



CSCS

Centro Svizzero di Calcolo Scientifico
Swiss National Supercomputing Centre

ETH zürich



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

FOSDEM'19

Victor Holanda Rusu, CSCS

February 3rd, 2019

 reframe@sympa.cscs.ch

 <https://eth-cscs.github.io/reframe>

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

 <https://reframe-slack.herokuapp.com>

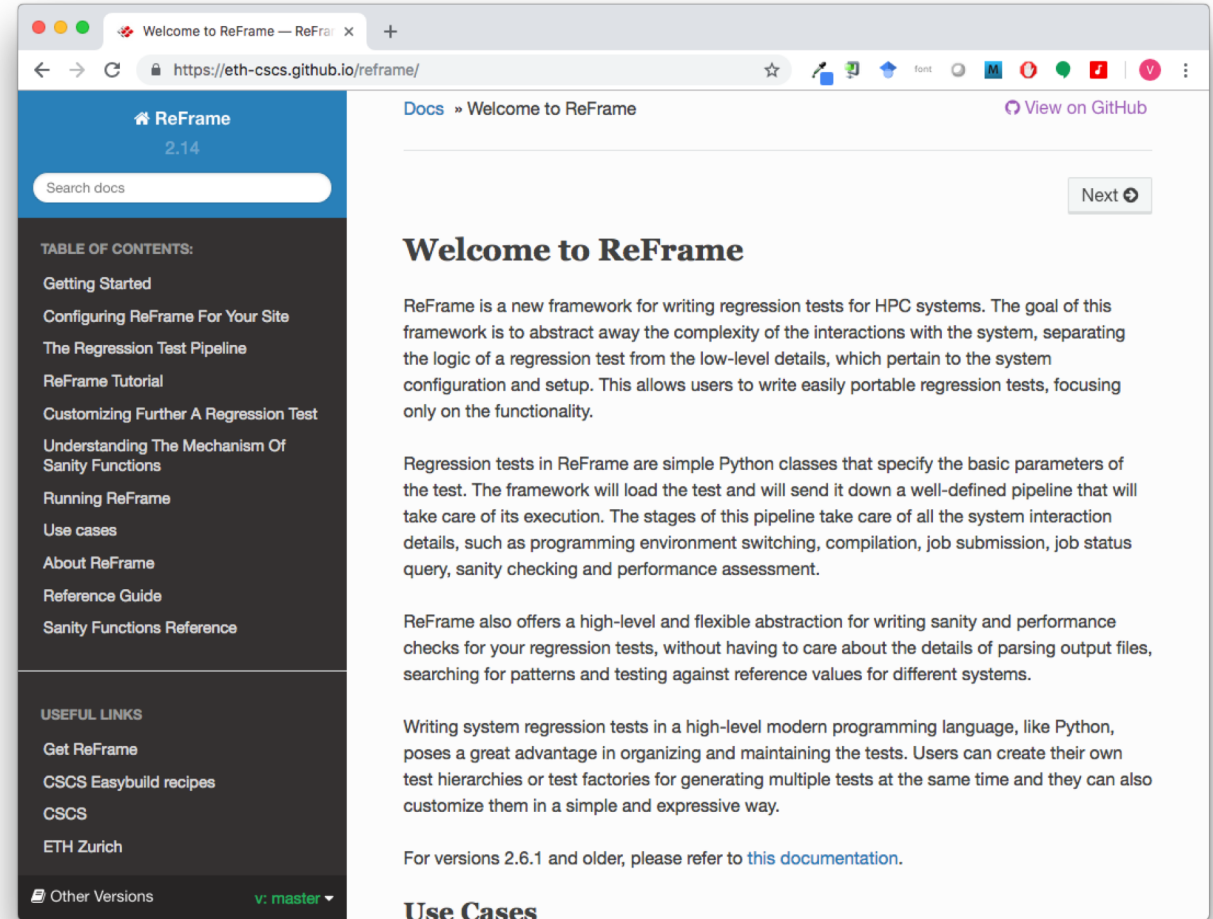
Background

- 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

What is ReFrame?

