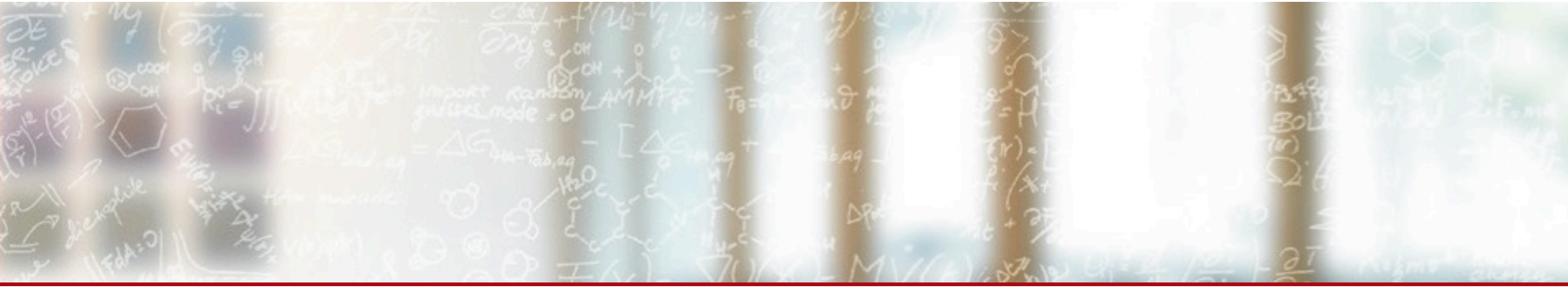


Regression Testing and Continuous Integration with ReFrame

CSCS User Lab Day – Meet the Swiss National Supercomputing Centre

Vasileios Karakasis, CSCS

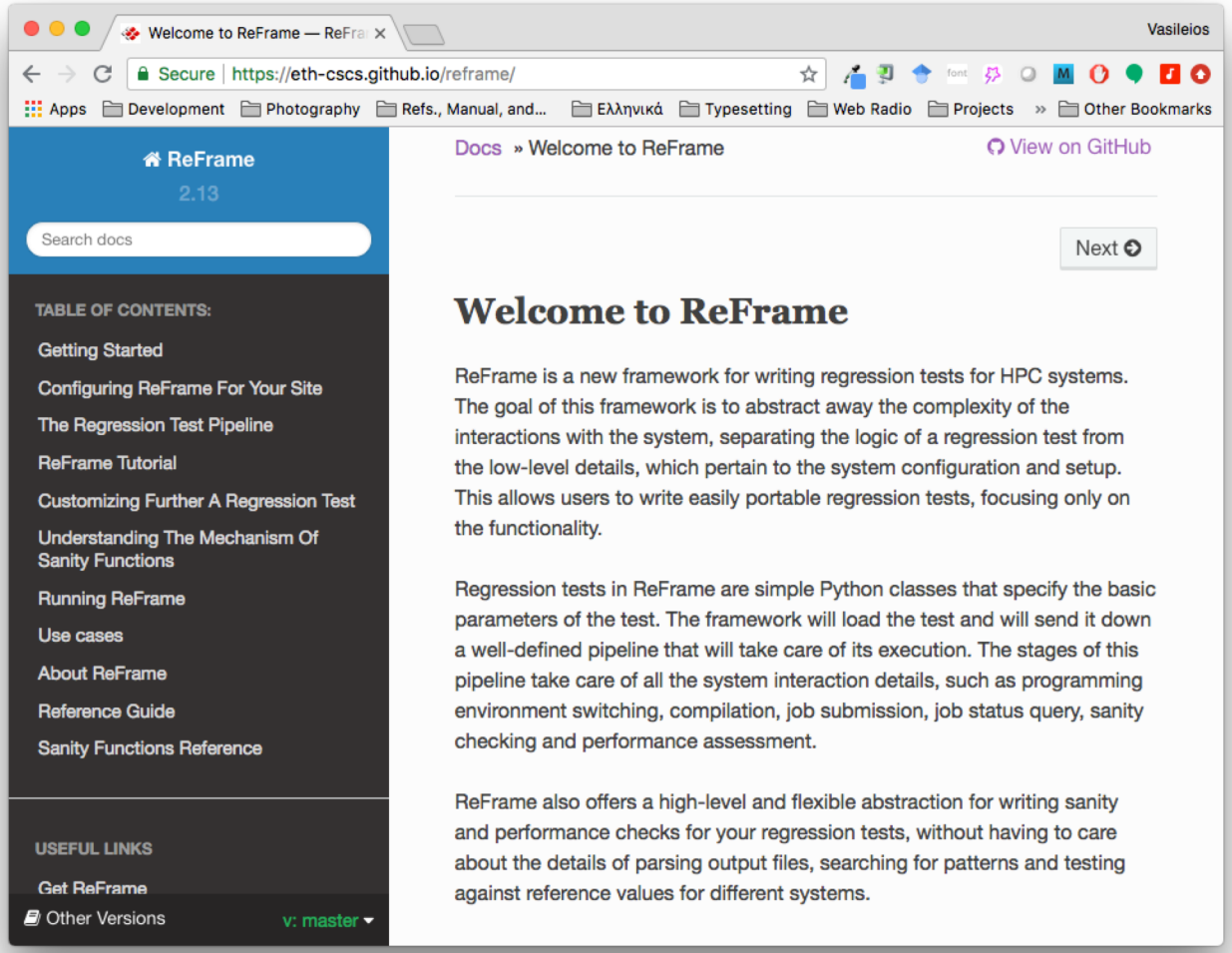
September 11, 2018

Table of Contents

What is ReFrame?

