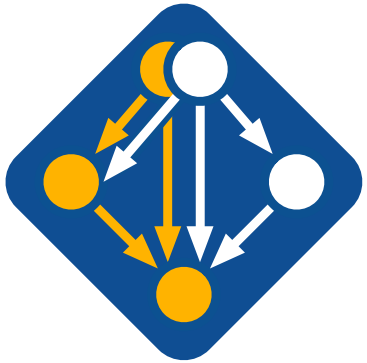


Answer Set Solving in Spack

Tackling combinatorial software complexity head-on

Scalable Tools Workshop

June 21, 2022

Todd Gamblin
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?

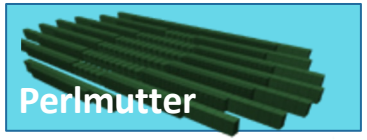
- Supercomputing PACKage manager
- Language-agnostic
  - Focused originally on build from source
  - Now focused on both source and binary
- Allows arbitrarily many installs of *any* package
- Inspired by Nix + Homebrew
  - More flexible package model than either
  - Solver, `spack.yaml` manifests, lockfiles, envs
- Thousands of users worldwide
  - 6,400 packages so far
  - 1,000+ contributors

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

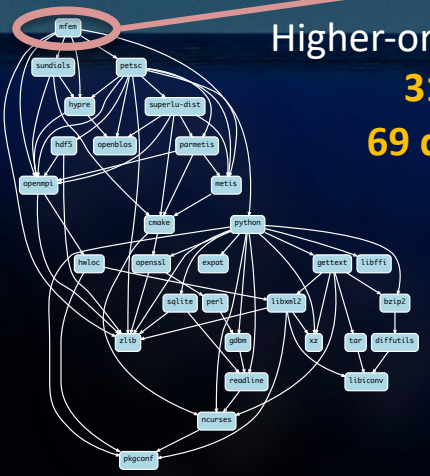
Argonne National Lab
Intel Xeon / Xe

Oak Ridge National Lab
AMD Zen / MI200 GPU

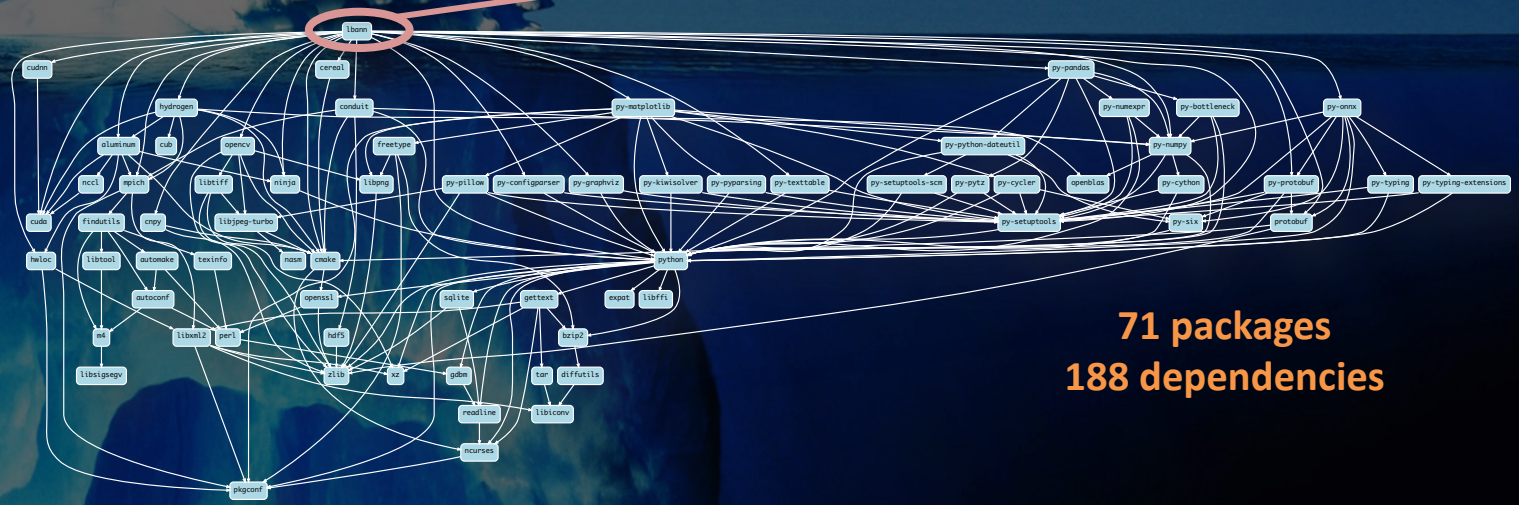
Lawrence Livermore National Lab
AMD Zen / AMD GPU

