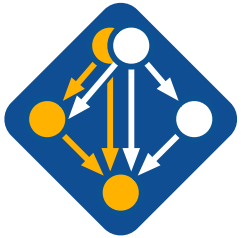


Using Spack to Accelerate Developer Workflows

ECP Annual Meeting
Half-day Tutorial
May 2, 2022

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

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

Tutorial Materials

Find these slides and associated scripts here:

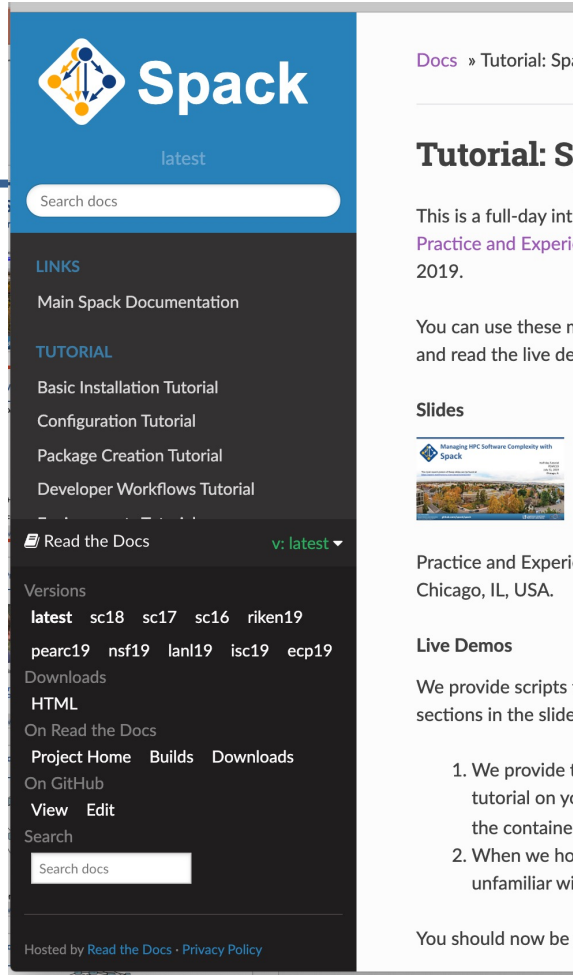
spack-tutorial.readthedocs.io

We will also have a chat room on Spack slack.
You can join here:

slack.spack.io

Join the “tutorial” channel!

We will monitor the chat during the tutorial, but we'll also help in person. You can ask questions here after the conference is over.



Tutorial Presenters

Greg Becker
LLNL

Richarda Butler
LLNL

Tamara Dahlgren
LLNL

Todd Gamblin
LLNL

Agenda (we are doing the first half of our full day tutorial)

For this ECP half-day tutorial:

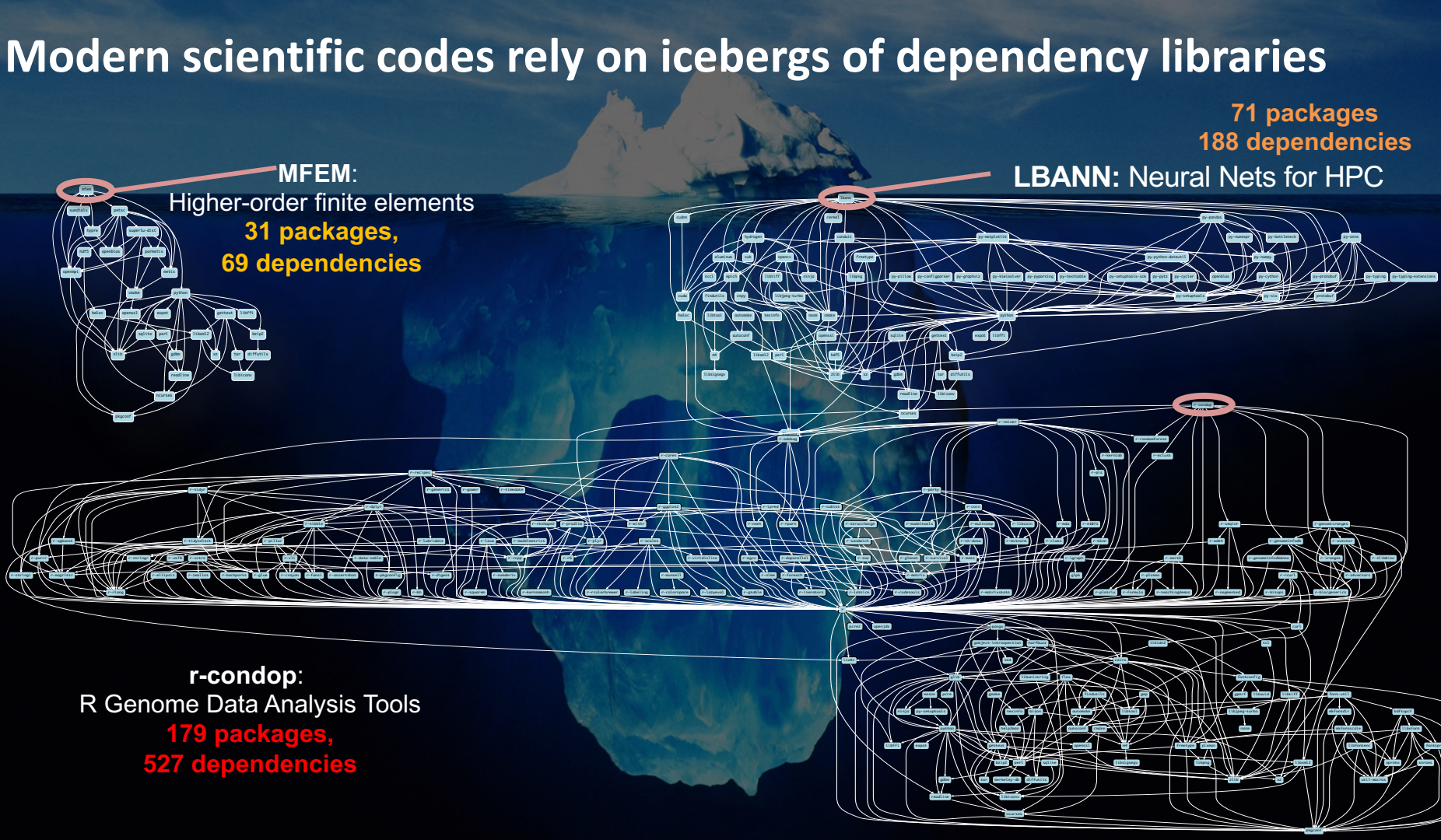
Intro	10:00 – 10:30
Basics	10:30 – 11:15
Concepts	11:15 – 11:30
Break	11:30 – 12:00
Environments	12:00 – 12:45
Configuration	12:45 – 1:30

(or Developer Workflows)

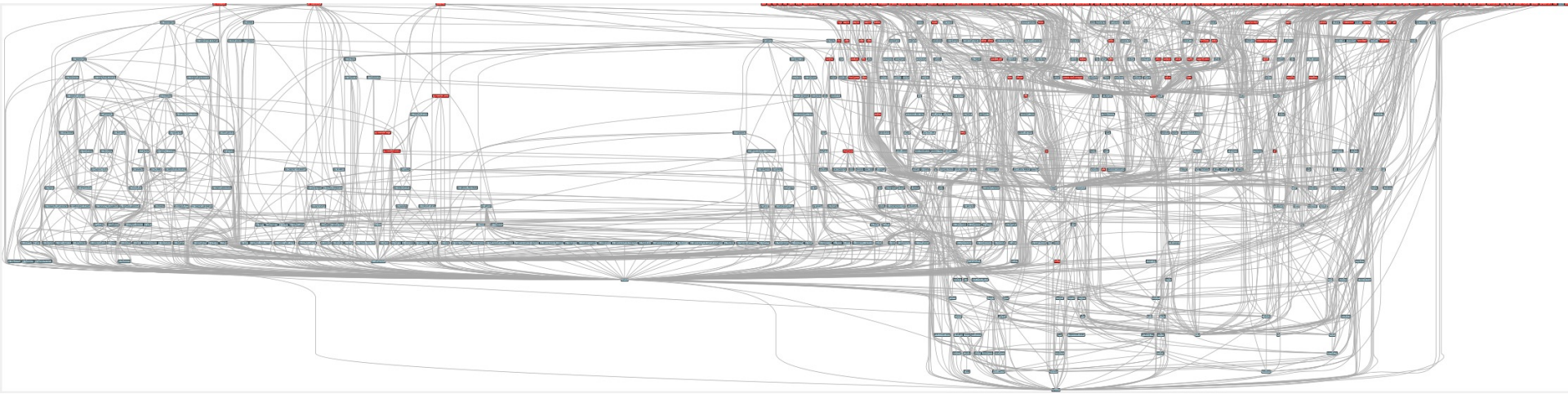
You can find the additional 6 sessions from our normal full-day tutorial at **spack-tutorial.readthedocs.io**:

Configuration	45 min
Packaging	45 min
Developer Workflows	45 min
Mirrors	20 min
Stacks	25 min
Scripting	25 min
Roadmap	20 min

Modern scientific codes rely on icebergs of dependency libraries



ECP's E4S stack is even larger than these codes



- Red boxes are the packages in it (about 100)
- Blue boxes are what *else* you need to build it (about 600)
- It's infeasible to build and integrate all of this manually

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

- **1:1 relationship between source code and binary (per platform)**
  - Good for reproducibility (e.g., Debian)
  - Bad for performance optimization
- **Binaries should be as portable as possible**
  - What most distributions do
  - Again, bad for performance
- **Toolchain is the same across the ecosystem**
  - One compiler, one set of runtime libraries
  - Or, no compiler (for interpreted languages)

Outside these boundaries, users are typically on their own

High Performance Computing (HPC) violates many of these assumptions

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

Some Supercomputers

Current

Oak Ridge National Lab
Power9 / NVIDIA

RIKEN
Fujitsu/ARM a64fx

Upcoming

Lawrence Berkeley National Lab
AMD Zen / NVIDIA

Argonne National Lab
Intel Xeon / Xe

Oak Ridge National Lab
AMD Zen / Radeon

Lawrence Livermore National Lab
AMD Zen / Radeon

What about containers?

- **Containers provide a great way to reproduce and distribute an already-built software stack**
- **Someone needs to build the container!**
  - This isn't trivial
  - Containerized applications still have hundreds of dependencies
- **Using the OS package manager inside a container is insufficient**
  - Most binaries are built unoptimized
  - Generic binaries, not optimized for specific architectures
- **HPC containers may need to be *rebuilt* to support many different hosts, anyway.**
  - Not clear that we can ever build one container for all facilities
  - Containers likely won't solve the N-platforms problem in HPC

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

Spack enables Software distribution for HPC

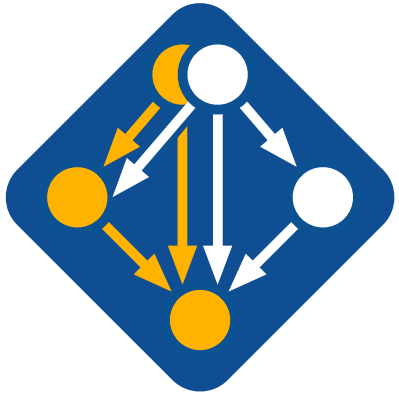
- Spack automates the build and installation of scientific software
- Packages are *parameterized,* so that users can easily tweak and tune configuration

No installation required: clone and go

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

Simple syntax enables complex installs

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

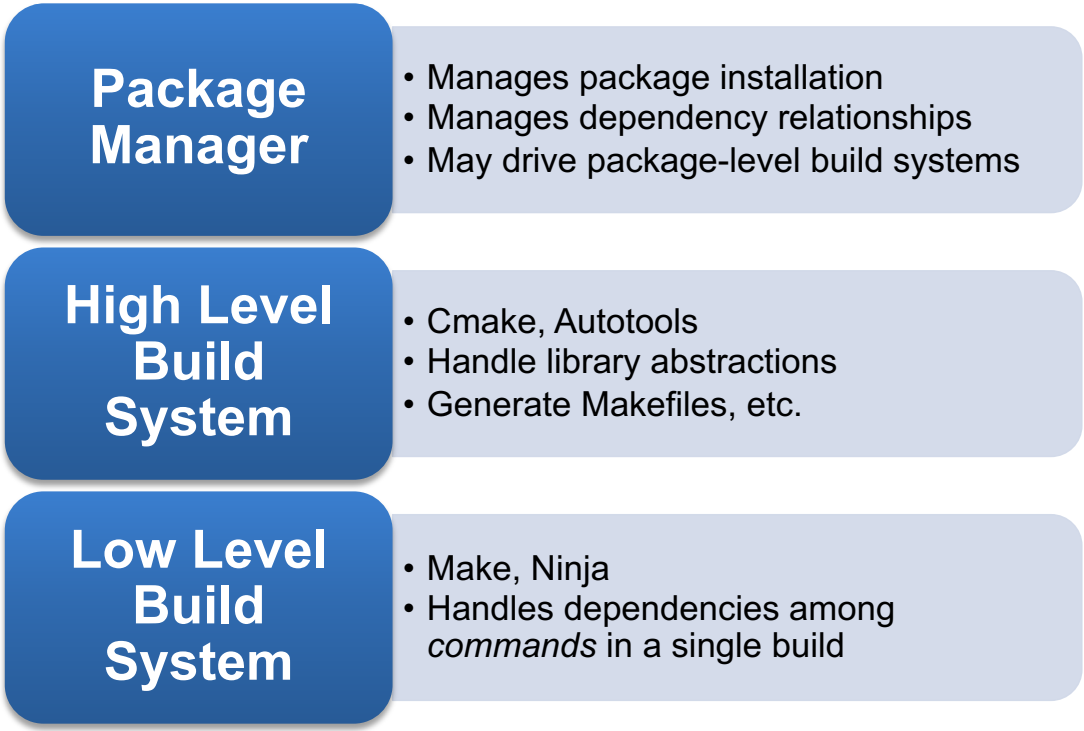


github.com/spack/spack

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

What's a package manager?

