



Managing HPC Software Complexity with Spack

ISC26 Tutorial
June 22, 2025

Todd Gamblin, Massimiliano Culpo, Gregory Becker, Adam Stewart, Michael Kuhn, Alec Scott, Harmen Stoppels and Kathleen Shea

 THE **LINUX** FOUNDATION

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





Todd Gamblin
LLNL



Alec Scott
LLNL



Greg Becker
LLNL



Massimiliano Culp
np-complete S.r.l.



Michael Kuhn
Otto von Guericke
University Magdeburg



Adam Stewart
TU Munich



Harmen Stoppels
Stoppels Consulting



Kathleen Shea
LLNL



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 **3,400 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 dropdown menu showing "v: latest", "Versions" with a list of version tags (latest, sc18, sc17, sc16, riken19, pearc19, nsf19, lan19, isc19, ecp19), "Downloads", "HTML", "On Read the Docs", "Project Home", "Builds", "Downloads", "On GitHub", "View", "Edit", "Search", and another search bar. 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 system, the container, and the workflow.
2. When we have unfamiliar workflows, we will provide a live demo.

You should now be able to...



Claim a VM instance at: bit.ly/spack-vms



	A	B	C	D	E
1	Spack Tutorial VM Instances				
2	Instructions:	1. Put your name in a box below to claim an account on a VM instance			
3		2. Log in to your VM:			
4		ssh <IP address>			
5		Login/password are both the username from your column below			
6					
7		Login / Password			
8	IP Address	spack1	spack2	spack3	spack4
9	35.90.43.21				
10	35.91.36.120		<input type="text" value="Your Name"/>		
11	34.217.149.171				
12	35.90.450.455				

If you're in the `spack2` column, your login and password are *both* `spack2`

```
ssh spack2@35.91.36.120
```

Claim a login by putting your name in the Google Sheet

Agenda (approximate)

Intro	09:00
Basics	09:15
Concepts	09:45
Environments	10:15
Break	11:00
Packaging	11:30
TBD*	12:15
End	13:00

*We will poll whether to do Deploying Large Software Stacks or Developer Workflows and Buildcaches & Containers

You can explore more sections on the tutorial site:

- Configuration
- Modules
- Deploying Large Stacks
- Buildcaches & Containers
- Developer Workflows
- Scripting

We would cover these in the full day version of the tutorial. (so tell ISC to accept it!)



We build codes from hundreds of small, complex pieces

Just when we're starting to solve the problem of how to create software using reusable parts, it founders on the nuts-and-bolts problems outside the software itself.

P. DuBois & T. Epperly. *Why Johnny Can't Build*. Scientific Programming. Sep/Oct 2003.

- Component-based software development dates back to the 60's
 - M.D. McIlroy, *Mass Produced Software Components*. NATO SE Conf., 1968
- **Pros are well known:**
 - Teams can and must reuse each others' work
 - Teams write less code, meet deliverables faster
- **Cons:**
 - Teams must ensure that components work together
 - Integration burden increases with each additional library
 - Integration must be repeated with each update to components
 - **Components must be vetted!**
- **Managing changes over time is becoming intractable**



Build-time incompatibility; fail fast

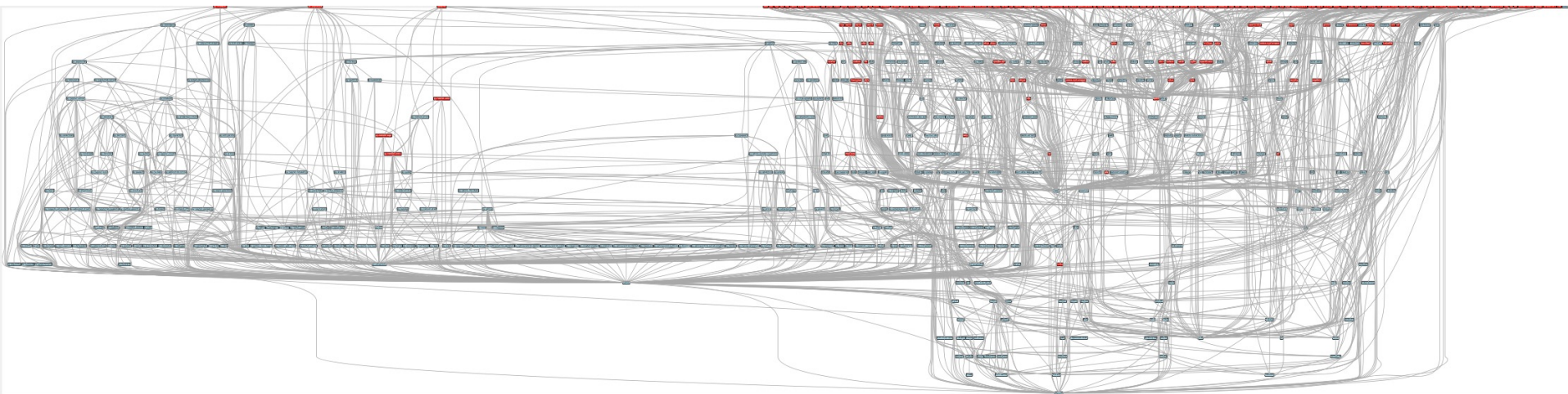


Appears to work; subtle errors later

Modern scientific codes rely on icebergs of dependency libraries



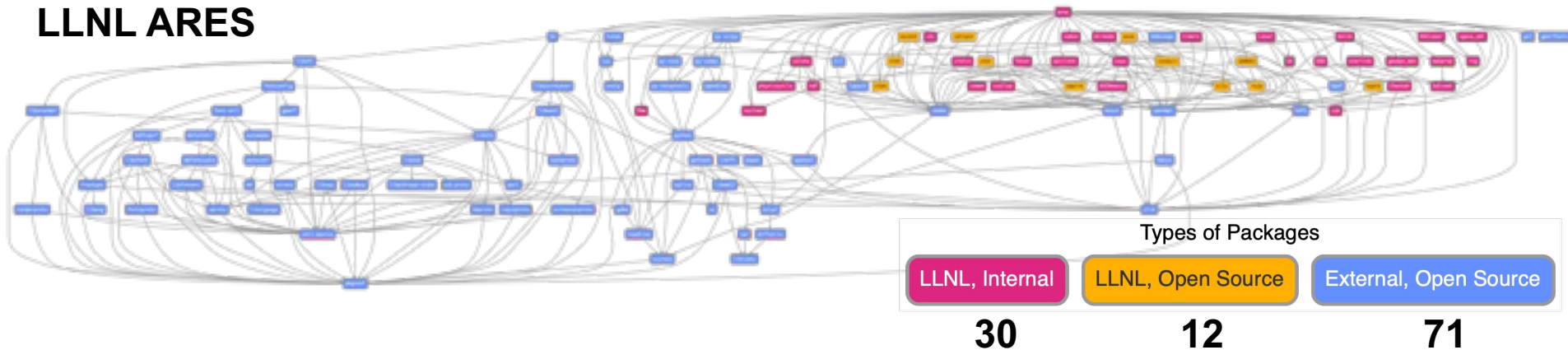
ECP's E4S stack is even larger than these codes



