A Finite Volume PDE Solver Using Python (FiPy)

Jonathan E. Guyer, Daniel Wheeler & James A. Warren

guyer@nist.gov daniel.wheeler@nist.gov jwarren@nist.gov

Metallurgy Division

Materials Science and Engineering Laboratory

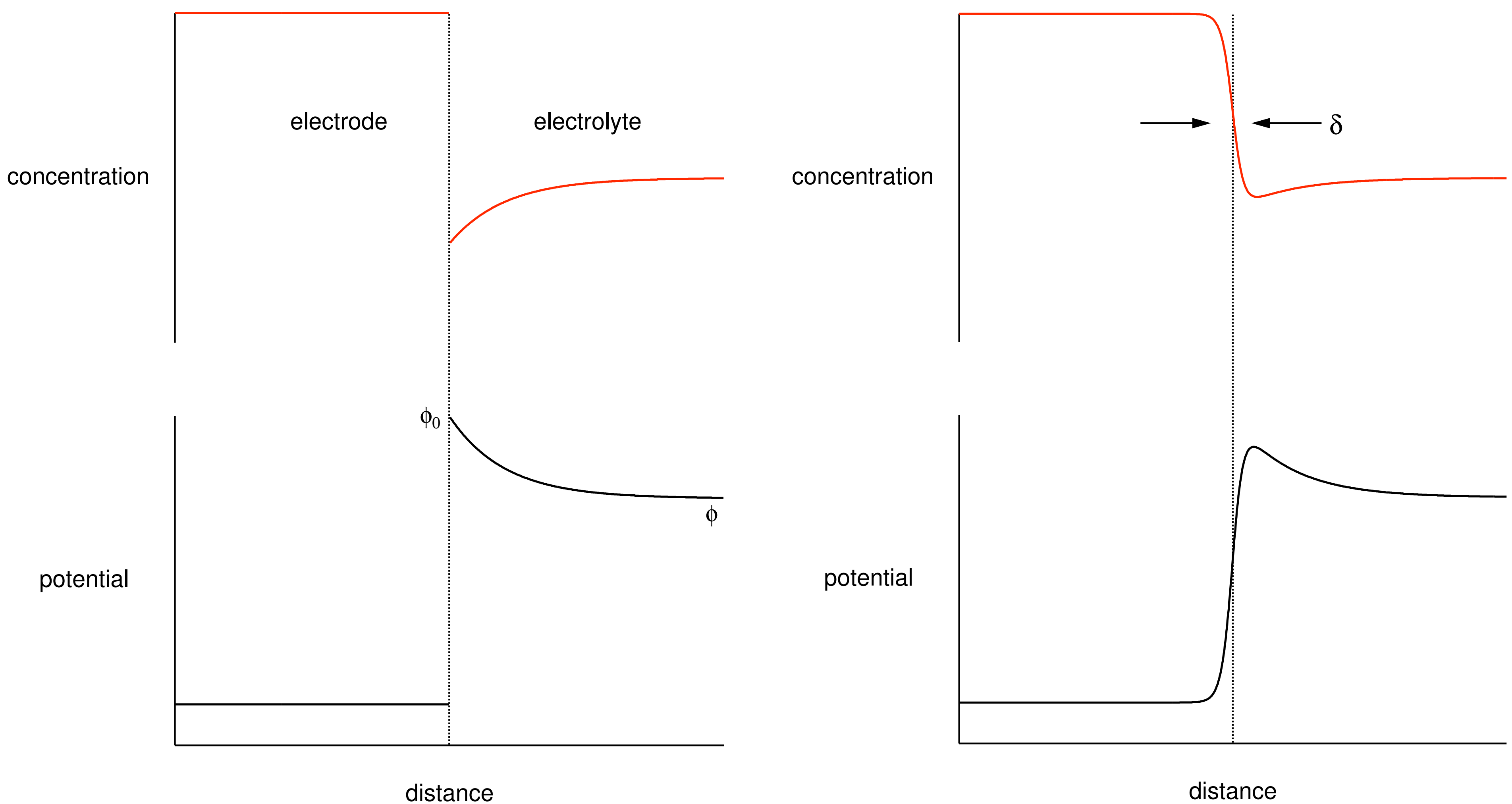
NIST **National Institute of Standards and Technology** • Technology Administration • U.S. Department of Commerce

Motivation

- Many interface tracking codes for solving Materials Science problems
- Commercial CFD codes are expensive and written in FORTRAN or C
- Interface tracking requires unstructured meshes
- Unstructured meshing requires Finite Element / Finite Volume
- Difficult for end users to customize existing codes in FORTRAN/C
  - Python is the code *and* the user script
- Good Python libraries to exploit
  - NumPy for array manipulation
  - PySparse for linear algebra
  - Gist for viewing
  - SciPy for C inlining within Python (`weave`)
  - Scientific Python for physical dimensions
  - Profiler - PyGTK GUI by NIST's Steve Langer

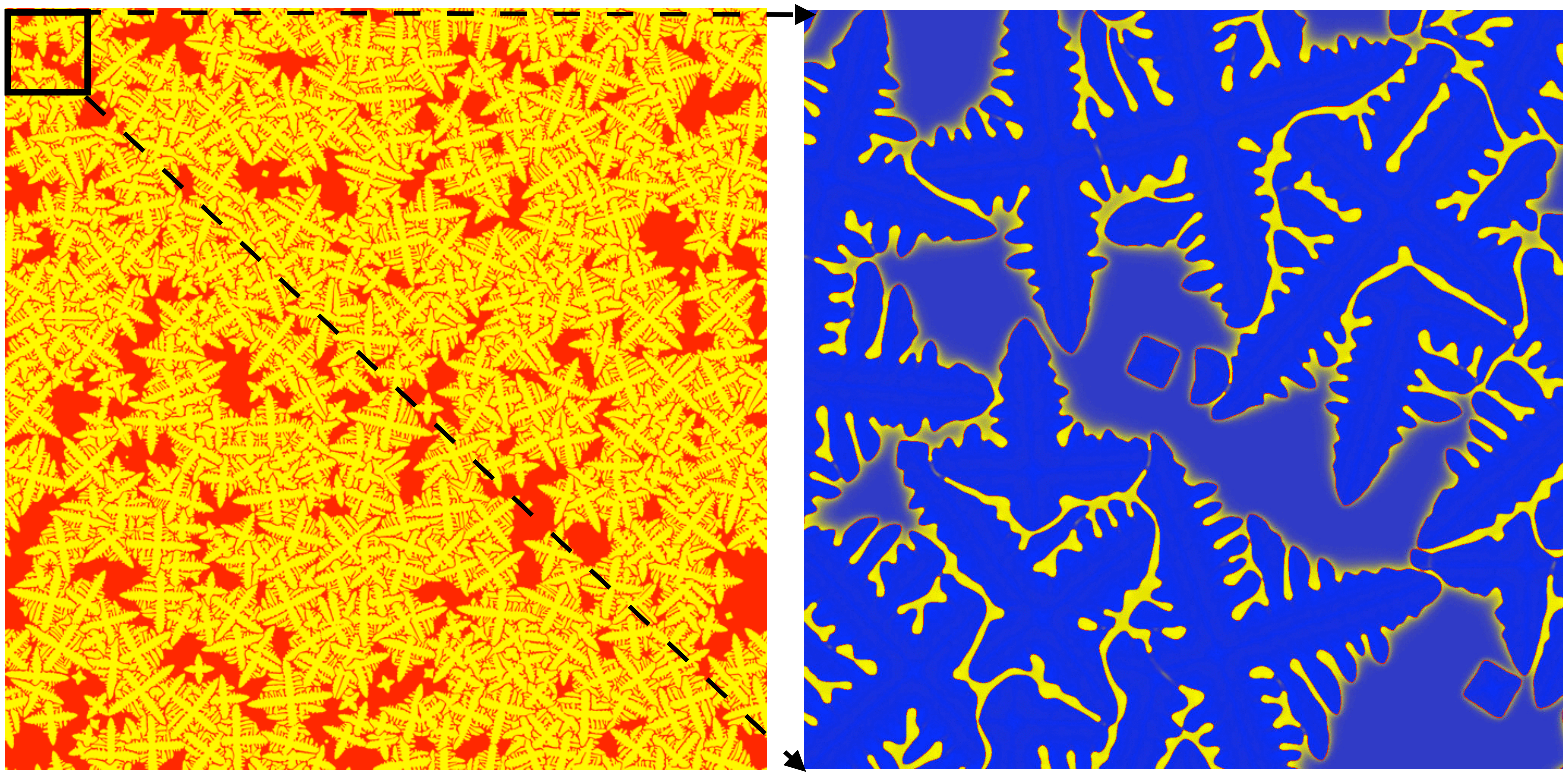
Research Background
Modeling Phase Transformations

