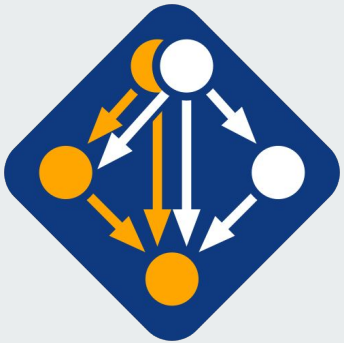
What's new in Spack?



Massimiliano Culpo
Harmen Stoppels

Easybuild User Meeting
22.04.2026

Agenda

1. Spack basics
2. Highlights since last year:
    a. Repo split
    b. Compilers as dependencies
    c. New installer UI
    d. Environment groups & compiler bootstrapping
3. Looking forward

What is Spack?

- Package manager with dependency resolver
- 8700+ packages available
- Multiple variants of the same package can be installed at a time

```
$ git clone https://github.com/spack/spack
$ . spack/share/spack/setup-env.sh
$ spack install hdf5
```

https://github.com/spack/spack

Spec syntax: describe package configuration

```
$ spack install hdf5
$ spack install hdf5@1.14                  # specific version
$ spack install hdf5@2.1 %clang@22         # different compiler
$ spack install hdf5@2.1 +threadsafe       # different variant
$ spack install hdf5@2.1 target=haswell    # cpu target
$ spack install hdf5@2.1 %mpi=mpich@5      # use mpich 5 for mpi
```

- 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

https://github.com/spack/spack

Package recipes are parametrized

- One `package.py` file per software project!

```python
from spack_repo.builtin.build_systems.cmake import CMakePackage
from spack.package import *

class Kripke(CMakePackage):
    """Kripke is a simple, scalable, 3D Sn deterministic particle transport mini-app."""

    homepage = "https://computation.llnl.gov/projects/co-design/kripke"
    url = "https://computation.llnl.gov/projects/co-design/download/kripke-openmp-1.1.tar.gz"

    version("1.2.7", sha256="3f7f2eef0d1ba5825780d626741eb0b3f026a096048d7ec4794d2a7dfbe2b8a6")
    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 [
            self.define_from_variant("ENABLE_OPENMP", "openmp"),
            self.define_from_variant("ENABLE_MPI", "mpi"),
        ]
```

Base package
(CMake support)

~~**Metadata** at~~ the class level

~~**Versions**~~

~~**Variants** (build~~ options)

Dependencies
(same spec syntax)

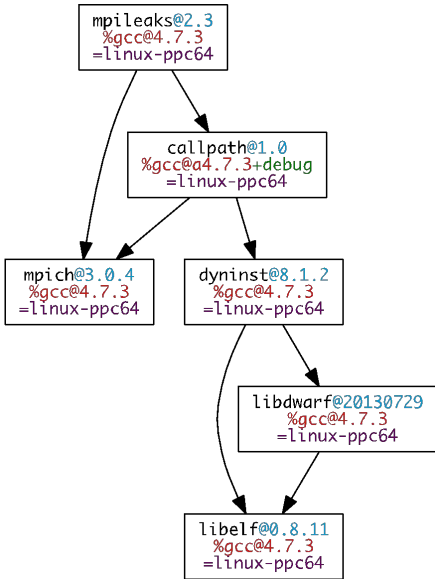
Install logic
in instance methods

Concretization fills in missing details

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

User input: *abstract* spec with some constraints

Concretize



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

Store

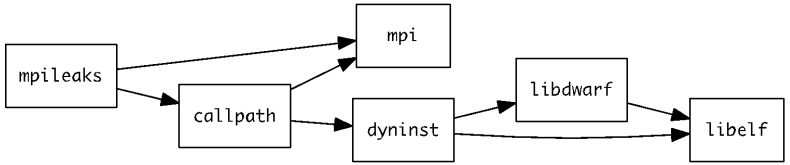
spec.json


```
{
  "spec": {
    "_meta": {
      "version": 5
    },
    "nodes": [
      {
        "name": "mpileaks",
        "version": "1.0",
        "arch": {
          "platform": "linux",
          "platform_os": "ubuntu24.04",
          "target": {
            "name": "zen2",
            "vendor": "AuthenticAMD",
            "features": [
              "abm",
              "aes",
              "avx",
              "avx2",
              "bmi1",
              "bmi2",
              "clflushopt",
              "clwb",
              "clzero",
              "cx16",
              "f16c",
```


