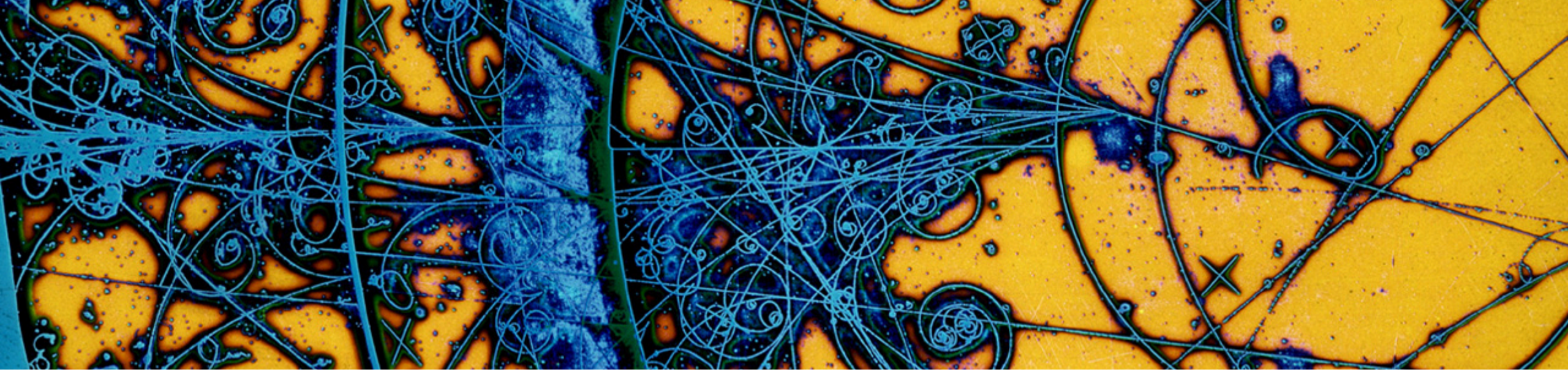
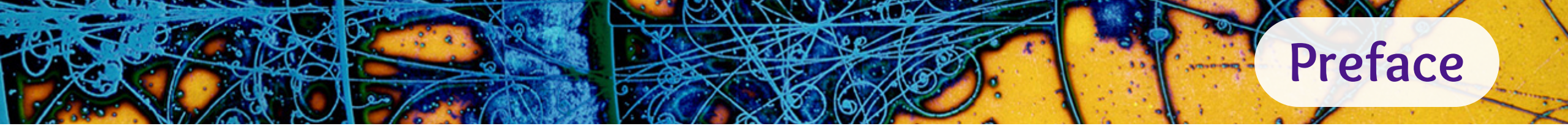


HSF SEMINAR — SOFTWARE PACKAGING · 24 JUNE 2026

Packaging and distributing the HEP ecosystem on conda-forge

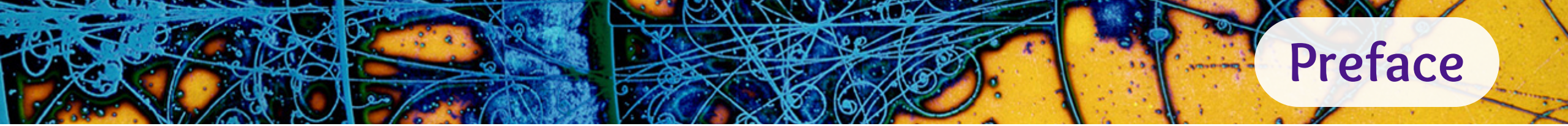
Chris Burr · Matthew Feickert

CERN · University of Wisconsin–Madison



Preface

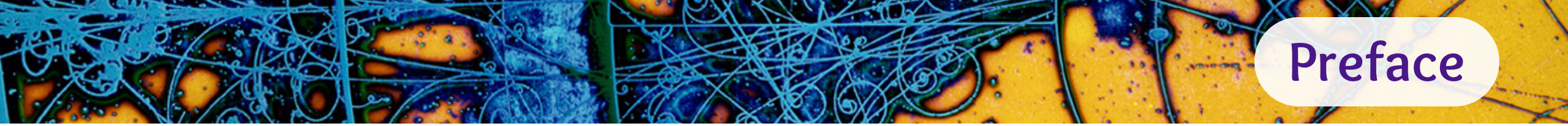
- Software packaging is a complex topic and this is a short talk



