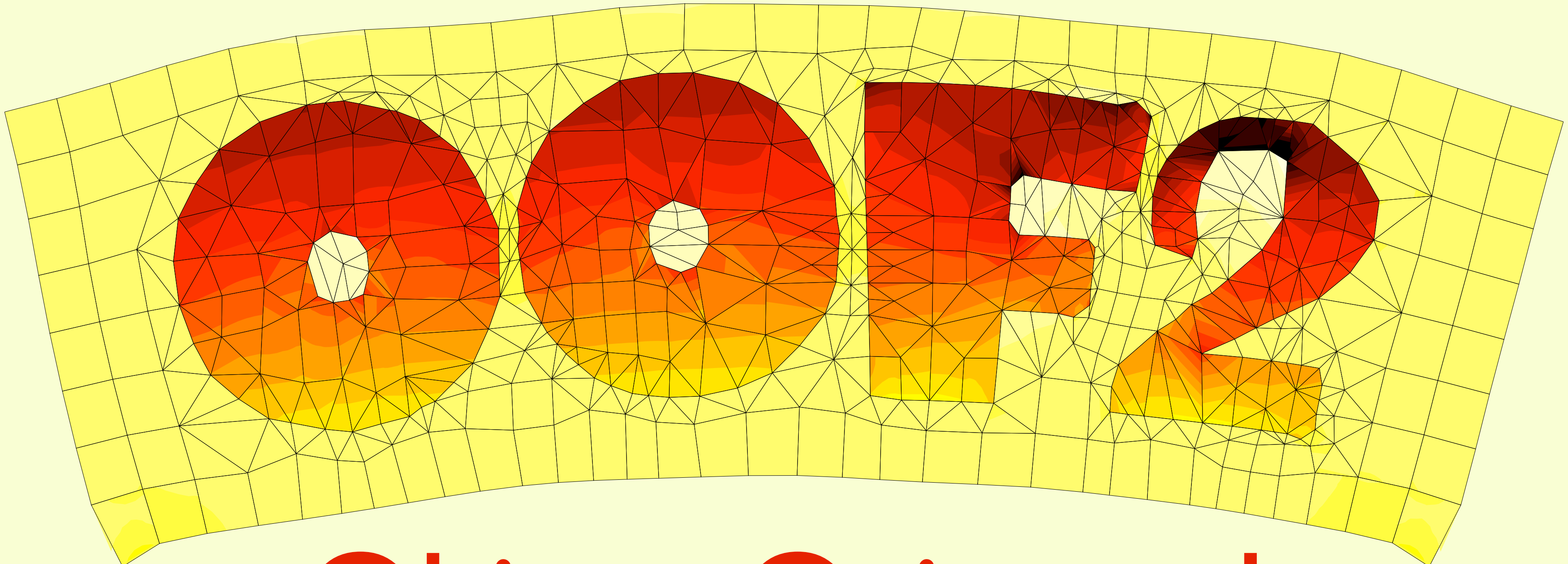


Object-Oriented Finite Element Analysis of Material Microstructures

Stephen A. Langer
Mathematical and Computational Sciences Division
Information Technology Laboratory
National Institute of Standards and Technology

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

Personnel

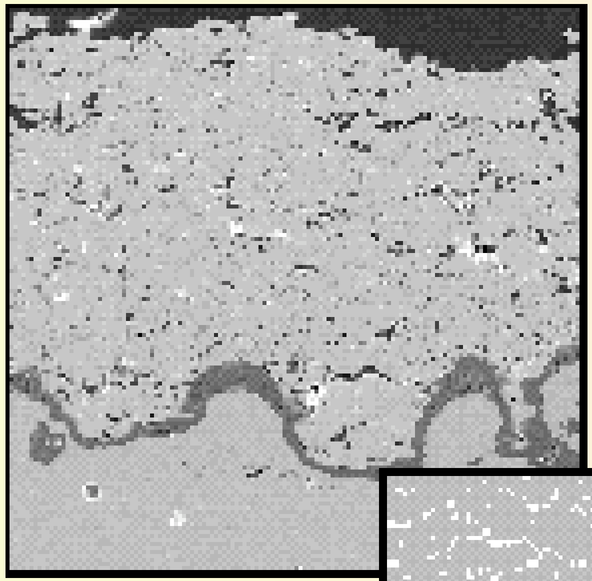
Steve Langer	NIST ITL MCSD
Andrew Reid*	Drexel University (ITL $)
Seung-Ill Haan*	U. Md, Baltimore County (CTCMS $)
Edwin Garcia*	Penn State University (NSF & CTCMS $)
Andrew Roosen	NIST MSEL
Eric Ma Kevin Chang Kang-Xing Jin Daniel Vlacich	Montgomery Blair High School Math & Science Magnet Program
Kyle Stemen	SURF, Kent State University
Edwin Fuller	NIST MSEL
W. Craig Carter	MIT
Zi-Kui Liu	PSU
Panos Charalambides	UMBC

* Guest Researchers at the
NIST Center for Theoretical and Computational Materials Science

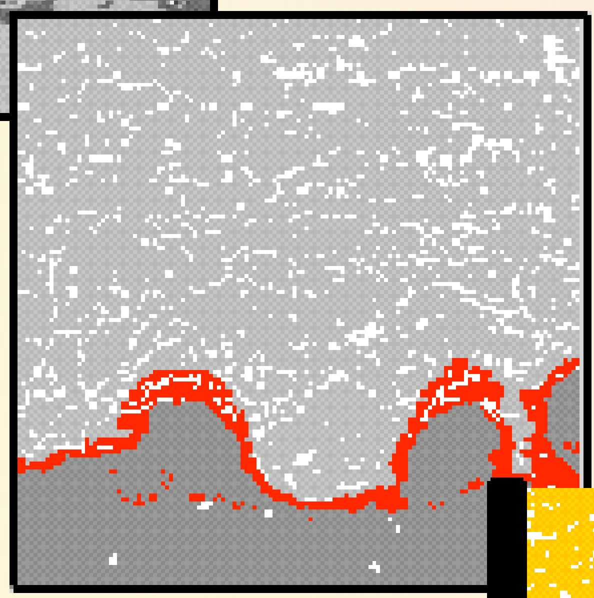
Outline of the talk

- What?
- Examples
- OOF2
  - Why?
  - How?
    - Design Goals
    - Ingredients

What is OOF?

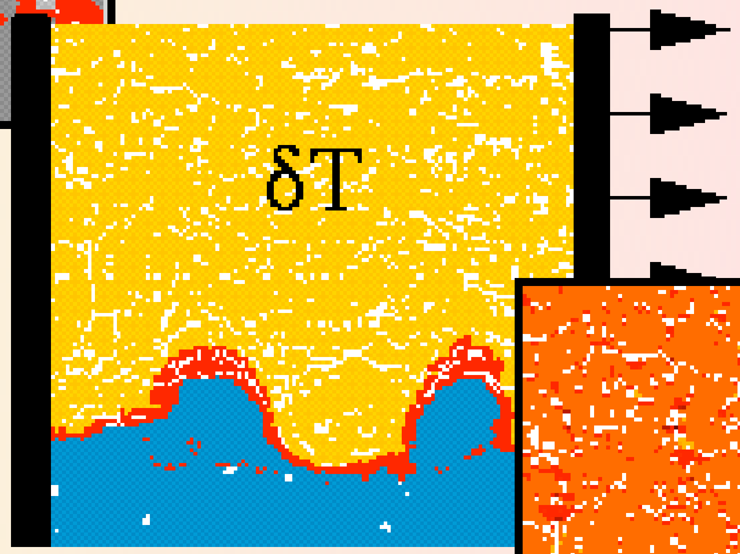


1. *Start with a micrograph*

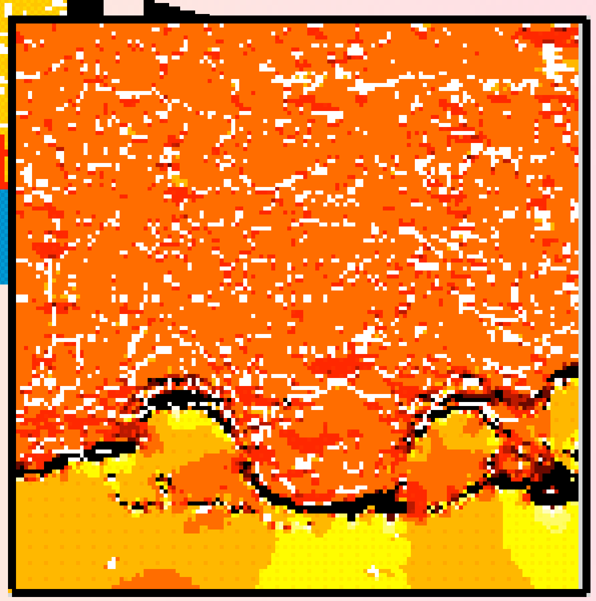


2. *Assign material properties*

3. *Perform virtual experiments*

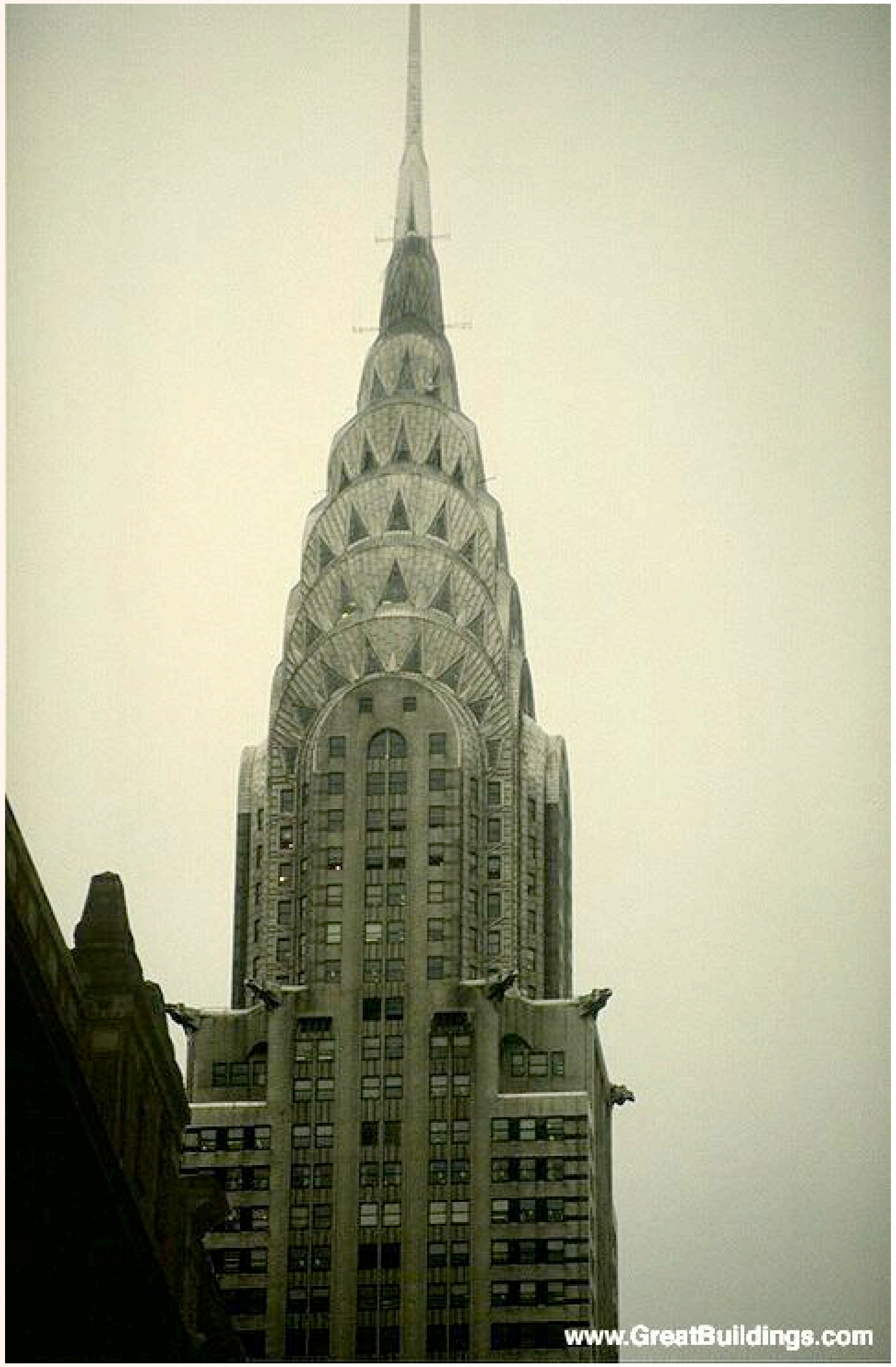


4. *Visualize and quantify*



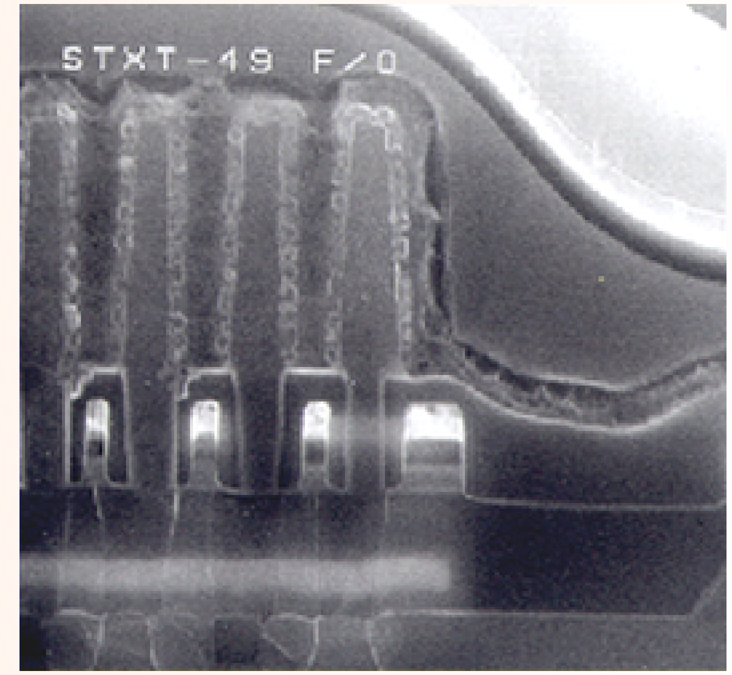
Why OOF?

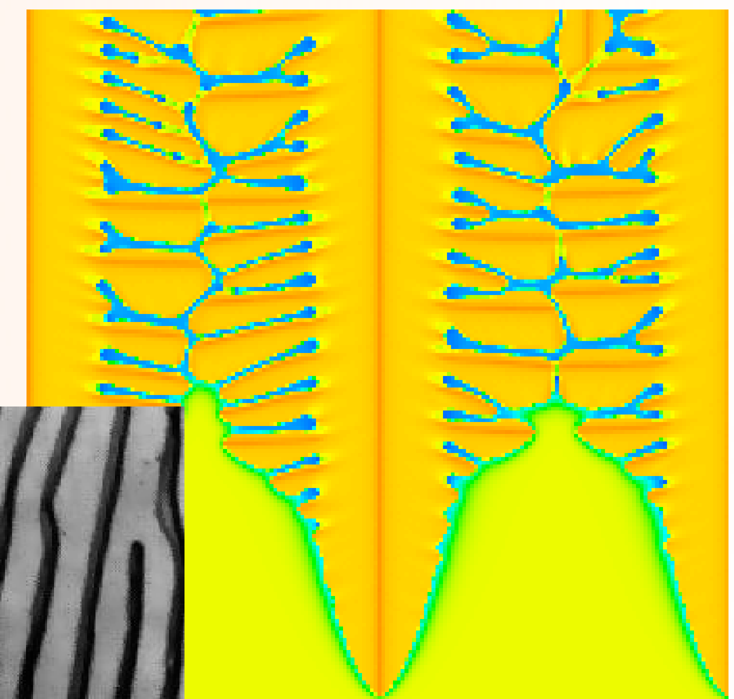
- Commercial finite element packages work best with large scale systems with regularly shaped components.

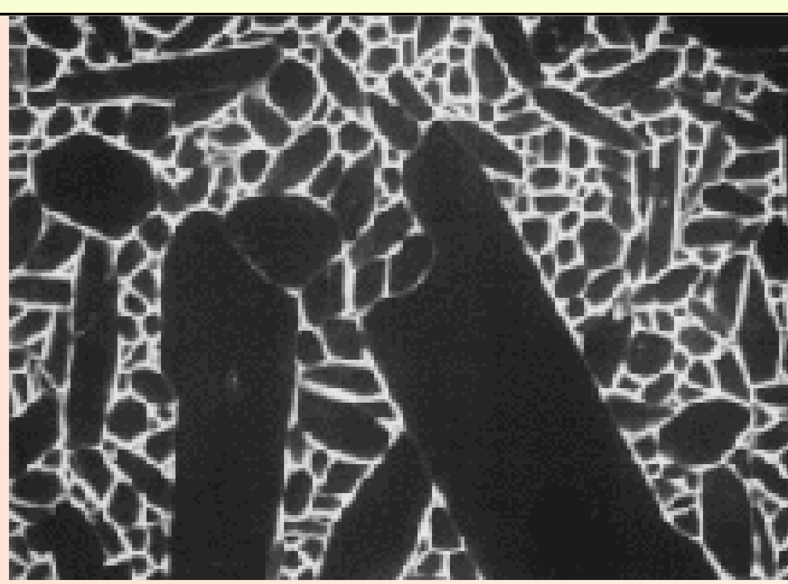


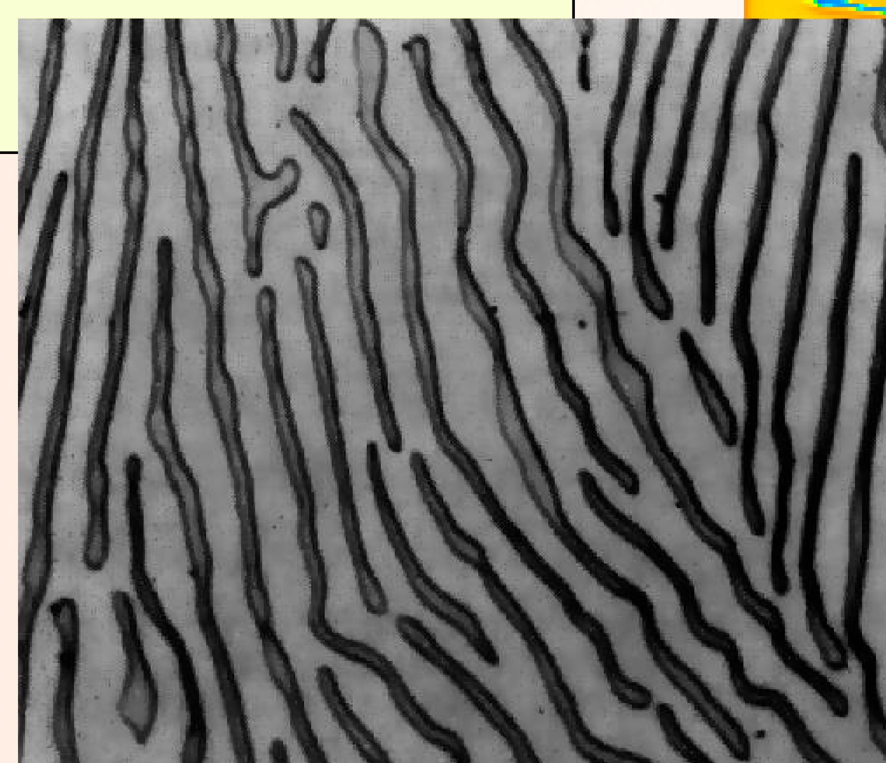
Why OOF?

