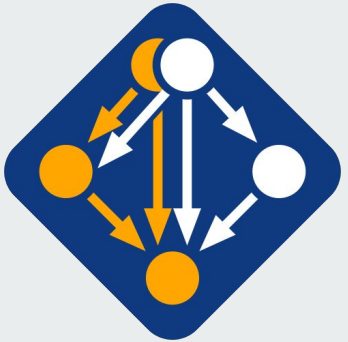
Spack v1.0 and beyond

Harmen Stoppels

FOSDEM
01.02.2026



Agenda

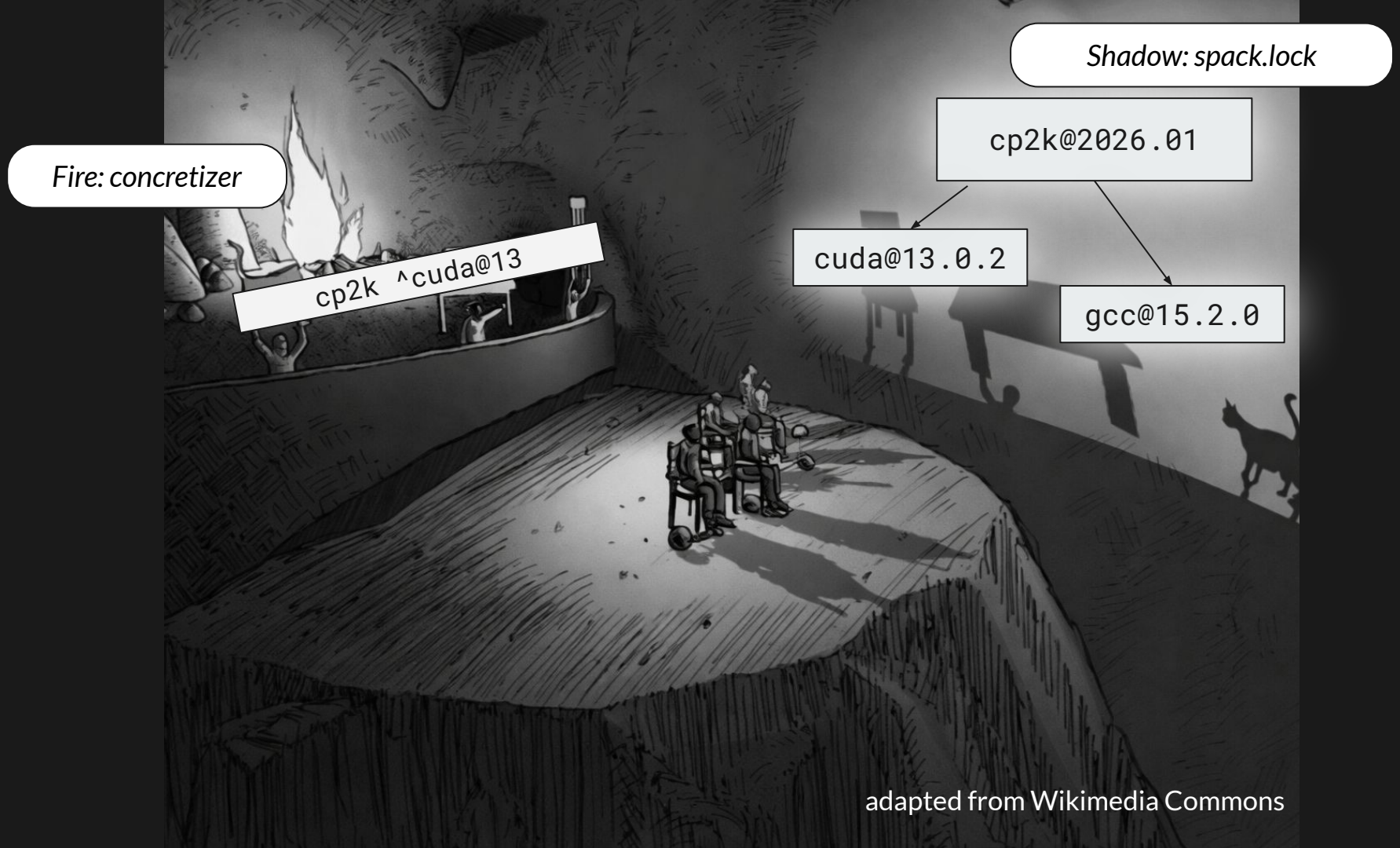
1. What's unique about Spack?
2. What's new in Spack?
    a. Repository split
    b. Experimental installer & jobserver
    c. Compilers as dependencies

Who am I?

