Managing HPC Software Complexity with Spack

CARLA25 DevOps School

September 22, 2025

Caetano Melone

LLNL-PRES-2011458

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.”

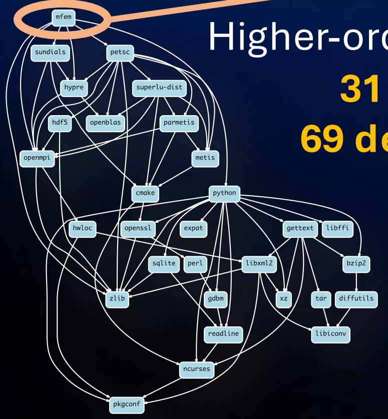
P. DuBois & T. Epperly. ***Why Johnny Can’t Build***. Scientific Programming. Sep/Oct 2003.

- **Pros**
  - 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**

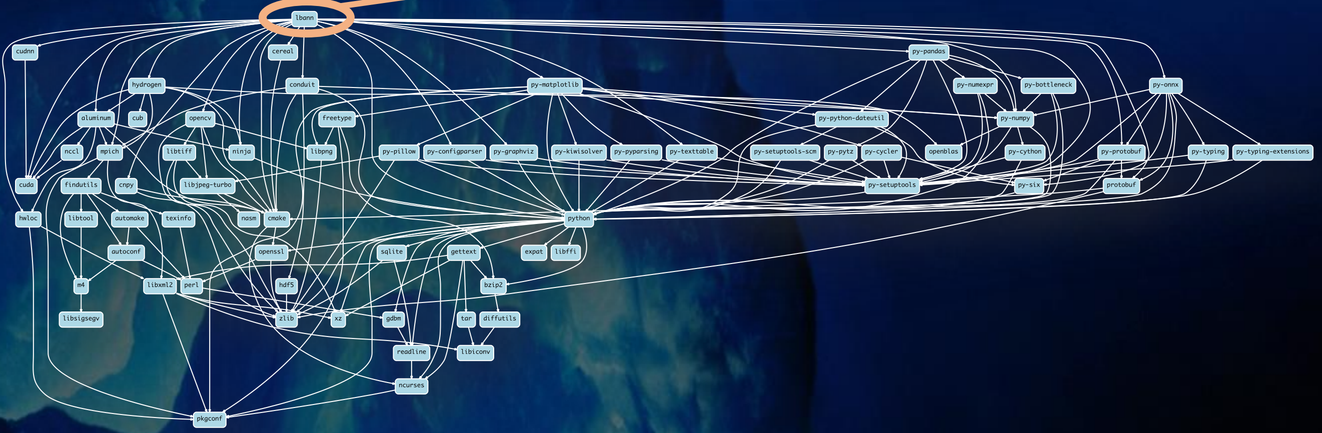
Build-time incompatibility; fail fast

Appears to work; subtle errors later

Modern scientific codes rely on icebergs of dependency libraries

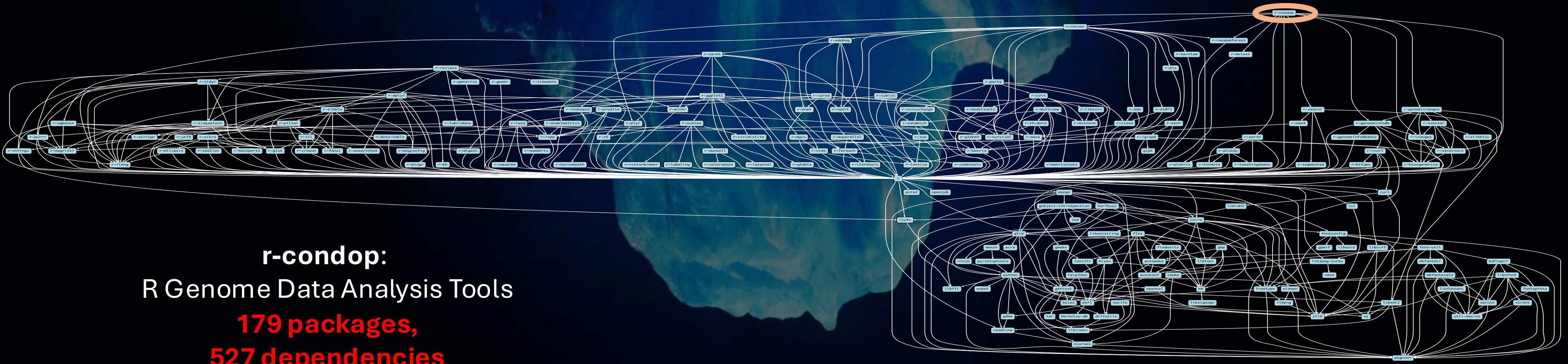


MFEM:
Higher-order finite elements
31 packages,
69 dependencies



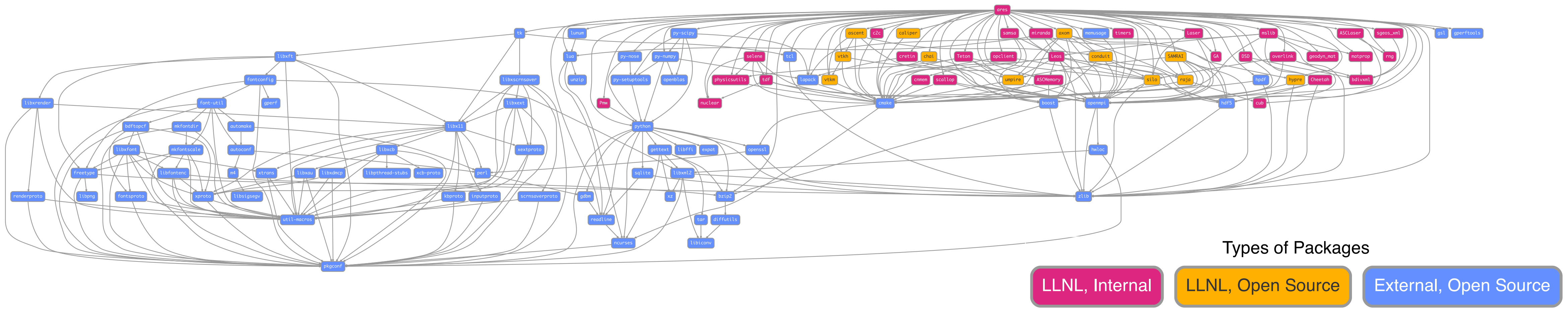
LBANN: Neural Nets for HPC

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 by hand?

