

Managing HPC Software Complexity with Spack

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

Supercomputing 2020 Full-day Tutorial
November 9-10, 2020
Virtual Event



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




Exascale Computing Project

 SC20
Everywhere more than hpc.

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:

spackpm.herokuapp.com

Join the “tutorial” channel!

We can engage with in-depth questions more directly on Slack!

The screenshot shows the Spack documentation page on readthedocs. The header features the Spack logo and the word "Spack". Below the header is a search bar labeled "Search docs" and a dropdown menu showing "latest". The main content area is divided into sections: "LINKS" containing "Main Spack Documentation"; "TUTORIAL" containing "Basic Installation Tutorial", "Configuration Tutorial", "Package Creation Tutorial", and "Developer Workflows Tutorial"; and "Read the Docs" with a dropdown menu showing "v: latest". Further down are sections for "Versions" (listing "latest", "sc18", "sc17", "sc16", "riken19", "pearc19", "nsf19", "lanl19", "isc19", "ecp19"), "Downloads", "HTML", "On Read the Docs" (links to "Project Home", "Builds", "Downloads", "On GitHub", "View", "Edit", "Search"), and a "Search docs" bar. At the bottom, it says "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 n
and read the live de

Slides



Practice and Experi
Chicago, IL, USA.

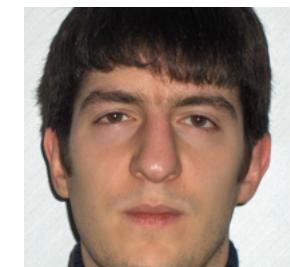
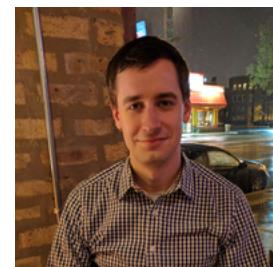
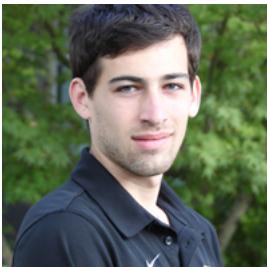
Live Demos

We provide scripts
sections in the slide

1. We provide a
tutorial on yo
the containe
2. When we ha
unfamiliar wi

You should now be

Tutorial Presenters



Todd Gamblin, Greg Becker, Peter Scheibel, Tammy Dahlgren
LLNL

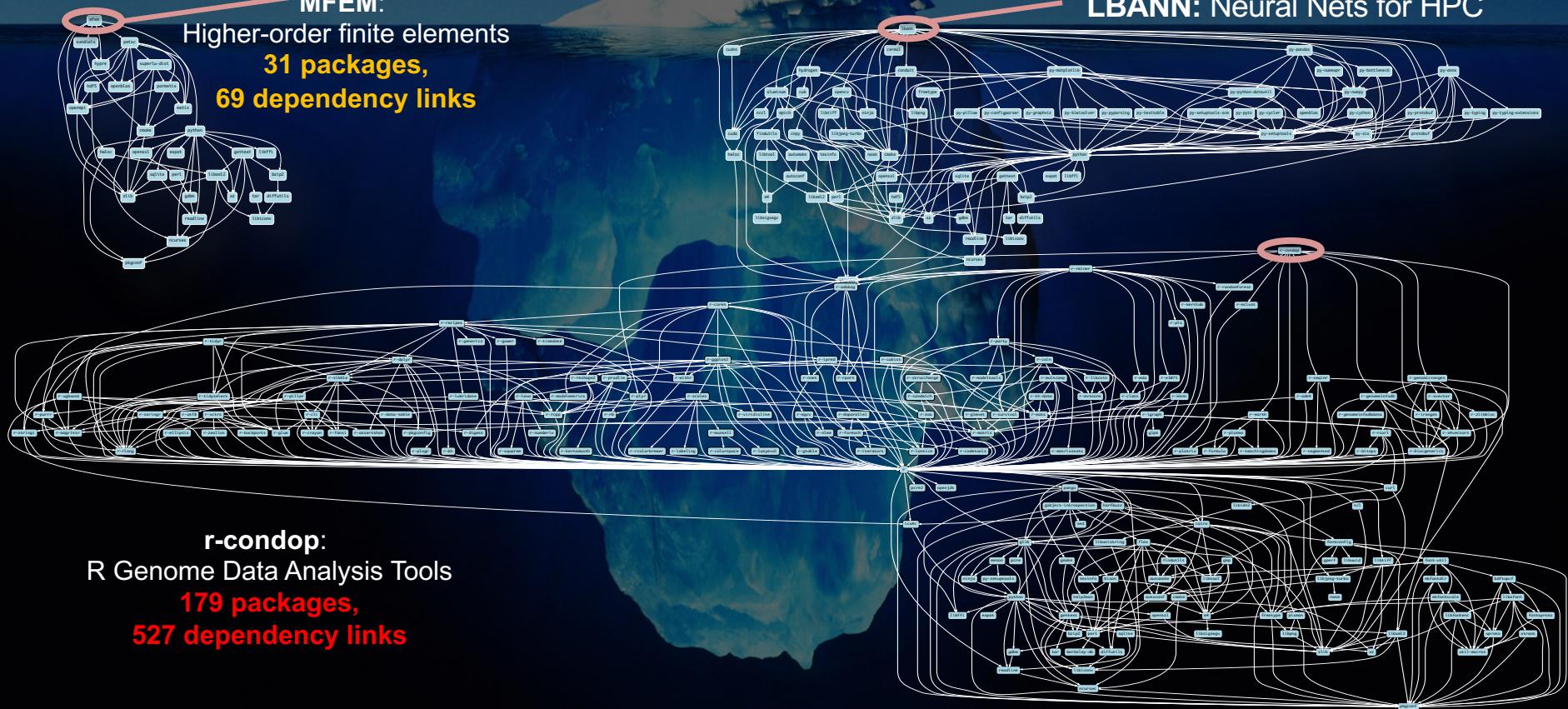
Adam Stewart
UIUC

Massimiliano
Culpo
NP Complete, S.r.l.

Modern scientific codes rely on icebergs of dependency libraries

R Genome Data Analysis Tools

71 packages
188 dependency links



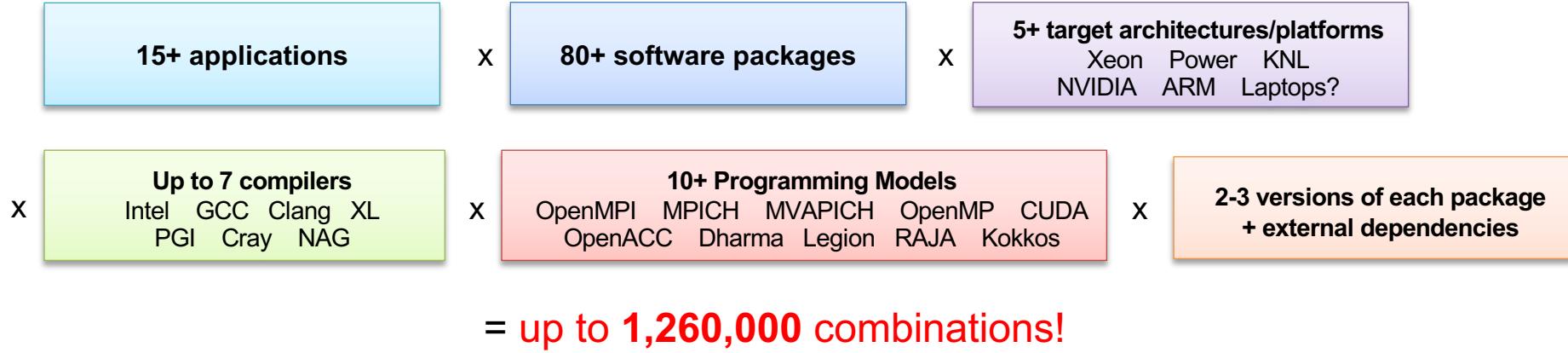
What is the “production” environment for HPC?

- Someone's home directory?
- LLNL? LANL? Sandia? ANL? LBL? TACC?
 - Environments at large-scale sites are very different
- Which MPI implementation?
- Which compiler?
- Which dependencies?
- Which versions of dependencies?
 - Many applications require specific dependency versions.



Real answer: there isn't a single production environment or a standard way to build.
Reusing someone else's software is HARD.

The complexity of the exascale ecosystem threatens productivity.



- Every application has its own stack of dependencies.
- Developers, users, and facilities dedicate (many) FTEs to building & porting.
- Often trade reuse and usability for performance.

We must make it easier to rely on others' software!

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 is a flexible package manager for HPC

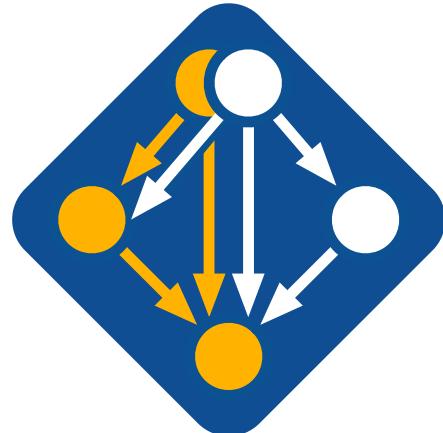
- How to install Spack:

```
$ git clone https://github.com/spack/spack
$ . spack/share/spack/setup-env.sh
```

- How to install a package:

```
$ spack install hdf5
```

- HDF5 and its dependencies are installed within the Spack directory.
- Unlike typical package managers, Spack can also install many variants of the same build.
 - Different compilers
 - Different MPI implementations
 - Different build options



github.com/spack/spack



@spackpm



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

Spack is used worldwide!

