Spack:
Package Management for HPC

Todd Gamblin (he/him)
Lawrence Livermore National Laboratory

Software Productivity and Sustainability track @ Argonne Training Program on Extreme-Scale Computing summer school

Contributors: Todd Gamblin (LLNL)

License, Citation and Acknowledgements

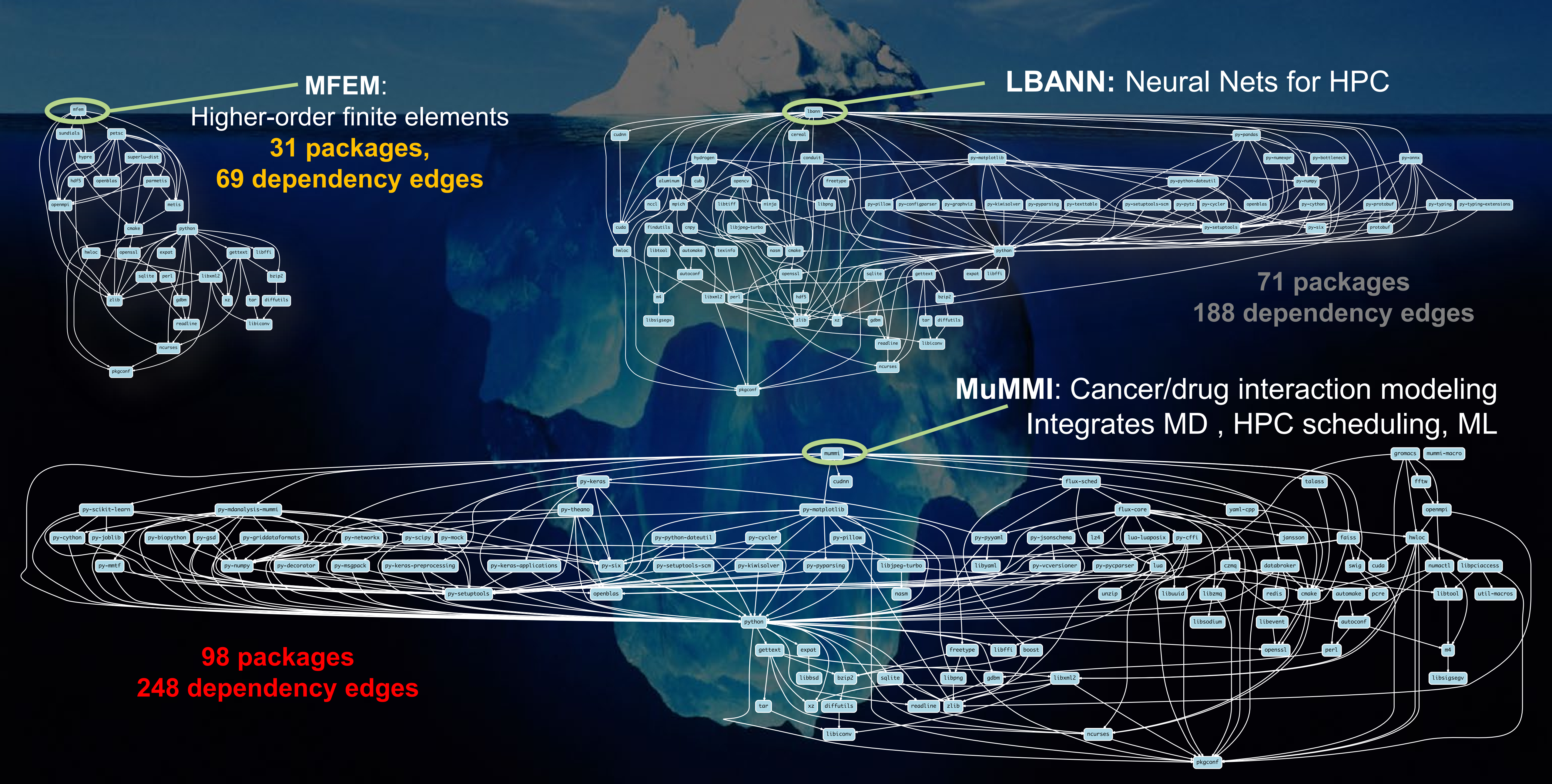
License and Citation

- **The requested citation the overall tutorial is: David E. Bernholdt, Anshu Dubey, Todd Gamblin, Jared O'Neal, and Boyana R. Norris, Software Productivity and Sustainability track, in Argonne Training Program on Extreme-Scale Computing, St. Charles, Illinois, 2022. DOI: 10.6084/m9.figshare.20416215.**
- Individual modules may be cited as *Speaker, Module Title*, in Better Scientific Software tutorial, ISC, 2022 …

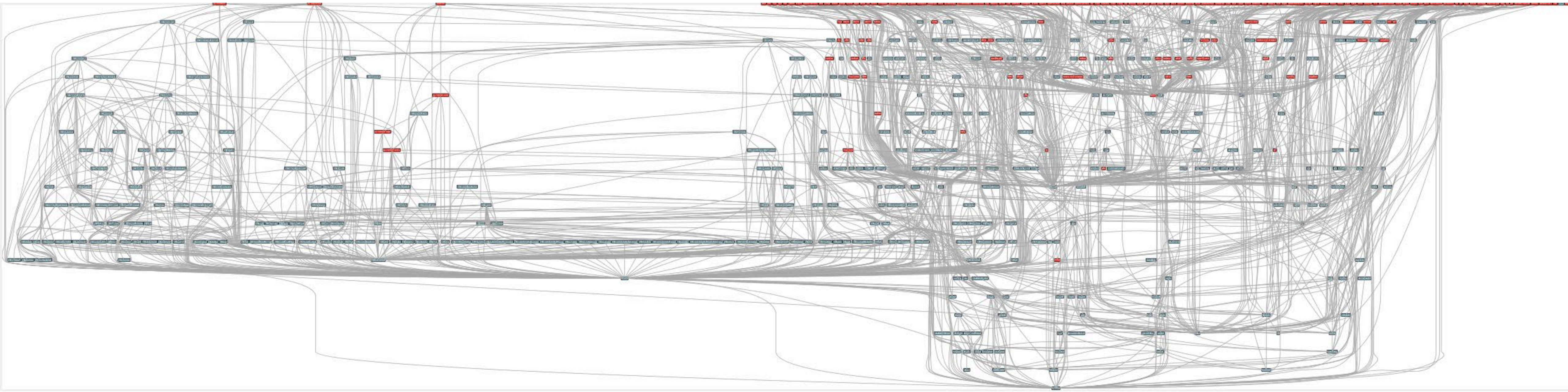
Acknowledgements

- This work was supported by the U.S. Department of Energy Office of Science, Office of Advanced Scientific Computing Research (ASCR), and by the Exascale Computing Project (17-SC-20-SC), a collaborative effort of the U.S. Department of Energy Office of Science and the National Nuclear Security Administration.
- This work was performed in part at the Argonne National Laboratory, which is managed by UChicago Argonne, LLC for the U.S. Department of Energy under Contract No. DE-AC02-06CH11357.
- This work was performed in part at the Lawrence Livermore National Laboratory, which is managed by Lawrence Livermore National Security, LLC for the U.S. Department of Energy under Contract No. DE-AC52-07NA27344.
- This work was performed in part at the Los Alamos National Laboratory, which is managed by Triad National Security, LLC for the U.S. Department of Energy under Contract No.89233218CNA000001
- This work was performed in part at the Oak Ridge National Laboratory, which is managed by UT-Battelle, LLC for the U.S. Department of Energy under Contract No. DE-AC05-00OR22725.
- This work was performed in part at Sandia National Laboratories. Sandia National Laboratories is a multi-mission laboratory managed and operated by National Technology and Engineering Solutions of Sandia, LLC., a wholly owned subsidiary of Honeywell International, Inc., for the U.S. Department of Energy's National Nuclear Security Administration under contract DE-NA0003525.
- This work was performed in part at University of Oregon through a subcontract with Argonne National Laboratory.