- Spack is a ***package manager***
  - **Does not** a replace Cmake/Autotools
  - Packages built by Spack can have any build system they want
- Spack manages ***dependencies***
  - Drives package-level build systems
  - Ensures consistent builds
- Determining magic configure lines takes time
  - Spack is a cache of recipes



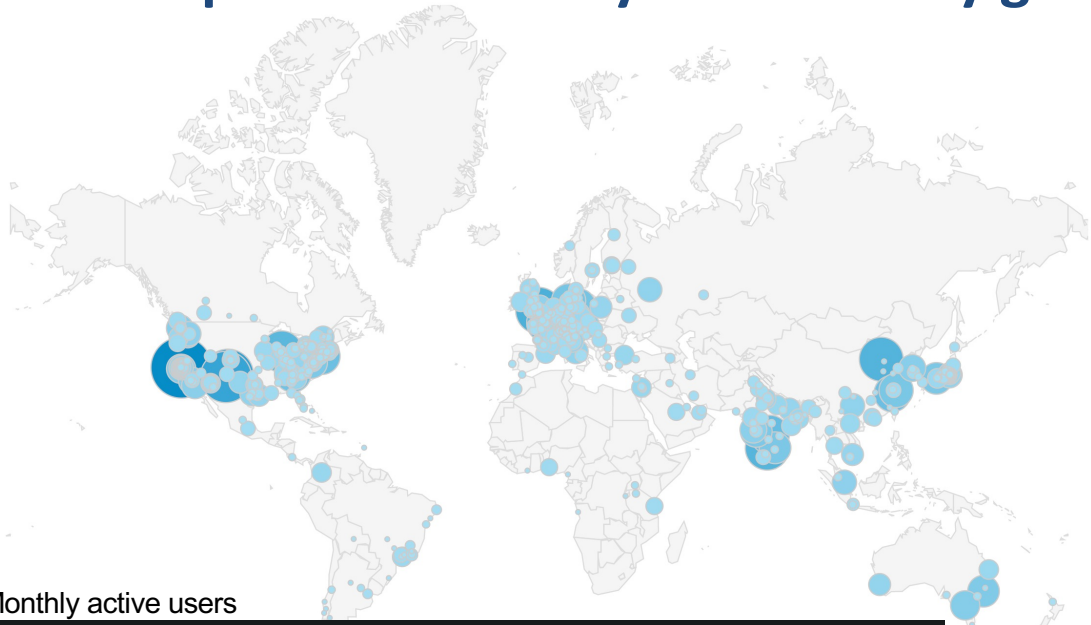
Who can use Spack?

People who want to use or distribute software for HPC!

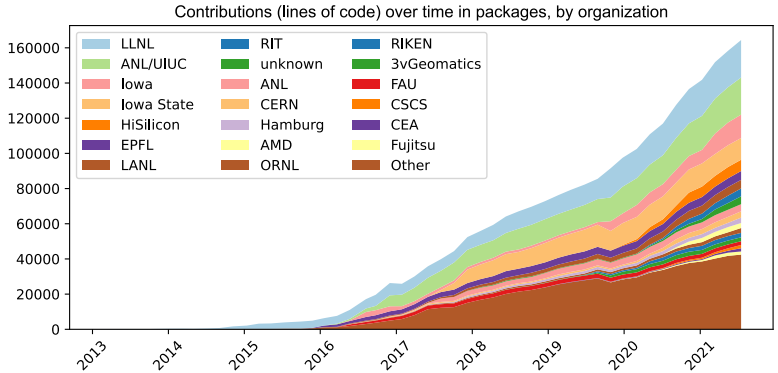
1. **End Users of HPC Software**
   - Install and run HPC applications and tools
2. **HPC Application Teams**
   - Manage third-party dependency libraries
3. **Package Developers**
   - People who want to package their own software for distribution
4. **User support teams at HPC Centers**
   - People who deploy software for users at large HPC sites

The Spack community is constantly growing!

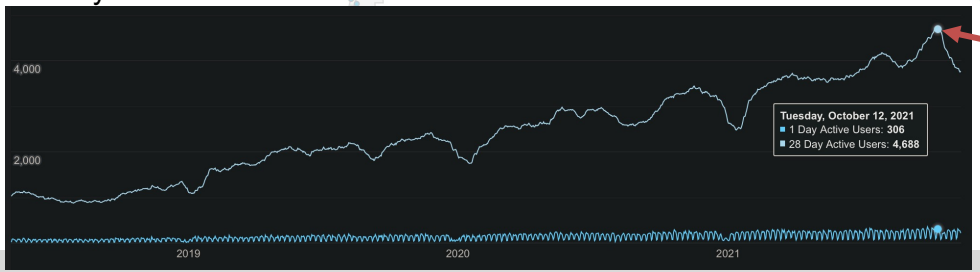
6,000+ software packages
930+ contributors



Package contribution rate remained steady in 2021

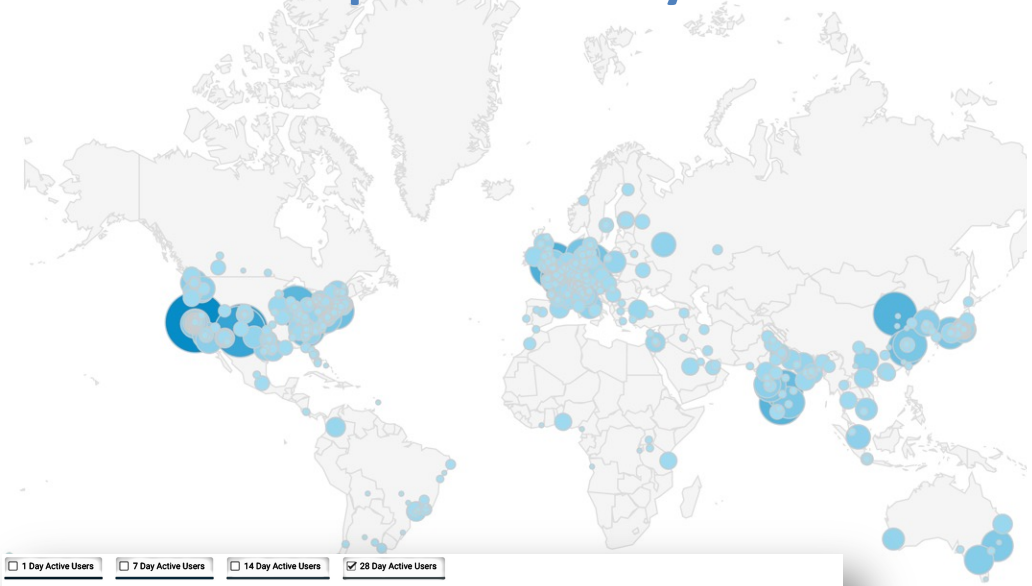


Monthly active users

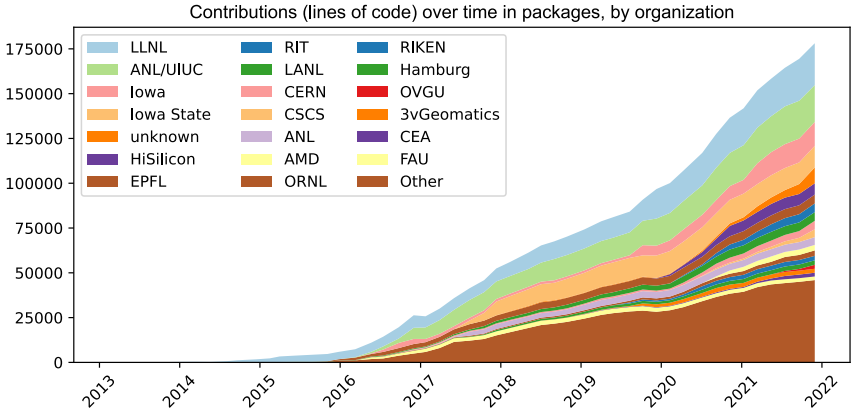


All time high of 4,688
monthly active users this October

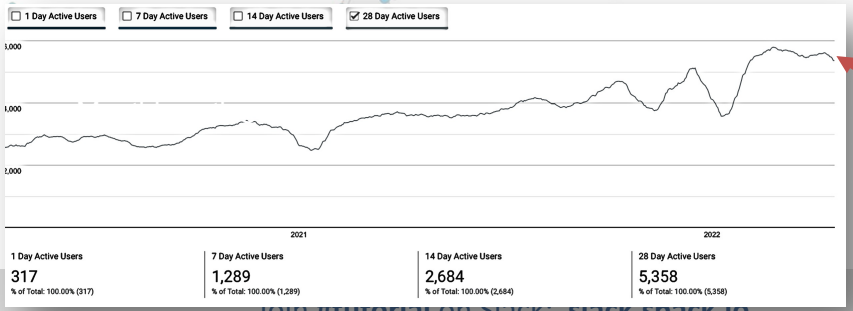
Spack sustains the HPC software ecosystem with the help of its many contributors



6,300+ software packages
Over 1,000 contributors

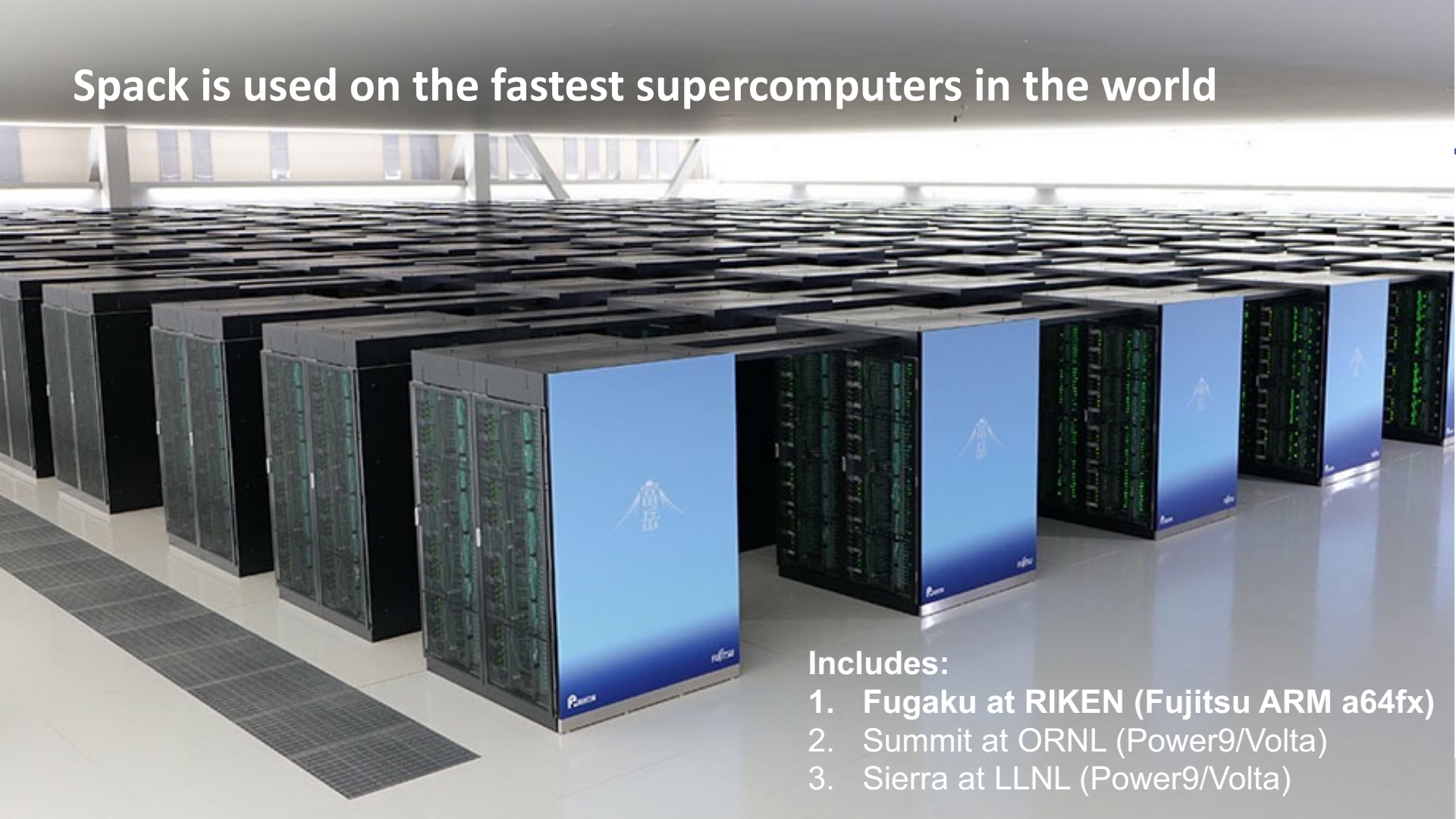


Most package contributions are ***not*** from DOE
But they help sustain the DOE ecosystem!



