Spack
A flexible package manager for HPC

Overview & Introduction to Basic Spack Concepts

Todd Gamblin
Center for Applied Scientific Computing

LLNL-PRES-806064
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

github.com/LLNL/spack

Lawrence Livermore National Laboratory

Spack is a flexible package manager for HPC

- How to install Spack:

```
$ git clone https://github.com/scalability-llnl/spack.git
```

- How to install a package:

```
$ cd spack/bin
$ ./spack install hdf5
```

- HDF5 and its dependencies are installed within the Spack directory.

- No additional setup required!

Get Spack!

http://github.com/LLNL/spack

What is the production environment for HPC?

- Someone’s home directory?
- LLNL? LANL? Sandia? ANL? LBL? TACC?
  - Environments at large-scale sites are very different.
- Which MPI implementation?
- Which compiler?
- Which dependencies?
- Which versions of dependencies?
  - Many applications require specific dependency versions.

Real answer: there isn’t a single production environment or a standard way to build.

HPC software is becoming increasingly complex

- Not much standardization in HPC
  - every machine/application has a different software stack
- Sites share unique hardware among teams with *very* different requirements
  - Users want to experiment with many exotic architectures, compilers, MPI versions
  - All of this is necessary to get the best ***performance***
- Example environment for some LLNL codes:

48 third party packages x **3 MPI versions** mvapich mvapich2 OpenMPI x **3-ish Platforms** Linux BlueGene Cray

x **Up to 7 compilers** Intel GCC XLC Clang PGI Cray Pathscale x **Oh, and 2-3 versions of each package** = ~7,500 combinations

We want an easy way to quickly sample the space, to build configurations on demand!

Most existing tools do not support combinatorial versioning

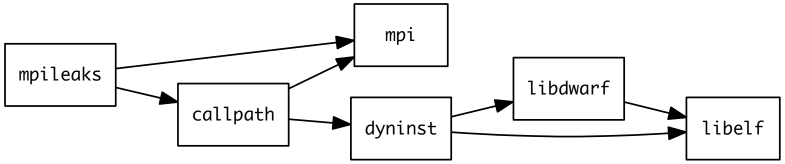
- Traditional binary package managers
  - RPM, yum, APT, yast, etc.
  - Designed to manage a single stack.
  - Install *one* version of each package in a single prefix (`/usr`).
  - Seamless upgrades to a *stable, well tested* stack

- Port systems
  - BSD Ports, portage, Macports, Homebrew, Gentoo, etc.
  - Minimal support for builds parameterized by compilers, dependency versions.

- Virtual Machines and Linux Containers (Docker)
  - Containers allow users to build environments for different applications.
  - Does not solve the build problem (someone has to build the image)
  - Performance, security, and upgrade issues prevent widespread HPC deployment.

Spack handles combinatorial software complexity.