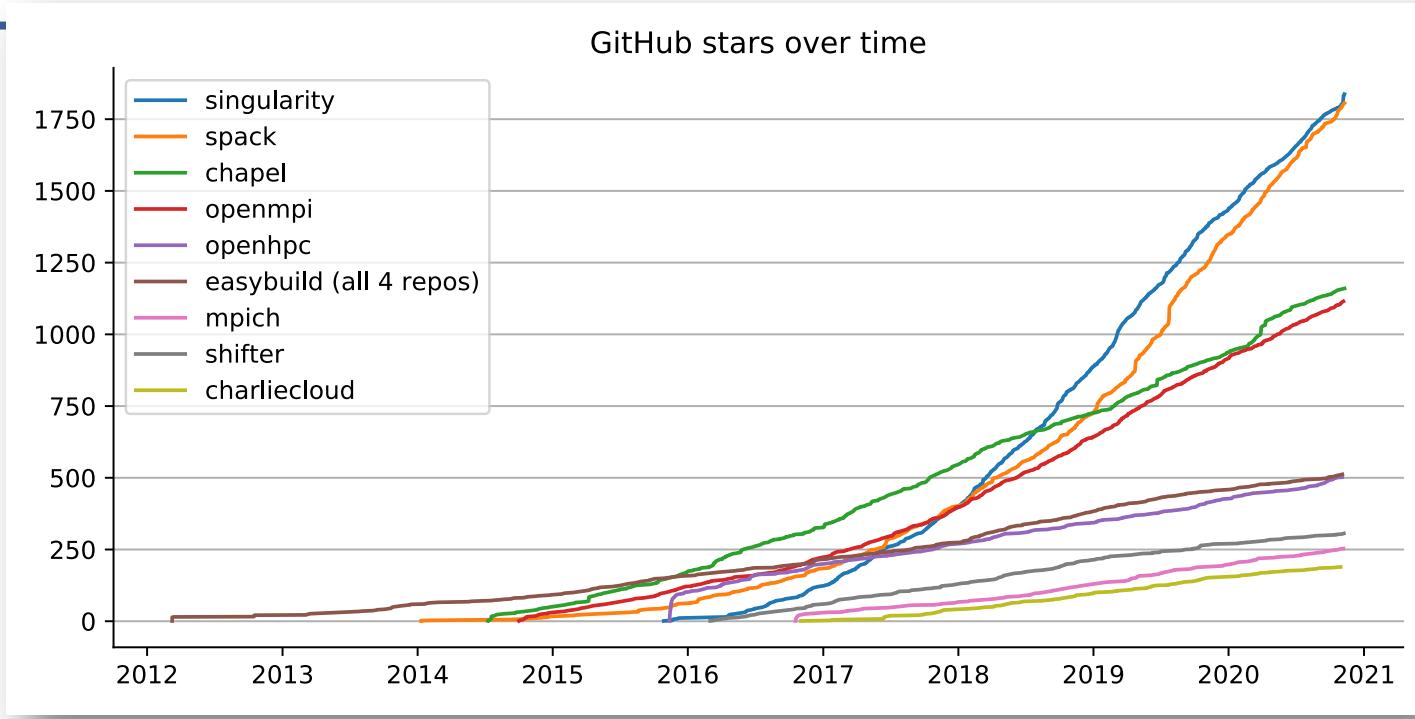
Recently surpassed **5,000** software packages
Recently surpassed **680** contributors

Over **3,200** users of Spack docs in Oct 2020
<https://spack.readthedocs.io>

Join #tutorial on Slack: spackpm.herokuapp.com

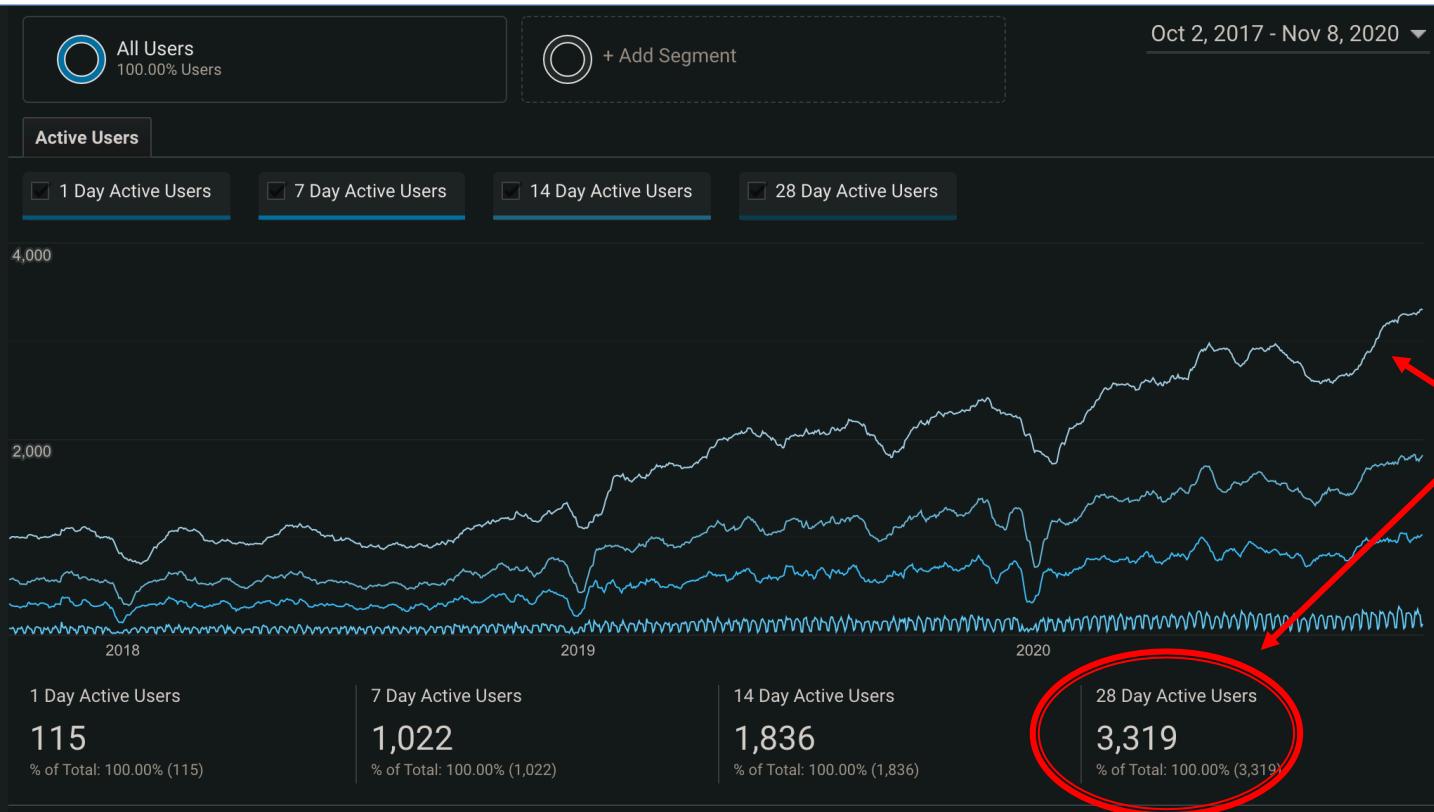
Materials: spack-tutorial.readthedocs.io

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



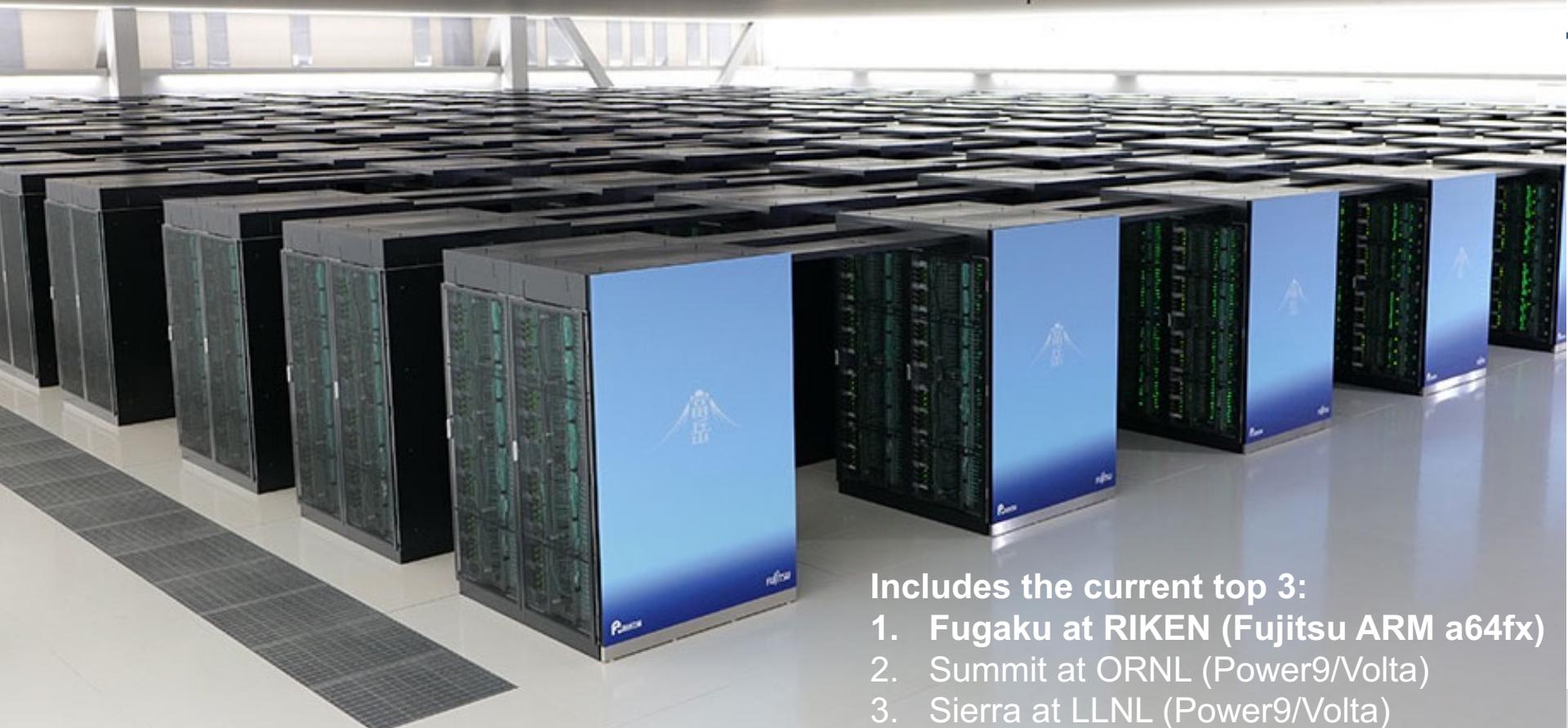
★ Star Spack at github.com/spack/spack if you like the tutorial!

