Some updates on binary cache handling in Spack

Todd Gamblin
Livermore Computing

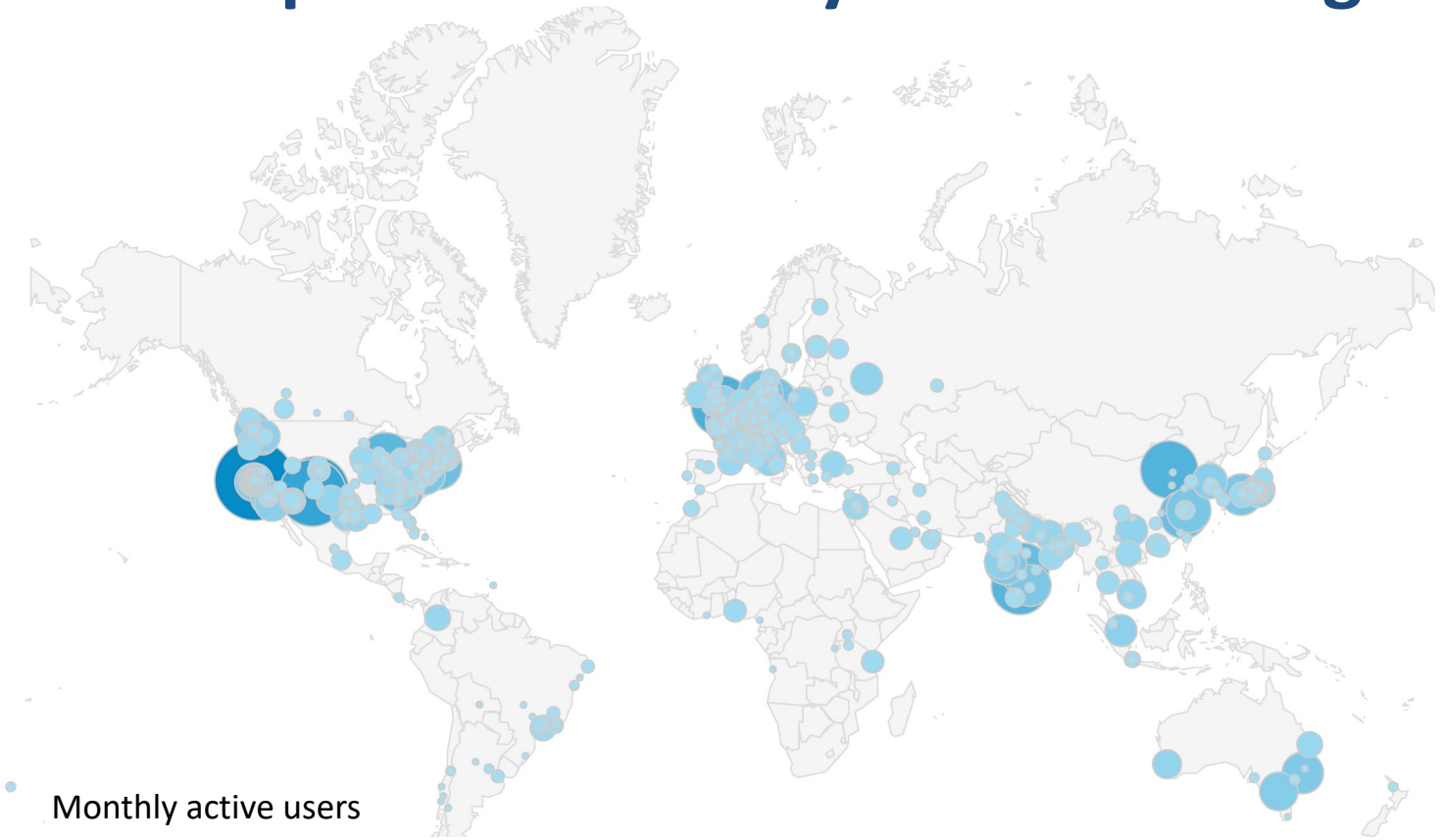
9th annual MVAPICH User Group Meeting (MUG'21)
August 25, 2021

LLNL-PRES-811108
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

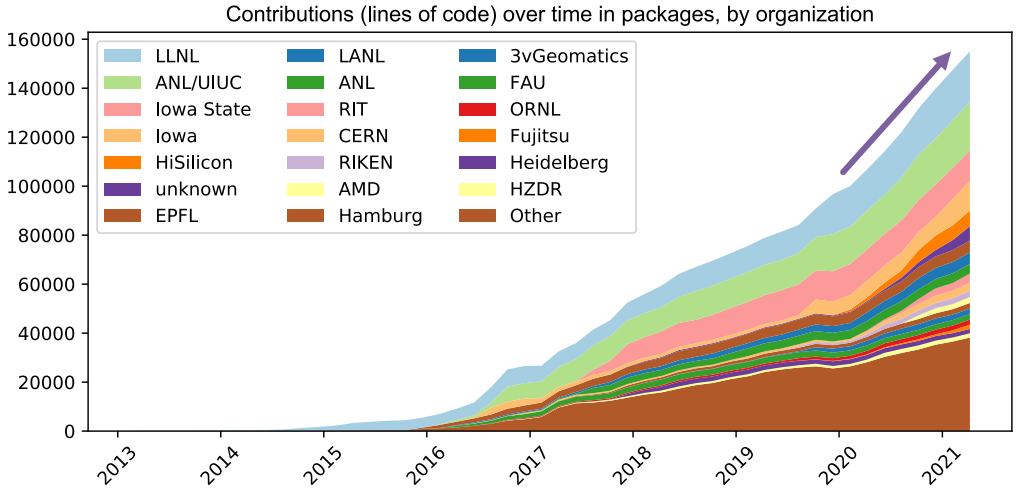
Lawrence Livermore National Laboratory

The Spack community continues to grow!

