KEY4HEP & EDM4HEP - Common Software for Future Colliders

CLICdp Monthly Meeting - 06.04.2020

Valentin Volkl (CERN)

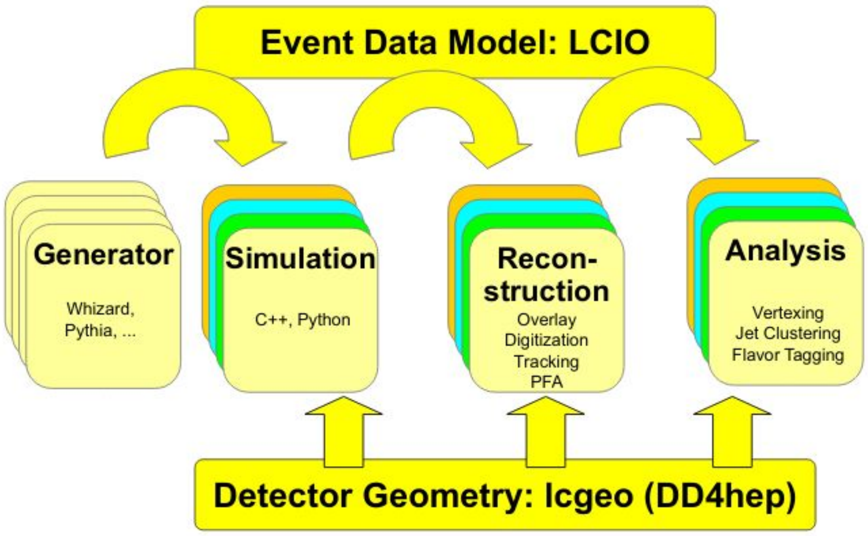
Table of Contents

Key4HEP Motivation

- Future detector studies critically rely on **well-maintained software stacks** to model detector concepts and to understand a detector's limitations and physics reach
- We have a scattered landscape of **specific software tools** on the one hand and **integrated frameworks** tailored for a specific experiment on the other hand
- Aim at a low-maintenance common stack for FCC, ILC/CLIC, CEPC with ready to use "plug-ins" to develop detector concepts
- Reached consensus among all communities for future colliders to develop a **common turnkey software stack** at recent Future Collider Software Workshop
- Identified as an important project in the CERN EP R&D initiative

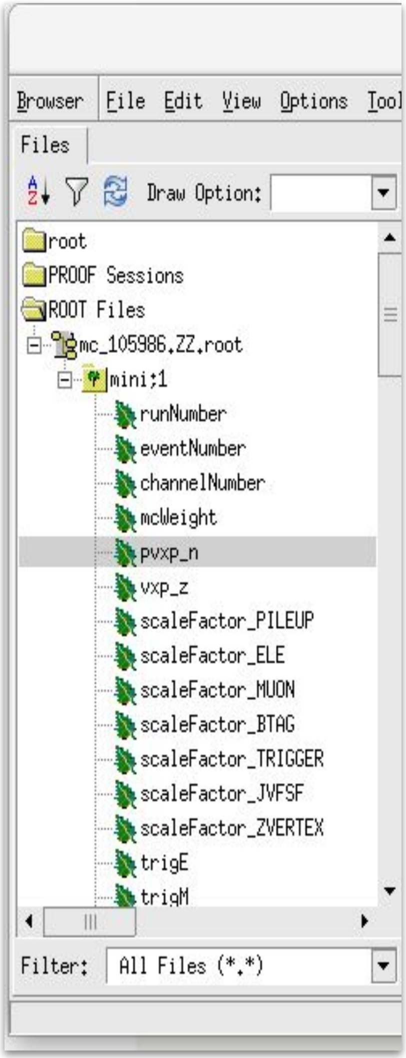
Transition to Key4HEP: Adiabatic Changes