Some 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

Spack enables software distribution for HPC

No installation required: clone and go

```
$ git clone --depth=2 https://github.com/spack/spack
$ 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 +threadssafe       $ 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

What’s a package manager?

- Spack is a ***package manager***
  - **Does not** replace a 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

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

1. **Functional Package Managers**
   - Nix https://nixos.org
   - Guix https://hpc.guix.info

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

Other HPC tools:

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

- **Conda / Mamba / Pixi** https://conda.io https://mamba.readthedocs.io https://prefix.dev
  - Very popular binary package ecosystem for data science
  - Not targeted at HPC; generally, has unoptimized binaries

What about containers?

- **Containers provide a great way to reproduce and distribute an already-built software stack**
- **Someone needs to build the container!**
  - Not 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**
  - Not clear that we can ever build one container for all facilities

Spack provides a *spec* syntax to describe customized package configurations

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

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

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

Spack packages are *parameterized* using the spec syntax

Python DSL defines many ways to build

```
from spack import *

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

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

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

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

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

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

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

Base package (CMake support)

Metadata at the class level

Versions

Variants (build options)

Dependencies (same spec syntax)

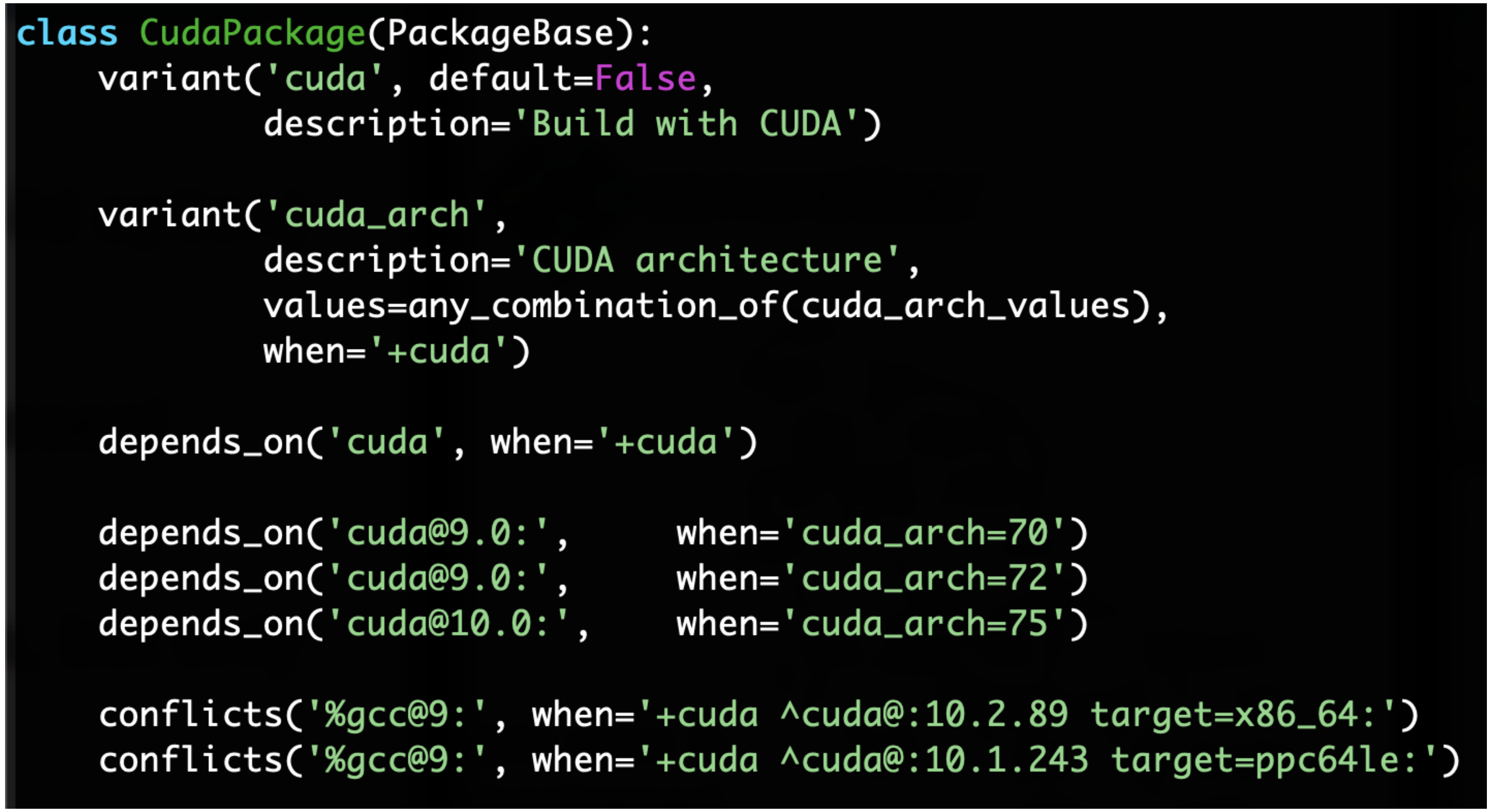
Install logic in instance methods

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

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

Conditional variants simplify packages

CudaPackage: a mix-in for packages that use CUDA



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

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

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

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

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

cuda is a variant (build option)

cuda_arch is only present if cuda is enabled

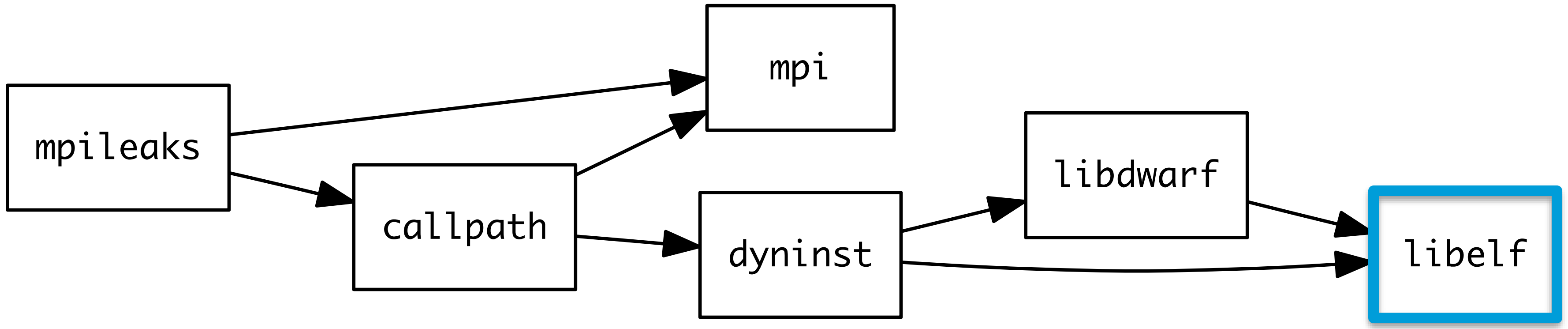
dependency on cuda, but only if cuda is enabled

