Master in High Performance Computing

Continuum mechanics and fluid-structure interaction problems: mathematical modelling and numerical approximation

Course Presentation

Luca Heltai <luca.heltai@sissa.it>

International School for Advanced Studies (www.sissa.it)
Mathematical Analysis, Modeling, and Applications (math.sissa.it)
Master in High Performance Computing (www.mhpc.it)
SISSA mathLab (mathlab.sissa.it)
King Abdullah University of Science and Technology (kaust.edu.sa)

جامعة الملك عبدالله
للعلوم والتقنية
King Abdullah University of Science and Technology

The Abdus Salam International Centre for Theoretical Physics

SISSA

Luca Heltai: minimal BIO

- Visiting professor @ KAUST
- Director of the **Master in High Performance Computing**
- Associate Professor of **Numerical Analysis** @ SISSA
- **Math PhD** @ University of Pavia & Courant Institute, NYU
- BsC in **Electronic Engineering** @ University of Pavia

Research Interests

- Finite Element Methods
- Boundary Element Methods
- Non-matching Discretisation Methods
- Error analysis
- High Performance Computing
- Open Source Software Development (www.dealii.org)
- Machine Learning and Artificial Intelligence

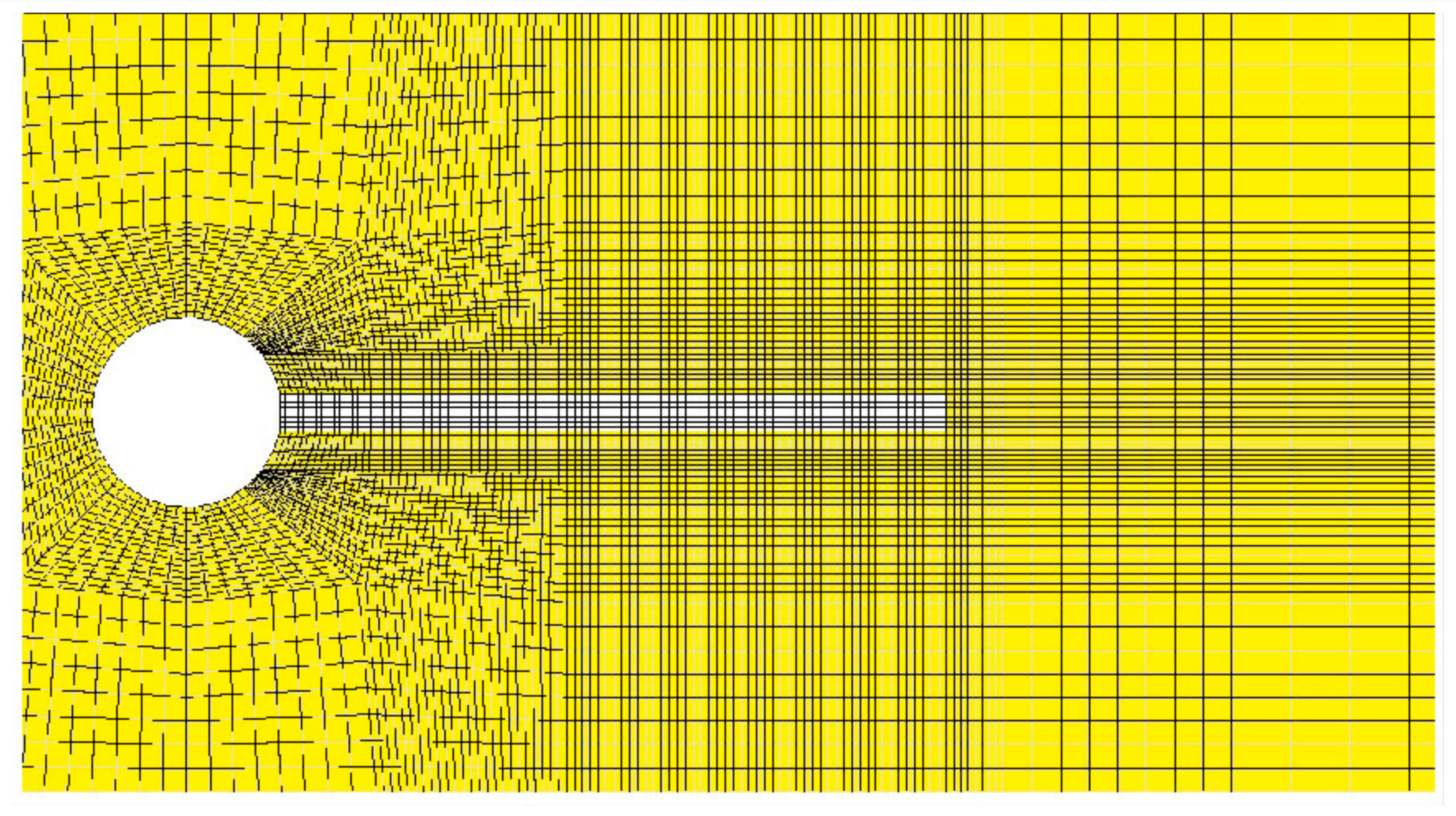
Fields of application

- Fluid structure interaction problems
  - Microswimmers
  - Naval hydro-dynamics
  - Brain bio-mechanics
- Micro and nano electronic devices
  - "quantum reinforced" continuum models
  - non-matching discretisation of defects and interfaces

Course content: three parallel paths

- basics of continuum mechanics
  - kinematics, dynamics, conservation equations, and PDEs
- mathematical modelling of FSI problems
