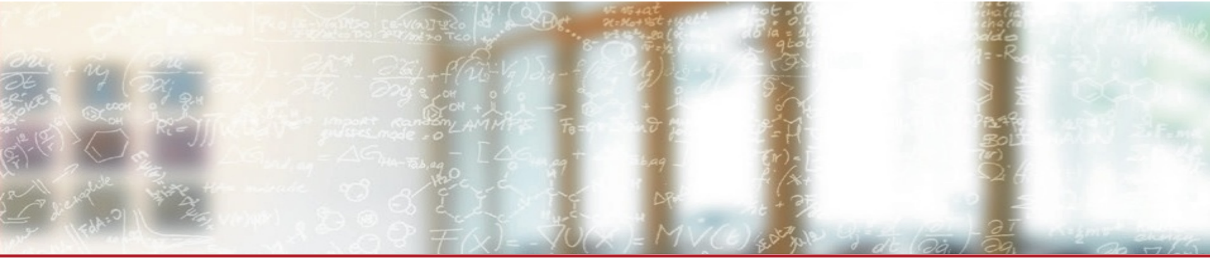
CSCS
Centro Svizzero di Calcolo Scientifico
Swiss National Supercomputing Centre

ETH zürich



Best Practices in HPC: Application and Programming Environment Testing

CSCS User Lab Day 2020

Vasileios Karakasis, Scientific Computing Support Group Lead, CSCS

August 31, 2020

Enabling science at CSCS

- Allow scientists to focus on their science, minimizing distractions from system issues, offering them a sane environment with all the tools they need
- Help scientists understand the system and how it affects their code’s functionality and performance and help them optimize it for the system
- Work with scientists for implementing and better supporting their workflows
- Onboard the scientists to new HPC technologies and help them make the best out of them

Providing a sane environment to scientists

- How can we ensure that the user experience is unaffected after a system upgrade or after an “innocent” change somewhere in the system?

- How testing of such complex systems can be made sustainable?
  - Consistency
  - Maintainability
  - Portability
  - Automation
  - Efficiency

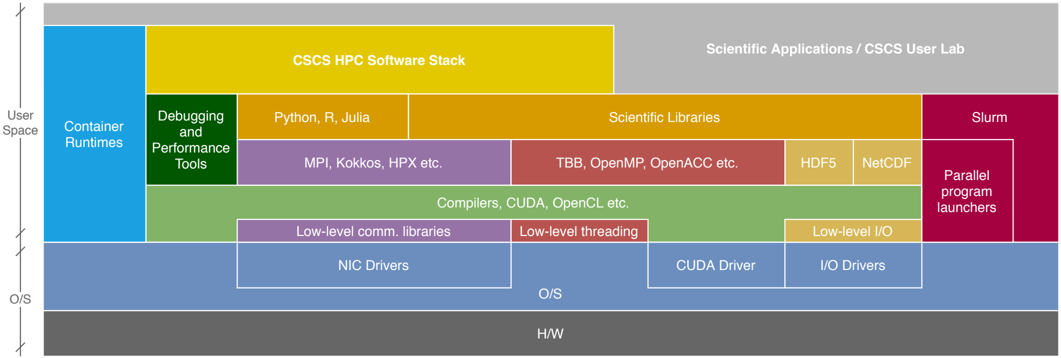
Testing: a big challenge overall!

- Writing proper tests require the same level of engineering effort as the application they test!
- Much less attractive to write
- As opposed to features, the value of tests is seldom visible in the short term
- Testing has several levels
- Automating tests becomes essential as projects grow
- Testing can never be complete for real-world applications

HPC system testing challenges

- Multiple interacting components
- Multiple programming environments
- Multiple libraries
- Multiple applications
- Multiple architectures
- Multiple clusters
- Functionality and performance are both important

A (very) simplified view of the scientific software stack



User Space

O/S

Scientific Applications / CSCS User Lab

CSCS HPC Software Stack

Container Runtimes

Debugging and Performance Tools

Python, R, Julia

Scientific Libraries

Slurm

MPI, Kokkos, HPX etc.

TBB, OpenMP, OpenACC etc.

HDF5

NetCDF

Parallel program launchers

Compilers, CUDA, OpenCL etc.

