STCF Core Software Status

Teng LI on behalf of the STCF core software development team

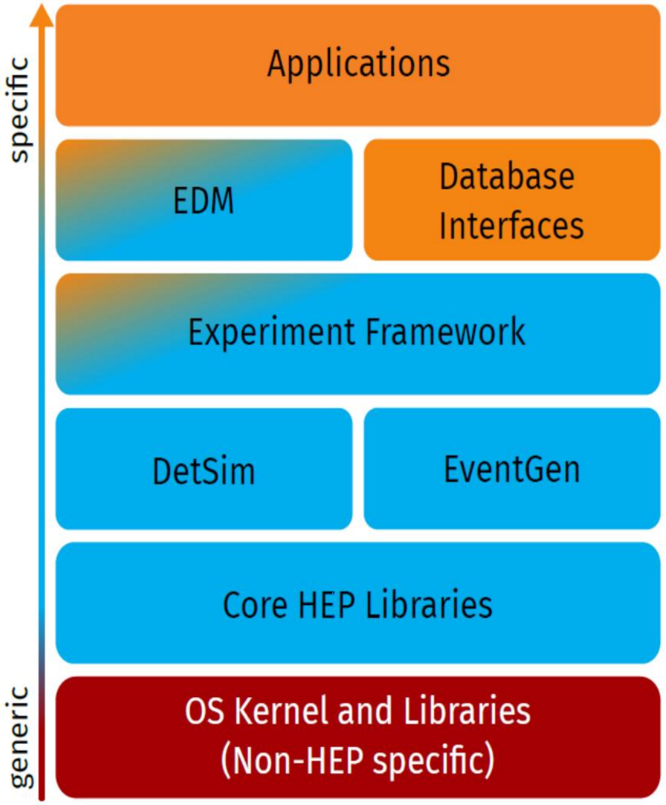
Shandong University

2025-7-4

2025年超级陶粲装置研讨会 湘潭

Main R&D Challenges for STCF Software

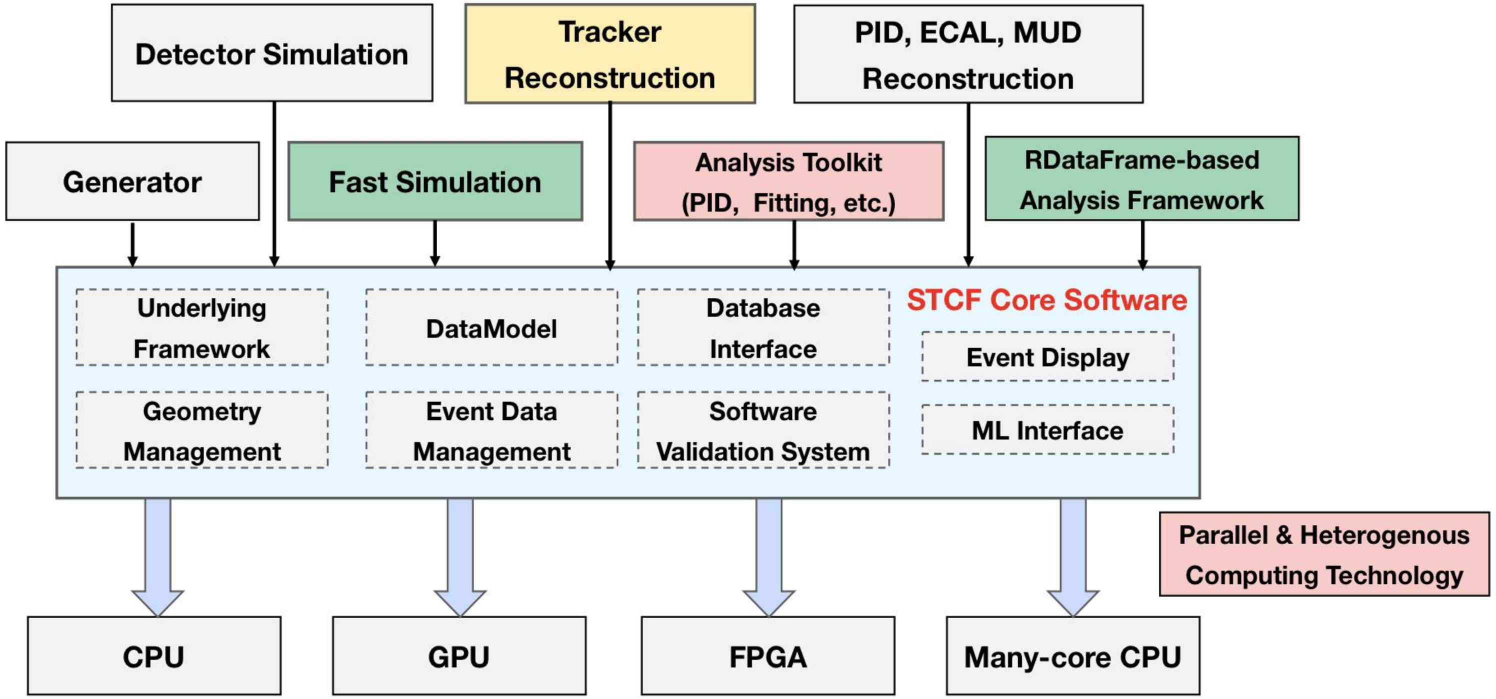
- Main R&D challenges and innovations for STCF core software
  - The amount of data requires much more advanced performance
    - Relying on pure single-threaded CPU resource to process **hundreds of PB** of data is hardly realistic
    - Parallel computing, as well asheterogeneous resources, need to be considered to overcome the challenges.
    - The core software needs to provide ready-to-use development and run time environment
  - Adoption of common software developed for future colliders
    - OSCAR is developed partially based on Key4hep, including EDM based on podio, geometry based on DD4hep etc.
  - Better support of ML-based applications



