# Finite Element Code Generation: Simplicity, Generality, Efficiency

Anders Logg

Simula Research Laboratory

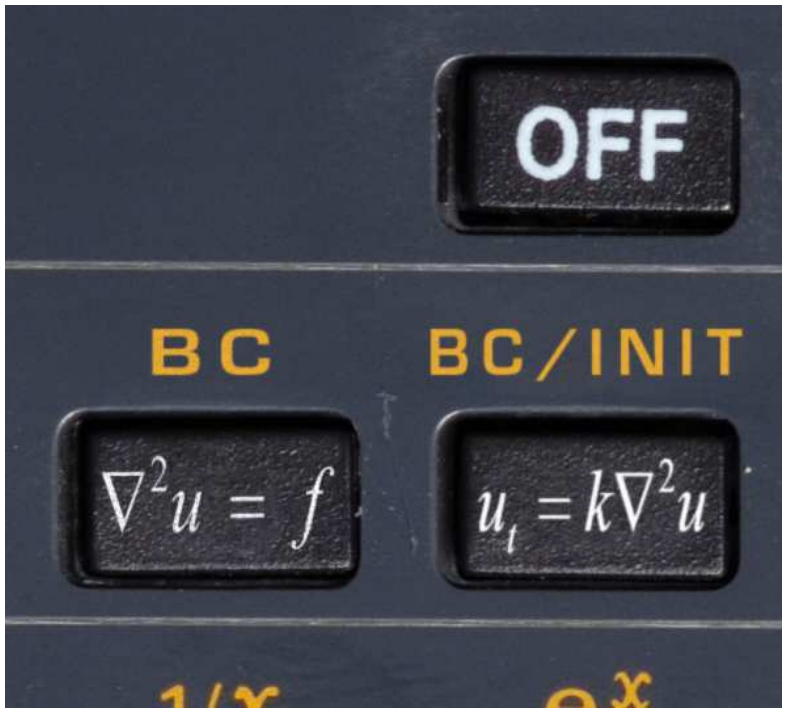
SCSE'07 Uppsala, August 15 2007

*Acknowledgments*: Martin Sandve Alnæs, Johan Hoffman, Johan Jansson, Claes Johnson, Robert C. Kirby, Matthew G. Knepley, Hans Petter Langtangen, Kent-Andre Mardal, Kristian Oelgaard, Marie Rognes, L. Ridgway Scott, Andy R. Terrel, Garth N. Wells, Åsmund Ødegård, Magnus Vikstrøm

# Solving differential equations

- Build a calculator
- One button for each equation?

- Too many buttons!



# Automate the solution of differential equations

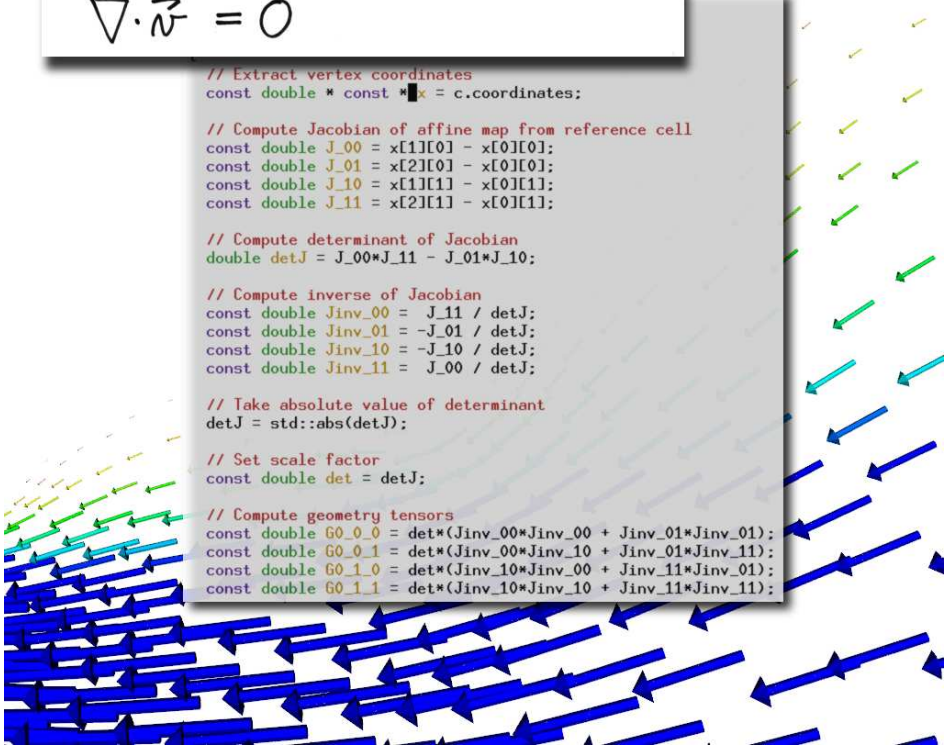
- Input: Differential equation
- Output: Solution

