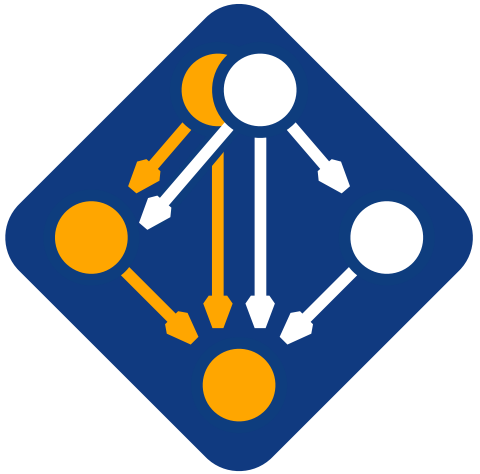
The Future of Spack



HEP Software Developer Tools & Packaging Working Group
October 20, 2021

Todd Gamblin
Advanced Technology Office
Livermore Computing

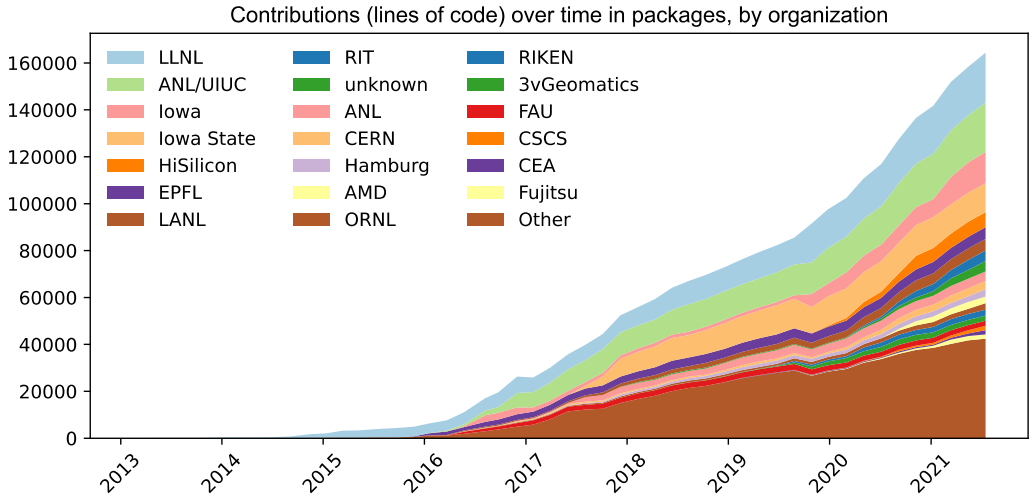
LLNL-PRES-826942
This work was performed under the auspices of the U.S. Department of Energy by Lawrence Livermore National Laboratory under contract DE-AC52-07NA27344. Lawrence Livermore National Security, LLC

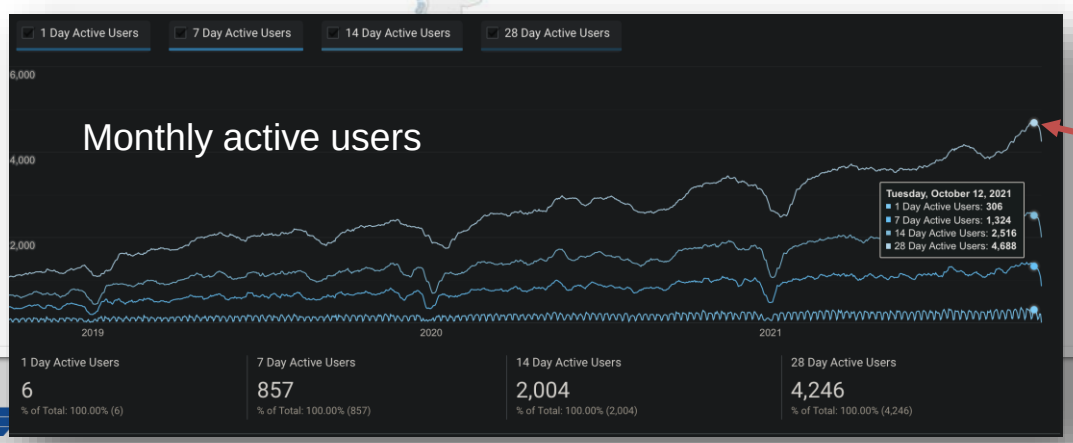
Lawrence Livermore National Laboratory

The Spack community continues to grow!

5,900+ software packages
900+ contributors

Package contribution rate increased in 2020





Broke 4,600 monthly active users on docs site in October 2021

Spack core development is sustained by a number of key DOE programs and partners

- Common Compute Environment
- Computation Systems & Software Environment
- Facility Operations & User Support

- Packaging Technologies Project
- Facility Integration
- Testing Task Force

Lawrence Livermore National Laboratory

radiuss

https://software.llnl.gov/radiuss

LDRD
LABORATORY DIRECTED RESEARCH & DEVELOPMENT

np-complete S.r.l.

TECH-X

Kitware

UNIVERSITY OF OREGON

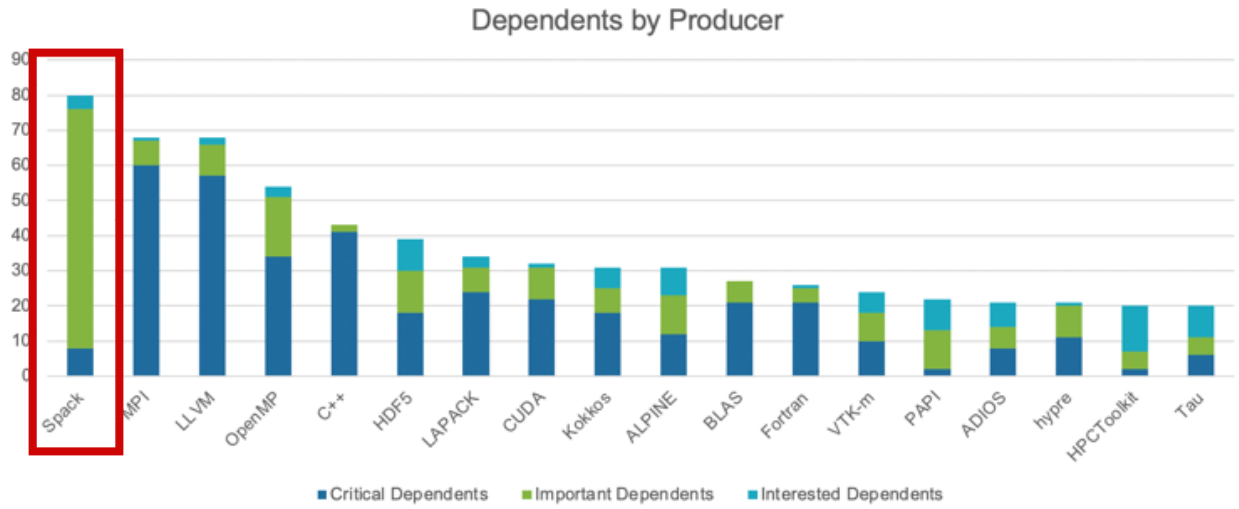
aws

Spack is a critical part of ECP

EXASCALE COMPUTING PROJECT

https://e4s.io

- Spack is used to integrate the software ecosystem for the three upcoming U.S. exascale systems
- ECP has built the Extreme Scale Scientific Software Stack (E4S) with Spack – more at https://e4s.io



Spack is the most depended-upon project in ECP

We have worked with HPE to improve Spack support in the Cray Programming Environment

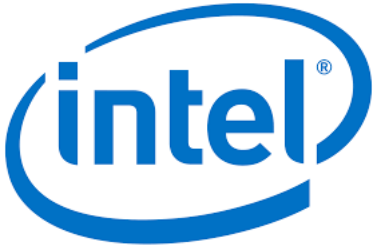
- Using Cray Programming Environment's MPI, libsci, etc. currently requires a fair amount of configuration
  - Users have to register externals and go through modules
  - PrgEnvs make it hard to be precise about dependencies
- PE team has worked with us to develop a JSON format to describe PE contents
- PE team has also been responsive with other requests:
  - Standalone compiler executables (no modules)
  - Standalone MPI wrappers (like normal MPICH)
- Things are getting better!


```
{
  "name": "cray-netcdf",
  "version": "4.7.4.0",
  "arch": {
    "platform": "cray",
    "platform_os": "sles15",
    "target": {
      "name": "x86_64"
    }
  },
  "compiler": {
    "name": "cce",
    "version": "9.0.0"
  },
  "parameters": {
    "shared": true,
    "dap": false,
    "hdf4": false,
    "parallel-netcdf": false,
    "mpi": true,
    "jna": false,
    "doxygen": false,
    "doc": false
  },
  "provides": "netcdf-fortran",
  "dependencies": {
    "cray-mpich": {
      "hash": "mvl4uwf63n4l7tuwfrdyqfxmmit7yu54",
      "type": [
        "link"
      ]
    },
    "cray-hdf5": {
      "hash": "tdma2n3hxn25lhlxn7dbkvt23jo2f3mc",
      "type": [
        "link"
      ]
    },
    "zlib": {
      "version": "1.2.11",
      "type": [
        "link"
      ]
    }
  },
  "prefix": "/opt/cray/pe/netcdf-hdf5parallel/4.7.4.0/cce/90",
  "rpm": "cray-netcdf-4.7.4.0-crayclang90-202007092040.ac3e2015515ab-0.sles15.x86_64.rpm",
  "hash": "hsm5hatcfepa6hbfynpu2iv6cxfiod7i"
},
```


Cray PE JSON descriptor

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

- **AWS** invests in cloud credits for Spack build farm
  - Joint Spack tutorial in July with AWS had 125+ participants
  - Joint AWS/AHUG Spack Hackathon drew 60+ participants
- **AMD** has contributed ROCm packages and compiler support
  - 55+ PRs mostly from AMD, also others
  - ROCm, HIP, aocc packages are all in Spack now
- **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



Spack GitHub remains a very busy place

September 20, 2021 – October 20, 2021

Period: 1 month

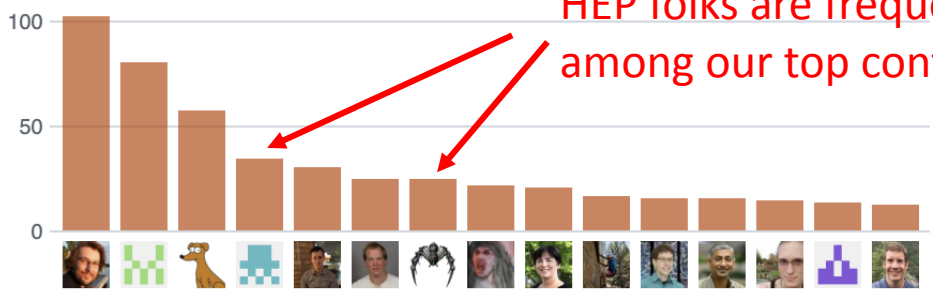
Overview

732 Active Pull Requests

143 Active Issues

634	98	79	64
Merged Pull Requests	Open Pull Requests	Closed Issues	New Issues

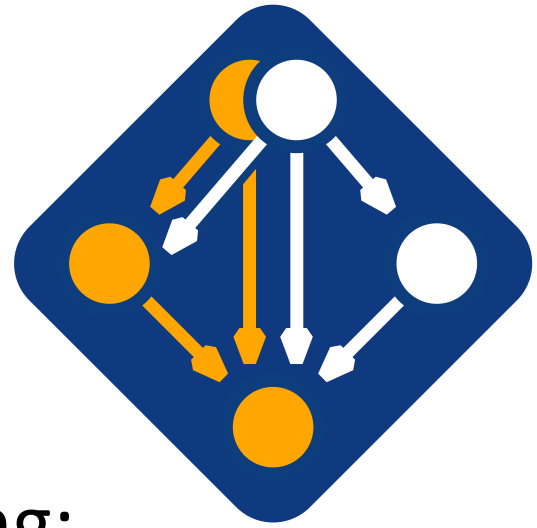
Excluding merges, **179 authors** have pushed **641 commits** to develop and **696 commits** to all branches. On develop, **1,273 files** have changed and there have been **22,569 additions** and **5,572 deletions**.



