# Introduction to CEPCSW

Zou Jiaheng

2020.09.17

# Outline

- General introduction
- CEPCSW and Key4HEP
- Examples

# The challenge of HEP software

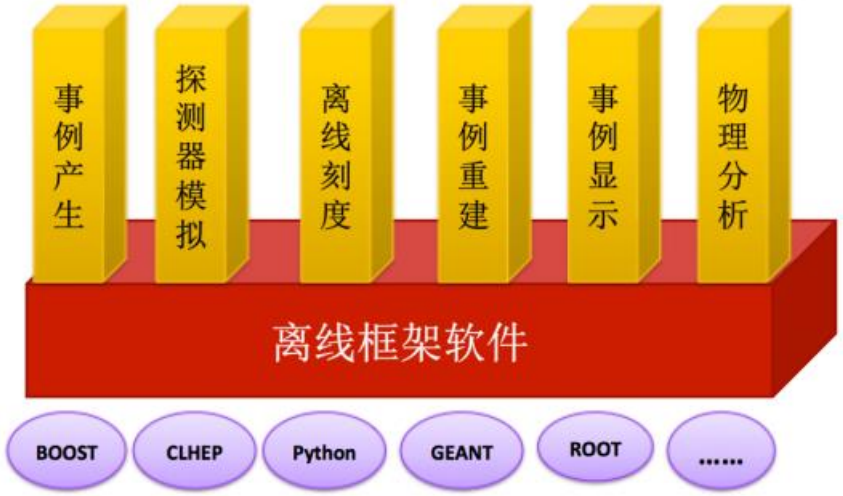
- What we need
  - Various requirements for data processing: simulation, reconstruction, etc
  - Robustness and performance: huge volume of data
  - Long term and continuous developing and maintenance
  - Easy to use: most of our users are not computer experts
- What we have
  - Physicists, but not programmers
- Solution
  - Effective organization
  - Software standardization

# Software Standardization

- A unified data processing software system
  - A unified software developing and running environment
  - The software fundamental architecture are well designed
  - Common reusable functionalities
  - Unified software interfaces
  - Coding conventions
  - Documents
- Improve the software quality
- Improve the communication efficiency between all developers and users

# Software based on a Framework

- Framework is applied by many experiments in HEP field
  - Gaudi: LHCb, BESIII, DYB, …
  - Marlin: iLC, CEPC CDR studies, …
  - SNiPER: JUNO, LHAASO, …
- In a framework
  - Divide and Conquer → Software Modularization
  - Modules are decoupled from each other
  - Modules are assembled and configured at run time




The 3 layer architecture

- Application modules
- Software framework
- Fundamental tools and libraries

# Outline

- General introduction
- CEPCSW and Key4HEP
- Examples

# Key4HEP

An agreement at a workshop in June, 2019

- CEPC
- CLIC
- FCC
- ILC
- SCTF

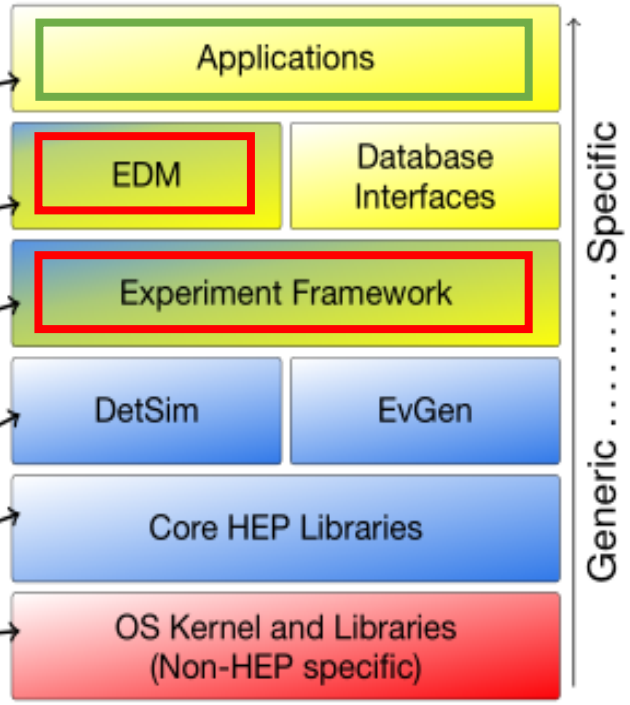
Software components sharing between different experiments

[Ref]: André Sailer, etc. , CHEP2019

https://indico.cern.ch/event/773049/contributions/3474763/attachments/1938664/3213633/191105_sailer_key4hep.pdf

Applications usually rely on large number of libraries, where some depend on others

- Interfaces to tracking and reconstruction libraries (PandoraPFA, ACTS)
- (More or less) experiment specific event data model libraries
- Experiment core orchestration layer, which controls 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 ...

