

Spack Update

EasyBuild User Meeting 2024

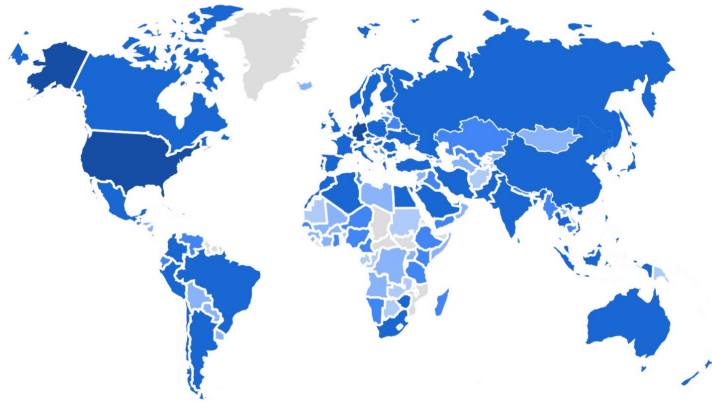
Todd Gamblin

Lawrence Livermore National Laboratory

April 24, 2024



Spack continues to grow

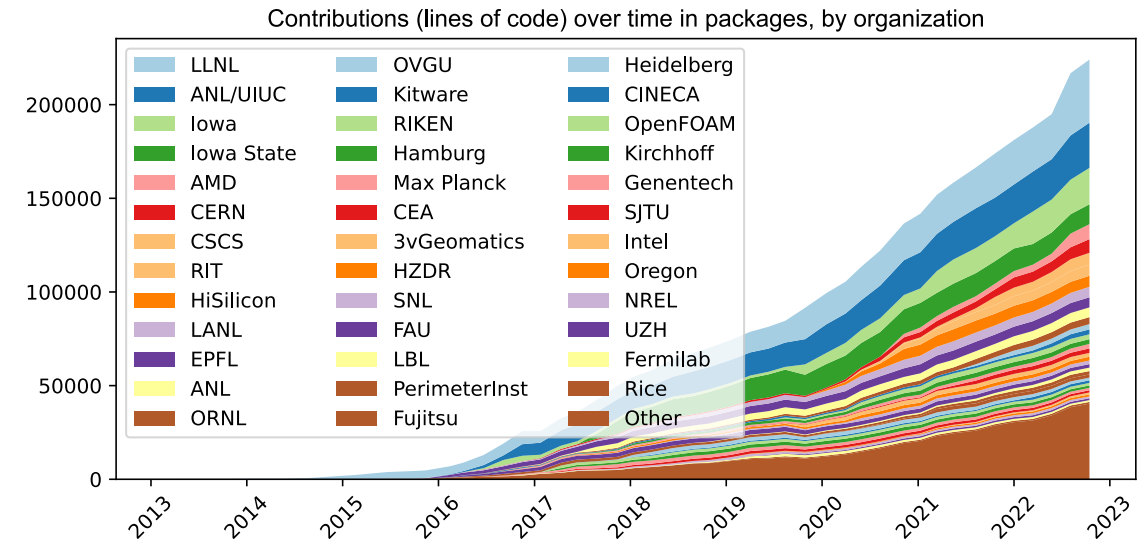


COUNTRY	USERS
United States	23K
Germany	5.3K
China	4.6K
India	4.5K
United Kingdom	3.3K
France	3K
Japan	2.4K

2023 aggregate user counts from GA4

(note: yearly user counts are almost certainly too large)

Over 7,900 software packages
Over 1,300 contributors

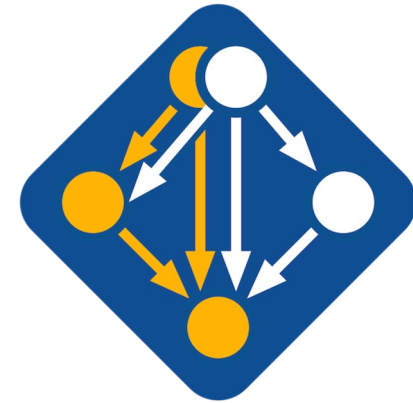


Most package contributions are **not** from DOE
But they help sustain the DOE ecosystem!

Contributors continue to grow worldwide

Spack is officially a Linux Foundation Project now!

- What does that mean?
 - Project has a legal 501(c)(6) non-profit company
 - This is a neutral legal entity
 - Can be in legal agreements (e.g. for distributing binaries)
 - Can get discounts on, e.g., Slack!
 - Project will have a Technical Steering Committee (TSC)
 - Plan is to make the main developer meetings more public
 - Also have official steering committee meetings
 - Main charter is written (mostly boilerplate)
 - Working on initial GOVERNANCE.md, initial TSC members
 - Trademark (Spack name, logo) assigned to Linux Foundation
 - Project resources owned by Linux Foundation
 - spack.io website
 - GitHub Organization



We are forming the High Performance Software Foundation



HIGH PERFORMANCE SOFTWARE FOUNDATION

- Intent to form announced at Supercomputing 2023
- Will officially kick off at ISC'24
- BOF on Monday, May 13, 1pm

