Developing Software in CEPCSW

Weidong Li

representing the CEPC software team

CEPC Workshop on New Physics and Flavor Physics

August 13-18, 2023, Shanghai

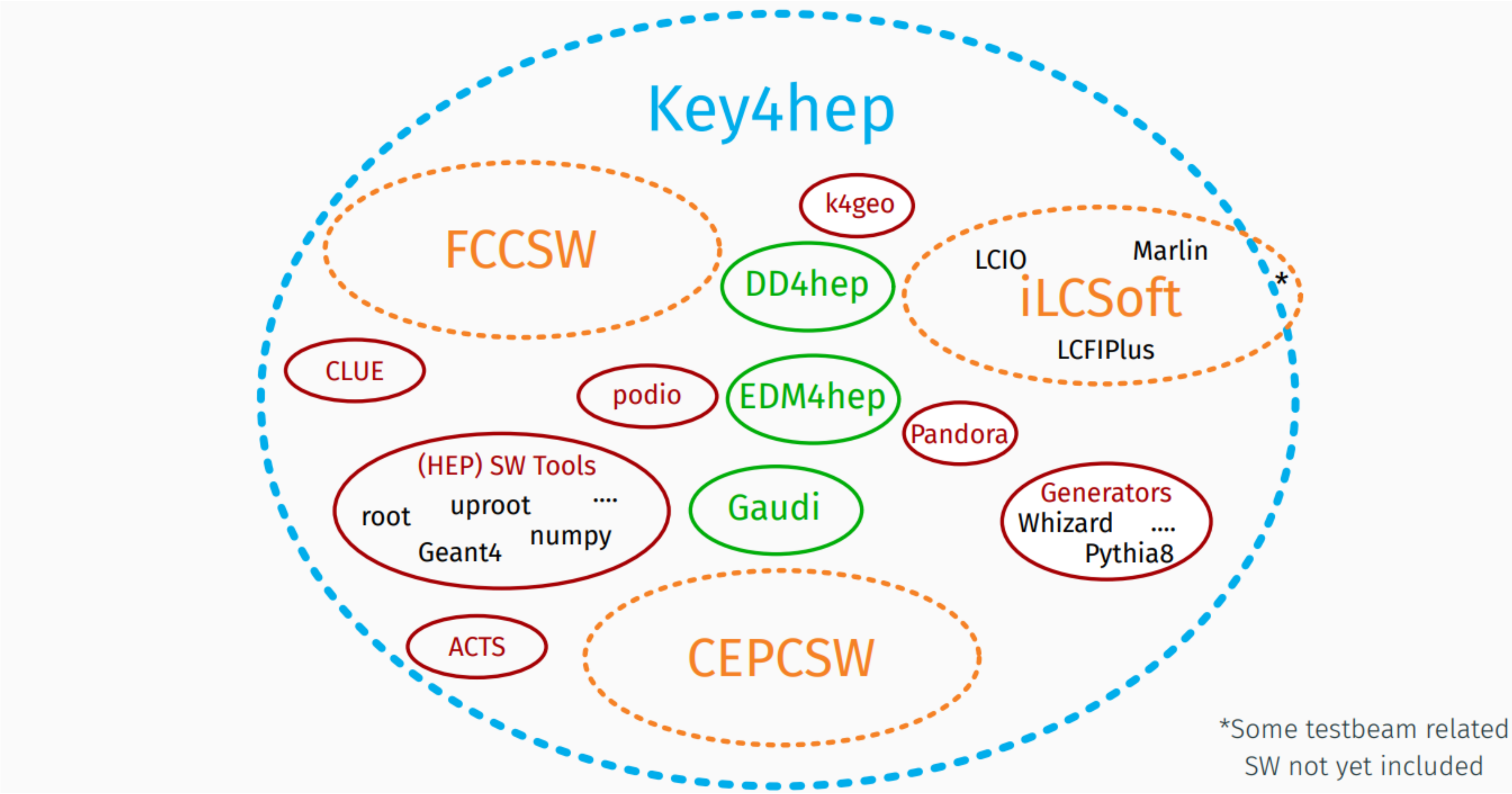
Contents

- Introduction to CEPCSW
- Software development
  - Drift chamber software
  - Analysis algorithm
- Latest progress
- Summary

History of CEPC software

- The development of CEPC software first started with the iLCSoft
  - Reused software iLCSoft modules: Marlin, LCIO, MokkaC, Gear
  - Developed CEPC components for simulation and reconstruction
  - Produced M.C. data for detector design and physics potential studies
  - CDR was released in Nov, 2018, based on results from the iLCSoft
- New CEPC software (CEPCSW) prototype was proposed at the Oxford workshop in April 2019
- The consensus among CEPC, CLIC, FCC, ILC and other future experiments was reached at the Bologna workshop in June, 2019
  - Develop a Common Turnkey Software Stack (Key4hep) for future collider experiments
  - Maximize the sharing of software components among different experiments

Common Software Stack: Key4hep



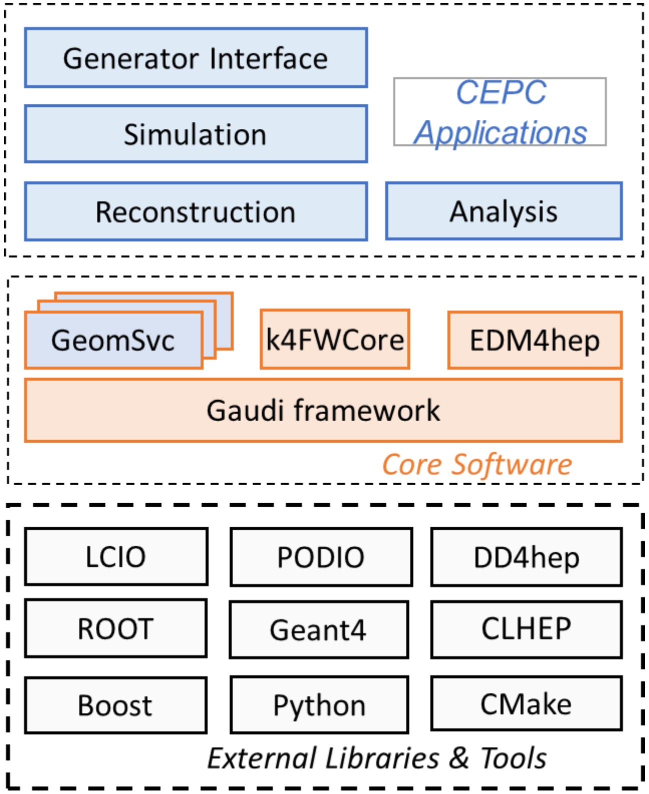
- Non-goal
  - Develop and maintain project specific software and workflows

    (T.Madlener | Key4hep & EDM4hep, CEPC workshop, Edinburgh)