Key4hep
Thomas Madlener,
Epiphany Conference 2021

Overview of STCF Core Software

- The task of the STCF core software
  - To fulfill official offline data processing tasks, i.e. detector simulation, digitization, calibration and reconstruction
  - Provide a common platform for users to perform data analysis
- Overview of STCF core software



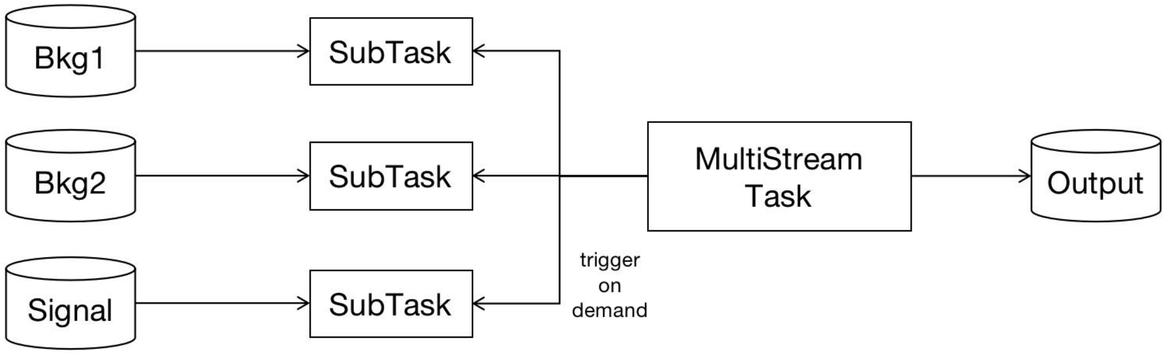
- The underlying framework
- Event data management
- Detector description and conditions data management
- Event display
- Support of ML, parallel computing, and heterogeneous computing
- Software and physics validation
- Software build, installation and distribution

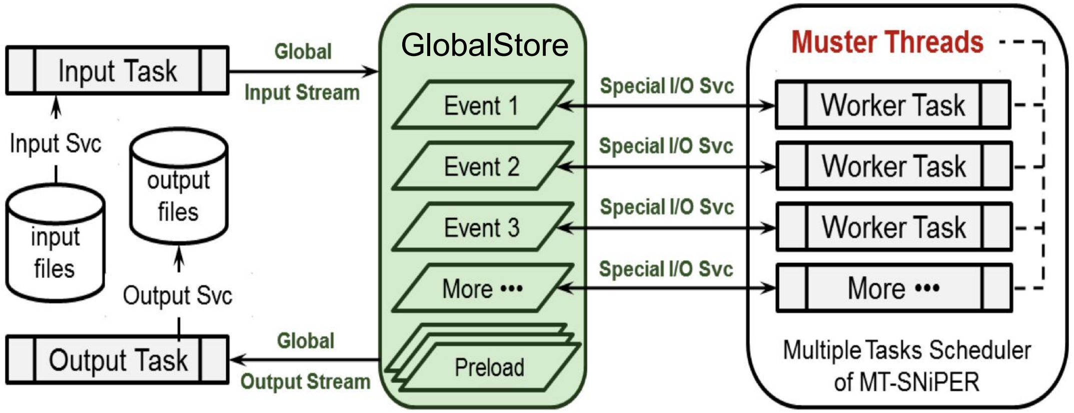
Recent Progress of OSCAR

- Several new releases since FTCF2025
  - 2.6.0, 2.6.1, **2.6.2 (current release)** and a few pre-releases
  - Most functionalities in place and stable, supporting MC production and physics studies
  - Supporting physics studies for the TDR as the first priority
- Major updates in 2025:
  - Great optimization of disk consumption and running speed of simulation and reconstruction jobs
  - Release of fast simulation software package
  - Release of Global-PID based on weighted combined likelihood method
  - Release of ACTS-based tracking
  - Major updates of EventDisplay
  - Lots of optimizations and fixes for various physics simulation studies
- Dedicated tutorial in 2025 Feb. Many analyzers get envolved

Underlying Framework: SNiPER

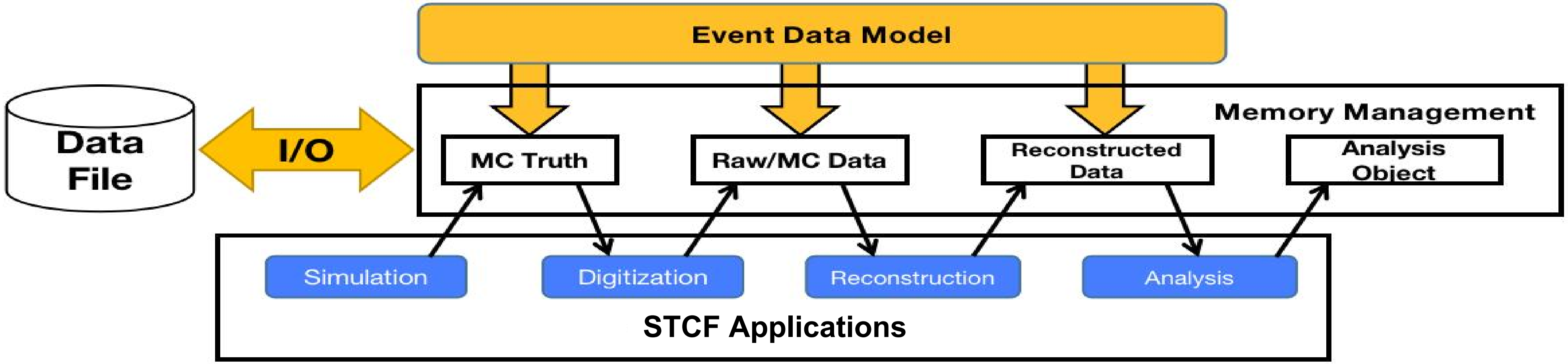
- The underlying framework builds the skeleton of OSCAR
  - Provide basic functionalities of **event loop control, algorithm scheduling, thread management, user interface, job configuration, logging** etc. (Like Gaudi for BOSS)
- OSCAR adopts SNiPER as the underlying framework
  - Lightweighted, efficient and highly extendable
  - Developed since 2012, maintained by **10+ developers from IHEP, SDU, etc.**
  - Adopted by JUNO (neutrino), LHAASO (cosmic ray), nEXO (neutrinoless double beta decay) and HERD (dark matter)
- Recent updates
  - Better support for inter- and intra- event level parallism



