Spack Status and Roadmap

EasyBuild User Meeting
January 25, 2022

Todd Gamblin

Advanced Technology Office
Lawrence Livermore National Laboratory

Spack enables Software distribution for HPC

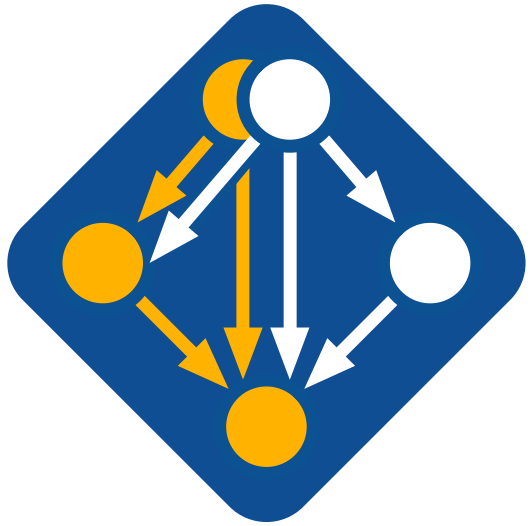
- Spack automates the build and installation of scientific software
- Packages are *parameterized,* so that users can easily tweak and tune configuration

No installation required: clone and go

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

Simple syntax enables complex installs

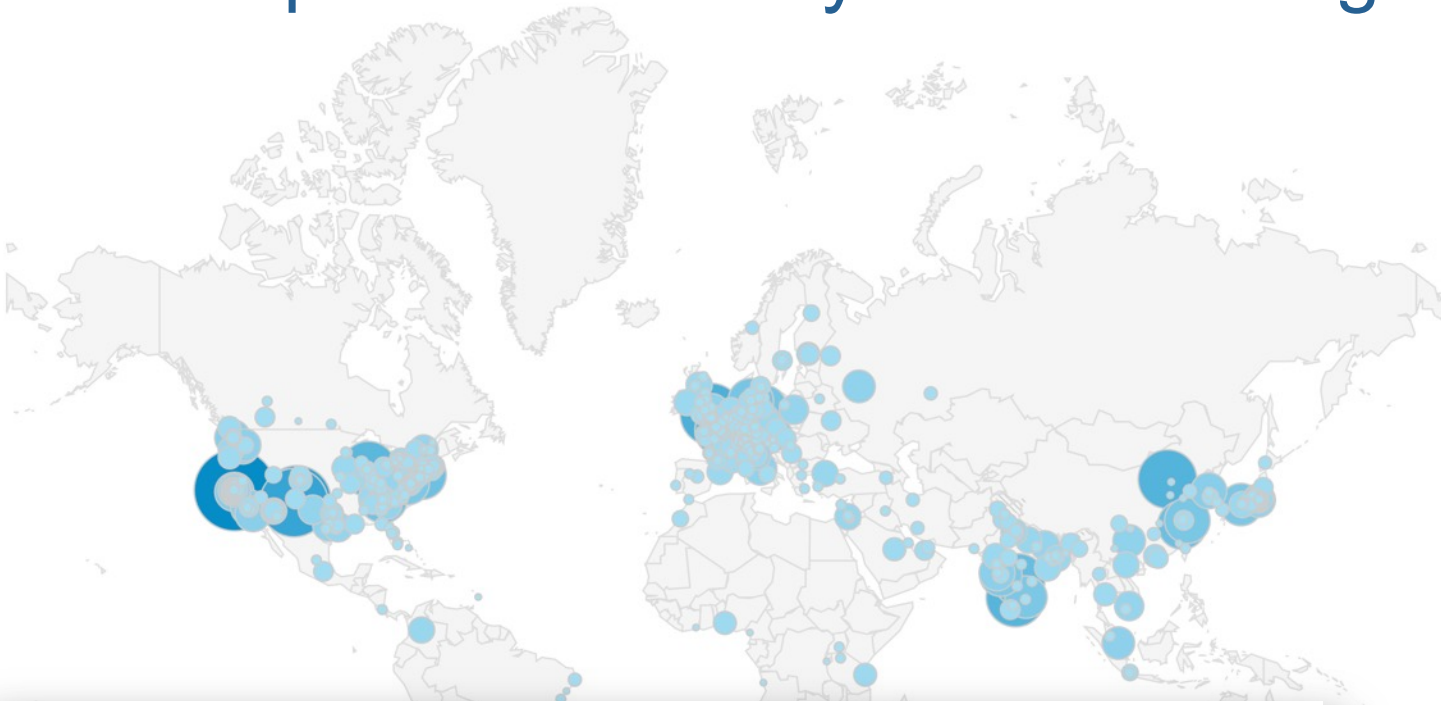
```
$ spack install hdf5@1.10.5                  $ spack install hdf5@1.10.5 cppflags="-O3 -g3"
$ spack install hdf5@1.10.5 %clang@6.0       $ spack install hdf5@1.10.5 target=haswell
$ spack install hdf5@1.10.5 +threadssafe     $ spack install hdf5@1.10.5 +mpi ^mpich@3.2
```



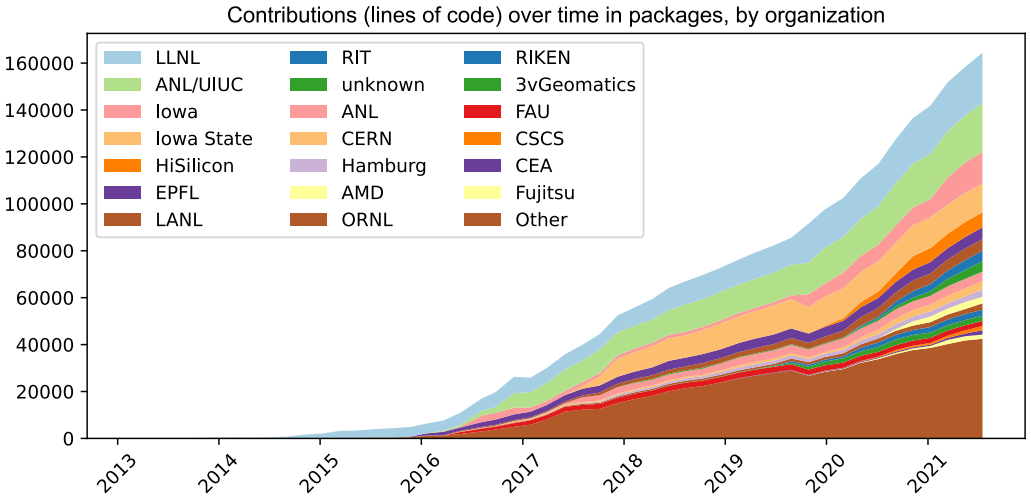
github.com/spack/spack

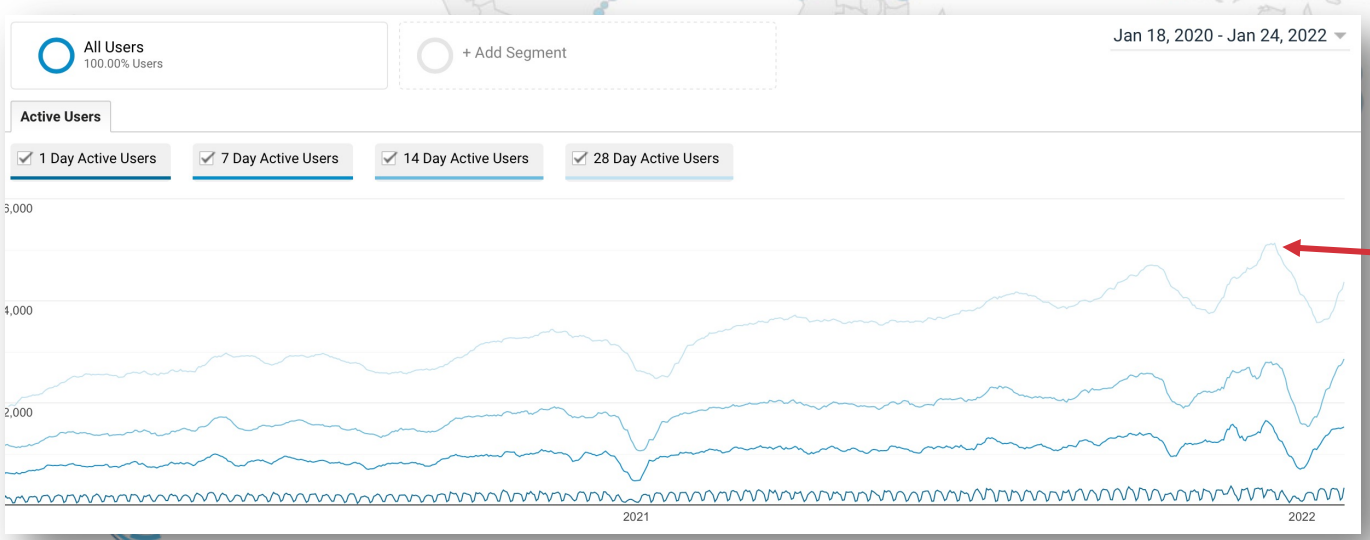
- Ease of use of mainstream tools, with flexibility needed for HPC
- In addition to CLI, Spack also:
  - Generates (but does **not** require) *modules*
  - Allows conda/virtualenv-like *environments*
  - Provides many devops features (CI, container generation, more)

The Spack community continues to grow!