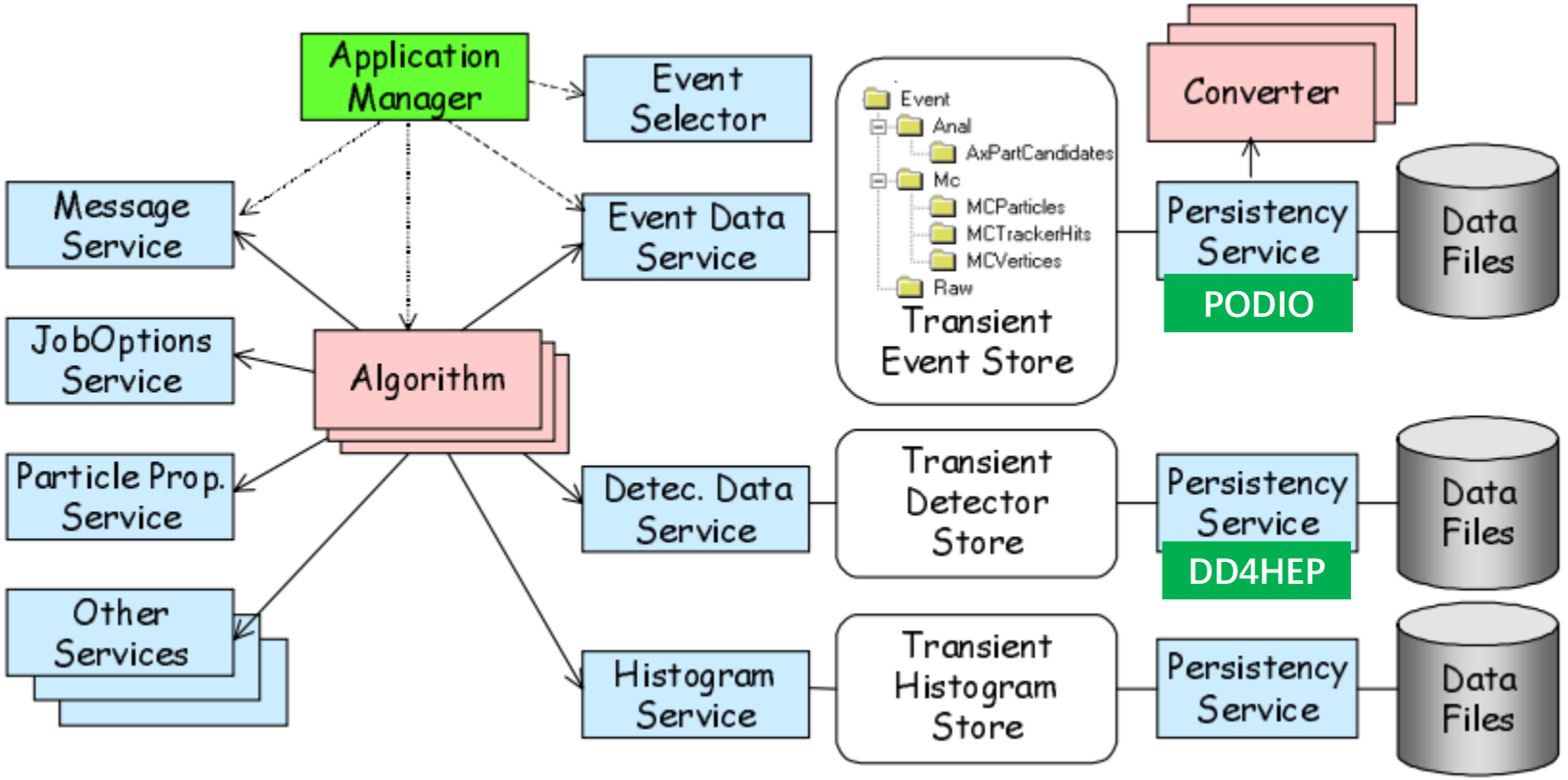


# CEPCSW

- Based on KEY4hep (Common Software Stack for HEP)
- Reuse existing components
  - EDM4hep, DD4hep, Gaudi, ROOT …
- Implement the specific components for CEPC
- Provide a ready-to-work environment to algorithm developers and physicists
  - Porting tracking algorithms from iLCSoft to CEPCSW
  - Integrate more algorithms and features
- Move from iLCSoft to the new software system finally

