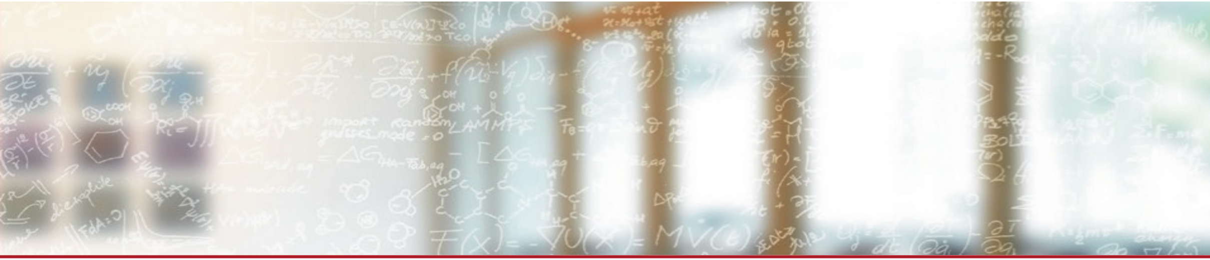
CSCS
Centro Svizzero di Calcolo Scientifico
Swiss National Supercomputing Centre

ETH zürich



Writing powerful HPC regression tests with ReFrame

6th EasyBuild User Meeting, 2021

Vasileios Karakasis, Scientific Computing Support Group Lead, CSCS

January 25, 2021

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

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

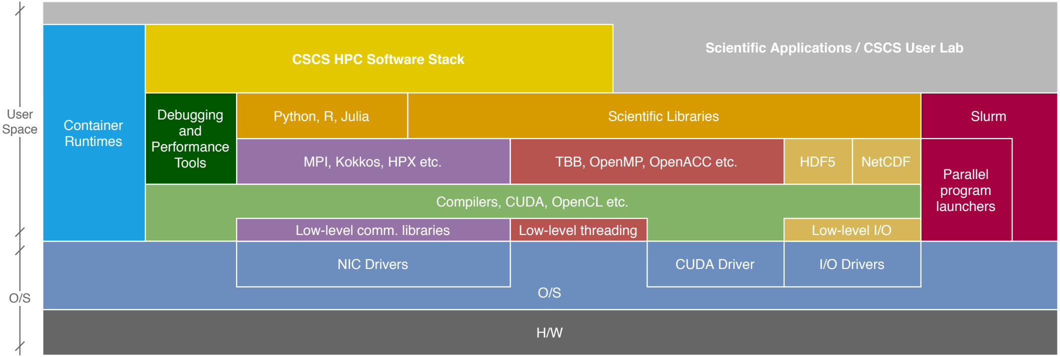
Testing: a big challenge overall!

- 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



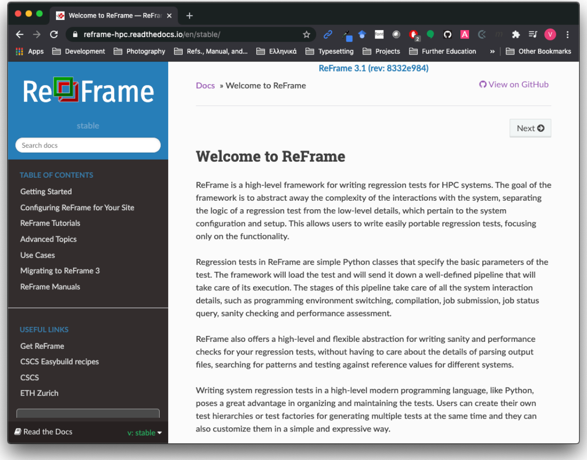
The HPC system testing landscape

- 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

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.

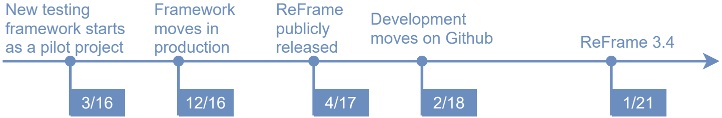
ReFrame also offers a high-level and flexible abstraction for writing sanity and performance checks for your regression tests, without having to care about the details of parsing output files, searching for patterns and testing against reference values for different systems.

