

ReFrame: A Regression Testing Framework Enabling Continuous Integration of Large HPC Systems

HPC Advisory Council 2018

Victor Holanda, Vasileios Karakasis, CSCS
Apr. 11, 2018



CSCS

Centro Svizzero di Calcolo Scientifico
Swiss National Supercomputing Centre

ETHzürich

ReFrame in a nutshell

Regression Testing of HPC Systems

Why is it so important?

- Ensures quality of service
- Reduces downtime
- Early detection of problems

Regression Testing of HPC Systems

But it's a painful story

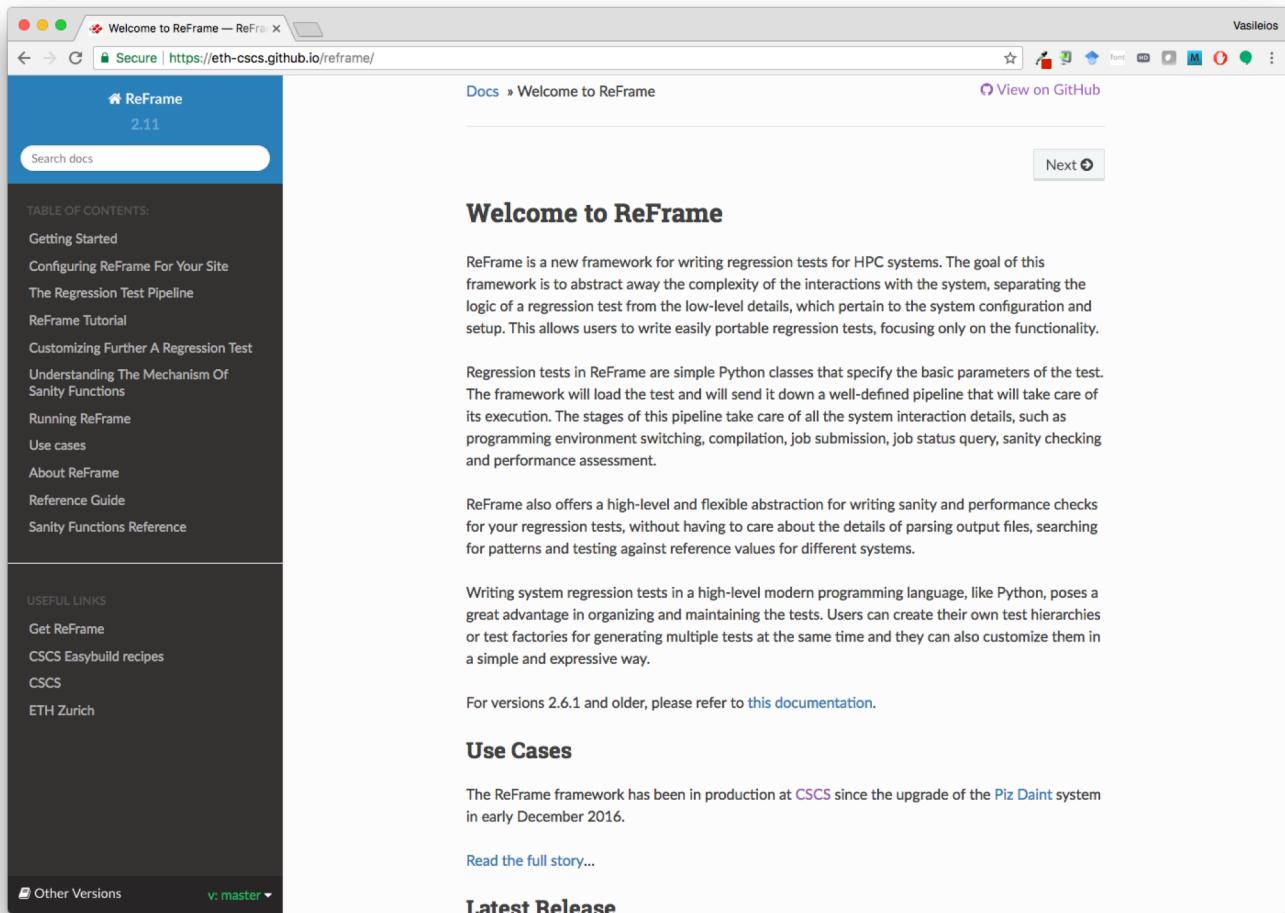
- In-house custom solutions per center
- Non portable monolithic regression tests
 - Tightly coupled to the system configuration and programming environments
- Large maintenance overhead
 - Replicated code of the system interaction details
 - Test's logic is lost in unrelated lower level details

Reluctance to implement new regression tests!

What Is ReFrame?

A new regression 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.



The screenshot shows a web browser displaying the official ReFrame documentation at <https://eth-cscs.github.io/reframe/>. The page title is "Welcome to ReFrame". On the left, there is a sidebar with a dark background containing a navigation menu and useful links. The main content area on the right contains sections such as "Welcome to ReFrame", "ReFrame is a new framework for writing regression tests for HPC systems.", "Regression tests in ReFrame are simple Python classes that specify the basic parameters of the test.", "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.", "For versions 2.6.1 and older, please refer to [this documentation](#).", "Use Cases", "The ReFrame framework has been in production at CSCS since the upgrade of the Piz Daint system in early December 2016.", "Read the full story...", and "Latest Release". The top right corner of the slide shows the name "Vasileios".

