Spack: Package management for HPC

ATPESC 2023
St Charles, Illinois
August 4, 2023

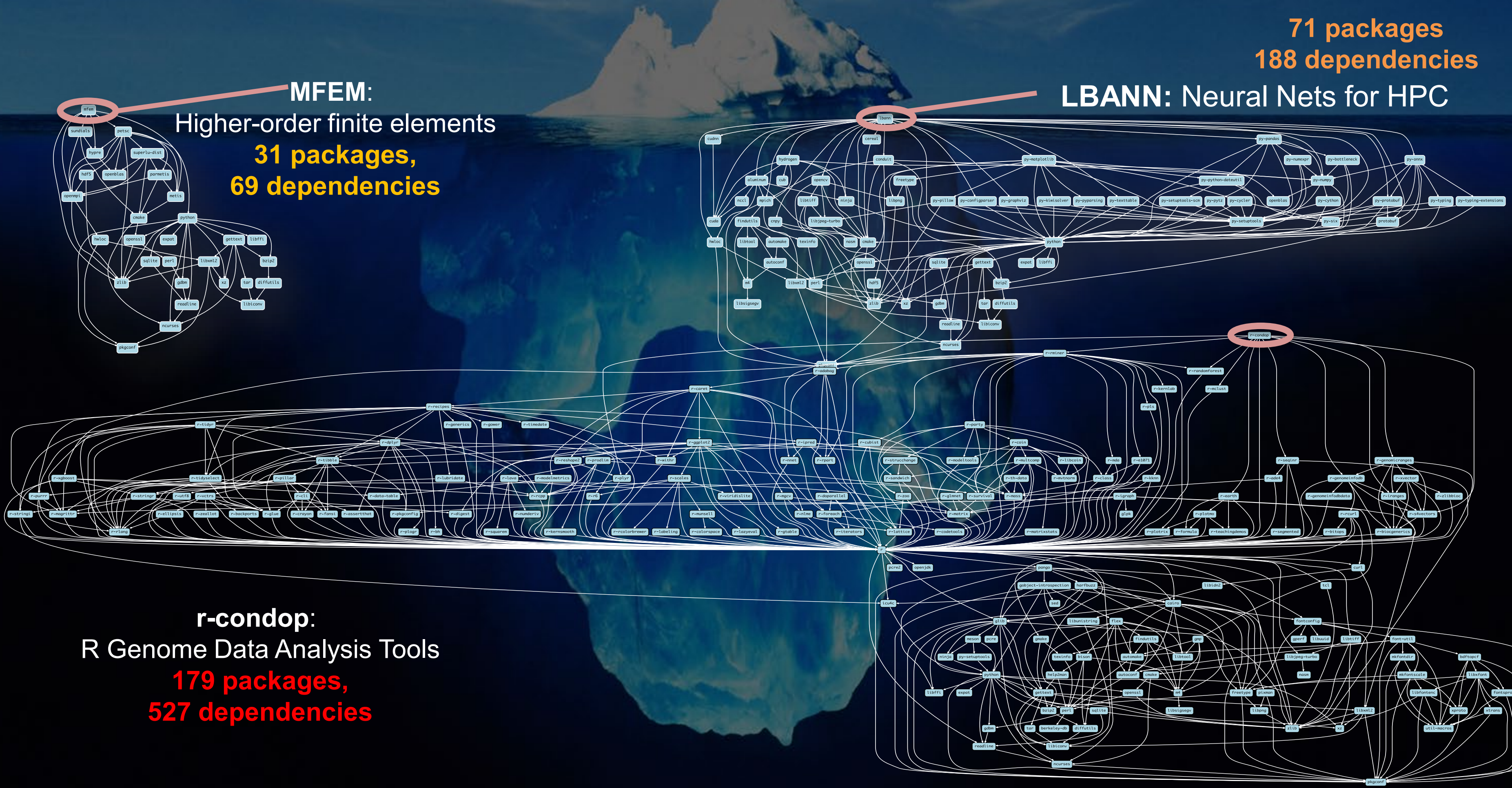
LLNL-PRES-806064

This work was performed under the auspices of the U.S. Department of Energy by Lawrence Livermore National Laboratory under contract DE-AC52-07NA27344. Lawrence Livermore National Security, LLC

spack.io

Lawrence Livermore National Laboratory

Modern scientific codes rely on icebergs of dependency libraries

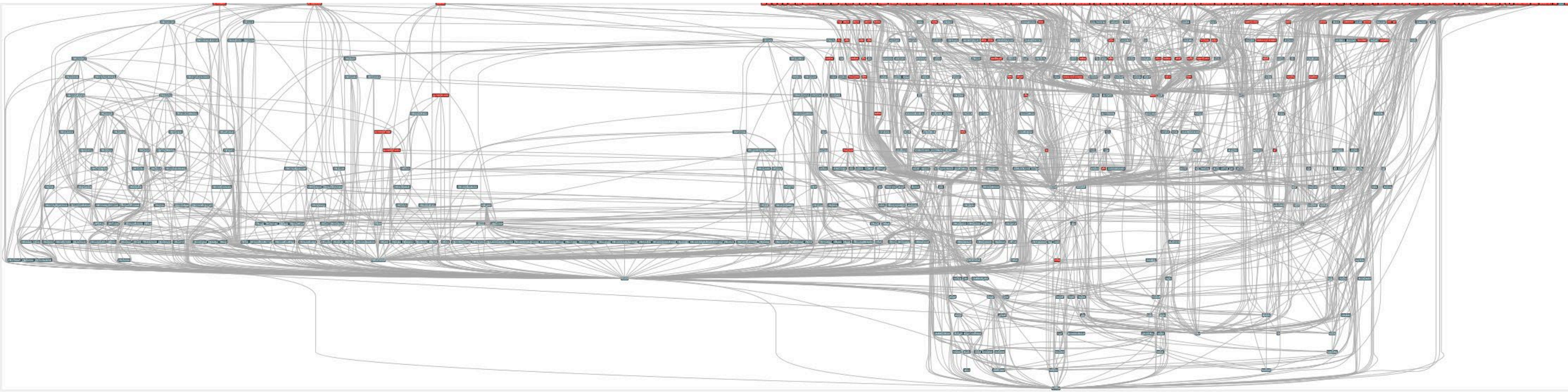


MFEM:
Higher-order finite elements
31 packages,
69 dependencies

71 packages
188 dependencies
LBANN: Neural Nets for HPC

r-condop:
R Genome Data Analysis Tools
179 packages,
527 dependencies

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 fairly common (but questionable) assumptions made by package managers (conda, pip, apt, etc.)

- **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)

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 may need to be *rebuilt* to support many different hosts, anyway.**
  - Not clear that we can ever build one container for all facilities
  - Containers likely won't solve the N-platforms problem in HPC

docker

Charliecloud

SHIFTER

We need something more flexible to **build** the containers

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 %clang@6.0
$ spack install hdf5@1.10.5 +threadssafe
$ spack install hdf5@1.10.5 cppflags="-O3 –g3"
$ spack install hdf5@1.10.5 target=haswell
$ 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)

What’s a package manager?

- Spack is a ***package manager***
  - **Does not** a replace Cmake/Autotools
  - Packages built by Spack can have any build system they want
- Spack manages ***dependencies***
  - Drives package-level build systems
  - Ensures consistent builds
- Determining magic configure lines takes time
  - Spack is a cache of recipes

Package Manager

- Manages package installation
- Manages dependency relationships
- May drive package-level build systems

High Level Build System

- Cmake, Autotools
- Handle library abstractions
- Generate Makefiles, etc.

Low Level Build System

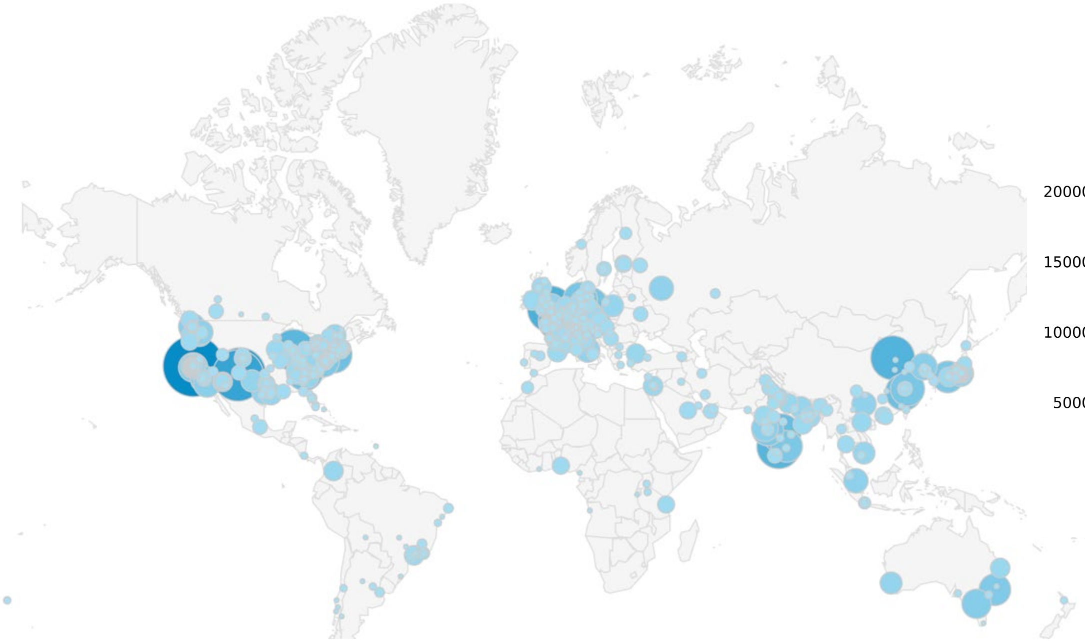
- Make, Ninja
- Handles dependencies among *commands* in a single build

Who can use Spack?

People who want to use or distribute software for HPC!

