



Managing HPC Software Complexity with Spack

ACM-REP24 Tutorial
June 18, 2024

Massimiliano Culpo, Adam Stewart, Todd Gamblin,
Gregory Becker, and Harmen Stoppels

THE **LINUX** FOUNDATION



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



Adam Stewart
TU Munich



Massimiliano Culpo
np-complete S.r.l.

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!

Join #tutorial on Slack: slack.spack.io

Materials: spack-tutorial.readthedocs.io

The screenshot shows the Spack documentation page on Read the Docs. The top navigation bar includes a search bar labeled "Search docs" and a dropdown menu showing "latest". Below the header, there are sections for "LINKS" (Main Spack Documentation), "TUTORIAL" (Basic Installation Tutorial, Configuration Tutorial, Package Creation Tutorial, Developer Workflows Tutorial), and "Read the Docs" (with a "v: latest" dropdown). A sidebar on the left lists "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", and "Search". At the bottom, it says "Hosted by Read the Docs · Privacy Policy".

Docs » Tutorial: Sp

Tutorial: S

This is a full-day int
Practice and Experi
2019.

You can use these n
and read the live de

Slides



Practice and Experi
Chicago, IL, USA.

Live Demos

We provide scripts
sections in the slide

1. We provide a
tutorial on yo
the container
2. When we ha
unfamiliar wi

You should now be



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	35.20.150.155				

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)

Intro	08:30
Environments and Reproducibility	08:50
Break	10:00
Core Concepts	10:30
Buildcaches and Containers	11:00
Packaging and Configuration	11:30
End	12:00

You can explore more sections on the tutorial site:

- Modules
- Developer Workflows
- Scripting
- Stack

We would cover these in a full day version of the tutorial.



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**

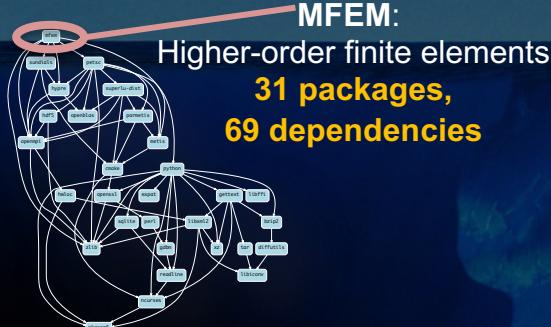


Build-time incompatibility; fail fast



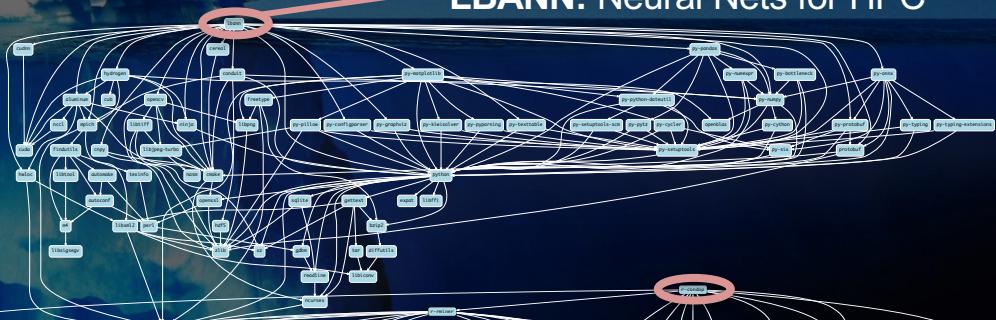
Appears to work; subtle errors later

Modern scientific codes rely on icebergs of dependency libraries

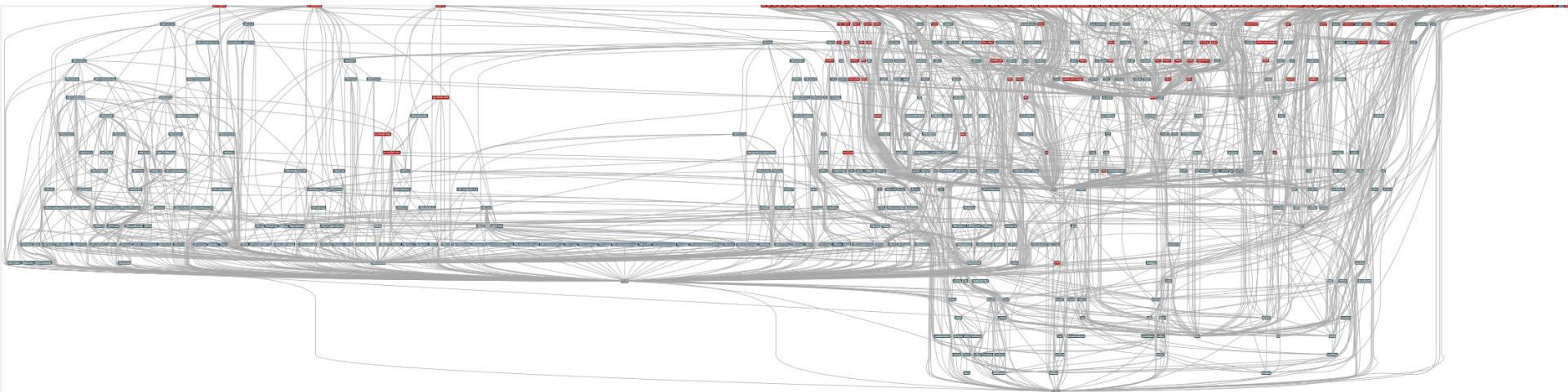


71 packages
188 dependencies

LBANN: Neural Nets for HPC



ECP's E4S stack is even larger than these codes

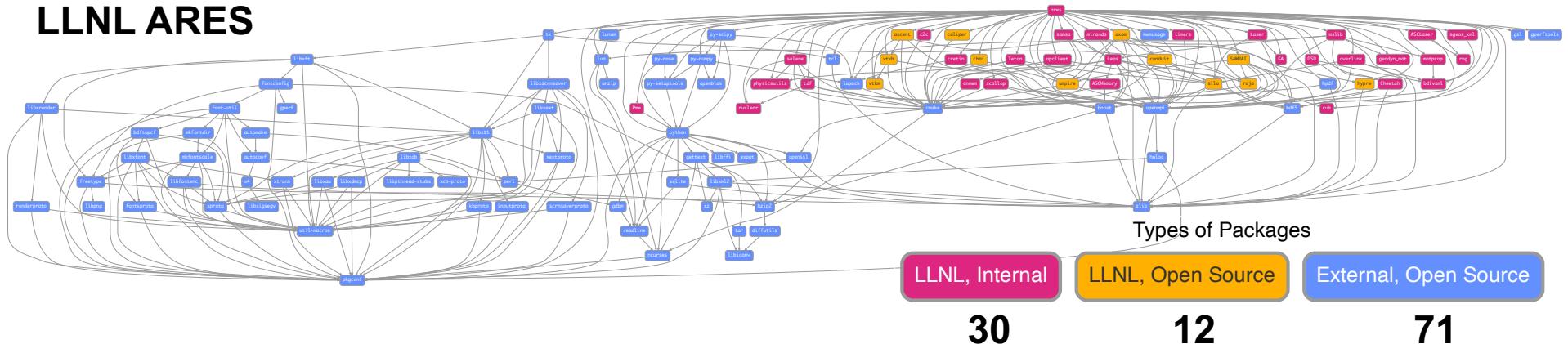


- Red boxes are the packages in it (about 100)
- Blue boxes are what *else* you need to build it (about 600)
- It's infeasible to build and integrate all of this manually



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

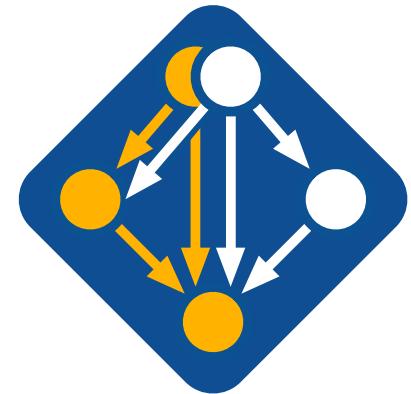
Spack enables Software distribution for HPC

- 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="--03 -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?

- Spack is a ***package manager***
 - Does not replace Cmake/Autotools
 - Packages built by Spack can have any build system they want
- Spack manages ***dependencies***
 - Drives package-level build systems
 - Ensures consistent builds
- Determining magic configure lines takes time
 - Spack is a cache of recipes

Package Manager

- Manages package installation
- Manages dependency relationships
- May drive package-level build systems

High Level Build System

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

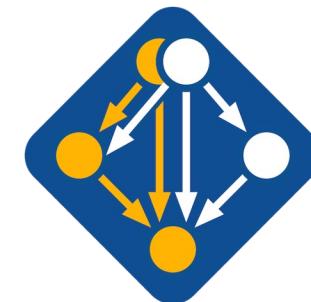
Low Level Build System

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



Spack is officially a Linux Foundation Project now!

