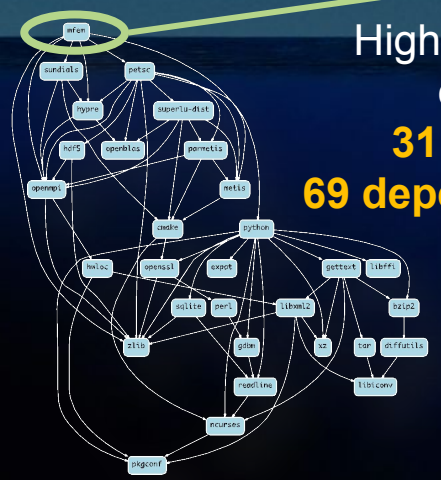
Intro to Spack

TREX Hackathon
November 12, 2021

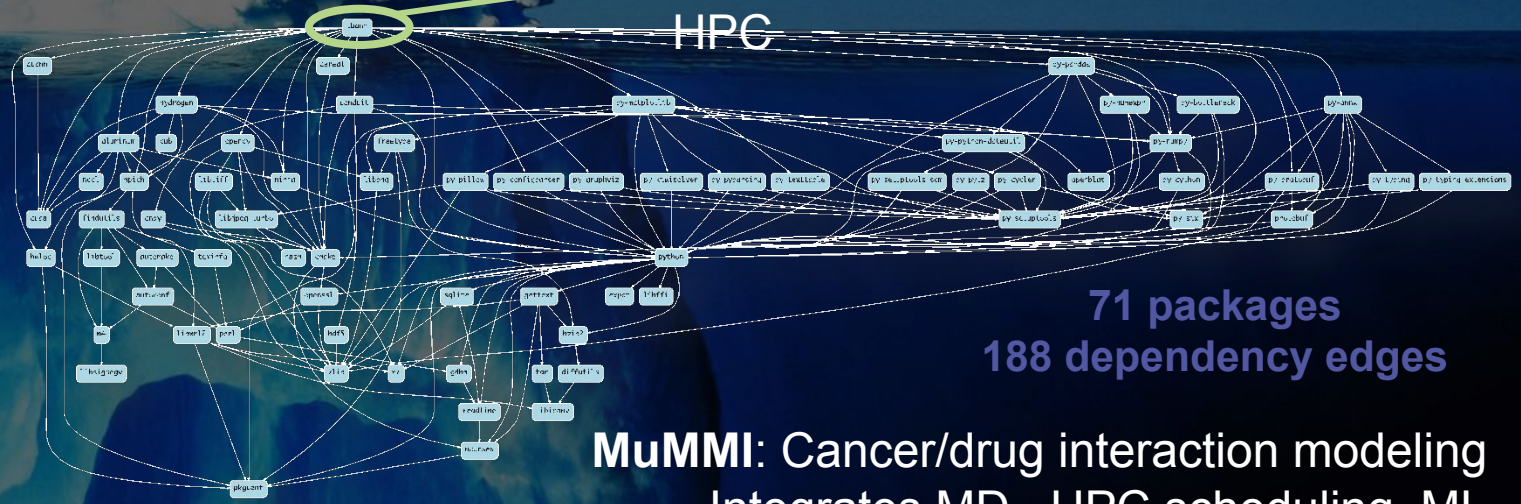
Todd Gamblin

Advanced Technology Office
Lawrence Livermore National Laboratory

HPC simulations rely on icebergs of dependency libraries



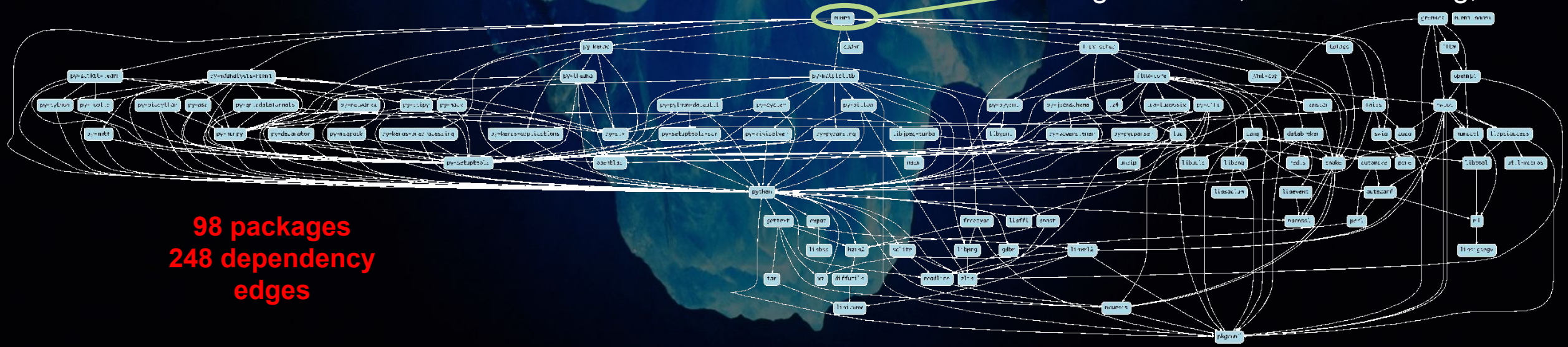
MFEM:
Higher-order finite elements
31 packages,
69 dependency edges



LBANN: Neural Nets for HPC

71 packages
188 dependency edges

MuMMI: Cancer/drug interaction modeling
Integrates MD , HPC scheduling, ML



98 packages
248 dependency edges

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

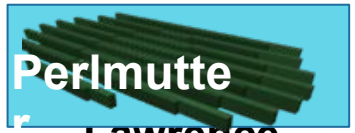
Some Supercomputers

Current

Oak Ridge National Lab
Power9 / NVIDIA

RIKEN
Fujitsu/ARM a64fx

Upcoming



Lawrence Berkeley National Lab
AMD Zen / NVIDIA

Argonne National Lab
Intel Xeon / Xe

Oak Ridge National Lab
AMD Zen / Radeon

Lawrence Livermore National Lab
AMD Zen / Radeon

Spack enables Software distribution for HPC

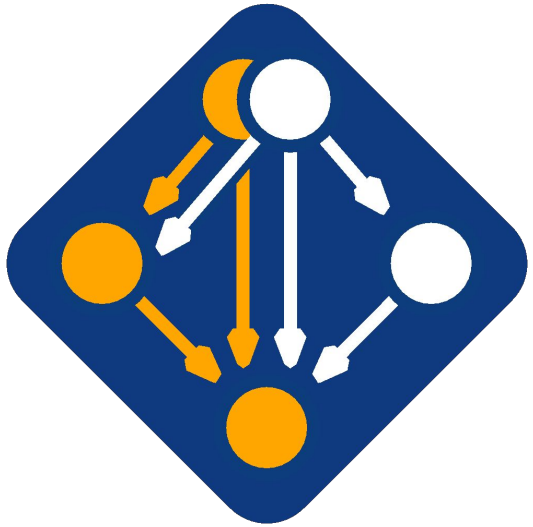
- Spack automates the build and installation of scientific software
- Packages are *parameterized,* so that users can easily tweak and tune configuration