Nearly 6,000 monthly active users
(per documentation site)

Spack is used on the fastest supercomputers in the world



Includes:

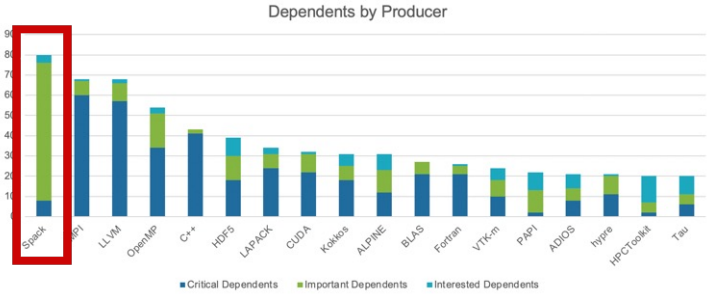
1. **Fugaku at RIKEN (Fujitsu ARM a64fx)**
2. Summit at ORNL (Power9/Volta)
3. Sierra at LLNL (Power9/Volta)

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

EXASCALE COMPUTING PROJECT

https://e4s.io

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



Spack is the most depended-upon project in ECP

One month of Spack development is pretty busy!

October 12, 2021 – November 12, 2021

Period: 1 month

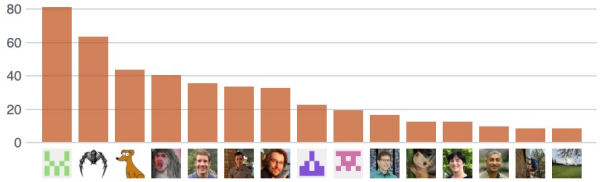
Overview

671 Active Pull Requests

145 Active Issues

536 Merged Pull Requests	**135** Open Pull Requests	**75** Closed Issues	**70** New Issues

Excluding merges, **173 authors** have pushed **571 commits** to develop and **634 commits** to all branches. On develop, **703 files** have changed and there have been **20,730 additions** and **3,807 deletions**.



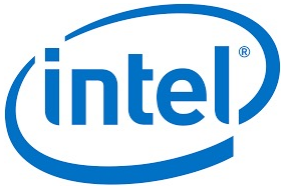
1 Release published by **1** person

v0.17.0
published 7 days ago

536 Pull requests merged by **151** people

Spack's widespread adoption has made it a de facto standard, drawing contribution and collaboration from many vendors

- **AWS** invests significantly in cloud credits for Spack build farm
  - Joint Spack tutorial with AWS had 125+ participants
  - Joint AWS/AHUG Spack Hackathon drew 60+ participants
- **AMD** has contributed ROCm packages and compiler support
  - 55+ PRs mostly from AMD, also others
  - ROCm, HIP, aocc packages are all in Spack now
- **HPE/Cray** is doing internal CI for Spack packages, in the Cray environment
- **Intel** contributing OneApi support and licenses for our build farm
- **NVIDIA** contributing NVHPC compiler support and other features
- **Fujitsu and RIKEN** have contributed a **huge** number of packages for ARM/a64fx support on Fugaku
- **ARM** and **Linaro** members contributing ARM support
  - 400+ pull requests for ARM support from various companies

Spack is not the only tool that automates builds

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

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

Other tools in the HPC Space:

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

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

Hands-on Time: Spack Basics

Follow script at spack-tutorial.readthedocs.io

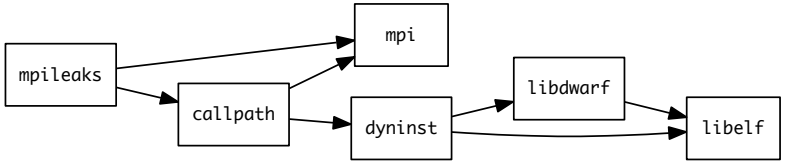
Core Spack Concepts

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



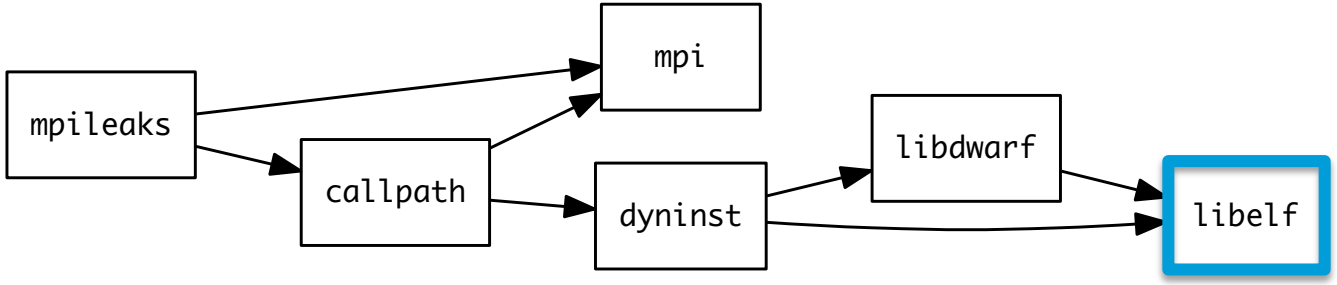
Installation Layout

Hash

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

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

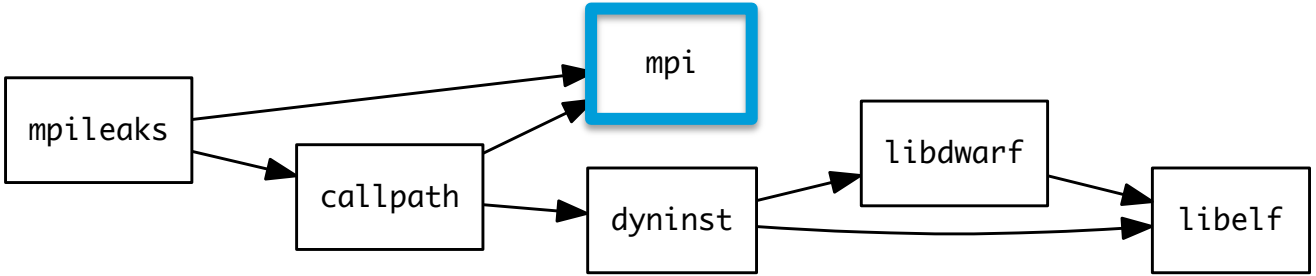
Spack Specs can constrain versions of dependencies




```
$ 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:
```

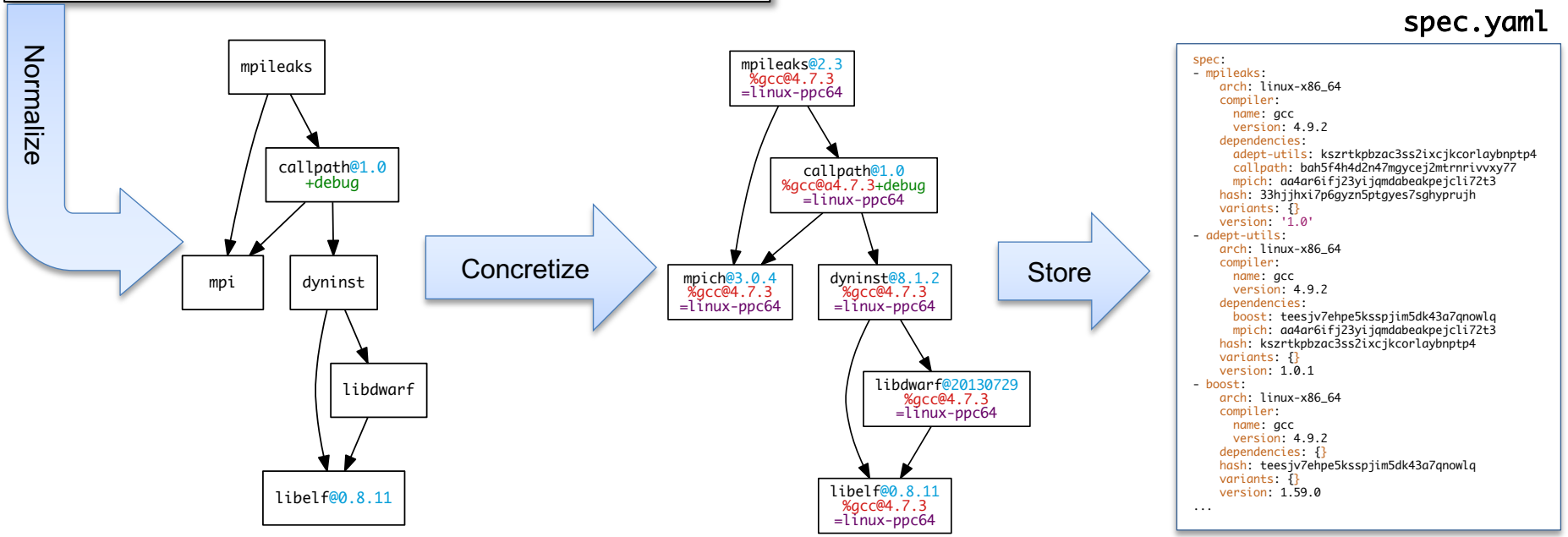
- Let Spack choose MPI implementation, as long as it provides MPI 2 interface:

```
$ 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

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

Abstract, normalized spec with some dependencies.

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

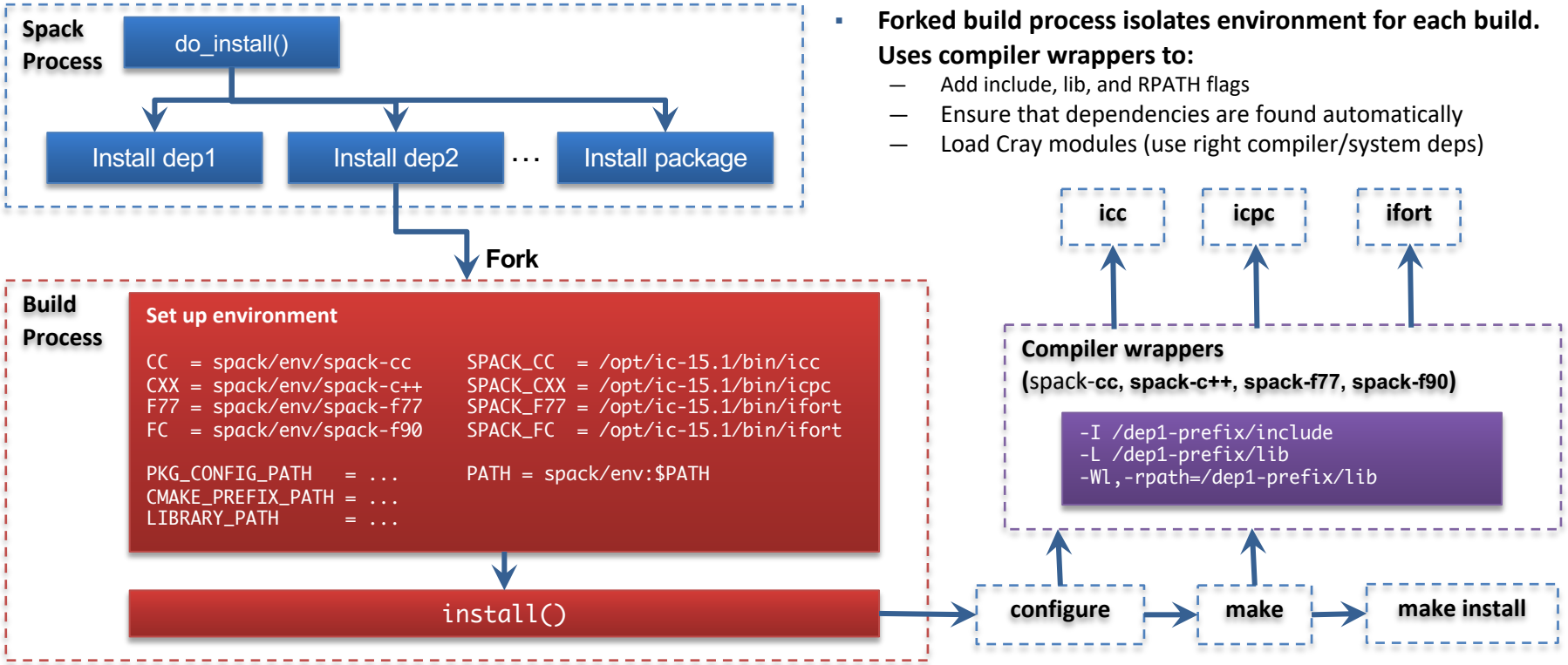
Detailed provenance is stored with the installed package

Use `spack spec` to see the results of concretization

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

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

Spack builds each package in its own compilation environment



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

Extensions and Python Support

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

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

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

    extends('python')

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

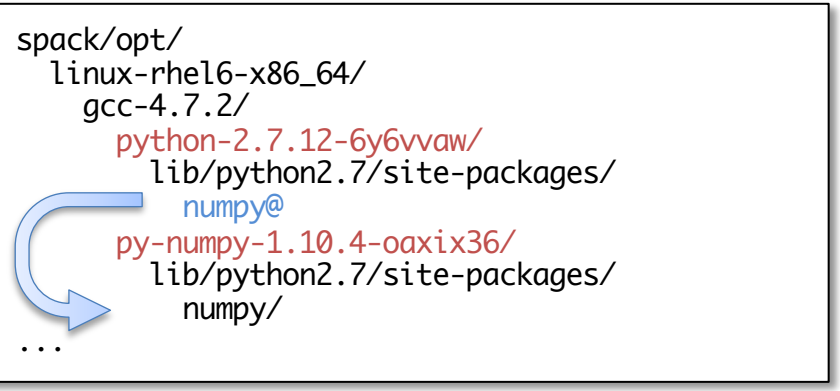
Spack extensions

- Some packages need to be installed within directory structure of other packages
- Examples of extension packages:
  - python libraries are a good example
  - R, Lua, perl
  - Need to maintain combinatorial versioning

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

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

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

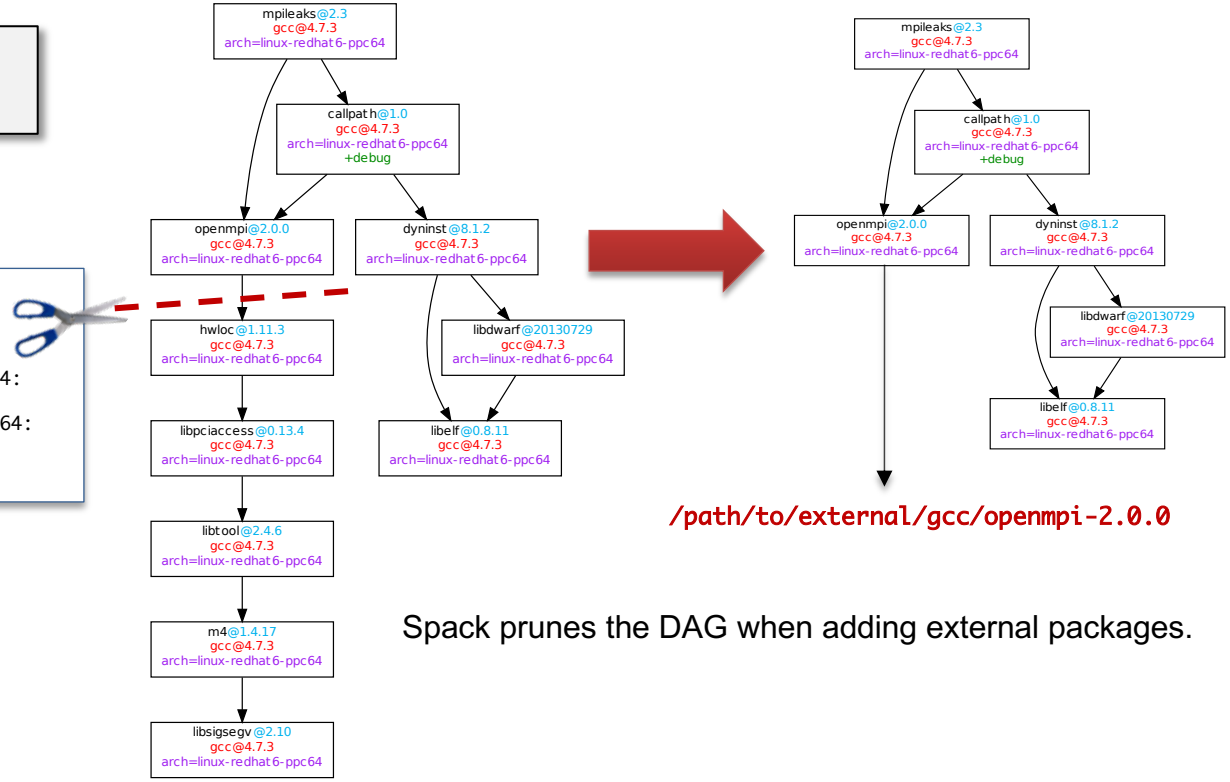


Building against externally installed software