Detailed provenance is stored with the installed package

Hashing handles combinatorial complexity

Dependency DAG



Installation Layout

Hash

```
opt
└── spack
    └── linux-zen2
        ├── callpath-1.0.4-cnjleehfmh3w3tclzvi6rveg4tswwapt
        ├── dyninst-13.0.0-kuzfbjmvviyr4qv3f7z2fayxy4gypivd
        ├── libelf-0.8.13-smygnxezmr7w3iiva7d7rcxyezbs4lg6
        ├── libdwarf-2.3.0-tglzjgrlstdo3vpzslc26wi5nhkwnvpx
        ├── openmpi-5.0.10-eyx4ylhim4ftx2bqkxxggx22xpc2js5o
        └── mpileaks-1.0-ameienzgxrxbg6rdwidc2t3ski64tfwk
```

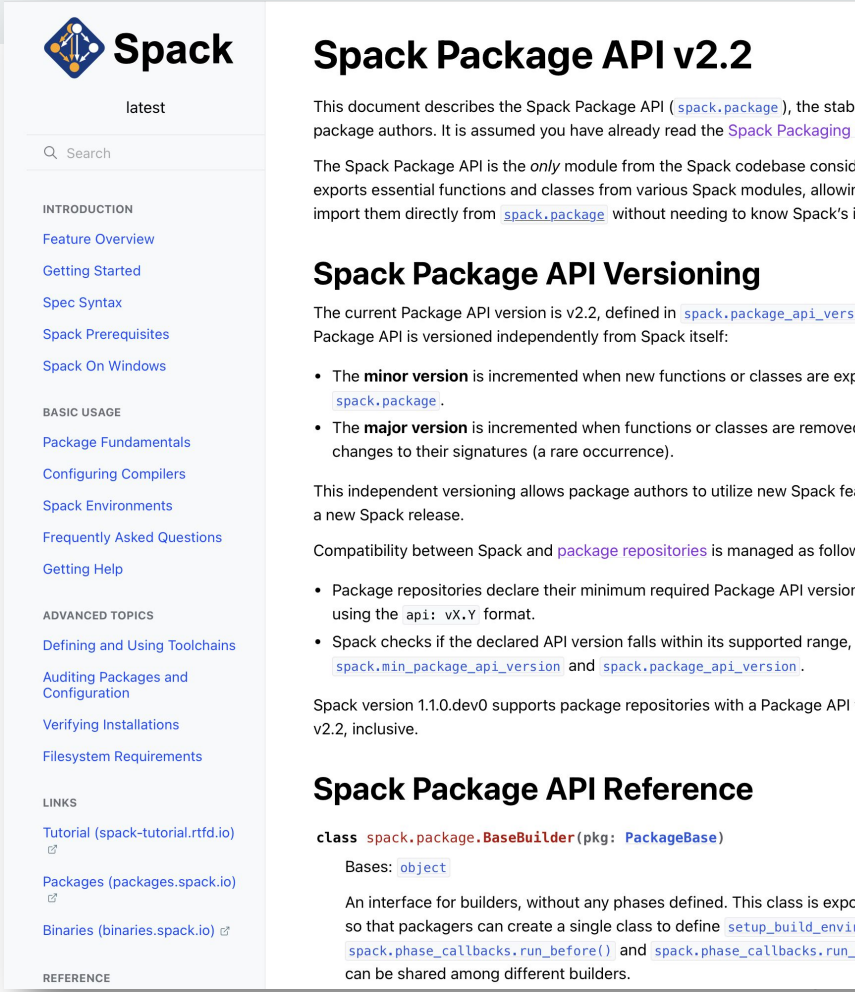
- Each unique dependency graph is a unique ***configuration***.
- Each configuration in a unique directory.
  - Multiple configurations of the same package can coexist.
- **Hash** of entire directed acyclic graph (DAG) is appended to each prefix.
- Installed packages automatically find dependencies
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

