Spack: Package Management for HPC

ATPESC 2024 Software Track

Todd Gamblin

Spack Project Lead
Distinguished Member of Technical Staff
Lawrence Livermore National Laboratory

THE LINUX FOUNDATION

Background

THE LINUX FOUNDATION

We build codes from hundreds of small, complex pieces

Just when we're starting to solve the problem of how to create software using reusable parts, it founders on the nuts-and-bolts problems outside the software itself.

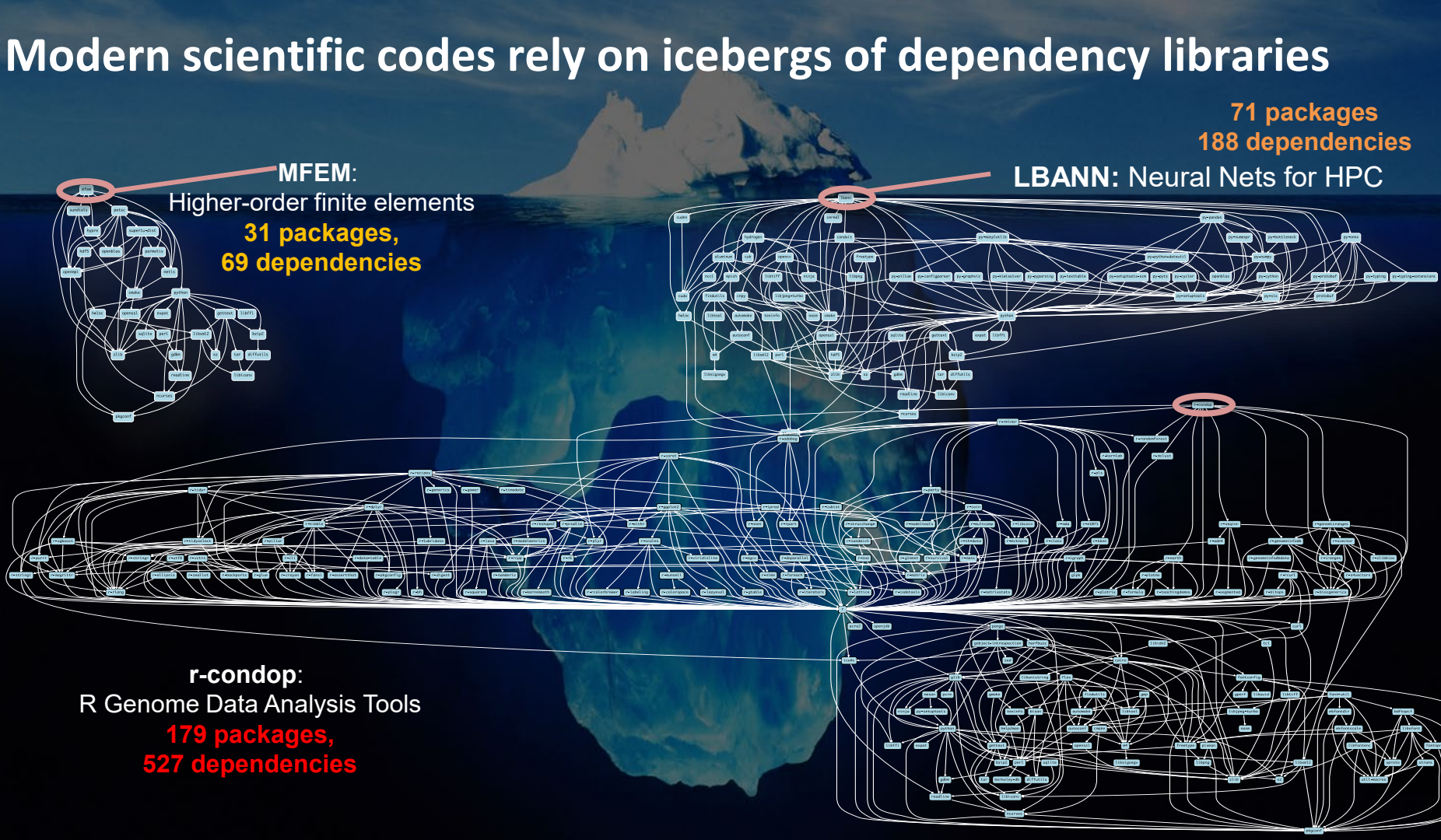
P. DuBois & T. Epperly. ***Why Johnny Can't Build***. Scientific Programming. Sep/Oct 2003.

- Component-based software development dates back to the 60's
  - M.D. McIlroy, *Mass Produced Software Components*. NATO SE Conf., 1968
- **Pros are well known:**
  - Teams can and must reuse each others' work
  - Teams write less code, meet deliverables faster
- **Cons:**
  - Teams must ensure that components work together
  - Integration burden increases with each additional library
  - Integration must be repeated with each update to components
  - **Components must be vetted!**
- **Managing changes over time is becoming intractable**

Build-time incompatibility; fail fast

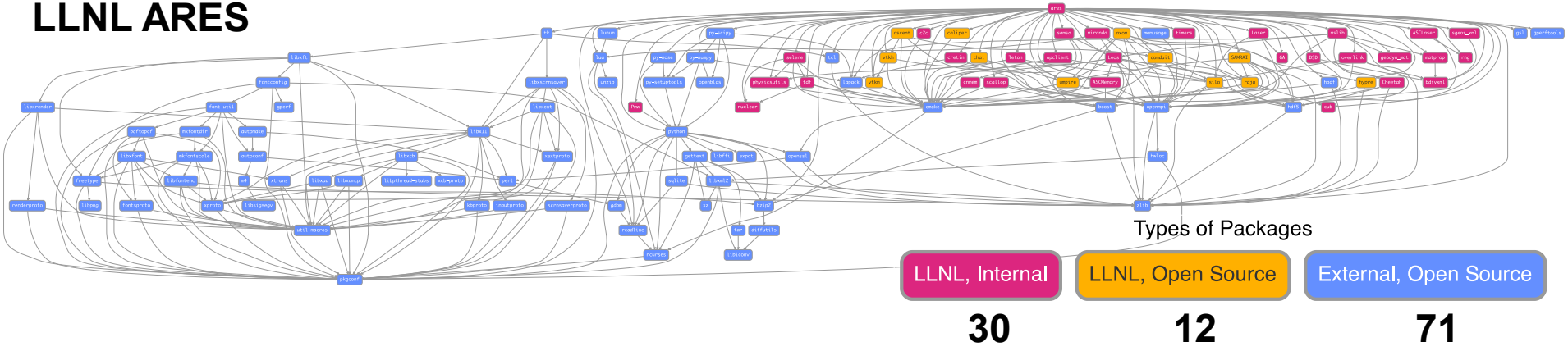
Appears to work; subtle errors later

Modern scientific codes rely on icebergs of dependency libraries



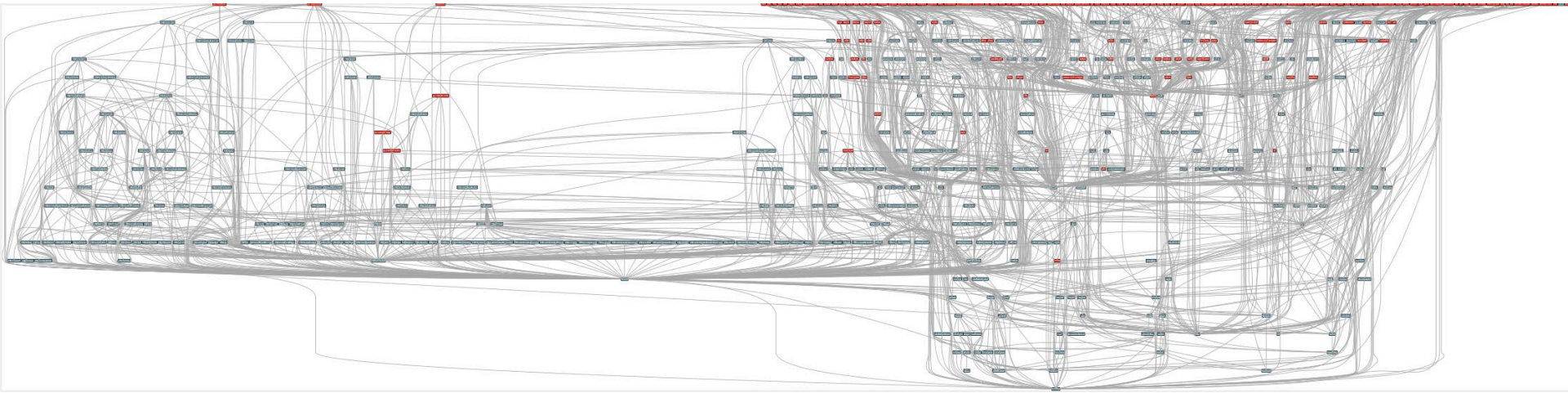
Even proprietary software builds on top of open source

LLNL ARES



- Open source is critical for nearly every application
- We *cannot* replace all these OSS components with our own
  - How do we put them all together effectively?
  - Do you *have* to integrate this stuff by hand?

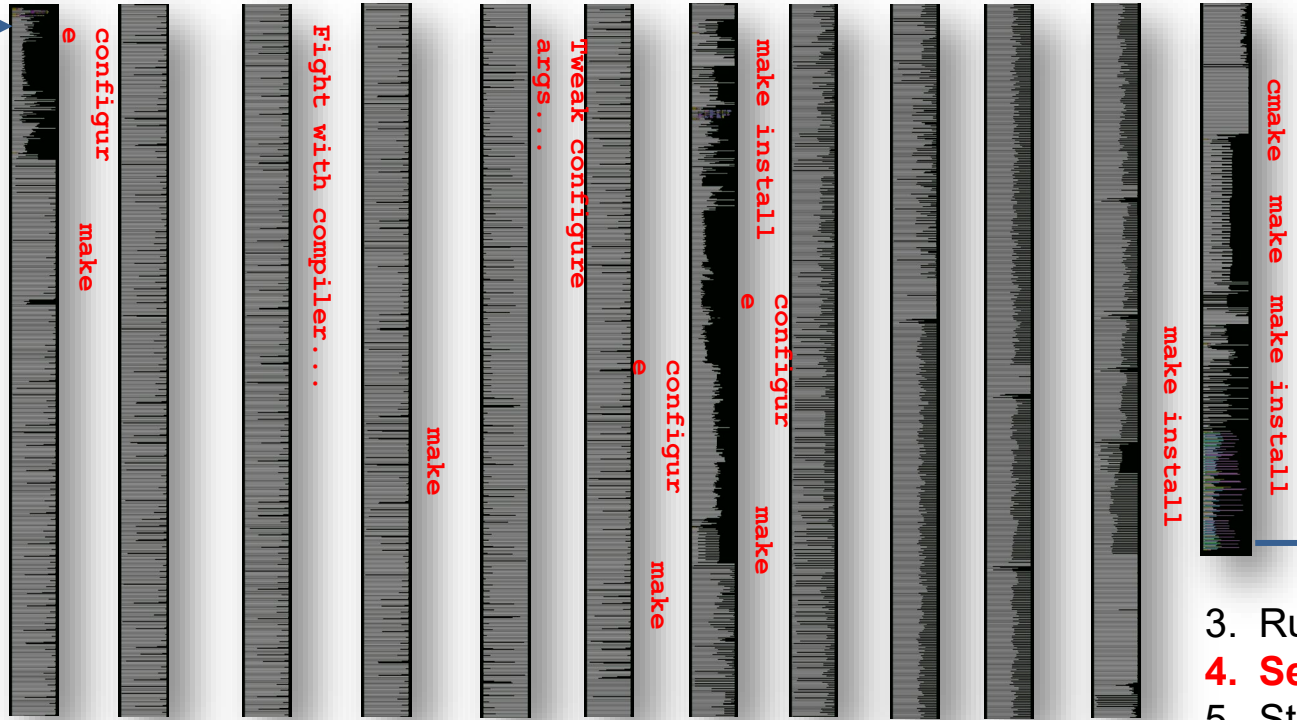
ECP's E4S stack is even larger than these codes



- Red boxes are the packages in it (about 100)
- Blue boxes are what *else* you need to build it (about 600)
- It's infeasible to build and integrate all of this manually

Some history:
How to install software on a supercomputer, circa 2013

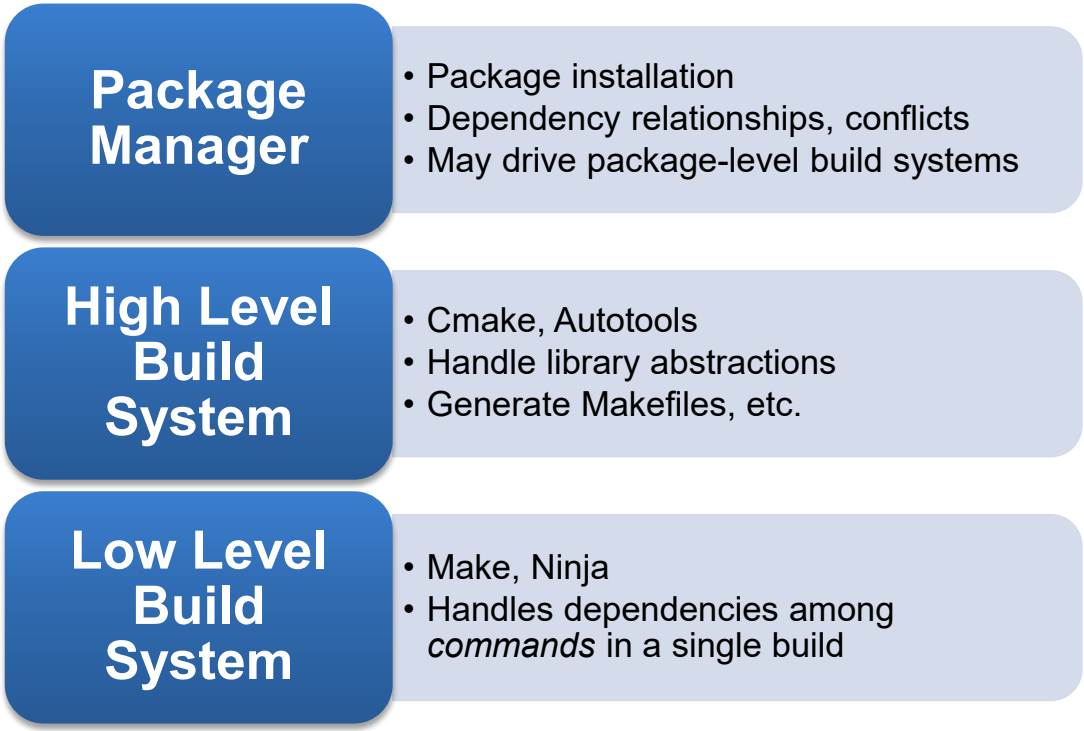
1. Download all 16 tarballs you need
2. Start building!



