Spack: Bringing Order to HPC Software Chaos

Scalable Tools Workshop 2015

August 3, 2015

http://bit.ly/spack-git

Todd Gamblin

Center for Applied Scientific Computing

Lawrence Livermore National Laboratory

Contributors

Matt Legendre

Greg Lee

Mike Collette

Bronis de Supinski

Scott Futral

LLNL-PRES-675781

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

What is the "production" environment for HPC codes?

- Someone's home directory?
- LLNL? LANL? Sandia? ANL? LBL? TACC?
  - Environments at these sites are very different.
- Which MPI?
- Which compiler?
- Which dependency versions?

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

Why is building so hard?

- Not much standardization in HPC
- Every machine and app has a different software stack (or several)
- We want to experiment with many exotic architectures, compilers, MPI versions
- All of this is necessary to get the best ***performance***

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**

= **~7,500** combinations

- OK, so we don't build **all** of these
  - Many combinations don't make sense
- We want an easy way to quickly sample the space
  - Build a configuration on demand!

How do HPC sites deal with combinatorial builds?

- **OS distribution does not deal with this**
  - OS typically has one version of each package, installed in a common prefix: `/usr`
- **HPC software typically installed manually in a directory hierarchy.**
  - Hierarchy often doesn't give you all the information you need about a build.
  - Typically run out of unique names for directories quickly.
- **Environment modules allow you to enable/disable packages.**

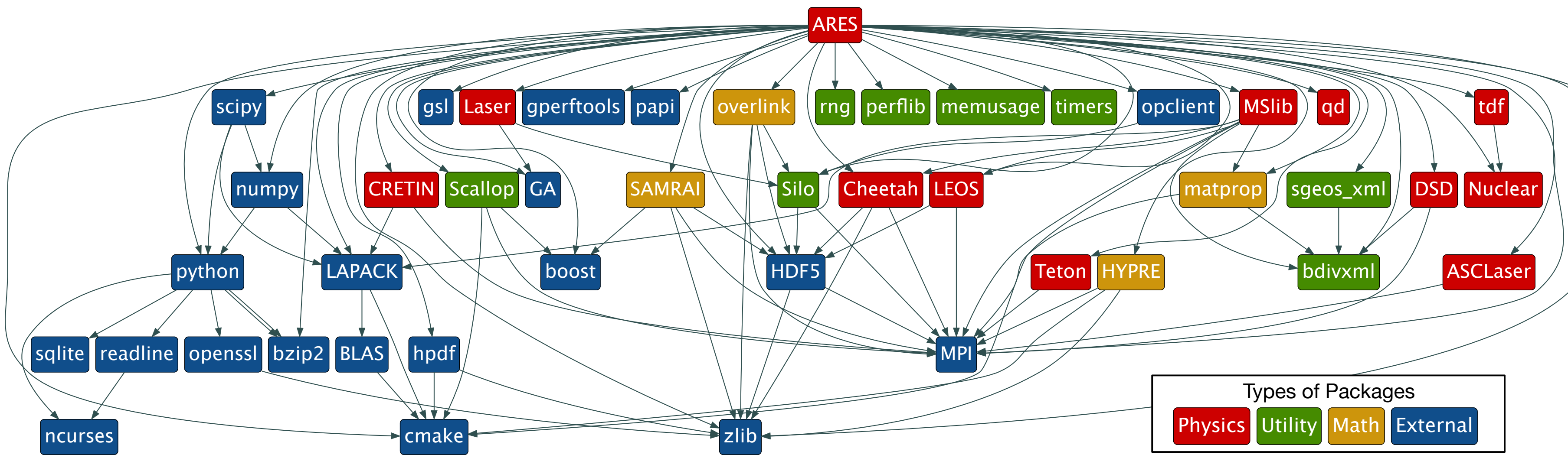
Site	Naming Convention
LLNL	`/ usr / global / tools / $arch / $package / $version` `/ usr / local / tools / $package-$compiler-$build-$version`
Oak Ridge	`/ $arch / $package / $version / $build`
TACC	`/ $compiler-$comp_version / $mpi / $mpi_version / $package / $version`

