Status of CEPC Core Software

Wenxing Fang, Xingtao Huang, Tao Lin,
Weidong Li, Teng Li, Jiaheng Zou

2022.10

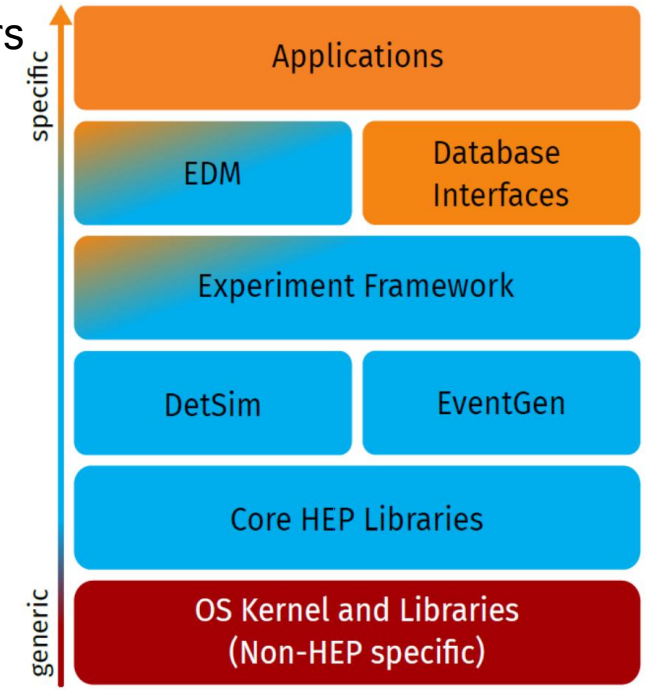
2022 International Workshop on the CEPC

Outline

- Introduction
- Overview of CEPCSW
- Progress and plans
- Summary

Key4hep: Foundation of CEPCSW

- HEP software usually consist of lots of applications
  - Application layer of modules / algorithms /processors performing physics task (*PandoraPFA, FastJet, ACTS,...*)
  - Data access and representation layer including EDM
  - Experiment core orchestration layer (*Gaudi, Marlin, ...*)
  - Specific components reused by many experiments (*DD4hep, Delphes, Pythia,...*)
  - Commonly used HEP core libraries (*ROOT, Geant4, CLHEP, ...*)
  - Commonly used tools and libraries (*Python, CMake, boost, ...*)

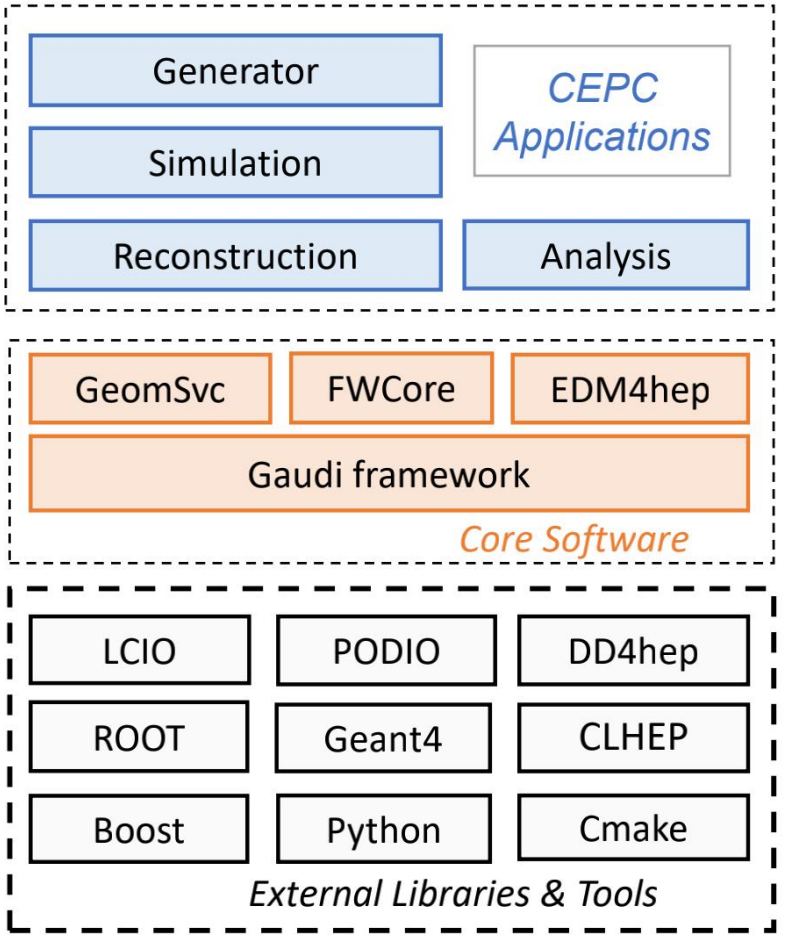


Thomas Madlener, Epiphany Conference 2021

- Turnkey software for future colliders
  - An agreement at Future Collider Software Workshop (Bologna, June 2019)
  - Software components sharing between different experiments
    - CPEC, CLIC, FCC, ILC, STCF

