Long-term plans for Spack

NITRD MAGIC meeting on Software Sustainability
December 1, 2021

Todd Gamblin
Advanced Technology Office
Livermore Computing

LLNL-PRES-826942
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

Lawrence Livermore National Laboratory

What is Spack?

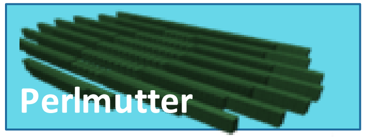
- Supercomputing PACKage manager
- Manages scientific software ecosystem
  - With flexibility needed to build packages for diverse HPC machines
- Language-agnostic
  - Focused originally on build from source
  - Now focused on both source and binary
- Has become a de-facto standard for packaging HPC software

Spack builds for machines like these
(and for your laptop/cloud node/cluster)

Current top systems

RIKEN
Fujitsu/ARM a64fx

ORNL/LLNL
Power9 / **NVIDIA GPU**



Lawrence Berkeley National Lab
AMD **Zen** / **NVIDIA GPU**

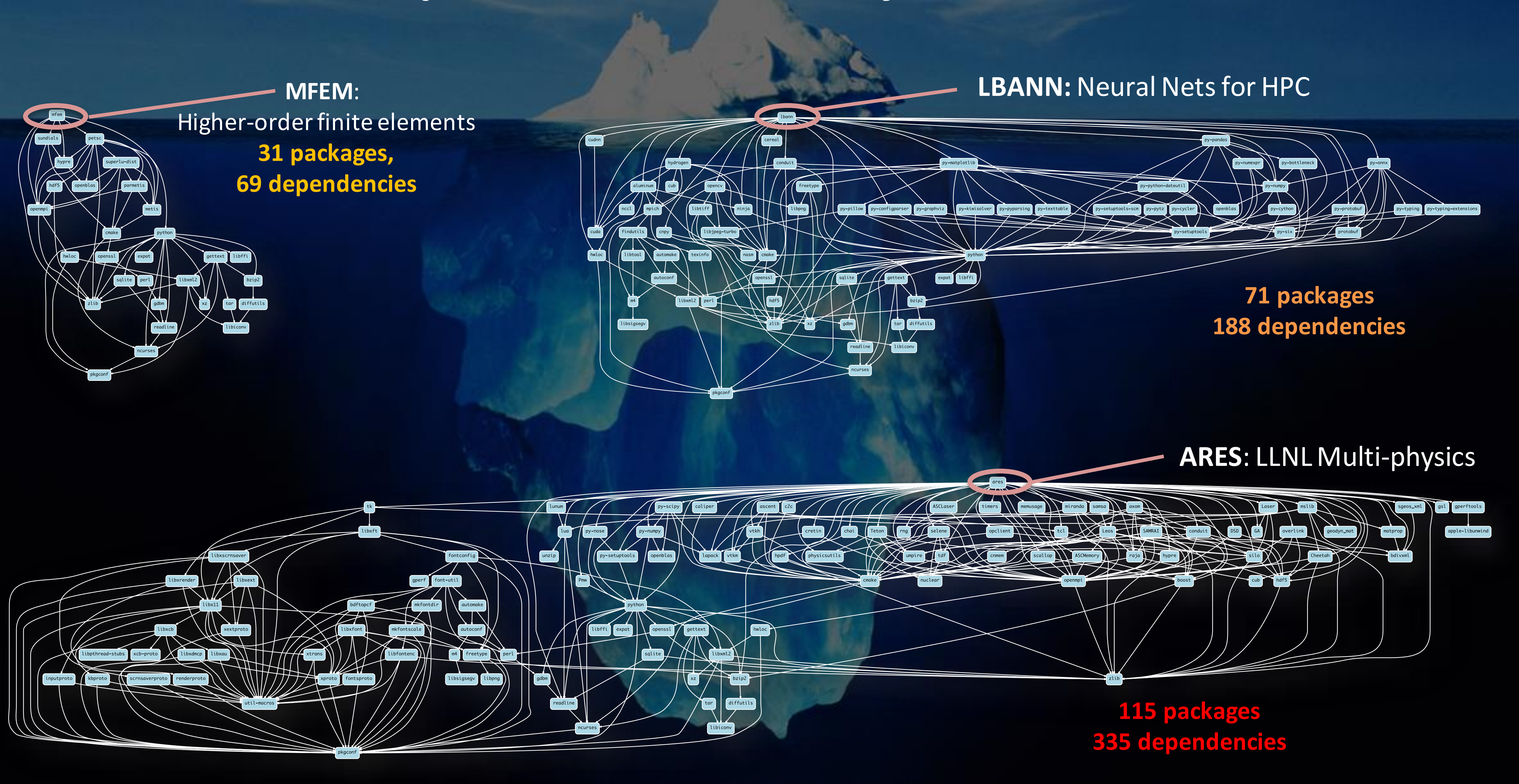
Machines coming soon

Argonne National Lab
Intel **Xeon** / **Xe**

Oak Ridge National Lab
AMD **Zen** / **MI200 GPU**

Lawrence Livermore National Lab
AMD **Zen** / **AMD GPU**

Scientific libraries span C++, C, Fortran, Python, Lua, and more



What does the Spack project look like?

What does the Spack project look like?

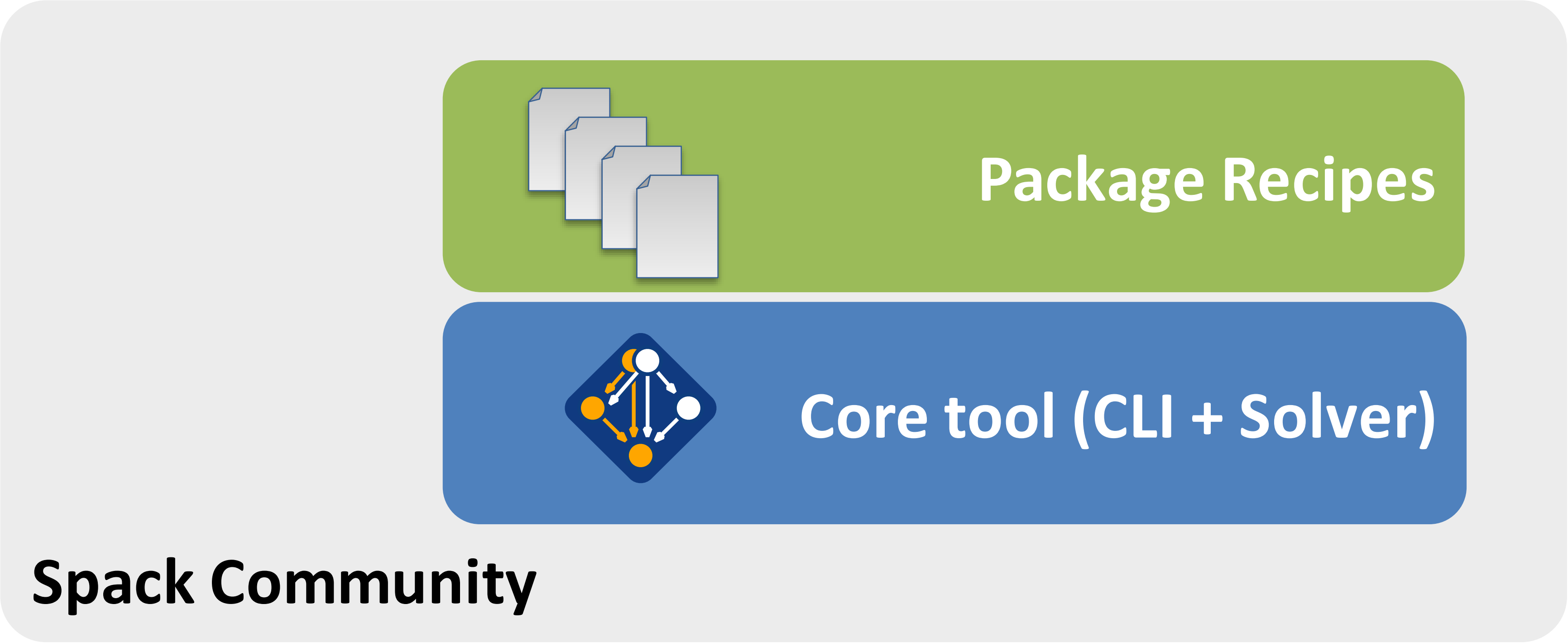
Spack Community

What does the Spack project look like?

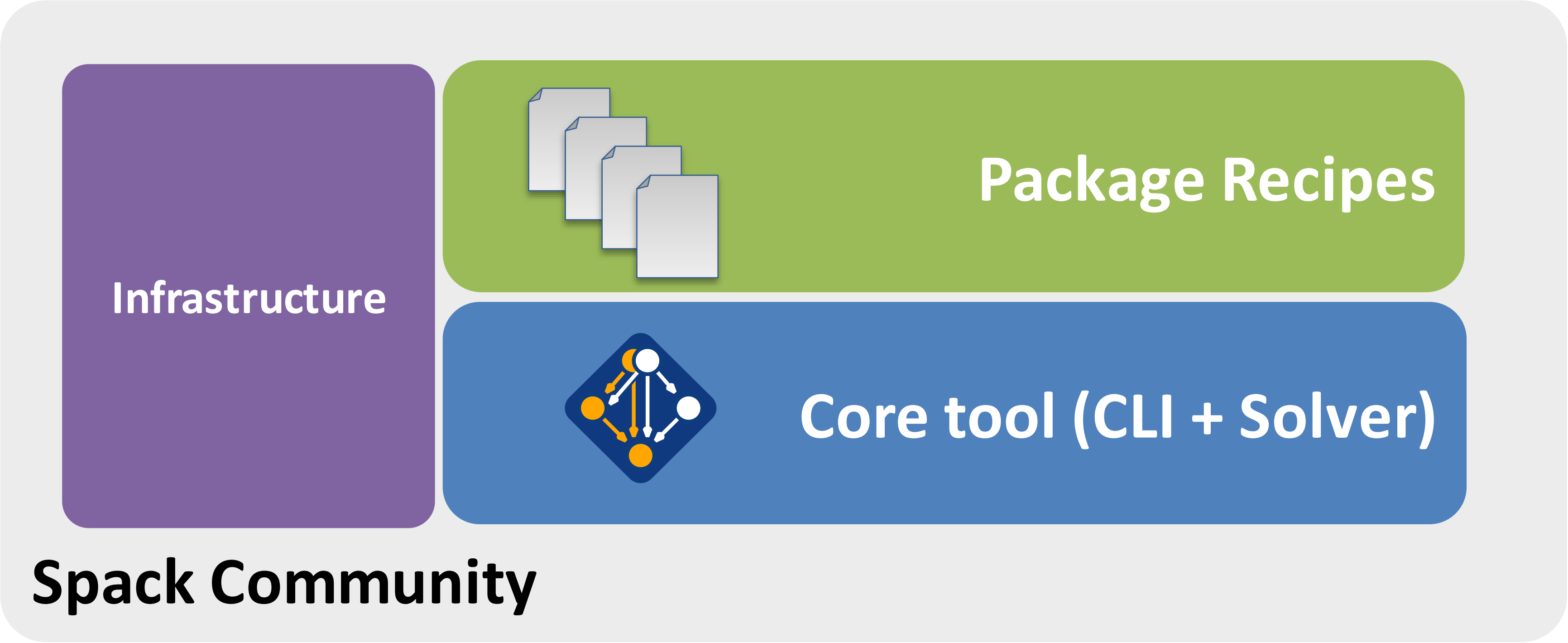
Core tool (CLI + Solver)

Spack Community

What does the Spack project look like?

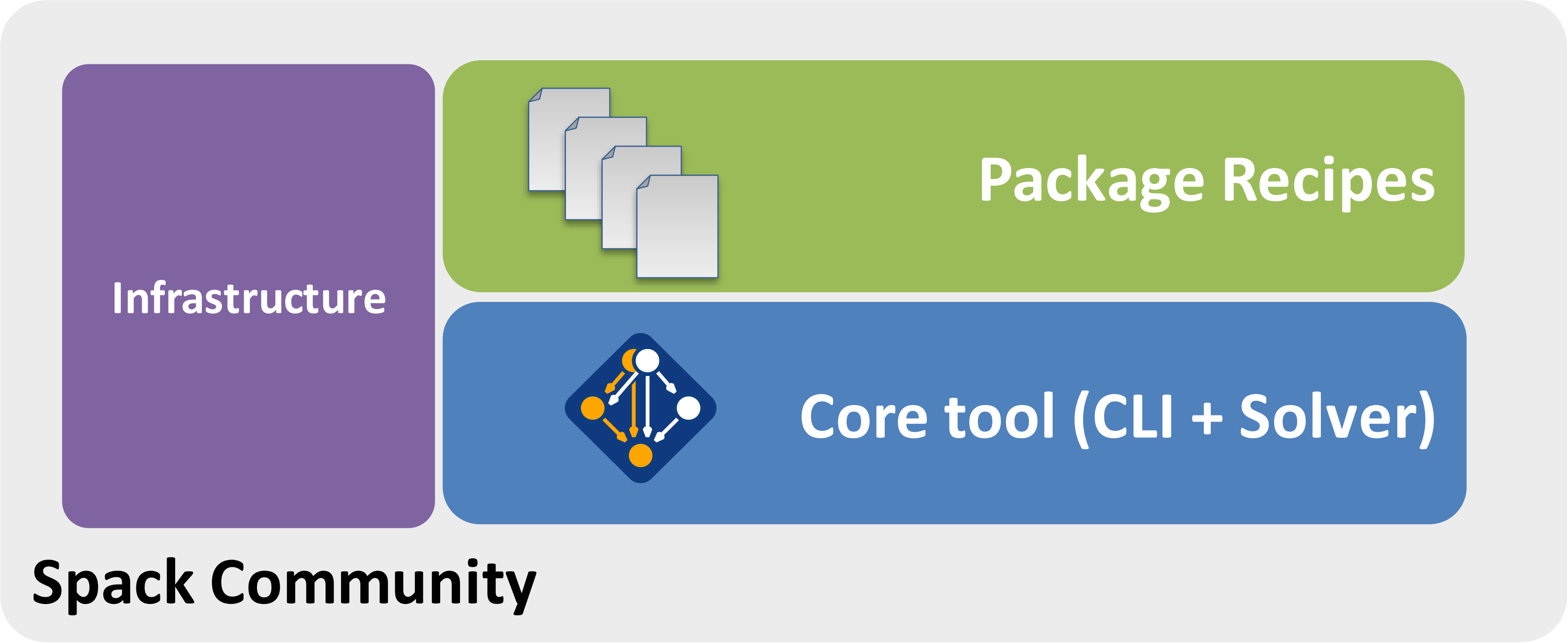


What does the Spack project look like?