# Gaudi: the Underlying Framework

- Various algorithms and unified data
- Transient data in memory and persistent data on disk
- Event loop




Use PODIO and DD4HEP in Key4HEP

# Key Components in Gaudi

- Algorithm
  - The concrete calculation to the event during event loop
  - Most frequently used by users
- Service
  - Common functionalities that can be invoked by other components
  - Usually be developed by experienced developers

Framework: stage

Services: lighting, music, scenery, etc.

Algorithms: actors

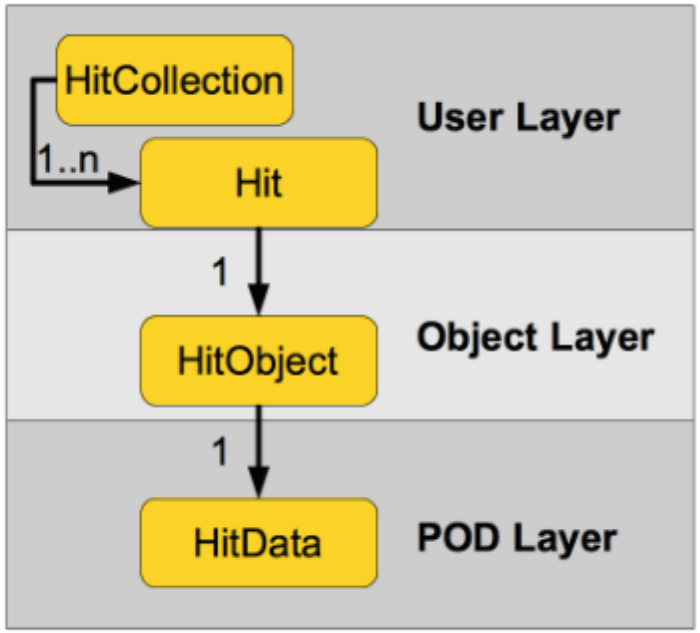
# EDM4hep and PODIO

[Ref]: F. Gaede, etc. , CHEP2019

https://indico.cern.ch/event/773049/contributions/3473254/attachments/1939721/3215730/gaede_podio_chep19.pdf

PODIO is originally developed in context of the FCC study

- user layer (API):
  - handles to EDM objects (e.g. **Hit**)
  - collections of EDM object handles (e.g. **HitCollection**).
- object layer
  - transient objects (e.g. **HitObject**) handling *references* to other objects and *vector members*
- POD layer
  - the actual POD data structures holding the persistent information (e.g. **HitData**)