Anchor Members

Premier



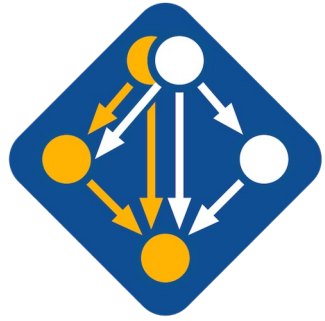
General



Associate



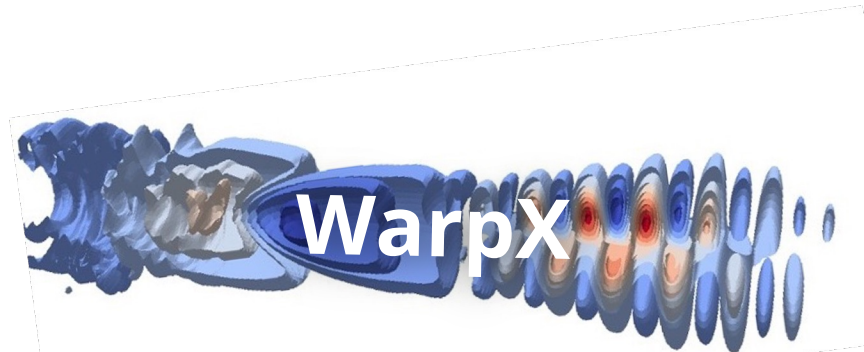
Initial Projects



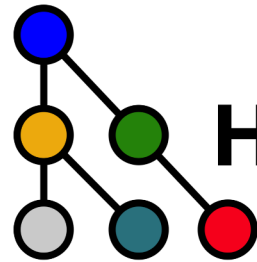
Spack



kokkos



APPTAINER



HPCToolkit



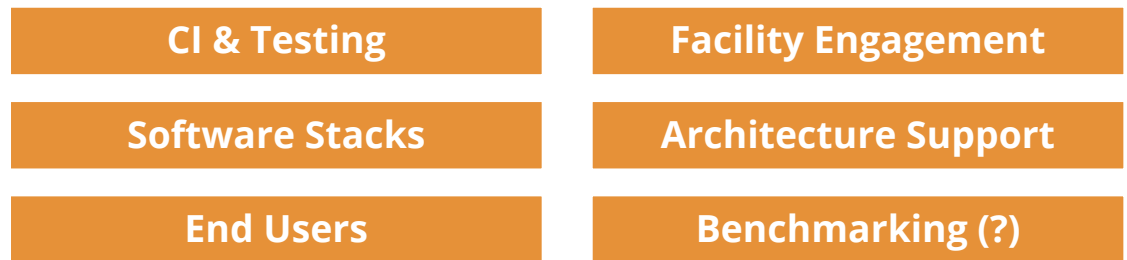
Proposed HPSF Structure



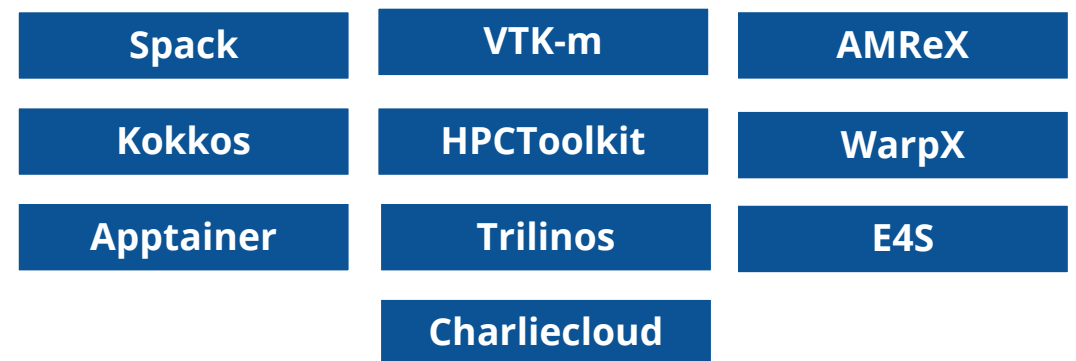
Collaborations



Working Groups



Projects



HPSF Goals

1. Provide neutral home for key HPC projects to enable collaboration between government, industry and academia
2. Promote use of HPSF projects
3. Ensure software is accessible and reliable with CI and turn-key builds
4. Ensure that HPC software is secure and ready for cloud through collaborations with CNCF and OpenSSF
5. Sponsor events and training to grow a diverse, skilled workforce for software in the HPSF ecosystem.

Increase adoption and contribution!

Spack provides a *spec* syntax to describe customized package configurations (constraints)

\$ 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"

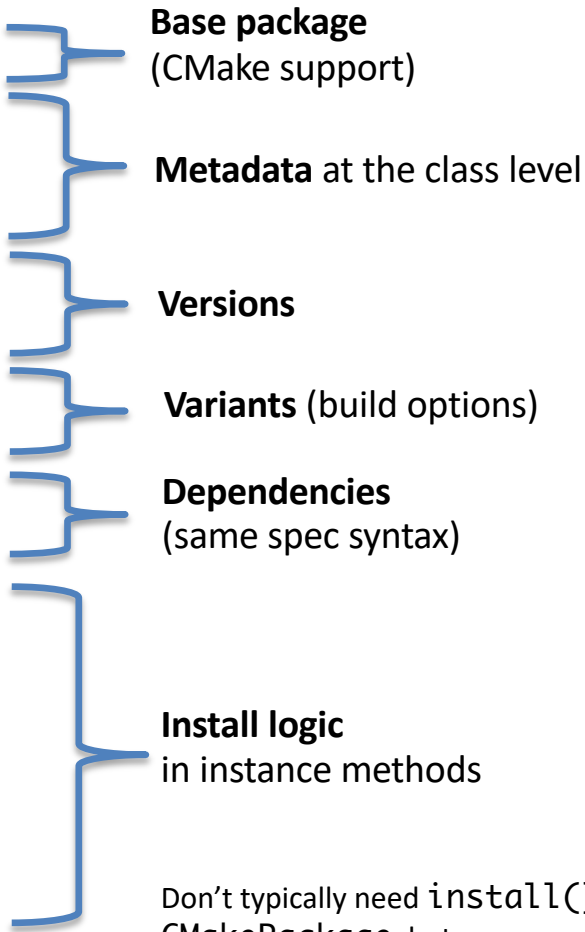
    version('1.2.3', sha256='3f7f2eef0d1ba5825780d626741eb0b3f026a096048d7ec4794d2a7dfbe2b8a6')
    version('1.2.2', sha256='eaf9ddf562416974157b34d00c3a1c880fc5296fce2aa2efa039a86e0976f3a3')
    version('1.1', sha256='232d74072fc7b848fa2adc8a1bc839ae8fb5f96d50224186601f55554a25f64a')