Environment modules can be hard to get right.

```
$ module avail

------------------------ /usr/share/Modules/modulefiles ------------------------
dot         module-git  module-info modules     null        use.own

--------------------------- /opt/modules/modulefiles ----------------------------
acml-gnu/4.4            intel/11.1              mvapich2-pgi-ofa/1.7
acml-gnu_mp/4.4         intel/12.0              mvapich2-pgi-psm/1.7
acml-intel/4.4          intel/12.1(default)     mvapich2-pgi-shmem/1.7
acml-intel_mp/4.4       intel/13.0              netcdf-gnu/4.1
acml-pathscale/4.0      intel/14.0              netcdf-intel/4.1
...

$ module load intel/12.0
$ module load mvapich2-pgi-shmem/1.7
```

- **Advantages:**
  - Allow you to swap different library versions dynamically, in your shell.
- **Disadvantages:**
  - Module system doesn't build software: only changes environment
  - Typically have to load the same module that you built with.
    - Easy to load wrong module; code no longer works.

Example: Spack has recently been adopted by ARES, an LLNL production code.



- **ARES is a 1, 2, and 3-D radiation hydrodynamics code**
  - **Used in munitions modeling and ICF simulation**
  - **Runs on LLNL and LANL machines**
- **Dependencies of ARES v3.0 shown above**
  - **47 component packages**
- **Spack automates the build of ARES and its dependencies**
  - **Also being used to automate post-build testing.**

ARES has uses Spack to test 36 different configurations

	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	

- **Above are nightly builds of ARES on machines at LLNL and LANL**
  - **Zin, Sequioa, Cielo**
- **4 code versions:**
  - **(C)**urrent Production
  - **(P)**revious Production
  - **(L)**ite
  - **(D)**evelopment
- **Team is currently porting to the new Trinity machine**

Spack handles combinatorial version complexity.

```
spack/opt/
    linux-x86_64/
        gcc-4.7.2/
            mpileaks-1.1-0f54bf/
    bgq/
        gcc-4.5.1/
            libelf-0.8.13-251fqb/
    ...
```

- Each unique DAG is a unique configuration.
- Many configurations can coexist.
- Each package ***configuration*** is installed in a unique directory.
- **Hash** appended to each prefix allows versioning of full dependency DAG.

- **Installed packages will automatically find their dependencies**
  - Binaries are installed with proper RPATHs
  - No need to use modules or customize LD_LIBRARY_PATH
  - Things continue to work *the way you built them*