CEPCSW Core software

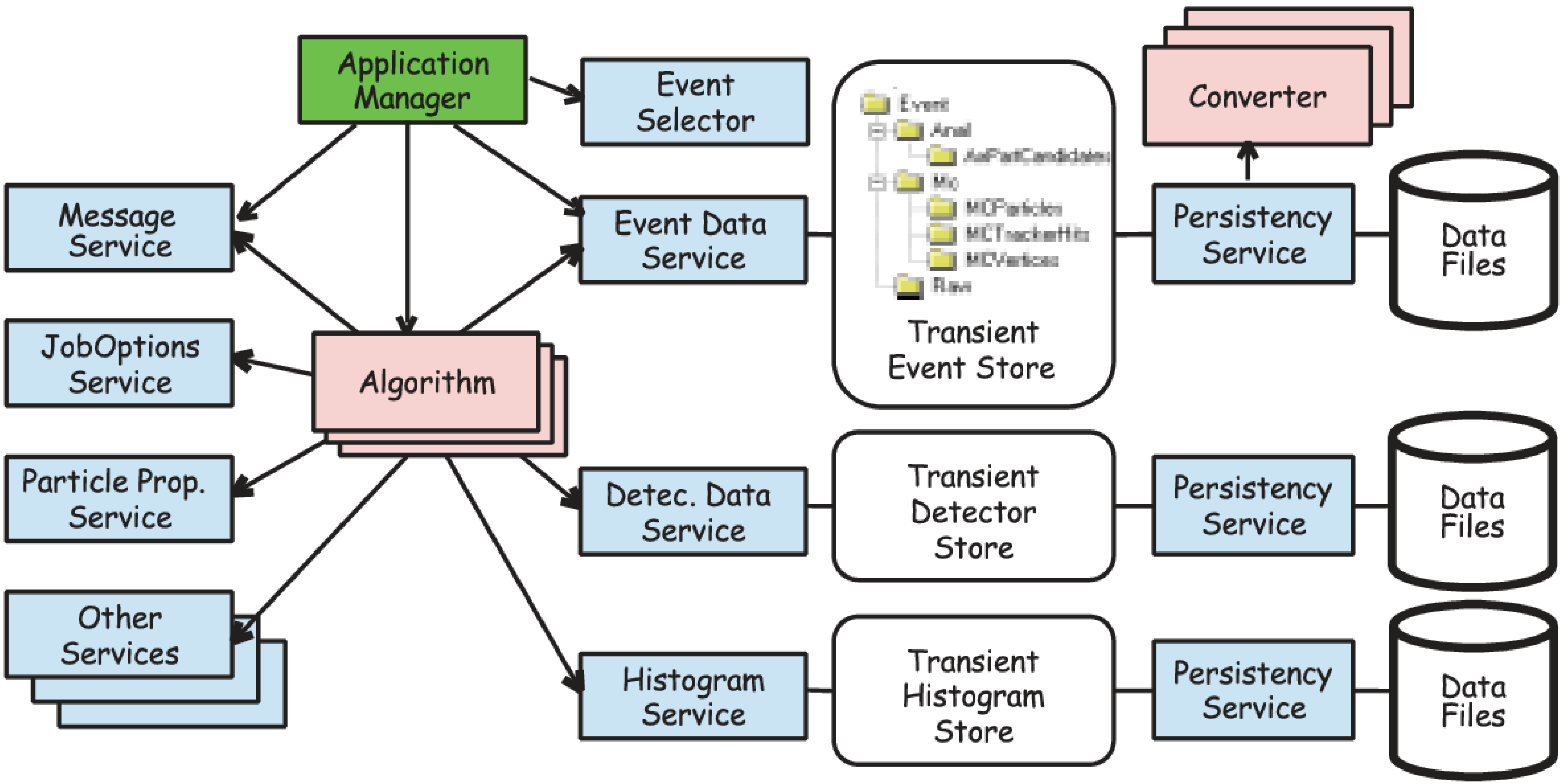
- CEPCSW software structure
  - Applications: simulation, reconstruction and analysis
  - Core software
  - External libraries

https://github.com/cepc/CEPCSW



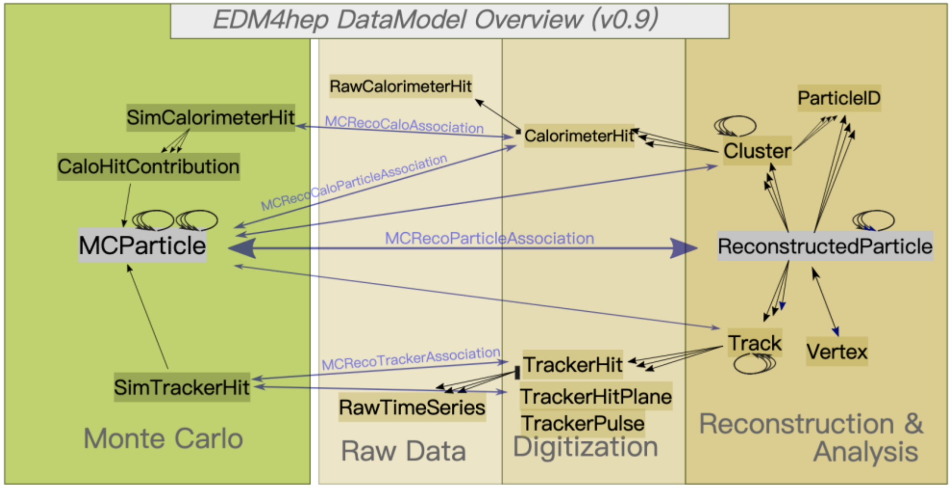
- Core software
  - Gaudi/Gaudi Hive: defines interfaces to all software components and controls their execution
  - EDM4hep: generic event data model
  - k4FWCore: manages the event data
  - DD4hep: geometry description
  - CEPC-specific components : GeomSvc, detector simulation, beam background mixing, fast simulation, machine learning interface, etc.

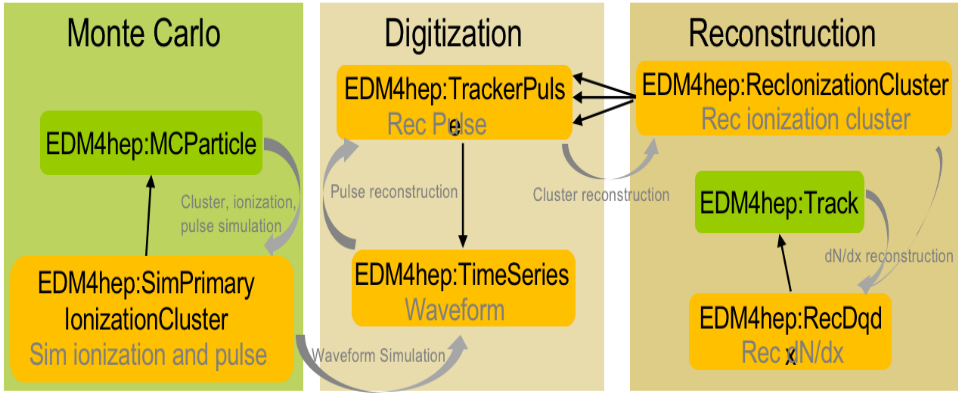
Gaudi framework



Event Data Model

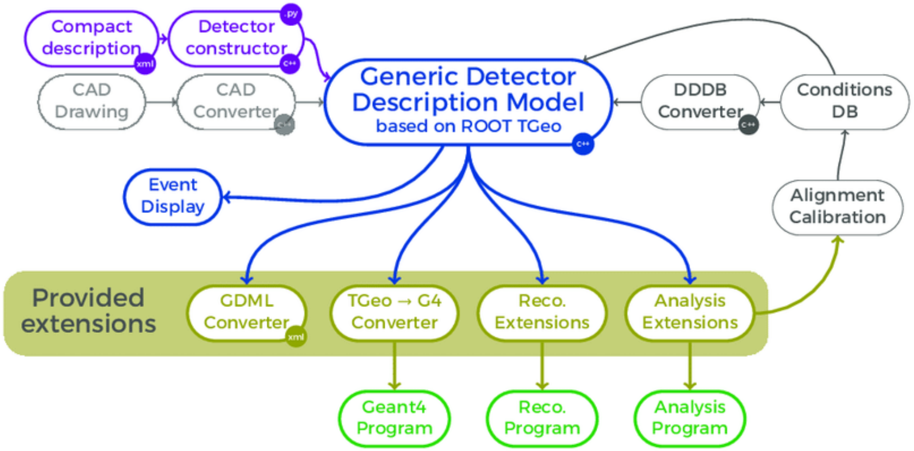
- EDM4hep is the common event data model (EDM) being developed for the future experiments like CEPC, CLIC, FCC, ILC, etc.
  - describing event objects created at different data processing stages and also reflecting the relationship between them.
- Due to the strong flexibility of EDM4hep, TPCHit was extended to accommodate the new needs:
  - By using the upstream mechanism of PODIO, a common EDM was implemented for both TPC and drift chamber