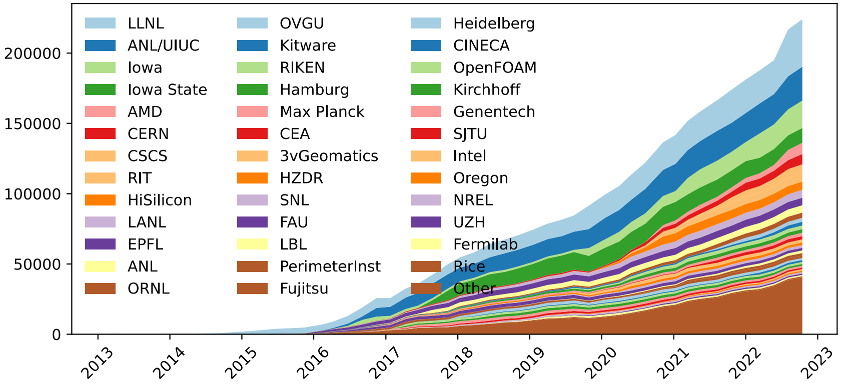
1. **End Users of HPC Software**
   - Install and run HPC applications and tools
2. **HPC Application Teams**
   - Manage third-party dependency libraries
3. **Package Developers**
   - People who want to package their own software for distribution
4. **User support teams at HPC Centers**
   - People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

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



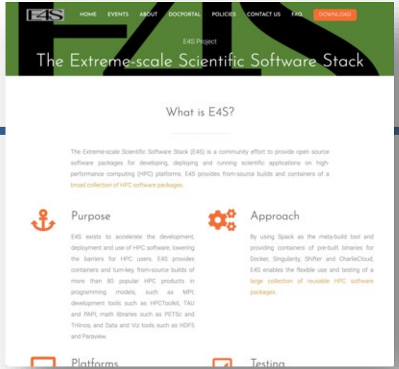
Over 6,900 software packages
Over 1,100 contributors

Contributions (lines of code) over time in packages, by organization



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

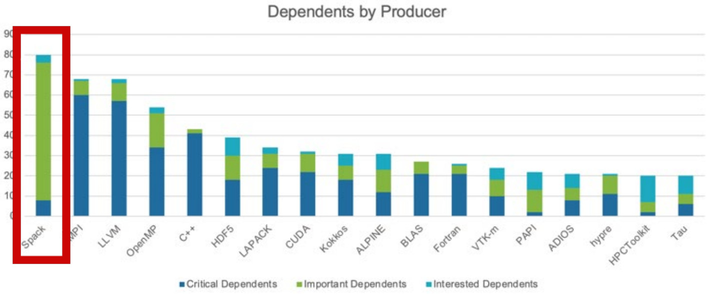
Spack is critical for ECP's mission to create a robust, capable exascale software ecosystem.



EXASCALE COMPUTING PROJECT

https://e4s.io

- Spack will be used to build software for the three upcoming U.S. exascale systems
- ECP has built the Extreme Scale Scientific Software Stack (E4S) with Spack – more at https://e4s.io
- Spack will be integral to upcoming ECP testing efforts.



Spack is the most depended-upon project in ECP

One month of Spack development is pretty busy!

April 21, 2023 – May 21, 2023

Period: 1 month ▾

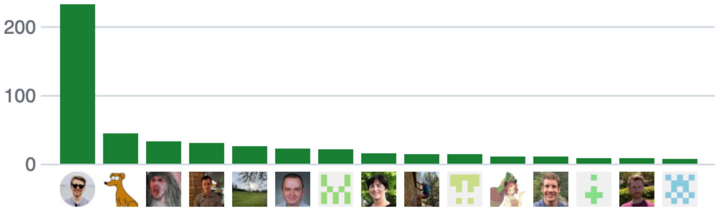
Overview

703 Active pull requests

145 Active issues

568	**135**	**99**	**46**
Merged pull requests	Open pull requests	Closed issues	New issues

Excluding merges, **109 authors** have pushed **568 commits** to develop and **625 commits** to all branches. On develop, **1,228 files** have changed and there have been **33,421 additions** and **17,043 deletions**.



Spack's widespread adoption has drawn contributions and collaborations with many vendors

- **AWS** invests significantly in cloud credits for Spack build farm
  - Joint Spack tutorial with AWS had 125+ participants
  - Joint AWS/AHUG Spack Hackathon drew 60+ participants
- **AMD** has contributed ROCm packages and compiler support
  - 55+ PRs mostly from AMD, also others
  - ROCm, HIP, aocc packages are all in Spack now
- **HPE/Cray** is doing internal CI for Spack packages, in the Cray environment
- **Intel** contributing OneApi support and licenses for our build farm
- **NVIDIA** contributing NVHPC compiler support and other features
- **Fujitsu and RIKEN** have contributed a **huge** number of packages for ARM/a64fx support on Fugaku
- **ARM** and **Linaro** members contributing ARM support
  - 400+ pull requests for ARM support from various companies

Spack v0.20.0 was released at ISC23!

Major new features:

1. requires() directive, enhanced package requirements
2. Exact versions with @=
3. New testing interface
4. More stable concretization
5. Weekly develop snapshot releases
6. Specs in buildcaches can be referenced by hash
7. New package and buildcache index websites
8. Default CMake and Meson build types are now Release

github.com/spack/spack

Full release notes:

https://github.com/spack/spack/releases/tag/v0.20.0

Spack is not the only tool that automates builds

1. **"Functional" Package Managers**
   - Nix — https://nixos.org/
   - Guix — https://www.gnu.org/s/guix/

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

Other tools in the HPC Space:

- **Easybuild** — http://hpcugent.github.io/easybuild/
  - 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** — https://conda.io
  - Very popular binary package ecosystem for data science
  - Not targeted at HPC; generally has unoptimized binaries

Most existing tools do not support combinatorial versioning

- Traditional binary package managers
  - RPM, yum, APT, yast, etc.
  - Designed to manage a single stack.
  - Install *one* version of each package in a single prefix (/usr).
  - Seamless upgrades to a *stable, well tested* stack

- Port systems
  - BSD Ports, portage, Macports, Homebrew, Gentoo, etc.
  - Minimal support for builds parameterized by compilers, dependency versions.

- Virtual Machines and Linux Containers (Docker)
  - Containers allow users to build environments for different applications.
  - Does not solve the build problem (someone has to build the image)
  - Performance, security, and upgrade issues prevent widespread HPC deployment.

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

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

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

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

Spack packages are *parameterized* using the spec syntax

Python DSL defines many ways to build

```
from spack import *

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

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

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

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

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

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

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

Base package (CMake support)

Metadata at the class level

Versions

Variants (build options)

Dependencies (same spec syntax)

Install logic in instance methods

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

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

Conditional variants simplify packages

CudaPackage: a mix-in for packages that use CUDA

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

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

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

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

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

cuda is a variant (build option)

cuda_arch is only present if cuda is enabled

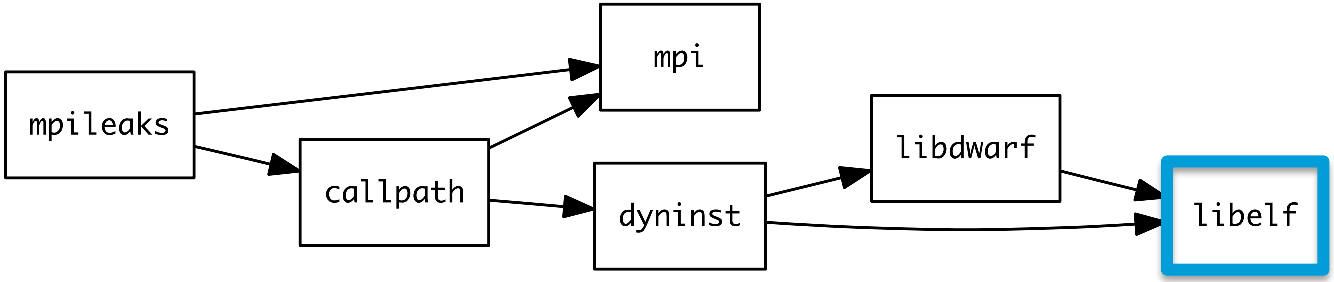
dependency on cuda, but only if cuda is enabled

constraints on cuda version

compiler support for x86_64 and ppc64le

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

Spack Specs can constrain versions of dependencies

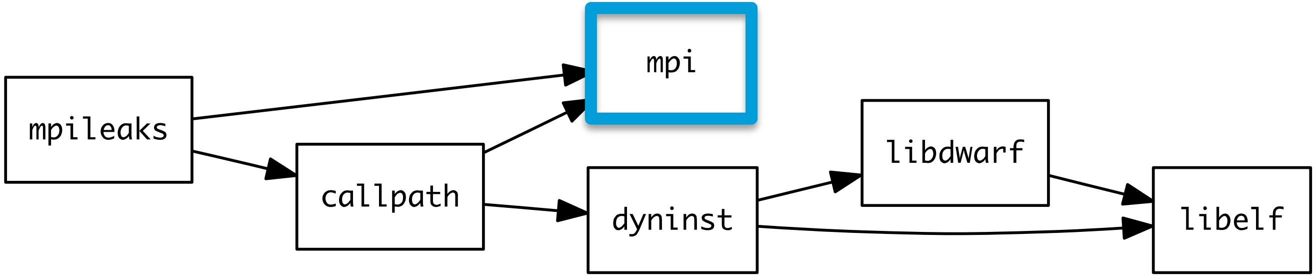



```
$ spack install mpileaks %intel@12.1 ^libelf@0.8.12
```

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

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

Spack handles ABI-incompatible, versioned interfaces like MPI



- mpi is a *virtual dependency*

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

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

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

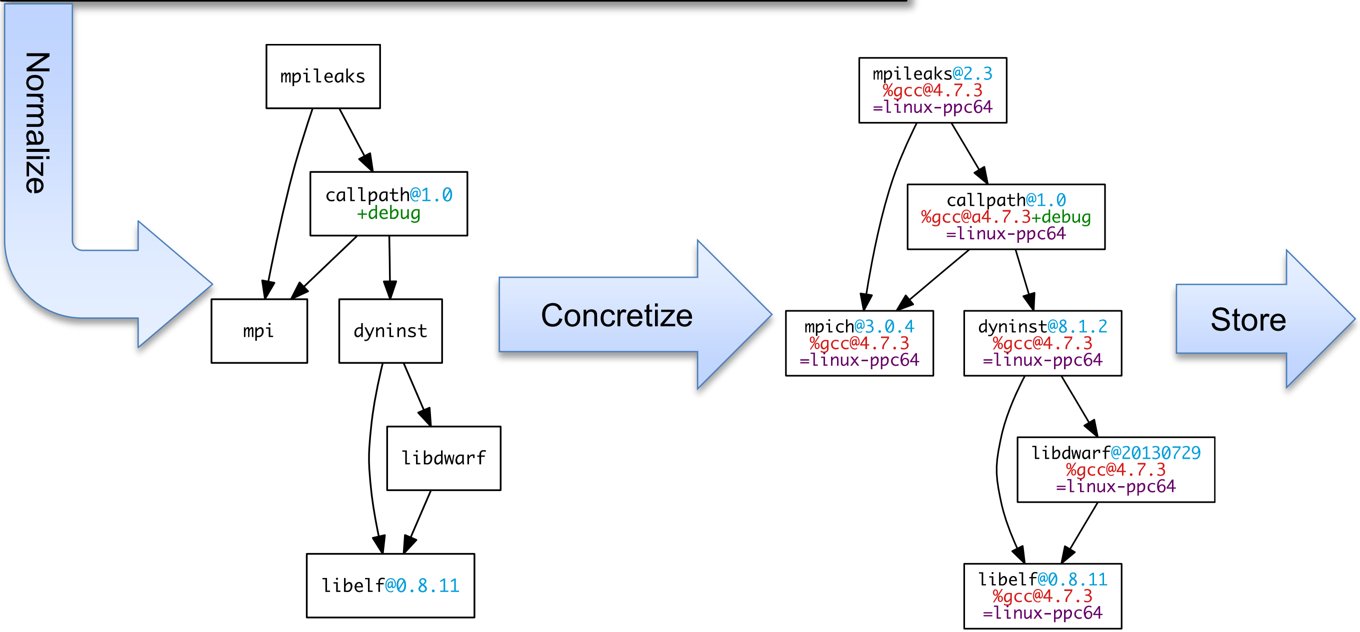
- Let Spack choose MPI implementation, as long as it provides MPI 2 interface:

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

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

mpileaks ^callpath@1.0+debug ^libelf@0.8.11

User input: *abstract* spec with some constraints



spec.yaml