constraints on cuda version

compiler support for x86_64 and ppc64le

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

Spack Specs can constrain versions of dependencies

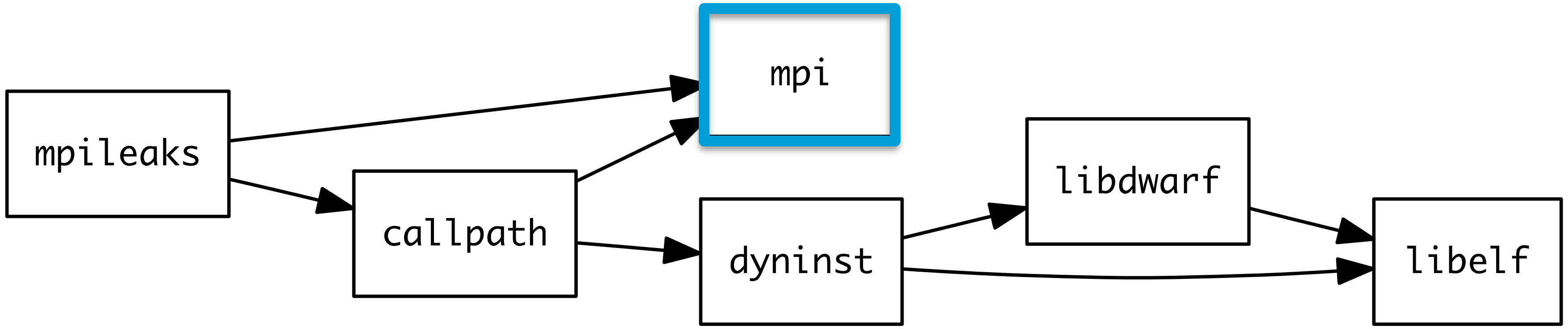


```
$ spack install mpileaks %intel@12.1 ^libelf@0.8.12
```

- Spack ensures *one* configuration of each library per DAG
  - Ensures ABI consistency.
  - User does not need to know DAG structure; only the dependency *names*.

- Spack can ensure that builds use the same compiler, or you can mix
  - Working on ensuring ABI compatibility when compilers are mixed.

Spack handles ABI-incompatible, versioned interfaces like MPI



- mpi is a *virtual dependency*

- Install the same package built with two different MPI implementations:

```
$ spack install mpileaks ^mvapich@1.9
```

```
$ spack install mpileaks ^openmpi@1.4:
```

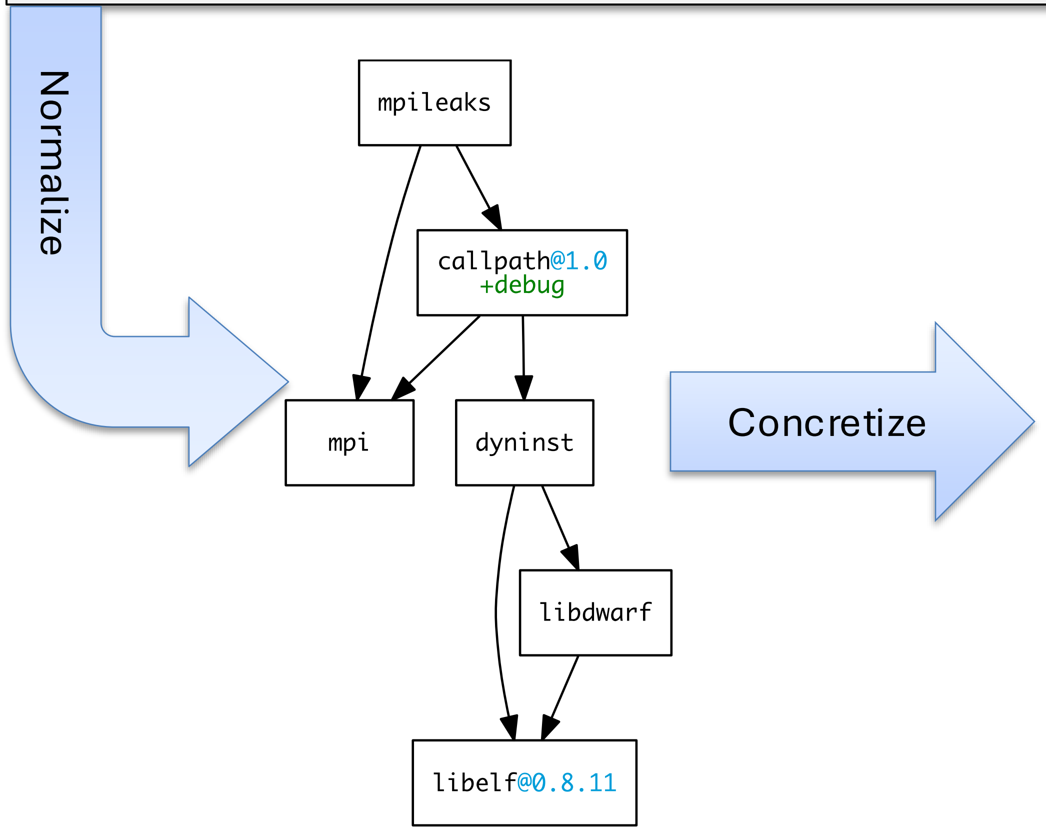
- Let Spack choose MPI implementation, as long as it provides MPI 2 interface:

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

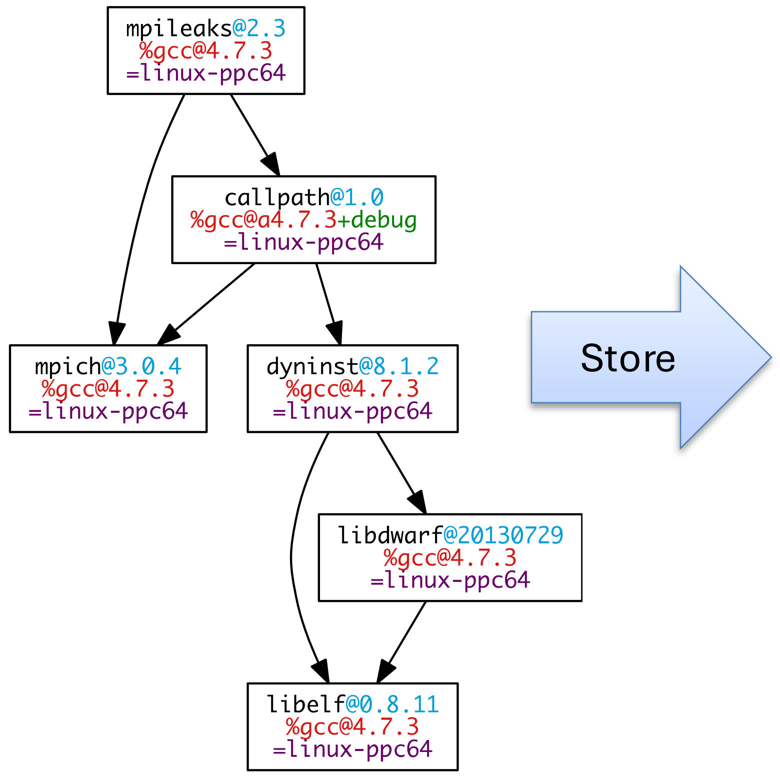
Concretization fills in missing configuration details when the user is not explicit

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

User input: *abstract* spec with some constraints



Abstract, normalized spec with some dependencies.



mpileaks@2.3 %gcc@4.7.3 =linux-ppc64
callpath@1.0 %gcc@a4.7.3+debug =linux-ppc64
mich@3.0.4 %gcc@4.7.3 =linux-ppc64
dyninst@8.1.2 %gcc@4.7.3 =linux-ppc64
libdwarf@20130729 %gcc@4.7.3 =linux-ppc64
libelf@0.8.11 %gcc@4.7.3 =linux-ppc64
Store

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

spec.yaml

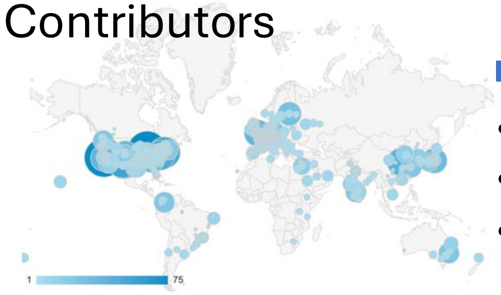
```
spec:
- mpileaks:
   arch: linux-x86_64
   compiler:
     name: gcc
     version: 4.9.2
   dependencies:
     adept-utils: kszrtkpbzac3ss2ixcjkcorlaybnptp4
     callpath: bah5f4h4d2n47mgycej2mtrnrivvxy77
     mpich: aa4ar6ifj23yijqmdabeakpejcli72t3
   hash: 33hjjhxi7p6gyzn5ptgyes7sghyprujh
   variants: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

Detailed provenance is stored with the installed package

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

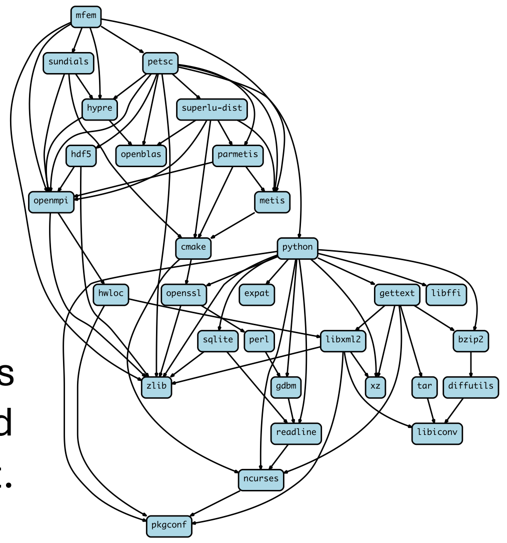
spack developers — default config packages.yaml (yaml)

admins, users — al preferences config packages.yaml (yaml)

users — local environment config spack.yaml (yaml)

users — Command line constraints

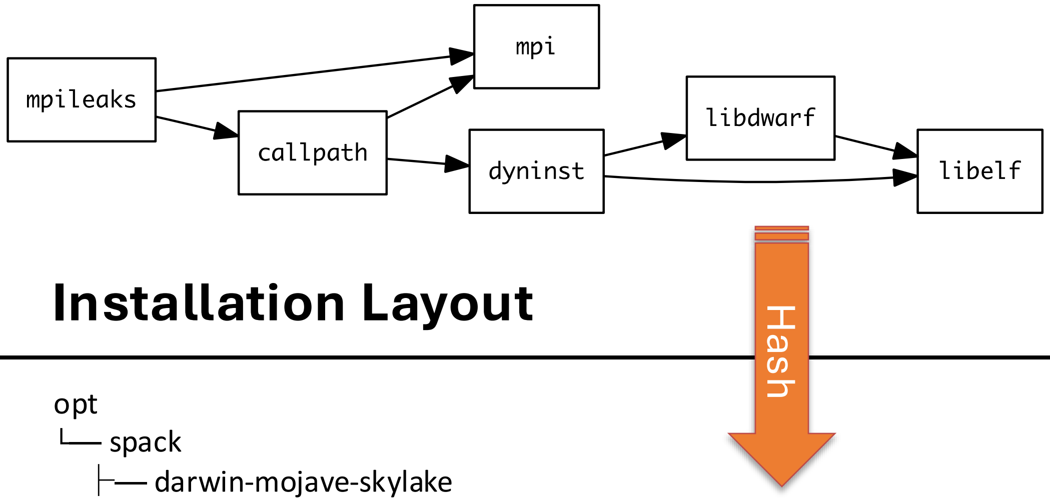
```
spack install hdf5@1.12.0 +debug
```



Concrete spec is fully constrained and can be built.

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-daqqpssxb6qbfrztsezkmhus3xoflbsy
    │       ├── sqlite-3.30.1-u64v26igxvxyn23hysmklfums6tgjv5r
    │       ├── xz-5.2.4-u5eawkvaoc7vonabe6nndkcfwuv233cj
    │       └── zlib-1.2.11-x46q4wm46ay4pltriijbgizxjrhbaka6
    ├── darwin-mojave-x86_64
    │   └── clang-10.0.0-apple
    │       └── coreutils-8.29-pl2kcytejqcys5dzecfrtjqxfdssvnob
```

