What’s new in Spack?

Easybuild User Meeting 2021

Todd Gamblin

Advanced Technology Office
Lawrence Livermore National Laboratory

Spack enables Software distribution for HPC

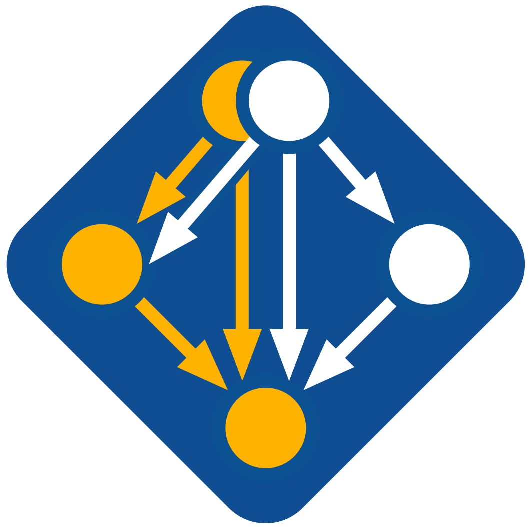
- Spack automates the build and installation of scientific software
- Packages are *parameterized,* so that users can easily tweak and tune configuration

No installation required: clone and go

```
$ git clone https://github.com/spack/spack
$ spack install hdf5
```

Simple syntax enables complex installs

```
$ spack install hdf5@1.10.5                    $ spack install hdf5@1.10.5 cppflags="-O3 –g3"
$ spack install hdf5@1.10.5 %clang@6.0         $ spack install hdf5@1.10.5 target=haswell
$ spack install hdf5@1.10.5 +threadssafe       $ spack install hdf5@1.10.5 +mpi ^mpich@3.2
```



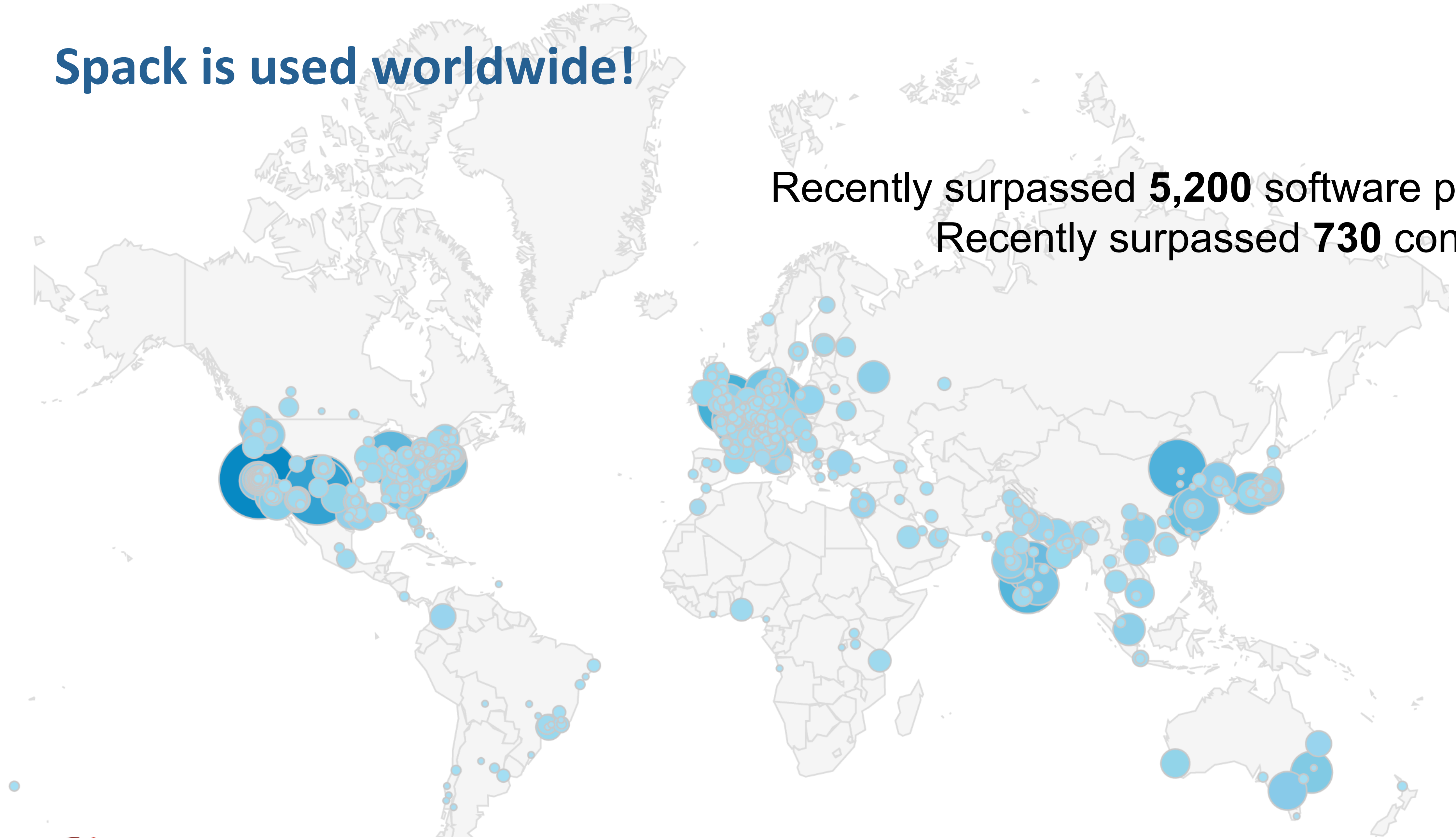
github.com/spack/spack

- Ease of use of mainstream tools, with flexibility needed for HPC
- In addition to CLI, Spack also:
  - Generates (but does **not** require) *modules*
  - Allows conda/virtualenv-like *environments*
  - Provides many devops features (CI, container generation, more)

EXASCALE COMPUTING PROJECT

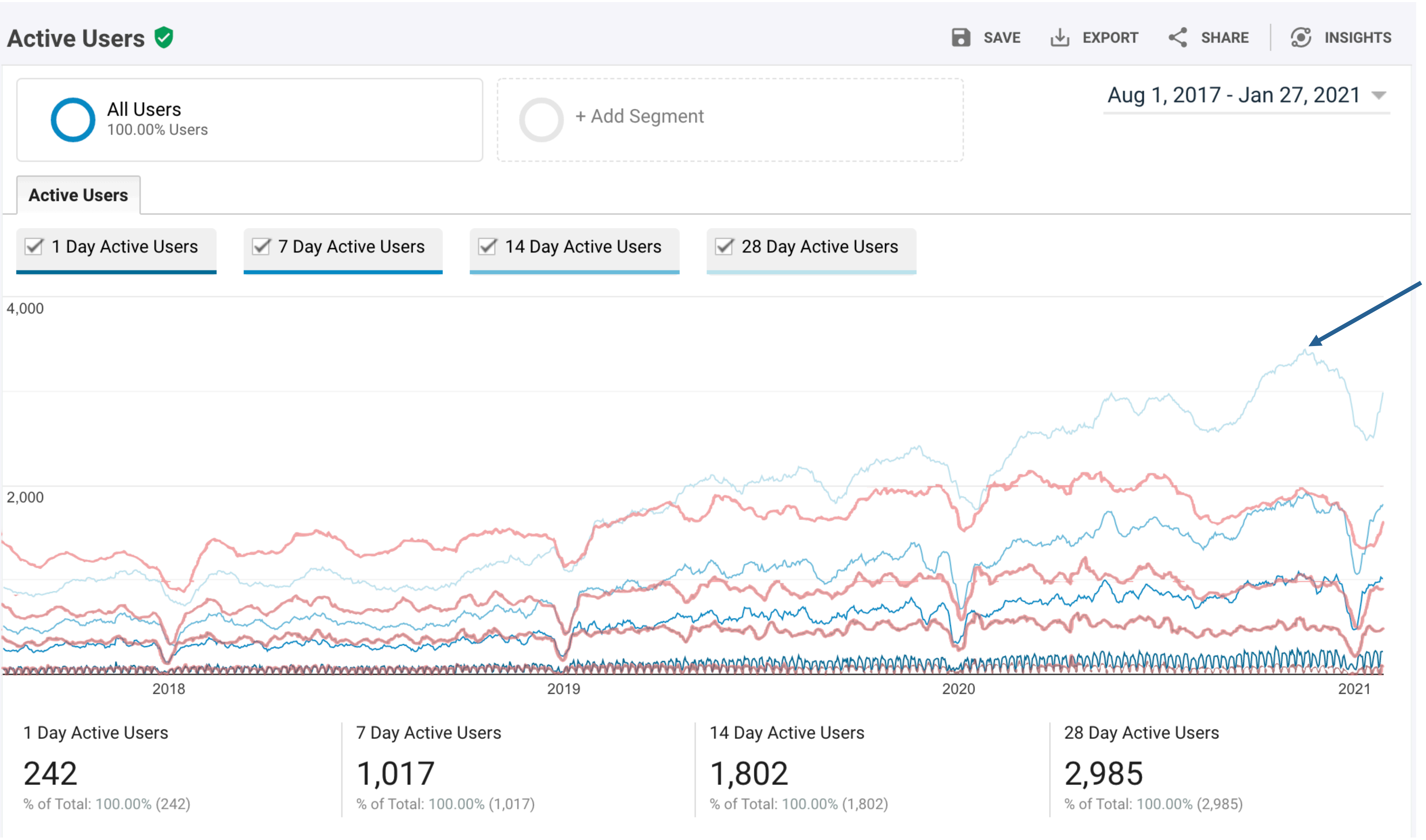
Spack is used worldwide!

Recently surpassed **5,200** software packages
Recently surpassed **730** contributors



EXASCALE COMPUTING PROJECT

The Spack community continues to grow steadily



Active Users

1 Day Active Users	7 Day Active Users	14 Day Active Users	28 Day Active Users
242	1,017	1,802	2,985
% of Total: 100.00% (242)	% of Total: 100.00% (1,017)	% of Total: 100.00% (1,802)	% of Total: 100.00% (2,985)

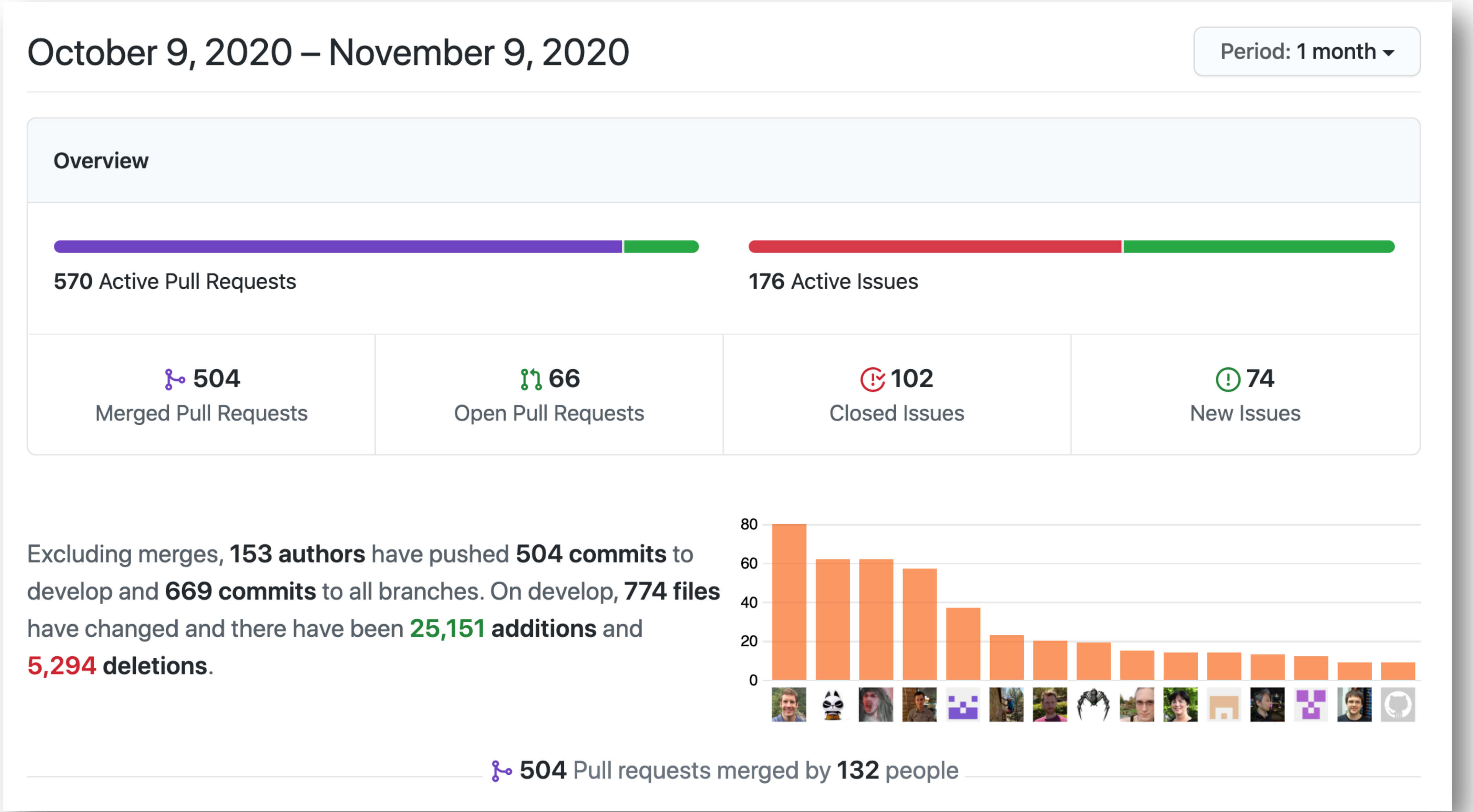
*as measured by active users on our readthedocs site

Broke 3,400 monthly active users this year

🟦 Spack

🟥 Easybuild

One month of Spack development is pretty busy!