- Harmen Stoppels
- Based in Zürich
- Independent developer at Stoppels Consulting
- Involved with Spack since 2020

- You can find me on Github: https://github.com/haampie
- And on the Spack Slack: https://slack.spack.io/

What's unique about Spack?



adapted from Wikimedia Commons

Nix/Guix

```
 (define-public python-scipy
   (package
     (name "python-scipy")
-    (version "1.12.0")
+    (version "1.16.3")
     (source
      (origin
        (method url-fetch)
        (uri (pypi-uri "scipy" version))
        (sha256
-        (base32 "18rn1..."))))
+        (base32 "1jxf6..."))))
```

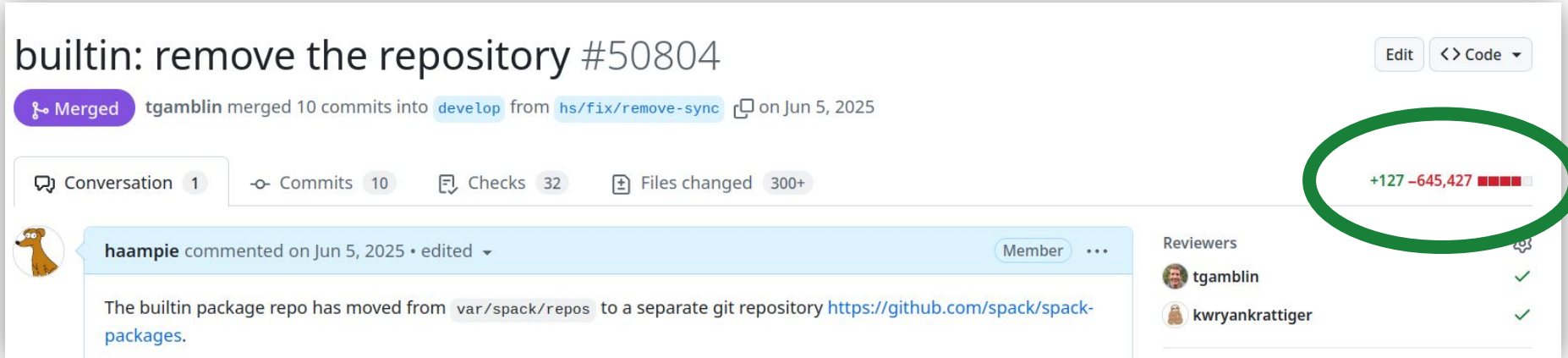
Spack

```
 class PyScipy(PythonPackage):

     pypi = "scipy/scipy-1.16.3.tar.gz"

+    version("1.16.3", sha256="01e87...")
     version("1.16.2", sha256="af029...")
     ...
     version("1.12.0", sha256="e1ad5...")
     ...
     version("1.7.0", sha256="998c5...")
```

Package repository split

Perfect PR

- Tool: spack/spack
- Packages: spack/spack-packages (with history)



https://github.com/spack/spack

Package API: spack <-> spack-packages

- `from spack.package import *`
- Versioned API as `vX.Y`
  - Rarely ever bump `X`
  - Only sometimes bump `Y`
- Set in stone: https://spack.readthedocs.io/en/latest/package_api.html

Experimental installer & jobserver

Experimental installer

- https://asciinema.org/a/755827
- Enabled via `config:installer:new`
- One `-j` to rule them all
  - **Package parallelism by default**
  - POSIX jobserver and client
- Basic TUI:
  - Overview mode
  - Logs mode
- Simple event loop with `epoll`/`kqueue`
- Feature-parity in Spack v1.2 (~June 2026)

