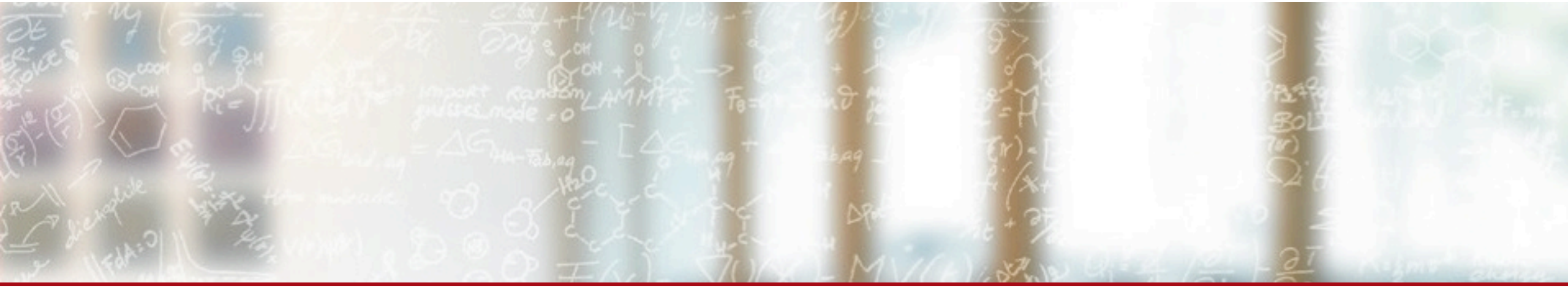
CSCS
Centro Svizzero di Calcolo Scientifico
Swiss National Supercomputing Centre

ETH zürich



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

4th EasyBuild User Meeting

Victor Holanda Rusu and Vasileios Karakasis, CSCS

January 31, 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

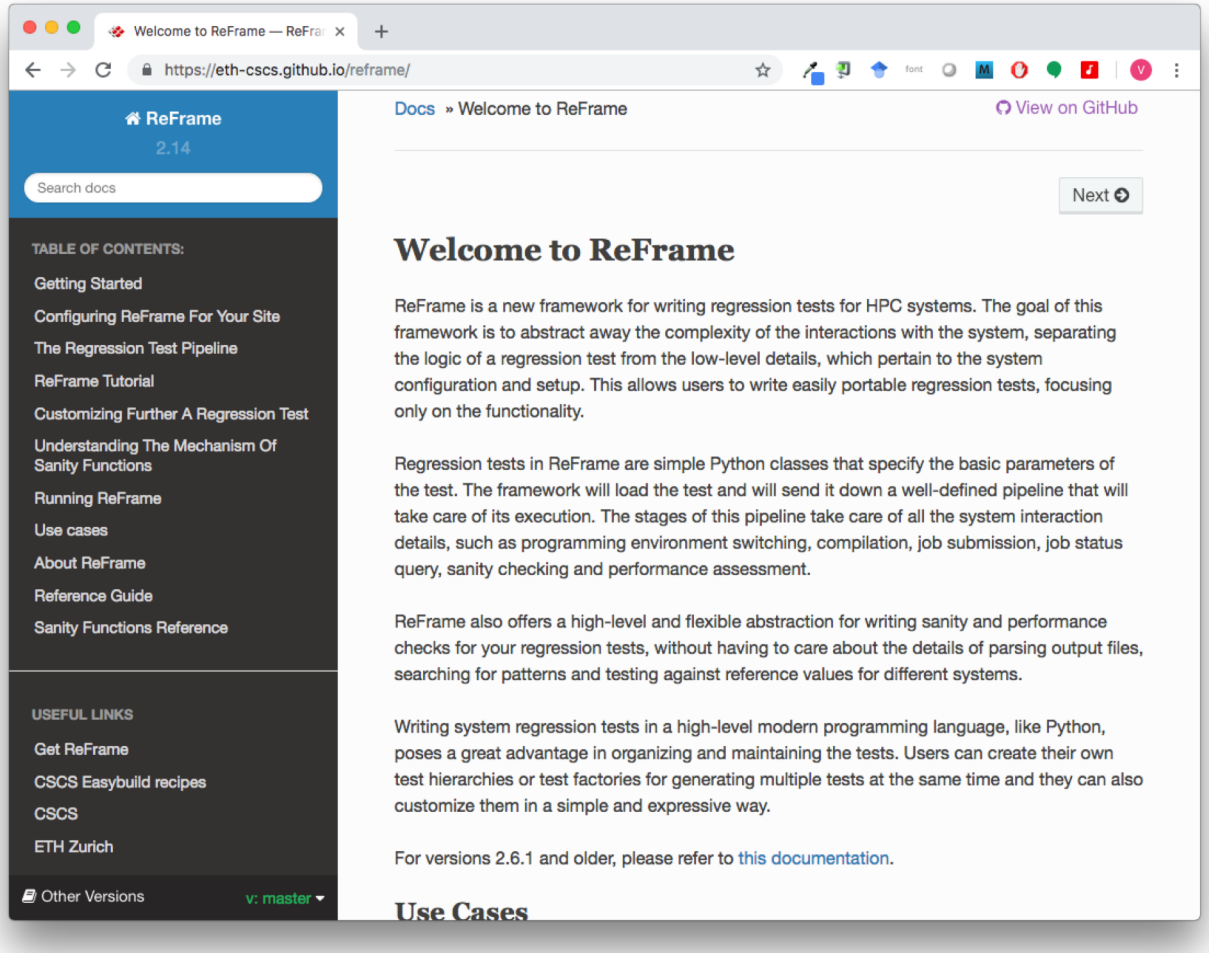
- Simple changes required significant team effort
  - Porting all tests to native SLURM took several weeks

