Spack, etc.

James Amundson
HSF Packaging Group
February 10, 2016

Why are we here?

We have already seen many presentations of existing packaging solutions.

I really did not think we needed any more.

This group to date, in my own view

- I had hoped, *but not expected,* to see one packaging solution that was general enough to form the basis for a common system.
  - I think we were all hoping that we could benefit from work done by others.
- We have seen many systems
  - Some custom tools
  - Some pre-existing tools
- None of the tools were general enough at the core to meet all needs of the parties in this group.
  - Generally, the tools meet the needs of the people who wrote them.
    - This includes my tool (Contractor).
- Nobody has a lot of extra effort to devote to support their tools for other people.
- I was ready to agree that a set of common package descriptions was the best feasible outcome for this group.
  - May still be the case…

General requirements

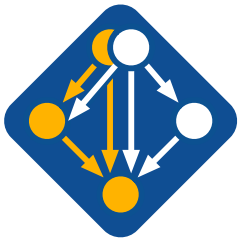
Some requirements I have seen. Not intended to be comprehensive.

- Platforms
  - Linux
    - SL, Ubuntu, other
    - x86, ARM, ?
  - Mac OSX
  - Supercomputers
    - Cray XT
    - Blue Gene
      - both involve cross-compiling
- Multiple versions
- Multiple compilers
- Multiple build variants
- Binary packaging and distribution
- Relocatable packages

Many of these requirements would be difficult to add to existing systems.

Spack

- The Supercomputing Package manager
- I first learned about Spack in November 2015.
- Information at https://github.com/LLNL/spack
- I will show a selection of slides from the talk available at https://tgamblin.github.io/files/Gamblin-Spack-SC15-Talk.pdf



The Spack Package Manager: Bringing Order to HPC Software Chaos

Supercomputing 2015 (SC15)
Austin, Texas
November 18, 2015

Todd Gamblin, Matthew LeGendre, Michael R. Collette,
Gregory L. Lee, Adam Moody, Bronis R. de Supinski, and Scott Futral

LLNL-PRES-803375
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

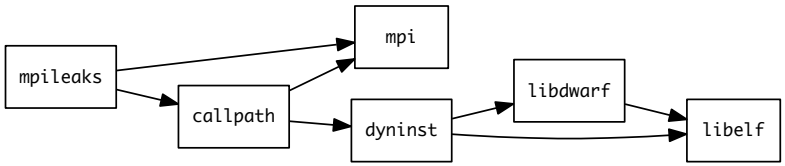
http://bit.ly/spack-git

Lawrence Livermore National Laboratory

Fermilab

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*

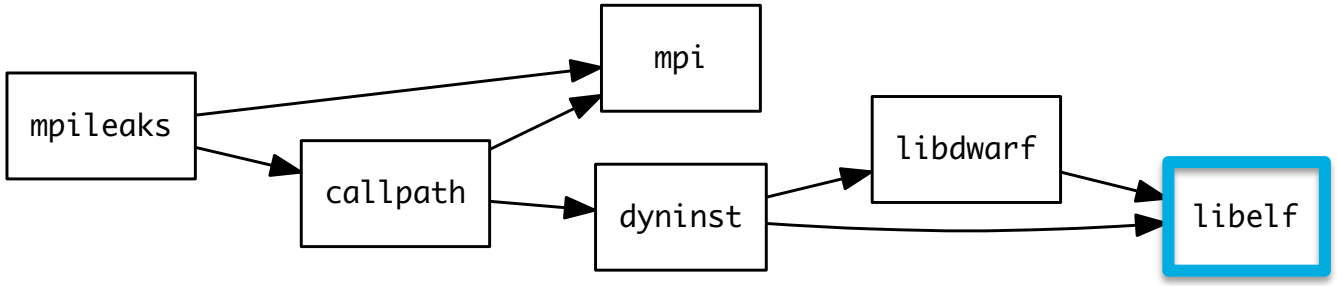
Spack provides a *spec* syntax to describe customized DAG configurations

```
$ 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

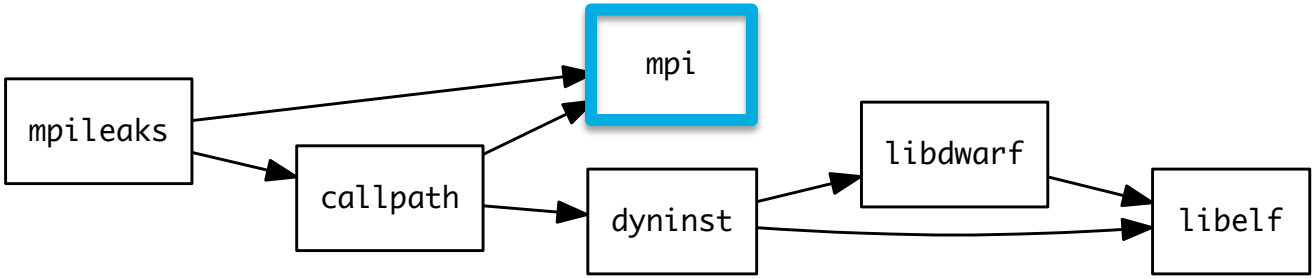
Spack Specs can constrain versions of dependencies