A new regression testing framework that

- allows writing **portable** HPC regression tests in Python,
- **abstracts away** the system interaction details,
- lets users focus solely on the **logic** of their test.



https://github.com/eth-cscs/reframe

Design Goals

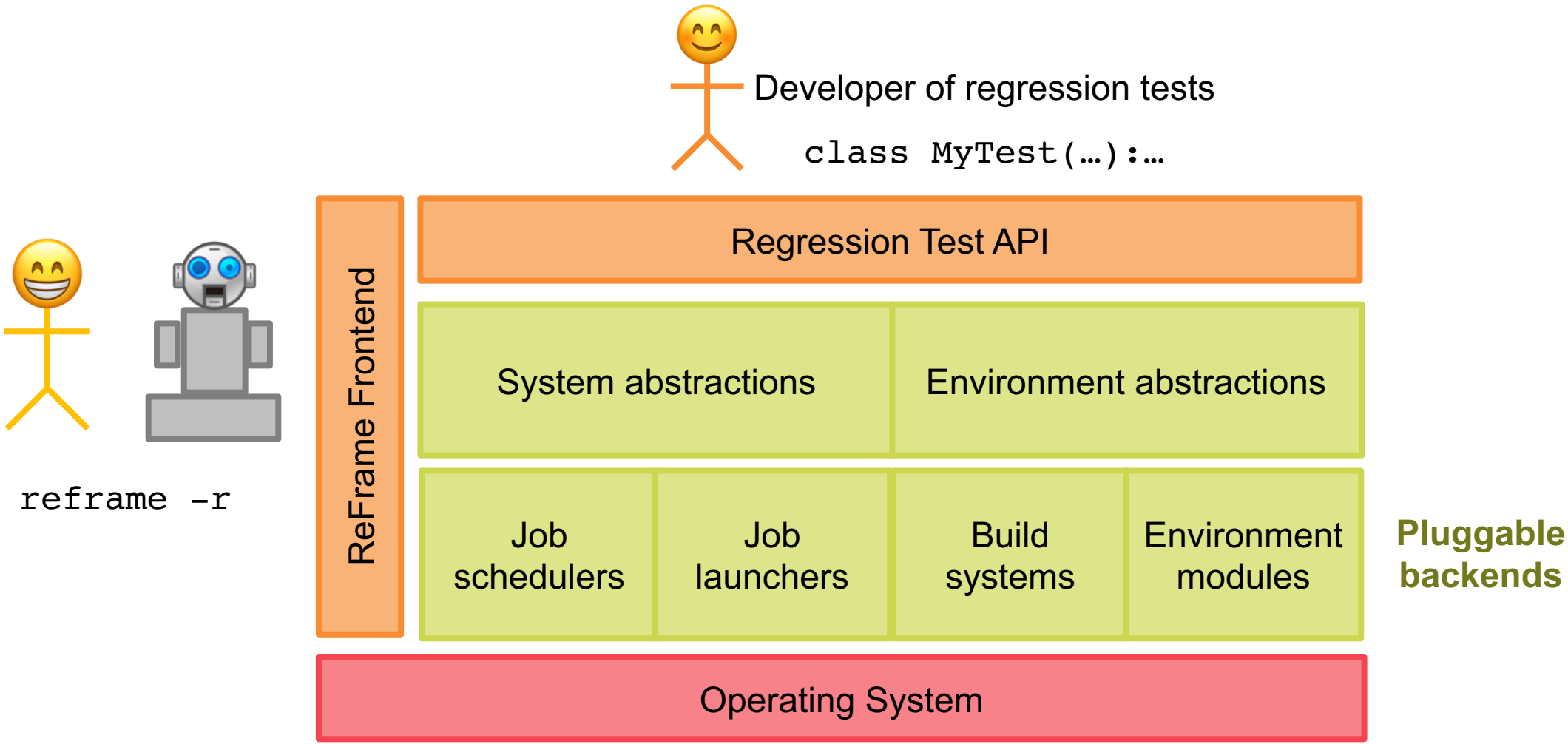
- Productivity
- Portability
- Speed and Ease of Use
- Robustness

Write once, test everywhere!

