

Programming in OpenFOAM®. Building blocks

- In the directory `$WM_PROJECT_DIR/applications/test`, you will find the source code of several test cases that show the usage of most of the OpenFOAM® classes.
- We highly encourage you to take a look at these test cases and try to understand how to use the classes.
- We will use these basic test cases to understand the following base classes: tensors, fields, mesh, and basic discretization.
- For your convenience, we already copied the directory `$WM_PROJECT_DIR/applications/test` into the directory `$PTOFC/programming_playground/test`

Programming in OpenFOAM®. Building blocks

- During this session we will study the building blocks to write basic programs in OpenFOAM®:
 - First, we will start by taking a look at the algebra of tensors in OpenFOAM®.
 - Then, we will take a look at how to generate tensor fields from tensors.
 - Next, we will learn how to access mesh information.
 - Finally we will see how to discretize a model equation and solve the linear system of equations using OpenFOAM® classes and templates.
 - And of course, we are going to program a little bit in C++. But do not be afraid, after all this is not a C++ course.
- Remember, all OpenFOAM® components are implemented in library form for easy re-use.
- OpenFOAM® encourage code re-use. So basically we are going to take something that already exist and we are going to modify it to fix our needs.
- We like to call this method CPAC (copy-paste-adapt-compile).

Programming in OpenFOAM®. Building blocks

Basic tensor classes in OpenFOAM®

- OpenFOAM® represents scalars, vectors and matrices as tensor fields. A zero rank tensor is a scalar, a first rank tensor is a vector and a second rank tensor is a matrix.
- OpenFOAM® contains a C++ class library named **primitive** (`$FOAM_SRC/OpenFOAM/primitives/`). In this library, you will find the classes for the tensor mathematics.
- In the following table, we show the basic tensor classes available in OpenFOAM®, with their respective access functions.

Tensor Rank	Common name	Basic class	Access function
0	Scalar	<code>scalar</code>	
1	Vector	<code>vector</code>	<code>x()</code> , <code>y()</code> , <code>z()</code>
2	Tensor	<code>tensor</code>	<code>xx()</code> , <code>xy()</code> , <code>xz()</code> ...

Programming in OpenFOAM®. Building blocks

Basic tensor classes in OpenFOAM®

- In OpenFOAM®, the second rank tensor (or matrix)

$$\mathbf{T} = \begin{pmatrix} 1 & 2 & 3 \\ 4 & 5 & 6 \\ 7 & 8 & 9 \end{pmatrix}$$

can be declared in the following way

```
tensor T(1, 2, 3, 4, 5, 6, 7, 8, 9);
```


- We can access the component T_{13} or T_{xz} using the **xz ()** access function,

Programming in OpenFOAM®. Building blocks

Basic tensor classes in OpenFOAM®

- For instance, the following statement,

```
Info << "Txz = " << T.xz ( ) << endl;
```


$$\mathbf{T} = \begin{pmatrix} 1 & 2 & 3 \\ 4 & 5 & 6 \\ 7 & 8 & 9 \end{pmatrix}$$

- Will generate the following screen output,

```
$> Txz = 3
```

- Notice that to output information to the screen in OpenFOAM®, we use the function **Info** instead of the function **cout** (used in standard C++).
- The function **cout** will work fine, but it will give you problems when running in parallel.

Programming in OpenFOAM®. Building blocks

Algebraic tensor operations in OpenFOAM®

- Tensor operations operate on the entire tensor entity.
- OpenFOAM® syntax closely mimics the syntax used in written mathematics, using descriptive functions (e.g. `mag`) or symbolic operators (e.g. `+`).
- OpenFOAM® also follow the standard rules of linear algebra when working with tensors.
- Some of the algebraic tensor operations are listed in the following table (where **a** and **b** are vectors, *s* is a scalar, and **T** is a tensor).