- Generate computer code
- Compute solution

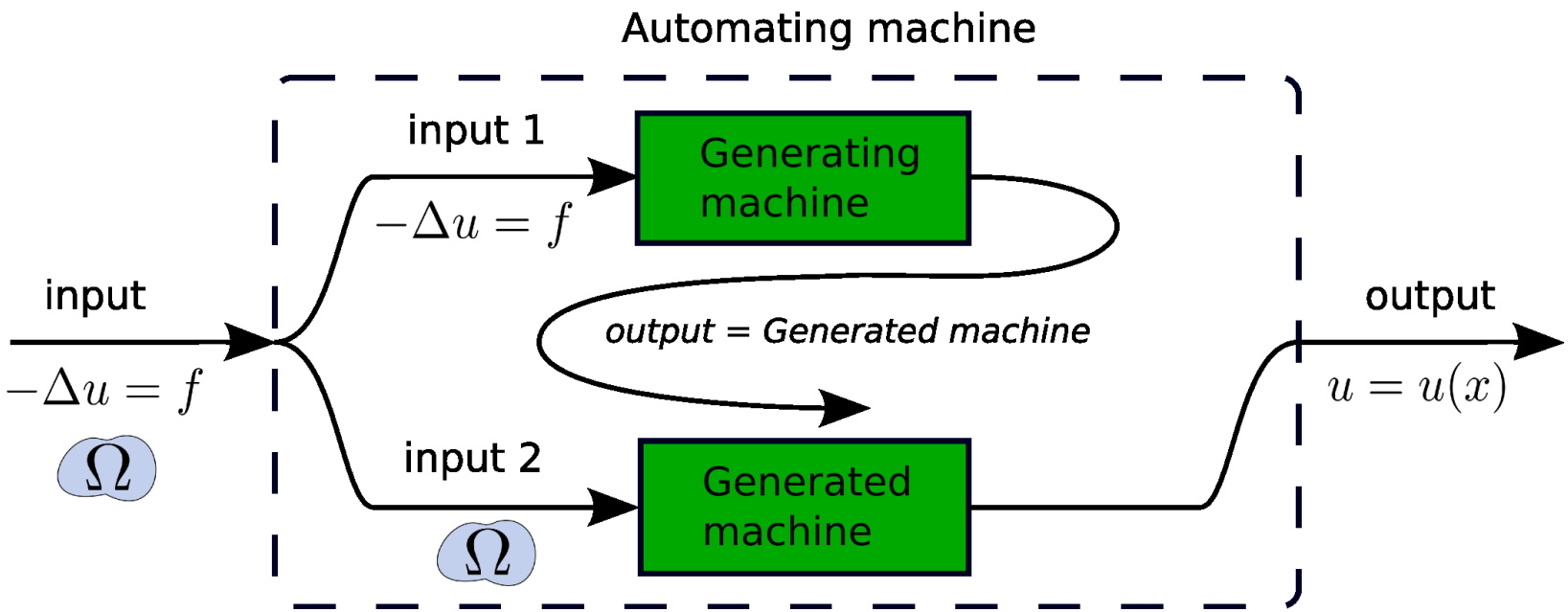
- Build a calculator for each equation!
- Automate the generation of calculators!

$$\rho\left(\frac{\partial \vec{u}}{\partial t} + \vec{u}\cdot\nabla\vec{u}\right) = -\nabla p + \mu\nabla^2\vec{u} + \rho\vec{b}$$
$$\nabla\cdot\vec{u} = 0$$

```
// Extract vertex coordinates
const double * const * x = c.coordinates;

// Compute Jacobian of affine map from reference cell
const double J_00 = x[1][0] - x[0][0];
const double J_01 = x[2][0] - x[0][0];
const double J_10 = x[1][1] - x[0][1];
const double J_11 = x[2][1] - x[0][1];

// Compute determinant of Jacobian
double detJ = J_00*J_11 - J_01*J_10;

// Compute inverse of Jacobian
const double Jinv_00 =  J_11 / detJ;
const double Jinv_01 = -J_01 / detJ;
const double Jinv_10 = -J_10 / detJ;
const double Jinv_11 =  J_00 / detJ;

// Take absolute value of determinant
detJ = std::abs(detJ);

// Set scale factor
const double det = detJ;

// Compute geometry tensors
const double G0_0_0 = det*(Jinv_00*Jinv_00 + Jinv_01*Jinv_01);
const double G0_0_1 = det*(Jinv_00*Jinv_10 + Jinv_01*Jinv_11);
const double G0_1_0 = det*(Jinv_10*Jinv_00 + Jinv_11*Jinv_01);
const double G0_1_1 = det*(Jinv_10*Jinv_10 + Jinv_11*Jinv_11);
```



# Automate the solution of differential equations



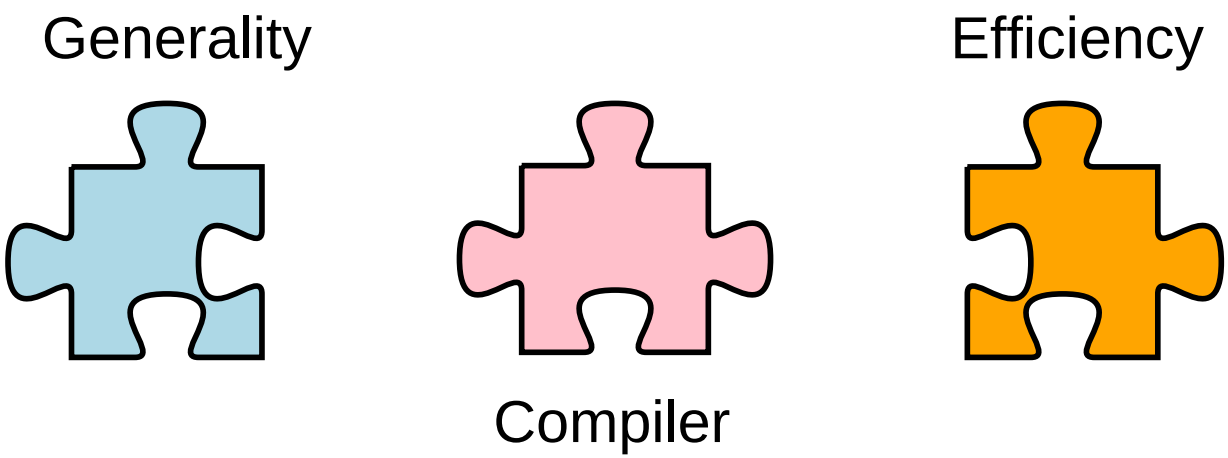
# Simplicity

- Simple and intuitive user interface
  - Close to mathematical abstractions
  - Close to mathematical notation
  - But give users what they expect: `Matrix`, `Vector`, `Mesh`, `Form` (?)
- Simplicity of implementation
  - Simple components: each component does one thing and does it well
  - Simple components: independent development
  - Simplicity by (new) mathematical ideas
  - Simplicity by (new) programming techniques

# Generality *and* efficiency

- Any equation
- Any (finite element) method
- Maximum efficiency

Possible to combine generality with efficiency by generating code



# Generality

- Any (multilinear) form
- Any element:

| | |
|---|---|
| $P_k$ | Arbitrary degree Lagrange elements |
| $DG_k$ | Arbitrary degree discontinuous elements |
| $BDM_k$ | Arbitrary degree Brezzi–Douglas–Marini |
| $RT_k$ | Arbitrary degree Raviart-Thomas |
| $CR_1$ | Crouzeix–Raviart |
| ... | (more in preparation) |