Dependency DAG



Hash

Installation Layout

```
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*

`spack list` shows what packages are available

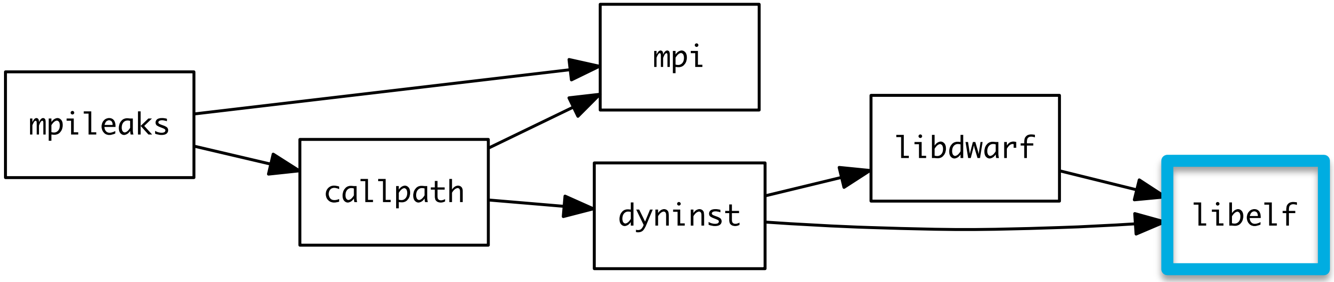
```
$ spack list
==> 308 packages.
activeharmony  cfitsio          fftw         gsl           libffi         matio              ompt-openmp       py-basemap        py-pil                py-virtualenv  szip
adept-utils    cgal             fish         gtkplus       libgcrypt      mbedtls            opari2            py-biopython      py-pillow             py-wheel       tar
apex           cgm              flex         harfbuzz      libgd          memaxes            openblas          py-blessings      py-pmw                py-yapf        task
arpack         cityhash         fltk         hdf           libgpg-error   mesa               openmpi           py-cffi           py-pychecker          python         taskd
asciidoc       cleverleaf       flux         hdf5          libjpeg-turbo  metis              openspeedshop     py-coverage       py-pycparser          qhull          tau
atk            cloog            fontconfig   hpx5          libjson-c      Mitos              openssl           py-cython         py-pyelftools         qt             tcl
atlas          cmake            freetype     hwloc         libmng         mpc                otf               py-dateutil       py-pygments           qthreads       texinfo
atop           cmocka           gasnet       hypre         libmonitor     mpe2               otf2              py-epydoc         py-pylint             R              the_silver_searcher
autoconf       coreutils        gcc          icu           libNBC         mpfr               pango             py-funcsigs       py-pypar              ravel          thrift
automaded      cppcheck         gdb          icu4c         libpciaccess   mpibash            papi              py-genders        py-pyparsing          readline       tk
automake       cram             gdk-pixbuf   ImageMagick   libpng         mpich              parallel-netcdf   py-gnuplot        py-pyqt               rose           tmux
bear           cscope           geos         isl           libsodium      mpileaks           paraver           py-h5py           py-pyside             rsync          tmuxinator
bib2xhtml      cube             gflags       jdk           libtiff        mrnet              paraview          py-ipython        py-pytables           ruby           trilinos
binutils       curl             ghostscript  jemalloc      libtool        mumps              parmetis          py-libxml2        py-python-daemon      SAMRAI         uncrustify
bison          czmq             git          jpeg          libunwind      munge              parpack           py-lockfile       py-pytz               samtools       util-linux
boost          damselfly        glib         judy          libuuid        muster             patchelf          py-mako           py-rpy2               scalasca       valgrind
bowtie2        dbus             glm          julia         libxcb         mvapich2           pcre              py-matplotlib     py-scientificpython   scorep         vim
boxlib         docbook-xml      global       launchmon     libxml2        nasm               pcre2             py-mock           py-scikit-learn       scotch         vtk
bzip2          doxygen          glog         lcms          libxshmfence   ncdu               pdt               py-mpi4py         py-scipy              scr            wget
cairo          dri2proto        glpk         leveldb       libxslt        ncurses            petsc             py-mx             py-setuptools         silo           wx
caliper        dtcmp            gmp          libarchive    llvm           netcdf             pidx              py-mysqldb1       py-shiboken           snappy         wxpropgrid
callpath       dyninst          gmsh         libcerf       llvm-lld       netgauge           pixman            py-nose           py-sip                sparsehash     xcb-proto
cblas          eigen            gnuplot      libcircle     lmdb           netlib-blas        pkg-config        py-numexpr        py-six                spindle        xerces-c
cbtf           elfutils         gnutls       libdrm        lmod           netlib-lapack      pmgr_collective   py-numpy          py-sphinx             spot           xz
cbtf-argonavis elpa             gperf        libdwarf      lua            netlib-scalapack   postgresql        py-pandas         py-sympy              sqlite         yasm
cbtf-krell     expat            gperftools   libedit       lwgrp          nettle             ppl               py-pbr            py-tappy              stat           zeromq
cbtf-lanl      extrae           graphlib     libelf        lwm2           ninja              protobuf          py-periodictable  py-twisted            sundials       zlib
cereal         exuberant-ctags  graphviz     libevent      m4             ompss              py-astropy        py-pexpect        py-urwid              swig           zsh
```

Spack provides a *spec* syntax to describe customized DAG 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 =bgq                        = cross-compile
```

