Spack: The Community's Road to HPSF and Version 1.0

ChapelCon 2025
Virtual Event
October 10, 2025

THE LINUX FOUNDATION

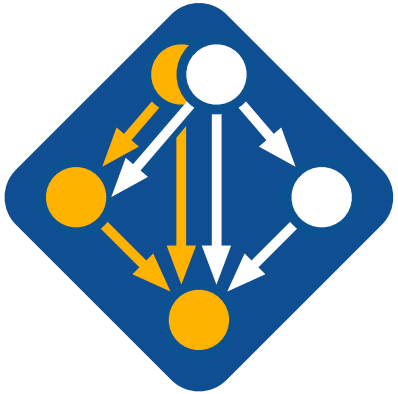
The most recent version of these slides can be found at:
https://spack-tutorial.readthedocs.io

What is Spack?

- Spack automates the build and installation of scientific software and its dependencies
- Packages are *parameterized,* so that users can easily tweak and tune configuration

Simple syntax enables complex installs

```
$ spack install hdf5@1.10.5                    $ spack install 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
```



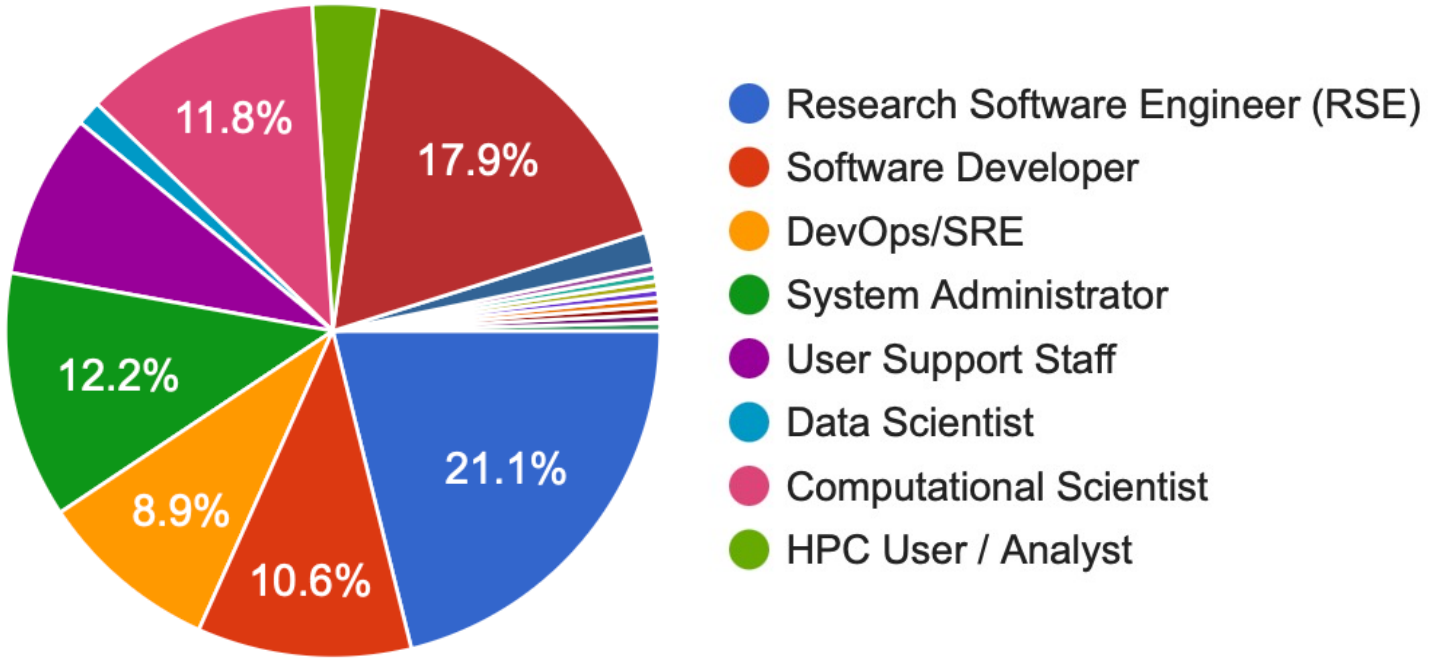
- Ease of use of mainstream tools, with flexibility needed for HPC
  - Enabled by dependency solving via *Clingo* Answer Set Programming (ASP) library
- In addition to CLI, Spack also:
  - Generates (but does **not** require) *modules*
  - Allows conda/virtualenv-like *environments*
  - Provides many devops features (CI, container generation, more)

github.com/spack/spack

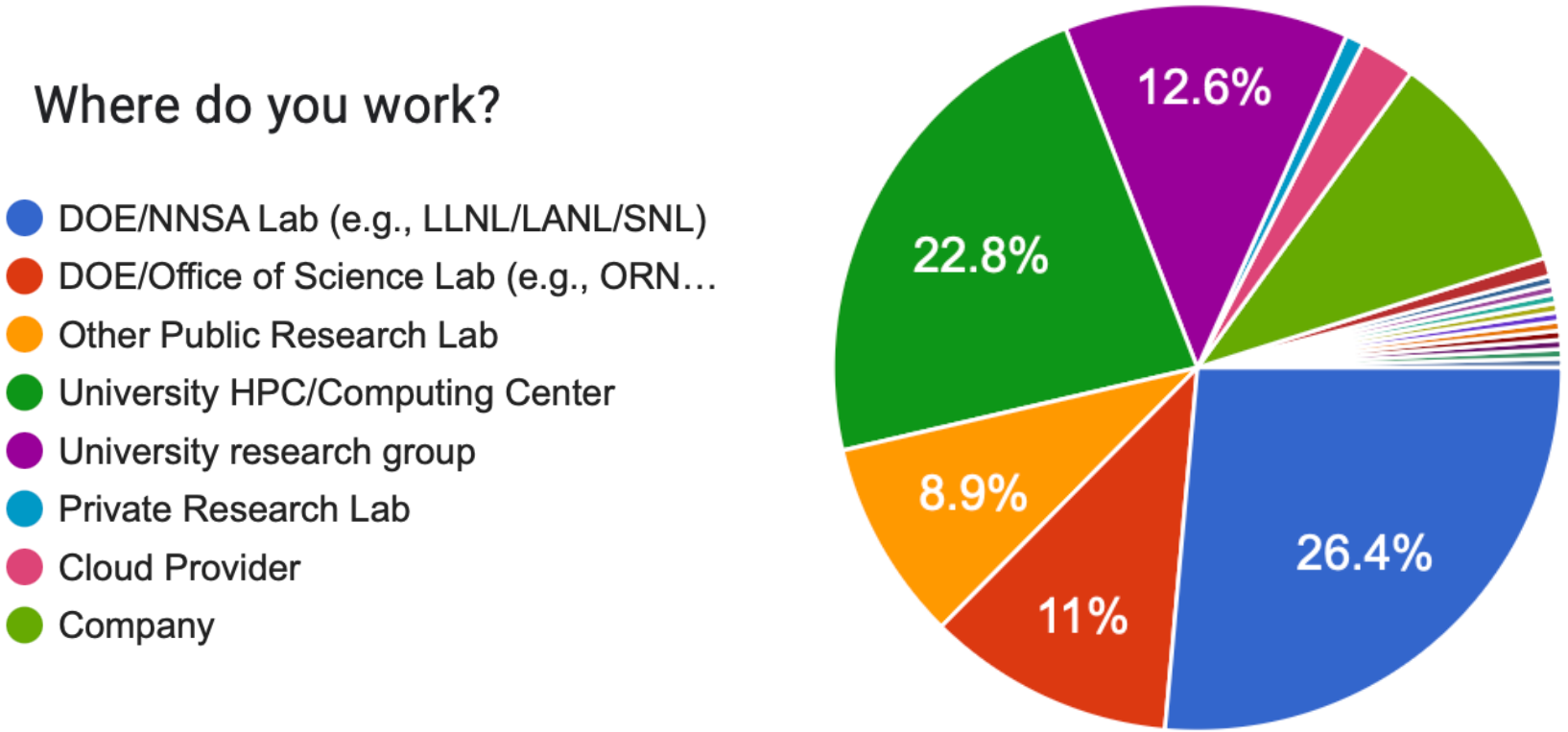
Spack's user base is quite diverse

What kind of user are you?

