Spack Update

Easybuild User Meeting 2023
April 25, 2023

Todd Gamblin
Livermore Computing
Lawrence Livermore National Laboratory

LLNL-PRES-837654

Spack enables Software distribution for HPC

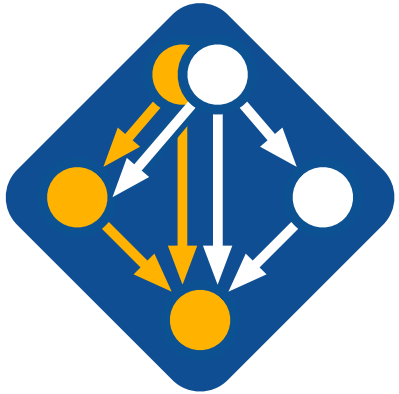
- Spack automates the build and installation of scientific software
- Packages are *parameterized;* users can easily tune for host environment

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

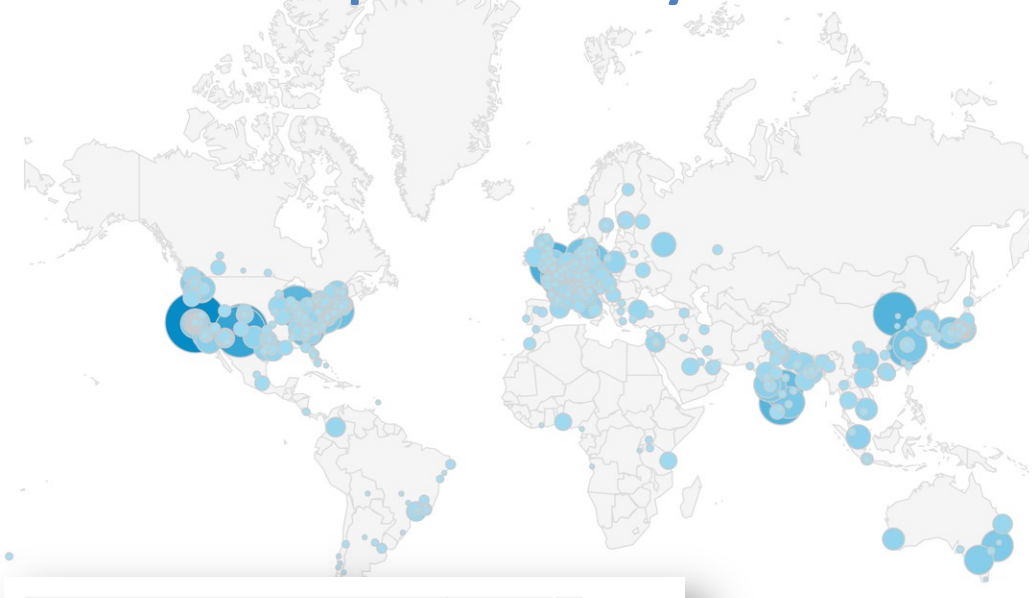


github.com/spack/spack

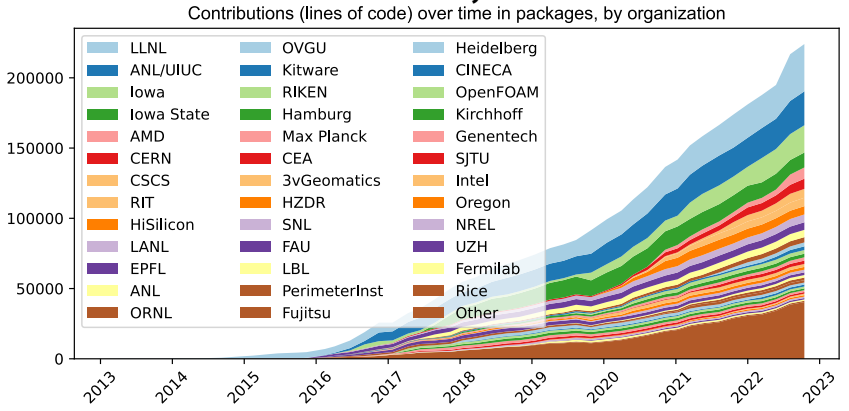
- Ease of use of mainstream tools, with flexibility needed for HPC tuning

- Major victories:
  - ARES porting time on a new platform was reduced from **2 weeks to 3 hours**
  - Deployment time for 1,300-package stack on Summit supercomputer reduced from **2 weeks to a 12-hour overnight build**
  - Official deployment tool for **ECP's E4S stack**

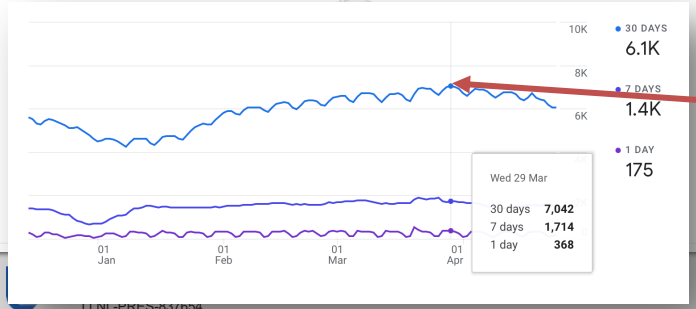
Spack sustains the HPC software ecosystem with the help of its many contributors



Over 7,100 software packages
Over 1,100 contributors



Most package contributions are ***not*** from DOE
But they help sustain the DOE ecosystem!

