



Managing HPC Software Complexity with Spack

PEARC26 Tutorial

July 27, 2025

Alec Scott, Kathleen Shea, and Caetano Melone

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



Alec Scott
LLNL



Caetano Melone
LLNL



Kathleen Shea
LLNL

Tutorial Materials

Find these slides and associated scripts here:

<https://spack-tutorial.readthedocs.io>

Join us on Slack

- <https://slack.spack.io>
- Join the **#tutorial** channel!
- You can ask questions here after the conference is over.
- **Over 3,800** people can help you on Slack!



Spack

latest

Search

LINKS

[Main Spack Documentation](#)

TUTORIAL

[Basic Installation Tutorial](#)

[Environments Tutorial](#)

[Configuration Tutorial](#)

[Package Creation Tutorial](#)

[Stacks Tutorial](#)

[Developer Workflows Tutorial](#)

[Binary Caches Tutorial](#)

[Scripting with Spack](#)

ADDITIONAL SECTIONS

[Module Files Tutorial](#)

[Spack Package Build Systems](#)

[Advanced Topics in Packaging](#)

Tutorial: Spack 101

This is an introduction to Spack with lectures and live demos. It was last presented at [and Experience in Advanced Research Computing \(PEARC\) 25 Conference](#) and was a full-day tutorial.

You can use these materials to teach a course on Spack at your own site and read the live demo scripts to see how Spack is used in practice.

Slides



[Download Slides](#)

Full citation: Todd Gamblin, Alec Scott, Kathleen Rasmussen, et al. Managing HPC Software Complexity with Spack. In Proceedings of the Advanced Research Computing 2025 (PEARC25) Conference, 2025.

Video

For the last recorded video of this tutorial, see the [HPCIC Tutorial 2024](#).

Live Demos

We provide scripts that take you step-by-step through basic Spack task sections in the slides above.

To run through the scripts, we provide the [spack/tutorial](#) container image.

```
$ docker pull ghcr.io/spack/tutorial:pearc25
$ docker run -it ghcr.io/spack/tutorial:pearc25
```

to start using the container. You should now be ready to run through our

Claim a VM Instance

spack.io/tutorial-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		Your Name		
11	34.217.149.171				
12	35.90.450.155				

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)

Morning

Intro	9:00 am
Basics	
Concepts	
Break	10:30 am
Environments	10:35 am
Configuration	
Lunch	12:30 pm

Afternoon

Software Stacks	1:30 pm
Packaging	
Break	3:00 pm
Developer Workflows	3:30 pm
Mirrors & Binary Caches	
Scripting	
End	5:00 pm

Modern scientific codes are built 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



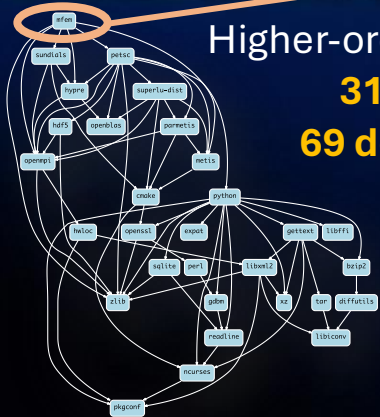
Appears to work; subtle errors later

Modern scientific codes rely on icebergs of dependency libraries

MFEM:

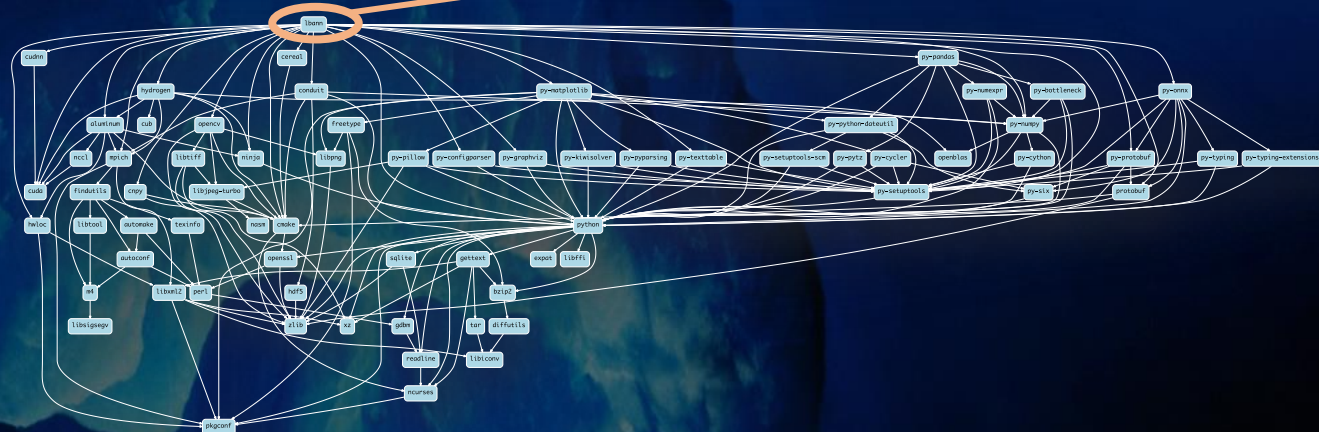
Higher-order finite elements

**31 packages,
69 dependencies**



LBANN: Neural Nets for HPC

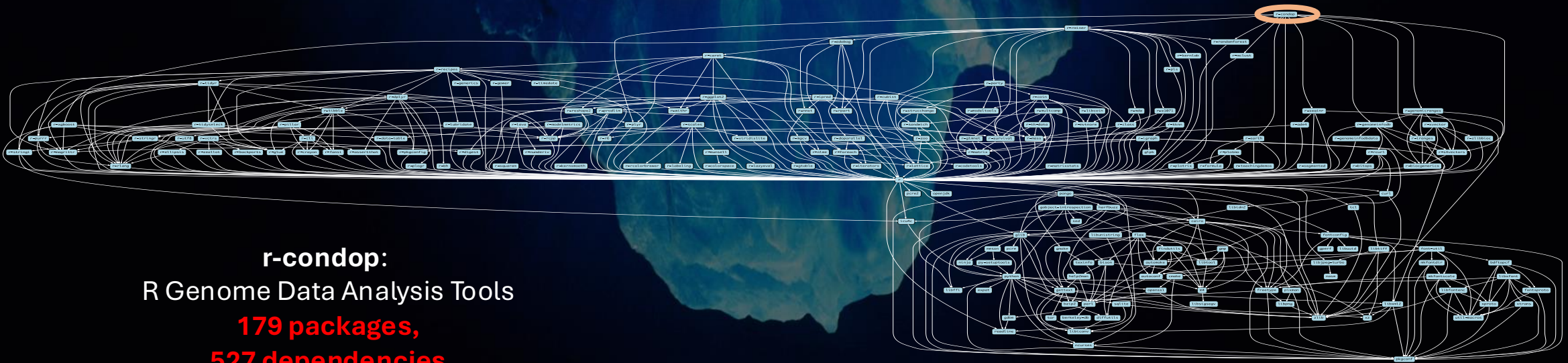
71 packages
188 dependencies



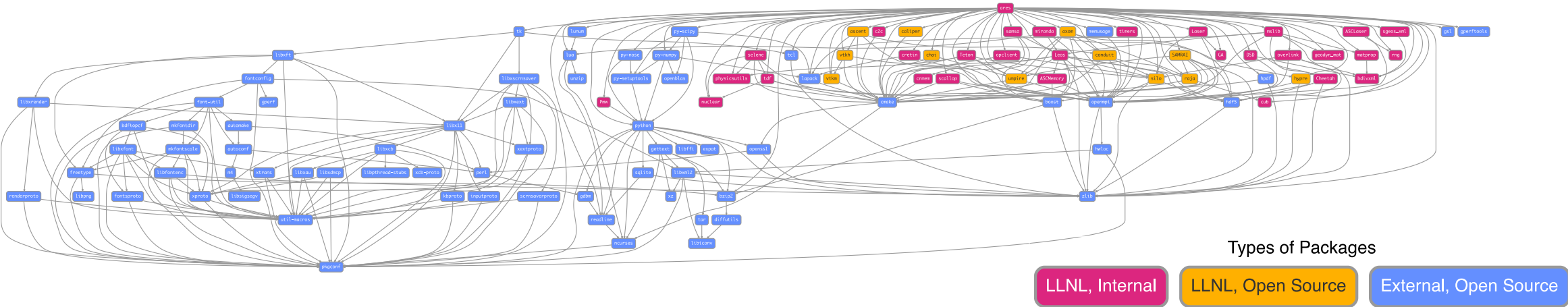
r-condop:

R Genome Data Analysis Tools

**179 packages,
527 dependencies**



Modern software integrates open source and internal packages



- 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

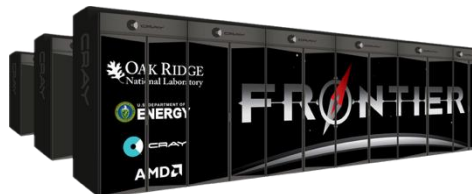
- **1:1 relationship between source code and binary (per platform)**
 - Good for reproducibility (e.g., Debian)
 - Bad for performance optimization
- **Binaries should be as portable as possible**
 - What most distributions do
 - Again, bad for performance
- **Toolchain is the same across the ecosystem**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

High Performance Computing (HPC) violates many of these assumptions

- **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
- **Code is typically distributed as source**
 - With exception of vendor libraries, compilers
- **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



**Lawrence Livermore
National Lab**
AMD **Zen** / **MI300A**



Oak Ridge National Lab
AMD **Zen** / **MI250X**



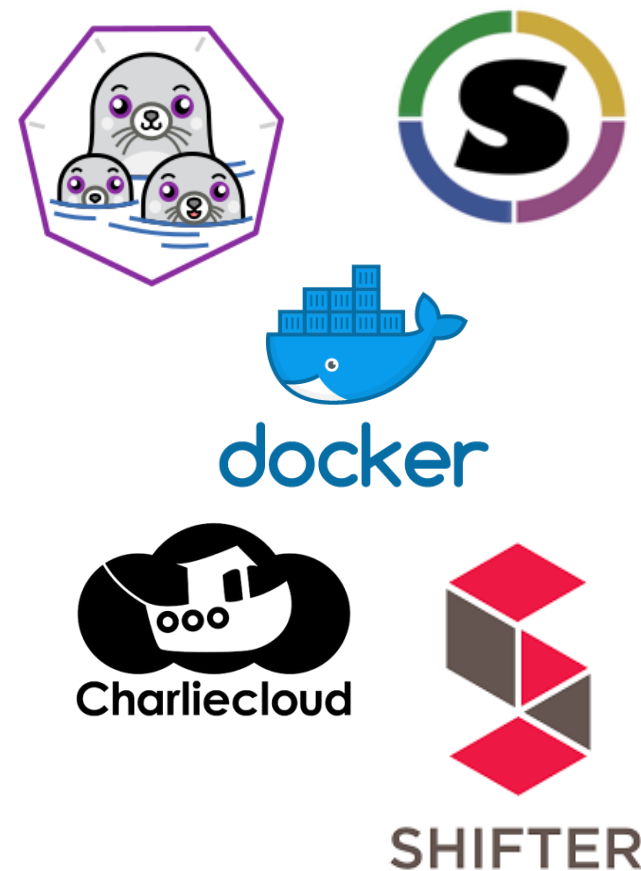
RIKEN
Fujitsu **ARM a64fx**



Argonne National Lab
Intel **Xeon** / **Xe**

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



Spack enables Software distribution for HPC



No installation required: clone and go

```
$ git clone --depth=2 https://github.com/spack/spack
$ . spack/share/spack/setup-env.sh
$ spack install hdf5
```

Simple syntax enables complex installs

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

- Packages are ***parameterized***, so that users can easily tweak and tune configuration
- Ease of use of mainstream tools, with flexibility needed for HPC

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 part of the High Performance Software Foundation (HPSF)

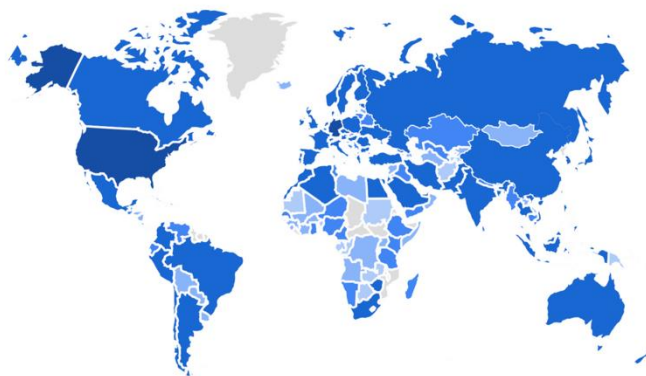
- **Project has a neutral legal entity**
 - 501(c)(6) non-profit company
- **Project has a Technical Steering Committee (TSC)**
 - Charter mandates TSC to make decisions
 - Governance defined at github.com/spack/governance
- Trademark (Spack name, logo) assigned to Linux Foundation
- Project resources owned by Linux Foundation
 - spack.io website
 - GitHub Organization



HPSF
HIGH PERFORMANCE
SOFTWARE FOUNDATION



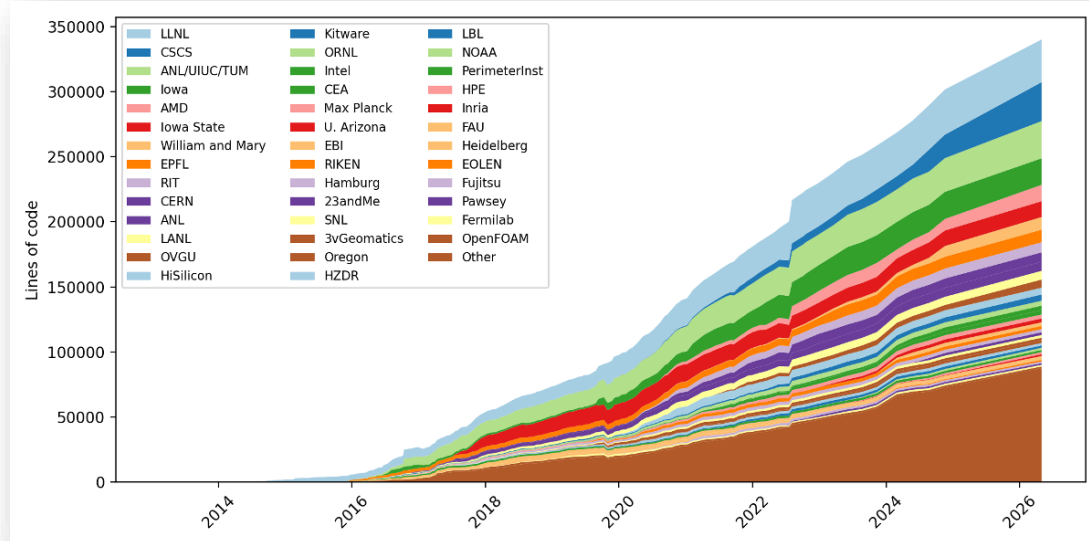
Spack sustains the HPC software ecosystem with the help of many contributors



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 8,900 software packages
Over 1,600 contributors



Contributors continue to grow worldwide!

One month of Spack development is pretty busy!