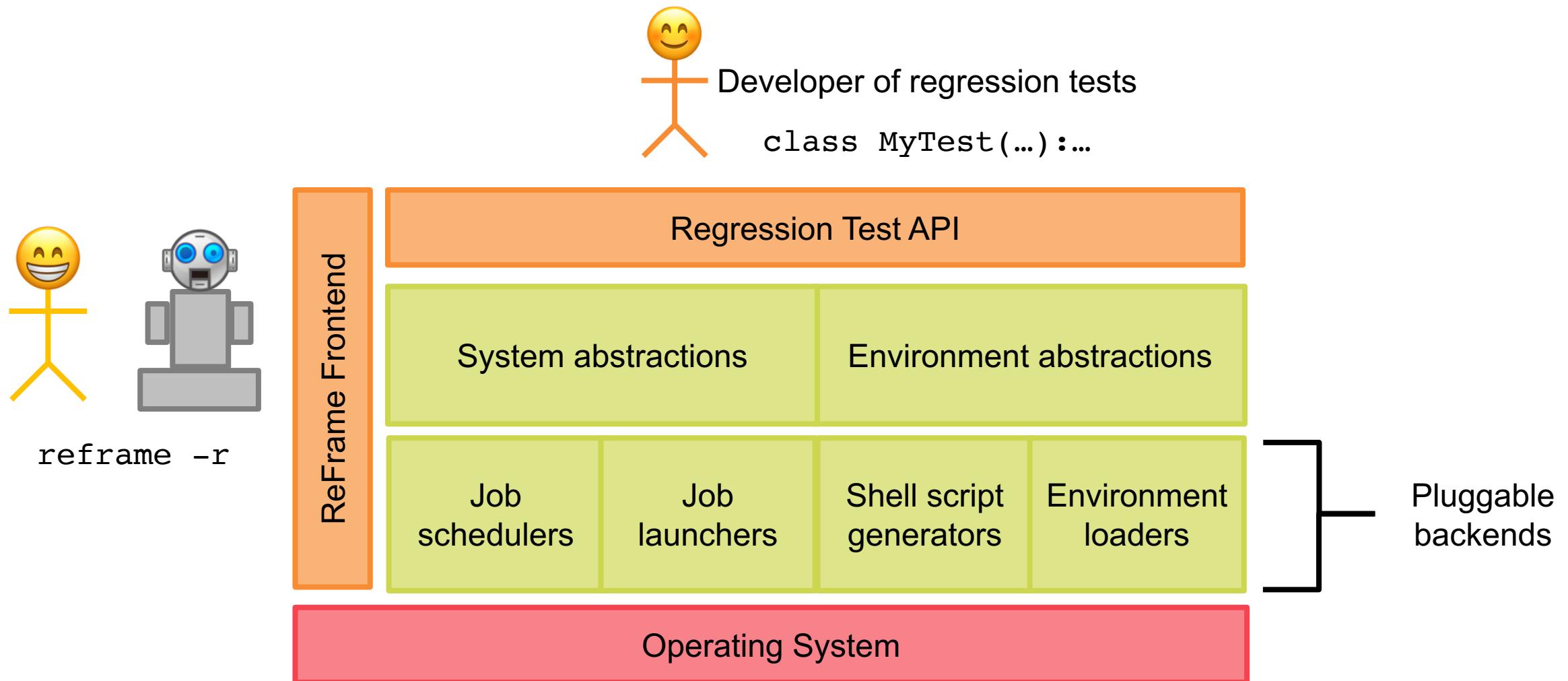
<https://github.com/eth-cscs/reframe>

Design Goals

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

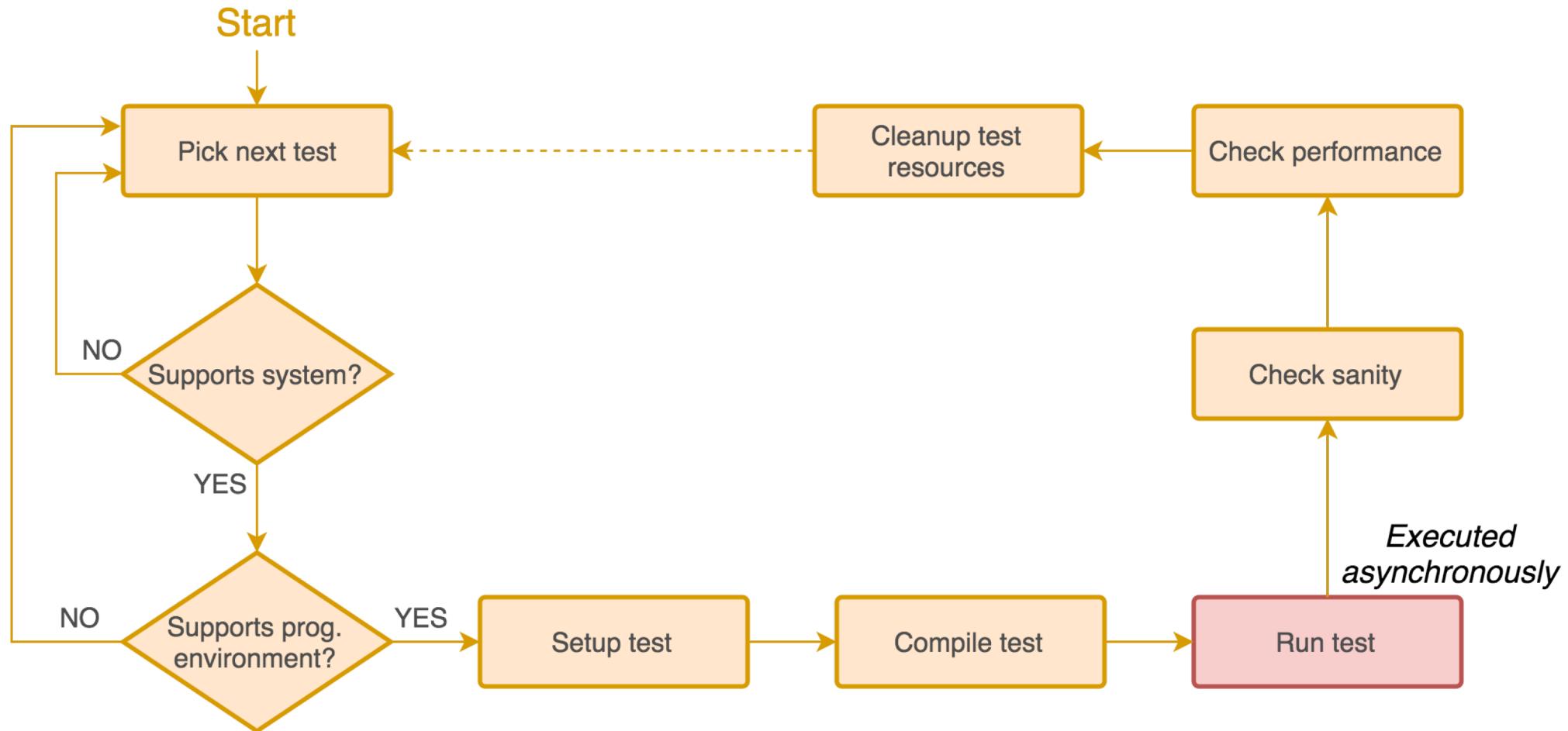
Write once, test everywhere!