Operation	Remarks	Mathematical description	OpenFOAM® description
Addition		$\mathbf{a} + \mathbf{b}$	<code>a + b</code>
Scalar multiplication		$s\mathbf{a}$	<code>s * a</code>
Outer product	rank a, b ≥ 1	\mathbf{ab}	<code>a * b</code>
Inner product	rank a, b ≥ 1	$\mathbf{a}\cdot\mathbf{b}$	<code>a & b</code>
Double inner product	rank a, b ≥ 2	$\mathbf{a}:\mathbf{b}$	<code>a && b</code>
Magnitude		$ \mathbf{a} $	<code>mag (a)</code>
Determinant		$\det \mathbf{T}$	<code>det (T)</code>

You can find a complete list of all operators in the programmer's guide

Programming in OpenFOAM®. Building blocks

Dimensional units in OpenFOAM®

- As we already know, OpenFOAM® is fully dimensional.
- Dimensional checking is implemented as a safeguard against implementing a meaningless operation.
- OpenFOAM® encourages the user to attach dimensional units to any tensor and it will perform dimension checking of any tensor operation.
- You can find the dimensional classes in the directory `$FOAM_SRC/OpenFOAM/dimensionedTypes/`
- The dimensions can be hardwired directly in the source code or can be defined in the input dictionaries.
- From this point on, we will be attaching dimensions to all the tensors.

Programming in OpenFOAM®. Building blocks

Dimensional units in OpenFOAM®

- Units are defined using the **dimensionSet** class tensor, with its units defined using the **dimensioned<Type>** template class, the **<Type>** being scalar, vector, tensor, etc. The **dimensioned<Type>** stores the variable name, the dimensions and the tensor values.
- For example, a tensor with dimensions is declare in the following way:

```
1  dimensionedTensor sigma
2  (
3    "sigma",
4    dimensionSet(1, -1, -2, 0, 0, 0, 0), ≡ σ =  $\begin{pmatrix} 10^6 & 0 & 0 \\ 0 & 10^6 & 0 \\ 0 & 0 & 10^6 \end{pmatrix}$ 
5    tensor(10e6,0,0,0,10e6,0,0,0,10e6)
6  );
```

- In line 1 we create the object **sigma**.
- In line 4, we use the class **dimensionSet** to attach units to the object **sigma**.
- In line 5, we set the input values of the tensor **sigma**.

Programming in OpenFOAM®. Building blocks

Units correspondence in dimensionSet

- The units of the class **dimensionSet** are defined as follows

dimensionSet (kg, m, s, K, mol, A, cd)

- Therefore, the tensor **sigma**,

```
1 dimensionedTensor sigma
2 (
3   "sigma",
4   dimensionSet(1, -1, -2, 0, 0, 0, 0), ≡ σ =  $\begin{pmatrix} 10^6 & 0 & 0 \\ 0 & 10^6 & 0 \\ 0 & 0 & 10^6 \end{pmatrix}$ 
5   tensor(10e6,0,0,0,10e6,0,0,0,10e6)
6 );
```

- Has pressure units or $kg\ m^{-1}\ s^{-2}$

Programming in OpenFOAM®. Building blocks

Dimensional units examples

- To attach dimensions to any tensor, you need to access dimensional units class.
- To do so, just add the header file *dimensionedTensor.H* to your program.

```
#include "dimensionedTensor.H"
...
...
...
dimensionedTensor sigma
(
    "sigma",
    dimensionSet(1, -1, -2, 0, 0, 0, 0),
    tensor(1e6,0,0,0,1e6,0,0,0,1e6)
);
Info<< "Sigma: " << sigma << endl;
...
...
...
```

- The output of the previous program should look like this:

```
sigma sigma [1 -1 -2 0 0 0 0] (1e+06 0 0 0 1e+06 0 0 0 1e+06)
```

Programming in OpenFOAM®. Building blocks

Dimensional units examples

- As for base tensors, you can access the information of dimensioned tensors.
- For example, to access the name, dimensions, and values of a dimensioned tensor, you can proceed as follows:

```
Info << "Sigma name: " << sigma.name ( ) << endl;  
Info << "Sigma dimensions: " << sigma.dimensions ( ) << endl;  
Info << "Sigma value: " << sigma.value ( ) << endl;
```

- To extract a value of a dimensioned tensor, you can proceed as follows:

```
Info<< "Sigma yy (22) value: " << sigma.value().yy() << endl;
```

- Note that the **value()** member function first converts the expression to a tensor, which has a **yy()** member function.
- The **dimensionedTensor** class does not have a **yy()** member function, so it is not possible to directly get its value by using **sigma.yy()**.