Low-level comm. libraries

Low-level threading

Low-level I/O

NIC Drivers

CUDA Driver

I/O Drivers

O/S

H/W

The HPC system testing landscape

- No or minimal testing; users discover the problems and open tickets

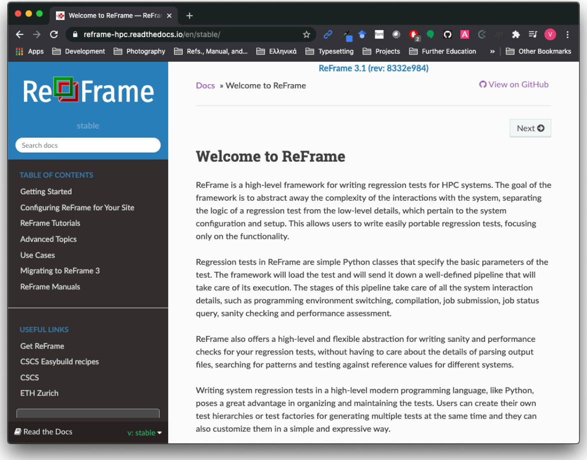
- Manual testing by the center's staff

- Ad-hoc, very site-specific "frameworks"
  - Non-portable tests
  - Lots of unnecessary test code
  - High maintenance costs
  - Low test coverage

The CSCS solution – ReFrame

ReFrame is a generic HPC 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,
- provides a runtime for running **efficiently** the regression tests.



ReFrame 3.1 (rev: 8332e984)

Docs » Welcome to ReFrame

View on GitHub

TABLE OF CONTENTS

- Getting Started
- Configuring ReFrame for Your Site
- ReFrame Tutorials
- Advanced Topics
- Use Cases
- Migrating to ReFrame 3
- ReFrame Manuals

USEFUL LINKS

- Get ReFrame
- CSCS Easybuild recipes
- CSCS
- ETH Zurich

Welcome to ReFrame

ReFrame is a high-level framework for writing regression tests for HPC systems. The goal of the 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.

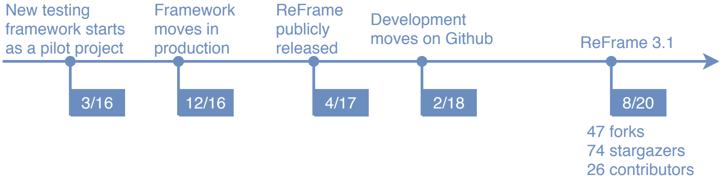
ReFrame also offers a high-level and flexible abstraction for writing sanity and performance checks for your regression tests, without having to care about the details of parsing output files, searching for patterns and testing against reference values for different systems.

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.

Design goals

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

ReFrame timeline



New testing framework starts as a pilot project

3/16

Framework moves in production

12/16

ReFrame publicly released

4/17

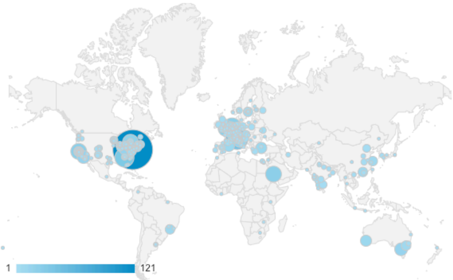
Development moves on Github

2/18

ReFrame 3.1

8/20

47 forks
74 stargazers
26 contributors



Documentation readers in 2020.

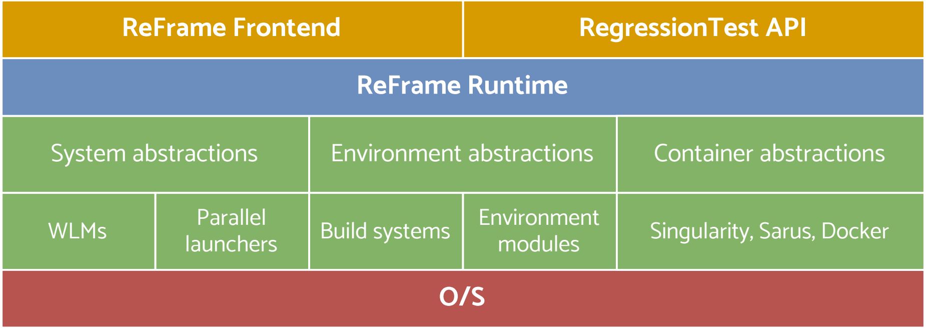
Key features

- Support for cycling through programming environments and system partitions
- Support for different WLMs, parallel job launchers and modules systems
- Support for sanity and performance tests
- Support for test factories
- Support for container runtimes
- Support for test dependencies
- Concurrent execution of regression tests
- Progress and result reports
- Performance logging
- Clean internal APIs that allow the easy extension of the framework's functionality