- While transitioning to DD4hep, need to be able to keep running the reconstruction
- Switch components one by one, validate changes
  - Geometry provided by DD4hep, no changes needed
  - Move framework from Marlin to Gaudi: wrap existing processors
  - Move from LCIO to EDM4hep
  - Replace wrapped processors with native Gaudi Algorithms
  - Provide installations (Spack)



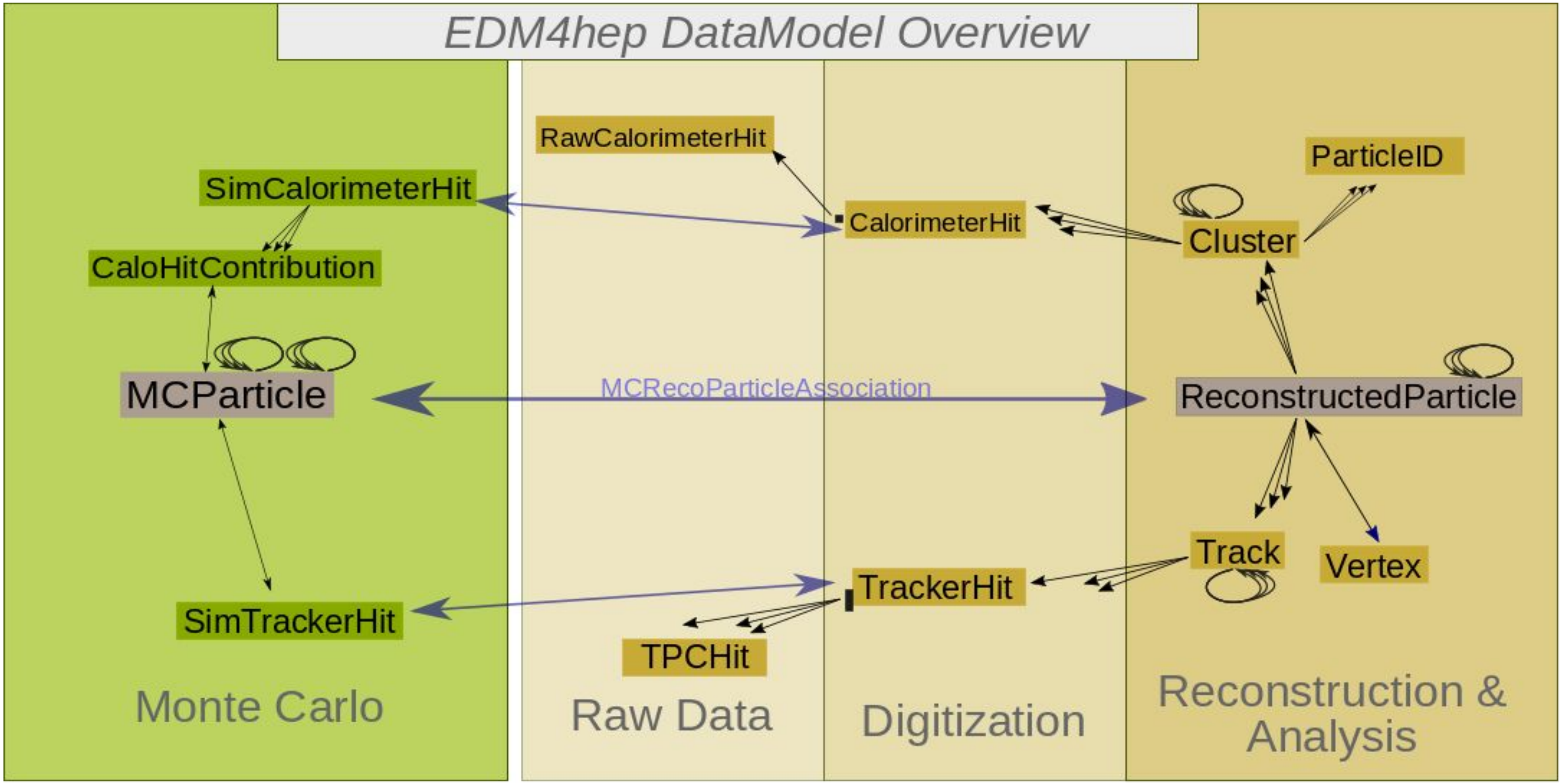
EDM4HEP - Introduction

- Event Data Model:
  - Describes structure of HEP Data:
    - definitions of **objects** and how they are **grouped**
    - **technical** implementation of persistency and processing
- Can be as simple as “Branch names in ROOT file”
  - But more sophisticated solutions can:
    - provide an **application programming interface** for HEP software