No installation required: clone and go

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

Simple syntax enables complex installs

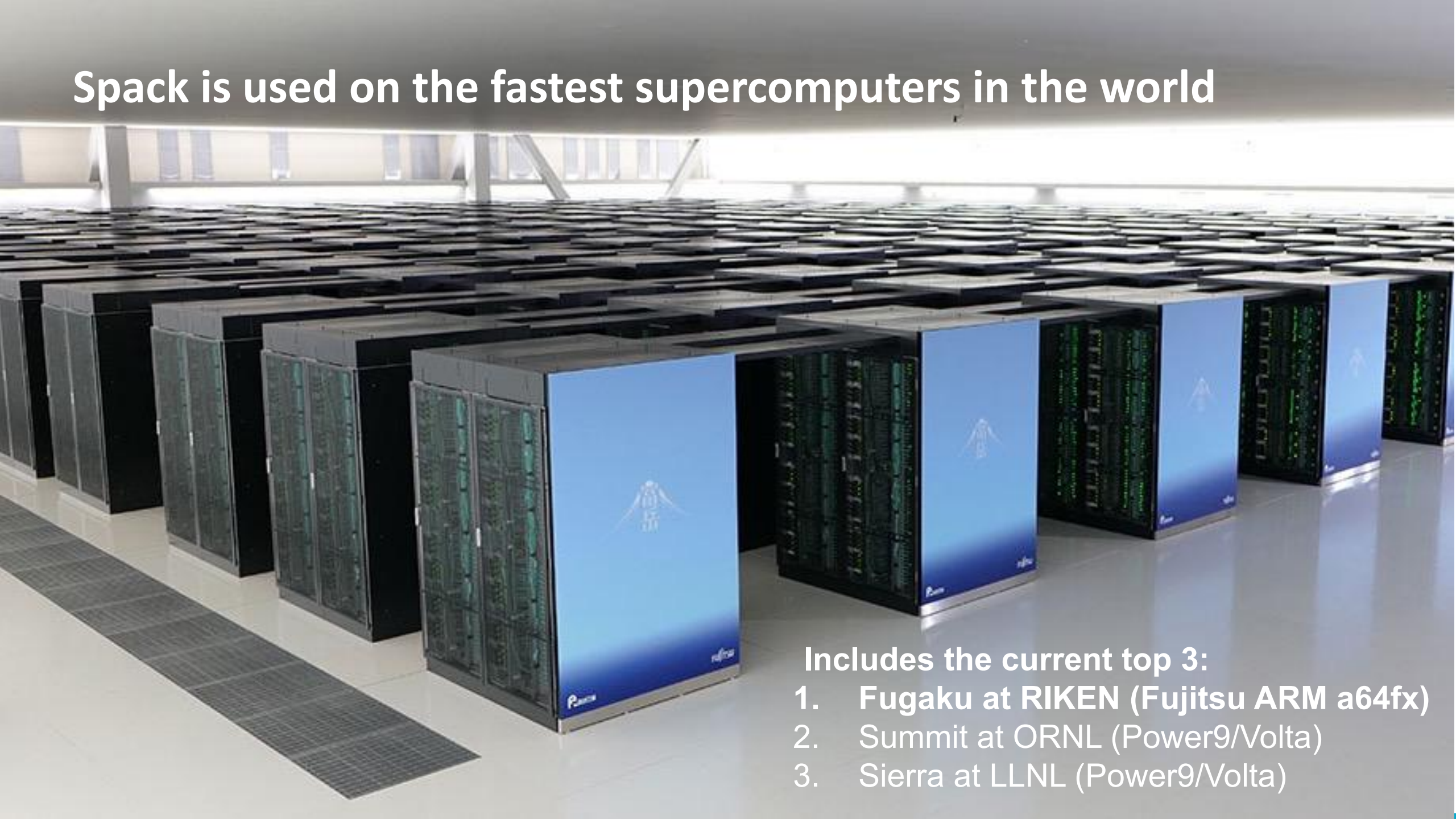
```
$ spack install hdf5@1.10.5
$ spack install hdf5@1.10.5 %clang@6.0
$ spack install hdf5@1.10.5 +threadssafe
$ spack install hdf5@1.10.5 cppflags="-O3 –g3"
$ spack install hdf5@1.10.5 target=haswell
$ spack install hdf5@1.10.5 +mpi ^mpich@3.2
```



github.com/spack/spack

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

Spack is used on the fastest supercomputers in the world



Includes the current top 3:

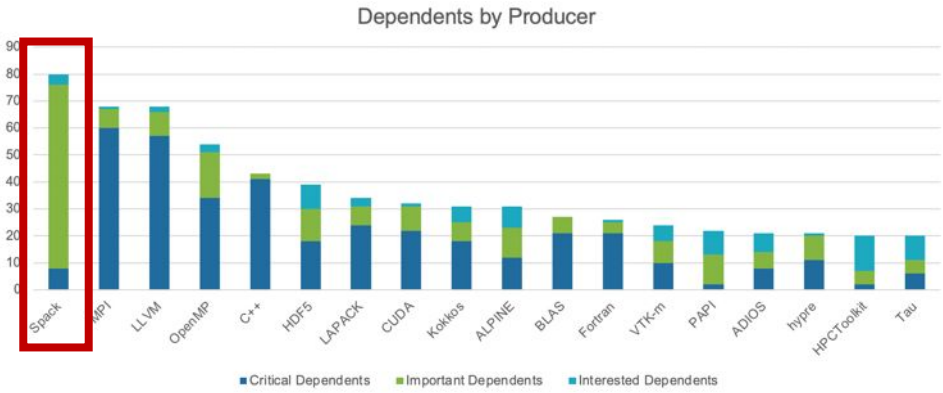
1. **Fugaku at RIKEN (Fujitsu ARM a64fx)**
2. Summit at ORNL (Power9/Volta)
3. Sierra at LLNL (Power9/Volta)

Spack is critical for ECP's mission to create a robust, capable exascale software ecosystem.

EXASCALE COMPUTING PROJECT

https://e4s.io

- Spack will be used to build software 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 will be integral to upcoming ECP testing efforts.

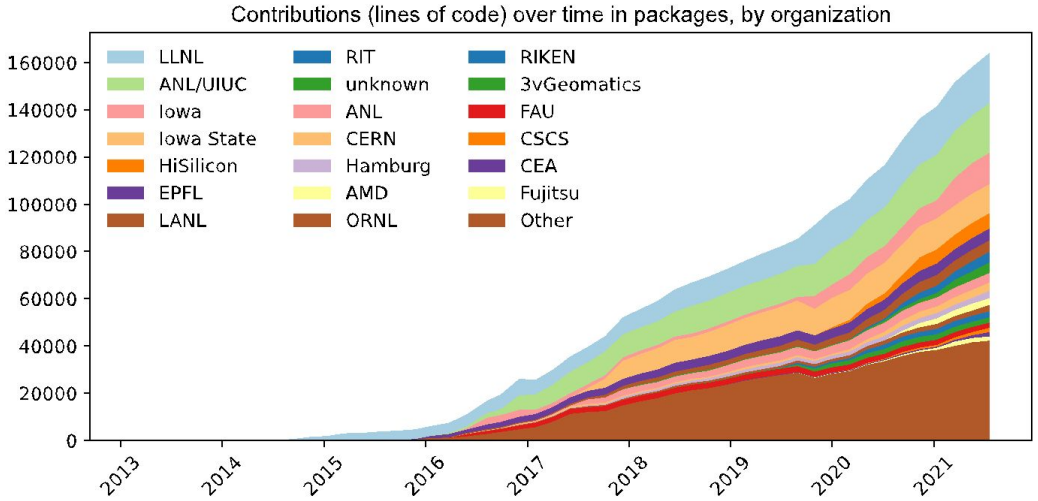


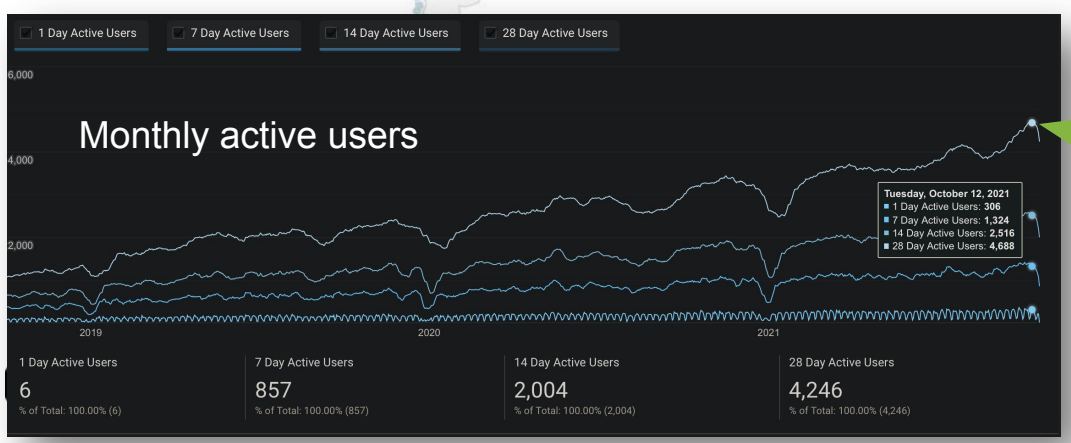
Spack is the most depended-upon project in ECP

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

One month of Spack development is pretty busy!

October 12, 2021 – November 12, 2021

Period: 1 month

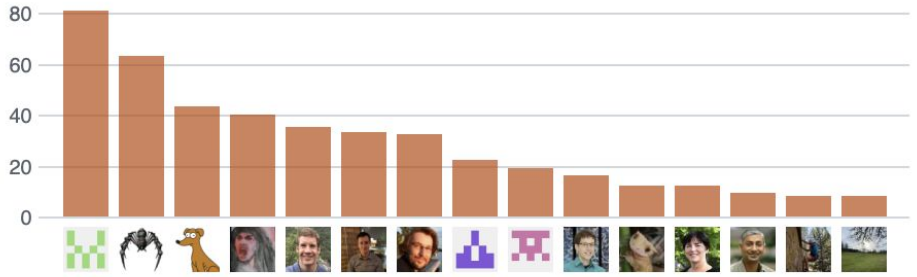
Overview

671 Active Pull Requests

145 Active Issues

536	135	75	70
Merged Pull Requests	Open Pull Requests	Closed Issues	New Issues

Excluding merges, **173 authors** have pushed **571 commits** to develop and **634 commits** to all branches. On develop, **703 files** have changed and there have been **20,730 additions** and **3,807 deletions**.



