What is good research software?
How can it be engineered?
Is it deliverable within academia?

Christopher Woods

EPSRC Research Software Engineer

Advanced Computing Research Centre / School of Chemistry

University of Bristol

What’s the problem with research software?

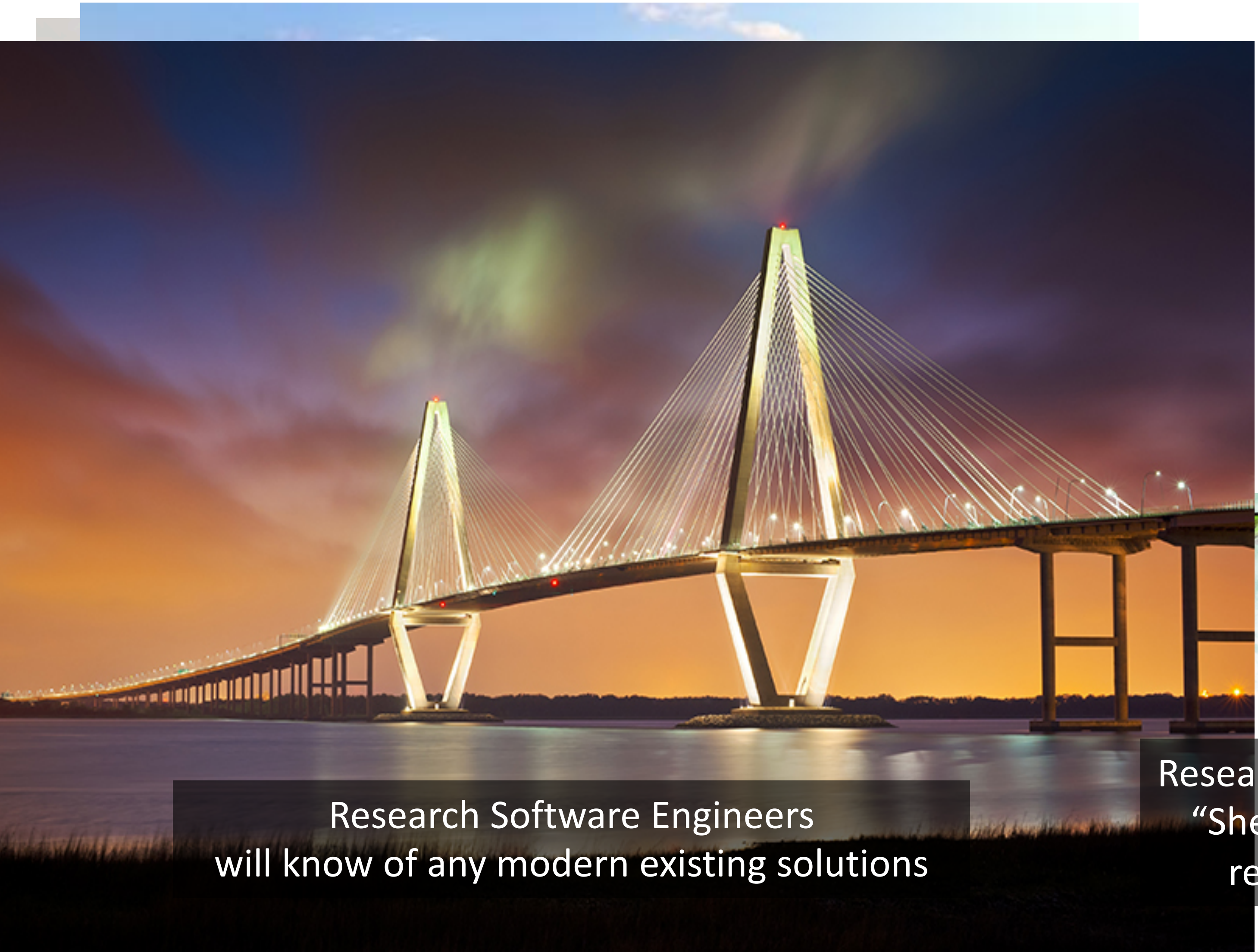
Significant cost to apply ad-hoc research software solutions at scale or in production

Re-engineering for scale-up or production use is expensive…
...especially if the researcher is no longer exploring this land...
...and they didn’t leave great notes...
...and a production solution is fundementally different to the ad-hoc solution.

Research Software Solution put into production

Research Software Solution applied to scaled-up problem

Research Software Engineers (RSEs)



Research Software Engineers
will know of any modern existing solutions

Research Software Engineers
“Sherpas” to support the
researcher explorers

UKRSE

- RSE groups all over the country
  - UCL, Bristol, Manchester, Sheffield etc. etc.
  - Support researchers via collaboration and training
- EPSRC RSE Fellowship (6 fellows, more coming?)
- UK Research Software Engineers Association
  - http://rse.ac.uk
- Annual conference
  - September 7-8 2017, Manchester (>200 attendees)

1. What is good research software?

2. How can it be engineered?

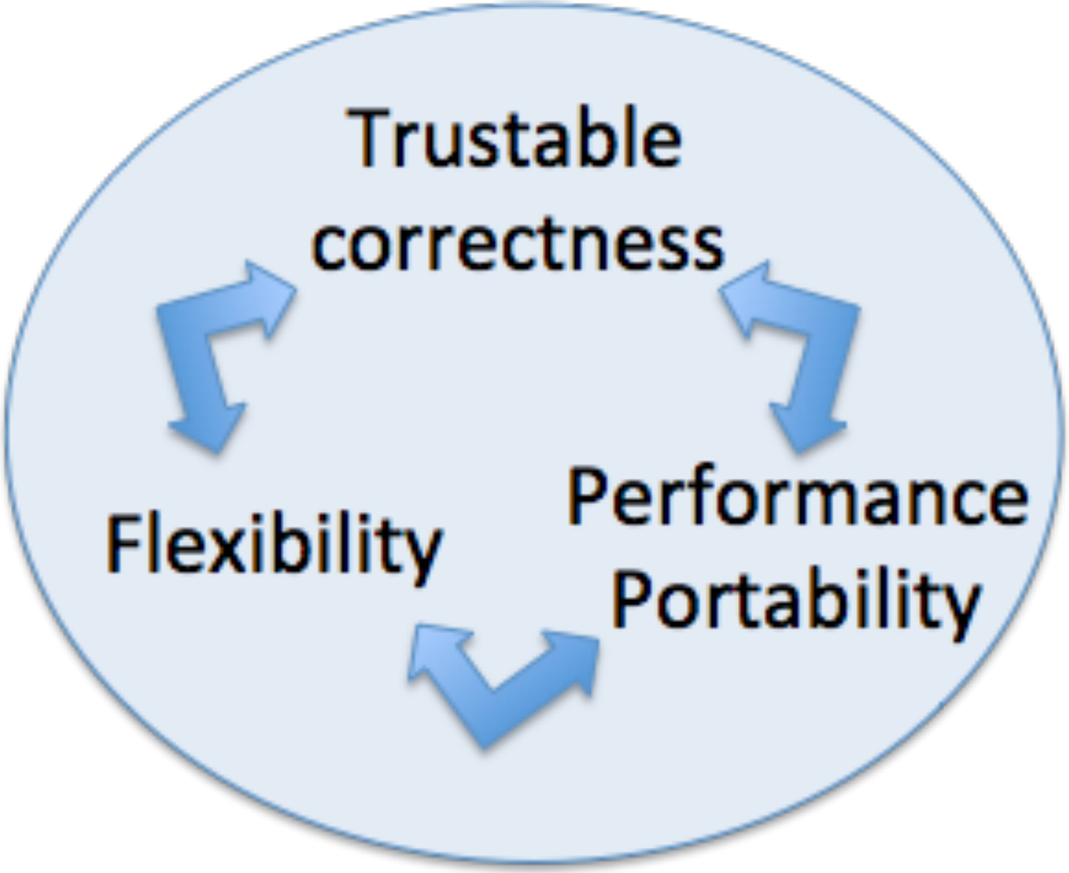
3. Is it deliverable in academia?

1. What is good research software?

2. How can it be engineered?

3. Is it deliverable in academia?

What is good research software?