- Phase Field



Research Background
Modeling Phase Transformations

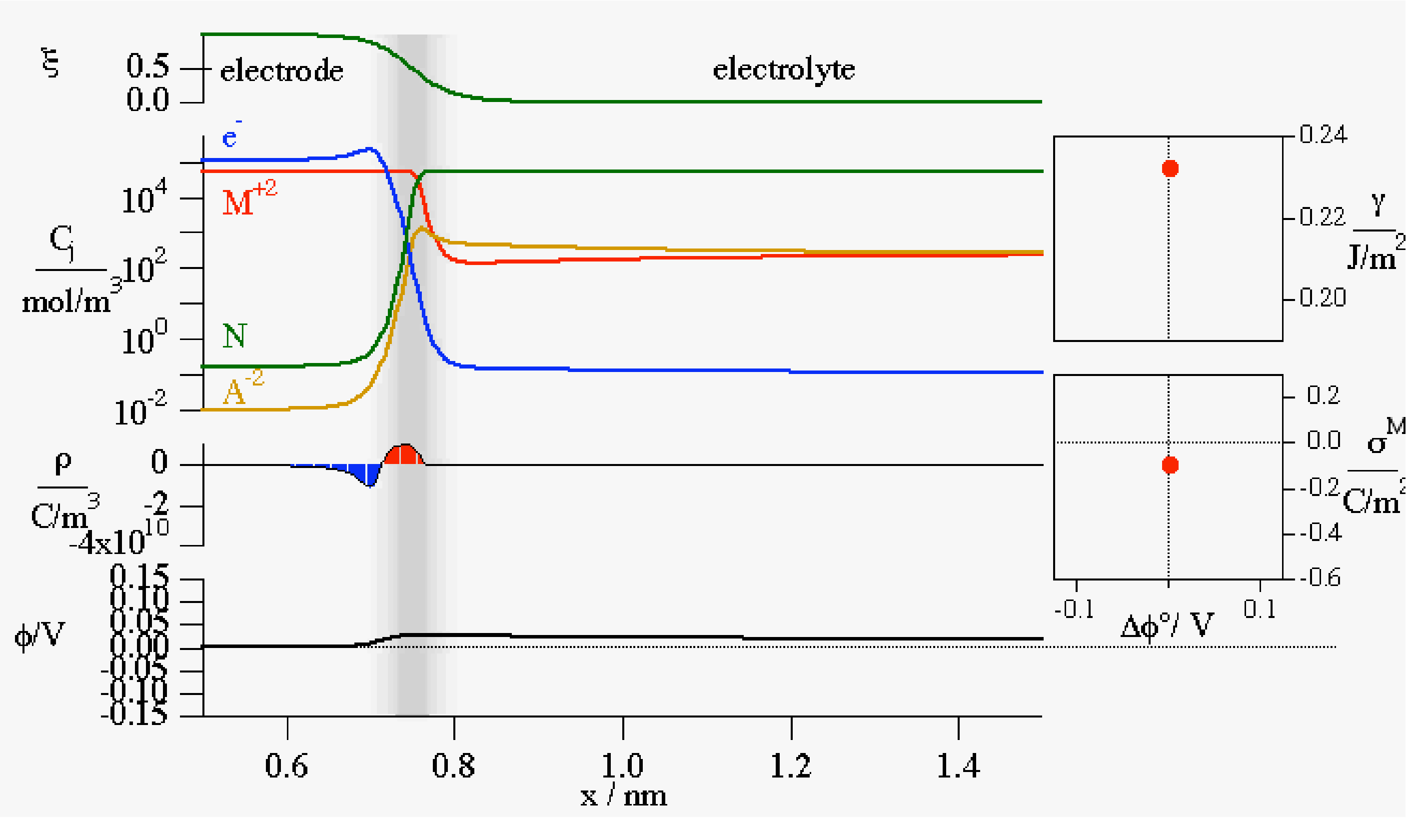
- Phase Field
  - Solidification
    - Dendrites
    - Grain boundary motion, impingement, rotation



