github.com/spack @spackpm

Building things: Spack, Software, and Sustainable Communities in HPC

ATPESC: Argonne Training Program on Extreme-Scale Computing

August 2, 2018
Q Center, St. Charles, IL

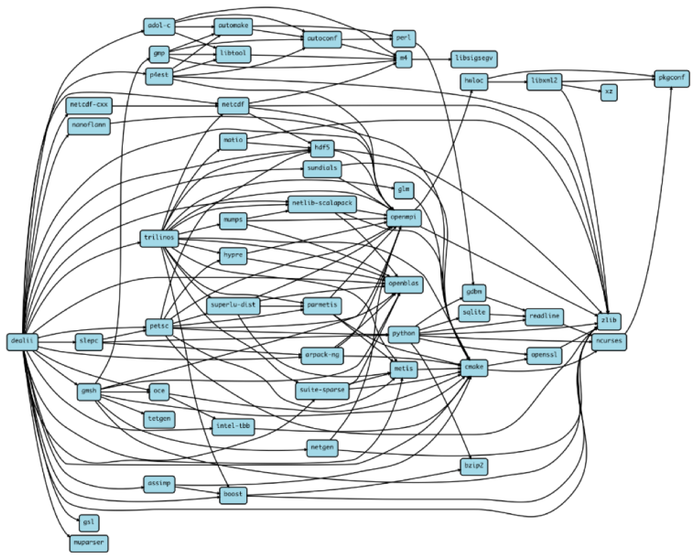
Todd Gamblin
Computer Scientist

LLNL-PRES-747560
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

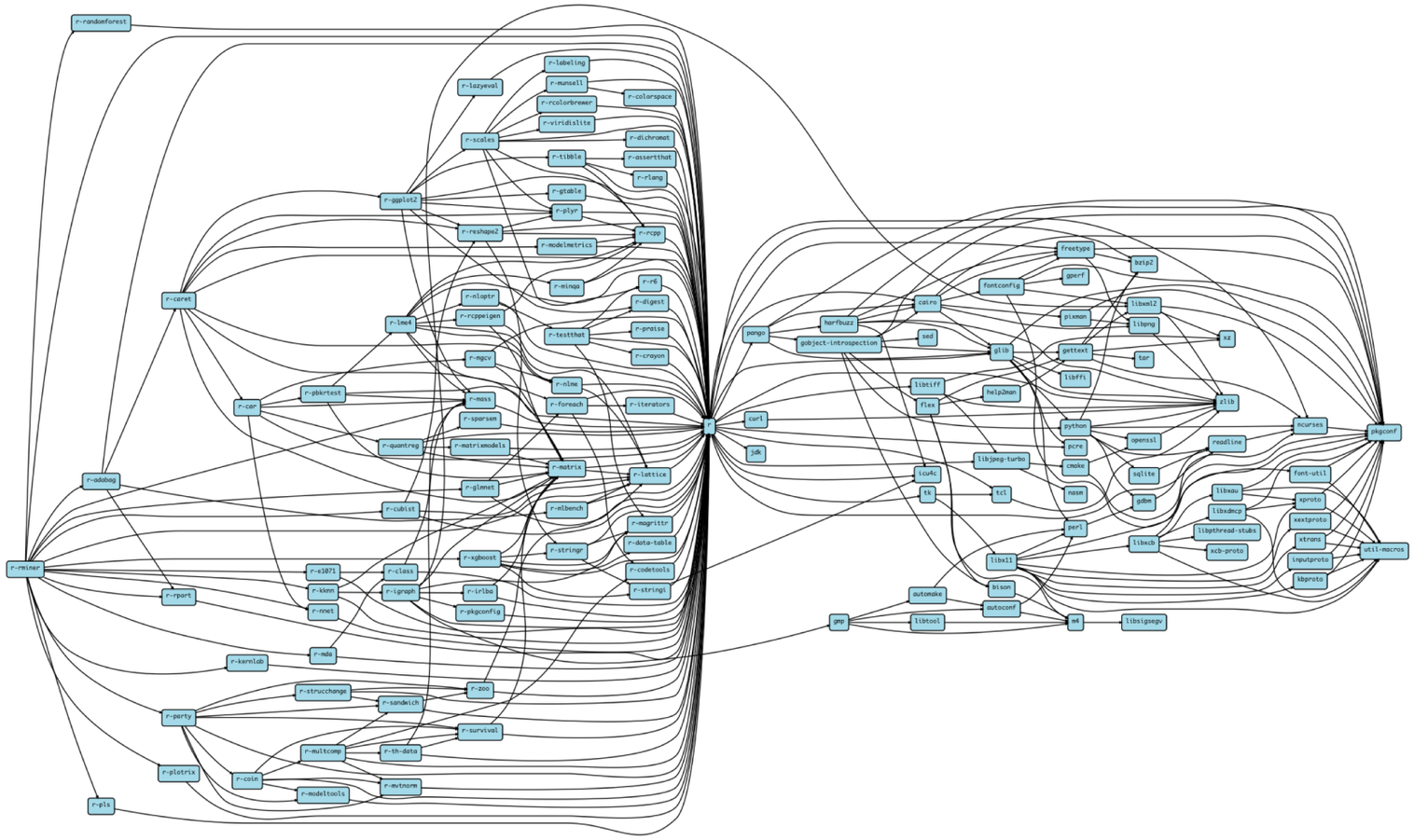
Lawrence Livermore National Laboratory

Scientific software is becoming extremely complex

Nalu: Generalized Unstructured Massively Parallel Low Mach Flow

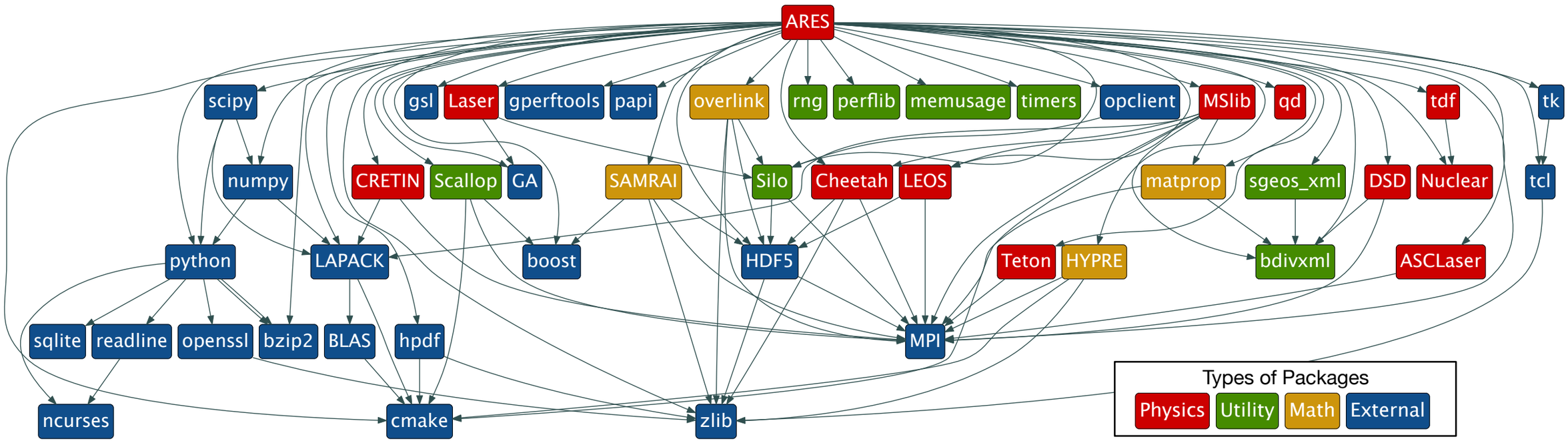


dealii: C++ Finite Element Library



R Miner: R Data Mining Library

Even proprietary codes are based on many open source libraries



- Half of this DAG is external (blue); *more* than half of it is open source
- Nearly *all* of it needs to be built specially for HPC to get the best performance

The Exascale Computing Project is building an entire *ecosystem*

15+ applications x **80+ software packages** x **5+ target architectures/platforms**
Xeon Power KNL
NVIDIA ARM Laptops?

x **Up to 7 compilers**
Intel GCC Clang XL
PGI Cray NAG

x **10+ Programming Models**
OpenMPI MPICH MVAPICH OpenMP CUDA
OpenACC Dharma Legion RAJA Kokkos

x **2-3 versions of each package + external dependencies**

= up to **1,260,000** combinations!

- Every application has its own stack of dependencies.
- Developers, users, and facilities dedicate (many) FTEs to building & porting.
- Often trade reuse and usability for performance.