Package repository and API

Package repo split

- Community wanted
  - package updates without tool changes (e.g. new bugs)
  - tool updates without package changes (reproducibility)
- As of Spack v1.0, we split it up:
  - `spack`: command line tool
  - `spack-packages`: repository with 8700+ recipes
- What glues them together is a stable package api
  - `from spack.package import *`

https://github.com/spack/spack

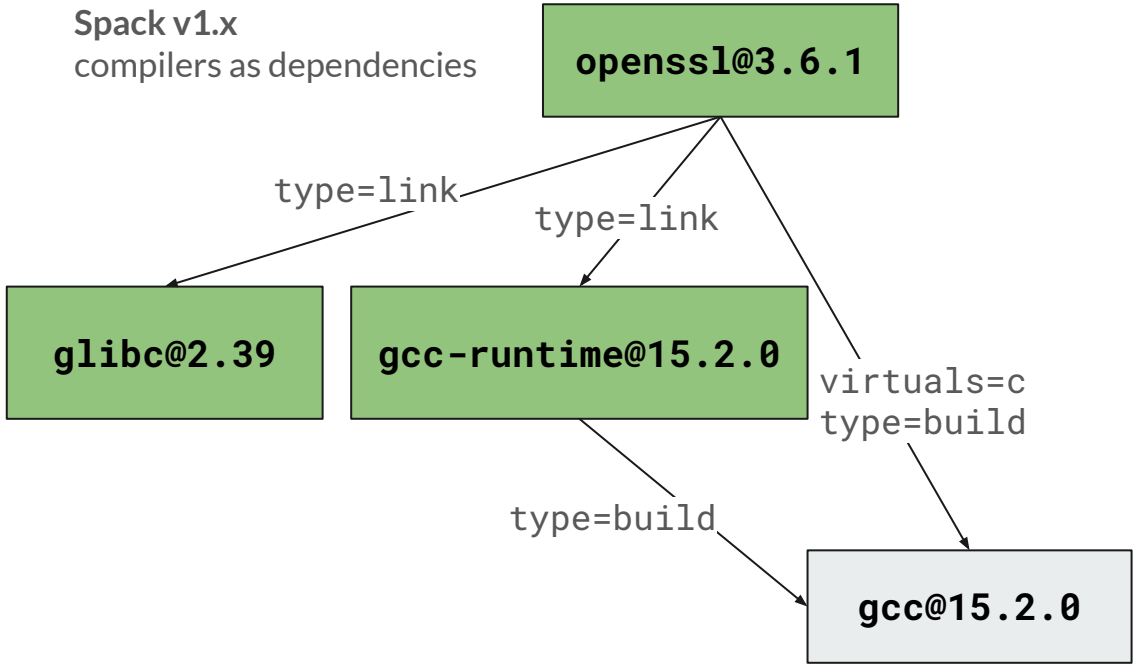


Compilers as dependencies

Spack v0.x
compilers as attributes

openssl@3.6.0
compiler: gcc@15.2.0

Spack v1.x
compilers as dependencies



https://github.com/spack/spack

Compilers as dependencies: basics

- Languages `c`, `cxx` and `fortran` are *virtuals*
- Ordinary packages *depend* on them: `depends_on("c", type="build")`
- Compiler packages *provide* them: `provides("fortran", when="+fortran")`
- Compilers *inject* dependencies (runtimes) into the parent node
  - `gcc-runtime`
  - `glibc`
- There can be multiple `glibc`s and `gcc-runtime`s in one DAG
- They provide virtuals, e.g. `libc`, `libgfortran@4`, `libgfortran@5`, ... to deal with ABI compat (prevents certain `gfortran` combinations.)

