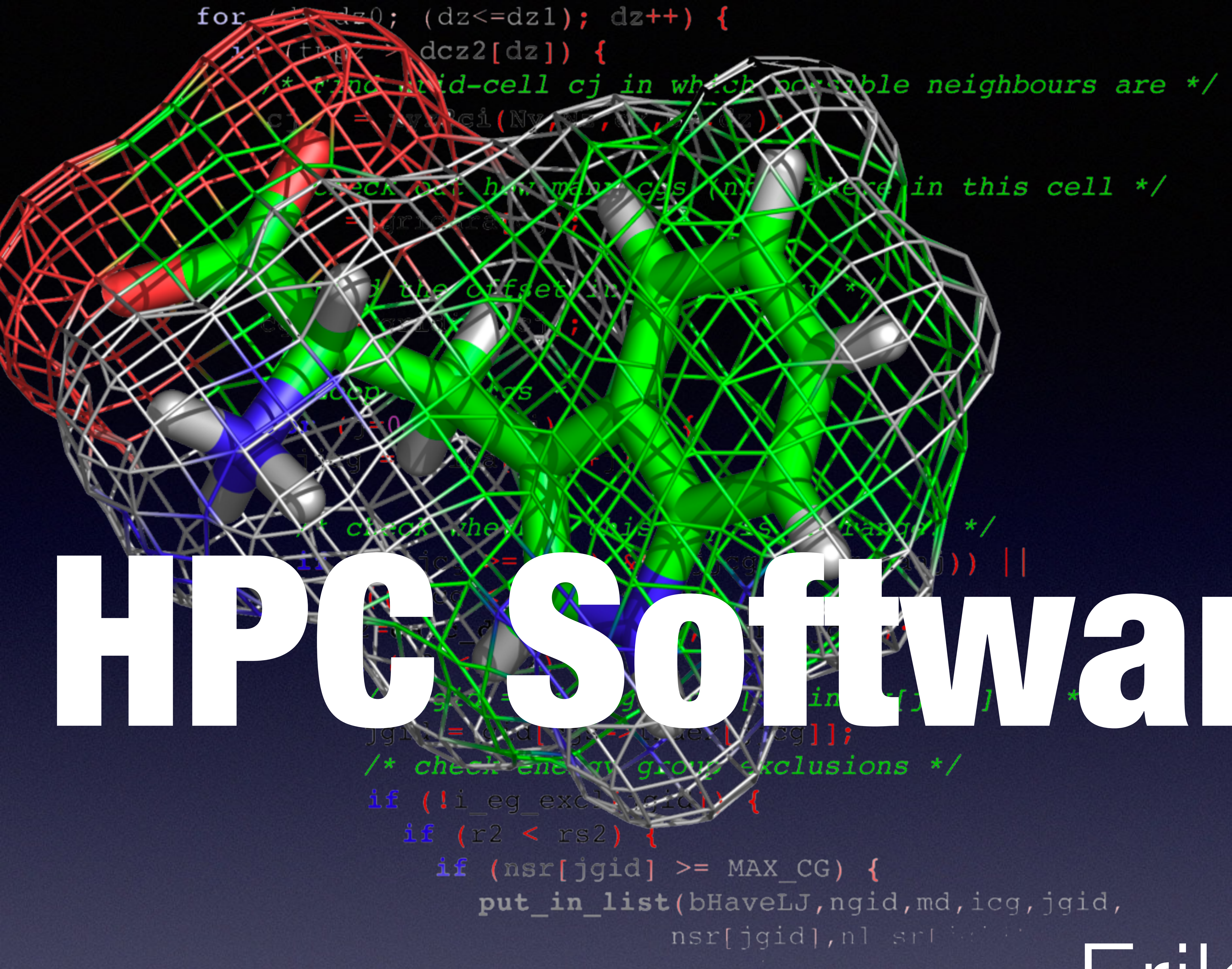


HPC Software Engineering

Erik Lindahl

XSEDE/PRACE/RIKEN/SciNet HPC Summer School Virtual Toronto 2021

SciLifeLab STOCKHOLM

STOCKHOLMS UNIVERSITET

KTH VETENSKAP OCH KONST

Experiences from 25 years of GROMACS development

2011: Successful, but increasingly painful?

- Simulation *hardware* project, turned software
- Early development based on our own needs
- Turned GPL in 2001, LGPL in 2012
- Organic growth of development
  - Roughly 10-15 core developers
  - Another 15-20 active contributors
- Currently 3,076,420 lines of C++11 code ("C++11")
  - Over the years we have used Fortran, C, Assembly
- Lots of old code. Lots of new code. Lots of complicated (read: bad) code written by scientists

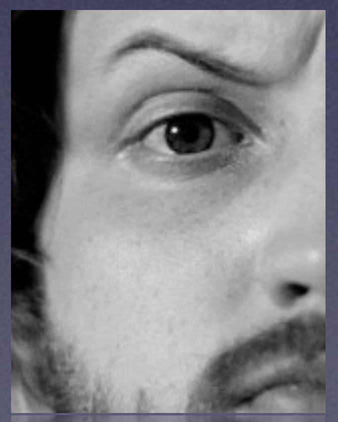
Source code repository: CVS

Build Chain: Automake/Autoconf/libtool

Bug Tracking: Bugzilla

Testing:



"The application of a systematic, disciplined, quantifiable approach to the development, operation and maintenance of software, and the study of these approaches, that is, the application of engineering to software."

Scientist mentality

- Trained in physics, chemistry, etc.
- Cares about their problem
- Cares about short-term deadlines (next paper!)
- New code = asset
- Writes more code than she reads

Software engineer mentality

- Trained in CS/software
- Cares about their code
- Cares about long-term maintenance (next machine)
- New code = liability
- Reads much more code than she writes

Without proper software engineering, you are taking on a technical debt that sooner or later will have to be repaid

"Technical Debt is a wonderful metaphor developed by Ward Cunningham to help us think about this problem. In this metaphor, doing things the quick and dirty way sets us up with a technical debt, which is similar to a financial debt. ***Like a financial debt, the technical debt incurs interest payments****, which come in the form of the extra effort that we have to do in future development because of the quick and dirty design choice.* ***We can choose to continue paying the interest, or we can pay down the principal by refactoring*** *the quick and dirty design into the better design. Although it costs to pay down the principal, we gain by reduced interest payments in the future."*

[Martin Fowler]

Why Open Source & Software Engineering Matter

The main point of open software is not cost, but that colleagues can check each other's code & assumptions and advance science by correcting flaws.

22 Aug 2018 in Research & Technology

DOI:10.1063/PT.6.1.20180822a

The war over supercooled water

How a hidden coding error fueled a seven-year dispute between two of condensed matter's top theorists.

Ashley G. Smart

8 COMMENTS | 4.6K SHARES

‹ PREV | NEXT ›

"One of the real travesties is that there's no way you could have reproduced [the Berkeley team's] algorithm—the way they had implemented their code—from reading their paper. If this had been disclosed, this saga might not have gone on for seven years."

Physics Today, Aug 22, 2018: *Recollection of Chandler/Limmer vs. Debenedetti 7-year fight over supercooled water; turned out to be algorithm implementation issue in code authors resisted sharing.*


PLAN
BUILD
CONTINUOUS
INTEGRATION
DEPLOY
OPERATE
CONTINUOUS FEEDBACK
REAL-TIME COMMUNICATION
REAL-TIME COMMUNICATION
draw.io
asana
box
PivotalTracker
Lucidchart
planio
flowdock
Google Drive
Google Docs
Wrike
smartsheet
Office
gliffy
Basecamp
Trello
Dropbox
Microsoft Teams
(x) matters
zoominfo.
CODE CLIMATE
git
GitLab
JFrog
npm
kubernetes
docker
Visual Studio Team Foundation Server
TeamCity
CODESHIP
Jenkins
buddybuild
shippable
circleci
Travis CI
split
HashiCorp
amazon web services
ANSIBLE
Google Cloud Platform
Microsoft Azure
XL RELEASE
heroku
XL DEPLOY
puppet
CHEF
rackspace
OpsGenie
VictorOps
pagerduty
BlueJeans
slack
New Relic.
snyk
bugsnag
Nagios
splunk
LOGGLY
ZABBIX
RAYGUN
SENTRY
DATADOG
bigpanda
dynatrace
Rollbar
APPDYNAMICS
[:] SourceClear
LogicMonitor
salesforce
SUBVERSION
GitHub
Se
Sonatype
MOCHA
FitNesse
node
ZEPHYR
freshdesk
INTERCOM
runnable
ZOHO CRM
BrowserStack
QMETRY
SAUCELABS
servicenow
zendesk
TestFairy
Jasmine
cucumber
Instabug
QASymphony
OMNIDESK

https://github.com/IHPCSS/software-engineering

Please DO steal this and use it as a template for your own project!

When that is not advanced enough:

https://gitlab.com/gromacs/gromacs

… or browse GitHub/GitLab for a huge number of other projects.
Check that your open souce license matches, then copy-refine-improve.

What changed in our code **last week**?

49 commits
183 files changed
5,375 line insertions
3,320 line deletions

What changed in our code **since Jan 1**?

671 commits
5,752 files changed
157,177 line insertions
1,622,410 line deletions*

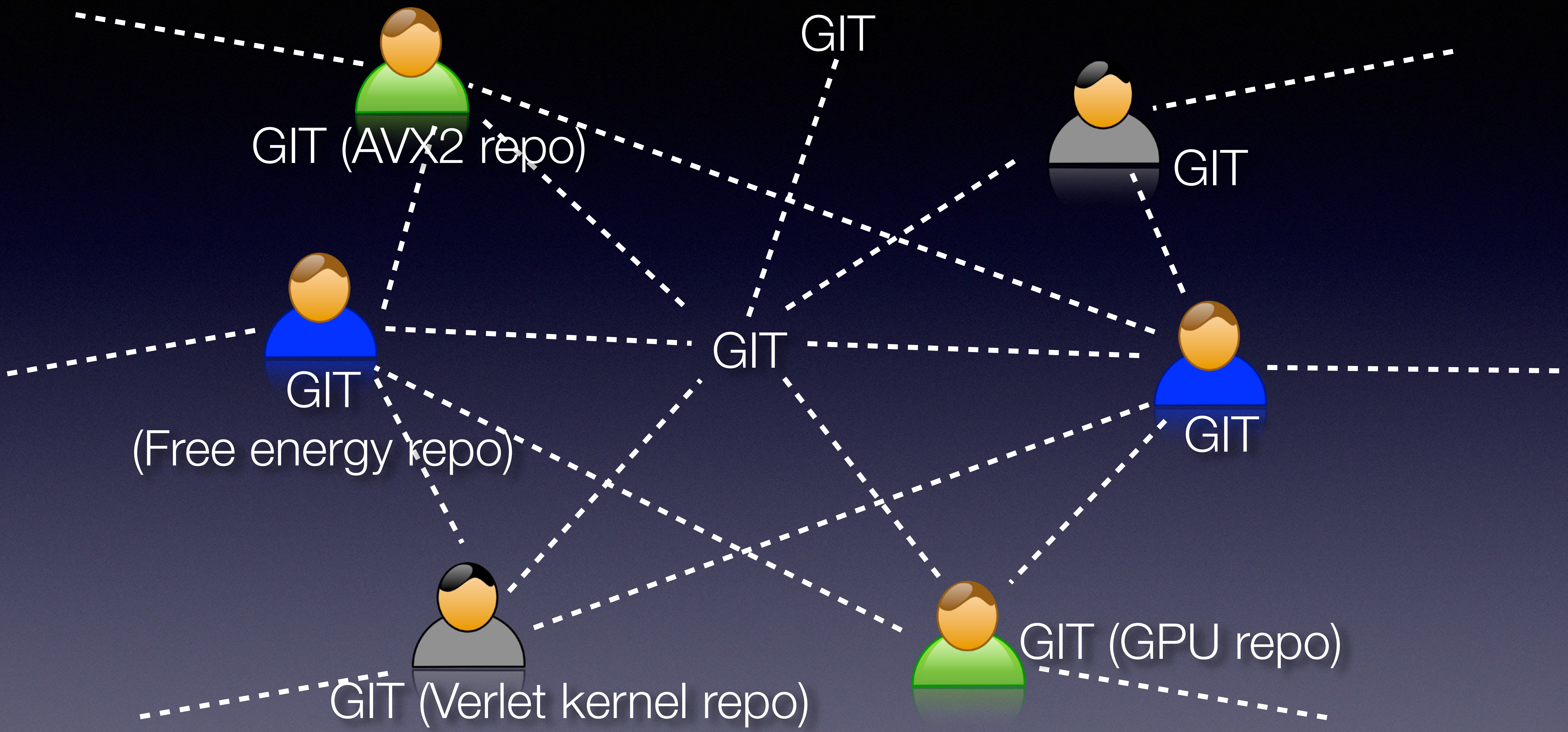
**Temporarily removed a bunch of kernels*

How would you start debugging if the new version crashes?

You have probably all seen this: Your program worked last week, but now there is something wrong

What if it crashes with “-O3”, but when you try to debug it works fine?

Better source control: GIT



Use gitlab.com, or github.com!

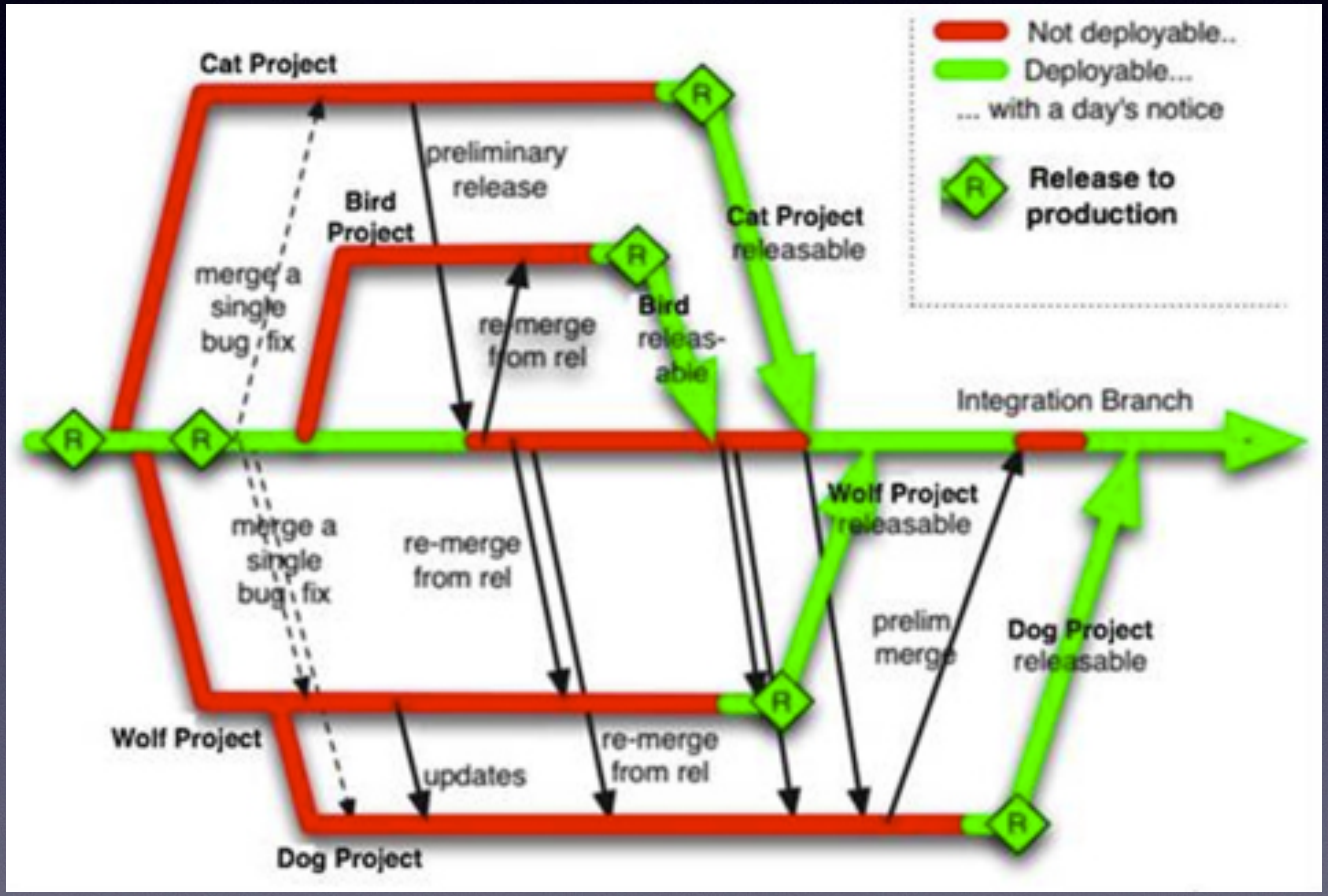
New example software engineering project for you to play around with:
https://github.com/IHPCSS/software-engineering

