

# Managing HPC Software Complexity with Spack

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

Supercomputing 2019 Full-day Tutorial  
November 18, 2018  
Dallas, Texas



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**

 **Sylabs.io**

 **NERSC**



 **ECP**  
EXTENSIBLE COMPUTING PROJECT

 **SC19**

# Tutorial Materials

Download the latest version of slides and handouts at:

[\*\*spack-tutorial.readthedocs.io\*\*](https://spack-tutorial.readthedocs.io)

For more:

- Spack website: [spack.io](https://spack.io)
- Spack GitHub repository: [github.com/spack/spack](https://github.com/spack/spack)
- Spack Reference Documentation: [spack.readthedocs.io](https://spack.readthedocs.io)

The screenshot shows the Spack website header with the Spack logo and the word "Spack" in large white text on a blue background. Below the header is a search bar labeled "Search docs". The main content area is dark grey and lists various links and tutorials. On the right side, there is a sidebar with links to "Docs", "Tutorial: Spack", and "Practice and Experience". Below the sidebar, there is a section for "Slides" with a thumbnail image and a section for "Live Demos" with a list of topics.

**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: Spack

## Tutorial: Spack

This is a full-day interactive tutorial on Spack. You can use these materials to prepare for the tutorial or read the live documentation.

### Slides

Managing HPC Software Complexity with Spack

### Practice and Experience

Chicago, IL, USA.

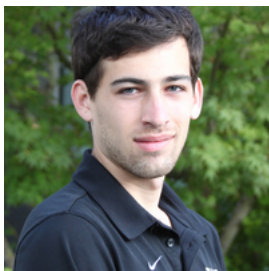
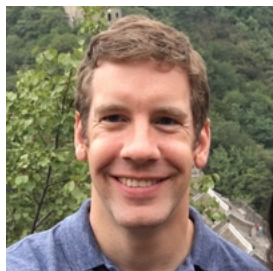
### Live Demos

We provide scripts and sections in the slides.

1. We provide a tutorial on your container.
2. When we have unfamiliar with

You should now be

# Tutorial Presenters



Todd Gamblin, Greg Becker, Peter Scheibel  
LLNL



Mario Melara  
NERSC

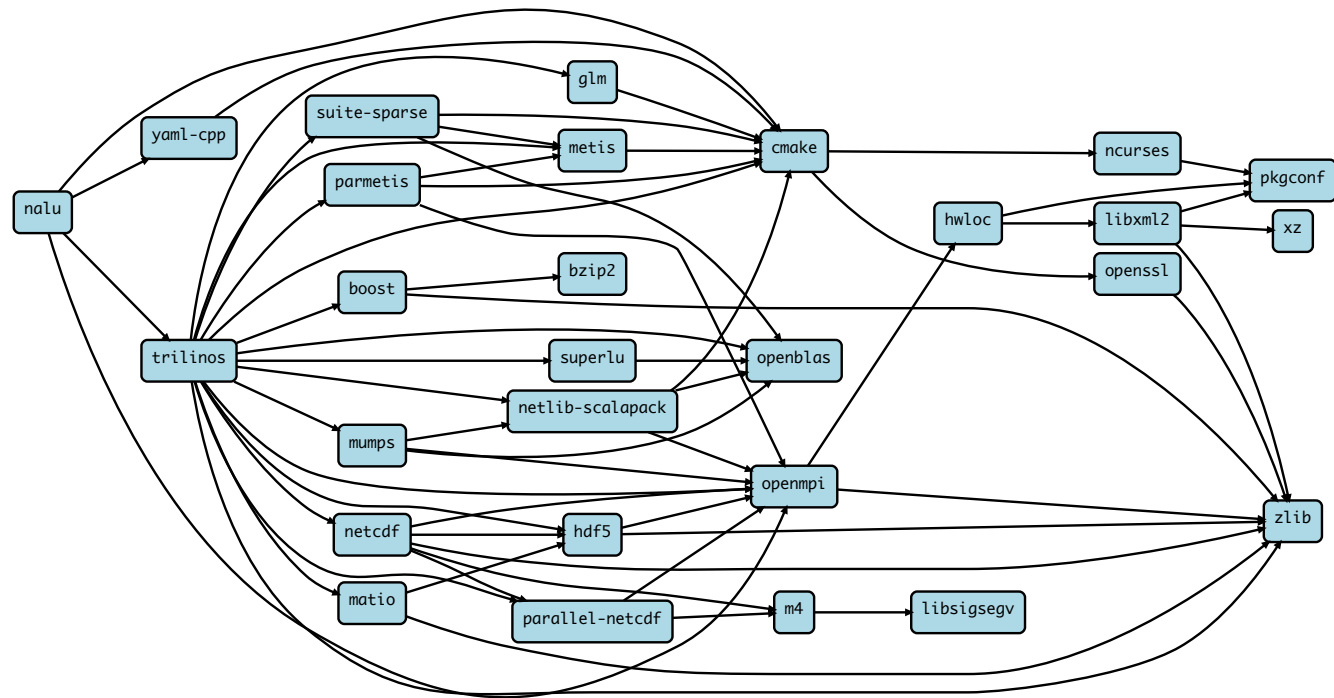


Adam Stewart  
UIUC



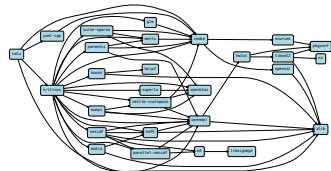
Massimiliano Culp  
Sylabs, Inc.

## Software complexity in HPC is growing

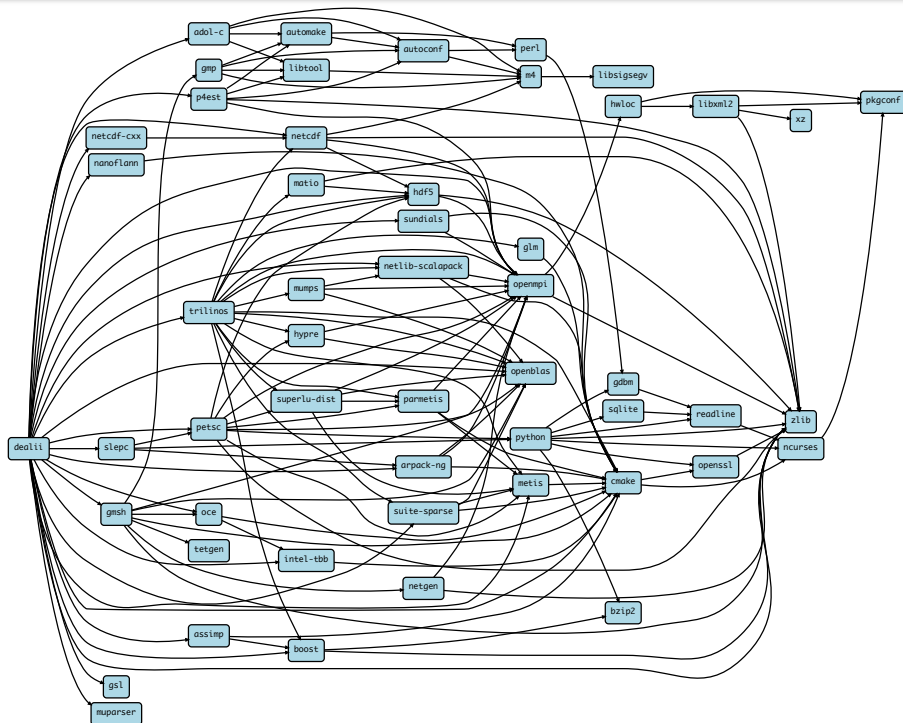


# Nalu: Generalized Unstructured Massively Parallel Low Mach Flow

# Software complexity in HPC is growing



Nalu: Generalized Unstructured Massively Parallel Low Mach Flow



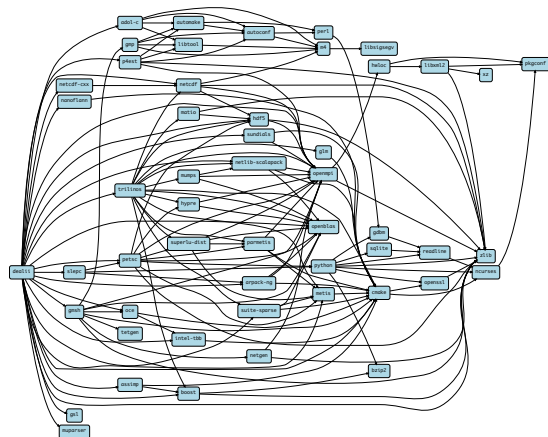
## dealii: C++ Finite Element Library

Follow along at [spack-tutorial.readthedocs.io](https://spack-tutorial.readthedocs.io)

