



EDM4hep and podio - The event data model of the Key4hep project and its implementation

vCHEP 2021

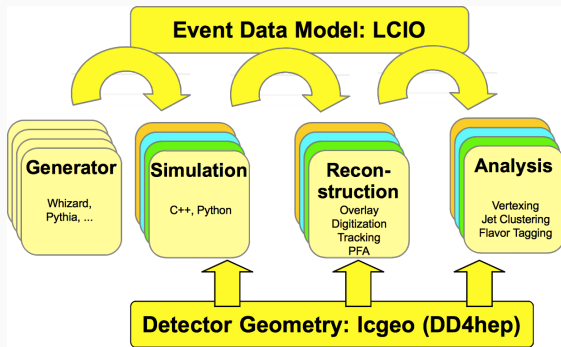
F. Gaede, G. Ganis, B. Hegner, C. Helsens,
T. Madlener, A. Sailer, G.A. Stewart, V. Volkl, J. Wang



May 18, 2021

- Event Data Model basics and EDM4hep
- podio - A generic EDM toolkit
- Latest developments and benchmarks
- Currently ongoing work and future plans

The EDM at the core of HEP software



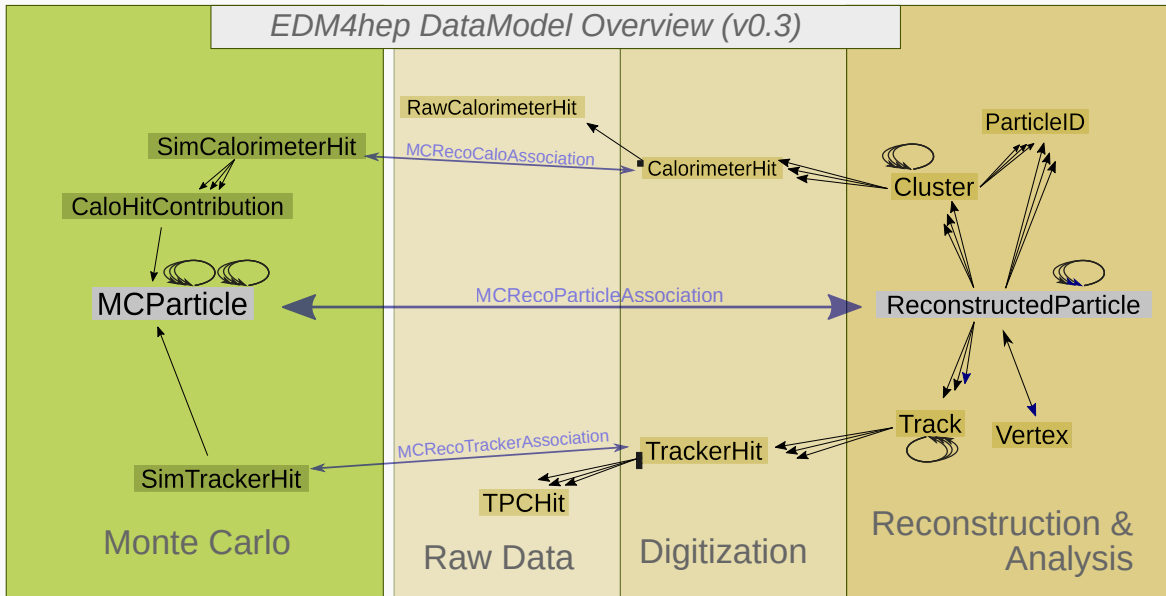
*iLCSoft components here, but general scheme applies

- 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

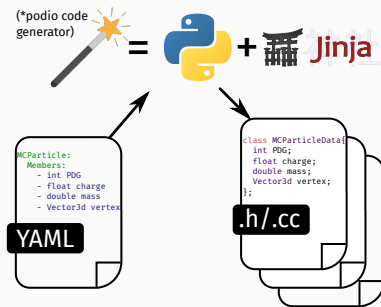
- The Key4hep project aims to define a common software stack for all future collider projects
 - see [A. Sailer, Key4hep: Status and Plans](#) (SW parallel session, Tue afternoon)
- EDM4hep is the common EDM that can be used by all communities in the Key4Hep project
 - ILC, CLIC, FCC-ee & FCC-hh, CEPC, ...
- 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 and FCC-edm