Preface

- Software packaging is a complex topic and this is a short talk
- For the purpose of this talk:

> Software packaging is a means to get software in a reproducible and reliable way



Preface

- Software packaging is a complex topic and this is a short talk
- For the purpose of this talk:

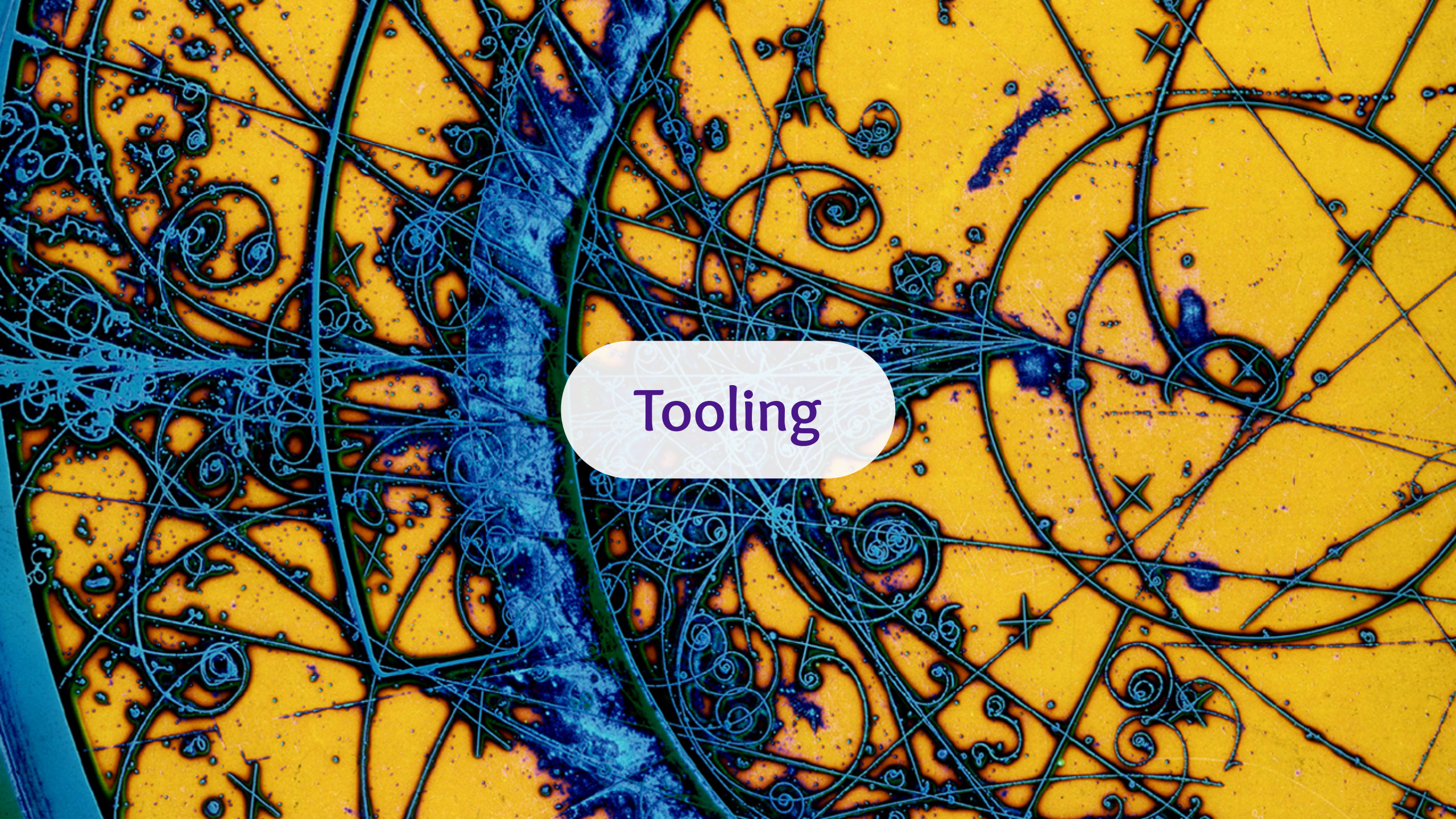
> Software packaging is a means to get software in a reproducible and reliable way

- Will break this down into three parts:

🛠️ **Tooling**
install & manage environments

📦 **Packages**
recipes & binaries

👷 **Development**
build & iterate


Tooling

What is the conda **ecosystem**?