3. Run code
4. **Segfault!?**
5. Start over…

What does a package managers do to help?

- **Does not** a replace Cmake/Autotools
- Manages ***dependencies***
  - Drives package-level build systems
  - Ensures **consistency** and **compatibility** among builds of packages in the ecosystem
- Stores **community knowledge**
  - Cache of package build recipes
  - Determining magic configure lines takes time



Many package managers (conda, pip, apt, etc.) make simplifying assumptions about the software ecosystem

- **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) and *just one language*

Outside these boundaries, users are typically on their own

High Performance Computing (HPC) violates many of these assumptions

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

Some Supercomputers

Oak Ridge National Lab
Power9 / NVIDIA

RIKEN
Fujitsu/ARM a64fx

Lawrence Berkeley National Lab
AMD Zen / NVIDIA

Argonne National Lab
Intel Xeon / Xe

Oak Ridge National Lab
AMD Zen / Radeon

Lawrence Livermore National Lab
AMD Zen / Radeon

What about containers?

- **Containers provide a great way to reproduce and distribute an already-built software stack**

- **Someone needs to build the container!**
  - This isn't trivial
  - Containerized applications still have hundreds of dependencies

- **Using the OS package manager inside a container is insufficient**
  - Most binaries are built unoptimized
  - Generic binaries, not optimized for specific architectures

- **HPC containers are often optimized per-system**
  - Not clear that we can ever build one container for all facilities

We need something more flexible to **build** versions of containers

Overview & Community

THE LINUX FOUNDATION

Spack enables Software distribution for HPC

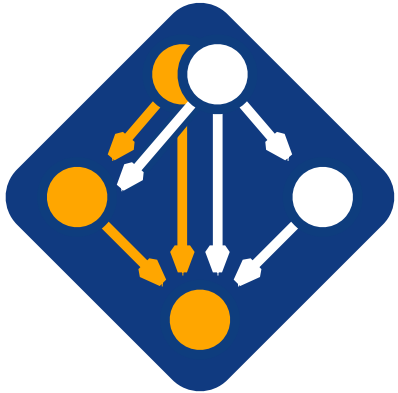
- Spack automates the build and installation of scientific software
- Packages are *parameterized,* so that users can easily tweak and tune configuration

No installation required: clone and go

```
$ git clone https://github.com/spack/spack
$ spack install hdf5
```