EDM4hep schema



podio as generator for EDM4hep

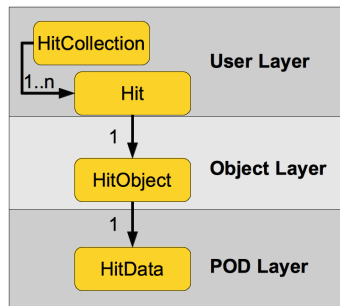
- Original HEP c++ EDMs are heavily Object Oriented
 - Deep inheritance structures
 - Thread-safety can be hard
 - Objects scattered in memory
- Data access can be slow with these approaches
- Use **podio** to generate thread safe code starting from a high level description of the desired EDM
 - Users are isolated from implementation details
 - Target different I/O backends for persistency
- Provide an easy to use interface to the users
 - Users should not need to worry about resource management
 - Treat python as first class citizen and allow “pythonic” usage



 [AIDASoft/podio](https://github.com/AIDASoft/podio)

The three layers of podio

- podio favors **composition** over inheritance and uses **plain-old-data (POD)** types wherever possible
- **User Layer** consists of handles to the EDM objects and offers the full functionality
- The **Object Layer** handles resources and references to other objects
- The actual PODs live in the **POD Layer**
- Layered design allows for efficient memory layout and performant I/O implementation
 - ROOT I/O is used by default
 - An SIO based has recently been added



Automatic code generation

```
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"
```

- Reusable components
- Fixed sized arrays as members
- 1 – 1 and 1 – N relations
- Additional user-provided code

- Validation and consistency check before code generation
 - Members can only be fundamental types or PODs
 - Relations only possible within the defined EDM
- Completely reworked. Now using easy to extend `jinja2` templates

podio - core features

- Support for variable size *VectorMembers*
 - Break the “POD-ness” of the datatypes
 - Same restrictions as for other data members
- Value semantics in c++
- C++17 compliant code generation
- Easy to use python interface via PyROOT

VectorMembers in yaml definition

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

C++ usage examples

```
// Vector members usage  
auto pid = ParticleID();  
pid.addToParameters(3.14);  
for (auto p : pid.getParameters()) { /**/ }  
  
// General usage of datatypes  
auto coll = MCParticleCollection();  
auto mc = coll.create();  
mc.setMass(3.096);  
  
for (auto p : mc.getParents()) {  
    const auto mass = p.getMass();  
}
```

python usage examples

```
store = EventStore('events.root')  
  
for event in store:  
    particles = event.get('MCParticles')  
    for p in particles:  
        print(p.getMass())
```

