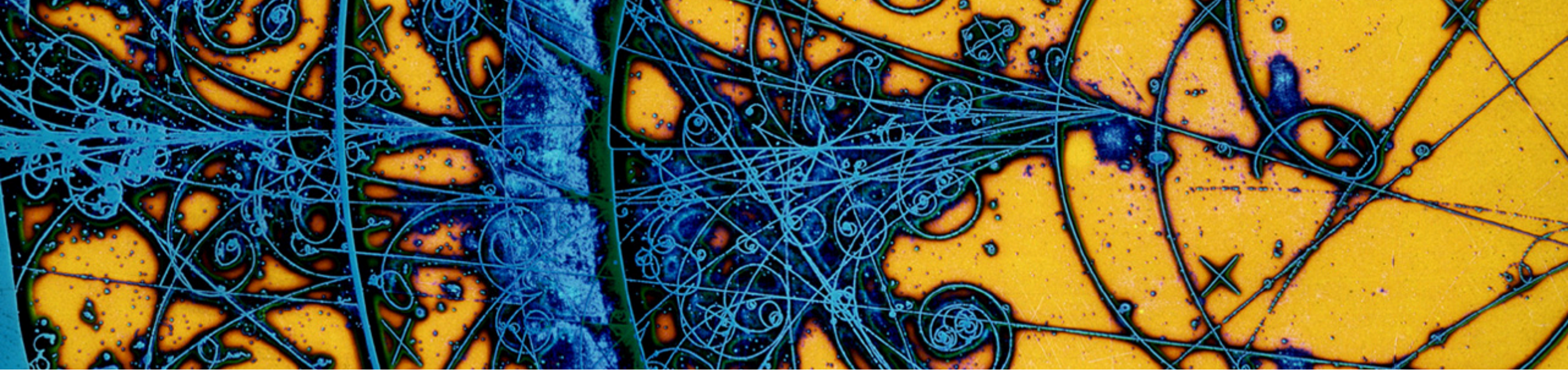


LEAPS EXPERIMENT CONTROL SIG · NOBUGS 2026 · 21 SEPTEMBER 2026

Lightweight Pixi/Conda environments using RattlerVFS and CVMFS

Chris Burr

CERN

Image: CERN-EX-66954B © 1998–2026 CERN

Who am I?

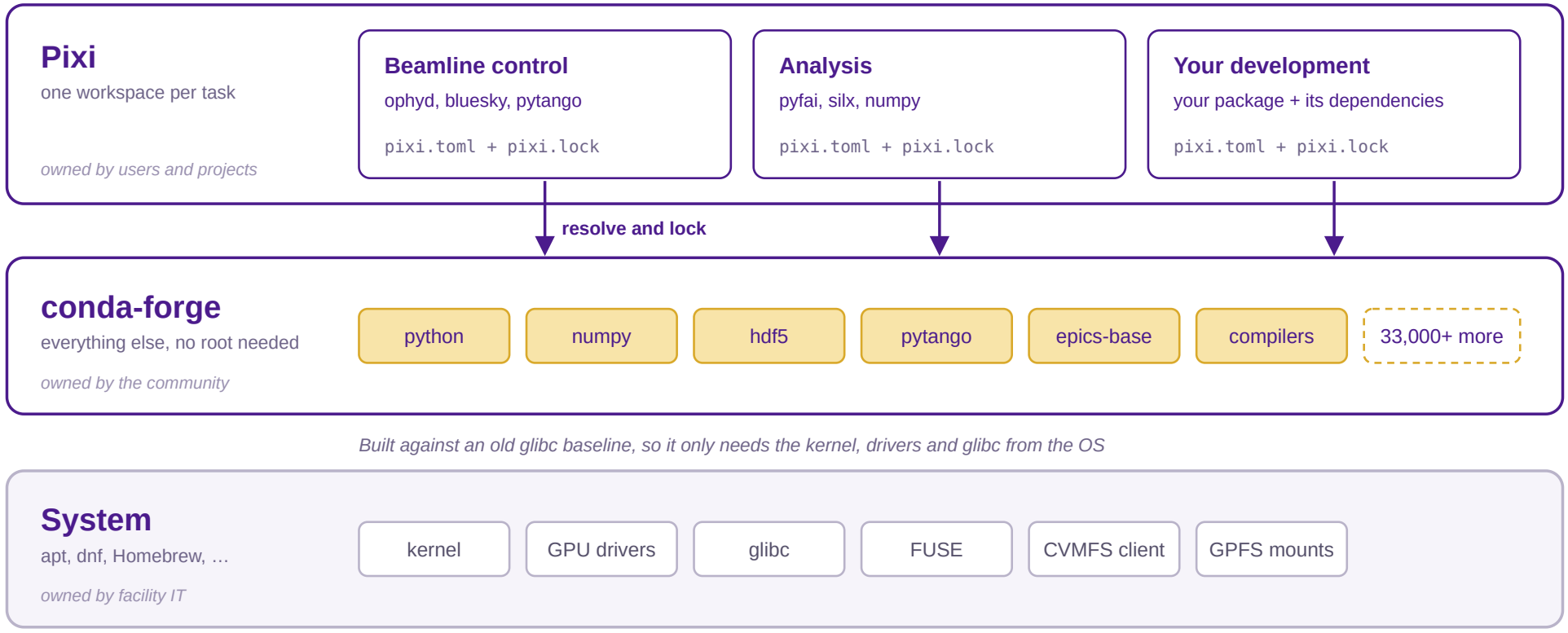
- **Chris Burr**, CERN — I work on software and computing for LHCb
  - PhD in particle physics, but I always had more of a computing/software interest
  - "Primary" job involves grid-computing, also responsible for LHCb's use of CVMFS
- Member of the conda-forge core leadership team
- I help direct the HEP Packaging Coordination community packaging effort

