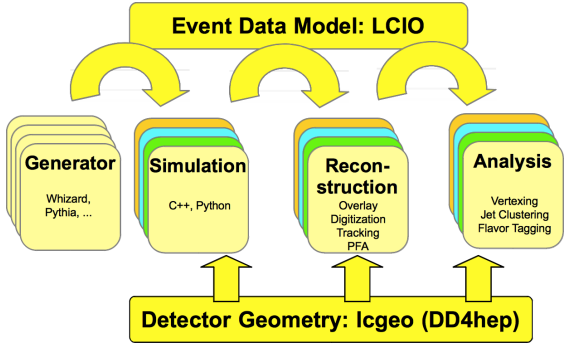
# EDM4hep: an event data model for future collider studies

LCWS 2021

Thomas Madlener
for the Key4Hep team

Mar 15, 2021

# The EDM at the core of HEP software

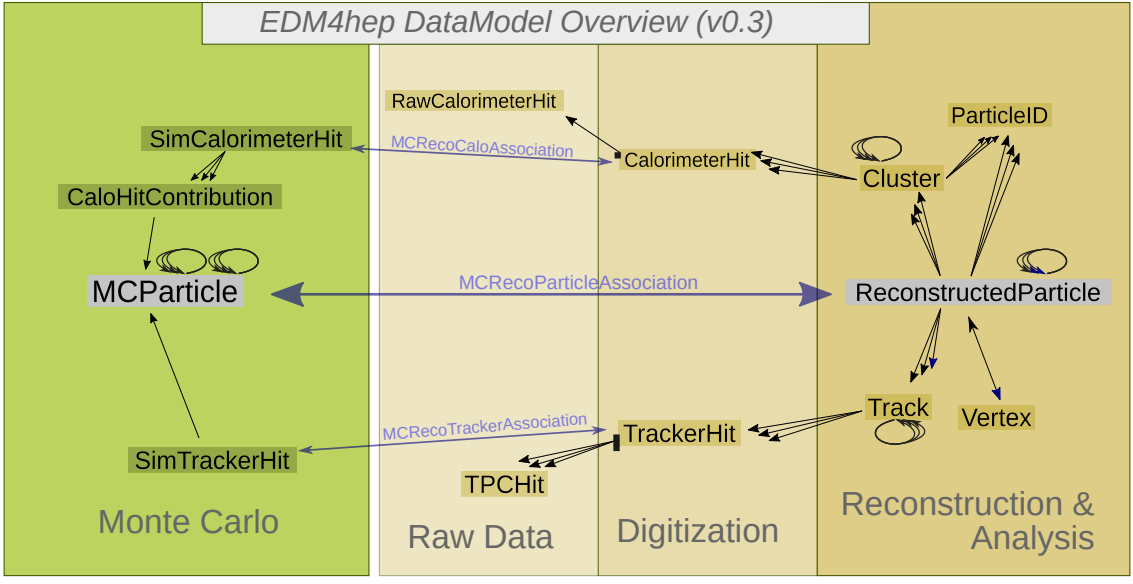



- Different components of HEP experiment software have to talk to each other
- The event data model defines the language for this communication
- Users express their ideas in the same language

# Goals for EDM4hep

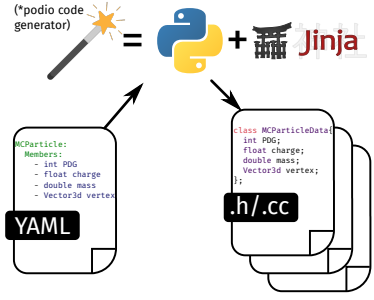
- The Key4Hep project aims to define a common software stack for all future collider projects (see next talks by V. Volkl and P. Fernandez)
- EDM4hep is the common EDM that can be used by all communities in the Key4Hep project
  - ILC, CLIC, FCC-ee & FCC-hh, CEPC, ...
- Support different use cases from these communities
  - Lepton and hadron collisions lead to different environments and requirements for an EDM
- Efficiently implemented, support multi-threading and potentially heterogeneous computing
- Use experience from LCIO which has already been succesfully shared by the LC communities

# EDM4hep schema



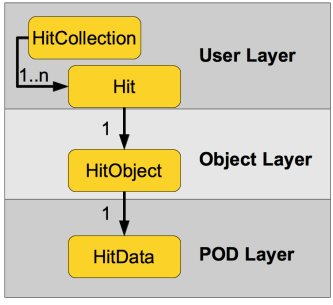
# podio as Generator for EDM4HEP

- Original HEP c++ EDMs are heavily Object Oriented
  - Deep inheritance structures
  - Objects scattered in memory
  - Hard to deal with in multi-threaded environments
- Data access can be very slow with these approaches
- Use `podio` to generate thread safe code starting from a high level description of the desired EDM
  - Target different I/O backends for persistency
  - Users are isolated from implementation details
- Treating python as first class citizen and allow “pythonic” usage
- Still under active development