- Commercial finite element packages work best with large scale systems with regularly shaped components.
- Materials systems are small scale and disordered.



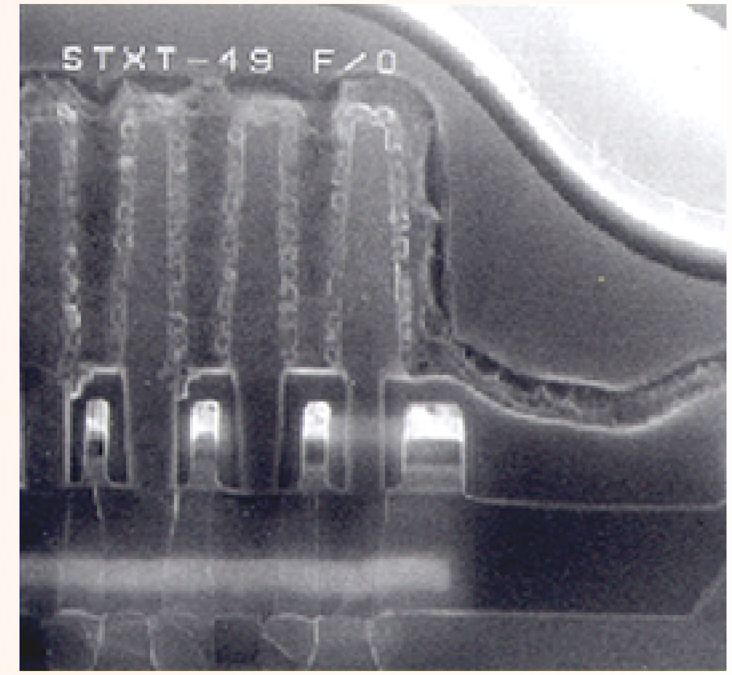


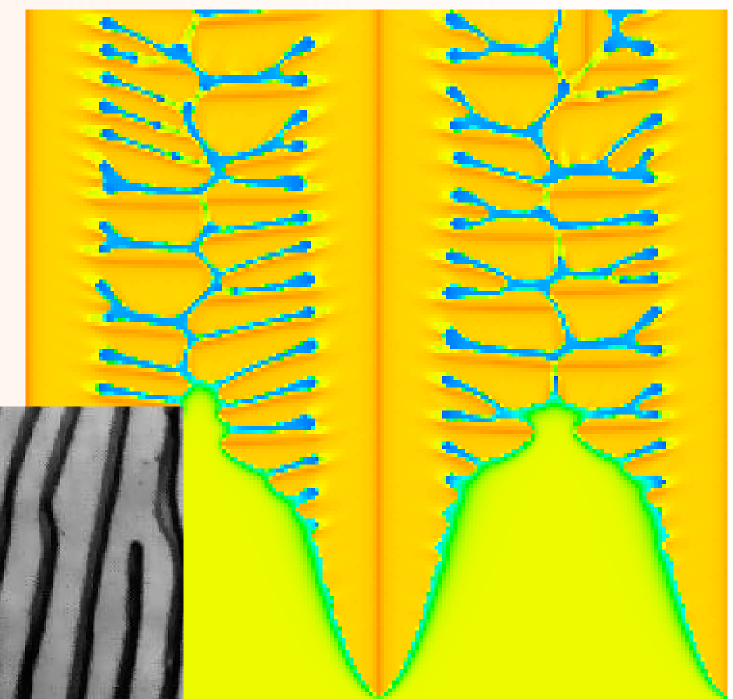


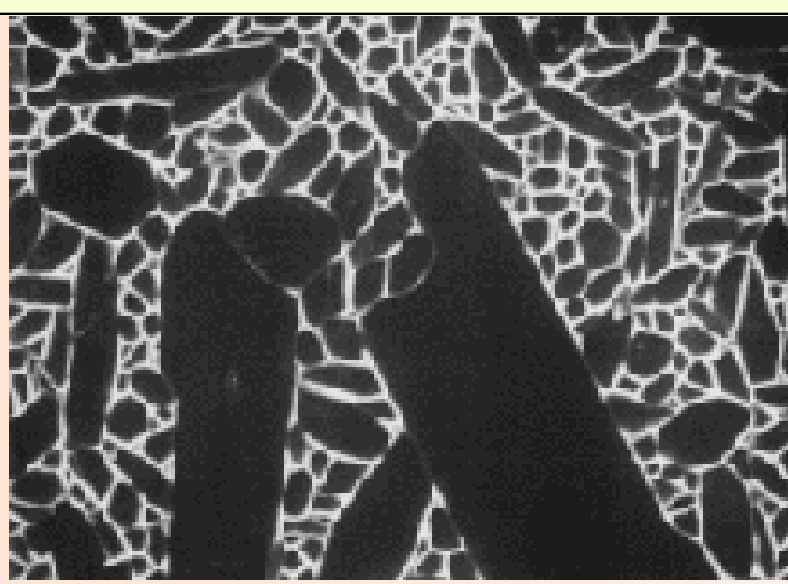


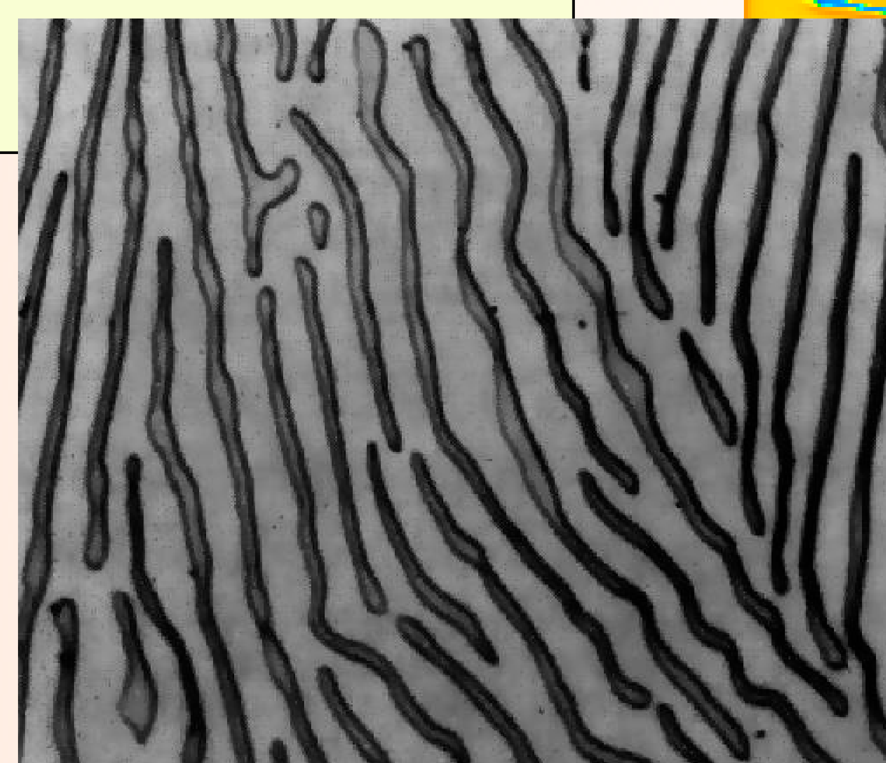
Why OOF?

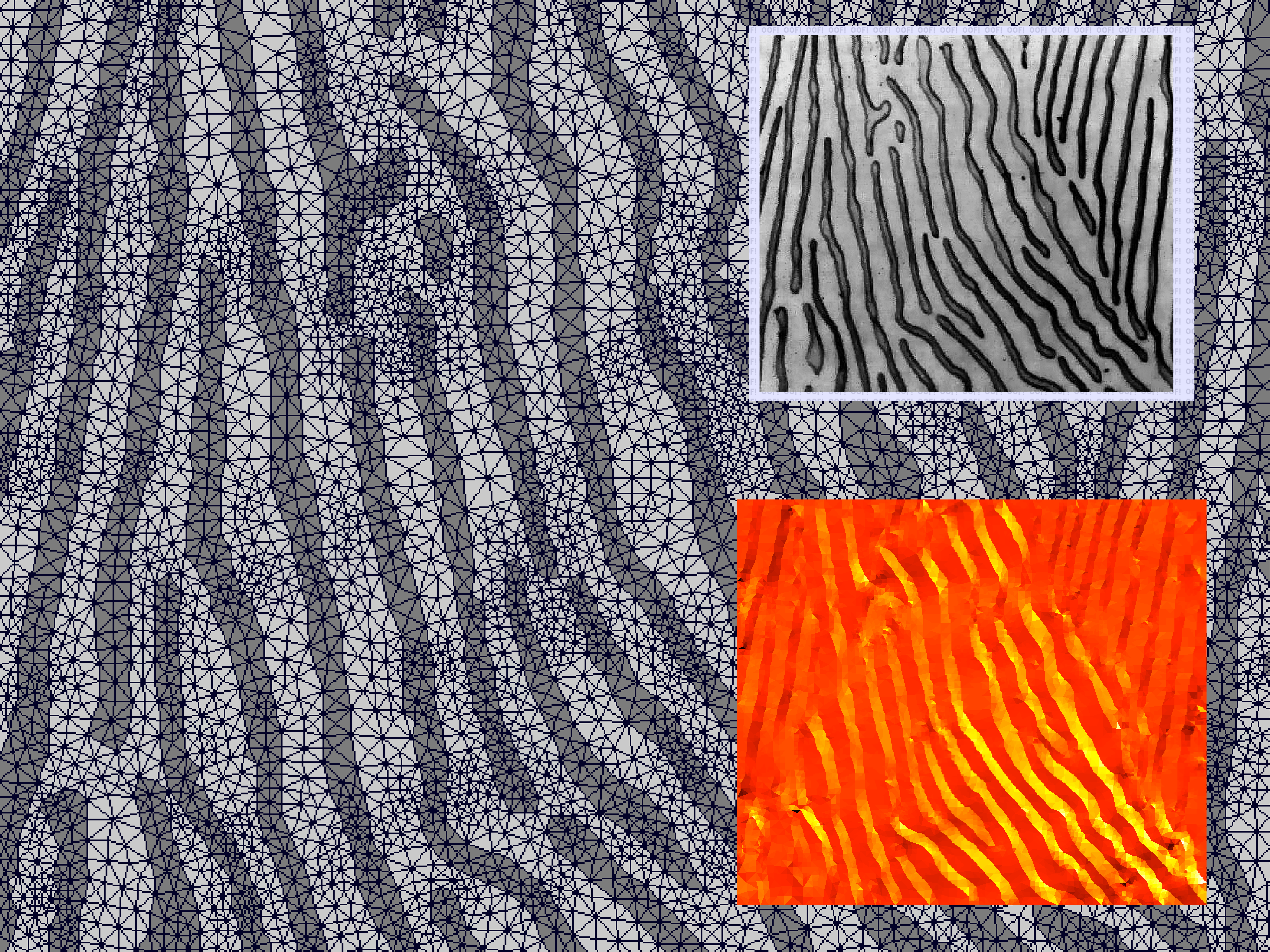
- Commercial finite element packages work best with large scale systems with regularly shaped components.
- Materials systems are small scale and disordered.
- OOF is designed to answer the questions that materials scientists want to ask.
- OOF is easy to use.



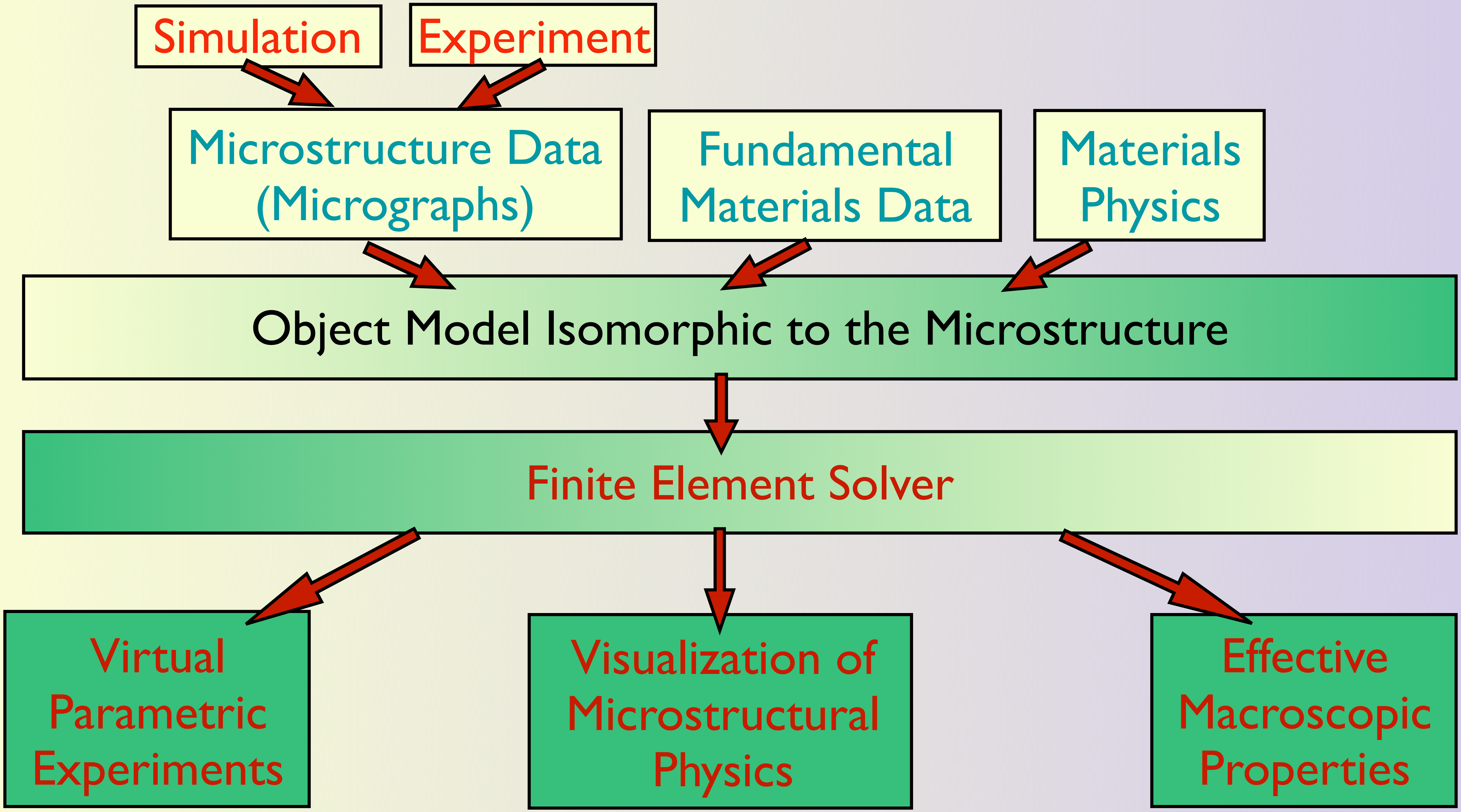








CONCEPTUAL ORGANIZATION



Easy-to-use Graphical User Interface

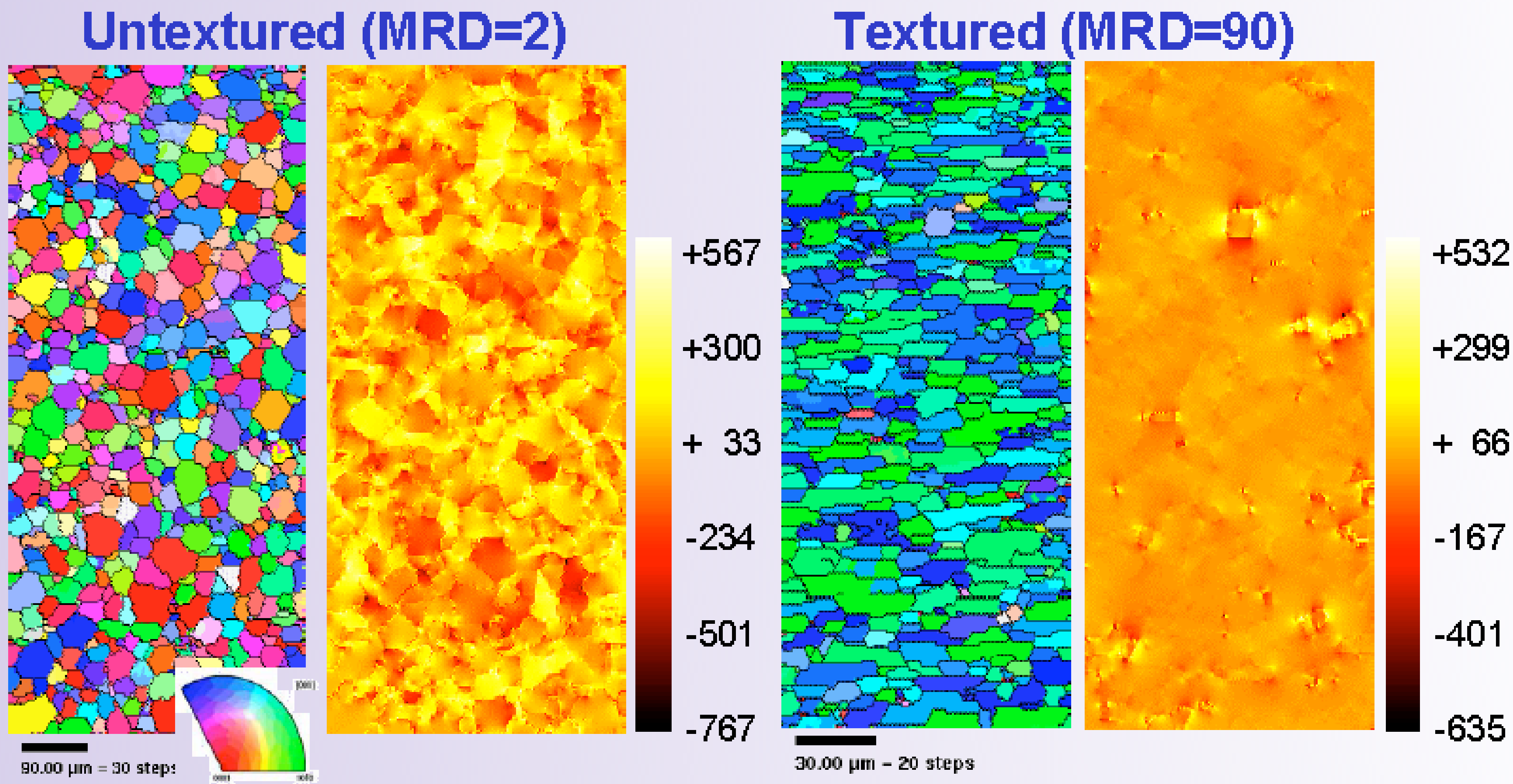
- Example Applications:
  - Thermal Barrier Coatings
  - Residual Stresses in Alumina
  - Marble
  - Piezoelectrics
  - Batteries