October 16, 2025 – November 16, 2025

Period: 1 month ▾

Overview



504 Active pull requests



23 Active issues



361

Merged pull requests



143

Open pull requests



8

Closed issues



15

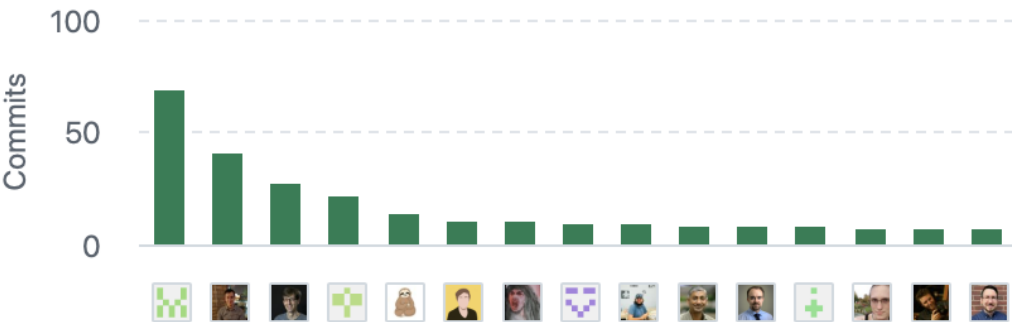
New issues

Summary

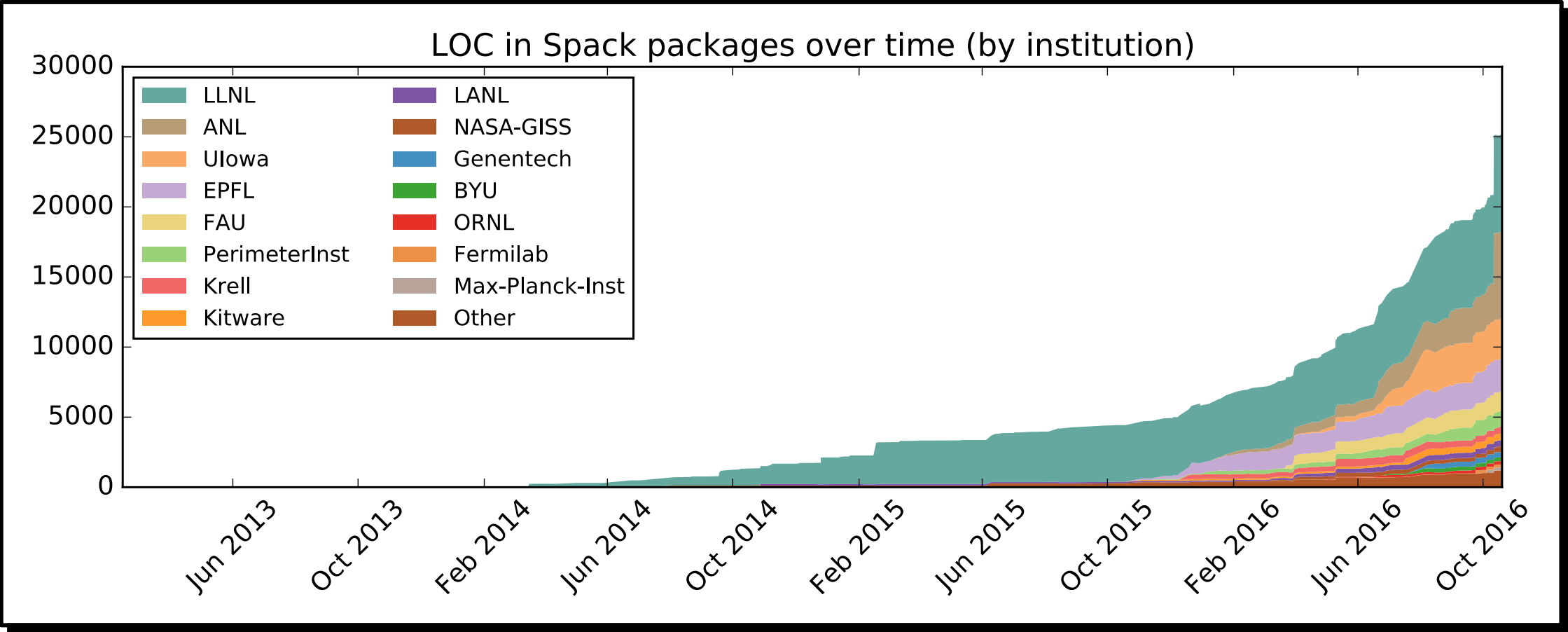
Excluding merges, **116 authors** have pushed **361 commits** to develop and **402 commits** to all branches.

On develop, **848 files** have changed and there have been **7,676 additions** and **1,868 deletions**

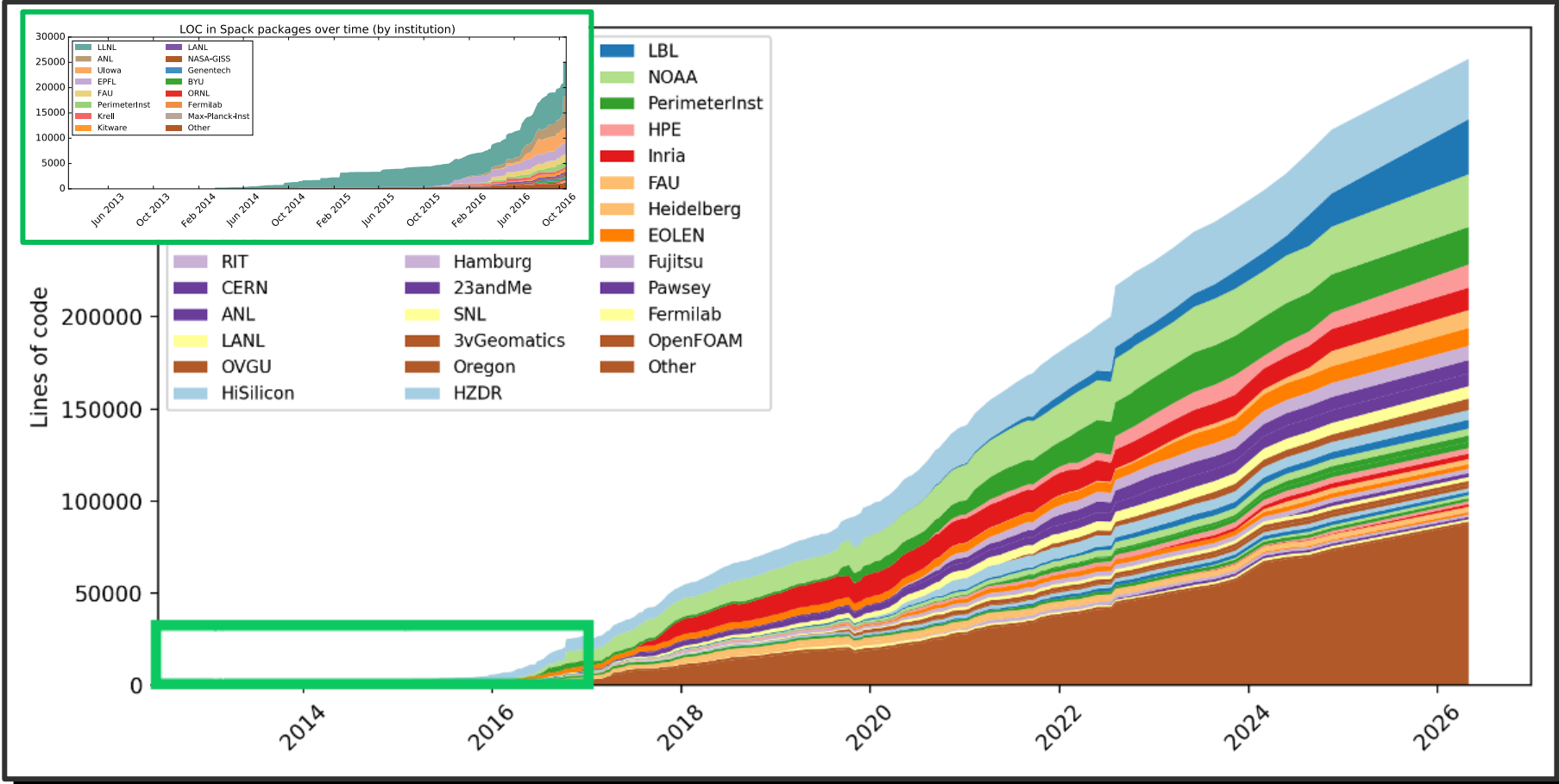
Top Committers



2016

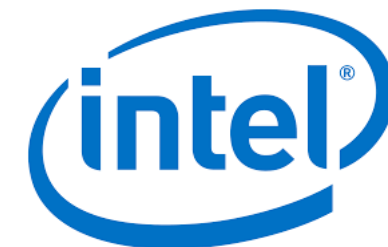


2026



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



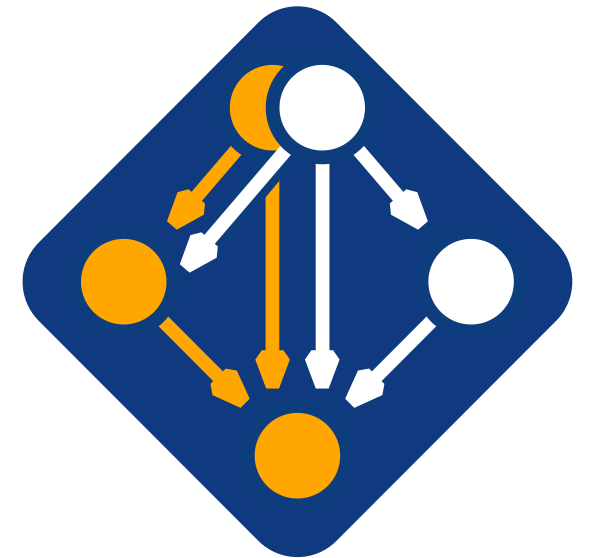
Spack v1.2.0 was released on June 21st!