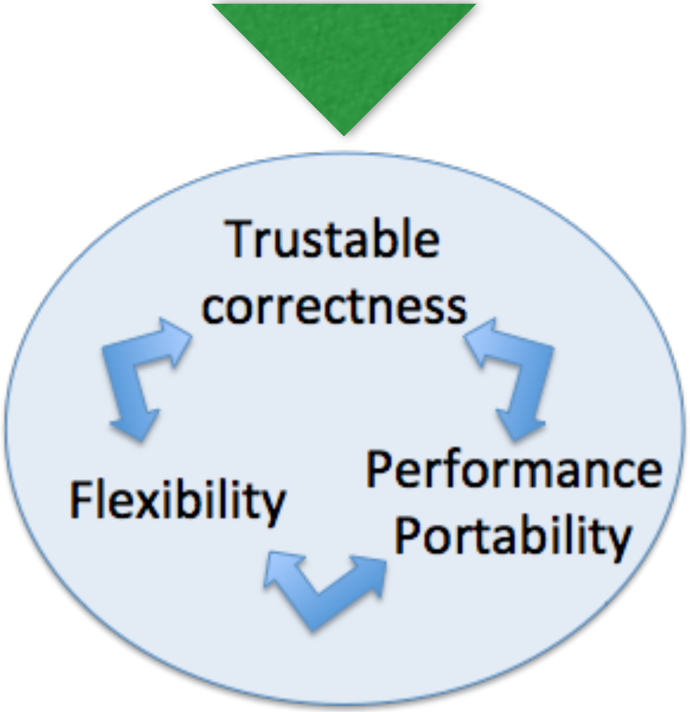


Trustable correctness

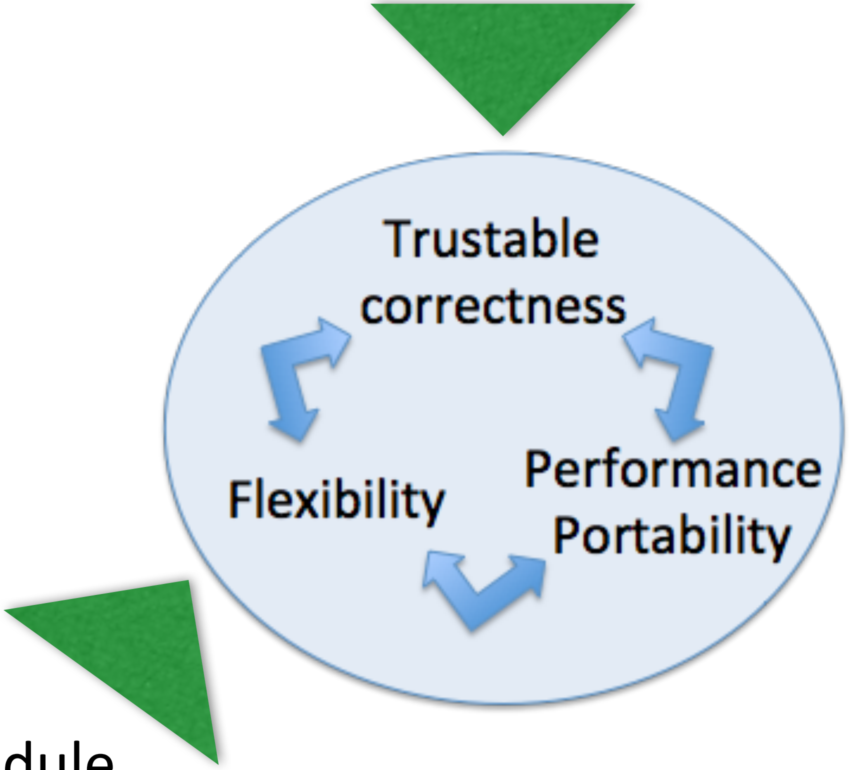
Flexibility

Performance Portability

Unit Tests / Regression Tests
Examples / Tutorials
Good Documentation



Unit Tests / Regression Tests
Examples / Tutorials
Good Documentation



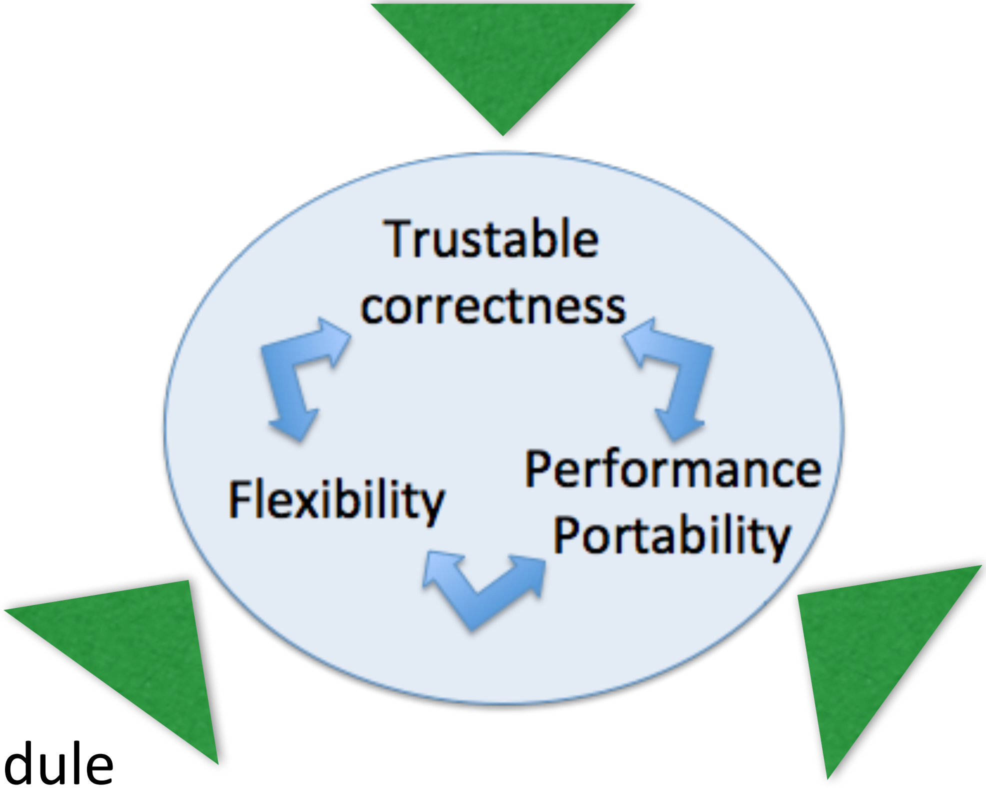
Trustable correctness

Flexibility

Performance Portability

Library / Module
Building Blocks
Re-usability

Unit Tests / Regression Tests
Examples / Tutorials
Good Documentation



Trustable correctness

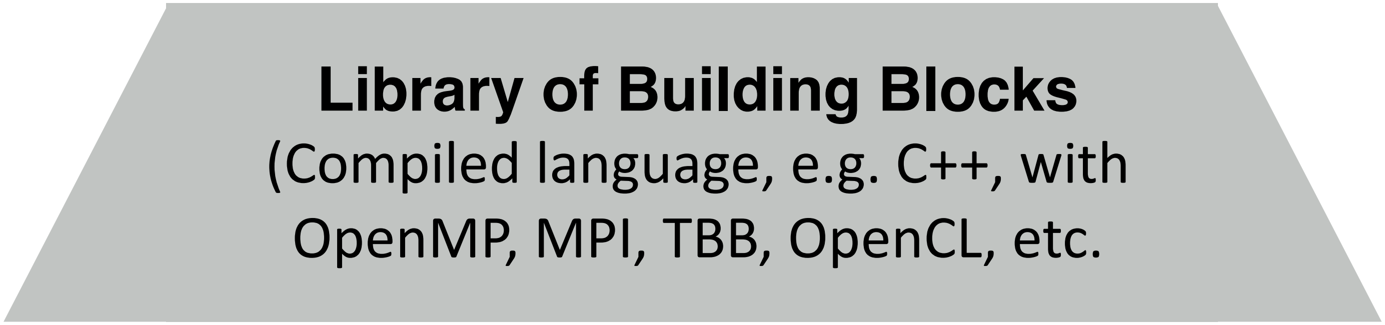
Flexibility

Performance Portability

Library / Module
Building Blocks
Re-usability

Use standards (OpenMP, MPI, TBB)
Use libraries (BLAS)
Common API to different back-ends

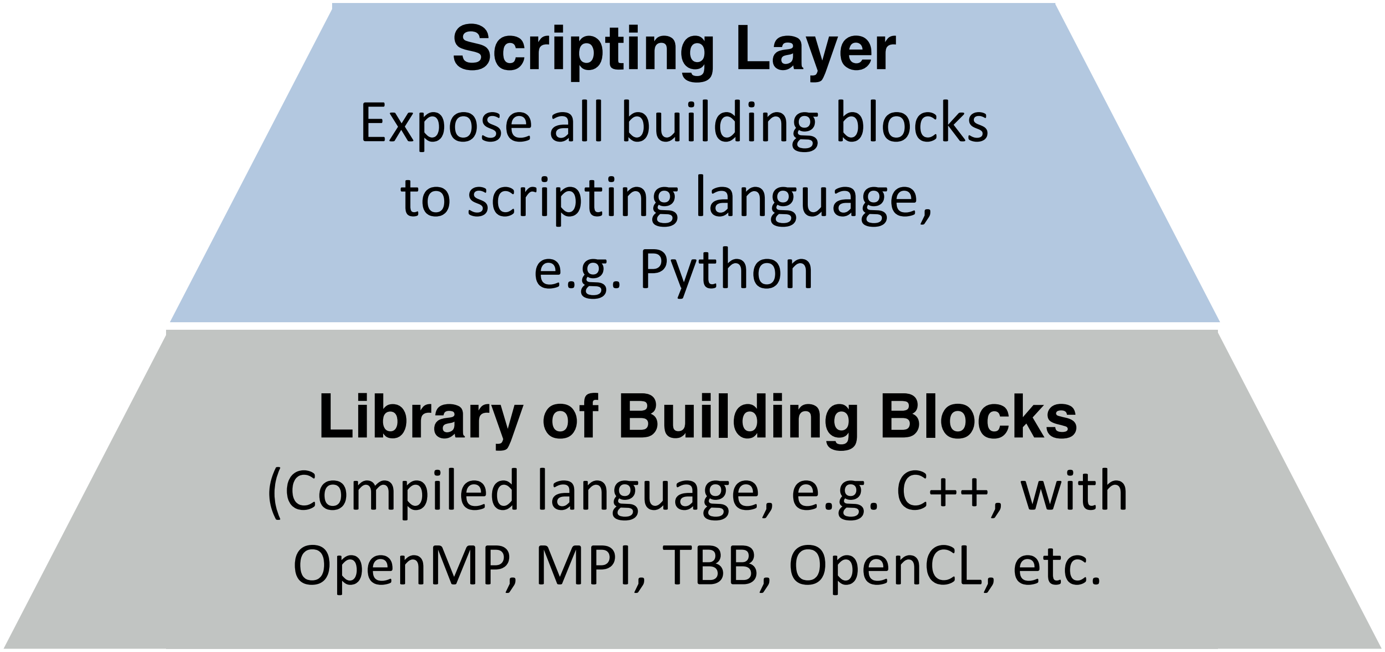
Three-Layer Model



Library of Building Blocks

(Compiled language, e.g. C++, with OpenMP, MPI, TBB, OpenCL, etc.

