



Managing HPC Software Complexity with Spack

Part of LLNL's HPCIC Tutorials (hpcic.llnl.gov)
August 6-7, 2024

Gregory Becker, Todd Gamblin, Alec Scott

 THE **LINUX** FOUNDATION



The most recent version of these slides can be found at:
<https://spack-tutorial.readthedocs.io>



Todd Gamblin
LLNL



Alec Scott
LLNL



Greg Becker
LLNL

Tutorial Materials

Find these slides and associated scripts here:

spack-tutorial.rtfd.io

We also have a chat room on Spack slack.
You can join here:

slack.spack.io

Join the **#tutorial** channel!

You can ask questions here after the conference is over.
Over **3,000 people** can help you on Slack!

The screenshot shows the homepage of the `spack-tutorial.rtfd.io` website. The header features the Spack logo and the word "Spack". Below the header is a search bar labeled "Search docs". A sidebar on the left contains links to "Main Spack Documentation", "Basic Installation Tutorial", "Configuration Tutorial", "Package Creation Tutorial", and "Developer Workflows Tutorial". The main content area includes sections for "Read the Docs" (with a dropdown menu for "latest"), "Versions" (listing "latest", "sc18", "sc17", "sc16", "riken19", "pearc19", "nsf19", "lanl19", "isc19", and "ecp19"), "Downloads", "HTML", "On Read the Docs", "Project Home", "Builds", "Downloads", "On GitHub", "View", "Edit", and "Search". At the bottom of the page is a footer with the text "Hosted by [Read the Docs](#) · [Privacy Policy](#)".

Docs » Tutorial: Spack 101

Tutorial: Spack 101

This is a full-day introduction to Practice and Experience in Ad 2019.

You can use these materials to and read the live demo scripts

Slides



Full citation
Managing HPC Software Complexity with Spack

Practice and Experience in Ad Chicago, IL, USA.

Live Demos

We provide scripts that take you sections in the slides above. You

1. We provide the [spack/tutori](#) tutorial on your local machine in the container.
2. When we host the tutu unfamiliar with Docker

You should now be ready to ru



Claim a VM instance at: bit.ly/spack-vms



	A	B	C	D	E
1	Spack Tutorial VM Instances				
2	Instructions:	1. Put your name in a box below to claim an account on a VM instance			
3		2. Log in to your VM:			
4		ssh <IP address>			
5		Login/password are both the username from your column below			
6					
7		Login / Password			
8	IP Address	spack1	spack2	spack3	spack4
9	35.90.43.21				
10	35.91.36.120				
11	34.217.149.171				
12	35.22.150.155				

ssh spack2@3.73.129.196

If you're in the spack2 column,
your login and password are
both spack2

Claim a login by putting your name in the Google Sheet



Agenda (approximate)

Tuesday, August 6

Intro	8:00 am
Basics	
Concepts	
Break	9:30 am
Environments	10:00 am
Configuration	
Lunch	11:30 am

Wednesday, August 7

Software Stacks	8:00 am
Packaging	
Break	9:30 am
Developer Workflows	10:00 am
Mirrors & Binary Caches	
Scripting	
End	11:30 am

We build codes from hundreds of small, complex pieces

Just when we're starting to solve the problem of how to create software using reusable parts, it founders on the nuts-and-bolts problems outside the software itself.

P. DuBois & T. Epperly. ***Why Johnny Can't Build***. Scientific Programming. Sep/Oct 2003.

- Component-based software development dates back to the 60's
 - M.D. McIlroy, *Mass Produced Software Components*. NATO SE Conf., 1968
- **Pros are well known:**
 - Teams can and must reuse each others' work
 - Teams write less code, meet deliverables faster
- **Cons:**
 - Teams must ensure that components work together
 - Integration burden increases with each additional library
 - Integration must be repeated with each update to components
 - **Components must be vetted!**
- **Managing changes over time is becoming intractable**



Build-time incompatibility; fail fast



Appears to work; subtle errors later

Modern scientific codes rely on icebergs of dependency libraries

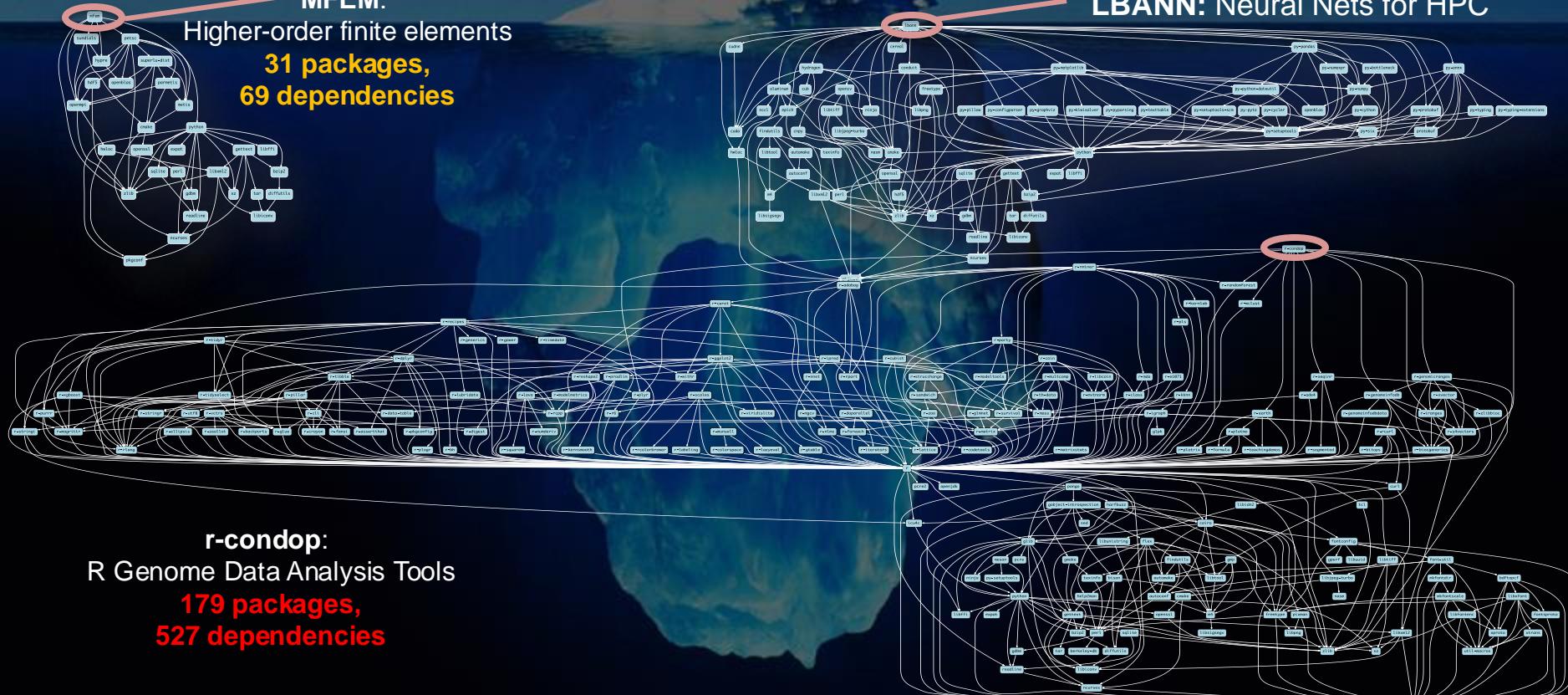
71 packages
188 dependencies

MFEM:
Higher-order finite elements

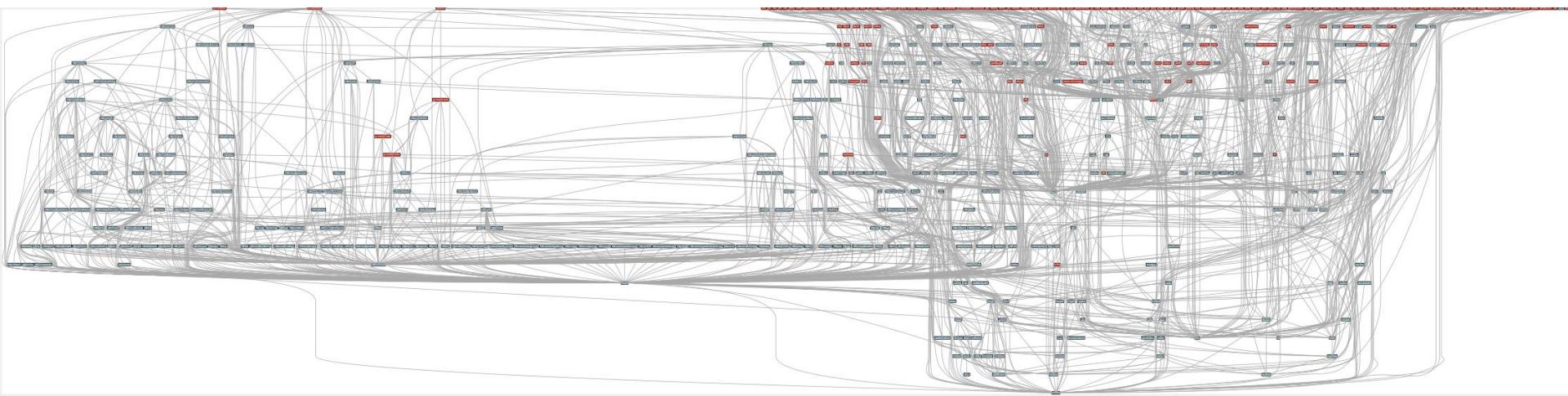
**31 packages,
69 dependencies**

LBANN: Neural Nets for HPC

r-condop: R Genome Data Analysis Tools



ECP's E4S stack is even larger than these codes

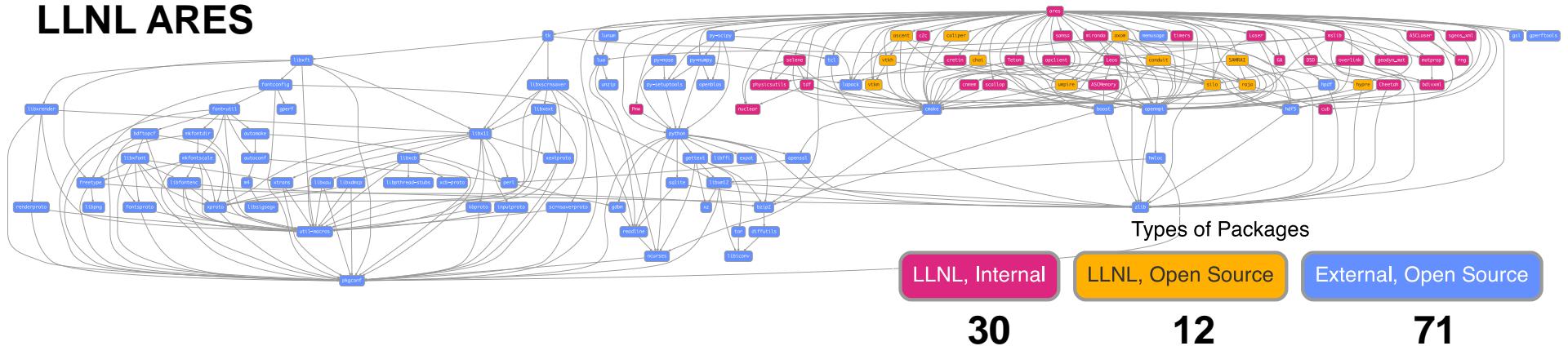


- Red boxes are the packages in it (about 100)
- Blue boxes are what *else* you need to build it (about 600)
- It's infeasible to build and integrate all of this manually



Modern software integrates open source and internal packages

LLNL ARES



- Most modern software uses *tons* of open source
- We *cannot* replace all these OSS components with our own
 - How do we put them all together effectively?
 - Do you *have* to integrate this stuff by hand?

