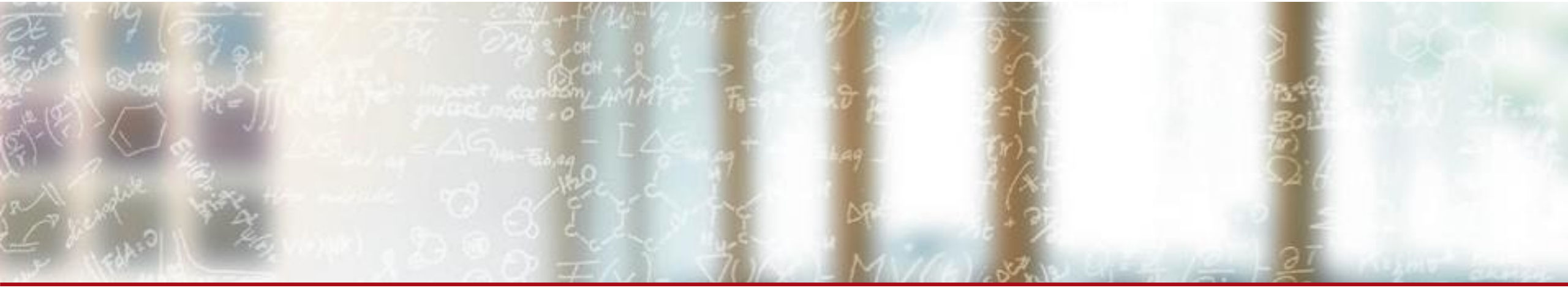
CSCS
Centro Svizzero di Calcolo Scientifico
Swiss National Supercomputing Centre

ETH zürich



Embracing ReFrame Programmable Configurations

Manitaras Theofilos-Ioannis, CSCS

8th EasyBuild User Meeting

April 24, 2023

Outline

- ReFrame configuration overview
- Programmable configuration for containers
- User-Environment based programmable configuration
- Conclusions

ReFrame configuration overview

The two flavors of ReFrame configuration (1 / 2)

```python
site_configuration = {
    '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', 'nvidia', 'cray'],
                    'max_jobs': 100,
                },
                .
                .
                .
            ]
        }
    ],
    'environments': [
        {
            'name': 'gnu',
            'modules': ['PrgEnv-gnu'],
            'cc': 'cc',
            'cxx': 'CC',
            'ftn': 'ftn',
            'target_systems': ['daint']
        },
         .
         .
         .
}
```

Python

```json
{
  "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",
            "nvidia",
            "cray"
          ],
          "max_jobs": 100
        },
        .
        .
  ],
  "environments": [
    {
      "name": "gnu",
      "modules": [
        "PrgEnv-gnu"
      ],
      "cc": "cc",
      "cxx": "CC",
      "ftn": "ftn",
      "target_systems": [
        "daint"
      ]
    },
    .
    .
  ],
}
```

JSON

The two flavors of ReFrame configuration (2 / 2)

- ReFrame can work with both the Python and the JSON flavors
- It is easy to convert from Python to JSON using the **--show-config** cli option
- The Python one is fully programmable using the language and its libraries
- ReFrame 4.0 and beyond supports splitting the configuration into multiple files
- For more information on the various options refer to the documentation

Programmable configuration for containers

Container Image Labels

Using the LABEL instruction in a Dockerfile, we can add information to a container image to be used by ReFrame

```
LABEL "rfm.features"="osu-micro-benchmarks;mpi;serial;openmp;cuda"
LABEL "rfm.cc"="mpicc"
LABEL "rfm.cxx"="mpic++"
LABEL "rfm.ftn"="mpif90"
```

```
LABEL "rfm.features"="osu-micro-benchmarks;mpi;serial;openmp"
LABEL "rfm.cc"="mpicc"
LABEL "rfm.cxx"="mpic++"
LABEL "rfm.ftn"="mpif90"
```

```
LABEL "rfm.features"="lammps"
```

```
LABEL "rfm.features"="gromacs;cuda"
```