    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)
```



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

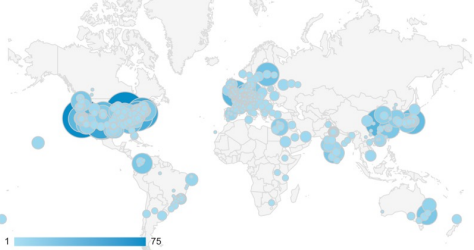
compiler support for x86_64
and ppc64le

Complexity has grown with the addition of GPU stacks and compiler information

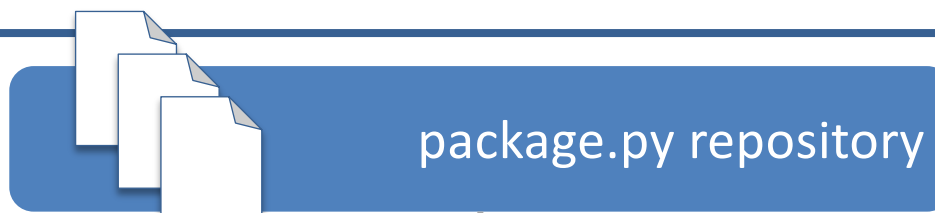
Spack's concretizer resolves *many* preferences into a *concrete*, installable specification

This part is NP-hard!

Contributors



- new versions
- new dependencies
- new constraints



package.py repository

spack developers



default config
packages.yaml

admins,
users



local preferences config
packages.yaml

users

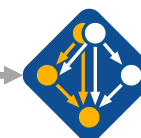


local environment config
spack.yaml

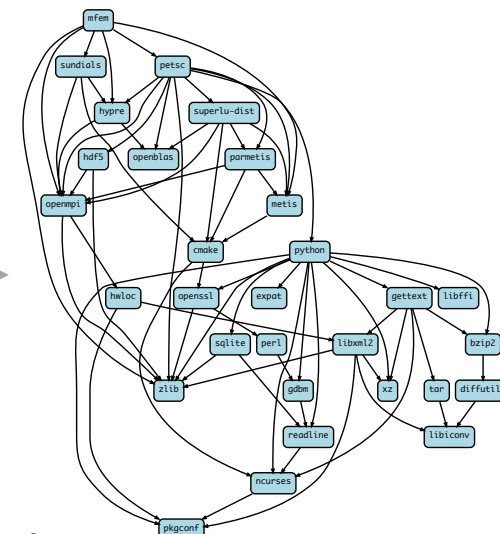
users

Command line constraints

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



concretizer



Concrete spec is fully constrained and can be built.

Is stored in spack.lock file after solve.

Spack v0.22 is coming in May

Lots of updates but will talk about two big ones here:

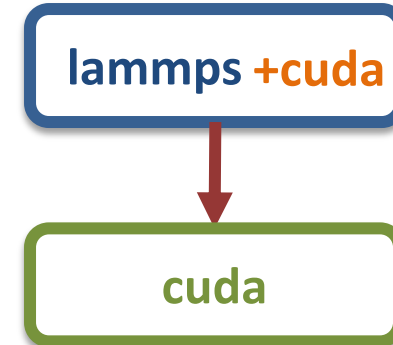
1. Compiler dependencies!
2. Python ecosystem support (package auto-generation)
 - These have both been in the works for a long time
 - Both made possible by recent solver work

Spack's concretizer is implemented using Answer Set Programming (ASP)

ASP looks like Prolog but is converted to SAT with optimization

Facts describe the graph

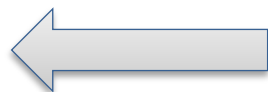
```
node("lammps").  
node("cuda").  
variant_value("lammps", "cuda", "True").  
depends_on("lammps", "cuda").
```



First-order rules (with variables) describe how to resolve nodes and metadata

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

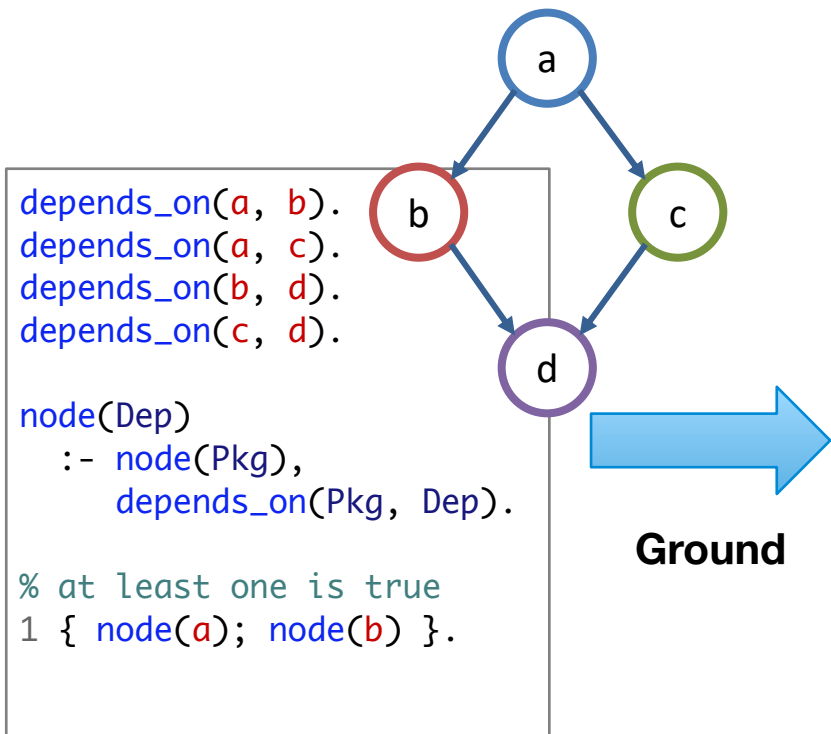
```
node("mpi")
```



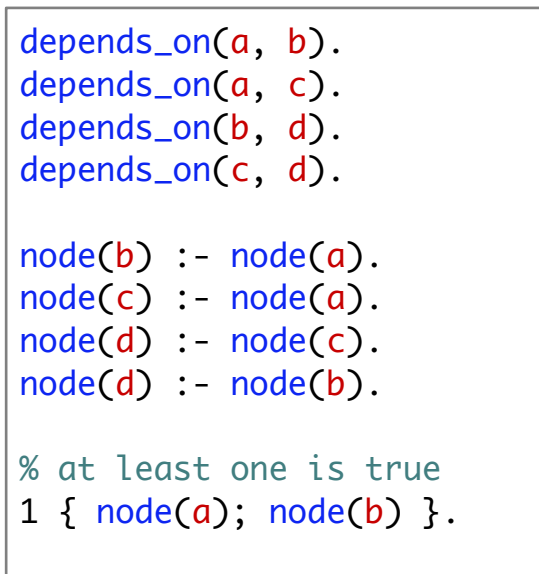
```
node("hdf5").  
depends_on("hdf5", "mpi").
```

Ground Rule

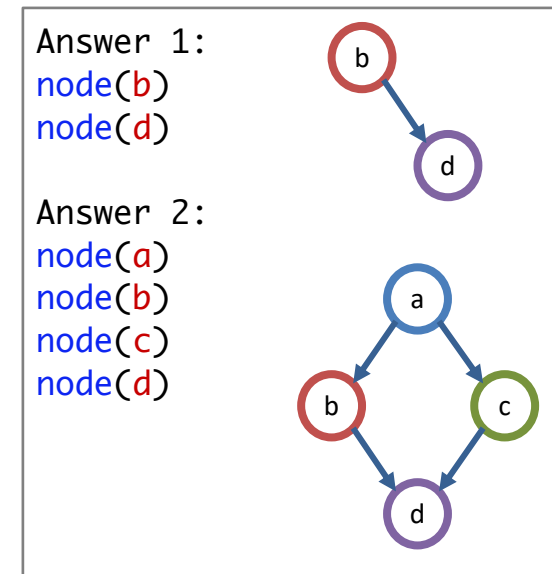
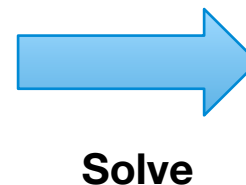
Grounding converts a first-order logic program into a propositional logic program, which can be solved.



First-order Logic Program