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. Nix, Guix: “Functional” Package Managers

- Pioneered using hashes to identify installations/configurations
- No concretizer/solver; handling multiple versions requires coding packages

<https://hpc.guix.info>



2. Build-from-source Package Managers

- Homebrew, LinuxBrew
- MacPorts
- Gentoo

<https://brew.sh>

<https://www.macports.org>

<https://hpc.guix.info>



3. Pip, UV

- Package managers for PyPI, Python ecosystem
- **Very** widely used, increasingly for (unoptimized) software binaries beyond Python

<https://pypi.org/project/pip/>

<https://docs.astral.sh/uv/>



4. Easybuild

- Installation tool Focused on HPC system administrators
- Different model w/fixed software stack – harder to tweak recipes for experimentation

<https://easybuild.io>



5. 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

spack.io/tutorial-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		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



Hands on Section

Basics



Presentation

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 / Podman)
 - 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@8.7.3	% custom build 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 architecture
\$ spack install mpileaks@3.3 ^mpich@3.2 %gcc@8.7.3	^ dependency constraints

- Each expression is a **spec** for a particular configuration
 - Each clause adds a constraint to the spec.
 - Constraints are optional – specify only what you need.
 - Customize install on the command line!
- Spec syntax is recursive
 - Full control over the combinatorial build space



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://github.com/llnl/kripke"
    url      = "https://github.com/llnl/Kripke/archive/refs/tags/v1.2.6.tar.gz"

    version("1.2.7", sha256="3f7f2eef0d1ba5825780d626741eb0b3f026a096048d7ec4794d2a7dfbe2b8a6")
    version("1.2.6", 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 [
            self.define_from_variant("ENABLE_OPENMP", "openmp"),
            self.define_from_variant("ENABLE_MPI", "mpi"),
        ]

    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.

Conditional variants simplify packages

```
from spack.package import *

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

    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)

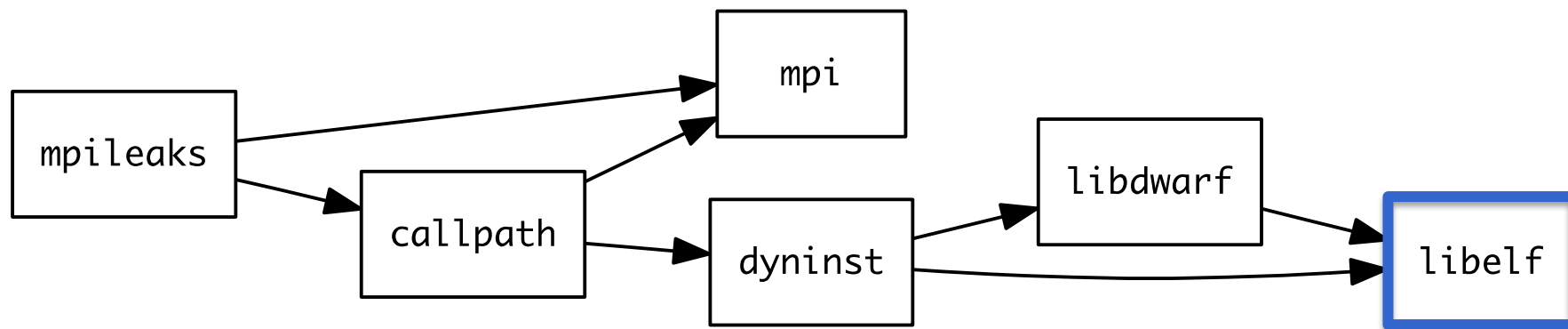
cuda_arch is only
present if cuda is
enabled

dependency on cuda,
but only if cuda is
enabled

compiler support for
x86_64 and ppc64le

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

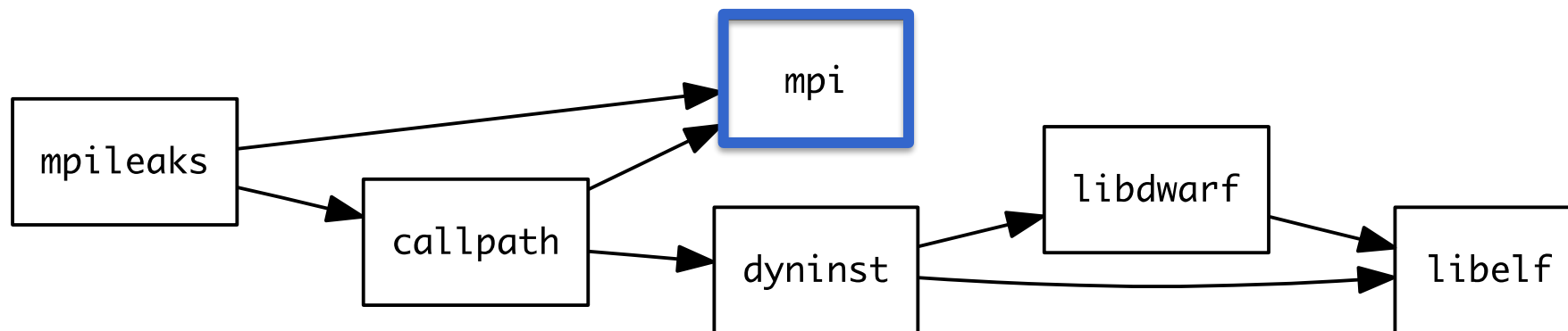
Specs can constrain versions of dependencies



```
$ 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.

Specs can constrain versions of dependencies



- `mpi` is a **virtual** dependency
- Install the same package build 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:

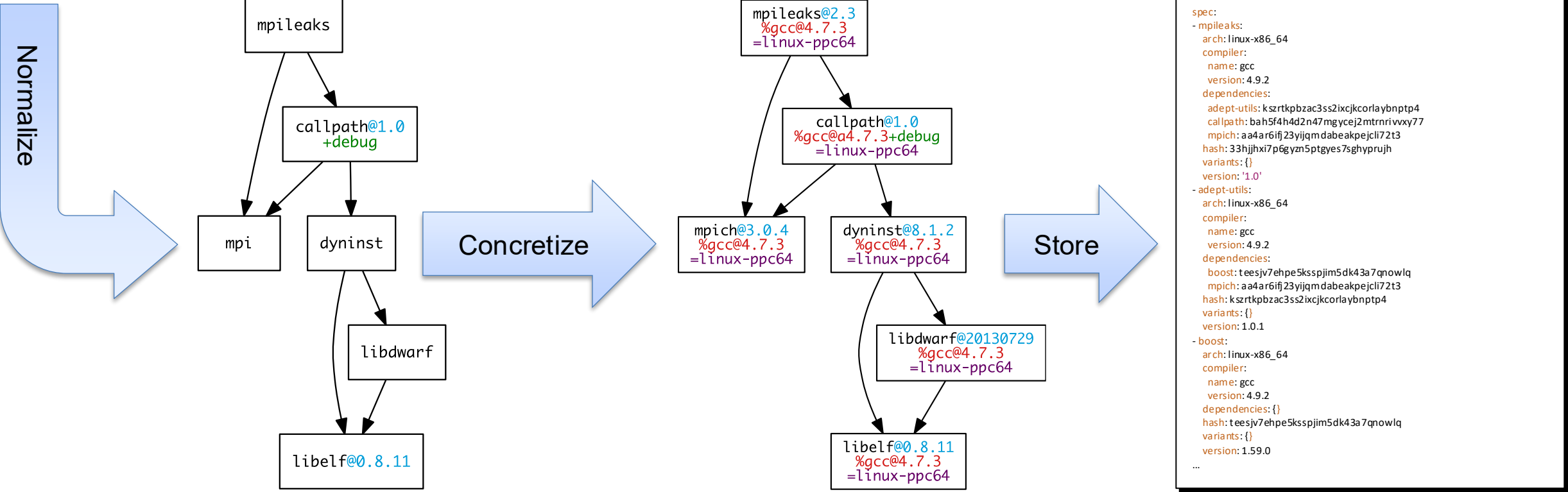
```
$ spack install mpileaks ^mpi@2
```

Concretize fills in missing configuration details

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

User input: *abstract* spec with some constraints

spec.yaml



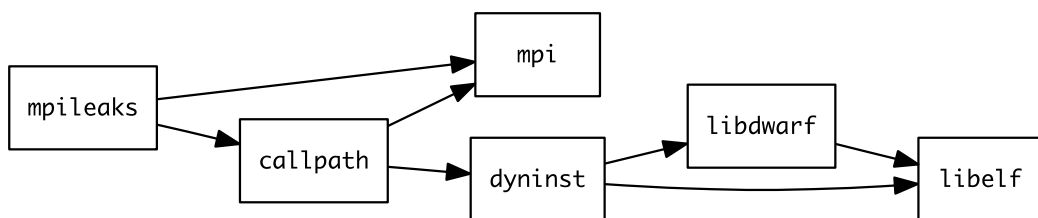
Abstract, normalized spec with some dependencies.

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

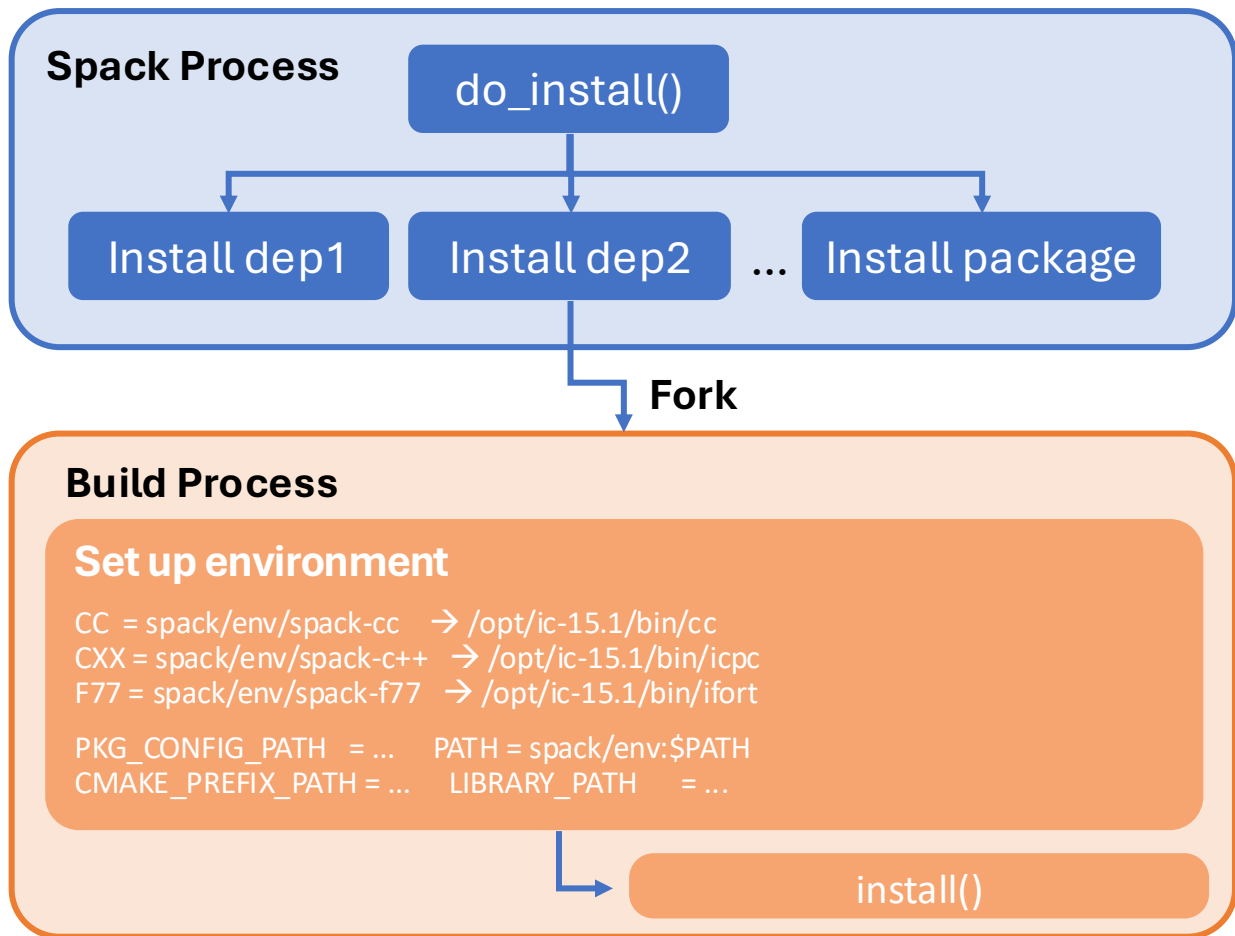
```

opt
├── spack
│   ├── darwin-mojave-skylake
│   │   ├── clang-10.0.0-apple
│   │   │   ├── bzip2-1.0.8-hc4sm4vuzpm4znmvrfzri4ow2mkphe2e
│   │   │   ├── python-3.7.6-daqqpssxb6qbfztsezkmhus3xoflbsy
│   │   │   └── zlib-1.2.11-x46q4wm46ay4pltrijbgizxjrhbaka6
│   ├── darwin-mojave-x86_64
│   │   ├── clang-10.0.0-apple
│   │   └── coreutils-8.29-pl2kcytejqcys5dzecfrtjqxfsdssvnob
  
```

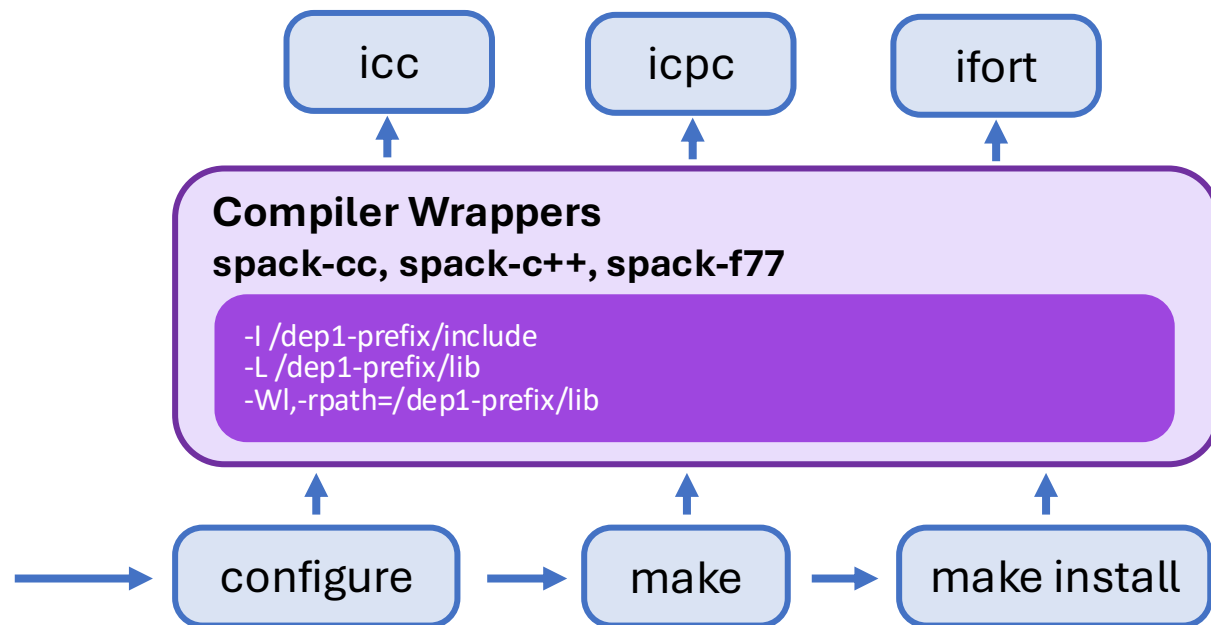


- Each unique dependency graph is a unique **configuration**.
- Each configuration is a unique directory.
 - Multiple configurations of the same package can coexist.
- **Hash** of entire directive acyclic graph (DAG) is appended to each prefix.
- Installed packages automatically find dependencies
 - 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)