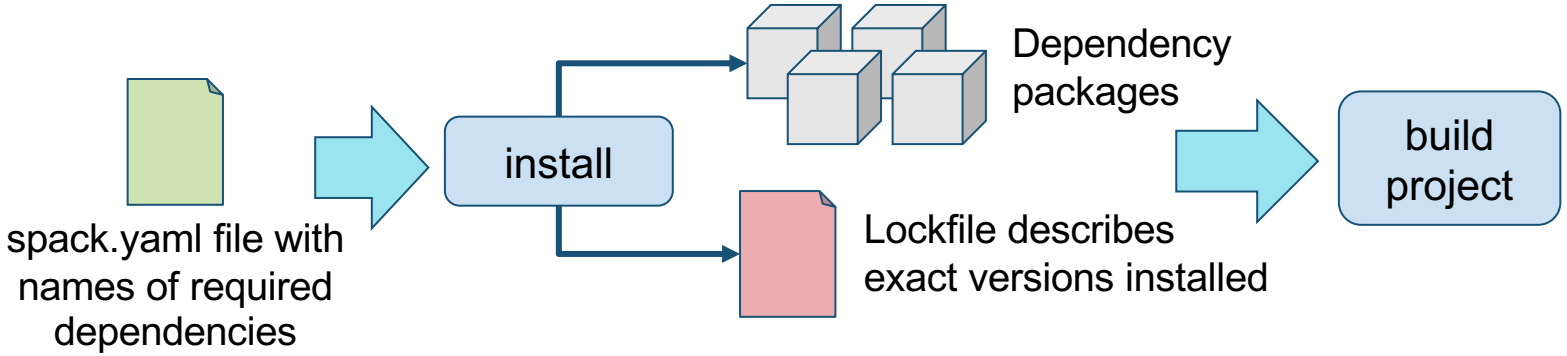
Over 6,000 software packages
900+ contributors





5,128 monthly active users on docs site in October 2022

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 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

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)

```
{
 "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": "builtin",
         "parameters": {
             "cxx": false,
             "debug": false,
             "fortran": false,
             "hl": false,
             "mpi": true,
```

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

```
class Cmake(Package):
    executables = ['cmake']

    @classmethod
    def determine_spec_details(cls, prefix, exes_in_prefix):
        exe_to_path = dict(
            (os.path.basename(p), p) for p in exes_in_prefix
        )
        if 'cmake' not in exe_to_path:
            return None

        cmake = spack.util.executable.Executable(exe_to_path['cmake'])
        output = cmake('--version', output=str)
        if output:
            match = re.search(r'cmake.*version\s+(\S+)', output)
            if match:
                version_str = match.group(1)
                return Spec('cmake@{0}'.format(version_str))
```

package.py

```
packages:
  cmake:
    externals:
      - spec: cmake@3.15.1
        prefix: /usr/local
```

spack.yaml configuration

spack external find

Automatically find and configure external packages on the system

spack test

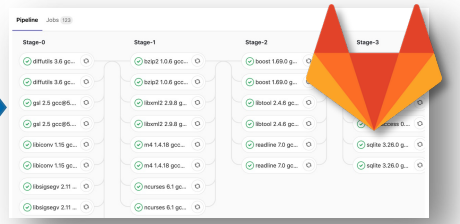
Packages know how to run their own test suites

```
class Libsigsegv(AutotoolsPackage, GNUMirrorPackage):
    """GNU libsigsegv is a library for handling page faults in user mode."""

    # ... spack package contents ...

    extra_install_tests = 'tests/.libs'

    def test(self):
        data_dir = self.test_suite.current_test_data_dir
        smoke_test_c = data_dir.join('smoke_test.c')

        self.run_test(
            'cc', [
                '-I%s' % self.prefix.include,
                '-L%s' % self.prefix.lib, '-lsigsegv',
                smoke_test_c,
                '-o', 'smoke_test'
            ]
            purpose='check linking')

        self.run_test(
            'smoke_test', [], data_dir.join('smoke_test.out'),
            purpose='run built smoke test')

        self.run_test('sigsegv1': ['Test passed'], purpose='check sigsegv1 output')
        self.run_test('sigsegv2': ['Test passed'], purpose='check sigsegv2 output')
```

package.py

```
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_18.04
      - 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



.gitlab-ci.yml CI pipeline

spack ci

Automatically generate parallel build pipelines
(more on this later)

```
spack:
  specs:
  - gromacs+mpi
  - mpich

  container:
    # Select the format of the recipe e.g. docker,
    # singularity or anything else that is currently su
    format: docker

    # Select from a valid list of images
    base:
      image: "centos:7"
      spack: develop

    # Whether or not to strip binaries
    strip: true

    # Additional system packages that are needed at run
    os_packages:
    - libgomp

    # Extra instructions
    extra_instructions:
      final: |
RUN echo 'export PS1="\[$(tput bold)\]\[$(tput setaf 1)

    # Labels for the image
    labels:
      app: "gromacs"
      mpi: "mpich"
```

```
# Build stage with Spack pre-installed and ready to be used
FROM spack/centos7:latest as builder

# What we want to install and how we want to install it
# is specified in a manifest file (spack.yaml)
RUN mkdir /opt/spack-environment \
&&  (echo "spack:" \
&&   echo "  specs:" \
&&   echo "  - gromacs+mpi" \
&&   echo "  - mpich" \
&&   echo "  concretization: together" \
&&   echo "  config:" \
&&   echo "    install_tree: /opt/software" \
&&   echo "  view: /opt/view") > /opt/spack-environment/spack.yaml

# Install the software, remove unnecessary deps
RUN cd /opt/spack-environment && spack install && spack gc -y

# Strip all the binaries
RUN find -L /opt/view/* -type f -exec readlink -f '{}' \; | \
    xargs file -i | \
    grep 'charset=binary' | \
    grep 'x-executable\|x-archive\|x-sharedlib' | \
    awk -F: '{print $1}' | xargs strip -s

# Modifications to the environment that are necessary to run
RUN cd /opt/spack-environment && \
    spack env activate --sh -d . >> /etc/profile.d/z10_spack_environment.sh

# Bare OS image to run the installed executables
FROM centos:7

COPY --from=builder /opt/spack-environment /opt/spack-environment
COPY --from=builder /opt/software /opt/software
COPY --from=builder /opt/view /opt/view
COPY --from=builder /etc/profile.d/z10_spack_environment.sh /etc/profile.d/z10_spack_en

RUN yum update -y && yum install -y epel-release && yum update -y
 && yum install -y libgomp \
 && rm -rf /var/cache/yum  && yum clean all