1 Release published by **1** person

v0.17.0
published 7 days ago

536 Pull requests merged by **151** people

Spack provides a *spec* syntax to describe customized installations

```
$ 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=zen2          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
(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 DSL allows *declarative* specification of complex constraints

CudaPackage: a mix-in for packages that

```
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

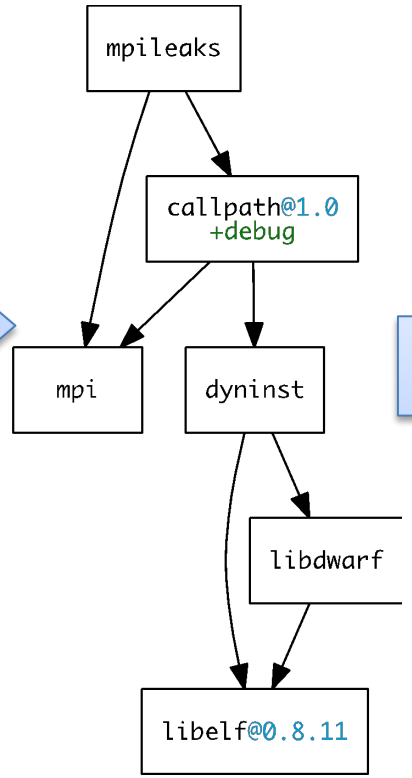
There is a lot of expressivity in this DSL.

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

mpileaks ^callpath@1.0+debug ^libelf@0.8.11

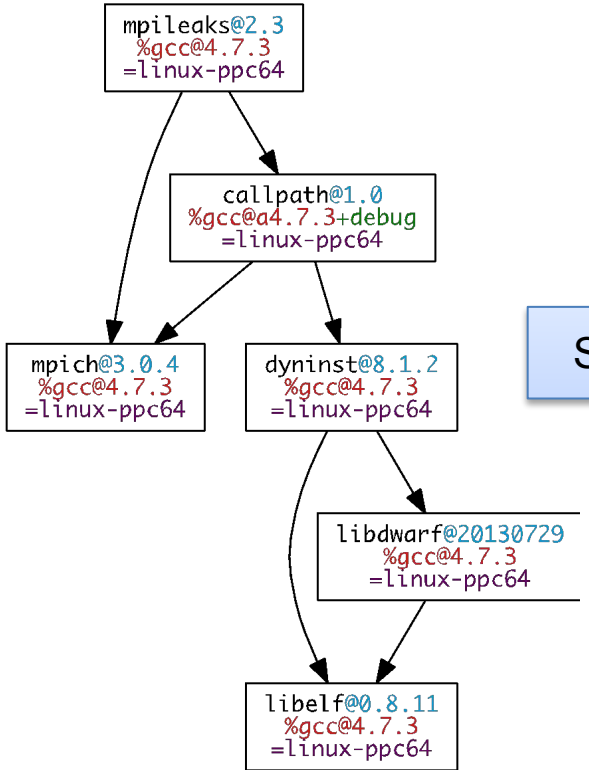
User input: *abstract* spec with some constraints

Normalize



Abstract, normalized spec with some dependencies.

Concretize



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

Store

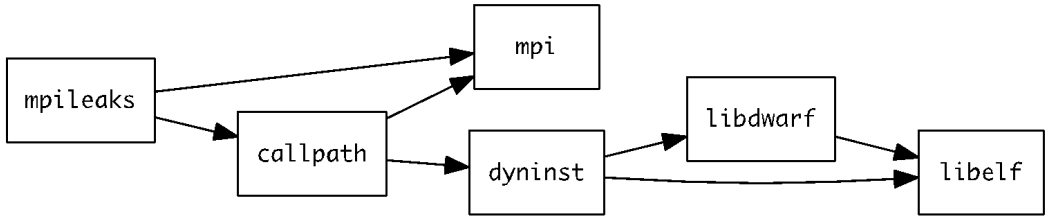
spec.yaml

```
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

Spack handles combinatorial software complexity

Dependency DAG



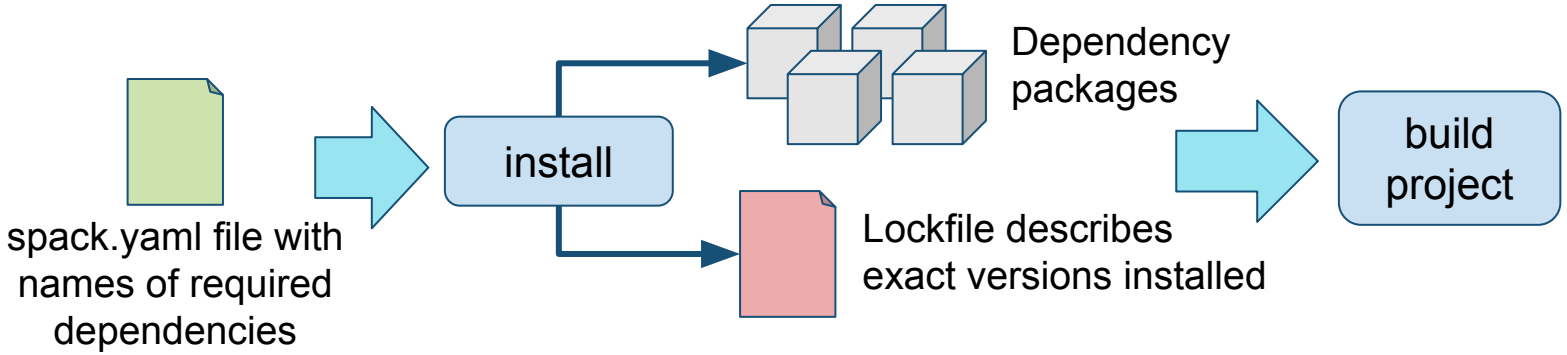
Installation Layout

Hash

```
opt
└── spack
    ├── darwin-mojave-skylake
    │   └── clang-10.0.0-apple
    │       ├── bzip2-1.0.8-hc4sm4vuzpm4znmvrfzri4ow2mkphe2e
    │       ├── python-3.7.6-daqqpssxb6qbfrztsezkmhus3xoflbsy
    │       ├── sqlite-3.30.1-u64v26igxvxyn23hysmklfums6tgjv5r
    │       ├── xz-5.2.4-u5eawkvaoc7vonabe6nndkcfwuv233cj
    │       └── zlib-1.2.11-x46q4wm46ay4pltriijbgizxjrhbaka6
    └── darwin-mojave-x86_64
        └── clang-10.0.0-apple
            └── coreutils-8.29-pl2kcytejqcys5dzecfrtjqxfdssvnob
```

- 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*

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 also be used to maintain configuration together with Spack packages.
  - E.g., versioning your own local software stack with consistent compilers/MPI implementations
  - Allows developers and site support engineers to easily version Spack configurations in a repository

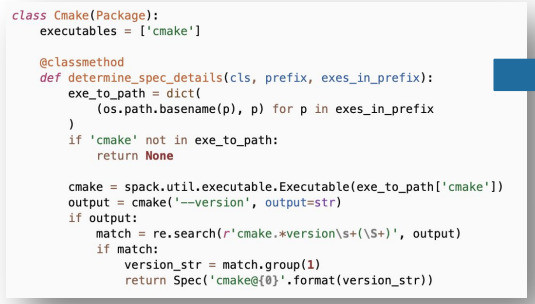
Simple spack.yaml file

```yaml
spack:
  # include external configuration
  include:
  - ../special-config-directory/
  - ./config-file.yaml

  # add package specs to the `specs` list
  specs:
  - hdf5
  - libelf
  - openmpi
```

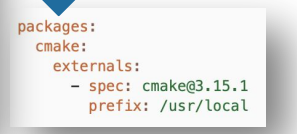
Concrete spack.lock file (generated)