Predict Thermal Conductivity κ of Ceramic Thermal Barrier Coatings for Turbine Blades

with James Ruud, NS Hari, James Grande, and Antonio Mogro-Campero,
GE Corporate R&D
Funded in part by DOE Advanced Turbine Systems Program

- TBC's allow jet engine blades to operate at higher temperatures.
- Physical measurements of κ are difficult, time consuming and expensive. Hardly ever done during quality control.
- OOF could replace measurements during research, development, design, and production.

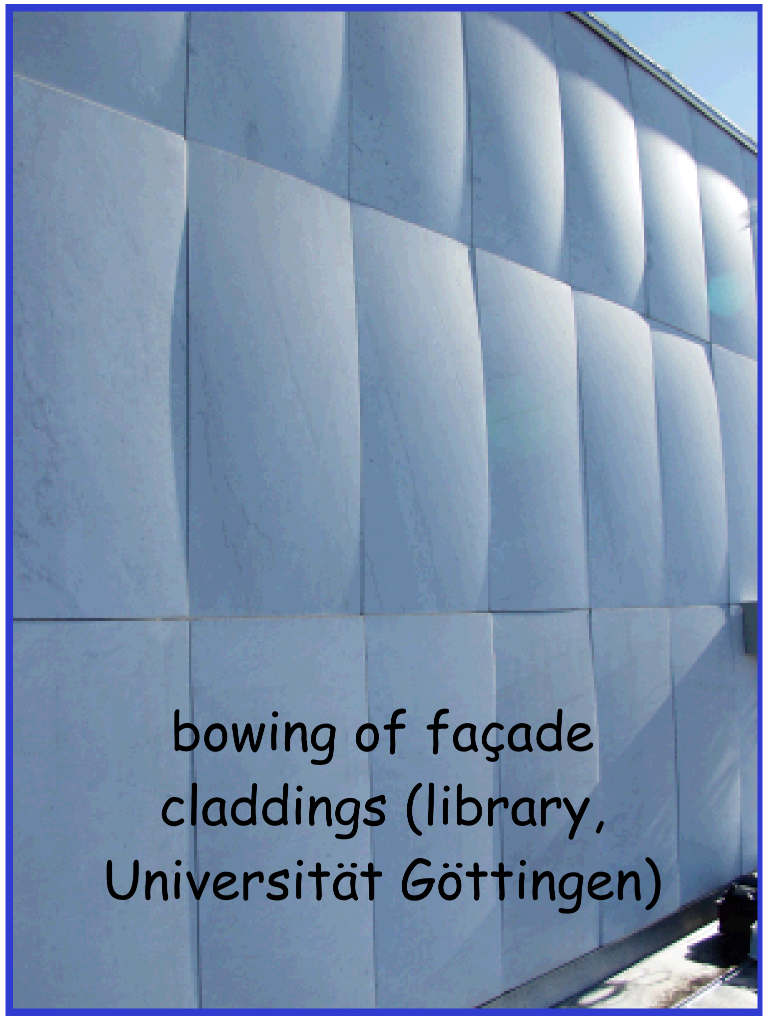
Residual Stresses in Alumina



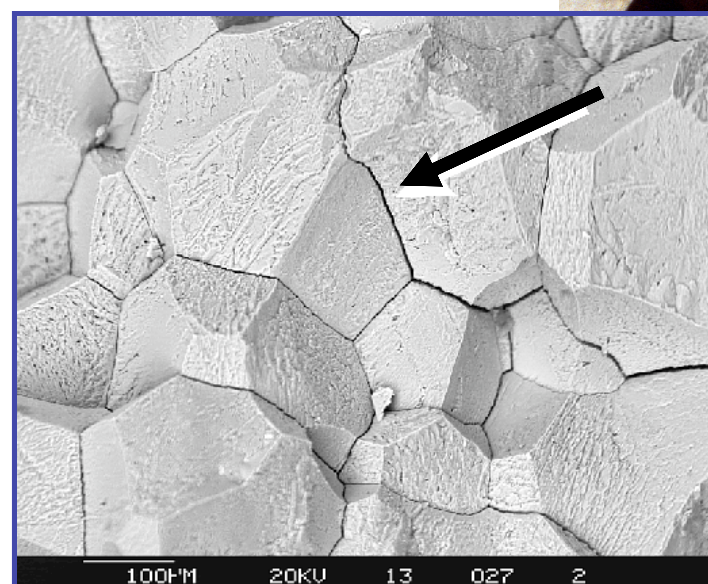
Stress invariant 1 ($\sigma_{11}+\sigma_{22}$) shown for ΔT = -1500°C.
Calculations under plane stress and free boundary conditions.
Total number of elements = 117612.

Venkata Vedula, Sandia

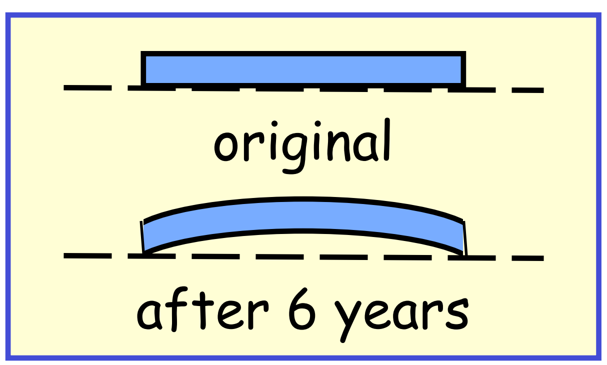
Thermal Degradation of Decorative Marbles



bowing of façade claddings (library, Universität Göttingen)



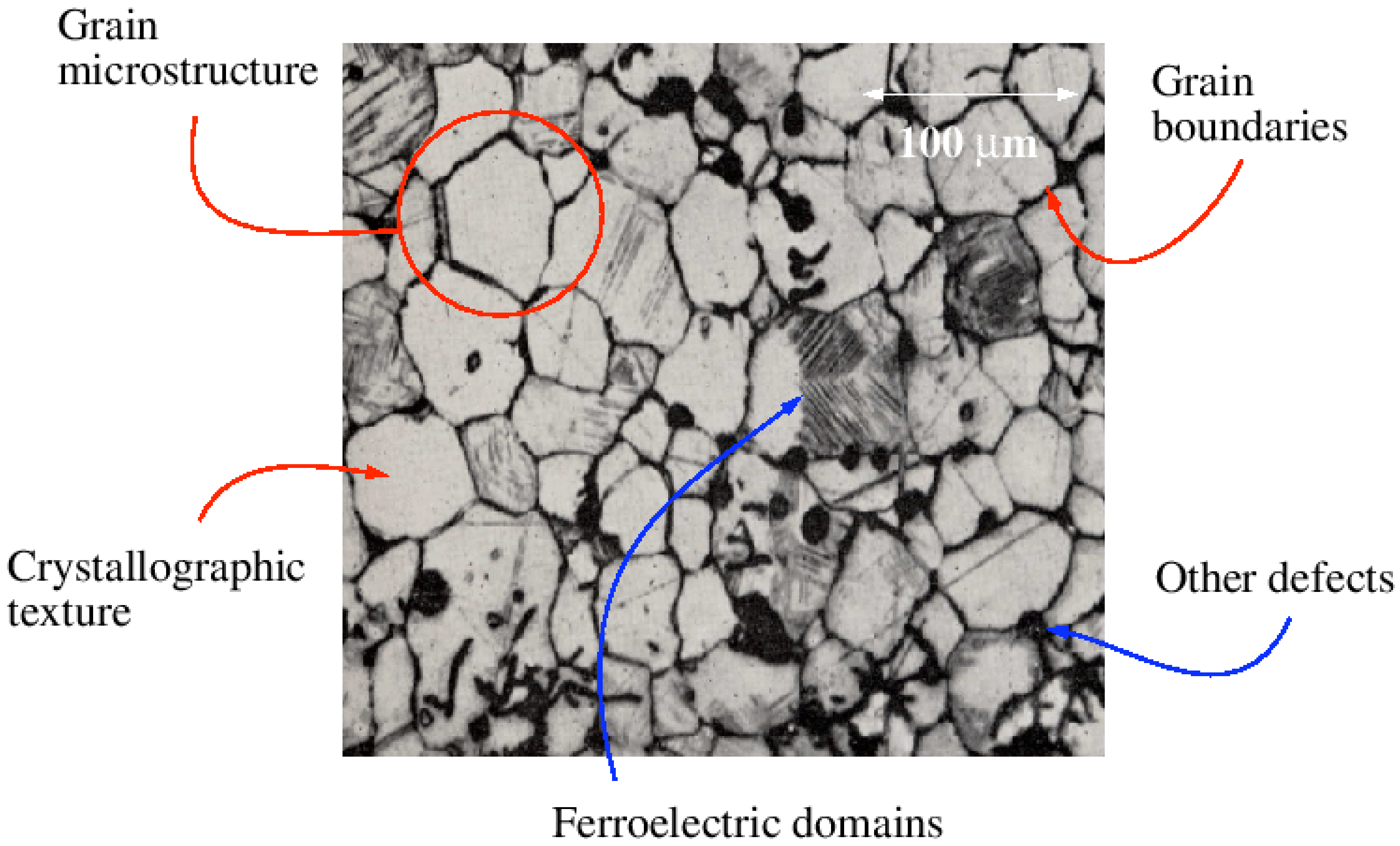
granular disintegration



Thomas Weiß and Siegfried Siegesmund, Universität Göttingen, Germany

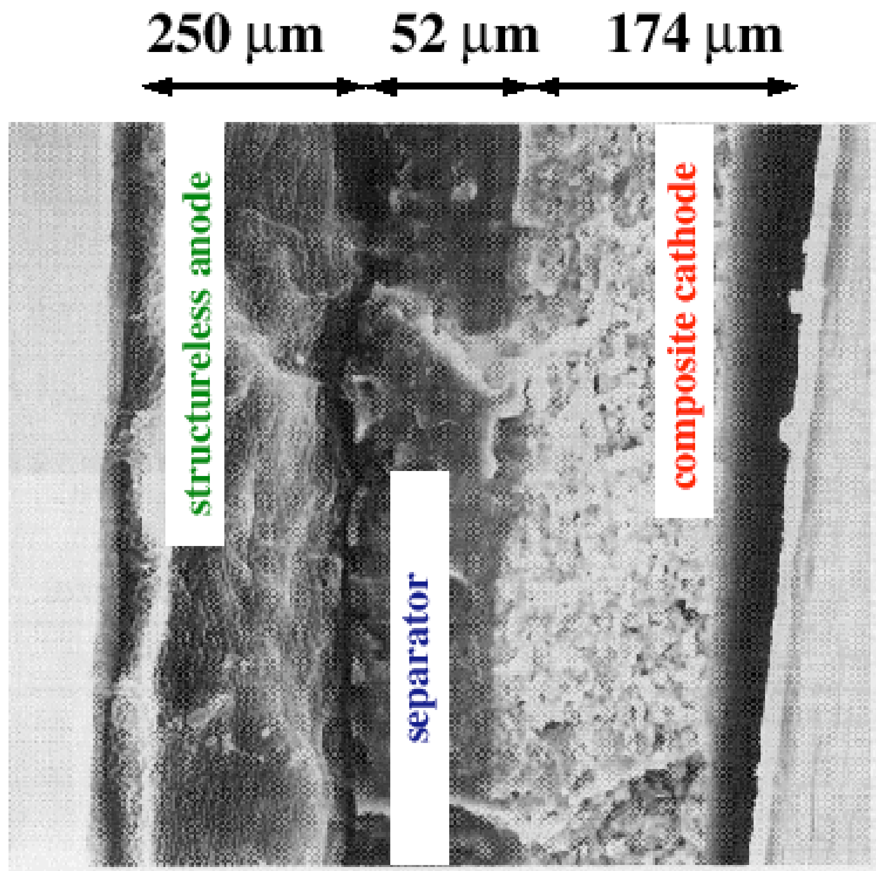
Microstructural Effects in Polycrystalline Piezoelectrics

Edwin Garcia



Jaffe, B. Cook, W. Jaffe, H. "Piezoelectric Ceramics" Academic Press, London, 1971, pg. 67

Microstructural Design of Rechargeable Batteries

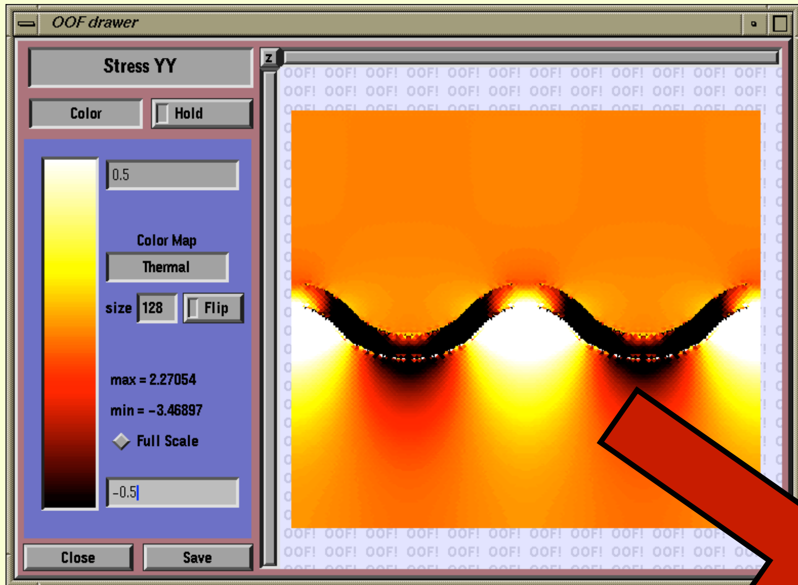


R. Edwin Garcia, Catherine M. Bishop, W. Craig Carter*, Stephen A. Langer†
Pimpa Limthongkul, and Yet–Ming Chiang

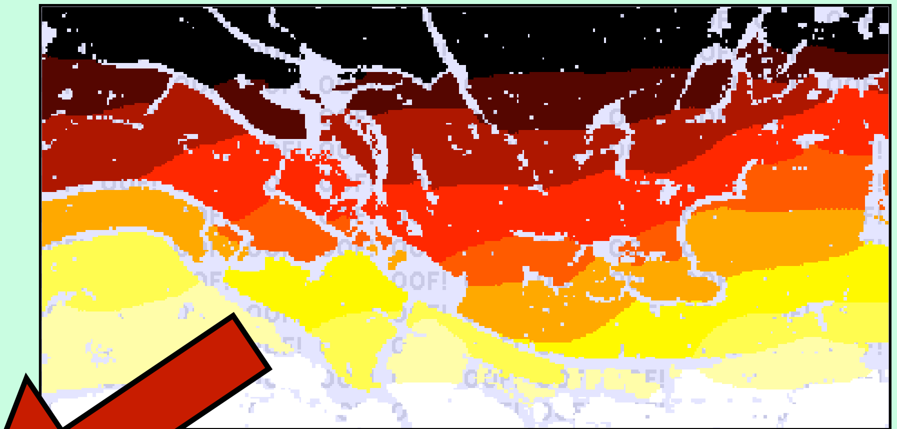
* Massachusetts Institute of Technology

† National Institute of Standards and Technology

Why OOF2?