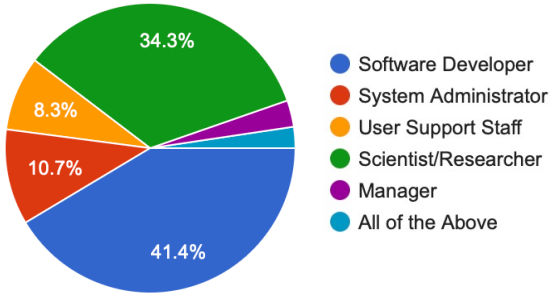


Over 7,000 monthly active users
(per documentation site)

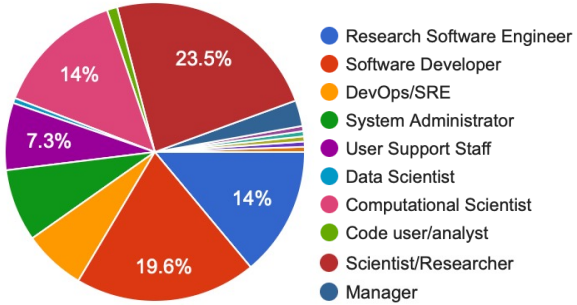
Spack users over the years

What type of user are you?

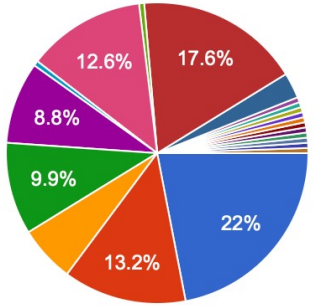
2020



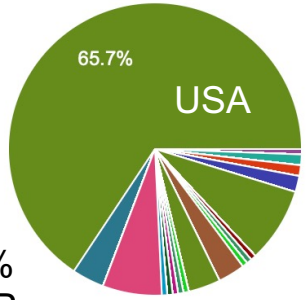
2021



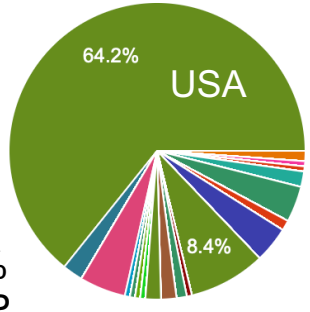
2022



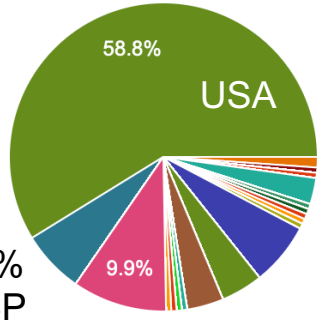
What country are you in?



36% ECP



37% ECP



26% ECP

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)

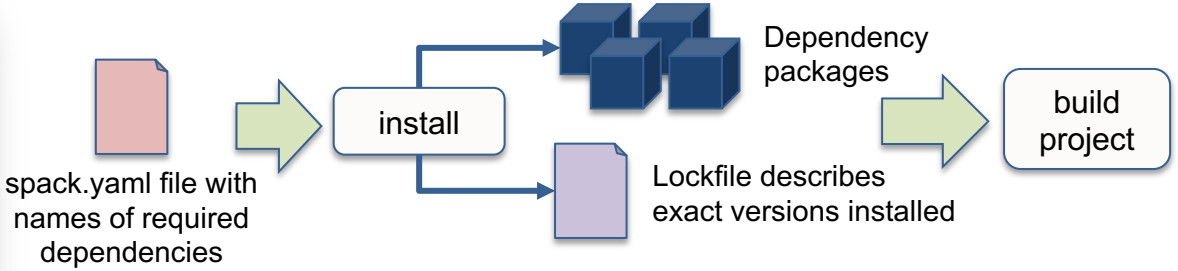
Install logic
in instance methods

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

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

Simple spack.yaml file

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

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



Concrete spack.lock file (generated)

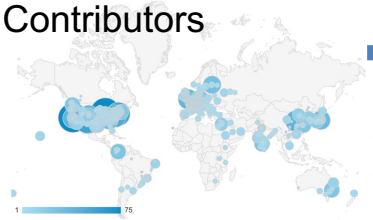
```
{
 "concrete_specs": {
   "6s63so2kstp3zyvjezglndmavy6l3nul": {
     "hdf5": {
         "version": "1.10.5",
         "arch": {
             "platform": "darwin",
             "platform_os": "mojave",
             "target": "x86_64"
         },
         "compiler": {
             "name": "clang",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

- 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

Concretization is at the core of Spack!

This problem is NP-hard!

Contributors



- new versions
- new dependencies
- new constraints

package.py repository

concretizer

spack developers

yaml default config `packages.yaml`

admins, users

yaml local preferences config `packages.yaml`

users

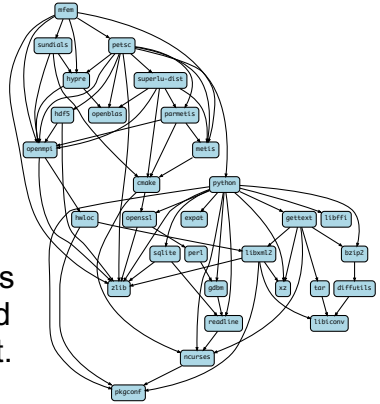
yaml environment config `spack.yaml`

users

Command line constraints

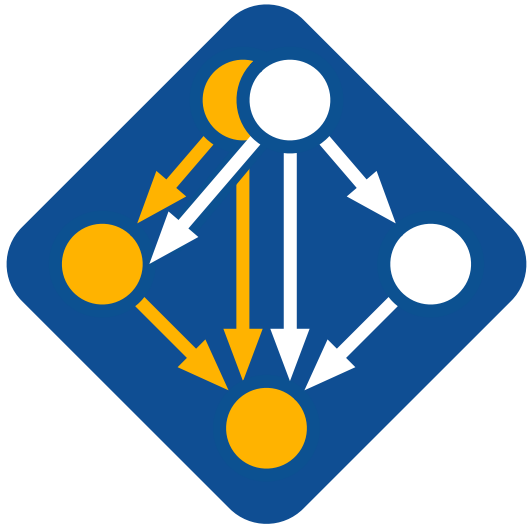
```
spack install hdf5@1.12.0 +debug
```

Concrete spec is fully constrained and can be built.



Spack v0.19.0 was released at SC22 in November

- Major new features:
  1. Package requirements
  2. Environment UI improvements
  3. Packages with multiple build systems
  4. Compiler/variant propagation
  5. Enhanced git versions
  6. Better Cray EX Support
  7. Testing and CI improvements
  8. Experimental binding link model



- This is the last Spack version that will support Python 2!
  - v0.20 will remove support
  - Already removed in develop

https://github.com/spack/spack/releases/tag/v0.19

You can now write hard requirements in packages.yaml

- Package preferences have always been soft preferences, BUT
  - Old concretizer was too naive to do soft preferences
  - they behaved more like hard prefs
- Users have asked for hard prefs
- We introduced a new syntax for require:
  - Less YAML, more spec syntax
  - Easier to remember
  - Supports one_of, any_of