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

Modern software integrates open source and internal packages

LLNL ARES



- Most modern software uses *tons* of open source
- We *cannot* replace all these OSS components with our own
 - How do we put them all together effectively?
 - Do you *have* to integrate this stuff by hand?

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

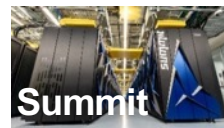
- **1:1 relationship between source code and binary (per platform)**
 - Good for reproducibility (e.g., Debian)
 - Bad for performance optimization
- **Binaries should be as portable as possible**
 - What most distributions do
 - Again, bad for performance
- **Toolchain is the same across the ecosystem**
 - 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



Oak Ridge National Lab
Power9 / NVIDIA



RIKEN
Fujitsu/ARM a64fx



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

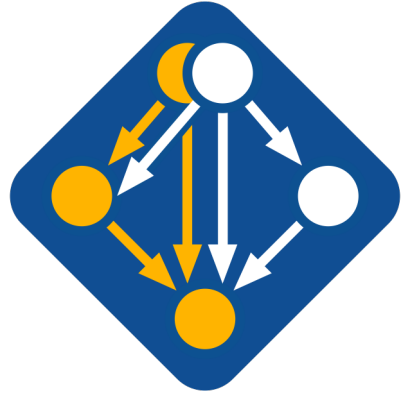
No installation required: clone and go

```
$ 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

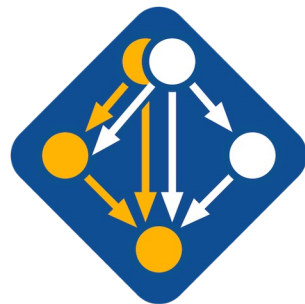
- People who want to package their own software for distribution

4. User support teams at HPC Centers

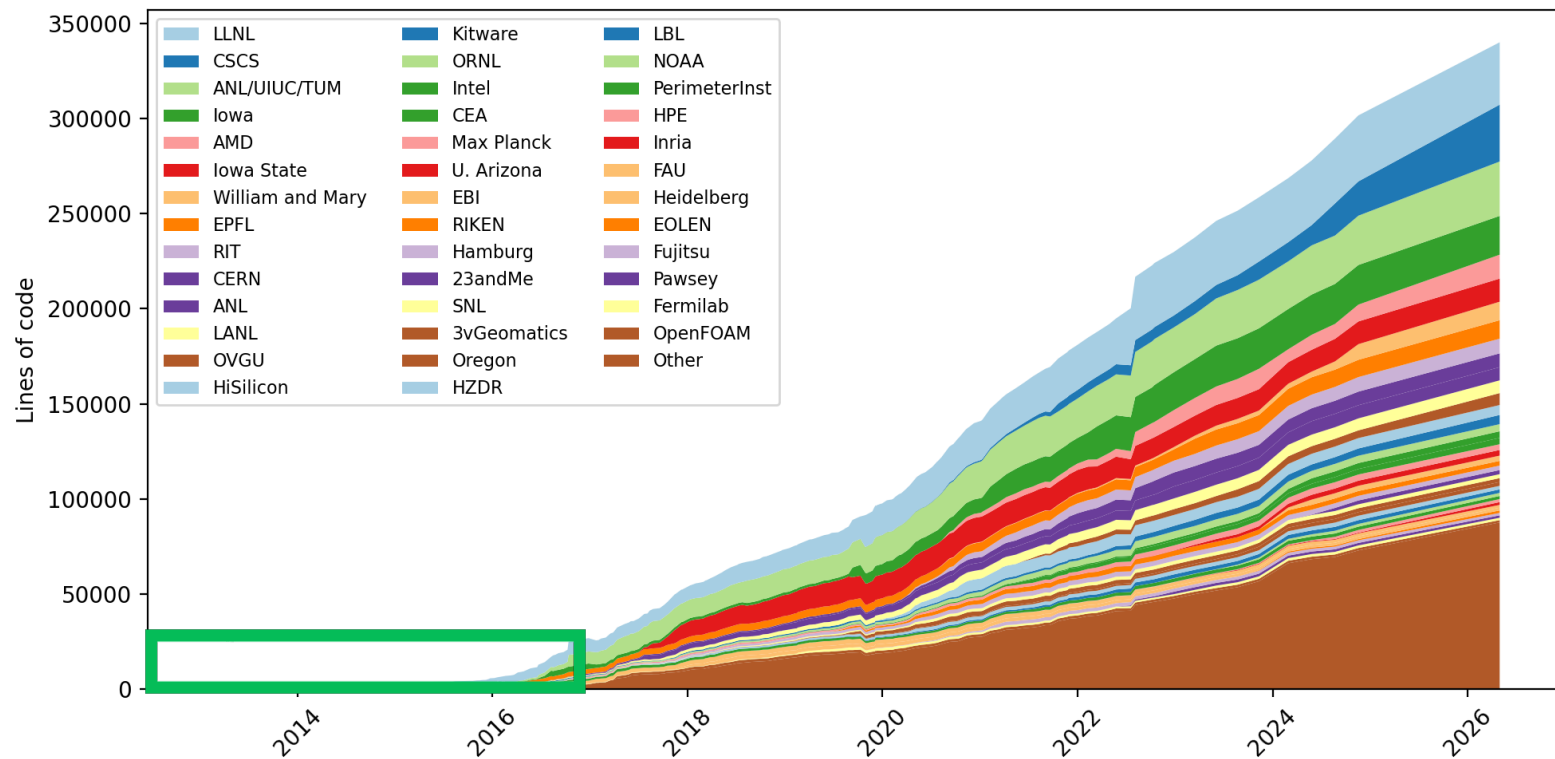
- People who deploy software for users at large HPC sites

Spack is a Linux Foundation Project!

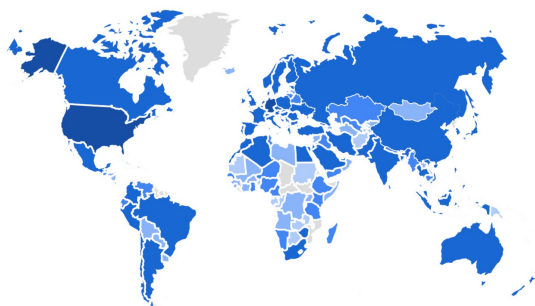
- What does that mean?
 - Project has a legal 501(c)(6) non-profit company
 - This is a neutral legal entity
 - Can be in legal agreements (e.g. for distributing binaries)
 - Can get discounts on, e.g., Slack!
 - Project will have a Technical Steering Committee (TSC)
 - Plan is to make the main developer meetings more public
 - Also have official steering committee meetings
 - Main charter is written (mostly boilerplate)
 - Working on initial GOVERNANCE.md, initial TSC members
 - Trademark (Spack name, logo) assigned to Linux Foundation
 - Project resources owned by Linux Foundation
 - spack.io website
 - GitHub Organization



Spack has grown tremendously since late 2015 from many contributors



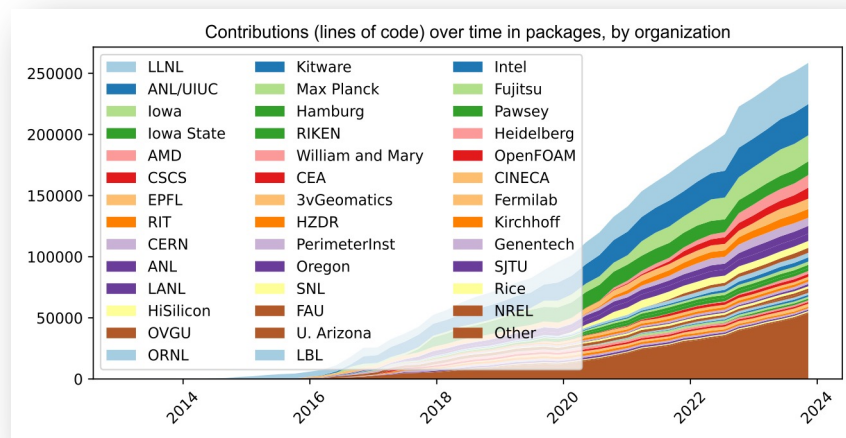
Spack sustains the HPC software ecosystem with the help of many contributors



COUNTRY	USERS
United States	23K
Germany	5.3K
China	4.6K
India	4.5K
United Kingdom	3.3K
France	3K
Japan	2.4K

2023 aggregate documentation user counts from GA4
(note: yearly user counts are almost certainly too large)

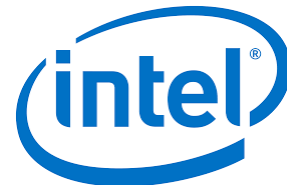
Over 8,800 software packages
Over 1,600 contributors



Contributors continue to grow worldwide!

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

- **AWS** is investing significantly in cloud credits for Spack
 - Supporting highly scalable cloud CI system with ~250k+/year in credits