A new regression testing framework that

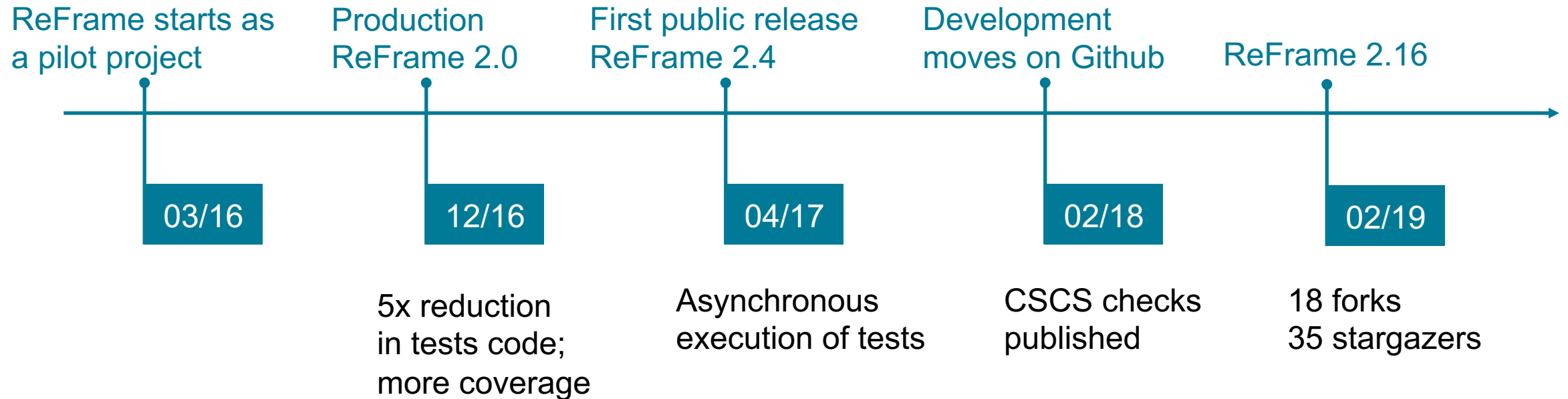
- allows writing **portable HPC** regression tests in Python,
- **abstracts away** the system interaction details,
- lets users focus solely on the **logic** of their test.



 <https://eth-cscs.github.io/reframe>

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

Timeline / ReFrame Evolution



Design Goals

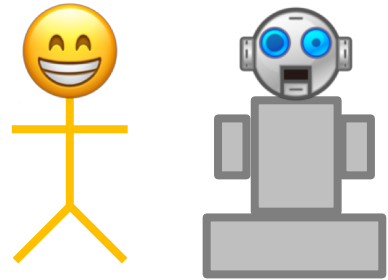
- Productivity
- Portability
- Speed and Ease of Use
- Robustness

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
- Complete documentation (tutorials, reference guide)
- ... and more (<https://github.com/eth-cscs/reframe>)