1. Installer tools for software packages and their dependencies:
   - `conda create ...`
   - `pixi init`

What is the conda **ecosystem**?

1. Installer tools for software packages and their dependencies:
   - ➤ `conda create ...`
   - ➤ `pixi init`

2. A channel for distributing software packages: `conda-forge`

What is the conda **ecosystem**?

1. Installer tools for software packages and their dependencies:
   - ➤ `conda create ...`
   - ➤ `pixi init`

2. A channel for distributing software packages: `conda-forge`

3. A file format for binary packages: `.conda`

What is the conda **ecosystem**?

1. Installer tools for software packages and their dependencies:
   - `conda create ...`
   - `pixi init`

2. A channel for distributing software packages: `conda-forge`

3. A file format for binary packages: `.conda`

4. Tools for building packages:
   - `conda-build`
   - `rattler-build`

What is the conda ecosystem?

1. Installer tools for software packages and their dependencies:
   - `conda create ...`
   - `pixi init`

2. A channel for distributing software packages: `conda-forge`

3. A file format for binary packages: `.conda`

4. Tools for building packages:
   - `conda-build`
   - `rattler-build`

5. Distributed cyberinfrastructure for building conda packages coherently:
   https://github.com/conda-forge/

What is the conda ecosystem **not**?

1. An operating system-wide package manager (like `apt` or `yum`)
   - ➤ In the context of HSF use cases these aren't very interesting

What is the conda ecosystem **not**?

1. An operating system-wide package manager (like `apt` or `yum`)
   - ➤ In the context of HSF use cases these aren't very interesting

2. A language-specific package manager (like `pip`)
   - ➤ conda packaging format is **language-agnostic**
   - ➤ Can package C/C++, Fortran, Rust, Python, Go, R ...

What is the conda ecosystem **not**?

1. An operating system-wide package manager (like `apt` or `yum`)
   - ➤ In the context of HSF use cases these aren't very interesting

2. A language-specific package manager (like `pip`)
   - ➤ conda packaging format is **language-agnostic**
   - ➤ Can package C/C++, Fortran, Rust, Python, Go, R ...

3. Anaconda, Inc. (the company)
   - ➤ conda is a community project with a varied elected steering council
   - ➤ conda-forge packages are **free and open source**
   - ➤ Anaconda supports conda-forge, but is only a small fraction of the community
   - ➤ Anaconda's paid offerings are not interesting to HEP users

What is the conda ecosystem **not**?

1. An operating system-wide package manager (like `apt` or `yum`)
   - ➤ In the context of HSF use cases these aren't very interesting

2. A language-specific package manager (like `pip`)
   - ➤ conda packaging format is **language-agnostic**
   - ➤ Can package C/C++, Fortran, Rust, Python, Go, R ...

3. Anaconda, Inc. (the company)
   - ➤ conda is a community project with a varied elected steering council
   - ➤ conda-forge packages are **free and open source**
   - ➤ Anaconda supports conda-forge, but is only a small fraction of the community
   - ➤ Anaconda's paid offerings are not interesting to HEP users

4. Something new

It's been a while...

> Looking back at your PyHEP 2019 conda-forge talk (interesting to see how much has stayed the same over 6+ years)

Packaging for Python and Beyond — PyHEP 2019

What has changed?

- Compiler toolchains are now much more mature
  - Originally hard to use outside of conda builds
  - Now they're very mature and well maintained, including CUDA

What has changed?

- Compiler toolchains are now much more mature
  - Originally hard to use outside of conda builds
  - Now they're very mature and well maintained, including CUDA
- Tooling is much faster
  - Used to advertise getting ROOT in under 5 minutes
  - Now it can be **~10 seconds** 🚀

```
~> time pixi exec root -l -b -q -e '1+1'
(int) 2

________________________________________________________
Executed in   13.64 secs
```

What has changed?

- Compiler toolchains are now much more mature
  - Originally hard to use outside of conda builds
  - Now they're very mature and well maintained, including CUDA

- Tooling is much faster
  - Used to advertise getting ROOT in under 5 minutes
  - Now it can be **~10 seconds** 🚀

```
~> time pixi exec root -l -b -q -e '1+1'
(int) 2

------------------------------------------------------
Executed in   13.64 secs
```

- Pixi provides a lot of "user experience" improvements

What does Pixi give you?