October 9, 2020 – November 9, 2020

Period: 1 month

Overview

570 Active Pull Requests

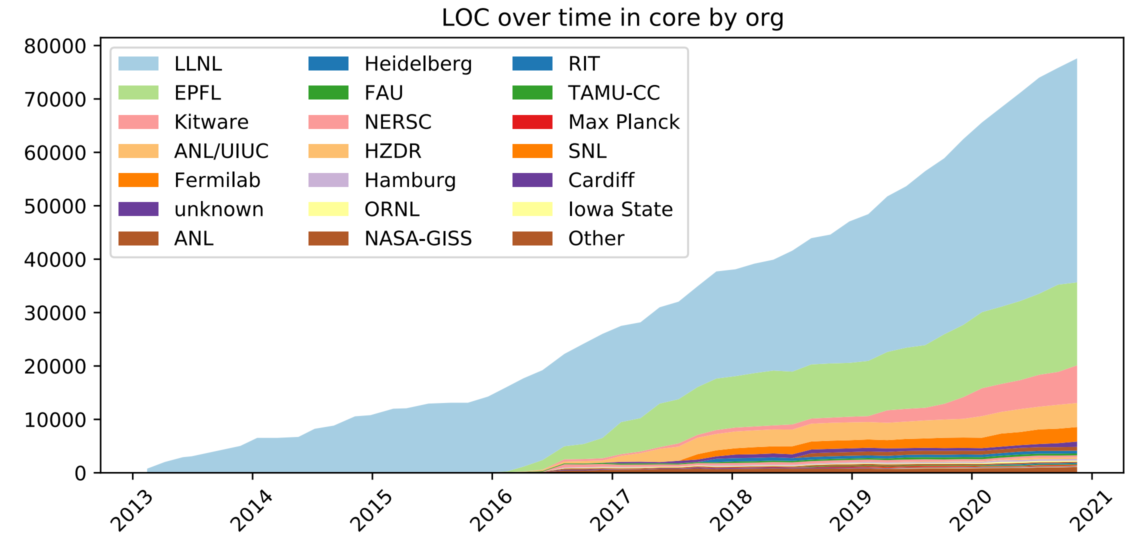
176 Active Issues

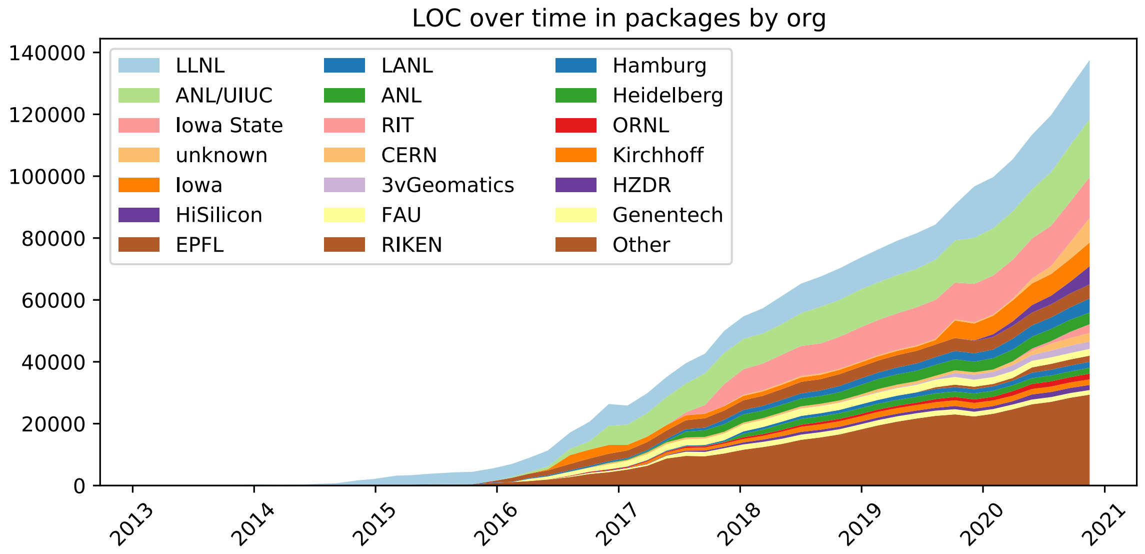
504 Merged Pull Requests	**66** Open Pull Requests	**102** Closed Issues	**74** New Issues

Excluding merges, **153 authors** have pushed **504 commits** to develop and **669 commits** to all branches. On develop, **774 files** have changed and there have been **25,151 additions** and **5,294 deletions**.

504 Pull requests merged by **132** people

Contributions to Spack continue to grow!

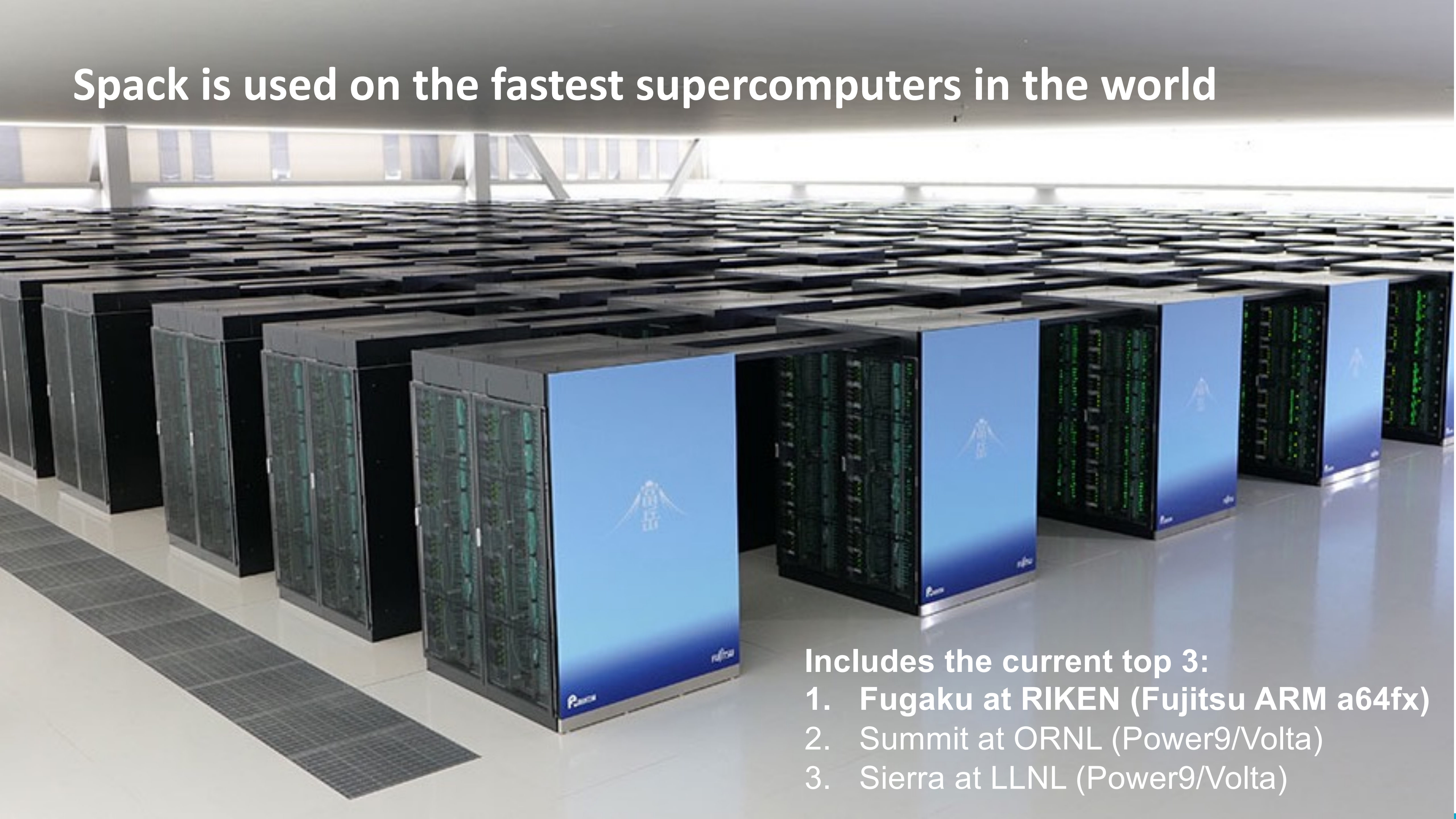




- In November 2015, LLNL provided most of the contributions to Spack
- Since then, we've gone from 300 to over 5,000 packages
- Most packages are from external contributors!
- Many contributions in core, as well.
- We are committed to sustaining Spack's open source ecosystem!

EXASCALE COMPUTING PROJECT

Spack is used on the fastest supercomputers in the world

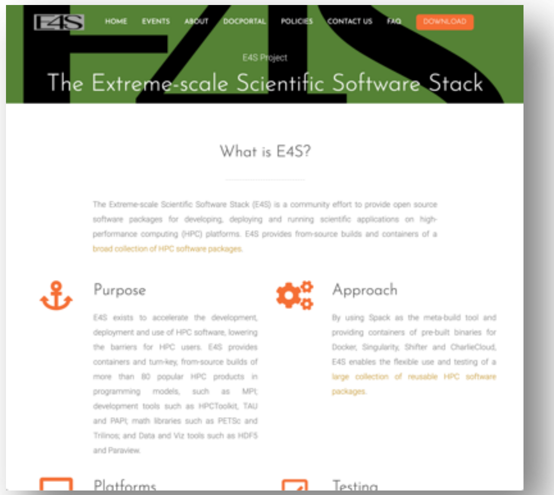


Includes the current top 3:

1. **Fugaku at RIKEN (Fujitsu ARM a64fx)**
2. Summit at ORNL (Power9/Volta)
3. Sierra at LLNL (Power9/Volta)

Spack is critical for ECP's mission to create a robust, capable exascale software ecosystem.

EXASCALE COMPUTING PROJECT



E4S Project

The Extreme-scale Scientific Software Stack

What is E4S?

The Extreme-scale Scientific Software Stack (E4S) is a community effort to provide open source software packages for developing, deploying and running scientific applications on high-performance computing (HPC) platforms. E4S provides from-source builds and containers of a broad collection of HPC software packages.

Purpose

E4S exists to accelerate the development, deployment and use of HPC software, lowering the barriers for HPC users. E4S provides containers and turn-key, from-source builds of more than 80 popular HPC products in programming models, such as MPI; development tools such as HPCToolkit, TAU and PAPI; math libraries such as PETSc and Trilinos; and Data and Viz tools such as HDF5 and Paraview.

Approach

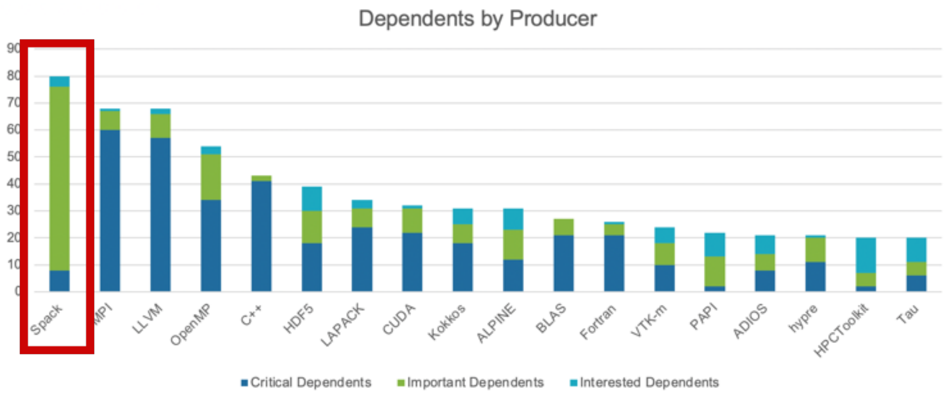
By using Spack as the meta-build tool and providing containers of pre-built binaries for Docker, Singularity, Shifter and CharlieCloud, E4S enables the flexible use and testing of a large collection of reusable HPC software packages.

Platforms

Testing

https://e4s.io

- Spack will be used to build software for the three upcoming U.S. exascale systems
- ECP has built the Extreme Scale Scientific Software Stack (E4S) with Spack – more at https://e4s.io
- Spack will be integral to upcoming ECP testing efforts.



Spack is the most depended-upon project in ECP

EXASCALE COMPUTING PROJECT

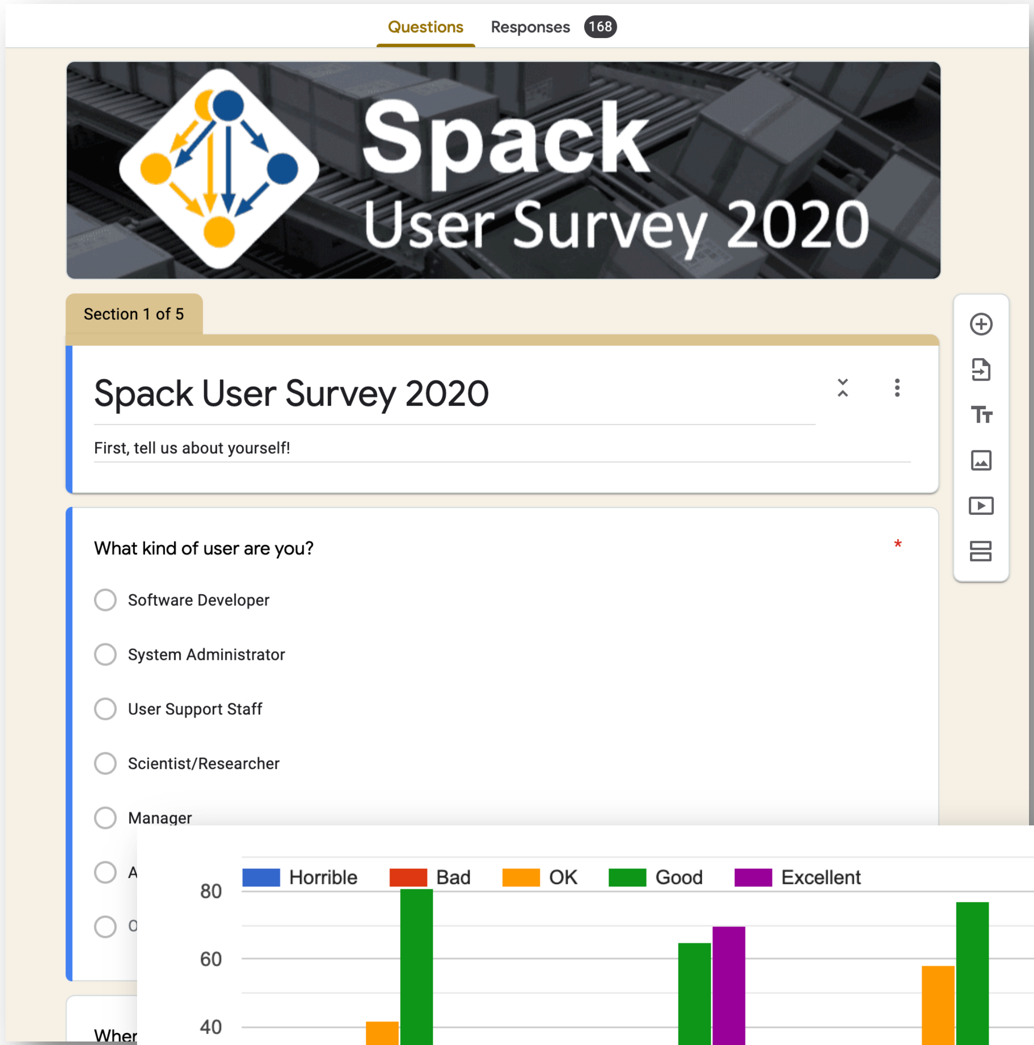
Spack User Survey 2020

- First widely distributed Spack Survey
  - Sent to all of Slack (900+ users)
  - All of Spack mailing list, ECP mailing list
- Got **169 responses!**
- **Takeaways:**
  - People like Spack and its community!
  - Docs and package stability need the most work
  - Concretizer features and dev features are the most wanted improvements

A writeup of the results is here:

https://spack.io/spack-user-survey-2020

Article also has links to the full survey data.