HPC simulations rely on icebergs of dependency libraries



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

Current

Oak Ridge National Lab
Power9 / NVIDIA

RIKEN
Fujitsu/ARM a64fx

Lawrence Berkeley National Lab
AMD Zen / NVIDIA

Oak Ridge National Lab
AMD Zen / Radeon

Upcoming

Argonne National Lab
Intel Xeon / Xe

Lawrence Livermore National Lab
AMD Zen / Radeon

IDEAS productivity

ECP EXASCALE COMPUTING PROJECT

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

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
```

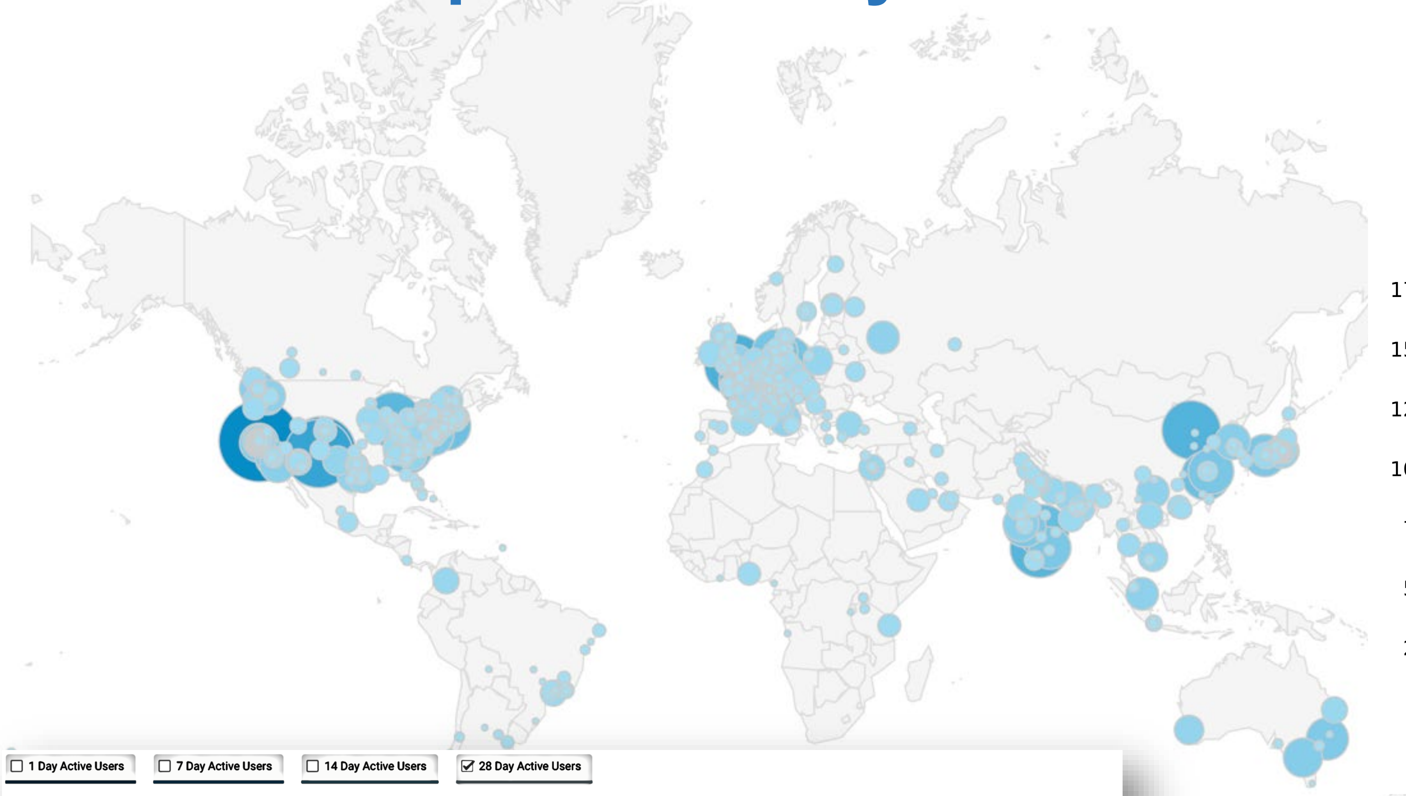
Simple syntax enables complex installs

```
$ 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
```

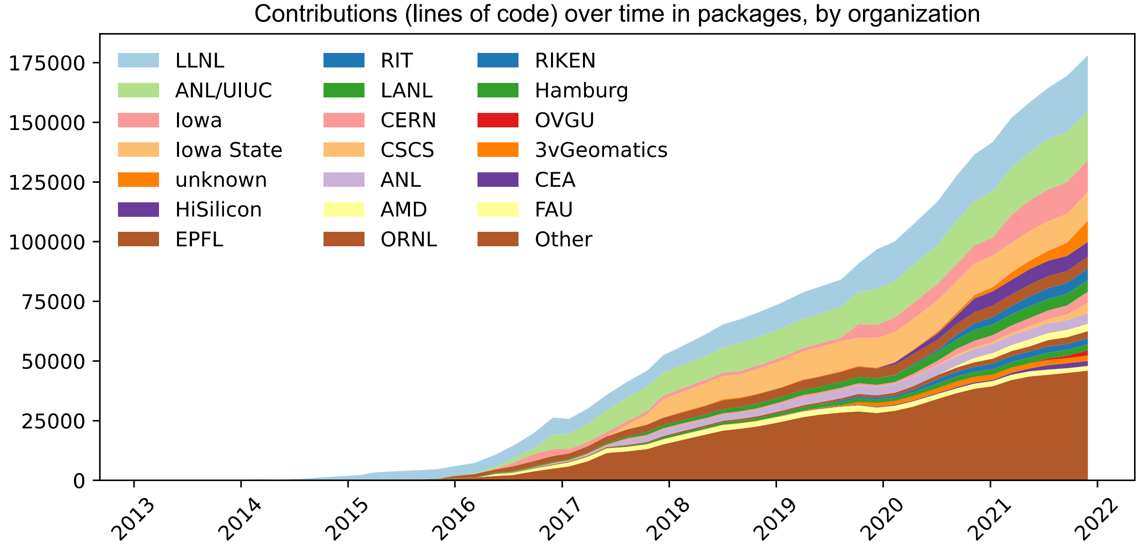
github.com/spack/spack

- 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)

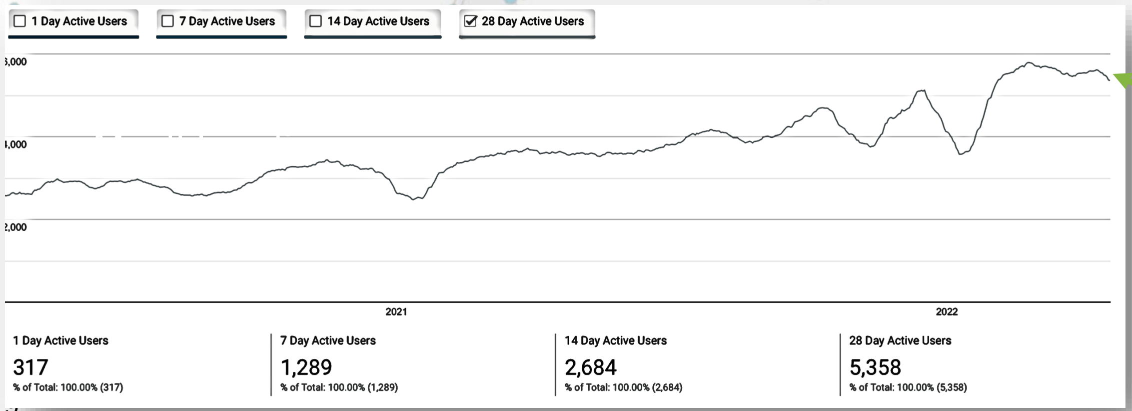
Spack sustains the HPC software ecosystem with the help of its many contributors



6,400+ software packages
Over 1,000 contributors



Most package contributions are ***not*** from DOE!



Nearly 6,000 monthly active users (per documentation site)

IDEAS productivity

ECP EXASCALE COMPUTING PROJECT

Spack provides a *spec* syntax to describe customized installations

$ spack install mpileaks	unconstrained
$ spack install mpileaks@3.3	@ custom version
$ spack install mpileaks@3.3 %gcc@4.7.3	% custom compiler
$ spack install mpileaks@3.3 %gcc@4.7.3 +threads	+/- build option
$ spack install mpileaks@3.3 cppflags="-O3 –g3"	set compiler flags
$ spack install mpileaks@3.3 target=zen2	set target microarchitecture
$ spack install mpileaks@3.3 ^mpich@3.2 %gcc@4.9.3	^ dependency information

- Each expression is a ***spec*** for a particular configuration
  - Each clause adds a constraint to the spec
  - Constraints are optional – specify only what you need.
  - Customize install on the command line!
- Spec syntax is recursive
  - Full control over the combinatorial build space

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

Spack packages are *templates*
They use a simple Python DSL to define how to build

