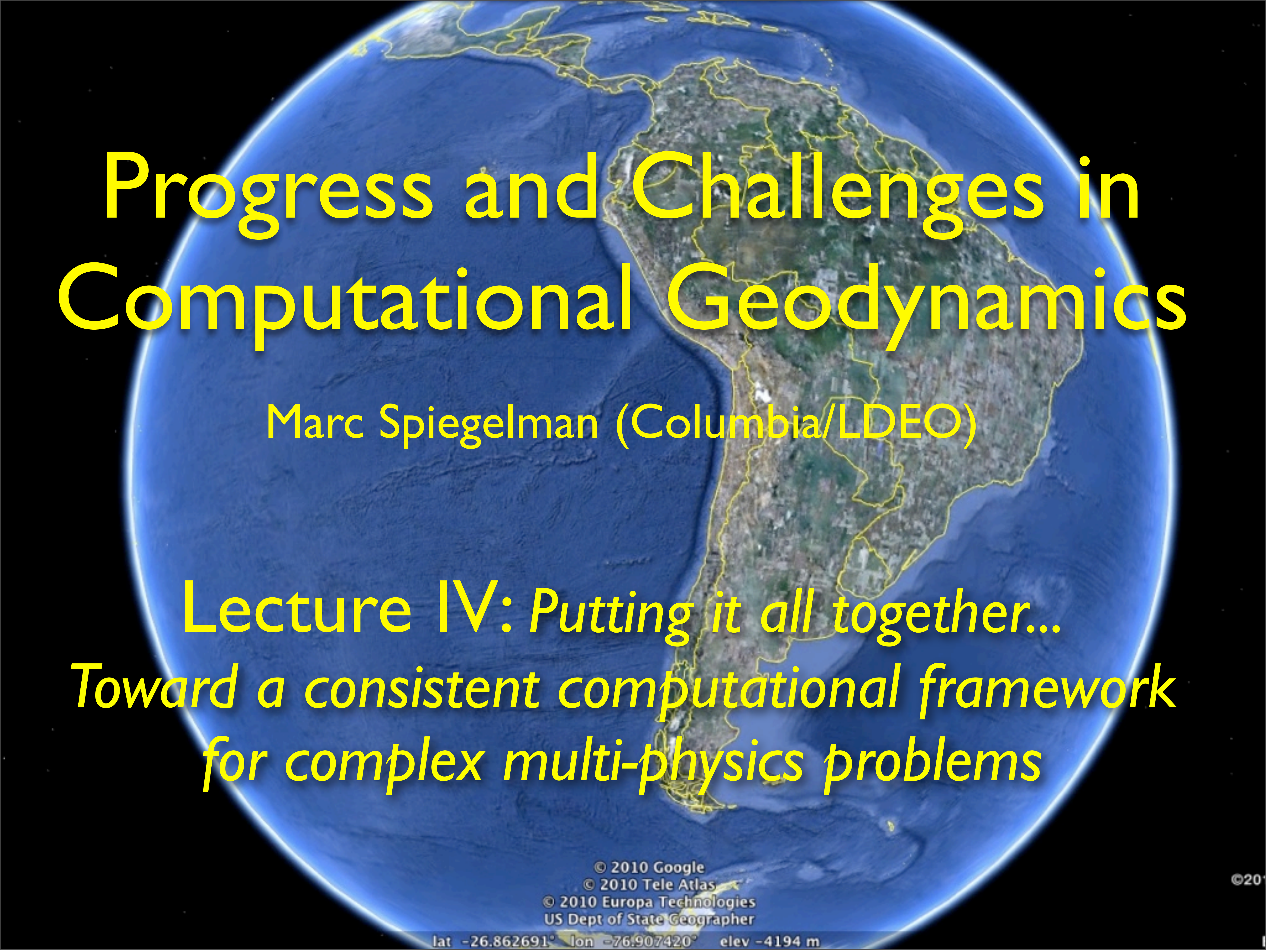
Progress and Challenges in Computational Geodynamics

Marc Spiegelman (Columbia/LDEO)

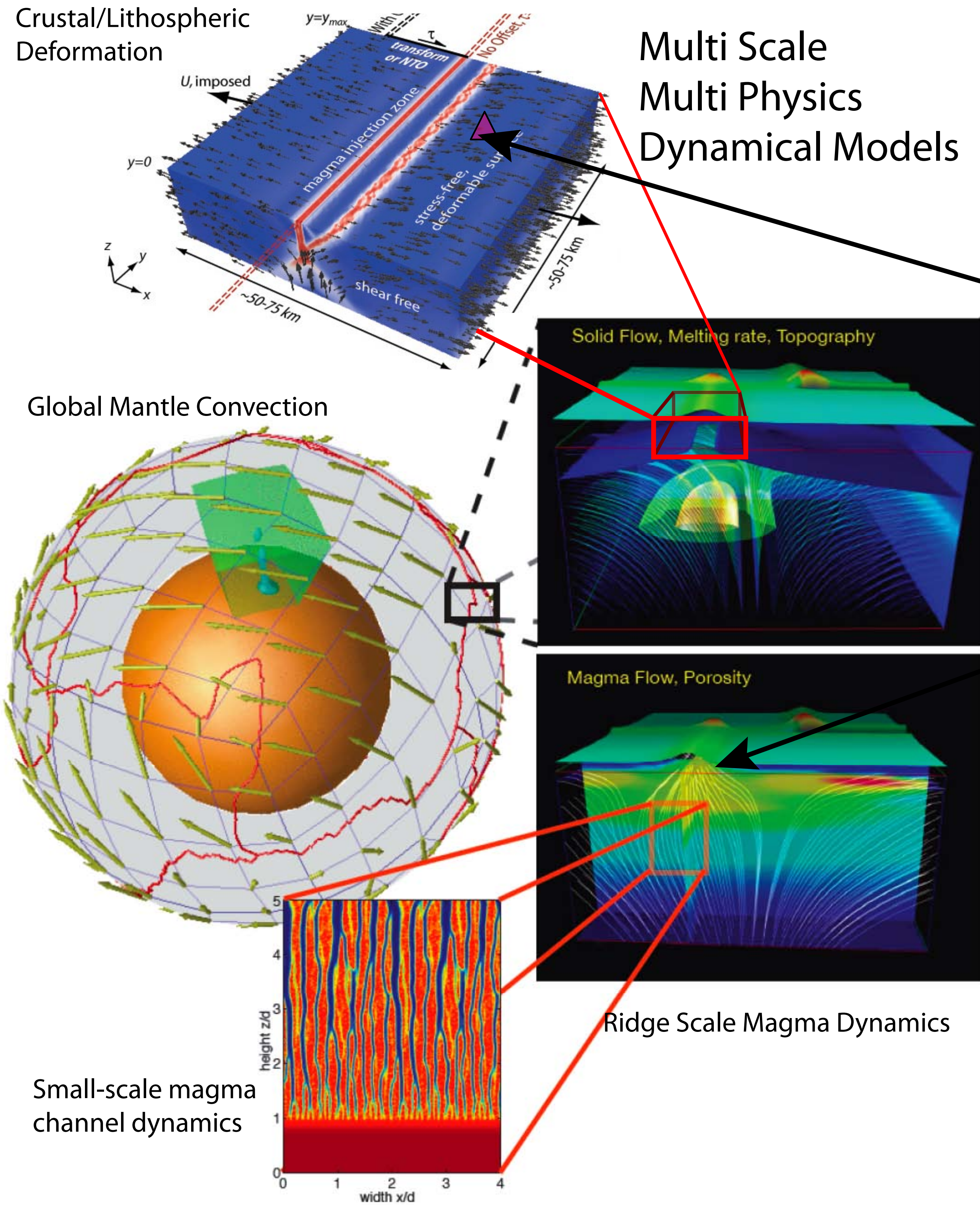
Lecture IV: *Putting it all together...*
Toward a consistent computational framework for complex multi-physics problems

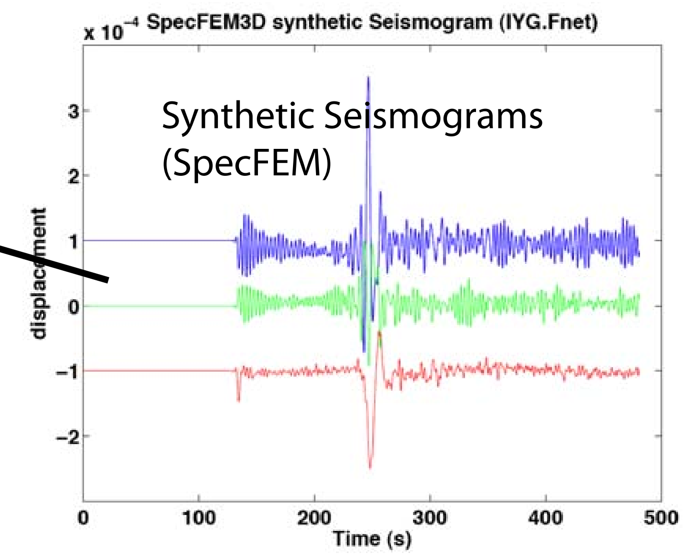


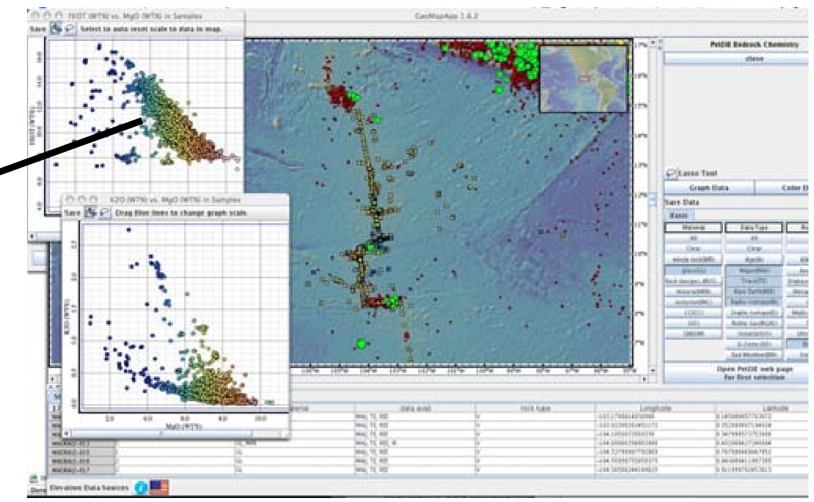
Outline

- Motivation: multi-physics issues arising from Solid Earth Geodynamics
- Some Philosophy
- Advanced software for PDE based modeling (FEniCS/PETSc)
- A model multi-physics problem: thermal convection
  - Discretization
  - Physics-based Block-Preconditioners
  - Results: Convergence/Performance
- The Future...