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

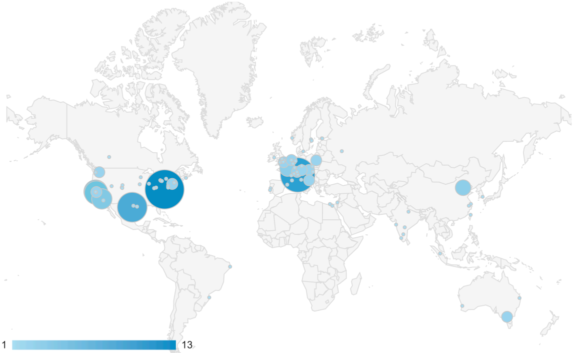
Framework moves in production — 12/16

ReFrame publicly released — 4/17

Development moves on Github — 2/18

ReFrame 3.4 — 1/21

50 forks
83 stars
28 contributors



Documentation readers in December 2020.

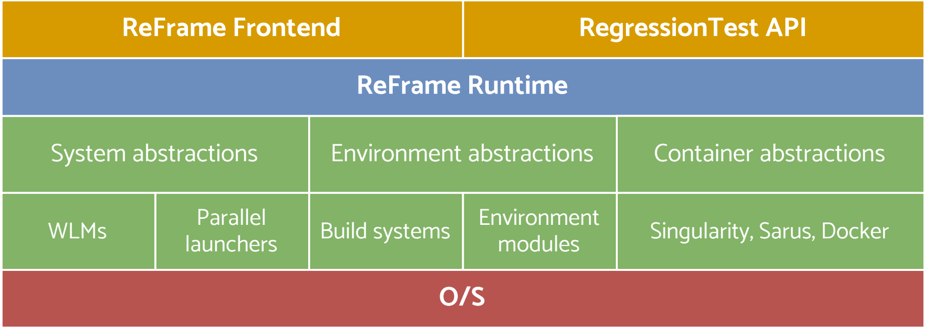
Key features

- 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

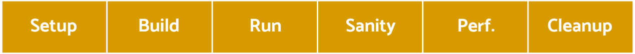
Environment modules

Singularity, Sarus, Docker

O/S

How ReFrame executes tests

All tests go through a well-defined pipeline.

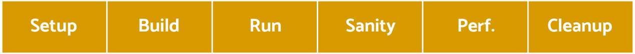


Setup | Build | Run | Sanity | Perf. | Cleanup

The regression test pipeline

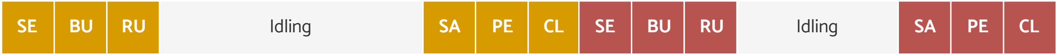
How ReFrame executes tests

All tests go through a well-defined pipeline.



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

The regression test pipeline

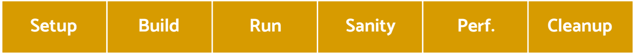


SE | BU | RU | Idling | SA | PE | CL | SE | BU | RU | Idling | SA | PE | CL

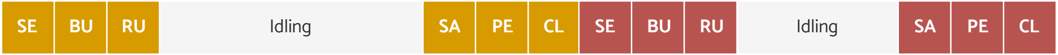
Serial execution policy

How ReFrame executes tests

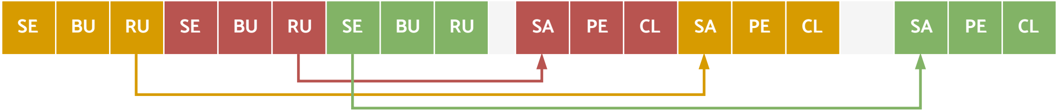
All tests go through a well-defined pipeline.



The regression test pipeline



Serial execution policy



Asynchronous execution policy

Configuring ReFrame

- Configuration is a big JSON object stored in a JSON or Python file
- Framework's behaviour can also be changed through environment variables or command-line
- Three key sections that hold the system-specific details:
  1. Systems
     - Hostname patterns that will let ReFrame recognize a system
     - Modules system used
     - Define system's virtual partitions
  2. Virtual partitions
     - Job scheduler and parallel job launcher
     - How access to this partition is granted
     - The programming environments to be tested on this partition
  3. Programming environments (toolchains)
     - Environment modules to load
     - Environment variables to set

Configuring ReFrame

```
'systems': [
  ...
  {
    'name': 'daint',
    'descr': 'Piz Daint Supercomputer',
    'hostnames': ['daint'],
    'modules_system': 'tmod32',
    'partitions': [
      {
        'name': 'gpu',
        'descr': 'Hybrid nodes',
        'scheduler': 'slurm',
        'launcher': 'srun',
        'access': ['-C gpu', '-A csstaff'],
        'environs': ['gnu', 'intel', 'pgi', 'cray'],
        'container_platforms': [
          {
            'type': 'Singularity',
            'modules': ['singularity']
          }
        ],
        'max_jobs': 100
      },
      ...
    ]
  },
  ...
]
```

```
'environments': [
  {
    'name': 'gnu',
    'modules': ['PrgEnv-gnu'],
    'cc': 'cc',
    'cxx': 'CC',
    'ftn': 'ftn',
    'target_systems': ['daint']
  },
  {
    'name': 'cray',
    'modules': ['PrgEnv-cray'],
    'cc': 'cc',
    'cxx': 'CC',
    'ftn': 'ftn',
    'target_systems': ['daint']
  },
  ...
],
...
```