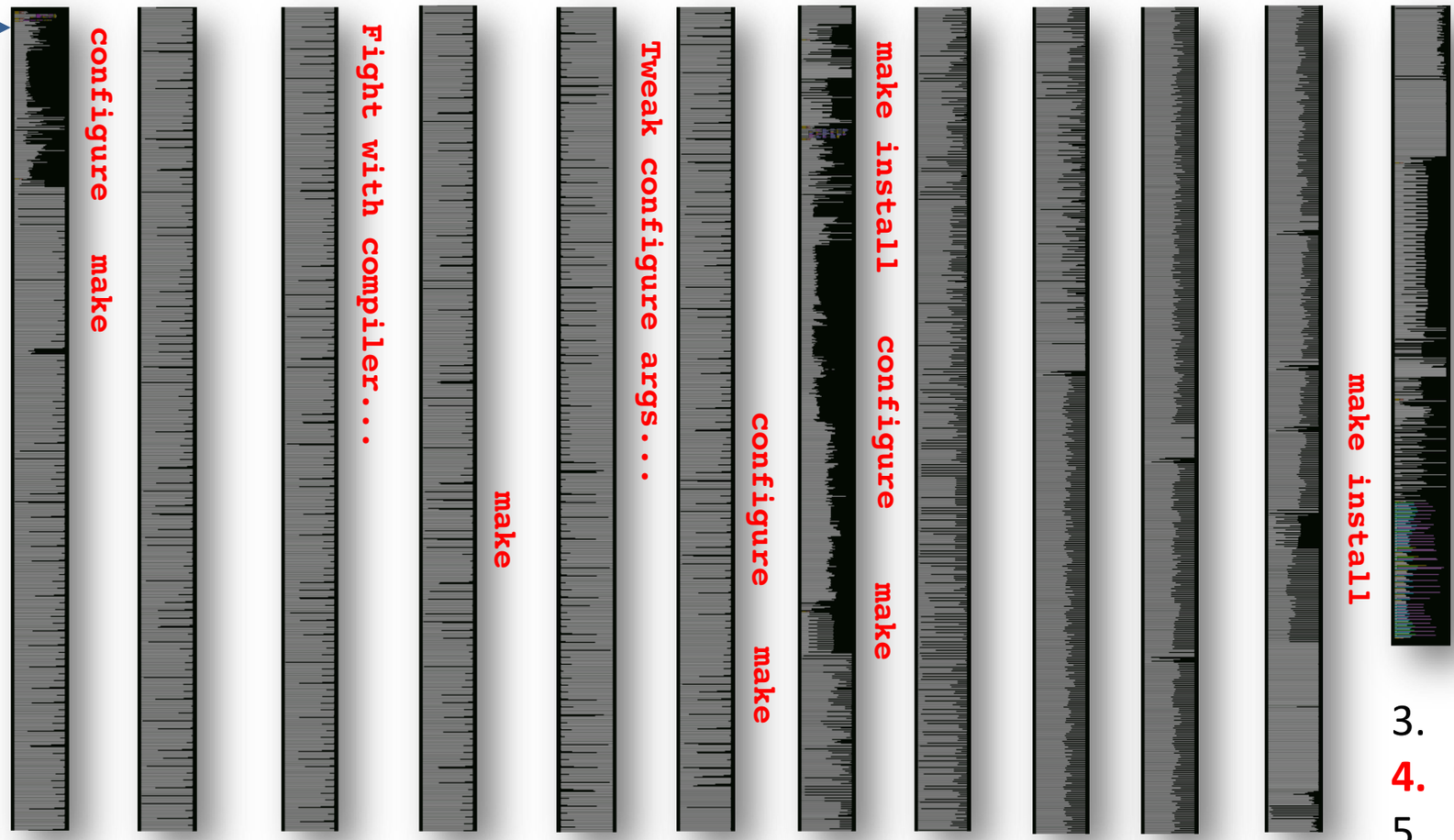
We must make it easier to rely on others' software!

How to install software on a Mac laptop, circa 2013

```
(gluon):~$ port install libelf
--->  Computing dependencies for libelf
--->  Fetching distfiles for libelf
--->  Verifying checksum(s) for libelf
--->  Extracting libelf
--->  Applying patches to libelf
--->  Configuring libelf
--->  Building libelf
--->  Staging libelf into destroot
--->  Installing libelf @0.8.13_2
--->  Activating libelf @0.8.13_2
--->  Cleaning libelf
--->  Updating database of binaries: 100.0%
--->  Scanning binaries for linking errors: 100.0%
--->  No broken files found.
(gluon):~$
```

How to install software on a supercomputer, circa 2013

1. Download all 16 tarballs you need
2. Start building!



3. Run code
4. **Segfault!?**
5. Start over…

What about modules?

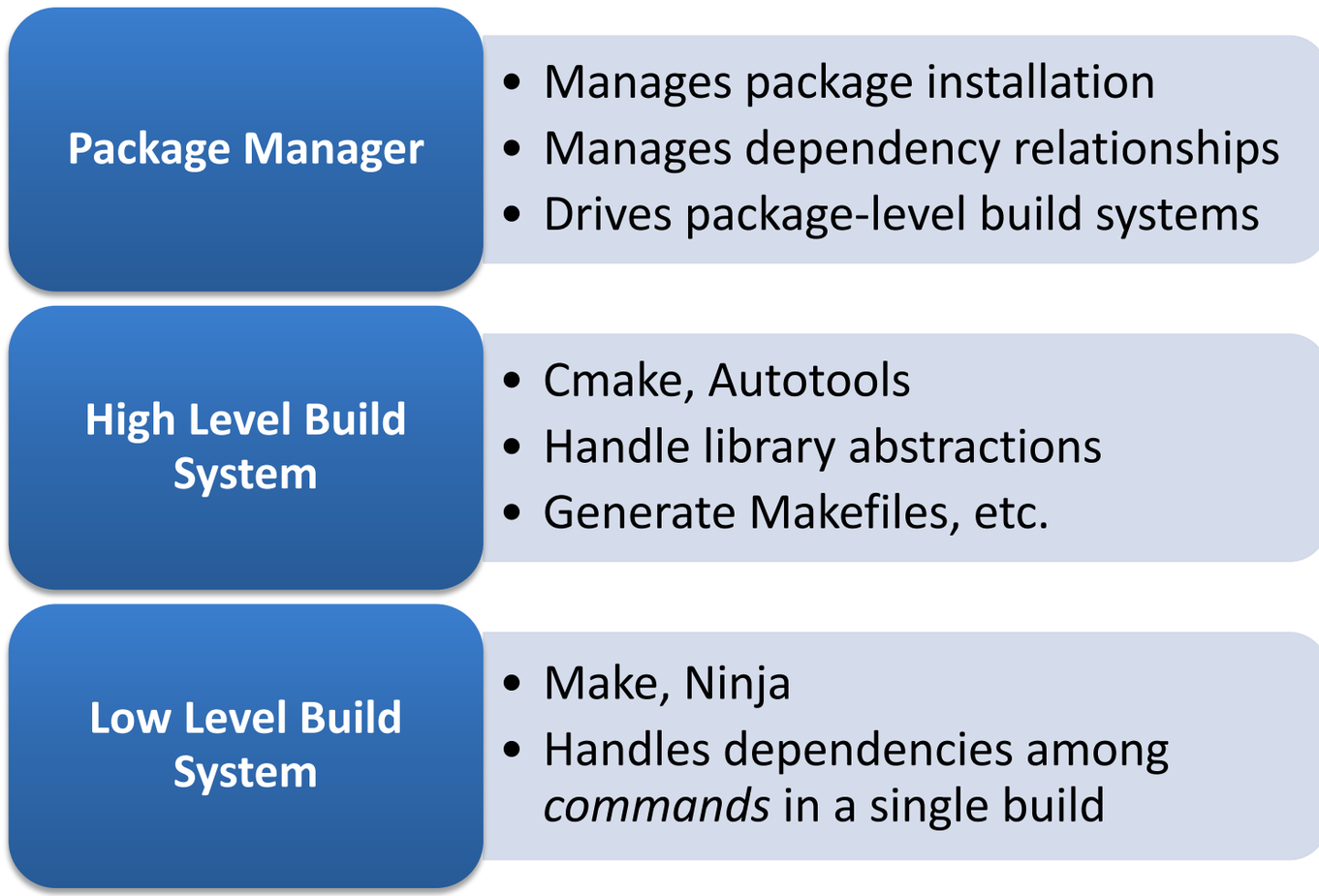
- Most supercomputers deploy some form of *environment modules*
  - TCL modules (dates back to 1995) and Lmod (from TACC) are the most popular

```
$ gcc
-bash: gcc: command not found

$ module load gcc/7.0.1
$ gcc –dumpversion
7.0.1
```

- Modules don't handle installation!
  - They only modify your environment (things like PATH, LD_LIBRARY_PATH, etc.)
- Someone (likely a team of people) has already installed gcc for you!
  - Also, you can *only* `module load` the things they've installed

What's a package manager?

- ***Package manager***
  - **Does not** a replace Cmake/Autotools
  - Packages built by Spack can have any build system they want
- PMs manage ***dependencies***
  - Drive package-level build systems
  - Or installs pre-built binaries
  - Ensures consistent configuration
- Determining magic configure lines takes time
  - PMs cache the work of others
  - Provide a way to encode recipes so that you can reuse others' effort!



So why didn't these things catch on in HPC?

- Traditional binary package managers don't support combinatorial versioning
  - RPM, yum, apt, yast, etc.
  - Designed to manage a *single* stack.
  - Install *one* version of each package in a single prefix (`/usr`).
  - Seamless upgrades to a *stable, well tested* stack
- Neither, typically, do port systems
  - BSD Ports, portage, Macports, Homebrew, Gentoo portage, etc.
  - Minimal support for builds parameterized by compilers, dependency versions.
- Issues:
  - HPC people want to experiment
  - Many of these typically require root access, can't have root on a supercomputer
  - Binaries aren't optimized
    - typically built for lowest-common-denominator hardware
- System package managers mostly used for low level system, not the HPC stack

HPC needed a better way to build software

- My frustrations:
  1. Constantly rebuilding graduate students' software for them
  2. Facilities spend lots of time building a comparatively small number of software packages
     - Quickly goes out of date
     - Not built with the right compiler, MPI, dependency version, etc.
     - App teams end up rebuilding anyway!
  3. Hard to distribute performance tools!
     - My research was going unused because it was hard to install.