Compilers as dependencies: syntax

- We've added new syntax: `%c,cxx=llvm@22 %fortran=gcc@15`
  - Depend on LLVM for the `c` and `cxx` language/virtual
  - Depend on GCC for the `fortran` language/virtual
- E.g. `conflicts("%cxx=gcc@14:")`

- On the command line, configured toolchains can be used to shorten expressions: `%clang_with_gfortran`

New installer: terminal UI

```
tmp.IH1ktuTu5K $
```

The terminal UI: logs mode

- Press **`v`** to toggle between overview and "follow logs"
- Press **`n`**/**`p`** to cycle through concurrent builds
- **`/`** and **`Enter`** to look at logs of a specific builds (name or hash)

```
Progress: 0/2  /: filter  v: logs  n/p: next/prev
[/] qydeqpd gromacs@2026.0 fetching from build cache
[/] 2apvqsq sirius@7.10.0 starting
```

New installer: jobserver

Composable parallelism

- **Old installer**: two independent knobs **`spack install -p4 -j4 llvm`**
  - **`-p`**: how many packages in parallel
  - **`-j`**: how many jobs per package
  - Maximum load: `4 * 4 = 16`.
  - **But**: only **16** jobs *if there are 4 concurrent builds.*
    The top level package **`llvm`** builds with *only **4** jobs* 😔

- **New installer**: single flag **`spack install -j16 llvm`**
  - Maximum load **16**
  - *Dynamic* number of jobs per package: maybe **`4/4/4/4`**, maybe **`6/2/1/5/2`**
  - Top level package **`llvm`** uses all **16** jobs 🎉
  - You can still use **`-p`** to limit the number parallel packages

https://github.com/spack/spack

2025: renaissance of the POSIX jobserver

- June 2025: **Ninja** v1.13.0**:**
  - After 9 years!
  - After 3 forks!
  - Recommended read: https://neugierig.org/software/blog/2020/05/ninja.html
- March 2026: **LLVM** v22.1.0
- June 2026**: Spack** v1.2.0 enabled by default (technically from Spack v1.1.0)

- May 2025: JuliaLang/**julia**#58591 considers it

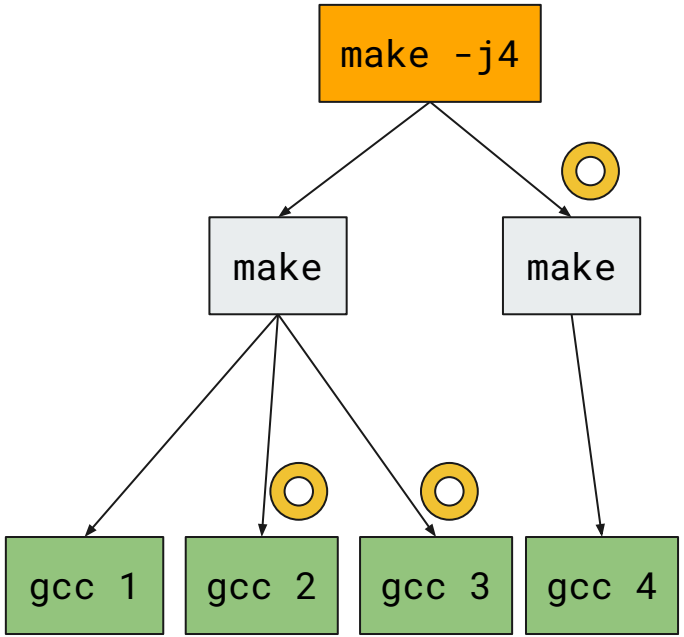
What's a jobserver

- Introduced by GNU Make in **1999**
- **Composable parallelism** across process tree
- Long-time support in GCC and Cargo

