Spack and HPSF Update

Spack's road to v1.0

Easybuild User Meeting 2025
Julich Supercomputing Center
March 24, 2025

THE LINUX FOUNDATION

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

Spack is now a Linux Foundation project

Spack is a “Core” project within the High Performance Software Foundation

What does this mean for the community?

- HPSF guidelines are helping us improve our project governance
- Linux Foundation has outreach experts who are helping us grow the community
- Collaborations with other HPC projects around CI and testing
  - CI working group is converting the CI system we built for Spack into a general resource for other projects
- HPSF can run events

HPSF
HIGH PERFORMANCE SOFTWARE FOUNDATION

Spack has a new Technical Steering Committee

Todd Gamblin, LLNL
TSC Chair

Greg Becker
LLNL

Massimiliano Culpo
n.p. complete s.r.l

Tammy Dahlgren
LLNL

Wouter Deconinck
U. Manitoba

Ryan Krattiger
Kitware

Mark Krentel
Rice University

John Parent
Kitware

Marc Paterno
Fermilab

Luke Peyralans
U. Oregon

Phil Sakievich
Sandia

Peter Scheibel
LLNL

Adam Stewart
TU Munich

Harmen Stoppels
Stoppels Consulting

HPSF
HIGH PERFORMANCE SOFTWARE FOUNDATION

Spack Governance

https://github.com/spack/governance

We aim to continue governing primarily through consensus

The Technical Steering Committee will vote to resolve technical discussions that cannot be resolved by consensus

The TSC meets monthly to discuss

- Big-picture technical priorities for Spack development
- Release schedule and feature sets
- Technical disagreements requiring votes
- Pull request and issue backlog and trajectory

The High Performance Software Foundation

- Started in May 2024
- Increasing collaboration among projects, labs, and industry
- HPSF Con in May 2025
- Activities:
  - CI working group developing “HPC CI” service for projects
  - Project lifecycle + mentoring
  - Outreach
- Bringing in new projects and members
  - 2 new members since launch, expecting more!

HPSF HIGH PERFORMANCE SOFTWARE FOUNDATION

Founding Projects

Spack

kokkos

APPTAINER

Viskores

TRILINOS

AMReX

HPCToolkit

Charliecloud

E4S

HPSF
HIGH PERFORMANCE SOFTWARE FOUNDATION

New Projects have joined!

More to come …

We will have the first Spack user group meeting May 7-8 at the first annual HPSF Conference

- HPSF Con will be May 5 - 8 2025
- Embassy Suites Chicago Magnificent Mile
  - 45 minutes from O'Hare
  - Renovated in 2018
- Program
  - **Monday + Tuesday:**
    - HPSF Plenary sessions
    - Working group collaborations
  - **Tues + Wed**
    - Project meetings
    - Tutorials