CMake interface for projects using podio

```
find_package(PODIO)

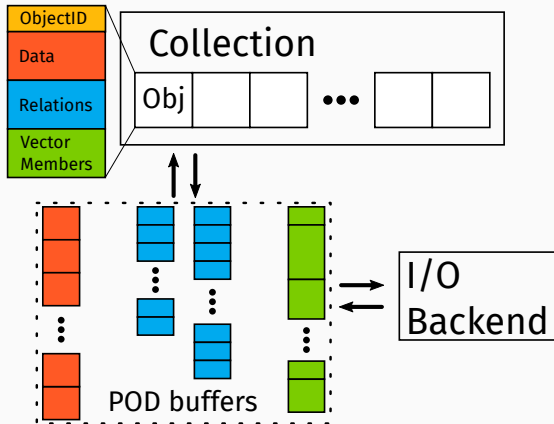
# generate the c++ code from the yaml definition
PODIO_GENERATE_DATAMODEL(edm4hep edm4hep.yaml headers sources IO_BACKEND_HANDLERS "ROOT;SIO")
# compile the core data model shared library (no I/O)
PODIO_ADD_DATAMODEL_CORE_LIB(edm4hep "${headers}" "${sources}")
# generate and compile the ROOT I/O dictionary
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 (searching LD_LIBRARY_PATH)

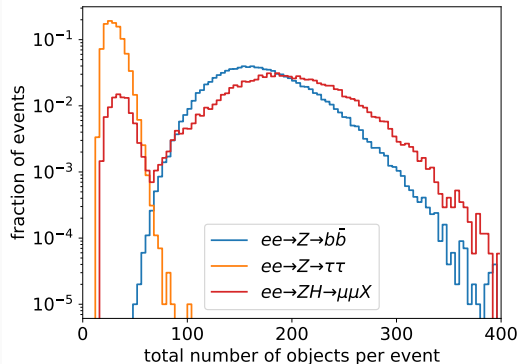
The I/O in podio

- I/O operations are based on collections
- All *Data* PODs are read/written as one array of PODs
- Relations are persisted via arrays of *ObjectIDs*
- *VectorMembers* are concatenated into one array per member
- I/O backends only have to be able to read and write arrays of PODs. The rest is handled by podio



I/O backends benchmarks using EDM4hep

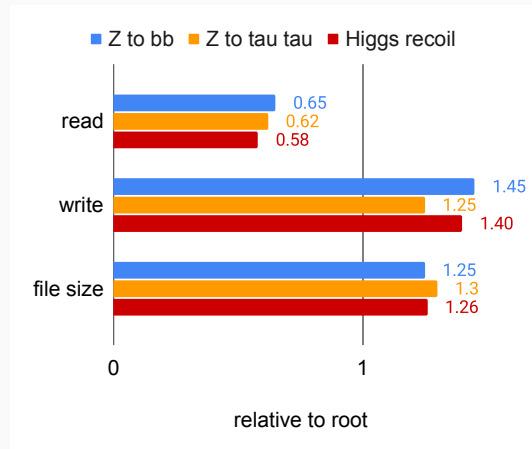
- Default **ROOT** based backend
 - Buffers are stored as branches in a **TTree**
- Backend based on **SIO** also available
 - Persistency library used in LCIO
 - Writes binary records of complete events
 -  [iLCSoft/SIO](https://github.com/iLCSoft/SIO)
-  [key4hep/k4SimDelphes](https://github.com/key4hep/k4SimDelphes) to generate “realistic” EDM4hep benchmark data
- Use podio benchmarking tools focussing on I/O times



- Z decays @ FCC-ee $\sqrt{s} = 91$ GeV
- “Higgs recoil” @ ILD $\sqrt{s} = 250$ GeV

Benchmark results

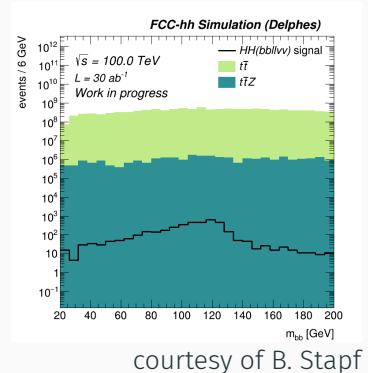
- SIO backend can read files faster
- ROOT backend can write files faster
- ROOT files are smaller than SIO files
- Semi-automatic benchmark setup caught regression with ROOT 6.22/06
 - Quickly identified with the help of ROOT developers
 - Overall improved implementation with 20–30 % speed-up
- Reading of partial event contents lead to speed up for ROOT but not SIO
 - SIO backend stores complete events



podio@4b10456, ROOT v6.22/08, SIO v00-01

Current status and next steps for EDM4hep

- LCIO and FCC-edm mainly used for lepton collider physics studies
- Key4hep and EDM4hep should support all kinds of future collider studies
 - First studies in context of FCC-hh are on their way
- Finalize EDM4hep schema
 - Main entities and relations well defined
 - Handling of “generic user data” under discussion
- Migration of existing software to Key4hep stack
 - FCC-edm was podio based. FCCSW migration almost done
 - Marlin processor wrapper with on-the-fly conversion between LCIO and EDM4hep is available



 [key4hep/k4MarlinWrapper](https://github.com/key4hep/k4MarlinWrapper)

Schema evolution



- Long overdue on our TODO list
- Allow to read older versions of an EDM and convert “on-the-fly”
 - Only deal with the current version in memory
- Automatic conversion code generation
 - User defined conversions for non-trivial changes (e.g. change of coordinate system)
- Work has started and this is currently **our top priority**

Next steps and future plans for podio

“Flat data formats”

- Data PODs trivially usable
- Relation handling is cumbersome
- Ongoing work (investigating *RNTuple*, provide utilities, ...)

“Reference collections”

- Non-owning collections that reference objects in other collections
- Well used feature in LCIO EDM

Building blocks for core functionality

- Consolidate standalone and framework implementation

