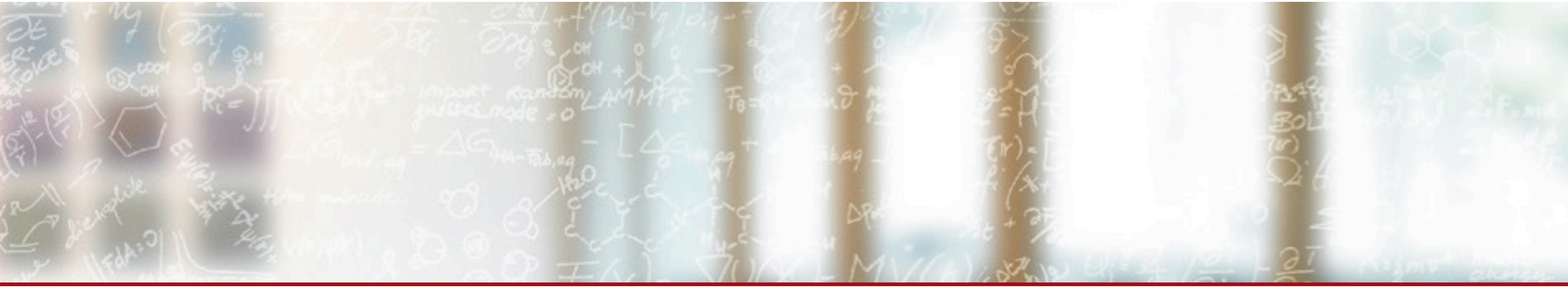
CSCS
Centro Svizzero di Calcolo Scientifico
Swiss National Supercomputing Centre

ETHzürich



ReFrame: A Regression Testing and Continuous Integration Framework for HPC systems

Fifth Annual Workshop on HPC User Support Tools – SC18

Vasileios Karakasis, CSCS

November 11, 2018

Table of Contents

1. Overview of ReFrame

2. Use Cases

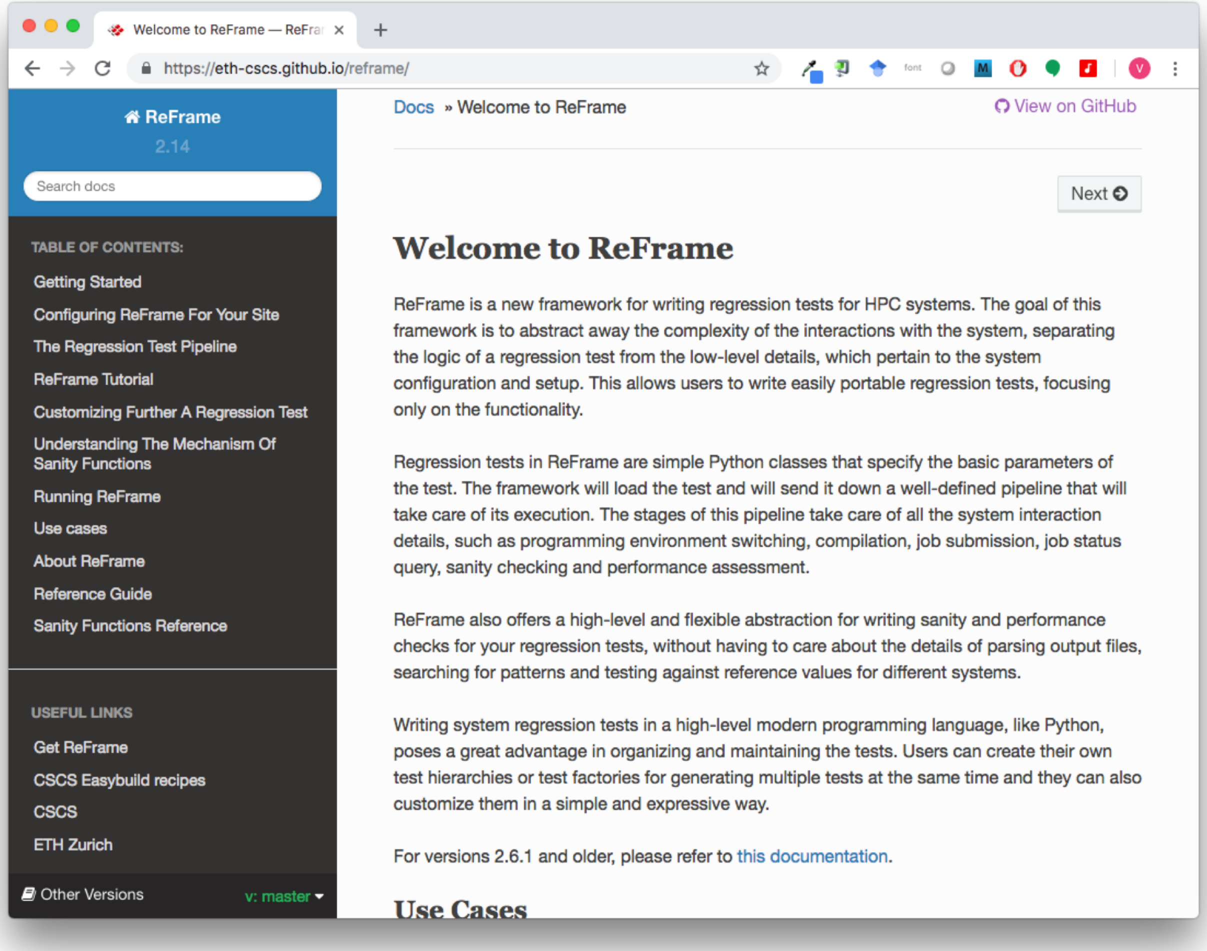
3. Demo

4. Q & A

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.


