KEY4HEP & EDM4HEP - Common Software for Future Colliders

CLIC Detector & Physics autumn collaboration meeting - 2020/10/02

Valentin Volkl (CERN), Placido Fernandez (CERN), Andre Sailer (CERN)

Table of Contents

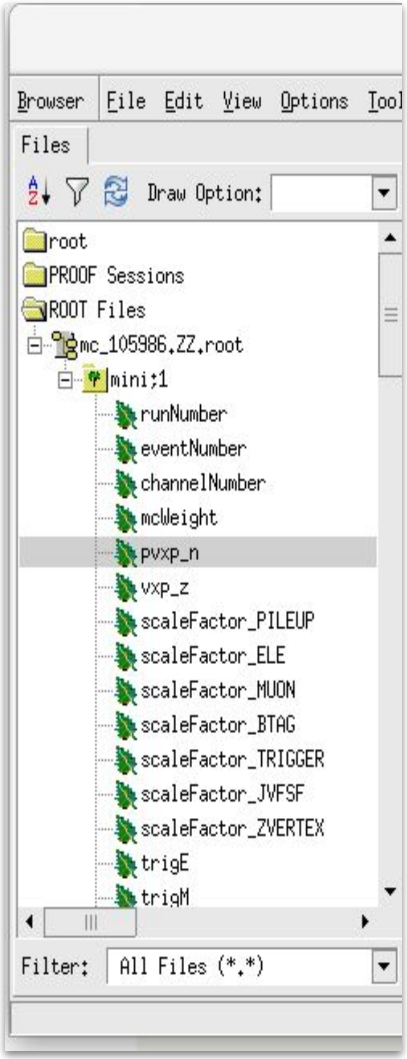
- **Key4HEP - Introduction and motivation**
- **EDM4HEP - Common Data Model Status**
- **Common Gaudi Framework Status**
- **Software Infrastructure and Organisation**
- **Packaging: Spack for Key4HEP**
- **GaudiMarlinWrapper**

