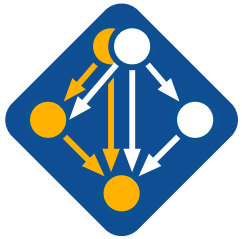


Spack 201

Intermediate Spack Tutorial

CORAL2 COE Spackathon
Los Alamos National Laboratory
November 5, 2019
Chicago, IL

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

LLNL-PRES-806064

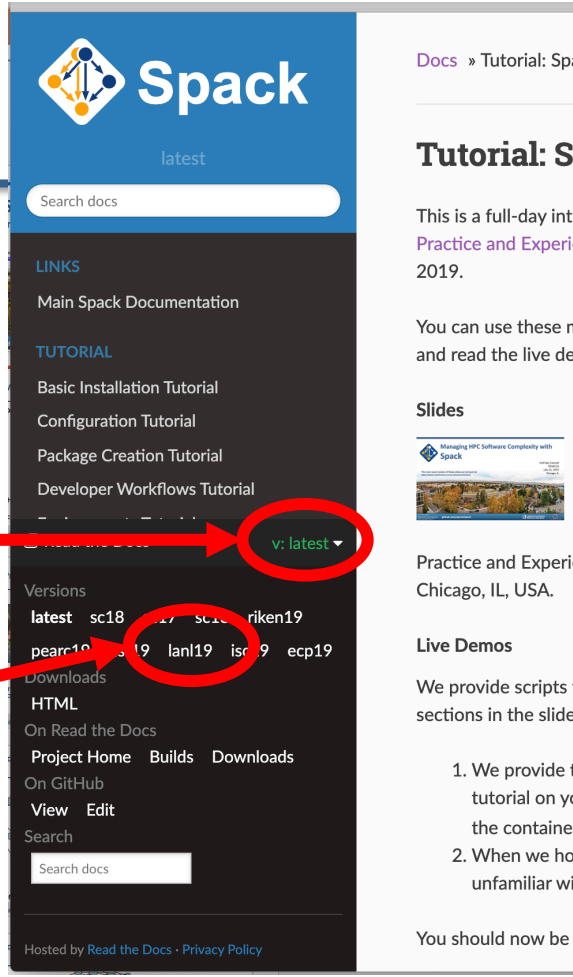
Tutorial Materials

Download the latest version of slides and handouts at:

spack-tutorial.readthedocs.io

Click v:latest at the bottom of the sidebar

Then click lanl19 to get to this version

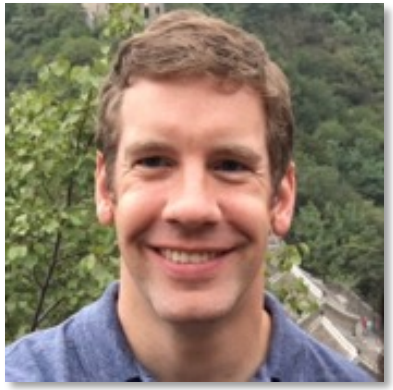


For more:

- Spack GitHub repository:
  http://github.com/spack/spack
- Spack Reference Documentation:
  http://spack.readthedocs.io

Tutorial Presenters

Greg Becker



Todd Gamblin

Lawrence Livermore National Laboratory

Spack v0.13.1 is the latest release

- **Major new features:**
  1. Chaining: use dependencies from external "upstream" Spack instances
  2. Views for Spack environments (covered today)
  3. Spack detects and builds *specifically* for your microarchitecture (not shown in tutorial)
     - named, understandable targets like skylake, broadwell, power9, zen2
  4. Spack stacks: combinatorial environments for facility deployment (covered today)
  5. Projections: ability to build easily navigable symlink trees environments (covered today)
  6. Support no-source packages (BundlePackage) to aggregate related packages
  7. Extensions: users can write custom commands that live outside of Spack repo
  8. ARM + Fujitsu compiler support
  9. GitLab Build Pipelines: Spack can generate a pipeline from a stack (covered in slides)

- **Over 3,500 packages (~700 added since last year)**

- **Full release notes:** https://github.com/spack/spack/releases/tag/v0.13.0

Tutorial Overview (times are estimates)

1.	**Welcome & Overview**	9:00 - 9:05
2.	**Core Spack Refresher**	9:05 – 9:15
3.	**Developer Workflows**	9:15 – 9:45
4.	**Environments, spack.yaml, spack.lock**	9:45 - 10:30
5.	-- 15 Minute Break --	
6.	**Spack Stacks**	10:45 - 11:15
7.	**Scripting and spack-python**	11:15 - 11:40
8.	**More New Features & the Road Ahead**	11:40 – 12:00