What git will give you

- Handles multiple developers beautifully
- Handles multiple feature branches in parallel with a stable production-quality one
- Develop based on features, not source files
- Pull/push patches between branches
- Revert a specific stupid thing I did 6 months ago, without changing subsequent patches
- Bisect changes to find which one of (say) 1,500 patches caused a bug

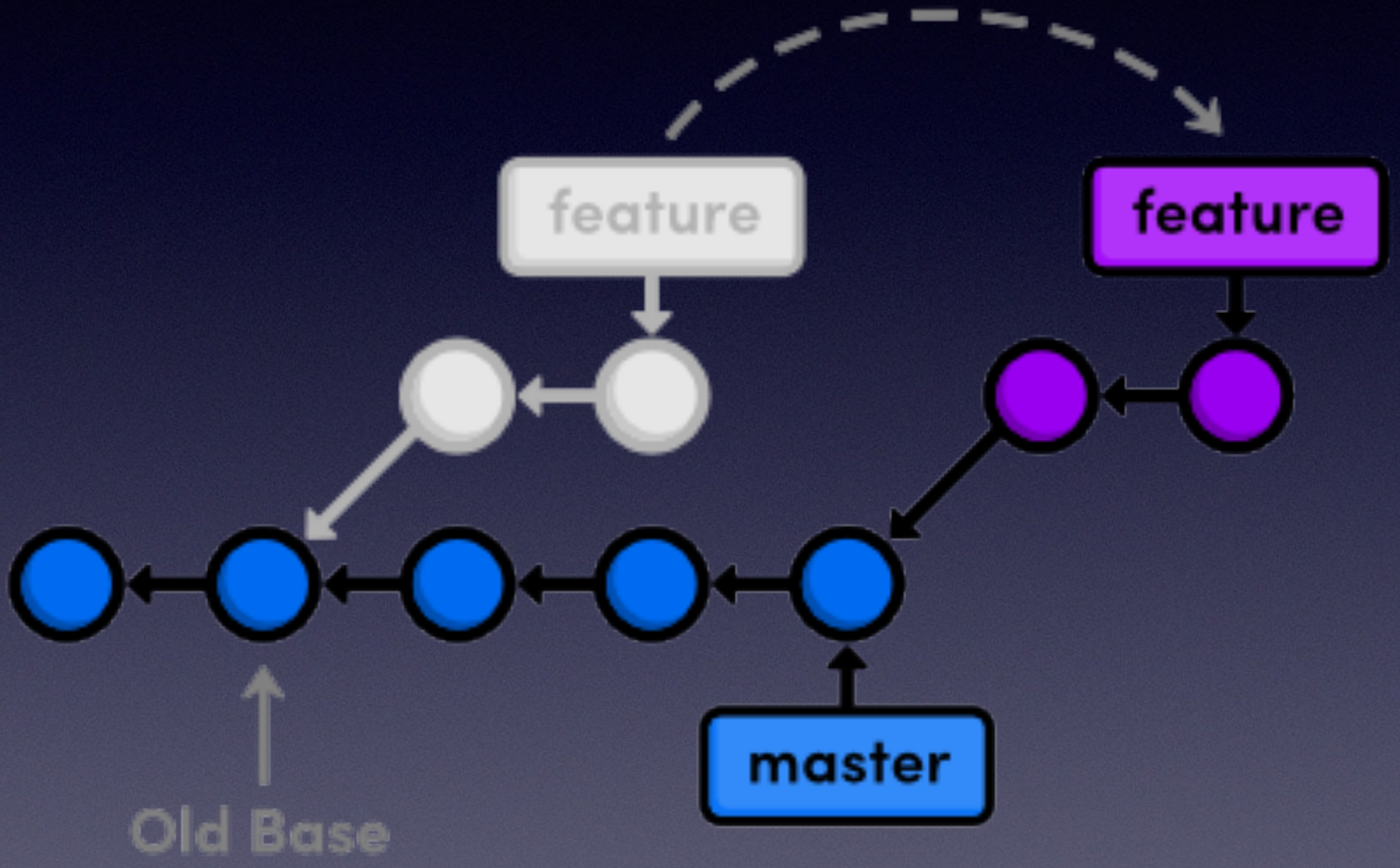
Drawback: Git is a VERY powerful tool, but the advanced features can be difficult to understand

One Possible Git Workflow: Multiple branches & merging



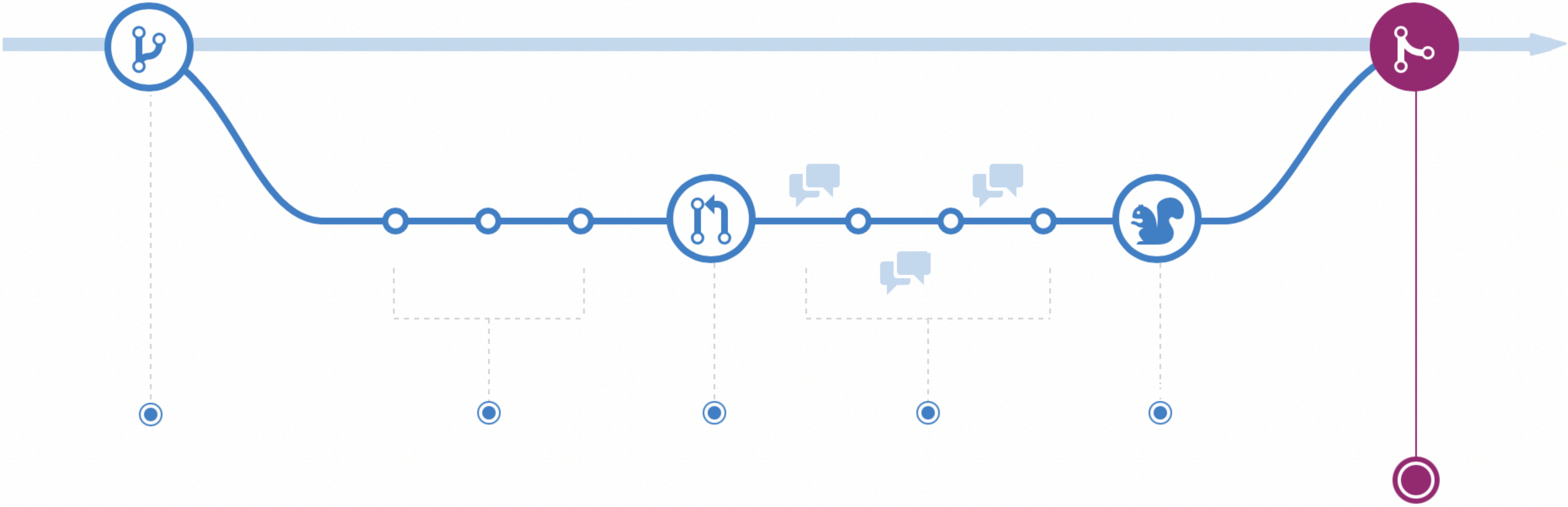
- Each feature is a new branch
- Think of the hybrid challenge:
  - Common base is the scalar version
  - Feature 1: MPI
  - Feature 2: OpenMP
  - Feature 3: OpenACC
- Imagine that these features have now been developed/improved over 3 months.
- **Each feature branch works great, but major pains when you need to combine them & release**

Better approach: (Constant) rebasing



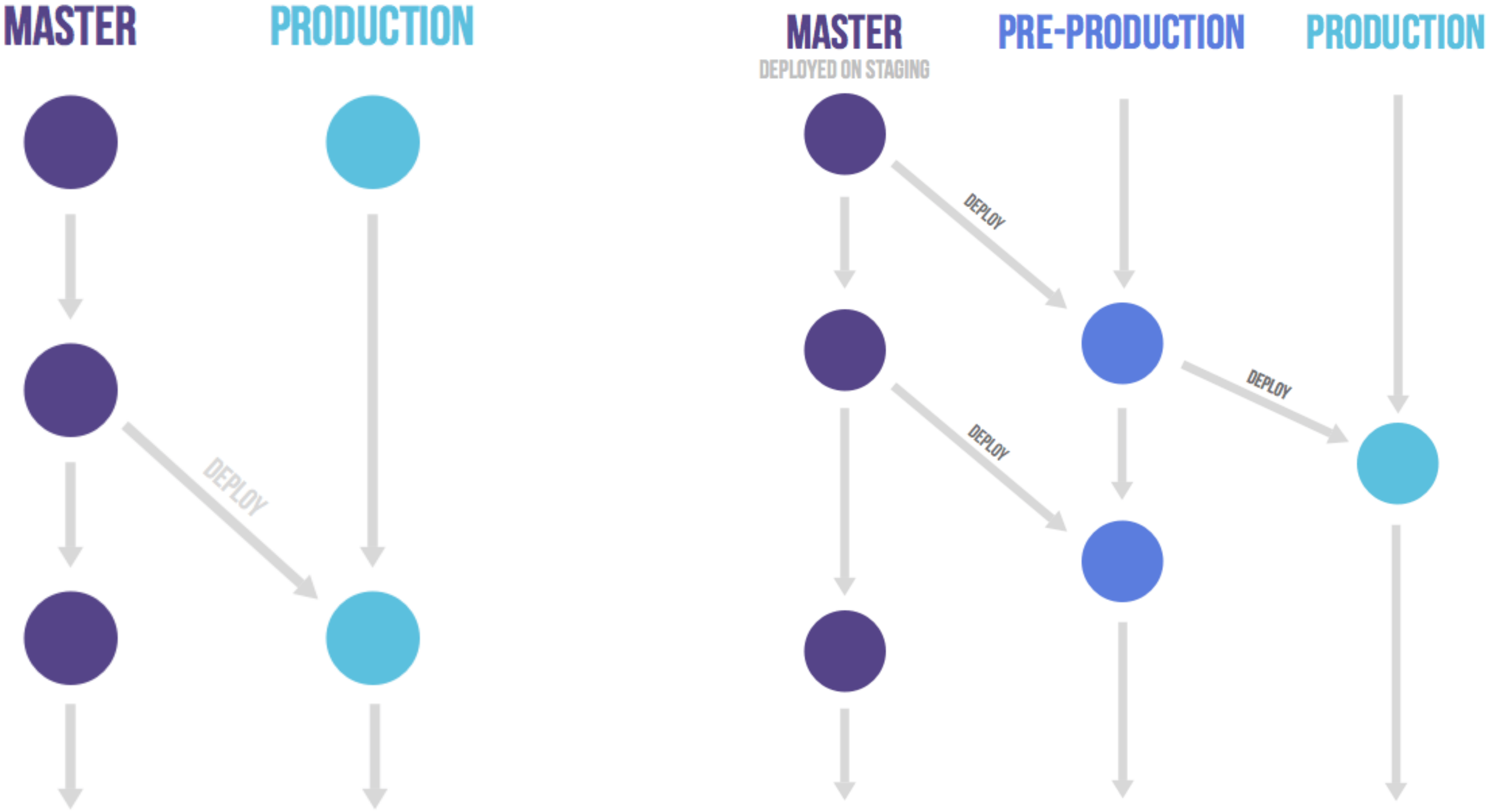
- Think of feature commits as work-in-progress (e.g. on my laptop) that have not yet made it into our common master branch
- A large project like GROMACS can have hundreds of such work-in-progress commits; each of them is independent of all other feature commits
- **When one feature commit is ready & merged into master, the other features should rebase to instead be a difference relative to the updated master state**
- You *can* continue to work with the old base while developing, but before committing your feature it has to be rebased
- Advantage: Clean changes, rapid deployment

GitHub Flow:



Merge

GitLab Flow:


MASTER
PRODUCTION
DEPLOY
MASTER
DEPLOYED ON STAGING
PRE-PRODUCTION
PRODUCTION
DEPLOY
DEPLOY
DEPLOY

Good git commits are

- Small (think 10-100 lines, not 1000)
- Decomposed as far as possible
- Limited to address a single issue
- Well documented
- Tested to work

Have a look at some of the commit in the IHPCSS-laplace repo!

Initial CMake support, change to use C++ compiler — Browse files

```
Trivial CMake file added that builds a binary
from the source in the current directory, and the
source has been renamed to use the C++ compiler.
```

master

Erik Lindahl committed on Jul 9, 2018 — 1 parent 962bdd9 — commit 45d2c4833b8ccd9477df6581e1131b73e8e16d79

Showing **2 changed files** with **14 additions** and **0 deletions**. Unified Split

14 CMakeLists.txt

```
@@ -0,0 +1,14 @@
+ #
+ # Example CMake file for John Urbanic's laplace
+ # solver, now using a C++ compiler.
+ #
+
+ # Make sure we don't have a pre-historic CMake version
+ cmake_minimum_required(VERSION 2.8)
+
+ # We want a name
+ project(Laplace)
+
+ # Build the binary 'laplace' from the source file (now renamed to be a c++ source)
+ add_executable(laplace laplace_serial.cpp)
+
```

0 laplace_serial.c → laplace_serial.cpp

File renamed without changes.

This type of commit will also be close to trivial to rebase!

Is your code portable?

Does your code compile on
windows (MSVC)?
PGI (Now NVIDIA) Compilers? Pathscale?
Blue Gene?
Fugaku (Fujitsu compilers)?
ARM? AArch64? With the ARM compiler? Clang? Gcc?
PowerPC (big endian)?
Google NativeClient?
RISC V?

OpenPower (little endian?)

What is a build chain?

The typical user progression:

- Issue compiler commands manually
- Start using Makefiles, edit Makefiles, give up
- Automate the *generation* of Makefiles

Configuration

- “Where is the X11 library? MKL? LibXML?”
- “Is this the buggy version 3.3.7 of the FFTW library?”
- “Is the Intel Math Kernel Library installed?”
- “Do we use that buggy gcc version?”
- “Does this compiler understand Xeon Phi AVX512?”
- “Which flags should be used to enable C++11 for this compiler?”
- “Is this a big or small endian system?”
- “Is a long integer 4 or 8 bytes on this host?”
- “How do we build a shared library here?”
- “How do we turn on OpenMP? OpenACC?”
- “What library should I link with to have gettimeofday() available?”
- “What C backend compiler is used with CUDA-8.0?”
- “What underscore naming standard does this Fortran compiler use?”
- “Is Doxygen available? Sphinx? Dot?”