Pixi and conda-forge in 60 seconds

- conda-forge is community-run build and distribution infrastructure
  - ~7800 contributors, 33,000+ packages, heavily automated
  - **Not Python specific** — C/C++, Fortran, Rust, Java, Julia, R, Perl, …
  - **Your ecosystem is already on it:** bluesky, ophyd, pytango, pyfai, silx, dials, tomopy, nexusformat, hdf5plugin
- Pixi is the workspace-shaped way to use it*
  - A `pixi.toml` describes the environments you need, a lock file makes them reproducible
  - No root, no modules, and the same environment on a laptop, a beamline PC and a cluster

** Julian Hofer (prefix.dev) is giving the keynote on Wednesday and will give more details.*

Pixi and conda-forge in 60 seconds



The OS provides what touches hardware, conda-forge provides the rest, and Pixi pins a slice of it for each workspace.

Problem: environments are millions of small files

- A conda/pixi environment is a *copy* of every file in every package
  - Fine(ish)* on a local NVMe drive, painful on a shared filesystem, a disaster for batch
  - I'm told EuXFEL sees up to **50 s** loading files from GPFS before a user job starts

** Even with reflink/hardlink tricks you can end up with 100s of GBs of software environments*

Problem: environments are millions of small files

- A conda/pixi environment is a *copy* of every file in every package
  - Fine(ish)* on a local NVMe drive, painful on a shared filesystem, a disaster for batch
  - I'm told EuXFEL sees up to **50 s** loading files from GPFS before a user job starts
- Facilities pre-cook environments centrally, works until someone wants to change something
  - Making your own workspace gives you a whole new environment
  - **You pay the full price of an environment to add one package**

** Even with reflink/hardlink tricks you can end up with 100s of GBs of software environments*

Problem: environments are millions of small files

- A conda/pixi environment is a *copy* of every file in every package
  - Fine(ish)* on a local NVMe drive, painful on a shared filesystem, a disaster for batch
  - I'm told EuXFEL sees up to **50 s** loading files from GPFS before a user job starts
- Facilities pre-cook environments centrally, works until someone wants to change something
  - Making your own workspace gives you a whole new environment
  - **You pay the full price of an environment to add one package**
- This is exactly the same problem I saw in LHCb