Three-Layer Model



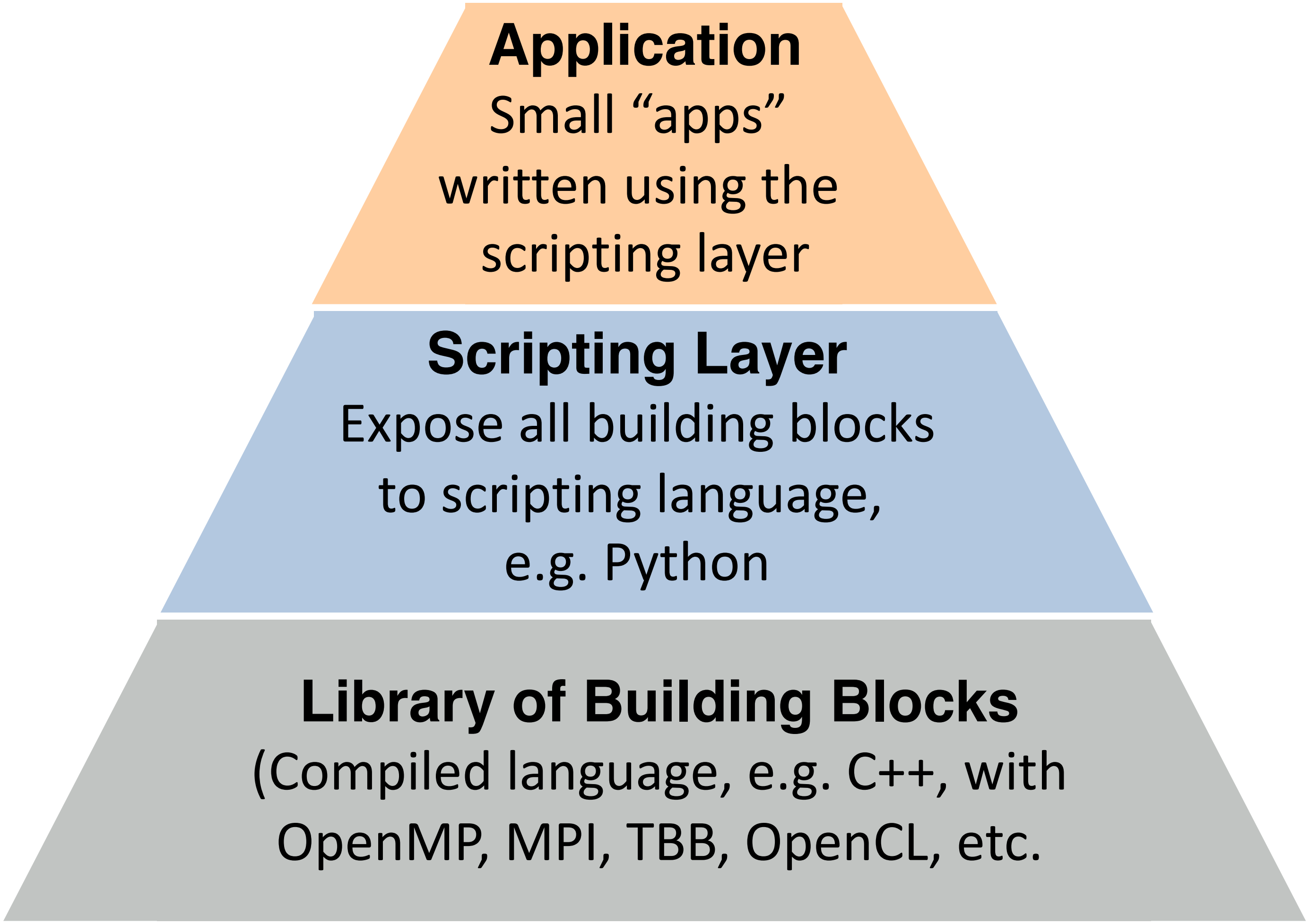
Scripting Layer

Expose all building blocks to scripting language, e.g. Python

Library of Building Blocks

(Compiled language, e.g. C++, with OpenMP, MPI, TBB, OpenCL, etc.

Three-Layer Model



Application

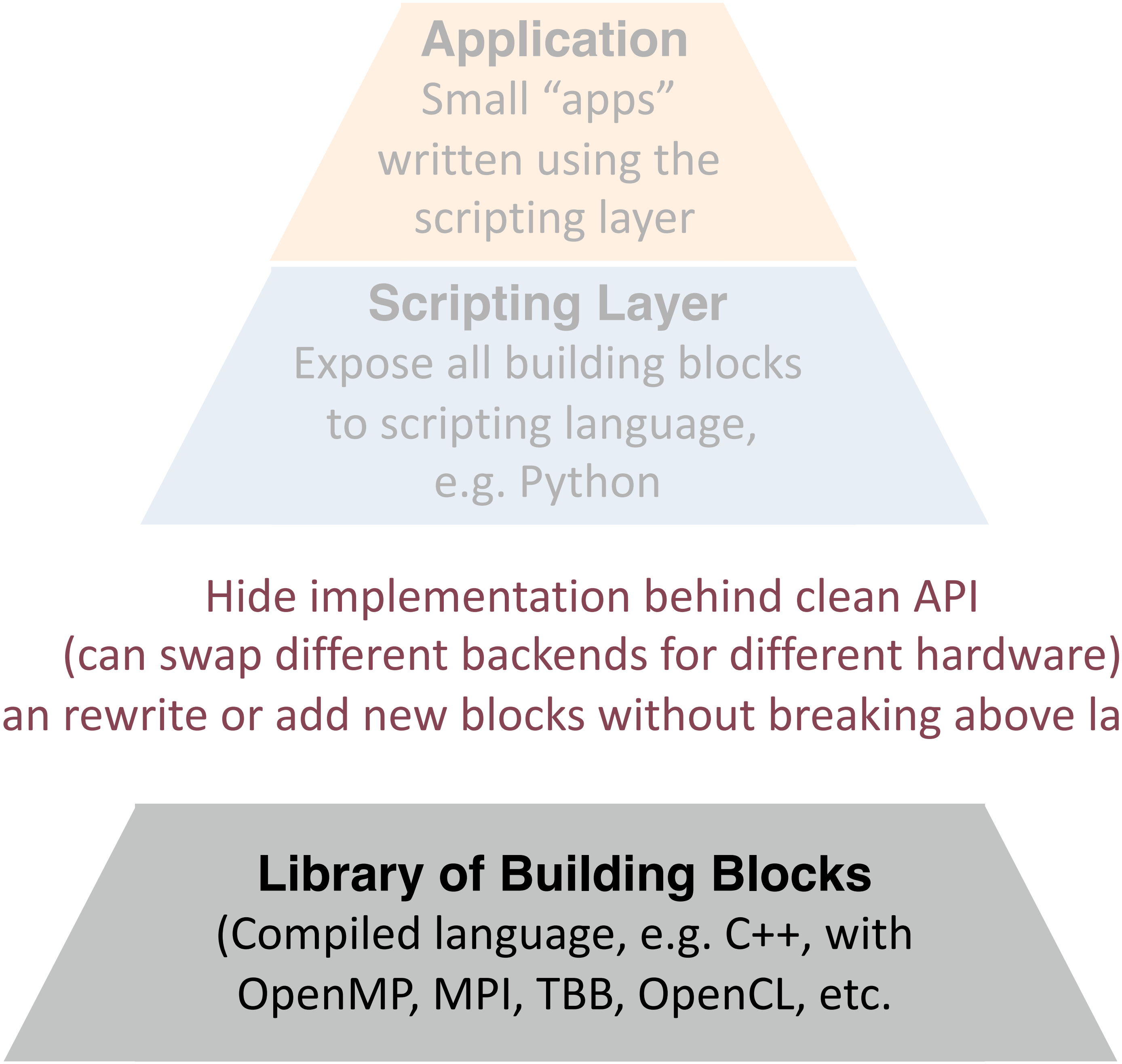
Small “apps” written using the scripting layer

Scripting Layer

Expose all building blocks to scripting language, e.g. Python

Library of Building Blocks

(Compiled language, e.g. C++, with OpenMP, MPI, TBB, OpenCL, etc.



Application

Small "apps" written using the scripting layer

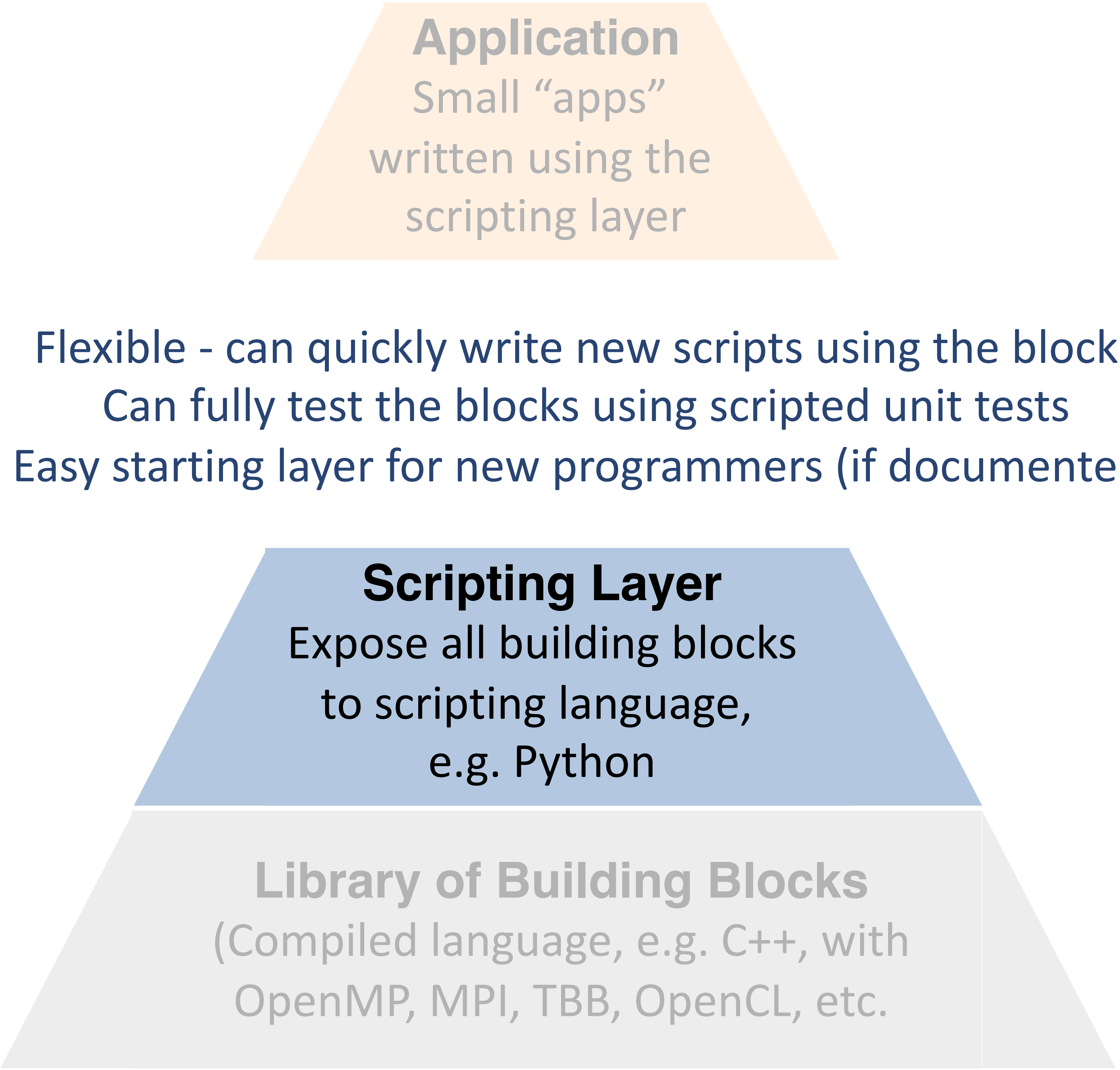
Scripting Layer

Expose all building blocks to scripting language, e.g. Python

Hide implementation behind clean API
(can swap different backends for different hardware)
(can rewrite or add new blocks without breaking above layers)

Library of Building Blocks

(Compiled language, e.g. C++, with OpenMP, MPI, TBB, OpenCL, etc.