```
from spack import *

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

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

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

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

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

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

  def install(self, spec, prefix):
    # Kripke does not provide install target, so we have to copy
    # things into place.
    mkdirp(prefix.bin)
    install('../spack-build/kripke', prefix.bin)
```

Base package (CMake support)

Metadata at the class level

Versions

Variants (build options)

Dependencies (same spec syntax)

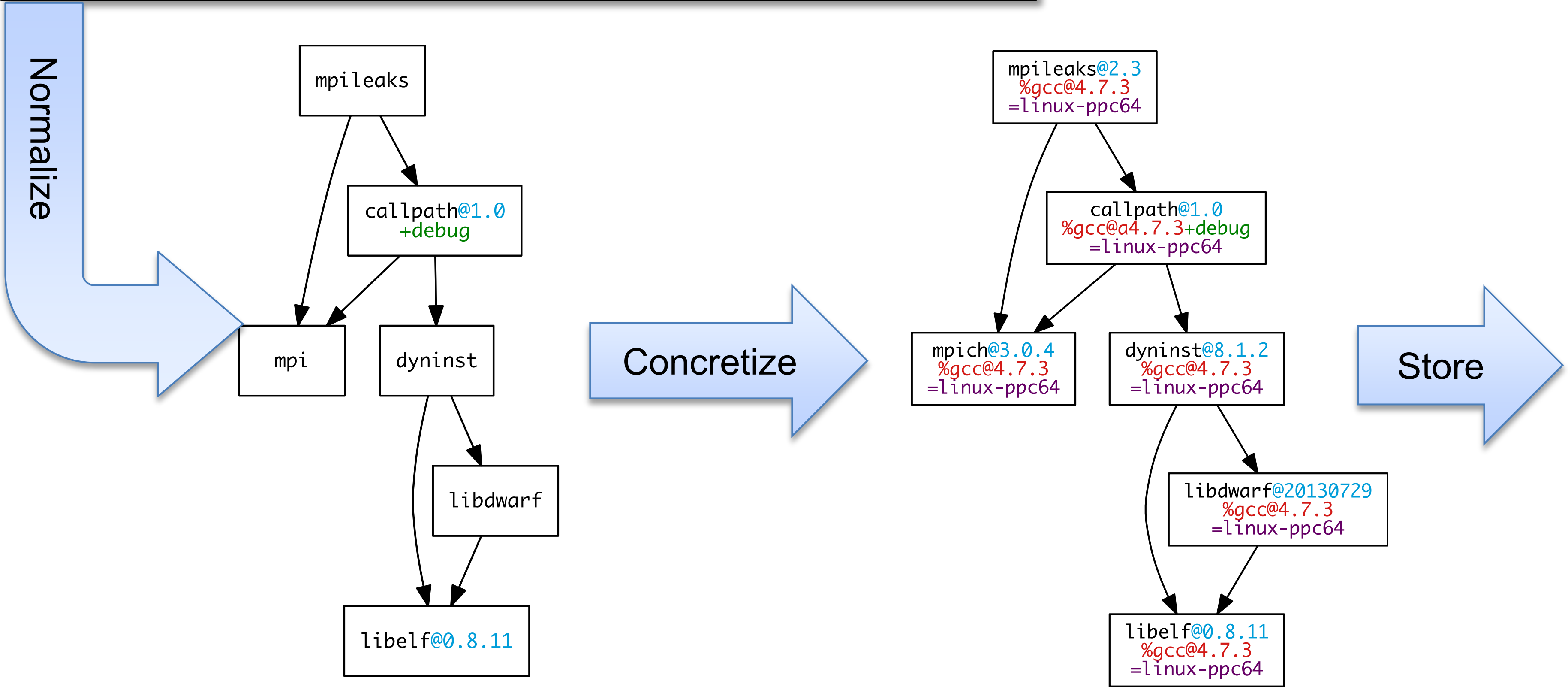
Install logic in instance methods

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

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



Abstract, normalized spec with some dependencies

Concrete spec is fully constrained and can be passed to install

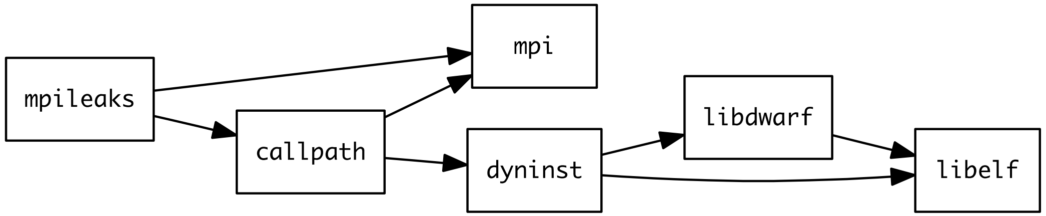
spec.yaml

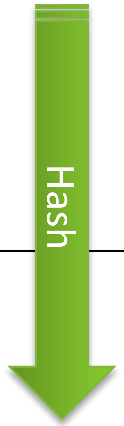