Core Spack Refresher:
Specs, Packages, and Concretization

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

```
$ spack install mpileaks                          unconstrained
$ spack install mpileaks@3.3                      @ custom version
$ spack install mpileaks@3.3 %gcc@4.7.3           % custom compiler
$ spack install mpileaks@3.3 %gcc@4.7.3 +threads  +/- build option
$ spack install mpileaks@3.3 cppflags="-O3 -g3"   set compiler flags
$ spack install mpileaks@3.3 target=skylake       set target microarchitecture
$ spack install mpileaks@3.3 ^mpich@3.2 %gcc@4.9.3  ^ dependency information
```

- 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 *templates*
They use a simple Python DSL to define how to build

Not shown: patches, **resources**, **conflicts**, other directives.

```
from spack import *

class Kripke(CMakePackage):
    """Kripke is a simple, scalable, 3D Sn deterministic particle
       transport proxy/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):
        # Kripke does not provide install target, so we have to copy
        # things into place.
        mkdirp(prefix.bin)
        install('../spack-build/kripke', prefix.bin)
```

Base package
(CMake support)

Metadata at the class level

Versions

Variants (build options)

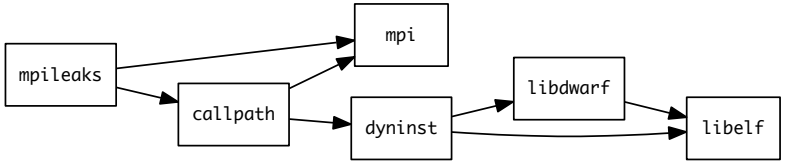
Dependencies
(note: same spec syntax)

Install logic
in instance methods

Don't typically need `install()` for `CMakePackage`, but we can work around codes that don't have it.

Spack handles combinatorial software complexity.

Dependency DAG



Installation Layout

Hash

```
spack/opt/
  linux-x86_64/
    gcc-4.7.2/
      mpileaks-1.1-0f54bf34cadk/
    intel-14.1/
      hdf5-1.8.15-lkf14aq3nqiz/
  bgq/
    xl-12.1/
      hdf5-1-8.16-fqb3a15abrwx/
  ...
```

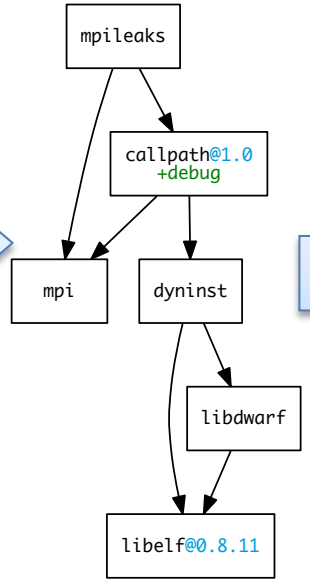
- Each unique dependency graph is a unique ***configuration***.
- Each configuration installed in a unique directory.
  - Configurations of the same package can coexist.
- **Hash** of entire directed acyclic graph (DAG) is appended to each prefix.
- Installed packages automatically find dependencies
  - Spack embeds RPATHs in binaries.
  - No need to use modules or set LD_LIBRARY_PATH
  - Things work *the way you built them*

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

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

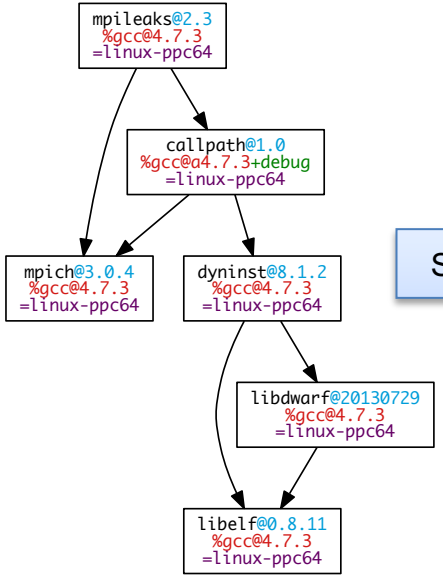
User input: *abstract* spec with some constraints

Normalize



Abstract, normalized spec with some dependencies.