- 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!
- Syntax abstracts details in the common case
  - Makes parameterization by version, compiler, and options easy when necessary

Spack Specs can constrain versions of dependencies

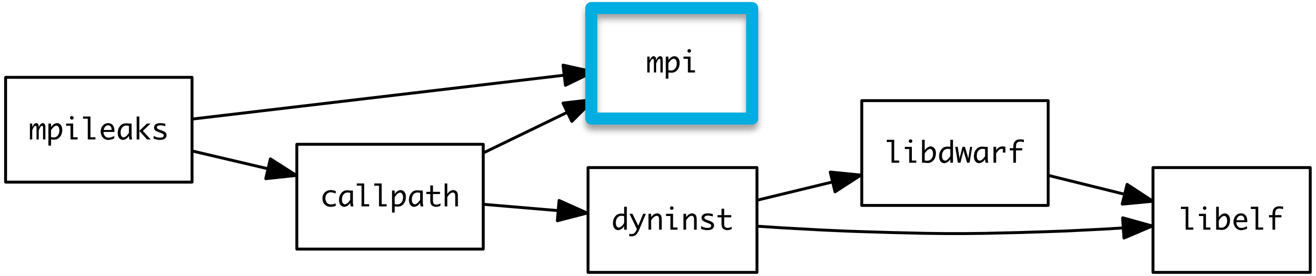



```
$ spack install mpileaks %intel@12.1 ^libelf@0.8.12
```

- Spack ensures *one* configuration of each library per DAG
  - Ensures ABI consistency.
  - User does not need to know DAG structure; only the dependency *names.*

- Spack can ensure that builds use the same compiler, or you can mix
  - Working on ensuring ABI compatibility when compilers are mixed.

Spack handles ABI-incompatible, versioned interfaces like MPI



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

```
$ spack install mpileaks ^mvapich@1.9
```

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

- Let Spack choose MPI version, as long as it provides MPI 2 interface:

```
$ spack install mpileaks ^mpi@2
```

Spack packages are simple Python scripts.

```
from spack import *

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

    homepage = "https://paradyn.org"

    version('8.2.1', 'abf60b7faabe7a2e', url="http://www.paradyn.org/release8.2/DyninstAPI-8.2.1.tgz")
    version('8.1.2', 'bf03b33375afa66f', url="http://www.paradyn.org/release8.1.2/DyninstAPI-8.1.2.tgz")
    version('8.1.1', 'd1a04e995b7aa709', url="http://www.paradyn.org/release8.1/DyninstAPI-8.1.1.tgz")

    depends_on("libelf")
    depends_on("libdwarf")
    depends_on("boost@1.42:")

    def install(self, spec, prefix):
        libelf = spec['libelf'].prefix
        libdwarf = spec['libdwarf'].prefix

        with working_dir('spack-build', create=True):
            cmake('..',
                  '-DBoost_INCLUDE_DIR=%s' % spec['boost'].prefix.include,
                  '-DBoost_LIBRARY_DIR=%s' % spec['boost'].prefix.lib,
                  '-DBoost_NO_SYSTEM_PATHS=TRUE'
                  *std_cmake_args)
            make()
            make("install")

    @when('@:8.1')
    def install(self, spec, prefix):
        configure("--prefix=" + prefix)
        make()
        make("install")
```

Metadata

Versions and URLs

Dependencies

Patches, variants (not shown)

Commands for installation

Access build config through the *spec* parameter.

