Managing HPC Software Complexity with Spack

The most recent version of these slides can be found at:
https://spack-tutorial.readthedocs.io

ISC 2021 Half-day Tutorial
June 24, 2021

LLNL-PRES-806064

This work was performed under the auspices of the U.S. Department of Energy by Lawrence Livermore National Laboratory under contract DE-AC52-07NA27344. Lawrence Livermore National Security, LLC

spack.io

Lawrence Livermore National Laboratory

Tutorial Materials

Find these slides and associated scripts here:

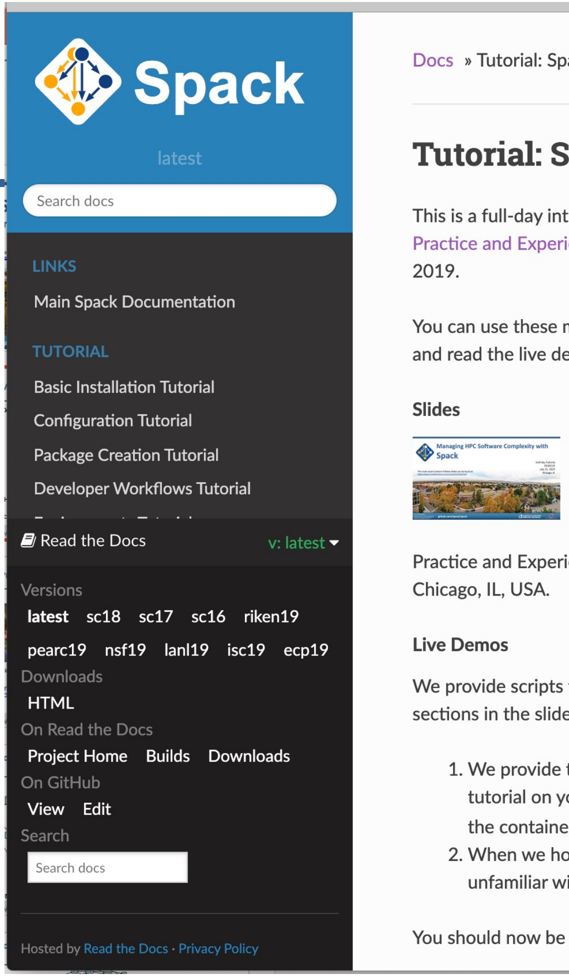
spack-tutorial.readthedocs.io

We will also have a chat room on Spack slack.
Get an invite here:

slack.spack.io

Join the “tutorial” channel!

We will give you login credentials for the hands-on exercises once you join Slack.



Spack

latest

Search docs

LINKS

Main Spack Documentation

TUTORIAL

Basic Installation Tutorial

Configuration Tutorial

Package Creation Tutorial

Developer Workflows Tutorial

Read the Docs v: latest

Versions

latest sc18 sc17 sc16 riken19

pearc19 nsf19 lanl19 isc19 ecp19

Downloads

HTML

On Read the Docs

Project Home Builds Downloads

On GitHub

View Edit

Search

Search docs

Hosted by Read the Docs · Privacy Policy

Docs » Tutorial: Sp

Tutorial: S

This is a full-day int
Practice and Experi
2019.

You can use these r
and read the live de

Slides

Managing HPC Software Complexity with Spack

Practice and Experi
Chicago, IL, USA.

Live Demos

We provide scripts
sections in the slide

1. We provide t
tutorial on yo
the containe
2. When we ho
unfamiliar wi

You should now be

Tutorial Presenters

Todd Gamblin

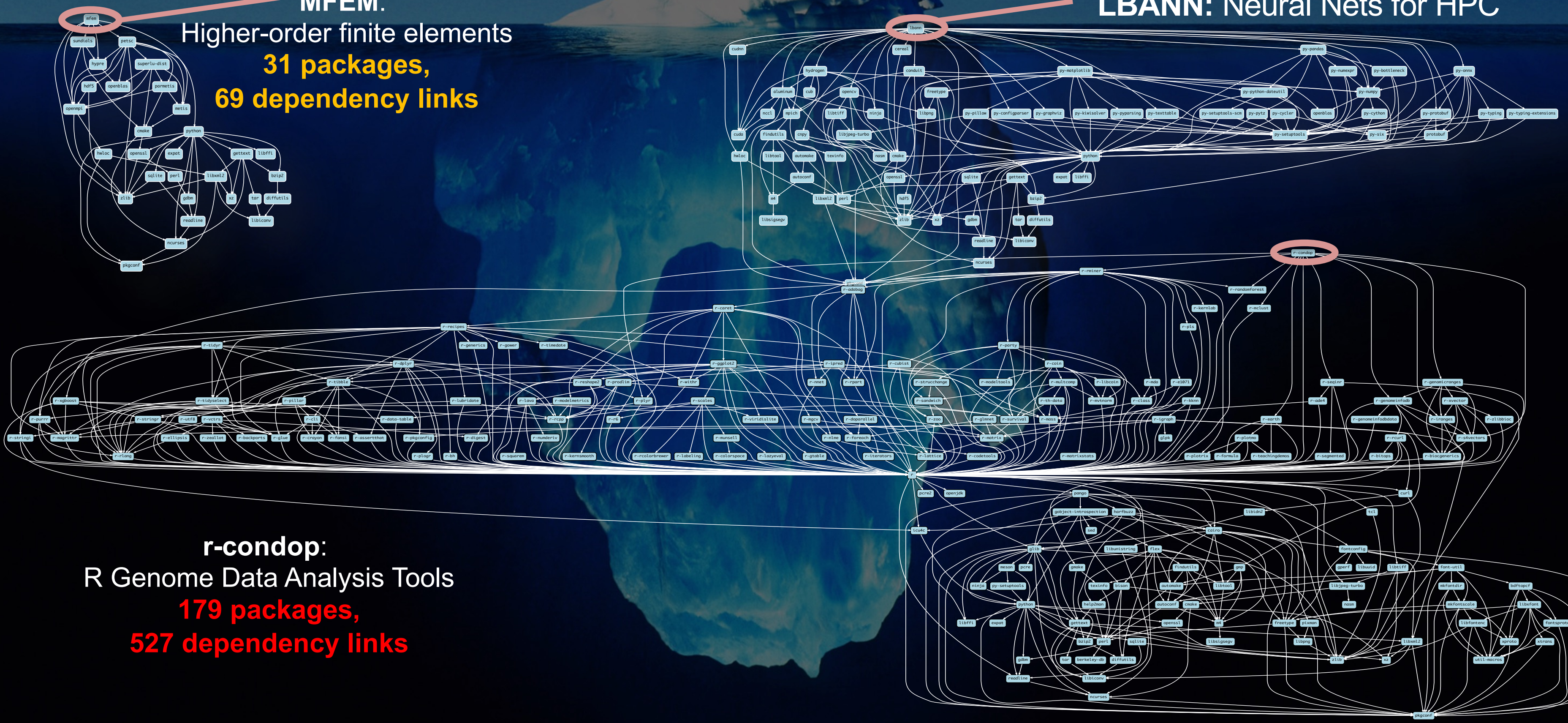
Greg Becker

Massimiliano Culpo

Tamara Dahlgren

Michael Kuhn

Modern scientific codes rely on icebergs of dependency libraries



MFEM:
Higher-order finite elements
31 packages,
69 dependency links

71 packages
188 dependency links
LBANN: Neural Nets for HPC

r-condop:
R Genome Data Analysis Tools
179 packages,
527 dependency links

Some fairly common (but questionable) assumptions made by package managers (conda, pip, apt, etc.)

- **1:1 relationship between source code and binary (per platform)**
  - Good for reproducibility (e.g., Debian)
  - Bad for performance optimization

- **Binaries should be as portable as possible**
  - What most distributions do
  - Again, bad for performance

- **Toolchain is the same across the ecosystem**
  - One compiler, one set of runtime libraries
  - Or, no compiler (for interpreted languages)

Outside these boundaries, users are typically on their own

High Performance Computing (HPC) violates many of these assumptions

- **Code is typically distributed as source**
  - With exception of vendor libraries, compilers
- **Often build many variants of the same package**
  - Developers' builds may be very different
  - Many first-time builds when machines are new
- **Code is optimized for the processor and GPU**
  - Must make effective use of the hardware
  - Can make 10-100x perf difference
- **Rely heavily on system packages**
  - Need to use optimized libraries that come with machines
  - Need to use host GPU libraries and network
- **Multi-language**
  - C, C++, Fortran, Python, others
    all in the same ecosystem

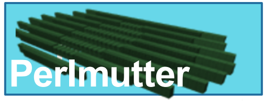
Some Supercomputers

Current

Oak Ridge National Lab
Power9 / NVIDIA

RIKEN
Fujitsu/ARM a64fx

Upcoming



Lawrence Berkeley National Lab
AMD Zen / NVIDIA

Argonne National Lab
Intel Xeon / Xe

Oak Ridge National Lab
AMD Zen / Radeon

Lawrence Livermore National Lab
AMD Zen / Radeon

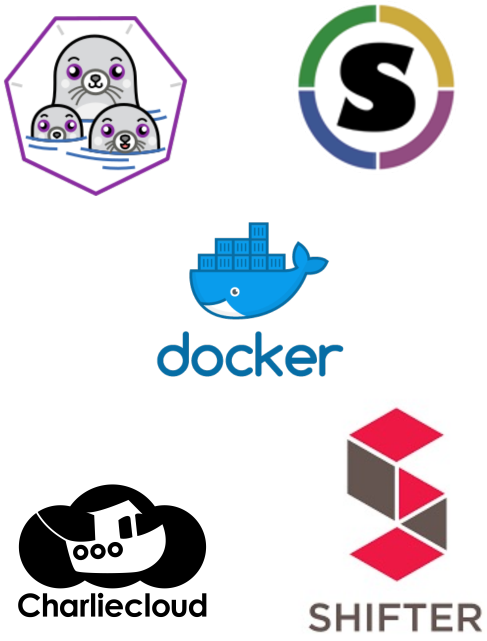
What about containers?

- **Containers provide a great way to reproduce and distribute an already-built software stack**

- **Someone needs to build the container!**
  - This isn't trivial
  - Containerized applications still have hundreds of dependencies

- **Using the OS package manager inside a container is insufficient**
  - Most binaries are built unoptimized
  - Generic binaries, not optimized for specific architectures

- **HPC containers may need to be *rebuilt* to support many different hosts, anyway.**
  - Not clear that we can ever build one container for all facilities
  - Containers likely won't solve the N-platforms problem in HPC



We need something more flexible to **build** the containers

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)

Who can use Spack?

People who want to use or distribute software for HPC!