Key Features

- Separation of system and prog. environment configuration from test's logic
- Support for cycling through prog. environments and system partitions
- Regression tests written in Python
  - Easy customization of tests
  - Flexibility in organizing the tests
- Support for sanity and performance tests
  - Allows complex and custom analysis of the output through an embedded mini-language for sanity and performance checking.
- Progress and result reports
- Performance logging with support for Graylog
- Clean internal APIs that allow the easy extension of the framework's functionality
- ... and more (*https://github.com/eth-cscs/reframe*)

ReFrame's architecture



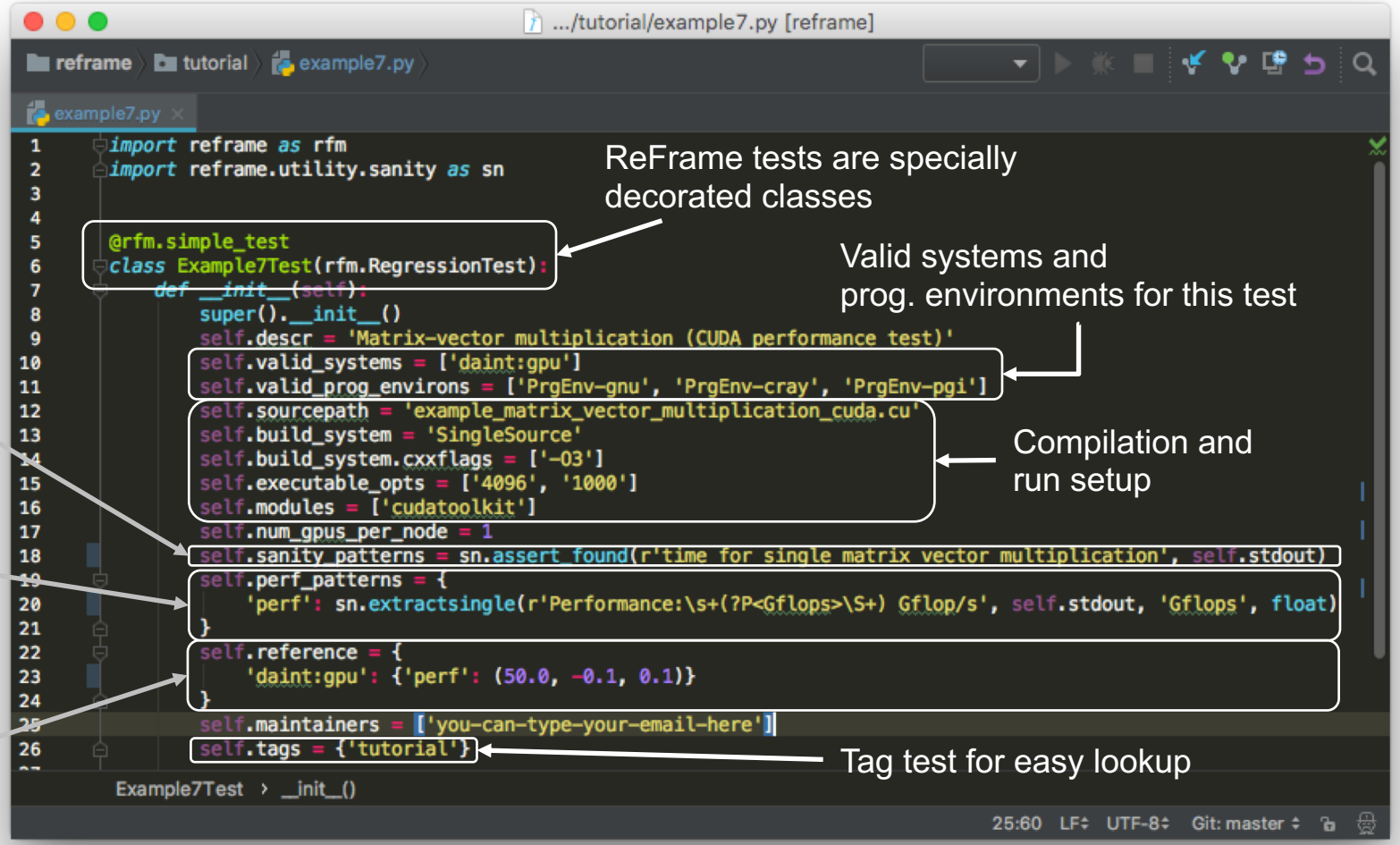
Writing a Regression Rest in ReFrame

A regression test writer should not care about...

- How to access to system partitions and if there are any.
- How (programming) environments are switched.
- How the test's environment is actually set up.
- How a job script is generated and if it's needed at all.
- How a sanity/performance pattern is looked up in the output.

ReFrame allows you to focus on the logic of your test.

Writing a Regression Rest in ReFrame




```python
import reframe as rfm
import reframe.utility.sanity as sn

@rfm.simple_test
class Example7Test(rfm.RegressionTest):
    def __init__(self):
        super().__init__()
        self.descr = 'Matrix-vector multiplication (CUDA performance test)'
        self.valid_systems = ['daint:gpu']
        self.valid_prog_environs = ['PrgEnv-gnu', 'PrgEnv-cray', 'PrgEnv-pgi']
        self.sourcepath = 'example_matrix_vector_multiplication_cuda.cu'
        self.build_system = 'SingleSource'
        self.build_system.cxxflags = ['-O3']
        self.executable_opts = ['4096', '1000']
        self.modules = ['cudatoolkit']
        self.num_gpus_per_node = 1
        self.sanity_patterns = sn.assert_found(r'time for single matrix vector multiplication', self.stdout)
        self.perf_patterns = {
            'perf': sn.extractsingle(r'Performance:\s+(?P<Gflops>\S+) Gflop/s', self.stdout, 'Gflops', float)
        }
        self.reference = {
            'daint:gpu': {'perf': (50.0, -0.1, 0.1)}
        }
        self.maintainers = ['you-can-type-your-email-here']
        self.tags = {'tutorial'}
```