Variants allow optional dependencies

- The user can define named *variants* (flags):

```
variant("python", default=False, “Build with python support”)
depends_on("python", when="+python")
```

- And use them to install:

```
$ spack install vim +python
$ spack install vim –python
```

- Dependencies may be optional according to other conditions:
  e.g., gcc dependency on mpc from 4.5 on:

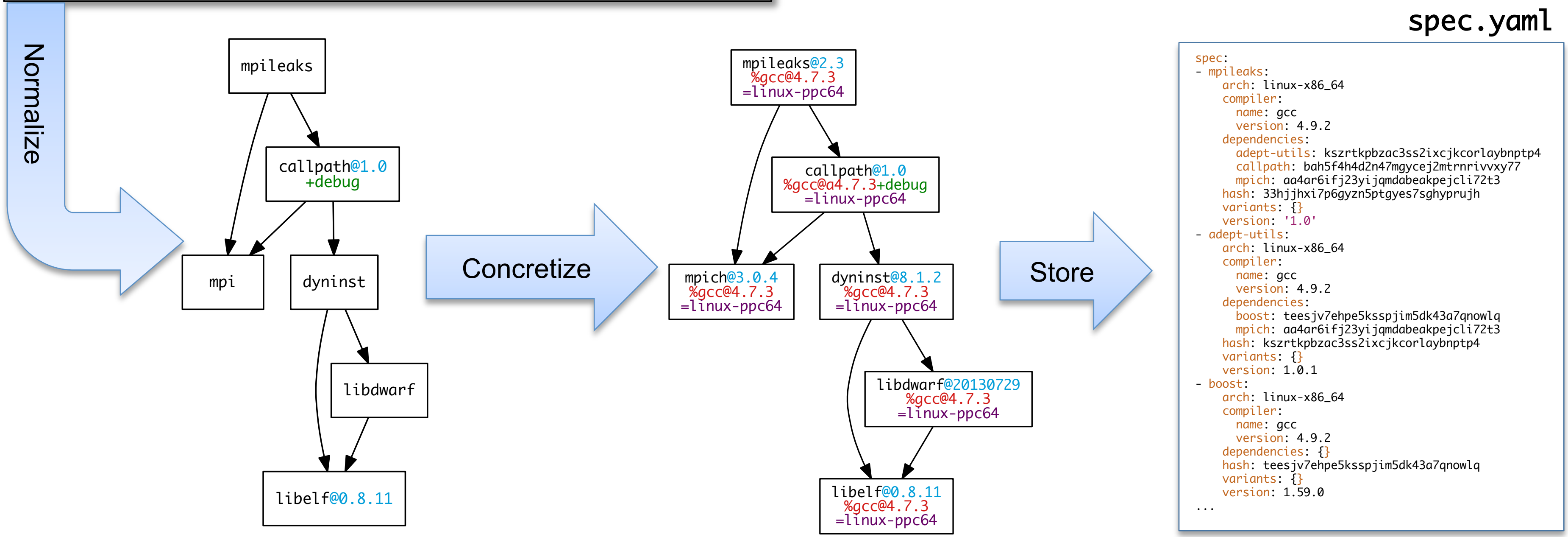
```
depends_on("mpc", when="@4.5:")
```

- DAG is not always complete before concretization!

Concretization fills in missing configuration details when the user is not explicit.