- What does that mean?
 - Project has a legal 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!
 - Project will have a Technical Steering Committee (TSC)
 - Plan is to make the main developer meetings more public
 - Also have official steering committee meetings
 - Main charter is written (mostly boilerplate)
 - Working on initial GOVERNANCE.md, initial TSC members
 - Trademark (Spack name, logo) assigned to Linux Foundation
 - Project resources owned by Linux Foundation
 - spack.io website
 - GitHub Organization



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

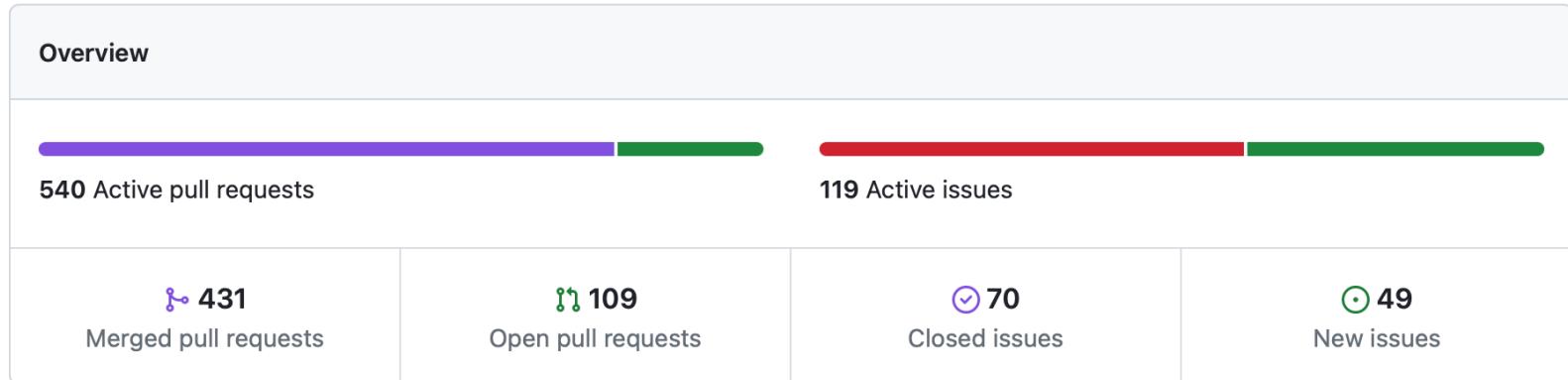
- **AWS** is investing significantly in cloud credits for Spack
 - Supporting highly scalable cloud CI system with ~250k+/year in credits
 - Integrating Spack with ParallelCluster product
 - Joint Spack tutorial with AWS drew 125+ participants
- **Google** is using Spack in their HPC Toolkit cloud cluster 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!

October 11, 2023 – November 11, 2023

Period: 1 month ▾



Excluding merges, **143 authors** have pushed **427 commits** to develop and **544 commits** to all branches. On develop, **695 files** have changed and there have been **18,789 additions** and **5,775 deletions**.



Join #tutorial on Slack: slack.spack.io

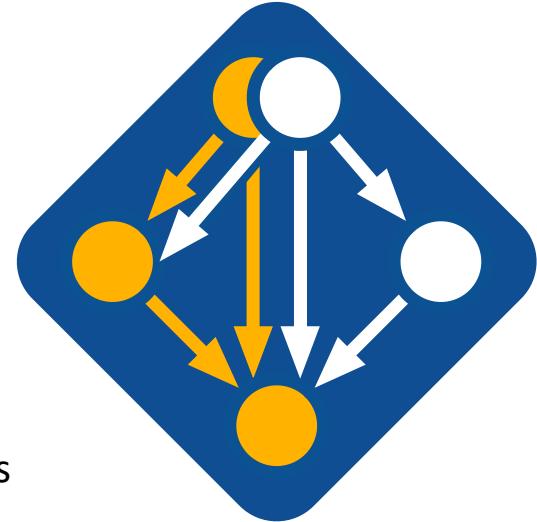
Materials: spack-tutorial.readthedocs.io



Spack v0.22.0 is 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>

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>

Hands-on Time: Environment and Reproducibility

Follow script at spack-tutorial.readthedocs.io



We'll resume at: 10:30 PT

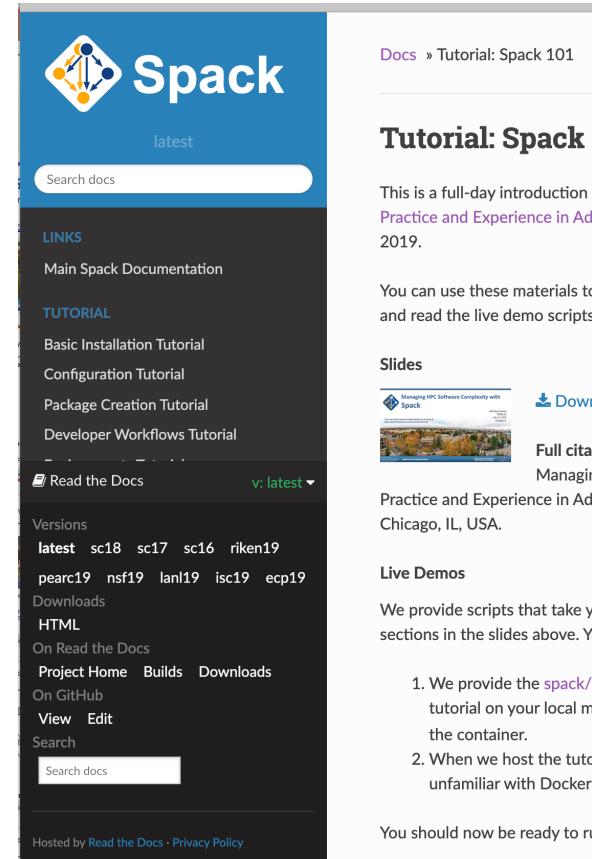
Find the slides and associated scripts here:

spack-tutorial.rtfd.io

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

slack.spack.io
Join the #tutorial channel!

Get a VM here →



The screenshot shows a documentation page for Spack. At the top right, there's a navigation bar with "Docs" and "Tutorial: Spack 101". Below it is a large header with the Spack logo and the word "Spack". A search bar says "Search docs". On the left, there's a sidebar with "LINKS" (Main Spack Documentation) and "TUTORIAL" (Basic Installation Tutorial, Configuration Tutorial, Package Creation Tutorial, Developer Workflows Tutorial). A dropdown menu "Read the Docs" shows "v: latest". The main content area has sections for "Versions" (latest, sc18, sc17, sc16, riken19, pearc19, nsf19, lan19, isc19, ecp19), "Downloads", "HTML", "On Read the Docs", "Project Home", "Builds", "Downloads", "On GitHub", "View", "Edit", and "Search". A footer at the bottom right says "Hosted by Read the Docs · Privacy Policy". To the right of the main content, there's a sidebar with "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 yo sections in the slides above. Yo", "1. We provide the spack/t tutorial on your local ma the container.", "2. When we host the tutu unfamiliar with Docker", and "You should now be ready to ru".

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, 3D Sn 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"

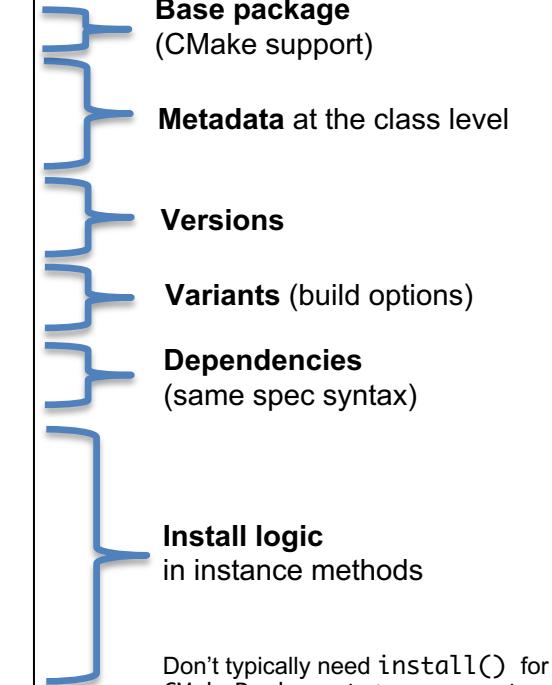
    version('1.2.3', sha256='3f7f2eef0d1ba5825780d626741eb0b3f026a096048d7ec4794d2a7dfbe2b8a6')
    version('1.2.2', sha256='eaf9ddf562416974157b34d00c3a1c880fc5296fce2aa2efa039a86e0976f3a3')
    version('1.1', sha256='232d74072fc7b848fa2adc8a1bc839ae8fb5f96d50224186601f55554a25f64a')

    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 [
            '--ENABLE_OPENMP=%s' % ('+openmp' in self.spec),
            '--ENABLE_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