Questions Responses 168

Spack User Survey 2020

Section 1 of 5

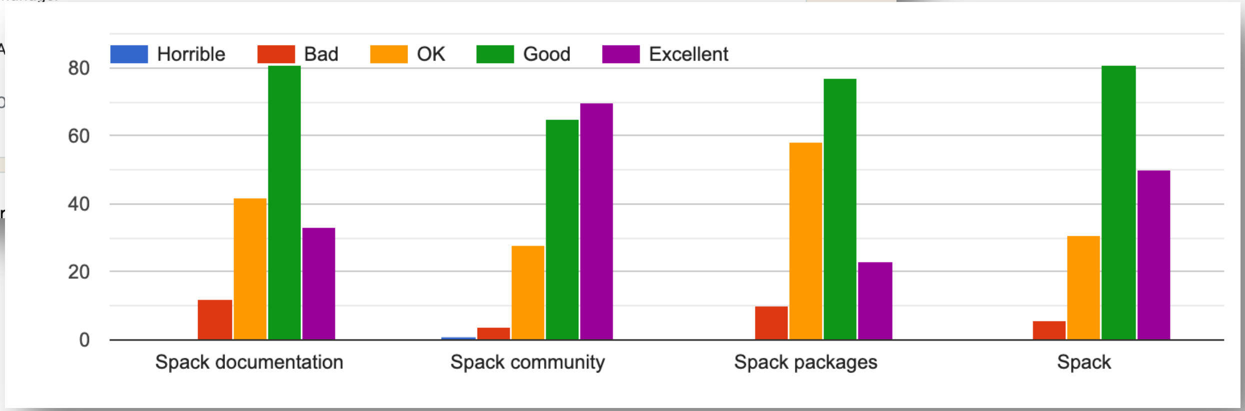
Spack User Survey 2020

First, tell us about yourself!

What kind of user are you? *

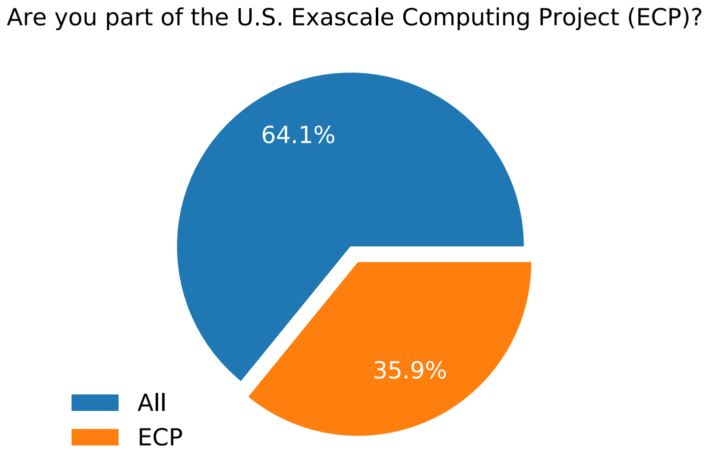
- Software Developer
- System Administrator
- User Support Staff
- Scientist/Researcher
- Manager
- A
- O

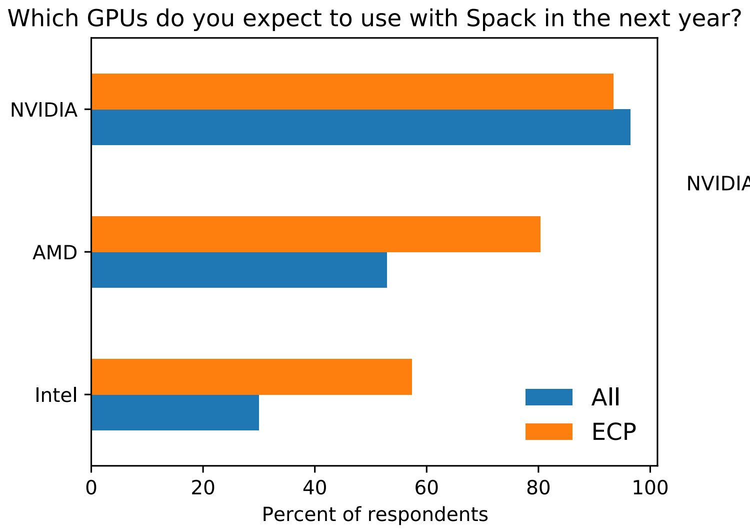
Wher

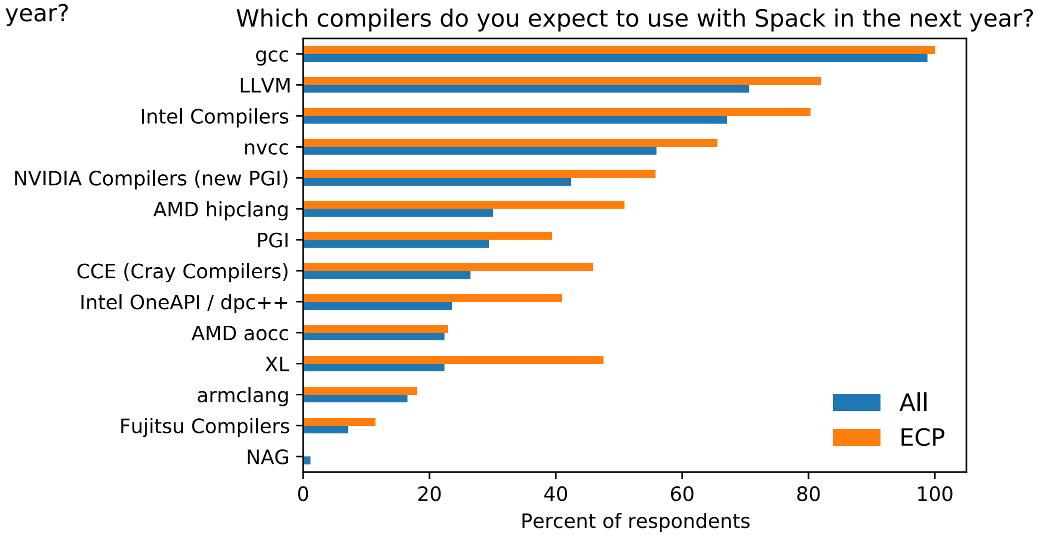


The Spack community is targeting a diverse range of GPUs and over 50% are targeting AMD

A detailed writeup of the results is at https://spack.io/spack-user-survey-2020/







Spack community is **~36% ECP**

GPU and compiler needs of ECP are more diverse than the broader Spack community.

We have seen an increase in industry contributions to Spack

- **Fujitsu and RIKEN** have contributed a **huge** number of packages for ARM/a64fx support on Fugaku
- **AMD** has contributed ROCm packages and compiler support
  - 55+ PRs mostly from AMD, also others
  - ROCm, HIP, aocc packages are all in Spack now
- **Intel** contributing oneapi support and compiler licenses for our build farm
- **NVIDIA** contributing NVHPC compiler support and other features
- **ARM** and **Linaro** members contributing ARM support
  - 400+ pull requests for ARM support from various companies
- **AWS** is collaborating with us on our build farm, making optimized binaries for ParallelCluster
  - Joint Spack tutorial in July with AWS had 125+ participants

Spack provides a *spec* syntax to describe customized installations

```
$ spack install mpileaks                                  unconstrained
$ spack install mpileaks@3.3                              @ custom version
$ spack install mpileaks@3.3 %gcc@4.7.3                   % custom compiler
$ spack install mpileaks@3.3 %gcc@4.7.3 +threads          +/- build option
$ spack install mpileaks@3.3 cppflags="-O3 –g3"           set compiler flags
$ spack install mpileaks@3.3 target=zen2                  set target microarchitecture
$ spack install mpileaks@3.3 ^mpich@3.2 %gcc@4.9.3        ^ dependency information
```

- Each expression is a ***spec*** for a particular configuration
  - Each clause adds a constraint to the spec
  - Constraints are optional – specify only what you need.
  - Customize install on the command line!
- Spec syntax is recursive
  - Full control over the combinatorial build space

Not shown: patches, **resources**, **conflicts**, other directives.

Spack packages are *templates*
They use a simple Python DSL to define how to build

```
from spack import *

class Kripke(CMakePackage):
    """Kripke is a simple, scalable, 3D Sn deterministic particle
       transport proxy/mini app.
    """

    homepage = "https://computation.llnl.gov/projects/co-design/kripke"
    url      = "https://computation.llnl.gov/projects/co-design/download/kripke-openmp-1.1.tar.gz"

    version('1.2.3', sha256='3f7f2eef0d1ba5825780d626741eb0b3f026a096048d7ec4794d2a7dfbe2b8a6')
    version('1.2.2', sha256='eaf9ddf562416974157b34d00c3a1c880fc5296fce2aa2efa039a86e0976f3a3')
    version('1.1', sha256='232d74072fc7b848fa2adc8a1bc839ae8fb5f96d50224186601f55554a25f64a')

    variant('mpi',    default=True, description='Build with MPI.')
    variant('openmp', default=True, description='Build with OpenMP enabled.')

    depends_on('mpi', when='+mpi')
    depends_on('cmake@3.0:', type='build')

    def cmake_args(self):
        return [
            '-DENABLE_OPENMP=%s' % ('+openmp' in self.spec),
            '-DENABLE_MPI=%s' % ('+mpi' in self.spec),
        ]

    def install(self, spec, prefix):
        # Kripke does not provide install target, so we have to copy
        # things into place.
        mkdirp(prefix.bin)
        install('../spack-build/kripke', prefix.bin)
```

Base package (CMake support)

Metadata at the class level

Versions

Variants (build options)

Dependencies (same spec syntax)

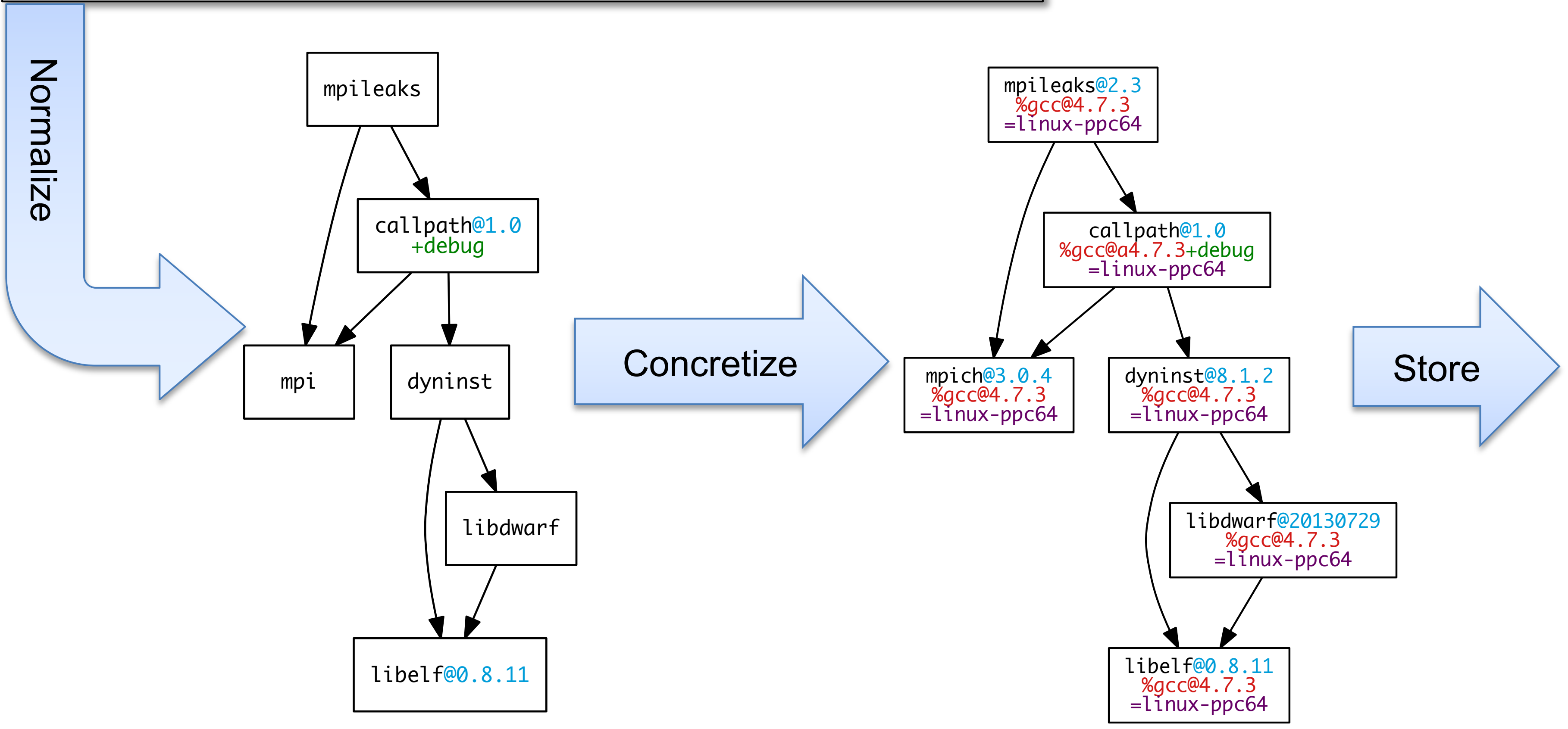
Install logic in instance methods

Don't typically need `install()` for `CMakePackage`, but we can work around codes that don't have it.