HPSF
HIGH PERFORMANCE SOFTWARE FOUNDATION

https://bit.ly/hpsfcon-2025

We have 34 contributed talks lined up for the very first Spack User Meeting

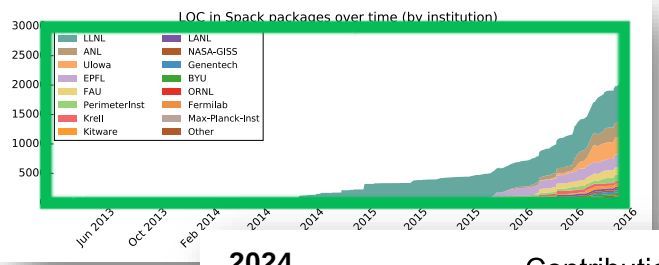
Wednesday, May 7	
Opening	
Welcome and Overview	Todd Gamblin
State of the Spack Community	Todd Gamblin
Spack v1.0	Greg Becker
Building Spack	
Optimizing Spack: Multi-Package Parallel Builds for Faster Installation	Kathleen Shea
Fast binary installation with Spack splicing	John Gouwar
Spack on Windows	John Parent
Spack CI: Past, Present, and Future	Ryan Krattiger
Collaborations using Spack	
E4S and Spack	Sameer Shende
Spack-stack: An interagency collaboration and spack extension	Dom Heinzeller
Packaging for HPC and HTC in High Energy and Nuclear Physics: Comparing Spack to other solutions	Wouter Deconinck
Developing and Distributing HEP Software Stacks with Spack	Kyle Knoepfel
GAIA: A Software Deployment Strategy, Ordeals, Success And General Applicability	Etienne Malaboeuf
Lightning Talks	
Closing Gaps in Spack for Software Application DevOps Infrastructure	Phil Sakievich
Development of Complex Software Stacks with Spack	Cedric Chevalier
Feedback on using Spack to deploy a development environment for the gyselalibxx library	Thomas Padioleau
Spack at the Linac Coherent Light Source: Progress, Success and Challenges	Valerio Mariani
Towards a Zero-Install Programming Environment	Mike Kiernan
Deploying Software on Frontier with NCCS Software Provisioning (NSP)	Fernando Posada
Organizational Approach to Spack Engagement: A case study	Phil Sakievich
Dynamic Resource Allocation for Continuous Integration Build Pipelines	Caetano Melone

Thursday, May 8	
Cloud, benchmarking, and containers	
Creating reproducible performance optimization pipelines with Spack and Ramble	Doug Jacobsen
Democratizing Access to Optimized HPC Software Through Build Caches	Stephen Sachs
Spack, containers, CMake: the good, the bad & the ugly in the CI & distribution of the PDI library	Julien Bigot
Spack Deployment Story at LBNL/UC Berkeley	Abhiram Chintangal
Developer Workflows: Challenges and lessons learned	
Lessons Learned from Developing and Shipping Advanced Scientific Compressors with Spack	Robert Underwood
Challenges mixing Spack-optimized hardware accelerator libraries on personal scientifiic computers	Pariksheet Nanda
An opinionated-default approach to enhance Spack Developer Experience	Kin fai Tse
Developing and Managing Data Acquisition Software Using Spack	Eric Flumerfelt
Site Deployment Stories	
From Complexity to Efficiency: SPACK's Impact on NSM Supercomputers	Samir Shaikh, Harshitha Ugave
Deploying a Large HPC Software Stack - Challenges and Experiences	Jose Gracia
Building and Maintaining OSS on Fugaku: RIKEN's Experience with Spack	Yuchi Otsuka
Using Spack to Build and Maintain a Facility-Specific Programming Environment	Nicholas Sly
Aurora PE: rethinking software integration in the exascale era	Sean Koyama
DevOps for Large Applications	
Driving Continuous Integration and Developer Workflows with Spack	Richard Berger
Implementing a Security Conscious Build Configuration Relay with a Shared Build Cache	Chris White
Spack-based WEAVE environment at LLNL	Lina Muryanto
DevOps for Monolithic Repositories using Spack	Phil Sakievich