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

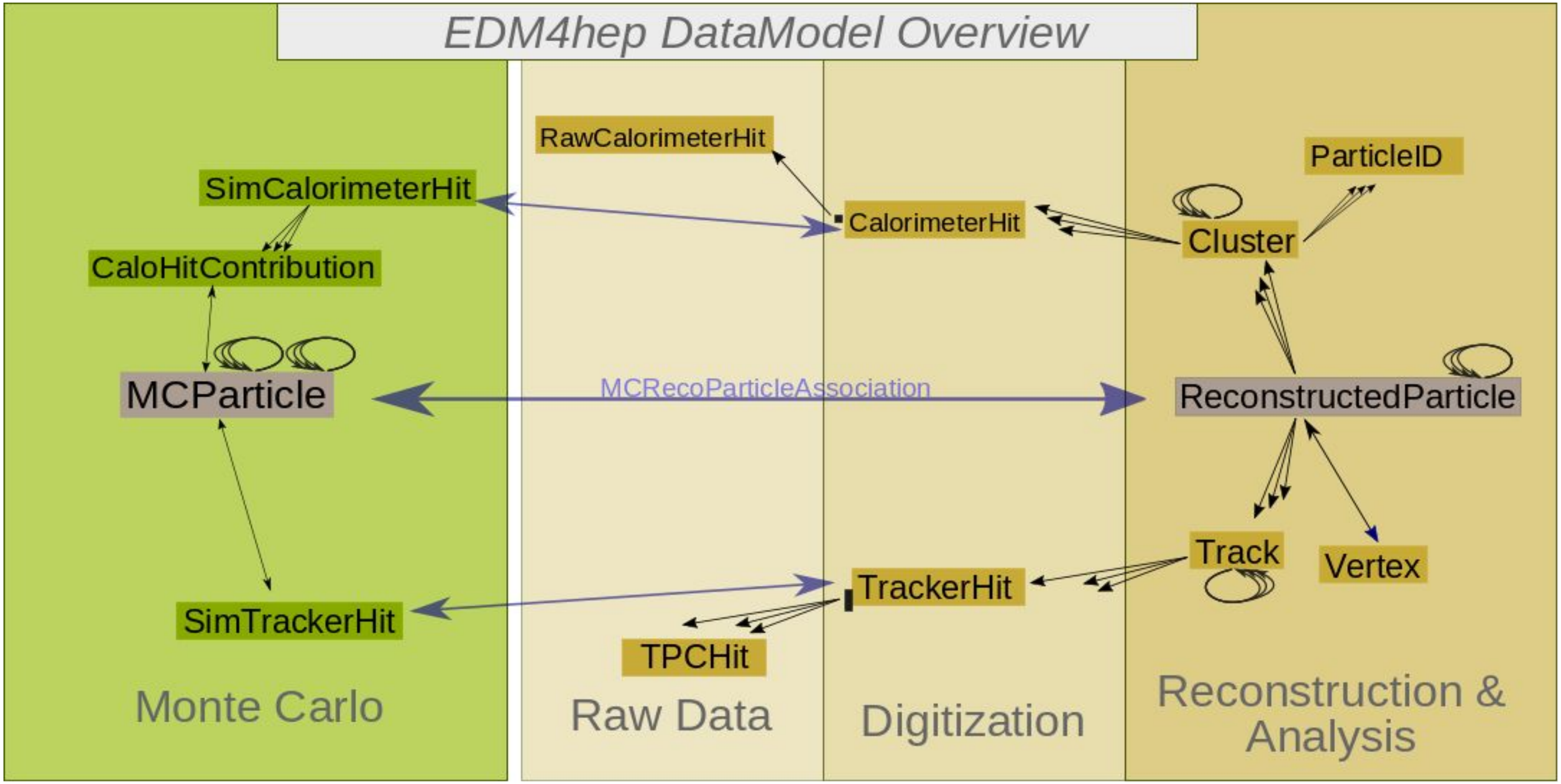
EDM4HEP - Introduction

- 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**



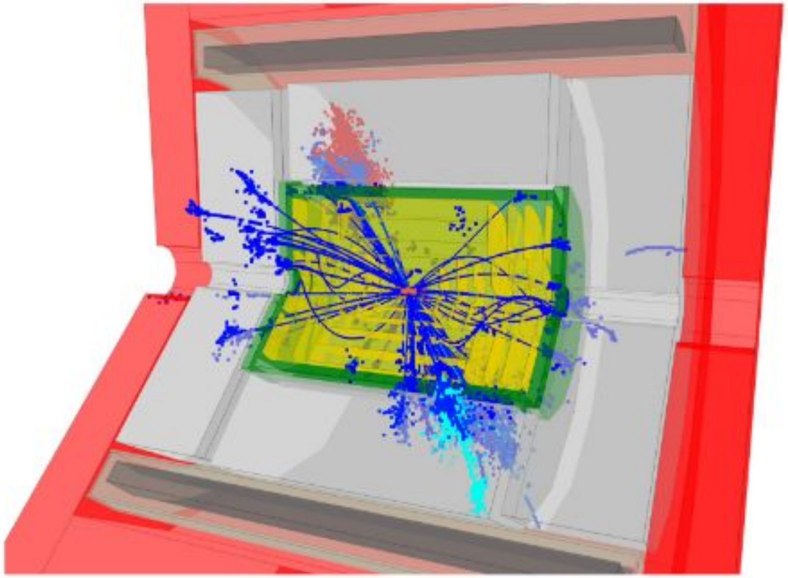
Relation Diagram

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



Applications: DDSim

DDSim (Standalone Geant4 simulation tool in DD4hep) can now produce EDM4hep files:



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

ddsim \
 --compactFile ${DD4hepINSTALL}/DDDetectors/compact/SiD.xml \
 --gun.particle pi+ \
 --part.userParticleHandler='' \
 --outputFile output_edm4hep.root
```

Application: DelphesEDM4HEP

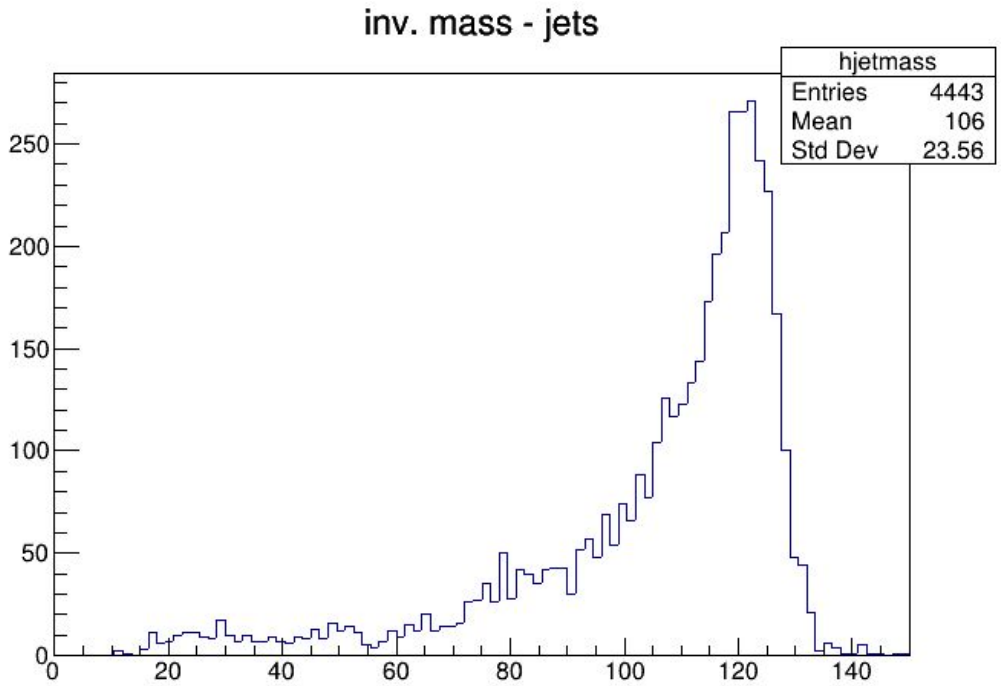
Second Plugin for Delphes output recently added:

- ◈ `DelphesPythia8_EDM4HEP`
- ◈ `DelphesSTDHEP_EDM4HEP`
- ◈ `DelphesHepMC_EDM4HEP`
- ◈ `DelphesROOT_EDM4HEP`

Adds executables like standard Delphes, outputting directly to EDM4HEP.

Higgs Recoil Analysis

example: [Link]



Key4HEP Framework

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

- Ongoing Work to collaborate more with Gaudi ecosystem (Gaussino)
- Add ACTS components

FCCSW - Key4HEP transition

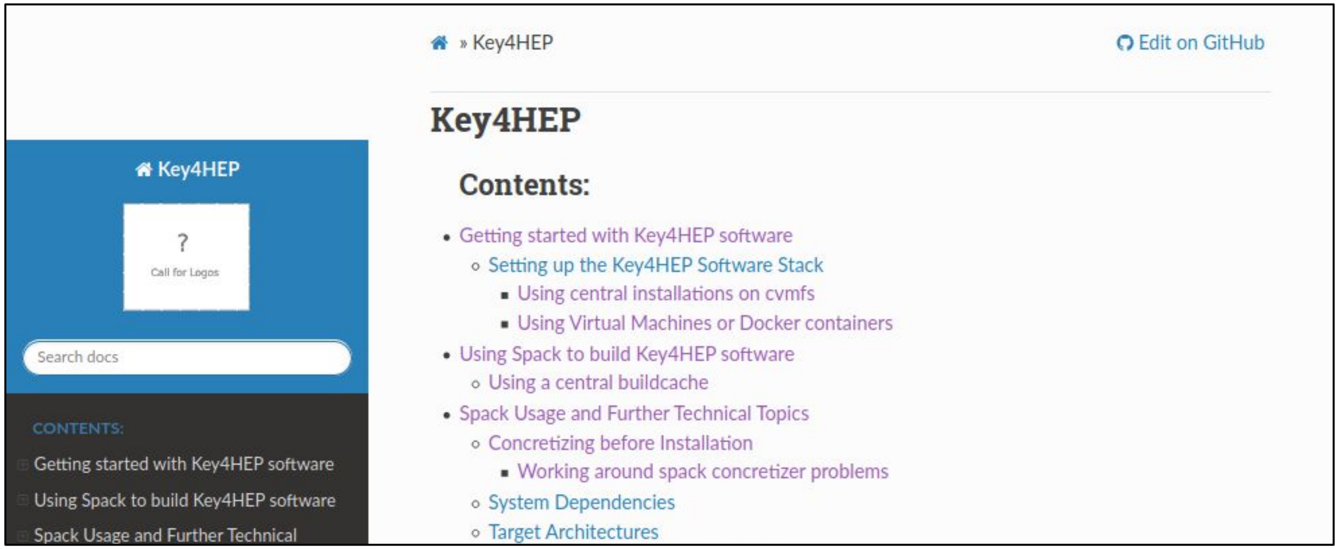
- Already Gaudi- and Podio based, so little technical challenges