Some fairly common (but questionable) assumptions made by package managers (conda, pip, apt, etc.)

- **1:1 relationship between source code and binary (per platform)**
 - Good for reproducibility (e.g., Debian)
 - Bad for performance optimization
- **Binaries should be as portable as possible**
 - What most distributions do
 - Again, bad for performance
- **Toolchain is the same across the ecosystem**
 - One compiler, one set of runtime libraries
 - Or, no compiler (for interpreted languages)

Outside these boundaries, users are typically on their own

High Performance Computing (HPC) violates many of these assumptions

- **Code is typically distributed as source**
 - With exception of vendor libraries, compilers
- **Often build many variants of the same package**
 - Developers' builds may be very different
 - Many first-time builds when machines are new
- **Code is optimized for the processor and GPU**
 - Must make effective use of the hardware
 - Can make 10-100x perf difference
- **Rely heavily on system packages**
 - Need to use optimized libraries that come with machines
 - Need to use host GPU libraries and network
- **Multi-language**
 - C, C++, Fortran, Python, others all in the same ecosystem

Some Supercomputers



Summit
Oak Ridge National Lab
Power9 / NVIDIA



Fugaku
RIKEN
Fujitsu/ARM a64fx



Perlmutter
Lawrence Berkeley National Lab
AMD Zen / NVIDIA



Aurora
Argonne National Lab
Intel Xeon / Xe



Frontier
Oak Ridge National Lab
AMD Zen / Radeon



El Capitan
Lawrence Livermore National Lab
AMD Zen / Radeon

What about containers?

- Containers provide a great way to reproduce and distribute an already-built software stack
- Someone needs to build the container!
 - This isn't trivial
 - Containerized applications still have hundreds of dependencies
- Using the OS package manager inside a container is insufficient
 - Most binaries are built unoptimized
 - Generic binaries, not optimized for specific architectures
- HPC containers may need to be *rebuilt* to support many different hosts, anyway.
 - Not clear that we can ever build one container for all facilities
 - Containers likely won't solve the N-platforms problem in HPC



We need something more flexible to **build** the containers

Spack enables Software distribution for HPC

- Spack automates the build and installation of scientific software
- Packages are *parameterized*, so that users can easily tweak and tune configuration

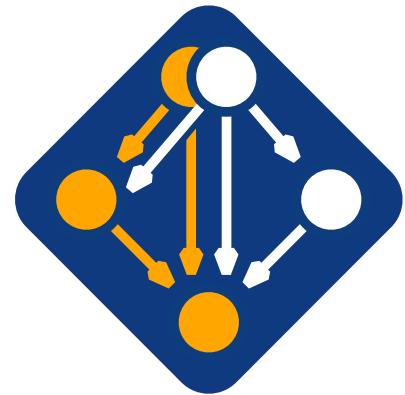
No installation required: clone and go

```
$ git clone https://github.com/spack/spack  
$ spack install hdf5
```

Simple syntax enables complex installs

```
$ spack install hdf5@1.10.5  
$ spack install hdf5@1.10.5 %clang@6.0  
$ spack install hdf5@1.10.5 +threadsafe
```

```
$ spack install hdf5@1.10.5 cppflags="-O3 -g3"  
$ spack install hdf5@1.10.5 target=haswell  
$ spack install hdf5@1.10.5 +mpi ^mpich@3.2
```



github.com/spack/spack

- Ease of use of mainstream tools, with flexibility needed for HPC
- In addition to CLI, Spack also:
 - Generates (but does **not** require) *modules*
 - Allows conda/virtualenv-like *environments*
 - Provides many devops features (CI, container generation, more)

What's a package manager?

- Does not replace Cmake/Autotools
- Manages ***dependencies***
 - Drives package-level build systems
 - Ensures **consistency** and **compatibility** among builds of packages in the ecosystem
- Stores **community knowledge**
 - Cache of package build recipes
 - Determining magic configure lines takes time

Package Manager

- Package installation
- Dependency relationships, conflicts
- May drive package-level build systems

High Level Build System

- Cmake, Autotools
- Handle library abstractions
- Generate Makefiles, etc.

Low Level Build System

- Make, Ninja
- Handles dependencies among *commands* in a single build



Who can use Spack?

People who want to use or distribute software for HPC!

1. End Users of HPC Software

- Install and run HPC applications and tools

2. HPC Application Teams

- Manage third-party dependency libraries

3. Package Developers

- People who want to package their own software for distribution

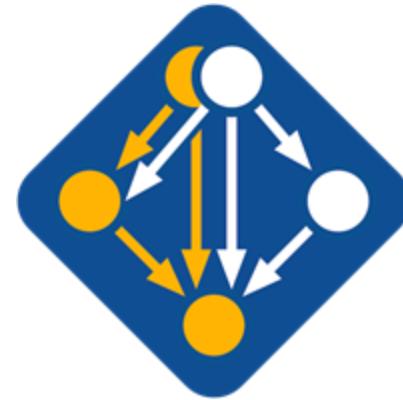
4. User support teams at HPC Centers

- People who deploy software for users at large HPC sites



Spack is officially a Linux Foundation Project now!

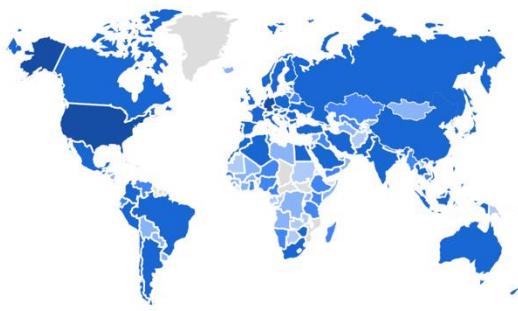
- Spack has a legal entity: a 501(c)(6) non-profit company
 - This is a neutral legal entity
 - Can be in legal agreements (e.g. for distributing binaries)
 - Can get discounts on, e.g., Slack
- Spack will have a Technical Steering Committee (TSC)
 - Plan is to make the main developer meetings more public
 - Also have official steering committee meetings
 - Working on initial GOVERNANCE.md, initial TSC members
- *Trademark* (Spack name, logo) assigned to LF
- Project website, GitHub org now co-owned by LF
- Part of the High Performance Software Foundation (HPSF) in LF



 **HIGH PERFORMANCE SOFTWARE FOUNDATION**

 **THE LINUX FOUNDATION**

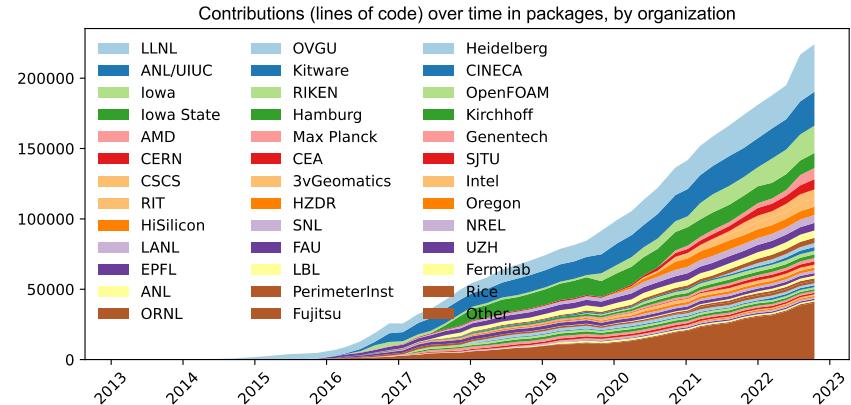
Spack sustains the HPC software ecosystem with the help of many contributors



COUNTRY	USERS
United States	23K
Germany	5.3K
China	4.6K
India	4.5K
United Kingdom	3.3K
France	3K
Japan	2.4K

2023 aggregate documentation user counts from GA4
(note: yearly user counts are almost certainly too large)

Over 7,900 software packages
Over 1,300 contributors



Contributors continue to grow worldwide!