```yaml
spec:
- mpileaks:
   arch: linux-x86_64
   compiler:
    name: gcc
    version: 4.9.2
   dependencies:
    adept-utils: kszrtkpbzac3ss2ixcjkcorlaybnptp4
    callpath: bah5f4h4d2n47mgycej2mtrnrivvxy77
    mpich: aa4ar6ifj23yijqmdabeakpejcli72t3
   hash: 33hjjhxi7p6gyzn5ptgyes7sghyprujh
   variants: {}
   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
...
```


Detailed provenance stored with installed package

Spack handles combinatorial software complexity

Dependency DAG





Installation Layout

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

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

IDEAS productivity ECP EXASCALE COMPUTING PROJECT

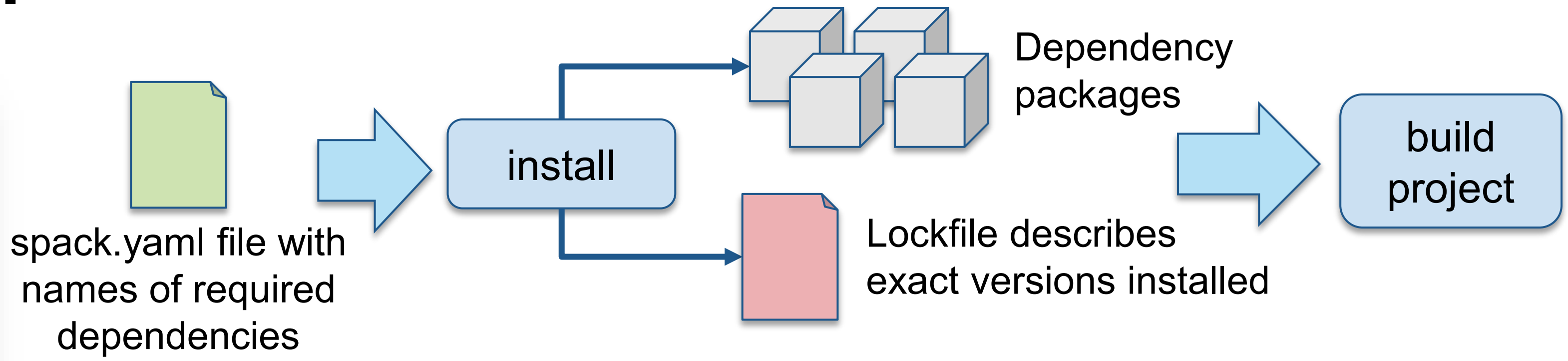
Spack environments enable users to build customized stacks from an abstract description

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



- 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

IDEAS productivity

ECP EXASCALE COMPUTING PROJECT

Spack can generate multi-stage container build recipes

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

  container:
    # Select the format of the rec
    # singularity or anything else
    format: docker

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

    # Whether or not to strip bina
    strip: true

    # Additional system packages t
    os_packages:
    - libgomp

    # Extra instructions
    extra_instructions:
      final: |
RUN echo 'export PS1="\[$(tput bol

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

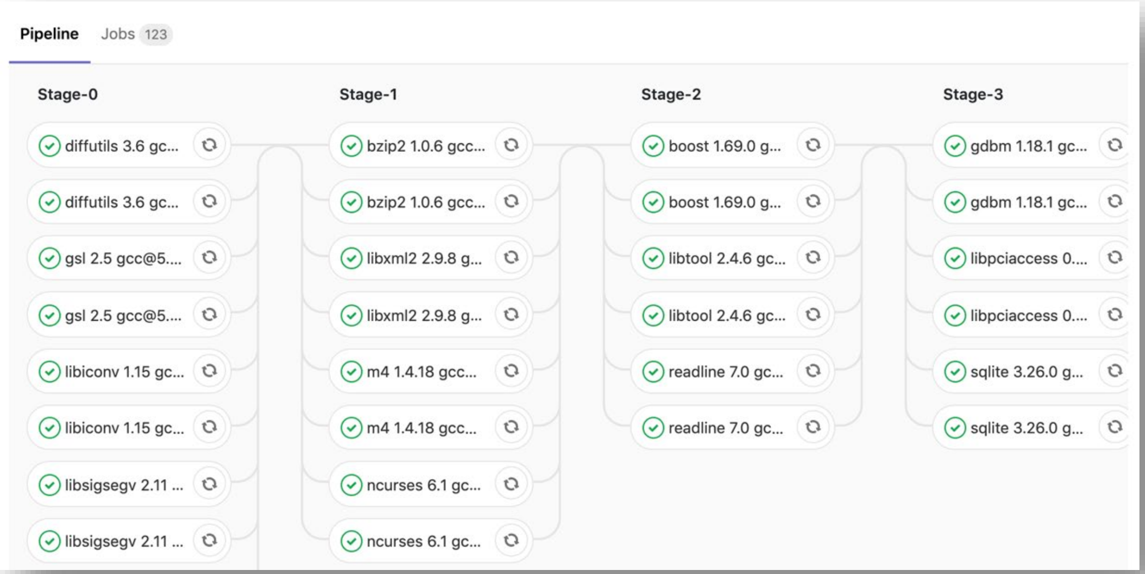
```dockerfile
# Build stage with Spack pre-installed and ready to be used
FROM spack/centos7:latest as builder

# 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 unecessary 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_envi

              e -y && yum install -y epel-release && yum update -y
              l -y libgomp \
&&             /cache/yum  && yum clean all

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

- Any Spack environment can be bundled into a container image
  - Optional container section allows finer-grained customization
- Generated Dockerfile uses multi-stage builds to minimize size of final image
  - Strips binaries
  - Removes unneeded build deps with spack gc
- Can also generate Singularity recipes

spack containerize

Spack has GitLab CI integration to automate package build pipelines

- Builds on Spack environments
  - Support auto-generating GitLab CI jobs
  - Can run in a Kube cluster or on bare metal runners at an HPC site
  - Sends progress to CDash



Kitware

```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_18.04
      - 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
```

E4S is ECP's curated, Spack-based software distribution

- **E4S is just a set of Spack packages**
  - **60+ packages (297 including dependencies)**
  - **Growing to include all of ST and more**
- Users can install E4S packages:
  - In their home directory
  - In a container
- Facilities can install E4S packages:
  - On bare metal
  - In a container
- Users and facilities can choose parts they want
  - **spack install** only the packages you want
  - Or just edit the list of packages (and configurations) you want in a **spack.yaml** file

```
spack:
  specs:
  - openpmd-api
  - py-libensemble^python@3.7.3
  - hypre
  - mfem
  - trilinos@12.14.1+dtk+intrepid2+shards
  - sundials
  - strumpack
  - superlu-dist
  - superlu
  - tasmanian
  - mercury
  - hdf5
  - adios2
  - dyninst
  - pdt
  - tau
  - hpctoolkit
  - adios
  - darshan-runtime
  - darshan-util
  - veloc
  - scr
  - parallel-netcdf
  - qthreads
  - papyrus@develop
  - bolt
  - raja
  - upcxx
  - kokkos+openmp
  - openmpi
  - umpire
  - libquo
  - globalarrays
  - gotcha
  - caliper
  - papi
  - py-jupyterhub
  - zfp
  - sz
  - libnrm
  - rempi
  - ninja
  - kokkos-kernels
  #- turbine
  #- aml
  #- unifyfs
  #- flecsi+cinch
  #- petsc
  #- faodel
  packages:
    all:
      providers:
        mpi: [spectrum-mpi]
      target: [ppc64le]
    cuda:
      buildable: false
      version: [10.1.243]
      modules:
        cuda@10.1.243: cuda/10.1.243
    spectrum-mpi:
      buildable: false
      version:
      - 10.3.1.2
      modules:
        spectrum-mpi@10.3.1.2: spectrum-mpi/10.3.1.2-20200121
  config:
    misc_cache: $spack/cache
    build_stage: $spack/build-stage
    install_tree: $spack/$padding:512

  view: false
  concretization: separately
```

E4S manifest (**spack.yaml**)

More on E4S at https://e4s.io

spack test: write tests directly in Spack packages, so that they can evolve with the software

```
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')
```

Tests are part of a regular Spack recipe class

Easily save source code from the package

User just defines a test() method

Retrieve saved source.
Link a simple executable.

Spack ensures that cc is a compatible compiler

Run the built smoke test and verify output

Run programs installed with package

spack external find (new in v0.15, updated for 0.16)

```python
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))
```

Logic for finding external installations in package.py

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

packages.yaml configuration

- Spack has has had compiler detection for a while
  - Finds compilers in your PATH
  - Registers them for use
- We can find any package now
  - Package defines:
    - possible command names
    - how to query the command
  - Spack searches for known commands and adds them to configuration
- Easily enable rapid setup of tools in an environment

IDEAS productivity

ECP EXASCALE COMPUTING PROJECT

spack develop lets developers work on many packages at once

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

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


$ ls
spack.yaml    axom/    mfem/


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

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

Spack v0.18.0 was released at ISC in early June!

- **Major new features:**
  1. --reuse enabled by default
     - Reuse installed packages and build caches
     - Use spack install --fresh to get the old behavior
  2. Finer-grained spec hash + provenance
  3. Better error messages
  4. Unify *when possible* in environments
  5. Cray manifest support
  6. Windows support
  7. New binary format + hardened package signing
  8. Bootstrap mirror generation (for air gaps)
  9. Makefile generation
  10. Conditional variant values and sticky variants

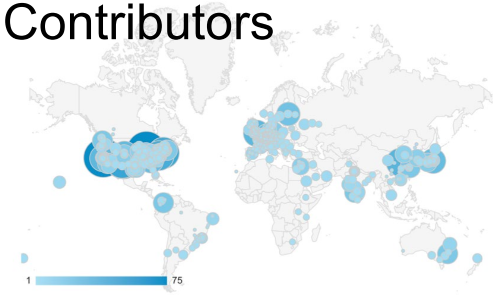
github.com/spack/spack

377 contributors to packages!
85 contributors to core!

Concretization is at the core of Spack!

This problem is NP-hard!

