



Spack Community BoF

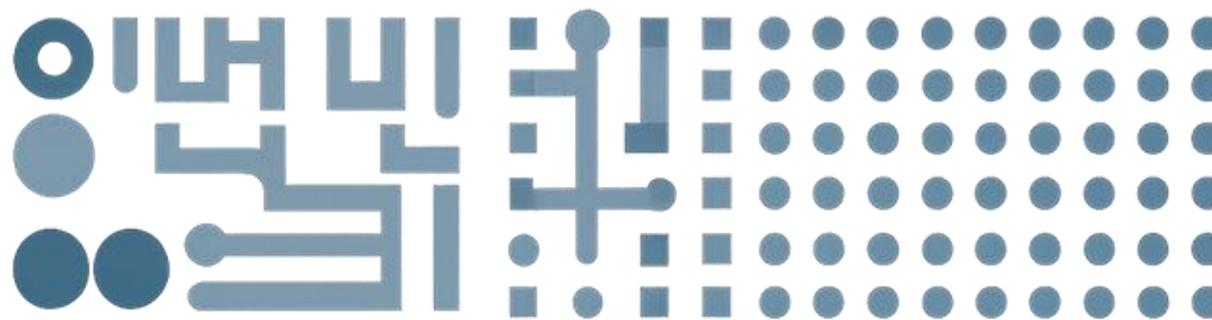
CASS Community BOF days 2026

BOF Days

February 10 - 12, 2026

<https://cass.community>

The
Consortium
for the
Advancement
of Scientific
Software
(CASS)



CASS



BOF DAYS

<https://cass.community/news/2026-02-10-cass-bof-days.html>

- **Inward-facing activities:** Strengthening software products
 - Improve development practices, sustainability, quality, and trustworthiness
 - Enhance user experience and integration within the broader ecosystem
- **Outward-facing activities:** Community engagement and discovery
 - Curate and evolve the software portfolio
 - Help teams connect with and grow their user communities
 - Enable the broader community to discover and adopt useful software

CASS Members

CORSA
Partnering with foundations to provide sustainable pathways for scientific software

FASTMATH
Stewardship, advancement, and integration for math and ML/AI packages

PESO
Stewarding, evolving and integrating a cohesive ecosystem for DOE software

RAPIDS
Stewardship, advancement, and integration for data, visualization and ML/AI packages

S4PST
Stewardship, advancement and engagement for programming systems

STEP
Stewardship, advancement of software tools for understanding performance and behavior

Sponsored by the Department of Energy, Office of Advanced Scientific Computing Research

Engage with CASS!

- Learn about CASS:
 - <https://cass.community/about/>
- Join the CASS Announcement list (low-volume):
 - <http://eepurl.com/iRiSnY>
- Find out more about our **software products**
 - Catalog: <https://cass.community/software/>
 - Collected as part of the [Extreme-Scale Scientific Software Stack](#) (E4S)
- Participate in **CASS Working Groups**
 - Impact Framework, Integration, Metrics, Software Ecosystem, User-Developer Experience, Workforce
 - <https://cass.community/working-groups/>

Spack v1.0 was released in July!

- We wanted:

- ✓ 2020 New ASP-based concretizer
- ✓ 2021 Reuse of existing installations
- ✓ 2022 Stable production CI
- ✓ 2022 Stable binary cache
- ✓ 2025 Compiler dependencies
- ✓ 2025 Separate builtin repo
- ✓ 2025 Stable package API

- v1.0:

- Changes the dependency model for compilers
- Enables users to use entirely custom packages
- Improves reproducibility
- Improves stability 🍀

- This is the largest change to Spack... ever.

How do we handle this?