  - Lagrangian, Eulerian, arbitrary Lagrangian Eulerian, non-matching, distributed Lagrange multipliers
- numerical implementations based on the finite element method
  - git, docker, visual studio code, and deal.II

Course objectives



In collaboration with D.Boffi, L.Gastaldi, F.Costanzo

In collaboration with Michal Wichrowski

Course objectives

- What you will learn:
  - Basics of continuum mechanics
  - Basic theory of Finite Elements
  - Fluid structure interaction
  - State-of-the-art implementation of FEM codes for FSI/continuum mechanics
  - How to use a modern C++ IDE, to build and debug your codes
  - How to use a large FEM library to solve complex PDE problems
  - How to properly document your code using Doxygen
  - How to use a proper Git workflow to develop your applications
  - How to leverage GitHub actions, google tests, and docker images to test and deploy your application
  - How hybrid parallelisation (threads + MPI + GPU) works in real life FEM applications

Outcome of the course — Theory

- You will learn how to model and study coupled problems of continuum mechanics, i.e.,
  - solid mechanics
  - fluid dynamics
  - everything in between

Outcome of the course — Practice

- You will learn how to use and develop FEM applications based on deal.II which:
  - Solve coupled, geometrically non-linear, physically non-linear, time dependent partial differential equations, on adaptively refined grids, in parallel
  - Uses modern version control tools (on GitHub)
  - Is tested automatically (through GitHub actions) every time you push a commit, or open a pull request
  - Is documented using Doxygen, and its web page is updated and deployed automatically every time you merge to master a new branch