** Even with reflink/hardlink tricks you can end up with 100s of GBs of software environments*

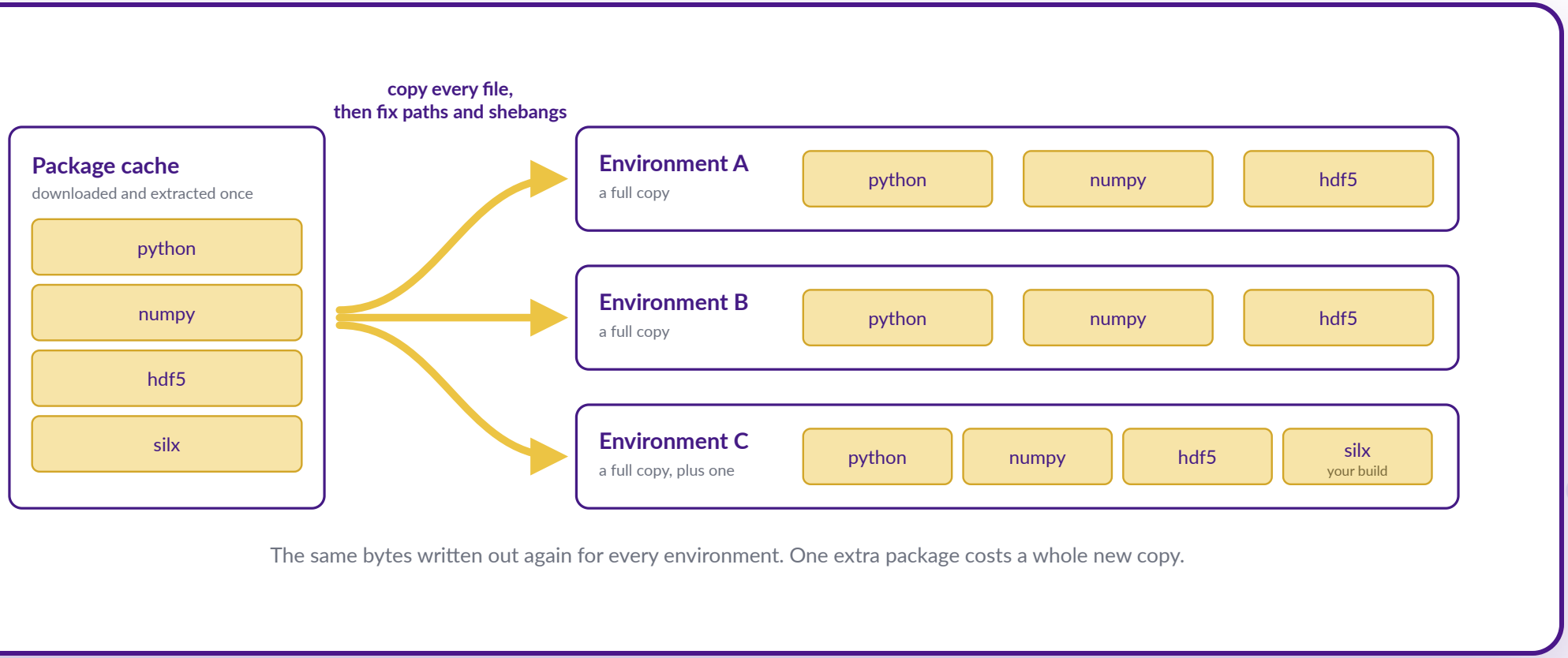
CVMFS in 60 seconds

- A read-only filesystem served over HTTP and aggressively cached
  - Content-addressed and de-duplicated: an identical file is stored and downloaded once
  - Files arrive **on demand** — mount a 100 TB repository, download only what you open
  - It's a Content Delivery Network (CDN) for software
- Several of you already use it to distribute analysis code (ESRF, SOLEIL, ALBA)
- **The bit that matters today:**
  - CVMFS is good at making a huge read-only tree appear on every machine
  - The global infrastructure is already there

What does "installing" actually mean?

- Installing a conda package is three steps:
  1. Download and extract the package to the local cache
  2. "Copy" the files to the install location
  3. Apply any necessary fixes (shebangs, hard coded paths, Python stuff, ...)
- **Steps 2 and 3 are the expensive bit** — disk, inodes and IO, paid again for *every* environment

What does "installing" actually mean?



