The FEniCS Project

Todd Dupont, Johan Hoffman, Johan Jansson, Claes Johnson,

Robert C. Kirby, Matthew Knepley, Mats Larson, Anders Logg, Ridgway Scott

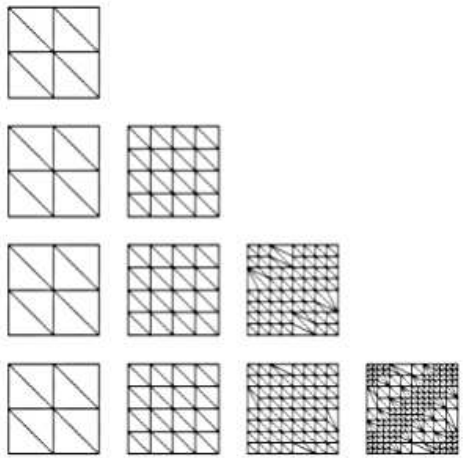
fenics@fenics.org

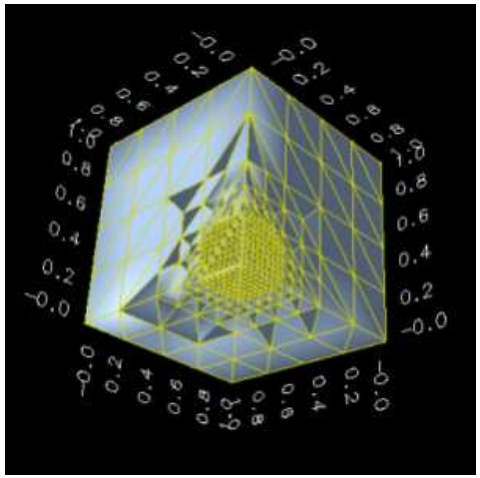
http://www.fenics.org/

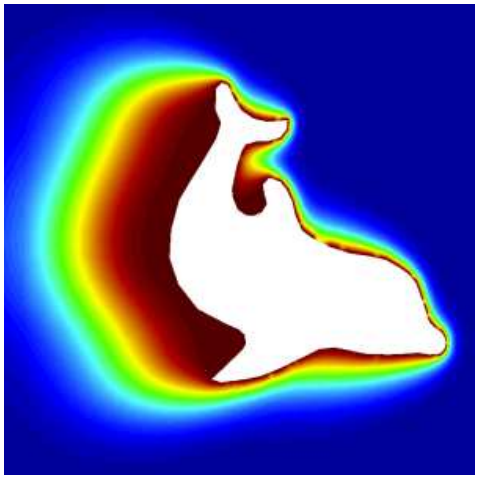
The **FEniCS** project

A free software project for the Automation of Computational Mathematical Modeling (ACMM), developed at

- The Toyota Technological Institute at Chicago
- The University of Chicago
- Chalmers University of Technology (Göteborg, Sweden)
- NADA, KTH (Stockholm, Sweden)
- Argonne National Laboratory (Chicago)







People

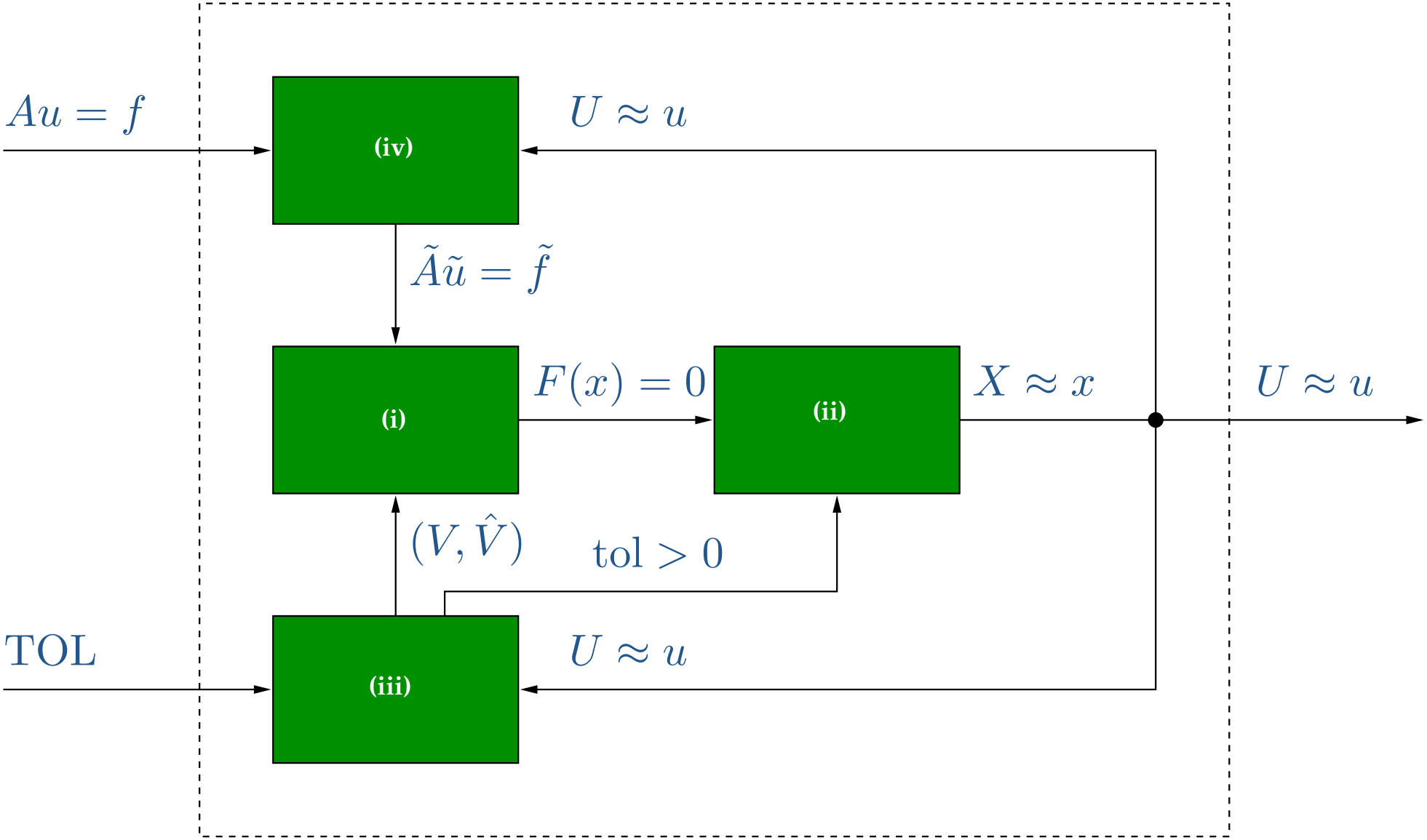
- Todd Dupont, University of Chicago
- Johan Hoffman, NADA
- Johan Jansson, Chalmers University of Technology
- Claes Johnson, Chalmers University of Technology
- Matthew Knepley, Argonne National Laboratory
- Robert C. Kirby, University of Chicago
- Mats G. Larson, Umeå University
- Anders Logg, Toyota Technological Institute at Chicago
- Ridgway Scott, University of Chicago
- F. Bengzon, N. Ericsson, G. Foufas, D. Heintz, R. Hemph, P. Ingelström, K. Kraft, A. Krusper, A. Mark, A. Nilsson, E. Svensson, J. Tilander, T. Svedberg, H. Svensson, W. Villanueva