246 responses

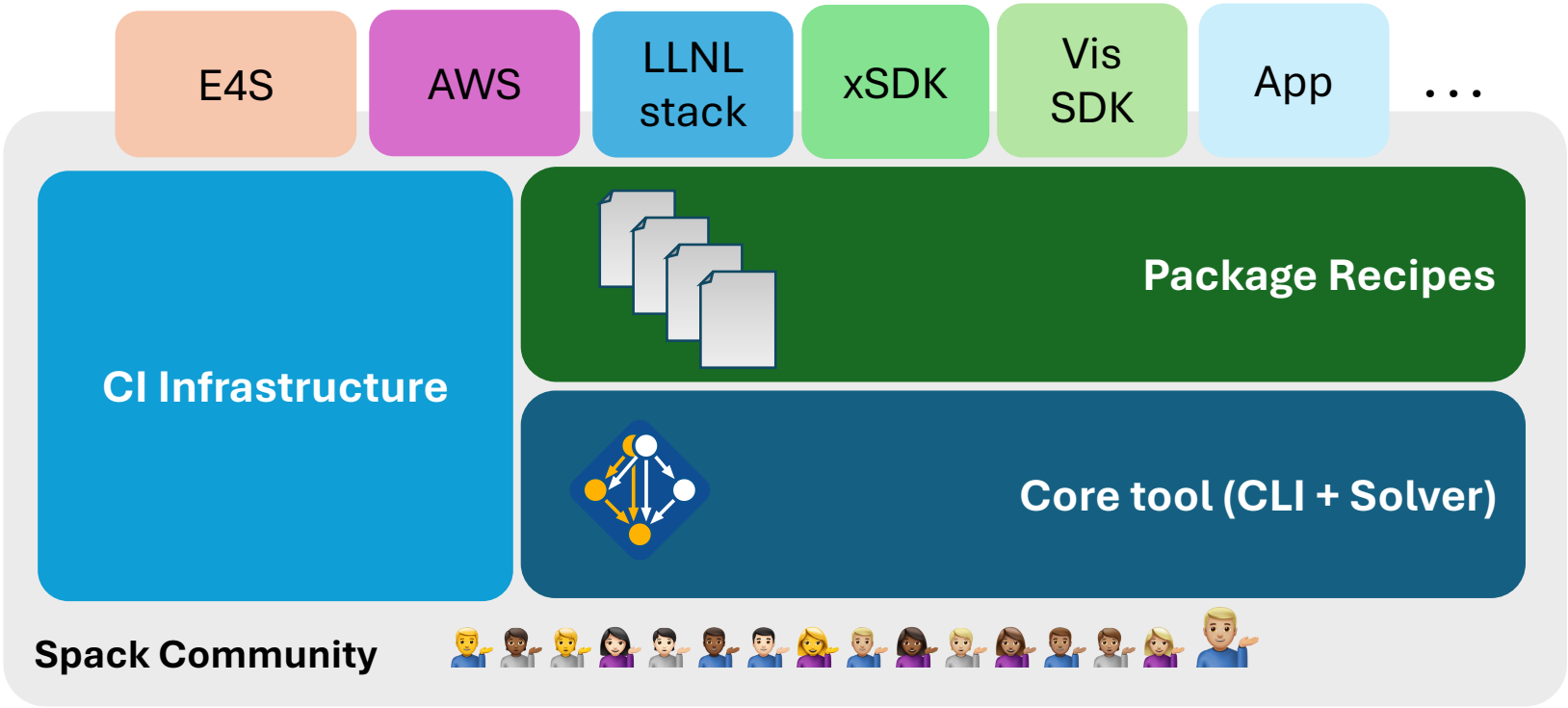


Workplace distribution is also diverse, with industrial use growing



2025

What does the Spack project look like?

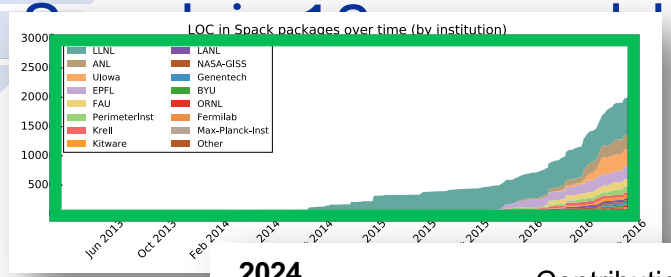


What are the goals of the Spack project?

- Build three things:
  - Core tool
  - CI infrastructure (scalable service)
  - Package ecosystem

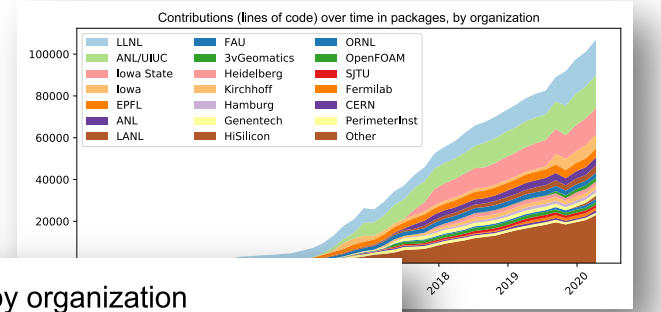
- Can’t sustain the package ecosystem ourselves, so we try to:
  1. Grow adoption
  2. Turn adopters into contributors
  3. Find new areas for adoption
  4. Turn *those* adopters into contributors
  5. Retain contributors!

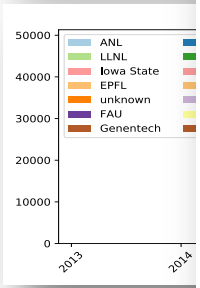
From the start, we’ve focused on growing the community