```
packages:
  libfabric:
    require: "@1.13.2"
  mpich:
    require:
    - one_of: ["+cuda", "+rocm"]
```

Improvements to Environments

- Environments now default to `concretizer:unify:true`
- `unify:false` was previous default
  - Really geared toward admins and deployers, who need to fine-tune envs anyway
  - Default should reflect average user's case

```
# This is a Spack Environment file.
#
# It describes a set of packages to be installed, along with
# configuration settings.
spack:
  # add package specs to the `specs` list
  specs: []
  view: true
  concretizer:
    unify: true
```

We have tried to reduce the likelihood of unintentionally modifying `spack.yaml`

- Two major changes to environment CLI:
  - `spack install` will no longer *add* root specs
  - `spack uninstall` will not *remove* root specs
- These commands will add/remove to the root specs:
  - `spack add`
  - `spack remove`
- There are options for the old install/uninstall behavior:
  - `spack install --add`
  - `spack uninstall --remove`
- Rationale:
  - An environment are like a skeleton that can be versioned in a repo
  - add/remove modify the skeleton
  - install/uninstall determine what parts you want

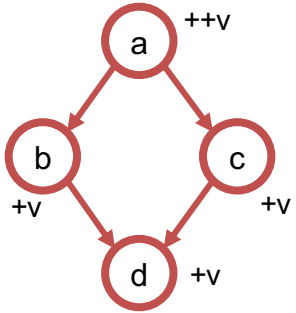
Package recipes now support multiple build systems

- An increasing number of packages in the ecosystem need this
  - Lots of switches to cmake/meson
  - Many need a different build system on Windows
- Package now has a *variant* that selects a *builder*
  - Now much easier to separate code for different build systems
  - Previous approach left multiple inheritance issues up to packagers

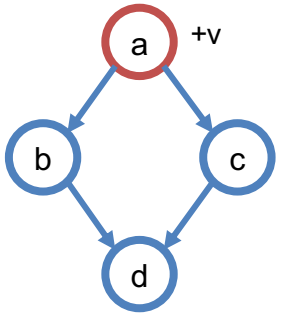
```
class ArpackNg(CMakePackage, AutotoolsPackage):

    build_system(
        conditional("cmake", when="@0.64:"),
        conditional("autotools", when="@:0.63"),
        default="cmake",
    )

class CMakeBuilder(spack.build_systems.cmake.CMakeBuilder):
    def cmake_args(self):
        pass

class Autotoolsbuilder(spack.build_systems.autotools.AutotoolsBuilder):
    def configure_args(self):
        pass
```

New feature: Variant and compiler flag propagation

- New `flag semantics`:
  - `package ++variant`: enabled variant that WILL be propagated to dependencies
  - `package +variant`: enabled variant that will NOT be propagated to dependencies
  - `package ~~variant`: disabled variant that will be propagated to dependencies
  - `package ~variant`: disabled variant that will NOT be propagated to dependencies
  - `package cflags==-g`: cflags will be propagated to dependencies
  - `package cflags=-g`: cflags will NOT be propagated to dependencies



- Previously:
  - Compiler flags propagated (were inherited by dependencies)
    - but variants were not
  - Now there's a choice, and the syntax is the same for variants and flags



- One breaking change:
  - `cflags=-g` would have propagated before
  - You will have to use `cflags==-g` to get the same effect

We have improved git versioning in Spack

- We introduced commit versions in 0.18
- Now you can do branches and tags with the `git.` prefix:

```
foo@abcdef1234abcdef1234abcdef1234abcdef1234       # raw commit
foo@git.abcdef1234abcdef1234abcdef1234abcdef1234   # commit with git prefix
foo@git.develop                                    # the develop branch
foo@git.0.19                                       # use the 0.19 tag
```

- You can also *force* Spack to interpret the commit as a concrete version:

```
# use mybranch, but treat it as version 3.2 for version comparison
foo@git.mybranch=3.2

# use the given commit, but treat it as develop for version comparison
foo@git.abcdef1234abcdef1234abcdef1234abcdef1234=develop
```

We are now treating Cray EX like Linux

- HPE/Cray has added several features that allow the Cray Programming Environment (PE) to be used in a module-less way:
  - Standalone Cray compiler drivers (`craycc`, `crayCC`, `crayftn`)
  - Cray now includes MPI wrappers with cray-mpich (`mpicc`, `mpic++`, `mpifc`, etc.)
- Cray EX machines can be either SuSE or RHEL
  - Need to model the linux distribution used properly
- Spack now treats Cray as just another Linux
  - Cray PE packages are a mixin
  - Can be detected with:

```
spack external read-cray-manifest
```

- Modules are no longer be required to build with PE externals!
  - We encourage NOT using modules to avoid surprises with PE updates

We have a new, experimental binding link model

- You can now pre-bind absolute paths to dependency libraries
  - This is like RPATH, but even HIGHER precedence
- Spack has always added RPATHs or RUNPATHs to installations
  - Can require searching a lot of installation paths
- Pre-binding avoids path searches
  - We resolve libraries in advance
  - ld.so only needs to open the library to use it
- Can greatly improve parallel startup time on many nodes