- 2D (triangles) and 3D (tetrahedra)

# Efficiency

- CPU time for computing the "element stiffness matrix"
- Straight-line C++ code generated by the FEniCS Form Compiler (FFC)
- Speedup vs a standard quadrature-based C++ code with loops over quadrature points

| Form | $q = 1$ | $q = 2$ | $q = 3$ | $q = 4$ | $q = 5$ | $q = 6$ | $q = 7$ | $q = 8$ |
|---|---|---|---|---|---|---|---|---|
| Mass 2D | 12 | 31 | 50 | 78 | 108 | 147 | 183 | 232 |
| Mass 3D | 21 | 81 | 189 | 355 | 616 | 881 | 1442 | 1475 |
| Poisson 2D | 8 | 29 | 56 | 86 | 129 | 144 | 189 | 236 |
| Poisson 3D | 9 | 56 | 143 | 259 | 427 | 341 | 285 | 356 |
| Navier–Stokes 2D | 32 | 33 | 53 | 37 | — | — | — | — |
| Navier–Stokes 3D | 77 | 100 | 61 | 42 | — | — | — | — |
| Elasticity 2D | 10 | 43 | 67 | 97 | — | — | — | — |
| Elasticity 3D | 14 | 87 | 103 | 134 | — | — | — | — |

# The FEniCS project

- Initiated in 2003
- Collaborators (in order of appearance):
  - (Chalmers University of Technology)
  - University of Chicago
  - Argonne National Laboratory
  - (Toyota Technological Institute)
  - Delft University of Technology
  - Royal Institute of Technology (KTH)
  - Simula Research Laboratory
  - Texas Tech
- Automation of Computational Mathematical Modeling (ACMM)
- www.fenics.org

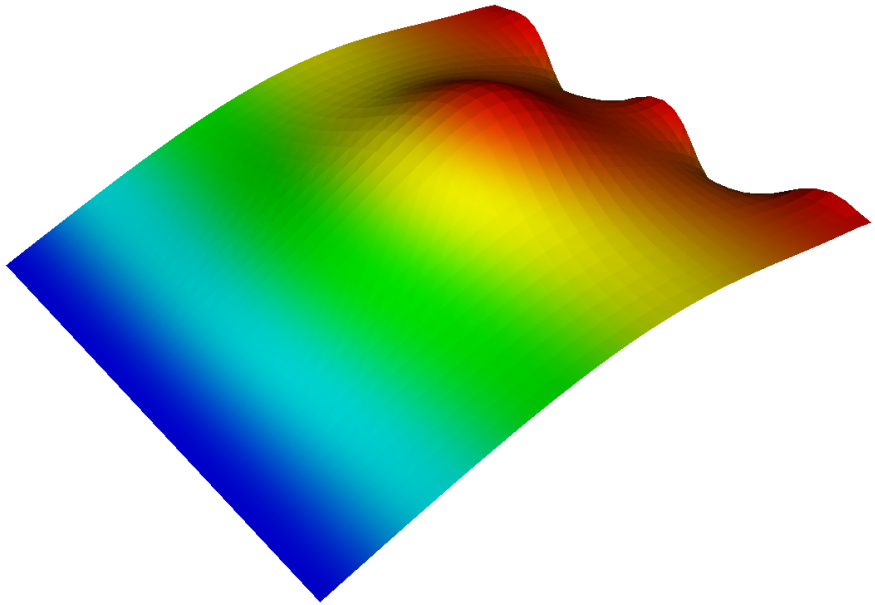
# Poisson's equation

## Differential equation

Differential equation:

$$-\Delta u = f$$

- Heat transfer
- Electrostatics
- Magnetostatics
- Fluid flow
- etc.



# Poisson's equation

## Variational formulation

Find $u \in V$ such that

$$a(v, u) = L(v) \quad \forall v \in \hat{V}$$

where

$$a(v, u) = \int_\Omega \nabla v \cdot \nabla u \, \mathrm{d}x$$

$$L(v) = \int_\Omega v f \, \mathrm{d}x$$

# Poisson's equation

## Implementation

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

v = TestFunction(element)
u = TrialFunction(element)
f = Function(element)

a = dot(grad(v), grad(u))*dx
L = v*f*dx
```

# Linear elasticity

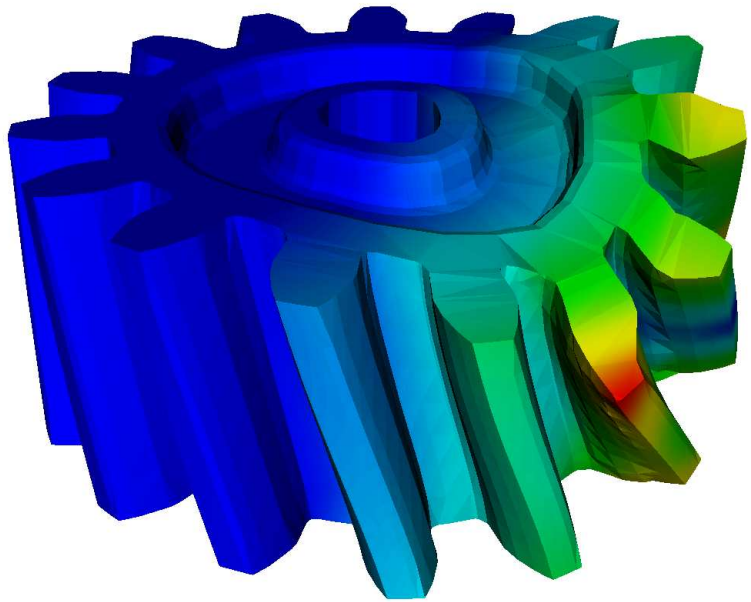
## Differential equation

Differential equation:

$$-\nabla \cdot \sigma(u) = f$$

where

$$\sigma(v) = 2\mu\epsilon(v) + \lambda \mathrm{tr}\, \epsilon(v)\, I$$
$$\epsilon(v) = \frac{1}{2}(\nabla v + (\nabla v)^\top)$$



- Displacement $u = u(x)$
- Stress $\sigma = \sigma(x)$

# Linear elasticity

## Variational formulation

Find $u \in V$ such that

$$a(v, u) = L(v) \quad \forall v \in \hat{V}$$

where

$$a(v, u) = \int_\Omega \nabla v : \sigma(u) \, \mathrm{d}x$$

$$L(v) = \int_\Omega v \cdot f \, \mathrm{d}x$$

# Linear elasticity

## Implementation