L. Gránásy, T. Börzsönyi, T. Pusztai, *Phys. Rev. Lett.* **88**, 205105 (2002)

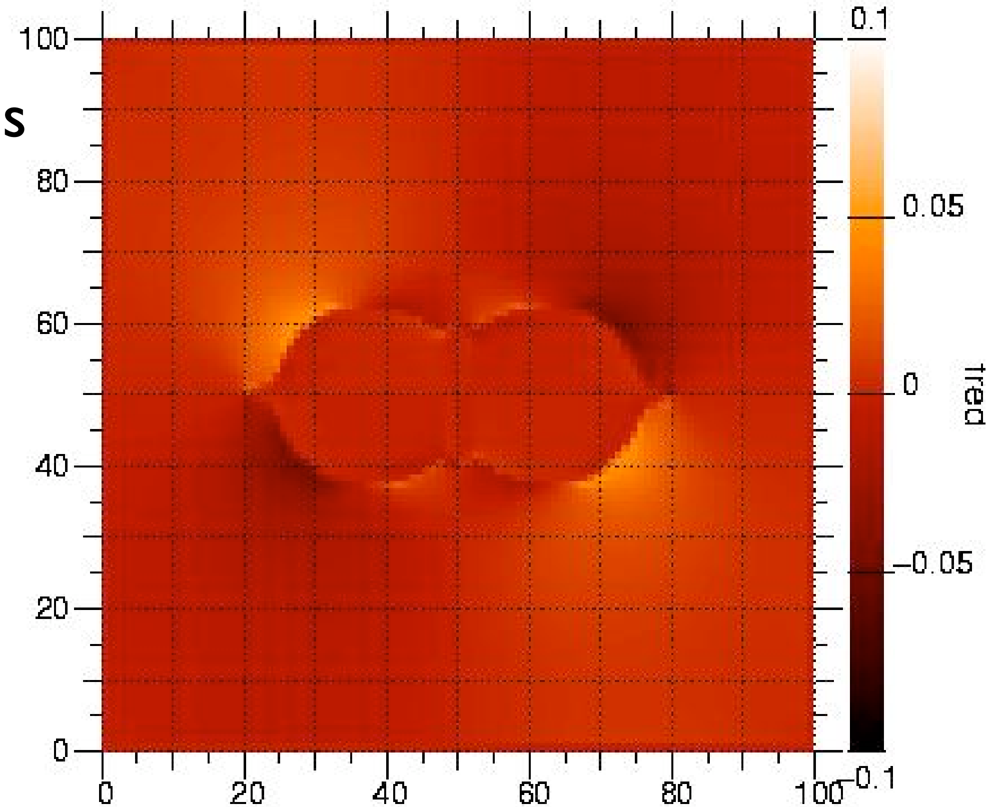
Research Background
Modeling Phase Transformations

- Phase Field
  - Solidification
    - Dendrites
    - Grain boundary motion, impingement, rotation
  - Electrochemistry



Research Background
Modeling Phase Transformations

- Phase Field
  - Solidification
    - Dendrites
    - Grain boundary motion, impingement, rotation
  - Electrochemistry
  - Stresses and strains

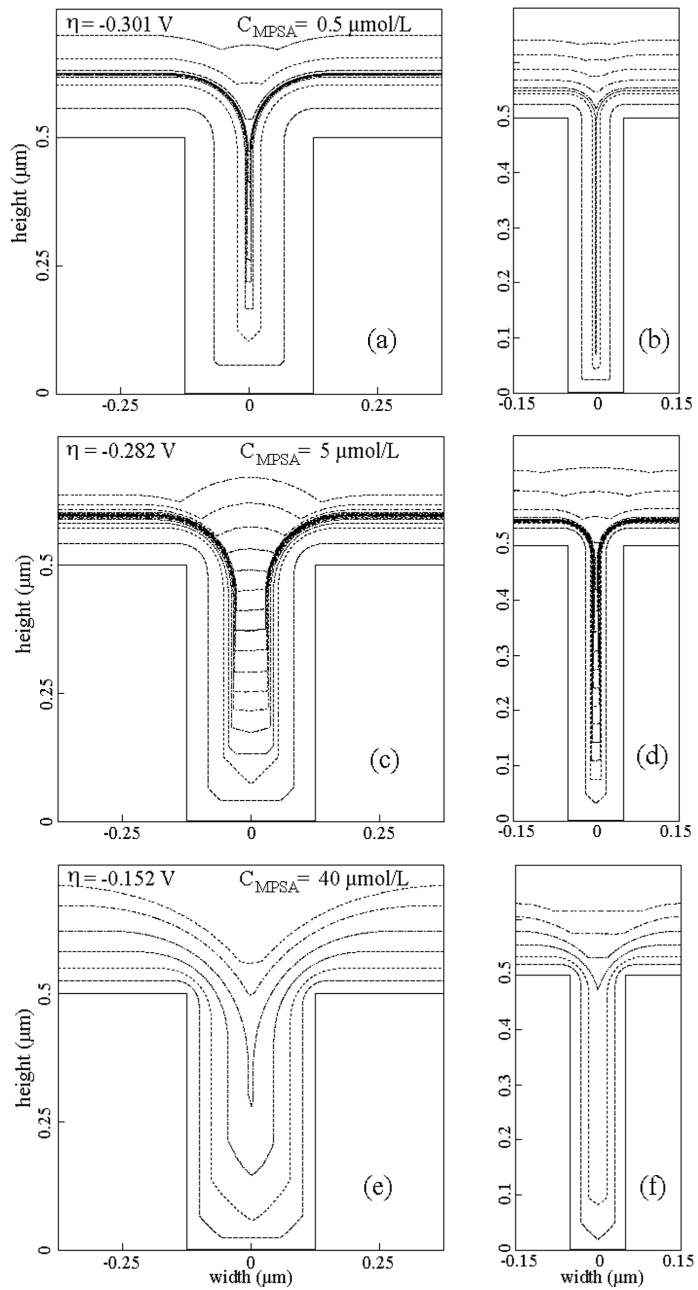


σ_{12}

J. Slutsker, G. B. McFadden, A. L. Roytburd,
W. J. Boettinger, and J. A. Warren

Research Background
Modeling Phase Transformations

- Phase Field
  - Solidification
    - Dendrites
    - Grain boundaries
  - Electrochemistry
  - Stresses and strains
- Level Set Method
  - Electrodeposition



National Institute of Standards and Technology
Technology Administration, U.S. Department of Commerce

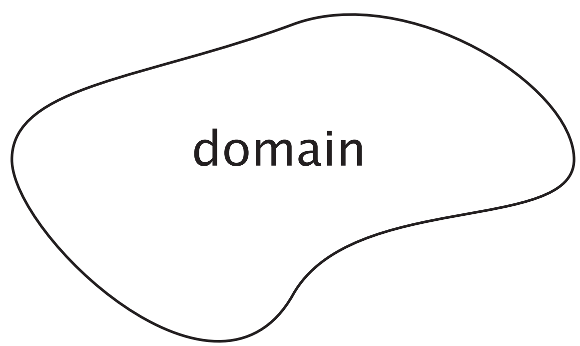
FiPy Overview

- Finite Volume code for solving coupled sets of PDEs
- Code currently addresses Phase Field models
- Compatible with unstructured meshes
- Code will be freely available (watch `<http://www.ctcms.nist.gov>`)
- Large archive of test problems
- User controls program flow

Finite Volume Method

- Solve a general PDE on a given domain for a field ϕ

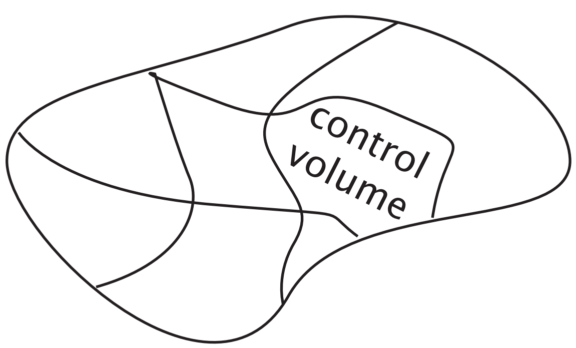
$$\underbrace{\frac{\partial(\rho\phi)}{\partial t}}_{\text{transient}} + \underbrace{\nabla\cdot(\vec{u}\phi)}_{\text{convection}} = \underbrace{\nabla\cdot(\Gamma\nabla\phi)}_{\text{diffusion}} + \underbrace{S_\phi}_{\text{source}}$$