CMake: Cross-platform replacement for Autoconf, Automake, Libtool
(instead of ./configure; make; make install)

GROMACS has ~100 CMake tests for features/bugs/libraries/compilers

- CheckCCompilerFlag.cmake
- CheckCXXCompilerFlag.cmake
- cmake_uninstall.cmake.in
- FindEXTRAE.cmake
- FindFFTW.cmake
- FindVMD.cmake
- gmxBuildTypeProfile.cmake
- gmxBuildTypeReference.cmake
- gmxBuildTypeReleaseWithAssert.cmake
- gmxBuildTypeThreadSanitizer.cmake
- gmxCFlags.cmake
- gmxDetectClang30.cmake
- gmxDetectGpu.cmake
- gmxDetectSimd.cmake
- gmxDetectTargetArchitecture.cmake
- gmxFindFlagsForSource.cmake
- gmxGCC44O3BugWorkaround.cmake
- gmxGenerateVersionInfo.cmake
- gmxManageBlueGene.cmake
- gmxManageFFTLibraries.cmake
- gmxManageGPU.cmake
- gmxManageLinearAlgebraLibraries.cmake
- gmxManageMPI.cmake
- gmxManageNvccConfig.cmake
- gmxManageOpenMP.cmake
- gmxManageSharedLibraries.cmake
- gmxManageSuffixes.cmake
- gmxOptionUtilities.cmake
- gmxSetBuildInformation.cmake
- gmxTestAVXMaskload.cmake
- gmxTestCatamount.cmake
- gmxTestCompilerProblems.cmake
- gmxTestCXX11.cmake
- gmxTestdlopen.cmake
- gmxTestFloatFormat.cmake
- gmxTestInlineASM.cmake
- gmxTestIsfinite.cmake
- gmxTestLargeFiles.cmake
- gmxTestLibXml2.cmake
- gmxTestMPI_IN_PLACE.cmake

```
MACRO(GMX_TEST_AVX_GCC_MASKLOAD_BUG VARIABLE AVX_CFLAGS)
    IF(NOT DEFINED ${VARIABLE})
        MESSAGE(STATUS "Checking for gcc AVX maskload bug")
        # some compilers like clang accept both cases,
        # so first try a normal compile to avoid flagging those as buggy.
        TRY_COMPILE(${VARIABLE}_COMPILEOK "${CMAKE_BINARY_DIR}"
                    "${CMAKE_SOURCE_DIR}/cmake/TestAVXMaskload.c"
                    COMPILE_DEFINITIONS "${AVX_CFLAGS}" )
        IF(${VARIABLE}_COMPILEOK)
            SET(${VARIABLE} 0 CACHE INTERNAL "Work around GCC bug in AVX maskload argument" FORCE)
            MESSAGE(STATUS "Checking for gcc AVX maskload bug - not present")
        ELSE()
            TRY_COMPILE(${VARIABLE}_COMPILEOK "${CMAKE_BINARY_DIR}"
                        "${CMAKE_SOURCE_DIR}/cmake/TestAVXMaskload.c"
                         COMPILE_DEFINITIONS "${AVX_CFLAGS} -DGMX_SIMD_X86_AVX_GCC_MASKLOAD_BUG" )
            IF(${VARIABLE}_COMPILEOK)
                SET(${VARIABLE} 1 CACHE INTERNAL "Work around GCC bug in AVX maskload argument" FORCE)
                MESSAGE(STATUS "Checking for gcc AVX maskload bug - found, will try to work around")
            ELSE()
                MESSAGE(WARNING "Cannot compile AVX code - assuming gcc AVX maskload bug not present." )
                MESSAGE(STATUS "Checking for gcc AVX maskload bug - not present")
            ENDIF()
        ENDIF()
    ENDIF()
ENDMACRO()
```

Optional components (FFT libs) and extensive regressiontests can be downloaded automatically

Generators: Makefiles, Eclipse, Xcode, VisualStudio, nmake, CodeBlocks, KDevelop3, etc.

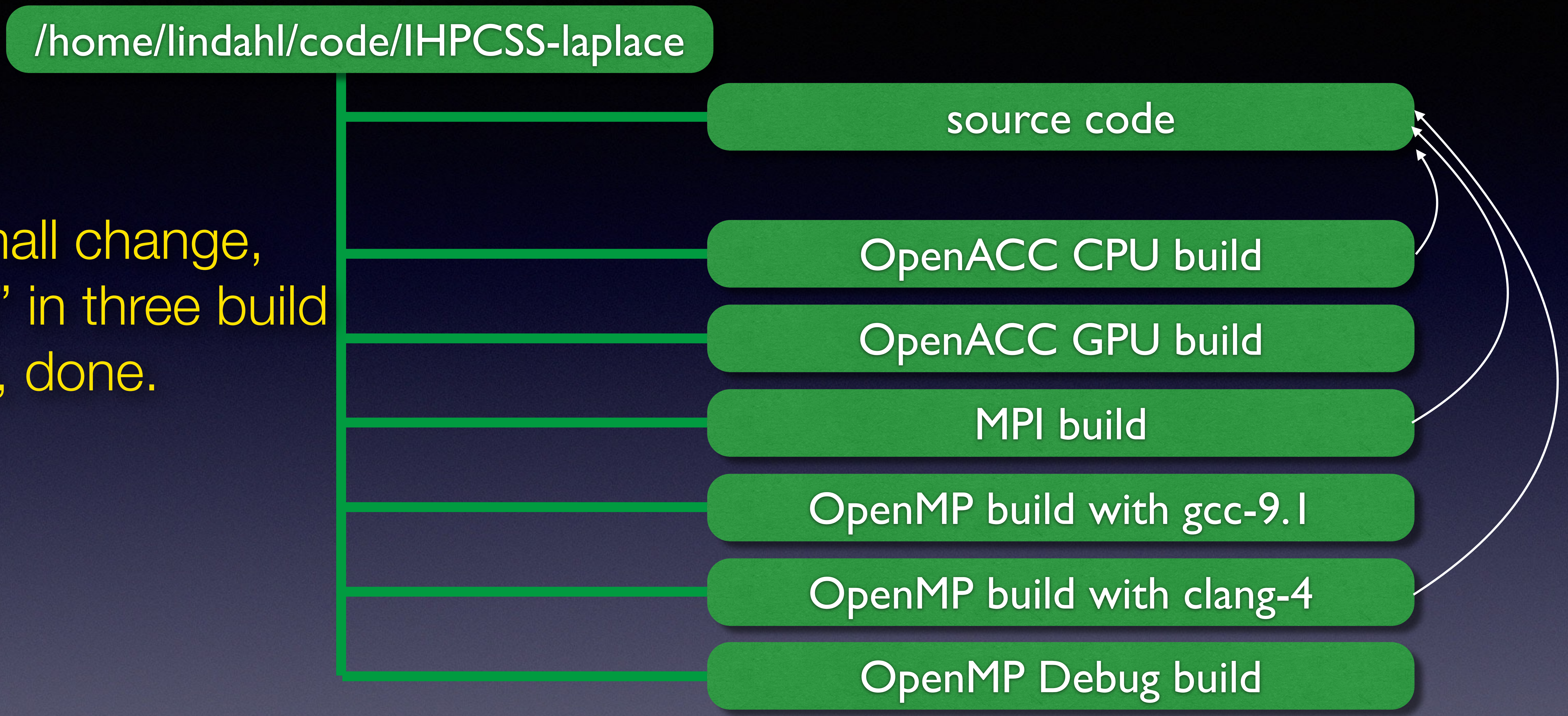
But don't start with GROMACS: Look at the CMakeLists.txt in the IHPCSS/software-engineering example: 75 lines and a few modules for complete detection of compilers, OpenMP, OpenACC, MPI, and everything else you'll see on the next few slides!

The complete CMakeLists.txt for the IHPCSS Laplace code

```
#
# Example CMake file for the IHPCSS Software Engineering
# project, based on John Urbanic's laplace
# solver ported to use a C++ compiler.
#

# Make sure we don't have a pre-historic CMake version
cmake_minimum_required(VERSION 3.0)
# Enable policy 0048 to allow setting version with the project command
set(CMP0048 NEW)

list(APPEND CMAKE_MODULE_PATH ${CMAKE_CURRENT_SOURCE_DIR}/cmake)

project(Software-engineering VERSION 0.2)
set(PROJECT_VERSION_STRING "${PROJECT_VERSION}")

set(CMAKE_LIBRARY_OUTPUT_DIRECTORY ${CMAKE_BINARY_DIR}/lib)
set(CMAKE_ARCHIVE_OUTPUT_DIRECTORY ${CMAKE_BINARY_DIR}/lib)
set(CMAKE_RUNTIME_OUTPUT_DIRECTORY ${CMAKE_BINARY_DIR}/bin)

set(CMAKE_BUILD_TYPE "Release" CACHE STRING "Choose type of build, options are: Debug, MinSizeRel, Release, RelWithDebInfo"

option(BUILD_TESTS "Enable unit test building" ON)

option(MPI     "Enable MPI compiler support" OFF)
option(OPENMP  "Enable OpenMP compiler support" OFF)
option(OPENACC "Enable OpenACC compiler support" OFF)

# Use GNUInstallDirs to set paths on multiarch systems.
include(GNUInstallDirs)

# Add the MPI/OpenMP/OpenACC compiler flags before other tests,
# since this might change the behavior on some platforms
```

```
if(MPI)
    find_package(MPI)
    if(MPI_CXX_FOUND)
        set(CMAKE_CXX_FLAGS "${CMAKE_CXX_FLAGS} ${MPI_CXX_COMPILE_FLAGS}")
        include_directories(${MPI_CXX_INCLUDE_PATH})
        set(CMAKE_EXE_LINKER_FLAGS ${MPI_CXX_LINK_FLAGS})
        set(CMAKE_SHARED_LINKER_FLAGS ${MPI_CXX_LINK_FLAGS})
        list(APPEND EXTRA_LIBRARIES ${MPI_CXX_LIBRARIES})
        set(HAVE_MPI TRUE)
    else()
        message(ERROR "MPI support requested, but no compiler support found.")
    endif()
endif()

if(OPENMP)
    find_package(OpenMP)
    if(OPENMP_FOUND)
        set(CMAKE_CXX_FLAGS "${CMAKE_CXX_FLAGS} ${OpenMP_CXX_FLAGS}")
        set(HAVE_OPENMP TRUE)
    else()
        message(ERROR "OpenMP support requested, but no compiler support found.")
    endif()
endif()

if(OPENACC)
    find_package(OpenACC)
    if(OPENACC_CXX_FOUND)
        set(CMAKE_CXX_FLAGS "${CMAKE_CXX_FLAGS} ${OpenACC_CXX_FLAGS}")
        set(HAVE_OPENACC TRUE)
        set(OPENACC_VERSION OpenACC_CXX_VERSION)
    else()
        message(ERROR "OpenACC support requested, but no compiler support found.")
    endif()
endif()


# Test and add some extra compiler flags
include(CompilerFlags)

add_subdirectory(src)
add_subdirectory(docs)
```

Out-of-source builds



Make a small change, run “make” in three build directories, done.

```
$ ~> mkdir build-openacc
$ ~> cd build-openacc
$ build-openacc> cmake -DOPENACC=ON ../path/to/source/directory
```

Living with your code for ~~years~~ decades:

Documentation

Direct source code documentation should stay in the source!

Doxygen

Generate documentation from source code

Doxygen is the de facto standard tool for generating documentation from annotated C++ sources, but it also supports other popular programming languages such as C, Objective-C, C#, PHP, Java, Python, IDL (Corba, Microsoft, and UNO/OpenOffice flavors), Fortran, VHDL, Tcl, and to some extent D.

Doxygen can help you in three ways:

1. It can generate an on-line documentation browser (in HTML) and/or an off-line reference manual (in LaTeX) from a set of documented source files. There is also support for generating output in RTF (MS-Word), PostScript, hyperlinked PDF, compressed HTML, and Unix man pages. The documentation is extracted directly from the sources, which makes it much easier to keep the documentation consistent with the source code.
2. You can configure doxygen to extract the code structure from undocumented source files. This is very useful to quickly find your way in large source distributions. Doxygen can also visualize the relations between the various elements by means of include dependency graphs, inheritance diagrams, and collaboration diagrams, which are all generated automatically.
3. You can also use doxygen for creating normal documentation (as I did for the doxygen user manual and web-site).

Doxygen is developed under Mac OS X and Linux, but is set-up to be highly portable. As a result, it runs on most other Unix flavors as well. Furthermore, executables for Windows are available.

Doxygen example - our random module:

```
/*! \brief ThreeFry2x64 random engine with 20 iteractions.
 *
 *  \tparam internalCounterBits, default 64.
 *
 *  This class provides very high quality random numbers that pass all
 *  BigCrush tests, it works with two 64-bit values each for keys and
 *  counters, and is most  efficient when we only need a few random values
 *  before restarting the counters with new values.
 */
template<unsigned int internalCounterBits = 64>
class ThreeFry2x64 : public ThreeFry2x64General<20, internalCounterBits>
{
    public:
        /*! \brief Construct ThreeFry random engine with 2x64 key values, 20 rounds.
         *
         *  \param key0   Random seed in the form of a 64-bit unsigned value.
         *  \param domain Random domain. This is used to guarantee that different
         *                applications of a random engine inside the code get different
         *                streams of random numbers, without requiring the user
         *                to provide lots of random seeds. Pick a value from the
         *                RandomDomain class, or RandomDomain::Other if it is
         *                not important. In the latter case you might want to use
         *                \ref gmx::DefaultRandomEngine instead.
         *
         *  \note The random domain is really another 64-bit seed value.
         *
         *  \throws InternalError if the high bits needed to encode the number of counter
         *          bits are nonzero.
         */
        ThreeFry2x64(uint64_t key0 = 0, RandomDomain domain = RandomDomain::Other) : ThreeFry2x64General<20, internal

        /*! \brief Construct random engine from 2x64-bit unsigned integers, 20 rounds
         *
         *  This constructor assigns the raw 128 bit key data from unsigned integers.
         *  It is meant for the case when you want full control over the key,
         *  for instance to compare with reference values of the ThreeFry
         *  function during testing.
         *
         *  \param key0   First word of key/random seed.
         *  \param key1   Second word of key/random seed.
         *
         *  \throws InternalError if the high bits needed to encode the number of counter
         *          bits are nonzero. To test arbitrary values, use 0 internal counter bits.
         */
        ThreeFry2x64(uint64_t key0, uint64_t key1) : ThreeFry2x64General<20, internalCounterBits>(key0, key1) {}
};
```

Gromacs 2019

Main Page | Modules | Other Docs | Namespaces | Classes | Files | Examples

Class List | Class Index | Class Hierarchy | Class Members

gmx › ThreeFry2x64

gmx::ThreeFry2x64< internalCounterBits > Class Template Reference

`#include <gromacs/random/threefry.h>`

▸ Inheritance diagram for gmx::ThreeFry2x64< internalCounterBits >:

▸ Collaboration diagram for gmx::ThreeFry2x64< internalCounterBits >:

Description

template<unsigned int internalCounterBits = 64>
class gmx::ThreeFry2x64< internalCounterBits >

ThreeFry2x64 random engine with 20 iteractions.

