High Performance Containers (HPC)

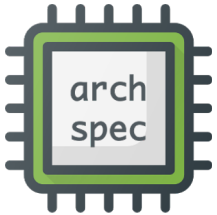
with

or

Spack

Todd Gamblin (LLNL)
tgamblin@llnl.gov

and



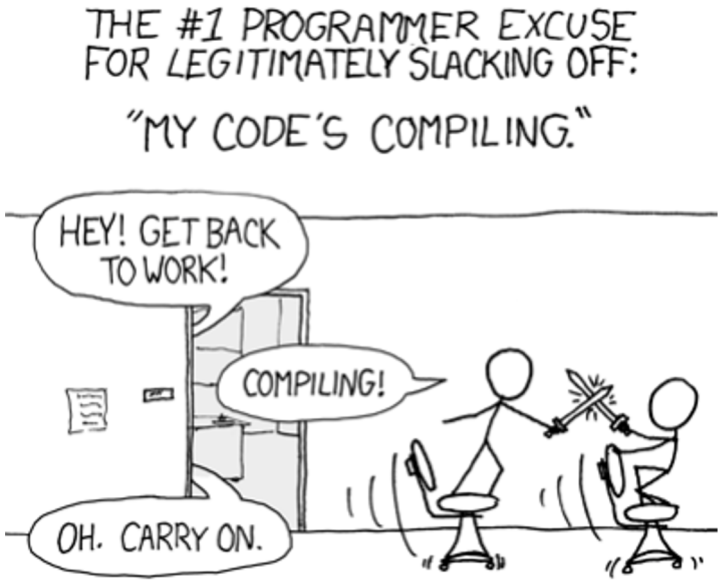
Kenneth Hoste (HPC-UGent)
kenneth.hoste@ugent.be

6th HPC Container Workshop (ISC'20)

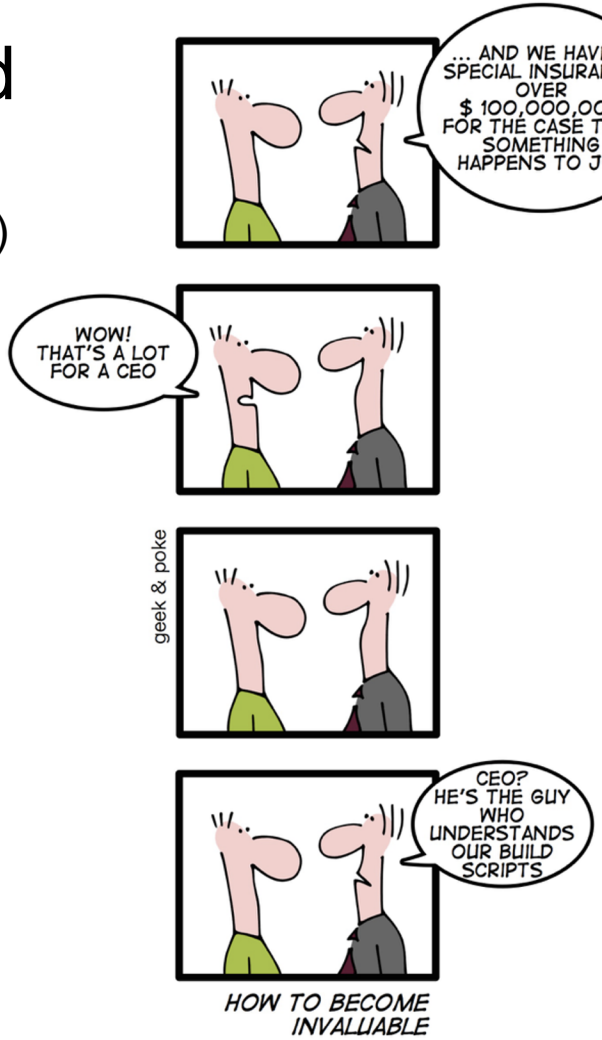
https://hpcw.github.io

Getting scientific software installed

- compilation from source (because of the ‘P’ in HPC)
- things get messy fast, and it’s getting worse...