- Workspace model of working
  - Add a `pixi.toml` Pixi manifest to describe the software you need
  - Can contain **multiple environments** and **multiple platforms** for different use cases
  - Can also describe commands to run in the environment ("tasks")

What does Pixi give you?

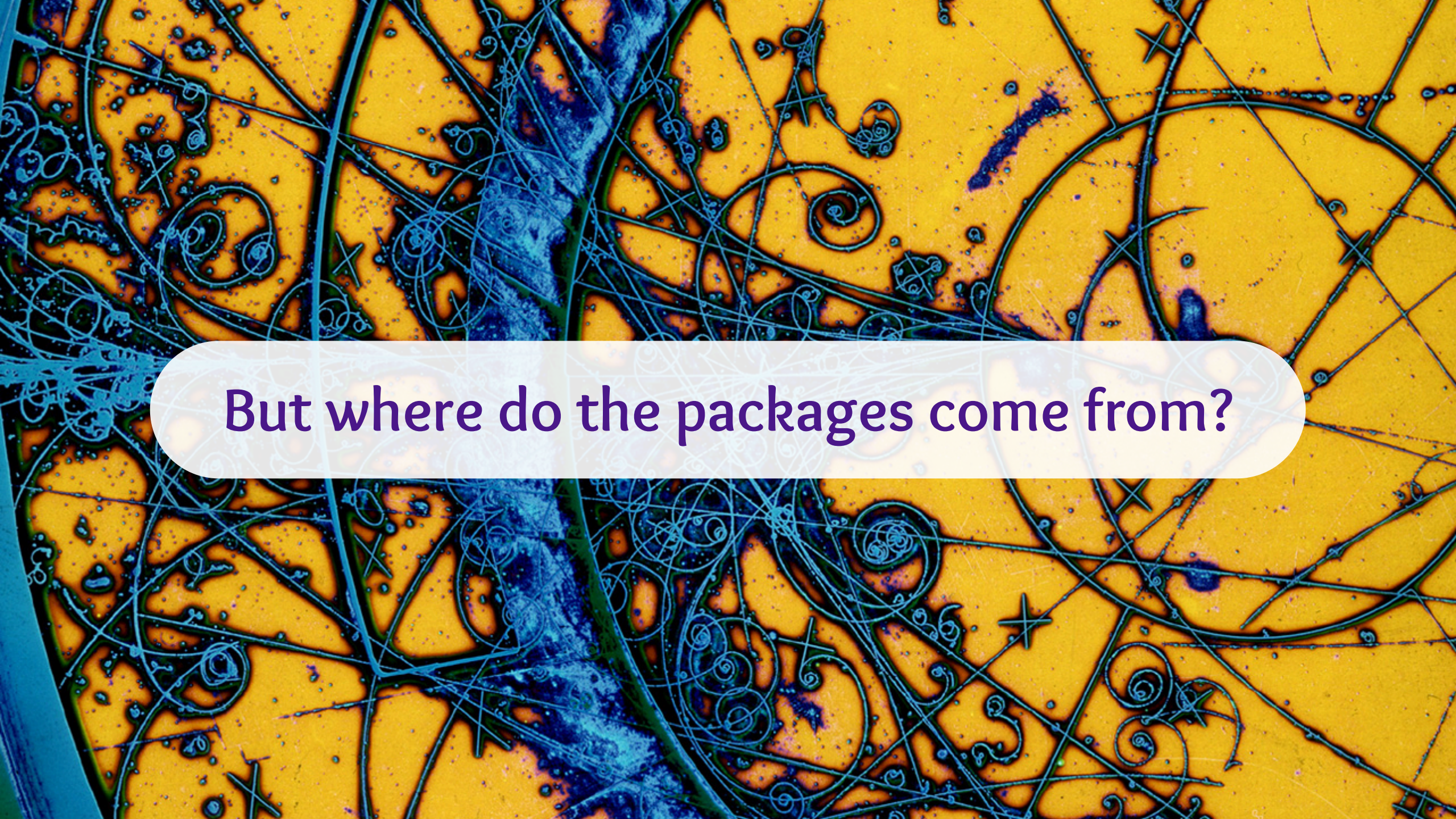
- Workspace model of working
  - Add a `pixi.toml` Pixi manifest to describe the software you need
  - Can contain **multiple environments** and **multiple platforms** for different use cases
  - Can also describe commands to run in the environment ("tasks")
- Pixi then automatically takes care of generating a digest-level lock file
  - Ensures that everyone has the same software installed*
  - This should typically be committed to the repository†

** Assuming the same OS (Linux/macOS) and CPU architecture.*
† Tools like renovate can be used to automatically update the lock file periodically.