Contributors



- new versions
- new dependencies
- new constraints

package.py repository

concretizer

spack developers — yaml — default config packages.yaml

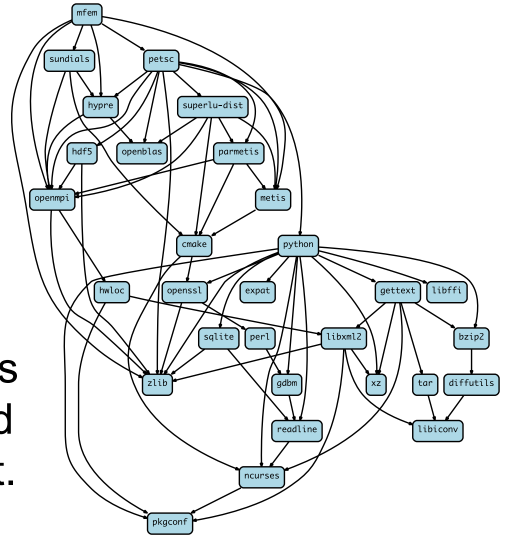
admins, users — yaml — preferences config packages.yaml

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.



IDEAS productivity

ECP EXASCALE COMPUTING PROJECT

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).
```

IDEAS productivity

ECP EXASCALE COMPUTING PROJECT

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's concretizer is now implemented in ASP

- Used Clingo, the Potassco grounder/solver package
- ASP program has 2 parts:
  1. Large list of facts generated from package recipes (problem instance)
     - 60k+ facts is typical – includes dependencies, options, etc.
  2. Small logic program (~700 lines of ASP code)
- Algorithm (the part we write) is conceptually simpler:
  – Generate facts for all possible dependencies
  – Send facts and our logic program to the solver
  – Rebuild a DAG from the results

```
%----------------------------------------------------------------------------
% 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

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

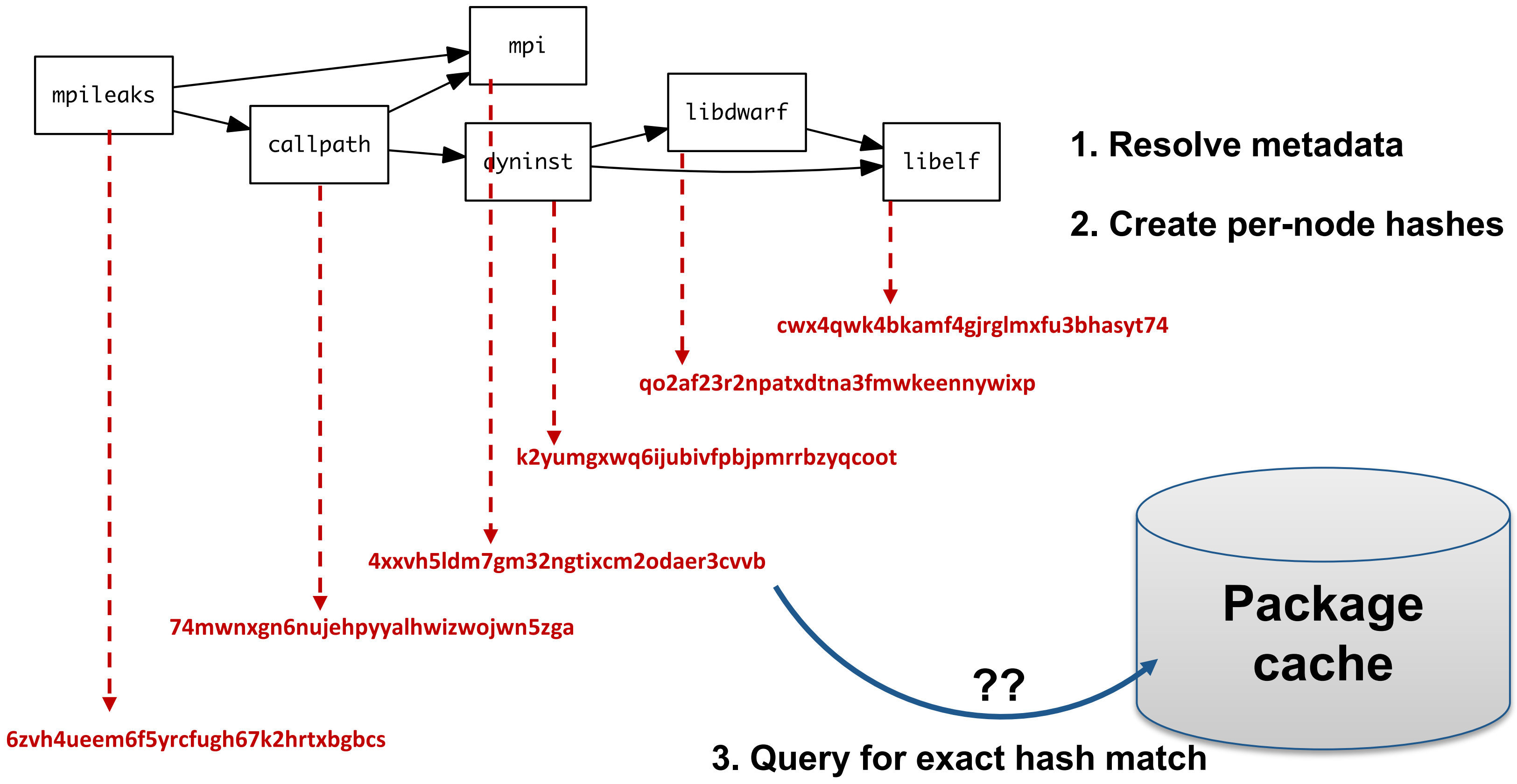
dependency on cuda, but only if cuda is enabled

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) }.
```

IDEAS productivity

EXASCALE COMPUTING PROJECT

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
```

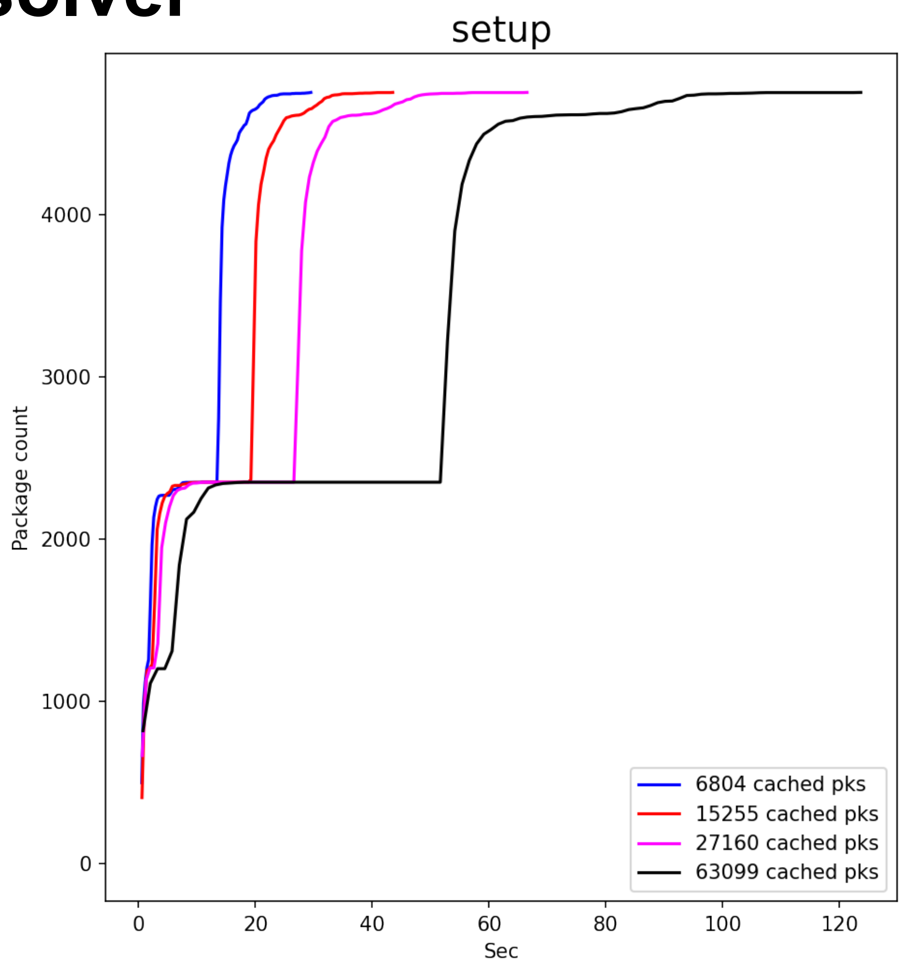
Pure hash-based reuse: all misses

```
(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
```