- 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

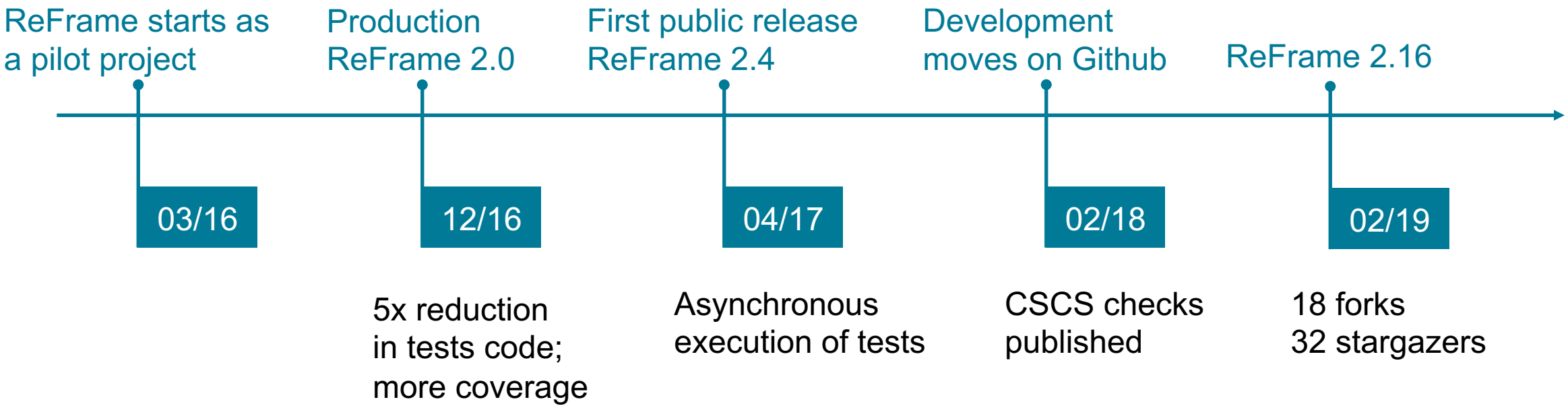
- allows writing **portable HPC** regression tests in Python,
- **abstracts away** the system interaction details,
- lets users focus solely on the **logic** of their test.



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!

