EDM4hep and PODIO

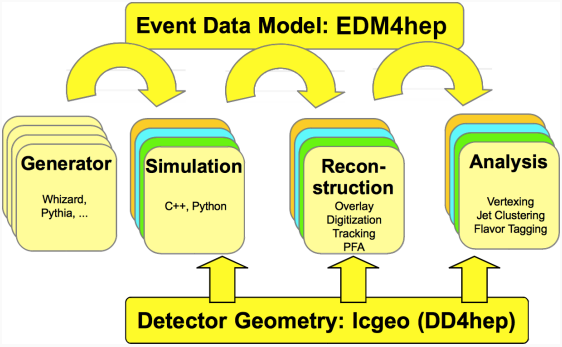
Introduction and Overview

Thomas Madlener
for the Key4hep team

EIC CompSW Weekly Meeting
June 15, 2022

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

The EDM at the core of HEP software

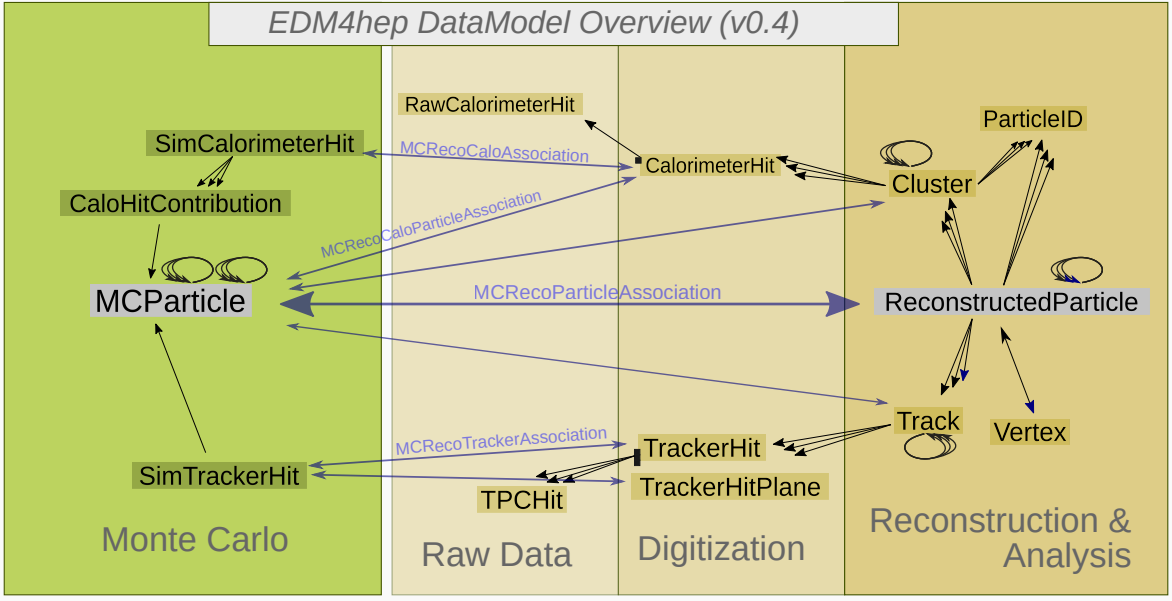


- Different components of HEP experiment software have to talk to each other
- The event data model defines the language for this communication
- Users express their ideas in the same language

EDM4hep goals

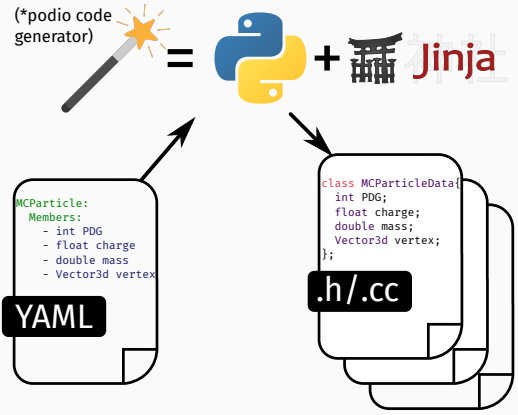
- The Key4hep project aims to define a common software stack for all future collider projects
  - Regular contributions from **ILC, CLIC, FCC-ee & FCC-hh, CEPC, EIC (ATHENA)**, ...
- EDM4hep is the **shared, common EDM** that can be used by **all communities** in the Key4hep project
- Support different use cases from these communities
  - Different collision environments lead to different requirements for an EDM
- Efficiently implemented, support multi-threading and have heterogeneous resources in mind
- Build on past experience from fcc-edm and LCIO, which has already been succesfully shared by the LC communities