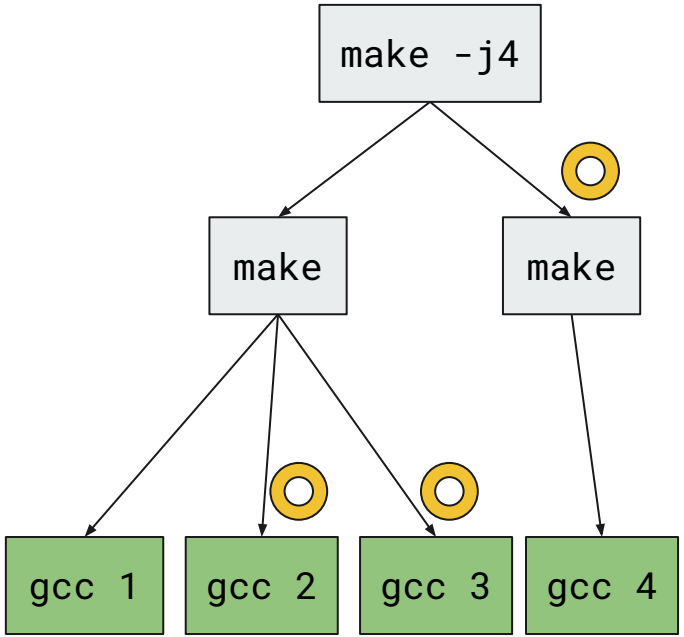
https://github.com/spack/spack

POSIX jobserver

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

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

2025: renaissance of the POSIX jobserver

- June 2025: **Ninja** v1.13.0**:**
  - After 9 years!
  - After 3 forks!
  - Highly recommended read: https://neugierig.org/software/blog/2020/05/ninja.html
- October 2025: **LLVM** v22.1.0 (expected)
- November 2025**: Spack** v1.1.0

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

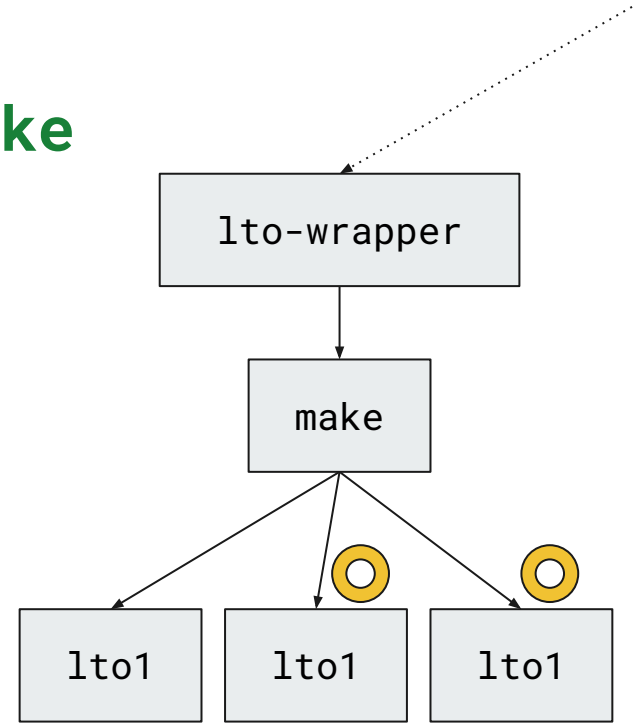
But: one bad apple spoils the whole barrel

- The server process has no clue who took tokens
- One process fails to return a token
- → oops, build continues with limited parallelism

Common workaround: delegate to `make`

- **GCC**'s `lto-wrapper` executable does this to this day:
  - Generate a temporary `Makefile`
  - Run `$MAKE`