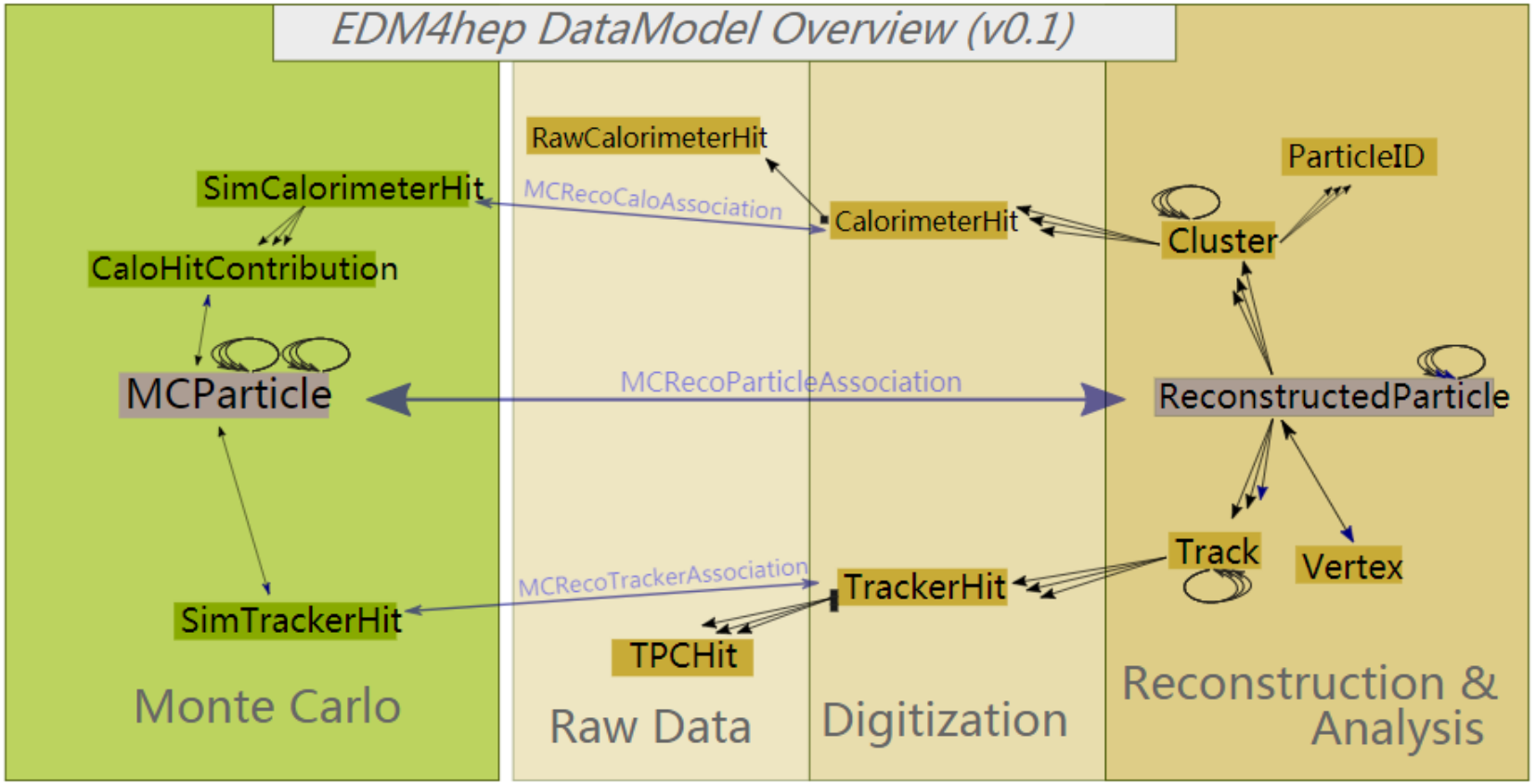

direct access to POD also possible - if needed for performance reason

# EDM4hep and PODIO

EDM4hep is a sub-project of Key4HEP.

EDM4hep V0.1 is released in April, 2020

- A generic event data model for future HEP collider experiments
- The code is generated by PODIO from a yaml file
- An event is described by a set of data collections
- The default storage backend is ROOT



# Outline

- General introduction
- CEPCSW and Key4HEP
- Examples

# Examples in CEPCSW

- CEPCSW/Examples/src: C++ source code
  - HelloWorld: a simple algorithm with a property (runtime configuration)
  - SecondAlg: invoke a service in an algorithm
  - DumpIDAlg: fill tuples/histograms and save to ROOT
  - Edm4hepTest: generate/access EDM4hep data objects in an algorithm
- CEPCSW/Examples/options: Python scripts
  - Configure and start to execute our applications
- Modules after compiling: dynamic-link libraries

```
zoujh@lxslc611 CEPCSW $ ls build.97.0.2.x86_64-slc6-gcc8-opt/lib/
CEPCSW.components    libDCHDigi.so            libDetDriftChamber.components  libDetSimAna.so        libDetSimSD.so     libG
CEPCSW.confdb        libDedxSvc.so            libDetDriftChamber.so          libDetSimCore.so       libDigi_Calo.so    libG
CEPCSW.confdb2       libDetCEPCv4.components  libDetEcalMatrix.components    libDetSimGeom.so       libEventSeeder.so  libG
libDataHelperLib.so  libDetCEPCv4.so          libDetEcalMatrix.so            libDetSimInterface.so  libExamples.so     libG
```