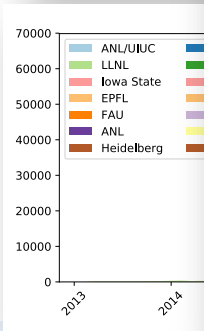


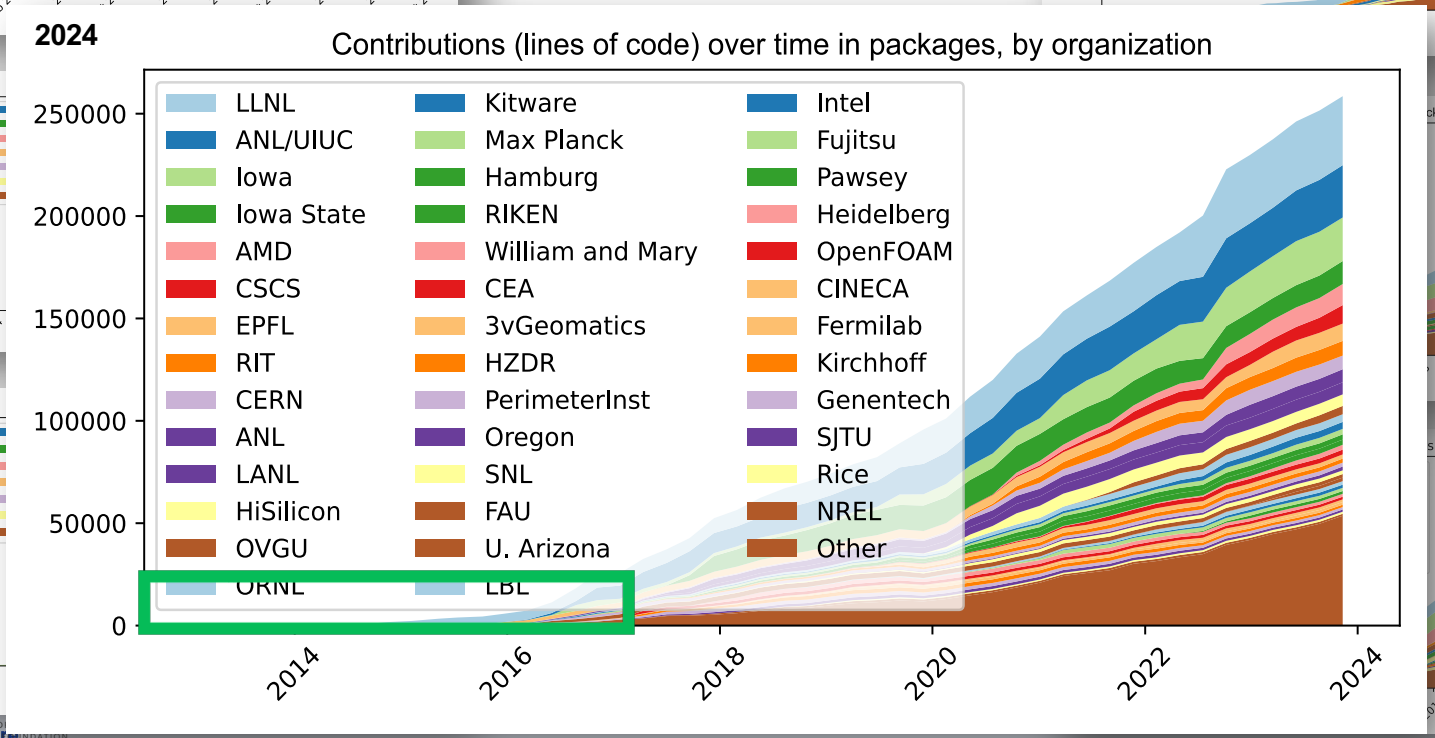
2016

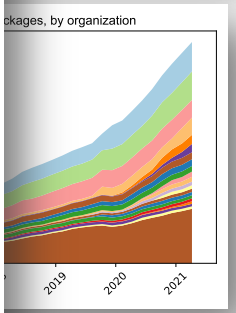
Fall 2020

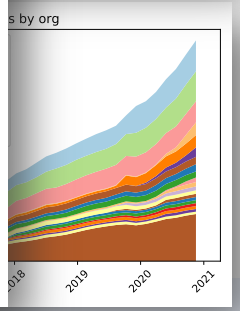




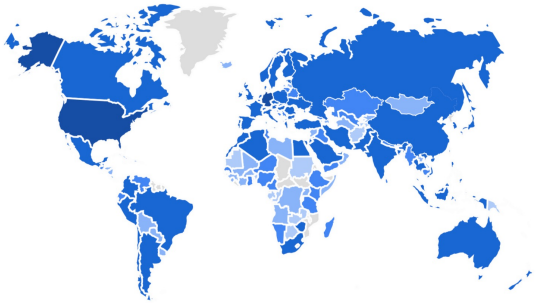








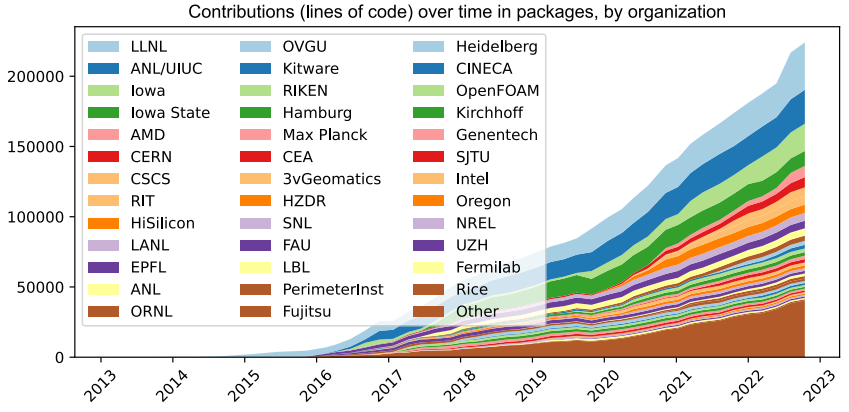
ECP is over, but Spack continues to grow



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 8,400 software packages
Over 1,500 contributors



Most package contributions are ***not*** from DOE
But they help sustain the DOE ecosystem!

How do we grow the project from here?

Spack and HPSF

THE LINUX FOUNDATION

We started conversations with Linux Foundation in December 2021, and talked through mid-2022

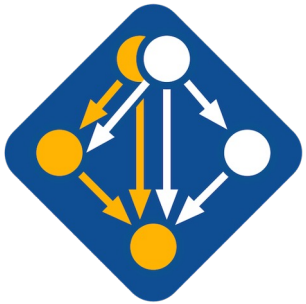
- We wanted:
  - A neutral project home
    - To encourage more participation in the project
  - A way to fund project activities:
    - More continuous integration resources
    - User meetings, Slack, etc.
- Talked to LF onboarding team
- Learned about LF's basic requirements:
  - Technical charter
  - Open Governance
  - Trademark assignment

We joined forces with Kokkos to start a larger umbrella, which eventually became HPSF

- Spack and Kokkos were two of the most adopted projects during ECP
- Enable performance portability at different levels
  - Spack: build level
  - Kokkos: application/runtime level
- Goals:
  - Leverage proven track record of community building
  - Leverage industry and labs' familiarity
  - Get more projects on board to build an umbrella organization

What does it mean to be a Linux Foundation Project?

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



With HPSF, we've formalized our governance with the 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 Governance in a nutshell

HPSF

Governing Board (GB)

GB Committees

Events & Training

Outreach

Technical Advisory Council (TAC)

Working Groups

CI / CD

Benchmarking

Binaries

Software Security

Safety and Security

Software Stacks

Technical Projects

Spack

Charliecloud

AMReX

HPCToolkit

APPTAINER

E4S

TRILINOS

kokkos

ENVIRONMENT MODULES

OpenCHAMI

CHAPEL

VISKORES

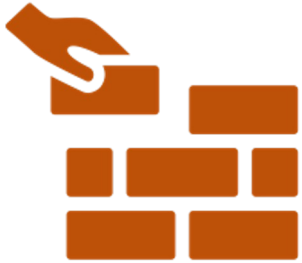
WarpX

HPX

What does HPSF do for projects?

HPSF

- **Build a community** for your project grounded in **neutral open source governance**
- **Find synergies on common needs** with other HPSF projects
  - CI, software engineering best practices, community upkeep, marketing
- **Share your knowledge** with and learn from other high performance software projects
- **Participate in working groups** that aim to bring HPC open source software to the wider computing world



The TAC has established a project lifecycle as a path to sustainability

HPSF CONFERENCE 2025

Emerging

- Committed to open governance
- Working towards best practices
- Important projects for the HPC ecosystem

Established

- Wide usage by at least 3 orgs of sufficient size and scope
- Steady commits from at least one organization
- Robust development practices

Core

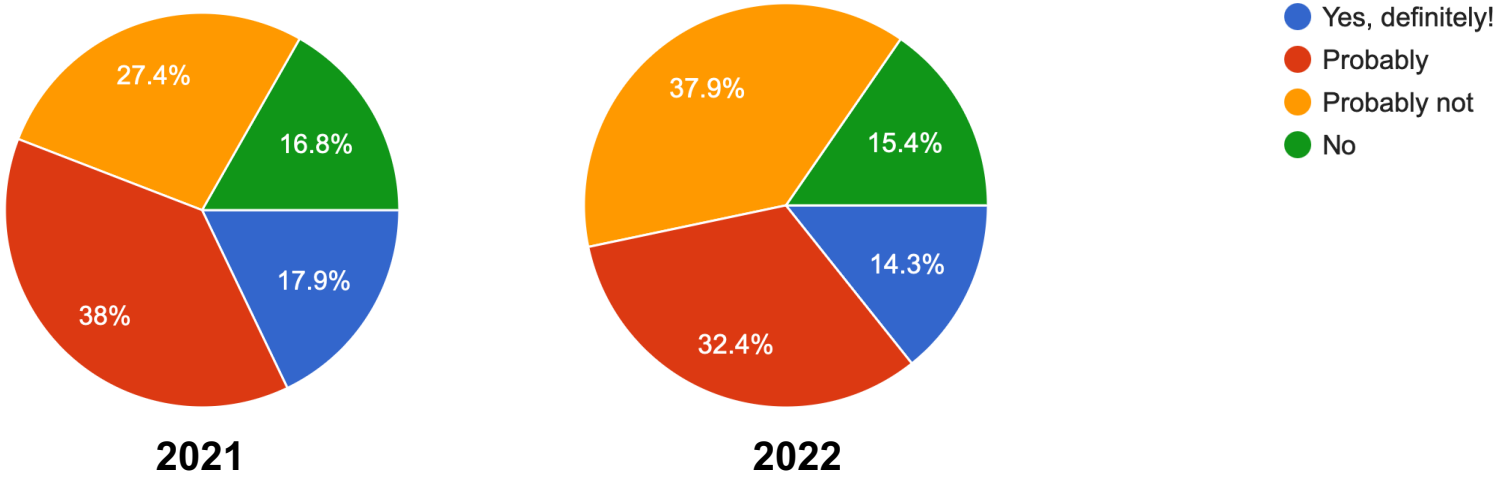
- Used commonly in production environments
- Steady commits from *more* than one organization
- Large, well-established project communities
- Sustainable cycle of development and maintenance

How does the Spack community like HPSF?

- We ran the **2025 Spack User Survey** from early April to early May
- We got the most responses ever
  - 2025: 246 responses
  - 2022: 182 responses
  - 2021: 179 responses
  - 2020: 169 responses
- Maybe the community is growing?
  - Or maybe we just waited a while between surveys. Or both

We asked the community *before* we joined HPSF

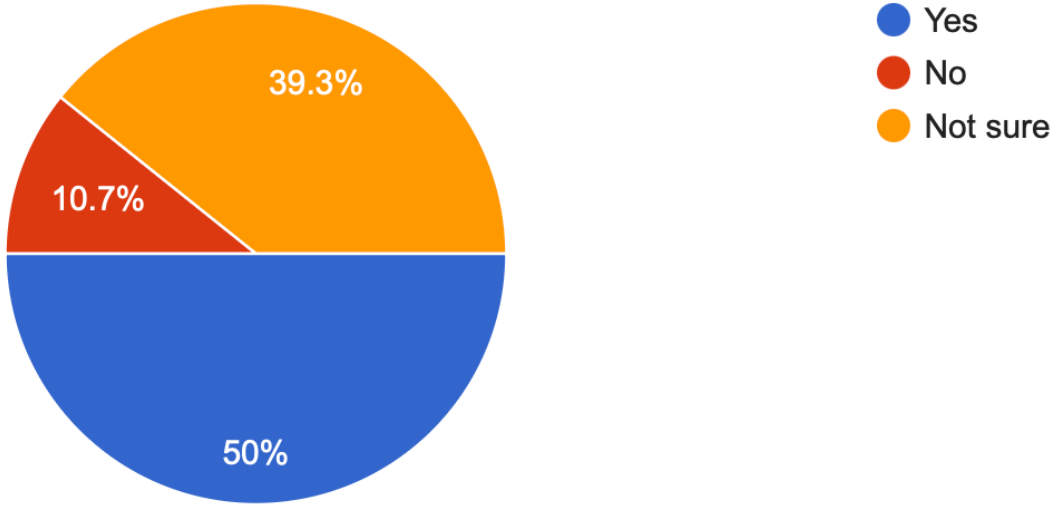
If there were a software foundation (ala Linux Foundation or CNCF) around Spack would it make you more likely to use/recommend Spack?



Response to LF/HPSF seems positive

Has Spack's transition into Linux Foundation / HPSF given you more confidence in the project?