https://xkcd.com/303



http://geek-and-poke.com/geekandpoke/2010/5/14/how-to-become-invaluable.html

github.com/easybuilders

easybuild.readthedocs.io

easybuild.slack.com
(join via easybuild-slack.herokuapp.com)

youtube.com/c/easybuilders

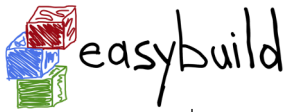
- framework for **installing scientific software on HPC systems**
- originally created by HPC team at Ghent University (Belgium) in 2009
- **stable since 2012**, frequent releases (every 6-8 weeks), extensively tested
- installations are **optimized for processor architecture of host** by default
- supports **almost 2000 different software packages** (excl. extensions),
  including LAMMPS, NWChem, OpenFOAM, PETSc, PyTorch, R, TensorFlow, Trilinos, …
  (see https://easybuild.rtfd.io/en/latest/version-specific/Supported_software.html)
- frequent hackathons & meetups, incl. 5 **EasyBuild User Meetings** (2016-2020)
- active mailing list and Slack channel (both with 250+ members)
- > 250 unique contributors (~100 yearly), **> 2,500 merged pull requests in 2019**!

Stacking software installations is trivial

- Availability of modules is sufficient to leverage existing installations (for example in Cray PE).
- Centrally installed software can be used as dependencies by end users who prefer maintaining their own software stack.
- Custom installations can shadow existing ones.

Easy customization through hooks & plugins

- Support for additional software & toolchains can be provided via `--include-*` options.
- Hooks are provided to customize the software installation procedures where required, without having to maintain patches.



GitHub integration

- create PRs from EasyBuild CLI

```
eb --new-pr example.eb
```

- use easyconfigs from PR, upload test report

```
eb --from-pr 12345 --upload-test-report
```

- PR preview: compare with existing easyconfigs

```
eb --preview-pr example.eb
```

Submitting software installations as jobs

```
$ eb example.eb -r --job
…
3 jobs submitted.
$ squeue
123   dep-one-1.0     RUNNING   node001
124   dep-two-2.0     RUNNING   node002
125   example-1.2.3   PENDING   (Dependency)
```

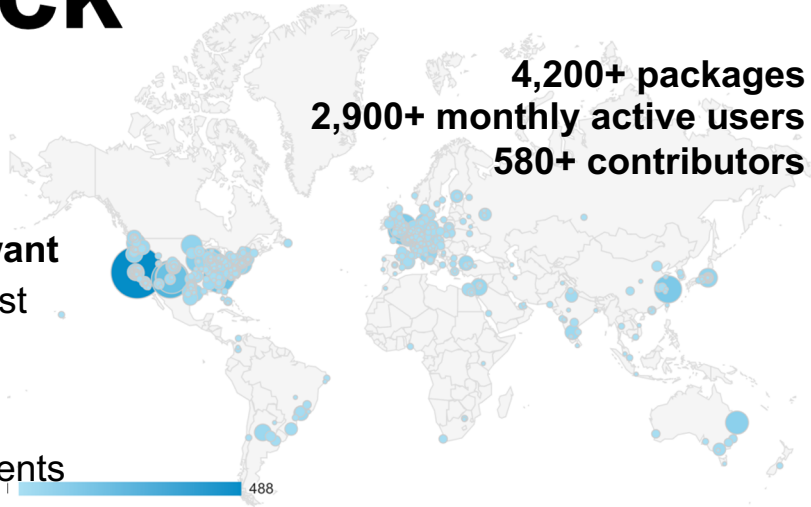
slurm

spack.io
github.com/spack
spack.readthedocs.io

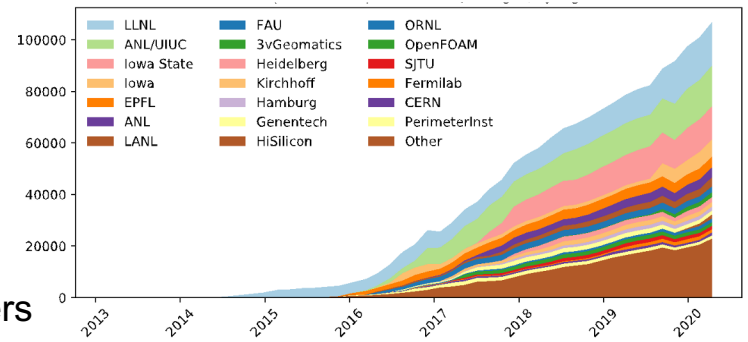
Spack

spackpm.slack.com
(join via spackpm.herokuapp.com)