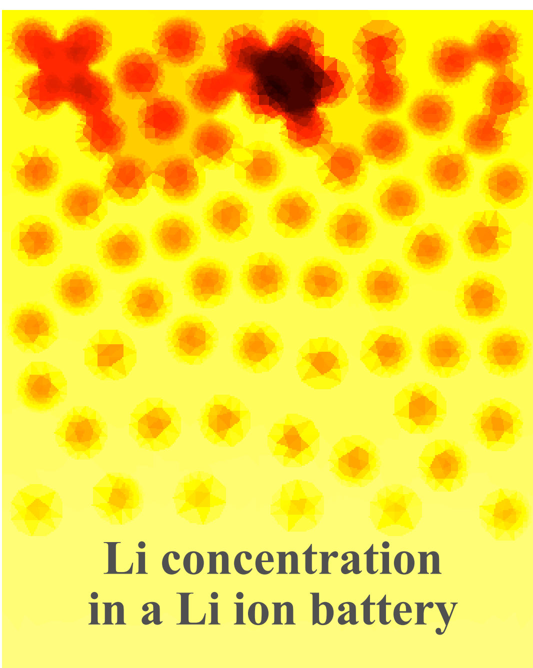


Classic OOF1 (elasticity)



Thermal OOF1
asticity & thermal diffusion)

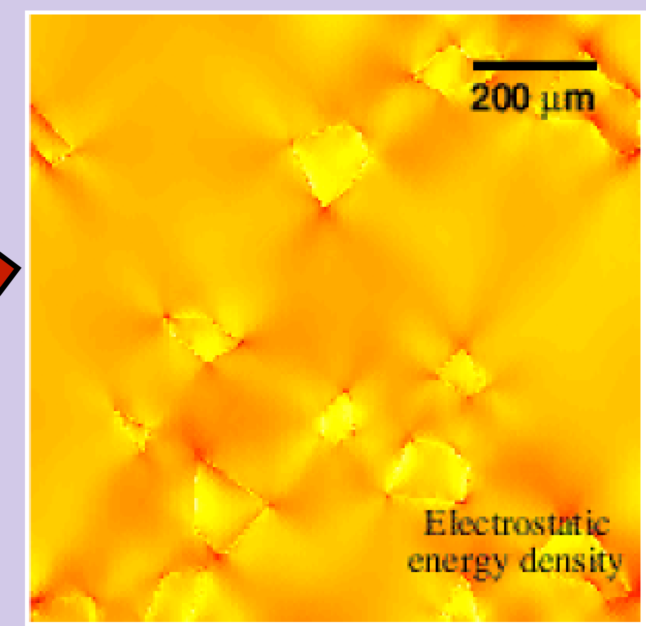
OOF2



Electrochemical OOF1
(time dependent, nonlinear)

?

??



Electromechanical OOF1
(adaptive refinement, nonlinear)

Why OOF2?

OOF2 reflects lessons learned from OOF1.

- More expandable.
- More flexible.

Emphases:

- *Extensibility* and *maintainability* through proper object-oriented design reflecting the underlying mathematics.
- *Generality* by making few assumptions about the problems being solved.
- *Usability* with a clear user interface.
- *Sanity* with a flexible infrastructure.

OOF1	OOF2
Separate mesh generation & solver programs	Single program
C++	C++ & Python
Unthreaded, single processor	Threaded, parallel processors (soon)
Extended with difficulty	Easily extendible
Fixed physics	Arbitrary couplings
Linear triangular elements	Higher order triangles & quads
	More tools, more outputs, more, more, more (I'm still not satisfied)

OOF2

- Easily extendible to a wide variety of problems
  - elasticity, plasticity, thermal conductivity, mass diffusion, electrical polarization, piezoelectricity, ferroelectricity, Darcy's Law fluid flow, …

$$\sigma = \sum_i k_i \nabla \phi_i \qquad -\nabla \cdot \sigma = f$$

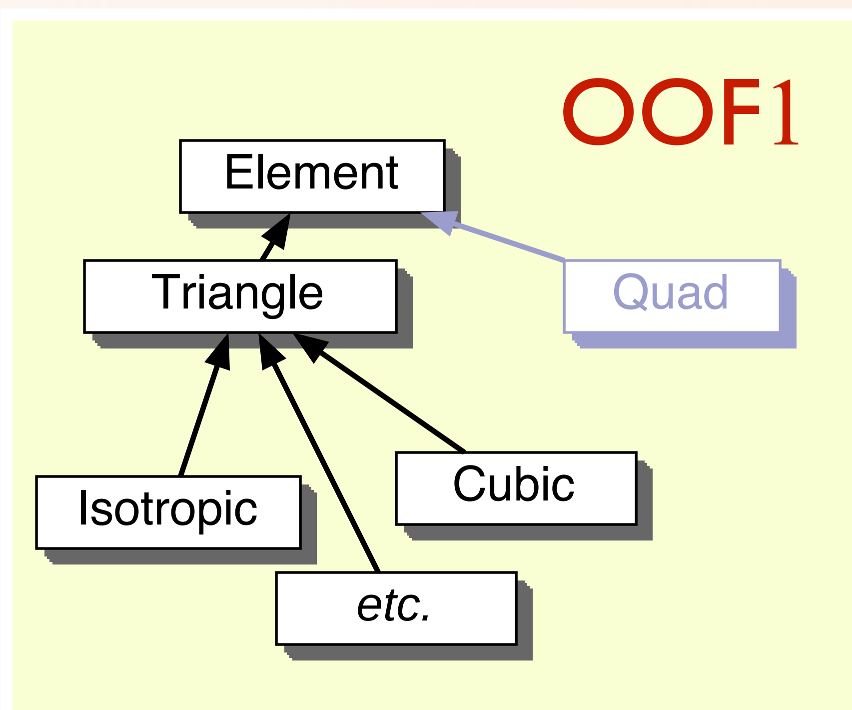
SCHEMATIC

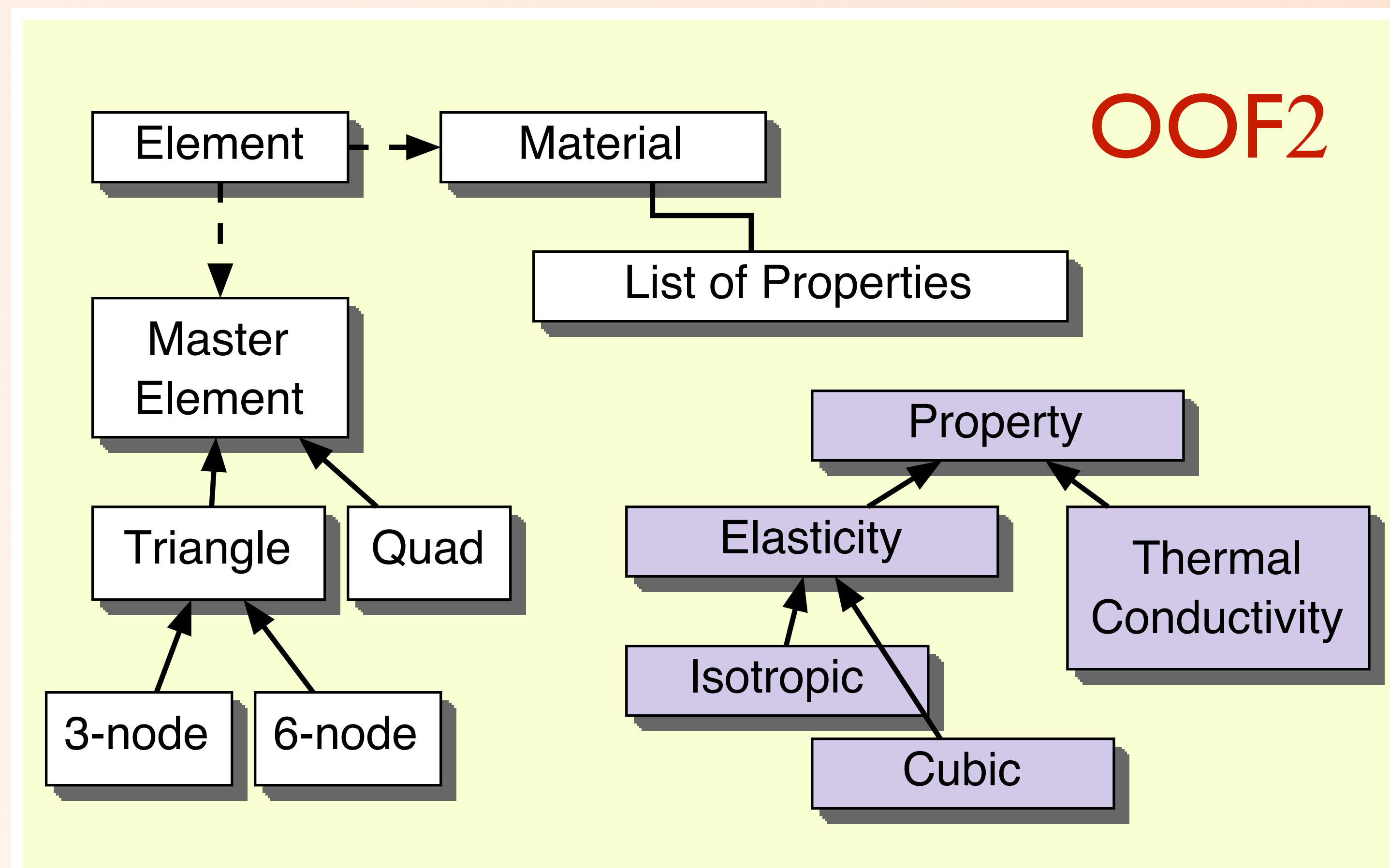
	Elasticity	Thermal Cond.	?
Field Φ	displacement	temperature	?
Flux σ	force	heat flow	?
Modulus k	C_{ijkl}	κ_{ij}	?
Force f	force	heat source, sink	?

- Designed for simple addition of new fields, fluxes, and equations.

Why OOF2?

- For example (proper design):
  - Physics and Finite Element class structure more closely tied to the underlying mathematics.
  - Allows more physics *and* more types of finite elements.





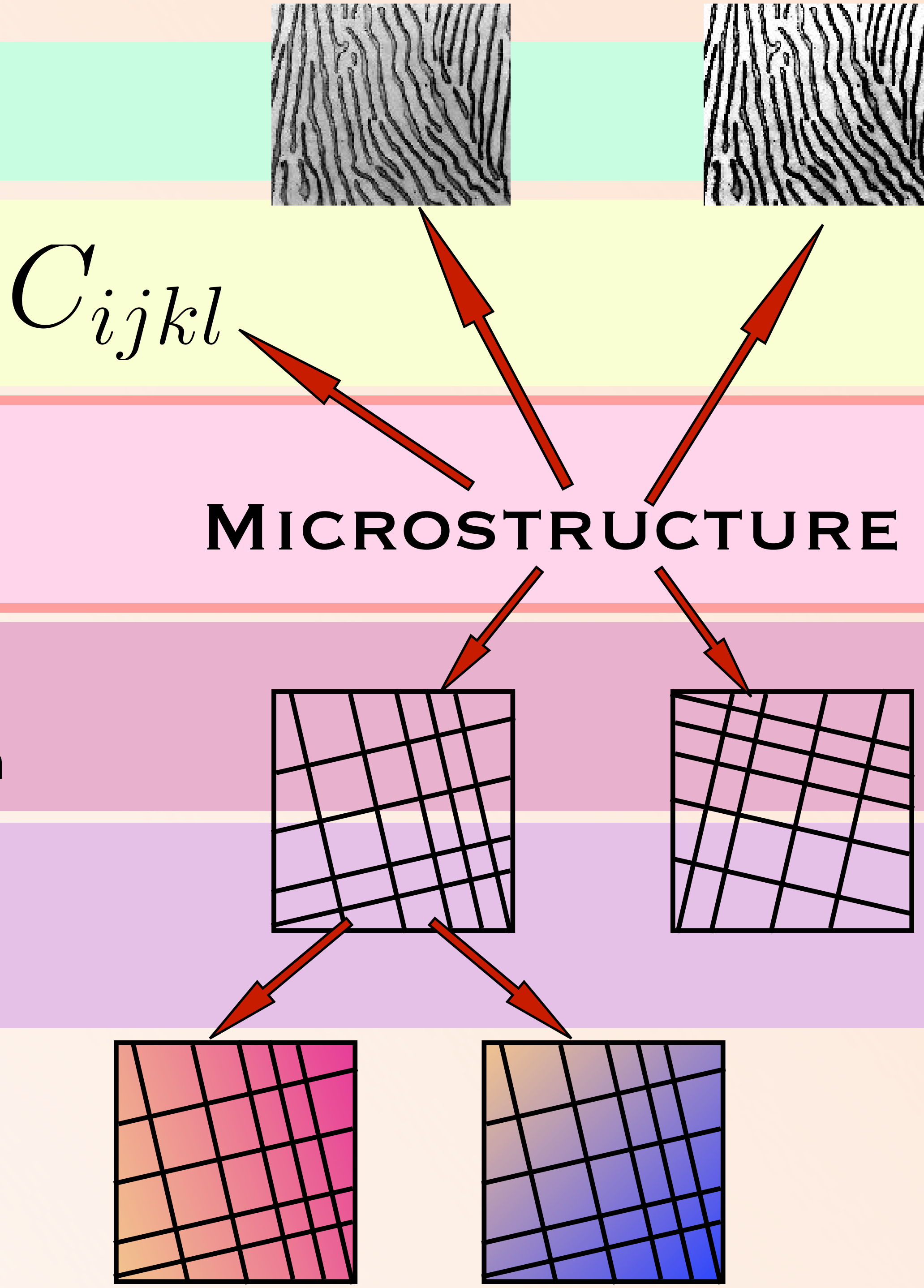
Properties can be coded completely independently from the element classes.

OOF2 Code Ingredients

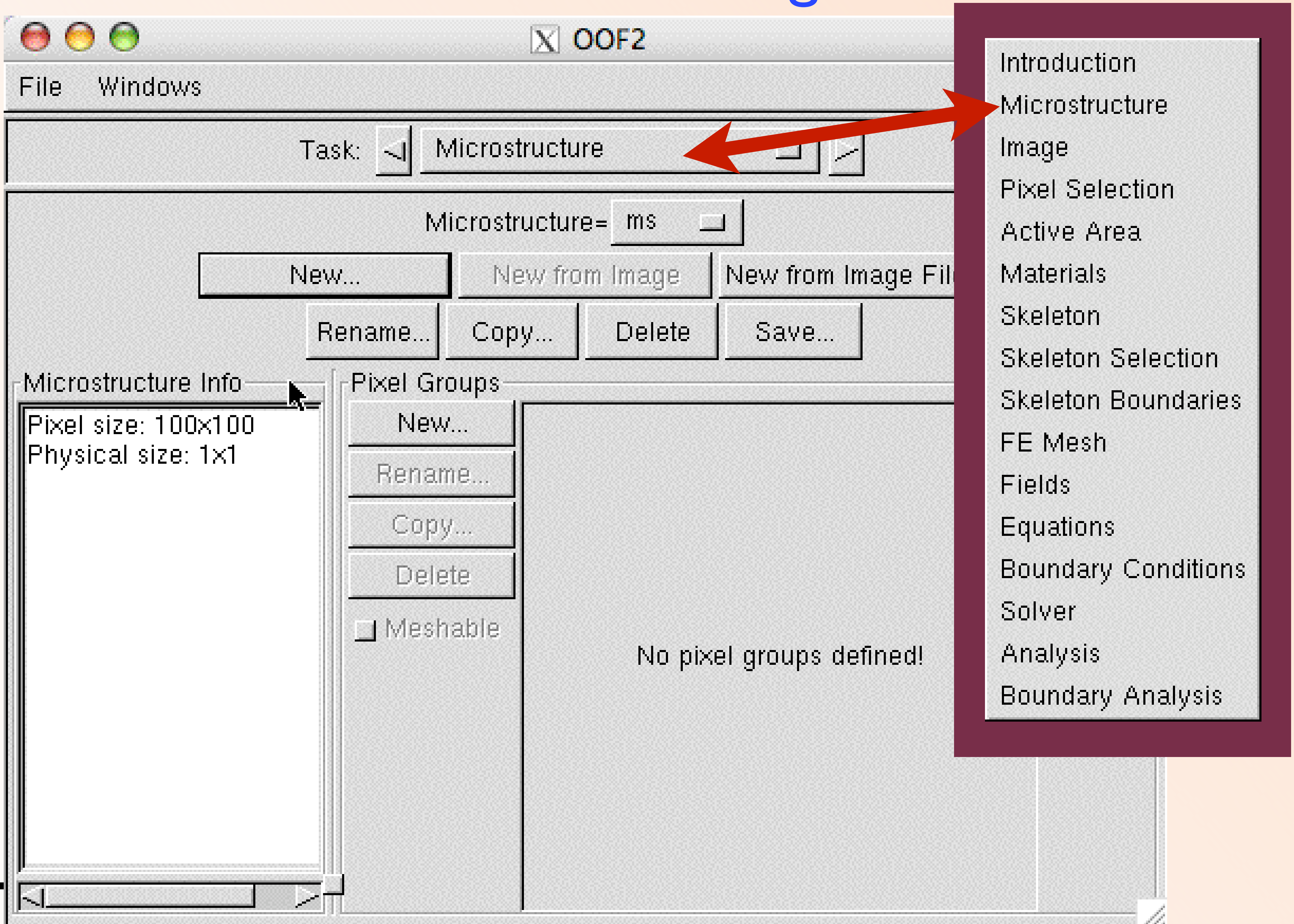
- C++ (core) and Python (interface).
- C++/Python glue code generated by SWIG.
- Libraries:
  - GTK+ graphics.
  - PETSc, MPI parallel solvers.
  - ImageMagick image manipulation.
  - IML++, MV++, SparseLib++ linear algebra.
- Threading

OOF2 Conceptual Ingredients

- image
- materials
  - assembled from lists of properties
- microstructure
  - materials assigned to groups of pixels
- skeleton
  - only the geometry of the finite element mesh
- mesh
  - skeleton + mathematics + physics
- solution