```
config:
  shared_linking:
    type: rpath
    bind: true
```

Enable this with config:shared_linking:bind:true

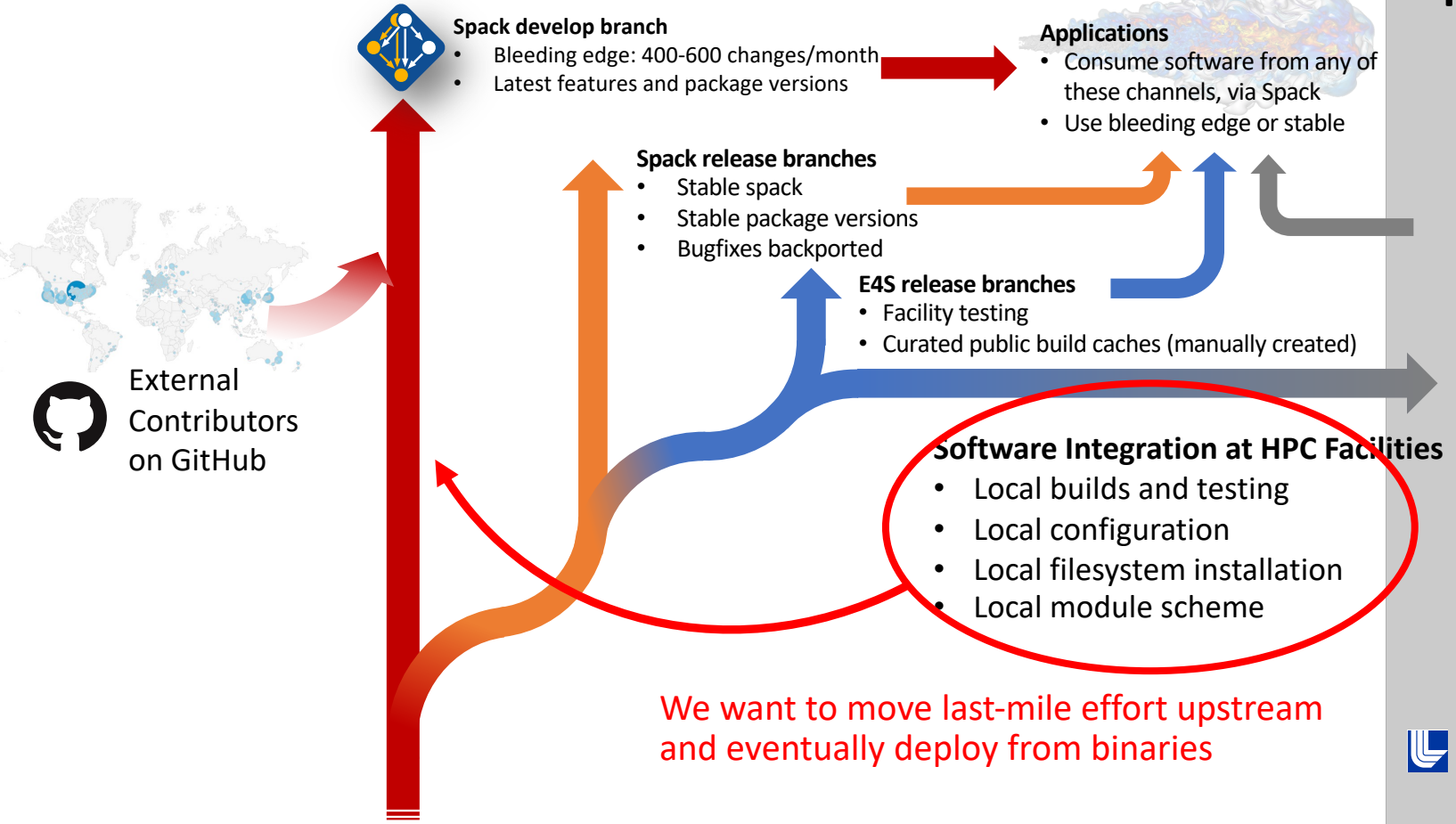
Major features for Spack v0.20 (in progress, ISC timeframe)

- `drop()` directive for inherited packages
  - Drop versions, variants, patches, etc. from parent
  - Allows subclasses to track upstream more easily
- Virtual / used variant information on graph edges
- `--reuse-deps`: reuse *only* dependencies in envs; roots are always fresh
- More regular/consistent/faster version comparison logic
  - @=4.5 to match *exactly* version 4.5
  - @4.5 matches 4.5.<anything>
- Improvements to `spack test`
  - pytest-like `test_foo`, `test_bar`, `test_baz` methods in packages
  - Plain python `assert` statements in methods
- Abstract hash references
  - Make `/abc234baf` a lazy reference instead of an immediate lookup
  - Allows use of hashes in more places
- Concrete environment inclusion
  - `include_concrete /path/to/env`
  - Allow building on past environments *without* reconcretizing
- libc compatibility model
  - Replace OS field of specs with libc version
  - Libc compatibility rules in solver
  - First step towards a proper libc node
- `require:` `when` conditions in config
  - Will make YAML config nearly as powerful as package.py

Major features for Spack v0.21 (SC23 timeframe)

- Improvements to error messages
  - Explain concretization errors
  - Track chains of constraints and report causes
- Separate concretization of build dependencies