- **Installation works just as well in $HOME as in shared FS.**

`spack list` shows what's available

```
$ spack list
==> 244 packages.
adept-utils  DSD          lapack         memusage      papi              py-pygments           scotch
ares         dtcmp        Laser          memwatch      paraver           py-pylint             scr
arpack       dyninst      launchmon      mesa          parmetis          py-pypar              sgeos
ASCLaser     extrae       lcms           metis         parpack           py-pyparsing          sgeos_xml
atk          flex         Leos           miranda       pcre              py-pyqt               sha
atlas        fontconfig   libarchive     Mitos         perflib           py-pyside             silo
autoconf     freetype     libcircle      mpc           petsc             py-pytz               spindle
automaded    ft_hash      libdrm         mpe2          pixman            py-rpy2               sqlite
automake     GA           libdwarf       mpfr          pmgr_collective   py-scientificpython   stat
bdivlibs     gasnet       libelf         mpibash       Pmw               py-scikit-learn       sundials
bdivxml      gcc          libevent       mpich         postgresql        py-scipy              swig
bib2xhtml    gdk-pixbuf   libffi         mpileaks      ppl               py-setuptools         szip
binutils     geos         libgcrypt      mpism         py-basemap        py-shiboken           task
bison        gidiplus     libgpg-error   mrnet         py-biopython      py-sip                taskd
boost        git          libjpeg-turbo  mslib         py-cffi           py-six                tau
boxlib       glib         libmng         muster        py-cython         py-sympy              tcl
bzip2        gmock        libmonitor     mvapich2      py-dateutil       py-virtualenv         tdf
cairo        gmp          libNBC         nasm          py-epydoc         python                Teton
callpath     gnutls       libpng         ncurses       py-genders        qd                    the_silver_searcher
cblas        gperf        libtiff        netcdf        py-gnuplot        qhull                 timers
cgm          gperftools   libtool        netgauge      py-h5py           qt                    tk
check        graphlib     libunwind      netlib-blas   py-ipython        qthreads              tmux
Cheetah      gsl          libuuid        nettle        py-libxml2        R                     tmuxinator
clang        gtkplus      libxcb         nuclear       py-mako           raja                  uncrustify