1. **End Users of HPC Software**
   - Install and run HPC applications and tools

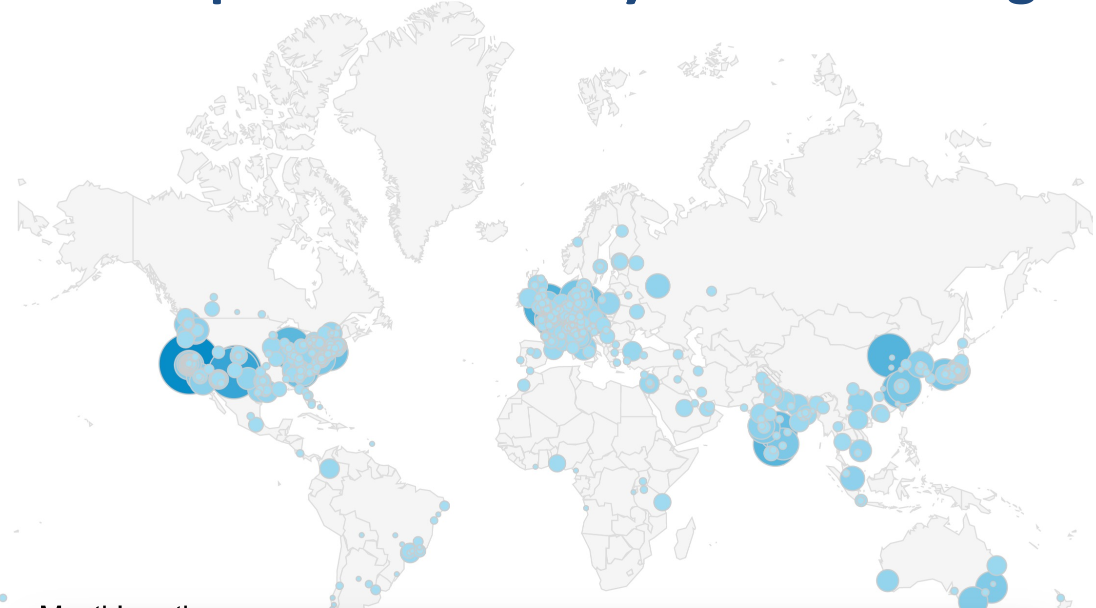
2. **HPC Application Teams**
   - Manage third-party dependency libraries

3. **Package Developers**
   - People who want to package their own software for distribution

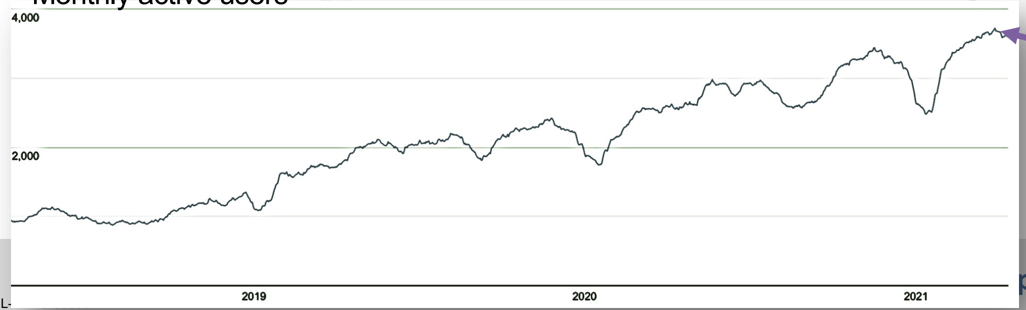
4. **User support teams at HPC Centers**
   - People who deploy software for users at large HPC sites

The Spack community continues to grow!

5,600+ software packages
820+ contributors



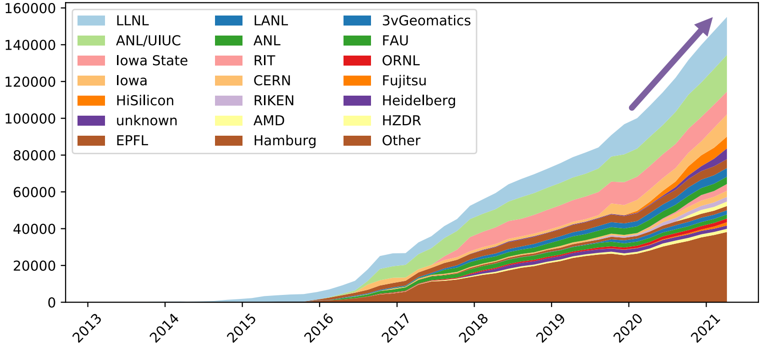
Monthly active users



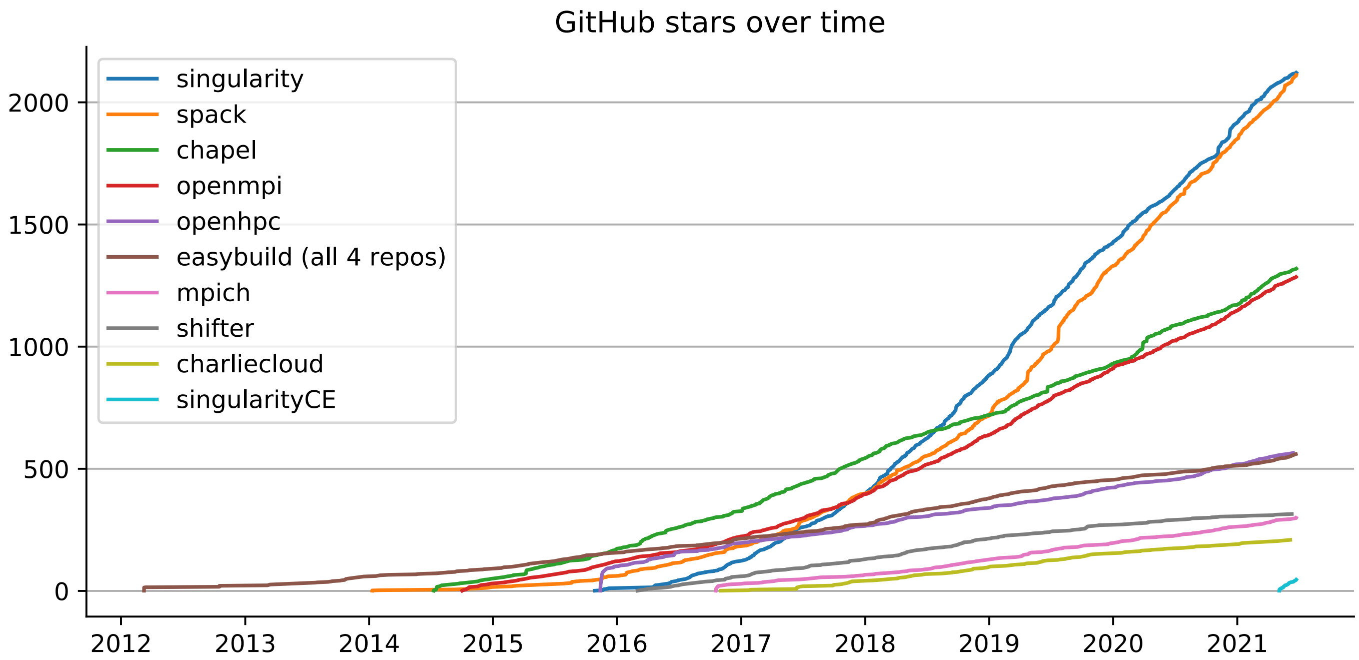
All time high of 3,700
monthly active users this March

Package contribution rate
increased in 2020

Contributions (lines of code) over time in packages, by organization

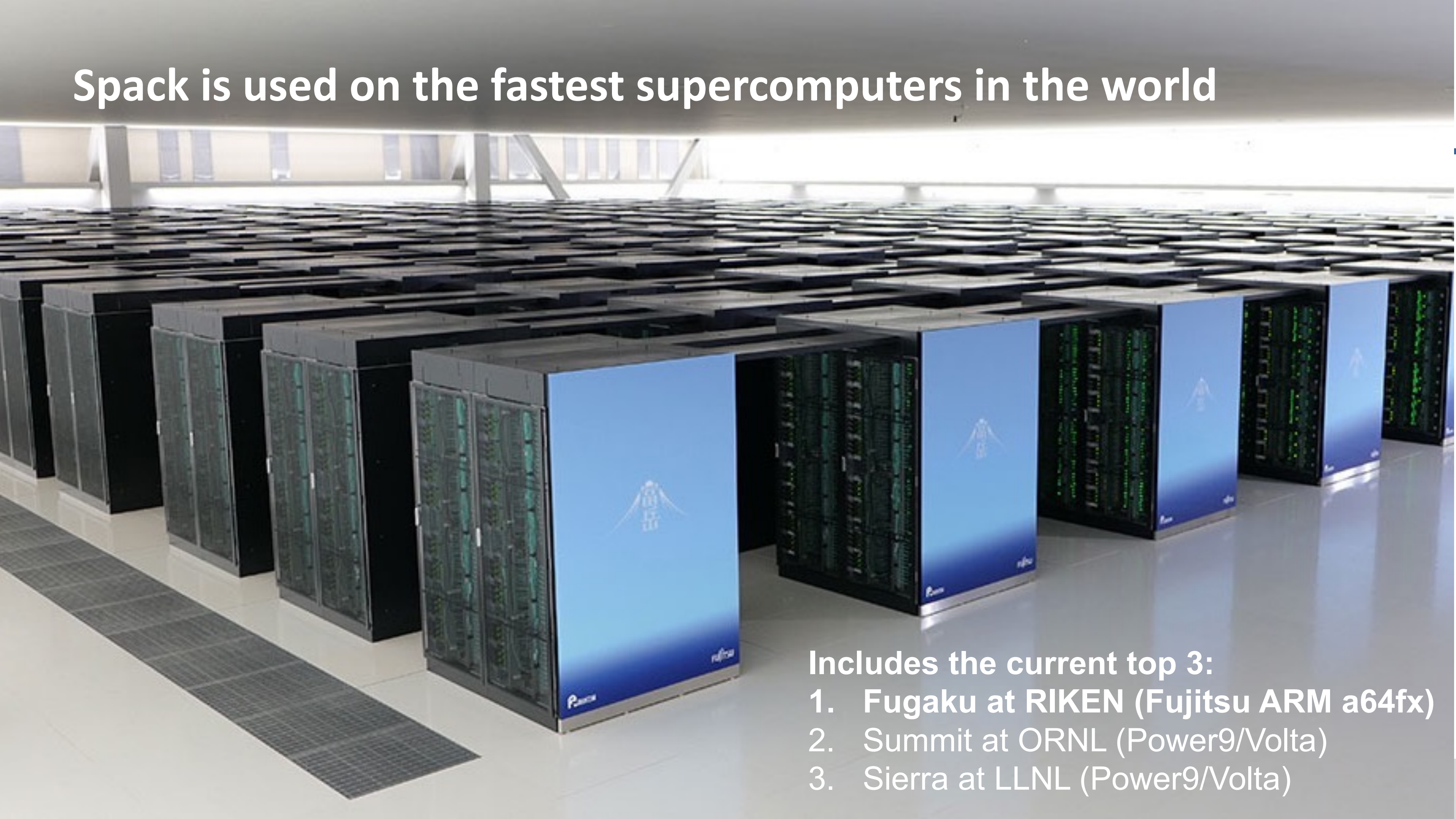


Spack has been gaining adoption rapidly (if stars are an indicator)



★ **Star Spack at github.com/spack/spack if you like the tutorial!**

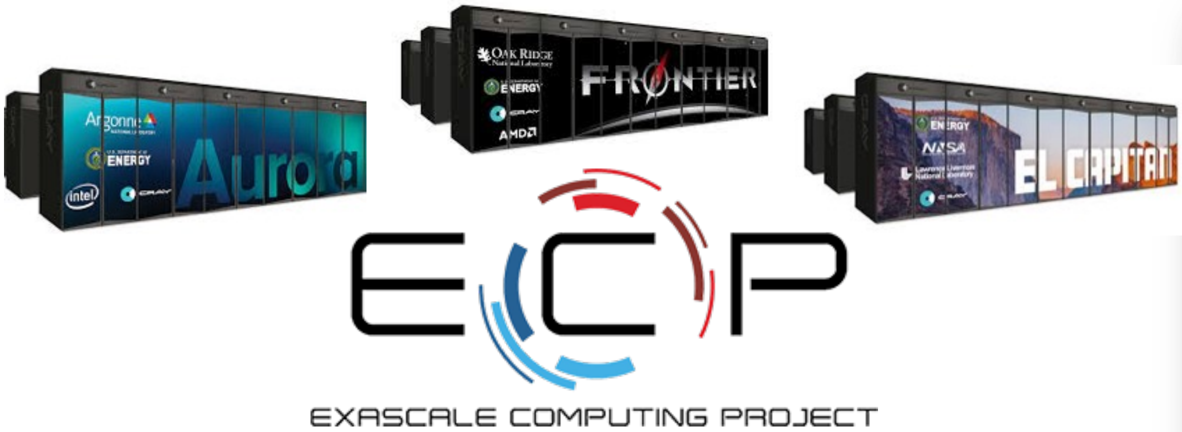
Spack is used on the fastest supercomputers in the world