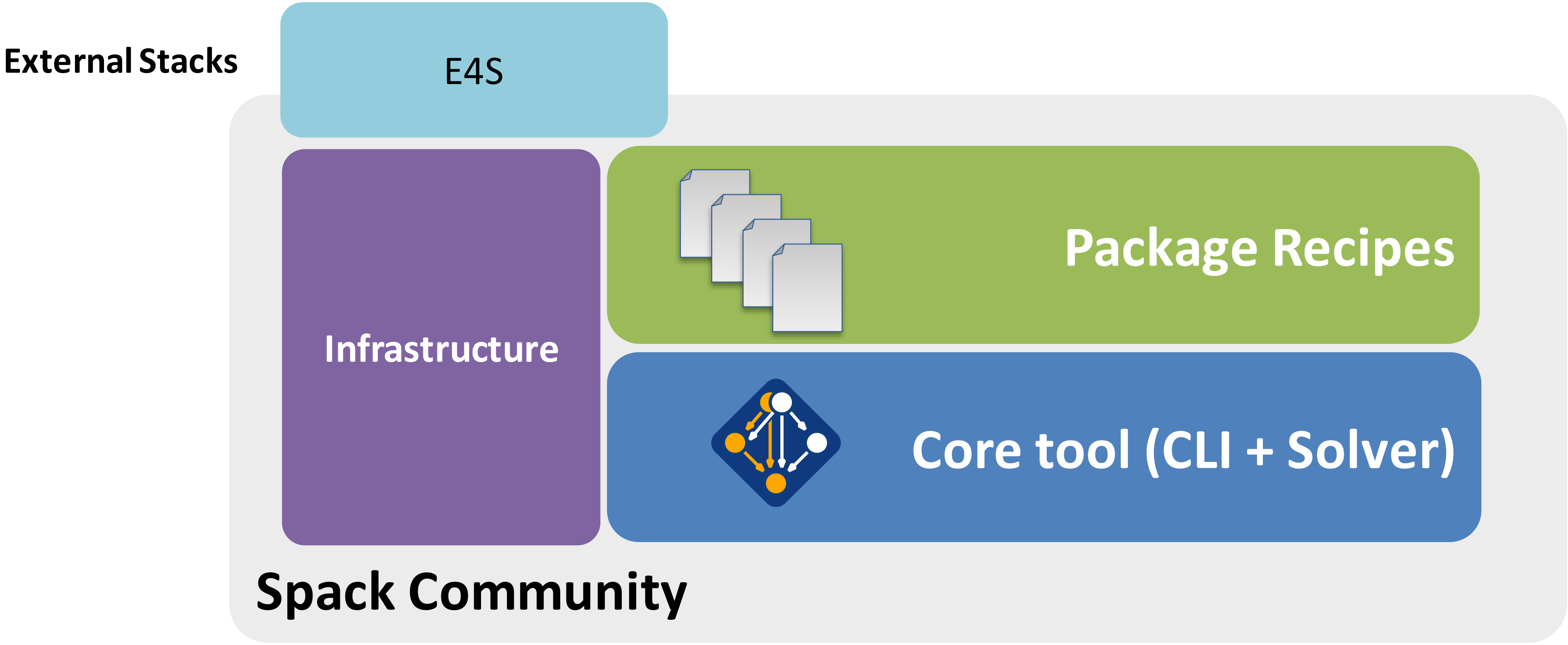


What does the Spack project look like?

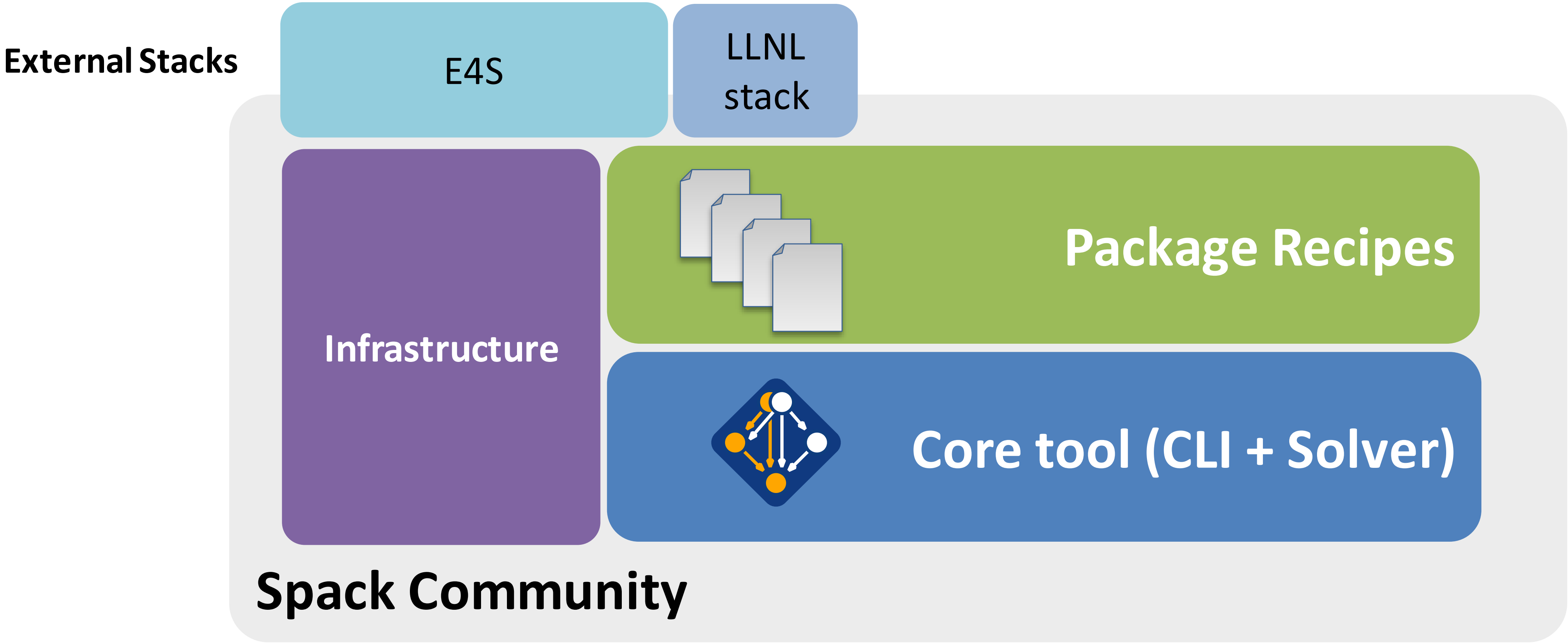
External Stacks



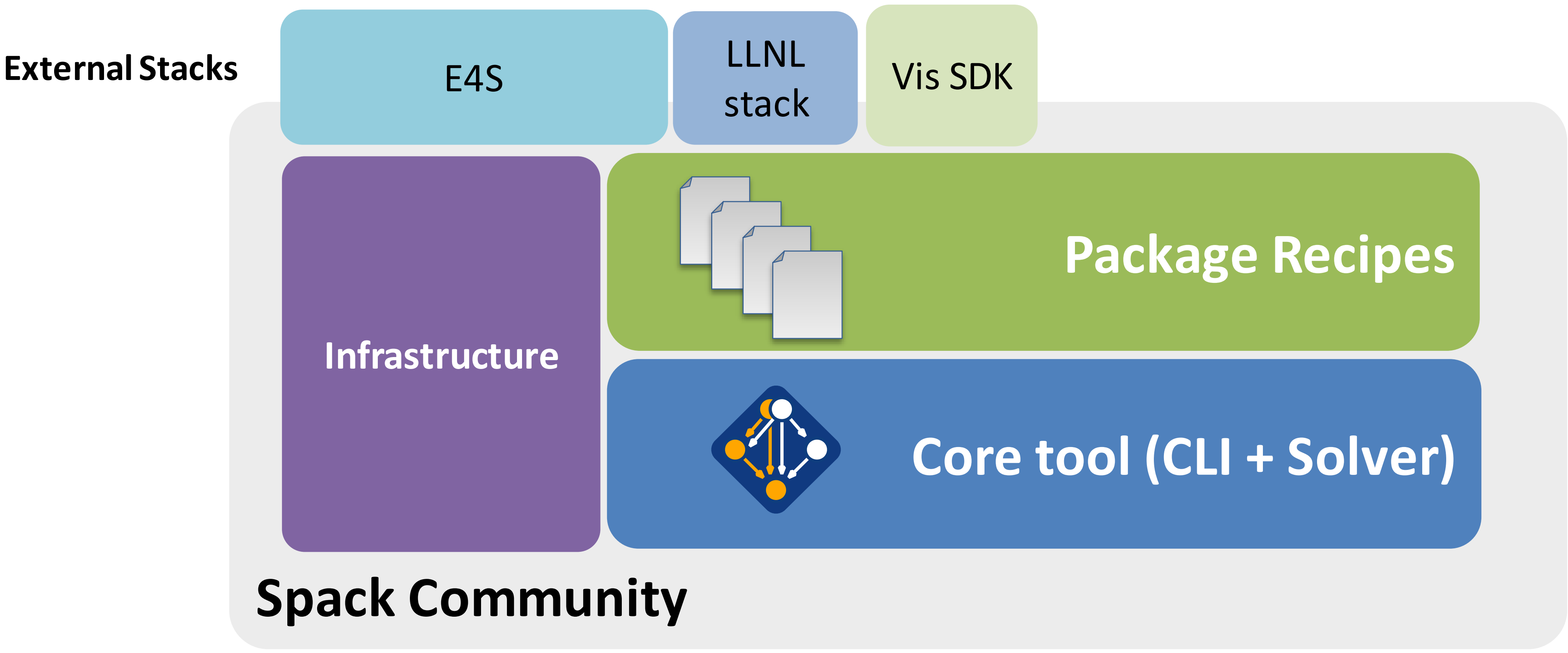
What does the Spack project look like?



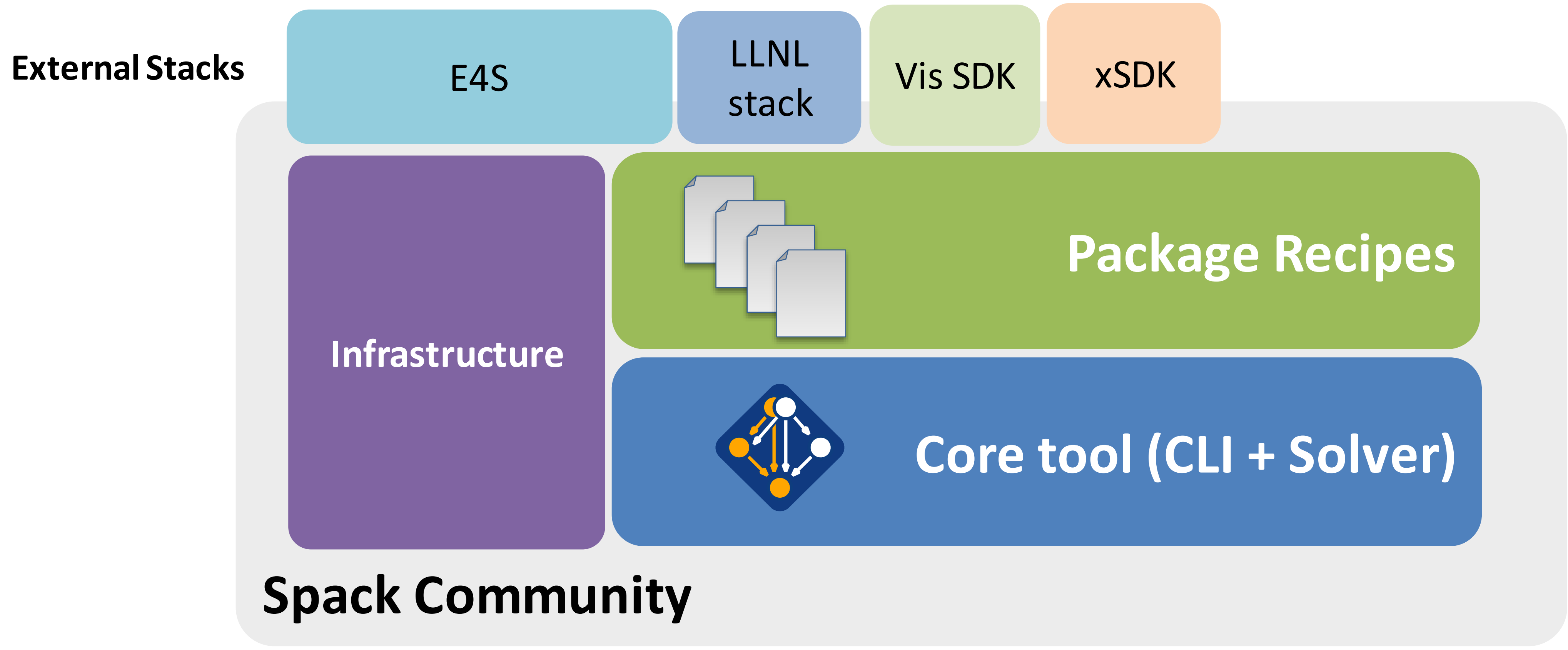
What does the Spack project look like?