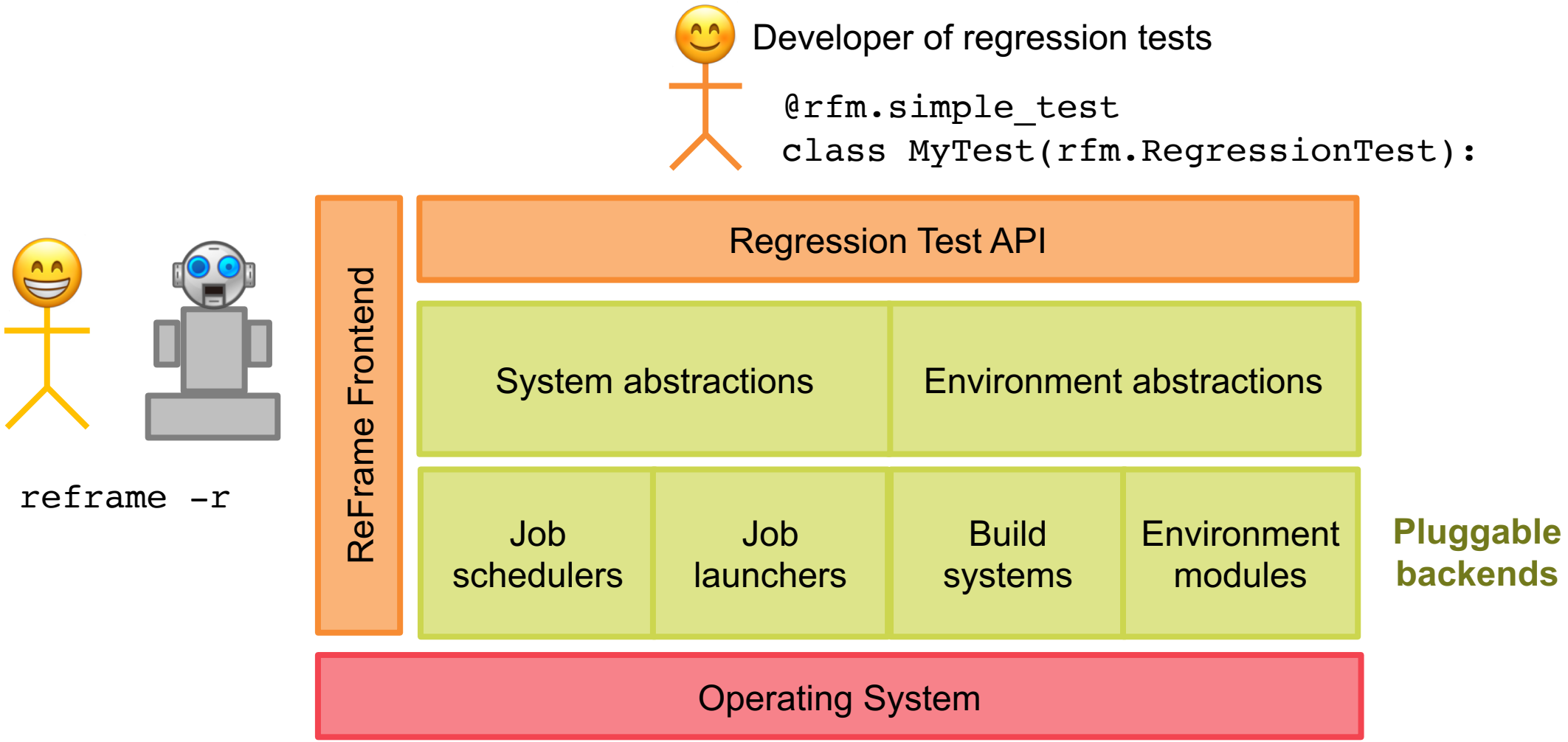
Key Features

- No external Python dependencies
- 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