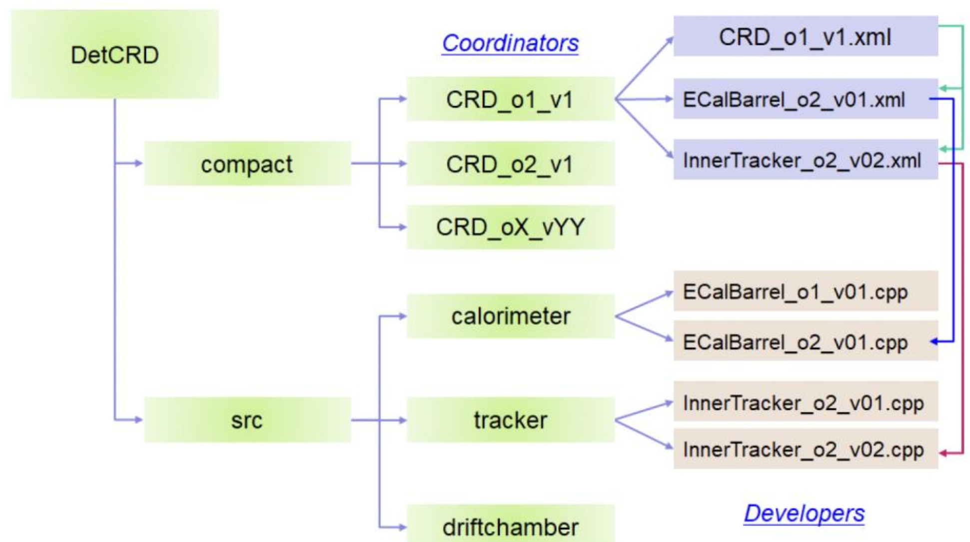




Detector Description

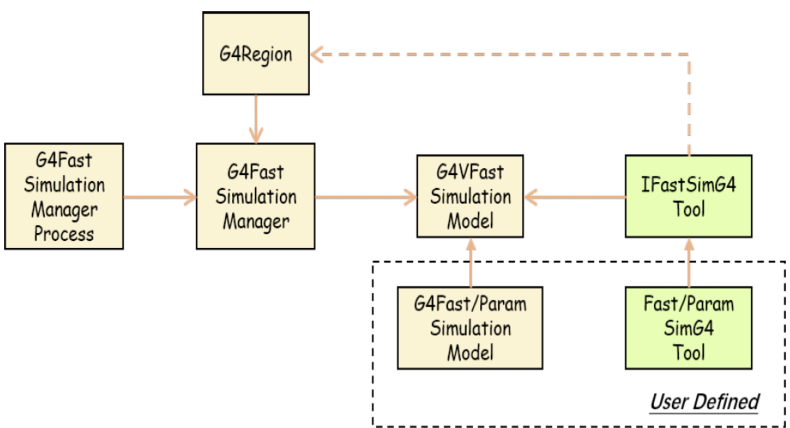
- DD4hep was adopted to provide a full detector description, which was generated from a single source
- Different detector design options are managed in the Git repository and a simulation job can be easily configured in runtime
- The non-uniform magnetic field was also implemented in CEPCSW

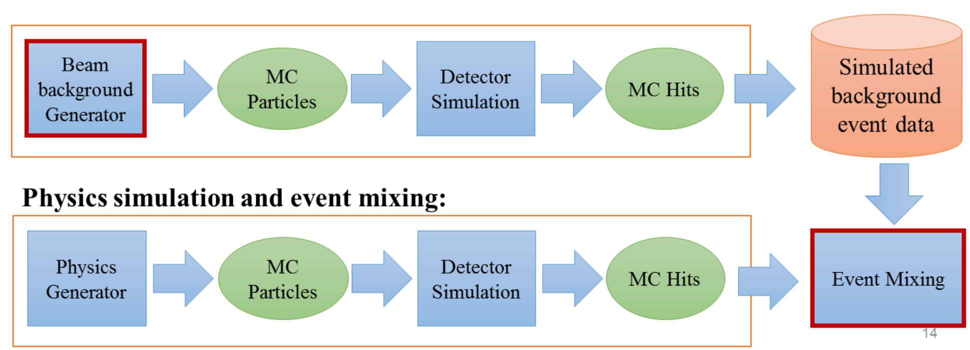




Simulation framework

- The simulation framework was developed and the simulation chain is complete for sub-detectors such as:
  - silicon detector, time projection chamber, drift chamber and calorimeters
- The region-based fast simulation interface was also developed to integrate different of fast simulation modules into the detector simulation
- An event mixing tool was also provided to mix different types of backgrounds with physics signals at hit level.



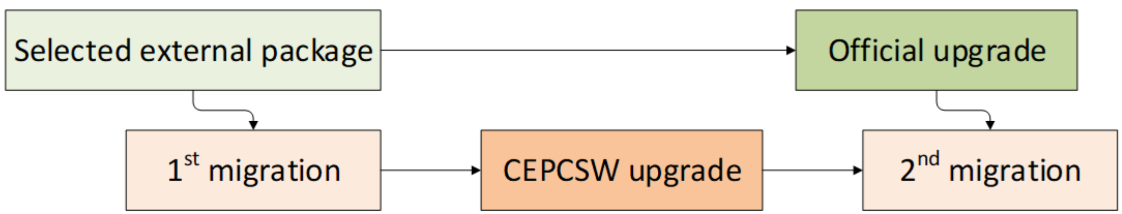


Migration of reconstruction algorithm

- Migration of tracking software from Marlin to CEPCSW

Migration/Implementation

- Migrated full tracking chain from Marlin
  - Switch class to GaudiAlgorithm
  - Switch data model to EDM4hep (first realized)



- Another implement way is in considering, to test coversion cost
  - Create a GaudiAlgorithm to covert data model and call
    - ✓ EDM4hep→LCIO→call event loop function→LCIO→EDM4hep
- Key4hep (best) in plan
  - CallingAlg to call prepared API
  - event model support with same code is important

Packages in CEPCSW

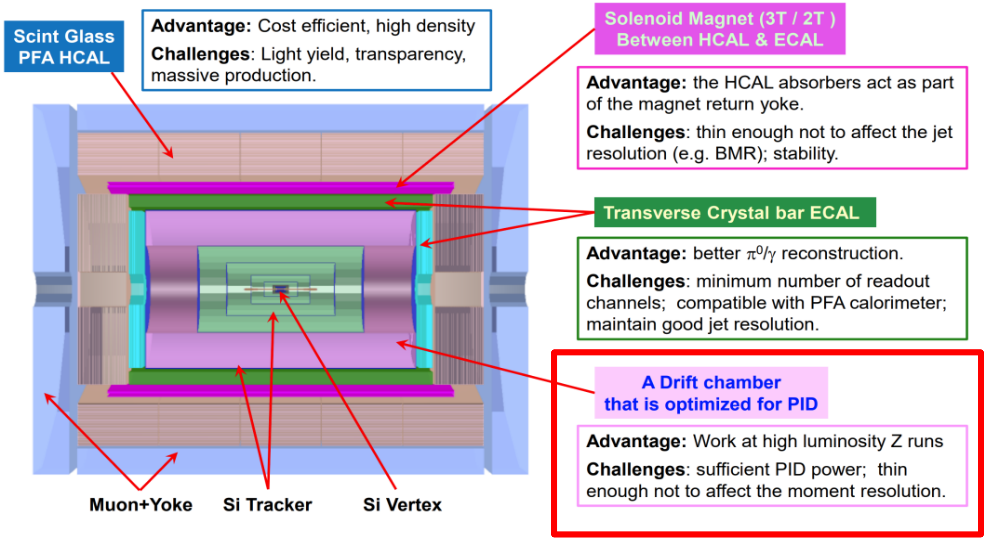
- Detector concepts
  - CDR (baseline design)
  - The 4th concept
- MC Generators
  - Multiple formats supported: HepMC, HepEvt, StdHep, LCIO
  - GuineaPig++ for MDI
  - Particle Gun
- Simulation
  - G4 simulation framework
  - Fast simulation algoritrhms e.g.ML-based dE/dx simulation
  - Digitization algorithms for silicon, CALO, drift chamber
- Reconstruction
  - Marlin based tracking algorithms for silicon detector
  - Tracking algorithm for drift chamber
  - Pandora-based PFA
  - Arbor-based PFA
- Analysis tools
  - RDataFrame-based analysis framework
- Examples and docs
  - Usage of EDM4hep, Identifier, etc.