Finite Volume Method

- Solve a general PDE on a given domain for a field ϕ
- Integrate PDE over arbitrary control volumes

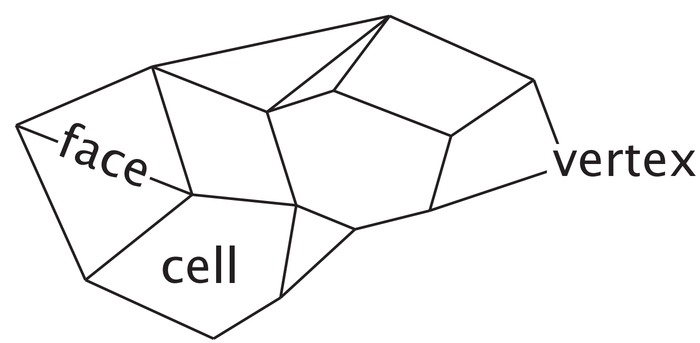
$$\underbrace{\int_V \frac{\partial(\rho\phi)}{\partial t}\,dV}_{\text{transient}} + \underbrace{\int_S (\vec{n}\cdot\vec{u})\phi\,dS}_{\text{convection}} = \underbrace{\int_S \Gamma(\vec{n}\cdot\nabla\phi)\,dS}_{\text{diffusion}} + \underbrace{\int_V S_\phi\,dV}_{\text{source}}$$



Finite Volume Method

- Solve a general PDE on a given domain for a field ϕ
- Integrate PDE over arbitrary control volumes
- Evaluate PDE over polyhedral control volumes

$$\underbrace{\frac{\rho\phi V - (\rho\phi V)^{\text{old}}}{\Delta t}}_{\text{transient}} + \underbrace{\sum_{\text{face}} [(\vec{n}\cdot\vec{u})A\phi]_{\text{face}}}_{\text{convection}} = \underbrace{\sum_{\text{face}} [\Gamma A \vec{n}\cdot\nabla\phi]_{\text{face}}}_{\text{diffusion}} + \underbrace{V S_\phi}_{\text{source}}$$

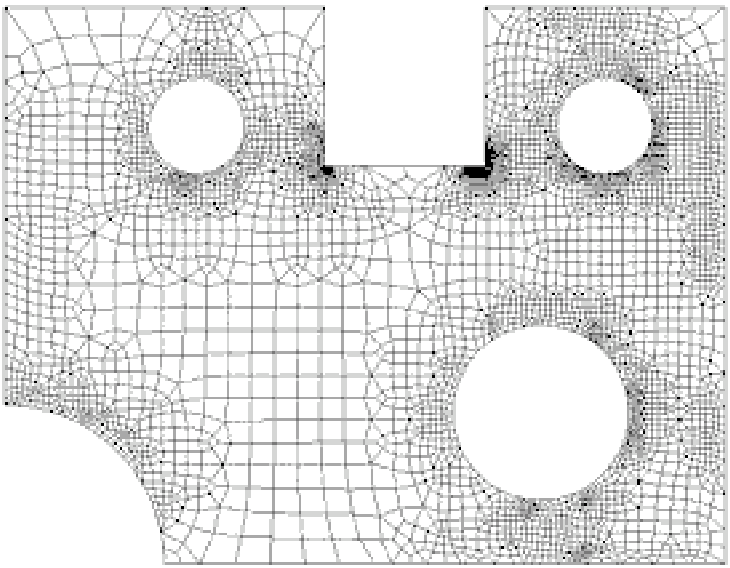


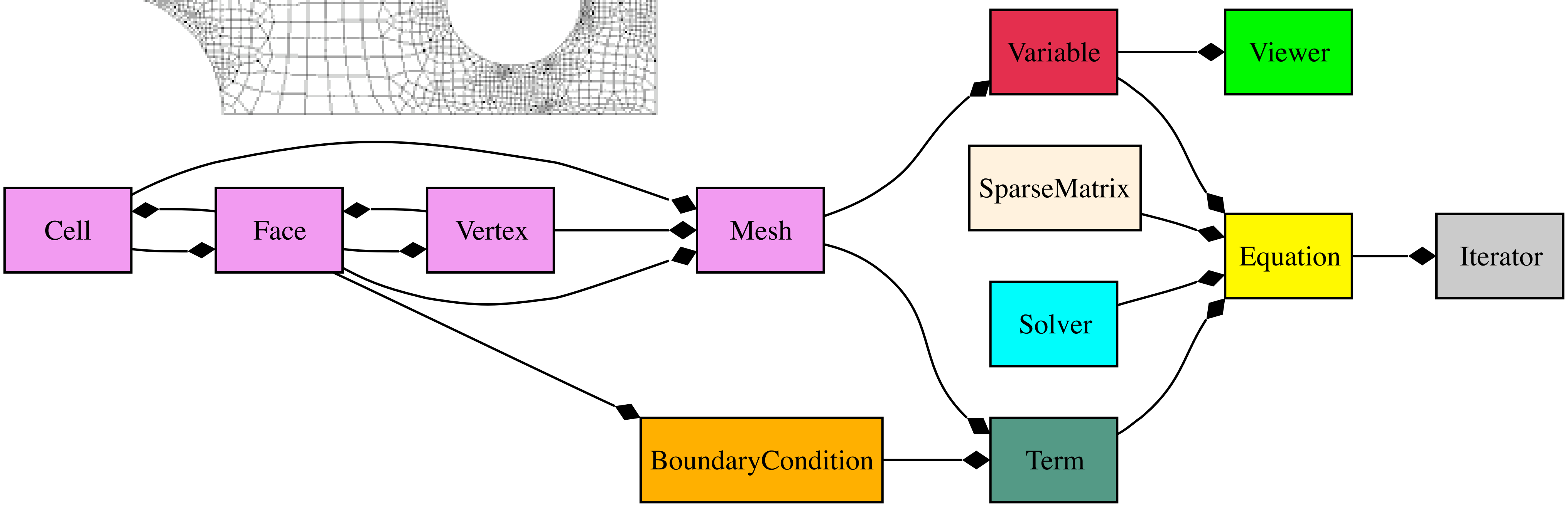
Finite Volume Method

- Solve a general PDE on a given domain for a field ϕ
- Integrate PDE over arbitrary control volumes
- Evaluate PDE over polyhedral control volumes
- Obtain a large coupled set of linear equations in ϕ

$$\begin{pmatrix} a_{11} & a_{12} & & \\ a_{21} & a_{22} & \ddots & \\ & \ddots & \ddots & \ddots \\ & & \ddots & a_{nn} \end{pmatrix} \begin{pmatrix} \phi_1 \\ \phi_2 \\ \vdots \\ \phi_n \end{pmatrix} = \begin{pmatrix} b_1 \\ b_2 \\ \vdots \\ b_n \end{pmatrix}$$