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]
```

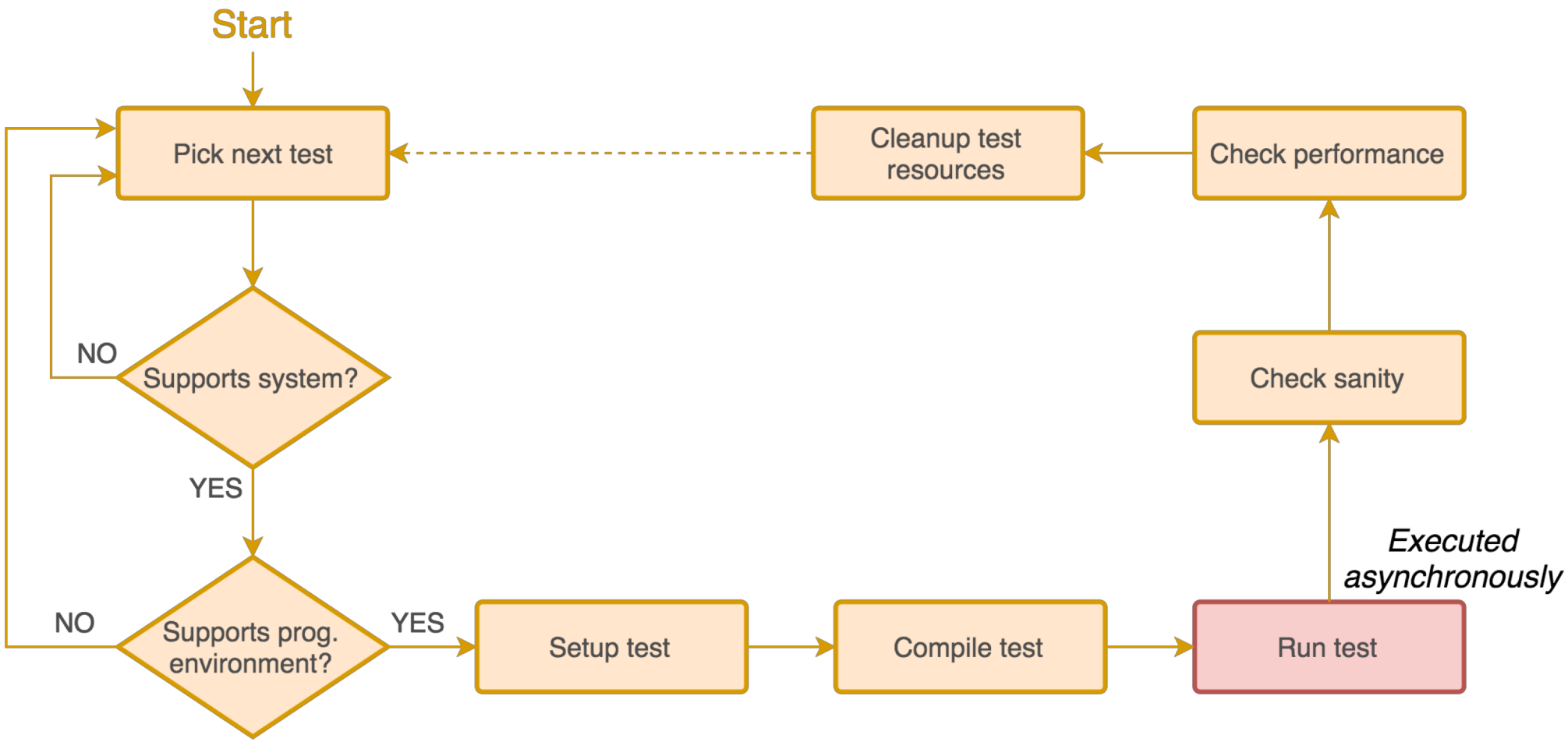
Use parameterized tests to create test factories!

n use inheritance to avoid ning 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)

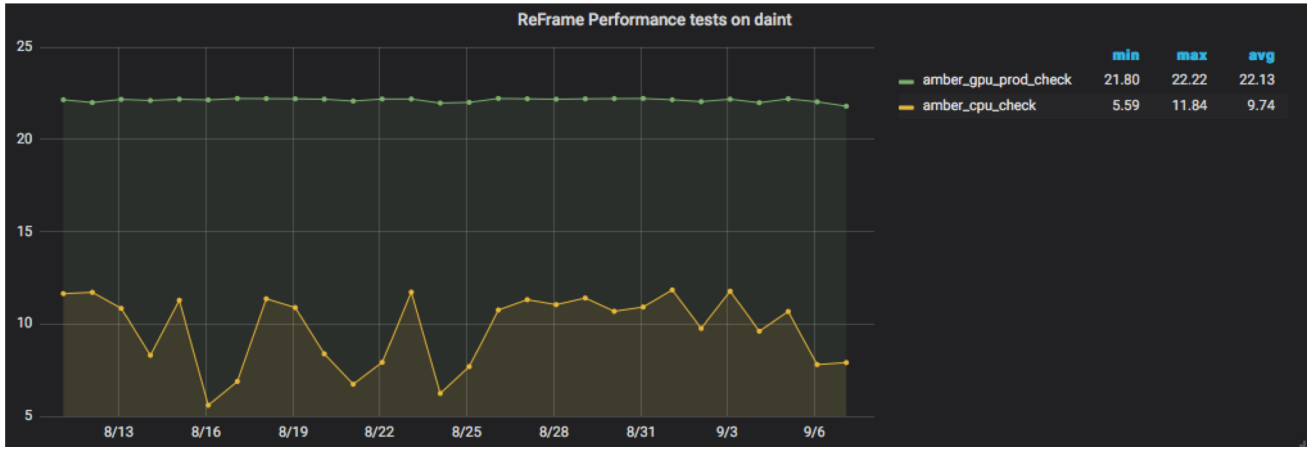
```
[==========] Running 1 check(s)
[==========] Started on Fri Sep  7 16:40:12 2018