https://github.com/spack/spack

POSIX jobserver: pouch of coins

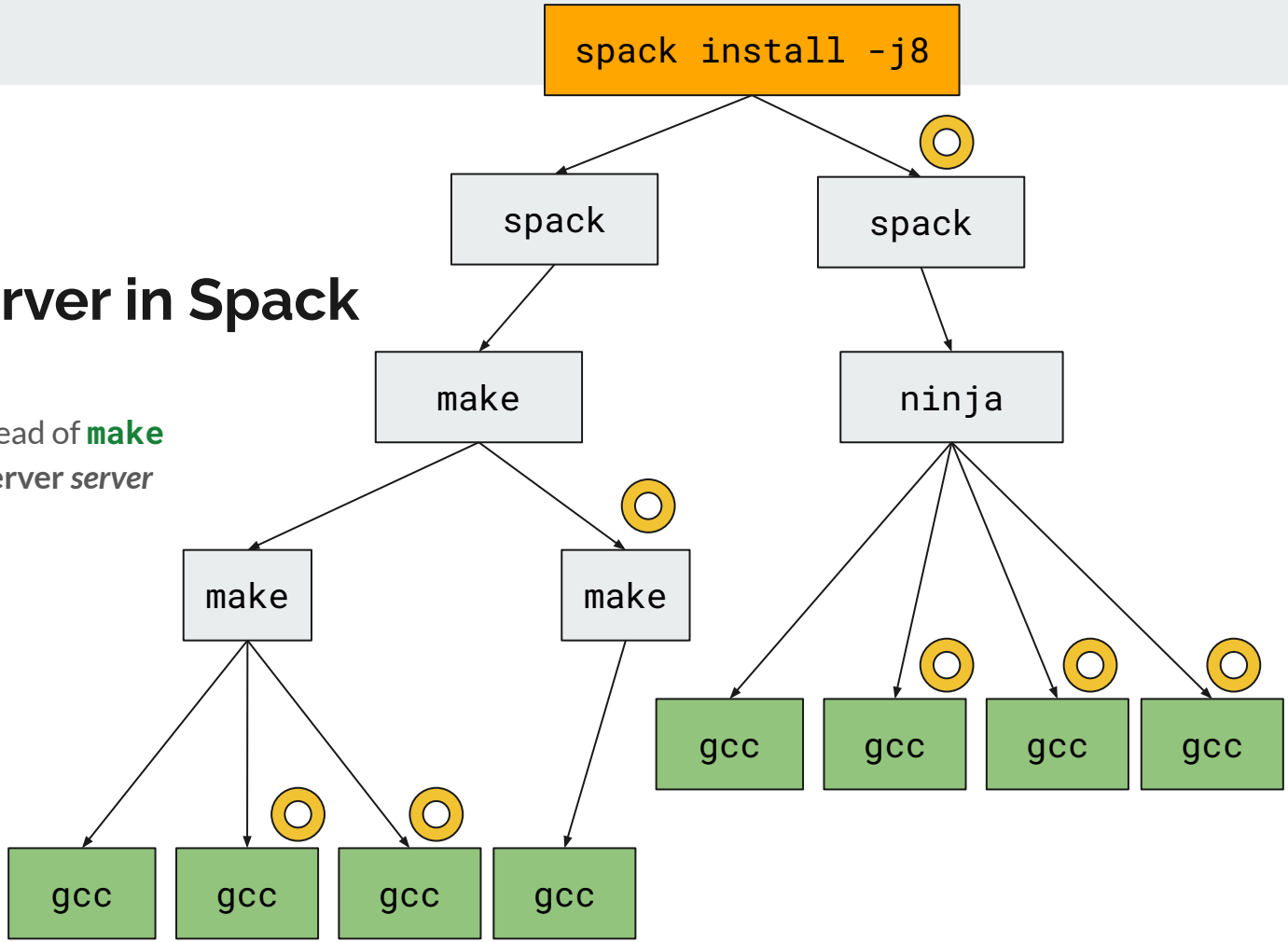
- Create or **`open`** a **`pipe`**
- **`write`** about **`N-1`** bytes (aka tokens/coins)
- Before starting *additional* job: **`read`** one byte
- After finishing *additional* job: **`write`** one byte
- Advertise pipe to child processes
  - **`MAKEFLAGS=--jobserver-auth=...`**



- Process tree has N leaf nodes
- Interior nodes are idle

POSIX jobserver in Spack

- Spack on top instead of **`make`**
- Spack is the **jobserver** ***server***



https://github.com/spack/spack

Bonus feature: dynamic -j

- Press **+** for more jobs and **–** for fewer

```
tmp.A8pdaT5WHC $
```

Multi-node parallelism

Multi-node parallelism

- *One* `spack install` process can install multiple packages in parallel
- But you can also have *multiple* `spack install` processes

- Example: `srun -N4 -n1 -c32 spack install -p2 -j32`
  - 4 nodes
  - 1 spack install process per node
  - 2 concurrent builds per spack install process
  - 32 jobs per node
- Or backgrounded: `spack install -p2 -j8 & spack install -p2 -j8 &`