HEP folks are frequently among our top contributors!

What are the priorities going forward?

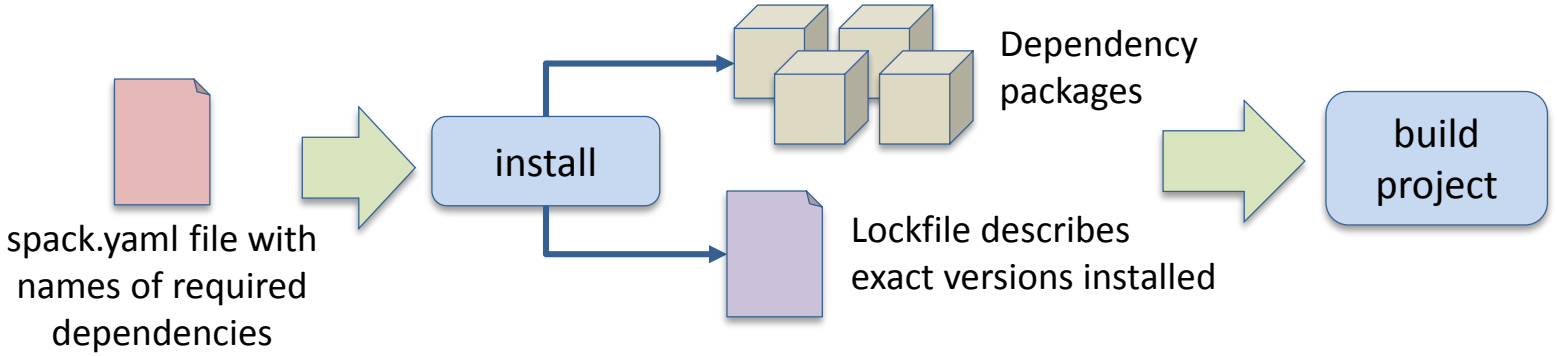
1. Improving support for GPUs and exascale machines (Cray PE)
2. Developer workflows
3. CI, Testing, and Binary Distribution
4. Deeper modeling of the software ecosystem
5. Broader reach into non-traditional HPC, including:
   - C++ community
   - Windows
   - Cloud environments



Our strategy going forward is to focus on building and testing for the compiler/runtime stacks on EA systems

- **We added much-needed support for new vendor compilers**
  - **oneapi**: Intel, **nvhpc**: NVIDIA, **aocc**: AMD
- **Working to add tests so that we can use new spack test capability on EA systems**
  - Community can integrate smoke tests into Spack packages
  - We will be running continuous smoke tests in CI for the ECP stack
- **GPU integration across the stack will be an ongoing focus**
  - Focus on CUDA and HIP packages and common usage
  - Focus on interoperability for mixed compilers, continue to work with vendors
- **Aggressive tests of relocatable binary builds on all EA systems**
  - Keep things working with constant pull request testing

Spack's environment abstraction allows users to work with a subset of packages



- Two files:
  - spack.yaml describes project requirements and configuration
  - spack.lock describes exactly what was installed
    - Allows you to reproduce an installation
- An environment can be versioned in a repository
  - Technique used by uberenv, serac, other projects to manage dependencies
- MAPP repository structure maps fairly well to a Spack environment

Simple spack.yaml file

```yaml
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": {
             "cxx": false,
             "debug": false,
             "fortran": false,
             "hl": false,
             "mpi": true,
```

We have built a number of developer workflow features on top of Spack environments (similarities with SpackDev)

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

    @classmethod
    def determine_spec_details(cls, prefix, exes_in_prefix):
        exe_to_path = dict(
            (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.*version\s+(\S+)', output)
            if match:
                version_str = match.group(1)
                return Spec('cmake@{0}'.format(version_str))
```

package.py

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

spack.yaml configuration

spack external find

Automatically find and configure external packages on the system

spack test

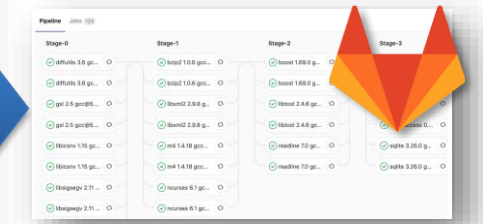
Packages know how to run their own test suites

```python
class Libsigsegv(AutotoolsPackage, GNUMirrorPackage):
    """GNU libsigsegv is a library for handling page faults in user mode."""

    # ... spack package contents ...

    extra_install_tests = 'tests/.libs'

    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%s' % self.prefix.include,
                '-L%s' % 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

```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
```

spack.yaml



.gitlab-ci.yml CI pipeline

spack ci

Automatically generate parallel build pipelines (more on this later)

```yaml
spack:
  specs:
  - gromacs+mpi
  - mpich

  container:
    # Select the format of the recipe e.g. docker,
    # singularity or anything else that is currently su
    format: docker

    # Select from a valid list of images
    base:
      image: "centos:7"
      spack: develop

    # Whether or not to strip binaries
    strip: true

    # Additional system packages that are needed at run
    os_packages:
    - libgomp

    # Extra instructions
    extra_instructions:
      final: |
RUN echo 'export PS1="\[$(tput bold)\]\[$(tput setaf 1)

    # Labels for the image
    labels:
      app: "gromacs"
      mpi: "mpich"
```

```
# Build stage with Spack pre-installed and ready to be used
FROM spack/centos7:latest as builder

# What we want to install and how we want to install it
# is specified in a manifest file (spack.yaml)
RUN mkdir /opt/spack-environment \
&&  (echo "spack:" \
&&   echo "  specs:" \
&&   echo "  - gromacs+mpi" \
&&   echo "  - mpich" \
&&   echo "  concretization: together" \
&&   echo "  config:" \
&&   echo "    install_tree: /opt/software" \
&&   echo "  view: /opt/view") > /opt/spack-environment/spack.yaml

# Install the software, remove unnecessary deps
RUN cd /opt/spack-environment && spack install && spack gc -y

# Strip all the binaries
RUN find -L /opt/view/* -type f -exec readlink -f '{}' \; | \
    xargs file -i | \
    grep 'charset=binary' | \
    grep 'x-executable\|x-archive\|x-sharedlib' | \
    awk -F: '{print $1}' | xargs strip -s

# Modifications to the environment that are necessary to run
RUN cd /opt/spack-environment && \
    spack env activate --sh -d . >> /etc/profile.d/z10_spack_environment.sh

# Bare OS image to run the installed executables
FROM centos:7

COPY --from=builder /opt/spack-environment /opt/spack-environment
COPY --from=builder /opt/software /opt/software
COPY --from=builder /opt/view /opt/view
COPY --from=builder /etc/profile.d/z10_spack_environment.sh /etc/profile.d/z10_spack_env

RUN yum update -y && yum install -y epel-release && yum update -y
 && yum install -y libgomp \
 && rm -rf /var/cache/yum  && yum clean all

RUN echo 'export PS1="\[$(tput bold)\]\[$(tput setaf 1)\][gromacs]\[$(tput setaf 2)\]\u\[$(tput
```

spack containerize

Turn environments into container build recipes

spack develop lets developers work on many packages at once

- Workflow:
  - Set up an environment for a code
  - Develop multiple packages from its dependencies
  - Easily rebuild with changes
- You need to supply a version to spack develop
  - Spack needs to know what version to check out.
  - Can be a git version, tarball, etc.
- Develop installations are install in the regular Spack install tree, with a unique hash

```
$ spack env activate .
$ spack add myapplication
$ spack develop axom@0.4.0
$ spack develop mfem@4.2.0



$ ls
spack.yaml    axom/    mfem/



$ cat spack.yaml
spack:
    specs:
        - myapplication    # depends on axom, mfem

    develop:
        - axom @0.4.0
        - mfem @develop
```

We have added git versioning to Spack

- Users can now specify a full, 40-char git commit as a version
  - Works in environments or on the command line

```
$ spack install zlib @53ce2713117ef2a8ed682d77b944df991c499252
```

- This was tricky because we needed a way to compare a commit to a version
  - MBS only needs to be able to fetch by commit, not compare
  - Packages have conditional logic with versions
  - We can compare versions to commits based on tags in a repository

- We developed an internal representation for commit versions
  - Lexicographic tuple comparison:

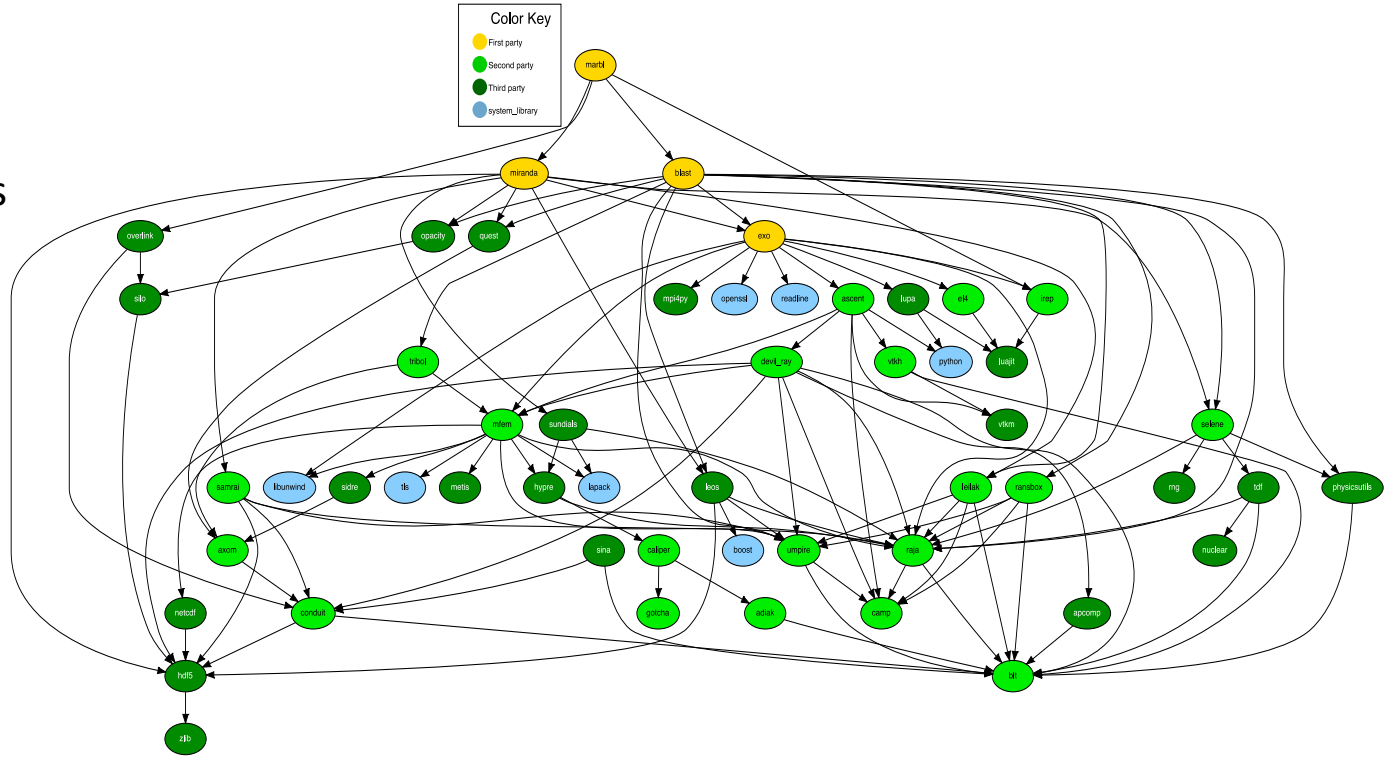
    (<version>, "", <commits since prior tag>)

  - Comes before any <version>.x
  - Allows commits to be compared by distance between versions.

- We can iterate on these semantics
  - there are certain types of versioning schemes for which they won't work

MARBL has about 50 dependencies (and growing)

- Not unlike other LLNL codes, but…
- MARBL is more deeply modular than prior codes
  - Designed to support modular *physics*
  - MARBL itself has two hydro options: Miranda & Blast
  - Code, build structure both assume that a simulation is comprised of *packages*
- Needed a way to simplify modular workflows
  - Need to work on several repos at once
  - Changes to the code are multiple pull requests
- LLNL and other HPC users likely won't use mono-repos
  - Issues:
    - Managing permissions
    - Code timescales
    - Independence of teams
  - Systems like Bazel are extremely hard to get working on non-Google environments.



c/o Robert Blake

The AML team has used Spack environments to accelerate their workflow for PyTorch

- **LLNL Applied ML team needed to deploy**
  - PyTorch + Kull development environment
  - On ppc64le with system MPI
- **Before Spack**
  - Everybody built from scratch
  - People wrote scripts and passed them around
  - **Days were spent trying to debug build differences**
- **After spack**
  - Versioned reproducible spack environments in a git repo
  - Standard environments in a shared team directory
  - **Team members can set up a reproducible, customizable environment in ~20 minutes.**
    - Change python version, PyTorch version on the fly
    - Leverage binary caches to avoid redundant builds.

```
spack:
 specs:
 - py-horovod
 - py-torch
 - python
 - py-h5py

packages:
  all:
    providers:
      mpi:
      - mvapich2@2.3
      lapack:
      - openblas threads=openmp
      blas:
      - openblas threads=openmp
    buildable:true
    variants: [+cuda cuda_arch=37]
    compiler: [gcc@7.3.0]
  ...
  python:
    version: [3.8.6]
  cudnn:
    version:
    - 8.0.4.30-11.1-linux-x64
  py-torch:
    buildable:true
    variants: +cuda +distributed
  mvapich2:
    externals:
    - spec: mvapich2@2.3.1%gcc@7.3.0
      prefix: /usr/tce/packages/mvapich2/mvapich2-2.3-gcc-7.3.0
  compilers:
  - compiler:
      operating_system: rhel7
      paths:
        cc: /usr/tce/packages/gcc/gcc-7.3.0/bin/gcc
        cxx: /usr/tce/packages/gcc/gcc-7.3.0/bin/g++
```

spack.yaml file

This stack is not as complex as MARBL, but seemed like a good basis.

Under this new workflow, users can describe their environment in a spack.yaml

- First section is familiar
  - List of packages with hashes
- spack.yaml contains:
  - Hashes
  - Per-package options
  - Compiler options
  - Flags
  - External packages
  - Local package recipe repositories
  - Etc.
- If this is too long, some of this can be moved to external includes

```
spack:
  specs:
  - marbl                    @develop build_type=Release
  - miranda                  @develop
  - blast                    @wktexports
  - exo                      @wktexports
  - adiak                    @950e3bfb91519ecb7b7ee7fa3063bfab23c0e2c9
  - ascent ~fortran~openmp   @587f6cf9503ef6176e59a046f6331baed5e36ce6
  - axom ~lua~openmp         @587f6cf9503ef6176e59a046f6331baed5e36ce6
  - blt                      @43022da4dfed5a50a02fbd0355defd03f12157cd
  - caliper~libdw            @85601f48e7f883fb87dec85e92c849eec2bb61f7
  - camp                     @85601f48e7f883fb87dec85e92c849eec2bb61f7
  - care                     @7f43ed9ed8400f6173b8434b6471142a8ffd4882
  - chai                     @d3282bc95c533efb90ec0a06085e455daa97df6b
  - conduit                  @f54f834eb8aaff4fc97613e04cfdb360997867be
  - dray ~test~utils~openmp  @c0bee76f2dce29139bde1084bf085d7d1c1b01b4
  - el4                      @aded490988f1d0a11ff74f9be7135d95e25e90ca
  - glvis                    @20aeb2c03ce70f445232dba74179e03c94de0c2c
  - gotcha                   @e0455990e57e5b74e16343816cd0d2d4f38d65de
  - irep                     @5d4d2893b25c4dfe4ad05dd6d8110179980c2a6b
  - leilak                   @1886056c398a6919bf8cce4216732fc1d8643954
  - mfem +shared             @9d8043b9e78dcdcd86639bbb28d3bd7b514fb5e2
  - raja ~openmp             @9cb6370bb2868e35ebba23cdce927f5f7f9da530
  - ransbox                  @edf072bfa7b3f6e0fd6eb106abbe65ae5f677abe
  - samrai                   @39017121bda44fff713fe3b01cb1e063be93023b
  - selene                   @6f9b15713c738d70b125bc08aef72925d961a02e
  - spheral                  @8cc54824c2937405203c3803ab44960fc26d506d
  - tribol                   @b9185d317bf14d87462ca345086931580c591eb4
  - umpire ~openmp           @5201a47a35e3844160dcbecd0916f8c96aa7dd07
  - vtkh                     @cd6004c94b083b096fda5f994b491b8229dacd79
  - hdf5                     @:1.8 +cxx+fortran~mpi
  - netcdf-c ~mpi
  - python                   @3.7.2
  - boost                    @1.76.0
  - leos                     @8.3.4.1
  view: false
  concretization: together

  repos::
    - ~/src/llnl.wci.mapp
    - $spack/var/spack/repos/builtin
    - ~/src/llnl.wci

  compilers:
  - compiler:
      spec: intel@18.0.2
      paths:
        cc: /usr/tce/bin/icc-18.0.2
        cxx: /usr/tce/bin/icpc-18.0.2
        f77: /usr/tce/bin/ifort-18.0.2
        fc: /usr/tce/bin/ifort-18.0.2
      flags: {}
      operating_system: rhel7
      target: x86_64
      modules: [gcc/4.9.3, intel/18.0.2]
```

options, versions/hashes

package repos

compiler info

```
packages:
  all:
    compiler: [intel@18.0.2]
    providers:
      mpi: [mvapich2]
      blas: [netlib-lapack]
      lapack: [netlib-lapack]
  hypre:
    variants: +shared
  mpi:
    buildable: false
    externals:
    - spec: mvapich2@2.3%intel@18.0.2 process_managers=slurm arch=linux-rhel7-ivybridge
      prefix: /usr/tce/packages/mvapich2/mvapich2-2.3-intel-18.0.2
  blas:
    buildable: false
  lapack:
    buildable: false
  netlib-lapack:
    buildable: false
    externals:
    - spec: netlib-lapack@3.6.1+shared
      prefix: /usr
  cuda:
    buildable: false
    externals:
    - spec: cuda@10.2
      prefix: /opt/cudatoolkit/10.2
  # Basic build deps
  autoconf:
    buildable: false
    externals:
    - spec: autoconf@2.69
      prefix: /usr
  automake:
    buildable: false
    externals:
    - spec: automake@1.13.4
      prefix: /usr
  bzip2:
    buildable: false
    externals:
    - spec: bzip2@1.0.6
      prefix: /usr
  cmake:
    version: [3.14.5]
    buildable: false
    externals:
    - spec: cmake@3.14.5
      prefix: /usr/tce/packages/cmake/cmake-3.14.5
  gettext:
    buildable: false
    externals:
    - spec: gettext@0.19.8.1
      prefix: /usr
  libtool:
    buildable: false
    externals:
    - spec: libtool@2.4.2
      prefix: /usr
  m4:
    buildable: false
    externals:
    - spec: m4@1.4.16
      prefix: /usr
  perl:
    buildable: false
    externals:
    - spec: perl@5.16.3
      prefix: /usr
  pkg-config:
    buildable: false
    externals:
    - spec: pkg-config@0.27.1
      prefix: /usr
  tar:
    buildable: false
    externals:
    - spec: tar@1.26
      prefix: /usr
```