 - Integrating Spack with ParallelCluster product
 - Joint Spack tutorial with AWS drew 125+ participants
- **Google** is using Spack in their HPC Toolkit cloud cluster product
 - List packages to deploy; automatically built and cached in cluster deployment
- **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 allowing us to do CI in the cloud for the Cray PE environment
 - Looking at tighter Spack integration with Cray PE
- **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



One month of Spack development is pretty busy!

May 21, 2026 – June 21, 2026

Period: 1 month ▾

Overview

138 Active pull requests

38 Active issues

87

Merged pull requests

51

Open pull requests

20

Closed issues

18

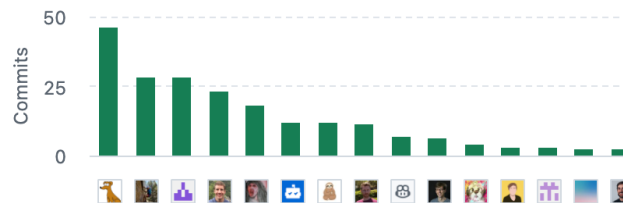
New issues

Summary

Excluding merges, **21 authors** have pushed **95 commits** to develop and **189 commits** to all branches.

On develop, **150 files** have changed and there have been **8,231 additions** and **4,275 deletions**

Top committers



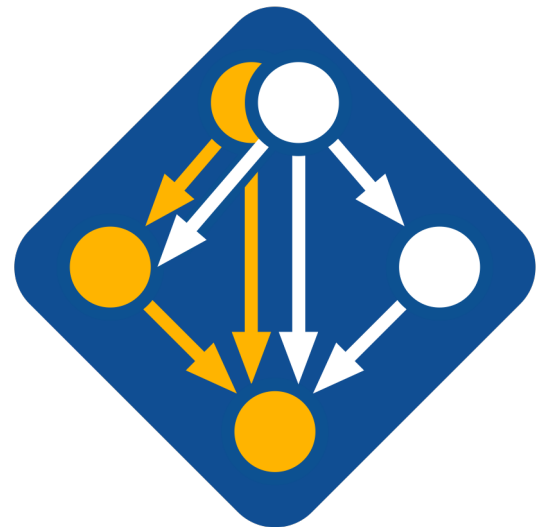
This tutorial is based on Spack v1.2.0, released early this morning!

Spack v1.2 (June 2026):

1. New installer with TUI and performance improvements
2. Spec groups in Spack environments
3. Concretization caching
4. SPDX-2.3 SBOM support
5. New `spack isolate` command to remove dependency on ~
6. [Experimental] landlock support to ensure reproducible builds
7. Much more!

Spack v1.1 (November 2025):

1. Config scopes defined as config
2. Enhanced control over compilers
3. Dependencies among external specs
4. More!



 github.com/spack/spack

Full release notes: <https://github.com/spack/spack/releases/tag/v1.2.0>

Spack is not the only HPC/AI/data science package manager out there



1. “Functional” Package Managers

- Nix
- Guix

<https://nixos.org>
<https://hpc.guix.info>



2. Build-from-source Package Managers

- Homebrew, LinuxBrew
- MacPorts
- Gentoo

<https://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

<https://easybuild.io>

▪ Conda / Mamba / Pixi

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

<https://conda.io>
<https://mamba.readthedocs.io>
<https://prefix.dev>



Claim a VM instance at: bit.ly/spack-vms



	A	B	C	D	E
1	Spack Tutorial VM Instances				
2	Instructions:	1. Put your name in a box below to claim an account on a VM instance			
3		2. Log in to your VM:			
4		ssh <IP address>			
5		Login/password are both the username from your column below			
6					
7		Login / Password			
8	IP Address	spack1	spack2	spack3	spack4
9	35.90.43.21				
10	35.91.36.120		<input type="text" value="Your Name"/>		
11	34.217.149.171				
12	35.90.450.455				

If you're in the `spack2` column, your login and password are *both* `spack2`