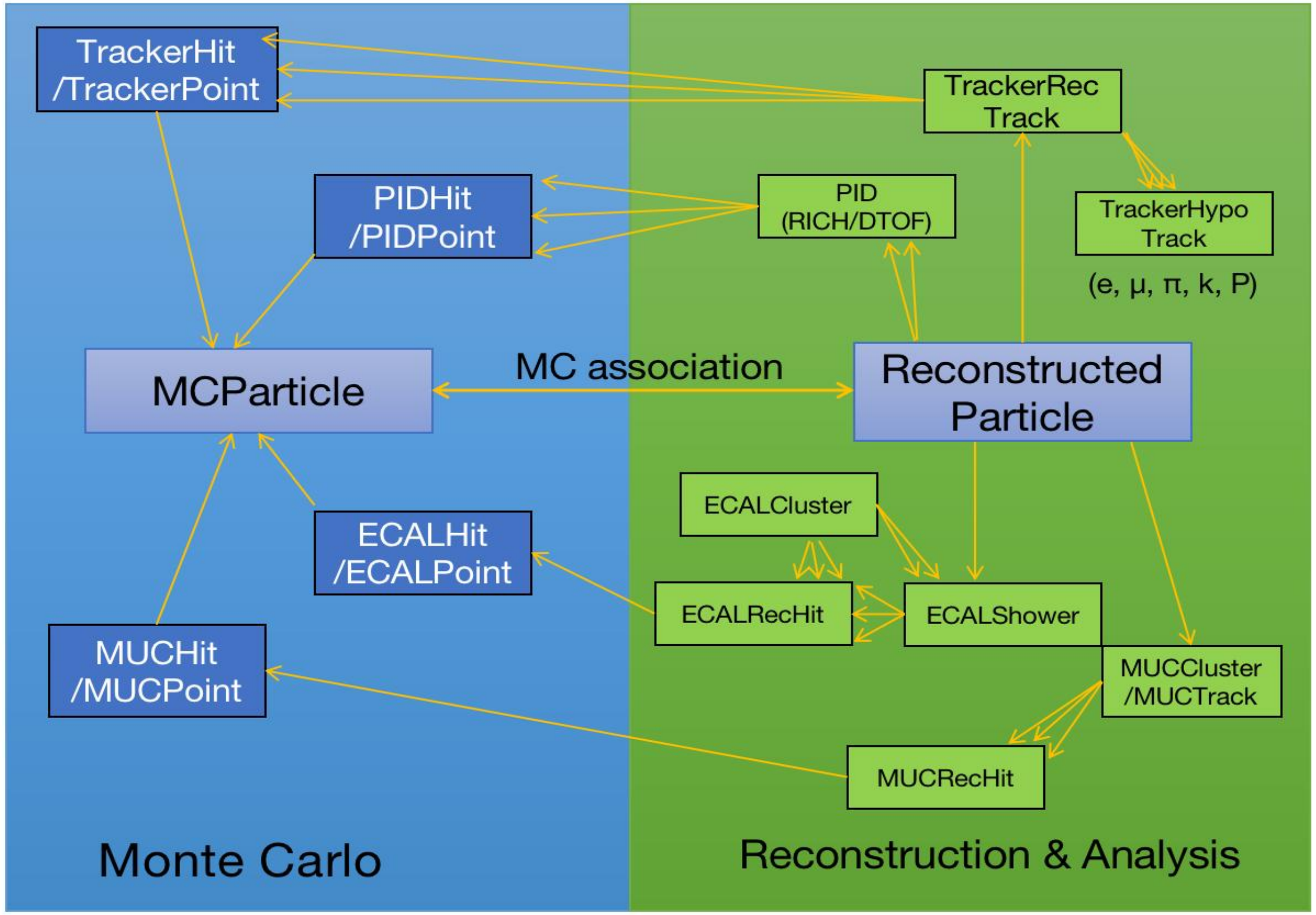


Event Data Management

- Event data management is the most crucial part of the framework
  - **Provide tools to define the Event Data Model (EDM)**
    - The definition of physics event data (MC particles, hits, readouts, tracks, clusters, reconstructed particles),
    - Construct relationship between data objects (e.g. which particle makes these hits? Which hists are used to fit a track, etc.)
  - **Provide automated memory management and data I/O functionalities**
  - **Provide backward and forward compatibility, very important for the long running of STCF.**
  - **Guarantee thread-safety, and provide high performance for MT applications**



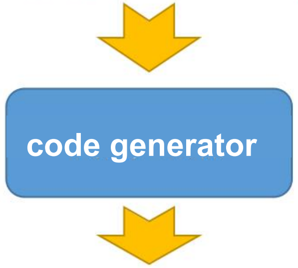
Event Data Model and of OSCAR



EDM classes defined in OSCAR

YAML

```
MCParticle:
  Description : "Data class for storing Monte Carlo particles"
  Author : "SDU"
  Members :
    - int trackID                  // track ID
    - int PDG                      //PDG code of the particle
    - int generatorStatus          //status of the particle as defined by the generator
    - int simulatorStatus          //status of the particle from the simulation program - use BIT constants
    - int type                     //particle type. only generatorStatus== 1 or 2 has the type
    - float charge                 //particle charge
    - float time                   //creation time of the particle in [ns] wrt. the event, e.g. for preassig
    - double mass                  //mass of the particle in [MeV]
    - Vector3d vertex              //production vertex of the particle in [mm].
    - Vector3d endpoint            //endpoint of the particle in [mm]
    - Vector3f momentum            //particle 3-momentum at the production vertex in [GeV]
    - Vector3f momentumAtEndpoint //particle 3-momentum at the endpoint in [GeV]
    - Vector3f spin                //spin (helicity) vector of the particle.
```