4,200+ packages
2,900+ monthly active users
580+ contributors



- *Package manager* for HPC systems
  - Targets **facilities**, **developers**, and **end users**
  - Flexible versioning, **build the configuration you want**
  - Build with arbitrary versions, configs, etc. - all coexist
  - Inspired by Nix's model, Homebrew's UI, Python
- 3-4 stable releases/year
  - backports for bug fixes and performance improvements
- Package testing via U.S. ECP's E4S distribution
  - Recently began testing E4S builds on pull requests
- Over 4,200 packages (including Python, R, etc.)
- Very large and supportive community
  - Started at LLNL in 2013
  - Now used at hundreds of sites, 6 of top 10 supercomputers
  - 100 daily active Slack users, ~25 actively posting



Flexible command line interface

```
$ spack install mpileaks
$ spack install mpileaks@3.3
$ spack install mpileaks@3.3 %gcc@4.7.3
$ spack install mpileaks@3.3 %gcc@4.7.3 +threads
$ spack install mpileaks@3.3 cppflags="-O3 -g3"
$ spack install mpileaks@3.3 target=skylake
$ spack install mpileaks@3.3 ^mpich@3.2 %gcc@4.9.3
```

*Packages are **templates**: build them how you want.*

Environments support manifest/lockfile model

```
spack:
  specs:
    - lammps +mpi +cuda
    - openmpi@3.1.6 fabrics=ucx
    - hdf5@1.10.6
```

spack install

```
{
 "concrete_specs": {
  "6s63so2kstp3zyvjezglndmavy6l3nul": {
   "hdf5": {
       "version": "1.10.5",
       "arch": {
           "platform": "darwin",
           "platform_os": "mojave",
           "target": "x86_64"
       },
       "compiler": {
           "name": "clang",
           "version": "10.0
       },
```

`spack.yaml`
User/site requirements

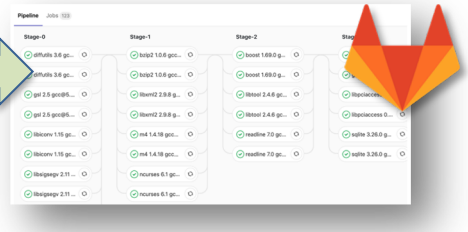
`spack.lock`
Rebuild exact versions, config

Use virtual envs anywhere: no modules required.

Spack

Powerful CI Integration

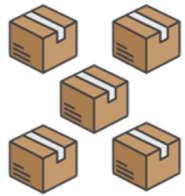
```
spack:
  specs:
    matrix:
      # 48 different builds + dependencies
      - [hdf5, lammps, gromacs, lulesh]
      - [%gcc, %clang]
      - [^mpich, ^openmpi]
      - [os=centos6, os=centos7]

  # build everything in OS containers
  gitlab-ci:
    - mappings:
      - centos6:
          match: os=centos6
          runner-attributes:
            image: spack/centos6
      - centos7:
          match: os=centos6
          runner-attributes:
            image: spack/centos7
```



`spack ci generate`

- Version environment in git repo
- Generate parallel build pipeline
- Trigger rebuilds on changes to env or spack (or both)