Application

Small “apps” written using the scripting layer

Flexible - can quickly write new scripts using the blocks
Can fully test the blocks using scripted unit tests
Easy starting layer for new programmers (if documented)

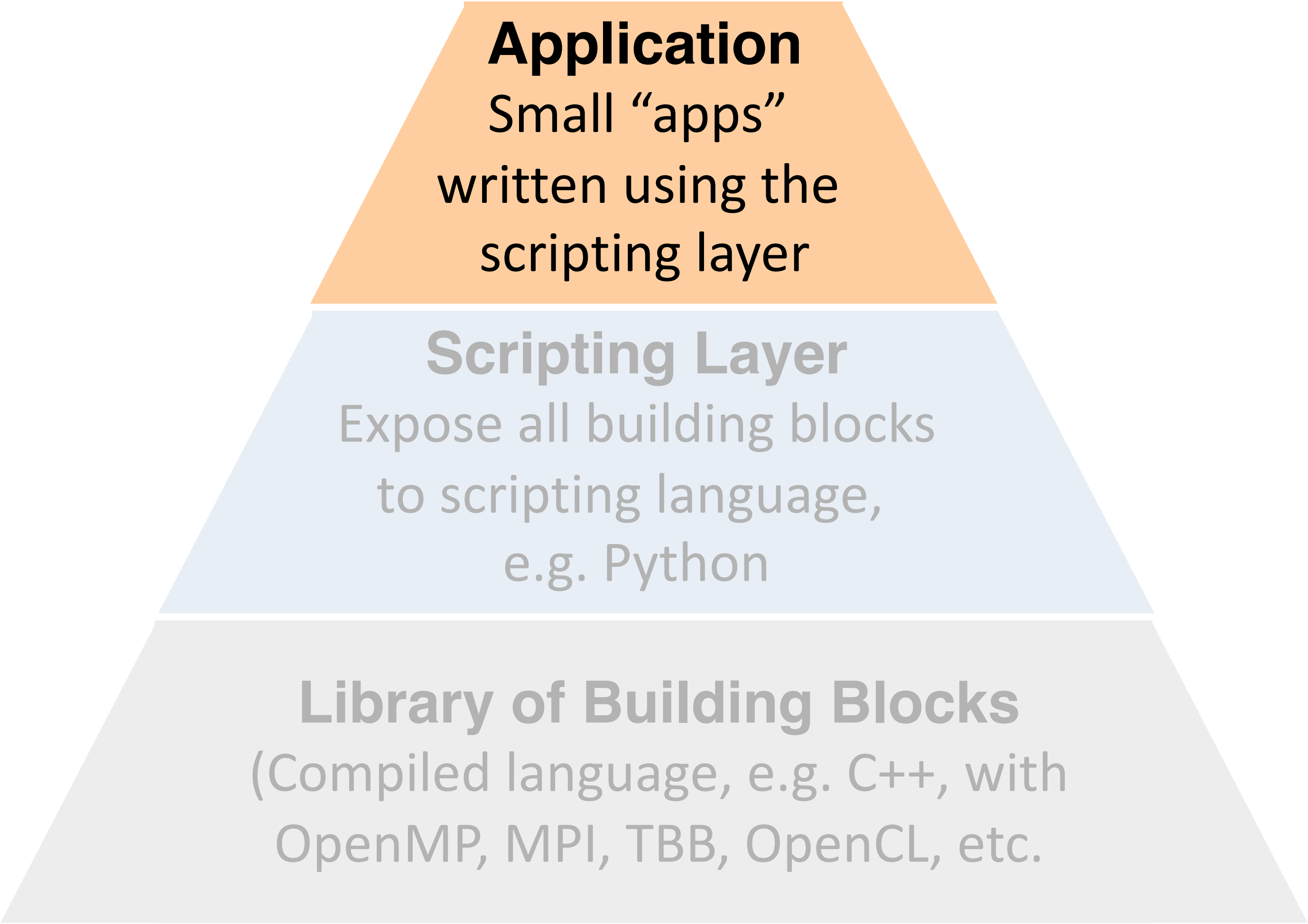
Scripting Layer

Expose all building blocks to scripting language, e.g. Python

Library of Building Blocks

(Compiled language, e.g. C++, with OpenMP, MPI, TBB, OpenCL, etc.

Easy to write apps as small scripts (using blocks)
Easy to modify apps / Easy to prototype new apps
Easy to combine apps together into workflows



Application
Small "apps"
written using the
scripting layer

Scripting Layer
Expose all building blocks
to scripting language,
e.g. Python

Library of Building Blocks
(Compiled language, e.g. C++, with
OpenMP, MPI, TBB, OpenCL, etc.

Interoperable Frameworks

- Writing this encourages you to develop a software framework
- You should aim to be interoperable with other frameworks – allow developers to mix and match bits of your framework with others
- Your code is less famous, but it is much more widely used and useful
  - Your code is part of something bigger
- You will be surprised by how people use your code

1. What is good research software?

2. How can it be engineered?

3. Is it deliverable in academia?

How Who

is the

what

when

and why

of software engineering

How *Who are you writing for?*

is the

what you need to do

when you need to do it

and *why it is important*

of software engineering

Who are you writing for?

Me

You

Most of the time, we write code that will only be used by ourselves.

(e.g. small scripts, simple analysis scripts, workflows etc.)

Who are you writing for?

You!

what you need to do

Just write your code and make sure it works.

when you need to do it

While writing your code

why it is important

The code is to complete a job now.
Spending time now on software sustainability is not worth it for a single-user, single-use script.

Who are you writing for?

Me

You

Who else is another important audience / user of your code?

Who are you writing for?

Future Me

Future You

The software you write today…
…you may want to re-use in the future.
(e.g. re-run an analysis, regenerate results to respond to referee comments etc.)

Who are you writing for?

“Future you”

what you need to do

Add comments to your code that “future you” can understand. Add a simple README on how to use it.

when you need to do it

While you are writing your code (“future you” can’t remember)

why it is important

To allow “future you” to understand what has been written, what the program does, and how to use it again.

Who are you writing for?

You

and your friends / co-workers

Your friends and co-workers may want to use your software
(or your supervisor wants to pass it to the next student…)

Who Your friends and co-workers

what & why

1. Add documentation on how to use your program.

Why? To stop you being treated as an interactive manual.

2. Add useful output and error messages / handling.

Why? To give useful information to users while the code is running, and to give useful error messages if the code cannot handle the user's input.

3. Add tests to ensure your program does what it says and has no known bugs.

Why? To minimise the risk of a bug in your code breaking your friends research and wasting their time.

Who Your friends and co-workers

what & *why*

when you need to do it

1. Add documentation on how to use your program.

Why? To stop you being treated as an interactive manual.

Before you share the code!

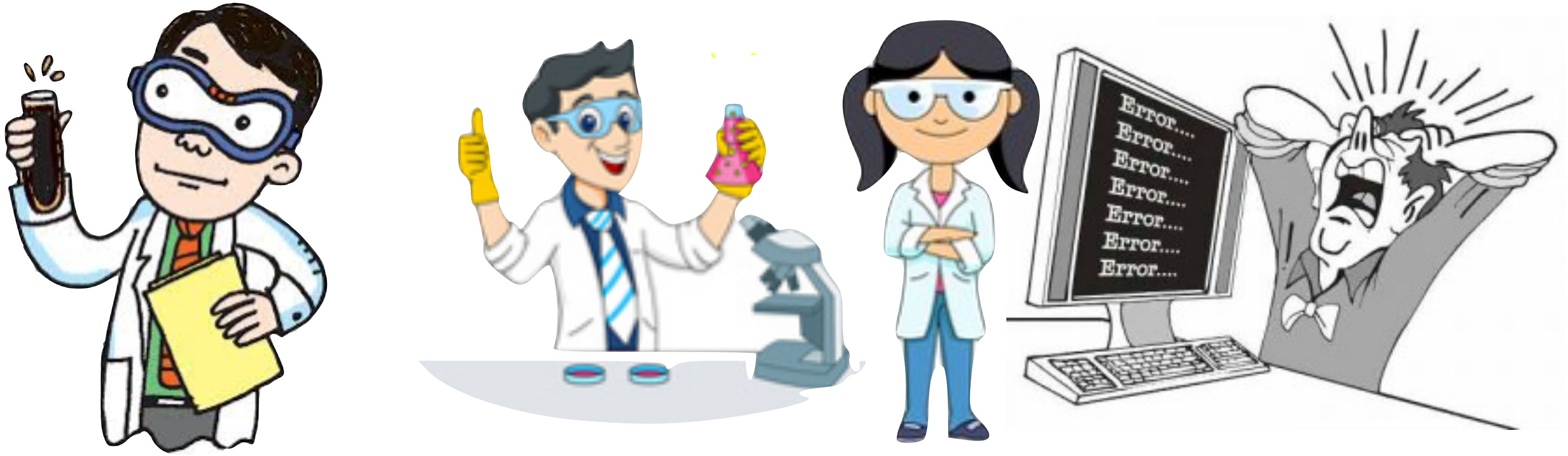
2. Add useful output and error messages / handling.

Why? To give useful information to users while the code is running, and to give useful error messages if the code cannot handle the user’s input.

3. Add tests to ensure your program does what it says and has no known bugs.

Why? To minimise the risk of a bug in your code breaking your friends research and wasting their time.

Who are you writing for?



You,

and your friends / co-workers

who want to
modify your code

The more your friends / co-workers use your code, the more they may want to modify it.

