Managing HPC Software Complexity with Spack

CINECA Full Day Tutorial
February 13th, 2022

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

Tutorial Materials

Find these slides and associated scripts here:

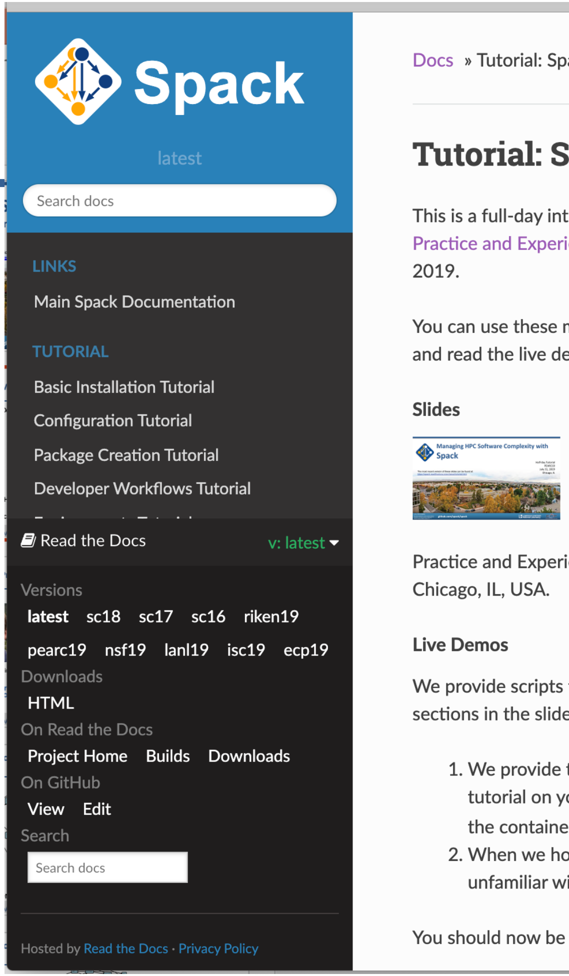
spack-tutorial.rtfd.io

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

slack.spack.io

Join the **#tutorial** channel!

You can ask questions here after the conference is over.
Over **2,000 people** can help you on Slack!



Spack

latest

Search docs

LINKS

Main Spack Documentation

TUTORIAL

Basic Installation Tutorial

Configuration Tutorial

Package Creation Tutorial

Developer Workflows Tutorial

Read the Docs v: latest

Versions
latest sc18 sc17 sc16 riken19
pearc19 nsf19 lanl19 isc19 ecp19

Downloads
HTML

On Read the Docs
Project Home Builds Downloads

On GitHub
View Edit

Search
Search docs

Hosted by Read the Docs · Privacy Policy

Docs » Tutorial: Sp

Tutorial: S

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

You can use these r
and read the live de

Slides

Practice and Experi
Chicago, IL, USA.

Live Demos

We provide scripts
sections in the slide

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

You should now be

Tutorial Presenters

Massimiliano Culpo
@alalazo

Harmen Stoppels
@haampie

Agenda (approximate)

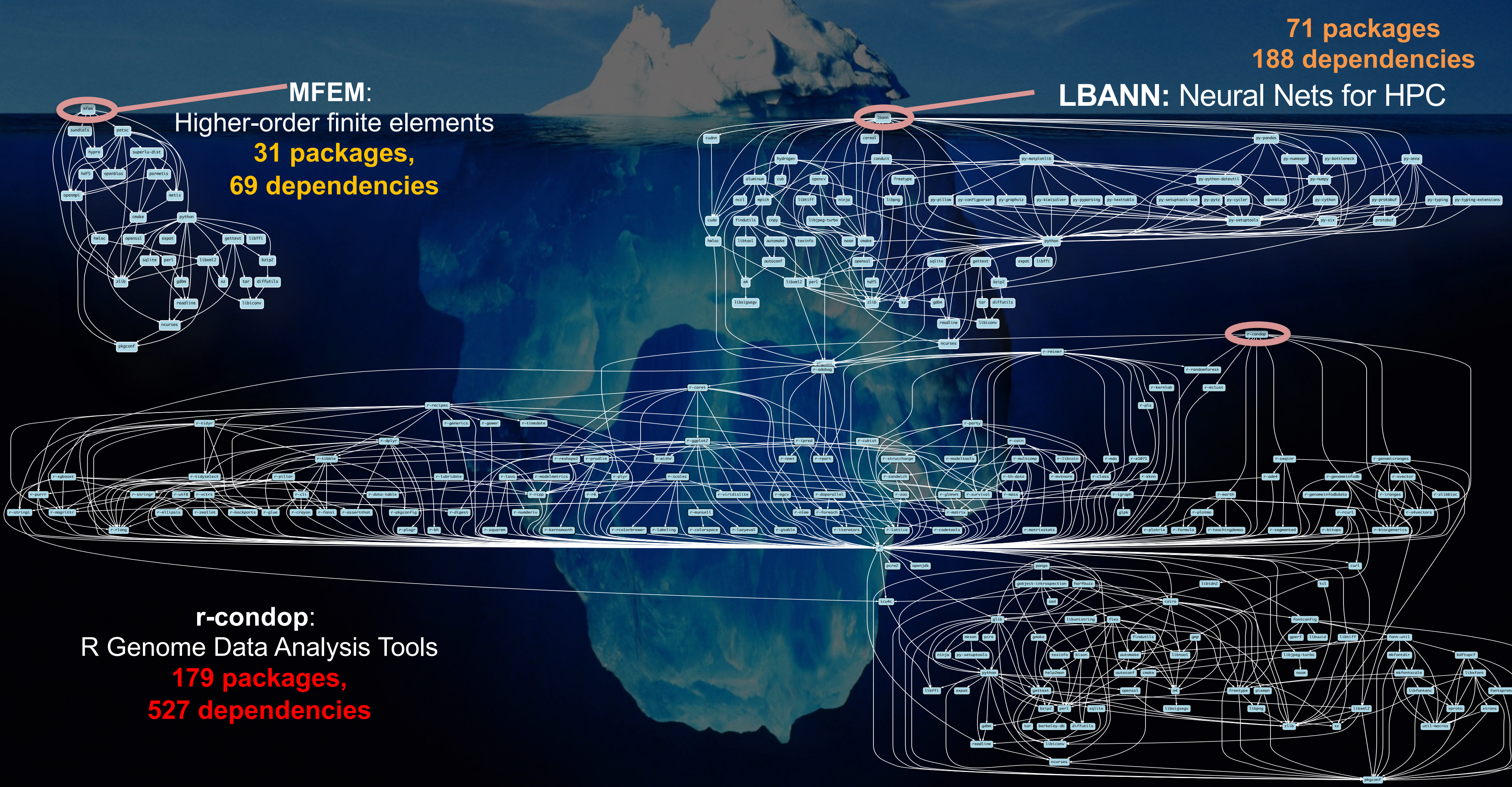
Morning

Intro	9:30 am
Basics	9:45 am
Concepts	10:30 am
Break	11:00 am
Environments	11:30 am
Configuration	12:15 am
Lunch	1:00 pm

Afternoon

Packaging	2:00 pm
Binary and Source Mirrors	3:00 pm
Break	3:30 pm
Stacks	4:00 pm
Developer workflows	5:00 pm
Roadmap / Questions	5:25 pm
End	5:30 pm

Modern scientific codes rely on icebergs of dependency libraries

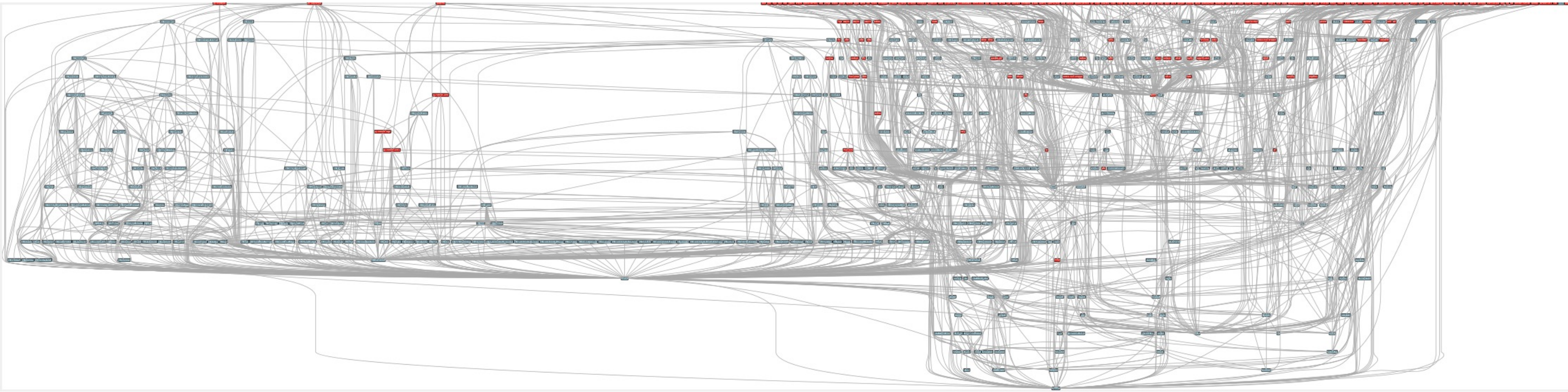


MFEM:
Higher-order finite elements
31 packages,
69 dependencies

71 packages
188 dependencies
LBANN: Neural Nets for HPC

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

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

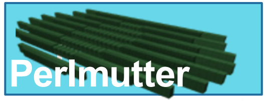
- **Code is typically distributed as source**
  - With exception of vendor libraries, compilers
- **Often build many variants of the same package**
  - Developers' builds may be very different
  - Many first-time builds when machines are new
- **Code is optimized for the processor and GPU**
  - Must 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



Lawrence Berkeley National Lab
AMD Zen / NVIDIA

Argonne National Lab
Intel Xeon / Xe

Upcoming

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

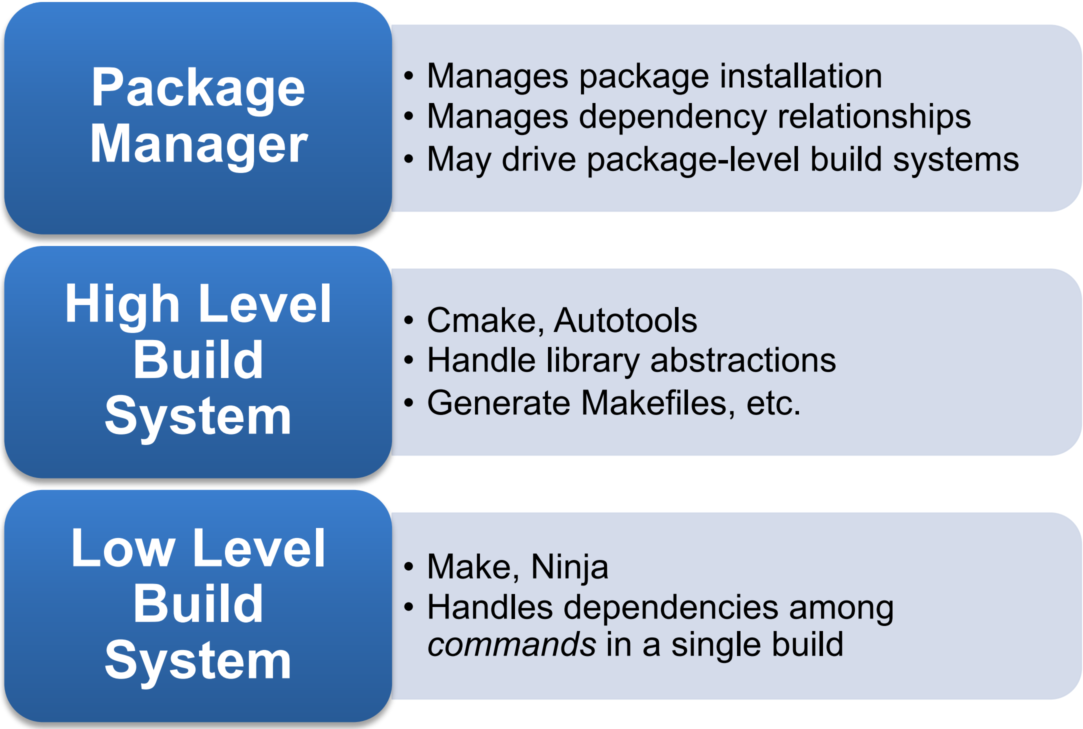
- Ease of use of mainstream tools, with flexibility needed for HPC
- In addition to CLI, Spack also:
  - Generates (but does **not** require) *modules*
  - Allows conda/virtualenv-like *environments*
  - Provides many devops features (CI, container generation, more)

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



Package Manager

- Manages package installation
- Manages dependency relationships
- May drive package-level build systems

High Level Build System

- Cmake, Autotools
- Handle library abstractions
- Generate Makefiles, etc.