ReFrame's architecture

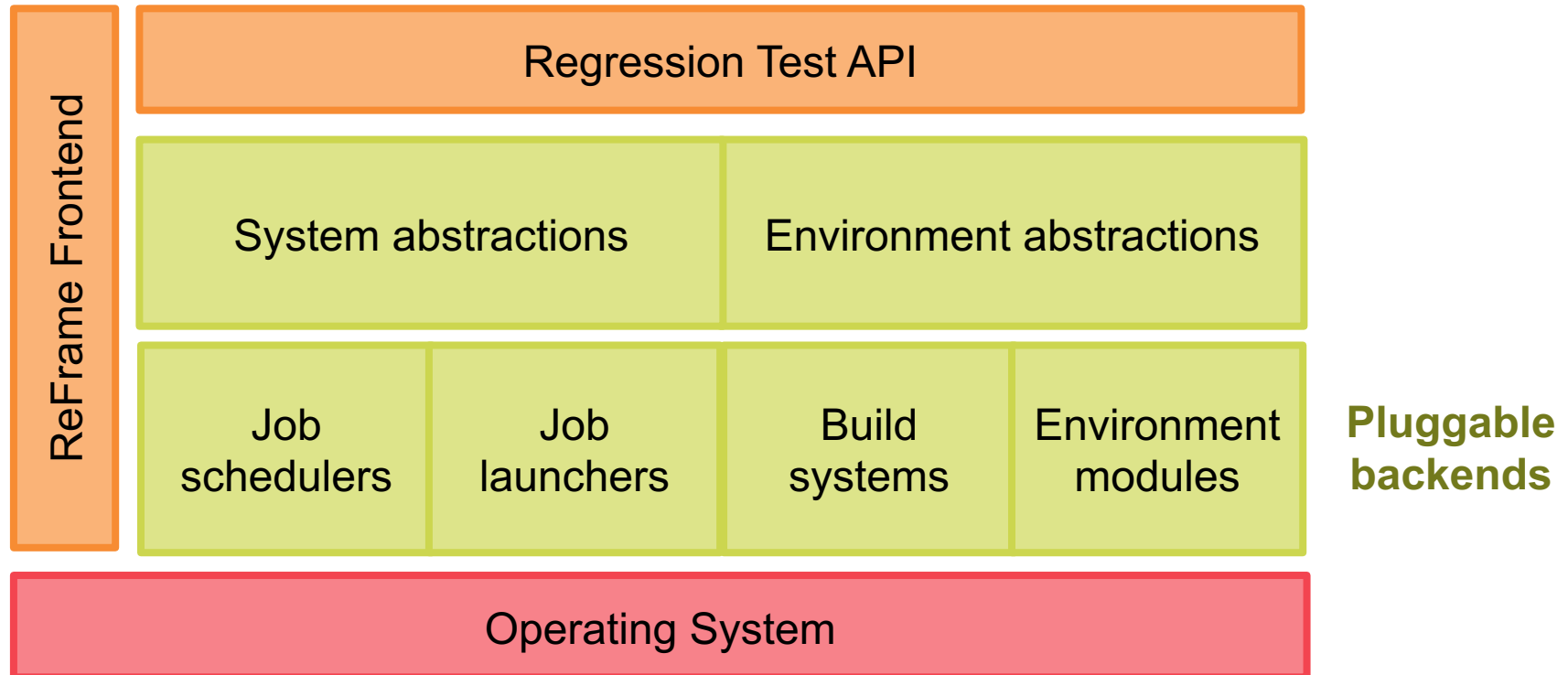


reframe -r



Developer of regression tests

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



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'}
```

Writing a Regression Test in ReFrame

```
@rfm.simple_test
class ArborBaseTest(rfm.RegressionTest):
    def __init__(self):
```

```
47 @rfm.parameterized_test(['haswell'], ['broadwell'], ['native'])
48 class ArborSIMDTest(ArborBaseTest):
49     def __init__(self, arch_kind):
50         super().__init__()
51         if arch_kind == 'haswell':
52             self.valid_systems = ['daint:gpu']
53         elif arch_kind == 'broadwell':
54             self.valid_systems = ['daint:mc', 'tresa']
55         elif arch_kind == 'native':
56             self.valid_systems = ['tresa']
57
58         self.arch_kind = arch_kind
59         self.build_system.config_opts += ['-DARB_VECTORIZE=ON',
                                           '-DARB_ARCH=%s' % self.arch_kind]
```

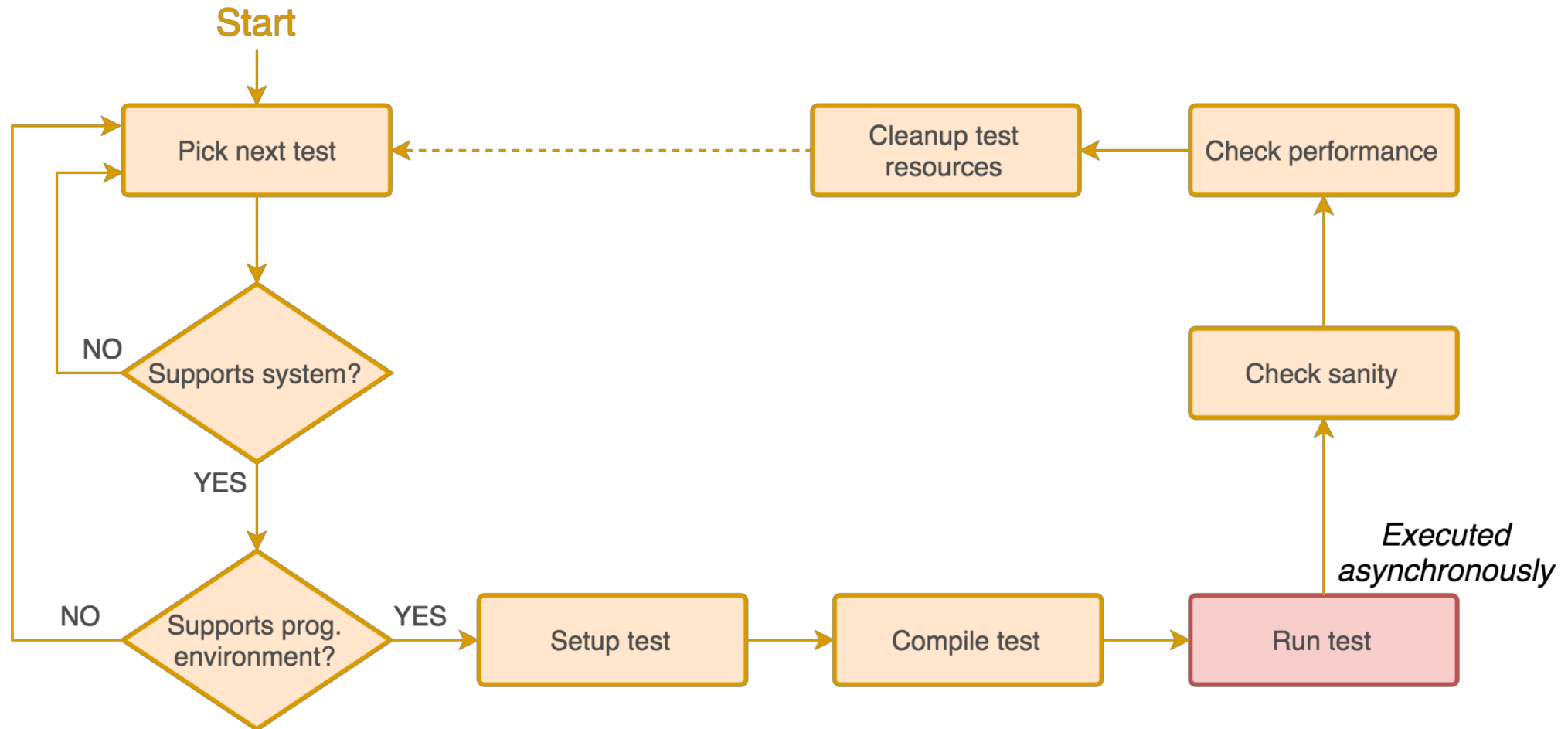
**Use parameterized tests
to create test factories!**

**Use inheritance to avoid
repeating common functionality!**