```
{
 "concrete_specs": {
   "6s63so2kstp3zyvjezglndmavy6l3
     "hdf5": {
         "version": "1.10.5",
         "arch": {
             "platform": "darwin"
             "platform_os": "moja
             "target": "x86_64"
         },
```

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



package.py



spack.yaml configuration

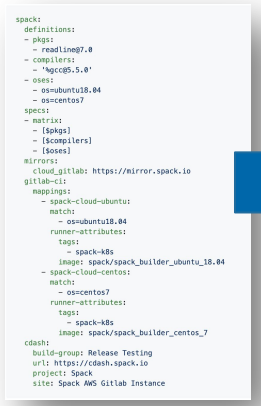
spack external find

Automatically find and configure external packages on the system

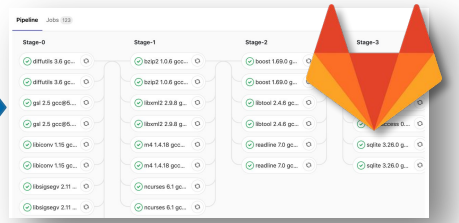
spack test

Packages know how to run their own test suites

package.py



spack.yaml



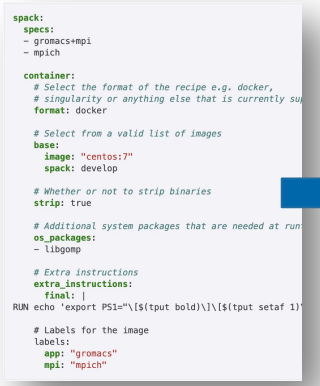
.gitlab-ci.yml CI pipeline

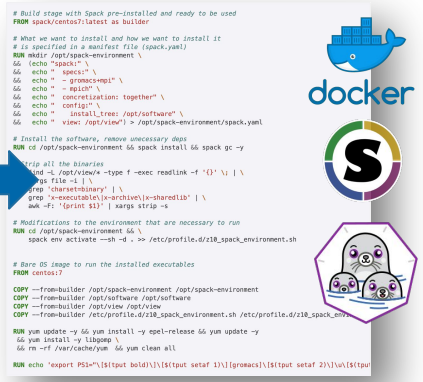
spack ci

Automatically generate parallel build pipelines
(more on this later)

spack containerize

Turn environments into container build recipes





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

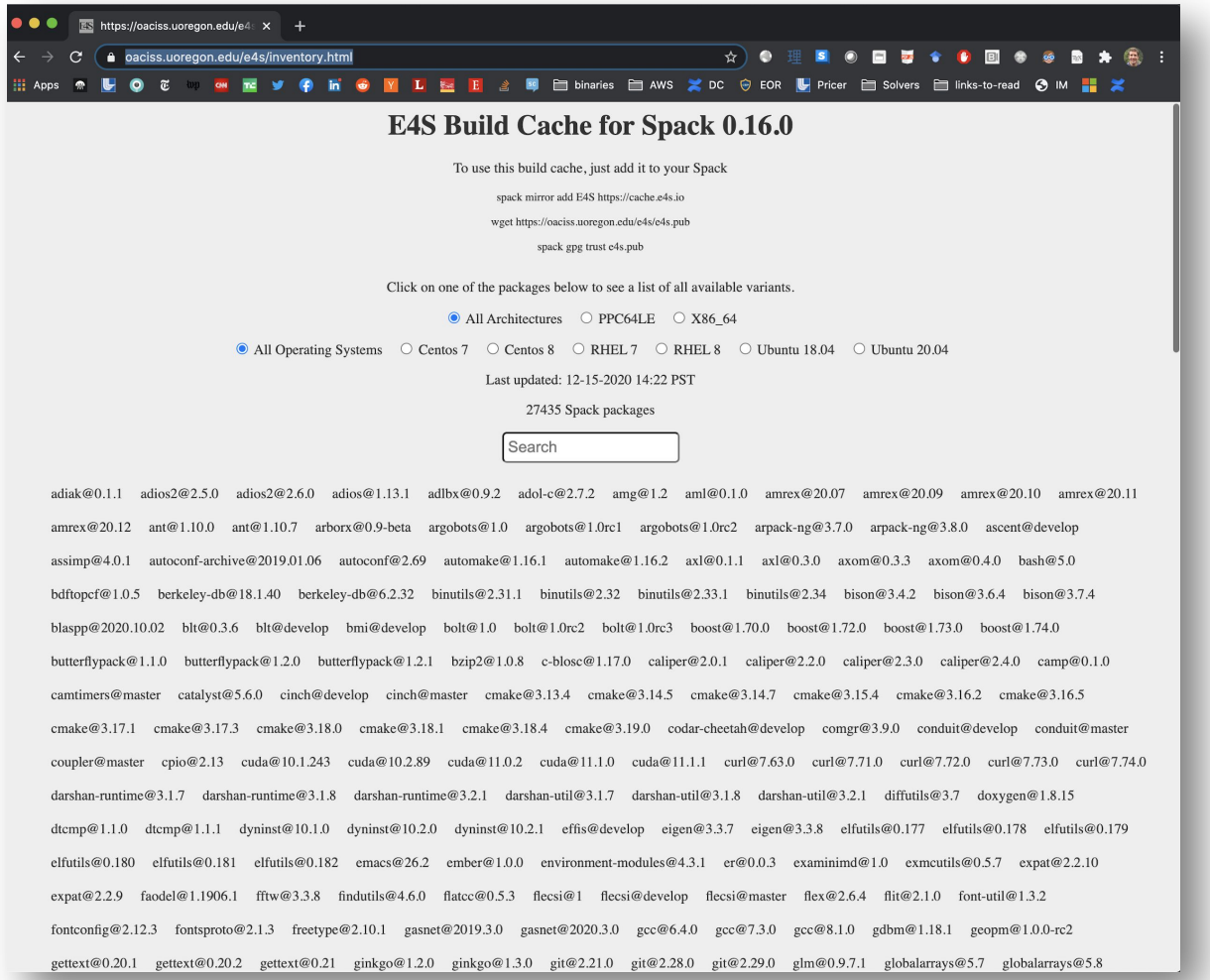
```
spack:
  specs:
  - openpmd-api
  - py-libensemble^python@3.7.3
  - hypre
  - mfem
  - trilinos@12.14.1+dtk+intrepid2+shards
  - sundials
  - strumpack
  - superlu-dist
  - superlu
  - tasmanian
  - mercury
  - hdf5
  - adios2
  - dyninst
  - pdt
  - tau
  - hpctoolkit
  - adios
  - darshan-runtime
  - 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

spack develop lets developers work on many packages at once

- Developer features so far have focused on single packages (spack dev-build, etc.)
- New spack develop feature enables development environments
  - Work on a code
  - Develop multiple packages from its dependencies
  - Easily rebuild with changes
- Builds on spack envirnoments
  - Required changes to the installation model for dev packages
  - dev packages don't change paths with configuration changes
  - Allows devs to iterate on builds quickly

```
$ 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
```

The AML team has used Spack environments to accelerate their workflow

c/o Robert Blake

- **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 customizable environment in ~20 minutes.**
    - Change python version, PyTorch version on the fly
    - Leverage binary caches to avoid redundant builds.

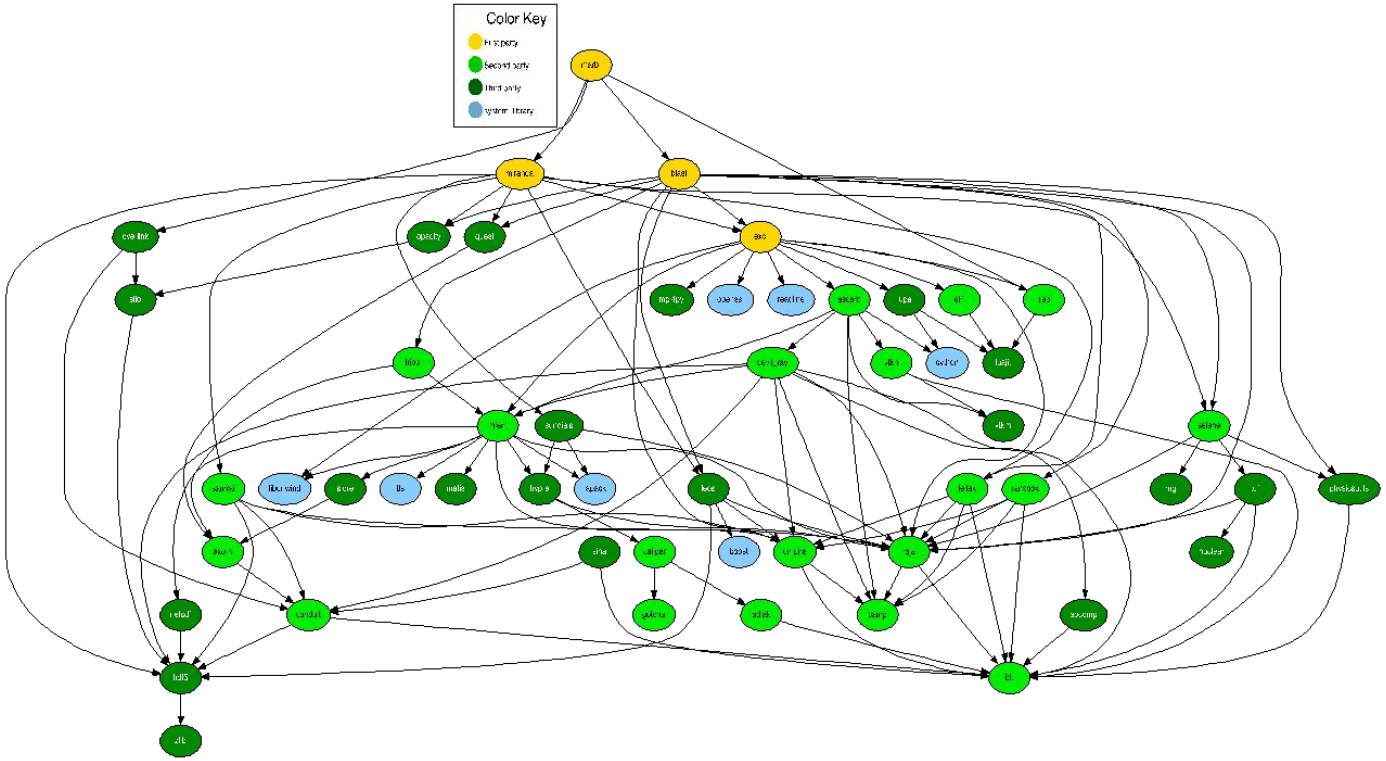
```
spack:
  specs:
  - py-horovod
  - py-torch
  - python
  - py-h5py

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

We wanted to translate this workflow to larger codes.

We have recently introduced some new features to support the development model of MARBL, an LLNL multi-physics code

- 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 doesn't (likely won't) use mono-repos
  - Issues:
    - Managing permissions
    - Code timescales
    - Independence of teams
- MARBL built MBS: a better poly-repo approach



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.

Using git versioning, we've been able to support MARBL's developer workflow

- First section is familiar
  - List of packages with hashes
- spack.yaml ties the modular MARBL code together:
  - hashes
  - parts of exo/build directory
- Some differences:
  - Packages in Spack are configurable
  - Can set per-package options
  - Compiler options, flags are configurable in Spack environments
- 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

Current MARBL spack.yaml