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

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

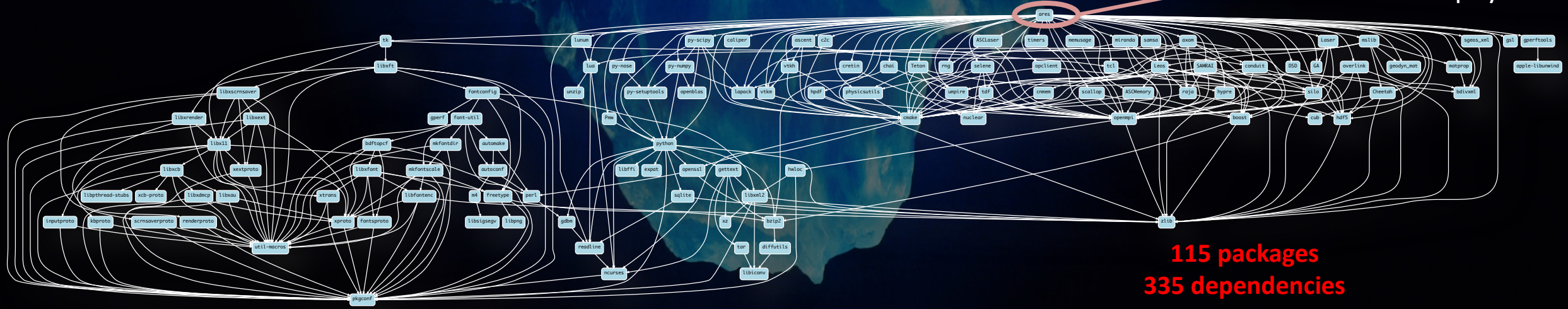


LBANN: Neural Nets for HPC



71 packages
188 dependencies

ARES: LLNL Multi-physics



115 packages
335 dependencies

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

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

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 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

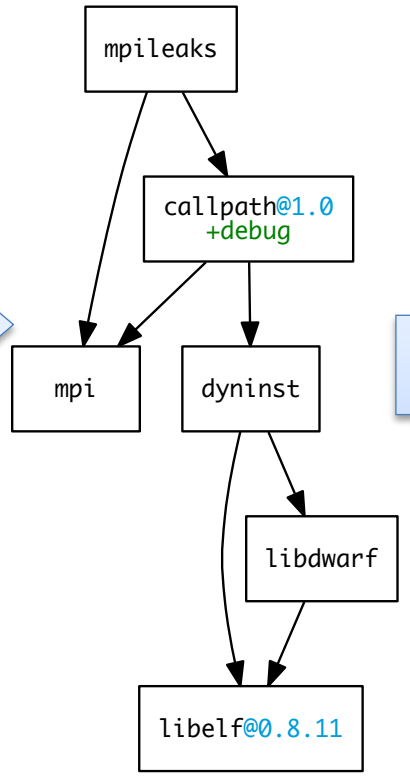
There is a lot of expressivity in this DSL.