Spack's widespread adoption has made it a de facto standard, drawing contribution and collaboration from vendors

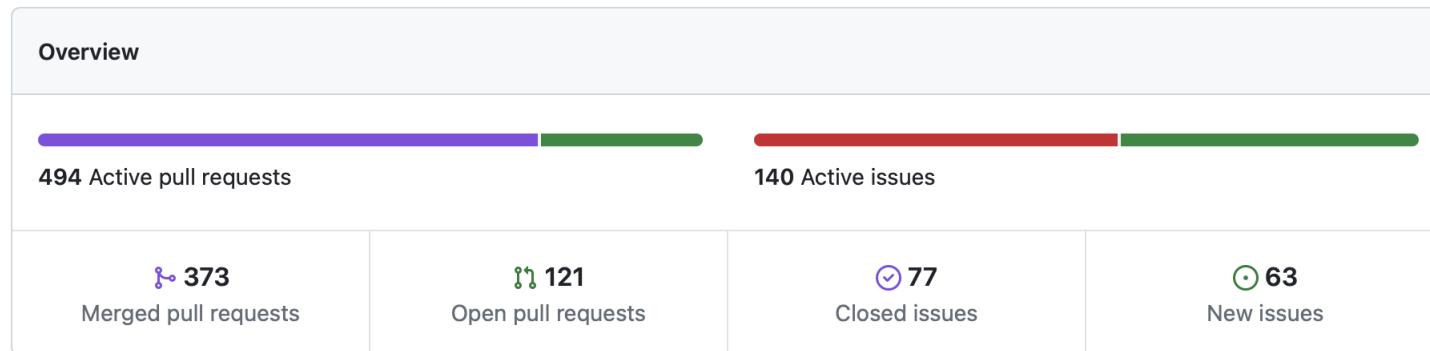
- **AWS** is investing significantly in cloud credits for Spack
 - Supporting highly scalable cloud CI system with ~250k+/year in credits
 - Integrating Spack with ParallelCluster product
 - Joint Spack tutorial with AWS drew 125+ participants
- **Google** is using Spack in their HPC Toolkit cloud cluster product
 - List packages to deploy; automatically built and cached in cluster deployment
- **AMD** has contributed ROCm packages and compiler support
 - 55+ PRs mostly from AMD, also others
 - ROCm, HIP, aocc packages are all in Spack now
- **HPE/Cray** is allowing us to do CI in the cloud for the Cray PE environment
 - Looking at tighter Spack integration with Cray PE
- **Intel** contributing OneAPI support and licenses for our build farm
- **NVIDIA** contributing NVHPC compiler support and other features
- **Fujitsu and RIKEN** have contributed a **huge** number of packages for ARM/a64fx support on Fugaku
- **ARM** and **Linaro** members contributing ARM support
 - 400+ pull requests for ARM support from various companies



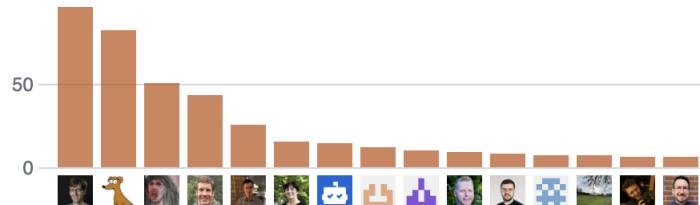
One month of Spack development is pretty busy!

June 20, 2024 – July 20, 2024

Period: 1 month ▾



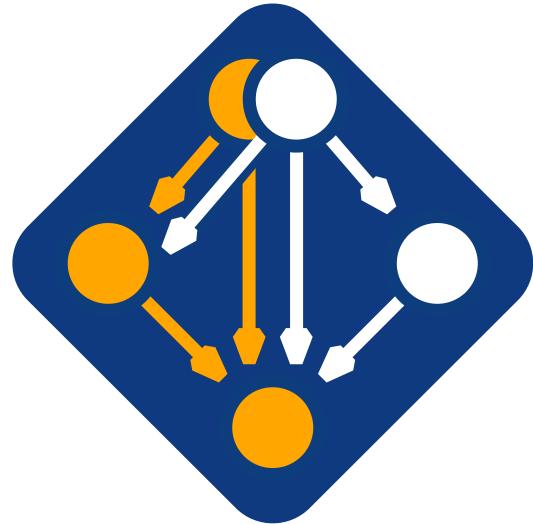
Excluding merges, **130 authors** have pushed **372 commits** to develop and **530 commits** to all branches. On develop, **4,038 files** have changed and there have been **17,703 additions** and **7,655 deletions**.



Spack v0.22.0 and v0.22.1 are out!

Highlights:

1. Compiler runtime dependencies
 - gcc-runtime, intel-oneapi-runtime, libgfortran, libc
 - OS compatibility on linux now uses libc version, not OS tag
2. Improved spack find UI for Environments
3. Improved command-line string quoting
4. Revert default spack install behavior to --reuse
5. More control over reused specs
6. New redistribute() directive
7. New conflict: and prefer: syntax for package preferences
8. include_concrete: in environments
9. python-venv isolation



github.com/spack/spack

Full release notes:

<https://github.com/spack/spack/releases/tag/v0.22.0>

<https://github.com/spack/spack/releases/tag/v0.22.1>

Spack is not the only HPC/AI/data science package manager out there



1. “Functional” Package Managers

- Nix
- Guix

<https://nixos.org>
<https://hpc.guix.info>



2. Build-from-source Package Managers

- Homebrew, LinuxBrew
- MacPorts
- Gentoo

<https://brew.sh>
<https://www.macports.org>
<https://gentoo.org>

Other tools in the HPC Space:



▪ Easybuild

- An installation tool for HPC
- Focused on HPC system administrators – different package model from Spack
- Relies on a fixed software stack – harder to tweak recipes for experimentation

<https://easybuild.io>



▪ Conda / Mamba / Pixi

- Very popular binary package ecosystem for data science
- Not targeted at HPC; generally has unoptimized binaries

<https://conda.io>
<https://mamba.readthedocs.io>
<https://prefix.dev>

Claim a VM instance at: bit.ly/spack-vms



	A	B	C	D	E
1	Spack Tutorial VM Instances				
2	Instructions:	1. Put your name in a box below to claim an account on a VM instance			
3		2. Log in to your VM:			
4		ssh <IP address>			
5		Login/password are both the username from your column below			
6					
7		Login / Password			
8	IP Address	spack1	spack2	spack3	spack4
9	35.90.43.21				
10	35.91.36.120				
11	34.217.149.171				
12	35.22.150.155				

ssh spack2@3.73.129.196

If you're in the spack2 column,
your login and password are
both spack2

Claim a login by putting your name in the Google Sheet



Hands-on Time: Spack Basics

Follow script at spack-tutorial.readthedocs.io



Core Spack Concepts



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 provides a *spec* syntax to describe customized package 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"  set compiler flags
$ spack install mpileaks@3.3 target=cascadelake  set target microarchitecture
$ spack install mpileaks@3.3 ^mpich@3.2 %gcc@4.9.3  ^ dependency constraints
```

- 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 packages are *parameterized* using the spec syntax

Python DSL defines many ways to build

```
from spack import *

class Kripke(CMakePackage):
    """Kripke is a simple, scalable, 3DSn deterministic particle transport mini-app."""

    homepage = "https://computation.llnl.gov/projects/co-design/kripke"
    url    = "https://computation.llnl.gov/projects/co-design/download/kripke-openmp-1.1.tar.gz"

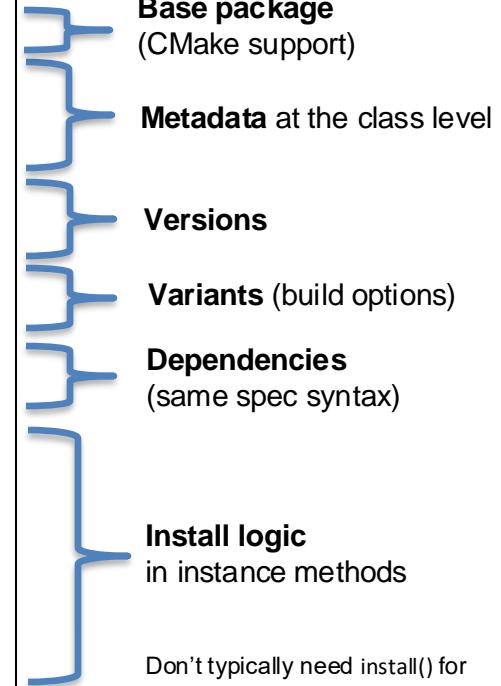
    version('1.2.3', sha256='3f7f2eef0d1ba5825780d626741eb0b3f026a096048d7ec4794d2a7dfbe2b8a6')
    version('1.2.2', sha256='eaf9ddf562416974157b34d00c3a1c880fc5296fce2aa2efa039a86e0976f3a3')
    version('1.1', sha256='232d74072fc7b848fa2adc8a1bc839ae8fb5f96d50224186601f55554a25f64a')

    variant('mpi', default=True, description='Build with MPI.')
    variant('openmp', default=True, description='Build with OpenMP enabled.')

    depends_on('mpi', when='+mpi')
    depends_on('cmake@3.0:', type='build')

    def cmake_args(self):
        return [
            '-DENABLE_OPENMP=%s' % ('+openmp' in self.spec),
            '-DENABLE_MPI=%s' % ('+mpi' in self.spec),
        ]

    def install(self, spec, prefix):
        mkdirp(prefix.bin)
        install('../spack-build/kripke', prefix.bin)
```



One package.py file per software project!



Conditional variants simplify packages

CudaPackage: a mix-in for packages that use CUDA

```
class CudaPackage(PackageBase):
    variant('cuda', default=False,
            description='Build with CUDA')

    variant('cuda_arch',
            description='CUDA architecture',
            values=any_combination_of(cuda_arch_values),
            when='+cuda')

    depends_on('cuda', when='+cuda')

    depends_on('cuda@9.0:',      when='cuda_arch=70')
    depends_on('cuda@9.0:',      when='cuda_arch=72')
    depends_on('cuda@10.0:',     when='cuda_arch=75')

    conflicts('%gcc@9:', when='+cuda ^cuda@:10.2.89 target=x86_64:')
    conflicts('%gcc@9:', when='+cuda ^cuda@:10.1.243 target=ppc64le:')
```

cuda is a variant (build option)

cuda_arch is only present if cuda is enabled

dependency on cuda, but only if cuda is enabled

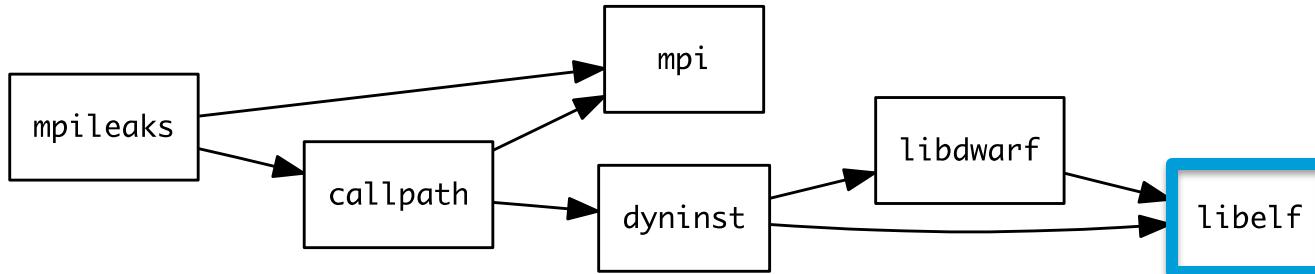
constraints on cuda version

compiler support for x86_64 and ppc64le

There is a lot of expressive power in the Spack package DSL.



Spack Specs can constrain versions of dependencies

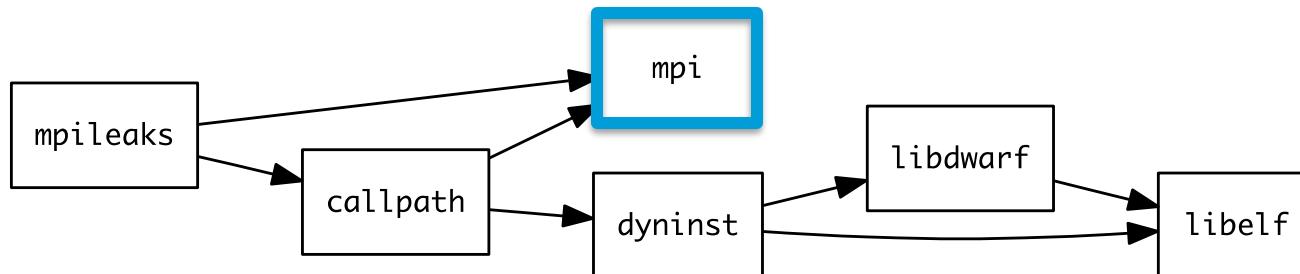