Users on our documentation site have continued to increase



Broke 3,000 monthly active users for the first time.

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



- 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

The screenshot shows the homepage of the E4S Project website. At the top, there is a navigation bar with links for HOME, EVENTS, ABOUT, DOCPORTAL, POLICIES, CONTACT US, FAQ, and DOWNLOAD. The main header features the text "The Extreme-scale Scientific Software Stack" next to the E4S logo. Below the header, there is a section titled "What is E4S?" with a detailed description of the project's purpose. Further down, there are sections for "Purpose" (with an anchor icon), "Approach" (with a gear icon), "Platforms" (with a square icon), and "Testing" (with a square icon). The "Approach" section includes a link to "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." The "Platforms" section lists "Frontier", "El Capitan", and "Aurora" as supported platforms.

<https://e4s.io>

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

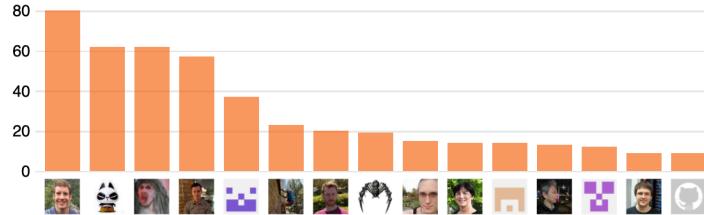
504
Merged Pull Requests

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**.



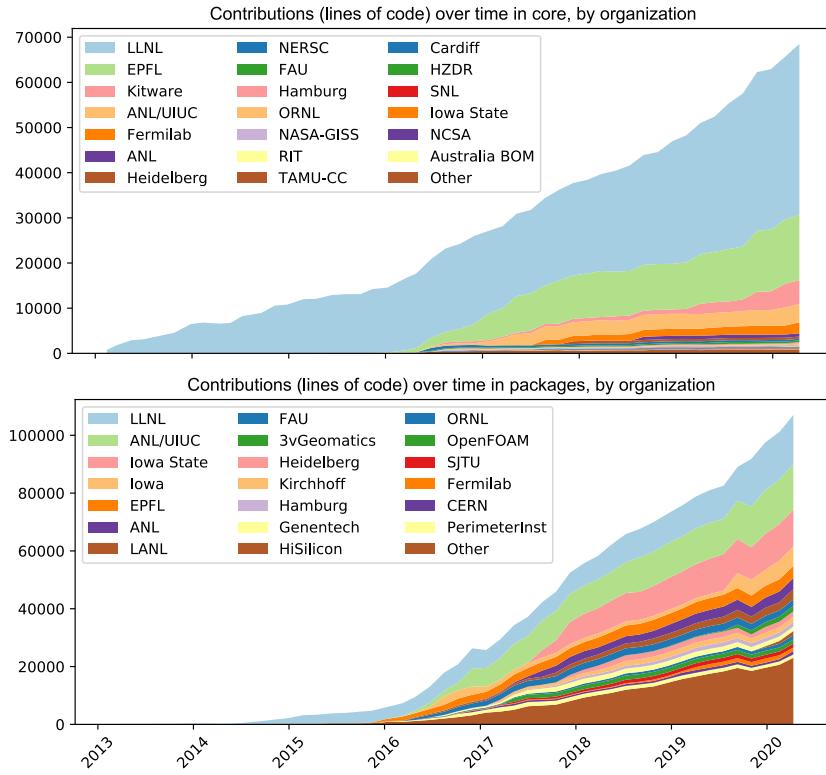
504 Pull requests merged by 132 people

Join #tutorial on Slack: spackpm.herokuapp.com

Materials: spack-tutorial.readthedocs.io



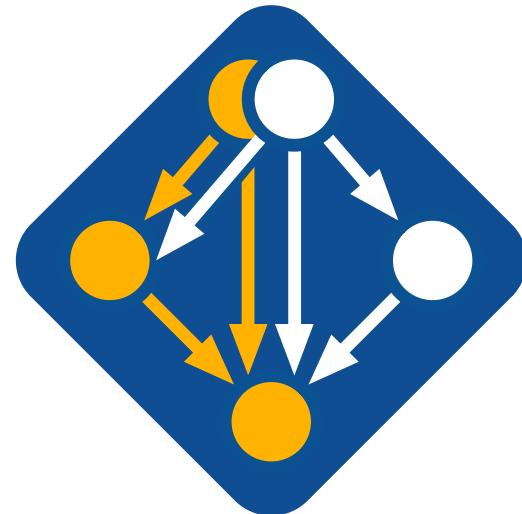
Contributions to Spack continue to grow!



- 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!

Spack 0.15 was released at the end of June

- Many new features:
 - Packages can specify how they should be found on the system
 - `spack external find` command
 - Source code mirror for all Spack packages (from 0.15.1)
 - Testing builds in GitLab for all packages in E4S
 - Better Cray support
 - Better compiler optimization support on macOS
 - `apple-clang` now its own compiler
 - Enhancements and simplification to configuration
 - `spack config add` / `spack config remove`
- Over 4,300 packages
 - 430 new since 0.14 in November 2019
- Several bugfix releases since (0.15.1, 0.15.2, 0.15.3, 0.15.4)



Spack is not the only tool that automates builds



1. “Functional” Package Managers

- Nix
- GNU Guix

<https://nixos.org/>
<https://www.gnu.org/s/guix/>

2. Build-from-source Package Managers

- Homebrew, LinuxBrew
- MacPorts
- Gentoo

<http://brew.sh>
<https://www.macports.org>
<https://gentoo.org>

Other tools in the HPC Space:



▪ 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

<http://hpcugent.github.io/easybuild/>

▪ Conda

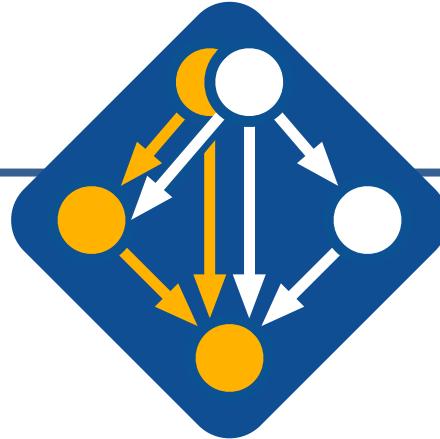
- Very popular binary package manager for data science
- Not targeted at HPC; generally has unoptimized binaries

<https://conda.io>



Join us at other events at SC!

- **CANOPIE-HPC workshop**
 - Thursday, Nov 12, 3:20-3:45pm
 - paper on archspec
 - spack's microarchitecture support library
- **Spack BOF**
 - Wednesday, Nov 18
 - 11:30am - 12:45pm EST
 - **Spack v0.16 to be announced**



SC20
Everywhere we are more than hpc.