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

User input: *abstract* spec with some constraints



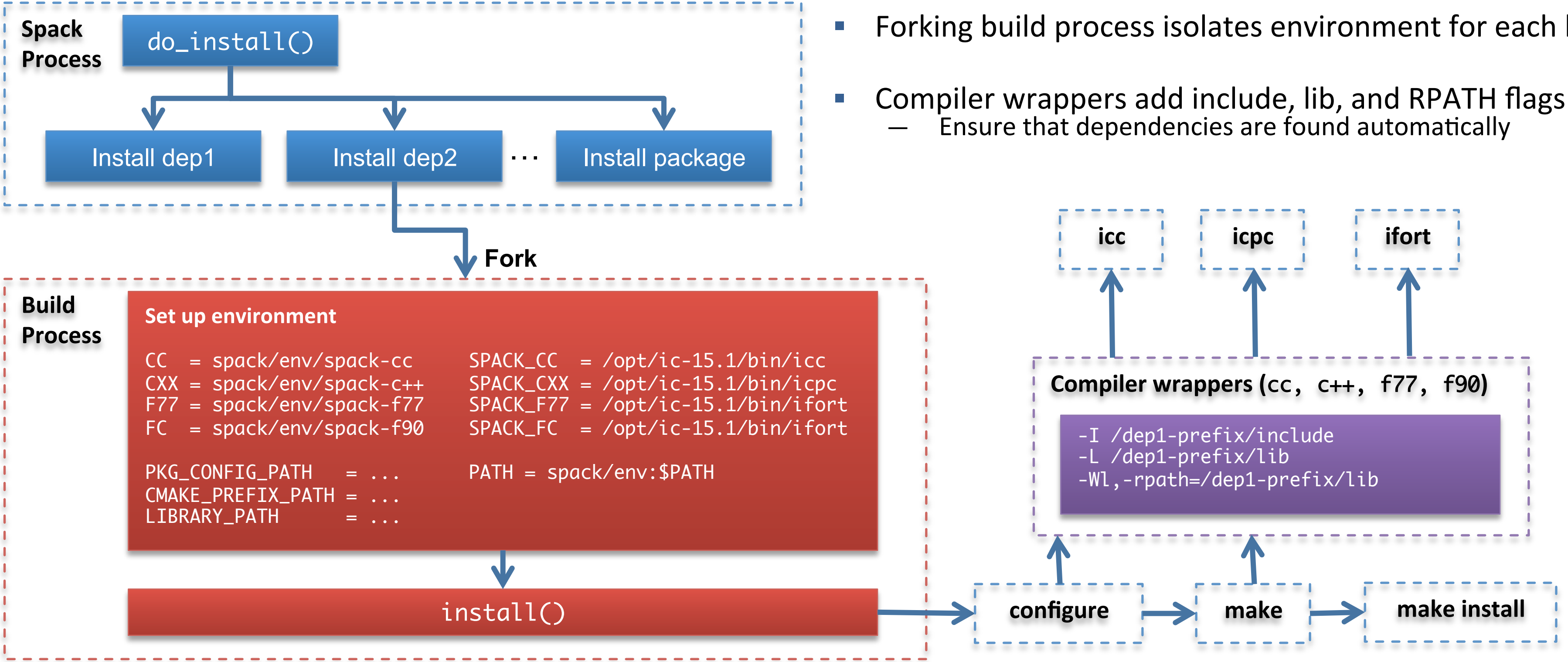
Abstract, normalized spec with some dependencies.

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

Detailed provenance is stored with the installed package

Spack builds each package in its own compilation environment

- Forking build process isolates environment for each build.
- Compiler wrappers add include, lib, and RPATH flags
  - Ensure that dependencies are found automatically



Use Case 1: Managing combinatorial installations

```
$ spack find
==> 103 installed packages.
-- linux-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-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-x86_64 / intel@14.0.2 ------------------------------
hwloc@1.9  mpich@3.0.4  starpu@1.1.4

-- linux-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-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
```

- spack find shows all installed configurations
  - Multiple versions of same package are ok.
- Packages are divided by architecture/compiler.
- Spack also generates module files.
  - Don't *have* to use them.

Using the Spec syntax, Spack can restrict queries

```
$ spack find mpich
==> 5 installed packages.
-- linux-x86_64 / gcc@4.4.7 --------------------------------
mpich@3.0.4

-- linux-x86_64 / gcc@4.8.2 --------------------------------
mpich@3.0.4

-- linux-x86_64 / intel@14.0.2 -----------------------------
mpich@3.0.4

-- linux-x86_64 / intel@15.0.0 -----------------------------
mpich@3.0.4

-- linux-x86_64 / intel@15.0.1 -----------------------------
mpich@3.0.4
```

- Querying by package name retrieves a subset

The Spec syntax doubles as a query language to allow refinement of searches.

```
$ spack find libelf
==> 5 installed packages.
-- linux-x86_64 / gcc@4.4.7 ---------
libelf@0.8.12  libelf@0.8.13

-- linux-x86_64 / gcc@4.8.2 ---------
libelf@0.8.13

-- linux-x86_64 / intel@15.0.0 ------
libelf@0.8.13

-- linux-x86_64 / intel@15.0.1 ------
libelf@0.8.13
```

Query versions of `libelf` package

List only those built with Intel compiler.

```
$ spack find libelf %intel
-- linux-x86_64 / intel@15.0.0 ------
libelf@0.8.13

-- linux-x86_64 / intel@15.0.1 ------
libelf@0.8.13
```

```
$ spack find libelf %intel@15.0.1
-- linux-x86_64 / intel@15.0.1 ------
libelf@0.8.13
```

Restrict to specific compiler version

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, and dependency versions may differ between builds.