Overview of CEPCSW

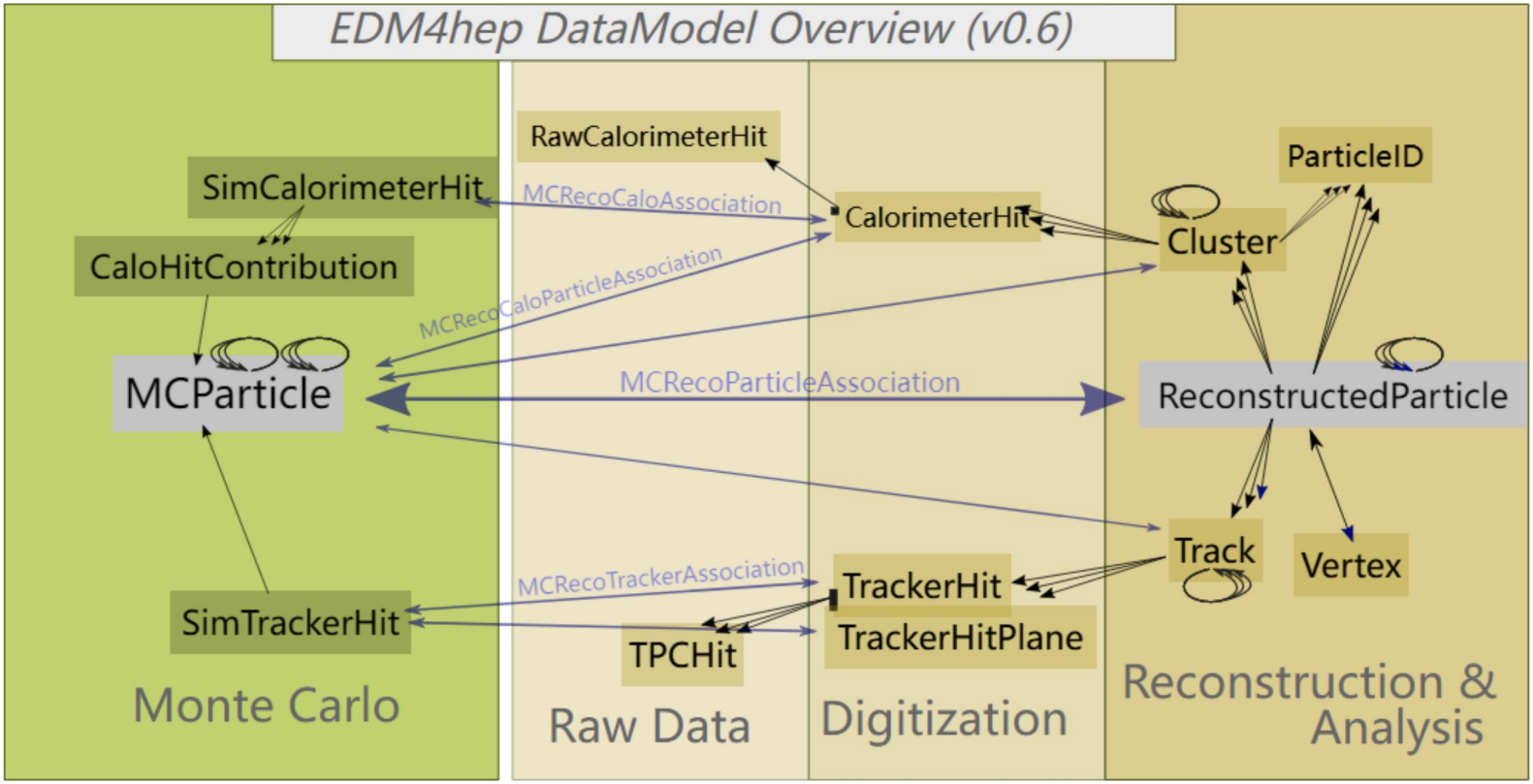
- Based on Key4HEP
- Reuse and extend existing components
  - Gaudi, EDM4hep, DD4hep, ...
- Implement the specific components for CEPC
  - Geometry, generator, simulation and reconstruction algorithms, etc.
- Provide ready-to-work environment to algorithm developers and physicists
  - Porting algorithms from iLCSoft to CEPCSW
  - Integrate/develop more algorithms and features