```
$ spack install mpileaks %intel@12.1 ^libelf@0.8.12
```

- 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 implementation, as long as it provides MPI 2 interface:

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

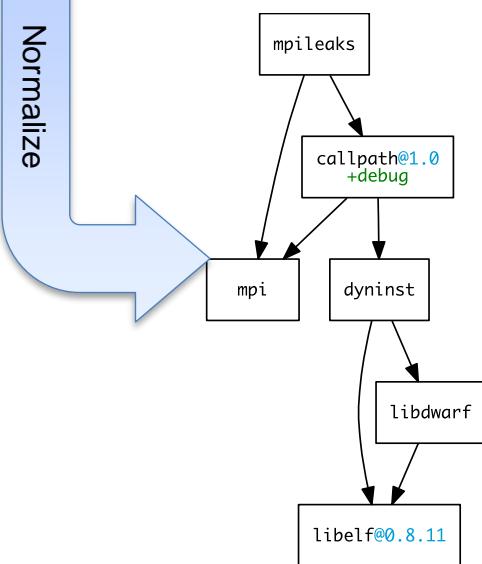


Concretization fills in missing configuration details when the user is not explicit.

mpileaks ^callpath@1.0+debug ^libelf@0.8.11

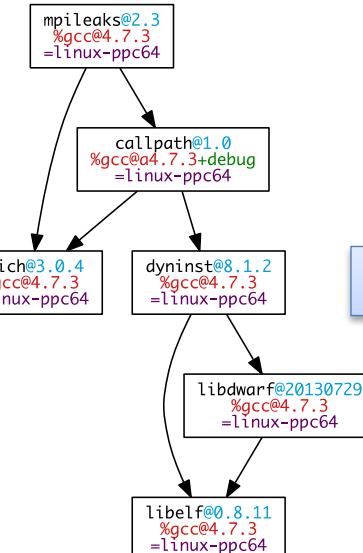
User input: abstract spec with some constraints

spec.json



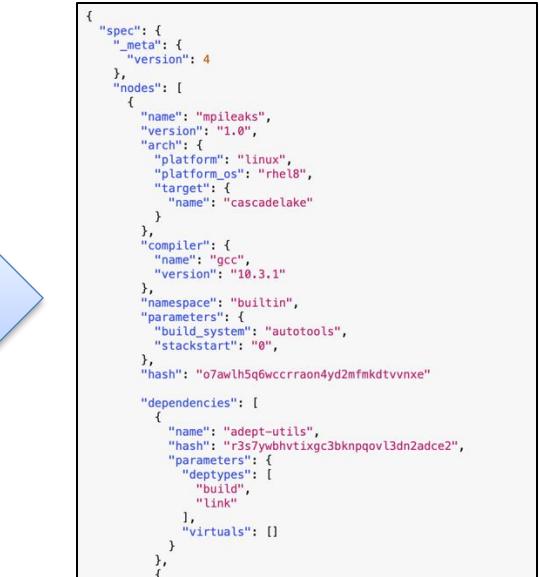
Abstract, normalized spec with some dependencies

Normalize



Concrete spec is fully constrained and can be passed to install

Concretize

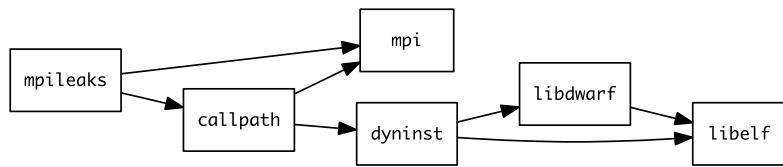


Detailed provenance stored with installed package



Spack handles combinatorial software complexity

Dependency DAG



Installation Layout

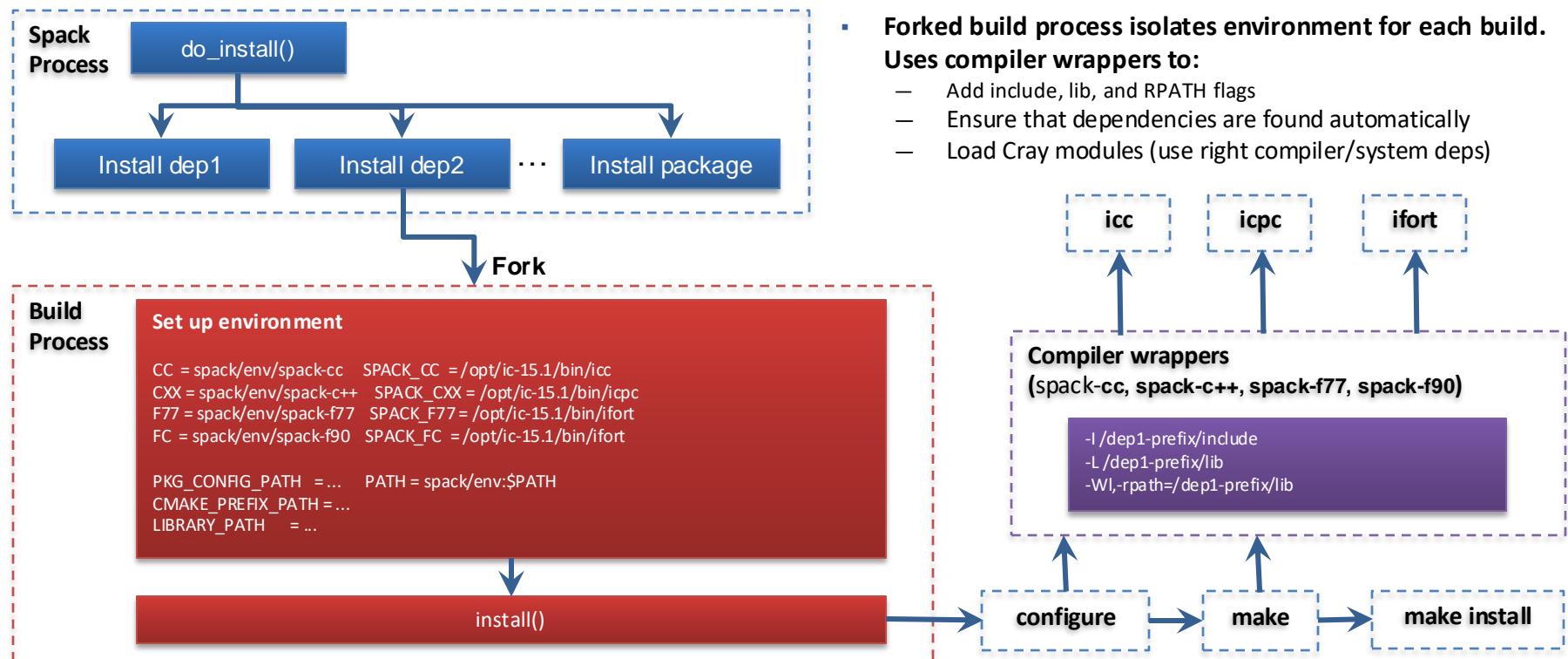
Hash

```
opt
└── spack
    ├── linux-rhel7-skylake
    │   └── gcc-8.3.0
    │       ├── mpileaks-1.0-hc4sm4vuzpm4znmvrfzri4ow2mkphe2e
    │       ├── callpath-1.0.4-daqqpssxb6qbfrztsekmhus3xoflpsy
    │       ├── openmpi-4.1.4-u64v26igvxxyyn23hysmklfums6tgjv5r
    │       ├── dyninst-12.1.0-u64v26igvxxyyn23hysmklfums6tgjv5r
    │       ├── libdwarf-20180129-u5eawkvaoc7vonabe6nnndkcfwuv233cj
    │       └── libelf-0.8.13-x46q4wm46ay4pltrijbgizxjrhbaka6
```

- Each unique dependency graph is a unique **configuration**.
- Each configuration in a unique directory.
 - Multiple 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*



An isolated compilation environment allows Spack to easily swap compilers

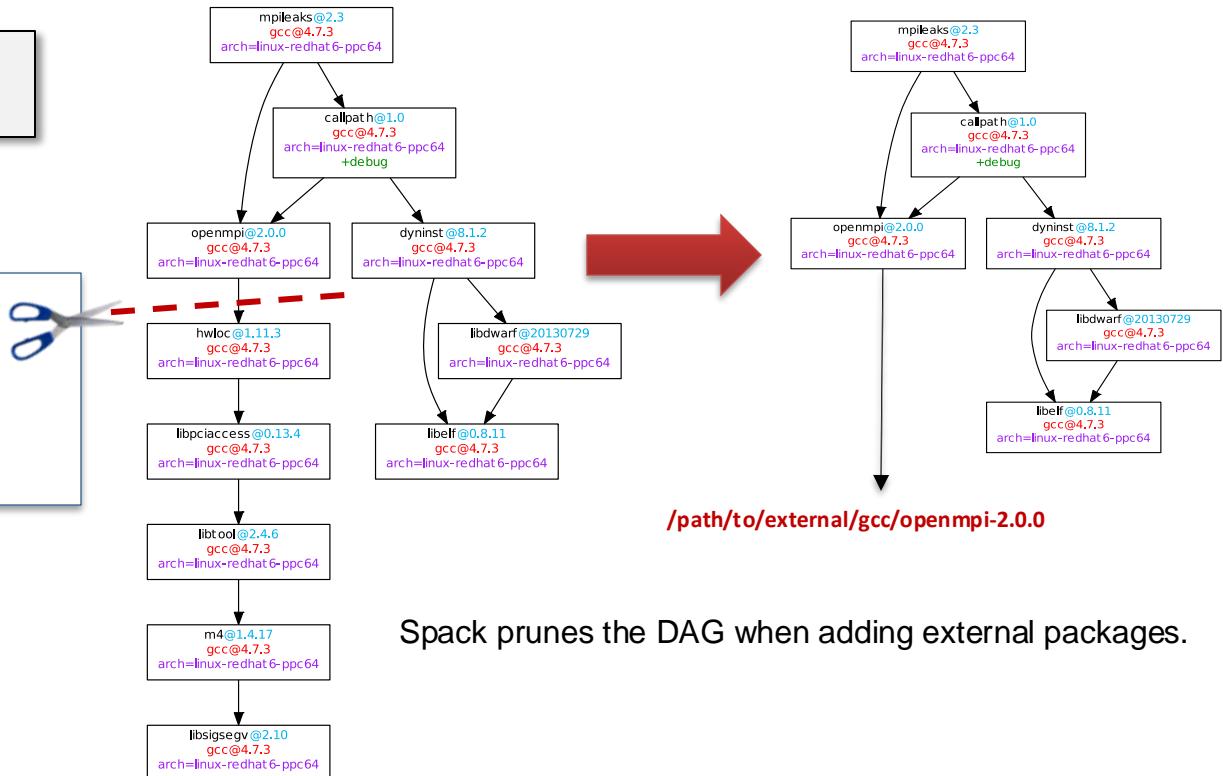


We can configure Spack to build with external software

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

packages.yaml