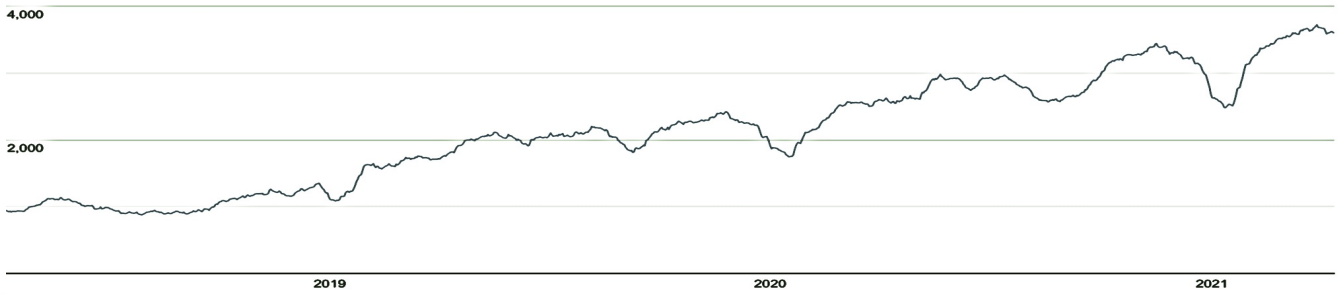
5,600+ software packages
820+ contributors