Use Case 2: Package Views for HPC Center Installs

```
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/
    ...
```

```
/software/
    linux-x86_64/
        gcc-4.7.2/
            mvapich-1.9/
                mpileaks-1.1/
        intel-14.1/
            mvapich-1.9/
                hdf5-1.8.15/
    bgq/
        xl-12.1/
            ibm-mpi/
                hdf5-1-8.16/
    ...
```

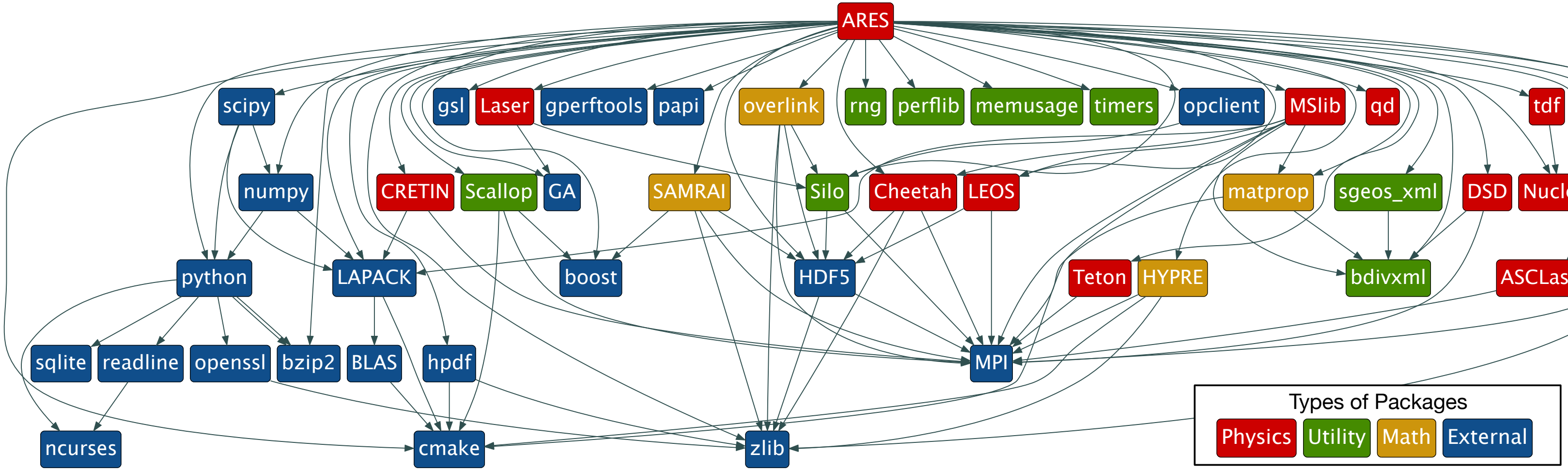
- Many users like to navigate a readable directory hierarchy
  - Spack's combinatorial package space is large and can be hard to navigate
- Spack can generate a coarser tree *view* of symbolic links
  - View is a projection from the higher-dimensional Spack space
  - Some names may conflict, but spec syntax allows us to express *preferences* to guide view creation.

