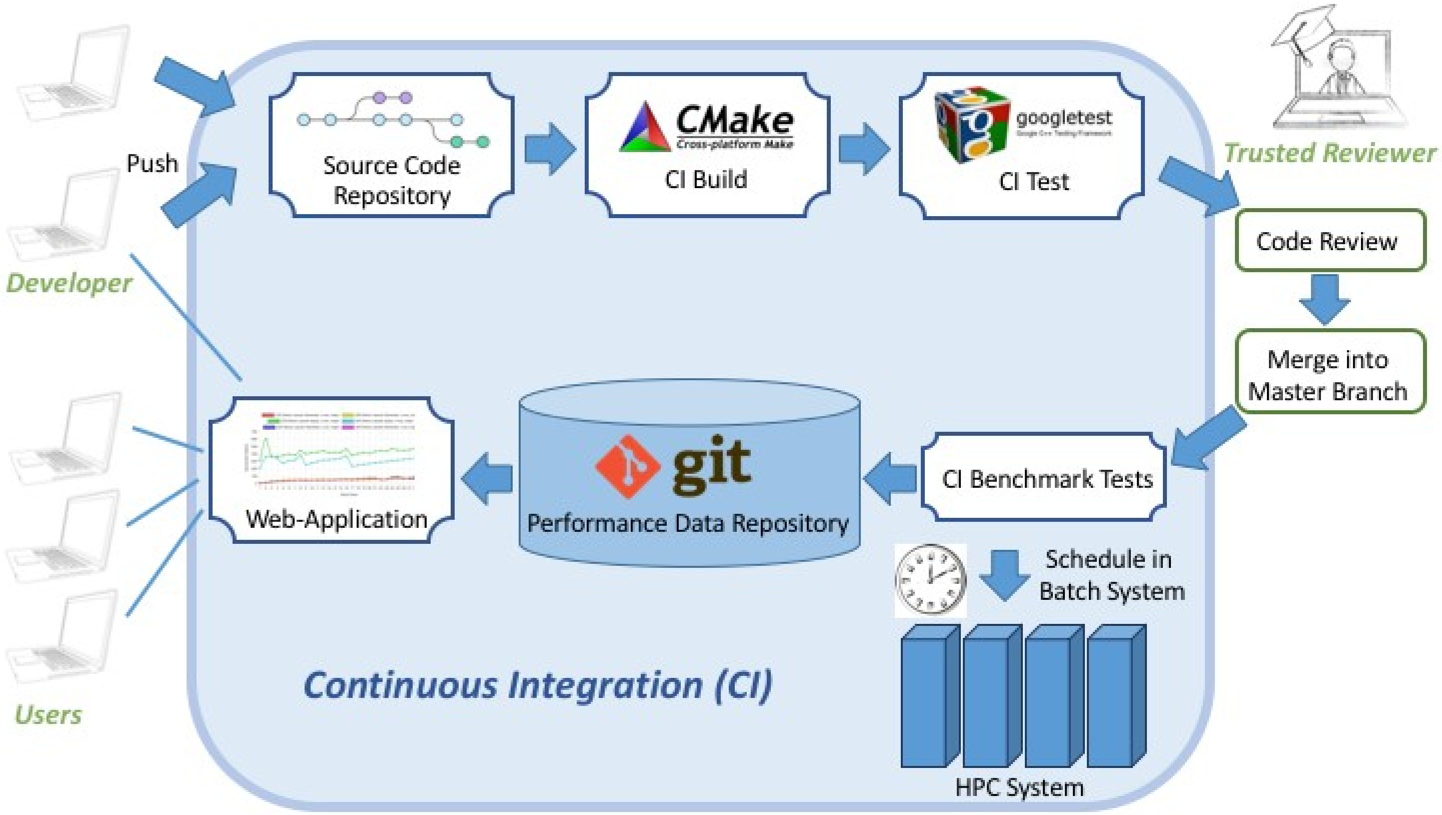
Karlsruhe Institute of Technology

A collaborative peer review process in grading coding assignments for HPC

Fritz Göbel[1], Pratik Nayak[1], Hartwig Anzt[1], Jan-Patrick Lehr[2]
[1]Steinbuch Centre for Computing (SCC), [2]TU Darmstadt (Scientific Computing)