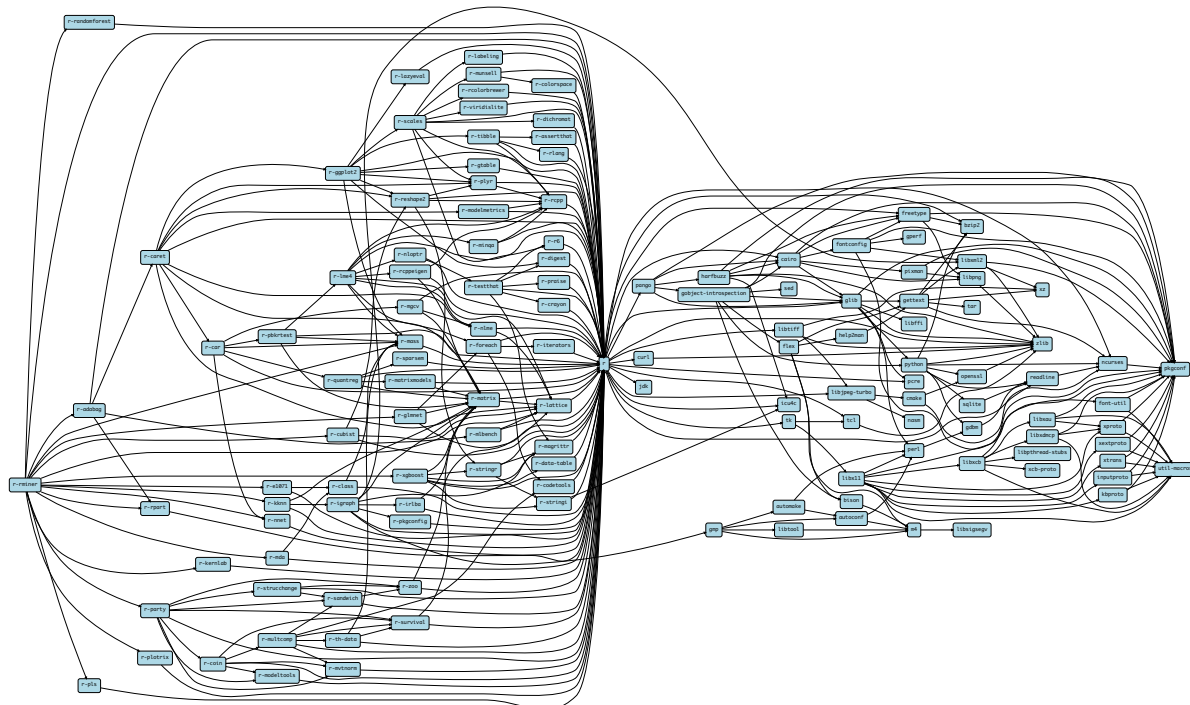
# Software complexity in HPC is growing



Nalu: Generalized Unstructured Massively Parallel Low Mach Flow



dealii: C++ Finite Element Library



R Miner: R Data Mining Library

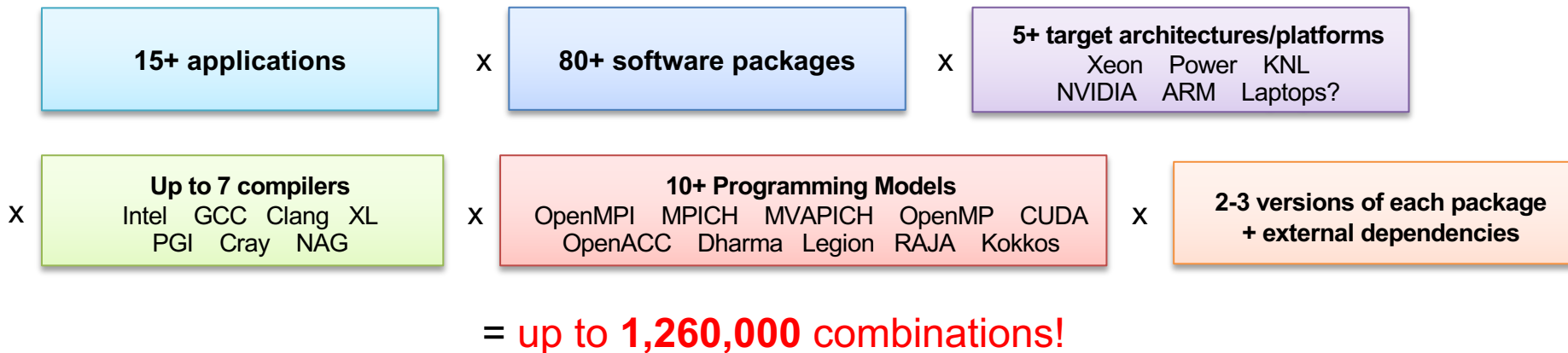
# What is the “production” environment for HPC?