external package prefs

MPI

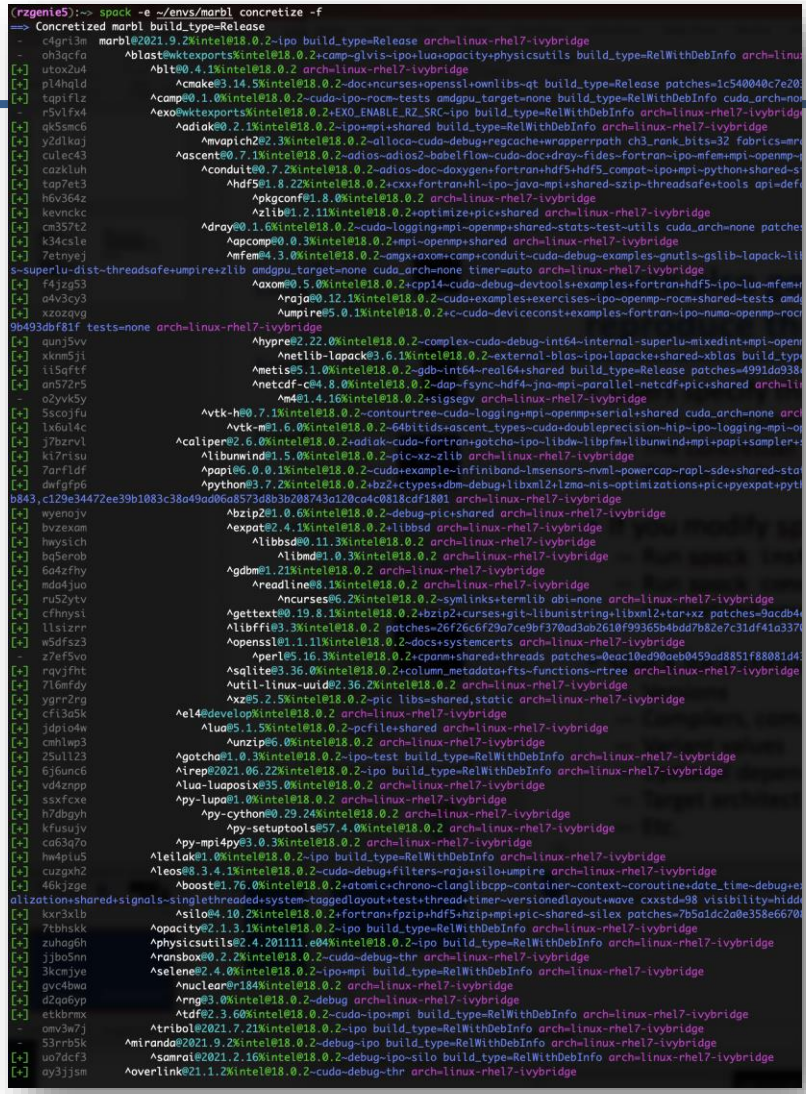
BLAS/LAPACK

build dependencies

Current MARBL spack.yaml

Spack generates a spack.lock file that enables you to reproduce the environment

- Users specify their constraints in spack.yaml
  - The rest of configuration is automated by the *concretizer*
  - The concretizer is a constraint solver that reconciles package requirements with yours
  - Details are beyond the scope of this presentation
- If you modify spack.yaml, you can either:
  - Run **spack install** again (this concretizes before installing)
  - Run **spack concretize --force** to see the concretized environment before installing (shown at right)
- spack.lock contains all the decisions the concretizer made:
  - Versions
  - Compilers, compiler versions
  - Variant values
  - Optional dependencies
  - Target architecture
- Open question: how best to manage spack.lock files



Fully concretized MARBL environment

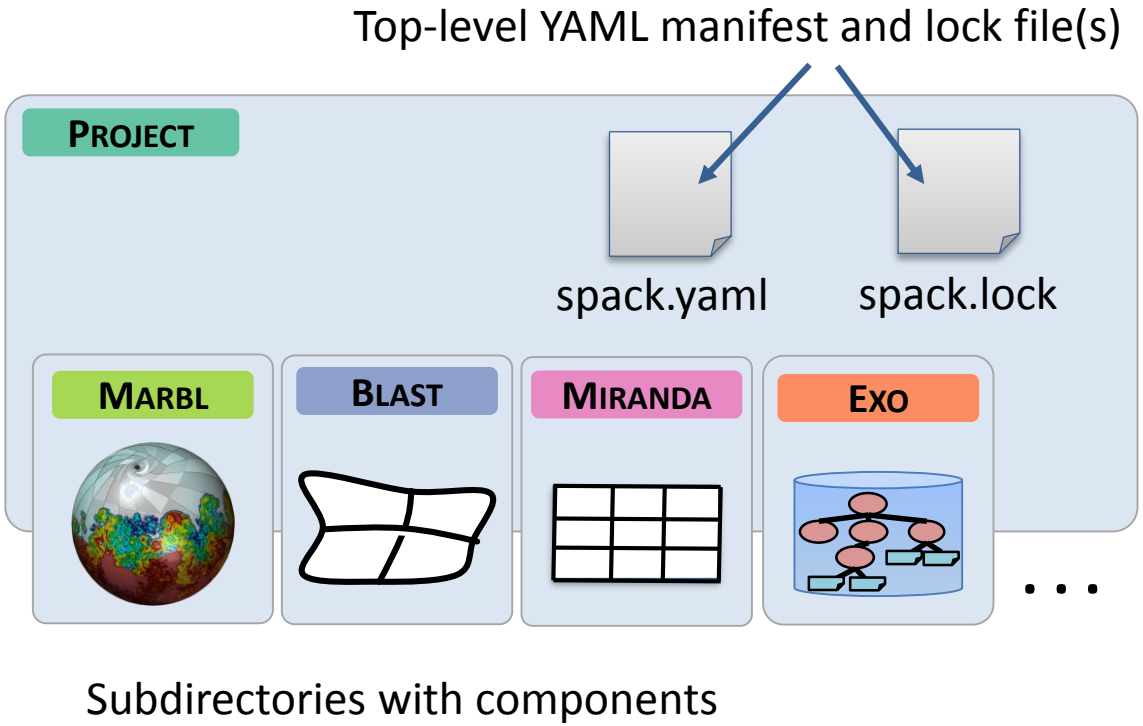
Spack workflow for developer environment

CLI workflow

```
$ git clone ssh://git@rzgitlab.llnl.gov:7999/mapp/mapp
$ cd mapp
$ spack env activate .
$ spack develop blast@develop
$ spack develop miranda@develop
$ srun –N 2 –n 16 --exclusive spack install
```

} Components to work on

spack can do multi-node builds



Subdirectories with components

E4S is ECP's curated, Spack-based software distribution

- **E4S is just a set of Spack packages**
  - **60+ packages (297 including dependencies)**
  - **Growing to include all of ST and more**

- Users can install E4S packages:
  - In their home directory
  - In a container

- Facilities can install E4S packages:
  - On bare metal
  - In a container

- Users and facilities can choose parts they want
  - `spack install` only the packages you want
  - Or just edit the list of packages (and configurations) you want in a `spack.yaml` file

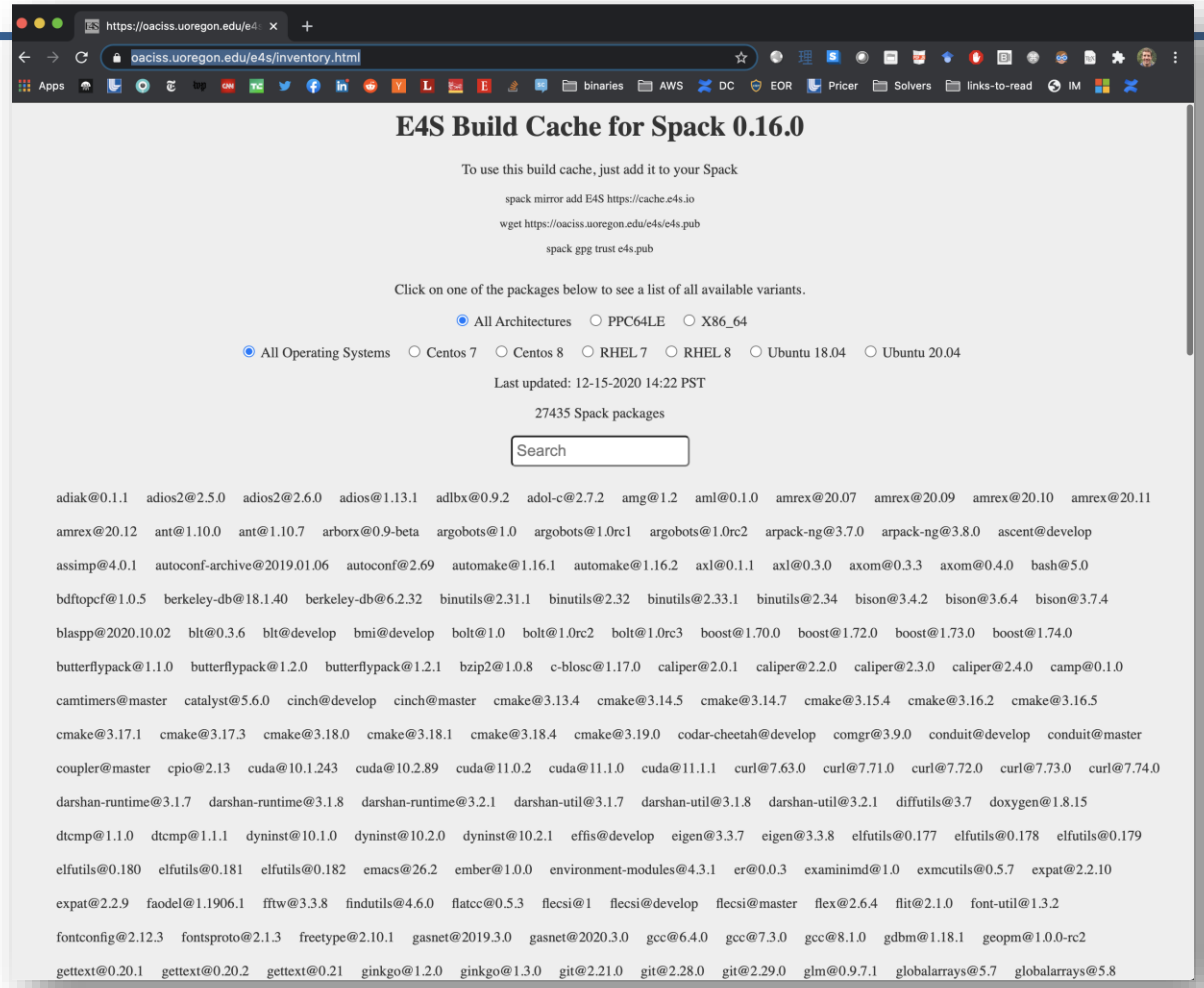
```
spack:
  specs:
  - openpmd-api
  - py-libensemble^python@3.7.3
  - hypre
  - mfem
  - trilinos@12.14.1+dtk+intrepid2+shards
  - sundials
  - strumpack
  - superlu-dist
  - superlu
  - tasmanian
  - mercury
  - hdf5
  - adios2
  - dyninst
  - pdt
  - tau
  - hpctoolkit
  - adios
  - darshan-runtime
  - darshan-util
  - veloc
  - scr
  - parallel-netcdf
  - qthreads
  - papyrus@develop
  - bolt
  - raja
  - upcxx
  - kokkos+openmp
  - openmpi
  - umpire
  - libquo
  - globalarrays
  - gotcha
  - caliper
  - papi
  - py-jupyterhub
  - zfp
  - sz
  - libnrm
  - rempi
  - ninja
  - kokkos-kernels
  #- turbine
  #- aml
  #- unifyfs
  #- flecsi+cinch
  #- petsc
  #- faodel
  packages:
    all:
      providers:
        mpi: [spectrum-mpi]
      target: [ppc64le]
    cuda:
      buildable: false
      version: [10.1.243]
      modules:
        cuda@10.1.243: cuda/10.1.243
    spectrum-mpi:
      buildable: false
      version:
      - 10.3.1.2
      modules:
        spectrum-mpi@10.3.1.2: spectrum-mpi/10.3.1.2-20200121
  config:
    misc_cache: $spack/cache
    build_stage: $spack/build-stage
    install_tree: $spack/$padding:512

  view: false
  concretization: separately
```