  - Allow different nodes to have different versions of build dependencies
- Compiler dependencies
  - Model libstdc++ in the graph
  - Enforce compatibility
- Spliced installation for binaries
  - Allow system libraries (like MPI) to be spliced and RPATH'd into a binary package
  - Build in build farm w/mpich, deploy on any compatible mpich-based MPI
  - Build in build farm w/openmpi, deploy on compatible openmpi-based clusters
- Named installation trees
  - Old PR, allows Spack to have a "facility" mode with a built-in upstream
  - Users install to home directly by default but reuse from upstream install tree

We want to reduce downstream work



Facilities

Argonne NATIONAL LABORATORY

OAK RIDGE National Laboratory

BERKELEY LAB

Los Alamos NATIONAL LABORATORY EST. 1943

Sandia National Laboratories

Lawrence Livermore National Laboratory

We set out to make a binary distribution with several goals

1. **Sustainable**: Don't change maintainer workflow!
   - Limited number of maintainers working mostly in GitHub PRs
   - Most *not* actively monitoring the develop branch
   - Most don't want to babysit builds
   - Don't want extra work to cut a binary release

2. **Rolling**: Releases for common branches:
   - **develop** (most users): continuously built cache
   - **releases/***: basically just the develop stack frozen at release time

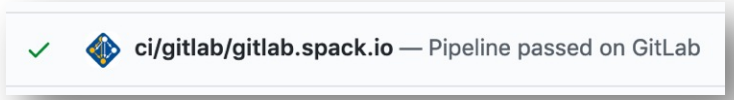
3. **Scalable**:
   - eventually support all 6,900+ packages

4. **Source-buildable**: Ensure that source builds *still* work in many environments