Welcome to ReFrame — ReFra
https://eth-cscs.github.io/reframe/
ReFrame
2.14
Search docs
TABLE OF CONTENTS:
Getting Started
Configuring ReFrame For Your Site
The Regression Test Pipeline
ReFrame Tutorial
Customizing Further A Regression Test
Understanding The Mechanism Of Sanity Functions
Running ReFrame
Use cases
About ReFrame
Reference Guide
Sanity Functions Reference
USEFUL LINKS
Get ReFrame
CSCS Easybuild recipes
CSCS
ETH Zurich
Other Versions
v: master
Docs » Welcome to ReFrame
View on GitHub
Next

Welcome to ReFrame

ReFrame is a new framework for writing regression tests for HPC systems. The goal of this framework is to abstract away the complexity of the interactions with the system, separating the logic of a regression test from the low-level details, which pertain to the system configuration and setup. This allows users to write easily portable regression tests, focusing only on the functionality.

Regression tests in ReFrame are simple Python classes that specify the basic parameters of the test. The framework will load the test and will send it down a well-defined pipeline that will take care of its execution. The stages of this pipeline take care of all the system interaction details, such as programming environment switching, compilation, job submission, job status query, sanity checking and performance assessment.

ReFrame also offers a high-level and flexible abstraction for writing sanity and performance checks for your regression tests, without having to care about the details of parsing output files, searching for patterns and testing against reference values for different systems.

Writing system regression tests in a high-level modern programming language, like Python, poses a great advantage in organizing and maintaining the tests. Users can create their own test hierarchies or test factories for generating multiple tests at the same time and they can also customize them in a simple and expressive way.

For versions 2.6.1 and older, please refer to this documentation.

Use Cases

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

Design Goals

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

Write once, test everywhere!