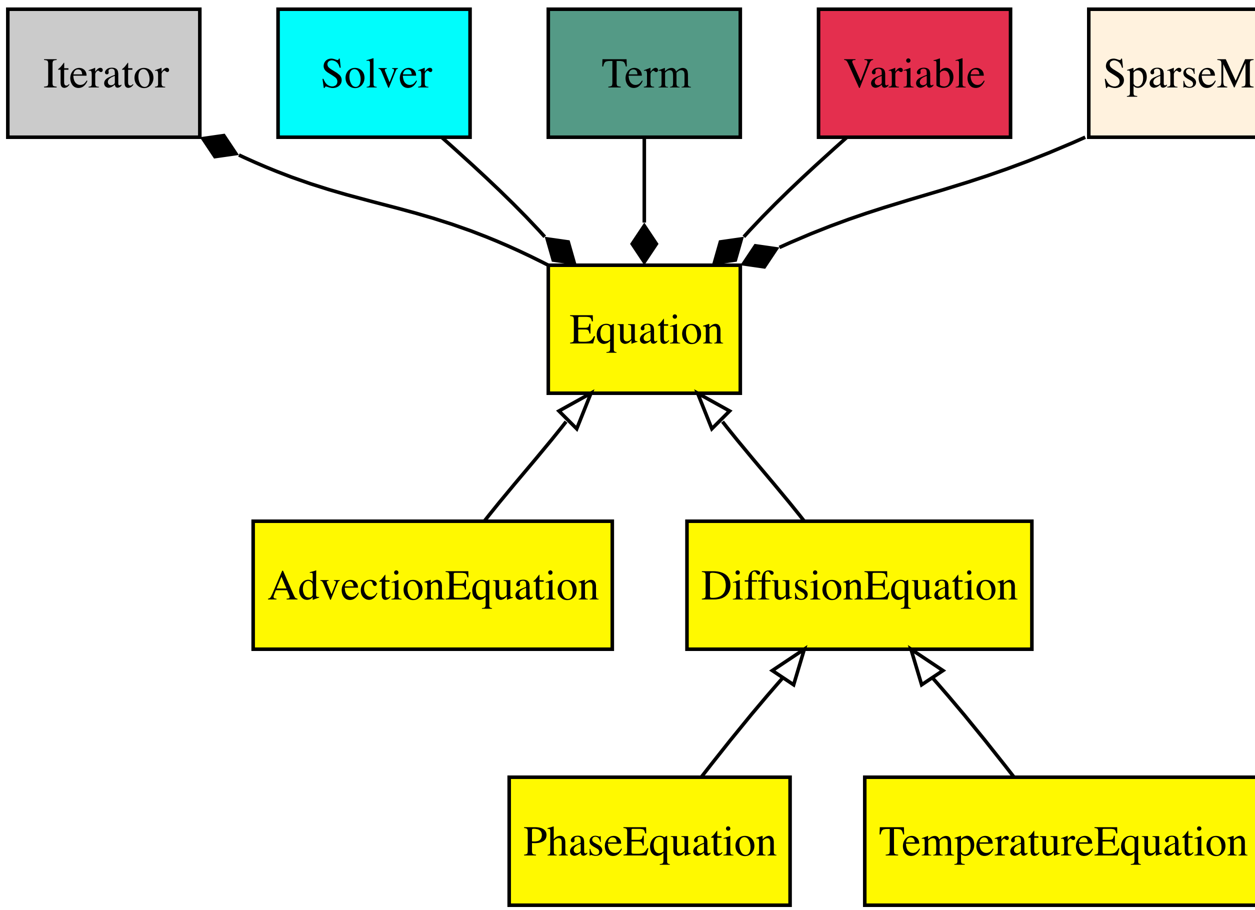
FiPy Design - Objects

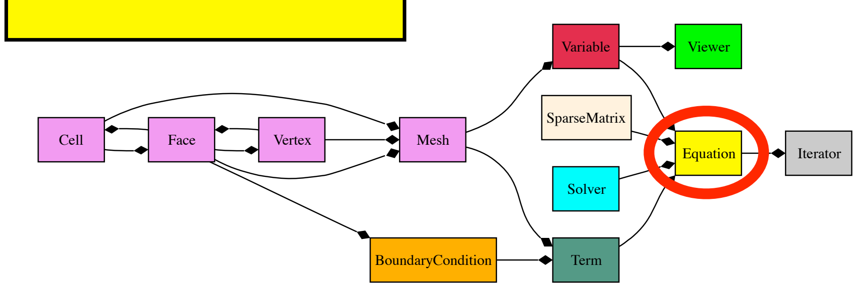




FiPy Design - Equations

$$\underbrace{\int_V \frac{\partial(\rho\phi)}{\partial t}\,dV}_{\text{transient}} + \underbrace{\int_S (\vec{n}\cdot\vec{u})\phi\,dS}_{\text{convection}} = \underbrace{\int_S \Gamma(\vec{n}\cdot\nabla\phi)\,dS}_{\text{diffusion}} + \underbrace{\int_V S_\phi\,dV}_{\text{source}}$$

