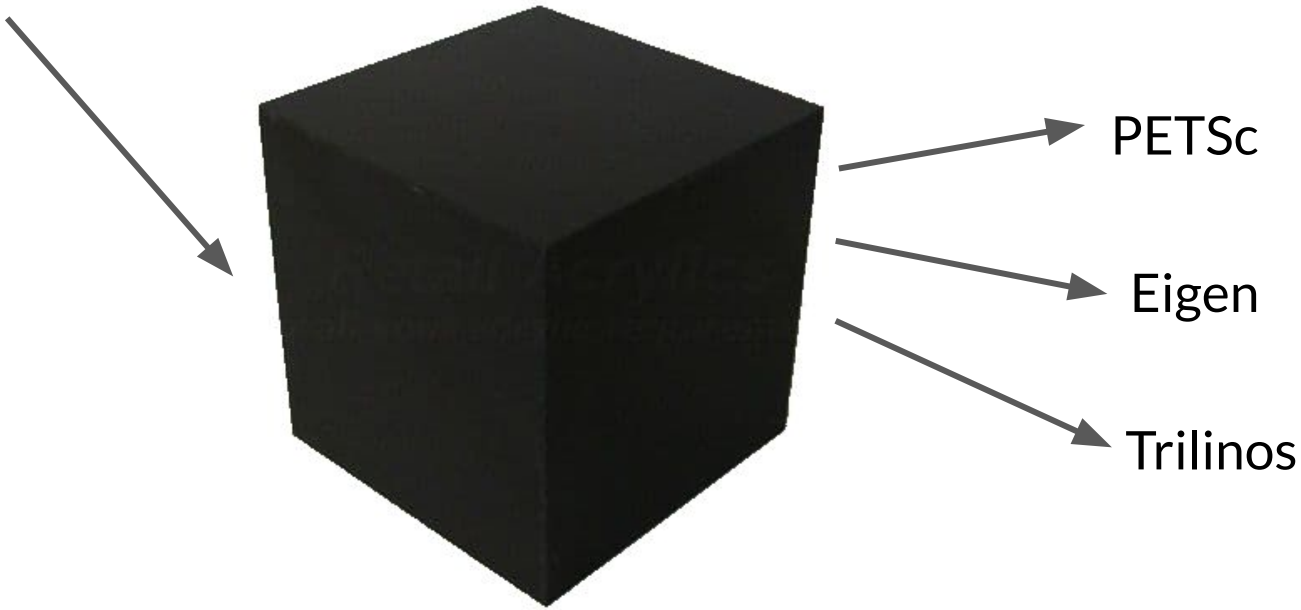
Turning FEniCS inside-out

Chris Richardson
chris@bpi.cam.ac.uk

DOLFIN 2017

```
GenericMatrix::set_local(...);
```

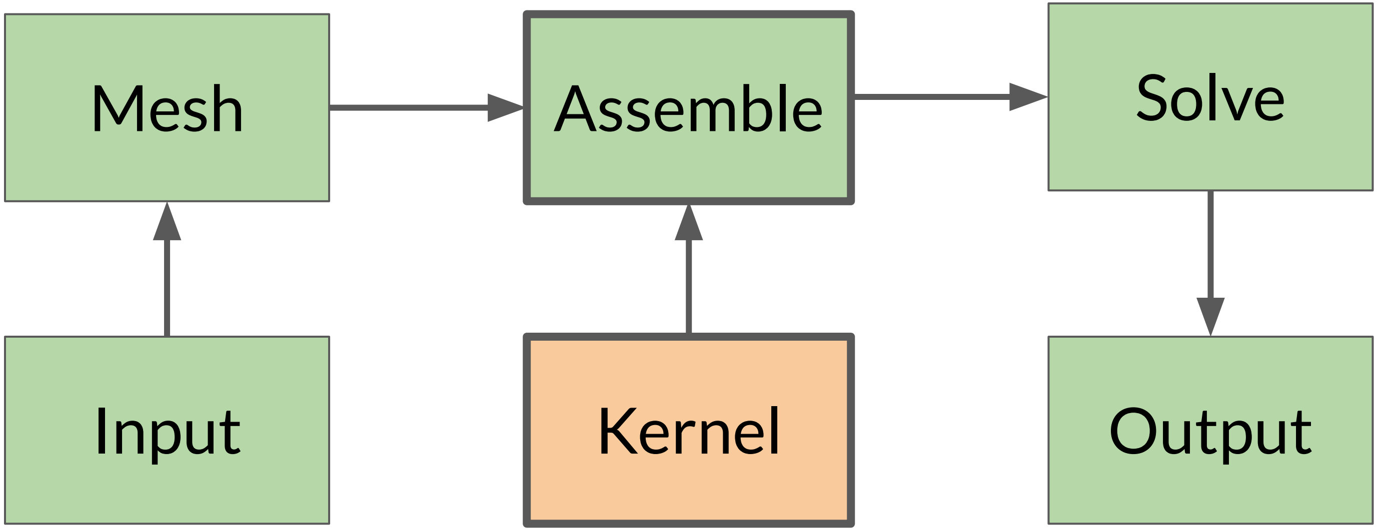


```
GenericTransport::move(src, dest);
```