- 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

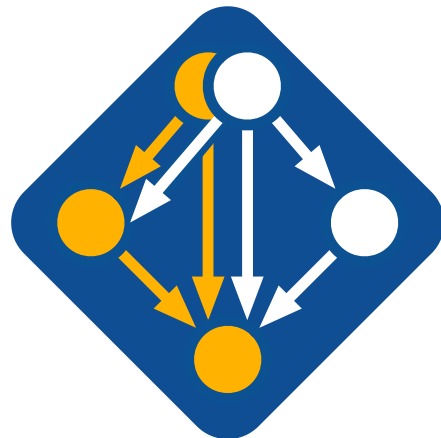
- How to install Spack:

```
$ 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](https://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

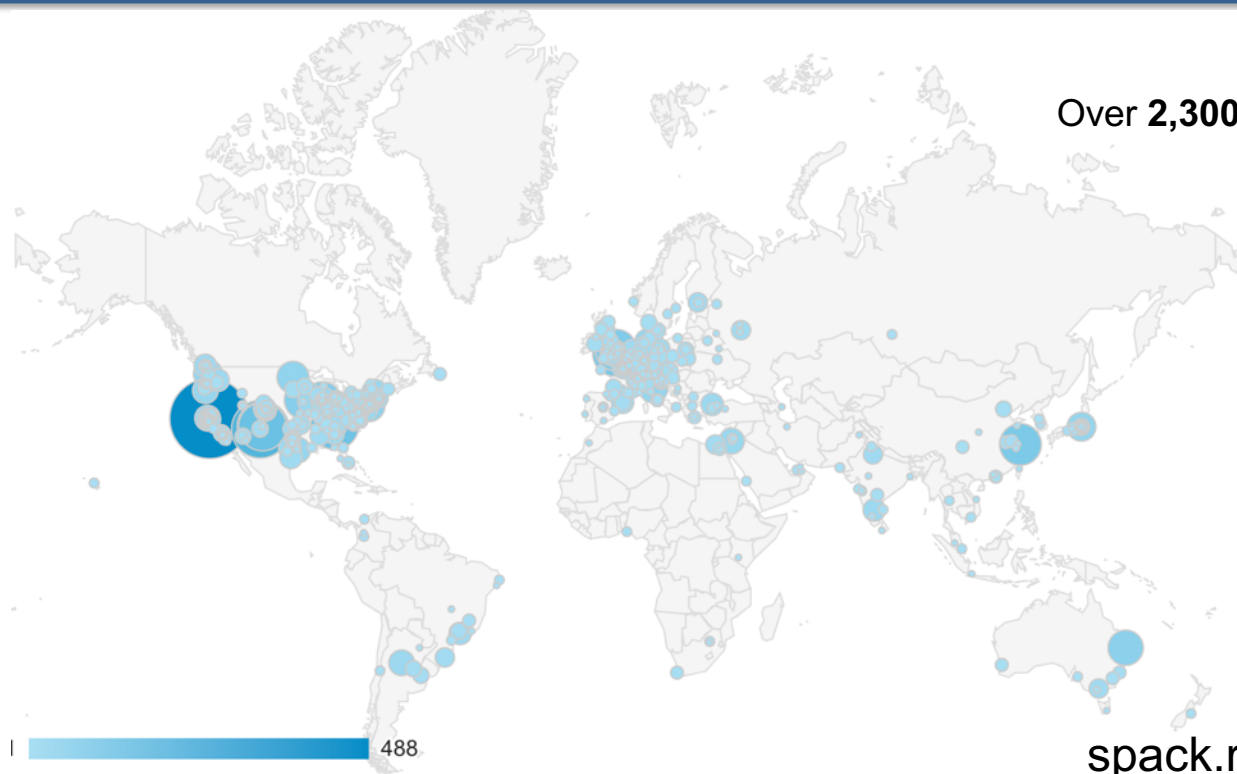
### 3. Package Developers

- 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!

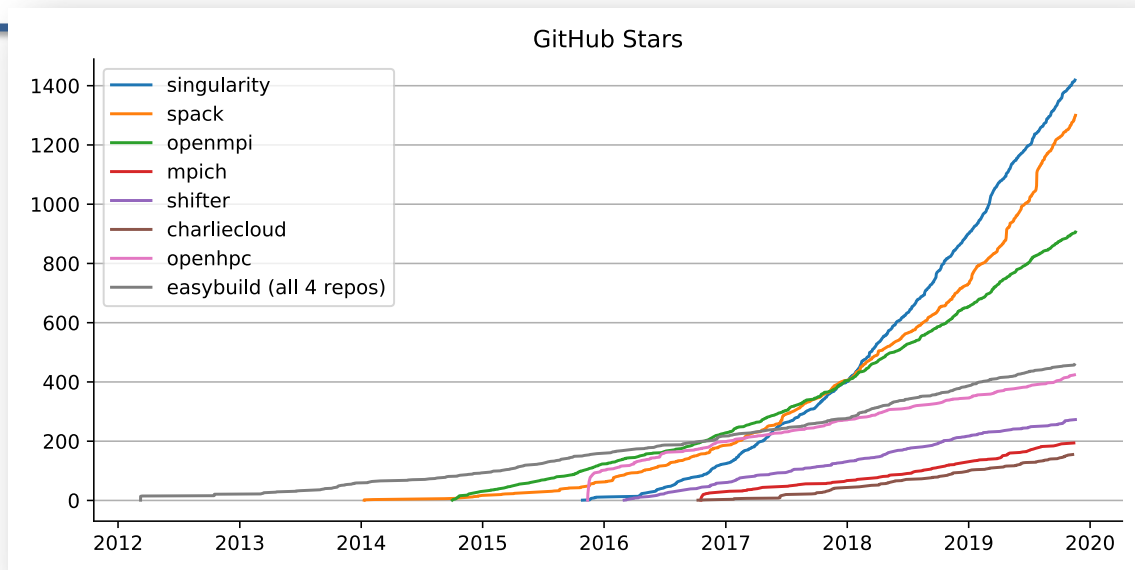


Over **3,500** software packages  
Over **2,300** monthly active users (on docs site)

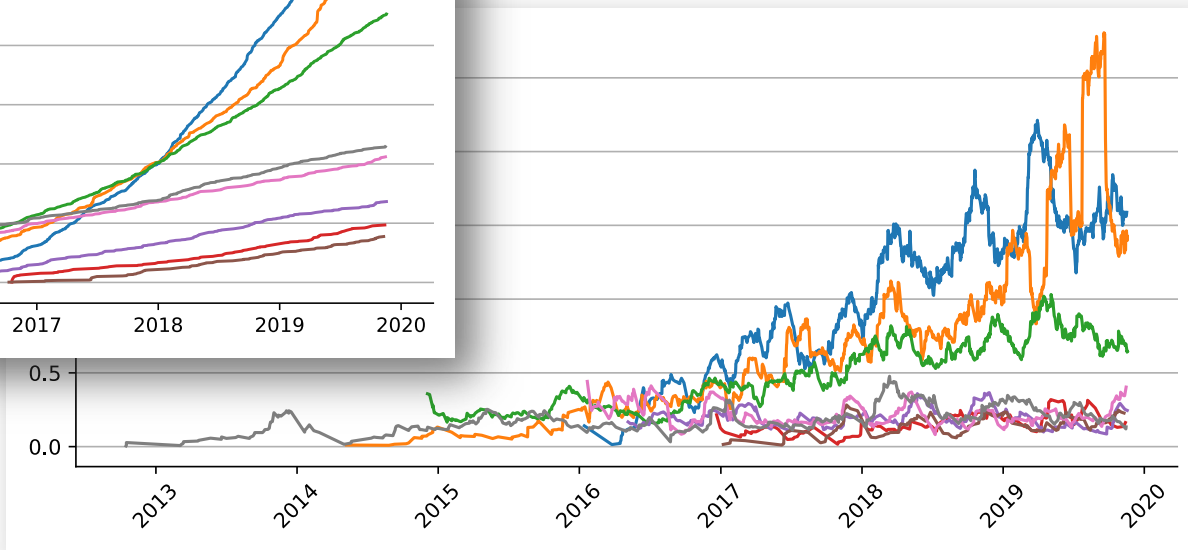
Over **450** contributors  
from labs, academia, industry

Plot shows sessions on  
spack.readthedocs.io for one month

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

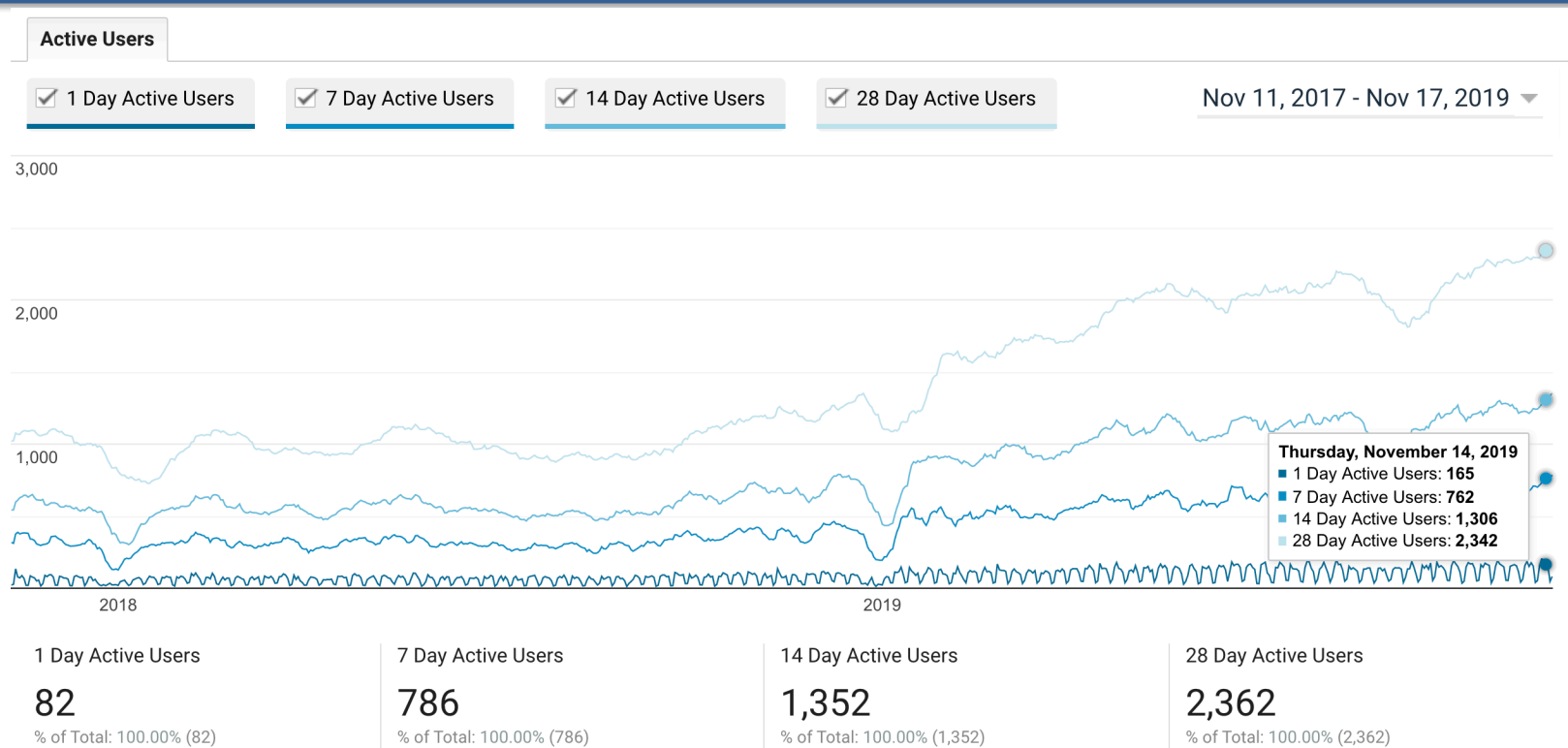


## Stars over time



Stars per day  
(same data, 60-day window)

# Users on our documentation site have also been increasing



# Spack is being used on many of the top HPC systems

- Official deployment tool for the U.S. Exascale Computing Project
- 7 of the top 10 supercomputers
- High Energy Physics community
  - Fermilab, CERN, collaborators
- Astra (Sandia)
- Fugaku (Japanese National Supercomputer Project)



Fugaku coming to RIKEN in 2021  
DOE/MEXT collaboration



Summit (ORNL), Sierra (LLNL)

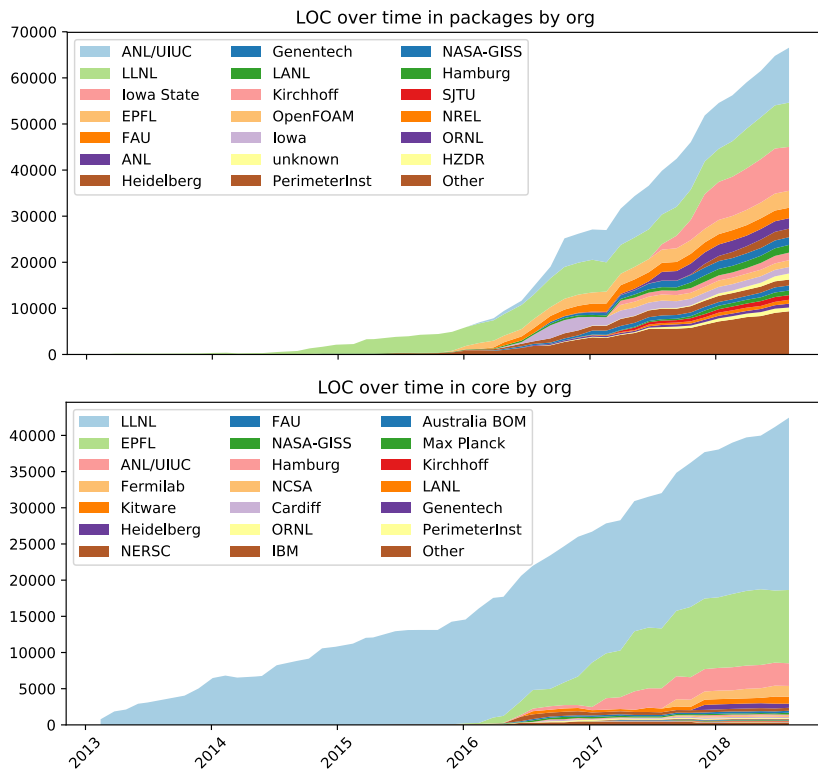


SuperMUC-NG (LRZ, Germany)



Edison, Cori, Perlmutter (NERSC)

# 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 3,500 packages
- Most packages are from external contributors!
- Many contributions in core, as well.
- We are committed to sustaining Spack's open source ecosystem!

# Spack v0.13.1 is the latest release

- **Major new features:**

1. Chaining: use dependencies from external "upstream" Spack instances
2. Views for Spack environments (covered today)
3. Spack detects and builds *specifically* for your microarchitecture (not shown in tutorial)
  - named, understandable targets like skylake, broadwell, power9, zen2
4. Spack stacks: combinatorial environments for facility deployment (covered today)
5. Projections: ability to build easily navigable symlink trees environments (covered today)
6. Support no-source packages (BundlePackage) to aggregate related packages
7. Extensions: users can write custom commands that live outside of Spack repo
8. ARM + Fujitsu compiler support
9. GitLab Build Pipelines: Spack can generate a pipeline from a stack (covered in slides)

- **Over 3,500 packages (~700 added since last year)**

- **Full release notes:** <https://github.com/spack/spack/releases/tag/v0.13.0>

# Related Work

## Spack is not the first tool to automate builds

- Inspired by copious prior work

### 1. “Functional” Package Managers

- Nix
- GNU Guix

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

### 2. Build-from-source Package Managers

- Homebrew
- MacPorts

<http://brew.sh>  
<https://www.macports.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 unoptimized binaries

<https://conda.io>

# Spack at SC19

## ■ Meet the developers at DOE's Booth 925

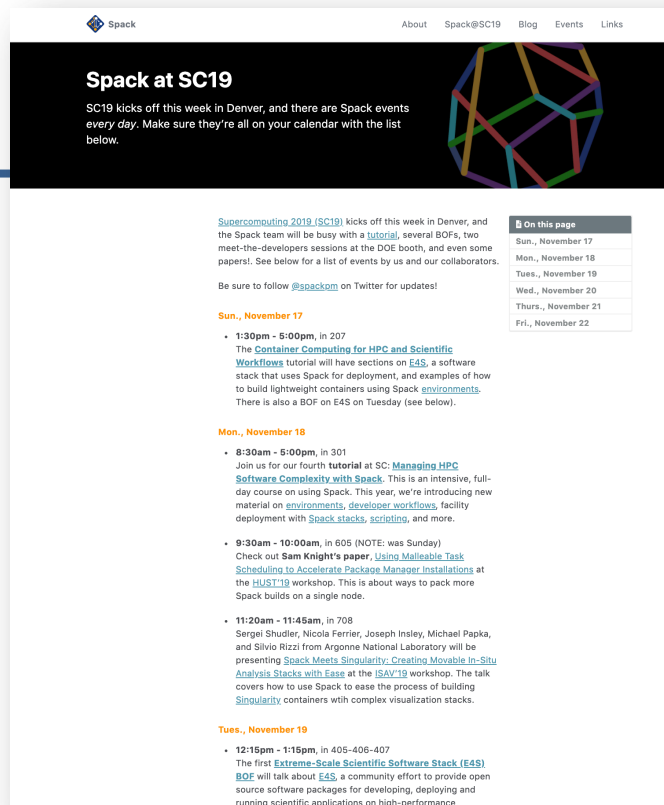
- Wednesday 10:00am – 11:00am
- Thursday 2:30pm – 3:30pm

## ■ BOFs:

- E4S BOF: Tues 12:15 – 1:15
- Getting Scientific Software Installed: Wed 12:15 – 1:15
- **Spack Community BOF:** Thurs 12:15 – 1:15

## ■ 3 papers at workshops

## ■ More!



The screenshot shows the Spack website's announcement for SC19. At the top, the Spack logo is on the left, and navigation links for 'About', 'Spack@SC19', 'Blog', 'Events', and 'Links' are on the right. A large black banner with the text 'Spack at SC19' and a sub-header 'SC19 kicks off this week in Denver, and there are Spack events every day. Make sure they're all on your calendar with the list below.' is prominent. To the right of the banner is a colorful geometric logo. Below the banner, the text 'Supercomputing 2019 (SC19) Kicks off this week in Denver, and the Spack team will be busy with a tutorial, several BOFs, two meet-the-developers sessions at the DOE booth, and even some papers! See below for a list of events by us and our collaborators.' is followed by a link to follow @spack on Twitter. A section titled 'On this page' lists dates from Sunday, November 17 to Friday, November 22. The main content area lists events for Sunday, November 17, including a tutorial on 'Container Computing for HPC and Scientific Workflows' and a BOF on 'E4S'. It also lists events for Monday, November 18, including a tutorial on 'Managing HPC Software Complexity with Spack' and a paper presentation. The list continues for Tuesday, November 19, with a BOF on 'E4S'.

# For a full list of events, visit [spack.io](https://spack.io)

# 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	libgpg-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	hdf5	libjson-c	mpc	otf	py-epydoc	py-pylint	ravel	thrift
asciidoc	cloog	fontconfig	hwloc	libmng	mpe2	otf2	py-funcsigs	py-pypar	readline	tk
atk	cmake	freetype	hypr	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	libpciaccess	mpich	paraver	py-h5py	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	samttools	util-linux
automake	cscope	geos	jdk	libtiff	mumps	parpack	py-lockfile	py-pytz	scalasca	valgrind
bear	cube	gflags	jemalloc	libtool	munge	patchelf	py-mako	py-rpy2	scorep	vim
bib2xhtml	curl	ghostscript	jpeg	libunwind	muster	pcr	py-matplotlib	py-scientificpython	scotch	vtk
binutils	czmq	git	judy	libuuid	mvapich2	pcr2	py-mock	py-sci-kit-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	pixmap	py-nose	py-sip	spindle	xerces-c
bzip2	dri2proto	glpk	libarchive	llvm	netgauge	pkg-config	py-numexpr	py-six	spot	xz
cairo	dtcm	gmp	libcerf	llvm-ll	netlib-blas	pmgr-collective	py-numpy	py-sphinx	sqlite	yasm
callpath	dyninst	gms	libcrcle	lmb	netlib-lapack	postgres	py-pandas	py-sympy	stat	zeromq
cbias	eigen	gnuplot	libdrm	lmod	netlib-scalapack	ppl	py-pbr	py-tappy	sundials	zlib
cbtf	elfutils	gnutls	libdw	lua	nettle	protobuf	py-periodictable	py-twisted	swig	zsh
cbtf-argonavis	elpa	gperf	libedit	lwm	ninja	py-astropy	py-pexpect	py-urwid	szip	
cbtf-krell	expat	gperf-tools	libelf	lwm2	omps	py-basemap	py-pil	py-virtualenv	tar	
cbtf-lanl	extrae	graphlib	libevent	matio	ompt-openmp	py-biopython	py-pillow	py-yapf	task	
cereal	exuberant-ctags	graphviz	libffi	mbedtls	opari2	py-blessings	py-pmw	python	taskd	
cfitsio	fftw	gs	libgrypt	memaxes	openblas	py-cffi	py-pychecker	qull	tau	

- Spack has over 3,500 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-numpy@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-setuptools@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                             callpath@1.0.2
```