```
nt:gpu', 'daint:mc']
['PrgEnv-gnu']
pts += ['-DARB_WITH_MPI=ON']
```

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

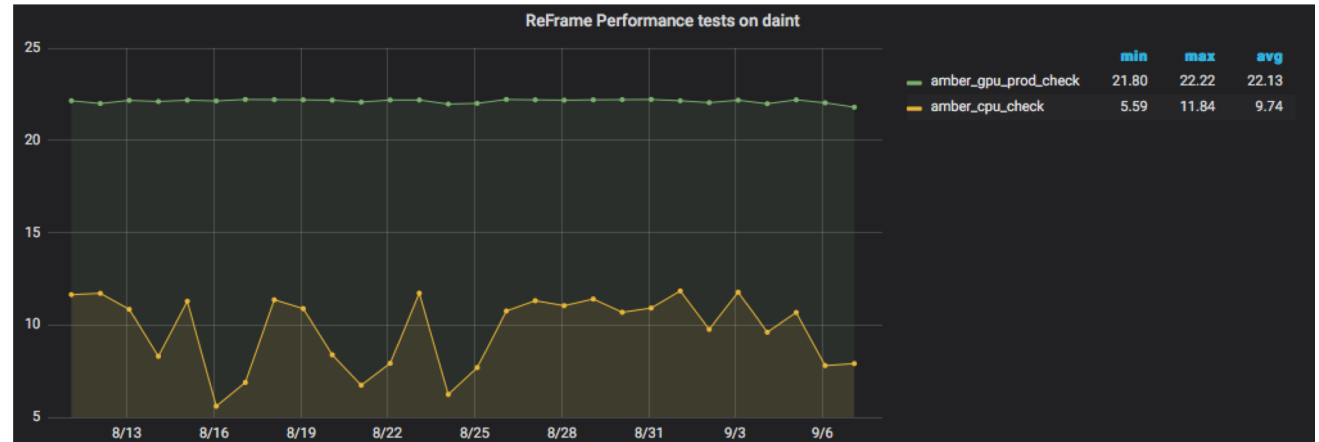
- * System partition: daint:gpu
 - * Environment: PrgEnv-gnu
 - * Stage directory: /path/to/stage/daint/gpu/PrgEnv-gnu/Example7Test
 - * Job type: batch job (id=823427)
 - * Maintainers: ['you-can-type-your-email-here']
 - * Failing phase: performance
 - * Reason: sanity error: 50.363125 is beyond reference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

Running ReFrame (examining 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-gnul|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-gnul|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.

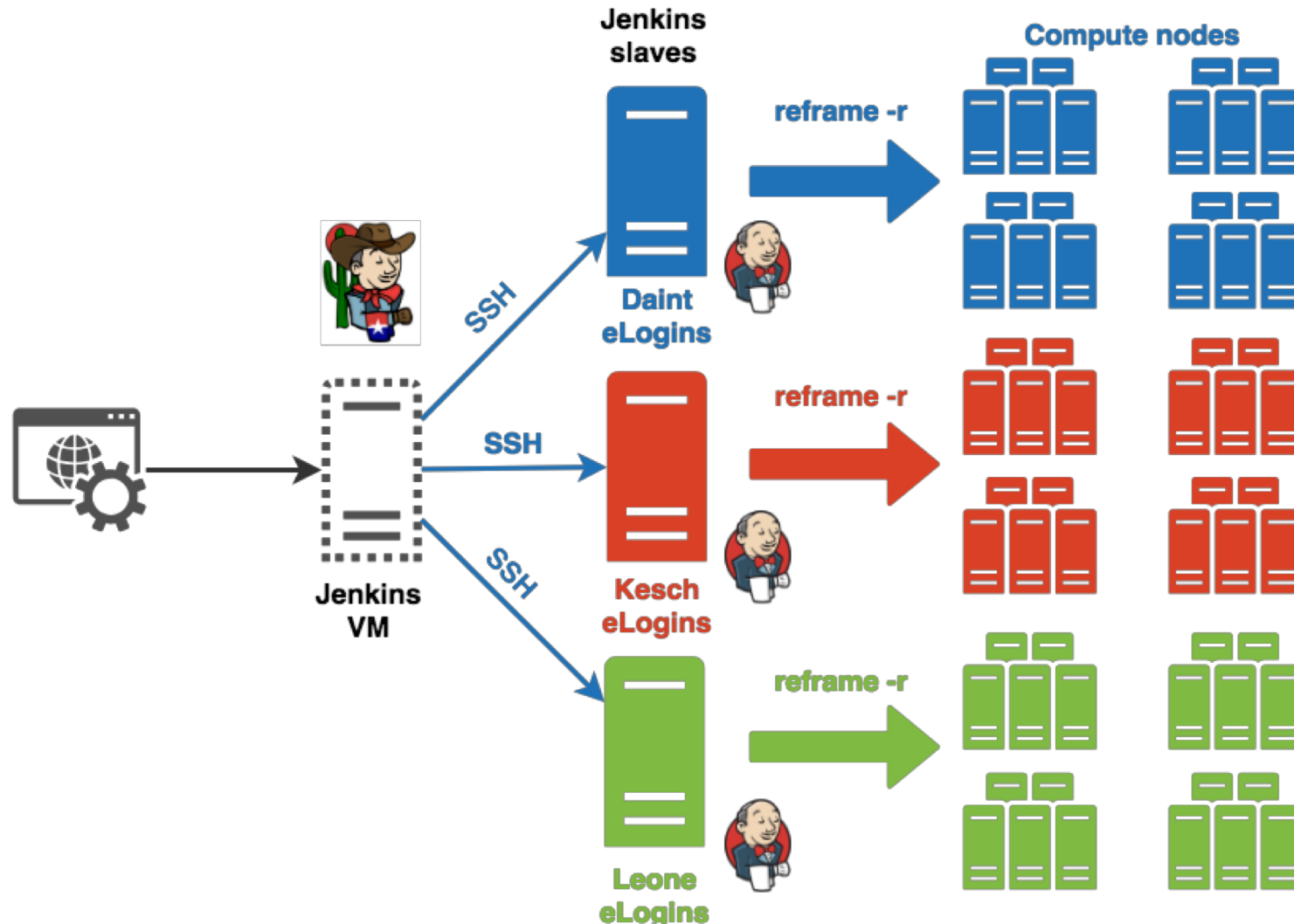


Using ReFrame at CSCS

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

The image displays three overlapping screenshots of the Jenkins web interface, illustrating the production set-up for ReFrame at CSCS.

- Left Screenshot:** Shows the Jenkins dashboard for the 'reframe-kesch-production-daily' pipeline. The 'Build History' table lists recent builds, with build #164 being the most recent (Nov 8, 2018, 12:43 AM). The 'Pipeline View' section shows the pipeline's status and configuration options.
- Middle Screenshot:** Shows the 'Pipeline View' for build #164, which completed successfully. The 'Stage View' section displays the build's progress, including the 'production / kesch - 43m 38s' stage. The 'Build History' table is also visible, showing the build's status and duration.