Main goal: the Automation of CMM

- Input: Model $Au = f$ and tolerance $\mathrm{TOL} > 0$
- Output: Solution $U \approx u$ satisfying $\|U - u\| \leq \mathrm{TOL}$

- Produce a solution U satisfying a given accuracy requirement, using a minimal amount of work

The Automation of CMM

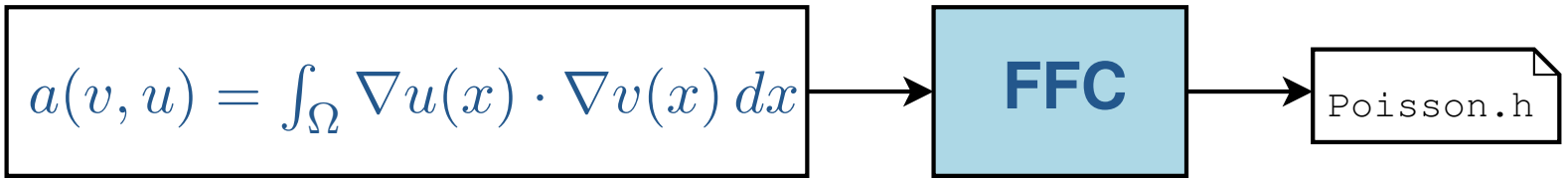


FEniCS components

- **DOLFIN**, the C++ interface of FEniCS
  - Hoffman, Jansson, Logg, et al.
- **FErari**, optimized form evaluation
  - Kirby, Knepley, Scott
- **FFC**, the FEniCS Form Compiler
  - Logg
- **FIAT**, automatic generation of finite elements
  - Kirby, Knepley
- **Ko**, simulation of mechanical systems
  - Jansson
- **Puffin**, light-weight version for Octave/MATLAB
  - Hoffman, Logg
- (PETSc)

FFC: the FEniCS Form Compiler

- Automates a key step in the implementation of finite element methods for partial differential equations
- Input: a variational form and a finite element
- Output: optimal C/C++ code

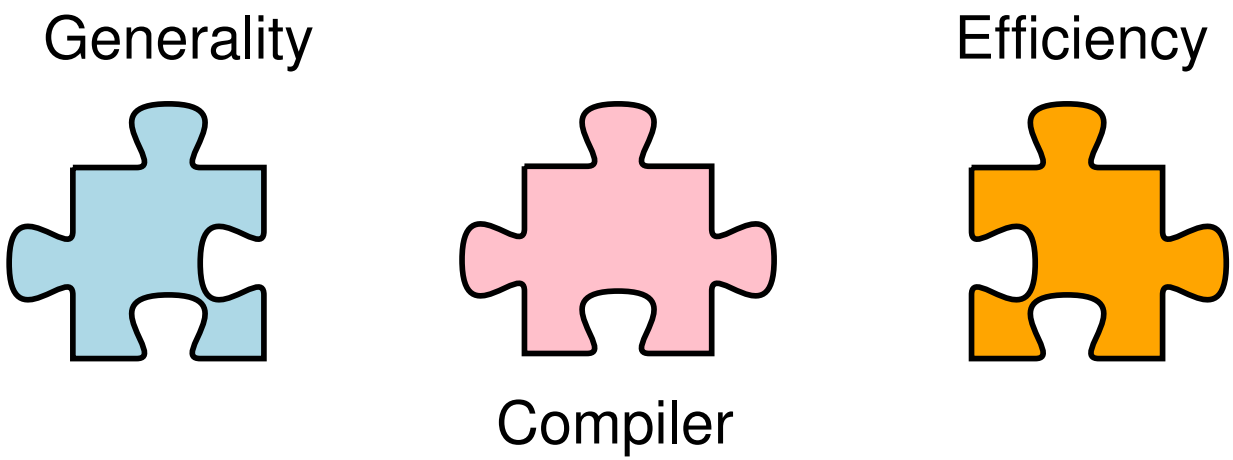



```
>> ffc [-l language] poisson.form
```

Design goals

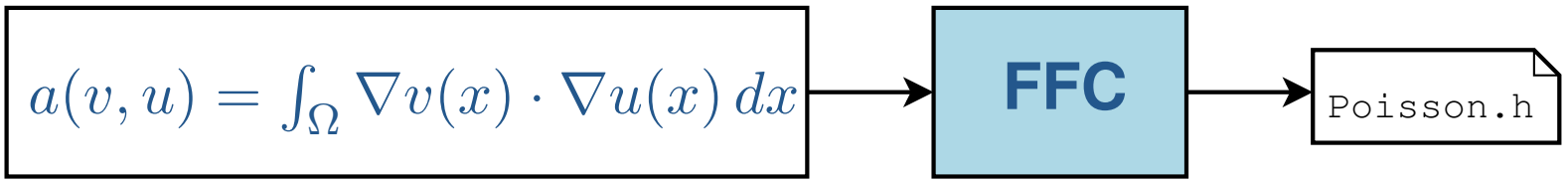
- Any form
- Any element
- Maximum efficiency

Possible to combine generality with efficiency by using a compiler approach:

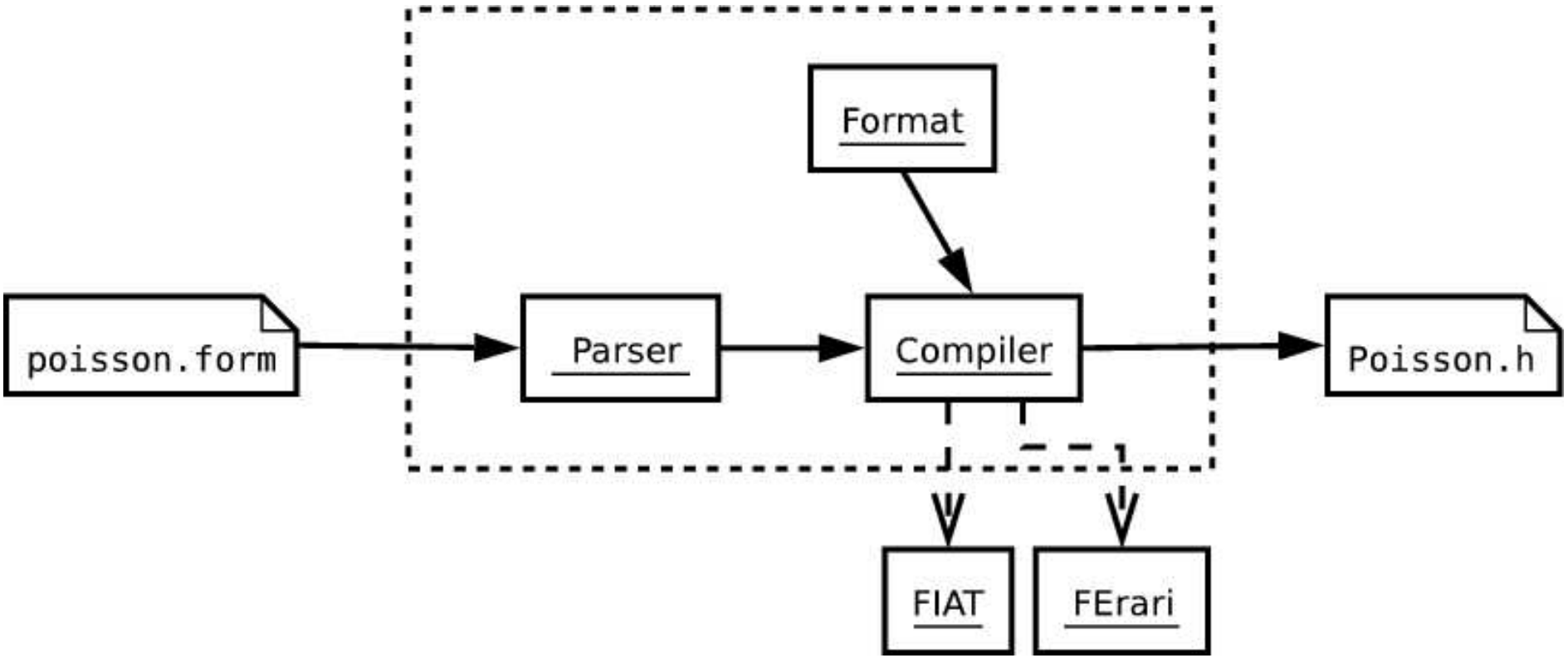


FFC

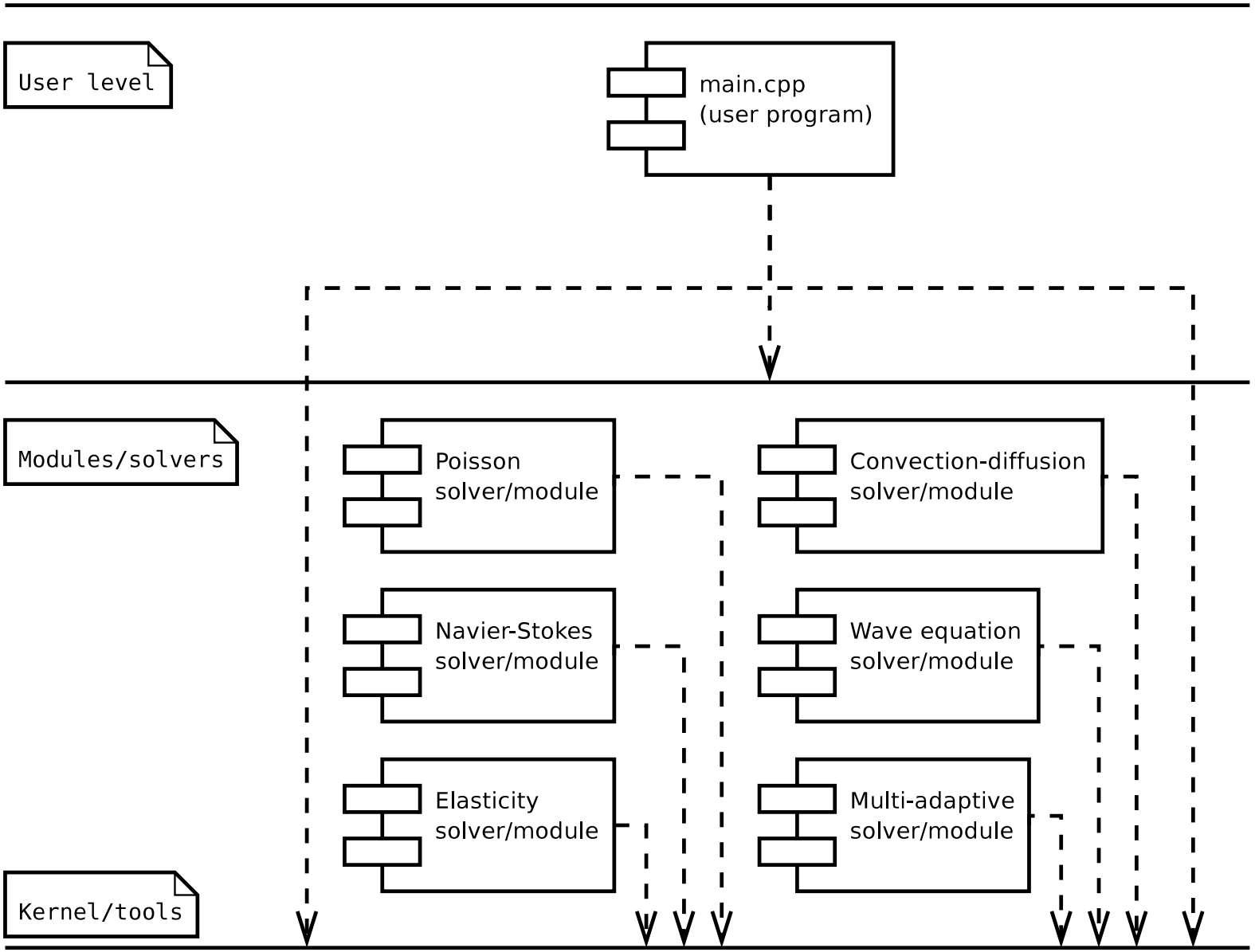
- Implemented in Python
- Access: Python module or command-line interface
- **FIAT** used as the finite element backend
- Optimization through **FErari**
- Supports multiple output languages (formats)
- Primary target: **DOLFIN**/PETSc