Spack Basics

Spack provides a *spec* syntax to describe customized DAG configurations

\$ 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=skylake	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

`spack list` shows what packages are available

```
$ spack list
==> 303 packages.
activeharmony cgal fish gtkplus libgd mesa openmpi py-coverage py-pycparser qt tcl
adept-utils cgm flex harfbuzz libpgp-error metis openspeedshop py-cython py-pyelftools qthreads texinfo
apex cityhash fltk hdf libjpeg-turbo Mitos openssl py-dateutil py-pygments R the_silver_searcher
arpack cleverleaf flux hdfs libjson-c mpc otf py-epydoc py-pylint ravel thrift
asciidoc cloog fontconfig hwlloc libmng mpe2 otf2 py-funcsigs py-pypar readline tk
atk cmake freetype hybre libmonitor mpfr pango py-genders py-pyparsing rose tmux
atlas cmocka gasnet icu libNBc mpibash papi py-gnuplot py-pyqt rsync tmuxinator
atop coreutils gcc icu4c libpicaaccess mpich paraver py-hspy py-pyside ruby trilinos
autoconf cppcheck gdb ImageMagick libpng mpileaks paraview py-ipython py-pytables SAMRAI uncrustify
automated cram gdk-pixbuf isl libsodium mrnet parmetis py-libxml2 py-python-daemon samtools util-linux
automake cscope geos jdk libtiff mumps parpack py-lockfile py-ptz scalasca valgrind
bear cube gflags jemalloc libtool munge patchelf py-mako py-rpy2 scorep vim
bib2xhtml curl ghostscript jpeg libunwind muster pcre py-matplotlib py-scientificpython scotch vtk
binutils czmq git judy libuuid mvapich2 pcre2 py-mock py-scikit-learn scr wget
bison damselfly glib julia libxcb nasm pdt py-mpi4py py-scipy silo wx
boost dbus glm launchmon libxml2 ncdu petsc py-mx py-setuptools snappy wxpropgrid
bowtie2 docbook-xml global lcms libxshmfence ncurses pidx py-mysqldb1 py-shiboken sparsehash xcb-proto
boxlib doxygen glog leveldb libxslt netcdf pixman py-nose py-sip spindle xerces-c
bzip2 dri2proto glpk libarchive llvm netgauge pkg-config py-numexpr py-six spot xz
cairo dtcmpl gmp libcerf llvm-lld netlib-blas pmgr_collective py-numpy py-sphinx sqlite yasm
callpath dyninst gmsk libcircle lmdb netlib-lapack postgresql py-pandas py-sympy stat zeromq
cblas eigen gnuplot libdrm lmod netlib-scalapack ppl py-pbr py-periodictable py-twisted sundials zlib
cbtf elfutils gnutls libdwarf lua nettle protobuf py-pexpect py-urwid szip tar
cbtf-argonavis elpa gperf libedit lwgrp ninja py-astropy py-baseemap py-pil py-virtualenv taskd tau
cbtf-krell expat gperf tools libelf lwm2 ompss py-basemap py-biopython py-pillow py-yapf
cbtf-lanl extrae graphlib libevent matio ompt-openmp py-blessings py-pmw python
cereal exuberant-ctags graphviz libffi mbedTLS opari2 py-cffi py-pychecker qhull
cfitsio fftw gsl libgcrypt memaxes openblas
```

- Spack has over 5,000 packages now.

`spack find` shows what is installed

```
$ spack find
==> 103 installed packages.
-- linux-rhel6-x86_64 / gcc@4.4.7 -----
ImageMagick@6.8.9-10 glib@2.42.1 libtiff@4.0.3 pango@1.36.8 qt@4.8.6
SAMRAI@3.9.1 graphlib@2.0.0 libtool@2.4.2 parmetis@4.0.3 qt@5.4.0
adept-utils@1.0 gtkplus@2.24.25 libxcb@1.11 pixman@0.32.6 ravel@1.0.0
atk@2.14.0 harfbuzz@0.9.37 libxml2@2.9.2 py-dateutil@2.4.0 readline@6.3
boost@1.55.0 hdf5@1.8.13 llvm@3.0 py-ipython@2.3.1 scotch@6.0.3
cairo@1.14.0 icu@54.1 metis@5.1.0 py-nose@1.3.4 starpu@1.1.4
callpath@1.0.2 jpeg@9a mpich@3.0.4 py-numumpy@1.9.1 stat@2.1.0
dyninst@8.1.2 libdwarf@20130729 ncurses@5.9 py-pytz@2014.10 xz@5.2.0
dyninst@8.1.2 libelf@0.8.13 ocr@2015-02-16 py-setup-tools@11.3.1 zlib@1.2.8
fontconfig@2.11.1 libffi@3.1 openssl@1.0.1h py-six@1.9.0
freetype@2.5.3 libmng@2.0.2 otf@1.12.5salmon python@2.7.8
gdk-pixbuf@2.31.2 libpng@1.6.16 otf2@1.4 qhull@1.0

-- linux-rhel6-x86_64 / gcc@4.8.2 -----
adept-utils@1.0.1 boost@1.55.0 cmake@5.6-special libdwarf@20130729 mpich@3.0.4
adept-utils@1.0.1 cmake@5.6 dyninst@8.1.2 libelf@0.8.13 openmpi@1.8.2

-- linux-rhel6-x86_64 / intel@14.0.2 -----
hwloc@1.9 mpich@3.0.4 starpu@1.1.4

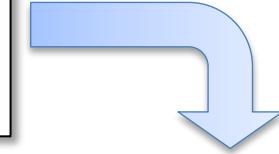
-- linux-rhel6-x86_64 / intel@15.0.0 -----
adept-utils@1.0.1 boost@1.55.0 libdwarf@20130729 libelf@0.8.13 mpich@3.0.4

-- linux-rhel6-x86_64 / intel@15.0.1 -----
adept-utils@1.0.1 callpath@1.0.2 libdwarf@20130729 mpich@3.0.4
boost@1.55.0 hwloc@1.9 libelf@0.8.13 starpu@1.1.4
```

- All the versions coexist!
 - Multiple versions of same package are ok.
- Packages are installed to automatically find correct dependencies.
- Binaries work *regardless of user's environment*.
- Spack also generates module files.
 - Don't have to use them.

Users can query the full dependency configuration of installed packages.

```
$ spack find callpath  
==> 2 installed packages.  
-- linux-rhel6-x86_64 / clang@3.4 --      -- linux-rhel6-x86_64 / gcc@4.9.2 -----  
callpath@1.0.2
```



Expand dependencies with `spack find -d`

```
$ spack find -dl callpath  
==> 2 installed packages.  
-- linux-rhel6-x86_64 / clang@3.4 -----  
xv2clz2    callpath@1.0.2  
ckjazss     ^adept-utils@1.0.1  
3ws43m4      ^boost@1.59.0  
ft7znm6      ^mpich@3.1.4  
qqnuet3     ^dyninst@8.2.1  
3ws43m4      ^boost@1.59.0  
g65rdud     ^libdwarf@20130729  
cj5p5fk     ^libelf@0.8.13  
cj5p5fk     ^libelf@0.8.13  
g65rdud     ^libdwarf@20130729  
cj5p5fk     ^libelf@0.8.13  
cj5p5fk     ^libelf@0.8.13  
ft7znm6     ^mpich@3.1.4
```



```
-- linux-rhel6-x86_64 / gcc@4.9.2 -----  
udltshs    callpath@1.0.2  
rfsu7fb     ^adept-utils@1.0.1  
ybet64y     ^boost@1.55.0  
aa4ar6i     ^mpich@3.1.4  
tmmnge5    ^dyninst@8.2.1  
ybet64y     ^boost@1.55.0  
g2mxrl2    ^libdwarf@20130729  
ynpai3j     ^libelf@0.8.13  
ynpai3j     ^libelf@0.8.13  
g2mxrl2    ^libdwarf@20130729  
ynpai3j     ^libelf@0.8.13  
ynpai3j     ^libelf@0.8.13  
aa4ar6i     ^mpich@3.1.4
```

- Architecture, compiler, versions, and variants may differ between builds.

Spack manages installed compilers

- Compilers are automatically detected
 - Automatic detection determined by OS
 - Linux: PATH
 - Cray: `module avail`
- Compilers can be manually added
 - Including Spack-built compilers

```
$ spack compilers
==> Available compilers
-- gcc -----
gcc@4.2.1      gcc@4.9.3

-- clang -----
clang@6.0
```

compilers.yaml

```
compilers:
- compiler:
  modules: []
  operating_system: ubuntu14
  paths:
    cc: /usr/bin/gcc/4.9.3/gcc
    cxx: /usr/bin/gcc/4.9.3/g++
    f77: /usr/bin/gcc/4.9.3/gfortran
    fc: /usr/bin/gcc/4.9.3/gfortran
    spec: gcc@4.9.3
- compiler:
  modules: []
  operating_system: ubuntu14
  paths:
    cc: /usr/bin/clang/6.0/clang
    cxx: /usr/bin/clang/6.0/clang++
    f77: null
    fc: null
    spec: clang@6.0
- compiler:
  ...
```