Concretization fills in missing configuration details when the user is not explicit.

```
mpileaks ^callpath@1.0+debug ^libelf@0.8.11
```

User input: *abstract* spec with some constraints



Normalize
mpileaks
callpath@1.0 +debug
mpi
dyninst
libdwarf
libelf@0.8.11
Concretize
mpileaks@2.3 %gcc@4.7.3 =linux-ppc64
callpath@1.0 %gcc@a4.7.3+debug =linux-ppc64
mpich@3.0.4 %gcc@4.7.3 =linux-ppc64
dyninst@8.1.2 %gcc@4.7.3 =linux-ppc64
libdwarf@20130729 %gcc@4.7.3 =linux-ppc64
libelf@0.8.11 %gcc@4.7.3 =linux-ppc64
Store

spec.yaml

```
spec:
- mpileaks:
    arch: linux-x86_64
    compiler:
      name: gcc
      version: 4.9.2
    dependencies:
      adept-utils: kszrtkpbzac3ss2ixcjkcorlaybnptp4
      callpath: bah5f4h4d2n47mgycej2mtrnrivvxy77
      mpich: aa4ar6ifj23yijqmdabeakpejcli72t3
    hash: 33hjjhxi7p6gyzn5ptgyes7sghyprujh
    variants: {}
    version: '1.0'
- adept-utils:
    arch: linux-x86_64
    compiler:
      name: gcc
      version: 4.9.2
    dependencies:
      boost: teesjv7ehpe5ksspjim5dk43a7qnowlq
      mpich: aa4ar6ifj23yijqmdabeakpejcli72t3
    hash: kszrtkpbzac3ss2ixcjkcorlaybnptp4
    variants: {}
    version: 1.0.1
- boost:
    arch: linux-x86_64
    compiler:
      name: gcc
      version: 4.9.2
    dependencies: {}
    hash: teesjv7ehpe5ksspjim5dk43a7qnowlq
    variants: {}
    version: 1.59.0
...
```

Abstract, normalized spec with some dependencies.

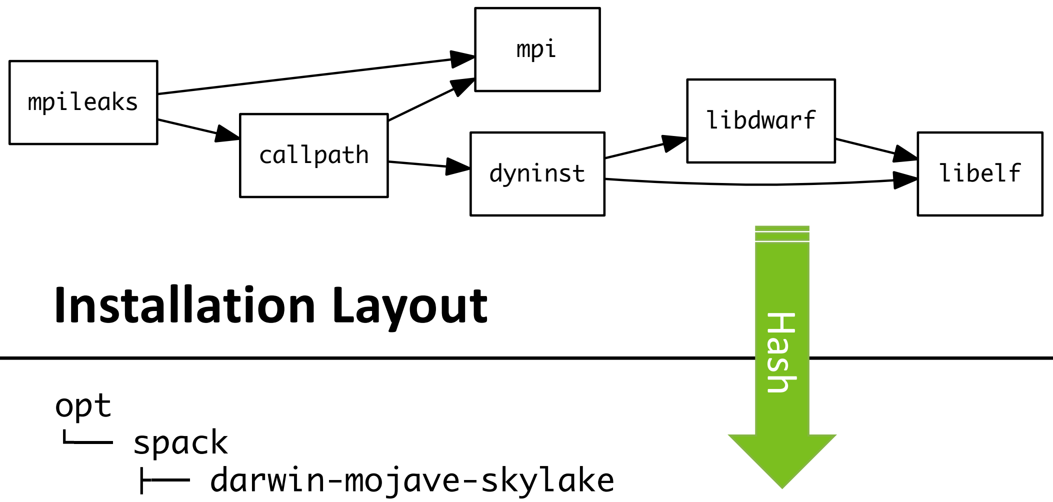
Concrete spec is fully constrained and can be passed to install.

Detailed provenance is stored with the installed package

EXASCALE COMPUTING PROJECT

Spack handles combinatorial software complexity

Dependency DAG

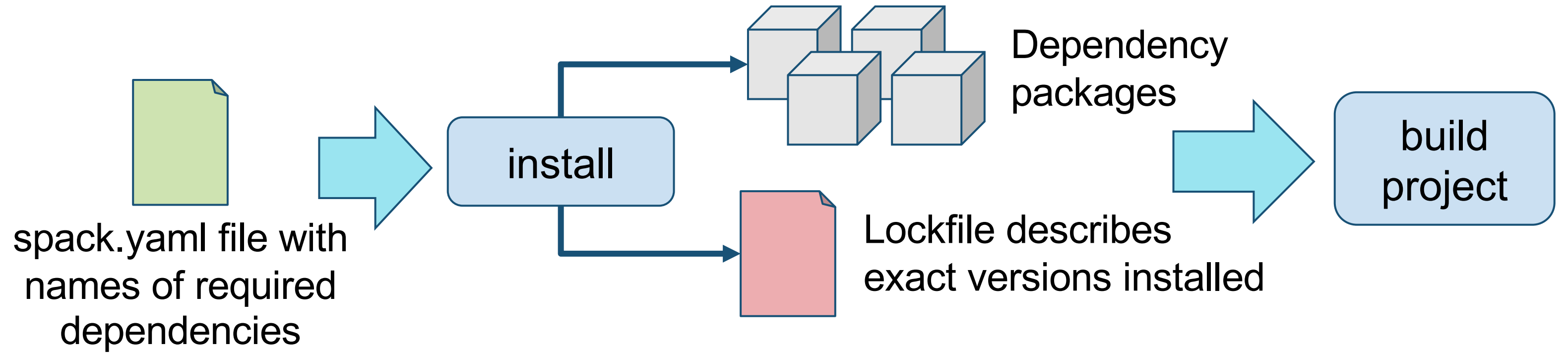


Installation Layout

```
opt
└── spack
    ├── darwin-mojave-skylake
    |   └── clang-10.0.0-apple
    |       ├── bzip2-1.0.8-hc4sm4vuzpm4znmvrfzri4ow2mkphe2e
    |       ├── python-3.7.6-daqqpssxb6qbfrztsezkmhus3xoflbsy
    |       ├── sqlite-3.30.1-u64v26igxvxyn23hysmklfums6tgjv5r
    |       ├── xz-5.2.4-u5eawkvaoc7vonabe6nndkcfwuv233cj
    |       └── zlib-1.2.11-x46q4wm46ay4pltriijbgizxjrhbaka6
    ├── darwin-mojave-x86_64
    |   └── clang-10.0.0-apple
    |       └── coreutils-8.29-pl2kcytejqcys5dzecfrtjqxfdssvnob
```

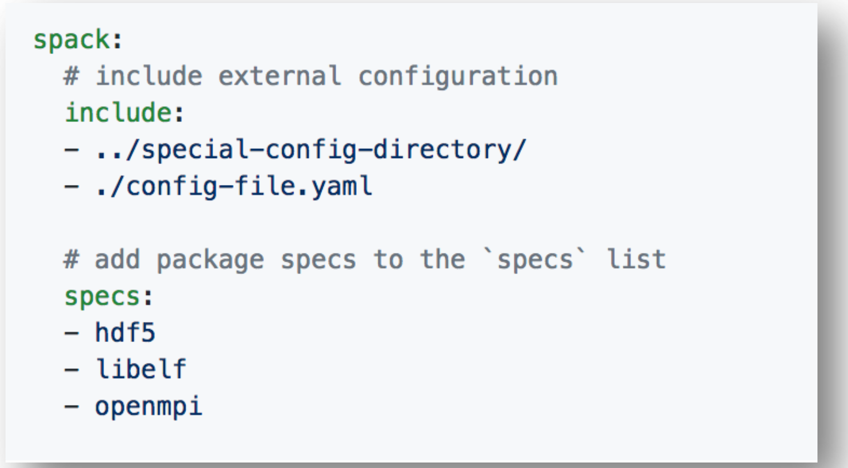
- Each unique dependency graph is a unique ***configuration***.
- Each configuration in a unique directory.
  - Multiple configurations of the same package can coexist.
- **Hash** of entire directed acyclic graph (DAG) is appended to each prefix.
- Installed packages automatically find dependencies
  - Spack embeds RPATHs in binaries.
  - No need to use modules or set LD_LIBRARY_PATH
  - Things work *the way you built them*

Spack environments enable users to build customized stacks from an abstract description


spack.yaml file with names of required dependencies
install
Dependency packages
Lockfile describes exact versions installed
build project

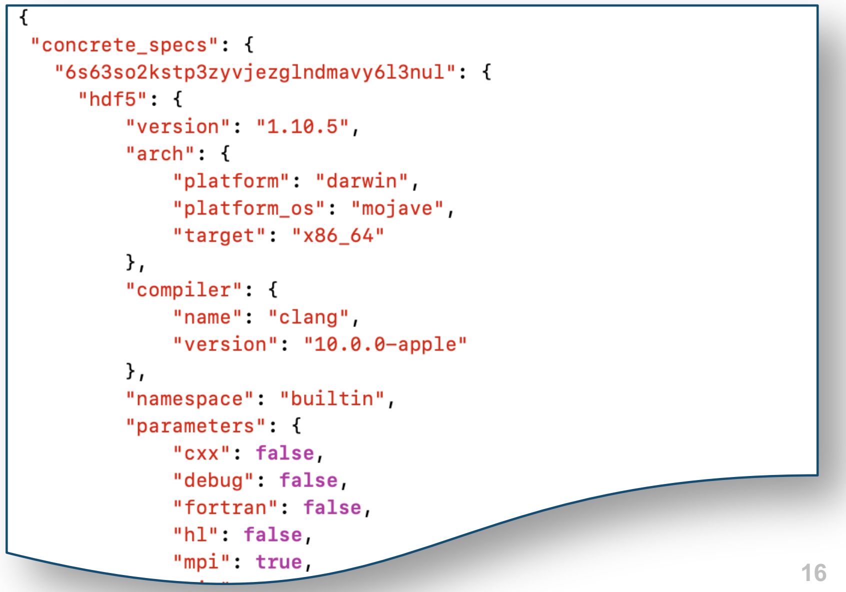
- spack.yaml describes project requirements
- spack.lock describes exactly what versions/configurations were installed, allows them to be reproduced.
- Can also be used to maintain configuration together with Spack packages.
  - E.g., versioning your own local software stack with consistent compilers/MPI implementations
  - Allows developers and site support engineers to easily version Spack configurations in a repository

Simple spack.yaml file



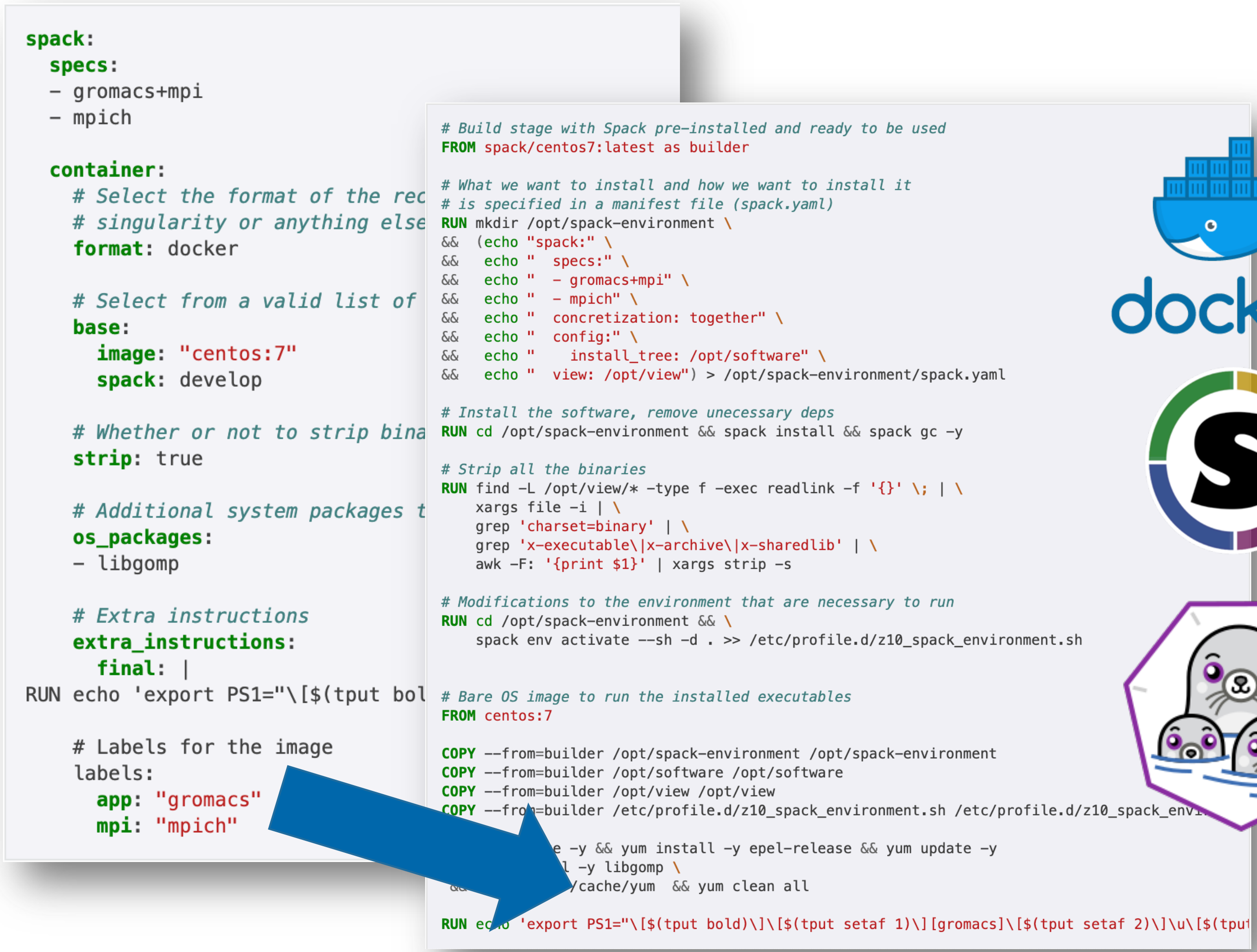
```
spack:
  # include external configuration
  include:
  - ../special-config-directory/
  - ./config-file.yaml

  # add package specs to the `specs` list
  specs:
  - hdf5
  - libelf
  - openmpi
```

Concrete spack.lock file (generated)



```
{
 "concrete_specs": {
   "6s63so2kstp3zyvjezglndmavy6l3nul": {
     "hdf5": {
         "version": "1.10.5",
         "arch": {
             "platform": "darwin",
             "platform_os": "mojave",
             "target": "x86_64"
         },
         "compiler": {
             "name": "clang",
             "version": "10.0.0-apple"
         },
         "namespace": "builtin",
         "parameters": {
             "cxx": false,
             "debug": false,
             "fortran": false,
             "hl": false,
             "mpi": true,
```

Spack can generate multi-stage container build recipes



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

  container:
    # Select the format of the rec
    # singularity or anything else
    format: docker