Prerequisites

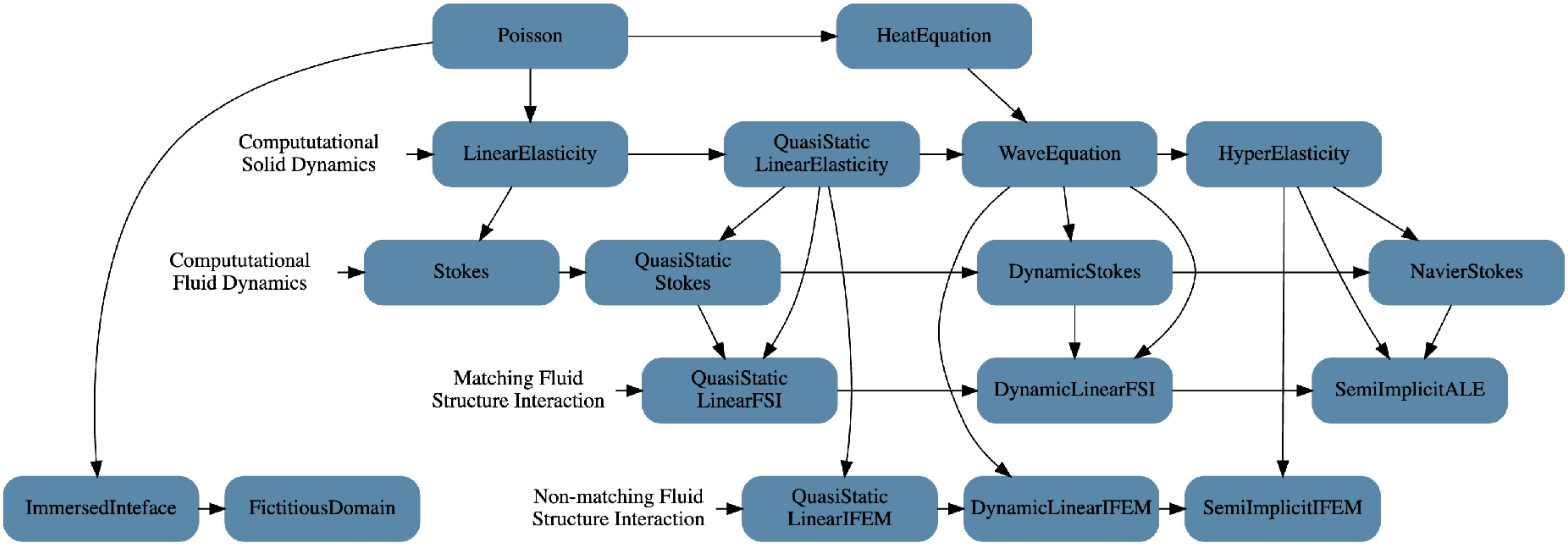
- Theory:
  - Some knowledge of Sobolev Spaces
    - Linear operators, Banach and Hilbert spaces, duality, etc.
  - Some knowledge on Numerical Analysis and Finite Elements
    - Quadrature, interpolation, Taylor expansions, Lagrangian Finite Elements, etc.

- Practice:
  - Some knowledge of C/C++
  - To run out of the box:
    - a machine with Visual Studio Code installed
    - Docker
    - A GitHub account
  - Do it yourself strategy:
    - deal.II with all its dependencies
    -

More Info

- Course pages:
  - Official Github course page (the most up to date information is found here): https://github.com/luca-heltai/fsi-suite
    - Course slides
    - Programs
  - Course recordings: https://bit.ly/2NY46LZ
  - Email:
    prof. Luca Heltai <luca.heltai@sissa.it>

Plan of the course (don't panic!)



Why deal.II (or any other Finite Element library)

- The numerical solution of partial differential equations is an immensely vast field!
- It requires us to know about:
  - Partial differential equations
  - Methods for discretizations, solvers, preconditioners
  - Programming
  - Adequate tools
  - **This course will cover all of this to some degree!**

Numerics of PDEs

There are 3 standard tools for the numerical solution of PDEs:

Finite element method (FEM)

Finite volume method (FVM)

Finite difference method (FDM)

Common features:

Split the domain into small volumes (cells)

Define balance relations on each cell

Obtain and solve very large (non-)linear systems

Problems:

Every code has to implement these steps

There is only so much time in a day

There is only so much expertise anyone can have

Numerics of PDEs

There are 3 standard tools for the numerical solution of PDEs:

Finite element method (FEM)

Finite volume method (FVM)

Finite difference method (FDM)

Common features:

Split the domain into small volumes (cells)

Define balance relations on each cell

Obtain and solve very large (non-)linear systems

In addition:

We don't just want a simple algorithm

We want state-of-the-art methods for everything

Numerics of PDEs

Examples of what we would like to have:

Adaptive meshes

Realistic, complex geometries

Quadratic or even higher order elements

Multigrid solvers

Scalability to 1000s of processors

Efficient use of current hardware

Graphical output suitable for high quality rendering

Q: How can we make all of this happen in a single code?

The hard reality

- Most research software today:
  - **Written by graduate students**
    - without a good overview of existing software
    - with little software experience
    - with little incentive to write high quality code
  - **Maintained by postdocs**
    - with little time
    - who need to consider the software primarily as a tool to publish papers
  - **Advised by faculty**
    - with no time
    - oftentimes also with little software experience

How we develop Software

Q: How can we make all of this happen in a single code?

Not a question of feasibility but of how we develop software:
Is every student developing their own software?
Or are we re-using what others have done?

Do we insist on implementing everything from scratch?
Or do we build our software on existing libraries?

How we develop Software

Q: How can we make all of this happen in a single code?

Not a question of feasibility but of how we develop software:
Is every student developing their own software?
Or are we re-using what others have done?