https://github.com/cepc/CEPCSW

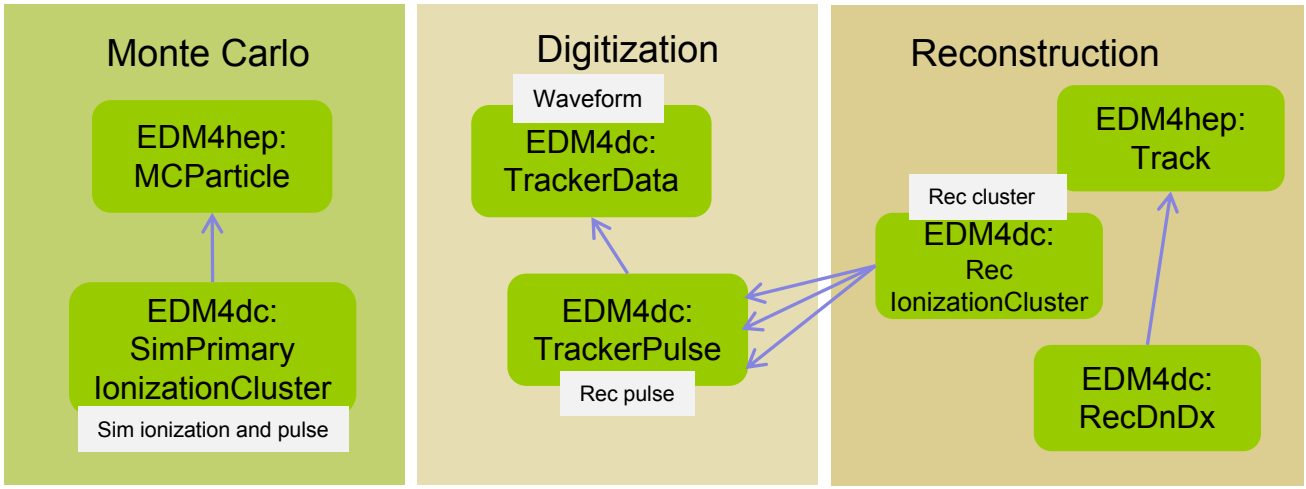
Common Event Data Model: EDM4hep

- CEPC uses EDM4hep, the common EDM that can be used by all communities in the Key4Hep project: ILC, CLIC, FCC-ee & FCC-hh ...
- Efficiently implemented, support multi-threading and potentially heterogeneous computing
- Use podio to generate thread safe code starting from yaml description



CEPC Extension of EDM4hep

- Currently the EDM4hep does not include the EDM for drift chamber dN/dx
- To facilitate dN/dx simulation/reconstruction, EDM4hep is extended via the upstream machanism of podio



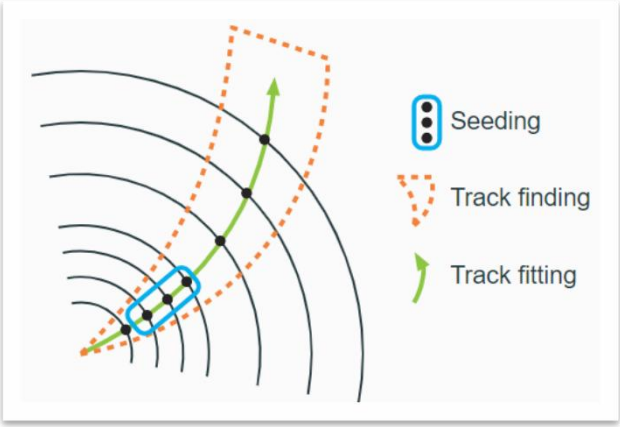
- The extended EDMs also can be used for TPC detector

