

Managing HPC Software Complexity with Spack

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

RADIUSS Tutorial Series
Aug 1-2, 2022



LLNL-PRES-806064

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

spack.io

 Lawrence Livermore
National Laboratory

Tutorial Materials

Find these slides and associated scripts here:

spack-tutorial.rtf.d.io

We also have a chat room on Spack slack.

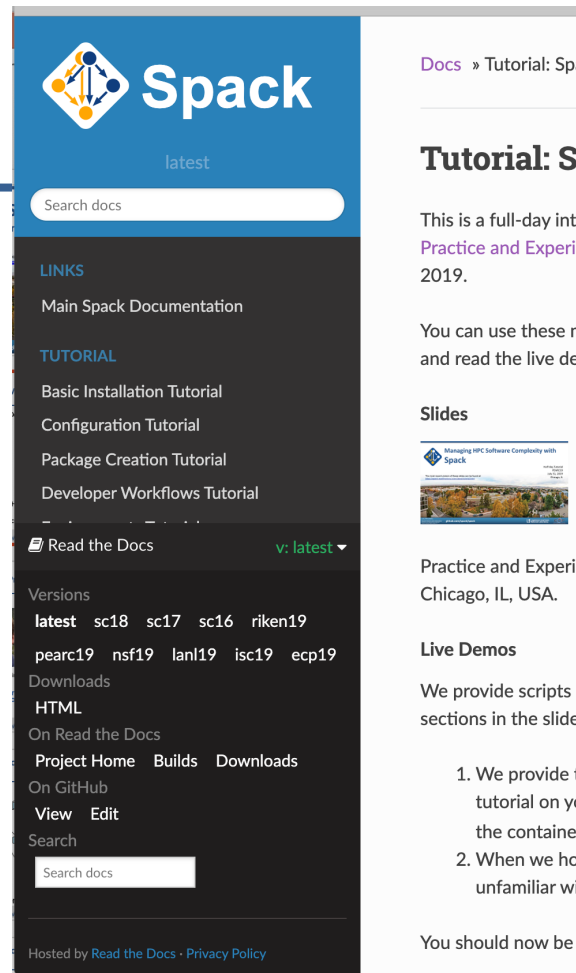
You can join here:

slack.spack.io

Join the [#tutorial](#) channel!

You can ask questions here after the conference is over.

Over **1,800 people** can help you on Slack!



The screenshot shows the Spack documentation website. At the top is the Spack logo and the word "Spack" in a large font. Below that is a search bar with the text "Search docs". The main content area is dark-themed and contains several sections: "LINKS" with links to "Main Spack Documentation", "TUTORIAL" with links to "Basic Installation Tutorial", "Configuration Tutorial", "Package Creation Tutorial", and "Developer Workflows Tutorial", "Read the Docs" with a version selector set to "v: latest", "Versions" listing various versions like "latest", "sc18", "sc17", "sc16", "riken19", "pearc19", "nsf19", "lan19", "isc19", "ecp19", "Downloads", "HTML", "On Read the Docs", "Project Home", "Builds", "Downloads", "On GitHub", "View", "Edit", and another "Search" bar with "Search docs" text. At the bottom of the page, it says "Hosted by Read the Docs · Privacy Policy".

Docs » Tutorial: Spack

Tutorial: Spack

This is a full-day introductory tutorial on Spack. Practice and Experience with Spack 2019.

You can use these notes and read the live demo.

Slides



Practice and Experience with Spack, Chicago, IL, USA.

Live Demos

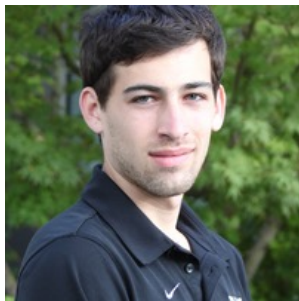
We provide scripts and sections in the slides.

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

You should now be



Tutorial Presenters



Greg Becker
LLNL



Richarda Butler
LLNL



Tamara Dahlgren
LLNL



Todd Gamblin
LLNL

Agenda (approximate, all times PDT)

Monday

Intro	8:00 am
Basics	8:15 am
Concepts	9:00 am
Break	9:30 am
Environments	10:00 am
Configuration	10:45 am
End	11:30 am

Tuesday

Packaging	8:00 am
Developer Workflows	8:45 am
Scripting	9:15 am
Break	9:30 am
Mirrors & Binary Caches	10:00 am
Modules	10:20 am
Roadmap / Questions	11:05 am
End	11:30 am



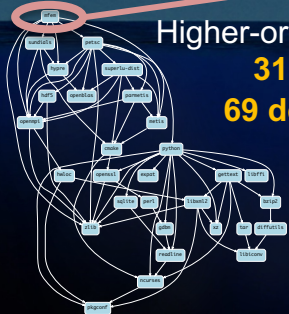
Modern scientific codes rely on icebergs of dependency libraries

71 packages
188 dependencies

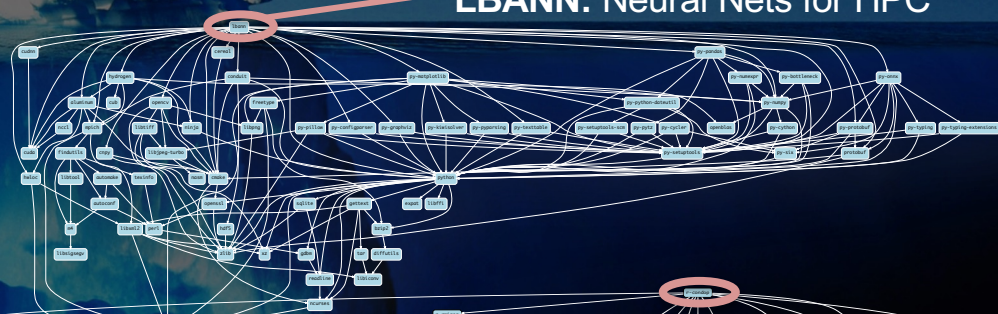
MFEM:

Higher-order finite elements

31 packages,
69 dependencies



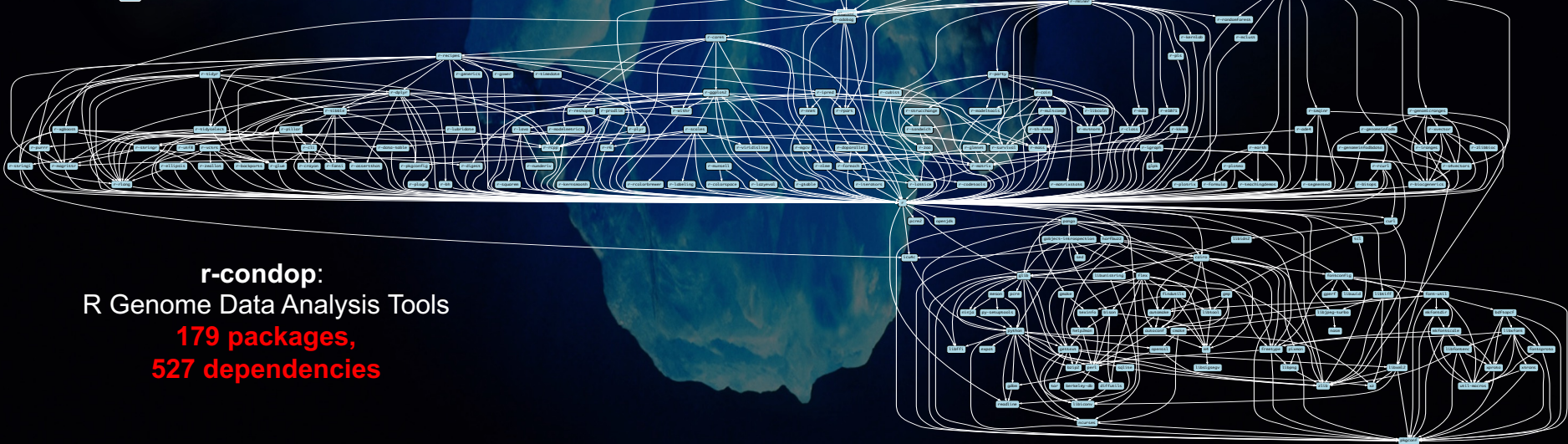
LBANN: Neural Nets for HPC



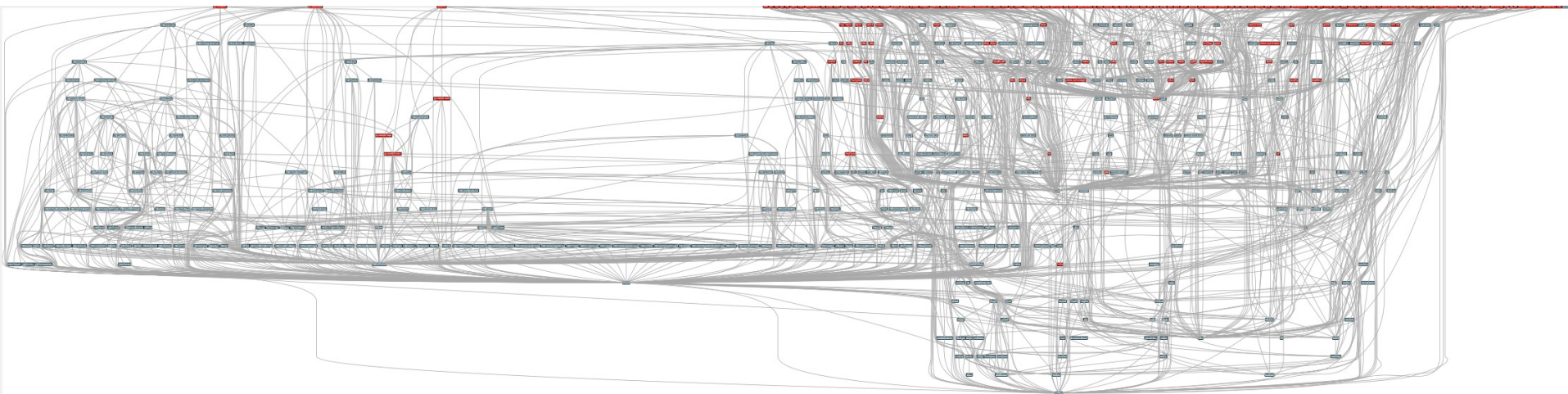
r-condop:

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

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



docker



Charliecloud



SHIFTER

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

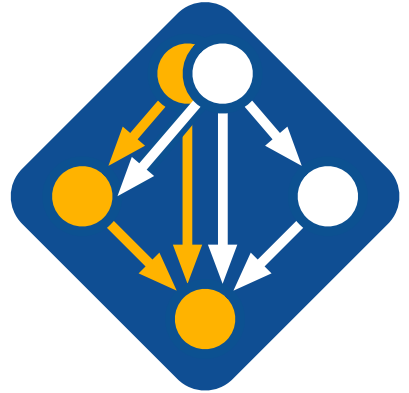
No installation required: clone and go

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

- 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)



github.com/spack/spack



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

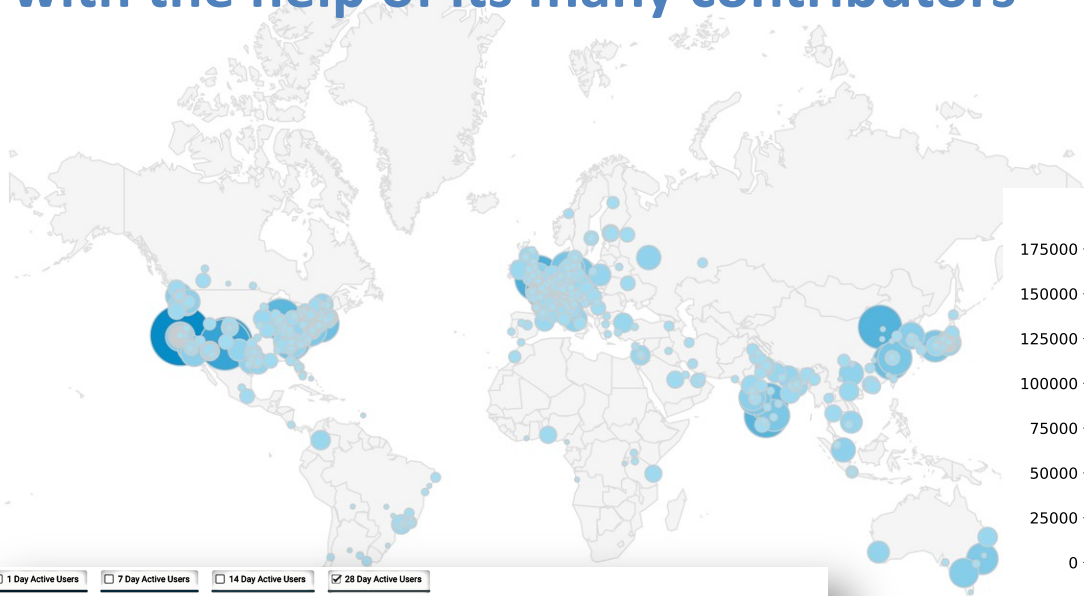
3. Package Developers

- People who want to package their own software for distribution

4. User support teams at HPC Centers

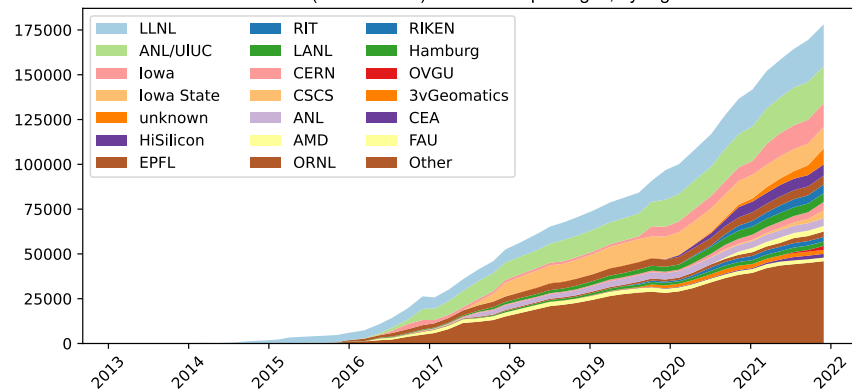
- People who deploy software for users at large HPC sites