- Requirements for a good solution:
  - Target users, admins, and developers (many roles in HPC)
  - Easy to use commands, no tedious build steps
  - Easy to contribute: package recipes should be in a language that HPC people already know
  - Rapidly build many different versions of software, experiment with performance options

Spack is a flexible package manager for HPC

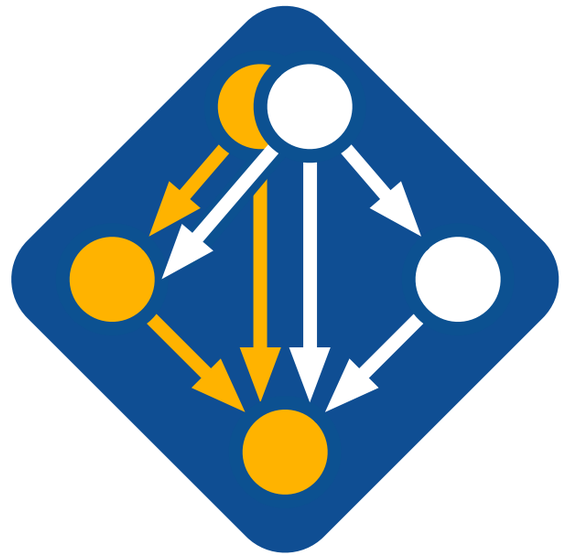
- How to install Spack (works out of the box):

```
$ git clone https://github.com/spack/spack
$ . spack/share/spack/setup-env.sh
```

- How to install a package:

```
$ spack install hdf5
```

- HDF5 and its dependencies are installed within the Spack directory.
- Unlike typical package managers, Spack can also install many variants of the same build.
  - Different compilers
  - Different MPI implementations
  - Different build options



Get Spack!

http://github.com/spack/spack

@spackpm

Spack provides the *spec* syntax to describe custom 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"   setting compiler flags
$ spack install mpileaks@3.3 os=CNL10 target=haswell  setting target for X-compile
$ spack install mpileaks@3.3 ^mpich@3.2 %gcc@4.9.3    ^ dependency information
```

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

`spack list` shows what packages are available

```
$ spack list
==> 303 packages.
activeharmony   cgal              fish          gtkplus       libgd          mesa            openmpi          py-coverage      py-pycparser      qt          tcl
adept-utils     cgm               flex          harfbuzz      libgpg-error   metis           openspeedshop    py-cython        py-pyelftools     qthreads    texinfo
apex            cityhash          fltk          hdf           libjpeg-turbo  Mitos           openssl          py-dateutil      py-pygments       R           the_silver_searcher
arpack          cleverleaf        flux          hdf5          libjson-c      mpc             otf              py-epydoc        py-pylint         ravel       thrift
asciidoc        cloog             fontconfig    hwloc         libmng         mpe2            otf2             py-funcsigs      py-pypar          readline    tk
atk             cmake             freetype      hypre         libmonitor     mpfr            pango            py-genders       py-pyparsing      rose        tmux
atlas           cmocka            gasnet        icu           libNBC         mpibash         papi             py-gnuplot       py-pyqt           rsync       tmuxinator
atop            coreutils         gcc           icu4c         libpciaccess   mpich           paraver          py-h5py          py-pyside         ruby        trilinos
autoconf        cppcheck          gdb           ImageMagick   libpng         mpileaks        paraview         py-ipython       py-pytables       SAMRAI      uncrustify
automaded       cram              gdk-pixbuf    isl           libsodium      mrnet           parmetis         py-libxml2       py-python-daemon  samtools    util-linux
automake        cscope            geos          jdk           libtiff        mumps           parpack          py-lockfile      py-pytz           scalasca    valgrind
bear            cube              gflags        jemalloc      libtool        munge           patchelf         py-mako          py-rpy2           scorep      vim
bib2xhtml       curl              ghostscript   jpeg          libunwind      muster          pcre             py-matplotlib    py-scientificpython scotch    vtk
binutils        czmq              git           judy          libuuid        mvapich2        pcre2            py-mock          py-scikit-learn   scr         wget
bison           damselfly         glib          julia         libxcb         nasm            pdt              py-mpi4py        py-scipy          silo        wx
boost           dbus              glm           launchmon     libxml2        ncdu            petsc            py-mx            py-setuptools     snappy      wxpropgrid
bowtie2         docbook-xml       global        lcms          libxshmfence   ncurses         pidx             py-mysqldb1      py-shiboken       sparsehash  xcb-proto
boxlib          doxygen           glog          leveldb       libxslt        netcdf          pixman           py-nose          py-sip            spindle     xerces-c
bzip2           dri2proto         glpk          libarchive    llvm           netgauge        pkg-config       py-numexpr       py-six            spot        xz
cairo           dtcmp             gmp           libcerf       llvm-lld       netlib-blas     pmgr_collective  py-numpy         py-sphinx         sqlite      yasm
callpath        dyninst           gmsh          libcircle     lmdb           netlib-lapack   postgresql       py-pandas        py-sympy          stat        zeromq
cblas           eigen             gnuplot       libdrm        lmod           netlib-scalapack ppl             py-pbr           py-tappy          sundials    zlib
cbtf            elfutils          gnutls        libdwarf      lua            nettle          protobuf         py-periodictable py-twisted        swig        zsh
cbtf-argonavis  elpa              gperf         libedit       lwgrp          ninja           py-astropy       py-pexpect       py-urwid          szip
cbtf-krell      expat             gperftools    libelf        lwm2           ompss           py-basemap       py-pil           py-virtualenv     tar
cbtf-lanl       extrae            graphlib      libevent      matio          ompt-openmp     py-biopython     py-pillow        py-yapf           task
cereal          exuberant-ctags   graphviz      libffi        mbedtls        opari2          py-blessings     py-pmw           python            taskd
cfitsio         fftw              gsl           libgcrypt     memaxes        openblas        py-cffi          py-pychecker     qhull             tau
```

- Spack has over 2,800 packages now.

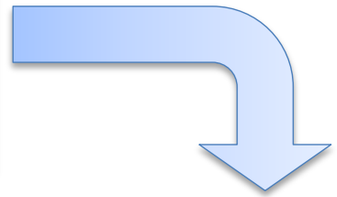
`spack find` shows what is installed

```
$ spack find
==> 103 installed packages.
-- linux-redhat6-x86_64 / gcc@4.4.7 -----------------------------
ImageMagick@6.8.9-10  glib@2.42.1        libtiff@4.0.3     pango@1.36.8          qt@4.8.6
SAMRAI@3.9.1          graphlib@2.0.0     libtool@2.4.2     parmetis@4.0.3        qt@5.4.0
adept-utils@1.0       gtkplus@2.24.25    libxcb@1.11       pixman@0.32.6         ravel@1.0.0
atk@2.14.0            harfbuzz@0.9.37    libxml2@2.9.2     py-dateutil@2.4.0     readline@6.3
boost@1.55.0          hdf5@1.8.13        llvm@3.0          py-ipython@2.3.1      scotch@6.0.3
cairo@1.14.0          icu@54.1           metis@5.1.0       py-nose@1.3.4         starpu@1.1.4
callpath@1.0.2        jpeg@9a            mpich@3.0.4       py-numpy@1.9.1        stat@2.1.0
dyninst@8.1.2         libdwarf@20130729  ncurses@5.9       py-pytz@2014.10       xz@5.2.0
dyninst@8.1.2         libelf@0.8.13      ocr@2015-02-16    py-setuptools@11.3.1  zlib@1.2.8
fontconfig@2.11.1     libffi@3.1         openssl@1.0.1h    py-six@1.9.0
freetype@2.5.3        libmng@2.0.2       otf@1.12.5salmon  python@2.7.8
gdk-pixbuf@2.31.2     libpng@1.6.16      otf2@1.4          qhull@1.0

-- linux-redhat6-x86_64 / gcc@4.8.2 -----------------------------
adept-utils@1.0.1  boost@1.55.0  cmake@5.6-special  libdwarf@20130729  mpich@3.0.4
adept-utils@1.0.1  cmake@5.6     dyninst@8.1.2      libelf@0.8.13      openmpi@1.8.2

-- linux-redhat6-x86_64 / intel@14.0.2 --------------------------
hwloc@1.9  mpich@3.0.4  starpu@1.1.4

-- linux-redhat6-x86_64 / intel@15.0.0 --------------------------
adept-utils@1.0.1  boost@1.55.0  libdwarf@20130729  libelf@0.8.13  mpich@3.0.4

-- linux-redhat6-x86_64 / intel@15.0.1 --------------------------
adept-utils@1.0.1  callpath@1.0.2  libdwarf@20130729  mpich@3.0.4
boost@1.55.0       hwloc@1.9       libelf@0.8.13      starpu@1.1.4
```

- All the versions coexist!
  - Multiple versions of same package are ok.
- Packages are installed to automatically find correct dependencies.
- Binaries work *regardless of user's environment*.
- Spack also generates module files.
  - Don't *have* to use them.

Users can query the full dependency configuration of installed packages.