Hands-on Time: Spack Basics

Follow script at spack-tutorial.readthedocs.io

Core Spack Concepts

Join #tutorial on Slack: spackpm.herokuapp.com

Materials: spack-tutorial.readthedocs.io

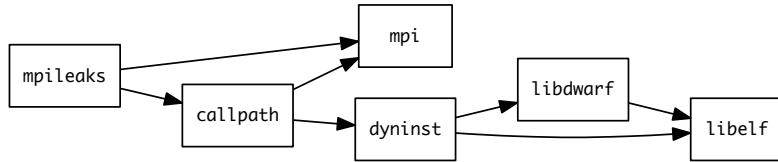


Most existing tools do not support combinatorial versioning

- 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

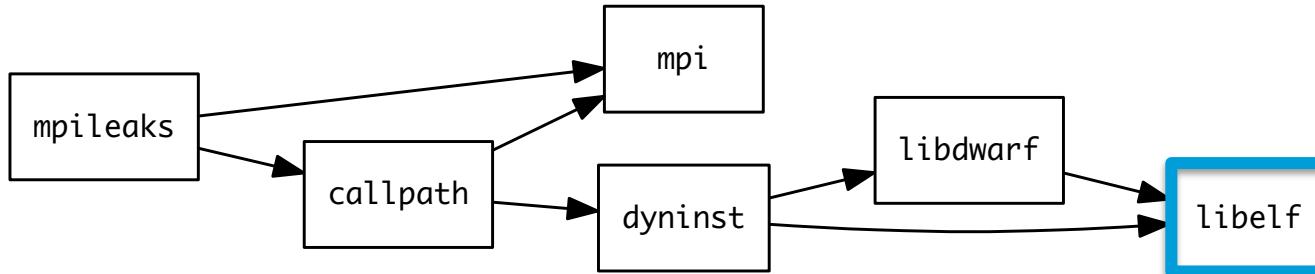


Installation Layout

```
opt
└── spack
    ├── darwin-mojave-skylake
    │   └── clang-10.0.0-apple
    │       ├── bzip2-1.0.8-hc4sm4vuzpm4znmvrfzri4ow2mkphe2e
    │       ├── python-3.7.6-daqqpsssxb6qbfrztsezkmhus3xoflbsy
    │       ├── sqlite-3.30.1-u64v26igvxyn23hysmk1fums6tgjv5r
    │       ├── xz-5.2.4-u5eawkvaoc7vonabe6nndkcfwuv233cj
    │       └── zlib-1.2.11-x46q4wm46ay4pltrijbgizxjrhbaka6
    └── darwin-mojave-x86_64
        └── clang-10.0.0-apple
            └── coreutils-8.29-p12kcytejqcys5dzecfrtjqxfdssvnob
```

- 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*

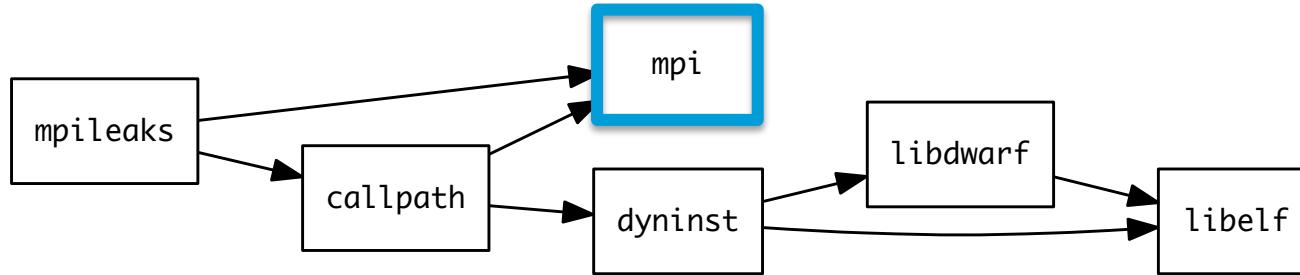
Spack Specs can constrain versions of dependencies



```
$ 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* 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:
```

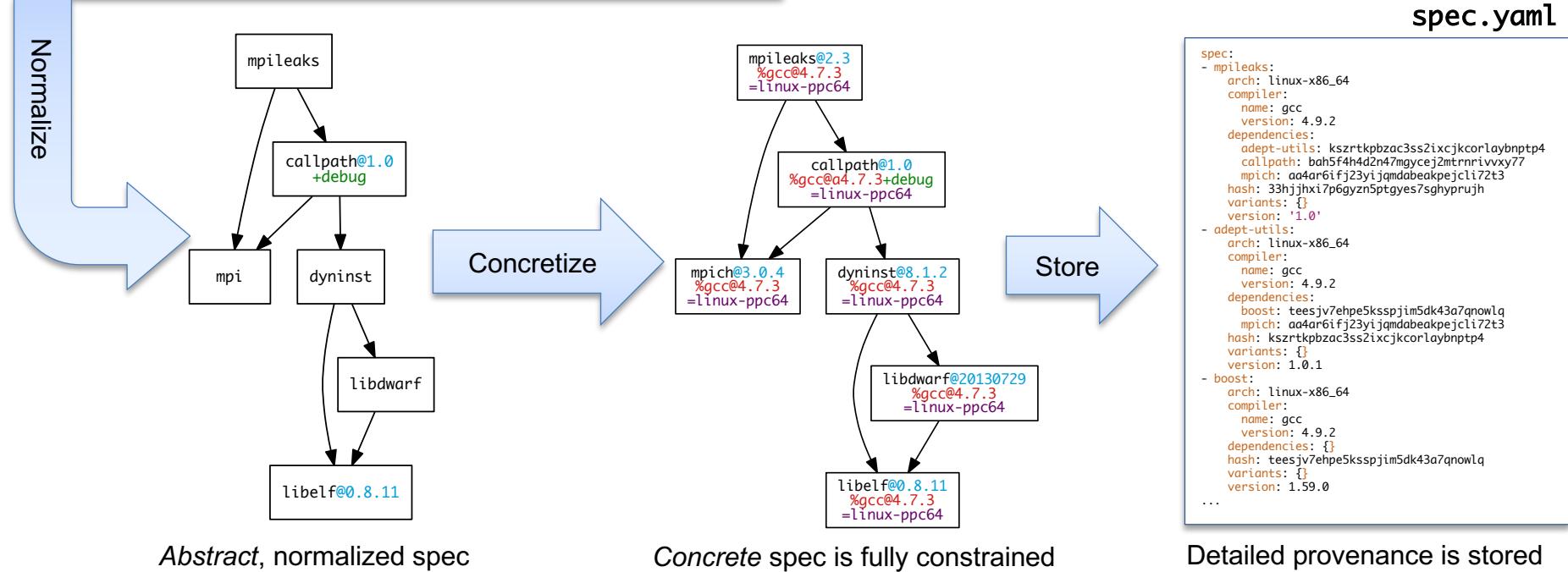
- Let Spack choose MPI implementation, as long as it provides MPI 2 interface:

```
$ 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*

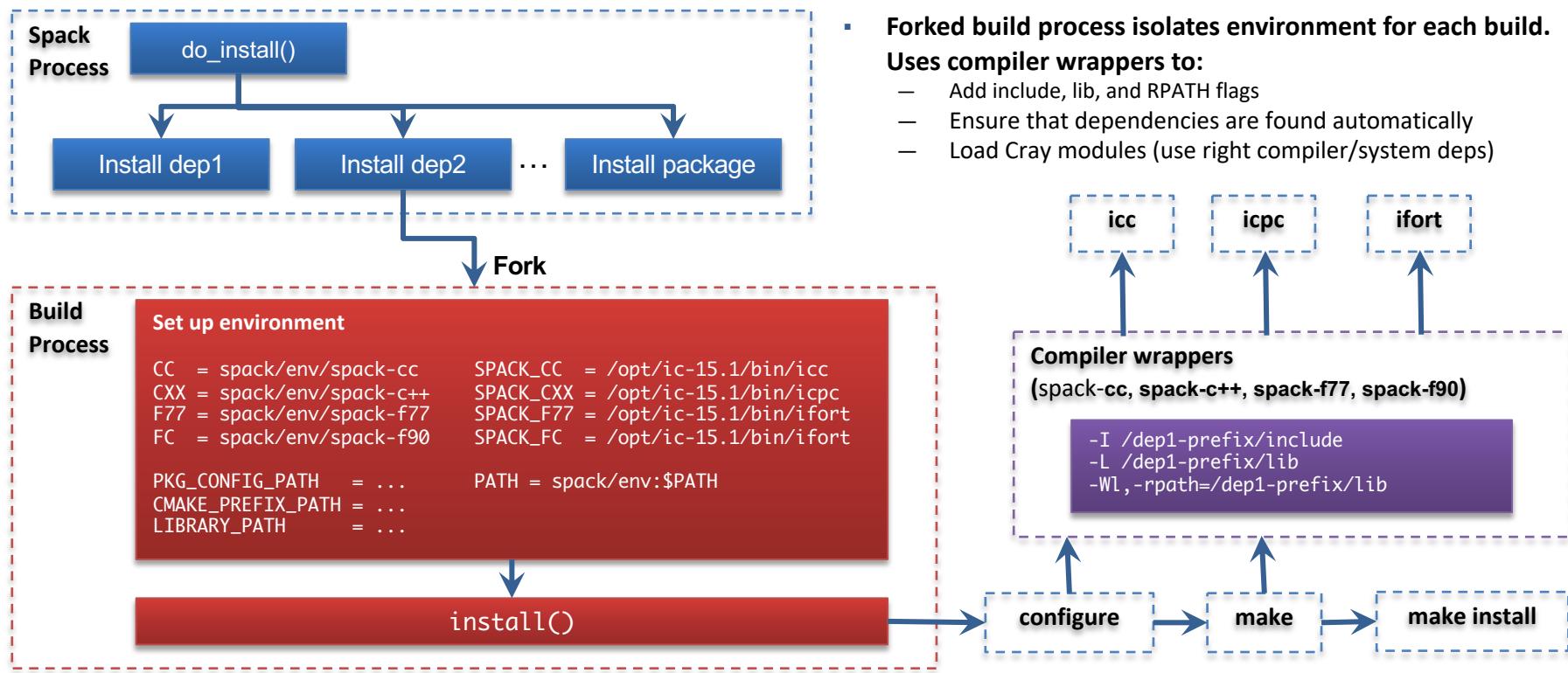


Use `spack spec` to see the results of concretization

```
$ 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
        ^bzzip2@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~cxxm~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



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
- Automatically activate for correct version of dependency
 - Provenance information from DAG
 - Activate all dependencies that are extensions as well...

```
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/
```