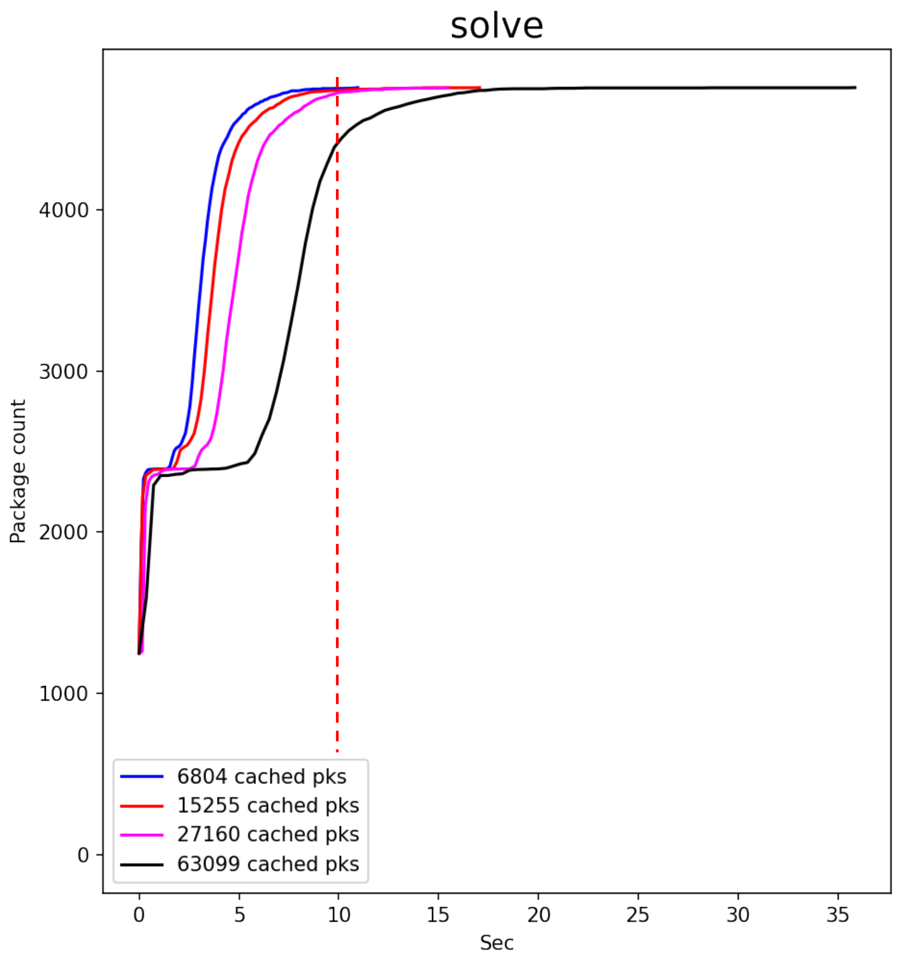
With --reuse: 16 packages were reusable

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)

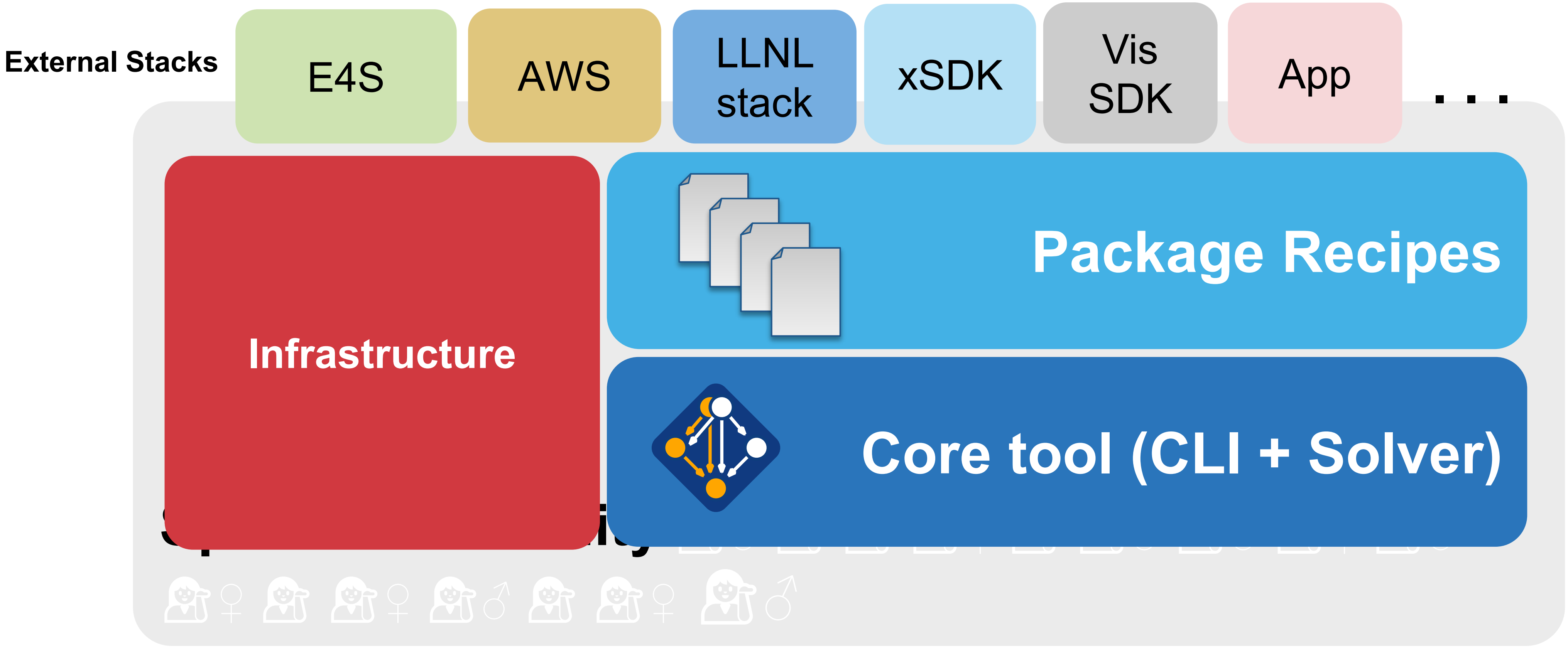


Even with 63k packages in a repo, nearly all package solves take < 10 sec

IDEAS productivity

ECP EXASCALE COMPUTING PROJECT

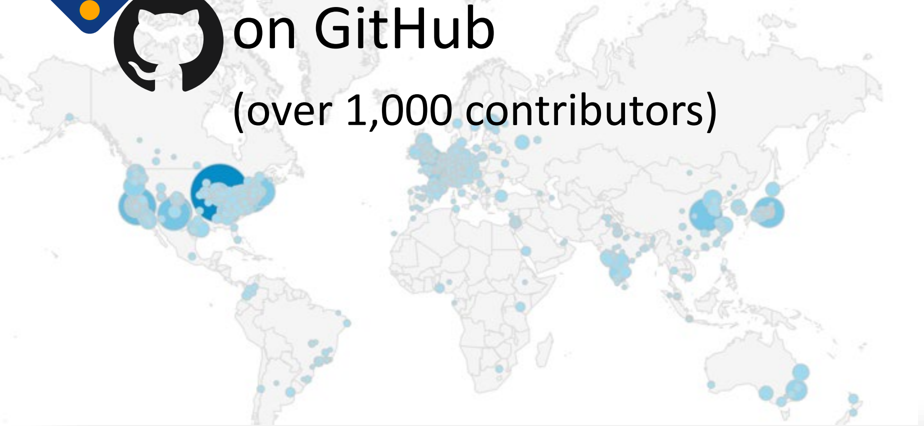
What does the Spack project look like?



CI has made Spack builds *much* more reliable!

Spack Contributions on GitHub

(over 1,000 contributors)



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

gitlab.spack.io

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

spack ci

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

ppc64le, GPU pipelines at **U. Oregon**

Pipelines at **LLNL** **(Cray PE soon)**

GitLab CI builds (changed) packages

- On every pull request
- On every release branch

Do users really need to build from source?