```
ssh spack2@3.73.129.196
```

Claim a login by putting your name in the Google Sheet

Hands-on Time: Spack Basics

Follow script at spack-tutorial.readthedocs.io



Core Spack Concepts



Spack supports 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 or direct dependency
\$ spack install mpileaks@3.3 +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	^ transitive 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.package 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

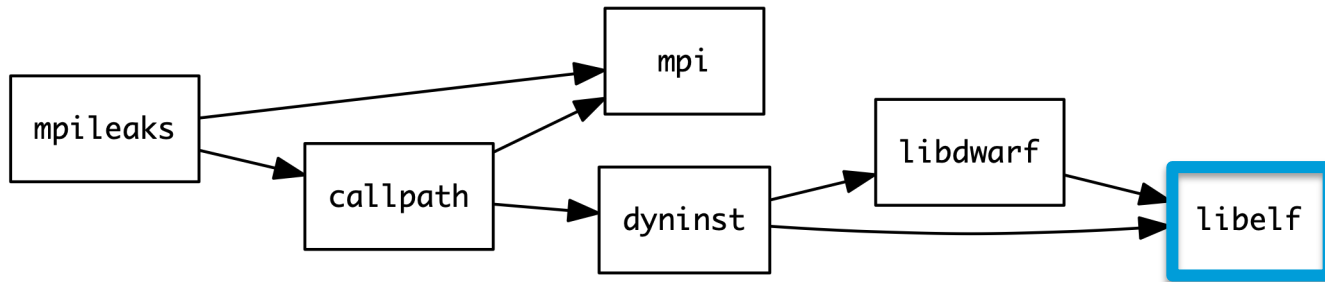
constraints on cuda version

compiler support for x86_64
and ppc64le

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



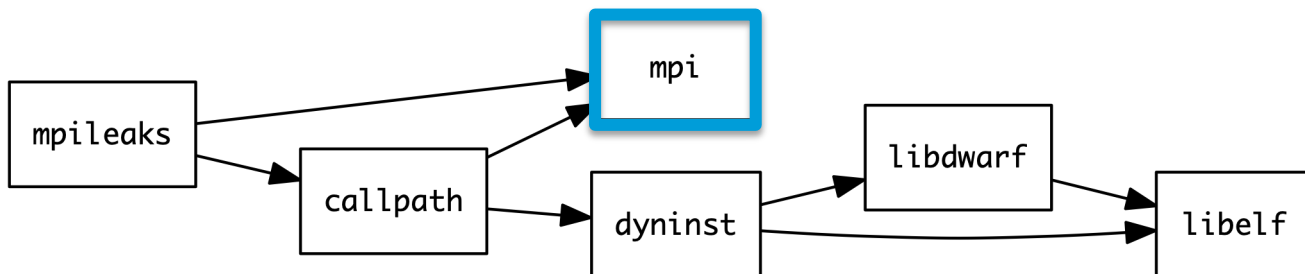
Spack Specs can constrain versions of dependencies



```
$ 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
```

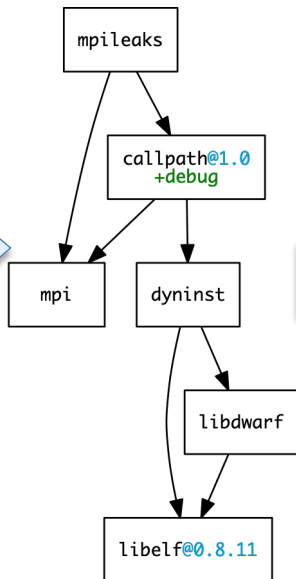
Concretization fills in missing configuration details when the user is not explicit.

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

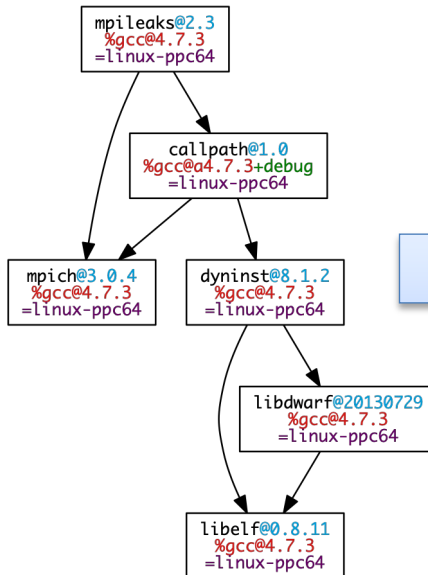
User input: *abstract* spec with some constraints

spec.yaml

Normalize



Concretize



Store