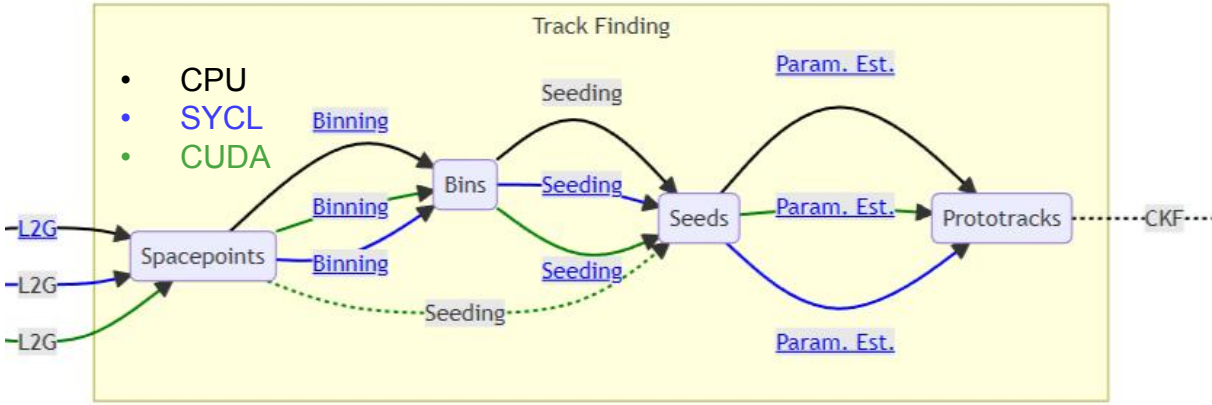
Status of CEPCSW

- CEPCSW is under rapid development
  - Current version: v0.2.6
  - CVMFS distribution: /cvmfs/cepcsw.ihep.ac.cn/prototype
  - Git repo: https://github.com/cepc/cepcsw
- Current efforts are focused on supporting simulation and reconstruction algorithms
- Most core software components are in place, while new technologies are being developed to improve the software and boost performance
  - Heterogeneous Computing
  - Machine Learning Integration based on ONNX
  - Analysis framework based on RDataframe
  - Automated Validation System

Heterogeneous Computing (1)

- Motivation
  - Due to physical limits of transistors, CPU speed increases slowly.
  - The CPU computing resources for data processing are limited.
  - Current existing algorithms need code re-engineering to adopt heterogeneous resources.
- TRACCC is a project below ACTS to demonstrate tracking chain on different kinds of computing hardware (CPU/GPU/FPGA).

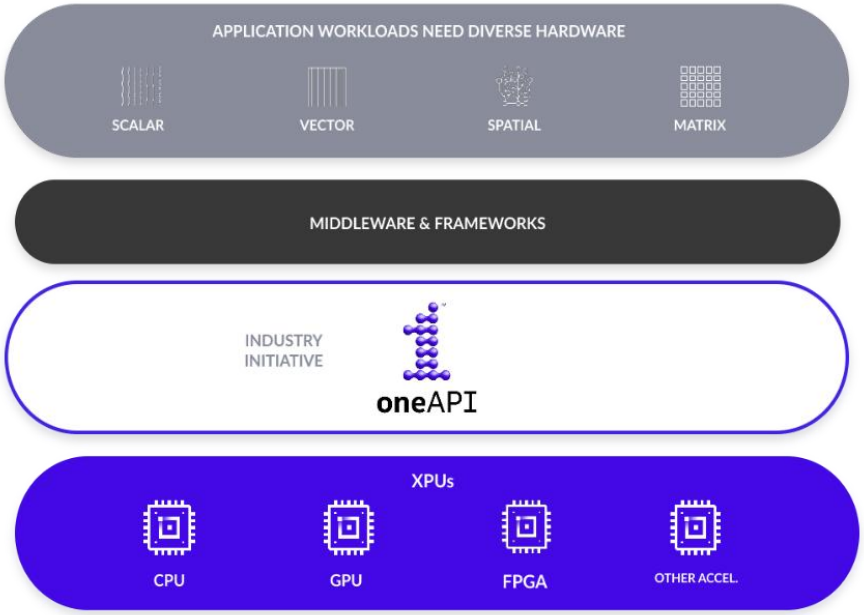




https://github.com/acts-project/traccc

Heterogeneous Computing (2)

- We plan to run the TRACCC seeding algorithm within the CEPCSW software framework, look at the possibility to extend the algorithm to accommodate the geometry of the CEPC pixel detector.
- At the same time, we would like to do some tests with EDM4hep to find out the performance of the event data model in the heterogeneous computing environment.



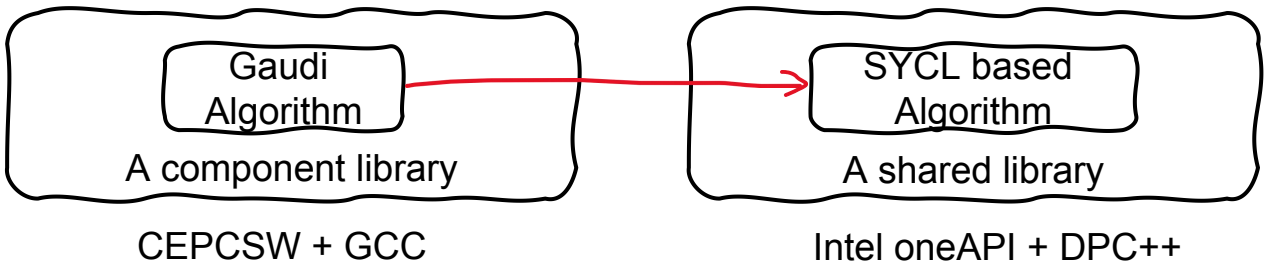
- SYCL enables the definition of data parallel functions, which can be offloaded to CPU/GPU/FPGA devices, by providing required APIs and runtime libraries.
- The oneAPI DPC++ is an extension on top of SYCL. So it can provide a unified programming model across multiple hardware architectures.

Heterogeneous Computing (3)

- ✓ The first work which has been done is to run TRACCC in a standalone environment. We managed to build and run TRACCC on both CPU/GPU.