FEniCS-X: A modular Finite Element code

"Keep it simple and give the user freedom to do things their own way"

C++ dolfinx library (with Python bindings)



Python: ufl, basix, ffcx

Matrix assembly plugins/callbacks

`dolfinx::fem::assemble_matrix(...)`

```
for cells in mesh:
  geom = geometry(cell)
  ...
  Ae = tabulate_tensor(geom, ...)
  global_mat_insert(Ae, ...)
```

`mat_insert()`

fem::Form

`tabulate_tensor()`

Turning things inside out: C++ lambda functions

```
auto mat_insert = [&A](int nrows, int* rows,
                       int ncols, int* cols, double *Ael)
{
  for (int i = 0; i < nrows; ++i)
    for (int j = 0; j < ncols; ++j)
      A[rows[i], cols[j]] += Ael[i*ncols + j];
}

dolfinx::fem::assemble_matrix(mat_insert, form, bcs);
```

1. Using other linear algebra backends: Trilinos

```
Tpetra::CrsMatrix<double> A;

auto mat_insert = [&A](int nrows, int* rows,
                       int ncols, int* cols, double *Ael)
{
  for (int i = 0; i < nrows; ++i)
  {
    ArrayView<const int> col_view(cols, ncols);
    ArrayView<double> col_view(Ael + i*ncols, nc);
    for (int j = 0; j < ncols; ++j)
      A.sumIntoLocalValues(rows[i], col_view, val_view);
  }
}
```

2. Lumping of mass matrix into diagonal

```
std::vector<double> diag;

auto vec_insert = [&diag](int nrows, int* rows,
                          int ncols, int* cols, double *Ael)
{
  for (int i = 0; i < nrows; ++i)
    for (int j = 0; j < ncols; ++j)
      diag[rows[i]] += Ael[i*ncols + j];
}

fem::assemble_matrix(vec_insert, form, bcs);
```

3. Action of a form (matrix-free)

```
auto mat_apply = [&uvec, &wvec](int nr, const int* rows,
              int nc, const int* cols, const double* Ae)
   {
     for (int i = 0; i < nr; ++i)
       for (int j = 0; j < nc; ++j)
         wvec[rows[i]] += Ae[i * nc + j] * uvec[cols[j]];
   };

fem::assemble_matrix(mat_apply, form, bcs);
```

4. Assemble diagonal, and action of L+U → Jacobi

```
std::vector<double> diag, w, u;

auto insert_diag = [&diag] (int nrows, int* rows,
                            int ncols, int* cols, double *Ael)
{
  for (int i = 0; i < nrows; ++i)
    diag[rows[i]] += Ael[i*ncols + i];
}

auto apply_lu = [&w, &u] (int nrows, int* rows,
                          int ncols, int* cols, double *Ael)
{
  for (int i = 0; i < nrows; ++i)
    for (int j = 0; j < ncols; ++j)
      if (j != i)
        w[rows[i]] += Ael[i*ncols + j] * u[cols[j]];
}
```

Python version? But don't do this without an adult present...

```
a = inner(grad(u), grad(v))*dx
a_form = fem.Form(a)
rdata = []
cdata = []
vdata = []

def mat_insert(rows, cols, vals):
    vdata.append(vals)
    rdata.append(numpy.repeat(rows, len(cols)))
    cdata.append(numpy.tile(cols, len(rows)))
    return 0

# Using Python callback is SLOW...
dolfinx.cpp.fem.assemble_matrix(mat_insert, a_form._cpp_object, [])
scipy.sparse.coo_matrix((vdata, (rdata, cdata)))
```

Summary

- FEniCS-X is more open to experimentation at the low level
- Plugin functionality via std::function in C++ for matrix insertion
    - FEniCS-X need know nothing about your LA backend
- Can also prototype in Python (but don't do this for a real application)
    - Better to use e.g. *`cppimport`* to wrap a snippet
- There are a number of things you can do with fem::assemble_matrix
- Mesh partitioning has some similar plugins with std::function