Actual E4S manifest (`spack.yaml`) for OLCF Ascent

More on E4S at https://e4s.io

E4S team has built a binary cache with over 50,000+ Spack binary packages

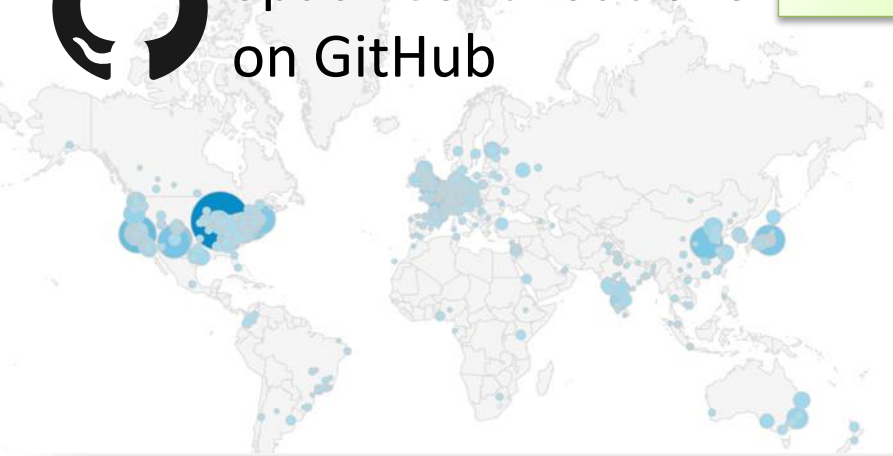
- Built for multiple OS's, architectures
- E4S team is working with ECP projects to accelerate their build pipelines
- Improved performance of cloud CI for one project by 10-100x
  - Previously, builds took too long for free cloud CI
  - Project can now iterate faster using Spack/E4S binaries
- We are rapidly building out binary build capabilities for Spack
  - Aim to have optimized binaries for most platforms in Frontier/El Capitan timeframe



https://oaciss.uoregon.edu/e4s/inventory.html

We have expanded our CI builds to trigger on pull requests, allowing us to do CI in the cloud for LLNL open source projects

Spack Contributions on GitHub



gitlab.spack.io

spack.yaml configuration

spack ci

x86_64 and **aarch64** pipelines in **AWS**

ppc64le pipelines at **U. Oregon**

Pipelines at **LLNL (new)**

ci/gitlab/gitlab.spack.io — Pipeline passed on GitLab

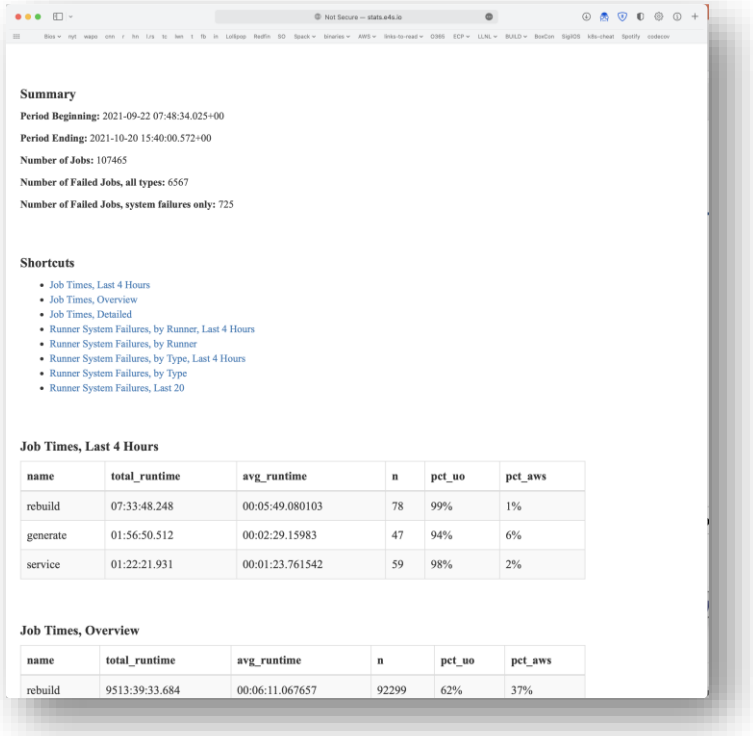
GitLab CI builds (changed) packages

- On every pull request
- On every release branch
- Different compilers (Intel soon!)

- **New security model supports untrusted contributions from forks**
  - Sandboxed build caches for test builds
  - Authoritative builds on mainline only after approved merge

Future CI directions focus on scalability and testing

- Scaling tests up to handle every PR has been very difficult
  - Driven by GitLab
  - Using Kubernetes builders
  - Using a cluster at U. Oregon
- Concretization of large environments was slowing turnaround
  - 55 min to concretize E4S environment (each spec separately)
  - Brought this down to 2.5 min this past weekend
- Amazon and E4S/UO team helping to pinpoint errors
- Once we have a stable, rolling release of spack develop branch, we'll make the build cache public
  - Rolling binaries for develop
  - Long-lived snapshots for each release



Summary

Period Beginning: 2021-09-22 07:48:34.025+00

Period Ending: 2021-10-20 15:40:00.572+00

Number of Jobs: 107465

Number of Failed Jobs, all types: 6567

Number of Failed Jobs, system failures only: 725

Shortcuts

- Job Times, Last 4 Hours
- Job Times, Overview
- Job Times, Detailed
- Runner System Failures, by Runner, Last 4 Hours
- Runner System Failures, by Runner
- Runner System Failures, by Type, Last 4 Hours
- Runner System Failures, by Type
- Runner System Failures, Last 20

Job Times, Last 4 Hours

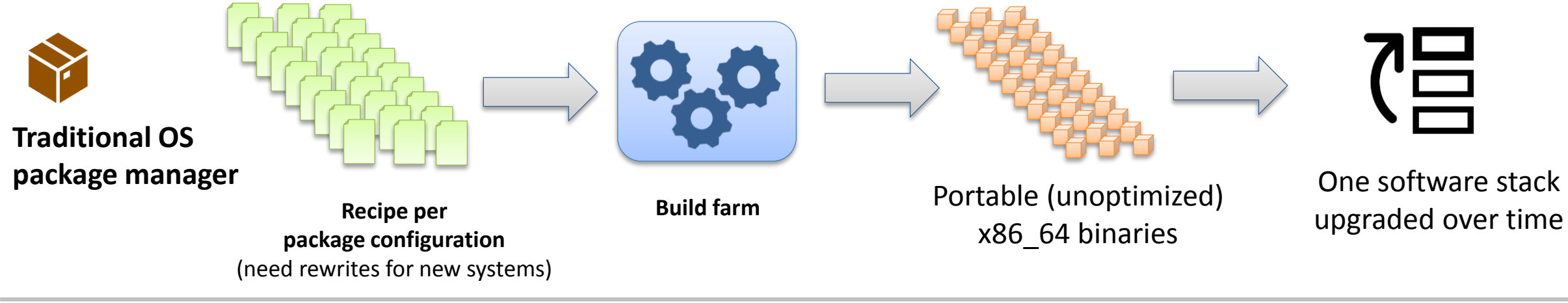
name	total_runtime	avg_runtime	n	pct_uo	pct_aws
rebuild	07:33:48.248	00:05:49.080103	78	99%	1%
generate	01:56:50.512	00:02:29.15983	47	94%	6%
service	01:22:21.931	00:01:23.761542	59	98%	2%

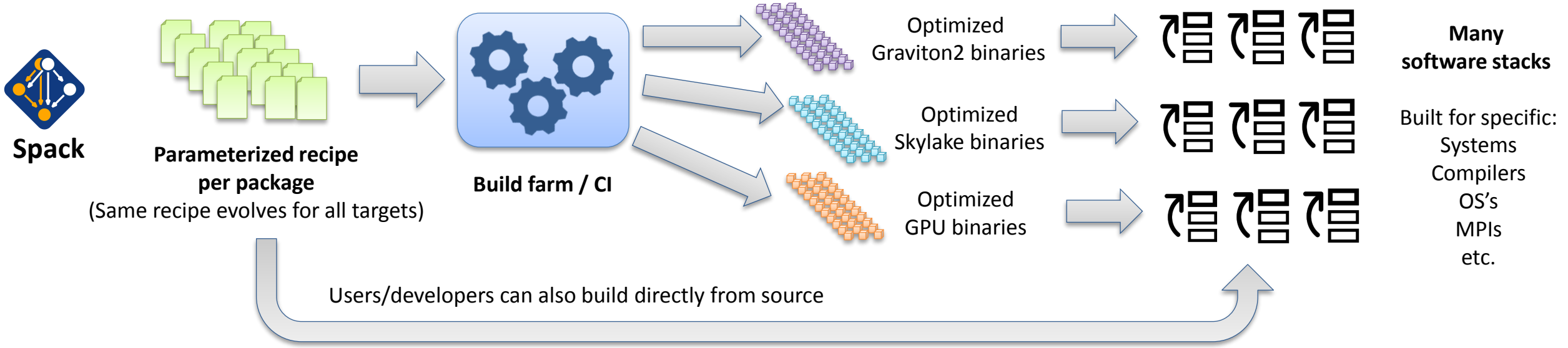
Job Times, Overview

name	total_runtime	avg_runtime	n	pct_uo	pct_aws
rebuild	9513:39:33.684	00:06:11.067657	92299	62%	37%

http://stats.e4s.io

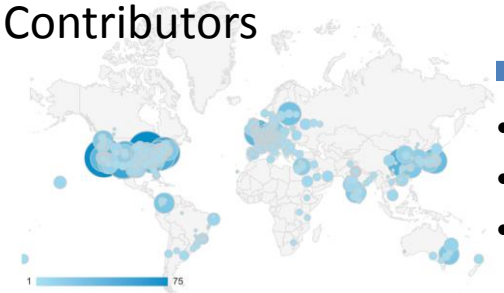
Spack's model lowers the maintenance burden of optimized software stacks