Use case 3: Python and other interpreted languages

```
$ spack install python@2.7.10
==> Building python.
==> Successfully installed python.
  Fetch: 5.01s.  Build: 97.16s.  Total: 103.17s.
[+] /home/gamblin2/spack/opt/spack/linux-x86_64/gcc-4.9.2/python-2.7.10-y2zr767

$ spack extensions python@2.7.10
==> python@2.7.10%gcc@4.9.2=linux-x86_64-y2zr767
==> 49 extensions:
geos          py-h5py        py-numpy      py-pypar             py-setuptools
libxml2       py-ipython     py-pandas     py-pyparsing         py-shiboken
py-basemap    py-libxml2     py-pexpect    py-pyqt              py-sip
py-biopython  py-lockfile    py-pil        py-pyside            py-six
py-cffi       py-mako        py-pmw        py-python-daemon     py-sphinx
py-cython     py-matplotlib  py-pychecker  py-pytz              py-sympy
py-dateutil   py-mock        py-pycparser  py-rpy2              py-virtualenv
py-epydoc     py-mpi4py      py-pyelftools py-scientificpython  py-yapf
py-genders    py-mx          py-pygments   py-scikit-learn      thrift
py-gnuplot    py-nose        py-pylint     py-scipy

==> 3 installed:
-- linux-x86_64 / gcc@4.9.2 -----------------------------------
py-nose@1.3.6  py-numpy@1.9.2  py-setuptools@18.1

==> None currently activated.

$ spack activate py-numpy
==> Activated extension py-setuptools-18.1-gcc-4.9.2-ru7w3lx
==> Activated extension py-nose-1.3.6-gcc-4.9.2-vudjpwc
==> Activated extension py-numpy-1.9.2-gcc@4.9.2-45hjazt

$ spack deactivate -a py-numpy
==> Deactivated extension py-numpy-1.9.2-gcc@4.9.2-45hjazt
==> Deactivated extension py-nose-1.3.6-gcc-4.9.2-vudjpwc
==> Deactivated extension py-setuptools-18.1-gcc-4.9.2-ru7w3lx
```

- Many interpreted languages have their own mechanisms for modules, e.g.:
  - Require installation into interpreter prefix
  - Breaks combinatorial versioning
- Spack installs each Python package in its own prefix
- "Activating" links an extension into the interpreter directory on demand
  - Supports .egg, merging .pth files
  - Mechanism is extensible to other languages
  - Similar to virtualenv, but Spack allows much more build customization.

Spack builds real LLNL codes



- ARES is a 1, 2, and 3-D radiation hydrodynamics code
- Spack automates the build of ARES and all of its dependencies
  - The ARES configuration shown above has 47 dependencies

ARES has used Spack to test 36 different configurations

- Nightly builds of ARES are shown at right.

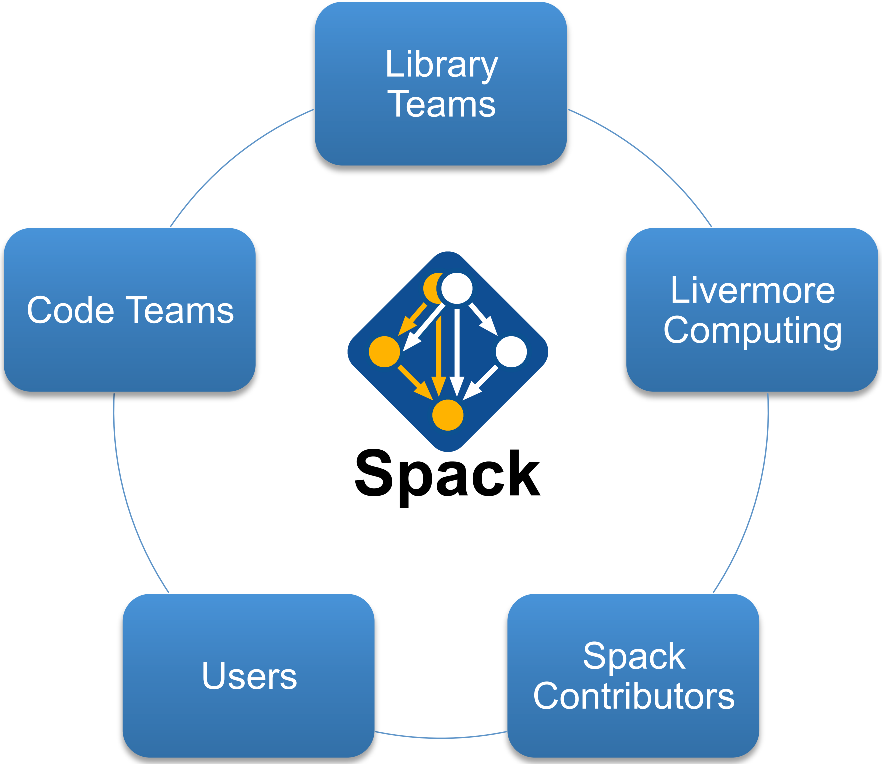
  4 code versions:
  - **(C)**urrent Production
  - **(P)**revious Production
  - **(L)**ite
  - **(D)**evelopment