- **Spack** used to promote this to power users:
  - `spack env depfile -o Makefile && make -j128`
  - But: graph is not static
  - But: overhead for short installs
  - New installer makes this redundant



https://github.com/spack/spack

But: not "modern" enough

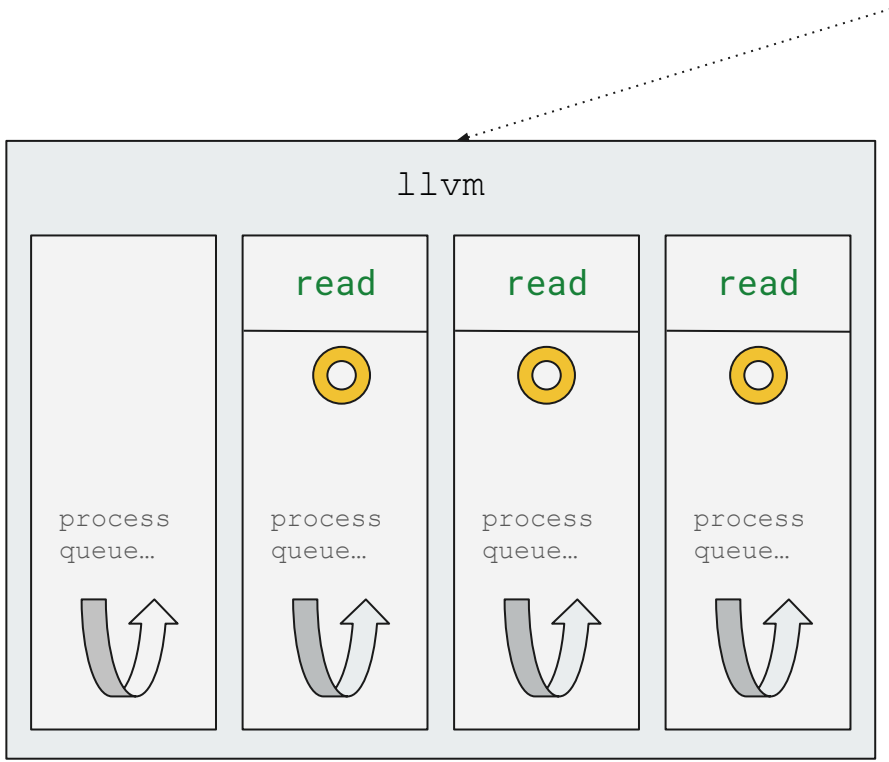
- Modern compilers/linkers use thread pools
  - As modern as … 2008: GNU gold linker
- Tasks are short-lived, `read` and `write` are pure latency

How LLVM deals with this*

**subject to change as slide is presented*

- Coarser granularity
- Start many threads
- Block them on `read` for a token
- Only then enter the worker loop

- But: how many threads do you start?
- But: ~ busy wait in blocked threads?



https://github.com/spack/spack

Jobserver wrongly assumes that it will be able to read tokens for all available jobs on setup #170184

Open

mgorny opened on Dec 1, 2025 — Member

llvm-project/llvm/lib/Support/Unix/Jobserver.inc
Lines 42 to 44 in 0ff0f52

```
42    /// For FIFO-based jobservers, it opens the named pipe. After setup, it drains
43    /// all available tokens from the jobserver to determine the total number of
44    /// available jobs (`NumJobs`), then immediately releases them back.
```

llvm-project/llvm/lib/Support/Unix/Jobserver.inc
Lines 93 to 103 in 0ff0f52

```
93    // Determine the total number of jobs by acquiring all available slots and
94    // then immediately releasing them.
95    SmallVector<JobSlot, 8> Slots;
96    while (true) {
97      auto S = tryAcquire();
98      if (!S.isValid())
99        break;
100     Slots.push_back(std::move(S));
101   }
102   NumJobs = Slots.size();
103   assert(NumJobs >= 1 && "Invalid number of jobs");
```

This wrongly assumes that all tokens will be immediately available at setup time. However, it is entirely likely that the jobserver will be performing some tasks already, and therefore not all tokens will be immediately available, resulting in wrong estimation. In fact, it is entirely possible that all job tokens will be consumed at the time and therefore `NumJobs` will become 0 and the assertion will fail.