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



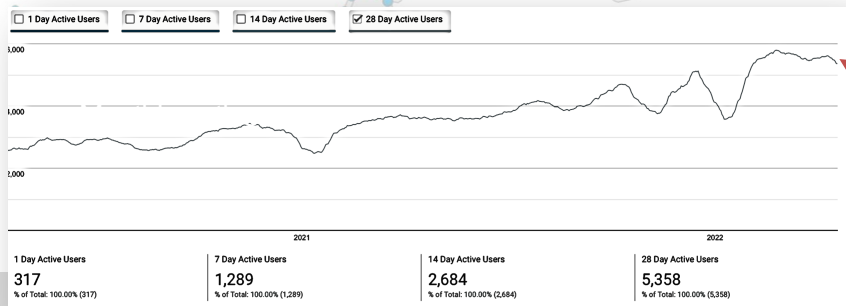
Over 6,400 software packages
Over 1,050 contributors

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)

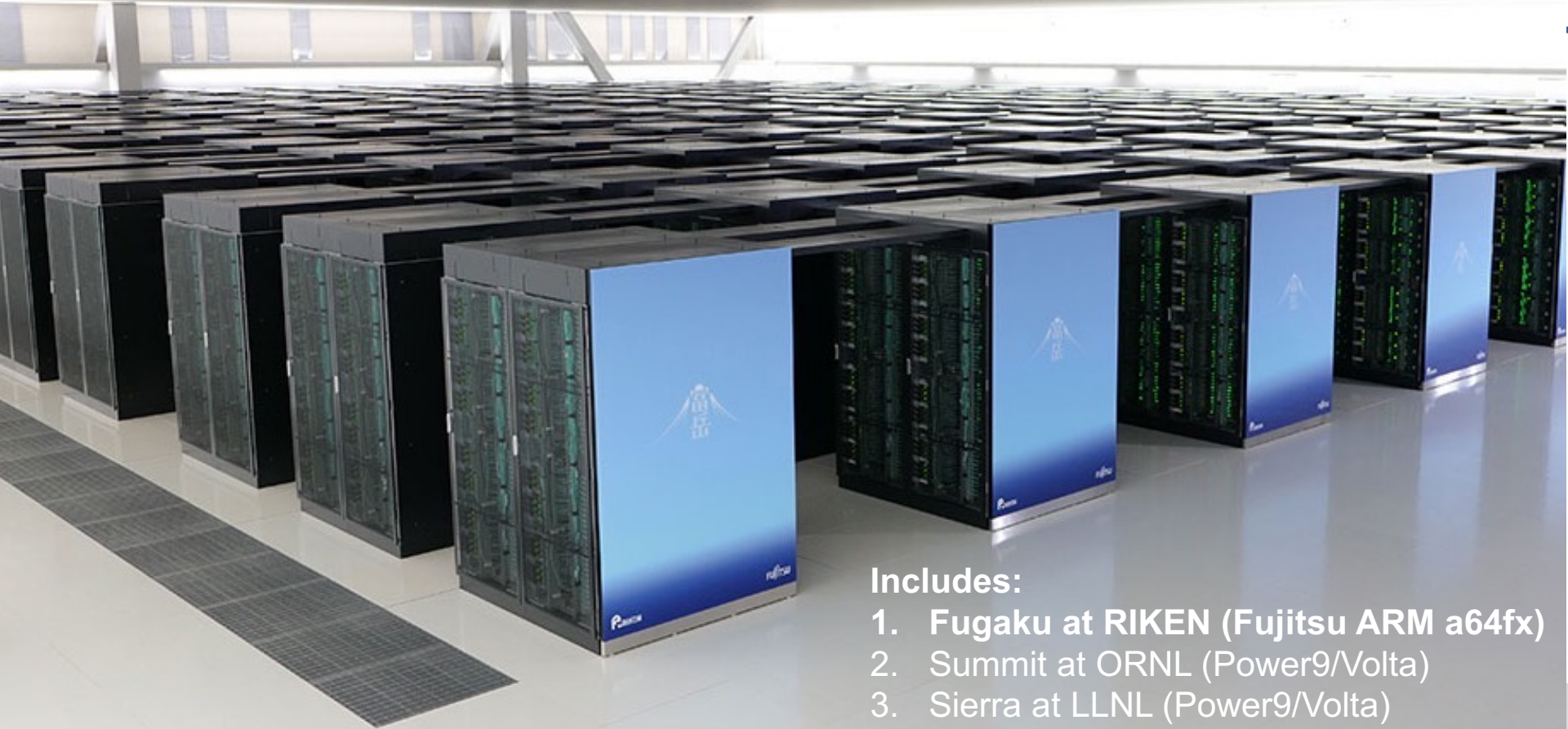


Join #tutorial on Slack: slack.spack.io

Materials: spack-tutorial.readthedocs.io



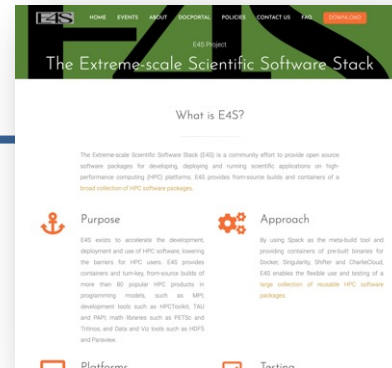
Spack is used on the fastest supercomputers in the world



Includes:

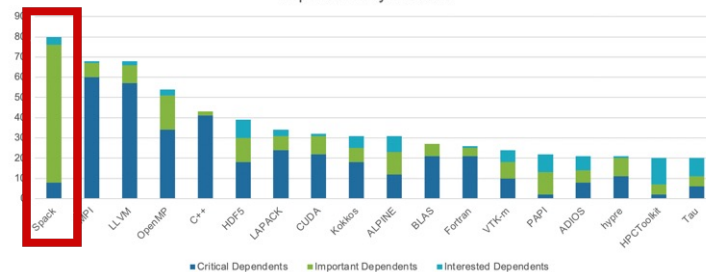
1. Fugaku at RIKEN (Fujitsu ARM a64fx)
2. Summit at ORNL (Power9/Volta)
3. Sierra at LLNL (Power9/Volta)

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



<https://e4s.io>

Dependents by Producer



Spack is the most depended-upon project in 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.



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

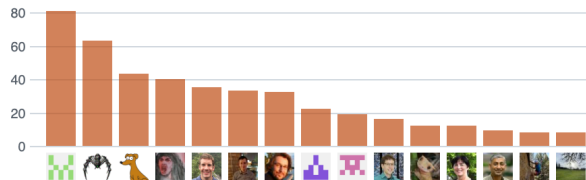
🏠 75

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

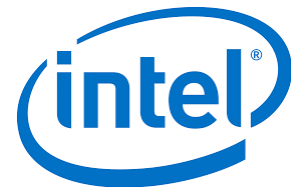
📦 v0.17.0

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 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

<code>\$ spack install mpileaks</code>	<code>unconstrained</code>
<code>\$ spack install mpileaks@3.3</code>	<code>@ custom version</code>
<code>\$ spack install mpileaks@3.3 %gcc@4.7.3</code>	<code>% custom compiler</code>
<code>\$ spack install mpileaks@3.3 %gcc@4.7.3 +threads</code>	<code>+/- build option</code>
<code>\$ spack install mpileaks@3.3 cppflags="-O3 -g3"</code>	<code>set compiler flags</code>
<code>\$ spack install mpileaks@3.3 target=cascadelake</code>	<code>set target microarchitecture</code>
<code>\$ spack install mpileaks@3.3 ^mpich@3.2 %gcc@4.9.3</code>	<code>^ dependency constraints</code>

- 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-openssl-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

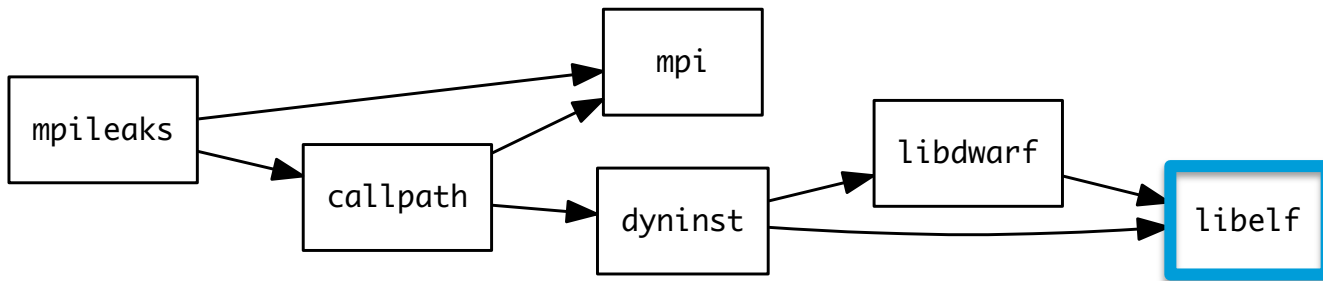
constraints on cuda version

compiler support for x86_64
and ppc64le

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



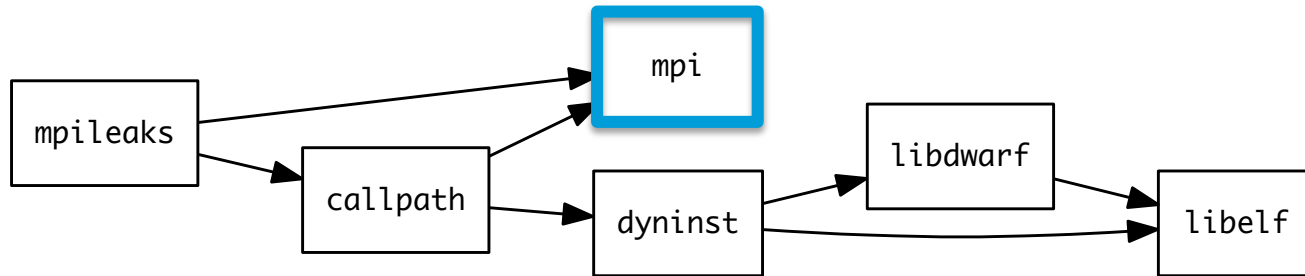
Spack Specs can constrain versions of dependencies



```
$ 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
```

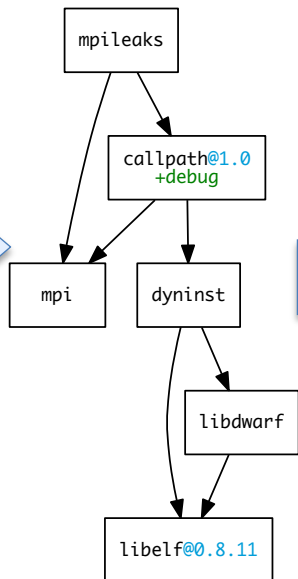
Concretization fills in missing configuration details when the user is not explicit.

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

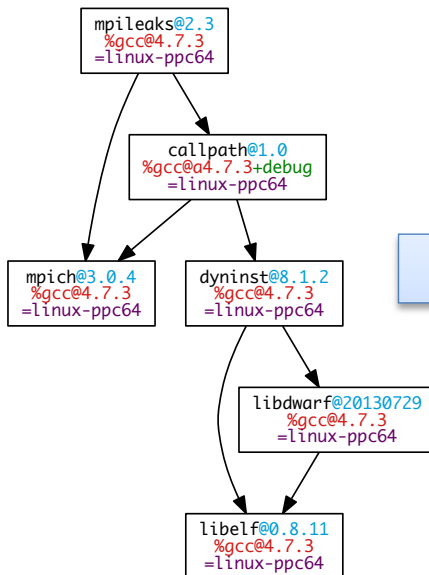
User input: *abstract* spec with some constraints

spec.yaml

Normalize



Concretize



Store

```
spec:
- mpileaks:
  arch: linux-x86_64
  compiler:
    name: gcc
    version: 4.9.2
  dependencies:
    adept-utils: kszrtkpbzac3ss2ixcjkcorlaybnptp4
    callpath: bah5f4h4d2n47mgycej2mitrnrivvy77
    mpich: aa4ar6ifj23yi jqmdabeakpejcli72t3
    hash: 33hjhxix7p6gyzn5ptgyes7sghyprujh
    variants: {}
    version: '1.0'
- adept-utils:
  arch: linux-x86_64
  compiler:
    name: gcc
    version: 4.9.2
  dependencies:
    boost: teesjv7ehpe5kssppjim5dk43a7qnowlq
    mpich: aa4ar6ifj23yi jqmdabeakpejcli72t3
    hash: kszrtkpbzac3ss2ixcjkcorlaybnptp4
    variants: {}
    version: 1.0.1
- boost:
  arch: linux-x86_64
  compiler:
    name: gcc
    version: 4.9.2
  dependencies: {}
  hash: teesjv7ehpe5kssppjim5dk43a7qnowlq
  variants: {}
  version: 1.59.0
...
```

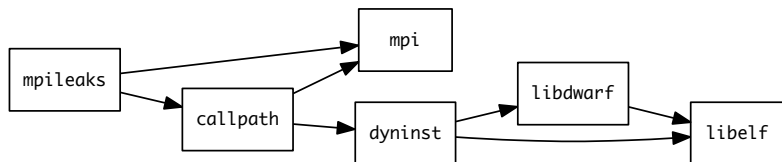
Abstract, normalized spec with some dependencies.