```
packages:  
  mpi:  
    buildable: False  
  paths:  
    openmpi@2.0.0 %gcc@4.7.3 arch=linux-rhel6-ppc64:  
      /path/to/external/gcc/openmpi-2.0.0  
    openmpi@1.10.3 %gcc@4.7.3 arch=linux-rhel6-ppc64:  
      /path/to/external/gcc/openmpi-1.10.3  
    ...
```



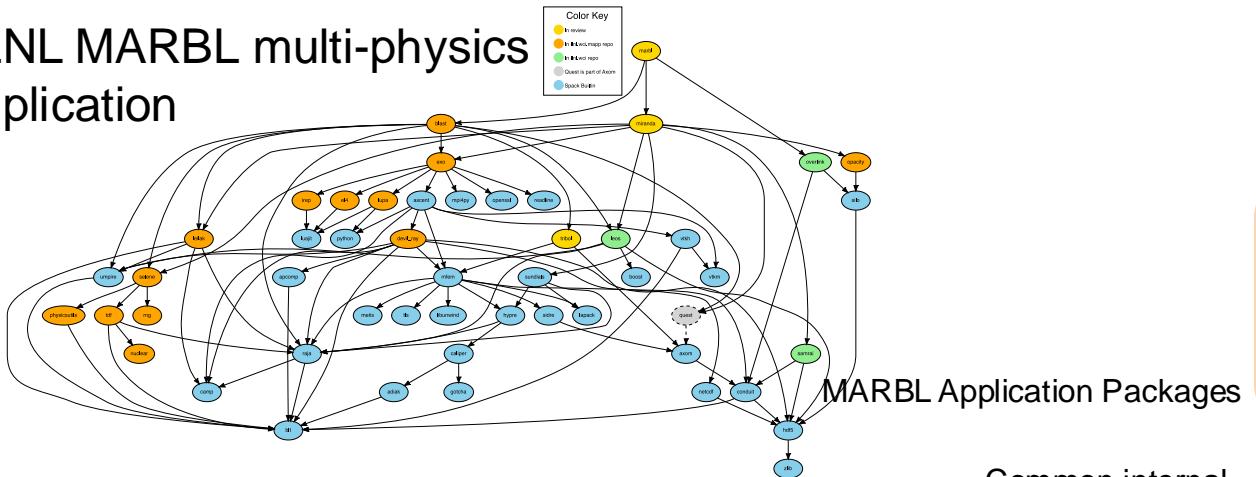
Users register external packages in a configuration file (more on these later).

Spack prunes the DAG when adding external packages.

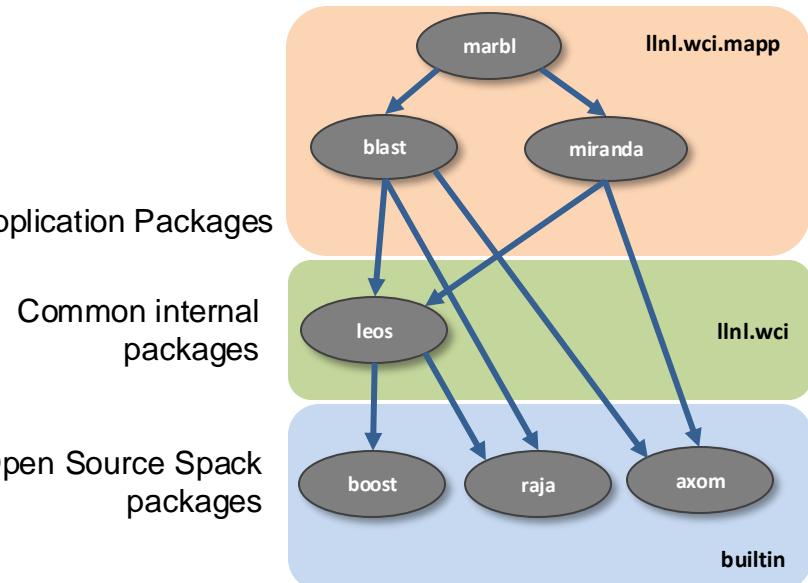


Spack package repositories allow stacks to be layered

LLNL MARBL multi-physics application

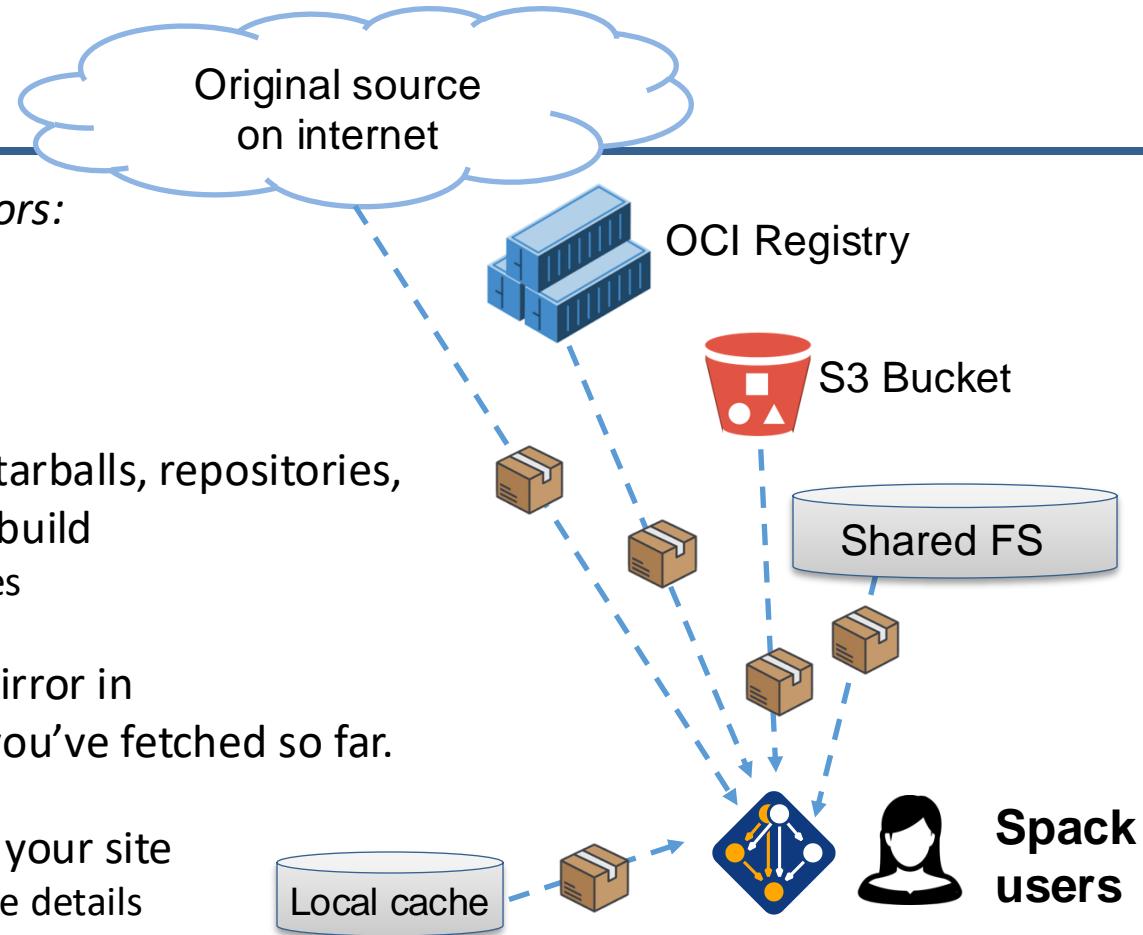