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

    # Whether or not to strip bina
    strip: true

    # Additional system packages t
    os_packages:
    - libgomp

    # Extra instructions
    extra_instructions:
      final: |
RUN echo 'export PS1="\[$(tput bol

    # Labels for the image
    labels:
      app: "gromacs"
      mpi: "mpich"
```

```
# 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_envi

          e -y && yum install -y epel-release && yum update -y
          l -y libgomp \
          /cache/yum  && yum clean all

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

- Any Spack environment can be bundled into a container image
  - Optional container section allows finer-grained customization

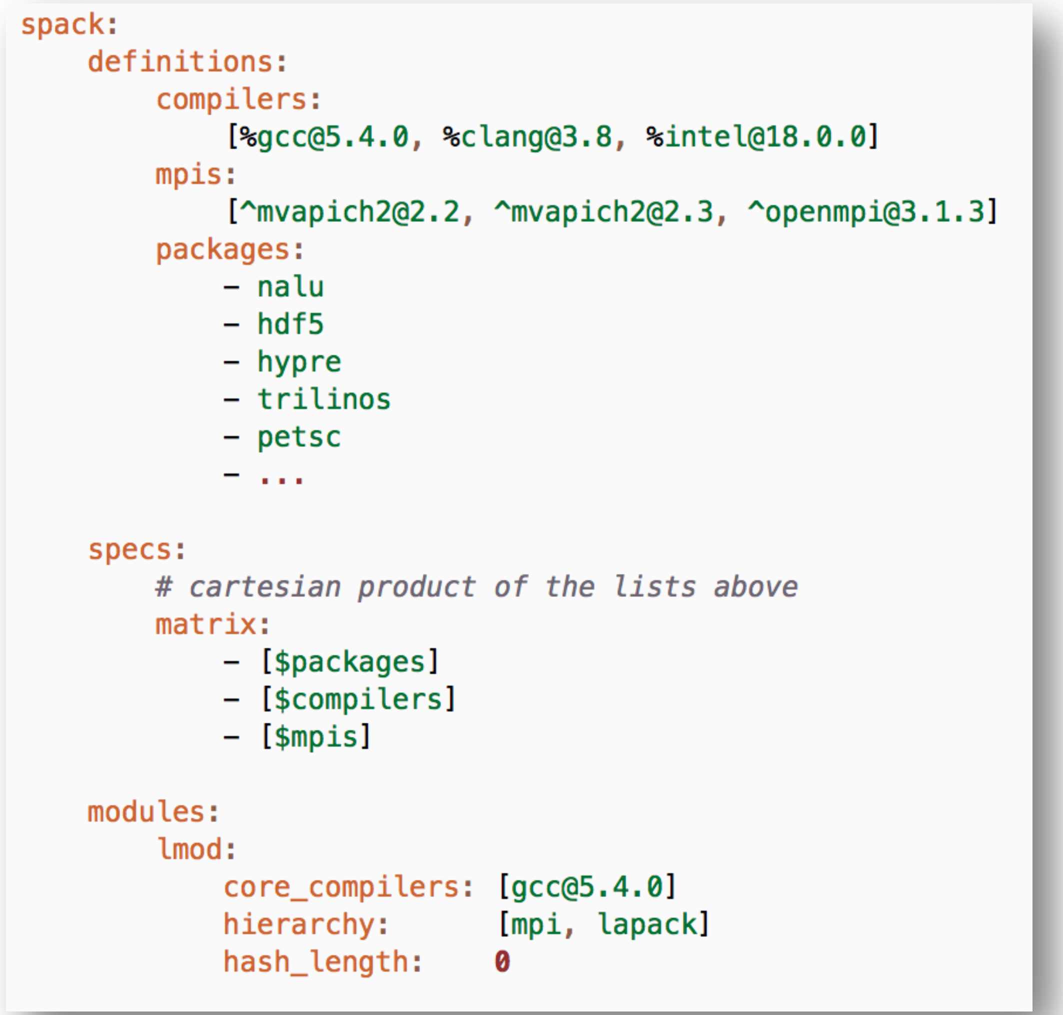
- Generated Dockerfile uses multi-stage builds to minimize size of final image
  - Strips binaries
  - Removes unneeded build deps with `spack gc`

- Can also generate Singularity recipes

- Originally included in Spack v0.14, updated for v0.16 to support arbitrary base images (OS distros)

`spack containerize`

Spack **stacks** are combinatorial environments for facility deployments

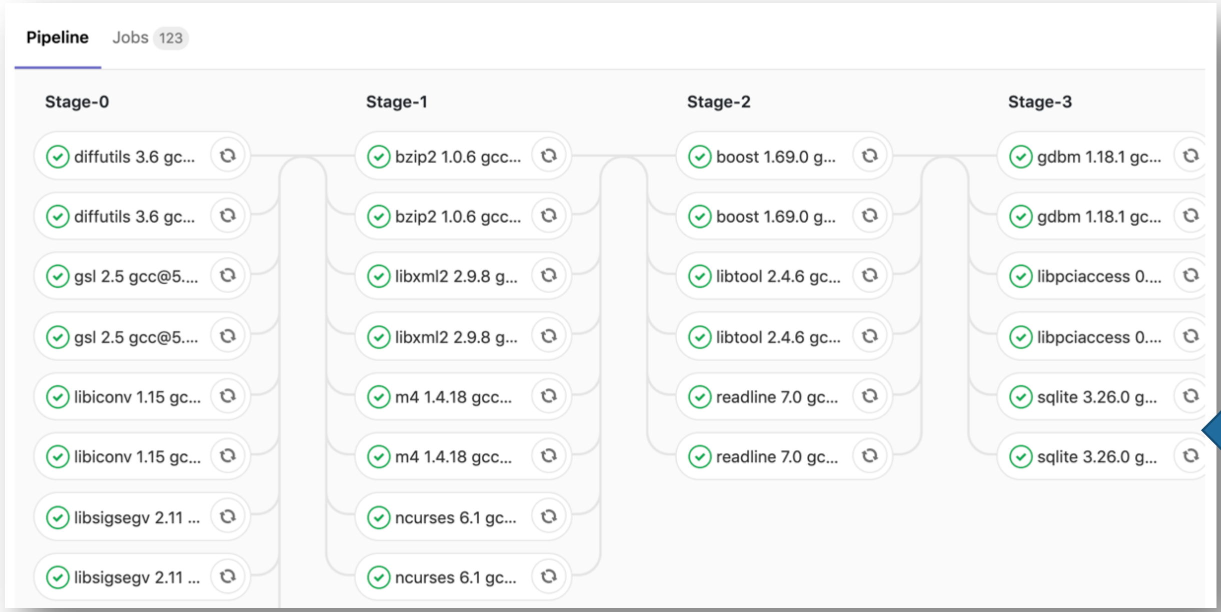


```
spack:
    definitions:
        compilers:
            [%gcc@5.4.0, %clang@3.8, %intel@18.0.0]
        mpis:
            [^mvapich2@2.2, ^mvapich2@2.3, ^openmpi@3.1.3]
        packages:
            - nalu
            - hdf5
            - hypre
            - trilinos
            - petsc
            - ...

    specs:
        # cartesian product of the lists above
        matrix:
            - [$packages]
            - [$compilers]
            - [$mpis]

    modules:
        lmod:
            core_compilers: [gcc@5.4.0]
            hierarchy:      [mpi, lapack]
            hash_length:    0
```

- Allow users to easily express large cross-products of builds
  - All the packages needed for a facility
  - Generate modules tailored to the site
  - Generate a directory layout to browse the packages
- Build on the environments workflow
  - Manifest + lockfile
  - Lockfile enables reproducibility
- Relocatable binaries allow the same binary to be used in a stack, regular install, or container build.
  - Difference is how the user interacts with the stack
  - Single-PATH stack vs. modules.

Spack has GitLab CI integration to automate package build pipelines

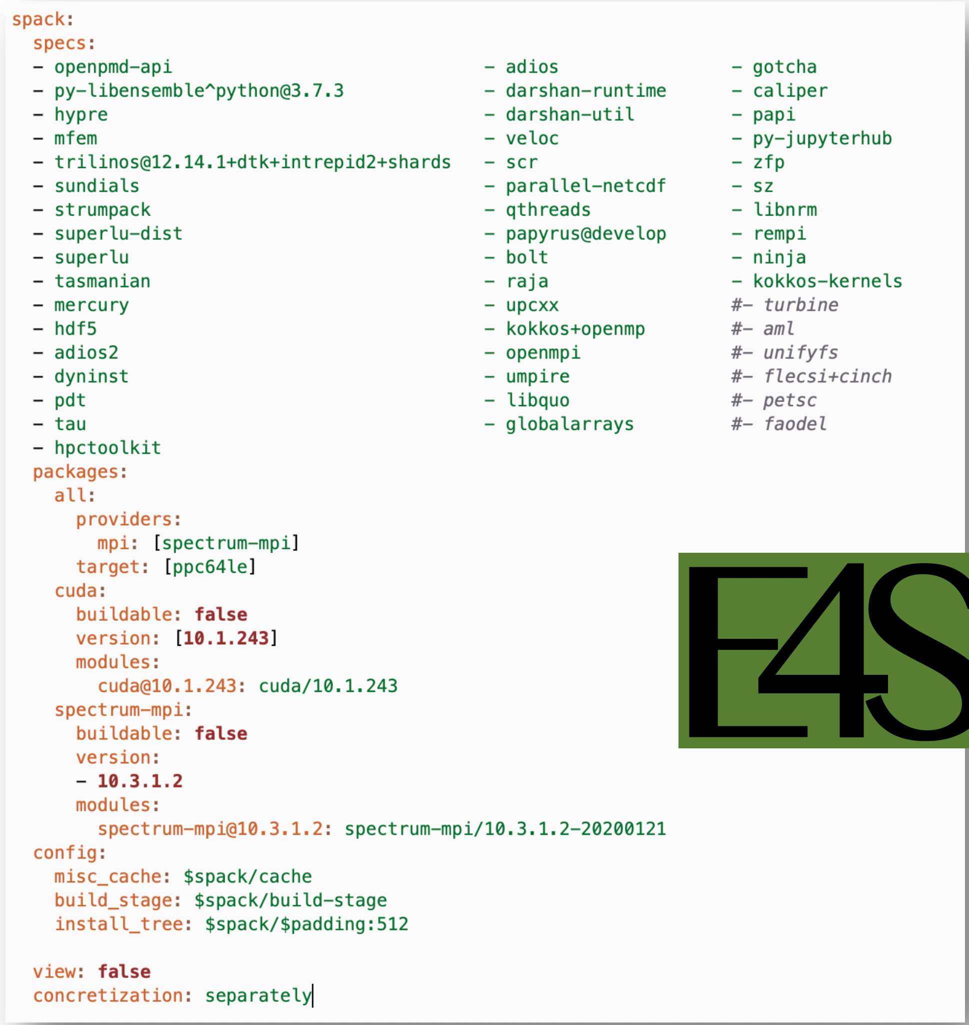
- Builds on Spack environments
  - Support auto-generating GitLab CI jobs
  - Can run in a Kube cluster or on bare metal runners at an HPC site
  - Sends progress to CDash



```yaml
spack:
  definitions:
  - pkgs:
    - readline@7.0
  - compilers:
    - '%gcc@5.5.0'
  - oses:
    - os=ubuntu18.04
    - os=centos7
  specs:
  - matrix:
    - [$pkgs]
    - [$compilers]
    - [$oses]
  mirrors:
    cloud_gitlab: https://mirror.spack.io
  gitlab-ci:
    mappings:
      - spack-cloud-ubuntu:
        match:
          - os=ubuntu18.04
        runner-attributes:
          tags:
            - spack-k8s
          image: spack/spack_builder_ubuntu_18.04
      - spack-cloud-centos:
        match:
          - os=centos7
        runner-attributes:
          tags:
            - spack-k8s
          image: spack/spack_builder_centos_7
  cdash:
    build-group: Release Testing
    url: https://cdash.spack.io
    project: Spack
    site: Spack AWS Gitlab Instance
```

E4S is ECP’s **curated**, Spack-based software distribution

- **E4S is just a set of Spack packages**
  - **60+ packages (297 including dependencies)**
  - **Growing to include all of ST and more**
- Users can install E4S packages:
  - In their home directory
  - In a container
- Facilities can install E4S packages:
  - On bare metal
  - In a container
- Users and facilities can choose parts they want
  - `spack install` only the packages you want
  - Or just edit the list of packages (and configurations) you want in a `spack.yaml` file