[----------] 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-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

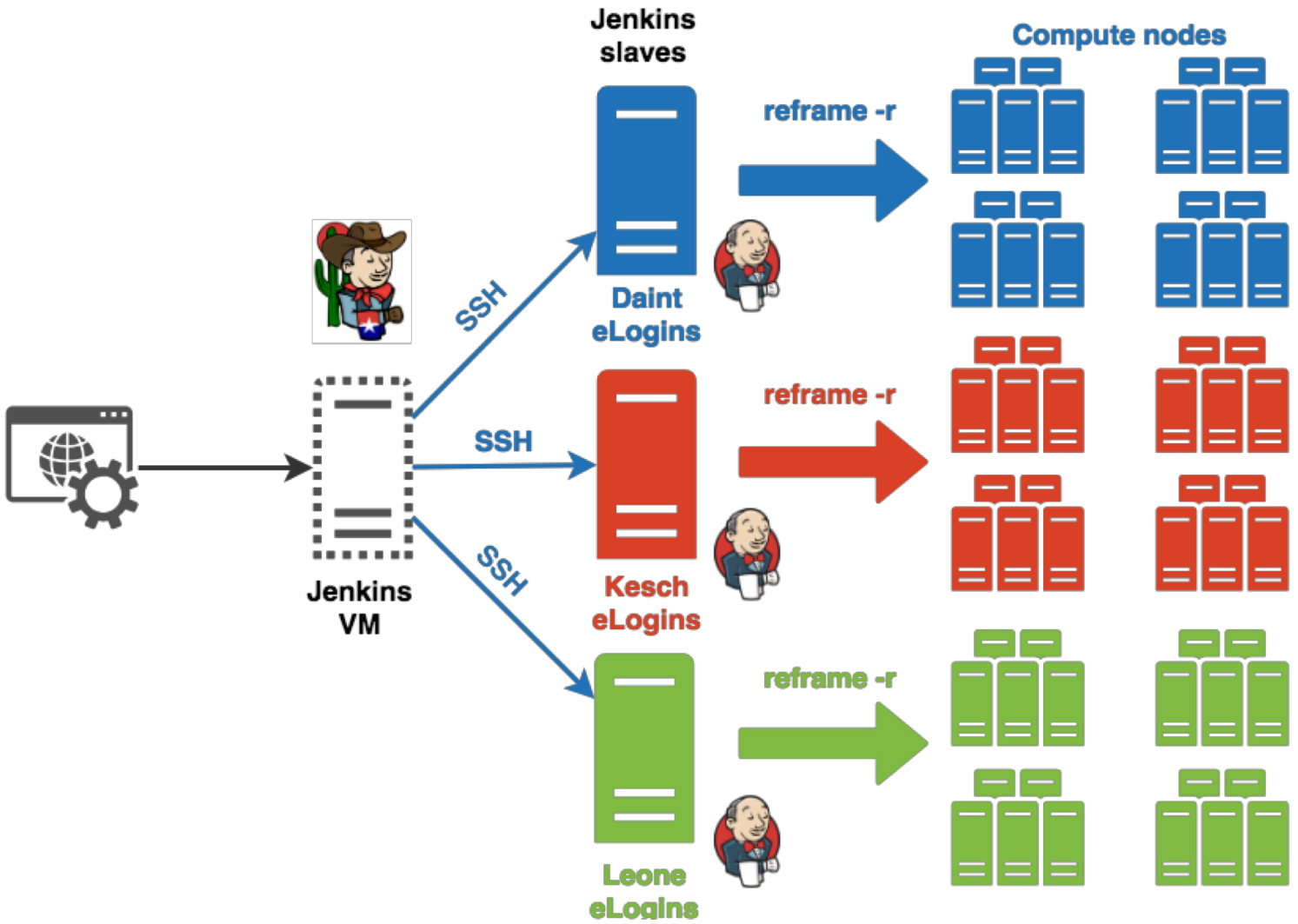
Using ReFrame at CSCS

ReFrame @ CSCS / Tests