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
```



github.com/spack/spack

- Ease of use of mainstream tools, with flexibility needed for HPC
- 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)
  - Supports *binary "buildcaches"* so that you don't have to build everything from source

Who can use Spack?

People who want to use or distribute software for HPC!

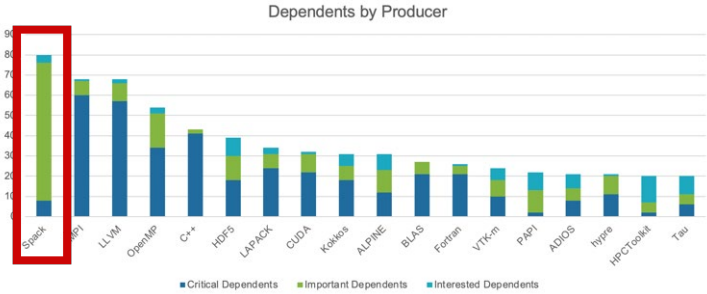
1. **End Users of HPC Software**
   - Install and run HPC applications and tools
2. **HPC Application Teams**
   - Manage third-party dependency libraries
3. **Package Developers**
   - People who want to package their own software for distribution
4. **User support teams at HPC Centers**
   - People who deploy software for users at large HPC sites

Spack was critical for ECP's mission to create a robust, capable exascale software ecosystem.

EXASCALE COMPUTING PROJECT

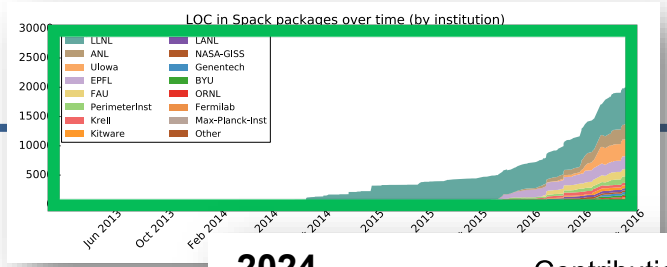
https://e4s.io

- Used for building software on the three U.S. exascale systems
- ECP built the Extreme Scale Scientific Software Stack (E4S) with Spack – more at https://e4s.io
- Project continues on ASC and ASCR funding



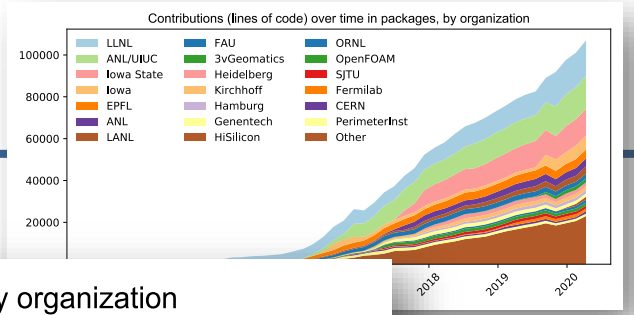
Spack was the most depended-upon project in ECP

LLNL-PRES-806064

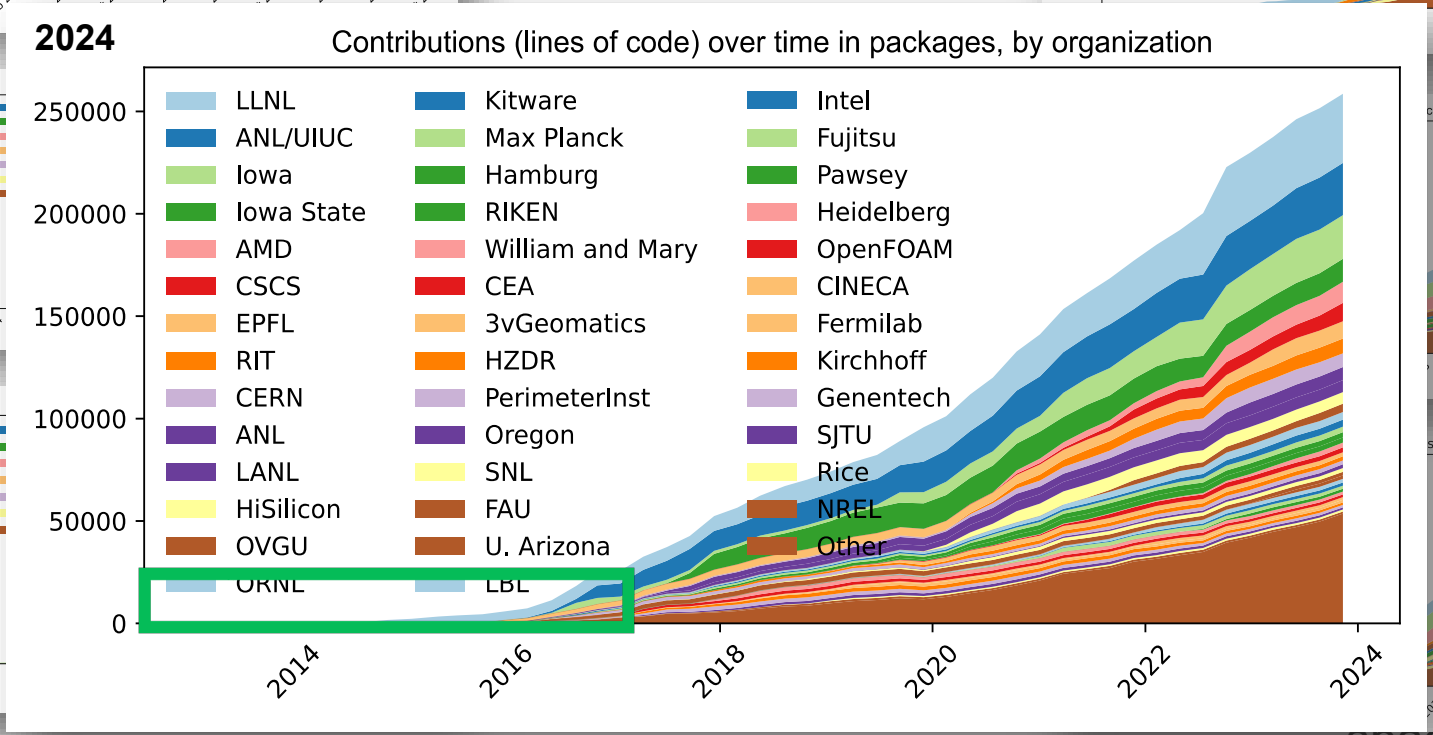


2016

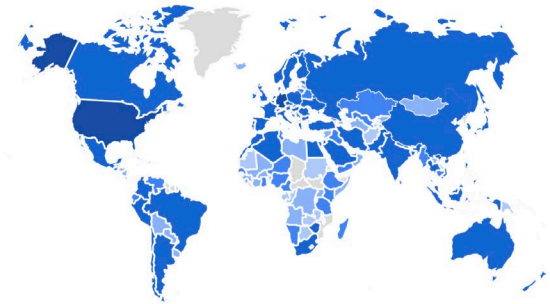
Fall 2020



2024



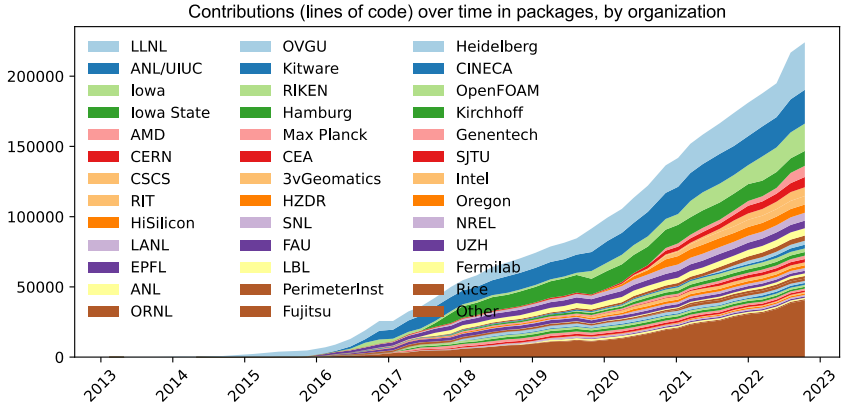
Spack sustains the HPC software ecosystem with the help of many contributors



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 8,000 software packages
Over 1,300 contributors

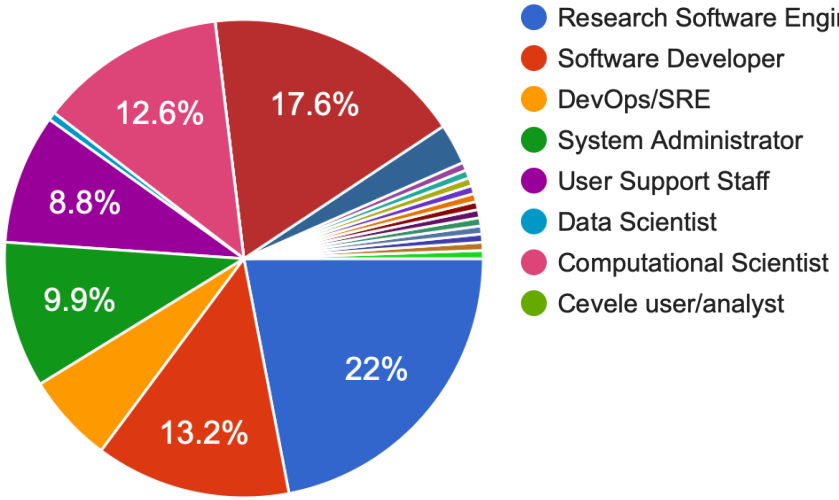


Contributors continue to grow worldwide!

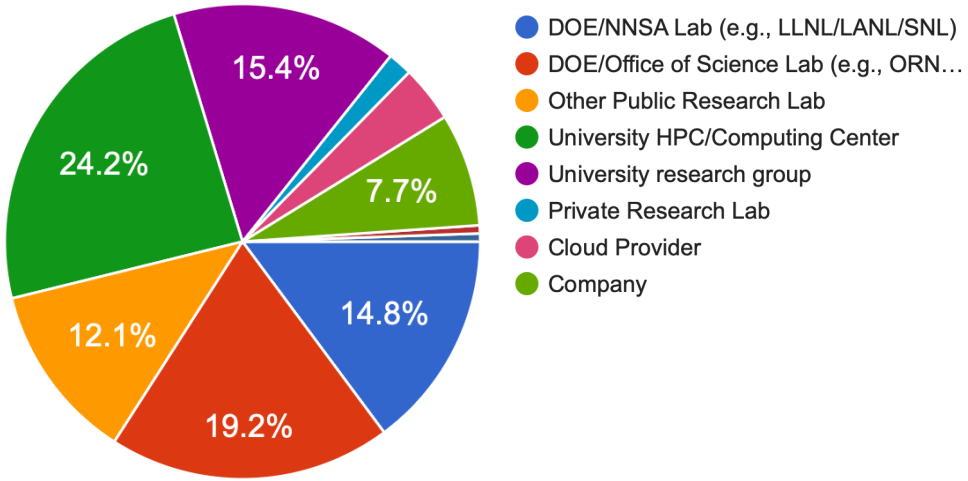
LLNL-PRES-806064

Spack users have diverse roles across many types of institutions

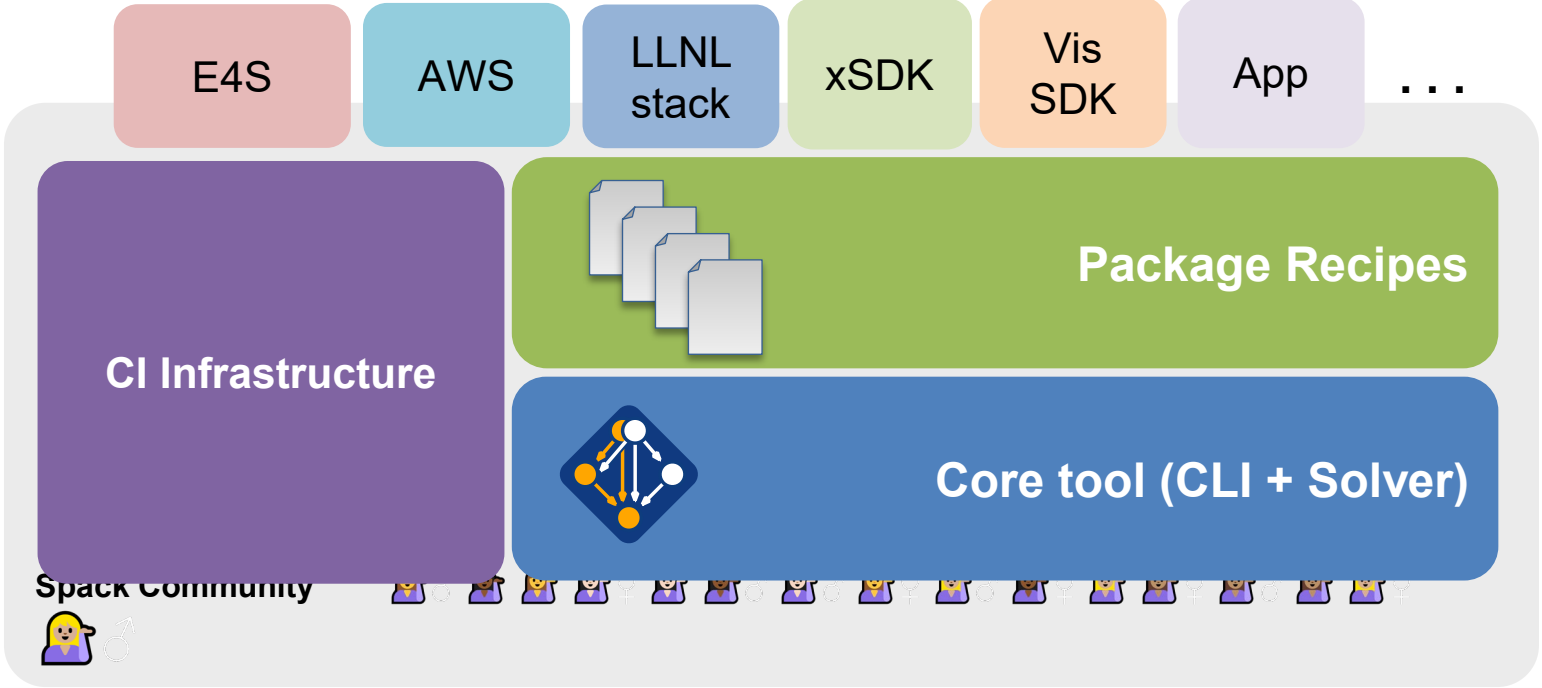
What type of user are you?



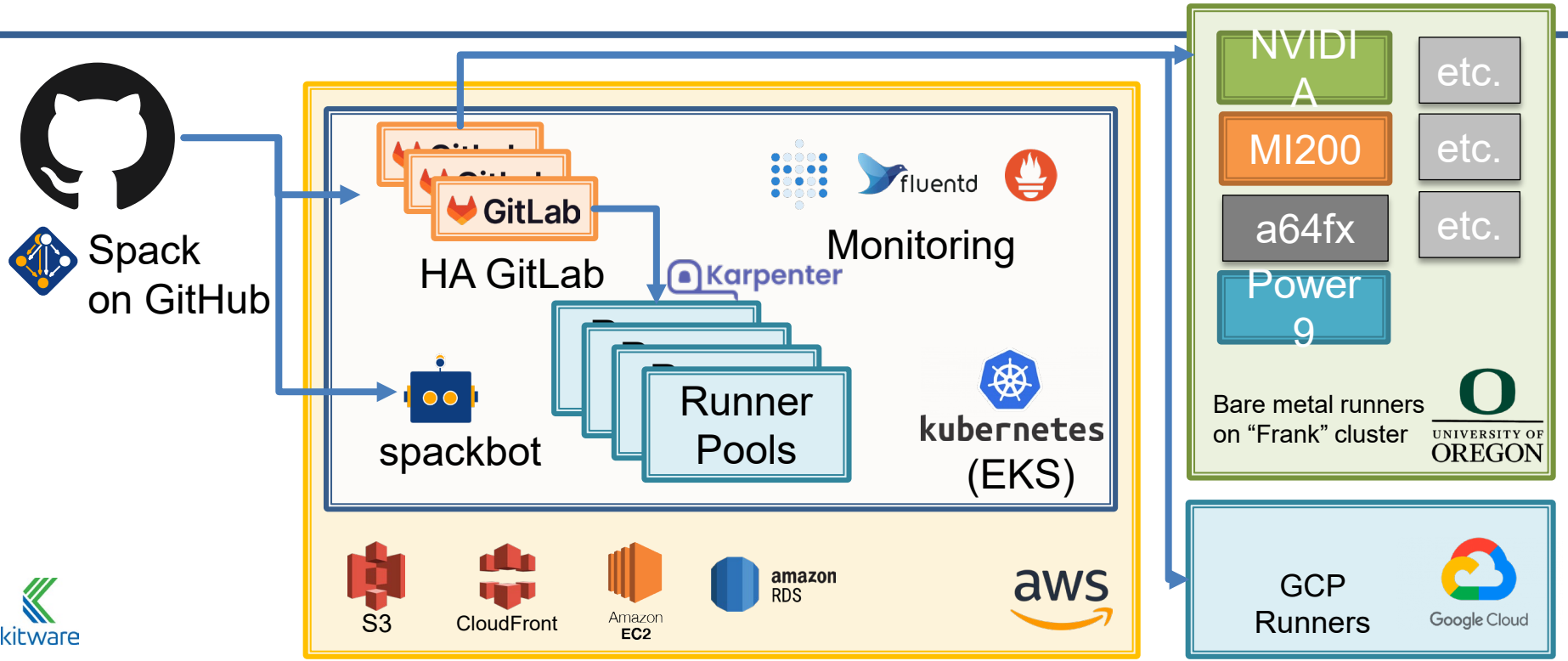
Where do you work?



What does the Spack project look like now?



Spack CI Architecture

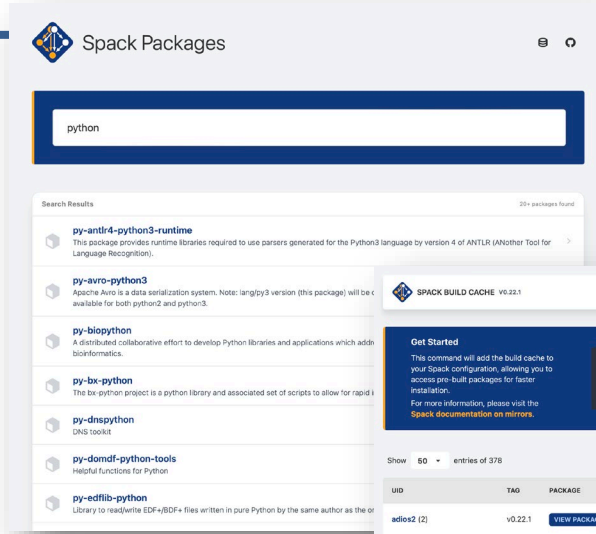


kitware

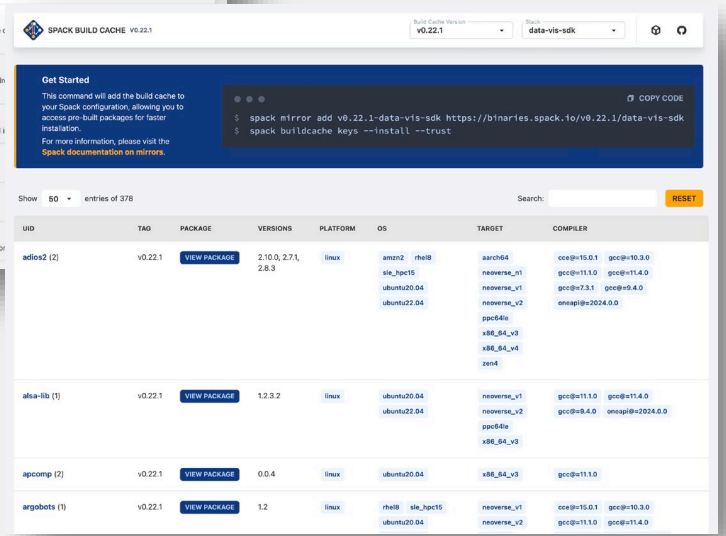
Lawrence Livermore
National Laboratory

CI enables us to maintain public *build caches*

- We test a subset of package builds:
  - In CI for every pull request
  - On release branches
  - On the develop branch
- We provide build caches for:
  - Every release
  - Latest develop branch
  - Weekly develop snapshots
  - Different architectures and software stacks



Find available packages at packages.spack.io



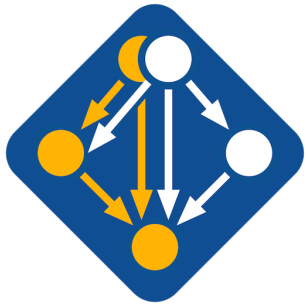
See available builds at cache.spack.io

LLNL-PRES-806064

Spack recently became a Linux Foundation Project!

What does that mean?

- 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 Foundation
- Spack project resources owned by Linux Foundation:
  - spack.io website
  - GitHub Organization



Why are we doing this?

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

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 tools in the HPC Space:

- **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

Spack Usage

THE LINUX FOUNDATION

Spack provides a *spec* syntax to describe customized installations

```
$ 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=zen2      set target microarchitecture
$ spack install mpileaks@3.3 ^mpich@3.2 %gcc@4.9.3   ^ dependency information
```

- 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 *templates*
They use a simple Python DSL to define how to build

Not shown: patches, **resources**, **conflicts**, other directives.