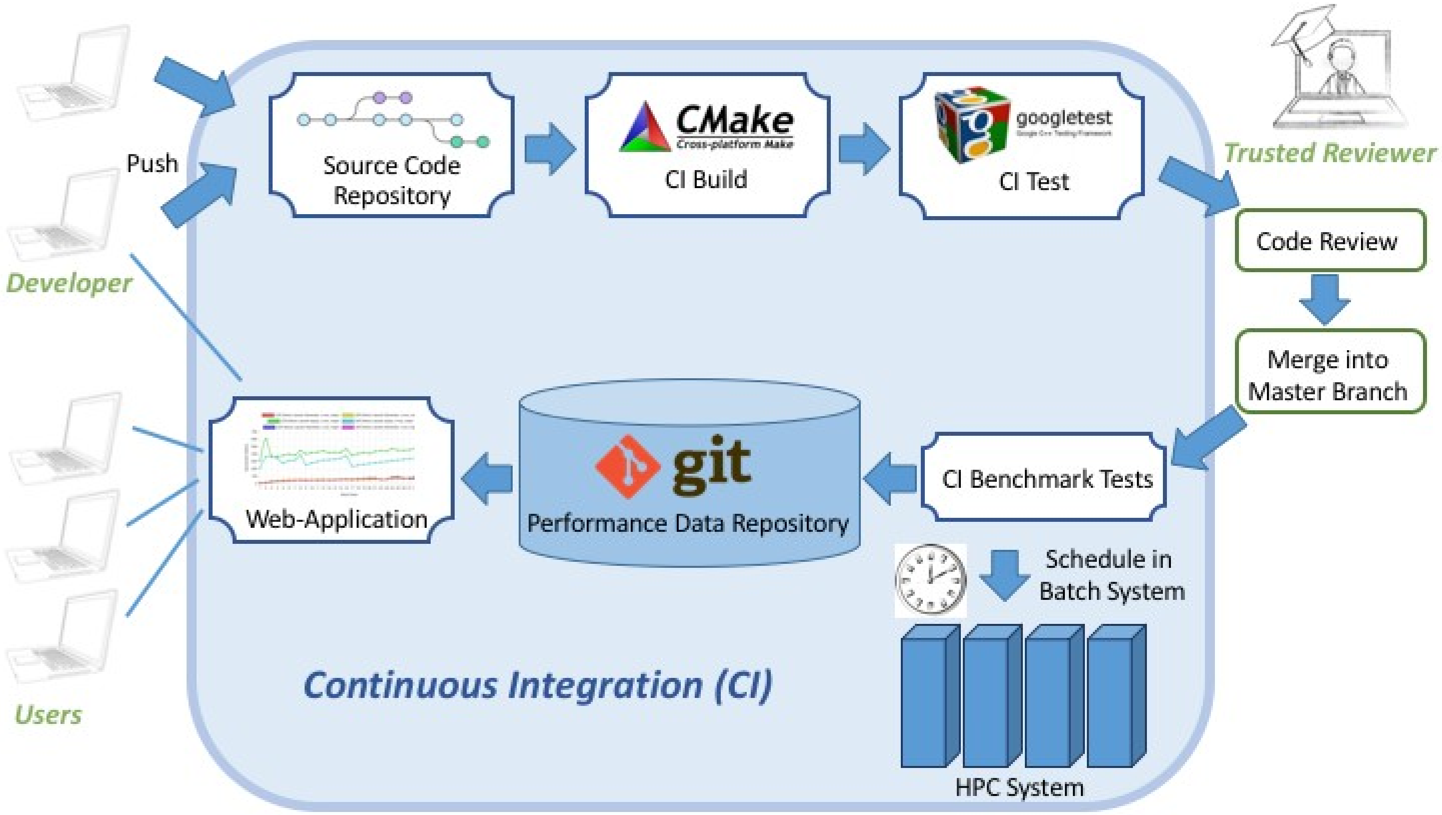
Motivation

Developing sustainable scientific software:



Motivation

Developing sustainable scientific software:



New student assistant / PhD student

Motivation

At KIT, we teach a class on Numerical Linear Algebra in HPC. Topics are:

- Matrix formats, linear solvers, preconditioning techniques, …
- Performance analysis
- Parallelization with OMP / CUDA / MPI

Motivation

At KIT, we teach a class on Numerical Linear Algebra in HPC. Topics are:

- Matrix formats, linear solvers, preconditioning techniques, …
- Performance analysis
- Parallelization with OMP / CUDA / MPI

Out of this class, we often get motivated people who

- Have great ideas,
- Know how to put their ideas into code,
- Want to contribute.

Motivation

At KIT, we teach a class on Numerical Linear Algebra in HPC. Topics are:

- Matrix formats, linear solvers, preconditioning techniques, …
- Performance analysis
- Parallelization with OMP / CUDA / MPI

Out of this class, we often get motivated people who

- Have great ideas,
- Know how to put their ideas into code,
- Want to contribute.

BUT: Handling version control systems, continuous integration etc. often poses an entry barrier!

Motivation

At KIT, we teach a class on Numerical Linear Algebra in HPC. Topics are:

- Matrix formats, linear solvers, preconditioning techniques, …
- Performance analysis
- Parallelization with OMP / CUDA / MPI

Out of this class, we often get motivated people who

- Have great ideas,
- Know how to put their ideas into code,
- Want to contribute.

