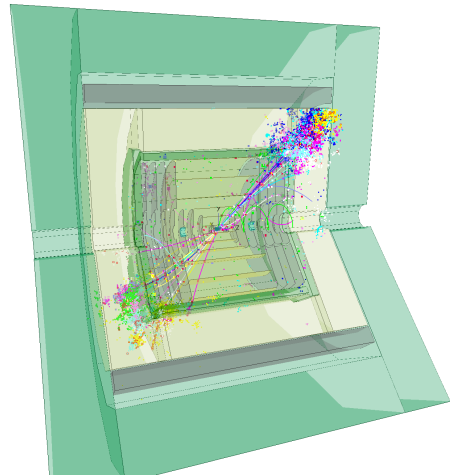
Turnkey Software Stack and Key4hep Plan of Work

André Sailer
(CERN-EP-SFT)

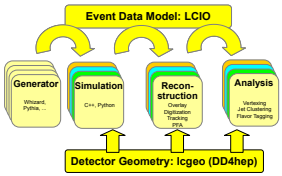
SFT Group Meeting
January 25, 2021

Table of Contents



What is Key4hep?

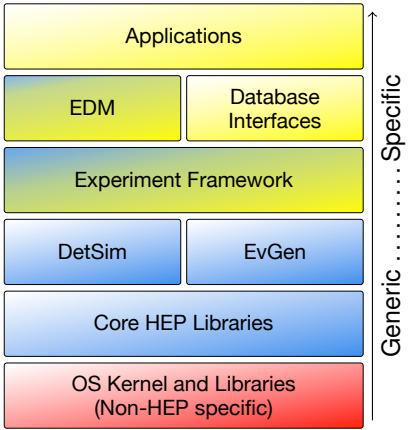
- The Vision: The Key4hep turnkey software stack connects and extends individual packages towards a complete data processing framework
  - Major ingredients: Event Data Model, Geometry Information, Processing Framework
  - Sharing common components reduces overhead for all users
  - Should be easy to use for librarians, developers, users
    - ⋆ easy to deploy, extend, set up
  - full of functionality: plenty of examples for simulation and reconstruction of detectors



iLCSoft components, but general scheme of ingredients applies

Ingredients

- Processing Framework: Gaudi v35
- Geometry Information: DD4hep
- Event Data Model: EDM4hep (based on podio)
- Adapting existing tools from iLCSoft, FCCSW, CEPCSW
- Packaging: Spack
- ROOT, Geant4, etc.



International Community

- Contributors/Interest from China, Germany, Italy, Russia, CERN
- Part of AIDAinnova
- Video Meetings: Since July 2019 regular EDM4hep meetings, since February 2020, regular Key4hep meetings
  - Tuesday, 9:00 AM CET, weekly alternating EDM4hep/Key4hep https://indico.cern.ch/category/11461/
- Github Project: http://github.com/key4hep
- Documentation: http://cern.ch/key4hep

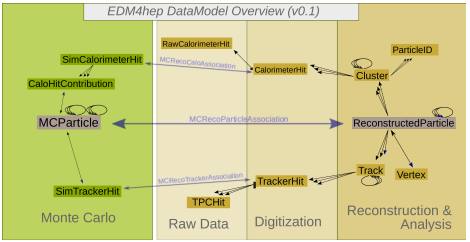
CERN EP RnD

- EP RnD work package 7.1 https://ep-rnd.web.cern.ch/topic/software/turnkey-software-stack
- Personnel
  - André Sailer
  - Placido Fernandez-Declara (CREMLINPlus)
  - Valentin Volkl

The Key4hep EDM: EDM4hep

For a high degree of interoperability, EDM4hep will provide a common event data model

- Using `PODIO` to manage the EDM (described by `yaml`) and easily change the persistency layer (ROOT, SIO, . . .)
- http://github.com/key4hep/edm4hep
- EDM4hep data model based on `LCIO` and FCC-edm implemented
- Key4hep triggered many developments for podio:
  - SIO backend
  - new templating
  - metadata
  - multi-threading investigations



Packaging with Spack

- Full Key4hep stack build with Spack
  - https://github.com/key4hep/key4hep-spack
  - Spack recipes for CEPCSW, FCCSW, iLCSoft, Key4hep
  - https://key4hep.github.io/key4hep-doc/spack-build-instructions/README.html
- Deployment to CVMFS
  - `/cvmfs/sw.hsf.org/key4hep/`
  - `/cvmfs/sw-nightlies.hsf.org/key4hep/`
- Many things learned about Spack and Spack workflow
- Presented at WLCG/HSF workshop https://indico.cern.ch/event/941278/contributions/4083868/

Simulation and Reconstruction

- Porting existing Simulation and Reconstruction tools from CEPCSW, FCCSW, iLCSoft
  - Gaudi based FCCSW adapted to EDM4hep and moving core functionality to “k4FWCore”
  - CEPC originating k4Pandora interface
  - k4MarlinWrapper to run iLCSoft Marlin Processors as Gaudi Algorithms
- EDM4hep extensions for Delphes and DD4hep