Do we insist on implementing everything from scratch?
Or do we build our software on existing libraries?

There has been a major shift on how we approach the second question in scientific computing over the past 10-15 years!

The secret to good scientific software is (re)using existing libraries!

Existing Software

There is excellent software for almost every purpose!

Basic linear algebra (dense vectors, matrices):
BLAS
LAPACK

Parallel linear algebra (vectors, sparse matrices, solvers):
PETSc
Trilinos

Meshes, finite elements, etc:
deal.II – the topic of this class
…

Visualization, dealing with parameter files, ...

Our experience

It is realistic for a student developing numerical methods using external libraries to have a code at the end of a PhD time that:

- Works in 2d and 3d
- On complex geometries
- Uses higher order finite element methods
- Uses multigrid solvers or preconditioners
- Solves a nonlinear, time dependent problem

Doing this from scratch would take 10+ years.

Common arguments…

Arguments against using other people's packages:

I would need to learn a new piece of software, how it works, its conventions. I would have to find my way around its documentation. Etc.

I think I'll be faster writing the code I want myself!

Common arguments…

Arguments against using other people's packages:

I would need to learn a new piece of software, how it works, its conventions. I would have to find my way around its documentation. Etc.

I think I'll be faster writing the code I want myself!

Answers:

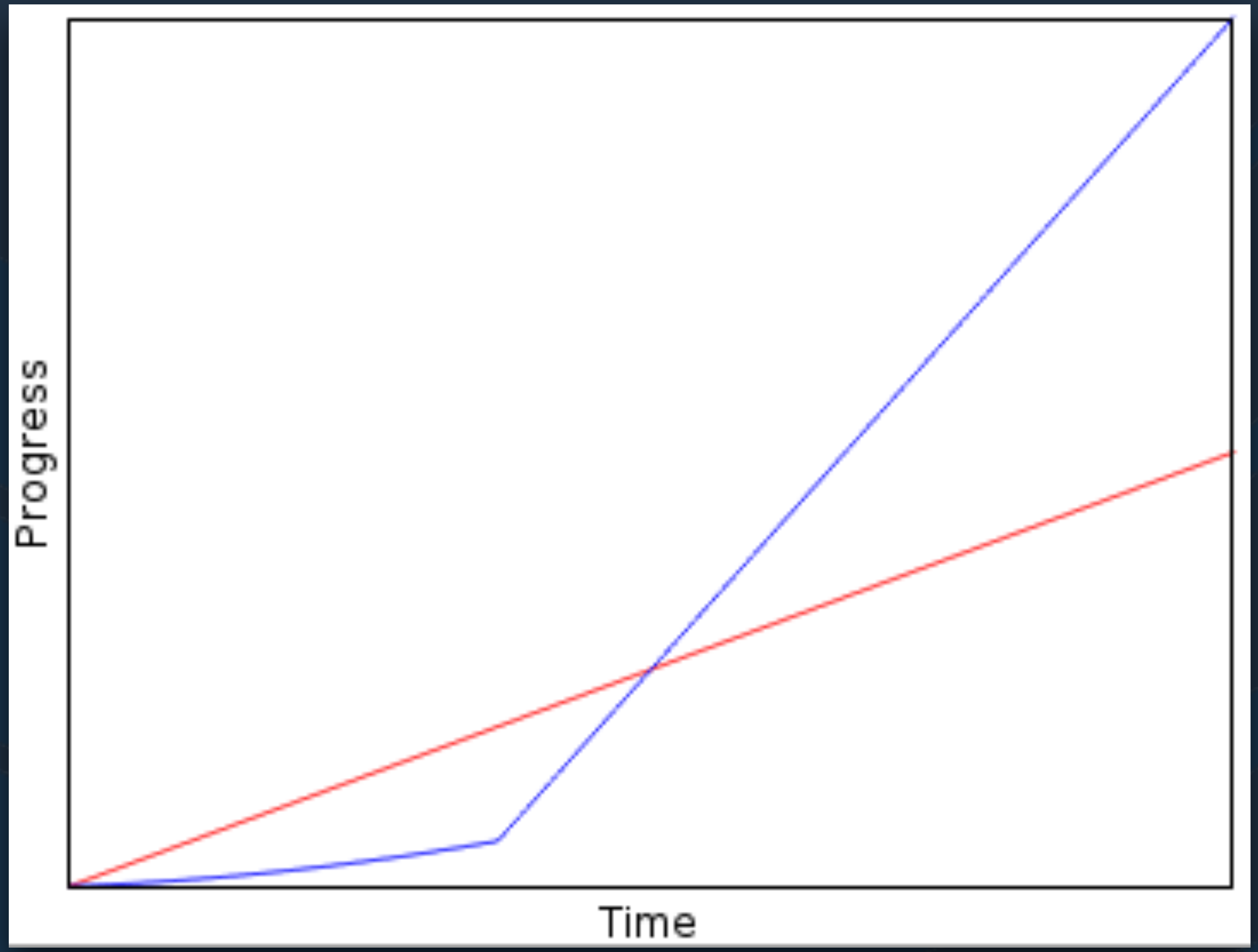
The first part is true.