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

```
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

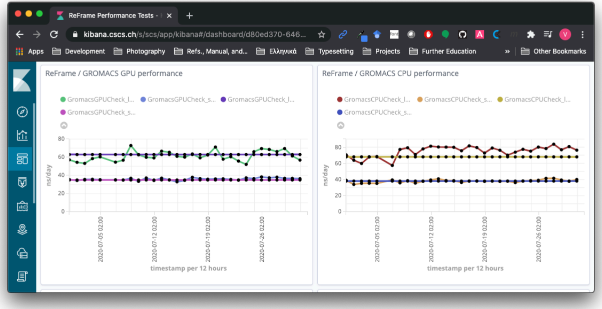
```python
import reframe as rfm
import reframe.utility.sanity as sn


@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

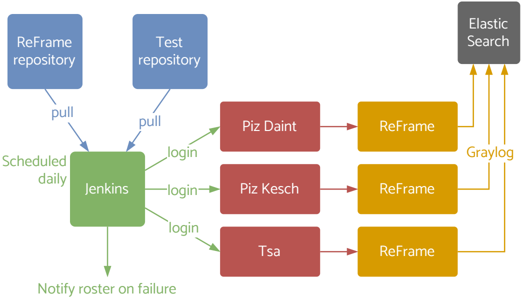
```
$ reframe -c tutorials/basics/hello/hello1.py -r
...
[==========] Running 1 check(s)
[==========] Started on Fri Jul 24 11:05:46 2020

[----------] 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 (regular files, Syslog, Graylog)



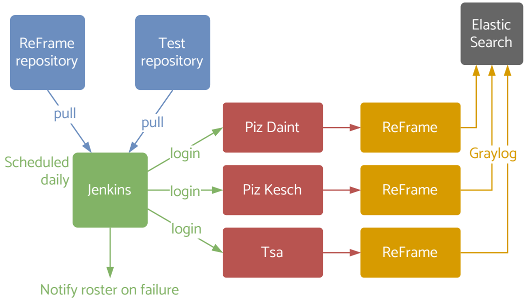
Continuous software stack and system testing



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, PyTorch
- 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

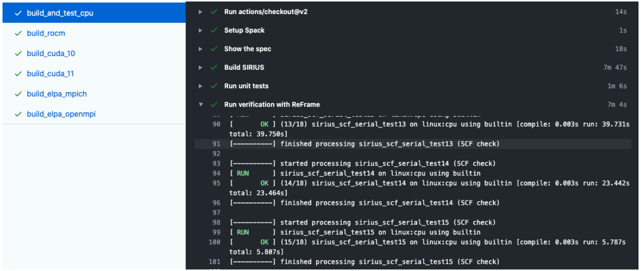
– Check the “cscs-checks/” directory @ https://github.com/eth-cscs/reframe
– Debugger and performance tools https://github.com/eth-cscs/hpctools

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.

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



✓ build_and_test_cpu
✓ build_rocm
✓ build_cuda_10
✓ build_cuda_11
✓ build_elpa_mpich
✓ build_elpa_openmpi

▸ ✓ 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 (27 members): reframe@cscs.ch
- Slack channel (66 members): https://reframe-slack.herokuapp.com/
- ReFrame test repositories: https://github.com/reframe-hpc

New Features since EUM'20

- ReFrame 3.0 (breaking changes)
  - Python 3.5 support was dropped
  - Completely revised configuration mechanism; old configuration files are no more valid
  - Overriding pipeline methods was deprecated; use the `@run_before()` and `@run_after()` decorators instead
- Straightforward installation through a bootstrap script
- The asynchronous execution policy is now the default
- Execution time profiling and progress report
- Improved and more detailed log messages that help debugging
- Detailed JSON report at the end of each run session
- Allow automatic test failures on non-zero exit codes
- Module crawling utility function to allow parameterization of tests per environment module
- Support for building tests remotely
- Allow dependencies across partitions and improved dependency handling
- Support for module collections
- Better verbosity control
- New powerful syntax for parameterized tests that allows you to dynamically expand the parameterization space (new in 3.4)
- Support for `spack load` for loading "modules" (new in 3.4)

Other tools worth looking at

- BuildTest
  - https://buildtest.readthedocs.io/
  - Talk by Shahzeb Siddiqui on Fri. 29, 2021 @ 15:00 UTC
- Pavilion2
  - https://pavilion2.readthedocs.io/

What’s next?

- Work towards test libraries and composable tests
  - New syntax elements and enhancements
  - https://github.com/eth-cscs/reframe/projects/23
- Gitlab integration