Concretize



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

Store

spec.yaml

```
spec:
- mpileaks:
    arch: linux-x86_64
    compiler:
      name: gcc
      version: 4.9.2
    dependencies:
      adept-utils: kszrtkpbzac3ss2ixcjkcorlaybnptp4
      callpath: bah5f4h4d2n47mgycej2mtrnrivvxy77
      mpich: aa4ar6ifj23yijqmdabeakpejcli72t3
    hash: 33hjjhxi7p6gyzn5ptgyes7sghyprujh
    variants: {}
    version: '1.0'
- adept-utils:
    arch: linux-x86_64
    compiler:
      name: gcc
      version: 4.9.2
    dependencies:
      boost: teesjv7ehpe5ksspjim5dk43a7qnowlq
      mpich: aa4ar6ifj23yijqmdabeakpejcli72t3
    hash: kszrtkpbzac3ss2ixcjkcorlaybnptp4
    variants: {}
    version: 1.0.1
- boost:
    arch: linux-x86_64
    compiler:
      name: gcc
      version: 4.9.2
    dependencies: {}
    hash: teesjv7ehpe5ksspjim5dk43a7qnowlq
    variants: {}
    version: 1.59.0
...
```

Detailed provenance is stored with the installed package

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
```

Developer Workflows

Follow script at http://spack-tutorial.rtfd.io
Under “Tutorial: Spack 101"

Environments,
`spack.yaml` and `spack.lock`

Follow script at **http://spack-tutorial.rtfd.io**
Under “Tutorial: Spack 101"

Spack Stacks

Follow script at http://spack-tutorial.rtfd.io
Under “Tutorial: Spack 101"

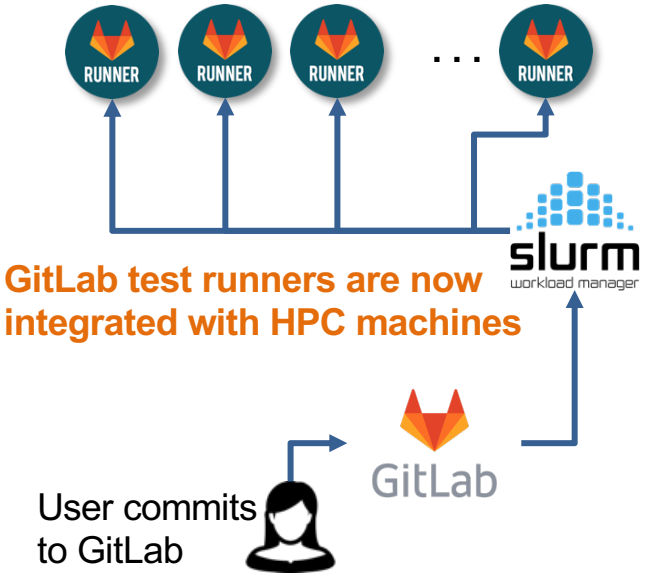
Scripting and spack-python

Follow script at **http://spack-tutorial.rtfd.io**
Under “Tutorial: Spack 101"

More New Features and the Road Ahead

We have been heavily involved in the ECP CI project.

- We have added security features to the open source GitLab product.
  - Integration with center identity management
  - Integration with schedulers like SLURM, LSF
- We are democratizing testing at Livermore Computing
  - Users can run tests across 30+ machines by editing a file
  - Previously, each team had to administer own servers
- ECP sites are deploying GitLab CI for users
  - All HPC centers can leverage these improvements
  - NNSA labs plan to deploy common high-side CI infrastructure
  - We are developing new security policies to allow external open source code to be tested safely on key machines



Spack now understands specific target microarchitectures

- We have developed a cross-platform library to detect and compare microarchitecture metadata
  - Detects based on /proc/cpuinfo (Linux), sysctl (Mac)
  - Allows comparisons for compatibility, e.g.:

```
skylake > broadwell
zen2 > x86_64
```

- Key features:
  - Know which compilers support which chips/which flags
  - Determine compatibility
  - Enable creation and reuse of optimized binary packages
  - Easily query available architecture features for portable build recipes

- We will be extracting this as a standalone library for other tools & languages
  - Hope to make this standard!

