Towards production readiness with the Key4hep software stack for future colliders

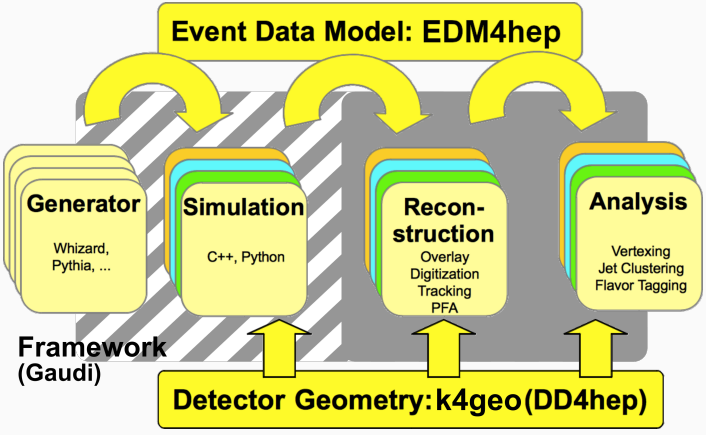
Thomas Madlener
for the Key4hep developers

LCWS 2024

Jul 10, 2024

This project has received funding from the European Union's Horizon 2020 Research and Innovation programme under grant agreement No 101004761.

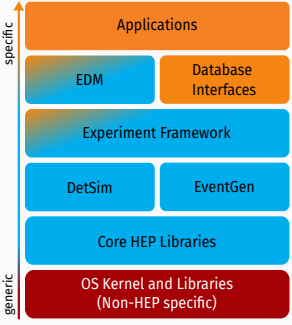
From generation to analysis - the general workflow



- Many steps involved from generating events to analyzing them
- Hundreds of SW packages
  - Building & deploying
  - Consistency
  - Reproducibility
- Try to give an overview of the **Key4hep** SW ecosystem
  - Highlight LC related parts

Key4hep - A (very) brief introduction

- Future detector studies rely on well maintained software for studying their potential
- Maintenance of a consistent HEP SW stack is non-trivial
  - Ecosystem of interacting components
- Sharing the burden allows everybody to reap the benefits
  - Make best use of scarce (human) resources
- **Regular contributions from ILC, CLIC, FCC, CEPC, EIC, (MuonCollider), ...**
- Support from major R&D initatives
  - CERN R&D for Future Experiments, AIDAinnova WP12, ECFA

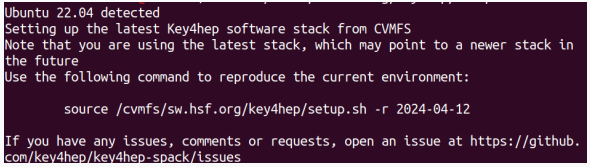


Keyhep releases and nightlies

- (Rolling) latest release of the complete Key4hep software stack
  - Full stacks for AlmaLinux9, Ubuntu22.04, (CentOS7)

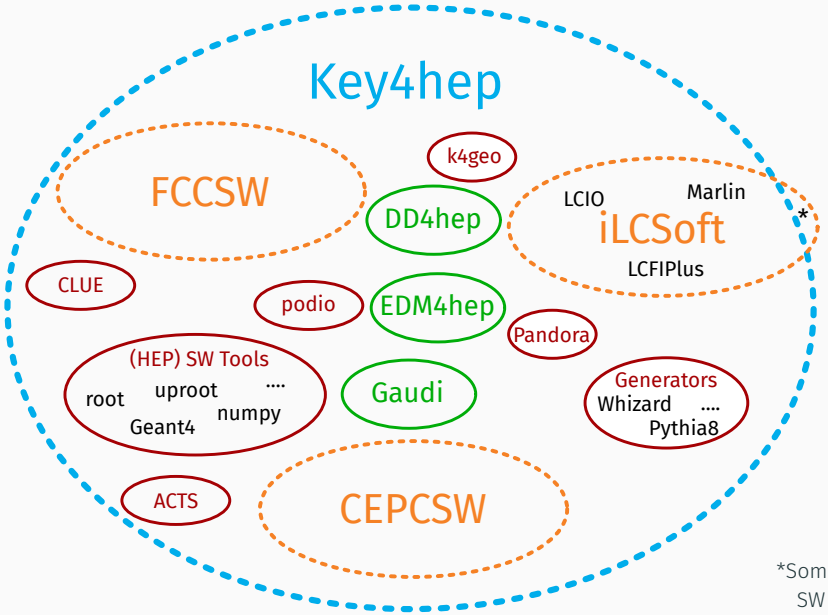
```
/cvmfs/sw.hsf.org/key4hep/setup.sh
/cvmfs/sw-nightlies.hsf.org/key4hep/setup.sh
```