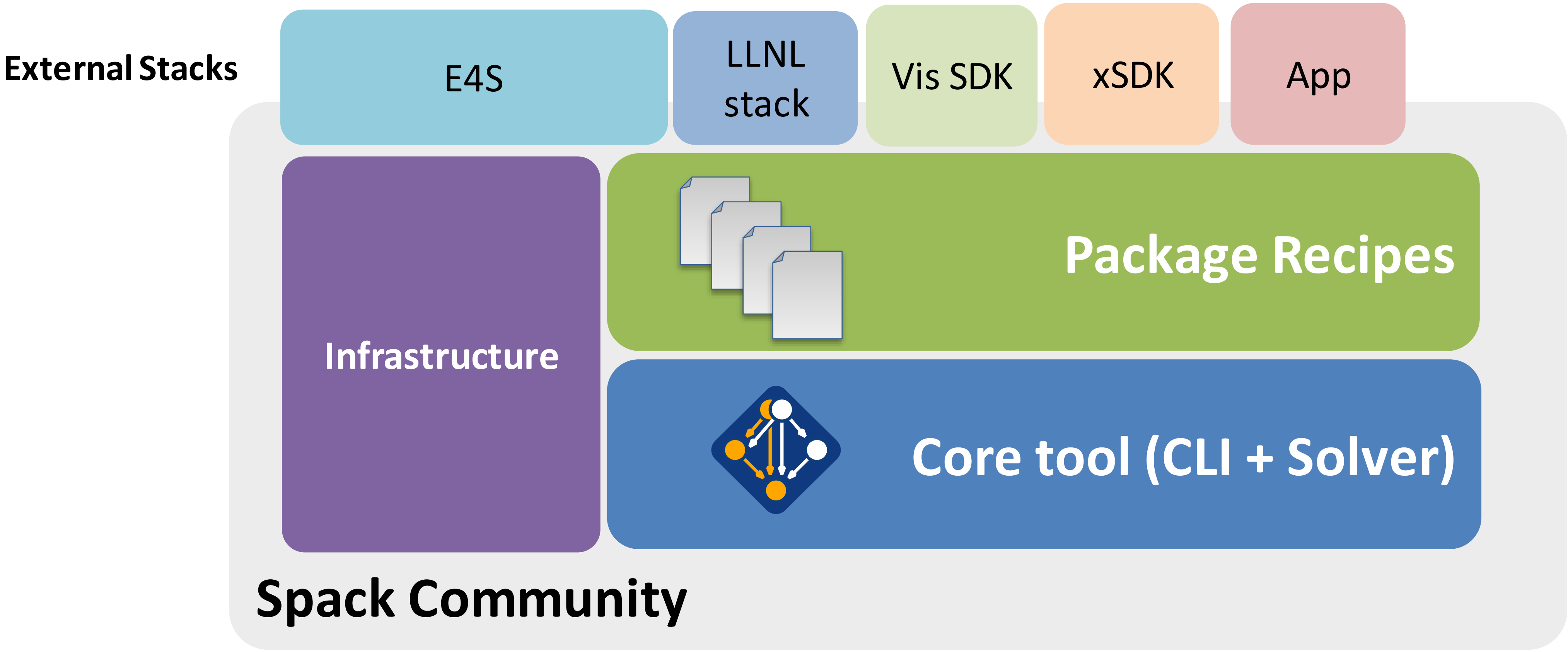
What does the Spack project look like?



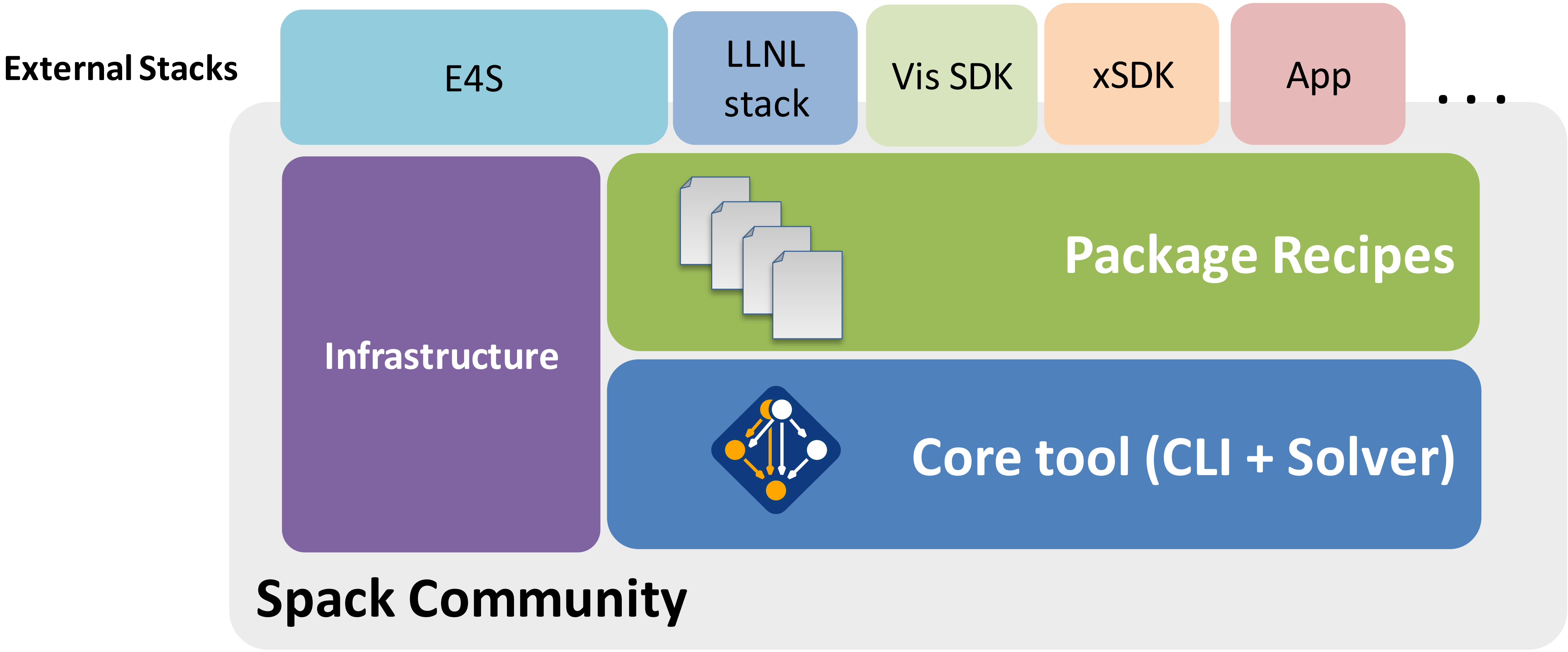
What does the Spack project look like?



What does the Spack project look like?



What does the Spack project look like?



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

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

```
$ spack install mpileaks                    unconstrained
```

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

```
$ spack install mpileaks                 unconstrained
$ spack install mpileaks@3.3                 @ custom version
```

- 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 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
```

- 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 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
```

- 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 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
```

- 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 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
```

- 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 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)

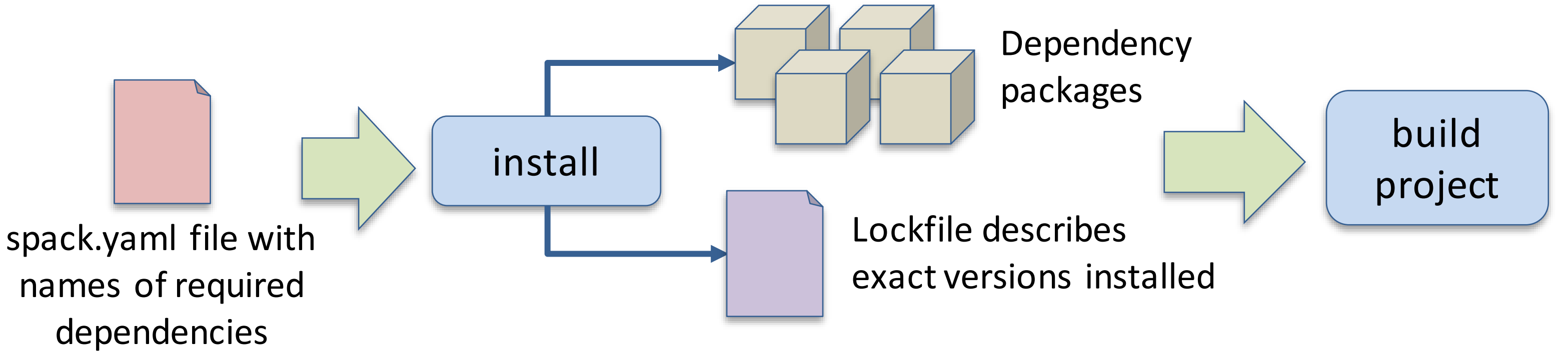
Dependencies (same spec syntax)