	Linux			*BG/Q*	*Cray XE6*
	MVAPICH	*MVAPICH2*	*OpenMPI*	*BG/Q MPI*	*Cray MPI*
GCC	C P L D			C P L D	
Intel 14	C P L D				
Intel 15	C P L D	D			
PGI		D	C P L D		C L D
Clang	C P L D			C L D	
XL				C P L D	

- Learning Spack and porting all libraries took a single developer 2 months, half-time.
- Previously, the team was only able to automate its development Linux builds.
  - Spack enabled thorough testing of many more configurations
  - Testing with Spack helped find compilation issues when using Clang compiler.
- Spack is helping the team port to LANL's new Trinity (Cray XC-40) machine

Build automation allows tedious work to be leveraged.

- Spack enables teams to share work.
  - Archives common library build recipes.
  - Prevents duplication of build effort.
  - We can share builds among LC, code teams, and users
- Patches allow rapid deployment of bug fixes
  - App team porting a library may not own its repo.
  - Library teams may not have time to fix issues quickly.
  - Code teams can fix quickly, then feed back changes.
- Python allowed quick adoption by code teams.
  - Many app developers already know Python
  - Spec syntax provides extra expressiveness.



Get Involved with Spack!

github.com/LLNL/spack

- **20+ organizations**
  39 contributors
  Sharing **320+ packages** and growing

- **Spack can be a central repository for tools**
  - Make it easy for others to use them!

- **Spack is used in production at LLNL**
  - Livermore Computing, ARES, MARBL, others.

- **Spack has a rapidly growing community.**
  - **NERSC** using Spack on Cori: Cray support.
  - **ANL** is using Spack on their Linux clusters.
  - **ORNL** working with us on Spack for CORAL.
  - **EPFL** (Switzerland) contributing core features.
  - **Kitware**: ParaView, other core features.

Lawrence Livermore National Laboratory

Coming soon: Compiler parameter studies

```
$ spack install ares cflags=‘-O3 -g -fast -fpack-struct’
```

- This would install ARES with the specified flags
  - Flags are injected via Spack’s compiler wrappers.
- Flags are propagated to dependencies automatically
  - Flags are included in the **DAG hash**
  - Each build is considered a **different version**
- This provides an easy harness for doing parameter studies for tuning codes
  - Previously working with large codes was very tedious.

Spack provides hooks that enable tools to work with large codes.

Future direction: Compiler wrappers for tools

- **Automatically adding source instrumentation to large codes is difficult**
  - Usually requires a lot of effort, especially if libraries need to be instrumented as well.
- **Spack could support Klocwork, Scalasca, TAU, thread sanitizers like archer, and others as "secondary" compiler wrappers.**
  - Allow user to build many instrumented versions of large codes, with many different compilers:

```
spack install application@3.3 %gcc@4.7.3 +archer
```

- **Spack packages again provide a general interface to build details.**
- **LLNL ARCHER debugging tool is looking into using this.**
  - Uses LLVM for instrumentation; needs to cover code **and** all libraries.

Future direction: Dependencies on compiler features

- Profusion of new compiler features frequently causes build confusion:
  - C++11 feature support
  - OpenMP language levels
  - CUDA compute capabilities

- Spack could allow packages to request compiler features like dependencies:

```
require('cxx11-lambda')
require('openmp@4:')
```

- Spack could:
  1. Ensure that a compiler with these features is used
  2. Ensure consistency among compiler runtimes in the same DAG.