- We want to:
 - Build build dependencies with the "easy" compilers
 - Build rest of DAG (the link/run dependencies) with the fancy compiler
- Works well for porting most scientific codes
 - Results in consistent compilers within processes
- What we actually do is run the concretizer separately for the pure build dependencies and the link dependencies
 - If something is shared between build and link, go with the link version.
- This is soon to be merged in.

spack install pkg1 %intel

Legend:
● Easy compiler
● Fancy compiler
B: build L: link R: run

Logos: Lawrence Livermore National Laboratory, github.com/spack, @spackpm, NSA

FOSDEM 18 .org



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



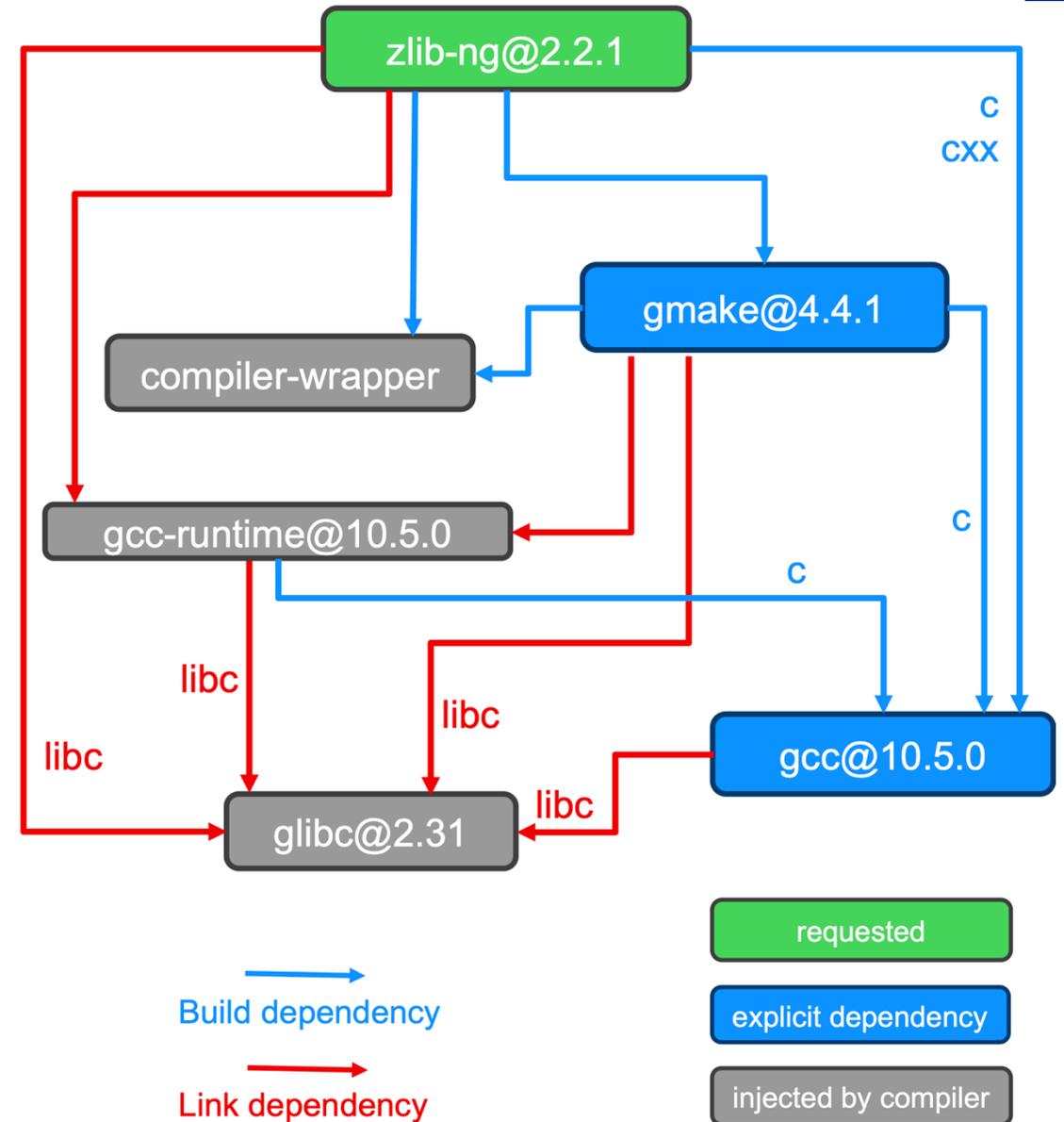
Spack v1.0 adds language dependencies

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

- 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)
- Required in 1.0 if you want to use c, cxx, or fortran
 - No-op in v0.23 and prior as we prepared for this feature