cloog        harfbuzz     libxml2        numpy         py-matplotlib     ravel                 util-linux
cmake        hdf5         libxshmfence   ompss         py-mpi4py         readline              vim
cndf         hpdf         libxslt        opari2        py-mx             rng                   vtk
coreutils    hwloc        llvm           opclient      py-nose           rose                  wget
cppcheck     hypre        llvm-lld       openmpi       py-numpy          ruby                  wx
cram         icu          lua            openssl       py-pandas         SAMRAI                wxpropgrid
cretin       icu4c        lwgrp          otf           py-pexpect        SandiaGeo             xcb-proto
cube         ImageMagick  lwm2           otf2          py-pil            scalasca              xz
dbus         isl          matprop        overlink      py-pmw            scallop               yasm
dmalloc      jdk          mcapm          pact          py-pychecker      scipy                 zlib
dri2proto    jpeg         memaxes        pango         py-pycparser      scorep
```

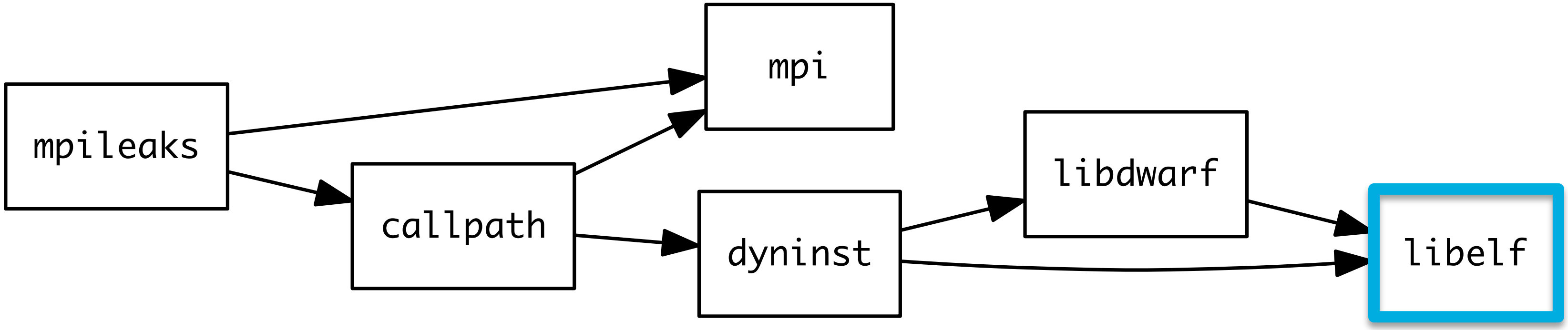
Spack provides a *spec* syntax to describe customized DAG configurations

```
$ spack install ares                              default: unconstrained

$ spack install ares@3.3                               @ custom version

$ spack install ares@3.3 %gcc@4.7.3                    % custom compiler

$ spack install ares@3.3 %gcc@4.7.3 +threads       +/- build option

$ spack install ares@3.3 =bgqos_0                      = 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!

- Package authors can use same syntax within package files
  - Makes it easy to parameterize build by version, compiler, arch, etc.

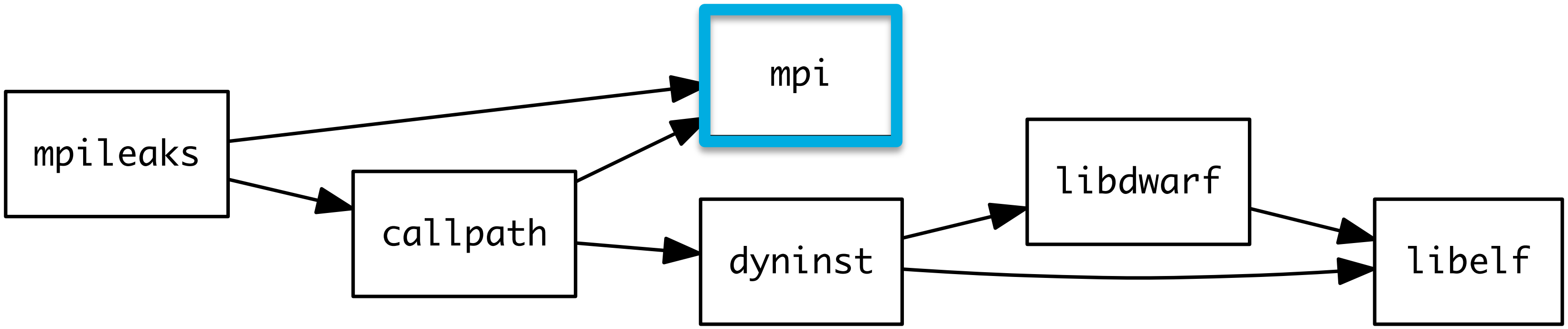
Specs can constrain dependency versions




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

- Spack ensures that all packages in the same install are built with the same version of libraries, like **`libelf`**.
- Spack can ensure that builds use the same compiler
  - Can also mix compilers but it's not default

Spack handles ABI-incompatible, versioned interfaces like MPI



Ask specifically for mvapich 1.9

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

Ask for openmpi 1.4 or higher

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

These install separately, in unique directories

Ask for an MPI that supports MPI-2 interface

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

Spack chooses an MPI version that satisfies constraint

Spack packages are simple Python