Multi-Scale Multi-physics problems in Solid Earth Geodynamics





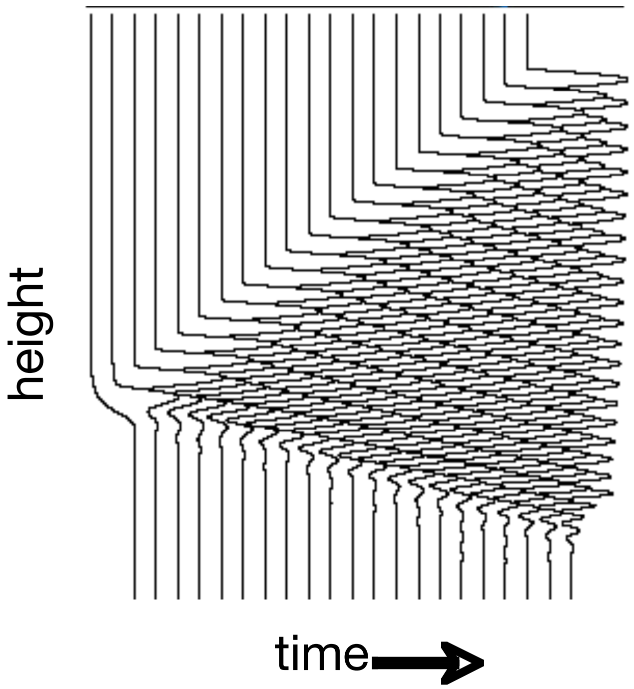


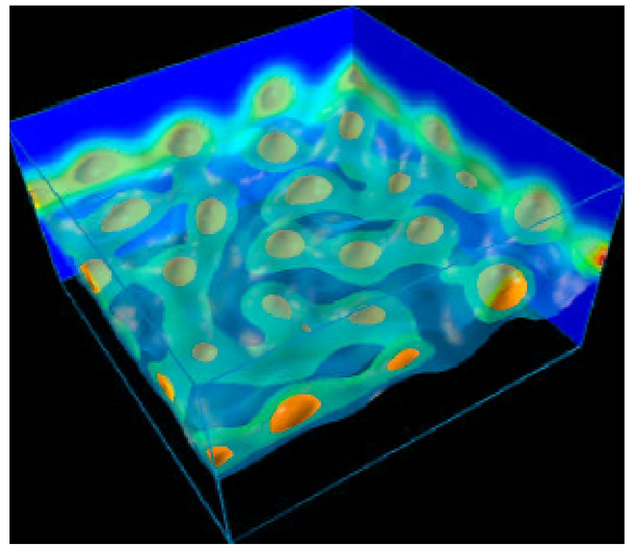
Geochemistry/Petrology (PetDB/GeoMapApp)

Observational Constraints

Basic Behavior of Magma PDEs

Nonlinear Porosity Waves

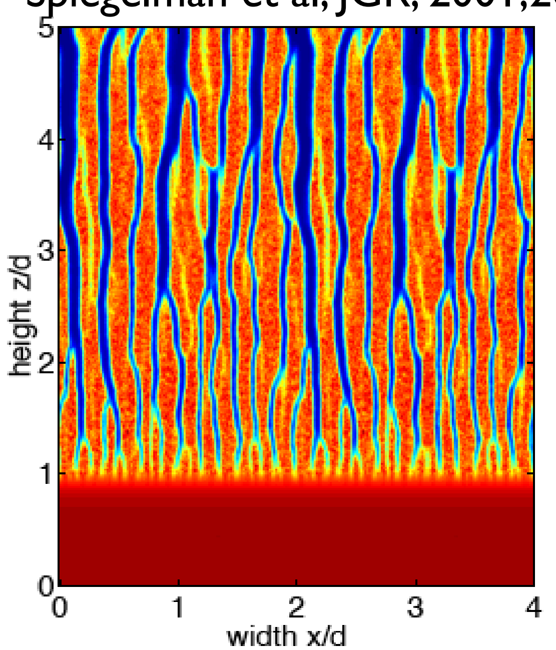


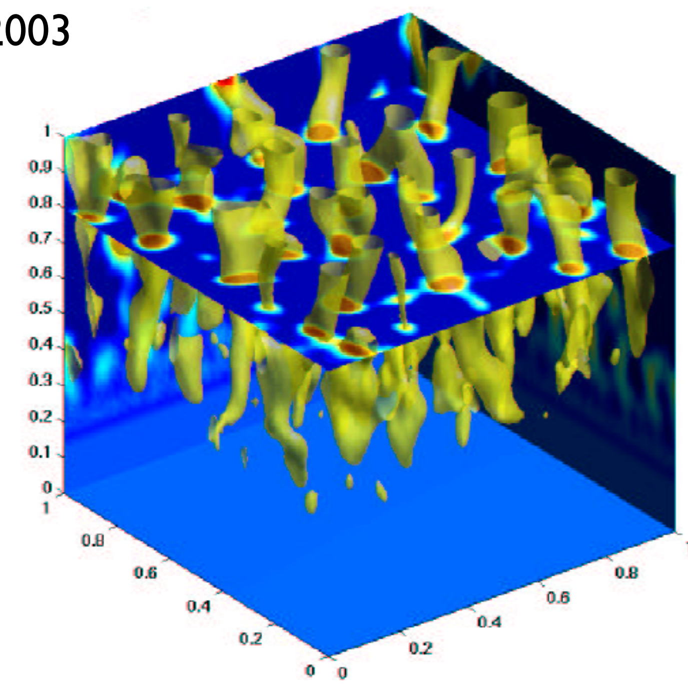


Spiegelman, JFM 93, W&S, GRL 94

Reactive Channel formation

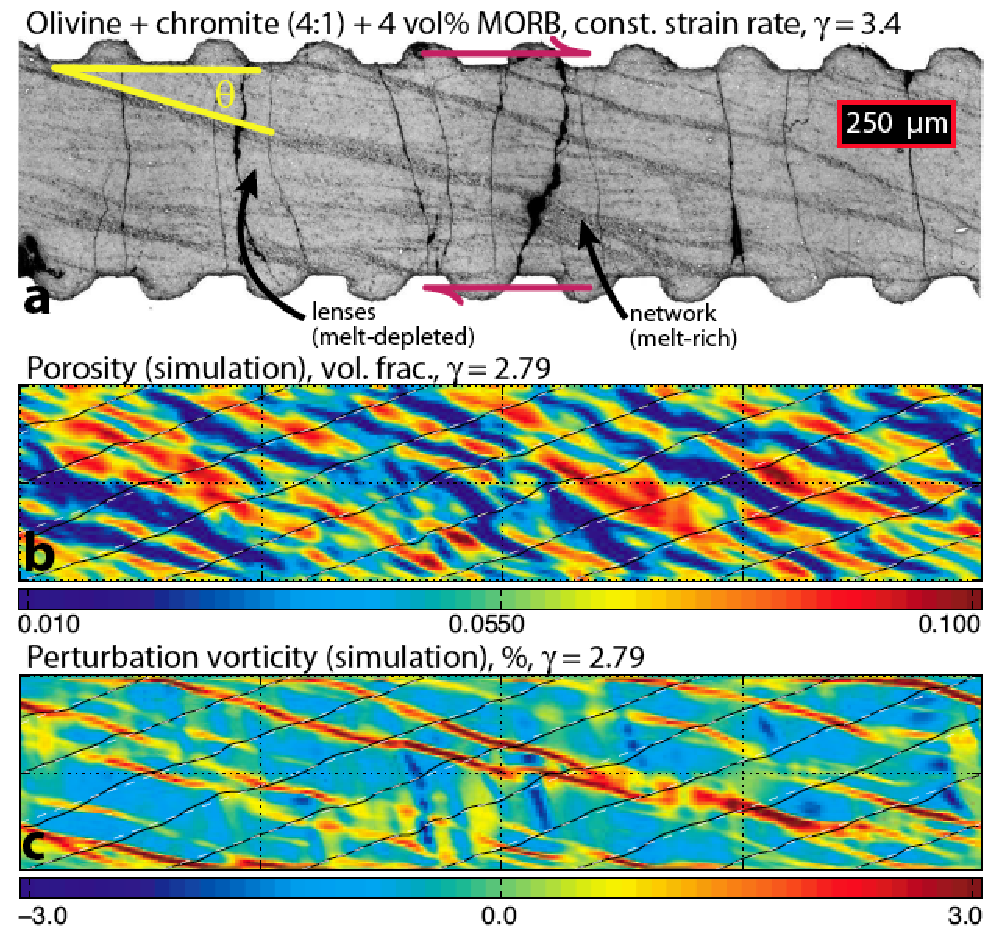
Spiegelman et al, JGR, 2001,2003





Mechanical Melt Localization

(Katz et al., 2006, Nature)

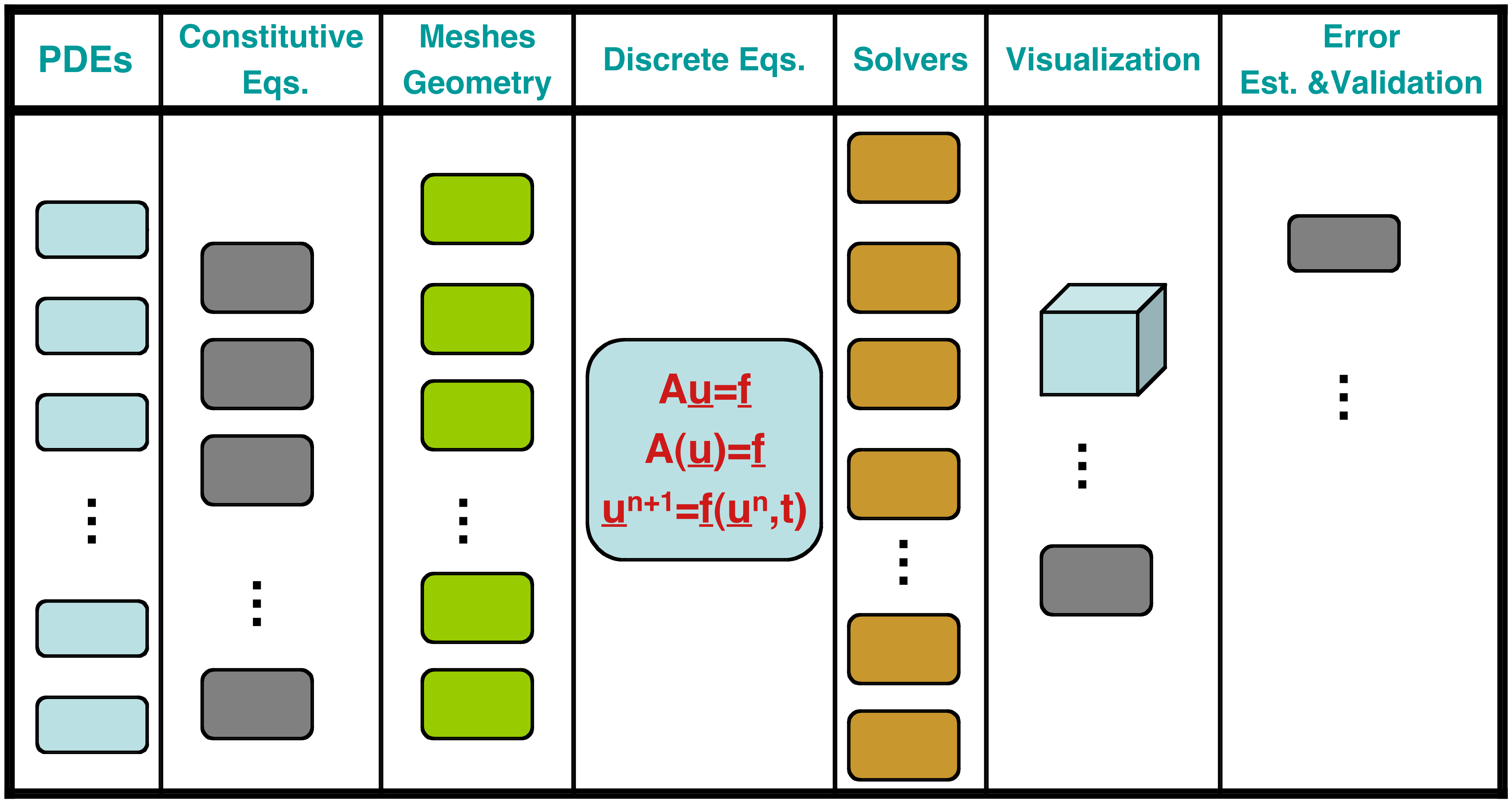


- Spontaneous formation of coherent structures at multiple scales
- Small changes in assumptions of coupling can radically change the physics
- Many questions remain about interactions and implications of multiple instabilities