Compiler Dependencies

- Compilers are now build dependencies
- Runtime libraries modeled as packages
 - gcc-runtime is injected as 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)





Spack 1.x introduces *toolchains*

toolchains.yaml

```
toolchains:  
  clang_gfortran:  
    - spec: %c=clang  
      when: %c  
    - spec: %cxx=clang  
      when: %cxx  
    - spec: %fortran=gcc  
      when: %fortran  
    - spec: cflags="-O3 -g"  
    - spec: cxxflags="-O3 -g"  
    - spec: fflags="-O3 -g"
```

`spack install foo %clang_gfortran`

```
toolchains:  
  intel_mvapich2:  
    - spec: %c=intel-oneapi-compilers @2025.1.1  
      when: %c  
    - spec: %cxx=intel-oneapi-compilers @2025.1.1  
      when: %cxx  
    - spec: %fortran=intel-oneapi-compilers @2025.1.1  
      when: %fortran  
    - spec: %mpi=mvapich2 @2.3.7-1 +cuda  
      when: %mpi
```

`spack install foo %intel_mvapich2`

- Can lump many dependencies, flags together and use them with a single name
- Any spec in a toolchain can be *conditional*
 - Only apply when needed



Splitting the package repository

- Spack is two things:
 - Command line tool `spack`
 - Package repository with 8,600+ recipes
- Community wanted
 - package updates without tool changes (e.g. new bugs)
 - tool updates without package changes (reproducibility)
- But coupling between tool and packages was tight
 1. Package classes are in core: `CMakePackage`, `AutotoolsBuilder`, etc.
 2. Compiler wrapper was not a package until recently
 3. Packages *live* in Spack's GitHub repository with a long (git) history



Spack now has a Stable Package API

- Repositories define API version used
 - Versioned *per commit*
- Spack defines API version(s) supported
 - Will complain if a repo is too new
- Packages can only import from:
 - `spack.package`
 - Core Python
- Any 1.x Spack will support the same package API as all prior 1.x versions
 - Won't break package API unless we bump the major version

Spack Package API v2.2

This document describes the Spack Package API (`spack.package`), the stable interface for Spack package authors. It is assumed you have already read the [Spack Packaging Guide](#).

The Spack Package API is the *only* module from the Spack codebase considered public API. It re-exports essential functions and classes from various Spack modules, allowing package authors to import them directly from `spack.package` without needing to know Spack's internal structure.

Spack Package API Versioning

The current Package API version is v2.2, defined in `spack.package_api_version`. Notice that the Package API is versioned independently from Spack itself:

- The **minor version** is incremented when new functions or classes are exported from `spack.package`.
- The **major version** is incremented when functions or classes are removed or have breaking changes to their signatures (a rare occurrence).

This independent versioning allows package authors to utilize new Spack features without waiting for a new Spack release.

Compatibility between Spack and [package repositories](#) is managed as follows:

- Package repositories declare their minimum required Package API version in their `repo.yaml` file using the `api: vX.Y` format.
- Spack checks if the declared API version falls within its supported range, specifically between `spack.min_package_api_version` and `spack.package_api_version`.

Spack version 1.1.0.dev0 supports package repositories with a Package API version between v1.0 and v2.2, inclusive.

Spack Package API Reference

```
class spack.package.BaseBuilder(pkg: PackageBase) [source]
    Bases: object
    An interface for builders, without any phases defined. This class is exposed in the package API, so that packagers can create a single class to define setup_build_environment() and spack.phase_callbacks.run_before() and spack.phase_callbacks.run_after() callbacks that can be shared among different builders.
```

https://spack.rtdf.io/en/latest/package_api.html



You can now specify the package repo version in an environment or config

Pin a commit

```
spack:  
  repos:  
    builtin:  
      git: https://github.com/spack/spack-packages.git  
      commit: aec1e3051c0e9fc7ef8feadf766435d6f8921490
```