```
spec:
- mpileaks:
  arch: linux-x86_64
  compiler:
    name: gcc
    version: 4.9.2
  dependencies:
    adept-utils: ksztktpbzac3ss2ixcjkcorlaybnpt4
    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: ksztktpbzac3ss2ixcjkcorlaybnpt4
    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
...
```

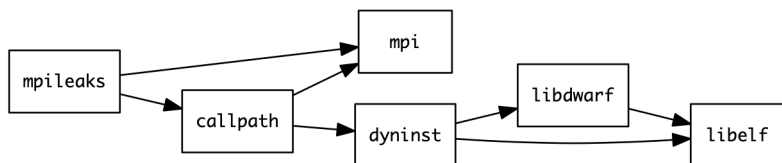
Abstract, normalized spec with some dependencies.

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

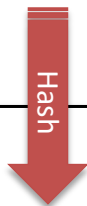
Detailed provenance is stored with the installed package

Hashing allows us to handle combinatorial complexity

Dependency DAG



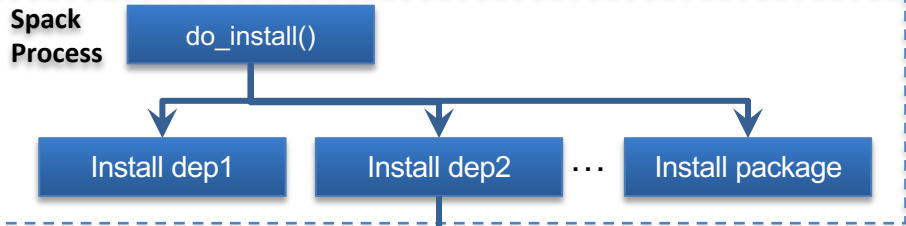
Installation Layout



```
opt
├── spack
│   ├── darwin-mojave-skylake
│   │   ├── clang-10.0.0-apple
│   │   │   ├── bzip2-1.0.8-hc4sm4vuzpm4znmvrfzri4ow2mkphe2e
│   │   │   ├── python-3.7.6-daqqpsxb6qbfrrztsezkmhus3xoflbsy
│   │   │   ├── sqlite-3.30.1-u64v26igxvxy23hysmklfums6tgjv5r
│   │   │   ├── xz-5.2.4-u5eawkvaoc7vonabe6nndkcfwuv233cj
│   │   │   └── zlib-1.2.11-x46q4wm46ay4pltrijbgizxjrhabaka6
│   ├── darwin-mojave-x86_64
│   │   ├── clang-10.0.0-apple
│   │   └── coreutils-8.29-pl2kcytejqcys5dzecfrtjxqfdssvnb
```

- 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*

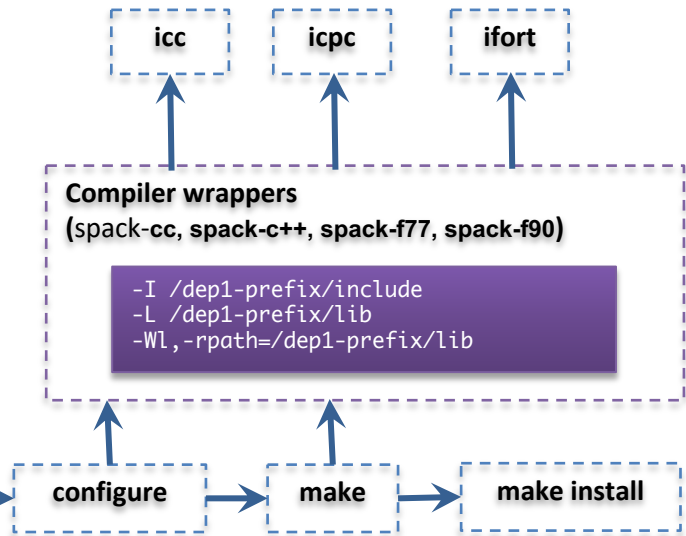
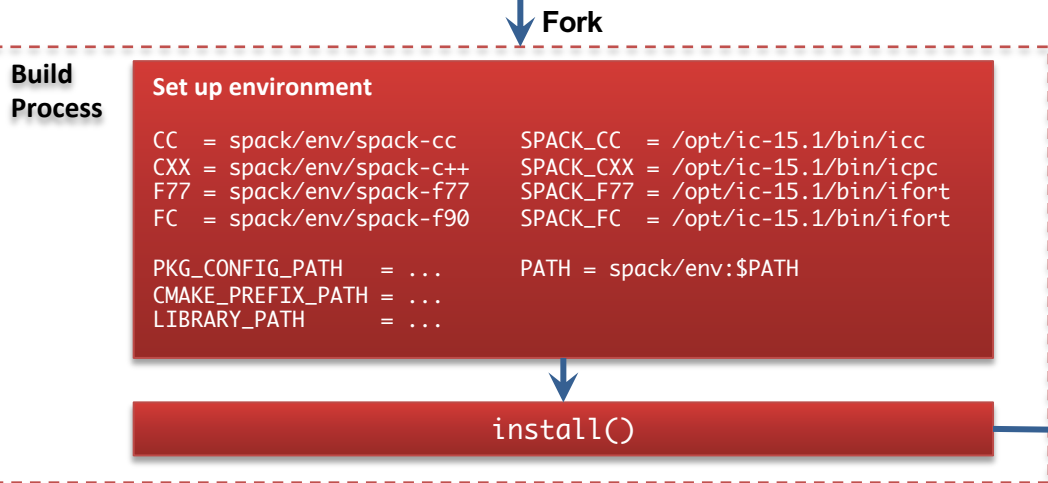
An isolated compilation environment allows Spack to easily swap compilers



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)



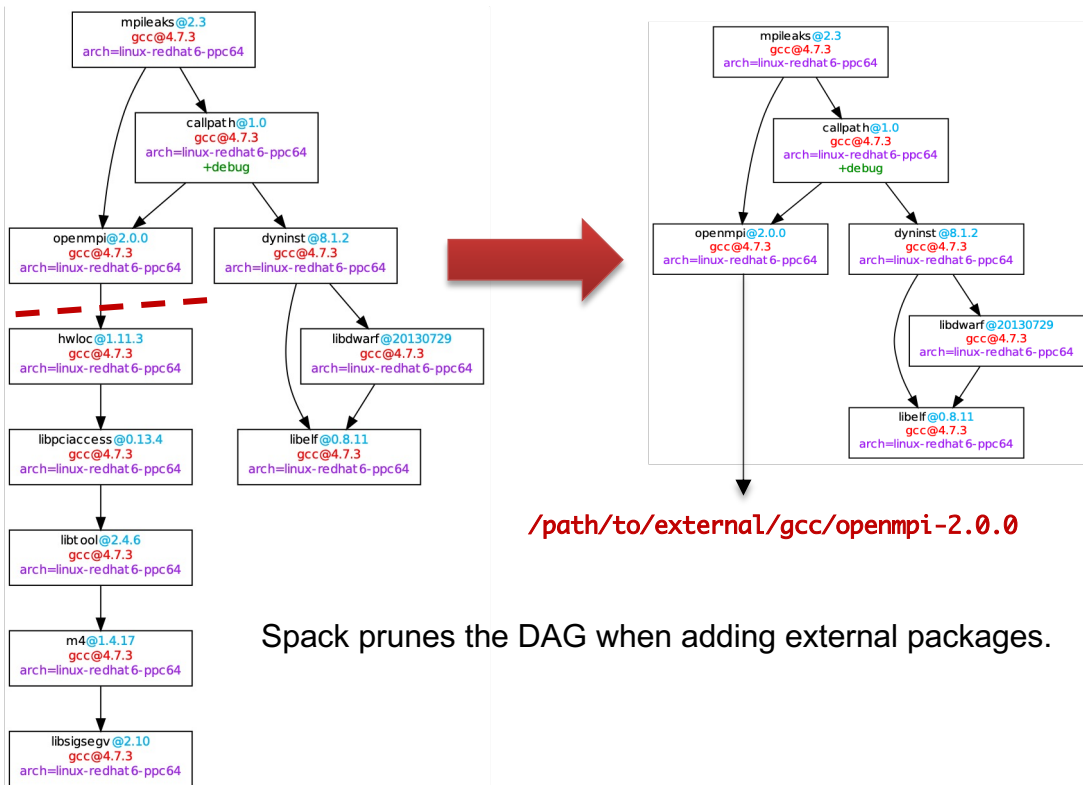
We can configure Spack to build with external software