High level view of a Spack package build

Contributors



- new versions
- new dependencies
- new constraints

package.py repository

concretizer

spack developers

default config packages.yaml

admins, users

yaml l preferences config packages.yaml

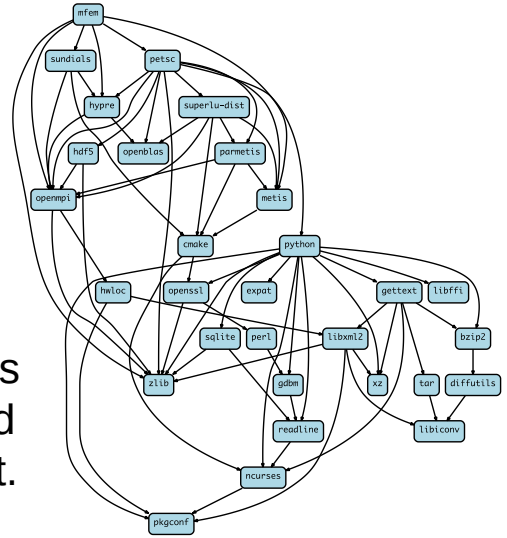
users

yaml cal environment config spack.yaml

users

Command line constraints

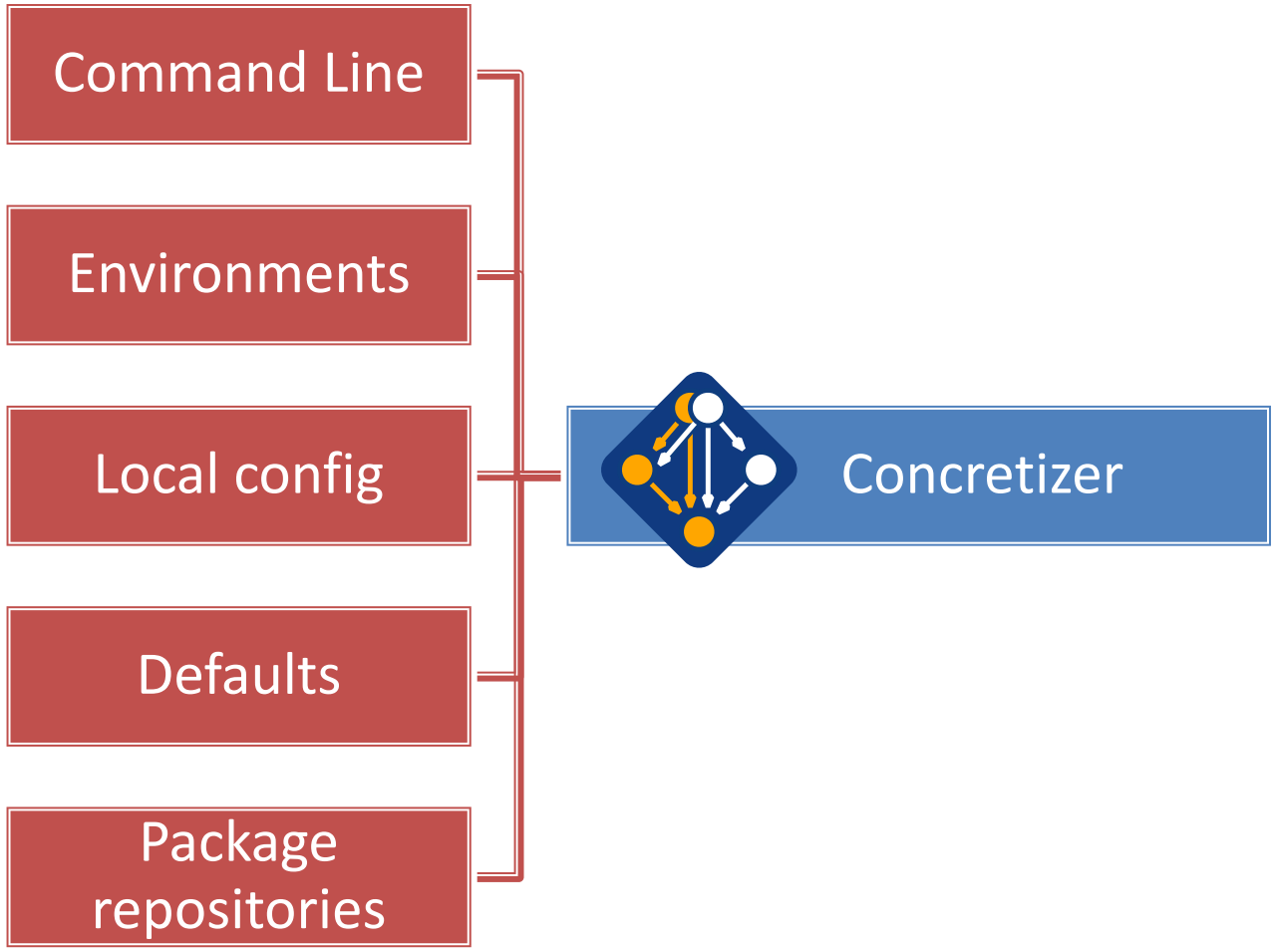
spack install hdf5@1.12.0 +debug



Concrete spec is fully constrained and can be built.

Spack's concretizer has gotten pretty complicated

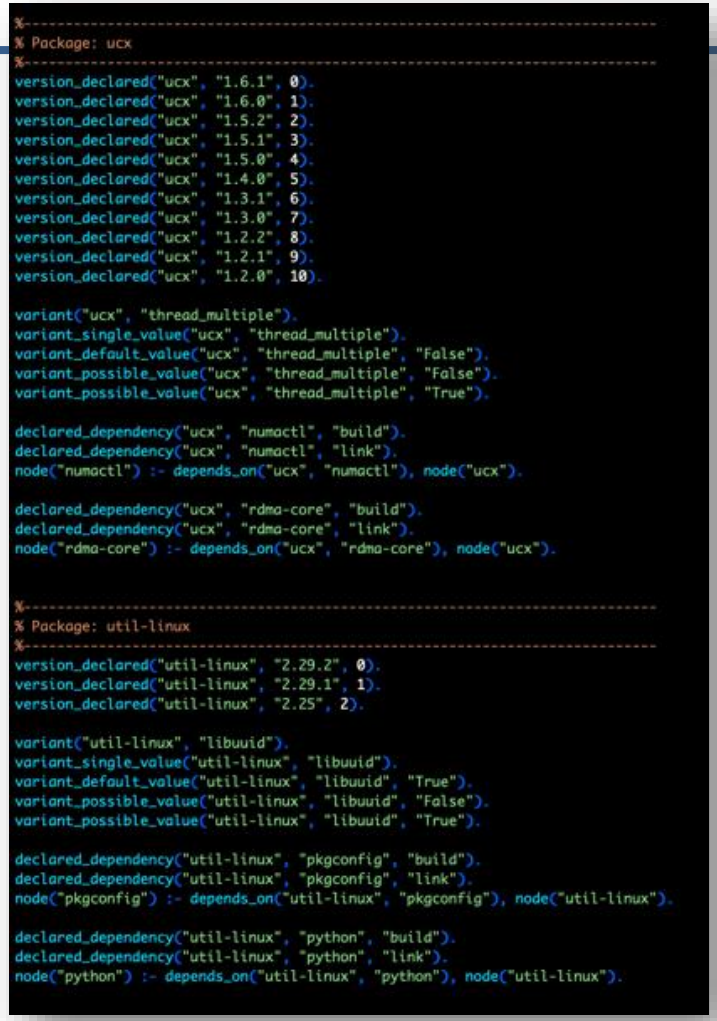
Sources for constraints



- Current implementation is ad-hoc:
  - Traverse the DAG
  - Evaluate conditions, add dependencies
  - Fill in defaults from many sources
  - Repeat until DAG doesn't change
- Issues:
  - Limited support for backtracking causes some graphs to resolve incorrectly
  - Some constraints are strictly ordered
  - Lots of conditional complexity
- Design doesn't scale to all the criteria
  - Hard to add new features/logic
  - Can be slow

The new concretizer will be default in 0.17

- 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
       - 20,000 – 30,000 facts is typical – includes dependencies, options, etc.
    2. Small logic program (~800 lines), including 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 current concretizer can't
  - Backtracking is a huge win – many issues resolved
  - Currently requires user to install clingo with Spack
  - Solver will be automatically installed from public binaries in 0.17.0



Some facts for the HDF5 package

The new concretizer enables significant simplifications to packages, particularly complex constraints in SDKs

- Dependencies and other constraints within SDKs could get very messy
- The new concretizer removes the need for some of the more painful constructs
- Also allows for new constructs, like specializing dependencies
  - When conditions are now much more general
  - Can be solved together with other constraints.

In some cases we needed cross-products of dependency options:

Before

```
depends_on('foo+A+B', when='+a+b')
depends_on('foo+A~B', when='+a~b')
depends_on('foo~A+B', when='~a+b')
depends_on('foo~A~B', when='~a~b')
```

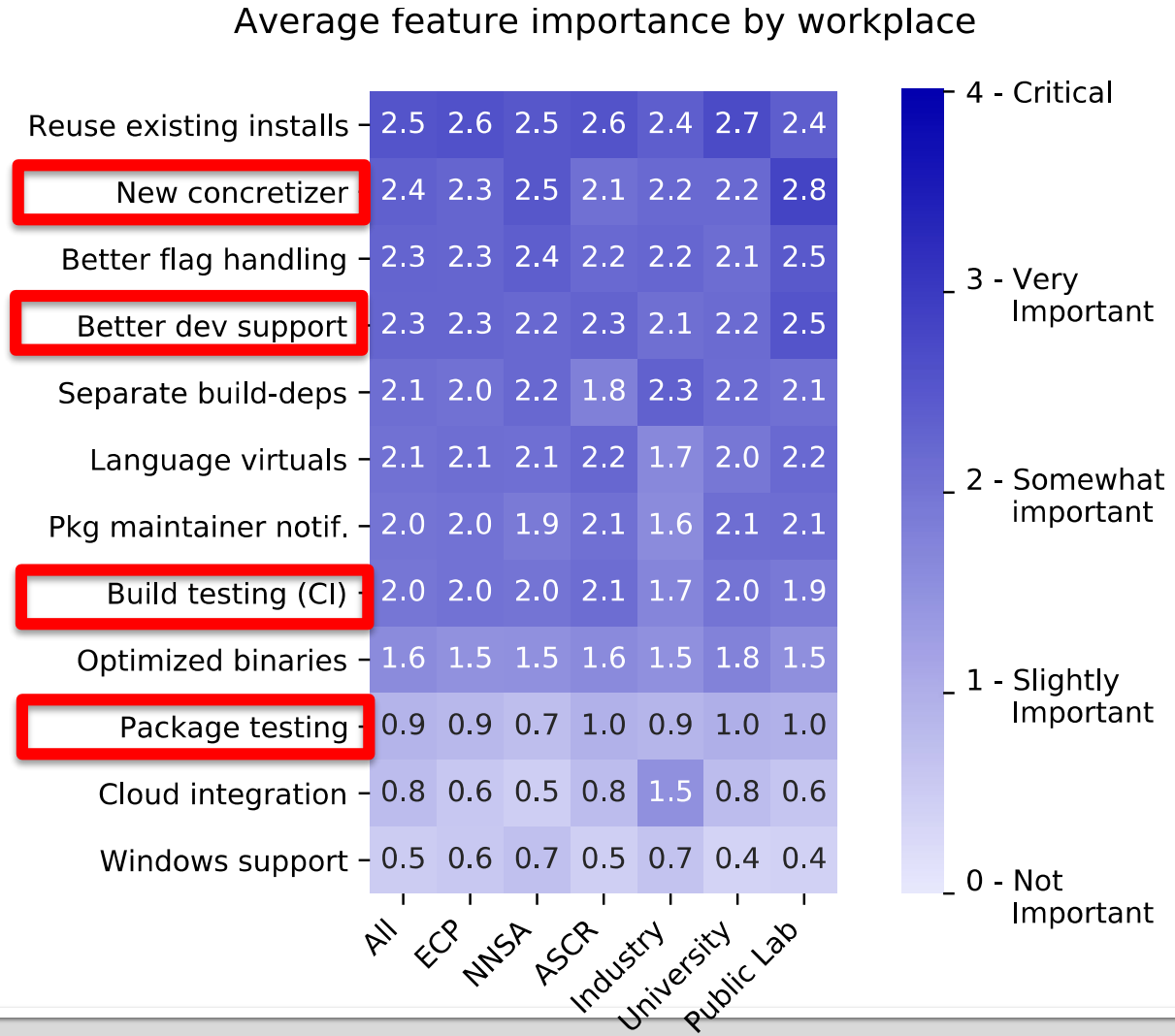
After

```
depends_on('foo')
depends_on('foo+A', when='+a')
depends_on('foo+B', when='+b')
```