```
$ spack arch --known-targets
Generic architectures (families)
    aarch64  ppc64  ppc64le  x86  x86_64

IBM - ppc64
    power7  power8  power9

IBM - ppc64le
    power8le  power9le

AuthenticAMD - x86_64
    barcelona  bulldozer  piledriver  steamroller  excavator  zen  zen2

GenuineIntel - x86_64
    nocona   westmere     haswell    mic_knl         cascadelake
    core2    sandybridge  broadwell  skylake_avx512  icelake
    nehalem  ivybridge    skylake    cannonlake

GenuineIntel - x86
    i686  pentium2  pentium3  pentium4  prescott
```

Extensive microarchitecture knowledge

```
class OpenBlas(Package):

    def configure_args(self, spec):
        args = []
        if 'avx512' in spec.target:
            args.append('--with-avx512')
        ...
        return args
```

Simple feature query

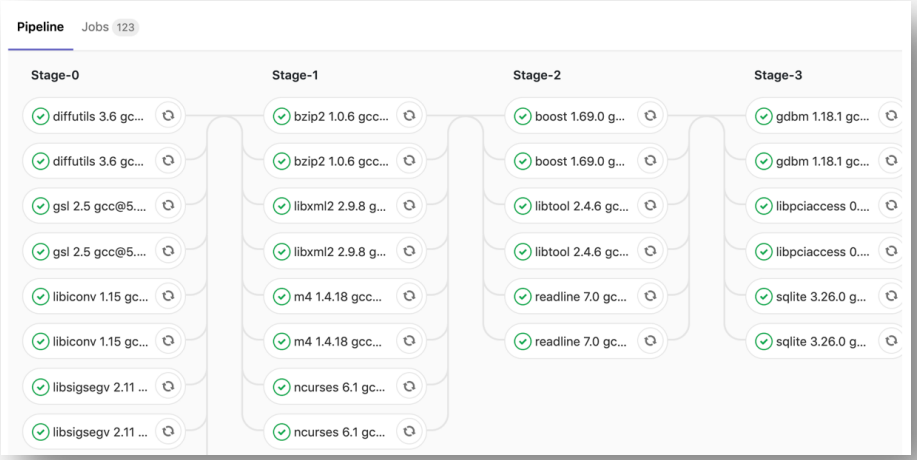
```
$ spack install lbann target=cascadelake