ReFrame's architecture



The Regression Test Pipeline

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



Some Features

- Support for Slurm (with and without srun) and simple batch scripts
- Support for different modules systems (Tmod, Lmod, no modules.)
- Seamless support of multiple prog. environments and HPC systems
- Flexible organization of the regression tests
- Progress and result reports
- Asynchronous execution of regression tests
- Complete documentation (tutorials, reference guide)
- And many more (<https://github.com/eth-cscs/reframe>)

Writing a regression test in ReFrame

A regression test writer should not care about...

- How access to system partitions is gained and if there are any.
- How (programming) environments are switched.
- How its environment is 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

Regression tests are Python classes

```
class CudaPerfTest(RegressionTest):
    def __init__(self, **kwargs):
        super().__init__('cuda_matrix_mult_test',
                         os.path.dirname(__file__), **kwargs)
        self.descr = 'Matrix-vector multiplication (CUDA performance test)'
        self.valid_systems = ['daint:gpu']
        self.valid_prog_environ = ['PrgEnv-gnu', 'PrgEnv-cray', 'PrgEnv-pgi']
        self.sourcepath = 'example_matrix_vector_multiplication_cuda.cu'
        self.executable_opts = ['4096', '1000']
        self.modules = ['cudatoolkit']
        self.num_gpus_per_node = 1
        self.sanity_patterns = sn.assert_found(r'The L2 norm of the resulting vector is: 4.096000E+03', self.stdout)
        self.perf_patterns = {
            'perf': sn.extractsingle(r'Performance:\s+(?P<Gflops>\S+)\s+Gflop/s', self.stdout, 'Gflops', float)
        }
        self.reference = {
            'daint:gpu': {'perf': (50.0, -0.1, 0.1)}
        }
```

List of environments to test

Automatic compiler detection

List of supported systems

Extract performance numbers from the output

Performance references per system

Running ReFrame

- Run tests sequentially:
 - `./bin/reframe -c /path/to/checks -r`
- Run tests asynchronously:
 - `./bin/reframe -c /path/to/checks --exec-policy=async -r`
- Test selection (by name, tag, prog. environment)
- Failure reports
- Configurable logging
- Performance logging → allows keeping historical data

Running ReFrame (sample output)

```
[=====] Running 1 check(s)
[=====] Started on Thu Mar 22 17:35:21 2018

[-----] started processing example7_check (CUDA matrixmul test)
[ RUN    ] example7_check on daint:gpu using PrgEnv-cray
[   OK   ] example7_check on daint:gpu using PrgEnv-cray
[ RUN    ] example7_check on daint:gpu using PrgEnv-gnu
[   OK   ] example7_check on daint:gpu using PrgEnv-gnu
[ RUN    ] example7_check on daint:gpu using PrgEnv-pgi
[   OK   ] example7_check on daint:gpu using PrgEnv-pgi
[-----] finished processing example7_check (CUDA matrixmul test)

[ PASSED ] Ran 3 test case(s) from 1 check(s) (0 failure(s))
[=====] Finished on Thu Mar 22 17:35:44 2018
```

Running ReFrame (sample failure)

```
[=====] Running 1 check(s)
[=====] Started on Thu Mar 22 17:56:19 2018

[-----] started processing example7_check (CUDA matrixmul test)
[ RUN   ] example7_check on daint:gpu using PrgEnv-gnu
[ FAIL  ] example7_check on daint:gpu using PrgEnv-gnu
[-----] finished processing example7_check (CUDA matrixmul test)

[ FAILED ] Ran 1 test case(s) from 1 check(s) (1 failure(s))
[=====] Finished on Thu Mar 22 17:56:27 2018
=====
SUMMARY OF FAILURES
-----
FAILURE INFO for example7_check
* System partition: daint:gpu
* Environment: PrgEnv-gnu
* Stage directory: /path/to/stage/gpu/example7_check/PrgEnv-gnu
* Job type: batch job (id=693731)
* Maintainers: []
* Failing phase: performance
* Reason: sanity error: 49.244815 is beyond reference value 70.0 (l=63.0, u=77.0)
```



CSCS

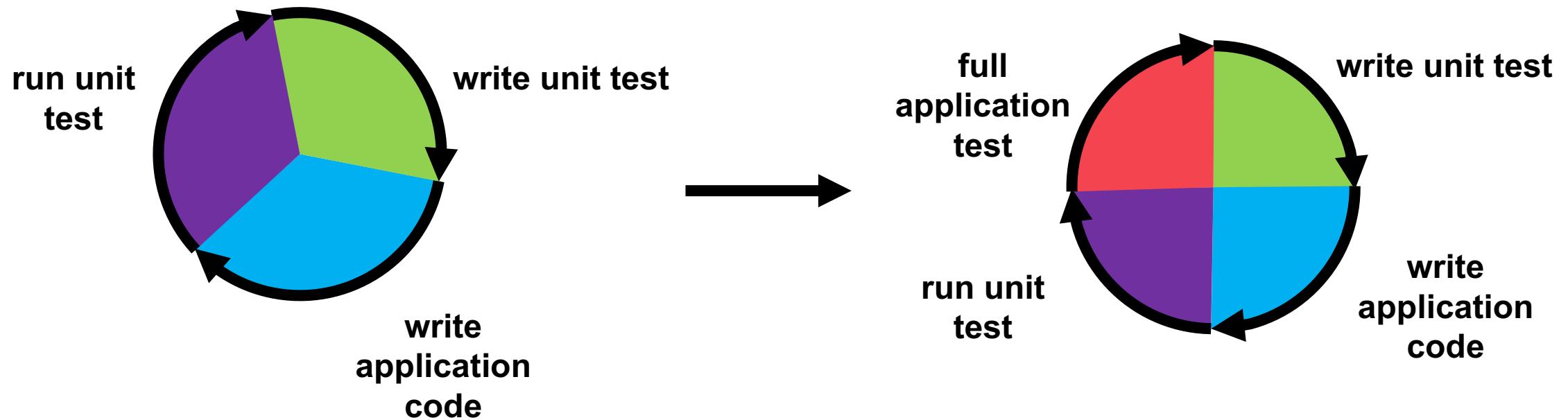
Centro Svizzero di Calcolo Scientifico
Swiss National Supercomputing Centre