50 packages in total

- Introduction to CEPCSW
- Software development
  - Drift chamber software
  - Analysis algorithm
- Latest progress
- Summary

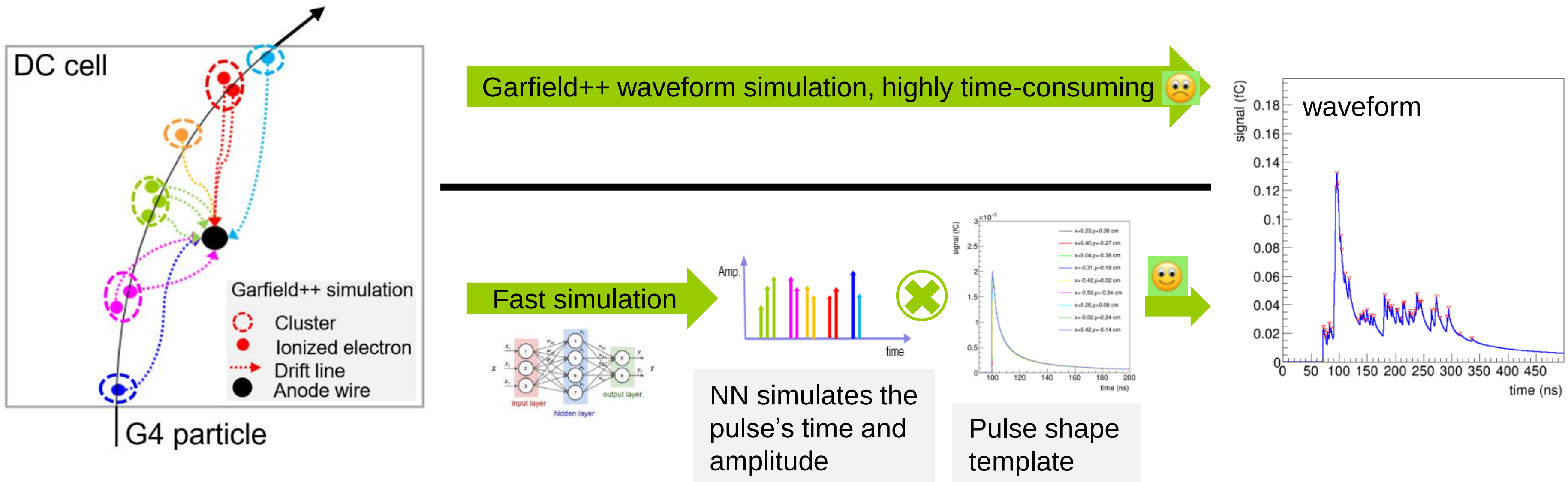
Drift chamber

- The CEPC experiment mainly aims to precisely measure the property of the Higgs boson.
- Physics requirements: high track efficiency (~100%), momentum resolution (<0.1%), PID (2σ p/K separation at P < ~ 20 GeV/c), etc.



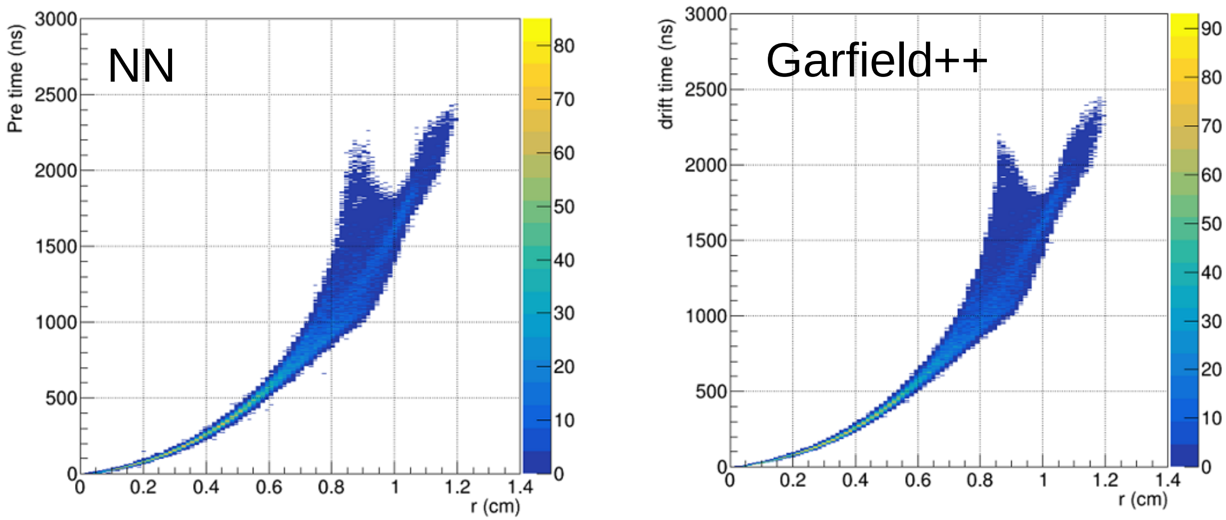
- For the 4th conceptual detector, silicon detector and drift chamber (DC) are designed to provide both tracking and PID for charged particles.
- Both detector design and physics potential studies needs strong support of simulation and reconstruction software.

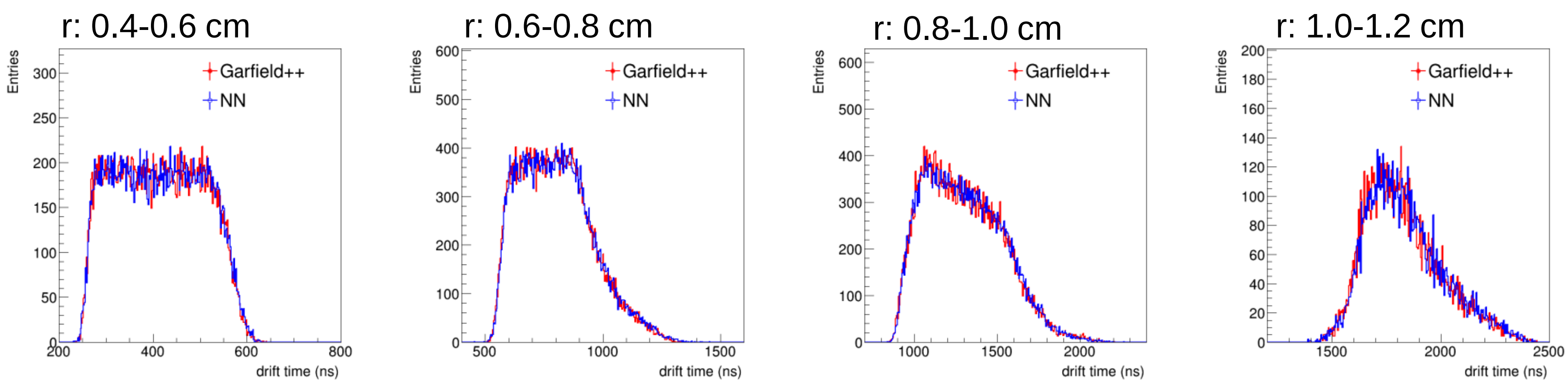
Simulation of drift chamber



- TrackHeedSimTool (Gaudi tool) was implemented by combining Geant4 and Garfield++ to simulate the complete response of the gaseous detector
  - Input: G4Step information (particle type, initial position, momenta, and step length)
  - Using TrackHeed(from Garfield++) to create the ionization electron-ion pairs (for both primary and secondary ionizations), the deposited energy will be used to update the energy of the G4Particle
  - Using NN to simulate the time and amplitude of each pulse for each ionized electron (for fast waveform simulation)
  - Output: primary, total ionization, and pulse information, saved in EDM