```
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.

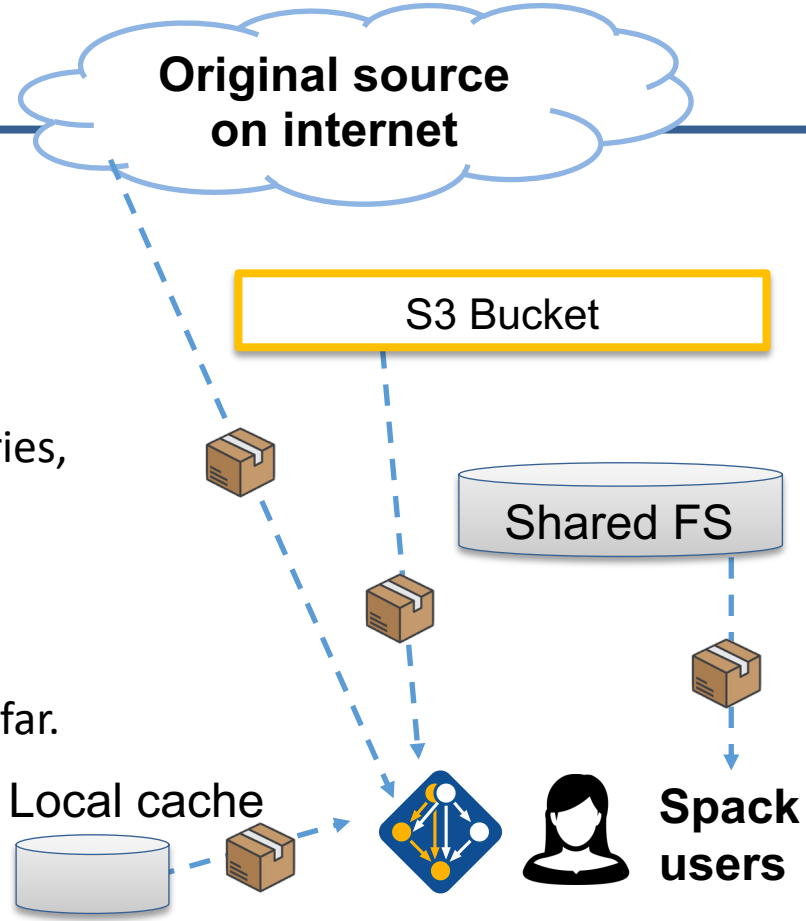
Spack package repositories

- 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)

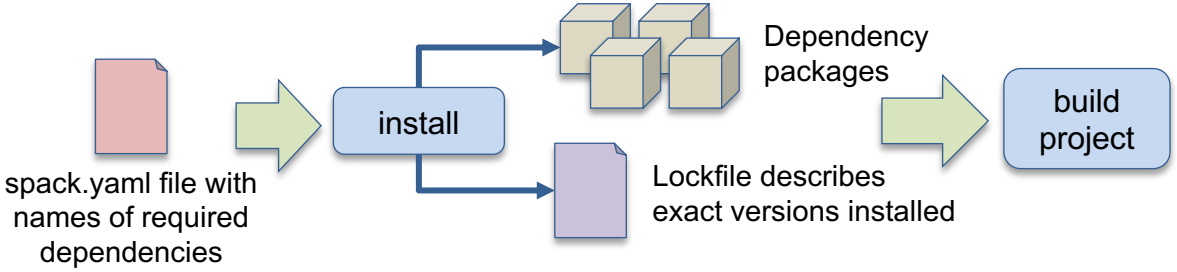
```
$ spack repo create /path/to/my_repo
$ spack repo add my_repo
$ spack repo list
==> 2 package repositories.
my_repo    /path/to/my_repo
builtin    spack/var/spack/repos/builtin
```

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



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



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

Simple spack.yaml file

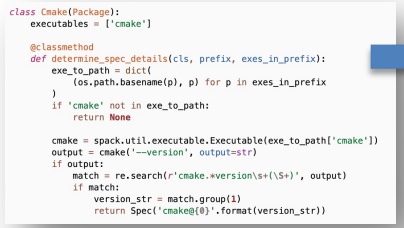
```
spack:
  # include external configuration
  include:
  - ../special-config-directory/
  - ./config-file.yaml

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

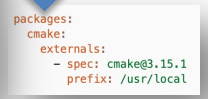
Concrete spack.lock file (generated)

```
{
 "concrete_specs": {
   "6s63so2kstp3zyvjezglndmavy6l3nul": {
     "hdf5": {
         "version": "1.10.5",
         "arch": {
             "platform": "darwin",
             "platform_os": "mojave",
             "target": "x86_64"
         },
         "compiler": {
             "name": "clang",
             "version": "10.0.0-apple"
         },
         "namespace": "buil
         "parameters":
```

Environments have enabled us to add build many features to support developer workflows



package.py



`spack.yaml` configuration

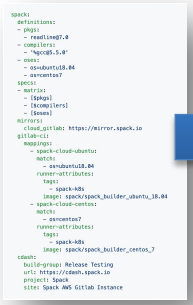
`spack external find`

Automatically find and configure external packages on the system

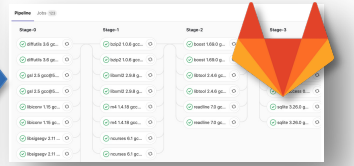
`spack test`

Packages know how to run their own test suites

package.py



`spack.yaml`



`.gitlab-ci.yml` CI pipeline

`spack ci`

Automatically generate parallel build pipelines (more on this later)

`spack containerize`

Turn environments into container build recipes

E4S is ECP's curated, Spack-based software distribution

- **E4S is just a set of Spack packages**
  - **60+ packages (297 including dependencies)**
  - **Growing to include all of ST and more**

- Users can install E4S packages:
  - In their home directory
  - In a container

- Facilities can install E4S packages:
  - On bare metal
  - In a container

- Users and facilities can choose parts they want
  - `spack install` only the packages you want
  - Or just edit the list of packages (and configurations) you want in a `spack.yaml` file

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

  view: false
  concretization: separately
```