```
from spack import *

class Kripke(CMakePackage):
    """Kripke is a simple, scalable, 3D Sn deterministic particle
       transport proxy/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):
        # Kripke does not provide install target, so we have to copy
        # things into place.
        mkdirp(prefix.bin)
        install('../spack-build/kripke', prefix.bin)
```

Base package
(CMake support)

Metadata at the class level

Versions

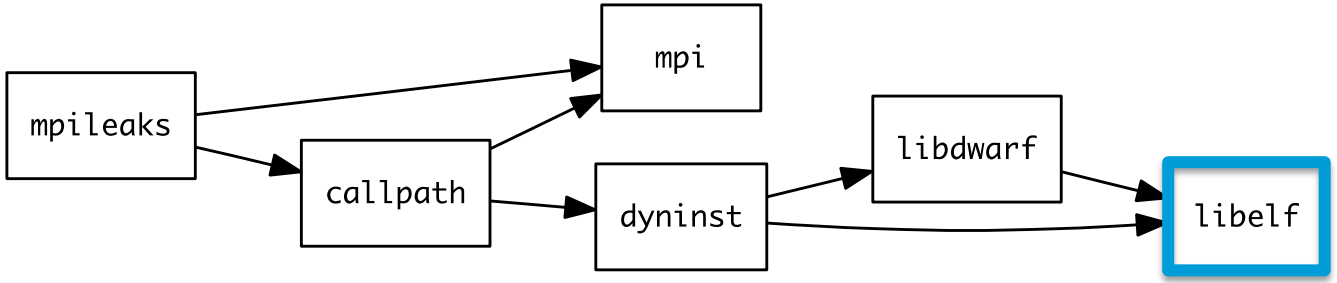
Variants (build options)

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.

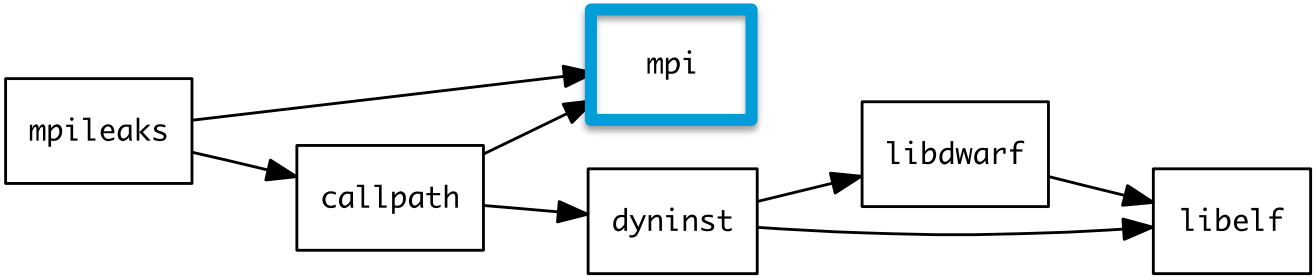
Spack Specs can constrain versions of dependencies




```
$ 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:

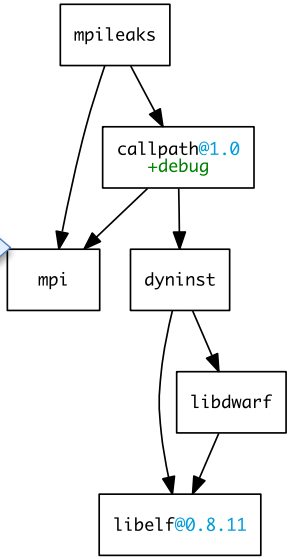
```
$ spack install mpileaks ^mpi@2
```

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

mpileaks ^callpath@1.0+debug ^libelf@0.8.11

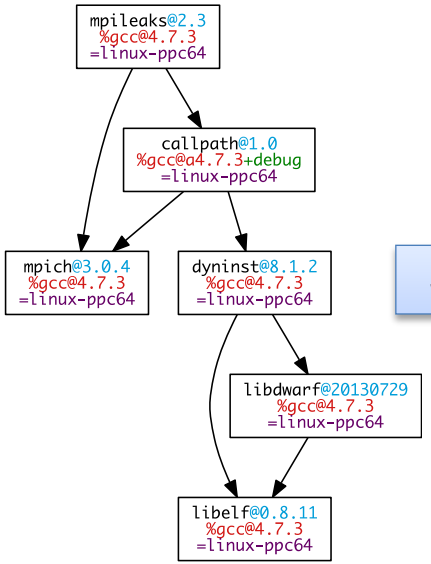
User input: *abstract* spec with some constraints

Normalize



Abstract, normalized spec with some dependencies

Concretize



Concrete spec is fully constrained and can be passed to install

Store

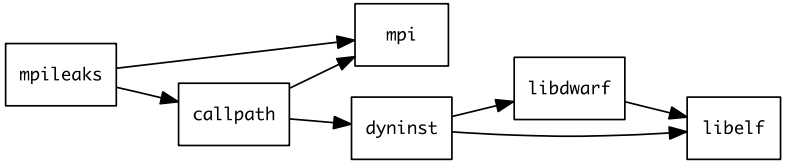
spec.yaml

```
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 stored with installed package

Spack handles combinatorial software complexity

Dependency DAG



Hash

Installation Layout

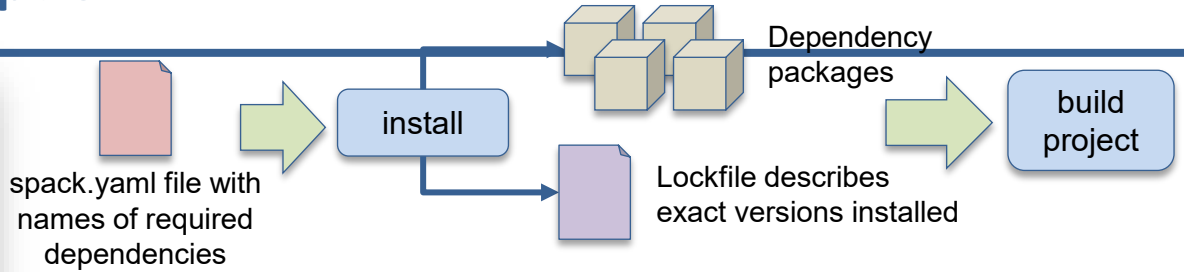
```
opt
└── spack
    ├── linux-rhel7-skylake
    │   └── gcc-8.3.0
    │       ├── mpileaks-1.0-hc4sm4vuzpm4znmvrfzri4ow2mkphe2e
    │       ├── callpath-1.0.4-daqqpssxb6qbfrztsezkmhus3xoflbsy
    │       ├── openmpi-4.1.4-u64v26igxvxyn23hysmklfums6tgjv5r
    │       ├── dyninst-12.1.0-u64v26igxvxyn23hysmklfums6tgjv5r
    │       ├── libdwarf-20180129-u5eawkvaoc7vonabe6nndkcfwuv233cj
    │       └── libelf-0.8.13-x46q4wm46ay4pltriijbgizxjrhbaka6
```

- 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*

Spack environments enable users to build customized stacks from an abstract description

Simple spack.yaml file

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

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



- spack.yaml describes project requirements
- spack.lock describes exactly what versions/configurations were installed, allows them to be reproduced.
- Can also be used to maintain configuration together with Spack packages.
  - E.g., versioning your own local software stack with consistent compilers/MPI implementations
  - Allows developers and site support engineers to easily version Spack configurations in a repository

Concrete spack.lock file (generated)

```
{
 "concrete_specs": {
   "6s63so2kstp3zyvjezglndmavy6l3nul": {
     "hdf5": {
         "version": "1.10.5",
         "arch": {
             "platform": "darwin",
             "platform_os": "mojave",
             "target": "x86_64"
         },
         "compiler": {
             "name": "clang",
             "version": "10.0.0-apple"
         },
         "namespace": "builti
         "parameters":
```

Environments have enabled us to add build many features to support developer workflows

package.py

spack.yaml configuration

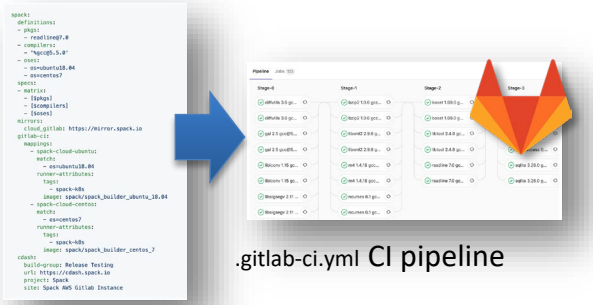
spack external find

Automatically find and configure external packages on the system

spack test

Packages know how to run their own test suites

package.py



spack.yaml

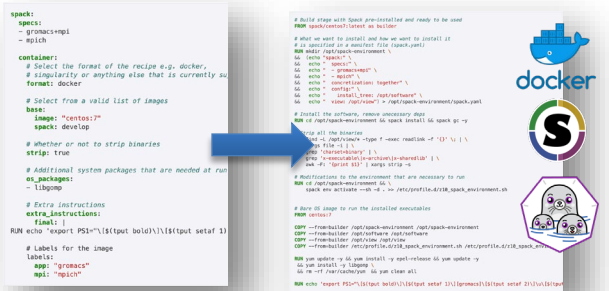
.gitlab-ci.yml CI pipeline

spack ci

Automatically generate parallel build pipelines
(more on this later)

spack containerize

Turn environments into container build recipes



spack develop lets developers work on many packages at once

- Developer features so far have focused on single packages (spack dev-build, etc.)
- New spack develop feature enables development environments
  - Work on a code
  - Develop multiple packages from its dependencies
  - Easily rebuild with changes
- Builds on spack environments
  - Required changes to the installation model for dev packages
  - dev packages don't change paths with configuration changes
  - Allows devs to iterate on builds quickly

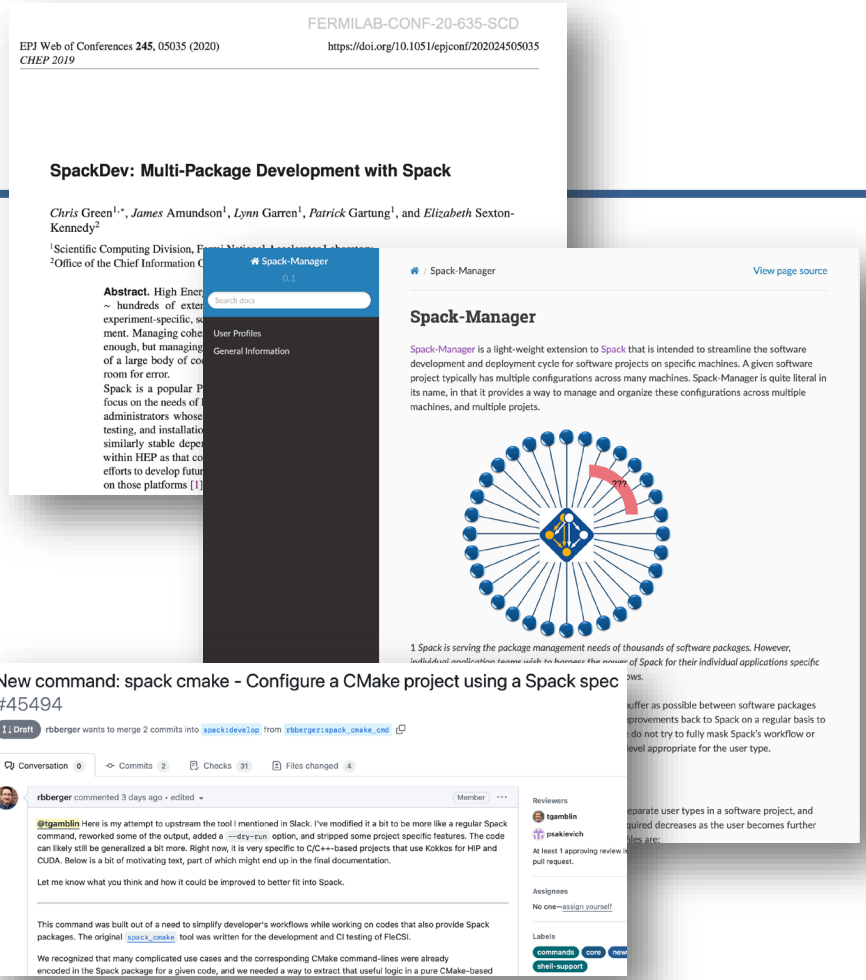
```
$ spack env activate .
$ spack add myapplication
$ spack develop axom@0.4.0
$ spack develop mfem@4.2.0



$ ls
spack.yaml    axom/    mfem/



$ cat spack.yaml
spack:
    specs:
        - myapplication    # depends on axom, mfem