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

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

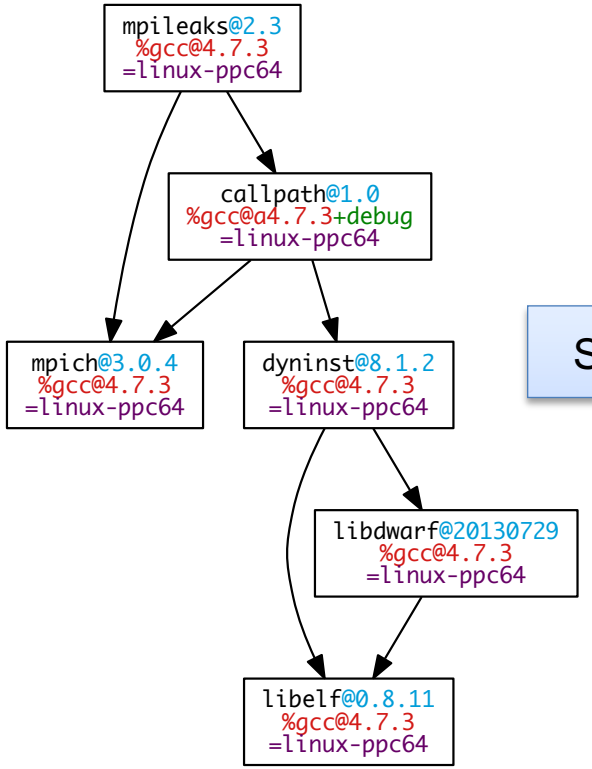
User input: *abstract* spec with some constraints

Normalize



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

Concretize



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

Store

spec.yaml

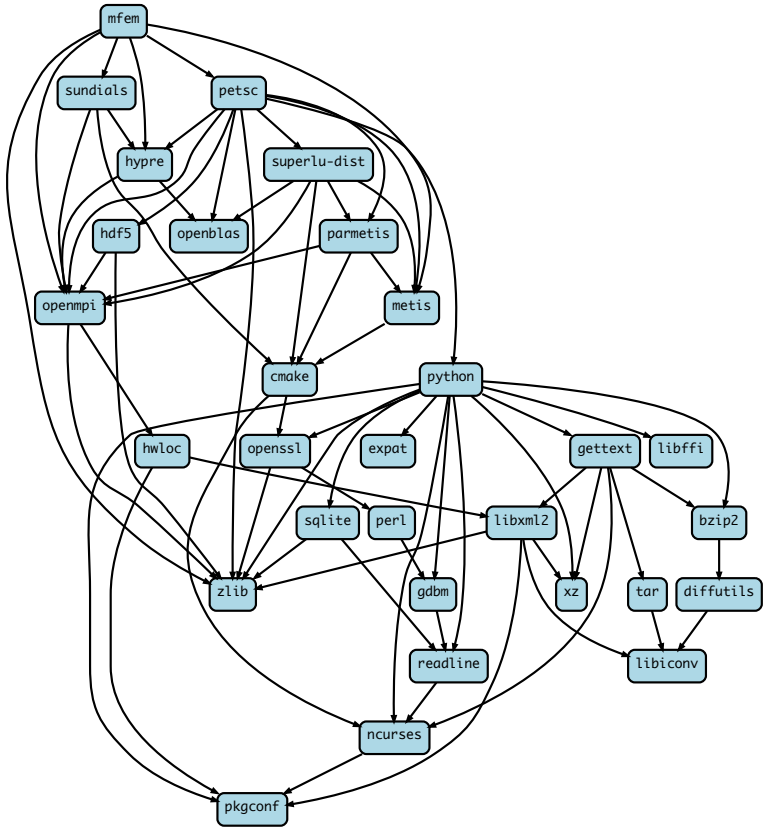
```
spec:
- mpileaks:
    arch: linux-x86_64
    compiler:
      name: gcc
      version: 4.9.2
    dependencies:
      adept-utils: kszrtkpbzac3ss2ixcjkcorlaybnptp4
      callpath: bah5f4h4d2n47mgycej2mtrnrivvxy77
      mpich: aa4ar6ifj23yijqmdabeakpejcli72t3
    hash: 33hjjhxi7p6gyzn5ptgyes7sghyprujh
    variants: {}
    version: '1.0'
- adept-utils:
    arch: linux-x86_64
    compiler:
      name: gcc
      version: 4.9.2
    dependencies:
      boost: teesjv7ehpe5ksspjim5dk43a7qnowlq
      mpich: aa4ar6ifj23yijqmdabeakpejcli72t3
    hash: kszrtkpbzac3ss2ixcjkcorlaybnptp4
    variants: {}
    version: 1.0.1
- boost:
    arch: linux-x86_64
    compiler:
      name: gcc
      version: 4.9.2
    dependencies: {}
    hash: teesjv7ehpe5ksspjim5dk43a7qnowlq
    variants: {}
    version: 1.59.0
...
```