Interface leads users through the tasks



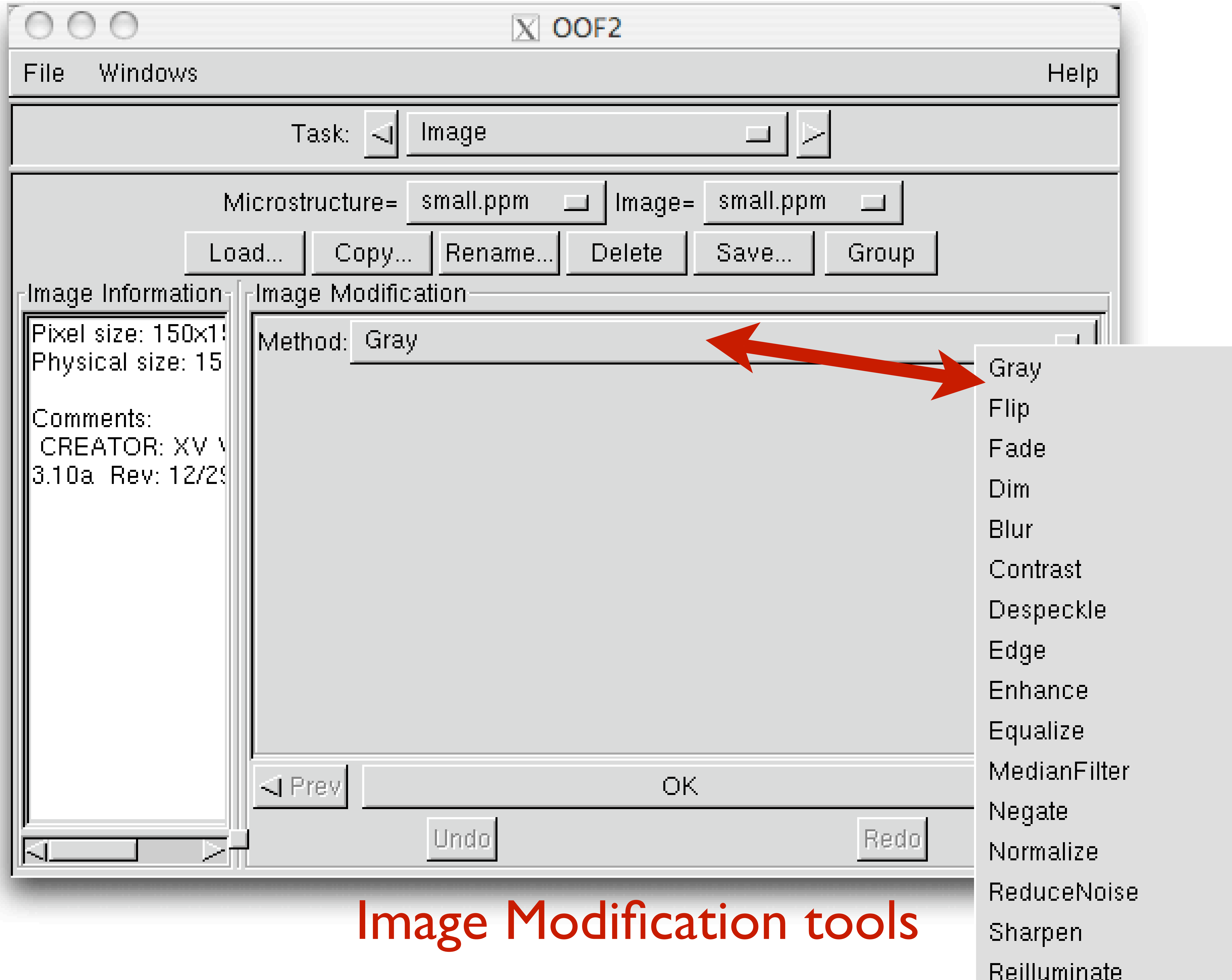
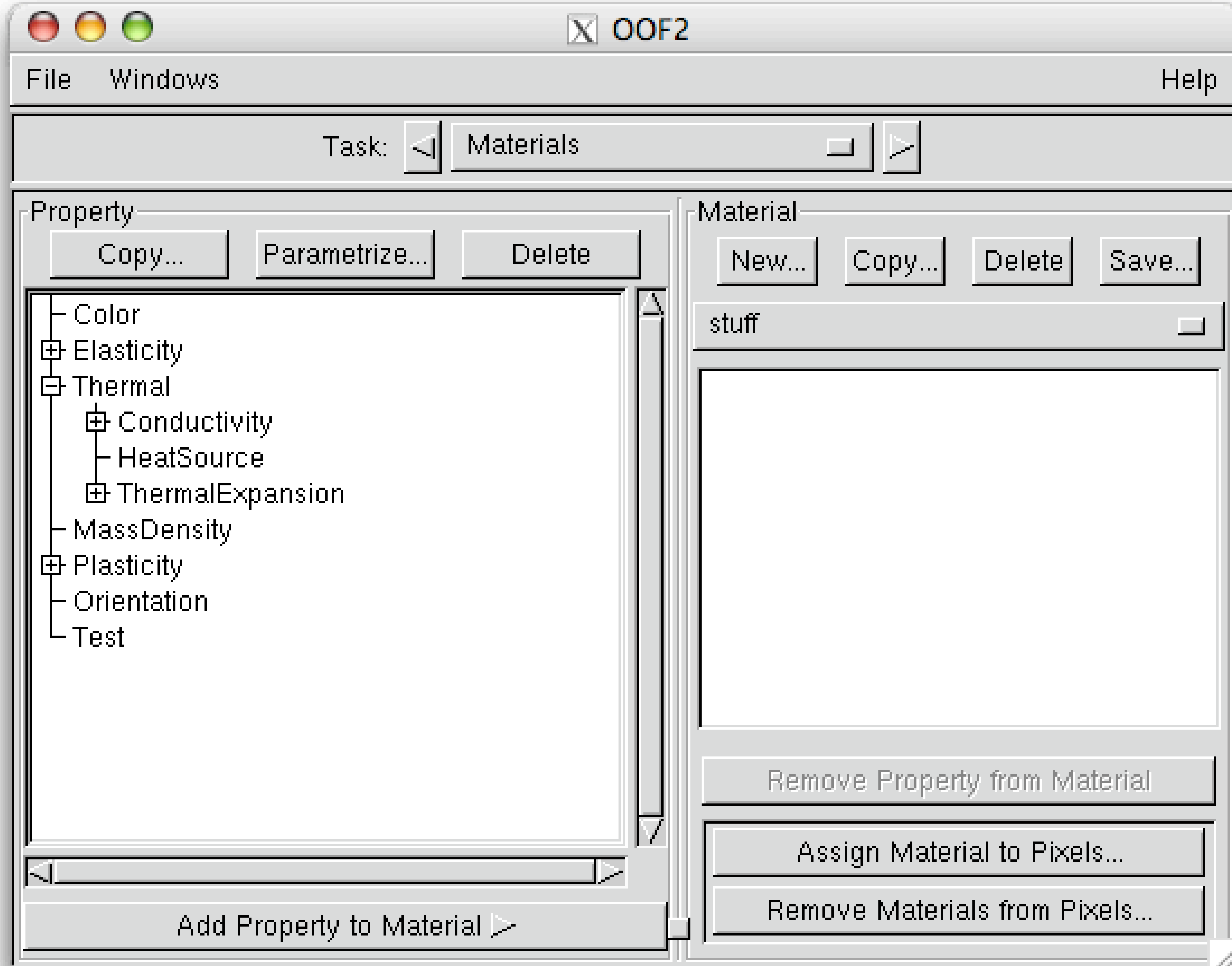
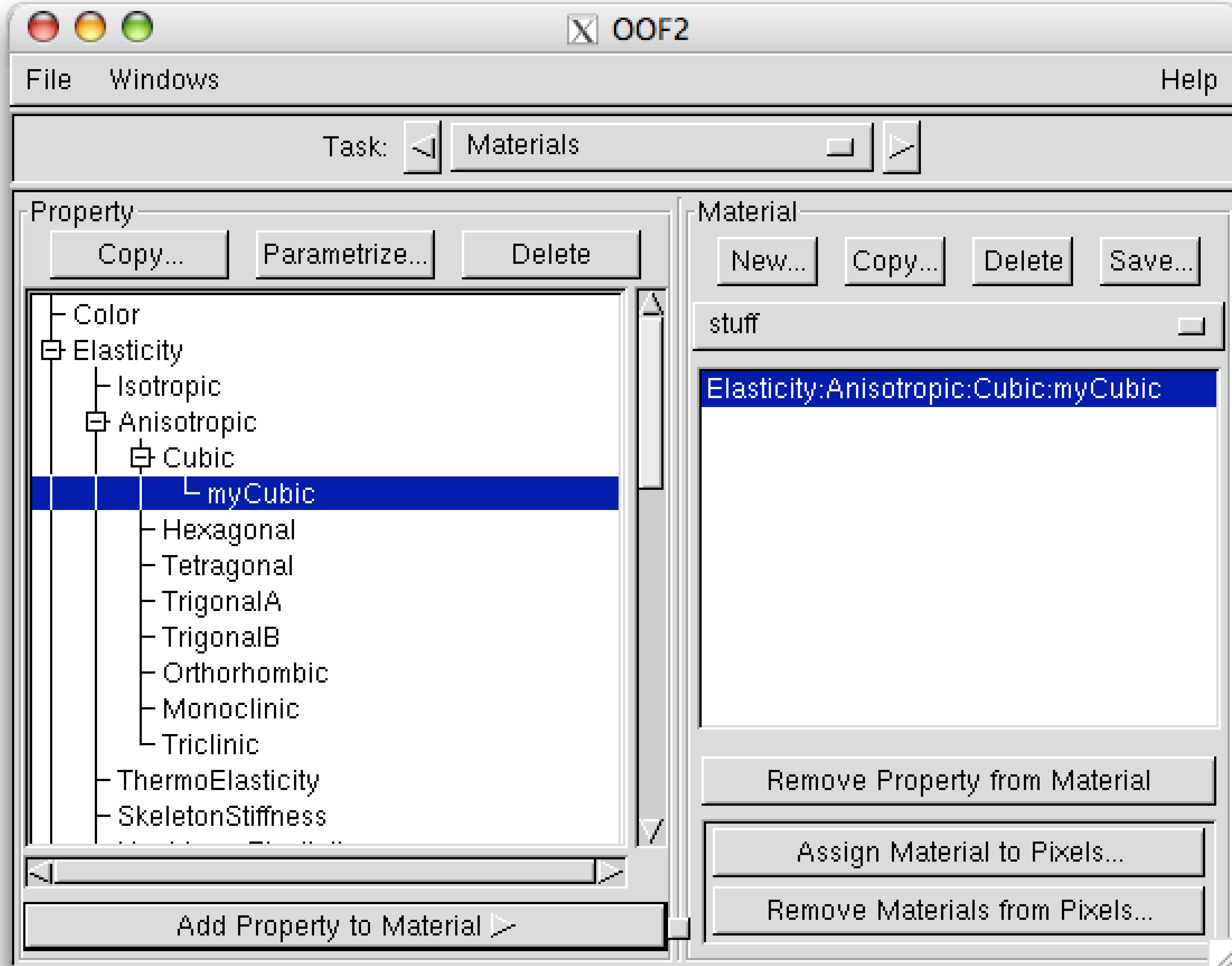