Package contribution rate increased in the past year

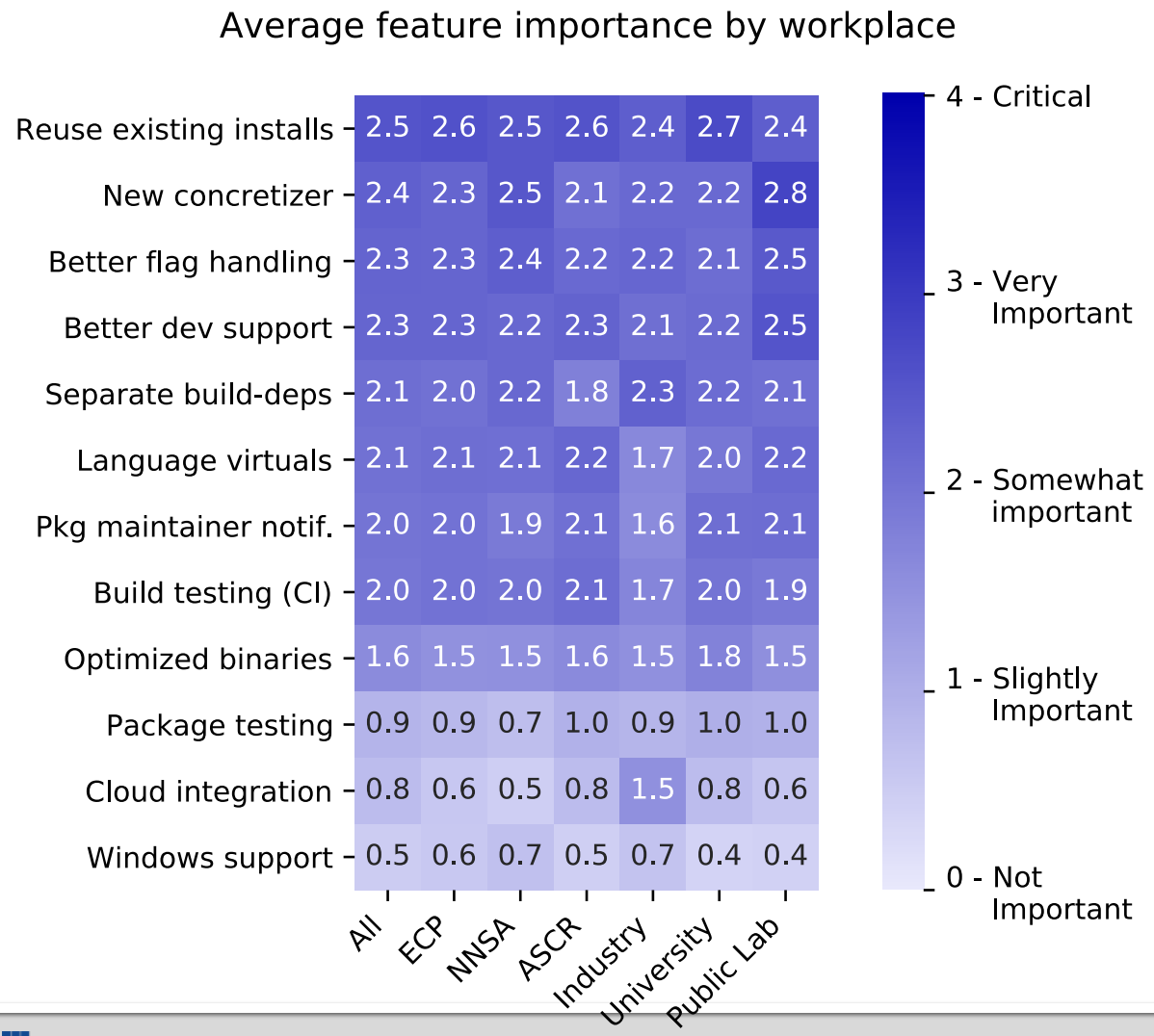


Monthly active users



All time high of 3,700 monthly active users this March

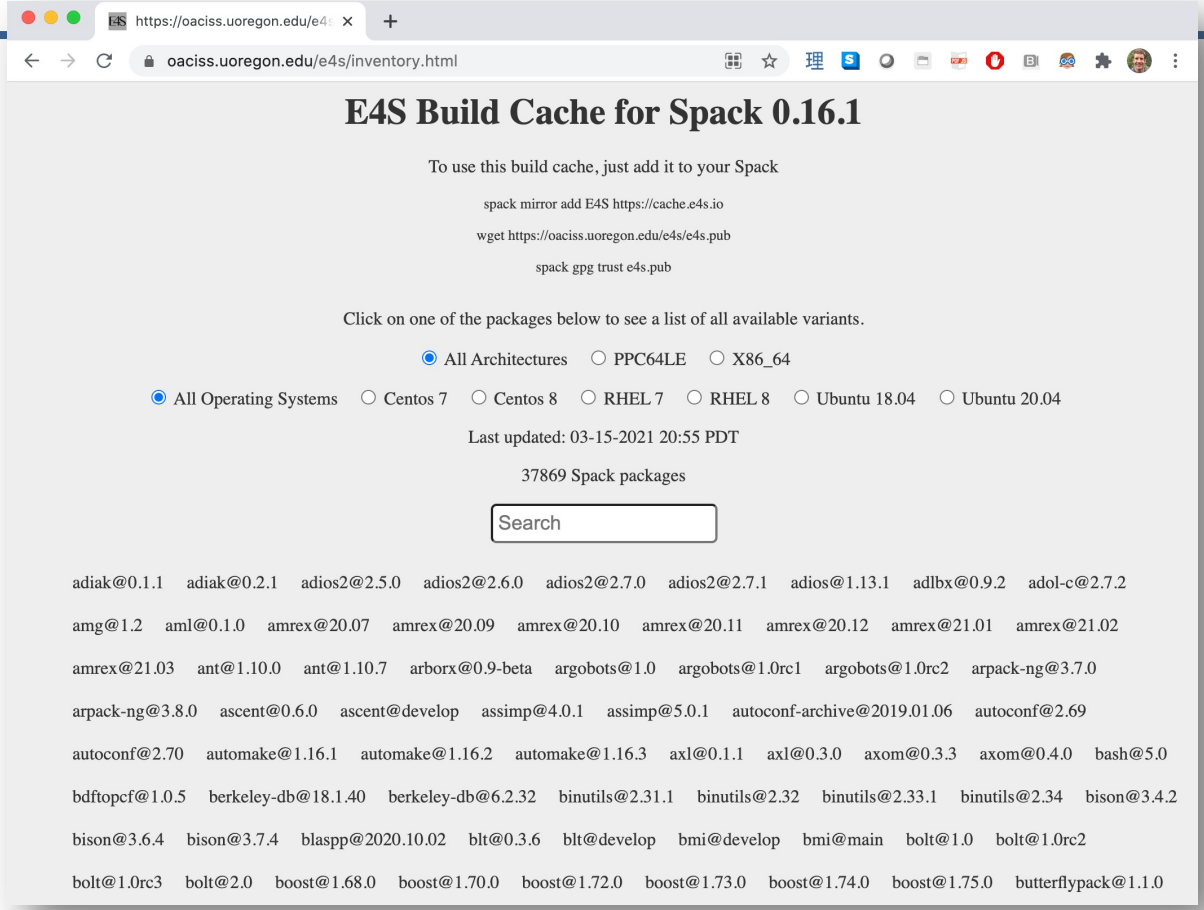
Four of the top six most wanted features in Spack are tied to the new concretizer



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

E4S team has built a binary cache with over 37,000 Spack binary packages

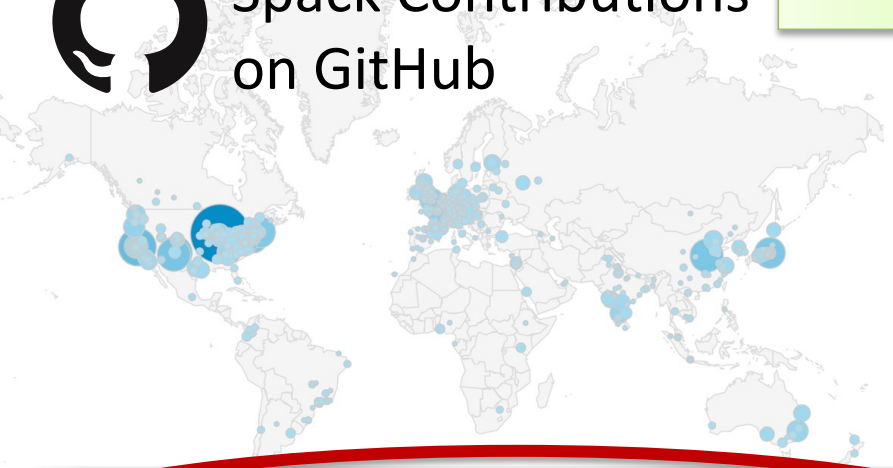
- Built for multiple OS's, architectures
- E4S team is working with ECP projects to accelerate their build pipelines
- Improved performance of cloud CI for one project by 10-100x
  - Previously, builds took too long for free cloud CI
  - Project can now iterate faster using Spack/E4S binaries



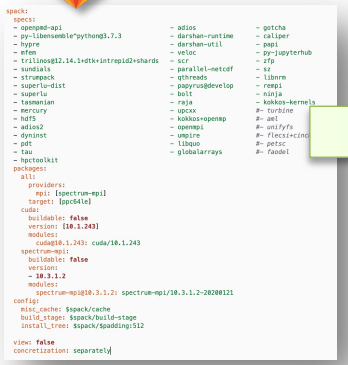
https://e4s.io (click on "build cache")

We are expanding our CI builds to include every pull request!

Spack Contributions on GitHub

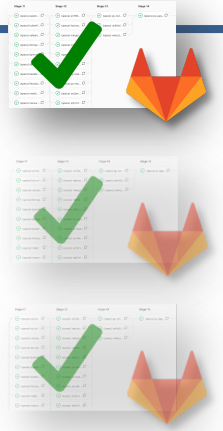


gitlab.spack.io



E4S `spack.yaml` configuration

`spack ci`



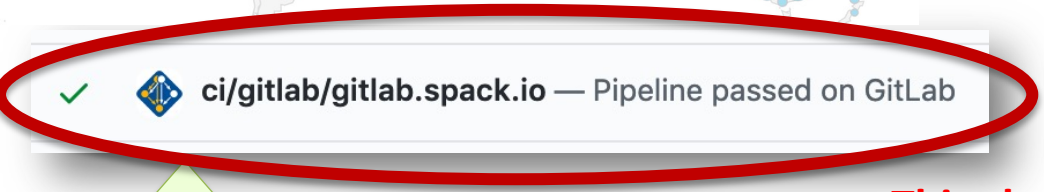
Pipelines running in **AWS**

Pipelines at **LLNL** **(in progress)**

Pipelines at **U. Oregon** **(in progress)**

GitLab CI builds (changed) packages

- On every pull request
- On every release branch
- Different compilers (Intel soon!)