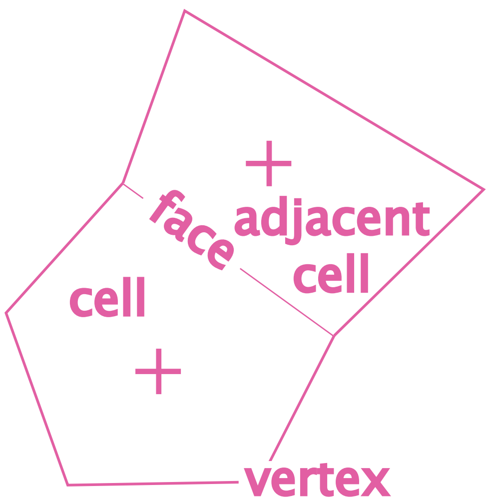


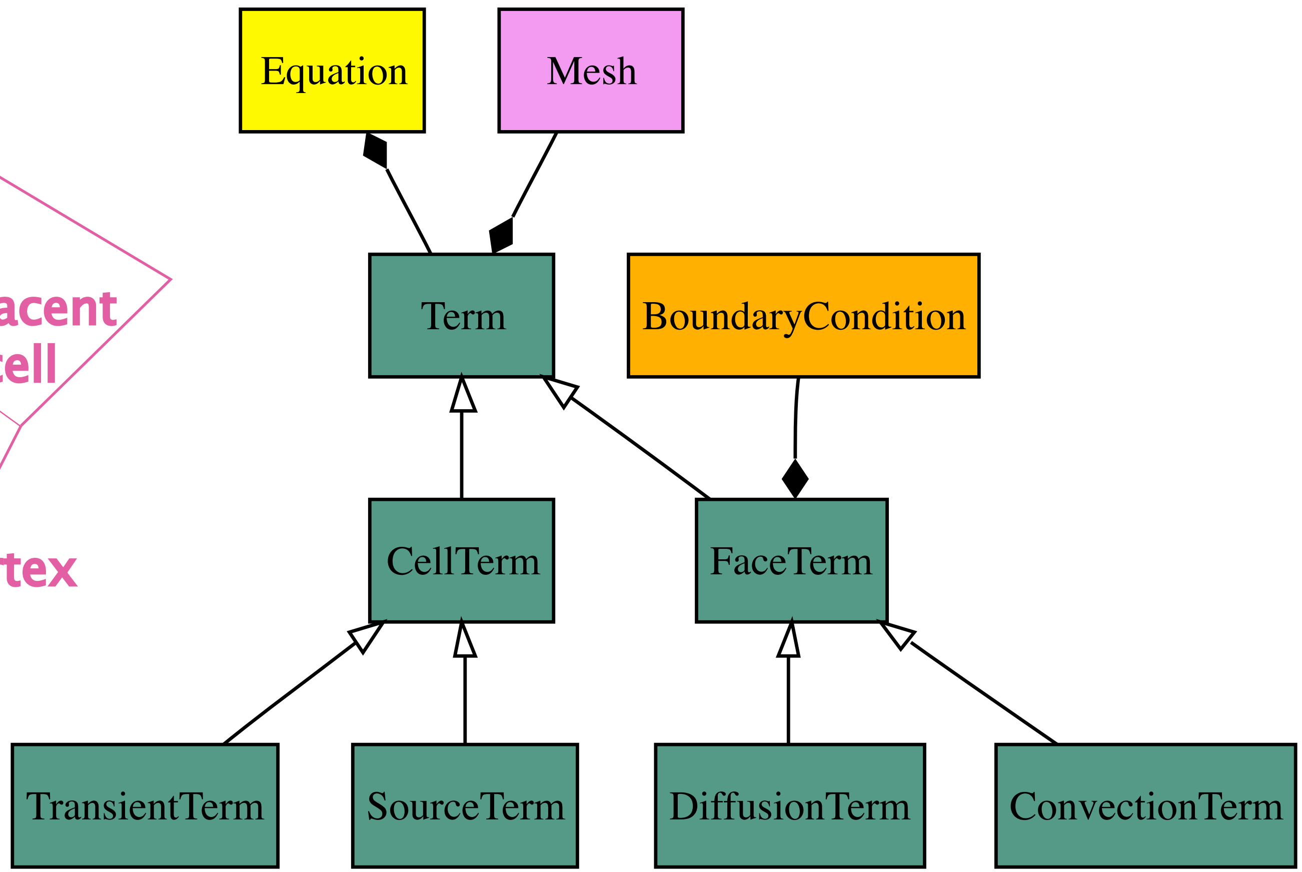


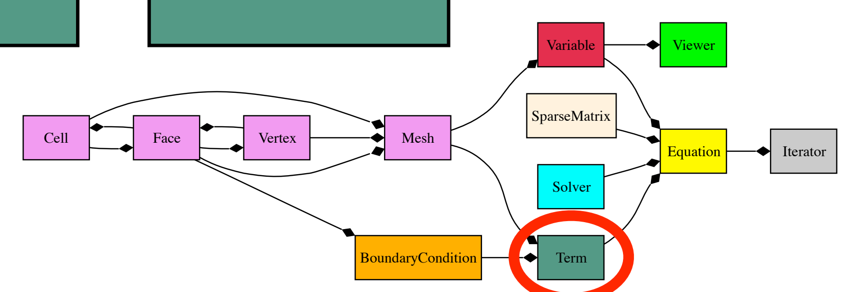
National Institute of Standards and Technology
Technology Administration, U.S. Department of Commerce

FiPy Design - Terms

$$\underbrace{\frac{\rho\phi V - (\rho\phi V)^{\text{old}}}{\Delta t}}_{\text{transient}} + \underbrace{\sum_{\text{face}}[(\vec{n}\cdot\vec{u})A\phi]_{\text{face}}}_{\text{convection}} = \underbrace{\sum_{\text{face}}[\Gamma A\vec{n}\cdot\nabla\phi]_{\text{face}}}_{\text{diffusion}} + \underbrace{VS_\phi}_{\text{source}}$$