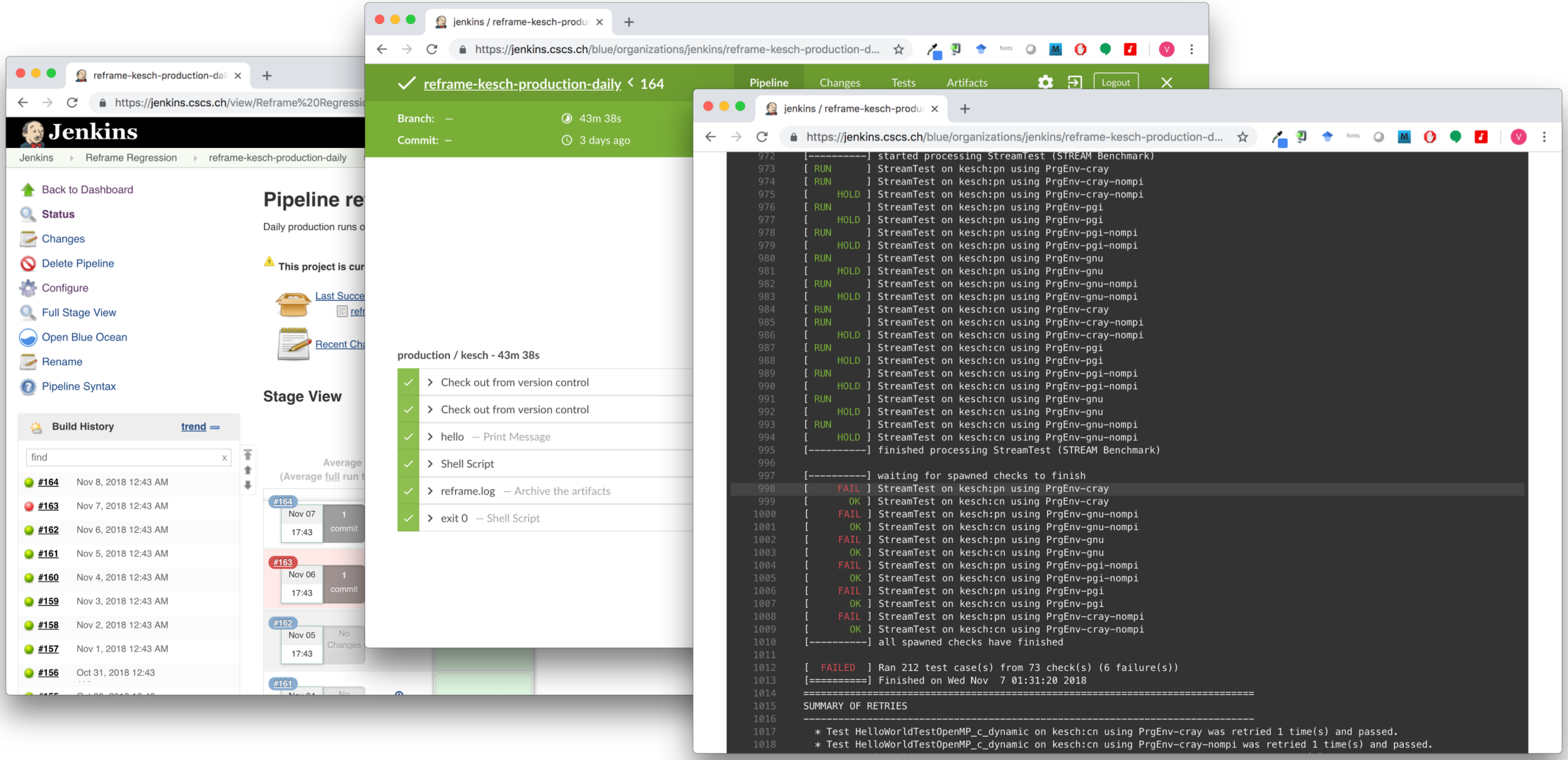
- Used for continuously testing systems in production
  - Piz Daint: 179 tests
  - Piz Kesch: 75 tests
  - Leone: 45 tests
  - **Total: 241 different tests (reused across systems)**