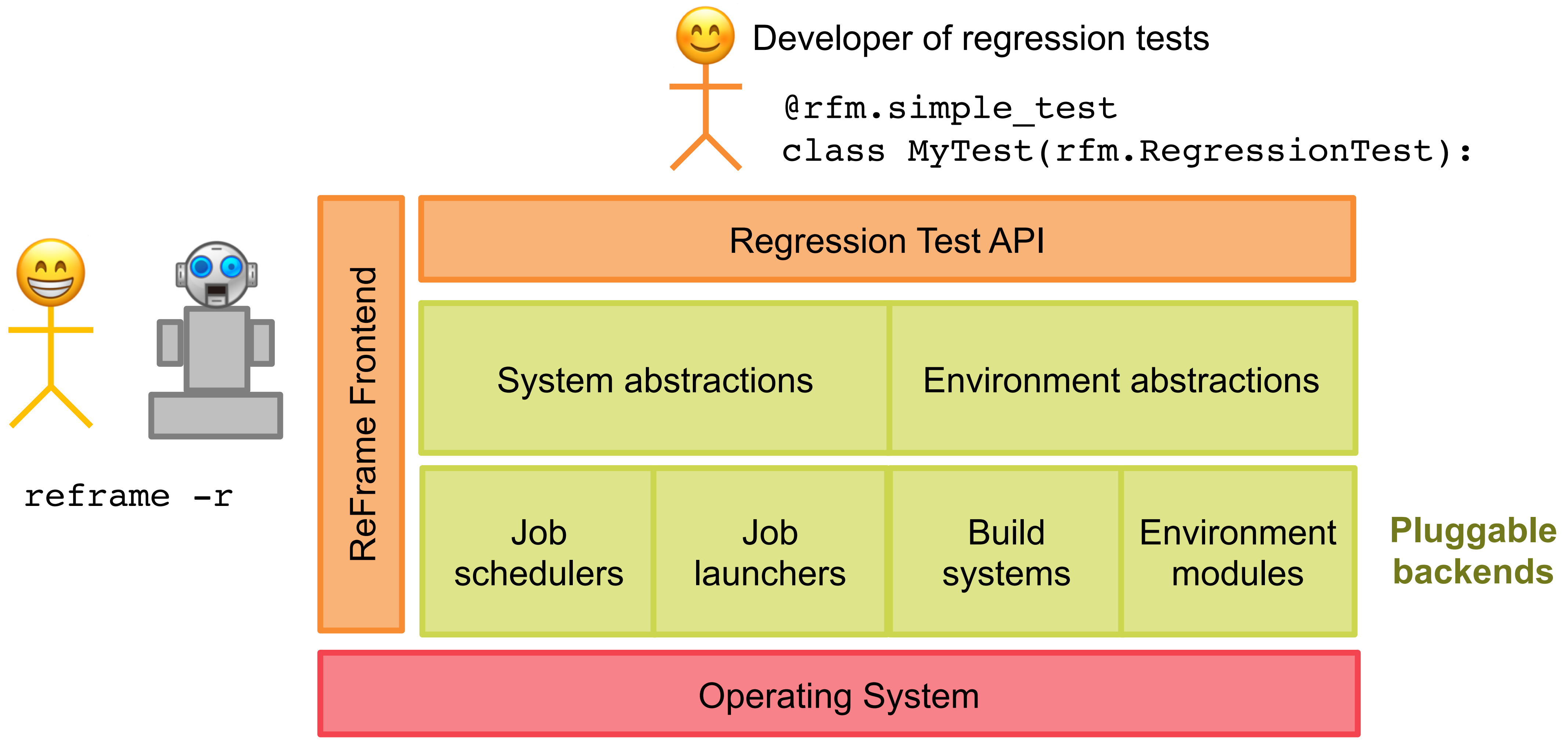
Key Features

- 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

More Features

- Multiple workload manager backends
  - SLURM
  - PBS/Torque
- Multiple parallel launcher backends
  - srun, mpirun, mpiexec etc.
- Multiple environment modules backends
  - Tmod, Tmod4, Lmod
- Build system backends
  - CMake, Autotools, Make
- Asynchronous execution of regression tests
- Complete documentation (tutorials, reference guide)
- ... and more (*https://github.com/eth-cscs/reframe*)

ReFrame’s architecture



Developer of regression tests

```
@rfm.simple_test
class MyTest(rfm.RegressionTest):
```

```
reframe -r
```

ReFrame Frontend

Regression Test API

System abstractions

Environment abstractions

Job schedulers

Job launchers

Build systems

Environment modules

Pluggable backends

Operating System

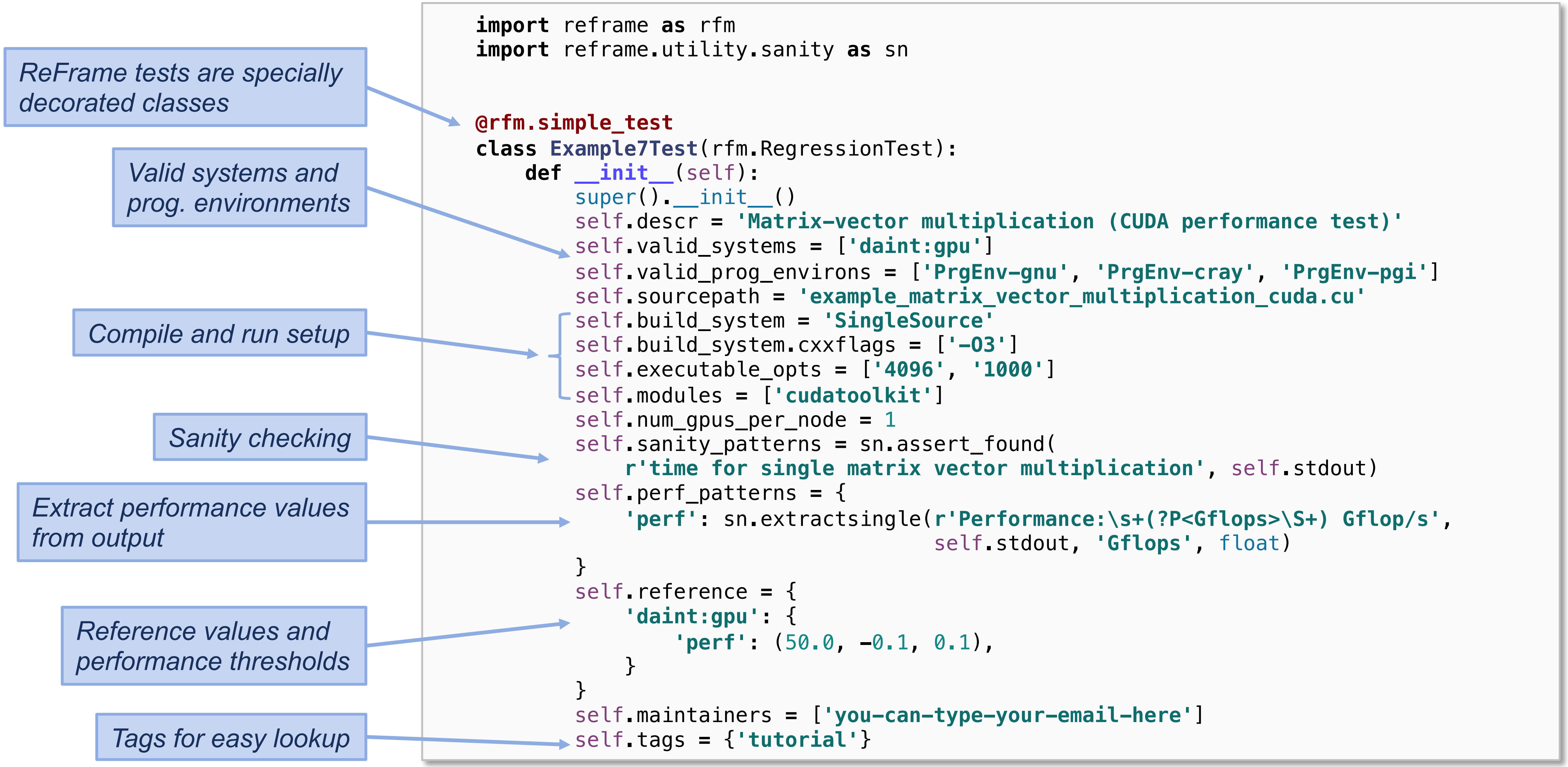
Writing a Regression Test in ReFrame

A regression test writer should not care about...