EDM4hep schema



podio as generator for EDM4hep

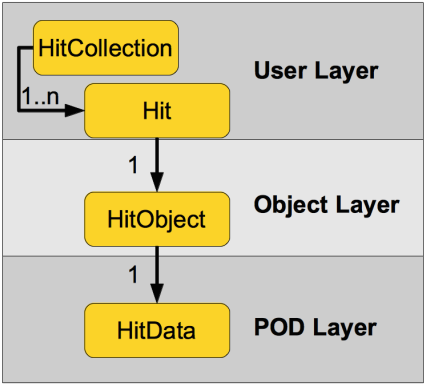
- Traditionally HEP c++ EDMs are heavily Object Oriented
- Use `podio` to generate thread safe code starting from a high level description
- Provide an easy to use interface to the users



AIDASoft/podio

The three layers of podio

- podio favors **composition over inheritance** and uses **plain-old-data (POD)** types wherever possible
- Layered design allows for efficient memory layout and performant I/O implementation



podio - datamodel definition

```yaml
components:
  edm4hep::Vector3f:
    Members: [float x, float y, float z]

datatypes:
  edm4hep::ReconstructedParticle:
    Description: "Reconstructed Particle"
    Author : "F.Gaede, DESY"
    Members:
      - edm4hep::Vector3f     momentum   // [GeV] particle momentum
      - std::array<float, 10> covMatrix  // energy-momentum covariance
    OneToOneRelations:
      - edm4hep::Vertex startVertex // start vertex associated to this particle
    OneToManyRelations:
      - edm4hep::Cluster clusters  // clusters that have been used for this particle
      - edm4hep::ReconstructedParticle particles // associated particles
    ExtraCode:
      declaration: "bool isCompund() const { return particles_size() > 0; }\n"

  edm4hep::ParticleID:
    VectorMembers:
      - float parameters // hypothesis params
```

*extracted from edm4hep.yaml

- Reusable components
- Fixed sized arrays as members
- *VectorMembers* for variable sized array members
- 1 – 1 and 1 – N relations
- Additional user-provided code

podio - features of generated code

```cpp
auto recos = ReconstructedParticleCollection();
// ... fill ...
for (auto reco : recos) {
  auto vtx = reco.getStartVertex();
  for (auto rp : reco.getParticles()) {
    auto mom = rp.getMomentum();
  }
}
```

← c++17 code with “value semantics”

↓ Python bindings via PyROOT

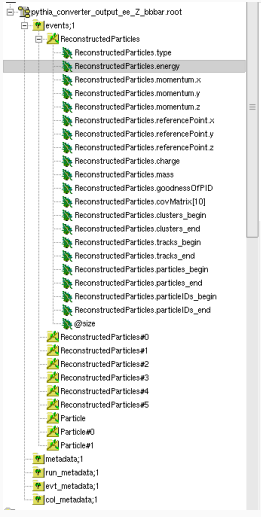
```python
recos = ReconstructedParticleCollection()
#... fill ...
for reco in recos:
  vtx = reco.getStartVertex()
  for rp in reco.getParticles():
    mom = rp.getMomentum()
```

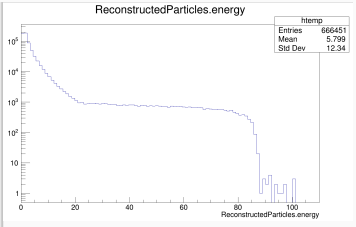
```python
d = ROOT.RDataFrame('events', 'events.root')
h = (d.Define('abs_pdg', 'abs(Particle.PDG)')
      .Define('mu_sel', 'abs_pdg == 13')
      .Define('mu_px',
              'Particle.momentum.x[mu_sel]')
      .Histo1D('mu_px'))
h.DrawCopy()
```