$ spack install petsc target=zen2
```

Specialized installations

Spack has added GitLab CI integration to automate package build pipelines

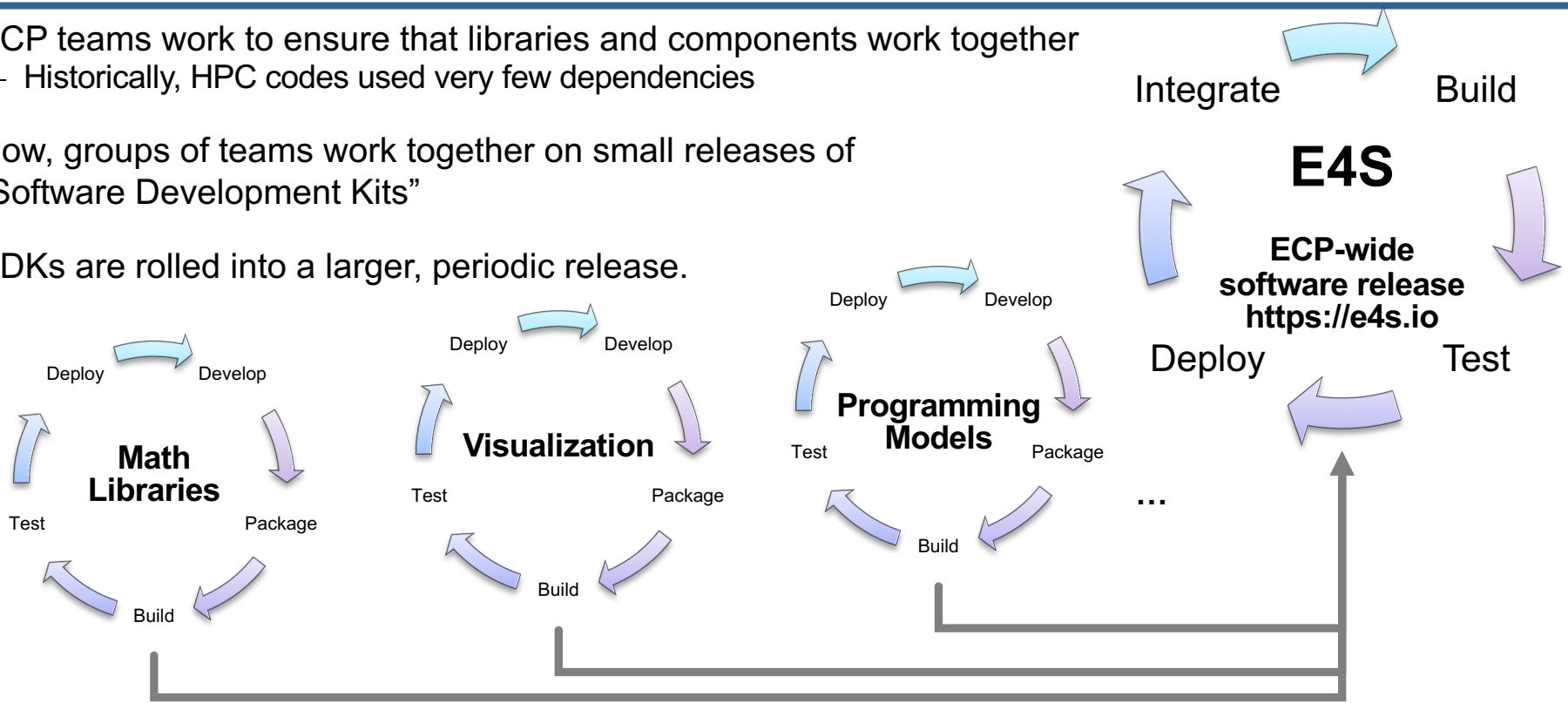
- Builds on Spack environments
  - Support auto-generating GitLab CI jobs
  - Can run in a Kube cluster or on bare metal runners at an HPC site
  - Sends progress to CDash



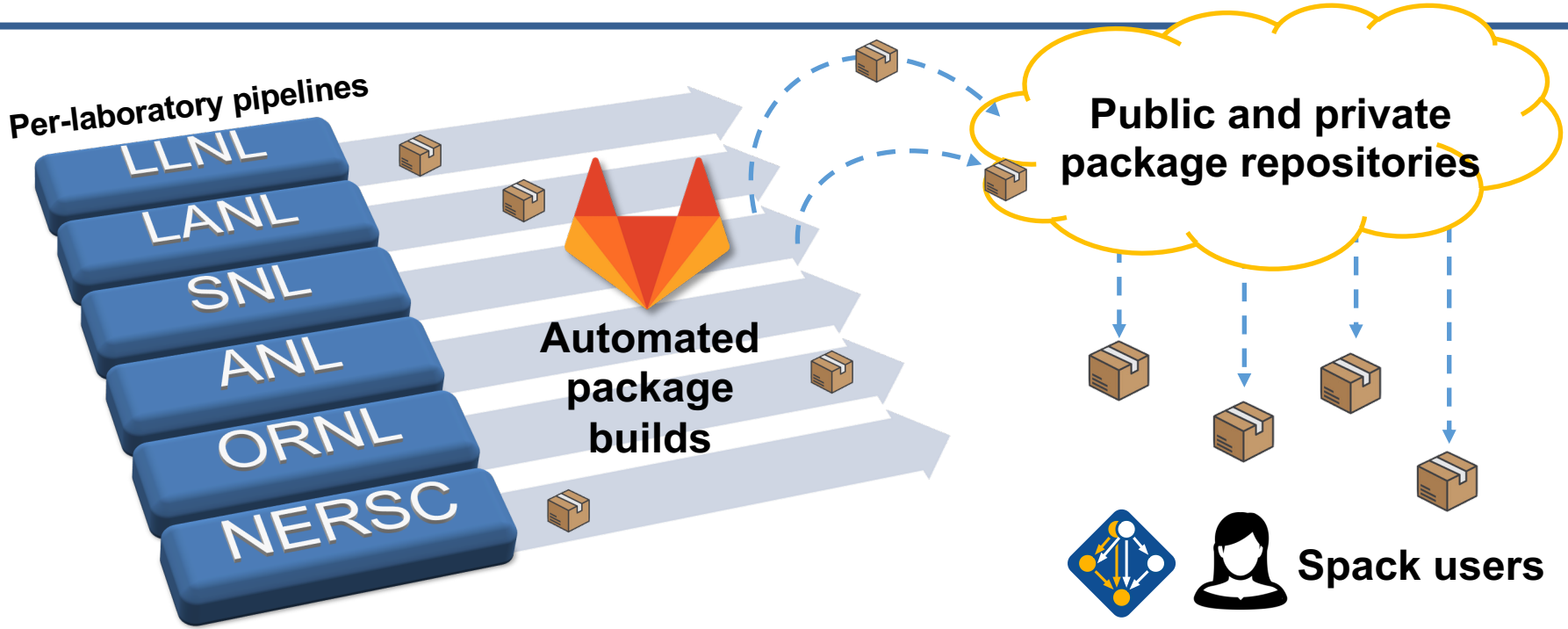