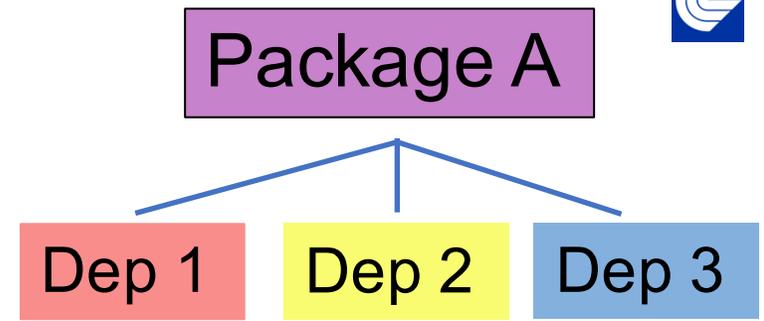
Work on a branch

```
spack:  
  repos:  
    builtin:  
      git: https://github.com/spack/spack-packages.git  
      destination: /path/to/clone/of/spack-packages  
      branch: develop
```



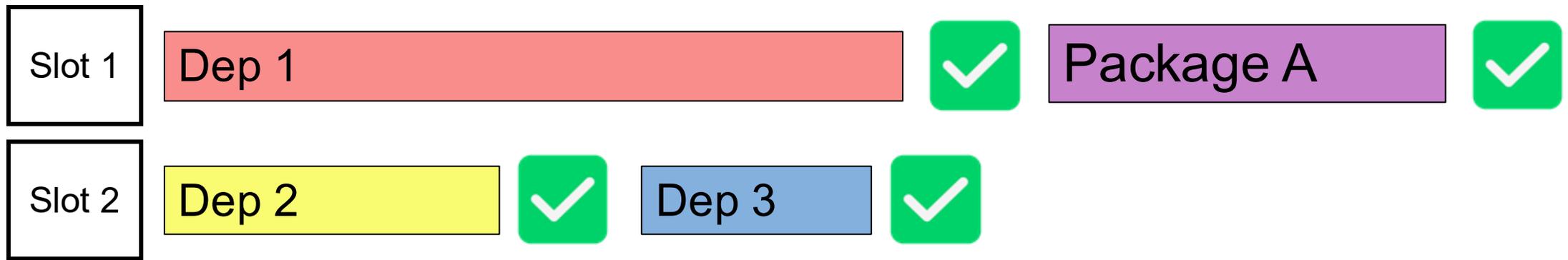
Bonus feature: Spack now supports concurrent builds!

- We sort of supported this already
 - But the user had to launch multiple spack processes
 - e.g., `srun -N 4 -n 16 spack install hdf5`



- Now spack handles on-node parallelism itself!
 - Spack now has a scheduler loop
 - Monitors dependencies, starts multiple processes, polls for completion
 - User can control max concurrent processes with '-p'

Queue:

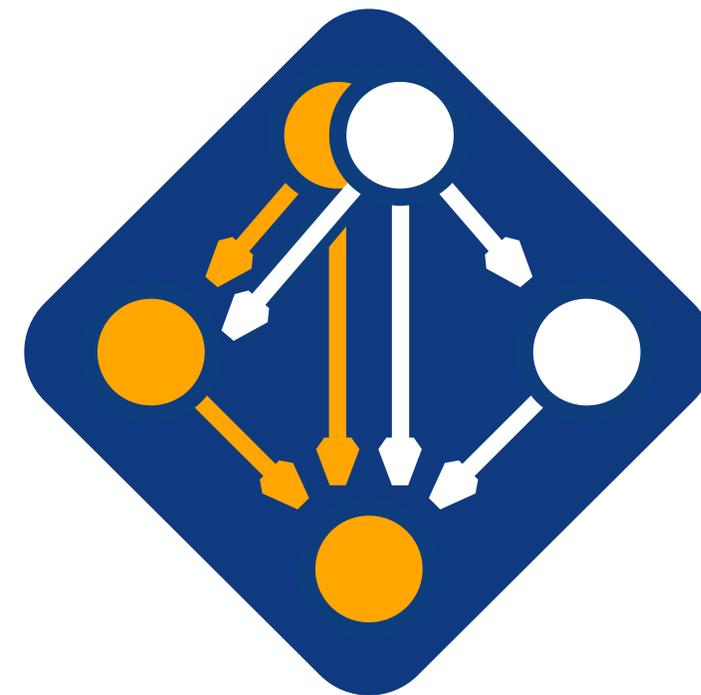




Spack v1.1.0 was released on November 14!

