Scanning the SWH archive through FUSE

(from 10000 processes)

Martin Kirchgessner
Journée Utilisateurs GRICAD, 6 Nov. 2025

Software Heritage

THE GREAT LIBRARY OF SOURCE CODE

Context

Software Heritage : collect, preserve and share *all* software source code

- **find** and **reference** all software source code
- **preserve and share** all software source code
- **enable analysis** of all software source code

Everything is published under the GPL

CodeCommons

2-years project to create the world's most comprehensive digital commons for code

codecommons.org

Project's infrastructure : CINES

CodeCommons : adding metadata

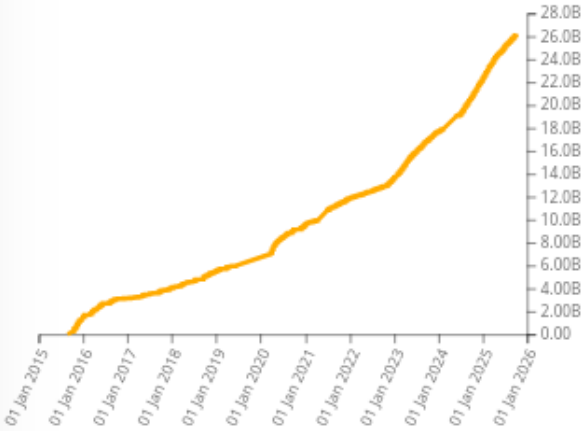
For all archived contents, detect:

- Programming langage
- Licences

A fairly big scan

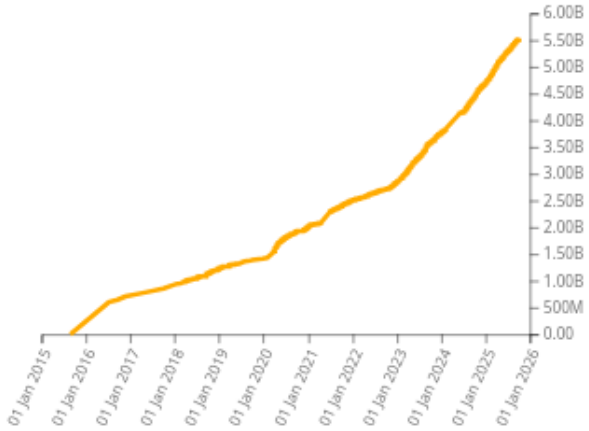
Source files

26 209 482 205



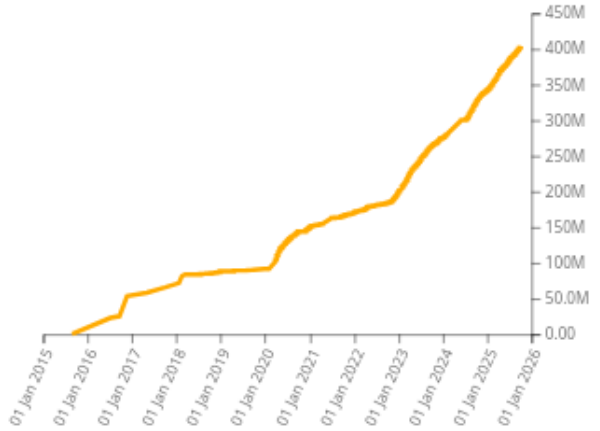
Commits

5 529 028 502



Projects

403 750 300



Directories

20 756 626 383

Authors

99 384 222

Releases

115 750 469

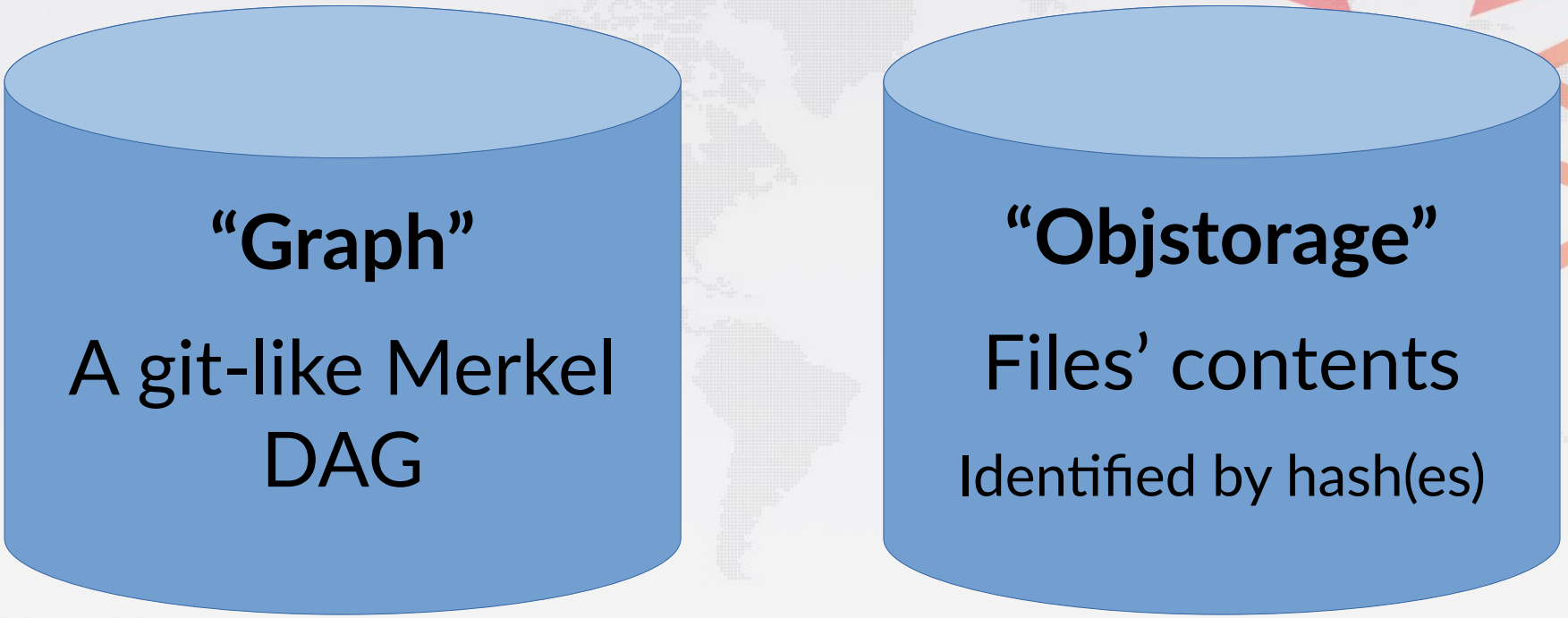
Meet the scanners