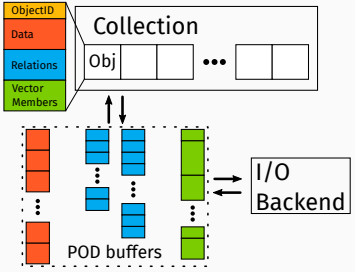
← Using `RDataFrame` to read ROOT files

podio supports different I/O backends

- Default **ROOT** backend
  - POD buffers are stored as branches in a `TTree`
  - Files can be interpreted **without EDM library**(!)
  - Can be used in `RDataFrame` or with `uproot`
- Alternative **SIO** backend
  - Persistency library used in `LCIO`
  - Complete events are stored as binary records
- Adding more I/O backends is possible







CMake interface for projects using podio

```cmake
find_package(PODIO)

# generate the c++ code from the yaml definition
PODIO_GENERATE_DATAMODEL(edm4hep edm4hep.yaml headers sources IO_BACKEND_HANDLERS "ROOT;SIO")
# compile the core data model shared library (no I/O)
PODIO_ADD_DATAMODEL_CORE_LIB(edm4hep "${headers}" "${sources}")
# generate and compile the ROOT I/O dictionary
PODIO_ADD_ROOT_IO_DICT(edm4hepDict edm4hep "${headers}" src/selection.xml)
# compile the SIOBlocks shared library for the SIO backend
PODIO_ADD_SIO_IO_BLOCKS(edm4hep "${headers}" "${sources}")

# Install the created targets
install(TARGETS edm4hep edm4hepDict edm4hepSioBlocks)
```

- Easy to use functions for integrating a podio generated EDM into a project
- Split into core EDM library and I/O handling for different backends
  - Pick what you need
  - I/O handling parts dynamically loaded by podio on startup

podio support in framework(s)

- podio has been used by `fcc-edm`
- key4hep/k4FWCore provides Gaudi integration for podio generated EDMs
  - Originally developed for FCCSW, now adapted by Key4hep
- juggler has essentially the same functionality as `k4FWCore`
  - Developed in the context of ATHENA
- Essentially re-implementation(s) of “standalone” ROOT writers/readers of podio
  - Including some differences that snuck in over time

Currently ongoing work

- Schema evolution of generated EDMs
  - Leverage existing code generation facilities
  - Allow for user intervention for non-trivial cases (e.g. change of coordinate system)
  - Incrementally implement required features
  - Foresee the possibility to exploit existing schema evolution from I/O backend (e.g. ROOT)
- `podio::Frame` - a generalized (event) data container
  - Offer a flexible mechanism for organizing data into different categories (e.g. Run, Event, ...)
  - Easy to use, thread-safe interface

Future plans

- Release v1.0 with backwards compatibility from then on
  - After schema evolution and Frame have been implemented / merged
- Consolidate the different I/O implementations (“standalone” vs. different framework implementations)
- Implement missing features (from our perspective)
  - User defined relations/associations between arbitrary types
  - *Interface types* that allow for easier high level workflows (e.g. tracker hits for different technologies)
- Implement existing feature requests from community
  - Make it possible to extend an existing EDM (AIDASoft/podio#263)
  - Allow for default initialization values in datatypes (AIDASoft/podio#266)
- Start exploring usage on heterogeneous resources