BUT: Handling version control systems, continuous integration etc. often poses an entry barrier!

→ Let’s include a realistic workflow in coding assignments!

Goals

- Keep a high standard of teaching different aspects of HPC:
  - Learning about different kinds of algorithms (solvers, preconditioners etc.)
  - Learning about different programming models (OMP, CUDA, HIP, MPI, …)
  - Give the opportunity to work with different platforms (NVIDIA, AMD, Intel, …)

- In teaching, include realistic workflow for:
  - Writing sustainable software
  - Maintaining code
  - Generating reproducible results

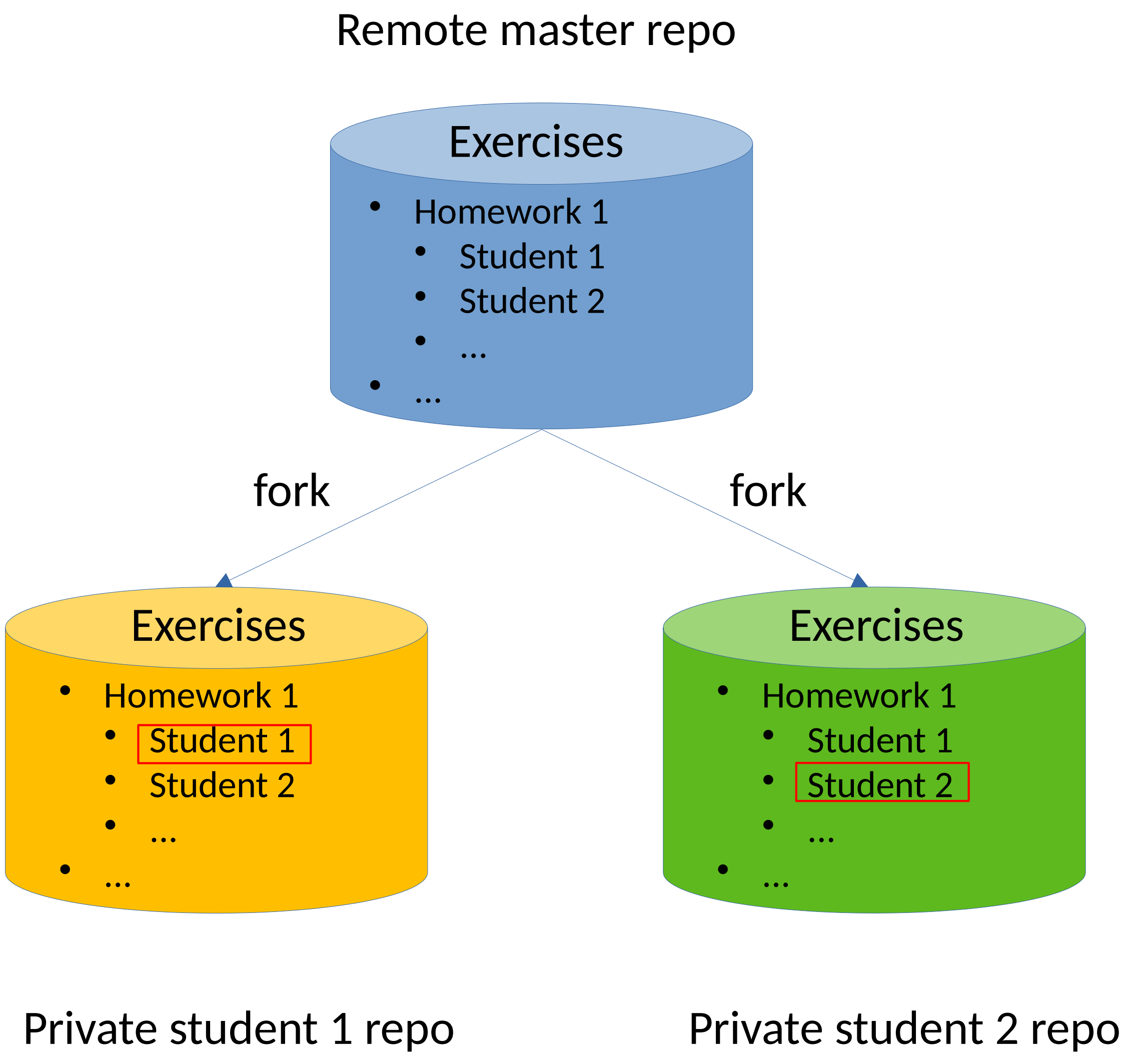
- Flatten learning curve for new student assistants / PhD students

- Enable them to quickly make strong contributions

Our Approach

General setting:

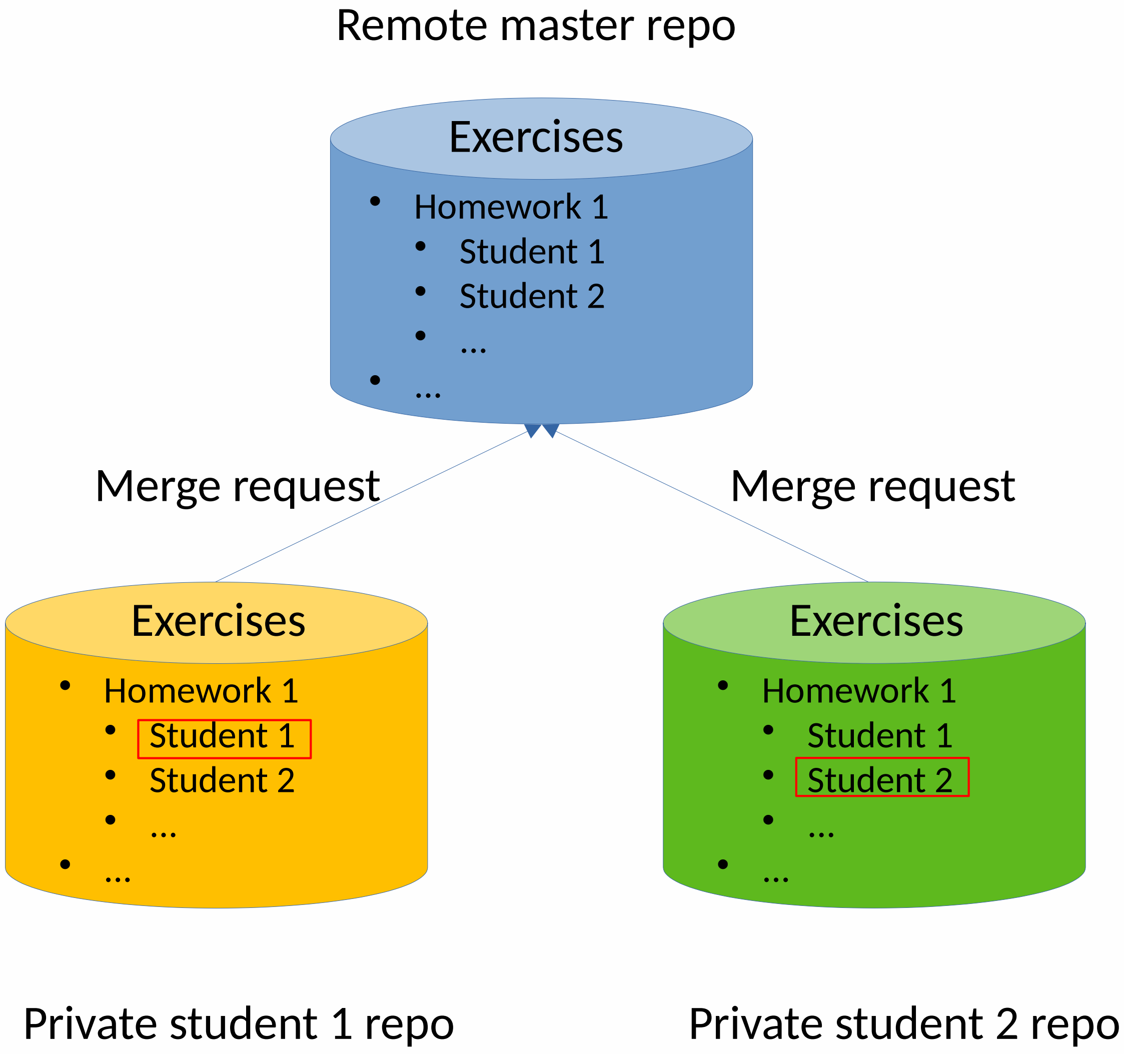
- One git repository for homework assignments
- Each assignment in a separate directory
  - One identical subdirectory for each student
  - Students work on private fork in their specific directory



Our Approach

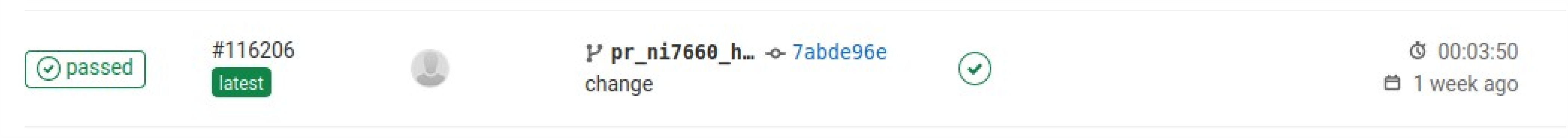
Submitting a finished homework assignment:

- Requirements:
  - Our CI system can compile the code
  - All provided unit tests pass
- On a given date, every student opens a merge request to the master repo