Template Parameters

internalCounterBits,default 64.

This class provides very high quality random numbers that pass all BigCrush tests, it works with two 64-bit values each for keys and co

Public Member Functions

ThreeFry2x64 (uint64_t key0=0, RandomDomain domain=RandomDomain::Other)
Construct ThreeFry random engine with 2x64 key values, 20 rounds. More...

ThreeFry2x64 (uint64_t key0, uint64_t key1)
Construct random engine from 2x64-bit unsigned integers, 20 rounds. More...

▸ **Public Member Functions inherited from gmx::ThreeFry2x64General< 20, internalCounterBits >**

Additional Inherited Members

▸ **Public Types inherited from gmx::ThreeFry2x64General< 20, internalCounterBits >**

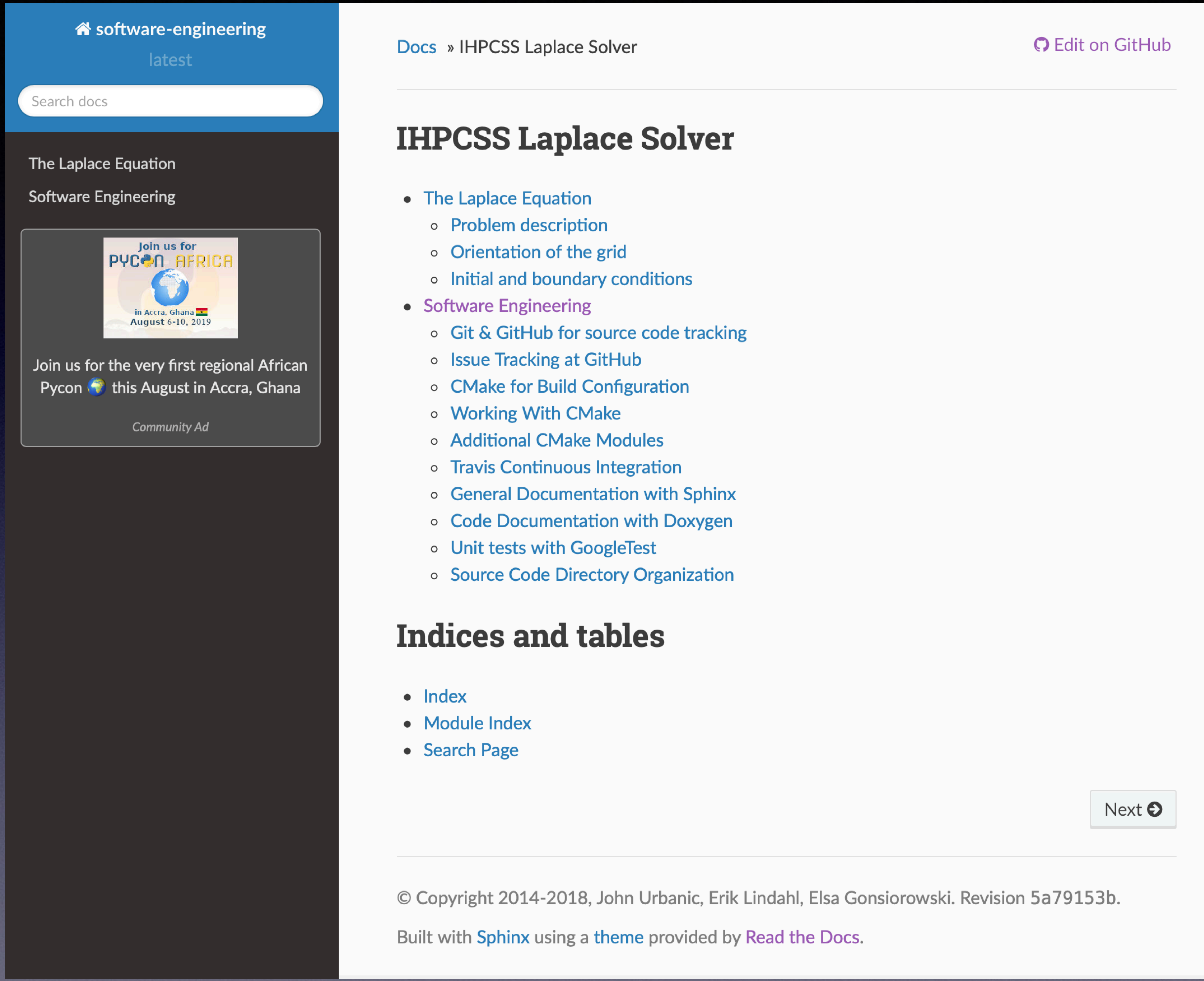
▸ **Static Public Member Functions inherited from gmx::ThreeFry2x64General< 20, internalCounterBits >**

The best comments don't explain what your code does, they explain WHY you do it this way!

The IHPCSS/software-engineering example comes with full Doxygen integration - but I have not yet had time to document the code! CMake finds "doxygen" automatically so you can do "make doxygen"

High level non-source-code documentation: SPHINX (from Python)

```
.. _laplace_equation:

The Laplace Equation
--------------------

Problem description
^^^^^^^^^^^^^^^^^^^

The Laplace equation is one of the most common in physics and
describes a large number of phenomena, including heat transfer.

This project is a simple example of how to implement a trivial
Laplace solver, and in particular how to extend it with reasonable
software engineering practices including an automated build system,
documentation, and some other bells and whistles.

Laplace's equation is a special case of Poisson's equation, and
valid when there are no sources or sinks adding or removing heat
in the system:

.. math:: \Delta u(x,y) = \frac{\partial^2 u}{\partial x^2} + \frac{\partial^2 u}{\partial y^2} = 0

If we discretize this equation on a grid where each cell has side h,
the first index (i) corresponds to x and the second (j) to y,
and approximate the derivatives from finite differences, we get

.. math:: \left( u_{i,j-1} + u_{i,j+1} + u_{i-1,j} + u_{i+1,j} - 4 u_{i,j} \right) / h^2 = 0,

which we can simplify into (note how h disappears)

.. math:: u_{i,j} = 0.25 \left( u_{i,j-1} + u_{i,j+1} + u_{i-1,j} + u_{i+1,j} \right).

.. image:: /_static/grid_elements.png
   :scale: 50%

Since this has to hold for every element in the grid, we need to iterate over the
grid until the solution converges - and that is the task of this code. There are
actually significantly more efficient algorithms to accomplish this
(using e.g. over-relaxation), but since the point of this example is to illustrate
software optimization and HPC software engineering practices rather than algorithms
to best solve Laplace's equation we won't implement that since it would complicate
the code.

Orientation of the grid
^^^^^^^^^^^^^^^^^^^^^^^
```



Fully integrated into IHPCSS-laplace. Check out the docs folder, and if you have sphinx/latex installed you can type "make sphinx-html" or "make sphinx-pdf".

… and we have integrated it with readthedocs.org! Any time a new change is pushed to the gibhub repo, documentation is built automatically at http://software-engineering.readthedocs.org

Finding & Preventing Bugs

Modularization

- Avoid code inter-dependencies
- Have modules doing clearly separate tasks
- Have a clear (documented) API for each module
- Make sure all code is thread-safe!
- Strict code organization:
  - One directory per module, e.g. src/foo - with documentation for that module
  - The ‘bar’ class is declared in src/foo/bar.h, implemented in src/foo/bar.cpp
- Write unit tests, not only regression tests
  - Unit tests for ‘bar’ class are placed in src/foo/tests/bar.cpp
- Design-for-Testability (DFT):
  Write unit test first, then the code implementation

Controversial (?): Move to C++

Languages?

- “REAL PROGRAMMERS CAN WRITE FORTRAN IN ANY LANGUAGE”
- "C combines the flexibility and power of assembly language with the user-friendliness of assembly language."
- “C makes it easy to shoot yourself in the foot; C++ makes it harder, but when you do it blows your whole leg off.”
- The actual C++ nightmare: *You accidentally create a dozen instances of yourself and shoot them all in the foot. Providing emergency medical care is impossible since you can't tell which are bitwise copies and which are just pointing at others and saying, "That's me over there."*

C++ Core guidelines (Herb Sutter & Bjarne Stroustrup):
https://github.com/isocpp/CppCoreGuidelines/blob/master/CppCoreGuidelines.md

The Case for C++

Modern: Threads, atomics, etc. part of C++11

Very powerful library with containers, algorithms

Strongly typed language

Still a low-level language - you control data exactly

Modern C++ has gotten rid of pointers, memory errors

Templates avoid code duplication

Some very advanced parallelization libraries: Intel TBB

Rapidly developing language, large ISO committee

Parallel Standard Template Library (STL) in C++17

Negative: It is a VERY complex language to master

Example: If you have ever worked with mutex:es to make sure only one thread accesses a critical region, you have likely bumped into race conditions or deadlocks e.g. when you forget to release a mutex in complex code.

These errors are insanely difficult to debug, since it depends in dynamic timing events - when you run it in the debugger there won't be any error!

Definition:

```
class Lock {
public:
     explicit Lock(Mutex *pm)
     : mutexPtr(pm)
     { lock(*mutexPtr); }

     ~Lock() { unlock(*mutexPtr) };

private:
     Mutex *mutexPtr;
}
```

Usage in client code:

```
Mutex m;

…

{
     Lock ml(&m);
     …
}
```

One more problem: What happens if you copy that class? Then the first object to go out of scope will release the mutex, while the second thinks it's still locked (=bad)!

Easy to fix in C++11: Just use a reference-counted shared pointer.
Note: no change to the client code.

Definition:

```
class Lock {
public:
     explicit Lock(Mutex *pm)
     : mutexPtr(pm, unlock)
     { lock(mutexPtr.get()); }

     ~Lock() { unlock(*mutexPtr) };

private:
     std::shared_ptr<Mutex> mutexPtr;
}
```

(The line `~Lock() { unlock(*mutexPtr) };` is struck through on the slide.)

Usage in client code:

```
Mutex m;

…

{
     Lock ml(&m);
     …
}
```

Surprise: C++ can be (much) faster than FORTRAN or C!

C/FORTRAN

```
int
myFunc(obj_t obj, int choiceA, int choice B)
{
    for(int i=0;i<obj.N;i++)
    {
        if(choiceA==1)
        {
            if(choiceB==1)
            {
                kernelcode1;
            }
            else if(choiceB==2)
            {
                kernelcode2;
            }
        }
        else if(choiceA==2)
        {
            if(choiceB==1)
            {
                kernelcode3;
            }
            else if(choiceB==2)
            {
                kernelcode4;
            }
        }
    }
}
```

```
calling code in different translation unit:

myFunc(obj,2,3);
```

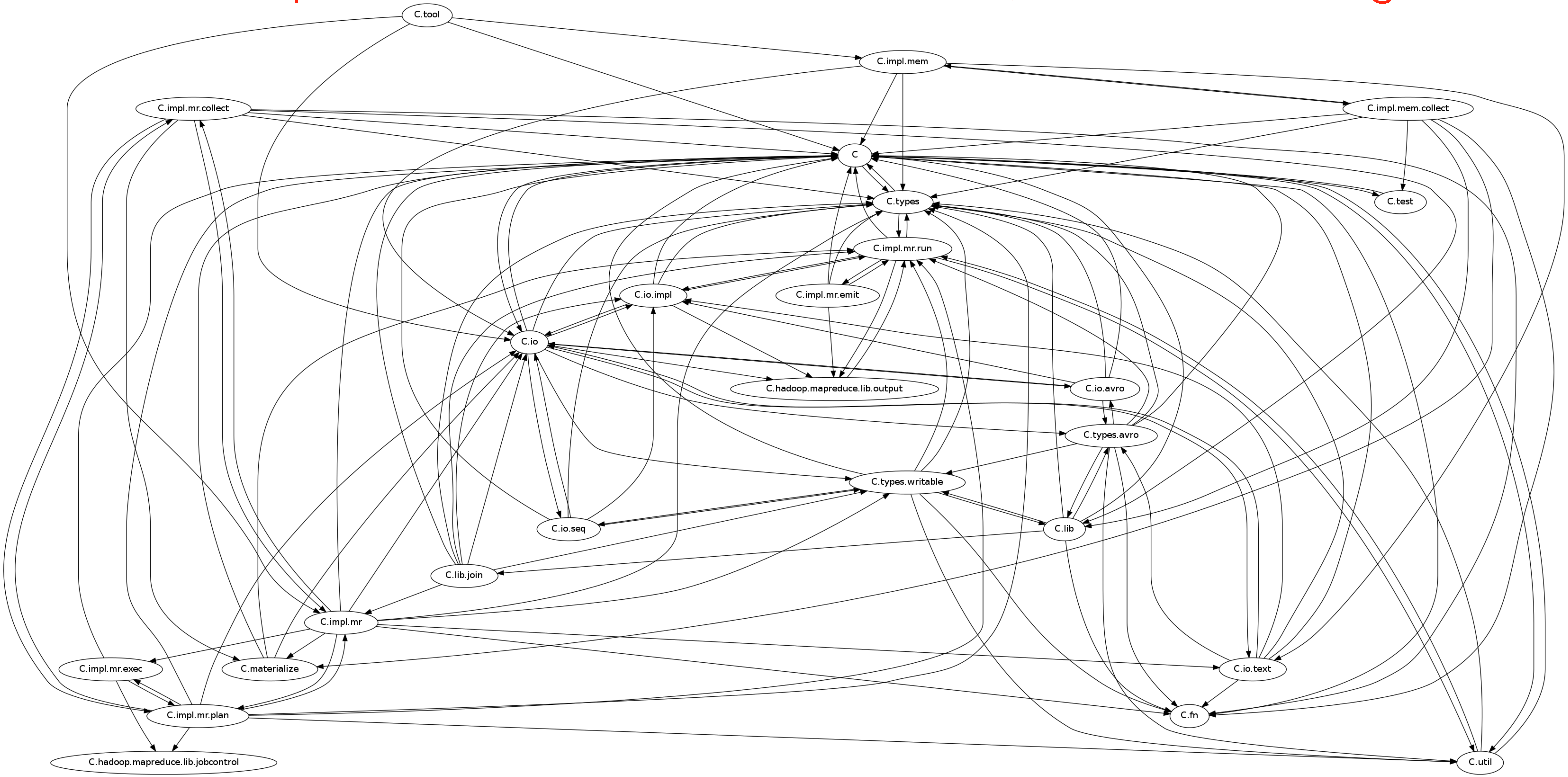
C++11

```
template <int choiceA, int choice B>
int
myFunc(obj_t obj)
{
    for(int i=0;i<obj.N;i++)
    {
        if(choiceA==1)
        {
            if(choiceB==1)
            {
                kernelcode1;
            }
            else if(choiceB==2)
            {
                kernelcode2;
            }
        }
        else if(choiceA==2)
        {
            if(choiceB==1)
            {
                kernelcode3;
            }
            else if(choiceB==2)
            {
                kernelcode4;
            }
        }
    }
}
```

```
calling code in different translation unit:

extern template int myFunc<2,3>(obj_t obj)
myFunc<2,3>(obj);
```