    - aid developers in writing more **efficient code**
    - enable **collaboration**



Relation Diagram

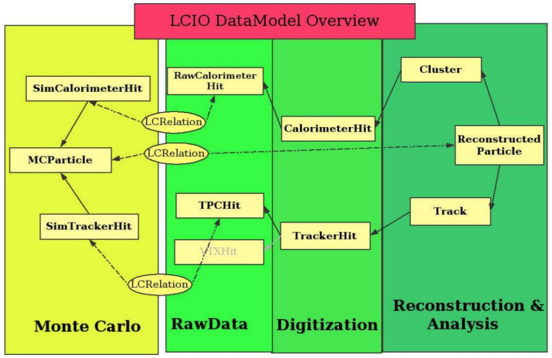
Code Reference under https://cern.ch/edm4hep

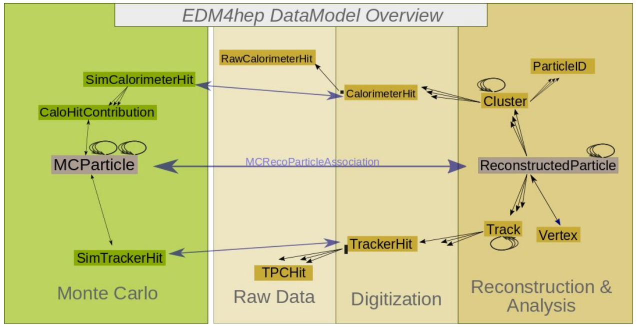


Currently (for the next few weeks) available as a beta version before use in production

Differences LCIO-EDM4hep

- Technical implementation with PODIO
  - Via PLCIO (F. Gaede)
- LCRelations replaced by Associations
- Use of unsigned long for CellIDs
  - Instead of two ints
- Missing RunHeader
  - Needs new functionality in Podio, will come with next version
- LCIO→ EDM4hep converter under development by colleagues from CEPC





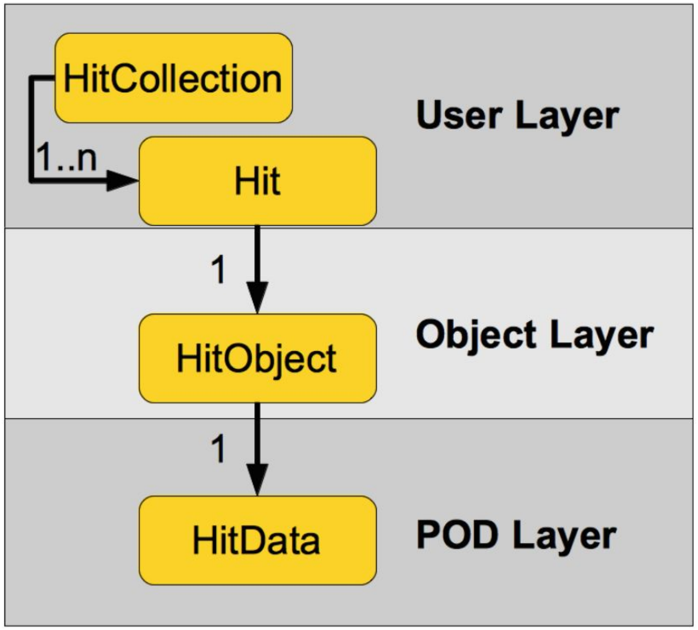
Technical: PODIO

Adapted from “Podio: recent developments in the Plain Old Data *toolkit* for HEP”