- Documentation
  - key4hep.github.io/key4hep-doc
  - Includes tutorials & How-tos



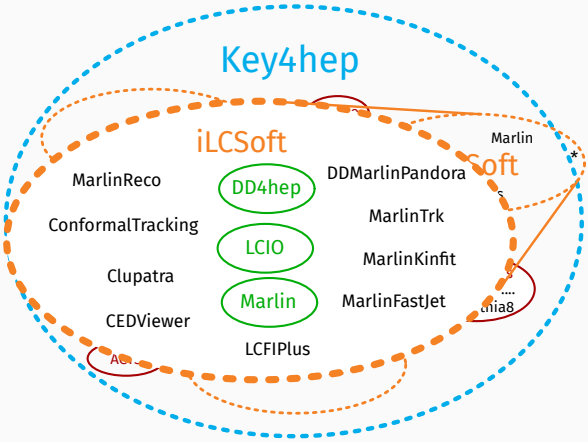
- **Release early and release often**
  - Make fixes available early
  - Discover problems and collect feedback as early as possible
- Biweekly, alternating meetings for Key4hep & EDM4hep
  - indico.cern.ch/category/11461/

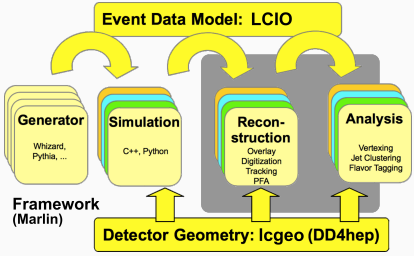
Key4hep (simplified) overview



*Some testbeam related SW not yet included

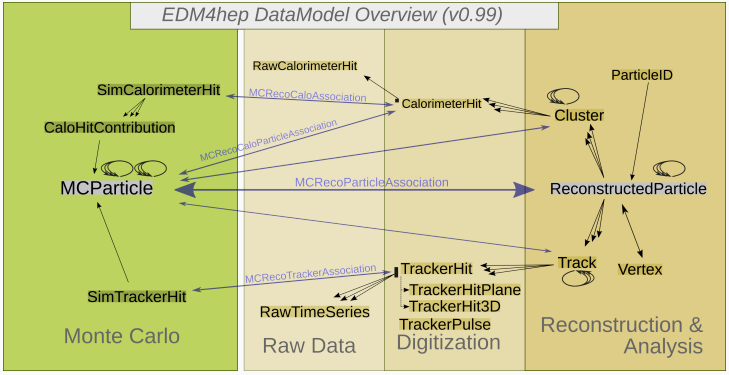
iLCSoft - The stack inside the stack





- Full suite of reconstruction & analysis tools
  - Used in numerous productions for LC studies
- LCIO vs EDM4hep
- Marlin vs Gaudi

EDM4hep - The common EDM for Key4hep

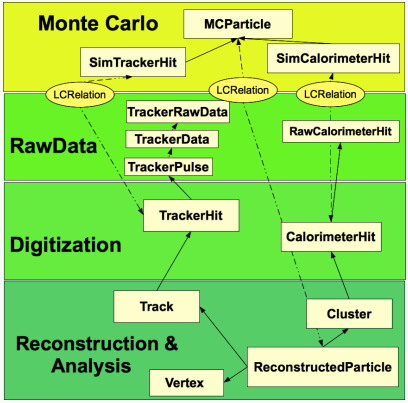


- Based on `LCIO` and `FCC-edm`
  - Focus on usability in reconstruction & analysis
- Currently finalizing v1.0
- Can easily be extended
  - Used by EDM4eic
  - Prototyping!
- Generated via `podio`
  - **v1.0 now available!** 🎉