    develop:
        - axom @0.4.0
        - mfem @develop
```

Teams are building development front-ends top of on Spack

- Many different approaches:
  - Thin **mack** wrapper for MARBL team at LLNL
  - **spack-manager** at Sandia
  - **SpackDev** at Fermilab
  - **spack-organizer** at CEA
  - **spack_cmake** at LANL

- Workflow seems to be converging around environments + spack develop

- We are trying to bring many of the features from these scripts into core
  - Most of the front-ends are opinionated in one way or another
  - We want spack to support but not force these workflows

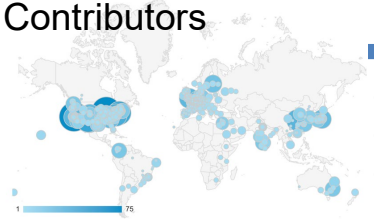


Solving and Complexity

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

yaml default config packages.yaml

admins, users

yaml local preferences config packages.yaml

users

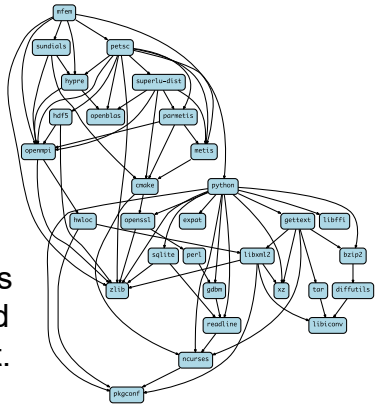
yaml local environment config spack.yaml

users

Command line constraints

spack install hdf5@1.12.0 +debug

Concrete spec is fully constrained and can be built.



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

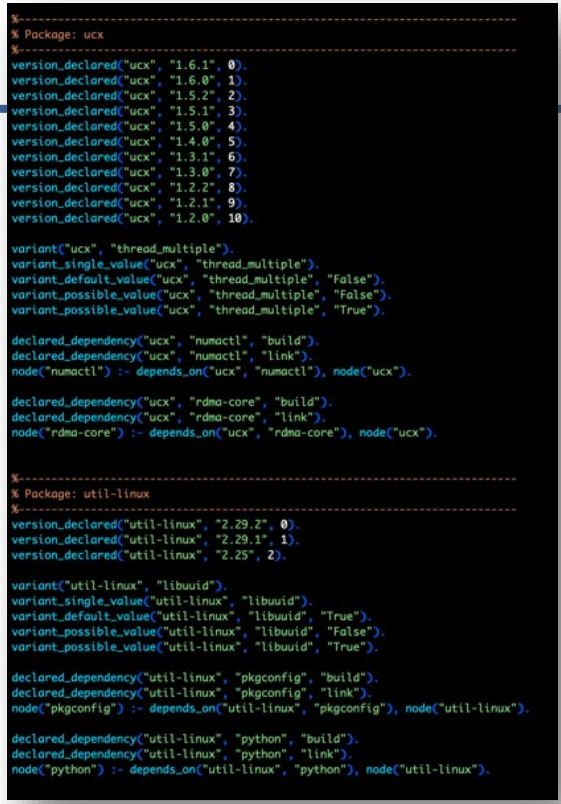
constraints on cuda version

compiler support for x86_64 and ppc64le

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

We reimplemented Spack's concretizer using *Answer Set Programming*

- Was originally a greedy, custom Python algorithm
- Answer Set Programming is a *declarative* programming paradigm
  - Looks like Prolog
  - Built around modern CDCL SAT solver techniques
  - Designed for combinatorial search problems
- ASP program has 2 parts:
  1. Large list of facts generated from package recipes (problem instance)
     - 60k+ facts is typical – includes dependencies, options, etc.
  2. Small logic program (~700 lines of ASP code)
- Algorithm (the part we write) is conceptually simpler:
  - Generate facts for all possible dependencies
  - Send facts and our logic program to the solver
  - Rebuild a DAG from the results
- We're using **Clingo**, the Potassco grounder/solver package



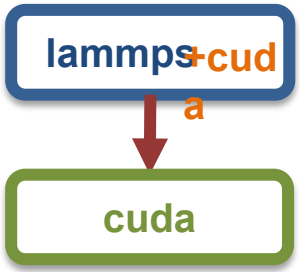
Some facts for HDF5 package

Spack's concretizer is implemented using Answer Set Programming (ASP)

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

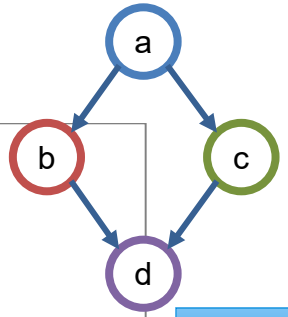
```
node(Dependency) :- node(Package), depends_on(Package, Dependency).
```

```
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) }.
```

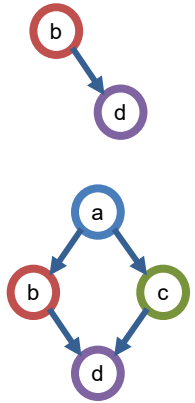
First-order Logic Program

ound

```
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

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

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



Stable Models (Answer Sets)

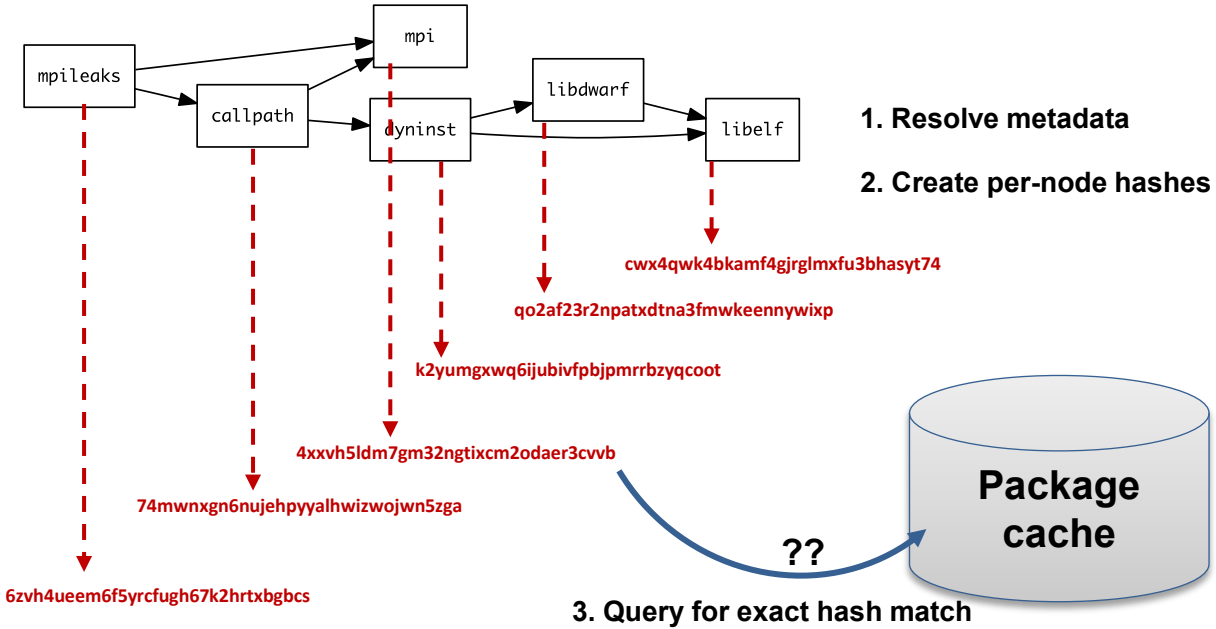
We use the *Clingo* solver from potassco.org

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!

Originally, Spack only reuse builds if hashes matched



- This is now:
  spack install --fresh
- 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").
```

Telling the solver to minimize builds is surprisingly simple in ASP

1. Allow the solver to *choose* a hash for any package:

```
{ hash(Package, Hash) : installed_hash(Package, Hash) } 1 :- node(Package).
```

2. Choosing a hash means we impose its constraints:

```
impose(Hash) :- hash(Package, Hash).
```

3. Define a build as something *without* a hash:

```
build(Package) :- not hash(Package, _), node(Package).
```

4. Minimize builds!

```
#minimize { 1@100,Package : build(Package) }.
```

With and without --reuse optimization

```
(spackle):solver> spack solve -Il hdf5
==> Best of 9 considered solutions.
==> Optimization Criteria:
  Priority  Criterion                                                  Installed  ToBuild
  1         number of packages to build (vs. reuse)                            -       20
  2         deprecated versions used                                           0        0
  3         version weight                                                     0        0
  4         number of non-default variants (roots)                             0        0
  5         preferred providers for roots                                      0        0
  6         default values of variants not being used (roots)                  0        0
  7         number of non-default variants (non-roots)                         0        0
  8         preferred providers (non-roots)                                    0        0
  9         compiler mismatches                                                0        0
  10        OS mismatches                                                      0        0
  11        non-preferred OS's                                                 0        0
  12        version badness                                                    0        2
  13        default values of variants not being used (non-roots)              0        0
  14        non-preferred compilers                                            0        0
  15        target mismatches                                                  0        0
  16        non-preferred targets                                              0        0

  -  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+sh
  -  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=defaul
 [+] 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
 [+] 53xbksk              ^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
 [+] znvkifs                  ^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

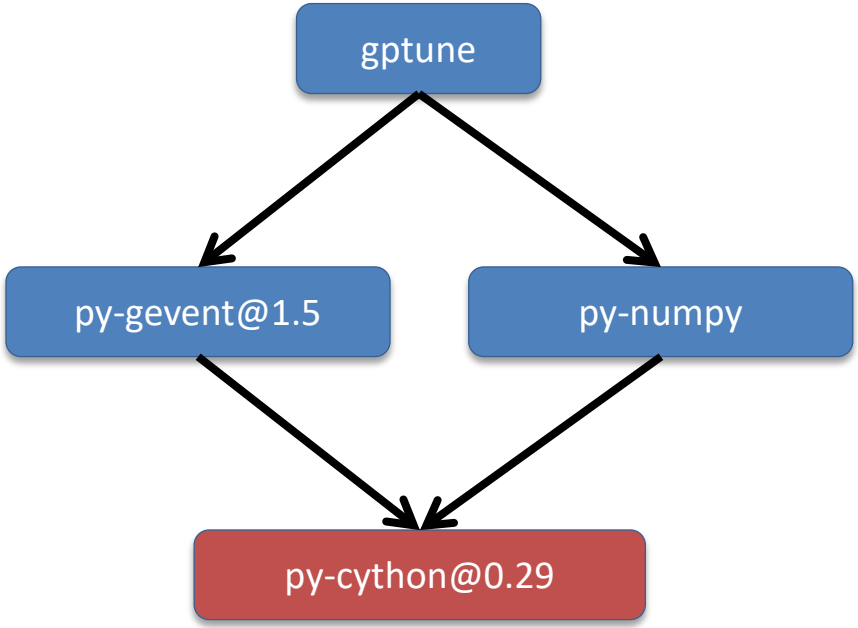
Recent Challenges

Some stats on problem sizes

- Main logic program is:
  - ~250 rules
  - 20 optimization criteria
  - 933 lines of ASP code
- Problem instances can vary quite a bit
  - Common dependencies get us some magic numbers
  - gmake's optional dependency on guile makes most solves consider at least 527 packages
  - gnuconfig is notably very simple ☺

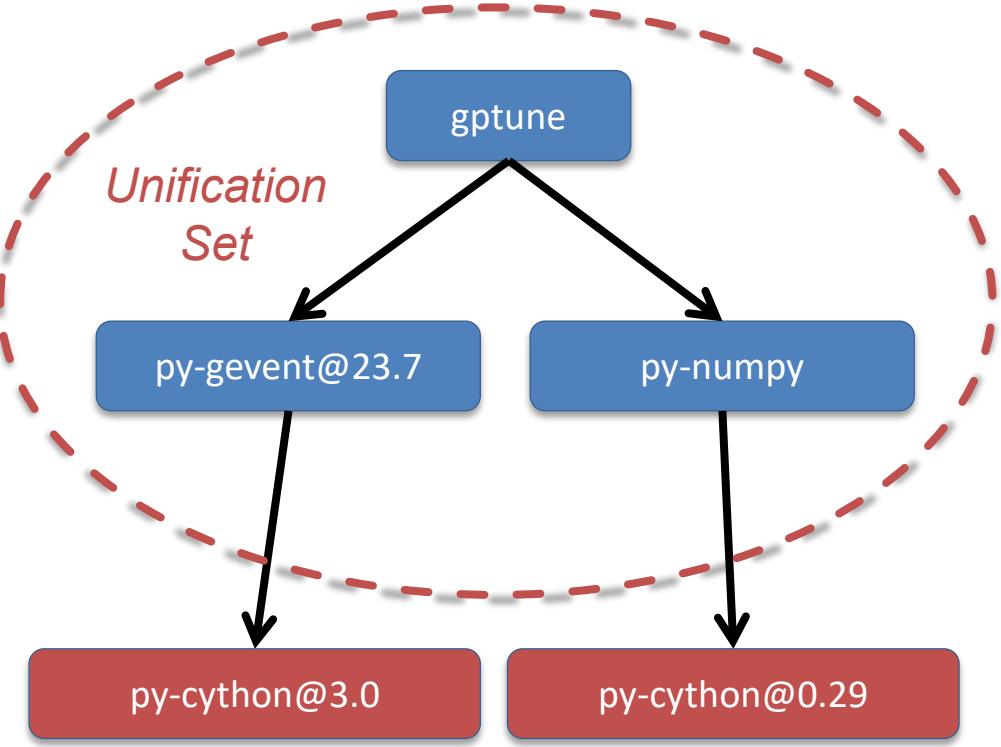
Package	Possible dependencies	Facts
gnuconfig	1	150
zlib	527	30,095
gmake	527	30,160
openmpi	527	109,021
qt	527	109,029
trilinos	694	224,142
root	699	146,372
mfem	714	273,078
r-condop	774	142,212
warpx	819	319,374
exawind	820	322,535

We have been working to make our solver more flexible



- Only one configuration per package allowed in the DAG
- Ensures ABI compatibility but is too restrictive
- 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

Start with deducing single dependency nodes:

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

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

Want to allow solver to **choose** to duplicate a node:

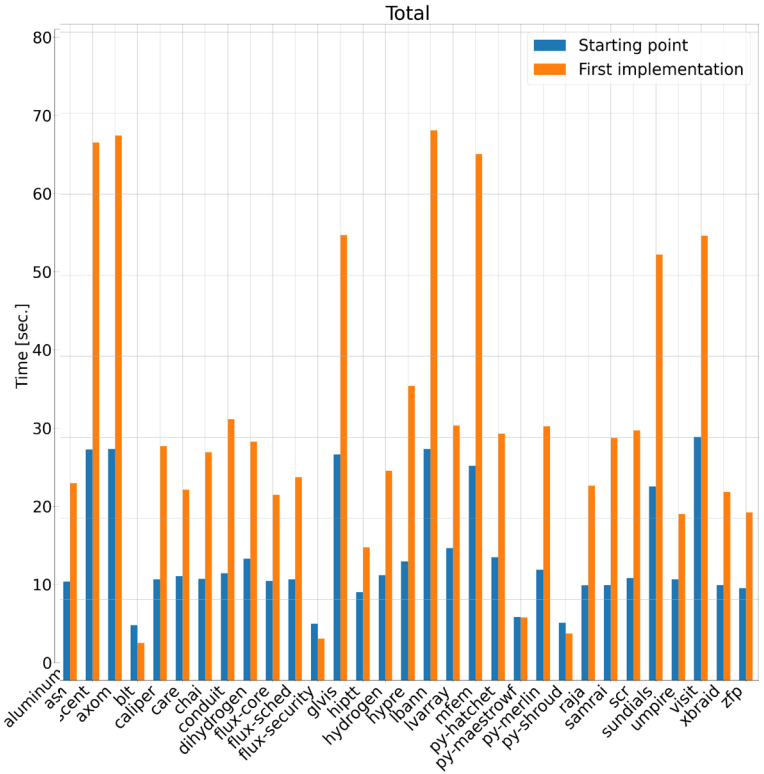
```
1 {
   depends_on(PkgNode, node(0..Y-1, DepNode), Type)
   : max_dupes(DepNode, Y)
} 1
   :- dependency_holds(PkgNode, DepNode).
```