```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
...
```

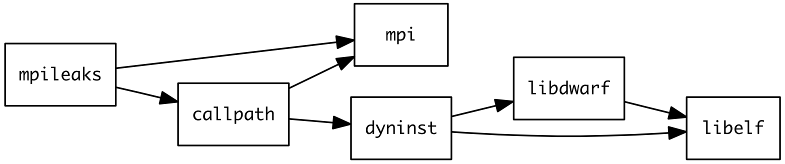

Abstract, normalized spec with some dependencies.

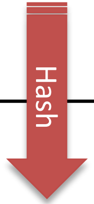
Concrete spec is fully constrained and can be passed to install.

Detailed provenance is stored with the installed package

Hashing allows us to handle combinatorial complexity

Dependency DAG



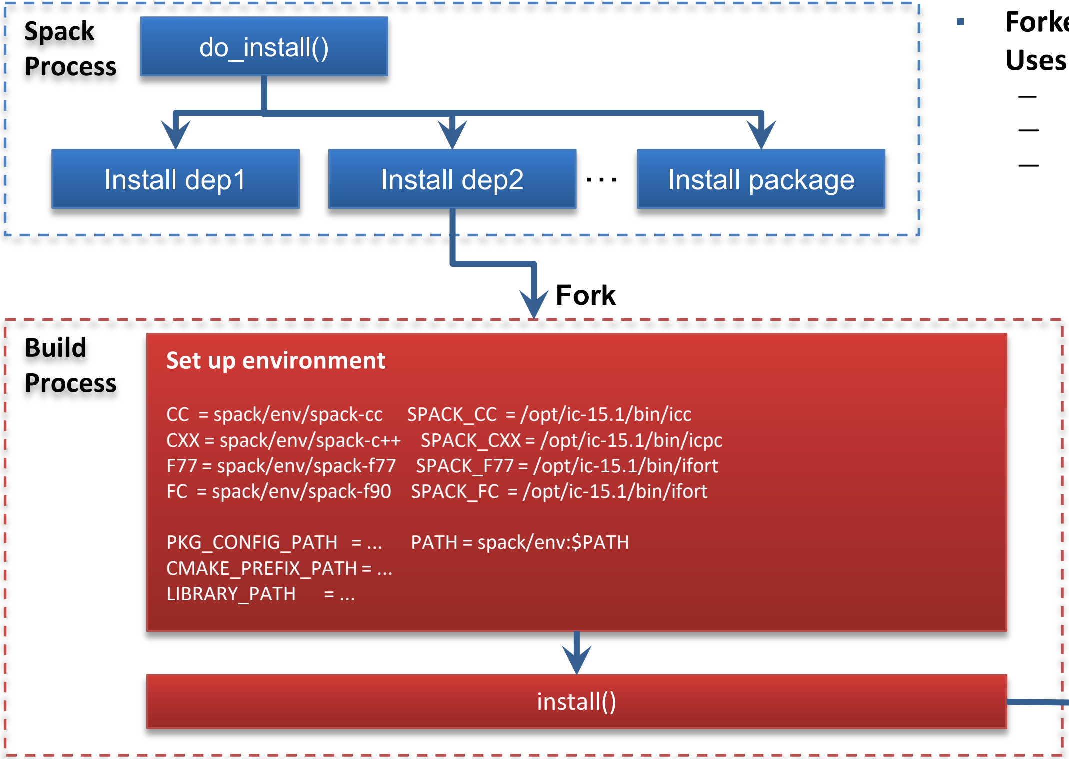


Installation Layout

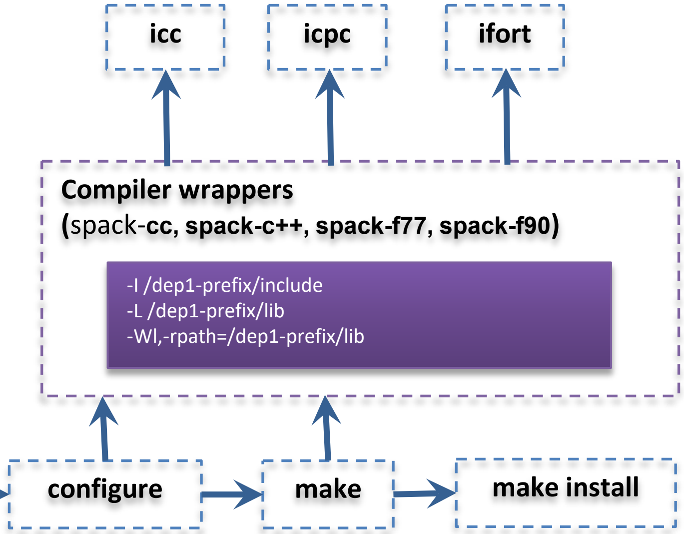
```
opt
└── spack
    ├── darwin-mojave-skylake
    │   └── clang-10.0.0-apple
    │       ├── bzip2-1.0.8-hc4sm4vuzpm4znmvrfzri4ow2mkphe2e
    │       ├── python-3.7.6-daqqpssxb6qbfrztsezkmhus3xoflbsy
    │       ├── sqlite-3.30.1-u64v26igxvxyn23hysmklfums6tgjv5r
    │       ├── xz-5.2.4-u5eawkvaoc7vonabe6nndkcfwuv233cj
    │       └── zlib-1.2.11-x46q4wm46ay4pltriijbgizxjrhbaka6
    ├── darwin-mojave-x86_64
    │   └── clang-10.0.0-apple
    │       └── coreutils-8.29-pl2kcytejqcys5dzecfrtjqxfdssvnob
```

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

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)

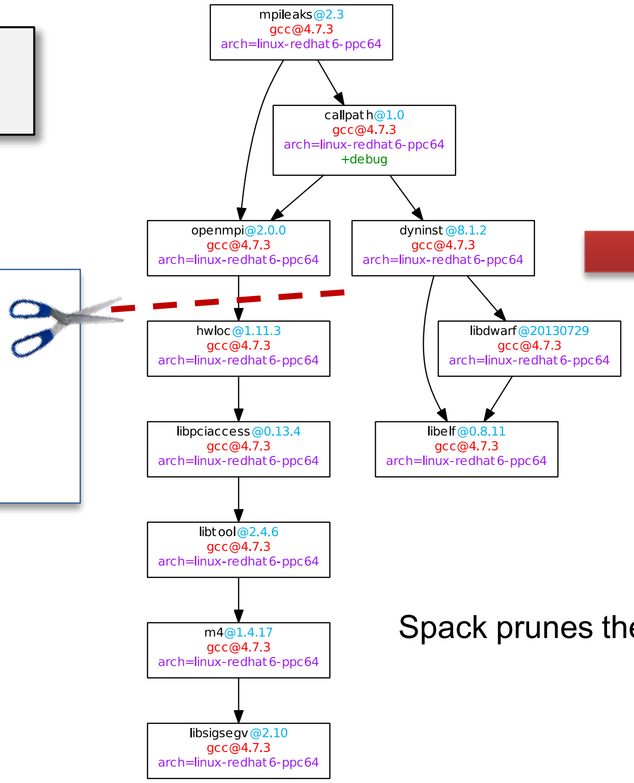


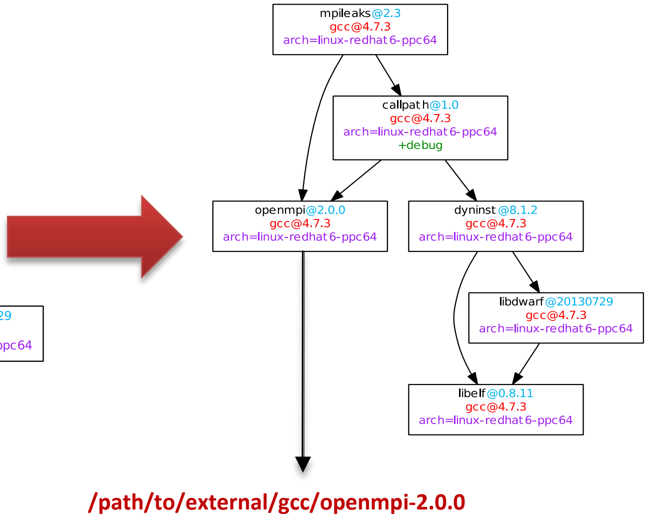
We can configure Spack to build with external software

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

packages.yaml