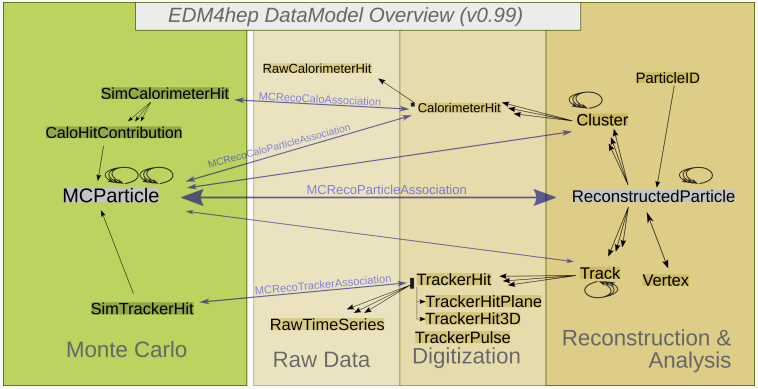
key4hep/EDM4hep
edm4hep.web.cern.ch
AIDASoft/podio
key4hep.web.cern.ch/podio

LCIO vs EDM4hep (at the highest level)

LCIO



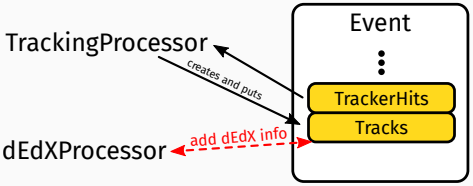
EDM4hep



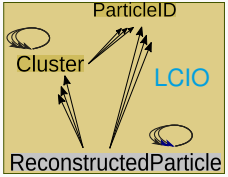
- Since EDM4hep is based on LCIO the high-level structure is very similar
- Some differences in philosophy and implementation
- **Conversion in both directions available in Key4hep** as a library

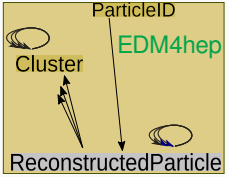
LCIO vs EDM4hep - The conceptual differences

- `LCRelations` vs `Associations`
  - LCIO: (in principle) arbitrary types
  - EDM4hep: dedicated associations only
- Mutability concept
  - LCIO: (almost) always
  - EDM4hep: only at creation time
- Revised some datatypes and their relations
  - `Track`, `Vertex`, `ReconstructedParticle`, `ParticleID`
- Ensure similar functionality with utilities
- Try to keep algorithms / processors as focused as possible



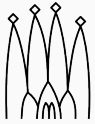

```
edm4hep::RecDqdx:
  Description: "dE/dx or dN/dx"
  Members:
    - edm4hep::Quantity dQdx  // value + error
  OneToOneRelations:
    - edm4hep::Track track    // computed from here
```





Experiment Framework & Core components

- `Gaudi`, originally developed by LHCb, now also used by ATLAS, FCCSW and smaller experiments
  - Supports concurrency
  - “Battle-proven” from data taking during LHC operations



- Key4hep has decided to adapt `Gaudi` as its experiment framework
  - Contribute to its development where necessary

The `k4FWCore` package

key4hep/k4FWCore

- Providing core functionality, e.g.
  - Data Service for EDM4hep / podio inputs
  - `k4run` for running options files
- Multithreading support via `Gaudi::Functional`
- Support for variable, runtime configurable number of inputs / outputs

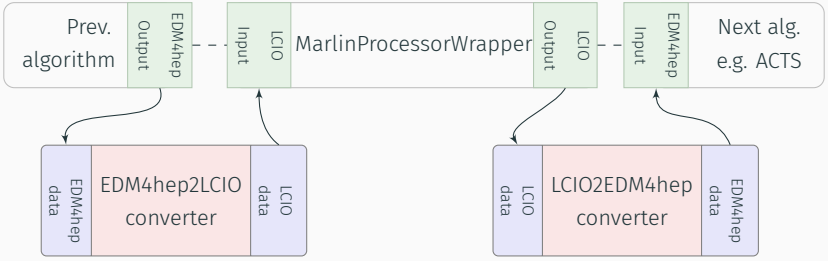
Marlin vs Gaudi

- Conceptually the two frameworks are very similar
  - Schedule different working units
  - Marshall data
- Most obvious differences in naming conventions
  - As always some differences emerge when looking at the details

	Marlin	Gaudi
language	C++	C++
working unit	Processor	Algorithm
config language	XML	Python
transient data format	LCIO	anything
set up function	`init`	`initialize`
work function	`processEvent`	`execute`
wrap up function	`end`	`finalize`

k4MarlinWrapper

- Wraps Marlin processor in a Gaudi algorithm and allows to run them unchanged
- Automatic, on-the-fly conversion between LCIO and EDM4hep
- Allows to “mix and match” existing reconstruction algorithms with new developments
  - Working horse for many FCC full simulation studies at the moment



ILD standard reconstruction in Key4hep

- All configuration available from iLCSoft/ILDConfig
- Everything that works in iLCSoft also works in Key4hep!

```
Marlin MarlinStdReco.xml --global.LCIOInputFiles=<input-file> [...]
```