- Licences : ScanCode
- Programming langages : Hyperpolyglot

They’re made to scan source trees…

...but that’s not how we archive 😶

How we archive



10.7 TB (compressed)

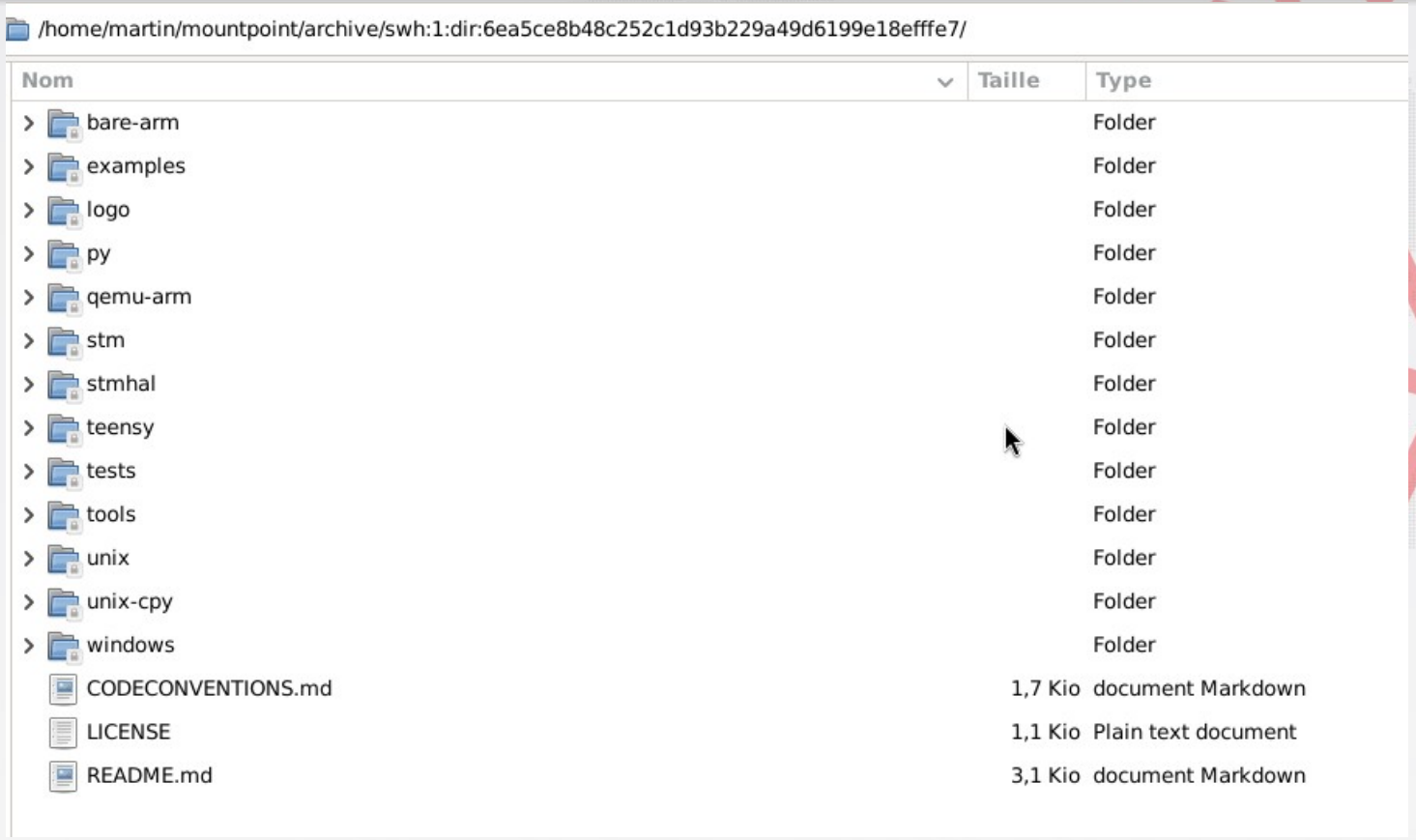
>2.6PB (compressed)

Swh-fuse basics

The whole archive, from any Linux

```
pip install swh-fuse
mkdir ~/mountpoint
swh fs mount ~/mountpoint
```