```
$ spack repo create /path/to/my_repo  
$ spack repo add my_repo  
$ spack repo list  
==> 2 package repositories.  
my_repo  /path/to/my_repo  
builtin  spack/var/spack/repos/builtin
```



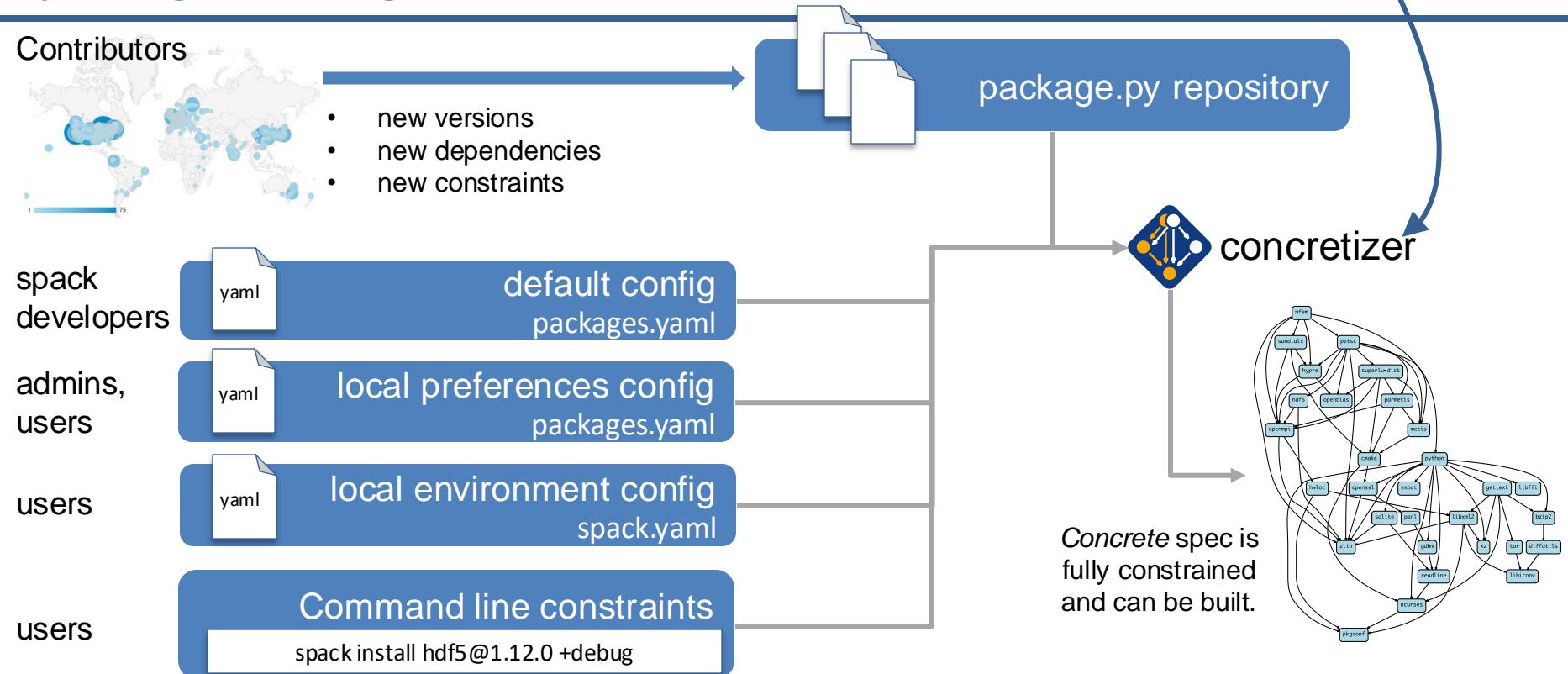
Spack mirrors

- Spack allows you to define *mirrors*:
 - Directories in the filesystem
 - On a web server
 - In an S3 bucket
- Mirrors are archives of fetched tarballs, repositories, and other resources needed to build
 - Can also contain binary packages
- By default, Spack maintains a mirror in `var/spack/cache` of everything you've fetched so far.
- You can host mirrors internal to your site
 - See the documentation for more details



The concretizer includes information from packages, configuration, and CLI

Dependency solving is NP-hard



*Concrete spec is
fully constrained
and can be built.*



We use logic programming to simplify package solving

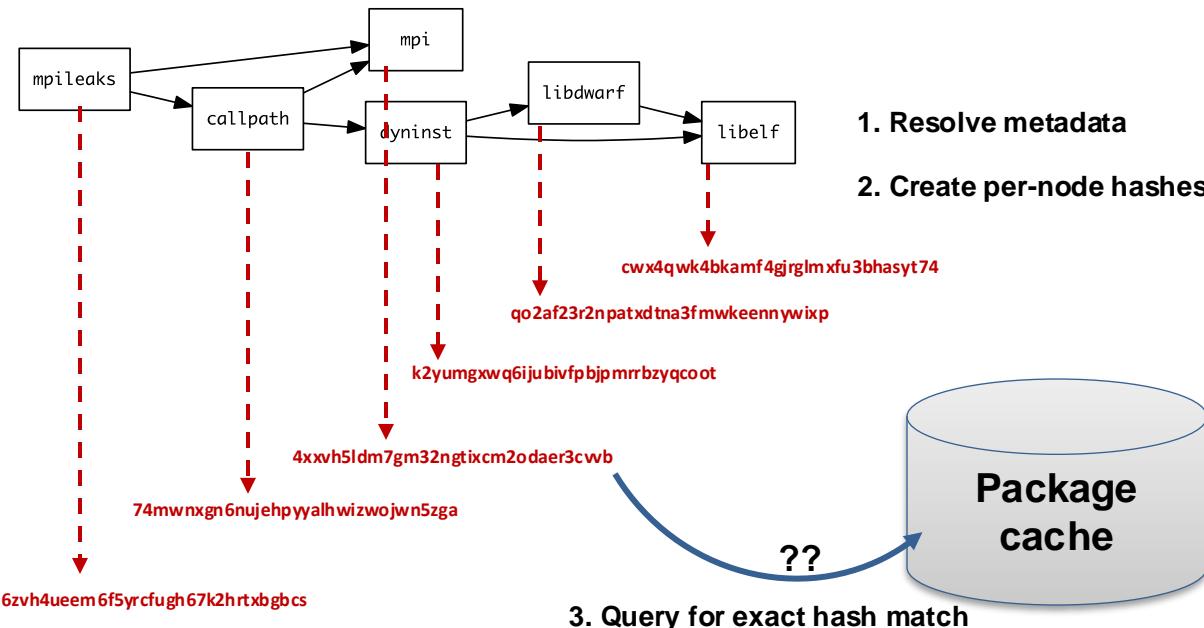
- New concretizer leverages Clingo (see potassco.org)
- Clingo is an Answer Set Programming (ASP) solver
 - ASP looks like Prolog; leverages SAT solvers for speed/correctness
 - ASP program has 2 parts:
 1. Large list of facts generated from our package repositories and config
 2. Small logic program (~800 lines)
 - includes constraints and optimization criteria
- New algorithm on the Spack side is conceptually simpler:
 - Generate facts for all possible dependencies, send to logic program
 - Optimization criteria express preferences more clearly
 - Build a DAG from the results
- New concretizer solves many specs that old concretizer can't
 - Backtracking is a huge win – many issues resolved
 - Conditional logic that was complicated before is now much easier

```
%-----  
% Package: ucx  
%-----  
version_declared("ucx", "1.6.1", 0).  
version_declared("ucx", "1.6.0", 1).  
version_declared("ucx", "1.5.2", 2).  
version_declared("ucx", "1.5.1", 3).  
version_declared("ucx", "1.5.0", 4).  
version_declared("ucx", "1.4.0", 5).  
version_declared("ucx", "1.3.1", 6).  
version_declared("ucx", "1.3.0", 7).  
version_declared("ucx", "1.2.2", 8).  
version_declared("ucx", "1.2.1", 9).  
version_declared("ucx", "1.2.0", 10).  
  
variant("ucx", "thread_multiple").  
variant_single_value("ucx", "thread_multiple").  
variant_default_value("ucx", "thread_multiple", "False").  
variant_possible_value("ucx", "thread_multiple", "False").  
variant_possible_value("ucx", "thread_multiple", "True").  
  
declared_dependency("ucx", "numactl", "build").  
declared_dependency("ucx", "numactl", "link").  
node("numactl") :- depends_on("ucx", "numactl"), node("ucx").  
  
declared_dependency("ucx", "rdma-core", "build").  
declared_dependency("ucx", "rdma-core", "link").  
node("rdma-core") :- depends_on("ucx", "rdma-core"), node("ucx").  
  
%-----  
% Package: util-linux  
%-----  
version_declared("util-linux", "2.29.2", 0).  
version_declared("util-linux", "2.29.1", 1).  
version_declared("util-linux", "2.25", 2).  
  
variant("util-linux", "libuuid").  
variant_single_value("util-linux", "libuuid").  
variant_default_value("util-linux", "libuuid", "True").  
variant_possible_value("util-linux", "libuuid", "False").  
variant_possible_value("util-linux", "libuuid", "True").  
  
declared_dependency("util-linux", "pkgconfig", "build").  
declared_dependency("util-linux", "pkgconfig", "link").  
node("pkgconfig") :- depends_on("util-linux", "pkgconfig"), node("util-linux").  
  
declared_dependency("util-linux", "python", "build").  
declared_dependency("util-linux", "python", "link").  
node("python") :- depends_on("util-linux", "python"), node("util-linux").
```

Some facts for the HDF5 package



--fresh only reuses builds if hashes match



- Hash matches are very sensitive to small changes
- In many cases, a satisfying cached or already installed spec can be missed
- Nix, Spack, Guix, Conan, and others reuse this way



--reuse (the default) is more aggressive

- --reuse tells the solver about all the installed packages!
- Add constraints for all installed packages, with their hash as the associated ID:

```
installed_hash("openssl", "lwatuuysmwkhuahrncywvn77icdhs6mn").  
imposed_constraint("lwatuuysmwkhuahrncywvn77icdhs6mn", "node", "openssl").  
imposed_constraint("lwatuuysmwkhuahrncywvn77icdhs6mn", "version", "openssl", "1.1.1g").  
imposed_constraint("lwatuuysmwkhuahrncywvn77icdhs6mn", "node_platform_set", "openssl", "darwin").  
imposed_constraint("lwatuuysmwkhuahrncywvn77icdhs6mn", "node_os_set", "openssl", "catalina").  
imposed_constraint("lwatuuysmwkhuahrncywvn77icdhs6mn", "node_target_set", "openssl", "x86_64").  
imposed_constraint("lwatuuysmwkhuahrncywvn77icdhs6mn", "variant_set", "openssl", "systemcerts", "True").  
imposed_constraint("lwatuuysmwkhuahrncywvn77icdhs6mn", "node_compiler_set", "openssl", "apple-clang").  
imposed_constraint("lwatuuysmwkhuahrncywvn77icdhs6mn", "node_compiler_version_set", "openssl", "apple-clang", "12.0.0").  
imposed_constraint("lwatuuysmwkhuahrncywvn77icdhs6mn", "concrete", "openssl").  
imposed_constraint("lwatuuysmwkhuahrncywvn77icdhs6mn", "depends_on", "openssl", "zlib", "build").  
imposed_constraint("lwatuuysmwkhuahrncywvn77icdhs6mn", "depends_on", "openssl", "zlib", "link").  
imposed_constraint("lwatuuysmwkhuahrncywvn77icdhs6mn", "hash", "zlib", "x2anksgssxsxa7pcnhzg5k3dhgacglze").
```



Telling the solver to minimize builds is surprisingly simple in ASP

1. Allow the solver to *choose* a hash for any package:

```
{ hash(Package, Hash) : installed_hash(Package, Hash) } 1 :- node(Package).
```

2. Choosing a hash means we impose its constraints:

```
impose(Hash) :- hash(Package, Hash).
```

3. Define a build as something *without* a hash:

```
build(Package) :- not hash(Package, _), node(Package).
```

4. Minimize builds!