```
$ spack find callpath
==> 2 installed packages.
-- linux-x86_64 / clang@3.4 --------     -- linux-x86_64 / gcc@4.9.2 ------------
callpath@1.0.2                           callpath@1.0.2
```



Expand dependencies with `spack find -d`

```
$ spack find -dl callpath
==> 2 installed packages.
-- linux-x86_64 / clang@3.4 -----------     -- linux-x86_64 / gcc@4.9.2 -----------
xv2clz2    callpath@1.0.2                    udltshs    callpath@1.0.2
ckjazss        ^adept-utils@1.0.1            rfsu7fb        ^adept-utils@1.0.1
3ws43m4            ^boost@1.59.0             ybet64y            ^boost@1.55.0
ft7znm6            ^mpich@3.1.4              aa4ar6i            ^mpich@3.1.4
qqnuet3        ^dyninst@8.2.1                tmnnge5        ^dyninst@8.2.1
3ws43m4            ^boost@1.59.0             ybet64y            ^boost@1.55.0
g65rdud            ^libdwarf@20130729        g2mxrl2            ^libdwarf@20130729
cj5p5fk                ^libelf@0.8.13        ynpai3j                ^libelf@0.8.13
cj5p5fk            ^libelf@0.8.13            ynpai3j            ^libelf@0.8.13
g65rdud        ^libdwarf@20130729            g2mxrl2        ^libdwarf@20130729
cj5p5fk            ^libelf@0.8.13            ynpai3j            ^libelf@0.8.13
cj5p5fk        ^libelf@0.8.13                ynpai3j        ^libelf@0.8.13
ft7znm6        ^mpich@3.1.4                  aa4ar6i        ^mpich@3.1.4
```

- Architecture, compiler, versions, and variants may differ between builds.

Spack packages are *templates*: they define how to build a spec

```
from spack import *

class Dyninst(Package):
    """API for dynamic binary instrumentation."""

    homepage = "https://paradyn.org"
    url = "http://www.paradyn.org/release8.1.2/DyninstAPI-8.1.2.tgz"

    version('8.2.1', 'abf60b7faabe7a2e')
    version('8.1.2', 'bf03b33375afa66f')
    version('8.1.1', 'd1a04e995b7aa709')

    depends_on("cmake", type="build")

    depends_on("libelf", type="link")
    depends_on("libdwarf", type="link")
    depends_on("boost @1.42: +multithreaded")

    def install(self, spec, prefix):
        with working_dir('spack-build', create=True):
            cmake('-DBoost_INCLUDE_DIR=' + spec['boost'].prefix.include,
                  '-DBoost_LIBRARY_DIR=' + spec['boost'].prefix.lib,
                  '-DBoost_NO_SYSTEM_PATHS=TRUE'
              '..')
            make()
            make("install")
```

Simple Python DSL

- Packages are classes (ala Homebrew)
- Directives use the same spec syntax

Metadata at the class level

Versions

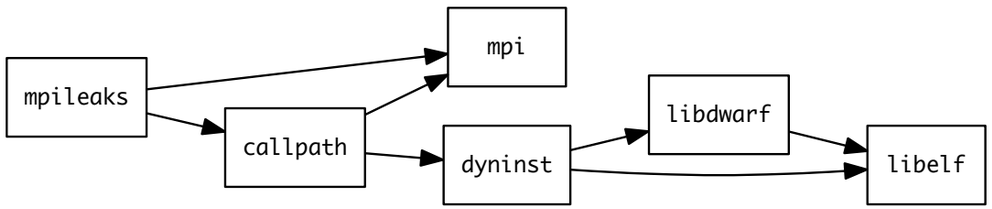
Dependencies

Patches, variants, resources, conflicts, etc. (not shown)

Install logic in instance methods

Spack handles combinatorial software complexity.

Dependency DAG



Installation Layout

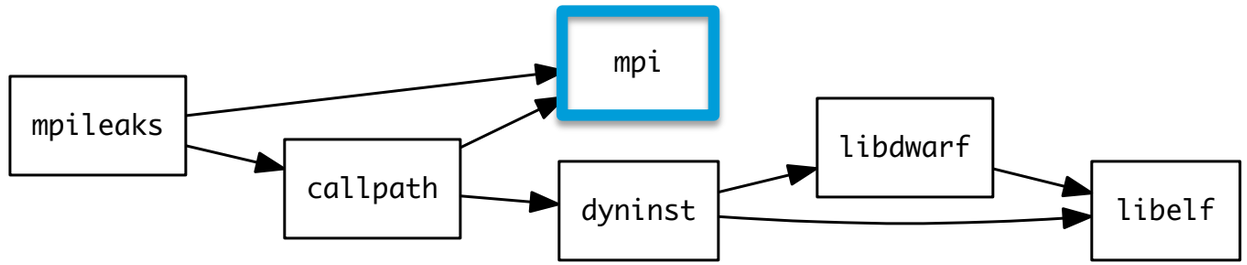
Hash

```
spack/opt/
  linux-x86_64/
    gcc-4.7.2/
      mpileaks-1.1-0f54bf34cadk/
    intel-14.1/
      hdf5-1.8.15-lkf14aq3nqiz/
  bgq/
    xl-12.1/
      hdf5-1-8.16-fqb3a15abrwx/
  ...
```

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

Depend on interfaces (not implementations) with virtual dependencies

- `mpi` is a *virtual dependency*
- Install the same package built with two different MPI implementations:




```
$ spack install mpileaks ^mvapich
```

```
$ spack install mpileaks ^openmpi@1.4:
```

- Virtual deps are replaced with a valid implementation at resolution time.
  - If the user didn't pick something and there are multiple options, Spack picks.

Virtual dependencies can be versioned:

```
class Mpileaks(Package):
    depends_on("mpi@2:")
```
dependent

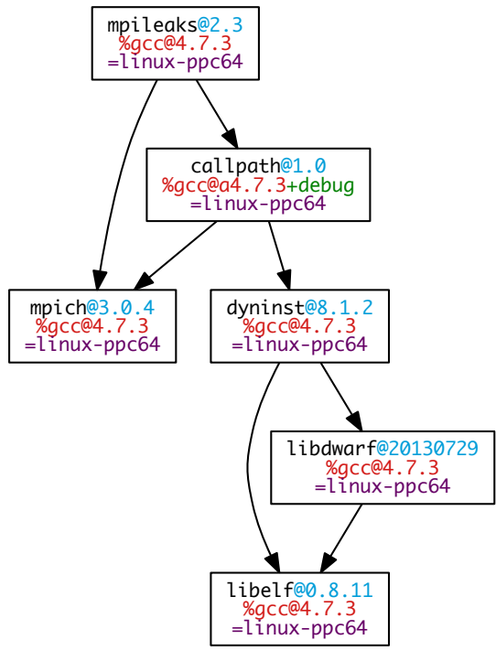
```
class Mvapich(Package):
    provides("mpi@1" when="@:1.8")
    provides("mpi@2" when="@1.9:")
```
provider