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-daqqpssxb6qbfzrtsezkmhus3xoflbsy
│   │   │   ├── openmpi-4.1.4-u64v26igvxyn23hysmklfums6tgjv5r
│   │   │   ├── dyninst-12.1.0-u64v26igvxyn23hysmklfums6tgjv5r
│   │   │   ├── libdwarf-20180129-u5eawkvaoc7vonabe6nndkcfwuv233cj
│   │   │   └── libelf-0.8.13-x46q4wm46ay4pltriijbgizxjrhbaka6
```

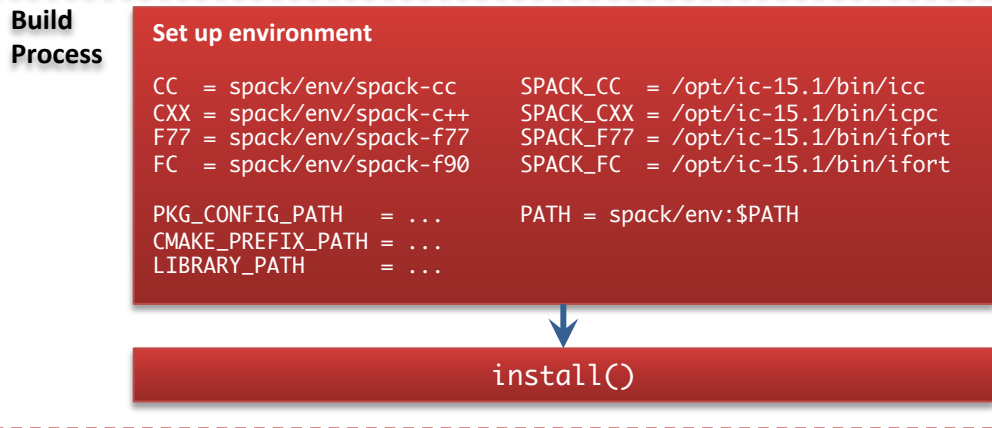
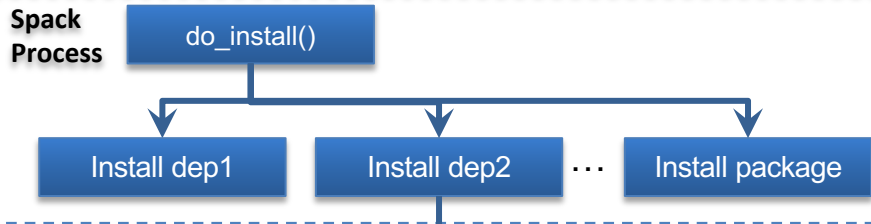
- Each unique dependency graph is a unique **configuration**.
- Each configuration in a unique directory.
 - Multiple configurations of the same package can coexist.

Hash of entire directed acyclic graph (DAG) is appended to each prefix.

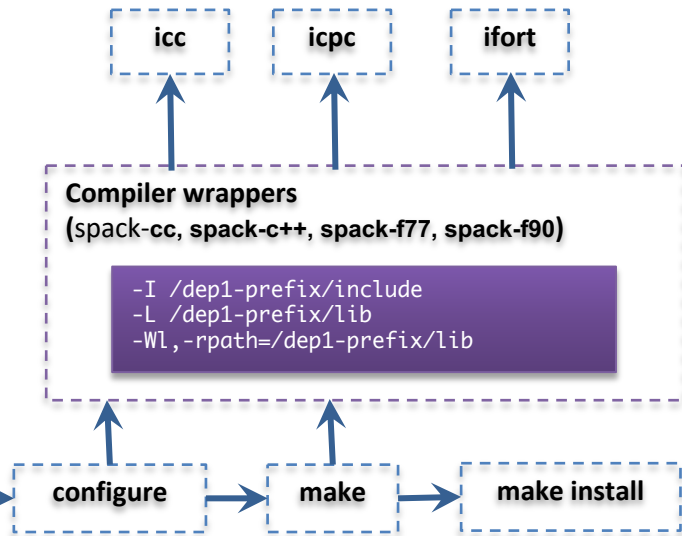
Installed packages automatically find dependencies

- 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



- **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)



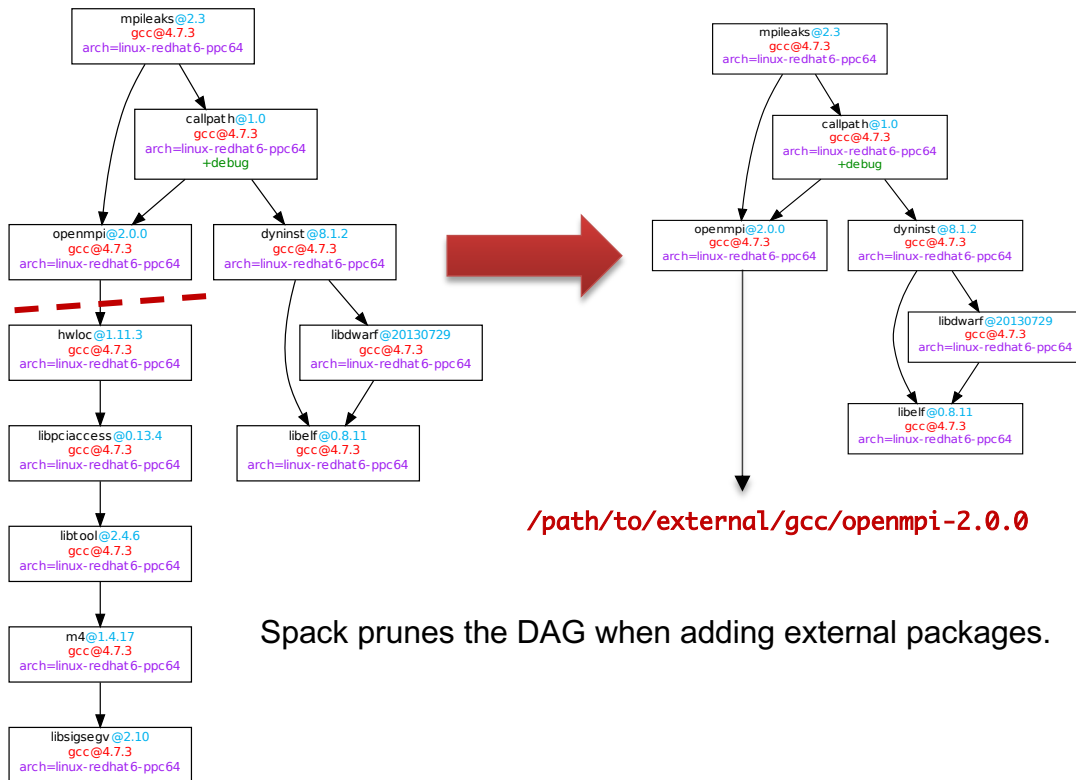
We can configure Spack to build with external software

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

packages.yaml

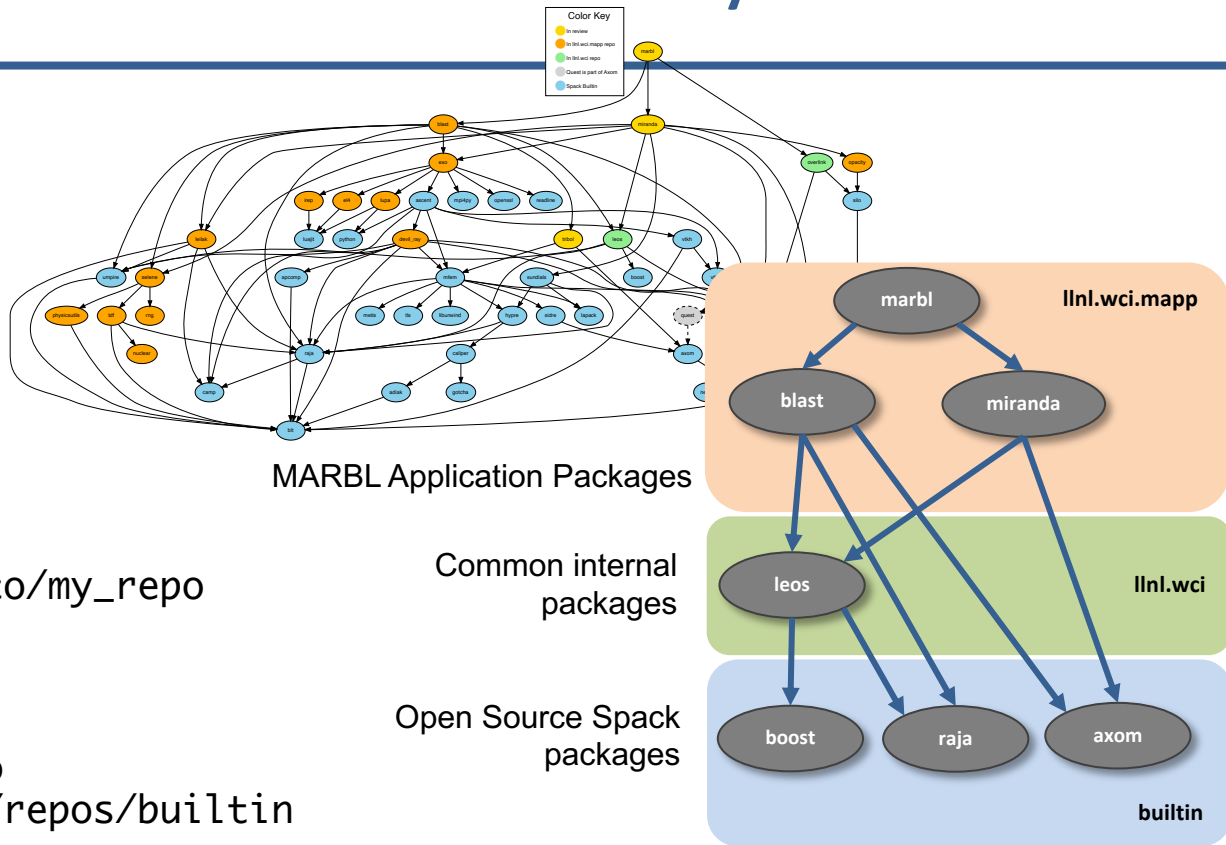
```
packages:  
  mpi:  
    buildable: False  
    paths:  
      openmpi@2.0.0 %gcc@4.7.3 arch=linux-rhel6-ppc64:  
        /path/to/external/gcc/openmpi-2.0.0  
      openmpi@1.10.3 %gcc@4.7.3 arch=linux-rhel6-ppc64:  
        /path/to/external/gcc/openmpi-1.10.3  
      ...
```

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



Spack package repositories allow stacks to be layered

LLNL MARBL multi-physics application



```
$ spack repo create /path/to/my_repo
```

```
$ spack repo add my_repo
```

```
$ spack repo list
```

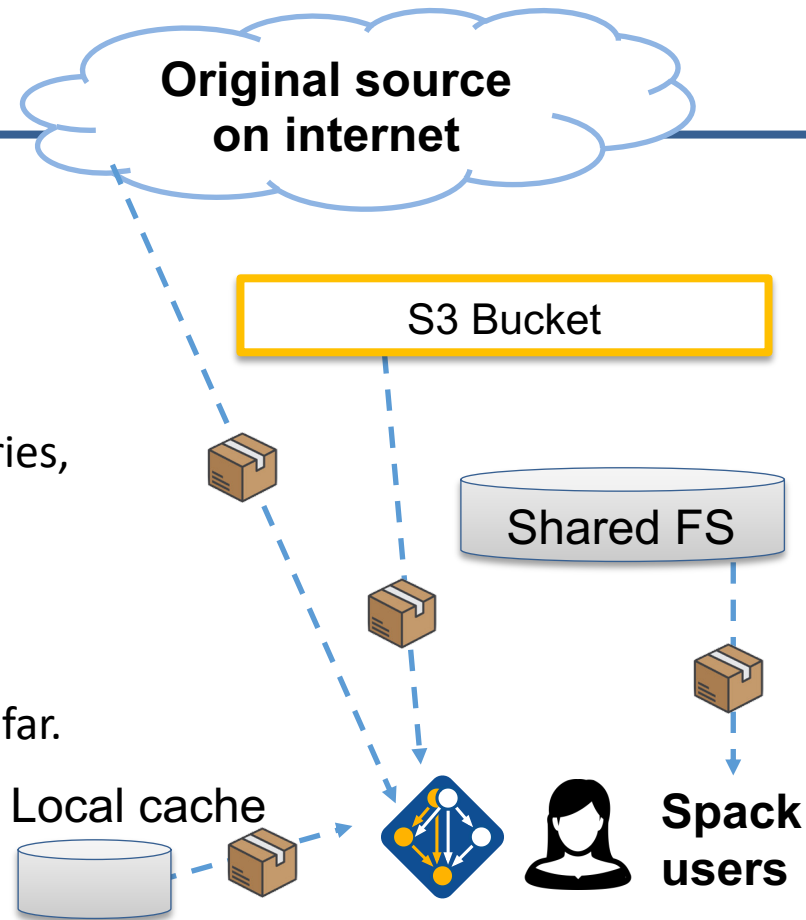
==> 2 package repositories.