Navigate by SWHID



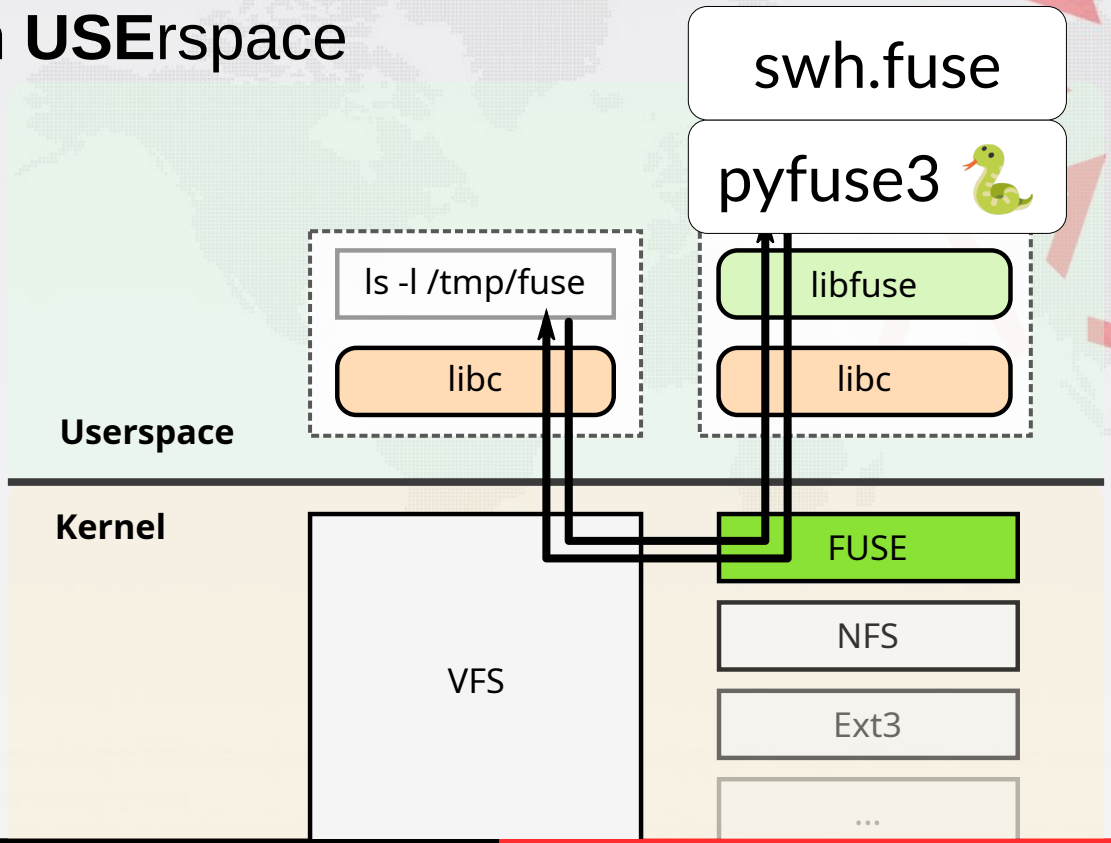
Load only what's read

Scanners don't read everything

→ avoid reconstructing a complete source tree