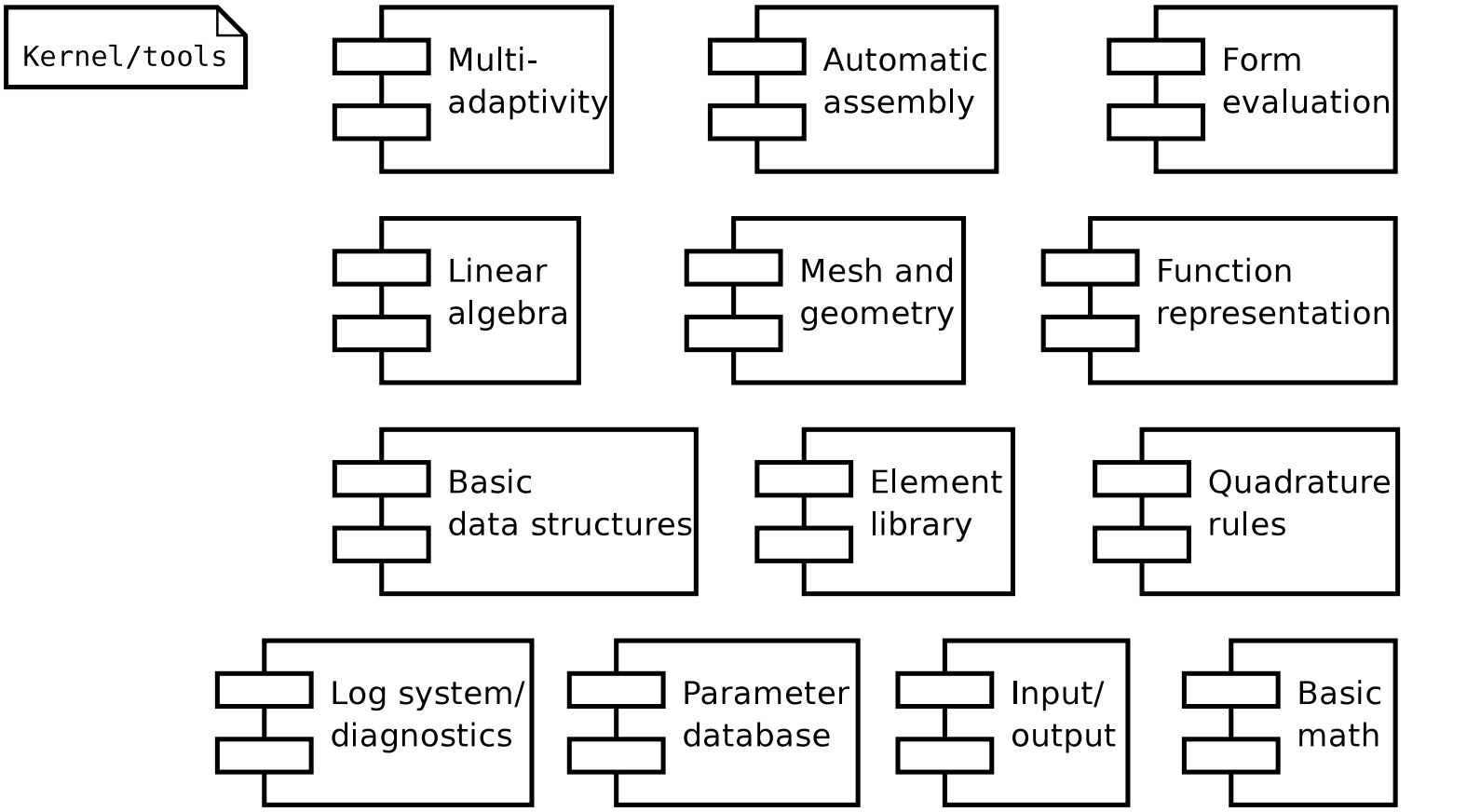
Components of **FFC**



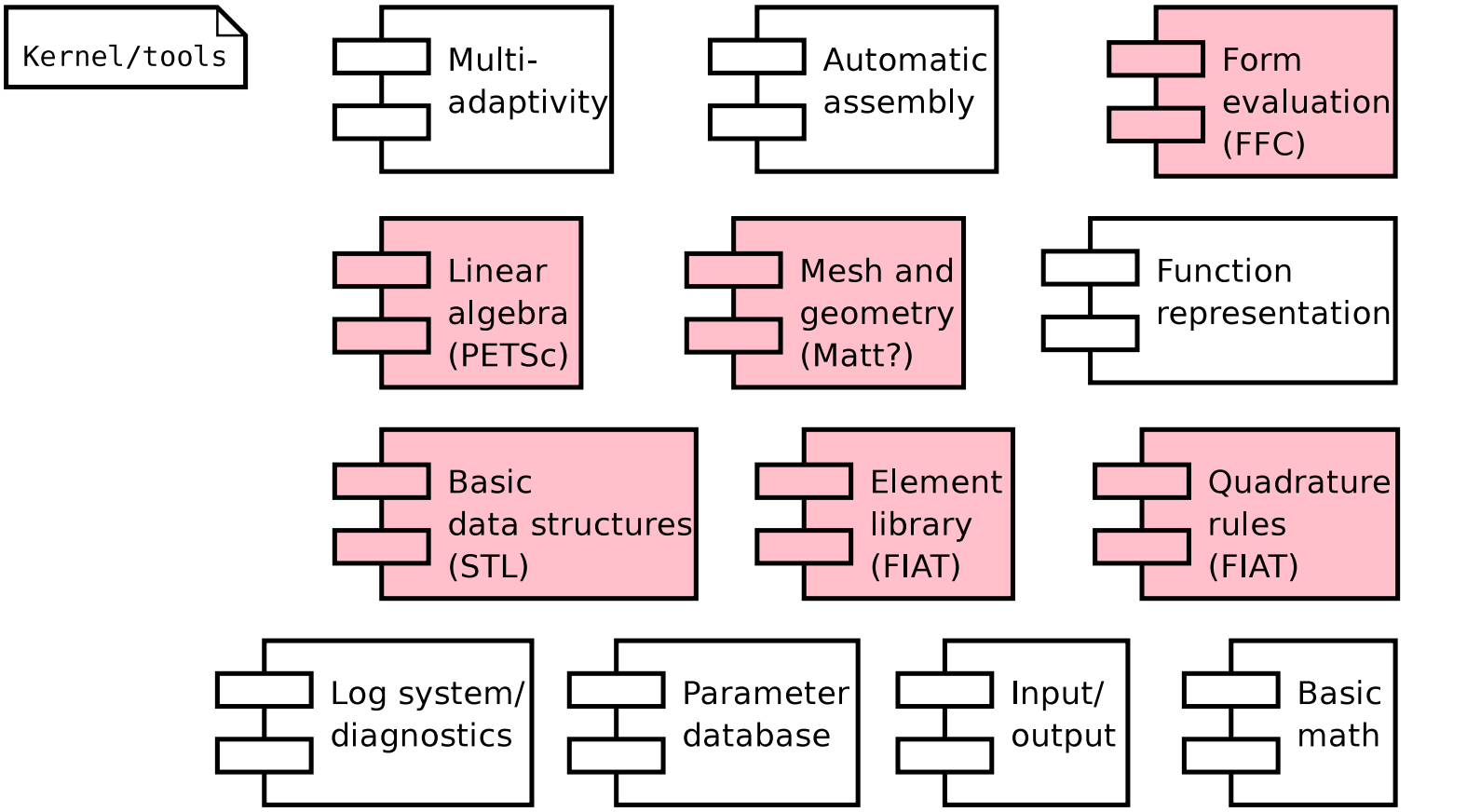
DOLFIN


User level
main.cpp
(user program)
Modules/solvers
Poisson
solver/module
Convection-diffusion
solver/module
Navier-Stokes
solver/module
Wave equation
solver/module
Elasticity
solver/module
Multi-adaptive