## Expand dependencies with `spack find -d`

```
$ spack find -dl callpath
==> 2 installed packages.
-- linux-rhel6-x86_64 / clang@3.4 -----      -- linux-rhel6-x86_64 / gcc@4.9.2 -----
xv2clz2      callpath@1.0.2                      udltshts      callpath@1.0.2
ckjazss      ^adept-utils@1.0.1                    rfsu7fb       ^adept-utils@1.0.1
3ws43m4      ^boost@1.59.0                          ybet64y       ^boost@1.55.0
ft7znm6      ^mpich@3.1.4                            aa4ar6i       ^mpich@3.1.4
qqnuet3      ^dyninst@8.2.1                          tmnng5        ^dyninst@8.2.1
3ws43m4      ^boost@1.59.0                          ybet64y       ^boost@1.55.0
g65rdud      ^libdwarf@20130729                      g2mxrl2       ^libdwarf@20130729
cj5p5fk      ^libelf@0.8.13                          ynpai3j       ^libelf@0.8.13
cj5p5fk      ^libelf@0.8.13                          ynpai3j       ^libelf@0.8.13
g65rdud      ^libdwarf@20130729                      g2mxrl2       ^libdwarf@20130729
cj5p5fk      ^libelf@0.8.13                          ynpai3j       ^libelf@0.8.13
cj5p5fk      ^libelf@0.8.13                          ynpai3j       ^libelf@0.8.13
ft7znm6      ^mpich@3.1.4                            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](https://spack-tutorial.readthedocs.io)

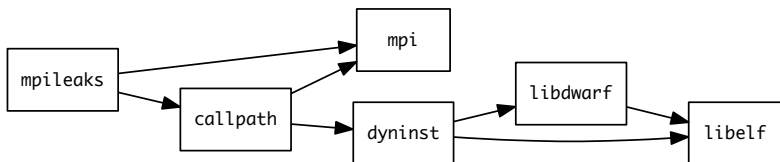
# Core Spack Concepts

# 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.

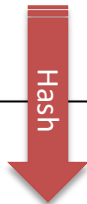
# Spack handles combinatorial software complexity.

## Dependency DAG



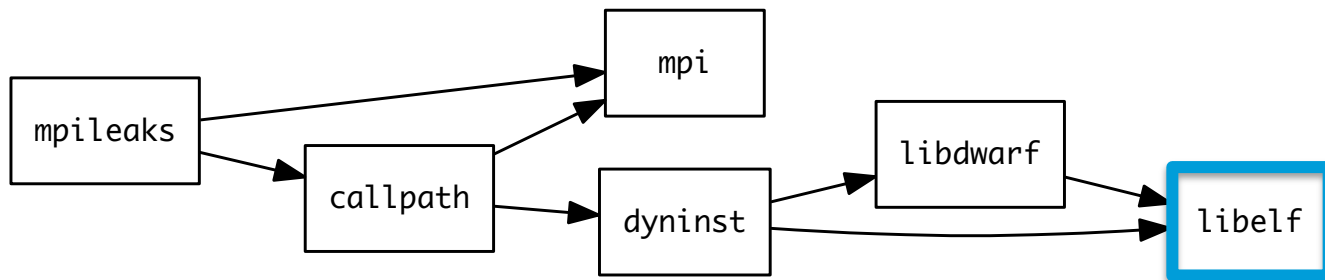
## Installation Layout

spack/opt/  
  linux-x86\_64/  
    gcc-4.7.2/  
      mpileaks-1.1-0f54bf34cadk/  
      intel-14.1/  
        hdf5-1.8.15-lkf14aq3nqiz/  
      bgq/  
        xl-12.1/  
          hdf5-1-8.16-fqb3a15abrwX/  
      ...



- Each unique dependency graph is a unique **configuration**.
- Each configuration installed in a unique directory.
  - 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*

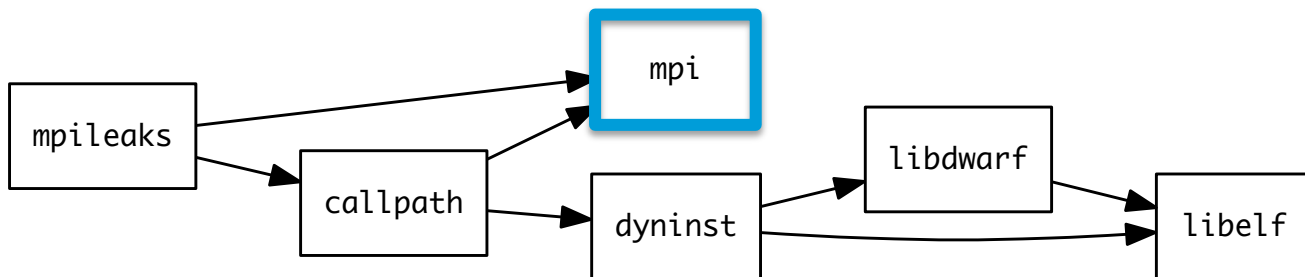
# Spack Specs can constrain versions of dependencies