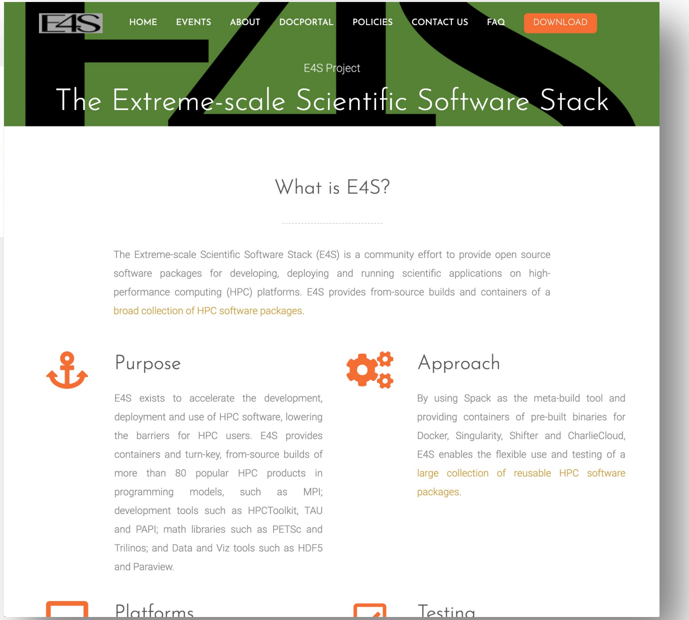
Includes the current top 3:

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

Spack is the deployment tool for the U.S. Exascale Computing Project


Argonne
ENERGY
intel
CRAY
Aurora
OAK RIDGE National Laboratory
ENERGY
CRAY
AMD
FRONTIER
ENERGY
NNSA
Lawrence Livermore National Laboratory
CRAY
EL CAPITAN
ECP
EXASCALE COMPUTING PROJECT

- Spack will be used to build software for the US's three upcoming exascale systems
- ECP has built the Extreme Scale Scientific Software Stack (E4S) with Spack – more at https://e4s.io
- We are helping ECP fulfill its mission – to create a robust and capable exascale software ecosystem



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

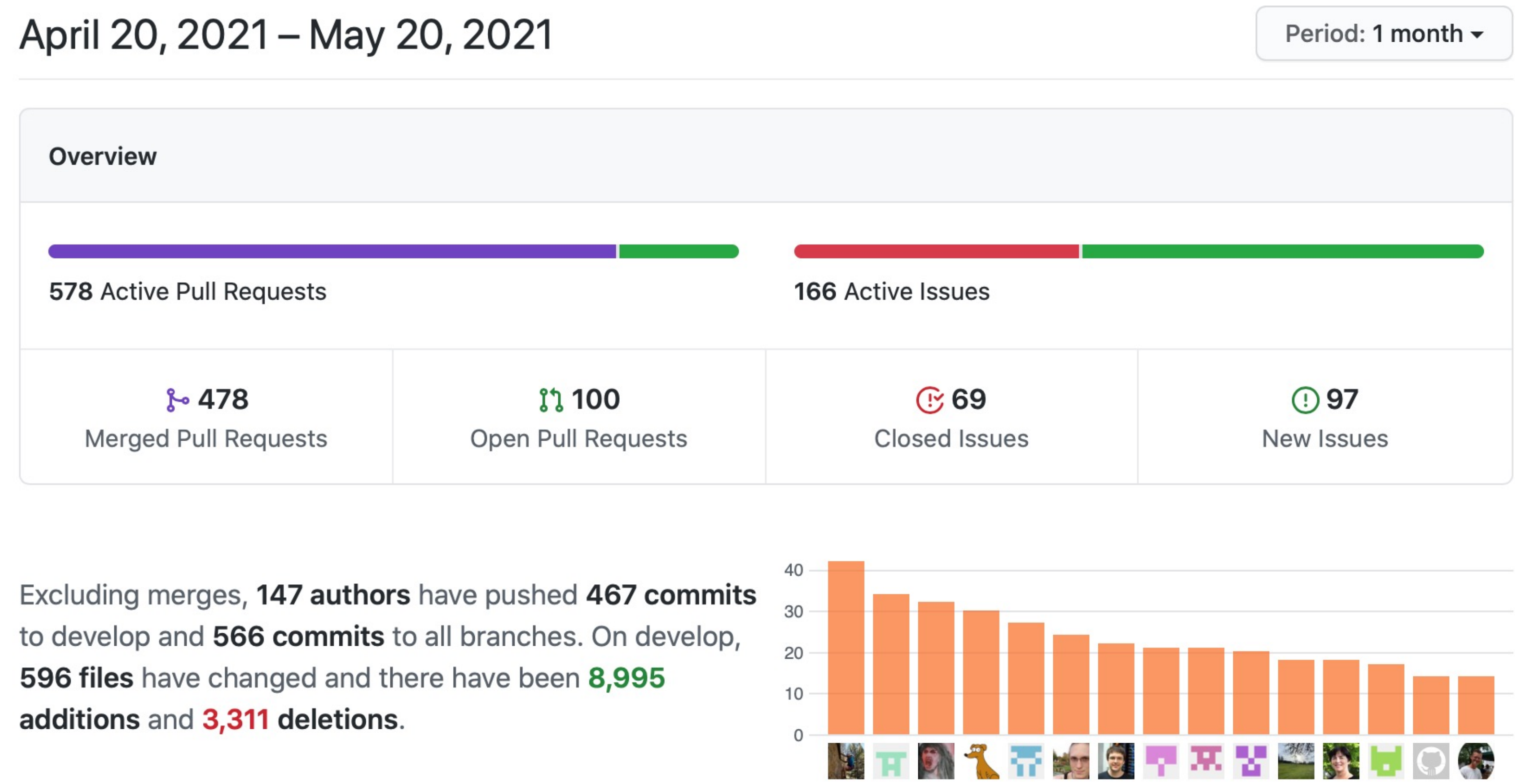
By using Spack as the meta-build tool and providing containers of pre-built binaries for Docker, Singularity, Shifter and CharlieCloud, E4S enables the flexible use and testing of a large collection of reusable HPC software packages.

Platforms

Testing

https://e4s.io

One month of Spack development is pretty busy!



April 20, 2021 – May 20, 2021

Period: 1 month ▾

Overview

578 Active Pull Requests

166 Active Issues

478	100	69	97
Merged Pull Requests	Open Pull Requests	Closed Issues	New Issues

Excluding merges, **147 authors** have pushed **467 commits** to develop and **566 commits** to all branches. On develop, **596 files** have changed and there have been **8,995 additions** and **3,311 deletions**.

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 User Survey 2020

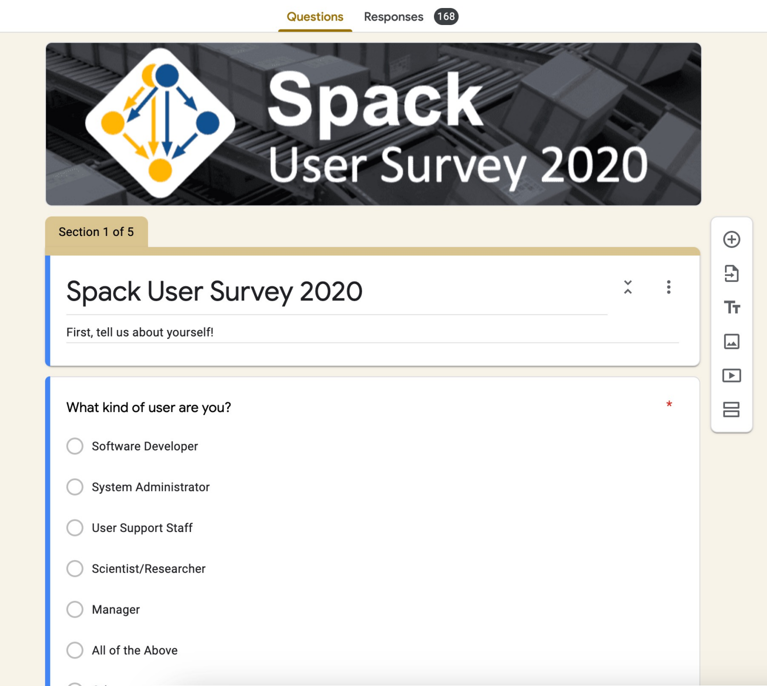
- First widely distributed Spack Survey
  - Sent to all of Slack (900+ users)
  - All of Spack mailing list, ECP mailing list

- 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

Results writeup and full survey data at:

https://spack.io/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

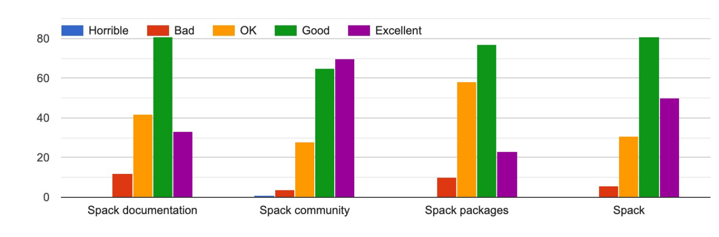
○ Scientist/Researcher

○ Manager

○ All of the Above

○ Other...

Where do you



Spack is not the only tool that automates builds

1. **"Functional" Package Managers**
   - Nix — https://nixos.org/
   - GNU Guix — https://www.gnu.org/s/guix/

2. **Build-from-source Package Managers**
   - Homebrew, LinuxBrew — http://brew.sh
   - MacPorts — https://www.macports.org
   - Gentoo — https://gentoo.org

Other tools in the HPC Space:

- **Easybuild** — http://hpcugent.github.io/easybuild/
  - An installation tool for HPC
  - Focused on HPC system administrators – different package model from Spack
  - Relies on a fixed software stack – harder to tweak recipes for experimentation

- **Conda** — https://conda.io
  - Very popular binary package manager for data science
  - Not targeted at HPC; generally has unoptimized binaries

Agenda

- Part 1

1.	Intro	Slides
2.	Basic Spack Usage	Hands-on
3.	Core Spack concepts	Slides
4.	Environments	Hands-on

Break

- Part 2

1.	Configuration	Hands-on
2.	Developer Workflows	Hands-on **(new!)**
3.	Binary Caches and Mirrors	Hands-on **(new!)**
4.	Future directions and roadmap	Slides

Hands-on Time: Spack Basics

Follow script at spack-tutorial.readthedocs.io

Core Spack Concepts

We will be resuming at 9am PT / 12 ET

If you have not yet joined us on slack,
get an invite here, join the tutorial channel,
and ask for a VM login!

Follow along with the tutorial here

Most existing tools do not support combinatorial versioning

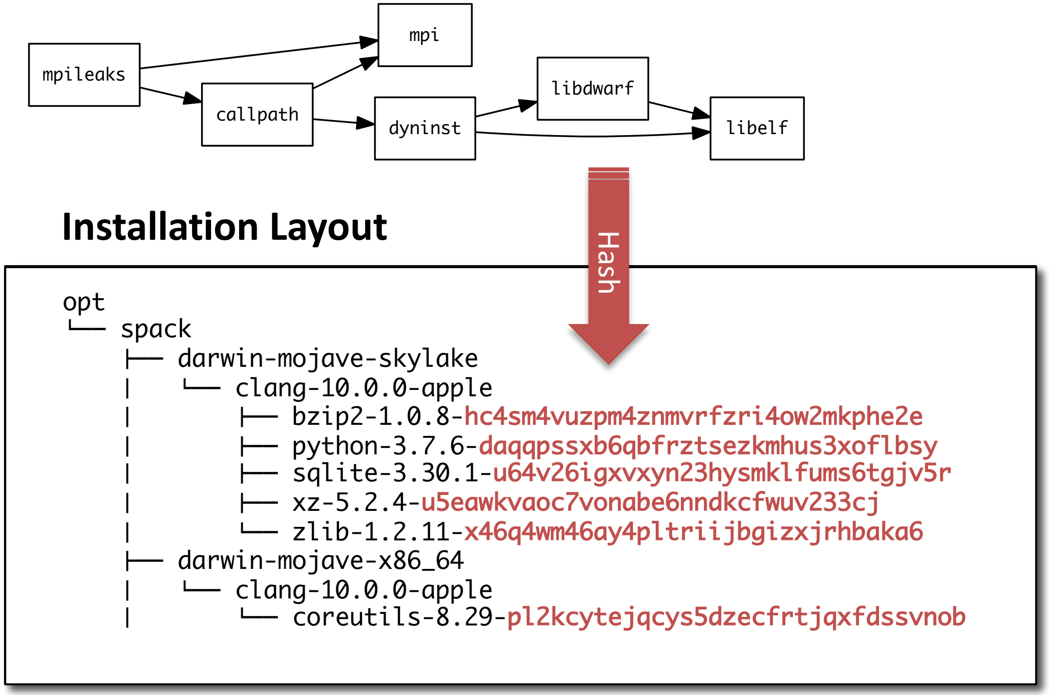
- Traditional binary package managers
  - RPM, yum, APT, yast, etc.
  - Designed to manage a single stack.
  - Install *one* version of each package in a single prefix (`/usr`).
  - Seamless upgrades to a *stable, well tested* stack

- Port systems
  - BSD Ports, portage, Macports, Homebrew, Gentoo, etc.
  - Minimal support for builds parameterized by compilers, dependency versions.