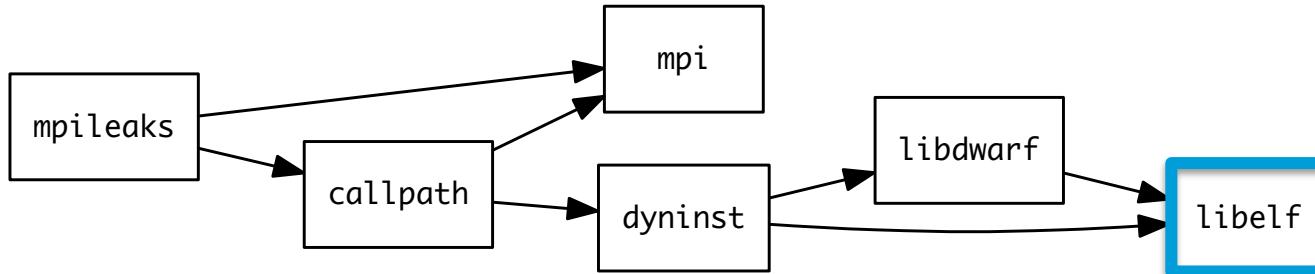
constraints on cuda version

compiler support for x86_64 and ppc64le

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



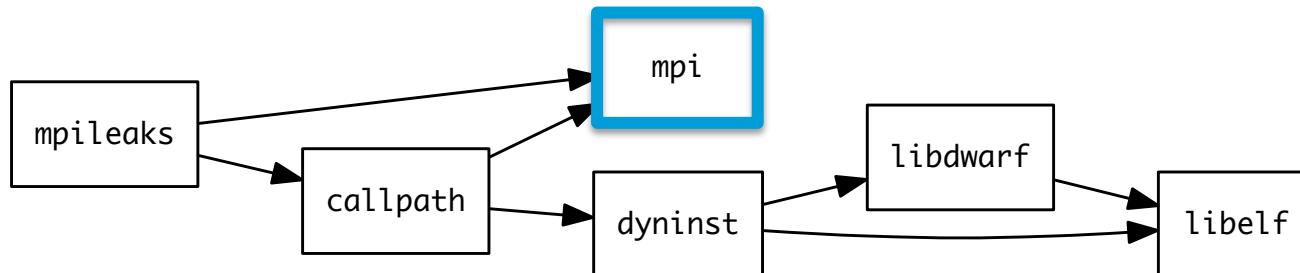
Spack Specs can constrain versions of dependencies



```
$ 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:

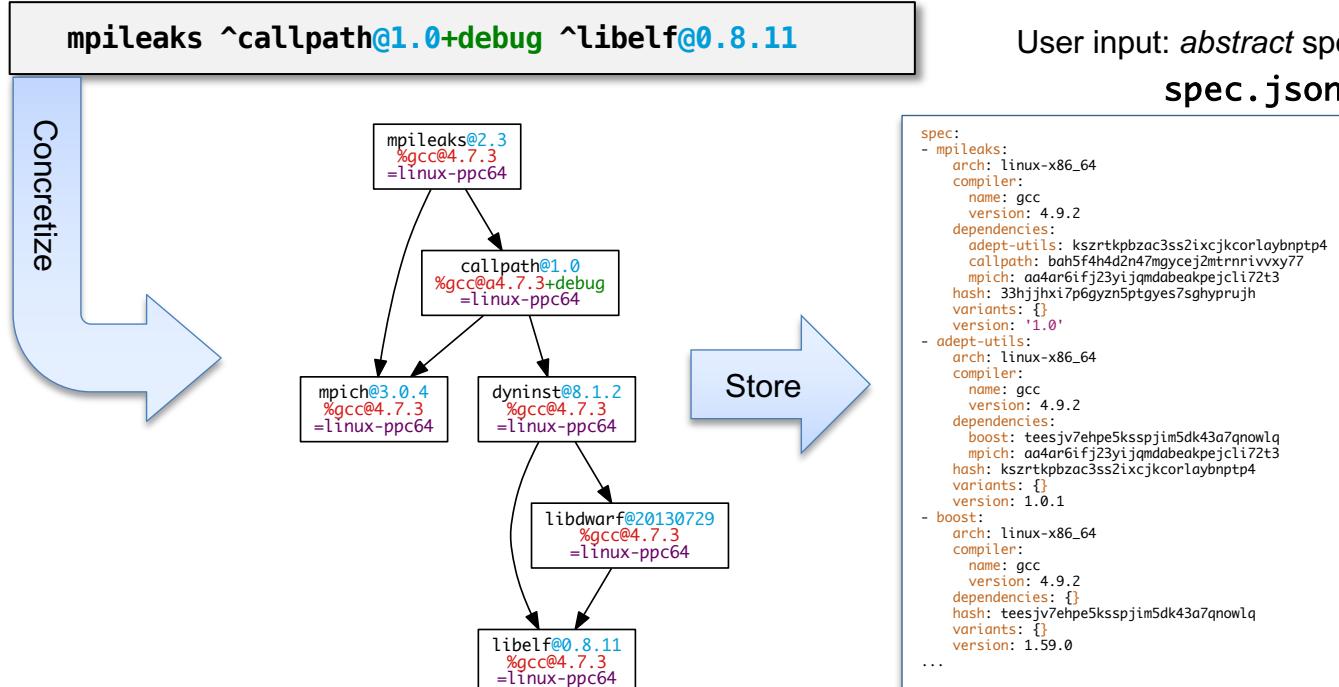
```
$ spack install mpileaks ^mvapich@1.9
```

```
$ 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.



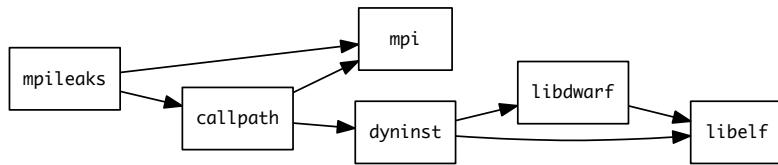
Concrete spec is fully constrained and can be passed to install.

Detailed provenance is stored with the installed package



Hashing allows us to handle combinatorial complexity

Dependency DAG



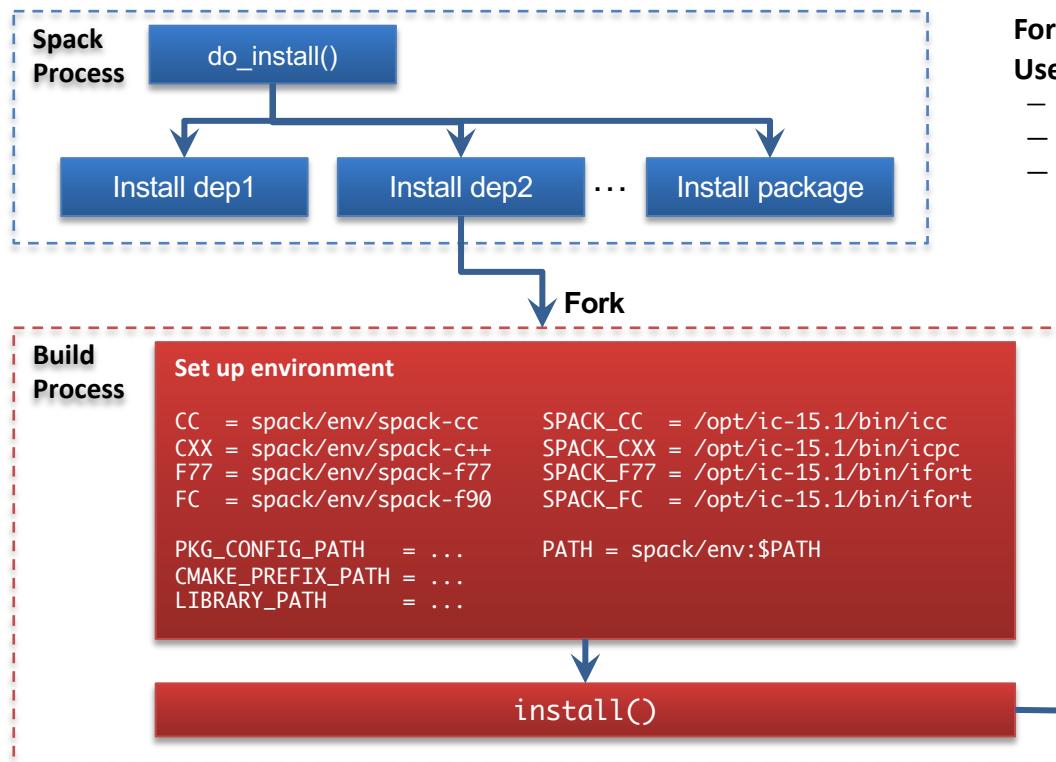
Installation Layout

```
opt
└── spack
    ├── darwin-mojave-skylake
    │   └── clang-10.0.0-apple
    │       ├── bzip2-1.0.8-hc4sm4vuzpm4znmvrfzri4ow2mkphe2e
    │       ├── python-3.7.6-daqqpsssxb6qbfrztsezkmhus3xoflbsy
    │       ├── sqlite-3.30.1-u64v26igxvbyn23hysmklfums6tgjv5r
    │       ├── xz-5.2.4-u5eawkvaoc7vonabe6nnndkcfwuv233cj
    │       └── zlib-1.2.11-x46q4wm46ay4pltrijbgizxjrhbaka6
    └── darwin-mojave-x86_64
        └── clang-10.0.0-apple
            └── coreutils-8.29
p12kcytejqcys5dzecfrtjqxfdssvnb
```

HASH

- 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*

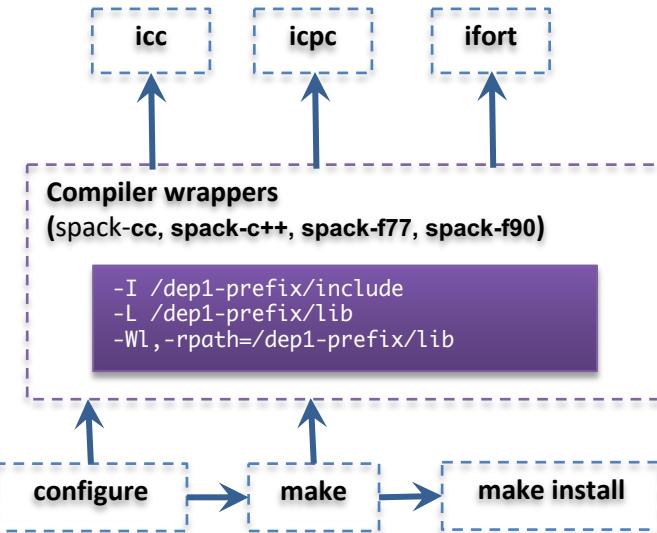
An isolated compilation environment allows Spack to easily swap compilers



Forked build process isolates environment for each build.