```
gaudi_project(GMPWrapper  v0r1

          USE K4FWCore v0r2

          USE Gaudi v34r0 )
```

- Event model has a fairly straightforward correspondence
- Still: Many files need to be touched
  - Not yet clear if radical or soft (converter-based) update preferred
- Biggest hurdle: touching all components brings technical debts to light.

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)



» Key4HEP

Edit on GitHub

Key4HEP

Contents:

- Getting started with Key4HEP software
  - Setting up the Key4HEP Software Stack
    - Using central installations on cvmfs
    - Using Virtual Machines or Docker containers
- Using Spack to build Key4HEP software
  - Using a central buildcache
- Spack Usage and Further Technical Topics
  - Concretizing before Installation
    - Working around spack concretizer problems
  - System Dependencies
  - Target Architectures

Key4HEP
?
Call for Logos
Search docs
CONTENTS:
Getting started with Key4HEP software
Using Spack to build Key4HEP software
Spack Usage and Further Technical

- Modern CMake Configuration
- Automated Builds and Continuous Integration
  - Use of SPACK package manager
- Distribution via CVMFS

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
source spack/share/spack/setup-env.sh
spack repo add k4-spack
# install the meta-package for the key4hep-stack
spack install key4hep-stack
```

Some Experiences

- Collaboration with Spack developers fairly smooth
  - Some HEP colleagues have merging rights on the spack repo
  - Some HEP packages actively maintain their package recipes (ACTS!)
- Rapid pace of changes in upstream repository
  - Stable builds will need to pin the spack version used.
  - But miss out on the latest features.
- Spack developers very responsive, but roadmap sometimes a bit opaque:
  - The *concretizer* developments have been much delayed
- The recipes are very nice to persistify build system know-how

```
conflicts("%gcc@8.3.1",
     msg="There are known issues with compilers from redhat's devtoolsets" \
    "which are therefore not supported." \
    "See https://root-forum.cern.ch/t/devtoolset-gcc-toolset-compatibility/38286")
```

- Parallel builds not yet attempted

Spack: use for developers

- Use in developing software is pushing spack’s intended purpose, but possible. Options:
    - Spack can build from branches.
    - Build can be done “as usual” after spack load / spack build-env
    - spack dev-build compiles local code according to the spack recipe
- Need to include build tools - would be nice to offload the build of these packages on LCG-releases

Towards the full LCG releases

- Ivan (SFT) has added a tremendous amount of packages - maybe 70% of packages included in the lcg releases already available in spack
- Key4HEP installation can be used as a test-bed

CVMFS directory tree

Already mounted in most places

```
/cvmfs/sw.hsf.org/key4hep/
|-- spackages /  $platform / $compiler / $pkgname-$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
```

Used to test some new cvmfs features

CVMFS directory tree

```
/cvmfs/sw.hsf.org/key4hep/
|-- spackages /  $platform / $compiler / $pkgname-$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
```

Contains some 300 packages

- 60 Experiment-specific
- 50 HEP-specific
- 200 System/General Purpose

14 GB install size, some 6h to build on single 4-core machine

CLICSoft transition to Key4hep - GaudiMarlinProcessor Update (Placido Fernandez, Andre Sailer)

GMPWrapper

- The Gaudi-Marlin-Processors Wrapper project brings *Marlin* functionality to the *Gaudi* framework, smoothly.
- It creates interfaces *(wraps)* around Marlin Processors, encapsulating them in Gaudi Algorithms.
- Current Marlin source code is kept intact, and it is just called on demand from the Gaudi Framework.

	Marlin	Gaudi
Language	C++	C++
Working Unit	Processor	Algorithm
Config Language	XML	Python
Set-up function	init	initialize
Working function	process	execute
Wrap-up function	end	finalize
Transient Data Format	LCIO	Anything / EDM4hep

GMPWrapper now

- Bugs were fixed, a manual (`README.md`) was included with instructions to compile, configure, run and test.
- Updated and modernization of the code base.
- Running examples are included as tests.
- A recipe to build it with Spack is also part of the *k4-spack* repo.
- It was included as part of Key4hep, moving there the repo:
  - https://github.com/key4hep/GMP
- CI is now included with GitHub Actions, checking syntax (`clang-format`), and running two basic functionality tests.

Dependencies

- GMP Wrapper can be built against an iLCSoft installation + Gaudi, Main dependencies:
  - **Gaudi**: to wrap Marlin processors and run the algorithms.
  - **Marlin**: to run the underlying processors
    - It will eventually disappear when only Gaudi Algorithms are used
  - **LCIO**: Event Data Model input/output
    - Can be changed, for EDM4hep i.e.
- Other dependencies:
  - **ROOT**, **Boost**
- Or simply:
  - `spack install key4hep-stack`

GMP Wrapper configuration and running

- Configuring and running the wrapper is done as in Gaudi, through a Python File
  - An algorighm is filled with wrapped Marlin Processors.
  - Processor parameters are defined for each instance, defining the Marlin processor to load a list of parameters of values
    - Converter for Marlin XML configuration files exists
- On algorithm initialization of a Marlin Processor, MARLIN_DLL environment variable is used to load the necessary libraries