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

More on E4S at https://e4s.io

c/o Robert Blake

The AML team has used Spack environments to accelerate their workflow

- **LLNL Applied ML team needed to deploy**
  - PyTorch + Kull development environment
  - On ppc64le with system MPI
- **Before Spack**
  - Everybody built from scratch
  - People wrote scripts and passed them around
  - **Days were spent trying to debug build differences**
- **After spack**
  - Versioned reproducible spack environments in a git repo
  - Standard environments in a shared team directory
  - **Team members can set up a customizable environment in ~20 minutes.**
    - Change python version, PyTorch version on the fly
    - Leverage binary caches to avoid redundant builds.

```
spack:
  specs:
  - py-horovod
  - py-torch
  - python
  - py-h5py

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

spack.yaml file

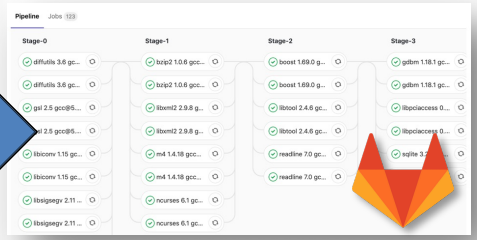
We wanted to translate this workflow to larger codes.

Spack environments are the foundation of Spack CI

- `spack ci` enables any environment to be turned into a build pipeline
- Pipeline generates a `.gitlab-ci.yml` file from spack.lock
- Pipelines can be used just to build, or to generate relocatable binary packages
  - Binary packages can be used to keep the same build from running twice
- Same repository used for spack.yaml can generate pipelines for project

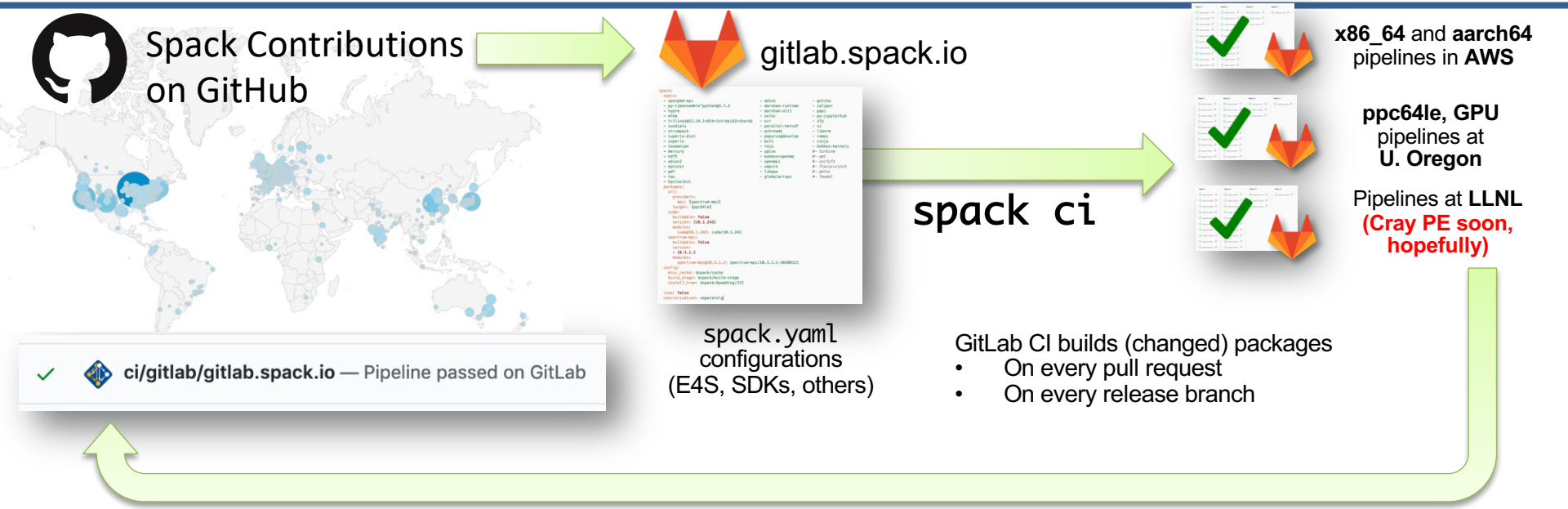
```yaml
spack:
  definitions:
  - pkgs:
    - readline@7.0
  - compilers:
    - '%gcc@5.5.0'
  - oses:
    - os=ubuntu18.04
    - os=centos7
  specs:
  - matrix:
    - [$pkgs]
    - [$compilers]
    - [$oses]
  mirrors:
    cloud_gitlab: https://mirror.spack.io
  gitlab-ci:
    mappings:
      - spack-cloud-ubuntu:
          match:
            - os=ubuntu18.04
          runner-attributes:
            tags:
              - spack-k8s
            image: spack/spack_builder_ubuntu_18.0
      - spack-cloud-centos:
          match:
            - os=centos7
          runner-attributes:
            tags:
              - spack-k8s
            image: spack/spack_builder_centos_7
  cdash:
    build-group: Release Testing
    url: https://cdash.spack.io
    project: Spack
    site: Spack AWS Gitlab Instance
```

spack.yaml



Parallel GitLab build pipeline

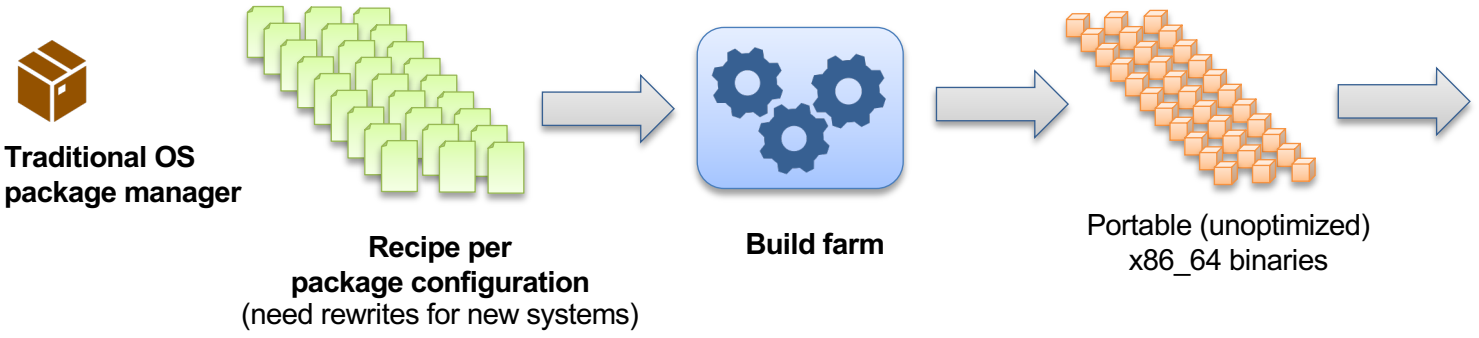
We are building a supply chain for HPC

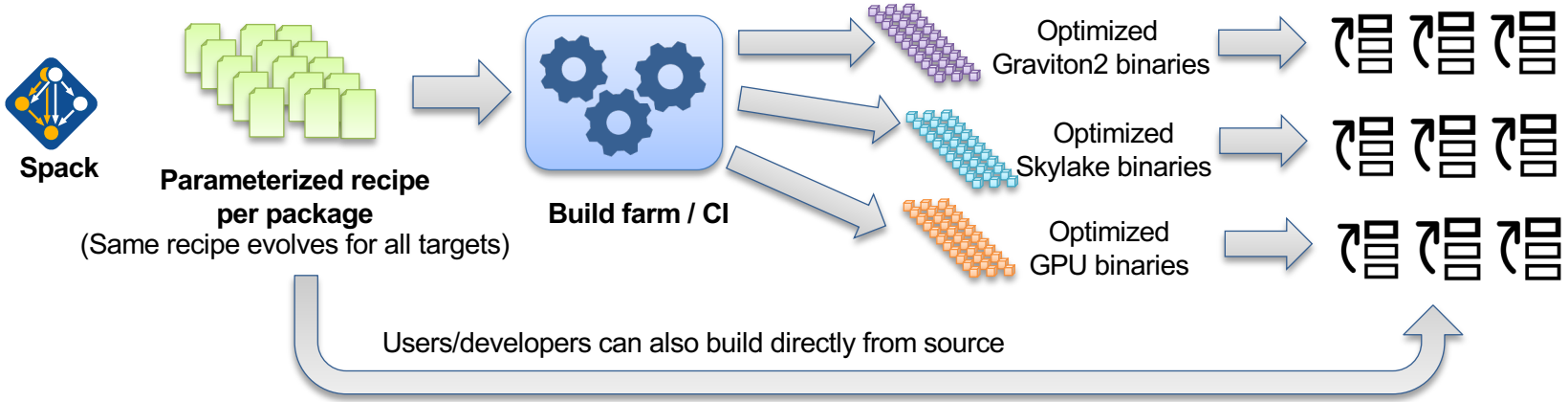


- **New security model supports untrusted contributions from forks**
  - Sandboxed build caches for test builds; Authoritative builds on mainline only after approved merge

This CI has *greatly* increased reliability of builds for users

Spack's model lowers the maintenance burden of optimized software stacks





We'll resume at: 12:00pm EST

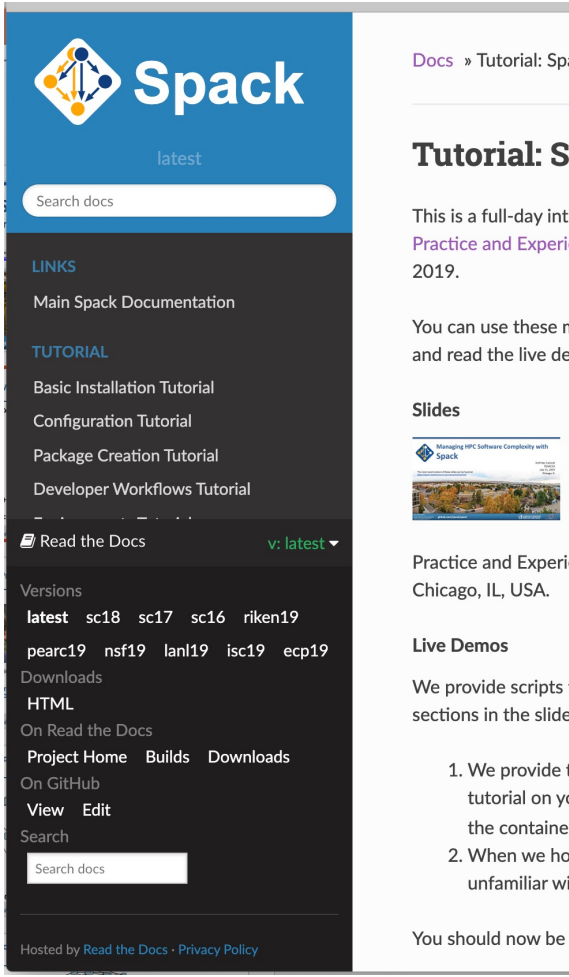
Find the slides and associated scripts here:

spack-tutorial.readthedocs.io

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

slack.spack.io

Join the "tutorial" channel!



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

Follow script at **spack-tutorial.readthedocs.io**

Hands-on Time: Configuration

Follow script at spack-tutorial.readthedocs.io

Hands-on Time: Creating Packages

Follow script at spack-tutorial.readthedocs.io

Hands-on Time:
Developer Workflows

Follow script at **spack-tutorial.readthedocs.io**

Hands-on Time:
Binary Caches and Mirrors

Follow script at **spack-tutorial.readthedocs.io**

Hands-on Time:
Stacks

Follow script at spack-tutorial.readthedocs.io

Hands-on Time:
Scripting

Follow script at spack-tutorial.readthedocs.io

More Features and the Road Ahead

Spack v0.17.0 was just released!

Major new features:

1. New Concretizer is now default
2. Binary bootstrapping enables us to get up and running fast
3. `spack install --reuse` aggressively reuses installed packages
4. Improved error messages
5. Conditional variants for more expressive packages
6. Git commit versioning
7. Overrides for default config directories
8. Improvements to `spack containerize`
9. New commands for querying packages and tests by tag

- 5,969 packages (920 added since 0.16)
- **Full release notes:** https://github.com/spack/spack/releases/tag/v0.17.0

Conditional variants simplify packages

CudaPackage: a mix-in for packages that use CUDA

```
class CudaPackage(PackageBase):
    variant('cuda', default=False,
            description='Build with CUDA')

    variant('cuda_arch',
            description='CUDA architecture',
            values=any_combination_of(cuda_arch_values),
            when='+cuda')

    depends_on('cuda', when='+cuda')

    depends_on('cuda@9.0:',     when='cuda_arch=70')
    depends_on('cuda@9.0:',     when='cuda_arch=72')
    depends_on('cuda@10.0:',    when='cuda_arch=75')

    conflicts('%gcc@9:', when='+cuda ^cuda@:10.2.89 target=x86_64:')
    conflicts('%gcc@9:', when='+cuda ^cuda@:10.1.243 target=ppc64le:')
```

`cuda` is a variant (build option)

`cuda_arch` is only present if cuda is enabled

dependency on `cuda`, but only if `cuda` is enabled

constraints on `cuda` version

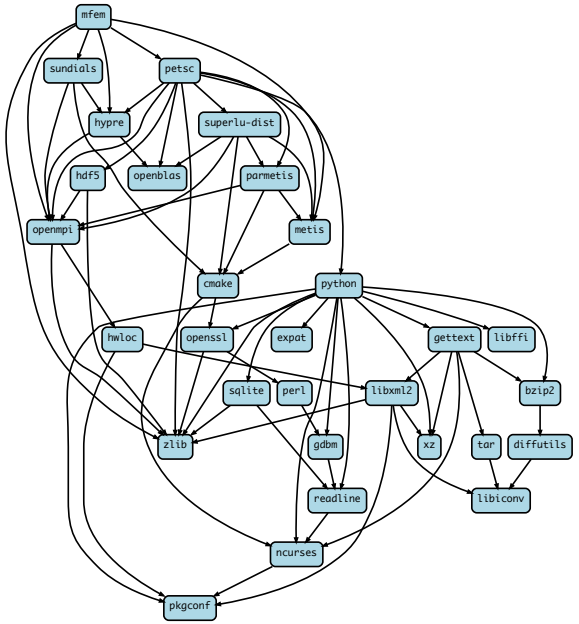
compiler support for `x86_64` and `ppc64le`

There is a lot of expressivity in this DSL.

Package solving is *combinatorial search* with *constraints* and *optimization*

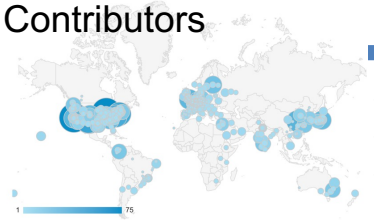
This problem is NP-hard!

- Search over a solution space:
  - Possible dependency graphs (nodes, edges)
  - Assignment of node and edge attributes
    - Version
    - Dependency, dependency type
    - Compiler, compiler version
    - Target
    - Compiler, compiler version
- Subject to validity constraints:
  - Version requirements
  - Target/compiler compatibility
  - Virtual providers
- Optimization picks “best” among valid solutions:
  - Most recent versions
  - Preferred variant values
  - Preferred compilers that support best targets (e.g., AVX-512)
  - Minimize number of builds



High level view of a Spack package build

Contributors



- new versions
- new dependencies
- new constraints

package.py repository

concretizer

spack developers

yaml default config `packages.yaml`

admins, users

yaml local preferences config `packages.yaml`

users

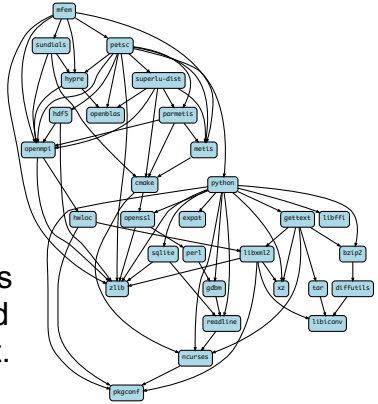
yaml local environment config `spack.yaml`

users

Command line constraints

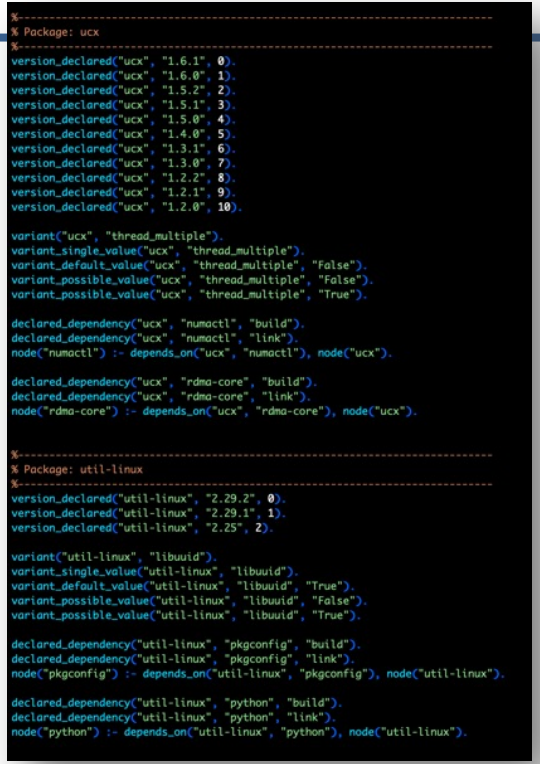
```
spack install hdf5@1.12.0 +debug
```

Concrete spec is fully constrained and can be built.



The new concretizer is now default in 0.17

- New concretizer leverages Clingo (see potassco.org)
- Clingo is an Answer Set Programming (ASP) solver
  - ASP looks like Prolog; leverages SAT solvers for speed/correctness
  - ASP program has 2 parts:
    1. Large list of facts generated from our package repositories and config
       - 20,000 – 30,000 facts is typical – includes dependencies, options, etc.
    2. Small logic program (~800 lines), including constraints and optimization criteria
- New algorithm on the Spack side is conceptually simpler:
  - Generate facts for all possible dependencies, send to logic program
  - Optimization criteria express preferences more clearly
  - Build a DAG from the results
- New concretizer solves many specs that current concretizer can't
  - Backtracking is a huge win – many issues resolved
  - Currently requires user to install clingo with Spack
  - Solver will be automatically installed from public binaries in 0.17.0



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

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

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

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


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

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

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

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

Some facts for the HDF5 package

With and without reuse optimization

Note the bifurcated optimization criteria