Programming in OpenFOAM®. Building blocks

OpenFOAM® lists and fields

- OpenFOAM® frequently needs to store sets of data and perform mathematical operations.
- OpenFOAM® provides an array template class **List<Type>**, making it possible to create a list of any object of class **Type** that inherits the functions of the **Type**. For example a **List** of **vector** is **List<vector>**.
- Lists of the tensor classes are defined in OpenFOAM® by the template class **Field<Type>**.
- For better code legibility, all instances of **Field<Type>**, e.g. **Field<vector>**, are renamed using **typedef** declarations as **scalarField**, **vectorField**, **tensorField**, **symmTensorField**, **tensorThirdField** and **symmTensorThirdField**.
- You can find the field classes in the directory `$FOAM_SRC/OpenFOAM/fields/Fields`.
- Algebraic operations can be performed between fields, subject to obvious restrictions such as the fields having the same number of elements.
- OpenFOAM® also supports operations between a field and a zero rank tensor, e.g. all values of a **Field U** can be multiplied by the **scalar 2** by simple coding the following line, **U = 2.0 * U**.

Programming in OpenFOAM®. Building blocks

Construction of a tensor field in OpenFOAM®

- To create fields, you need to access the tensor class.
- To do so, just add the header file *tensorField.H* to your program. This class inherit all the tensor algebra.

```
#include "tensorField.H"
...
...
...
tensorField tf1(2, tensor::one);
Info<< "tf1: " << tf1 << endl;
tf1[0] = tensor(1, 2, 3, 4, 5, 6, 7, 8, 9);
Info<< "tf1: " << tf1 << endl;
Info<< "2.0*tf1: " << 2.0*tf1 << endl;
...
...
...
```

- In this example, we created a list of two tensor fields (**tf1**), and both tensors are initialized to one.
- We can access components on the list using the access operator [].

Programming in OpenFOAM®. Building blocks

Example of use of tensor and field classes

- In the directory `$PTOFC/programming_playground/my_tensor` you will find a tensor class example.
- The original example is located in the directory `$PTOFC/programming_playground/test/tensor`. Feel free to compare the files to spot the differences.
- Before compiling the file, let us recall how applications are structure,

```
working_directory/  
├─ applicationName.C  
├─ header-files.H  
└─ Make  
    ├─ files  
    └─ options
```

- `applicationName.C`: is the actual source code of the application.
- `header_files.H`: header files required to compile the application.

Programming in OpenFOAM®. Building blocks

Example of use of tensor and field classes

- Before compiling the file, let us recall how applications are structure.

```
working_directory/  
├─ applicationName.C  
├─ header-files.H  
└─ Make  
    ├─ files  
    └─ options
```

- The **Make** directory contains compilation instructions.
 - *files*: names all the source files (.C), it specifies the name of the new application and the location of the output file.
 - *options*: specifies directories to search for include files and libraries to link the solver against.
- At the end of the file *files*, you will find the following line of code,
EXE = \$(FOAM_USER_APPBIN)/my_Test-tensor
- This is telling the compiler to name your application **my_Test-tensor** and to copy the executable in the directory **\$FOAM_USER_APPBIN**.
- To avoid conflicts between applications, always remember to give a proper name and a location to your programs and libraries.

Programming in OpenFOAM®. Building blocks

Example of use of tensor and field classes

- Let us now compile the tensor class example. Type in the terminal:

1. `$> cd $PTOFC/programming_playground/my_tensor`
2. `$> wmake`
3. `$> my_Test-tensor`

- In step 2, we used `wmake` (distributed with OpenFOAM®) to compile the source code.
- The name of the executable will be **my_Test-tensor** and it will be located in the directory `$FOAM_USER_APPBIN` (as specified in the file *Make/files*)
- At this point, take a look at the output and study the file *Test-tensor.C*. Try to understand what we have done.
- After all, is not that difficult. Right?