What does Pixi give you?

- Workspace model of working
  - Add a `pixi.toml` Pixi manifest to describe the software you need
  - Can contain **multiple environments** and **multiple platforms** for different use cases
  - Can also describe commands to run in the environment ("tasks")
- Pixi then automatically takes care of generating a digest-level lock file
  - Ensures that everyone has the same software installed*
  - This should typically be committed to the repository†
- 🪄 Pixi takes care of managing all of the software environments for you 🪄

** Assuming the same OS (Linux/macOS) and CPU architecture.*
† Tools like renovate can be used to automatically update the lock file periodically.


But where do the packages come from?

Getting packages

- How to build software ("recipes")
- How to avoid building software ("binaries")
- How to know it will work together
  - Build the entire universe for every cosmetic change? (à la Nix†)
  - Rely on compatibility between binaries (à la conda-forge)

† Nix is an excellent tool and it's a perfectly valid approach, but I think it's the wrong one for HEP.

Binary distribution

- conda is primarily a binary package manager
  - Recipes are used to build binaries for multiple platforms
  - Binaries are distributed via "channels", the most popular of which is conda-forge

Binary distribution

- conda is primarily a binary package manager
  - Recipes are used to build binaries for multiple platforms
  - Binaries are distributed via "channels", the most popular of which is conda-forge
- Potentially a different model than what you're used to, but is a very pragmatic approach

Binary distribution

- conda is primarily a binary package manager
  - Recipes are used to build binaries for multiple platforms
  - Binaries are distributed via "channels", the most popular of which is conda-forge
- Potentially a different model than what you're used to, but is a very pragmatic approach
- For two binaries to be compatible it doesn't matter:†
  - Which C++ compiler was used to build them
  - Which C++ standard is used
  - Which Linux distribution you use

† This is massively over simplified but we don't have time to go into the details here.

conda-forge

- Shared CI and distribution infrastructure.
- ~7800 contributors, 33,000+ packages, 43 billion+ downloads
  - contributors maintain one or more packages
  - core team maintains the infrastructure and keeps the ecosystem healthy
- Heavy use of automation to manage version updates, rebuilds, and ABI change "migrations"
- Isn't frozen: you can contribute to add / update / fix packages
  - Uploaded binaries are immutable, associated metadata isn't
- HEP leadership:
  - Chris Burr is member of core leadership team
  - Chris Burr, Matthew Feickert are members of conda-forge/staged-recipes review team

The HEP Packaging Coordination project

- A community project to get **as much HEP software as possible** onto conda-forge.
  - Directed by: Chris Burr, Matthew Feickert, Lindsey Gray, Giordon Stark, ...you!
- Contributors across HEP: ATLAS, Belle II, CMS, DIRAC, IRIS-HEP, LEGEND, LHCb, ROOT, Scikit-HEP, SHiP, theory/pheno...
- **120+ HEP packages** already: ROOT, Pythia8, FastJet, Awkward Array, Rivet, CMS Combine, ...
- Installing should be trivial

hep-packaging-coordination


Development

What is development?

- Someone writing
  - libraries for general use (e.g. ROOT, FastJet, Rivet, Pythia, ...).
  - software for a specific experiment (e.g. LHCb, ATLAS, CMS, Belle II, ...).
  - a niche analysis tool for use inside an experiment

What is development?

- Someone writing
  - libraries for general use (e.g. ROOT, FastJet, Rivet, Pythia, ...).
  - software for a specific experiment (e.g. LHCb, ATLAS, CMS, Belle II, ...).
  - a niche analysis tool for use inside an experiment
- Someone doing
  - analysis for physics
  - quick one-off detector study

Pixi-build

- The workspace model mentioned works really well for all of these cases
  - No global state which can prevent you from working on multiple projects
  - Collaborate effectively by having the same software everywhere

Pixi-build

- The workspace model mentioned works really well for all of these cases
  - No global state which can prevent you from working on multiple projects
  - Collaborate effectively by having the same software everywhere
- Pixi-build takes this a step further: you can depend on unreleased software
  - Point to branches in git repositories
  - Point to local clones of your dependencies