```
element = VectorElement("Lagrange", "tetrahedron", 1)

v = TestFunction(element)
u = TrialFunction(element)
f = Function(element)

def epsilon(v):
    return 0.5*(grad(v) + transp(grad(v)))

def sigma(v):
    return 2*mu*epsilon(v) + lmbda*mult(trace(epsilon(v)), I)

a = dot(grad(v), sigma(u))*dx
L = dot(v, f)*dx
```

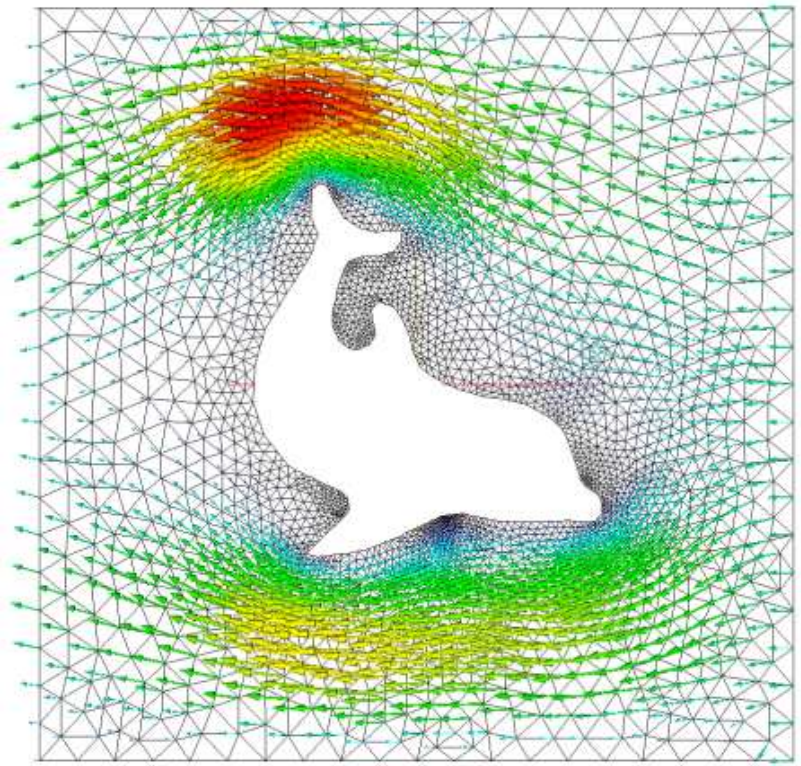
# The Stokes equations

## Differential equation

Differential equation:

$$-\Delta u + \nabla p = f$$
$$\nabla \cdot u = 0$$

- Fluid velocity $u = u(x)$
- Pressure $p = p(x)$



# The Stokes equations

## Variational formulation

Find $(u, p) \in V$ such that

$$a((v, q), (u, p)) = L((v, q)) \quad \forall (v, q) \in \hat{V}$$

where

$$a((v, q), (u, p)) = \int_\Omega \nabla v \cdot \nabla u - \nabla \cdot v\, p + q \nabla \cdot u \,\mathrm{d}x$$

$$L((v, q)) = \int_\Omega v \cdot f \,\mathrm{d}x$$

# The Stokes equations

## Implementation

```
P2 = VectorElement("Lagrange", "triangle", 2)
P1 = FiniteElement("Lagrange", "triangle", 1)
TH = P2 + P1

(v, q) = TestFunctions(TH)
(u, p) = TrialFunctions(TH)

f = Function(P2)

a = (dot(grad(v), grad(u)) - div(v)*p + q*div(u))*dx
L = dot(v, f)*dx
```

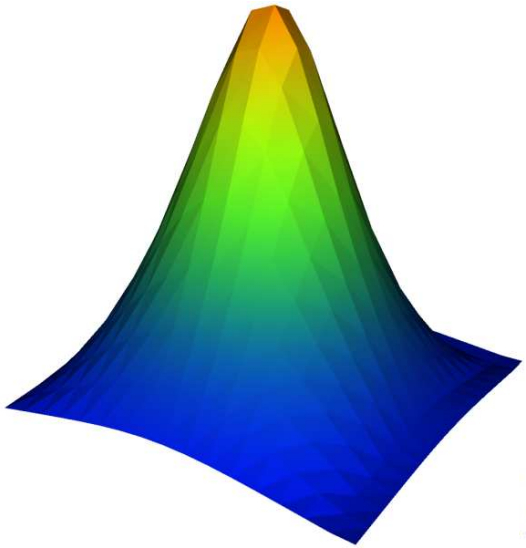
# Mixed Poisson with $H(\mathrm{div})$ elements

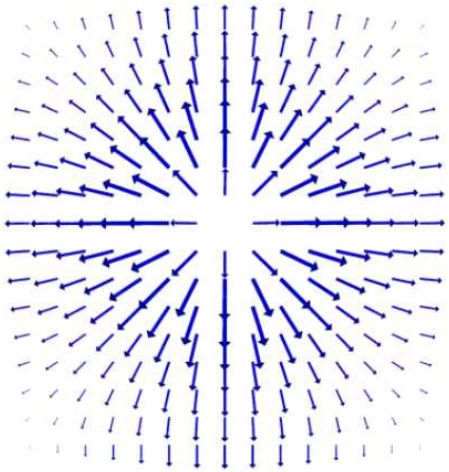
## Differential equation

Differential equation:

$$\sigma + \nabla u = 0$$
$$\nabla \cdot \sigma = f$$

- $u \in L_2$
- $\sigma \in H(\mathrm{div})$





# Mixed Poisson with $H(\mathrm{div})$ elements

## Variational formulation

Find $(\sigma, u) \in V$ such that

$$a((\tau, w), (\sigma, u)) = L((\tau, w)) \quad \forall (\tau, w) \in \hat{V}$$

where

$$a((\tau, w), (\sigma, u)) = \int_\Omega \tau \cdot \sigma - \nabla \cdot \tau \, u + w \nabla \cdot \sigma \, \mathrm{d}x$$

$$L((\tau, w)) = \int_\Omega w \, f \, \mathrm{d}x$$

# Mixed Poisson with $H(\mathrm{div})$ elements

## Implementation