```
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

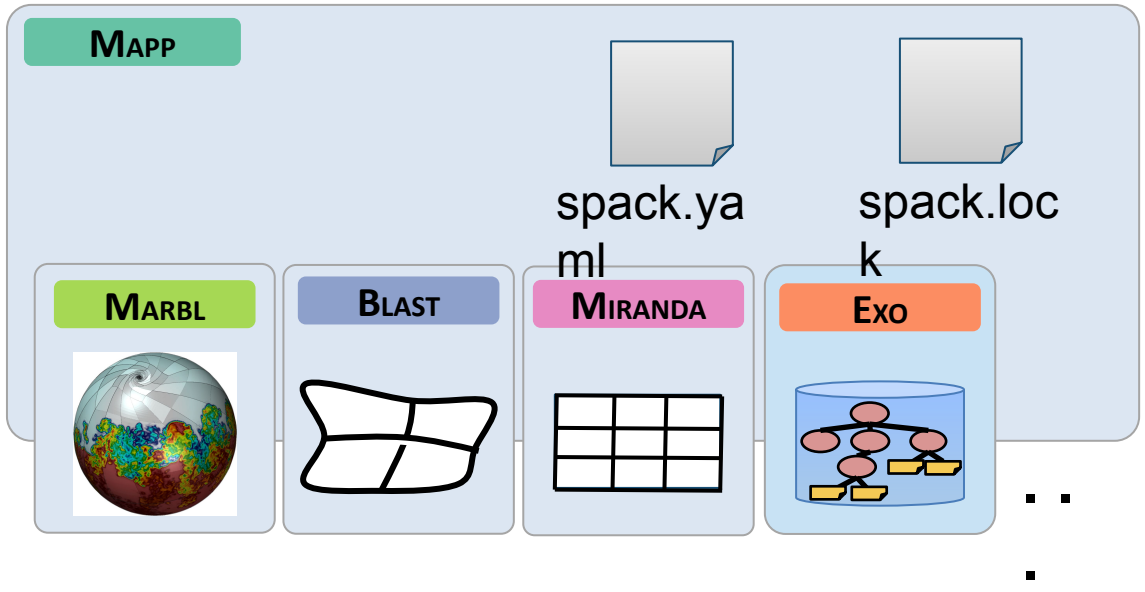
Spack workflow for developer environment