Detailed provenance 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



There are much better solutions out there than SAT Solvers

- **SAT: Boolean Satisfiability**
  - Hard to model the problem space with just True and False to work with
  - Optimization is very hard to implement on top (and slow)

- **SMT: Satisfiability modulo theories**
  - Support for integer math, implications, higher level logic operations
  - Support for multi-criteria optimization
  - Traction in the formal verification community
  - Can generate unsatisfiable cores and proofs for error cases (but proofs are complex)

- **ASP: Answer Set Programming (not the other ASP)**
  - Clingo (from the Potassco project) is very actively developed, and very fast
  - Looks like prolog; boils down to SAT
  - Easy to read, declarative modeling language
  - Support for multi-criteria optimization
  - Can produce unsatisfiable cores that help explain errors.

All of these use fast SAT solvers, but ASP is a much higher level paradigm.

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'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

ASP makes it easy to put the logic in one place, declaratively

Define the space:
each package must be assigned exactly one version.

Disallow conflicted versions

Minimize the total of all version weights

```
% If something is a package, it has only one version and that must be a
% possible version.
1 { version(P, V) : version_possible(P, V) } 1 :- node(P).

% If a version is declared but conflicted, it's not possible.
version_possible(P, V) :- version_declared(P, V), not version_conflict(P, V).

% version weight and optimization
version_weight(P, V, N) :- version(P, V), version_declared(P, V, N).
#minimize{ N@8,P,V : version_weight(P, V, N) }.
```

Previously complicated ABI/target logic became very simple

- Every node in the DAG has a compiler and a target architecture
  - Some compilers don't support generating code for some targets
  - But we want to pick the best target possible for each compiler
- Previously this required some complicated logic mixed in with the rest of the solve

Each node has 1 target assigned

Disallow cases where the compiler doesn't support the target.

Minimize the total weight of all targets

```
% one target per node -- optimization will pick the "best" one
1 { node_target(P, T) : target(T) } 1 :- node(P).

% can't use targets on node if the compiler for the node doesn't support them
:- node_target(P, T), not compiler_supports_target(C, V, T),
   node_compiler(P, C), node_compiler_version(P, C, V).

% if a target is set explicitly, respect it
node_target(P, T) :- node(P), node_target_set(P, T).

% each node has the weight of its assigned target
node_target_weight(P, N) :- node(P), node_target(P, T), target_weight(T, N).
#minimize{ N@5,P : node_target_weight(P, N) }.
```

Encoding generalized conditions was more tricky

- Every condition in the solve gets an id
  - Triggers (requirements) and imposed constraints are associated with the condition id.
  - Different types of conditions have different semantics (dependencies, conflicts, etc.)