Who Friendly / local developers

what & why

1. Fully document your code (APIs, public/private code, broken parts, roadmap of features, simple developer guide etc.)

Why? To allow other developers to understand the intent of the code and how they can safely make modifications.

2. Add you code to a version control system.

Why? To allow others to contribute their code to a shared space and let you merge and manage different contributions.

Who Friendly / local developers

what & why

3. Set up an issue tracker and forum to co-ordinate releases.

Why? To allow you to communicate with developers, and share knowledge of what code is broken, and co-ordinate what all of you are doing. Provides a visible written record (avoids fights!)

4. Create robust regression tests to catch breakage.

Why? To stop others from breaking important features in your code whenever they commit a change. Detects faults early (assigns blame, places onus on developers not to break code, or to fix it themselves.)

Who Friendly / local developers

what & *why*

when you need to do it

While you are writing your code and before other developers start trying to contribute

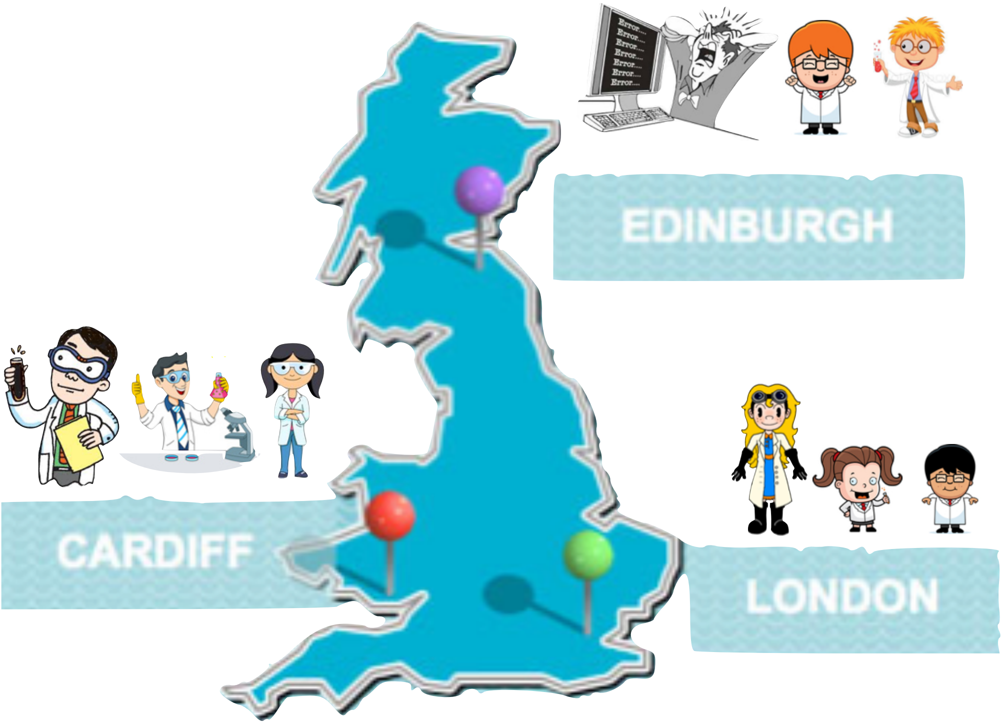
3. Set up an issue tracker and forum to co-ordinate releases.

Why? To allow you to communicate with developers, and share knowledge of what code is broken, and co-ordinate what all of you are doing. Provides a visible written record (avoids fights!)

4. Create robust regression tests to catch breakage.

Why? To stop others from breaking important features in your code whenever they commit a change. Detects faults early (assigns blame, places onus on developers not to break code, or to fix it themselves.)

Who are you writing for?



Used by people in other friendly research groups, e.g. your collaborators, friends, former colleagues who have moved jobs or been promoted

Who Non-local collaborators

what & why

1. Add good user-level documentation (user manual, simple tutorials and examples)

Why? To allow new people to use your code without you providing training (else you will have to travel/skype!)

2. Add web-based forums, issue trackers, version control.

Why? To allow you to co-ordinate use and development without being physically present or able to meet face-to-face. Web-based, as must exist outside your university (or be accessible to people who don't work at your uni)

Who Non-local collaborators

what & why

3. Add easy install scripts, ensure code is portable.

Why? To allow your code to compile and install on collaborators computers, which may have different OSs than yours (some may use OS X or Windows…!)

4. Create a governance structure for the software

Why? Many people depend on you and your software. You need to plan for your succession, e.g. if you are promoted, leave for industry, or suffer a bus error. Increasingly, the code is no longer "yours", but is beginning to belong to its community.

Who Non-local collaborators

what & *why*

when you need to do it

While you are writing your code and before you release it to your collaborators

3. Add easy install scripts, ensure code is portable.

Why? To allow your code to compile and install on collaborators computers, which may have different OSs than yours (some may use OS X or Windows…!)

4. Create a governance structure for the software

Why? Many people depend on you and your software. You need to plan for your succession, e.g. if you are promoted, leave for industry, or suffer a bus error. Increasingly, the code is no longer “yours”, but is beginning to belong to its community.

Who are you writing for?

Certificate of Brilliance

This certificate that you are known to produce absolutely amazing software

So far everyone uses your software
does so because they know and trust you…

Who are you writing for?

You want to write software for anyone to use, anywhere in the World.

You don’t know them, and they don’t know you.

Who Anyone (e.g. People Who Don’t Know You)

what & why

1. Create a website to market and allow people to discover your code.

Why? To allow others to quickly find and install your code. You need to market it, or else it is invisible.

2. Package the code. Make it extremely easy to install.

Why? To allow others to use your code as easily and quickly as possible after download. From my experience, PWDKYs will give up on your software if they cannot install it and have it working within five minutes. Everyone hates dependency hell.

Who Anyone (e.g. People Who Don’t Know You)

what & why

3. Develop on-line training workshops and examples.

Why? To allow others to independently learn how to use your software. They will never meet you or talk to you, so give them everything they need to learn independently.

4. Create a good suite of unit and regression tests.

Why? To allow others to independently verify your code is working. Remember, they don’t know you, so they should not trust you or your software. Give them tests so that they can learn to trust your code.

Who Anyone (e.g. People Who Don’t Know You)

what & why

5. Make your code compatible with other software and with data standards (e.g. standard input/output formats such as PDB, mol2, etc. etc.)

Why? To allow others to maximise your code’s usefulness by merging it into their workflows and developing derived software. Try to make your code a “good citizen” and play nicely with other software packages. If you do, then your users will thank you, and they will create interesting workflows that are beyond anything you can imagine.

Who Anyone (e.g. People Who Don’t Know You)

what & why

when you need to do it

5. Make your code compatible with other software and with data standards (e.g. standard input/output formats such as PDB, mol2, etc. etc.)

Why? To allow others to maximise your code’s usefulness by merging it into their workflows and developing derived software. Try to make your code a “good citizen” and play nicely with other software packages. If you do, then your users will thank you, and they will create interesting workflows that are beyond anything you can imagine.

While you are writing your code, and (ideally) before you release it to the public.

(*) Don’t leave these things as jobs for the final release, and then subject your users to unfinished alpha, beta or pre-release candidates. You don’t want your code to develop a reputation of being difficult to install, difficult to learn, broken or buggy

Who are you writing for?

You want anyone to be able to download
and use your program, anywhere in the World

Who are you writing for?

You want anyone to be able to download
and use your program, anywhere in the World
and you want it to be sustainable
and useful long into the future.

Who Anyone, and for a long time into the future

what & why

1. Create a management team, with release planning, feature planning, and change management.

Why? To ensure there is a clear roadmap for the software, and that there is succession planning in place to keep development going as people join and leave the project

2. Create a support team to deal with support requests, feature requests etc.

Why? To ensure that users are supported, and that the software adapts to changes in operating systems, hardware and use cases for the code. To keep the trust of your community, you will need to spend a lot of time supporting them and keeping them engaged.

Who Anyone, and for a long time into the future

what & why

3. Create a training team to handle creation and management of documentation, training materials, workshops.

Why? To ensure that knowledgeable user and developer communities can grow and feel engaged. So that you have the people to develop and host training events around the World.

4. Create a clear funding mechanism that will support the team and software long into the future.

Why? To ensure that people and teams are funded, and that people can be employed to do the unglamorous and otherwise unrewarding jobs.

Who Anyone, and for a long time into the future

what & *why*

3. Create a training team to handle creation and management of documentation, training materials, workshops.

when you need to do it

While you are managing the software project (you may not now have time for any coding…)

Why? To ensure that knowledgeable user and developer communities can grow and feel engaged. So that you have the people to develop and host training events around the World.

4. Create a clear funding mechanism that will support the team and software long into the future.

Why? To ensure that people and teams are funded, and that people can be employed to do the unglamorous and otherwise unrewarding jobs.

Who has the time for all of this?

- Not every code needs to be released sustainably for anyone to use for all time.
  - Recognise early on WHO you are writing your code for.
  - Don’t share personal “quick and dirty” software with others unless you have a plan for how to support and develop it into the future.

- Have an exit plan. How will you ensure that you are not the one supporting the code for the rest of your life…

Who has the time for all of this?

- Work with others. Rather than start a new code from scratch, join an existing community. Contribute to existing code and projects. That way, you don’t need to set everything up yourself.