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!**

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
```

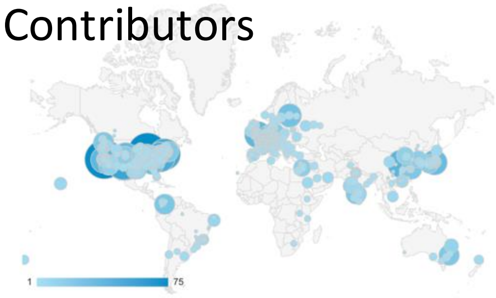
- spack.yaml describes project requirements
  - Facility stack
  - Application development environment
  - ML framework + simulations built together
  - Etc.
- spack.lock describes exactly what versions/configurations were installed, allows them to be reproduced.

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's *concretizer* leverages ASP solvers to turn abstract constraints into a fully specified, buildable graph

Contributors



- new versions
- new dependencies
- new constraints

package.py repository

concretizer

spack developers — yaml — default config packages.yaml

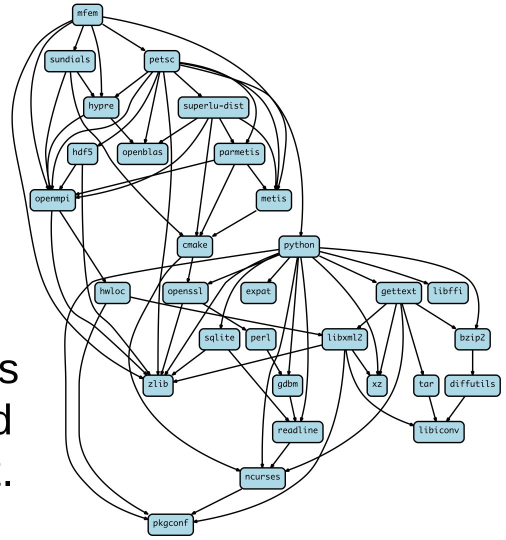
admins, users — yaml — local preferences config packages.yaml

users — yaml — local environment config spack.yaml

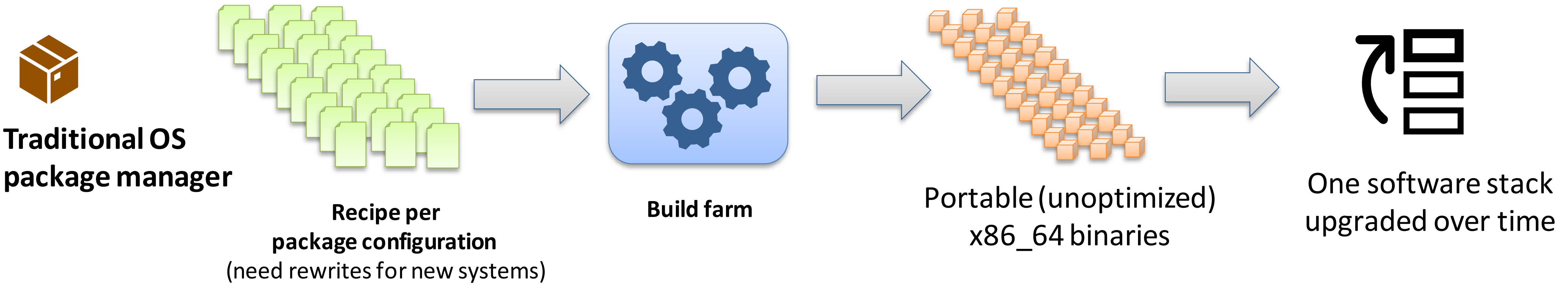
users — Command line constraints

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

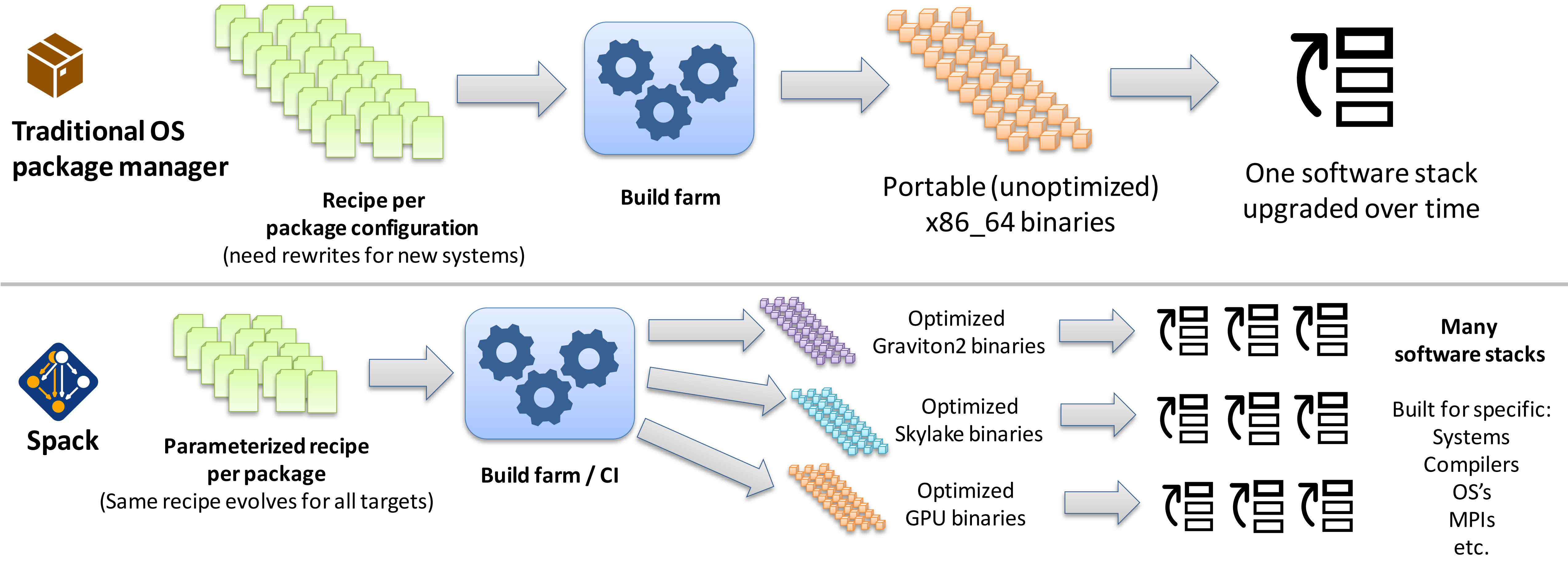
Concrete spec is fully constrained and can be built.



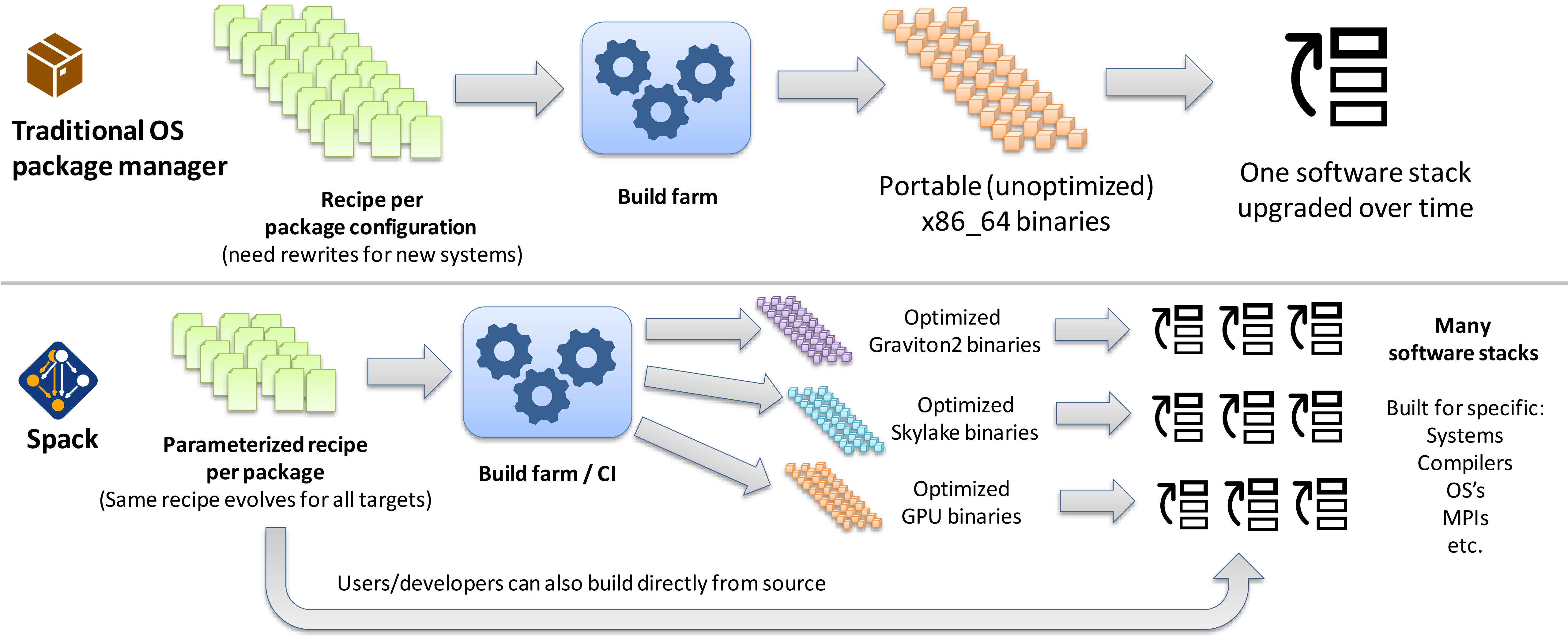
Spack's model lowers the maintenance burden of optimized software stacks



Spack's model lowers the maintenance burden of optimized software stacks



Spack's model lowers the maintenance burden of optimized software stacks

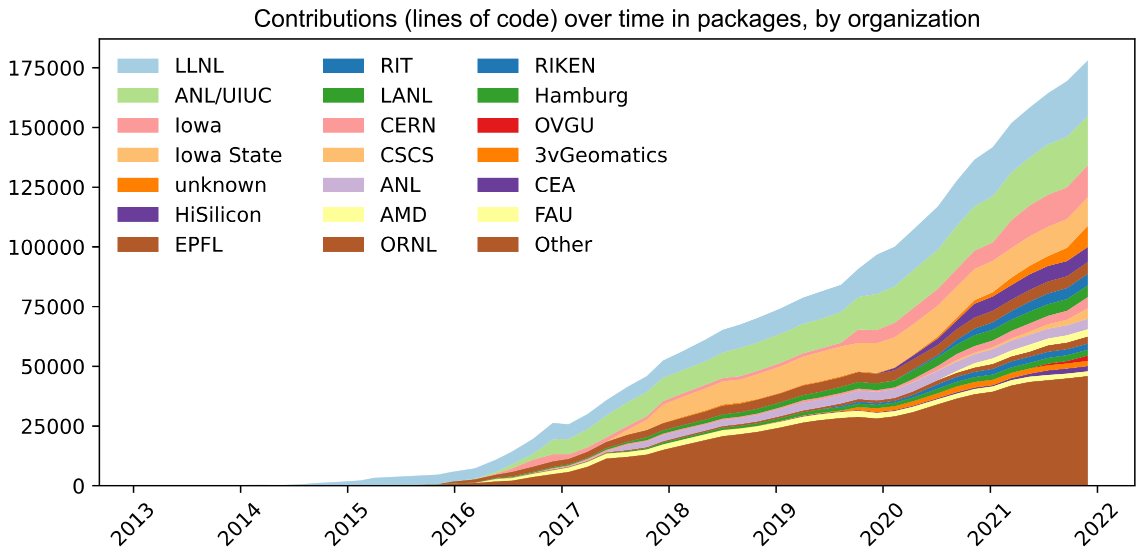


What are the sustainability challenges?

- **Community**
  - Must keep up with incoming pull requests and package updates
  - Identify strong contributors and prioritize their work (e.g., HEP, CSCS, others)
- **Infrastructure**
  - Critical for keeping the package builds working
  - Help from U. Oregon and AWS has been essential
- **Deep technical challenges**
  - Package model + semantics are constantly being improved
  - Deeper modeling of compatibility & ABI
  - Scaling solvers to ever-more-complex ecosystems
- **Maintenance**
  - Keeping core features working *and* integrating new research

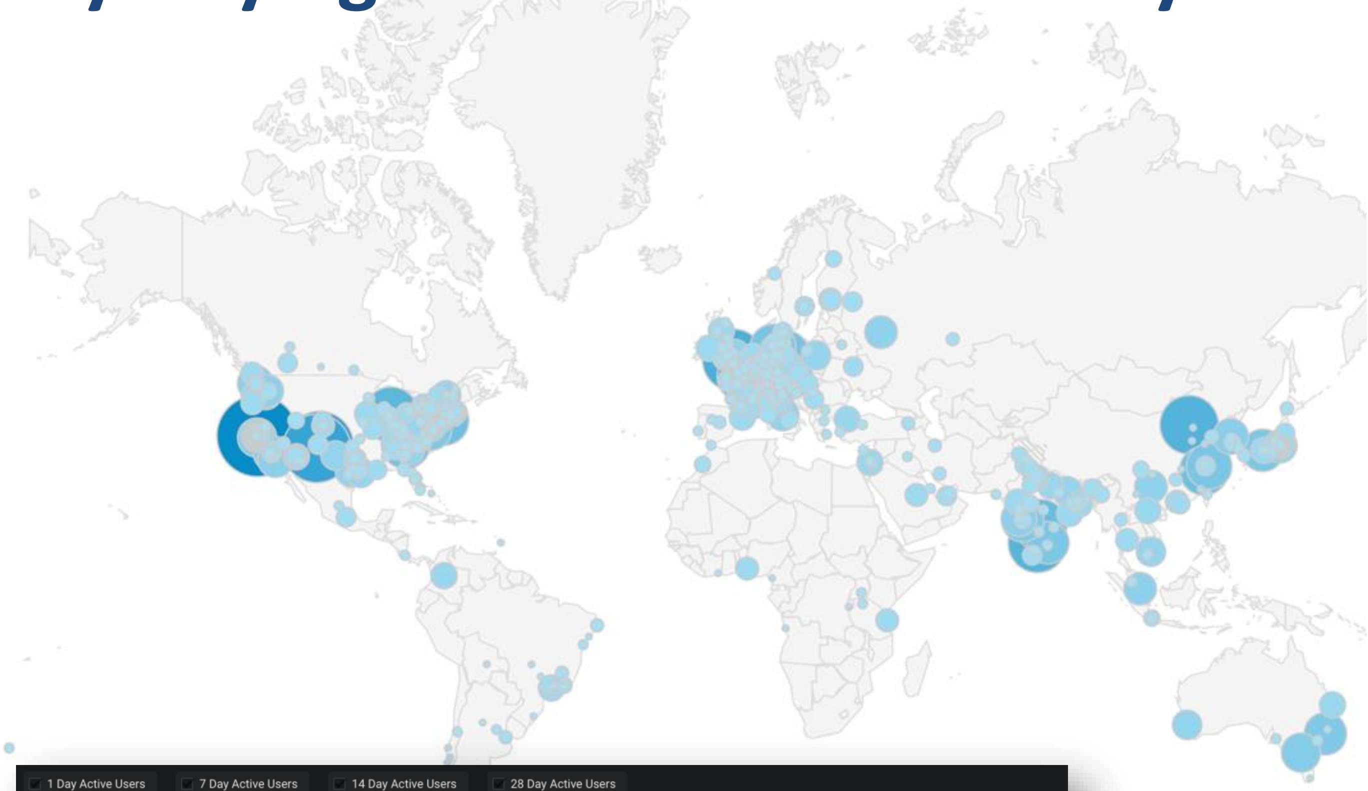
Spack help to sustain the HPC software ecosystem by relying on the efforts of many contributors

6,000+ software packages
960+ contributors

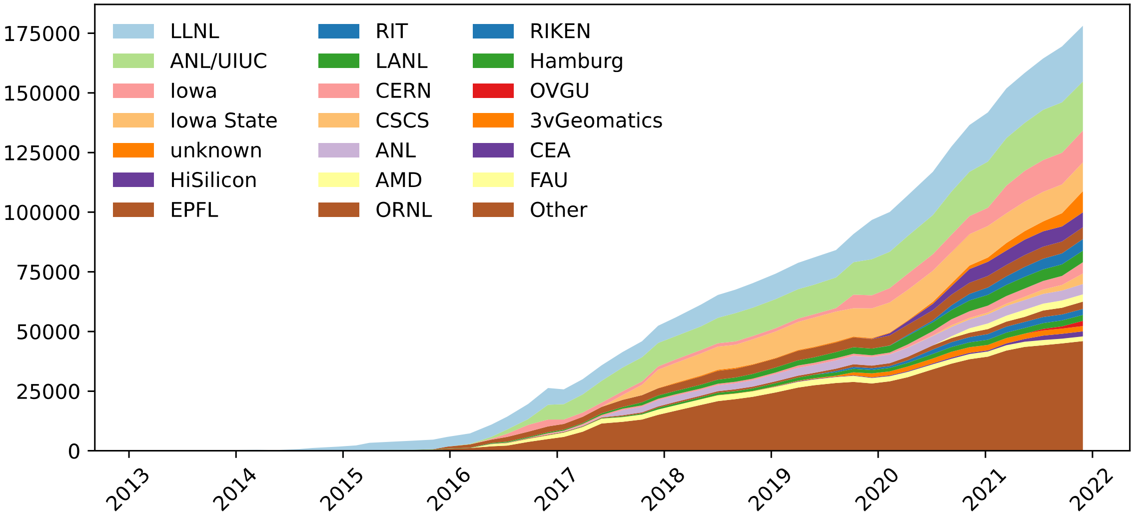


Most package contributions are ***not*** from DOE
But they help sustain the DOE ecosystem!

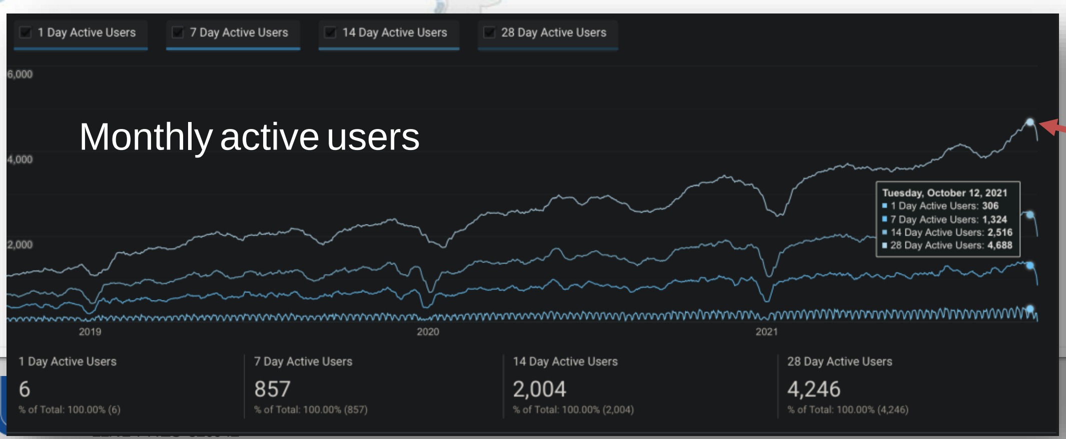
Spack help to sustain the HPC software ecosystem by relying on the efforts of many contributors



6,000+ software packages
960+ contributors



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



Over 4,600 monthly active users of documentation site in October 2021

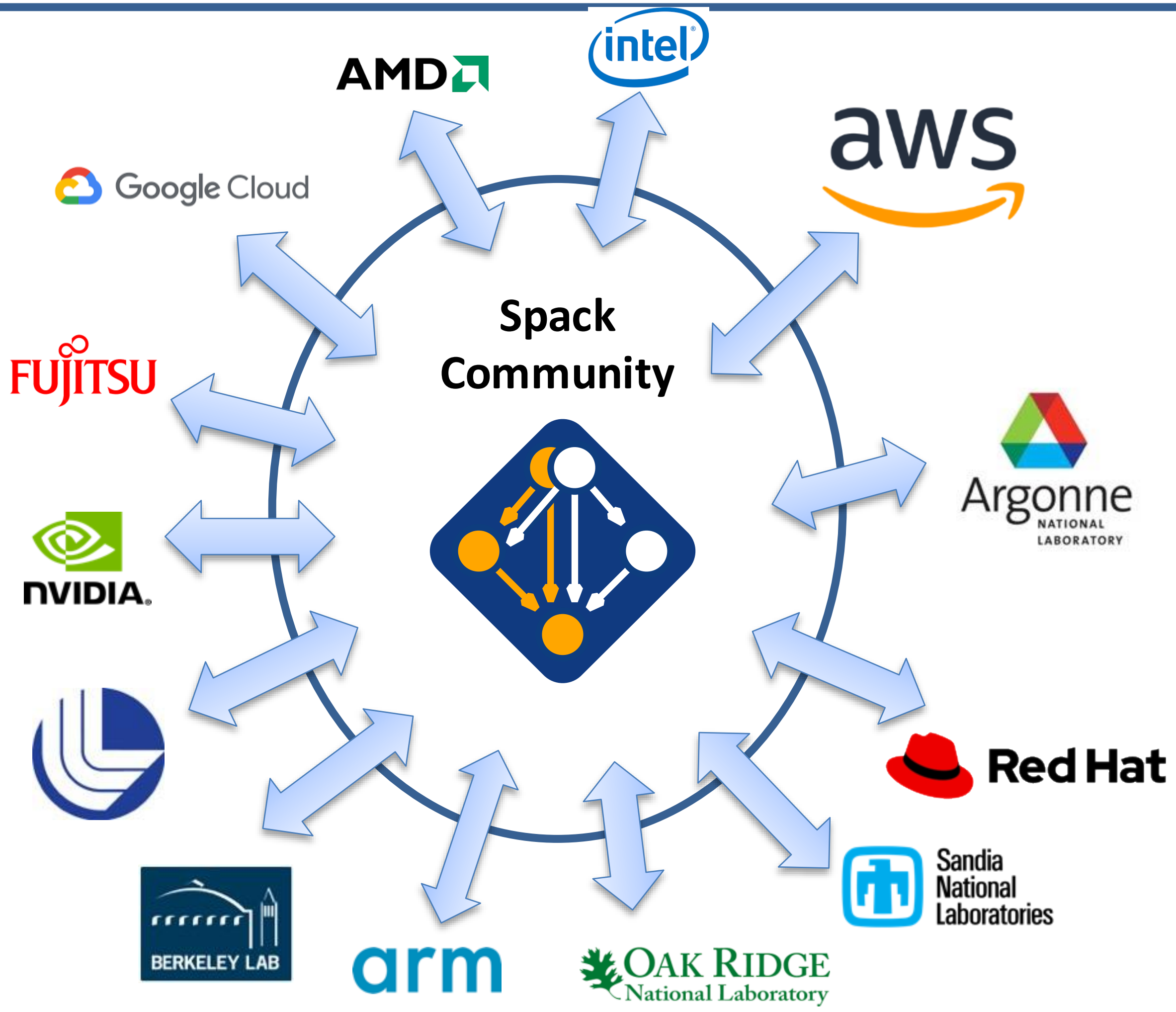
It takes many maintainers to manage all of the contributions

- **~6 core developers**