```
my_repo    /path/to/my_repo
```

```
builtin    spack/var/spack/repos/builtin
```

Spack mirrors

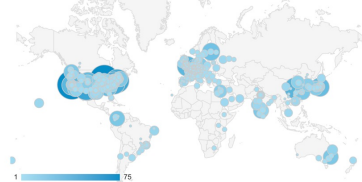
- Spack allows you to define *mirrors*:
 - Directories in the filesystem
 - On a web server
 - In an S3 bucket
- Mirrors are archives of fetched tarballs, repositories, and other resources needed to build
 - Can also contain binary packages
- By default, Spack maintains a mirror in `var/spack/cache` of everything you've fetched so far.
- You can host mirrors internal to your site
 - See the documentation for more details



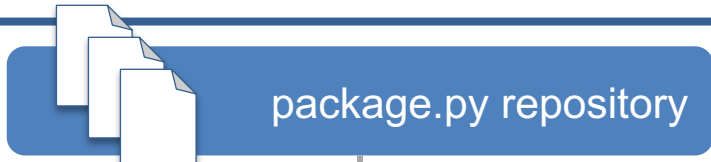
The concretizer includes information from packages, configuration, and CLI

Dependency solving is NP-hard

Contributors



- new versions
- new dependencies
- new constraints



package.py repository



concretizer



spack developers



default config
packages.yaml

admins,
users



local preferences config
packages.yaml

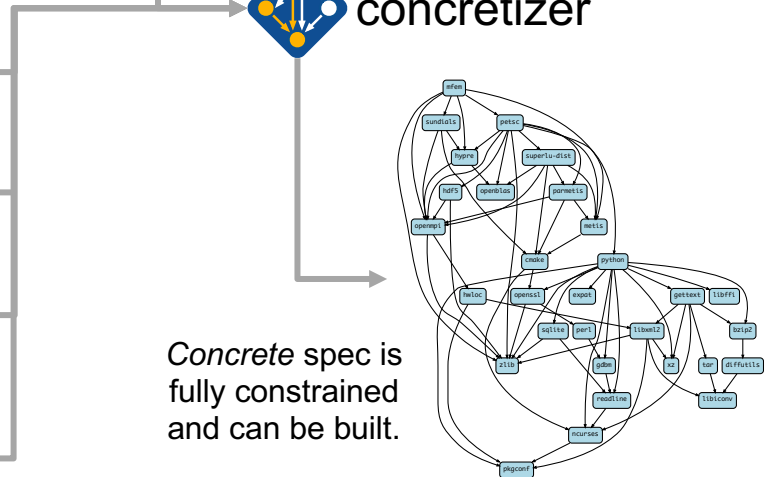
users



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.



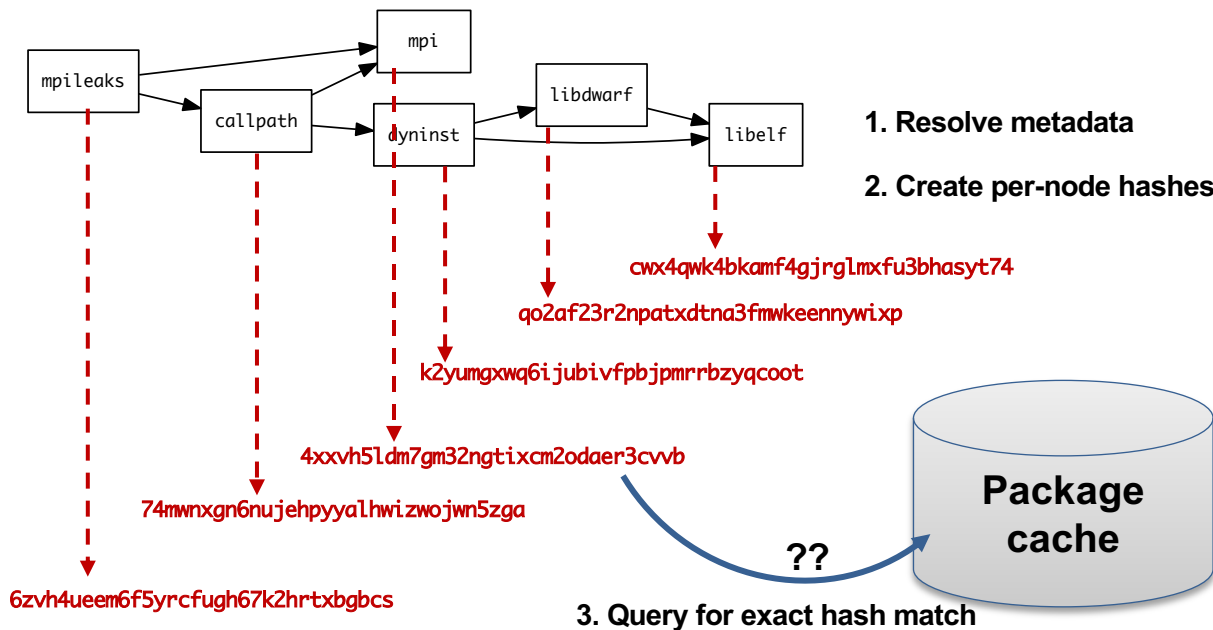
We use logic programming to simplify package solving

- 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



- 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", "lwatuysmwkhuahrncywvn77icdhs6mn").
imposed_constraint("lwatuysmwkhuahrncywvn77icdhs6mn", "node", "openssl").
imposed_constraint("lwatuysmwkhuahrncywvn77icdhs6mn", "version", "openssl", "1.1.1g").
imposed_constraint("lwatuysmwkhuahrncywvn77icdhs6mn", "node_platform_set", "openssl", "darwin").
imposed_constraint("lwatuysmwkhuahrncywvn77icdhs6mn", "node_os_set", "openssl", "catalina").
imposed_constraint("lwatuysmwkhuahrncywvn77icdhs6mn", "node_target_set", "openssl", "x86_64").
imposed_constraint("lwatuysmwkhuahrncywvn77icdhs6mn", "variant_set", "openssl", "systemcerts", "True").
imposed_constraint("lwatuysmwkhuahrncywvn77icdhs6mn", "node_compiler_set", "openssl", "apple-clang").
imposed_constraint("lwatuysmwkhuahrncywvn77icdhs6mn", "node_compiler_version_set", "openssl", "apple-clang", "12.0.0").
imposed_constraint("lwatuysmwkhuahrncywvn77icdhs6mn", "concrete", "openssl").
imposed_constraint("lwatuysmwkhuahrncywvn77icdhs6mn", "depends_on", "openssl", "zlib", "build").
imposed_constraint("lwatuysmwkhuahrncywvn77icdhs6mn", "depends_on", "openssl", "zlib", "link").
imposed_constraint("lwatuysmwkhuahrncywvn77icdhs6mn", "hash", "zlib", "x2anksgssxsxa7pcnhzg5k3dhgacglze").
```


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

```
(spack):solver> spack solve -II 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

```

- zzzgfs3 hdf5@1.10.7%apple-clang@13.0.0-cxx-fortran-hl-ipo-java-mpi+shared-szip-threadsafe+tools api=default t
- nsyl0vq Acmake@3.21.4%apple-clang@13.0.0-docs+ncurses+openmpi+ownlibs-qt build_type=Release arch=darwin-bi
- xdbaqeo ^ncurses@6.2%apple-clang@13.0.0-0-symlinks+termlib abi=none arch=darwin-bigsur-skylake
- kfuneok ^pkgconf@1.8.0%apple-clang@13.0.0 arch=darwin-bigsur-skylake
- 5ekd4ap ^openmpi@1.11%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
- xgt3t1s ^berkeley-db@18.1.40%apple-clang@13.0.0+cxx+docs+stl patches=b231fcc4d5c481f
- 65edjff6 ^bzp2@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
- fu7f5sr ^libiconv@1.16%apple-clang@13.0.0 libs=shared,static arch=darwin-bigsur-sky
- 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
- xev1jij ^zlib@1.2.11%apple-clang@13.0.0+optimize+pic+shared arch=darwin-bigsur-skylake
- xel1fobh ^openmpi@4.1.1%apple-clang@13.0.0-atomic-cuda-cxx-cxx_exceptions+gpgfs-internal-hwloc-java-legacy
- zrnus75 ^hwloc@2.6.0%apple-clang@13.0.0-cairo-cuda-glibudev+libxml2-netloc-nvml-opencl-pci-rocm+sho
- 1b4fnkf ^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
- h7jalvy ^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
```

```
(spack):spack> spack solve --reuse -II 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

```

- yfknfnp hdf5@1.10.7%apple-clang@12.0.5-cxx-fortran-hl-ipo-java-mpi+shared-szip-threadsafe+tools api=default
- zdam26e Acmake@3.21.1%apple-clang@12.0.5-docs+ncurses+openmpi+ownlibs-qt build_type=Release arch=darwin
- s315zxr ^ncurses@6.2%apple-clang@12.0.5-symlinks+termlib abi=none arch=darwin-bigsur-skylake
- us36bwr ^openmpi@1.11%apple-clang@12.0.5-docs+systemcerts arch=darwin-bigsur-skylake
- 74mwmxg ^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-atomic-cuda-cxx-cxx_exceptions+gpgfs-internal-hwloc-java-leg
- gjxyb77 ^hwloc@2.6.0%apple-clang@12.0.5-cairo-cuda-glibudev+libxml2-netloc-nvml-opencl-pci-rocm+sho
- skdn5zf ^libxml2@2.9.12%apple-clang@12.0.5-python arch=darwin-bigsur-skylake
- 47a0t3 ^libiconv@1.16%apple-clang@12.0.5 libs=shared,static arch=darwin-bigsur-skylake
- x2ymgx ^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
- hnc66ug ^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
- shngtd ^libedit@3.1-20210216%apple-clang@12.0.5 arch=darwin-bigsur-skylake
- jbkmtdd ^perl@5.34.0%apple-clang@12.0.5+cpanm+shared+threads arch=darwin-bigsur-skylake
- cnvkifs ^berkeley-db@18.1.40%apple-clang@12.0.5+cxx+docs+stl patches=b231fcc4d5c481f
- 7d5woqt ^bzp2@1.0.8%apple-clang@12.0.5-debug-pic+shared arch=darwin-bigsur-skylake
- vhd131 ^gdbm@1.19%apple-clang@12.0.5 arch=darwin-bigsur-skylake
- agy3v41 ^readline@8.1%apple-clang@12.0.5 arch=darwin-bigsur-skylake
```

Pure hash-based reuse: all misses

With reuse: 16 packages were reusable



Use `spack spec` to see the results of concretization

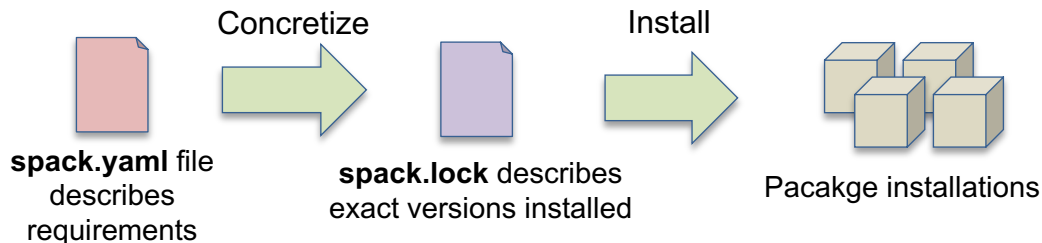
```
$ spack spec mpileaks
Input spec
```

```
-----
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



- 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": {
    "6s63so2kstp3zyvjezglndmavy6l3nu1": {
      "hdf5": {
        "version": "1.10.5",
        "arch": {
          "platform": "darwin",
          "platform_os": "mojave",
          "target": "x86_64"
        },
      },
      "compiler": {
        "name": "clang",
        "version": "10.0.0-apple"
      },
    },
    "namespace": "builtin",
    "parameters": {
```

We'll resume at: 10:30pm EDT

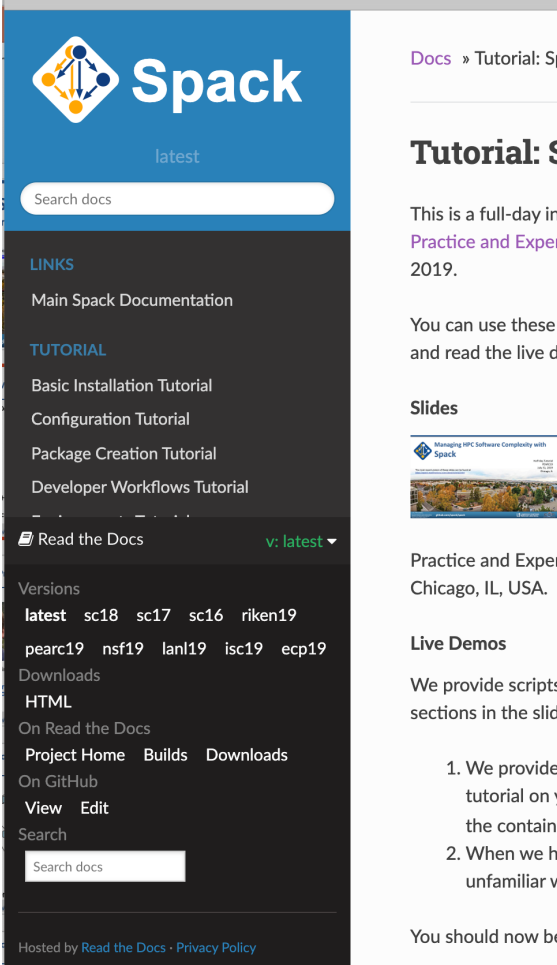
Find the slides and associated scripts here:

spack-tutorial.readthedocs.io

Remember to join Spack slack so you can get help after ISC!

slack.spack.io

Join the [#tutorial](#) channel!