Pixi-build

- The workspace model mentioned works really well for all of these cases
  - No global state which can prevent you from working on multiple projects
  - Collaborate effectively by having the same software everywhere
- Pixi-build takes this a step further: you can depend on unreleased software
  - Point to branches in git repositories
  - Point to local clones of your dependencies
- Kind of like Python's editable installs except for any language

An aside: Use of conda in LHCb

- Most LHCb software is on conda-forge (even extremely LHCb-specific software)
- Already used for:
  - User login environment on CVMFS/lxplus
  - All grid middleware
  - Hosting web services
  - User analysis environments on CVMFS (lb-conda)
  - Local user environments
- Notable exception is the "physics stack"
  - Have been experimenting with this more recently
  - Extremely promising, main questions are around the nicest way to integrate it

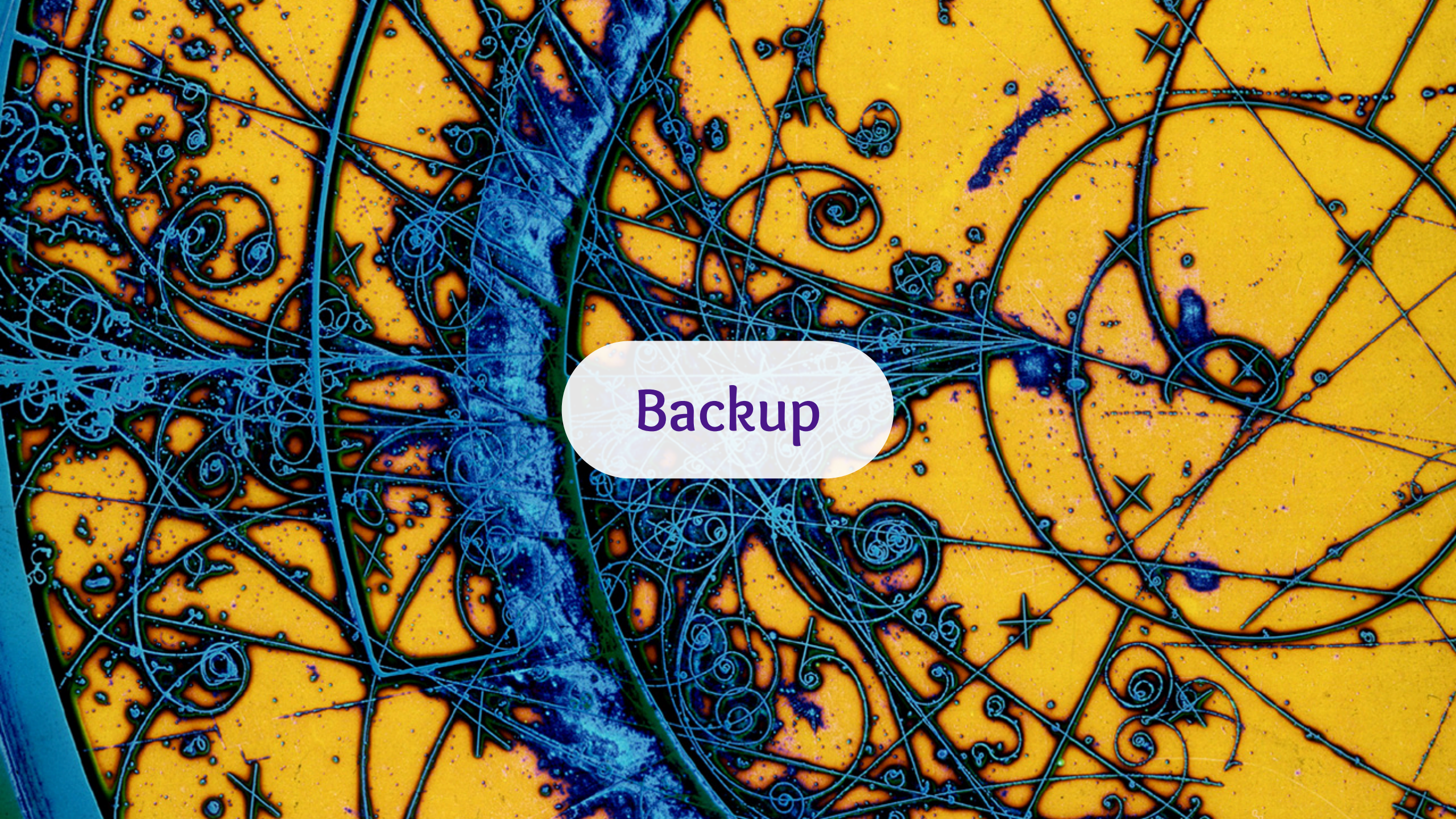
Summary

- conda-forge provides the recipes/binaries
- The conda community provides the tooling
- HEP Packaging Coordination coordinates efforts within HEP