Experience using EDM4hep and podio

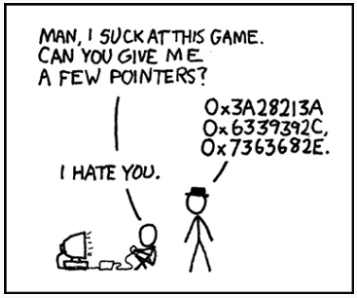
- EDM4hep is actively used for physics and detector studies and the main "customer" of podio
- "Re-implemented" by `eicd`
  - Adapted EDM4hep after first using their own podio generated EDM
  - Defines a few extra data types
- Biweekly meetings to discuss podio and EDM4hep
  - indico.cern.ch/category/11461/ - Tuesdays at 9:00AM (CEST), but US friendly timeslot possible
- Community effort is very successful
  - No showstoppers observed so far
  - Problems / issues are usually resolved quickly
  - Biggest issue so far has been missing schema evolution

Summary

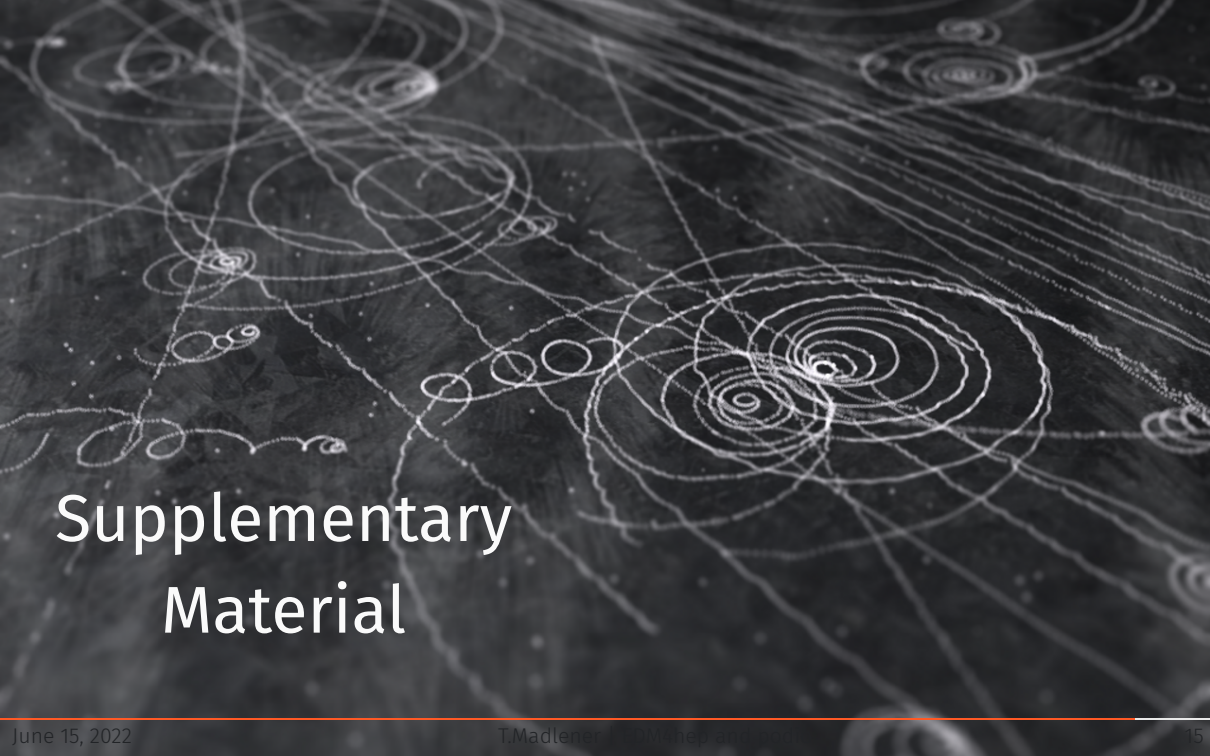
- EDM4hep is the shared, common EDM for the Key4hep project
- It is generated via the podio EDM toolkit
- Tackling schema evolution and adding the `Frame` are the major milestones for podio v1.0
  - Some additional features will be tackled after that
- EDM4hep is successfully used for physics and detector studies
- Community effort is a success
  - Also happy to welcome new contributors and communities