Spac
k

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

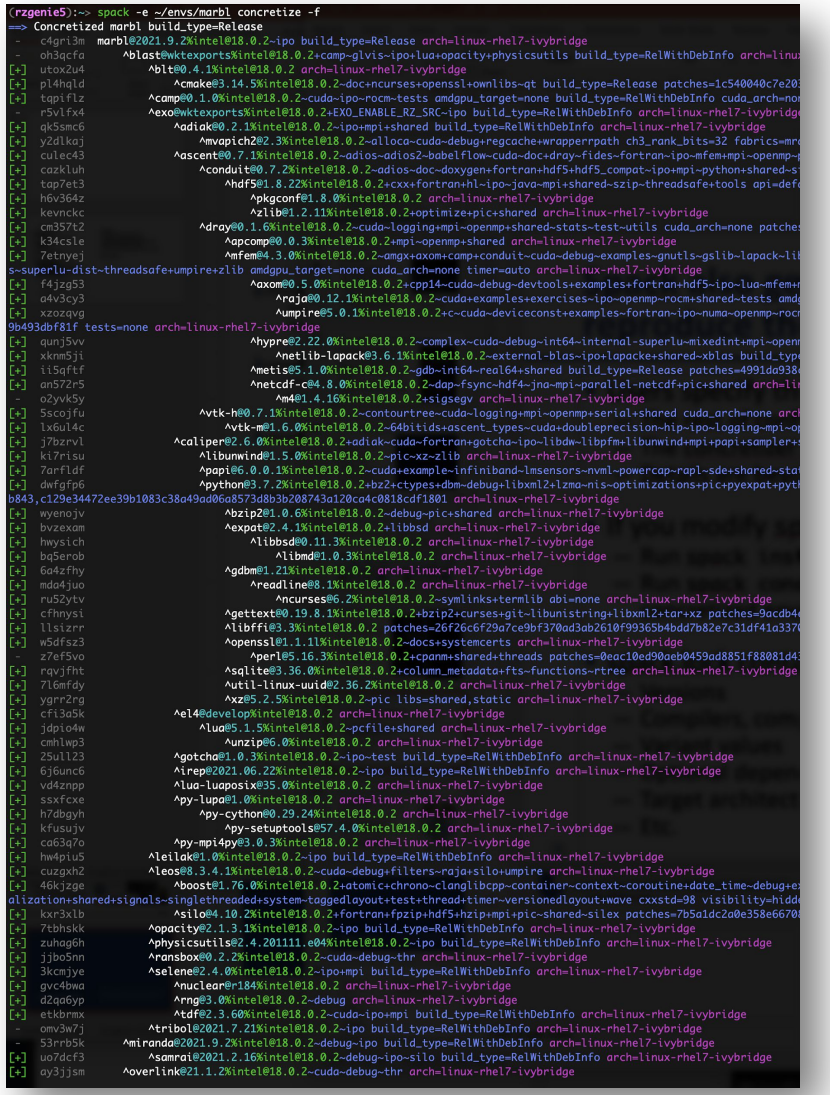
We can find ways to shorten this

spack can do multi-node builds



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



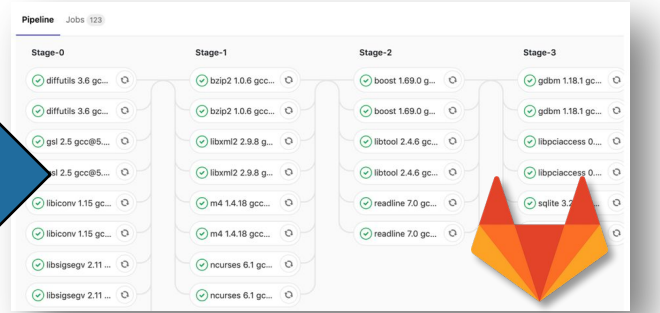
Fully concretized MARBL environment

Spack 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

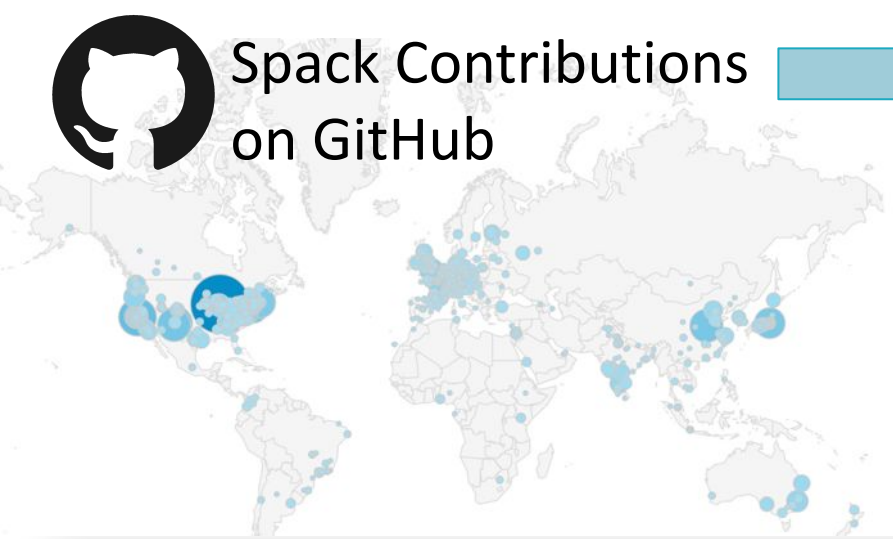
```yaml
spack:
  definitions:
  - pkgs:
    - readline@7.0
  - compilers:
    - '%gcc@5.5.0'
  - oses:
    - os=ubuntu18.04
    - os=centos7
  specs:
  - matrix:
    - [$pkgs]
    - [$compilers]
    - [$oses]
  mirrors:
    cloud_gitlab: https://mirror.spack.io
  gitlab-ci:
    mappings:
      - spack-cloud-ubuntu: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



Parallel GitLab build pipeline

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

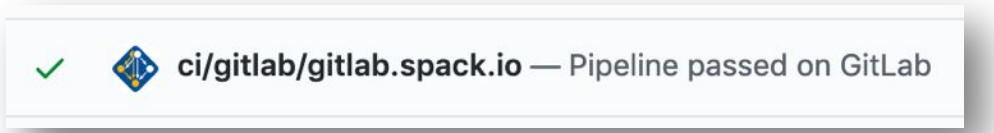
x86_64 and **aarch64** pipelines in **AWS**

ppc64le pipelines at **U. Oregon**

Pipelines at **LLNL (new)**

GitLab CI builds (changed) packages

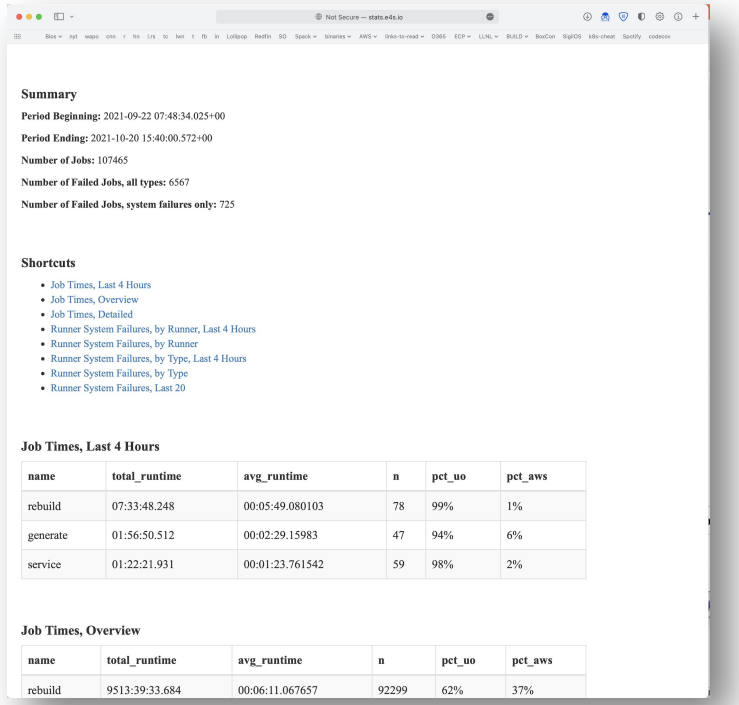
- On every pull request
- On every release branch
- Different compilers (Intel soon!)



- **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 with parallelization and caching
- Amazon and E4S/UO team helping to pinpoint errors
- We are now doing about 100,000 builds/month
- 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

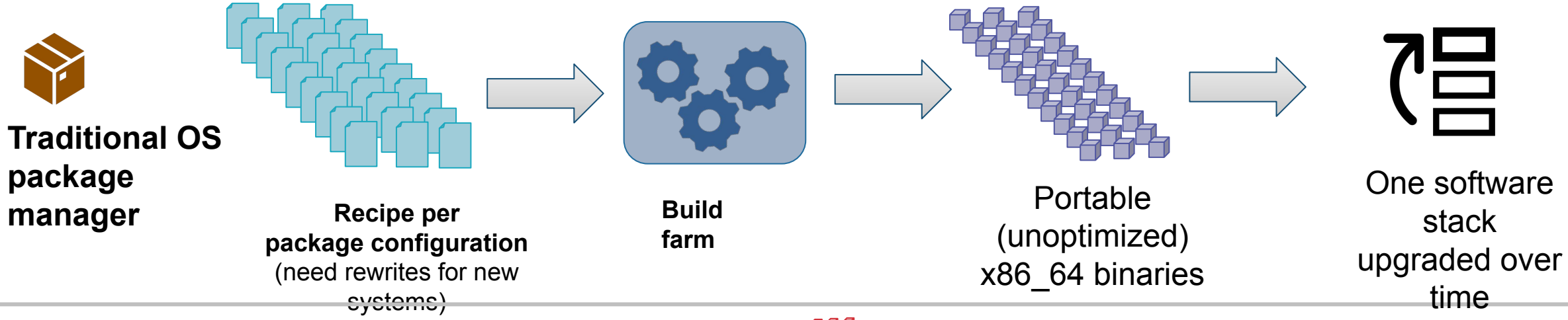
name	total_runtime	avg_runtime	n	pct_uo	pct_aws
rebuild	07:33:48.248	00:05:49.080103	78	99%	1%
generate	01:56:50.512	00:02:29.15983	47	94%	6%
service	01:22:21.931	00:01:23.761542	59	98%	2%

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



Spack v0.17.0 was just released!

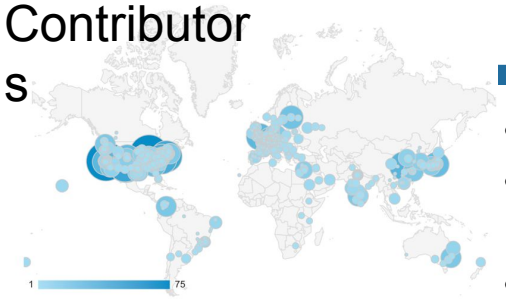
Major new features:

1. New Concretizer is now default
2. Binary bootstrapping enables us to get up and running fast
3. `spack install --reuse` aggressively reuses installed packages
4. Improved error messages
5. Conditional variants for more expressive packages
6. Git commit versioning
7. Overrides for default config directories
8. Improvements to `spack containerize`
9. New commands for querying packages and tests by tag

- 5,969 packages (920 added since 0.16)
- **Full release notes:** https://github.com/spack/spack/releases/tag/v0.17.0

High level view of a Spack package build

Contributors



- new versions
- new dependencies
- new constraints

package.py repository

spack developers admins, users

user s

user s

yaml default config packages.yaml

yaml local preferences config packages.yaml

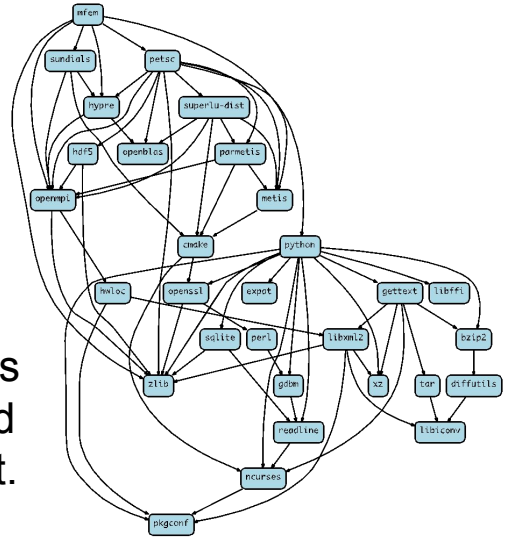
yaml local environment config spack.yaml

Command line constraints

spack install hdf5@1.12.0 +debug

concretizer

Concrete spec is fully constrained and can be built.

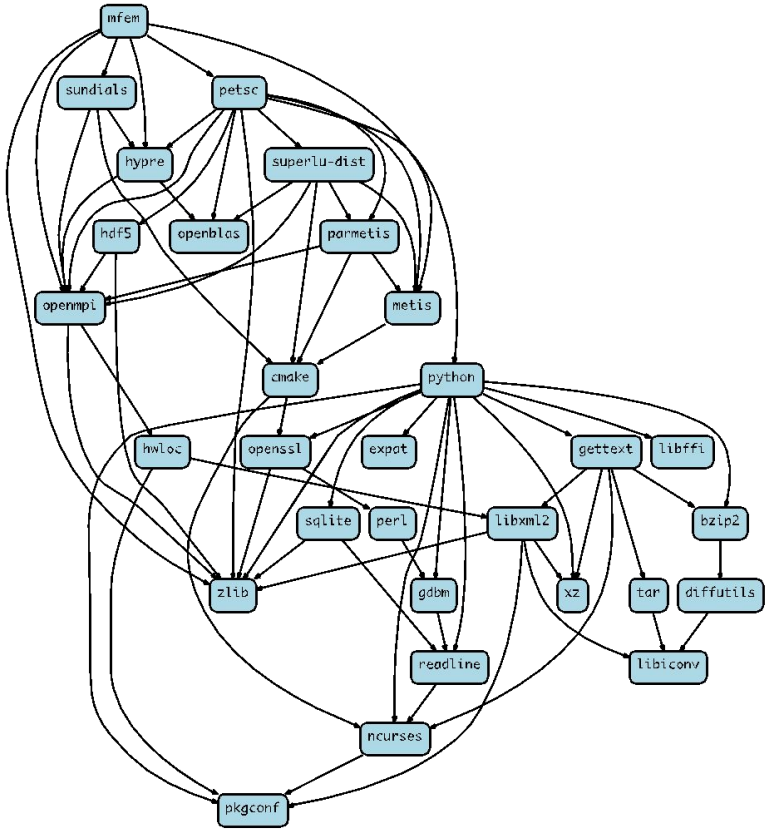


EXASCALE COMPUTING PROJECT

Package solving is *combinatorial search* with *constraints* and *optimization*

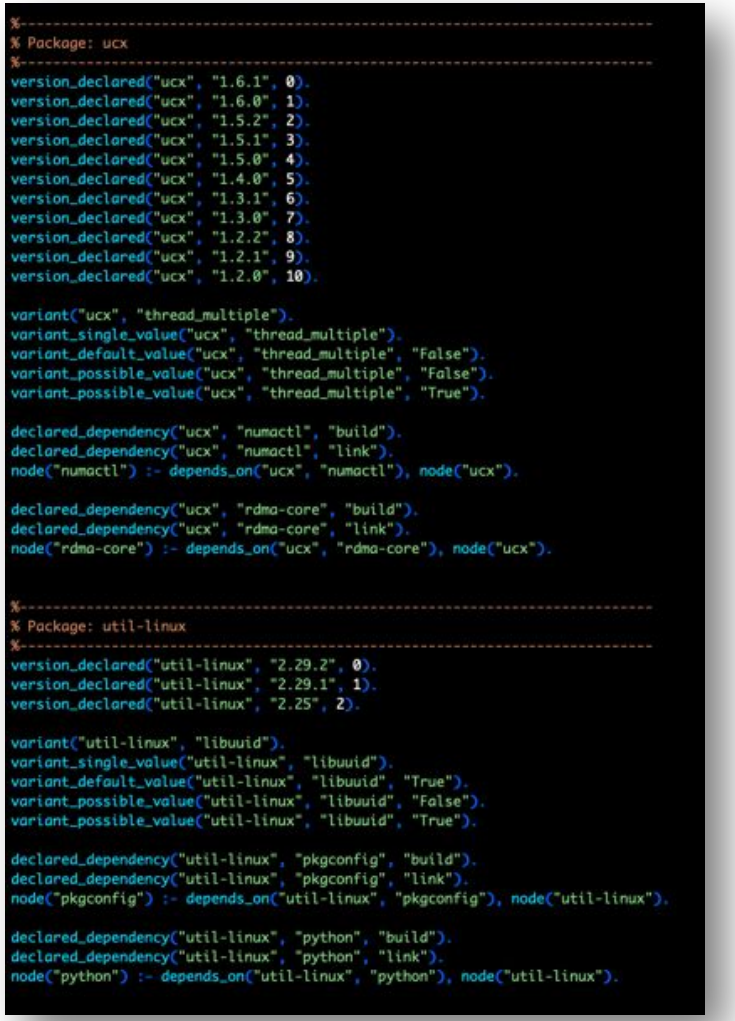
- Search over a solution space:
  - Possible dependency graphs (nodes, edges)
  - Assignment of node and edge attributes
    - Version
    - Dependency, dependency type
    - Compiler, compiler version
    - Target
    - Compiler, compiler version
- Subject to validity constraints:
  - Version requirements
  - Target/compiler compatibility
  - Virtual providers
- Optimization picks “best” among valid solutions:
  - Most recent versions
  - Preferred variant values
  - Preferred compilers that support best targets (e.g., AVX-512)
  - Minimize number of builds

This problem is NP-hard!



The new concretizer is now 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