HPSF
HIGH PERFORMANCE SOFTWARE FOUNDATION

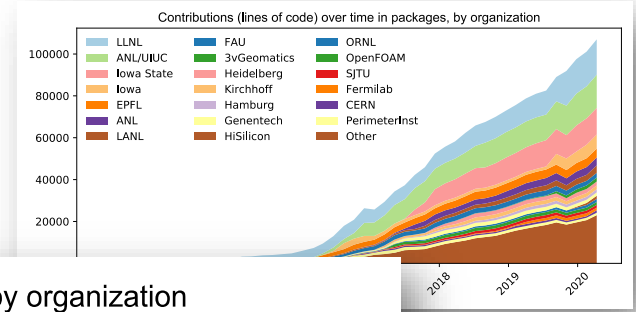
https://spack.io/sum25-schedule/

Kenneth declined 😢

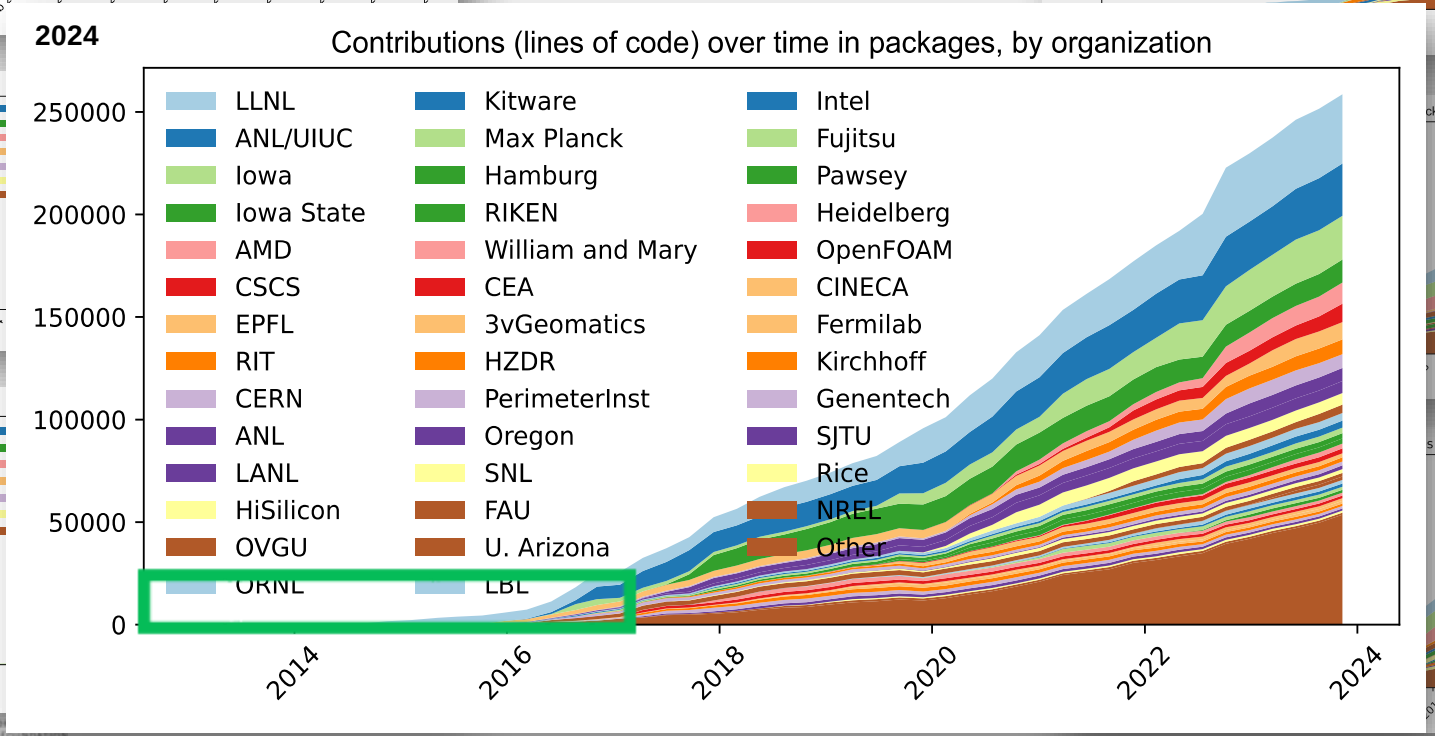


2016

Fall
2020

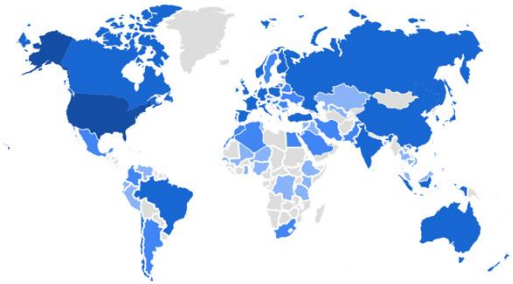


2024

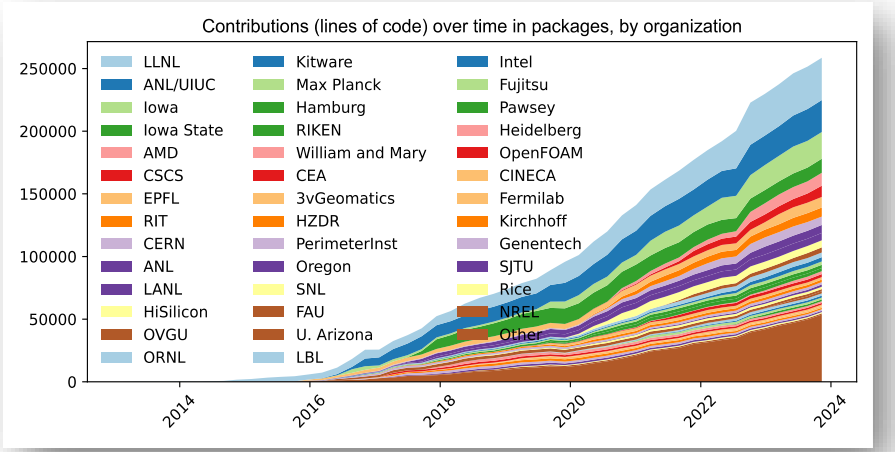


Spack sustains the HPC software ecosystem with the help of many contributors

Over 8,300 software packages
Over 1,460 contributors



COUNTRY	ACTIVE USERS	
United States	2.1K	↑4.2%
China	351	↑8.7%
Germany	325	↓23.2%
United Kingdom	301	↑4.2%
France	216	↑4.9%
India	209	↓16.7%
Hong Kong	178	↑50.8%



July 2024: 5,400 Active Users (per GA4)
Typically 120 – 150 contributors / month

Contributors continue to grow worldwide!

Spack v0.23.0 was released in November