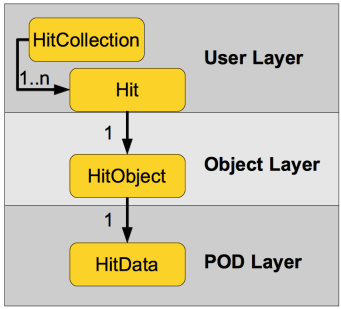
Plans for 2021

- Work on EDM4hep and podio, adapt as needed
- Packaging: work with SPI on the Spack build stacks
- Adapt iLCSoft algorithms to Key4hep (Gaudi, EDM4hep)
  - On-the-fly and in memory conversion between LCIO ⇆ EDM4hep to
  - Run LCFIPlus Flavour Tagging based on Delphes output
  - Run CLIC reconstruction inside the Gaudi framework, starting with wrapper, then going to native algorithms
  - Use adapted algorithms also in FCC reconstruction (e.g., PandoraPFA)
- Adapt FCCSW to Key4hep
- Integrate other EP RnD software projects
- Integrated with a distributed computing solution (iLCDirac) for validation and physics studies

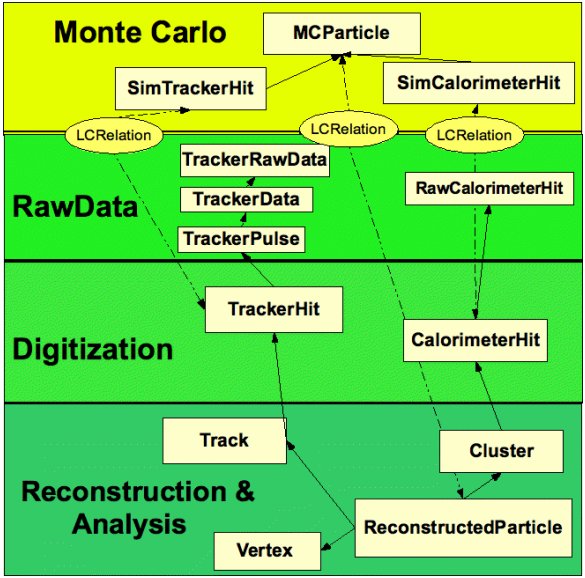
Backup Slides

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 first class citizen
- Different I/O implementations, but currently only ROOT



LCIO


Monte Carlo
MCParticle
SimTrackerHit
SimCalorimeterHit
LCRelation
LCRelation
LCRelation
TrackerRawData
TrackerData
TrackerPulse
RawCalorimeterHit
RawData
TrackerHit
CalorimeterHit
Digitization
Track
Cluster
Reconstruction &
Analysis
ReconstructedParticle
Vertex

Marlin & Gaudi

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	`processEvent`	`execute`
wrap up function	`end`	`finalize`
Transient data format	`LCIO`	anything

- To start using Gaudi: use a generic wrapper around the processors.
- https://github.com/key4hep/k4MarlinWrapper
- Read LCIO files and pass the `LCIO::Event` to our processors

*Of course subtle differences emerge

Wrapper Configuration

- Translate the `XML` to `python`, using a stand alone python script
- Pass arbitrary number, types, and names of parameters to the processor

Marlin/XML

```
<processor name="VXDBarrelDigitiser" type="DDPlanarDigiProcessor">
  <parameter name="SubDetectorName" type="string">Vertex </parameter>
  <parameter name="IsStrip" type="bool">false </parameter>
  <parameter name="ResolutionU" type="float"> 0.003 0.003 0.003 0.003 0.003 0.003 </parameter>
  <parameter name="ResolutionV" type="float"> 0.003 0.003 0.003 0.003 0.003 0.003  </parameter>
  <parameter name="SimTrackHitCollectionName" type="string" lcioInType="SimTrackerHit">VertexBarrelColl
  <parameter name="SimTrkHitRelCollection" type="string" lcioOutType="LCRelation">VXDTrackerHitRelation
  <parameter name="TrackerHitCollectionName" type="string" lcioOutType="TrackerHitPlane">VXDTrackerHits
  <parameter name="Verbosity" type="string">WARNING </parameter>
</processor>
```

Wrapper Configuration

- Translate the XML to python, using a stand alone python script
- Pass arbitrary number, types, and names of parameters to the processor

Gaudi/Python

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

Configuration: Control flow

- XML `execute` section translated to a python list

```xml
<execute>
  <processor name="MyAIDAProcessor"/>
  <processor name="EventNumber" />
  <processor name="InitDD4hep"/>
  <processor name="Config" />
  <!-- ... -->
</execute>
```

```python
algList = []
algList.append(lcioReader)
algList.append(MyAIDAProcessor)
algList.append(EventNumber)
algList.append(InitDD4hep)
algList.append(OverlayFalse)
algList.append(VXDBarrelDigitiser)
#...
```