```
%-----------------------------------------------------------------------------
% Package: ucx
%-----------------------------------------------------------------------------
version_declared("ucx", "1.6.1", 0).
version_declared("ucx", "1.6.0", 1).
version_declared("ucx", "1.5.2", 2).
version_declared("ucx", "1.5.1", 3).
version_declared("ucx", "1.5.0", 4).
version_declared("ucx", "1.4.0", 5).
version_declared("ucx", "1.3.1", 6).
version_declared("ucx", "1.3.0", 7).
version_declared("ucx", "1.2.2", 8).
version_declared("ucx", "1.2.1", 9).
version_declared("ucx", "1.2.0", 10).

variant("ucx", "thread_multiple").
variant_single_value("ucx", "thread_multiple").
variant_default_value("ucx", "thread_multiple", "False").
variant_possible_value("ucx", "thread_multiple", "False").
variant_possible_value("ucx", "thread_multiple", "True").

declared_dependency("ucx", "numactl", "build").
declared_dependency("ucx", "numactl", "link").
node("numactl") :- depends_on("ucx", "numactl"), node("ucx").

declared_dependency("ucx", "rdma-core", "build").
declared_dependency("ucx", "rdma-core", "link").
node("rdma-core") :- depends_on("ucx", "rdma-core"), node("ucx").


%-----------------------------------------------------------------------------
% Package: util-linux
%-----------------------------------------------------------------------------
version_declared("util-linux", "2.29.2", 0).
version_declared("util-linux", "2.29.1", 1).
version_declared("util-linux", "2.25", 2).

variant("util-linux", "libuuid").
variant_single_value("util-linux", "libuuid").
variant_default_value("util-linux", "libuuid", "True").
variant_possible_value("util-linux", "libuuid", "False").
variant_possible_value("util-linux", "libuuid", "True").

declared_dependency("util-linux", "pkgconfig", "build").
declared_dependency("util-linux", "pkgconfig", "link").
node("pkgconfig") :- depends_on("util-linux", "pkgconfig"), node("util-linux").

declared_dependency("util-linux", "python", "build").
declared_dependency("util-linux", "python", "link").
node("python") :- depends_on("util-linux", "python"), node("util-linux").
```

Some facts for the HDF5 package

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’
)
```

Conditional variants were previously not possible:

```
variant("cuda_arch", when="+cuda")
```