Generic package metadata can be used with any duplicate node

```
pkg_fact("alsa-lib",version_declared("1.2.3.2",0,"package_py")).
pkg_fact("alsa-lib",version_declared("1.2.2",1,"package_py")).
pkg_fact("alsa-lib",version_declared("1.1.4.1",2,"package_py")).

pkg_fact("alsa-lib",condition(20)).
condition_reason(20,"alsa-lib depends on python when +python").
pkg_fact("alsa-lib",condition_trigger(20,15)).
```

- Facts that come from package descriptions can be used with all duplicate nodes
- We now have to ground multiple copies of most of our rules
- Performance still scales with total number of possible nodes
  - Small numbers of duplicates don't really explode the solve

First try at allowing duplicates in a single solve



Increased solve times by >> 2x in some cases

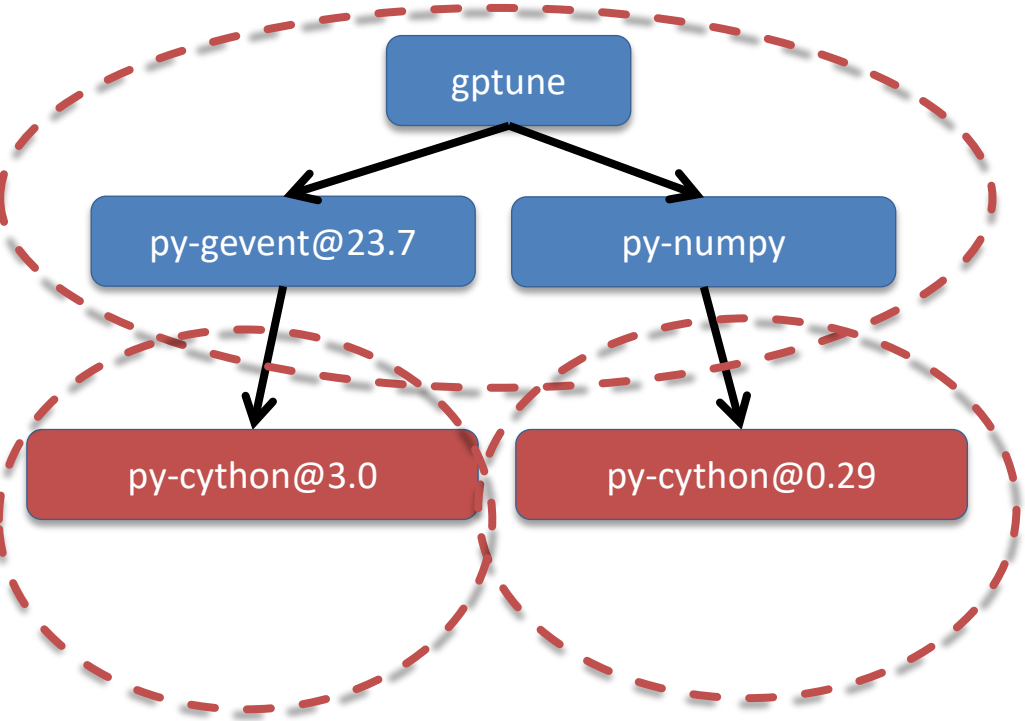
It turns out that cycle detection in the solver is *expensive*

```
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
- Similar issue arose for variant propagation in graph
  - Fixed by reworking variant propagation not to track paths

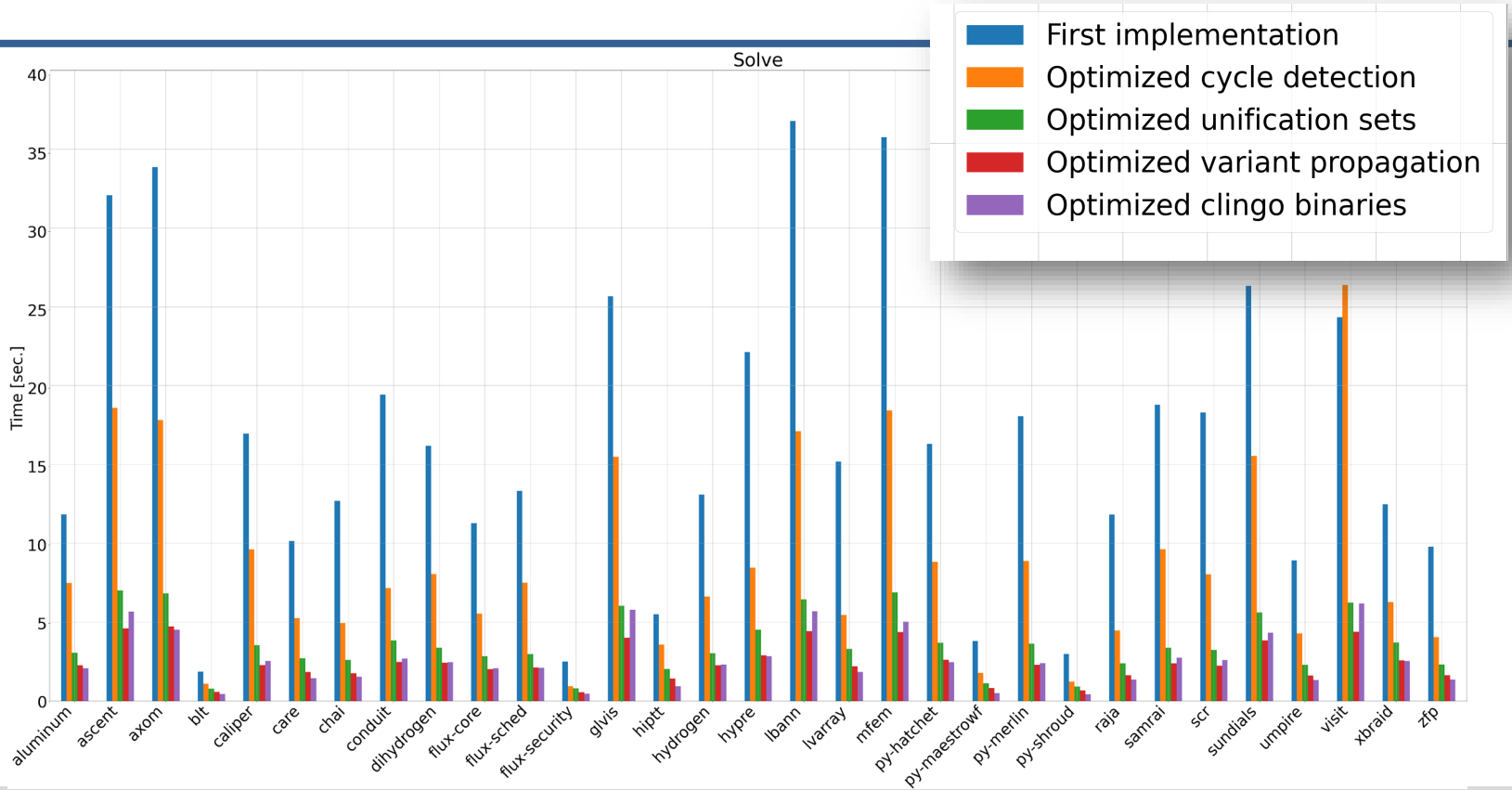
50%+ improvement in solve time

Unification sets can be expensive too

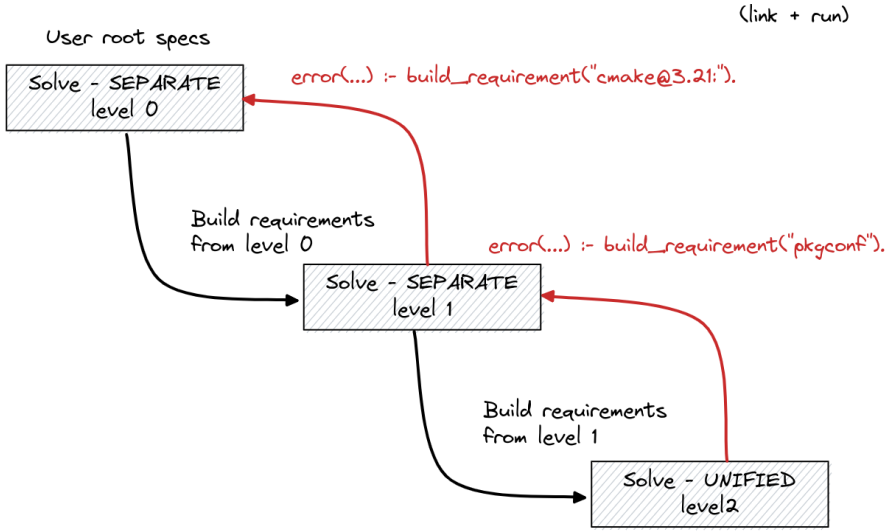


- Unification set creation was originally recursive for *any* build dependencies
  - Ends up blowing up grounding
- Mitigation:
  - Only create new sets for explicitly *marked* build tools
  - Transitive build dependencies that are not from marked build tools go into a *common* unification set
- Need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

Optimizations: Solve Time



It was not trivial to come up with this model



- In addition to this “coupled” method, 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 solve to solve was awkward
  - Packagers needed to “help” the solver
- Requiring packagers to provide solve hints in packages isn’t practical

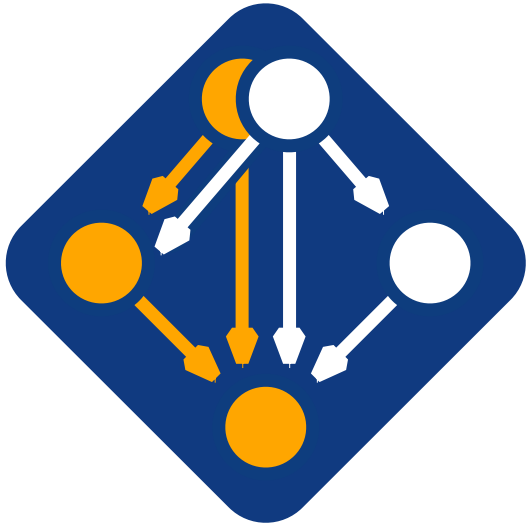
Recent Work

THE LINUX FOUNDATION

Spack v0.22.0 is out!

Highlights:

1. Compiler runtime dependencies
   - gcc-runtime, intel-oneapi-runtime, libgfortran, libc
   - OS compatibility on linux now uses libc version, not OS tag
2. Improved spack find UI for Environments
3. Improved command-line string quoting
4. Revert default spack install behavior to --reuse
5. More control over reused specs
6. New redistribute() directive
7. New conflict: and prefer: syntax for package preferences
8. include_concrete: in environments
9. python-venv isolation



github.com/spack/spack

Full release notes:
https://github.com/spack/spack/releases/tag/v0.22.0

We've made a lot of progress on compiler dependencies

- Compiler *runtime libraries* represented in the graph
  - C++, Fortran runtimes
- libc is now represented in dependency graphs on Linux
  - No more need to rely on OS tag for compatibility information
- Reuse binaries *without* their compiler needing to be configured locally
- Improved buildcache hit rate using libraries for compatibility

Compilers can now model their own runtimes

- New method, runtime_constraints, for injecting runtimes into graphs
- Currently supported for gcc, intel-oneapi
  - still working on others
- Allows solver to take libstdc++, fortran rutime compatibility into account.
- Example:
  - Intel compilers now (correctly) depend on gcc-runtime

```python
class Gcc(AutotoolsPackage, GNUMirrorPackage):
    # ...

    @classmethod
    def runtime_constraints(cls, *, spec, pkg):
        """Callback function to inject runtime-related rules into the solver.

        Rule-injection is obtained through method calls of the ``pkg`` argument.