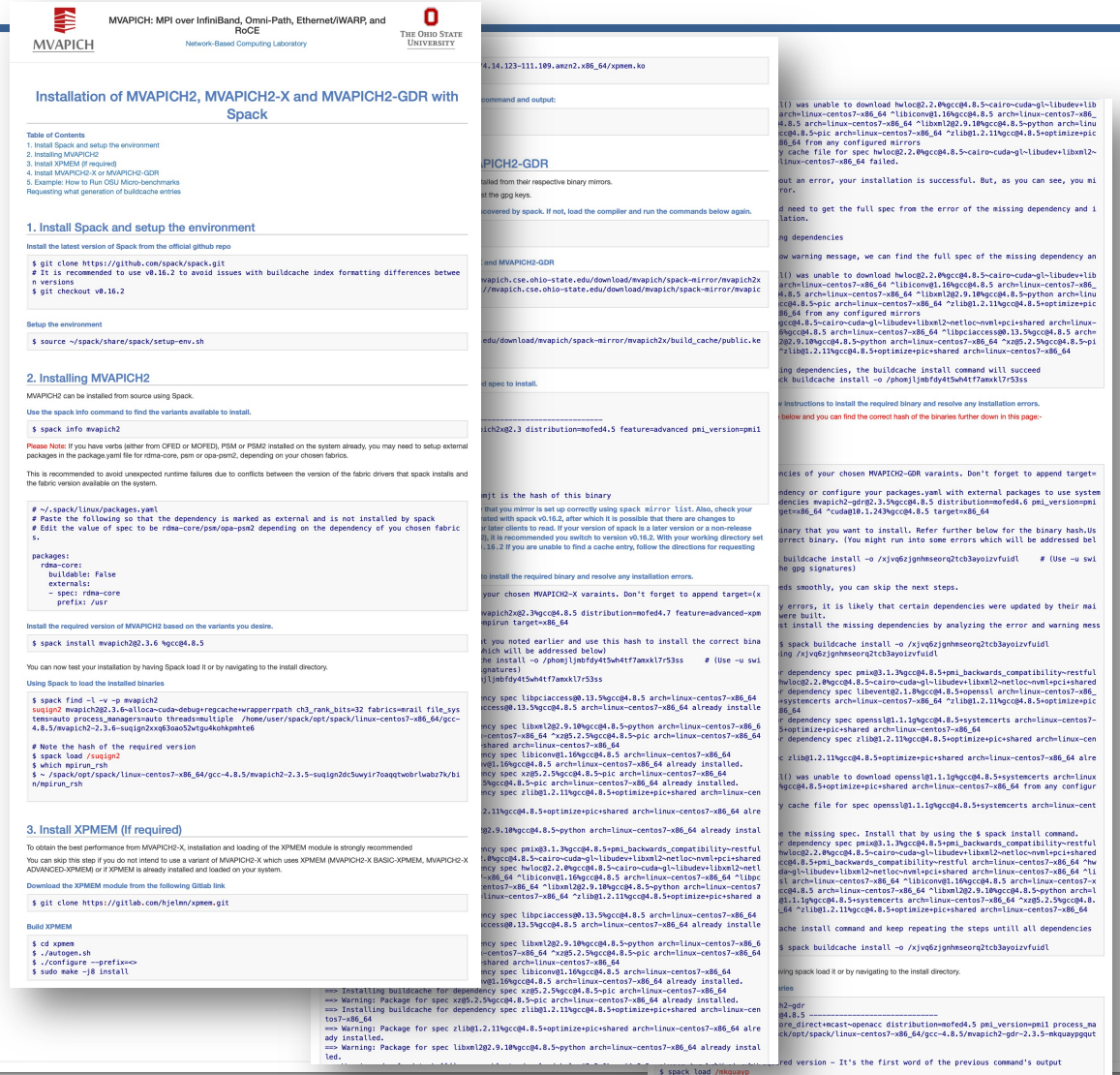


This check is (finally) required!

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

Instructions for building MVAPICH2-GDR from build caches are… baroque to say the least 😬

- Clone spack
- Register externals
- Add mirror
- `wget` and trust key
  - Can now be replaced with:
    `spack buildcache keys --install –trust`
- Install buildcaches by hash, **per dependency**
  - And deal with weird errors if you encounter them
- Eventually get the right mvapich2-gdr package installed from the build cache



MVAPICH: MPI over InfiniBand, Omni-Path, Ethernet/iWARP, and RoCE
Network-Based Computing Laboratory

Installation of MVAPICH2, MVAPICH2-X and MVAPICH2-GDR with Spack

Table of Contents
1. Install Spack and setup the environment
2. Installing MVAPICH2
3. Install XPMEM (if required)
4. Install MVAPICH2-X or MVAPICH2-GDR
5. Example: How to Run OSU Micro-benchmarks
Requesting what generation of buildcache entries

1. Install Spack and setup the environment

Install the latest version of Spack from the official github repo

```
$ git clone https://github.com/spack/spack.git
# It is recommended to use v0.16.2 to avoid issues with buildcache index formatting differences betwee
n versions
$ git checkout v0.16.2
```

Setup the environment

```
$ source ~/spack/share/spack/setup-env.sh
```

2. Installing MVAPICH2

MVAPICH2 can be installed from source using Spack.

Use the spack info command to find the variants available to install.

```
$ spack info mvapich2
```

Please Note: If you have verbs (either from OFED or MOFED), PSM or PSM2 installed on the system already, you may need to setup external packages in the package.yaml file for rdma-core, psm or opa-psm2, depending on your chosen fabrics.

This is recommended to avoid unexpected runtime failures due to conflicts between the version of the fabric drivers that spack installs and the fabric version available on the system.

```
# ~/.spack/linux/packages.yaml
# Paste the following so that the dependency is marked as external and is not installed by spack
# Edit the value of spec to be rdma-core/psm/opa-psm2 depending on the dependency of you chosen fabric
s.

packages:
  rdma-core:
    buildable: False
    externals:
    - spec: rdma-core
      prefix: /usr
```

Install the required version of MVAPICH2 based on the variants you desire.

```
$ spack install mvapich2@2.3.6 %gcc@4.8.5
```

You can now test your installation by having Spack load it or by navigating to the install directory.