```
(spackle):solver> spack solve -Il hdf5
==> Best of 9 considered solutions.
==> Optimization Criteria:
  Priority  Criterion                                                  Installed  ToBuild
  1         number of packages to build (vs. reuse)                            -       20
  2         deprecated versions used                                           0        0
  3         version weight                                                     0        0
  4         number of non-default variants (roots)                             0        0
  5         preferred providers for roots                                      0        0
  6         default values of variants not being used (roots)                  0        0
  7         number of non-default variants (non-roots)                         0        0
  8         preferred providers (non-roots)                                    0        0
  9         compiler mismatches                                                0        0
  10        OS mismatches                                                      0        0
  11        non-preferred OS's                                                 0        0
  12        version badness                                                    0        2
  13        default values of variants not being used (non-roots)              0        0
  14        non-preferred compilers                                            0        0
  15        target mismatches                                                  0        0
  16        non-preferred targets                                              0        0

  -  zzngfs3  hdf5@1.10.7%apple-clang@13.0.0~cxx~fortran~hl~ipo~java+mpi+shared~szip~threadsafe+tools api=default b
  -  nsylovq      ^cmake@3.21.4%apple-clang@13.0.0~doc+ncurses+openssl+ownlibs~qt build_type=Release arch=darwin-bi
  -  xdbaqeo          ^ncurses@6.2%apple-clang@13.0.0~symlinks+termlib abi=none arch=darwin-bigsur-skylake
  -  kfureok              ^pkgconf@1.8.0%apple-clang@13.0.0 arch=darwin-bigsur-skylake
  -  5ekd4ap          ^openssl@1.1.1l%apple-clang@13.0.0~docs certs=system arch=darwin-bigsur-skylake
  -  xz6a265              ^perl@5.34.0%apple-clang@13.0.0+cpanm+shared+threads arch=darwin-bigsur-skylake
  -  xgt3tls                  ^berkeley-db@18.1.40%apple-clang@13.0.0+cxx~docs+stl patches=b231fcc4d5cff05e5c3a4814
  -  65edjf6                  ^bzip2@1.0.8%apple-clang@13.0.0~debug~pic+shared arch=darwin-bigsur-skylake
  -  662adoo                      ^diffutils@3.8%apple-clang@13.0.0 arch=darwin-bigsur-skylake
  -  fu7tfsr                          ^libiconv@1.16%apple-clang@13.0.0 libs=shared,static arch=darwin-bigsur-skyla
  -  vjg67nd                  ^gdbm@1.19%apple-clang@13.0.0 arch=darwin-bigsur-skylake
  -  tjceldr                      ^readline@8.1%apple-clang@13.0.0 arch=darwin-bigsur-skylake
  -  xevvljj                  ^zlib@1.2.11%apple-clang@13.0.0+optimize+pic+shared arch=darwin-bigsur-skylake
  -  xelfobh      ^openmpi@4.1.1%apple-clang@13.0.0~atomics~cuda~cxx~cxx_exceptions+gpfs~internal-hwloc~java~legacy
  -  zruns75          ^hwloc@2.6.0%apple-clang@13.0.0~cairo~cuda~gl~libudev+libxml2~netloc~nvml~opencl~pci~rocm+sha
  -  ib4fnkf              ^libxml2@2.9.12%apple-clang@13.0.0~python arch=darwin-bigsur-skylake
  -  dwiv2ys                  ^xz@5.2.5%apple-clang@13.0.0~pic libs=shared,static arch=darwin-bigsur-skylake
  -  blitnbl          ^libevent@2.1.12%apple-clang@13.0.0+openssl arch=darwin-bigsur-skylake
  -  h7jalyu          ^openssh@8.7p1%apple-clang@13.0.0 arch=darwin-bigsur-skylake
  -  7v7bqx2              ^libedit@3.1-20210216%apple-clang@13.0.0 arch=darwin-bigsur-skylake
```

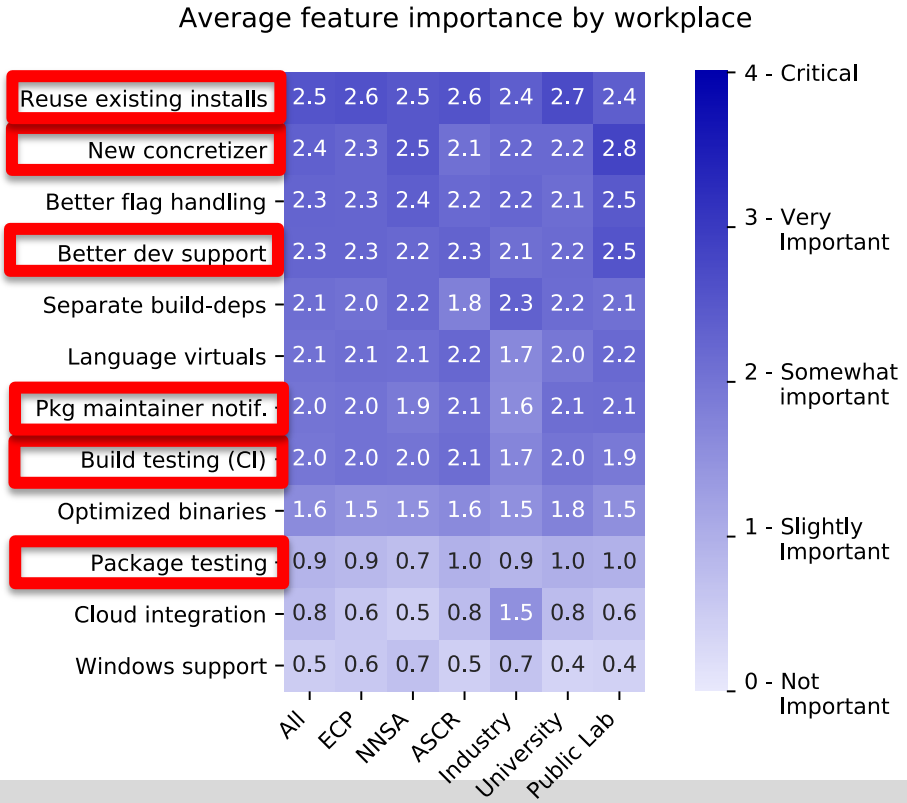
Pure hash-based reuse: all misses

```
(spackle):spack> spack solve --reuse -Il hdf5
==> Best of 10 considered solutions.
==> Optimization Criteria:
  Priority  Criterion                                                  Installed  ToBuild
  1         number of packages to build (vs. reuse)                            -        4
  2         deprecated versions used                                           0        0
  3         version weight                                                     0        0
  4         number of non-default variants (roots)                             0        0
  5         preferred providers for roots                                      0        0
  6         default values of variants not being used (roots)                  0        0
  7         number of non-default variants (non-roots)                         2        0
  8         preferred providers (non-roots)                                    0        0
  9         compiler mismatches                                                0        0
  10        OS mismatches                                                      0        0
  11        non-preferred OS's                                                 0        0
  12        version badness                                                    6        0
  13        default values of variants not being used (non-roots)              1        0
  14        non-preferred compilers                                           15        4
  15        target mismatches                                                  0        0
  16        non-preferred targets                                              0        0