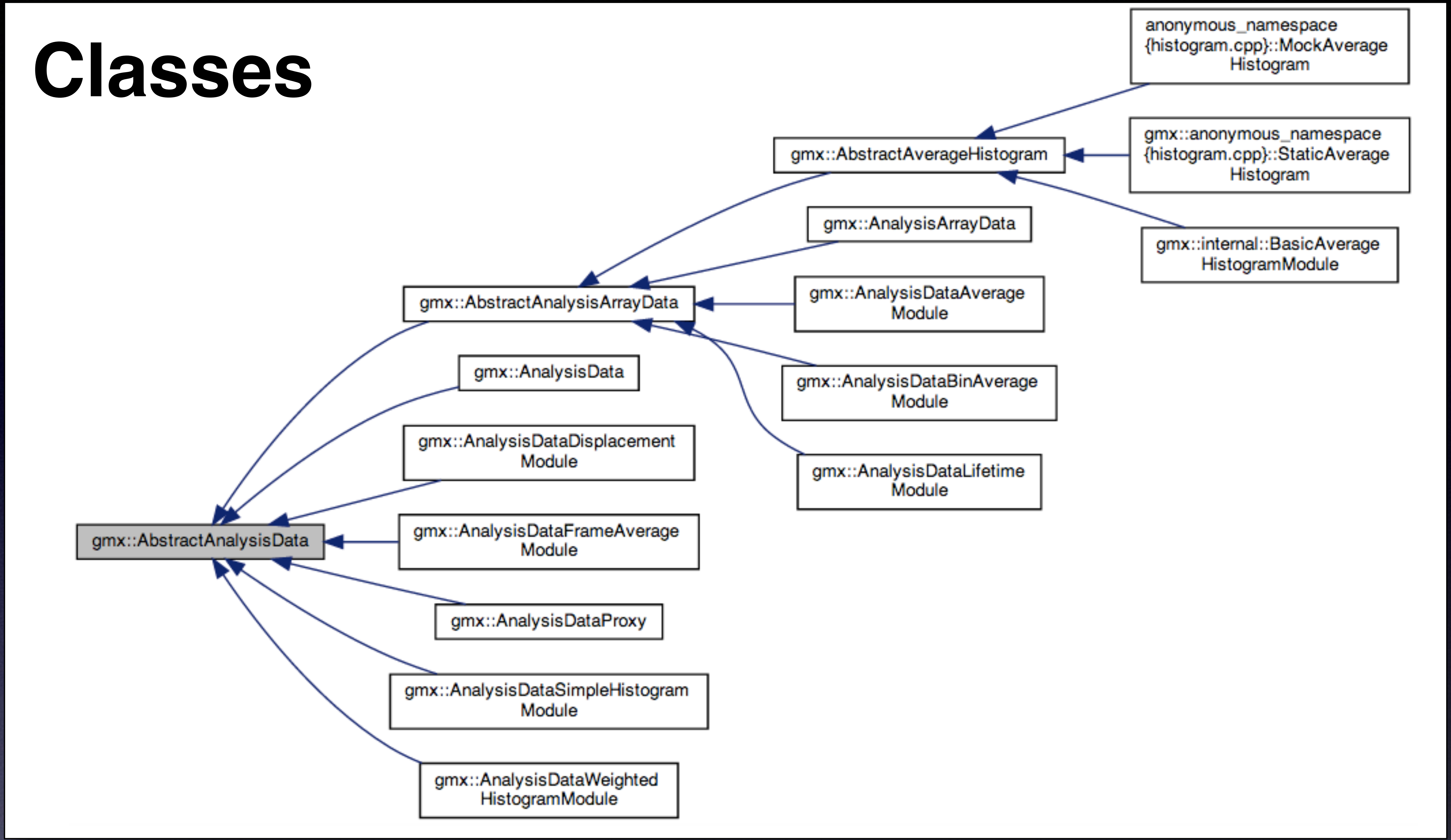
This C++ code will be fully expanded by the compiler. No conditionals present in the generated assembly code.

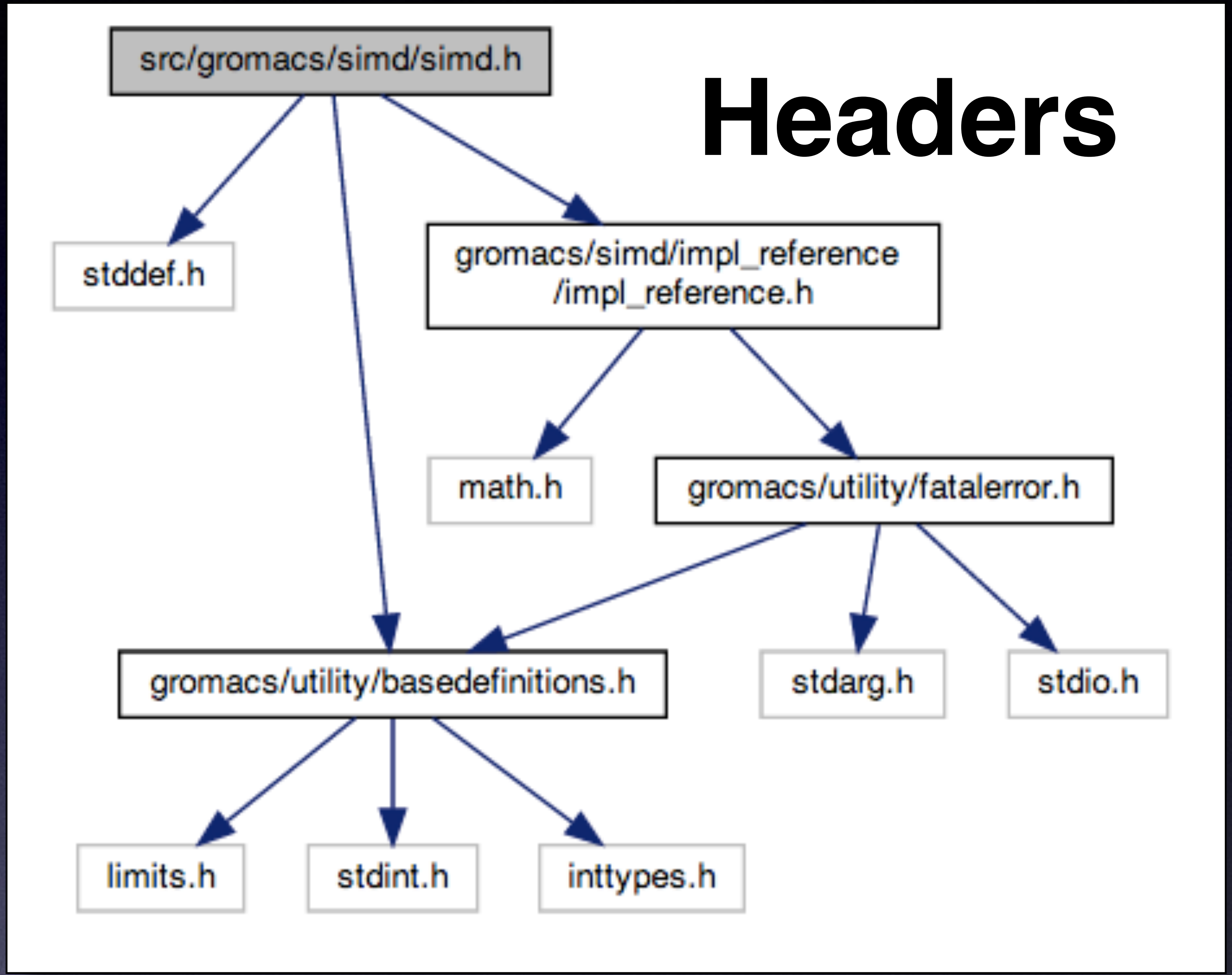
Circular dependencies are bad. If a test fails, where is the bug here?



"It has been discovered that C++ provides a remarkable facility for concealing the trivial details of a program - such as where its bugs are." (David Keppel)

Modularization: Just say 'no' to circular dependencies





This is hard, but Doxygen helps you detect it!

For our project (GROMACS), our code management system will not allow any developer to submit a file with a circular dependency.

Aggressive unit testing: "Trust, but verify"

googletest – Google C++ Testing Framework – Google Project Hosting

https code.google.com/p/googletest/

Umeå Univer...mi i Skolan | How to Do ...n — Medium | Computation...gy Council | Varför denna...naljapen.se | How to make...s Technica | Low Resolut...lar Domains | Examining p...Med

erik.lindahl@gmail.com ▼ | My favorites ▼

googletest

Google C++ Testing Framework

Project Home | Downloads | Wiki | Issues | Source

Summary | People

Project Information

Starred by 2339 users

Project feeds

Code license

New BSD License

Labels

Cplusplus, Testing, Framework, Tests, Unittests, Cpp, Google

Members

j...@google.com, zhanyong...@gmail.com, w...@google.com, ko...@google.com, sbe...@google.com, billydon...@google.com

8 committers

Featured

Downloads

gtest-1.7.0.zip

g+1 1k

Google's framework for writing C++ tests on a variety of platforms (Linux, Mac OS X, Windows, Cygwin, Windows CE, and Symbian). Based on the xUnit architecture. Supports automatic test discovery, a rich set of assertions, user-defined assertions, death tests, fatal and non-fatal failures, value- and type-parameterized tests, various options for running the tests, and XML test report generation.

Getting Started ¶

After downloading Google Test, unpack it, read the README file and the documentation wiki pages (listed on the right side of this front page).

Who Is Using Google Test?

In addition to many internal projects at Google, Google Test is also used by the following notable projects:

- The Chromium projects (behind the Chrome browser and Chrome OS)
- The LLVM compiler
- Protocol Buffers (Google's data interchange format)

If you know of a project that's using Google Test and want it to be listed here, please let `googletestframework@googlegroups.com` know.

Google Test-related open source projects

Google Test UI is test runner that runs your test binary, allows you to track its progress via a progress bar, and displays a list of test failures. Clicking on one shows failure text. Google Test UI is written in C#.

Example Gromacs unit tests: The idea is that you should test *everything*

```
TEST_P(FFTTest1D, Real)
{
    const int rx = GetParam();
    const int cx = (rx/2+1);
    ASSERT_LE(cx*2, static_cast<int>(sizeof(inputdata)/sizeof(real)));

    in_  = std::vector<real>(inputdata, inputdata+cx*2);
    out_ = std::vector<real>(cx*2);
    real* in  = &in_[0];
    real* out = &out_[0];

    gmx_fft_init_1d_real(&fft_, rx, flags_);

    gmx_fft_1d_real(fft_, GMX_FFT_REAL_TO_COMPLEX, in, out);
    checker_.checkSequenceArray(cx*2, out, "forward");
    gmx_fft_1d_real(fft_, GMX_FFT_COMPLEX_TO_REAL, in, out);
    checker_.checkSequenceArray(rx, out, "backward");
}
```

```
TEST_F(SimdFloatingpointTest, gmxSimdGetMantissaR)
{
    GMX_EXPECT_SIMD_REAL_EQ(setSimdRealFrom3R(1.219097320577810839026256,
                                              1.166738027848349235071623,
                                              1.168904015004464724825084), gmx_simd_get_mantissa_r(rSimd_Exp));
#if (defined GMX_SIMD_HAVE_DOUBLE) && (defined GMX_DOUBLE)
    GMX_EXPECT_SIMD_REAL_EQ(setSimdRealFrom3R(1.241261238952345623563251,
                                              1.047294723759123852359232,
                                              1.856066204750275957395734), gmx_simd_get_mantissa_r(rSimd_ExpDouble));
#endif
}

TEST_F(SimdFloatingpointTest, gmxSimdSetExponentR)
{
    gmx_simd_real_t x0 = setSimdRealFrom3R(0.5, 11.5, 99.5);
    gmx_simd_real_t x1 = setSimdRealFrom3R(-0.5, -11.5, -99.5);

    GMX_EXPECT_SIMD_REAL_EQ(setSimdRealFrom3R(pow(2.0, 60.0), pow(2.0, -41.0), pow(2.0, 54.0)),
                            gmx_simd_set_exponent_r(setSimdRealFrom3R(60.0, -41.0, 54.0)));
#if (defined GMX_SIMD_HAVE_DOUBLE) && (defined GMX_DOUBLE)
    GMX_EXPECT_SIMD_REAL_EQ(setSimdRealFrom3R(pow(2.0, 587.0), pow(2.0, -462.0), pow(2.0, 672.0)),
                            gmx_simd_set_exponent_r(setSimdRealFrom3R(587.0, -462.0, 672.0)));
#endif
    /* Rounding mode in gmx_simd_set_exponent_r() must be consistent with gmx_simd_round_r() */
    GMX_EXPECT_SIMD_REAL_EQ(gmx_simd_set_exponent_r(gmx_simd_round_r(x0)), gmx_simd_set_exponent_r(x0));
    GMX_EXPECT_SIMD_REAL_EQ(gmx_simd_set_exponent_r(gmx_simd_round_r(x1)), gmx_simd_set_exponent_r(x1));
}
```

Do you think it’s overkill to test that hardware rounding works? In March 2014, this very test caught that IBM Power7 VMX uses different rounding modes for SIMD and normal floating-point to integer conversions…

Spring 2018: Our unit tests caught that IBM had semi-silently had to change their *binary* ABI for Power8/9 since their compiler specifications partly violated the C++ standard. Fedora running all our unit tests caught it immediately, and a few hours later we had a workaround in the code.

Spring 2019: Our unit tests failed on the specific combination of gcc-7 and Intel AVX-512 hardware, but only with -O3 flags. Turned out to be a bug in the gcc-7 AVX-512 loop unrolling optimization.

Good unit tests should isolate bugs to *tiny* parts of your code
In C++, each method in a class should ideally have exhaustive unit tests

```
TEST(NormalDistributionTest, Output)
{
    gmx::test::TestReferenceData        data;
    gmx::test::TestReferenceChecker     checker(data.rootChecker());

    gmx::ThreeFry2x64<8>                rng(123456, gmx::RandomDomain::Other);
    gmx::NormalDistribution<real>       dist(2.0, 5.0);
    std::vector<real>                   result;

    for (int i = 0; i < 10; i++)
    {
        result.push_back(dist(rng));
    }
    checker.checkSequence(result.begin(), result.end(), "NormalDistribution");
}
```

Are *you* aware of the peculiarities of rounding differences depending on whether your CPU hardware uses fused multiply-add (FMA) vs. separate multiply & add?

Test that a simple call to a normal distribution random generator returns the expected 10 numbers.