```
class Openmpi(Package):
    provides("mpi@:2.2" when="@1.6.5:")
```
provider

Concretization fills in missing parts of requested specs.

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

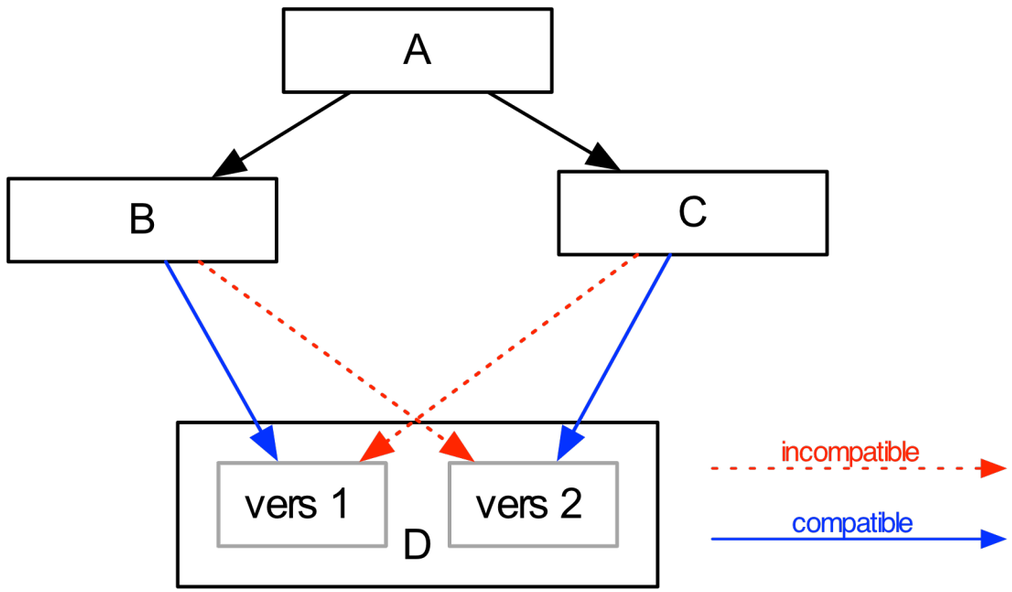
Concretize



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

- Workflow:
  1. Users input only an *abstract* spec with some constraints
  2. Spack makes choices according to policies (site/user/etc.)
  3. Spack installs *concrete* configurations of package + dependencies
- Dependency resolution is an NP-complete problem!
  - Different versions/configurations of packages require different versions/configurations of dependencies
  - Concretizer searches for a configuration that satisfies all the requirements
  - This is basically a SAT/SMT solve

Dependency Resolution is an NP-complete problem!

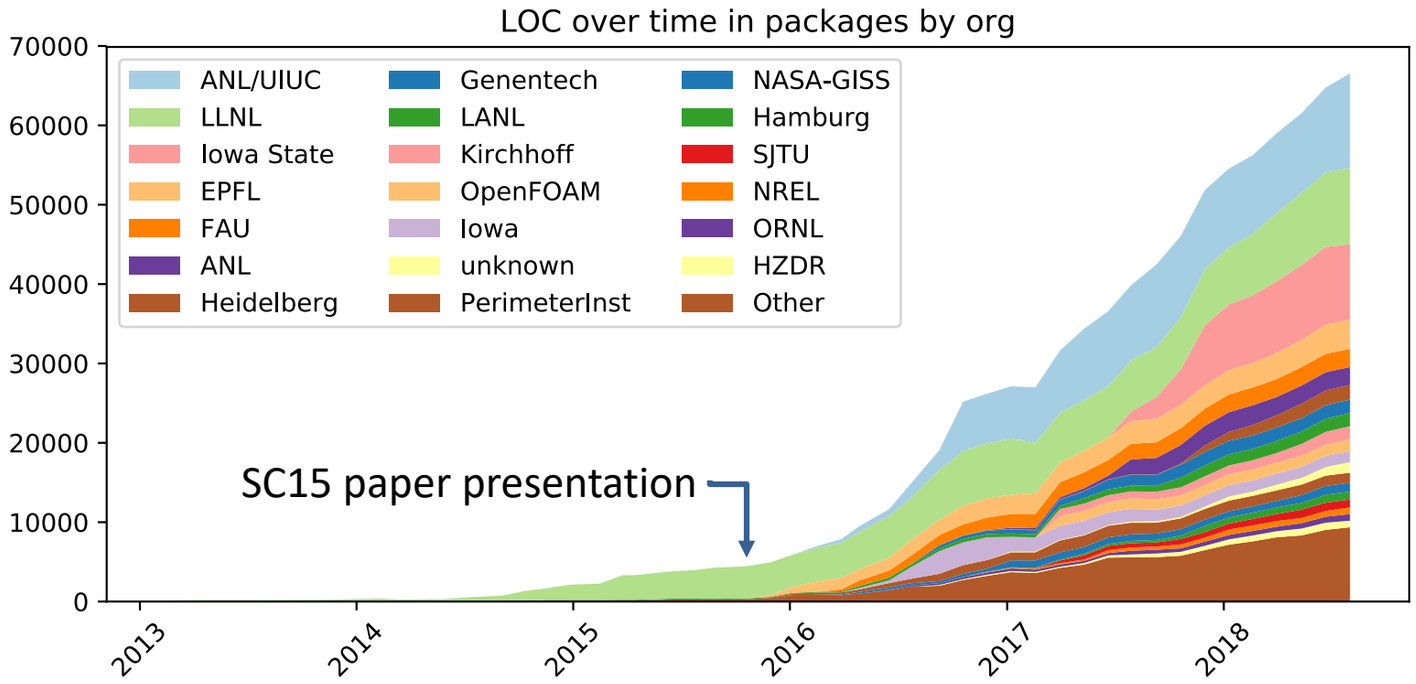


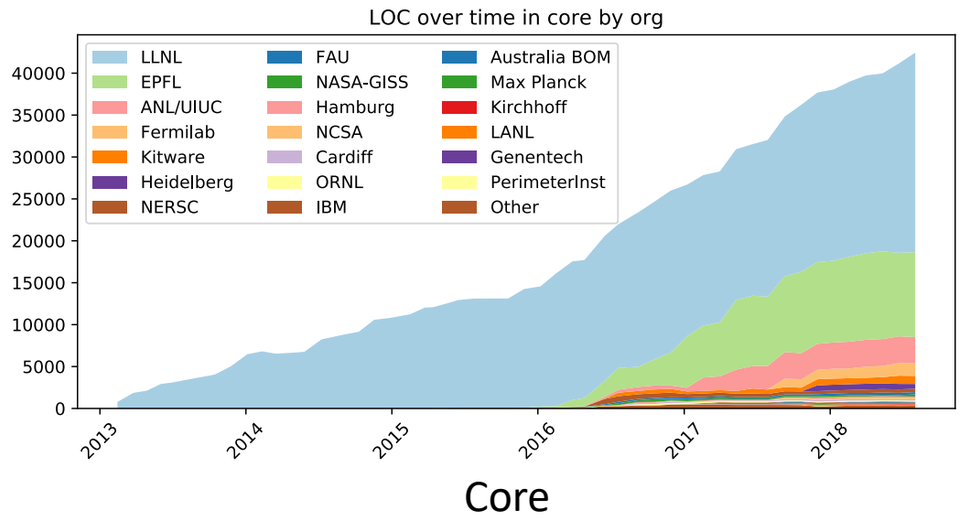
Unsatisfiable!

- Different versions of packages require different versions of dependencies
  - Concretizer searches for a configuration that satisfies all the requirements
  - Can show that SAT/SMT solve is equivalent problem
- Resolution is NP-complete for *just* package and version metadata
  - Concretization also includes compilers, variants, architecture, optional dependencies, virtual dependencies
  - We have some leeway because multiple stacks can coexist within Spack (unlike system PMs)
  - Even within one DAG there can be issues!

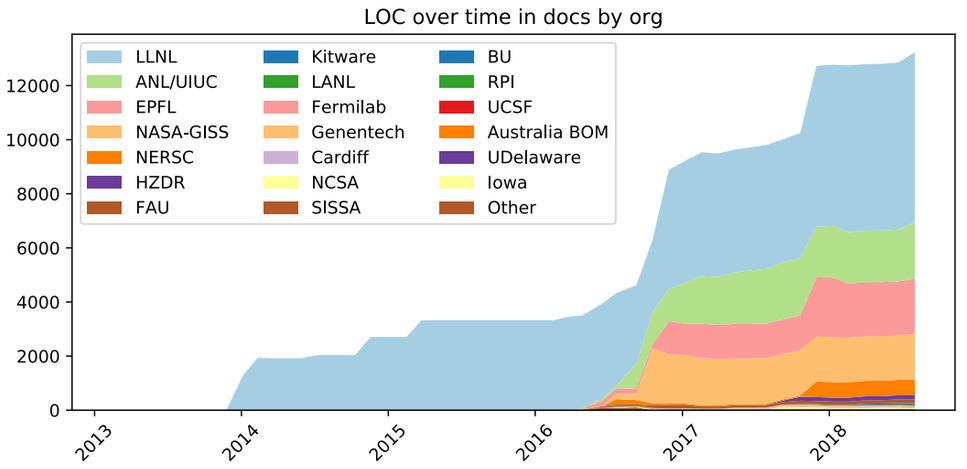
https://research.swtch.com/version-sat

Contributions to Spack have taken off





Core



Documentation

- 300 contributors on GitHub so far!
- Outside organizations make most of the package contributions, but all can leverage
- LLNL packages comprise < 20% of total

July 2, 2018 – August 2, 2018

Period: 1 month

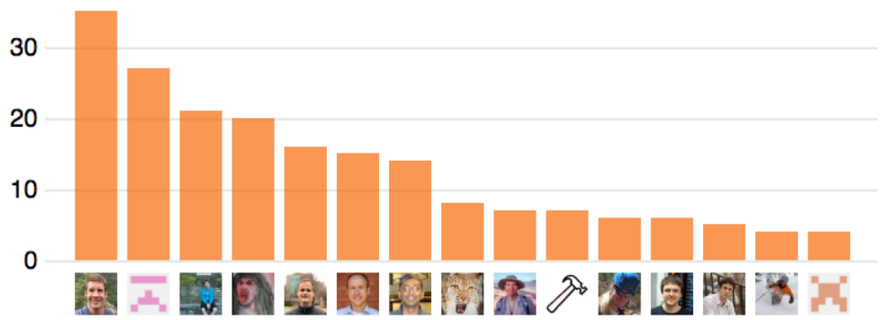
Overview

206 Active Pull Requests

98 Active Issues

161 Merged Pull Requests	**45** Proposed Pull Requests	**58** Closed Issues	**40** New Issues

Excluding merges, **55 authors** have pushed **202 commits** to develop and **251 commits** to all branches. On develop, **761 files** have changed and there have been **14,324 additions** and **4,332 deletions**.



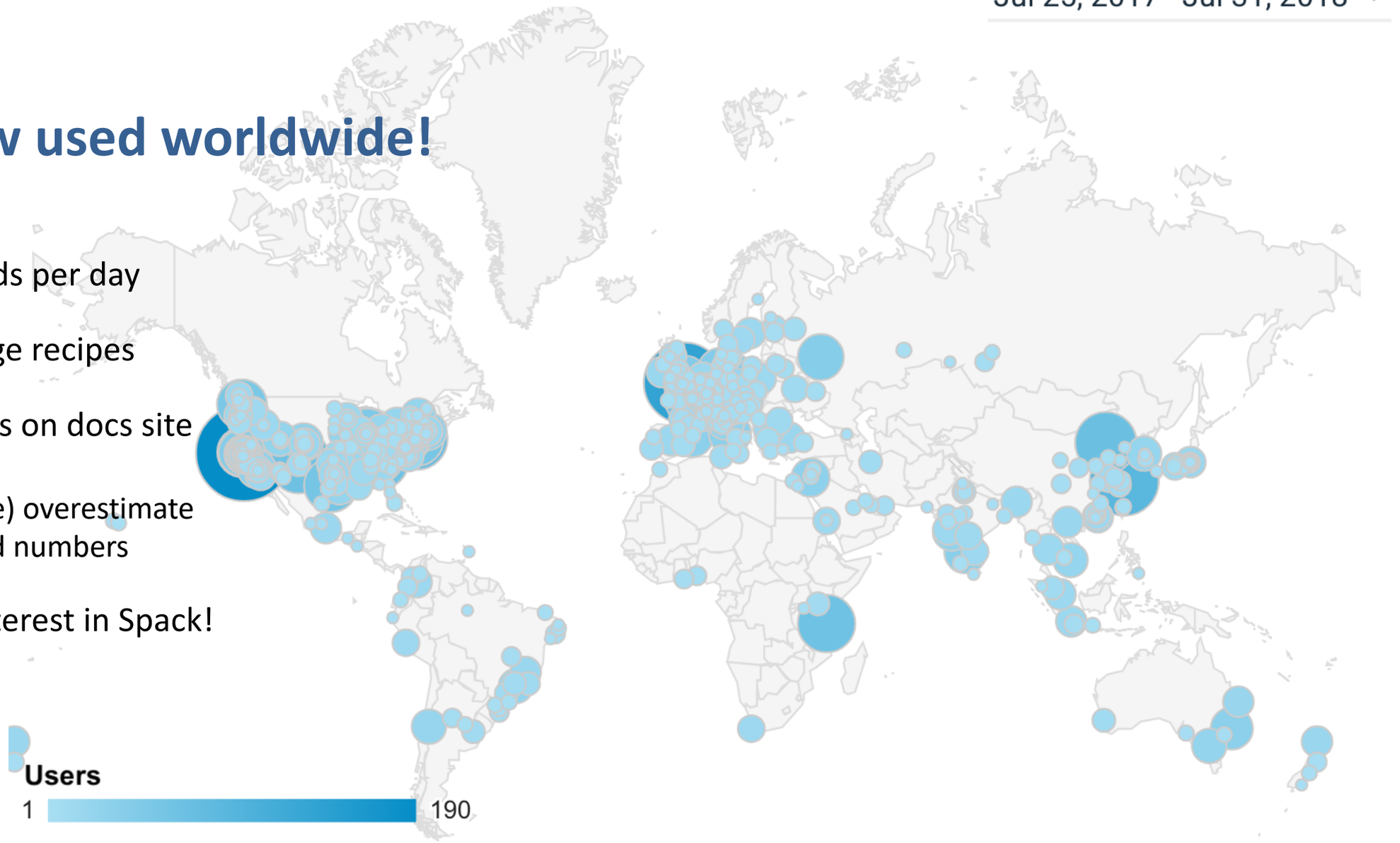
161 Pull requests merged by **52** people

#16 most contributed-to Python project on GitHub

https://krihelinator.xyz/languages/Python

Spack is now used worldwide!

- **400-500** downloads per day
- Over **2,800** package recipes
- **7,800** unique users on docs site in the past year