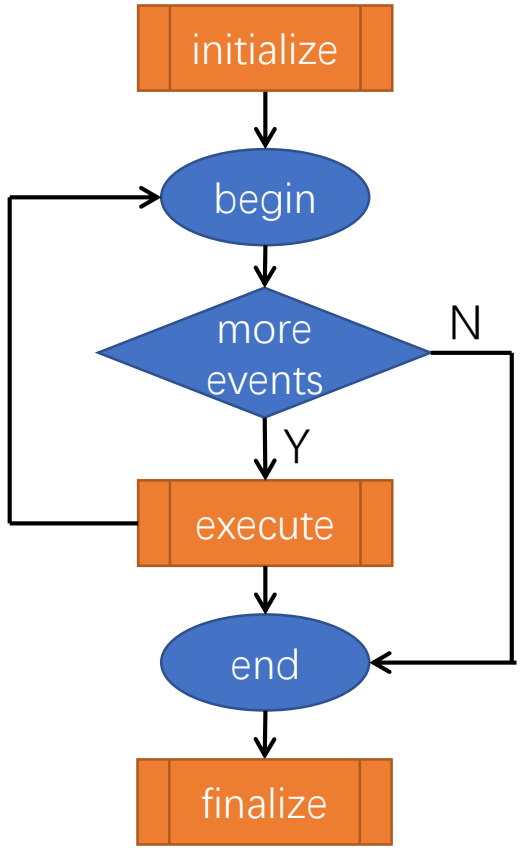
# Example #1: HelloWorld

```
1  #ifndef HelloAlg_h
2  #define HelloAlg_h
3
4  #include <GaudiKernel/Algorithm.h>
5  #include "GaudiKernel/Property.h"
6
7  class HelloAlg: public Algorithm {
8  public:
9      HelloAlg(const std::string& name, ISvcLocator* pSvcLocator);
10
11     StatusCode initialize() override;
12     StatusCode execute() override;
13     StatusCode finalize() override;
14
15 private:
16
17     Gaudi::Property<int> m_int{this, "MyInt", 42};
18 };
19
20
21 #endif
```

- Be inherited from the Algorithm base class
- 3 interfaces to implement
  - initialize()
  - execute()
  - finalize()
- Configuration with Gaudi::Property

# HelloWorld Implementation and Execution

```
10  StatusCode
11  HelloAlg::initialize() {
12      StatusCode sc;
13
14      info() << "MyInt: " << m_int.value() << endmsg;
15
16      return sc;
17  }
18
19  StatusCode
20  HelloAlg::execute() {
21      StatusCode sc;
22      return sc;
23  }
24
25  StatusCode
26  HelloAlg::finalize() {
27      StatusCode sc;
28      return sc;
29  }
```




Be invoked one time at the beginning

Be invoked one time for each event (event loop)

Be invoked one time at the end

# HelloWorld Configuration and Running

**helloalg.py**

```
1  #!/usr/bin/env python
2
3  from Gaudi.Configuration import *
4
5  from Configurables import HelloAlg
6
7  helloalg = HelloAlg("helloAlg")
8  helloalg.MyInt = 42
9
10 # ApplicationMgr
11 from Configurables import ApplicationMgr
12 ApplicationMgr( TopAlg = [helloalg],
13                 EvtSel = 'NONE',
14                 EvtMax = 10,
15 )
```

**A Gaudi::Property in C++**

```
Gaudi::Property<int> m_int{this, "MyInt", 42};
```

A string name in C++

An attribute in Python

```
zoujh@lxslc611 CEPCSW $ ./run.sh Examples/options/helloalg.py
# setting LC_ALL to "C"
# --> Including file '/workfs/bes/zoujh/key4hep/CEPCSW/Example
# <-- End of file '/workfs/bes/zoujh/key4hep/CEPCSW/Examples/o
ApplicationMgr    SUCCESS
====================================================================
                                                   Welcome to
                                                running on lxslc711.
====================================================================
ApplicationMgr       INFO Application Manager Configured succe
helloAlg             INFO MyInt: 42
EventLoopMgr      WARNING Unable to locate service "EventSelec
EventLoopMgr      WARNING No events will be processed from ext
```

# Example #2: SecondAlg

```
10  StatusCode
11  SecondAlg::initialize() {
12      StatusCode sc;
13
14      info() << "Retrieving the FirstSvc... " << endmsg;
15
16      m_firstsvc = service("FirstSvc");
17
18      return sc;
19  }
20
21  StatusCode
22  SecondAlg::execute() {
23      StatusCode sc;
24
25      m_firstsvc->shoot();
26
27      return sc;
28  }
```