Highlights:

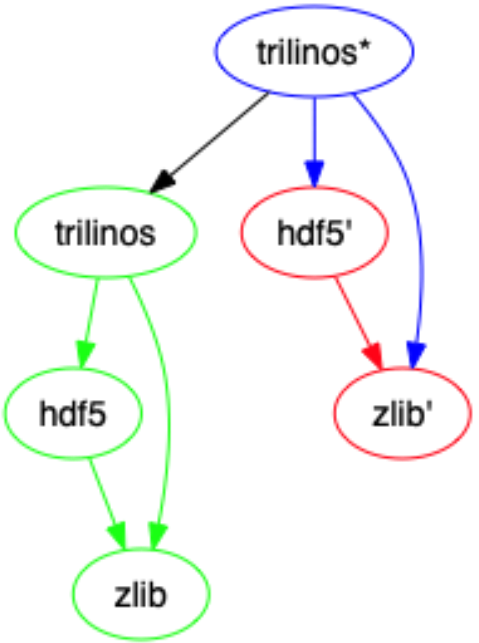
- Spec splicing
- Broader variant propagation
- Query specs by namespace
- Spack commands respect `concretizer:unify` config outside environments
- Improved formatting for `spack spec` and `spack find` commands
- `spack env track` converts environments from independent to managed
- New software stacks in CI and public binary cache
  - ML stack for linux/aarch64
  - Developer tools stack for macos/aarch64
- 329 new packages since v0.22
- Thank you to the 373 contributors to this release
  - 60 contributors to Spack core, 357 contributors to Spack packages

Spec splicing makes binary swapping possible

Reuse binary packages built against one dependency while using a new dependency.