```
$ 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.

Dependencies in Spack may be optional.

- The user can define named *variants*:

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

- And use them to install:

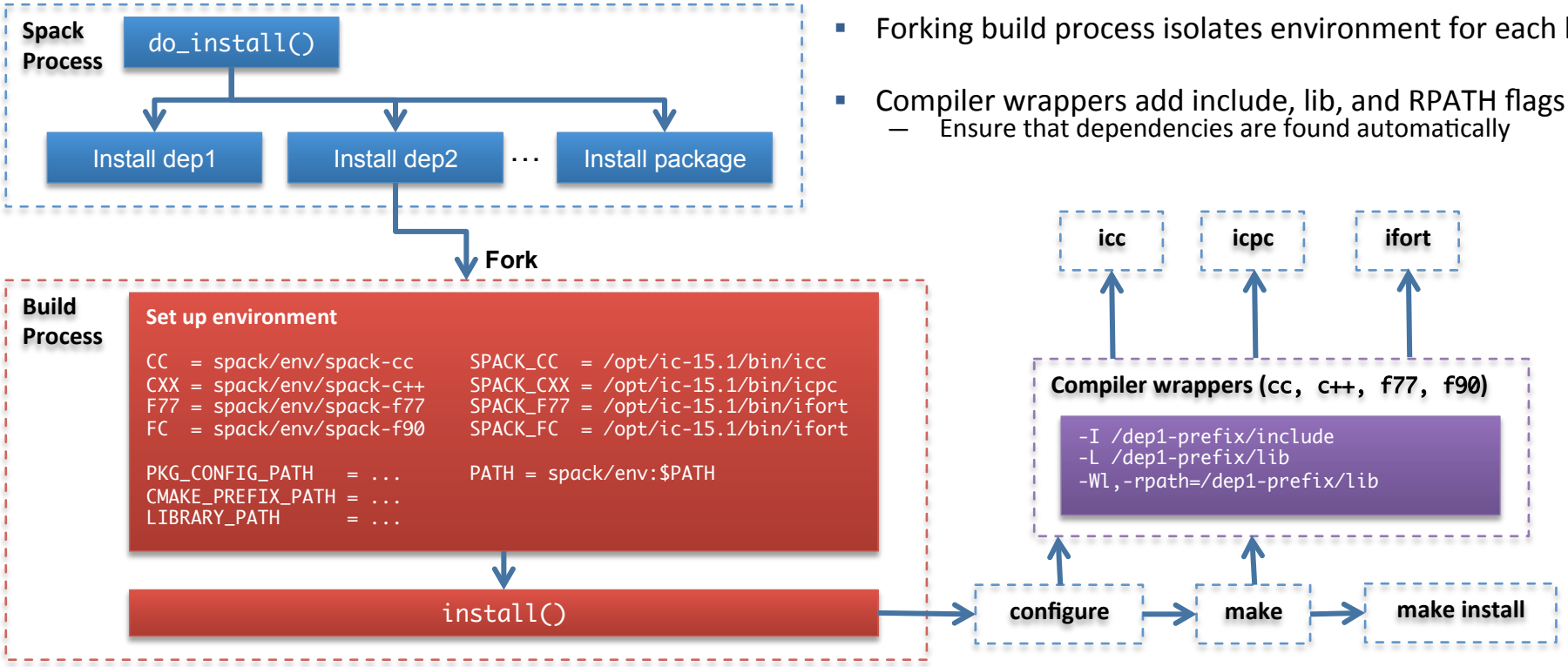
```
$ spack install vim +python
$ spack install vim –python
```