- Virtual Machines and Linux Containers (Docker)
  - Containers allow users to build environments for different applications.
  - Does not solve the build problem (someone has to build the image)
  - Performance, security, and upgrade issues prevent widespread HPC deployment.

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
```

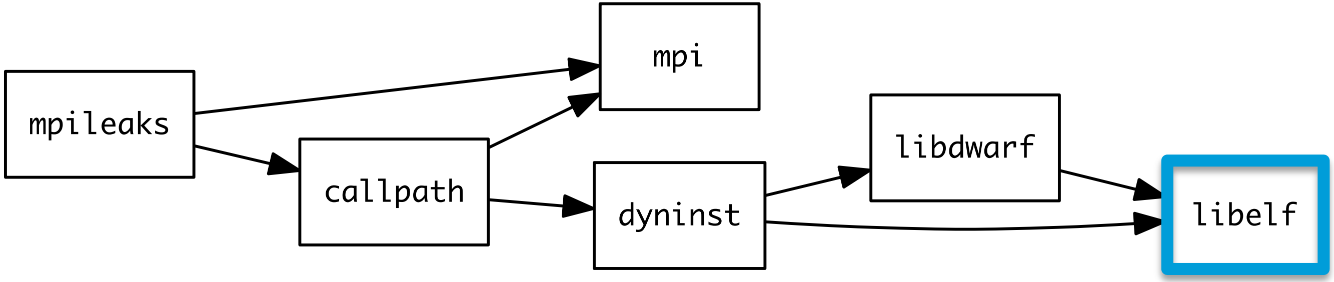
- Each unique dependency graph is a unique ***configuration***.

- 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 Specs can constrain versions of dependencies



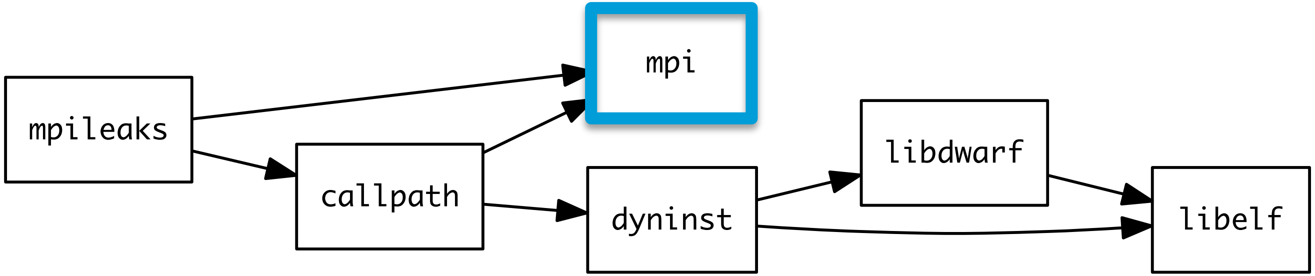
mpileaks
mpi
callpath
dyninst
libdwarf
libelf

```
$ spack install mpileaks %intel@12.1 ^libelf@0.8.12
```

- Spack ensures *one* configuration of each library per DAG
  - Ensures ABI consistency.
  - User does not need to know DAG structure; only the dependency *names.*

- Spack can ensure that builds use the same compiler, or you can mix
  - Working on ensuring ABI compatibility when compilers are mixed.

Spack handles ABI-incompatible, versioned interfaces like MPI


mpi
mpileaks
callpath
dyninst
libdwarf
libelf

- mpi is a *virtual dependency*

- Install the same package built with two different MPI implementations:

```
$ spack install mpileaks ^mvapich@1.9
```

```
$ spack install mpileaks ^openmpi@1.4:
```

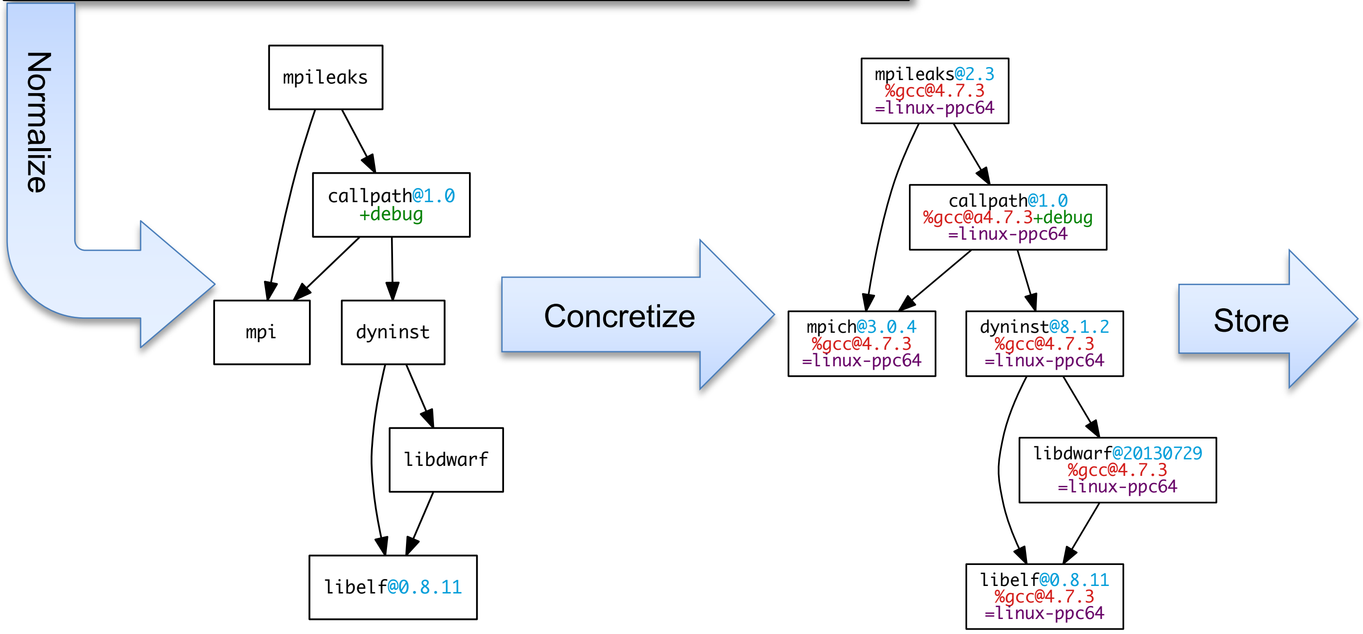
- Let Spack choose MPI implementation, as long as it provides MPI 2 interface:

```
$ spack install mpileaks ^mpi@2
```

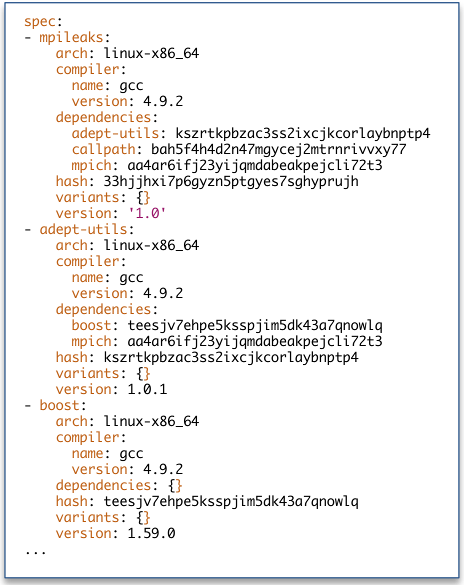
Concretization fills in missing configuration details when the user is not explicit.

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

User input: *abstract* spec with some constraints



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.

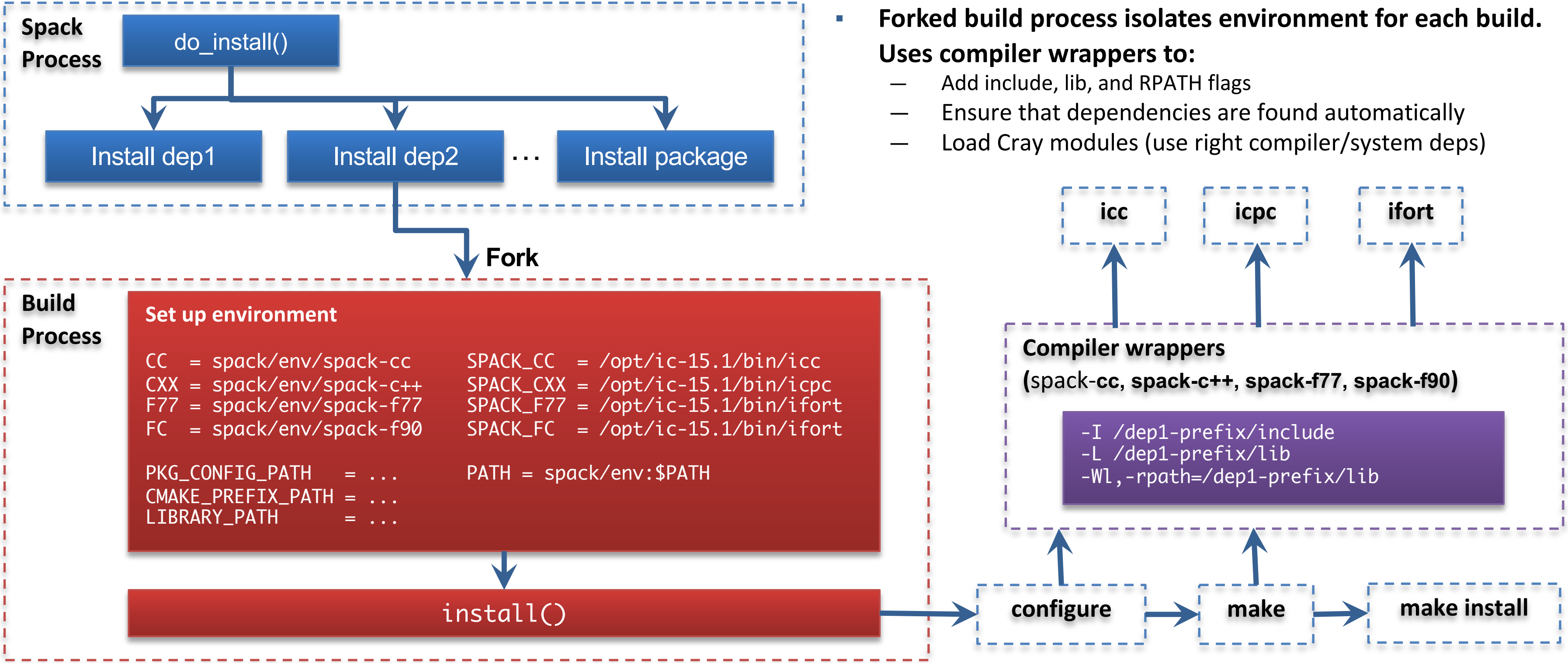
Detailed provenance is stored with the installed package

Use `spack spec` to see the results of concretization