Using Spack to load the installed binaries

```
$ spack find -l -v -p mvapich2
suqign2 mvapich2@2.3.6=alloca-cuda-debug+regcache+wrapperrpath ch3_rank_bits=32 fabrics=mrail file_sys
tems=auto process_managers=auto threads=multiple  /home/user/spack/opt/spack/linux-centos7-x86_64/gcc-
4.8.5/mvapich2-2.3.6-suqign2xxq63oao52wtgu4kohkpmhte6

# Note the hash of the required version
$ spack load /suqign2
$ which mpirun_rsh
$ ~ /spack/opt/spack/linux-centos7-x86_64/gcc-4.8.5/mvapich2-2.3.5-suqign2dc5uwyir7oaqqtwobrlwabz7k/bi
n/mpirun_rsh
```

3. Install XPMEM (If required)

To obtain the best performance from MVAPICH2-X, installation and loading of the XPMEM module is strongly recommended

You can skip this step if you do not intend to use a variant of MVAPICH2-X which uses XPMEM (MVAPICH2-X BASIC-XPMEM, MVAPICH2-X ADVANCED-XPMEM) or if XPMEM is already installed and loaded on your system.

Download the XPMEM module from the following Gitlab link

```
$ git clone https://gitlab.com/hjelmn/xpmem.git
```

Build XPMEM

```
$ cd xpmem
$ ./autogen.sh
$ ./configure --prefix=<>
$ sudo make -j8 install
```

`spack external find` will help with external registration (added in 0.15, significantly updated in 0.16)

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

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

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

Logic for finding external installations in `package.py`

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

`packages.yaml` configuration

- Spack has had compiler detection for a while
  - Finds compilers in your PATH
  - Registers them for use
- We can find many packages now
  - Package defines:
    - possible command names
    - how to query the command
  - Spack searches for known commands and adds them to configuration
- Community can easily enable tools to be set up rapidly

The new concretizer arrived in Spack v0.16

- Leverages **Clingo** (see https://potassco.org)
- Clingo is an Answer Set Programming (ASP) solver
  - ASP looks like Prolog; leverages SAT solvers for speed/correctness
  - ASP program has 2 parts:
    1. Large list of facts generated from our package repositories and config
       - 30,000 facts is typical – includes dependencies, options, etc.
    2. Small logic program (~700 lines), including constraints and optimization criteria
- New algorithm on the Spack side is conceptually simpler:
  - Generate facts for all possible dependencies, send to logic program
  - Optimization criteria express preferences more clearly
  - Build a DAG from the results
- New concretizer solves many specs that current concretizer can't
  - Backtracking is a huge win – many issues resolved
  - Currently requires user to install clingo with Spack
  - Solver will be automatically installed from public binaries in 0.17.0

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

The new concretizer enables significant simplifications to packages, particularly complex constraints in SDKs

- Dependencies and other constraints within SDKs could get very messy
- The new concretizer removes the need for some of the more painful constructs
- Also allows for new constructs, like specializing dependencies
  - When conditions are now much more general
  - Can be solved together with other constraints.

In some cases we needed cross-products of dependency options:

Before

```
depends_on('foo+A+B', when='+a+b')
depends_on('foo+A~B', when='+a~b')
depends_on('foo~A+B', when='~a+b')
depends_on('foo~A~B', when='~a~b')
```

After

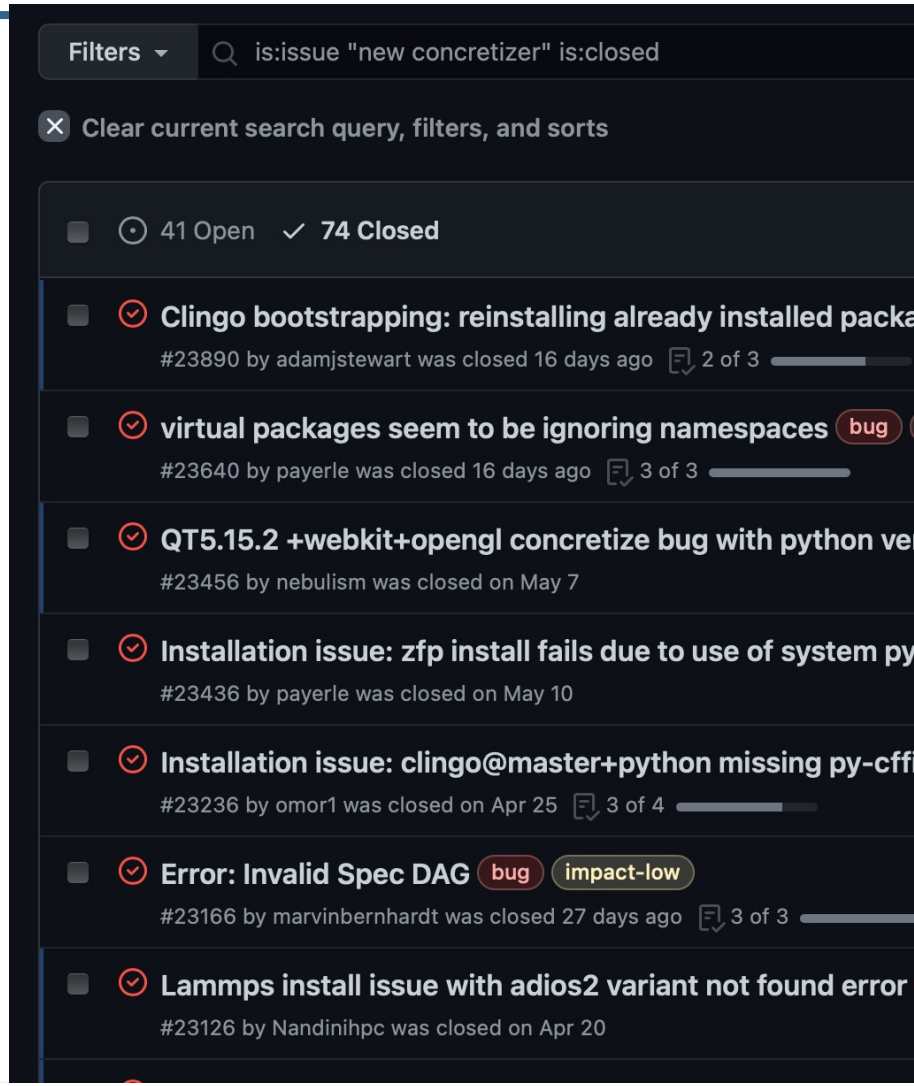
```
depends_on('foo')
depends_on('foo+A', when='+a')
depends_on('foo+B', when='+b’)
```