- Now also with Gaudi

```
k4run ILDReconstruction.py --inputFiles=<input-file> [...]
```

  - Works with EDM4hep and LCIO inputs
  - EDM4hep output by default, LCIO output via `--lcioOutput=[true|only]`
- Facilitates collaboration with other projects, e.g. CLD
- Full migration of all workflows will take some time but process started
  - Some new developments already done exclusively in Gaudi configuration

iLCDirac - Using Key4hep in production

- iLCDirac provides functionality to run jobs on grid resources
- Extensively used for productions using iLCSoft
- Key4hep releases & nightlies available through iLCDirac
  - Similar to using `Marlin`

GaudiApp

Gaudi Application to run applications based on Gaudi.

New in version v32r0p1.

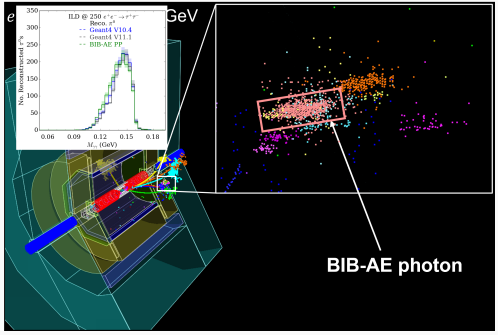
Usage:

```
>>> ga = GaudiApp()
>>> ga.setVersion('key4hep-latest')
>>> ga.setExecutable("k4run")
>>> ga.setSteeringFile('k4simdelphesalg_pythia.py')
>>> ga.setPythia8Card('p8_ee_ggqq_ecm91.cmd')
>>> ga.setExtraCLIArguments("--GenAlg.PythiaInterface.pythiaca
>>> ga.setEnergy(91.2)
>>> ga.setNumberOfEvents(50)
>>> ga.setOutputFile('output.root')
```

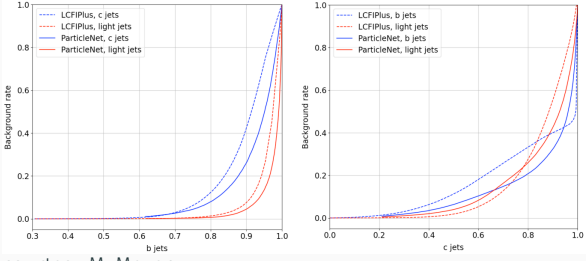
see the documentation

Using Key4hep for new developments

- DDFastShowerML - integration of generative shower models into DD4hep
  - Study reco of ML gen. showers
- See P. McKeown's PhD thesis
- More ML models integrated