ReFrame's architecture

```
reframe <options> -r
```

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



ReFrame Frontend

RegressionTest API

ReFrame Runtime

System abstractions

Environment abstractions

Container abstractions

WLMs

Parallel launchers

Build systems

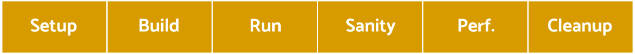
Environment modules

Singularity, Sarus, Docker

O/S

How ReFrame executes tests

All tests go through a well-defined pipeline.



Setup

Build

Run

Sanity

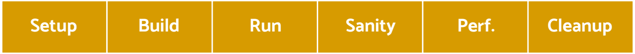
Perf.

Cleanup

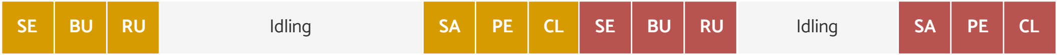
The regression test pipeline

How ReFrame executes tests

All tests go through a well-defined pipeline.


Setup | Build | Run | Sanity | Perf. | Cleanup

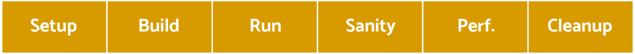
The regression test pipeline


SE | BU | RU | Idling | SA | PE | CL | SE | BU | RU | Idling | SA | PE | CL

Serial execution policy

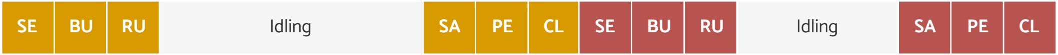
How ReFrame executes tests

All tests go through a well-defined pipeline.



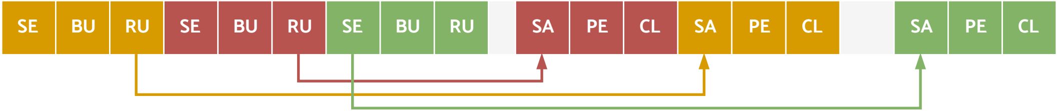
Setup | Build | Run | Sanity | Perf. | Cleanup

The regression test pipeline



SE | BU | RU | Idling | SA | PE | CL | SE | BU | RU | Idling | SA | PE | CL

Serial execution policy



SE | BU | RU | SE | BU | RU | SE | BU | RU | SA | PE | CL | SA | PE | CL | SA | PE | CL

Asynchronous execution policy

A “Hello, World!” ReFrame test

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


@rfm.simple_test
class HelloTest(rfm.RegressionTest):
    def __init__(self):
        self.valid_systems = ['*']
        self.valid_prog_environs = ['*']
        self.sourcepath = 'hello.c'
        self.sanity_patterns = sn.assert_found(r'Hello, World\!', self.stdout)
```

A “Hello, World!” ReFrame test

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


@rfm.simple_test
class HelloTest(rfm.RegressionTest):
    def __init__(self):
        self.valid_systems = ['*']
        self.valid_prog_environs = ['*']
        self.sourcepath = 'hello.c'
        self.sanity_patterns = sn.assert_found(r'Hello, World\!', self.stdout)
```

```
$ reframe -c tutorials/basics/hello/hello1.py -r
...
[==========] Running 1 check(s)
[==========] Started on Fri Jul 24 11:05:46 2020

[----------] started processing HelloTest (HelloTest)
[ RUN      ] HelloTest on generic:default using builtin
[----------] finished processing HelloTest (HelloTest)

[----------] waiting for spawned checks to finish
[       OK ] (1/1) HelloTest on generic:default using builtin [compile: 0.378s run: 0.299s total: 0.712s]
[----------] all spawned checks have finished

[  PASSED  ] Ran 1 test case(s) from 1 check(s) (0 failure(s))
[==========] Finished on Fri Jul 24 11:05:47 2020
```