Assignees
yxsamliu

Labels
llvm:support

Type
No type

Projects
No projects

Milestone
No milestone

Relationships
None yet

Development
Code wit
No branches or pull r

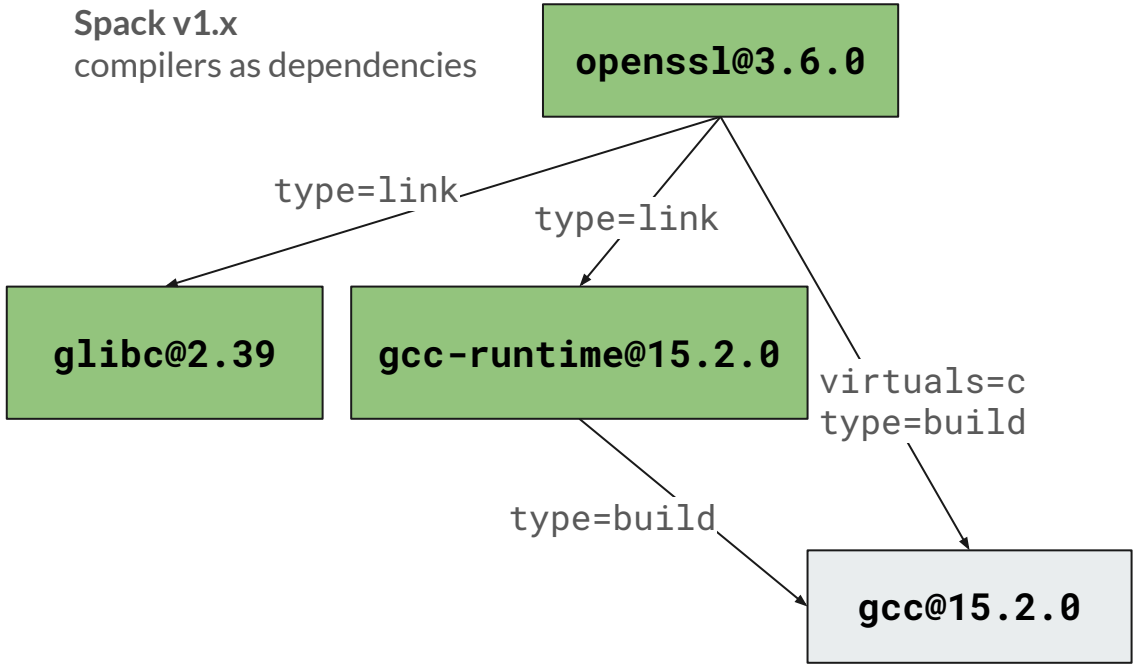
Notifications
You're receiving notifi
subscribed to this thr