```
$ spack spec mpileaks
Input spec
------------------------------
  mpileaks

Concretized
------------------------------
  mpileaks@1.0%gcc@5.3.0 arch=darwin-elcapitan-x86_64
      ^adept-utils@1.0.1%gcc@5.3.0 arch=darwin-elcapitan-x86_64
          ^boost@1.61.0%gcc@5.3.0+atomic+chrono+date_time~debug+filesystem~graph
           ~icu_support+iostreams+locale+log+math~mpi+multithreaded+program_options
           ~python+random +regex+serialization+shared+signals+singlethreaded+system
           +test+thread+timer+wave arch=darwin-elcapitan-x86_64
              ^bzip2@1.0.6%gcc@5.3.0 arch=darwin-elcapitan-x86_64
              ^zlib@1.2.8%gcc@5.3.0 arch=darwin-elcapitan-x86_64
          ^openmpi@2.0.0%gcc@5.3.0~mxm~pmi~psm~psm2~slurm~sqlite3~thread_multiple~tm~verbs+vt arch=darwin-elcapitan-x86_64
              ^hwloc@1.11.3%gcc@5.3.0 arch=darwin-elcapitan-x86_64
                  ^libpciaccess@0.13.4%gcc@5.3.0 arch=darwin-elcapitan-x86_64
                      ^libtool@2.4.6%gcc@5.3.0 arch=darwin-elcapitan-x86_64
                          ^m4@1.4.17%gcc@5.3.0+sigsegv arch=darwin-elcapitan-x86_64
                              ^libsigsegv@2.10%gcc@5.3.0 arch=darwin-elcapitan-x86_64
      ^callpath@1.0.2%gcc@5.3.0 arch=darwin-elcapitan-x86_64
          ^dyninst@9.2.0%gcc@5.3.0~stat_dysect arch=darwin-elcapitan-x86_64
              ^libdwarf@20160507%gcc@5.3.0 arch=darwin-elcapitan-x86_64
                  ^libelf@0.8.13%gcc@5.3.0 arch=darwin-elcapitan-x86_64
```

Spack builds each package in its own compilation environment

- **Forked build process isolates environment for each build. Uses compiler wrappers to:**
  - Add include, lib, and RPATH flags
  - Ensure that dependencies are found automatically
  - Load Cray modules (use right compiler/system deps)



```
CC  = spack/env/spack-cc      SPACK_CC  = /opt/ic-15.1/bin/icc
CXX = spack/env/spack-c++     SPACK_CXX = /opt/ic-15.1/bin/icpc
F77 = spack/env/spack-f77     SPACK_F77 = /opt/ic-15.1/bin/ifort
FC  = spack/env/spack-f90     SPACK_FC  = /opt/ic-15.1/bin/ifort

PKG_CONFIG_PATH   = ...       PATH = spack/env:$PATH
CMAKE_PREFIX_PATH = ...
LIBRARY_PATH      = ...
```

```
-I /dep1-prefix/include
-L /dep1-prefix/lib
-Wl,-rpath=/dep1-prefix/lib
```

Extensions and Python Support

- Spack installs each package in its own prefix

- Some packages need to be installed within directory structure of other packages
  - i.e., Python modules installed in $prefix/lib/python-<version>/site-packages
  - Spack supports this via extensions

```
class PyNumpy(Package):
    """NumPy is the fundamental package for scientific computing with Python."""

    homepage = "https://numpy.org"
    url      = "https://pypi.python.org/packages/source/n/numpy/numpy-1.9.1.tar.gz"
    version('1.9.1', ' 78842b73560ec378142665e712ae4ad9’)

    extends('python’)

    def install(self, spec, prefix):
        setup_py("install“, "--prefix={0}".format(prefix))
```

Spack extensions

- Some packages need to be installed within directory structure of other packages

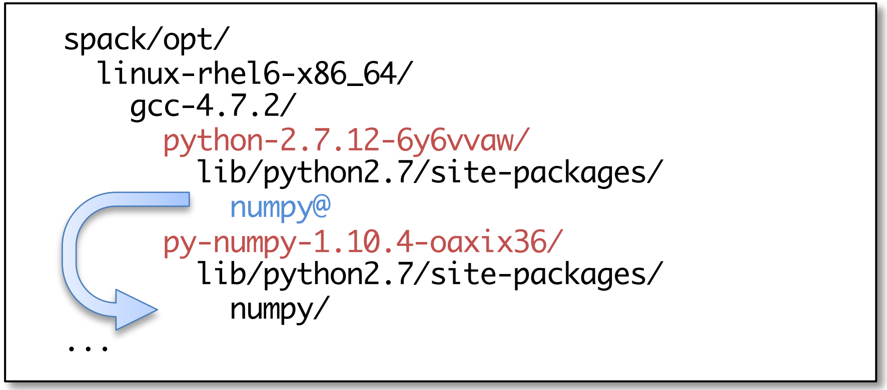
- Examples of extension packages:
  - python libraries are a good example
  - R, Lua, perl
  - Need to maintain combinatorial versioning

```
$ spack activate py-numpy @1.10.4
```

- Symbolic link to Spack install location

- This is an older feature – we are encouraging users to use **spack environments** instead
  - More on this later!

```
spack/opt/
  linux-rhel6-x86_64/
    gcc-4.7.2/
      python-2.7.12-6y6vvaw/
        lib/python2.7/site-packages/
          ..
      py-numpy-1.10.4-oaxix36/
        lib/python2.7/site-packages/
          numpy/
...
```



```
spack/opt/
  linux-rhel6-x86_64/
    gcc-4.7.2/
      python-2.7.12-6y6vvaw/
        lib/python2.7/site-packages/
          numpy@
      py-numpy-1.10.4-oaxix36/
        lib/python2.7/site-packages/
          numpy/
...
```

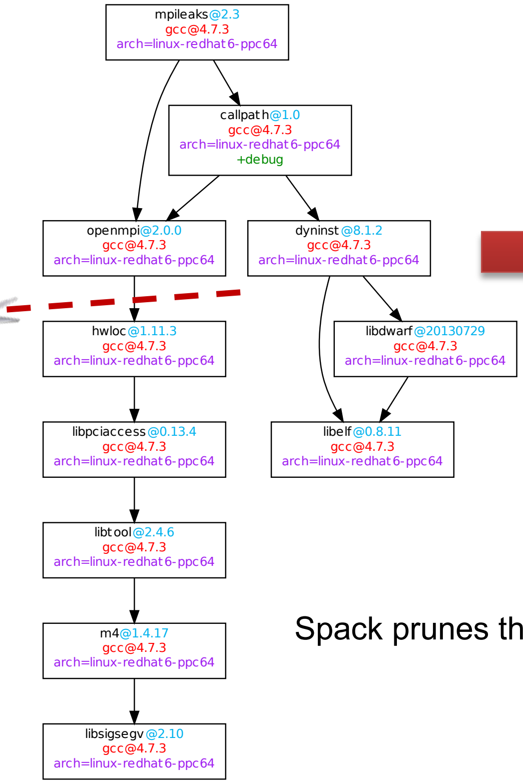
Building against externally installed software

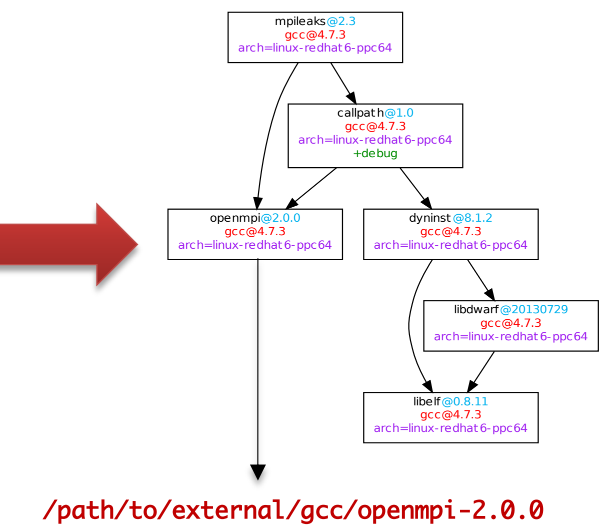
```
mpileaks ^callpath@1.0+debug
    ^openmpi ^libelf@0.8.11
```

packages.yaml

```
packages:
  mpi:
    buildable: False
    paths:
      openmpi@2.0.0 %gcc@4.7.3 arch=linux-rhel6-ppc64:
        /path/to/external/gcc/openmpi-2.0.0
      openmpi@1.10.3 %gcc@4.7.3 arch=linux-rhel6-ppc64:
        /path/to/external/gcc/openmpi-1.10.3
      ...
```

Users register external packages in a configuration file (more on these later).


mpileaks@2.3 gcc@4.7.3 arch=linux-redhat6-ppc64
callpath@1.0 gcc@4.7.3 arch=linux-redhat6-ppc64 +debug
openmpi@2.0.0 gcc@4.7.3 arch=linux-redhat6-ppc64
dyninst@8.1.2 gcc@4.7.3 arch=linux-redhat6-ppc64
hwloc@1.11.3 gcc@4.7.3 arch=linux-redhat6-ppc64
libdwarf@20130729 gcc@4.7.3 arch=linux-redhat6-ppc64
libpciaccess@0.13.4 gcc@4.7.3 arch=linux-redhat6-ppc64
libelf@0.8.11 gcc@4.7.3 arch=linux-redhat6-ppc64
libtool@2.4.6 gcc@4.7.3 arch=linux-redhat6-ppc64
m4@1.4.17 gcc@4.7.3 arch=linux-redhat6-ppc64
libsigsegv@2.10 gcc@4.7.3 arch=linux-redhat6-ppc64


mpileaks@2.3 gcc@4.7.3 arch=linux-redhat6-ppc64
callpath@1.0 gcc@4.7.3 arch=linux-redhat6-ppc64 +debug
openmpi@2.0.0 gcc@4.7.3 arch=linux-redhat6-ppc64
dyninst@8.1.2 gcc@4.7.3 arch=linux-redhat6-ppc64
libdwarf@20130729 gcc@4.7.3 arch=linux-redhat6-ppc64
libelf@0.8.11 gcc@4.7.3 arch=linux-redhat6-ppc64
/path/to/external/gcc/openmpi-2.0.0

Spack prunes the DAG when adding external packages.

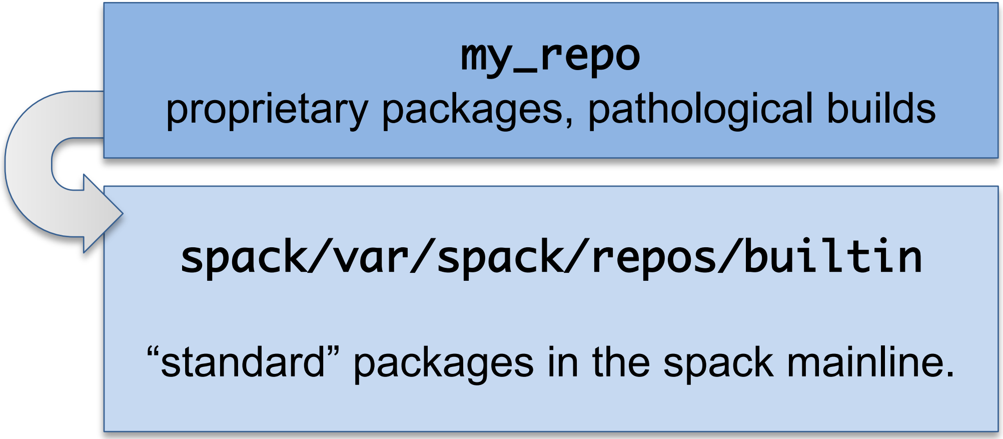
Spack package repositories

- Spack supports external package repositories
  - Separate directories of package recipes

- Many reasons to use this:
  - Some packages can’t be released publicly
  - Some sites require ~~bizarre~~ custom builds
  - Override default packages with site-specific versions

- Packages are composable:
  - External repositories can be layered on top of the built-in packages
  - Custom packages can depend on built-in packages (or packages in other repos)