RUN echo 'export PS1="\[$(tput bold)\]\[$(tput setaf 1)\][gromacs]\[$(tput setaf 2)\]\n\[$(tput
```

spack containerize

Turn environments into container build recipes

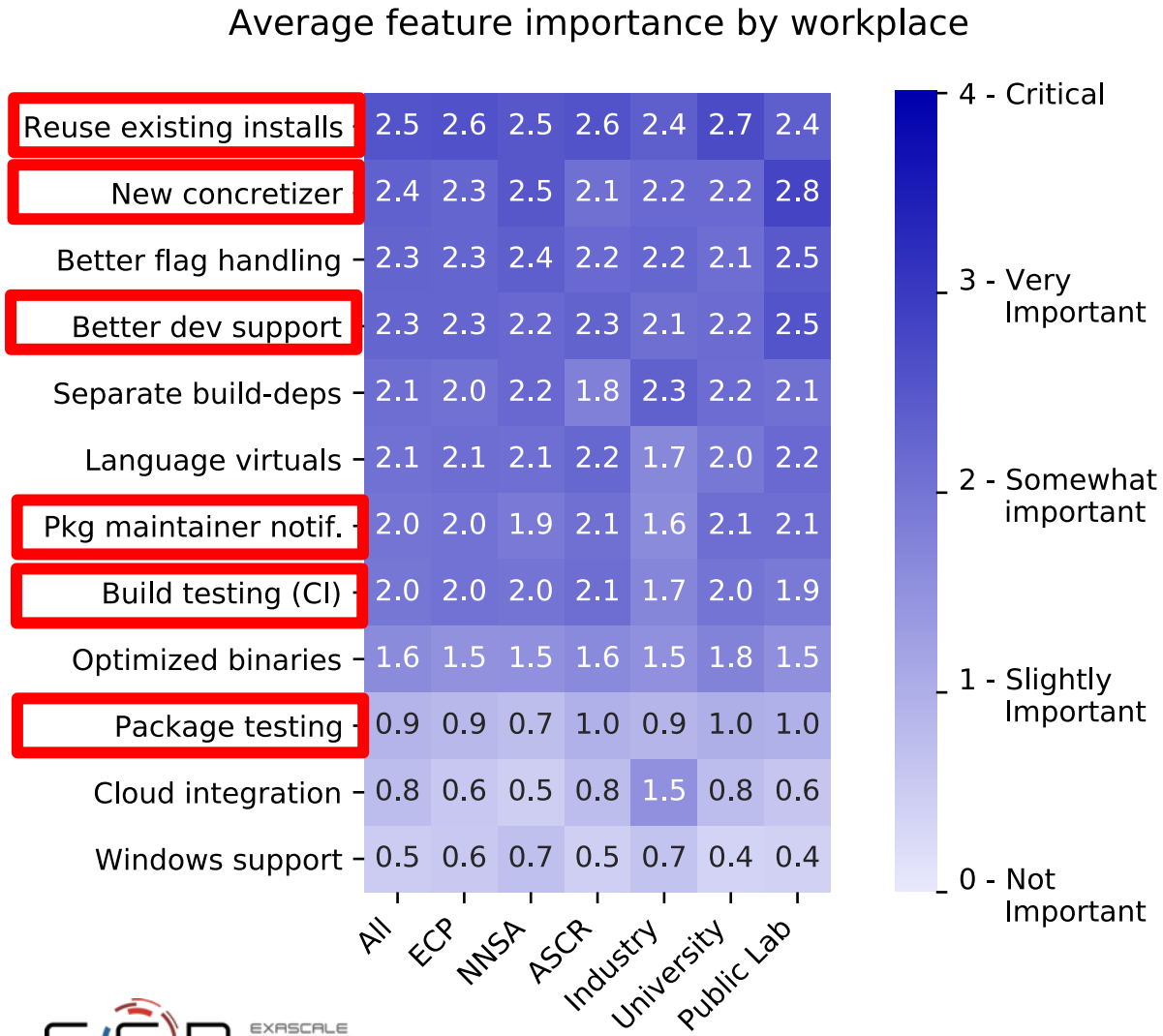
Spack v0.17.0 was just released!

Major new features:

1. New Concretizer is now default
2. Binary bootstrapping enables us to get up and running fast
3. `spack install --reuse` aggressively reuses installed packages
4. Improved error messages (for new concretizer)
5. Conditional variants for more expressive packages
6. Git commit versioning
7. Overrides for default config directories
8. Improvements to `spack containerize`
9. New commands for querying packages and tests by tag

- 5,969 packages (920 added since 0.16)
- **Full release notes:** https://github.com/spack/spack/releases/tag/v0.17.0

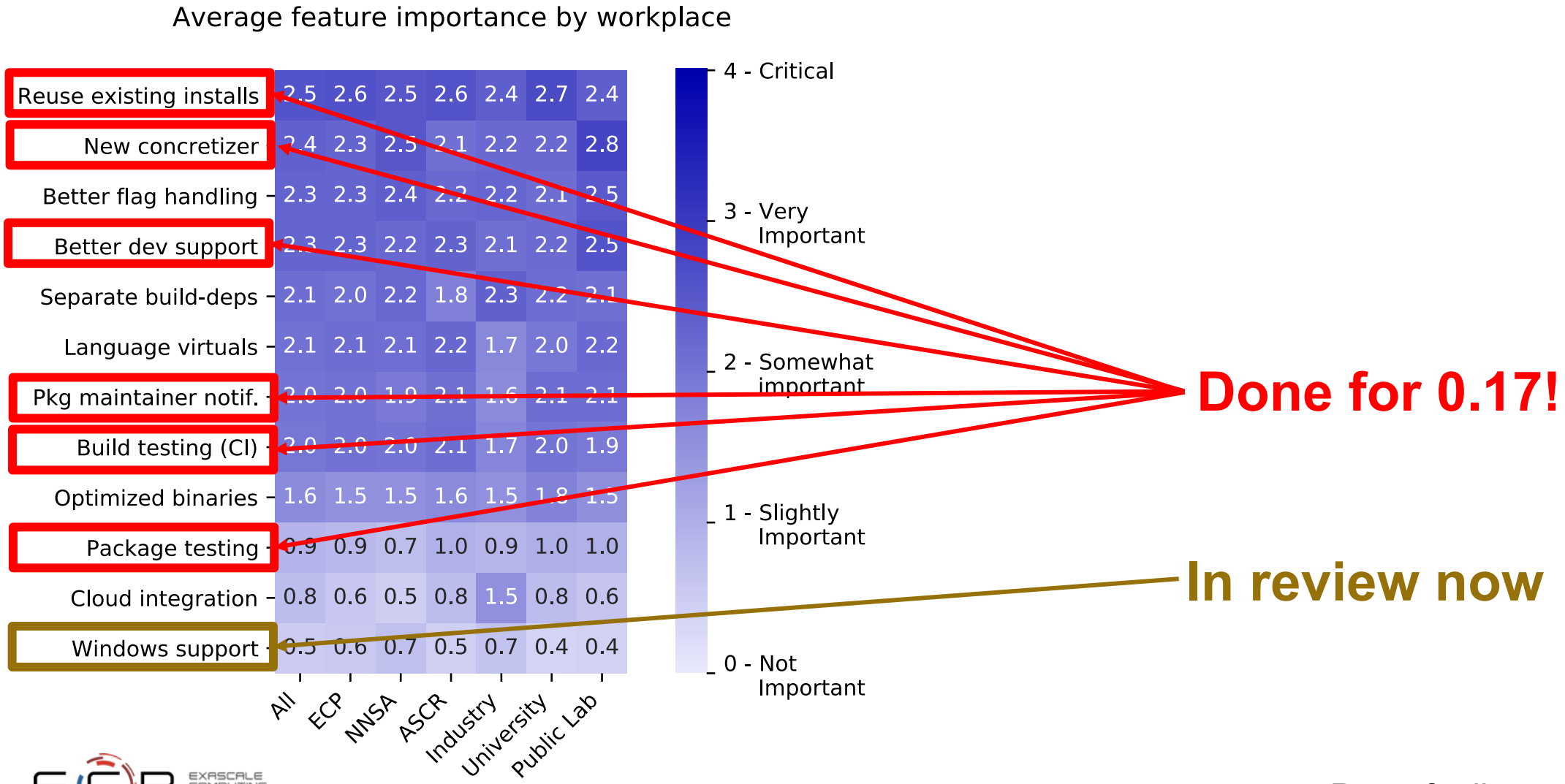
Four of the top six most wanted features in Spack were tied to the new concretizer



- Complexity of packages in Spack is increasing
  - many more package solves require backtracking than a year ago
  - Many variants, conditional dependencies, special compiler requirements
- More aggressive reuse of existing installs requires better dependency resolution
  - Need to be able to analyze how to configure the build to work with installed packages
- Separate resolution of build dependencies also requires a more sophisticated solver
  - Makes the solve even more combinatorial
  - Needed to support mixed compilers, version conflicts between different package's build requirements

Part of milestone STED09-8

Four of the top six most wanted features in Spack were tied to the new concretizer



Part of milestone STED09-8

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 envirnoments
  - 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
```

c/o Robert Blake

LLNL Applied Machine Learning team has used Spack environments to accelerate their workflow

- **LLNL Applied ML team needed to deploy**
  - PyTorch + Kull development environment
  - On ppc64le with system MPI
- **Before Spack**
  - Everybody built from scratch
  - People wrote scripts and passed them around
  - **Days were spent trying to debug build differences**
- **After spack**
  - Versioned reproducible spack environments in a git repo
  - Standard environments in a shared team directory
  - **Team members can set up a customizable environment in ~20 minutes.**
    - Change python version, PyTorch version on the fly
    - Leverage binary caches to avoid redundant builds.