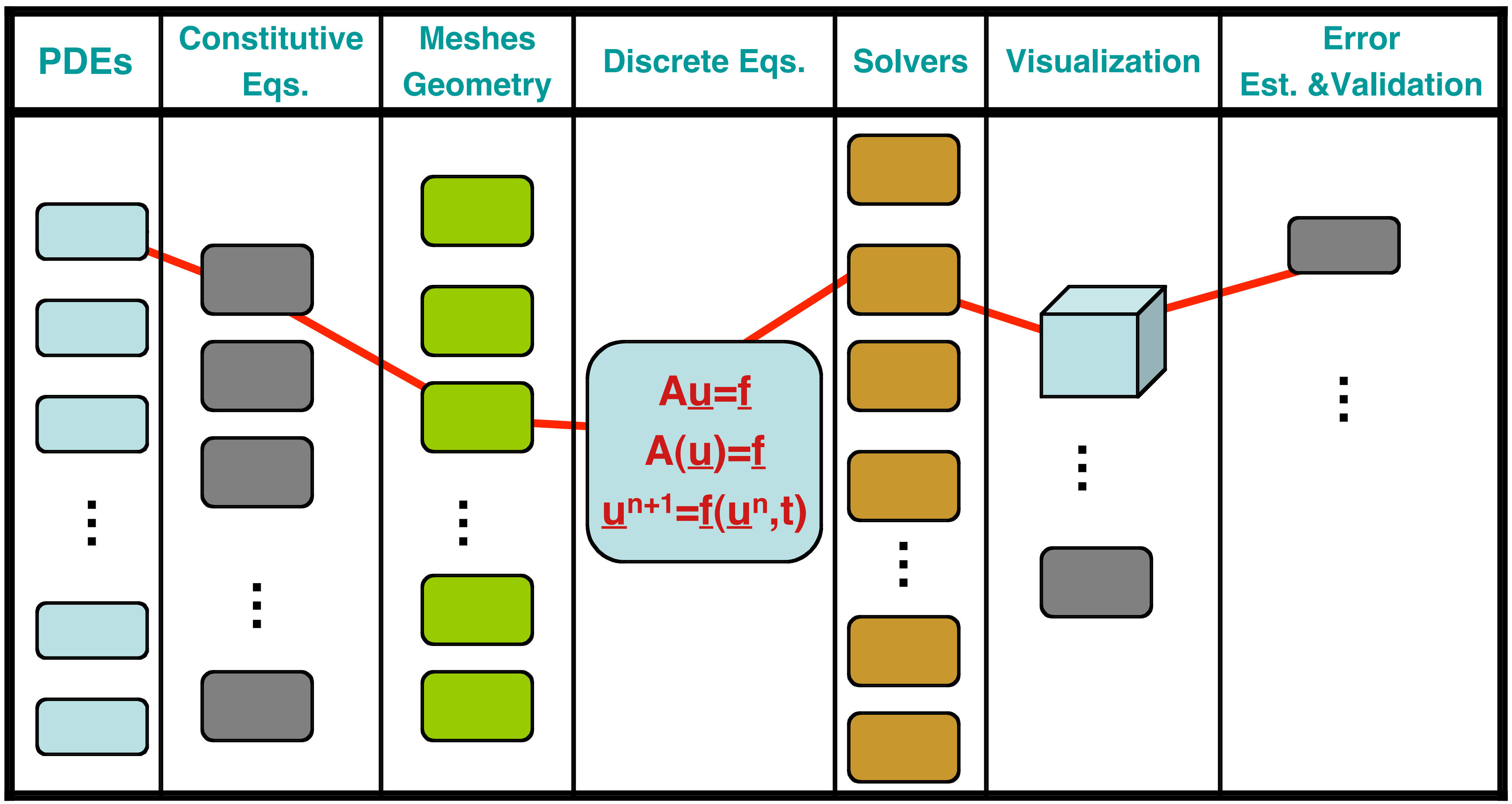
Some Questions

- How do we manage the complexity of Multi-Physics, Multi-Scale problems?
- How do we sensibly explore model space and decide what is important and what can be ignored, homogenized etc?
- How do we write Multi-Physics software that
  - allows reuse, interoperability?
  - takes advantage of advanced hardware?
  - Gives flexibility to the end user to make their own decisions, and do their own science?

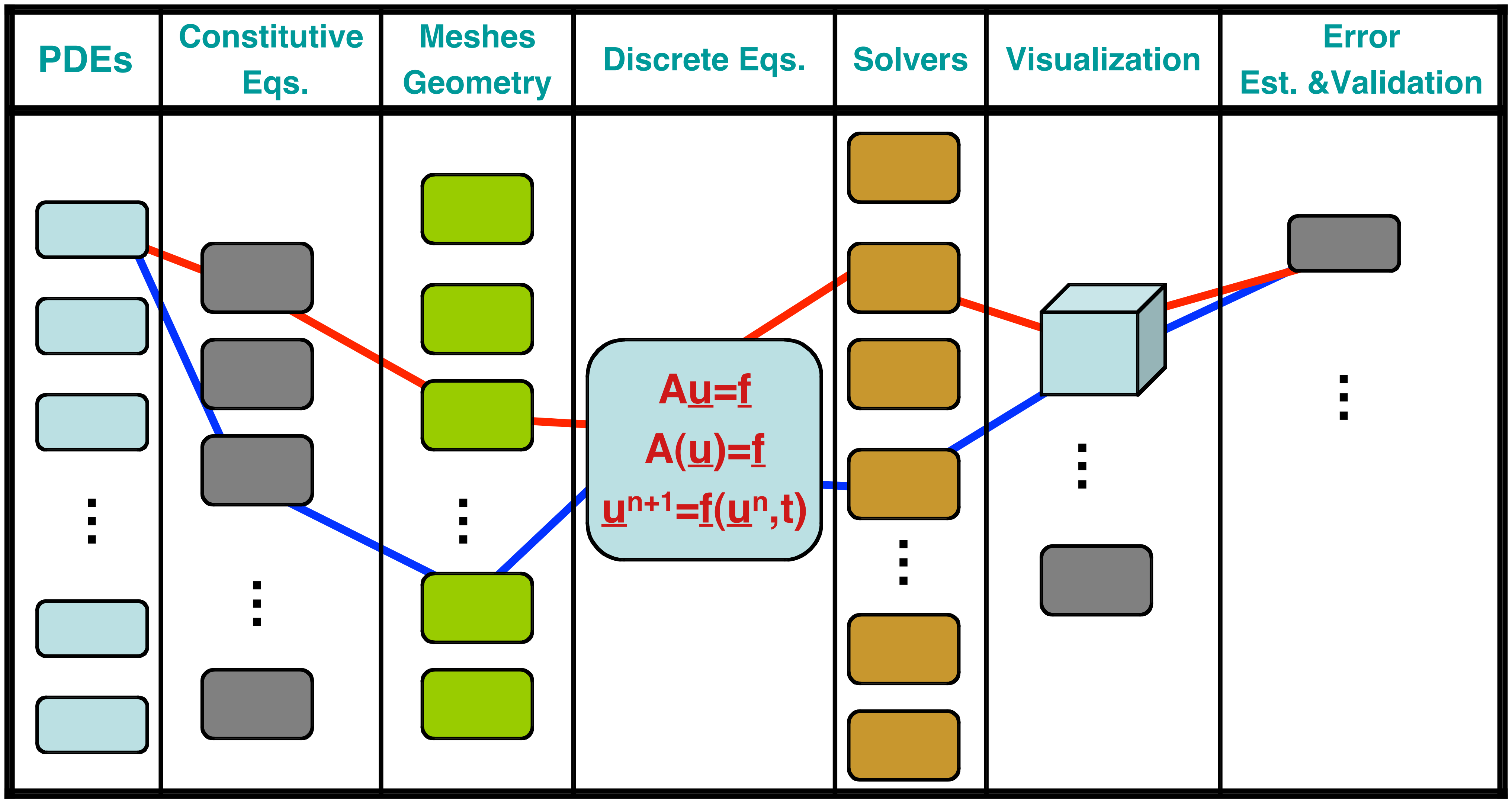
The Structure of Computational Problems
(numerical PDE's)



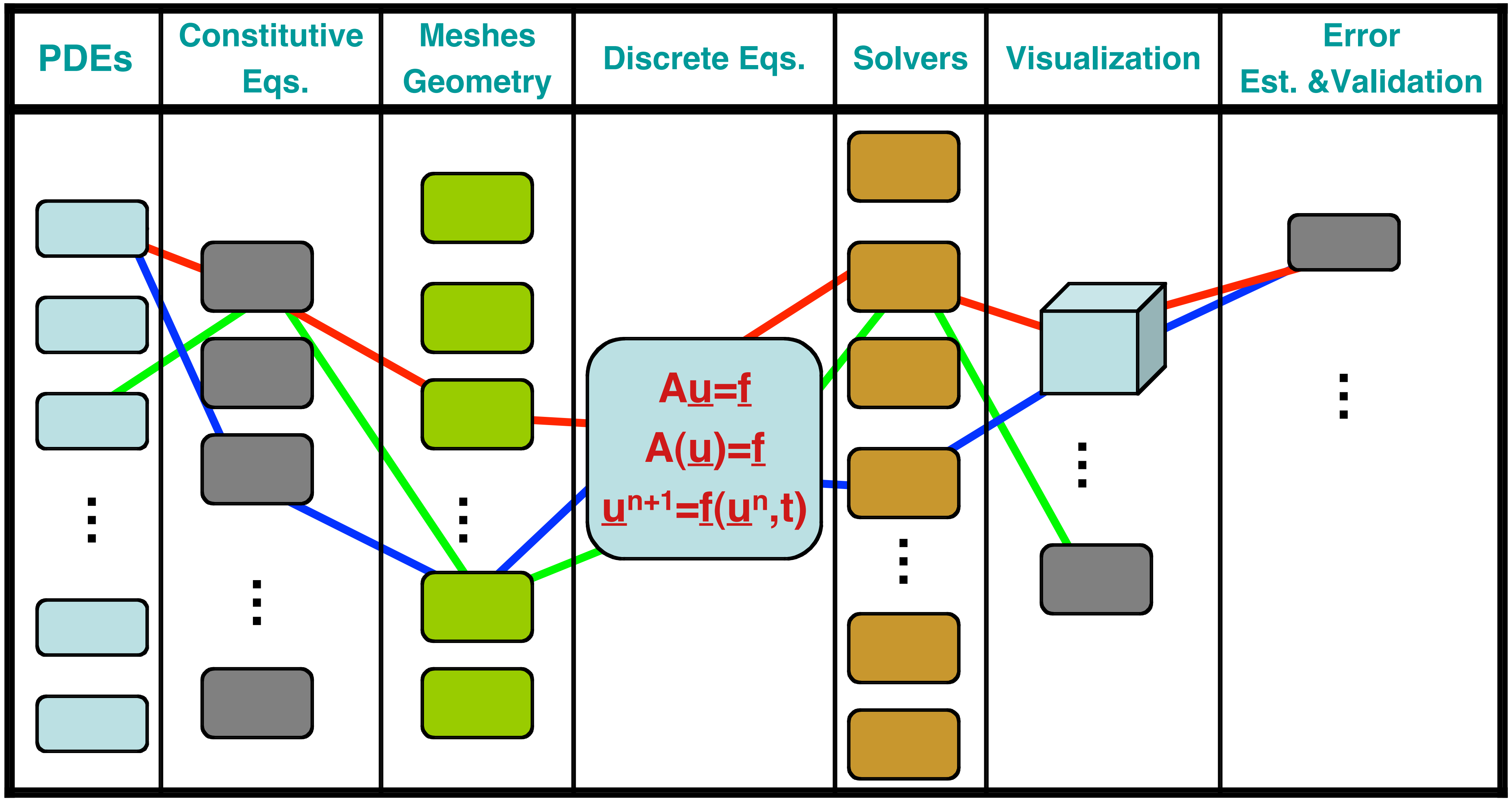
The Structure of Computational Problems
(numerical PDE's)