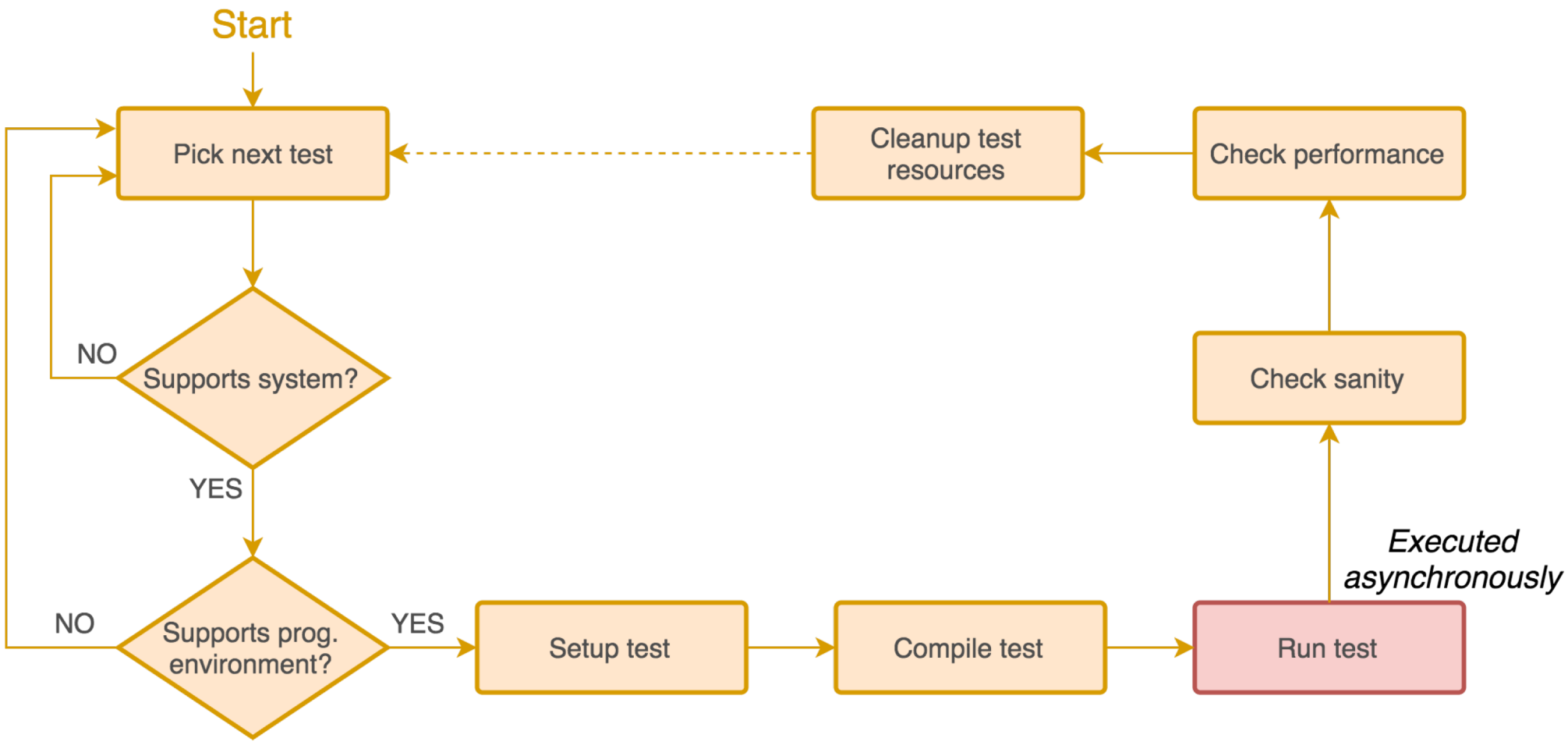
Sanity checking

Extract performance values from output

Reference values and perf. thresholds

The Regression Test Pipeline / How ReFrame Executes Tests

A series of well defined phases that each regression test goes through



The Regression Test Pipeline / How ReFrame Executes Tests

- Tests may skip some pipeline stages
  - Compile-only tests
  - Run-only tests
- Users may define additional actions before or after every pipeline stage by overriding the corresponding methods of the regression test API.
  - E.g., override the setup stage for customizing the behavior of the test per programming environment and/or system partition.

- Frontend passes through three phases and drives the execution of the tests
  1. Regression test discovery and loading
  2. Regression test selection (by name, tag, prog. environment support etc.)
  3. Regression test listing or execution

Running ReFrame

```
reframe -C /path/to/config.py -c /path/to/checks -r
```

- ReFrame uses three directories when running:
  1. Stage directory: Stores temporarily all the resources (static and generated) of the tests
     - Source code, input files, generated build script, generated job script, output etc.
     - This directory is removed if the test finishes successfully.
  2. Output directory: Keeps important files from the run for later reference
     - Job and build scripts, outputs and any user-specified files.
  3. Performance log directory: Keeps performance logs for the performance tests

- ReFrame generates a summary report at the end with detailed failure information.