A “Hello, World!” ReFrame test

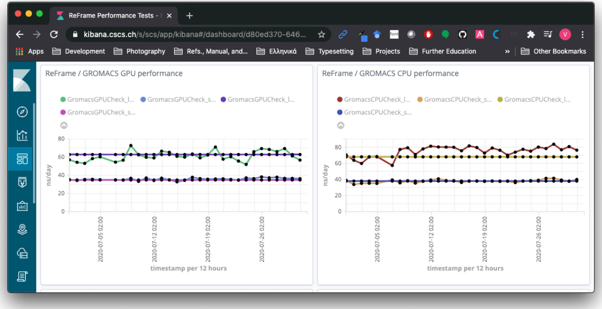
```python
import reframe as rfm
import reframe.utility.sanity as sn


@rfm.simple_test
class HelloTest(rfm.RegressionTest):
    def __init__(self):
        self.valid_systems = ['*']
        self.valid_prog_environs = ['*']
        self.sourcepath = 'hello.c'
        self.sanity_patterns = sn.assert_found(r'Hello, World\!', self.stdout)
```

See ReFrame tutorials for all the details: https://reframe-hpc.readthedocs.io/en/stable/tutorials.html

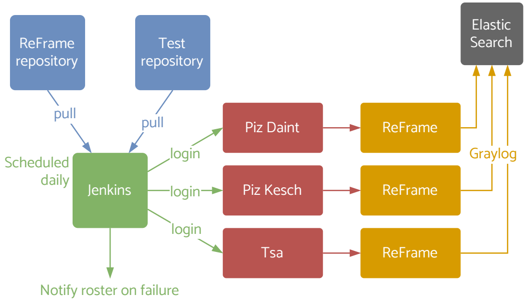
```
$ reframe -c tutorials/basics/hello/hello1.py -r
...
[==========] Running 1 check(s)
[==========] Started on Fri Jul 24 11:05:46 2020

[----------] started processing HelloTest (HelloTest)
[ RUN      ] HelloTest on generic:default using builtin
[----------] finished processing HelloTest (HelloTest)

[----------] waiting for spawned checks to finish
[       OK ] (1/1) HelloTest on generic:default using builtin [compile: 0.378s run: 0.299s total: 0.712s]
[----------] all spawned checks have finished

[  PASSED  ] Ran 1 test case(s) from 1 check(s) (0 failure(s))
[==========] Finished on Fri Jul 24 11:05:47 2020
```

Performance monitoring

- Every time a performance test is run, ReFrame can log its performance through several channels (normal files, Syslog, Graylog)



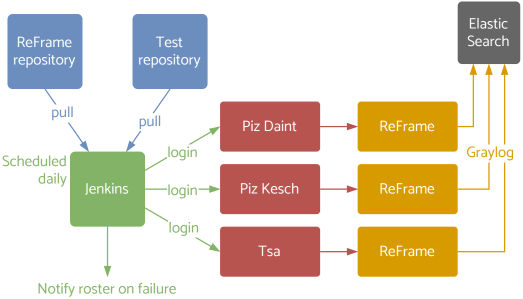
Continuous software stack and system testing



Several test categories identified by tags:

- Cray PE tests: only PE functionality
- Production tests: entire HPC software stack
- Maintenance tests: selection of tests for running before/after maintenance sessions
- Benchmarks
- > 350 tests reused across systems

Continuous software stack and system testing



Several test categories identified by tags:

- Cray PE tests: only PE functionality
- Production tests: entire HPC software stack
- Maintenance tests: selection of tests for running before/after maintenance sessions
- Benchmarks
- > 350 tests reused across systems

Experiences from Piz Daint:

- Enabling ReFrame as early as possible during a system upgrade streamlines the process
- Reveals several regressions in the programming environment that need to be fixed
- Builds confidence when finally everything is GREEN
- During production operation, it highlights possible system problems

CSCS ReFrame test suite

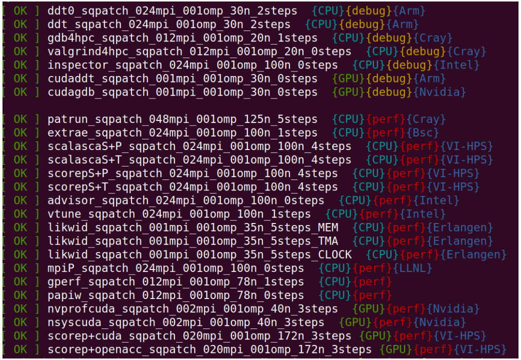
- HPC applications: Amber, CP2K, CPMD, QuantumEspresso, GROMACS, LAMMPS, NAMD, OpenFoam, Paraview, TensorFlow
- Libraries: Boost, GridTools, HPX, HDF5, NetCDF, Magma, Scalapack, Trilinos, PETSc
- Programming environment: GPU, MPI, MPI+X functionality, OpenACC, CPU affinity
- Slurm functionality
- Performance and debugging tools
- I/O tests: IOR
- Microbenchmarks: CUDA, CPU, MPI
- Container runtime checks
- OpenStack: S3 API