Accessible by the programmable configuration at runtime

Programmable configuration workflow (1/2)

```
images = os.environ.get('RFM_CONTAINER_IMAGE', None)

if images:
    images = images.split(';')

environs = []

if images is not None:
   for im in images:
       image_feats = json.loads(
           subprocess.run(['skopeo', 'inspect', f'docker://{im}'],
                          stdout=subprocess.PIPE).stdout
       )

       features = image_feats['Labels']['rfm.features'].split(';')
       environ_names = [im.replace(":", "_").replace("/", "_")]

       for e in environ_names:
           env = {
               'name': e,
               'target_systems': ['daint'],
               'features': features,
               'env_vars': [['IMAGE', im]]
           }

       environs.append(env)
```

1. Get image names
2. Inspect each image using skopeo
3. Retrieve features and use image name as the environment one
4. Create an environment per image name for the target systems

Programmable configuration workflow (2/2)

```
if environs:
    partitions = [
        {
            'name': f'gpu',
            .
            .
            'environs': [e['name'] for e in environs],
            .
            .
            'features': ['gpu', 'nvgpu', 'remote'],
            .
            .
            .
        },
    ]
site_configuration = {
    'systems': [
        {
            'name': 'daint',
            .
            .
        },
    ],
    'environments': environs,
    .
    .
}
```

5. Assign the environment names

6. Put the actual environments in the site configuration

ReFrame tests with features

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

@rfm.simple_test
class OSULatency(rfm.RunOnlyRegressionTest):
    valid_systems = ['*']
    valid_prog_environs = ['+osu-micro-benchmarks']
    .
    .
    @run_after('setup')
    def setup_container_platform(self):
        self.container_platform.image = self.current_environ.env_vars['IMAGE']
        .
        .
@rfm.simple_test
class OSULatencyCuda(OSULatency):
    valid_systems = ['+nvgpu']
    valid_prog_environs = ['+osu-micro-benchmarks +cuda']
    .
    .
    .
@rfm.simple_test
class LAMMPSCPUCheck(rfm.RunOnlyRegressionTest):
    scale = parameter(['small', 'large'])
    valid_systems = ['*']
    valid_prog_environs = ['+lammps']
    .
    .
```

Required features

Retrieve image name injected by the configuration

Both the system and the environment have to provide the required features

Putting it all together