GMP configuration example

```
digiVxd = MarlinProcessorWrapper("VXDBarrelDigitiser")
digiVxd.OutputLevel = DEBUG
digiVxd.ProcessorType = "DDPlanarDigiProcessor"
digiVxd.Parameters = [
    "SubDetectorName", "Vertex", END_TAG,
    "IsStrip", "false", END_TAG,
    "ResolutionU", "0.003", "0.003", "0.003", "0.003", "0.003", "0.003", END_TAG,
    "ResolutionV", "0.003", "0.003", "0.003", "0.003", "0.003", "0.003", END_TAG,
    "SimTrackHitCollectionName", "VertexBarrelCollection", END_TAG,
    "SimTrkHitRelCollection", "VXDTrackerHitRelations", END_TAG,
    "TrackerHitCollectionName", "VXDTrackerHits", END_TAG,
    "Verbosity" , "DEBUG", END_TAG,]
algList.append(digiVxd)
```

CLIC reconstruction

It successfully computes the full CLIC reconstruction:

- The CLIC reconstruction computes a sequence that includes different Overlays, Digitisers, reconstruction, tracker and vertex finding algorithms.
- Using the updated converter, `clicReconstruction.xml` can be translated to `clicReconstruction.py`
- The converter add all algorithms to the list and leaves the configurable ones commented.

A full example is described in Key4hep documentation:

- https://key4hep.github.io/key4hep-doc/examples/clic.html

Future directions

- Move from LCIO to EDM4HEP.
    - Converter available in K4LCIOReader
- Replace wrapped MarlinProcceors by actual Gaudi Algorithms
    - Benefit from the different functionalities Gaudi offers
    - Use multi-threaded/functional Gaudi, for the future
    - Seamlessly integrate for other users of Key4hep
- Start using it in real scenarios to test how resilient it is.
- How to approach the transition?
- Gradual conversion from Marlin Processors to Gaudi Algorithms
- Transition to EDM4hep, before Processors conversion?
- Conversions during runtime?

Future directions

Gaudi
EDM4hep

Marlin
LCIO

Conclusion

- Key4hep: Joint software developments between ***STC/SCT, FCC, ILC/CLIC, muon collider, CEPC***
- Common detector geometry descriptions in DD4HEP
- Common event data model EDM4HEP
- Shared software installations done with Spack
- Transition to common framework (Gaudi) ongoing, GaudiMarlinWrapper finished

Testing

- Added testing with CTest:
  - Simple test that runs some Marlin Processors -> InitDD4hep -> VXDBarrelDigitiser
  - Muon.slcio is used for input, without hits
  - Second test generates an input file with ddsim
  - It runs a similar list of algorithms with actual hits
  - Output checks for regex with `INFO Application Manager Terminated successfully`

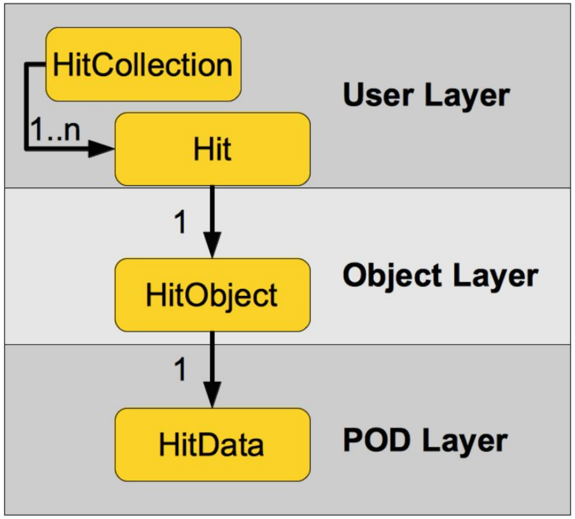
```
ddsim \
    --steeringFile $ILCSOFT/ClicPerformance/HEAD/clicConfig/clic_steer.py \
    --inputFiles $ILCSOFT/ClicPerformance/HEAD/Tests/yyxyev_000.stdhep -N 4 \
    --compactFile $ILCSOFT/lcgeo/HEAD/CLIC/compact/CLIC_o3_v14/CLIC_o3_v14.xml \
    --outputFile $GMP_tests_DIR/inputFiles/testSimulation.slcio
```

Technical: PODIO

- 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

Recent/Ongoing Developments:

- **Addition of Metadata**
- **Template engine based on Jinja2**
- **Another backend (SIO)**
- **Improved RDataFrame Interface**


HitCollection
1..n
Hit
User Layer
1
HitObject
Object Layer
1
HitData
POD Layer