- Right Screenshot:** Shows the 'Log' view for build #164, displaying the output of the 'StreamTest' stage. The log indicates that the tests passed, with a summary of retries at the bottom.



CSCS

Centro Svizzero di Calcolo Scientifico
Swiss National Supercomputing Centre

ETH zürich

Using ReFrame with a CI service

ReFrame integration with CI service

- CSCS CI service
 - Based on Jenkins
 - Run on CSCS HPC systems
 - On the remote side there is a Jenkins VM that can only run sbatch to the compute nodes
 - Integration steps
 1. Add a Jenkinsfile to project
 2. Add a batch script for running ReFrame on the compute nodes
 3. Add configuration entry for the target systems
 4. Add ReFrame tests
- Travis – Github
 - Runs a VM on the cloud
 - Integration steps
 1. Add .travis.yml file
 2. Add configuration entry for the Travis VM
 3. Add ReFrame tests

ReFrame with CSCS CI service

The image shows a GitHub Actions workflow for testing ReFrame with the CSCS CI service. The workflow is named 'Test Arbor Demo' and is triggered by a push to the 'main' branch. It includes steps for checking out the code, running shell scripts, and archiving artifacts. The terminal output shows the execution of the ReFrame tests, which passed successfully.

Workflow Details:

- Branch: — 15m 24s No changes
- Commit: — 14 minutes ago GitHub pull request
- Description: [ci] ReFrame tests and integration with CSCS' CI service
- Testing - 15m 18s
- Steps:
 - ✓ > Check out from version control
 - ✓ > Check out from version control
 - ✓ > Shell Script
 - ✓ > arbor-ci.out,reframe.out,reframe.log — Archive the artifacts
 - ✓ > Recursively delete the current directory from the workspace

Terminal Output:

```
Submitted batch job 11594570
Command line: ../reframe/bin/reframe --system=daint:gpu -C ci/rfm-config.py -c ci/arbor_tests.py --
prefix=/scratch/snx3000/jenscscs/arbor-ci-fef64d1-19 --exec-policy=async -r
Reframe version: 2.16-dev1
Launched by user: jenscscs
Launched on host: nid02854
Reframe paths
=====
Check prefix      :
Check search path : 'ci/arbor_tests.py'
Stage dir prefix  : /scratch/snx3000/jenscscs/arbor-ci-fef64d1-19/stage/
Output dir prefix : /scratch/snx3000/jenscscs/arbor-ci-fef64d1-19/output/
Perf. logging prefix : /scratch/snx3000/jenscscs/arbor-ci-fef64d1-19/perflogs
[=====] Running 4 check(s)
[=====] Started on Tue Jan 29 16:11:39 2019

[-----] started processing ArborBaseTest (ArborBaseTest)
[ RUN      ] ArborBaseTest on daint:gpu using PrgEnv-gnu
[-----] finished processing ArborBaseTest (ArborBaseTest)

[-----] started processing ArborMPIITest (ArborMPIITest)
[ RUN      ] ArborMPIITest on daint:gpu using PrgEnv-gnu
[-----] finished processing ArborMPIITest (ArborMPIITest)

[-----] started processing ArborGpuTest (ArborGpuTest)
[ RUN      ] ArborGpuTest on daint:gpu using PrgEnv-gnu
[-----] finished processing ArborGpuTest (ArborGpuTest)


[-----] started processing ArborSIMDTest_haswell (ArborSIMDTest_haswell)
[ RUN      ] ArborSIMDTest_haswell on daint:gpu using PrgEnv-gnu
[-----] finished processing ArborSIMDTest_haswell (ArborSIMDTest_haswell)


[-----] waiting for spawned checks to finish
[ OK      ] ArborGpuTest on daint:gpu using PrgEnv-gnu
[ OK      ] ArborBaseTest on daint:gpu using PrgEnv-gnu
[ OK      ] ArborMPIITest on daint:gpu using PrgEnv-gnu
[ OK      ] ArborSIMDTest_haswell on daint:gpu using PrgEnv-gnu
[-----] all spawned checks have finished