Uses compiler wrappers to:

- Add include, lib, and RPATH flags
- Ensure that dependencies are found automatically
- Load Cray modules (use right compiler/system deps)

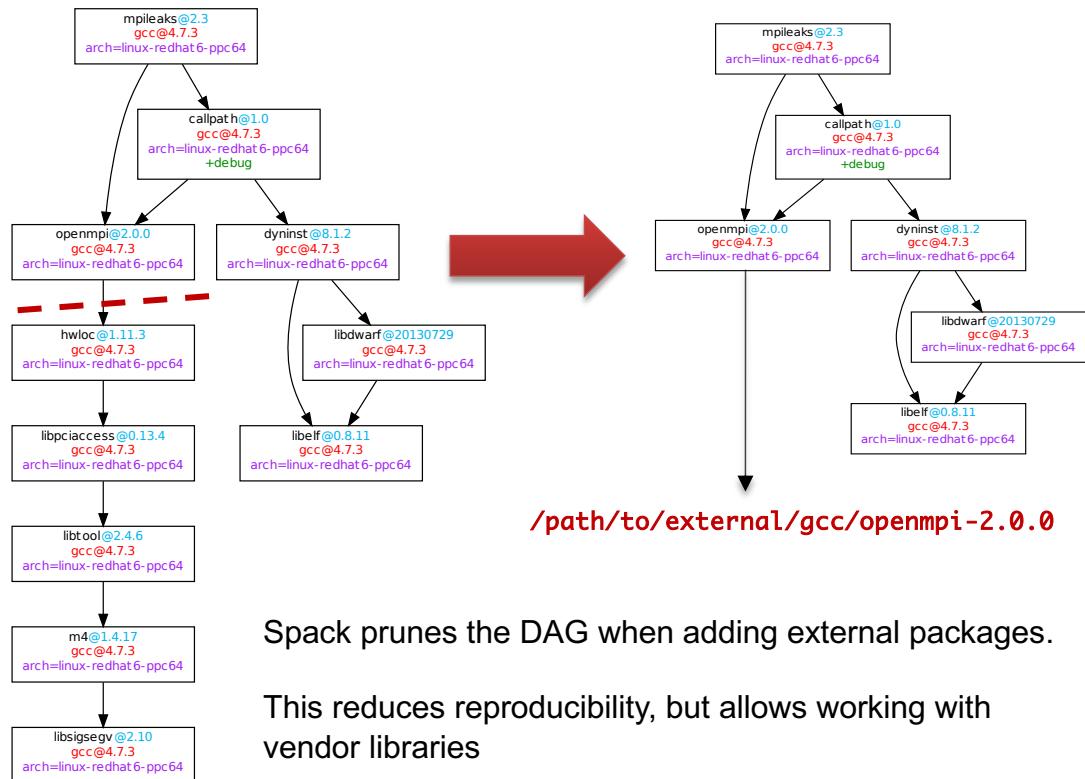


We can configure Spack to build with external software

```
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  
      ...
```



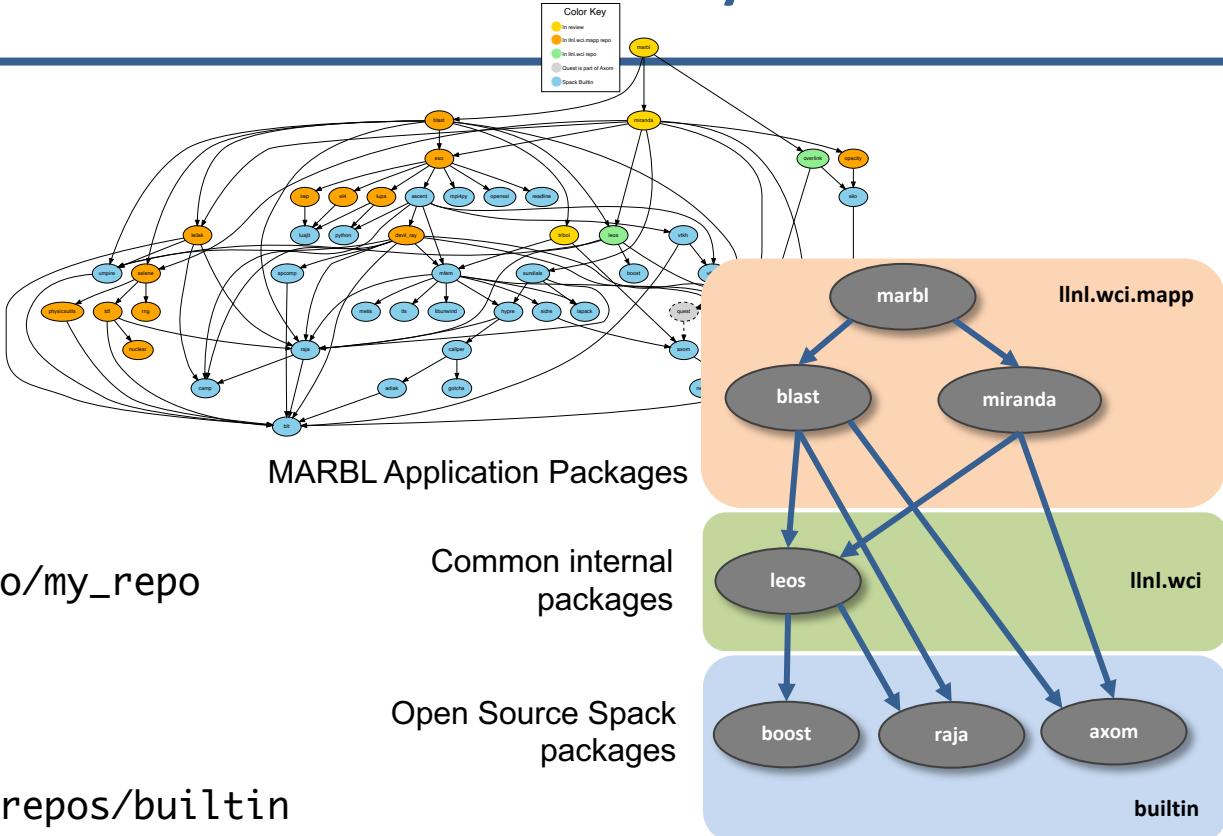
Users register external packages in a configuration file (more on these later).

Spack prunes the DAG when adding external packages.

This reduces reproducibility, but allows working with vendor libraries

Spack package repositories allow stacks to be layered

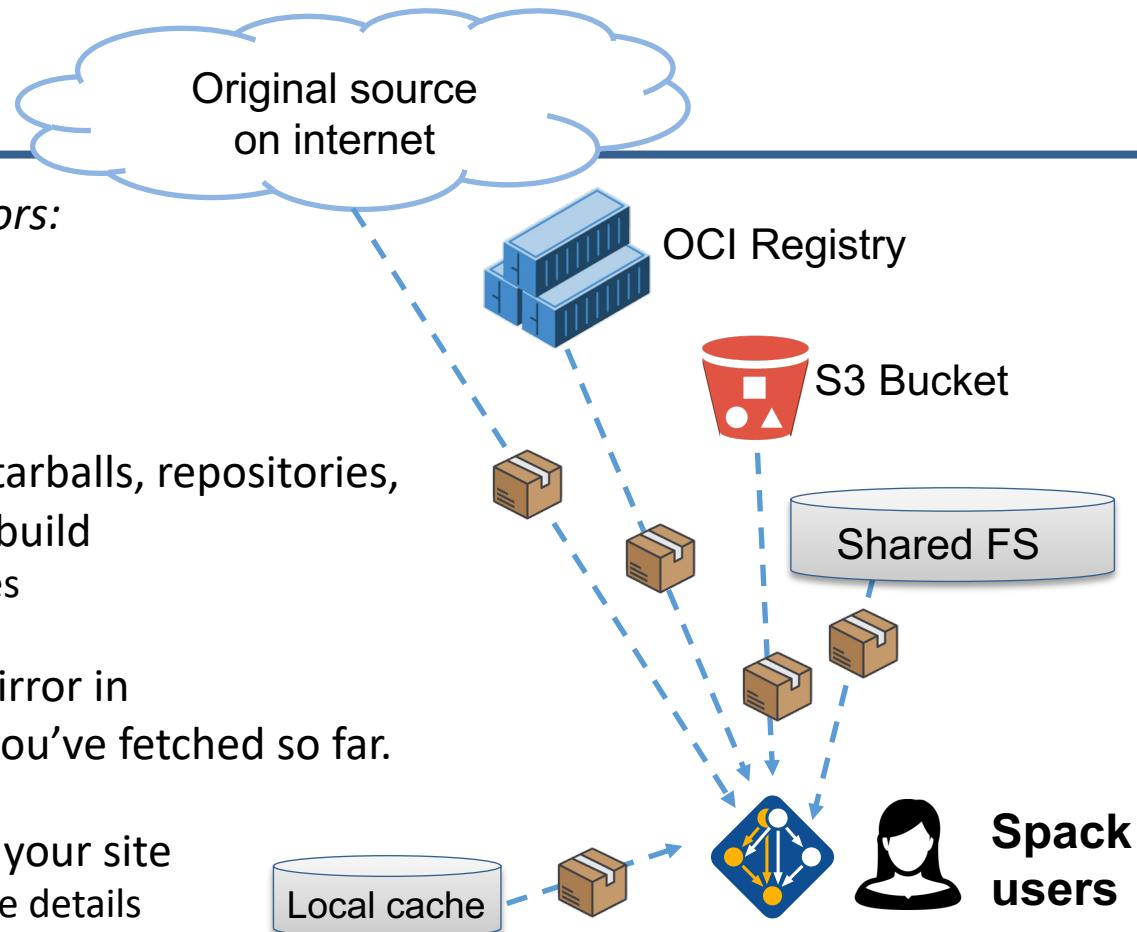
LLNL MARBL multi-physics application



```
$ 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

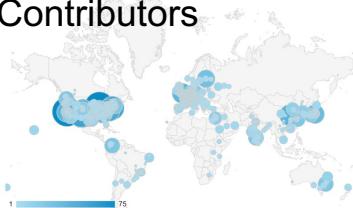
- Spack allows you to define *mirrors*:
 - Directories in the filesystem
 - On a web server
 - In an S3 bucket
- Mirrors are archives of fetched tarballs, repositories, and other resources needed to build
 - Can also contain binary packages
- By default, Spack maintains a mirror in `var/spack/cache` 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

Contributors



- new versions
- new dependencies
- new constraints

package.py repository

spack
developers



default config
packages.yaml

admins,
users



local preferences config
packages.yaml

users

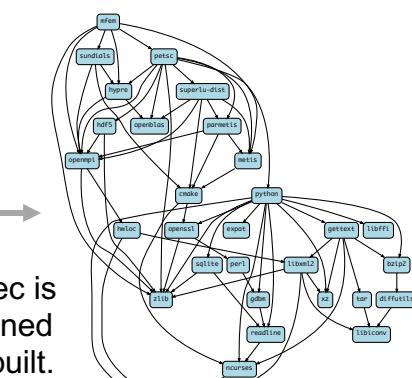
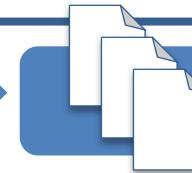


local environment config
spack.yaml

users

Command line constraints