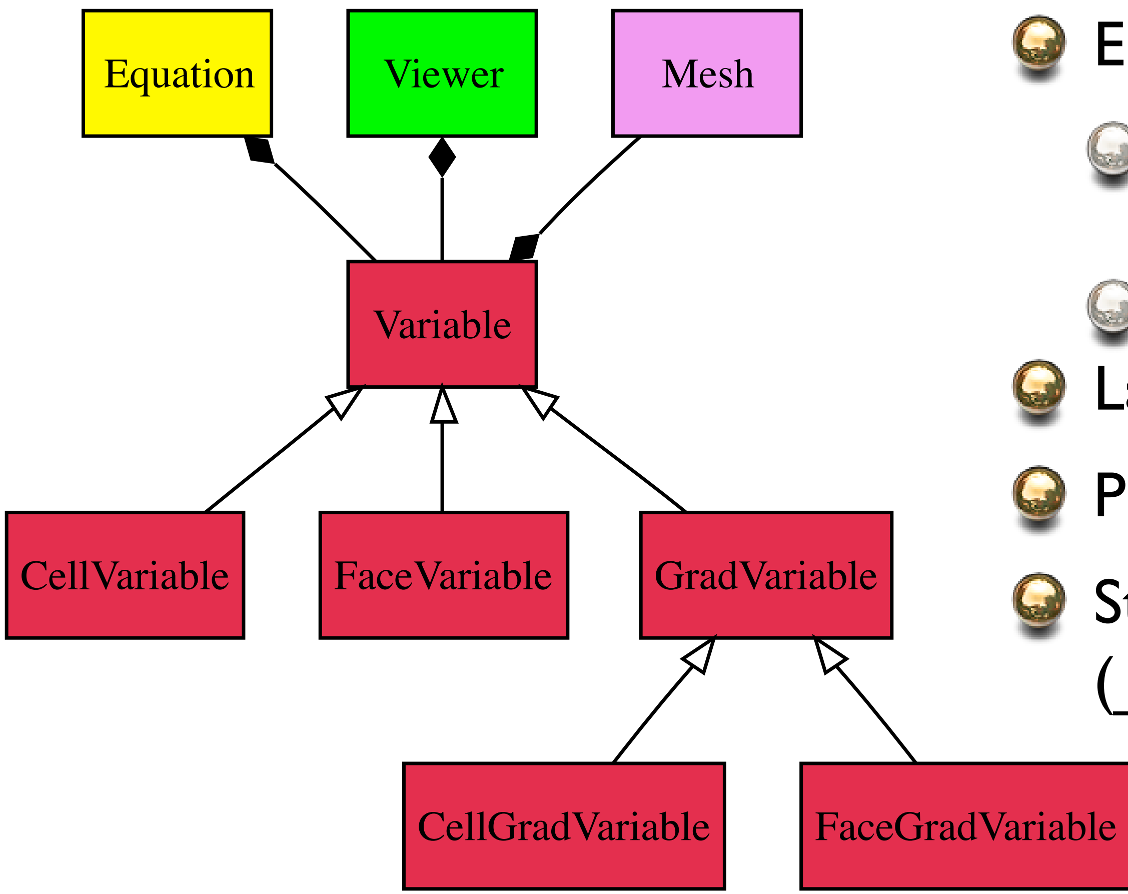




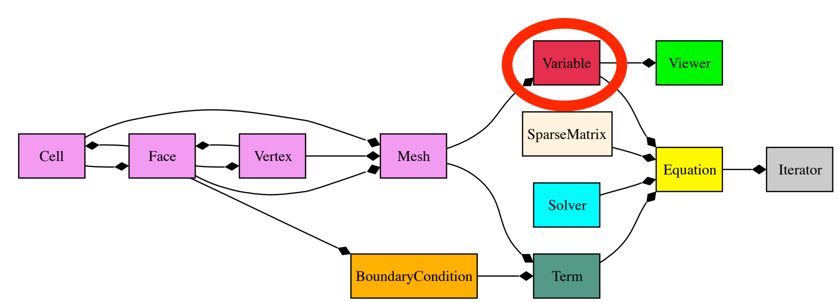
National Institute of Standards and Technology
Technology Administration, U.S. Department of Commerce

FiPy Design - Variables

$$\underbrace{\frac{\rho\phi V - (\rho\phi V)^{\text{old}}}{\Delta t}}_{\text{transient}} + \underbrace{\sum_{\text{face}}[(\vec{n}\cdot\vec{u})A\phi]_{\text{face}}}_{\text{convection}} = \underbrace{\sum_{\text{face}}[\Gamma A\vec{n}\cdot\nabla\phi]_{\text{face}}}_{\text{diffusion}} + \underbrace{VS_\phi}_{\text{source}}$$



- Either:
  - solution variables (evaluated by `Equation`)
  - set by intermediate calculation
- Lazy evaluation
- Physical dimensions
- Standard operators apply (`__add__`, `__div__`, etc.)



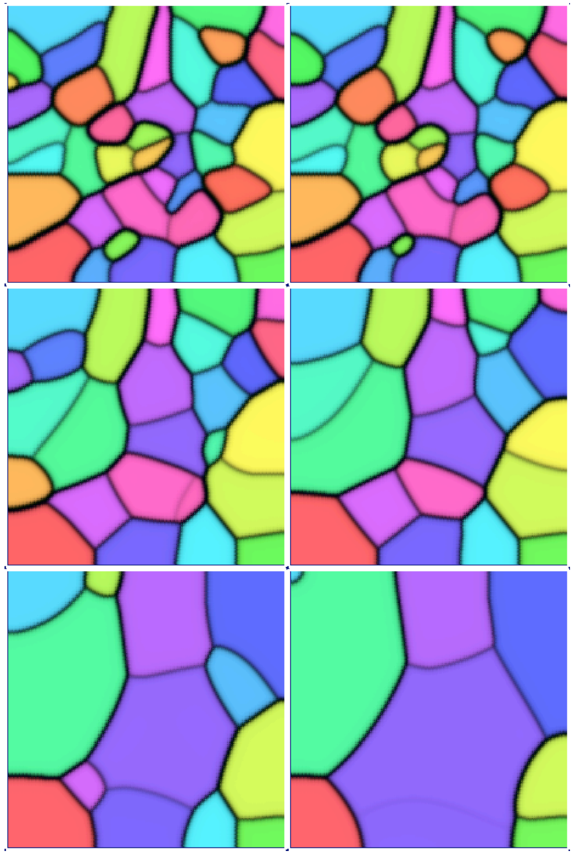
NIST
National Institute of Standards and Technology
Technology Administration, U.S. Department of Commerce

Example Problem - Grain Impingement
Governing Equations

- Phase field variable
  - $0 \leq \phi \leq 1$

$$\underbrace{Q(\phi, \nabla\theta)\frac{\partial \phi}{\partial t}}_{\text{transient}} = \underbrace{\alpha^2 \nabla^2 \phi}_{\text{diffusion}} - \underbrace{\frac{\partial f}{\partial \phi} - \frac{\partial g}{\partial \phi} s |\nabla \phi| - \frac{\partial h}{\partial \phi}\frac{\epsilon^2}{2}|\nabla \phi|^2}_{\text{source}}$$

R. Koybayashi, W. C. Carter, and J. A. Warren



- Orientation variable
  - $-\pi \leq \theta \leq \pi$

$$\underbrace{P(\phi, \nabla\theta)\frac{\partial \theta}{\partial t}}_{\text{transient}} = \underbrace{\nabla \cdot \left[h \epsilon^2 \nabla \theta + g s \frac{\nabla \theta}{|\nabla \theta|}\right]}_{\text{diffusion}}$$

NIST
National Institute of Standards and Technology
Technology Administration, U.S. Department of Commerce

Example Problem - Input File

```
class ImpingementSystem:
    def __init__(self, n = 100, dx = 0.01, steps = 10, plotsteps = 5, timeStepDuration = 0.02):

      :
      :

      # create mesh
      mesh = Grid2D(dx, dy, nx, ny)

      # create variables
      :
      :

      # create viewers
      self.viewers = [Grid2DGistViewer(var = field, palette = 'rainbow.gp') for field in (phase,theta)]

      # create equations
      :
      :

      # create iterator
      self.it = Iterator(equations = (thetaEq, phaseEq))

      :
      :

    def run(self):
      :
      :
```

Example Problem - Input File

```
class ImpingementSystem:
    def __init__(self, n = 100, dx = 0.01, steps = 10, plotsteps = 5, timeStepDuration = 0.02):

      :
      :

      # create mesh
      mesh = Grid2D(dx, dy, nx, ny)

      # create variables
      phase = CellVariable(name = 'PhaseField', mesh = mesh, value = 0.)
      theta = ModularVariable(name = 'Theta', mesh = mesh, value = 0.)

      setCells = mesh.getCells(lambda cell: cell.getCenter()[0] > L/2.)
      phase.setValue(1.,setCells)

      # create viewers
      self.viewers = [Grid2DGistViewer(var = field, palette = 'rainbow.gp') for field in (phase,theta)]

      # create equations
      :
      :

      # create iterator
      self.it = Iterator(equations = (thetaEq, phaseEq))

      :
      :

    def run(self):
      :
      :
```

NIST
National Institute of Standards and Technology
Technology Administration, U.S. Department of Commerce