"If cmake is at version 3.15 or higher, and NOT using vendored libraries (ownlibs), it depends on libarchive version 3.3.3 or higher"

Spack Package DSL

```
depends_on('libarchive@3.3.3:',      when='@3.15.0:~ownlibs')
```

ASP Facts

```
condition(175).
condition_requirement(175,"node","cmake").
condition_requirement(175,"version_satisfies","cmake","3.15.0:").
condition_requirement(175,"variant_value","cmake","ownlibs","False").
imposed_constraint(175,"version_satisfies","libarchive","3.3.3:").
dependency_condition(175,"cmake","libarchive").
dependency_type(175,"build").
dependency_type(175,"link").
```

Code to trigger and impose general conditions is surprisingly simple (to read)

- *Conditional Rules* allow us to build new rules from input facts
  - Colon here says "if this requirement is true, then put it in the rule body"

```
condition_holds(ID) :-
  condition(ID);
  attr(Name, A1)          : condition_requirement(ID, Name, A1);
  attr(Name, A1, A2)      : condition_requirement(ID, Name, A1, A2);
  attr(Name, A1, A2, A3) : condition_requirement(ID, Name, A1, A2, A3).
```

Same rule, different *arity*

- A condition that holds imposes its constraints (unless canceled w/do_not_impose)

```
impose(ID) :- condition_holds(ID), not do_not_impose(ID).
```

- Impose all constraints if `impose(ID)` is true.

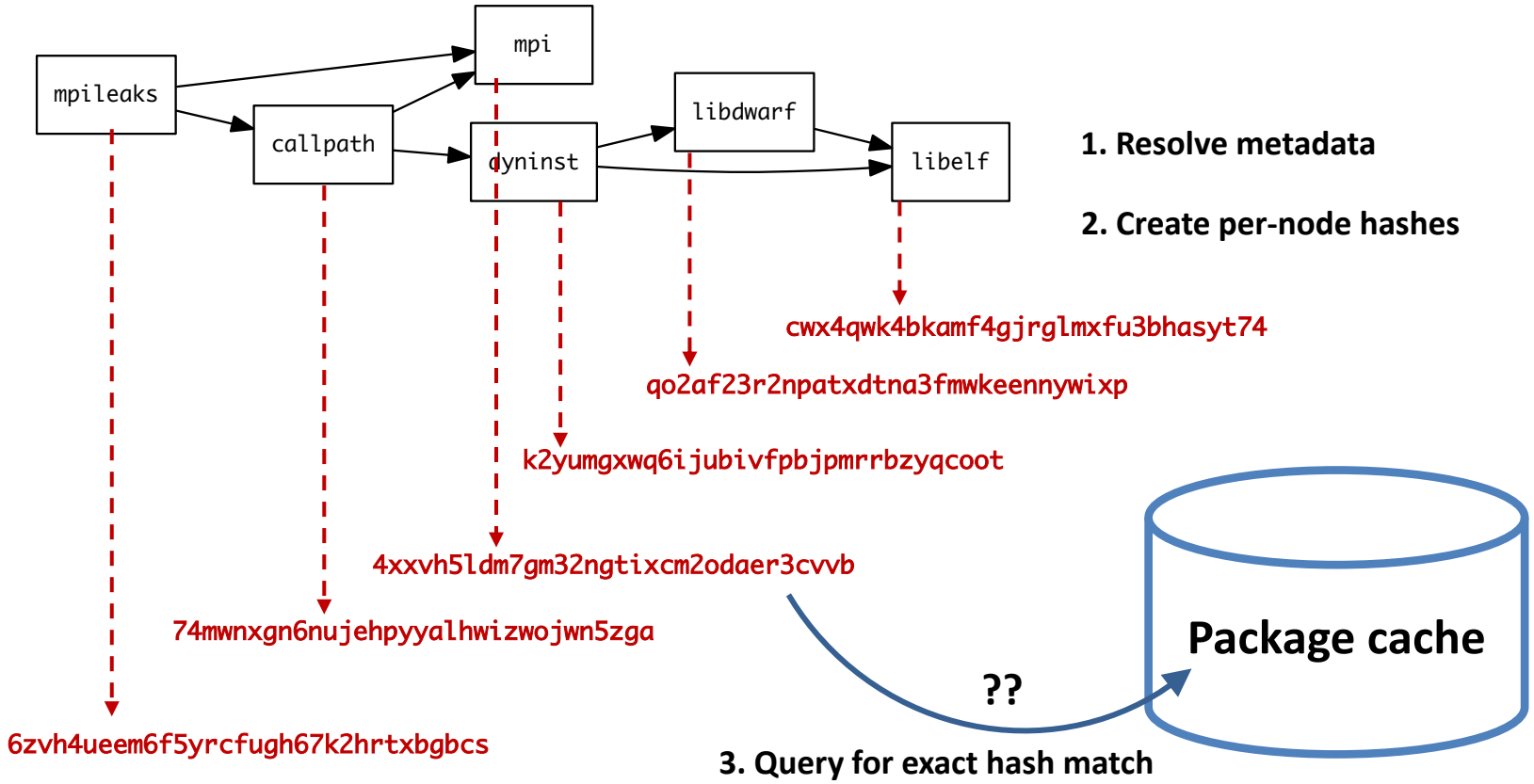
```
attr(Name, A1)          :- impose(ID), imposed_constraint(ID, Name, A1).
attr(Name, A1, A2)      :- impose(ID), imposed_constraint(ID, Name, A1, A2).
attr(Name, A1, A2, A3) :- impose(ID), imposed_constraint(ID, Name, A1, A2, A3).
```