- How to access 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 Test in ReFrame



ReFrame tests are specially decorated classes

Valid systems and prog. environments

Compile and run setup

Sanity checking

Extract performance values from output

Reference values and performance thresholds

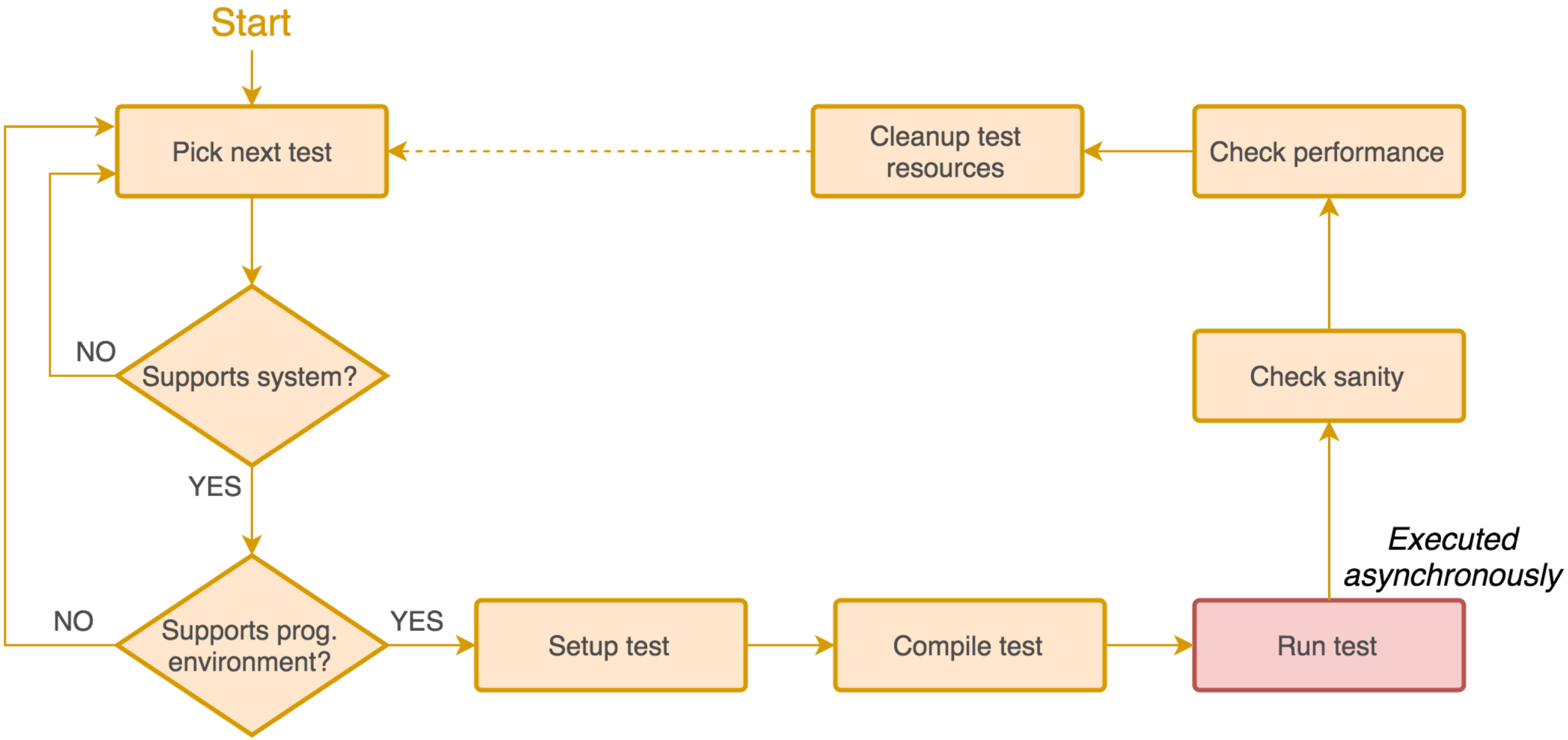
Tags for easy lookup

```
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'}
```