[ PASSED   ] Ran 4 test case(s) from 4 check(s) (0 failure(s))
[=====] Finished on Tue Jan 29 16:22:11 2019


> arbor-ci.out,reframe.out,reframe.log — Archive the artifacts <1s
> Recursively delete the current directory from the workspace 9s
```

ReFrame with Travis



**All checks have passed**
2 successful checks

[Show all checks](#)

**This branch has no conflicts with the base branch**
Merging can be performed automatically.




Merge pull request ▾

You can also [open this in GitHub Desktop](#) or view [command line instructions](#).

Current Branches Build History Pull Requests > Build #30


More options 


oo Pull Request #1 first commit

Commit 5365088 
#1: first commit 
Branch master 









 Victor authored  Victor Holanda Rusu committed

#30 started

 Running for 4 min 7 sec

 Cancel build

Build Jobs

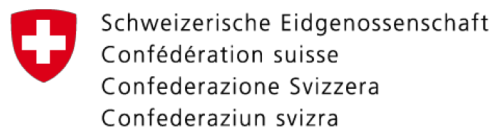
oo # 30.1	 </> Python: 3.6	 MATRIX_EVAL="CC=gcc-4.9 && CXX=g++-4.9"	 4 min 7 sec	
oo # 30.2	 </> Python: 3.6	 MATRIX_EVAL="CC=gcc-5 && CXX=g++-5"	 3 min 58 sec	
oo # 30.3	 </> Python: 3.6	 MATRIX_EVAL="CC=gcc-6 && CXX=g++-6"	 -	
oo # 30.4	 </> Python: 3.6	 MATRIX_EVAL="CC=gcc-7 && CXX=g++-7"	 -	

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
 - Seamless support for containers
 - Benchmarking mode
 - Bug reports, feature requests, help @ <https://github.com/eth-cscs/reframe>

Who is running ReFrame



Ohio Supercomputer Center
An **OH-TECH** Consortium Member



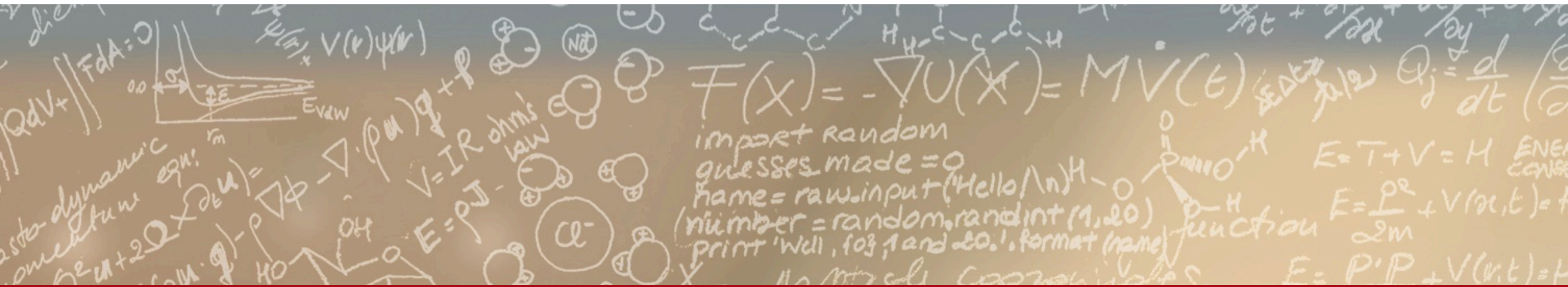
RUTGERS Office of Advanced Research Computing

ASML



Acknowledgements

- Framework contributions
 - Andreas Jocksch
 - Christopher Bignamini
 - Matthias Kraushaar
 - Rafael Sarmiento
 - Samuel Omlin
 - Theofilos Manitaras
 - Vasileios Karakasis
 - Victor Holanda
- Regression tests
 - SCS and OPS team

**CSCS**Centro Svizzero di Calcolo Scientifico
Swiss National Supercomputing Centre**ETH** zürich

Thank you for your attention.

reframe@sympa.cscs.ch<https://eth-cscs.github.io/reframe><https://github.com/eth-cscs/reframe><https://reframe-slack.herokuapp.com>