Config	Hardware	OS	Compiler	SYCL backend	Bulid traccc	Run traccc
1	Intel CPU (IHEP login node)	CentOS 7.8	LCG 101 (GCC 10.3 + clang 12) + oneAPI DPC++	CPU	OK	OK
2	Intel CPU + NVIDIA RTX 8000 (workstation)	CentOS 7.9	LCG 101 (GCC 11.1) + intel/llvm (2021-12)	CUDA 11.2	OK	OK

- ✓ Then we integrate the TRACCC seeding algorithm to the CEPCSW by writing a Gaudi algorithm as a wrapper.
  - With GCC, we can compile the algorithm into a component library.
  - Then another library containing the SYCL-based implementation, which the Gaudi algorithm depends on, was created via clang++ from DPC++.



Even though two different compilers were used, the Gaudi algorithm can be run successfully and the SYCL code is invoked properly.

Machine Learning Integration (1)

- Motivation
  - Integrate different pre-trained machine learning models into the framework
- ONNX[1] is an open format built to represent machine learning models.
  - Support to convert from other models to ONNX, such as Tensorflow, PyTorch etc.
  - Easy to run inference on different platforms, such as ONNX Runtime, ONNX MLIR etc.
- Some applications of ONNX in HEP
  - Fast simulation in Geant4 using ONNX inference interface [2]
  - Fast Inference for Machine Learning in ROOT TMVA [3]

[1] https://onnx.ai
[2] Anna Zaborowska *et al.*, Fast Simulation : from Classical to Machine Learning Models
[3] Sitong An et al., Fast Inference for Machine Learning in ROOT/TMVA

Machine Learning Integration (2)

- Example has been provided in CEPCSW
  - Based on ONNX C++ runtime
- An algorithm only needs to
  - Manage the session object of ONNX runtime
  - Specify the model path.
  - Take care of the input and output data

```
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;
    }
}
```

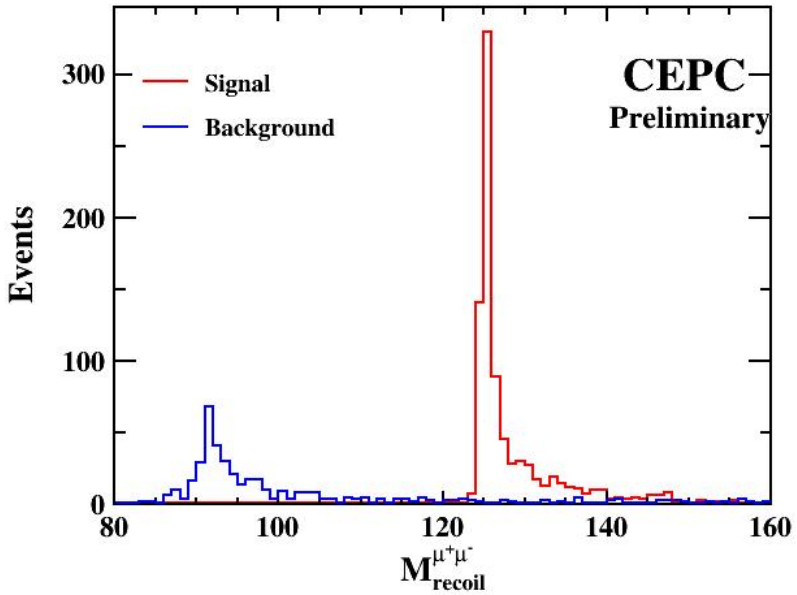
```
bool OrtInferenceAlg::initialize() {

    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);
```

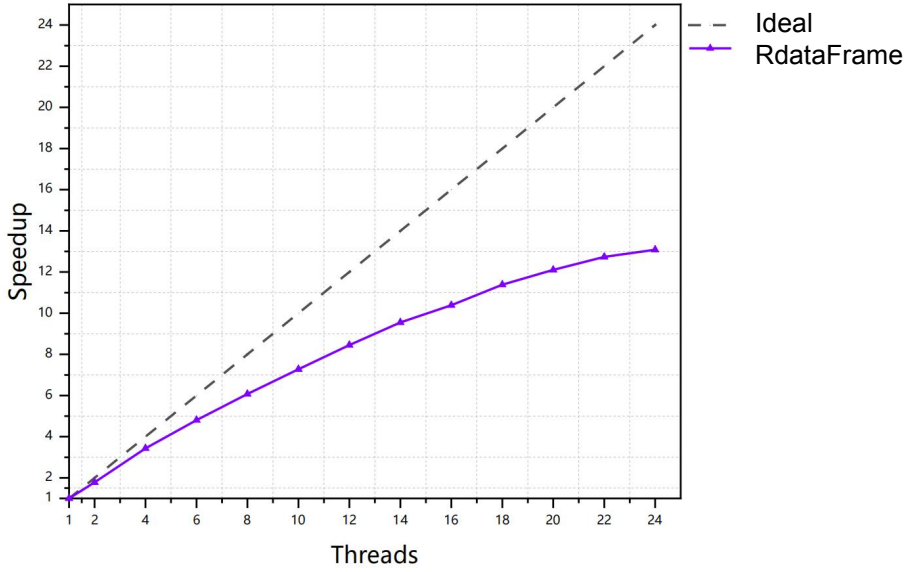
Development of Analysis toolkit based on RDataFrame