Propositional Program



Stable Models (Answer Sets)

Answer 1: Only node(b) is true

Answer 2: Both node(a) and node(b) are true

We use the *Clingo* solver from potassco.org

ASP searches for *stable models* of the input program

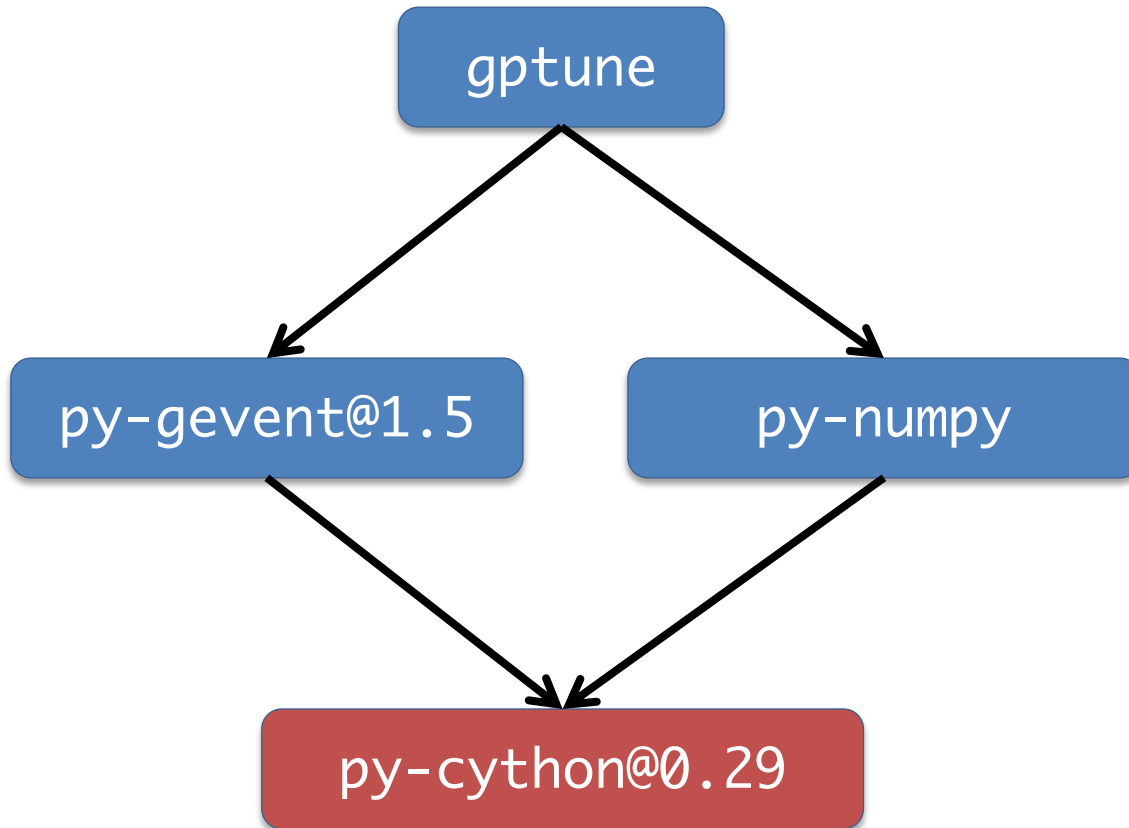
- Stable models are also called *answer sets*
- A *stable model* (loosely) is a set of true atoms that can be deduced from the inputs, where every rule is idempotent.
 - Similar to fixpoints
 - Put more simply: a *set of atoms where all your rules are true!*
- Unlike Prolog:
 - Stable models contain everything that can be derived (vs. just querying values)
 - ASP is guaranteed to complete!

Some stats on problem sizes

- Main logic program is:
 - ~250 rules
 - 20 optimization criteria
 - 933 lines of ASP code
- Problem instances can vary quite a bit
 - Common dependencies get us some magic numbers
 - gmake's optional dependency on guile makes most solves consider at least 527 packages
 - gnuconfig is notably very simple 😊

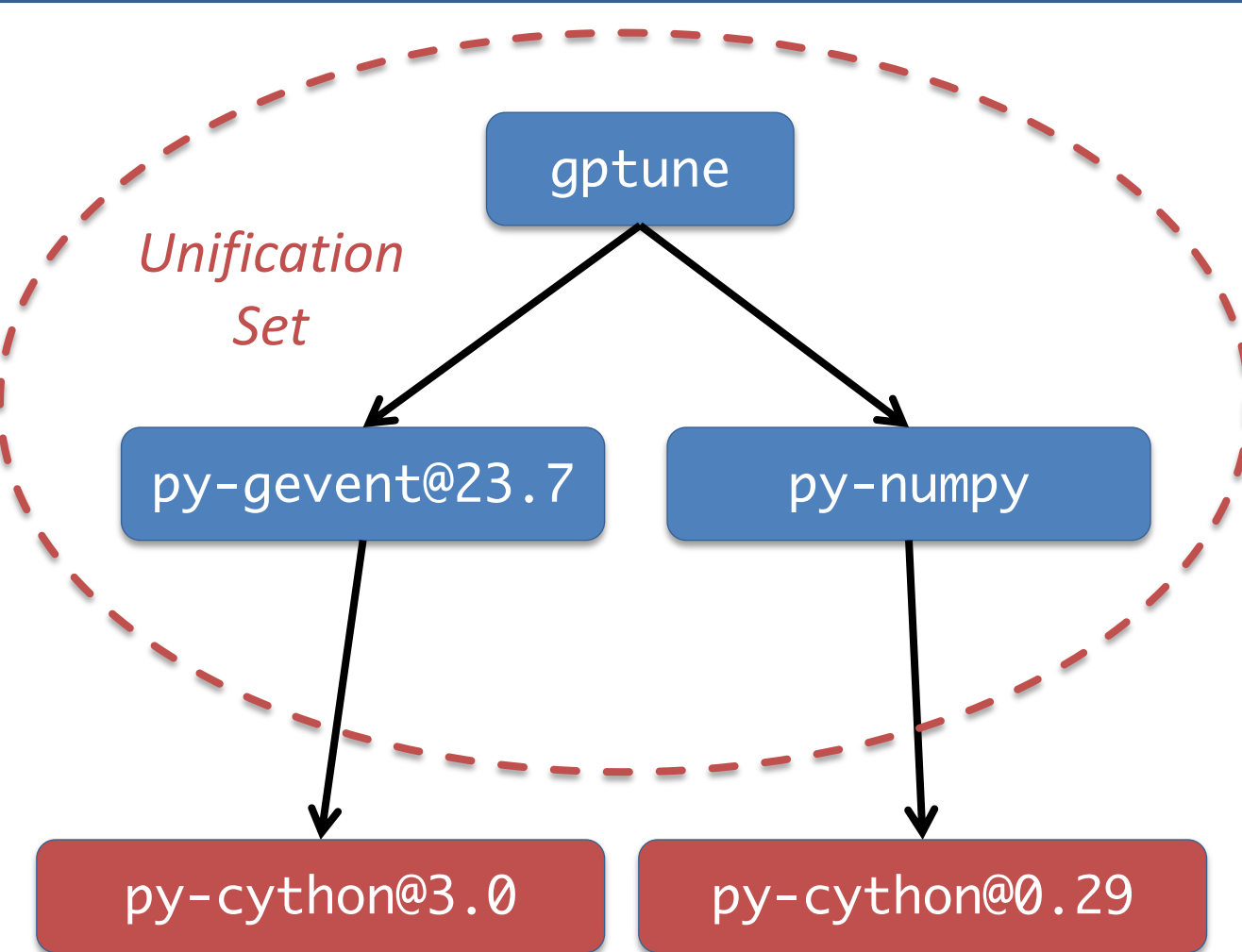
Package	Possible dependencies	Facts
gnuconfig	1	150
zlib	527	30,095
gmake	527	30,160
openmpi	527	109,021
qt	527	109,029
trilinos	694	224,142
root	699	146,372
mfem	714	273,078
r-condop	774	142,212
warpx	819	319,374
exawind	820	322,535

We have been working to make our solver more flexible



- Only one configuration per package allowed in the DAG
- Ensures ABI compatibility but is too restrictive
- In the example py-numpy needs to use py-cython@0.29 as a build tool
- That enforces using an old py-gevent, because newer versions depend on py-cython@3.0 or greater

Objective: dependency splitting



- The constraint on build dependencies can be relaxed, without compromising the ABI compatibility
- Having a single configuration of a package is now enforced on unification sets
- These are the set of nodes used together at runtime (the one shown is for gptune)
- This allows us to use the latest version of py-gevent, because now we can have two versions of py-cython

We want to dynamically “split” nodes when needed


Start with deducing single dependency nodes:

```
node(DependencyName)
  :- dependency_holds(PkgName, DependencyName)
```

Want to allow solver to **choose** to duplicate a node:

Converted node identifier
from **name** to **(name, id)**

```
1 {
  depends_on(PkgNode, node(0..Y-1, DepNode), Type)
  : max_dupes(DepNode, Y)
} 1
:- dependency_holds(PkgNode, DepNode).
```



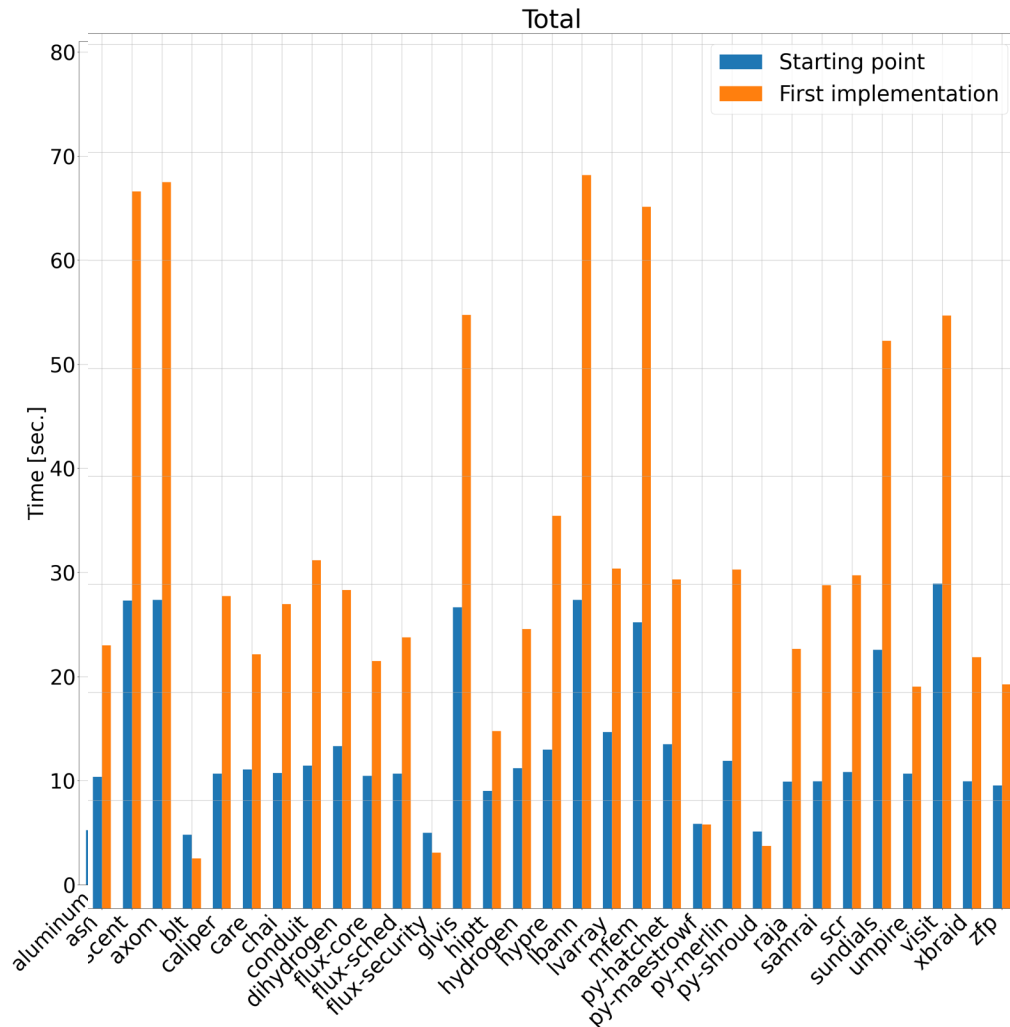
Generic package metadata can be used with any duplicate node