– Check the "cscs-checks/" directory @ https://github.com/eth-cscs/reframe
– Debugger and performance tools https://github.com/eth-cscs/hpctools

ReFrame at other sites

- National Energy Research Scientific Computing Center, USA
  - Software stack validation
  - Performance testing and benchmarking
  - Integration with Gitlab CI/CD solution developed within ECP
  - V. Karakasis et al., “Enabling Continuous Testing of HPC Systems using ReFrame”, HUST’19
- Ohio Supercomputing Center, USA
  - Software stack validation
  - Integration with CI/CD
  - S. Khuvis et al., “A Continuous Integration-Based Framework for Software Management”, PEARC’19
- KAUST (SA), PAWSEY (AUS), NIWA (NZ), GATech (USA), Univ. of Birmingham (UK) and many more.

Advanced application testing and performance analysis



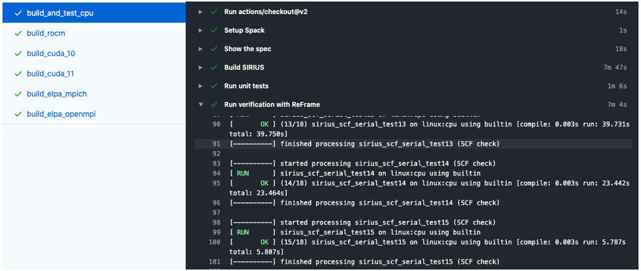
```
[ OK ] ddt0_sqpatch_024mpi_001omp_30n_2steps  {CPU}{debug}{Arm}
[ OK ] ddt_sqpatch_024mpi_001omp_30n_2steps  {CPU}{debug}{Arm}
[ OK ] gdb4hpc_sqpatch_012mpi_001omp_20n_1steps  {CPU}{debug}{Cray}
[ OK ] valgrind4hpc_sqpatch_012mpi_001omp_20n_0steps  {CPU}{debug}{Cray}
[ OK ] inspector_sqpatch_024mpi_001omp_100n_0steps  {CPU}{debug}{Intel}
[ OK ] cudaddt_sqpatch_001mpi_001omp_30n_0steps  {GPU}{debug}{Arm}
[ OK ] cudagdb_sqpatch_001mpi_001omp_30n_0steps  {GPU}{debug}{Nvidia}

[ OK ] patrun_sqpatch_048mpi_001omp_125n_5steps  {CPU}{perf}{Cray}
[ OK ] extrae_sqpatch_024mpi_001omp_100n_1steps  {CPU}{perf}{Bsc}
[ OK ] scalascaS+P_sqpatch_024mpi_001omp_100n_4steps  {CPU}{perf}{VI-HPS}
[ OK ] scalascaS+T_sqpatch_024mpi_001omp_100n_4steps  {CPU}{perf}{VI-HPS}
[ OK ] scorepS+P_sqpatch_024mpi_001omp_100n_4steps  {CPU}{perf}{VI-HPS}
[ OK ] scorepS+T_sqpatch_024mpi_001omp_100n_4steps  {CPU}{perf}{VI-HPS}
[ OK ] advisor_sqpatch_024mpi_001omp_100n_0steps  {CPU}{perf}{Intel}
[ OK ] vtune_sqpatch_024mpi_001omp_100n_1steps  {CPU}{perf}{Intel}
[ OK ] likwid_sqpatch_001mpi_001omp_35n_5steps_MEM  {CPU}{perf}{Erlangen}
[ OK ] likwid_sqpatch_001mpi_001omp_35n_5steps_TMA  {CPU}{perf}{Erlangen}
[ OK ] likwid_sqpatch_001mpi_001omp_35n_5steps_CLOCK  {CPU}{perf}{Erlangen}
[ OK ] mpiP_sqpatch_024mpi_001omp_100n_0steps  {CPU}{perf}{LLNL}
[ OK ] gperf_sqpatch_012mpi_001omp_78n_1steps  {CPU}{perf}
[ OK ] papiw_sqpatch_012mpi_001omp_78n_0steps  {CPU}{perf}
[ OK ] nvprofcuda_sqpatch_002mpi_001omp_40n_3steps  {GPU}{perf}{Nvidia}
[ OK ] nsyscuda_sqpatch_002mpi_001omp_40n_3steps  {GPU}{perf}{Nvidia}
[ OK ] scorep+cuda_sqpatch_020mpi_001omp_172n_3steps {GPU}{perf}{VI-HPS}
[ OK ] scorep+openacc_sqpatch_020mpi_001omp_172n_3steps {GPU}{perf}{VI-HPS}
```