```
$ spack repo create /path/to/my_repo
$ spack repo add my_repo
$ spack repo list
==> 2 package repositories.
my_repo     /path/to/my_repo
builtin     spack/var/spack/repos/builtin
```



my_repo

proprietary packages, pathological builds

spack/var/spack/repos/builtin

“standard” packages in the spack mainline.

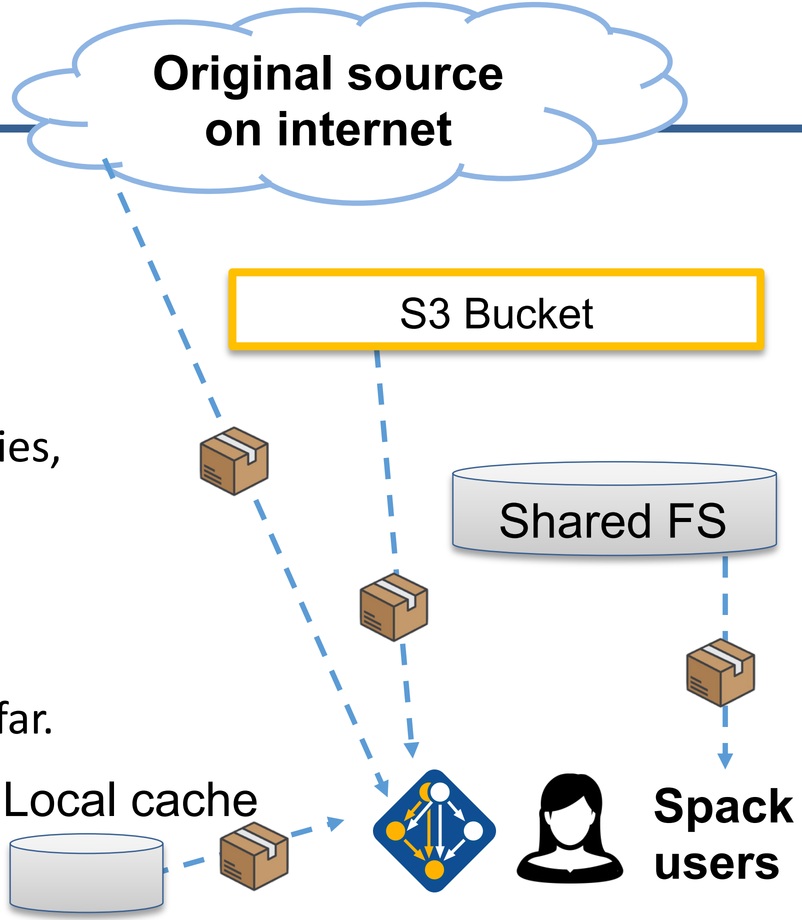
Spack mirrors

- Spack allows you to define *mirrors:*
  - Directories in the filesystem
  - On a web server
  - In an S3 bucket

- Mirrors are archives of fetched tarballs, repositories, and other resources needed to build
  - Can also contain binary packages

- By default, Spack maintains a mirror in var/spack/cache of everything you've fetched so far.

- You can host mirrors internal to your site
  - See the documentation for more details



Environments,

spack.yaml and spack.lock

Follow script at spack-tutorial.readthedocs.io

Hands-on Time: Configuration

Follow script at spack-tutorial.readthedocs.io

Tutorial Materials

Find these slides and associated scripts here:

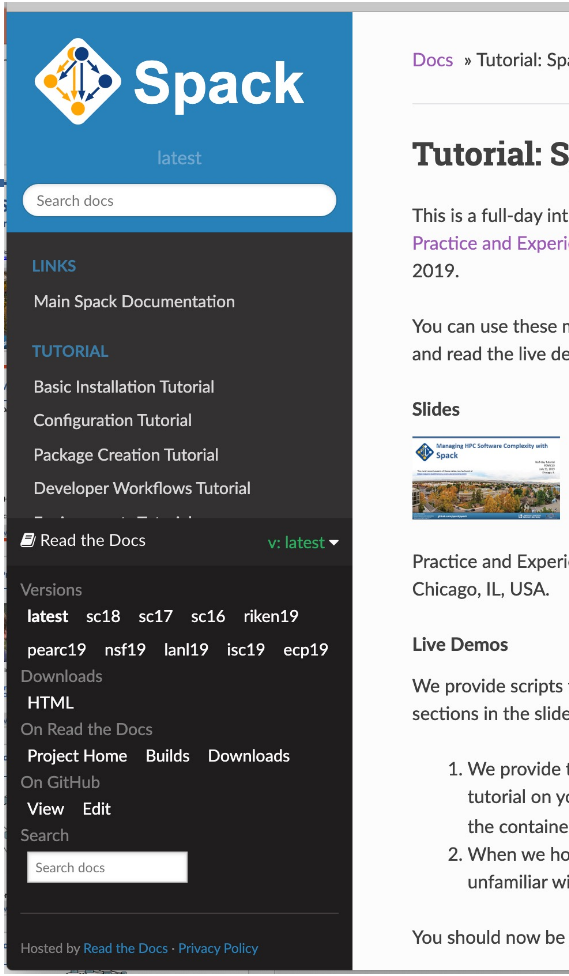
spack-tutorial.readthedocs.io

We will also have a chat room on Spack slack.
Get an invite here:

slack.spack.io

Join the "tutorial" channel!

We will give you login credentials for the hands-on exercises once you join Slack.



Spack

latest

Search docs

LINKS

Main Spack Documentation

TUTORIAL

Basic Installation Tutorial

Configuration Tutorial

Package Creation Tutorial

Developer Workflows Tutorial

Read the Docs v: latest

Versions

latest sc18 sc17 sc16 riken19

pearc19 nsf19 lanl19 isc19 ecp19

Downloads

HTML

On Read the Docs

Project Home Builds Downloads

On GitHub

View Edit

Search

Search docs

Hosted by Read the Docs · Privacy Policy

Docs » Tutorial: Sp

Tutorial: S

This is a full-day int
Practice and Experi
2019.

You can use these r
and read the live de

Slides

Practice and Experi
Chicago, IL, USA.

Live Demos

We provide scripts
sections in the slide

1. We provide t
tutorial on yo
the containe
2. When we ho
unfamiliar wi

You should now be

Hands-on Time: Creating Packages

Follow script at spack-tutorial.readthedocs.io

Hands-on Time:
Developer Workflows

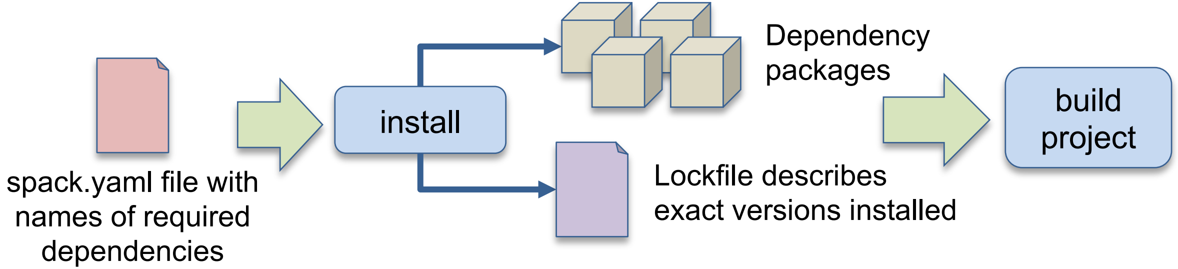
Follow script at spack-tutorial.readthedocs.io

Hands-on Time:
Binary Caches and Mirrors

Follow script at spack-tutorial.readthedocs.io

More New Features and the Road Ahead

Spack environments are the basis for complex workflows



spack.yaml file with names of required dependencies

install

Dependency packages

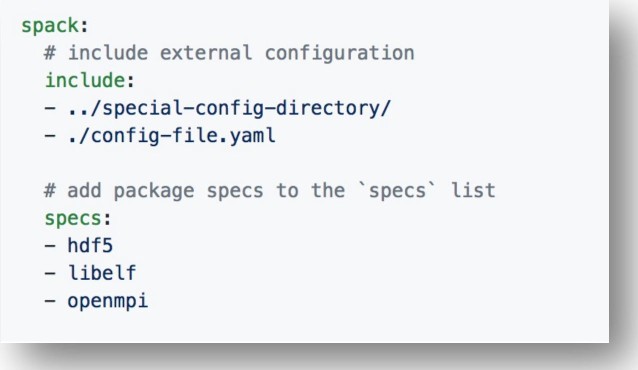
Lockfile describes exact versions installed

build project

- Two files:
  - spack.yaml describes project requirements
  - spack.lock records installed versions and configurations exactly
  - Enables reproducibility for many configurations

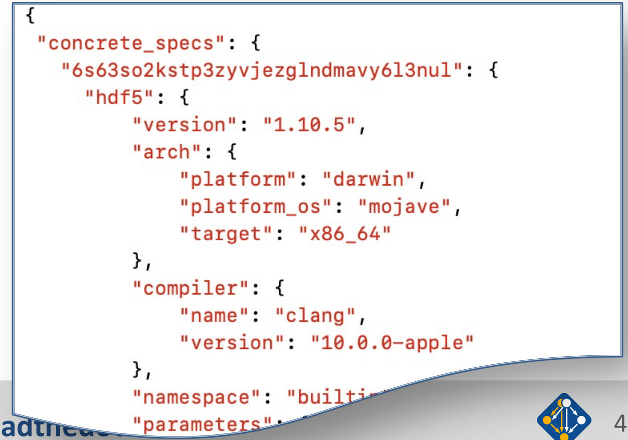
- Can use environments for:
  - Creating containers (spack containerize)
  - Auto-generate continuous integration builds (spack ci)
  - Deployment (matrix, spack stacks)
  - **Developer workflows (new!)**

Simple spack.yaml file



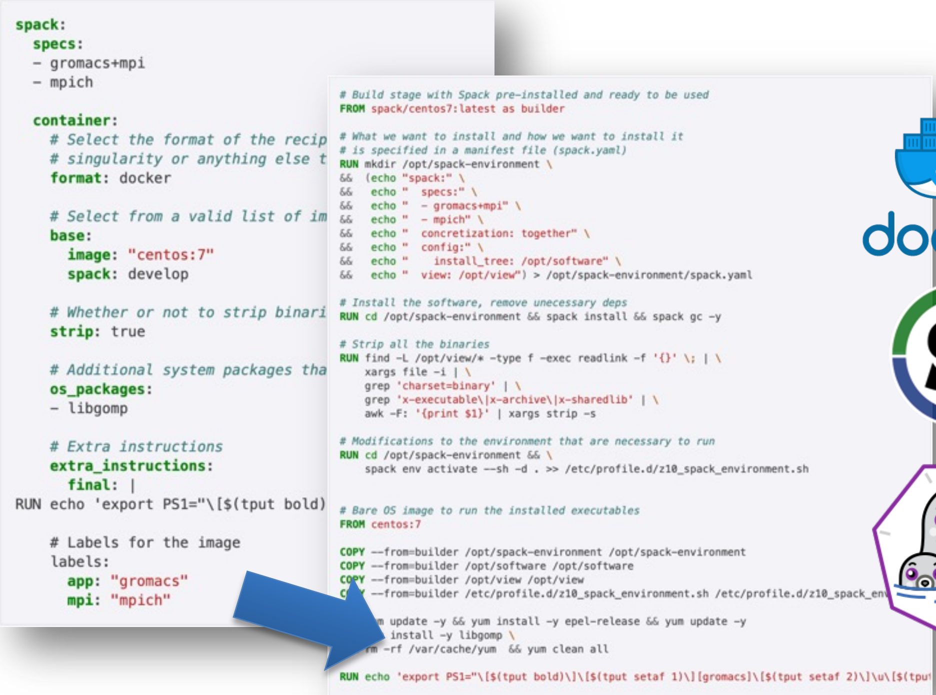
```yaml
spack:
  # include external configuration
  include:
  - ../special-config-directory/
  - ./config-file.yaml

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

Concrete spack.lock file (generated)



```json
{
 "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": "builti
         "parameters":
```

Generate container images from environments (0.14)