Image Modification tools

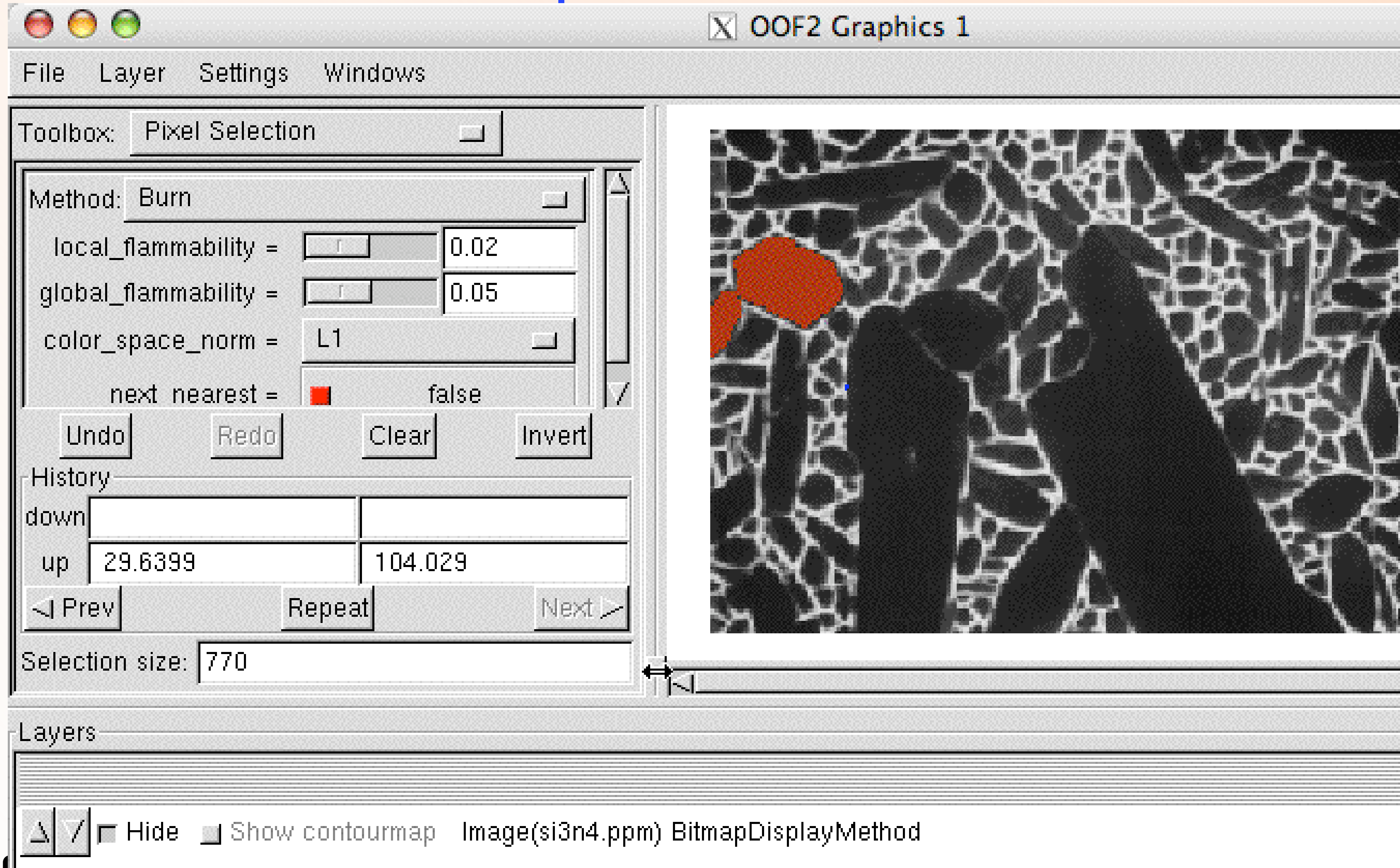


Material Construction GUI

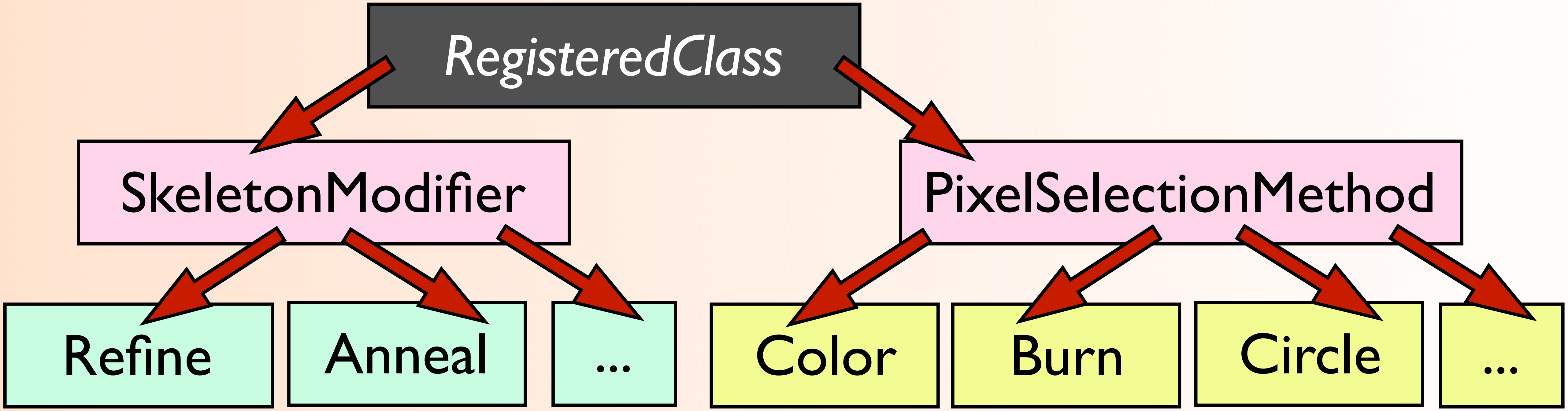


Material Construction GUI

Graphics Window

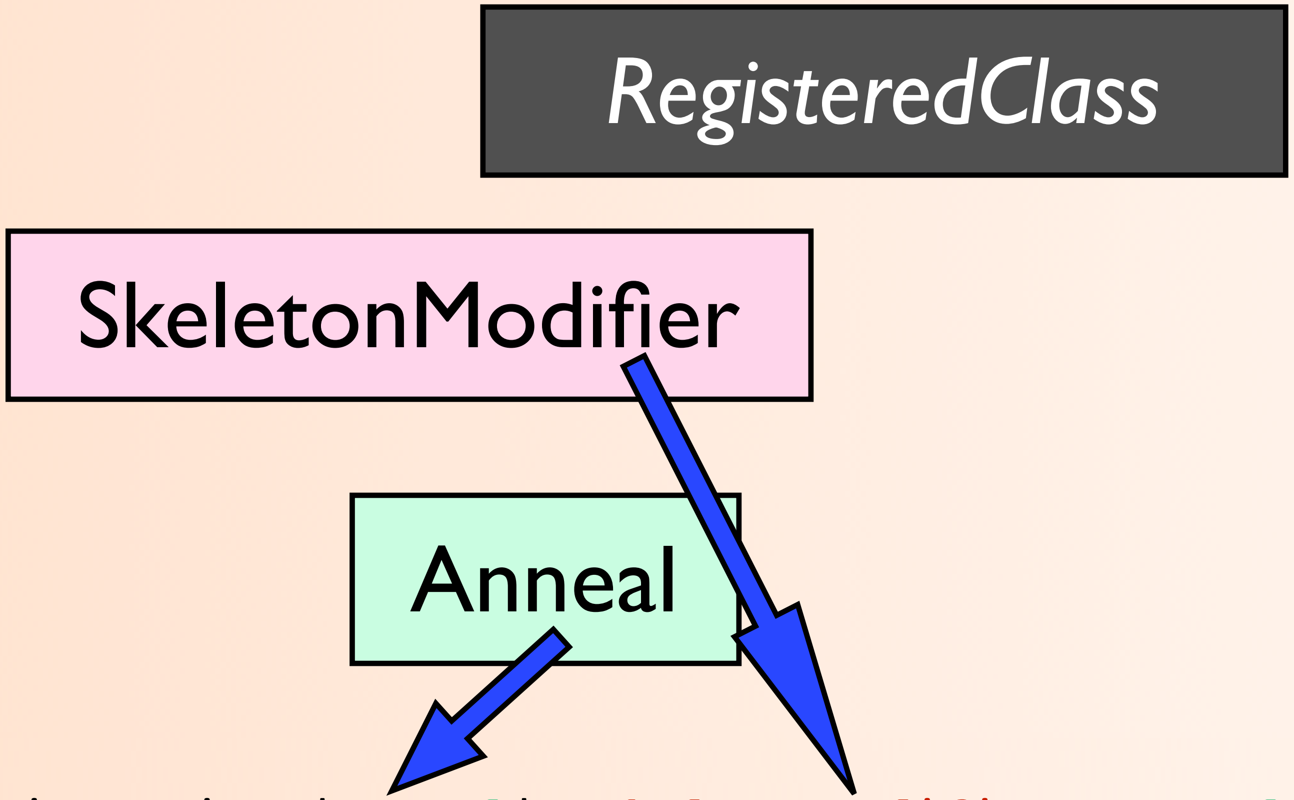


Extensibility via Class Hierarchy

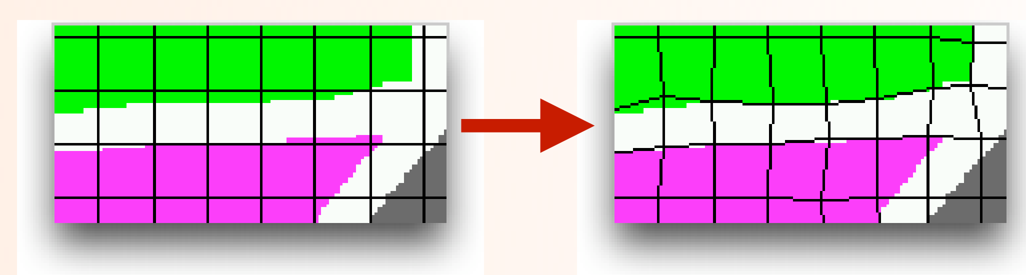


- Registered classes represent:
  - Operations on images, meshes, etc.
  - Material properties.
  - Parameters for the above.
- *Registrations* describe how to create objects in the classes
- Menus and GUI components are created *automatically* from Registrations.

Extensibility via Class Hierarchy




```
Registration('Anneal', SkeletonModifier, Anneal,
  params=[
    RegisteredParameter('targets', FiddleNodesTargets, tip='Which nodes to fiddle.'),
    RegisteredParameter('criterion', skeletonmodifier.SkelModCriterion, tip = 'Acceptance..')
    FloatParameter('T', value = 0.0, tip='Effective “temperature” of ...'),
    FloatParameter('delta', value=1.0, tip='Width of the distribution of ...'),
    RegisteredParameter('iteration', IterationManager, tip='Iteration method')
  ],
  tip='Move nodes randomly and accept the ones that meet the acceptance criterion.'
 )
```



Extensibility via Class Hierarchy

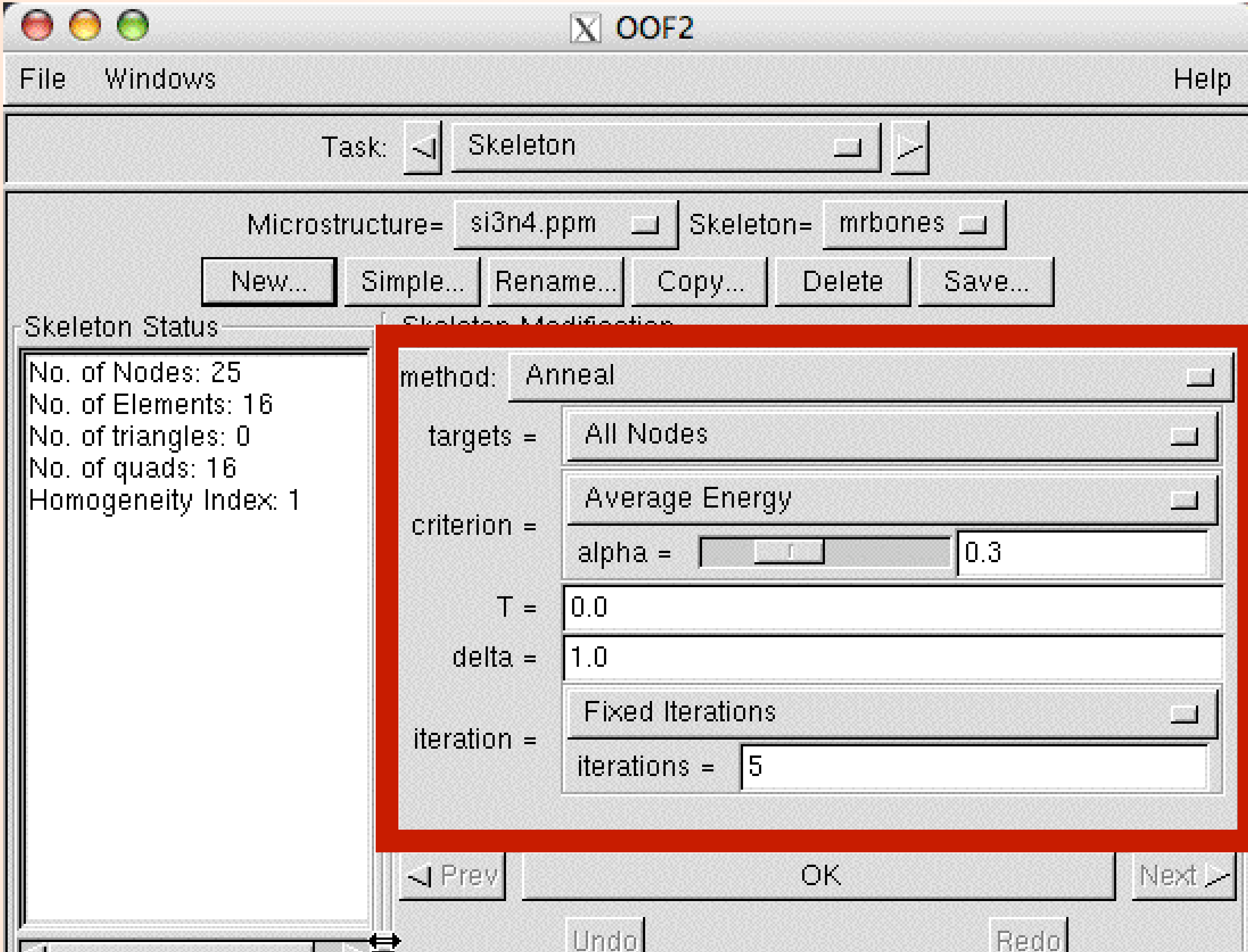
RegisteredClass

SkeletonModifier

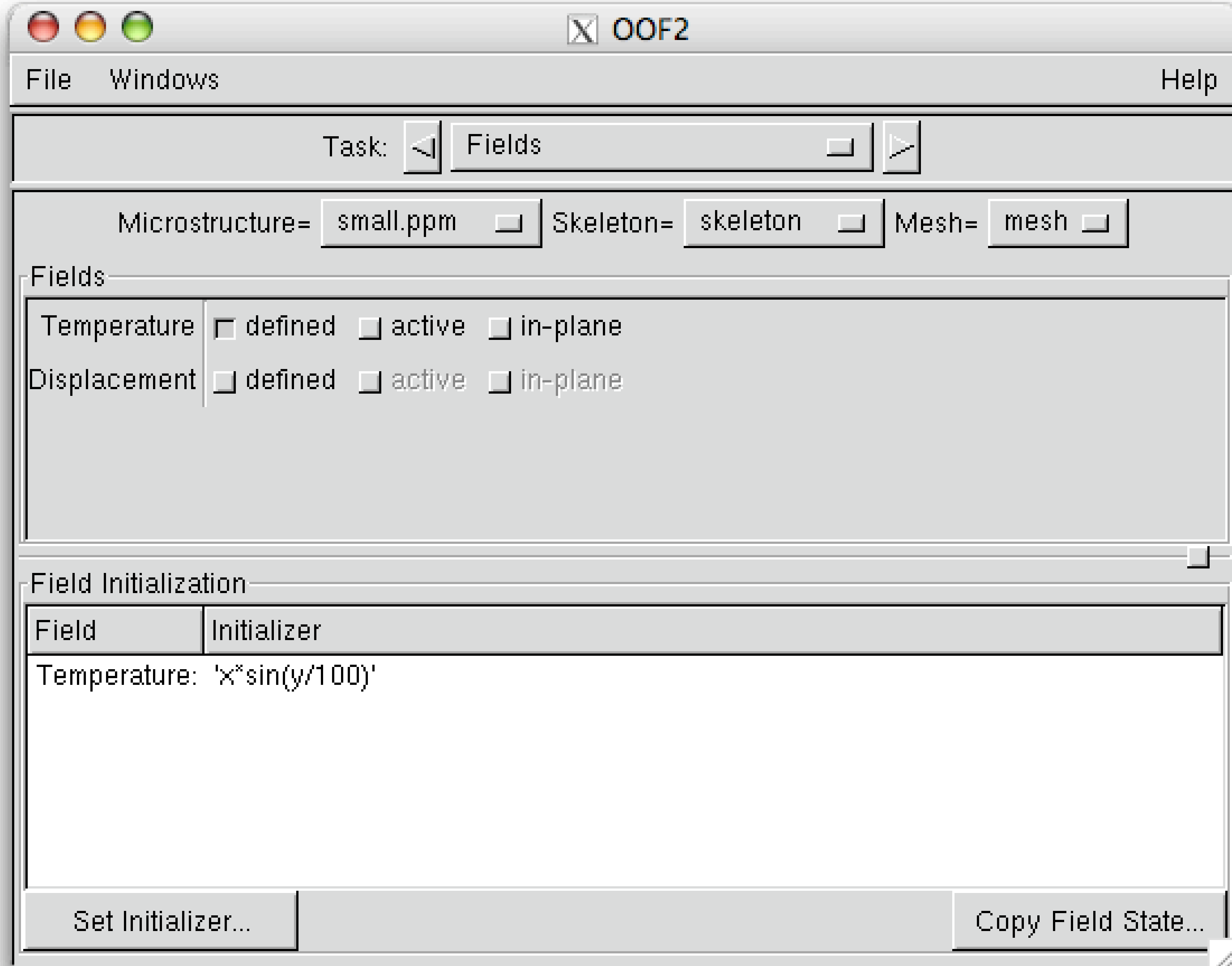
Anneal

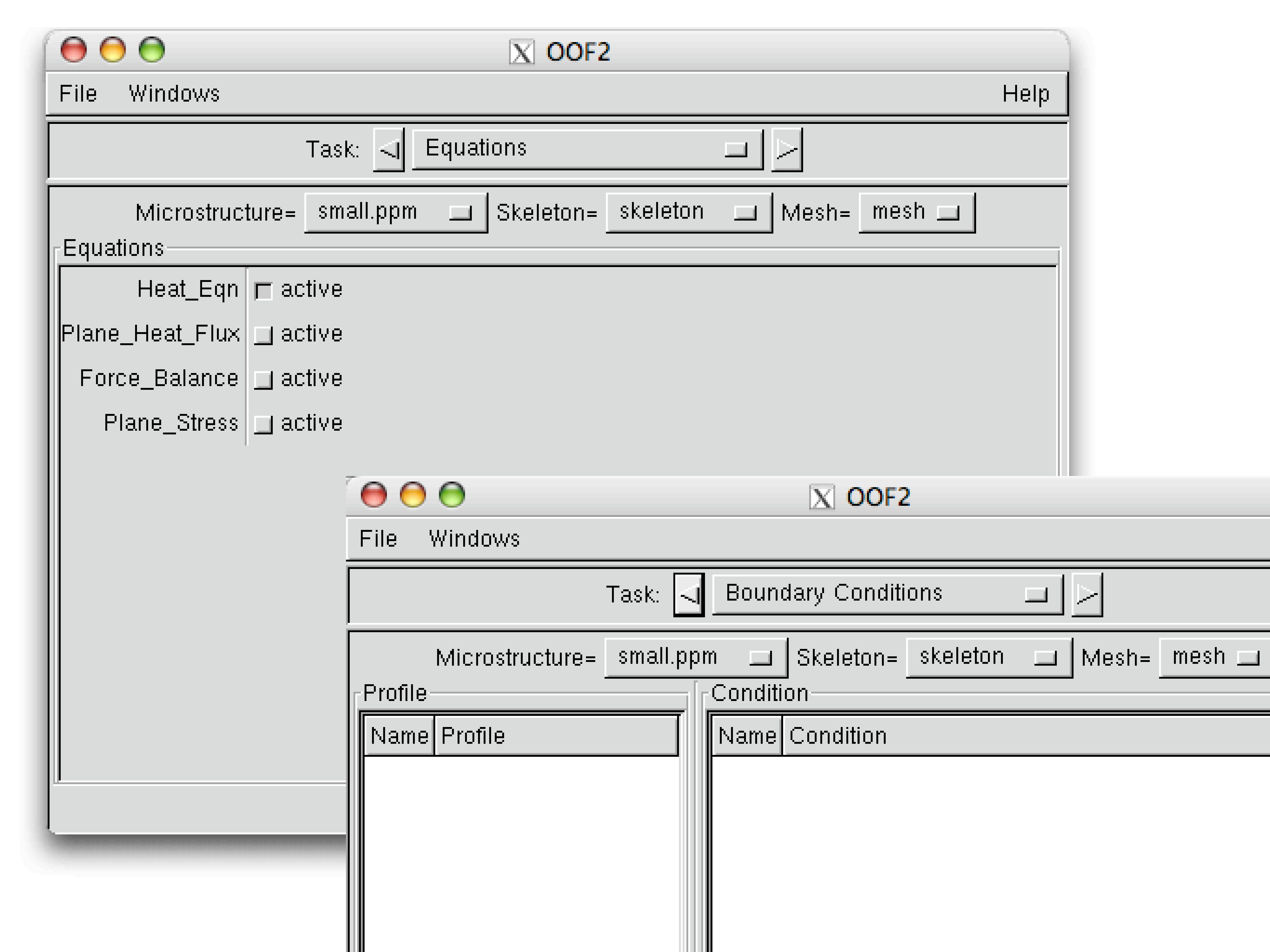
```
Registration('Anneal', SkeletonModifier, Anneal,
  params=[
    RegisteredParameter('targets', FiddleNodesTargets, tip='Which nodes to fiddle.'),
    RegisteredParameter('criterion', skeletonmodifier.SkelModCriterion, tip = 'Acceptance..'
    FloatParameter('T', value = 0.0, tip='Effective “temperature” of ...'),
    FloatParameter('delta', value=1.0, tip='Width of the distribution of ...'),
    RegisteredParameter('iteration', IterationManager, tip='Iteration method')
  ],
  tip='Move nodes randomly and accept the ones that meet the acceptance criterion.'
 )
```

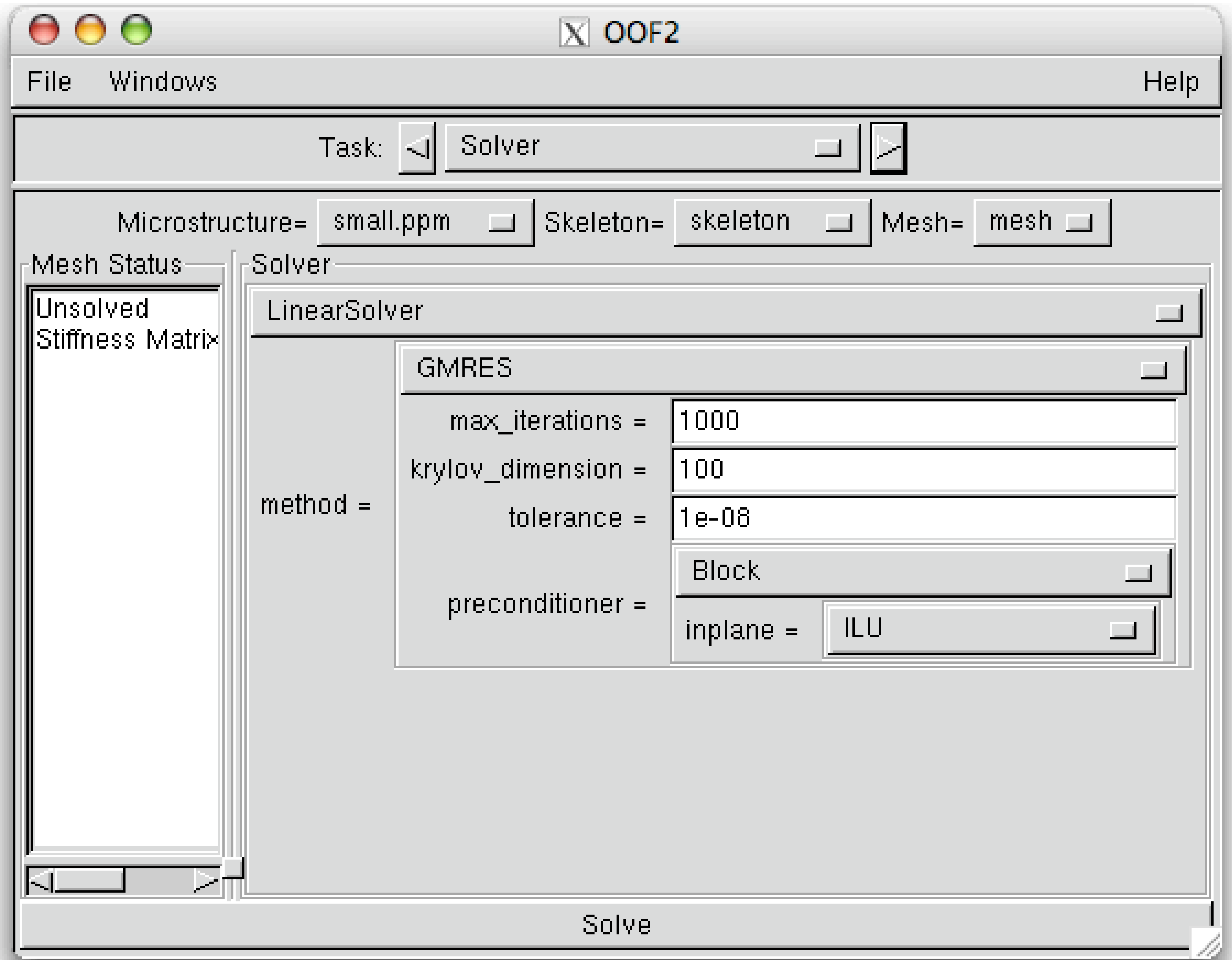
Parameters provide all information needed to construct an object.