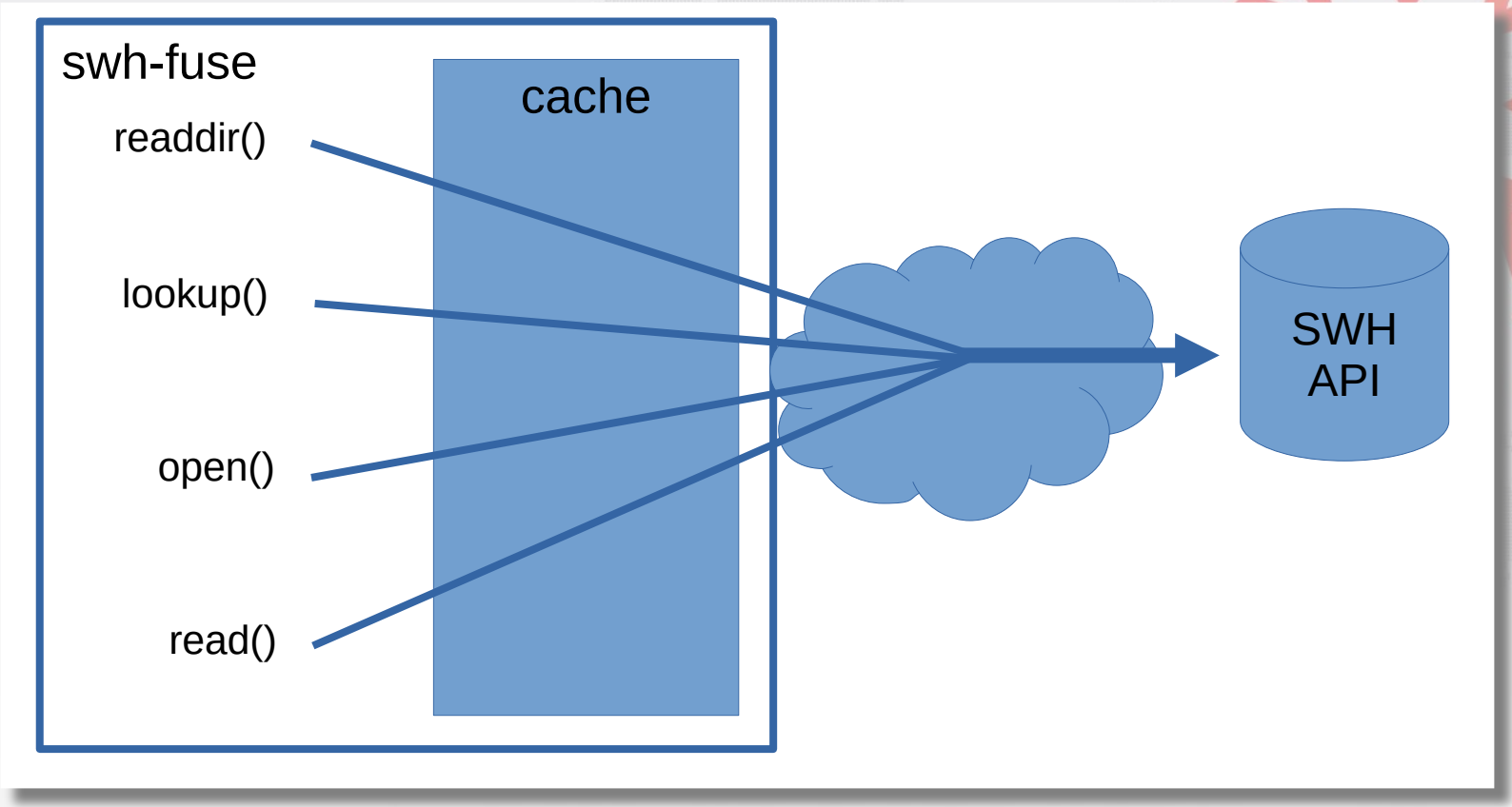
It’s all FUSE

- **F**ilesystem in **USE**rspace



Schema from Wikipedia:FUSE

Everything comes from the Web API

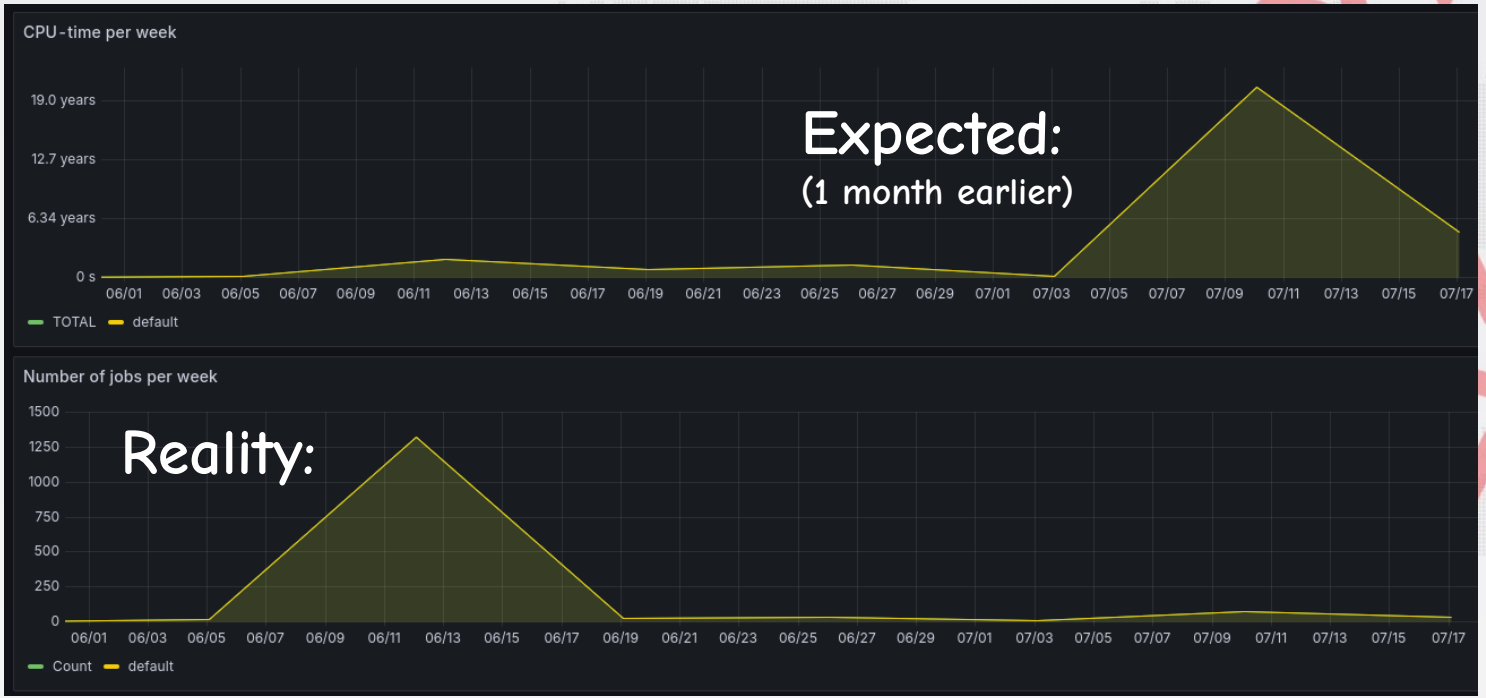


But we have 400m repos to scan

🏭🏭🏭🏭 Go HPC 🏭🏭🏭🏭

- Kraken, then Adastra
- 50-500 nodes with 192 cores / 700GB RAM
- Super-fast network and NFS
  - ⚠️ **We are, unusually, I/O bound**