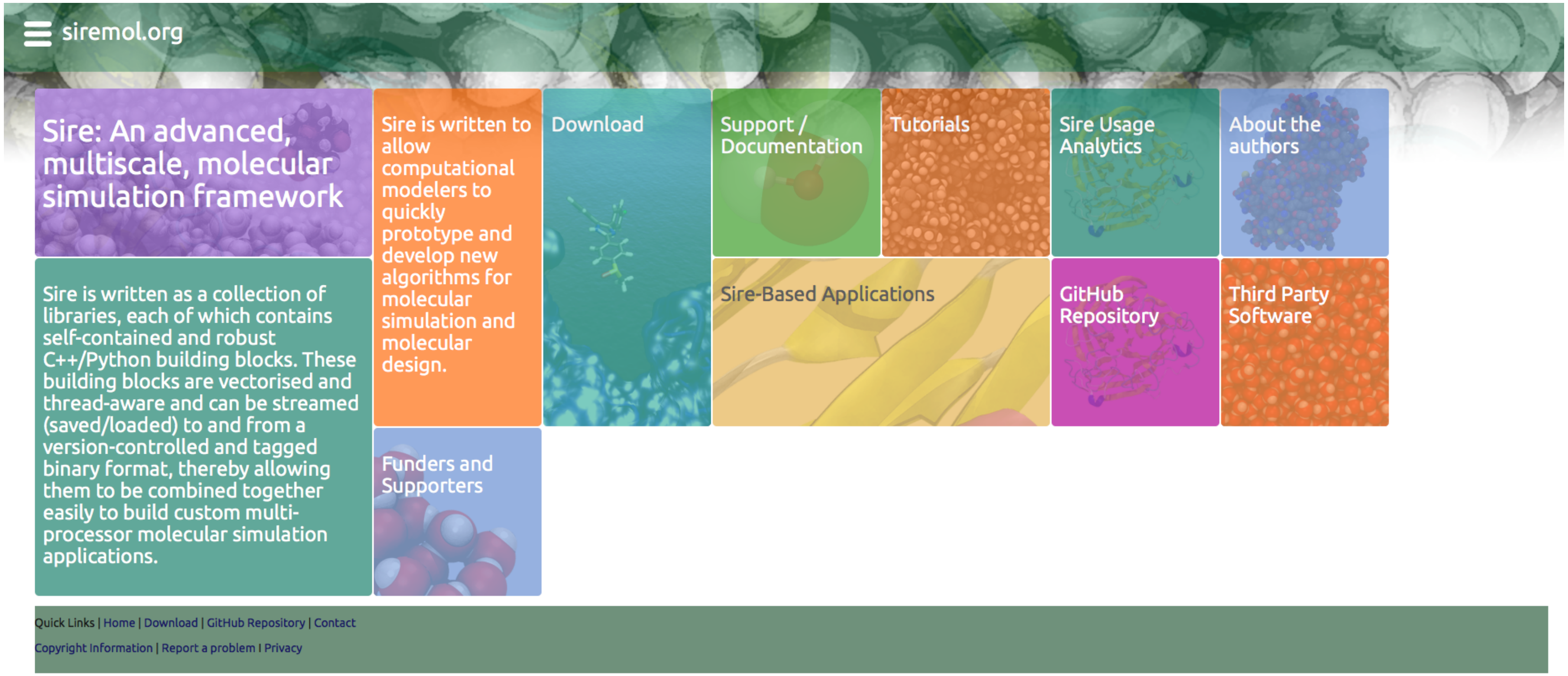
- Be warned, other projects don’t appreciate “code dumps” of unsupportable code… At the very least expect to have to write documentation, tests, examples and to answer support requests on the forums

1. What is good research software?

2. How can it be engineered?

3. Is it deliverable in academia?

Lessons from Developing Sire



siremol.org

Sire: An advanced, multiscale, molecular simulation framework

Sire is written to allow computational modelers to quickly prototype and develop new algorithms for molecular simulation and molecular design.

Download

Support / Documentation

Tutorials

Sire Usage Analytics

About the authors

Sire is written as a collection of libraries, each of which contains self-contained and robust C++/Python building blocks. These building blocks are vectorised and thread-aware and can be streamed (saved/loaded) to and from a version-controlled and tagged binary format, thereby allowing them to be combined together easily to build custom multi-processor molecular simulation applications.

Sire-Based Applications

GitHub Repository

Third Party Software

Funders and Supporters

Quick Links | Home | Download | GitHub Repository | Contact

Copyright Information | Report a problem I Privacy

http://siremol.org

What does it do?

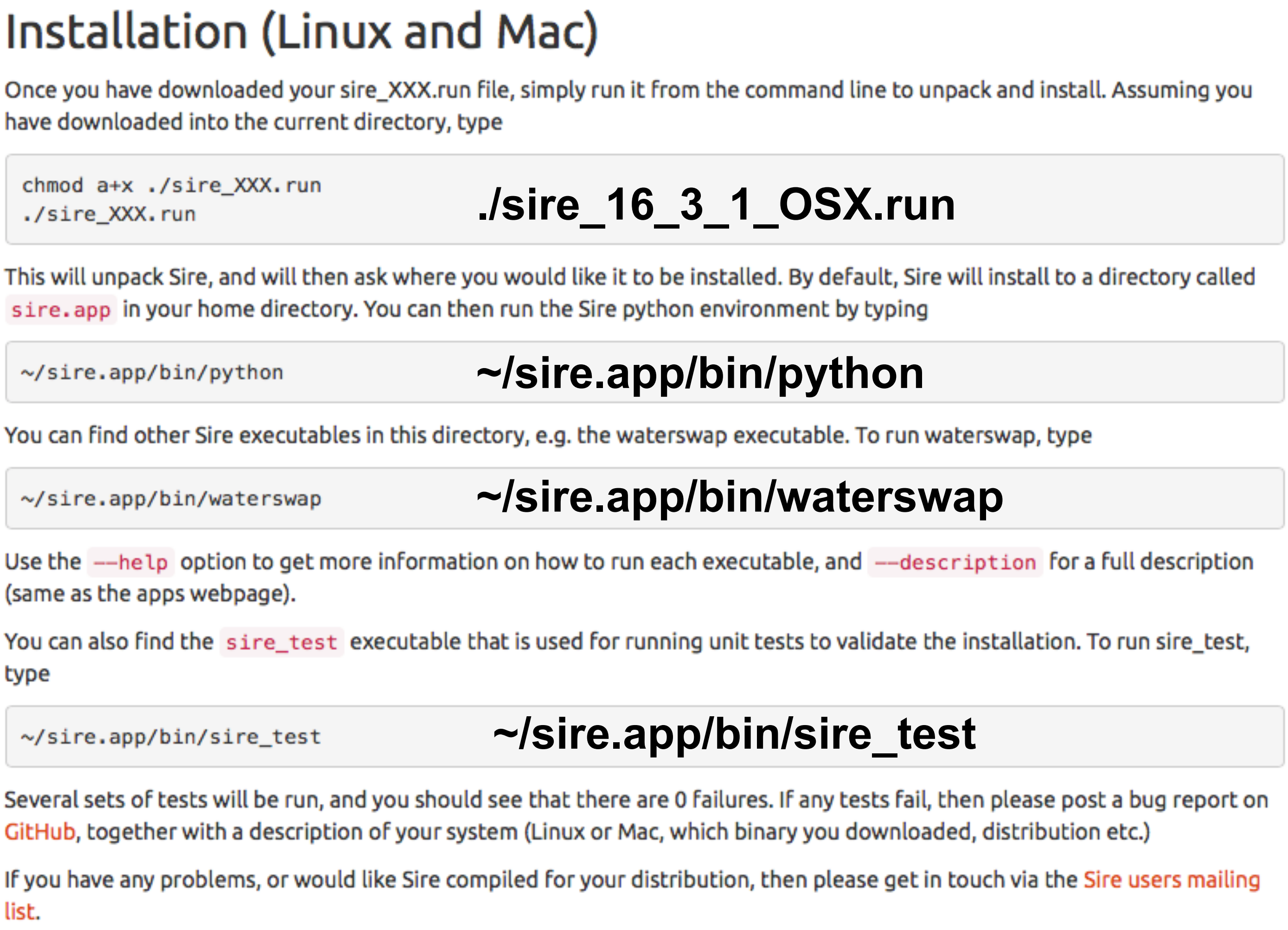
- Molecular simulation framework
- Written in C++. Objects exposed to Python
- > 200,000 lines of code, developed since 2005
- Provides building blocks to rapidly write new molecular simulation programs (apps)
- Example apps include “waterswap”, “ligandswap”, “nautilus”, “sommd”, “FESetup”

Who develops Sire?

- Software development led from Bristol
- Secondary team at Edinburgh
- Community development in partnership with CCP-BioSim and STFC
- Funding from EPSRC/BBSRC/EU and Industry
  - **but no specific funding for Sire – grants fund method development and scientific research**

How is it published?

- Sire is open source (GPL2)
- Main website is http://siremol.org
- Release planning on GitHub wiki / GitHub issues
- Development on a public GitHub repository
- Unit testing + Continuous integration via Travis-CI
- Published as source and binary regularly through the year, i.e. 2016.3.1



Installation (Linux and Mac)

Once you have downloaded your sire_XXX.run file, simply run it from the command line to unpack and install. Assuming you have downloaded into the current directory, type

```
chmod a+x ./sire_XXX.run
./sire_XXX.run
```

./sire_16_3_1_OSX.run

This will unpack Sire, and will then ask where you would like it to be installed. By default, Sire will install to a directory called `sire.app` in your home directory. You can then run the Sire python environment by typing

```
~/sire.app/bin/python
```

~/sire.app/bin/python

You can find other Sire executables in this directory, e.g. the waterswap executable. To run waterswap, type

```
~/sire.app/bin/waterswap
```

~/sire.app/bin/waterswap

Use the `--help` option to get more information on how to run each executable, and `--description` for a full description (same as the apps webpage).

You can also find the `sire_test` executable that is used for running unit tests to validate the installation. To run sire_test, type

```
~/sire.app/bin/sire_test
```

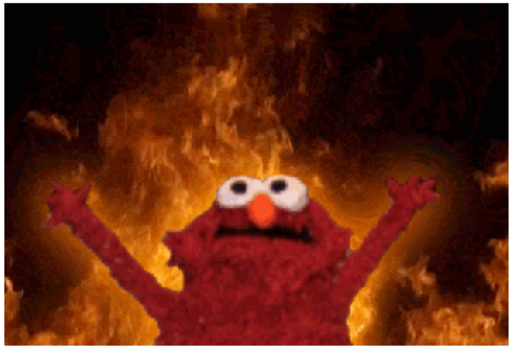
~/sire.app/bin/sire_test

Several sets of tests will be run, and you should see that there are 0 failures. If any tests fail, then please post a bug report on GitHub, together with a description of your system (Linux or Mac, which binary you downloaded, distribution etc.)

If you have any problems, or would like Sire compiled for your distribution, then please get in touch via the Sire users mailing list.