Our Approach

A valid submissione will result in a passing pipeline:



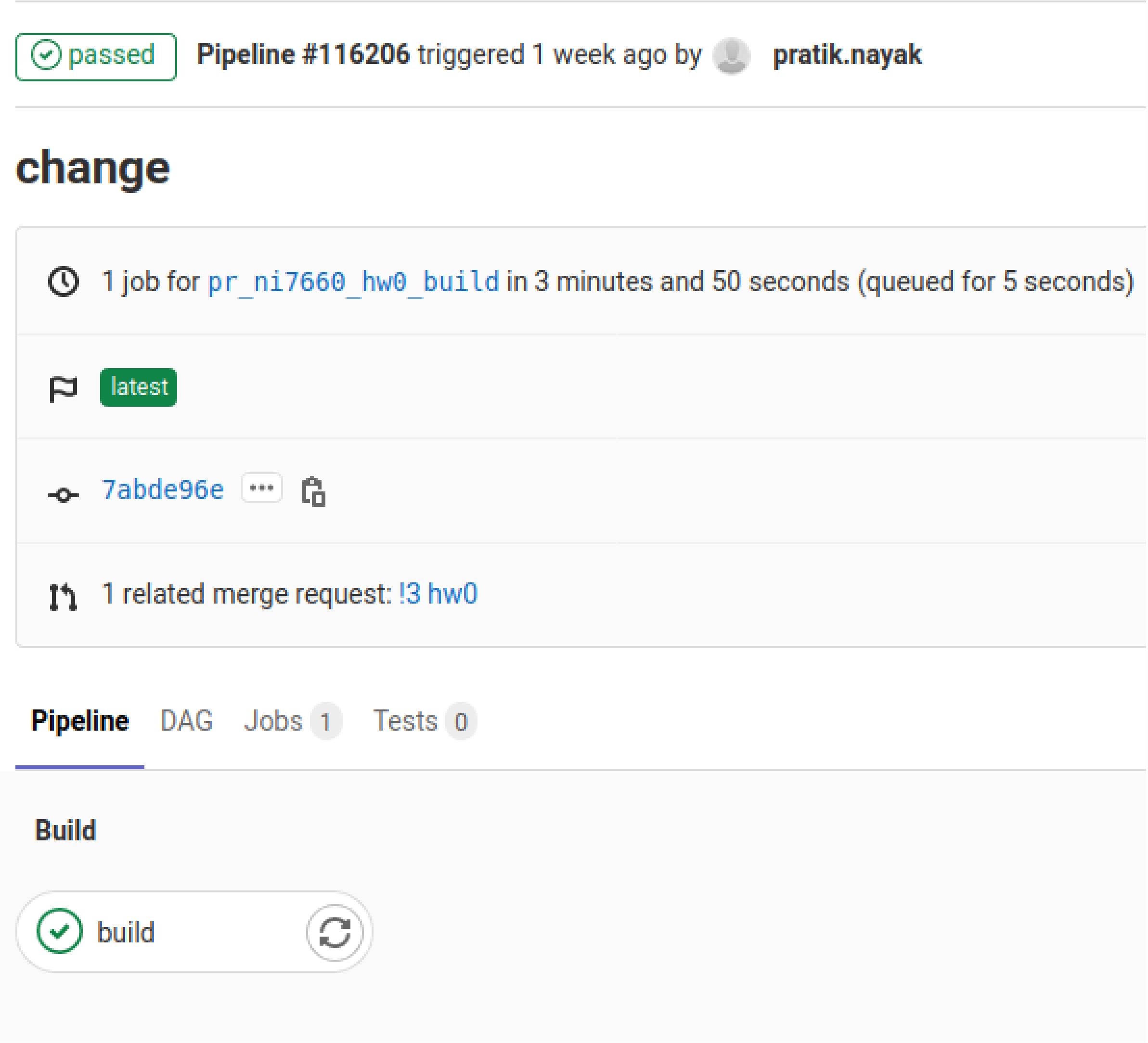
passed #116206 latest pr_ni7660_h… 7abde96e change 00:03:50 1 week ago

Our Approach

The pipeline builds:

- the homework code
- the ginkgo open source library to compare results against

The pipeline runs on our group’s CI system.



passed Pipeline #116206 triggered 1 week ago by pratik.nayak

change

1 job for pr_ni7660_hw0_build in 3 minutes and 50 seconds (queued for 5 seconds)