AIDASoft/podio

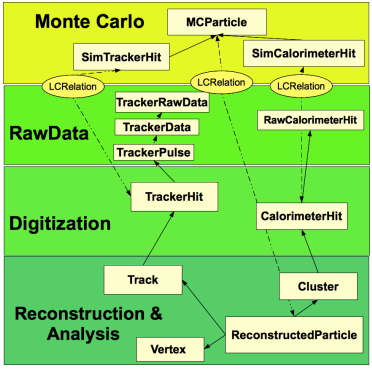
# The Philosophy of podio



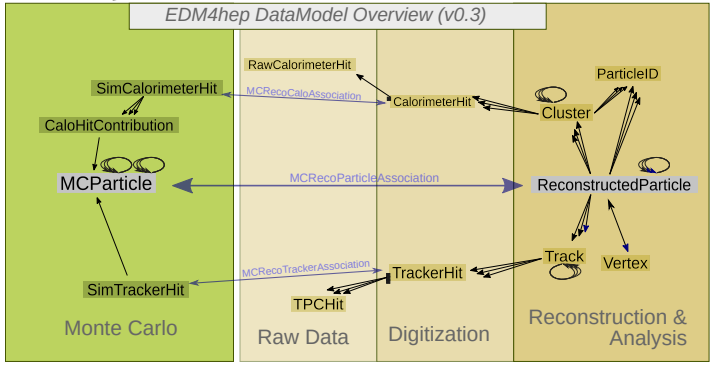

- **User Layer** consists of handles to the EDM objects and offers the full functionality
- The **Object Layer** handles resources and references to other layers
- The actual POD data live in the **POD Layer**
- Easy to use through a clear design of ownership
  - Users should not have to care about this
- Layered design allows for efficient memory layout and performant I/O implementation
  - ROOT I/O is used by default
  - An SIO based implementation is available

# LCIO vs EDM4hep - A side-by-side comparison

**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

# From LCIO to EDM4hep - The easy parts

## LCIO

```
auto* coll = new LCCollectionVec(MCPARTICLE);
auto* mc = new MCParticleImpl;
coll->addElement(mc);

mc->setMass(3.096);

auto* mc2 = static_cast<MCParticle*>(
  coll->getElementAt(0));
auto mass = mc2.getMass();

for (auto* p : mc2.getParents()) { /**/ }
```

## EDM4hep

```
auto coll = MCParticleCollection();
auto mc = coll.create();


mc.setMass(3.096);


auto mc2 = coll[0];
auto mass = mc2.getMass();

for (auto p : mc2.getParents()) { /**/ }
```

- The most common use cases work very similarly with mainly syntactic differences
  - pointer vs. value semantics
- Differences in reader/writer handling are “hidden” by framework code

# From LCIO to EDM4hep - The parts that still require work

## LCIO

```
auto recoMCNav = LCRelationNavigator(
  evt->getCollection("RecoMCTruthLink"));


auto relRecos =
  recoMCNav->getRelatedToObjects(mc);



//
```

## EDM4hep

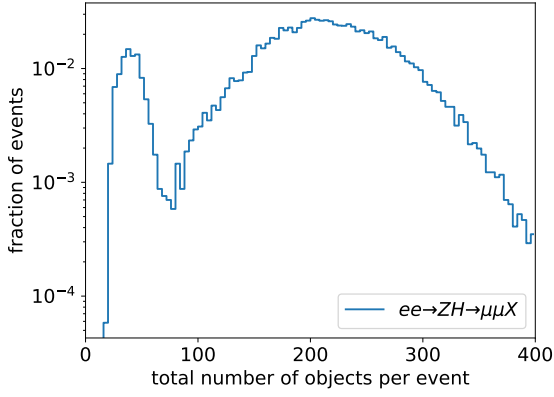
```
auto recoMCAssoc =
  store.get<MCRecoParticleAssocCollection>(
    "RecoMCTruthLink");

std::vector<ReconstructedParticle> relRecos;
for (const auto assoc : relMCAssoc) {
  if (assoc.getSim() == mc) {
    relRecos.push_back(assoc.getRec());
  }
}
```