- 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!

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.

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.

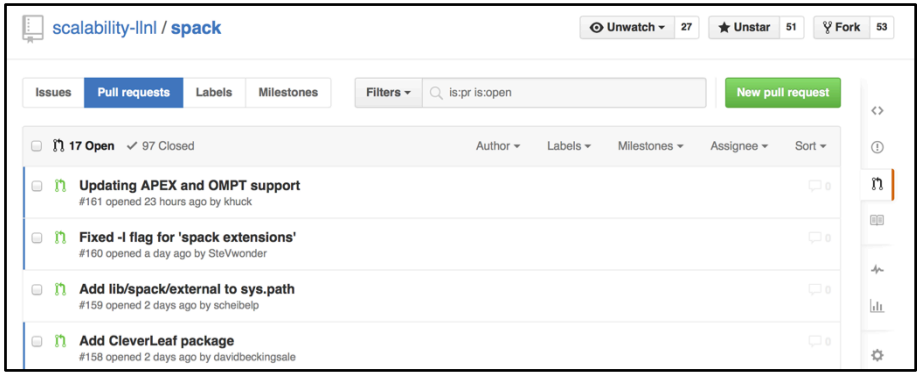
```
$ spack find libelf %intel
-- linux-x86_64 / intel@15.0.0 ------
libelf@0.8.13

-- 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

Many new feature developments are in progress

- Current:
  - Lmod hierarchy integration
  - External dependencies
    - Autodetect system MPI and other packages
  - Custom compiler flag injection
  - XML Test output (JUnit)
    - Each dependency exposed as test case
  - Better Cray environment integration

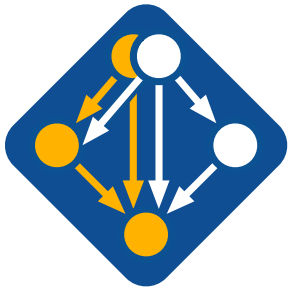


http://bit.ly/spack-git

- Planned:
  - Use compiler wrappers to apply tools to large codes
    - Klocwork, thread sanitizers, etc.
  - Dependencies on compiler features (C++11, lambdas, OpenMP versions)
  - Automatic ABI checking & upgrading

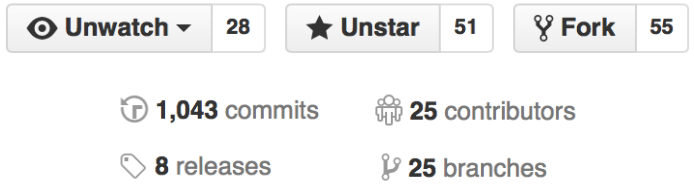
The Spack project is growing rapidly.

- Spack is flexible enough for HPC needs
  - From single users of small clusters, to large code teams on top-10 supercomputers.
- Spack is starting to be used in production at LLNL
  - Build, test, and deployment by code teams.
  - Tools, libraries, and Python at Livermore Computing.
  - Build research projects for students, postdocs.
- Spack has a rapidly growing external community.
  - NERSC is working with LLNL on Cray support for Cori.
  - Argonne/IIT cluster challenge project.
  - Kitware contributing ParaView builds & features.
  - INRIA using Spack to package MORSE numerical software
  - Users and contributors at EPFL, U. Oregon, Sandia, LANL.



Get Spack!

https://bit.ly/spack-git



My observations from working with Spack

- Spack has a polished user interface
  - spack edit foo
    - edits foo package definition
  - spack create http://compacc.fnal.gov/~amundson/garply-1.0.tar.gz
    - creates garply package file template
  - Commands make helpful suggestions
- Spack has excellent documentation
- The Spack code is very clean
  - I was able to write my first trivial extension during a meeting
  - I was able to write a non-trivial extension in a couple of hours
    - All of this was without noticing that there was developer information in the manual
- Spack has an active community
  - Google Group
  - Regular phone meetings
- The Spack authors are familiar with earlier work
  - The Spack paper cites EasyBuild, Conda, WAF, Contractor, Homebrew, etc.

Going back to earlier points

- I had hoped, *but not expected,* to see one packaging solution that was general enough to form the basis for a common system.
  - I think we were all hoping that we could benefit from work done by others.
  - Spack represents a significant amount of work done by others.