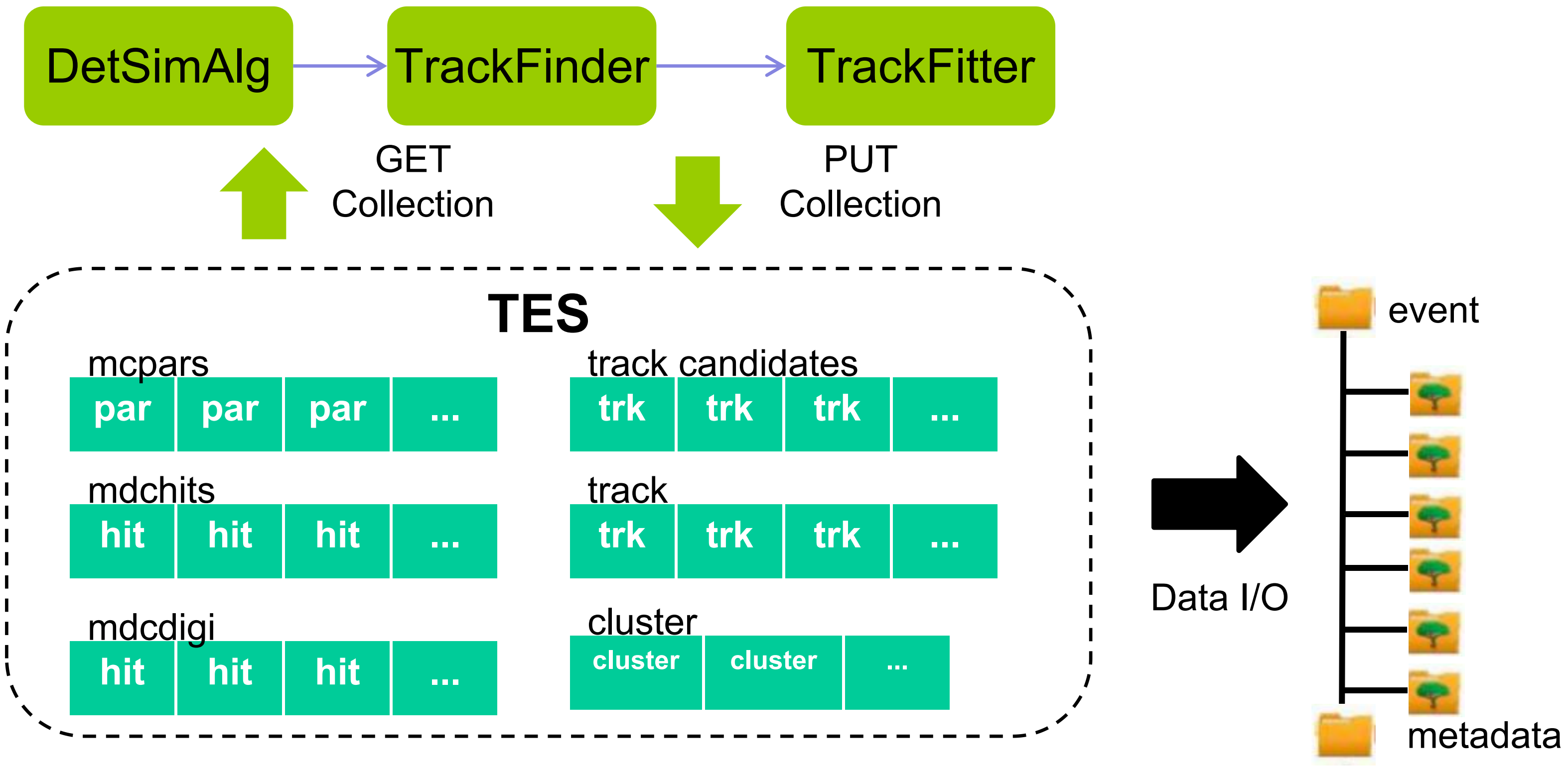
C++

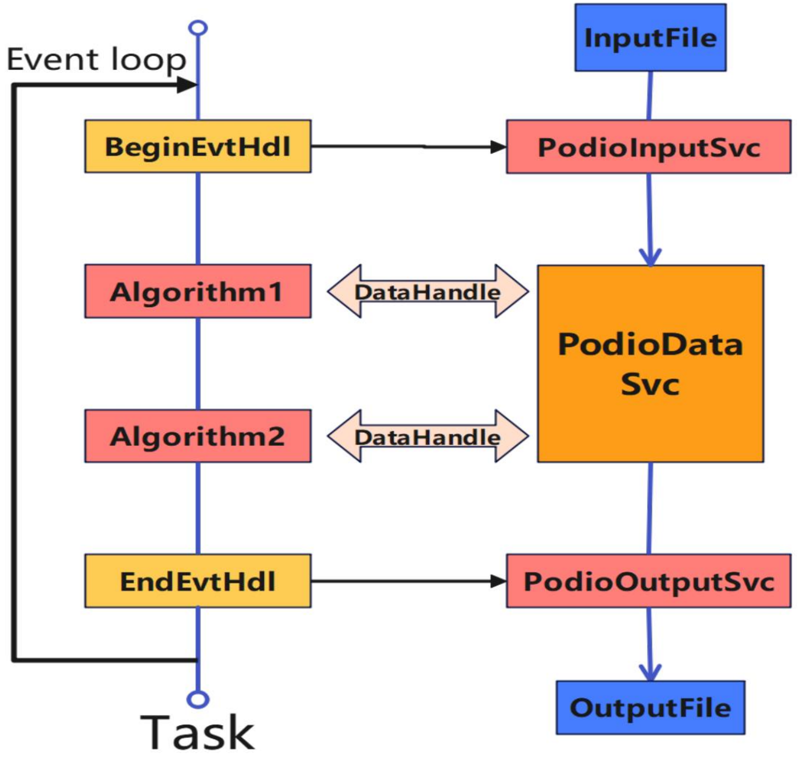
MCParticle.h	MCParticle.cpp
MutableMCParticle.h	MutableMCParticle.cpp
MCParticleObj.h	MCParticleObj.cpp
MCParticleData.h	MCParticleData.cpp
MCParticleCollection.h	MCParticleCollection.cpp
MCParticleCollectionData.h	MCParticleCollectionData.cpp