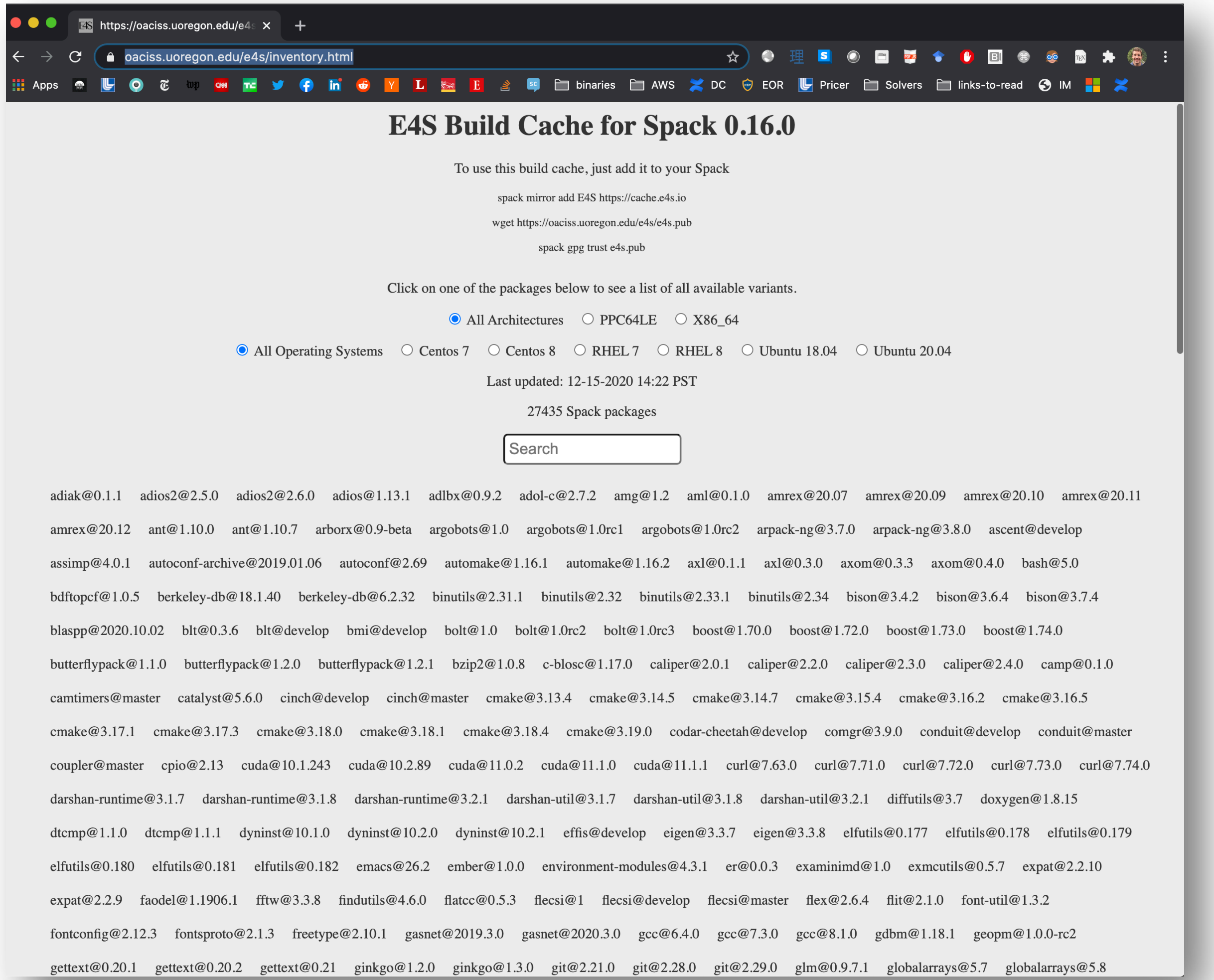
```yaml
spack:
  specs:
  - openpmd-api                             - adios           - gotcha
  - py-libensemble^python@3.7.3             - darshan-runtime - caliper
  - hypre                                   - darshan-util    - papi
  - mfem                                    - veloc           - py-jupyterhub
  - trilinos@12.14.1+dtk+intrepid2+shards   - scr             - zfp
  - sundials                                - parallel-netcdf - sz
  - strumpack                               - qthreads        - libnrm
  - superlu-dist                            - papyrus@develop - rempi
  - superlu                                 - bolt            - ninja
  - tasmanian                               - raja            - kokkos-kernels
  - mercury                                 - upcxx           #- turbine
  - hdf5                                    - kokkos+openmp   #- aml
  - adios2                                  - openmpi         #- unifyfs
  - dyninst                                 - umpire          #- flecsi+cinch
  - pdt                                     - libquo          #- petsc
  - tau                                     - globalarrays    #- faodel
  - hpctoolkit
  packages:
    all:
      providers:
        mpi: [spectrum-mpi]
      target: [ppc64le]
    cuda:
      buildable: false
      version: [10.1.243]
      modules:
        cuda@10.1.243: cuda/10.1.243
    spectrum-mpi:
      buildable: false
      version:
      - 10.3.1.2
      modules:
        spectrum-mpi@10.3.1.2: spectrum-mpi/10.3.1.2-20200121
  config:
    misc_cache: $spack/cache
    build_stage: $spack/build-stage
    install_tree: $spack/$padding:512

  view: false
  concretization: separately
```

Actual E4S manifest (`spack.yaml`) for OLCF Ascent

More on E4S at https://e4s.io

E4S team has built a binary cache with over 27,000 Spack binary packages

- Built for multiple OS's, architectures
- E4S team is working with ECP projects to accelerate their build pipelines
- Improved performance of cloud CI for one project by 10-100x
  - Previously, builds took too long for free cloud CI
  - Project can now iterate faster using Spack/E4S binaries
- We are rapidly building out binary build capabilities for Spack
  - Aim to have optimized binaries for most platforms in Frontier/El Capitan timeframe



https://oaciss.uoregon.edu/e4s/inventory.html

E4S Build Cache for Spack 0.16.0

To use this build cache, just add it to your Spack

```
spack mirror add E4S https://cache.e4s.io
wget https://oaciss.uoregon.edu/e4s/e4s.pub
spack gpg trust e4s.pub
```

Click on one of the packages below to see a list of all available variants.

[✓ All Architectures] PPC64LE X86_64

[✓ All Operating Systems] Centos 7 Centos 8 RHEL 7 RHEL 8 Ubuntu 18.04 Ubuntu 20.04

Last updated: 12-15-2020 14:22 PST

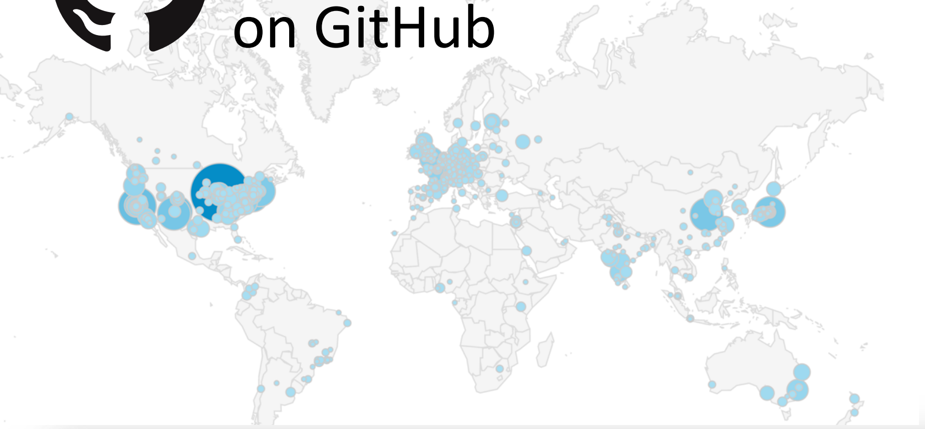
27435 Spack packages

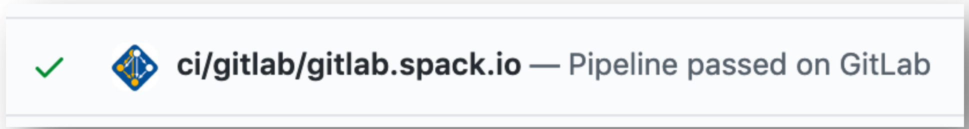
Search

adiak@0.1.1 adios2@2.5.0 adios2@2.6.0 adios@1.13.1 adlbx@0.9.2 adol-c@2.7.2 amg@1.2 aml@0.1.0 amrex@20.07 amrex@20.09 amrex@20.10 amrex@20.11
amrex@20.12 ant@1.10.0 ant@1.10.7 arborx@0.9-beta argobots@1.0 argobots@1.0rc1 argobots@1.0rc2 arpack-ng@3.7.0 arpack-ng@3.8.0 ascent@develop
assimp@4.0.1 autoconf-archive@2019.01.06 autoconf@2.69 automake@1.16.1 automake@1.16.2 axl@0.1.1 axl@0.3.0 axom@0.3.3 axom@0.4.0 bash@5.0
bdftopcf@1.0.5 berkeley-db@18.1.40 berkeley-db@6.2.32 binutils@2.31.1 binutils@2.32 binutils@2.33.1 binutils@2.34 bison@3.4.2 bison@3.6.4 bison@3.7.4
blaspp@2020.10.02 blt@0.3.6 blt@develop bmi@develop bolt@1.0 bolt@1.0rc2 bolt@1.0rc3 boost@1.70.0 boost@1.72.0 boost@1.73.0 boost@1.74.0
butterflypack@1.1.0 butterflypack@1.2.0 butterflypack@1.2.1 bzip2@1.0.8 c-blosc@1.17.0 caliper@2.0.1 caliper@2.2.0 caliper@2.3.0 caliper@2.4.0 camp@0.1.0
camtimers@master catalyst@5.6.0 cinch@develop cinch@master cmake@3.13.4 cmake@3.14.5 cmake@3.14.7 cmake@3.15.4 cmake@3.16.2 cmake@3.16.5
cmake@3.17.1 cmake@3.17.3 cmake@3.18.0 cmake@3.18.1 cmake@3.18.4 cmake@3.19.0 codar-cheetah@develop comgr@3.9.0 conduit@develop conduit@master
coupler@master cpio@2.13 cuda@10.1.243 cuda@10.2.89 cuda@11.0.2 cuda@11.1.0 cuda@11.1.1 curl@7.63.0 curl@7.71.0 curl@7.72.0 curl@7.73.0 curl@7.74.0
darshan-runtime@3.1.7 darshan-runtime@3.1.8 darshan-runtime@3.2.1 darshan-util@3.1.7 darshan-util@3.1.8 darshan-util@3.2.1 diffutils@3.7 doxygen@1.8.15
dtcmp@1.1.0 dtcmp@1.1.1 dyninst@10.1.0 dyninst@10.2.0 dyninst@10.2.1 effis@develop eigen@3.3.7 eigen@3.3.8 elfutils@0.177 elfutils@0.178 elfutils@0.179
elfutils@0.180 elfutils@0.181 elfutils@0.182 emacs@26.2 ember@1.0.0 environment-modules@4.3.1 er@0.0.3 examinimd@1.0 exmcutils@0.5.7 expat@2.2.10
expat@2.2.9 faodel@1.1906.1 fftw@3.3.8 findutils@4.6.0 flatcc@0.5.3 flecsi@1 flecsi@develop flecsi@master flex@2.6.4 flit@2.1.0 font-util@1.3.2
fontconfig@2.12.3 fontsproto@2.1.3 freetype@2.10.1 gasnet@2019.3.0 gasnet@2020.3.0 gcc@6.4.0 gcc@7.3.0 gcc@8.1.0 gdbm@1.18.1 geopm@1.0.0-rc2
gettext@0.20.1 gettext@0.20.2 gettext@0.21 ginkgo@1.2.0 ginkgo@1.3.0 git@2.21.0 git@2.28.0 git@2.29.0 glm@0.9.7.1 globalarrays@5.7 globalarrays@5.8

We are expanding our CI builds to include every pull request!

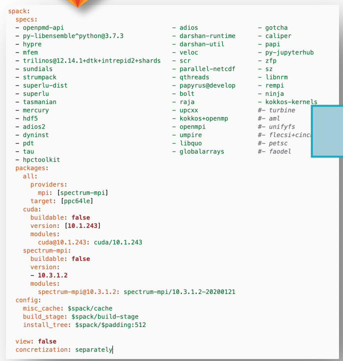
Spack Contributions on GitHub





✓ ci/gitlab/gitlab.spack.io — Pipeline passed on GitLab

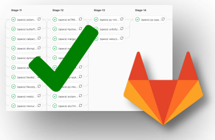
gitlab.spack.io



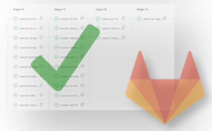
```
spack:
  specs:
  - openpmd-api                              - adios             - gotcha
  - py-libensemble^python@3.7.3              - darshan-runtime   - caliper
  - hypre                                    - darshan-util      - papi
  - mfem                                     - veloc             - py-jupyterhub
  - trilinos@12.14.1+dtk+intrepid2+shards    - scr               - zfp
  - sundials                                 - parallel-netcdf   - sz
  - strumpack                                - qthreads          - libnrm
  - superlu-dist                             - papyrus@develop   - rempi
  - superlu                                  - bolt              - ninja
  - tasmanian                                - raja              - kokkos-kernels
  - mercury                                  - upcxx             #- turbine
  - hdf5                                     - kokkos+openmp     #- aml
  - adios2                                   - openmpi           #- unifyfs
  - dyninst                                  - umpire            #- flecsi+cinc
  - pdt                                      - libquo            #- petsc
  - tau                                      - globalarrays      #- faodel
  - hpctoolkit
  packages:
    all:
      providers:
        mpi: [spectrum-mpi]
      target: [ppc64le]
    cuda:
      buildable: false
      version: [10.1.243]
      modules:
        cuda@10.1.243: cuda/10.1.243
    spectrum-mpi:
      buildable: false
      version:
      - 10.3.1.2
      modules:
        spectrum-mpi@10.3.1.2: spectrum-mpi/10.3.1.2-20200121
  config:
    misc_cache: $spack/cache
    build_stage: $spack/build-stage
    install_tree: $spack/$padding:512

  view: false
  concretization: separately
```

E4S `spack.yaml` configuration

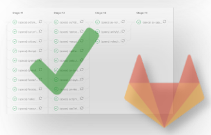
`spack ci`



Pipelines running in **AWS**



Pipelines at **LLNL**
(in progress)



Pipelines at **U. Oregon**
(in progress)

GitLab CI builds (changed) packages
- On every pull request
- On every release branch
- Different compilers (Intel soon!)

- **New security support contributions from forks**
  - Sandboxed build caches for test builds
  - Authoritative builds on mainline only after approved merge

EXASCALE COMPUTING PROJECT

Kitware aws

Spack v0.16.0 was released in November

Major new features:

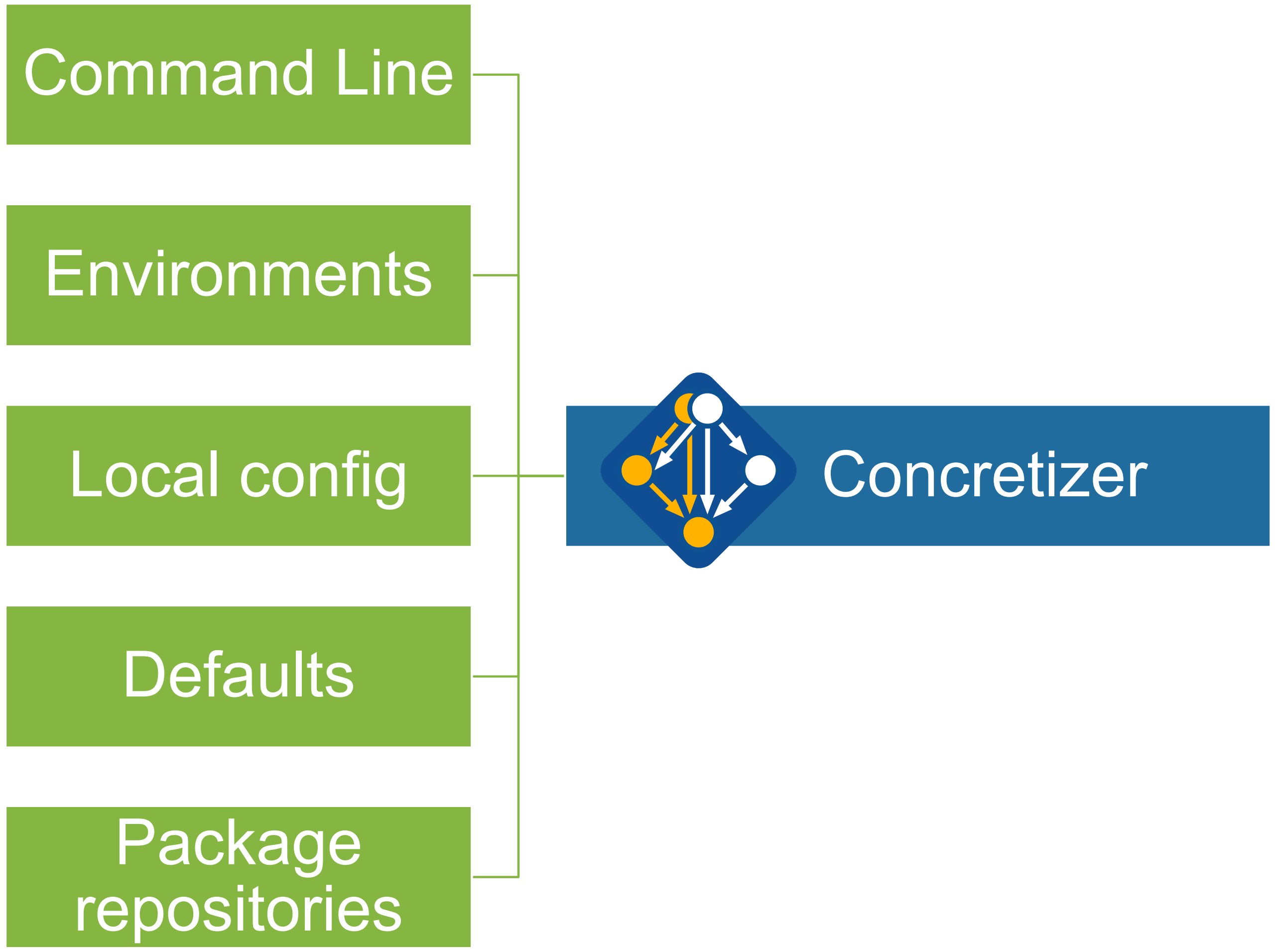
1. **New Concretizer (experimental)**
2. `spack test` **(experimental)**
3. `spack develop`
4. **Parallel environment builds**
5. **Custom base images for** `spack containerize`
6. `spack external find` **support**
   - now finds 15 common packages (including perl, MPI, others)
7. **Support for aocc, nvhpc, and oneapi compilers**