Demo: environment + build cache

Sharing binaries

- `spack buildcache push ./binaries` `# tarball them`

- `spack mirror add my_buildcache ./binaries` `# use them`
- `spack install`

Installing development versions

```
spack install cp2k@master dev_path=/home/harmen/projects/cp2k/cp2k
```

Named group of specs

Some workflows are iterative or require multiple envs

```
# Create the environment
$ spack env create stack
$ spack env activate stack

# First step is install the compiler
$ spack add gcc@15
$ spack concretize && spack install

# Then concretize the stack
$ spack add gromacs hdf5 libtree
$ spack concretize && spack install
```

Bootstrapping a compiler:

- Iterative in the same environment
- Use a dedicated environment for compilers

Multiple configurations of the same stack:

- Often needs multiple environments
- E.g. pure CPU vs. CUDA vs ROCm

https://github.com/spack/spack

Spack v1.2: group of specs in environments

```
spack:
  specs:
  - group: compiler
    specs:
    - gcc@15.2

  - group: apps
    needs: [compiler]
    specs:
    - hdf5 %gcc@15.2
    - libtree %gcc@15.2
```

```
$ spack env create stack && spack env activate stack
$ spack config edit
[ ... ]
$ spack concretize && spack install
```

Groups are named and can have dependencies

Groups are concretized independently

If a group depends on another:

- It's concretized after its dependencies
- Specs from dependencies are *always* reused in concretization

Groups can override configuration

https://github.com/spack/spack

Demo: bootstrapping a compiler + create OCI image

```
$
```

Deployment at HPC sites

Intermezzo: understanding Spack config scopes

```
$ spack config scopes -vp
Scope           Type          Status  Path
command_line    internal      active
spack           path          active  /home/culpo/PycharmProjects/spack/etc/spack/
user            include,path  active  /home/culpo/.spack/
site            include,path  active  /home/culpo/PycharmProjects/spack/etc/spack/site/
system          include,path  active  /etc/spack/
defaults        path          active  /home/culpo/PycharmProjects/spack/etc/spack/defaults/
defaults:linux  include,path  active  /home/culpo/PycharmProjects/spack/etc/spack/defaults/linux/
defaults:base   include,path  active  /home/culpo/PycharmProjects/spack/etc/spack/defaults/base/
_builtin        internal      active
```

https://github.com/spack/spack

Deploy different configurations of the same stack

```
- group: ml-linux-x86_64-rocm
    specs:
    - $jax_specs
    - matrix:
      - [$keras_specs]
      exclude:
      - py-keras backend=jax
      - py-keras backend=torch
    override:
      packages:
        all:
          require:
            - ~cuda
            - +rocm
            - amdgpu_target=gfx90a
```

```
109  08:23:14  ==> Concretizing the 'ml-linux-x86_64-rocm' group of specs
110  08:23:14  ==> Starting concretization pool with 8 processes
111  08:23:37  ==>    22.4s [  9%] aw7mmqf py-tensorboardx
112  08:23:39  ==>    24.3s [ 18%] 3jkvóaq py-tensorboard
113  08:23:41  ==>    27.0s [ 27%] ózsglbl py-transformers
114  08:23:42  ==>    28.0s [ 36%] t7zf3eu py-scikit-learn
115  08:23:50  ==>    35.5s [ 45%] z2iikhó py-jax
116  08:23:51  ==>    36.7s [ 54%] gp4h5wp py-tensorflow
117  08:23:52  ==>    37.5s [ 63%] j23kpyi py-jaxlib
118  08:23:53  ==>    39.0s [ 72%] f54twco py-keras backend=tensorflow
119  08:23:58  ==>    19.3s [ 81%] 726n2bg py-tensorflow-metadata
120  08:24:03  ==>    26.6s [ 90%] 56amp7n py-tensorflow-datasets
121  08:24:14  ==>    32.9s [100%] zcjuinb py-tensorflow-probability
122  08:24:14  ==> Concretizing the 'ml-linux-x86_64-cuda' group of specs
123  08:24:14  ==> Starting concretization pool with 8 processes
124  08:24:41  ==>    26.7s [  3%] ózsglbl py-transformers
125  08:24:47  ==>    32.8s [  6%] j23kpyi py-jaxlib
```