```
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()
```

Usage with heterogenous resources

- PODs based design should help
- Collecting possible use and benchmark cases

More benchmarks

- Fully automatize setup
- Also look at non-I/O parts

Summary

- LCIO and FCC-edm inspired EDM4hep has become quite mature and is used already for physics studies by the ILC, CLIC, FCC and CEPC communities
 - Feedback is very welcome. **Give it a go and tell us what needs to be improved**
 - Tue 09:00 AM CET, alternating Key4hep/EDM4hep: indico.cern.ch/category/11461/
- Many “under the hood” developments in podio for better usability and maintainability
- Integrated SIO as a second I/O backend
- Semi-automatic setup can be used for benchmarking I/O performance
 - Helped us discover and fix a performance regression
- Started to tackle long standing issue of missing schema evolution
- No shortage of work in the near future



Backup

Pointers to Resources

- Key4HEP

key4hep.github.io/key4hep-doc

 [key4hep](https://github.com/key4hep)

- EDM4HEP

 [key4hep/EDM4hep](https://github.com/key4hep/EDM4hep)

cern.ch/edm4hep

- k4SimDelphes

 [key4hep/k4SimDelphes](https://github.com/key4hep/k4SimDelphes)

 [delphes/delphes](https://github.com/delphes/delphes)

cp3.irmp.ucl.ac.be/projects/delphes

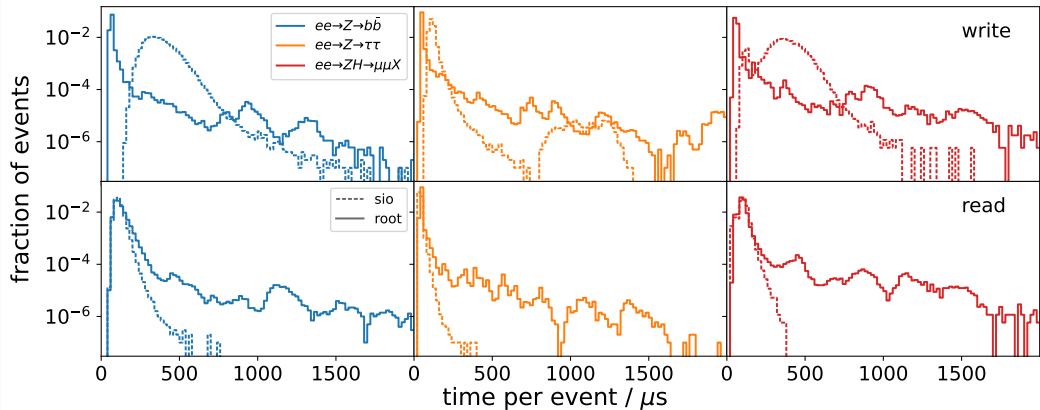
- podio

 [AIDASoft/podio](https://github.com/AIDASoft/podio)



xkcd.com/138

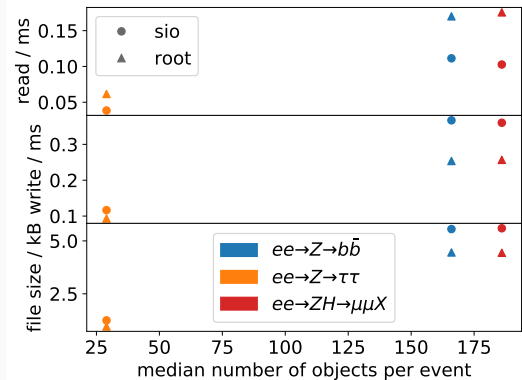
Per event benchmark distributions



- ROOT backend features long tails in per event read and write times
 - This is the reason for slower total read times compared to SIO
 - Might be controllable with non-default settings. But phase space is large
- SIO backend times seem to generally scale with the size of the event

Scaling with event content

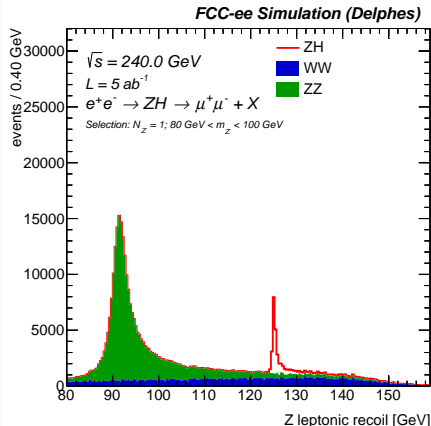
- No unexpected behavior vs event size in per event read and write times or resulting file sizes
- More data points necessary for establishing anything more empirical



k4SimDelphes - First steps towards physics

- k4SimDelphes uses delphes to do the simulation and reconstruction and creates output files in EDM4HEP format
- Quick way to get your hands dirty and do some physics with EDM4HEP
- Integrated into Key4hep framework
- Available as standalone executables
 - E.g. DelphesPythia8_EDM4HEP, DelphesSTDHEP_EDM4HEP, ...
- Part of a coherent approach to generation / simulation in Key4HEP
 - Ideally no difference between the different approaches to simulate detector response

 [key4hep/k4SimDelphes](https://github.com/key4hep/k4SimDelphes)



courtesy of C. Helsens