   - Users still build from source frequently
   - Don't assume everyone will be using binaries

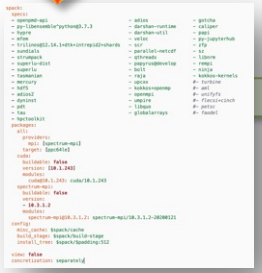
5. **Secure:**
   - Ensure that binaries are just as trustworthy as sources

Spack relies on cloud CI to ensure that builds continue working

Spack Contributions on GitHub
(over 1,000 contributors)



gitlab.spack.io



spack.yaml configurations (E4S, SDKs, AWS, others)

spack ci

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

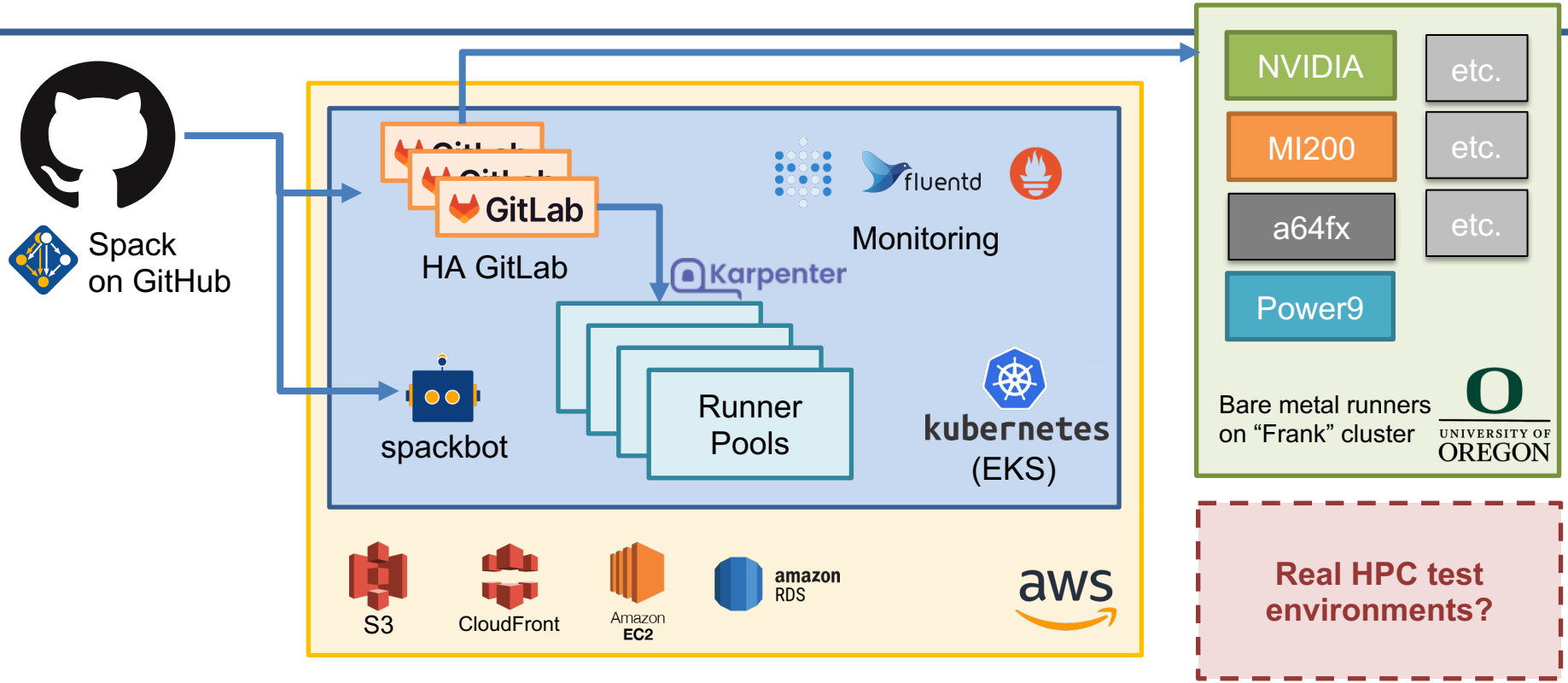
ppc64le, GPU pipelines at **U. Oregon**

Pipelines at **LLNL** **(Cray PE soon)**

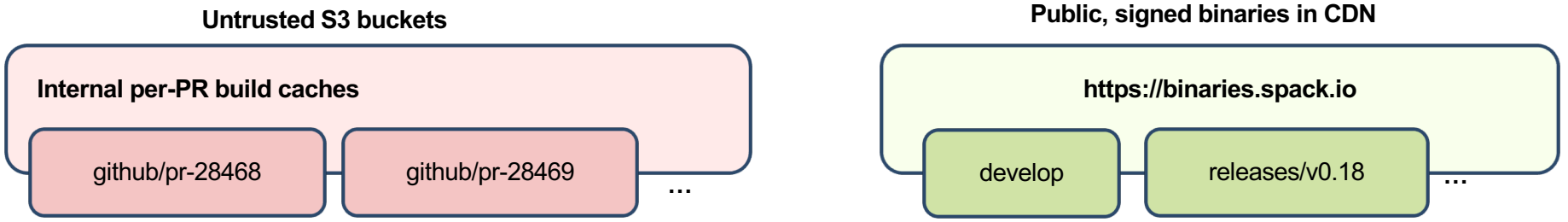
GitLab CI builds (changed) packages
- On every pull request
- On every release branch

Do users really need to build from source?

Spack CI Architecture



We use separate build environments for PRs and develop to ensure trust



Contributors submit package changes

- Iterate on builds in PR
- Caches prevent unnecessary rebuilds

Maintainers review PRs

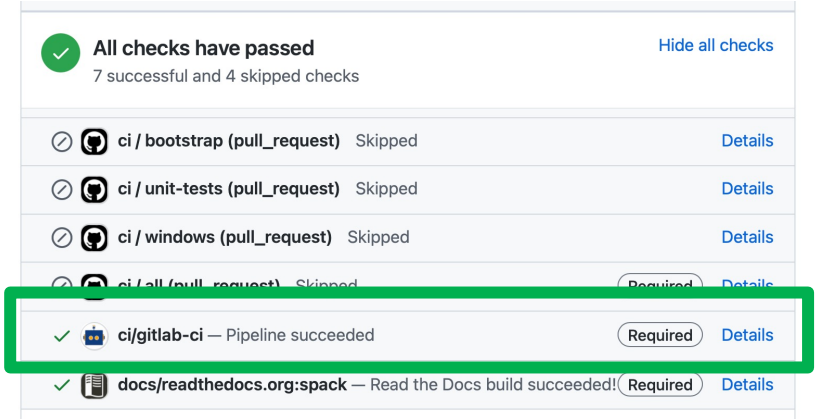
- Verify PR build succeeded
- Review package code
- Merge to develop

Rebuild and Sign

- Published binaries built ONLY from approved code
- Protected signing runners
- Ephemeral keys

- Moves bulk of binary maintenance upstream, onto PRs
  - Production binaries never reuse binaries from untrusted environment

We announced our public binary cache last June. We're maintaining ~4,600 builds in CI!



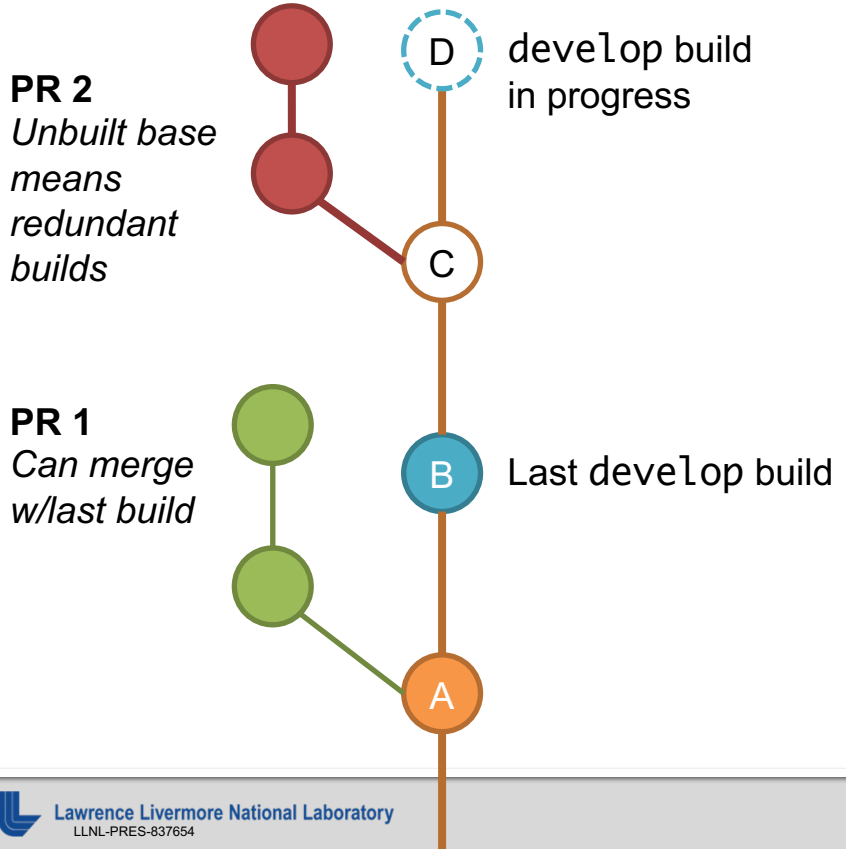
✅ Easy (mostly) for contributors!

✅ Easy for users!