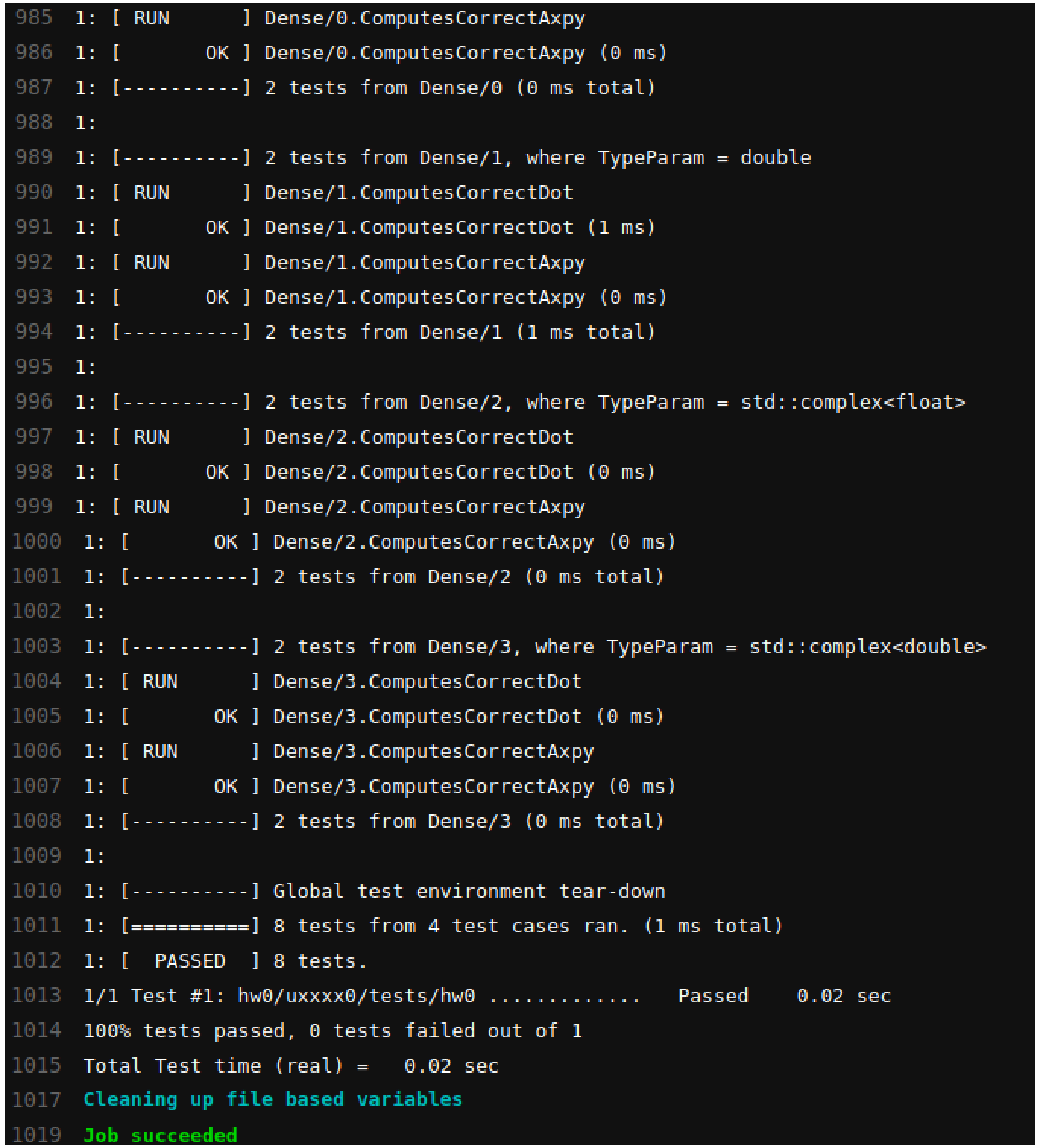
latest

7abde96e

1 related merge request: !3 hw0

Our Approach

- For each kernel, unit tests are run for real and complex, single and double precision input.
- We use gtest for unit testing