- 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



CSCS
Centro Svizzero di Calcolo Scientifico
Swiss National Supercomputing Centre

ETH zürich

Integrating ReFrame with EasyBuild

ReFrame EasyBuild Integration

- EasyBuild sanity testing is not enough to test full software installations
  - It cannot perform multi-node tests, MPI-based tests, etc
  - Application testing depends on input file and on the system
- Software installation could be connected with sanity and performance testing, in an optative scheme
- One binding mechanism could be the EasyBuild Hooks
  - Advantages
    - Fully programmable
    - No code changes on ReFrame
  - Problems
    - Proper binding requires knowledge of EB internal APIs
    - Tight binding between EB version and ReFrame version
    - Lack of command line arguments to pass to hooks
- Propose to have a separate regression test step in EB pipeline

https://github.com/eth-cscs/production/pull/965/files

```
import copy
import os
import random
import shlex
import subprocess
from vsc.utils import fancylogger
from easybuild.tools.build_log import EasyBuildError, dry_run_msg
from easybuild.tools.modules import get_software_root
from easybuild.tools.config import build_option, log_path

_log = fancylogger.getLogger('hooks', fname=False)
def post_sanitycheck_hook(self, *args, **kwargs):
    def prepend_fake_module_path():
        env = copy.deepcopy(os.environ)
        fake_mod_path = self.make_module_step(fake=True)
        self.modules_tool.prepend_module_path(os.path.join(fake_mod_path, self.mod_subdir), priority=10000)
        return (fake_mod_path, env)

    self.modules_tool.load(['reframe'], allow_reload=False)
    CUDA = get_software_root('CUDA')
    dry_run = build_option('extended_dry_run')
    silent = build_option('silent')
    if not dry_run:
        fake_mod_data = prepend_fake_module_path(purge=True)
    rfm_cmd = "reframe --nocolor -r -o %s -s %s --perflogdir %s --save-log-files " % (output_dir, stage_dir, perflogs_dir)
    if self.name == 'GROMACS':
        rfm_cmd += " -n gromacs_cpu_prod_check -M %s:%s" % (self.name, self.short_mod_name)
    elif self.name == 'Amber' and CUDA:
        rfm_cmd += " -n amber_gpu_prod_check -M %s:%s" % (self.name, self.short_mod_name)
    else:
        _log.info("No dedicated ReFrame test found. Skipping ReFrame run...")
        if dry_run:
            dry_run_msg("No dedicated ReFrame test found. Skipping ReFrame run...\n", silent=silent)
        return
    if not dry_run:
        run_reframe(rfm_cmd, dir=rfm_run_dir, shell=False)
        self.clean_up_fake_module(fake_mod_data)
    else:
        dry_run_msg("ReFrame command: %s" % rfm_cmd, silent=silent)
```

ich

ReFrame

```
eb --hooks=refr
== temporary lo
1qJS_q/easybuil
== processing E
/users/hvictor/
18.08-cuda-9.1.
== building and
== fetching fil
== creating bui
== unpacking...
== patching...
== preparing...
== configuring.
== building...
== testing...
== installing..
== taking care
== postprocessi
== sanity check
== Running post
== cleaning up.
== creating mod
== permissions.
== packaging...
== COMPLETED: I
== Results of t
/apps/daint/UES
18.08-cuda-9.1/
== Build succee
== Temporary lo
have been remov
== Temporary di
```

```
== 2019-01-25 13:21:10,334 reframe.py:33 INFO ReFrame execution was successful: Command line:
/apps/common/UES/jenkins/SLES12/easybuild/softwa\
re/reframe/2.16-dev0/bin/reframe --nocolor -r -o
/apps/daint/UES/6.0.UP07/sandboxes/hvictor/easybuild/software/Amber/18-9-4-CrayGNU-18.08-cuda\
-9.1/easybuild/reframe_output -s /scratch/snx3000/hvictor/easybuild/tmp-6463570/reframe_stage --perflogdir
/apps/daint/UES/6.0.UP07/sandboxes/\
hvictor/easybuild/software/Amber/18-9-4-CrayGNU-18.08-cuda-9.1/easybuild/reframe_perflogs --save-log-files -n
amber_gpu_prod_check -M Amber:Am\
ber/18-9-4-CrayGNU-18.08-cuda-9.1
Reframe version: 2.16-dev0
Launched by user: hvictor
Launched on host: daint101
Reframe paths
=============
    Check prefix      : /apps/common/UES/jenkins/SLES12/easybuild/software/reframe/2.16-dev0
(R) Check search path : 'checks/'
    Stage dir prefix     : /scratch/snx3000/hvictor/easybuild/tmp-6463570/reframe_stage/
    Output dir prefix    : /apps/daint/UES/6.0.UP07/sandboxes/hvictor/easybuild/software/Amber/18-9-4-CrayGNU-
18.08-cuda-9.1/easybuild/reframe\
_output/
    Perf. logging prefix : /apps/daint/UES/6.0.UP07/sandboxes/hvictor/easybuild/software/Amber/18-9-4-CrayGNU-
18.08-cuda-9.1/easybuild/reframe\
_perflogs
[==========] Running 1 check(s)
[==========] Started on Fri Jan 25 13:07:03 2019


[----------] started processing amber_gpu_prod_check (Amber parallel GPU production check)
[ RUN      ] amber_gpu_prod_check on daint:gpu using PrgEnv-gnu
[       OK ] amber_gpu_prod_check on daint:gpu using PrgEnv-gnu
[----------] finished processing amber_gpu_prod_check (Amber parallel GPU production check)


[  PASSED  ] Ran 1 test case(s) from 1 check(s) (0 failure(s))
[==========] Finished on Fri Jan 25 13:21:10 2019
```