```
spack install hdf5@1.12.0 +debug
```



Concrete spec
is
fully constrained
and can be built.

We use logic programming to simplify package solving

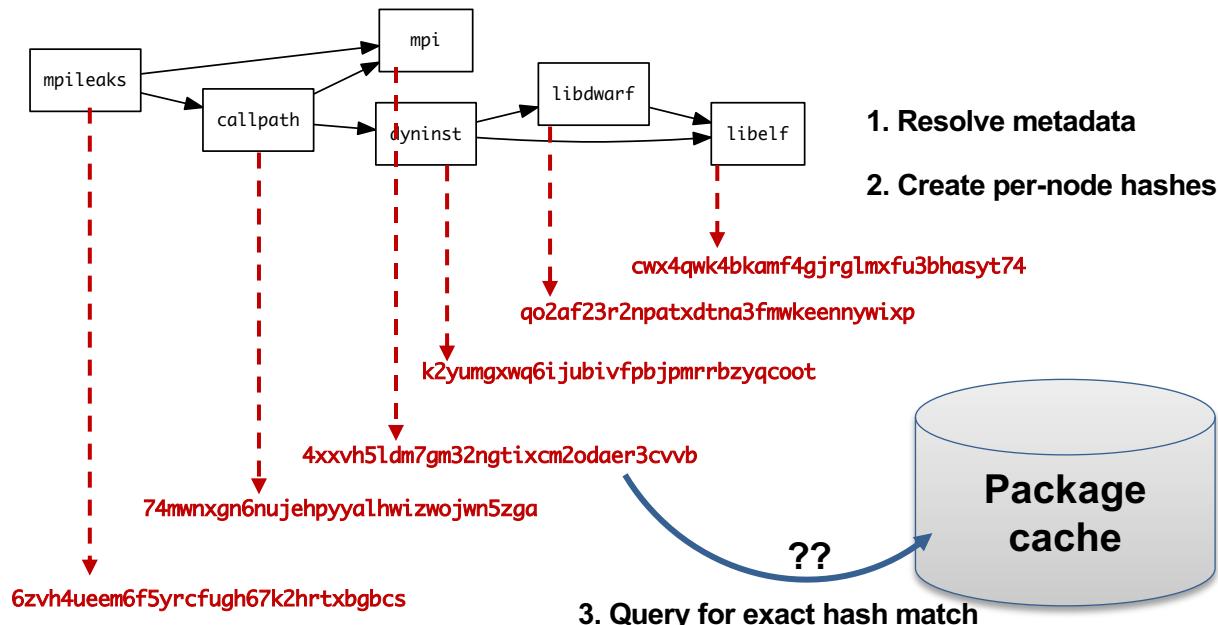
- New concretizer leverages Clingo (see potassco.org)
- Clingo is an Answer Set Programming (ASP) solver
 - ASP looks like Prolog; leverages SAT solvers for speed/correctness
 - ASP program has 2 parts:
 1. Large list of facts generated from our package repositories and config
 2. Small logic program (~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("uctx", "1.6.1", 0).  
version_declared("uctx", "1.6.0", 1).  
version_declared("uctx", "1.5.2", 2).  
version_declared("uctx", "1.5.1", 3).  
version_declared("uctx", "1.5.0", 4).  
version_declared("uctx", "1.4.0", 5).  
version_declared("uctx", "1.3.1", 6).  
version_declared("uctx", "1.3.0", 7).  
version_declared("uctx", "1.2.2", 8).  
version_declared("uctx", "1.2.1", 9).  
version_declared("uctx", "1.2.0", 10).  
  
variant("uctx", "thread_multiple").  
variant_single_value("uctx", "thread_multiple").  
variant_default_value("uctx", "thread_multiple", "False").  
variant_possible_value("uctx", "thread_multiple", "False").  
variant_possible_value("uctx", "thread_multiple", "True").  
  
declared_dependency("uctx", "numactl", "build").  
declared_dependency("uctx", "numactl", "link").  
node("numactl") :- depends_on("uctx", "numactl"), node("uctx").  
  
declared_dependency("uctx", "rdma-core", "build").  
declared_dependency("uctx", "rdma-core", "link").  
node("rdma-core") :- depends_on("uctx", "rdma-core"), node("uctx").  
  
%-----  
% Package: util-linux  
%-----  
version_declared("util-linux", "7.29.2", 0).  
version_declared("util-linux", "7.29.1", 1).  
version_declared("util-linux", "7.25", 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 (now 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","lwatuuysmwkhahrnrywvn77icdhs6mn").  
imposed_constraint("lwatuuysmwkhahrnrywvn77icdhs6mn","node","openssl").  
imposed_constraint("lwatuuysmwkhahrnrywvn77icdhs6mn","version","openssl","1.1.1g").  
imposed_constraint("lwatuuysmwkhahrnrywvn77icdhs6mn","node_platform_set","openssl","darwin").  
imposed_constraint("lwatuuysmwkhahrnrywvn77icdhs6mn","node_os_set","openssl","catalina").  
imposed_constraint("lwatuuysmwkhahrnrywvn77icdhs6mn","node_target_set","openssl","x86_64").  
imposed_constraint("lwatuuysmwkhahrnrywvn77icdhs6mn","variant_set","openssl","systemcerts","True").  
imposed_constraint("lwatuuysmwkhahrnrywvn77icdhs6mn","node_compiler_set","openssl","apple-clang").  
imposed_constraint("lwatuuysmwkhahrnrywvn77icdhs6mn","node_compiler_version_set","openssl","apple-clang","12.0.0").  
imposed_constraint("lwatuuysmwkhahrnrywvn77icdhs6mn","concrete","openssl").  
imposed_constraint("lwatuuysmwkhahrnrywvn77icdhs6mn","depends_on","openssl","zlib","build").  
imposed_constraint("lwatuuysmwkhahrnrywvn77icdhs6mn","depends_on","openssl","zlib","link").  
imposed_constraint("lwatuuysmwkhahrnrywvn77icdhs6mn","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
  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

- zznqf3  hdf5@1.10.7%apple-clang@13.0.0-cxx-fortran-hl-ipa-java+mpi+shared-szip+threadsafe+tools api=default b
  ^cmake@3.21.4%apple-clang@13.0.0-doc+ncurses+openssl+owlibs+qt build_type=Release arch=darwin-bigsur-skylake
  ^ncurses@6.2%apple-clang@13.0.0-symlinks+termlib abi=None arch=darwin-bigsur-skylake
  ^krfureok  ^pkcconf@1.8.0%apple-clang@13.0.0 arch=darwin-bigsur-skylake
  ^openssl@1.1.1%apple-clang@13.0.0-docs certs+system arch=darwin-bigsur-skylake
  ^perl@5.34.0%apple-clang@13.0.0+cpnm+shared+threads arch=darwin-bigsur-skylake
  ^berkeley-db@18.1.40%apple-clang@13.0.0+cxx+docs+stl patches=b231fcc4d5cff05e5c3a4814f
  ^bzzip2@1.0.8%apple-clang@13.0.0-debug-pic+shared arch=darwin-bigsur-skylake
  ^diffutils@3.8%apple-clang@13.0.0 arch=darwin-bigsur-skylake
  ^libiconv@1.16%apple-clang@13.0.0 libs=shared,static arch=darwin-bigsur-skylake
  ^gdbm@1.19%apple-clang@13.0.0 arch=darwin-bigsur-skylake
  ^readline@8.1%apple-clang@13.0.0 arch=darwin-bigsur-skylake
  ^zlib@1.2.11%apple-clang@13.0.0+optimize+pic+shared arch=darwin-bigsur-skylake
  ^openmp@4.1.1%apple-clang@13.0.0+atomic+cxx-exceptions+gfps+internal-hwloc+java+legacy
  ^hwloc@2.6.0%apple-clang@13.0.0-cairo-cuda-gl-libudev+libxml2-netloc-nvml+opencl+pci+rocm+sh
  ^libxml2@9.12%apple-clang@13.0.0-python arch=darwin-bigsur-skylake
  ^xz@5.2.5%apple-clang@13.0.0-pic libs=shared,static arch=darwin-bigsur-skylake
  ^libevent@2.1.2%apple-clang@13.0.0+openssl arch=darwin-bigsur-skylake
  ^openssl@8.7%apple-clang@13.0.0 arch=darwin-bigsur-skylake
  ^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
  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-ipa-java+mpi+shared-szip+threadsafe+tools api=default b
  ^cmake@21.1%apple-clang@12.0.5-doc+ncurses+openssl+owlibs+qt build_type=Release arch=darwin-bigsur-skylake
  ^ncurses@6.2%apple-clang@12.0.5-symlinks+termlib abi=None arch=darwin-bigsur-skylake
  ^openssl@1.1.1%apple-clang@12.0.5-docs+systemcerts arch=darwin-bigsur-skylake
  ^perl@5.34.0%apple-clang@12.0.5+cpnm+shared+threads arch=darwin-bigsur-skylake
  ^berkeley-db@18.1.40%apple-clang@12.0.5+cxx+docs+stl patches=b231fcc4d5cff05e5c3a4814f
  ^bzzip2@1.0.8%apple-clang@12.0.5-debug-pic+shared arch=darwin-bigsur-skylake
  ^diffutils@3.8%apple-clang@12.0.5+atomic+cxx-exceptions+gfps+internal-hwloc+java+legacy
  ^hwloc@2.6.0%apple-clang@12.0.5-cairo-cuda-gl-libudev+libxml2-netloc-nvml+opencl+pci+rocm+sh
  ^libxml2@9.12%apple-clang@12.0.5-python arch=darwin-bigsur-skylake
  ^xz@5.2.5%apple-clang@12.0.5-pic libs=shared,static arch=darwin-bigsur-skylake
  ^libiconv@1.16%apple-clang@12.0.5 libs=shared,static arch=darwin-bigsur-skylake
  ^gdbm@1.19%apple-clang@12.0.5 arch=darwin-bigsur-skylake
  ^readline@8.1%apple-clang@12.0.5 arch=darwin-bigsur-skylake
  ^zlib@1.2.11%apple-clang@12.0.5+optimize+pic+shared arch=darwin-bigsur-skylake
  ^openmp@4.1.1%apple-clang@12.0.5+atomic+cxx-exceptions+gfps+internal-hwloc+java+legacy
  ^hwloc@2.6.0%apple-clang@12.0.5-cairo-cuda-gl-libudev+libxml2-netloc-nvml+opencl+pci+rocm+sh
  ^libxml2@9.12%apple-clang@12.0.5-python arch=darwin-bigsur-skylake
  ^xz@5.2.5%apple-clang@12.0.5-pic libs=shared,static arch=darwin-bigsur-skylake
  ^libevent@2.1.2%apple-clang@12.0.5+openssl arch=darwin-bigsur-skylake
  ^openssl@8.6%apple-clang@12.0.5+atomic+cxx-exceptions+gfps+internal-hwloc+java+legacy
  ^hwloc@2.6.0%apple-clang@12.0.5-cairo-cuda-gl-libudev+libxml2-netloc-nvml+opencl+pci+rocm+sh
  ^libxml2@9.12%apple-clang@12.0.5-python arch=darwin-bigsur-skylake
  ^xz@5.2.5%apple-clang@12.0.5-pic libs=shared,static arch=darwin-bigsur-skylake
  ^libedit@3.1-20210216%apple-clang@12.0.5 arch=darwin-bigsur-skylake
  ^perl@5.34.0%apple-clang@12.0.5+cpnm+shared+threads arch=darwin-bigsur-skylake
  ^berkeley-db@18.1.40%apple-clang@12.0.5+cxx+docs+stl patches=b231fcc4d5cff05e5c3a4814f
  ^bzzip2@1.0.8%apple-clang@12.0.5-debug-pic+shared arch=darwin-bigsur-skylake
  ^gdbm@1.19%apple-clang@12.0.5 arch=darwin-bigsur-skylake
  ^readline@8.1%apple-clang@12.0.5 arch=darwin-bigsur-skylake
```