Programming in OpenFOAM®. Building blocks

- At this point, we are a little bit familiar with tensor, fields, and lists in OpenFOAM®.
- They are the base to building applications in OpenFOAM®.
- Let us now take a look at the whole solution process:
 - Creation of the tensors.
 - Mesh assembly.
 - Fields creation.
 - Equation discretization.
- All by using OpenFOAM® classes and template classes

Programming in OpenFOAM®. Building blocks

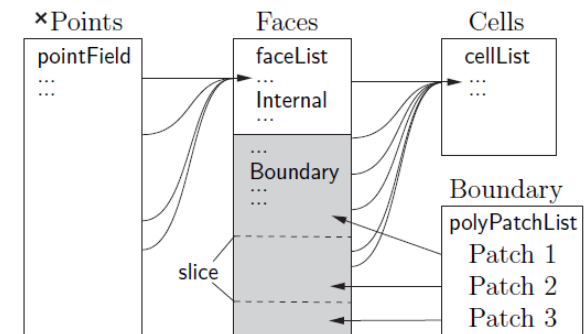
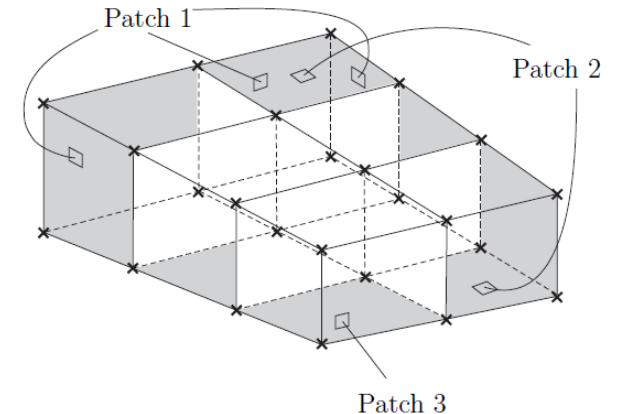
Discretization of a tensor field in OpenFOAM®

- The discretization is done using the FVM (Finite Volume Method).
- The cells are contiguous, *i.e.*, they do not overlap and completely fill the domain.
- Dependent variables and other properties are stored at the cell centroid.
- No limitations on the number of faces bounding each cell.
- No restriction on the alignment of each face.
- The mesh class **polyMesh** is used to construct the polyhedral mesh using the minimum information required.
- You can find the **polyMesh** classes in the directory `$FOAM_SRC/OpenFOAM/meshes`
- The **fvMesh** class extends the **polyMesh** class to include additional data needed for the FVM discretization.
- You can find the **fvMesh** classes in the directory `$FOAM_SRC/src/finiteVolume/fvMesh`

Programming in OpenFOAM®. Building blocks

Discretization of a tensor field in OpenFOAM®

- The template class **geometricField** relates a tensor field to a **fvMesh**.
- Using typedef declarations **geometricField** is renamed to **volField** (cell center), **surfaceField** (cell faces), and **pointField** (cell vertices).
- You can find the **geometricField** classes in the directory `$FOAM_SRC/OpenFOAM/fields/GeometricFields`.
- The template class **geometricField** stores internal fields, boundary fields, mesh information, dimensions, old values and previous iteration values.
- A **geometricField** inherits all the tensor algebra of its corresponding field, has dimension checking, and can be subjected to specific discretization procedures.
- Let us now access the mesh information of a simple case.



Programming in OpenFOAM®. Building blocks

Data stored in the `fvMesh` class

Class	Description	Symbol	Access function
<code>volScalarField</code>	Cell volumes	V	<code>v()</code>
<code>surfaceVectorField</code>	Face area vector	\mathbf{S}_f	<code>Sf()</code>
<code>surfaceScalarField</code>	Face area magnitude	$ \mathbf{S}_f $	<code>magSf()</code>
<code>volVectorField</code>	Cell centres	\mathbf{C}	<code>C()</code>
<code>surfaceVectorField</code>	Face centres	\mathbf{C}_f	<code>Cf()</code>
<code>surfaceScalarField</code>	Face fluxes	ϕ_g	<code>Phi()</code>

Programming in OpenFOAM®. Building blocks

Accessing fields defined in a mesh

- To access fields defined at cell centers of the mesh you need to use the class **volField**.
- The class **volField** can be accessed by adding the header *volFields.H* to your program.

```
volScalarField p  
(  
    IOobject  
    (  
        "p",  
        runtime.timeName(),  
        mesh,  
        IOobject::MUST_READ,  
        IOobject::AUTO_WRITE  
    ),  
    mesh  
);  
Info<< p << endl;  
Info<< p.boundaryField()[0] << endl;
```