Based on YAML definition, generate EDM C++ code accordingly

Transient Event Store and Data I/O

- **Transient Event Store** (TES) is where EDM objects are stored in memory
  - TES in OSCAR is developed based on podio::EventStore
  - Being migrated to podio::Frame (code mostly ready)
  - Support both serial and parallel applied software



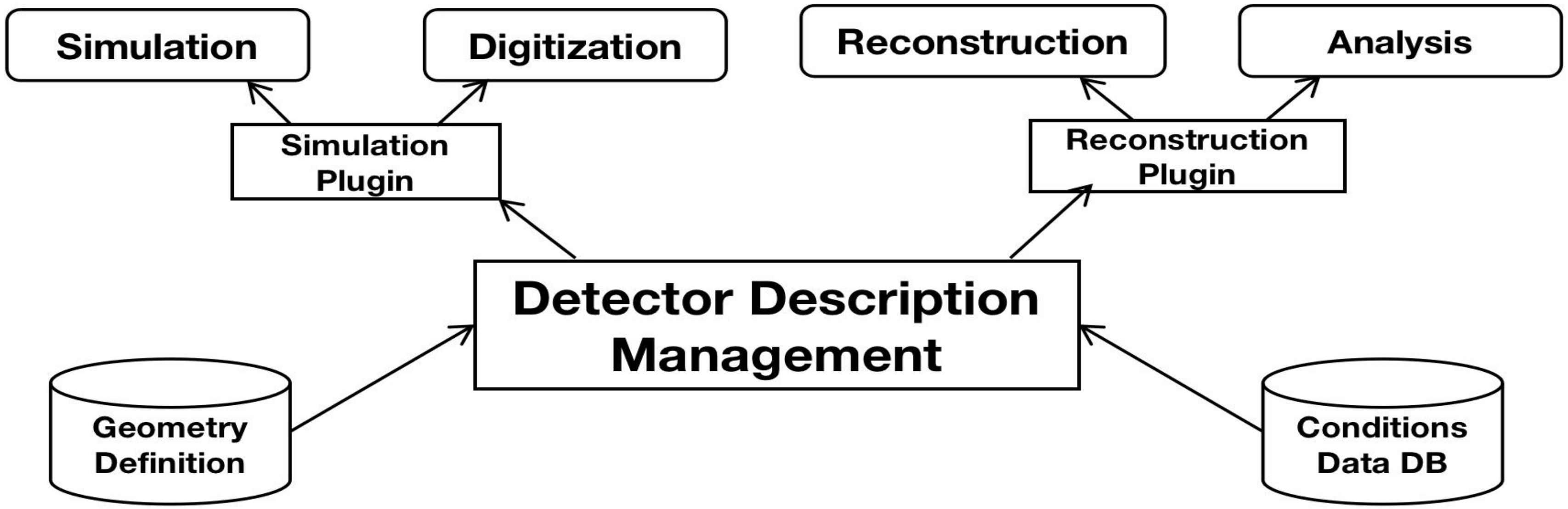


Implementation of TES and data I/O

- PodioDataSvc
- PodioInputSvc
- PodioOutputSvc

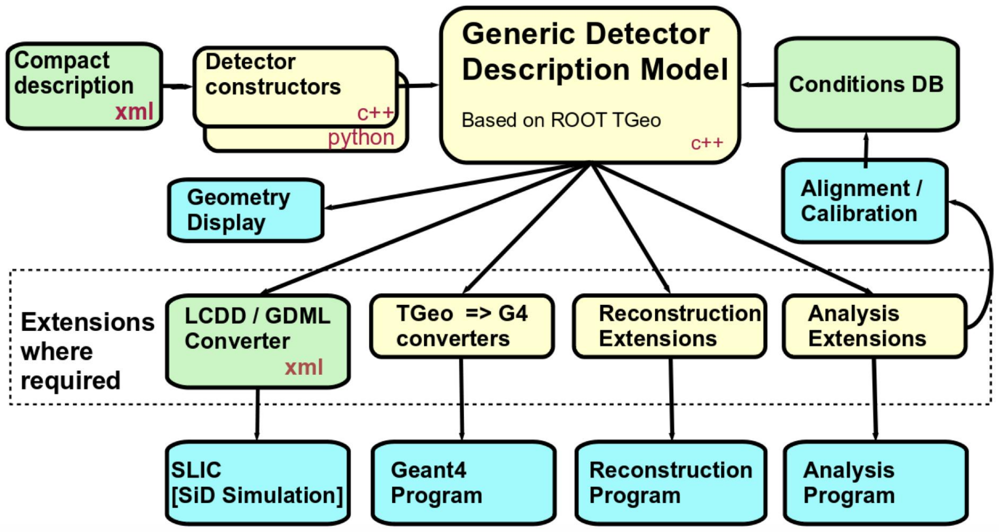
Detector Description Management: **Requirements**