Relocatable Binary Packages

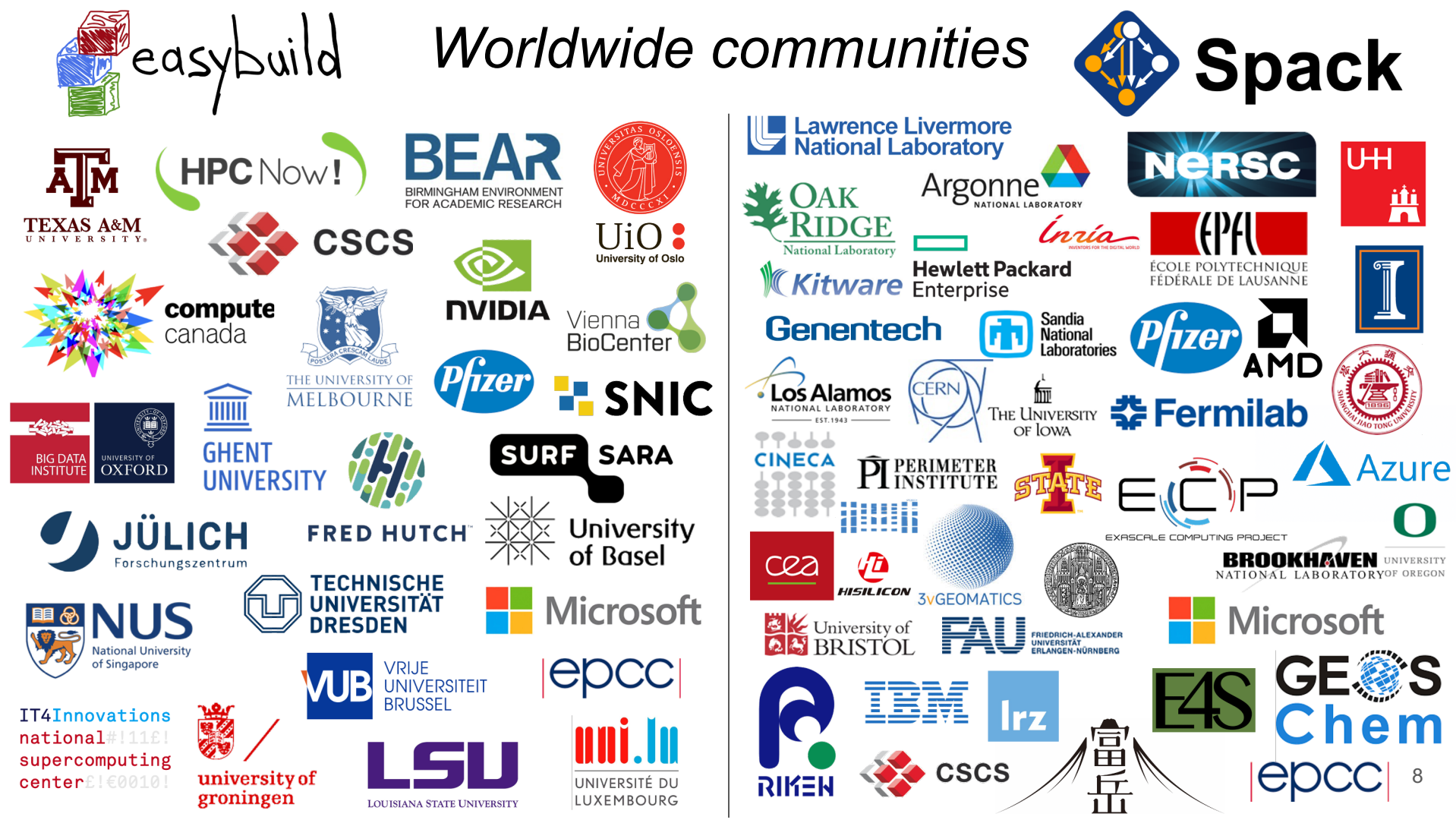


- Spack isn't just from-source builds
- Can install relocatable binary packages as well
  - Install trilinos in minutes
  - Greatly accelerate CI build pipelines by caching already-built packages
- E4S distro (https://e4s.io) offers binary packages you can try right now

easybuild vs Spack : similarities

- implemented in Python (compatible with both Python 2 & 3)
- open source, development on GitHub
- installing (scientific) software without admin privileges
- focus on HPC: compiling software from source, **performance** is key
- **generate environment modules** for HPC deployments
- similar high-level structure (framework + easyblocks, core + packages)
- highly configurable, well documented
- **worldwide communities**
- broad spectrum of supported software (1000s)
- **support for building containers!**

Worldwide communities



vs  Spack : key differences

license	GPLv2	MIT or Apache 2.0 (permissive)
OS support	Linux, Cray PE	Linux, macOS, Cray PE
command line interface	eb example-1.2.3.eb --robot	spack install example@1.2.3
packages	easyblocks + easyconfigs	templated packages
configuration	INI cfg files, env. vars, CLI options	YAML files, environments, CLI
main focus	HPC support teams, and end users	HPC support teams, software developers, devops, end users
dependency management	fixed versions	flexibility + lock files
access to installed software	environment modules	modules, environments, spack load
unique aspects & features	integration with resource managers (Slurm), GitHub integration, central easyconfigs repository, ...	concretization (dependency solver), binary packages, environments, CI integration (GitLab), uninstall