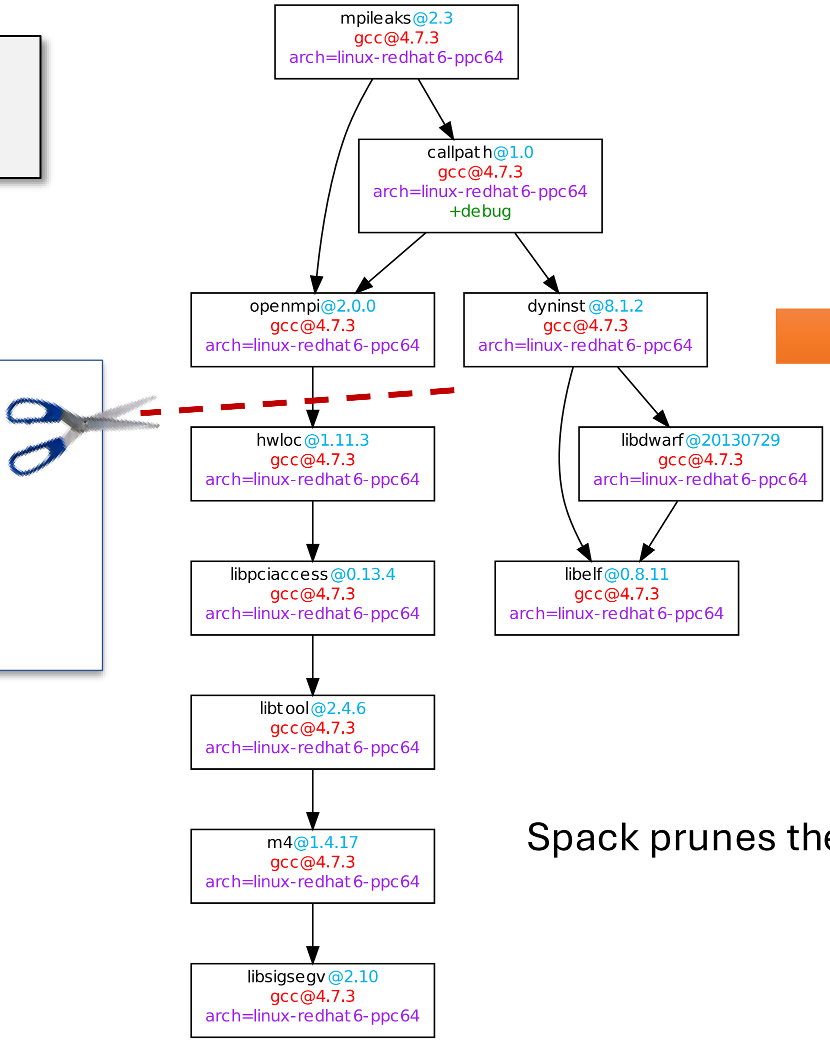
- Each unique dependency graph is a unique ***configuration***.
- Each configuration in a unique directory.
  - Multiple configurations of the same package can coexist.
- **Hash** of entire directed acyclic graph (DAG) is appended to each prefix.
- Installed packages automatically find dependencies
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

We can configure Spack to build with external software

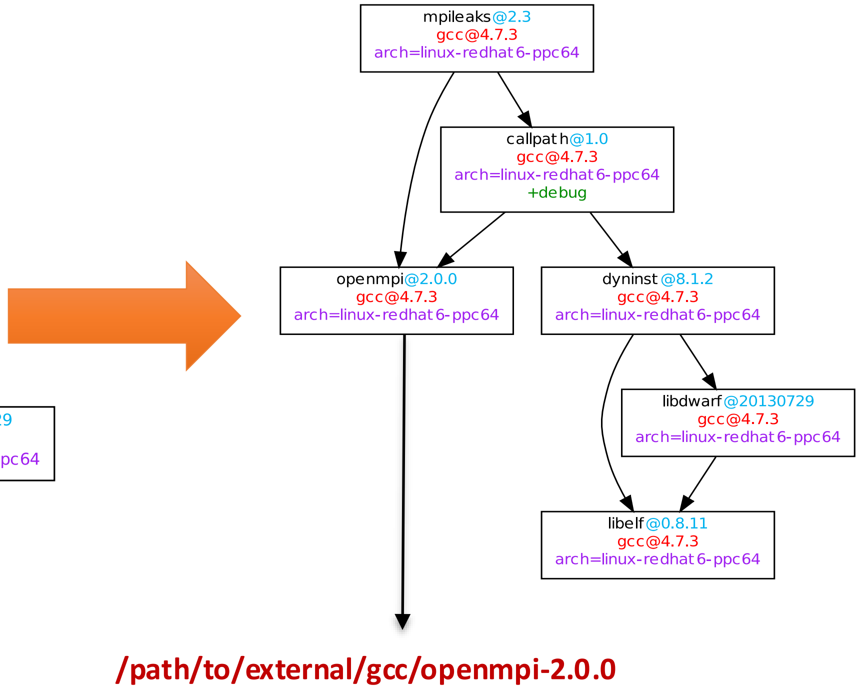
```
mpileaks ^callpath@1.0+debug
  ^openmpi ^libelf@0.8.11
```