```
packages:
  mpi:
    buildable: False
    paths:
      openmpi@2.0.0 %gcc@4.7.3 arch=linux-rhel6-ppc64:
        /path/to/external/gcc/openmpi-2.0.0
      openmpi@1.10.3 %gcc@4.7.3 arch=linux-rhel6-ppc64:
        /path/to/external/gcc/openmpi-1.10.3
      ...
```



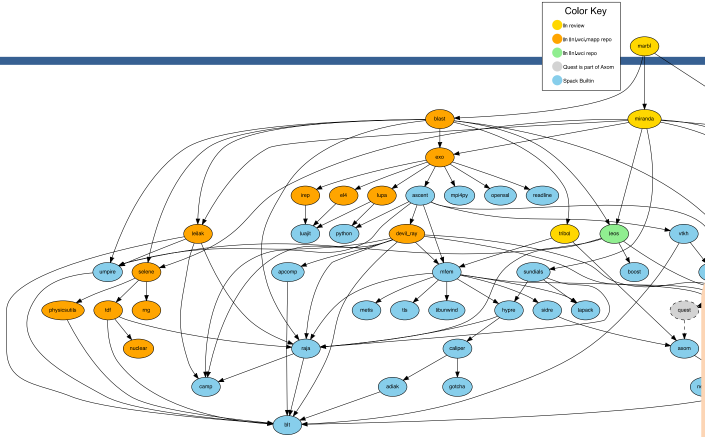


Users register external packages in a configuration file (more on these later).

Spack prunes the DAG when adding external packages.

Spack package repositories allow stacks to be layered

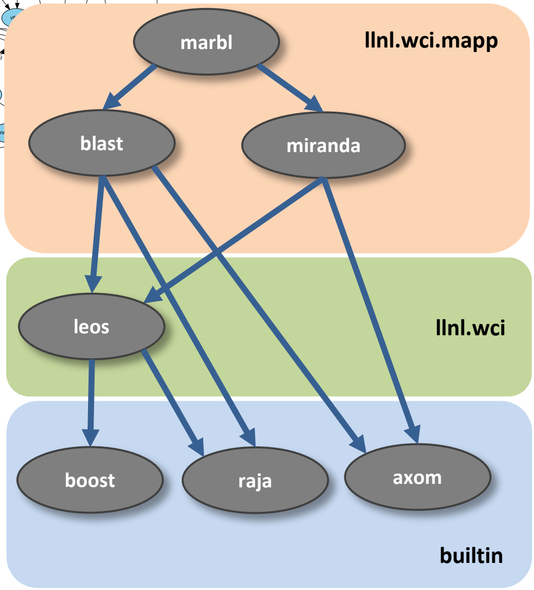
LLNL MARBL multi-physics application



MARBL Application Packages

Common internal packages

Open Source Spack packages



$ spack repo create /path/to/my_repo
$ spack repo add my_repo
$ spack repo list
==> 2 package repositories.
my_repo   /path/to/my_repo
builtin   spack/var/spack/repos/builtin

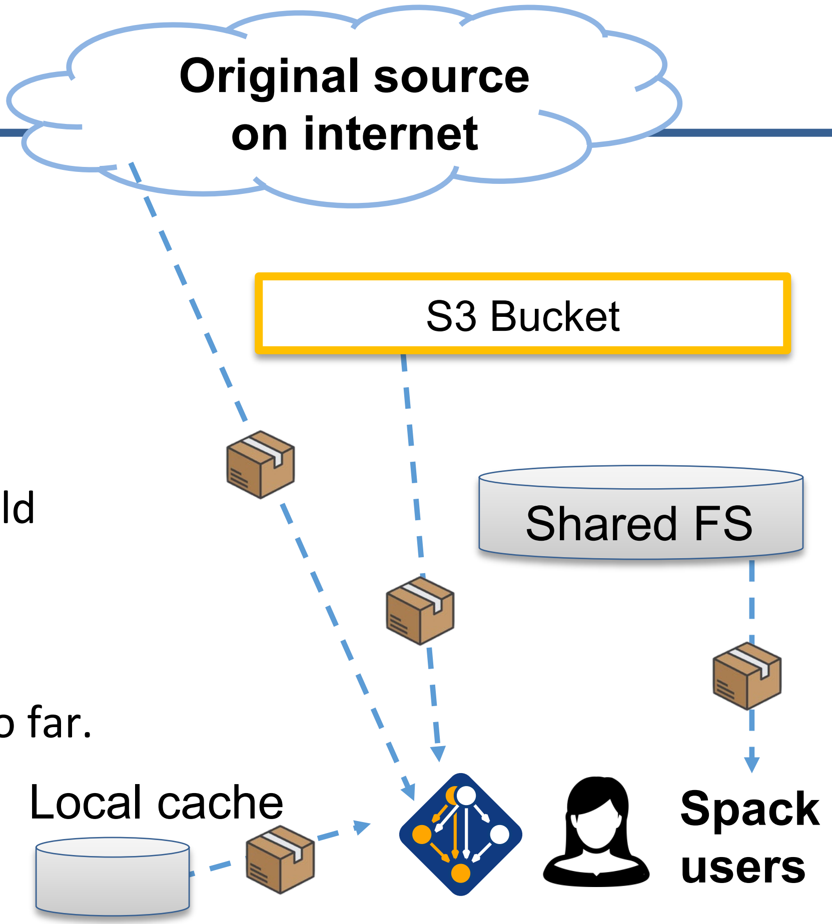
Spack mirrors

- Spack allows you to define *mirrors:*
  - Directories in the filesystem
  - On a web server
  - In an S3 bucket

- Mirrors are archives of fetched tarballs, repositories, and other resources needed to build
  - Can also contain binary packages

- By default, Spack maintains a mirror in var/spack/cache of everything you've fetched so far.

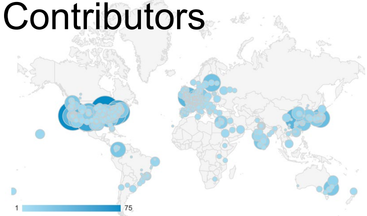
- You can host mirrors internal to your site
  - See the documentation for more details



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

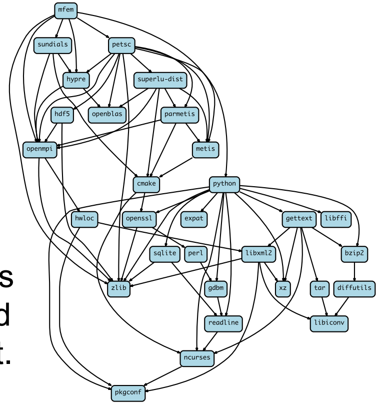
admins, users — yaml — local preferences config packages.yaml

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.



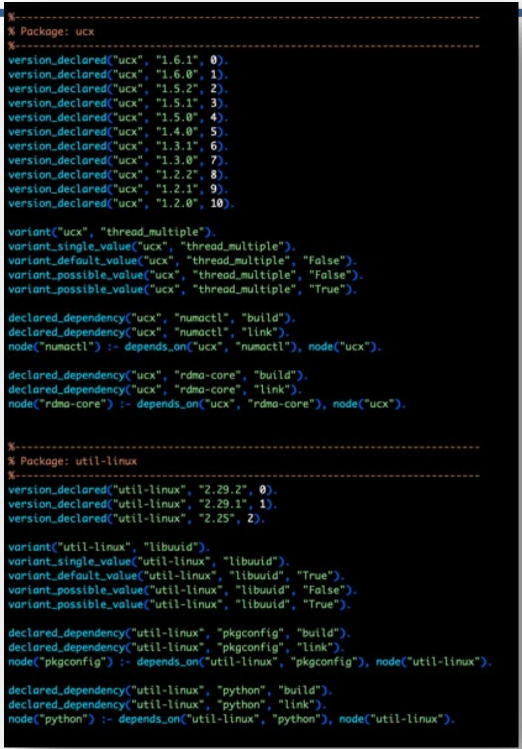
We use logic programming to simplify package solving

- New concretizer leverages Clingo (see potassco.org)

- Clingo is an Answer Set Programming (ASP) solver
  - ASP looks like Prolog; leverages SAT solvers for speed/correctness
  - ASP program has 2 parts:
    1. Large list of facts generated from our package repositories and config
    2. Small logic program (~800 lines)
       - includes constraints and optimization criteria

- New algorithm on the Spack side is conceptually simpler:
  - Generate facts for all possible dependencies, send to logic program
  - Optimization criteria express preferences more clearly
  - Build a DAG from the results

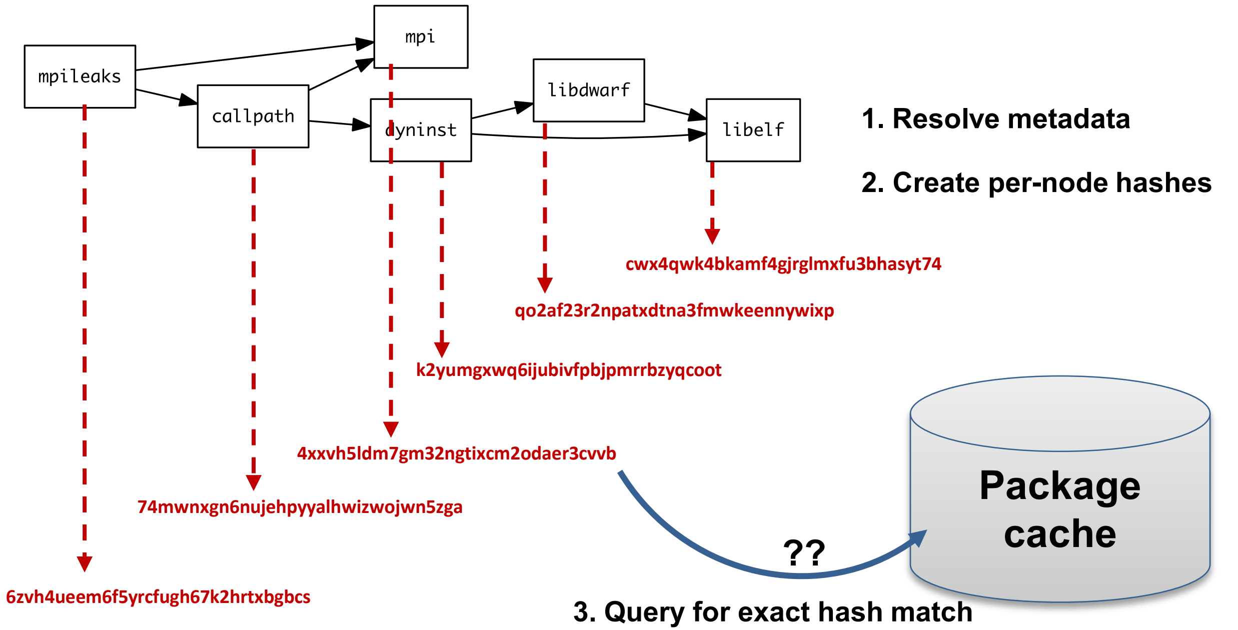
- New concretizer solves many specs that old concretizer can't
  - Backtracking is a huge win – many issues resolved
  - Conditional logic that was complicated before is now much easier



```
%----------------------------------------------------------------------------
% Package: ucx
%----------------------------------------------------------------------------
version_declared("ucx", "1.6.1", 0).
version_declared("ucx", "1.6.0", 1).
version_declared("ucx", "1.5.2", 2).
version_declared("ucx", "1.5.1", 3).
version_declared("ucx", "1.5.0", 4).
version_declared("ucx", "1.4.0", 5).
version_declared("ucx", "1.3.1", 6).
version_declared("ucx", "1.3.0", 7).
version_declared("ucx", "1.2.2", 8).
version_declared("ucx", "1.2.1", 9).
version_declared("ucx", "1.2.0", 10).

variant("ucx", "thread_multiple").
variant_single_value("ucx", "thread_multiple").
variant_default_value("ucx", "thread_multiple", "False").
variant_possible_value("ucx", "thread_multiple", "False").
variant_possible_value("ucx", "thread_multiple", "True").

declared_dependency("ucx", "numactl", "build").
declared_dependency("ucx", "numactl", "link").
node("numactl") :- depends_on("ucx", "numactl"), node("ucx").

declared_dependency("ucx", "rdma-core", "build").
declared_dependency("ucx", "rdma-core", "link").
node("rdma-core") :- depends_on("ucx", "rdma-core"), node("ucx").