ReFrame EasyBuild Integration

```
eb --regression-framework=reframe --regression-arguments="--system generic" Amber-18-9-4-CrayGNU-18.08-cuda-9.1.eb -f
== temporary log file in case of crash /run/user/23962/easybuild/tmp/eb-1qJS_q/easybuild-fjlrgD.log
== processing EasyBuild easyconfig /users/hvictor/EASYBUILD/production/easybuild/easyconfigs/a/Amber/Amber-18-9-4-CrayGNU-18.08-cuda-9.1.eb
== building and installing Amber/18-9-4-CrayGNU-18.08-cuda-9.1...
== fetching files...
== creating build dir, resetting environment...
== unpacking...
== patching...
== preparing...
== configuring...
== building...
== testing...
== installing...
== taking care of extensions...
== postprocessing...
== sanity checking...
== Running regression tests...
== cleaning up...
== creating module...
== permissions...
== packaging...
== COMPLETED: Installation ended successfully
== Results of the build can be found in the log file(s) /apps/daint/UES/6.0.UP07/sandboxes/hvictor/easybuild/software/Amber/18-9-4-CrayGNU-18.08-cuda-9.1/easybuild/easybuild-Amber-18-9-4-20190125.132121.log
== Build succeeded for 1 out of 1
== Temporary log file(s) /run/user/23962/easybuild/tmp/eb-1qJS_q/easybuild-fjlrgD.log* have been removed.
== Temporary directory /run/user/23962/easybuild/tmp/eb-1qJS_q has been removed.
```

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

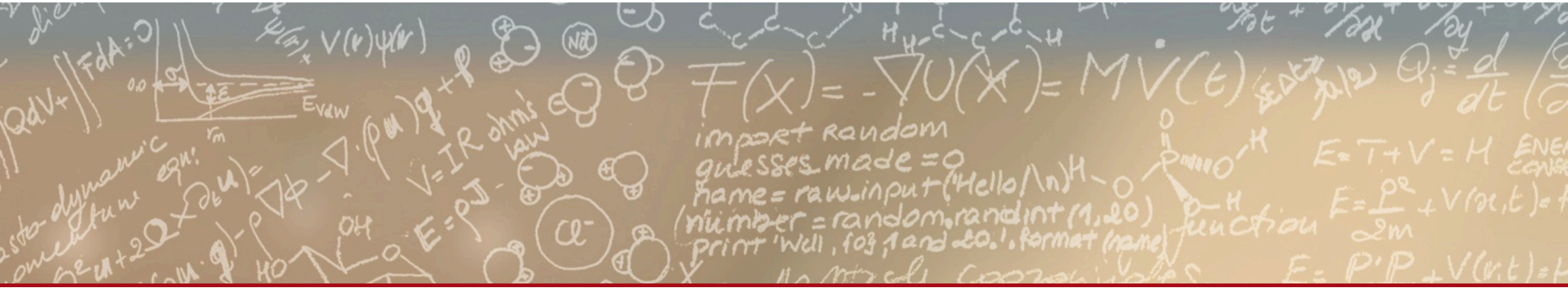
Conclusions and Future Directions

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
  - Container support
  - Benchmarking mode

- Bug reports, feature requests, help @ https://github.com/eth-cscs/reframe



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