- **5,050** packages (Over **1,500** added since 0.13.1 a year ago)
- **Full release notes:** https://github.com/spack/spack/releases/tag/v0.16.0

EXASCALE COMPUTING PROJECT

Spack’s concretizer has gotten pretty complicated

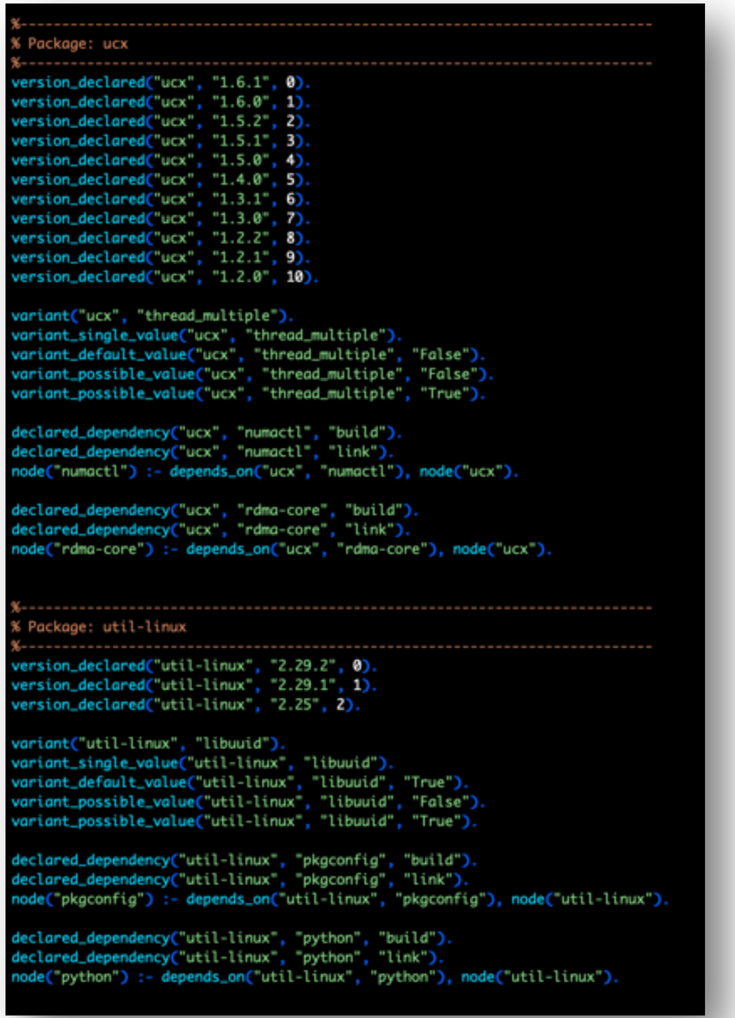
Sources for constraints



- Current implementation is ad-hoc:
  - Traverse the DAG
  - Evaluate conditions, add dependencies
  - Fill in defaults from many sources
  - Repeat until DAG doesn’t change
- Issues:
  - Limited support for backtracking causes some graphs to resolve incorrectly
  - Some constraints are strictly ordered
  - Lots of conditional complexity
- Design doesn’t scale to all the criteria
  - Hard to add new features/logic
  - Can be slow

The new concretizer is finally here!

- New concretizer leverages Clingo (see potassco.org)
- Clingo is an Answer Set Programming (ASP) solver
  - ASP looks like Prolog; leverages SAT solvers for speed/correctness
  - ASP program has 2 parts:
    1. Large list of facts generated from our package repositories and config
       - 20,000 – 30,000 facts is typical – includes dependencies, options, etc.
    2. Small logic program (~700 lines), including constraints and optimization criteria
- New algorithm on the Spack side is conceptually simpler:
  - Generate facts for all possible dependencies, send to logic program
  - Optimization criteria express preferences more clearly
  - Build a DAG from the results
- New concretizer solves many specs that current concretizer can't
  - Backtracking is a huge win – many issues resolved
  - Currently requires user to install clingo with Spack
  - Solver will be automatically installed from public binaries in 0.17.0



```
%----------------------------------------------------------------------------
% Package: ucx
%----------------------------------------------------------------------------
version_declared("ucx", "1.6.1", 0).
version_declared("ucx", "1.6.0", 1).
version_declared("ucx", "1.5.2", 2).
version_declared("ucx", "1.5.1", 3).
version_declared("ucx", "1.5.0", 4).
version_declared("ucx", "1.4.0", 5).
version_declared("ucx", "1.3.1", 6).
version_declared("ucx", "1.3.0", 7).
version_declared("ucx", "1.2.2", 8).
version_declared("ucx", "1.2.1", 9).
version_declared("ucx", "1.2.0", 10).

variant("ucx", "thread_multiple").
variant_single_value("ucx", "thread_multiple").
variant_default_value("ucx", "thread_multiple", "False").
variant_possible_value("ucx", "thread_multiple", "False").
variant_possible_value("ucx", "thread_multiple", "True").

declared_dependency("ucx", "numactl", "build").
declared_dependency("ucx", "numactl", "link").
node("numactl") :- depends_on("ucx", "numactl"), node("ucx").

declared_dependency("ucx", "rdma-core", "build").
declared_dependency("ucx", "rdma-core", "link").
node("rdma-core") :- depends_on("ucx", "rdma-core"), node("ucx").


%----------------------------------------------------------------------------
% Package: util-linux
%----------------------------------------------------------------------------
version_declared("util-linux", "2.29.2", 0).
version_declared("util-linux", "2.29.1", 1).
version_declared("util-linux", "2.25", 2).

variant("util-linux", "libuuid").
variant_single_value("util-linux", "libuuid").
variant_default_value("util-linux", "libuuid", "True").
variant_possible_value("util-linux", "libuuid", "False").
variant_possible_value("util-linux", "libuuid", "True").

declared_dependency("util-linux", "pkgconfig", "build").
declared_dependency("util-linux", "pkgconfig", "link").
node("pkgconfig") :- depends_on("util-linux", "pkgconfig"), node("util-linux").

declared_dependency("util-linux", "python", "build").
declared_dependency("util-linux", "python", "link").
node("python") :- depends_on("util-linux", "python"), node("util-linux").
```

Some facts for the HDF5 package

EXASCALE COMPUTING PROJECT

spack test: write tests directly in Spack packages, so that they can evolve with the software

```
class Libsigsegv(AutotoolsPackage, GNUMirrorPackage):
    """GNU libsigsegv is a library for handling page faults in user mode."""

    # ... spack package contents ...

    extra_install_tests = ‘tests/.libs’

    def test(self):
        data_dir = self.test_suite.current_test_data_dir
        smoke_test_c = data_dir.join(‘smoke_test.c’)

        self.run_test(
            'cc’, [
                '-I%s' % self.prefix.include,
                '-L%s' % self.prefix.lib, '-lsigsegv’,
                smoke_test_c,
                '-o', 'smoke_test'
            ]
            purpose='check linking’)

        self.run_test(
            ‘smoke_test’, [], data_dir.join('smoke_test.out’),
            purpose=‘run built smoke test’)

        self.run_test('sigsegv1': ['Test passed’], purpose='check sigsegv1 output’)
        self.run_test('sigsegv2': ['Test passed’], purpose='check sigsegv2 output’)
```

Tests are part of a regular Spack recipe class

Easily save source code from the package

User just defines a `test()` method

Retrieve saved source.
Link a simple executable.

Spack ensures that `cc` is a compatible compiler

Run the built smoke test and verify output

Run programs installed with package

EXASCALE COMPUTING PROJECT

`spack external find` (new in v0.15, updated for 0.16)

```python
class Cmake(Package):
    executables = ['cmake']

    @classmethod
    def determine_spec_details(cls, prefix, exes_in_prefix):
        exe_to_path = dict(
            (os.path.basename(p), p) for p in exes_in_prefix
        )
        if 'cmake' not in exe_to_path:
            return None

        cmake = spack.util.executable.Executable(exe_to_path['cmake'])
        output = cmake('--version', output=str)
        if output:
            match = re.search(r'cmake.*version\s+(\S+)', output)
            if match:
                version_str = match.group(1)
                return Spec('cmake@{0}'.format(version_str))
```

Logic for finding external installations in `package.py`

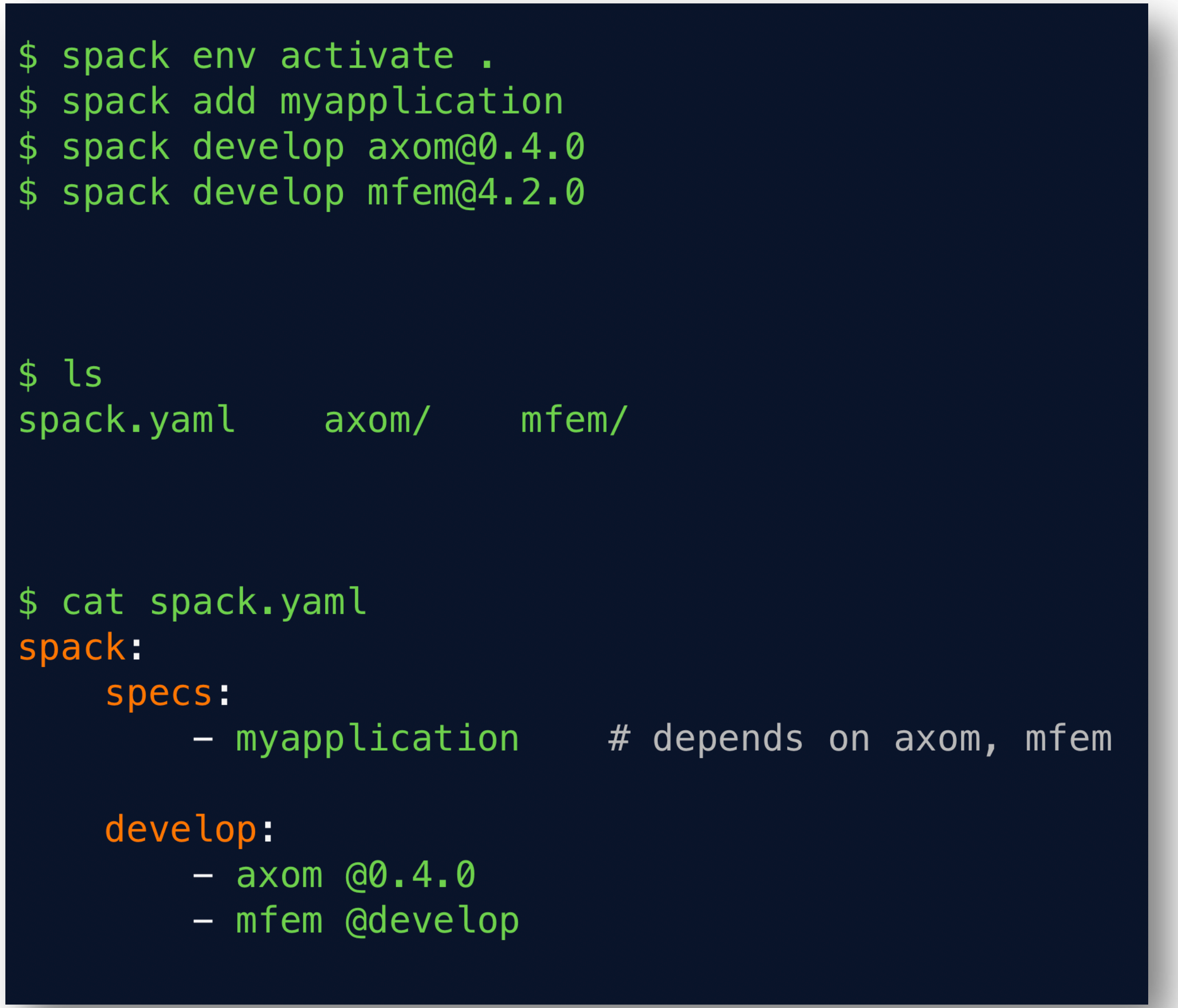
```yaml
packages:
  cmake:
    externals:
    - spec: cmake@3.15.1
      prefix: /usr/local
```

`packages.yaml` configuration

- Spack has has had compiler detection for a while
  - Finds compilers in your PATH
  - Registers them for use
- We can find any package now
  - Package defines:
    - possible command names
    - how to query the command
  - Spack searches for known commands and adds them to configuration
- Community can easily enable tools to be set up rapidly