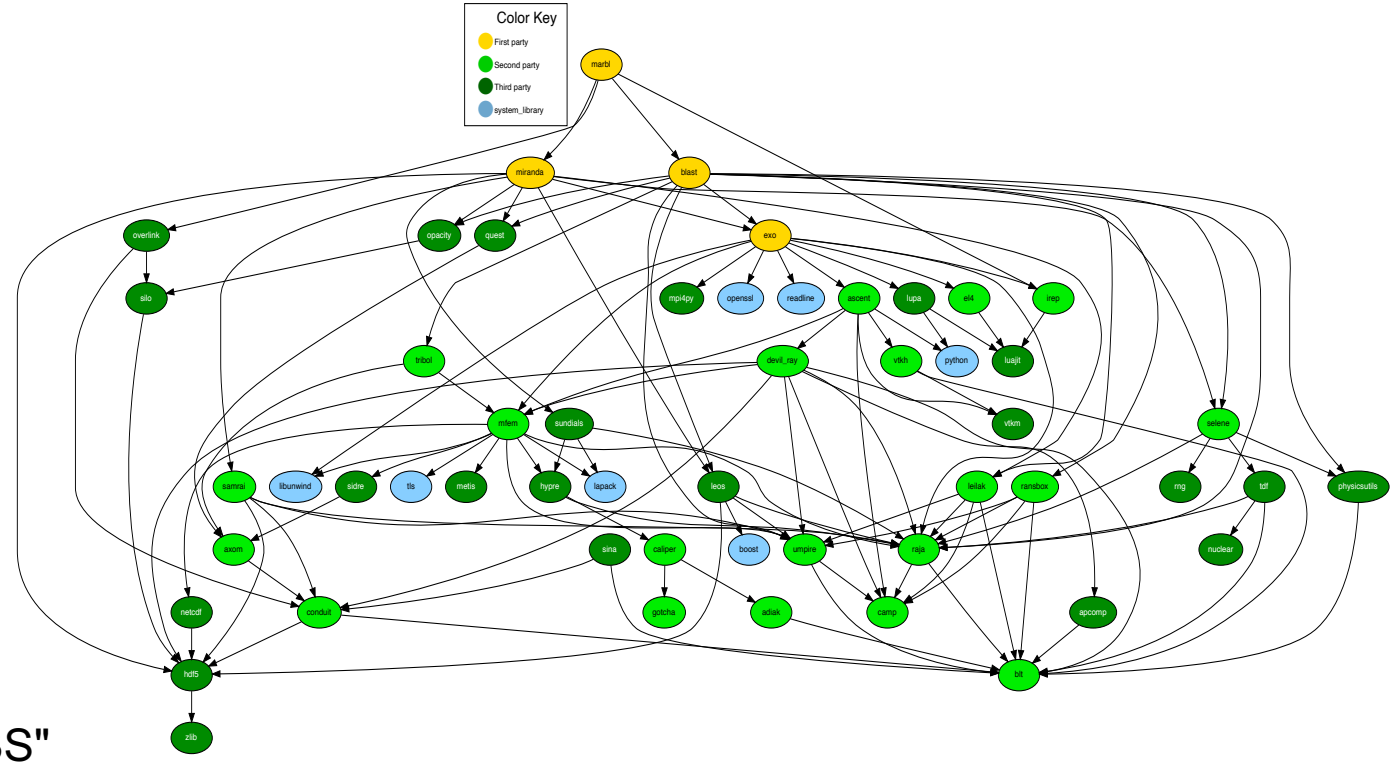
```yaml
spack:
  specs:
  - py-horovod
  - py-torch
  - python
  - py-h5py

packages:
    all:
      providers:
        mpi:
        - mvapich2@2.3
        lapack:
        - openblas threads=openmp
        blas:
        - openblas threads=openmp
      buildable: true
      variants: [+cuda cuda_arch=37]
      compiler: [gcc@7.3.0]
    ...
    python:
      version: [3.8.6]
    cudnn:
      version:
      - 8.0.4.30-11.1-linux-x64
    py-torch:
      buildable: true
      variants: +cuda +distributed
    mvapich2:
      externals:
      - spec: mvapich2@2.3.1%gcc@7.3.0
        prefix: /usr/tce/packages/mvapich2/mvapich2-2.3-gcc-7.3.0
    compilers:
      - compiler:
          operating_system: rhel7
          paths:
            cc: /usr/tce/packages/gcc/gcc-7.3.0/bin/gcc
            cxx: /usr/tce/packages/gcc/gcc-7.3.0/bin/g++
```

`spack.yaml` file

We wanted to translate this workflow to larger codes.

We have recently introduced some new features to support the development model of MARBL, an LLNL multi-physics code

- Not unlike other LLNL codes, but…
- MARBL is more deeply modular than prior codes
  - Designed to support modular *physics*
  - MARBL itself has two hydro options: Miranda & Blast
  - Code, build structure both assume that a simulation is comprised of *packages*
- Needed a way to simplify modular workflows
  - Need to work on several repos at once
  - Changes to the code are multiple pull requests
- LLNL doesn't (likely won't) use mono-repos
  - Managing permissions is hard
  - Code timescales vary
  - Independence of teams is important
- Wanted replacement for existing makefile-based "MBS" build system
  - Probably the coolest makefile-based build system we've seen



Using git versioning, we've been able to support MARBL's developer workflow

- First section is familiar
  - List of packages with hashes
- `spack.yaml` ties the modular MARBL code together:
  - hashes
  - parts of exo/build directory
- Some differences:
  - Packages in Spack are configurable
  - Can set per-package options
  - Compiler options, flags are configurable in Spack environments
- If this is too long, some of this can be moved to external includes

```
spack:
  specs:
  - marbl                  @develop build_type=Release
  - miranda                @develop
  - blast                  @wktexports
  - exo                    @wktexports
  - adiak                  @950e3bfb91519ecb7b7ee7fa3063bfab23c0e2c9
  - ascent ~fortran~openmp @587f6cf9503ef6176e59a046f6331baed5e36ce6
  - axom ~lua~openmp       @587f6cf9503ef6176e59a046f6331baed5e36ce6
  - blt                    @43022da4dfed5a50a02fbd0355defd03f12157cd
  - caliper~libdw          @85601f48e7f883fb87dec85e92c849eec2bb61f7
  - camp                   @85601f48e7f883fb87dec85e92c849eec2bb61f7
  - care                   @7f43ed9ed8400f6173b8434b6471142a8ffd4882
  - chai                   @d3282bc95c533efb90ec0a06085e455daa97df6b
  - conduit                @f54f834eb8aaff4fc97613e04cfdb360997867be
  - dray ~test~utils~openmp @c0bee76f2dce29139bde1084bf085d7d1c1b01b4
  - el4                    @aded490988f1d0a11ff74f9be7135d95e25e90ca
  - glvis                  @20aeb2c03ce70f445232dba74179e03c94de0c2c
  - gotcha                 @e0455990e57e5b74e16343816cd0d2d4f38d65de
  - irep                   @5d4d2893b25c4dfe4ad05dd6d8110179980c2a6b
  - leilak                 @1886056c398a6919bf8cce4216732fc1d8643954
  - mfem +shared           @9d8043b9e78dcdcd86639bbb28d3bd7b514fb5e2
  - raja ~openmp           @9cb6370bb2868e35ebba23cdce927f5f7f9da530
  - ransbox                @edf072bfa7b3f6e0fd6eb106abbe65ae5f677abe
  - samrai                 @39017121bda44fff713fe3b01cb1e063be93023b
  - selene                 @6f9b15713c738d70b125bc08aef72925d961a02e
  - spheral                @8cc54824c2937405203c3803ab44960fc26d506d
  - tribol                 @b9185d317bf14d87462ca345086931580c591eb4
  - umpire ~openmp         @5201a47a35e3844160dcbecd0916f8c96aa7dd07
  - vtkh                   @cd6004c94b083b096fda5f994b491b8229dacd79
  - hdf5                   @:1.8 +cxx+fortran~mpi
  - netcdf-c ~mpi
  - python                 @3.7.2
  - boost                  @1.76.0
  - leos                   @8.3.4:1
  view: false
  concretization: together

  repos::
    - ~/src/llnl.wci.mapp
    - $spack/var/spack/repos/builtin
    - ~/src/llnl.wci

  compilers:
  - compiler:
      spec: intel@18.0.2
      paths:
        cc: /usr/tce/bin/icc-18.0.2
        cxx: /usr/tce/bin/icpc-18.0.2
        f77: /usr/tce/bin/ifort-18.0.2
        fc: /usr/tce/bin/ifort-18.0.2
      flags: {}
      operating_system: rhel7
      target: x86_64
      modules: [gcc/4.9.3, intel/18.0.2]
```

options, versions/hashes

package repos

compiler info

```
packages:
  all:
    compiler: [intel@18.0.2]
    providers:
      mpi: [mvapich2]
      blas: [netlib-lapack]
      lapack: [netlib-lapack]
  hypre:
    variants: +shared
  mpi:
    buildable: false
    externals:
    - spec: mvapich2@2.3%intel@18.0.2 process_managers=slurm arch=linux-rhel7-ivybridge
      prefix: /usr/tce/packages/mvapich2/mvapich2-2.3-intel-18.0.2
  blas:
    buildable: false
  lapack:
    buildable: false
  netlib-lapack:
    buildable: false
    externals:
    - spec: netlib-lapack@3.6.1+shared
      prefix: /usr
  cuda:
    buildable: false
    externals:
    - spec: cuda@10.2
      prefix: /opt/cudatoolkit/10.2
  # Basic build deps
  autoconf:
    buildable: false
    externals:
    - spec: autoconf@2.69
      prefix: /usr
  automake:
    buildable: false
    externals:
    - spec: automake@1.13.4
      prefix: /usr
  bzip2:
    buildable: false
    externals:
    - spec: bzip2@1.0.6
      prefix: /usr
  cmake:
    version: [3.14.5]
    buildable: false
    externals:
    - spec: cmake@3.14.5
      prefix: /usr/tce/packages/cmake/cmake-3.14.5
  gettext:
    buildable: false
    externals:
    - spec: gettext@0.19.8.1
      prefix: /usr
  libtool:
    buildable: false
    externals:
    - spec: libtool@2.4.2
      prefix: /usr
  m4:
    buildable: false
    externals:
    - spec: m4@1.4.16
      prefix: /usr
  perl:
    buildable: false
    externals:
    - spec: perl@5.16.3
      prefix: /usr
  pkg-config:
    buildable: false
    externals:
    - spec: pkg-config@0.27.1
      prefix: /usr
  tar:
    buildable: false
    externals:
    - spec: tar@1.26
      prefix: /usr
```