```python
from spack import *

class Dyninst(Package):
    """API for dynamic binary instrumentation.  Modify programs while they
       are executing without recompiling, re-linking, or re-executing."""

    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:")


    # new version uses cmake
    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',
                  '-DLIBELF_INCLUDE_DIR=%s'   % join_path(libelf.include, 'libelf'),
                  '-DLIBELF_LIBRARIES=%s'     % join_path(libelf.lib, 'libelf.so'),
                  '-DLIBDWARF_INCLUDE_DIR=%s' % libdwarf.include,
                  '-DLIBDWARF_LIBRARIES=%s'   % join_path(libdwarf.lib, 'libdwarf.so'),
                  *std_cmake_args)
            make()
            make("install")

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

Metadata

Version/URLs

Dependencies

Patches (not shown)

Commands for install

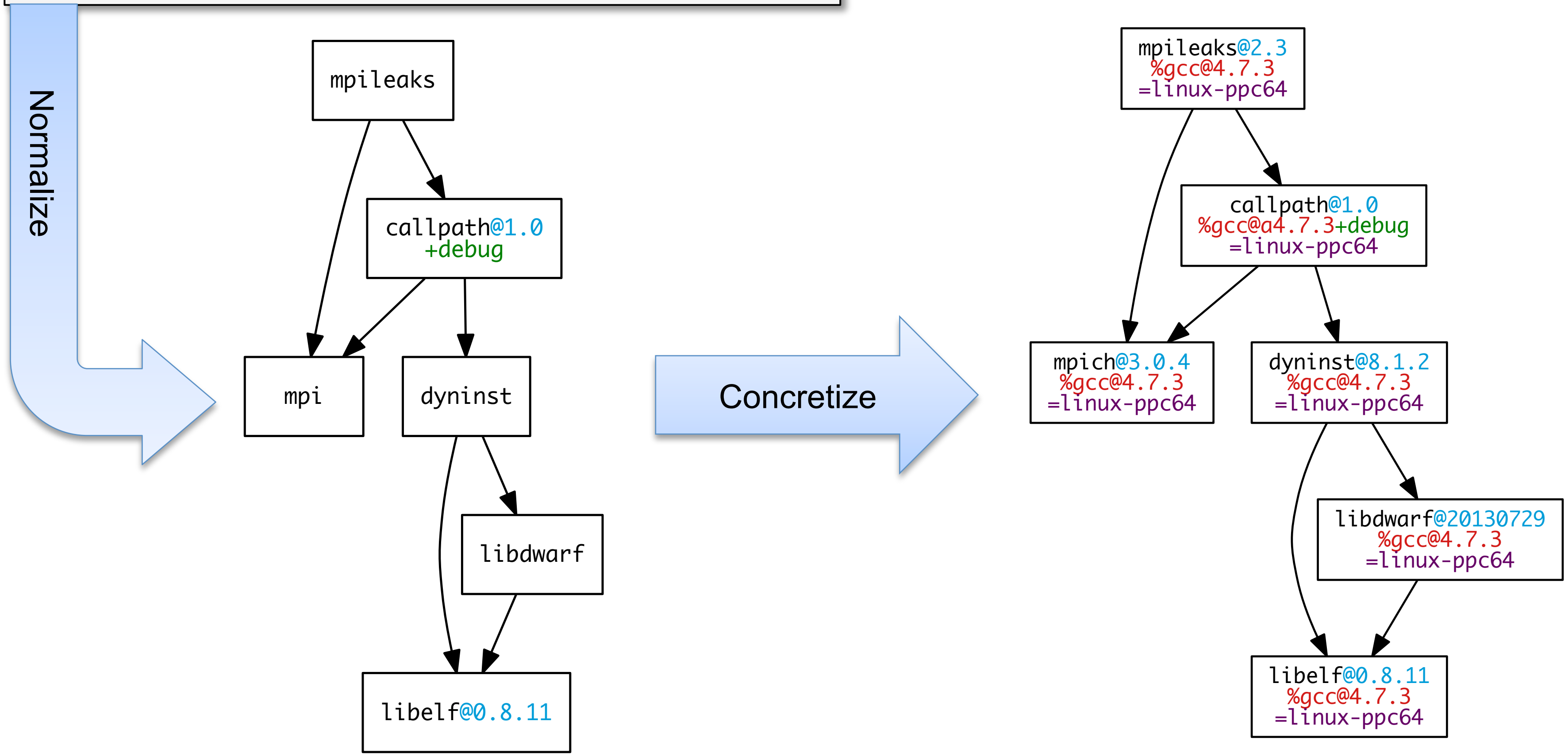
Access build config through *spec.*

- Package files live in repositories.
- ‘spack create’ command generates boilerplate package given a URL.

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

User input: *abstract* spec with some constraints

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



Abstract, normalized spec has all dependencies.

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

Spack supports optional dependencies

- Based on user-enabled variants:

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

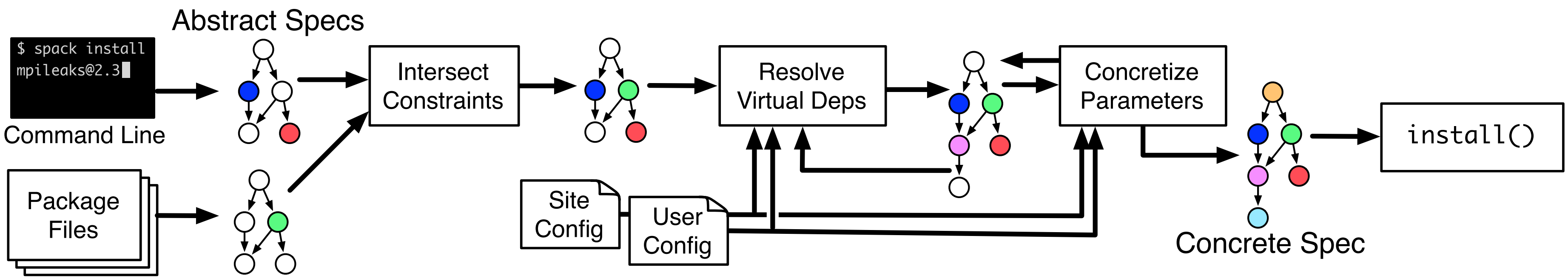
```
spack install vim +python
```

- And according to other spec conditions
  e.g., gcc dependency on mpc from 4.5 on:

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

- DAG is not always complete before concretization!