- LCIO has a 15 years head start in tooling, hiding some of the complexities
- Port the tooling to EDM4hep as we go along and as necessary
- Some things are technically not yet possible in EDM4hep
  - E.g. *subset collections* need a workaround at the moment
- See P. Fernandez's talk for reasons, why you might not have to worry about this

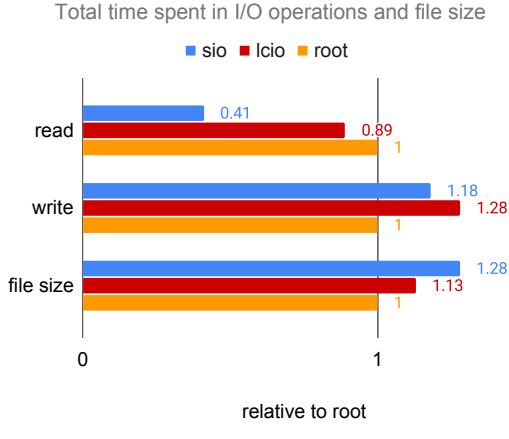
# What about performance?

- key4hep/k4SimDelphes to generate "realistic" benchmark data
  - Only high level objects
- Physics scenario: Higgs recoil @ILD ($\sqrt{s} = 250$ GeV)
  - Mix of small and medium sized events
- Comparing the SIO and ROOT I/O backends available for EDM4hep and LCIO
- Focussing on I/O performance
- Using benchmarking utilities that come with podio



# Benchmark results overview

- For EDM4hep the chosen podio I/O backend can have a significant effect[1]
- LCIO can be read faster than EDM4hep + ROOT
- EDM4hep + SIO more than twice as fast for reading
- EDM4hep + ROOT with smallest resulting files

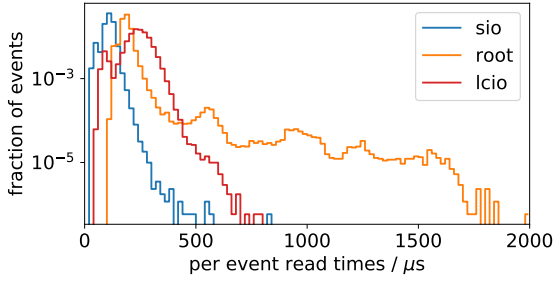


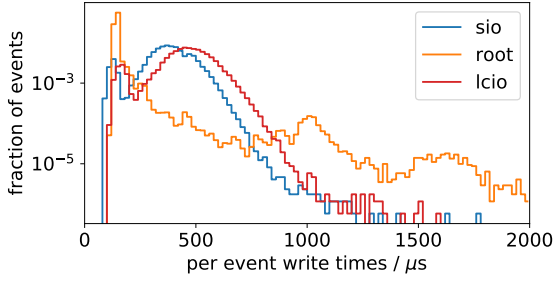

[1]Fixes for the ROOT I/O backend leading to better performance pending

SIO v00-01, LCIO v02-16-01, ROOT v6.20/04

# Per event benchmark results

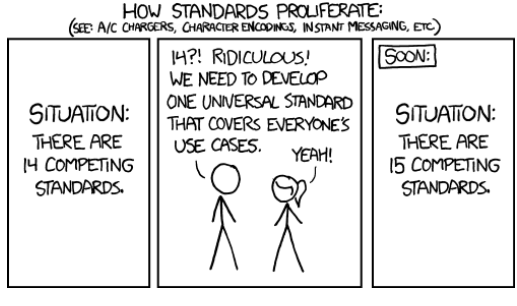
- SIO and LCIO generally seem to scale with the size of the event
- ROOT has very long tails in the per event I/O times
  - This is what makes ROOT slower overall
- Using non-default settings for ROOT might change things
  - Very large problem space



# Currently ongoing work and next steps

- Studies in FCC-hh to see how well EDM4hep works in hadron collision
  - So far LCIO EDM based design developed for lepton colliders seems to work well
- Support for true reference collections in podio/EDM4hep
- Support for storing generic data
  - `LCGenericObject` in LCIO
  - Difficult balance between flexibility, performance and ease-of-use
  - Avoid “EDM inside EDM4hep”
- Support of “flat data formats” for analysis




xkcd.com/927

# Conclusions

- The event data model is at the core of every HEP experiment software framework
- **EDM4hep aims to define a common EDM for future collider projects**
- A first version based on LCIO is available and under evaluation
  - **Feedback is highly welcome!**
- Benchmarks on a “realistic” physics scenario show that EDM4hep can offer better I/O performance
- Still some work ahead
  - Can we support lepton and hadron colliders with one EDM?
  - Tooling around EDM4hep
  - Missing features