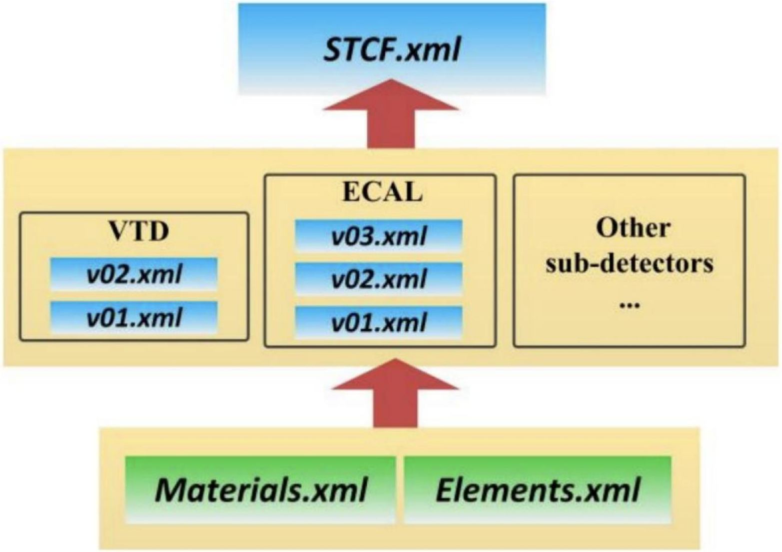
- A powerful detector description management system is necessary across the full offline data processing workflow
  - Provide simple method for **geometry description definition**
  - Provide consistent detector description for all applications
  - Provide **geometry conversion** for different applications, and versioning management
  - Provide interface for conditions data and detector alignment
  - Provide simple and **ready-to-use interfaces** for applications

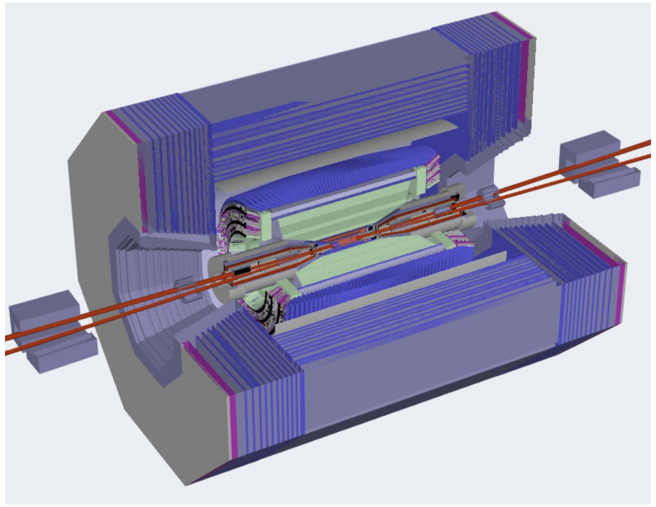


Geometry Management System

- Geometry Management System (GMS) in OSCAR is based on DD4hep
- Single source of detector information for detector description, simulation reconstruction and event display
  - Complete geometry defined with XML files and C++ parser
  - Various plugins for applications
  - Interface for alighment and conditions data
- Full detector defined and stably used, now being further refined (e.g. implementing supporting structures)