```
#minimize { 1@100,Package : build(Package) }.
```



With and without --reuse optimization

```
(spackle):solver> spack solve -Il hdf5
=> Best of 9 considered solutions.
=> Optimization Criteria:
  Priority Criterion           Installed  ToBuild
  1   number of packages to build (vs. reuse)      -    20
  2   deprecated versions used                   0    0
  3   version weight                           0    0
  4   number of non-default variants (roots)     0    0
  5   preferred providers for roots            0    0
  6   default values of variants not being used (roots) 0    0
  7   number of non-default variants (non-roots) 0    0
  8   preferred providers (non-roots)          0    0
  9   compiler mismatches                     0    0
 10  OS mismatches                          0    0
 11  non-preferred OS's                      0    0
 12  version badness                        0    2
 13  default values of variants not being used (non-roots) 0    0
 14  non-preferred compilers                 0    0
 15  target mismatches                     0    0
 16  non-preferred targets                  0    0

- zznfgf3 hdf5@1.10.7%apple-clang@13.0.0-cxx-fortran-hl-ip0-java+mpi+shared-szip-threadsafe+tools api=default b
- nslvlog ^cmake@3.21.4%apple-clang@13.0.0-doc+ncurses+openssl+owlbins+qt build_type=Release arch=darwin-b
- xdbaqeo ^ncurses@6.2%apple-clang@13.0.0-symlinks+termlib abi=none arch=darwin-bigsur-skylake
- kfureok ^pkconf@1.8.0%apple-clang@13.0.0 arch=darwin-bigsur-skylake
- 5ekd4ap ^openssl@1.1.1%apple-clang@13.0.0-docs certs+system arch=darwin-bigsur-skylake
- xx2625s ^perl@5.34.0%apple-clang@13.0.0+cpam+shared+threads arch=darwin-bigsur-skylake
- xgt3tls ^berkeley-db@18.1.40%apple-clang@13.0.0+cxx+dos+stl patches=b231fcc4d5cff05e5c3a4814f
- 65edjf6 ^bztp2@1.0.8%apple-clang@13.0.0-debug+pic+shared arch=darwin-bigsur-skylake
- 662ad0o ^diffutils@3.8%apple-clang@13.0.0 arch=darwin-bigsur-skylake
- fu7tfsr ^libiconv@1.16%apple-clang@13.0.0 libs=shared,static arch=darwin-bigsur-skylake
- vjg67nd ^gdbm@1.19%apple-clang@13.0.0 arch=darwin-bigsur-skylake
- tjceldr ^readline@8.1%apple-clang@13.0.0 arch=darwin-bigsur-skylake
- xevlijj ^zlib@1.2.11%apple-clang@13.0.0+optimize+pic+shared arch=darwin-bigsur-skylake
- xelfovbb ^openmp@4.1.1%apple-clang@13.0.0-atomics+cuda-cxx-exceptions+gfps+internal-hwloc-java+legacy
- zrunrs75 ^hwloc@2.6.0%apple-clang@13.0.0-cairo-cuda-gl-libudev+libxml2-netloc-nvml+opencl-pci+rocm+sh
- iib4fnkf ^libxml2@2.9.12%apple-clang@13.0.0-python arch=darwin-bigsur-skylake
- dwiv2ys ^xz@5.2.5%apple-clang@13.0.0-pic libs=shared,static arch=darwin-bigsur-skylake
- blitnbl ^libevent@2.1.12%apple-clang@13.0.0+openssl arch=darwin-bigsur-skylake
- h7jalyu ^openssl@1.1.1-20210216%apple-clang@13.0.0 arch=darwin-bigsur-skylake
- 7v7bxq2 ^libedit@3.1-20210216%apple-clang@13.0.0 arch=darwin-bigsur-skylake
```

Pure hash-based reuse: all misses

```
(spackle):spack> spack solve --reuse -Il hdf5
=> Best of 10 considered solutions.
=> Optimization Criteria:
  Priority Criterion           Installed  ToBuild
  1   number of packages to build (vs. reuse)      -    4
  2   deprecated versions used                   0    0
  3   version weight                           0    0
  4   number of non-default variants (roots)     0    0
  5   preferred providers for roots            0    0
  6   default values of variants not being used (roots) 0    0
  7   number of non-default variants (non-roots) 2    0
  8   preferred providers (non-roots)          0    0
  9   compiler mismatches                     0    0
 10  OS mismatches                          0    0
 11  non-preferred OS's                      0    0
 12  version badness                        6    0
 13  default values of variants not being used (non-roots) 1    0
 14  non-preferred compilers                 15   4
 15  target mismatches                     0    0
 16  non-preferred targets                  0    0

- yfkfnsp hdf5@1.10.7%apple-clang@12.0.5-cxx-fortran-hl-ip0-java+mpi+shared-szip-threadsafe+tools api=default b
- zd4m26e ^cmake@3.21.1%apple-clang@12.0.5-doc+ncurses+openssl+owlbins+qt build_type=Release arch=darwin-b
- s3152xr ^ncurses@6.2%apple-clang@12.0.5-symlinks+termlib abi=none arch=darwin-bigsur-skylake
- us36bwr ^openssl@1.1.1%apple-clang@12.0.5-docs+systemcerts arch=darwin-bigsur-skylake
- 74mmwxg ^openmp@4.1.1%apple-clang@12.0.5+atomic+cuda-cxx-exceptions+gfps+internal-hwloc-java+leg
- Bijsfnel ^hwloc@2.6.0%apple-clang@12.0.5-cairo-cuda-gl-libudev+libxml2-netloc-nvml+opencl-pci+rocm+
- jxexybz7 ^libiconv@1.16%apple-clang@12.0.5 libs=shared,static arch=darwin-bigsur-skylake
- ckdhnzf ^libxml2@2.9.12%apple-clang@12.0.5-python arch=darwin-bigsur-skylake
- k7xuat3 ^xz@5.2.5%apple-clang@12.0.5-pic libs=shared,static arch=darwin-bigsur-skylake
- k2yungmx ^pkconf@1.8.0%apple-clang@12.0.5 arch=darwin-bigsur-skylake
- grgtlcd ^openssl@1.1.1%apple-clang@12.0.5+openssl arch=darwin-bigsur-skylake
- inc66ug ^libevent@2.1.12%apple-clang@12.0.5+openssl arch=darwin-bigsur-skylake
- 63kbksk ^openmp@4.1.1%apple-clang@12.0.5+atomic+cuda-cxx-exceptions+gfps+internal-hwloc-java+leg
- snhgl1dt ^libedit@3.1-20210216%apple-clang@12.0.5 arch=darwin-bigsur-skylake
- qbkmtdd ^perl@5.34.0%apple-clang@12.0.5+cpam+shared+threads arch=darwin-bigsur-skylake
- tnvknvls ^berkeley-db@18.1.40%apple-clang@12.0.5+cxx+dos+stl patches=b231fcc4d5cff05e5c3a4814f
- 7dswqot ^xz@5.2.5%apple-clang@12.0.5-debug+pic+shared arch=darwin-bigsur-skylake
- vhdfl3i ^gdbm@1.19%apple-clang@12.0.5 arch=darwin-bigsur-skylake
- gqy3v4l ^readline@8.1%apple-clang@12.0.5 arch=darwin-bigsur-skylake
```

With reuse: 16 packages were reusable



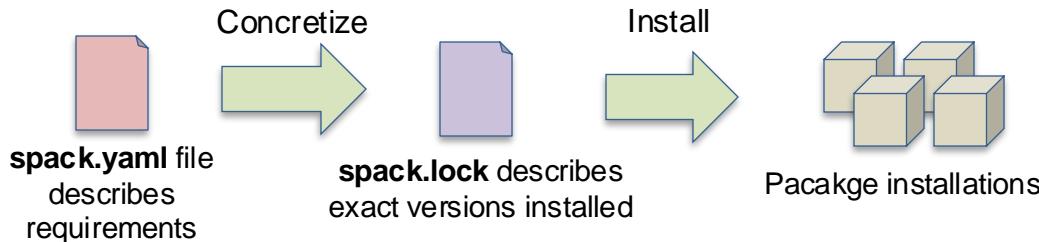
Use `spack spec` to see the results of concretization

```
$ spack spec mpileaks
Input spec
-----
mpileaks

Concretized
-----
mpileaks@1.0%gcc@5.3.0 arch=darwin-elcapitan-x86_64
^adept-utils@1.0.1%gcc@5.3.0 arch=darwin-elcapitan-x86_64
^boost@1.61.0%gcc@5.3.0+atomic+chrono+date_time~debug+filesystem~graph
~icu_support+iostreams+locale+log+math~mpi+multithreaded+program_options
~python+random+regex+serialization+shared+signals+singlethreaded+system
+test+thread+timer+wave arch=darwin-elcapitan-x86_64
^bzip2@1.0.6%gcc@5.3.0 arch=darwin-elcapitan-x86_64
^zlib@1.2.8%gcc@5.3.0 arch=darwin-elcapitan-x86_64
^openmpi@2.0.0%gcc@5.3.0~mxm~pmi~psm~psm2~slurm~sqlite3~thread_multiple~tm~verbs+vt arch=darwin-elcapitan-x86_64
^hwloc@1.11.3%gcc@5.3.0 arch=darwin-elcapitan-x86_64
^libpciaccess@0.13.4%gcc@5.3.0 arch=darwin-elcapitan-x86_64
^libtool@2.4.6%gcc@5.3.0 arch=darwin-elcapitan-x86_64
^m4@1.4.17%gcc@5.3.0+sigsegv arch=darwin-elcapitan-x86_64
^libsigsegv@2.10%gcc@5.3.0 arch=darwin-elcapitan-x86_64
^callpath@1.0.2%gcc@5.3.0 arch=darwin-elcapitan-x86_64
^dyninst@9.2.0%gcc@5.3.0~stat_dysect arch=darwin-elcapitan-x86_64
^libdwarf@20160507%gcc@5.3.0 arch=darwin-elcapitan-x86_64
^libelf@0.8.13%gcc@5.3.0 arch=darwin-elcapitan-x86_64
```



Spack environments enable users to build customized stacks from an abstract description



- spack.yaml describes project requirements
- spack.lock describes exactly what versions/configurations were installed, allows them to be reproduced.
- Can be used to maintain configuration of a software stack.
 - Can easily version an environment in a repository

Simple spack.yaml file