ML-based simulation

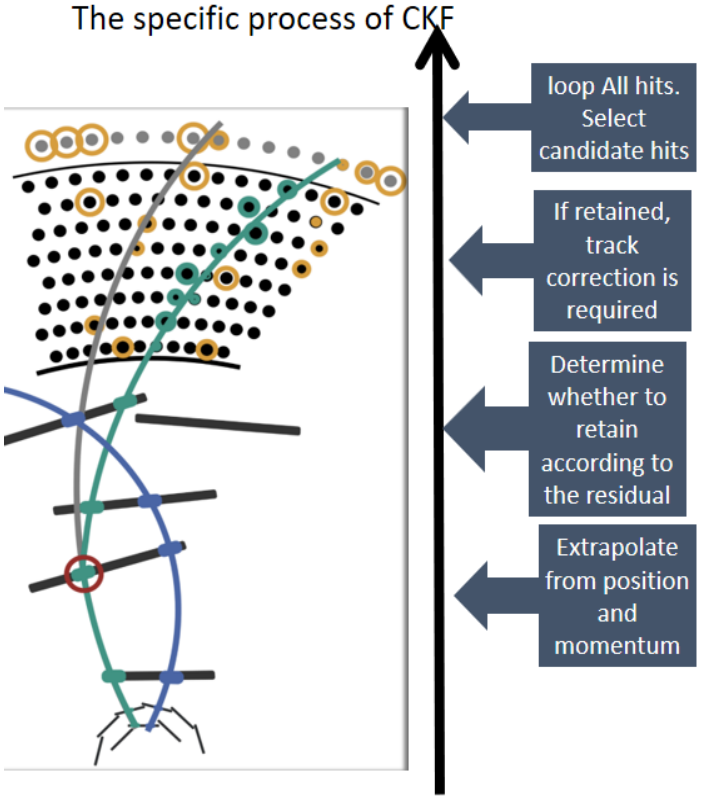




❖ Good agreement between the NN and Garfield++ simulation

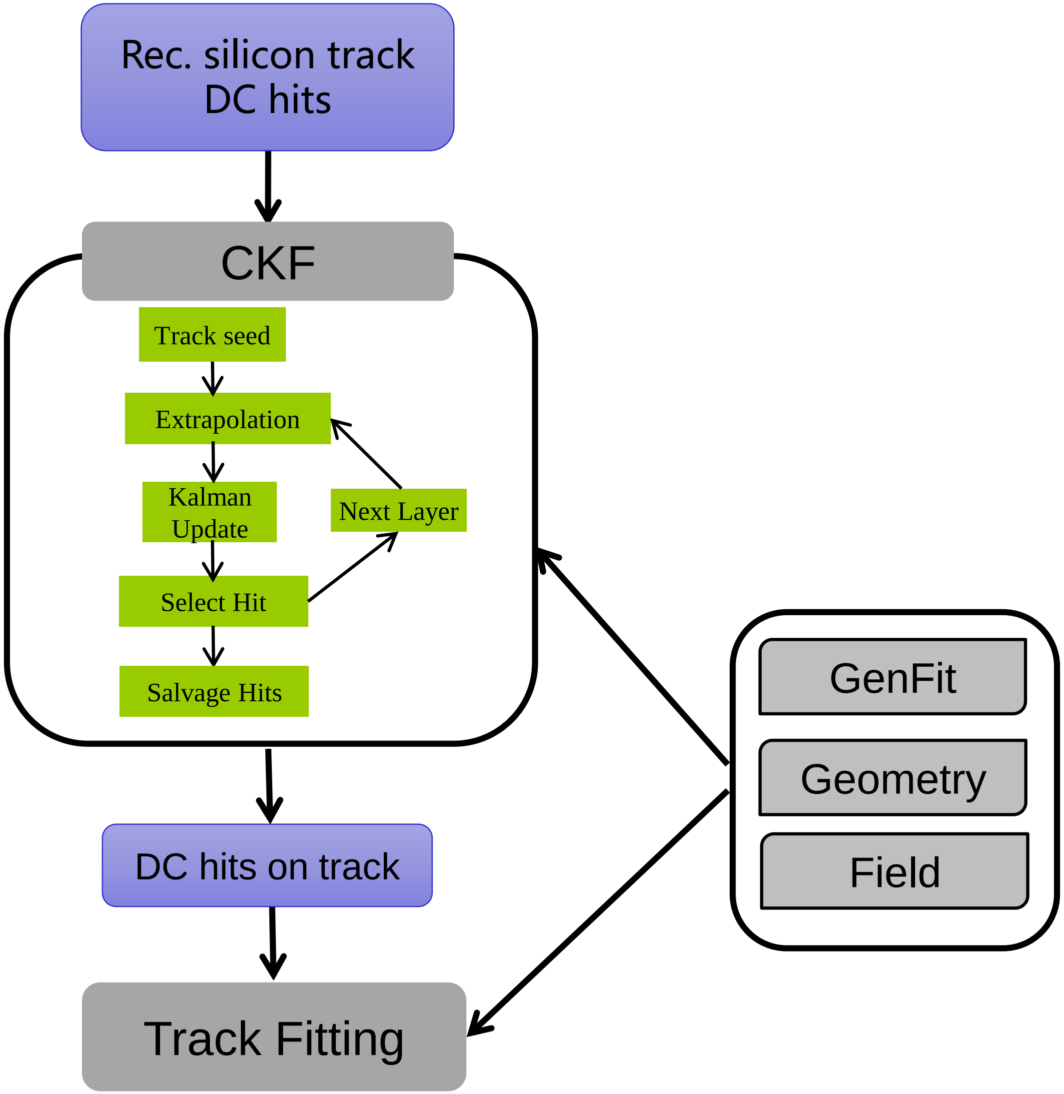
Track reconstruction (1)

- Tracking with Combinatorial Kalman Filter (CKF) method
  - Used by many high energy physics experiments
- Track finding with CKF in drift chamber
  - Migrate from Belle2
  - Track segments reconstructed in the silicon detector, called seeds, are extrapolated to the DC and all the DC hits belonging to the track are collected
- Track fitting tool: Genfit
  https://github.com/GenFit/GenFit/
  - Experiment-independent generic track fitting toolkit
  - Official track fitting for BelleII, also used by PANDA, COMET, GEM-TPC etc.
  - Using DAF kalman filter



Track reconstruction (2)

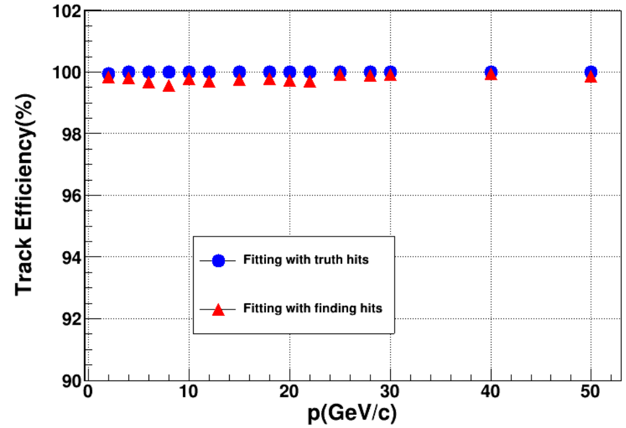
- Implementation
  - Extrapolation based on GenFit
  - Field, material and geometry from DD4hep
  - A data format converter between TrackerHit(from EDM4hep) and CDCWireHit
- Status
  - Performance validation completed



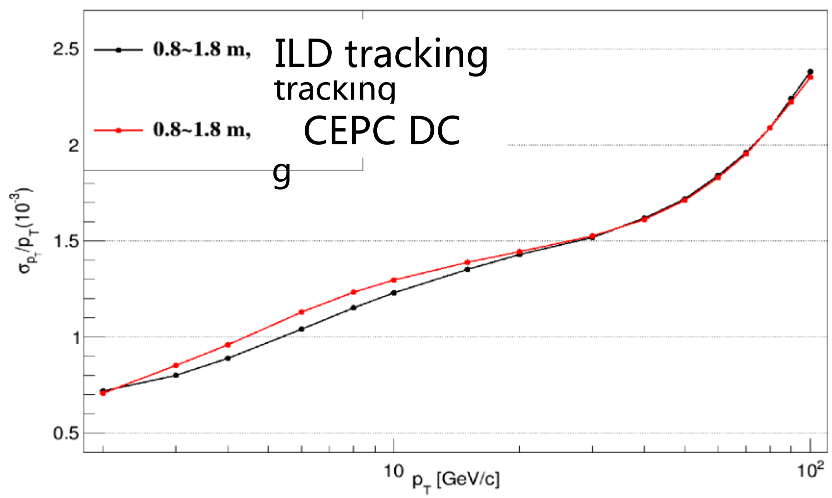
Workflow of track reconstruction

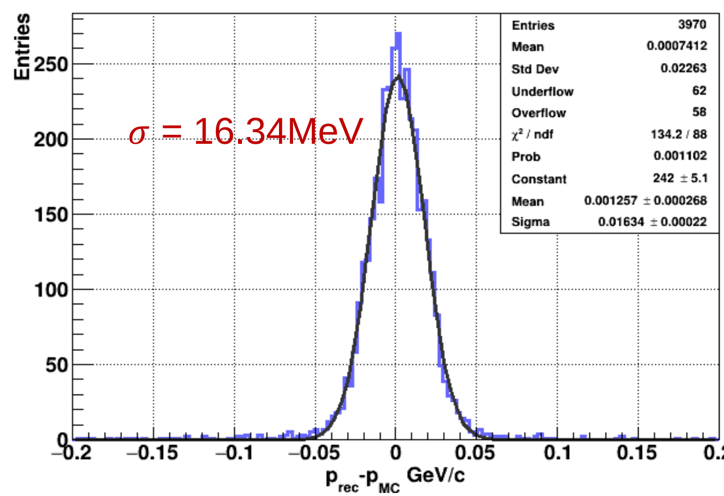
Tracking performance

- Data sample: Single particle $\mu^-, \theta = 50°$
- Track Efficiency = N_1/N_2
  - N_1 is the number of track satisfying:
    - $chi^2 < 400$
    - $N_{DC\ hits\ on\ track} > 50$
  - N_2 is the numbre of silicon track