242 responses



The Road to Spack v1.0

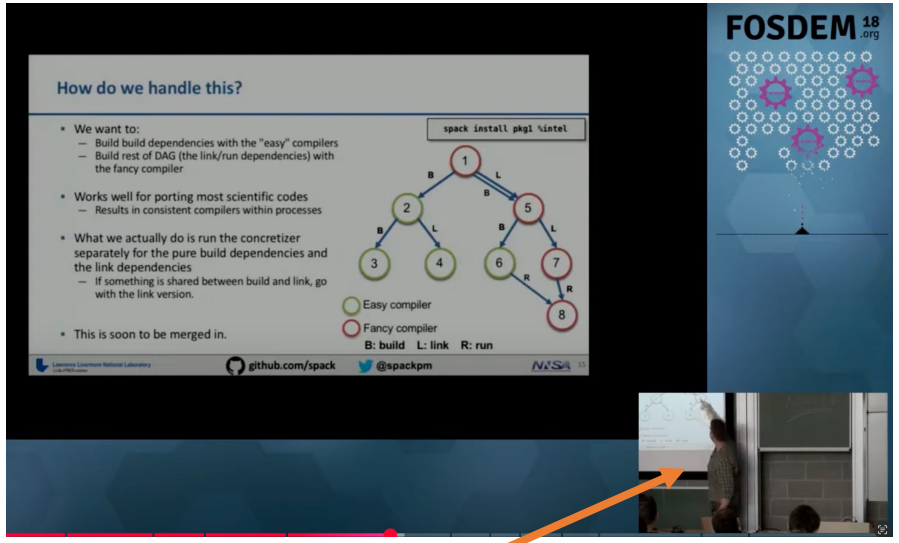
THE LINUX FOUNDATION

The road to v1.0 has been long

- 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.



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

Spack packages use a *lot* of (declarative) conditional logic

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

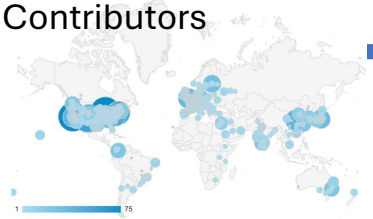
constraints on `cuda` version

compiler support for `x86_64` and `ppc64le`

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

First challenge: we needed a new concretizer to model the expressiveness of the DSL

Contributors



- new versions
- new dependencies
- new constraints

package.py repository

This part is NP-hard!

concretizer

spack developers

yaml default config packages.yaml

admins, users

yaml local preferences config packages.yaml

users

yaml local environment config spack.yaml

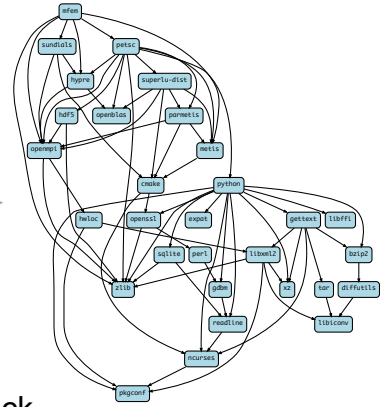
users

Command line constraints

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

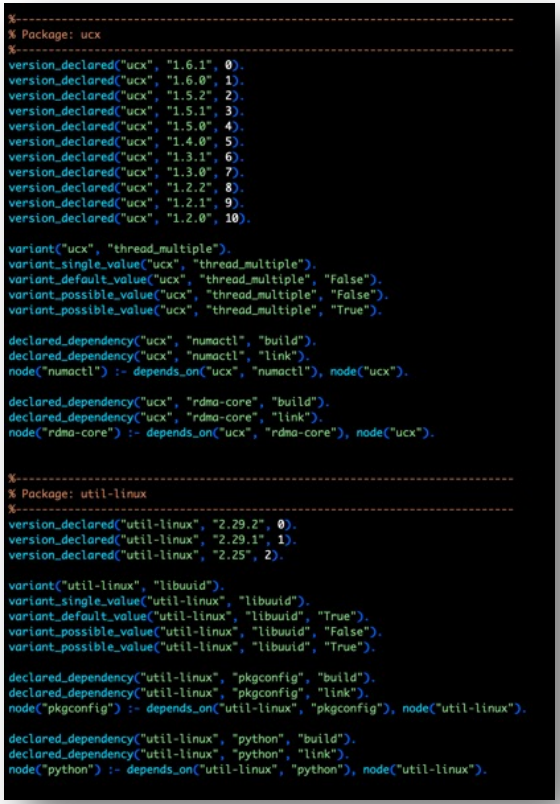
Concrete spec is fully constrained and can be built.

Is stored in spack.lock file after solve.



We reimplemented Spack's concretizer using *Answer Set Programming (ASP)*

- Originally a greedy, custom Python algorithm
- ASP is a *declarative* programming paradigm
  - Looks like Prolog
  - Built around modern CDCL SAT solver techniques
- ASP program has 2 parts:
  1. Large list of facts generated from recipes (problem instance)
  2. Small logic program (~700 lines of ASP code)
- Algorithm is conceptually simpler:
  - Generate facts for all possible dependencies
  - Send facts and our logic program to the solver
  - Read results and rebuild the resolved DAG
- Using **Clingo**, the Potassco grounder/solver package

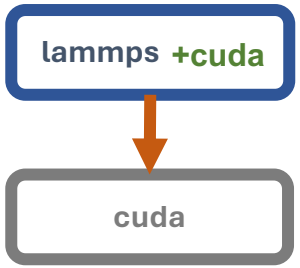


Some facts for HDF5 package

ASP looks like Prolog but is converted to SAT with optimization

Facts describe the graph

```
node("lammps").
node("cuda").
variant_value("lammps", "cuda", "True").
depends_on("lammps", "cuda").
```



First-order rules (with variables) describe how to resolve nodes and metadata

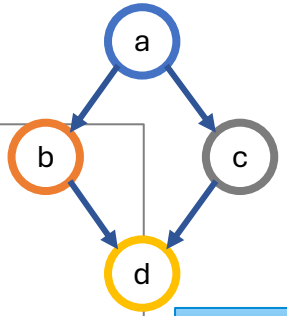
```
node(Dependency) :- node(Package), depends_on(Package, Dependency).
```

```
node("mpi")
```

```
node("hdf5").
depends_on("hdf5", "mpi").
```

Ground Rule

Grounding converts a first-order logic program into a propositional logic program, which can be solved.




```
depends_on(a, b).
depends_on(a, c).
depends_on(b, d).
depends_on(c, d).

node(Dep)
  :- node(Pkg),
     depends_on(Pkg, Dep).

% at least one is true
1 { node(a); node(b) }.
```

First-order Logic Program

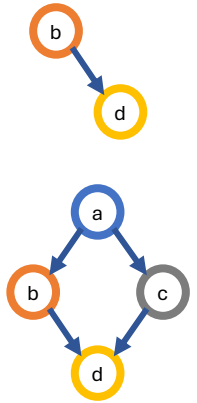
Ground

```
depends_on(a, b).
depends_on(a, c).
depends_on(b, d).
depends_on(c, d).

node(b) :- node(a).
node(c) :- node(a).
node(d) :- node(c).
node(d) :- node(b).

% at least one is true
1 { node(a); node(b) }.
```

Propositional Program

Solve

```
Answer 1:
node(b)
node(d)

Answer 2:
node(a)
node(b)
node(c)
node(d)
```



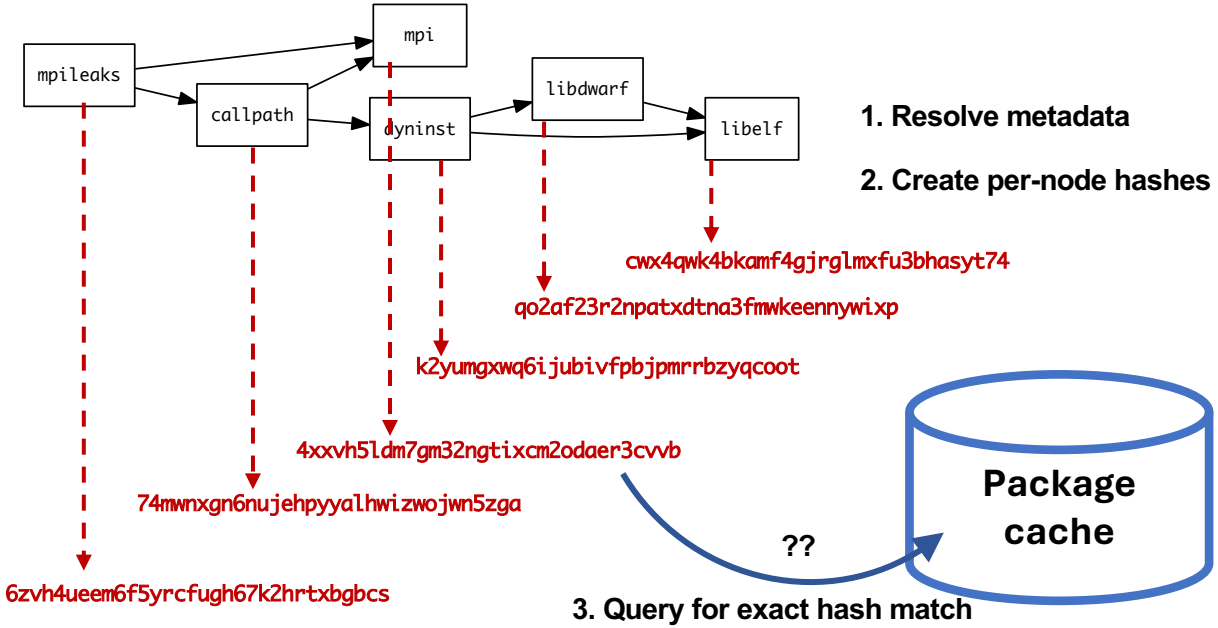
Stable Models (Answer Sets)

Answer 1: Only `node(b)` is true
Answer 2: Both `node(a)` and `node(b)` are true

ASP searches for *stable models* of the input program

- Stable models are also called ***answer sets***
- A ***stable model*** (loosely) is a set of true atoms that can be deduced from the inputs, where every rule is idempotent.
  - Similar to fixpoints
  - Put more simply: a *set of atoms where all your rules are true!*
- Unlike Prolog:
  - Stable models contain everything that can be derived (vs. just querying values)
  - Good ways to do optimization to select the “best” stable model
  - ASP is guaranteed to complete!