Low Level Build System

- Make, Ninja
- Handles dependencies among *commands* in a single build

Who can use Spack?

People who want to use or distribute software for HPC!

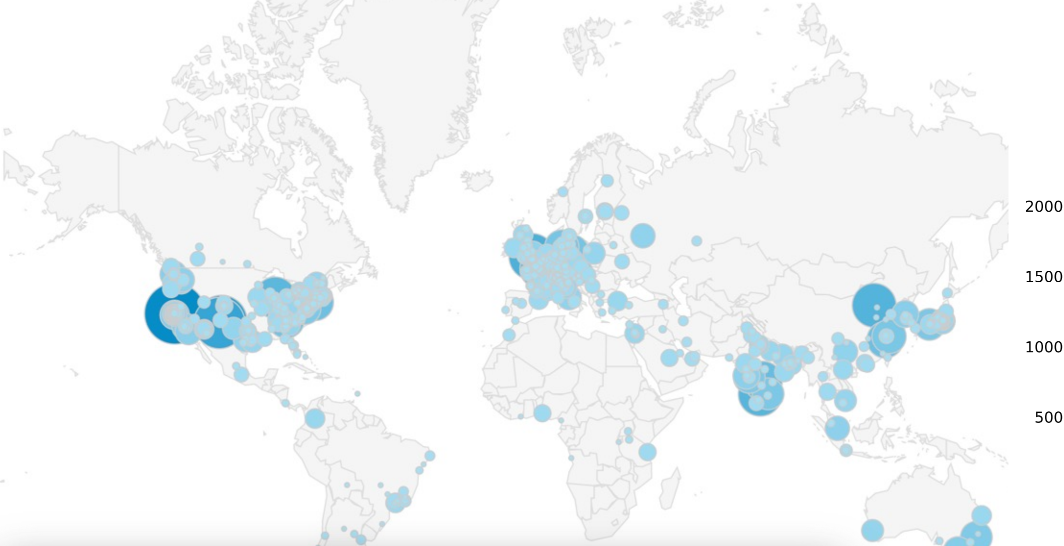
1. **End Users of HPC Software**
   - Install and run HPC applications and tools

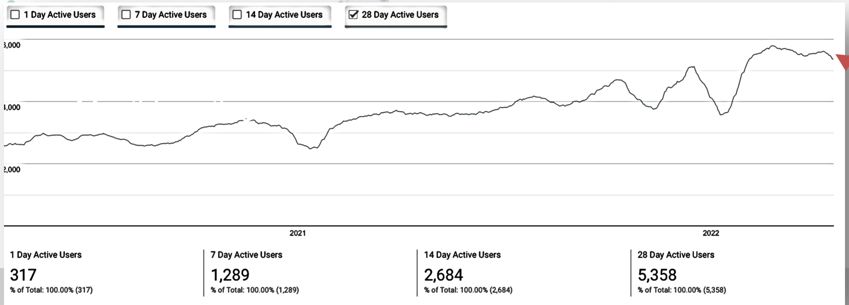
2. **HPC Application Teams**
   - Manage third-party dependency libraries

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





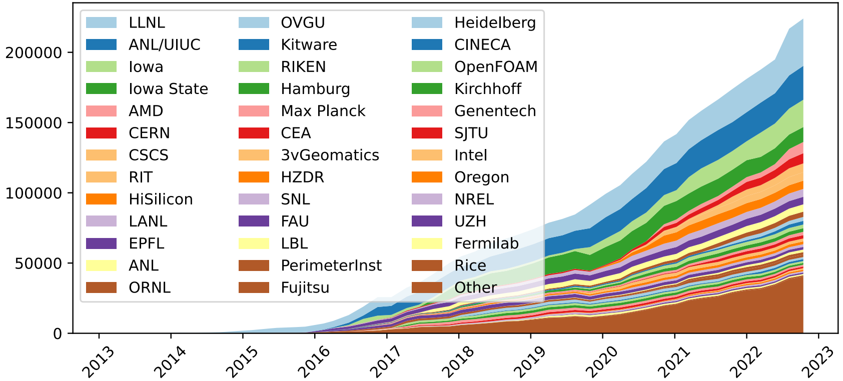
☐ 1 Day Active Users ☐ 7 Day Active Users ☐ 14 Day Active Users ☑ 28 Day Active Users

1 Day Active Users	7 Day Active Users	14 Day Active Users	28 Day Active Users
317	1,289	2,684	5,358
% of Total: 100.00% (317)	% of Total: 100.00% (1,289)	% of Total: 100.00% (2,684)	% of Total: 100.00% (5,358)

Over 6,700 software packages

Over 1,100 contributors

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



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

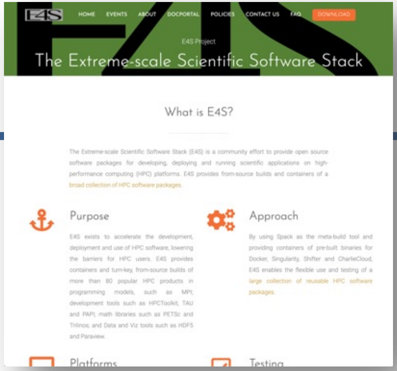
Nearly 6,000 monthly active users
(per documentation site)

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

ECP

EXASCALE COMPUTING PROJECT

- 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.



E4S Project

The Extreme-scale Scientific Software Stack

What is E4S?

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

Purpose

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

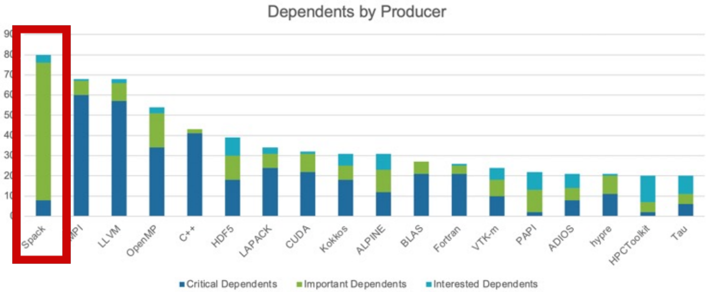
Approach

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

Platforms

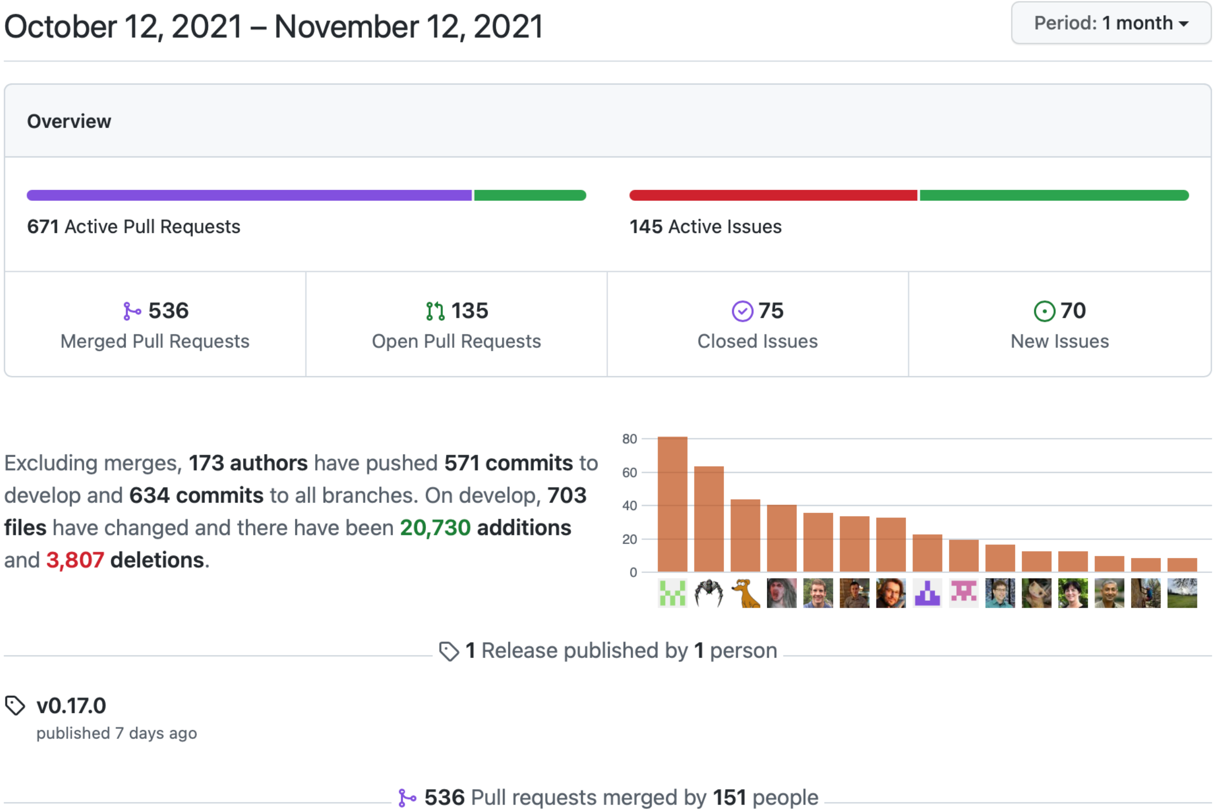
Testing

https://e4s.io



Spack 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 drawn contributions and collaborations with 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 v0.19.1 was released last week!

- Major new features in v0.19:
  1. **Package requirements**
  2. **Environment UI improvements**
  3. **Packages with multiple build systems**
  4. Compiler/variant propagation
  5. Enhanced git versions
  6. Better Cray EX Support
  7. Testing and CI improvements
  8. Experimental binding link model

***Bold items covered in today's tutorial**

github.com/spack/spack

Spack is not the only tool that automates builds

1. **“Functional” Package Managers**
   - Nix
   - GNU Guix

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

2. **Build-from-source Package Managers**
   - Homebrew, LinuxBrew
   - MacPorts
   - Gentoo

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

Other tools in the HPC Space:

- **Easybuild**
  - An installation tool for HPC
  - Focused on HPC system administrators – different package model from Spack
  - Relies on a fixed software stack – harder to tweak recipes for experimentation

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

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

https://conda.io

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 provides a *spec* syntax to describe customized package 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=cascadelake      set target microarchitecture
$ spack install mpileaks@3.3 ^mpich@3.2 %gcc@4.9.3   ^ dependency constraints
```

- Each expression is a ***spec*** for a particular configuration
  - Each clause adds a constraint to the spec
  - Constraints are optional – specify only what you need.
  - Customize install on the command line!

- Spec syntax is recursive
  - Full control over the combinatorial build space

Spack packages are *parameterized* using the spec syntax

Python DSL defines many ways to build