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
      ...
```



mpileaks@2.3 gcc@4.7.3 arch=linux-redhat6-ppc64
callpath@1.0 gcc@4.7.3 arch=linux-redhat6-ppc64 +debug
openmpi@2.0.0 gcc@4.7.3 arch=linux-redhat6-ppc64
dyninst@8.1.2 gcc@4.7.3 arch=linux-redhat6-ppc64
hwloc@1.11.3 gcc@4.7.3 arch=linux-redhat6-ppc64
libdwarf@20130729 gcc@4.7.3 arch=linux-redhat6-ppc64
libpciaccess@0.13.4 gcc@4.7.3 arch=linux-redhat6-ppc64
libelf@0.8.11 gcc@4.7.3 arch=linux-redhat6-ppc64
libtool@2.4.6 gcc@4.7.3 arch=linux-redhat6-ppc64
m4@1.4.17 gcc@4.7.3 arch=linux-redhat6-ppc64
libsigsegv@2.10 gcc@4.7.3 arch=linux-redhat6-ppc64



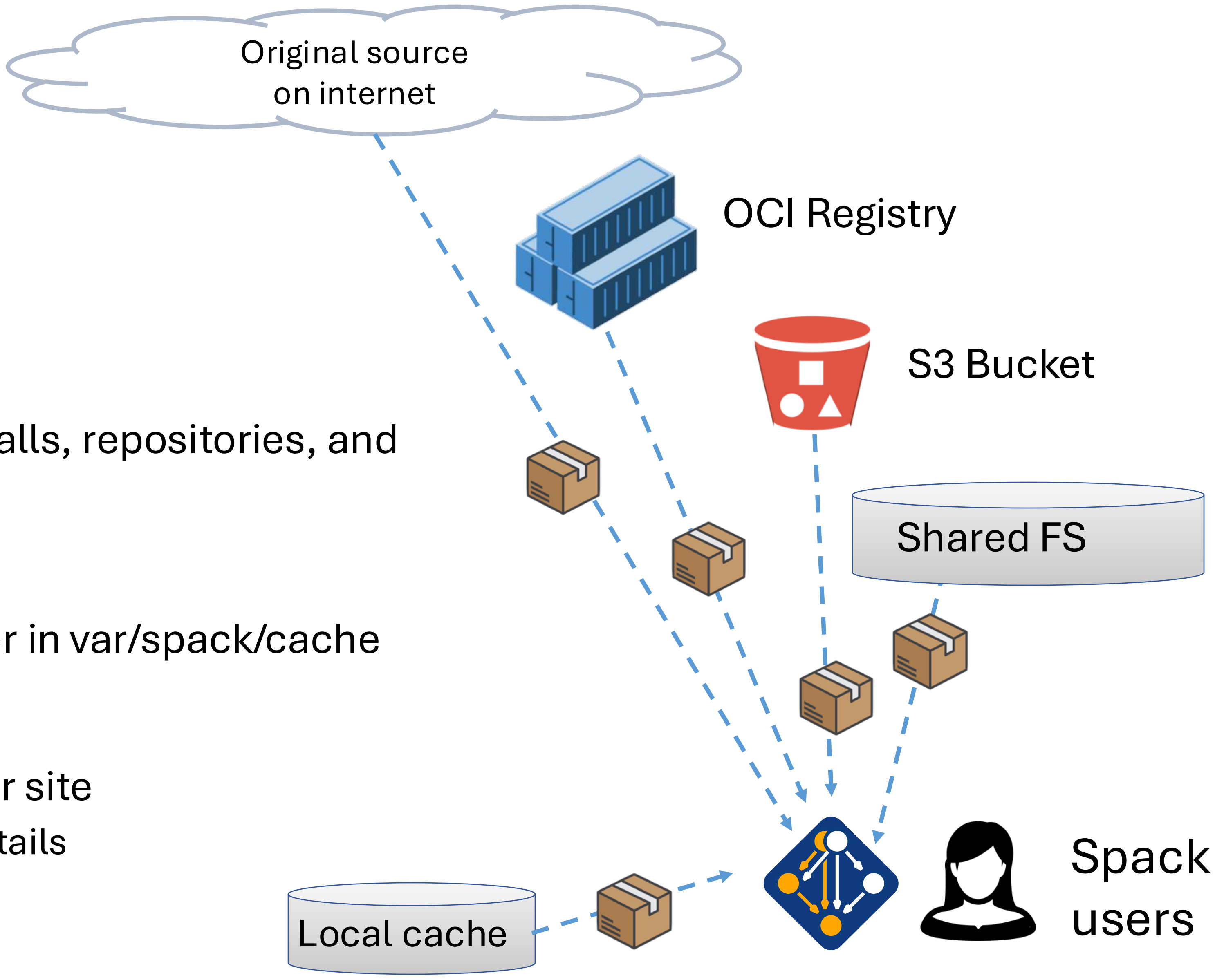
mpileaks@2.3 gcc@4.7.3 arch=linux-redhat6-ppc64
callpath@1.0 gcc@4.7.3 arch=linux-redhat6-ppc64 +debug
openmpi@2.0.0 gcc@4.7.3 arch=linux-redhat6-ppc64
dyninst@8.1.2 gcc@4.7.3 arch=linux-redhat6-ppc64
libdwarf@20130729 gcc@4.7.3 arch=linux-redhat6-ppc64
libelf@0.8.11 gcc@4.7.3 arch=linux-redhat6-ppc64
/path/to/external/gcc/openmpi-2.0.0

Users register external packages in a configuration file

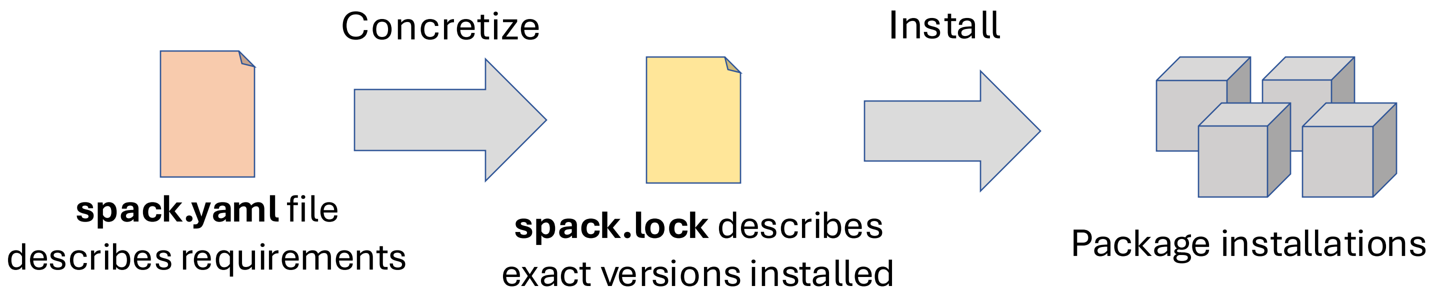
Spack prunes the DAG when adding external packages.

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, 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

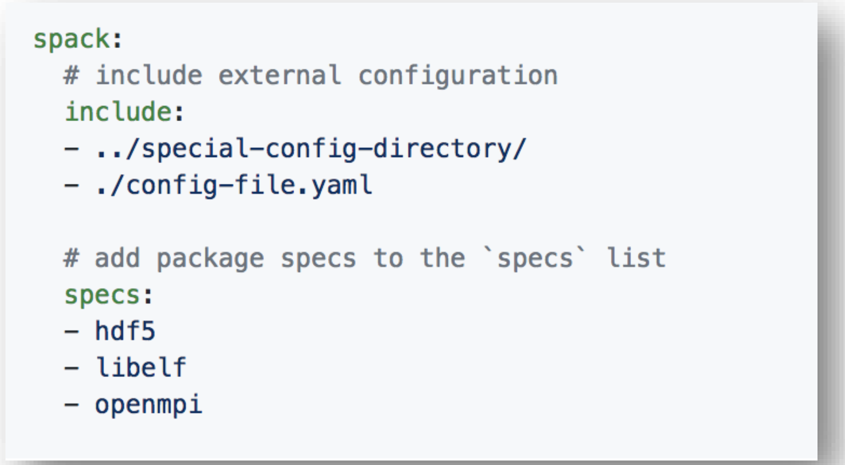


Environments enable users to build customized stacks from an abstract description


Concretize
Install
spack.yaml file describes requirements
spack.lock describes exact versions installed
Package installations

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

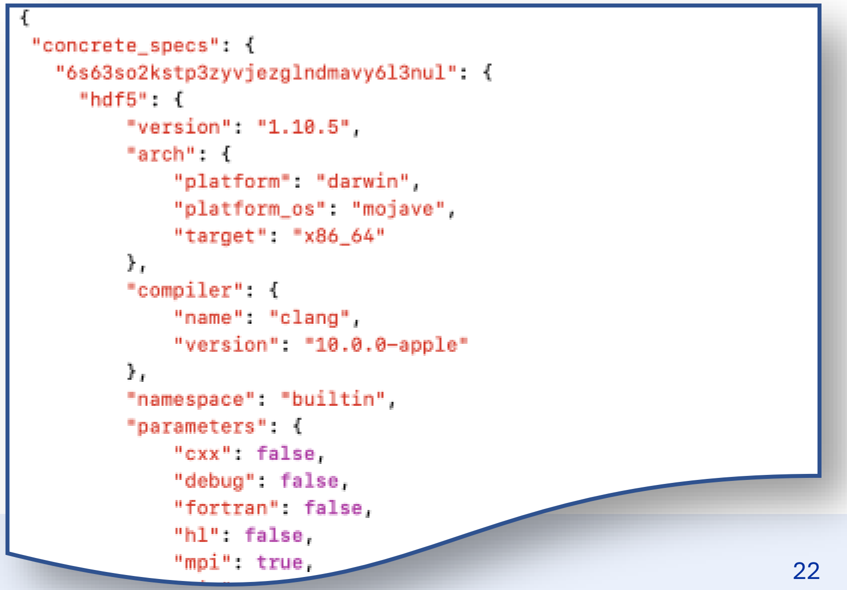
Simple spack.yaml file



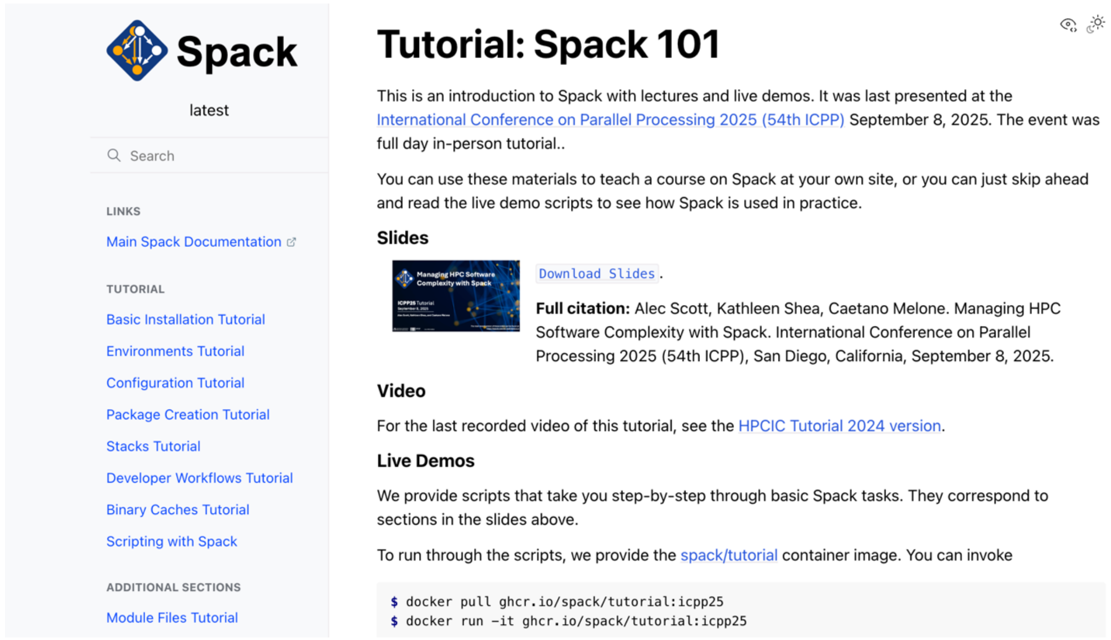
```
spack:
  # include external configuration
  include:
  - ../special-config-directory/
  - ./config-file.yaml