ETHzürich

ReFrame inside a CI infrastructure

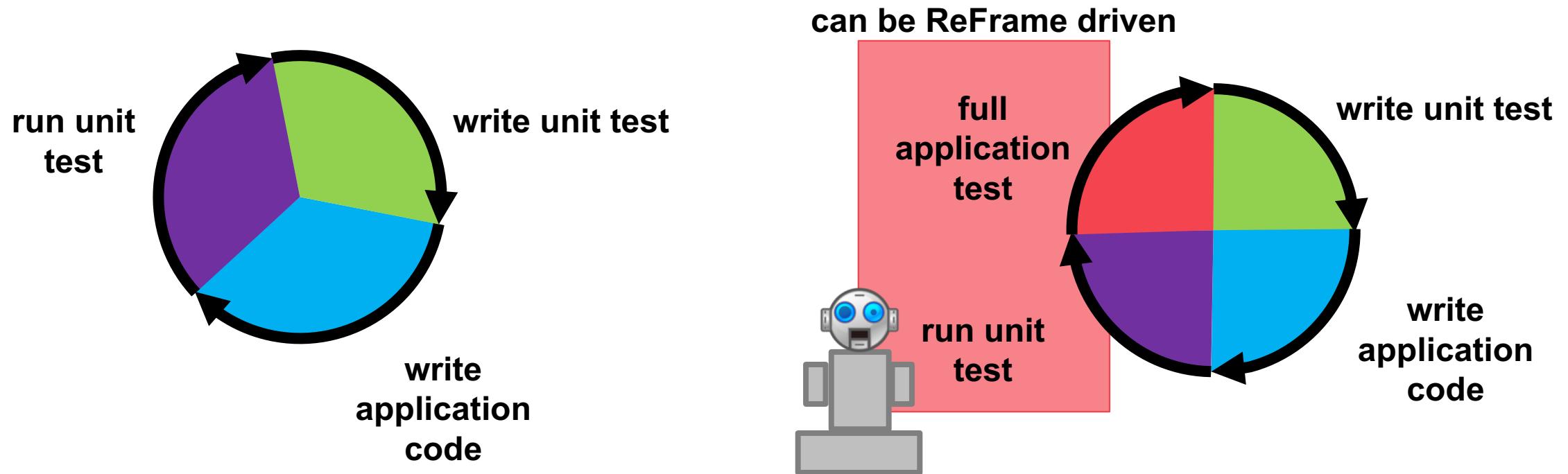
Running ReFrame as a CI tool for HPC applications

- Improve the development cycle of HPC applications
- Develop anywhere and test anywhere



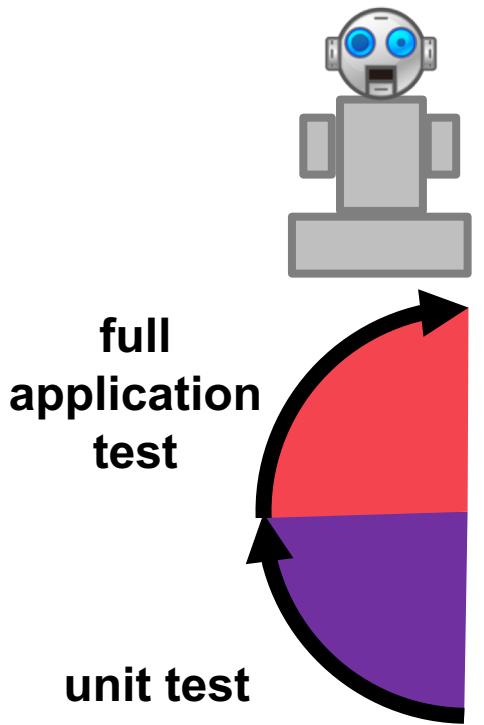
Running ReFrame as a CI tool for HPC applications

- Improve the development cycle of HPC applications
- Develop anywhere and test anywhere



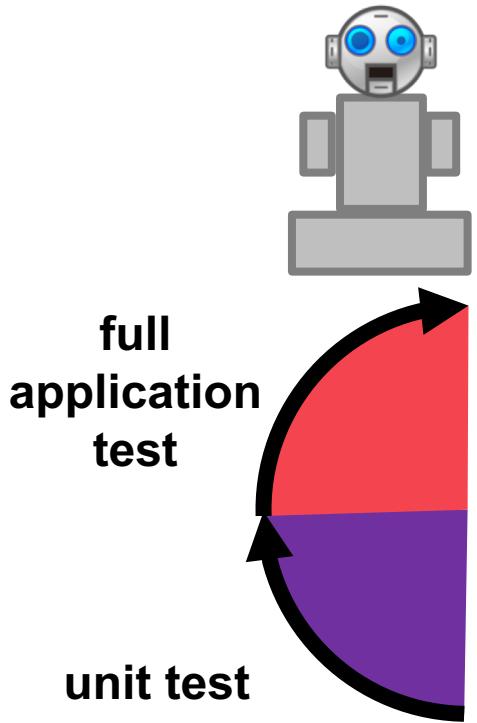
CI tool for HPC applications

- Login into different systems
- Loop over the proper programming environments
- Compile the code
- Create job scripts (if system has a queue system)
- Run unit tests
- Run different input files
- Collect sanity data
- Collect performance data
- Keep track if the code is still performing