- PODIO is an Event Data Model *toolkit* for HEP
  - developed in the Horizon2020 project AIDA2020
  - based on the use of **PODs** for the event data objects (Plain Old Data objects)
- PODIO originally developed in the context of the FCC study
  - addressing the problem of creating an EDM in a generic way
  - EDM described in yaml, C++ code auto-generated
  - Allowing potential re-use by other HEP groups
- PODIO is used since its first release by the FCC studies (see FCC-EDM)

PODIO Core Features

- Three **layers:**
  - **User layer** (API): collections of EDM object handles, *HitCollection*
  - **Object layer:** transient objects (HitObject)
  - **POD layer**: persistent information
- Clear **ownership**: objects owned by EventStore are persisted, other objects ref-counted
- Python as a first class citizen
- Different **I/O** implementations, but currently only ROOT



Quick access

How to process a edm4hep ROOT file:

With TTree::Draw:

```
events->Draw(“MCParticles.momentum.x”)
```

With ROOTDataFrame:

```
ROOT::RDataFrame df("events", "edm4hep_events.root");
auto df2 = df.Define("MCParticles_pt", edm4hep::pt, {"MCParticles"});
auto h = df.Histo1D(“MCParticles_pt”); h->Draw();
```

Quick access

How to process a edm4hep ROOT file:

With PODIO EventStore: (link to complete example)

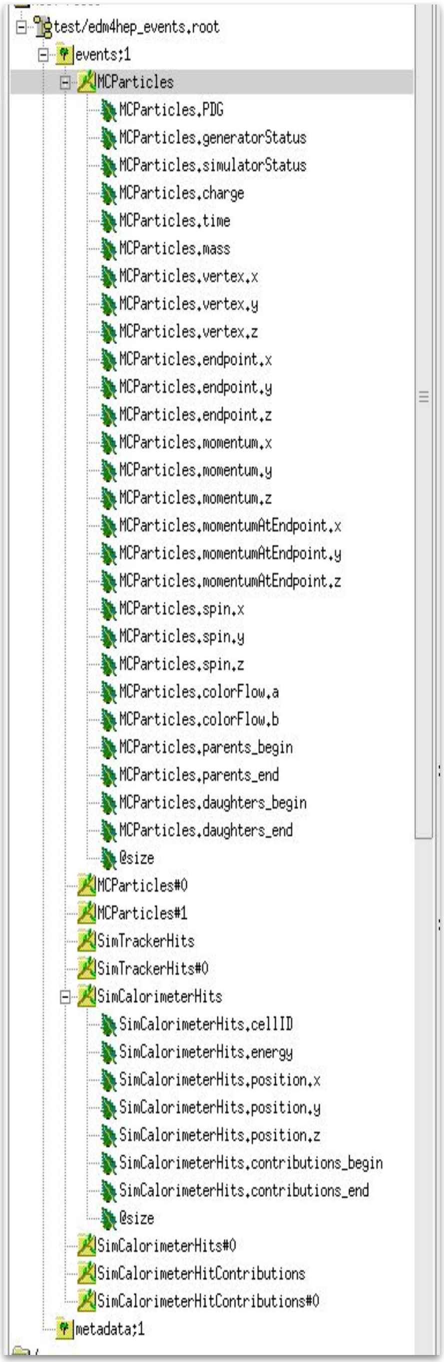
```
auto reader = podio::ROOTReader();
auto store = podio::EventStore();
reader.openFile("edm4hep_events.root");
store.setReader(&reader);
auto& mcps = store.get<edm4hep::MCParticleCollection>("MCParticles");
auto mcp1 = mcps[0];
auto mcp1_daughter = mcp1.getDaughters(0);
...
```

Quick access

How to process a edm4hep ROOT file:

With PODIO EventStore, Python:

```
from EventStore import EventStore
store = EventStore("edm4hep_events.root")
for i, event in enumerate(store):
  particles = store.get("MCParticles")
  for p in particles:
    print p.momentum()
```



Gaudi/Marlin Wrapper

Apart from some naming conventions, very similar ideas in the two frameworks

	Marlin	Gaudi
language	C++	C++
Working unit	Processor	Algorithm
Configuration Language	XML	Python
Set-up function	init	initialize
Working function	process	execute
Wrap-up function	end	finalize
Transient Data Format	LCIO	anything

Converter from Marlin to Gaudi steering file available

- To start using Gaudi: use a generic wrapper around the processors
- Prototype: https://github.com/andresailer/GMP
- Read LCIO files and pass the LCIO::Event to our processors
- Currently working on moving the MarlinWrapper from a proof of concept to being more widely usable

Key4HEP Core Framework components

Meanwhile, developments on core functionality of the Gaudi-based framework:

- K4FWCore:
  - Data Service for Podio Collections
  - Overlay for backgrounds
  - https://github.com/key4hep/K4FWCore
- K4-project-template
  - Template repository showing how to build new components on top of the core Key4HEP framework
  - https://github.com/key4hep/k4-project-template

Software Infrastructure

- Regular meetings
  - https://indico.cern.ch/category/11461/
- Docpages
  - https://cern.ch/key4hep (main documentation site))
  - https://cern.ch/edm4hep (doxygen code reference)
- Modern CMake Configuration
- Automated Builds and Continuous Integration
  - Use of SPACK package manager
- Distribution via CVMFS
  - /cvmfs/sw.hsf.org/
  - /cvmfs/sw-nightlies.hsf.org

CVMFS directory tree

```
/cvmfs/sw.hsf.org/key4hep/
|-- releases/ $LCG_version / $platform / $pkgname-$spackhash / (bin  … )
|-- views   / $K4_version  / $platform / (bin include share … init.sh)
|-- setup.sh
|-- contrib

/cvmfs/sw-nightlies.hsf.org/key4hep/
|-- nightlies/ $timestamp / $platform / $pkgname-$spackhash / (bin  … )
|-- views   / $timestamp  / $platform / (bin include share … init.sh)
|-- setup.sh
|-- contrib
```

CVMFS directory tree:

Try it out on lxplus:

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

And use it to run a simulation:

```
ddsim --compactFile
/cvmfs/sw-nightlies.hsf.org/key4hep/views/latest/x86_64-cent
os7-gcc8-opt/DDDetectors/compact/SiD.xml -N 10 -G
--gun.particle pi+ --outputFile my_edm4hep.root
--part.userParticleHandler=''
```

Spack for Key4HEP

- Spack is a package manager
  - Does not replace CMake, Autotools, …
  - Comparable to apt, yum, homebrew, ...
    - But not tied to operating system
    - And no central repository for binaries!
- Originally written for/by HPC community
  - Emphasis on dealing with **multiple configurations** of the same packages
    - Different versions, compilers, external library versions …
    - … may coexist on the same system
  - `Spec:` Syntax to describe package version configuration and dependencies
- Repository added with Key4HEP package recipes