```
[rfmuser@daint ~]$ export RFM_CONTAINER_IMAGE='teojgo/mvapich2_osu:5.9_cuda;teojgo/mpich_osu:5.9;teojgo/lammps:latest'
[rfmuser@daint ~]$ reframe <cli options>
.
.
[==========] Running 6 check(s)
.
.
[----------] start processing checks
[ DRY      ] LAMMPSCPUCheck %scale=large /e4795b9b @daint:gpu+teojgo_lammps_latest
[       OK ] ( 1/14) LAMMPSCPUCheck %scale=large /e4795b9b @daint:gpu+teojgo_lammps_latest
[ DRY      ] LAMMPSCPUCheck %scale=large /e4795b9b @daint:mc+teojgo_lammps_latest
[       OK ] ( 2/14) LAMMPSCPUCheck %scale=large /e4795b9b @daint:mc+teojgo_lammps_latest
[ DRY      ] LAMMPSCPUCheck %scale=small /867ddbf9 @daint:gpu+teojgo_lammps_latest
[       OK ] ( 3/14) LAMMPSCPUCheck %scale=small /867ddbf9 @daint:gpu+teojgo_lammps_latest
[ DRY      ] LAMMPSCPUCheck %scale=small /867ddbf9 @daint:mc+teojgo_lammps_latest
[       OK ] ( 4/14) LAMMPSCPUCheck %scale=small /867ddbf9 @daint:mc+teojgo_lammps_latest
[ DRY      ] OSULatencyCuda /42896efa @daint:gpu+teojgo_mvapich2_osu_5.9_cuda
[       OK ] ( 5/14) OSULatencyCuda /42896efa @daint:gpu+teojgo_mvapich2_osu_5.9_cuda
[ DRY      ] OSUBandwidthCuda /9b75552e @daint:gpu+teojgo_mvapich2_osu_5.9_cuda
[       OK ] ( 6/14) OSUBandwidthCuda /9b75552e @daint:gpu+teojgo_mvapich2_osu_5.9_cuda
[ DRY      ] OSUBandwidth /bd272703 @daint:gpu+teojgo_mvapich2_osu_5.9_cuda
[       OK ] ( 7/14) OSUBandwidth /bd272703 @daint:gpu+teojgo_mvapich2_osu_5.9_cuda
[ DRY      ] OSUBandwidth /bd272703 @daint:gpu+teojgo_mpich_osu_5.9
[       OK ] ( 8/14) OSUBandwidth /bd272703 @daint:gpu+teojgo_mpich_osu_5.9
[ DRY      ] OSUBandwidth /bd272703 @daint:mc+teojgo_mvapich2_osu_5.9_cuda