Full concretization algorithm iterates until DAG does not change



- Current algorithm is greedy
  - Will not backtrack once a decision is made.
- Can fail to find a build that satisfies user's query
  - Haven't seen this actually happen for current packages
- Really needs a full constraint solver (coming soon!)

Spack builds each package in an isolated environment

1. **Concretize the spec to be built**
2. **Fork a new process.**
3. **Set CC, CXX, F77, FC to Spack compiler wrappers.**
   - Builds that don't respect these must be patched by package authors (typically an easy Makefile fix)
4. **Set parameters for compiler wrappers as environment variables.**
   - SPACK_CC, SPACK_CXX, SPACK_F77, SPACK_FC → paths to real compilers
5. **Set env variables so that dependencies are found:**
   - PATH, PKG_CONFIG_PATH, CMAKE_PREFIX_PATH, LIBRARY_PATH, etc.
6. **During install(), compiler wrappers add flags for deps and RPATHs automatically:**

```
-I /dep/prefix/include
-L /dep/prefix/lib
-Wl,-rpath=/dep/prefix/lib
```

- **Environment allows compilers to be swapped on demand**
- **Flags & vars allow dependencies to be found automatically by build systems**
- **RPATHs ensure that package runs *regardless of end-user's environment***

‘spack find’ shows what’s installed

```
$ spack find
==> 103 installed packages.
-- chaos_5_x86_64_ib / 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
bzip2@1.0.6           hwloc@1.9          mesa@8.0.5          py-matplotlib@1.4.2   sqlite@3.8.5
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
cmake@3.0.2           launchmon@1.0.1    mpileaks@1.0        py-pygments@2.0.1     tcl@8.6.3
cram@1.0.1            lcms@2.6           mrnet@4.1.0         py-pyparsing@2.0.3    tk@src
dbus@1.9.0            libdrm@2.4.33      muster@1.0.1        py-pyside@1.2.2       xcb-proto@1.11
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

-- chaos_5_x86_64_ib / 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

-- chaos_5_x86_64_ib / intel@14.0.2 -----------------------------
hwloc@1.9  mpich@3.0.4  starpu@1.1.4

-- chaos_5_x86_64_ib / intel@15.0.0 -----------------------------
adept-utils@1.0.1  boost@1.55.0  libdwarf@20130729  libelf@0.8.13  mpich@3.0.4

-- chaos_5_x86_64_ib / 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
```

Multiple builds of same MPI package

```
$ spack find mpich
==> 5 installed packages.
-- chaos_5_x86_64_ib / gcc@4.4.7 --------------------------------
mpich@3.0.4

-- chaos_5_x86_64_ib / gcc@4.8.2 --------------------------------
mpich@3.0.4

-- chaos_5_x86_64_ib / intel@14.0.2 -----------------------------
mpich@3.0.4

-- chaos_5_x86_64_ib / intel@15.0.0 -----------------------------
mpich@3.0.4

-- chaos_5_x86_64_ib / intel@15.0.1 -----------------------------
mpich@3.0.4
```

Spec constraints double as a query syntax to allow refinement

```
$ spack find libelf
==> 5 installed packages.
-- chaos_5_x86_64_ib / gcc@4.4.7 ---------
libelf@0.8.12  libelf@0.8.13

-- chaos_5_x86_64_ib / gcc@4.8.2 ---------
libelf@0.8.13

-- chaos_5_x86_64_ib / intel@15.0.0 ------
libelf@0.8.13

-- chaos_5_x86_64_ib / intel@15.0.1 ------
libelf@0.8.13
```

Query versions of `libelf` package

List only those built with intel compiler.

```
$ spack find libelf %intel
-- chaos_5_x86_64_ib / intel@15.0.0 ------
libelf@0.8.13

-- chaos_5_x86_64_ib / intel@15.0.1 ------
libelf@0.8.13
```