external package prefs

MPI

BLAS/LAPACK

build dependencies

Current MARBL spack.yaml

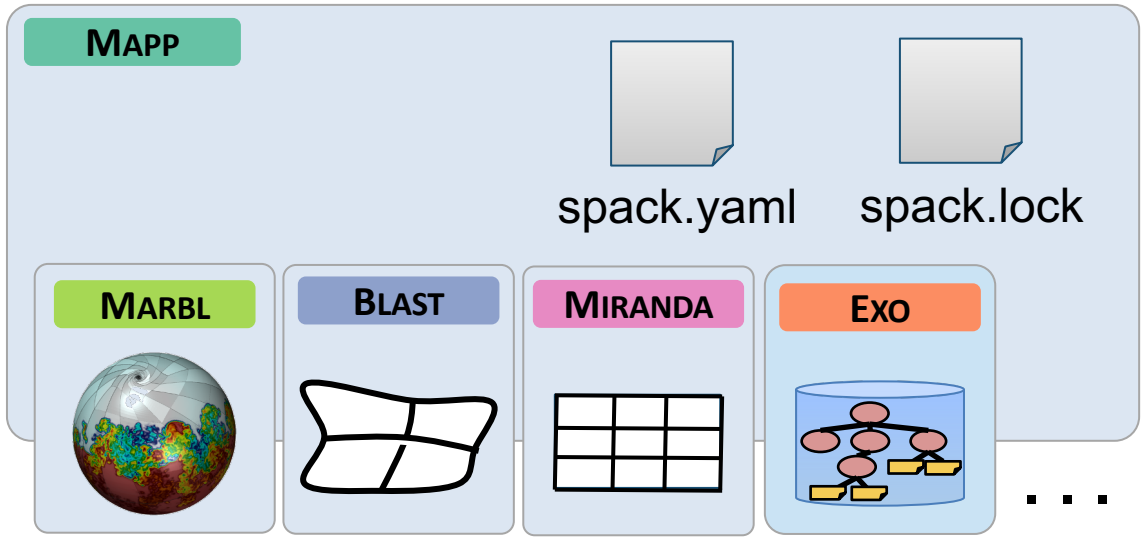
Spack workflow for developer environment

Spack

```
$ git clone ssh://git@rzgitlab.llnl.gov:7999/mapp/mapp
$ cd mapp
$ spack env activate .
$ spack develop marbl@develop
$ spack develop blast@develop
$ spack develop miranda@develop
$ spack develop exo@develop
$ srun –N 2 –n 16 --exclusive spack install
```

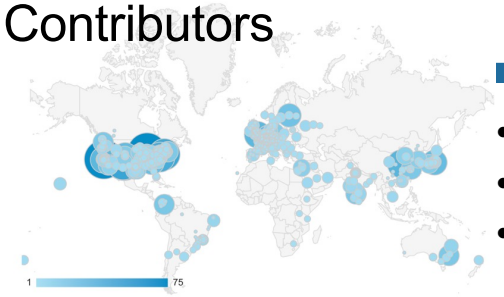
We can find ways to shorten this

spack can do multi-node builds



Concretization is at the core of Spack!

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

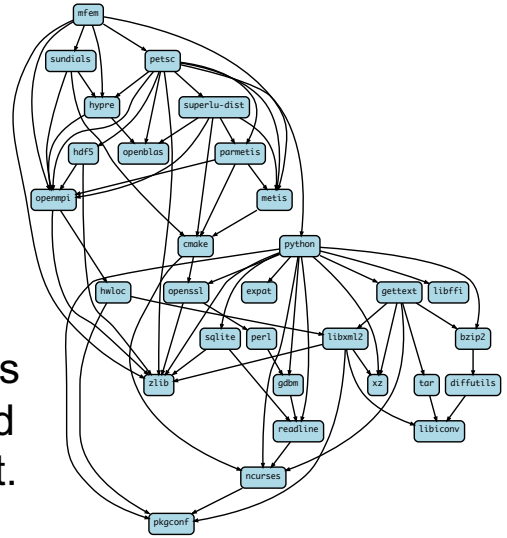
yaml environment config `spack.yaml`

users

Command line constraints

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

Concrete spec is fully constrained and can be built.



This problem is NP-hard!

In 0.17, the new concretizer is default!

- Used Clingo, the Potassco grounder/solver package
- ASP program has 2 parts:
  1. Large list of facts generated from package recipes & installed packages
     - 15,000 facts is typical – includes dependencies, options, etc.
  2. Small logic program (~500 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
- Binary bootstrapping allows us to get clingo up and running quickly
  - First C++ dependency in Spack

```
%-----------------------------------------------------------------------------
% 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 HDF5 package

Crash course in ASP

- ASP syntax is derived from **Prolog**
- Basic piece of a program is a *term*
- Terms can easily represent any data structure, e.g. this is a graph with:
  - 2 nodes, one with a variant value
  - 1 dependency edge
- Terms followed by '.' are called *facts*
  - Facts say "this is true!"

```
enable_some_feature.

node("lammps").

node("cuda").

variant_value("lammps", "cuda", "False").

depends_on("lammps", "cuda", "link").
```

Crash course in ASP

- ASP programs also have ***rules***.
  - Rules can derive additional facts.
- **:-** can be read as **"if"**
  - The **head** (left side) is true
  - **If** the **body** (right side) is true
- **Comma** in the body is like "**and**"
  - Writing same head twice is like "or"
- Capital words are **variables**
  - Rules are instantiated with all possible substitutions for variables.

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

```
node("cuda")
```

```
node("lammps").
depends_on("lammps", "cuda", "link").
```

Crash course in ASP

- ***Constraints*** say what ***cannot*** happen

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

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

- ***Choice rules*** give the solver freedom to choose from possible options:

```
% if a package is in the graph, solver must choose exactly one version
% out of that package's possible versions
1 { version(V) : possible_version(Package, V) } 1 :- node(Package).
```

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)
  - ASP is guaranteed to complete!

Spack DSL allows *declarative* specification of complex constraints

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

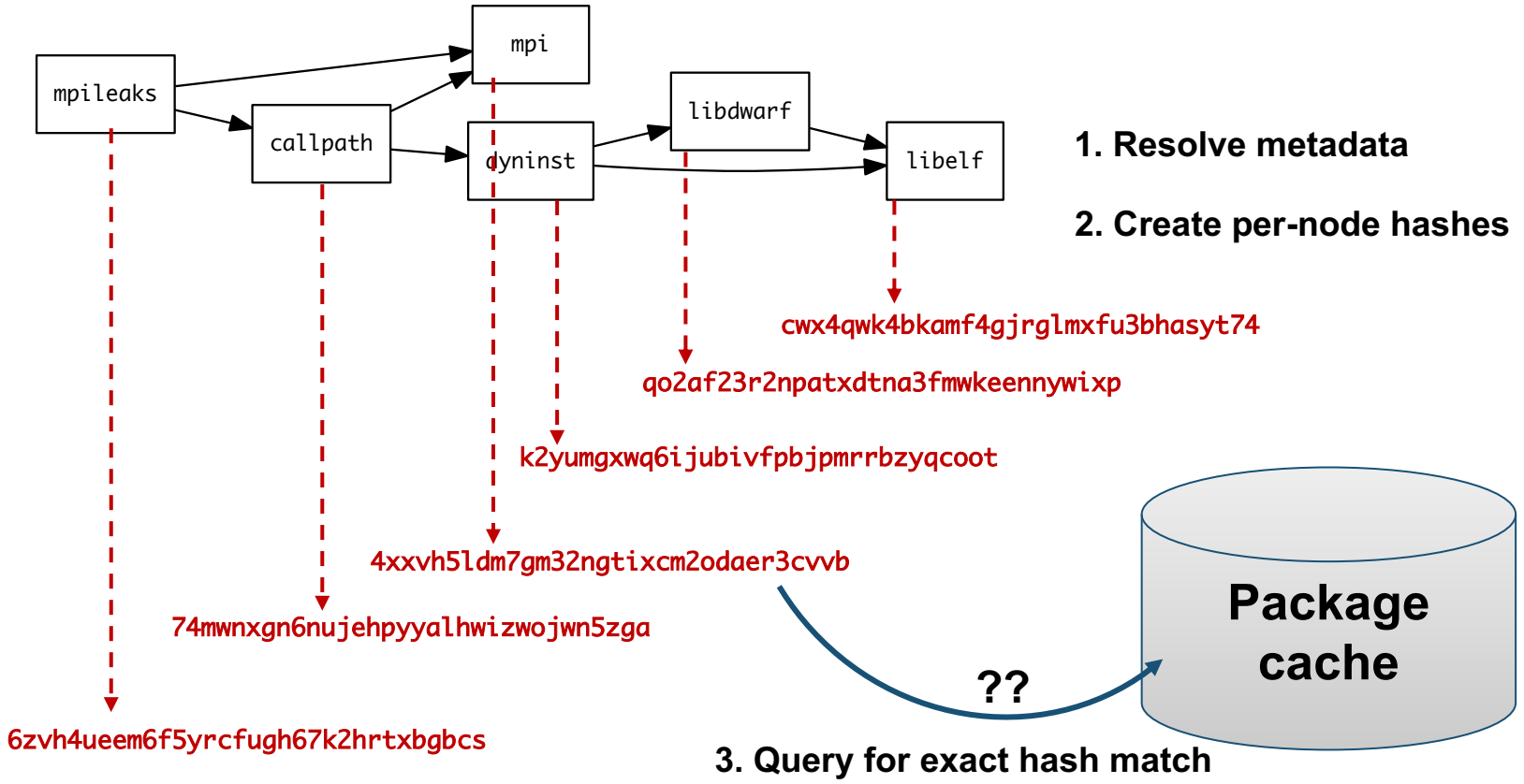
dependency on cuda, but only if cuda is enabled

constraints on cuda version

compiler support for x86_64 and ppc64le

There is a lot of expressivity in this DSL.

Many packaging systems reuse builds via metadata hashes



- 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

We can be more aggressive about reusing packages.

- First, we need to tell 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: it's just the *impose* half of a generalized condition.

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

Note the bifurcated optimization criteria

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

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

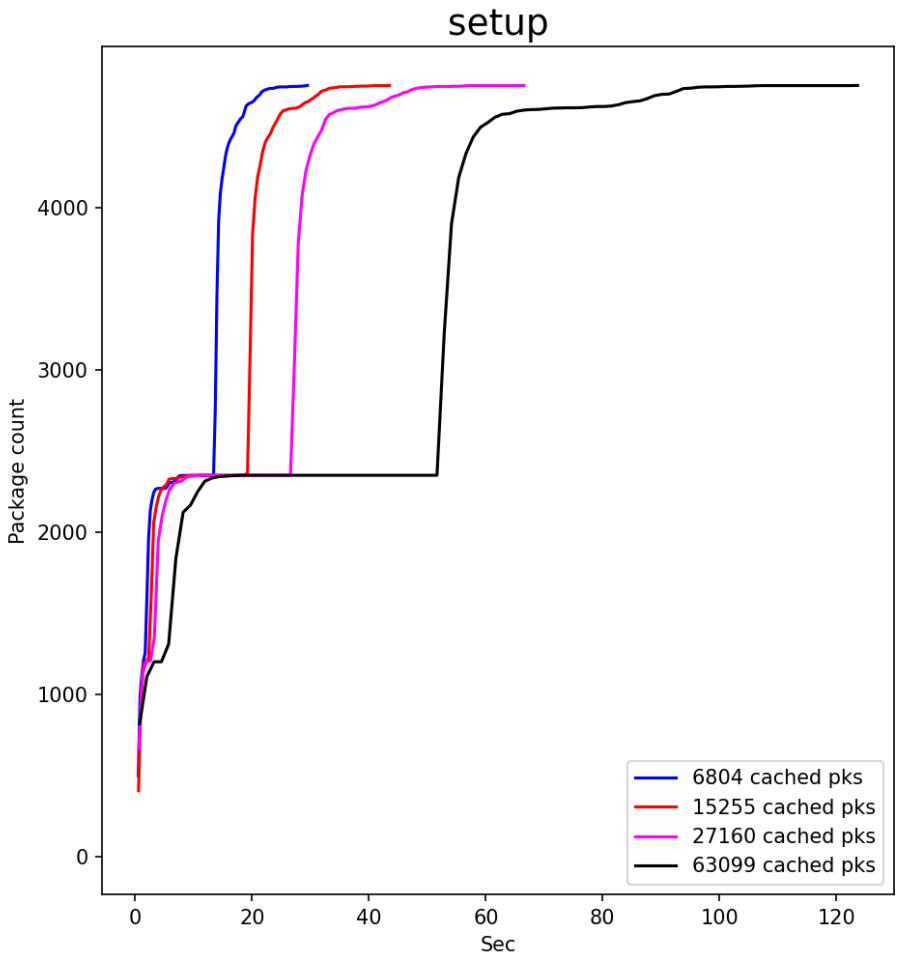
Pure hash-based reuse: all misses

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

  -   yfkfnsp  hdf5@1.10.7%apple-clang@12.0.5~cxx~fortran~hl~ipo~java+mpi+shared~szip~threadsafe+tools api=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
```

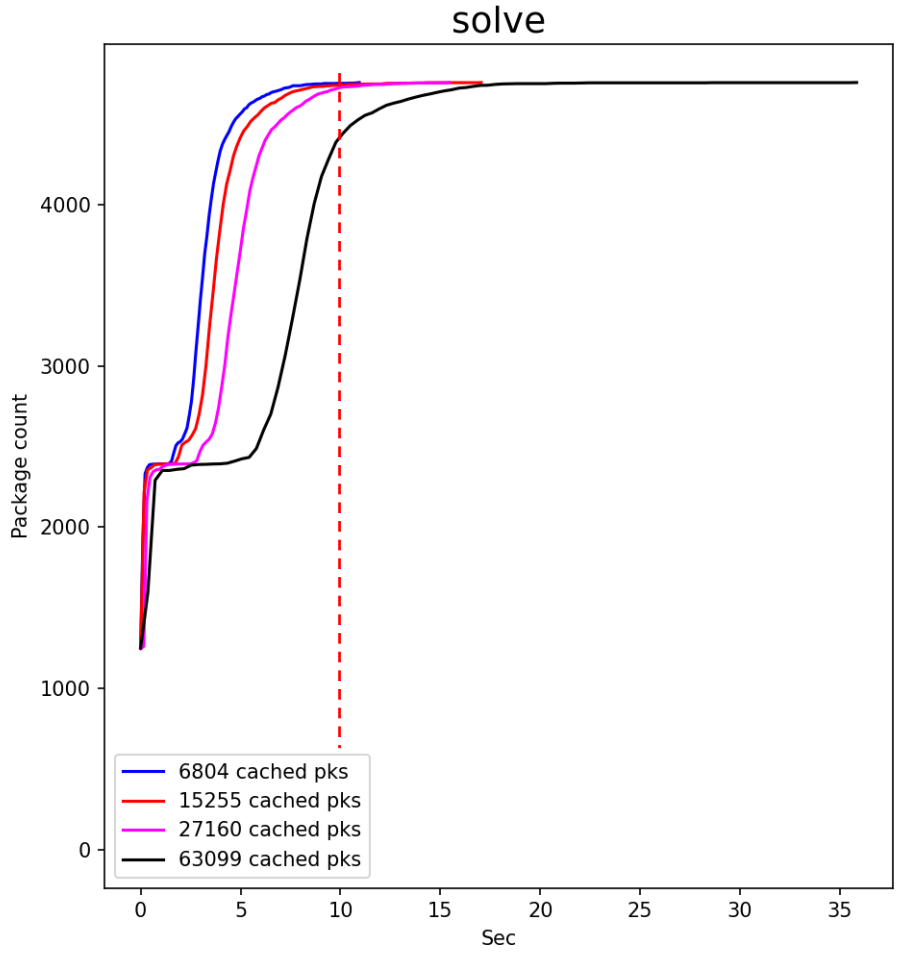
With --reuse: 16 packages were actually acceptable for this build

So far, it looks like we can handle very large problem sizes with the reusing solver

- Cumulative distribution of setup and solve times
- Hypothesis: we don't see big combinatorial blow-up b/c we're strict about dependency hashes
- Next: try mixed ABI, but *prefer* "pure" source-built dependencies



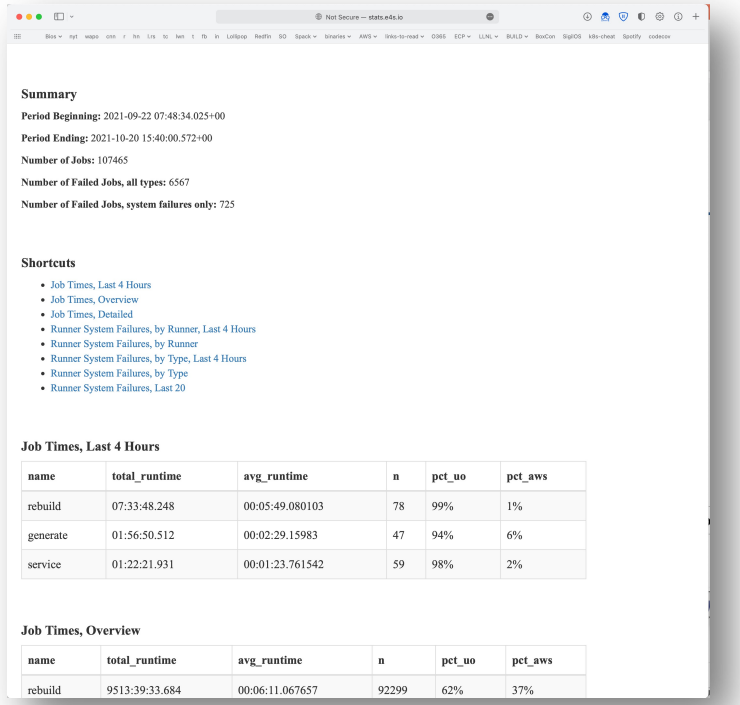
Most of the time is spent in setup (reading data in Python – can be sped up w/caching)