%----------------------------------------------------------------------------
% Package: util-linux
%----------------------------------------------------------------------------
version_declared("util-linux", "2.29.2", 0).
version_declared("util-linux", "2.29.1", 1).
version_declared("util-linux", "2.25", 2).

variant("util-linux", "libuuid").
variant_single_value("util-linux", "libuuid").
variant_default_value("util-linux", "libuuid", "True").
variant_possible_value("util-linux", "libuuid", "False").
variant_possible_value("util-linux", "libuuid", "True").

declared_dependency("util-linux", "pkgconfig", "build").
declared_dependency("util-linux", "pkgconfig", "link").
node("pkgconfig") :- depends_on("util-linux", "pkgconfig"), node("util-linux").

declared_dependency("util-linux", "python", "build").
declared_dependency("util-linux", "python", "link").
node("python") :- depends_on("util-linux", "python"), node("util-linux").
```

Some facts for the HDF5 package

--fresh only reuses builds if hashes match



- 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) is more aggressive

- --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-skyl
 -   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

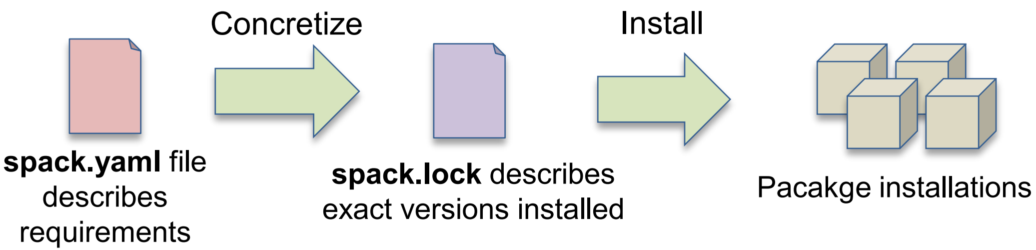
```
(spackle):spack> spack solve --reuse -Il hdf5
==> Best of 10 considered solutions.
==> Optimization Criteria:
  Priority  Criterion                                            Installed  ToBuild
  1         number of packages to build (vs. reuse)                      -        4
  2         deprecated 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=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
[+]  k2yumgx                  ^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

Use `spack spec` to see the results of concretization

```
$ spack spec mpileaks
Input spec
------------------------------
  mpileaks

Concretized
------------------------------
  mpileaks@1.0%gcc@5.3.0 arch=darwin-elcapitan-x86_64
      ^adept-utils@1.0.1%gcc@5.3.0 arch=darwin-elcapitan-x86_64
          ^boost@1.61.0%gcc@5.3.0+atomic+chrono+date_time~debug+filesystem~graph
           ~icu_support+iostreams+locale+log+math~mpi+multithreaded+program_options
           ~python+random +regex+serialization+shared+signals+singlethreaded+system
           +test+thread+timer+wave arch=darwin-elcapitan-x86_64
              ^bzip2@1.0.6%gcc@5.3.0 arch=darwin-elcapitan-x86_64
              ^zlib@1.2.8%gcc@5.3.0 arch=darwin-elcapitan-x86_64
          ^openmpi@2.0.0%gcc@5.3.0~mxm~pmi~psm~psm2~slurm~sqlite3~thread_multiple~tm~verbs+vt arch=darwin-elcapitan-x86_64
              ^hwloc@1.11.3%gcc@5.3.0 arch=darwin-elcapitan-x86_64
                  ^libpciaccess@0.13.4%gcc@5.3.0 arch=darwin-elcapitan-x86_64
                      ^libtool@2.4.6%gcc@5.3.0 arch=darwin-elcapitan-x86_64
                          ^m4@1.4.17%gcc@5.3.0+sigsegv arch=darwin-elcapitan-x86_64
                              ^libsigsegv@2.10%gcc@5.3.0 arch=darwin-elcapitan-x86_64
      ^callpath@1.0.2%gcc@5.3.0 arch=darwin-elcapitan-x86_64
          ^dyninst@9.2.0%gcc@5.3.0~stat_dysect arch=darwin-elcapitan-x86_64
              ^libdwarf@20160507%gcc@5.3.0 arch=darwin-elcapitan-x86_64
                  ^libelf@0.8.13%gcc@5.3.0 arch=darwin-elcapitan-x86_64
```