With and without reuse optimization

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

  -   zzngfs3  hdf5@1.10.7%apple-clang@13.0.0~cxx~fortran~hl~ipo~java+mpi+shared~szip~threadsafe+tools api=default b
  -   nsylovq      ^cmake@3.21.4%apple-clang@13.0.0~doc+ncurses+openssl+ownlibs~qt build_type=Release arch=darwin-bi
  -   xdbaqeo          ^ncurses@6.2%apple-clang@13.0.0~symlinks+termlib abi=none arch=darwin-bigsur-skylake
  -   kfureok              ^pkgconf@1.8.0%apple-clang@13.0.0 arch=darwin-bigsur-skylake
  -   5ekd4ap          ^openssl@1.1.1l%apple-clang@13.0.0~docs certs=system arch=darwin-bigsur-skylake
  -   xz6a265              ^perl@5.34.0%apple-clang@13.0.0+cpanm+shared+threads arch=darwin-bigsur-skylake
  -   xgt3tls                  ^berkeley-db@18.1.40%apple-clang@13.0.0+cxx~docs+stl patches=b231fcc4d5cff05e5c3a4814
  -   65edjf6                  ^bzip2@1.0.8%apple-clang@13.0.0~debug~pic+shared arch=darwin-bigsur-skylake
  -   662adoo                      ^diffutils@3.8%apple-clang@13.0.0 arch=darwin-bigsur-skylake
  -   fu7tfsr                          ^libiconv@1.16%apple-clang@13.0.0 libs=shared,static arch=darwin-bigsur-skyla
  -   vjg67nd                  ^gdbm@1.19%apple-clang@13.0.0 arch=darwin-bigsur-skylake
  -   tjceldr                      ^readline@8.1%apple-clang@13.0.0 arch=darwin-bigsur-skylake
  -   xevvljj                  ^zlib@1.2.11%apple-clang@13.0.0+optimize+pic+shared arch=darwin-bigsur-skylake
  -   xelfobh      ^openmpi@4.1.1%apple-clang@13.0.0~atomics~cuda~cxx~cxx_exceptions+gpfs~internal-hwloc~java~legacy
  -   zruns75          ^hwloc@2.6.0%apple-clang@13.0.0~cairo~cuda~gl~libudev+libxml2~netloc~nvml~opencl~pci~rocm+sha
  -   ib4fnkf              ^libxml2@2.9.12%apple-clang@13.0.0~python arch=darwin-bigsur-skylake
  -   dwiv2ys                  ^xz@5.2.5%apple-clang@13.0.0~pic libs=shared,static arch=darwin-bigsur-skylake
  -   blitnbl          ^libevent@2.1.12%apple-clang@13.0.0+openssl arch=darwin-bigsur-skylake
  -   h7jalyu          ^openssh@8.7p1%apple-clang@13.0.0 arch=darwin-bigsur-skylake
  -   7v7bqx2              ^libedit@3.1-20210216%apple-clang@13.0.0 arch=darwin-bigsur-skylake
```

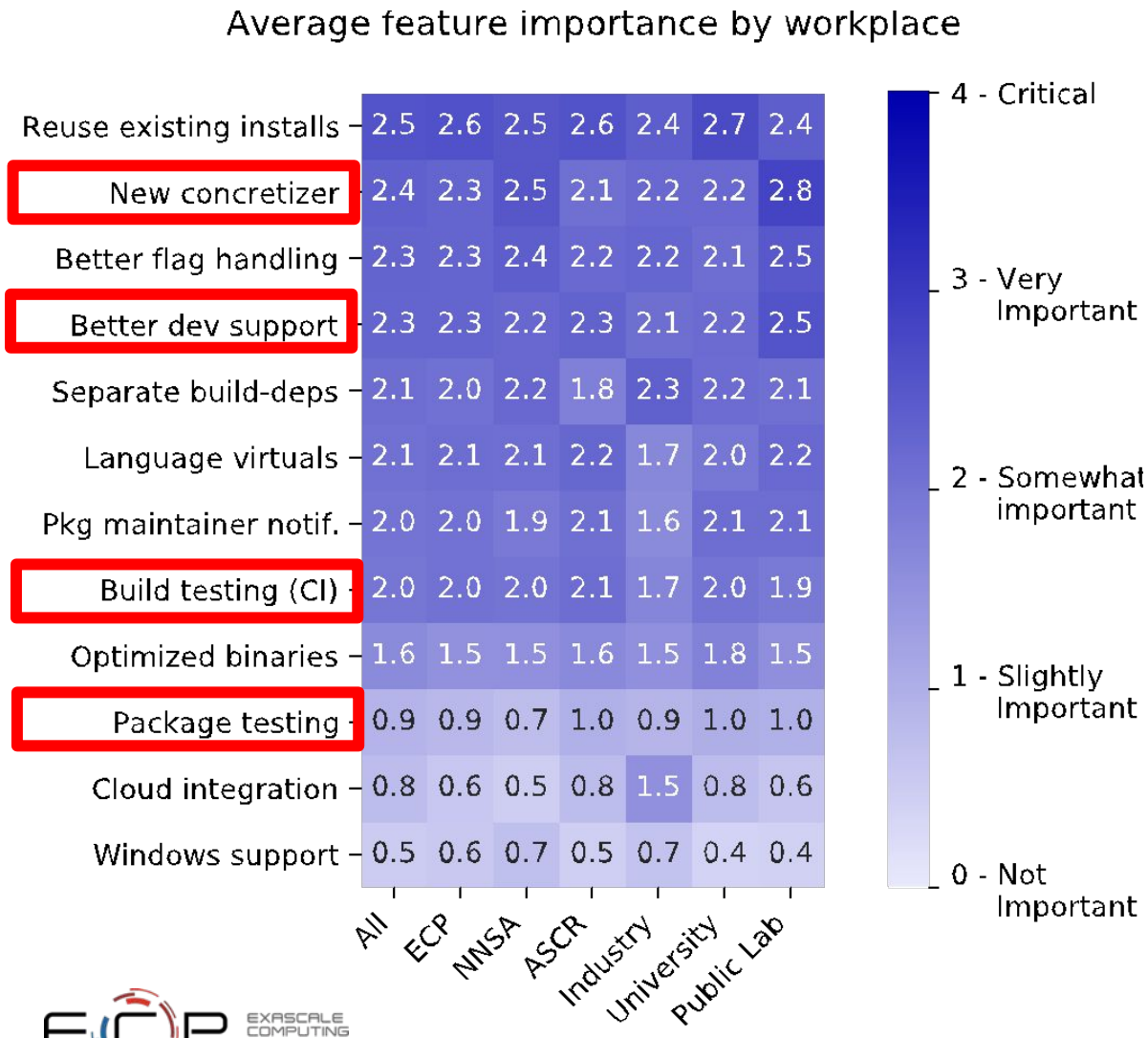
Pure hash-based reuse: all misses

Note the bifurcated optimization criteria

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

  -   yfkfnsp  hdf5@1.10.7%apple-clang@12.0.5~cxx~fortran~hl~ipo~java+mpi+shared~szip~threadsafe+tools api=defaul
 [+]  zd4m26e      ^cmake@3.21.1%apple-clang@12.0.5~doc+ncurses+openssl+ownlibs~qt build_type=Release arch=darwin
 [+]  53i52xr          ^ncurses@6.2%apple-clang@12.0.5~symlinks+termlib abi=none arch=darwin-bigsur-skylake
 [+]  us36bwr          ^openssl@1.1.1l%apple-clang@12.0.5~docs+systemcerts arch=darwin-bigsur-skylake
 [+]  74mwnxg              ^zlib@1.2.11%apple-clang@12.0.5+optimize+pic+shared arch=darwin-bigsur-skylake
  -   3ijfnel      ^openmpi@4.1.1%apple-clang@12.0.5~atomics~cuda~cxx~cxx_exceptions+gpfs~internal-hwloc~java~leg
  -   jxexyb7          ^hwloc@2.6.0%apple-clang@12.0.5~cairo~cuda~gl~libudev+libxml2~netloc~nvml~opencl~pci~rocm+
 [+]  ckdn5zf              ^libxml2@2.9.12%apple-clang@12.0.5~python arch=darwin-bigsur-skylake
 [+]  k7auat3                  ^libiconv@1.16%apple-clang@12.0.5 libs=shared,static arch=darwin-bigsur-skylake
 [+]  k2yumgx                  ^xz@5.2.5%apple-clang@12.0.5~pic libs=shared,static arch=darwin-bigsur-skylake
 [+]  grgtlcd              ^pkgconf@1.8.0%apple-clang@12.0.5 arch=darwin-bigsur-skylake
  -   nnc66ug          ^libevent@2.1.12%apple-clang@12.0.5+openssl arch=darwin-bigsur-skylake
 [+]  63xbksk          ^openssh@8.6p1%apple-clang@12.0.5 arch=darwin-bigsur-skylake
 [+]  snhgldt              ^libedit@3.1-20210216%apple-clang@12.0.5 arch=darwin-bigsur-skylake
 [+]  qbkmtdd          ^perl@5.34.0%apple-clang@12.0.5+cpanm+shared+threads arch=darwin-bigsur-skylake
 [+]  tnvkifs              ^berkeley-db@18.1.40%apple-clang@12.0.5+cxx~docs+stl patches=b231fcc4d5cff05e5c3a4814f
 [+]  7d5woqt              ^bzip2@1.0.8%apple-clang@12.0.5~debug~pic+shared arch=darwin-bigsur-skylake
 [+]  vh6di3i              ^gdbm@1.19%apple-clang@12.0.5 arch=darwin-bigsur-skylake
 [+]  qgy3v4l                  ^readline@8.1%apple-clang@12.0.5 arch=darwin-bigsur-skylake
```

With reuse: 16 packages were actually acceptable

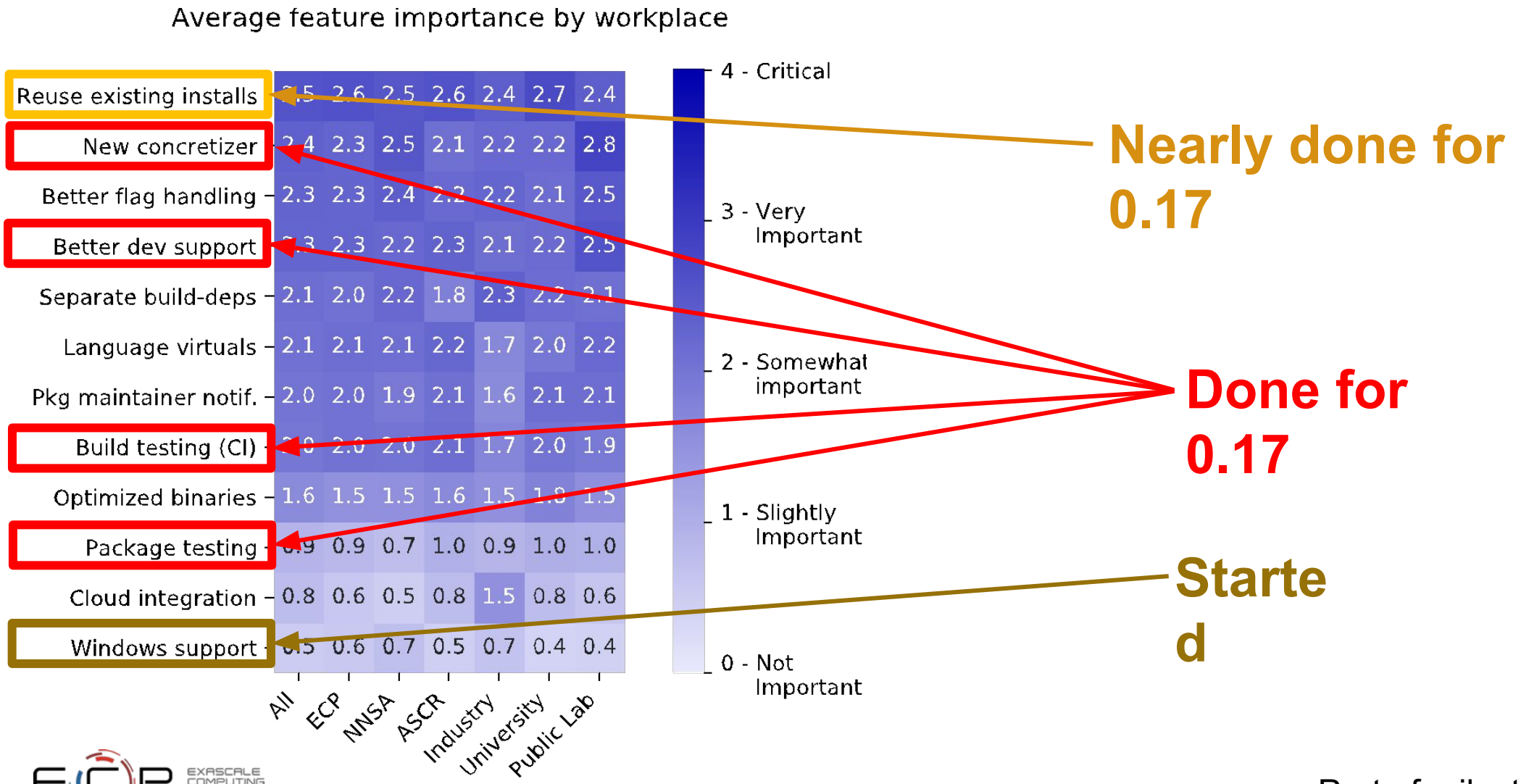
Four of the top six most wanted features in Spack were tied to the new concretizer



- 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

Part of milestone STED09-8

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



Part of milestone STED09-8

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!
  - You're already on our **Slack channel** (spackpm.herokuapp.com)
  - Join our **Google Group** (see GitHub repo for info)
  - Submit **GitHub issues** and **pull requests**!

Star us on GitHub!

github.com/spack/spack

Follow us on Twitter!

@spackpm

We hope to make distributing & using HPC software easy!

Approved for public release