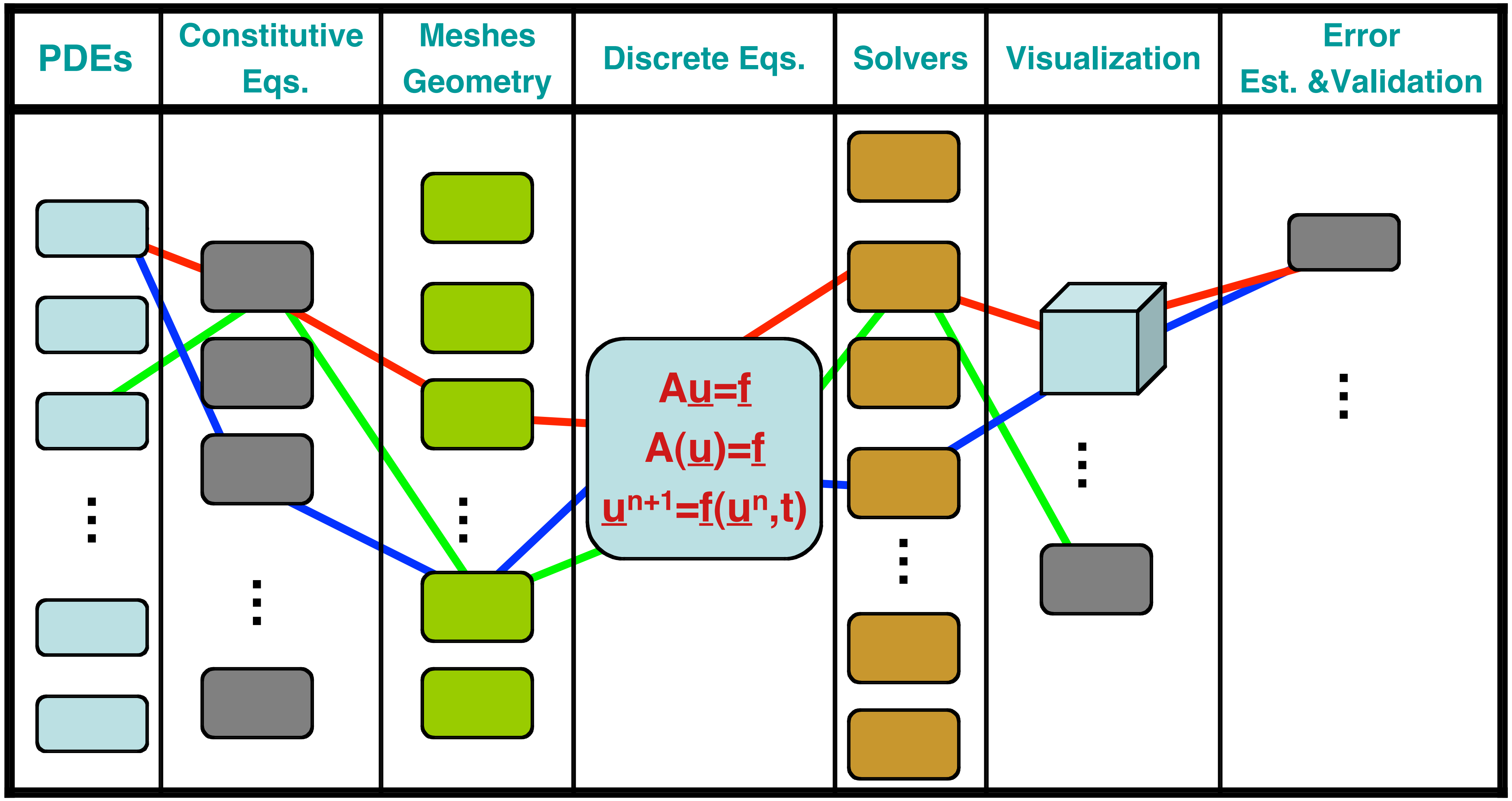
The Structure of Computational Problems (numerical PDE's)



The Structure of Computational Problems
(numerical PDE's)



The Structure of Computational Problems (numerical PDE's)

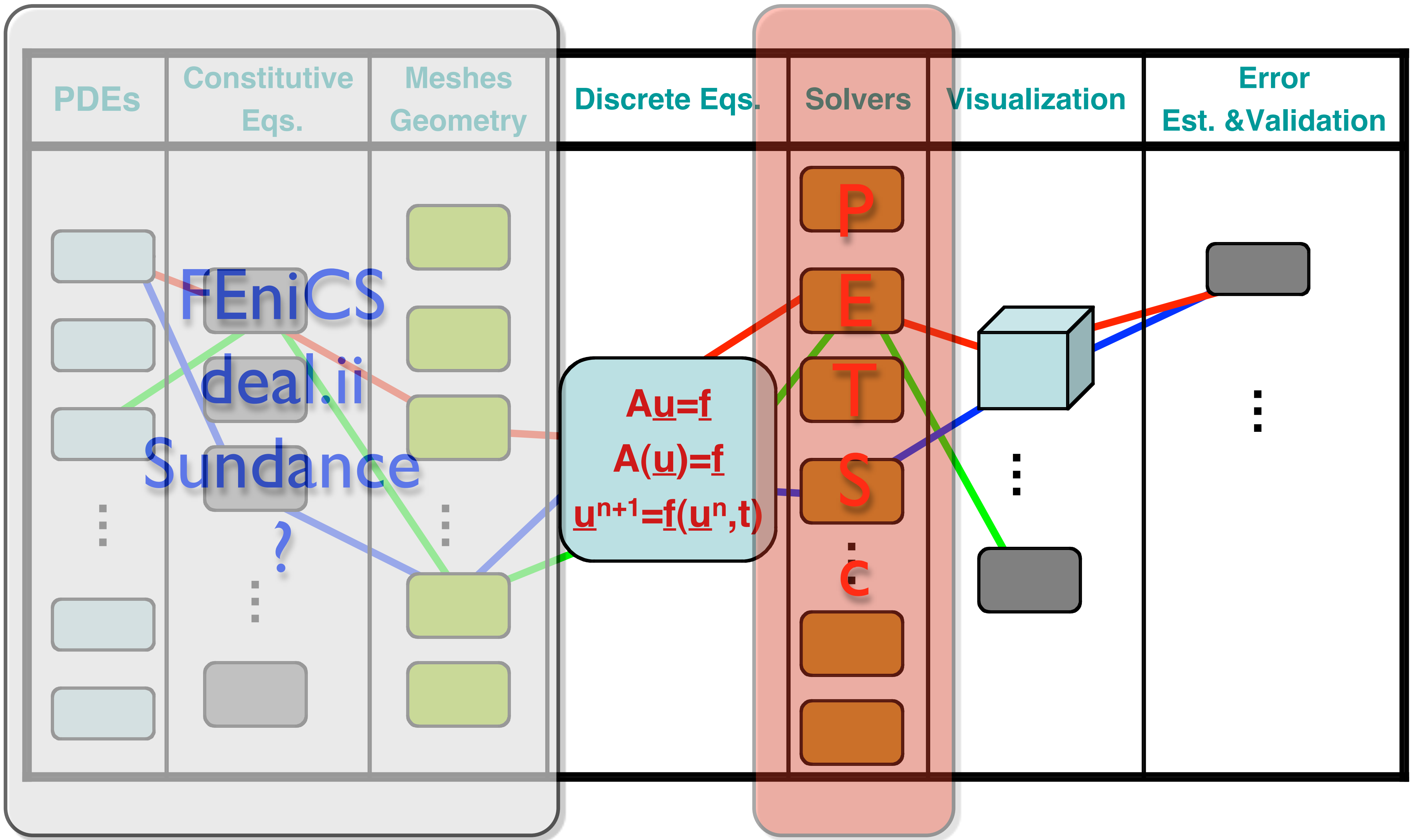


- Computation is really a set of (educated) choices
- Traditionally those choices are made early and hardwired into codes
- Some of this approach is still central to GPU programming.
- Is there a better way?

Software for Multi-physics computation

- PETSc (www.mcs.anl.gov/petsc)
  - Parallel Linear/non-linear solvers
  - Wide range of solver options
    - sparse Direct (umfpack/MUMPS etc.)
    - Algebraic Multi-Grid (HYPRE, ml)
    - Preconditioned Newton-Krylov
    - *Composite & FieldSplit Block Preconditioners*
  - *All chooseable at run time (command line options)*
- The user is responsible for providing A,b or F(u),J(u), PETSc provides interfaces to everything else.

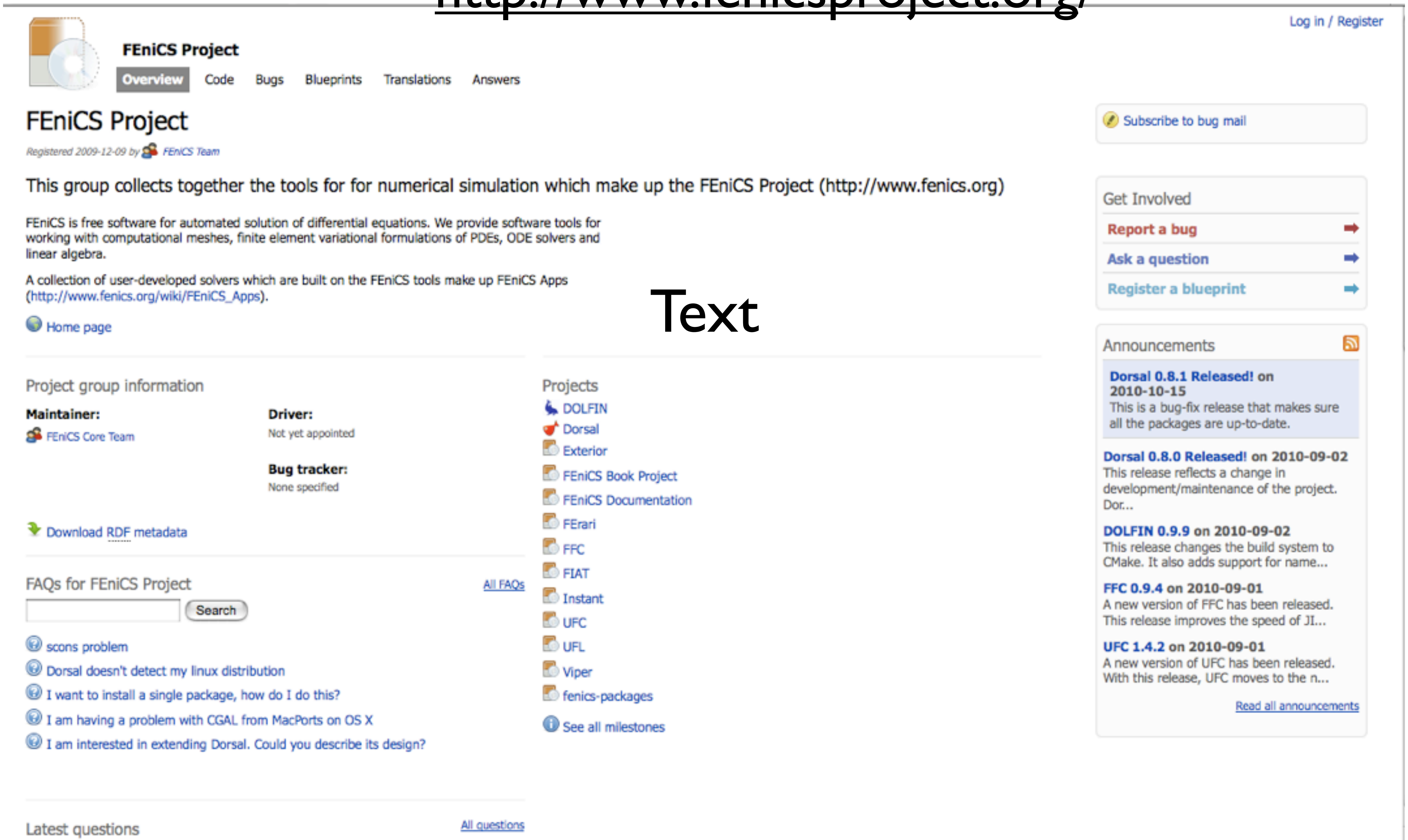
The Structure of Computational Problems (numerical PDE's)



Software for Multi-physics computation

https://launchpad.net/fenics-project

http://www.fenicsproject.org/



Text

Advanced Software for Multi-physics computation

- FEniCS (www.fenicsproject.org)
  - UFL: (Unified Form Language), a python extension for describing variational forms
  - FFC: Form Compiler for automatic FEM code generation from .ufl
  - Dolfin: high-level C++ libraries (and Python bindings) for mesh handling, automatic discretization/assembly and Function abstraction (u.eval(x))