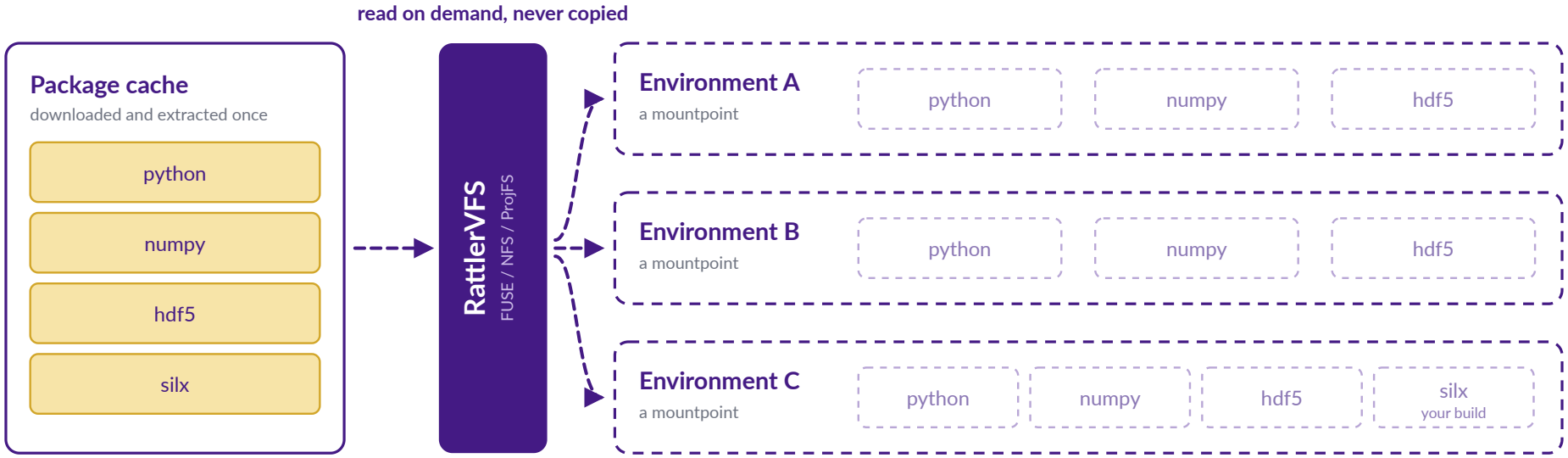
The same bytes written out again for every environment. One extra package costs a whole new copy.

RattlerVFS: make steps 2 and 3 virtual

- A virtual filesystem which implements steps 2 and 3, on demand
  - No copying, no disk usage, no IO overhead
  - An environment stops being a directory tree and becomes a mountpoint
- Works on Linux/macOS/Windows using FUSE/NFS/ProjFS*

** Not all backends work on all operating systems.*

RattlerVFS: make steps 2 and 3 virtual



Nothing is copied. Files are proxied out of the cache and fixed up as they are read, so an environment costs no disk and no IO.

* Not all backends work on all operating systems.

Demo: RattlerVFS in action

- The RattlerVFS underpinnings are in good shape conda/rattler#2566
- I have a proof-of-concept Pixi integration*
- Prebuilt binaries: `chrisburr/pixi-rattlerfs`
  - A rebranded build of my pixi fork, so it can live beside a real `pixi`

```
$ pixi-rattlerfs run python -c "import numpy; print(numpy.__file__)"
```

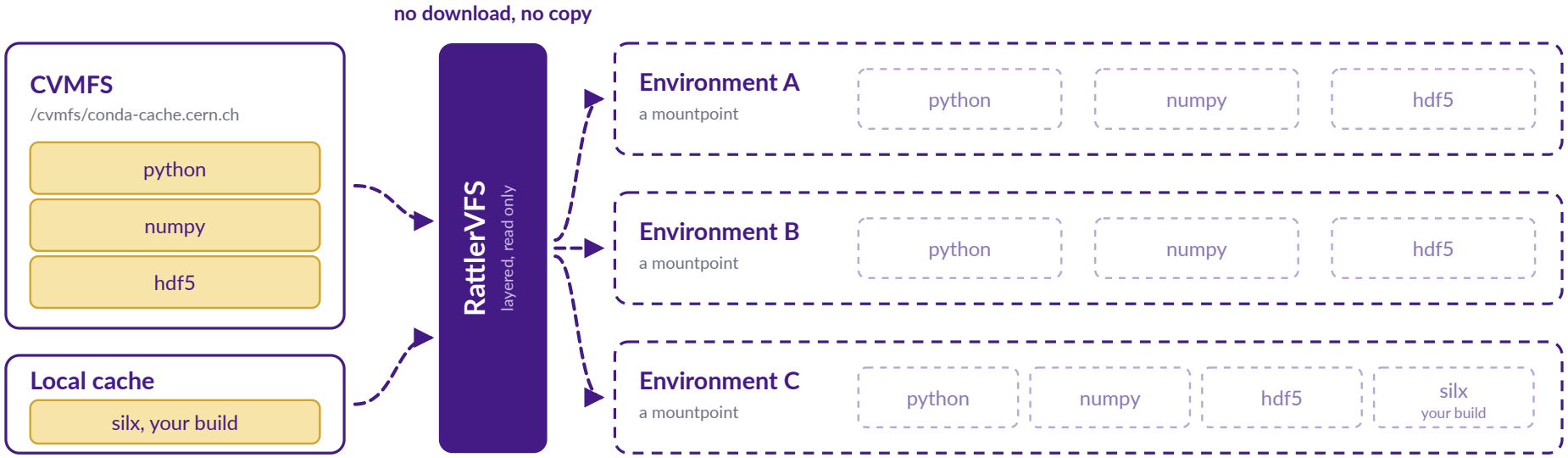
** This was a personal prototype and is in no way endorsed by the prefix.dev team!*