Create scalar volField p

Assign and initialization of scalar volField to the mesh

Output some information

Programming in OpenFOAM®. Building blocks

Accessing fields using for loops

- To access fields using **for** loops, we can use OpenFOAM® macro **forAll**, as follows,

```
forAll(mesh.boundaryMesh(), patchI)
Info << "Patch " << patchI << ": " << mesh.boundary()[patchI].name() << " with "
    << mesh.boundary()[patchI].Cf().size() << " faces. Starts at total face "
    << mesh.boundary()[patchI].start() << endl;
```

↑ Outputs size of patch (number of faces)

↓ Outputs name of patch

- In the previous statement **mesh.boundaryMesh()** is the size of the loop, and **patchI** is the iterator. The iterator always starts from zero.
- The **forAll** loop is equivalent to the standard **for** loop in C++.

```
for (int i = 0; i < mesh.boundaryMesh().size(); i++)
Info << "Patch " << i << ": " << mesh.boundary()[i].name() << " with "
    << mesh.boundary()[i].Cf().size() << " faces. Starts at total face "
    << mesh.boundary()[i].start() << endl;
```

↑ Outputs starting face of patch

- Notice that we used as iterator **i** instead of **patchI**, this does not make any difference.

Programming in OpenFOAM®. Building blocks

Equation discretization in OpenFOAM®

- At this stage, OpenFOAM® converts the PDEs into a set of linear algebraic equations, $\mathbf{A} \mathbf{x} = \mathbf{b}$, where \mathbf{x} and \mathbf{b} are **volFields** (**geometricField**).
- \mathbf{A} is a **fvMatrix**, which is created by the discretization of a **geometricField** and inherits the algebra of its corresponding field, and it supports many of the standard algebraic matrix operations.
- The **fvm** (**finiteVolumeMethod**) and **fvc** (**finiteVolumeCalculus**) classes contain static functions for the differential operators, and discretize any **geometricField**.
- **fvm** returns a **fvMatrix**, and **fvc** returns a **geometricField**.
- In the directories `$FOAM_SRC/finiteVolume/finiteVolume/fvc` and `$FOAM_SRC/finiteVolume/finiteVolume/fvm` you will find the respective classes.
- Remember, the PDEs or ODEs we want to solve involve derivatives of tensor fields with respect to time and space. What we are doing at this point, is applying the finite volume classes to the fields, and assembling a linear system.

Programming in OpenFOAM®. Building blocks

Discretization of the basic PDE terms in OpenFOAM®

The list is not complete

Term description	Mathematical expression	fvm:: fvc::
Laplacian	$\nabla^2 \phi$, $\nabla \cdot \Gamma \nabla \phi$	laplacian(phi) laplacian(Gamma, phi)
Time derivative	$\frac{\partial \phi}{\partial t}$, $\frac{\partial \rho \phi}{\partial t}$	ddt(phi) ddt(rho, phi)
Convection	$\nabla \cdot (\psi)$, $\nabla \cdot (\psi \phi)$	div(psi, scheme) div(psi, phi)
Source	$\rho \phi$	Sp(rho, phi) SuSp(rho, phi)