        Documentation for this function is temporary. When the API will be in its final state,
        we'll document the behavior at https://spack.readthedocs.io/en/latest/

        Args:
            spec: spec that will inject runtime dependencies
            pkg: object used to forward information to the solver
        """
        pkg("*").depends_on(
            "gcc-runtime",
            when="%gcc",
            type="link",
            description="If any package uses %gcc, it depends on gcc-runtime",
        )
        pkg("*").depends_on(
            f"gcc-runtime@{str(spec.version)}:",
            when=f"%{str(spec)}",
            type="link",
            description=f"If any package uses %{str(spec)}, "
            f"it depends on gcc-runtime@{str(spec.version)}:",
        )
```

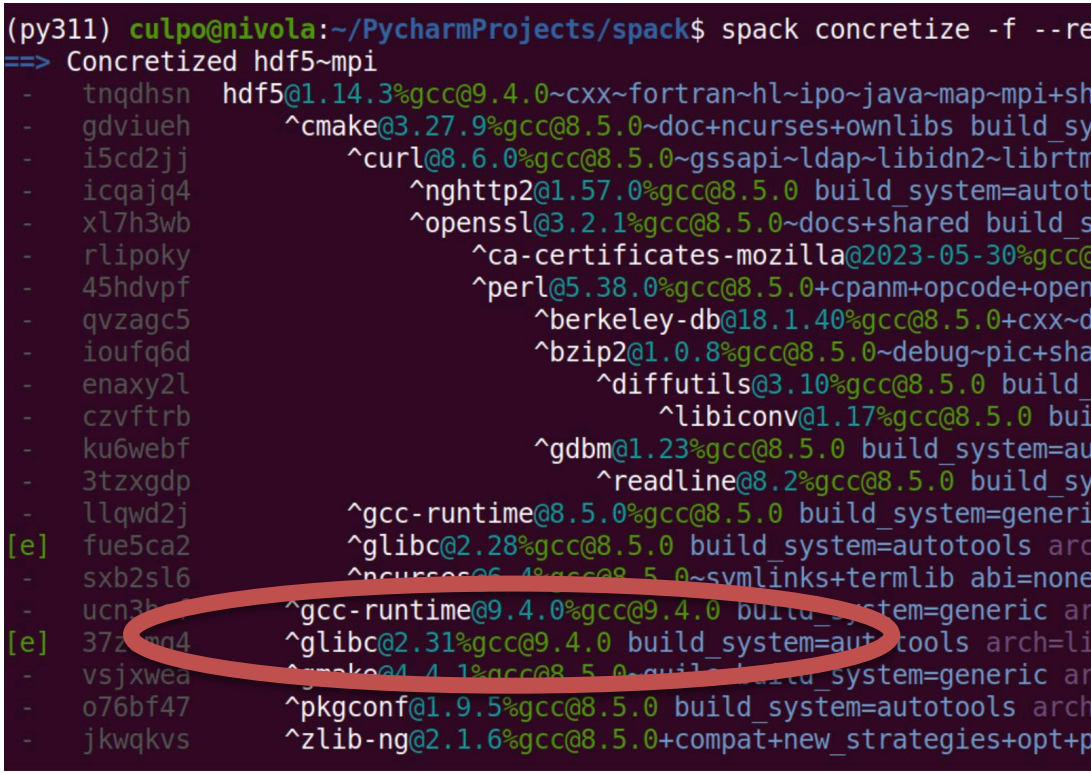
Packages now declare the languages they depend on

- Languages are *almost* virtuals
  - HDF5 package depends on cxx and fortran
  - Handled specially internally, until compilers are nodes
- Imply a compiler *and* compiler package can specify runtime libraries to inject
- Allows solver to mix compilers *correctly*
  - Runtimes are unified like other nodes
  - *Package authors* can model toolchain properties
  - probably not for most package authors, but very powerful
- TBD:
  - Other runtimes like clang libraries and OpenMP
  - Compilers as nodes in the graph

```
class Hdf5(CMakePackage):
    """HDF5 is a data model, library, and file format for storing and managi
    data. It supports an unlimited variety of datatypes, and is designed for
    flexible and efficient I/O and for high volume and complex data.
    """

    homepage = "https://portal.hdfgroup.org"
    url = "https://support.hdfgroup.org/ftp/HDF5/releases/hdf5-1.14/hdf5-1.1
    list_url = "https://support.hdfgroup.org/ftp/HDF5/releases"
    list_depth = 3
    git = "https://github.com/HDFGroup/hdf5.git"
    maintainers("lrknox", "brtnfld", "byrnHDF", "gheber", "hyoklee", "lkurz"

    tags = ["e4s", "windows"]
    executables = ["^h5cc$", "^h5pcc$"]

    test_requires_compiler = True

    license("custom")

    depends_on("cxx", type="build", when="+cxx")
    depends_on("fortran", type="build", when="+fortran")
```

We've also added libc as a dependency

- libc is a virtual
  - glibc and musl packages are providers
  - (nearly) every graph has libc in it, via the compiler
  - Can be external or built by Spack
- We are *not* building libc for every stack in Spack
  - Automatically detect system libc version
  - Add a node to the graph to be used for binary compatibility
- No longer using OS tags for buildcaches
  - Now use libc for this
  - *many* more buildcache hits



Libc modeling makes for a much better buildcache experience

- Currently on develop (emacs 100% from binary):

```
(py311) culpo@nivola:~/PycharmProjects/spack$ spack install emacs
[+] /usr (external glibc-2.17-2hhcy7kzv3wlfqcascwhvup4uysp4hoy)
[+] /home/culpo/PycharmProjects/spack/opt/spack/linux-centos7-x86_64_v3/gcc-10.2.1/gcc-runtime-10.2.1-4gmidou73wvttvhlun564olcopuijl2i
[+] /home/culpo/PycharmProjects/spack/opt/spack/linux-centos7-x86_64_v3/gcc-10.2.1/gmp-6.2.1-nkhm7cmp6samsykyksuqda77xbxnibfn
[+] /home/culpo/PycharmProjects/spack/opt/spack/linux-centos7-x86_64_v3/gcc-10.2.1/berkeley-db-18.1.40-thoi4z7lozgaednxhjd4ojhw2lcsgo5g
[+] /home/culpo/PycharmProjects/spack/opt/spack/linux-centos7-x86_64_v3/gcc-10.2.1/gmake-4.4.1-ucqstlhik7pmv5ijyckmr6mxv46vgeuj
==> Installing nasm-2.15.05-e2kvtqiaava2osr3jbyptdufjgov4dgc [6/39]
==> Fetching https://binaries.spack.io/develop/developer-tools-manylinux2014/build_cache/linux-centos7-x86_64_v3-gcc-10.2.1-nasm-2.15.05-e2kvtqiaava2osr3jbyptdufjgov4dgc.spec.json.sig
gpg: Signature made mar 23 apr 2024, 14:38:11 CEST
gpg:                using RSA key D2C7EB3F2B05FA86590D293C04001B2E3DB0C723
gpg: Good signature from "Spack Project Official Binaries <maintainers@spack.io>" [ultimate]
==> Fetching https://binaries.spack.io/develop/developer-tools-manylinux2014/build_cache/linux-centos7-x86_64_v3/gcc-10.2.1/nasm-2.15.05/linux-centos7-x86_64_v3-gcc-10.2.1-nasm-2.15.05-e2kvtqiaava2osr3jbyptd
==> Extracting nasm-2.15.05-e2kvtqiaava2osr3jbyptdufjgov4dgc from binary cache
==> nasm: Successfully installed nasm-2.15.05-e2kvtqiaava2osr3jbyptdufjgov4dgc
  Search: 0.00s.  Fetch: 2.61s.  Install: 0.17s.  Extract: 0.11s.  Relocate: 0.04s.  Total: 2.78s
[+] /home/culpo/PycharmProjects/spack/opt/spack/linux-centos7-x86_64_v3/gcc-10.2.1/nasm-2.15.05-e2kvtqiaava2osr3jbyptdufjgov4dgc
==> Installing pcre-8.45-xi42ks7asbq2sqmdc7bdsmhzzhlie6m4 [7/39]
==> Fetching https://binaries.spack.io/develop/developer-tools-manylinux2014/build_cache/linux-centos7-x86_64_v3-gcc-10.2.1-pcre-8.45-xi42ks7asbq2sqmdc7bdsmhzzhlie6m4.spec.json.sig
gpg: Signature made mar 23 apr 2024, 14:38:12 CEST
gpg:                using RSA key D2C7EB3F2B05FA86590D293C04001B2E3DB0C723
gpg: Good signature from "Spack Project Official Binaries <maintainers@spack.io>" [ultimate]
==> Fetching https://binaries.spack.io/develop/developer-tools-manylinux2014/build_cache/linux-centos7-x86_64_v3/gcc-10.2.1/pcre-8.45/linux-centos7-x86_64_v3-gcc-10.2.1-pcre-8.45-xi42ks7asbq2sqmdc7bdsmhzzhli
==> Extracting pcre-8.45-xi42ks7asbq2sqmdc7bdsmhzzhlie6m4 from binary cache
==> pcre: Successfully installed pcre-8.45-xi42ks7asbq2sqmdc7bdsmhzzhlie6m4
  Search: 0.00s.  Fetch: 2.34s.  Install: 0.18s.  Extract: 0.14s.  Relocate: 0.03s.  Total: 2.52s
[+] /home/culpo/PycharmProjects/spack/opt/spack/linux-centos7-x86_64_v3/gcc-10.2.1/pcre-8.45-xi42ks7asbq2sqmdc7bdsmhzzhlie6m4
==> Installing tree-sitter-0.22.2-x5wmc6tind5qxo2zrswb7vyj3xuhak2d [8/39]
==> Fetching https://binaries.spack.io/develop/developer-tools-manylinux2014/build_cache/linux-centos7-x86_64_v3-gcc-10.2.1-tree-sitter-0.22.2-x5wmc6tind5qxo2zrswb7vyj3xuhak2d.spec.json.sig
gpg: Signature made mar 23 apr 2024, 14:38:22 CEST
gpg:                using RSA key D2C7EB3F2B05FA86590D293C04001B2E3DB0C723
gpg: Good signature from "Spack Project Official Binaries <maintainers@spack.io>" [ultimate]
==> Fetching https://binaries.spack.io/develop/developer-tools-manylinux2014/build_cache/linux-centos7-x86_64_v3/gcc-10.2.1/tree-sitter-0.22.2/linux-centos7-x86_64_v3-gcc-10.2.1-tree-sitter-0.22.2-x5wmc6tind
==> Extracting tree-sitter-0.22.2-x5wmc6tind5qxo2zrswb7vyj3xuhak2d from binary cache
==> tree-sitter: Successfully installed tree-sitter-0.22.2-x5wmc6tind5qxo2zrswb7vyj3xuhak2d
```

Roadmap

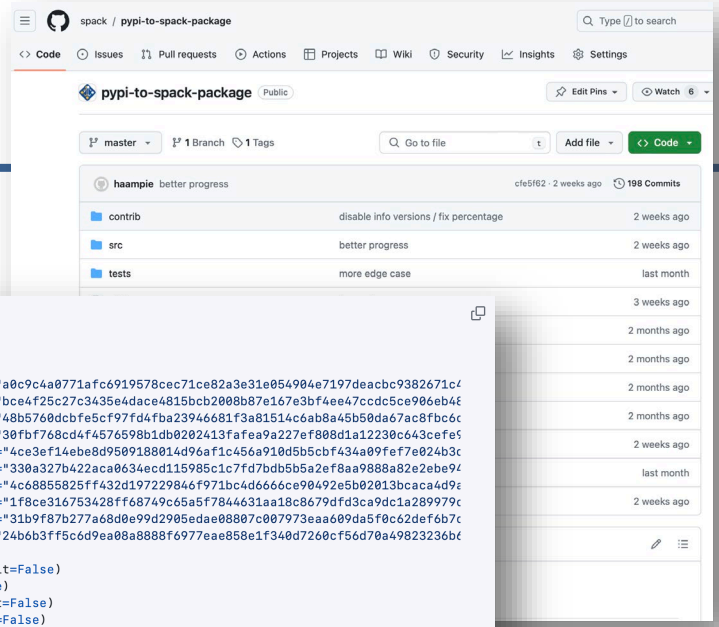
Roadmap for this year

1. Finish compiler dependencies
   - Compilers will appear as build dependencies
   - All special compiler logic generified for any build dependency
   - Enable bootstrapping compilers from buildcache
   - Continue to flesh out low-level compiler runtime libraries
2. Harden public build caches and make them more broadly compatible
3. Enable bare-metal MPI/CUDA/ROCm installations
4. Make the build cache on by default; build only what we need from source
5. Automatic Python package generation
6. Continue to improve Windows support
7. Speed improvements for concretization and metadata management

Lots of exciting work ahead!

Autogenerating Python packages

- Python ecosystem is a lot to maintain
  - Increasingly, we need to update faster than we can sustain through pull requests
  - Reviewing version ranges has been painful
- We added features to our version system to support Python better
  - Prerelease, alpha, beta version comparison
- We can now generate Python packages from PyPI metadata
  - Only for pure python packages
  - Native packages will be maintained as regular Spack packages
  - Overlay on top of auto-generated python packages