Packages retain pointers to their original configuration for provenance

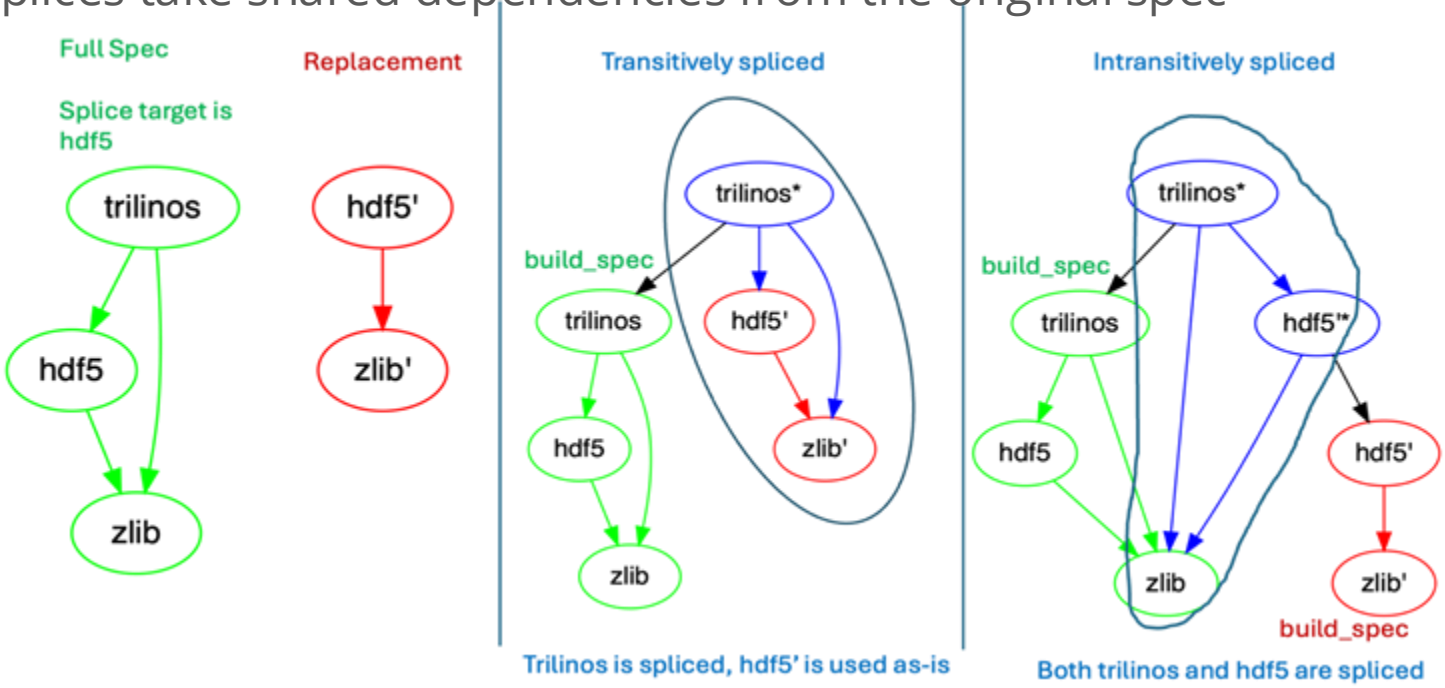
Relocation logic is repurposed for “rewiring” spec to its new configuration



Transitive and Intransitive Splices

“Transitive” splices take shared dependencies from the new dependency

“Intransitive” splices take shared dependencies from the original spec



Explicit Splicing

```
concretizer:
  splice:
    explicit:
    - target: mpi
      replacement: mvapich2/abcdef
      transitive: false
```

Any spec that concretizes to depend on mpi will be spliced to use the local mvapich2 with hash abcdef.

Explicit splicing requires the user to ensure ABI compatibility

Automatic Splicing

```
concretizer:
  splice:
    automatic: true
```

Packages have a new directive `can_splice`

```
can_splice("foo@1.1+a", when="@1.1", match_variants=["bar"])
```

“This package at version 1.1 can be spliced in for any package that satisfies `"foo@1.1+a"` as long as the “bar” variant values are equal

If splicing is enabled, the concretizer will apply these constraints and optimize for package reuse.

Splicing in the wild (from #appreciation on Spack Slack)

Tuesday, March 18th

Cameron Rutherford 🤗 10:34 AM

I just enabled splicing and the public build cache, and saw my build go from 1hr+ from source to only a handful of minutes. While I would need to re-compile to target `x86_64_v4` or to use a newer compiler, this is awesome for bootstrapping my dev-environment (where performance doesn't matter so much) with packages like `neovim`, `stow`, `xclip`, `python` and `npm`.