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



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

Simple spack.yaml file

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

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

Concrete spack.lock file (generated)


```json
{
 "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

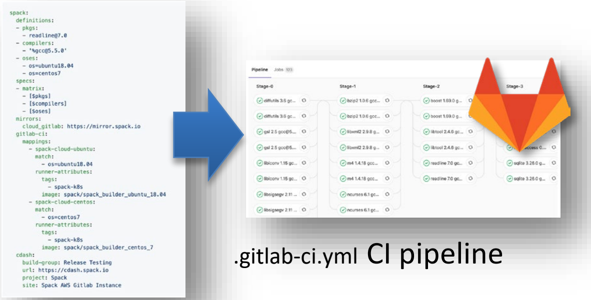
spack external find

Automatically find and configure external packages on the system

spack test

Packages know how to run their own test suites

package.py



spack.yaml

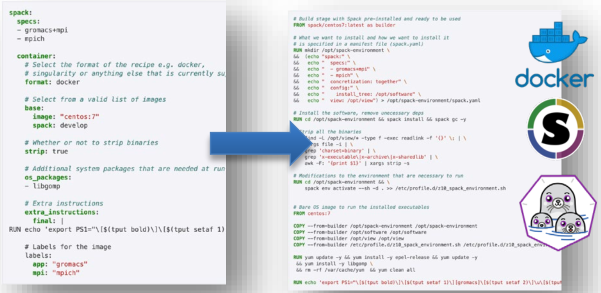
.gitlab-ci.yml CI pipeline

spack ci

Automatically generate parallel build pipelines (more on this later)

spack containerize

Turn environments into container build recipes

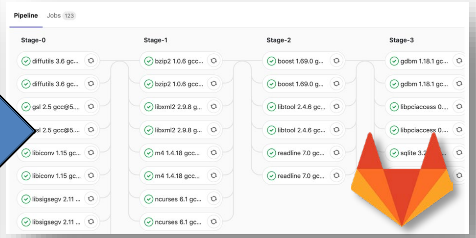


Spack environments are the foundation of Spack CI

- spack ci enables any environment to be turned into a build pipeline
- Pipeline generates a .gitlab-ci.yml file from spack.lock
- Pipelines can be used just to build, or to generate relocatable binary packages
  - Binary packages can be used to keep the same build from running twice
- Same repository used for spack.yaml can generate pipelines for project

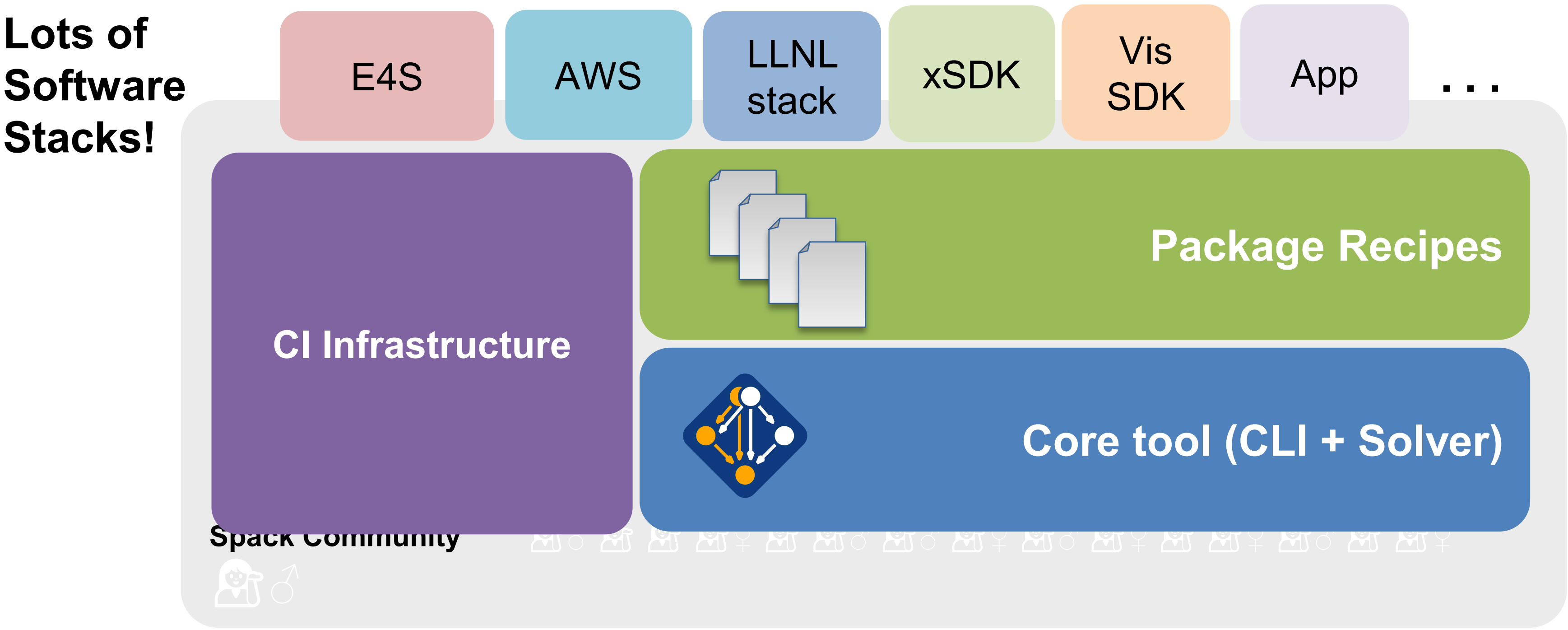
```yaml
spack:
  definitions:
  - pkgs:
    - readline@7.0
  - compilers:
    - '%gcc@5.5.0'
  - oses:
    - os=ubuntu18.04
    - os=centos7
  specs:
  - matrix:
    - [$pkgs]
    - [$compilers]
    - [$oses]
  mirrors:
    cloud_gitlab: https://mirror.spack.io
  gitlab-ci:
    mappings:
      - spack-cloud-ubuntu:
        match:
          - os=ubuntu18.04
        runner-attributes:
          tags:
            - spack-k8s
          image: spack/spack_builder_ubuntu_18.0
      - spack-cloud-centos:
        match:
          - os=centos7
        runner-attributes:
          tags:
            - spack-k8s
          image: spack/spack_builder_centos_7
  cdash:
    build-group: Release Testing
    url: https://cdash.spack.io
    project: Spack
    site: Spack AWS Gitlab Instance
```

spack.yaml

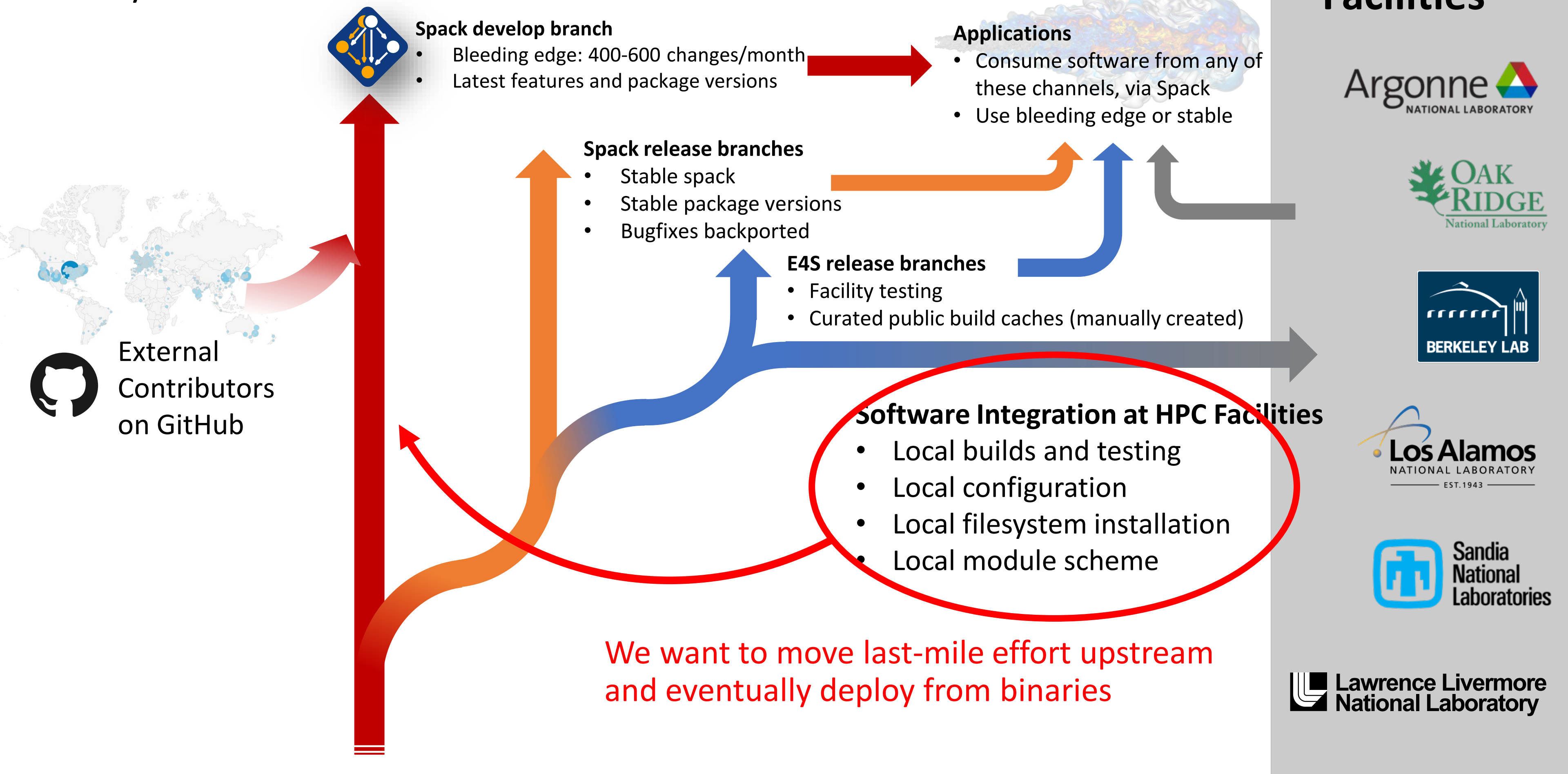


Parallel GitLab build pipeline

The Spack project enables communities to build their own software stacks



Large-scale collaboration enables us to support many downstream consumers



Spack relies on cloud CI to ensure that builds continue working

Spack Contributions on GitHub
(over 1,000 contributors)

gitlab.spack.io

spack.yaml configurations (E4S, SDKs, AWS, others)

spack ci

x86_64 and **aarch64** pipelines in **AWS**

ppc64le, GPU pipelines at UO (Frank machine)

HPE/Cray pipelines

GitLab CI builds (changed) packages
- On every pull request
- On every release branch

✓ **ci/gitlab/gitlab.spack.io** — Pipeline passed on GitLab

We have greatly simplified the process of creating a stack

- Lists of packages aimed at communities
  - E4S HPC distribution
    - Power, macOS, OneAPI versions
  - Various ML stacks
    - CPU
    - CUDA
    - ROCm
  - LLNL-specific stacks
  - AWS user stacks
- Easy to build same stack many different ways using versatile recipes
- No more boilerplate!

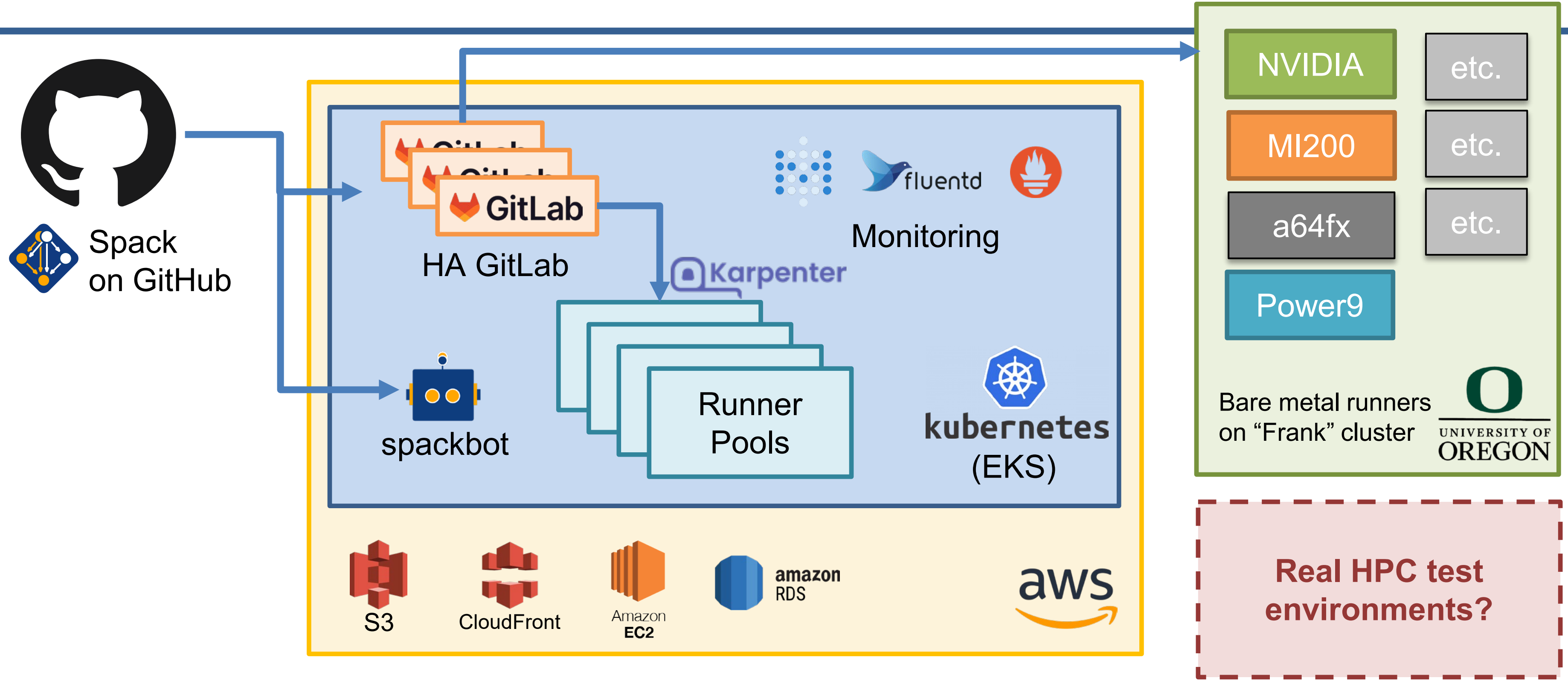
Config parameters

List of packages

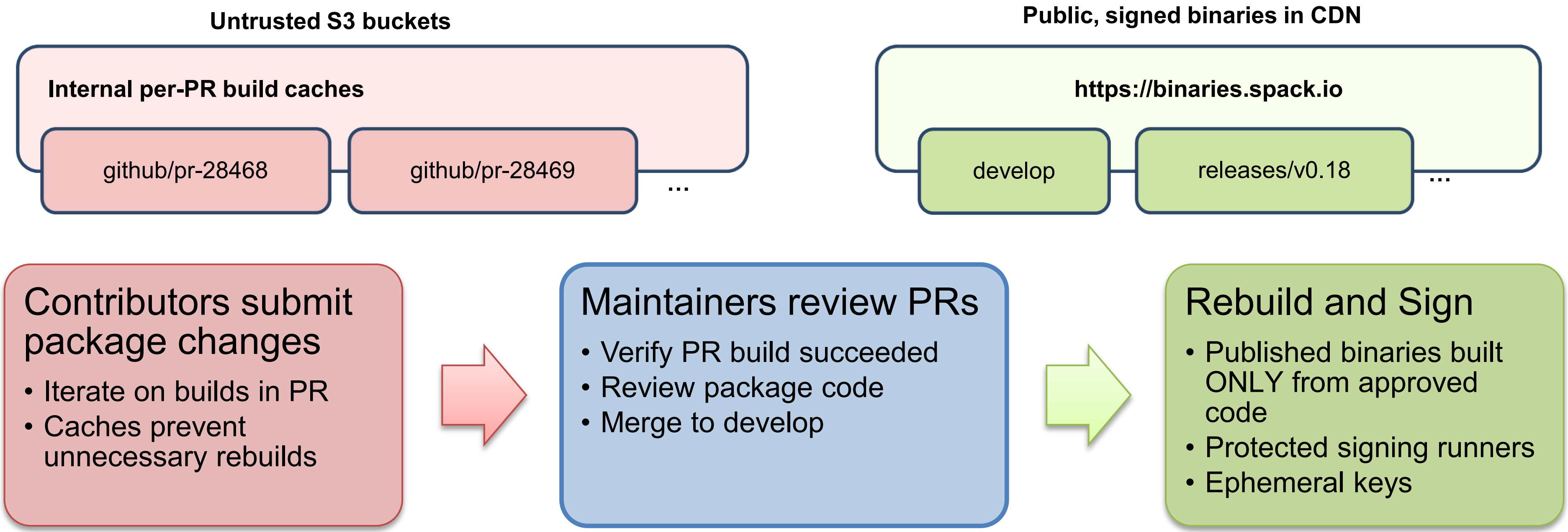
```yaml
packages:
  all:
    target: [x86_64_v3]
    variants: ~rocm+cuda cuda_arch=80
  llvm:
    # https://github.com/spack/spack/issues/27999
    require: ~cuda