Get the service instance

Invoke the service

```
zoujh@lxslc611 CEPCSW $ ./run.sh Examples/options/secondalg.py
# setting LC_ALL to "C"
# --> Including file '/workfs/bes/zoujh/key4hep/CEPCSW/Examples
# <-- End of file '/workfs/bes/zoujh/key4hep/CEPCSW/Examples/op
ApplicationMgr    SUCCESS
=============================================================
                                                  Welcome to C
                                               running on lxslc711.i
=============================================================
ApplicationMgr       INFO Application Manager Configured succes
secondAlg            INFO Retrieving the FirstSvc...
EventLoopMgr      WARNING Unable to locate service "EventSelect
EventLoopMgr      WARNING No events will be processed from exte
ApplicationMgr       INFO Application Manager Initialized succe
ApplicationMgr       INFO Application Manager Started successfu
FirstSvc             INFO shoot it.
FirstSvc             INFO shoot it.
FirstSvc             INFO shoot it.
FirstSvc             INFO shoot it.
FirstSvc             INFO shoot it.
FirstSvc             INFO shoot it.
FirstSvc             INFO shoot it.
FirstSvc             INFO shoot it.
FirstSvc             INFO shoot it.
FirstSvc             INFO shoot it.
ApplicationMgr       INFO Application Manager Stopped successfu
EventLoopMgr         INFO Histograms converted successfully acc
```

# Example #3: DumpIDAlg

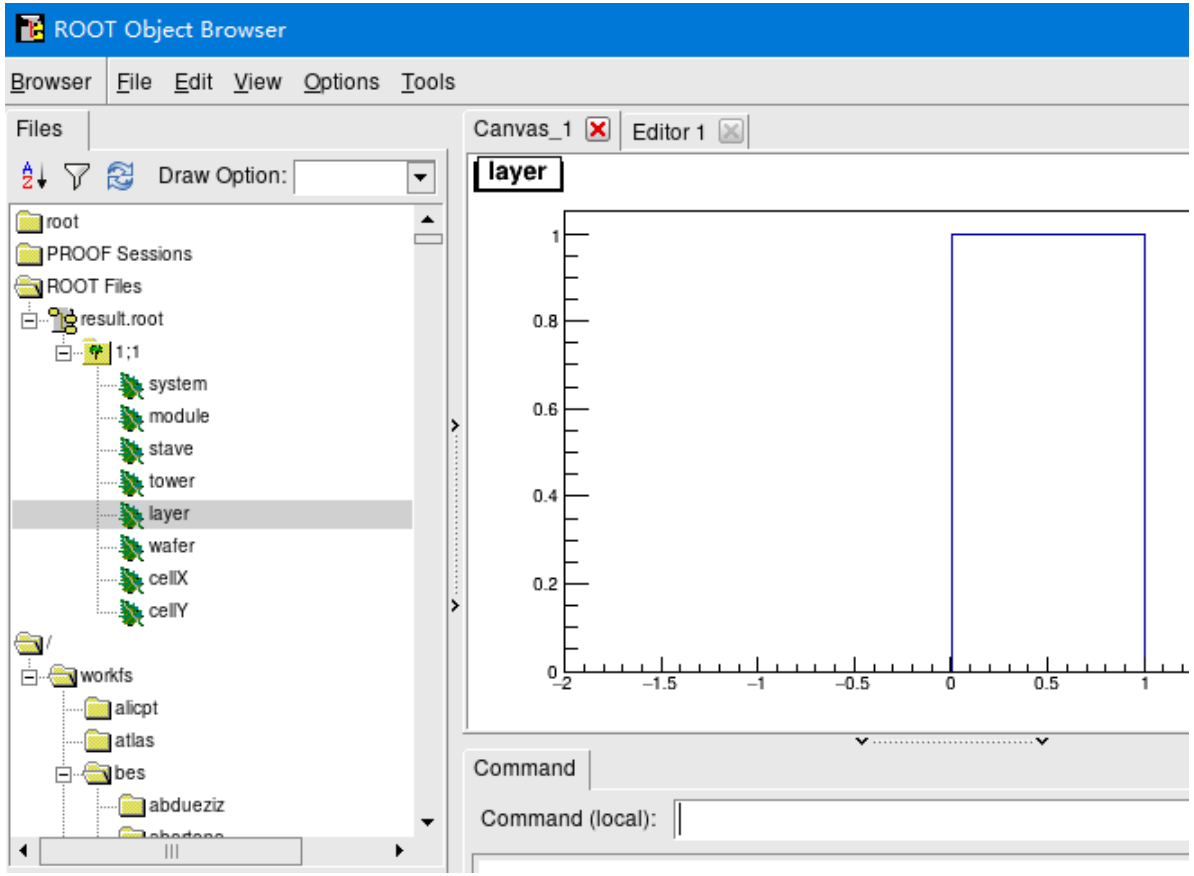
**NTuple declaration in the header file**

```
38      // strore all the id for later analysis
39      NTuple::Tuple* m_tuple_id = nullptr;
40
41      NTuple::Item<int> m_id_system;
42      NTuple::Item<int> m_id_module;
```