  - Probably a (large) overestimate
  - Hard to get good numbers
- There is steady interest in Spack!



Spack is being used on many of the top HPC systems

- At HPC sites for reproducible software stack+ modules
  - Reduced Summit deploy time from 2 weeks to 12 hrs.
  - EPFL deploys its software stack with Jenkins + Spack
  - NERSC, LLNL, ANL, other US DOE sites
  - SJTU in China
- Within ECP as part of their software release process
  - ECP-wide software distribution
  - SDK workflows
- Within High Energy Physics (HEP) community
  - HEP (Fermi, CERN) have contributed many features to support their workflow
- Many others

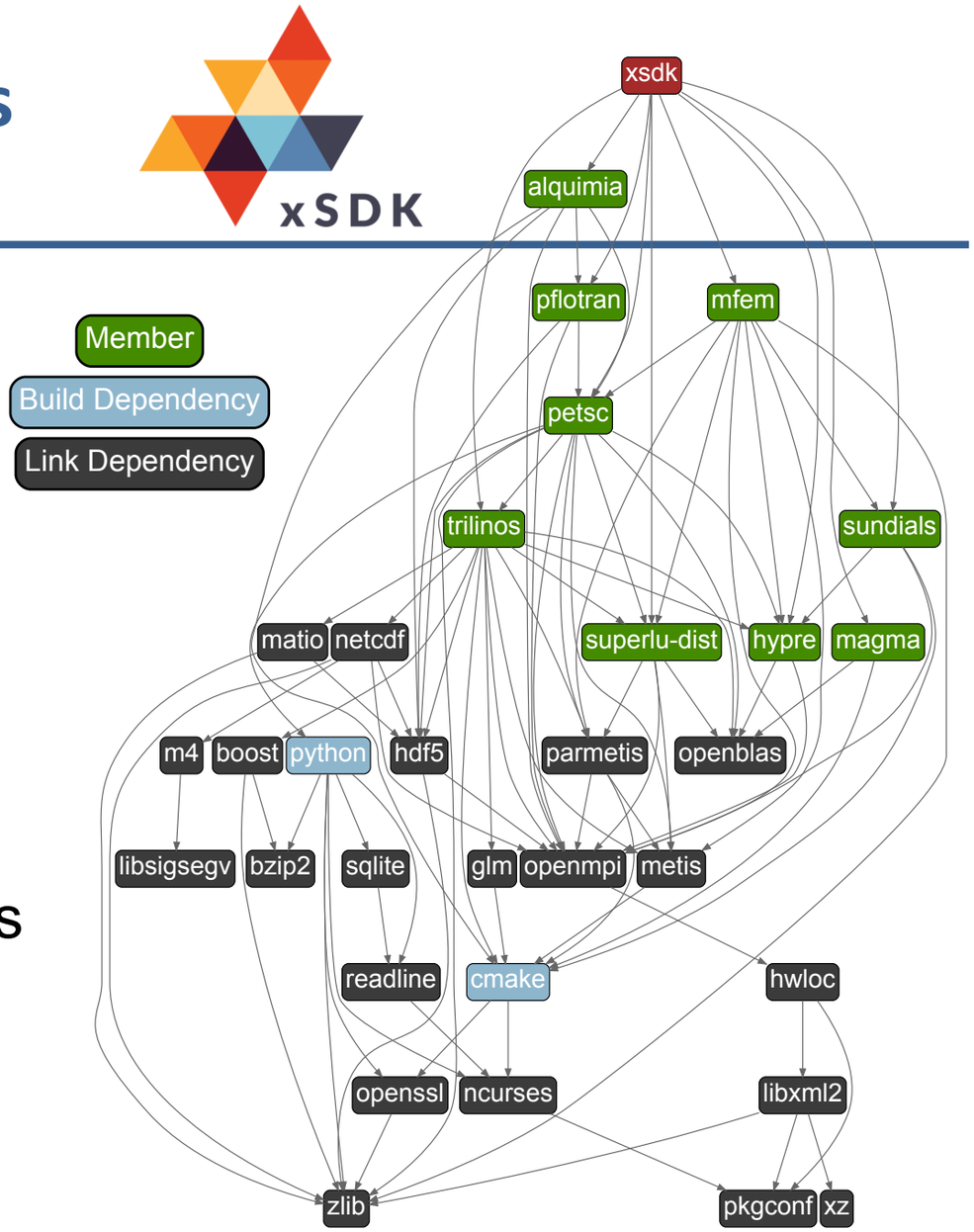
They wouldn't let me put a sticker on it…

ÉCOLE POLYTECHNIQUE FÉDÉRALE DE LAUSANNE

INFORMATIQUE SCIENTIFIQUE & SUPPORT APPLICATIF
SCITAS
SCIENTIFIC IT AND APPLICATION SUPPORT

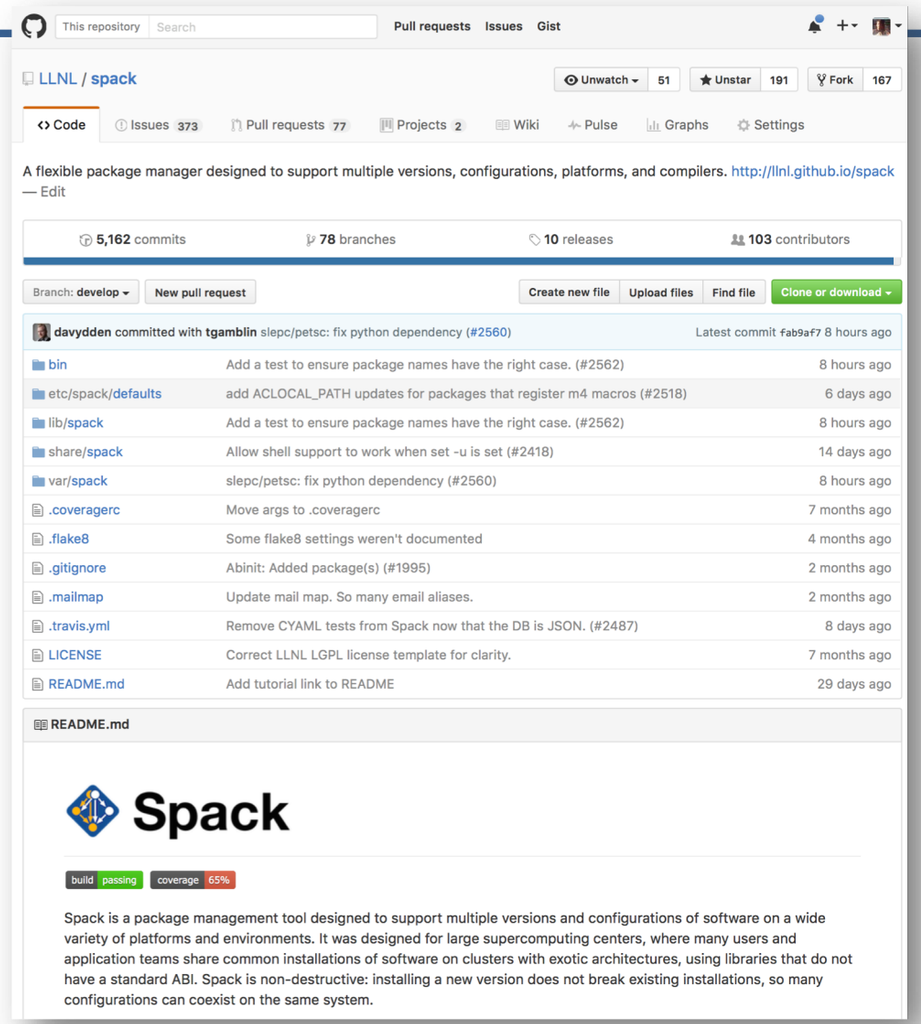
Under ECP, “SDK” teams manage releases using Spack

- xSDK pioneered the SDK concept
  - 8 member projects, 22 required dependencies
  - Includes many major solver library teams
  - Next release will have 10+ additional packages
- Teams work together on regular releases
  - Helps to work out compatibility issues
  - Gets developers talking to each other
  - Encourages teams to factor into smaller libraries
- ECP is establishing more SDKs for different areas
  - SDKs will be released using Spack
  - Each will be its own meta-package like xSDK



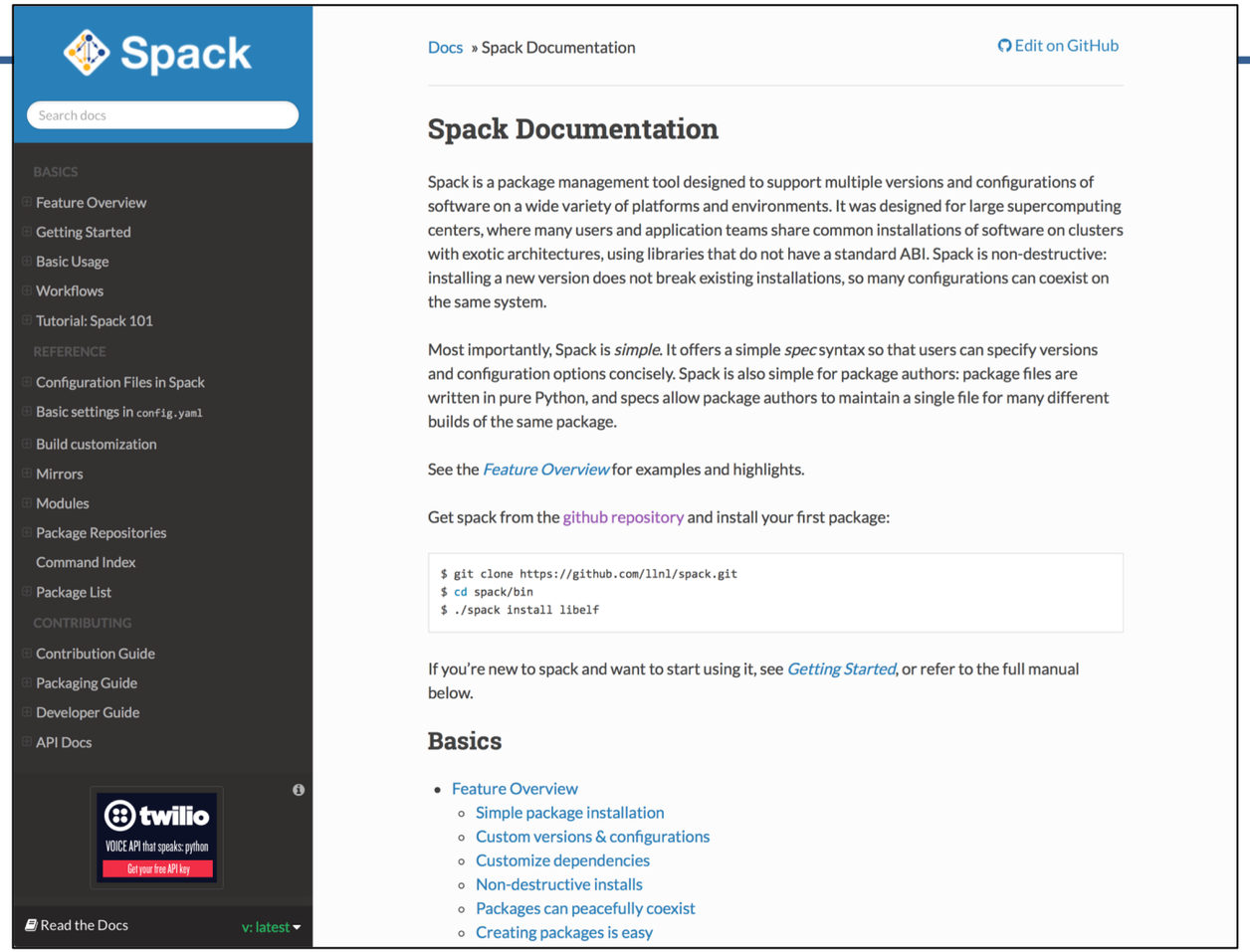
How did we build a community?

- Mostly luck, and also by lowering barriers for people to join.
- Without git & GitHub, we wouldn't be able to manage contributions from nearly as many people
  - Git handles concurrent development easily with forks and branches
- Many people are familiar with how to contribute to github projects
  - Pull requests are well understood and accessible



Documentation is critical for recruiting users and contributors

- Recent GitHub survey showed that users are much more inclined to use a well documented project
  - Obvious? Maybe.
- readthedocs.org makes hosting documentation easy
  - Sphinx .rst files are fairly easy to write
  - Auto-generate docs from the main GitHub repository
  - Built-in versioning
- Remember that users can contribute docs!
  - Make it easy for them.



spack.readthedocs.io

Testing and Continuous Integration are critical for scaling a large project.

- Spack currently uses Travis CI to test Spack *itself*.
  - Every contribution is tested against regression tests in the cloud.