```
$ 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
```

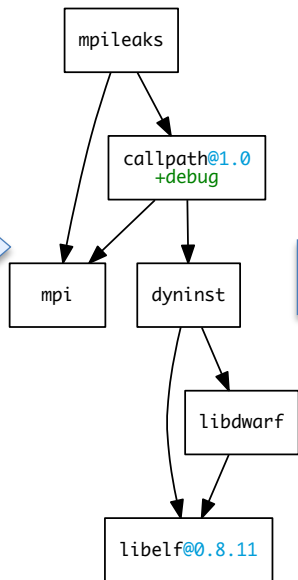
# Concretization fills in missing configuration details when the user is not explicit.

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

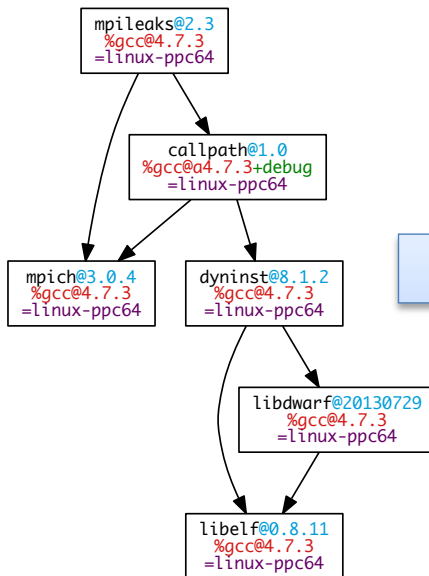
User input: *abstract* spec with some constraints

spec.yaml

Normalize



Concretize



Store

```
spec:
- mpileaks:
  arch: linux-x86_64
  compiler:
    name: gcc
    version: 4.9.2
  dependencies:
    adept-utils: kszrtkpbzac3ss2ixcjkcorlaybnptp4
    callpath: bah5f4h4d2n47mgycej2mtnrriuvxy77
    mpich: aa4ar6ifj23yijamdabeakpejcli72t3
    hash: 33hjxhxi7p6gyzn5ptgyes7sghyprujh
    variants: {}
    version: '1.0'
- adept-utils:
  arch: linux-x86_64
  compiler:
    name: gcc
    version: 4.9.2
  dependencies:
    boost: teesjv7ehpe5ksspjim5dk43a7qnowlq
    mpich: aa4ar6ifj23yijamdabeakpejcli72t3
    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
...
```

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
```