Pointers to software (re)sources

- Key4hep
  - key4hep.github.io/key4hep-doc
  - key4hep - github organisation
- EDM4hep
  - key4hep/EDM4hep
  - cern.ch/edm4hep
- podio
  - AIDASoft/podio
- k4MarlinWrapper
  - key4hep/k4MarlinWrapper
- FCCAnalyses
  - HEP-FCC/FCCAnalyses

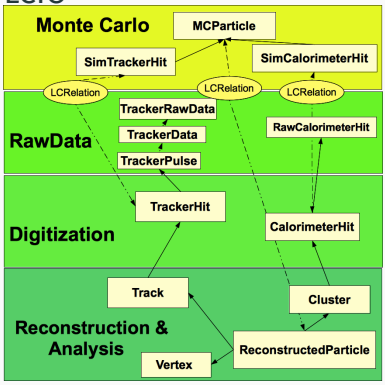


xkcd.com/138

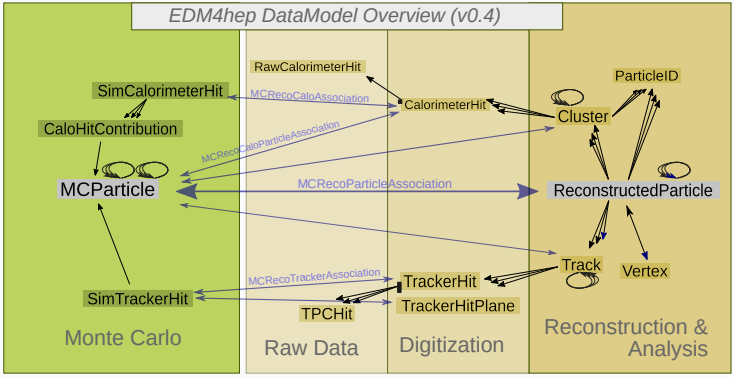

Supplementary
Material

LCIO vs EDM4hep

LCIO

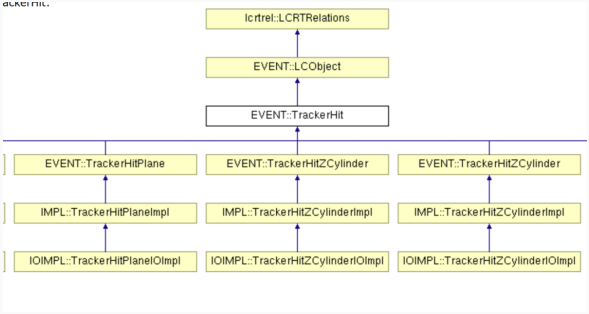


EDM4hep



- Since EDM4hep is based on LCIO the high-level structure is very similar
- Largest differences between the two are due to their implementations
- LCIO has over 15 years of usage. A lot of time to develop tools for it.
  - Not nearly as far with EDM4hep

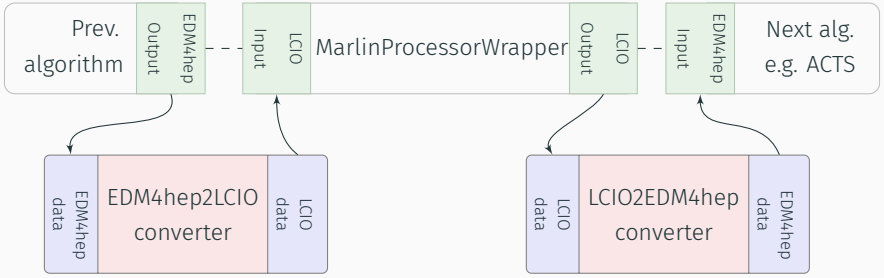
Interface types



- LCIO uses “classic” polymorphism
  - Common `LCObject` base type for all data types
  - `Impl` classes offer the mutable interface
- This is used in some places to add some structure to data types
  - E.g. `TrackerHit` has various implementations different detector technologies
- Not solved for podio (and EDM4hep)

k4MarlinWrapper

- Wraps **Marlin processor** in a Gaudi algorithm and allows to **run them unchanged**
  - Can run a full ILD / CLIC reconstruction and analysis chain via Gaudi
- Converter script to turn Marlin XML steering files into Gaudi python option files
- Automatic, on-the-fly conversion between LCIO and EDM4hep
  - Allows to “mix and match” existing Marlin processors with Gaudi algorithms