```
spack:  
  # include external configuration  
  include:  
    - ./special-config-directory/  
    - ./config-file.yaml  
  
  # add package specs to the `specs` list  
  specs:  
    - hdf5  
    - libelf  
    - openmpi
```

Concrete spack.lock file (generated)

```
{  
  "concrete_specs": {  
    "6s63so2kstp3zyvjezglndmavy6l3nul": {  
      "hdf5": {  
        "version": "1.10.5",  
        "arch": {  
          "platform": "darwin",  
          "platform_os": "mojave",  
          "target": "x86_64"  
        },  
        "compiler": {  
          "name": "clang",  
          "version": "10.0.0-apple"  
        },  
        "namespace": "builtin",  
        "parameters": {}  
      }  
    }  
  }  
}
```

We'll resume at: 11:00 am

Get a VM here →



Find the slides and associated scripts here:

spack-tutorial.rtfd.io

Remember to join Spack slack so you can get help later!

slack.spack.io

Join the #tutorial channel!

The screenshot shows the "Spack" documentation homepage. The header features the Spack logo (a blue diamond with orange nodes) and the word "Spack". Below the header, there's a search bar labeled "Search docs". A sidebar on the left contains links for "LINKS" (Main Spack Documentation, TUTORIAL, Basic Installation Tutorial, Configuration Tutorial, Package Creation Tutorial, Developer Workflows Tutorial), "Read the Docs" (with a dropdown menu showing "v: latest"), "Versions" (latest, sc18, sc17, sc16, riken19, pearc19, nsf19, lanl19, isc19, ecp19), "Downloads" (HTML, On Read the Docs, Project Home, Builds, Downloads), "On GitHub" (View, Edit), "Search" (Search docs), and footer links for "Hosted by Read the Docs · Privacy Policy".

Docs » Tutorial: Spack 101

Tutorial: Spack 101

This is a full-day introduction to Practice and Experience in Ad 2019.

You can use these materials to and read the live demo scripts

Slides



Full citat

Managin

Practice and Experience in Ad Chicago, IL, USA.

Live Demos

We provide scripts that take you sections in the slides above. Yo

1. We provide the [spack/tut](#) tutorial on your local machine in the container.
2. When we host the tutorial unfamiliar with Docker

You should now be ready to ru

Environments, spack.yaml and spack.lock

Follow script at spack-tutorial.readthedocs.io



Hands-on Time: Configuration

Follow script at spack-tutorial.readthedocs.io



We'll resume at: 1:30 pm

Get a VM here →



Find the slides and associated scripts here:

spack-tutorial.rtfd.io

Remember to join Spack slack so you can get help later!

slack.spack.io

Join the #tutorial channel!

The screenshot shows the Spack documentation homepage. The header features the Spack logo (a blue diamond with orange nodes) and the word "Spack". Below the header, there's a search bar labeled "Search docs". A sidebar on the left contains links for "LINKS" (Main Spack Documentation, TUTORIAL, Basic Installation Tutorial, Configuration Tutorial, Package Creation Tutorial, Developer Workflows Tutorial), "Read the Docs" (with a dropdown menu showing "v: latest"), "Versions" (latest, sc18, sc17, sc16, riken19, pearc19, nsf19, lanl19, isc19, ecp19), "Downloads" (HTML, On Read the Docs, Project Home, Builds, Downloads), "On GitHub" (View, Edit), "Search" (Search docs), and footer links for "Hosted by Read the Docs · Privacy Policy".

Docs » Tutorial: Spack 101

Tutorial: Spack 101

This is a full-day introduction to Practice and Experience in Ad 2019.

You can use these materials to and read the live demo scripts

Slides



Practice and Experience in Ad Chicago, IL, USA.

Live Demos

We provide scripts that take you sections in the slides above. Yo

1. We provide the [spack/tut](#) tutorial on your local machine in the container.
2. When we host the tutorial unfamiliar with Docker

You should now be ready to ru

Hands-on Time: Stacks

Follow script at spack-tutorial.readthedocs.io



Spack packages are *parameterized* using the spec syntax

Python DSL defines many ways to build

```
from spack import *

class Kripke(CMakePackage):
    """Kripke is a simple, scalable, 3DSn deterministic particle transport mini-app."""

    homepage = "https://computation.llnl.gov/projects/co-design/kripke"
    url    = "https://computation.llnl.gov/projects/co-design/download/kripke-openmp-1.1.tar.gz"

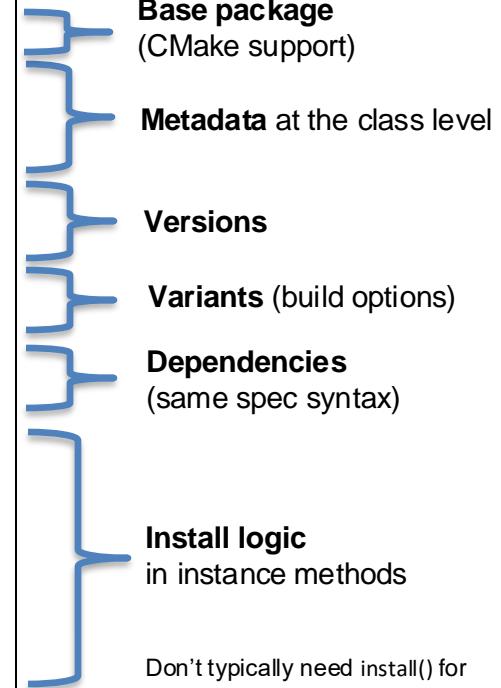
    version('1.2.3', sha256='3f7f2eef0d1ba5825780d626741eb0b3f026a096048d7ec4794d2a7dfbe2b8a6')
    version('1.2.2', sha256='eaf9ddf562416974157b34d00c3a1c880fc5296fce2aa2efa039a86e0976f3a3')
    version('1.1', sha256='232d74072fc7b848fa2adc8a1bc839ae8fb5f96d50224186601f55554a25f64a')

    variant('mpi', default=True, description='Build with MPI.')
    variant('openmp', default=True, description='Build with OpenMP enabled.')

    depends_on('mpi', when='+mpi')
    depends_on('cmake@3.0:', type='build')

    def cmake_args(self):
        return [
            '-DENABLE_OPENMP=%s' % ('+openmp' in self.spec),
            '-DENABLE_MPI=%s' % ('+mpi' in self.spec),
        ]

    def install(self, spec, prefix):
        mkdirp(prefix.bin)
        install('../spack-build/kripke', prefix.bin)
```



One package.py file per software project!



Hands-on Time: Creating Packages

Follow script at spack-tutorial.readthedocs.io



Hands-on Time: Scripting

Follow script at spack-tutorial.readthedocs.io



We'll resume at: 3:00 pm

Get a VM here →



Find the slides and associated scripts here:

spack-tutorial.rtfd.io

Remember to join Spack slack so you can get help later!

slack.spack.io

Join the #tutorial channel!

The screenshot shows the Spack documentation homepage. The header features the Spack logo (a blue diamond with orange nodes) and the word "Spack". Below the header, there's a search bar labeled "Search docs". A sidebar on the left contains links for "LINKS" (Main Spack Documentation, TUTORIAL, Basic Installation Tutorial, Configuration Tutorial, Package Creation Tutorial, Developer Workflows Tutorial), "Read the Docs" (with a dropdown menu showing "v: latest"), "Versions" (latest, sc18, sc17, sc16, riken19, pearc19, nsf19, lanl19, isc19, ecp19), "Downloads" (HTML, On Read the Docs, Project Home, Builds, Downloads), "On GitHub" (View, Edit), "Search" (Search docs), and footer links for "Hosted by Read the Docs · Privacy Policy".

Docs » Tutorial: Spack 101

Tutorial: Spack 101

This is a full-day introduction to Practice and Experience in Ad 2019.

You can use these materials to and read the live demo scripts

Slides



Full citat

Managin

Practice and Experience in Ad Chicago, IL, USA.

Live Demos

We provide scripts that take you sections in the slides above. Yo

1. We provide the [spack/tut](#) tutorial on your local machine in the container.
2. When we host the tutorial unfamiliar with Docker

You should now be ready to ru

Hands-on Time: Developer Workflows

Follow script at spack-tutorial.readthedocs.io



Hands-on Time: Binary Caches and Mirrors

Follow script at spack-tutorial.readthedocs.io



Roadmap for this year

- Finish compiler dependencies
 - Compilers will appear as build dependencies
 - All special compiler logic generified for any build dependency
 - Enable bootstrapping compilers from buildcache
 - Continue to flesh out low-level compiler runtime libraries
- Harden public buildcaches and make them more broadly compatible
- Enable bare-metal MPI/CUDA/ROCm installations
- Make the buildcache on by default; build only what we need from source
- Continue to improve Windows support
- Speed improvements for concretization and metadata management

Lots of exciting work ahead!

When would we go to “Version 1.0”?

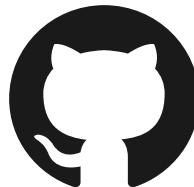
Big things we've wanted for 1.0 are:

- New concretizer
- production CI **Done!**
- production public build cache
- Compilers as dependencies
- Buildcache hardening **Aiming for November**
- Stable package API **Aiming for June**
 - Enables separate package repository

We are getting very close!

Join the Spack community!

- Join us and 3,000 others on Spack slack
- Contribute packages, documentation, and features on GitHub
- Continue the tutorial at spack-tutorial.rtfd.io



★ Star us on GitHub!
github.com/spack/spack



We hope to make distributing & using HPC software easy!



Disclaimer

This document was prepared as an account of work sponsored by an agency of the United States government. Neither the United States government nor Lawrence Livermore National Security, LLC, nor any of their employees makes any warranty, expressed or implied, or assumes any legal liability or responsibility for the accuracy, completeness, or usefulness of any information, apparatus, product, or process disclosed, or represents that its use would not infringe privately owned rights. Reference herein to any specific commercial product, process, or service by trade name, trademark, manufacturer, or otherwise does not necessarily constitute or imply its endorsement, recommendation, or favoring by the United States government or Lawrence Livermore National Security, LLC. The views and opinions of authors expressed herein do not necessarily state or reflect those of the United States government or Lawrence Livermore National Security, LLC, and shall not be used for advertising or product endorsement purposes.