# 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-oaix36/  
              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-oaix36/  
              lib/python2.7/site-packages/  
                numpy/
```



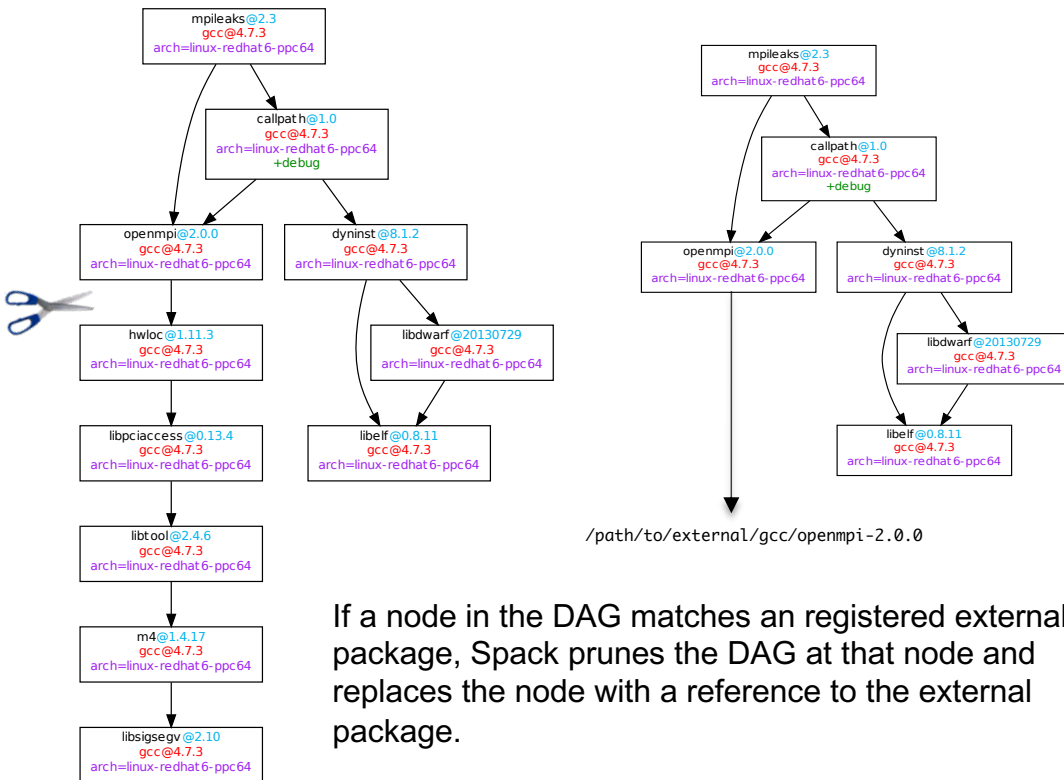
# Building against externally installed software

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

## packages.yaml

```
packages:
  mpi:
    buildable: False
  openmpi:
    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
    openmpi@2.0.0 %intel@16.0.0 arch=linux-rhel6-ppc64:
      /path/to/external/intel/openmpi-2.0.0
    openmpi@1.10.3 %intel@16.0.0 arch=linux-rhel6-ppc64:
      /path/to/external/intel/openmpi-1.10.3
  ...
```

A user registers external packages with Spack.



If a node in the DAG matches an registered external package, Spack prunes the DAG at that node and replaces the node with a reference to the external package.

# Spack package repositories

- Some packages can not be released publicly
- Some users have use cases that require ~~bizarre~~ custom builds
- Packaging issues should not prevent users from updating Spack
  - Solution: separate repositories
  - A repository is simply a directory of package files
- Spack supports external repositories that can be layered on top of the built-in packages
- Custom packages can depend on built-in packages (or packages in other repositories)

```
$ 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
```

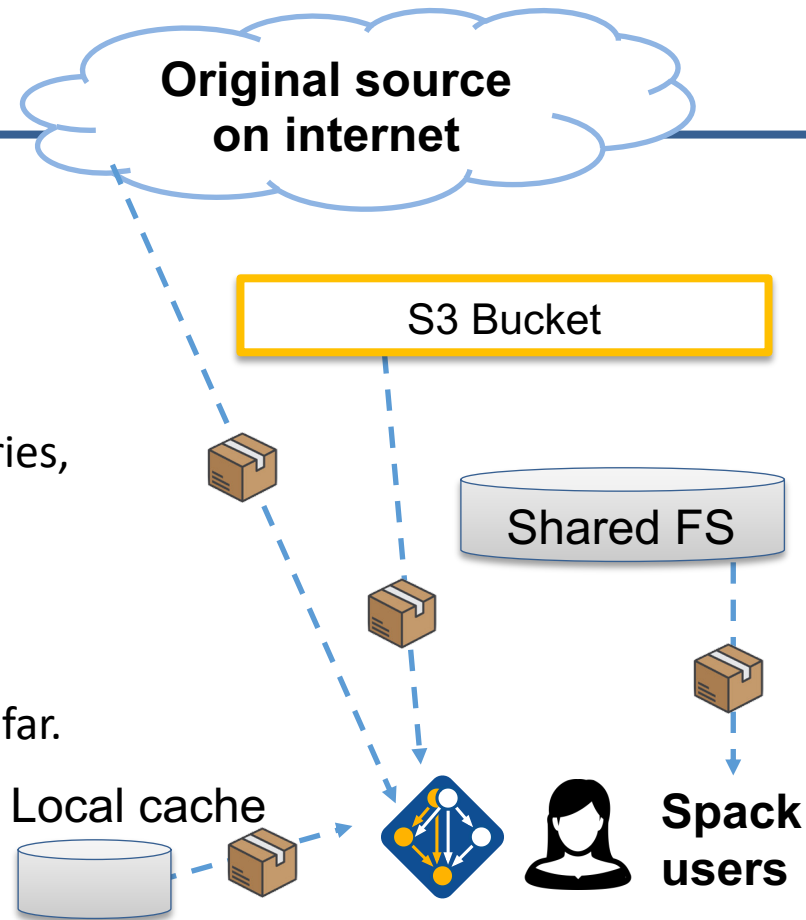


**my\_repo**  
proprietary packages, pathological builds

**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](https://spack-tutorial.readthedocs.io)

# Making your own Spack Packages



# Creating your own Spack Packages

- Package is a recipe for building
- Each package is a Python class
  - Download information
  - Versions/Checksums
  - Build options
  - Dependencies
  - Build instructions
- Package collections are repos
  - Spack has a “builtin” repo in **var/spack/repos/builtin**

## `$REPO/packages/zlib/package.py`

```
from spack import *

class Zlib(Package):
    """A free, general-purpose, legally unencumbered lossless
    data-compression library."""

    homepage = "http://zlib.net"
    url      = "http://zlib.net/zlib-1.2.8.tar.gz"

    version('1.2.8', '44d667c142d7cda120332623eab69f40')

    depends_on('cmake', type='build')

    def install(self, spec, prefix):
        configure('--prefix={0}'.format(prefix))

        make()
        make('install')
```

# Spack packages are *templates* for builds

- Each package has one class
  - zlib for Intel compiler and zlib for GCC compiler are built with the same recipe.
- Can add conditional logic using spec syntax
  - Think of package as *translating* a concrete DAG to build instructions.
  - Dependencies are already built
  - No searching or testing; just do what the DAG says
- Compiler wrappers handle many details automatically.
  - Spack feeds compiler wrappers to (cc, c++, f90, ...) to autoconf, cmake, gmake, ...
  - Wrappers select compilers, dependencies, and options under the hood.

## package.py

```
def install(self, spec, prefix):
    config_opts=['--prefix='+prefix]

    if '~shared' in self.spec:
        config_opts.append('--disable-shared')
    else:
        config_opts.append('--enable-shared')

    configure(config_opts)
    make()
    make('install')
```

# Spack builds each package in its own compilation environment

**Spack Process**

do\_install()

Install dep1

Install dep2

...

Install package

Fork

**Build Process**

Set up environment

```
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 = ...
```

install()

- **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)

icc

icpc

ifort

**Compiler wrappers**

(spack-cc, spack-c++, spack-f77, spack-f90)

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

configure

make

make install

# Writing Packages - Versions and URLs

`$REPO/packages/mvapich/package.py`

```
class Mvapich2(Package):  
    homepage = "http://mvapich.cse.ohio-state.edu/"  
    url = "http://mvapich.cse.ohio-state.edu/download/mvapich/mv2/mvapich2-2.2rc2.tar.gz"  
  
    version('2.2rc2', 'f9082ffc3b853ad1b908cf7f169aa878')  
    version('2.2b', '5651e8b7a72d7c77ca68da48f3a5d108')  
    version('2.2a', 'b8ceb4fc5f5a97add9b3ff1b9cbe39d2')  
    version('2.1', '0095ceecb19bbb7fb262131cb9c2cdd6')
```

- Package downloads are hashed with MD5 by default
  - Also supports SHA-1, SHA-256, SHA-512
  - We'll be switching to SHA-256 or higher soon.
- Download URLs can be automatically extrapolated from URL.
  - Extra options can be provided if Spack can't extrapolate URLs
- Options can also be provided to fetch from VCS repositories

# Writing Packages – Variants and Dependencies

`$REPO/packages/petsc/package.py`