  -  yfkfnsp  hdf5@1.10.7%apple-clang@12.0.5~cxx~fortran~hl~ipo~java+mpi+shared~szip~threadsafe+tools api=defaul
 [+] zd4m26e      ^cmake@3.21.1%apple-clang@12.0.5~doc+ncurses+openssl+ownlibs~qt build_type=Release arch=darwin
 [+] 53i52xr          ^ncurses@6.2%apple-clang@12.0.5~symlinks+termlib abi=none arch=darwin-bigsur-skylake
 [+] us36bwr          ^openssl@1.1.1l%apple-clang@12.0.5~docs+systemcerts arch=darwin-bigsur-skylake
 [+] 74mwnxg              ^zlib@1.2.11%apple-clang@12.0.5+optimize+pic+shared arch=darwin-bigsur-skylake
  -  3ijfnel          ^openmpi@4.1.1%apple-clang@12.0.5~atomics~cuda~cxx~cxx_exceptions+gpfs~internal-hwloc~java~leg
  -  jxexyb7              ^hwloc@2.6.0%apple-clang@12.0.5~cairo~cuda~gl~libudev+libxml2~netloc~nvml~opencl~pci~rocm+
 [+] ckdn5zf                  ^libxml2@2.9.12%apple-clang@12.0.5~python arch=darwin-bigsur-skylake
 [+] k7auat3                      ^libiconv@1.16%apple-clang@12.0.5 libs=shared,static arch=darwin-bigsur-skylake
 [+] kzyumgx                      ^xz@5.2.5%apple-clang@12.0.5~pic libs=shared,static arch=darwin-bigsur-skylake
 [+] grgtlcd                  ^pkgconf@1.8.0%apple-clang@12.0.5 arch=darwin-bigsur-skylake
  -  nnc66ug              ^libevent@2.1.12%apple-clang@12.0.5+openssl arch=darwin-bigsur-skylake
 [+] 63xbksk              ^openssh@8.6p1%apple-clang@12.0.5 arch=darwin-bigsur-skylake
 [+] snhgldt                  ^libedit@3.1-20210216%apple-clang@12.0.5 arch=darwin-bigsur-skylake
 [+] qbkmtdd              ^perl@5.34.0%apple-clang@12.0.5+cpanm+shared+threads arch=darwin-bigsur-skylake
 [+] tnvkifs                  ^berkeley-db@18.1.40%apple-clang@12.0.5+cxx~docs+stl patches=b231fcc4d5cff05e5c3a4814f
 [+] 7d5woqt                  ^bzip2@1.0.8%apple-clang@12.0.5~debug~pic+shared arch=darwin-bigsur-skylake
 [+] vh6di3i                  ^gdbm@1.19%apple-clang@12.0.5 arch=darwin-bigsur-skylake
 [+] qgy3v4l                      ^readline@8.1%apple-clang@12.0.5 arch=darwin-bigsur-skylake
```

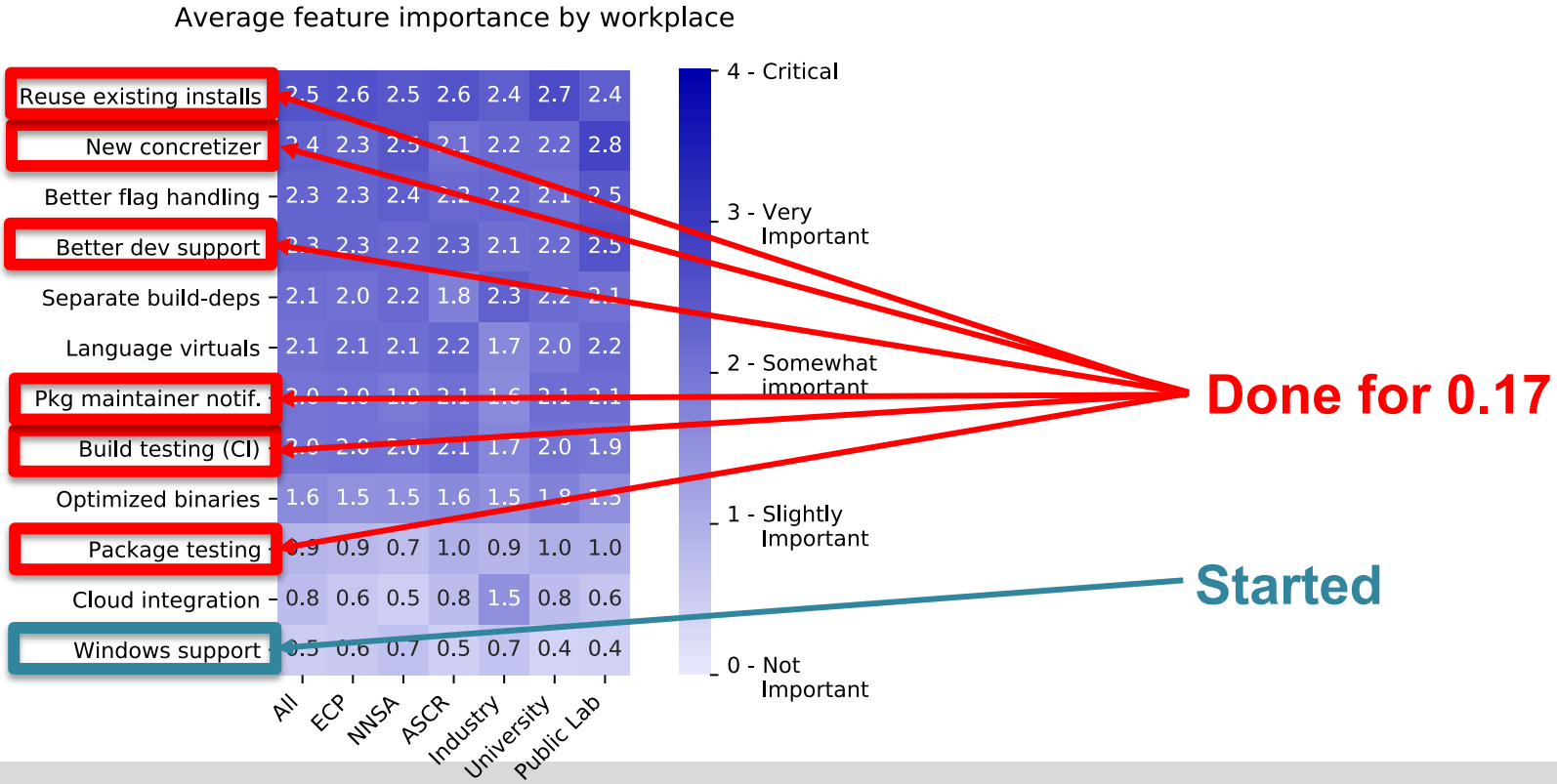
With reuse: 16 packages were actually acceptable

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



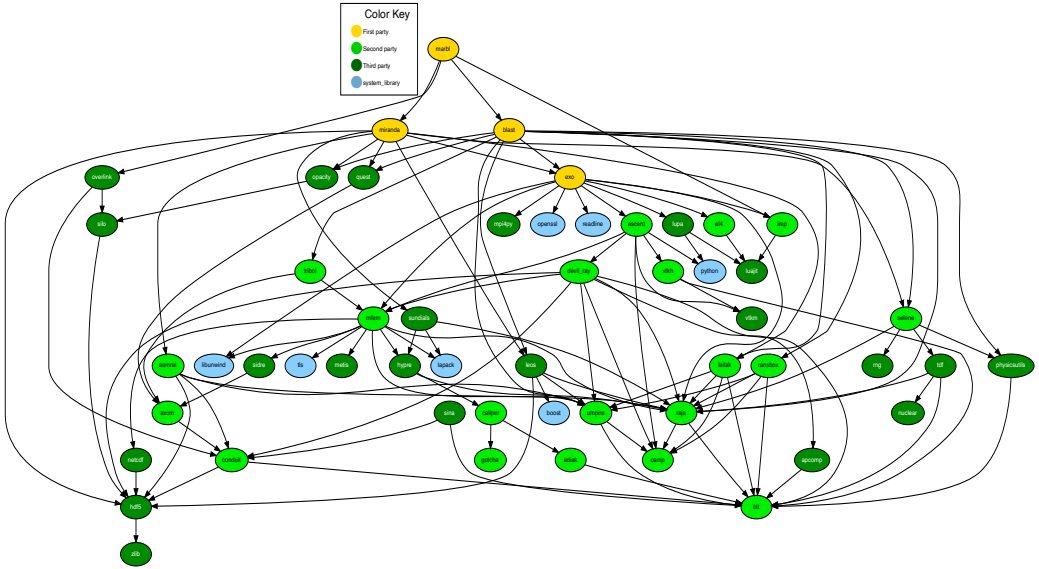
- Complexity of packages in Spack is increasing
  - many more package solves require backtracking than a year ago
  - Many variants, conditional dependencies, special compiler requirements
- More aggressive reuse of existing installs requires better dependency resolution
  - Need to be able to analyze how to configure the build to work with installed packages
- Separate resolution of build dependencies also requires a more sophisticated solver
  - Makes the solve even more combinatorial
  - Needed to support mixed compilers, version conflicts between different package's build requirements

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



We have recently introduced some new features to support the development model of MARBL, an LLNL multi-physics code

- Not unlike other LLNL codes, but…
- MARBL is more deeply modular than prior codes
  - Designed to support modular *physics*
  - MARBL itself has two hydro options: Miranda & Blast
  - Code, build structure both assume that a simulation is comprised of *packages*
- Needed a way to simplify modular workflows
  - Need to work on several repos at once
  - Changes to the code are multiple pull requests
- LLNL doesn’t (likely won’t) use mono-repos
  - Issues:
    - Managing permissions
    - Code timescales
    - Independence of teams
- MARBL built MBS: a better poly-repo approach



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

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

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



$ ls
spack.yaml    axom/    mfem/



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

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

We have added git versioning to Spack

- Users can now specify a full, 40-char git commit as a version
  - Works in environments or on the command line

```
$ spack install zlib @53ce2713117ef2a8ed682d77b944df991c499252
```

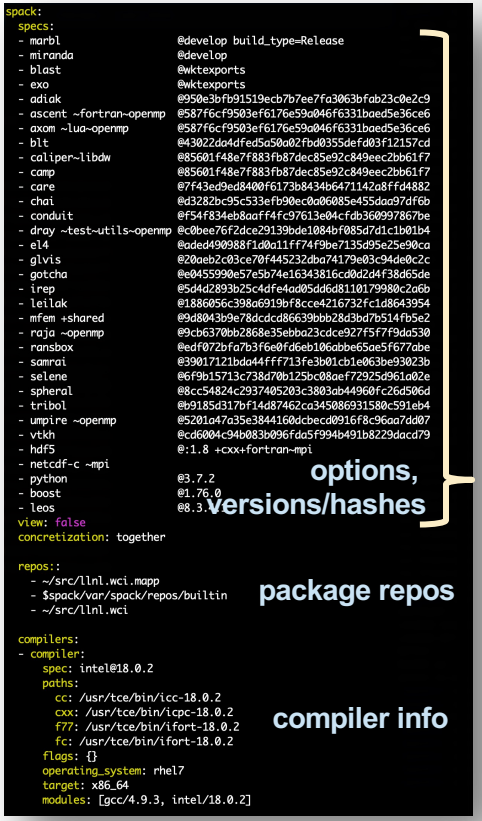
- This was tricky because we needed a way to compare a commit to a version
  - MBS only needs to be able to fetch by commit, not compare
  - Packages have conditional logic with versions
  - We can compare versions to commits based on tags in a repository
- We developed an internal representation for commit versions
  - Lexicographic tuple comparison:

```
(<version>, "", <commits since prior tag>)
```

  - Comes before any <version>.x
  - Allows commits to be compared by distance between versions.

Using git versioning, we've been able to support MARBL's developer workflow

- First section is familiar
  - List of packages with hashes
- `spack.yaml` ties the modular MARBL code together:
  - hashes
  - parts of exo/build directory
- Some differences:
  - Packages in Spack are configurable
  - Can set per-package options
  - Compiler options, flags are configurable in Spack environments
- If this is too long, some of this can be moved to external includes



Current MARBL spack.ya

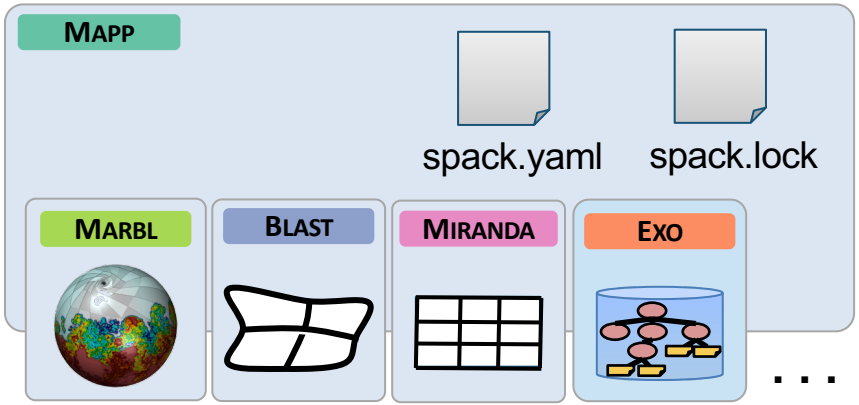
Spack workflow for developer environment

Spack