Links:

- HEP Packaging Coordination — https://hep-packaging-coordination.github.io/.github/
- Matthew's CHEP 2026 talk — https://matthewfeickert-talks.github.io/talk-chep-2026/
- "Conda, Pixi and RattlerFS" — https://talks.chrisburr.me/2026-06-22-pixi-and-rattlerfs/


Questions?


Backup

What does typical end-user use look like?

```
$ pixi init example && cd example  # create workspace
$ pixi add contur  # declaratively add tools
✔ Added contur >=3.1.4,<4
$ pixi run contur ...  # execute commands or tasks
$ pixi list rivet  # inspect environments
Name   Version  Build                Size  Kind   Source
rivet  4.1.3    py314h9404863_2  53.69 MiB  conda  https://conda.anaconda
$ pixi shell  # drop into interactive subshells

(debug) $ command -v contur
/tmp/example/.pixi/envs/default/bin/contur
```

pixi global install

- Install CLI tools that are always available, each in its **own isolated environment**
- Only the exposed executables land on your PATH — no dependency clashes between tools
  - Could have a pyroot640 and pyroot642 installed at the same time, for example

```
~> pixi global install root --with ipython
~> pixi global expose add --environment root pyroot=ipython
✓ Exposed executable pyroot from environment root.

~> which pyroot
/Users/cburr/.pixi/bin/pyroot          # just a wrapper on PATH

~> pyroot
IPython 9.14.1 -- An enhanced Interactive Python.
```

pixi exec

- Run a command in a **throwaway environment** — nothing is installed
- Pixi fetches what's needed, runs it, then it's gone — ideal for one-offs & CI

```
~> pixi exec root -l -b -q -e '1+1'
(int) 2
```

Multiple channels

- There is precedent for using multiple conda channels together
- Most significant ones use conda-forge as a base:
  - Bioconda for bioinformatics software
  - RoboStack for robotics software
  - Emscripten Forge for the `emscripten-wasm32` platform
- For HEP I think it's better to work within conda-forge
  - Get a lot of benefits from the shared infrastructure
  - Avoids overlap with multi-purpose software (ROOT, Geant4, ...)
- I think it can make sense to have smaller channels for niche software
  - LHCb will likely end up with a channel for their physics stack, for example

RattlerFS: What is the problem?

- The main trade off for conda is disk space and IO usage
  - No different from containers in that regard
  - Pixi tries to be smart with reflinking/hardlinks, laziness, etc.
- It's fine on a local NVMe drive (e.g. your laptop)
  - On hard drives it's not ideal but manageable
  - On AFS it's almost unusable
  - Having it in every grid job would be a disaster (there is a reason we have CVMFS!)

RattlerFS Primer 1: What does "install" mean?

- What does "installing a conda package" mean?
  1. Download and extract the package to the local cache
  2. "Copy" the files to the install location
  3. Apply any necessary fixes (e.g. shebangs, hard coded paths, python stuff, ...)
- RattlerFS is a virtual filesystem which implements step 2+3
  - Proxies data from the local cache to the install location on demand
  - No need to copy files, no disk usage or IO overhead!

RattlerFS Primer 2: One step further with CVMFS

> Download and extract the package to the local cache

- We already have a tool for that: CVMFS!
- Cache all conda-forge packages on CVMFS
  - RattlerFS can proxy them to the install location on demand:

```
$ ls /cvmfs/conda-cache.cern.ch/prototype-v2/*
linux-64/  noarch/  osx-arm64/
```

RattlerFS: Status

- This works on Linux/macOS/Windows using FUSE/NFS/ProjFS*
- The upstream package that provides conda tooling (`rattler`) is interested
- "Just" need to open the (large) pull request...

** Not all backends work on all operating systems*