Building containers with Spack

1. Make an environment

```
spack:
  specs:
  - gromacs+mpi
  - mpich
```

2. Run a command

```
spack containerize
```

3. Build generated container recipe

```
# Build stage with Spack pre-installed and ready to be used
FROM spack/centos7:latest as builder

# What we want to install and how we want to install it
# is specified in a manifest file (spack.yaml)
RUN mkdir /opt/spack-environment \
&&  (echo "spack:" \
&&   echo "  specs:" \
&&   echo "  - gromacs+mpi" \
&&   echo "  - mpich" \
&&   echo "  concretization: together" \
&&   echo "  config:" \
&&   echo "    install_tree: /opt/software" \
&&   echo "  view: /opt/view") > /opt/spack-environment/spack.yaml

# Install the software, remove unecessary deps
RUN cd /opt/spack-environment && spack install && spack gc -y

# Strip all the binaries
RUN find -L /opt/view/* -type f -exec readlink -f '{}' \; | \
    xargs file -i | \
    grep 'charset=binary' | \
    grep 'x-executable\|x-archive\|x-sharedlib' | \
    awk -F: '{print $1}' | xargs strip -s

# Modifications to the environment that are necessary to run
RUN cd /opt/spack-environment && \
    spack env activate --sh -d . >> /etc/profile.d/z10_spack_environment.sh


# Bare OS image to run the installed executables
FROM centos:7

COPY --from=builder /opt/spack-environment /opt/spack-environment
COPY --from=builder /opt/software /opt/software
COPY --from=builder /opt/view /opt/view
COPY --from=builder /etc/profile.d/z10_spack_environment.sh /etc/profile.d/z10_spack_environ

RUN yum update -y && yum install -y epel-release && yum update -y
 && yum install -y libgomp \
 && rm -rf /var/cache/yum  && yum clean all

RUN echo 'export PS1="\[$(tput bold)\]\[$(tput setaf 1)\][gromacs]\[$(tput setaf 2)\]\u\[$(tput
```

- **Can generate a container image from any Spack environment**
  - Installs packages + dependencies
  - Container does NOT require modules
- **Generated builds are multi-stage (small images!)**
  - Final image is stripped
  - Contains only what's needed to run
  - Compilers, build deps, etc. are removed
- **Optionally customize:**
  - base image
  - labels
  - format (singularity/Dockerfile)
  - extra commands to run

```
container:
  # Select the format of the recipe e.g. docker,
  # singularity or anything else that is currently su
  format: docker

  # Select from a valid list of images
  base:
    image: "centos:7"
    spack: develop
```

Building containers with 

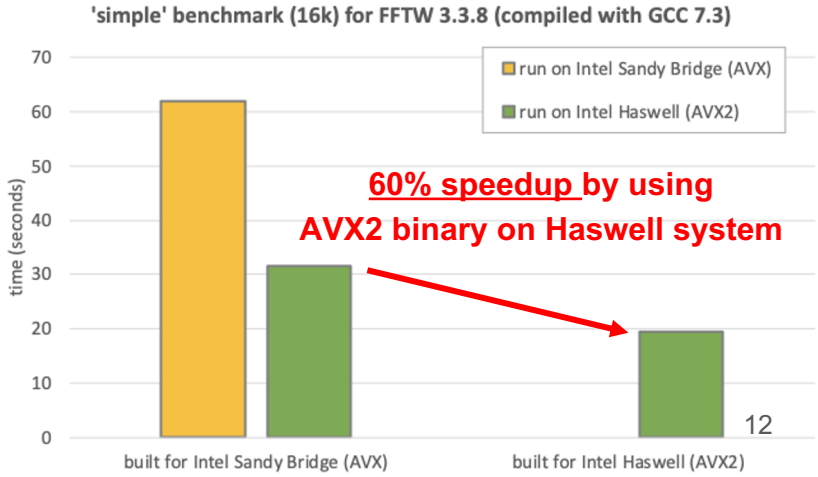
easybuild.readthedocs.io/en/latest/Containers.html

youtube.com/watch?v=FPqRDWbtv1M

- **experimental** support for both Docker & Singularity
- generate container recipe using a single command: `eb --containerize`
- recipe will use EasyBuild to install specified software and all missing dependencies
- use existing container image as a base, or start from scratch (bare OS)
- modules will be loaded automatically when starting container image
- currently focused on using RHEL-based OS in container
- … or you can just use EasyBuild as an easy way to install software in a container