ϕ vol<type>Field

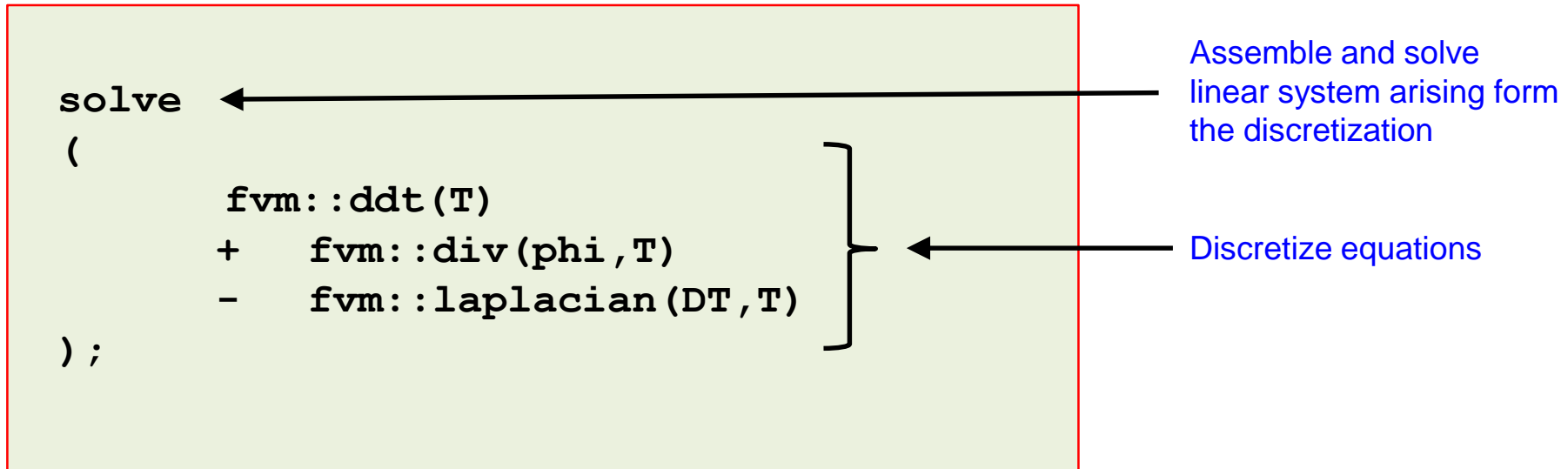
ρ scalar, volScalarField

ψ surfaceScalarField

Programming in OpenFOAM®. Building blocks

Discretization of the basic PDE terms in OpenFOAM®

- To discretize the fields in a valid mesh, we need to access the finite volume class. This class can be accessed by adding the header `fvCFD.H` to your program.
- To discretize the scalar transport equation in a mesh, we can proceed as follows,

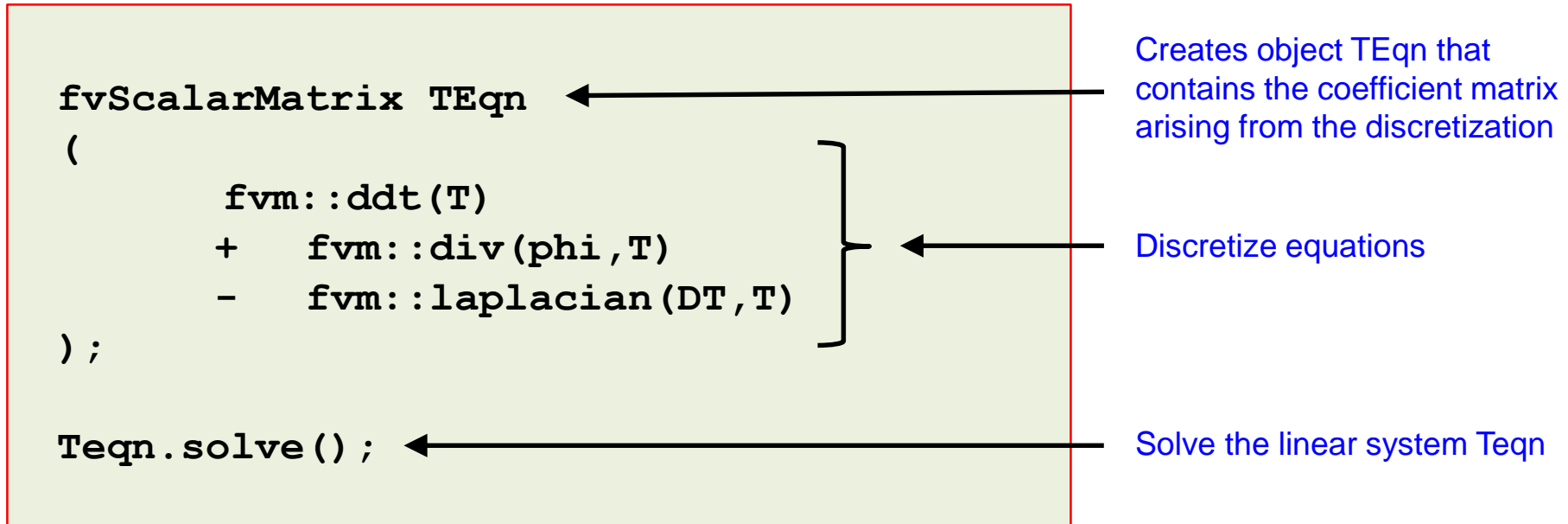


- Remember, you will need to first create the mesh, and initialize the variables and constants. That is, all the previous steps.
- Finally, everything we have done so far inherits all parallel directives. There is no need for specific parallel programming.