The second is not!

You get to use a lot of functionality you could never in a lifetime implement yourself.

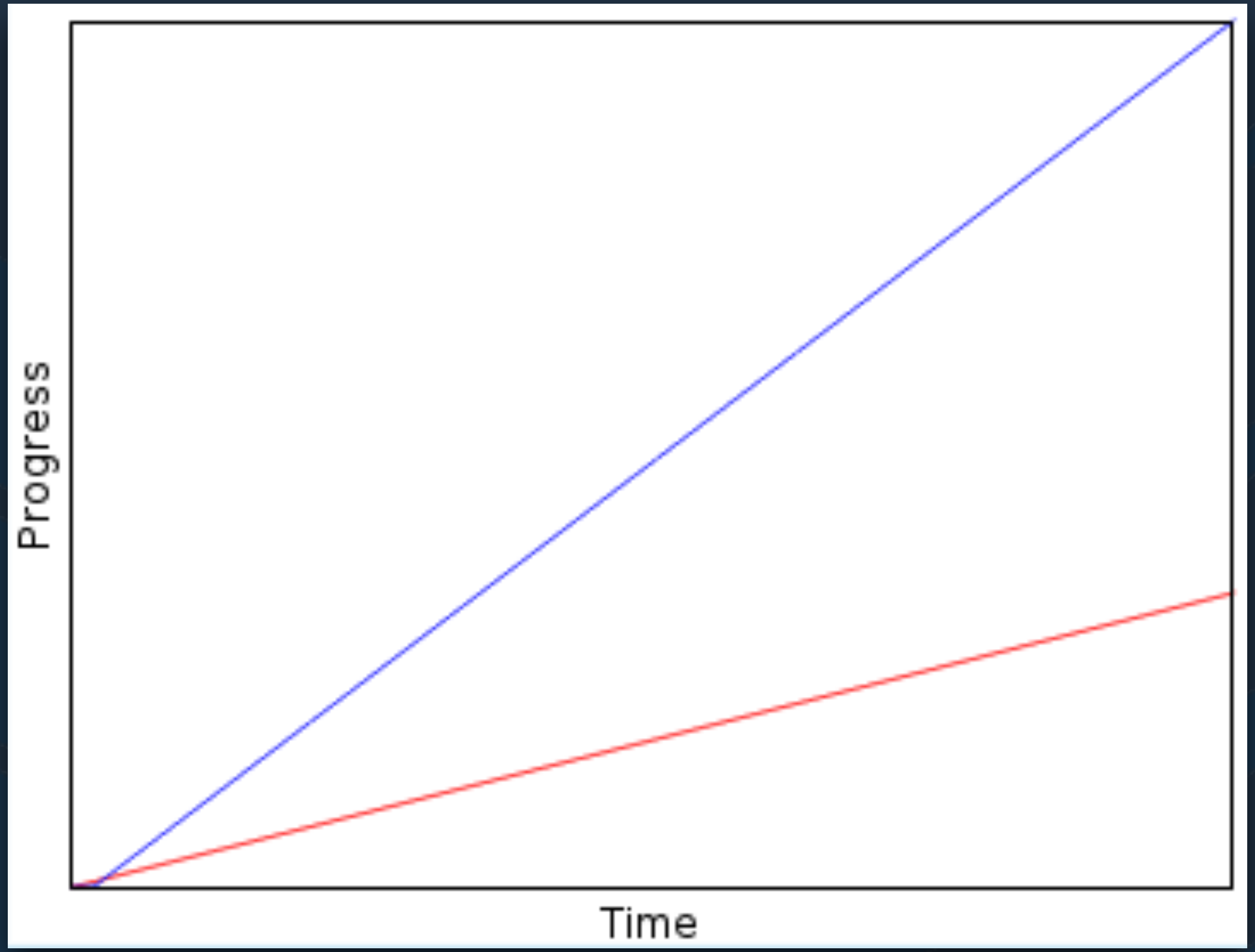
Think of how we use Matlab today!