CI tool for HPC applications

- Login into different systems
- Loop over the proper programming environments
- Compile the code
- Create job scripts (if system has a queue system)
- Run unit tests
- Run different input files
- Collect sanity data
- Collect performance data
- Keep track if the code is still performing



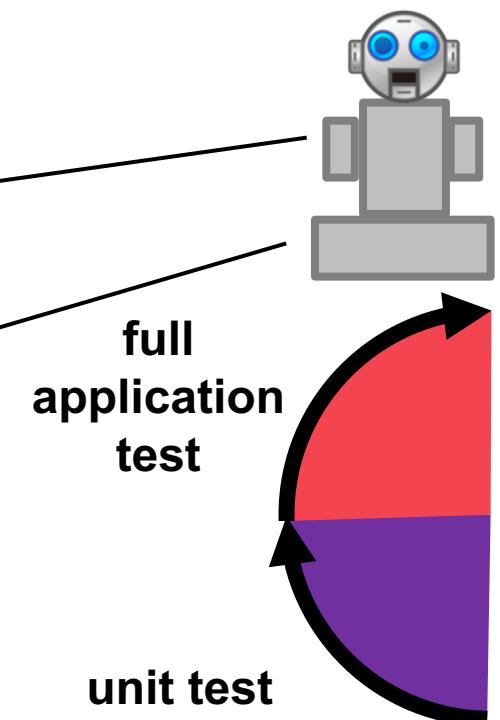
Support to send data
to elastic search
databases!

CI tool for HPC applications

- ~~Login into different systems~~
- Loop over the proper programming environments
- Compile the code
- Create job scripts (if system has a queue system)
- Run unit tests
- Run different input files
- Collect sanity data
- Collect performance data
- Keep track if the code is still performing

ReFrame

CI infrastructure



Support to send data
to elastic search
databases!

Running ReFrame as a CI tool for HPC applications

1. Add new **system** to ReFrame configuration inside your project.
2. Store your **ReFrame tests** in your project.
3. Run your tests on target system using ReFrame.

Use the same tests to run on Piz Daint, your laptop or a Travis VM!

Demo Time

1. Running ReFrame
2. Integration with TRAVIS (<https://github.com/victorusu/promd/pull/1>)

Travis – PROMD demo

The screenshot shows a GitHub pull request page for a repository named 'victorusu / promd'. The pull request is titled 'ReFrame demo #1' and is currently open, showing 1 commit from 'promd_test' into the 'master' branch. The commit message is 'victorusu wants to merge 1 commit into master from promd_test'. The commit details show 'Conversation 0', 'Commits 1', and 'Files changed 4,090'. The commit itself is by 'victorusu' and includes the following actions:

- victorusu commented 18 days ago: Start of the project
- victorusu added the **enhancement** label 18 days ago
- victorusu self-assigned this 18 days ago
- set reframe version to 2.11 (commit hash: c25a07b)
- victorusu changed the title from first commit to ReFrame demo 6 minutes ago

On the right side of the page, there are several sections: 'Reviewers' (No reviews), 'Assignees' (victorusu), 'Labels' (enhancement), 'Projects' (None yet), 'Milestone' (No milestone), and 'Notifications' (You're receiving notifications because you were assigned). A 'Merge pull request' button is at the bottom left, and a note says 'You can also open this in GitHub Desktop or view command line instructions.'