```
from spack import *

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

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

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

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

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

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

    def install(self, spec, prefix):
        mkdirp(prefix.bin)
        install('../spack-build/kripke', prefix.bin)
```

Base package (CMake support)

Metadata at the class level

Versions

Variants (build options)

Dependencies (same spec syntax)

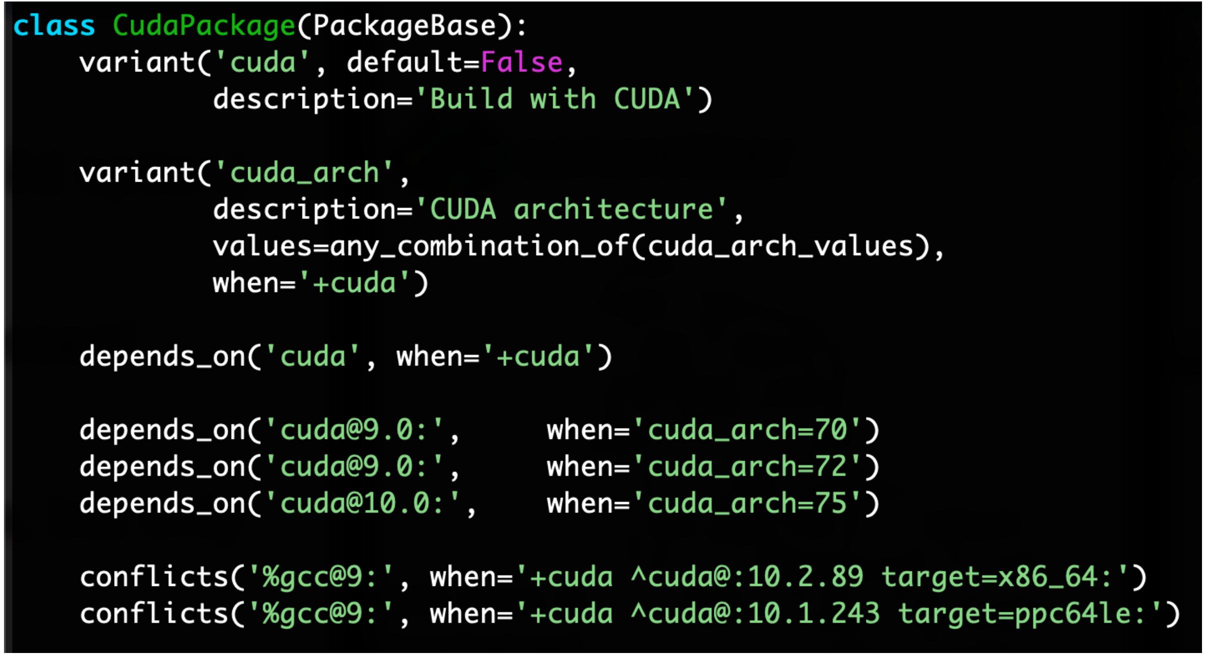
Install logic in instance methods

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

***One* package.py file per software project!**

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

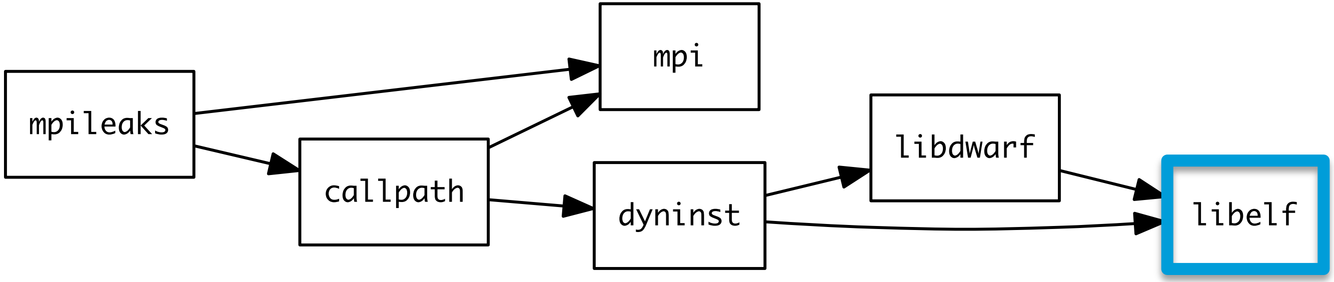
dependency on cuda, but only if cuda is enabled

constraints on cuda version

compiler support for x86_64 and ppc64le

There is a lot of expressive power in the Spack package DSL.

Spack Specs can constrain versions of dependencies

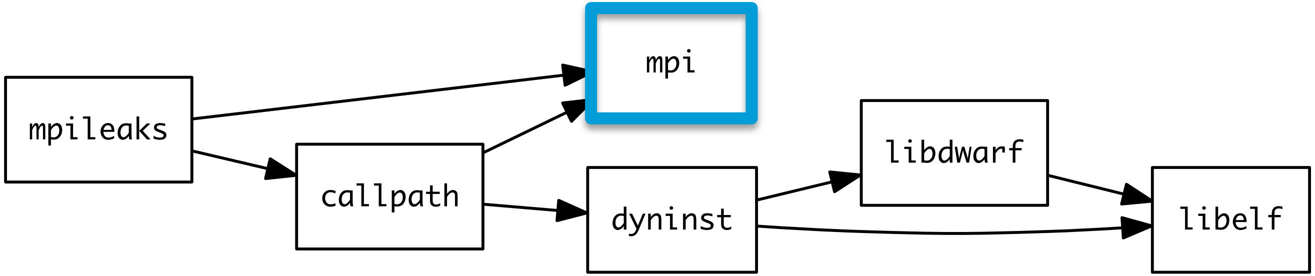


```
$ spack install mpileaks %intel@12.1 ^libelf@0.8.12
```

- 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:
```

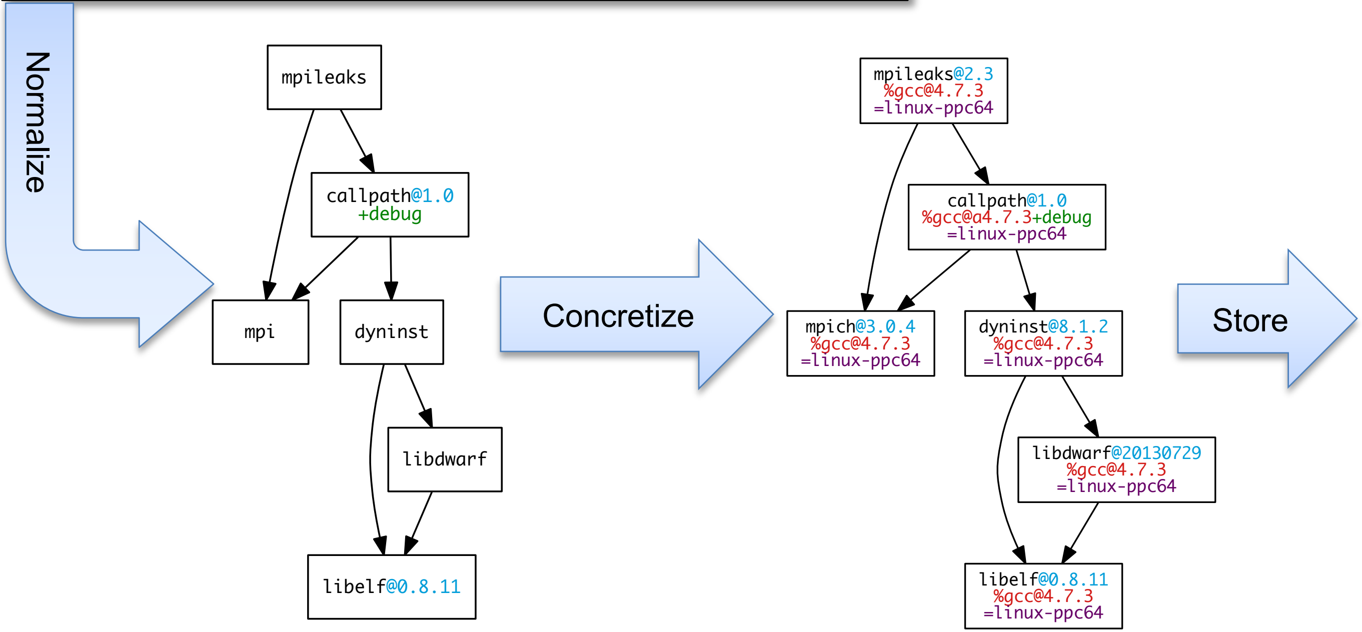
- Let Spack choose MPI implementation, as long as it provides MPI 2 interface:

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

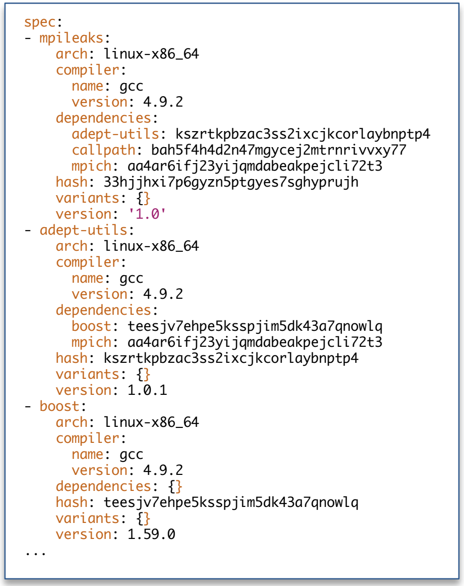
Concretization fills in missing configuration details when the user is not explicit.

```
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
...
```

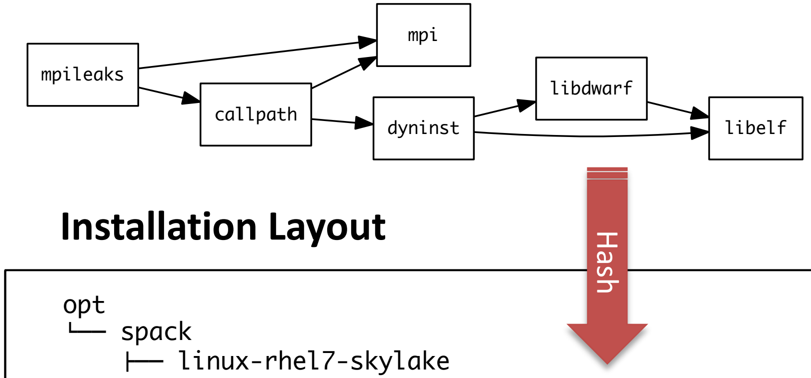
Abstract, normalized spec with some dependencies.

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

Detailed provenance is stored with the installed package

Hashing allows us to handle combinatorial complexity

Dependency DAG



Installation Layout

```
opt
└── spack
    ├── linux-rhel7-skylake
    |   └── gcc-8.3.0
    |       ├── mpileaks-1.0-hc4sm4vuzpm4znmvrfzri4ow2mkphe2e
    |       ├── callpath-1.0.4-daqqpssxb6qbfrztsezkmhus3xoflbsy
    |       ├── openmpi-4.1.4-u64v26igxvxyn23hysmklfums6tgjv5r
    |       ├── dyninst-12.1.0-u64v26igxvxyn23hysmklfums6tgjv5r
    |       ├── libdwarf-20180129-u5eawkvaoc7vonabe6nndkcfwuv233cj
    |       └── libelf-0.8.13-x46q4wm46ay4pltriijbgizxjrhbaka6
```

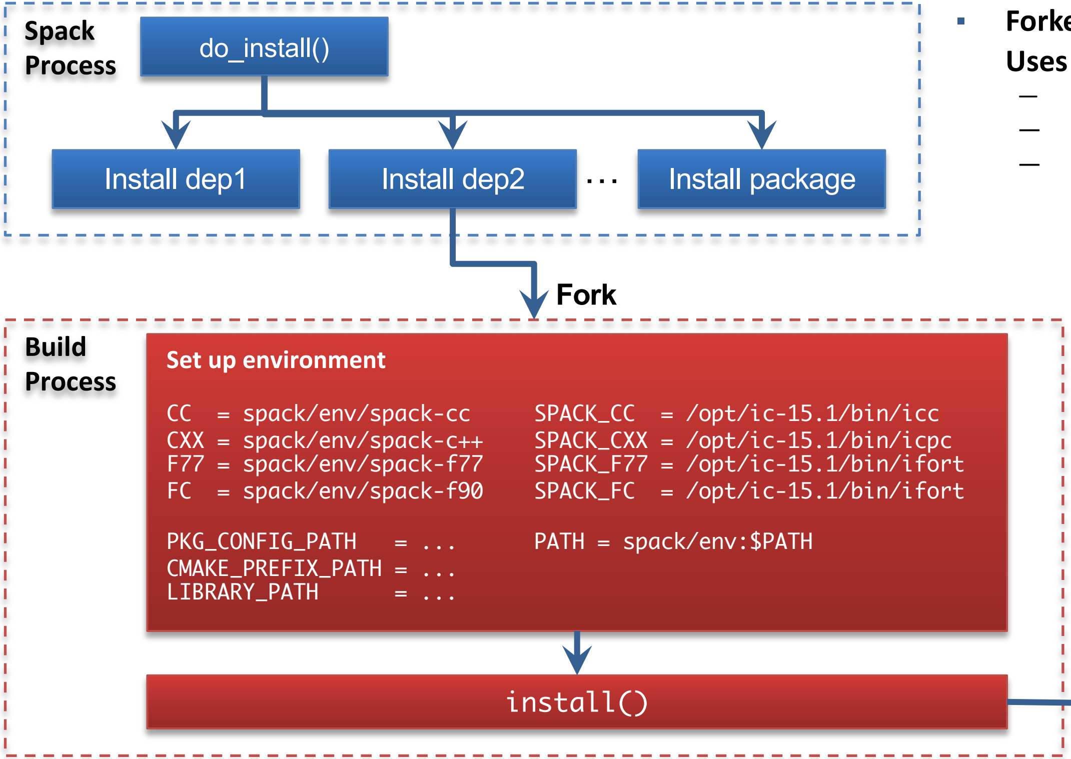
- Each unique dependency graph is a unique ***configuration***.