Programming in OpenFOAM®. Building blocks

Discretization of the basic PDE terms in OpenFOAM®

- The previous discretization is equivalent to,



- Here, **fvScalarMatrix** contains the matrix derived from the discretization of the model equation.
- fvScalarMatrix** is used for scalar fields and **fvVectorMatrix** is used for vector fields.
- This syntax is more general, since it allows the easy addition of terms to the model equations.

Programming in OpenFOAM®. Building blocks

Discretization of the basic PDE terms in OpenFOAM®

- At this point, OpenFOAM® assembles and solves the following linear system,

$$\begin{bmatrix} a_{11} & a_{12} & & & & \\ a_{21} & a_{22} & a_{23} & & & \\ & \ddots & \ddots & \ddots & & \\ \ddots & & \ddots & \ddots & \ddots & \\ & a_S & & a_W & a_P & a_E & a_N \\ & & \ddots & \ddots & \ddots & \ddots & \\ & & & \ddots & \ddots & \ddots & \ddots \\ & & & & \ddots & \ddots & a_{PP} \end{bmatrix} \times \begin{bmatrix} x_S \\ x_W \\ x_P \\ x_E \\ x_N \end{bmatrix} = \begin{bmatrix} b_S \\ b_W \\ b_P \\ b_E \\ b_N \end{bmatrix}$$

The diagram illustrates the linear system $Ax = b$. The matrix A is sparse and square, with non-zero entries $a_{11}, a_{12}, a_{21}, a_{22}, a_{23}, \dots, a_S, a_W, a_P, a_E, a_N, \dots, a_{PP}$. The vector x contains unknown quantities x_S, x_W, x_P, x_E, x_N . The vector b contains boundary conditions and source terms b_S, b_W, b_P, b_E, b_N . Arrows point from the matrix, vector, and vector to their respective labels.

↓

Coefficient Matrix (sparse, square)
The coefficients depend on geometrical quantities,
fluid properties and non linear equations

↓

Unknow quantity

↘

Boundary conditions
and source terms

Programming in OpenFOAM®. Building blocks

Example of use of tensor and field classes

- Let us study a **fvMesh** example. First let us compile the program `my_Test-mesh`. Type in the terminal,
 1. `$> cd $PTOFC/programming_playground/my_mesh/`
 2. `$> wmake`
- To access the mesh information, we need to use this program in a valid mesh.
 1. `$> cd $PTOFC/programming_playground/my_mesh/cavity`
 2. `$> blockMesh`
 3. `$> my_Test-mesh`
- At this point, take a look at the output and study the file `Test-mesh.C`. Try to understand what we have done.
- FYI, the original example is located in the directory `$PTOFC/programming_playground/test/mesh`.

Programming in OpenFOAM®. Building blocks

A few OpenFOAM® programming references

- You can access the API documentation in the following link, <https://cpp.openfoam.org/v5/>
- You can access the coding style guide in the following link, <https://openfoam.org/dev/coding-style-guide/>
- You can report programming issues in the following link, <https://bugs.openfoam.org/rules.php>
- You can access openfoamwiki coding guide in the following link, http://openfoamwiki.net/index.php/OpenFOAM_guide
- You can access the user guide in the following link, <https://cfd.direct/openfoam/user-guide/>
- You can read the OpenFOAM® Programmer's guide in the following link (it seems that this guide is not supported anymore), <http://foam.sourceforge.net/docs/Guides-a4/ProgrammersGuide.pdf>

A few good C++ references

- **The C++ Programming Language.** B. Stroustrup. 2013, Addison-Wesley.
- **The C++ Standard Library.** N. Josuttis. 2012, Addison-Wesley.
- **C++ for Engineers and Scientists.** G. J. Bronson. 2012, Cengage Learning.
- **Sams Teach Yourself C++ in One Hour a Day.** J. Liberty, B. Jones. 2004, Sams Publishing.
- **C++ Primer.** S. Lippman, J. Lajoie, B. Moo. 2012, Addison-Wesley.
- <http://www.cplusplus.com/>
- <http://www.learncpp.com/>
- <http://www.cprogramming.com/>
- <http://www.tutorialspoint.com/cplusplus/>
- <http://stackoverflow.com/>