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

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

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

    # Whether or not to strip binari
    strip: true

    # Additional system packages tha
    os_packages:
    - libgomp

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

    # 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_env

    m update -y && yum install -y epel-release && yum update -y
     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\[$(tpu
```

- 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`

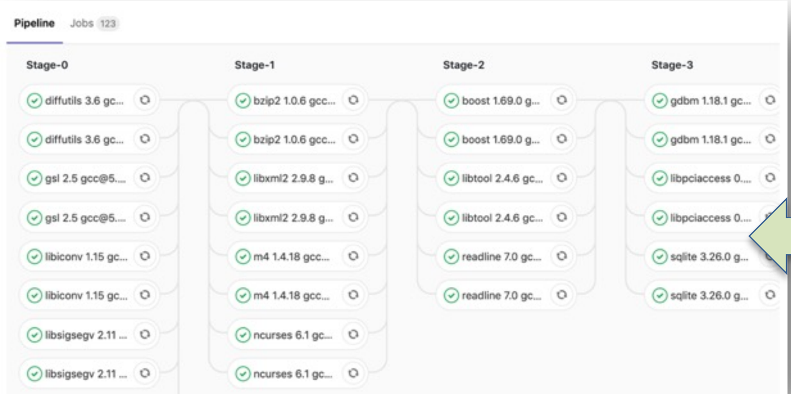
- Can also generate Singularity recipes

`spack containerize`

Spack can generate CI Pipelines from environments

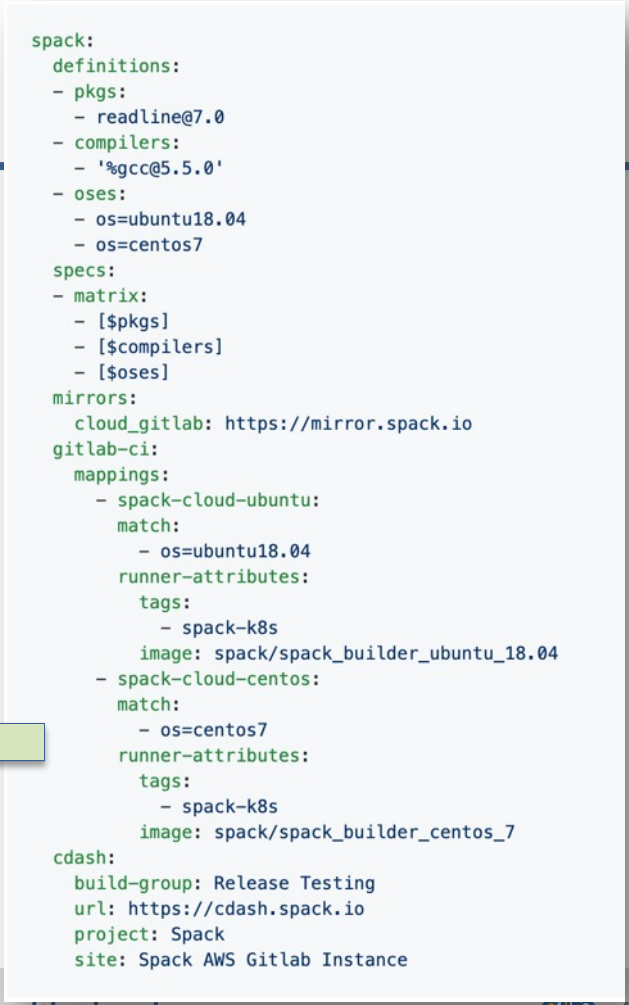
- User adds a `gitlab-ci` section to environment
  - Spack maps builds to GitLab runners
  - Generate gitlab-ci.yml with `spack ci` command

- Can run in a Kube cluster or on bare metal at an HPC site
  - Sends progress to CDash



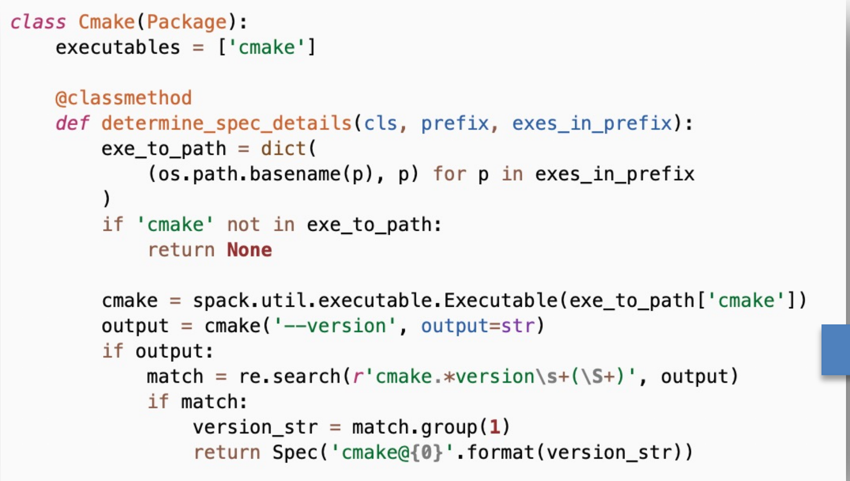
Stage-0	Stage-1	Stage-2	Stage-3
diffutils 3.6 gc...	bzip2 1.0.6 gcc...	boost 1.69.0 g...	gdbm 1.18.1 gc...
diffutils 3.6 gc...	bzip2 1.0.6 gcc...	boost 1.69.0 g...	gdbm 1.18.1 gc...
gsl 2.5 gcc@5....	libxml2 2.9.8 g...	libtool 2.4.6 gc...	libpciaccess 0....
gsl 2.5 gcc@5....	libxml2 2.9.8 g...	libtool 2.4.6 gc...	libpciaccess 0....
libiconv 1.15 gc...	m4 1.4.18 gcc...	readline 7.0 gc...	sqlite 3.26.0 g...
libiconv 1.15 gc...	m4 1.4.18 gcc...	readline 7.0 gc...	sqlite 3.26.0 g...
libsigsegv 2.11 ...	ncurses 6.1 gc...		
libsigsegv 2.11 ...	ncurses 6.1 gc...		

`spack ci`



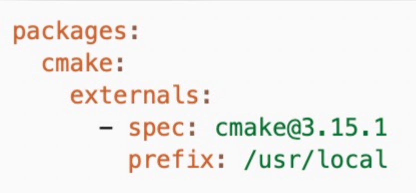
```yaml
spack:
  definitions:
  - pkgs:
    - readline@7.0
  - compilers:
    - '%gcc@5.5.0'
  - oses:
    - os=ubuntu18.04
    - os=centos7
  specs:
  - matrix:
    - [$pkgs]
    - [$compilers]
    - [$oses]
  mirrors:
    cloud_gitlab: https://mirror.spack.io
  gitlab-ci:
    mappings:
      - spack-cloud-ubuntu:
        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
```

spack external find



```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 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 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.

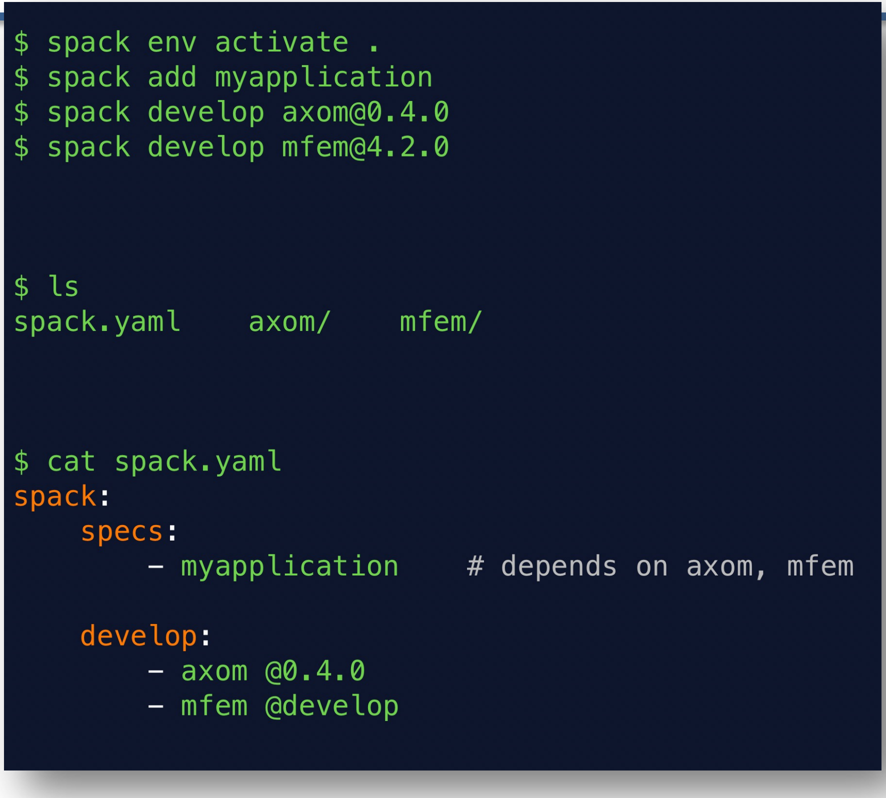
Spack ensures that `cc` is a compatible compiler

Run the built smoke test and verify output

Run programs installed with package

`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

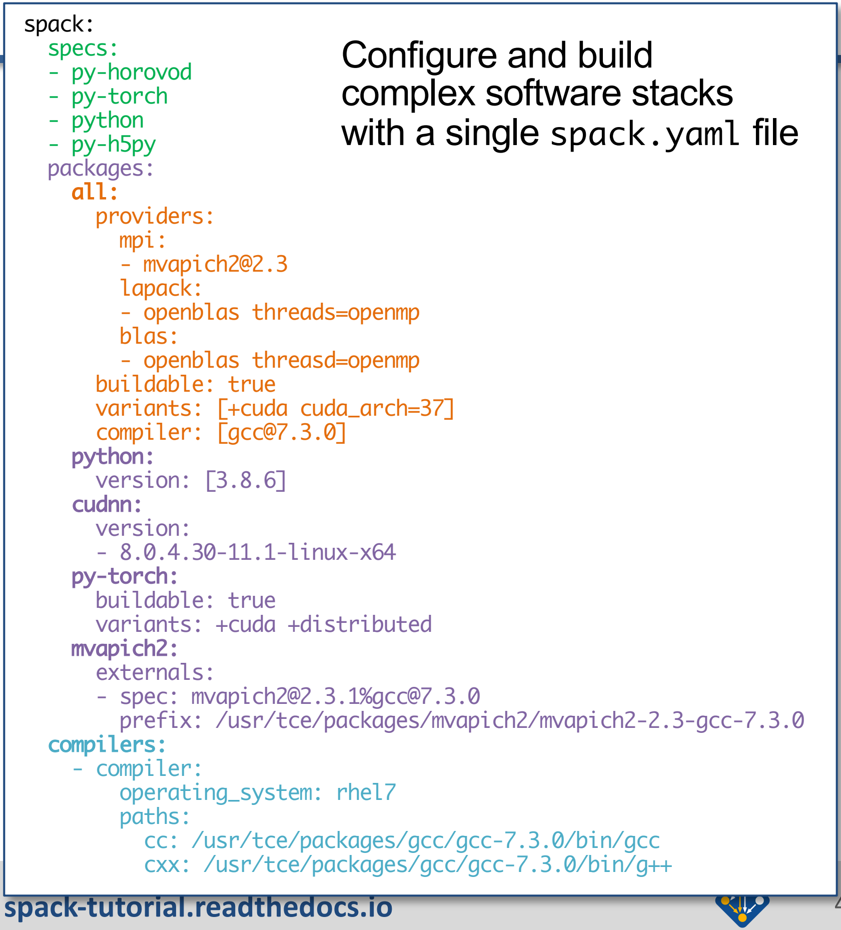


```
$ spack env activate .
$ spack add myapplication
$ spack develop axom@0.4.0
$ spack develop mfem@4.2.0



$ ls
spack.yaml    axom/    mfem/



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

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

Spack helped streamline the AML team's development environments.

- **Before Spack**
  - Everybody built their own python/pytorch from scratch
  - People wrote scripts and passed them around
  - Scripts slowly accumulated modifications and magic
  - **Days were spent trying to debug build differences**

- **After spack**
  - Versioned reproducible spack enviroments in a repo
  - Standard environments in a shared team directory
  - **Any team member can get a customizable working environment in ~20 minutes.**
    - Change python version, change pytorch version, etc.

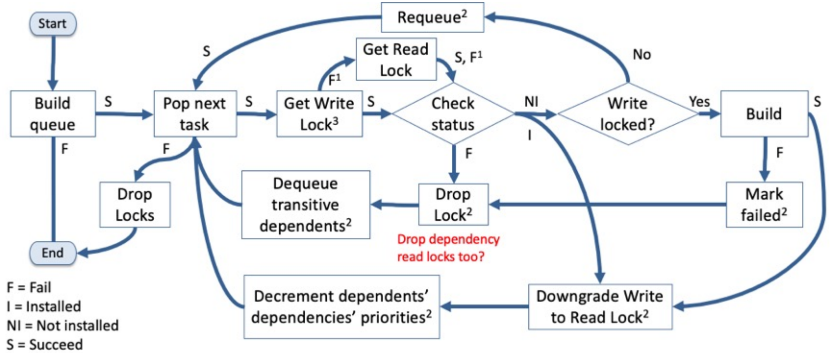


Configure and build complex software stacks with a single `spack.yaml` file