solver/module
Kernel/tools

DOLFIN

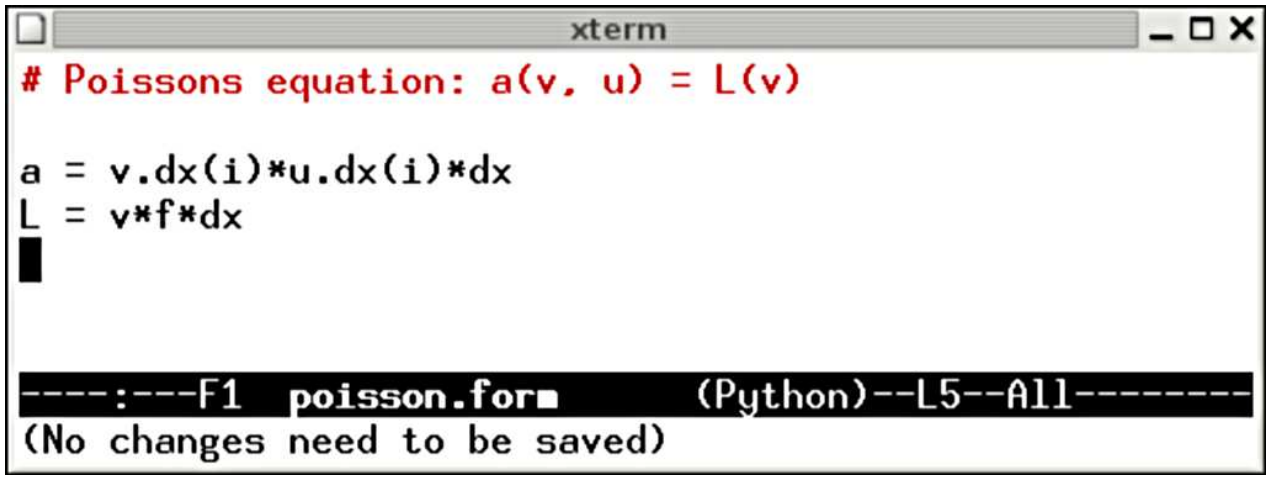

Kernel/tools
Multi-
adaptivity
Automatic
assembly
Form
evaluation
Linear
algebra
Mesh and
geometry
Function
representation
Basic
data structures
Element
library
Quadrature
rules
Log system/
diagnostics
Parameter
database
Input/
output
Basic
math