Restrict to specific compiler version

```
$ spack find libelf %intel@15.0.1
-- chaos_5_x86_64_ib / intel@15.0.1 ------
libelf@0.8.13
```

Query full dependency configuration

```
$ spack find -d callpath
==> 2 installed packages.
-- chaos_5_x86_64_ib / gcc@4.4.7 ---------
    callpath@1.0.2-5dce4318
        ^adept-utils@1.0-5adef8da
            ^boost@1.55.0
            ^mpich@3.0.4
        ^dyninst@8.1.2-b040c20e
            ^libdwarf@20130729-b52fac98
                ^libelf@0.8.13

-- chaos_5_x86_64_ib / intel@15.0.1 -------
    callpath@1.0.2-63c842f9
        ^adept-utils@1.0.1-ae1dfc92
            ^boost@1.55.0
            ^mpich@3.0.4
        ^dyninst@8.1.2-ba05df97
            ^libdwarf@20130729-ab4816c7
                ^libelf@0.8.13
```

```
$ spack find callpath
==> 2 installed packages.
-- chaos_5_x86_64_ib / gcc@4.4.7 -------
callpath@1.0.2

-- chaos_5_x86_64_ib / intel@15.0.1 ----
callpath@1.0.2
```

Expand dependencies with spack `find -d`

- Not just architecture and compiler, but dependency versions may differ between builds.

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.

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 expose tools like Scalasca, TAU, etc. as "secondary" compiler wrappers.**
  - Allow user to build many instrumented versions of large codes, with many different compilers:

```
spack install ares@3.3 %gcc@4.7.3 +tau
```

- **LLNL PRUNER debugging tool is looking into this.**
  - Uses LLVM for instrumentation; needs to cover all libraries.

Future direction: Automatic ABI checking

- **We're starting to add the ability to link to external packages**
  - Vendor MPI
  - OS-provided packages that are costly to rebuild
- **External packages are already built, so:**
  - Can't always match compiler exactly
  - Can't always match dependency versions exactly
- **Need to guarantee that the RPATH'd version of a library is compatible with one that an external package was built with**
  - Allows more builds to succeed
  - Potentially violates ABI compatibility
- **Looking into using `libabigail` from RedHat to do some checking at install time.**

Related work

- **Most OS package managers don't handle combinatorial builds (and shouldn't)**
  - Maintain single, stable (or latest) version of most packages.
  - Allow smooth upgrades and predictable user experience.
  - Generally you pick a single compiler
- **Gentoo Prefix**
  - Based on Gentoo Linux: builds from source, installs into common prefix
  - Allows different compilers, but requires modifying packages (not parameterized)
  - Different major versions are allows, different versions allowed through multiple prefixes.
- **Nix**
  - Allows many separate configurations, packages are cryptographically hashed.
  - Multi-compiler support is limited, no virtual dependencies, no simple HPC build parameterization.
- **HPC package managers:**
  - **Smithy** (ORNL): No dependency management; only install automation
  - **EasyBuild** (HPC U. Ghent)
    - Requires a package file per configuration of software
    - Currently 3300 package config files for 600 packages (!)
  - **Hashdist**
    - Similar goals to Spack, different platform targets (small scale HPC)
    - No spec syntax, more package file and profile editing required.
    - Compiler/architecture support is limited
    - Team is implementing many Spack features now. Potential for long-term convergence

Spack has a growing community.

- **Spack is starting to be used in production at LLNL**
  - Used for tool installation at Livermore Computing (LC)
  - Used by ARES, NextGen teams, others.
  - Will enable a common deployment environment for LC and codes.
- **Spack has a growing external community**
  - Tri-labs: Participation in Spack calls by Sandia, LANL
  - Argonne, IIT, INRIA, Krell Institute, Stowers Medical Research Center
  - Recently NERSC looking at Spack for their Cori system (same arch as Trinity)
- **Sites can easily leverage efforts by sharing builds.**

- **Get Spack!**

  Github: **http://bit.ly/spack-git**

  Mailing List: **http://groups.google.com/d/spack**