  - Core feature development + community management
  - Paid by ASC and ECP
- **Extended core team at Kitware, TechX**
  - Build farm maintenance, CI automation
  - Windows support (temporary push)
  - Subcontracted through LLNL (ASC) and ECP
- **~30 trusted package maintainers on GitHub**
  - Can merge pull requests
  - Picked from the community based on quality of contributions
- **~150 "package maintainers" (so far)**
  - Can't approve merges
  - Are notified of changes to packages of interest
  - Provide reviews on packages to help trusted maintainers

With ECP ending in 2023, ~50% of this funding goes away

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

- **Spack is not sustainable without a community**
  - Broad adoption incentivizes contributors
  - Cloud resources and automation critical to scale
- **Continue to prioritize features that get us external buy-in**
  - Niche HPC features aren't sustainable alone
  - Cloud, containerization, Windows, C++ community features are all aimed at adoption in the broader market
- **Wide adoption in HPC gets us industry attention**
  - Cloud HPC is a growth area
  - Use Spack to bridge between traditional HPC and cloud
  - Work to ensure that good Spack support is an essential feature for vendors to provide to their customers
- **Portability and generality will become increasingly important as cloud environments diversify architectures**



We are already collaborating with key vendors who can help us sustain parts of the software stack

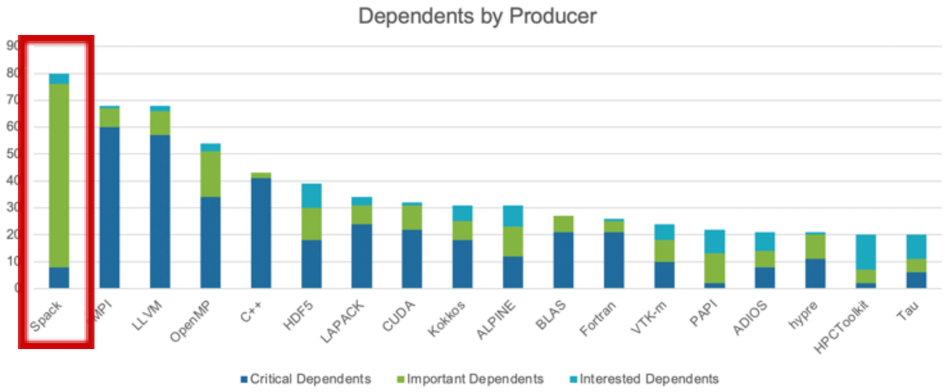
- **AWS** invests significant $$ in cloud credits for Spack build farm
  - Joint Spack tutorial in July with AWS had 125+ participants
  - Joint AWS/AHUG Spack Hackathon drew 60+ participants
- **AMD** has contributed ROCm packages and compiler support
  - 55+ PRs mostly from AMD, also others
  - ROCm, 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 dmany packages for ARM/a64fx support on Fugaku
- **ARM** + **Linaro** members contributing 100s of PRs for ARM support

Spack is critical for supporting ECP's E4S stack, which we hope will be sustained after ECP

EXASCALE COMPUTING PROJECT

https://e4s.io

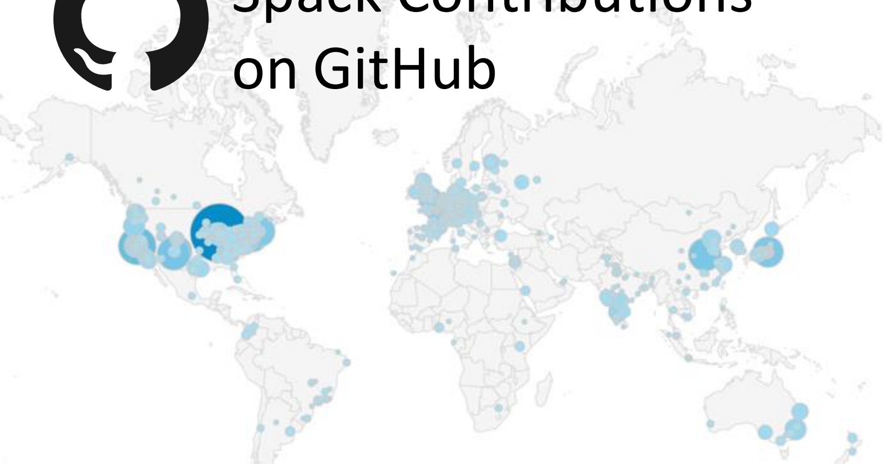
- Spack will be used to build software for the three upcoming U.S. exascale systems
- ECP has built the Extreme Scale Scientific Software Stack (E4S) with Spack – more at https://e4s.io
- Spack will be integral to upcoming ECP testing efforts.



Spack is the most depended-upon project in ECP

We have added security and scaled our public CI so that it can build entire stacks on every pull request

Spack Contributions on GitHub



We have added security and scaled our public CI so that it can build entire stacks on every pull request

Spack Contributions on GitHub

gitlab.spack.io

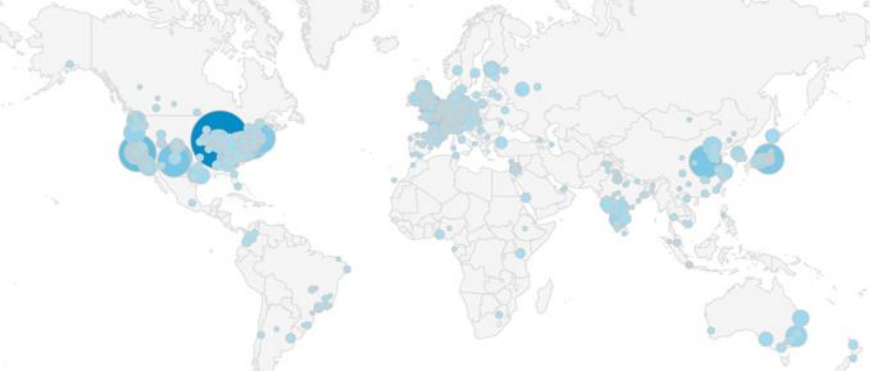