- 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.

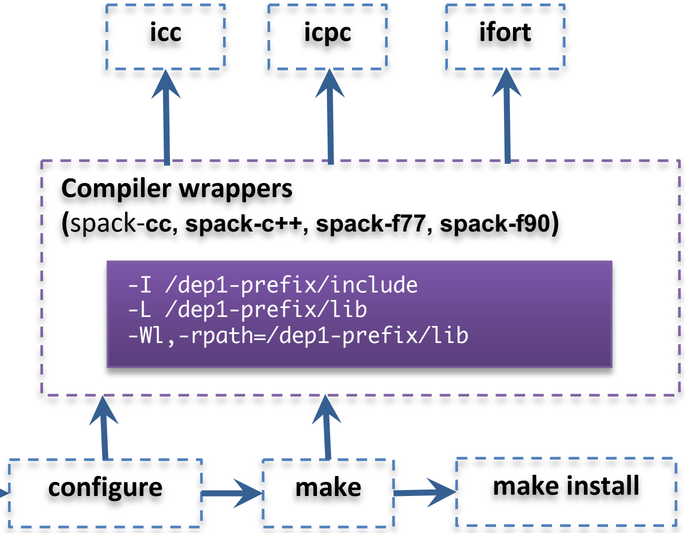
Installed packages automatically find dependencies
- Spack embeds RPATHs in binaries.
- No need to use modules or set LD_LIBRARY_PATH
- Things work *the way you built them*

An isolated compilation environment allows Spack to easily swap compilers



```
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      = ...
```

- **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)



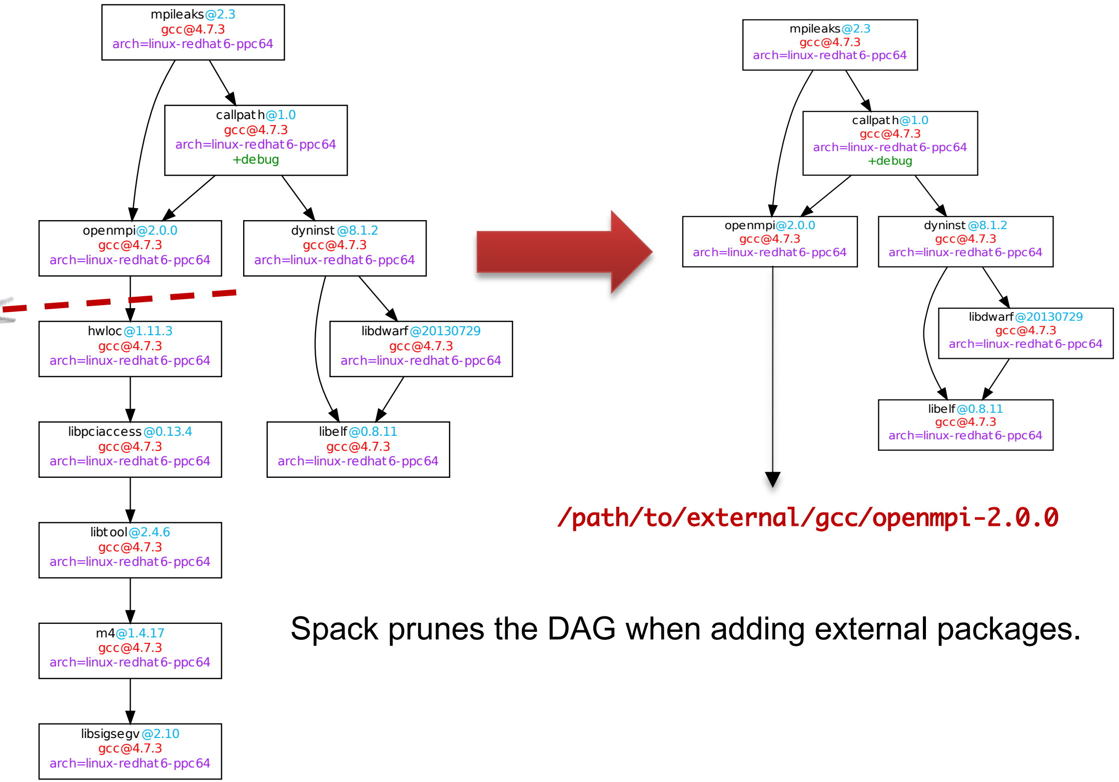
```
-I /dep1-prefix/include
-L /dep1-prefix/lib
-Wl,-rpath=/dep1-prefix/lib
```

We can configure Spack to build with external 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
      ...
```

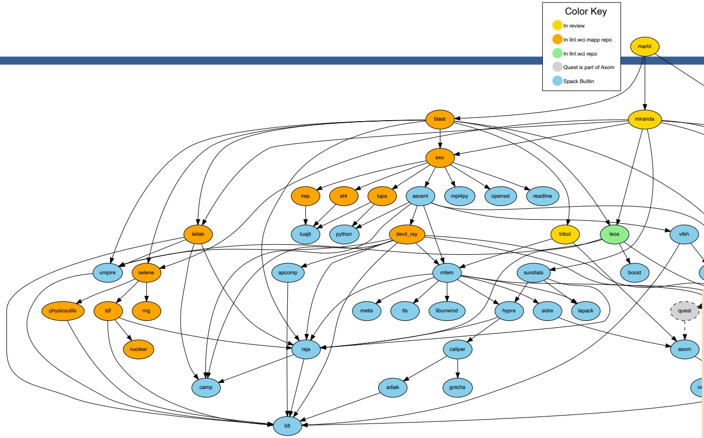


Users register external packages in a configuration file (more on these later).

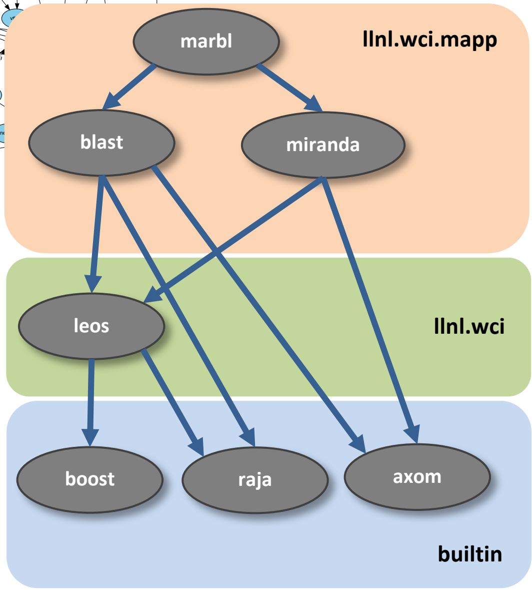
Spack prunes the DAG when adding external packages.

Spack package repositories allow stacks to be layered

LLNL MARBL multi-physics application



MARBL Application Packages



Common internal packages

Open Source Spack packages

```
$ 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
```

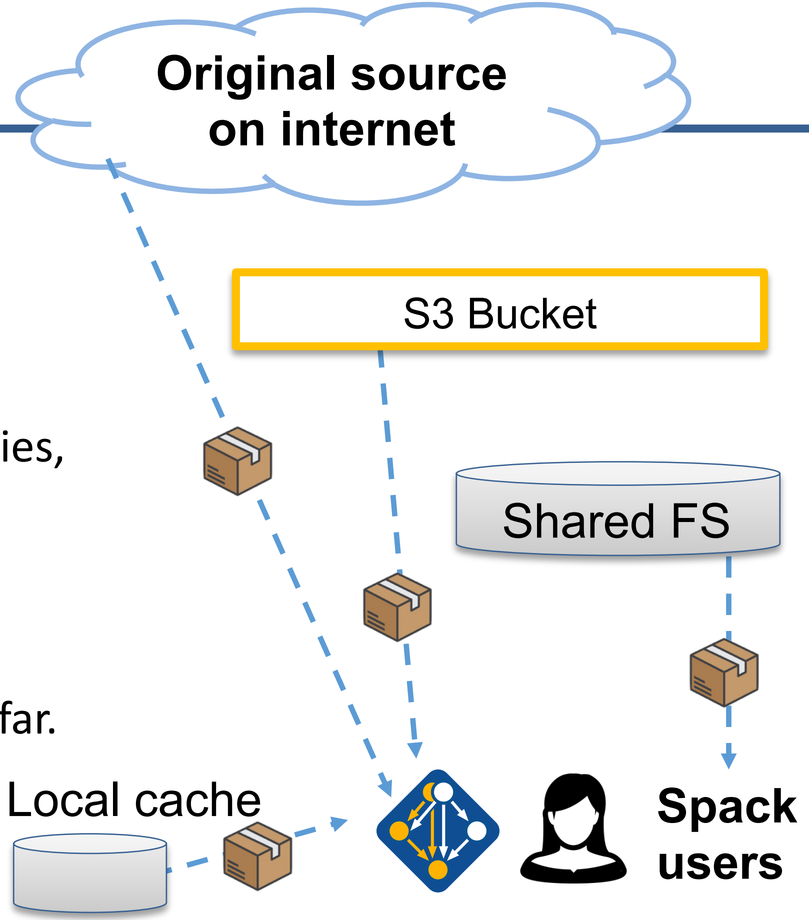
Spack mirrors

- 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.

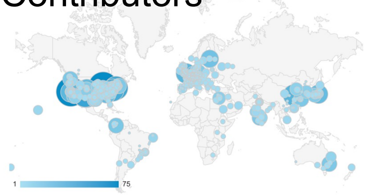
- You can host mirrors internal to your site
  - See the documentation for more details



The concretizer includes information from packages, configuration, and CLI

Dependency solving is NP-hard

Contributors



- new versions
- new dependencies
- new constraints

package.py repository

spack developers

yaml

default config
`packages.yaml`

admins, users

yaml

local preferences config
`packages.yaml`

users

yaml

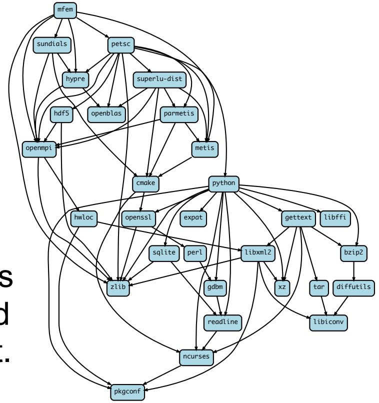
local environment config
`spack.yaml`

users

Command line constraints

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

concretizer



Concrete spec is fully constrained and can be built.

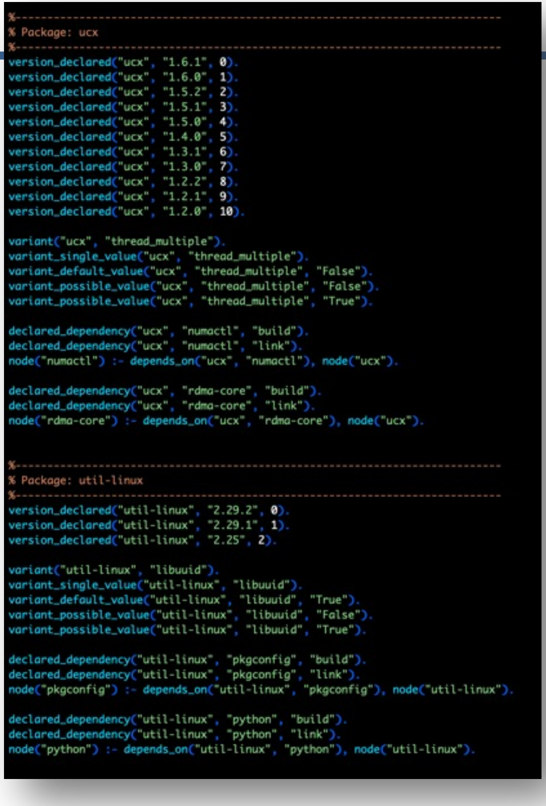
We use logic programming to simplify package solving

- 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
    2. Small logic program (~800 lines)
       - includes 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

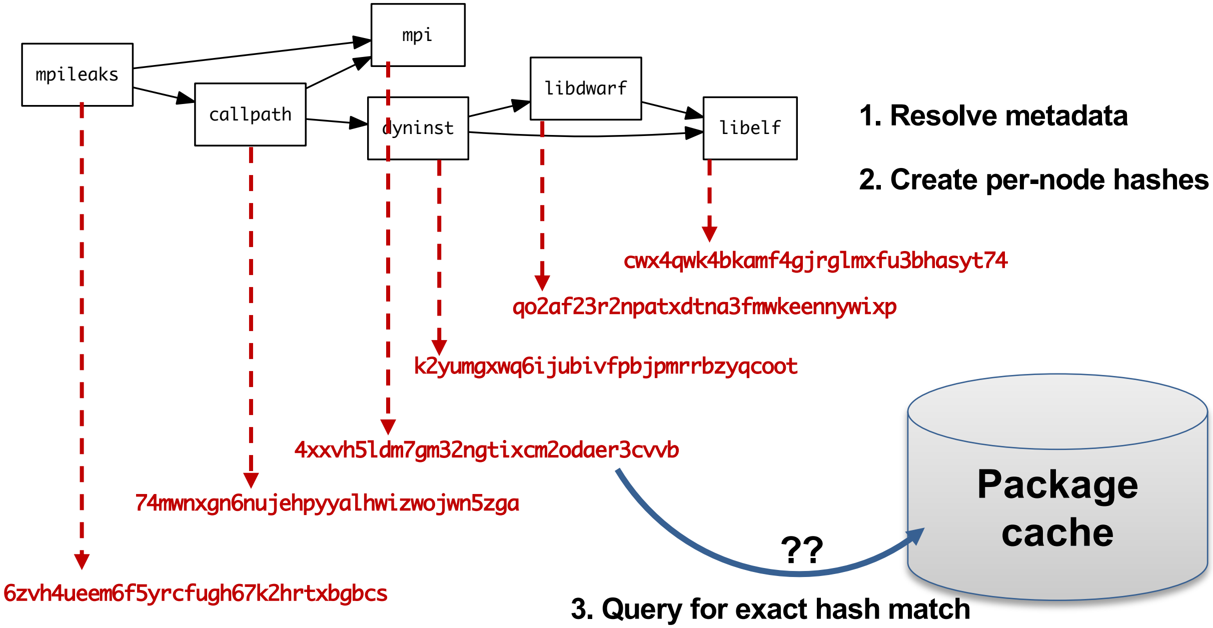
- New concretizer solves many specs that old concretizer can’t
  - Backtracking is a huge win – many issues resolved
  - Conditional logic that was complicated before is now much easier