definitions:
  - packages:
    # Horovod
    - py-horovod

    # Hugging Face
    - py-transformers

    # JAX
    - py-jax
    - py-jaxlib

    # Keras
    - py-keras
    - py-keras-applications
    - py-keras-preprocessing
    - py-keras2onnx

    # PyTorch
    - py-botorch
    - py-efficientnet-pytorch
    - py-gpytorch
    - py-kornia
    - py-pytorch-gradual-warmup-lr
    - py-pytorch-lightning
    - py-segmentation-models-pytorch
```

github.com/spack/spack

Spack CI Architecture



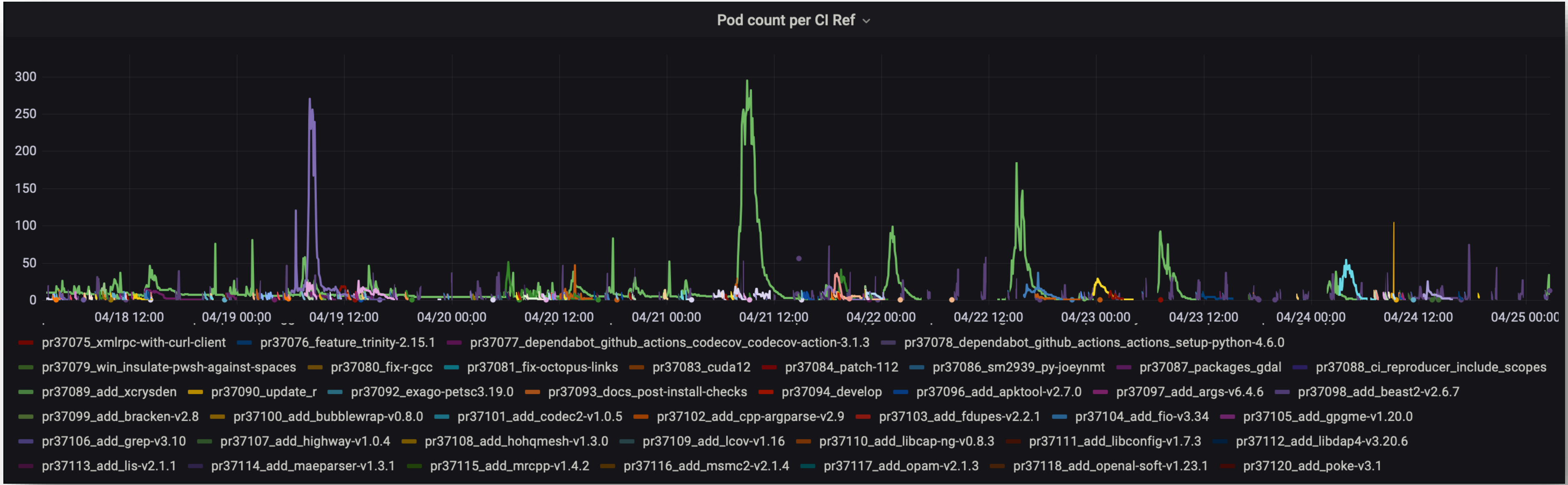
We ensure rapid turnaround *and* protect against malicious binaries by bifurcating our pipeline



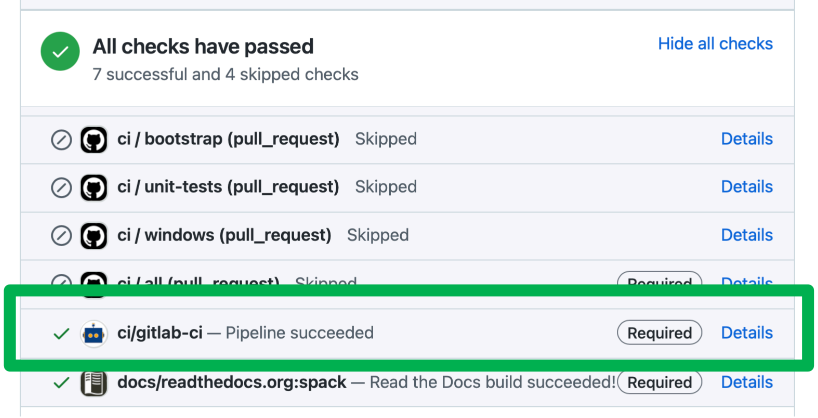
- Moves bulk of binary maintenance upstream, onto PRs
  - Production binaries never reuse binaries from untrusted environment

Our CI system enables us to build entire software stacks within a single pull request

- Users can write a simple file and fire up 300+ builders to build thousands of packages
- We're currently handling 50,000 – 100,000 package builds *per week*



We announced our public binary cache at ISC22. We're maintaining ~4,600 builds in CI!



☑ Easy (mostly) for contributors!

☑ Easy for users!

⚠ Still need HPC CI, but working on it

 # latest v0.18.x release binaries
spack mirror add v018 https://binaries.spack.io/releases/v0.18

 # rolling release: bleeding edge binaries
spack mirror add develop https://binaries.spack.io/develop

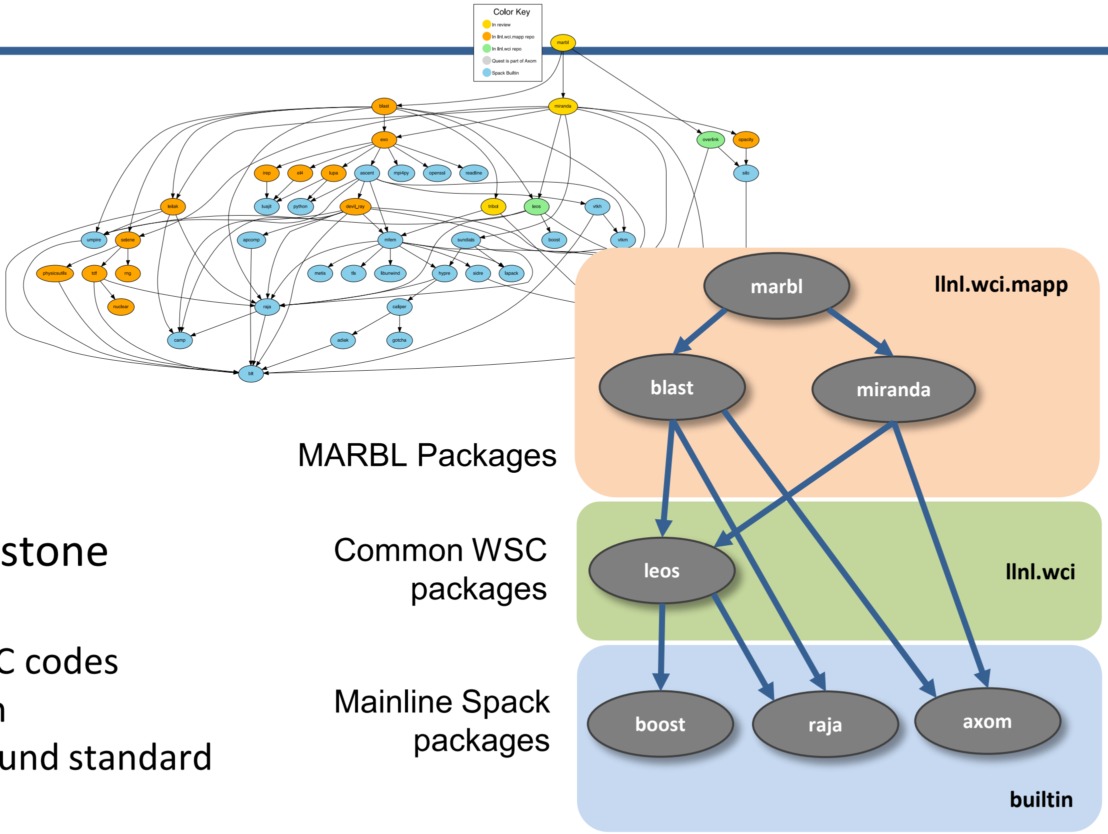
So, what else could go wrong?

We are working with code teams to develop standard workflows for layered build farms

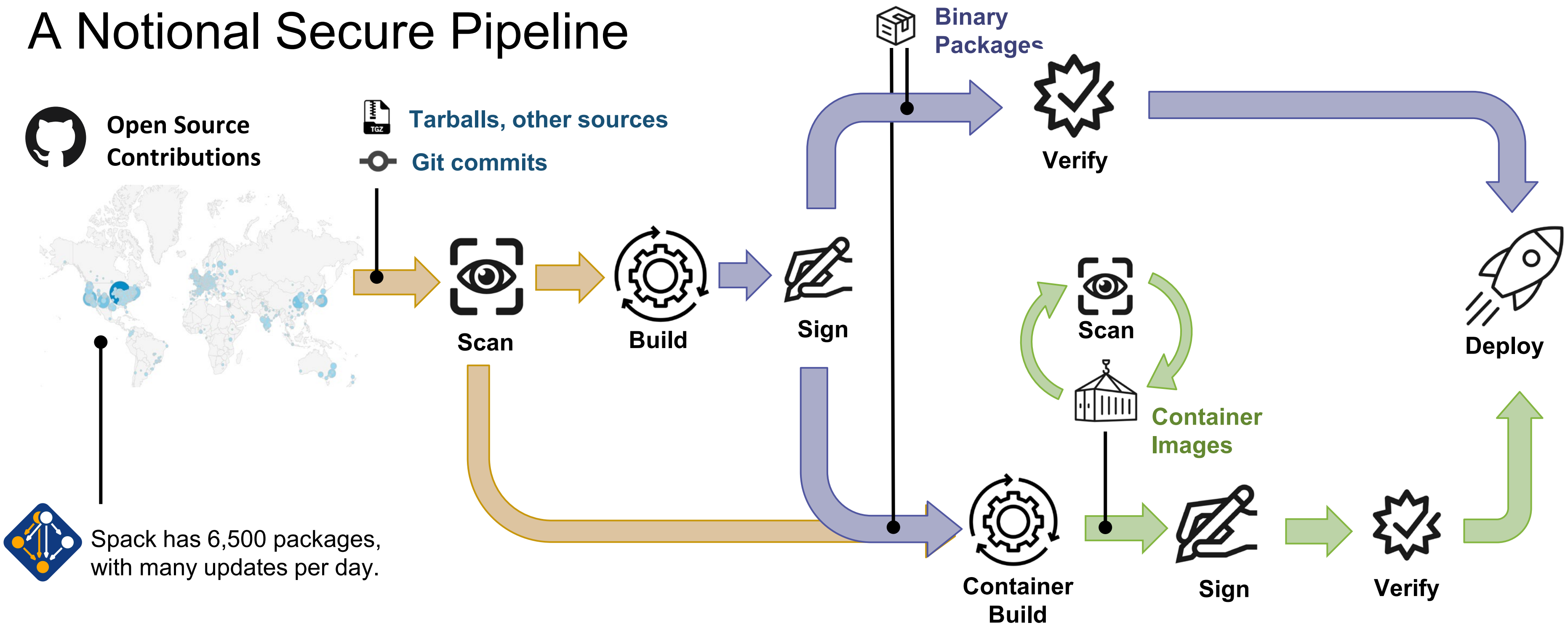
- We are working with the MARBL team to move their development environment to Spack

- We have established a build and deployment working group among WSC codes

- We aim to put together an L2 milestone for next year to:
  - Make a common build farm for WSC codes
  - Layer with Spack's public build farm
  - Gradually bring teams together around standard build configurations and workflows



A Notional Secure Pipeline



- We need a standard set of guidelines that *we accept* for supply chain integrity
  - Labs are trending towards GitLab, Spack for HPC
  - Standard container formats can help with scanning
  - Standard SBOM format could help sites cross-validate codes
- “Thorn Thymus” LDRD Strategic Initiative is working on new ways to recognize malware
  - Could integrate this into our pipeline when it’s ready

Spack retains more software provenance than most SBOMs

- Spec for zlib is at left
  - Contains much of the metadata SBOM asks for
  - Plus performance/build info of interest to HPC folks
- Patch, archive, and package recipe hashes allow you to verify the build
  - These are currently not exposed
  - We hash them and include the result
  - Can easily replace the hash with specific archive/patch hashes
- SBOM generation from this data is in progress
  - All Spack installs will have SBOMs to leverage industry tooling