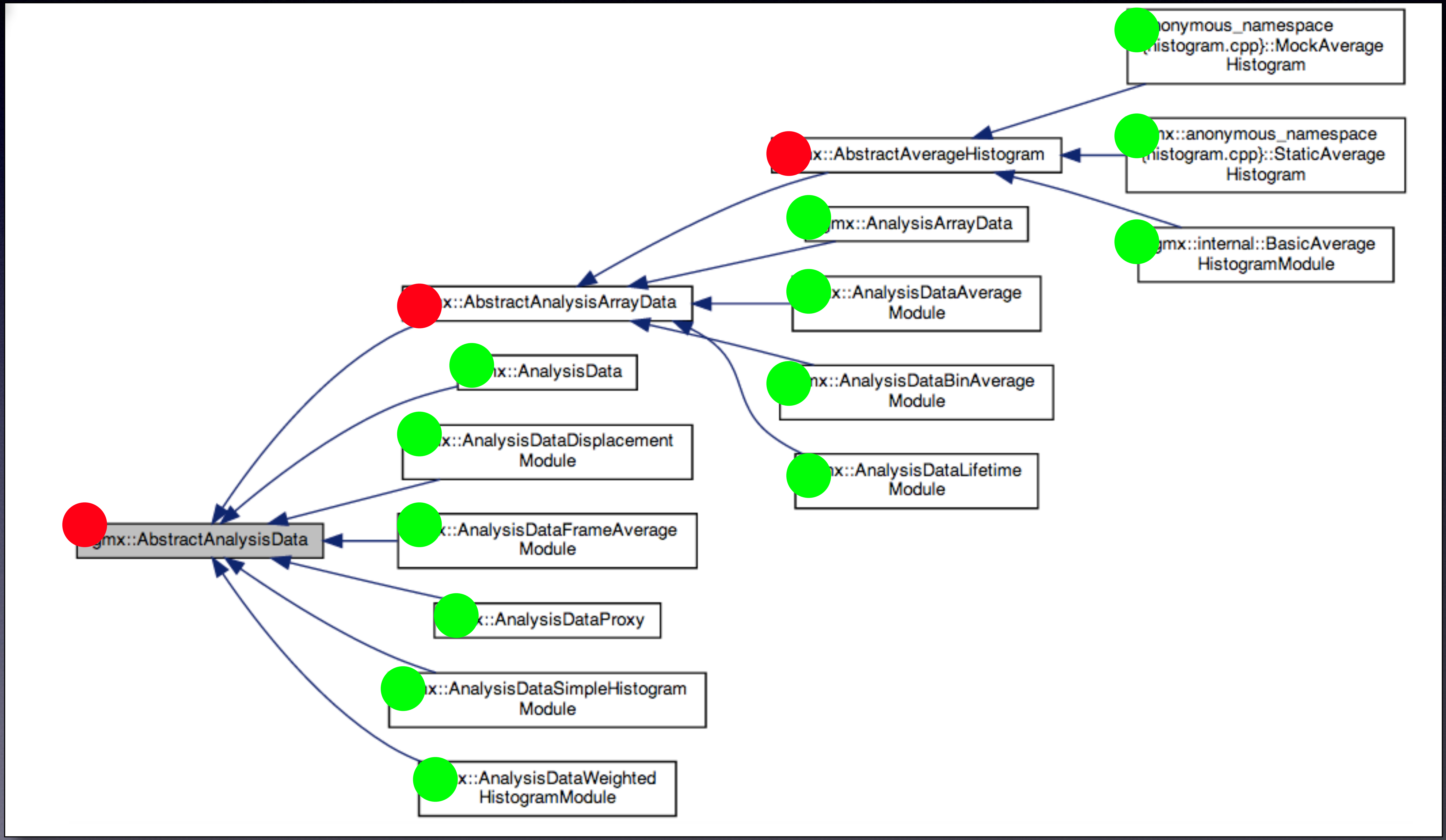
Why? Because we found that libstdc++ and libcxx do not use the same algorithm, so code will not produce the same results. We need to use our own algorithm - make sure it keeps working.

No need to ask: Of course we have integrated GoogleTest support into the IHPCSS/software-engineering repo - but I have not had time to write the actual tests yet. However, as you add more tests, they will all execute if you just issue “make check”.

Imagine a project with ~1000 classes, and that the class diagram below is a small excerpt (it's from Gromacs).

All classes have close-to-exhaustive unit tests - but your latest build now fails the unit test.
Green means the unit test for this class was OK, red means it failed.

Where do you look for the bug?



If each unit test targets a small method/function, you have isolated the bug to within ~50 lines-of-code before even opening your editor.

Commits - how code makes it into Gromacs

Who is allowed to write to your code repository?

Problems if you think some less talented developers might submit buggy code

Such as this one →

Code Review

Mark Abraham @mark.j.abraham started a thread on the diff 1 day ago
Last updated by Alan Gray 6 hours ago

Toggle thread

src/gromacs/ewald/pme_pp_comm_gpu_impl.h

```
    /*! \brief Creates PME-PP GPU communication object.
     *
-    * \param[in] comm             Communicator used for simulation
-    * \param[in] pmeRank          Rank of PME task
-    * \param[in] deviceContext    GPU context.
-    * \param[in] deviceStream     GPU stream.
+    * \param[in] comm               Communicator used for simulation
+    * \param[in] pmeRank            Rank of PME task
+    * \param[in] pmeCpuForceBuffer Buffer for PME force in CPU memory
+    * \param[in] deviceContext      GPU context.
+    * \param[in] deviceStream       GPU stream.
     */
-   Impl(MPI_Comm comm, int pmeRank, const DeviceContext& deviceContext, const DeviceStream&
 deviceStream);
+   Impl(MPI_Comm                 comm,
+        int                      pmeRank,
+        std::vector<gmx::RVec>& pmeCpuForceBuffer,
```

Mark Abraham @mark.j.abraham · 1 day ago — Owner

use `ArrayRef<gmx::RVec>` rather than `vector&`, assuming we agree with my suggestion elsewhere to avoid resizing this vector in this class.

Alan Gray @alangray3 · 8 hours ago — Developer

I tried this but couldn't get it to work. I think this is because the data buffer gets reallocated whenever the number of atoms changes. The pme-pp object currently stores the outer container (passed in at construction), and therefore accessing the data() method every search step automatically points to any reallocated location. I'm not sure if this is compatible with using an arrayref, please let me know if you think it should be.

Mark Abraham @mark.j.abraham · 7 hours ago — Owner

That ArrayRef does indeed need updating every time we repartition, because it might have been reallocated.

Alan Gray @alangray3 · 6 hours ago — Developer

Thanks for clarifying - probably most elegant to keep the current solution then.

Mark Abraham @mark.j.abraham started a thread on an old version of the diff 1 day ago
Last updated by Alan Gray 8 hours ago

Toggle thread

src/gromacs/ewald/pme_pp_comm_gpu_impl.cu

```
void PmePpCommGpu::Impl::reinit(int size)
{
+    // Reallocate buffers used for staging PME force
+    reallocateDeviceBuffer(&d_pmeForces_, size, &d_pmeForcesSize_, &d_pmeForcesSizeAlloc_,
deviceContext_);
+    pmeCpuForceBuffer_.resize(size);
```

Mark Abraham @mark.j.abraham · 1 day ago — Owner

This is redundant with the resize that happens in `gmx_pme_receive_f`. The latter happens every step, which is not great either. There should be an object to manage receiving PME forces that owns this vector and resizes it every time we repartition, but that's a long-standing problem for the core team to resolve. So I suggest we remove this resize, and convert `pmeCpuForceBuffer` to `ArrayRef<RVec>` as suggested elsewhere.

Alan Gray @alangray3 changed this line in version 3 of the diff 8 hours ago

Alan Gray @alangray3 · 8 hours ago — Developer

Now removed the resize() (see above).

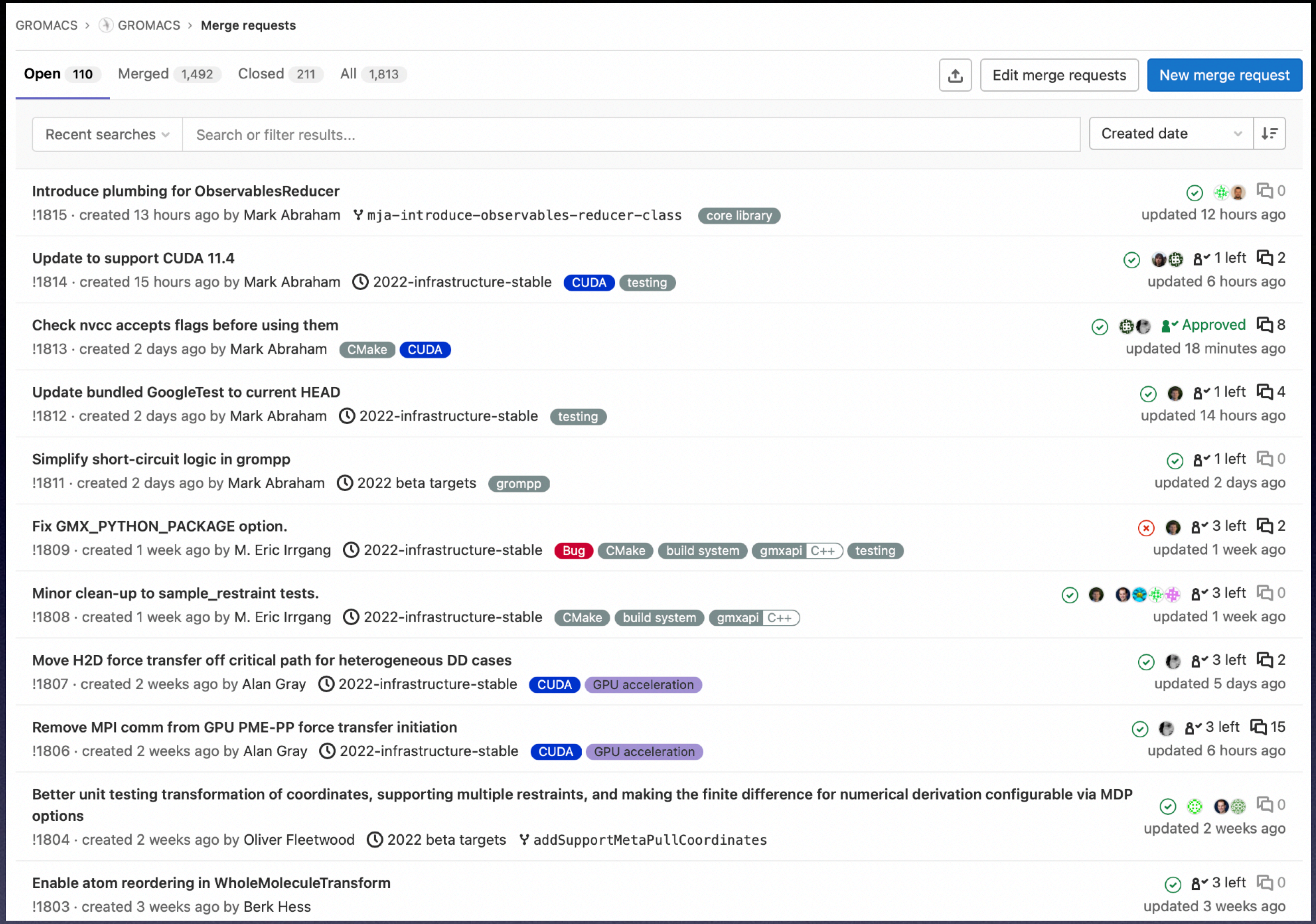
Reply…

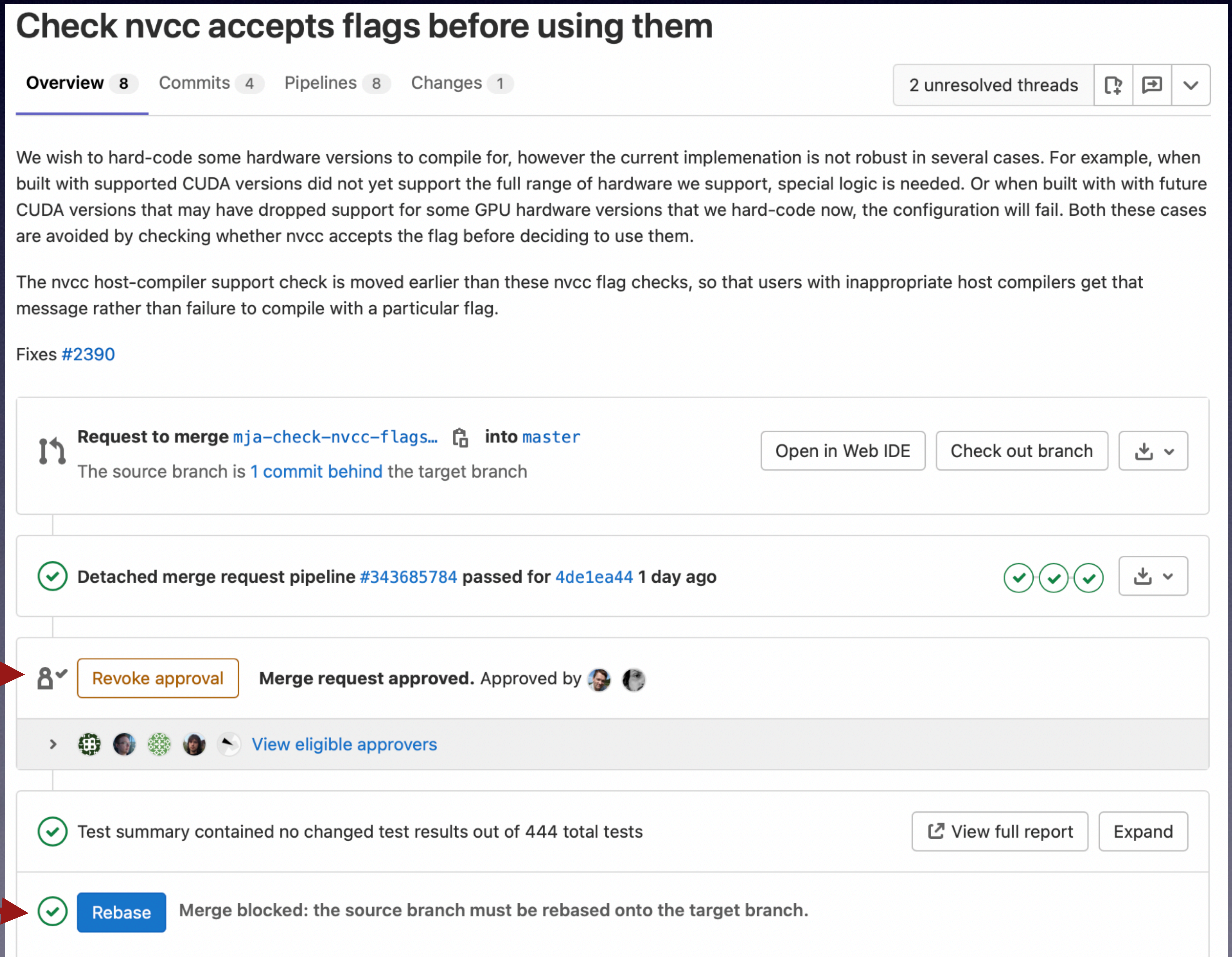
Resolve thread

The team has to accept that NOBODY can be allowed to bypass code review.

Each new comment on the MR (merge request) in GitLab opens a “thread”, which is “resolved” when the commenter is happy.

We have a TON of merge requests in flight. With full dependency tracking, patches can be rebased onto others by hitting a rebase button, or even edited on-the-fly in the window





Anybody can add comments. When two eligible developers say OK (Erik + Szilard here), the patch can be merged into master by a maintainer - but note how GitLab blocks that until it's been rebased!