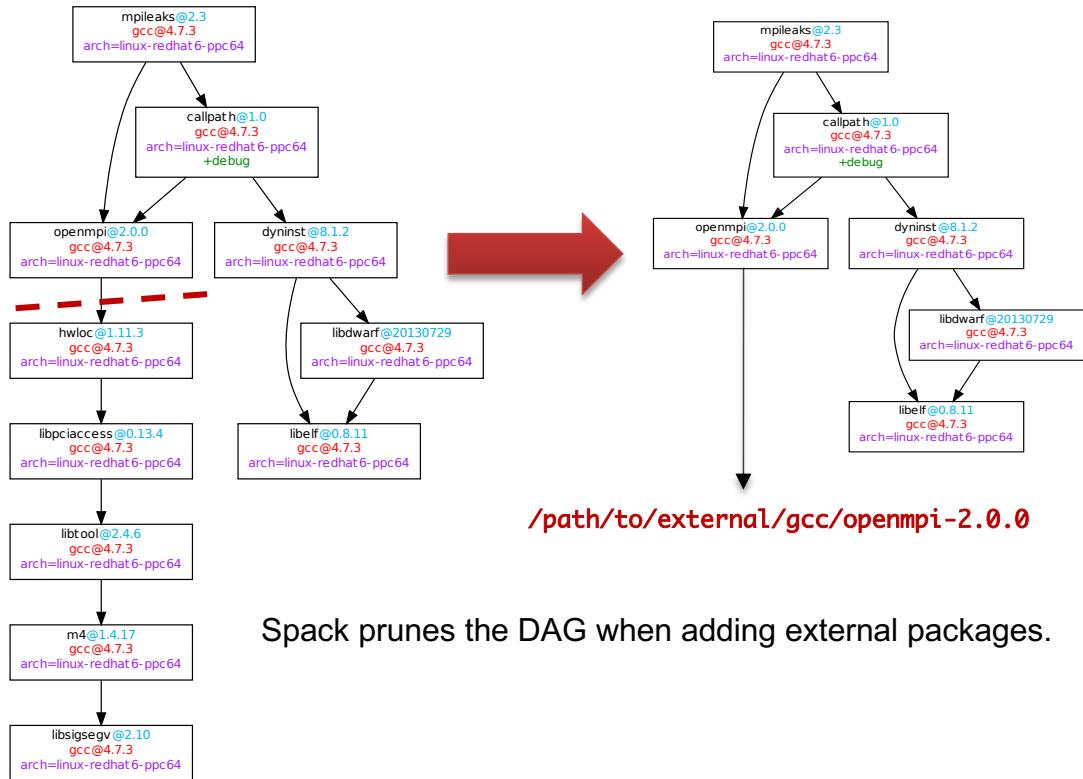


Building against externally installed software

```
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).

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)

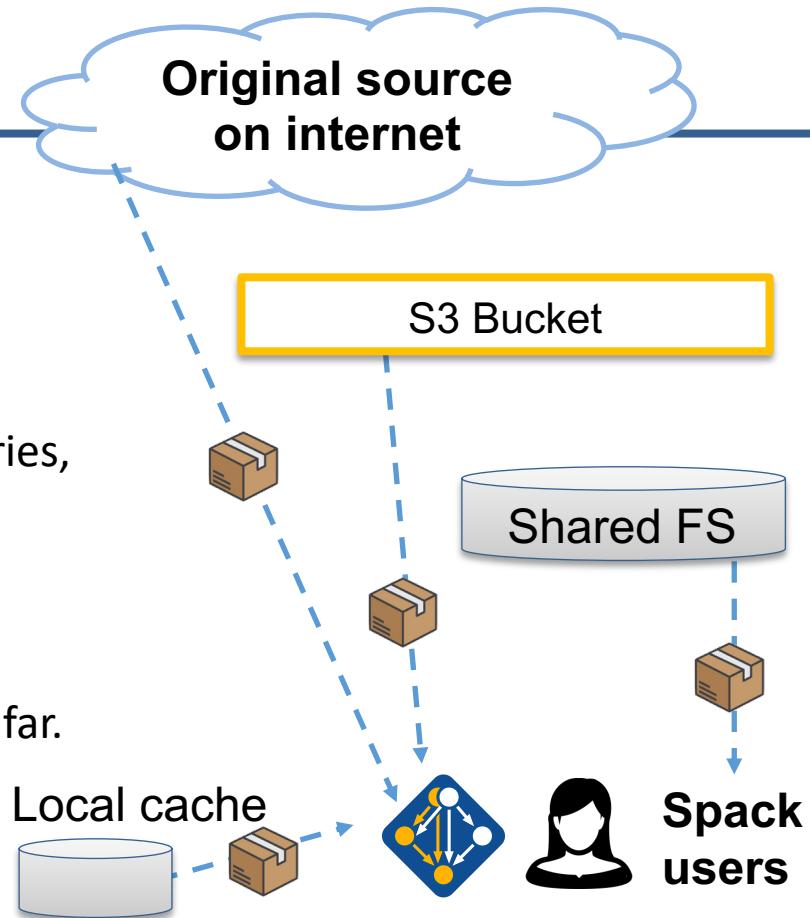
```
$ spack repo create /path/to/my_repo  
$ spack repo add my_repo  
$ spack repo list  
==> 2 package repositories.  
my_repo      /path/to/my_repo  
builtin      spack/var/spack/repos/builtin
```

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



Hands-on Time: Configuration

Follow script at spack-tutorial.readthedocs.io

Making your own Spack Packages

Join #tutorial on Slack: spackpm.herokuapp.com

Materials: spack-tutorial.readthedocs.io



Hands-on Time: Creating Packages

Follow script at spack-tutorial.readthedocs.io

Day 2

Spack Review

Join #tutorial on Slack: spackpm.herokuapp.com

Materials: spack-tutorial.readthedocs.io



Spack provides a *spec* syntax to describe customized DAG configurations

\$ 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=skylake	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

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/minimal 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"

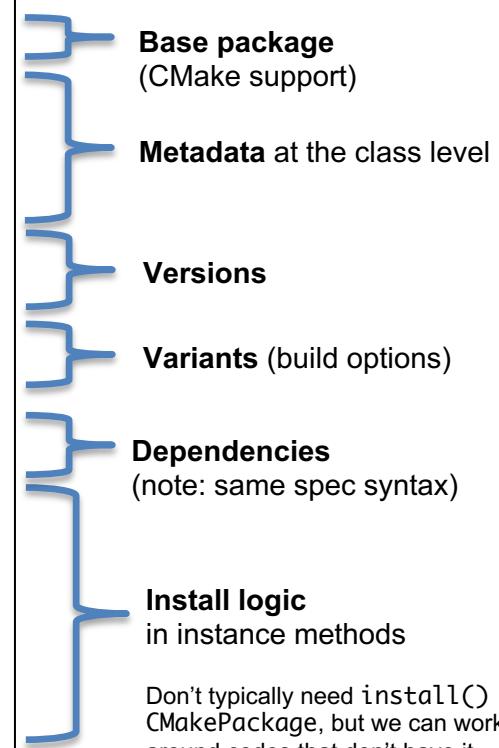
    version('1.2.3', sha256='3f7f2eef0d1ba5825780d626741eb0b3f026a096048d7ec4794d2a7dfbe2b8a6')
    version('1.2.2', sha256='eaf9ddf562416974157b34d00c3a1c880fc5296fce2aa2efa039a86e0976f3a3')
    version('1.1', sha256='232d74072fc7b848fa2adc8a1bc839ae8fb5f96d50224186601f55554a25f64a')

    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')

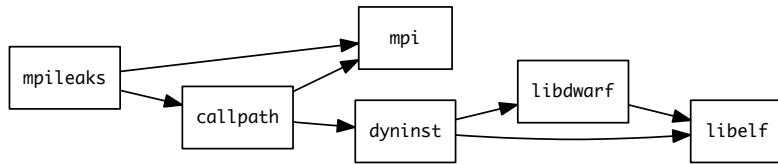
    def cmake_args(self):
        return [
            '-DENABLE_OPENMP=%s' % ('+openmp' in self.spec),
            '-DENABLE_MPI=%s' % ('+mpi' in self.spec),
        ]

    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)
```