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`

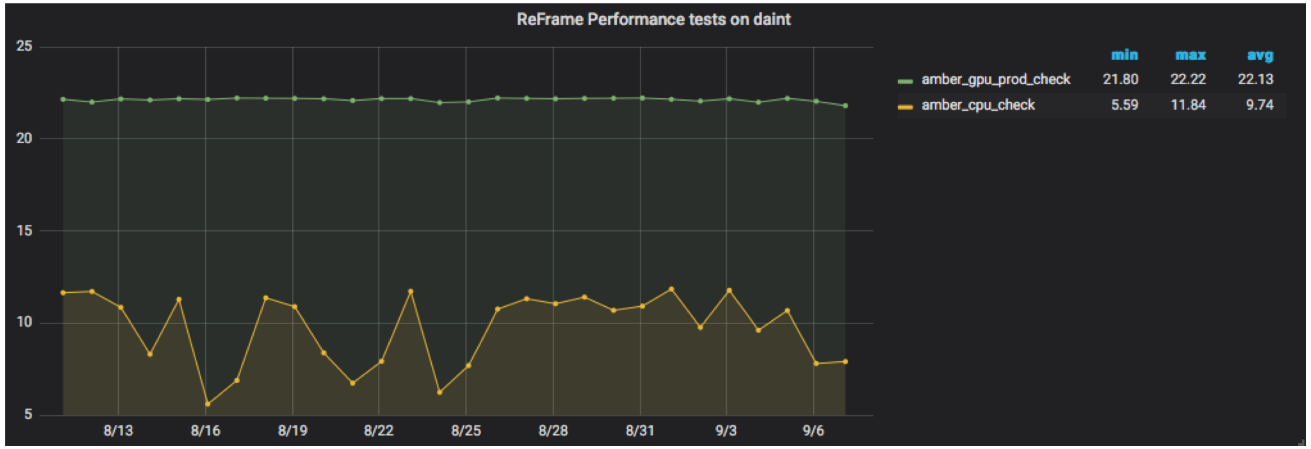
- Auto-generated job script and execution’s standard output/error
  - `rfm_<testname>_job.sh`
  - `rfm_<testname>_job.out`
  - `rfm_<testname>_job.err`

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.



Using ReFrame at CSCS

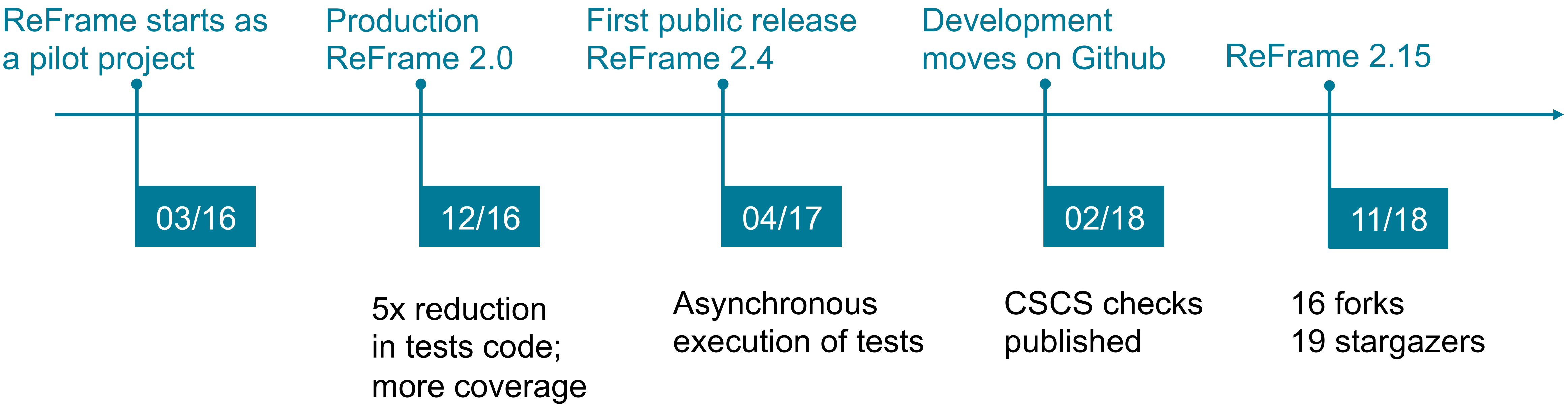
Background

- CSCS had a shell-script based regression suite
  - Tests very tightly coupled to system details
  - Lots of code replication across tests
  - 15K lines of test code

- Simple changes required significant team effort
  - Porting all tests to native Slurm took several weeks

- Fixing even simple bugs was a tedious task
  - Tens of regression test files had to be fixed