Example: Poisson's Equation

- Strong form: Find $u \in \mathcal{C}^2(\Omega)$ with $u = 0$ on $\partial\Omega$ such that

$$-\nabla^2 u = f \text{ in } \Omega$$

- Weak Form: Find $u \in H_0^1(\Omega)$ such that

$$\int_\Omega \mathbf{\nabla} v(x) \cdot \mathbf{\nabla} u(x) dx = \int_\Omega v(x) f(x) dx \text{ for all } v \in H_0^1(\Omega)$$

- Standard notation: Find $u \in V$ such that

$$a(v, u) = L(v) \text{ for all } v \in \hat{V}$$

with $a : \hat{V} \times V \to \mathbb{R}$ a *bilinear form* and $L : \hat{V} \to \mathbb{R}$ a *linear form*

Poisson.ufl

```
element = FiniteElement("Lagrange", triangle, 1)

v = TestFunction(element)
u = TrialFunction(element)
f = Coefficient(element)
g = Coefficient(element)

a = inner(grad(v), grad(u))*dx
L = v*f*dx + v*g*ds
```

compile with

```
ffc -l dolfin -O Poisson.ufl
```

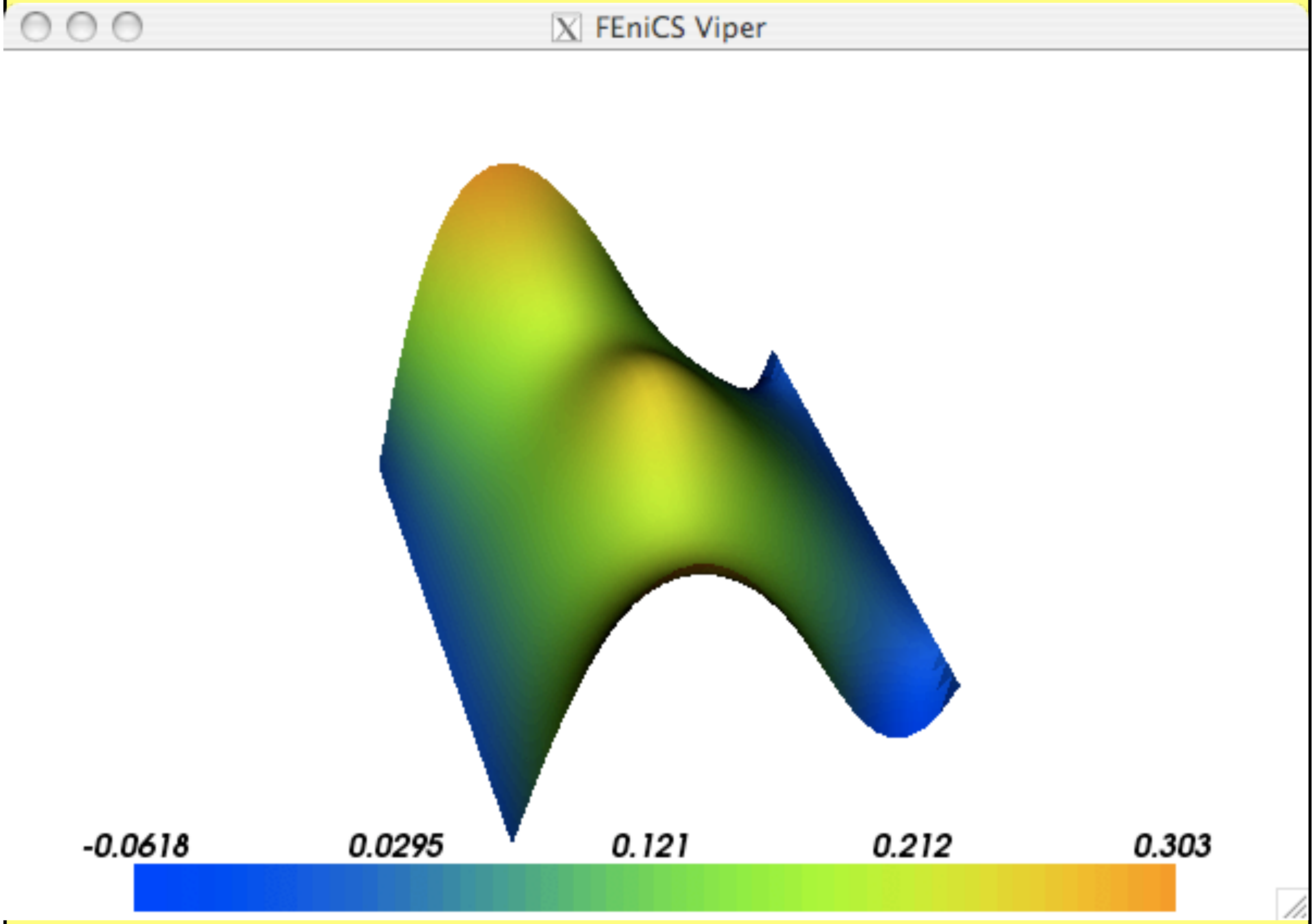
(generates 2110 lines of compilable C++ code as Poisson.h)

Full solution of Poisson with Python interface

```
from dolfin import *

# Create mesh and define function space
mesh = UnitSquare(32, 32)
V = FunctionSpace(mesh, "Lagrange", 1)

# Define Dirichlet boundary (x = 0 or x = 1)
def boundary(x):
    return x[0] < DOLFIN_EPS or x[0] > 1.0 - DOLFIN_EPS

# Define boundary condition
u0 = Constant(0.0)
bc = DirichletBC(V, u0, boundary)

# Define variational problem
u = TrialFunction(V)
v = TestFunction(V)
f = Expression("10*exp(-(pow(x[0] - 0.5, 2) + pow(x[1] - 0.5, 2)) / 0.02)")
g = Expression("sin(5*x[0])")
a = inner(grad(u), grad(v))*dx
L = f*v*dx + g*v*ds

# Compute solution
problem = VariationalProblem(a, L, bc)
u = problem.solve()

# Save solution in VTK format
file = File("poisson.pvd")
file << u

# Plot solution
plot(u, interactive=True)
```

Full solution of Poisson with Python interface

```
from dolfin import *
```



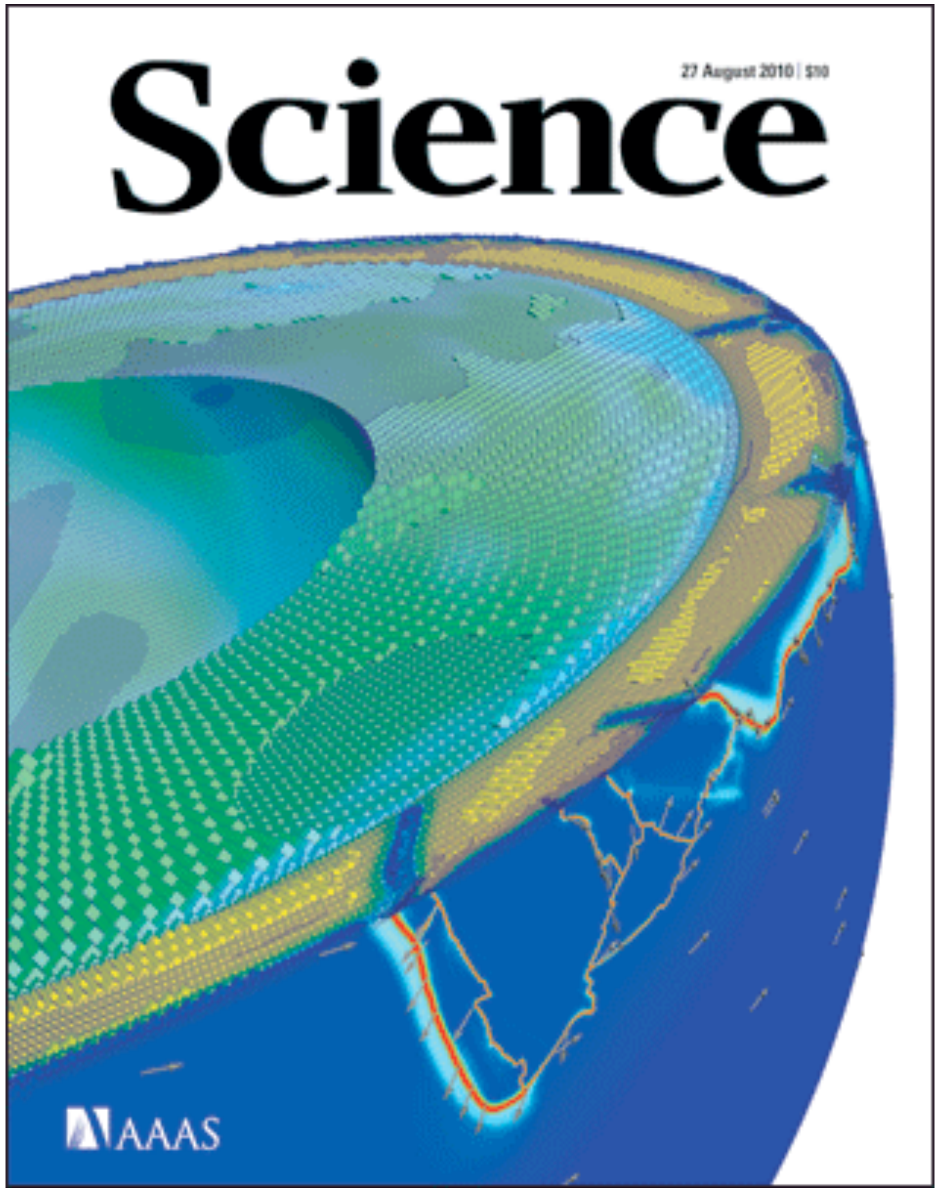

```
# Plot solution
plot(u, interactive=True)
```

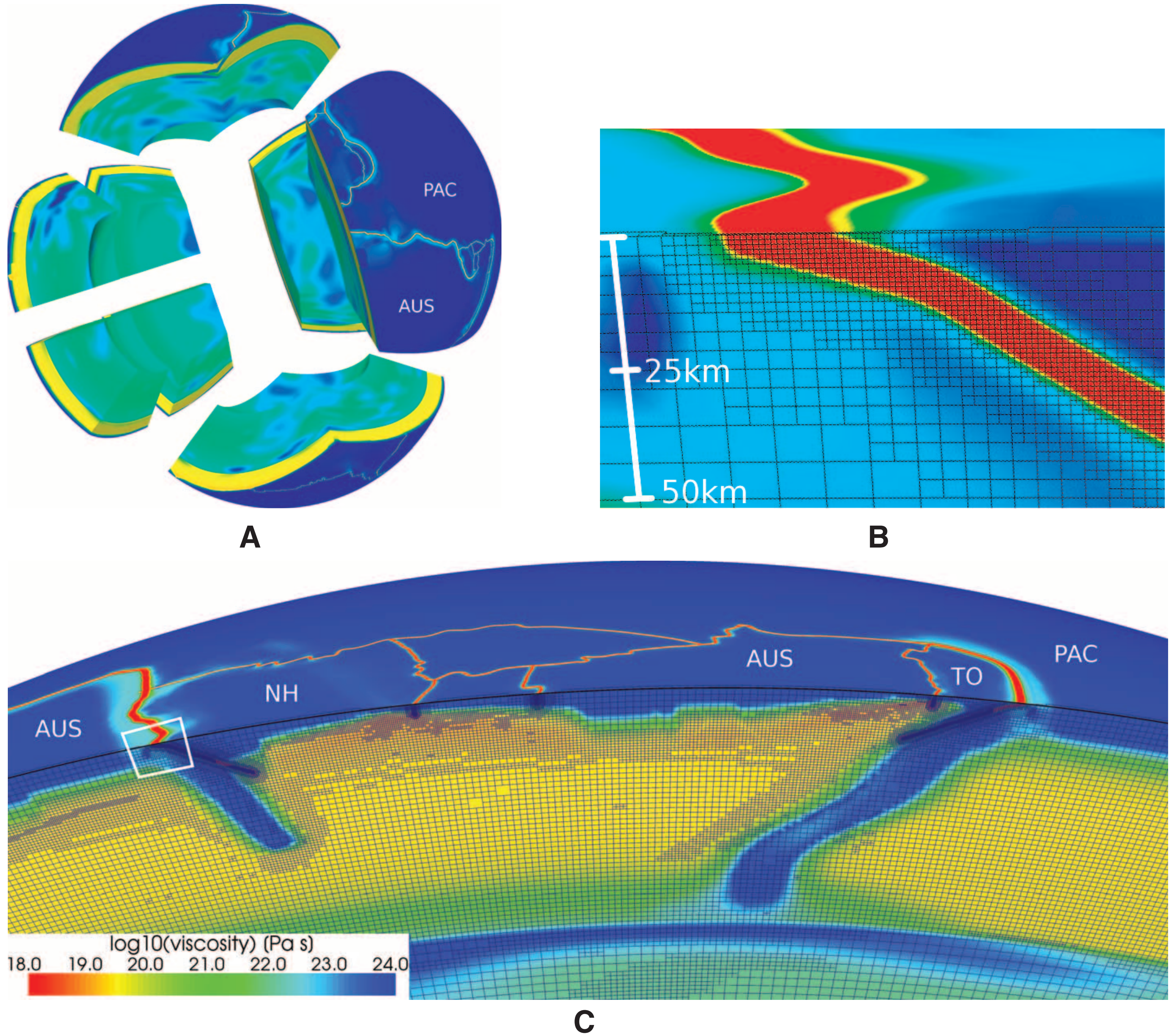
Issues for Multi-physics problems:

- How to rapidly explore/compose different multi-physics models/couplings
- How to maintain control on convergence of global non-linear problem
- How to rapidly change solvers/pre-conditioners as problems change/evolve (defer solver bets)
- How to leverage *existing* algorithms and intuition in designing effective physics-based preconditioners

A model problem:

Infinite Prandtl Number Thermal Convection





The Dynamics of Plate Tectonics and Mantle Flow: From Local to Global Scales

Georg Stadler,[1] Michael Gurnis,[2]* Carsten Burstedde,[1] Lucas C. Wilcox,[1]†
Laura Alisic,[2] Omar Ghattas[1,3,4]

A model problem:

Infinite Pr thermal convection

$$\frac{DT}{Dt} = \frac{1}{\mathrm{Ra}} \nabla^2 T$$

$$-\boldsymbol{\nabla} \cdot \eta(\boldsymbol{\nabla}\mathbf{V} + \boldsymbol{\nabla}\mathbf{V}^T) + \boldsymbol{\nabla} P = T\mathbf{k}$$

$$\boldsymbol{\nabla} \cdot \mathbf{V} = 0$$

with $\eta = \eta(T, \mathbf{V}, \ldots)$

A model problem:

Infinite Pr thermal convection

$$\frac{DT}{Dt} = \frac{1}{\mathrm{Ra}}\nabla^2 T$$

$$-\boldsymbol{\nabla}\cdot\eta(\boldsymbol{\nabla}\mathbf{V} + \boldsymbol{\nabla}\mathbf{V}^T) + \boldsymbol{\nabla}P = T\mathbf{k}$$

$$\boldsymbol{\nabla}\cdot\mathbf{V} = 0$$

with $\eta = \eta(T, \mathbf{V}, \ldots)$

- Coupled Parabolic/Elliptic problem (Adv diff + Stokes)
- usually solved with splitting/Picard iteration assuming quasi-linearity of each equation

Discretization

- FEM in space with mixed element [P2, (P2,P2), P1] for [T,V,P]
- 2nd order Semi-Lagrangian Crank-Nicolson scheme for Energy Equation

Semi-Lagrangian Crank-Nicolson scheme for Energy Equation

$$\frac{T - T^*}{\Delta t} = \frac{1}{2\mathrm{Ra}} \left(\nabla^2 T + (\nabla^2 T)^* \right)$$

where $T^* = T_n(x^*)$ i.e. Temperature at the previous time step and take-off point. Alternatively

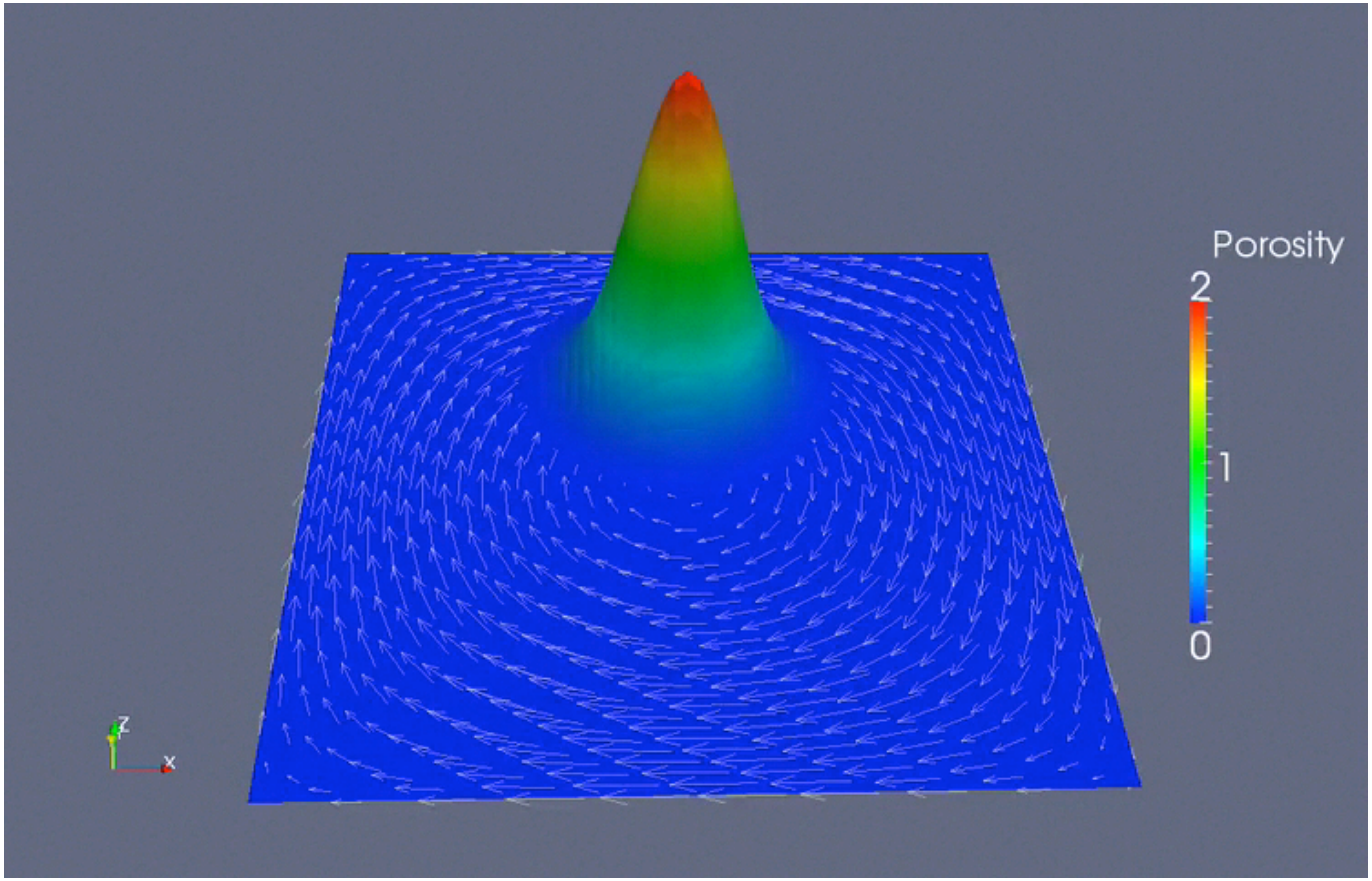
SLCN

$$T - \frac{\Delta t}{2\mathrm{Ra}} \nabla^2 T = g(x^*)$$

where

$$g(x) = T_n + \frac{\Delta t}{2\mathrm{Ra}} \nabla^2 T_n$$

Semi-Lagrangian Advection schemes in FEM



Weak Forms

Weak form of Residual $L = F[u] = F_T + F_V + F_P$

$$
\begin{aligned}
F_T &= \int_\Omega \left[s(T - g^*) + \frac{\Delta t}{2\mathrm{Ra}} \boldsymbol{\nabla} s \cdot \boldsymbol{\nabla} T \right] dx \\
F_V &= \int_\Omega \left[2\eta \boldsymbol{\nabla}^s \mathbf{u} : \boldsymbol{\nabla}^s \mathbf{V} - p \boldsymbol{\nabla} \cdot \mathbf{u} - T \mathbf{u} \cdot \mathbf{k} \right] dx \\
F_P &= \int_\Omega q \boldsymbol{\nabla} \cdot \mathbf{V} dx
\end{aligned}
$$

Weak form of Jacobian $J[u]$

$$a(v, \delta u) = \delta L = \delta F = J[u] \delta u$$

RayleighBenard.ufl

```
#  Choose a mixed vector space
#          for u = [ T,V, P]
P2 = FiniteElement("Lagrange", triangle,2)
P2v = VectorElement("Lagrange", triangle, 2)
P1 = FiniteElement("Lagrange", triangle, 1)
ME  =  MixedElement([P2,P2v, P1])

# quadrature element for Semi-Lagrangian
QE = FiniteElement("Quadrature", triangle,4)

#set test functions and trial functions
(s,u, q) = TestFunctions(ME)
du = TrialFunction(ME)

# solution from last iteration
u0 = Coefficient(ME)

# split mixed functions
(dT,dv, dp,) = split(du)
(T,v, p) = split(u0)

# SemiLagrangianFunctions
gstar = Coefficient(QE)
g = Coefficient(P2)

#parameters and functions
Ra = Constant(triangle)
dt = Constant(triangle) # time step
hdt = 0.5*dt # half time step

#Viscosity function
b = Constant(triangle)
c = Constant(triangle)
x = P2.cell().x
eta = exp(-b*T + c*(1-x[1]))

# weak form of residuals
L1= (s*(T-gstar)+hdt/Ra*inner(grad(s),grad(T)))*dx
L2 = (inner(sym(grad(u)), 2*eta*sym(grad(v))) - div(u)*p - T*u[1])*dx
L3 = q*div(v)*dx

L = L1 + L2 + L3

# Bilinear form for Jacobian with
#         added approximate Semi-Lagrangian block

a = derivative(L,u0,du) +  s*hdt*inner(grad(g),dv)*dx
```

RayleighBenard.ufl

```
#  Choose a mixed vector space
#          for u = [ T,V, P]
P2 = FiniteElement("Lagrange", triangle,2)
P2v = VectorElement("Lagrange", triangle, 2)
P1 = FiniteElement("Lagrange", triangle, 1)
ME  =  MixedElement([P2,P2v, P1])

# quadrature element for Semi-Lagrangian
QE = FiniteElement("Quadrature", triangle,4)

#set test functions and trial functions
(s,u, q) = TestFunctions(ME)
du = TrialFunction(ME)

# solution from last iteration
u0 = Coefficient(ME)

# split mixed functions
(dT,dv, dp,) = split(du)
(T,v, p) = split(u0)

# SemiLagrangianFunctions
gstar = Coefficient(QE)
g = Coefficient(P2)
```

```
#parameters and functions
Ra = Constant(triangle)
dt = Constant(triangle) # time step
hdt = 0.5*dt # half time step

#Viscosity function
b = Constant(triangle)
c = Constant(triangle)
x = P2.cell().x
eta = exp(-b*T + c*(1-x[1]))

# weak form of residuals
L1= (s*(T-gstar)+hdt/Ra*inner(grad(s),grad(T)))*dx
L2 = (inner(sym(grad(u)), 2*eta*sym(grad(v))) - div(u)*p - T*u[1])*dx
L3 = q*div(v)*dx

L = L1 + L2 + L3

# Bilinear form for Jacobian with
#         added approximate Semi-Lagrangian block

a = derivative(L,u0,du) +  s*hdt*inner(grad(g),dv)*dx
```