```
BDM1 = FiniteElement("Brezzi-Douglas-Marini", "triangle", 1)
DG0 = FiniteElement("Discontinuous Lagrange", "triangle", 0)

element = BDM1 + DG0

(tau, w) = TestFunctions(element)
(sigma, u) = TrialFunctions(element)

f = Function(DG0)

a = (dot(tau, sigma) - div(tau)*u + w*div(sigma))*dx
L = w*f*dx
```

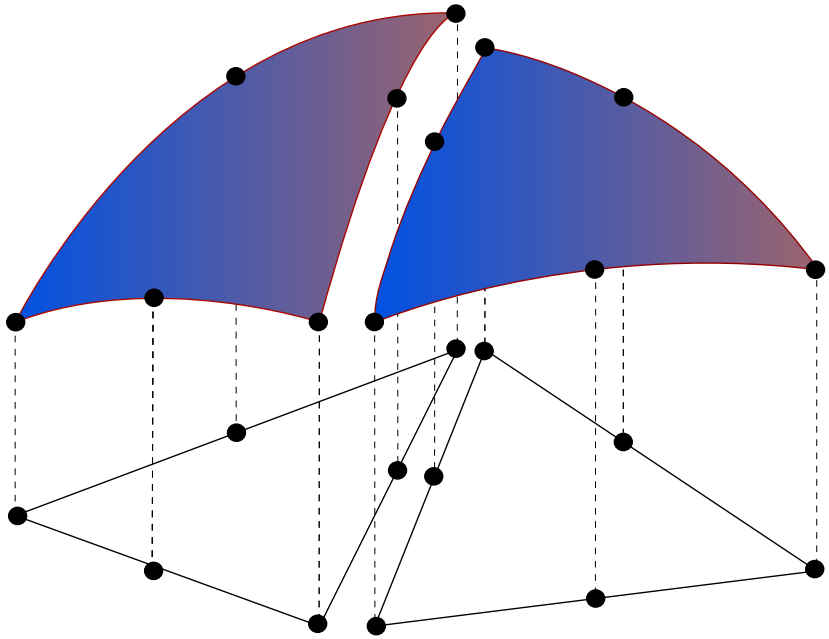
# Poisson's equation with DG elements

## Differential equation

Differential equation:

$$-\Delta u = f$$

- $u \in L_2$
- $u$ discontinuous across element boundaries



# Poisson's equation with DG elements

## Variational formulation (interior penalty method)

Find $u \in V$ such that

$$a(v, u) = L(v) \quad \forall v \in \hat{V}$$

where

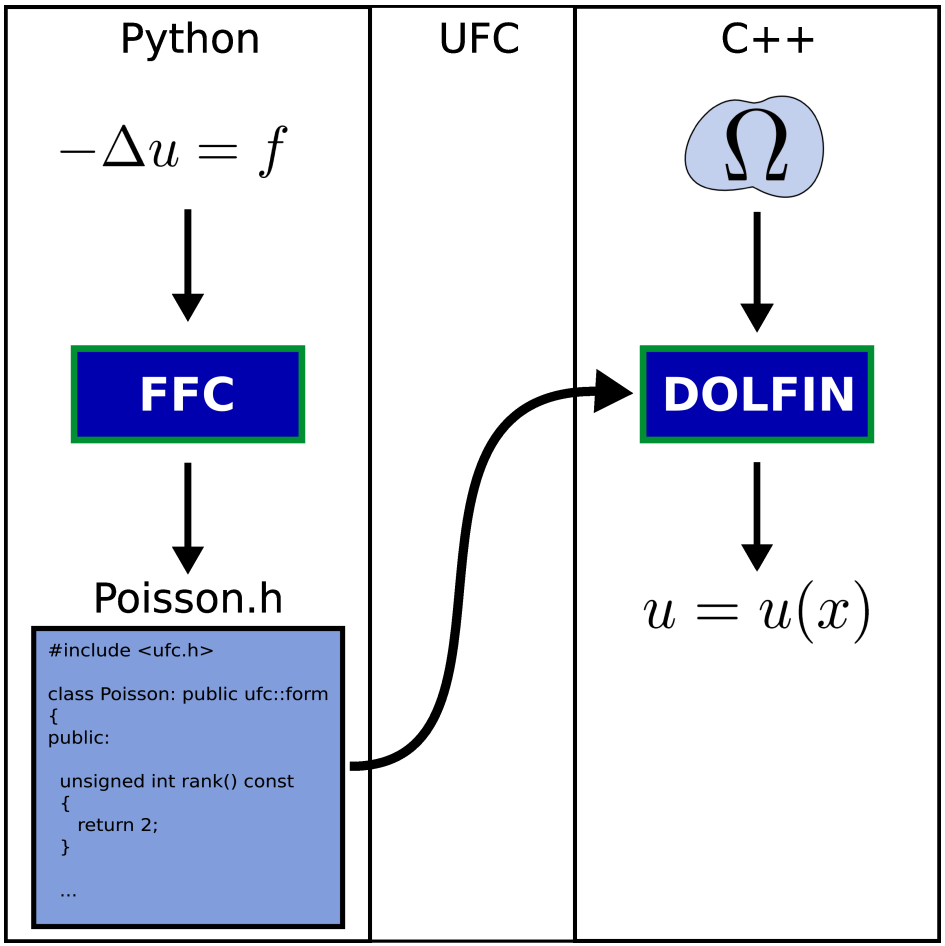
$$\begin{aligned} a(v,u) &= \int_\Omega \nabla v \cdot \nabla u \,\mathrm{d}x \\ &\quad + \sum_S \int_S -\langle \nabla v \rangle \cdot [\![u]\!]_n - [\![v]\!]_n \cdot \langle \nabla u \rangle + (\alpha/h)[\![v]\!]_n \cdot [\![u]\!]_n \,\mathrm{d}S \\ &\quad + \int_{\partial\Omega} -\nabla v \cdot [\![u]\!]_n - [\![v]\!]_n \cdot \nabla u + (\gamma/h) v u \,\mathrm{d}s \\ L(v) &= \int_\Omega v f \,\mathrm{d}x + \int_{\partial\Omega} v g \,\mathrm{d}s \end{aligned}$$

# Poisson's equation with DG elements

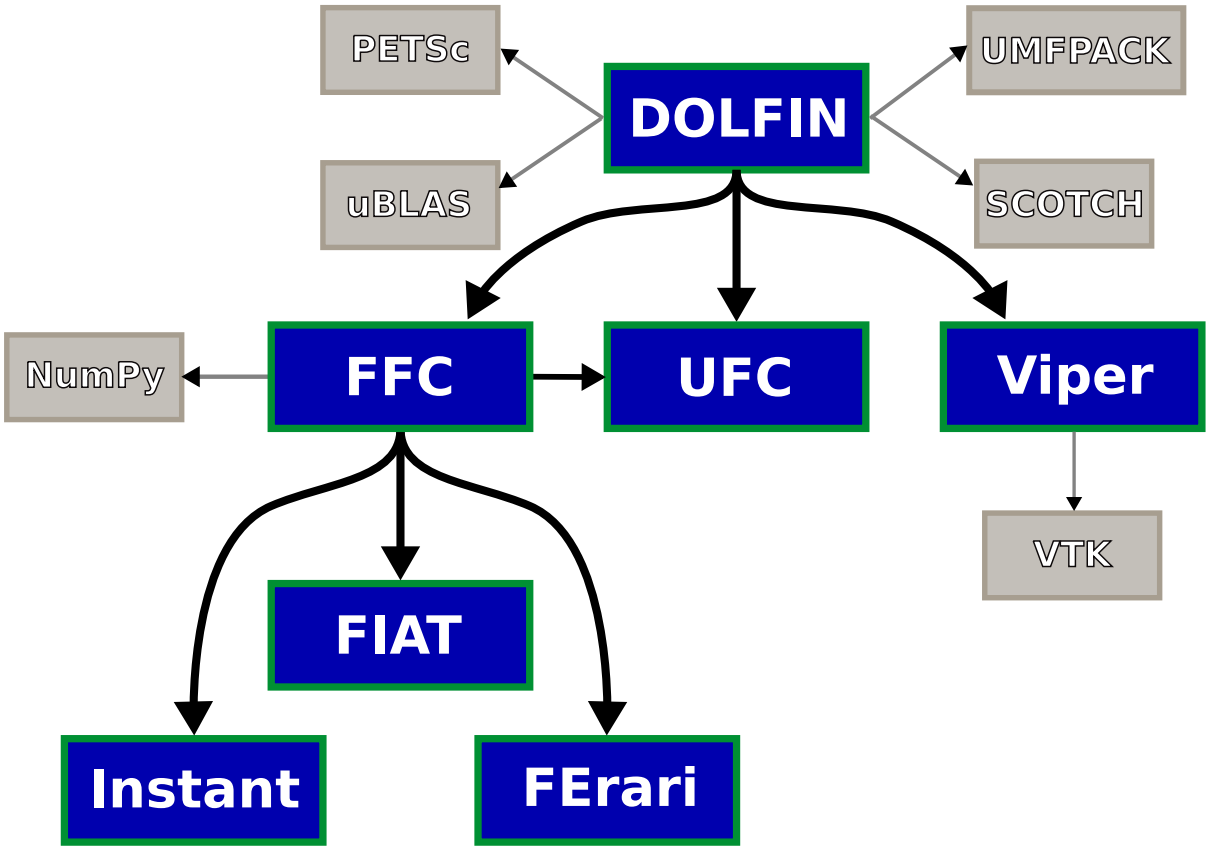
## Implementation