[       OK ] ( 9/14) OSUBandwidth /bd272703 @daint:mc+teojgo_mvapich2_osu_5.9_cuda
[ DRY      ] OSUBandwidth /bd272703 @daint:mc+teojgo_mpich_osu_5.9
[       OK ] (10/14) OSUBandwidth /bd272703 @daint:mc+teojgo_mpich_osu_5.9
[ DRY      ] OSULatency /1583f36f @daint:gpu+teojgo_mvapich2_osu_5.9_cuda
[       OK ] (11/14) OSULatency /1583f36f @daint:gpu+teojgo_mvapich2_osu_5.9_cuda
[ DRY      ] OSULatency /1583f36f @daint:gpu+teojgo_mpich_osu_5.9
[       OK ] (12/14) OSULatency /1583f36f @daint:gpu+teojgo_mpich_osu_5.9
[ DRY      ] OSULatency /1583f36f @daint:mc+teojgo_mvapich2_osu_5.9_cuda
[       OK ] (13/14) OSULatency /1583f36f @daint:mc+teojgo_mvapich2_osu_5.9_cuda
[ DRY      ] OSULatency /1583f36f @daint:mc+teojgo_mpich_osu_5.9
[       OK ] (14/14) OSULatency /1583f36f @daint:mc+teojgo_mpich_osu_5.9
[----------] all spawned checks have finished
```

Images to test

One environment per image

Both system and environ features have to be satisfied

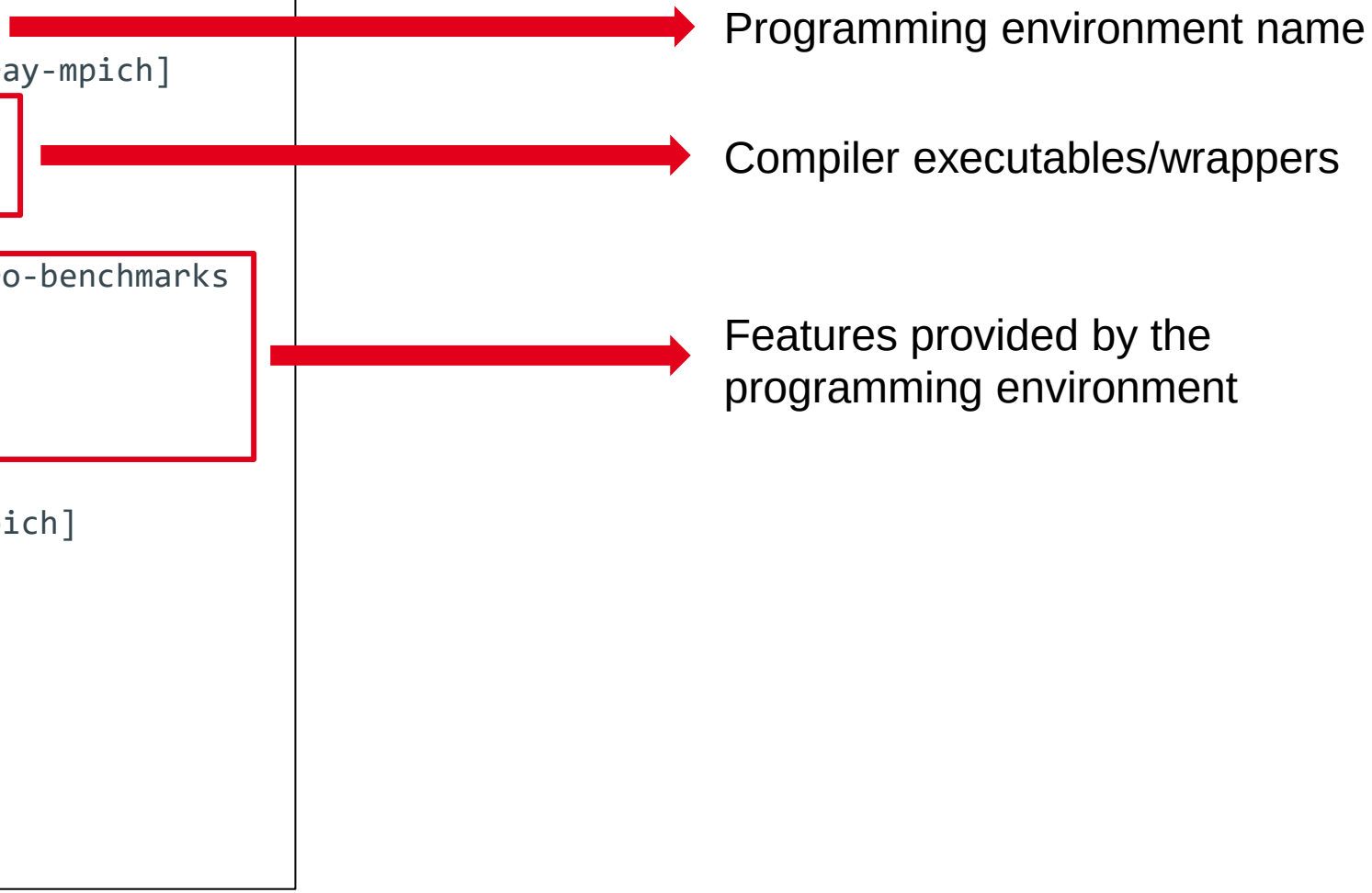
User environment based programmable configuration

CSCS User Environments in a Nutshell

- Squashfs images containing a software stack comprised of one or more programming environments
- Created using the CSCS developed stackinator cli application which accepts yaml files describing the components of the stack
- Mounted on a user selected path via a CSCS developed Slurm plugin
- Can make use of host libraries needed for best performance
- Are accompanied by a yaml file (work in progress) which is consumed by ReFrame the provided environments and their features

Example uenv describing the features per environment