With reuse: 16 packages were reusable



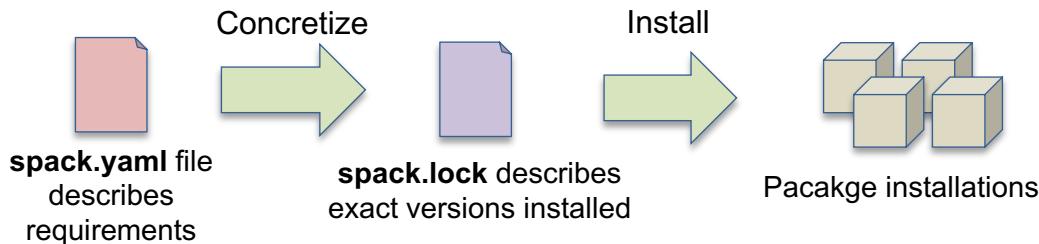
Use `spack spec` to see the results of concretization

```
$ 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
    ^bzzip2@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~cxxm~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": {}  
      }  
    }  
  }  
}
```

Hands-on Time: Binary Caches and Mirrors

Follow script at spack-tutorial.readthedocs.io



Hands-on Time: Packages and Configuration

Follow script at spack-tutorial.readthedocs.io



More Features and the Road Ahead



Environments have enabled us to add build many features to support developer workflows

```

class Cmake(Package):
    executables = ['cmake']

    @classmethod
    def determine_spect_details(cls, prefix, exes_in_prefix):
        exe_to_path = dict()
        for p in os.path.basename(p), p for p in exes_in_prefix
    ) if 'cmake' not in exe_to_path:
        return None

    cmake = spack.util.executable.Executable(exe_to_path['cmake'])
    output = cmake('--version', output=str)
    if output:
        match = re.search(r'cmake-[+]\d+(\.\d+)*', output)
        if match:
            version_str = match.group(1)
            return Spec('cmake@%s'.format(version_str))

```

```
spu
Autom
packages:
cmake:
externals:
- spec: cmake@3.15.1
  prefix: /usr/local
```

package.py

spack.yaml configuration

spack external find

Automatically find and configure external packages on the system

configuration Packages know how to run their own test suites

spack test

Packages know how to run their own test suites

```

class LsbSigsegvAutotestPackage(GNMRrrorPackage):
    """GNMRrror package for handling page faults in user mode."""

    # ... spack package contents ...

    extra_install_tests = ['tests/libsigsegv']

    def test(self):
        data_dir = self.test_suite.current_test_data_dir
        smoke_test_c = data_dir.join('smoke_test.c')

        self.run_test(
            'cc', [
                '-I' + self.prefix.include,
                '-L' + self.prefix.lib, '-lsigsegv',
                smoke_test_c,
                '-o', 'smoke_test'
            ],
            purpose='check linking')
        self.run_test(
            'smoke_test', [], data_dir.join('smoke_test.out'),
            purpose='run built smoke test')
        self.run_test('sigsegv1': ['Test passed'], purpose='check sigsegv1 output')
        self.run_test('sigsegv2': ['Test passed'], purpose='check sigsegv2 output')

```

package.py

.gitlab-ci.yml CI pipeline

spack ci

Automatically generate parallel build pipelines
(more on this later)

```

$pack:
$  specs:
$    + gromacs+mpi
$    + openmp
$    + cuda
$  container:
$    # Select the format of the recipe e.g. docker,
$    # singularity or anything else that is currently supported
$    format: docker
$  build:
$    # Select from a valid list of images
$    base:
$      image: "centos7"
$    spec1: develop
$    spec2: develop
$  no_copy:
$    # Whether or not to strip binaries
$    strip: true
$  dependencies:
$    # Additional system packages that are needed at run-time
$    os_packages:
$      - libgomp
$  extra_instructions:
$    extra_instructions:
$      file1 = 1
RUN echo "export PS1=\$(tput bold)\$(tput setaf 1)${yellow}[\${HOSTNAME}] ${cyan}\w${reset}\$(tput sgr0)" > /etc/profile.d/00000000000000000000000000000000.sh
# Labels for the image
labels:
  app: "gromacs"
  mpi: "openmp"

```

spack containerize

Turn environments into container build recipes

Join #tutorial on Slack: slack.spack.io

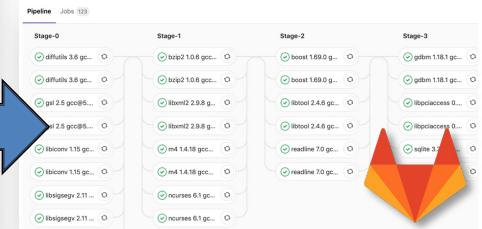
Materials: spack-tutorial.readthedocs.io