```
class Petsc(Package):
    variant('mpi',      default=True,  description='Activates MPI support')
    variant('complex',  default=False, description='Build with complex numbers')
    variant('hdf5',      default=True,  description='Activates support for HDF5 (only parallel)')

    depends_on('blas')
    depends_on('python@2.6:2.7')

    depends_on('mpi', when='+mpi')
```

- Variants are named, have default values and help text
- Other packages can be dependencies
  - **when** clause provides conditional dependencies
  - Can depend on specific versions or other variants

# Writing Packages – Build Recipes

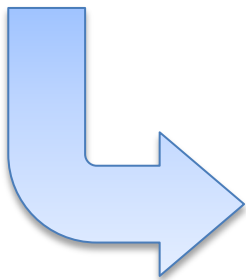
- Functions wrap common ops
  - cmake, configure, patch, make, ...
  - **Executable** and **which** for new wrappers.
- Commands executed in clean environment
- Full Python functionality
  - Patch up source code
  - Make files and directories
  - Calculate flags
  - ...

\$REPO/packages/dyninst/package.py

```
def install(self, spec, prefix):  
    with working_dir("build", create=True):  
        cmake("../", *std_cmake_args)  
        make()  
        make("install")  
  
@when('@:8.1')  
def install(self, spec, prefix):  
    configure("--prefix=" + prefix)  
    make()  
    make("install")
```

# Create new packages with spack create

```
$ spack create http://zlib.net/zlib-1.2.8.tar.gz
```



`$REPO/packages/zlib/package.py`

```
class Zlib(Package):
    # FIXME: Add a proper url for your package's homepage here.
    homepage = "http://www.example.com"
    url      = "http://zlib.net/zlib-1.2.8.tar.gz"
    version('1.2.8', '44d667c142d7cda120332623eab69f40')

    def install(self, spec, prefix):
        # FIXME: Modify the cmake line to suit your build system here.
```

- `spack create <url>` will create a skeleton for a package
  - Spack reasons about URL, hash, version, build recipe.
  - Generates boilerplate for Cmake, Makefile, autotools, Python, R, Waf, Perl
    - Not intended to completely write the package, but gets you 80% of the way there.
- `spack edit <package>` for subsequent changes

# Hands-on Time: Creating Packages

Follow script at [spack-tutorial.readthedocs.io](https://spack-tutorial.readthedocs.io)

# Hands-on Time: Environment Modules

Follow script at [spack-tutorial.readthedocs.io](https://spack-tutorial.readthedocs.io)

# Environments, `spack.yaml` and `spack.lock`

Follow script at [spack-tutorial.readthedocs.io](https://spack-tutorial.readthedocs.io)

# Spack Stacks

Follow script at [spack-tutorial.readthedocs.io](https://spack-tutorial.readthedocs.io)

# Developer Workflows

Follow script at [spack-tutorial.readthedocs.io](https://spack-tutorial.readthedocs.io)

# Scripting and spack-python

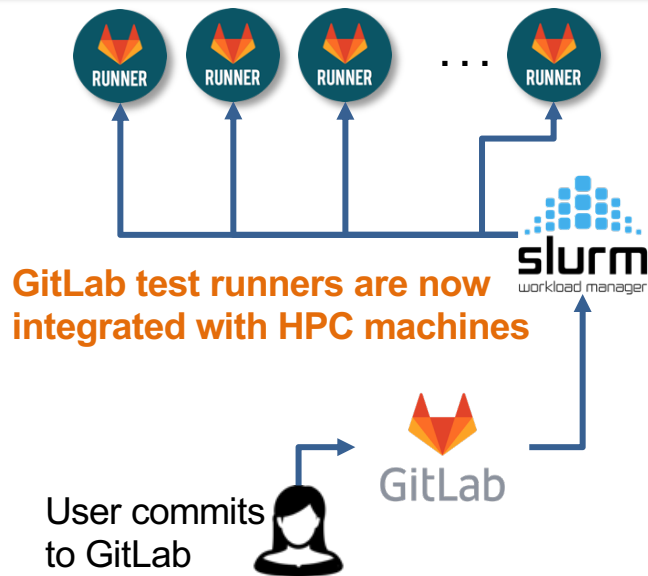
Follow script at [spack-tutorial.readthedocs.io](https://spack-tutorial.readthedocs.io)

# More New Features and the Road Ahead



# We have been heavily involved in the ECP CI project.

- We have added security features to the open source GitLab product.
  - Integration with center identity management
  - Integration with schedulers like SLURM, LSF
- We are democratizing testing at Livermore Computing
  - Users can run tests across 30+ machines by editing a file
  - Previously, each team had to administer own servers
- ECP sites are deploying GitLab CI for users
  - All HPC centers can leverage these improvements
  - NNSA labs plan to deploy common high-side CI infrastructure
  - We are developing new security policies to allow external open source code to be tested safely on key machines



# Spack now 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

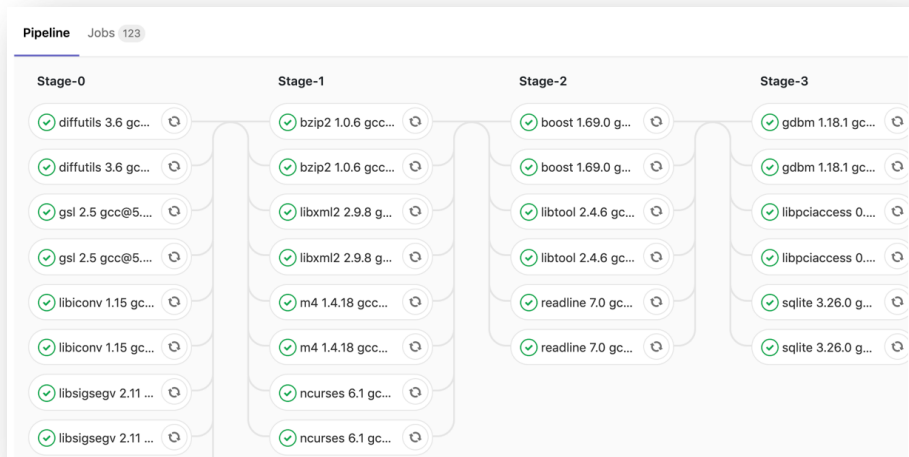
```
$ spack install lbann target=cascadelake  
$ spack install petsc target=zen2
```

Specialized installations



# Spack has added GitLab CI integration to automate package build pipelines

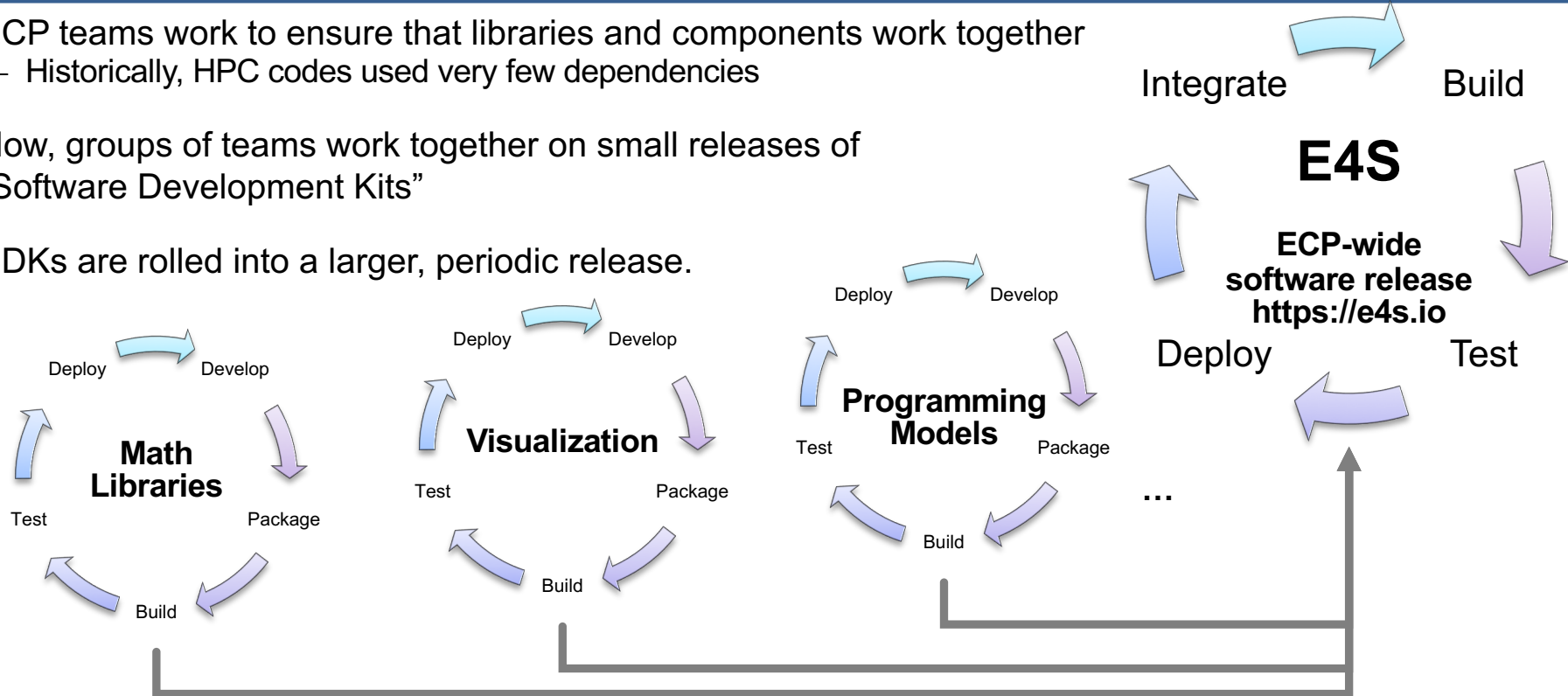
- Builds on Spack environments
  - Support auto-generating GitLab CI jobs
  - Can run in a Kube cluster or on bare metal runners at an HPC site
  - Sends progress to CDash



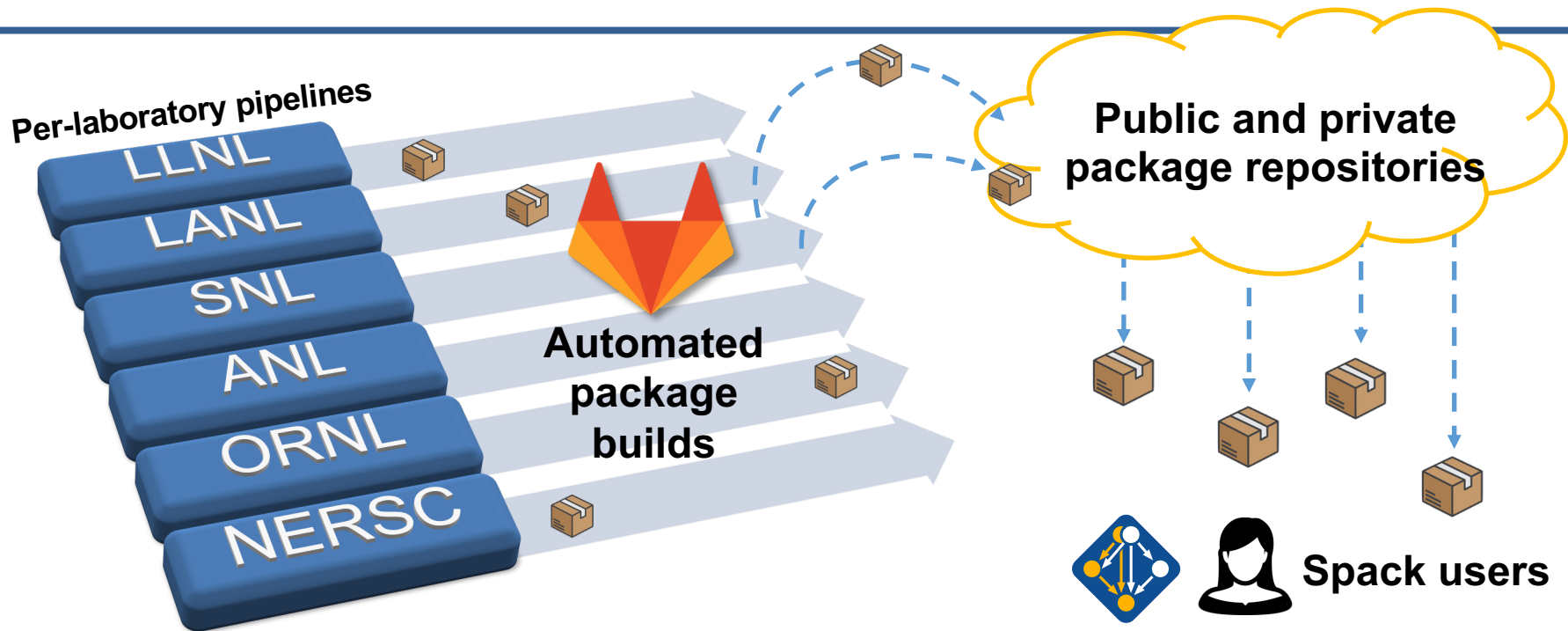
```
spack:
  definitions:
    - pkgs:
      - readline@7.0
    - compilers:
      - '%gcc@5.5.0'
    - oses:
      - os=ubuntu18.04
      - os=centos7
  specs:
    - matrix:
      - [$pkgs]
      - [$compilers]
      - [$oses]
  mirrors:
    cloud_gitlab: https://mirror.spack.io
  gitlab-ci:
    mappings: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