One step further with CVMFS

> 1. Download and extract the package to the local cache

- We already have a tool for that: CVMFS!
  - `/cvmfs/conda-cache.cern.ch` holds the extracted conda-forge cache
  - Add a custom local layer if needed
  - RattlerVFS proxies it, mutating the data in-flight as needed
- conda environments no longer have IO cost

One step further with CVMFS



One shared, de-duplicated cache for every machine. An environment shares everything except the package that differs.

Problem: cold starts cost round trips, not bytes

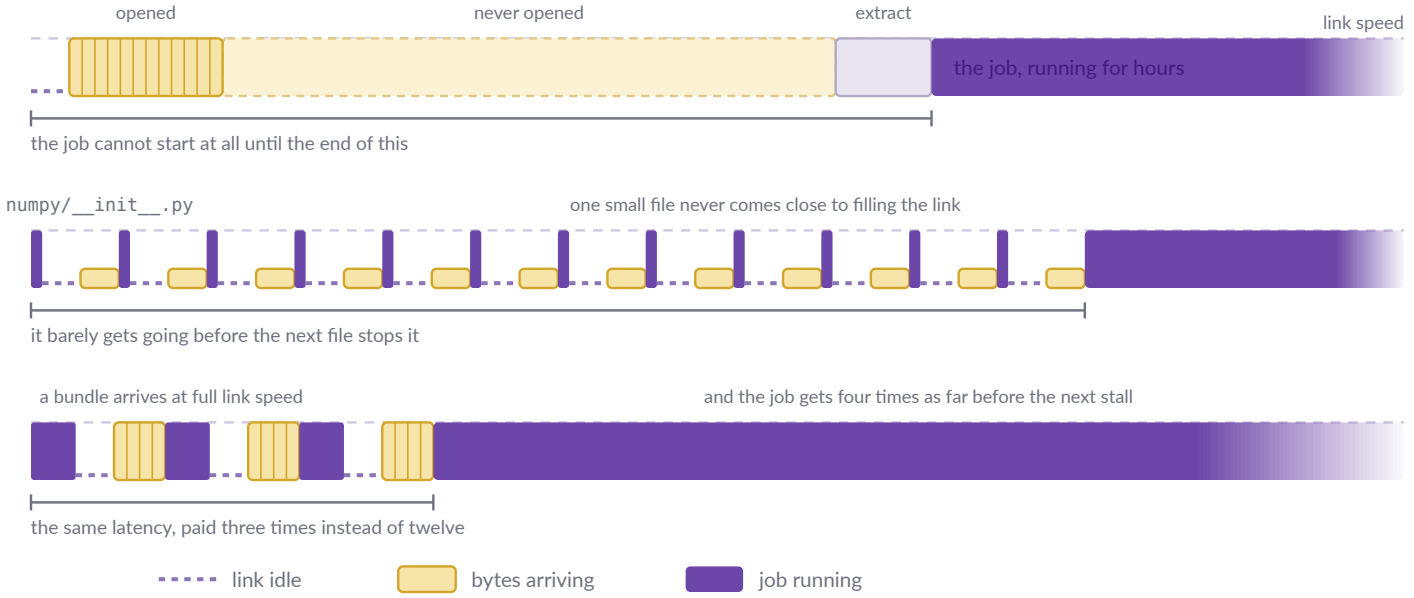
- CVMFS is better than downloading
  - **Content addressed downloads:** deduplicate at the file level, not the package level
  - **Download files on-demand:** don't pay for files in an environment you don't use
- But downloading at the file level makes you sensitive to latency...
- CVMFS's new "file bundle" feature fixes this (see CHEP 2026)
  - Can use heuristics to know files that will be used together (e.g. Python static analysis)
  - Proof of concept: chrisburr/cvmfs-filebundle-gen