Spack environments are the foundation of Spack CI

- `spack ci` enables any environment to be turned into a build pipeline
- Pipeline generates a `.gitlab-ci.yml` file from `spack.lock`
- Pipelines can be used just to build, or to generate relocatable binary packages
 - Binary packages can be used to keep the same build from running twice
- Same repository used for `spack.yaml` can generate pipelines for project

```
spack:  
  definitions:  
    - pkgs:  
      - readline@7.0  
    - compilers:  
      - 'gcc@5.5.0'  
    - oses:  
      - os=ubuntu18.04  
      - os=centos7  
  specs:  
    - matrix:  
      - [pkgs]  
      - [compilers]  
      - [oses]  
  mirrors:  
    cloud_gitlab: https://mirror.spack.io  
gitlab-ci:  
  mappings:  
    - spack-cloud-ubuntu:  
      match:  
        - os=ubuntu18.04  
    runner_attributes:  
      tags:  
        - spack-k8s  
      image: spack/spack_builder_ubuntu_18.0  
    - spack-cloud-centos:  
      match:  
        - os=centos7  
    runner_attributes:  
      tags:  
        - spack-k8s  
      image: spack/spack_builder_centos_7  
cdash:  
  build-group: Release Testing  
  url: https://cdash.spack.io  
  project: Spack  
  site: Spack AWS Gitlab Instance
```

`spack.yaml`



Parallel GitLab build pipeline

We've made a lot of progress on compiler dependencies

- Compiler *runtime libraries* represented in the graph
 - C++, Fortran runtimes
- libc is now represented in dependency graphs on Linux
 - No more need to rely on OS tag for compatibility information
- Reuse binaries *without* their compiler needing to be configured locally
- Improved buildcache hit rate using libraries for compatibility



Compilers can now model their own runtimes

- New method, runtime_constraints, for injecting runtimes into graphs
- Currently supported for gcc, intel-oneapi
 - still working on others
- Allows solver to take libstdc++, fortran runtime compatibility into account.
- Example:
 - Intel compilers now (correctly) depend on gcc-runtime

```
class Gcc(AutotoolsPackage, GNUMirrorPackage):
    # ...

    @classmethod
    def runtime_constraints(cls, *, spec, pkg):
        """Callback function to inject runtime-related rules into the solver.

        Rule-injection is obtained through method calls of the ``pkg`` argument.

        Documentation for this function is temporary. When the API will be in its final state,
        we'll document the behavior at https://spack.readthedocs.io/en/latest/

        Args:
            spec: spec that will inject runtime dependencies
            pkg: object used to forward information to the solver
        """
        pkg("*").depends_on(
            "gcc-runtime",
            when="%gcc",
            type="link",
            description="If any package uses %gcc, it depends on gcc-runtime",
        )
        pkg("*").depends_on(
            f"gcc-runtime@{str(spec.version)}:",
            when=f"%{str(spec)}",
            type="link",
            description=f"If any package uses %{str(spec)}, "
            f"it depends on gcc-runtime@{str(spec.version)}:",
        )
```



Packages now declare the languages they depend on

- Languages are *almost* virtuals
 - HDF5 package depends on cxx and fortran
 - Handled specially internally, until compilers are nodes
- Imply a compiler *and* compiler package can specify runtime libraries to inject
- Allows solver to mix compilers *correctly*
 - Runtimes are unified like other nodes
 - *Package authors* can model toolchain properties
 - probably not for most package authors, but very powerful
- TBD:
 - Other runtimes like clang libraries and OpenMP
 - Compilers as nodes in the graph

```
class Hdf5(CMakePackage):  
    """HDF5 is a data model, library, and file format for storing and managing  
    data. It supports an unlimited variety of datatypes, and is designed for  
    flexible and efficient I/O and for high volume and complex data.  
    """  
  
    homepage = "https://portal.hdfgroup.org"  
    url = "https://support.hdfgroup.org/ftp/HDF5/releases/hdf5-1.14/hdf5-1.14.4  
    list_url = "https://support.hdfgroup.org/ftp/HDF5/releases"  
    list_depth = 3  
    git = "https://github.com/HDFGroup/hdf5.git"  
    maintainers("lrknox", "brtnfld", "byrnHDF", "gheber", "hyoklee", "lkurz")  
  
    tags = ["e4s", "windows"]  
    executables = ["^h5cc$", "^h5pcc$"]  
  
    test_requires_compiler = True  
  
    license("custom")  
  
    depends_on("cxx", type="build", when="+cxx")  
    depends_on("fortran", type="build", when="+fortran")
```



We've also added libc as a dependency

- libc is a virtual
 - glibc and musl packages are providers
 - (nearly) every graph has libc in it, via the compiler
 - Can be external or built by Spack
 - We are *not* building libc for every stack in Spack
 - Automatically detect system libc version
 - Add a node to the graph to be used for binary compatibility
 - No longer using OS tags for buildcaches
 - Now use libc for this
 - *many* more buildcache hits

```
(py311) culpo@nivola:~/PycharmProjects/spack$ spack concretize -f --ref  
=> Concretized hdf5~mpi  
- tndqhsn  hdf5@1.14.3%gcc@9.4.0~cxx~fortran~hl~ipo~java~map~mpi+sha...  
- gdviueh   ^cmake@3.27.9%gcc@8.5.0~doc+ncurses+ownlibs build_sy...  
- i5cd2jj   ^curl@8.6.0%gcc@8.5.0~gssapi~ldap~libidn2~libr...  
- icqajq4   ^nghttp2@1.57.0%gcc@8.5.0 build_system=autot...  
- xl7h3wb   ^openssl@3.2.1%gcc@8.5.0~docs+shared build_s...  
- rlipoky   ^ca-certificates-mozilla@2023-05-30%gcc@8.5.0+...  
- 45hdvpf   ^perl@5.38.0%gcc@8.5.0+cpanm+opcode+open...  
- qvzagc5   ^berkeley-db@18.1.40%gcc@8.5.0+cxx+...  
- ioufq6d   ^bzip2@1.0.8%gcc@8.5.0~debug~pic+sha...  
- enaxy2l   ^diffutils@3.10%gcc@8.5.0 build_...  
- czvftrb   ^libiconv@1.17%gcc@8.5.0 bui...  
- ku6webf   ^gdbm@1.23%gcc@8.5.0 build_system=au...  
- 3tzxgdp   ^readline@8.2%gcc@8.5.0 build_sy...  
- llqwd2j   ^gcc-runtime@8.5.0%gcc@8.5.0 build_system=gen...  
[e] fue5ca2   ^glibc@2.28%gcc@8.5.0 build_system=autotools arc...  
- sxb2sl6   ^ncurses@6.4%gcc@8.5.0~svmlinks+term...  
- ucn3h     ^gcc-runtime@9.4.0%gcc@9.4.0 build_system=generic ar...  
[e] 37zmg4    ^glibc@2.31%gcc@9.4.0 build_system=autotools arch=li...  
- vsjxwea   ^zsh@4.1.1%gcc@8.5.0~gui build_system=generic ar...  
- o76bf47   ^pkgconf@1.9.5%gcc@8.5.0 build_system=autotools arch...  
- jkwqkvs   ^zlib-ng@2.1.6%gcc@8.5.0+compat+new strategies+opt+pr...
```