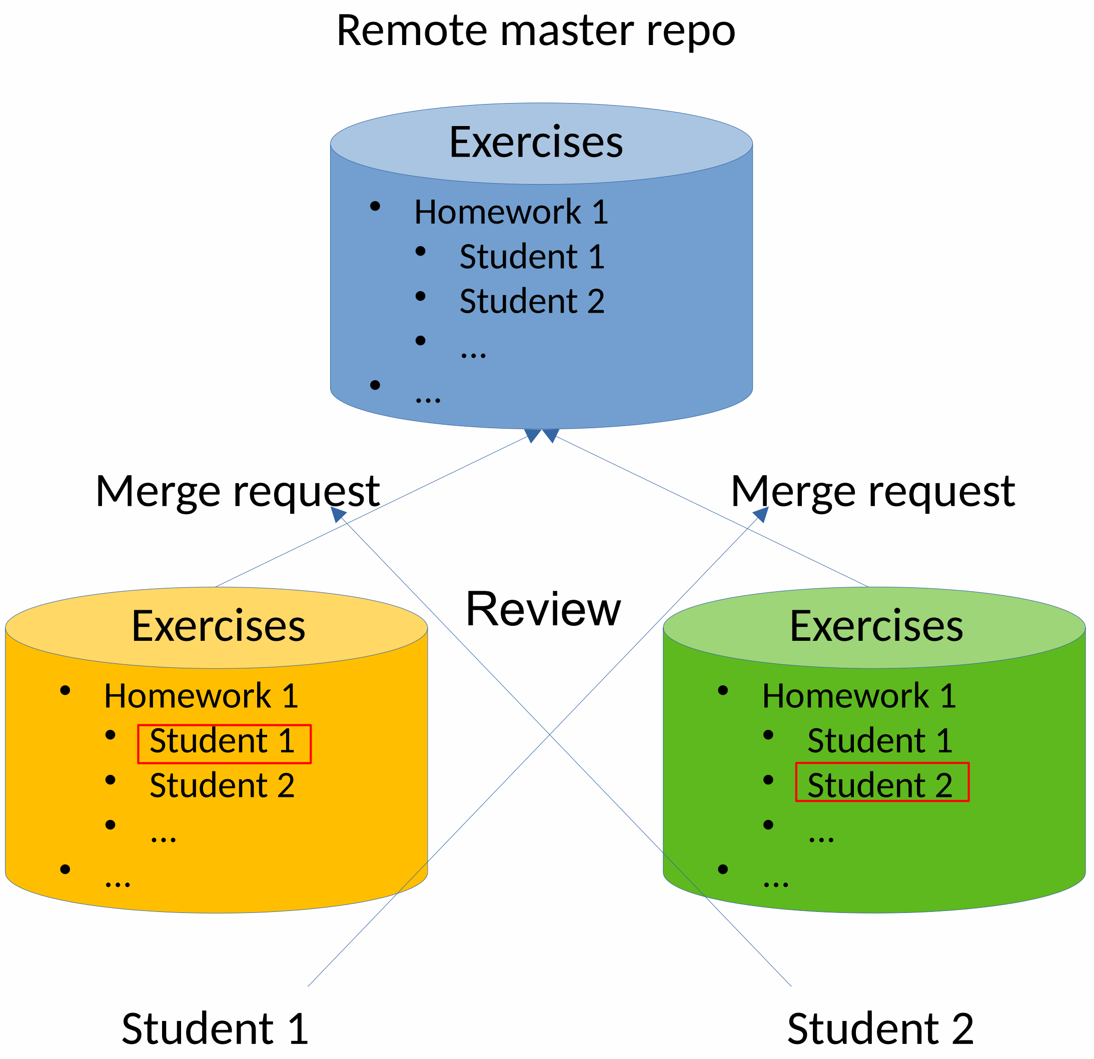


```
1: [ RUN      ] Dense/0.ComputesCorrectAxpy
1: [       OK ] Dense/0.ComputesCorrectAxpy (0 ms)
1: [----------] 2 tests from Dense/0 (0 ms total)
1:
1: [----------] 2 tests from Dense/1, where TypeParam = double
1: [ RUN      ] Dense/1.ComputesCorrectDot
1: [       OK ] Dense/1.ComputesCorrectDot (1 ms)
1: [ RUN      ] Dense/1.ComputesCorrectAxpy
1: [       OK ] Dense/1.ComputesCorrectAxpy (0 ms)
1: [----------] 2 tests from Dense/1 (1 ms total)
1:
1: [----------] 2 tests from Dense/2, where TypeParam = std::complex<float>
1: [ RUN      ] Dense/2.ComputesCorrectDot
1: [       OK ] Dense/2.ComputesCorrectDot (0 ms)
1: [ RUN      ] Dense/2.ComputesCorrectAxpy
1: [       OK ] Dense/2.ComputesCorrectAxpy (0 ms)
1: [----------] 2 tests from Dense/2 (0 ms total)
1:
1: [----------] 2 tests from Dense/3, where TypeParam = std::complex<double>
1: [ RUN      ] Dense/3.ComputesCorrectDot
1: [       OK ] Dense/3.ComputesCorrectDot (0 ms)
1: [ RUN      ] Dense/3.ComputesCorrectAxpy
1: [       OK ] Dense/3.ComputesCorrectAxpy (0 ms)
1: [----------] 2 tests from Dense/3 (0 ms total)
1:
1: [----------] Global test environment tear-down
1: [==========] 8 tests from 4 test cases ran. (1 ms total)
1: [  PASSED  ] 8 tests.
1/1 Test #1: hw0/uxxxx0/tests/hw0 ............   Passed    0.02 sec
100% tests passed, 0 tests failed out of 1
Total Test time (real) =   0.02 sec
Cleaning up file based variables
Job succeeded
```