```
# latest v0.18.x release binaries
spack mirror add v018 https://binaries.spack.io/releases/v0.18

# rolling release: bleeding edge binaries
spack mirror add develop https://binaries.spack.io/develop
```

So we're done, right? What could go wrong?

There is a delicate balance between redundant builds and quick PR turnaround



- We don't require PRs to be up to date
  - Too slow – we'd never get any PRs in
- GitHub normally merges with develop HEAD
  - Can cause a lot of redundant PR builds
  - Need think hard about what merge commit gets sent to GitLab
- **PR 1** can merge with **B** and get cache reuse
- **PR 2** is ahead of B
  - Merging with **C** or **D** means builds are redundant w/**D**
- Launching *many* like **PRs 2** can be ***very*** wasteful
  - So we wait for **D**

Long PR wait times can frustrate contributors

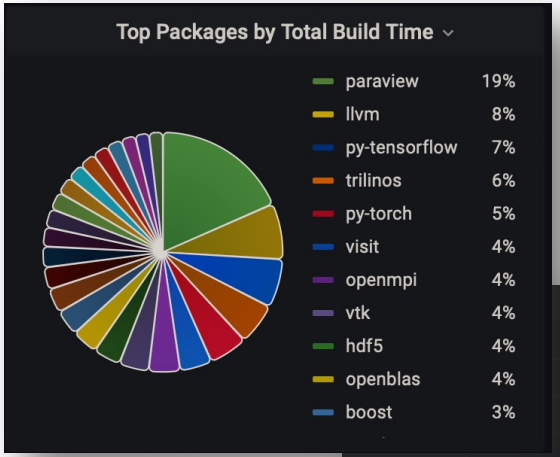
Builds have been noticeably more reliable since we added CI 🎉

but…

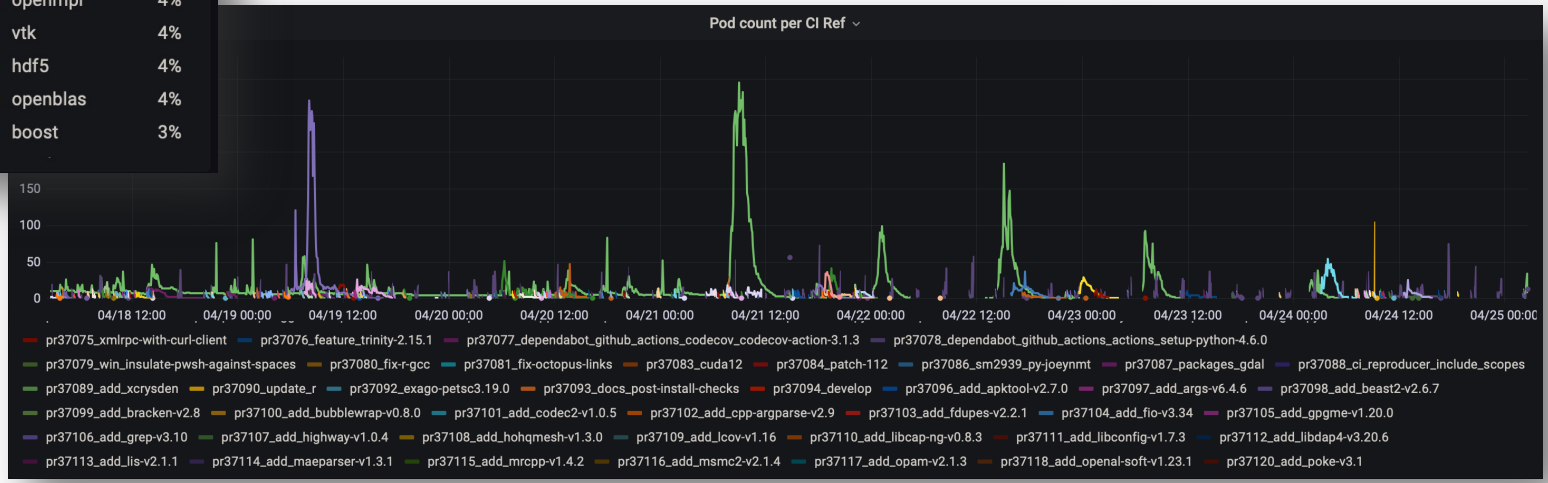
1. Most Spack committers are adding small changes (e.g., new version/option)
   - Small changes **can trigger hundreds of builds**!
2. PR tests **may not reflect software configuration** when all PRs are integrated
3. Likelihood of an **unrelated system error** in any one build is very high. 💥💣💥
4. GitLab can't always differentiate a real **build failure vs. a system failure** 🐞
   - Using k8s runners makes this worse
   - Hard to know when to auto-restart
5. Committers have to **babysit** pipelines and restart failed jobs

Contributors have become more frustrated as stacks have grown

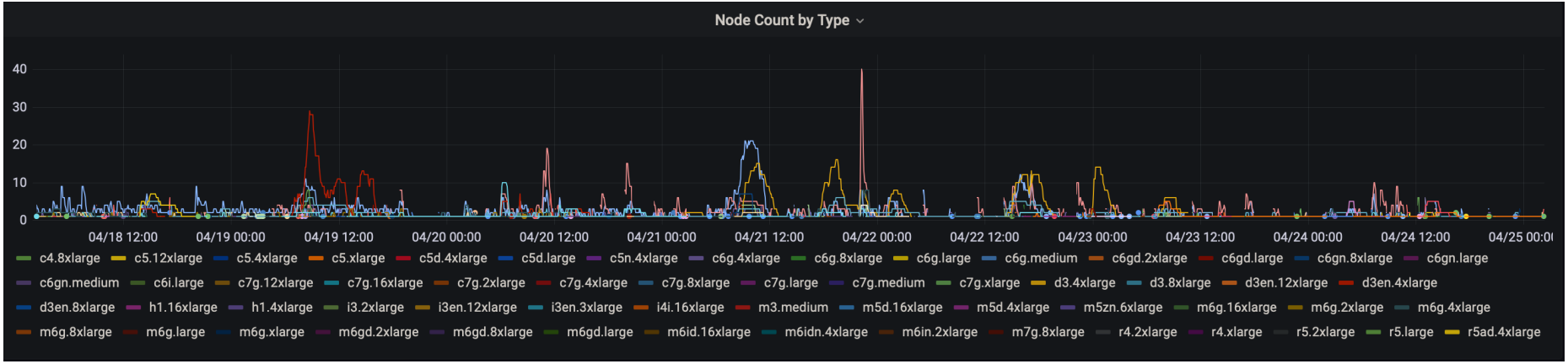
We have used monitoring to improve understanding of CI workload



- There is room for improvement with ParaView builds
  - Not sure why it's significantly higher than other packages
- Can trace load on the system to specific PRs/branches:



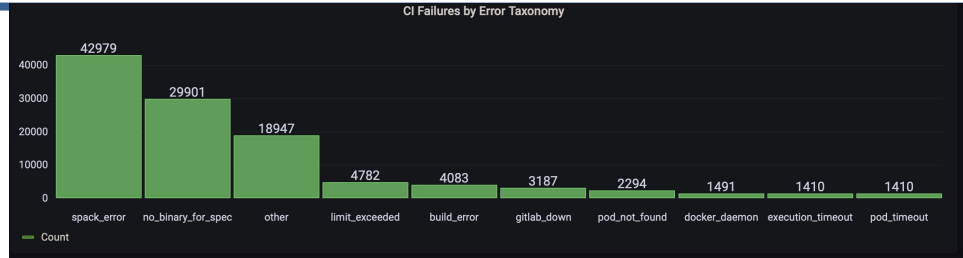
Karpenter allows us to manage many instance types

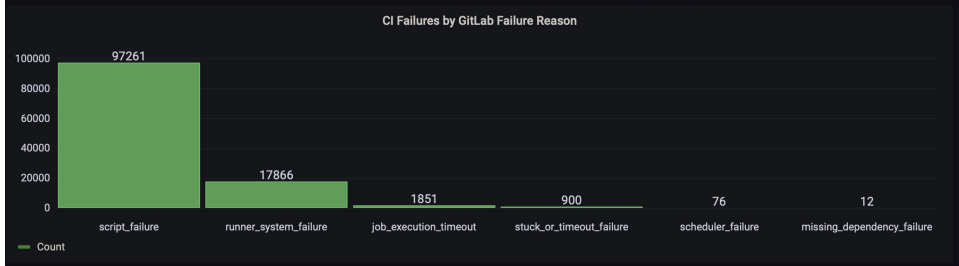


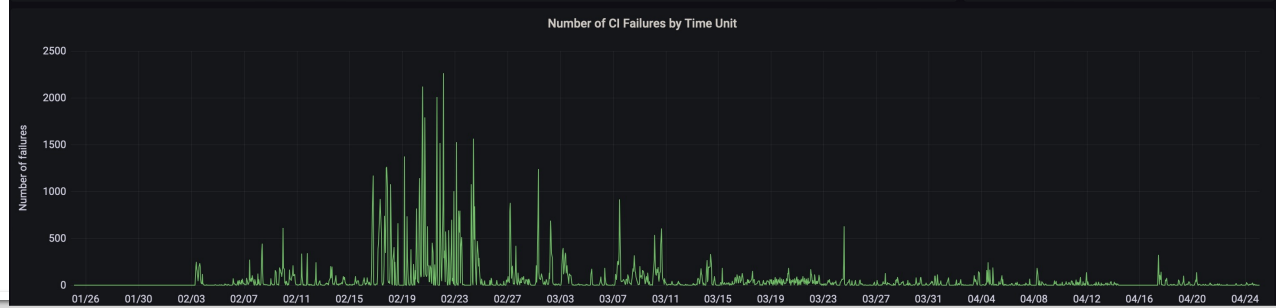
- We request instances with archspec tags
- Karpenter maps archspec names to instance types
  - Not automatic currently, but makes things way easier
- We use multiple instance types to get greater spot capacity

In February, our monitoring revealed a significant problem with pod scheduling

- Karpenter tool was killing build jobs to consolidate nodes
  - Led to many pipeline failures
- Designed for stateless services, NOT build jobs
  - Build is more like batch

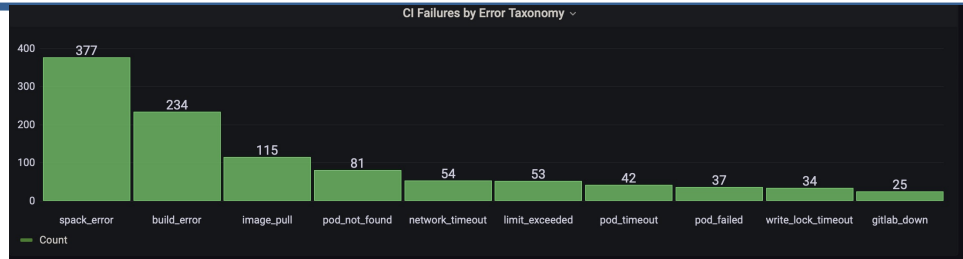


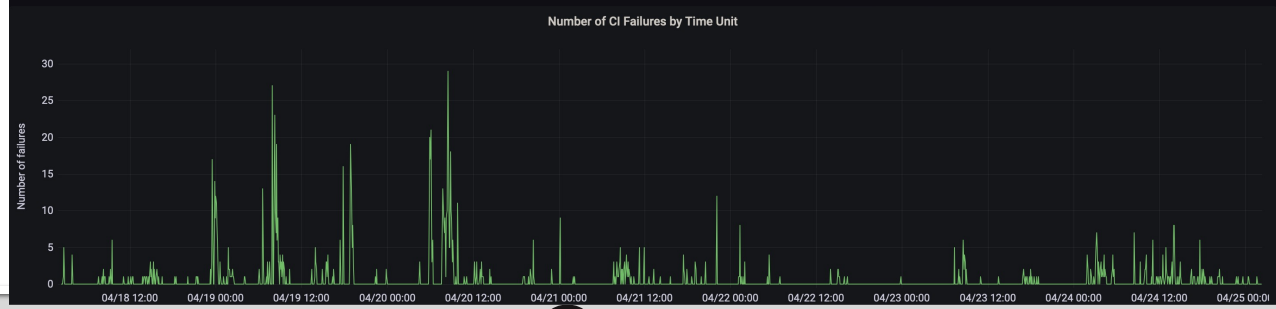




We were able to trace most of our reliability issues to Karpenter (container scheduling) configuration

- Tweaked a few settings:
  - Disabled consolidation for build jobs
  - Added affinity preferences to pack pods tighter
- Went from ~12% failure rate to 2% failures
  - Most remaining errors are normal development issues
  - Working to differentiate from system issues.





Some takeaways

- Spack community continues to grow
  - European user and contributor base appears to be increasing
- We are moving fast on feature and package development
  - Many features done per release
  - Environments, workflows, testing, new package and concretization features
  - Lots of work on core to make package installation versatile and easy
- CI is helping us keep the software stack stable
  - Contributors can easily rebuild all dependents of a single package
  - Now even easier to add a new stack to CI
- Better monitoring is helping us keep CI stable!
  - Still learning how to run a reliable, distributed cloud service
  - Concerted effort to improve dashboards and reduce errors has helped a lot