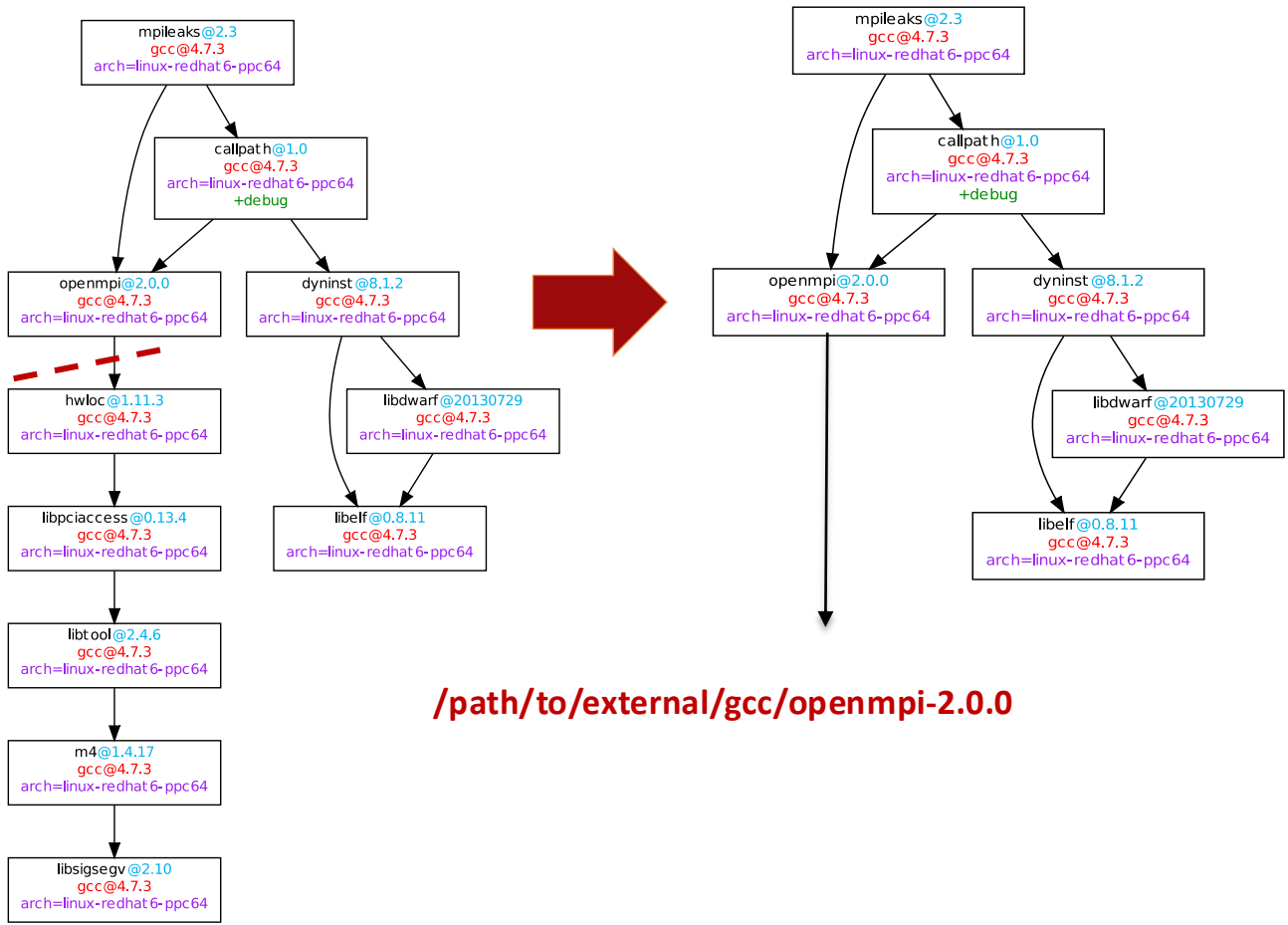
Spack can build with external software

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

packages.yaml

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

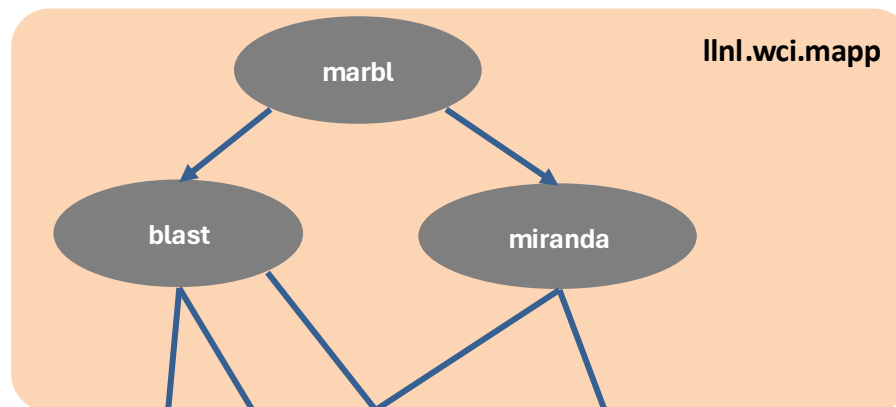
Users register external packages in a configuration file (more on these later).



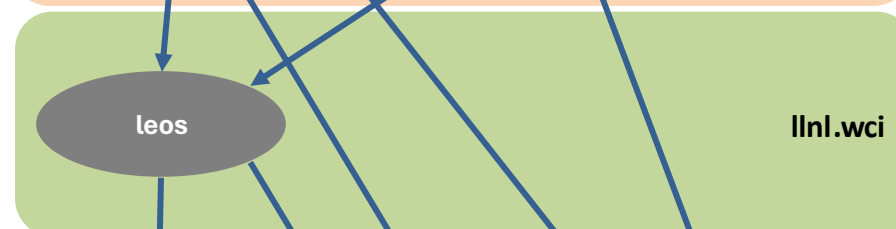
Spack prunes the DAG when adding external packages.