DOLFIN becomes an interface



Basic usage: compiling a form

1. Implement the form using your favorite text editor (emacs):



2. Compile the form using **FFC**:

```
>> ffc poisson.form
```

This will generate C++ code (`Poisson.h`) for **DOLFIN**

Basic usage: solving the PDE

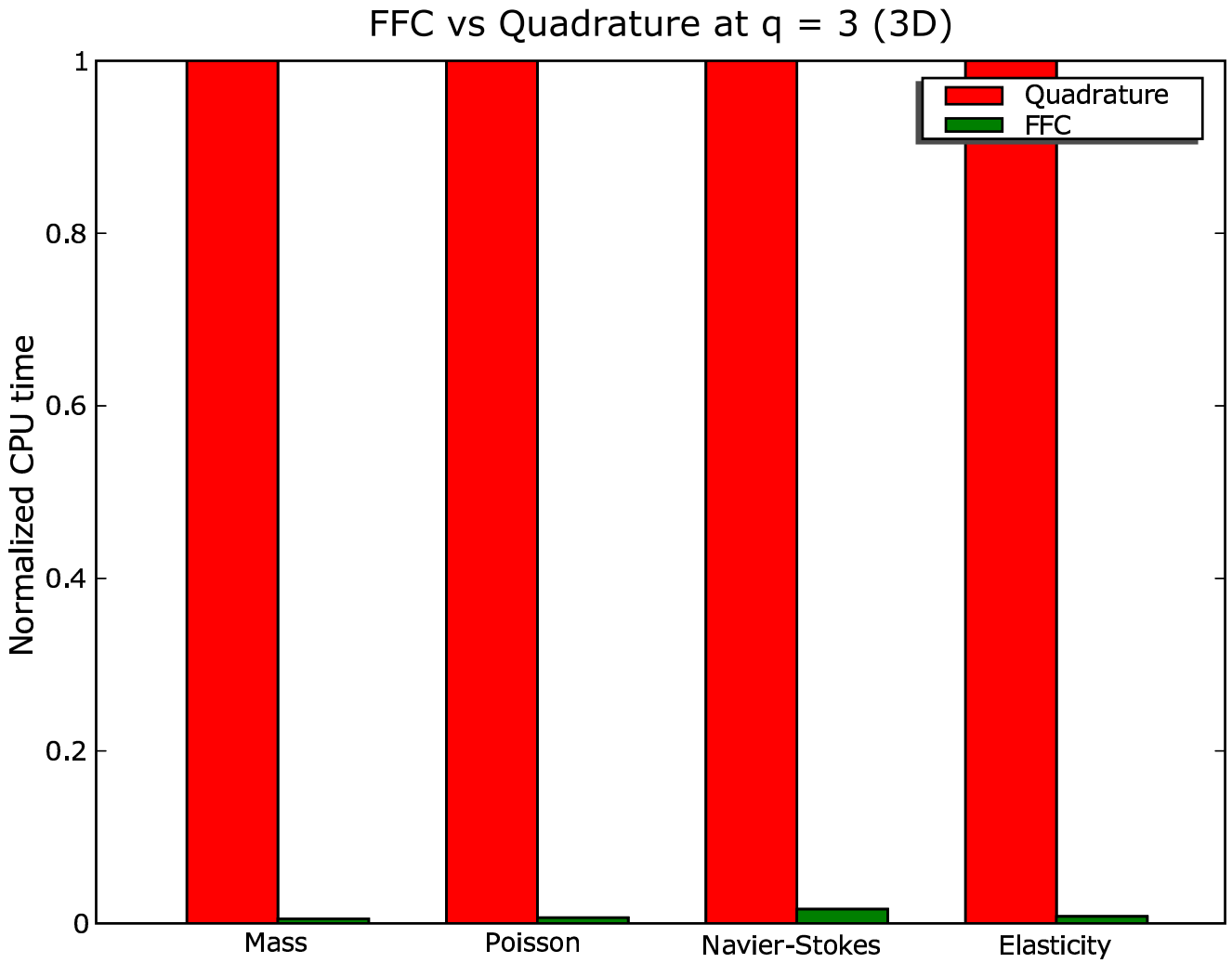
```
#include <dolfin.h>
#include "Poisson.h"
using namespace dolfin;
int main()
{
  Poisson::BilinearForm a;
  Poisson::LinearForm L(f);

  Matrix A;
  Vector x, b;
  FEM::assemble(a, L, A, b, mesh, bc);
  GMRES::solve(A, x, b);

  Function u(x, mesh, a.trial());
  File file("poisson.m");
  file << u;

  return 0;
}
```

Impressive speedups



Test case 1: the mass matrix

- Mathematical notation:

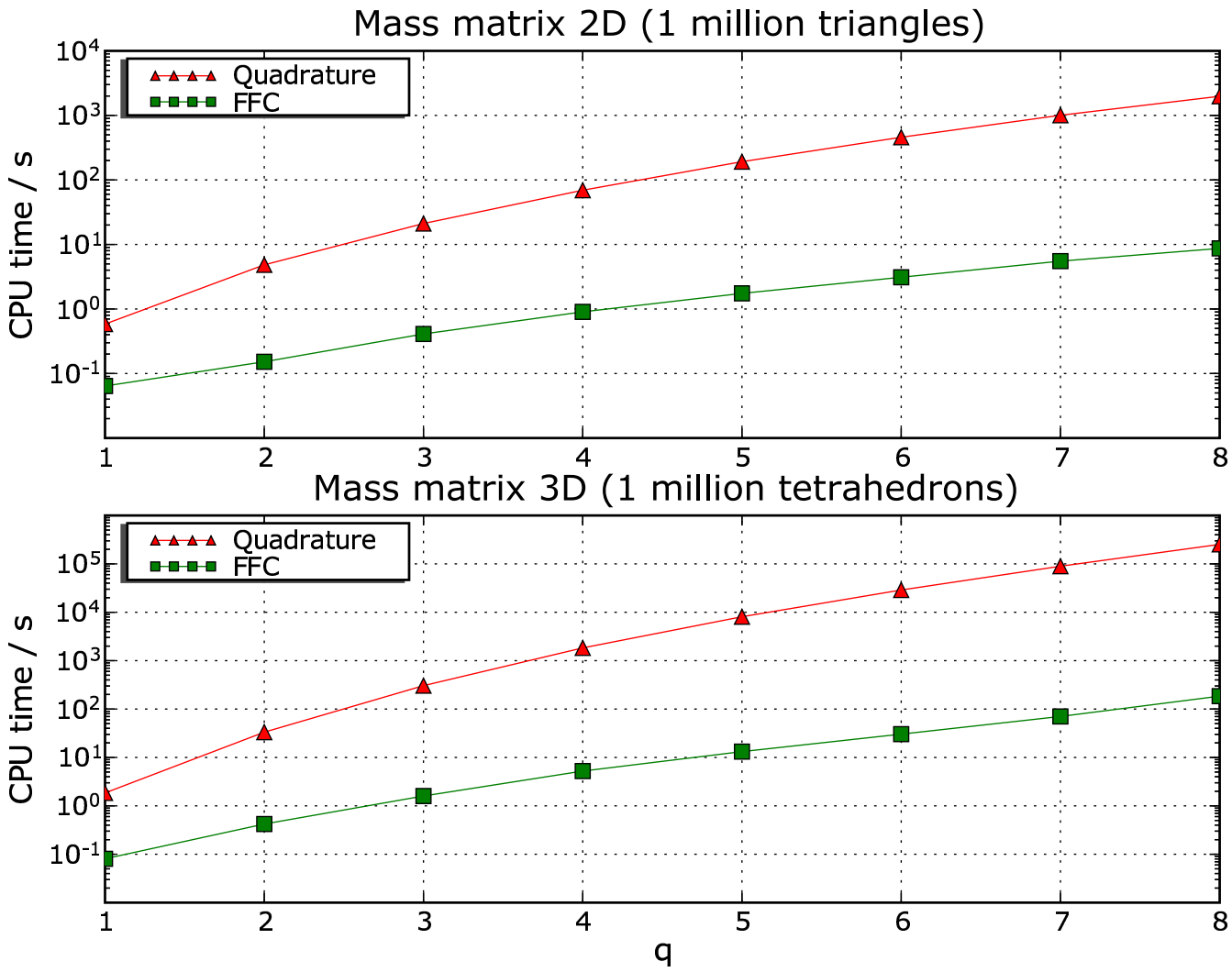
$$a(v, u) = \int_\Omega uv \, dx$$

- FFC implementation:

```
v = BasisFunction(element)
u = BasisFunction(element)

a = u*v*dx
```

Results


Mass matrix 2D (1 million triangles)
Quadrature
FFC
CPU time / s
10^4
10^3
10^2
10^1
10^0
10^{-1}
1
2
3
4
5
6
7
8
Mass matrix 3D (1 million tetrahedrons)
Quadrature
FFC
CPU time / s
10^5
10^4
10^3
10^2
10^1
10^0
10^{-1}
1
2
3
4
5
6
7
8
q

Test case 2: Poisson

- Mathematical notation:

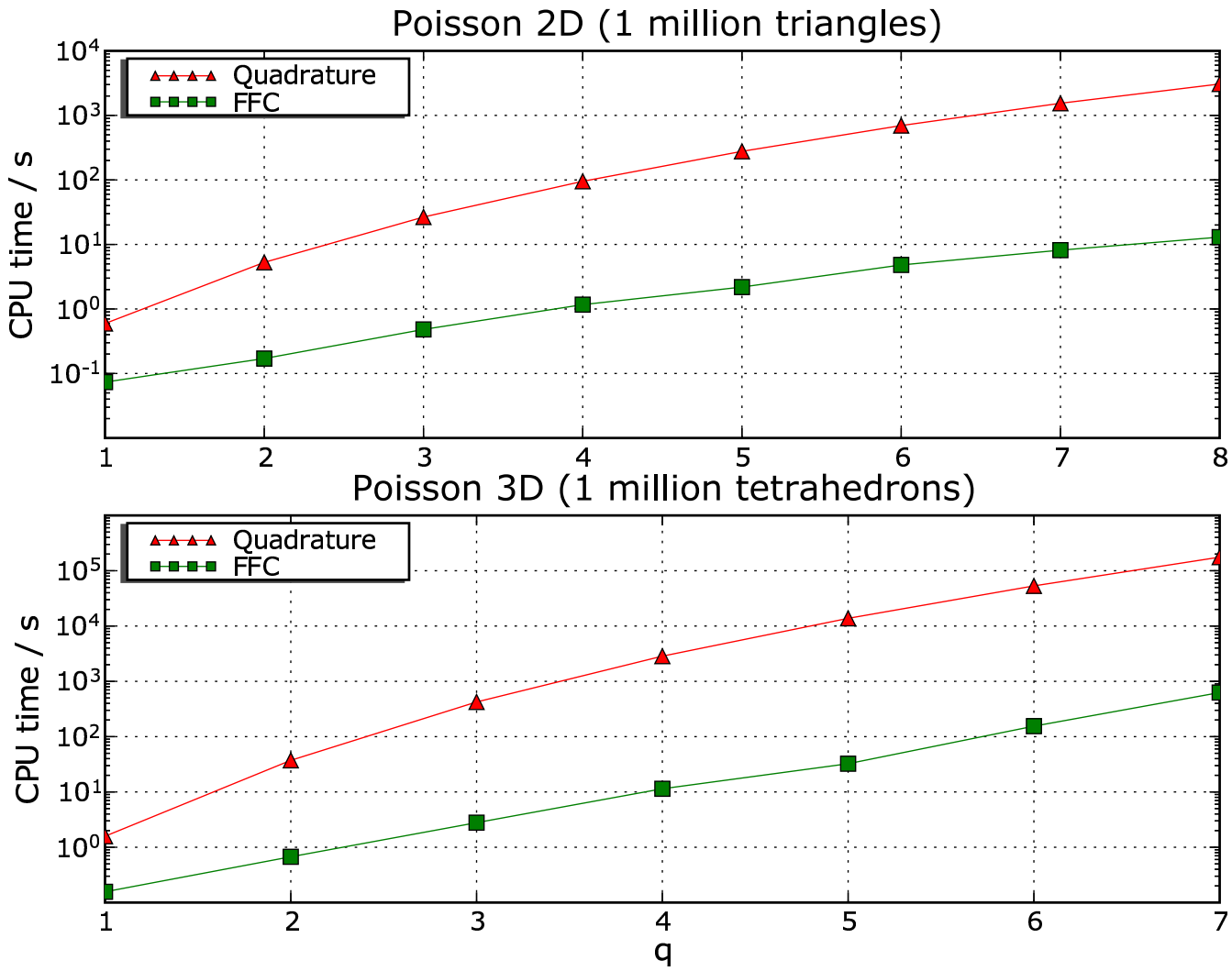
$$a(v,u) = \int_\Omega \nabla u \cdot \nabla v \, dx = \int_\Omega \sum_{i=1}^{d} \frac{\partial u}{\partial x_i} \frac{\partial v}{\partial x_i} \, dx$$

- FFC implementation:

```
v = BasisFunction(element)
u = BasisFunction(element)

a = u.dx(i)*v.dx(i)*dx
```

Results


Poisson 2D (1 million triangles)
Quadrature
FFC
CPU time / s
10^4
10^3
10^2
10^1
10^0
10^{-1}
1
2
3
4
5
6
7
8
Poisson 3D (1 million tetrahedrons)
Quadrature
FFC
CPU time / s
10^5
10^4
10^3
10^2
10^1
10^0
1
2
3
4
5
6
7
q