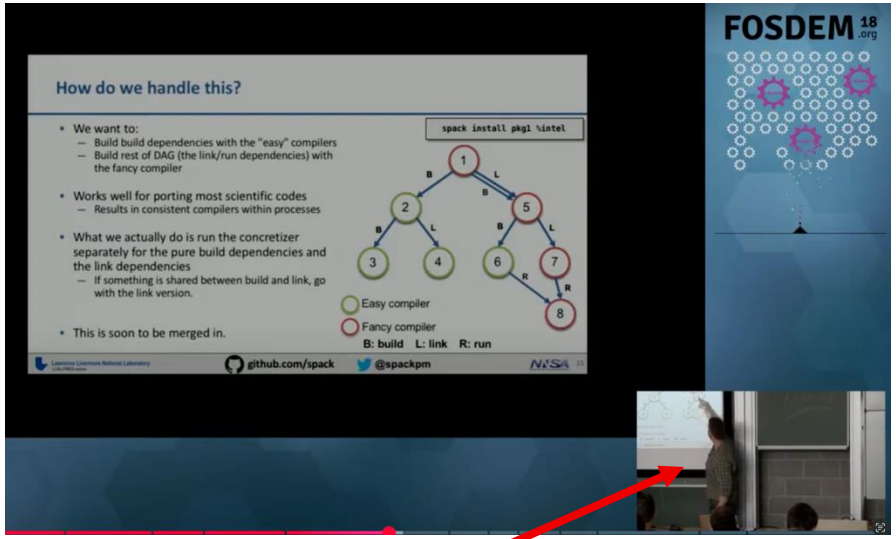
🎉 6



6 replies Last reply 5 days ago

The road to v1.0 has been long

- We wanted:
  - ✓ New ASP-based concretizer
  - ✓ Reuse of existing installations
  - ✓ Stable production continuous integration
  - ✓ Stable binary cache
  - Compiler dependencies (nearly done!)
  - Stable package API
  - Separate builtin repo from Spack tool
- v1.0 will:
  - Change the spec model for compilers
  - Enable users to use entirely custom packages
  - Improve reproducibility
  - Improve stability 🤞
- This is the largest change to Spack since the new concretizer



Me, presenting how simple all this would be at FOSDEM in 2018

HPSF
HIGH PERFORMANCE SOFTWARE FOUNDATION

Introducing Language Dependencies

```
depends_on("c", type="build")
depends_on("cxx", type="build")
depends_on("fortran", type="build")
```

- You now need to specify these to use c, cxx, or fortran
  - No-op in the release as we prepare for compilers as dependencies
  - Backported to v0.22 release to assist teams working across Spack releases

- Spack has historically made these compilers available to every package
  - A compiler was actually “something that supports c + cxx + fortran + f77”
  - Made for a lot of special cases
  - Also makes for duplication of purely interpreted packages (e.g. python)

We are releasing experimental alpha versions as we approach the 1.0 release

Release schedule:

- Releases are continuously rebased on develop
- Include compilers as dependencies
- Anticipate merging this to develop in February