- Combined measurements of Silicon and Drift Chamber





Momentum distribution of 10GeV/c μ^-

Analysis algorithm: base class

- Event loop in Gaudi
- Derived from Algorithm

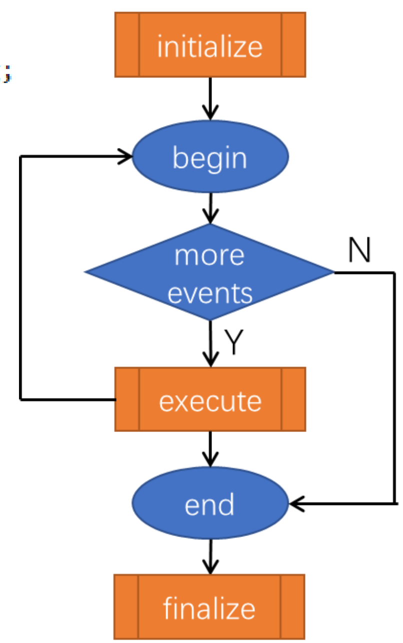
```
StatusCode
HelloAlg::initialize() {
    StatusCode sc;

    info() << "MyInt: " << m_int.value() << endmsg;

    return sc;
}

StatusCode
HelloAlg::execute() {
    StatusCode sc;
    return sc;
}

StatusCode
HelloAlg::finalize() {
    StatusCode sc;
    return sc;
}
```



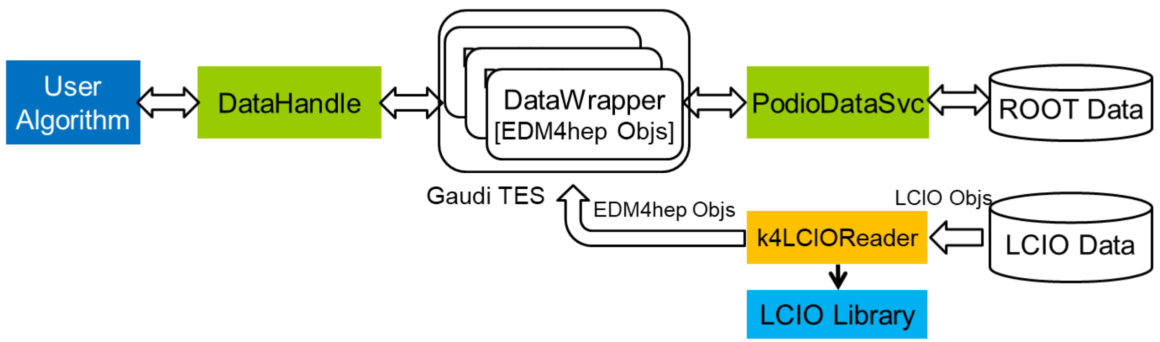
Be invoked one time at the beginning

Be invoked one time for each event (event loop)

Be invoked one time at the end

Analysis algorithm: reading data

- Both LCIO and EDM4hep data are supported
  - k4LCIOReader is used to read an event from LCIO data files and create EDM4hep data objects on the fly
  - Switch different data format in the python configuration scripts



- DataHandle is used to read data collections in the C++ Algorithm

type

```
        DataHandle<edm4hep::SimCalorimeterHitCollection> m_EcalBarrelCol{"EcalBarrelCollection",
            Gaudi::DataHandle::Reader, this};
```

name

Analysis algorithm: writing data

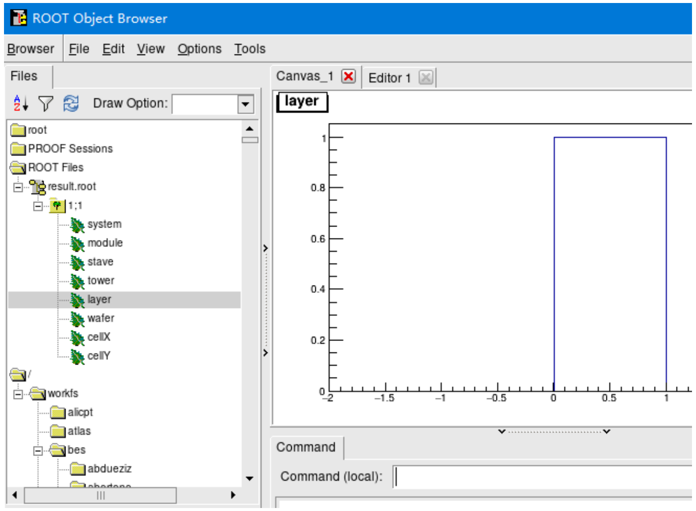
- DataHandle is also used to save a collection to an EDM4hep file

```
DataHandle<edm4hep::SimCalorimeterHitCollection> m_simCaloHitCol{"SimCalorimeterHit", Gaudi::DataHandle::Writer, this};
```

- User defined NTuples
  - NTuple declaration in the header file

```
    // strore all the id for later analysis
    NTuple::Tuple* m_tuple_id = nullptr;

    NTuple::Item<int> m_id_system;
    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
  - In CEPCSW
    - Examples/src/DumpIDAlg

Analysis algorithm: job configuration

- A job is fully configurable with Python script
  - algorithms, services, runtime parameters, I/O files ……
- Gaudi Property: initialize a C++ variable in the Python script
  - A Gaudi::Property (with a string name) in C++

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

  - An attribute in Python

```
helloalg = HelloAlg("helloAlg")
helloalg.MyInt = 42
```

- Many examples can be found in "CEPCSW/Examples"

There isn't an official naming policy to the data collections yet.

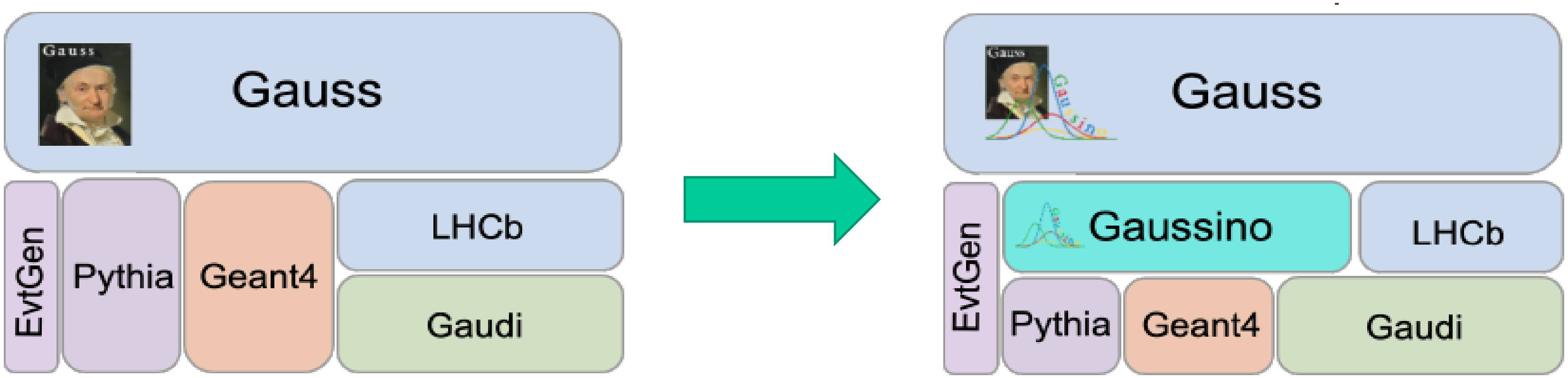
The data collection names should be treated carefully in both C++ and Python