New Features:

1. More compiler control
 - Propagated compiler preferences with %%
 - Unmixing preferences
2. Customizable/overridable configuration scopes
3. Git fetching
4. External packages can have dependencies



Experimental Features:

5. New installer UI for parallel builds
6. Concretizer caching

 github.com/spack/spack

Full release notes: <https://github.com/spack/spack/releases/tag/v1.1.0>

Compiler unmixing

- v1.0, support for compilers as nodes:
 - Made it much easier to mix language compiler on different packages
 - Did not offer options to constrain compiler selection when needed.
- v1.1 allows you to disable this:

```
> spack config add concretizer:compiler_mixing:false
```

```
> spack spec chai%c=oneapi ^umpire%c=gcc  
==> Error: failed to concretize `chai %c=oneapi ^umpire %c=gcc` for the following reasons:  
1. Compiler mixing is disabled
```



Propagated Compiler Preferences with %%

- %% now specifies:
 - a direct dependency for a node, *and*
 - a preference for its transitive link/run dependencies
- Like prefer: in configuration
- Enables you to:
 - prefer: for defaults
 - require: for exceptions

```
spack:
  definitions:
    - compilers: ["%%llvm_gfortran", "%%c,cxx,fortran=gcc"]

  specs:
    - matrix:
      - [mpileaks, hdf5, trilinos]
      - [$compilers]

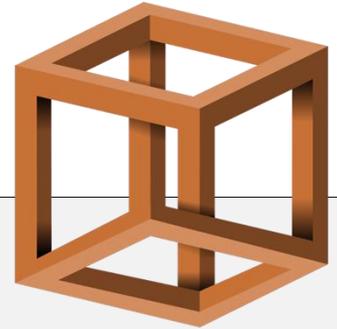
  toolchains:
    llvm_gfortran:
      - spec: "%c=llvm"
        when: "%c"
      - spec: "%cxx=llvm"
        when: "%cxx"
      - spec: "%fortran=gcc"
        when: "%fortran"

  packages:
    callpath:
      require:
        - "%c=gcc"

  concretizer:
    unify: false
```



Config scopes are now defined in configuration



- Much debate about ~/.spack
 - Can be a footgun for users
 - Hard for the owner of a spack instance or an env to have full control
- All config now stems from the spack instance
 - Instance (spack scope) has full control
 - User, site, system scopes are included
 - Environments can override with include:: []
- Envs and instance have full control!
 - Can customize how scopes behave

```
include:  
# user configuration scope  
- name: "user"  
  path: "~/.spack"  
  optional: true  
  when: "'SPACK_DISABLE_LOCAL_CONFIG' not in env'  
  
# site configuration scope  
- name: "site"  
  path: "$spack/etc/spack/site"  
  optional: true  
  
# system configuration scope  
- name: "system"  
  path: "/etc/spack"  
  optional: true  
  when: "'SPACK_DISABLE_LOCAL_CONFIG' not in env'
```

`$spack/etc/spack`



Including remote configuration from Git

```
include:  
- git: https://github.com/spack/spack-configs  
  branch: main  
  when: os == "centos7"  
  paths:  
  - USC/config/config.yaml  
  - USC/config/packages.yaml
```

- Much like remote repositories (added in v1.0)
- Can pin included configuration to a branch or commit