```
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).

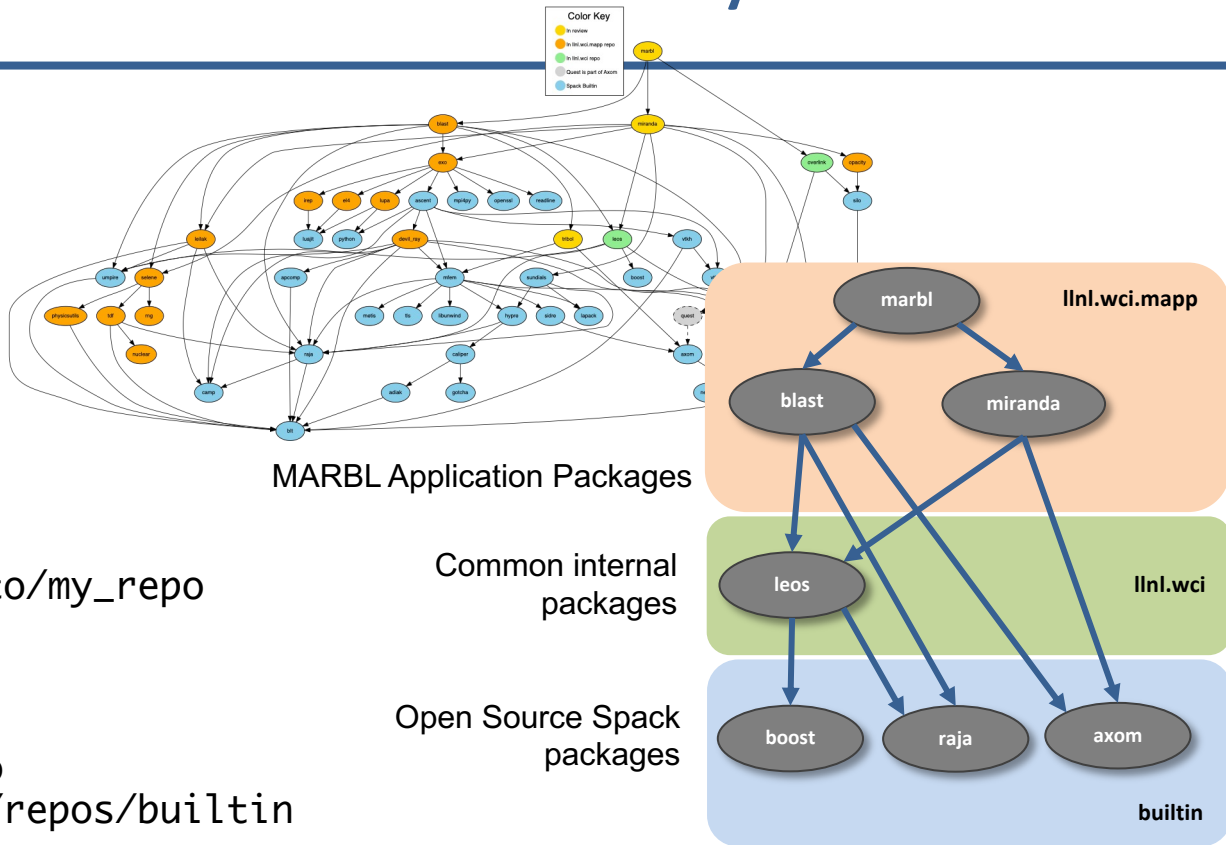


`/path/to/external/gcc/openmpi-2.0.0`

Spack prunes the DAG when adding external packages.