```yaml
spack:
  specs:
  - openpmd-api
  - py-libensemble^python@3.7.3
  - hypre
  - mfem
  - trilinos@12.14.1+dtk+intrepid2+shards
  - sundials
  - strumpack
  - superlu-dist
  - superlu
  - tasmanian
  - mercury
  - hdf5
  - adios2
  - dyninst
  - pdt
  - tau
  - hpctoolkit
  - adios
  - darshan-runtime
  - 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
```


spack.yaml configurations (E4S, SDKs, others)

We have added security and scaled our public CI so that it can build entire stacks on every pull request

Spack Contributions on GitHub



gitlab.spack.io

```yaml
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
```

spack.yaml
configurations
(E4S, SDKs, others)

spack ci

We have added security and scaled our public CI so that it can build entire stacks on every pull request

Spack Contributions on GitHub

gitlab.spack.io

```yaml
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
```

spack.yaml configurations (E4S, SDKs, others)

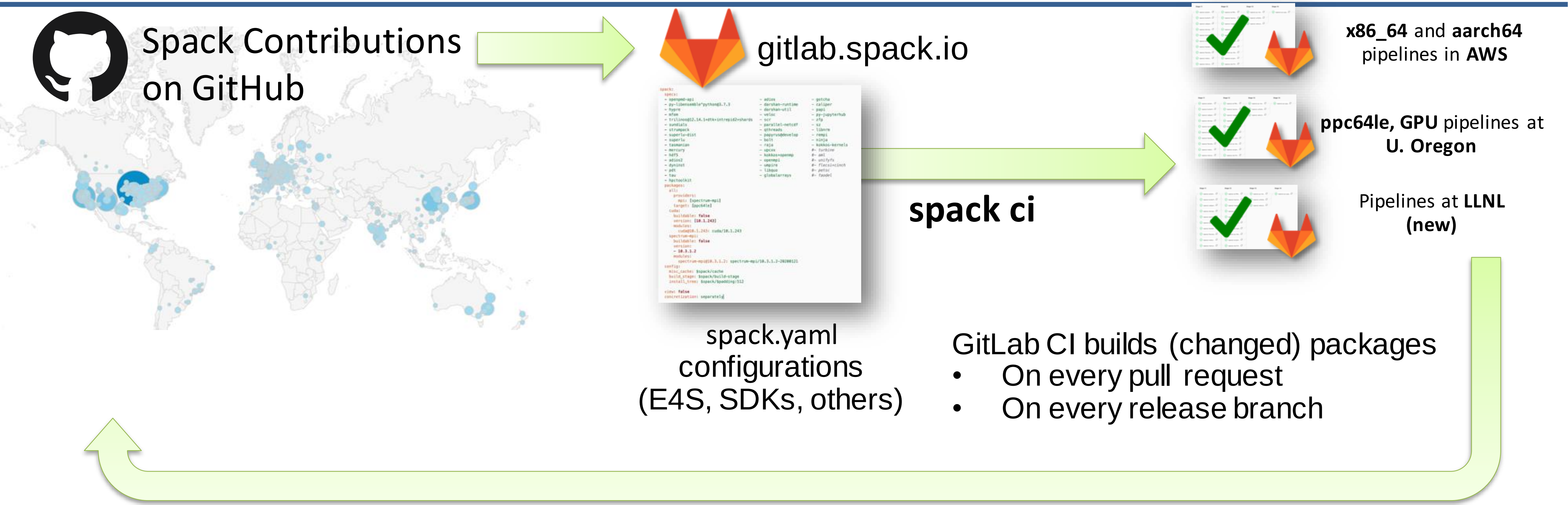
spack ci

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

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

Pipelines at **LLNL** **(new)**

We have added security and scaled our public CI so that it can build entire stacks on every pull request



- **New security model supports untrusted contributions from forks**
  - Sandboxed build caches for test builds
  - Authoritative builds on mainline only after approved merge

We have added security and scaled our public CI so that it can build entire stacks on every pull request

Spack Contributions on GitHub

gitlab.spack.io

spack.yaml configurations (E4S, SDKs, others)

spack ci

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

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

Pipelines at **LLNL** **(new)**

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

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

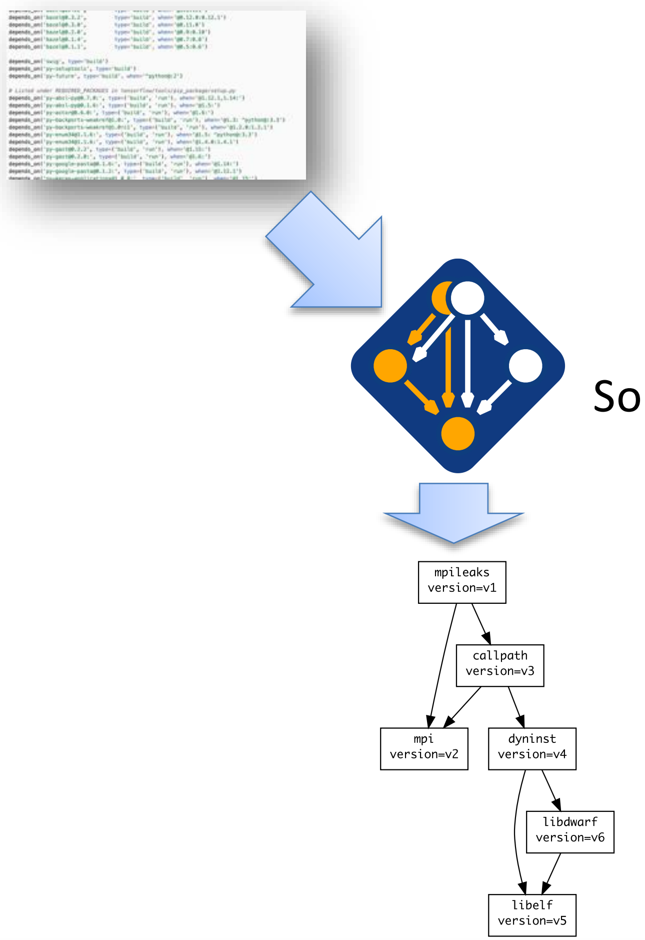
- **New security model supports untrusted contributions from forks**
  - Sandboxed build caches for test builds
  - Authoritative builds on mainline only after approved merge

BUILD is a 3-year strategic initiative, aimed at reducing human maintenance burden

- Basic premise: humans can't generate all the compatibility constraints
  - Version ranges, conflicts, in Spack packages not precise
  - rely on maintainers to get right.

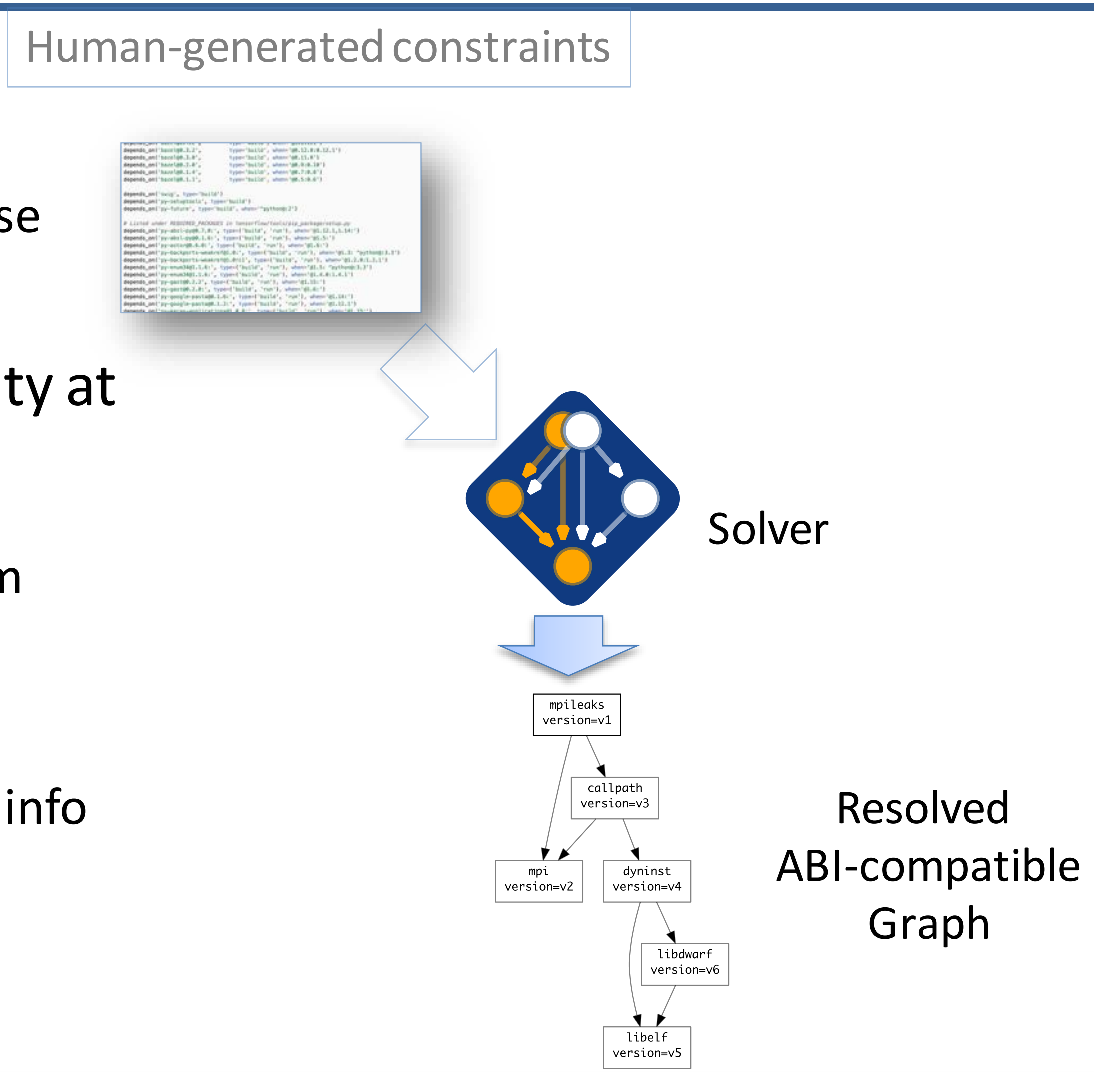
- BUILD aims to understand software compatibility at the binary level
  - Develop ABI compatibility models
  - Enable *automatic* and ABI-compatible reuse of system binaries, foreign binary packages

- **WIP: add ABI constraints to the solver**
  - Don't just check with coarse compiler/target/version info
  - Guarantee that the executable will:
    - Link correctly
    - Run w/o symbol and certain type errors



BUILD is a 3-year strategic initiative, aimed at reducing human maintenance burden

- Basic premise: humans can’t generate all the compatibility constraints
  - Version ranges, conflicts, in Spack packages not precise
  - rely on maintainers to get right.
- BUILD aims to understand software compatibility at the binary level
  - Develop ABI compatibility models
  - Enable *automatic* and ABI-compatible reuse of system binaries, foreign binary packages
- **WIP: add ABI constraints to the solver**
  - Don't just check with coarse compiler/target/version info