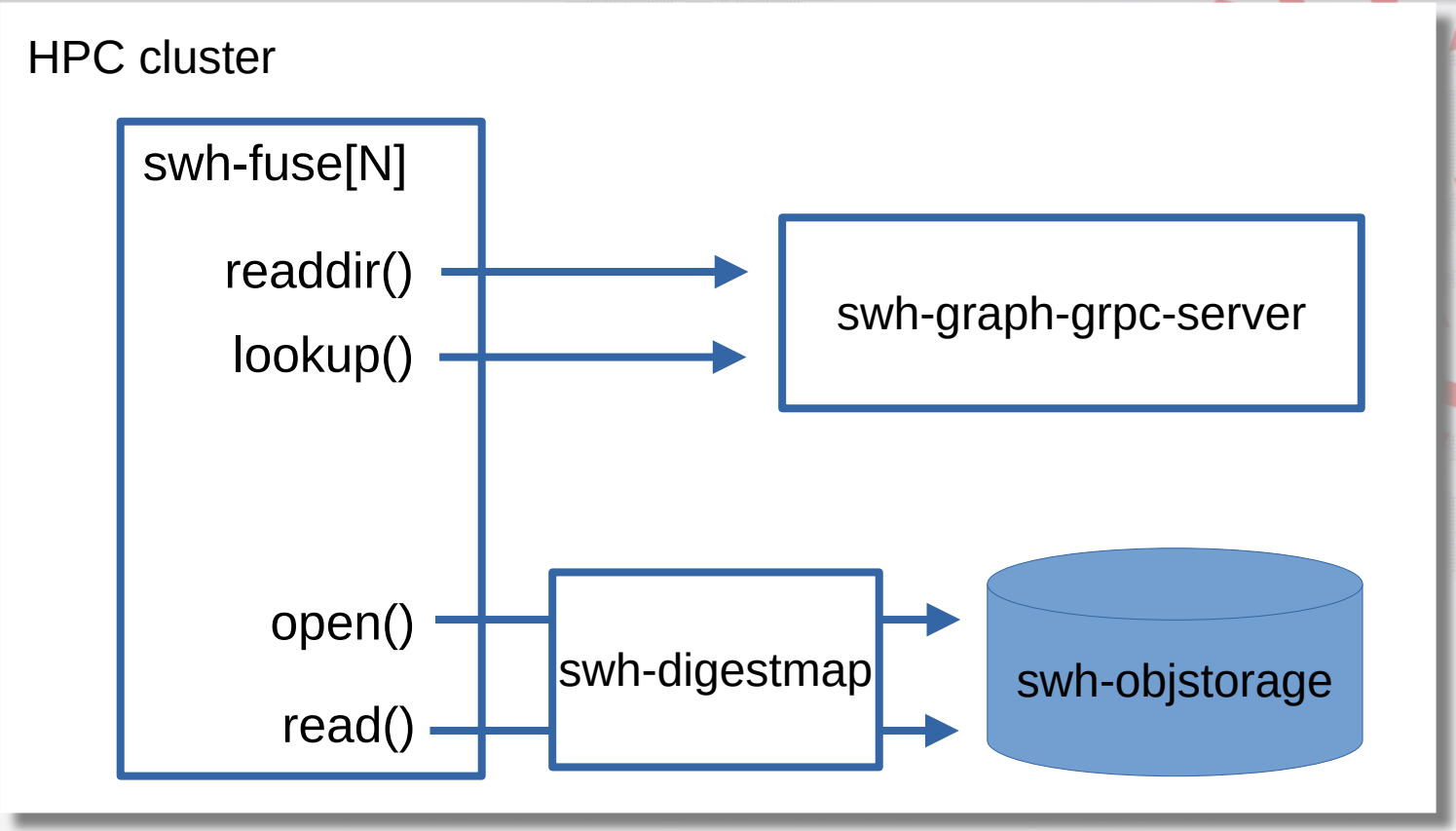
My freeride in 2 graphs



Total 1512 jobs, 26h full-cluster time

Step 1: skip the Web API

Swh-fuse, HPC edition



Matching hashes

- Graph identifies files by SWHID (20 bytes)
- Objstorage identifies by sha1 (20 bytes)
- Dict[bytes, bytes] hardly fits 26b entries

swh.digestmap

- Rust module (PyO3)
  - Based on sux::Vfunc
- ~25µs / get()
  - over 2,3b entries on SSD
  - Averaged over 200 concurrent processes

Step 2 : multi-processing

Dispatch repositories among cores

Small parameters, no I/O with main process…

Use Python's ProcessPoolExecutor

Dispatch repositories among nodes

200 processes on one node ✔️

200 processes * 50 nodes 💥

How to blow up Python startup time

Initial implem:

- ProcessPoolExecutor(max_workers=200)
- Each worker launches
  1) swh-fuse as a (namespaced) sub-process
  2) the scanner

50 machines = 10000 Python starting in parallel

 “stat storm” with Nix/Guix on HPC

A better recipe

1)Mount-unmount once

- Triggers imports

2)ProcessPoolExecutor.map()

→ starts Python once per node 😌

Switching to AppTainer

This venv contains >600 modules

- 📈 Flake install times
- Testing is hard, and harder in Nix

 swhfuse.sif weights 760Mb

Fast enough ?

🚀 Fast enough

In 2 hours, with 50 nodes we scanned:

- 1m repositories with 10000 Hyperpolyglot
- 700k repositories with 16450 Scancodes

Loading Hoyt

2,3b files (from TheStackV2)

- Writing @8000/s, that took 4 days
- 58TB

+ 5TB graph (one node dedicated to graph-server)

+ 113GB digestmap (`cp` to /var/tmp @ 1GB/s)