```
%------------------------------------------------------------------
% 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

`--fresh` only reuses builds if hashes match


mpi
mpileaks
callpath
dyninst
libdwarf
libelf
cwx4qwk4bkamf4gjrglmxfu3bhasyt74
qo2af23r2npatxdtna3fmwkeennywixp
k2yumgxwq6ijubivfpbjpmrrbzyqcoot
4xxvh5ldm7gm32ngtixcm2odaer3cvvb
74mwnxgn6nujehpyyalhwizwojwn5zga
6zvh4ueem6f5yrcfugh67k2hrtxbgbcs
??
Package cache

1. Resolve metadata

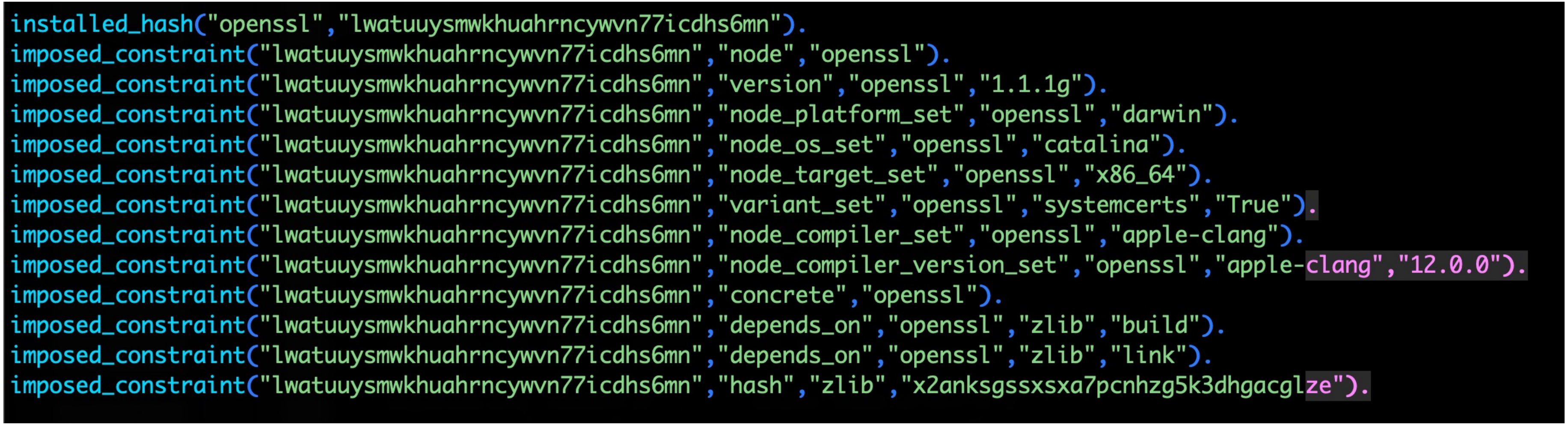
2. Create per-node hashes

3. Query for exact hash match

- Hash matches are very sensitive to small changes
- In many cases, a satisfying cached or already installed spec can be missed
- Nix, Spack, Guix, Conan, and others reuse this way

`--reuse` (now the default) is more aggressive

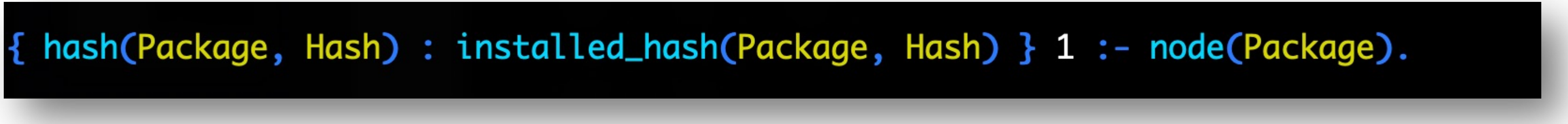
- --reuse tells the solver about all the installed packages!
- Add constraints for all installed packages, with their hash as the associated ID:



```
installed_hash("openssl","lwatuuysmwkhuahrncywvn77icdhs6mn").
imposed_constraint("lwatuuysmwkhuahrncywvn77icdhs6mn","node","openssl").
imposed_constraint("lwatuuysmwkhuahrncywvn77icdhs6mn","version","openssl","1.1.1g").
imposed_constraint("lwatuuysmwkhuahrncywvn77icdhs6mn","node_platform_set","openssl","darwin").
imposed_constraint("lwatuuysmwkhuahrncywvn77icdhs6mn","node_os_set","openssl","catalina").
imposed_constraint("lwatuuysmwkhuahrncywvn77icdhs6mn","node_target_set","openssl","x86_64").
imposed_constraint("lwatuuysmwkhuahrncywvn77icdhs6mn","variant_set","openssl","systemcerts","True").
imposed_constraint("lwatuuysmwkhuahrncywvn77icdhs6mn","node_compiler_set","openssl","apple-clang").
imposed_constraint("lwatuuysmwkhuahrncywvn77icdhs6mn","node_compiler_version_set","openssl","apple-clang","12.0.0").
imposed_constraint("lwatuuysmwkhuahrncywvn77icdhs6mn","concrete","openssl").
imposed_constraint("lwatuuysmwkhuahrncywvn77icdhs6mn","depends_on","openssl","zlib","build").
imposed_constraint("lwatuuysmwkhuahrncywvn77icdhs6mn","depends_on","openssl","zlib","link").
imposed_constraint("lwatuuysmwkhuahrncywvn77icdhs6mn","hash","zlib","x2anksgssxsxa7pcnhzg5k3dhgaclze").
```

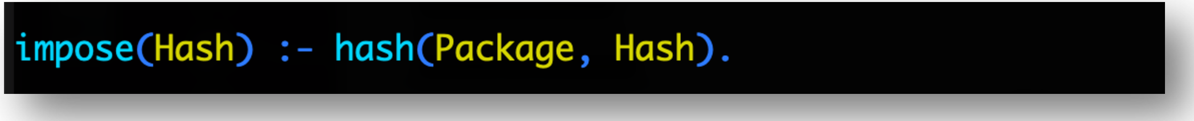
Telling the solver to minimize builds is surprisingly simple in ASP

1. Allow the solver to *choose* a hash for any package:



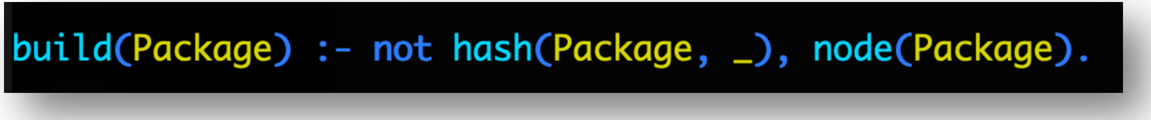
```
{ hash(Package, Hash) : installed_hash(Package, Hash) } 1 :- node(Package).
```

2. Choosing a hash means we impose its constraints:



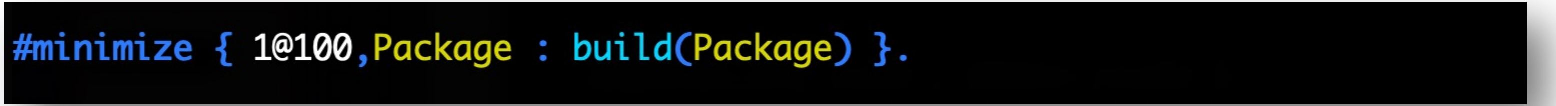
```
impose(Hash) :- hash(Package, Hash).
```

3. Define a build as something *without* a hash:



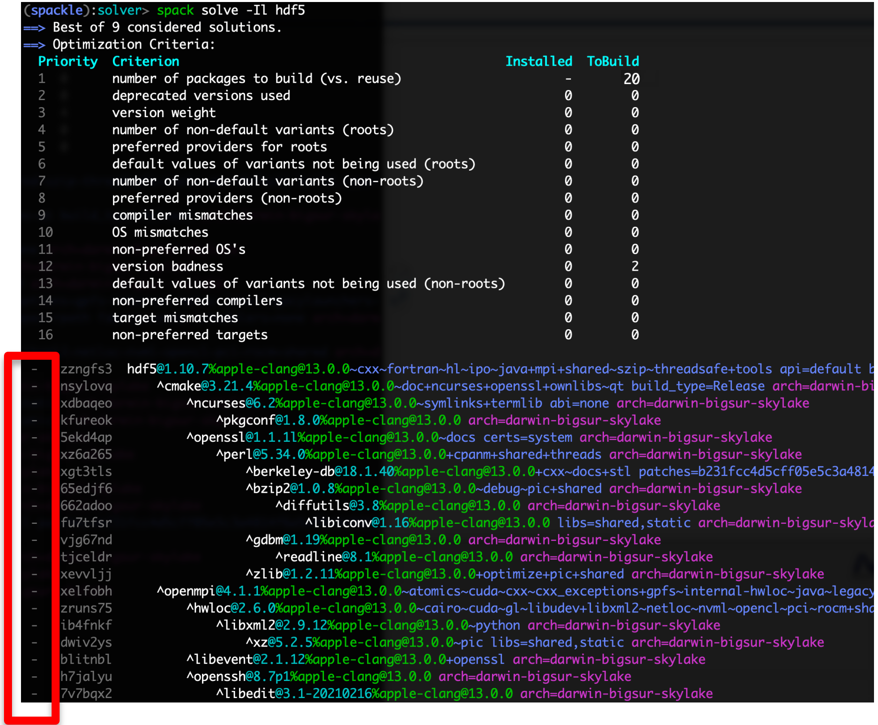
```
build(Package) :- not hash(Package, _), node(Package).
```