  - Guarantee that the executable will:
    - Link correctly
    - Run w/o symbol and certain type errors

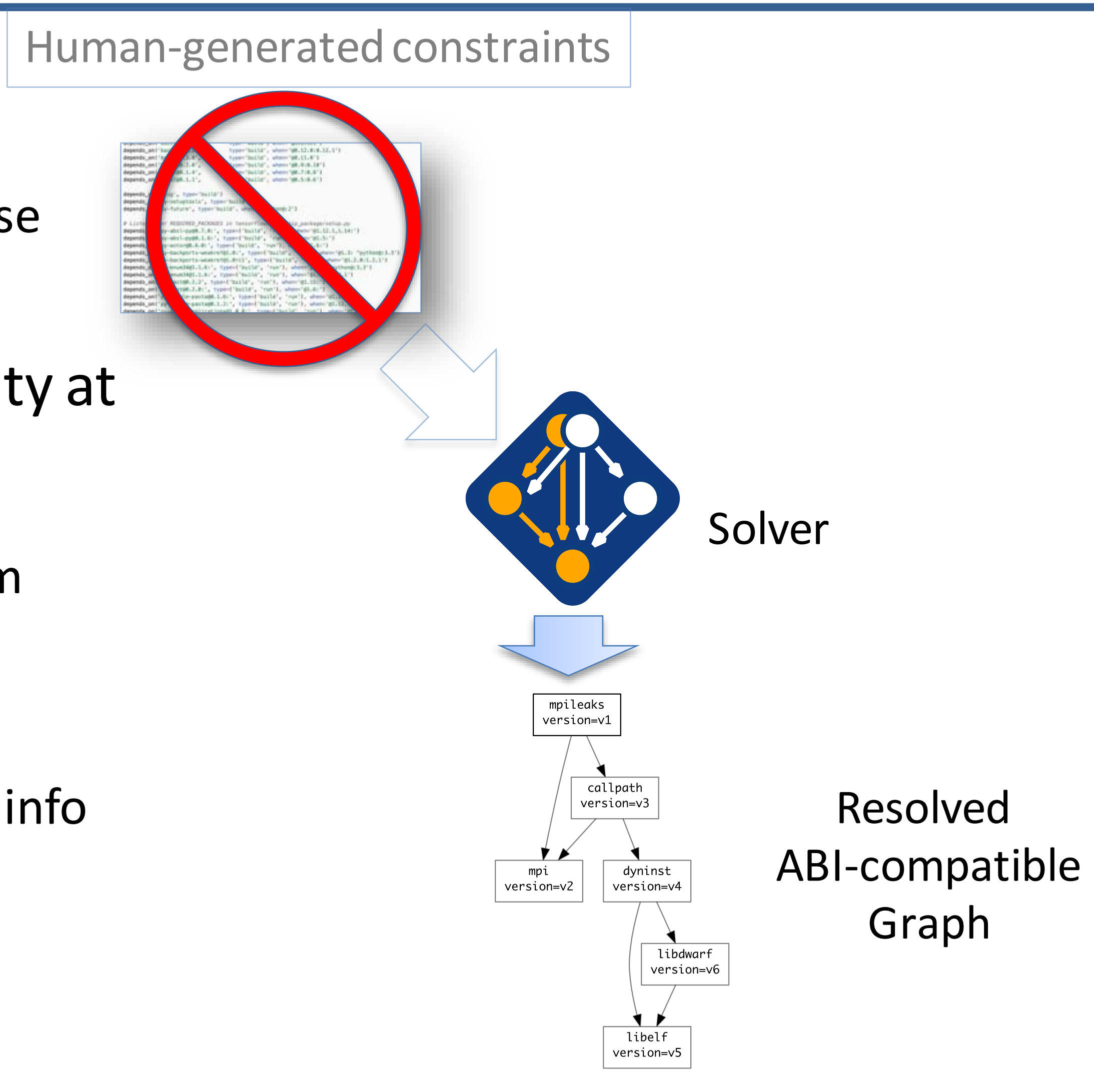


BUILD is a 3-year strategic initiative, aimed at reducing human maintenance burden

- Basic premise: humans can't generate all the compatibility constraints
  - Version ranges, conflicts, in Spack packages not precise
  - rely on maintainers to get right.

- BUILD aims to understand software compatibility at the binary level
  - Develop ABI compatibility models
  - Enable *automatic* and ABI-compatible reuse of system binaries, foreign binary packages

- **WIP: add ABI constraints to the solver**
  - Don't just check with coarse compiler/target/version info
  - Guarantee that the executable will:
    - Link correctly
    - Run w/o symbol and certain type errors

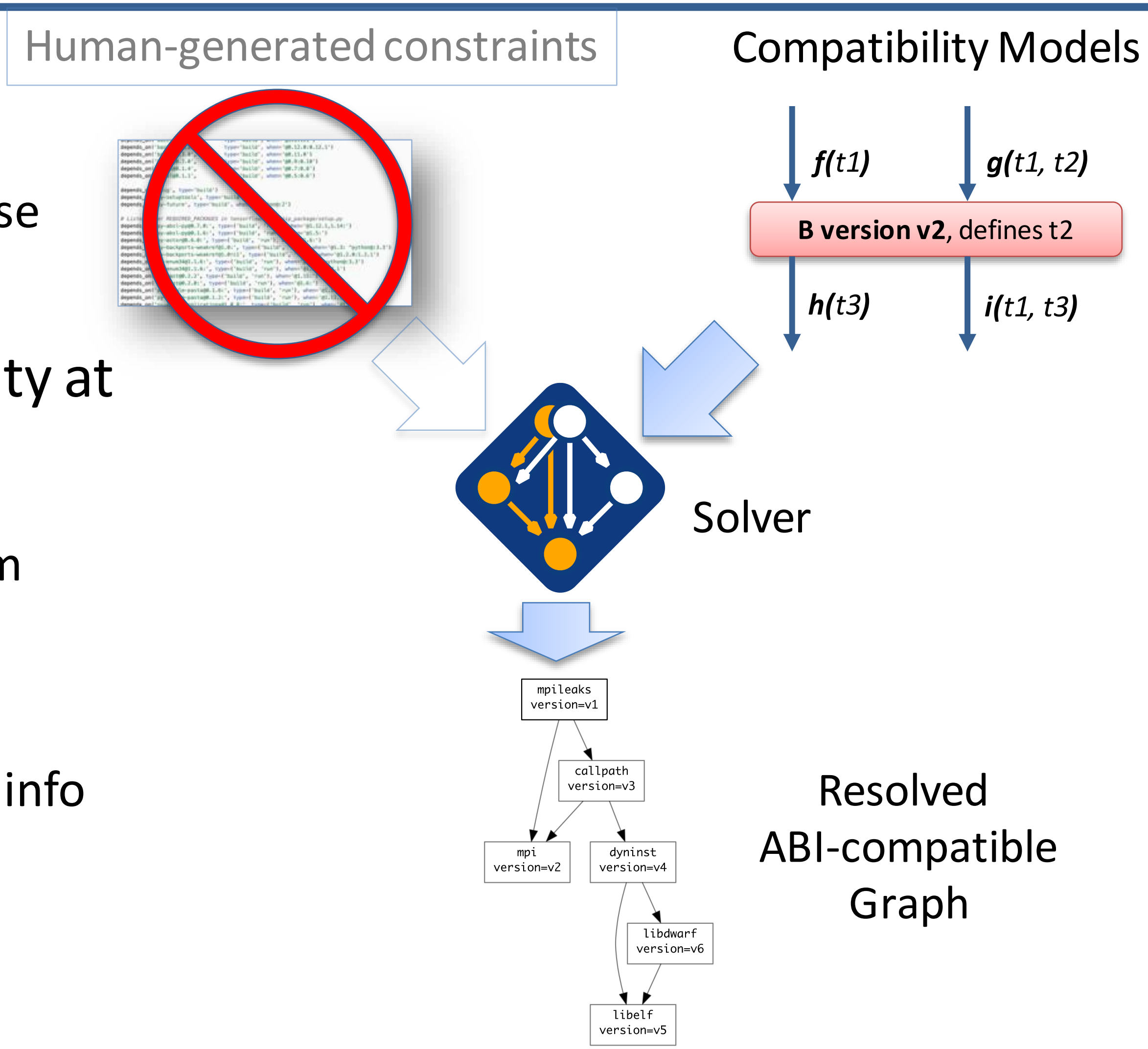


BUILD is a 3-year strategic initiative, aimed at reducing human maintenance burden

- Basic premise: humans can't generate all the compatibility constraints
  - — Version ranges, conflicts, in Spack packages not precise
  - — rely on maintainers to get right.

- BUILD aims to understand software compatibility at the binary level
  - — Develop ABI compatibility models
  - — Enable *automatic* and ABI-compatible reuse of system binaries, foreign binary packages

- **WIP: add ABI constraints to the solver**
  - — Don't just check with coarse compiler/target/version info
  - — Guarantee that the executable will:
    - Link correctly
    - Run w/o symbol and certain type errors



Key Spack priorities for future sustainability

1. **Preserve Spack core team and feature development after ECP**
   - Seek out sustainability funding
   - Ensure we can manage the growing community
   - Look at alternate governance models (foundations?)
     - Would this make it easier to scale?

2. **Increase build automation to match rate of contribution**
   - More CI, more binaries, more platforms

3. **Generalize Spack's model to make the software stack as portable as possible**
   - ABI research on BUILD
   - Compiler dependency model
   - Better modeling of runtime libraries for GPUs, OpenMP
   - Improved solver constraints

4. **Continue to grow collaborator base with key features**
   - Windows support
   - More developer features
   - Continuous integration
   - Public binary cache for faster installations

Lawrence Livermore National Laboratory

Spack DSL allows *declarative* specification of complex constraints

CudaPackage: superclass for packages that use CUDA

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

    with when("+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 = cuda is on
~cuda = cuda is off

cuda_arch and dependency on cuda are only present if cuda is enabled

Map compute capability to cuda version

Compiler support determined by architecture and CUDA version

DSL is designed to model software in *all* configurations (not just one)

In Spack, *concretization* converts an abstract spec to a real (concrete) installation

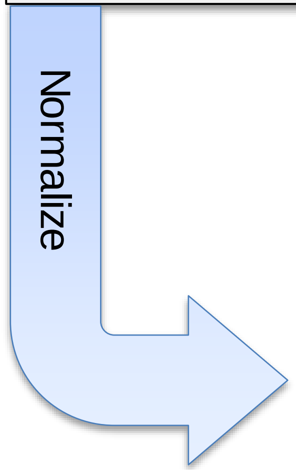
```
mpileaks ^callpath@1.0+debug ^libelf@0.8.11
```

User input: *abstract* spec with some constraints

In Spack, *concretization* converts an abstract spec to a real (concrete) installation

mpileaks ^callpath@1.0+debug ^libelf@0.8.11

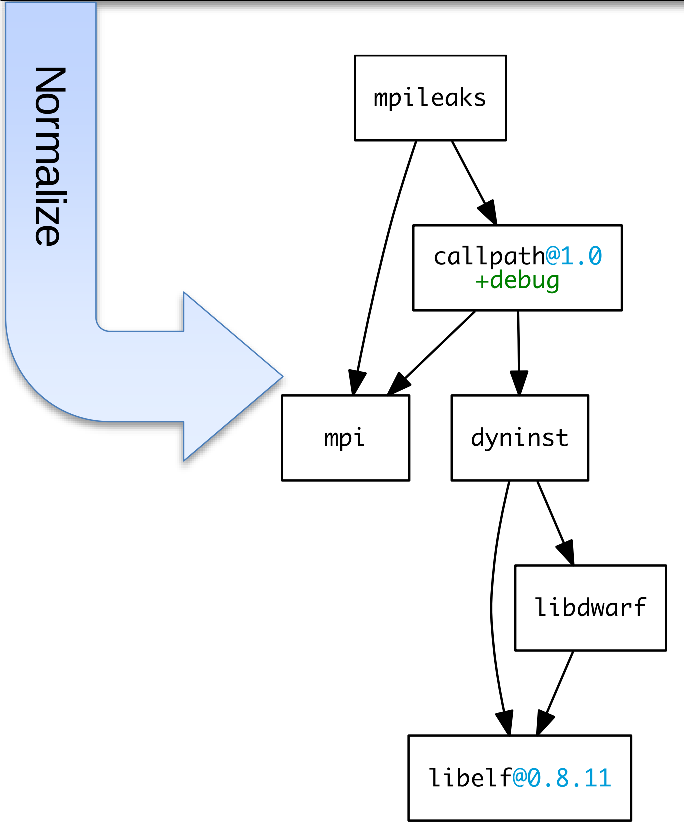