Our Approach

Review process:

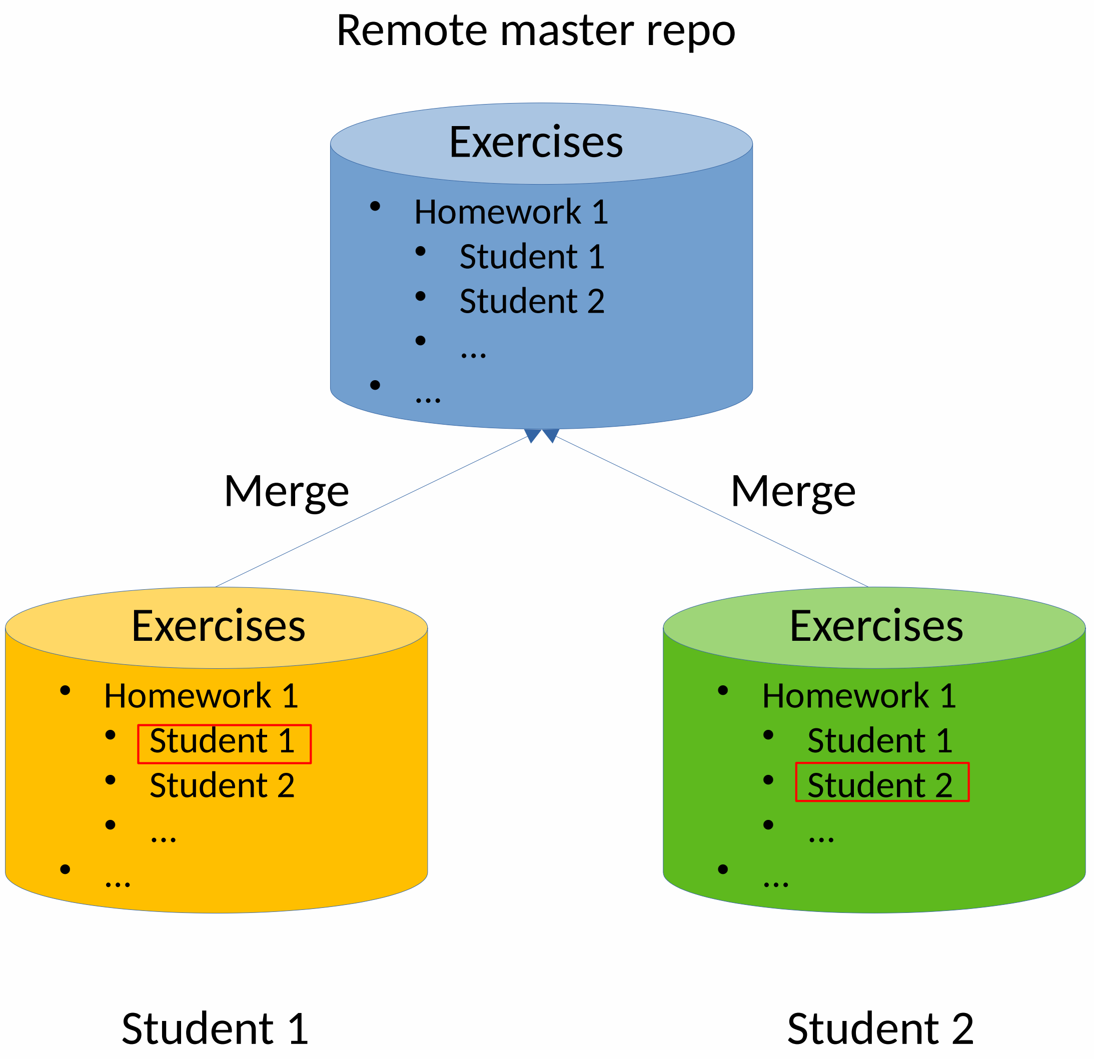
- Every week, each student reviews the merge request of one other student
- Reviewing criteria could be:
  - Readability
  - Performance
  - Pointing to possibly missed edge cases
  - …
- Conventional comments as general reviewing guideline (https://conventionalcomments.org/)



Our Approach

Final submission:

- After having the chance to enhance their code with respect to the review
- On a fixed date, all merge requests will be merged
- The merged code will be the final submission which in the end will be graded



Our Approach

Grading

- 10 points in total:
  - 5 points for technical report / analysis
  - 4 points for code
    - 2 points for working code
    - 1.5 points for code quality and performance
    - 0.5 points for employing suggested changes
  - 1 point for helpful / respectful code review

Challenges

- We usually have around 5 students, definitely less than 10 – how could we make this approach scalable for more attendees?
- Suggestions for realistic time frames for reviewing / adjusting submission according to code review?
- Suggestions on managing the overhead of the code review process and creating robust frameworks for students to work on?