4. Minimize builds!



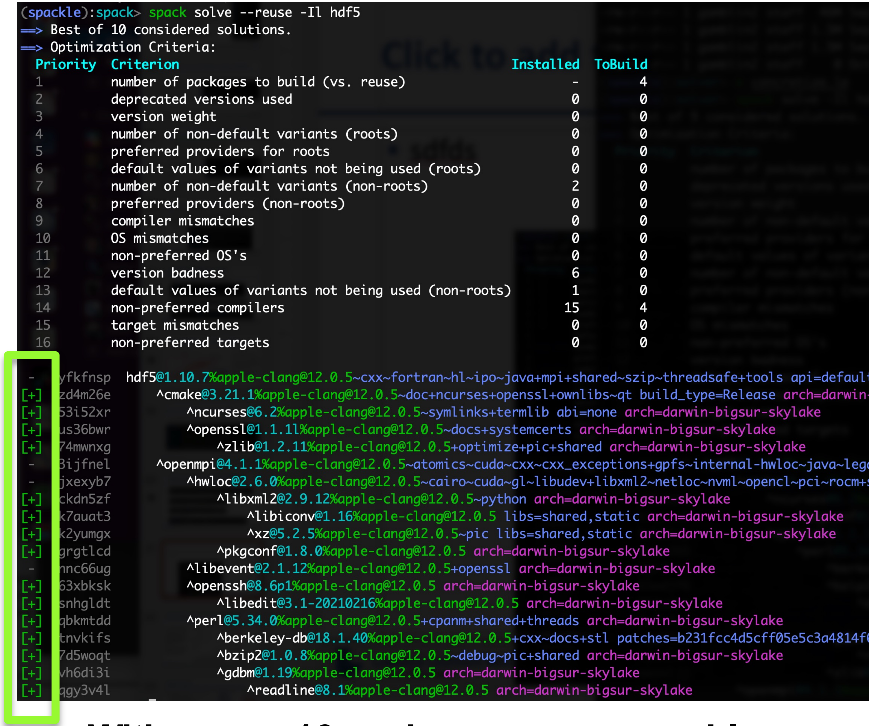
```
#minimize { 1@100,Package : build(Package) }.
```

With and without --reuse optimization



```
(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



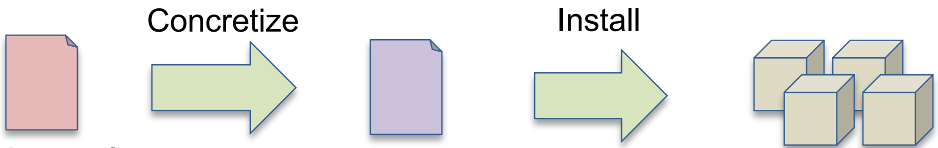
```
(spackle):spack> spack solve --reuse -Il hdf5
==> Best of 10 considered solutions.
==> Optimization Criteria:
  Priority  Criterion                                            Installed  ToBuild
  1         number of packages to build (vs. reuse)                      -        4
  2         deprecated 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
[+]  k2yumgx                  ^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 reusable

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 environments enable users to build customized stacks from an abstract description



Concretize

Install

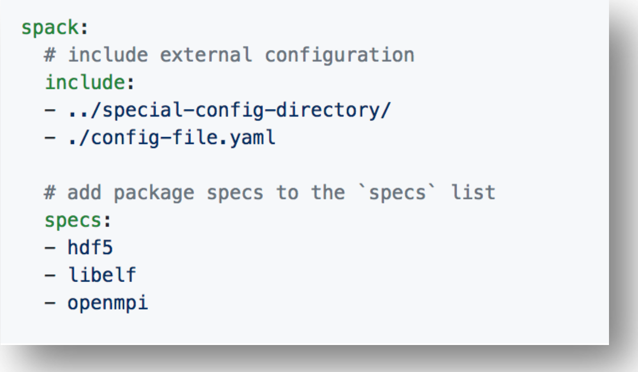
spack.yaml file describes requirements

spack.lock describes exact versions installed

Pacakge installations

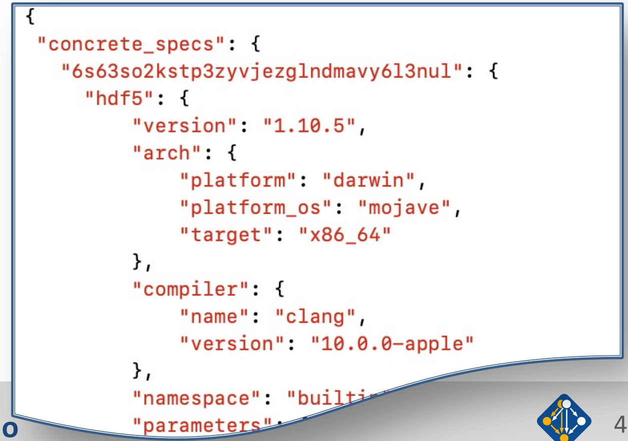
- spack.yaml describes project requirements
- spack.lock describes exactly what versions/configurations were installed, allows them to be reproduced.
- Can be used to maintain configuration of a software stack.
  - Can easily version an environment in a repository

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": "builti
         "parameters":
```

We'll resume at:
11:30am

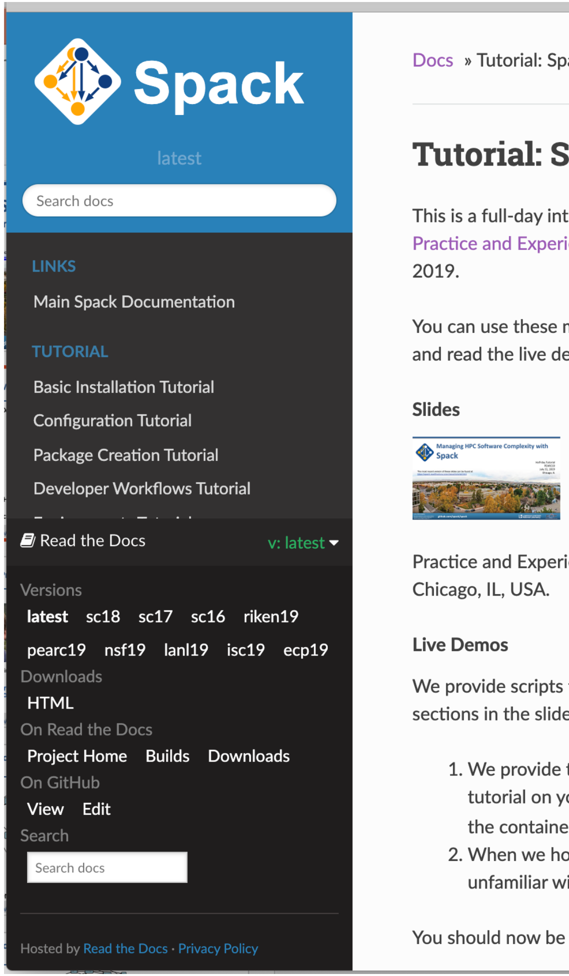
Find the slides and associated scripts here:

spack-tutorial.readthedocs.io

Remember to join Spack slack so you can get help!

slack.spack.io

Join the **#tutorial** channel!



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: Spa

Tutorial: S

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

You can use these n
and read the live de

Slides

Practice and Experi
Chicago, IL, USA.

Live Demos

We provide scripts
sections in the slide

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

You should now be

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

Follow script at spack-tutorial.readthedocs.io

Hands-on Time: Configuration

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

We’ll resume at:
2:00pm

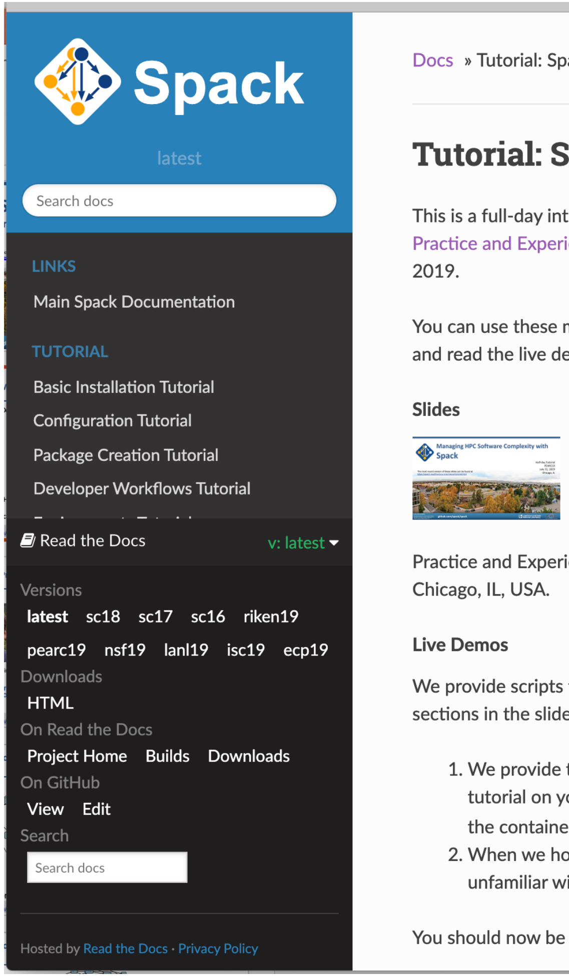
Find the slides and associated scripts here:

spack-tutorial.readthedocs.io

Remember to join Spack slack so you can get help!

slack.spack.io

Join the **#tutorial** channel!



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: Spa

Tutorial: S

This is a full-day int

Practice and Experi

2019.

You can use these r

and read the live de

Slides

Practice and Experi

Chicago, IL, USA.

Live Demos

We provide scripts

sections in the slide

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

You should now be

Hands-on Time:
Creating Packages

Follow script at spack-tutorial.readthedocs.io

Hands-on Time:
Mirrors and Build Caches

Follow script at spack-tutorial.readthedocs.io

We’ll resume at:
4:00pm

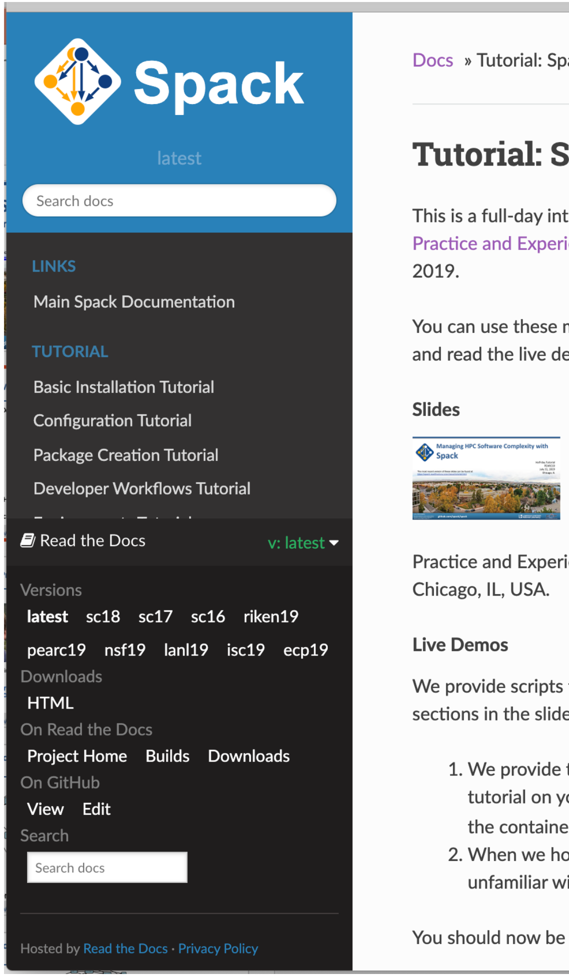
Find the slides and associated scripts here:

spack-tutorial.readthedocs.io

Remember to join Spack slack so you can get help!

slack.spack.io

Join the **#tutorial** channel!



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: Spa

Tutorial: S

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

You can use these r
and read the live de

Slides

Practice and Experi
Chicago, IL, USA.

Live Demos

We provide scripts
sections in the slide

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

You should now be

Hands-on Time: Stacks

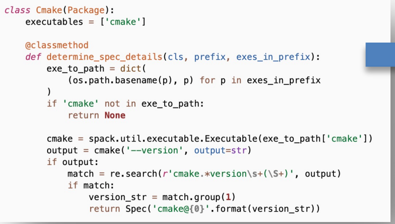
Follow script at spack-tutorial.readthedocs.io

Hands-on Time:
Developer Workflows

Follow script at spack-tutorial.readthedocs.io

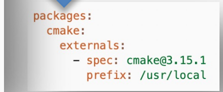
More Features and the Road Ahead

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



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

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

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

package.py



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

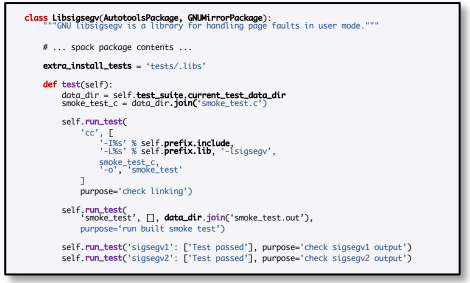
`spack.yaml` configuration

spack external find

Automatically find and configure external packages on the system

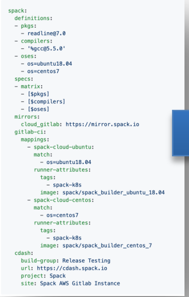
spack test

Packages know how to run their own test suites



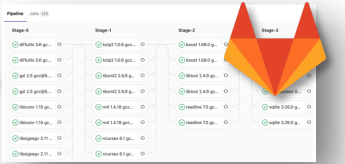
```python
class Libsigsegv(AutotoolsPackage, GNUMirrorPackage):
    """GNU libsigsegv is a library for handling page faults in user mode."""