```
from Configurables import PodioInput
podioinput = PodioInput("PodioReader", collections=[
    "EventHeader",
    "MCParticle",
    "SimCalorimeterHit"
    ])
```

- Introduction to CEPCSW
- Software development
  - Drift chamber software
  - Analysis algorithm
- Latest progress
- Summary

Gaussino-based simulation (1)

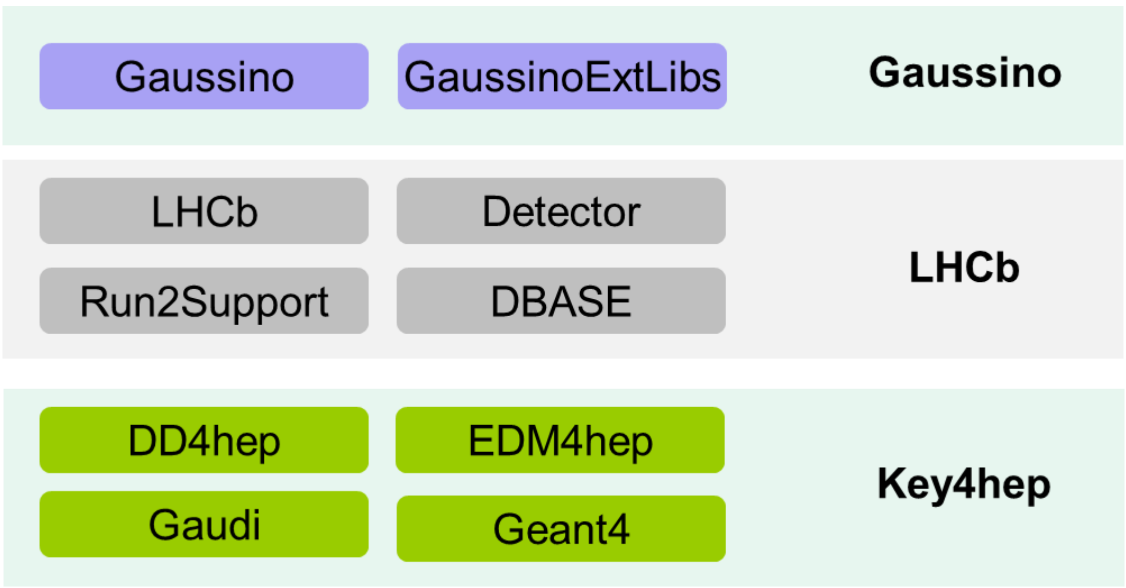
- CEPC also works together with Key4hep project members and is re-implementing CEPC detector simulation with Gaussino
- Evolution of the simulation framework from LHCb
  - Better support for multi-threading, machine learning, fast simulation methods
  - Gauss-on-Gaussino is a new version of LHCb simulation framework



- Gaussino is being added to Key4hep by extracting experiment-independent parts from Gauss

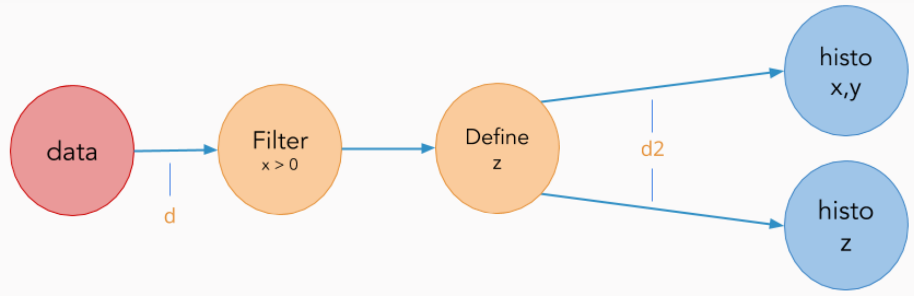
Gaussino-based simulation (2)

- Now Gaussino still depends on LHCb software and can not be used by other experiments directly
- Development of CEPC-on-Gaussino was planned with the following three steps
  - Using the original version having the dependency on the LHCb software
  - Creating the modified version in which the LHCb dependency is removed
  - Directly using the Key4hep version (not available at the moment)



Analysis toolkit based on RDataFrame (1)

- RDataFrame is a powerful tool for data analysis
  - Program language: Python and C++
  - Declarative programming and parallel processing are supported
  - EDM4hep data can be read directly
  - Being used by many experiments such as FCC-ee

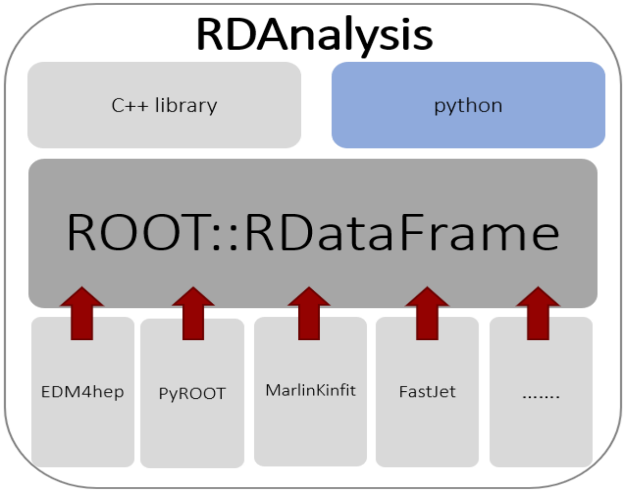


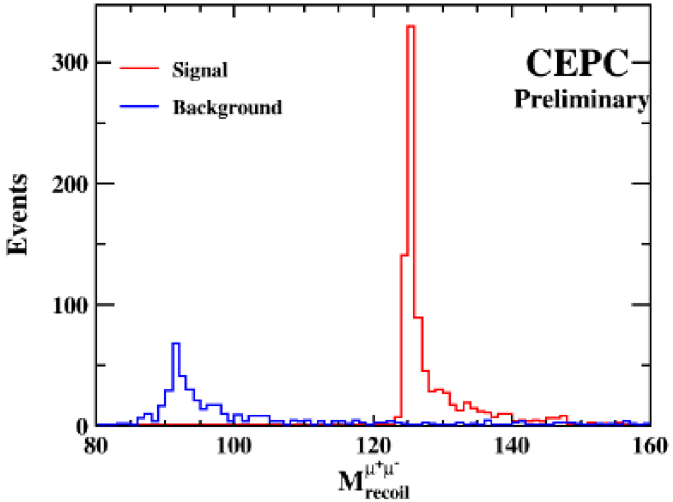