**Book the NTuple in initialize()**

**Assign values to NTuple items and save it in execute()**

**NTuple analysis in ROOT**



# Example #4: Edm4hepTest

EDM4hep data objects (or collections) are managed by DataHandle

- A template class to handle any EDM4hep data collections in memory
- A string name: it's also the branch name in ROOT files
- Reader/Writer flag

Two algorithms

- Edm4hepWriteAlg: generate new EDM4hep data objects and register them to the framework
- Edm4hepReadAlg: read EDM4hep data objects in memory

Algorithms are decoupled from memory management and data I/O

# Edm4hepWriteAlg

- DataHandle declarations in header file

```
DataHandle<edm4hep::EventHeaderCollection> m_headerCol{"EventHeader", Gaudi::DataHandle::Writer, this};
DataHandle<edm4hep::MCParticleCollection> m_mcParCol{"MCParticle", Gaudi::DataHandle::Writer, this};
DataHandle<edm4hep::SimCalorimeterHitCollection> m_simCaloHitCol{"SimCalorimeterHit", Gaudi::DataHandle::Writer, this};
DataHandle<edm4hep::CaloHitContributionCollection> m_caloHitContCol{"CaloHitContribution", Gaudi::DataHandle::Writer, this};
```

- Create a SimCalorimeterHitCollection

```
auto simCaloCol = m_simCaloHitCol.createAndPut();
```

- Create a SimCalorimeterHit object

```
auto hit = simCaloCol->create();
```

- Data collection is managed by Gaudi. Data object is managed by collection.

# Edm4hepReadAlg

- DataHandle declarations in header file

```
27          DataHandle<edm4hep::EventHeaderCollection> m_headerCol{"EventHeader", Gaudi::DataHandle::Reader, this};
28          DataHandle<edm4hep::MCParticleCollection> m_mcParCol{"MCParticle", Gaudi::DataHandle::Reader, this};
29          DataHandle<edm4hep::SimCalorimeterHitCollection> m_calorimeterCol{"SimCalorimeterCol",
30                  Gaudi::DataHandle::Reader, this};
31          DataHandle<edm4hep::CaloHitContributionCollection> m_caloContribCol{"SimCaloContributionCol",
32                  Gaudi::DataHandle::Reader, this};
```

- Configure the collection names via Gaudi property

  In the constructor of Edm4hepReadAlg

```
declareProperty("HeaderCol", m_headerCol);
declareProperty("MCParticleCol", m_mcParCol, "MCParticle collection
declareProperty("SimCalorimeterHitCol", m_calorimeterCol, "MCParticl
```

# Data Input Configuration

**In Examples/options/edm4hep_read.py**

```
5   from Configurables import K4DataSvc
6   dsvc = K4DataSvc("EventDataSvc", input="test.root")
7
8   from Configurables import PodioInput
9   podioinput = PodioInput("PodioReader", collections=[
10      "EventHeader",
11      "MCParticle",
12      "SimCalorimeterHit"
13      ])
14
15  from Configurables import Edm4hepReadAlg
16  alg = Edm4hepReadAlg("Edm4hepReadAlg")
17  #alg.HeaderCol.Path = "EventHeader"
18  #alg.MCParticleCol.Path = "MCParticle"
19  alg.SimCalorimeterHitCol.Path = "SimCalorimeterHit"
```

Set input file(s)

Only read the data collections that we need
This is important to improve the software execution efficiency

Configure the data collection names according to the concrete input file(s)

# To Developers and also Physicists

- Divide and Conquer → Software Modularization
  - Most modules are decoupled & most developers can work independently
- Cooperation
  - Modules are assembled by framework to form a powerful data processing system
- Software Reuse
  - Fundamental tools and libraries
  - Gaudi framework: shared in HEP field
  - Key4HEP: unified Event Data Model (EDM4hep) and generic algorithms for future collider experiments

# Thanks!