Specializing a virtual did not previously work:

```
depends_on('blas’)
depends_on(
    'openblas threads=openmp’, when='^openblas’
)
```

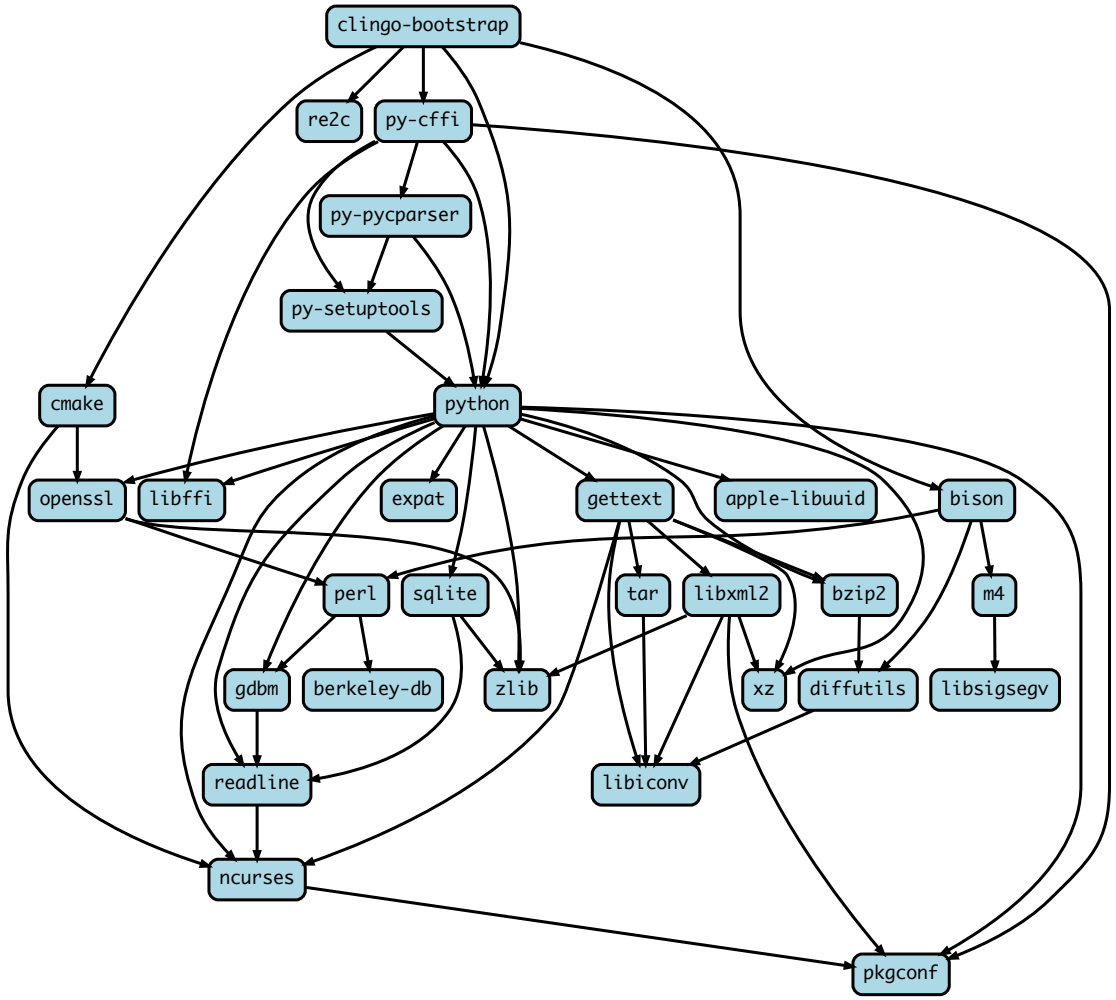
People seem to like the new concretizer a lot!

- Resolution to many issues is now “try the new concretizer”
- More expressive constraints in packages
- People building large stacks especially find that:
  - Can concretize “together” without false conflicts
  - Easier to supply constraints on the CLI
  - Easier to specify constraints in stacks
- Still quite a few open issues
  - Can iterate much more rapidly on concretization issues
  - Changes to support new criteria are generally small
  - Getting optimization order right is tricky



Clingo is a C++ library with a python interface

- This means Spack now has a very central, non-python dependency (Clingo)
- Concretization is the first thing you have to do to install anything
  - But now the concretizer depends on Clingo!
- Clingo itself has some dependencies:
  - py-cffi (build/run)
  - python (build/run)
  - cmake (build)
  - re2c (build)
  - bison (build)
- Most of complexity is in build dependencies
- Also requires C++14 to build



Step 1: just make the new concretizer available (v0.16.0)

- Ensure that clingo is installed in the Python you use to run Spack
- Add this to config.yaml:

```
config:
   concretizer: clingo   # default is “original”
```

- Fails if clingo isn’t actually importable from python
- Requires user to deal with installing clingo

Step 2: Bootstrap automatically from source (now in v0.16.2)

- Same config change
- No need to manually install clingo
  - Spack bootstraps a spec using the old concretizer
  - Builds the spec in a special “Bootstrap store”
  - Uses regular old Spack packages to achieve this
- Upside: fully automatic
- Downsides:
  - Time consuming
  - Build may fail (e.g., with an old C++ compiler)
  - Building from source is slow
  - High cost (in time) to pay for every Spack installation

```
config:
  concretizer: clingo
```

Step 3: Bootstrap from binaries (done on develop, will be in v0.17.0 release)

- We can now build Clingo binaries on GitHub
- GitHub actions workflow uses `manylinux` approach developed for Python to build very compatible binaries
  - Build in Centos5 image for old glibc
  - For each of macos, inux and for each Python version
- Spack uses these binaries by default
  - Fast -- no more building clingo from source!
  - Users can still opt out and bootstrap from source if they want.