```cpp
// d2 is a new data-frame, a transformed version of d
auto d2 = d.Filter("x > 0")
           .Define("z","x*x + y*y");
// make histograms out of it
auto hz = d2.Histo1D("z");
auto hxy = d2.Histo2D({"hxy","hxy",16,-1,1,64,-1,1},"x","y");
```

- Development of analysis tool for CEPCSW
  - Development of common components (functions)
    - Analysis functions in C++: event selection, filtering, Jet clustering, vertex fitting
    - Python for configuration: define analysis functions, input samples, output variables
  - Performance test

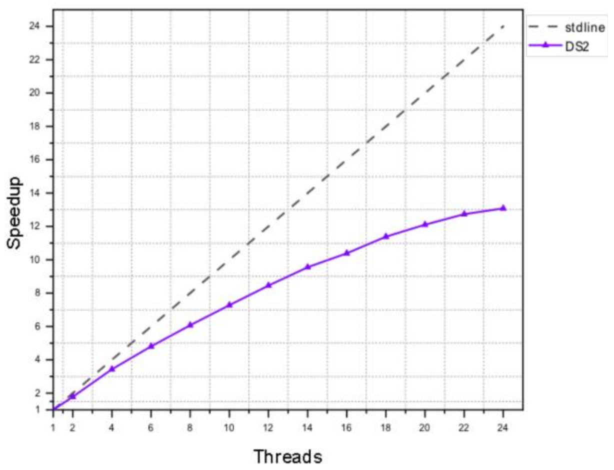
Analysis toolkit based on RDataFrame (2)

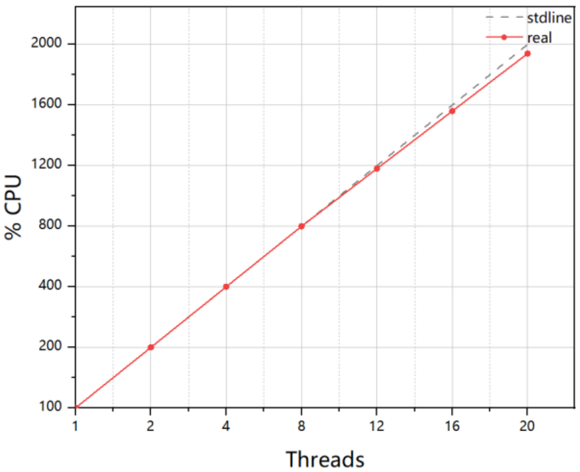
- Several packages are ported from FCC analysis, more are being implemented
  - FastJet, MarlinKinfit
  - Vertex fit, jet tag, PID etc.
- Functionalities and performance test with two analysis channels
  - e+e- -> Z(mumu)H
  - e+e- ->H(2jet) mumu





Identical results with Marlin

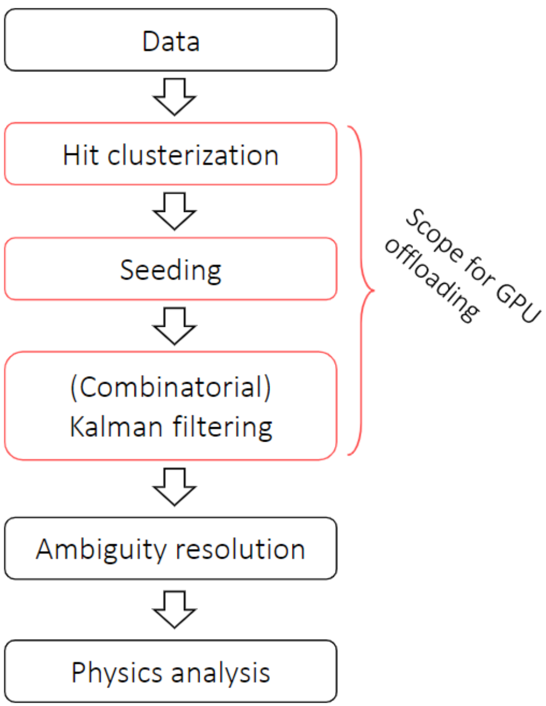


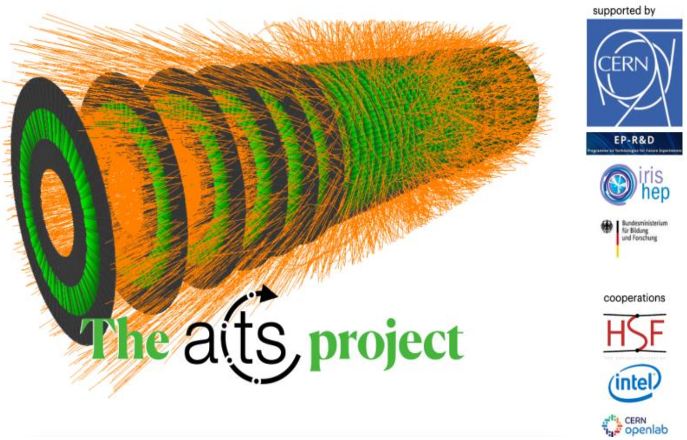


performance test

Heterogeneous Computing (1)

- TRACCC: one of ACTS R&D projects
  - Full chain demonstrator for track reconstruction on CPU/GPU





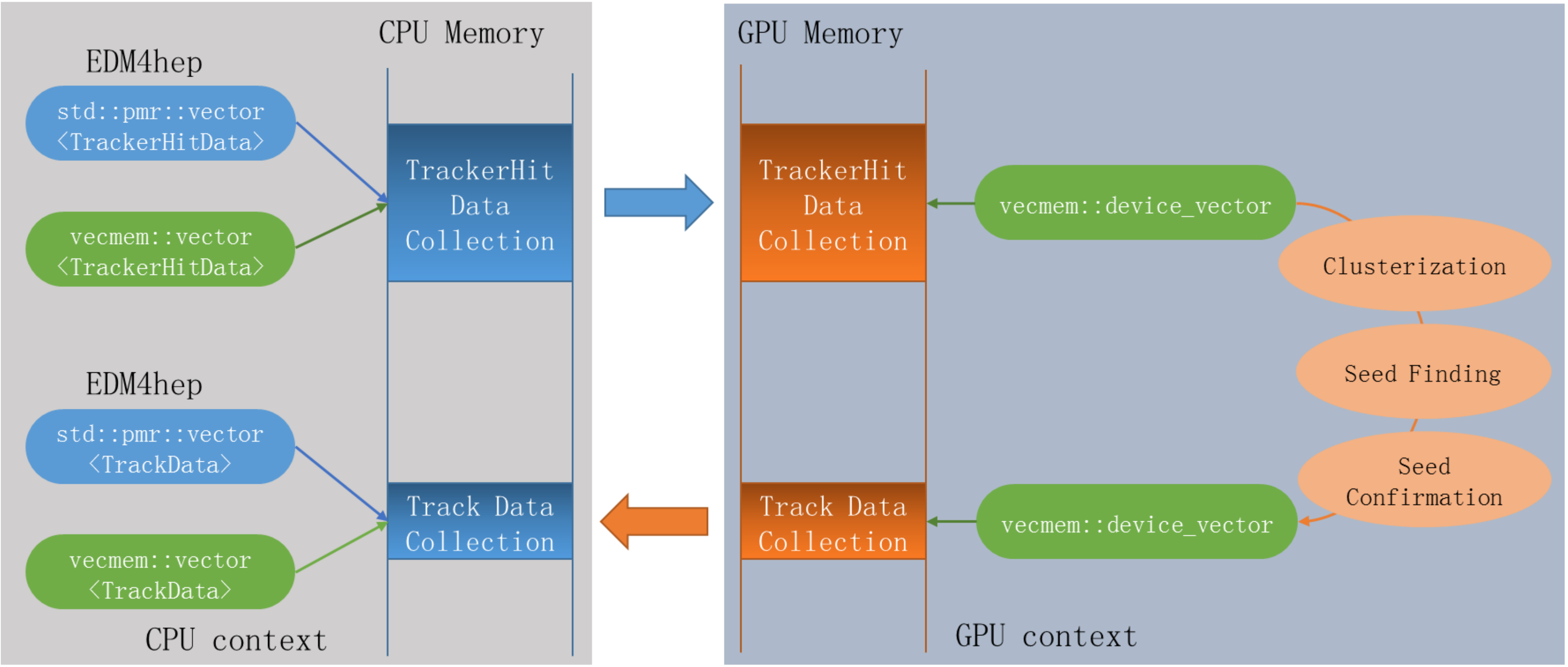
Category	Algorithms	CPU	CUDA	SYCL	Futhark
Clusterization	CCL	✓	✓	✓	✓
	Measurement creation	✓	✓	✓	✓
	Spacepoint formation	✓	✓	✓	○
Track finding	Spacepoint binning	✓	✓	✓	○
	Seed finding	✓	✓	✓	○
	Track param estimation	✓	✓	✓	○
	Combinatorial KF	●	●	○	○
Track fitting	KF	✓	✓	✓	○

✓: exists, ●: work started, ○: work not started yet

https://github.com/acts-project/traccc

Heterogeneous Computing (2)

- Building a bridge between EDM4hep and TRACCC
  - Common memory for both EDM4hep and TRACCC
  - No data conversion is needed between them



Machine Learning Integration

- ONNX/ONNX Runtime have been integrated with CEPCSW
- Provided an example, OrtInferenceAlg,
  - In initialize()
    - Create a session object of ONNX runtime
    - Load and run an ONNX model
  - In execute()
    - Compute output for an input data
- Fast pulse simulation in the drift chamber provided as an example (MLP)

```
Ort::MemoryInfo info("Cpu", OrtDeviceAllocator, 0, OrtMemTypeDefault);

auto input_tensor = Ort::Value::CreateTensor(info,
                                             inputs.data(),
                                             inputs.size(),
                                             dims.data(),
                                             dims.size());
std::vector<Ort::Value> input_tensors;
input_tensors.push_back(std::move(input_tensor));

auto output_tensors = m_session->Run(Ort::RunOptions{ nullptr },
                                     m_input_node_names.data(),
                                     input_tensors.data(),
                                     input_tensors.size(),
                                     m_output_node_names.data(),
                                     m_output_node_names.size());

for (int i = 0; i < output_tensors.size(); ++i) {
    LogInfo << "[" << i << "]"
            << " output name: " << m_output_node_names[i]
            << " results (first 10 elements): "
            << std::endl;
    const auto& output_tensor = output_tensors[i];
    const float* v_output = output_tensor.GetTensorData<float>();

    for (int j = 0; j < 10; ++j) {
        LogInfo << "[" << i << "]" << "[" << j << "] "
                << v_output[j]
                << std::endl;
    }
}
```

```
bool OrtInferenceAlg::initialize() {

   m_env = std::make_shared<Ort::Env>(ORT_LOGGING_LEVEL_WARNING, "ENV");
   m_seesion_options = std::make_shared<Ort::SessionOptions>();
   m_seesion_options->SetIntraOpNumThreads(m_intra_op_nthreads);
   m_seesion_options->SetInterOpNumThreads(m_inter_op_nthreads);

   m_session = std::make_shared<Ort::Session>(*m_env, m_model_file.c_str(), *m_seesion_options);
```

Summary

- The CEPCSW was developed based on the common software stack Key4hep
- The CEPCSW is ready for
  - developing simulation and reconstruction algorithms
  - generating simulated data and performing physics studies
- CEPCSW is being enhanced:
  - Gaussino-based simulation
  - Support for heterogeneous computing
  - Integration with Machine Learning
  - RDataFrame-based analysis tool