Unifying the ml-linux pipelines:

- Reduced number of builds by 50%
- Consolidated configuration in one place

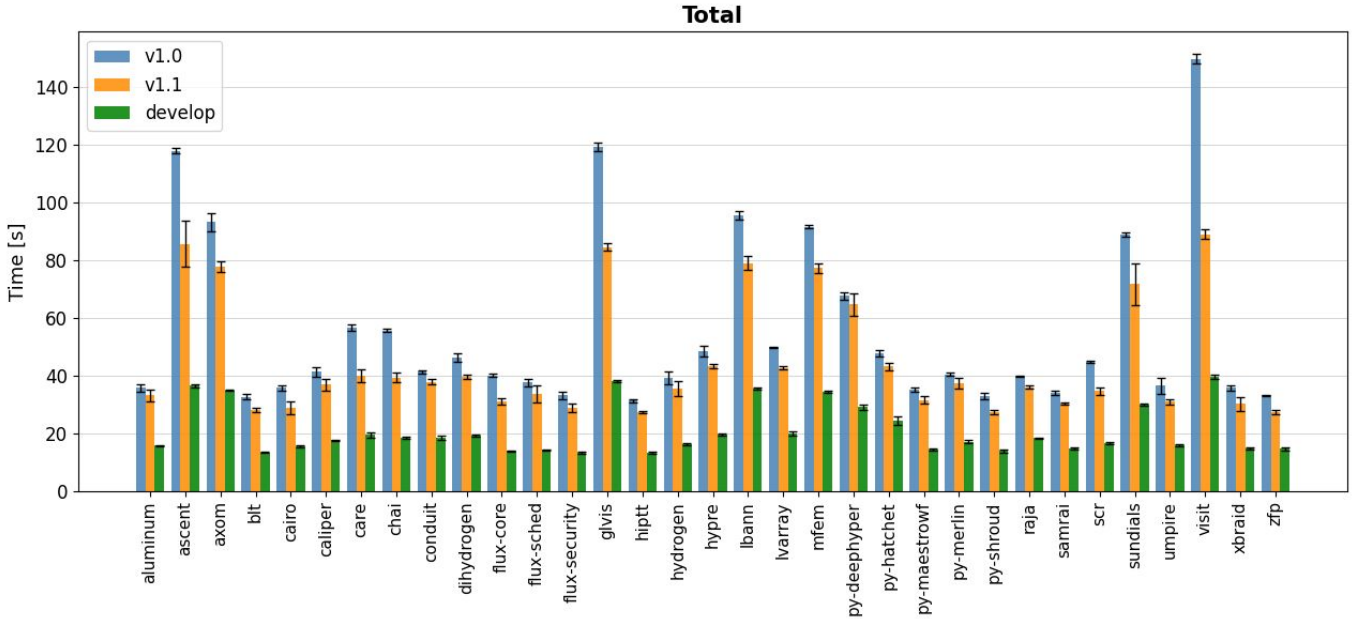
https://github.com/spack/spack

Improved concretizer performance

Lots of performance improvements for Spack v1.2

- clingo binaries with optimizations (PGO, LTO, mimalloc, etc.)
- Improved encoding in ASP
  - Reduced the dimension of the grounded problem
  - Improved heuristics when searching for a solution
- Improved performance of problem setup
  - Lazy package metadata evaluation
  - Immutable abstract specs and reduced allocations

Comparison across Spack versions



https://github.com/spack/spack

Other features in Spack v1.2

Spack v1.2 - expected June 2026

- New installer: jobserver + new UI
- Support for “group of specs” in environments
- Improved solver performance
- Improvements to error messages
- SBOM and more detailed provenance
- XDG compliance
- pip installable Spack
- Concretization cache

https://github.com/spack/spack

Post-v1.2 ideas
(not in a release milestone yet)

Post-v1.2 plans

- Better build isolation and reproducibility
- Automate the updates to Go, Rust, Python, etc.
- Compiler bootstrapping *in the solver*
- Native Spack container images (a.k.a. bootstrap libc)

Questions?