Public GitHub Actions workflow for building bootstrap binaries

Hash caching

- Hashes in Spack are created by concretization process
  - Spack as originally designed doesn't remember hashes unless they're installed
- Build caches are registries of hashes
  - Currently Spack can show you what's in a build cache
  - It doesn't remember what the hashes *mean*
- Same for spack spec and concretized environments
  - Spack forgets hashes!
- We're implementing a hash cache so that you can be specific about things in build caches
  - Remember every hash we've ever shown
  - Allow users to refer to them on install
- This will allow you to install specific mvapich—gdr hashes without all the pain

```
(spackle):spack> spack spec -l hdf5
Input spec
--------------------------------
hdf5

Concretized
--------------------------------
igdg32d  hdf5@1.10.7%apple-clang@12.0.5~cxx~fortran~hl~ipo~java+mp
zd4m26e      ^cmake@3.21.1%apple-clang@12.0.5~doc+ncurses+openssl-
53i52xr          ^ncurses@6.2%apple-clang@12.0.5~symlinks+termlib
4tfrmuu              ^pkgconf@1.8.0%apple-clang@12.0.5 arch=darwin
us36bwr          ^openssl@1.1.1l%apple-clang@12.0.5~docs+systemcer
qbkmtdd              ^perl@5.34.0%apple-clang@12.0.5+cpanm+shared-
tnvkifs                  ^berkeley-db@18.1.40%apple-clang@12.0.5+c
7d5woqt                  ^bzip2@1.0.8%apple-clang@12.0.5~debug~pic
xc23j75                      ^diffutils@3.7%apple-clang@12.0.5 arc
k7auat3                          ^libiconv@1.16%apple-clang@12.0.5
vh6di3i                  ^gdbm@1.19%apple-clang@12.0.5 arch=darwin
qgy3v4l                      ^readline@8.1%apple-clang@12.0.5 arch
74mwnxg                  ^zlib@1.2.11%apple-clang@12.0.5+optimize-
kmoelef      ^openmpi@4.1.1%apple-clang@12.0.5~atomics~cuda~cxx~cx
2gtvyze          ^hwloc@2.5.0%apple-clang@12.0.5~cairo~cuda~gl~lib
ckdn5zf              ^libxml2@2.9.12%apple-clang@12.0.5~python arc
k2yumgx                  ^xz@5.2.5%apple-clang@12.0.5~pic libs=sha
5noysxq          ^libevent@2.1.12%apple-clang@12.0.5+openssl arch=
63xbksk          ^openssh@8.6p1%apple-clang@12.0.5 arch=darwin-big
snhgldt              ^libedit@3.1-20210216%apple-clang@12.0.5 arch

(spackle):spack> spack spec /5noysxq
==> Error: No installed spec matches the hash: '5noysxq'
```

Very unsatisfying!

Final frontier: reusing concretizer

- New concretizer allows us to add arbitrary optimization criteria
  - We can tell it to maximize number of hashes included in a build
  - Or minimize number of builds vs. binary installs
- Need to tell the concretizer about hashes in advance
- This is thankfully quite simple in ASP!

Minimizing builds, maximizing hashes automatically

Add facts about installed packages:

```
installed_hash("openssl","lwatuuysmwkhuahrncywvn77icdhs6mn").
imposed_constraint("lwatuuysmwkhuahrncywvn77icdhs6mn","node","openssl").
imposed_constraint("lwatuuysmwkhuahrncywvn77icdhs6mn","version","openssl","1.1.1g").
imposed_constraint("lwatuuysmwkhuahrncywvn77icdhs6mn","node_platform","openssl","darwin").
imposed_constraint("lwatuuysmwkhuahrncywvn77icdhs6mn","node_os","openssl","catalina").
imposed_constraint("lwatuuysmwkhuahrncywvn77icdhs6mn","node_target","openssl","x86_64").
imposed_constraint("lwatuuysmwkhuahrncywvn77icdhs6mn","node_compiler","openssl","apple-clang").
imposed_constraint("lwatuuysmwkhuahrncywvn77icdhs6mn","node_compiler_version","openssl","apple-clang","12.0.0").
imposed_constraint("lwatuuysmwkhuahrncywvn77icdhs6mn","hash","openssl","lwatuuysmwkhuahrncywvn77icdhs6mn").
imposed_constraint("lwatuuysmwkhuahrncywvn77icdhs6mn","depends_on","openssl","zlib","build").
imposed_constraint("lwatuuysmwkhuahrncywvn77icdhs6mn","depends_on","openssl","zlib","link").
imposed_constraint("lwatuuysmwkhuahrncywvn77icdhs6mn","hash","zlib","x2anksgssxsxa7pcnhzg5k3dhgacglze").
```

Tell concretizer to minimize builds by choosing hashes:

```
% the solver is free to choose at most one installed hash for each package in the solution
{ hash(Package, Hash) : installed_hash(Package, Hash) } 1 :- node(Package).

% if a hash is selected, we impose all of its constraints
impose(Hash) :- hash(Package, Hash).

% if we haven't selected a hash for a package, and if it is not external, we build it
build(Package) :- not hash(Package, _), not external(Package), node(Package).
#minimize { 1@18,Package : build(Package) }.
```

Reuse coming soon in 0.17!