courtesy P. McKeown

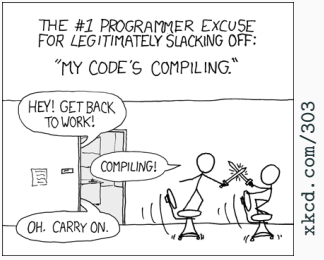


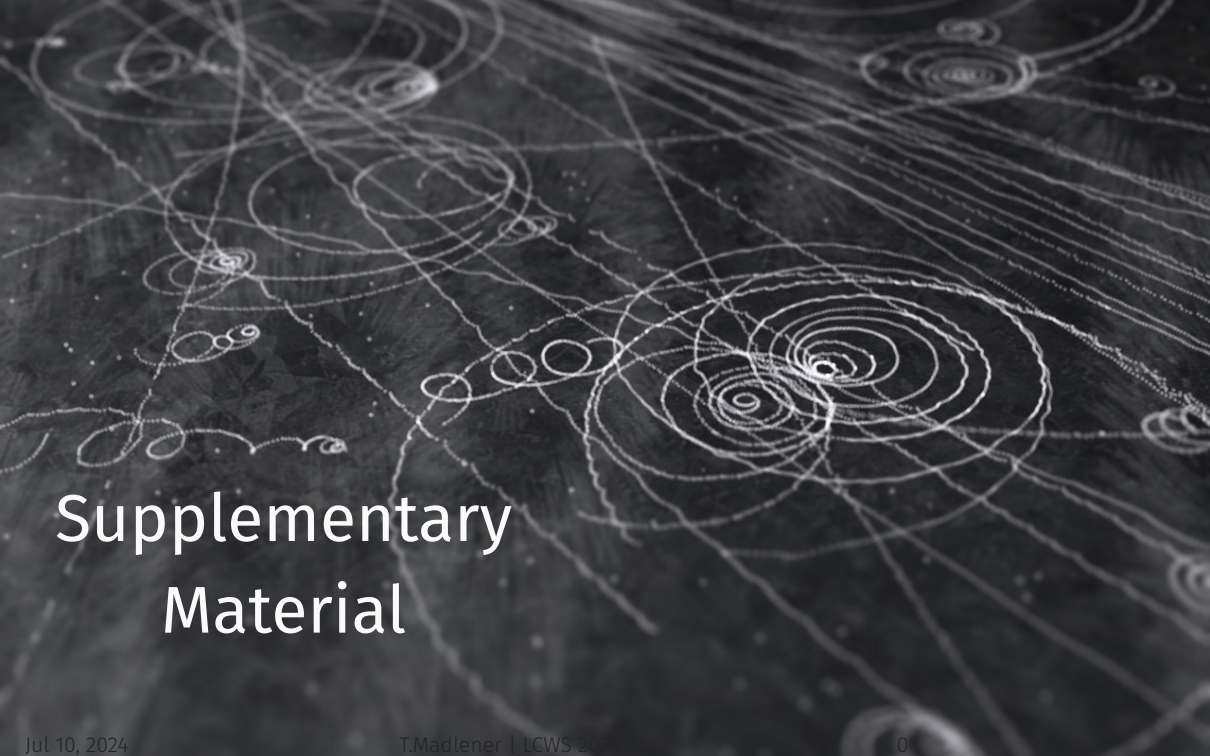
courtesy M. Meyer

- MarlinMLFlavorTagging - Using deep ML flavor taggers w/ Marlin
- Basis for similar developments in Gaudi
- Avaialable in Key4hep nightlies

Summary & Outlook

- Key4hep provides a common software stack for all future collider projects
- **Very successful in bringing together communities and focusing on common approaches**
- Existing reconstruction & analysis software from iLCSoft works unchanged
- Currently finalizing first stable versions of core components
- Migration towards “Key4hep native” ongoing for ILD
- Still lots of work ahead
  - Many exciting possibilities

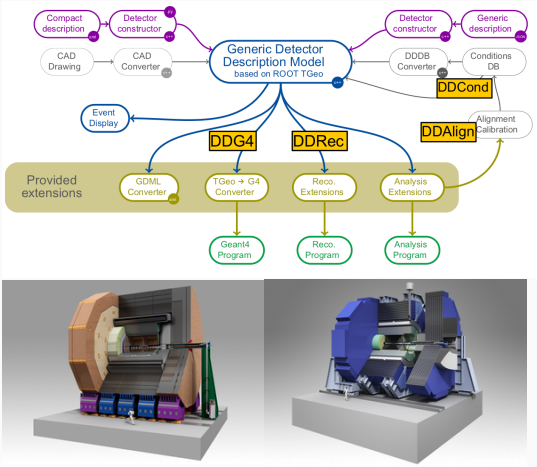



Supplementary
Material

DD4hep - Detector description

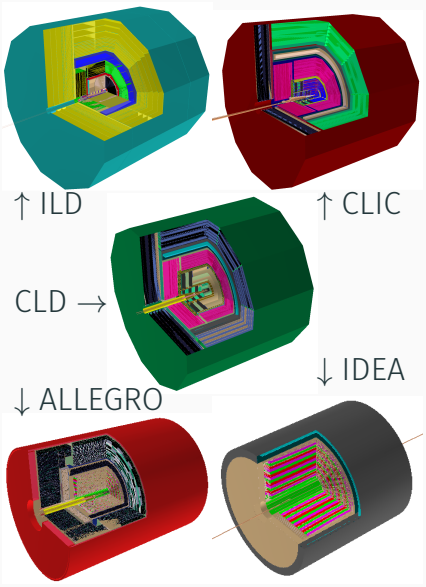
dd4hep.web.cern.ch

- Complete detector description
  - Geometry, materials, visualization, readout, alignment, calibration, ...
- From a **single source of information**
  - Simulation, reconstruction, analysis
- Comes with a powerful plug-in mechanism that allows customization
- More or less “industry standard” now
  - FCC, ILC, CLIC, EIC, LHCb, CMS, ODD, ...
- `ddsim` - standalone simulation executable