  - Use ReFrame to generate dynamic pipelines for running tests through Gitlab
  - https://github.com/eth-cscs/reframe/pull/1641
- Improvements in the runtime for increasing concurrency

Touching base with ReFrame development

- Stable releases every 6 weeks, dev releases every two.
  - Train model: whatever is ready, gets in, whatever not, gets in the next one
- We will stick to semantic versioning
- Upcoming release schedule: https://github.com/eth-cscs/reframe/projects/
- Sprints: https://github.com/eth-cscs/reframe/milestones
- Core Dev Team: @vkarak, @teojgo, @rsarm, @ekouts, @victorusu

Touching base with ReFrame development

- Stable releases every 6 weeks, dev releases every two.
  - Train model: whatever is ready, gets in, whatever not, gets in the next one
- We will stick to semantic versioning
- Upcoming release schedule: https://github.com/eth-cscs/reframe/projects/
- Sprints: https://github.com/eth-cscs/reframe/milestones
- Core Dev Team: @vkarak, @teojgo, @rsarm, @ekouts, @victorusu

DISCLAIMER

- We are not full time on it!
  - Issues might be late to catch the “release train”
  - Issues might get spilled over to subsequent sprints
  - Priorities might change based on our needs

Touching base with ReFrame development

- Stable releases every 6 weeks, dev releases every two.
  - Train model: whatever is ready, gets in, whatever not, gets in the next one
- We will stick to semantic versioning
- Upcoming release schedule: https://github.com/eth-cscs/reframe/projects/
- Sprints: https://github.com/eth-cscs/reframe/milestones
- Core Dev Team: @vkarak, @teojgo, @rsarm, @ekouts, @victorusu

DISCLAIMER

- We are not full time on it!
  - Issues might be late to catch the "release train"
  - Issues might get spilled over to subsequent sprints
  - Priorities might change based on our needs

Contributions are more than welcome!

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
- Powerful runtime

- Help → mailing list, Slack, Github
- Bug reports, feature requests → Github

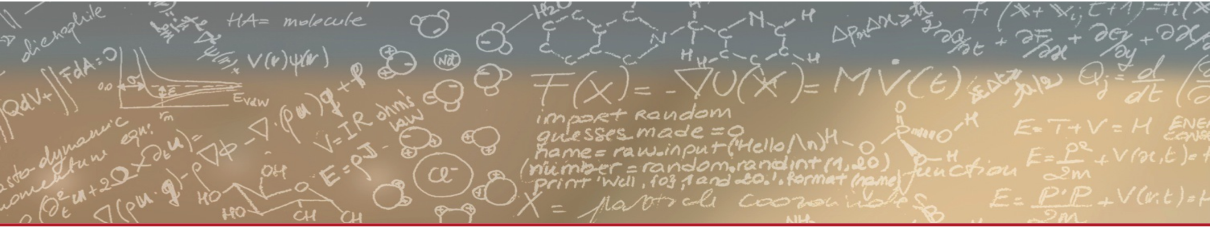
https://github.com/eth-cscs/reframe

Some logistics about the tutorial

- Tue @ 13:15 UTC, Thu @ 10:15 UTC, Fri @ 13:00 UTC
- Please reply to Victor's e-mail by sending your SSH public key for the access to the cluster
- Please join the `#tutorial-eum21` channel in ReFrame's slack
  - https://reframe-slack.herokuapp.com

CSCS
Centro Svizzero di Calcolo Scientifico
Swiss National Supercomputing Centre

ETH zürich


HA= molecule
V(r)ψ(r)
V=IR ohm's law
import random
guesses_made = 0
name = raw_input('Hello\n')
number = random.randint(1, 20)
print 'Well, {0}, I and 20.'.format(name)
function
E = T + V = H

Thank you for your attention

reframe@cscs.ch

https://reframe-hpc.readthedocs.io

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

https://reframe-slack.herokuapp.com

@ReFrameHPC