- v.1.0.0 will be published before ISC'25

Prereleases so far

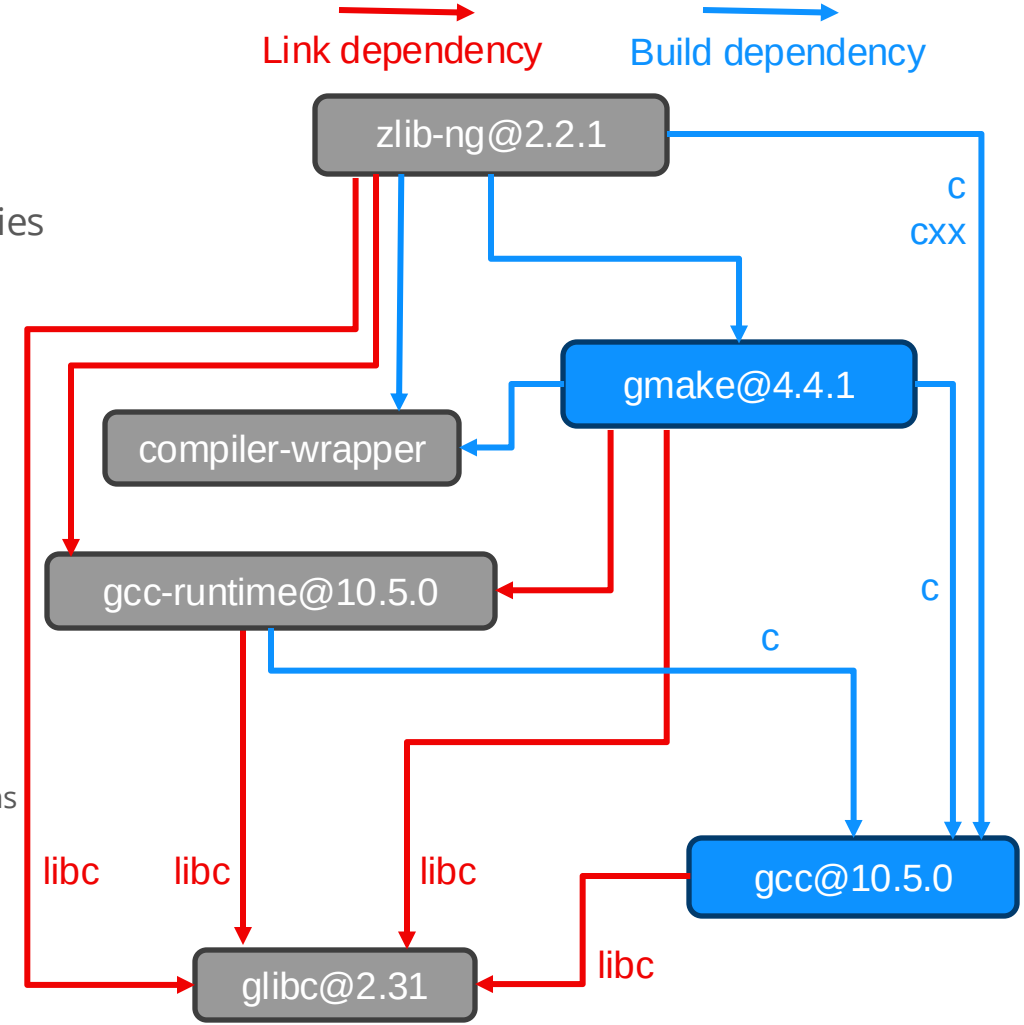
v1.0.0-alpha1	**2024-11-21**
v1.0.0-alpha2	**2024-12-09**
v1.0.0-alpha3	**2025-01-13**

HPSF
HIGH PERFORMANCE SOFTWARE FOUNDATION

Visit github.com/spack/spack/releases

Compiler Dependencies

- Compilers are now proper build dependencies
- Runtime libraries modeled as packages
- gcc-runtime node is injected as a link dependency by gcc
- packages depend on c, cxx, fortran virtuals, which are satisfied by gcc node
- glibc is an automatically detected external
  - Injected as a `libc` virtual dependency
  - Does not require user configuration
  - Will eventually be able to choose implementations (e.g., musl)



HPSF
HIGH PERFORMANCE SOFTWARE FOUNDATION

Configuring compilers in Spack v1.*

Spack v0.x

compilers.yaml

```
compilers:
  - compiler:
      spec: gcc@12.3.1
      paths:
         c: /usr/bin/gcc
         cxx: /usr/bin/g++
         fc: /usr/bin/gfortran
      modules: [...]
```

Spack v1.x

packages.yaml

```
packages:
  gcc:
    externals:
    - spec: gcc@12.3.1+binutils
      prefix: /usr
      extra_attributes:
        compilers:
          c: /usr/bin/gcc
          cxx: /usr/bin/g++
          fc: /usr/bin/gfortran
        modules: [...]
```