Test case 3: Navier–Stokes

- Mathematical notation:

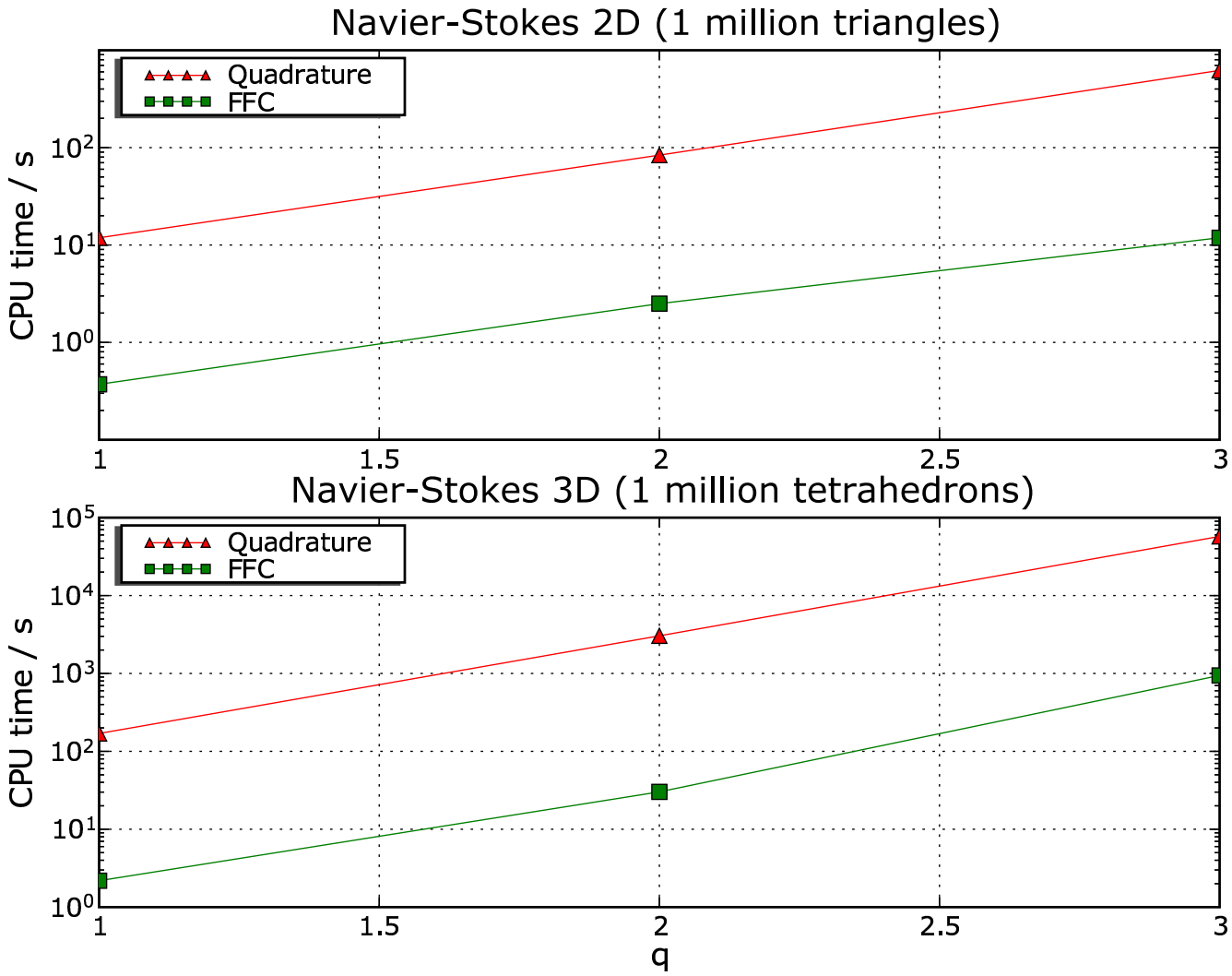
$$a(v,u) = \int_\Omega (w \cdot \nabla u) v \, dx = \int_\Omega \sum_{i=1}^{d} \sum_{j=1}^{d} w_j \frac{\partial u_i}{\partial x_j} v_i \, dx$$

- FFC implementation:

```
v = BasisFunction(element)
u = BasisFunction(element)
w = Function(element)

a = w[j]*u[i].dx(j)*v[i]*dx
```

Results


Navier-Stokes 2D (1 million triangles)
Quadrature
FFC
CPU time / s
10^2
10^1
10^0
1
1.5
2
2.5
3
Navier-Stokes 3D (1 million tetrahedrons)
Quadrature
FFC
CPU time / s
10^5
10^4
10^3
10^2
10^1
10^0
1
1.5
2
2.5
3
q

Test case 4: Linear elasticity

- Mathematical notation:

$$\begin{aligned} a(v,u) &= \int_\Omega \tfrac{1}{4}(\nabla u + (\nabla u)^\top) : (\nabla v + (\nabla v)^\top)\, dx \\ &= \int_\Omega \sum_{i=1}^{d} \sum_{j=1}^{d} \tfrac{1}{4}(\tfrac{\partial u_i}{\partial x_j} + \tfrac{\partial u_j}{\partial x_i})(\tfrac{\partial v_i}{\partial x_j} + \tfrac{\partial v_j}{\partial x_i})\, dx \end{aligned}$$

- FFC implementation:

```
v = BasisFunction(element)
u = BasisFunction(element)

a = 0.25*(u[i].dx(j) + u[j].dx(i)) * \
          (v[i].dx(j) + v[j].dx(i)) * dx
```

Future directions for **FEniCS**

- New parallel mesh component (05)
- Completely parallel assembly/solve (05)
- Implementation of non-standard elements (05/06): *Crouzeix–Raviart*, *Raviart–Thomas*, *Nedelec*, *Brezzi–Douglas–Marini*, *Brezzi–Douglas–Fortin–Marini*, *Arnold–Winther*, *Taylor–Hood*, ...
- Automatic generation of dual problems (06)
- Automatic generation of error estimates (06)
- Manuals, tutorials, mini-courses (05/06)

- Further information:

`http://www.fenics.org/`