The screenshot shows the Spack documentation page on Read the Docs. The top navigation bar is blue with the Spack logo and the word "Spack" in white. Below the logo, the word "latest" is displayed. A search bar is located in the top right corner. The main content area is dark grey and contains several sections: "LINKS" with a link to "Main Spack Documentation"; "TUTORIAL" with links to "Basic Installation Tutorial", "Configuration Tutorial", "Package Creation Tutorial", and "Developer Workflows Tutorial"; "Read the Docs" with a "v: latest" dropdown; "Versions" with a list of version tags: "latest", "sc18", "sc17", "sc16", "riken19", "pearc19", "nsf19", "lan19", "isc19", "ecp19"; "Downloads"; "HTML"; "On Read the Docs" with links to "Project Home", "Builds", and "Downloads"; "On GitHub" with links to "View" and "Edit"; and a "Search" bar with a "Search docs" input field. The footer of the page includes the text "Hosted by Read the Docs · Privacy Policy". On the right side of the page, there is a sidebar with a "Docs" link, a "Tutorial: S" heading, a paragraph of text, a "Slides" section with a thumbnail image, and a "Live Demos" section with a list of items.


Docs » Tutorial: Sp

Tutorial: S

This is a full-day int
[Practice and Experi](#)
2019.

You can use these n
and read the live de

Slides



[Practice and Experi](#)
Chicago, IL, USA.

Live Demos

We provide scripts
sections in the slide

1. We provide 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: 1:30pm EDT

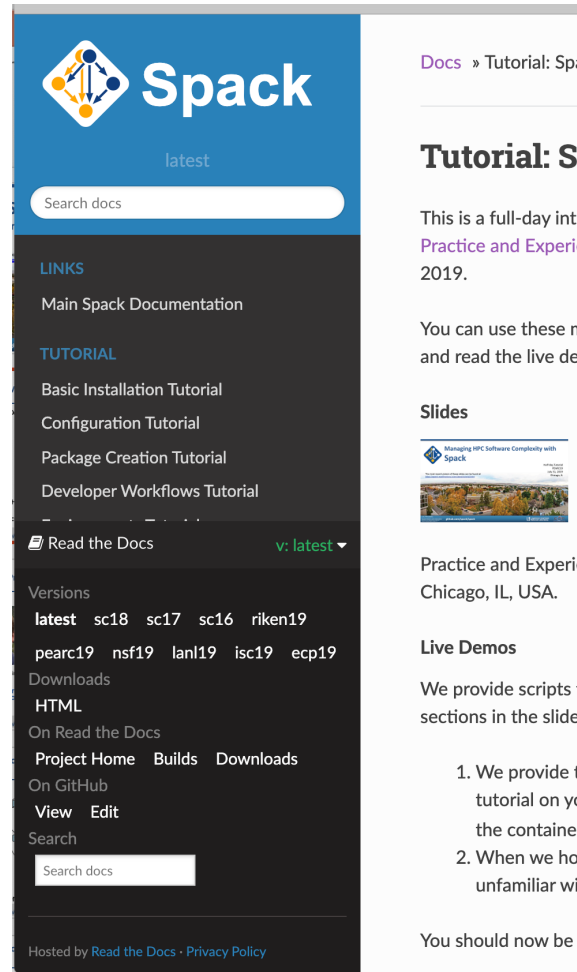
Find the slides and associated scripts here:

spack-tutorial.readthedocs.io

Remember to join Spack slack so you can get help after ISC!

slack.spack.io

Join the [#tutorial](#) channel!



The screenshot shows the Spack documentation website. At the top left is the Spack logo, a diamond shape with four nodes and connecting lines. To its right is the word "Spack" in a large, white, sans-serif font. Below the logo and name is the word "latest" in a smaller font. A search bar with the placeholder text "Search docs" is positioned below the "latest" text. The main content area is dark grey and contains several sections: "LINKS" with a link to "Main Spack Documentation"; "TUTORIAL" with links to "Basic Installation Tutorial", "Configuration Tutorial", "Package Creation Tutorial", and "Developer Workflows Tutorial"; "Read the Docs" with a green "v: latest" dropdown; "Versions" with a list of version identifiers: "latest", "sc18", "sc17", "sc16", "riken19", "pearc19", "nsf19", "lan19", "isc19", "ecp19"; "Downloads"; "HTML"; "On Read the Docs" with links to "Project Home", "Builds", and "Downloads"; "On GitHub" with links to "View" and "Edit"; and another "Search" bar with the placeholder text "Search docs". At the bottom of the page, it says "Hosted by Read the Docs · Privacy Policy". On the right side of the page, there is a sidebar with a "Docs" link and a "Tutorial: Spack" section. Below this, there is a "Tutorial: Spack" section with the text "This is a full-day introductory tutorial on Spack. Practice and Experience with Spack 2019." and "You can use these notes and read the live demo." Below this is a "Slides" section with a thumbnail image and the text "Managing HPC Software Complexity with Spack". Below the slides is a "Practice and Experience with Spack 2019" section with the text "Chicago, IL, USA." and a "Live Demos" section with the text "We provide scripts and sections in the slides." Below this is a list of two items: "1. We provide a full-day tutorial on your system, covering the containerization of the workflow." and "2. When we have time, we will show you some unfamiliar workflows." Below the list is the text "You should now be able to..."

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: Scripting

Follow script at spack-tutorial.readthedocs.io

We'll resume at: 3:30pm EDT

Find the slides and associated scripts here:

spack-tutorial.readthedocs.io

Remember to join Spack slack so you can get help after ISC!

slack.spack.io

Join the [#tutorial](#) channel!

The screenshot shows the Spack documentation website. At the top left is the Spack logo, a diamond shape with four nodes and connecting lines. To its right is the word "Spack" in a large, white, sans-serif font. Below the logo and name is the word "latest" in a smaller font. A search bar with the placeholder text "Search docs" is positioned below the "latest" text. The main content area is dark grey and contains several sections: "LINKS" with a link to "Main Spack Documentation"; "TUTORIAL" with links to "Basic Installation Tutorial", "Configuration Tutorial", "Package Creation Tutorial", and "Developer Workflows Tutorial"; "Read the Docs" with a green "v: latest" dropdown; "Versions" with a list of version tags: "latest", "sc18", "sc17", "sc16", "riken19", "pearc19", "nsf19", "lan19", "isc19", "ecp19"; "Downloads"; "HTML"; "On Read the Docs" with links to "Project Home", "Builds", and "Downloads"; "On GitHub" with links to "View" and "Edit"; and another "Search" bar with the placeholder text "Search docs". At the bottom of the page, it says "Hosted by Read the Docs · Privacy Policy". On the right side of the page, there is a sidebar with a "Docs" link and a "Tutorial: Spack" section. Below this, there is a "Tutorial: Spack" section with the text "This is a full-day introductory tutorial on Spack. Practice and Experience with Spack 2019." and "You can use these notes and read the live demo." Below this is a "Slides" section with a thumbnail image and the text "Managing HPC Software Complexity with Spack". Below the slides is a "Practice and Experience with Spack 2019" section with the text "Chicago, IL, USA." and a "Live Demos" section with the text "We provide scripts and sections in the slides." Below this is a list of two items: "1. We provide a full-day introductory tutorial on your system, covering the containerization of the workflow." and "2. When we have time, we will demonstrate the workflow on unfamiliar workflows." Below the list is the text "You should now be able to..."

Hands-on Time: Modules

Follow script at spack-tutorial.readthedocs.io



Hands-on Time: Mirrors and Build Caches

Follow script at spack-tutorial.readthedocs.io



More Features and the Road Ahead

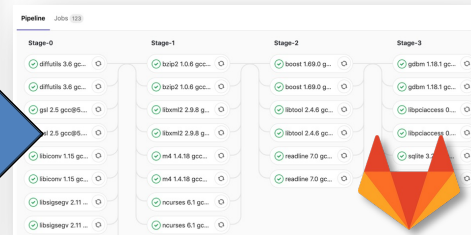


Spack environments are the foundation of Spack CI

- `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

```
spack:
  definitions:
  - pkgs:
    - readline@7.0
  - compilers:
    - '%gcc@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-groups: 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

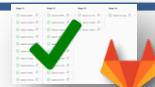


gitlab.spack.io