Problem: cold starts cost round trips, not bytes

Download the package
nothing runs until all of it has landed

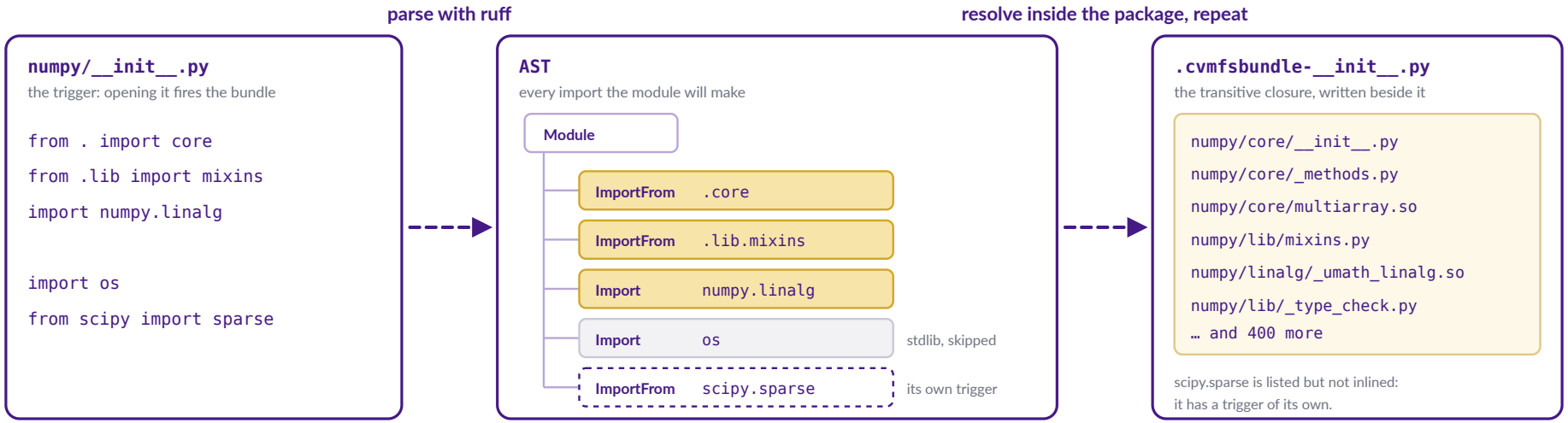
Fetch each file
runs at once, then stalls on every file

Fetch bundles
runs at once, and gets further each time



Height is the share of the link in use. The job amortises its startup, but every job pays it again.