```
DG1 = FiniteElement("Discontinuous Lagrange", "triangle", 1)

v = TestFunction(DG1)
u = TrialFunction(DG1)

f = Function(DG1)
g = Function(DG1)
n = FacetNormal("triangle")
h = MeshSize("triangle")

a = dot(grad(v), grad(u))*dx - dot(avg(grad(v)), jump(u, n))*dS
  - dot(jump(v, n), avg(grad(u)))*dS
  + alpha/h('+')*dot(jump(v, n), jump(u, n))*dS
  - dot(grad(v), jump(u, n))*ds - dot(jump(v, n), grad(u))*ds
  + gamma/h*v*u*ds
L = v*f*dx + v*g*ds
```

# Code generation system



# Software map




SyFi

Ko

Puffin

PyCC

# Recent updates

- UFC, a framework for finite element assembly
- DG, BDM and RT elements
- A new improved mesh library, adaptive refinement
- Mesh and graph partitioning
- Improved linear algebra supporting PETSc and uBLAS
- Optimized code generation (FErari)
- Improved ODE solvers
- Python bindings
- Bugzilla database, pkg-config, improved manual, compiler support, demos, file formats, built-in plotting, ...

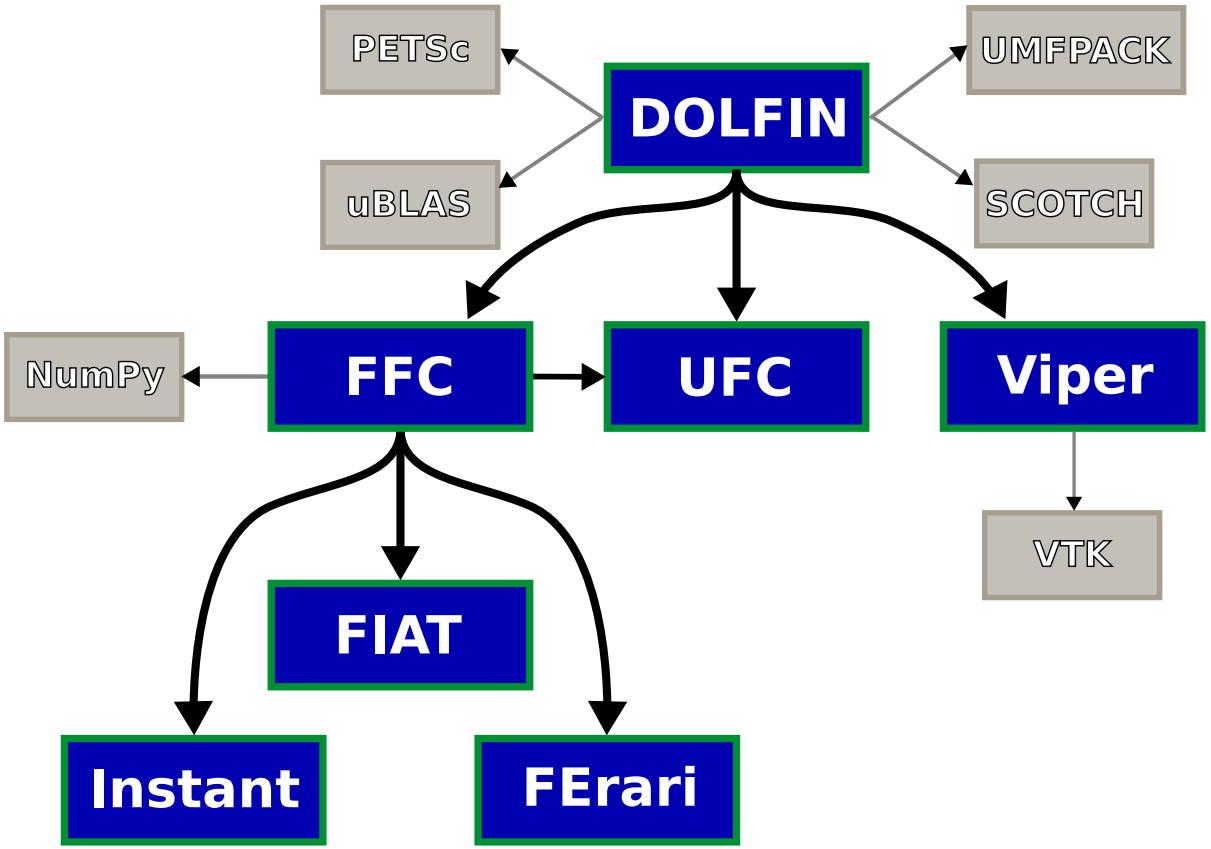
# Future plans (highlights)

- UFL/UFC
- Automation of error control
  - Automatic generation of dual problems
  - Automatic generation of a posteriori error estimates
- Improved geometry support (MeshBuilder)
- Debian/Ubuntu packages
- Finite element exterior calculus

→ `www.fenics.org` ←

*Additional slides*

# Software components



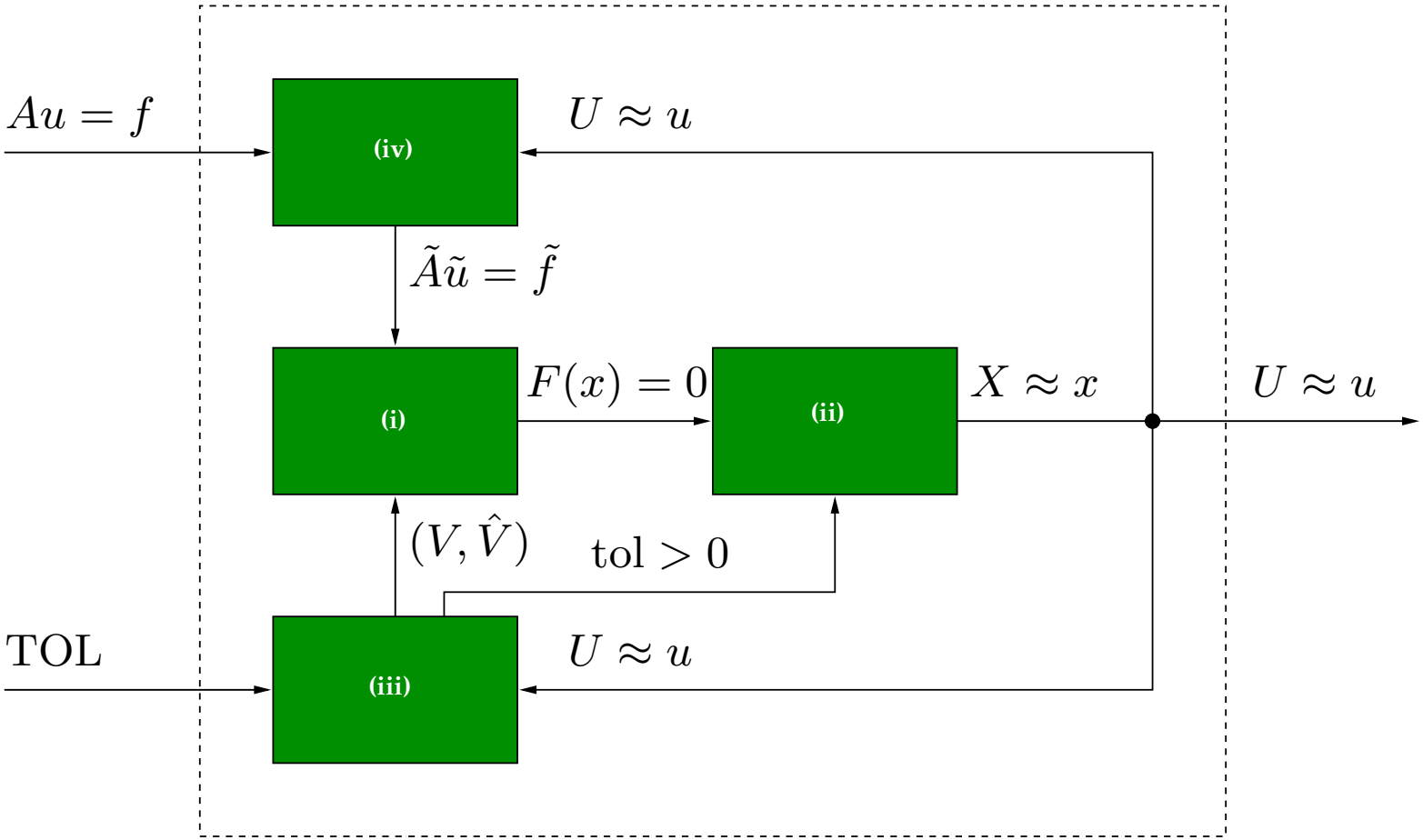

SyFi

Ko

Puffin

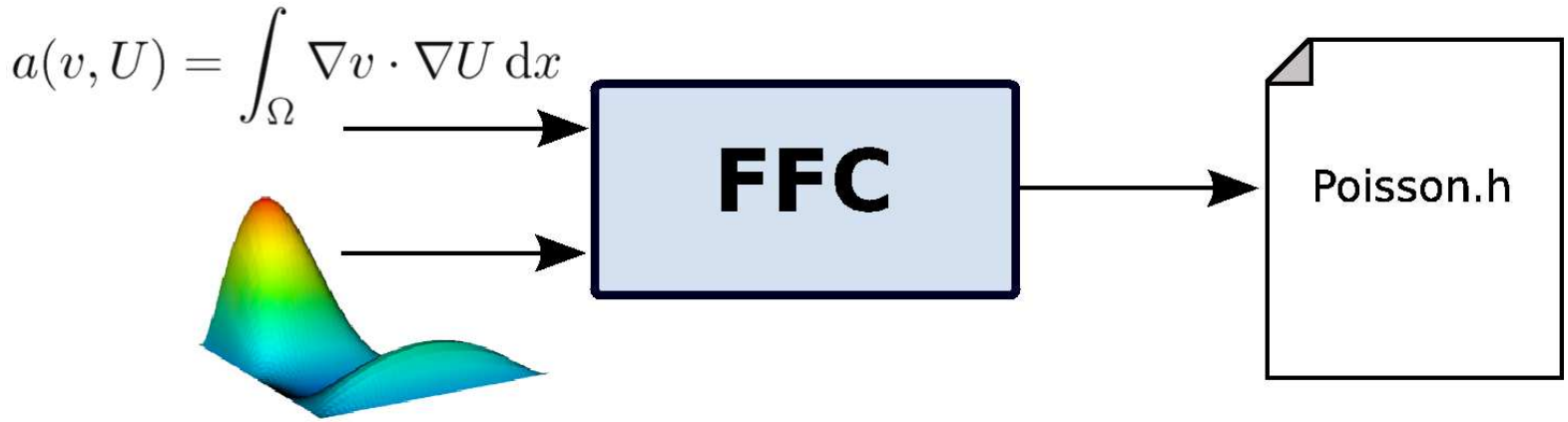
PyCC

# Automation of CMM



# A common framework: UFL/UFC

- UFL - Unified Form Language
- UFC - Unified Form-assembly Code
- Unify, standardize, extend
- Working prototypes: FFC (Logg), SyFi (Mardal)



# Compiling Poisson's equation: non-optimized, 16 ops