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
```

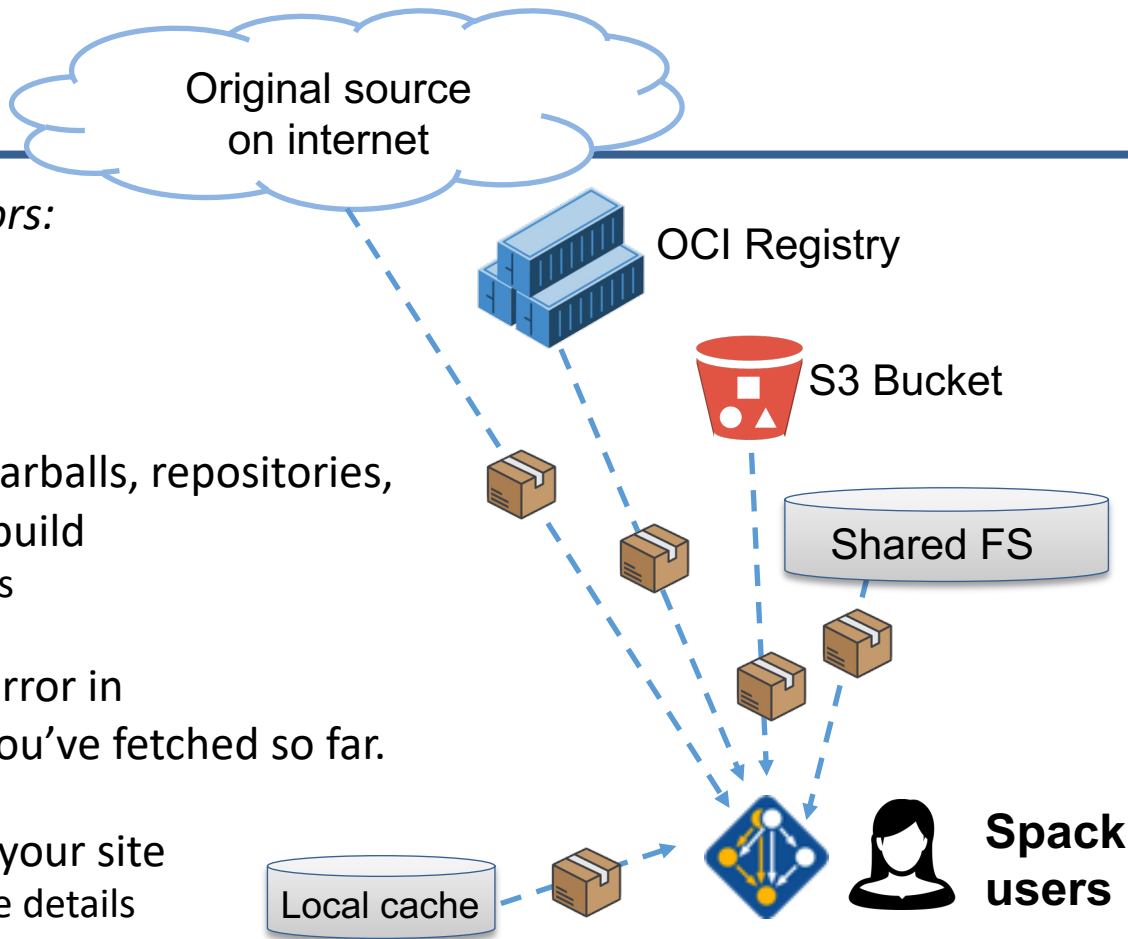
==> 2 package repositories.

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

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

Spack mirrors

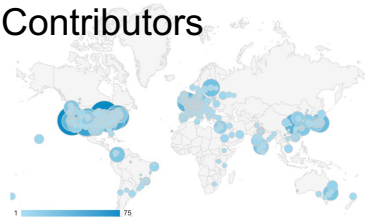
- Spack allows you to define *mirrors*:
 - Directories in the filesystem
 - On a web server
 - In an S3 bucket
- Mirrors are archives of fetched tarballs, repositories, and other resources needed to build
 - Can also contain binary packages
- By default, Spack maintains a mirror in `var/spack/cache` 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



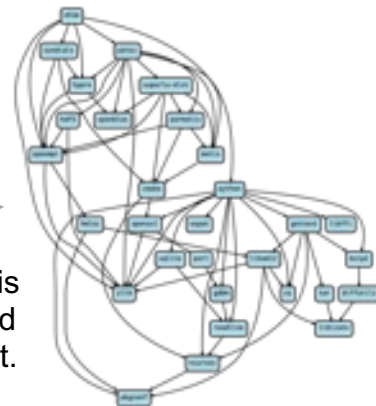
local environment config
spack.yaml

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

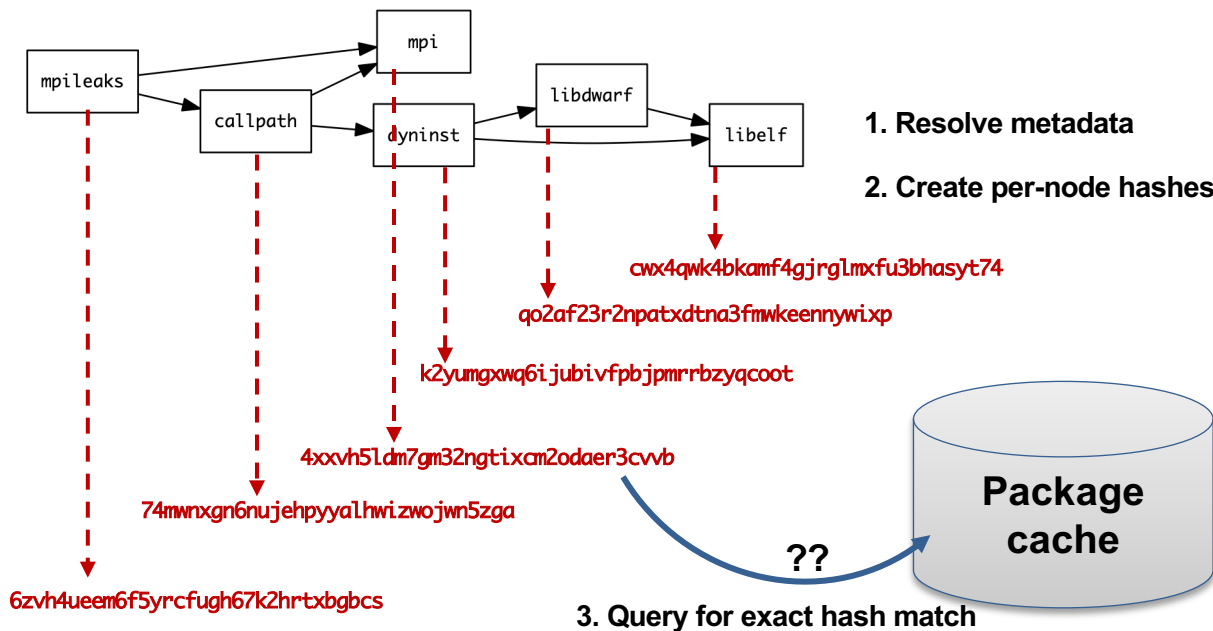
- New concretizer leverages Clingo (see potassco.org)
- Clingo is an Answer Set Programming (ASP) solver
 - ASP looks like Prolog; leverages SAT solvers for speed/correctness
 - ASP program has 2 parts:
 1. Large list of facts generated from our package repositories and config
 2. Small logic program (~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 -ll 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 ^adifutils@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
- xev1ljj ^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
- zrnuns75 ^hwloc@2.6.0%apple-clang@13.0.0-cairo-cuda-gl-libudev+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
```