```
from spack.package import *


class PyBlack(PythonPackage):
    version("24.3.0", sha256="a0c9c4a0771afc6919578cec71ce82a3e31e054904e7197deacbc9382671c4
    version("24.2.0", sha256="bce4f25c27c3435e4dace4815bcb2008b87e167e3bf4ee47ccdc5ce906eb48
    version("24.1.1", sha256="48b5760dcbfe5cf97fd4fba23946681f3a81514c6ab8a45b50da67ac8fbc6c
    version("24.1.0", sha256="30fbf768cd4f4576598b1db0202413fafea9a227ef808d1a12230c643cefe9
    version("23.12.1", sha256="4ce3ef14ebe8d9509188014d96af1c456a910d5b5cbf434a09fef7e024b3c
    version("23.12.0", sha256="330a327b422aca0634ecd115985c1c7fd7bdb5b5a2ef8aa9888a82e2ebe94
    version("23.11.0", sha256="4c68855825ff432d197229846f971bc4d6666ce90492e5b02013bcaca4d9a
    version("23.10.1", sha256="1f8ce316753428ff68749c65a5f7844631aa18c8679dfd3ca9dc1a289979c
    version("23.10.0", sha256="31b9f87b277a68d0e99d2905edae08807c007973eaa609da5f0c62def6b7c
    version("23.9.1", sha256="24b6b3ff5c6d9ea08a8888f6977eae858e1f340d7260cf56d70a49823236b6

    variant("colorama", default=False)
    variant("d", default=False)
    variant("jupyter", default=False)
    variant("uvloop", default=False)

    with default_args(type="run"):
        depends_on("py-aiohttp@3.7.4:", when="@23.12:23.12.0,24:24.1.0 platform=linux")
        depends_on("py-aiohttp@3.7.4:", when="@23.12:23.12.0,24:24.1.0 platform=freebsd")
        depends_on("py-aiohttp@3.7.4:", when="@23.12:23.12.0,24:24.1.0 platform=darwin")
        depends_on("py-aiohttp@3.7.4:", when="@23.12:23.12.0,24:24.1.0 platform=cray")
        depends_on("py-aiohttp@3.7.4:", when="@21.10-beta0:21,22.10:+d")
        depends_on("py-click@8.0.0:", when="@22.10:")
        depends_on("py-colorama@0.4.3:", when="@20:21,22.10:+colorama")
        depends_on("py-ipython@7.8:", when="@21.8-beta0:21,22.10:+jupyter")
        depends_on("py-mypy-extensions@0.4.3:", when="@20:21,22.10:")
        depends_on("py-packaging@22:", when="@23.1.0:")
        depends_on("py-pathspec@0.9:", when="@22.10:")
        depends_on("py-platformdirs@2.0.0:", when="@21.8-beta0:21,22.10:")
        depends_on("py-tokenize-rt@3.2:", when="@21.8-beta0:21,22.10:+jupyter")
        depends_on("py-tomli@1.1:", when="@22.10: ^python@:3.10")
        depends_on("py-typing-extensions@4.0.1:", when="@23.9: ^python@:3.10")
        depends_on("py-uvloop@0.15.2:", when="@21.5-beta2:21,22.10:+uvloop")
```

Generating Python

```
$ ./src/package.py generate --clean spack_requirements.txt
```

- Had to rework version system to handle some frequent python conventions
  - Prereleases
  - Alpha, beta
  - Release candidate
- Now PyPI metadata can map to Spack depends_on() constraints
- Some caveats
  - Only doing this for pure python packages
  - Wheels (and PyPI database) lack build dependency info
  - May not be able to use spack develop with these

```
from spack.package import *


class PyBlack(PythonPackage):
    version("24.3.0", sha256="a0c9c4a0771afc6919578cec71ce82a3e31e054904e7197deacbc9382671c4
    version("24.2.0", sha256="bce4f25c27c3435e4dace4815bcb2008b87e167e3bf4ee47ccdc5ce906eb48
    version("24.1.1", sha256="48b5760dcbfe5cf97fd4fba23946681f3a81514c6ab8a45b50da67ac8fbc6c
    version("24.1.0", sha256="30fbf768cd4f4576598b1db0202413fafea9a227ef808d1a12230c643cefe9
    version("23.12.1", sha256="4ce3ef14ebe8d9509188014d96af1c456a910d5b5cbf434a09fef7e024b3c
    version("23.12.0", sha256="330a327b422aca0634ecd115985c1c7fd7bdb5b5a2ef8aa9888a82e2ebe94
    version("23.11.0", sha256="4c68855825ff432d197229846f971bc4d6666ce90492e5b02013bcaca4d9a
    version("23.10.1", sha256="1f8ce316753428ff68749c65a5f7844631aa18c8679dfd3ca9dc1a289979c
    version("23.10.0", sha256="31b9f87b277a68d0e99d2905edae08807c007973eaa609da5f0c62def6b7c
    version("23.9.1", sha256="24b6b3ff5c6d9ea08a8888f6977eae858e1f340d7260cf56d70a49823236b6

    variant("colorama", default=False)
    variant("d", default=False)
    variant("jupyter", default=False)
    variant("uvloop", default=False)

    with default_args(type="run"):
        depends_on("py-aiohttp@3.7.4:", when="@23.12:23.12.0,24:24.1.0 platform=linux")
        depends_on("py-aiohttp@3.7.4:", when="@23.12:23.12.0,24:24.1.0 platform=freebsd")
        depends_on("py-aiohttp@3.7.4:", when="@23.12:23.12.0,24:24.1.0 platform=darwin")
        depends_on("py-aiohttp@3.7.4:", when="@23.12:23.12.0,24:24.1.0 platform=cray")
        depends_on("py-aiohttp@3.7.4:", when="@21.10-beta0:21,22.10:+d")
        depends_on("py-click@8.0.0:", when="@22.10:")
        depends_on("py-colorama@0.4.3:", when="@20:21,22.10:+colorama")
        depends_on("py-ipython@7.8:", when="@21.8-beta0:21,22.10:+jupyter")
        depends_on("py-mypy-extensions@0.4.3:", when="@20:21,22.10:")
        depends_on("py-packaging@22:", when="@23.1.0:")
        depends_on("py-pathspec@0.9:", when="@22.10:")
        depends_on("py-platformdirs@2.0.0:", when="@21.8-beta0:21,22.10:")
        depends_on("py-tokenize-rt@3.2:", when="@21.8-beta0:21,22.10:+jupyter")
        depends_on("py-tomli@1.1:", when="@22.10: ^python@:3.10")
        depends_on("py-typing-extensions@4.0.1:", when="@23.9: ^python@:3.10")
        depends_on("py-uvloop@0.15.2:", when="@21.5-beta2:21,22.10:+uvloop")
```

When would we go to “Version 1.0”?

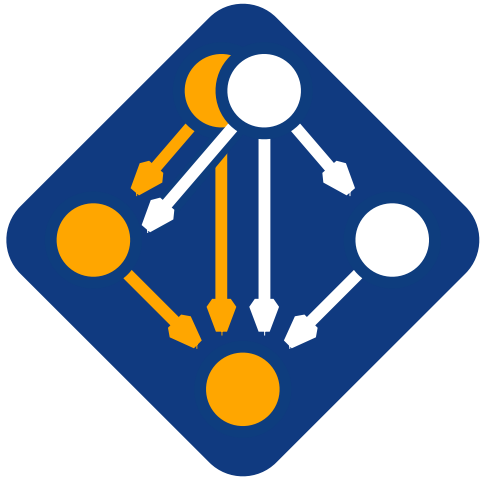
Big things we’ve wanted for 1.0 are:

- New concretizer — **Done!**
- production CI — **Done!**
- production public build cache — **Done!**
- Compilers as dependencies — **Aiming for November**
- Buildcache hardening — **Aiming for November**
- Stable package API — **Aiming for June**
  - Enables separate package repository — **Aiming for June**

We are getting very close!

Join the Spack community!

- Join us and 3,200 others on **slack.spack.io**
- Contribute packages, documentation, and features on **GitHub**
- Continue the tutorial at **spack-tutorial.rtfd.io**



slack.spack.io

Star us on GitHub!

github.com/spack/spack

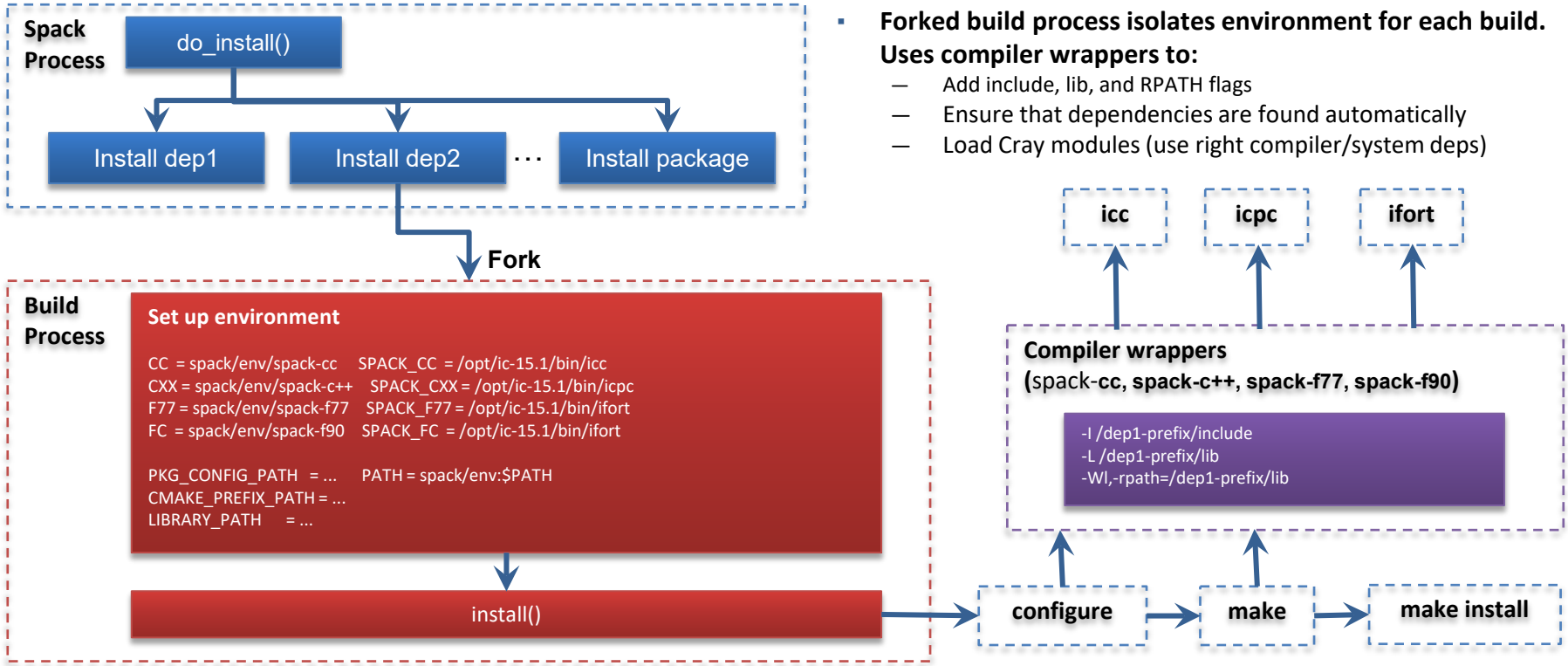
@spackpm

We hope to make distributing & using HPC software easy!

LLNL-PRES-806064

THE LINUX FOUNDATION

An isolated compilation environment allows Spack to easily swap compilers



- **Forked build process isolates environment for each build.**
  Uses compiler wrappers to:
  - Add include, lib, and RPATH flags
  - Ensure that dependencies are found automatically
  - Load Cray modules (use right compiler/system deps)

OCI build caches

- Spack can now use arbitrary OCI registries as buildcaches

```
$ spack mirror add my_registry oci://user/image                                  # Dockerhub
$ spack mirror add my_registry oci://ghcr.io/haampie/spack-test          # GHCR
$ spack mirror set --push --oci-username ... --oci-password ... my_registry  # set login
$ spack buildcache push my_registry [specs...]$ spack solve hdf5 ^cmake@3.0.1
```

- And you can optionally add a base image to get *runnable* images

```
$ spack buildcache push --base-image ubuntu:23.04 my_registry python
Pushed ... as [image]:python-3.11.2-65txfcpqbmpawclvtasuog4yzmxwaoia.spack

$ docker run --rm -it [image]:python-3.11.2-65txfcpqbmpawclvtasuog4yzmxwaoia.spack
```

ASP has enabled us to improve Spack's error messages

```
$ spack solve hdf5 ^cmake@3.0.1

==> Error: concretization failed for the following reasons:

1. Cannot satisfy 'cmake@3.0.1'
2. Cannot satisfy 'cmake@3.0.1'
      required because hdf5 ^cmake@3.0.1 requested from CLI
3. Cannot satisfy 'cmake@3.18:' and 'cmake@3.0.1
      required because hdf5 ^cmake@3.0.1 requested from CLI
        required because hdf5 depends on cmake@3.18: when @1.13:
      required because hdf5 ^cmake@3.0.1 requested from CLI
4. Cannot satisfy 'cmake@3.12:' and 'cmake@3.0.1
      required because hdf5 depends on cmake@3.12:
        required because hdf5 ^cmake@3.0.1 requested from CLI
      required because hdf5 ^cmake@3.0.1 requested from CLI
```

Spack's solver uses "generalized" conditions in ASP

- ASP rules: when the right side is true, the left is true

```
variant_value(Pkg, Variant, Value) :- variant_set(Pkg, Variant, Value), node(Pkg).
```

- Generalized conditions:
  - When condition requirements are True, the condition holds
  - When the condition holds, the imposed constraints are attributes of the spec

```
condition_holds(ID).
  :- attr(Name, A1), : condition_requirement(Name, A1);
     attr(Name, A1 , A2), : condition_requirement(Name, A1, A1).
attr(Name, A1) :- condition_holds(ID), imposed_constraint(ID, Name, A1).
attr(Name, A1, A2) :- condition_holds(ID), imposed_constraint(ID, Name, A1, A2).
```

- This is how all conditional logic in Spack is implemented

We now build chains of causes and effects from the error message back to other sources of the problem

```
condition_cause(Effect, Cause) :-
  condition_holds(Effect),
  condition_holds(Cause),
  attr(Name, A1),
  condition_requirement(Effect, Name, A1),
  imposed_constraint(Cause, Name, A1, A2).
```

Arity 1

```
condition_cause(Effect, Cause) :-
  condition_holds(Effect),
  condition_holds(Cause),
  attr(Name, A1, A2),
  condition_requirement(Effect, Name, A1, A2),
  imposed_constraint(Cause, Name, A1, A2).
```

Arity 2

etc.