```
void eval(real block[], const AffineMap& map) const
{
  [...]

  block[0] = 0.5*G0_0_0 + 0.5*G0_0_1 +
             0.5*G0_1_0 + 0.5*G0_1_1;
  block[1] = -0.5*G0_0_0 - 0.5*G0_1_0;
  block[2] = -0.5*G0_0_1 - 0.5*G0_1_1;
  block[3] = -0.5*G0_0_0 - 0.5*G0_0_1;
  block[4] = 0.5*G0_0_0;
  block[5] = 0.5*G0_0_1;
  block[6] = -0.5*G0_1_0 - 0.5*G0_1_1;
  block[7] = 0.5*G0_1_0;
  block[8] = 0.5*G0_1_1;
}
```

# Compiling Poisson's equation: `ffc -O`, 11 ops

```
void eval(real block[], const AffineMap& map) const
{
  [...]

  block[1] = -0.5*G0_0_0 + -0.5*G0_1_0;
  block[0] = -block[1] + 0.5*G0_0_1 + 0.5*G0_1_1;
  block[7] = -block[1] + -0.5*G0_0_0;
  block[6] = -block[7] + -0.5*G0_1_1;
  block[8] = -block[6] + -0.5*G0_1_0;
  block[2] = -block[8] + -0.5*G0_0_1;
  block[5] = -block[2] + -0.5*G0_1_1;
  block[3] = -block[5] + -0.5*G0_0_0;
  block[4] = -block[1] + -0.5*G0_1_0;
}
```

# Compiling Poisson's equation: `ffc -f blas`, 36 ops

```
void eval(real block[], const AffineMap& map) const
{
  [...]

  cblas_dgemv(CblasRowMajor, CblasNoTrans,
              blas.mi, blas.ni, 1.0,
              blas.Ai, blas.ni, blas.Gi,
              1, 0.0, block, 1);
}
```

# Key Features

- Simple and intuitive object-oriented API, C++ or Python
- Automatic and efficient evaluation of variational forms
- Automatic and efficient assembly of linear systems
- General families of finite elements, including arbitrary order continuous and discontinuous Lagrange elements, BDM, RT
- Arbitrary mixed elements
- High-performance parallel linear algebra
- General meshes, adaptive mesh refinement
- mcG($q$)/mdG($q$) and cG($q$)/dG($q$) ODE solvers
- Support for a range of output formats for post-processing, including DOLFIN XML, ParaView/Mayavi/VTK, OpenDX, Octave, MATLAB, GiD
- Built-in plotting

# Linear algebra

- Complete support for PETSc
  - High-performance parallel linear algebra
  - Krylov solvers, preconditioners
- Complete support for uBLAS
  - BLAS level 1, 2 and 3
  - Dense, packed and sparse matrices
  - C++ operator overloading and expression templates
  - Krylov solvers, preconditioners added by DOLFIN
- Uniform interface to both linear algebra backends
- LU factorization by UMFPACK for uBLAS matrix types
- Eigenvalue problems solved by SLEPc for PETSc matrix types
- Matrix-free solvers ("virtual matrices")