Timeline / ReFrame Evolution



ReFrame starts as a pilot project

03/16

Production ReFrame 2.0

12/16

5x reduction in tests code; more coverage

First public release ReFrame 2.4

04/17

Asynchronous execution of tests

Development moves on Github

02/18

CSCS checks published

ReFrame 2.15

11/18

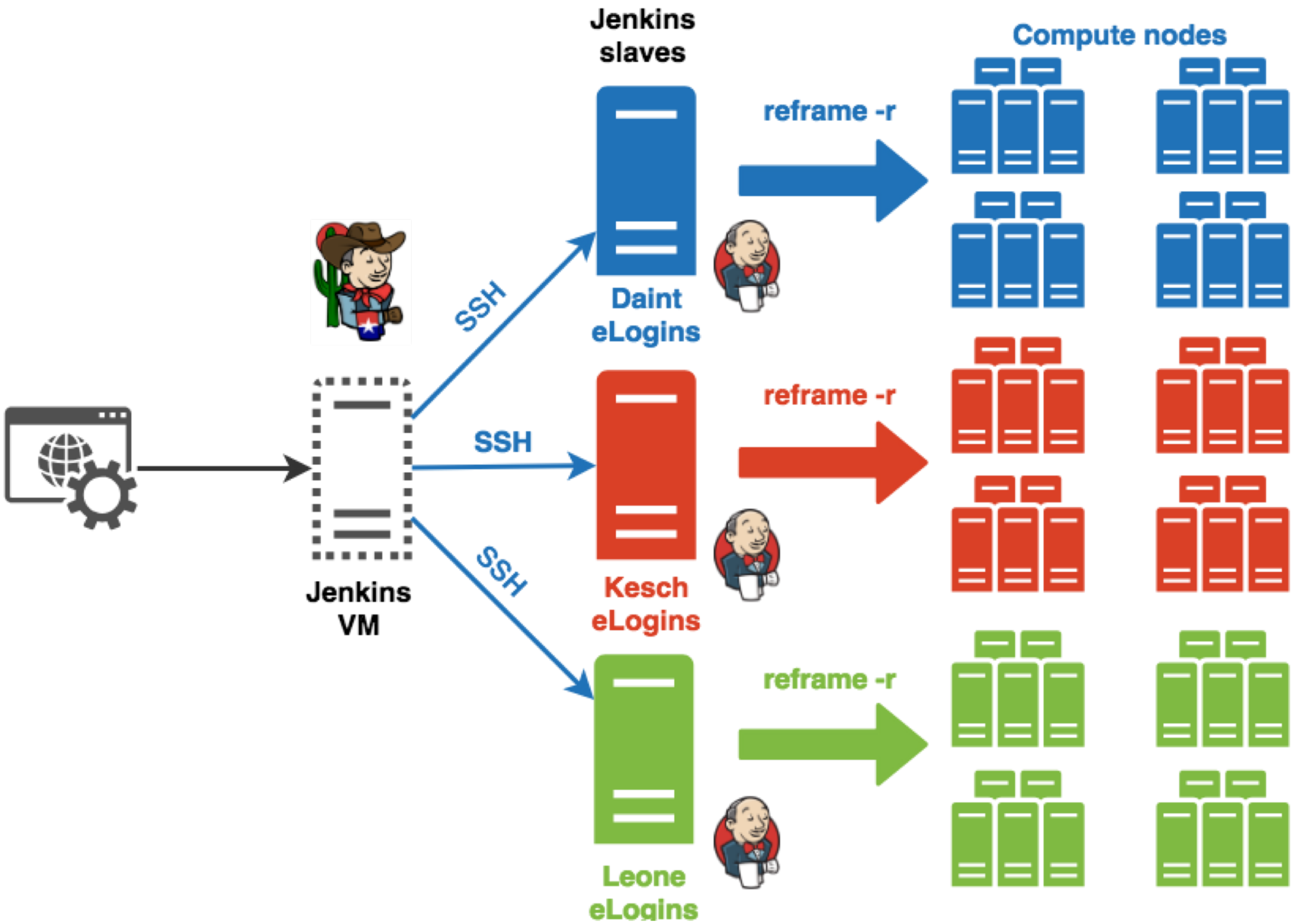
16 forks
19 stargazers

ReFrame @ CSCS / Tests

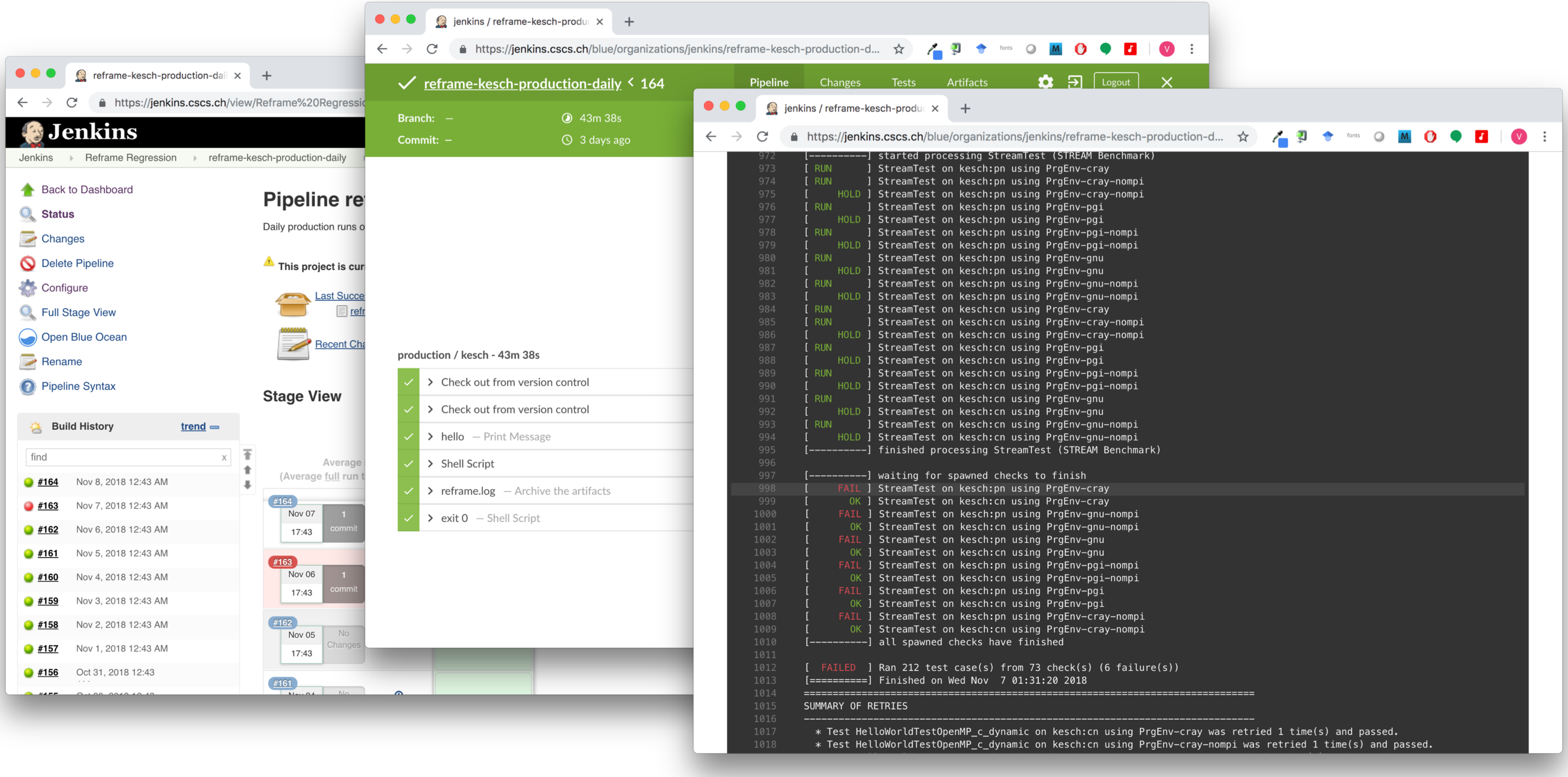
- Used for continuously testing systems in production
  - Piz Daint: 179 tests
  - Piz Kesch: 75 tests
  - Leone: 45 tests
  - **Total: 241 different tests (reused across systems)**

- Three categories of tests
  1. Production (90min)
     - Applications, libraries, programming environments, profiling tools, debuggers, microbenchmarks
     - Sanity and performance
     - Run nightly by Jenkins
  2. Maintenance (10min)
     - Programming environment sanity and key user applications performance
     - Before/after maintenance sessions
  3. Diagnostics

ReFrame @ CSCS / Production set-up



ReFrame @ CSCS / Production set-up



reframe-kesch-production-dai ×
https://jenkins.cscs.ch/view/Reframe%20Regressio