- We will provide a tool for migrating configuration
- We will still support *reading* the old configuration until at *least* v1.1
- All fields from `compilers.yaml` are supported in `extra_attributes`

Breaking changes 

1. It is no longer safe to assume every node has a compiler.
   a. The tokens `{compiler}`, `{compiler.version}`, and `{compiler.name}` in `Spec.format` expand to `none` if a Spec does not depend on C, C++, or Fortran.
   b. `spec.compiler` will default to the c compiler if present, else cxx, else fortran for backwards compatibility.
   c. The new default install tree projection is `{architecture.platform}/{architecture.target}/{name}-{version}-{hash}`

2. The syntax `spec["name"]` will only search link/run dependencies and *direct* build dependencies.
   - Previously, this would find deep, transitive deps, which was almost always the wrong behavior.
   - You can still hop around in the graph, e.g. `spec["cmake"]["bzip2"]` will find cmake's link dependency

3. The % sigil in specs means "direct build dependency".
   - Can now say: `foo %cmake@3.26 ^bar %cmake@3.31`
   - ^ dependencies are unified, % dependencies are not

More on direct dependencies with %

- You could previously write:

```
pkg %gcc +foo     # +foo would associate with pkg, not gcc – will error in 1.0
```

- Now you'll need to write:

```
pkg +foo %gcc     # +foo associates with pkg
```

- We want these to be symmetric:

```
pkg +foo %dep +bar  # `pkg +foo` depends on `dep +bar` directly
pkg +foo ^dep +bar  # `pkg +foo` depends on `dep +bar` directly or transitively
```

- spack style --spec-strings --fix can remedy this automatically
  - Fixes YAML files, scripts, package.py files
  - Alternative was to have a very hard-to-explain syntax – we surveyed users and they decided it was better to break a bit than to explaining the subtleties of the first 10 years of Spack forever

Step 2: Splitting out the packages

- Spack is 2 things:
  - Core tool
  - 8,400+ package.py files

- Coupling between core and packages is tight in some places:
  1. Package base classes for using build systems are in core (cmake, autotools, etc.)
  2. Compiler wrappers used to inject flags and RPATHs are in core
  3. Package files are used after installation, e.g., at load/unload time
     - Leads to drift between old installations and package files
  4. Packages *live* in Spack's GitHub repository -- not easy to separate

We reducing coupling between packages and core

1. Build system classes are moving *into* the package repository
   - Need to provide a way to include “utility” code from package.py files
2. Compiler wrappers will *become* a package
   - Also improves build provenance and reproducibility
3. Generate shell code for environment changes *at install time*
   - Bakes load/unload logic into installations and binary packages
   - Removes need for package.py files to remember all past versions
4. Spack packages will live in a separate GitHub repository
   - Need Spack to bootstrap this new repository
   - Will need to download automatically on first install

Some complexities left to navigate

- Compiler wrappers have already become their own package
  - Now injected by compilers
  - Still some coupling with the build environment
    - Spack sets variables to control RPATH flags
- In some cases Spack still knows compiler and runtime library names
  - A few optimizations in the solver know about, e.g., gcc-runtime, intel runtime, etc.
  - Working to fully generify this without losing solver performance
- Some parts of our tests rely on builtin packages
  - May need to mock these, or ensure that tests auto-checkout builtin repo

v1.0 Release plan

- This week:
  - Merge compiler dependencies
  - Start fielding bug reports on develop
- April-May:
  - Split out the builtin package repository
  - Ensure bootstrapping / repo cloning is smooth
- June
  - Release v1.0 at the Spack BOF at ISC 2025
- Likely to be done after v1.0 (longtime RIKEN ask)
  - First solver capabilities for cross-compiles
  - Think about adapting package builds for cross-compiles
    - Easy for some (cmake)
    - Harder for others (autotools or autotools-like)

HPSF
HIGH PERFORMANCE SOFTWARE FOUNDATION

https://bit.ly/spack-changes for more