Even with 63k packages in a repo, nearly all package solves take < 10 sec

Future CI directions focus on scalability and testing

- Scaling tests up to handle every PR has been very difficult
  - Driven by GitLab
  - Using Kubernetes builders
  - Using a cluster at U. Oregon
- Concretization of large environments was slowing turnaround
  - 55 min to concretize E4S environment (each spec separately)
  - Brought this down to 2.5 min with parallelization and caching
- Amazon and E4S/UO team helping to pinpoint errors
- We are now doing about 100,000 builds/month
- Once we have a stable, rolling release of spack develop branch, we'll make the build cache public
  - Rolling binaries for develop
  - Long-lived snapshots for each release



Summary

Period Beginning: 2021-09-22 07:48:34.025+00

Period Ending: 2021-10-20 15:40:00.572+00

Number of Jobs: 107465

Number of Failed Jobs, all types: 6567

Number of Failed Jobs, system failures only: 725

Shortcuts

- Job Times, Last 4 Hours
- Job Times, Overview
- Job Times, Detailed
- Runner System Failures, by Runner, Last 4 Hours
- Runner System Failures, by Runner
- Runner System Failures, by Type, Last 4 Hours
- Runner System Failures, by Type
- Runner System Failures, Last 20

Job Times, Last 4 Hours

name	total_runtime	avg_runtime	n	pct_uo	pct_aws
rebuild	07:33:48.248	00:05:49.080103	78	99%	1%
generate	01:56:50.512	00:02:29.15983	47	94%	6%
service	01:22:21.931	00:01:23.761542	59	98%	2%

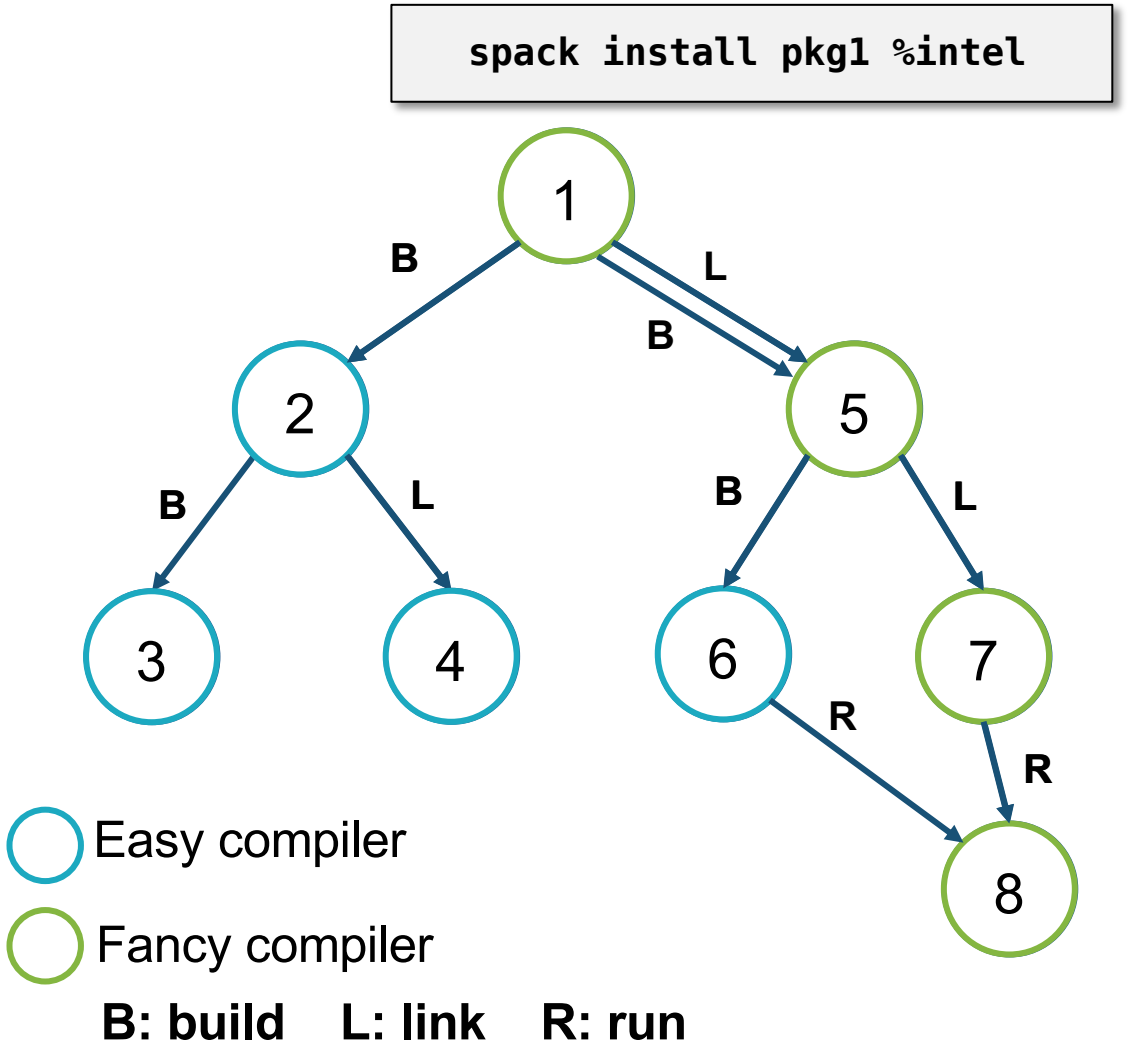
Job Times, Overview

name	total_runtime	avg_runtime	n	pct_uo	pct_aws
rebuild	9513:39:33.684	00:06:11.067657	92299	62%	37%

http://stats.e4s.io

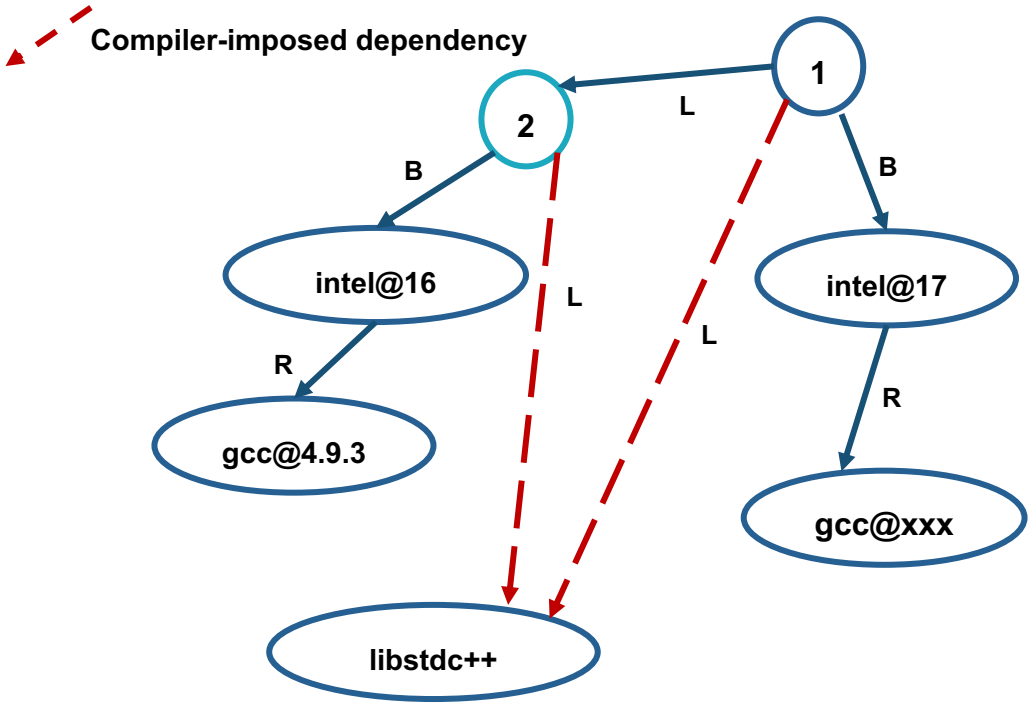
Spack v0.18 roadmap:
Separate concretization of build dependencies