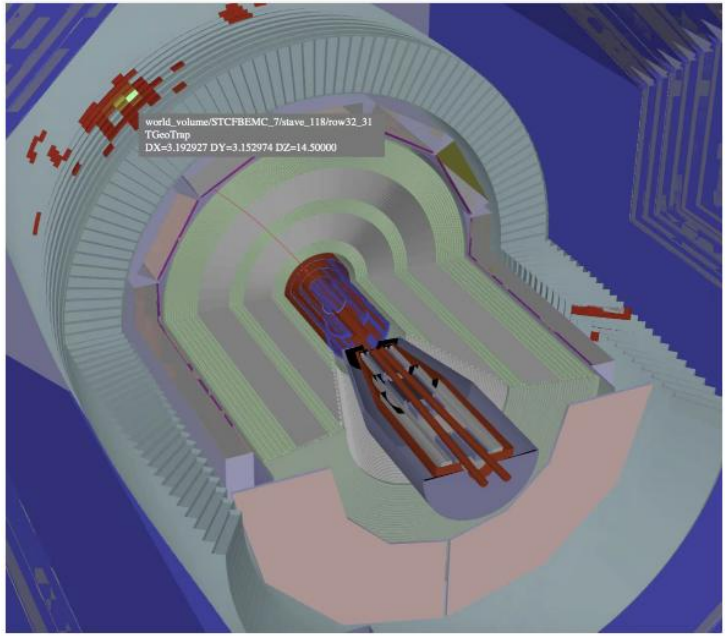


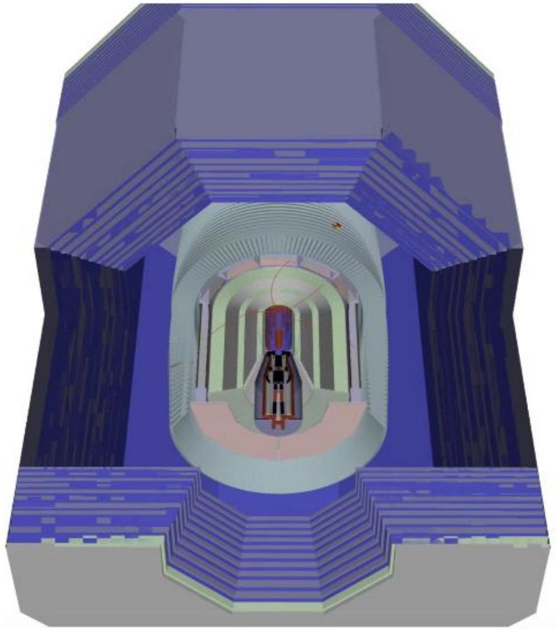


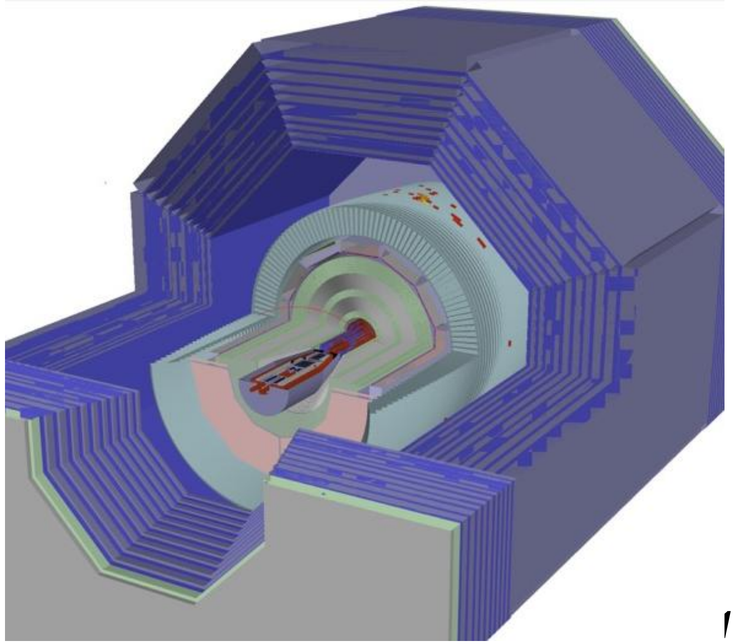


Detector and Event Display

- A common geometry and event display system is being developed
  - Based on Web3D technology and the open-source JSRoot framework
  - 3D engine and graphic libbrary based on Three.JS
  - Geometry information from detector description from DD4hep (XML), and event data from podio
  - Major updates in 2025, now supporting the latest event data format (display of tracker and ECAL hits, tracks and showers are supported)