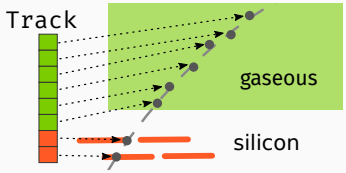


k4geo - The detector geometry repository

- Central repository for detector models
- Many existing detector models from LC studies
- Many recent developments for FCC detector concepts
- “Plug and play” approach for subdetectors
  - Use CLD inner tracker in ILD for TPC studies at FCC



Interface types and their use in EDM4hep




```yaml
interfaces:
  edm4hep::TrackerHit:
    Types: [edm4hep::TrackerHit3D, edm4hep::TrackerHitPlane]
    Members:
      - edm4hep::Vector3f position [mm] // hit position

datatypes:
  edm4hep::Track:
    OneToManyRelations:
      - edm4hep::TrackerHit trackerHits // hits of this track
```

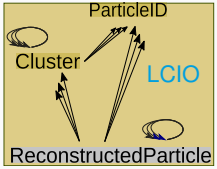
```cpp
auto track = edm4hep::Track{};
track.addHit(edm4hep::TrackerHit3D{});
track.addHit(edm4hep::TrackerHitPlane{});

const auto hits = track.getHits();
hits[0].isA<edm4hep::TrackerHit3D>(); // <-- true
hits[0].as<edm4hep::TrackerHit3D>();  // <-- "cast back"
hits[1].isA<edm4hep::TrackerHit3D>(); // <-- false
hits[1].as<edm4hep::TrackerHit3D>();  // <-- exception!
```

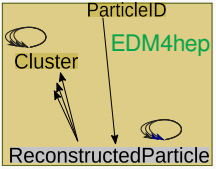
- General interface can be useful to "gloss over some details"
- Value semantics prevent inheritance based approach
  - Pointers in interfaces break consistency
  - No base class to inherit from
- Introduce *interfaces* as new category in YAML definition
  - Define desired functionality
  - **No collections!**
  - Use like normal *datatypes*
  - "Casting back" is possible

ParticleID handling EDM4hep vs LCIO

- Remove `ParticleID` relation from `Cluster`
  - Found no usage in ILD / CLIC reconstruction
- Make `ParticleID` have a one-to-one relation to `ReconstructedParticle`
  - Also remove `particleIDUsed`
- `ParticleID` has been (ab)used in LCIO as *transient parameter (values) store*
  - Will require change of pattern for EDM4hep
- Simple use cases become simpler with EDM4hep
- Tooling keeps the rest at the same level
- Some usability improvements wrt LCIO
  - keyhep/EDM4hep#298

key4hep/EDM4hep#268





ParticleID related utilities

- `edm4hep::utils::PIDHandler` similar to `UTIL::PIDHandler`
  - Get related `ParticleID`s from a `ReconstructedParticle`
  - Retrieve some PID metadata
  - Slightly more modern interface for EDM4hep
- Handling of necessary metadata very different
  - LCIO: *collection parameters* - tight coupling
  - EDM4hep: *file level metadata* - looser coupling
  - Gory details here and here
- See the documentation for more usage examples
- **Feedback very much appreciated!**

ParticleID handling comparison

Getting the dE/dx distance wrt an electron for all particles

```cpp
using namespace EVENT;
using namespace UTIL;

auto recos = event->getCollection("PandoraPFOs");
auto pidHandler = PIDHandler(recos);

const auto dEdxId = pidHandler.getAlgorithmID(_dEdxname);
const auto dEdx_e_Id = pidHandler.getAlgorithmID(dEdxId,
                                                 "e_dEdx_dist");

for (int i = 0; i < recos->getNumberOfElements(); ++i) {
  auto p = static_cast<ReconstructedParticle*>(
             recos->getElementAt(i));

  if (p->getCharge() == 0.0) {
    continue; // only charged particles have tracks
  }

  const auto& dEdxParams = pidHandler.getParticleID(p, dEdxId);
  const auto dEdx_e_dist = dEdxParams[dEdx_e_Id];

  // do something with the particle and the dEdx distance
}
```

LCIO

```cpp
using namespace edm4hep;
using namespace edm4hep::utils;

const auto dEdx = event.get<ParticleIDCollection>("dEdx");


const auto dEdxMeta = PIDHandler::getAlgoInfo(metadata, "dEdx");
const auto dEdx_e_Id = getParamIndex(dEdxMeta, "e_dEdx_dist");


for (const auto pid : dEdx) {
  const auto p = pid.getParticle();









  const auto dEdxParams = dEdx.getParameters();
  const auto dEdx_e_dist = dEdxParams[dEdx_e_Id];

  // do something with the particle and the dEdx distance
}
```

EDM4hep