```
environments:
  - name: cray
    modules: [cray-mpich]
    cc: mpicc
    cxx: mpic++
    ftn: mpif90
    features:
      - osu-micro-benchmarks
      - cuda
      - serial
      - openmp
      - mpi
  - name: lammps
    modules: [mpich]
    cc: mpicc
    cxx: mpic++
    ftn: mpif90
    features:
      - lammps
      - serial
      - openmp
      - mpi
```



Programmable configuration workflow (1/3)

```
uenv_file = os.environ.get('UENV_FILE', None)
uenv_mount = os.environ.get('UENV_MOUNT', '/user-environment')

uenv_access = []
uenv_modules_path = []
image_name = None
partitions = []
features = []

if uenv_file is not None:
    uenv_access = [
        f'--uenv-file={uenv_file}',
        f'--uenv-mount={uenv_mount}',
    ]
    uenv_modules_path = [f'module use {uenv_mount}/modules']
    image_path = pathlib.Path(uenv_file)
    image_name = image_path.stem

with open(image_path.parent / f'{image_name}.yaml') as image_envs:
    image_environments = yaml.load(image_envs.read(), Loader=yaml.BaseLoader)
```

1. Retrieve image path and mount point

2. Create Slurm options

3. Load the yaml file

Programmable configuration workflow (2/3)

```
if image_name is not None:
    environs = image_environments['environments']
    environ_names =  ([f'{image_name}_{e["name"]}'for e in environs] or
                      [f'{image_name}_builtin'])

    partitions = [
        {
            'name': f'nvgpu',
            .
            .
            'environs': environ_names,
            .
            .
            .
            'access': ['-pnvgpu'] + uenv_access,
            .
            .
            'features': ['gpu', 'nvgpu', 'remote', 'uenv'],
            .
            .
            'prepare_cmds': uenv_modules_path,
            'launcher': 'srun'
        },
        .
        .
    ]
```

4. Get the environment names

5. Use the above as the partition environments

6. Prepare the module path (optional)

Programmable configuration workflow (3/3)

```
environs = image_environments['environments']

if environs:
    actual_environs = []

for e in environs:
    env = {
        'target_systems': ['clariden-uenv']
    }
    env.update(e)
    env['name'] = f'{image_name}_{e["name"]}'
    actual_environs.append(env)

site_configuration = {
    'systems': [
        {
            'name': 'clariden-uenv',
            .
            .
            'partitions': partitions
        }
    ],
    .
    .
    'environments': actual_environs,
    .
    .
}
```

7. Create the actual environments

8. Assign the partitions

9. Assign the environments

Putting it all together