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-daqqpsssxb6qbfrztsezkmhus3xoflbsy
    │       ├── sqlite-3.30.1-u64v26igvxyn23hysmk1fums6tgjv5r
    │       ├── xz-5.2.4-u5eawkvaoc7vonabe6nndkcfwuv233cj
    │       └── zlib-1.2.11-x46q4wm46ay4pltrijbgizxjrhbaka6
    └── darwin-mojave-x86_64
        └── clang-10.0.0-apple
            └── coreutils-8.29-p12kcytejqcys5dzecfrtjqxfdssvnob
```

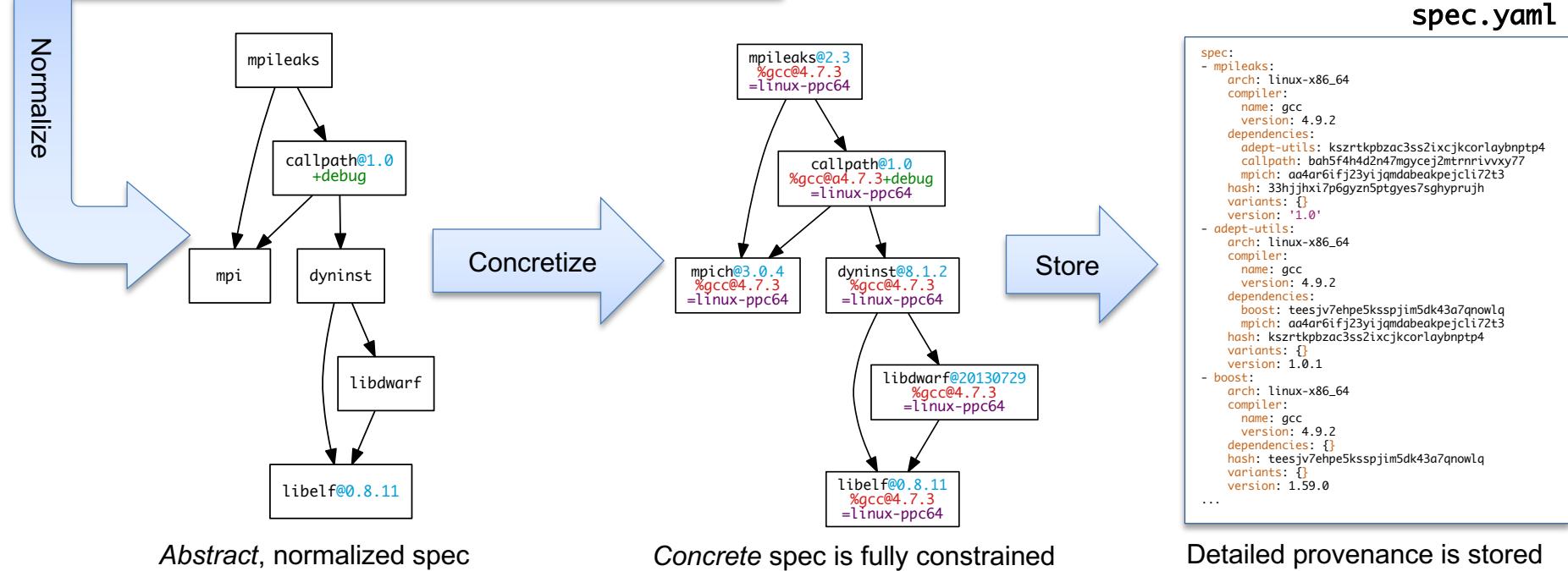
Hash

- 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*

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*



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

Join #tutorial on Slack: spackpm.herokuapp.com

Materials: spack-tutorial.readthedocs.io

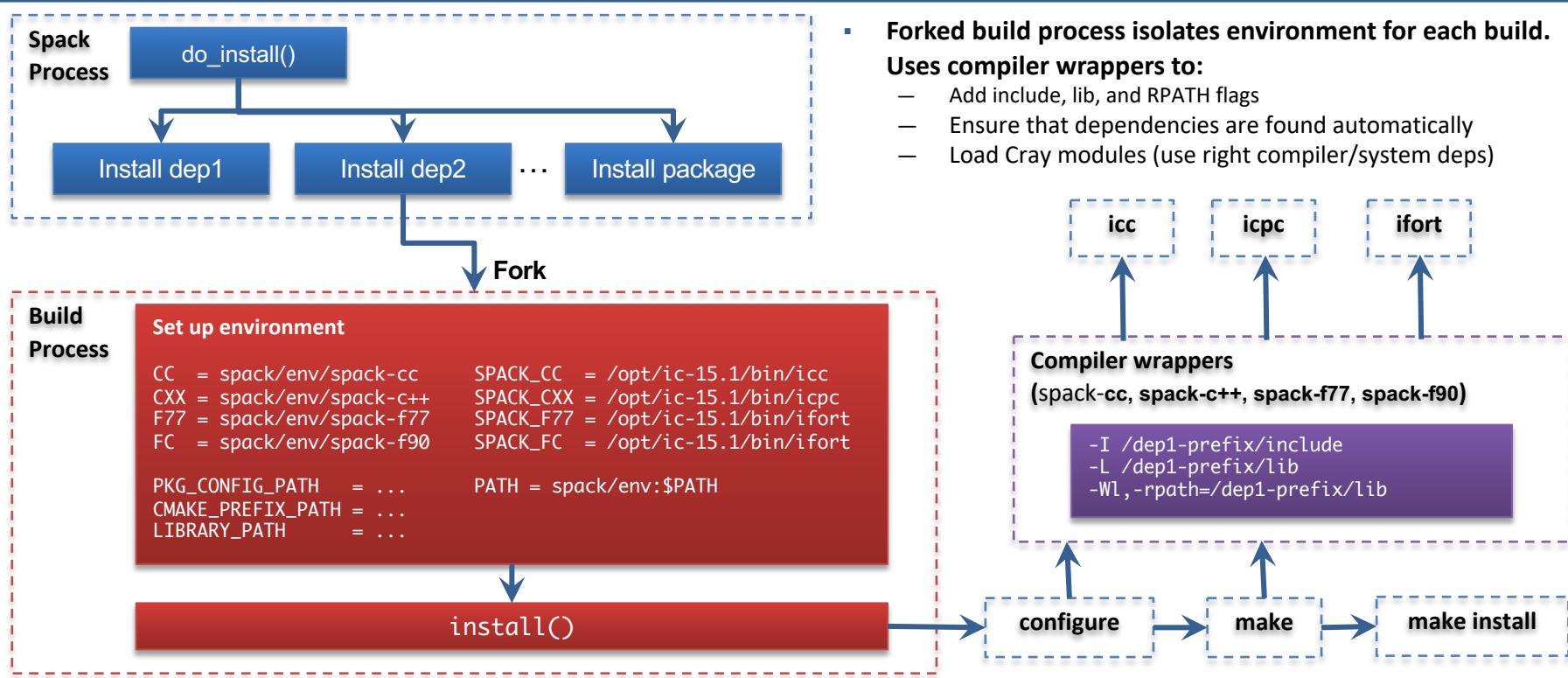


Use `spack spec` to see the results of concretization

```
$ 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
        ^bzzip2@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~cxxm~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



Environments, spack.yaml and spack.lock

Follow script at spack-tutorial.readthedocs.io

Spack Stacks

Follow script at spack-tutorial.readthedocs.io

Hands-on Time: Environment Modules

Follow script at spack-tutorial.readthedocs.io

Developer Workflows

Follow script at spack-tutorial.readthedocs.io

Scripting and spack-python

Follow script at spack-tutorial.readthedocs.io

More New Features and the Road Ahead

Join #tutorial on Slack: spackpm.herokuapp.com

Materials: spack-tutorial.readthedocs.io



Spack understands specific target microarchitectures

- We have developed a cross-platform library to detect and compare microarchitecture metadata
 - Detects based on /proc/cpuinfo (Linux), sysctl (Mac)
 - Allows comparisons for compatibility, e.g.:

```
skylake > broadwell  
zen2 > x86_64
```

- Key features:
 - Know which compilers support which chips/which flags
 - Determine compatibility
 - Enable creation and reuse of optimized binary packages
 - Easily query available architecture features for portable build recipes
- We will be extracting this as a standalone library for other tools & languages
 - Hope to make this standard!

```
$ spack arch --known-targets  
Generic architectures (families)  
aarch64 ppc64 ppc64le x86 x86_64  
  
IBM - ppc64  
power7 power8 power9  
  
IBM - ppc64le  
power8le power9le  
  
AuthenticAMD - x86_64  
barcelona bulldozer piledriver steamroller excavator zen zen2  
  
GenuineIntel - x86_64  
nocona westmere haswell mic_knl cascadelake  
core2 sandybridge broadwell skylake_avx512 icelake  
nehalem ivybridge skylake cannonlake  
  
GenuineIntel - x86  
i686 pentium2 pentium3 pentium4 prescott
```



Extensive microarchitecture knowledge

```
class OpenBlas(Package):  
  
    def configure_args(self, spec):  
        args = []  
        if 'avx512' in spec.target:  
            args.append('--with=avx512')  
        ...  
        return args
```

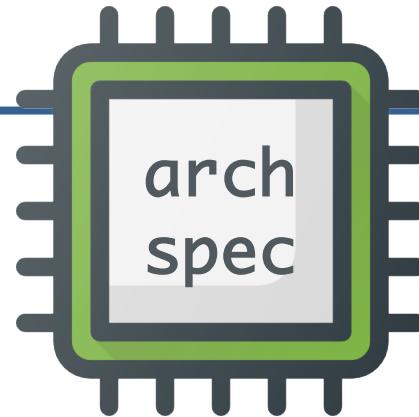
Simple feature query

```
$ spack install lbann target=cascadelake  
$ spack install petsc target=zen2
```

Specialized installations

Check out Archspec!

- Standalone library, extracted from Spack
- Use fine-grained, human-readable labels, e.g.:
 - broadwell, haswell, skylake
 - instead of x86_64, aarch64, ppc64 etc.
- Query capabilities and more
 - “Does haswell support AVX-512?” “no.”
 - “How do I compile for broadwell withicc?”



 github.com/archspec

ReadTheDocs: archspec.readthedocs.io

License: Apache 2.0 OR MIT

pip3 install archspec

archspec will be presented Thursday at the CANOPIE-HPC workshop!