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

Concrete spack.lock file (generated)



```
{
 "concrete_specs": {
   "6s63so2kstp3zyvjezglndmavy613nul": {
     "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,
```


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

Tutorial: Spack 101

This is an introduction to Spack with lectures and live demos. It was last presented at the International Conference on Parallel Processing 2025 (54th ICPP) September 8, 2025. The event was full day in-person tutorial..

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

Slides

Download Slides.

Full citation: Alec Scott, Kathleen Shea, Caetano Melone. Managing HPC Software Complexity with Spack. International Conference on Parallel Processing 2025 (54th ICPP), San Diego, California, September 8, 2025.

Video

For the last recorded video of this tutorial, see the HPCIC Tutorial 2024 version.

Live Demos

We provide scripts that take you step-by-step through basic Spack tasks. They correspond to sections in the slides above.

To run through the scripts, we provide the spack/tutorial container image. You can invoke

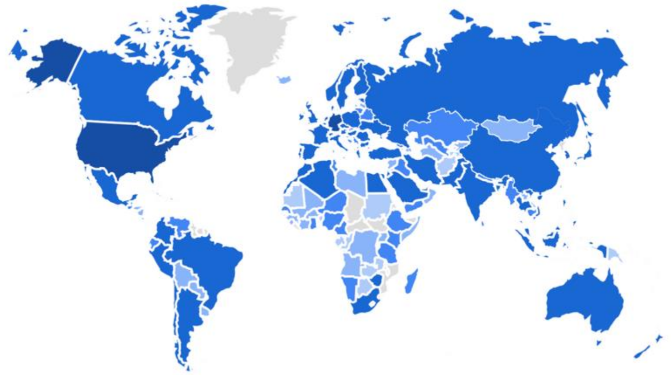
```
$ docker pull ghcr.io/spack/tutorial:icpp25
$ docker run -it ghcr.io/spack/tutorial:icpp25
```