User input: *abstract* spec with some constraints



In Spack, *concretization* converts an abstract spec to a real (concrete) installation

mpileaks ^callpath@1.0+debug ^libelf@0.8.11

User input: *abstract* spec with some constraints

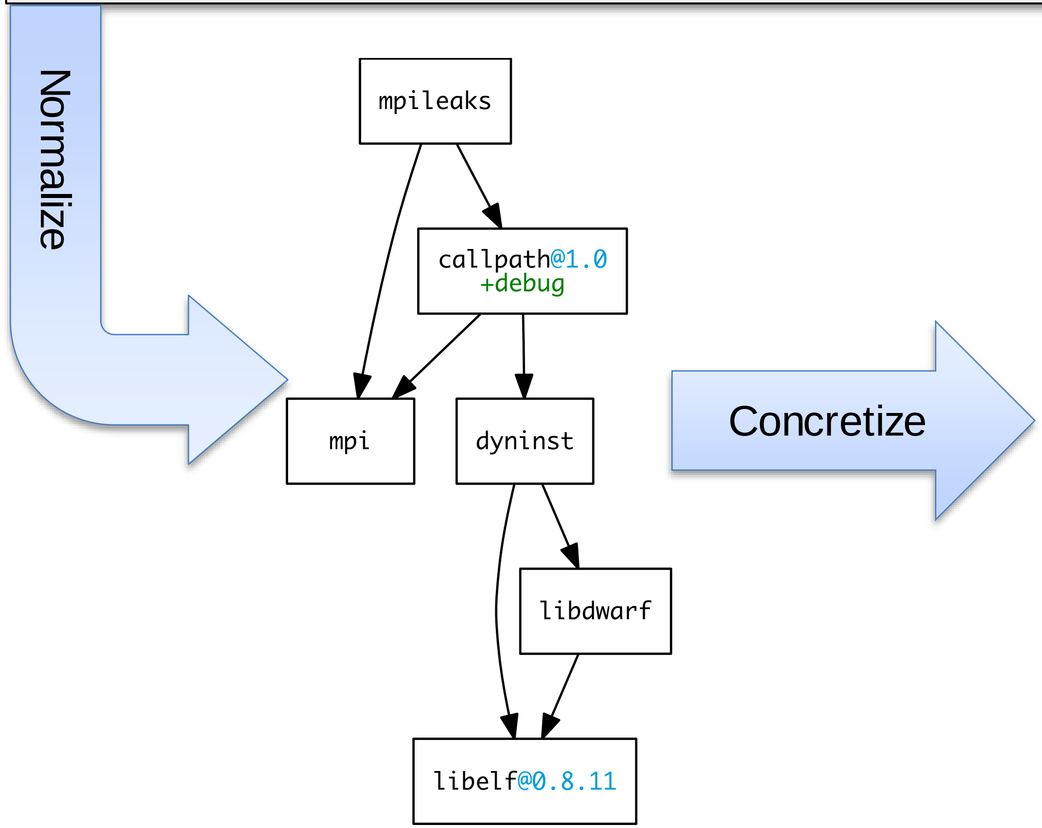


Abstract, normalized spec
with dependencies known *a priori*.

In Spack, *concretization* converts an abstract spec to a real (concrete) installation

mpileaks ^callpath@1.0+debug ^libelf@0.8.11

User input: *abstract* spec with some constraints

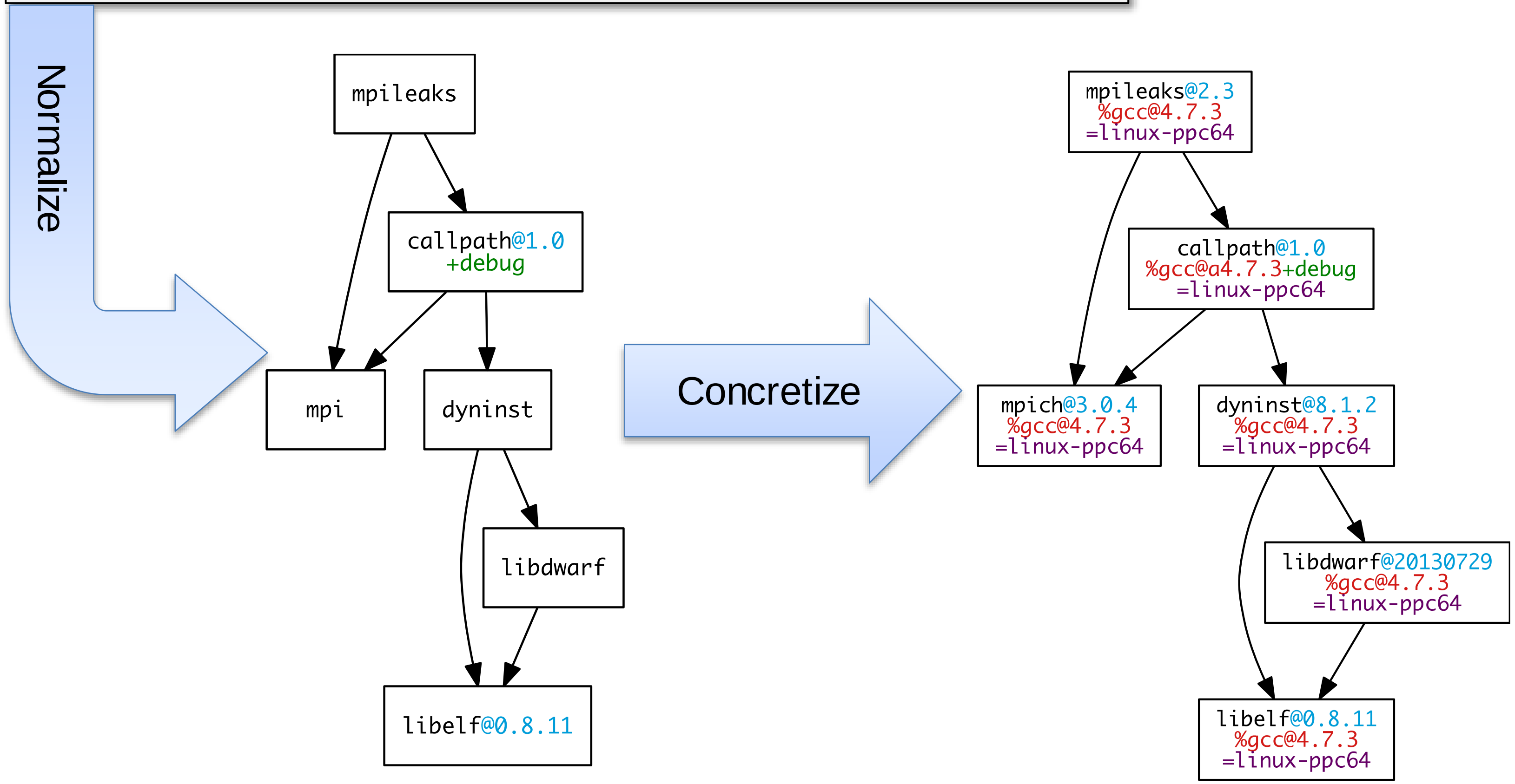


Abstract, normalized spec
with dependencies known *a priori*.

In Spack, *concretization* converts an abstract spec to a real (concrete) installation

mpileaks ^callpath@1.0+debug ^libelf@0.8.11

User input: *abstract* spec with some constraints



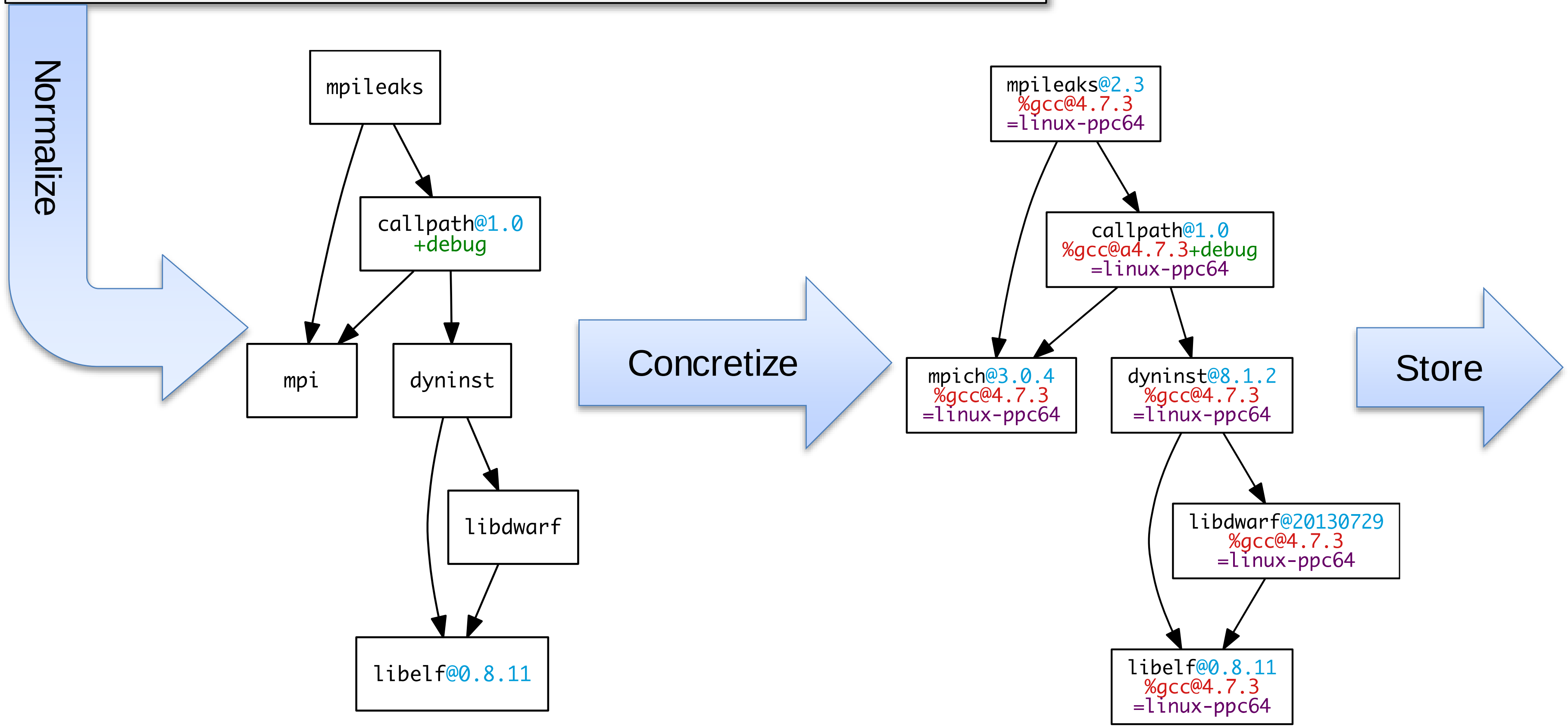
Abstract, normalized spec with dependencies known *a priori*.

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

In Spack, *concretization* converts an abstract spec to a real (concrete) installation

mpileaks ^callpath@1.0+debug ^libelf@0.8.11

User input: *abstract* spec with some constraints



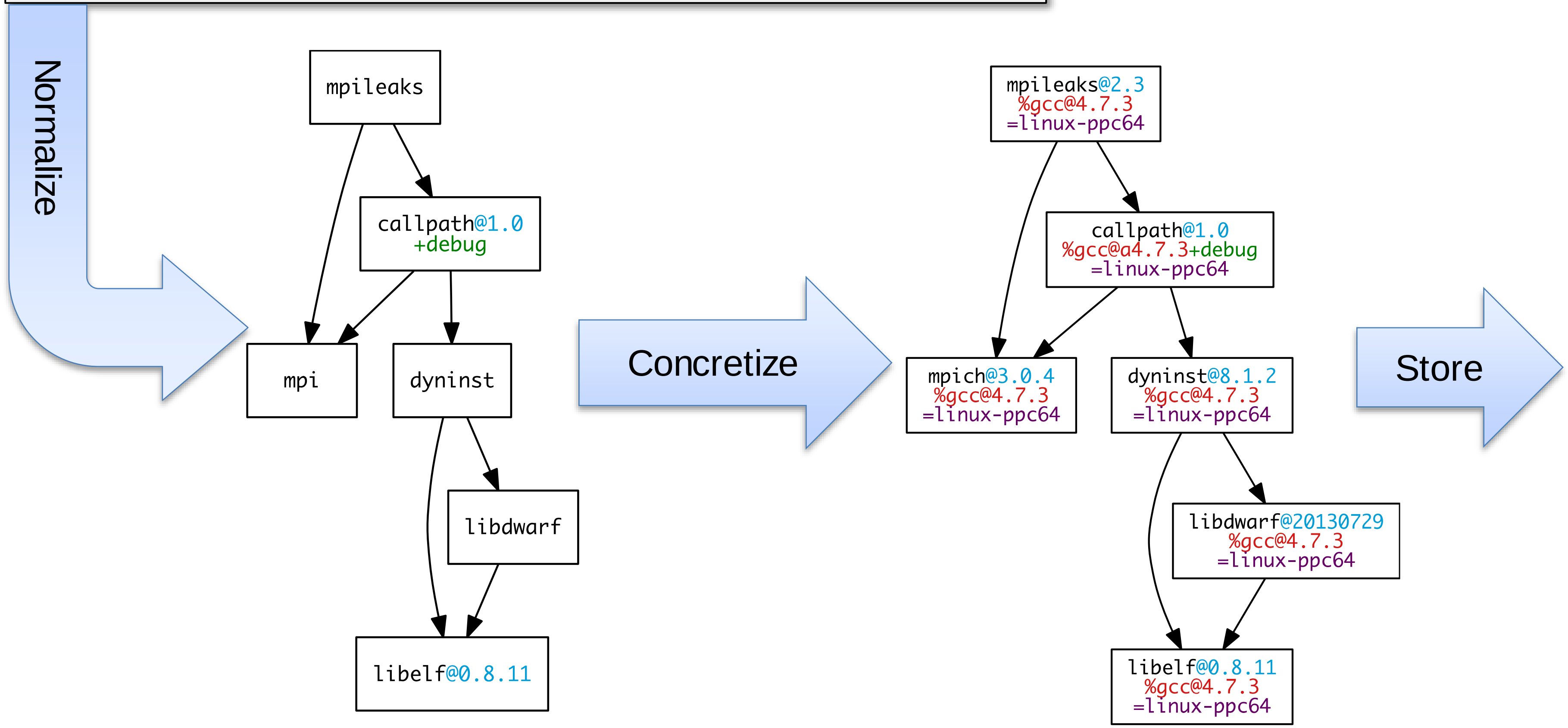
Abstract, normalized spec
with dependencies known *a priori*.

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

In Spack, *concretization* converts an abstract spec to a real (concrete) installation

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 dependencies known *a priori*.

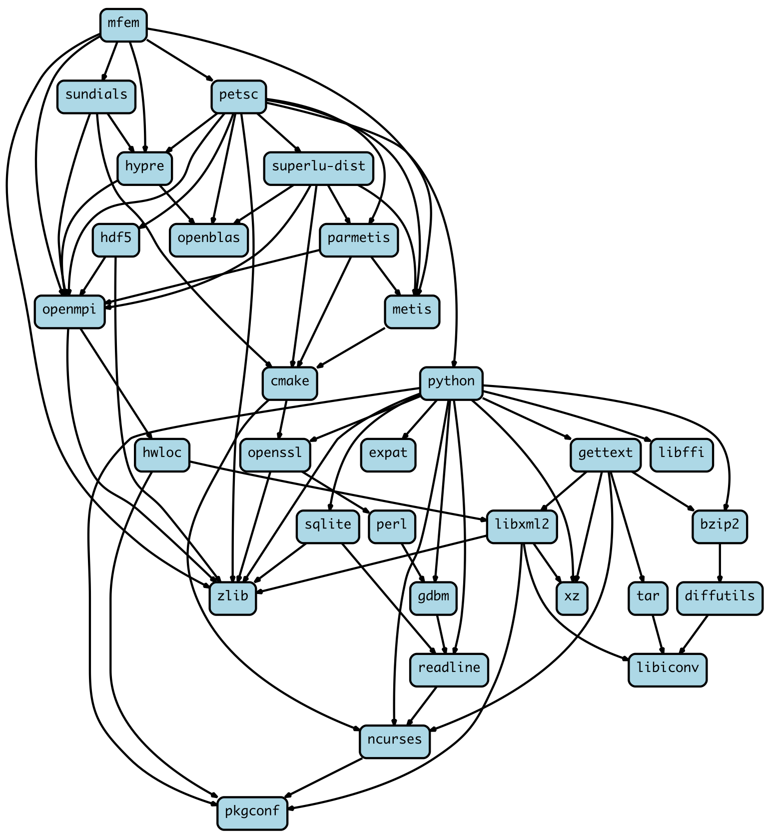
Concrete spec is fully constrained and can be passed to install.

Detailed provenance is stored with the installed package

Package solving is *combinatorial search* with *constraints* and *optimization*

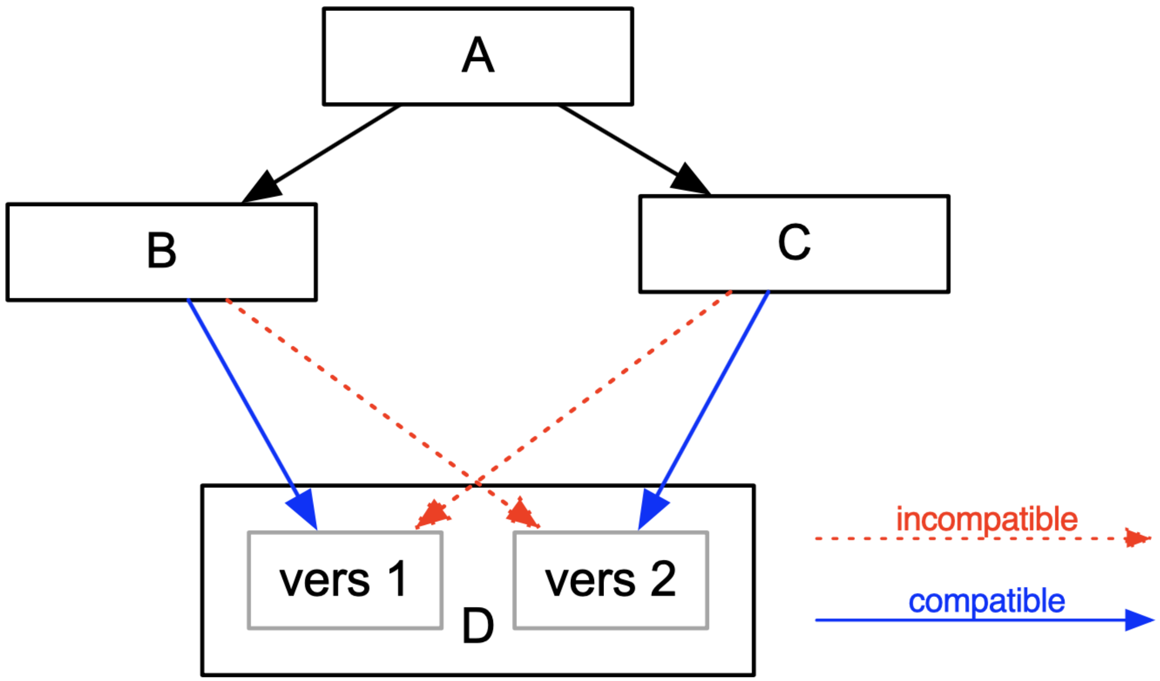
This problem is NP-hard!

- Search over a solution space:
  - Possible dependency graphs (nodes, edges)
  - Assignment of node and edge attributes
    - Version
    - Dependency, dependency type
    - Compiler, compiler version
    - Target
    - Compiler, compiler version
- Subject to validity constraints:
  - Version requirements
  - Target/compiler compatibility
  - Virtual providers
- Optimization picks "best" among valid solutions:
  - Most recent versions
  - Preferred variant values
  - Preferred compilers that support best targets (e.g., AVX-512)
  - Minimize number of builds



Dependency solving is NP-complete

- Most language runtimes only support one package version in memory at a time
  - Must pick **exactly one** version of each package in the graph
- **Impossible to choose a version of D that satisfies both B and C**
  - Must back out and choose new B or C versions
  - Repeat until we find ones with compatible constraints on D



https://research.swtch.com/version-sat

This is just with versions.

"Any opinions, findings, conclusions or recommendations expressed in this material are those of the author(s) and do not necessarily reflect the views of the Networking and Information Technology Research and Development Program."

The Networking and Information Technology Research and Development (NITRD) Program

Mailing Address: NCO/NITRD, 2415 Eisenhower Avenue, Alexandria, VA 22314

Physical Address: 490 L'Enfant Plaza SW, Suite 8001, Washington, DC 20024, USA Tel: 202-459-9674, Fax: 202-459-9673, Email: nco@nitrd.gov, Website: https://www.nitrd.gov