Before

```
(spackle):spack> spack solve -Il llvm
==> Best of 7 considered solutions.
==> Optimization Criteria:
  Priority  Criterion                                  Value
  1         number of packages to build (vs. reuse)     16
  2         deprecated versions used                     0
  3         version weight                               9
  4         number of non-default variants (roots)       1
  5         multi-valued variants                       -1
  6         preferred providers for roots                0
  7         number of non-default variants (non-roots)  10
  8         preferred providers (non-roots)             20
  9         compiler mismatches                          0
  10        os mismatches                                0
  11        version badness                              9
  12        count of non-root multi-valued variants      0
  13        non-preferred compilers                      0
  14        target mismatches                            0
  15        non-preferred targets                        0

 -   2p6rpiv  llvm@8.0.1%apple-clang@12.0.0~all_targets+clang~code_signing+compiler-rt~cuda~flang~gold+inte
 -   rs5nwwn      ^cmake@3.20.5%apple-clang@12.0.0~doc+ncurses~openssl+ownlibs~qt build_type=Release arch=d
 -   xrklbl5          ^ncurses@6.2%apple-clang@12.0.0~symlinks+termlib abi=none arch=darwin-bigsur-skylake
 -   ztt3pgs              ^pkgconf@1.7.4%apple-clang@12.0.0 arch=darwin-bigsur-skylake
 -   hjw2gx7      ^hwloc@2.5.0%apple-clang@12.0.0~cairo~cuda~gl~libudev~libxml2~netloc~nvml~pci+shared arch
 -   zipu7bi      ^perl-data-dumper@2.173%apple-clang@12.0.0 arch=darwin-bigsur-skylake
 -   raet62c          ^perl@5.34.0%apple-clang@12.0.0+cpanm+shared+threads arch=darwin-bigsur-skylake
 -   akjl2di              ^berkeley-db@18.1.40%apple-clang@12.0.0+cxx~docs+stl patches=b231fcc4d5cff05e5c3a
 -   l7gigul              ^bzip2@1.0.8%apple-clang@12.0.0~debug~pic+shared arch=darwin-bigsur-skylake
 -   vazus7p                  ^diffutils@3.7%apple-clang@12.0.0 arch=darwin-bigsur-skylake
 -   vtlbzpn                      ^libiconv@1.16%apple-clang@12.0.0 arch=darwin-bigsur-skylake
 -   xkdly6s              ^gdbm@1.19%apple-clang@12.0.0 arch=darwin-bigsur-skylake
 -   gmhw2jt                  ^readline@8.1%apple-clang@12.0.0 arch=darwin-bigsur-skylake
[+]  xflsoui              ^zlib@1.2.11%apple-clang@12.0.0+optimize+pic+shared arch=darwin-bigsur-skylake
 -   inxzdsq      ^python@3.9.6%apple-clang@12.0.0+bz2~ctypes+dbm~debug~libxml2~lzma~nis~optimizations+pic~
 -   m6ef5no          ^apple-libuuid@1353.100.2%apple-clang@12.0.0 arch=darwin-bigsur-skylake
 -   otyemkg          ^gettext@0.21%apple-clang@12.0.0+bzip2+curses+git~libunistring~libxml2~tar~xz arch=da
```

After

```
(spackle):spack> spack solve -Il --reuse llvm
==> Best of 7 considered solutions.
==> Optimization Criteria:
  Priority  Criterion                                  Value
  1         number of packages to build (vs. reuse)      5
  2         deprecated versions used                     2
  3         version weight                               0
  4         number of non-default variants (roots)       1
  5         multi-valued variants                       -1
  6         preferred providers for roots                0
  7         number of non-default variants (non-roots)   5
  8         preferred providers (non-roots)             10
  9         compiler mismatches                          0
  10        os mismatches                                8
  11        version badness                             51
  12        count of non-root multi-valued variants     -5
  13        non-preferred compilers                      0
  14        target mismatches                            0
  15        non-preferred targets                      253

 -   ifi254  llvm@12.0.1%apple-clang@12.0.0~all_targets+clang~code_signing+compiler-rt~cuda~flang~gold+in
 -   xkcrrd      ^cmake@3.20.5%apple-clang@12.0.0~doc+ncurses+openssl+ownlibs~qt build_type=Release arch=
[+]  ei4aax          ^ncurses@6.2%apple-clang@12.0.0~symlinks+termlib arch=darwin-catalina-x86_64
[+]  lvatuuy          ^openssl@1.1.1g%apple-clang@12.0.0+systemcerts arch=darwin-catalina-x86_64
[+]  xanksg              ^zlib@1.2.11%apple-clang@12.0.0+optimize+pic+shared arch=darwin-catalina-x86_64
 -   o2m54j      ^hwloc@2.5.0%apple-clang@12.0.0~cairo~cuda~gl~libudev+libxml2~netloc~nvml~pci+shared arc
[+]  dzlvh6          ^libxml2@2.9.10%apple-clang@12.0.0~python arch=darwin-catalina-x86_64
[+]  xjlxcv              ^libiconv@1.16%apple-clang@12.0.0 arch=darwin-catalina-x86_64
[+]  ga5h76              ^xz@5.2.5%apple-clang@12.0.0~pic arch=darwin-catalina-x86_64
 -   oeef2w          ^pkgconf@1.7.4%apple-clang@12.0.0 arch=darwin-bigsur-x86_64
 -   tk3evu      ^perl-data-dumper@2.173%apple-clang@12.0.0 arch=darwin-bigsur-x86_64
[+]  jgcc6o          ^perl@5.30.3%apple-clang@12.0.0+cpanm+shared+threads arch=darwin-catalina-x86_64
[+]  d3hxmr              ^gdbm@1.18.1%apple-clang@12.0.0 arch=darwin-catalina-x86_64
[+]  43hjqd                  ^readline@8.0%apple-clang@12.0.0 arch=darwin-catalina-x86_64
[+]  2lf5pr      ^python@3.7.8%apple-clang@12.0.0+bz2+ctypes+dbm~debug+libxml2+lzma~nis~optimizations+pic
[+]  6taixw          ^bzip2@1.0.8%apple-clang@12.0.0+shared arch=darwin-catalina-x86_64
[+]  2dnmfp          ^expat@2.2.9%apple-clang@12.0.0~libbsd arch=darwin-catalina-x86_64
[+]  glf2co          ^gettext@0.20.2%apple-clang@12.0.0+bzip2+curses+git~libunistring+libxml2+tar+xz arch
[+]  d675hk              ^tar@1.32%apple-clang@12.0.0 arch=darwin-catalina-x86_64
[+]  5e4c7r          ^libffi@3.3%apple-clang@12.0.0 patches=26f26c6f29a7ce9bf370ad3ab2610f99365b4bdd7b82e
[+]  nrcpot          ^sqlite@3.31.1%apple-clang@12.0.0+column_metadata+fts~functions~rtree arch=darwin-ca
[+]  d6nj46      ^z3@4.8.7%apple-clang@12.0.0+python arch=darwin-catalina-x86_64
[+]  mlsl7o          ^py-setuptools@49.2.0%apple-clang@12.0.0 arch=darwin-catalina-x86_64
```

Lawrence Livermore National Laboratory