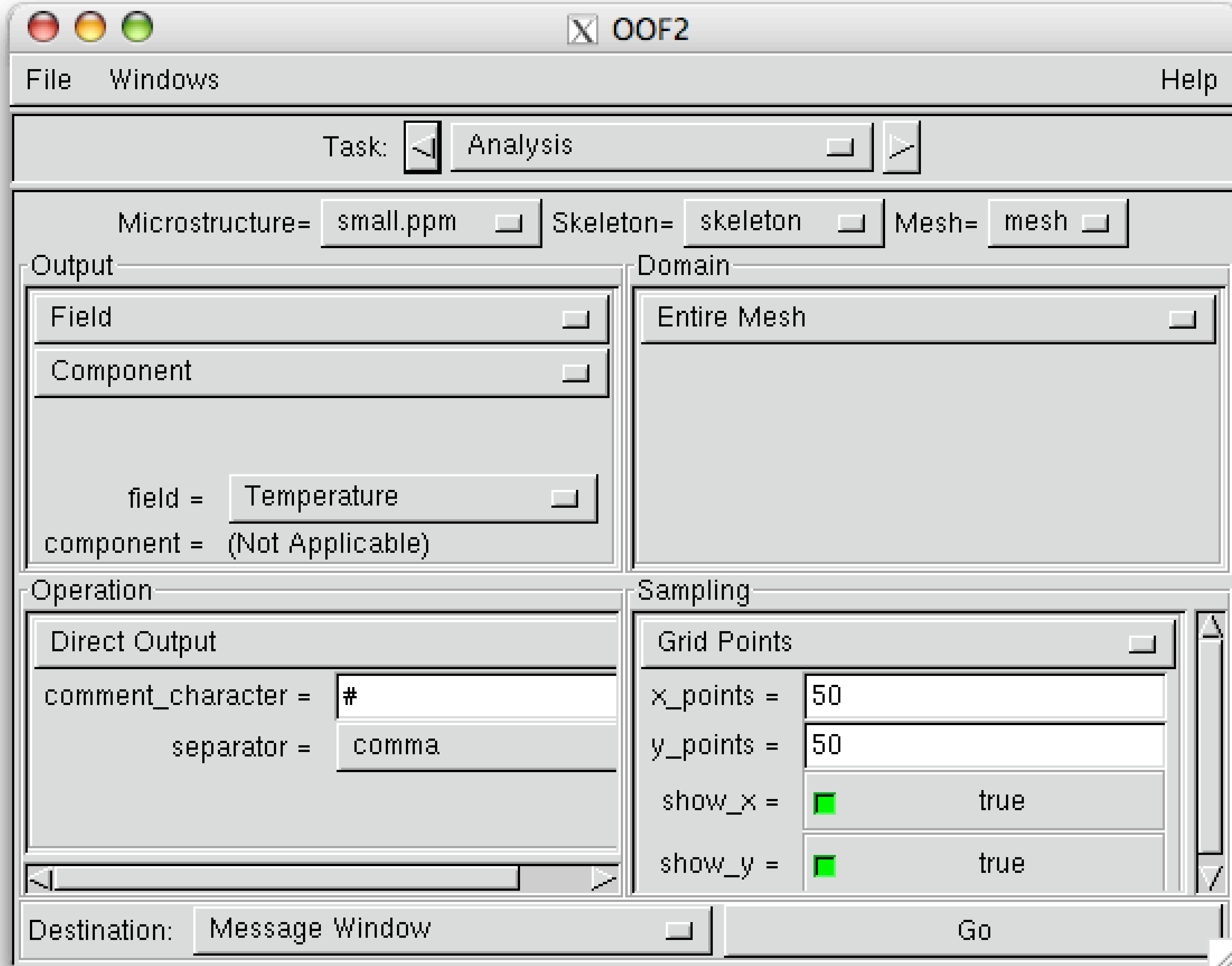


RegisteredClassFactory is built automatically in the GUI


OOF2
File
Windows
Help
Task:
Fields
Microstructure=
small.ppm
Skeleton=
skeleton
Mesh=
mesh
Fields
Temperature
defined
active
in-plane
Displacement
defined
active
in-plane
Field Initialization
Field
Initializer
Temperature: 'x*sin(y/100)'
Set Initializer...
Copy Field State...


OOF2
File
Windows
Help
Task:
Equations
Microstructure=
small.ppm
Skeleton=
skeleton
Mesh=
mesh
Equations
Heat_Eqn
active
Plane_Heat_Flux
active
Force_Balance
active
Plane_Stress
active
OOF2
File
Windows
Task:
Boundary Conditions
Microstructure=
small.ppm
Skeleton=
skeleton
Mesh=
mesh
Profile
Name
Profile
Condition
Name
Condition


OOF2
File
Windows
Help
Task:
Solver
Microstructure=
small.ppm
Skeleton=
skeleton
Mesh=
mesh
Mesh Status
Unsolved
Stiffness Matrix
Solver
LinearSolver
GMRES
method =
max_iterations =
1000
krylov_dimension =
100
tolerance =
1e-08
preconditioner =
Block
inplane =
ILU
Solve



OOF2
File
Windows
Help
Task:
Analysis
Microstructure=
small.ppm
Skeleton=
skeleton
Mesh=
mesh
Output
Field
Component
field =
Temperature
component =
(Not Applicable)
Domain
Entire Mesh
Operation
Direct Output
comment_character =
#
separator =
comma
Sampling
Grid Points
x_points =
50
y_points =
50
show_x =
true
show_y =
true
Destination:
Message Window
Go

Extending OOF2 with new Physics

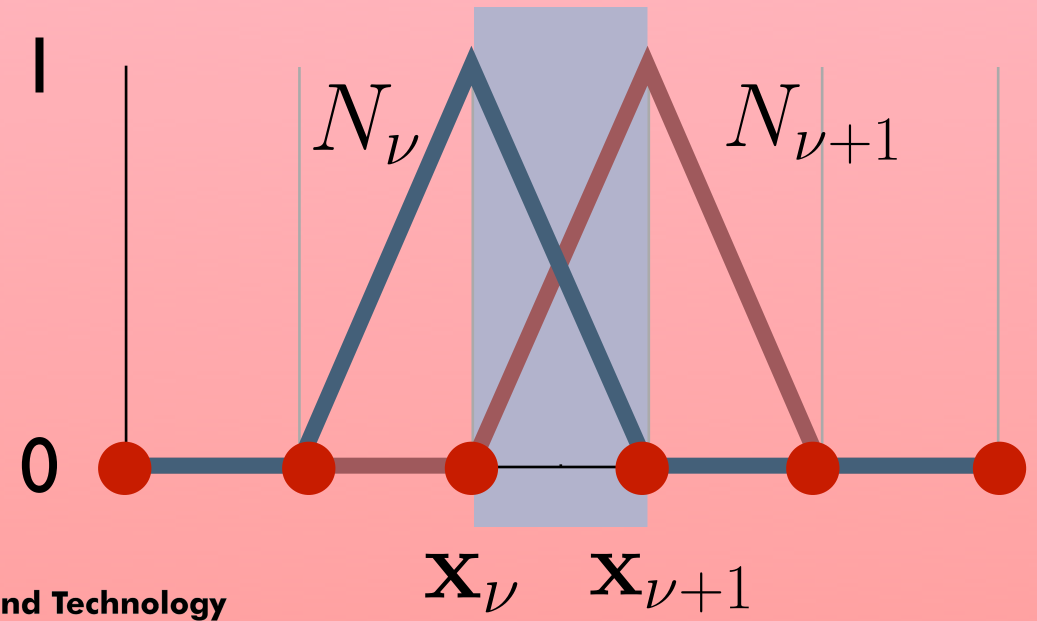
New Fields require just a few lines of Python:

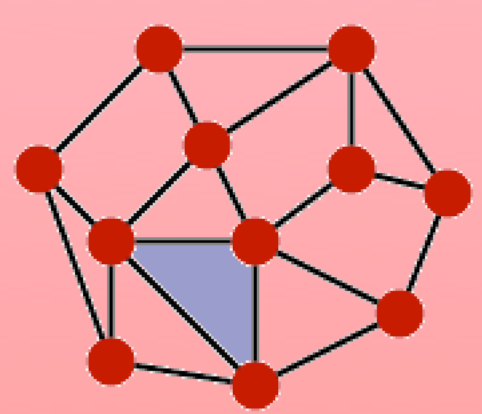
```
temperature = defineField(ScalarField("Temperature"))
heat_flux = defineFlux(VectorFlux("Heat_Flux"))
heat_eqn = defineEquation(DivergenceEquation("Heat", heat_flux, 1))
planeheatflux_eqn = defineEquation(PlaneFluxEquation("Plane_Heat_Flux",
                                                       heat_flux, 1)),

displacement = defineField(TwoVectorField("Displacement"))
stress_flux = defineFlux(SymmetricTensorFlux("Stress"))
forcebalance_eqn = defineEquation(DivergenceEquation("Force_Balance",
                                                      stress_flux, spacedim))
planestress_eqn = defineEquation(PlaneFluxEquation("Plane_Stress",
                                                    stress_flux, 3))
```

Actually using new Fields in material properties requires a bit more effort…

Finite Elements in 50 Words or Less*

- Divide space into *elements*.
- Evaluate fields at *nodes* between elements: $u_{n\nu} = u_n(\mathbf{x}_\nu)$
- Interpolate fields in elements via *shape functions* $N_\nu(\mathbf{x})$
- $$u_n(\mathbf{x}) = \sum_\nu u_{n\nu} N_\nu(\mathbf{x})$$
-
- Substitute expansion into equations, multiply by a test function, integrate by parts, and solve the resulting system of linear equations for the unknowns $u_{n\nu}$.





*Pedants will insist upon "fewer" instead of "less" here, but "less" is the colloquial usage.

Adding New Material Properties

- A "Property" is a term in a flux: $\sigma = \sum_i k_i \nabla \phi_i$ SCHEMATIC
- Define $\sigma = \mathbf{M} \cdot u$
  - u is the vector of all field values at all nodes of an element
  - $\mathbf{M}$ is the "flux matrix"
- Developer must provide a routine to compute an element's contribution to $\mathbf{M}$ at $\mathbf{x}$ for node ν.
  - This can be done with no explicit knowledge of the element geometry.

Example: Elasticity

- Displacement component *l* at point **x**: $u_l(\mathrm{x})$
- Stress component *ij* at **x**: $\sigma_{ij}(\mathrm{x}) = C_{ijkl}\partial_k u_l(\mathrm{x})$
- Expand in shape functions: $u_l(\mathrm{x}) = N_\nu(\mathrm{x})u_{l\nu}$
  - $u_{l\nu}$ is displacement component *l* at node ν.
  - $\sigma_{ij}(\mathrm{x}) = C_{ijkl}\partial_k N_\nu(\mathrm{x})u_{l\nu}$
- Compare to $\sigma_{ij}(\mathrm{x}) = M_{ij}^{k\nu}u_{k\nu}$
- Find $M_{ij}^{k\nu}(\mathrm{x}) = C_{ijkl}\partial_l N_\nu(\mathrm{x})$

Example: Elasticity

```
void Elasticity::fluxmatrix(const FEMesh *mesh, const Element *element,
                            const ElementFuncNodeIterator &nu,
                            Flux *flux, const MasterPosition &x) const
{
  if(*flux != *stress_flux) {
    throw ErrProgrammingError("Unexpected flux", __FILE__, __LINE__);
  }

  const Cijkl modulus = cijkl(mesh, element, x);
  double sf = nu.shapefunction(x);
  double dsf0 = nu.dshapefunction(0, x);
  double dsf1 = nu.dshapefunction(1, x);

  for(SymTensorIndex ij; !ij.end(); ++ij) {
    for(FieldIterator ell=displacement->iterator(); !ell.end(); ++ell) {
      SymTensorIndex ell0(0, ell.integer());
      SymTensorIndex ell1(1, ell.integer());
      stress_flux->matrix_element(mesh, ij, displacement, ell, nu) +=
                               modulus(ij, ell0)*dsf0 + modulus(ij, ell1)*dsf1;
    }
 if(!displacement->in_plane(mesh)) {
      Field *oop = displacement->out_of_plane();
      for(FieldIterator ell=oop->iterator(ALL_INDICES); !ell.end(); ++ell) {
          stress_flux->matrix_element(mesh, ij, oop, ell, nu)
                      += modulus(ij, SymTensorIndex(2,ell.integer())) * sf;
      }
    }
  }
}
```


Disclaimer: slightly simplified to fit on the screen...

Example: Elastic

Node ν

```
void Elasticity::fluxmatrix(const FEMesh *mesh, const Element *element,
                            const ElementFuncNodeIterator &nu,
                            Flux *flux, const MasterPosition &x) const
{
  if(*flux != *stress_flux) {
    throw Err                r("Unexpected flux", __FILE__,
  }

  const Cijkl modulus = cijkl(mesh, element, x);
  double sf = nu.shapefunction(x);
  double dsf0 = nu.dshapefunction(0, x);
  double dsf1 = nu.dshapefunction(1, x);

  for(SymTensorIndex ij; !ij.end(); ++ij) {
    for(FieldIterator ell=displacement->iterator(); !ell.end(); ++ell) {
      SymTensorIndex ell0(0, ell.integer());
      SymTensorIndex ell1(1, ell.integer());
      stress_flux->matrix_element(mesh, ij, displacement, ell, nu) +=
                        modulus(ij, ell0)*dsf0 + modulus(ij, ell1)*dsf1;
    }
   if(!displacement->in_plane(mesh)) {
      Field *oop = displacement->out_of_plane();
      for(FieldIterator ell=oop->iterator(ALL_INDICES); !ell.end(); ++ell) {
          stress_flux->matrix_element(mesh, ij, oop, ell, nu)
                += modulus(ij, SymTensorIndex(2,ell.integer())) * sf;
      }
    }
  }
}
```

Flux σ

Position **x**

Disclaimer: slightly simplified to fit on the screen...

Example: Elasticity

```
void Elasticity::fluxmatrix(const FEMesh *mesh, const Element *element,
                            const ElementFuncNodeIterator &nu,
                            Flux *flux, const MasterPosition &x) const
{
```

```
if(*flux != *stress_flux) {
  throw ErrProgrammingError("Unexpected flux", __FILE__, __LINE__);
}
```

```
  const Cijkl modulus = cijkl(mesh, element, x);
  double sf = nu.shapefun
  double dsf0 = nu.dshape
  double dsf1 = nu.dshapefunction(1, x);

  for(SymTensorIndex ij; !ij.end(); ++ij) {
    for(FieldIterator ell=displacement->iterator(); !ell.end(); ++ell) {
      SymTensorIndex ell0(0, ell.integer());
      SymTensorIndex ell1(1, ell.integer());
      stress_flux->matrix_element(mesh, ij, displacement, ell, nu) +=
                        modulus(ij, ell0)*dsf0 + modulus(ij, ell1)*dsf1;
    }
   if(!displacement->in_plane(mesh)) {
      Field *oop = displacement->out_of_plane();
      for(FieldIterator ell=oop->iterator(ALL_INDICES); !ell.end(); ++ell) {
          stress_flux->matrix_element(mesh, ij, oop, ell, nu)
                  += modulus(ij, SymTensorIndex(2,ell.integer())) * sf;
      }
    }
  }
}
```

Sanity Check

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

Disclaimer: slightly simplified to fit on the screen...

Example: Elasticity

```
void Elasticity::fluxmatrix(const FEMesh *mesh, const Element *element,
                            const ElementFuncNodeIterator &nu,
                            Flux *flux, const MasterPosition &x) const
{
  if(*flux != *stress_flux) {
    throw ErrProgrammingError("Unexpected flux", __FILE__, __LINE__);
  }

  const Cijkl modulus = cijkl(mesh, element, x);
  double sf = nu.shapefunction(x);
  double dsf0 = nu.dshapefunction(0, x);
  double dsf1 = nu

  for(SymTensorInd
    for(FieldItera                                              l) {
      SymTensorInd
      SymTensorInd
      stress_flux->matrix_element(mesh, ij, displacement, ell, nu) +=
                             modulus(ij, ell0)*dsf0 + modulus(ij, ell1)*dsf1;
    }
 if(!displacement->in_plane(mesh)) {
     Field *oop = displacement->out_of_plane();
     for(FieldIterator ell=oop->iterator(ALL_INDICES); !ell.end(); ++ell) {
         stress_flux->matrix_element(mesh, ij, oop, ell, nu)
                 += modulus(ij, SymTensorIndex(2,ell.integer())) * sf;
     }
   }
  }
}
```

Elastic modulus computed by virtual function call to derived class (eg. CubicElasticity)

Disclaimer: slightly simplified to fit on the screen...