  - We only merge pull requests if they pass Spack's test suite
  - Also enforce style guidelines
- The contribution process is also documented.
- Travis is free and easy!
  - Commit a single, short .travis.yml file to your repo
  - Tests are run automatically
- We would **never** be able to handle so many contributions if we had to do all this manually.

Travis CI

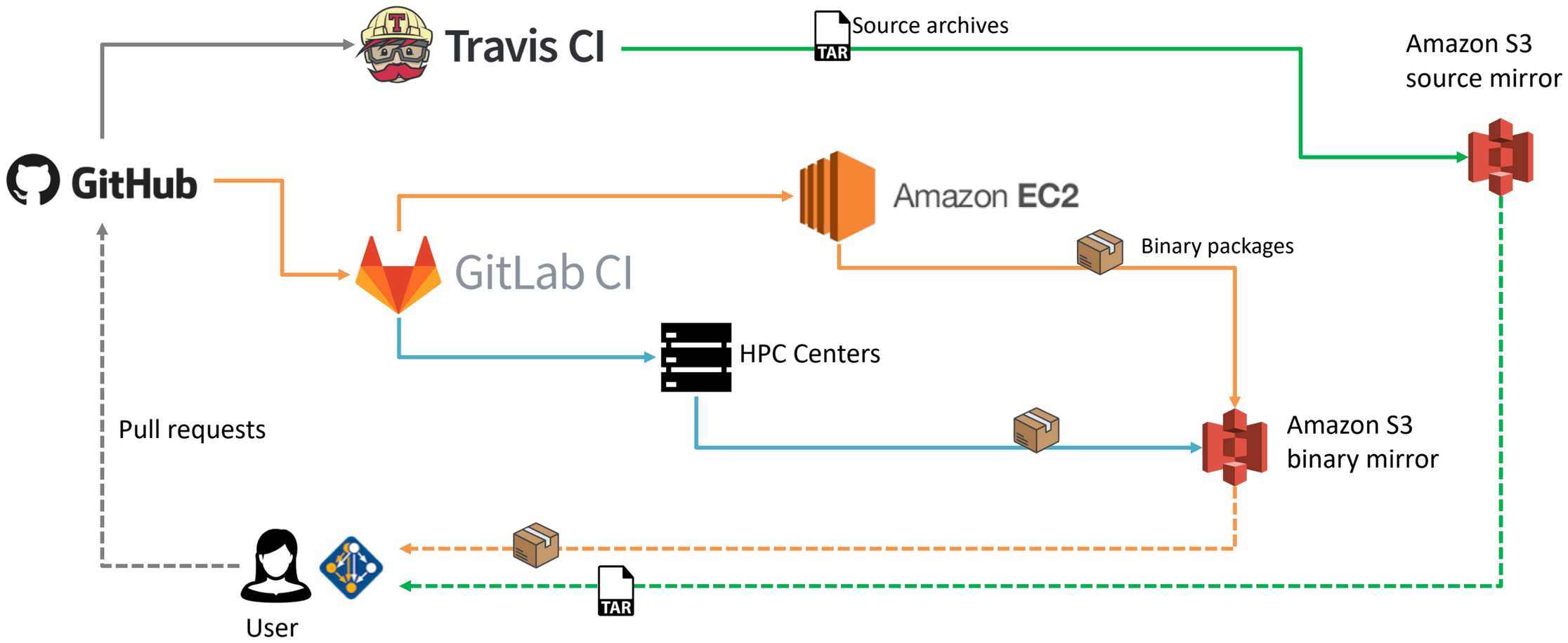
Spack Roadmap

1. **Infrastructure for binary distribution**
   - Source mirror with archives for all projects in Spack
   - Cloud-based build farm for relocatable binary packaging

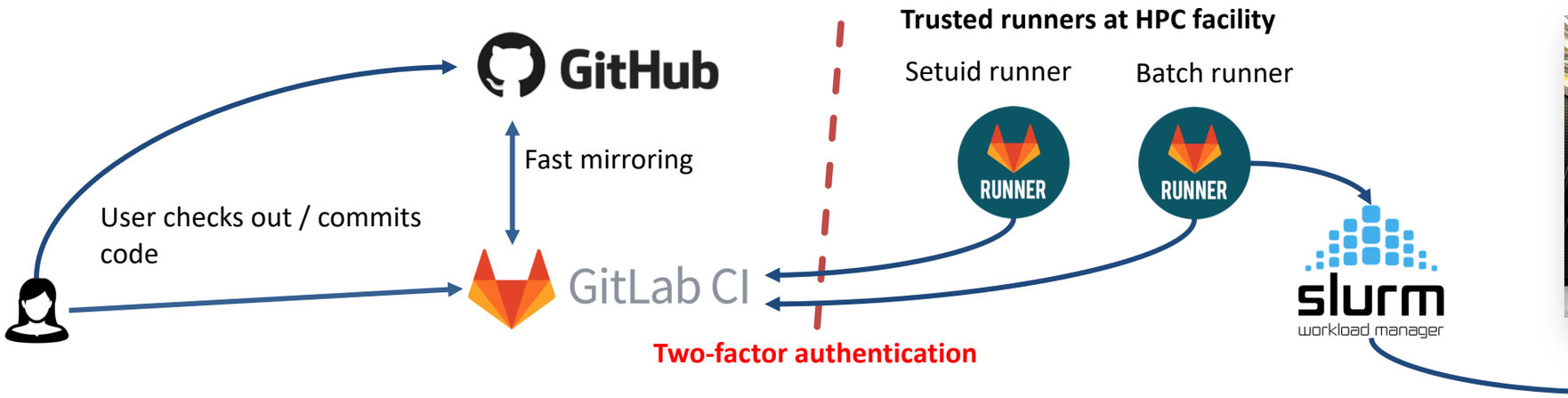
2. **“Environments” for developer dependency management**
   - Ability to have separate contexts in Spack
   - Manifest/lockfile model supported in Spack

3. **We are developing a new concretizer for Spack**
   - Needed to support the above two roadmap elements

We are building more extensive CI infrastructure to enable testing real package *builds*

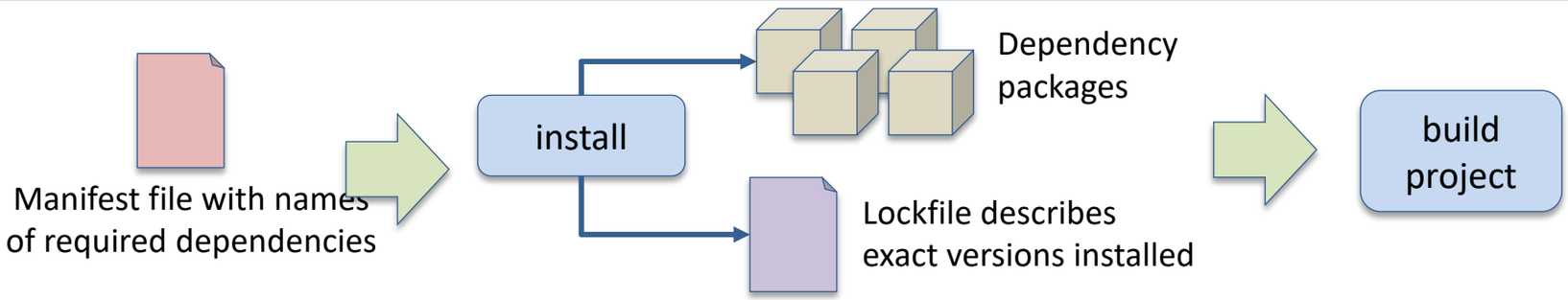


Through ECP, we are working with Onyx Point to deliver continuous integration for HPC centers



- CI at HPC centers is notoriously difficult
  - Security concerns prevent most CI tools from being run by staff or by users
  - HPC centers really need to deploy trusted CI services for this to work
- Contracted Onyx Point to develop a secure CI system for HPC centers:
  - Setuid runners (run CI jobs as users); Batch integration (similar, but parallel jobs); multi-center runner support
- This effort required cooperation from 6 labs and ECP!
- Getting everyone on the same page about requirements was key to enabling the project.

Package managers are a key part of reproducible workflows

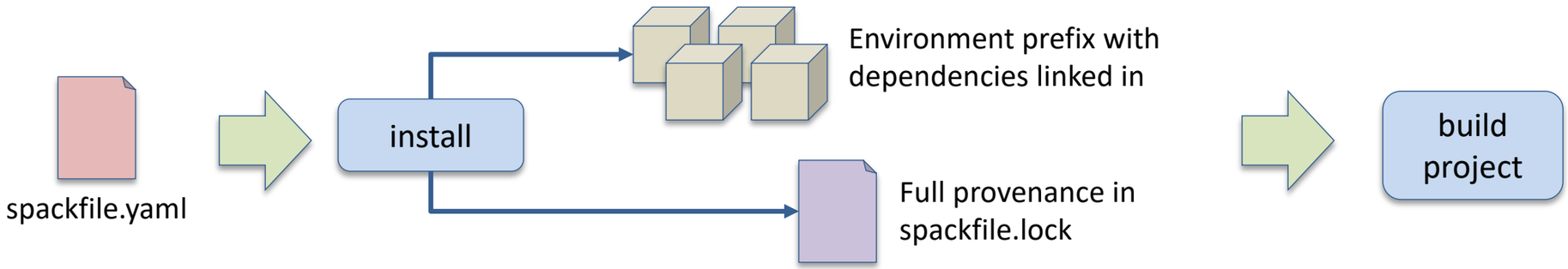


- Package managers are used to install packages needed before a build
- Also used by HPC facilities to manage system software on supercomputers
  - OS level (RPM, yum, yast, OpenHPC, etc.)
  - Scientific software level (increasingly Spack, Easybuild)
- Manifest / Lockfile model pioneered by Bundler is becoming standard
  - Lockfile can be used to exactly reproduce a prior installation
  - Many language-specific examples: Bundler, Cargo, npm, pipenv, etc.
- This model is very similar to Spack's concretization!

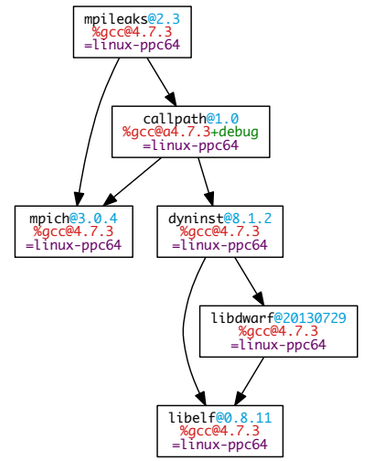
```
# Install various dependencies
addons:
  apt:
    packages:
      - gfortran
      - mercurial
      - graphviz
      - gnupg2
      - cmake
      - r-base
      - r-base-core
      - r-base-dev
      - perl
      - perl-base