Specializing a virtual did not previously work:

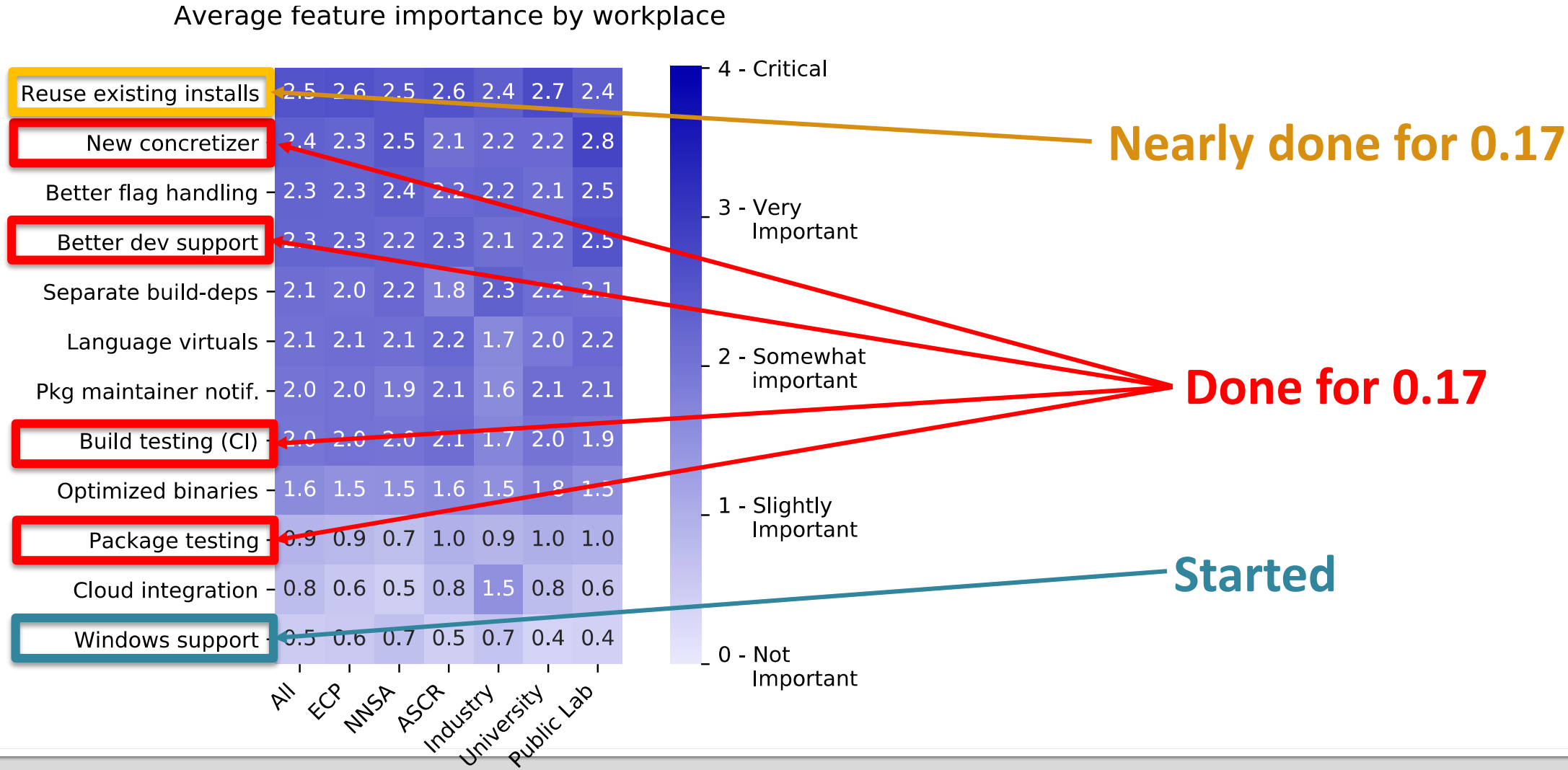
```
depends_on('blas')
depends_on(
  'openblas threads=openmp', when='^openblas'
)
```

Four of the top six most wanted features in Spack were tied to the new concretizer



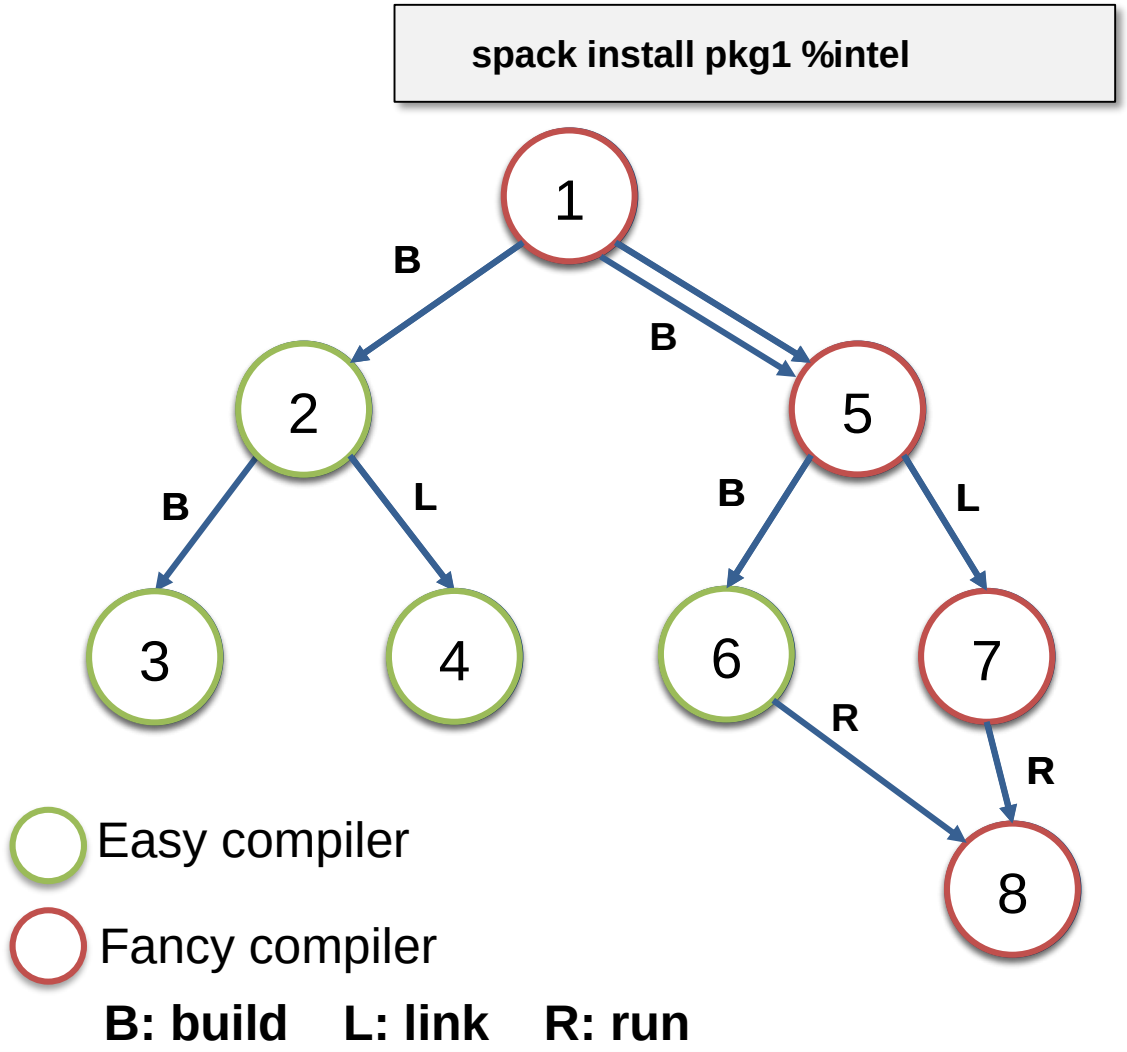
- Complexity of packages in Spack is increasing
  - many more package solves require backtracking than a year ago
  - Many variants, conditional dependencies, special compiler requirements
- More aggressive reuse of existing installs requires better dependency resolution
  - Need to be able to analyze how to configure the build to work with installed packages
- Separate resolution of build dependencies also requires a more sophisticated solver
  - Makes the solve even more combinatorial
  - Needed to support mixed compilers, version conflicts between different package's build requirements

Four of the top six most wanted features in Spack are tied to the new concretizer