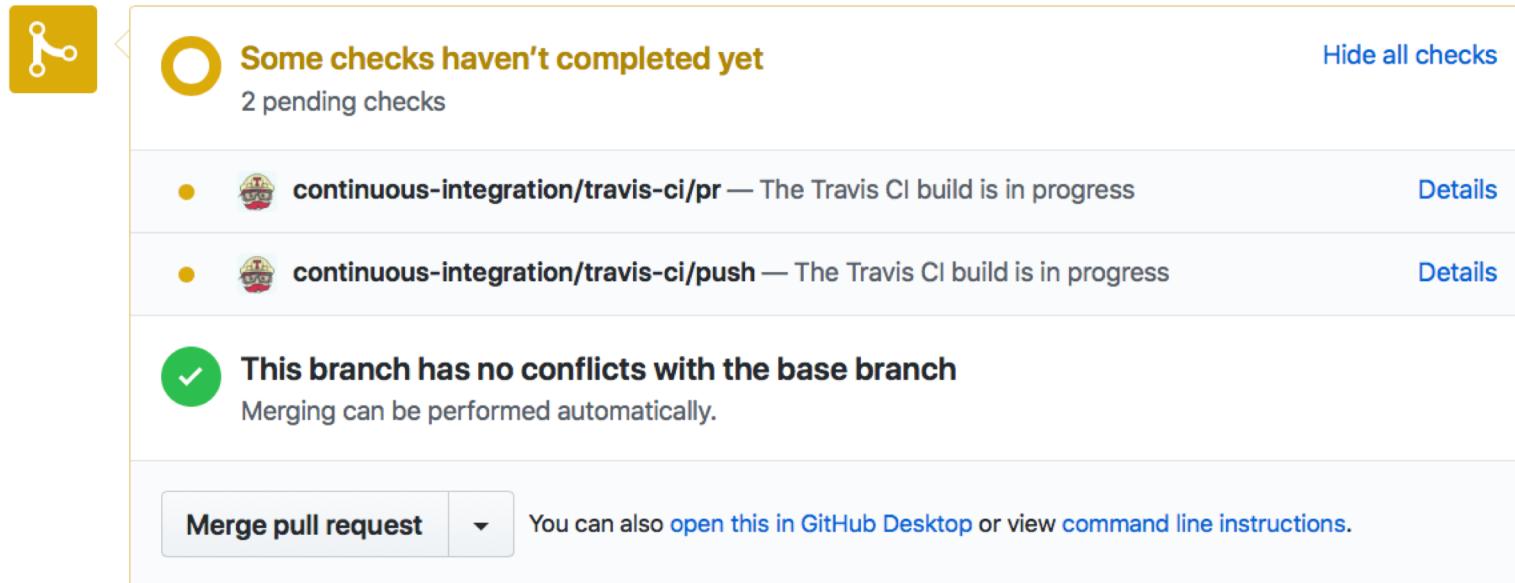
Travis – PROMD demo

217  promd.py

View   

```
...     ... @@ -0,0 +1,217 @@
1+import itertools
2+import os
3+
4+import reframe.utility.sanity as sn
5+from reframe.core.pipeline import RegressionTest
6+
7+
8+class PROMDCheck(RegressionTest):
9+    def __init__(self, num_cores, variant, input, **kwargs):
10+        super().__init__("promd_%s_%s_%s_check" % (input, num_cores, variant),
11+                         os.path.dirname(__file__), **kwargs)
12+
13+        output_file = '%s.omd' % input
14+        self.compiler_option = "omp" if variant == 'omp' else "promd"
15+        self.num_cores_avail = 4
16+
17+        #self.sourcepath = 'promd'
18+        self.sourcesdir = 'promd'
19+        self.keep_files = [output_file]
20+
21+        self.valid_systems = ['daint:gpu', 'generic']
22+        self.descr = 'PROMD %s %s check with %s cpus ' % (input, variant, num_cores)
23+
```

Travis – PROMD demo

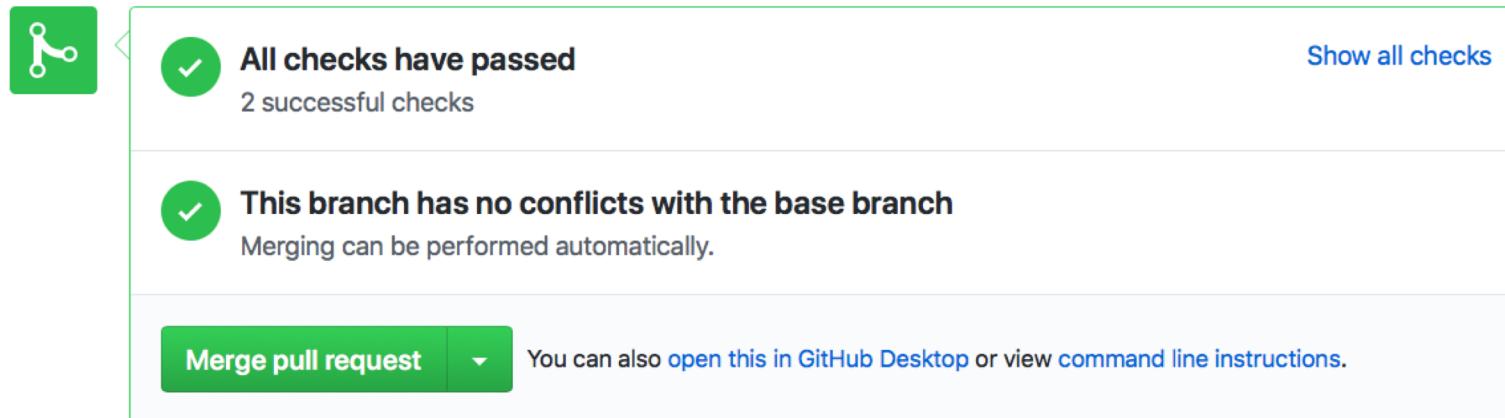


A screenshot of a GitHub pull request interface. At the top left is a yellow icon with a gear and a checkmark. To its right, a large orange circle contains a white gear icon, with the text "Some checks haven't completed yet" next to it. Below this, "2 pending checks" is written. To the right of the pending checks, there is a blue link "Hide all checks". The main body of the interface lists two pending checks, each with a small orange dot icon and a Travis CI logo. The first check is for "continuous-integration/travis-ci/pr" and the second for "continuous-integration/travis-ci/push", both stating "The Travis CI build is in progress". To the right of each check is a blue "Details" link. Below these sections, a green circle with a white checkmark contains the text "This branch has no conflicts with the base branch". Underneath this, a smaller text says "Merging can be performed automatically.". At the bottom left is a button labeled "Merge pull request" with a dropdown arrow. Next to it is a message: "You can also [open this in GitHub Desktop](#) or view command line instructions.".

Travis – PROMD demo

```
626 $ sudo pip install --upgrade pip
636 $ python3 ./reframe/reframe.py -c promd.py --system generic -r
637 Command line: ./reframe/reframe.py -c promd.py --system generic -r
638 Reframe version: 2.11
639 Launched by user: travis
640 Launched on host: travis-job-52ce71d7-d471-4cbd-8d28-67d708f2212a
641 Reframe paths
642 =====
643     Check prefix      :
644     Check search path : 'promd.py'
645     Stage dir prefix  : /home/travis/build/victorusu/promd/stage/
646     Output dir prefix : /home/travis/build/victorusu/promd/output/
647     Logging dir       : /home/travis/build/victorusu/promd/logs
648 [=====] Running 9 check(s)
649 [=====] Started on Mon Apr  9 14:03:49 2018
650
651 [-----] started processing promd_methane_1_serial_check (PROMD methane serial check with 1 cpus )
652 [ RUN    ] promd_methane_1_serial_check on generic:login using builtin-gcc
653 [      OK ] promd_methane_1_serial_check on generic:login using builtin-gcc
654 [-----] finished processing promd_methane_1_serial_check (PROMD methane serial check with 1 cpus )
655
656 [-----] started processing promd_ethane_1_serial_check (PROMD ethane serial check with 1 cpus )
657 [ RUN    ] promd_ethane_1_serial_check on generic:login using builtin-gcc
```