```
spack:
  homepage: https://spack.io
  license: MIT
  build_systems:
  - cmake
  - meson
  - make
  - make4g
  - make4j
  - make4m
  - make4p
  - make4s
  - make4t
  - make4u
  - make4v
  - make4w
  - make4x
  - make4y
  - make4z
  - make4aa
  - make4ab
  - make4ac
  - make4ad
  - make4ae
  - make4af
  - make4ag
  - make4ah
  - make4ai
  - make4aj
  - make4ak
  - make4al
  - make4am
  - make4an
  - make4ao
  - make4ap
  - make4aq
  - make4ar
  - make4as
  - make4at
  - make4au
  - make4av
  - make4aw
  - make4ax
  - make4ay
  - make4az
  - make4ba
  - make4bb
  - make4bc
  - make4bd
  - make4be
  - make4bf
  - make4bg
  - make4bh
  - make4bi
  - make4bj
  - make4bk
  - make4bl
  - make4bm
  - make4bn
  - make4bo
  - make4bp
  - make4bq
  - make4br
  - make4bs
  - make4bt
  - make4bu
  - make4bv
  - make4bw
  - make4bx
  - make4by
  - make4bz
  - make4ca
  - make4cb
  - make4cc
  - make4cd
  - make4ce
  - make4cf
  - make4cg
  - make4ch
  - make4ci
  - make4cj
  - make4ck
  - make4cl
  - make4cm
  - make4cn
  - make4co
  - make4cp
  - make4cq
  - make4cr
  - make4cs
  - make4ct
  - make4cu
  - make4cv
  - make4cw
  - make4cx
  - make4cy
  - make4cz
  - make4da
  - make4db
  - make4dc
  - make4dd
  - make4de
  - make4df
  - make4dg
  - make4dh
  - make4di
  - make4dj
  - make4dk
  - make4dl
  - make4dm
  - make4dn
  - make4do
  - make4dp
  - make4dq
  - make4dr
  - make4ds
  - make4dt
  - make4du
  - make4dv
  - make4dw
  - make4dx
  - make4dy
  - make4dz
  - make4ea
  - make4eb
  - make4ec
  - make4ed
  - make4ee
  - make4ef
  - make4eg
  - make4eh
  - make4ei
  - make4ej
  - make4ek
  - make4el
  - make4em
  - make4en
  - make4eo
  - make4ep
  - make4eq
  - make4er
  - make4es
  - make4et
  - make4eu
  - make4ev
  - make4ew
  - make4ex
  - make4ey
  - make4ez
  - make4fa
  - make4fb
  - make4fc
  - make4fd
  - make4fe
  - make4ff
  - make4fg
  - make4fh
  - make4fi
  - make4fj
  - make4fk
  - make4fl
  - make4fm
  - make4fn
  - make4fo
  - make4fp
  - make4fq
  - make4fr
  - make4fs
  - make4ft
  - make4fu
  - make4fv
  - make4fw
  - make4fx
  - make4fy
  - make4fz
  - make4ga
  - make4gb
  - make4gc
  - make4gd
  - make4ge
  - make4gf
  - make4gg
  - make4gh
  - make4gi
  - make4gj
  - make4gk
  - make4gl
  - make4gm
  - make4gn
  - make4go
  - make4gp
  - make4gq
  - make4gr
  - make4gs
  - make4gt
  - make4gu
  - make4gv
  - make4gw
  - make4gx
  - make4gy
  - make4gz
  - make4ha
  - make4hb
  - make4hc
  - make4hd
  - make4he
  - make4hf
  - make4hg
  - make4hh
  - make4hi
  - make4hj
  - make4hk
  - make4hl
  - make4hm
  - make4hn
  - make4ho
  - make4hp
  - make4hq
  - make4hr
  - make4hs
  - make4ht
  - make4hu
  - make4hv
  - make4hw
  - make4hx
  - make4hy
  - make4hz
  - make4ia
  - make4ib
  - make4ic
  - make4id
  - make4ie
  - make4if
  - make4ig
  - make4ih
  - make4ii
  - make4ij
  - make4ik
  - make4il
  - make4im
  - make4in
  - make4io
  - make4ip
  - make4iq
  - make4ir
  - make4is
  - make4it
  - make4iu
  - make4iv
  - make4iw
  - make4ix
  - make4iy
  - make4iz
  - make4ja
  - make4jb
  - make4jc
  - make4jd
  - make4je
  - make4jf
  - make4jg
  - make4jh
  - make4ji
  - make4jj
  - make4jk
  - make4jl
  - make4jm
  - make4jn
  - make4jo
  - make4jp
  - make4jq
  - make4jr
  - make4js
  - make4jt
  - make4ju
  - make4jv
  - make4jw
  - make4jx
  - make4jy
  - make4jz
  - make4ka
  - make4kb
  - make4kc
  - make4kd
  - make4ke
  - make4kf
  - make4kg
  - make4kh
  - make4ki
  - make4kj
  - make4kk
  - make4kl
  - make4km
  - make4kn
  - make4ko
  - make4kp
  - make4kq
  - make4kr
  - make4ks
  - make4kt
  - make4ku
  - make4kv
  - make4kw
  - make4kx
  - make4ky
  - make4kz
  - make4la
  - make4lb
  - make4lc
  - make4ld
  - make4le
  - make4lf
  - make4lg
  - make4lh
  - make4li
  - make4lj
  - make4lk
  - make4ll
  - make4lm
  - make4ln
  - make4lo
  - make4lp
  - make4lq
  - make4lr
  - make4ls
  - make4lt
  - make4lu
  - make4lv
  - make4lw
  - make4lx
  - make4ly
  - make4lz
  - make4ma
  - make4mb
  - make4mc
  - make4md
  - make4me
  - make4mf
  - make4mg
  - make4mh
  - make4mi
  - make4mj
  - make4mk
  - make4ml
  - make4mn
  - make4mo
  - make4mp
  - make4mq
  - make4mr
  - make4ms
  - make4mt
  - make4mu
  - make4mv
  - make4mw
  - make4mx
  - make4my
  - make4mz
  - make4na
  - make4nb
  - make4nc
  - make4nd
  - make4ne
  - make4nf
  - make4ng
  - make4nh
  - make4ni
  - make4nj
  - make4nk
  - make4nl
  - make4nm
  - make4nn
  - make4no
  - make4np
  - make4nq
  - make4nr
  - make4ns
  - make4nt
  - make4nu
  - make4nv
  - make4nw
  - make4nx
  - make4ny
  - make4nz
  - make4oa
  - make4ob
  - make4oc
  - make4od
  - make4oe
  - make4of
  - make4og
  - make4oh
  - make4oi
  - make4oj
  - make4ok
  - make4ol
  - make4om
  - make4on
  - make4oo
  - make4op
  - make4oq
  - make4or
  - make4os
  - make4ot
  - make4ou
  - make4ov
  - make4ow
  - make4ox
  - make4oy
  - make4oz
  - make4pa
  - make4pb
  - make4pc
  - make4pd
  - make4pe
  - make4pf
  - make4pg
  - make4ph
  - make4pi
  - make4pj
  - make4pk
  - make4pl
  - make4pm
  - make4pn
  - make4po
  - make4pp
  - make4pq
  - make4pr
  - make4ps
  - make4pt
  - make4pu
  - make4pv
  - make4pw
  - make4px
  - make4py
  - make4pz
  - make4qa
  - make4qb
  - make4qc
  - make4qd
  - make4qe
  - make4qf
  - make4qg
  - make4qh
  - make4qi
  - make4qj
  - make4qk
  - make4ql
  - make4qm
  - make4qn
  - make4qo
  - make4qp
  - make4qq
  - make4qr
  - make4qs
  - make4qt
  - make4qu
  - make4qv
  - make4qw
  - make4qx
  - make4qy
  - make4qz
  - make4ra
  - make4rb
  - make4rc
  - make4rd
  - make4re
  - make4rf
  - make4rg
  - make4rh
  - make4ri
  - make4rj
  - make4rk
  - make4rl
  - make4rm
  - make4rn
  - make4ro
  - make4rp
  - make4rq
  - make4rr
  - make4rs
  - make4rt
  - make4ru
  - make4rv
  - make4rw
  - make4rx
  - make4ry
  - make4rz
  - make4sa
  - make4sb
  - make4sc
  - make4sd
  - make4se
  - make4sf
  - make4sg
  - make4sh
  - make4si
  - make4sj
  - make4sk
  - make4sl
  - make4sm
  - make4sn
  - make4so
  - make4sp
  - make4sq
  - make4sr
  - make4ss
  - make4st
  - make4su
  - make4sv
  - make4sw
  - make4sx
  - make4sy
  - make4sz
  - make4ta
  - make4tb
  - make4tc
  - make4td
  - make4te
  - make4tf
  - make4tg
  - make4th
  - make4ti
  - make4tj
  - make4tk
  - make4tl
  - make4tm
  - make4tn
  - make4to
  - make4tp
  - make4tq
  - make4tr
  - make4ts
  - make4tt
  - make4tu
  - make4tv
  - make4tw
  - make4tx
  - make4ty
  - make4tz
  - make4ua
  - make4ub
  - make4uc
  - make4ud
  - make4ue
  - make4uf
  - make4ug
  - make4uh
  - make4ui
  - make4uj
  - make4uk
  - make4ul
  - make4um
  - make4un
  - make4uo
  - make4up
  - make4uq
  - make4ur
  - make4us
  - make4ut
  - make4uu
  - make4uv
  - make4uw
  - make4ux
  - make4uy
  - make4uz
  - make4va
  - make4vb
  - make4vc
  - make4vd
  - make4ve
  - make4vf
  - make4vg
  - make4vh
  - make4vi
  - make4vj
  - make4vk
  - make4vl
  - make4vm
  - make4vn
  - make4vo
  - make4vp
  - make4vq
  - make4vr
  - make4vs
  - make4vt
  - make4vu
  - make4vv
  - make4vw
  - make4vx
  - make4vy
  - make4vz
  - make4wa
  - make4wb
  - make4wc
  - make4wd
  - make4we
  - make4wf
  - make4wg
  - make4wh
  - make4wi
  - make4wj
  - make4wk
  - make4wl
  - make4wm
  - make4wn
  - make4wo
  - make4wp
  - make4wq
  - make4wr
  - make4ws
  - make4wt
  - make4wu
  - make4wv
  - make4ww
  - make4wx
  - make4wy
  - make4wz
  - make4xa
  - make4xb
  - make4xc
  - make4xd
  - make4xe
  - make4xf
  - make4xg
  - make4xh
  - make4xi
  - make4xj
  - make4xk
  - make4xl
  - make4xm
  - make4xn
  - make4xo
  - make4xp
  - make4xq
  - make4xr
  - make4xs
  - make4xt
  - make4xu
  - make4xv
  - make4xw
  - make4xx
  - make4xy
  - make4xz
  - make4ya
  - make4yb
  - make4yc
  - make4yd
  - make4ye
  - make4yf
  - make4yg
  - make4yh
  - make4yi
  - make4yj
  - make4yk
  - make4yl
  - make4ym
  - make4yn
  - make4yo
  - make4yp
  - make4yq
  - make4yr
  - make4ys
  - make4yt
  - make4yu
  - make4yv
  - make4yw
  - make4yx
  - make4yy
  - make4yz
  - make4za
  - make4zb
  - make4zc
  - make4zd
  - make4ze
  - make4zf
  - make4zg
  - make4zh
  - make4zi
  - make4zj
  - make4zk
  - make4zl
  - make4zm
  - make4zn
  - make4zo
  - make4zp
  - make4zq
  - make4zr
  - make4zs
  - make4zt
  - make4zu
  - make4zv
  - make4zw
  - make4zx
  - make4zy
  - make4zz
```

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)

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

- GitLab CI builds (changed) packages
- On every pull request
 - On every release branch

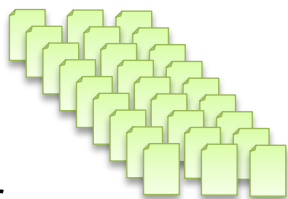
- **New security model supports untrusted contributions from forks**
 - Sandboxed build caches for test builds; Authoritative builds on mainline only after approved merge

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

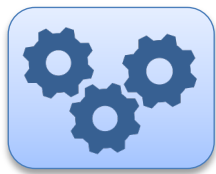
Spack's model lowers the maintenance burden of optimized software stacks



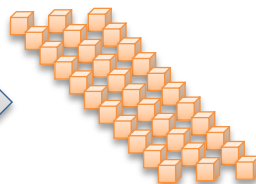
Traditional OS package manager



Recipe per package configuration
(need rewrites for new systems)



Build farm



Portable (unoptimized)
x86_64 binaries



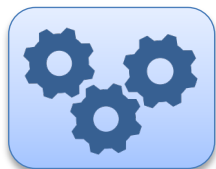
One software stack upgraded over time



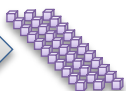
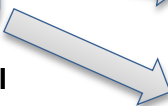
Spack



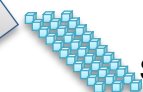
Parameterized recipe per package
(Same recipe evolves for all targets)



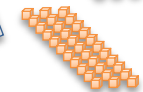
Build farm / CI



Optimized Graviton2 binaries



Optimized Skylake binaries



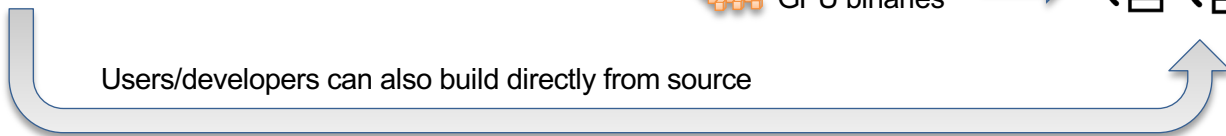
Optimized GPU binaries



Many software stacks

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

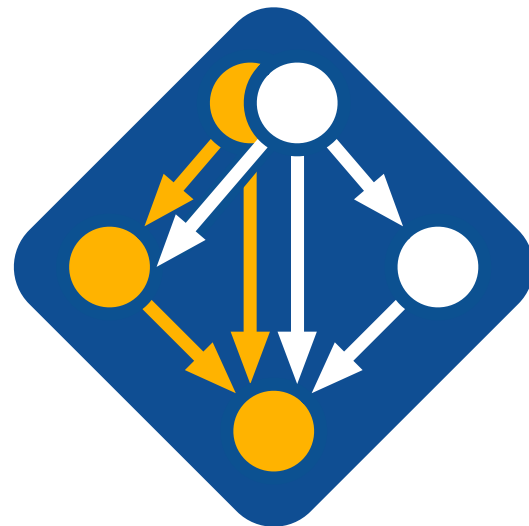
Users/developers can also build directly from source



Spack v0.18.0 was released on May 28!

- Major new features:

1. `--reuse` enabled by default
 - Reuse installed packages and build caches
 - Use `spack install --fresh` to get the old behavior
2. Finer-grained spec hash + provenance
3. Better error messages
4. Unify *when possible* in environments
5. Cray manifest support
6. Windows support
7. New binary format + hardened package signing
8. Bootstrap mirror generation (for air gaps)
9. Makefile generation
10. Conditional variant values and sticky variants



github.com/spack/spack

With v0.18, Spack has a public binary cache

```
# latest v0.18.x release binaries
```

```
spack mirror add https://binaries.spack.io/releases/v0.18
```

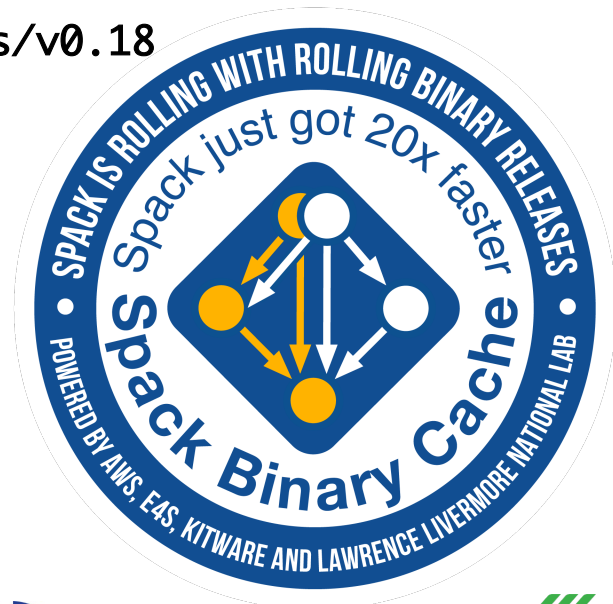
```
# rolling release: bleeding edge binaries
```