Lessons learned from Sire: portability

- **Compiling Sire in 2011**



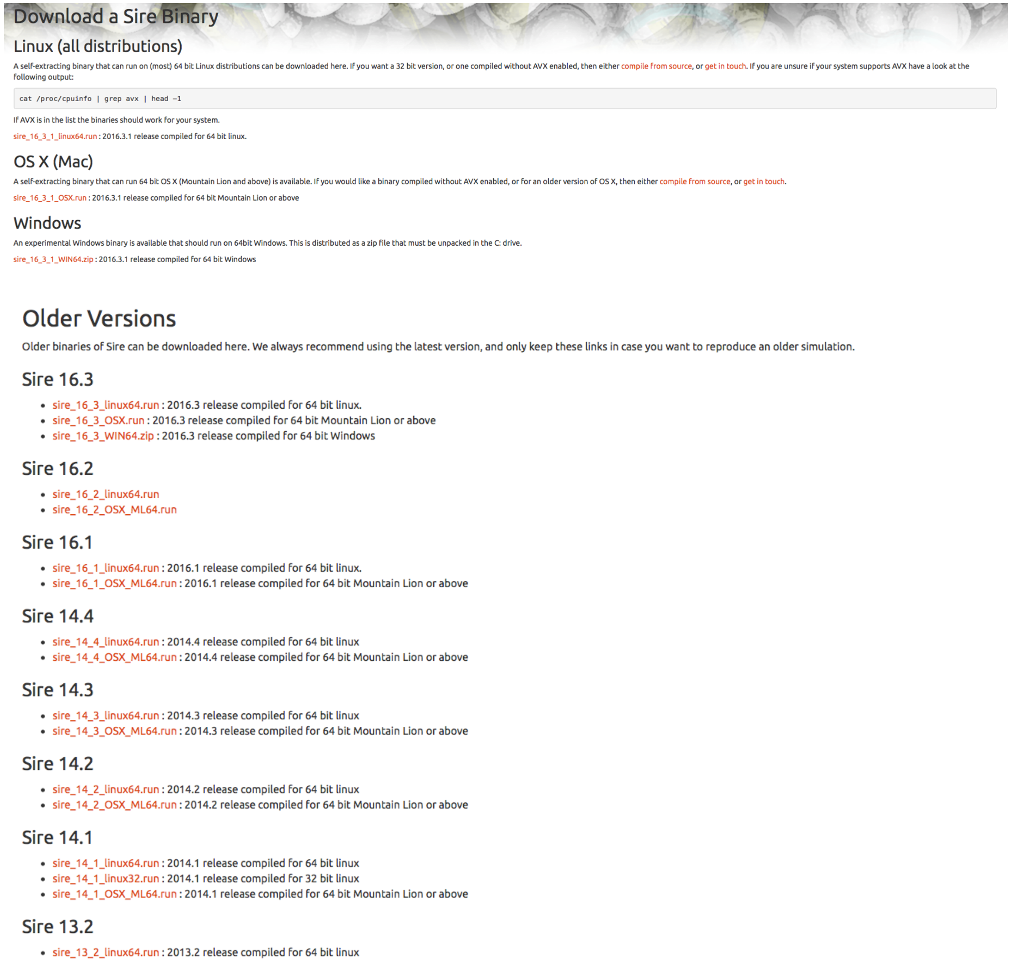
- **Compiling Sire in 2016**

```
git clone https://github.com/michellab/sire
cd sire
./compile-sire.sh
```

- How?
  - Switch to conda to manage python dependencies
  - Lots of cmake magic from Chris Woods

Reproducibility

- Binaries bundle all dependent libraries
  - Linux 64bit > 2008
  - OS X >= 10.8 (2008)
  - Win64 >= 7 (2009)
- Old versions always available



Download a Sire Binary

Linux (all distributions)

A self-extracting binary that can run on (most) 64 bit Linux distributions can be downloaded here. If you want a 32 bit version, or one compiled without AVX enabled, then either compile from source, or get in touch. If you are unsure if your system supports AVX have a look at the following output:

```
cat /proc/cpuinfo | grep avx | head -1
```

If AVX is in the list the binaries should work for your system.

sire_16_3_1_linux64.run : 2016.3.1 release compiled for 64 bit linux.

OS X (Mac)

A self-extracting binary that can run 64 bit OS X (Mountain Lion and above) is available. If you would like a binary compiled without AVX enabled, or for an older version of OS X, then either compile from source, or get in touch.

sire_16_3_1_OSX.run : 2016.3.1 release compiled for 64 bit Mountain Lion or above

Windows

An experimental Windows binary is available that should run on 64bit Windows. This is distributed as a zip file that must be unpacked in the C: drive.

sire_16_3_1_WIN64.zip : 2016.3.1 release compiled for 64 bit Windows

Older Versions

Older binaries of Sire can be downloaded here. We always recommend using the latest version, and only keep these links in case you want to reproduce an older simulation.

Sire 16.3

- sire_16_3_linux64.run : 2016.3 release compiled for 64 bit linux.
- sire_16_3_OSX.run : 2016.3 release compiled for 64 bit Mountain Lion or above
- sire_16_3_WIN64.zip : 2016.3 release compiled for 64 bit Windows

Sire 16.2

- sire_16_2_linux64.run
- sire_16_2_OSX_ML64.run

Sire 16.1

- sire_16_1_linux64.run : 2016.1 release compiled for 64 bit linux.
- sire_16_1_OSX_ML64.run : 2016.1 release compiled for 64 bit Mountain Lion or above

Sire 14.4

- sire_14_4_linux64.run : 2014.4 release compiled for 64 bit linux
- sire_14_4_OSX_ML64.run : 2014.4 release compiled for 64 bit Mountain Lion or above

Sire 14.3

- sire_14_3_linux64.run : 2014.3 release compiled for 64 bit linux
- sire_14_3_OSX_ML64.run : 2014.3 release compiled for 64 bit Mountain Lion or above

Sire 14.2

- sire_14_2_linux64.run : 2014.2 release compiled for 64 bit linux
- sire_14_2_OSX_ML64.run : 2014.2 release compiled for 64 bit Mountain Lion or above

Sire 14.1

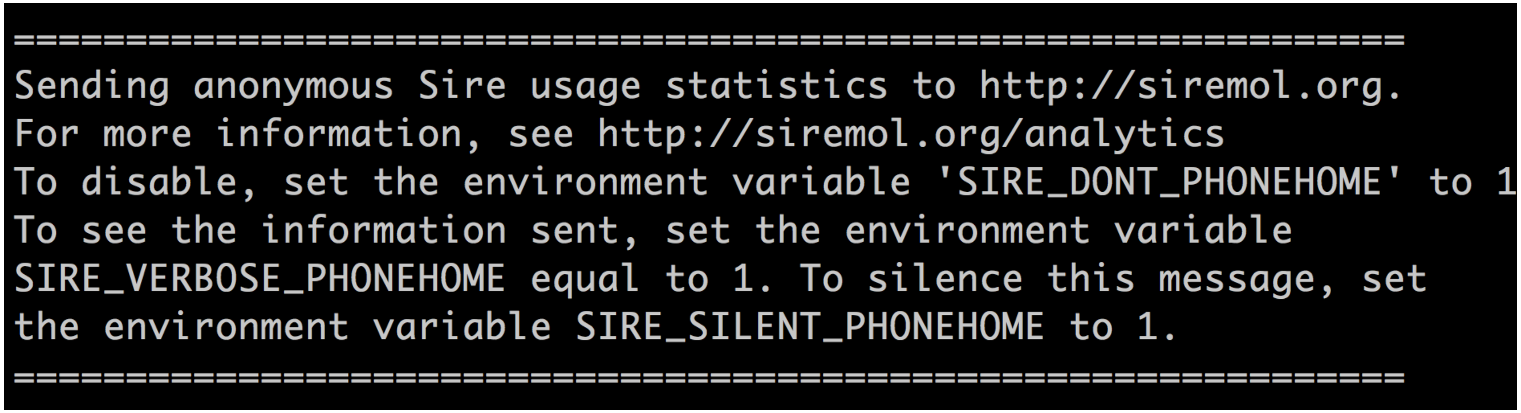
- sire_14_1_linux64.run : 2014.1 release compiled for 64 bit linux
- sire_14_1_linux32.run : 2014.1 release compiled for 32 bit linux
- sire_14_1_OSX_ML64.run : 2014.1 release compiled for 64 bit Mountain Lion or above

Sire 13.2

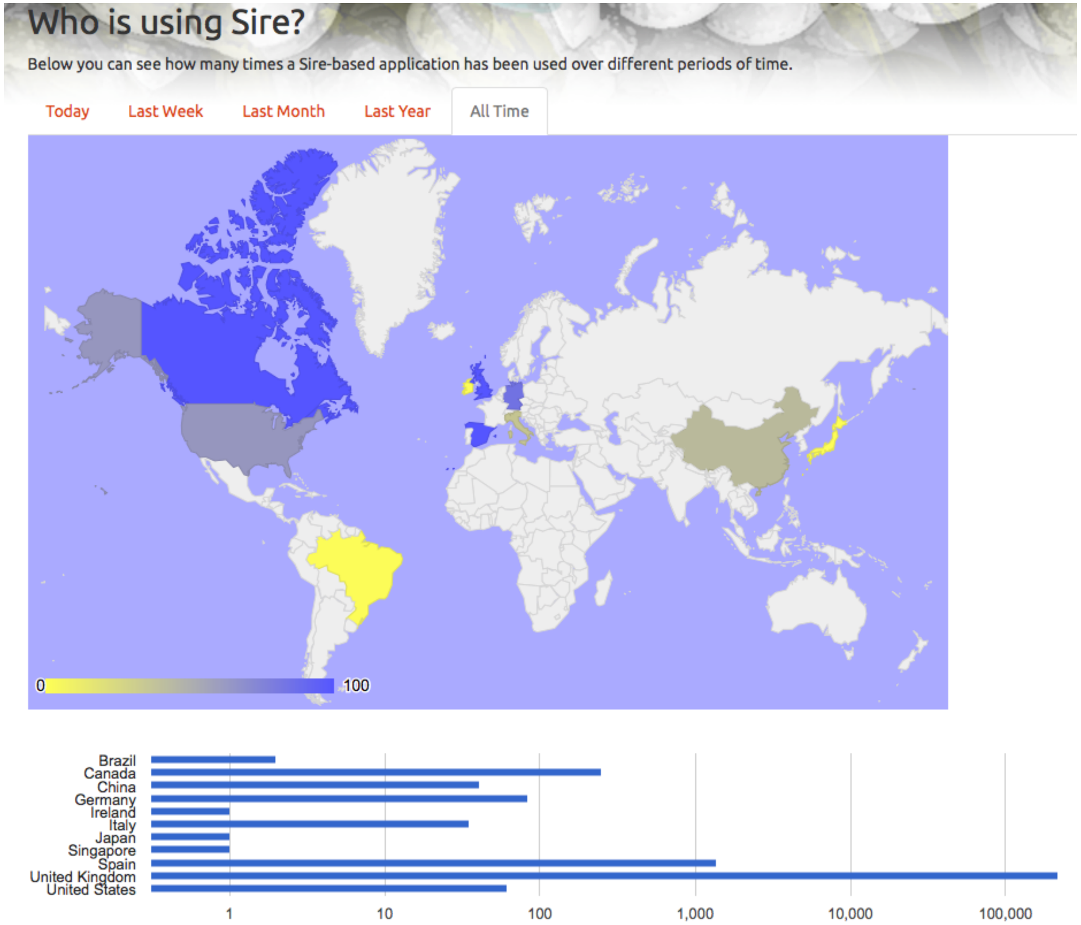
- sire_13_2_linux64.run : 2013.2 release compiled for 64 bit linux