New Installer UI for parallel builds

```
[+] h6ce6ly gmake@4.4.1 /home/harmen/opt/linux-zen2/gmake-4.4.1-h6ce6lyjisykmc3u5s6mek32thio7pin
[+] uqjco4t util-macros@1.20.1 /home/harmen/opt/linux-zen2/util-macros-1.20.1-uqjco4tqxenptueobq5qbkggieysrp5u
Progress: 4/83 /: filter v: logs n/p: next/prev
[/] sjmydhp boost@1.88.0 staging
[/] ivlelhr libiconv@1.18 configure
[/] ilgps2i zstd@1.5.7 install
[/] gxytvc6 xz@5.6.3 configure
[/] ab3hpqc berkeley-db@18.1.40 configure
[+] 34yfszo libsigsegv@2.14 /home/harmen/opt/linux-zen2/libsigsegv-2.14-34yfszo5kr5sv7uv2gtk6iawgmq3a2pl
[/] xbvrehx gsl@2.8 configure
[/] 67pdlcf openblas@0.3.30 build
[+] 5tnnek5 libmd@1.1.0 /home/harmen/opt/linux-zen2/libmd-1.1.0-5tnnek5jh5gib7ntbv3vrhi2r5lr3imv
[/] tvo6sf2 libffi@3.5.2 build
[/] 6d7cdec pkgconf@2.5.1 build
[/] lonviug zlib-ng@2.2.4 build
```

- Multiple concurrent builds
- View live logs interactively
- Share a jobserver among builds
- Not yet feature complete

- Enable new installer with:

```
> spack config add config:installer:new
```

Spack v1.1.1 was released... yesterday!

Usability enhancements:

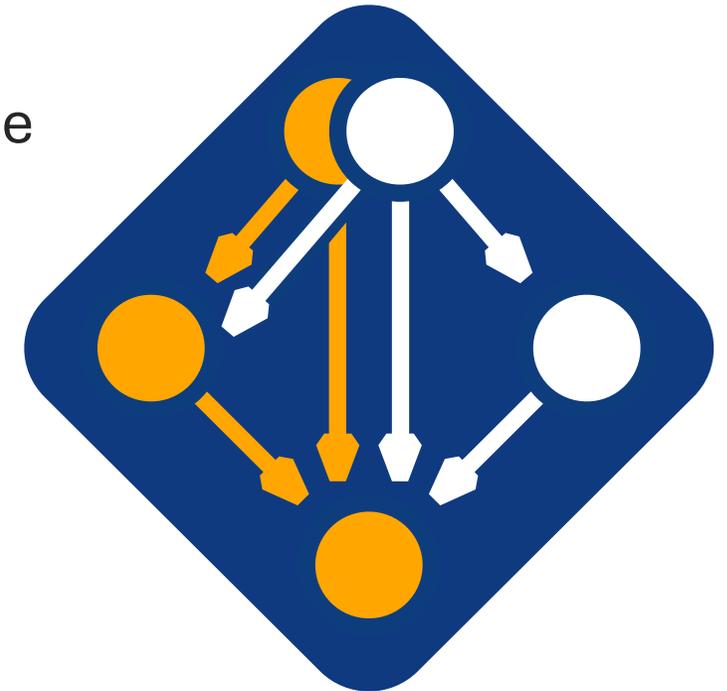
- Experimental new installer is more feature-complete
- Python 3.14 support
- Better errors for missing/deprecated versions
- Better spack info display