Package repositories allow stacks to be layered

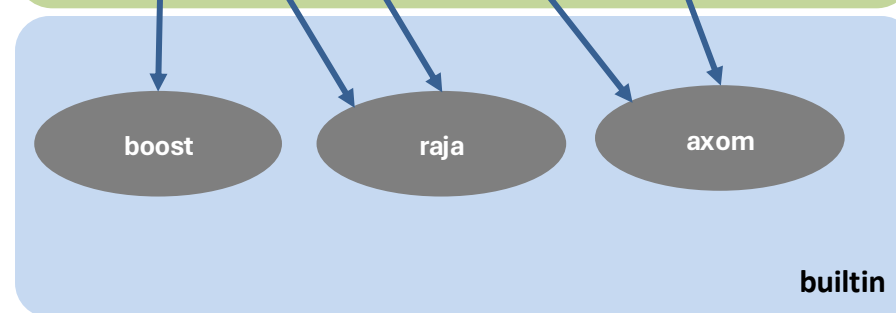
MARBL Application Packages



Common internal packages



Open Source Spack Packages



```
$ spack repo create /path
```

```
$ spack repo add my_repo
```

```
$ spack repo list
```

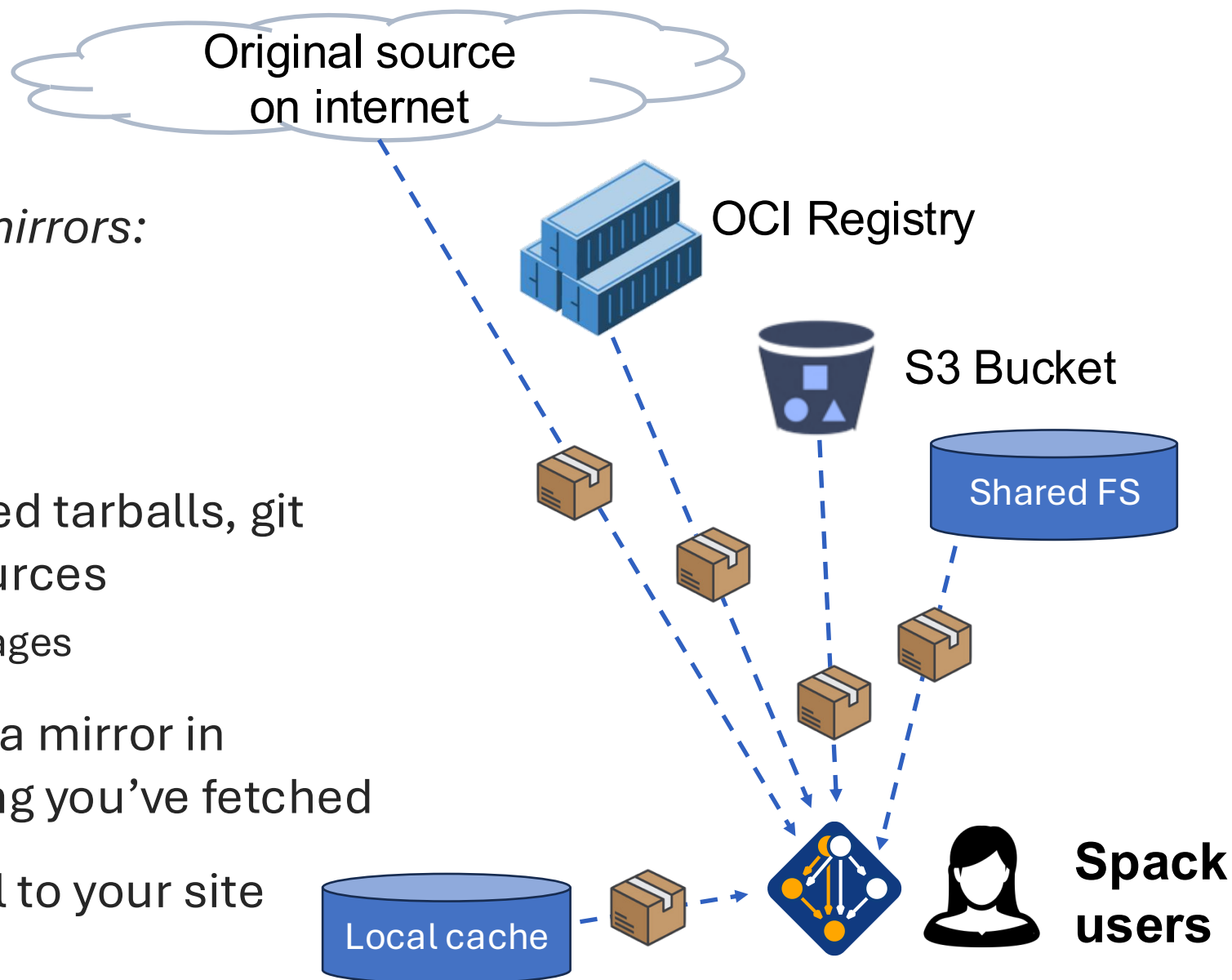
```
=> 2 package repositories
```

```
my_repo  /path
```

```
builtin  spack-packages
```

Spack mirrors

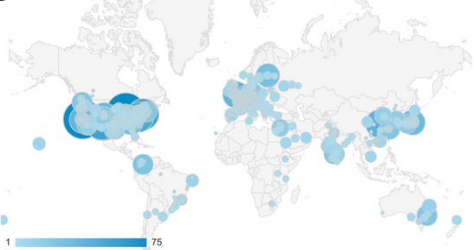
- Spack allows you to define *mirrors*:
 - Directories in the filesystem
 - On a web server
 - In an S3 bucket
- Mirrors are archives of fetched tarballs, git repositories, and other resources
 - Can also contain binary packages
- By default, Spack maintains a mirror in `var/spack/cache` of everything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



The concretizer includes information from packages, configuration, and CLI

Dependency solving is NP-hard

Contributors



- new versions
- new dependencies
- new constraints



spack developers



default config
packages.yaml

admins,
users



local preferences config
packages.yaml

users



local environment config
spack.yaml

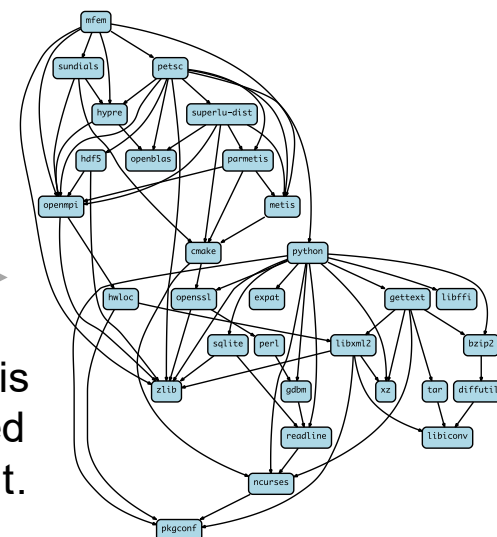
users

Command line constraints

spack install hdf5@1.12.0 +debug



Concrete spec is fully constrained and can be built.



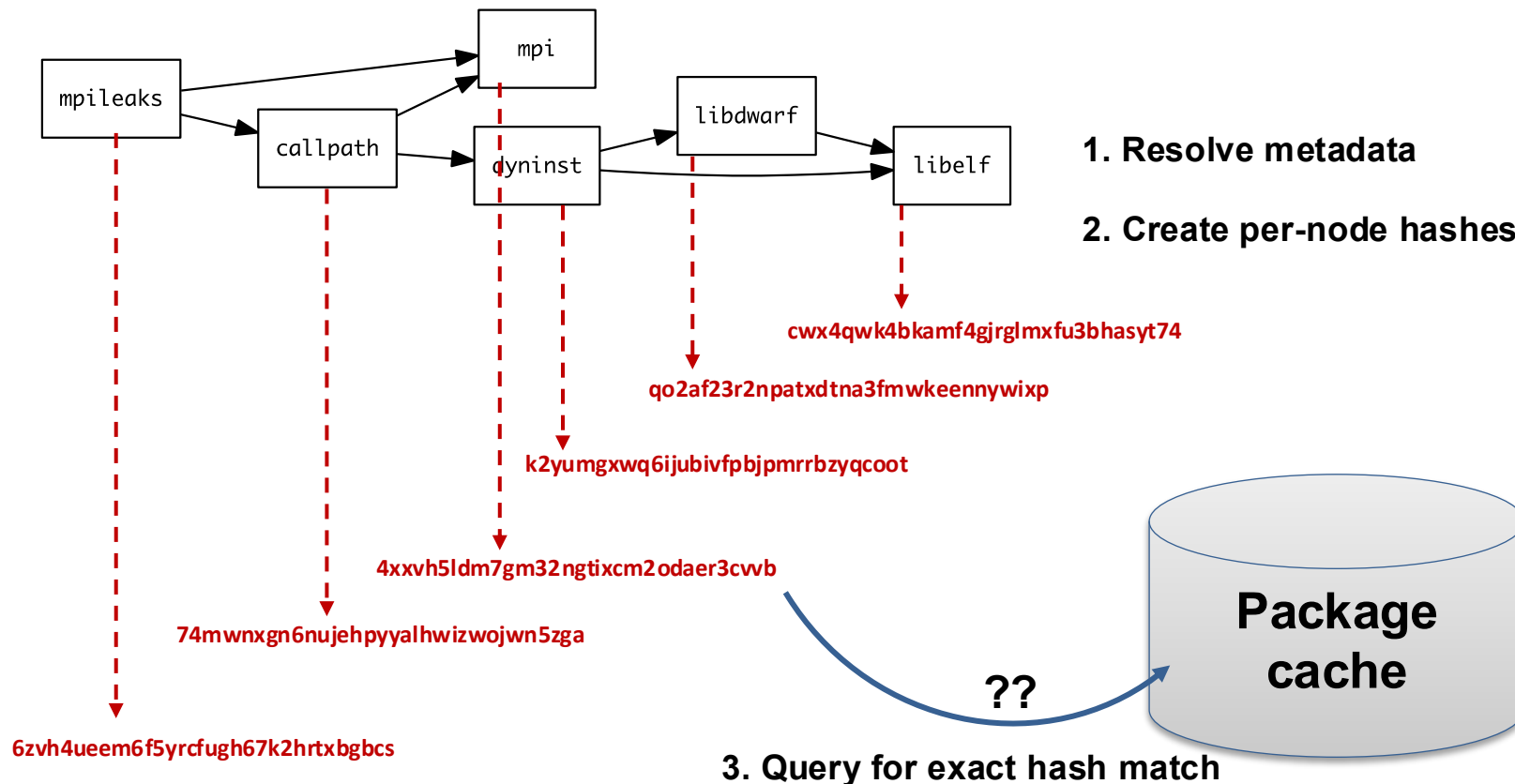
We use logic programming to simplify package solving