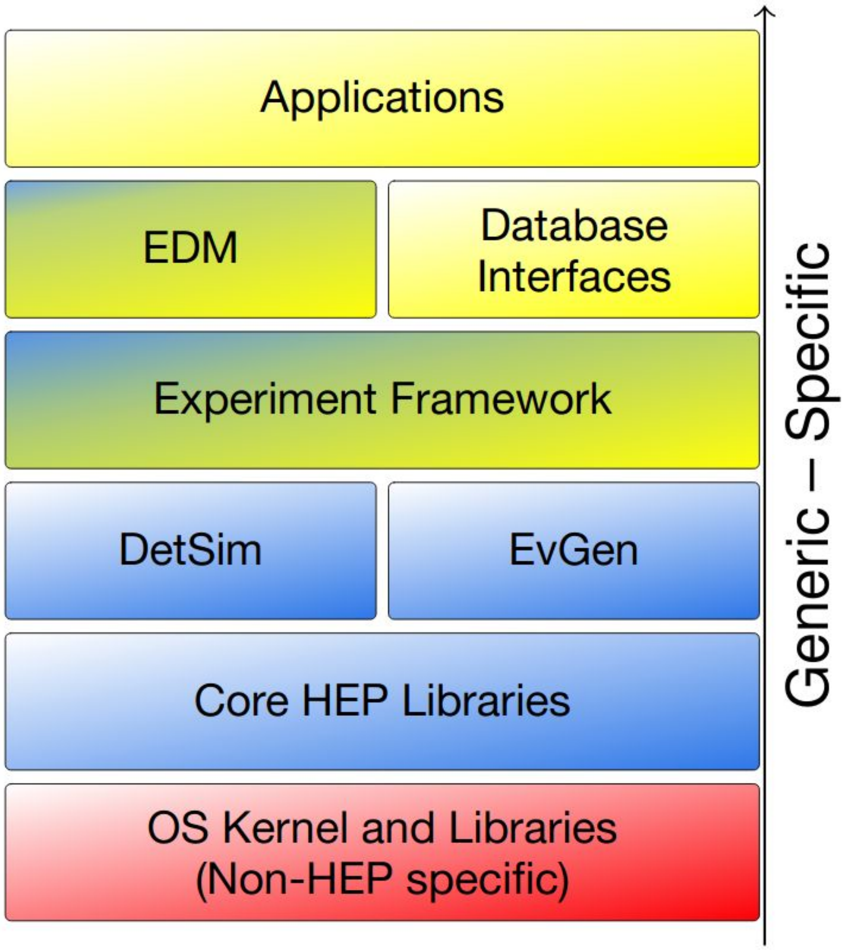
```
git clone https://github.com/spack/spack.git
git clone https://github.com/key4hep/k4-spack.git
alias spack='python $PWD/spack/bin/spack'
spack repo add k4-spack
# install the meta-package for the key4hep-stack
spack install key4hep-stack
```

Conclusion

- Given the general agreement on moving to a common HEP software stackfrom future experiments
- Support joint developments between STC/SCT, and also FCC, ILC/CLIC,muon collider, CEPC
- Common detector geometry descriptions in DD4HEP
- Common event data model EDM4HEP
- Glue it all together with Gaudi in KEY4HEP

A typical HEP Software Stack

- Interfaces to tracking and reconstruction libraries (PandoraPFA, ACTS
- (More or less) experiment specific event datamodel libraries
- Experiment core orchestration layer, whichcontrols everything else: Marlin, Gaudi,CMSSW, AliRoot
- Packages used by many experiments: DD4hep, Pythia, . . .
- Usual core libraries (ROOT, Geant4, CLHEP,. . .)
- Non-HEP libraries: boost, python, cmake. . .



Interoperability

- Level 0 - Common Data Formats
  - Allows interoperability between different programs, even running on different hardware
  - E.g.: HepMC event records, LCIO, GDML, ALFA Messages
- Level 1 - Callable Interfaces
  - Basic calling interfaces defined by the programming languages, language calls possible
  - Can be dependent on the compiler and language version
  - Details are important: error/exception handling, thread safety, dependencies, runtime setup
- Level 2 - Introspection Capabilities
  - Software elements to facilitate the interaction of objects in a generic manner: Dictionaries, Scripting interfaces
  - E.g.: PyROOT to interact with any ROOT (C++) class via the python interpreter
- Level 3 - Component Model
  - Software components of a common framework offer maximum re-use
  - Standard way to configure components, logging, object lifetime and ownership, plug-in mechanism
  - Requires adoption of single framework

The right interoperability point between packages varies, but choosing it correctly provides great quality of life for developers and users