- 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



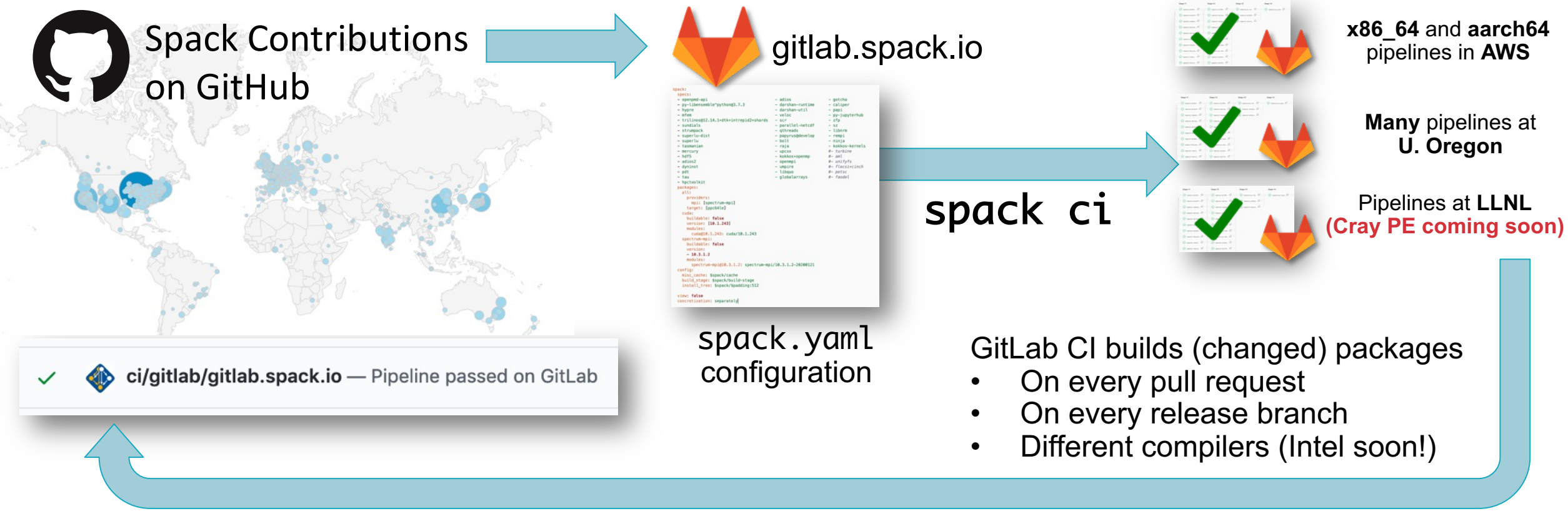
Spack 0.18 Roadmap: compilers as dependencies

- **We need deeper modeling of compilers to handle compiler interoperability**
  - libstdc++, libc++ compatibility
  - Compilers that depend on compilers
  - Linking executables with multiple compilers
- **First prototype is complete!**
  - We've done successful builds of some packages using compilers as dependencies
  - We need the new concretizer to move forward!
- **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



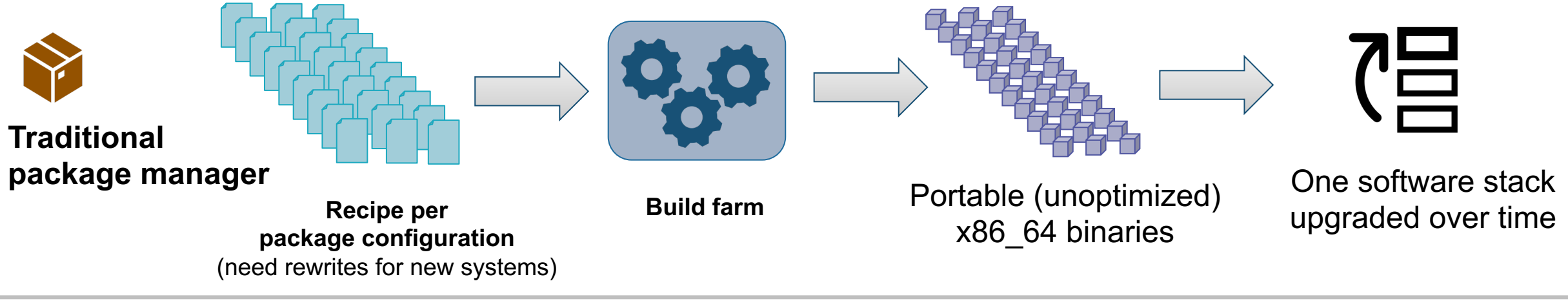
Compilers and runtime libs fully modeled as dependencies

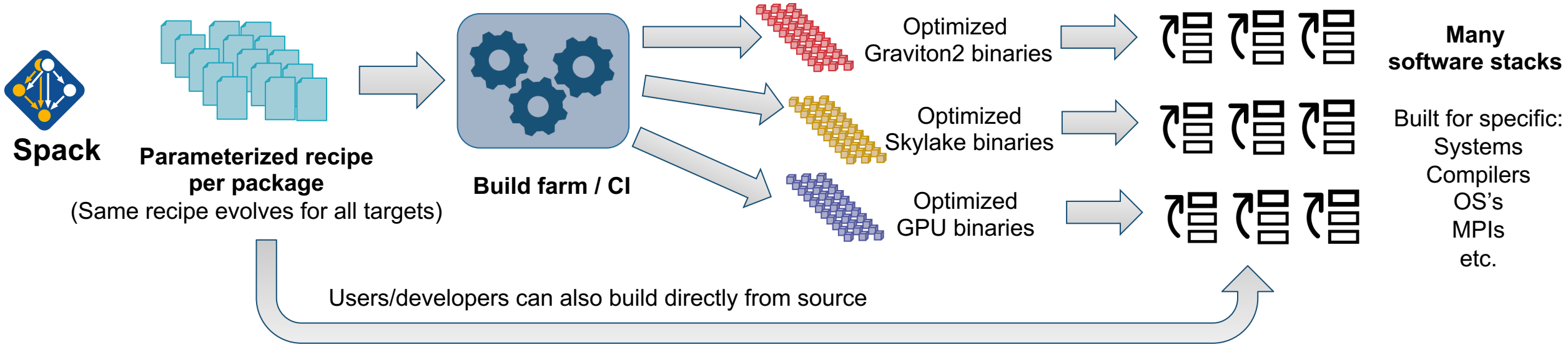
Spack v0.18 roadmap: provide a public binary build cache



- **Our security model supports untrusted contributions from forks**
  - Sandboxed build caches for PR builds
  - Authoritative builds on **`develop`** only *after* approved merge

We aim to lower the burden of maintaining a binary distribution and make it easy to mix source builds with binaries.





Easyconfigs vs. Spack Packages

Builtin spack packages

~6,000 packages **162k lines of Python**

Language	files	blank	comment	code
Python	6189	58957	59655	161824
diff	1349	6691	34550	57965

Easyconfigs (not including Easyblocks)

~2,500 packages **449k lines of Python**

Language	files	blank	comment	code
Python	14732	119997	56387	449023
diff	1483	12961	71640	104540

- Tradeoff is testing: Easyconfigs are "one" thing per config, Spack package.py's are many things.
  - Build farm is our solution for this: ensure there are stable binary releases

Under the BUILD project, we are looking at building models for build reliability

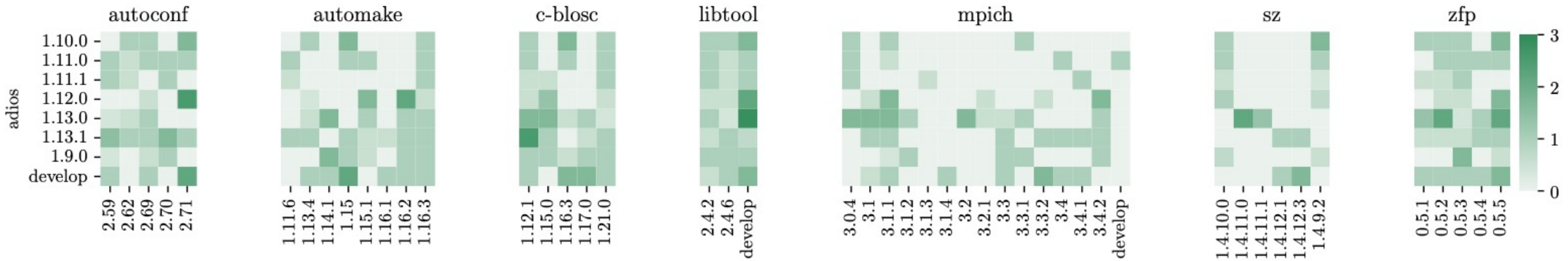


Figure 8: Heatmap of Adios and its dependencies with scores indicating which version pairs are highly likely to build.

- Basic premise: humans can't generate all the compatibility constraints
  - Version ranges, conflicts, in Spack packages not precise
  - We rely on maintainers to get these right
  - How much of this burden can we automate?
- Plot shows experiments with ADIOS – we've built a model for build success
  - Aim is to include this type of data in the solver to improve source builds.

Approved for public release