`spack develop` lets developers work on many packages at once

- Developer features so far have focused on single packages (`spack dev-build`, etc.)
- New spack develop feature enables development environments
  - Work on a code
  - Develop multiple packages from its dependencies
  - Easily rebuild with changes
- Builds on spack envirnoments
  - Required changes to the installation model for dev packages
  - dev packages don't change paths with configuration changes
  - Allows devs to iterate on builds quickly



```
$ spack env activate .
$ spack add myapplication
$ spack develop axom@0.4.0
$ spack develop mfem@4.2.0



$ ls
spack.yaml    axom/    mfem/



$ cat spack.yaml
spack:
    specs:
        - myapplication    # depends on axom, mfem

    develop:
        - axom @0.4.0
        - mfem @develop
```

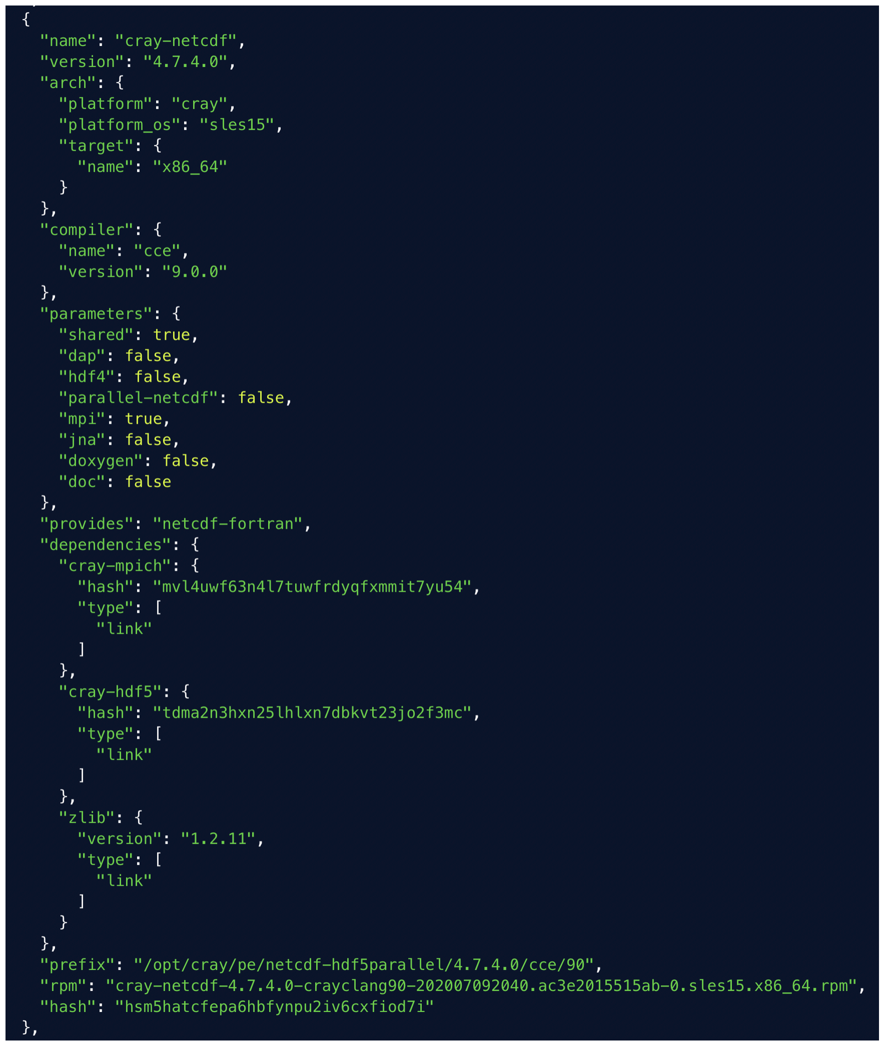
Under ECP, we are working to support the many exascale and pre-exascale platforms

- **v0.16 has much-needed support for new vendor compilers**
  - **oneapi**: Intel
  - **nvhpc**: NVIDIA
  - **aocc**: AMD
- **Tammy Dahlgren leading initiative to use spack test to test E4S on ECP early access systems**
  - We will be running continuous smoke tests for the ECP stack
- **GPU integration across the stack will be an ongoing focus**
  - **3 GPUs**: AMD, NVIDIA and Intel

ECP
EXASCALE COMPUTING PROJECT

Roadmap: We are working with HPE/Cray on tighter PE integration

- Using Cray Programming Environment's MPI, libsci, etc. currently requires a fair amount of configuration
  – Users have to register externals and go through modules
  – PrgEnvs make it hard to be precise about dependencies
- PE team has worked with us to develop a JSON format to describe PE contents
  – All packages and dependencies
  – Build provenance (compilers, targets, etc.)
  – Installation prefix
  – Which RPM it came from (interesting for containers)
- We'll be auto-detecting PE packages from this JSON
  – No more manual setup for PE packages
  – Manifest is included in current HPC/Cray PE releases
  – Currently iterating w/HPE on bugfixes, adding Spack support



```
{
  "name": "cray-netcdf",
  "version": "4.7.4.0",
  "arch": {
    "platform": "cray",
    "platform_os": "sles15",
    "target": {
      "name": "x86_64"
    }
  },
  "compiler": {
    "name": "cce",
    "version": "9.0.0"
  },
  "parameters": {
    "shared": true,
    "dap": false,
    "hdf4": false,
    "parallel-netcdf": false,
    "mpi": true,
    "jna": false,
    "doxygen": false,
    "doc": false
  },
  "provides": "netcdf-fortran",
  "dependencies": {
    "cray-mpich": {
      "hash": "mvl4uwf63n4l7tuwfrdyqfxmmit7yu54",
      "type": [
        "link"
      ]
    },
    "cray-hdf5": {
      "hash": "tdma2n3hxn25lhlxn7dbkvt23jo2f3mc",
      "type": [
        "link"
      ]
    },
    "zlib": {
      "version": "1.2.11",
      "type": [
        "link"
      ]
    }
  },
  "prefix": "/opt/cray/pe/netcdf-hdf5parallel/4.7.4.0/cce/90",
  "rpm": "cray-netcdf-4.7.4.0-crayclang90-202007092040.ac3e2015515ab-0.sles15.x86_64.rpm",
  "hash": "hsm5hatcfepa6hbfynpu2iv6cxfiod7i"
},
```

Cray PE JSON descriptor

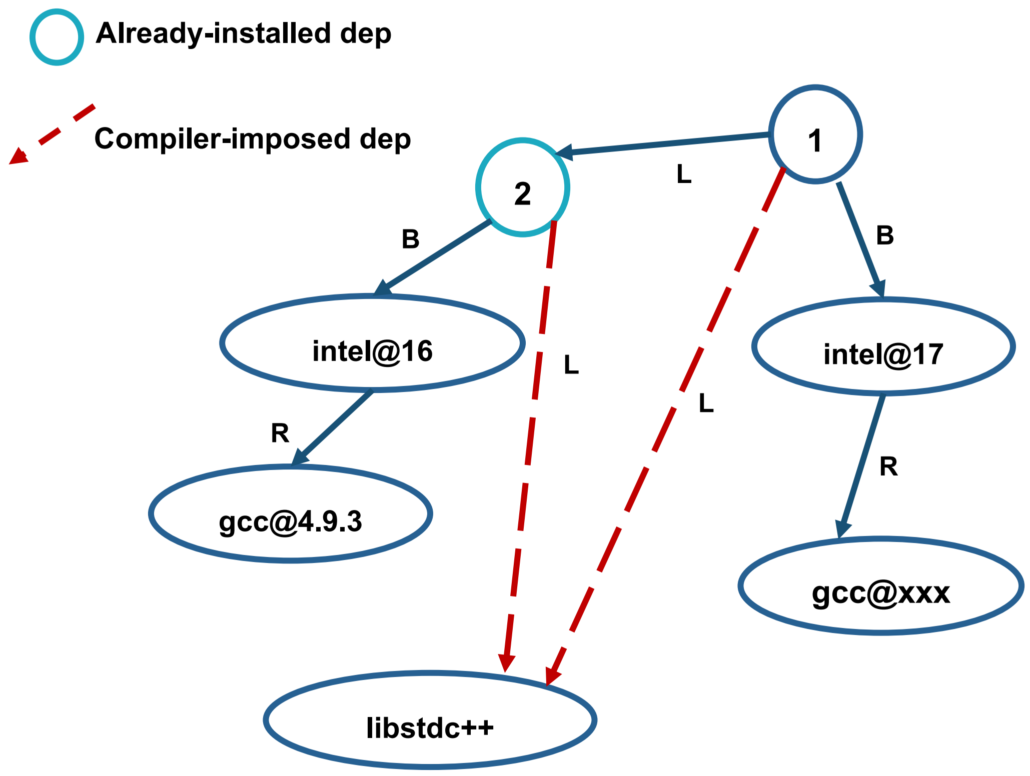
Spack 0.17 Roadmap: permissions and directory structure

- **Sharing a Spack instance**
  - Many users want to be able to install Spack on a cluster and `module load spack`
  - Installations in the Spack prefix are shared among users
  - Users would `spack install` to their home directory by default.
  - This requires us to move most state ***out*** of the Spack prefix
    - Installations would go into ~/.spack/…

- **Getting rid of configuration in ~/.spack**
  - While *installations* may move to the home directory, *configuration* there is causing issues
  - User configuration is like an unwanted global (e.g., LD_LIBRARY_PATH 😬)
    - Interferes with CI builds (many users will `rm -rf ~/.spack` to avoid it)
    - Goes against a lot of our efforts for reproducibility
    - Hard to manage this configuration between multiple machines
  - Environments are a much better fit
    - Make users keep configuration like this in an environment instead of a single config

Spack 0.17 Roadmap: compilers as dependencies

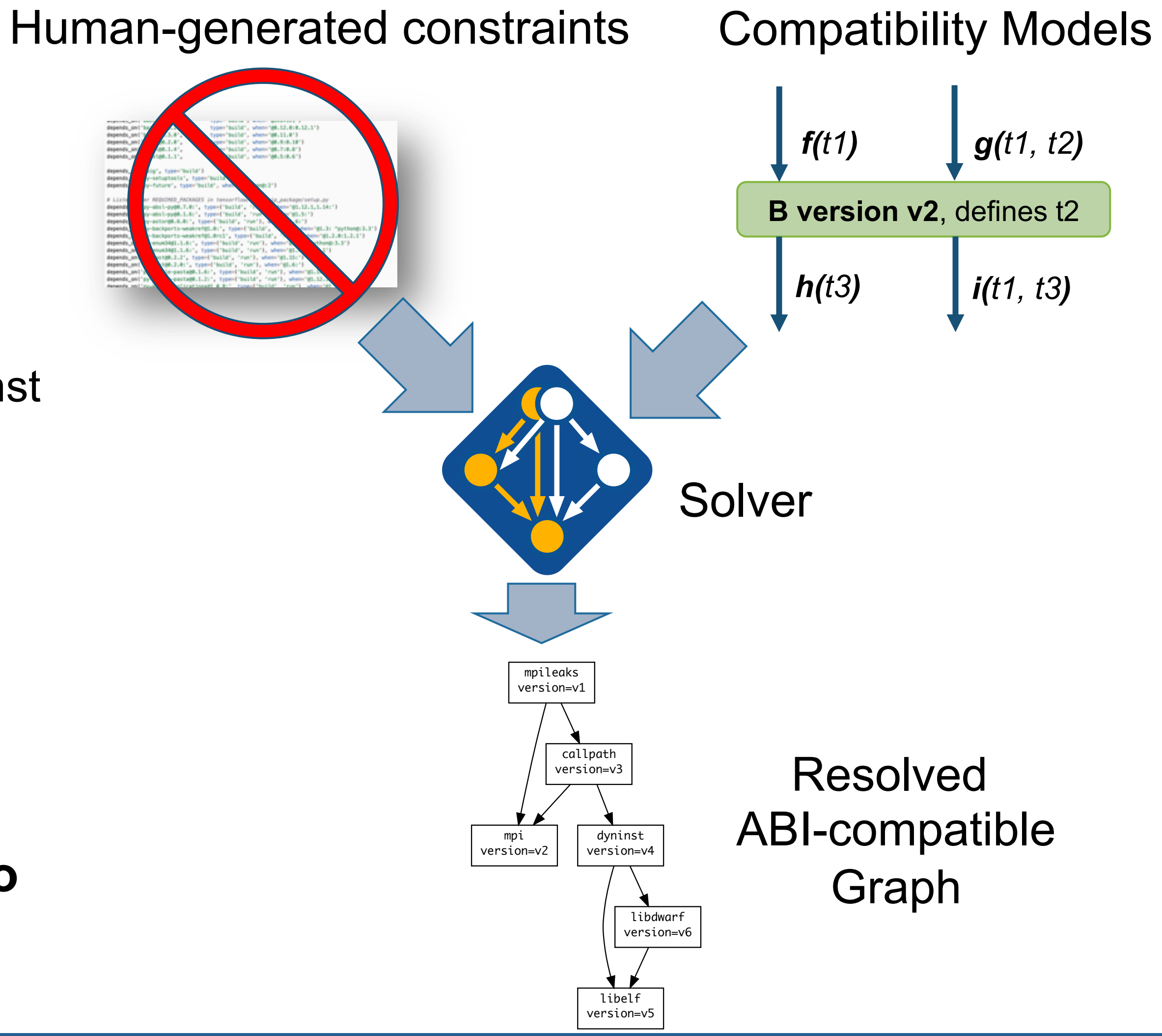
- **We need deeper modeling of compilers to handle compiler interoperability**
  - libstdc++, libc++ compatibility
  - Compilers that depend on compilers
  - Linking executables with multiple compilers
- **First prototype is complete!**
  - We've done successful builds of some packages using compilers as dependencies
  - We need the new concretizer to move forward!
- **Packages that depend on languages**
  - Depend on **cxx@2011**, **cxx@2017**, **fortran@1995**, etc
  - Depend on **openmp@4.5**, other compiler features
  - Model languages, openmp, cuda, etc. as virtuals



Compilers and runtime libs fully modeled as dependencies

LLNL recently kicked off a 3-year research project called BUILD

- **Basic premise: humans can’t generate all the compatibility constraints**
  - Version ranges, conflicts, in Spack packages not precise
  - rely on maintainers to get right.
- **BUILD aims to understand software compatibility**
  - Develop ABI compatibility models
  - Extract ABI information from binaries using libabigail, dyninst
  - Augment compatibility rules in solvers with ABI info
  - Enable *automatic* and ABI-compatible reuse of system binaries, foreign binary packages
- **Past 10-20 years have brought enormous improvements in solver technology**
  - CDCL algorithms, optimizing SMT and ASP solvers
  - Time is right to attack packaging with better solving
- **BUILD will integrate binary compatibility checks into dependency solvers**



More in our talk in the FOSDEM Dependency Management Devroom on Sunday!