```
pkg_fact("alsa-lib", version_declared("1.2.3.2", 0, "package_py")).
pkg_fact("alsa-lib", version_declared("1.2.2", 1, "package_py")).
pkg_fact("alsa-lib", version_declared("1.1.4.1", 2, "package_py")).

pkg_fact("alsa-lib", condition(20)).
condition_reason(20, "alsa-lib depends on python when +python").
pkg_fact("alsa-lib", condition_trigger(20, 15)).
```

- Facts that come from package descriptions can be used with all duplicate nodes
- We now have to ground multiple copies of most of our rules
- Performance still scales with total number of possible nodes
 - Small numbers of duplicates don't really explode the solve

First try at allowing duplicates in a single solve



**Increased solve times by
>> 2x in some cases**

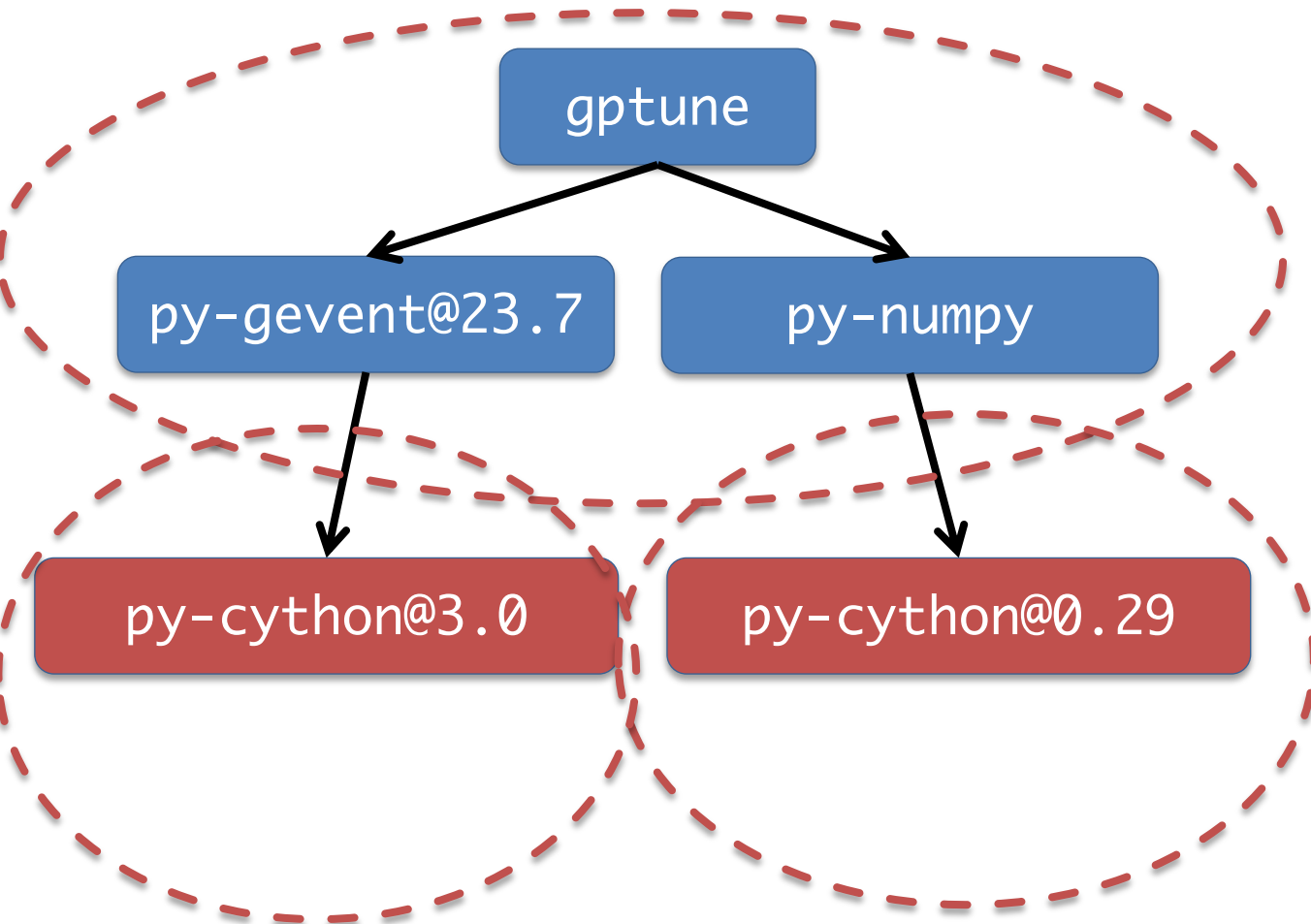
It turns out that cycle detection in the solver is *expensive*

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

- Has to maintain path() predicate representing paths between nodes
- Cycles are actually rare in solutions
 - Switched to post-processing for cycle detection
 - Only do expensive solve if a cycle is detected in a solution
- Similar issue arose for variant propagation in graph
 - Fixed by reworking variant propagation not to track paths

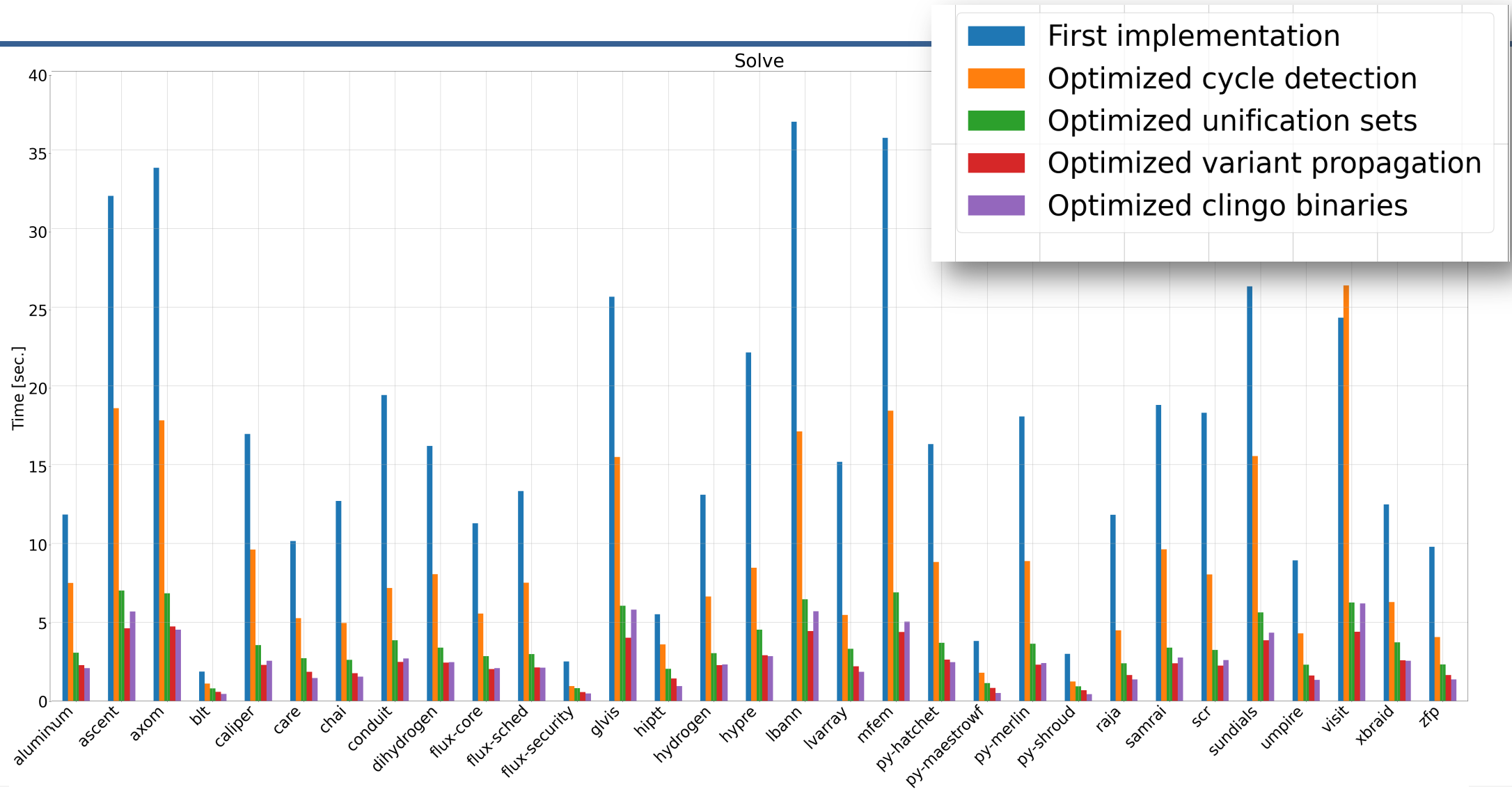
**50%+ improvement
in solve time**

Unification sets can be expensive too

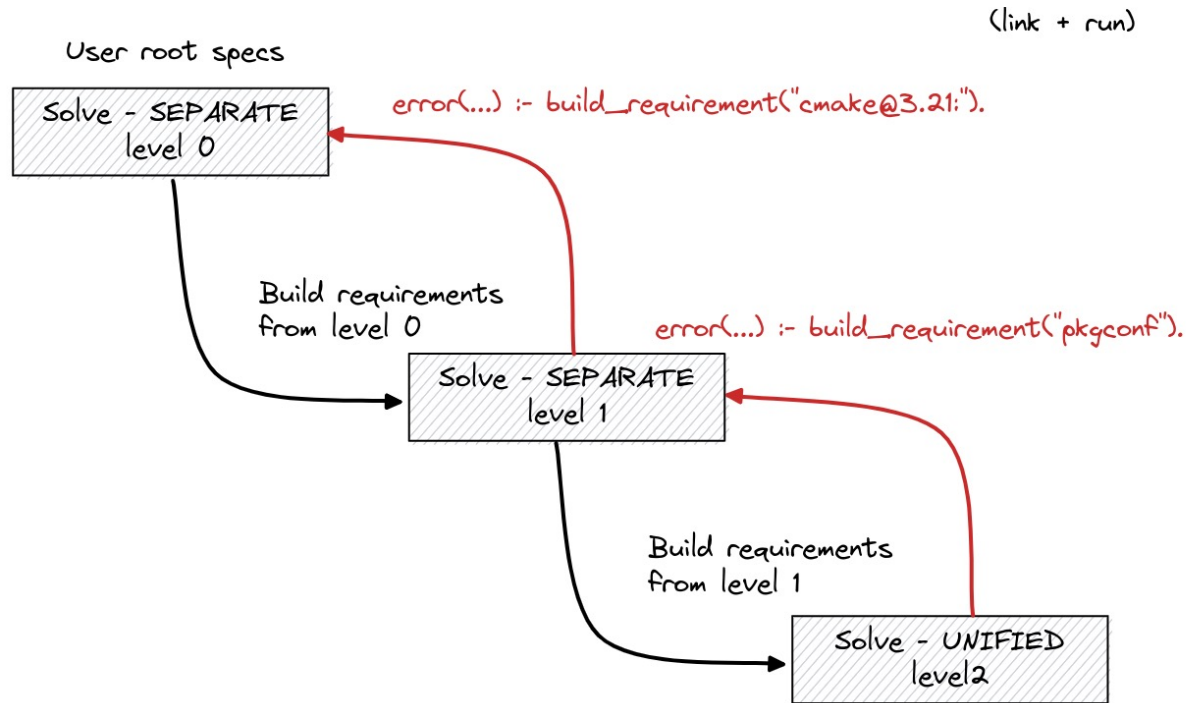


- Unification set creation was originally recursive for *any* build dependencies
 - Ends up blowing up grounding
- Mitigation:
 - Only create new sets for explicitly *marked* build tools
 - Transitive build dependencies that are not from marked build tools go into a *common* unification set
- Need better heuristics to split when necessary for full generality

Optimizations: Solve Time



It was not trivial to come up with this model



- In addition to this “coupled” method, we tried an iterative version with multiple solves
- Multiple solves had some disadvantages:
 - Slower due to overhead of multiple solves
 - Not coupled, so feedback from solve to solve was awkward
 - Packagers needed to “help” the solver
- Requiring packagers to provide solve hints in packages isn’t practical

We've made a lot of progress on compiler dependencies

- Compiler *runtime libraries* represented in the graph
 - C++, Fortran runtimes
- libc is now represented in dependency graphs on Linux
 - No more need to rely on OS tag for compatibility information
- Reuse binaries *without* their compiler needing to be configured locally
- Improved buildcache hit rate using libraries for compatibility

Compilers can now model their own runtimes

- New method, `runtime_constraints`, for injecting runtimes into graphs
- Currently supported for `gcc`, `intel-oneapi`
 - still working on others
- Allows solver to take `libstdc++`, `fortran runtime compatibility` into account.
- Example:
 - Intel compilers now (correctly) depend on `gcc-runtime`

```
class Gcc(AutotoolsPackage, GNUMirrorPackage):
    # ...

    @classmethod
    def runtime_constraints(cls, *, spec, pkg):
        """Callback function to inject runtime-related rules into the solver.

        Rule-injection is obtained through method calls of the ``pkg`` argument.