```
spack:
  specs:
  - py-horovod
  - py-torch
  - python
  - py-h5py
  packages:
    all:
      providers:
        mpi:
        - mvapich2@2.3
        lapack:
        - openblas threads=openmp
        blas:
        - openblas threasd=openmp
      buildable: true
      variants: [+cuda cuda_arch=37]
      compiler: [gcc@7.3.0]
    python:
      version: [3.8.6]
    cudnn:
      version:
      - 8.0.4.30-11.1-linux-x64
    py-torch:
      buildable: true
      variants: +cuda +distributed
    mvapich2:
      externals:
      - spec: mvapich2@2.3.1%gcc@7.3.0
        prefix: /usr/tce/packages/mvapich2/mvapich2-2.3-gcc-7.3.0
  compilers:
    - compiler:
        operating_system: rhel7
        paths:
          cc: /usr/tce/packages/gcc/gcc-7.3.0/bin/gcc
          cxx: /usr/tce/packages/gcc/gcc-7.3.0/bin/g++
```

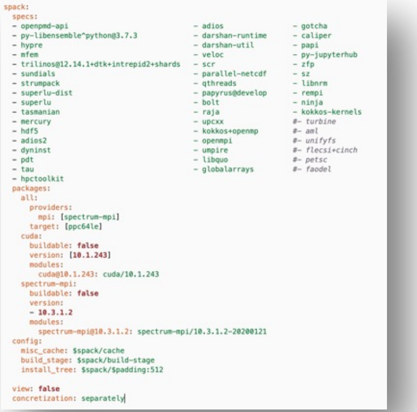
Spack’s parallel build support can complete 297 E4S packages in 85 minutes on a single node

```
srun -N 1 -n 8 spack install .
```



Distributed locking algorithm

- Previously only got parallelism in single insta
  - Now, all packages in an environment are built bottom up
- We have developed a novel lock-based algorithm
  - Requires no scheduler integration or server
  - Uses only reader/writer fcntl locks to coordinate across processes/nodes
  - Works on any distributed file system with flock enabled
- Easily build entire environment manifests at once



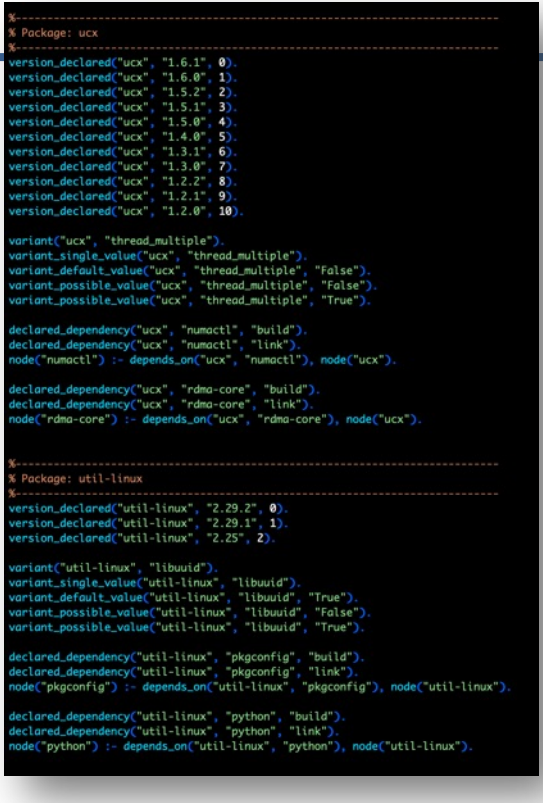
```
spack:
  specs:
  - openpmd-api                               - adios              - gotcha
  - py-libensemble^python@3.7.3               - darshan-runtime    - caliper
  - hypre                                     - darshan-util       - papi
  - mfem                                      - veloc              - 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
```

E4S Manifest

Build configuration is its own many-dimensional constraint optimization problem

- **The new concretizer in v0.16.0** allows us to solve this problem
  - Uses *Answer Set Programming* – framework for solving NP-hard optimization problems
  - Unlike other systems, package manager has insight into build details and configuration

- ASP program has 2 parts:
  1. Large list of facts
     - generated from our package repositories
     - 20,000 – 30,000 facts is typical
     - includes dependencies, versions, options, etc.
  2. Small logic program
     - ~800 lines of ASP code
     - 300 rules + 11 optimization criteria



```
%----------------------------------------------------------------------
% 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").
```

Sample ASP input for Spack solver

The new concretizer enables significant simplifications to packages, particularly complex constraints in SDKs

- Dependencies and other constraints within SDKs could get very messy

- The new concretizer removes the need for some of the more painful constructs

- Also allows for new constructs, like specializing dependencies
  - When conditions are now much more general
  - Can be solved together with other constraints.

In some cases we needed cross-products of dependency options:

Before

```
depends_on('foo+A+B', when='+a+b')
depends_on('foo+A~B', when='+a~b')
depends_on('foo~A+B', when='~a+b')
depends_on('foo~A~B', when='~a~b')
```

After

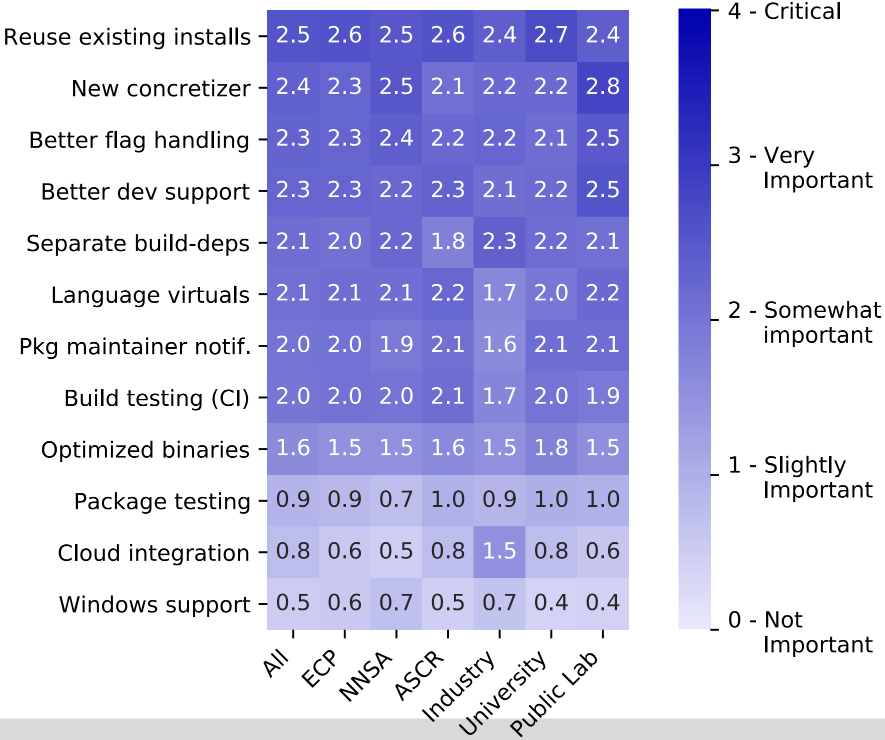
```
depends_on('foo')
depends_on('foo+A', when='+a')
depends_on('foo+B', when='+b’)
```

Specializing a virtual did not previously work:

```
depends_on('blas’)
depends_on(
    'openblas threads=openmp’, when='^openblas’
)
```

Four of the top six most wanted features in Spack are tied to the new concretizer

Average feature importance by workplace



	All	ECP	NNSA	ASCR	Industry	University	Public Lab
Reuse existing installs	2.5	2.6	2.5	2.6	2.4	2.7	2.4
New concretizer	2.4	2.3	2.5	2.1	2.2	2.2	2.8
Better flag handling	2.3	2.3	2.4	2.2	2.2	2.1	2.5
Better dev support	2.3	2.3	2.2	2.3	2.1	2.2	2.5
Separate build-deps	2.1	2.0	2.2	1.8	2.3	2.2	2.1
Language virtuals	2.1	2.1	2.1	2.2	1.7	2.0	2.2
Pkg maintainer notif.	2.0	2.0	1.9	2.1	1.6	2.1	2.1
Build testing (CI)	2.0	2.0	2.0	2.1	1.7	2.0	1.9
Optimized binaries	1.6	1.5	1.5	1.6	1.5	1.8	1.5
Package testing	0.9	0.9	0.7	1.0	0.9	1.0	1.0
Cloud integration	0.8	0.6	0.5	0.8	1.5	0.8	0.6
Windows support	0.5	0.6	0.7	0.5	0.7	0.4	0.4

- Complexity of packages in Spack is increasing
  - many more package solves require backtracking than a year ago
  - Many variants, conditional dependencies, special compiler requirements

- More aggressive reuse of existing installs requires better dependency resolution
  - Need to be able to analyze how to configure the build to work with installed packages

- Separate resolution of build dependencies also requires a more sophisticated solver
  - Makes the solve even more combinatorial
  - Needed to support mixed compilers, version conflicts between different package’s build requirements

Part of milestone STED09-8

We will be releasing v0.17 soon

Main goals:

1. Get rid of the old concretizer, make the new concretizer default
2. Improve and harden binary cache workflows
3. Make Spack able to optimize for reuse of installed packages and packages from binary mirrors
4. Make “shared” spack instances for facilities more manageable
5. Get rid of pain points like ~/.spack configuration

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/…

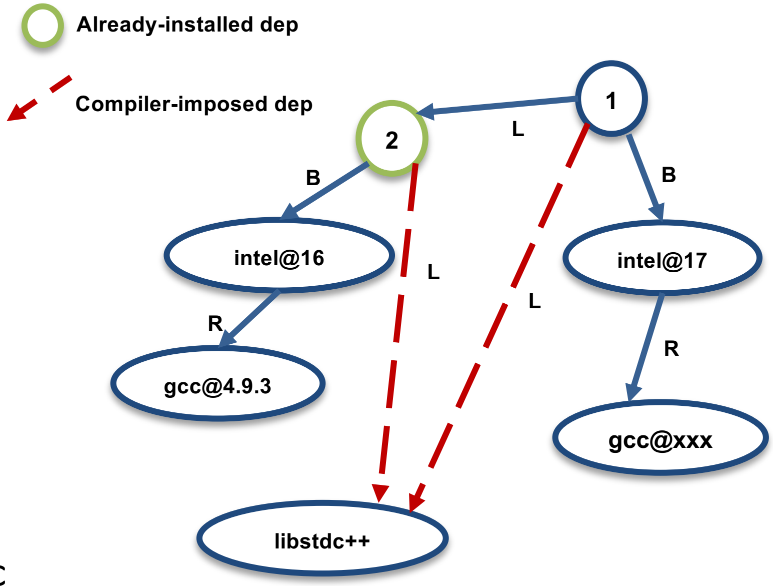
- **Getting rid of configuration in ~/.spack**
  - While *installations* may move to the home directory, *configuration* there is causing issues
  - User configuration is like an unwanted global (e.g., LD_LIBRARY_PATH 😬)
    - Interferes with CI builds (many users will `rm -rf ~/.spack` to avoid it)
    - 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 0.18 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

Join the Spack community!

- There are lots of ways to get involved!
  - Contribute packages, documentation, or features at **github.com/spack/spack**
  - Contribute your configurations to **github.com/spack/spack-configs**

- Talk to us!
  - You're already on our **Slack channel** (spackpm.herokuapp.com)
  - Join our **Google Group** (see GitHub repo for info)
  - Submit **GitHub issues** and **pull requests**!

★ Star us on GitHub!

github.com/spack/spack

Follow us on Twitter!

@spackpm

We hope to make distributing & using HPC software easy!

Lawrence Livermore National Laboratory

Disclaimer
This document was prepared as an account of work sponsored by an agency of the United States government. Neither the United States government nor Lawrence Livermore National Security, LLC, nor any of their employees makes any warranty, expressed or implied, or assumes any legal liability or responsibility for the accuracy, completeness, or usefulness of any information, apparatus, product, or process disclosed, or represents that its use would not infringe privately owned rights. Reference herein to any specific commercial product, process, or service by trade name, trademark, manufacturer, or otherwise does not necessarily constitute or imply its endorsement, recommendation, or favoring by the United States government or Lawrence Livermore National Security, LLC. The views and opinions of authors expressed herein do not necessarily state or reflect those of the United States government or Lawrence Livermore National Security, LLC, and shall not be used for advertising or product endorsement purposes.