- RDataFrame provides powerful and flexible way analyzing data
  - Support declarative programming
  - Fastest way to analyze data
  - Full support analysis in both Python and C++
  - Actively used by FCC-ee for flavour, higgs and top physics
- Development of analysis tool for CEPCSW
  - Use CEPCSW to convert LCIO data produced with Marlin to EDM4hep
  - Common components (functions) for analyzing EDM4hep data are being developed
    - Analysis function code in C++: event selection, filtering, vertexing, PID, Jet clustering, producing ROOT ntuples,…
    - Python for configuration: define analysis functions, output variables, input samples, etc…

Development of Analysis toolkit based on RDataFrame

- The development starts with a inclusive analysis: Higgs recoil analysis and Higgs width measurement in e+e- -> Z(mumu)H
  - Basic functionalities are tested: same results obtained from Marlin and RDataframe
  - Performance tests are performed with multi-threading enabled

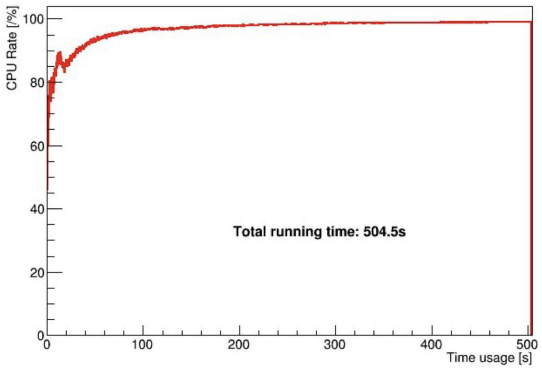


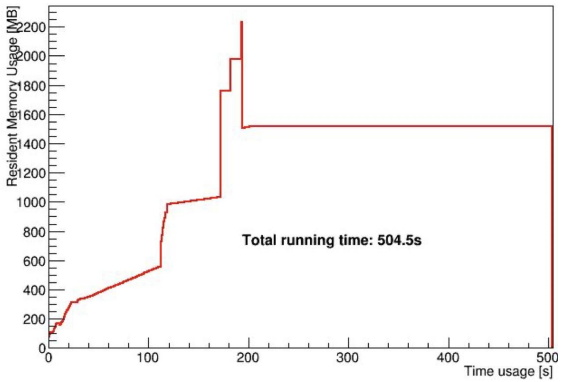


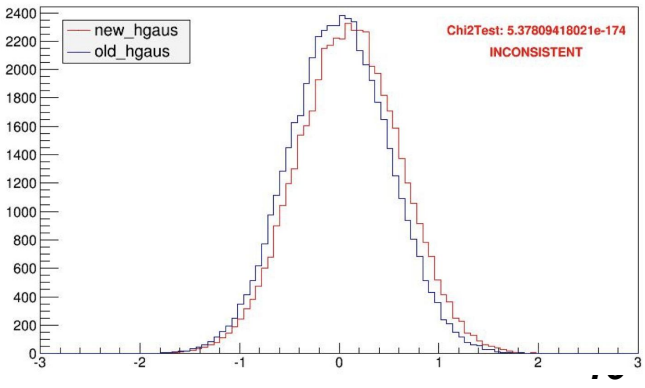
Scalability of RDataFrame with multithreading enabled

Automated Validation System

- An automated validation system is being developed for software validation at different levels
  - Unit test, integrated test, performance test, physical validation etc.
- A toolkit is developed for building software validation workflow
  - Provide interfaces to define and run unit tests
  - Support performance profiling
  - Support results validation based on statistical methods
- Automated physical validation system based on massive data production is being developed