Example: Elasticity

```
void Elasticity::fluxmatrix(const FEMesh *mesh, const Element *element,
                            const ElementFuncNodeIterator &nu,
                            Flux *flux, const MasterPosition &x) const
{
  if(*flux != *stress_flux) {
    throw ErrProgrammingError("Unexpected flux", __FILE__, __LINE__);
  }

  const Cijkl modulus = cijkl(mesh, element, x);
  double sf = nu.shapefunction(x);
  double dsf0 = nu.dshapefunction(0, x);
  double dsf1 = nu.dshapefunction(1, x);

  for(SymTensorIndex ij; !ij.end(); ++ij) {
    for(FieldIterator ell=d                                   l) {
      SymTensorIndex ell0(0
      SymTensorIndex ell1(1
      stress_flux->matrix_e                                  =
                                                             ll1)*dsf1;
    }
 if(!displacement->in_plane(mesh)) {
      Field *oop = displacement->out_of_plane();
      for(FieldIterator ell=oop->iterator(ALL_INDICES); !ell.end(); ++ell) {
          stress_flux->matrix_element(mesh, ij, oop, ell, nu)
                        += modulus(ij, SymTensorIndex(2,ell.integer())) * sf;
      }
    }
  }
}
```

Shape function evaluation
for node ν at point $\mathbf{x}$

Example: Elasticity

```
void Elasticity::fluxmatrix(const FEMesh *mesh, const Element *element,
                            const ElementFuncNodeIterator &nu,
                            Flux *flux, const MasterPosition &x) const
{
  if(*flux != *stress_flux) {
    throw ErrProgrammingError("Unexpected flux", __FILE__, __LINE__);
  }

  cons
  double sf = nu.shapefunction(x);
  double dsf0 = nu.dshapefunction(0, x
  double dsf1 = nu.dshapefunction(1, x

  for(SymTensorIndex ij; !ij.end(); ++ij) {
    for(FieldIterator ell=displacement->iterator(); !ell.end(); ++ell) {
      SymTensorIndex ell0(0, ell.integer());
      SymTensorIndex ell1(1, ell.integer());
      stress_flux->matrix_element(mesh, ij, displacement, ell, nu) +=
                        modulus(ij, ell0)*dsf0 + modulus(ij, ell1)*dsf1;
    }
   if(!displacement->in_pla
      Field *oop = displa
      for(FieldIterator e                              nd(); ++ell) {
          stress_flux->ma
                      +=                               ger())) * sf;
      }
    }
  }
}
```

For all stress components ij

For all displacement components l

$$M_{ij}^{l\nu}(\mathrm{x}) = \sum_k C_{ijkl}\partial_k N_\nu(\mathrm{x})$$

$l\nu \Rightarrow$ degree of freedom

$ij \Rightarrow$ stress component

Disclaimer: slightly simplified to fit on the screen...

Example: Elasticity

```
void Elasticity::fluxmatrix(const FEMesh *mesh, const Element *element,
                            const ElementFuncNodeIterator &nu,
                            Flux *flux, const MasterPosition &x) const
{
  if(*flux != *stress_flux) {
    throw ErrProgrammingError("Unexpected flux", __FILE__, __LINE__);
  }

  const Cijkl modulus = cijkl(mesh, element, x);
  double sf = nu.shapefunction(x);
  double dsf0 = nu.dshapefunction(0, x);
  double dsf1 = nu.dshapefunction(1, x);

  for(SymTensorIndex ij; !ij.end(); ++ij) {
    for(FieldIterator ell=displacement->iterator(); !ell.end(); ++ell) {
      SymTensorIndex ell0(0, ell
      SymTensorIndex ell1(1, ell
      stress_flux->matrix_elemen
                               modulus(ij, ell0)*dsf0 + modulus(ij, ell1)*dsf1;
    }
```

Contribution from out-of-plane strains

```
if(!displacement->in_plane(mesh)) {
     Field *oop = displacement->out_of_plane();
     for(FieldIterator ell=oop->iterator(ALL_INDICES); !ell.end(); ++ell) {
         stress_flux->matrix_element(mesh, ij, oop, ell, nu)
                     += modulus(ij, SymTensorIndex(2,ell.integer())) * sf;
     }
   }
 }
```

Example: Elasticity

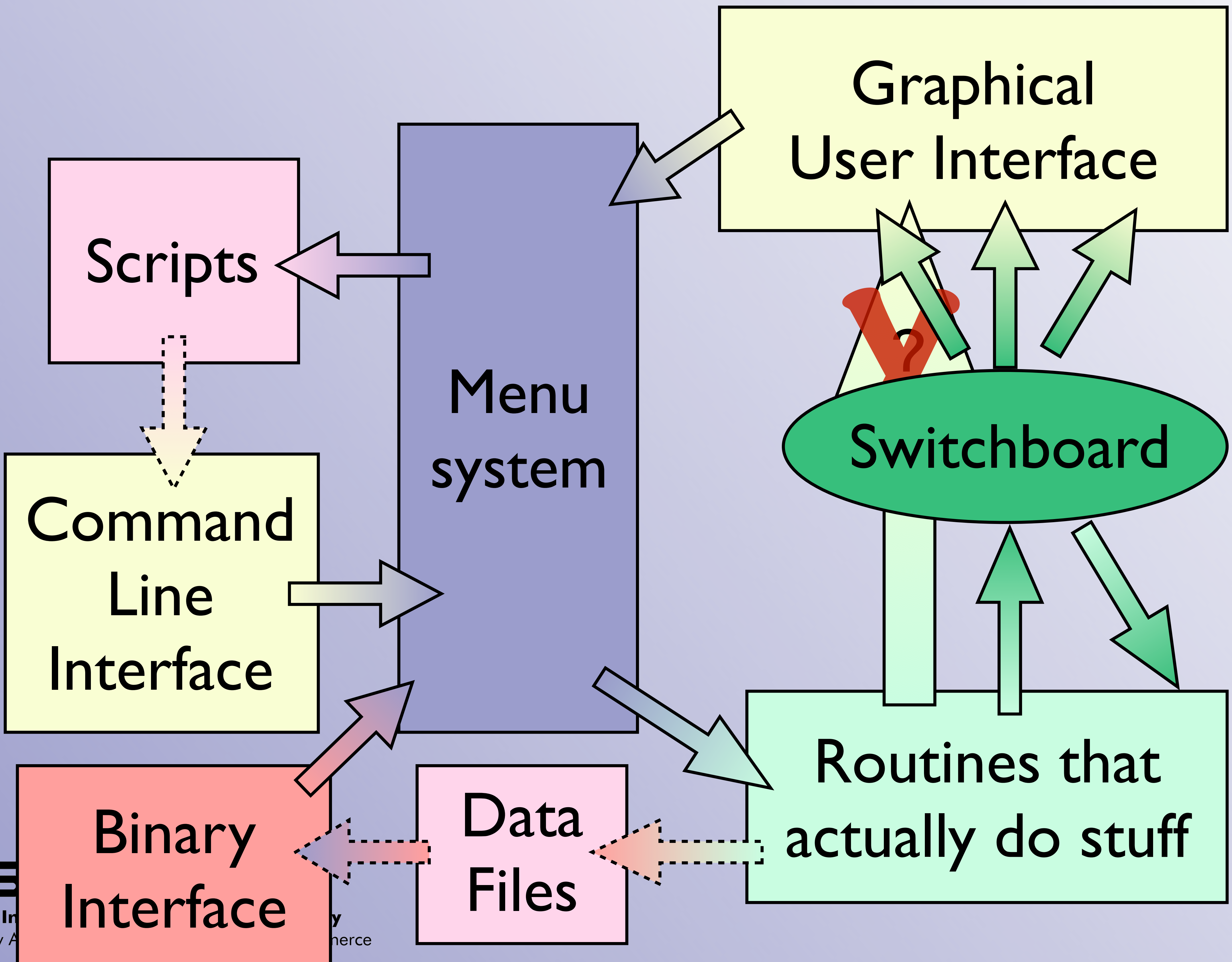
```
void Elasticity::fluxmatrix(const FEMesh *mesh, const Element *element,
                            const ElementFuncNodeIterator &nu,
                            Flux *flux, const MasterPosition &x) const
{
  if(*flux != *stress_flux) {
    throw ErrProgrammingError("Unexpected flux", __FILE__, __LINE__);
  }

  const Cijkl modulus = cijkl(mesh, el
  double sf = nu.shapefunction(x);
  double dsf0 = nu.dshapefunction(0, x
  double dsf1 = nu.dshapefunction(1, x

  for(SymTensorIndex ij; !ij.end(); ++
    for(FieldIterator ell=displacement
      SymTensorIndex ell0(0, ell.intege
      SymTensorIndex ell1(1, ell.intege
      stress_flux->matrix_element(mesh
                             modulus(
    }
 if(!displacement->in_plane(mesh)) {
      Field *oop = displacement->out_of_plane();
      for(FieldIterator ell=oop->iterator(ALL_INDICES); !ell.end(); ++ell) {
          stress_flux->matrix_element(mesh, ij, oop, ell, nu)
                      += modulus(ij, SymTensorIndex(2,ell.integer())) * sf;
      }
    }
  }
}
```

- ***No*** explicit dependence on:
  - Element geometry
    - triangle, quadrilateral
  - Element order
    - linear, quadratic...
  - Equation
    - divergence, plane-stress
  - Other material properties

More Infrastructure

- Underlying menu driven structure (in Python):
  - Specify name, callback function, menu, argument parameters.
  - Menu items created explicitly, or implicitly from Registrations.
- Communication between different code components is by means of a "switchboard"
  - Objects send messages to switchboard.
  - Other objects subscribe to messages.
  - Sending object doesn't have to know who (if anybody) is listening.
  - Allows modular development and use.

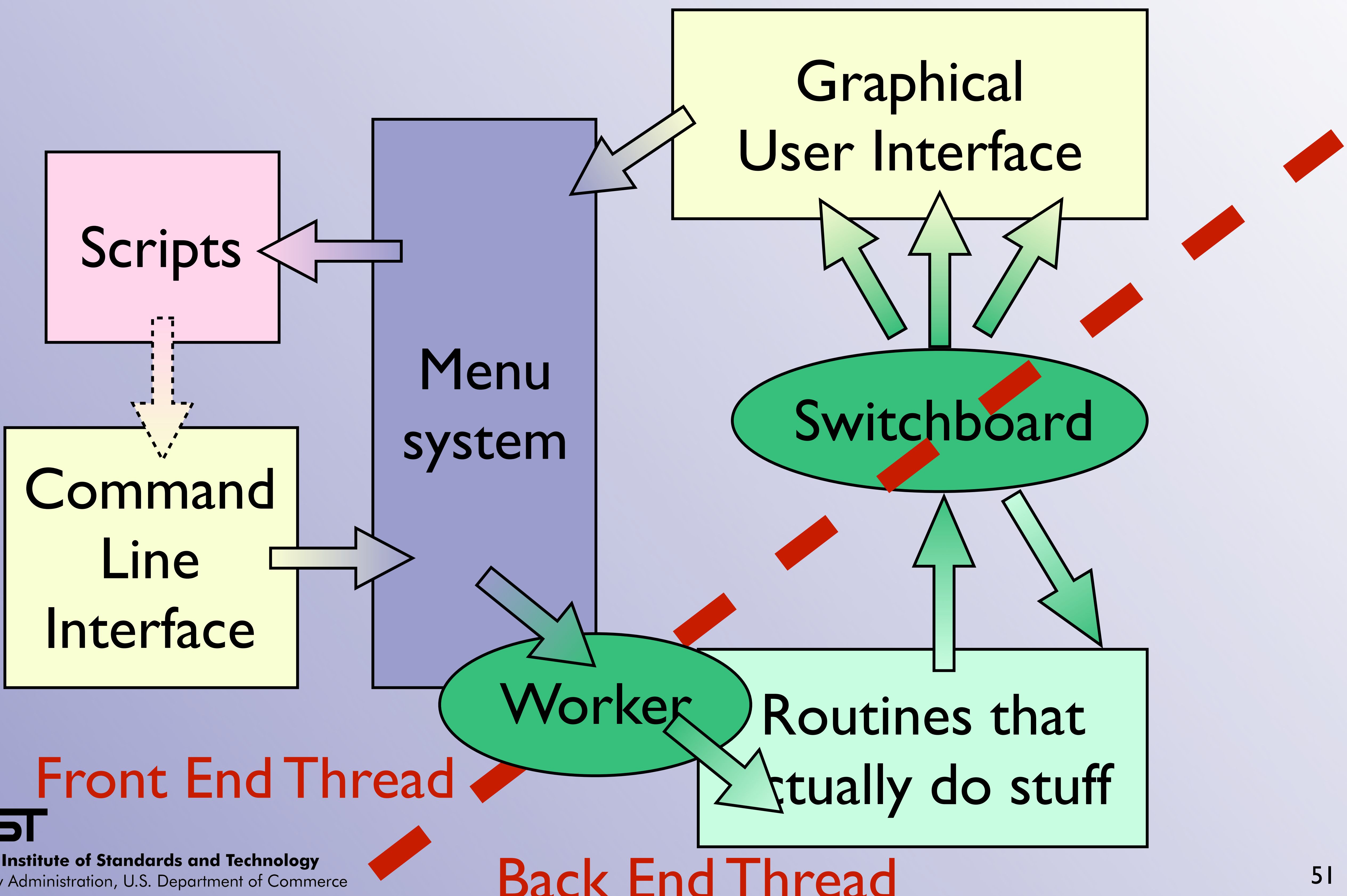
OOF2 Control Structure



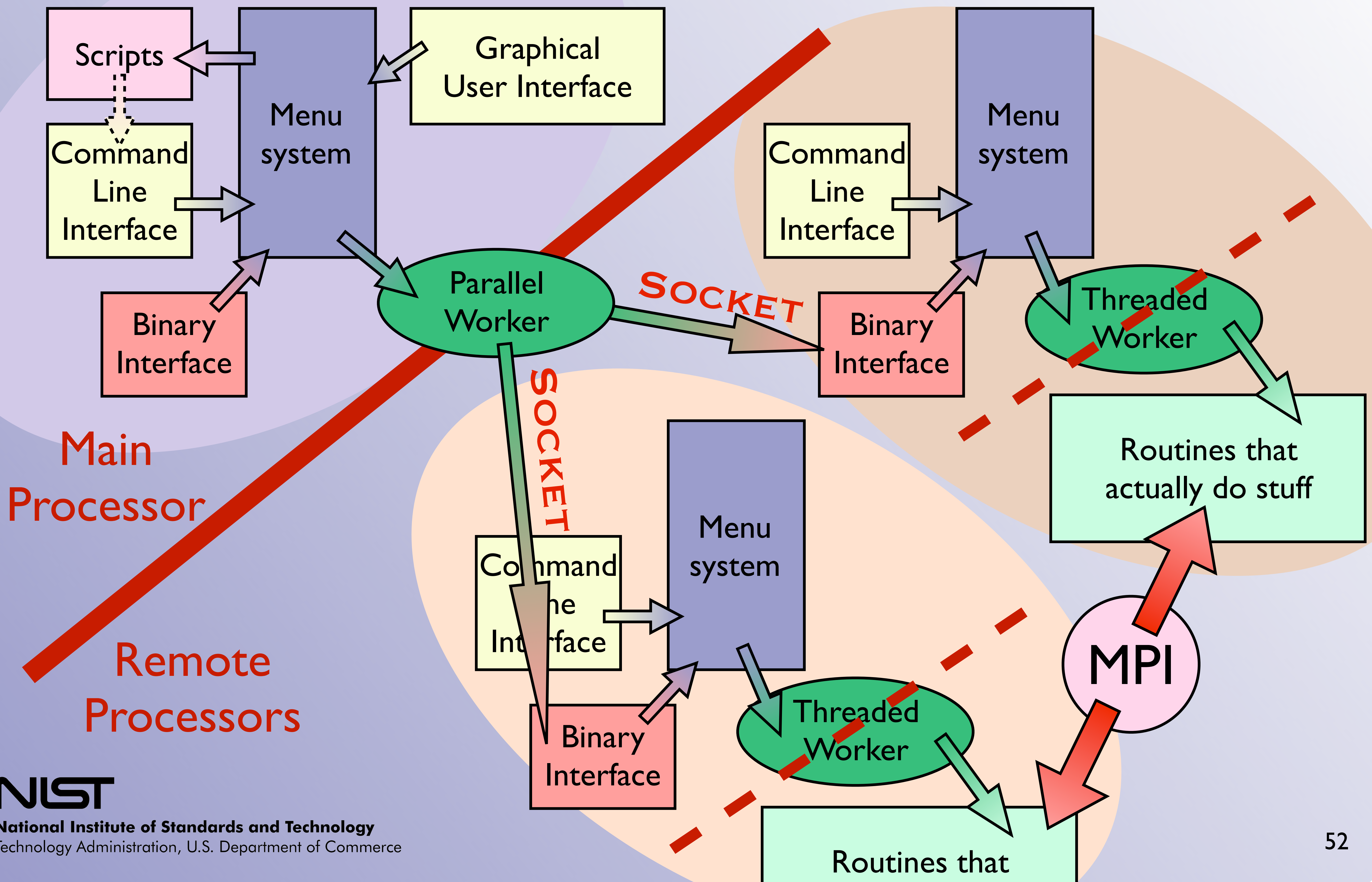
GUI, Threading & Parallel Processing

- OOF is meant to be an interactive system in which users can experiment with different scenarios in real time.
  - Need a responsive multithreaded interface.
  - Parallel back-end for quick turnaround.
- Still, lengthy computations need to be performed in batch mode, without a GUI.
- "Worker" classes added to menu system to handle different modes of operation.
  - TextWorker, GUIWorker, ThreadedWorker, etc.

OOF2 Control Structure



OOF2 Control Structure



http://www.ctcms.nist.gov/oof/

- OOF1
  - source code
  - precompiled binaries
  - manuals & tutorials
- OOF2
  - source code with built-in tutorials
  - precompiled binaries (soon)
  - manuals (soon)
- Mailing list