IDEAS productivity

ECP EXASCALE COMPUTING PROJECT

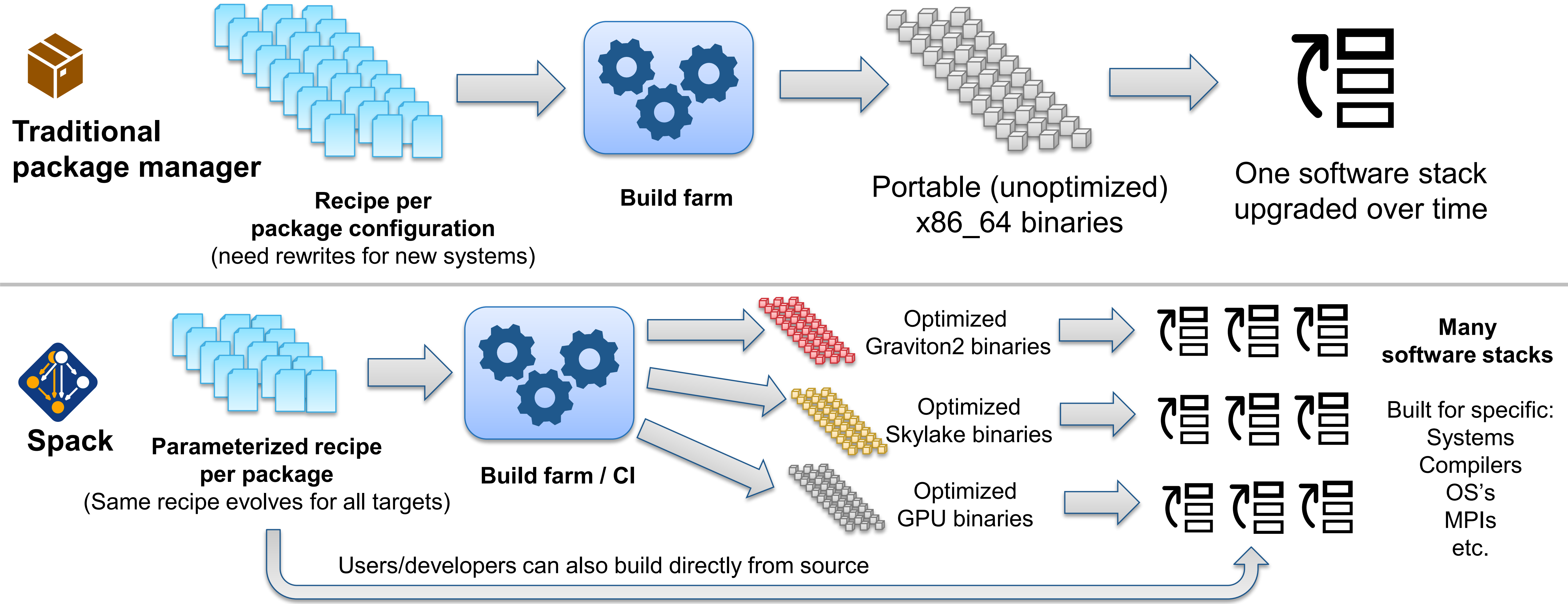
With v0.18, Spack has a public binary cache

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

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

- Over 3,000 builds in the cache so far:
  - Amazon Linux 2 x86_64_v4
  - Amazon Linux 2 aarch64
  - Amazon Linux 2 graviton2
  - Ubuntu 18.04 x86_64

Do we trust binaries?

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



Our infrastructure enables us to sustainably manage a binary distribution

Untrusted S3 buckets

Internal per-PR build caches

- github/pr-28468
- github/pr-28469
- …

Public, signed binaries in CloudFront distribution

https://binaries.spack.io

- develop
- releases/v0.18
- …

Contributors submit package changes

- Iterate on builds in PR
- Caches prevent unnecessary rebuilds

Maintainers review PRs

- Verify PR build succeeded
- Review package code
- Merge to develop

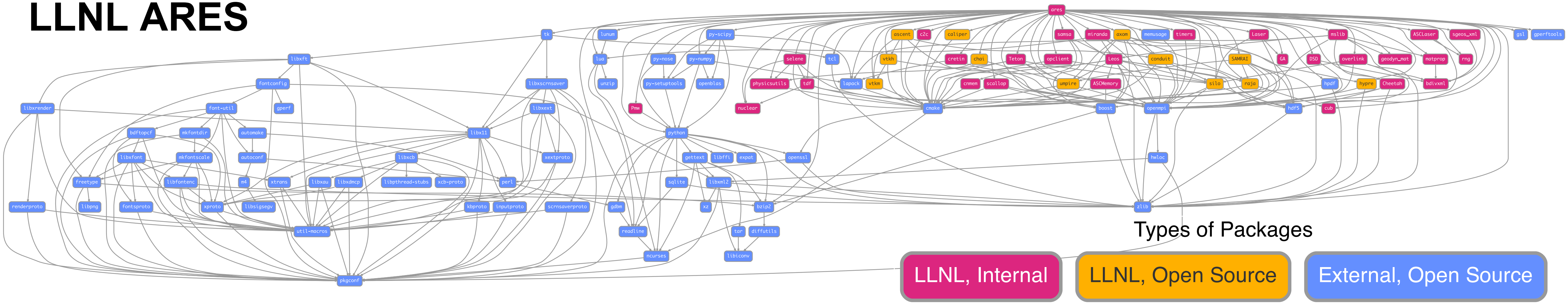
Rebuild and Sign

- Published binaries built ONLY from approved code
- Protected signing runners
- Ephemeral keys

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

Why should we care about this for our HPC codes?

LLNL ARES

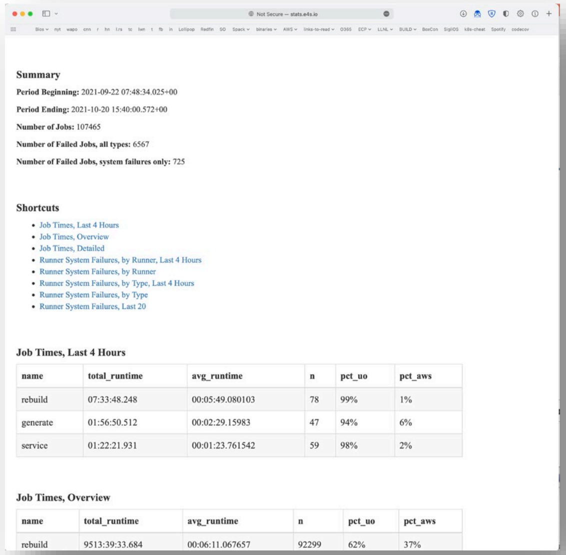


LLNL, Internal	LLNL, Open Source	External, Open Source
30	**12**	**71**

- Our codes use a lot of external software
  - *Most* packages are external open source
  - Many LLNL packages are also open source and developed in the open
- We *cannot* replace all these OSS components with our own
  - How do we vet all these components?
- **Key question:** Who/what do you trust to validate the components?
  - Current processes are not scalable and not automated!

We will continue scaling this infrastructure out!

- We are doing 40k builds per week!
  - There are lots of optimizations left to do on the build pipelines
  - We think we can eventually scale to all 6,400 Spack packages
- Goal: make source builds unnecessary for most users
  - Source builds are optimized for x86_64_v4 (avx512), graviton, etc.
  - Source builds will still be seamless – key for reproducibility
  - Use `spack develop` to tweak (almost) any binary you can install
- We will keep scaling OS, compiler, and arch support
  - Current crop of compilers and OS's is a bit old – expect a refresh
  - Cray PE build coming soon!
- Amazon Linux 2 builds work on AWS ParallelCluster NOW!