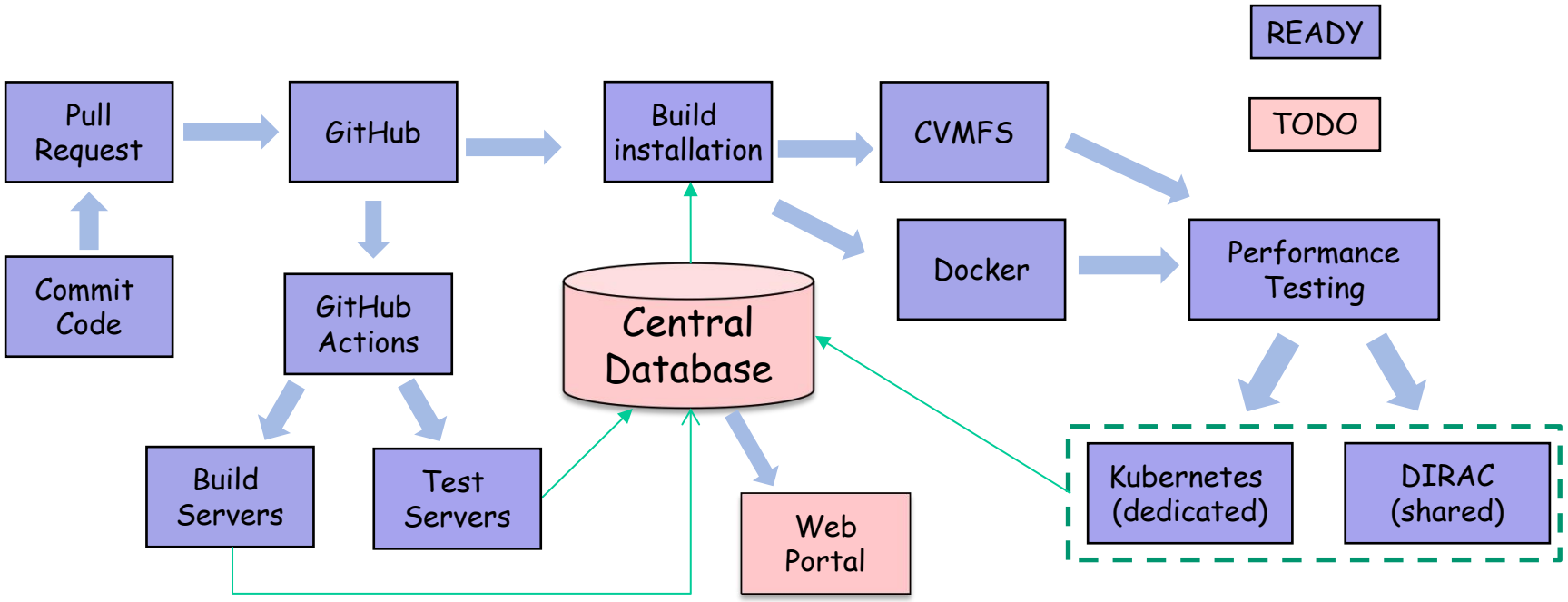






Automated Validation System

- The validation system is being integrated with the Github Action system
  - Full validation workflow can be triggered by commit/merge-request
  - A web-based monitoring dashboard is also being developed
- ~ O(200) cores are now available for running validation jobs



Summary

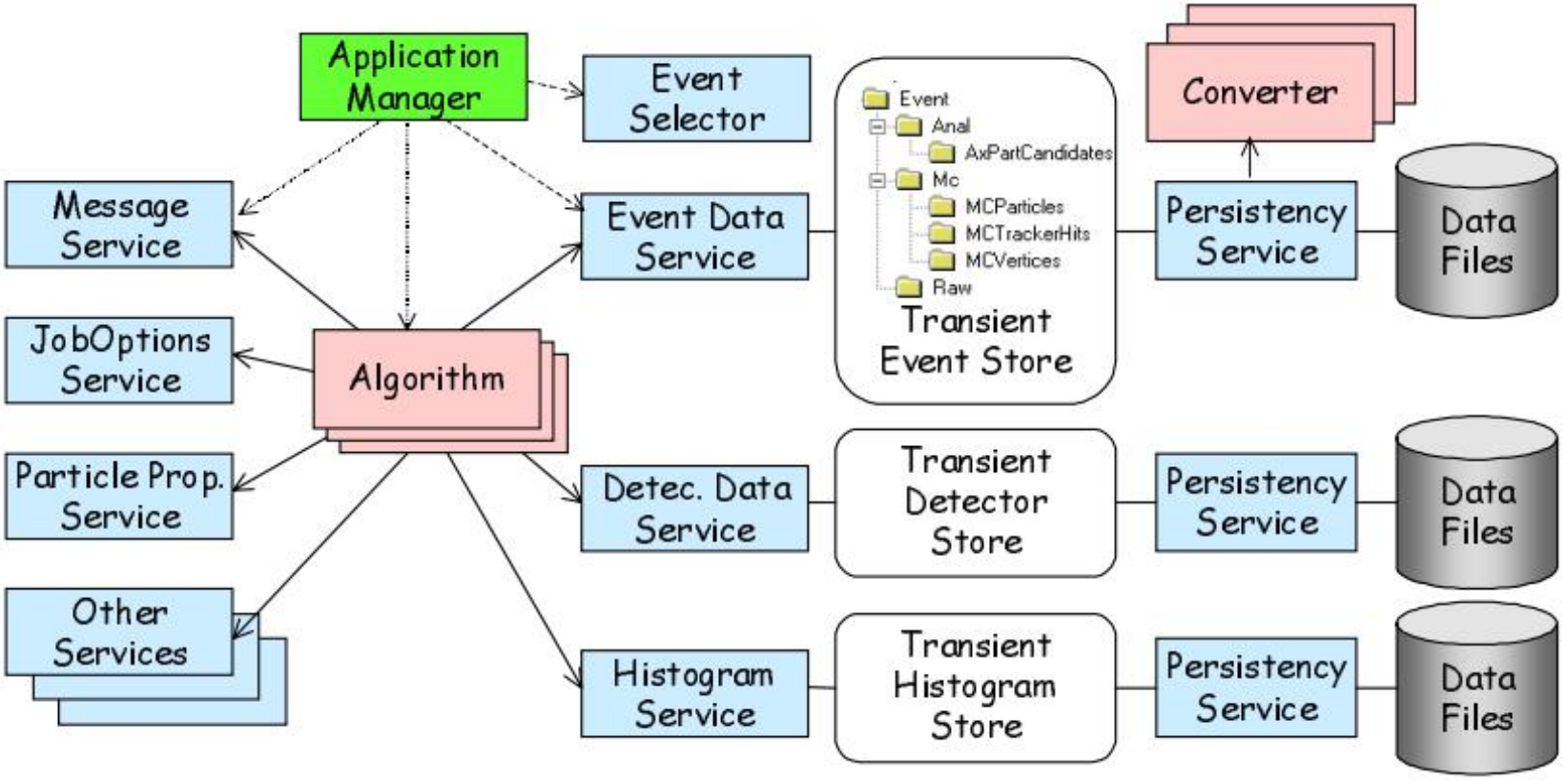
- CEPCSW framework is being developed in collaboration with the Key4hep project
- Most CEPCSW core software components are in place and function well to support detector simulation and reconstruction studies
- Latest developments are focused on:
  - EDM extention
  - Heterogeneous computing
  - Integration of ML models
  - RDataFrame based Analysis framework
  - Automated validation system