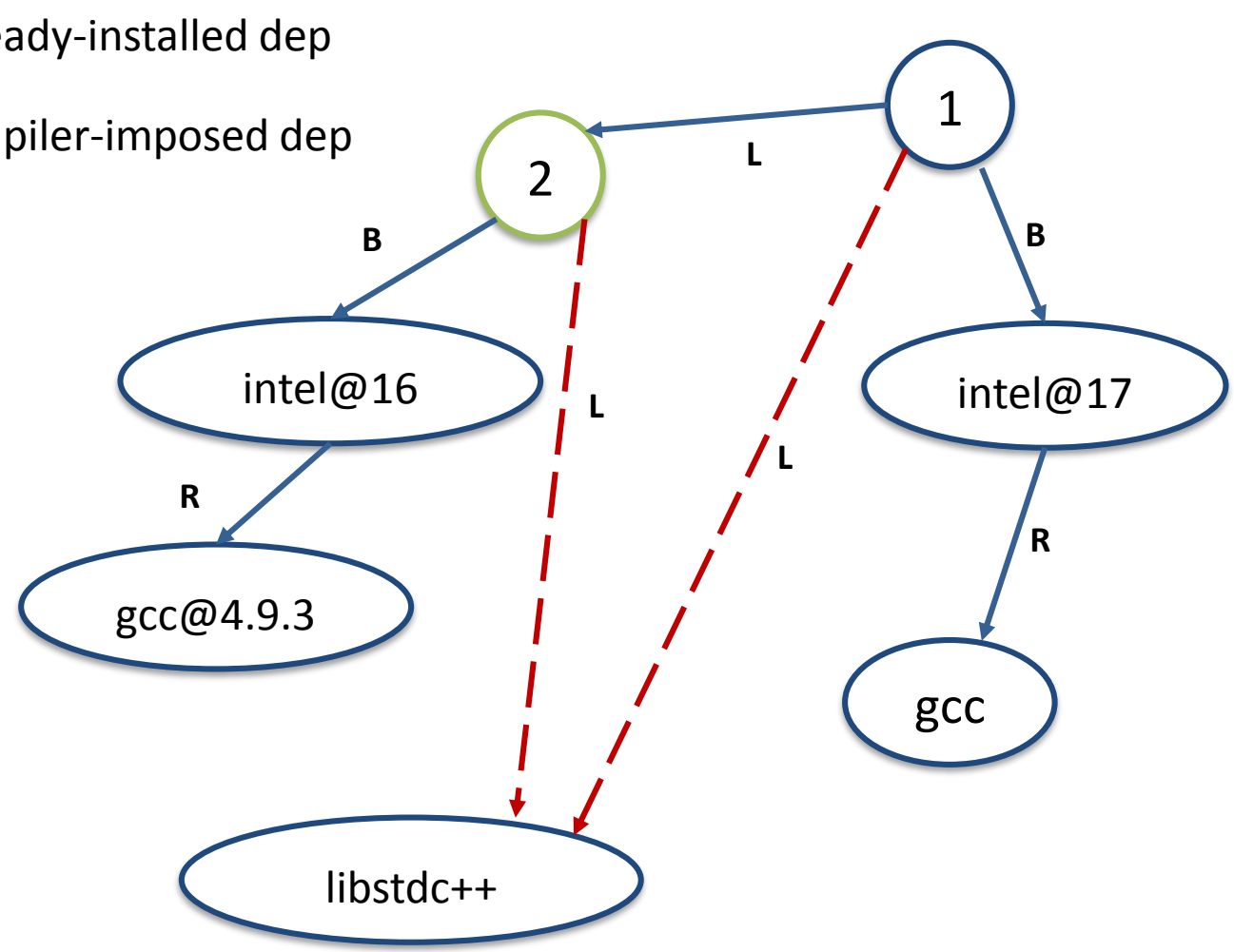
Separate concretization of build dependencies

- We want to:
  - Build build dependencies with the "easy" compilers
  - Build rest of DAG (the link/run dependencies) with the fancy compiler
- This required significant concretizer modifications
- Gets into issues like bootstrapping



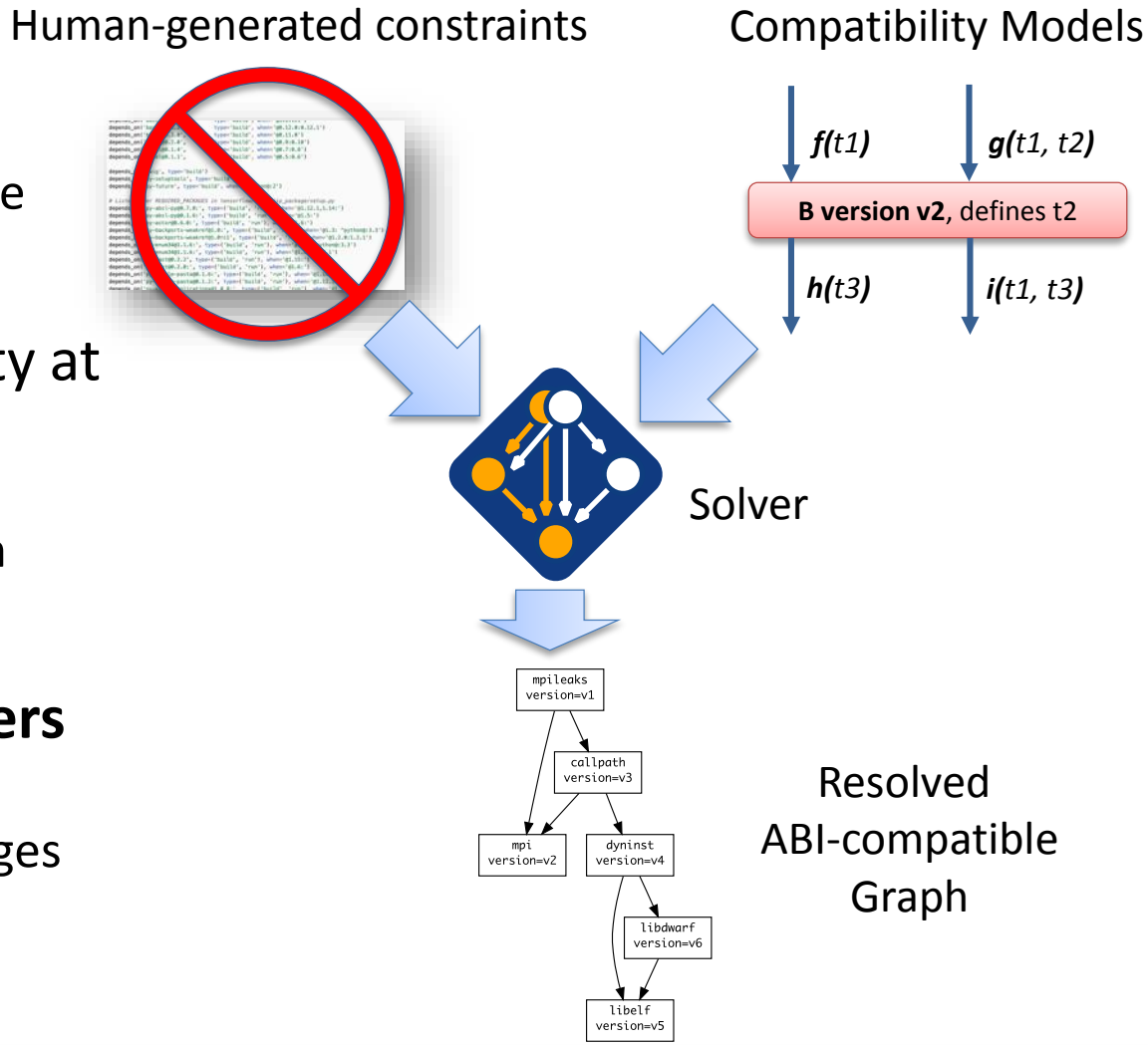
New compiler dependency model

- Now consistency is enforced via link dependencies from 1 and 2
- If the libstdc++ versions from 1 and 2 don't match, then this won't resolve
- If they do, then we know that the C++ libs are compatible and can build this, even with the old dependency.
- We currently use some heuristics to enforce
  - Moving to SAT makes a lot of sense, again
- Will enable us to support langauge virtuals



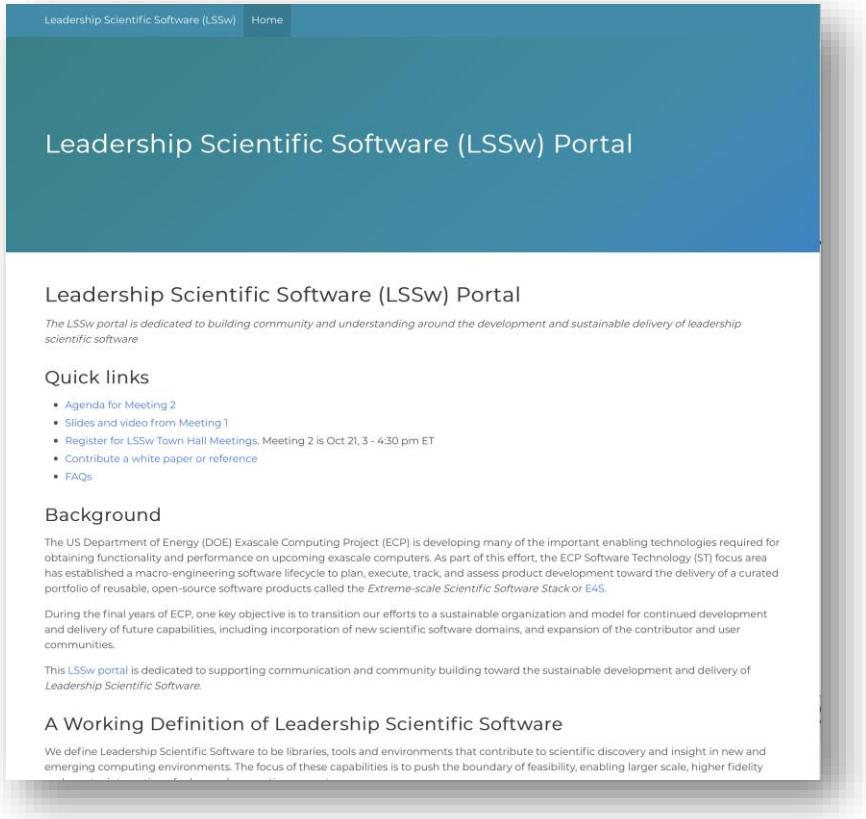
Ongoing research:
BUILD is a 3-year research project, started at LLNL in 2020

- Basic premise: humans can't generate all the compatibility constraints
  - Version ranges, conflicts, in Spack packages not precise
  - rely on maintainers to get right.
- BUILD aims to understand software compatibility at the binary level
  - Develop ABI compatibility models
  - Enable *automatic* and ABI-compatible reuse of system binaries, foreign binary packages
- **WIP: better dependency solvers can enable users to solve *around* system dependencies**
  - find "closest" match to a prior build, using new packages
  - Reproduce a prior build with new requirements



After ECP

- We are looking at longer-term sustainability directions after ECP
- Opportunities (everything is in flux at this point):
  - ASCR Workshop on the Science of Scientific-Software Development and Use
    - just came out
  - Leadership Scientific Software Meeting series https://lssw.io
- We want to be part of any post-ECP sustainability effort!
  - Likely some type of work in conjunction with E4S



Spack's long-term strategy is based around broad adoption and collaboration

- **Not sustainable without a community**
  - Broad adoption incentivizes contributors
  - Cloud resources and automation absolutely necessary
- **Spack preserves build knowledge in a cross-platform, reusable way**
  - Minimize rewriting recipes when porting
- **CI ensures builds continue to work as packages evolve**
  - Keep packages flexible but verify key configurations
- **Any suggestions on sustainability models would be appreciated!**