Usage Tracking

- From Sire 2015.1 we have “phonehome” analytics built into Sire
- Sends usage information back to siremol.org



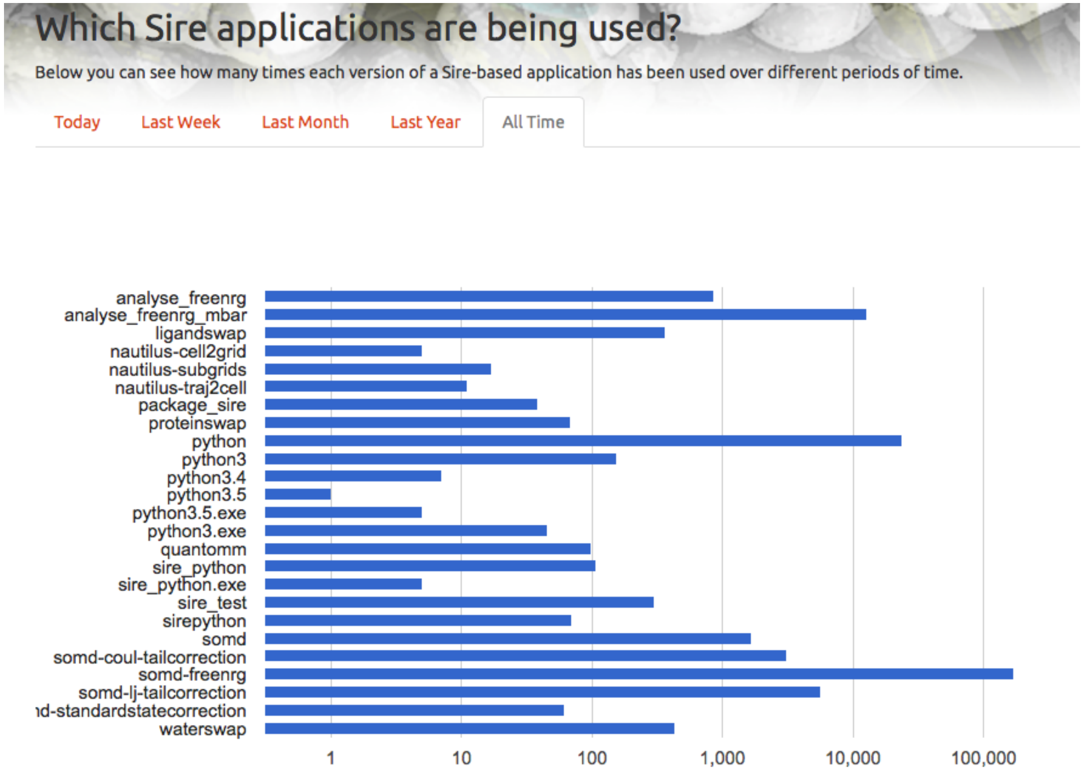
```
==============================================================
Sending anonymous Sire usage statistics to http://siremol.org.
For more information, see http://siremol.org/analytics
To disable, set the environment variable 'SIRE_DONT_PHONEHOME' to 1
To see the information sent, set the environment variable
SIRE_VERBOSE_PHONEHOME equal to 1. To silence this message, set
the environment variable SIRE_SILENT_PHONEHOME to 1.
==============================================================
```



Who is using Sire?

Below you can see how many times a Sire-based application has been used over different periods of time.

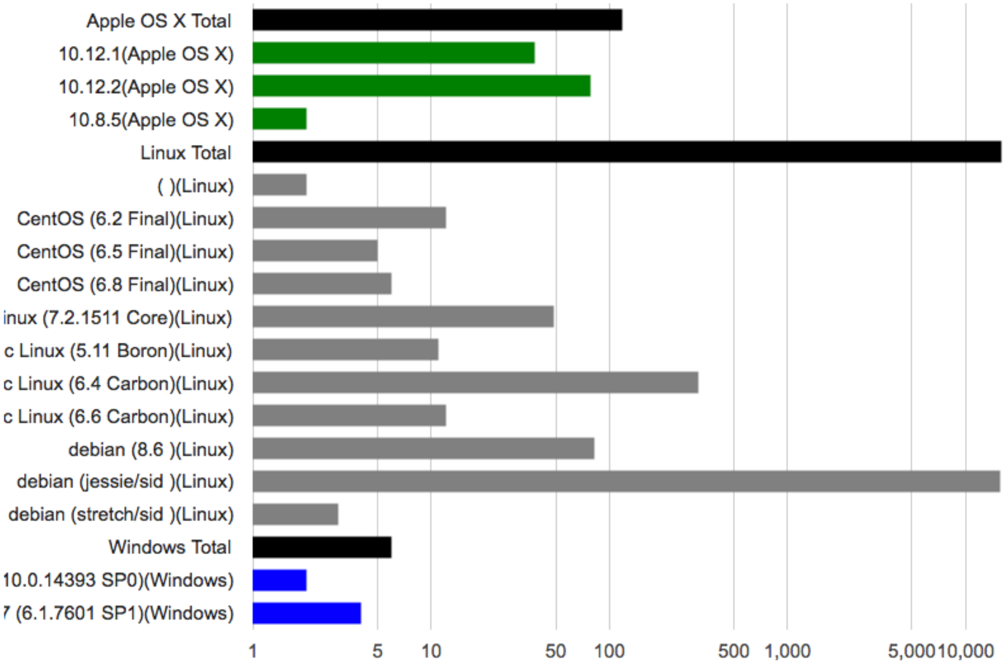
- **Country of user**
- Which app is being used
- Operating system
- Version of Sire
- Compile options etc.
- Basic computer info



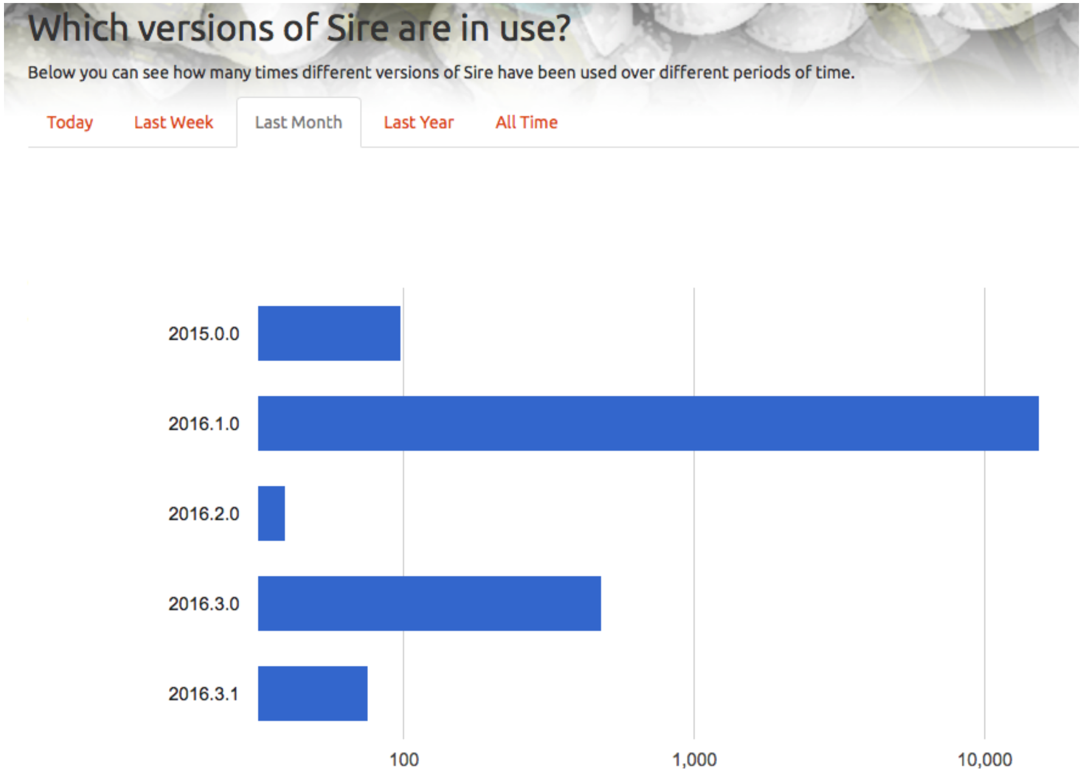
Which Sire applications are being used?

Below you can see how many times each version of a Sire-based application has been used over different periods of time.

- Country of user
- **Which app is being used**
- Operating system
- Version of Sire
- Compile options etc.
- Basic computer info



- Country of user
- Which app is being used
- **Operating system**
- Version of Sire
- Compile options etc.
- Basic computer info



Which versions of Sire are in use?

Below you can see how many times different versions of Sire have been used over different periods of time.

- Country of user
- Which app is being used
- Operating system
- **Version of Sire**
- Compile options etc.
- Basic computer info

Just started download tracking - ~40 downloads per week, from around the World

Conclusions

- Research software engineering is more work than just writing code…
- It is deliverable within academia, but time and effort have to be provided
- It is difficult to do everything yourself
- Contributing to an existing project is more sustainable and requires less effort
- Write a software management plan before embarking on writing a new code

Acknowledgements

- Research Software Engineering
  - Neil Chue Hong, Software Sustainability Institute
  - UKRSE: http://rse.ac.uk
  - EPSRC RSE Fellowship scheme
- Sire
  - Julien Michel, University of Edinburgh
    - Antonia Mey, Gaetano Calabró
  - Hannes Loeffler, STFC
  - Adrian Mulholland, University of Bristol