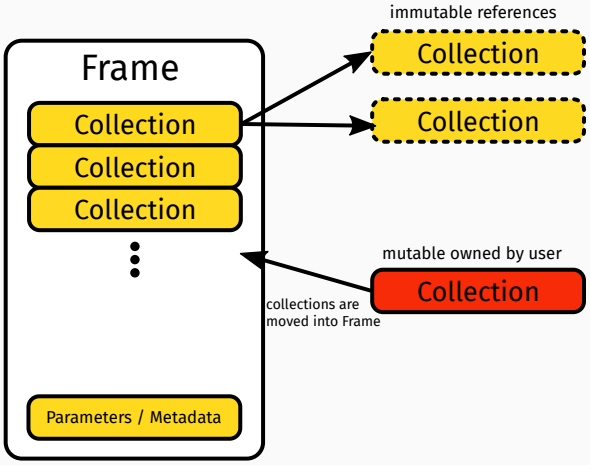
FCCAnalyses

HEP-FCC/FCCAnalyses

- `FCCAnalyses` is a python analysis framework based on `RDataFrame`
  - Comes with high level reco functionality
  - Extensible via C++
- **Not specific to FCC!** Can be run with EDM4hep inputs
- Declarative style of the analysis
- Currently being reworked for improved usability

```
(self.df
 # define an alias for muon index collection
 .Alias("Muon0", "Muon#0.index")
 # define the muon collection
 .Define("muons",  "ReconstructedParticle::get(Muon0, ReconstructedParticles)"
 #select muons on pT
 .Define("selected_muons", "ReconstructedParticle::sel_pt(10.)(muons)")
variables = {
    "mz":{"name":"zed_leptonic_m","title":"m_{Z} [GeV]","bin":125,"xmin":0,"xmax":250},
    "mz_zoom":{"name":"zed_leptonic_m","title":"m_{Z} [GeV]","bin":40,"xmin":80,"xmax":100},
    "leptonic_recoil_m":{"name":"zed_leptonic_recoil_m","title":"Z leptonic recoil [GeV]","bi
    "leptonic_recoil_m_zoom":{"name":"zed_leptonic_recoil_m","title":"Z leptonic recoil [GeV]
    "leptonic_recoil_m_zoom1":{"name":"zed_leptonic_recoil_m","title":"Z leptonic recoil [GeV
    "leptonic_recoil_m_zoom2":{"name":"zed_leptonic_recoil_m","title":"Z leptonic recoil [GeV
    "leptonic_recoil_m_zoom3":{"name":"zed_leptonic_recoil_m","title":"Z leptonic recoil [GeV
    "leptonic_recoil_m_zoom4":{"name":"zed_leptonic_recoil_m","title":"Z leptonic recoil [GeV
 # find zed candidates from  di-muon resonances
 .Define("zed_leptonic",         "ReconstructedParticle::resonanceBuilder(91)("
 # write branch with zed mass
 .Define("zed_leptonic_m",       "ReconstructedParticle::get_mass(zed_leptonic
 # write branch with zed transverse momenta
 .Define("zed_leptonic_pt",      "ReconstructedParticle::get_pt(zed_leptonic)"
 # calculate recoil of zed_leptonic
 .Define("zed_leptonic_recoil",  "ReconstructedParticle::recoilBuilder(240)(ze
 # write branch with recoil mass
 .Define("zed_leptonic_recoil_m","ReconstructedParticle::get_mass(zed_leptonic_
 .Define("zed_leptonic_charge","ReconstructedParticle::get_charge(zed_leptonic
```

podio::Frame basics



- Ownership of the data reflected in the interface
  - Immutable access only for stored data
  - Mutable user owned data has to be explicitly *moved* into the Frame
- Defines an *interval of validity* or category for the stored data (e.g. Run, Event, ...)
- Thread-safe
- First version being tested
  - AIDASoft/podio#287