```
spack mirror add https://binaries.spack.io/develop
```

- 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

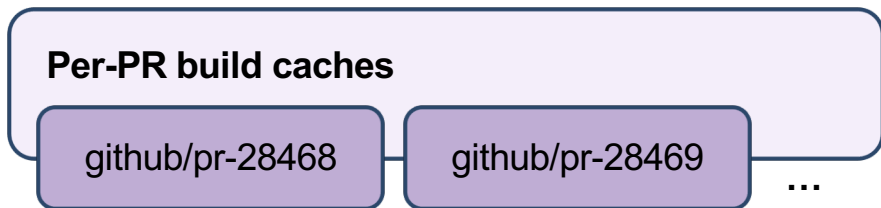
- Expect this list to expand!



kitware

Our infrastructure enables us to sustainably manage a binary distro

Separate, untrusted S3 buckets



Public, signed binaries in CloudFront distribution



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 v0.18 uses a different hash to identify builds

- **Coarse DAG hash prior to v0.18:**

- Hash included nodes and metadata about their link and run dependencies
- Information about build dependencies was not stored (to avoid rapidly changing hashes)
- Hash would not change if one of your `package.py` files was updated

- **Full DAG hash in v0.18:**

- Includes metadata about build, link, and run dependencies (all dependencies)
- Database stores build dependencies (better provenance)
- Hash includes a canonical hash of the `package.py` recipe

- **Some important points:**

- Hashes of already-installed specs and buildcaches will **not** change
- Churn is minimized by enabling `--reuse` by default (no issues with hash misses)
 - Won't rebuild every time there is a new `cmake` version, unless you ask for it with `--fresh`
- You can now have graphs now with multiple versions of the same build dependency

Spack can now find Cray PE manifests

- May 2022 Cray PE will ship with Spack-friendly package descriptions
- You can find installed packages and register them as externals with:

```
spack external read-cray-manifest
```

- This will register packages from the PE with Spack
 - Adds to database and `packages.yaml`
 - Use `spack install --reuse` to build with found packages.
- Should result in much less configuration required to use the Cray PE

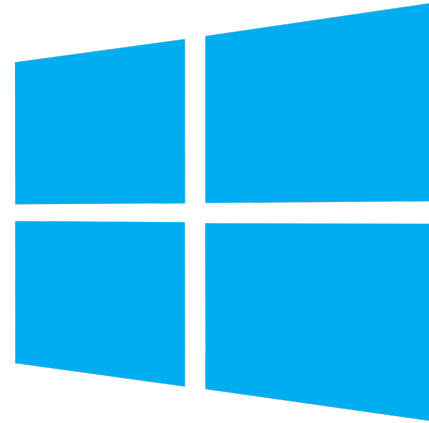
Unifying *when possible* in environments

- Spack environments have traditionally concretized two ways:
 - together: can only have one version of every dependency
 - separately: each package in the environment can have its own
- `unify:when_possible` feature is a best-effort middle ground:
 - Dependencies that can be consolidated (e.g. to an old/middle version) will be
 - Dependencies that conflict will be built separately
 - RPATH will continue to help keep things sane
- Solver work to do this was quite complex
 - Using multi-shot solving
 - Solve for runtime dependencies first
 - Then solve for build dependencies
 - Not fully optimal, but very fast
 - Approach brought E4S environment concretization from 2 hrs to ~1 minutes

concretizer:
`unify: when_possible`

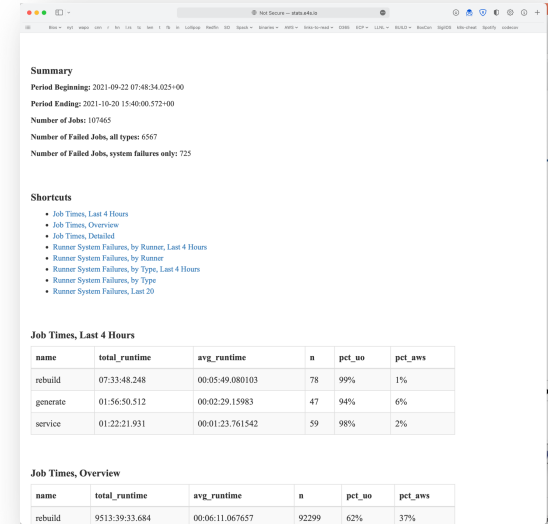
Spack on Windows is here!

- Until now, we've only supported Linux and macs
- Initial Windows support is in
 - Lots of core work to get to this point
 - Still a long way to go for all features
- 14 package files ported to Windows initially
 - Need more – hoping the community will help!
- Kitware and TechX did main development of this feature
- Hoping this gets us more exposure in other communities



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
 - when_possible will help even more, as it reduces this to one solve
- 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



The screenshot shows a web browser displaying the stats.e4s.io website. The page has a 'Summary' section with the following information:

- Period Beginning: 2021-09-22 07:48:34.025100
- Period Ending: 2021-10-20 15:40:00.572100
- Number of Jobs: 107465
- Number of Failed Jobs, all types: 6567
- Number of Failed Jobs, system failures only: 725

Below the summary is a 'Shortcuts' section with several links:

- 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

The 'Job Times, Last 4 Hours' section contains a table with the following data:

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	00:02:28.15983	47	94%	6%
service	01:22:21.931	00:01:23.761542	59	98%	2%

The 'Job Times, Overview' section contains a table with the following data:

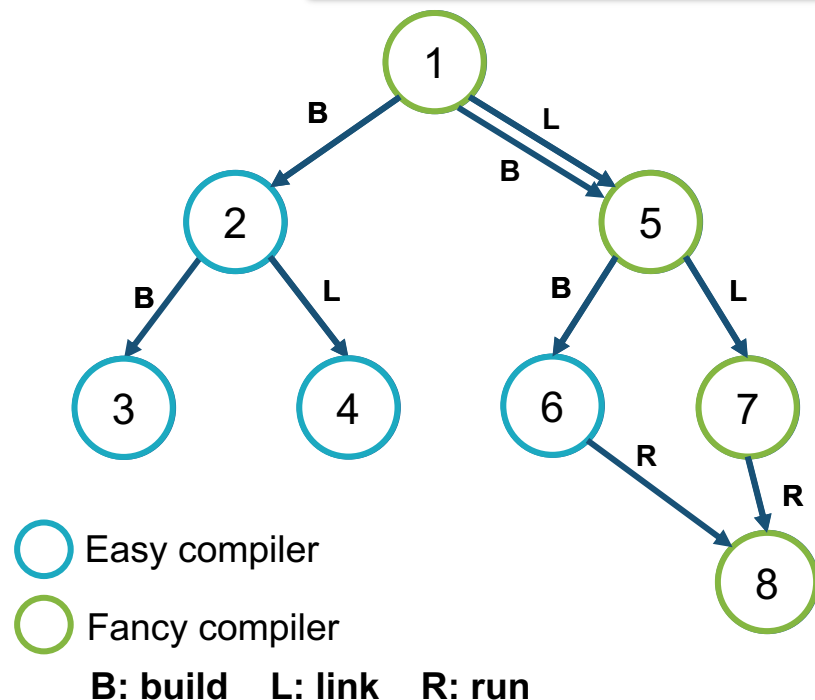
name	total_runtime	avg_runtime	n	pct_uo	pct_aws
rebuild	9513:39:33.684	00:06:11.067657	92299	62%	37%

<https://stats.e4s.io>

Spack v0.19 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 \leftrightarrow 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
```



Spack 0.19 Roadmap: compilers as dependencies

- **We need deeper modeling of compilers to handle compiler interoperability**

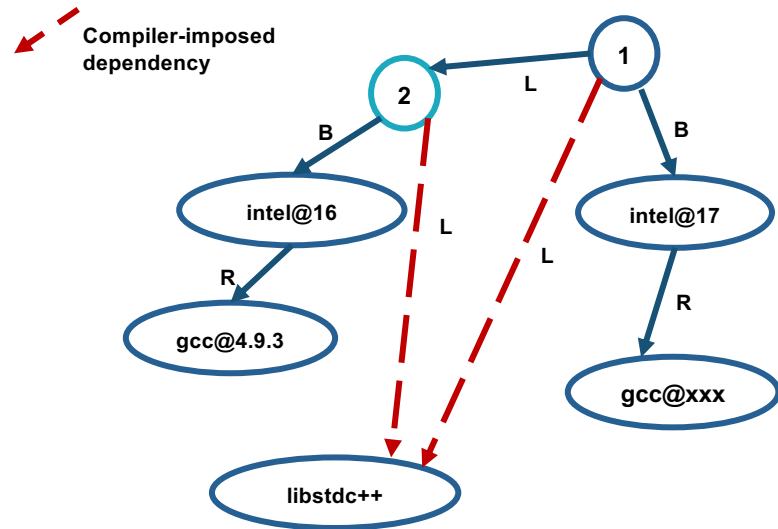
- libstdc++, libc++ compatibility
- Compilers that depend on compilers
- Linking executables with multiple compilers

- **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**

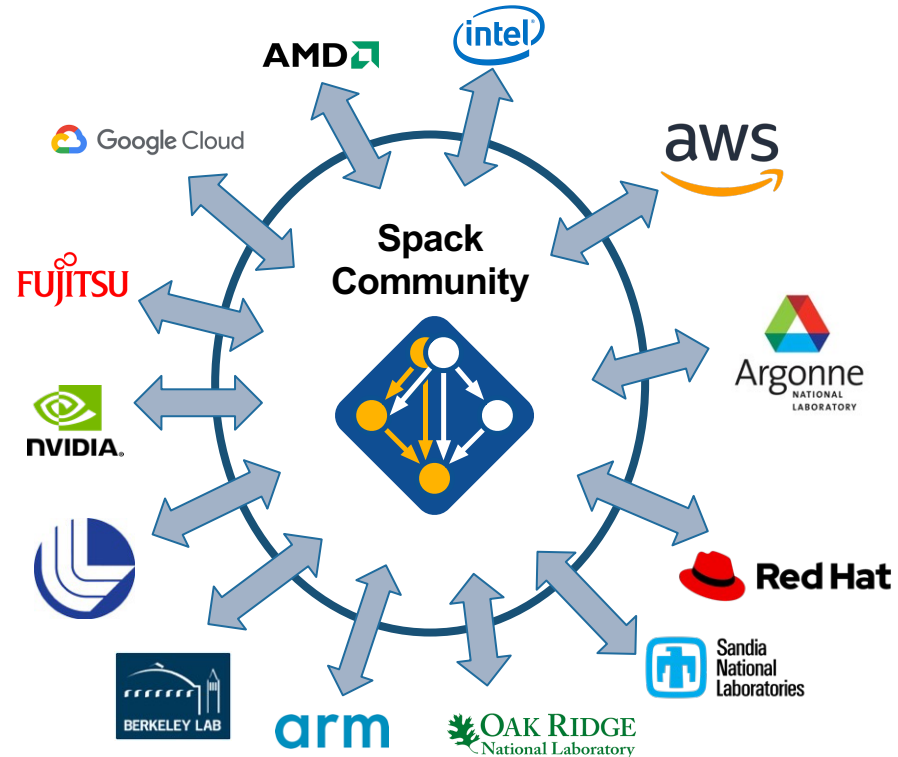
- Depend on **cxx@2011**, **cxx@2017**, **fortran@1995**, etc
- Depend on **openmp@4.5**, other compiler features
- Model languages, openmp, cuda, etc. as virtuals



Compilers and runtime libs fully modeled as dependencies

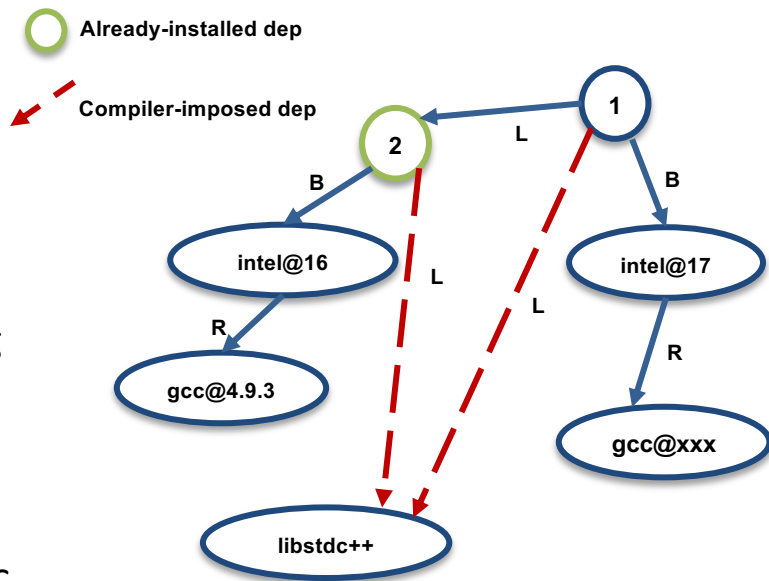
Spack's long-term strategy is based around broad adoption and collaboration

- **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 0.19 Roadmap: compilers as dependencies