Containers vs HPC

- container images are typically built with *generically optimized* binaries
  - to ensure they can always be run, regardless of processor architecture of the host
  - similar to packages installed by traditional package managers (yum/dnf, apt, …)
  - also applies to conda (to some extent)
- result: capabilities of the host processor may largely go unused
- impact on performance can be dramatic
- basically, **performance is *sacrificed*** to provide “mobility of compute”
- this performance aspect is largely ignored in the container community...



A library to reason about system architecture aspects

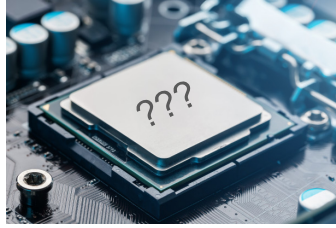
github.com/archspec

archspec.readthedocs.io

license: Apache 2.0 or MIT

- originally part of Spack, now a standalone library
- currently focused on CPU microarchitectures,
  intention is to extend the concept to network, GPUs, …
- JSON file with “database” of CPU microarchitectures (Intel, AMD, ARM, POWER)
- **fine-grained human-readable labels**, a.k.a. codenames (haswell, skylake, …)
-  Python library available to leverage this data: `pip install archspec`
- **query instruction set support & compatibility between microarchitectures, ...**
- cross-project collaboration between Kenneth Hoste (EasyBuild)
  and Todd Gamblin + Massimiliano Culpo (Spack)

What can archspec be used for?



- collect host info (via `/proc/cpuinfo` on Linux, `sysctl` command on macOS)
- determine codename of host CPU via archspec CLI ⟶

```
$ archspec cpu
skylake
```

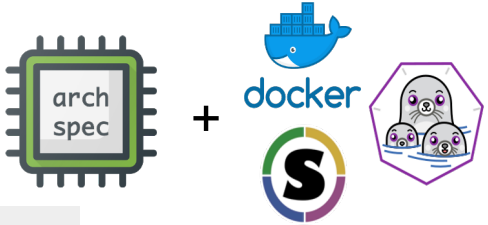
- check **compatibility** of host with given microarchitecture:

  “Can the current host run binaries that were compiled for Intel Haswell?”

- query **capabilities** of microarchitectures: *“Does Intel Haswell support AVX-512?”*
- query **compiler flags**: *“How do I compile for Intel Broadwell with icc?”*
- **partial ordering** of microarchitectures, picking best match from list of options:

  “Given AVX, AVX2 and AVX-512 binaries, what’s the best match on a Broadwell system?”

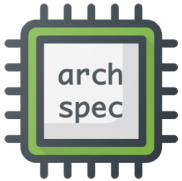
Enabling HPC (High Performance Containers)



Through `archspec` we can avoid having to choose between performance and mobility of compute!

- build **optimized** container images for a specific CPU microarchitecture
- **tag** container images with the CPU microarchitecture they were built for
- **check compatibility** with host from container and act on it

  (informative error message, warn about potential performance loss)
- picking **most suitable container image** for host w.r.t. CPU microarchitecture
- avoid need for using fat binaries (increases container size, requires rebuilds)



We need your help!

- We would like to see the HPC community to **standardize** the labels/codenames
- CPU vendors could contribute to the `archspec-json` repo
  - add feature info for new microarchitectures
  - review current data & submit corrections (if needed)
- Companies (Cray, NVIDIA, Red Hat, …) could adopt archspec & contribute to it
- Additional language bindings can be implemented (Go is already WIP)
- We're open to ideas & contributions to extend the concept to GPUs, network, ...
- **Reach out to us!**

Kenneth: kenneth.hoste@ugent.be @boegel @kehoste

Todd: tgamblin@llnl.gov @tgamblin