    # ... spack package contents ...

    extra_install_tests = 'tests/.libs'

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

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

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

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

package.py



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

spack.yaml



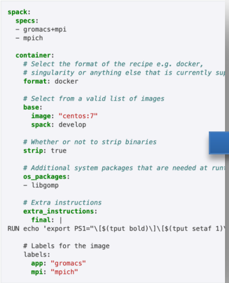
`.gitlab-ci.yml` CI pipeline

spack ci

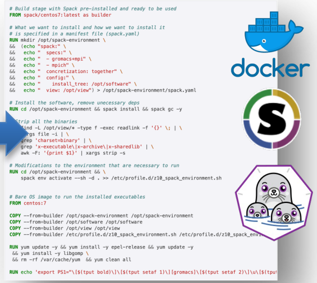
Automatically generate parallel build pipelines (more on this later)

spack containerize

Turn environments into container build recipes



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

  container:
    # Select the format of the recipe e.g. docker,
    # singularity or anything else that is currently sup
    format: docker

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

    # Whether or not to strip binaries
    strip: true

    # Additional system packages that are needed at run
    os_packages:
    - libgomp

    # Extra instructions
    extra_instructions:
      final: |
RUN echo 'export PS1="\[$(tput bold)\]\[$(tput setaf 1)\

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



```dockerfile
# Build stage with Spack pre-installed and ready to be used
FROM spack/centos7:latest as builder

# What we want to install and how we want to install it
# is specified in a manifest file (spack.yaml)
RUN mkdir /opt/spack-environment \
&&  (echo "spack:" \
&&   echo "  specs:" \
&&   echo "  - gromacs+mpi" \
&&   echo "  - mpich" \
&&   echo "  concretization: together" \
&&   echo "  config:" \
&&   echo "    install_tree: /opt/software" \
&&   echo "  view: /opt/view") > /opt/spack-environment/spack.yaml

# Install the software, remove unecessary deps
RUN cd /opt/spack-environment && spack install && spack gc -y

# Strip all the binaries
RUN find -L /opt/view/* -type f -exec readlink -f '{}' \; | \
    xargs file -i | \
    grep 'charset=binary' | \
    grep 'x-executable\|x-archive\|x-sharedlib' | \
    awk -F: '{print $1}' | xargs strip -s

# Modifications to the environment that are necessary to run
RUN cd /opt/spack-environment && \
    spack env activate --sh -d . >> /etc/profile.d/z10_spack_environment.sh


# Bare OS image to run the installed executables
FROM centos:7

COPY --from=builder /opt/spack-environment /opt/spack-environment
COPY --from=builder /opt/software /opt/software
COPY --from=builder /opt/view /opt/view
COPY --from=builder /etc/profile.d/z10_spack_environment.sh /etc/profile.d/z10_spack_envi

RUN yum update -y && yum install -y epel-release && yum update -y
 && yum install -y libgomp \
 && rm -rf /var/cache/yum  && yum clean all

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

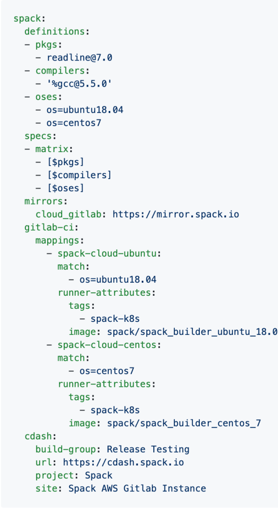
Spack environments are the foundation of Spack CI

- `spack ci` enables any environment to be turned into a build pipeline

- Pipeline generates a `.gitlab-ci.yml` file from spack.lock

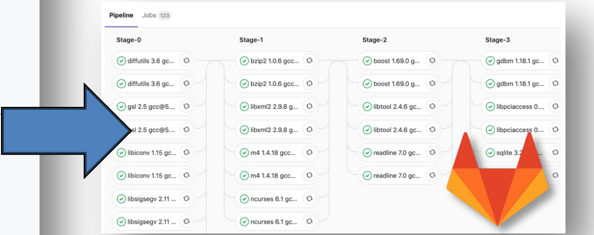
- Pipelines can be used just to build, or to generate relocatable binary packages
  - Binary packages can be used to keep the same build from running twice

- 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
```

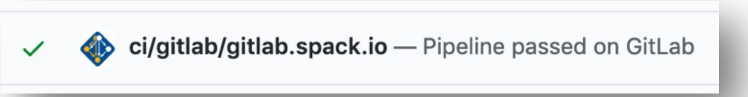
spack.yaml



Parallel GitLab build pipeline

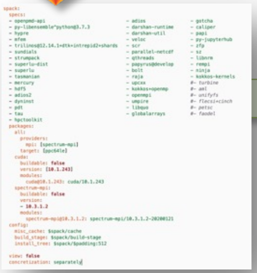
We are building a supply chain for HPC

Spack Contributions on GitHub



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

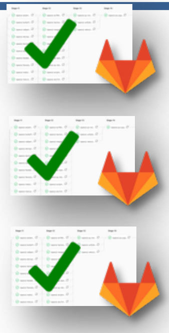
gitlab.spack.io



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

  view: false
  concretization: separately
```

`spack.yaml` configurations (E4S, SDKs, others)

`spack ci`



x86_64 and **aarch64** pipelines in **AWS**

ppc64le, GPU pipelines at **U. Oregon**

Pipelines at **LLNL** **(Cray PE soon, hopefully)**

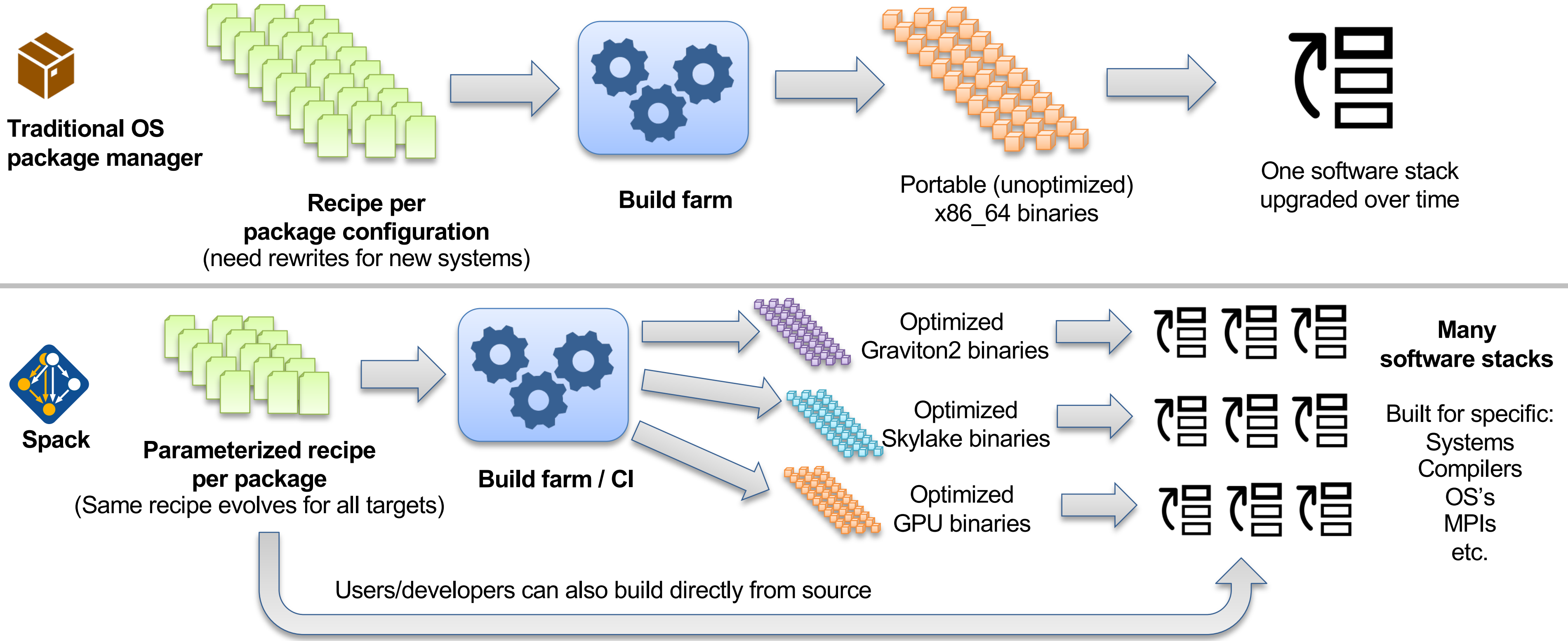
GitLab CI builds (changed) packages

- On every pull request
- On every release branch

- **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



Traditional OS package manager

Recipe per package configuration
(need rewrites for new systems)

Build farm

Portable (unoptimized)
x86_64 binaries

One software stack
upgraded over time

Spack

Parameterized recipe per package
(Same recipe evolves for all targets)

Build farm / CI

Optimized
Graviton2 binaries

Optimized
Skylake binaries

Optimized
GPU binaries

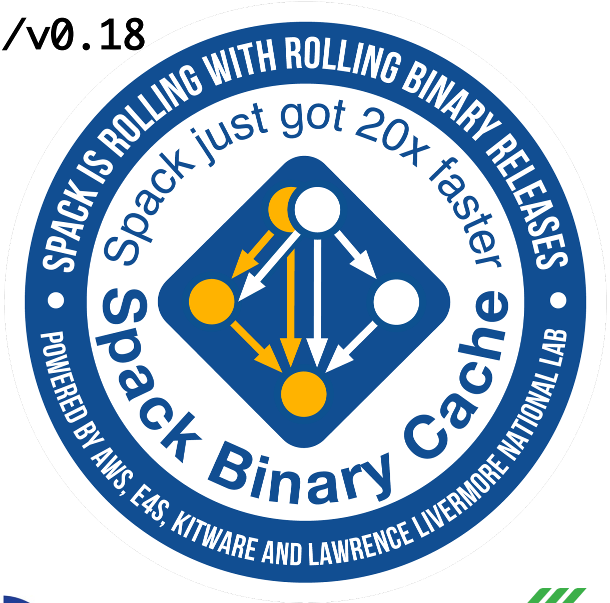
Many software stacks

Built for specific:
Systems
Compilers
OS's
MPIs
etc.

Users/developers can also build directly from source

We started providing public binaries in June 2022