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

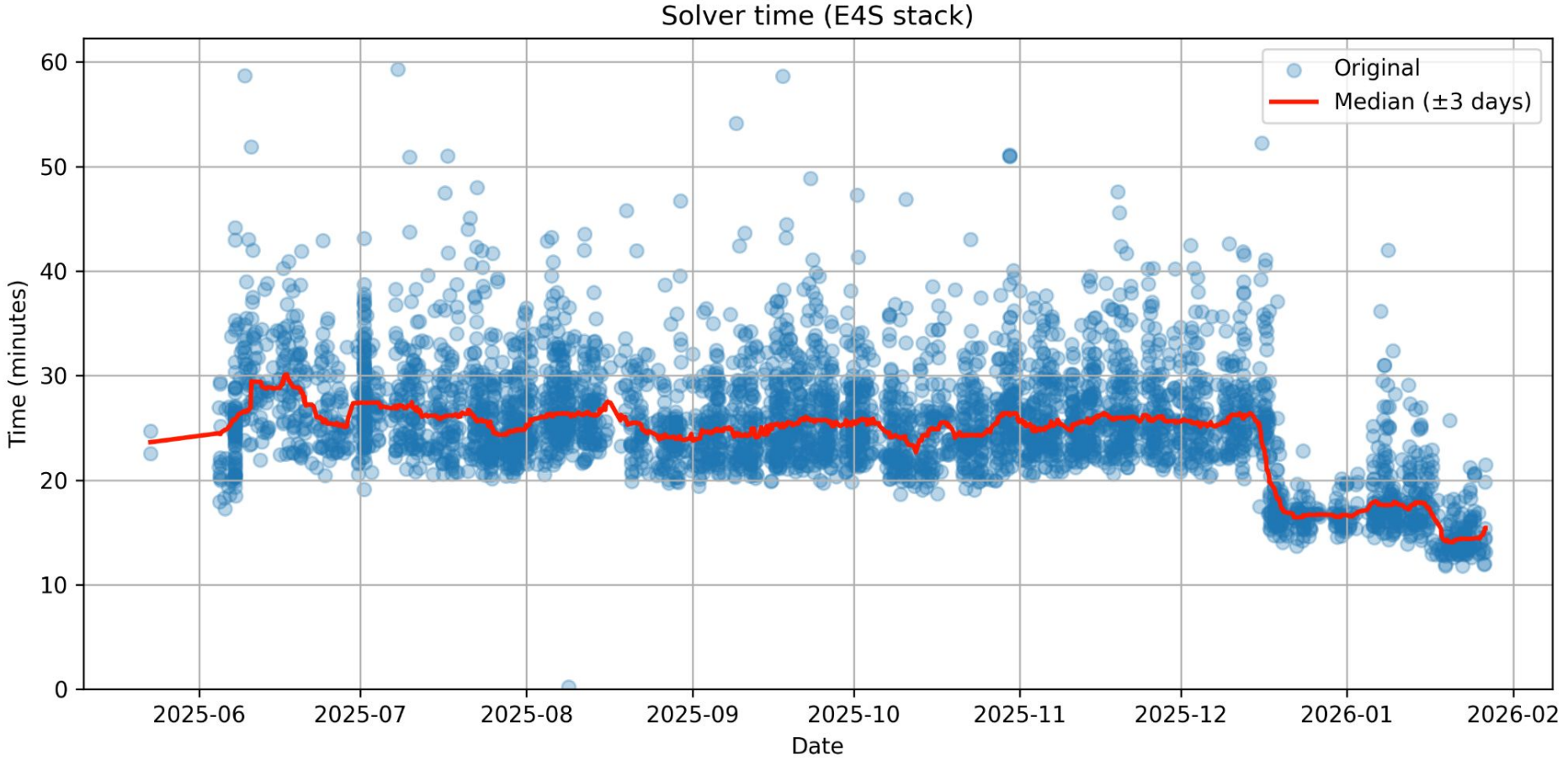
- We've added new syntax: `%c,cxx=llvm@22 %fortran=gcc@15`
  - Depend on LLVM for the `c` and `cxx` language/virtual
  - Depend on GCC for the `fortran` language/virtual
- E.g. `conflicts("%cxx=gcc@14:")`

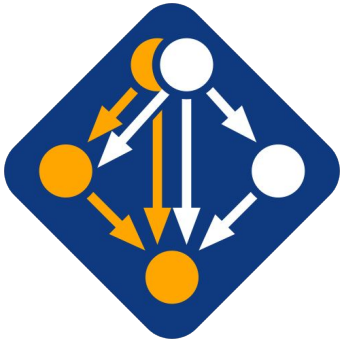
- On the command line, configured toolchains can be used to shorten expressions: `%clang_with_gfortran`

Stability and performance

Recent performance improvements

- `clingo` solver binaries with PGO, LTO, `mimalloc` (old news)
- Benchmarking and regression tracking
  - Easy now, against fixed `spack/spack-packages` commit
- "Dimension reduction" in problem encoding
- Lazy package metadata evaluation
- Immutable abstract `Spec`s & reduced allocations.
- `python3.15 -m profiling.sampling run --heatmap ...`
- `sys.monitoring` to locate "problematic" instructions in hot paths
  `BUILD_LIST`, `DEREF_CELL`, ...
  See the `heavyops` Python package


Solver time (E4S stack)
Original
Median (±3 days)
Time (minutes)
0
10
20
30
40
50
60
2025-06
2025-07
2025-08
2025-09
2025-10
2025-11
2025-12
2026-01
2026-02
Date



- https://github.com/spack/spack
- https://spack-tutorial.readthedocs.io/en/latest/
- E-mail: harmen@stoppels.ch