- 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

```
(spackle):solver> spack solve -Il hdf5
=> Best of 9 considered solutions.
=> Optimization Criteria:
```

Priority	Criterion	Installed	ToBuild
1	number of packages to build (vs. reuse)	-	20
2	deprecated versions used	0	0
3	version weight	0	0
4	number of non-default variants (roots)	0	0
5	preferred providers for roots	0	0
6	default values of variants not being used (roots)	0	0
7	number of non-default variants (non-roots)	0	0
8	preferred providers (non-roots)	0	0
9	compiler mismatches	0	0
10	OS mismatches	0	0
11	non-preferred OS's	0	0
12	version badness	0	2
13	default values of variants not being used (non-roots)	0	0
14	non-preferred compilers	0	0
15	target mismatches	0	0
16	non-preferred targets	0	0

```
hdf5@1.10.7%apple-clang@13.0.0~cxx~fortran~hl~ipo~java~mpi+shared~zip~threadsafe+tools api=default b
^cmake@3.21.4%apple-clang@13.0.0~doc~ncurses+openssl+ownlibs~qt build_type=Release arch=darwin-bi
^ncurses@6.2%apple-clang@13.0.0~symlinks+term lib abi=none arch=darwin-bigsur-skylake
^pkgconf@1.8.0%apple-clang@13.0.0 arch=darwin-bigsur-skylake
^openssl@1.1.1%apple-clang@13.0.0~docs certs=system arch=darwin-bigsur-skylake
^perl@5.34.0%apple-clang@13.0.0~cpanm+shared+threads arch=darwin-bigsur-skylake
^berkeley-db@18.1.4.0%apple-clang@13.0.0~cxx~docs~stl patches=b231fcc4d5cff05e5c3a4814f
^bzip2@1.0.8%apple-clang@13.0.0~debug~pic+shared arch=darwin-bigsur-skylake
^diffutils@3.8%apple-clang@13.0.0 arch=darwin-bigsur-skylake
^libiconv@1.16%apple-clang@13.0.0 libs=shared,static arch=darwin-bigsur-skylake
^gdbm@1.19%apple-clang@13.0.0 arch=darwin-bigsur-skylake
^readline@8.1%apple-clang@13.0.0 arch=darwin-bigsur-skylake
^zlib@1.2.11%apple-clang@13.0.0+optimize+pic+shared arch=darwin-bigsur-skylake
^openmpi@4.1.1%apple-clang@13.0.0~atomics~cuda~cxx~exceptions+gpfs~internal~hwloc~java~legacy
^hwloc@2.6.0%apple-clang@13.0.0~cairo~cuda~gl~libudev~libxml2~netloc~nvml~opencl~pci~rocm+sha
^libxml2@2.9.12%apple-clang@13.0.0~python arch=darwin-bigsur-skylake
^xz@5.2.5%apple-clang@13.0.0~pic libs=shared,static arch=darwin-bigsur-skylake
^libevent@2.1.12%apple-clang@13.0.0+openssl arch=darwin-bigsur-skylake
^openssh@8.7p1%apple-clang@13.0.0 arch=darwin-bigsur-skylake
^libedit@3.1-20210216%apple-clang@13.0.0 arch=darwin-bigsur-skylake
```

Pure hash-based reuse: all misses

```
(spackle):spack> spack solve --reuse -Il hdf5
=> Best of 10 considered solutions.
=> Optimization Criteria:
```

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

```
hdf5@1.10.7%apple-clang@12.0.5~cxx~fortran~hl~ipo~java~mpi+shared~zip~threadsafe+tools api=default
^cmake@3.21.1%apple-clang@12.0.5~doc~ncurses+openssl+ownlibs~qt build_type=Release arch=darwin-
^ncurses@6.2%apple-clang@12.0.5~symlinks+term lib abi=none arch=darwin-bigsur-skylake
^openssl@1.1.1%apple-clang@12.0.5~docs+systemcerts arch=darwin-bigsur-skylake
^zlib@1.2.11%apple-clang@12.0.5+optimize+pic+shared arch=darwin-bigsur-skylake
^openmpi@4.1.1%apple-clang@12.0.5~atomics~cuda~cxx~exceptions+gpfs~internal~hwloc~java~leg
^hwloc@2.6.0%apple-clang@12.0.5~cairo~cuda~gl~libudev~libxml2~netloc~nvml~opencl~pci~rocm+sh
^libxml2@2.9.12%apple-clang@12.0.5~python arch=darwin-bigsur-skylake
^libiconv@1.16%apple-clang@12.0.5 libs=shared,static arch=darwin-bigsur-skylake
^xz@5.2.5%apple-clang@12.0.5~pic libs=shared,static arch=darwin-bigsur-skylake
^pkgconf@1.8.0%apple-clang@12.0.5 arch=darwin-bigsur-skylake
^libevent@2.1.12%apple-clang@12.0.5+openssl arch=darwin-bigsur-skylake
^openssh@8.6p1%apple-clang@12.0.5 arch=darwin-bigsur-skylake
^libedit@3.1-20210216%apple-clang@12.0.5 arch=darwin-bigsur-skylake
^perl@5.34.0%apple-clang@12.0.5~cpanm+shared+threads arch=darwin-bigsur-skylake
^berkeley-db@18.1.4.0%apple-clang@12.0.5~cxx~docs~stl patches=b231fcc4d5cff05e5c3a4814f
^bzip2@1.0.8%apple-clang@12.0.5~debug~pic+shared arch=darwin-bigsur-skylake
^gdbm@1.19%apple-clang@12.0.5 arch=darwin-bigsur-skylake
^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



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
```

- 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

Concrete spack.lock file (generated)

```
{
  "concrete_specs": {
    "6s63so2kstp3zyvjezgldmavy6l3nul": {
      "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": {
          "cxx": false,
          "debug": false,
          "fortran": false,
          "hl": false,
          "mpi": true,
          "openmpi": true
        }
      }
    }
  }
}
```



Hands on Section

Environments



Hands on Section

Configuration



Hands on Section

Software Stacks



Hands on Section

Creating Packages



Hands on Section

Binary Caches & Mirrors



Hands on Section

Developer Workflows



Hands on Section

Scripting



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.rtf.d.io



slack.spack.io



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



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



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



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

We hope to make distributing & using HPC software easy!