Welcomed to joining CEPCSW!
We hope to work together with developers in the community.
https://github.com/cepc/cepcsw

Backup

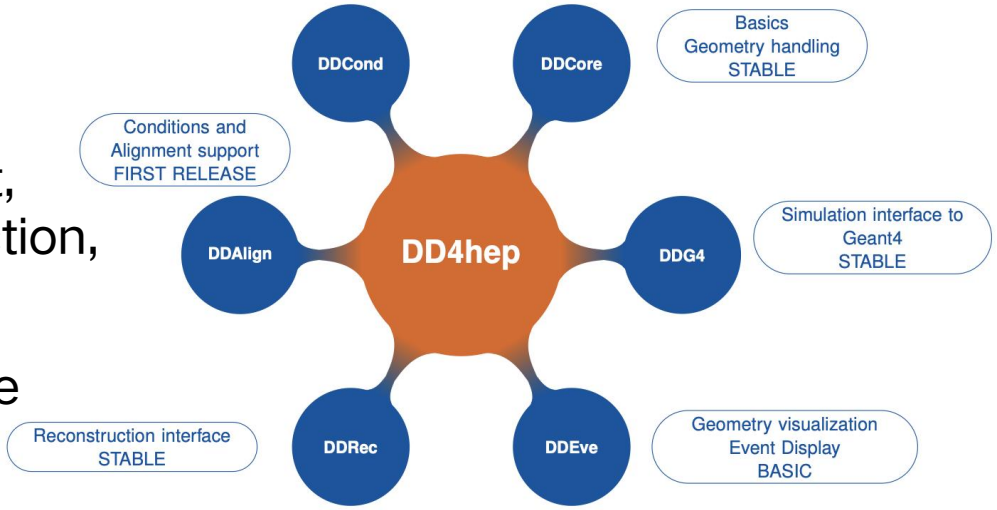
Data Processing Framework: Gaudi

- Modular design and well defined component interfaces
- Three basic categories of data characterized by their lifetime
- Seperation of algorithm, transient and persistent data in the event loop



Geometry Description Toolkit: DD4hep

- DD4hep provides complete detector description in CEPCSW
  - Provides geometry, materials, visualization, readout, calibration…
- Single source of information to ensure consistent description
  - In simulation, reconstruction, analysis
- Supports full experiment life cycle
  - Detector concept development, detector optimization, construction, operation
  - Facile transition from one stage to the next
- See latest developments:
  https://github.com/AIDASoft/DD4hep



Data Input/Output

- The default EDM4hep data format: ROOT
- k4FWCore: the default data I/O components
  - PodioDataSvc: read/write podio data collections in ROOT
  - DataHandler: register data collections to Gaudi
- k4LCIOReader: read LCIO data generated by Marlin
  - Data converters