        Documentation for this function is temporary. When the API will be in its final state,
        we'll document the behavior at https://spack.readthedocs.io/en/latest/

        Args:
            spec: spec that will inject runtime dependencies
            pkg: object used to forward information to the solver
        """
        pkg("*").depends_on(
            "gcc-runtime",
            when="%gcc",
            type="link",
            description="If any package uses %gcc, it depends on gcc-runtime",
        )
        pkg("*").depends_on(
            f"gcc-runtime@{str(spec.version)}:",
            when=f"%{str(spec)}",
            type="link",
            description=f"If any package uses %{str(spec)}, "
            f"it depends on gcc-runtime@{str(spec.version)}:",
        )
```

Packages now declare the languages they depend on

- Languages are *almost* virtuals
 - HDF5 package depends on `cxx` and `fortran`
 - Handled specially internally, until compilers are nodes
- Imply a compiler *and* compiler package can specify runtime libraries to inject
- Allows solver to mix compilers *correctly*
 - Runtimes are unified like other nodes
 - *Package authors* can model toolchain properties
 - probably not for most package authors, but very powerful
- TBD:
 - Other runtimes like clang libraries and OpenMP
 - Compilers as nodes in the graph

```
class Hdf5(CMakePackage):
    """HDF5 is a data model, library, and file format for storing and managing
    data. It supports an unlimited variety of datatypes, and is designed for
    flexible and efficient I/O and for high volume and complex data.
    """

    homepage = "https://portal.hdfgroup.org"
    url = "https://support.hdfgroup.org/ftp/HDF5/releases/hdf5-1.14/hdf5-1.14"
    list_url = "https://support.hdfgroup.org/ftp/HDF5/releases"
    list_depth = 3
    git = "https://github.com/HDFGroup/hdf5.git"
    maintainers("lrknox", "brtnfld", "byrnHDF", "gheber", "hyoklee", "lkurz")

    tags = ["e4s", "windows"]
    executables = ["^h5cc$", "^h5pcc$"]

    test_requires_compiler = True

    license("custom")

    depends_on("cxx", type="build", when="+cxx")
    depends_on("fortran", type="build", when="+fortran")
```

We've also added libc as a dependency

- libc is a virtual
 - glibc and musl packages are providers
 - (nearly) every graph has libc in it, via the compiler
 - Can be external or built by Spack
- We are *not* building libc for every stack in Spack
 - Automatically detect system libc version
 - Add a node to the graph to be used for binary compatibility
- No longer using OS tags for buildcaches
 - Now use libc for this
 - *many* more buildcache hits

```
(py311) culpo@nivola:~/PycharmProjects/spack$ spack concretize -f --re
==> Concretized hdf5~mpi
-   tnqdhns   hdf5@1.14.3%gcc@9.4.0~cxx~fortran~hl~ipo~java~map~mpi+sh
-   gdviueh   ^cmake@3.27.9%gcc@8.5.0~doc+ncurses+ownlibs build_sy
-   i5cd2jj   ^curl@8.6.0%gcc@8.5.0~gssapi~ldap~libidn2~librtm
-   icqajq4   ^nghttp2@1.57.0%gcc@8.5.0 build_system=autot
-   xl7h3wb   ^openssl@3.2.1%gcc@8.5.0~docs+shared build_s
-   rlipoky   ^ca-certificates-mozilla@2023-05-30%gcc@
-   45hdvpf   ^perl@5.38.0%gcc@8.5.0+cpanm+opcode+open
-   qvzagc5   ^berkeley-db@18.1.40%gcc@8.5.0+cxx~d
-   ioufq6d   ^bzip2@1.0.8%gcc@8.5.0~debug~pic+sha
-   enaxy2l   ^diffutils@3.10%gcc@8.5.0 build
-   czvftrb   ^libiconv@1.17%gcc@8.5.0 bui
-   ku6webf   ^gdbm@1.23%gcc@8.5.0 build_system=au
-   3tzxgdp   ^readline@8.2%gcc@8.5.0 build_sy
-   llqwd2j   ^gcc-runtime@8.5.0%gcc@8.5.0 build_system=generi
[e] fue5ca2   ^glibc@2.28%gcc@8.5.0 build_system=autotools arch
-   sxb2sl6   ^ncurses@6.4%gcc@8.5.0~svmlinks+termlib abi=none
-   ucn3b...  ^gcc-runtime@9.4.0%gcc@9.4.0 build_system=generic arch
[e] 37z...mg4  ^glibc@2.31%gcc@9.4.0 build_system=autotools arch=li
-   vsjxwea   ^gmake@4.4.1%gcc@8.5.0~guile build_system=generic arch
-   o76bf47   ^pkgconf@1.9.5%gcc@8.5.0 build_system=autotools arch
-   jkwqkvs   ^zlib-ng@2.1.6%gcc@8.5.0+compat+new_strategies+opt+p
```

Libc modeling makes for a much better buildcache experience

- Currently on develop (emcas 100% from binary):

```
(py311) culpo@nivo1a:~/PycharmProjects/spack$ spack install emacs
[+] /usr (external glibc-2.17-2hhcy7kzv3wlfqcascwhvup4uysp4hoy)
[+] /home/culpo/PycharmProjects/spack/opt/spack/linux-centos7-x86_64_v3/gcc-10.2.1/gcc-runtime-10.2.1-4gmidou73wvttvhun564olcopuijl2i
[+] /home/culpo/PycharmProjects/spack/opt/spack/linux-centos7-x86_64_v3/gcc-10.2.1/gmp-6.2.1-nkhm7cmp6samsykyksuqda77xbxnibfn
[+] /home/culpo/PycharmProjects/spack/opt/spack/linux-centos7-x86_64_v3/gcc-10.2.1/berkeley-db-18.1.40-thoi4z7lozgaednxhd4ojhw2lcsgo5g
[+] /home/culpo/PycharmProjects/spack/opt/spack/linux-centos7-x86_64_v3/gcc-10.2.1/gmake-4.4.1-ucqstlhik7pmv5ijyckmr6mxv46vgeuj
=> Installing nasm-2.15.05-e2kvtqiaava2osr3jbyptdufjgov4dgc [6/39]
=> Fetching https://binaries.spack.io/develop/developer-tools-manylinux2014/build_cache/linux-centos7-x86_64_v3/gcc-10.2.1-nasm-2.15.05-e2kvtqiaava2osr3jbyptdufjgov4dgc.spec.json.sig
gpg: Signature made mar 23 apr 2024, 14:38:11 CEST
gpg: using RSA key D2C7EB3F2B05FA86590D293C04001B2E3DB0C723
gpg: Good signature from "Spack Project Official Binaries <maintainers@spack.io>" [ultimate]
=> Fetching https://binaries.spack.io/develop/developer-tools-manylinux2014/build_cache/linux-centos7-x86_64_v3/gcc-10.2.1/nasm-2.15.05/linux-centos7-x86_64_v3/gcc-10.2.1-nasm-2.15.05-e2kvtqiaava2osr3jbyptdufjgov4dgc
=> Extracting nasm-2.15.05-e2kvtqiaava2osr3jbyptdufjgov4dgc from binary cache
=> nasm: Successfully installed nasm-2.15.05-e2kvtqiaava2osr3jbyptdufjgov4dgc
Search: 0.00s. Fetch: 2.61s. Install: 0.17s. Extract: 0.11s. Relocate: 0.04s. Total: 2.78s
[+] /home/culpo/PycharmProjects/spack/opt/spack/linux-centos7-x86_64_v3/gcc-10.2.1/nasm-2.15.05-e2kvtqiaava2osr3jbyptdufjgov4dgc
=> Installing pcre-8.45-xi42ks7asbq2sqmcd7bdsmhzzhlie6m4 [7/39]
=> Fetching https://binaries.spack.io/develop/developer-tools-manylinux2014/build_cache/linux-centos7-x86_64_v3/gcc-10.2.1-pcre-8.45-xi42ks7asbq2sqmcd7bdsmhzzhlie6m4.spec.json.sig
gpg: Signature made mar 23 apr 2024, 14:38:12 CEST
gpg: using RSA key D2C7EB3F2B05FA86590D293C04001B2E3DB0C723
gpg: Good signature from "Spack Project Official Binaries <maintainers@spack.io>" [ultimate]
=> Fetching https://binaries.spack.io/develop/developer-tools-manylinux2014/build_cache/linux-centos7-x86_64_v3/gcc-10.2.1/pcre-8.45/linux-centos7-x86_64_v3/gcc-10.2.1-pcre-8.45-xi42ks7asbq2sqmcd7bdsmhzzhlie6m4
=> Extracting pcre-8.45-xi42ks7asbq2sqmcd7bdsmhzzhlie6m4 from binary cache
=> pcre: Successfully installed pcre-8.45-xi42ks7asbq2sqmcd7bdsmhzzhlie6m4
Search: 0.00s. Fetch: 2.34s. Install: 0.18s. Extract: 0.14s. Relocate: 0.03s. Total: 2.52s
[+] /home/culpo/PycharmProjects/spack/opt/spack/linux-centos7-x86_64_v3/gcc-10.2.1/pcre-8.45-xi42ks7asbq2sqmcd7bdsmhzzhlie6m4
=> Installing tree-sitter-0.22.2-x5wmc6tind5qxo2zrswb7vyj3xuhak2d [8/39]
=> Fetching https://binaries.spack.io/develop/developer-tools-manylinux2014/build_cache/linux-centos7-x86_64_v3/gcc-10.2.1-tree-sitter-0.22.2-x5wmc6tind5qxo2zrswb7vyj3xuhak2d.spec.json.sig
gpg: Signature made mar 23 apr 2024, 14:38:22 CEST
gpg: using RSA key D2C7EB3F2B05FA86590D293C04001B2E3DB0C723
gpg: Good signature from "Spack Project Official Binaries <maintainers@spack.io>" [ultimate]
=> Fetching https://binaries.spack.io/develop/developer-tools-manylinux2014/build_cache/linux-centos7-x86_64_v3/gcc-10.2.1/tree-sitter-0.22.2/linux-centos7-x86_64_v3/gcc-10.2.1-tree-sitter-0.22.2-x5wmc6tind5qxo2zrswb7vyj3xuhak2d
=> Extracting tree-sitter-0.22.2-x5wmc6tind5qxo2zrswb7vyj3xuhak2d from binary cache
=> tree-sitter: Successfully installed tree-sitter-0.22.2-x5wmc6tind5qxo2zrswb7vyj3xuhak2d
Search: 0.00s. Fetch: 2.34s. Install: 0.00s. Extract: 0.02s. Relocate: 0.04s. Total: 2.32s
```

Managing Python packages in Spack has become unwieldy

- Several reasons:
 - Lots of small packages
 - Dependencies tend to be overconstrained
 - Finding version ranges that allow Spack users to integrate has been hard
 - Adam Stewart has become increasingly busy; moved on from Python ecosystem (only ML packages now)
- So we decided to automate
 - Generate pure python packages
 - Continue to model packages with native builds by hand

Autogenerating Python packages

- Python ecosystem is a lot to maintain
- Increasingly, we need to update faster than we can sustain through pull requests
- Reviewing version ranges has been painful
- Decided to auto-generate Python packages from public metadata

The screenshot shows the GitHub interface for the repository 'spack / pypi-to-spack-package'. The repository is public and has 198 commits. The file list includes:

File Name	Commit Message	Last Commit
contrib	disable info versions / fix percentage	2 weeks ago
src	better progress	2 weeks ago
tests	more edge case	last month
.gitignore	ignore /repo	3 weeks ago
COPYRIGHT		2 months ago
LICENSE-APACHE		2 months ago
LICENSE-MIT		2 months ago
NOTICE		2 months ago
README.md	...	2 weeks ago
pyproject.toml	tests	last month
spack_requirements.txt	new pkgs	2 weeks ago

At the bottom of the repository page, there is a section for the README, which includes the text: **PyPI to Spack package.py**

Generating Python