Problem: cold starts cost round trips, not bytes



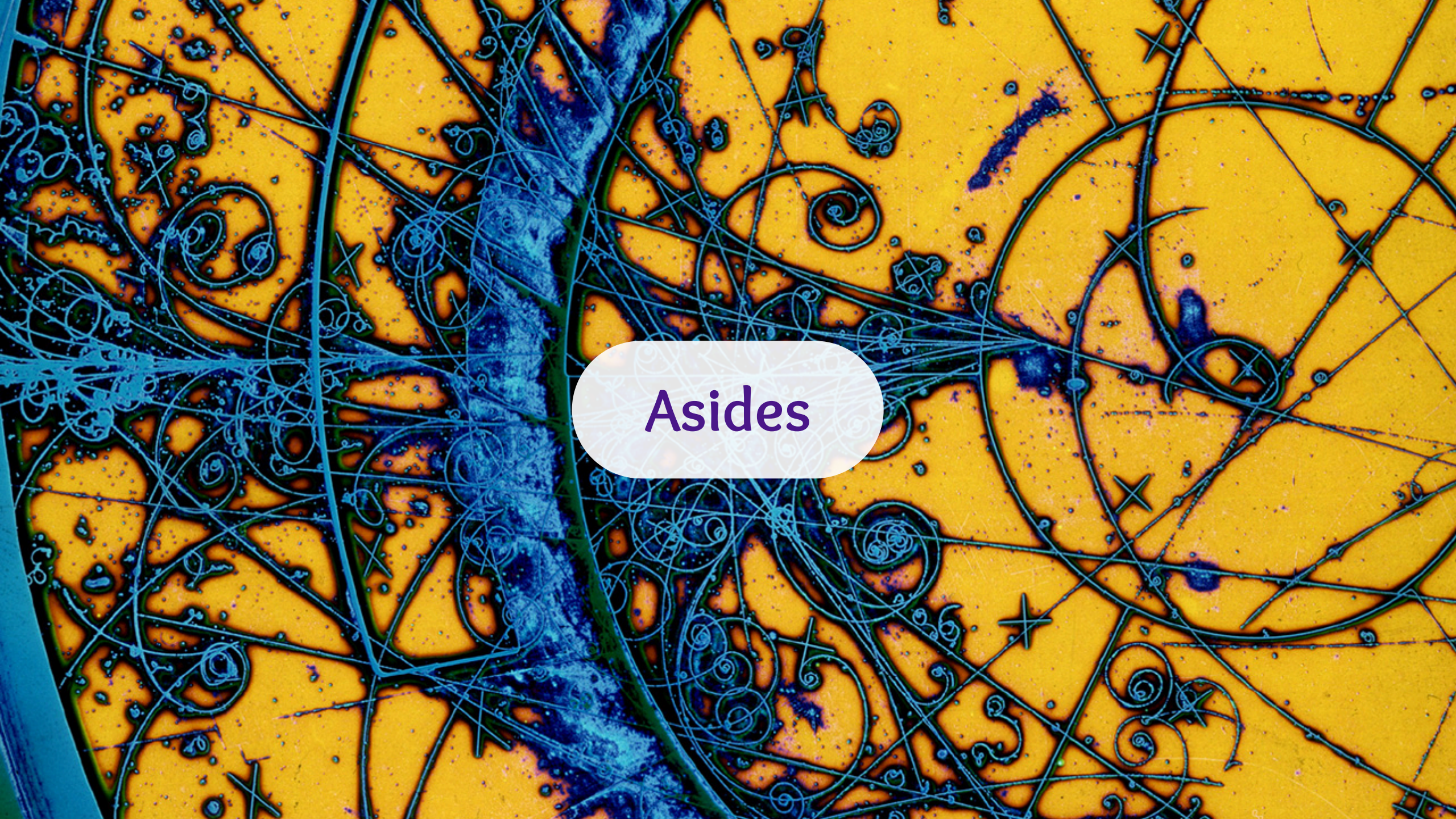
One open fetches everything the import will read, in a single parallel sweep.

What's left to do

- **Upstreaming:**
  - this has to land in `rattler`
  - the `pixi` side of things needs to be designed/implemented
- **CVMFS tuning and set up:**
  - file bundles and catalogue layout for this access pattern
  - how to host a common repository (like unpacked.cern.ch?)
- **Integration:** with your facilities and tools?
- **Extensions:** PyPI wheels? direct libcvmfs?

What's left to do with whom?

- **Upstreaming:** with prefix.dev
  - this has to land in `rattler`
  - the `pixi` side of things needs to be designed/implemented
- **CVMFS tuning and set up:** with CERN and conda-forge
  - file bundles and catalogue layout for this access pattern
  - how to host a common repository (like unpacked.cern.ch?)
- Integration with your facilities and tools?
- **Extensions:** PyPI wheels? direct libcvmfs? with prefix.dev