Travis – PROMD demo





CSCS

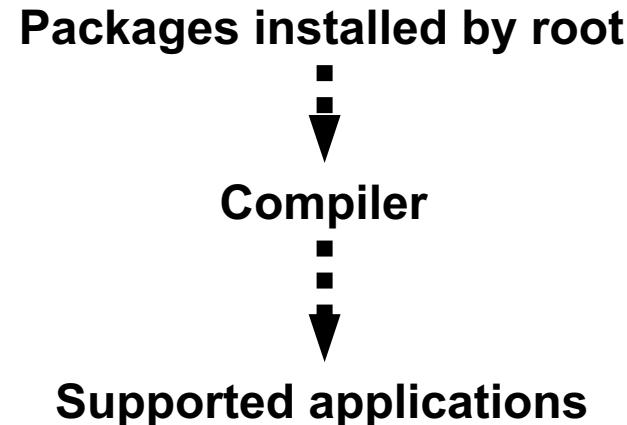
Centro Svizzero di Calcolo Scientifico
Swiss National Supercomputing Centre

ETHzürich

CSCS Use Case

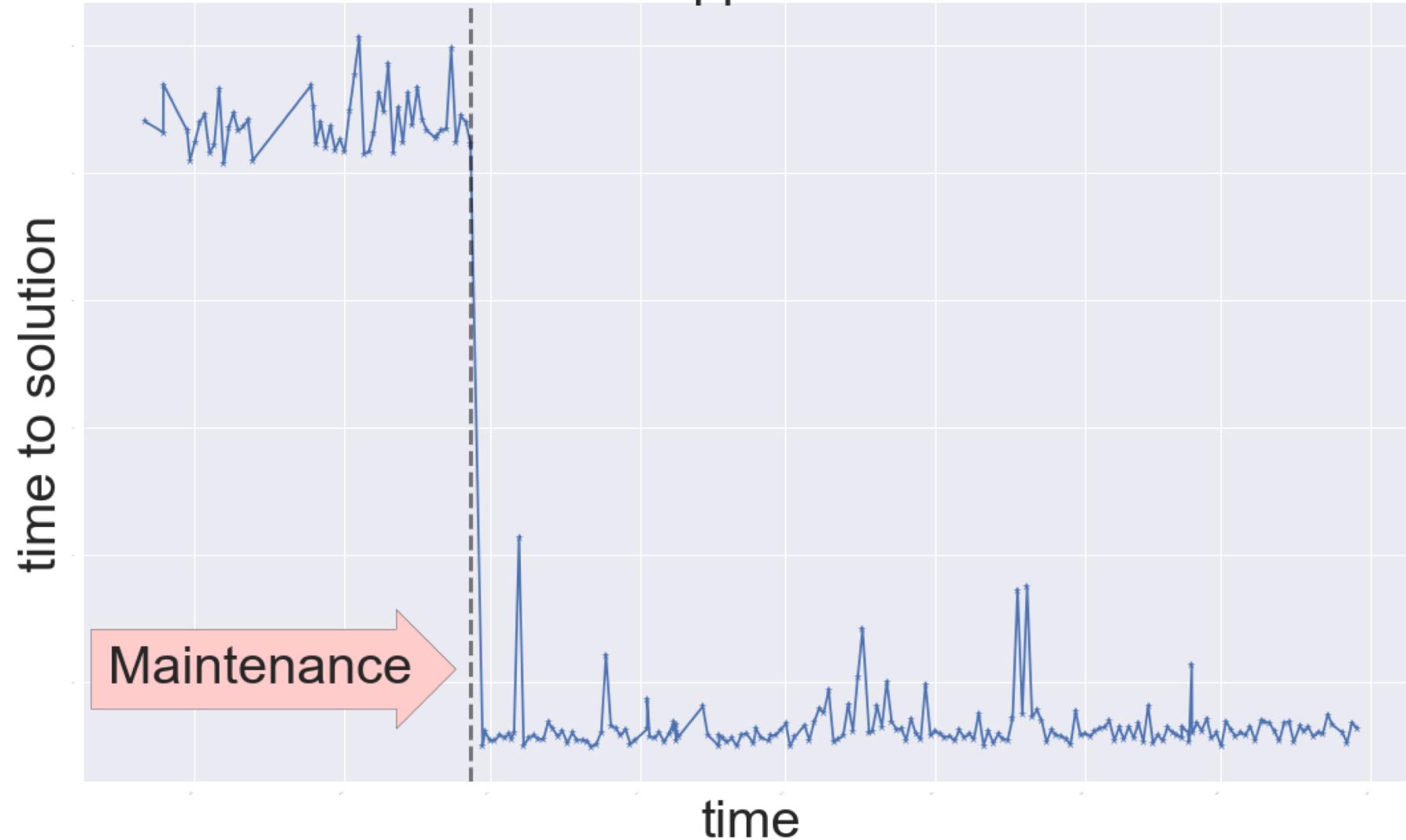
The CSCS Use Case

- ReFrame is used to test all major systems in production
 - The same tests are used for all systems with slight adaptations.
- Wide variety of performance and sanity tests implemented
 - Applications
 - Libraries
 - Programming environment tests
 - I/O benchmarks
 - Performance tools and debuggers
 - Job scheduler tests
- Two execution modes
 - **Production:** A wide aspect of the sanity and performance tests running daily
 - **Maintenance:** Key functionality and performance tests run during maintenances