RayleighBenard.ufl

```
#  Choose a mixed vector space
#          for u = [ T,V, P]
P2 = FiniteElement("Lagrange", triangle,2)
P2v = VectorElement("Lagrange", triangle, 2)
P1 = FiniteElement("Lagrange", triangle, 1)
ME  =  MixedElement([P2,P2v, P1])

# quadrature element for Semi-Lagrangian
QE = FiniteElement("Quadrature", triangle,4)

#set test functions and trial functions
(s,u, q) = TestFunctions(ME)
du = TrialFunction(ME)

# solution from last iteration
u0 = Coefficient(ME)

# split mixed functions
(dT,dv, dp,) = split(du)
(T,v, p) = split(u0)

# SemiLagrangianFunctions
gstar = Coefficient(QE)
g = Coefficient(P2)
```

```
#parameters and functions
Ra = Constant(triangle)
dt = Constant(triangle) # time step
hdt = 0.5*dt # half time step

#Viscosity function
b = Constant(triangle)
c = Constant(triangle)
x = P2.cell().x
eta = exp(-b*T + c*(1-x[1]))

# weak form of residuals
L1= (s*(T-gstar)+hdt/Ra*inner(grad(s),grad(T)))*dx
L2 = (inner(sym(grad(u)), 2*eta*sym(grad(v))) - div(u)*p - T*u[1])*dx
L3 = q*div(v)*dx

L = L1 + L2 + L3

# Bilinear form for Jacobian with
#        added approximate Semi-Lagrangian block

a = derivative(L,u0,du) +  s*hdt*inner(grad(g),dv)*dx
```

Assembled Weak Form for Newton

Assembled Block Newton form (iso-viscous Stokes)

$$J\delta\mathbf{u} = \begin{bmatrix} A & B & 0 \\ -M & K & G \\ 0 & G^T & 0 \end{bmatrix} \begin{bmatrix} \delta_T \\ \delta_{\mathbf{V}} \\ \delta_P \end{bmatrix} = -\mathbf{F}$$

where

$$\begin{aligned} A &= M + \frac{\Delta t}{2\mathrm{Ra}}\mathcal{L} \\ B &= -\frac{\Delta t}{2}\int_\Omega s\boldsymbol{\nabla} g \cdot \delta\mathbf{V} dx \end{aligned}$$

Assembled Weak Form for Newton

Assembled Block Newton form (iso-viscous Stokes)

$$J\delta\mathbf{u} = \begin{bmatrix} A & B & 0 \\ -M & K & G \\ 0 & G^T & 0 \end{bmatrix} \begin{bmatrix} \delta_T \\ \delta_{\mathbf{V}} \\ \delta_P \end{bmatrix} = -\mathbf{F}$$

Stokes

where

$$\begin{aligned} A &= M + \frac{\Delta t}{2\mathrm{Ra}}\mathcal{L} \\ B &= -\frac{\Delta t}{2}\int_\Omega s\boldsymbol{\nabla} g \cdot \delta\mathbf{V} dx \end{aligned}$$

Assembled Weak Form for Newton

Assembled Block Newton form (iso-viscous Stokes)

$$J\delta\mathbf{u} = \begin{bmatrix} A & B & 0 \\ -M & K & G \\ 0 & G^T & 0 \end{bmatrix} \begin{bmatrix} \delta_T \\ \delta_{\mathbf{V}} \\ \delta_P \end{bmatrix} = -\mathbf{F}$$

where

$$\begin{aligned} A &= M + \frac{\Delta t}{2\mathrm{Ra}}\mathcal{L} \\ B &= -\frac{\Delta t}{2}\int_\Omega s\boldsymbol{\nabla} g \cdot \delta\mathbf{V} dx \end{aligned}$$

Assembled Weak Form for Newton

Variable Viscosity adds additional blocks

$$J\delta\mathbf{u} = \begin{bmatrix} A & B & 0 \\ -M + \eta_T & K + \eta_V & G \\ 0 & G^T & 0 \end{bmatrix} \begin{bmatrix} \delta_T \\ \delta_{\mathbf{V}} \\ \delta_P \end{bmatrix} = -\mathbf{F}$$

where

$$\begin{aligned} A &= M + \frac{\Delta t}{2\mathrm{Ra}}\mathcal{L} \\ B &= -\frac{\Delta t}{2}\int_\Omega s\boldsymbol{\nabla} g \cdot \delta\mathbf{V} dx \end{aligned}$$

Some Physics Based Preconditioners
(iso-viscous Stokes)

GSP-Picard Splitting as approximate Jacobian

$$P = \begin{bmatrix} A & \\ \begin{bmatrix} -M \\ 0 \end{bmatrix} & \begin{bmatrix} K & G \\ G^T & 0 \end{bmatrix} \end{bmatrix}$$

Nested GSP - iterative Stokes Solver

$$P = \begin{bmatrix} A & \\ \begin{bmatrix} -M \\ 0 \end{bmatrix} & \begin{bmatrix} \hat{K} & G \\ 0 & \hat{S} \end{bmatrix} \end{bmatrix}$$

Software Requirements

- Rapid and Flexible Composition and Assembly of residuals and block Jacobians.
- Flexible Composition of block preconditioners
- Both Functionalities currently exist in available software
    - FEniCS (www.fenics.org): UFL/FFC/Dolfin
    - PETSc (www.anl.gov/...): PCFieldsplit

PETSc FieldSplit Preconditioners

- Define splits (memory layout done in main code):
  - fieldsplit_0 T
  - fieldsplit_1: Stokes [V, P]
  - Define Nested Splits (for fieldsplit stokes PC's)
    - fieldsplit_1_fieldsplit_0: P
    - fieldsplit_1_fieldsplit_1: V

- Then individual (PC/KSP) pairs can be defined for any split at runtime with command line options
- For Block Triangular Preconditioners use

```
-pc_fieldsplit_type multiplicative
```

PETSc FieldSplit Preconditioners: Examples

-options_file petsc_direct.options

```
# Options file describing (LU,preonly)
# Direct solve of full Jacobian
#
# Set tolerance for Newton iteration
-snes_rtol 1.e-6
-snes_atol 1.e-9
-snes_monitor

# Set (KSP/PC) for Linear solve
-ksp_type preonly
-pc_type lu
-pc_factor_mat_solver_package umfpack
```

PETSc FieldSplit Preconditioners: Examples

petsc_fieldsplit_direct.options

```
# Set Tolerance for Newton Iteration
-snes_rtol 1.e-6
-snes_atol 1.e-9

# KSP for full Jacobian
-ksp_type fgmres      change to preonly for classic picard it
-ksp_rtol 1.e-4

# Fieldsplit Block Preconditioner for Jacobian
-pc_fieldsplit_type multiplicative

# solve Temperature Block (split 0) directly
-fieldsplit_0_ksp_type preonly
-fieldsplit_0_pc_type lu
-fieldsplit_0_pc_factor_mat_solver_package umfpack
# solve Stokes Block (split 1) directly
-fieldsplit_1_ksp_type preonly
-fieldsplit_1_pc_type lu
-fieldsplit_1_pc_factor_mat_solver_package umfpack
```

PETSc FieldSplit Preconditioners: Examples

petsc_nested_fieldsplit.options

```
# Set Tolerance for Newton Iteration
-snes_rtol 1.e-6
-snes_atol 1.e-9

# KSP for full Jacobian
-ksp_type fgmres
-ksp_rtol 1.e-3 -ksp_atol 1.e-10

# Fieldsplit Block Preconditioner for Jacobian
-pc_fieldsplit_type multiplicative

# precondition Temperature Block (split 0) iteratively
-fieldsplit_0_ksp_type cg
-fieldsplit_0_pc_type sor
-fieldsplit_0_ksp_rtol 1.e-4

# precondition Stokes Block (split 1) with Fieldsplit UT
```

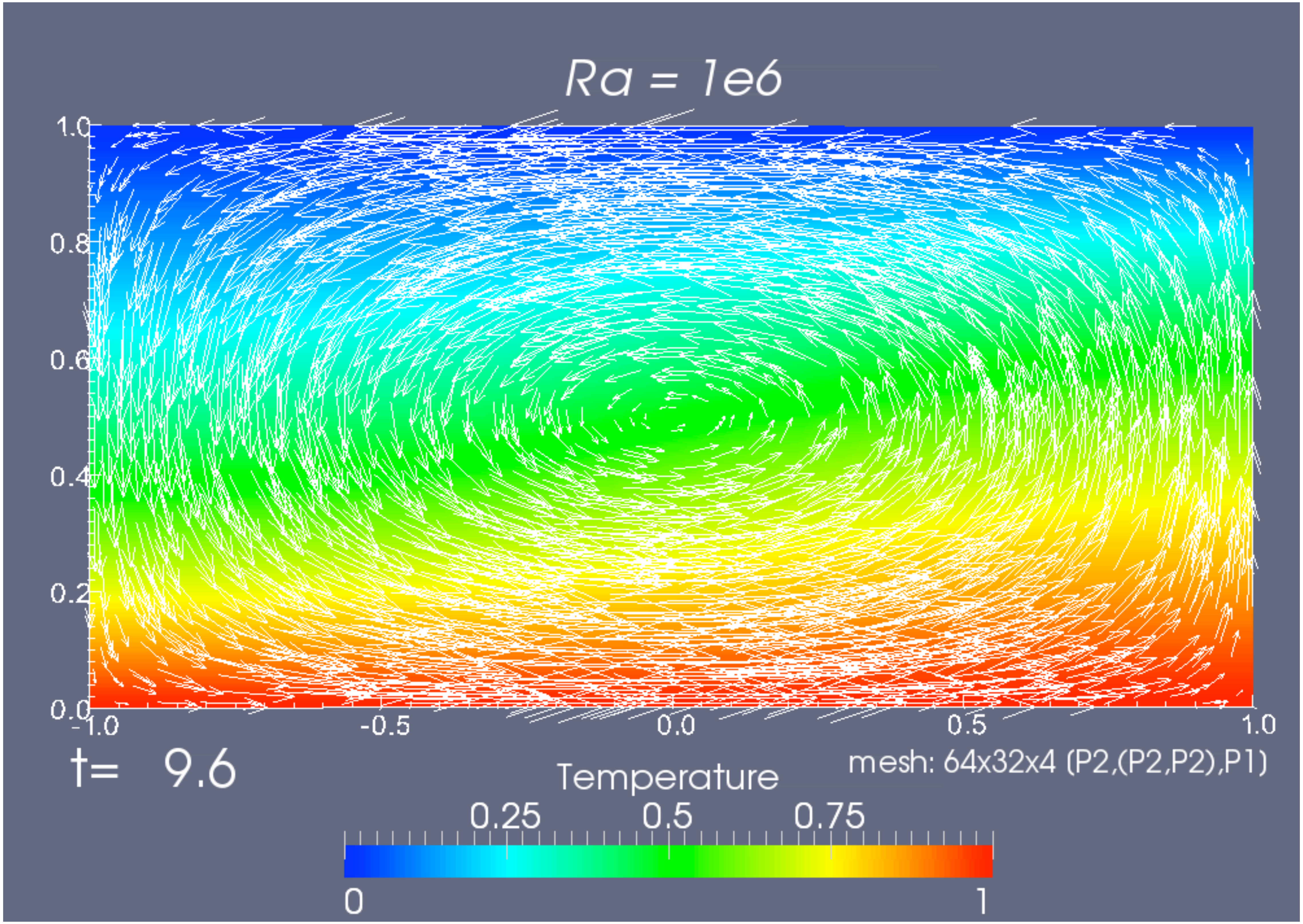
PETSc FieldSplit Preconditioners: Examples

petsc_nested_fieldsplit.options (cont'd)

```
# precondition Stokes Block (split 1)
# Stokes: use upper triangular preconditioner

-fieldsplit_1_ksp_rtol 1.e-4
-fieldsplit_1_ksp_type fgmres
-fieldsplit_1_ksp_monitor
-fieldsplit_1_pc_fieldsplit_type multiplicative

# pressure split (1,0)
-fieldsplit_1_fieldsplit_0_ksp_type cg
-fieldsplit_1_fieldsplit_0_pc_type sor
-fieldsplit_1_fieldsplit_0_ksp_max_it 2

# Velocity split (1,1)
-fieldsplit_1_fieldsplit_1_ksp_type preonly
-fieldsplit_1_fieldsplit_1_pc_type hypre
```