Running ReFrame (sample output)

```
[==========] Running 1 check(s)
[==========] Started on Fri Sep  7 15:32:50 2018

[----------] started processing Example7Test (Matrix-vector multiplication using CUDA)
[ RUN      ] Example7Test on daint:gpu using PrgEnv-cray
[       OK ] Example7Test on daint:gpu using PrgEnv-cray
[ RUN      ] Example7Test on daint:gpu using PrgEnv-gnu
[       OK ] Example7Test on daint:gpu using PrgEnv-gnu
[ RUN      ] Example7Test on daint:gpu using PrgEnv-pgi
[       OK ] Example7Test on daint:gpu using PrgEnv-pgi
[----------] finished processing Example7Test (Matrix-vector multiplication using CUDA)

[  PASSED  ] Ran 3 test case(s) from 1 check(s) (0 failure(s))
[==========] Finished on Fri Sep  7 15:33:42 2018
```

Running ReFrame (sample failure)

```
[==========] Running 1 check(s)
[==========] Started on Fri Sep  7 16:40:12 2018

[----------] started processing Example7Test (Matrix-vector multiplication using CUDA)
[ RUN      ] Example7Test on daint:gpu using PrgEnv-gnu
[     FAIL ] Example7Test on daint:gpu using PrgEnv-gnu
[----------] finished processing Example7Test (Matrix-vector multiplication using CUDA)

[  FAILED  ] Ran 1 test case(s) from 1 check(s) (1 failure(s))
[==========] Finished on Fri Sep  7 16:40:22 2018

==============================================================================
SUMMARY OF FAILURES
------------------------------------------------------------------------------
FAILURE INFO for Example7Test
  * System partition: daint:gpu
  * Environment: PrgEnv-gnu
  * Stage directory: /path/to/stage/daint/gpu/PrgEnv-gnu/Example7Test
  * Job type: batch job (id=823427)
  * Maintainers: ['you-can-type-your-email-here']
  * Failing phase: performance
  * Reason: sanity error: 50.363125 is beyond reference value 70.0 (l=63.0, u=77.0)
------------------------------------------------------------------------------
```

Running ReFrame (examining a failure)

- ReFrame executes each test case from a separate stage directory:
  - `/path/to/stage/<system>/<partition>/<testname>/<environ>`

- Auto-generated build script and compilation's standard output/error
  - `rfm_<testname>_build.sh`
  - `rfm_<testname>_build.out`
  - `rfm_<testname>_build.err`

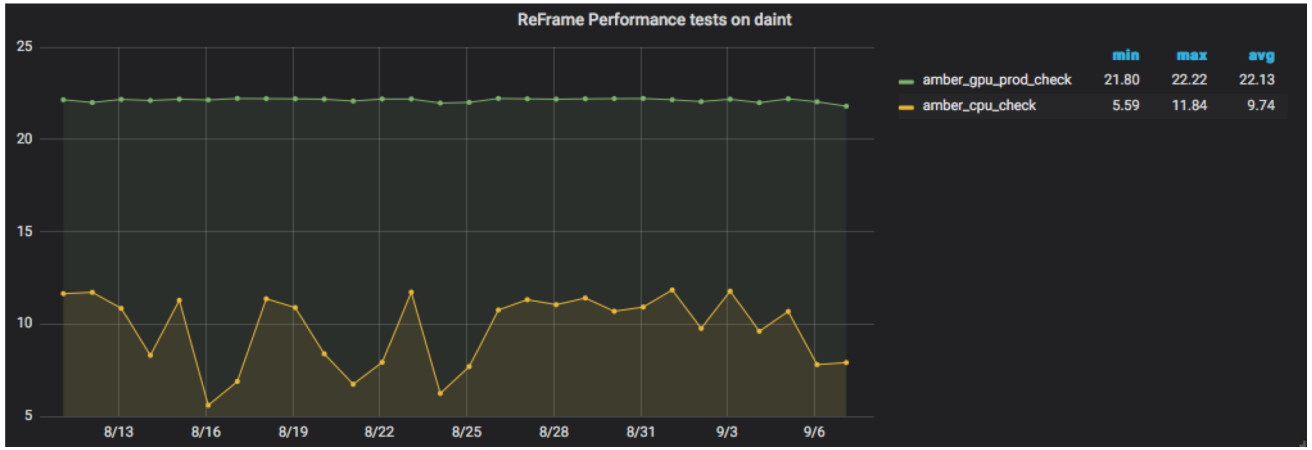
- Auto-generated job script and execution's standard output/error
  - `rfm_<testname>_job.sh`
  - `rfm_<testname>_job.out`
  - `rfm_<testname>_job.err`

Running ReFrame (examining performance logs)