Jenkins
Jenkins › Reframe Regression › reframe-kesch-production-daily
Back to Dashboard
Status
Changes
Delete Pipeline
Configure
Full Stage View
Open Blue Ocean
Rename
Pipeline Syntax
Build History
trend
find

Pipeline re

Daily production runs o

This project is cur

Last Succe

refr

Recent Cha

Stage View

Average
(Average full run t

#164	Nov 8, 2018 12:43 AM
#163	Nov 7, 2018 12:43 AM
#162	Nov 6, 2018 12:43 AM
#161	Nov 5, 2018 12:43 AM
#160	Nov 4, 2018 12:43 AM
#159	Nov 3, 2018 12:43 AM
#158	Nov 2, 2018 12:43 AM
#157	Nov 1, 2018 12:43 AM
#156	Oct 31, 2018 12:43

#164	Nov 07 17:43	1 commit
#163	Nov 06 17:43	1 commit
#162	Nov 05 17:43	No Changes
#161		

jenkins / reframe-kesch-produ ×
https://jenkins.cscs.ch/blue/organizations/jenkins/reframe-kesch-production-d...
Pipeline
Changes
Tests
Artifacts
Logout

✓ reframe-kesch-production-daily ‹ 164

Branch: — 43m 38s

Commit: — 3 days ago

production / kesch - 43m 38s

- ✓ › Check out from version control
- ✓ › Check out from version control
- ✓ › hello — Print Message
- ✓ › Shell Script
- ✓ › reframe.log — Archive the artifacts
- ✓ › exit 0 — Shell Script