```yaml
spack:
  definitions:
  - pkgs:
    - readline@7.0
  - compilers:
    - '%gcc@5.5.0'
  - oses:
    - os=ubuntu18.04
    - os=centos7
  specs:
  - matrix:
    - [$pkgs]
    - [$compilers]
    - [$oses]
  mirrors:
    cloud_gitlab: https://mirror.spack.io
  gitlab-ci:
    mappings:
      - spack-cloud-ubuntu:
        match:
          - os=ubuntu18.04
        runner-attributes:
          tags:
            - spack-k8s
          image: spack/spack_builder_ubuntu_18.04
      - 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
```

ECP is working towards a periodic, hierarchical release process

- ECP teams work to ensure that libraries and components work together
  - Historically, HPC codes used very few dependencies
- Now, groups of teams work together on small releases of "Software Development Kits"
- SDKs are rolled into a larger, periodic release.



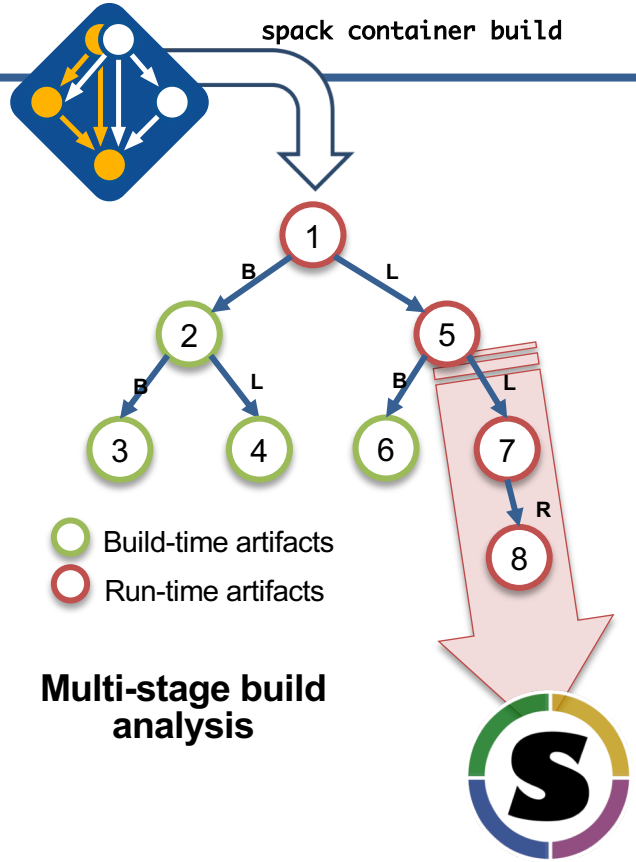
Automated builds using ECP CI will enable a robust, widely available HPC software ecosystem.



With pipeline efforts at E6 labs, users will no longer need to *build* their own software for high performance.

Spack focus areas in FY20

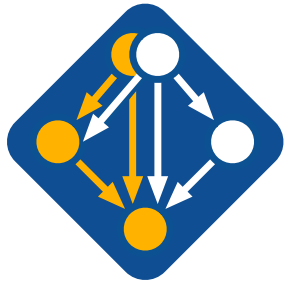
- **Multi-stage container generation with Spack**
  - Add support to Spack to generate *multi-stage* container builds that exclude build dependencies from artifacts automatically
- **Build Hardening with Spack Pipelines**
  - Continue working with E4S team to harden container builds
- **Parallel builds**
  - "`srun spack install`" will use the entire allocation to build
- **New concretizer based on fast ASP/SAT solvers**
- **Improved dependency models for compilers**
  - icpc depends on g++ for its libstdc++, and other ABI nightmares



Multi-stage build analysis

Join the Spack community!

- There are lots of ways to get involved!
  - Contribute packages, documentation, or features at **github.com/spack/spack**
  - Contribute your configurations to **github.com/spack/spack-configs**



- Talk to us!
  - Join our **Google Group** (see GitHub repo for info)
  - Join our **Slack channel** (see GitHub repo for info)
  - Submit GitHub issues and talk to us!

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

Follow us on Twitter!
@spackpm

We hope to make distributing & using HPC software easy!

Lawrence Livermore
National Laboratory