- `/path/to/reframe/prefix/perflogs/<testname>.log`
  - A single file named after the test's name is updated every time the test is run
  - Log record output is fully configurable

```
2018-09-07T15:32:59|reframe 2.14-dev2|Example7Test on daint:gpu using PrgEnv-cray|jobid=823394|perf=49.71432|ref=50.0 (l=-0.1, u=0.1)
2018-09-07T15:33:11|reframe 2.14-dev2|Example7Test on daint:gpu using PrgEnv-gnu|jobid=823395|perf=50.1609|ref=50.0 (l=-0.1, u=0.1)
2018-09-07T15:33:42|reframe 2.14-dev2|Example7Test on daint:gpu using PrgEnv-pgi|jobid=823396|perf=51.078648|ref=50.0 (l=-0.1, u=0.1)
2018-09-07T16:40:22|reframe 2.14-dev2|Example7Test on daint:gpu using PrgEnv-gnu|jobid=823427|perf=50.363125|ref=70.0 (l=-0.1, u=0.1)
```

- ReFrame can also send logs to a Graylog server, where you can plot them with web tools.



CSCS
Centro Svizzero di Calcolo Scientifico
Swiss National Supercomputing Centre

ETH zürich

Using ReFrame with the CSCS CI service

Use Case

- Target code: Arbor Library
  - A library for implementing performance portable network simulations of multi-compartment neuron models.
  - https://github.com/eth-cscs/arbor

- Goals
  - Run the library's unit tests with...
    - different supported compilers (GNU, Clang)
    - different code configurations (base, GPU-enabled, SIMD-enabled)
  - Reuse the same ReFrame tests for other target systems
    - My laptop (where Clang is available)

Limitations

- The Jenkins VM is allowed to run only sbatch
- VM's environment is completely different from the target environment (Daint)

ReFrame cannot be run directly from the Jenkins VM, but...

we can configure it properly for Daint's compute nodes and sbatch it.

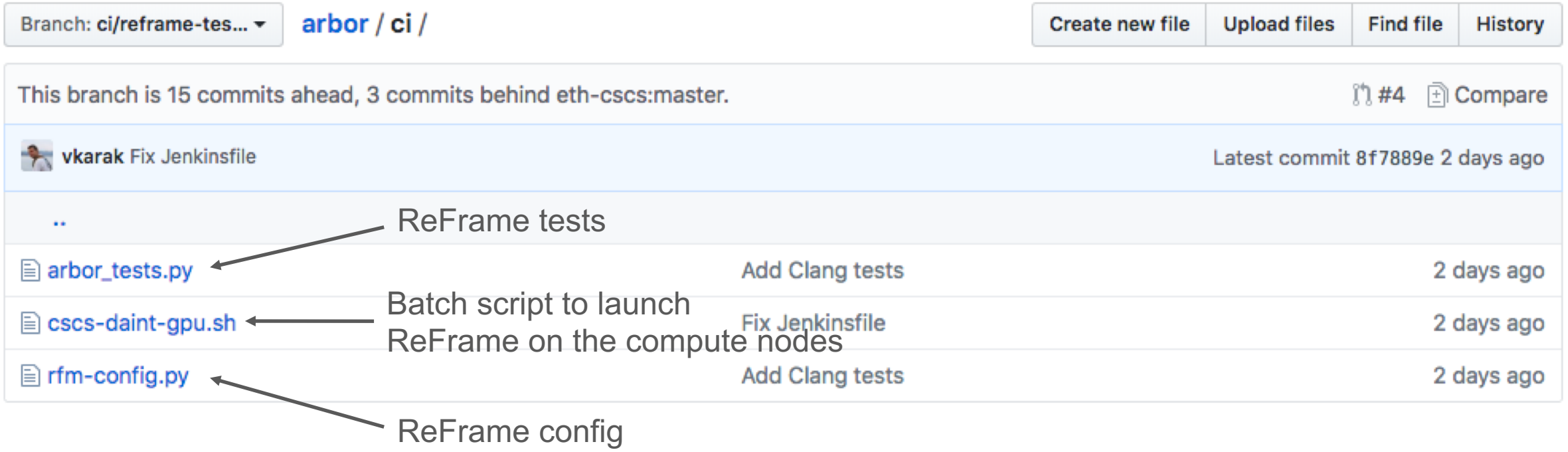
Configuration Steps

1. Set up the Jenkins project (as shown previously)

2. Prepare the code repository
   - Add a Jenkinsfile at the top-level directory
   - Add a batch script for running ReFrame on the compute nodes
   - Add the ReFrame tests
   - Add a ReFrame configuration file for the target systems

3. Open a pull request

Code Repository Set-up



Demo branch: https://github.com/vkarak/arbor/tree/ci/reframe-tests/ci

ReFrame Configuration

- Daint
  - Job scheduler: `local+srun` (i.e., do not submit any job, but use srun to parallel launch executables)
  - Programming environments: Cray, GNU, Intel, PGI
- My laptop
  - Job scheduler: `local` (i.e., do not submit any job and do not use any parallel launcher)
  - Programming environments: Clang
- Other systems (such as TravisCI's VMs) etc.