- The hpctools repository showcases how to use ReFrame together with HPC tools. It is designed to
  - contribute to CSCS effort in automated regression testing,
  - demonstrate the usage of debuggers and performance tools,
  - share ReFrame checks.

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

Application CI testing with ReFrame

- SIRIUS library uses ReFrame for running its verification tests
  - Tests are located in the repository
  - Tests are triggered on very PR as a separate step in the CI pipeline
  - ReFrame is fetched on-the-fly and runs the tests
  - The same tests can be easily reused for different target systems



```
▸ ✓ Run actions/checkout@v2                14s
▸ ✓ Setup Spack                             1s
▸ ✓ Show the spec                          18s
▸ ✓ Build SIRIUS                        7m 47s
▸ ✓ Run unit tests                      1m 6s
▾ ✓ Run verification with ReFrame       7m 4s
[       OK ] (13/18) sirius_scf_serial_test13 on linux:cpu using builtin [compile: 0.003s run: 39.731s
total: 39.750s]
[----------] finished processing sirius_scf_serial_test13 (SCF check)

[----------] started processing sirius_scf_serial_test14 (SCF check)
[ RUN      ] sirius_scf_serial_test14 on linux:cpu using builtin
[       OK ] (14/18) sirius_scf_serial_test14 on linux:cpu using builtin [compile: 0.003s run: 23.442s
total: 23.464s]
[----------] finished processing sirius_scf_serial_test14 (SCF check)

[----------] started processing sirius_scf_serial_test15 (SCF check)
[ RUN      ] sirius_scf_serial_test15 on linux:cpu using builtin
[       OK ] (15/18) sirius_scf_serial_test15 on linux:cpu using builtin [compile: 0.003s run: 5.787s
total: 5.807s]
[----------] finished processing sirius_scf_serial_test15 (SCF check)
```

https://github.com/electronic-structure/SIRIUS

ReFrame community

- Mailing list (25 members): reframe@cscs.ah
- Slack channel (59 members): https://reframe-slack.herokuapp.com/
- ReFrame test repositories: https://github.com/reframe-hpc

Contributing tests for Piz Daint

- CSCS users, particularly those with large allocations, are welcome to contribute ReFrame tests that exercise their application or workflow
  - The tests will become part of our test battery that runs on upgrades
  - The tests should be short, self-contained and tested on Piz Daint
  - Users should maintain their tests
  - Users can contribute through a pull request in the project's repository or contact us at help@cscs.ch

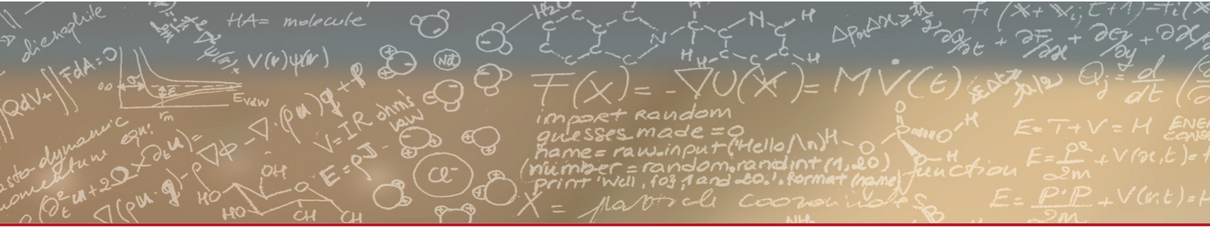
- ReFrame is already installed and configured on Piz Daint
  - `module load reframe`
  - https://user.cscs.ch/tools/reframe/

Conclusions

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
- Easy integration with CI/CD workflows

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



Thank you for your attention

- ✉ reframe@cscs.ch
- https://reframe-hpc.readthedocs.io
- https://github.com/eth-cscs/reframe
- https://reframe-slack.herokuapp.com