We use optimization to choose the “best” of all valid solutions

- Tend to be a lot of *valid* solutions in Spack
  - Many versions can satisfy given constraints
  - Most packages have loose constraints
- Choosing configurations that are “intuitive” to users can be difficult
  - Build specification is more of an art than a science
- We’ve added 15 ”base” optimization criteria to our solver
- Currently, criteria for roots are prioritized and other nodes are aggregated.
  - Would *really* like DAG precedence
  - Has proven hard to implement efficiently so far but we have ideas

Priority	Criterion (to be minimized)
1	Deprecated versions used
2	Version oldness (roots)
3	Non-default variant values (roots)
4	Non-preferred providers (roots)
5	Unused default variant values (roots)
6	Non-default variant values (non-roots)
7	Non-preferred providers (non-roots)
8	Compiler mismatches
9	OS mismatches
10	Non-preferred OS’s
11	Version oldness (non-roots)
12	Unused default variant values (non-roots)
13	Non-preferred compilers
14	Target mismatches
15	Non-preferred targets

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
```

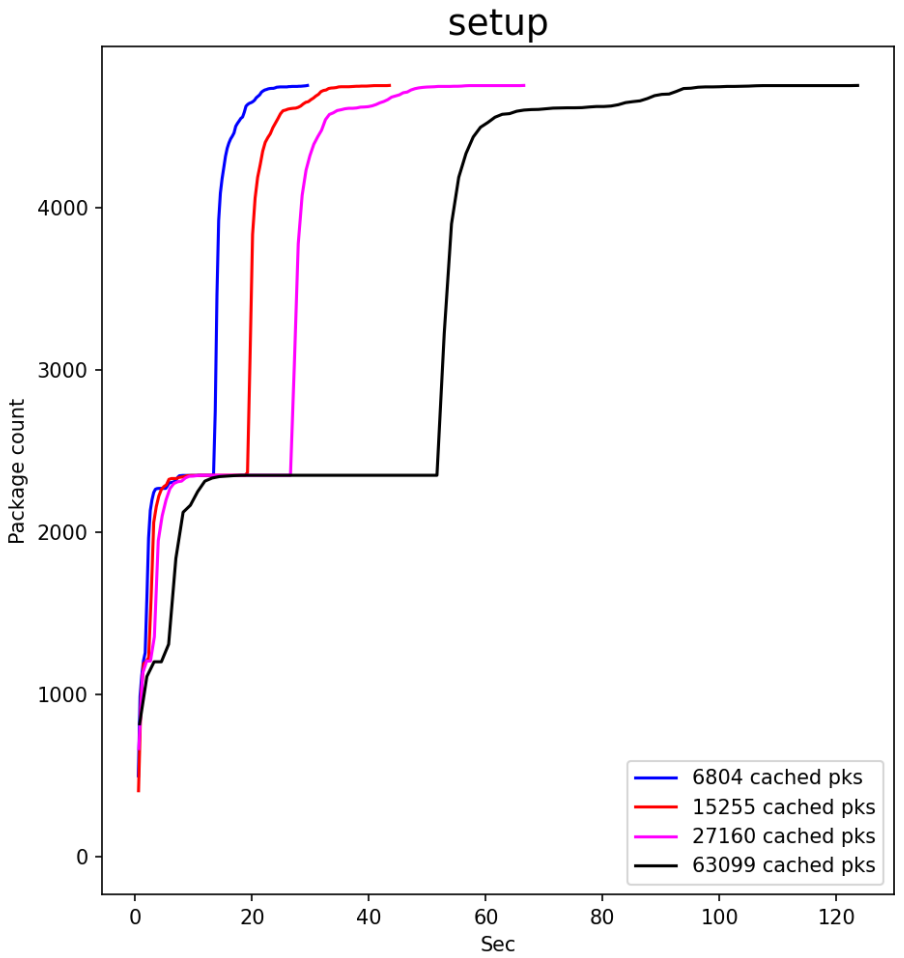
With reuse: 16 packages were actually acceptable

We had to take some special steps to make the builds less weird

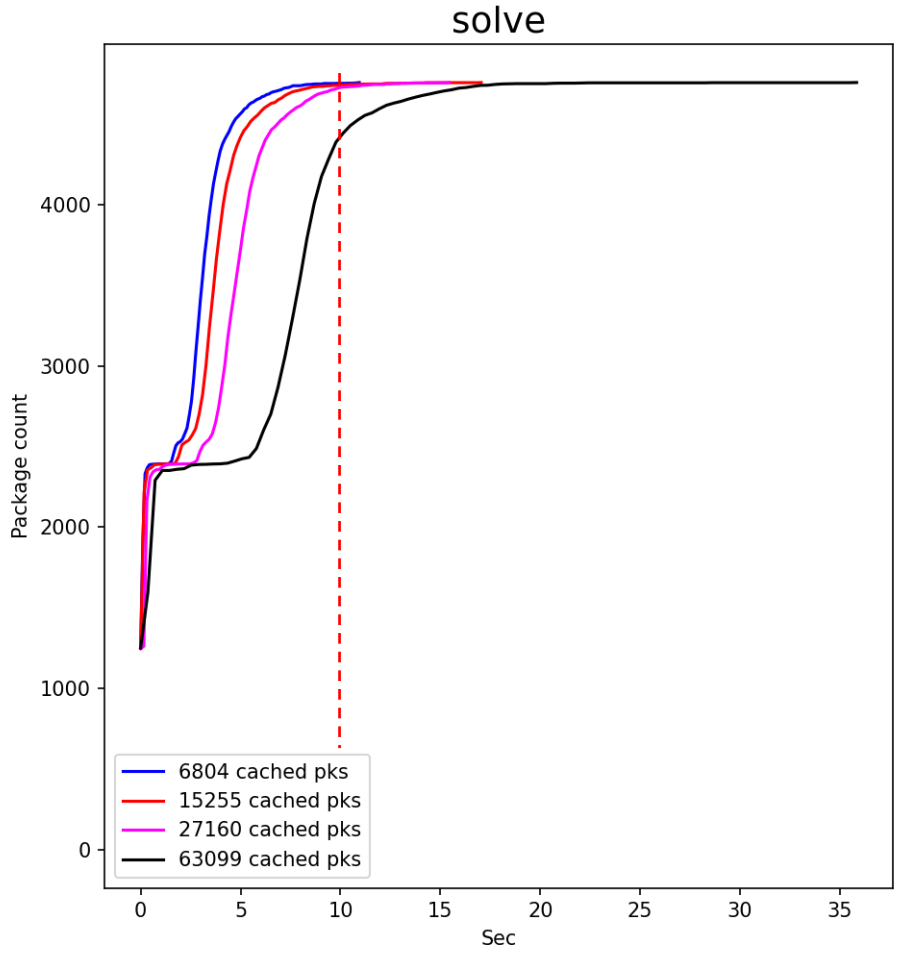
- If we *just* minimize builds, we get strange behavior when we have to build things new
  - E.g.: Cmake depends on openssl for https
  - Minimizing builds will toggle this feature *off*, but most users wan a functional Cmake
- We made the following change:
  - Prioritize minimizing builds, *unless we have to build*
  - Prioritize *defaults* for specs we *have* to build
- Rationale:
  - If you've installed some version of something, you're *probably* ok with that version
  - You can use –fresh to get a completely up-to-date install