Demo

Excerpts from the Spack Tutorial

spack-tutorial.rtfd.io

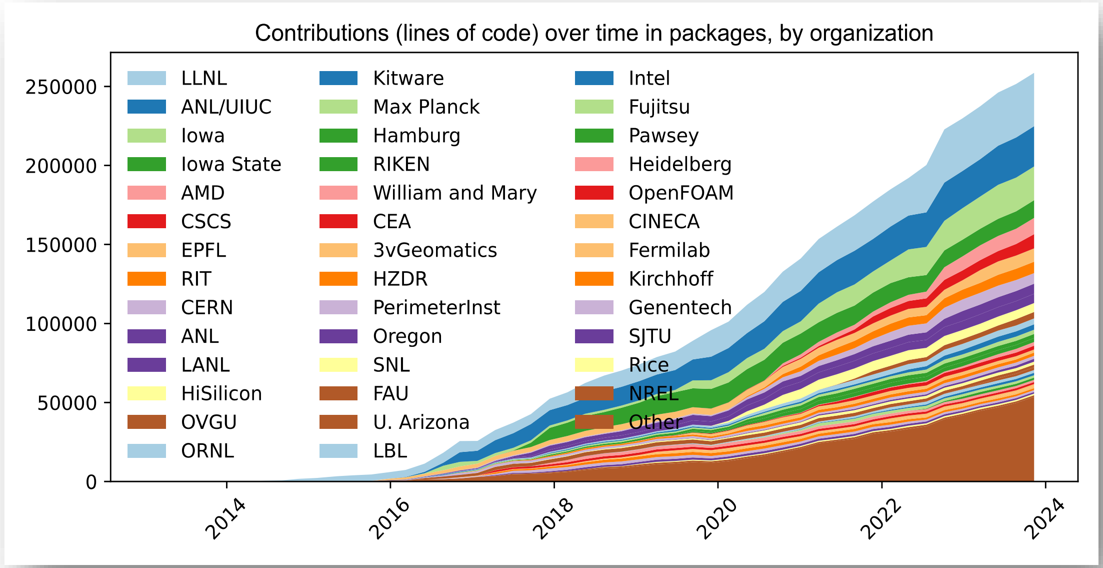
Spack sustains the HPC software ecosystem with the help of many contributors



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

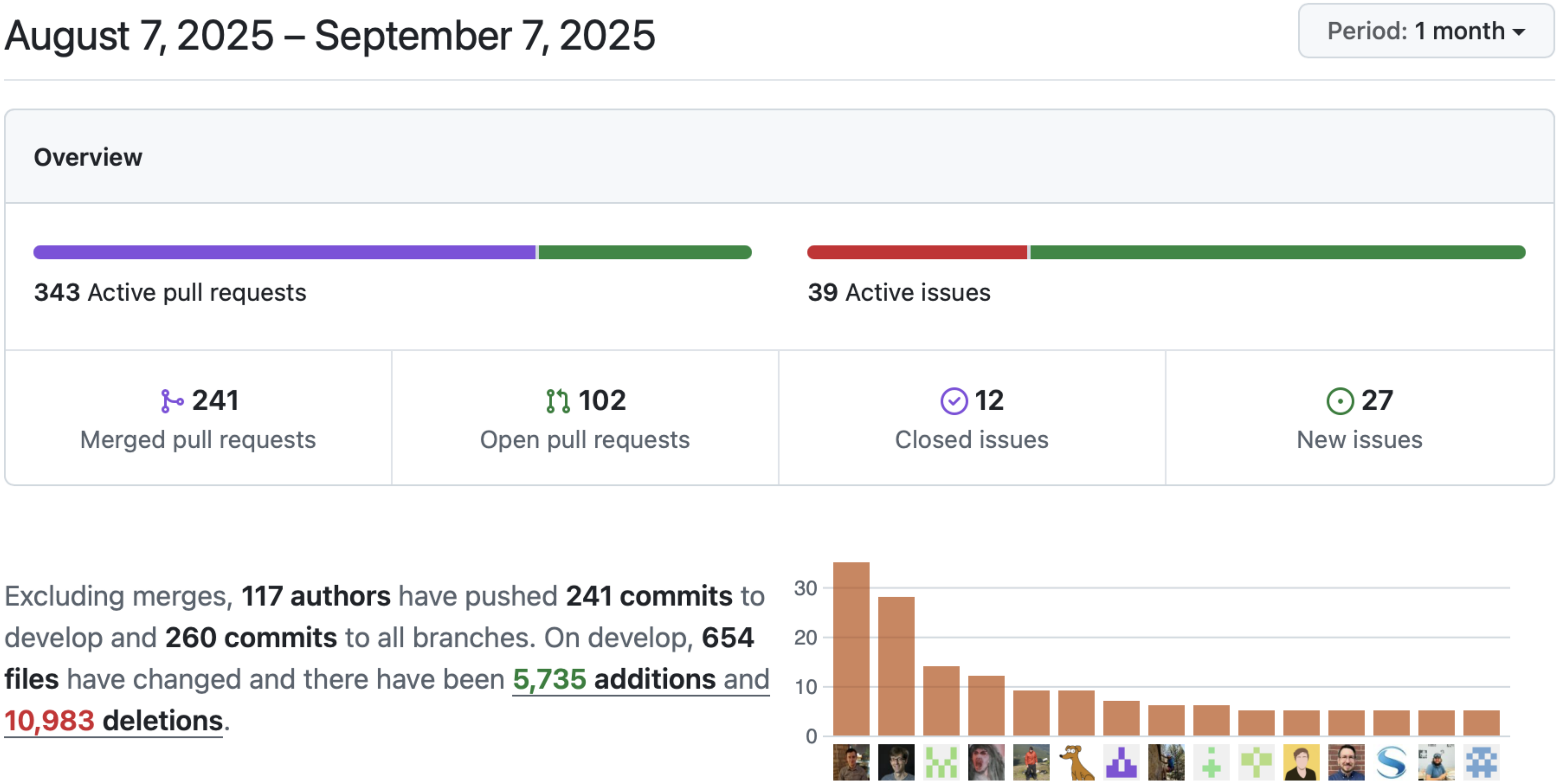
2023 aggregate documentation user counts from GA4
(note: yearly user counts are almost certainly too large)

Over 8,500 software packages
Over 1,500 contributors



Contributors continue to grow worldwide!

One month of Spack development is pretty busy!



August 7, 2025 – September 7, 2025

Period: 1 month

Overview

343 Active pull requests

39 Active issues

241	**102**	**12**	**27**
Merged pull requests	Open pull requests	Closed issues	New issues

Excluding merges, **117 authors** have pushed **241 commits** to develop and **260 commits** to all branches. On develop, **654 files** have changed and there have been **5,735 additions** and **10,983 deletions**.

Spack's widespread adoption has enabled collaborations with 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 is 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

Connect with the Spack community

Spack is part of the
High Performance Software Foundation

Join us at the Spack User Meeting at
HPSFCon 2026 next year!

@hpsf.bsky.social

hpsf.io

- Join us and 3,900+ others on Spack slack
- Contribute packages, docs, and features on GitHub
- Follow the tutorial at spack-tutorial.rtfd.io

slack.spack.io

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

@spackpm.bsky.social

@spack@hpc.social

@spackpm

spack.io

We hope to make distributing & using HPC software easy!