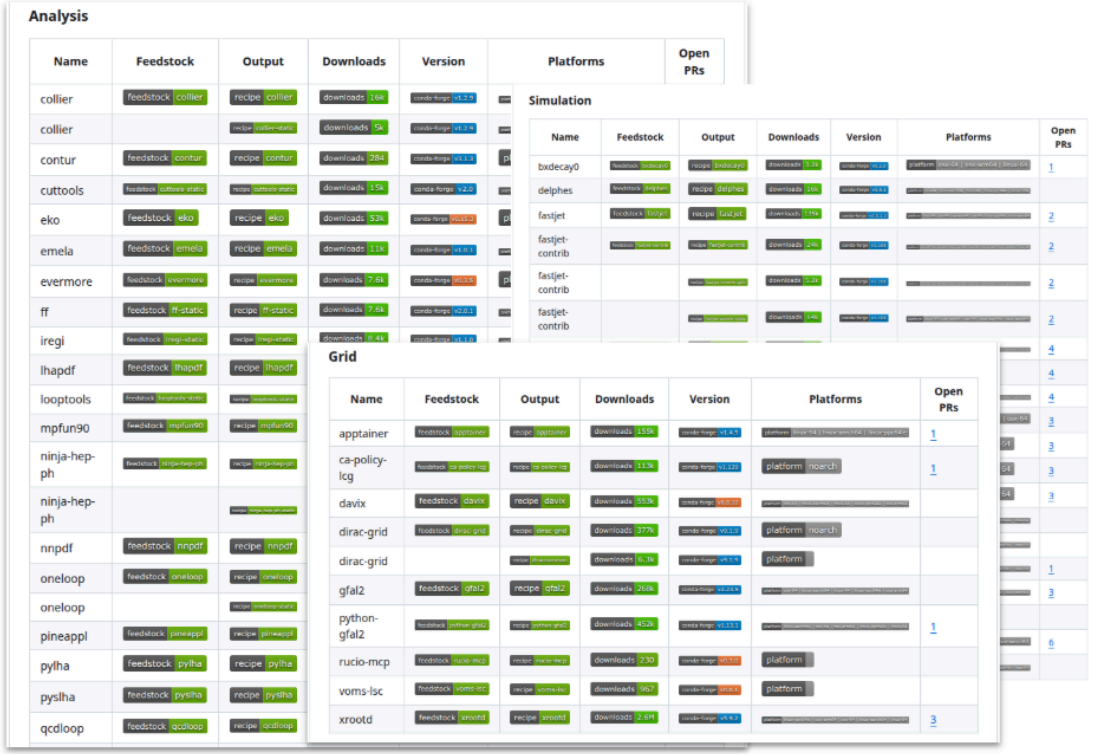


Asides

HEP Packaging Coordination

- Maintaining builds requires work and sometimes expertise
- conda-forge is the baseline but community projects are useful for specific applications
- I'd recommend keeping within the conda-forge channel by default

- HEP Packaging Coordination
  - **120+ packages:** ROOT, Pythia8, FastJet, Rivet, …
  - ATLAS, Belle II, CMS, DIRAC, LHCb, SHiP, Scikit-HEP, …



hep-packaging-coordination

Long term reproducibility and maintenance

- In LHCb we have the need to:
  - have stability during a year of data-taking without being frozen
  - keep maintaining that software and its dependencies indefinitely

Long term reproducibility and maintenance

- In LHCb we have the need to:
  - have stability during a year of data-taking without being frozen
  - keep maintaining that software and its dependencies indefinitely
- For example:
  1. Start the year with a frozen baseline
  2. During the year: Need to make minimal changes to the baseline
  3. End of year: Continue indefinitely while still making minor patches as needed

Long term reproducibility and maintenance

- In LHCb we have the need to:
  - have stability during a year of data-taking without being frozen
  - keep maintaining that software and its dependencies indefinitely
- For example:
  1. Start the year with a frozen baseline
  2. During the year: Need to make minimal changes to the baseline
  3. End of year: Continue indefinitely while still making minor patches as needed
- This doesn't match conda-forge's rolling release model
  - Tooling to support this would be straightforward
  - **Do you have a similar need?**

Summary

- Conda-forge+pixi can provide everything that is needed in userspace

Summary

- Conda-forge+pixi can provide everything that is needed in userspace
- **RattlerVFS makes an environment a mountpoint instead of a copy**
  - Point it at a conda-forge cache on CVMFS: no download, no copy, no IO
  - A user can add their one package without giving up the shared install

Summary

- Conda-forge+pixi can provide everything that is needed in userspace
- **RattlerVFS makes an environment a mountpoint instead of a copy**
  - Point it at a conda-forge cache on CVMFS: no download, no copy, no IO
  - A user can add their one package without giving up the shared install
- The pieces exist today: `rattler_vfs`, a Pixi prototype, a cache on CVMFS at CERN
  - What's left is upstreaming, CVMFS tuning, and integrating into your workflows

Summary

- Conda-forge+pixi can provide everything that is needed in userspace
- **RattlerVFS makes an environment a mountpoint instead of a copy**
  - Point it at a conda-forge cache on CVMFS: no download, no copy, no IO
  - A user can add their one package without giving up the shared install
- The pieces exist today: `rattler_vfs`, a Pixi prototype, a cache on CVMFS at CERN
  - What's left is upstreaming, CVMFS tuning, and integrating into your workflows
- **Mostly I just want this to exist**
  - Happy to lead parts of it, happy for someone else to lead parts of it
  - If it fits the LEAPS-Infratech call, even better

Questions?