Pure hash-based reuse: all misses

```
(spack)> spack solve --reuse -ll 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

```
- yfknfsp hdf5@1.10.7%apple-clang@12.0.5-cxx-fortran-hl-ipo-java-mpi+shared-szip-threadsafe+tools api=default
- z4dm26e 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 certs=system 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-gl-libudev+libxml2-netloc-nvml-opencl-pci-rocm+sho
- skdn5zf ^libxml2@2.9.12%apple-clang@12.0.5-python arch=darwin-bigsur-skylake
- 47aut3t ^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
- shng1dt ^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
```

With reuse: 16 packages were reusable



Use `spack spec` to see the results of concretization

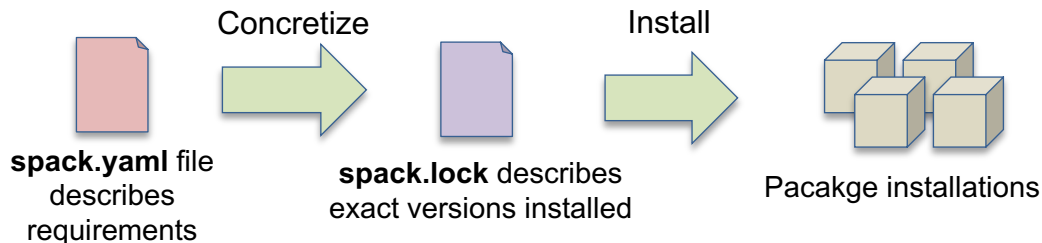
```
$ spack spec mpileaks
Input spec
```

```
-----
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": {
```

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

Follow script at spack-tutorial.readthedocs.io



We'll resume at: 11:30 CEST

Find the slides and associated scripts here:

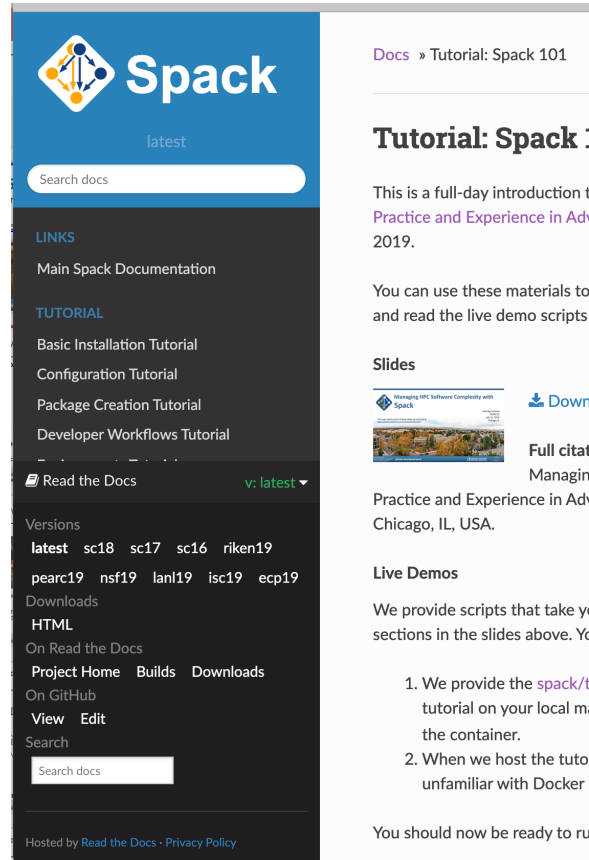
spack-tutorial.rtfd.io

Remember to join Spack slack so you can get help later!

slack.spack.io

Join the [#tutorial](#) channel!

Get a VM here →



The screenshot shows the Spack documentation website. The header features the Spack logo and the word "latest". Below the header is a search bar labeled "Search docs". The main content area is divided into sections: "LINKS" with links to "Main Spack Documentation", "TUTORIAL" with links to "Basic Installation Tutorial", "Configuration Tutorial", "Package Creation Tutorial", and "Developer Workflows Tutorial", and "Versions" with a list of version tags: "latest", "sc18", "sc17", "sc16", "riken19", "pearc19", "nsf19", "lan19", "isc19", and "ecp19". There are also links for "Downloads", "HTML", "On Read the Docs", "Project Home", "Builds", "Downloads", "On GitHub", "View", "Edit", and "Search". The footer includes "Hosted by Read the Docs · Privacy Policy".


Docs » Tutorial: Spack 101

Tutorial: Spack 101

This is a full-day introduction to Spack, covering the topics of [Practice and Experience in Advanced HPC Computing](#) 2019.

You can use these materials to prepare for the tutorial, and read the live demo scripts.

Slides

 [Download](#)

Full citation: Managing HPC Software Complexity with Spack. Practice and Experience in Advanced HPC Computing, Chicago, IL, USA.

Live Demos

We provide scripts that take you through the sections in the slides above. You can find them in the [Scripts](#) section.

1. We provide the [spack/tutorial](#) tutorial on your local machine, so you can run the container.
2. When we host the tutorial, we provide a VM that is unfamiliar with Docker.

You should now be ready to run the tutorial.



But wait! There's more!



HPSF



Spack is part of the
High Performance Software Foundation

Join us at the Spack User Meeting at
HPSFCon 2027 next year!

hpsf.io

Join us after ISC!

- Join us and 3,700+ others on Spack slack
- Contribute packages, docs, and features on GitHub
- Continue the tutorial at spack-tutorial.rtdf.io



slack.spack.io



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



[@spackpm.bsky.social](https://bsky.app/profile/spackpm.bsky.social)



[@spack@hpc.social](https://mstdn.social/@spack)



[@spackpm](https://x.com/spackpm)

We hope to make distributing & using HPC software easy!



HPSF

HIGH PERFORMANCE
SOFTWARE FOUNDATION

LLNL-PRES-806064

Join #tutorial on Slack: spackpm.herokuapp.com

Materials: spack-tutorial.readthedocs.io



Hands-on Time: Creating Packages

Follow script at spack-tutorial.readthedocs.io



Hands-on Time: Stacks

Follow script at spack-tutorial.readthedocs.io



Hands-on Time: Binary Caches and Mirrors

Follow script at spack-tutorial.readthedocs.io

Hands-on Time: Developer Workflows

Follow script at spack-tutorial.readthedocs.io