```
$ git clone ssh://git@rzgitlab.llnl.gov:7999/mapp/mapp
$ cd mapp
$ spack env activate .
$ spack develop marbl@develop
$ spack develop blast@develop
$ spack develop miranda@develop
$ spack develop exo@develop
$ srun –N 2 –n 16 --exclusive spack install
```

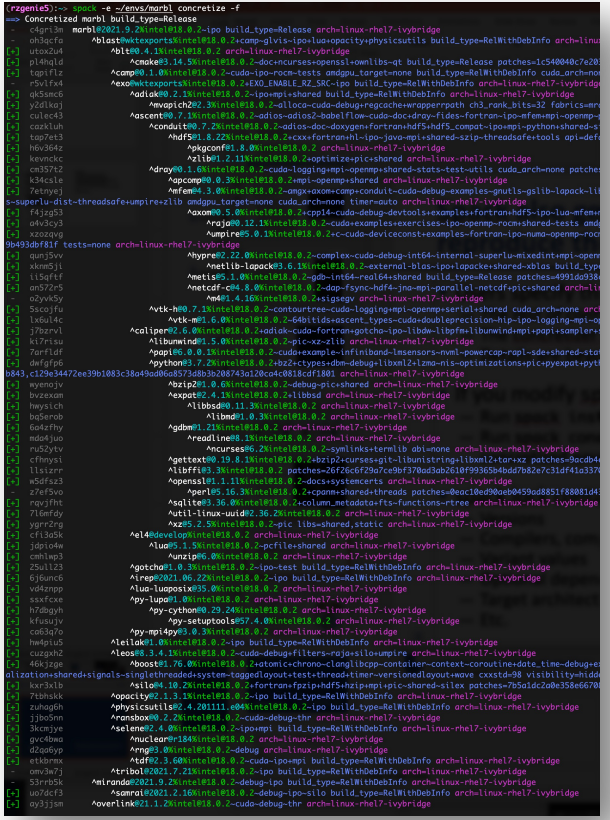
We can find ways to shorten this

spack can do multi-node builds



Spack generates a spack.lock file that enables you to reproduce the environment

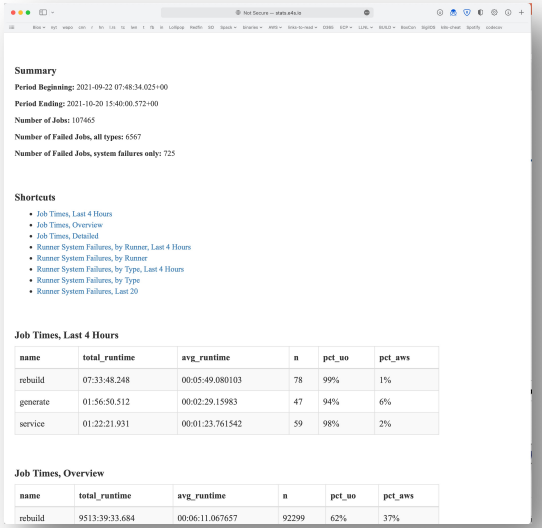
- Users specify their constraints in spack.yaml
  - The rest of configuration is automated by the *concretizer*
  - The concretizer is a constraint solver that reconciles package requirements with yours
  - Details are beyond the scope of this presentation
- If you modify `spack.yaml`, you can either:
  - Run **`spack install`** again (this concretizes before installing)
  - Run **`spack concretize --force`** to see the concretized environment before installing (shown at right)
- spack.lock contains all the decisions the concretizer made:
  - Versions
  - Compilers, compiler versions
  - Variant values
  - Optional dependencies
  - Target architecture
- Open question: how best to manage `spack.lock` files



Fully concretized MARBL environment

Future CI directions focus on scalability and testing

- Scaling tests up to handle every PR has been very difficult
  - Driven by GitLab
  - Using Kubernetes builders
  - Using a cluster at U. Oregon
- Concretization of large environments was slowing turnaround
  - 55 min to concretize E4S environment (each spec separately)
  - Brought this down to 2.5 min with parallelization and caching
- Amazon and E4S/UO team helping to pinpoint errors
- We are now doing about 100,000 builds/month
- Once we have a stable, rolling release of spack develop branch, we'll make the build cache public
  - Rolling binaries for develop
  - Long-lived snapshots for each release



Summary

Period Beginning: 2021-09-22 07:48:34.025+00

Period Ending: 2021-10-20 15:40:00.572+00

Number of Jobs: 107465

Number of Failed Jobs, all types: 6567

Number of Failed Jobs, system failures only: 725

Shortcuts

- Job Times, Last 4 Hours
- Job Times, Overview
- Job Times, Detailed
- Runner System Failures, by Runner, Last 4 Hours
- Runner System Failures, by Runner
- Runner System Failures, by Type, Last 4 Hours
- Runner System Failures, by Type
- Runner System Failures, Last 20

Job Times, Last 4 Hours

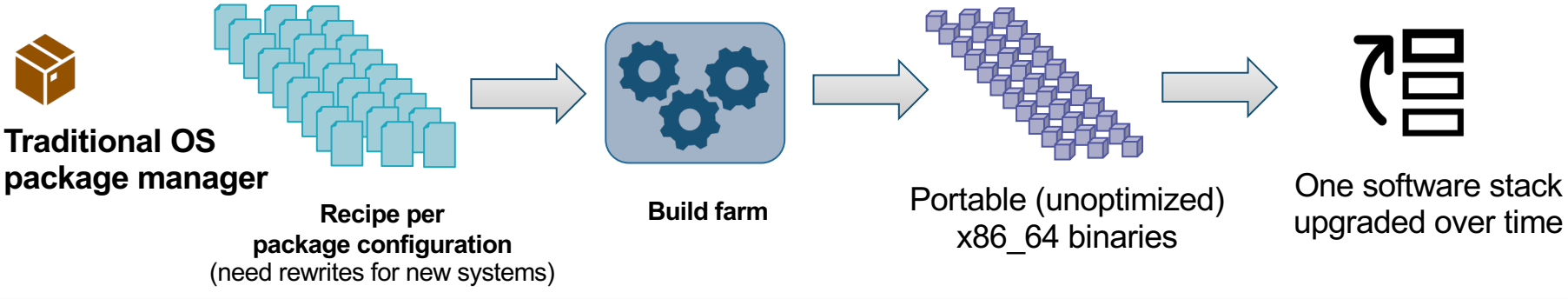
name	total_runtime	avg_runtime	n	pct_uo	pct_aws
rebuild	07:33:48.248	00:05:49.080103	78	99%	1%
generate	01:56:50.512	00:02:29.15983	47	94%	6%
service	01:22:21.931	00:01:23.761542	59	98%	2%

Job Times, Overview

name	total_runtime	avg_runtime	n	pct_uo	pct_aws
rebuild	9513:39:33.684	00:06:11.067657	92299	62%	37%

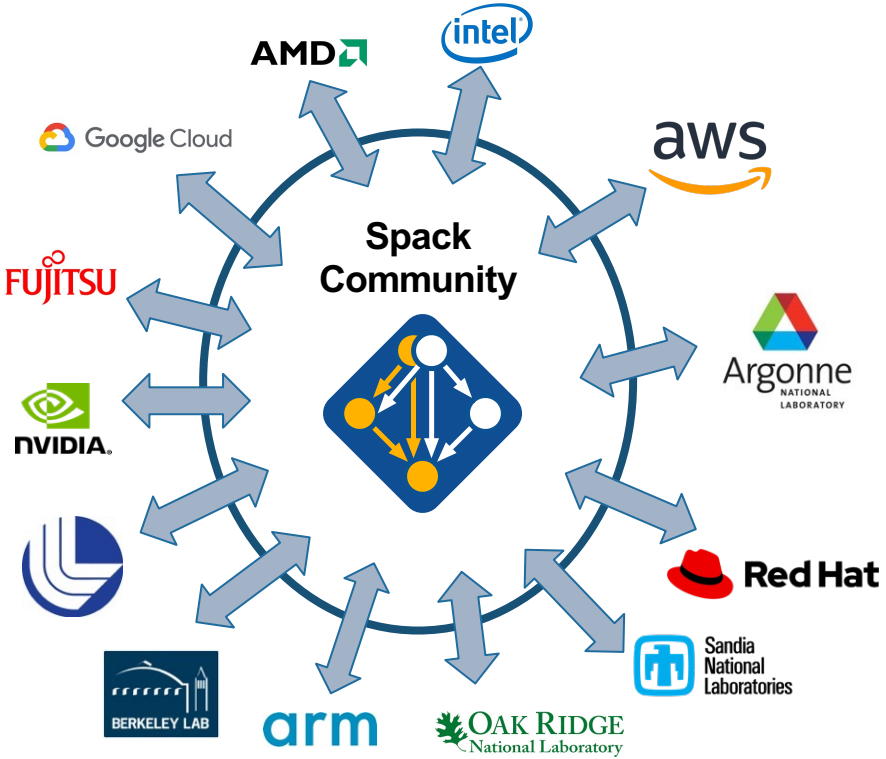
http://stats.e4s.io

Spack's model lowers the maintenance burden of optimized software stacks



Spack's long-term strategy is based around broad adoption and collaboration

- **Not sustainable without a community**
  - Broad adoption incentivizes contributors
  - Cloud resources and automation absolutely necessary
- **Spack preserves build knowledge in a cross-platform, reusable way**
  - Minimize rewriting recipes when porting
- **CI ensures builds continue to work as packages evolve**
  - Keep packages flexible but verify key configurations
- **Any suggestions on sustainability models would be appreciated!**



Spack 0.18 Roadmap: public build cache, reuse by default

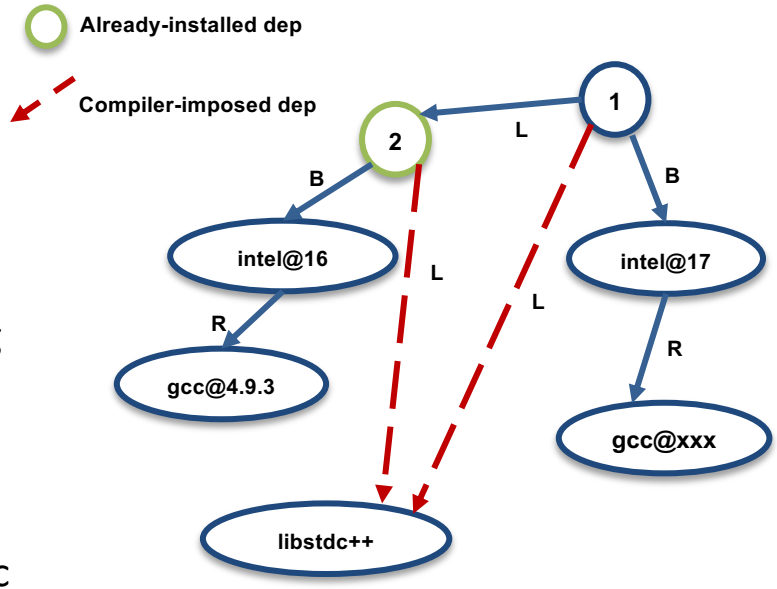
- Spack v0.18 will be released in the next week or two

- Major features:
  - Public binary mirror (TBD at ISC 2022)
  - Default concretization mode will aggressively **reuse** dependencies
  - Specs will retain full provenance in the database
  - New testing features (running tests in CI pipelines)
  - Introspect Cray environments to find externals automatically

- Additional features
  - Many improvements to binary management
  - New parallel build features for environments
  - Many concretization improvements

Spack 0.19 Roadmap: compilers as dependencies

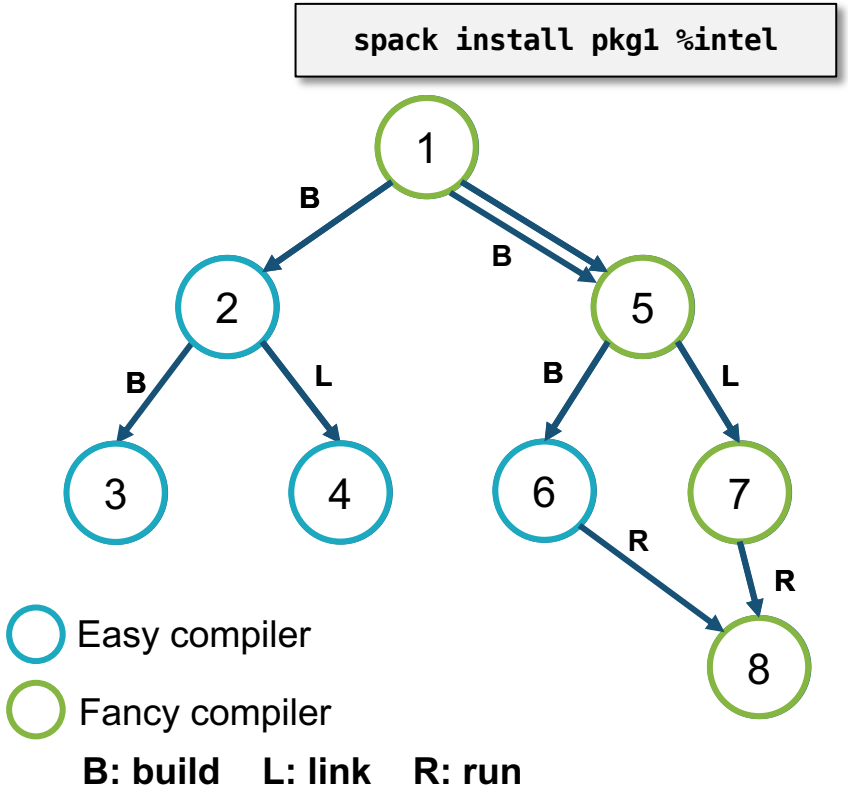
- **We need deeper modeling of compilers to handle compiler interoperability**
  - libstdc++, libc++ compatibility
  - Compilers that depend on compilers
  - Linking executables with multiple compilers
- **First prototype is complete!**
  - We've done successful builds of some packages using compilers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



Compilers and runtime libs fully modeled as dependencies

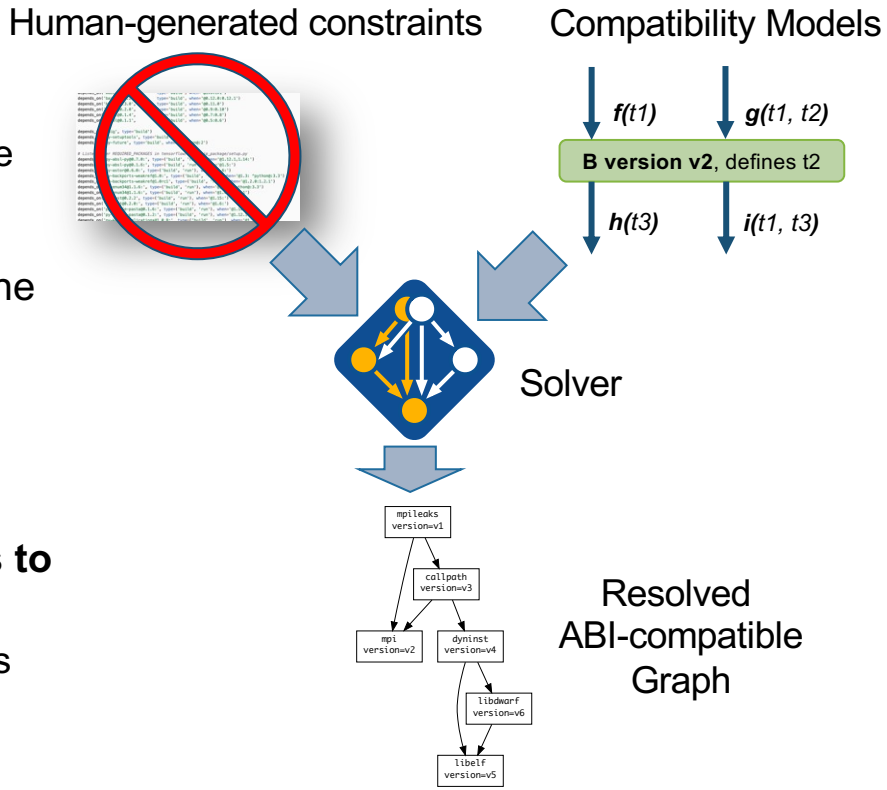
Separate concretization of build dependencies

- We want to:
  - Build build dependencies with the "easy" compilers
  - Build rest of DAG (the link/run dependencies) with the fancy compiler
- This required significant concretizer modifications
- Gets into issues like bootstrapping



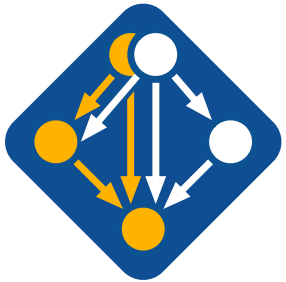
Ongoing research:
BUILD is a 3-year research project, started at LLNL in 2020

- Basic premise: humans can't generate all the compatibility constraints
  - Version ranges, conflicts, in Spack packages not precise
  - rely on maintainers to get right.
- BUILD aims to understand software compatibility at the binary level
  - Develop ABI compatibility models
  - Enable *automatic* and ABI-compatible reuse of system binaries, foreign binary packages
- **WIP: better dependency solvers can enable users to solve *around* system dependencies**
  - find "closest" match to a prior build, using new packages
  - Reproduce a prior build with new requirements



Join the Spack community!

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



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

★ Star us on GitHub!

github.com/spack/spack

Follow us on Twitter!

@spackpm

We hope to make distributing & using HPC software easy!