Second challenge: Spack's original concretizer did not reuse existing installations



- 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) was enabled by ASP

- --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","lwatuuysmwkhuahrncywvn77icdhs6mn").
imposed_constraint("lwatuuysmwkhuahrncywvn77icdhs6mn","node","openssl").
imposed_constraint("lwatuuysmwkhuahrncywvn77icdhs6mn","version","openssl","1.1.1g").
imposed_constraint("lwatuuysmwkhuahrncywvn77icdhs6mn","node_platform_set","openssl","darwin").
imposed_constraint("lwatuuysmwkhuahrncywvn77icdhs6mn","node_os_set","openssl","catalina").
imposed_constraint("lwatuuysmwkhuahrncywvn77icdhs6mn","node_target_set","openssl","x86_64").
imposed_constraint("lwatuuysmwkhuahrncywvn77icdhs6mn","variant_set","openssl","systemcerts","True").
imposed_constraint("lwatuuysmwkhuahrncywvn77icdhs6mn","node_compiler_set","openssl","apple-clang").
imposed_constraint("lwatuuysmwkhuahrncywvn77icdhs6mn","node_compiler_version_set","openssl","apple-clang","12.0.0").
imposed_constraint("lwatuuysmwkhuahrncywvn77icdhs6mn","concrete","openssl").
imposed_constraint("lwatuuysmwkhuahrncywvn77icdhs6mn","depends_on","openssl","zlib","build").
imposed_constraint("lwatuuysmwkhuahrncywvn77icdhs6mn","depends_on","openssl","zlib","link").
imposed_constraint("lwatuuysmwkhuahrncywvn77icdhs6mn","hash","zlib","x2anksgssxsxa7pcnhzg5k3dhgacglze").
```

Minimizing 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) }.
```

There's more to it than this, but you get the idea…

With and without --reuse optimization

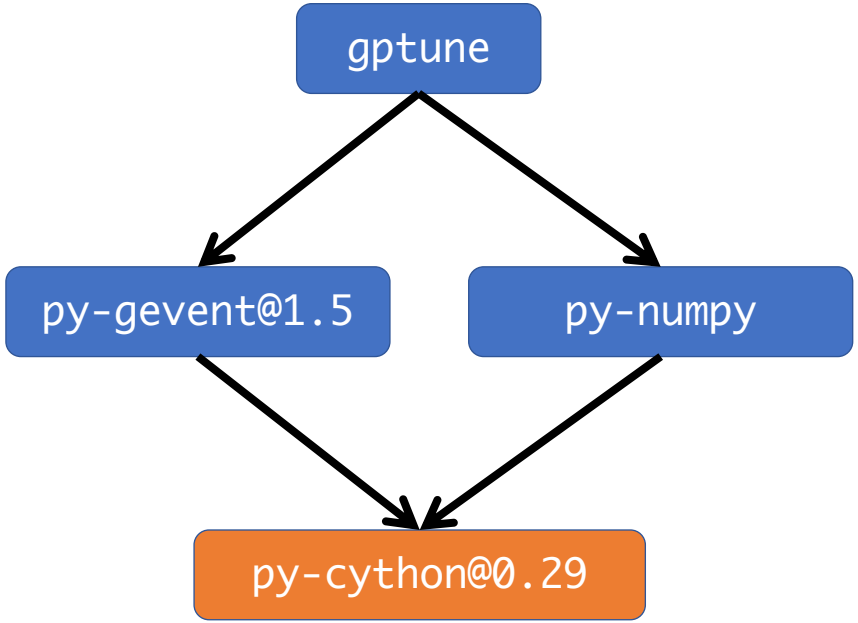
```
(spackle):solver> spack solve -Il hdf5
==> Best of 9 considered solutions.
==> Optimization Criteria:
  Priority  Criterion                                                  Installed  ToBuild
  1         number of packages to build (vs. reuse)                            -       20
  2         deprecated 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

  -   zzngfs3  hdf5@1.10.7%apple-clang@13.0.0~cxx~fortran~hl~ipo~java+mpi+shared~szip~threadsafe+tools api=default b
  -   nsylovq      ^cmake@3.21.4%apple-clang@13.0.0~doc+ncurses+openssl+ownlibs~qt build_type=Release arch=darwin-bi
  -   xdbaqeo          ^ncurses@6.2%apple-clang@13.0.0~symlinks+termlib abi=none arch=darwin-bigsur-skylake
  -   kfureok              ^pkgconf@1.8.0%apple-clang@13.0.0 arch=darwin-bigsur-skylake
  -   5ekd4ap          ^openssl@1.1.1l%apple-clang@13.0.0~docs certs=system arch=darwin-bigsur-skylake
  -   xz6a265              ^perl@5.34.0%apple-clang@13.0.0+cpanm+shared+threads arch=darwin-bigsur-skylake
  -   xgt3tls                  ^berkeley-db@18.1.40%apple-clang@13.0.0+cxx~docs+stl patches=b231fcc4d5cff05e5c3a4814
  -   65edjf6                  ^bzip2@1.0.8%apple-clang@13.0.0~debug~pic+shared arch=darwin-bigsur-skylake
  -   662adoo                      ^diffutils@3.8%apple-clang@13.0.0 arch=darwin-bigsur-skylake
  -   fu7tfsr                          ^libiconv@1.16%apple-clang@13.0.0 libs=shared,static arch=darwin-bigsur-skyla
  -   vjg67nd                  ^gdbm@1.19%apple-clang@13.0.0 arch=darwin-bigsur-skylake
  -   tjceldr                      ^readline@8.1%apple-clang@13.0.0 arch=darwin-bigsur-skylake
  -   xevvljj                  ^zlib@1.2.11%apple-clang@13.0.0+optimize+pic+shared arch=darwin-bigsur-skylake
  -   xelfobh      ^openmpi@4.1.1%apple-clang@13.0.0~atomics~cuda~cxx~cxx_exceptions+gpfs~internal-hwloc~java~legacy
  -   zruns75          ^hwloc@2.6.0%apple-clang@13.0.0~cairo~cuda~gl~libudev+libxml2~netloc~nvml~opencl~pci~rocm+sha
  -   ib4fnkf              ^libxml2@2.9.12%apple-clang@13.0.0~python arch=darwin-bigsur-skylake
  -   dwiv2ys                  ^xz@5.2.5%apple-clang@13.0.0~pic libs=shared,static arch=darwin-bigsur-skylake
  -   blitnbl          ^libevent@2.1.12%apple-clang@13.0.0+openssl arch=darwin-bigsur-skylake
  -   h7jalyu          ^openssh@8.7p1%apple-clang@13.0.0 arch=darwin-bigsur-skylake
  -   7v7bqx2              ^libedit@3.1-20210216%apple-clang@13.0.0 arch=darwin-bigsur-skylake
```

Pure hash-based reuse: all misses

```
(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

  -   yfkfnsp  hdf5@1.10.7%apple-clang@12.0.5~cxx~fortran~hl~ipo~java+mpi+shared~szip~threadsafe+tools api=default
  [+] zd4m26e      ^cmake@3.21.1%apple-clang@12.0.5~doc+ncurses+openssl+ownlibs~qt build_type=Release arch=darwin
  [+] 53i52xr          ^ncurses@6.2%apple-clang@12.0.5~symlinks+termlib abi=none arch=darwin-bigsur-skylake
  [+] us36bwr          ^openssl@1.1.1l%apple-clang@12.0.5~docs+systemcerts arch=darwin-bigsur-skylake
  [+] 74mwnxg              ^zlib@1.2.11%apple-clang@12.0.5+optimize+pic+shared arch=darwin-bigsur-skylake
  -   3ijfnel          ^openmpi@4.1.1%apple-clang@12.0.5~atomics~cuda~cxx~cxx_exceptions+gpfs~internal-hwloc~java~leg
  -   jxexyb7              ^hwloc@2.6.0%apple-clang@12.0.5~cairo~cuda~gl~libudev+libxml2~netloc~nvml~opencl~pci~rocm+
  [+] ckdn5zf                  ^libxml2@2.9.12%apple-clang@12.0.5~python arch=darwin-bigsur-skylake
  [+] k7auat3                      ^libiconv@1.16%apple-clang@12.0.5 libs=shared,static arch=darwin-bigsur-skylake
  [+] kzyumgx                      ^xz@5.2.5%apple-clang@12.0.5~pic libs=shared,static arch=darwin-bigsur-skylake
  [+] grgtlcd                  ^pkgconf@1.8.0%apple-clang@12.0.5 arch=darwin-bigsur-skylake
  -   nnc66ug              ^libevent@2.1.12%apple-clang@12.0.5+openssl arch=darwin-bigsur-skylake
  [+] 63xbksk              ^openssh@8.6p1%apple-clang@12.0.5 arch=darwin-bigsur-skylake
  [+] snhgldt                  ^libedit@3.1-20210216%apple-clang@12.0.5 arch=darwin-bigsur-skylake
  [+] qbkmtdd              ^perl@5.34.0%apple-clang@12.0.5+cpanm+shared+threads arch=darwin-bigsur-skylake
  [+] tnvkifs                  ^berkeley-db@18.1.40%apple-clang@12.0.5+cxx~docs+stl patches=b231fcc4d5cff05e5c3a4814f
  [+] 7d5woqt                  ^bzip2@1.0.8%apple-clang@12.0.5~debug~pic+shared arch=darwin-bigsur-skylake
  [+] vh6di3i                  ^gdbm@1.19%apple-clang@12.0.5 arch=darwin-bigsur-skylake
  [+] qgy3v4l                      ^readline@8.1%apple-clang@12.0.5 arch=darwin-bigsur-skylake
```