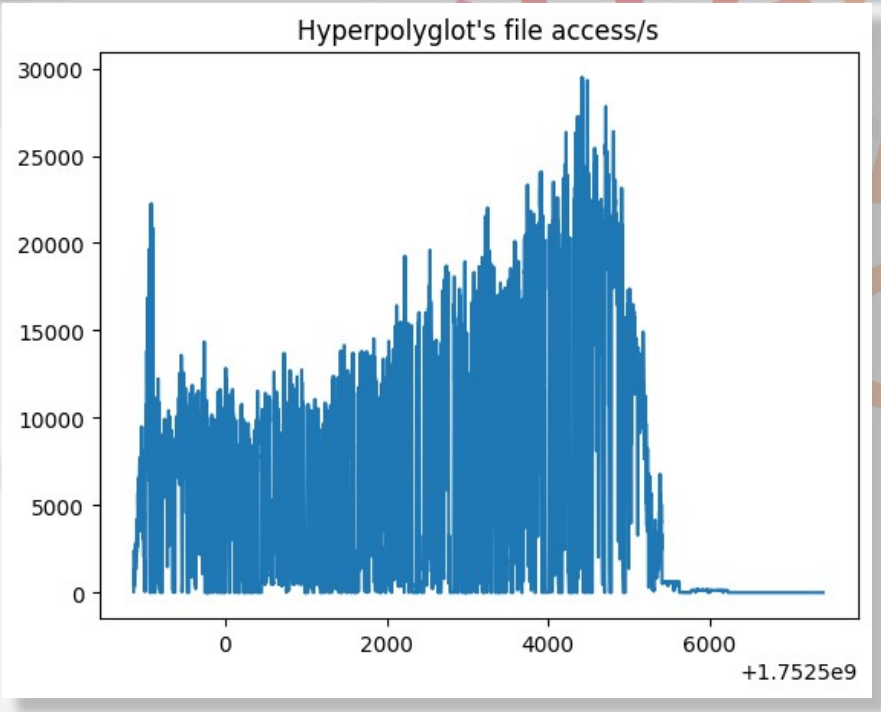
Reading from Hoyt

Peak @30k file read/s

- Constrained by metadata servers, probably

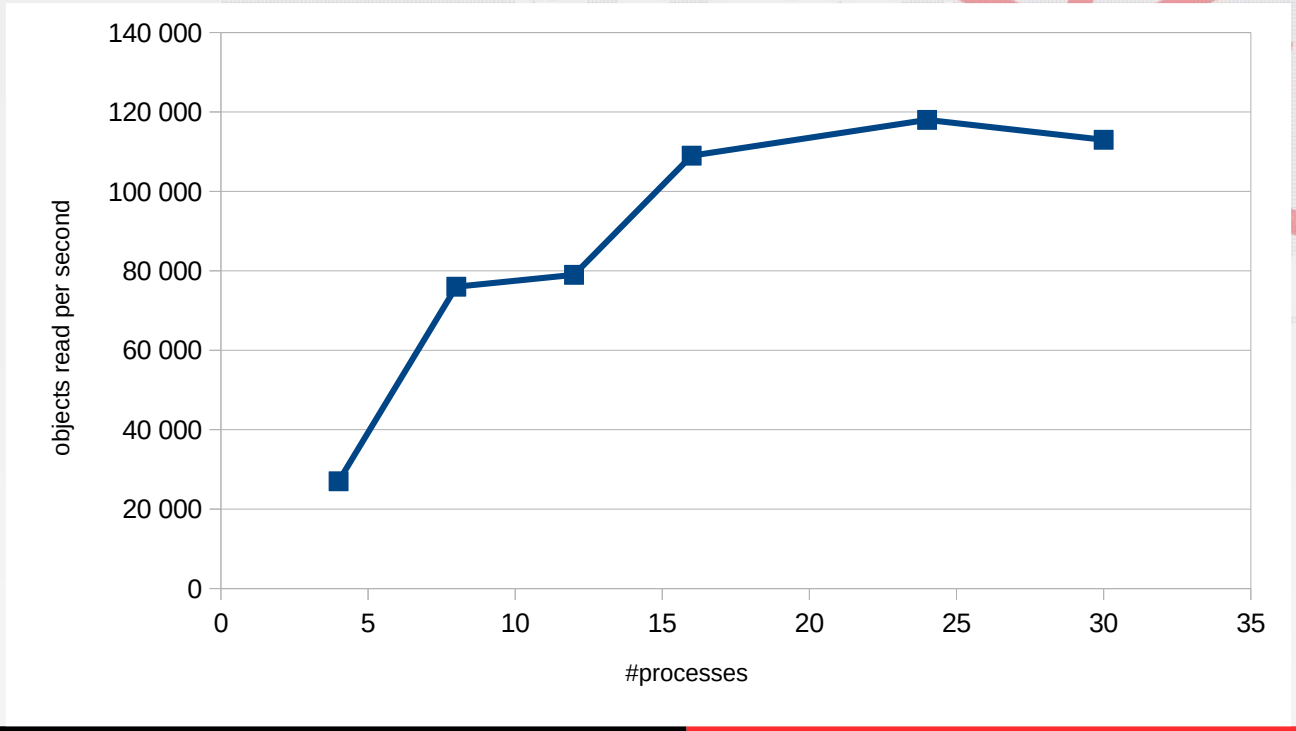
Compare to: Amazon S3

- 2,3b downloads @15k download/s



Reading from Hoyt BONUS

Reading randomly in an swh.shard (385GB) containing 108m files, compressed



Conclusion

User testimonials

“c'est plus rapide que je pensais”

Victor Gambier (gambierv-ext)

- Scanning >50k repos with Coccinelle
  - Using a local graph, but contents from S3
- Still better than Github from G5K

Current work

1) Few file formats good at random lookup
   - Production archive uses a custom swh.shard
   - Ongoing research by Francesco Tosoni (ftosoni-ext)

2) How to store results ?

More details on

docs.softwareheritage.org / API ref / swh.fuse

Or ask now !