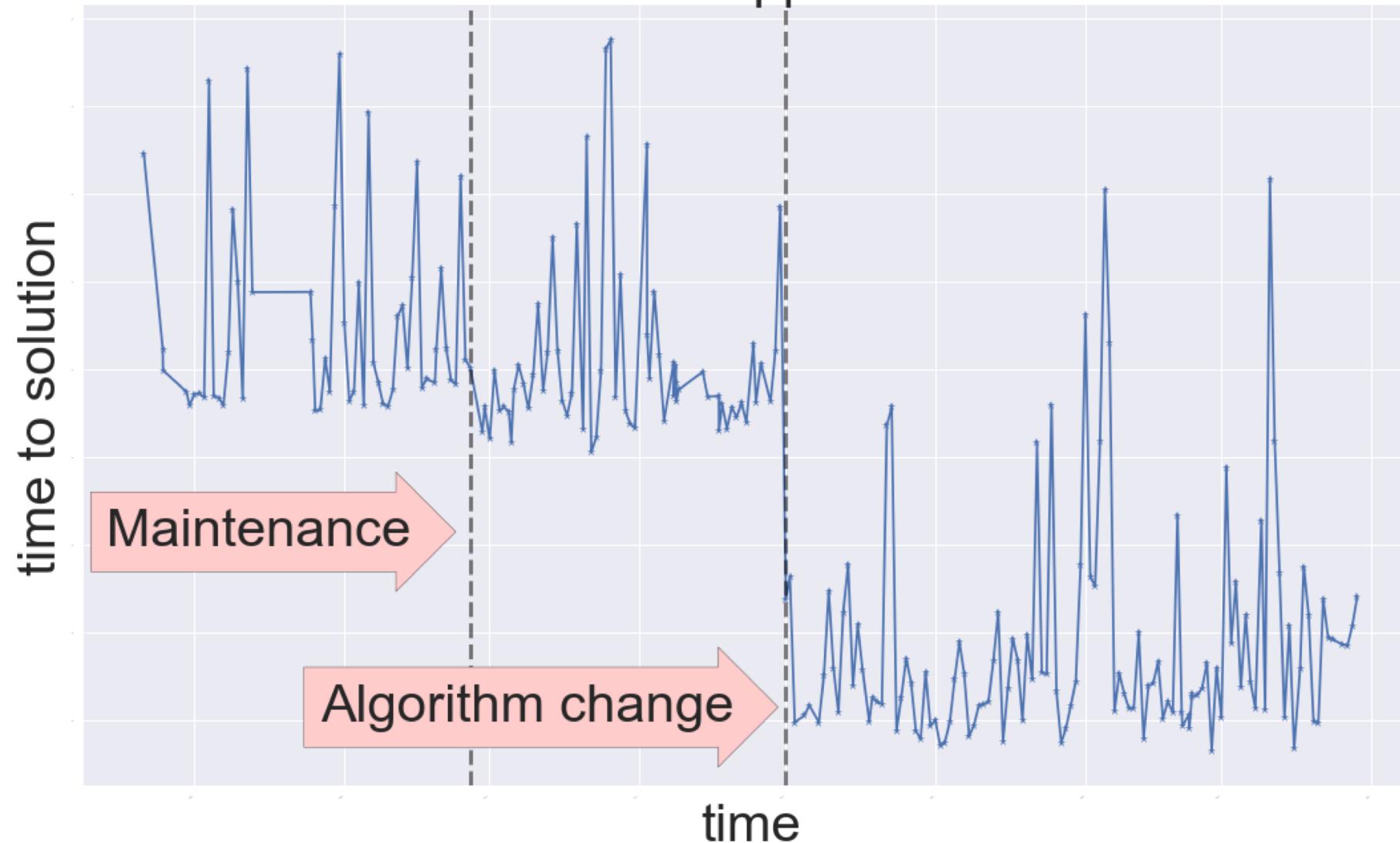
The CSCS Use Case

An application



The CSCS Use Case

Another application



The CSCS Use Case

Comparison to our former shell script based solution

Maintenance Burden	Shell-script based suite	ReFrame
Total size of tests	14635 loc	2985 loc
Average test file size	179 loc	93 loc
Average effective test size	179 loc	25 loc

5x reduction in the amount of code of regression tests

Open development

- Development of ReFrame is open on Github
 - <https://github.com/eth-cscs/reframe>
- Actively developed
 - New features and enhancements are added every month
 - Bugs are addressed promptly
- Quick release cycle (2-3 weeks)
- Hundreds of realistic regression tests
- Full documentation
 - Github.io page: <https://eth-cscs.github.io/reframe/index.html>
 - Step-by-step tutorial
 - Reference guide

Summary

ReFrame leverages the complexity of regression testing of HPC systems and paves the way for enabling continuous integration of HPC applications.

- *Decouples the logic of the tests from the system details.*
- *Lets you write portable regression tests and decreases their maintenance cost.*
- *Lets you write your regression tests in a modern programming language.*

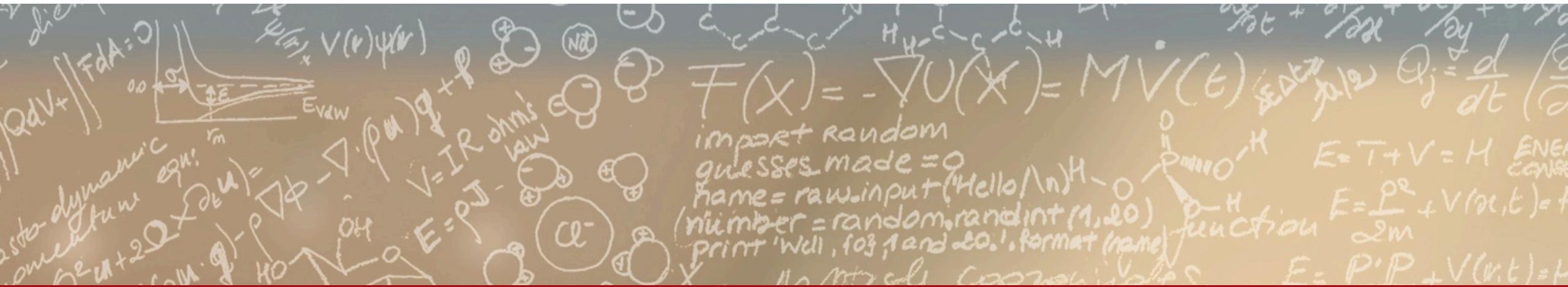
Try it out, give us some feedback and why not contribute back!



CSCS

Centro Svizzero di Calcolo Scientifico
Swiss National Supercomputing Centre

ETH zürich



Thank you for your attention.

ReFrame

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