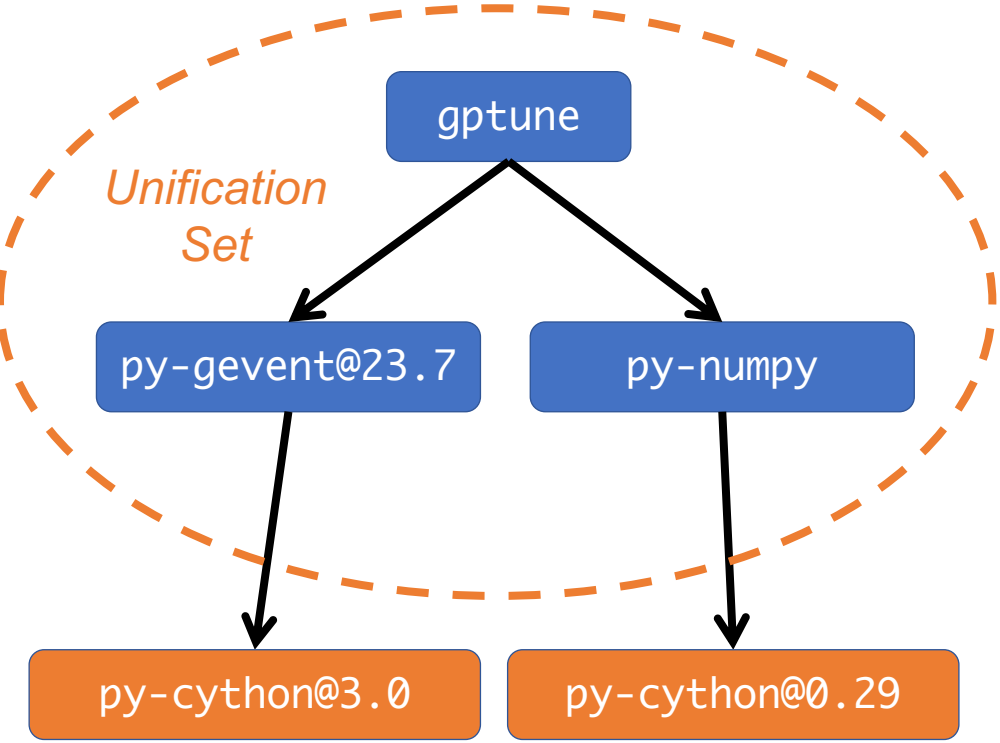
With reuse: 16 packages were reusable

Third challenge: we needed to allow multiple versions of build dependencies in the DAG



- Only one configuration per package allowed in the DAG
  - Ensures ABI compatibility but is too restrictive
- Needed to enable compiler mixing with compiler dependencies
- Also needed for Python ecosystem
  - In the example py-numpy needs to use py-cython@0.29 as a build tool
  - That enforces using an old py-gevent, because newer versions depend on py-cython@3.0 or greater

Objective: dependency splitting



- The constraint on build dependencies can be relaxed, without compromising the ABI compatibility
- Having a single configuration of a package is now enforced on unification sets
- These are the set of nodes used together at runtime (the one shown is for gptune)
- This allows us to use the latest version of py-gevent, because now we can have two versions of py-cython

We want to dynamically “split” nodes when needed

1. Start with deducing single dependency nodes:

```
node(DependencyName)
    :- dependency_holds(PkgName, DependencyName)
```

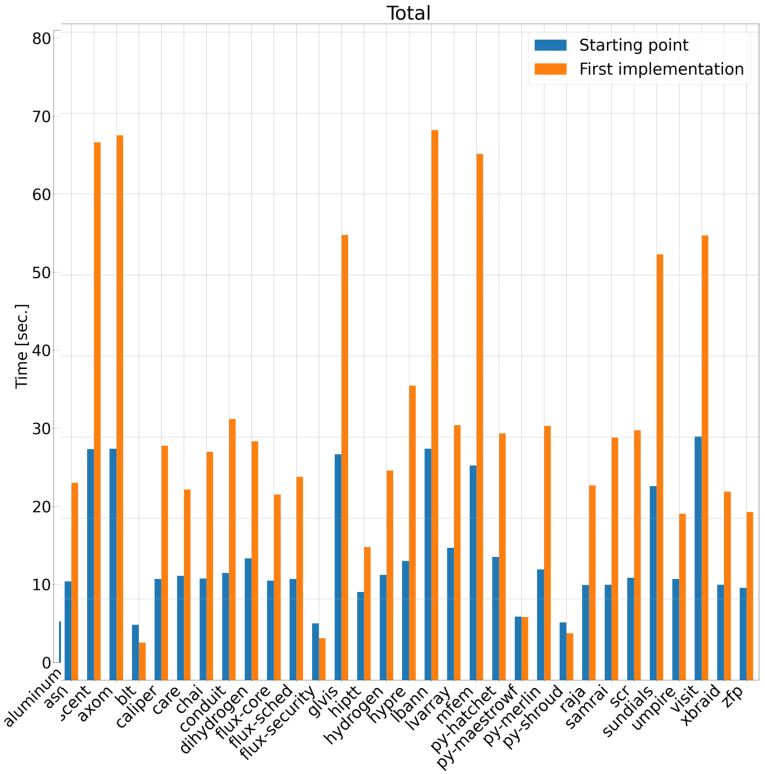
2. Allow solver to **choose** to duplicate a node:

Converted node identifier from **name** to (**name, id**)

```
1 {
    depends_on(PkgNode, node(0..Y-1, DepNode), Type)
    : max_dupes(DepNode, Y)
} 1
    :- dependency_holds(PkgNode, DepNode).
```

3. Re-encode package metadata so that it can be associated with duplicates

First try at allowing duplicates in a single solve



Increased runtimes by >> 2x in some cases

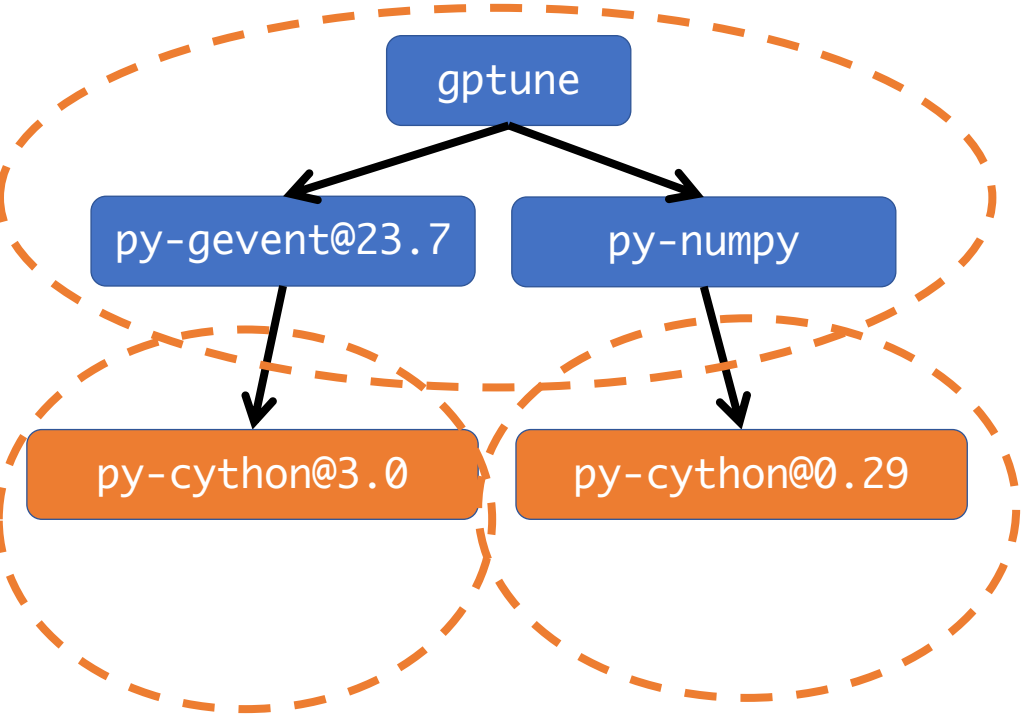
Cycle detection in the solver is *expensive*

```
path(A, B) :- depends_on(A, B).
path(A, C) :- path(A, B), depends_on(B, C).

% this constraint says "no cycles"
:- path(A, B), path(B, A).
```