```
# latest v0.18.x release binaries
spack mirror add https://binaries.spack.io/releases/v0.18


# rolling release: bleeding edge binaries
spack mirror add https://binaries.spack.io/develop
```

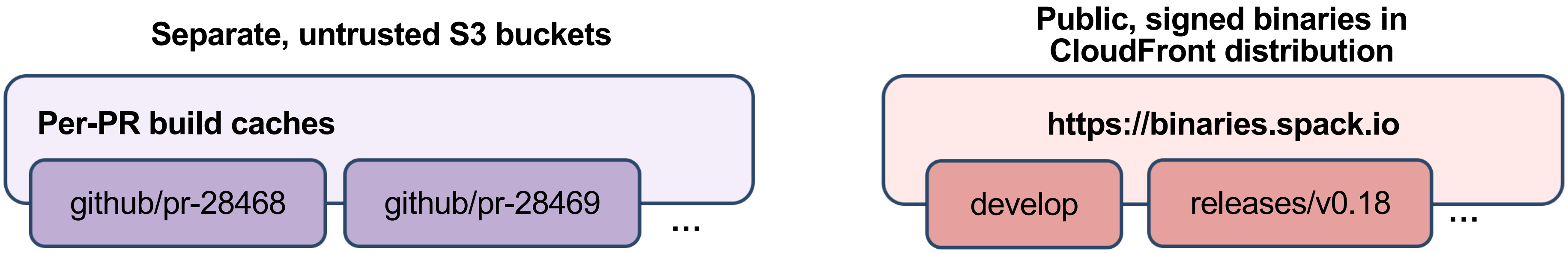

SPACK IS ROLLING WITH ROLLING BINARY RELEASES
Spack just got 20x faster
Spack Binary Cache
POWERED BY AWS, E4S, KITWARE AND LAWRENCE LIVERMORE NATIONAL LAB

- Over 3,000 builds in the cache so far:

Amazon Linux 2	x86_64_v4
Amazon Linux 2	aarch64
Amazon Linux 2	graviton2
Ubuntu 18.04	x86_64

- Expect this list to expand!

Our infrastructure enables us to sustainably manage a binary distro



Separate, untrusted S3 buckets

Per-PR build caches

github/pr-28468

github/pr-28469

…

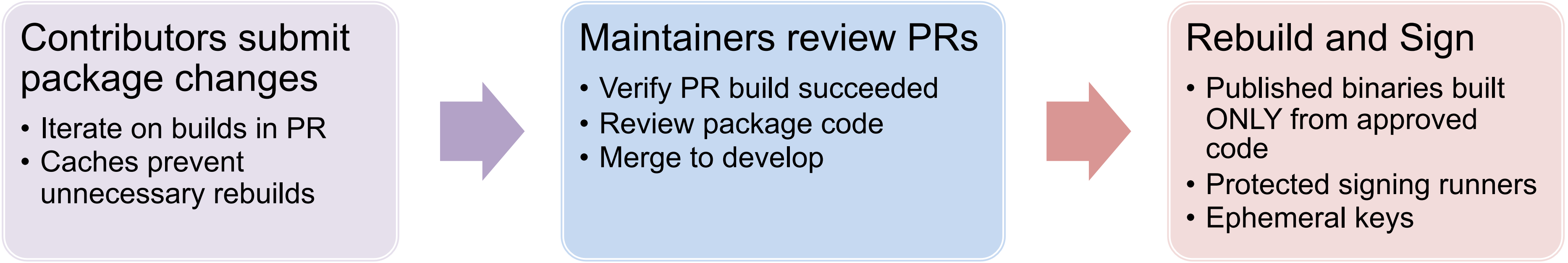
Public, signed binaries in CloudFront distribution

https://binaries.spack.io

develop

releases/v0.18

…



Contributors submit package changes

- Iterate on builds in PR
- Caches prevent unnecessary rebuilds

Maintainers review PRs

- Verify PR build succeeded
- Review package code
- Merge to develop

Rebuild and Sign

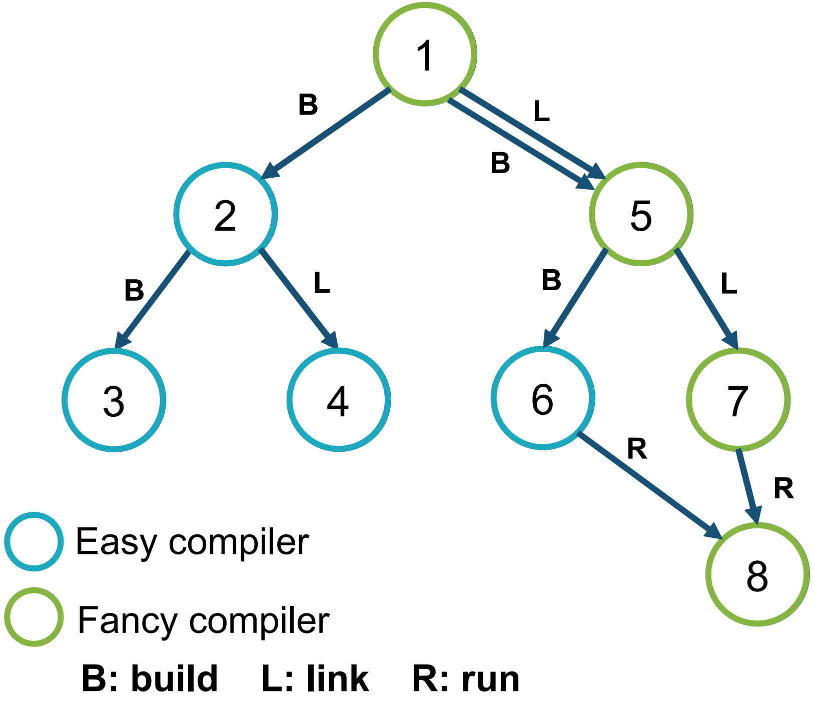
- Published binaries built ONLY from approved code
- Protected signing runners
- Ephemeral keys

- Moves bulk of binary maintenance upstream, onto PRs
  - Production binaries never reuse binaries from untrusted environment

Spack v0.20 roadmap:
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
- 2 approaches to modify concretization:
  1. **Separate solves**
     - Solve run and link dependencies first
     - Solve for build dependencies separately
     - May restrict possible solutions (build ←→ run env constraints)
  2. **Separate models**
     - *Allow* a bigger space of packages in the solve
     - Solve *all* runtime environments together
     - May explode (even more) combinatorially

```
spack install pkg1 %intel
```



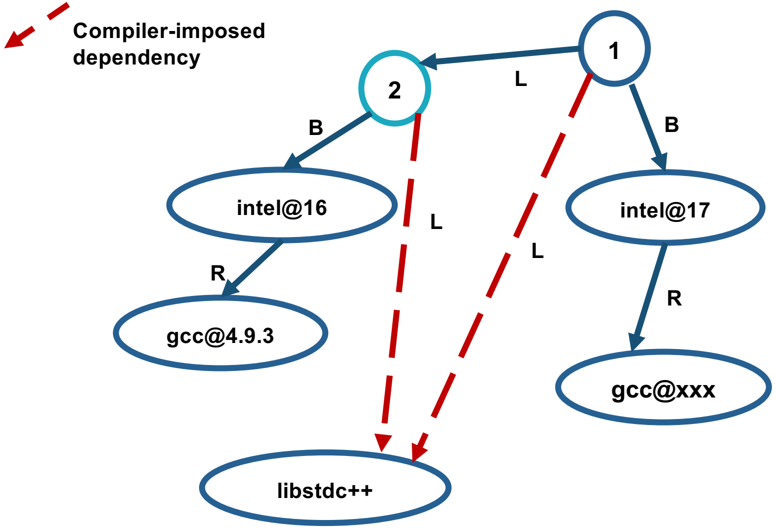
○ Easy compiler

○ Fancy compiler

B: build L: link R: run

Spack 0.20 Roadmap: compilers as dependencies

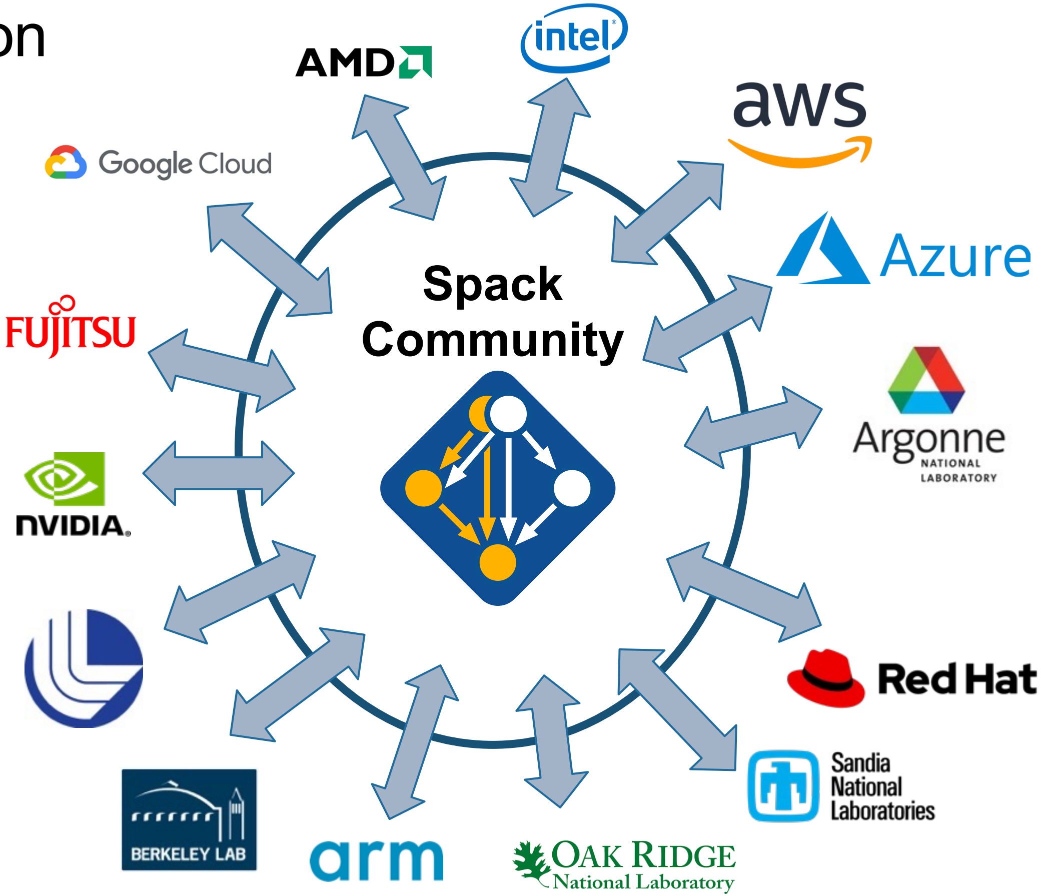
- **We need deeper modeling of compilers to handle compiler interoperability**
  - libstdc++, libc++ compatibility
  - Compilers that depend on compilers
  - Linking executables with multiple compilers
- **First prototype is complete!**
  - We've done successful builds of some packages using compilers 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

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
- **Growing contributor base and continuing to automate are the most important priorities**
  - **377 contributors to 0.18 release!**


Spack
Community
AMD
intel
aws
Azure
Argonne NATIONAL LABORATORY
Red Hat
Sandia National Laboratories
OAK RIDGE National Laboratory
arm
BERKELEY LAB
NVIDIA
FUJITSU
Google Cloud

When would we go 1.0?

- Big things we’ve wanted for 1.0 are:
  - New concretizer
  - production CI
  - production public build cache
  - Compilers as dependencies
  - Stable package API
    - Enables separate package repository
- After 0.19 we will hopefully have all of these
  - Maybe there won’t be a 0.20!

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!