- None of the tools were general enough at the core to meet all needs of the parties in this group.
  - Spack has a very well-designed core. It solves hard problems.

Spack is not (yet) completely ready for our needs

- Requirements listed earlier:
  - Platforms
    - Linux yes
      - SL, Ubuntu, other yes
      - x86, ARM, ? yes…
    - Mac OSX mostly yes
    - Supercomputers Yes!
      - Cray XT yes
      - Blue Gene yes
        - both involve cross-compiling yes
  - Multiple versions yes
  - Multiple compilers yes
  - Multiple build variants yes
  - Binary packaging and distribution no(t in Spack proper)
  - Relocatable packages no(t in Spack proper)

Relocatability: an aside

- Relocatability is good
  - But…
    - It costs us effort to produce relocatable products
    - It costs the user effort to get consistent combinations
      - ups is our main tool at Fermilab for dealing with the user problem
      - ups relies heavily on LD_LIBRARY_PATH
- Why do we need relocatability?
  - Non-root user installs
  - CVMFS
  - Grid jobs
    - closely related to first two reasons
  - User plugins
  - Others?

Relocatability: “LD_LIBRARY_PATH”

- The quotes mean the appropriate variable for platform.
  - e.g., DYLD_LIBRARY_PATH on OS X.
- Environment variables are good for user preferences. They are fluid.
  - Good for things that are not known at compile time
- Compile A, linking against library B
  - When you run A, do you consider the location of library B a user preference? Wasn’t it known at compile time?
    - Plugins are a different story.
- “LD_LIBRARY_PATH” is not inherited in OSX El Capitan
  - Whether we like it or not

Relocatability: RPATH

- RPATH has many advantages over LD_LIBRARY_PATH
- What about relocatability?
- OS X:
  - relative RPATHs
- OS X and Linux:
  - relocation tools
    - install_name_tool (OSX), patchelf and chrpath(Linux)
    - I think these tools satisfy our actual needs.
      - Perhaps there are use cases I have not yet considered. Tell me.
- Plugins:
  - There are multiple ways to get around RPATH for plugins

My (prototype) extensions to Spack

- Binary packaging, including relocatability
  - Utilizes patchelf
  - Done at proof-of-concept level
- Export one or more packages to new directory tree
  - Also utilizes patchelf
  - Useful for production, end users, etc.
  - Work in progress

- I will happily share my modifications with interested parties
  - Not polished enough to submit to Spack proper

Spack for the future

- We are actively investigating using Spack for Fermilab software
  - Synergia
  - art
  - LArSoft
- Development work is still required
- Not yet committed
- Also interested in using Spack for the development layer
  - As opposed to purely for externals
  - Probably the subject for an entire other meeting
- I encourage this group to consider Spack for an HSF solution