install:
  - pip install --upgrade pip
  - pip install --upgrade six
  - pip install --upgrade setuptools
  - pip install --upgrade codecov
  - pip install --upgrade flake8
```

Excerpt from a `.travis.yml` file

Spack environments will support the manifest/lockfile model



- Ability to manage multiple environments independently
  - On the command line with regular spack commands, or
  - Projects can have a spackfile.yaml describing dependencies
- Lock file contains complete provenance
  - Lock file can be used to reproduce the same build on the same machine
  - Or to reproduce a “close as possible” build on a different machine



Binary packaging and environments require improvements to concretization

- Currently, Spack looks only at command line and package files for constraints
  - Does not make special efforts to reuse already-installed binaries
- For both environments and binary packaging, we need to reuse:
  - Available packages in a binary mirror
  - Available packages in the current environment
- New concretizer needs to do solves not just for package/version but also:
  - Build options (potentially multi-valued)
  - Compilers and compiler versions
  - Virtual dependencies
  - Binary compatibility and compiler constraints
- This is like existing manifest/lockfile solvers, but with many more constraints

Summary: Insights from Spack

- Don’t be afraid to tackle problems that are somewhat outside your primary domain.
  - I didn’t expect to be working on a build system!
- If you think your project could have impact, build a community!
  - Think about more than just your use case
  - Even if your contributors’ needs are different, they can help your project
  - Every new Spack package gets us more core contributions, more robustness
- General open source guidelines:
  - Documentation!
  - Continuous integration!
  - Use a language the community knows
  - Delegate and spread the work out to your contributors
- Be open to putting more work in. You could get much more out.

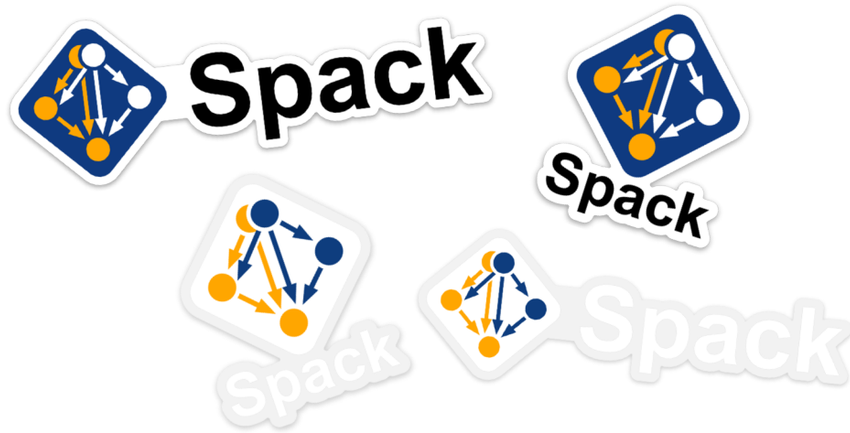
Visit us!

https://spack.io

github.com/spack

@spackpm

Get stickers!



CASC

Center for Applied
Scientific Computing