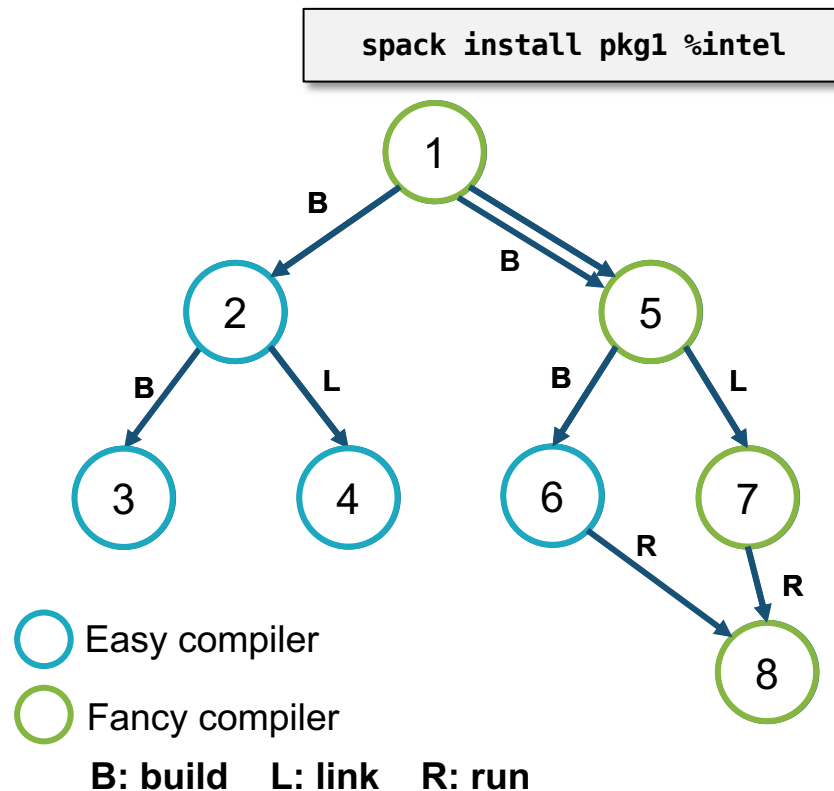
- **We need deeper modeling of compilers to handle compiler interoperability**
 - libstdc++, libc++ compatibility
 - Compilers that depend on compilers
 - Linking executables with multiple compilers
- **First prototype is complete!**
 - We've done successful builds of some packages using compilers as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



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!

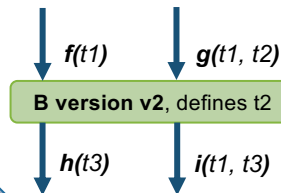
Ongoing research: BUILD is a 3-year research project, started at LLNL in 2020

- 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

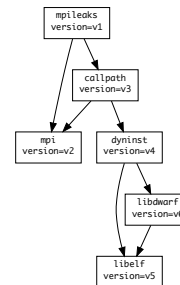
Human-generated constraints



Compatibility Models

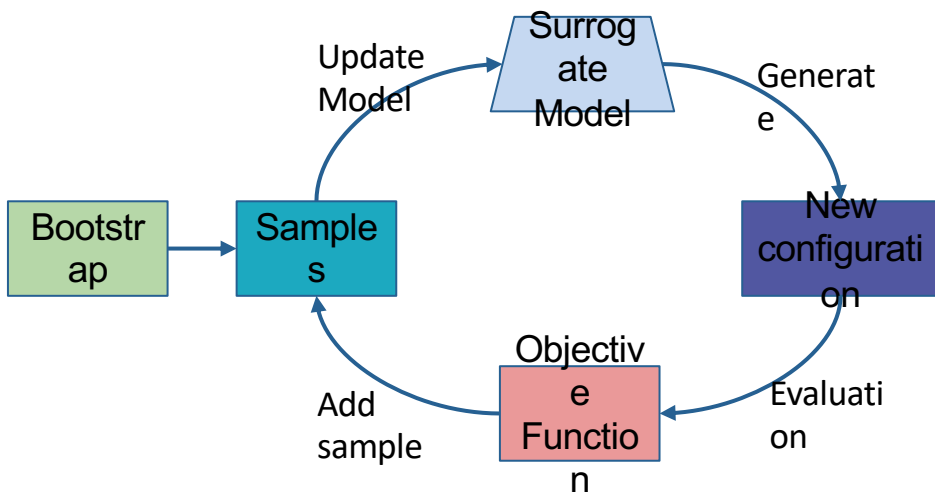


Solver



Resolved
ABI-compatible
Graph

Reliabil: An Active Learning based Configuration Selection Framework*

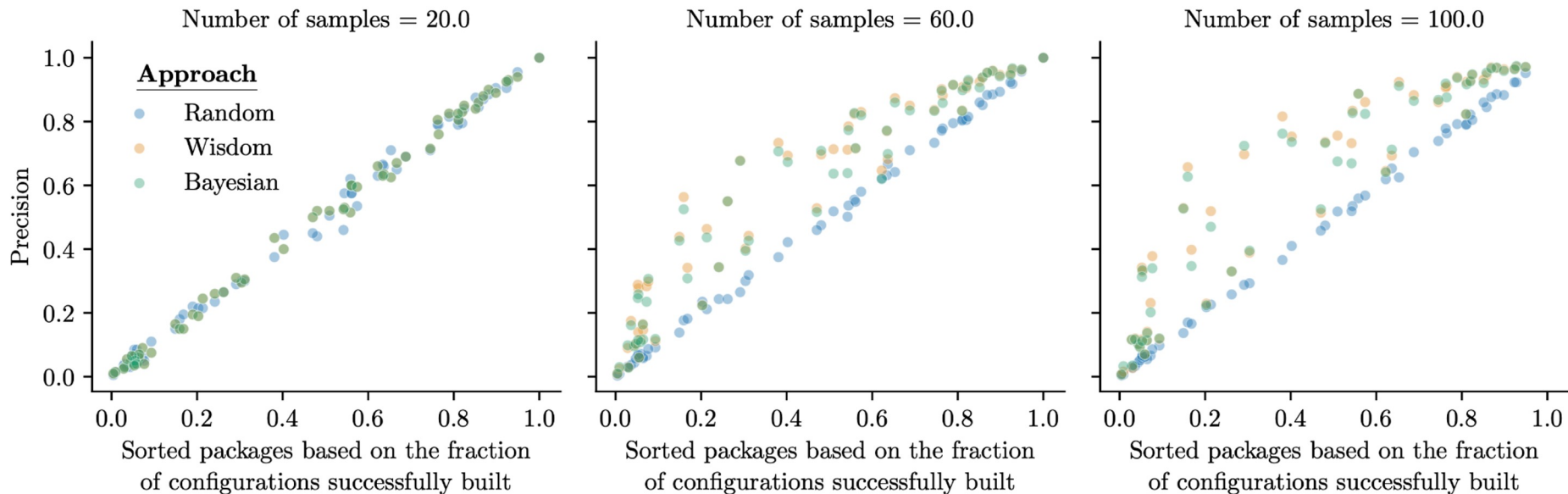


Reliabil iterates between fitting model and using it to select samples

- An active-learning-based approach for identifying high-fidelity package build configurations
- Iterative sampling method using only a limited set of samples.
— Suitable when the true objective function evaluations are expensive
- Surrogate model is used to compute the value of the objective for a configuration

*Reliabil: Searching for High-Fidelity Builds Using Active Learning; H.Menon, K. Parasyris, T. Scogland, T. Gamblin; MSR'2022

Reliabilbuild has significantly higher precision than *Random* selection



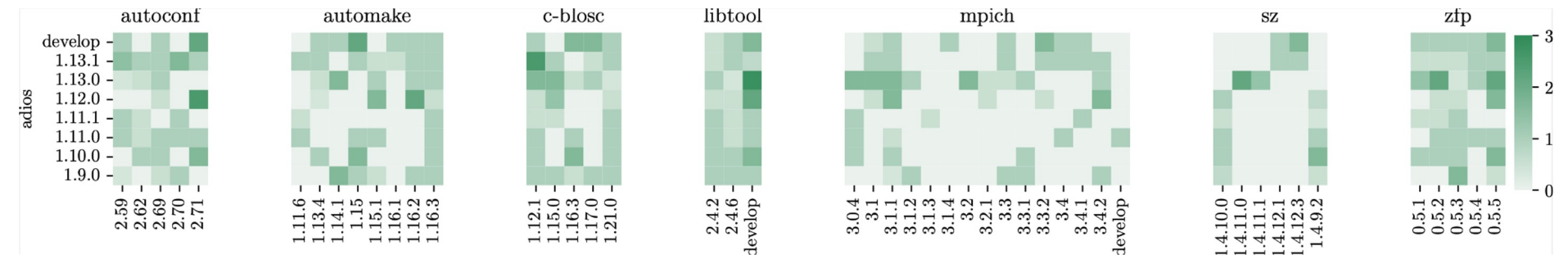
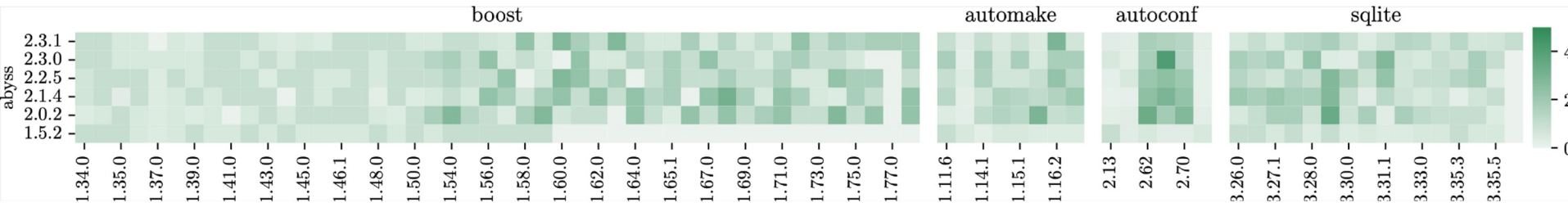
Package Importance Analysis

Root package	Dependency ranking				
abyss	autoconf: 0.37	autoconf+m4: 0.37	autoconf+perl: 0.37	libtool+autoconf: 0.29	abyss+autoconf: 0.27
adios	autoconf+perl: 0.27	autoconf+m4: 0.27	autoconf: 0.27	libtool: 0.22	libtool+m4: 0.22
ascent	vtk-h+openmpi: 0.14	vtk-h: 0.14	vtk-h+vtk-m: 0.14	conduit+zlib: 0.12	conduit+hdf5: 0.12
axom	lua: 0.08	lua+ncurses: 0.08	lua+readline: 0.08	lua+unzip: 0.08	axom+openmpi: 0.07
bolt	autoconf+perl: 0.37	autoconf+m4: 0.37	autoconf: 0.37	automake+autoconf: 0.32	automake+perl: 0.30
hypre	openblas+perl: 0.07	openblas: 0.07	hypre+openblas: 0.03	hypre+mpich: 0.02	mpich+findutils: 0.01
hpx	hpx+boost: 0.24	hpx+hwloc: 0.24	hpx+pkgconf: 0.24	hpx+python: 0.24	hpx: 0.24
heffte	heffte: 0.35	heffte+openmpi: 0.30	heffte+fftw: 0.24	cuda+libxml2: 0.19	mpich+findutils: 0.19
hdf5	mpich+findutils: 0.03	mpich+pkgconf: 0.03	mpich+libxml2: 0.03	mpich: 0.03	mpich+libpciaccess: 0.03
ninja	ninja+python: 0.03	python+ncurses: 0.01	python+readline: 0.01	python+pkgconf: 0.01	python+libffi: 0.01
omega-h	omega-h+zlib: 0.24	trilinos: 0.24	trilinos+openblas: 0.24	omega-h: 0.24	omega-h+trilinos: 0.18
openmpi	json-c: 0.30	mariadb+lz4: 0.30	meson: 0.30	gmp: 0.30	python+libffi: 0.30
openpmd-api	hdf5: 0.19	hdf5+zlib: 0.19	hdf5+openmpi: 0.19	hdf5+pkgconf: 0.19	hdf5+cmake: 0.19
papyrus	papyrus+mpich: 0.11	cmake+ncurses: 0.08	cmake: 0.08	papyrus+cmake: 0.08	mpich+findutils: 0.04
plasma	plasma: 0.52	plasma+openblas: 0.26	openblas+perl: 0.13	openblas: 0.13	plasma+cmake: 0.12

- A particular choice of version for packages can significantly affect the build outcome
- Importance metric: We use Jensen-Shannon (JS) divergence to compute the difference between the good and bad distribution.
- Some packages impact the build outcome more than others

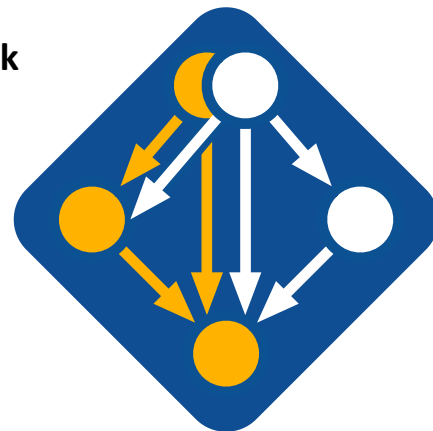
Relative ranking of dependencies based on importance can guide the exploration process

Pairwise Version Constraints Analysis



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](https://twitter.com/spackpm)

We hope to make distributing & using HPC software easy!



Disclaimer

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