Maintaining quality & avoiding breaking stuff

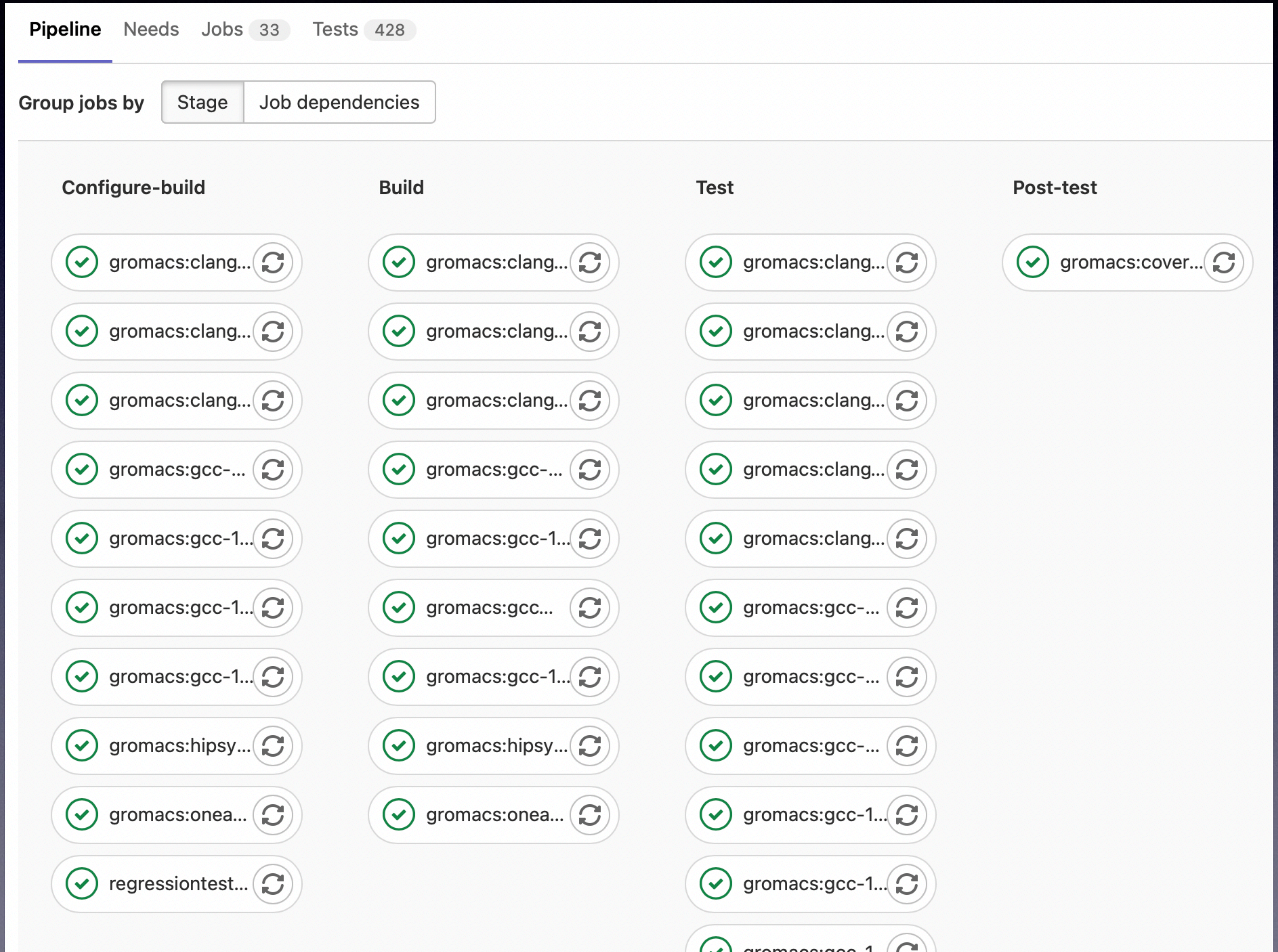
How do I make sure that *I* don’t make mistakes?

Continuous Integration

Part of our GitLab environment

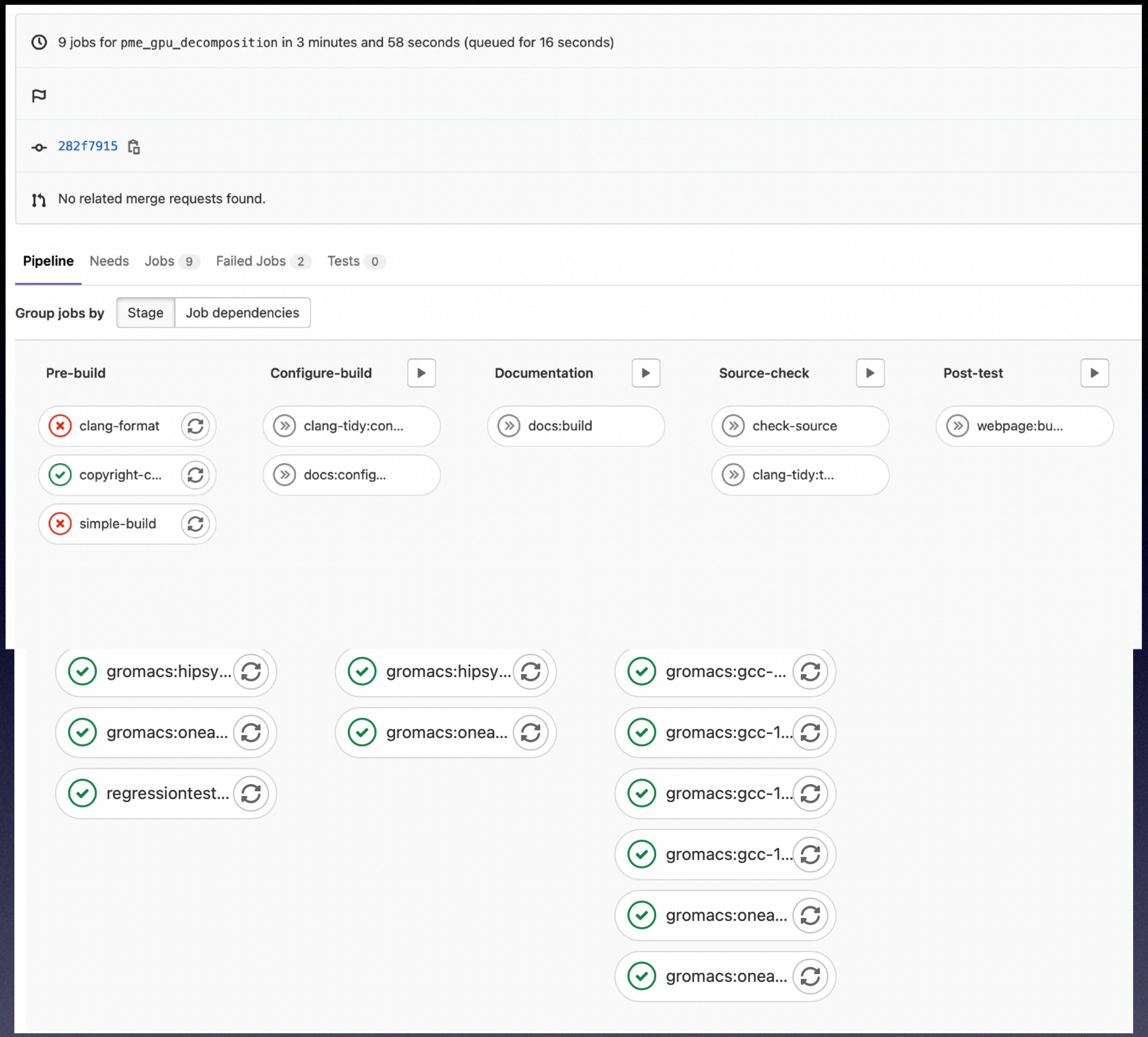
Every single merge request is tested automatically, including both builds and regression tests.

- Moderate usage is free both on GitHub (Travis) and GitLab
- For large usage, you can use your own servers or pay them
- Catches Cmake build errors
- Catches unit test failures



GROMACS CI tests for *every* commit

- Unit Tests: Do modules reproduce reference values?
  … on x86, Power 9, ARM, CUDA, OpenCL, SYCL CPU & SYCL GPU.
- Integration tests: Does a normal full run work?
- Regression tests: Are previous simulation results identical?
- Physical validation tests: Do we reproduce statistical ensemble fluctuations?
- Clang AddressSanitizer: Catch simple memory errors
- Clang MemorySanitizer: Like Valgrind - memory debugging
- Clang/GCC ThreadSanitizer: Thread synchronization errors
- Clang Static Analyzer: Logical execution dependency errors
- Cppcheck: Another static analyzer
- Uncrustify: Proper code formatting, no tabs, brace standards?
- Doxygen: All classes/methods/arguments/variables documented?
- Coming: Performance regression testing



Pre-submit GROMACS testing:

Changes cannot be committed until this entire pipeline is all green

Post-submit GROMACS testing:

Rare hardware and longer-running performance tests are performed once each patch has been approved, or nightly.

All 6 | Active 4 | Inactive 2 | New schedule

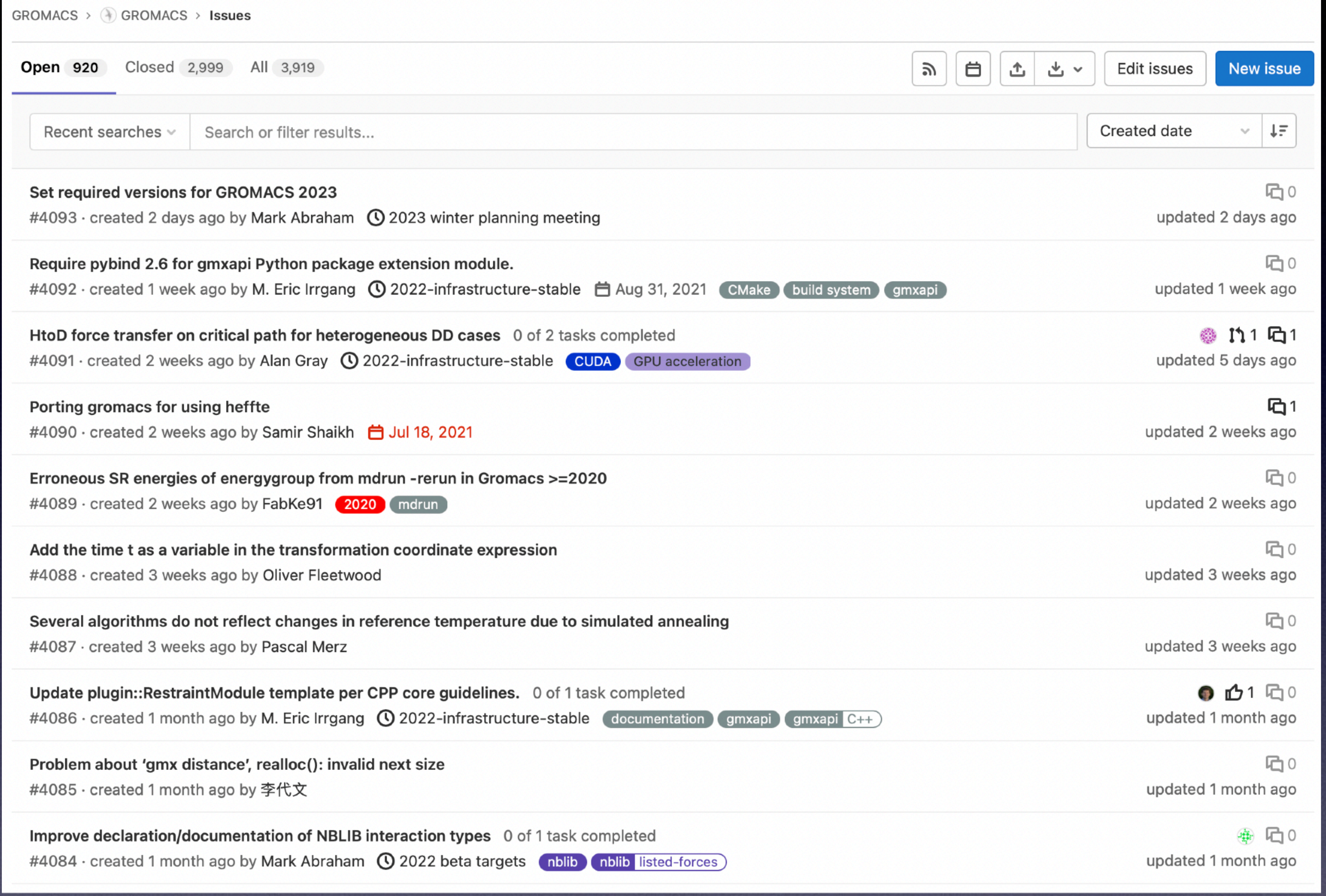
Description	Target	Last Pipeline	Next Run	Owner	
Nightly 2021	release-2021	#343967915	in 6 hours	Paul Bauer	Take ownership
Nightly 2021 release	release-2021	#343951753	in 5 hours	Paul Bauer	Take ownership
Nightly 2020	release-2020	#330183910	Inactive	Paul Bauer	Take ownership
Nightly master	master	#343893584	in 1 hour	Paul Bauer	Take ownership
Nightly 2020 release	release-2020	#330184061	Inactive	Paul Bauer	Take ownership
Nightly master release	master	#343950513	in 5 hours	Paul Bauer	Take ownership

Travis CI

https://travis-ci.org

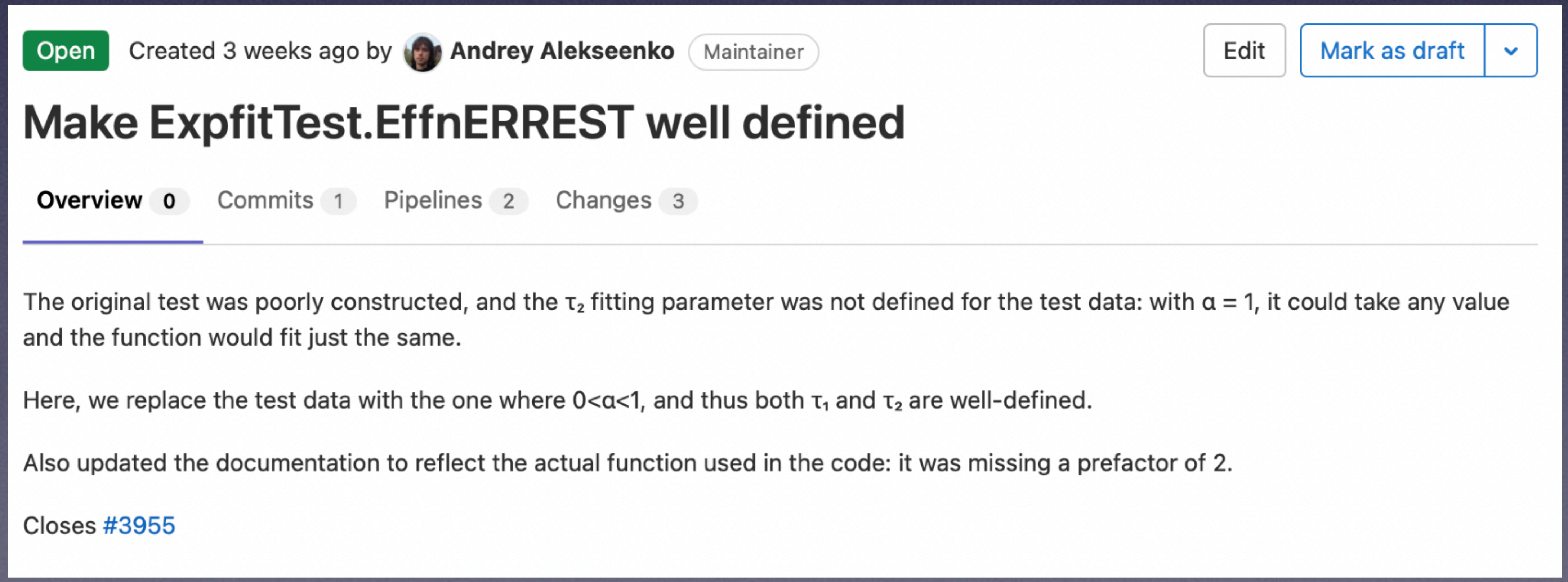
- Before moving to GitLab, GROMACS used Jenkins which is *very* powerful, but you need to set it up yourself to do advanced stuff, and/or arrange access to special hardware
- If your needs are more modest, Travis-CI is a much simpler environment that offers *free* CI testing of open source GitHub repositories
- Of course this is enables for the IHPCSS-laplace repo: Every time I push an update, the code is built, followed by execution of the unit tests.
- If you look at the two badges at GitHub, green colors mean both the Travis CI and ReadTheDocs builds are OK.
- Suggested exercise: Clone/rename the repo, and turn on both Travis & ReadTheDocs automated builds in your version of it!