Libc modeling makes for a much better buildcache experience

- Currently on develop (emacs 100% from binary):

```
(py311) culpo@nivola:~/PycharmProjects/spack$ spack install emacs
[+] /usr (external glibc-2.17-2hhcy7kzv3lfqcaschwup4uysp4hoy)
[+] /home/culpo/PycharmProjects/spack/opt/spack/linux-centos7-x86_64_v3/gcc-10.2.1/gcc-runtime-10.2.1-4gmidou73wvttvhun564olcopuijl2i
[+] /home/culpo/PycharmProjects/spack/opt/spack/linux-centos7-x86_64_v3/gcc-10.2.1/gmp-6.2.1-nkhm7cmp6samsykyksuqda77xbxnibfn
[+] /home/culpo/PycharmProjects/spack/opt/spack/linux-centos7-x86_64_v3/gcc-10.2.1/berkeley-db-18.1.40-thoi4z7lozaednxhjd4ojhw2lcsgo5g
[+] /home/culpo/PycharmProjects/spack/opt/spack/linux-centos7-x86_64_v3/gcc-10.2.1/gmake-4.4.1.ucqstlhik7pmv5ijyckmr6mxv46vgeuj
=> Installing nasm-2.15.05-e2kvtpqiaava2osr3jbyptdufjgov4dgc [6/39]
=> Fetching https://binaries.spack.io/develop/developer-tools-manylinux2014/build_cache/linux-centos7-x86_64_v3-gcc-10.2.1-nasm-2.15.05-e2kvtpqiaava2osr3jbyptdufjgov4dgc.spec.json.sig
gpg: Signature made mar 23 apr 2024, 14:38:11 CEST
gpg:           using RSA key D2C7EB3F2B05FA86590D293C04001B2E3DB0C723
gpg: Good signature from "Spack Project Official Binaries <maintainers@spack.io>" [ultimate]
=> Fetching https://binaries.spack.io/develop/developer-tools-manylinux2014/build_cache/linux-centos7-x86_64_v3/gcc-10.2.1/nasm-2.15.05/linux-centos7-x86_64_v3-gcc-10.2.1-nasm-2.15.05-e2kvtpqiaava2osr3jbyptdufjgov4dgc.spec.json.sig
gpg: Signature made mar 23 apr 2024, 14:38:11 CEST
gpg:           using RSA key D2C7EB3F2B05FA86590D293C04001B2E3DB0C723
gpg: Good signature from "Spack Project Official Binaries <maintainers@spack.io>" [ultimate]
=> Extracting nasm-2.15.05-e2kvtpqiaava2osr3jbyptdufjgov4dgc from binary cache
=> nasm: Successfully installed nasm-2.15.05-e2kvtpqiaava2osr3jbyptdufjgov4dgc
  Search: 0.00s.  Fetch: 2.61s.  Install: 0.17s.  Extract: 0.11s.  Relocate: 0.04s.  Total: 2.78s
[+] /home/culpo/PycharmProjects/spack/opt/spack/linux-centos7-x86_64_v3/gcc-10.2.1/nasm-2.15.05-e2kvtpqiaava2osr3jbyptdufjgov4dgc
=> Installing pcre-8.45-xi42ks7asbq2sqmdc7bdsmhzzhlie6m4 [7/39]
=> Fetching https://binaries.spack.io/develop/developer-tools-manylinux2014/build_cache/linux-centos7-x86_64_v3-gcc-10.2.1-pcre-8.45-xi42ks7asbq2sqmdc7bdsmhzzhlie6m4.spec.json.sig
gpg: Signature made mar 23 apr 2024, 14:38:12 CEST
gpg:           using RSA key D2C7EB3F2B05FA86590D293C04001B2E3DB0C723
gpg: Good signature from "Spack Project Official Binaries <maintainers@spack.io>" [ultimate]
=> Fetching https://binaries.spack.io/develop/developer-tools-manylinux2014/build_cache/linux-centos7-x86_64_v3/gcc-10.2.1/pcre-8.45/linux-centos7-x86_64_v3-gcc-10.2.1-pcre-8.45-xi42ks7asbq2sqmdc7bdsmhzzhlie6m4
=> pcre: Successfully installed pcre-8.45-xi42ks7asbq2sqmdc7bdsmhzzhlie6m4
  Search: 0.00s.  Fetch: 2.34s.  Install: 0.18s.  Extract: 0.14s.  Relocate: 0.03s.  Total: 2.52s
[+] /home/culpo/PycharmProjects/spack/opt/spack/linux-centos7-x86_64_v3/gcc-10.2.1/pcre-8.45-xi42ks7asbq2sqmdc7bdsmhzzhlie6m4
=> Installing tree-sitter-0.22.2-x5wmc6tind5qxo2zrswb7vyj3xuhak2d [8/39]
=> Fetching https://binaries.spack.io/develop/developer-tools-manylinux2014/build_cache/linux-centos7-x86_64_v3-gcc-10.2.1-tree-sitter-0.22.2-x5wmc6tind5qxo2zrswb7vyj3xuhak2d.spec.json.sig
gpg: Signature made mar 23 apr 2024, 14:38:22 CEST
gpg:           using RSA key D2C7EB3F2B05FA86590D293C04001B2E3DB0C723
gpg: Good signature from "Spack Project Official Binaries <maintainers@spack.io>" [ultimate]
=> Fetching https://binaries.spack.io/develop/developer-tools-manylinux2014/build_cache/linux-centos7-x86_64_v3/gcc-10.2.1/tree-sitter-0.22.2/linux-centos7-x86_64_v3-gcc-10.2.1-tree-sitter-0.22.2-x5wmc6tind5qxo2zrswb7vyj3xuhak2d
=> Extracting tree-sitter-0.22.2-x5wmc6tind5qxo2zrswb7vyj3xuhak2d from binary cache
=> tree-sitter: Successfully installed tree-sitter-0.22.2-x5wmc6tind5qxo2zrswb7vyj3xuhak2d
  Search: 0.00s.  Fetch: 2.24s.  Install: 0.00s.  Extract: 0.03s.  Relocate: 0.04s.  Total: 2.23s
```



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

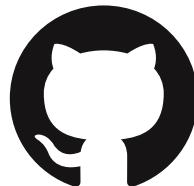
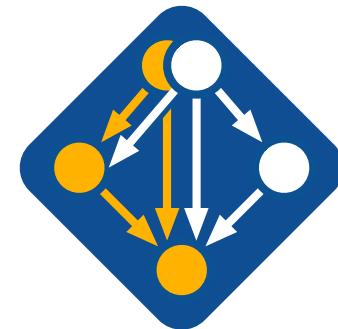
Continue to improve Windows support

Speed improvements for concretization and metadata management

Lots of exciting work ahead!

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
- Come to the HPSF BOF tomorrow at 1pm!



★ 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.

Roadmap:

Separate concretization of build dependencies

We want to:

- Build build dependencies with the "easy" compilers
- Build rest of DAG (the link/run dependencies) with the fancy compiler

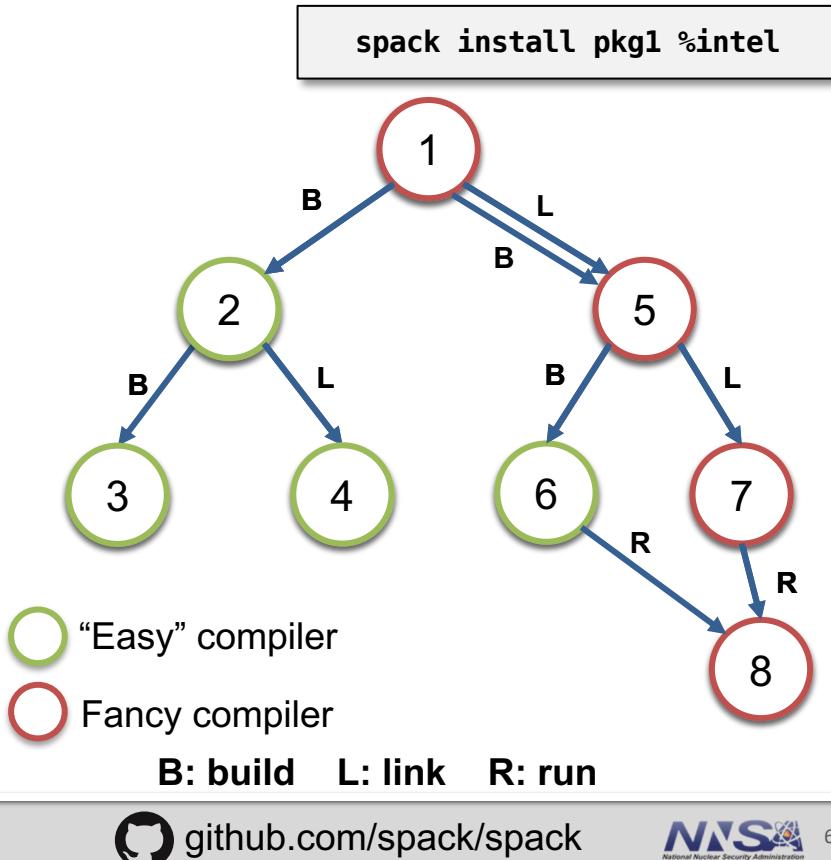
2 approaches to modify concretization:

1. Separate solves

- Solve run and link dependencies first
- Solve for build dependencies separately
- May restrict possible solutions (build \leftrightarrow run env constraints)

2. Separate models

- Allow a bigger space of packages in the solve
- Solve all runtime environments together
- May explode (even more) combinatorially



Roadmap: Compilers as dependencies

**Need separate concretization of build dependencies
to make this work**

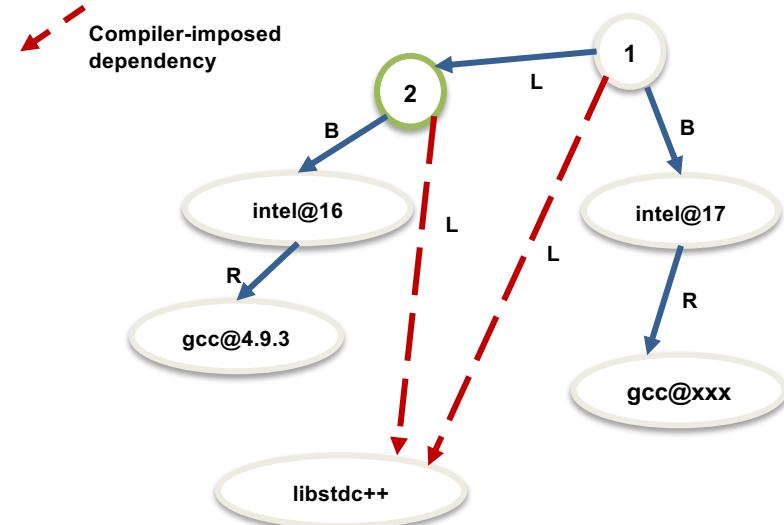
- Model compiler as build dep (not unified)
- Runtimes as link deps (unified)
- Ensure compatibility between runtimes when using multiple compilers together

**We need deeper modeling of compilers to handle
compiler interoperability**

- libstdc++, libc++ compatibility
- Compilers that depend on compilers
- Linking executables with multiple compilers

Packages that depend on languages

- Depend on `cxx@2011`, `cxx@2017`, `fortran@1995`, etc
- Depend on `openmp@4.5`, other compiler features
- Model languages, openmp, cuda, etc. as virtuals



**Compilers and runtime libs fully modeled
as dependencies**