- To get this to work:
  - All criteria must be formulated as minimizations
  - No built configuration can be "better" than a reused configuration

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

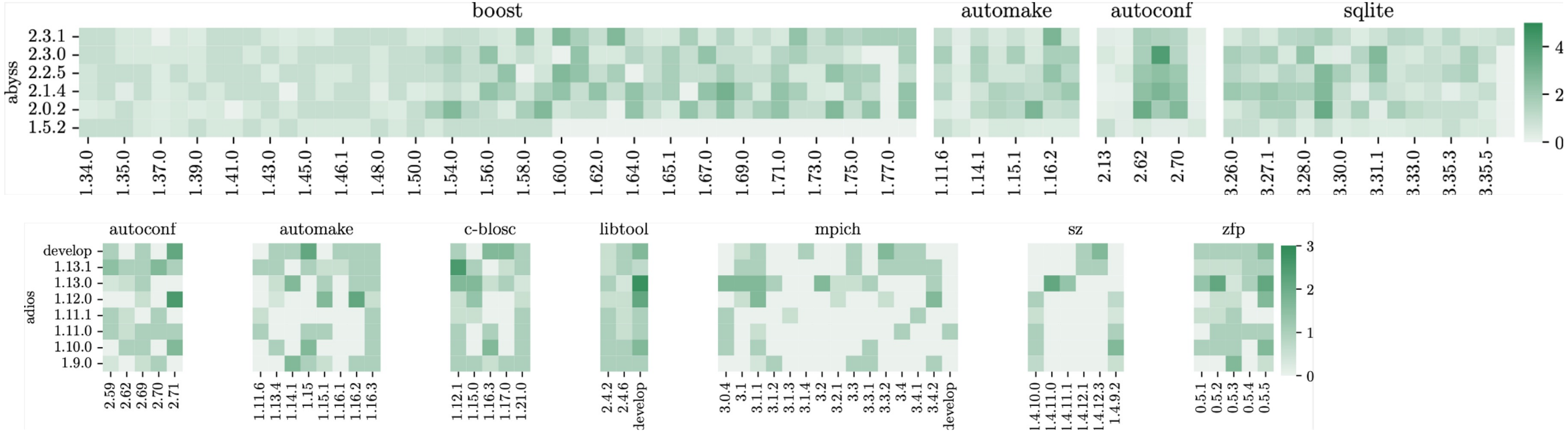


Most of the time is spent in setup (reading data in Python – can be sped up w/caching)



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

Future work: integrate pairwise success likelihood into the solver, to guide it to high-likelihood solutions.



We can explore the build space and find configurations that are likely to work

- We run rouhly 40k builds in CI each week
- Fuzz the CI builds to build a model like the one above

Next steps:

- Use these weights with an optimization function to guide the solver
- Investigate online training, update strategies to keep model current for low CPU cost
- Investigate transfer learning for porting from one system to another