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  
  RUN mkdir /opt/spack-environment  
  RUN (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 unnecessary deps  
  RUN cd /opt/spack-environment && spack install && spack gc --  
  # 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_en  
  
  # Update the system  
  RUN yum update -y && yum install -y epel-release && yum update -y  
  RUN rm -rf /var/cache/yum && yum clean all  
  
RUN echo 'export PS1="\[$(tput bold)\]\[$(tput setaf 1)\][gromacs]\[$(tput setaf 2)\]\u\[$(tpu
```



spack containerize

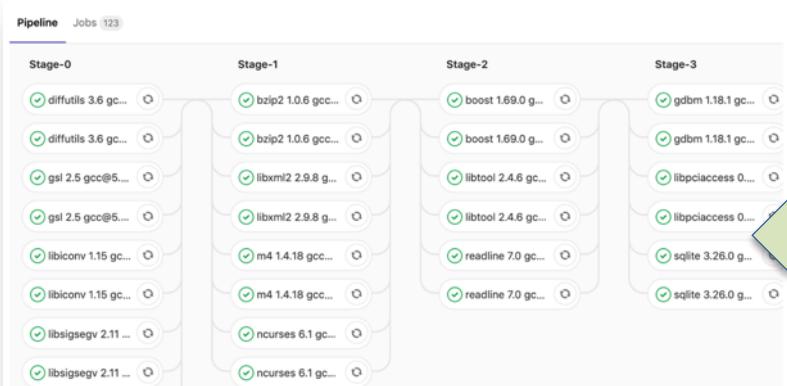
Join #tutorial on Slack: spackpm.herokuapp.com

Materials: spack-tutorial.readthedocs.io



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



spack ci

```
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
```



Join #tutorial on Slack: spackpm.herokuapp.com

Materials: spack-tutorial.readthedocs.io



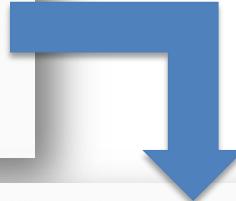
spack external find

```
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



```
packages:
  cmake:
    paths:
      cmake@3.15.1: /usr/local
```

packages.yaml configuration

- Spack has 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 0.15.1 introduced a source code mirror for all packages

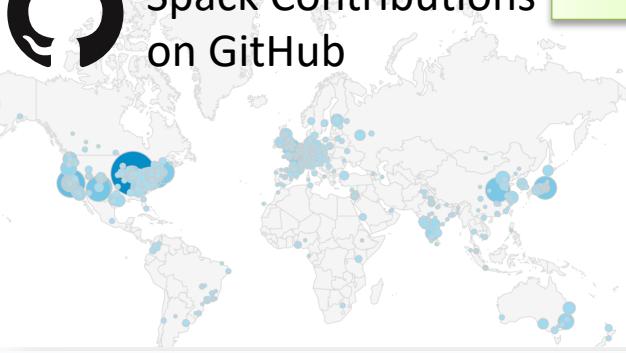
- Spack has long been vulnerable to unreliable hosted code
 - Outages at large sites like Sourceforge or GNU would leave Spack essentially down too.
 - Outages have been known to persist for long periods of time (GNU).
- Spack now has a dedicated S3 bucket with *all* sources, patches, and other resources needed to build
 - This is the first place we'll download things now
- **If other sites go down, Spack is still in business.**



We are testing our development branch and our release branches with ECP's E4S software stack



Spack Contributions
on GitHub



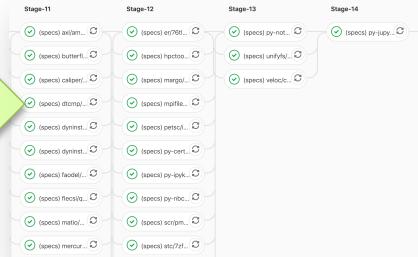
✓ ci/gitlab/gitlab.spack.io — Pipeline passed on GitLab



gitlab.spack.io

```
spec:
  - name: spack
    version: 0.13.2
    source_type: git
    source: https://github.com/spack/spack.git
    patchlevel: 0
    patch:
      - https://github.com/spack/spack/pull/10000.patch
    build:
      - ./configure
      - make
    install:
      - make install
  - name: boost
    version: 1.65.1
    source_type: git
    source: https://github.com/boostorg/boost.git
    patchlevel: 0
    patch:
      - https://github.com/boostorg/boost/pull/10000.patch
    build:
      - ./bootstrap.sh --prefix=$PREFIX
      - ./b2 --prefix=$PREFIX
    install:
      - ./b2 install
```

spack ci



E4S spack.yaml
configuration

GitLab CI builds (changed) packages

- On every pull request
- On every release branch

- E4S is about 250 packages – will eventually include all ECP software products
 - We've started with E4S built for CentOS6, CentOS7, Ubuntu18
 - Will start expanding to include more packages from Spack, ECP facility platforms, more instances
- More on E4S at <https://e4s.io>

Spack v0.16 roadmap: `spack test` allows tests to be written directly in Spack packages

```
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

Save source code from the build

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



SC20
Everywhere more
we are than hpc.

Spack v0.16 roadmap: experimental new concretizer

- Used Clingo, the Potassco grounder/solver package
- ASP program has 2 parts:
 1. Large list of facts generated from our package repositories
 - 6,000 – 9,000 facts is typical – includes dependencies, options, etc.
 2. Small logic program (~130 lines)
- New algorithm (at least our part) is conceptually simpler:
 - Generate facts for all possible dependencies
 - Send facts and our logic program to the solver
 - Build a DAG from the results
- New concretizer can so far solve many situations that current concretizer can't
 - Backtracking is a win
 - Still requires an external solver
 - We'll make it experimental until 0.17.0, when we will figure out how best to automatically include an ASP or SMT solver

```
%-----  
% Package: ucx  
%-----  
version_declared("uctx", "1.6.1", 0).  
version_declared("uctx", "1.6.0", 1).  
version_declared("uctx", "1.5.2", 2).  
version_declared("uctx", "1.5.1", 3).  
version_declared("uctx", "1.5.0", 4).  
version_declared("uctx", "1.4.0", 5).  
version_declared("uctx", "1.3.1", 6).  
version_declared("uctx", "1.3.0", 7).  
version_declared("uctx", "1.2.2", 8).  
version_declared("uctx", "1.2.1", 9).  
version_declared("uctx", "1.2.0", 10).  
  
variant("uctx", "thread_multiple").  
variant_single_value("uctx", "thread_multiple").  
variant_default_value("uctx", "thread_multiple", "False").  
variant_possible_value("uctx", "thread_multiple", "False").  
variant_possible_value("uctx", "thread_multiple", "True").  
  
declared_dependency("uctx", "numactl", "build").  
declared_dependency("uctx", "numactl", "link").  
node("numactl") :- depends_on("uctx", "numactl"), node("uctx").  
  
declared_dependency("uctx", "rdma-core", "build").  
declared_dependency("uctx", "rdma-core", "link").  
node("rdma-core") :- depends_on("uctx", "rdma-core"), node("uctx").  
  
%-----  
% 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 HDF5 package

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 complex ABI issues

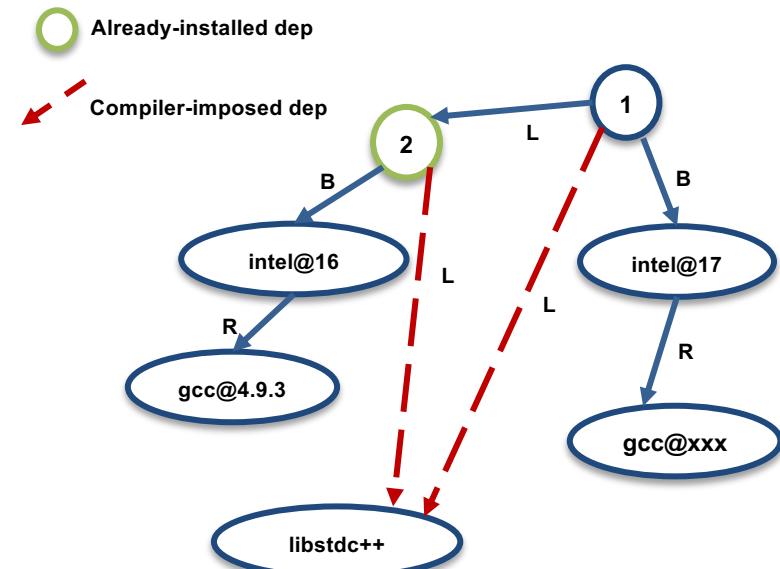
- libstdc++, libc++ compatibility
 - Compilers that depend on compilers

- Future GPU, OpenMP target, etc. libraries have similar issues

- Entire stack for a large code needs to be consistent
 - We currently do not have visibility into what's under the compiler

- Packages that depend on languages

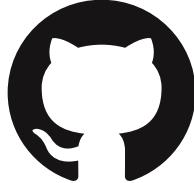
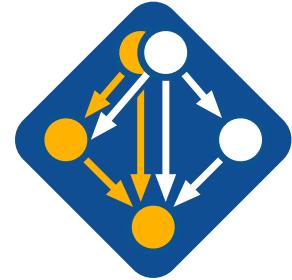
- Depend on `cxx@2011`, `cxx@2017`, `fortran@1995`, etc.
 - Model languages, openmp, cuda, etc. as virtuals



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!
 - Join our **Slack channel** (spackpm.herokuapp.com)
 - Join our **Google Group** (see GitHub repo for info)
 - Submit GitHub issues and talk to us!



★ Star us on GitHub!
github.com/spack/spack



Follow us on Twitter!
[@spackpm](https://twitter.com/spackpm)

We hope to make distributing & using HPC software easy!