It actually works:

A hybrid FEniCS/PETSc code for Infinite Prandtl Number Thermal Convection



Results: Convergence

FS_direct_preonly	FS_direct	FS_Nested
(64x32x4 triangles, cfl=1, n=1) 0 SNES norm 1.440775725816e-04 1 SNES norm 1.121163685700e-05 2 SNES norm 8.684216284984e-07 3 SNES norm 6.724036897955e-08 4 SNES norm 5.206153641239e-09 5 SNES norm 4.030909787299e-10	0 SNES norm 1.440775934047e-04 0 KSP norm 1.440775934047e-04 1 KSP norm 2.180436488650e-07 2 KSP norm 8.082898217229e-11 1 SNES norm 4.501827567666e-07 0 KSP norm 4.501827567666e-07 1 KSP norm 8.111259702458e-09 2 KSP norm 1.591023715205e-11 2 SNES norm 3.448948297201e-10	0 SNES norm 1.440775934702e-04 0 KSP norm 1.440775934702e-04 1 KSP norm 8.547666994358e-05 2 KSP norm 5.143221835930e-06 3 KSP norm 1.625345646434e-06 4 KSP norm 1.066985886044e-07 1 SNES norm 3.882662679061e-07 0 KSP norm 3.882662679061e-07 1 KSP norm 1.179958777726e-07 2 KSP norm 1.040916620187e-08 3 KSP norm 6.871221156164e-09 4 KSP norm 1.060162940204e-09 5 KSP norm 2.618492833850e-10 2 SNES norm 9.208414734450e-10

FS_direct_preonly	FS_direct	FS_Nested
(128x64x4 triangles, cfl=2, n=1) 0 SNES norm 7.209390761755e-05 1 SNES norm 5.610105695322e-06 2 SNES norm 4.345432491886e-07 3 SNES norm 3.364588778704e-08 4 SNES norm 2.605064460130e-09 5 SNES norm 2.016992518469e-10	0 SNES norm 7.209390761755e-05 0 KSP norm 7.209390761755e-05 1 KSP norm 1.090781352139e-07 2 KSP norm 4.037606468691e-11 1 SNES norm 2.252626498074e-07 0 KSP norm 2.252626498074e-07 1 KSP norm 4.060434945128e-09 2 KSP norm 7.945568136744e-12 2 SNES norm 1.726435313005e-10	0 SNES norm 7.209390763950e-05 0 KSP norm 7.209390763950e-05 1 KSP norm 4.302281006644e-05 2 KSP norm 2.575155542825e-06 3 KSP norm 8.172703781034e-07 4 KSP norm 5.354585680104e-08 1 SNES norm 1.942724998785e-07 0 KSP norm 1.942724998785e-07 1 KSP norm 5.946234196409e-08 2 KSP norm 5.243563962848e-09 3 KSP norm 3.441167666531e-09 4 KSP norm 5.364061333279e-10 5 KSP norm 1.317802056418e-10 2 SNES norm 4.603579469808e-10

Results: Convergence/performance

PC	KSP/SNES				time/FS_Direct_s	
(64x32x4 triangles, cfl=1, n=1)	n=1		n=230		n=1	n=230
FS_direct_preonly	1	5	1	8	2.17	3.34
FS_direct	2	2	4	4	1.00	1.89
FS_nested	4.5	2	6.5	4	1.00	2.18
Direct	1	2	1	4	1.09	1.89

PC	KSP/SNES				time/FS_Direct_s	
(128x64x4 triangles, cfl=2, n=1)	n=1		n=230		n=1	n=230
FS_direct_preonly	1	5	1	7	10.19	11.04
FS_direct	2	2	4	4	4.38	8.21
FS_nested	4.5	2	6	4	6.00	12.62
Direct	1	2	1	4	5.95	10.70

Magma Dynamics

(McKenzie Tutorial Notes (CIG), Katz et al, 2007 Pepi)

$$\frac{D\phi}{Dt} = (1-\phi)\frac{\mathcal{P}}{\xi} + \Gamma/\rho_s$$

Compressible Flow

$$-\boldsymbol{\nabla}\cdot\frac{K}{\mu}\boldsymbol{\nabla}\mathcal{P} + \frac{\mathcal{P}}{\xi} = \boldsymbol{\nabla}\cdot\frac{K}{\mu}\left[\boldsymbol{\nabla}P^* + \Delta\rho\mathbf{g}\right] + \Gamma\frac{\Delta\rho}{\rho_f\rho_s}$$

$$\boldsymbol{\nabla}\cdot\mathbf{V} = \frac{\mathcal{P}}{\xi}$$

"Incompressible" Flow

$$\boldsymbol{\nabla}P^* = \boldsymbol{\nabla}\cdot\eta\left(\boldsymbol{\nabla}\mathbf{V} + \boldsymbol{\nabla}\mathbf{V}^T\right) - \phi\Delta\rho\mathbf{g}$$

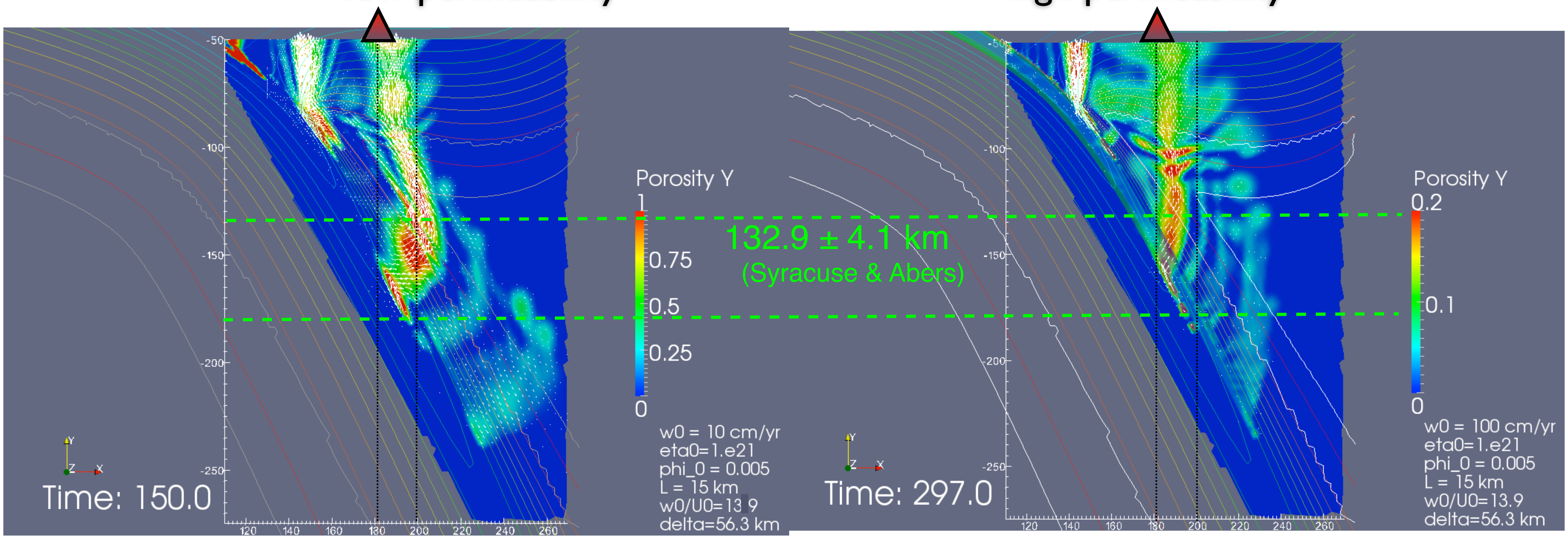
with

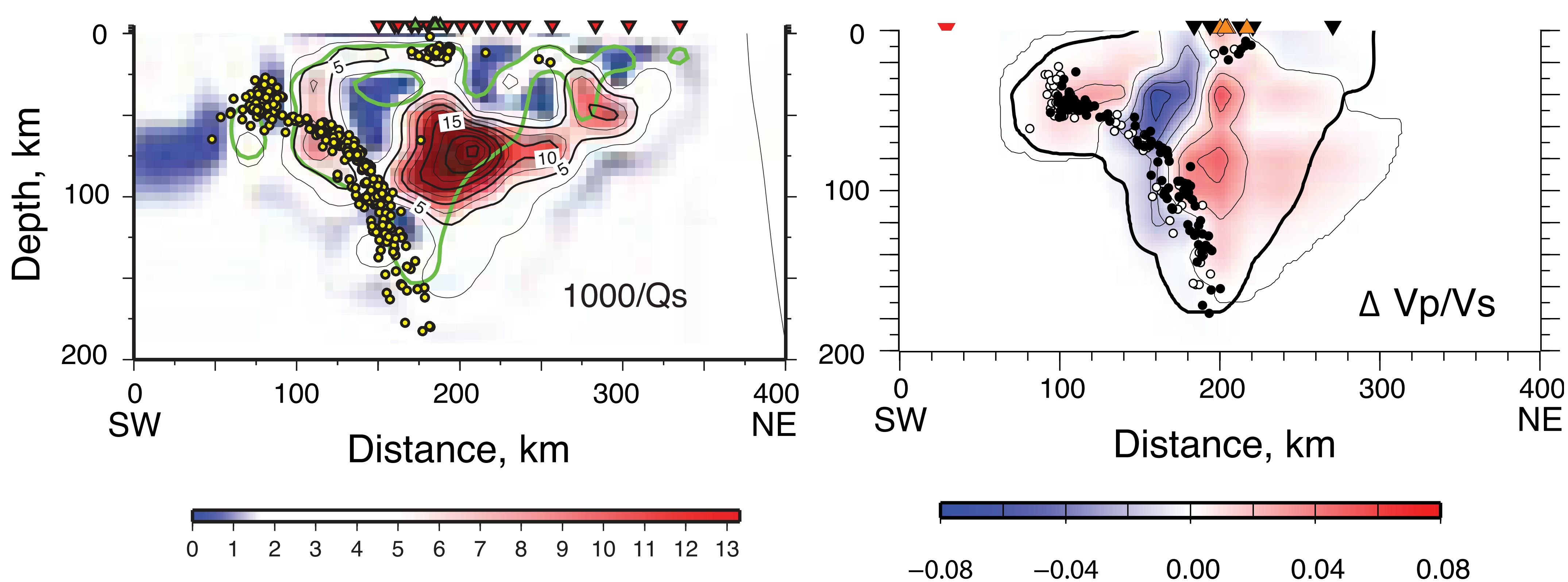
- $\xi = (\zeta - 2\eta/3) = \eta\left(\frac{1}{\phi} - \frac{2}{3}\right) \approx \eta/\phi$
- $\Delta\rho = \rho_s - \rho_f$

Comparison to TUCAN Data

low permeability

high permeability





Conclusions

- Advances in Software (FEniCS/PETSc) allows flexible and rapid composition of multi-physics models, discretization and block-preconditioners
- Allows choices of model coupling/solvers to be made at, or close to run time (but doesn't help you make those choices.)
- Newton with block pre-conditioners allows monitoring convergence of global non-linear problem.
- This approach has already led to working scientific codes.
- Physics based fieldsplit pre-conditioners are more efficient than Picard splitting, and comparable to sparse-direct (which is hard to beat in 2-D)

Ongoing Issues

- Performance tuning and profiling needs to be done to understand timing differences.
- Selective block assembly is needed for efficiency.
- Parallelism: Both FEniCS/PETSc are parallel (but need to implement parallel Semi-Lagrangian or choose a different advection scheme). Questions of performance and scaleability.
- Science Challenges: Full Magma Dynamics (RBConvection + fluids), 3-D, more non-linear couplings.
- Comparison to other multi-physics approaches
- But proof-of-concept exists.