jenkins / reframe-kesch-produ ×
https://jenkins.cscs.ch/blue/organizations/jenkins/reframe-kesch-production-d...

```
[----------] started processing StreamTest (STREAM Benchmark)
[ RUN      ] StreamTest on kesch:pn using PrgEnv-cray
[ RUN      ] StreamTest on kesch:pn using PrgEnv-cray-nompi
[     HOLD ] StreamTest on kesch:pn using PrgEnv-cray-nompi
[ RUN      ] StreamTest on kesch:pn using PrgEnv-pgi
[     HOLD ] StreamTest on kesch:pn using PrgEnv-pgi
[ RUN      ] StreamTest on kesch:pn using PrgEnv-pgi-nompi
[     HOLD ] StreamTest on kesch:pn using PrgEnv-pgi-nompi
[ RUN      ] StreamTest on kesch:pn using PrgEnv-gnu
[     HOLD ] StreamTest on kesch:pn using PrgEnv-gnu
[ RUN      ] StreamTest on kesch:pn using PrgEnv-gnu-nompi
[     HOLD ] StreamTest on kesch:pn using PrgEnv-gnu-nompi
[ RUN      ] StreamTest on kesch:cn using PrgEnv-cray
[ RUN      ] StreamTest on kesch:cn using PrgEnv-cray-nompi
[     HOLD ] StreamTest on kesch:cn using PrgEnv-cray-nompi
[ RUN      ] StreamTest on kesch:cn using PrgEnv-pgi
[     HOLD ] StreamTest on kesch:cn using PrgEnv-pgi
[ RUN      ] StreamTest on kesch:cn using PrgEnv-pgi-nompi
[     HOLD ] StreamTest on kesch:cn using PrgEnv-pgi-nompi
[ RUN      ] StreamTest on kesch:cn using PrgEnv-gnu
[     HOLD ] StreamTest on kesch:cn using PrgEnv-gnu
[ RUN      ] StreamTest on kesch:cn using PrgEnv-gnu-nompi
[     HOLD ] StreamTest on kesch:cn using PrgEnv-gnu-nompi
[----------] finished processing StreamTest (STREAM Benchmark)

[----------] waiting for spawned checks to finish
[     FAIL ] StreamTest on kesch:pn using PrgEnv-cray
[       OK ] StreamTest on kesch:cn using PrgEnv-cray
[     FAIL ] StreamTest on kesch:pn using PrgEnv-gnu-nompi
[       OK ] StreamTest on kesch:cn using PrgEnv-gnu-nompi
[     FAIL ] StreamTest on kesch:pn using PrgEnv-gnu
[       OK ] StreamTest on kesch:cn using PrgEnv-gnu
[     FAIL ] StreamTest on kesch:pn using PrgEnv-pgi-nompi
[       OK ] StreamTest on kesch:cn using PrgEnv-pgi-nompi
[     FAIL ] StreamTest on kesch:pn using PrgEnv-pgi
[       OK ] StreamTest on kesch:cn using PrgEnv-pgi
[     FAIL ] StreamTest on kesch:pn using PrgEnv-cray-nompi
[       OK ] StreamTest on kesch:cn using PrgEnv-cray-nompi
[----------] all spawned checks have finished

[  FAILED  ] Ran 212 test case(s) from 73 check(s) (6 failure(s))
[==========] Finished on Wed Nov  7 01:31:20 2018
==============================================================================
SUMMARY OF RETRIES
------------------------------------------------------------------------------
  * Test HelloWorldTestOpenMP_c_dynamic on kesch:cn using PrgEnv-cray was retried 1 time(s) and passed.
  * Test HelloWorldTestOpenMP_c_dynamic on kesch:cn using PrgEnv-cray-nompi was retried 1 time(s) and passed.
```

Conclusions and Future Directions

ReFrame is a powerful tool that allows you to continuously test 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

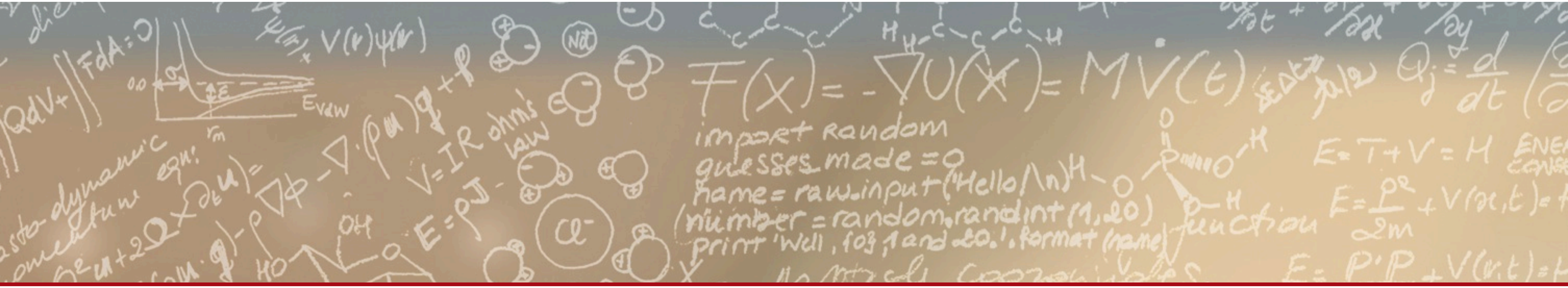
- ReFrame is being actively developed with a regular release cycle.
- Future directions
  - Test dependencies
  - Remote and asynchronous compilation of tests
- Bug reports, feature requests, help @ https://github.com/eth-cscs/reframe

Acknowledgements

- Framework contributions
  - Andreas Jocksch
  - Matthias Kraushaar
  - Rafael Sarmiento
  - Samuel Omlin
  - Theofilos Manitaras
  - Victor Holanda

- Regression tests
  - SCS and OPS team

ReFrame Demo (link)


F(X) = -∇U(X) = MV(t)
import random
guesses_made = 0
name = raw_input('Hello\n')
number = random.randint(1, 20)
print 'Well, {0}, I and 20.'.format(name)
V=IR ohm's law
E = T+V = H
function

Thank you for your attention.