I’m faster!



Blue: use external libraries
Red: do it yourself

The real picture…



Blue: use external libraries
Red: do it yourself

جامعة الملك عبدالله للعلوم والتقنية
King Abdullah University of Science and Technology
The Abdus Salam International Centre for Theoretical Physics
SISSA

Common Arguments…

Arguments against using other people's packages:

I want my students to actually understand the methods they are doing. So I let them code things from scratch!

Common Arguments…

Arguments against using other people's packages:

I want my students to actually understand the methods they are doing. So I let them code things from scratch!

Answers:

Yes, there is value to that.

But: if you know quadrature in 2d, why implement it again in 3d?

So let them write a toy code and throw it away after 3 months and do it right based on existing software.

Common Arguments…

Arguments against using other people's packages:

How do I know that that software I'm supposed to use doesn't have bugs? How can I *trust* other people's software?

With my own software, at least I know that I don't have bugs!

Answer 1:

You can't be serious to think that your own software has no bugs!

Common Arguments...

Arguments against using other people's packages:

How do I know that that software I'm supposed to use doesn't have bugs? How can I *trust* other people's software?

With my own software, at least I know that I don't have bugs!

Answer 2:

The packages I will talk about are developed by professionals with a lot of experience

They have extensive testsuites

For example, deal.II runs 3,000+ tests **after every single change**

Bottomline:

When having to implement software for a particular problem, re-use what others have done already

There are many high-quality, open source software libraries for every purpose in scientific computing

Use them:
– You will be far more **productive**
– You will be able to use **state-of-the-art** methods
– You will have far **fewer bugs** in your code

If you are a graduate student:

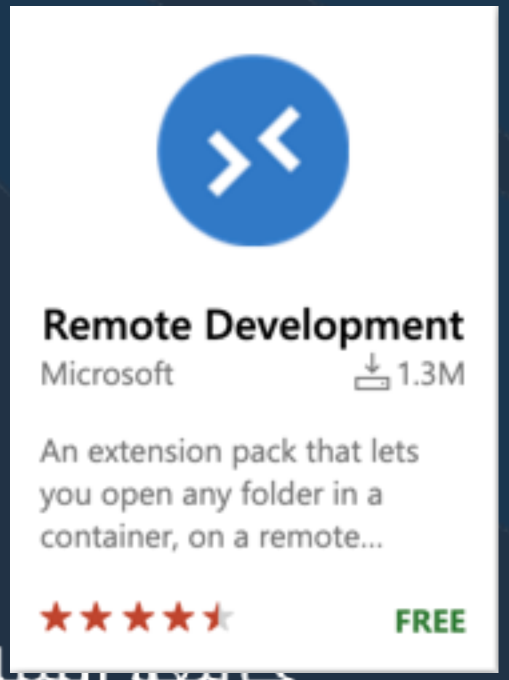
Use them because you will be able to impress your advisor with quick results!

Roadmap for next lectures:

- Recap on Sobolev spaces, and basic tensor analysis
- Recap on the Finite Element Method
- Familiarising with the tools:
  - Version control system (git)
  - Modern IDEs (VSCode)
  - Remote development using Docker inside VSCode
- “Road to Poisson”

Setting up VSCode

- Download and install **Docker**: https://www.docker.com/products/docker-desktop
  - Read some doc: https://www.docker.com/get-started
- Download and install: https://code.visualstudio.com/download
  - Read some doc: https://code.visualstudio.com/docs
  - Install the following extension:



Open the course repository

- Clone the repository of the course to a directory of your liking
- Open the directory containing the repository with Visual Studio Code (the directory contains a hidden folder, called “.**devcontainer”,** used by VSCode to understand the
- VSCode should ask you if you want to reopen the folder within a container. Say yes.
- VSCode will now download a docker image. The first time around, this will take some time.