```
$ ./src/package.py generate --clean spack_requirements.txt
```

- Had to rework version system to handle some frequent python conventions
 - Prereleases
 - Alpha, beta
 - Release candidate
- Now PyPI metadata can map to Spack `depends_on()` constraints
- Some caveats
 - Only doing this for pure python packages
 - Wheels (and PyPI database) lack build dependency info
 - May not be able to use `spack develop` with these

```
from spack.package import *

class PyBlack(PythonPackage):
    version("24.3.0", sha256="a0c9c4a0771afc6919578cec71ce82a3e31e054904e7197deacbc9382671c4")
    version("24.2.0", sha256="bce4f25c27c3435e4dace4815bcb2008b87e167e3bf4ee47ccdc5ce906eb4f")
    version("24.1.1", sha256="48b5760dcbfe5cf97fd4fba23946681f3a81514c6ab8a45b50da67ac8fbc6c")
    version("24.1.0", sha256="30fbf768cd4f4576598b1db0202413fafea9a227ef808d1a12230c643cefe9")
    version("23.12.1", sha256="4ce3ef14ebe8d9509188014d96af1c456a910d5b5cbf434a09fef7e024b3c")
    version("23.12.0", sha256="330a327b422aca0634ecd115985c1c7fd7bdb5b5a2ef8aa9888a82e2ebe94")
    version("23.11.0", sha256="4c68855825ff432d197229846f971bc4d6666ce90492e5b02013bcaca4d9e")
    version("23.10.1", sha256="1f8ce316753428ff68749c65a5f7844631aa18c8679dfd3ca9dc1a289979c")
    version("23.10.0", sha256="31b9f87b277a68d0e99d2905edae08807c007973eaa609da5f0c62def6b7c")
    version("23.9.1", sha256="24b6b3ff5c6d9ea08a8888f6977eae858e1f340d7260cf56d70a49823236b6")

    variant("colorama", default=False)
    variant("d", default=False)
    variant("jupyter", default=False)
    variant("uvloop", default=False)

    with default_args(type="run"):
        depends_on("py-aiohhttp@3.7.4:", when="@23.12:23.12.0,24:24.1.0 platform=linux")
        depends_on("py-aiohhttp@3.7.4:", when="@23.12:23.12.0,24:24.1.0 platform=freebsd")
        depends_on("py-aiohhttp@3.7.4:", when="@23.12:23.12.0,24:24.1.0 platform=darwin")
        depends_on("py-aiohhttp@3.7.4:", when="@23.12:23.12.0,24:24.1.0 platform=cray")
        depends_on("py-aiohhttp@3.7.4:", when="@21.10-beta0:21,22.10:+d")
        depends_on("py-click@8.0.0:", when="@22.10:")
        depends_on("py-colorama@0.4.3:", when="@20:21,22.10:+colorama")
        depends_on("py-ipython@7.8:", when="@21.8-beta0:21,22.10:+jupyter")
        depends_on("py-mypy-extensions@0.4.3:", when="@20:21,22.10:")
        depends_on("py-packaging@22:", when="@23.1.0:")
        depends_on("py-pathspect@0.9:", when="@22.10:")
        depends_on("py-platformdirs@2.0.0:", when="@21.8-beta0:21,22.10:")
        depends_on("py-tokenize-rt@3.2:", when="@21.8-beta0:21,22.10:+jupyter")
        depends_on("py-tomli@1.1:", when="@22.10: ^python@:3.10")
        depends_on("py-typing-extensions@4.0.1:", when="@23.9: ^python@:3.10")
        depends_on("py-uvloop@0.15.2:", when="@21.5-beta2:21,22.10:+uvloop")
```

Summary

- HPSF is kicking off in May at ISC
 - Spack, other projects now in HPSF
 - Part of Linux Foundation
- Split build dependency work has enabled compiler dependencies
- Compiler dependencies coming in v0.22
 - Compiler runtime libraries like libstdc++, fortran
 - Libc now in Spack
- Python package generation!



Disclaimer

This document was prepared as an account of work sponsored by an agency of the United States government. Neither the United States government nor Lawrence Livermore National Security, LLC, nor any of their employees makes any warranty, expressed or implied, or assumes any legal liability or responsibility for the accuracy, completeness, or usefulness of any information, apparatus, product, or process disclosed, or represents that its use would not infringe privately owned rights. Reference herein to any specific commercial product, process, or service by trade name, trademark, manufacturer, or otherwise does not necessarily constitute or imply its endorsement, recommendation, or favoring by the United States government or Lawrence Livermore National Security, LLC. The views and opinions of authors expressed herein do not necessarily state or reflect those of the United States government or Lawrence Livermore National Security, LLC, and shall not be used for advertising or product endorsement purposes.