# ECP is working towards a periodic, hierarchical release process

- ECP teams work to ensure that libraries and components work together
  - Historically, HPC codes used very few dependencies
- Now, groups of teams work together on small releases of “Software Development Kits”
- SDKs are rolled into a larger, periodic release.



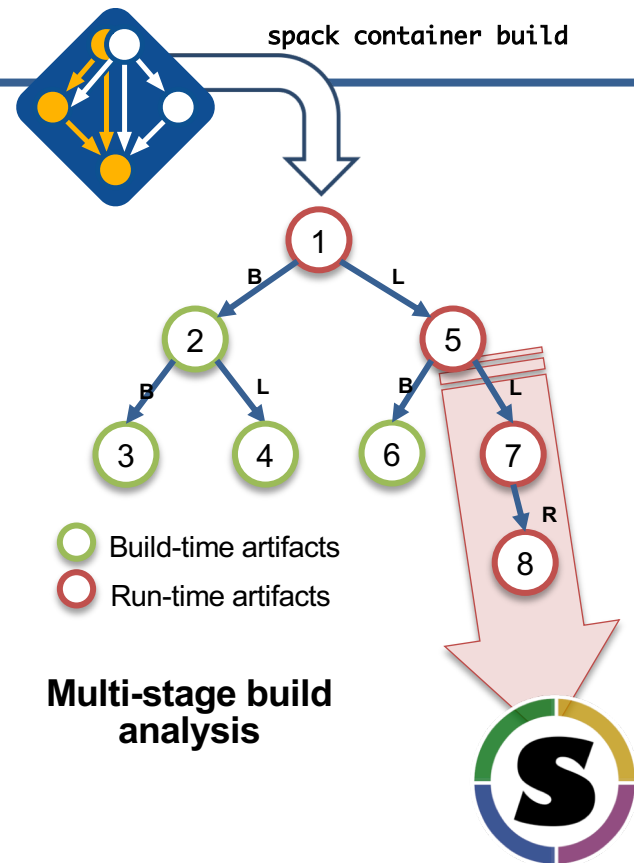
Automated builds using ECP CI will enable a robust, widely available HPC software ecosystem.



With pipeline efforts at E6 labs, users will no longer need to *build* their own software for high performance.

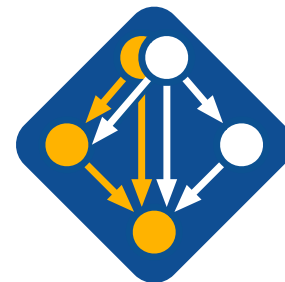
# Spack focus areas in FY20

- **Multi-stage container generation with Spack**
  - Add support to Spack to generate *multi-stage* container builds that exclude build dependencies from artifacts automatically
- **Build Hardening with Spack Pipelines**
  - Continue working with E4S team to harden container builds
- **Parallel builds**
  - “srun spack install” will use the entire allocation to build
- **New concretizer based on fast ASP/SAT solvers**
- **Improved dependency models for compilers**
  - icpc depends on g++ for its libstdc++, and other ABI nightmares



# Join the Spack community!

- There are lots of ways to get involved!
  - Contribute packages, documentation, or features at [github.com/spack/spack](https://github.com/spack/spack)
  - Contribute your configurations to [github.com/spack/spack-configs](https://github.com/spack/spack-configs)
- Talk to us!
  - Join our **Google Group** (see GitHub repo for info)
  - Join our **Slack channel** (see GitHub repo for info)
  - Submit GitHub issues and talk to us!



★  
Star us on GitHub!  
[github.com/spack/spack](https://github.com/spack/spack)



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

We hope to make distributing & using HPC software easy!