- Has to maintain path() predicate representing paths between nodes
- Cycles are actually rare in solutions
  - Switched to post-processing for cycle detection
  - Only do expensive solve if a cycle is detected in a solution
- Eventually moved this calculation *into* the solver using some custom directives from the developers

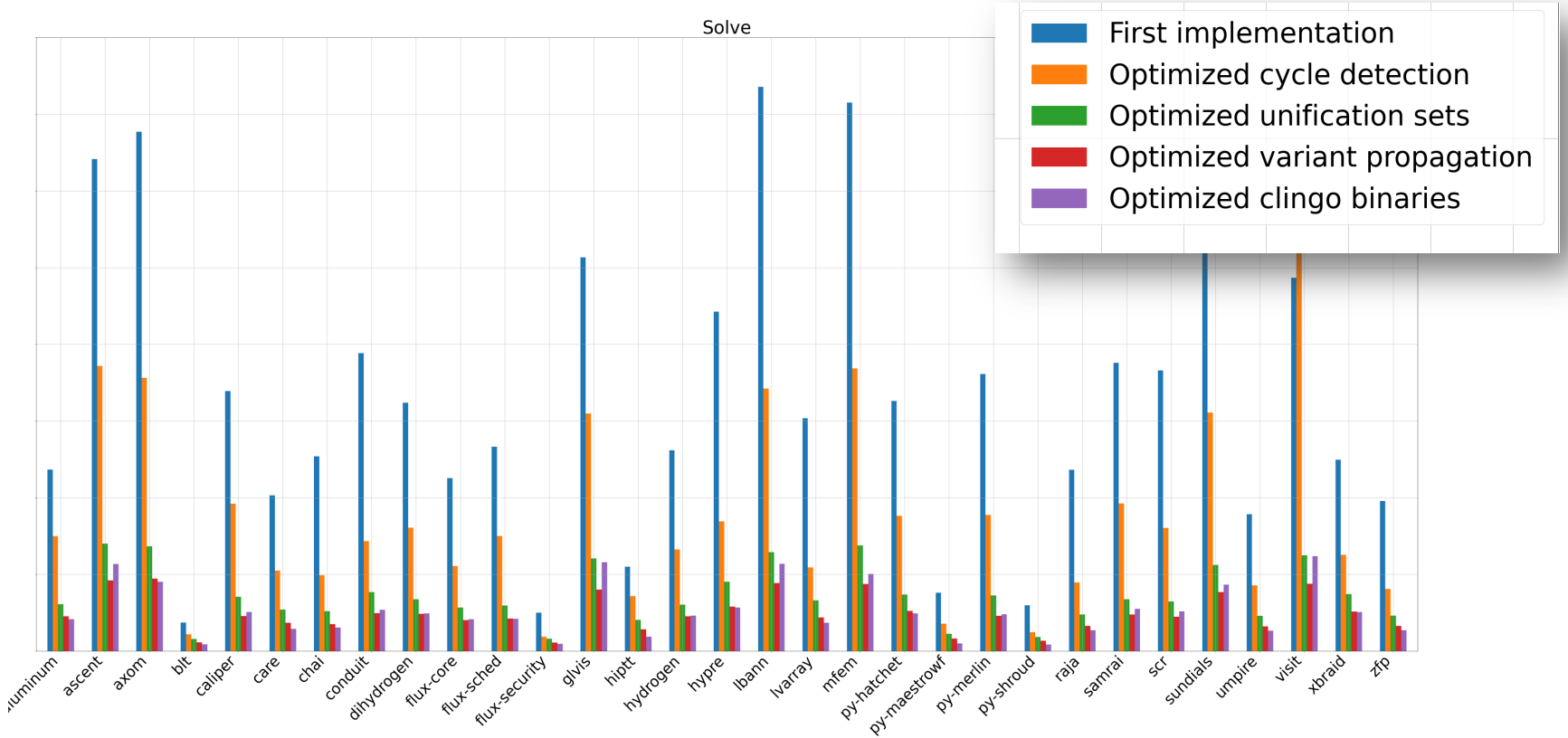
50%+ improvement in solve time

Unification sets can be expensive too

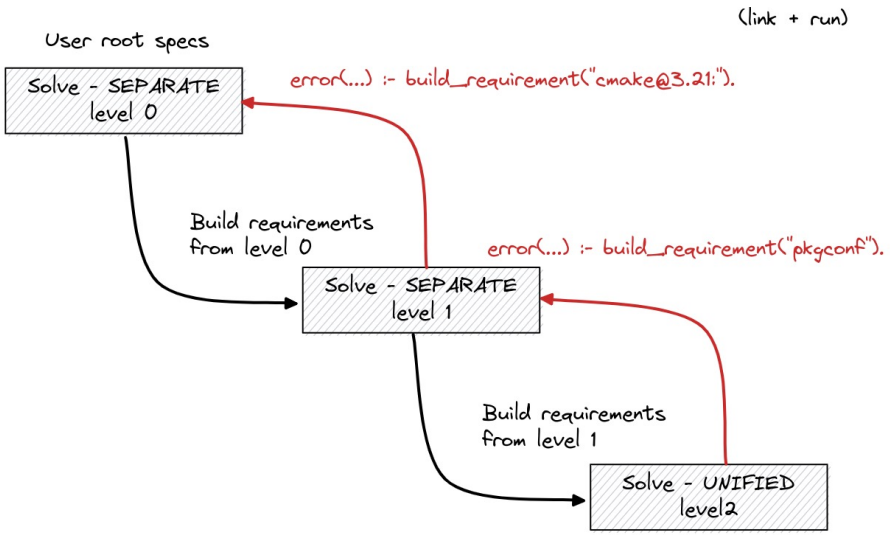


- Unification set creation was originally recursive for *any* build dependencies
  - Ends up blowing up grounding
- Mitigation:
  - Only create new sets for explicitly *marked* build tools
  - Transitive build dependencies that are not from marked build tools go into a *common* unification set
- Need better heuristics to split when necessary for full generality

Through many different optimizations, we were able to reclaim enough performance to make duplicate build dependencies tractable



It was not trivial to find a model that was both performant *and* tightly coupled



- We tried an iterative version with multiple solves
- Multiple solves had some disadvantages:
  - Slower due to overhead of multiple solves
  - Not coupled, so feedback from build to run environment (and back) was awkward
  - Packagers needed to “help” the solver
- Requiring packagers to provide solve hints in packages isn’t practical

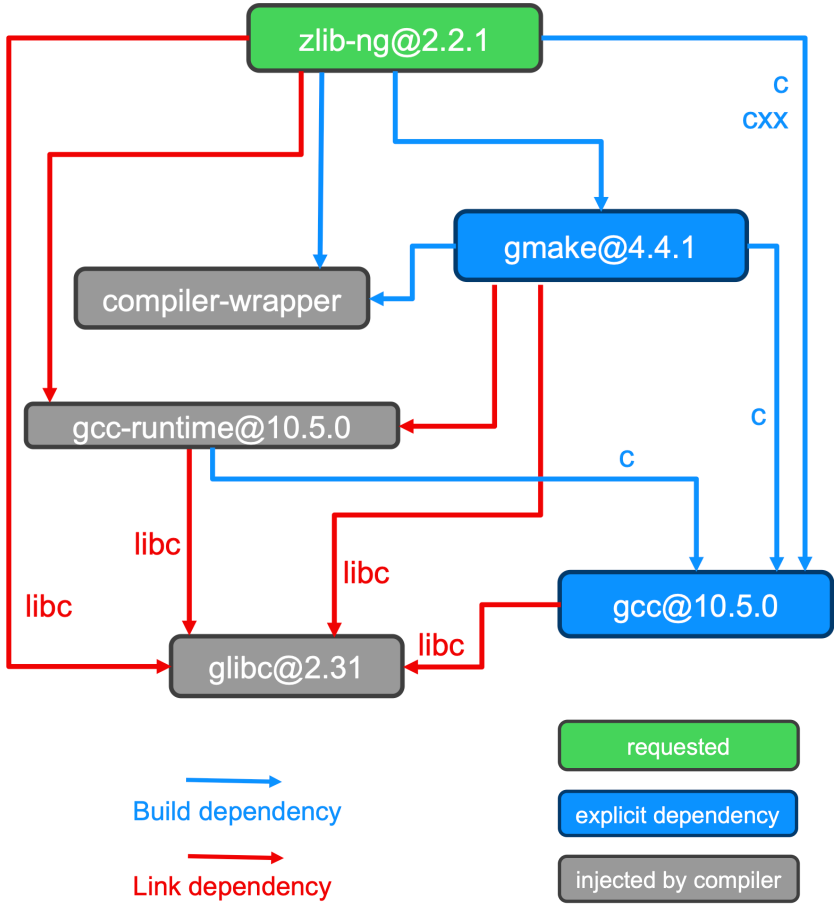
Fourth Challenge: v1.0 adds language dependencies

```
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