```
[rfmuser@uenv-cluster]$ export UENV_FILE=<path to uenv squashfs>
[rfmuser@uenv-cluster]$ reframe <cli options>
.
.
[==========] Running 28 check(s)
.
.
[----------] start processing checks
[ DRY      ] OSULatencyCuda /42896efa @clariden-uenv:nvgpu+cray-mpich_cuda11.8_osu59_cray
[       OK ] ( 1/54) OSULatencyCuda /42896efa @clariden-uenv:nvgpu+cray-mpich_cuda11.8_osu59_cray
[ DRY      ] OSUBandwidthCuda /9b75552e @clariden-uenv:nvgpu+cray-mpich_cuda11.8_osu59_cray
[       OK ] ( 2/54) OSUBandwidthCuda /9b75552e @clariden-uenv:nvgpu+cray-mpich_cuda11.8_osu59_cray
[ DRY      ] OSUBandwidth /bd272703 @clariden-uenv:nvgpu+cray-mpich_cuda11.8_osu59_cray
[       OK ] ( 3/54) OSUBandwidth /bd272703 @clariden-uenv:nvgpu+cray-mpich_cuda11.8_osu59_cray
.
.
[ DRY      ] OSULatency /1583f36f @clariden-uenv:amdgpu+cray-mpich_cuda11.8_osu59_cray
[       OK ] ( 6/54) OSULatency /1583f36f @clariden-uenv:amdgpu+cray-mpich_cuda11.8_osu59_cray
[ DRY      ] HelloWorldTestMPIOpenMP %linking=static %lang=f90 /0163156f @clariden-uenv:nvgpu+cray-mpich_cuda11.8_osu59_cray
[       OK ] ( 7/54) HelloWorldTestMPIOpenMP %linking=static %lang=f90 /0163156f @clariden-uenv:nvgpu+cray-mpich_cuda11.8_osu59_cray
[ DRY      ] HelloWorldTestMPIOpenMP %linking=static %lang=f90 /0163156f @clariden-uenv:amdgpu+cray-mpich_cuda11.8_osu59_cray
[       OK ] ( 8/54) HelloWorldTestMPIOpenMP %linking=static %lang=f90 /0163156f @clariden-uenv:amdgpu+cray-mpich_cuda11.8_osu59_cray
[ DRY      ] HelloWorldTestMPIOpenMP %linking=static %lang=cpp /417f4cbe @clariden-uenv:nvgpu+cray-mpich_cuda11.8_osu59_cray
.
.
[       OK ] (51/54) HelloWorldTestSerial %linking=dynamic %lang=cpp /c361f2ee @clariden-uenv:nvgpu+cray-mpich_cuda11.8_osu59_cray
[ DRY      ] HelloWorldTestSerial %linking=dynamic %lang=cpp /c361f2ee @clariden-uenv:amdgpu+cray-mpich_cuda11.8_osu59_cray
[       OK ] (52/54) HelloWorldTestSerial %linking=dynamic %lang=cpp /c361f2ee @clariden-uenv:amdgpu+cray-mpich_cuda11.8_osu59_cray
[ DRY      ] HelloWorldTestSerial %linking=dynamic %lang=c /dd310517 @clariden-uenv:nvgpu+cray-mpich_cuda11.8_osu59_cray
[       OK ] (53/54) HelloWorldTestSerial %linking=dynamic %lang=c /dd310517 @clariden-uenv:nvgpu+cray-mpich_cuda11.8_osu59_cray
[ DRY      ] HelloWorldTestSerial %linking=dynamic %lang=c /dd310517 @clariden-uenv:amdgpu+cray-mpich_cuda11.8_osu59_cray
[       OK ] (54/54) HelloWorldTestSerial %linking=dynamic %lang=c /dd310517 @clariden-uenv:amdgpu+cray-mpich_cuda11.8_osu59_cray
[----------] all spawned checks have finished
[  PASSED  ] Ran 54/54 test case(s) from 28 check(s) (0 failure(s), 0 skipped, 0 aborted)
```

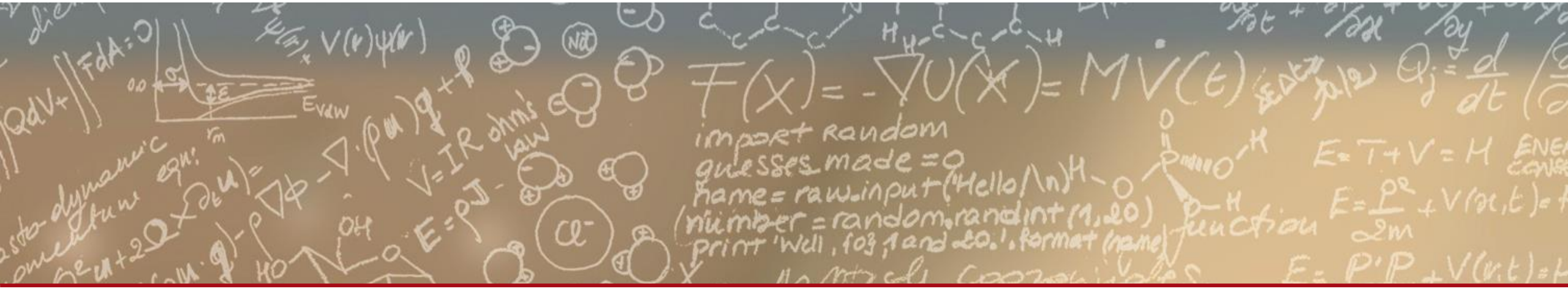
Uenv to test

One partition per uenv

Conclusions

Conclusions

- Programmable configuration gives full access to the Python language and libraries
- The final configuration can be generated on the fly based on the environment(s) and/or system(s) to be tested
- Generating a configuration based on the features of system(s)/environments(s) moves the responsibility on them to state what features they provide
- LABELS in container images can be used by ReFrame to generate a configuration at runtime and run the required tests
- By inspecting the features that the uenv stacks provide (e.g via a yaml file) ReFrame can test the corresponding functionality



Thank you for your attention

Questions?