Not Secure — stats.e4s.io

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

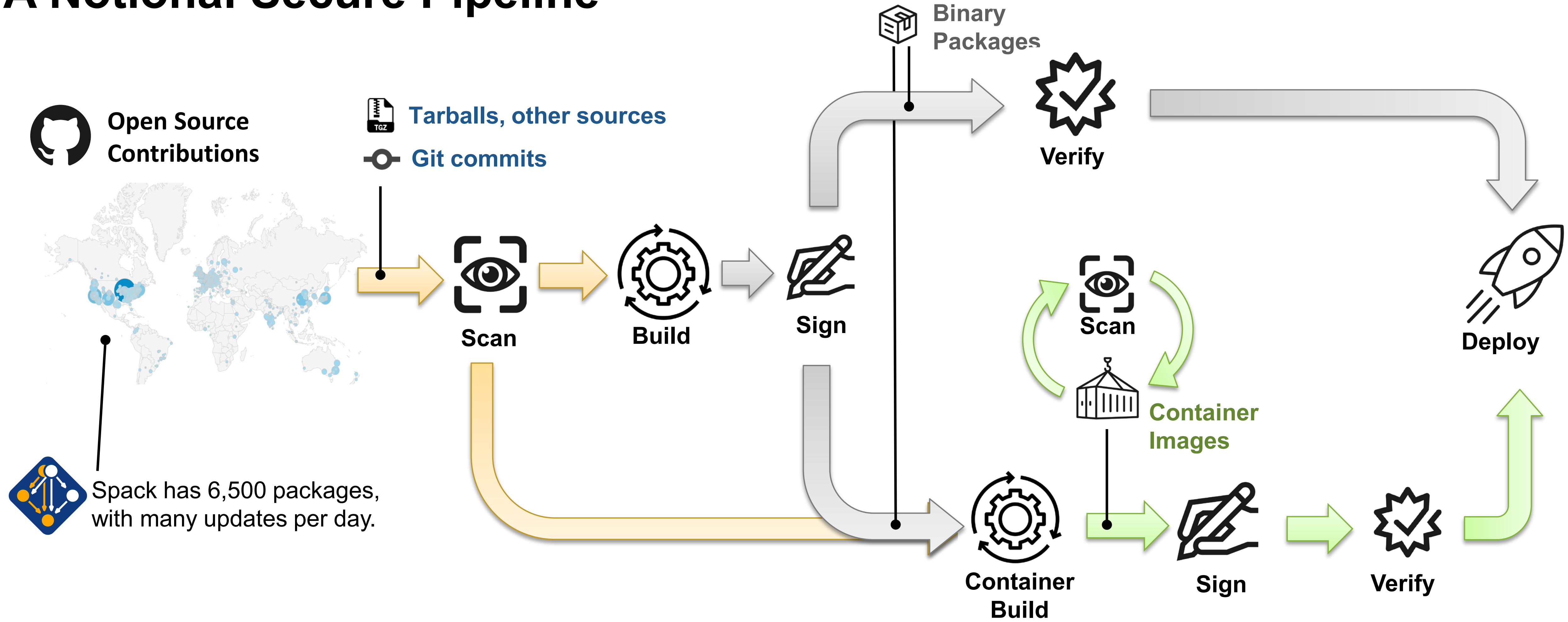
name	total_runtime	avg_runtime	n	pct_uo	pct_aws
rebuild	07:33:48.248	00:05:49.080103	78	99%	1%
generate	01:56:50.512	00:02:29.15983	47	94%	6%
service	01:22:21.931	00:01:23.761542	59	98%	2%

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%

Build stats at https://stats.e4s.io

A Notional Secure Pipeline



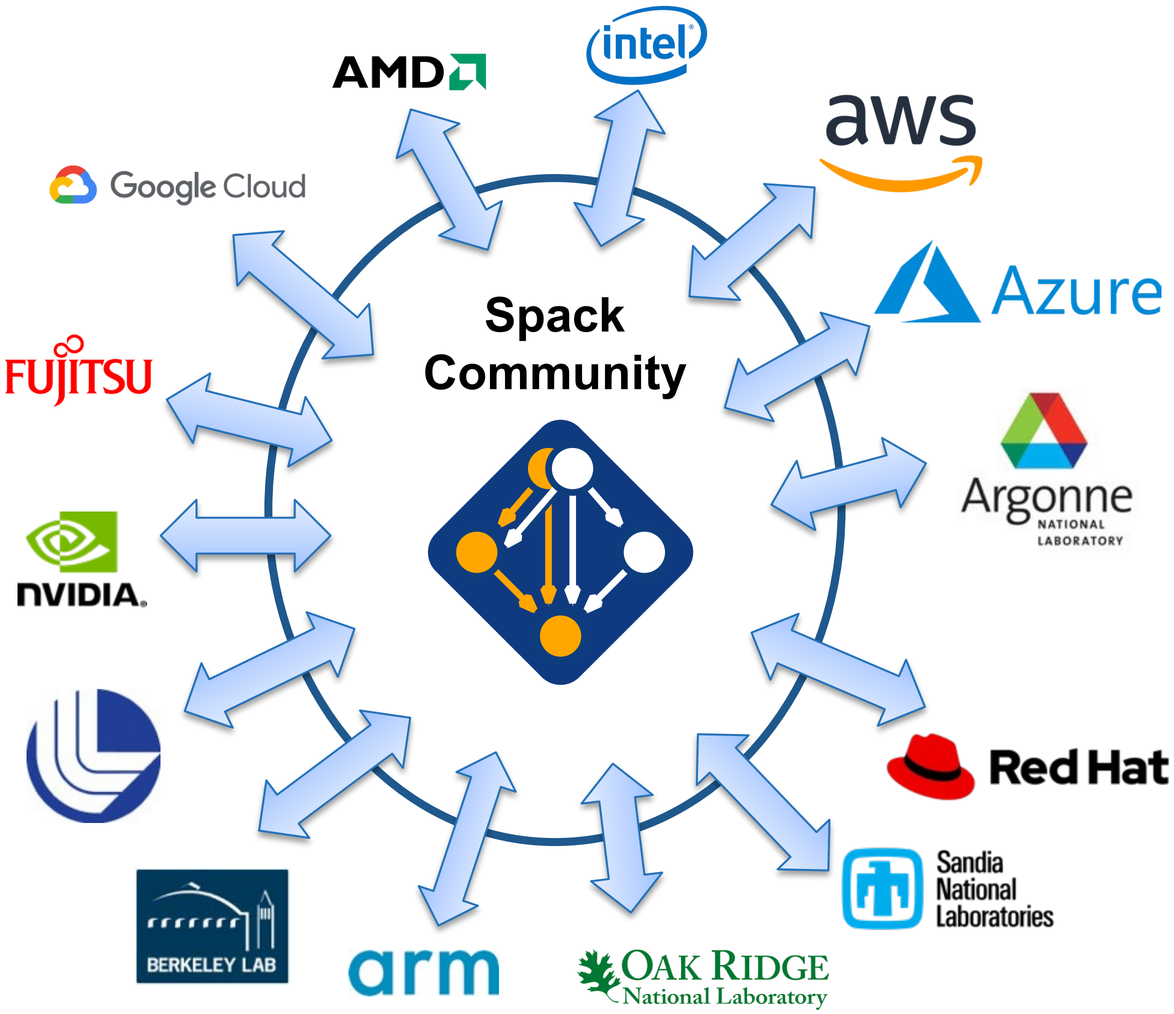
- We are working to establish a set of guidelines for supply chain integrity
  - Labs are trending towards GitLab, Spack for HPC
  - Standard container formats can help with scanning
  - Standard Software Bill of Materials (SBOM) format could help sites cross-validate codes
- Spack can help to standardize some of this.

IDEAS productivity

ECP EXASCALE COMPUTING PROJECT

Spack's long-term strategy is based around broad adoption and collaboration

- **Not sustainable without a community**
  - Broad adoption incentivizes contributors
  - Cloud resources and automation absolutely necessary
- **Preserves build knowledge in a cross-platform, reusable way**
  - Minimize rewriting recipes when porting
- **CI ensures builds continue to work as packages evolve**
  - Keep packages flexible but verify key configurations
- **Growing contributor base and automation are the top priorities**
  - **377 contributors to 0.18 release!**



IDEAS productivity

ECP EXASCALE COMPUTING PROJECT

Other resources

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

Tutorial
https://spack-tutorial.readthedocs.io

Documentation
https://spack.readthedocs.io

Slack (1,900+ users)
https://slack.spack.io

Follow us on Twitter!
@spackpm

IDEAS productivity

ECP EXASCALE COMPUTING PROJECT

Questions?

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