```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
```

```yaml
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

Configuring compilers in Spack v1.*

Spack v0.x

compilers.yaml

```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

```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: [...]
```

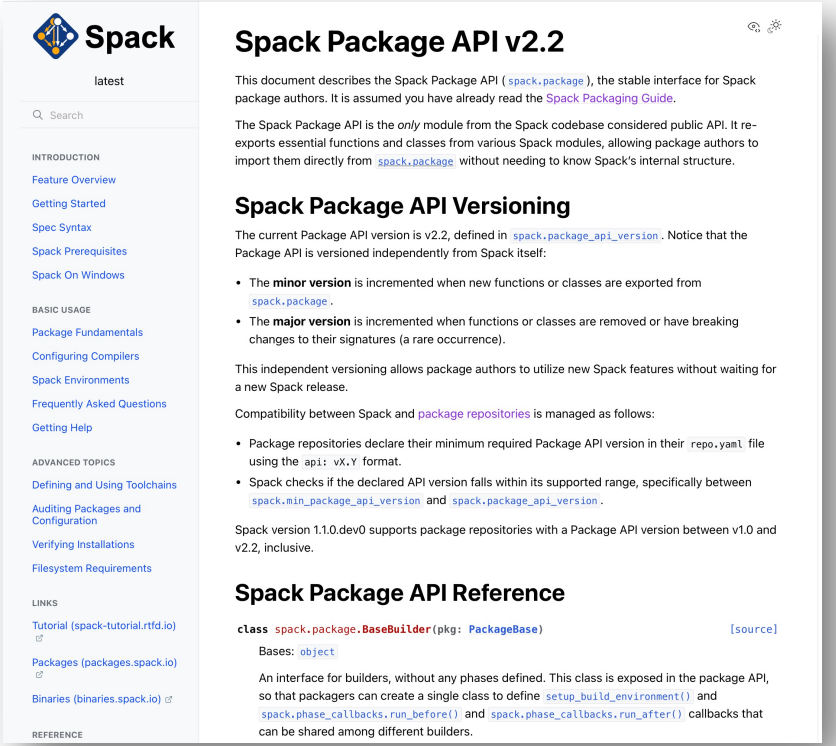
- We automatically convert compilers.yaml, when no compiler is configured
- We will still support *reading* the old configuration until at *least* v1.1
- All fields from `compilers.yaml` are supported in `extra_attributes`

Final challenge: Splitting the package repository

- Spack is two things:
  - Command line tool `spack`
  - Package repository with 8,500+ 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



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

Package split process

- Sync packages to **`spack/spack-packages`**
  - Git history is preserved 😌
- Turn package repositories into Python namespace packages
  - **`spack.pkg.builtin`** is now **`spack_repo.builtin`**
- Move build systems to **`spack_repo.builtin.build_systems`**
- Update packages to use fewer Spack internals
- Enable CI on **`spack/spack-packages`**
- Make Spack support Git-based package repositories

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

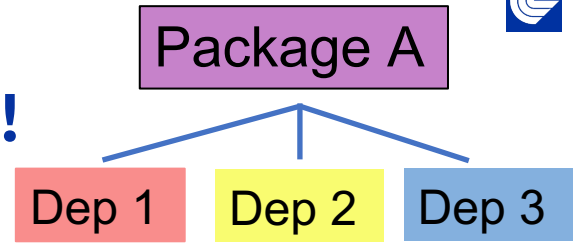
```
spack:
  repos:
    builtin:
      git: https://github.com/spack/spack-packages.git
      destination: /path/to/clone/of/spack-packages
      branch: develop
```

Useful commands after repo split

1. `spack repo migrate`: fixes imports in custom repos for you
2. `spack repo set --destination ~/spack-pkgs builtin`: put packages in your favorite location
3. (`spack repo update`: update & pin package repos SOON™)

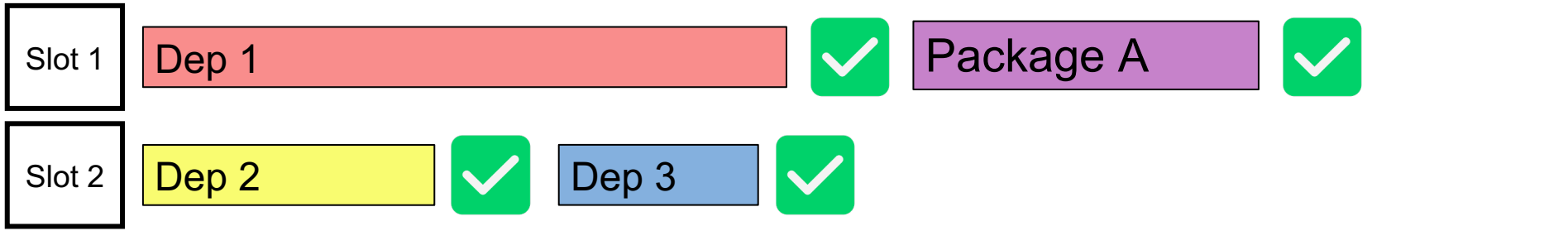
New docs: **https://spack.readthedocs.io/en/latest/repositories.html**

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'



But wait! There's more!

HPSF

Spack is a core project in the High Performance Software Foundation

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

@hpsf.bsky.social

hpsf.io

Join us online!

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

slack.spack.io

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

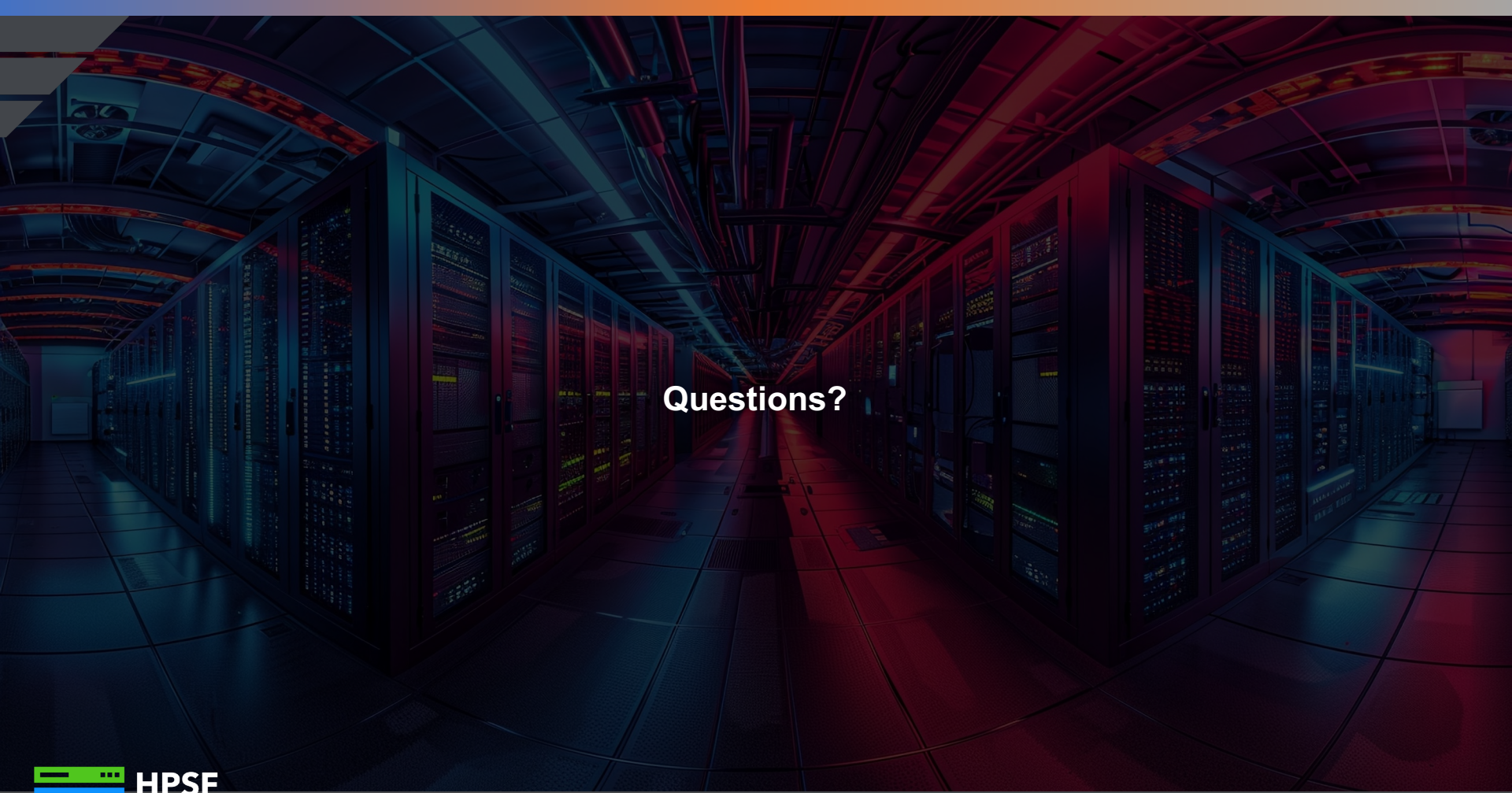
@spackpm.bsky.social

@spack@hpc.social

@spackpm

spack.io

We hope to make distributing & using HPC software easy!


Questions?