Major concretization performance improvements

- Optimized solver encoding
- Reduced grounding size for solver

Numerous bugfixes

Full release notes: <https://github.com/spack/spack/releases/tag/v1.1.1>



 github.com/spack/spack



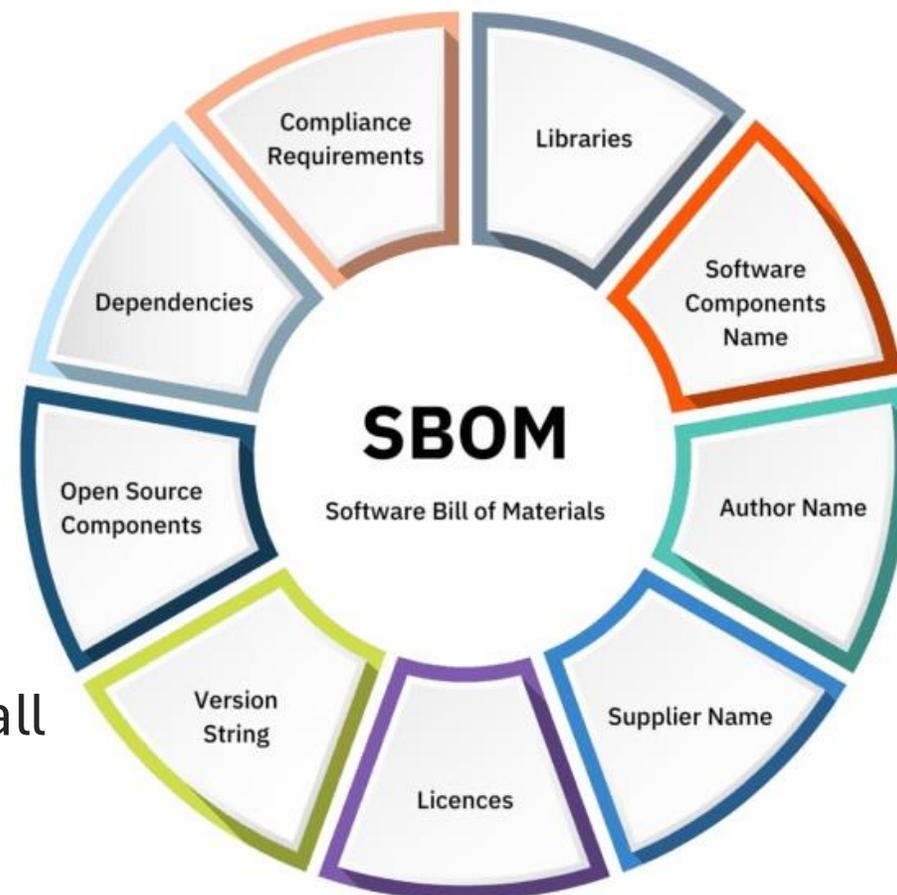
Roadmap for v1.2 – next Spring

- pip install spack
- SBOMs and more detailed provenance
- Error message improvements
- Numerous performance improvements
 - Faster load, activate
- Including environments in other environments
- Buildcache views
- Fully featured new installer/concretization cache



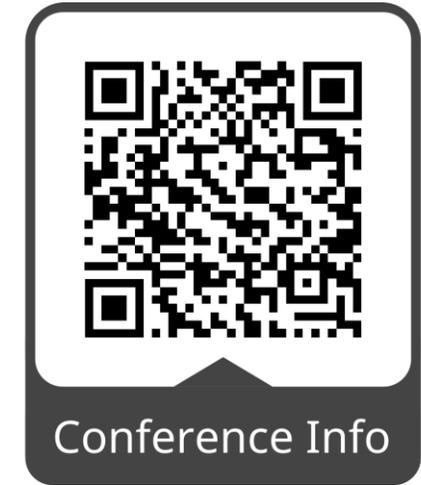
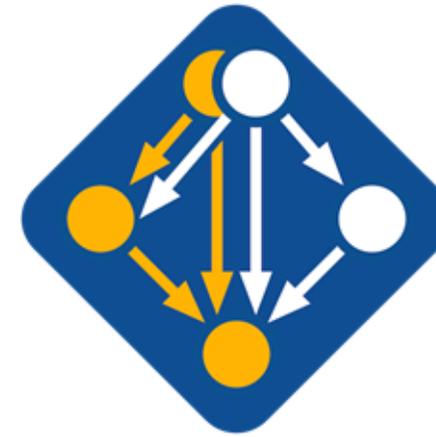
SBOMs can provide detailed provenance for packages

- Software Bill of Materials (SBOM)
 - Versions, hashes, organization of origin, etc.
 - Executive order in the last administration encouraged projects to add for security
- SBOMs enable supply chain security
 - Scanning installations for CVEs
 - Understanding outdated versions
 - Understanding where software came from
- Plan: automatically generate SBOMs with spack install
 - Expose SBOM location for scanners
 - Disclose vulnerabilities
 - Make DevSecOps easy for Spack users



Join the Spack community!

at the 2nd annual Spack user meeting



2 days of Spack user meeting

Call for presentatons will go out in the next couple weeks.

and online



spack.io