Issue tracking



- Version 1.2.3 has bug X!
- Windows builds broke
- How is the work going on refactoring module Y?
- Should we improve scaling by method Z or W? Why did we decide to modify that loop in file F in git change Icfca5a?

Automatic referencing in commit messages!

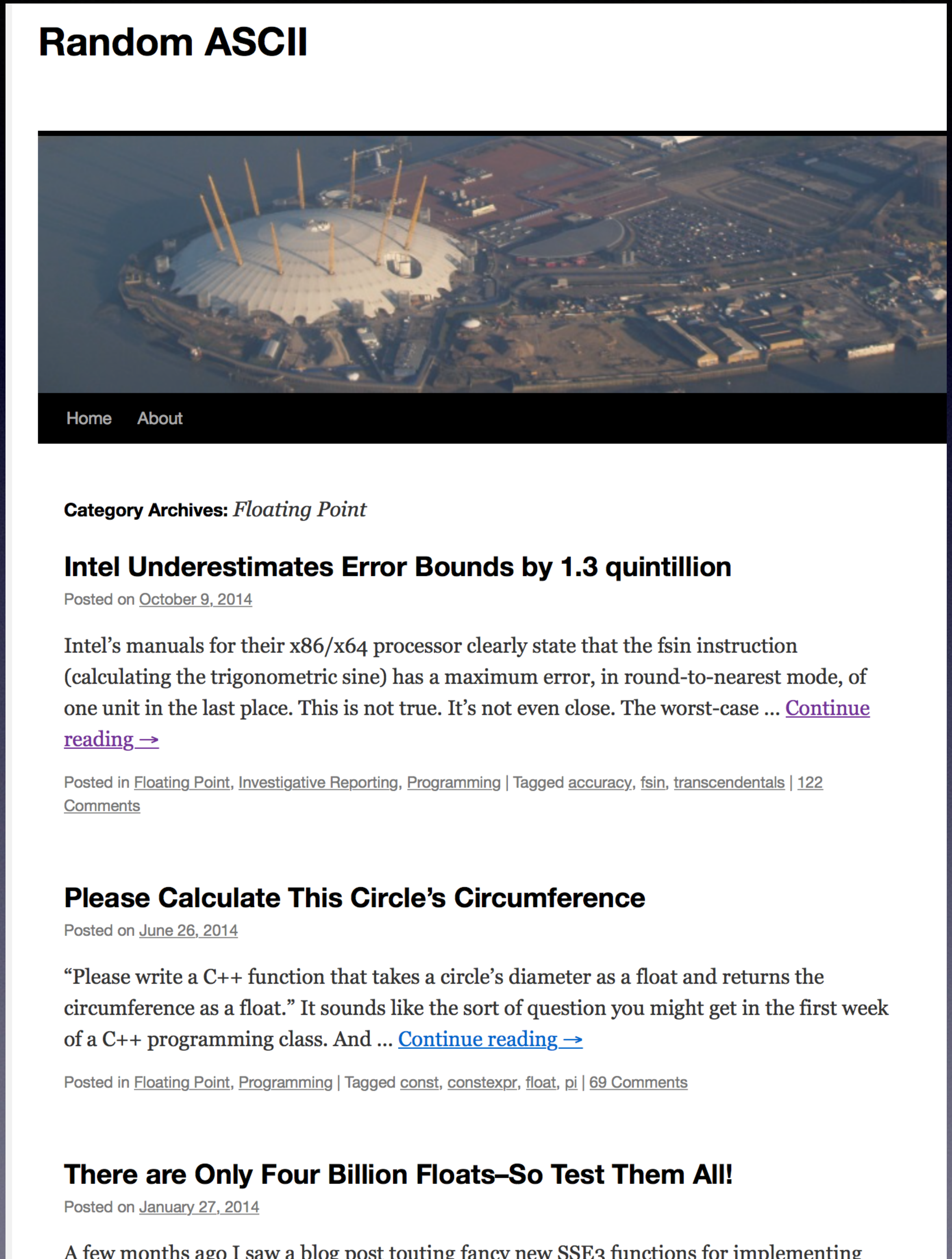


For IHPCSS/software-engineering, we use the integrated issue tracker in GitHub, but this too supports automated referencing e.g. for closing bugs.

http://randomascii.wordpress.com/category/floating-point/

Series of blog posts by Bruce Dawson about IEEE754 floating point

You **should** read this if you are working with scientific codes using floating-point!


Random ASCII

Home About

Category Archives: *Floating Point*

Intel Underestimates Error Bounds by 1.3 quintillion

Posted on October 9, 2014

Intel's manuals for their x86/x64 processor clearly state that the fsin instruction (calculating the trigonometric sine) has a maximum error, in round-to-nearest mode, of one unit in the last place. This is not true. It's not even close. The worst-case … Continue reading →

Posted in Floating Point, Investigative Reporting, Programming | Tagged accuracy, fsin, transcendentals | 122 Comments

Please Calculate This Circle's Circumference

Posted on June 26, 2014

"Please write a C++ function that takes a circle's diameter as a float and returns the circumference as a float." It sounds like the sort of question you might get in the first week of a C++ programming class. And … Continue reading →

Posted in Floating Point, Programming | Tagged const, constexpr, float, pi | 69 Comments

There are Only Four Billion Floats–So Test Them All!

Posted on January 27, 2014

A few months ago I saw a blog post touting fancy new SSE3 functions for implementing

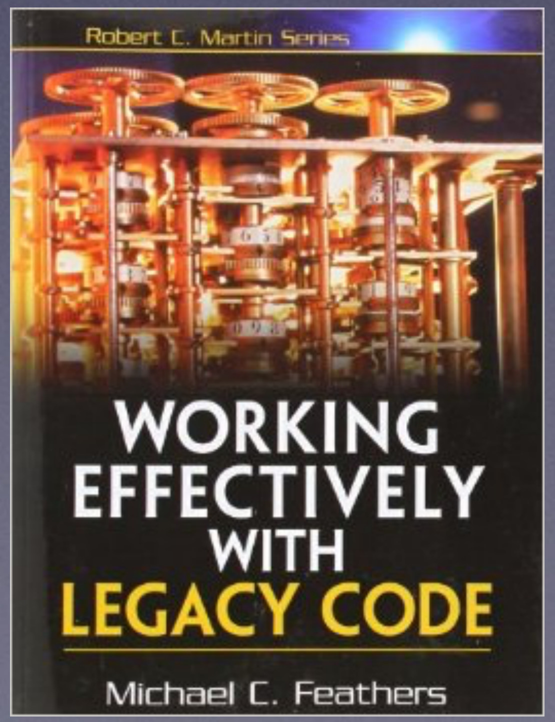
Teaser - this might not always return x = 0:

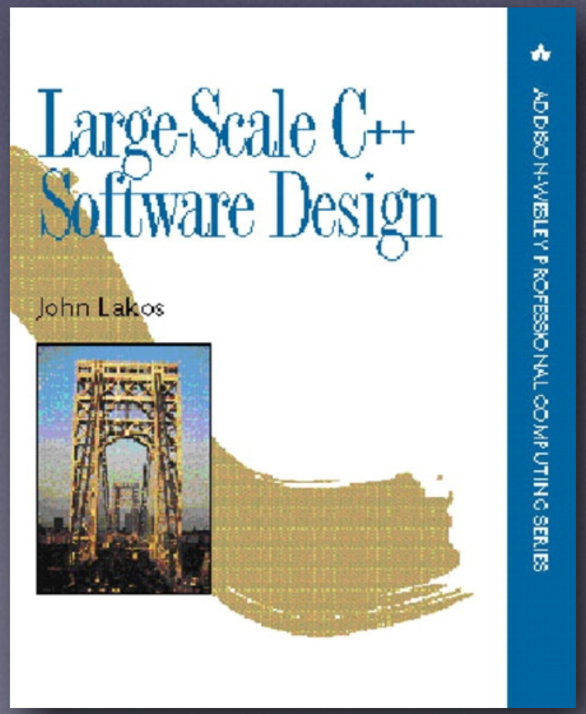
$$x = a \cdot b - a \cdot b$$

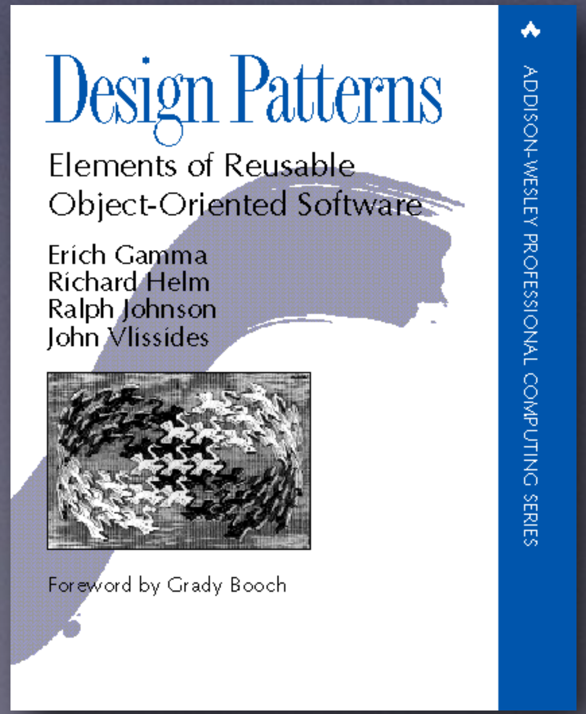
More worthwhile reading:
"What every computer scientist should know about floating-point arithmetic"
[David Goldberg]

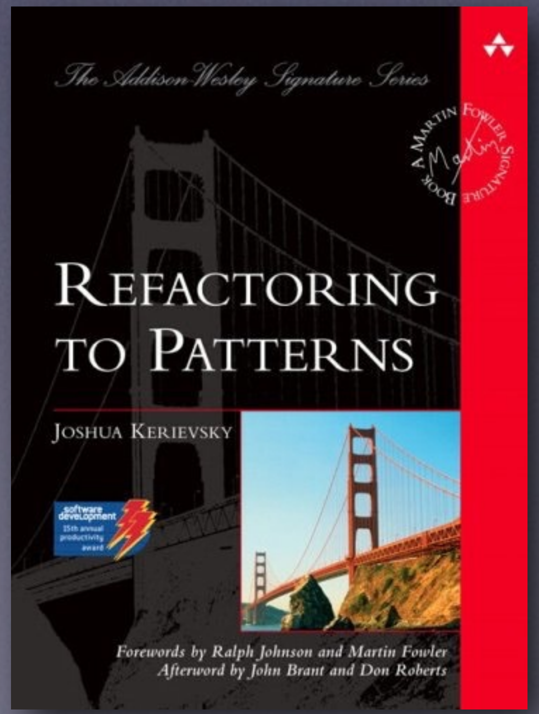
Some good reading

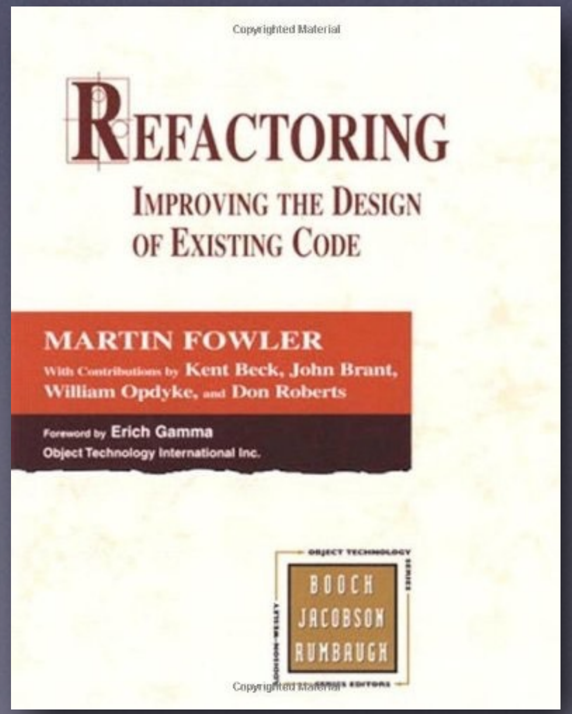
- Working effectively with legacy code [Michael Feathers]
- Large-scale C++ software design [John Lakos]
- Design Patterns - Elements of Reusable Object-oriented software [Gamma, Helm, Johnson, Vlissides] “Gang of four”
- Refactoring to Patterns [Joshua Kerievsky]
- Refactoring - improving the design of existing code [Martin Fowler]
- Effective C++ - 55 specific ways to improve your programs and design [Scott Meyers]
- Patterns for concurrent, parallel, and distributed systems: http://www.cs.wustl.edu/~schmidt/patterns-ace.html
- What everybody should know about floating-point math: http://randomascii.wordpress.com/category/floating-point/

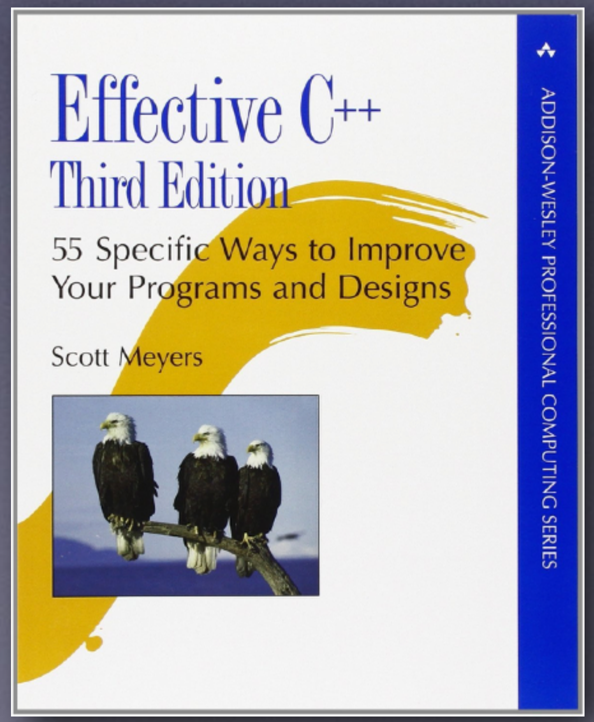












Acknowledgments

GROMACS: Berk Hess, Szilard Pall, Mark Abraham, Aleksei Iiupinov, John Eblen, Roland Shultz, Christian Wennberg, Viveca Lindahl

RELION: Dari Kimanius, Björn Forsberg, Sjors Scheres, Alexey Amunts, Marta Carroni, Shintaro Aibara

NVIDIA: Mark Berger, Duncan Poole, Julia Levites, Jiri Kraus, Nikolay Markovskiy

INTEL: Charles Congdon, Sheng Fu, Kristina Kermanshahche, Yuping Zhao

CSCS: Thomas Schulthess, Victor Holanda, Prashant Kanduri PDC: Erwin Laure