```
{
 "spec": {
  "_meta": {
   "version": 3
  },
  "nodes": [
   {
    "name": "zlib",
    "version": "1.2.12",
    "arch": {
     "platform": "darwin",
     "platform_os": "bigsur",
     "target": {
      "name": "skylake"
     }
    },
    "compiler": {
     "name": "apple-clang",
     "version": "13.0.0"
    },
    "namespace": "builtin",
    "parameters": {
     "optimize": true,
     "pic": true,
     "shared": true,
     "cflags": [],
     "cppflags": [],
     "cxxflags": [],
     "fflags": [],
     "ldflags": [],
     "ldlibs": []
    },
    "hashes": {
     "archive": "91844808532e5ce316b3c010929493c0244f3d37593afd6de04f71821d5136d9",
     "patches": [
      "0d38234384870bfd34dfcb738a9083952656f0c766a0f5990b1893076b084b76"
     ],
     "package_hash": "6kkliqdv67ucuvfpfdwaacy5bz6s6en4"
    },
    "hash": "zbntgjjnd2wgvvkfi55y45ms3p7wg5ns"
   }
  ]
 }
}
```


Schema version

Package name

Version

Compiler, target architecture

Origin package repo

Variants, build options, flags

Hashes of archive, patches, build recipe

Hash of entire spec

Future directions we would like to pursue

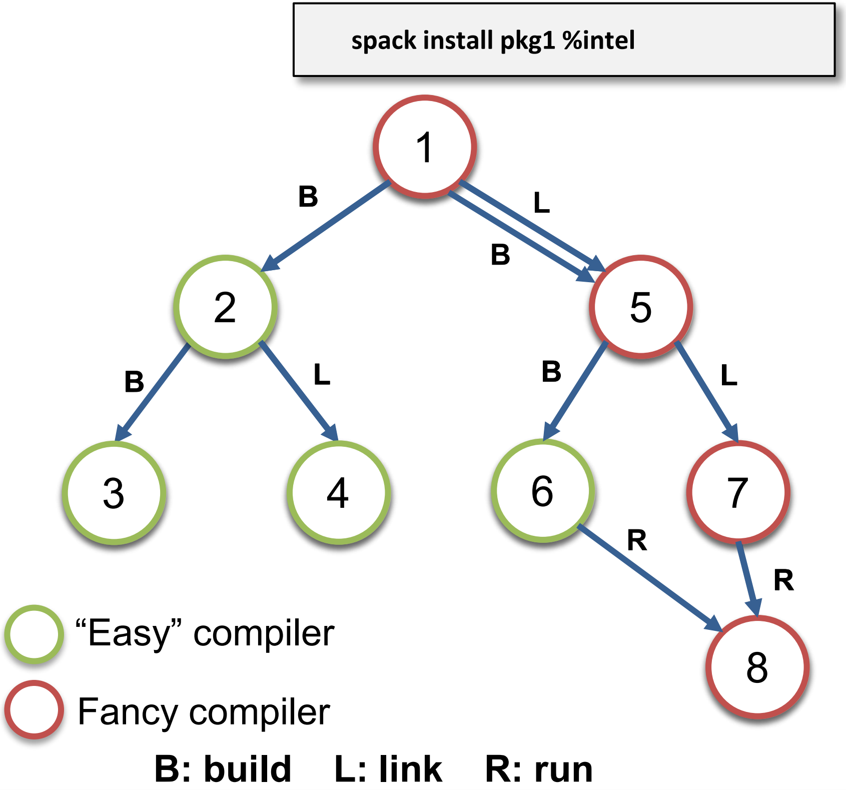
- Build pipeline hardening / scanning
  - Add scanning and assurance stages to our build pipeline
- Work with other projects to add assurance technologies
  - OpenSSF project has automated checks that can be integrated with CI pipelines
  - LLNL Thorn Thymus project has scanning
- Package curation
  - Identify and label projects within Spack that meet security standards
  - Curate a vetted sub-distribution of software
  - Work with projects like E4S
- Certified system images (for embedded devices, HPC, cloud, containers, etc.)
  - Configure and build a custom OS image with only selected components/options
  - Spack currently supports software *above* libc, but not libc
  - Contributors from the embedded community are working with us on this low-level support
    - May be used to replace tools like Yocto, OpenWRT, Gentoo

Roadmap:
Separate concretization of build dependencies

- We want to:
  - Build build dependencies with the "easy" compilers
  - Build rest of DAG (the link/run dependencies) with the fancy compiler

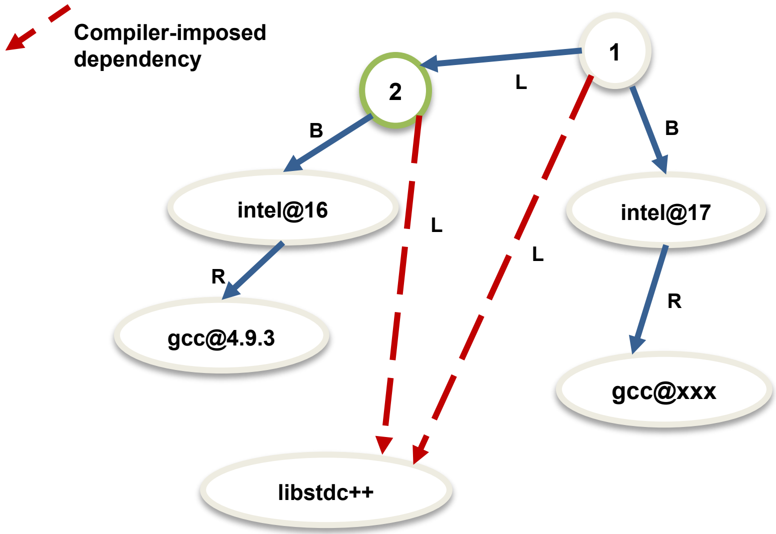
- 2 approaches to modify concretization:

1. **Separate solves**
   - Solve run and link dependencies first
   - Solve for build dependencies separately
   - May restrict possible solutions (build ←→ run env constraints)
2. **Separate models**
   - *Allow* a bigger space of packages in the solve
   - Solve *all* runtime environments together
   - May explode (even more) combinatorially



Roadmap: Compilers as dependencies

- **Need separate concretization of build dependencies to make this work**
  - Model compiler as build dep (not unified)
  - Runtimes as link deps (unified)
  - Ensure compatibility between runtimes when using multiple compilers together
- **We need deeper modeling of compilers to handle compiler interoperability**
  - libstdc++, libc++ compatibility
  - Compilers that depend on compilers
  - Linking executables with multiple compilers
- **Packages that depend on languages**
  - Depend on **cxx@2011**, **cxx@2017**, **fortran@1995**, etc
  - Depend on **openmp@4.5**, other compiler features
  - Model languages, openmp, cuda, etc. as virtuals



Compilers and runtime libs fully modeled as dependencies

When would we go to "Version 1.0"?

Big things we've wanted for 1.0 are:

- New concretizer
- production CI
- production public build cache
- Compilers as dependencies
- Stable package API
  - Enables separate package repository

Done!

We are still working on the last 3 here, but getting much closer!

Join the Spack community!

- There are lots of ways to get involved!
  - Contribute packages, documentation, or features at **github.com/spack/spack**
  - Contribute your configurations to **github.com/spack/spack-configs**

- Talk to us!
  - You're already on our **Slack channel** (spackpm.herokuapp.com)
  - Join our **Google Group** (see GitHub repo for info)
  - Submit **GitHub issues** and **pull requests**!

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

Follow us on Twitter!
@spackpm

We hope to make distributing & using HPC software easy!

Lawrence Livermore National Laboratory