Example Problem - Input File

```
class ImpingementSystem:
    def __init__(self, n = 100, dx = 0.01, steps = 10, plotsteps = 5, timeStepDuration = 0.02):

         :
         :
         :

         # create equations
         thetaEq = ThetaEquation(var = theta,
            solver = LinearPCGSolver( tolerance = 1.e-15, steps = 2000 ),
            boundaryConditions = ( FixedFlux(mesh.getExteriorFaces(), 0.), ),
            parameters = {
               'time step duration' : timeStepDuration,
               'beta' : 1e5,
               'mu' : 1e3
            },
            fields = (phase,))

         phaseEq = PhaseEquation(var = phase,
            mPhi = Type1MPhiVariable,
            solver = LinearPCGSolver(tolerance = 1.e-15, steps = 1000),
            boundaryConditions = ( FixedFlux(mesh.getExteriorFaces(), 0.), ),
            parameters = {
               'tau' : 0.1,
               'time step duration' : timeStepDuration
            },
            fields = (theta,))

         # create iterator
         self.it = Iterator(equations = (thetaEq, phaseEq))

         :
         :
```

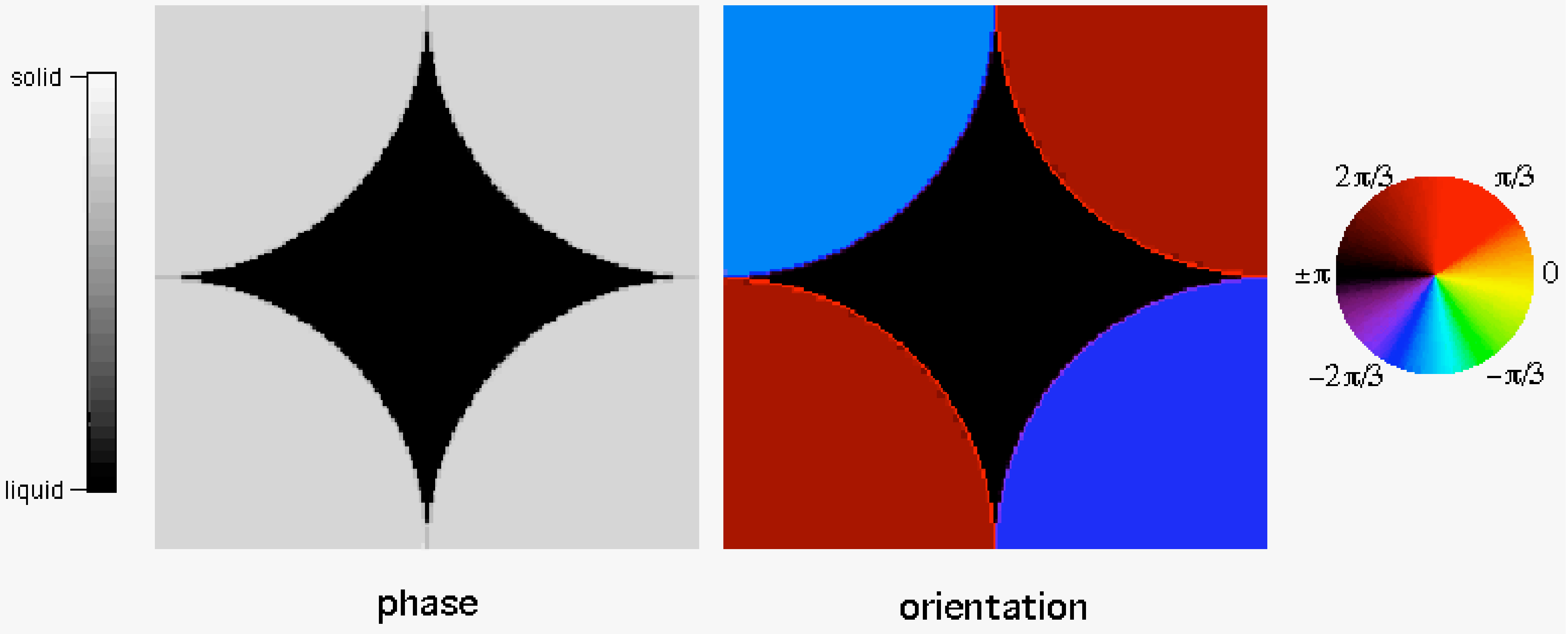
Example Problem - Input File

```
class ImpingementSystem:
    def __init__(self, n = 100, dx = 0.01, steps = 10, plotsteps = 5, timeStepDuration = 0.02):

       :
       :
       :
       :

    def run(self):
     for plotstep in range(self.plotsteps):
        for step in range(self.steps):
          self.it.timestep()
         for viewer in self.viewers:
          viewer.plot()
```

Example Problem - Grain Impingement



Example Problem - Efficiency

N	FORTRAN (s)	`Numeric` (s)	Penalty
100	0.0619	12.8	$\times 207$
400	0.188	40.2	$\times 214$
1600	0.573	157	$\times 273$
6400	3.06	707	$\times 231$
25600	12.0	4060	$\times 338$
102400	37.3	???	$\times$???
409600	148	???	$\times$???

- NumPy is much faster than plain Python, but still orders of magnitude slower than tailored FORTRAN code
- NumArray is faster for large arrays?
- Significant overhead in transition between Python and C for lots of simple operations

$$\vec{A} = \vec{B} \times \vec{C} + \tan \vec{D}$$

Example Problem - Efficiency

N	FORTRAN (s)	`Numeric` (s)	Penalty	Inline C (s)	Penalty
100	0.0619	12.8	$\times 207$	2.45	$\times 39.6$
400	0.188	40.2	$\times 214$	2.95	$\times 15.3$
1600	0.573	157	$\times 273$	5.22	$\times$ 9.10
6400	3.06	707	$\times 231$	15.3	$\times$ 4.99
25600	12.0	4060	$\times 338$	66.0	$\times$ 5.53
102400	37.3	???	$\times$???	243	$\times$ 6.50
409600	148	???	$\times$???	1510	$\times 10.2$

- Costly operations are not neatly encapsulated in a generalized library
- C code is embedded within Python code using SciPy's `weave`
- Design in Python, optimize in C
- Python and C variants are maintained in parallel (`--inline` at command line)

Example Problem - Efficiency

N	FORTRAN (s)	`Numeric` (s)	Penalty	Inline C (s)	Penalty
100	0.0619	12.8	×207	2.45	×39.6
400	0.188	40.2	×214	2.95	×15.3
1600	0.573	157	×273	5.22	× 9.10
6400	3.06	707	×231	15.3	× 4.99
25600	12.0	4060	×338	66.0	× 5.53
102400	37.3	???	×???	243	× 6.50
409600	148	???	×???	1510	×10.2

- Maintaining clean *and* fast code is challenging
  - `blitz` isn't fast enough
  - `weave` is a bit clumsy
  - Swig?
  - Pyrex? 10:30 am Thursday

Further Work

- Meshing packages
- DX viewer
- Documentation
- Simplify installation
- Even more test cases
- Level Sets
- ???