ReFrame Tests

```
68  @rfm.parameterized_test(['haswell'], ['broadwell'], ['native'])
69  class ArborSIMDTest(ArborBaseTest):
70      def __init__(self, arch_kind):
71          super().__init__()
72          if arch_kind == 'haswell':
73              self.valid_systems = ['daint:gpu']
74          elif arch_kind == 'broadwell':
75              self.valid_systems = ['daint:mc', 'tresa']
76          elif arch_kind == 'native':
77              self.valid_systems = ['tresa']
78
79          self.arch_kind = arch_kind
80
81      @property
82      def cmake_options(self):
83          return ['-DARB_WITH_ASSERTIONS=ON',
                    '-DARB_VECTORIZE=ON',
                    '-DARB_ARCH=%s' % self.arch_kind]
```

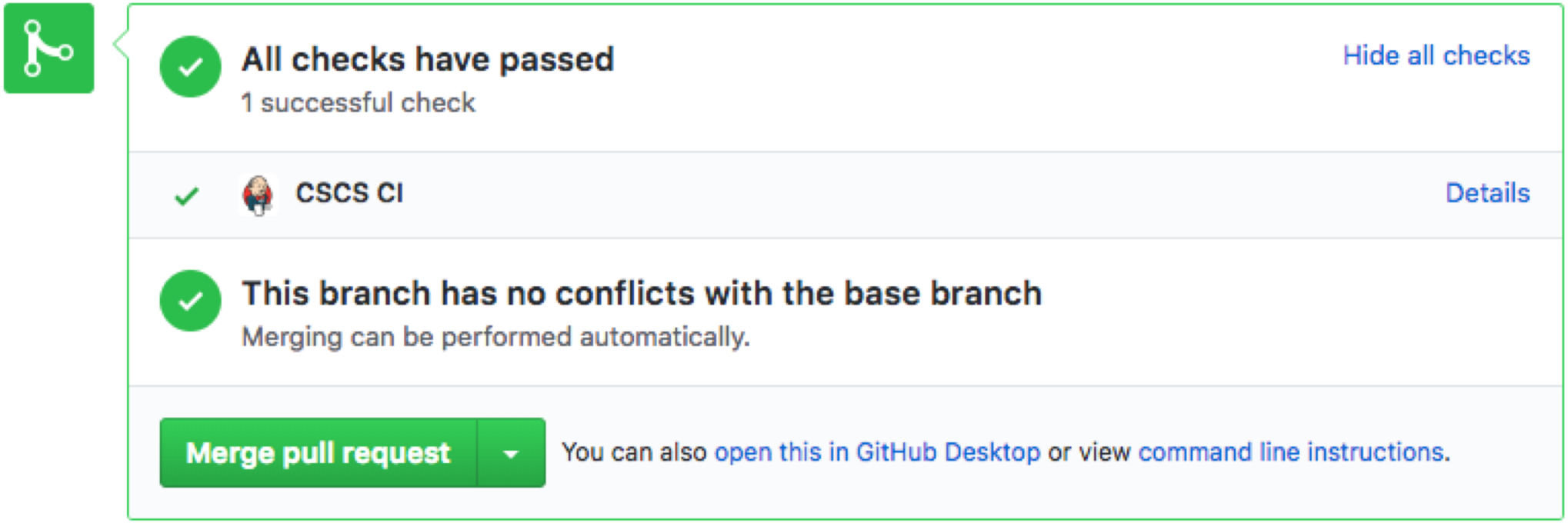
Use parameterized tests to create test factories!

You can use inheritance to avoid redefining common functionality!

```
rBaseTest):

_()
ems = ['daint:gpu', 'daint:mc']
_environs = ['PrgEnv-gnu']

elf):
WITH_ASSERTIONS=ON',
WITH_MPI=ON']
= [
_C_COMPILER=%s' % environ.cc,
_CXX_COMPILER=%s' % environ.cxx
ke_options
d_cmd = [
uild',
d',
. %s' % ' '.join(config_opts)

p(partition, environ, **job_opts)

36      @property
37      def cmake_options(self):
38          return ['-DARB_WITH_ASSERTIONS=ON']
```

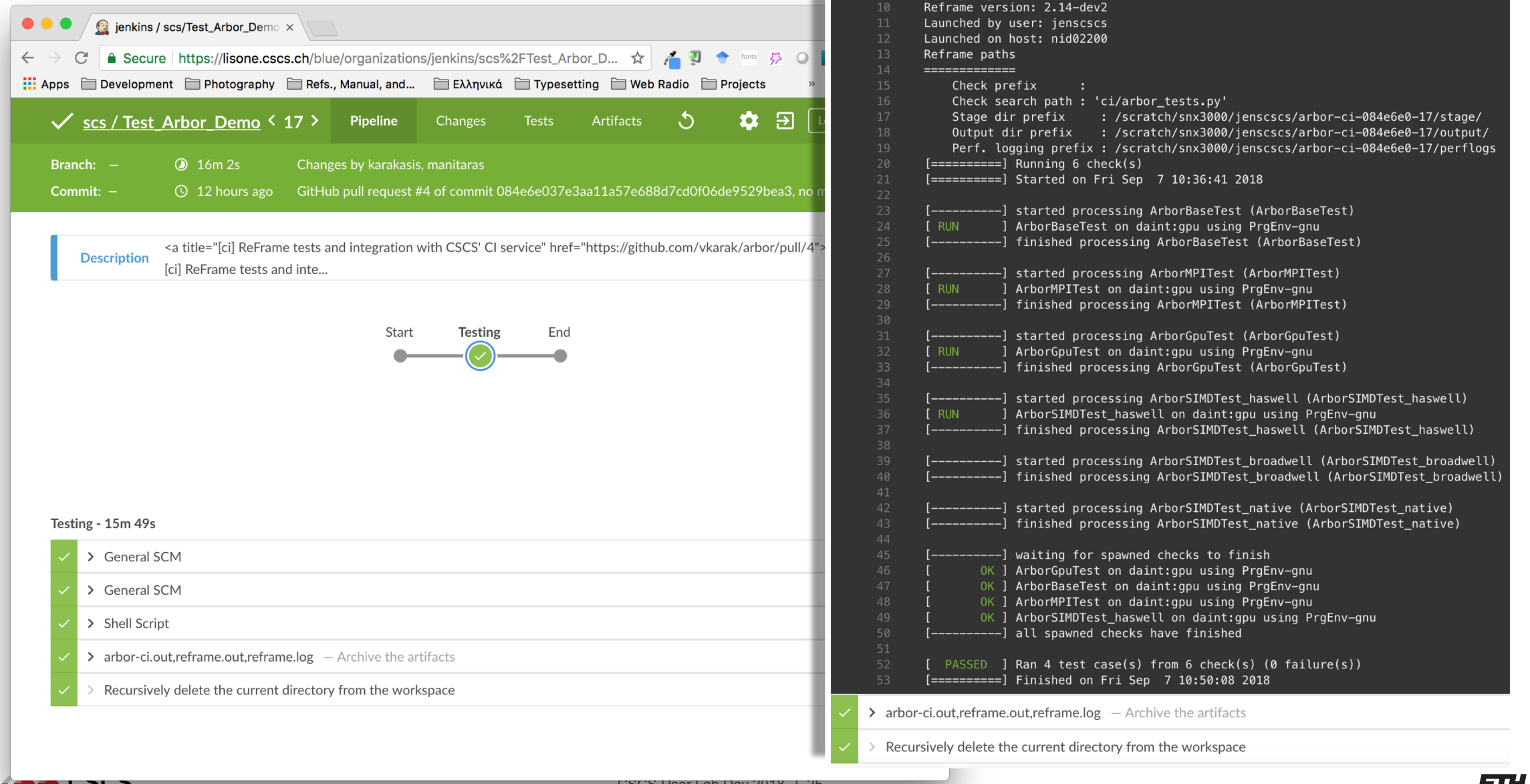
Open a Pull Request

- As soon as you open a PR or push to the PR, CSCS' CI will be triggered.



Demo PR: https://github.com/vkarak/arbor/pull/4

Examine ReFrame's Output Inside Jenkins

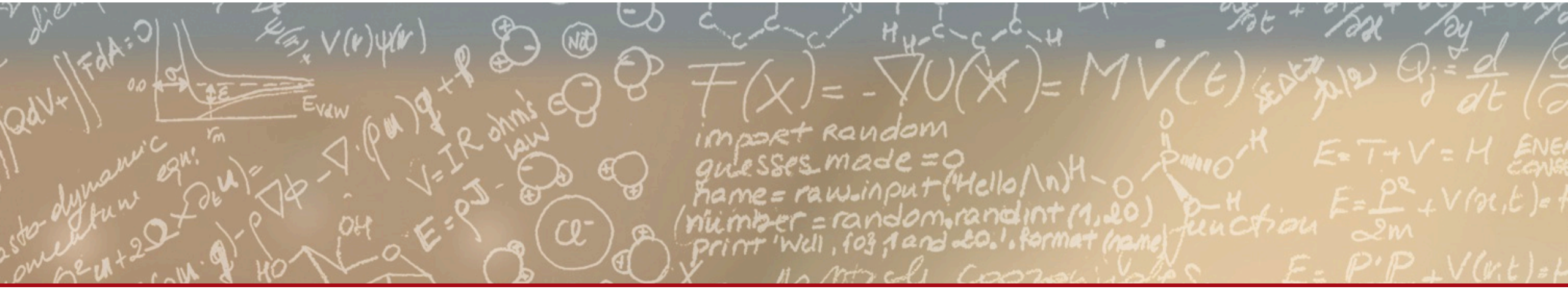


Conclusions and Future Directions

ReFrame is a powerful tool that allows you to leverage continuous integration in an HPC environment without having to deal with the low-level system interaction details.

- High-level tests written in Python
- Portability across HPC system platforms
- Comprehensive reports and reproducible methods

- ReFrame is being actively developed with a monthly release-cycle.
- Future directions
  - Full integration with CSCS' CI service (no need for the intermediate submission script)
  - Test dependencies
  - Support for containers
  - Frontend and API improvements
- Bug reports, feature requests, help @ https://github.com/eth-cscs/reframe



Thank you for your attention.