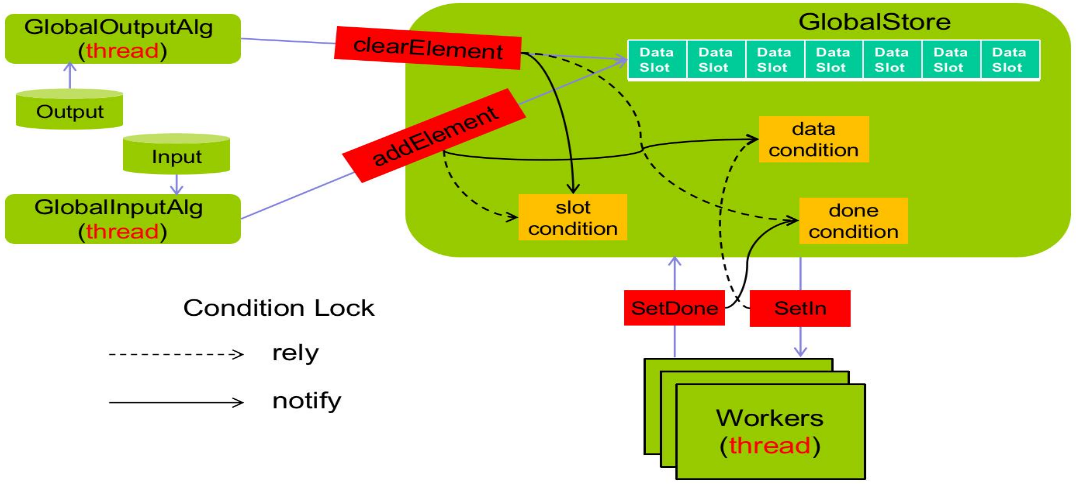


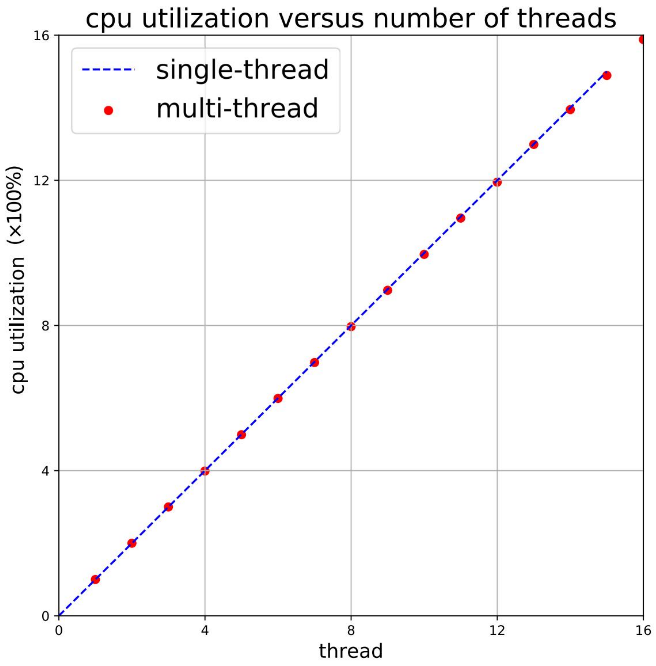


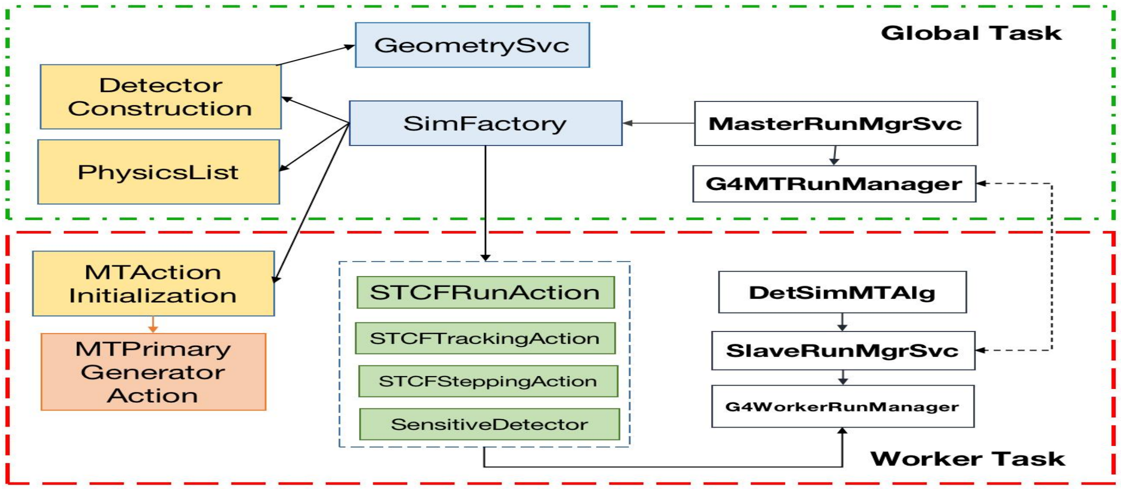


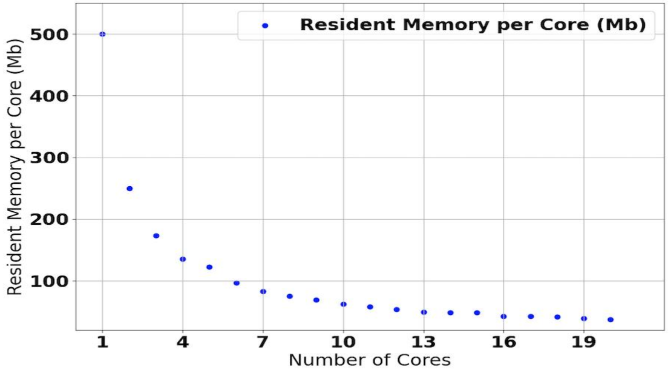
Parallelized Data Processing

- Parallelized detector simulation and reconstruction applications are implemented
  - Basic performance tests show promising scalability



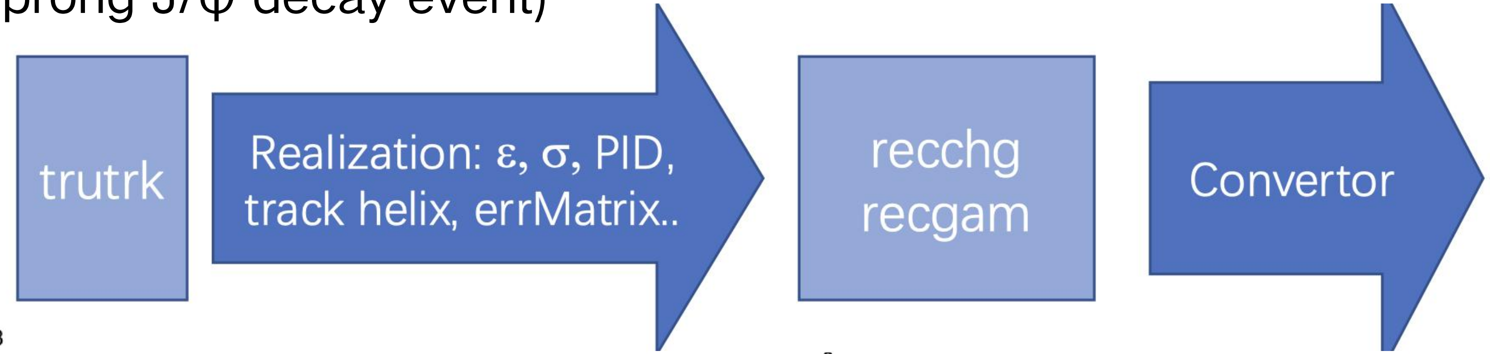


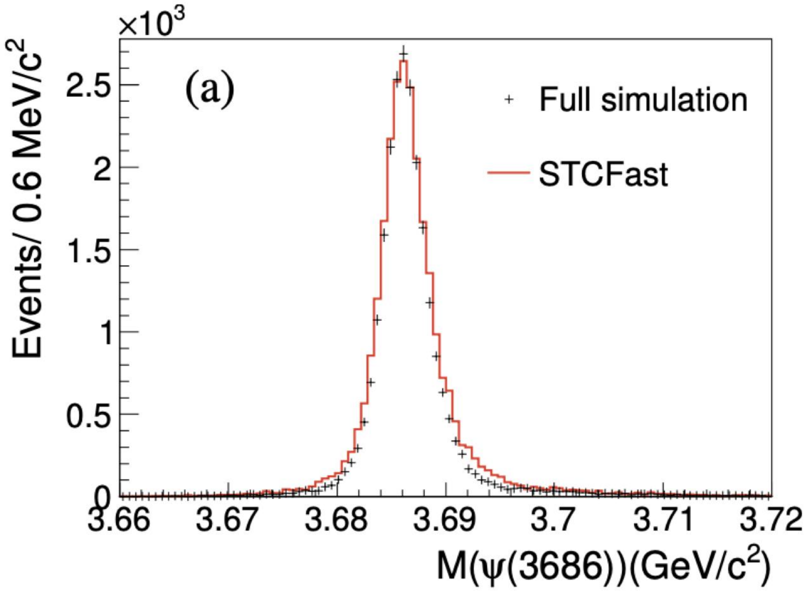


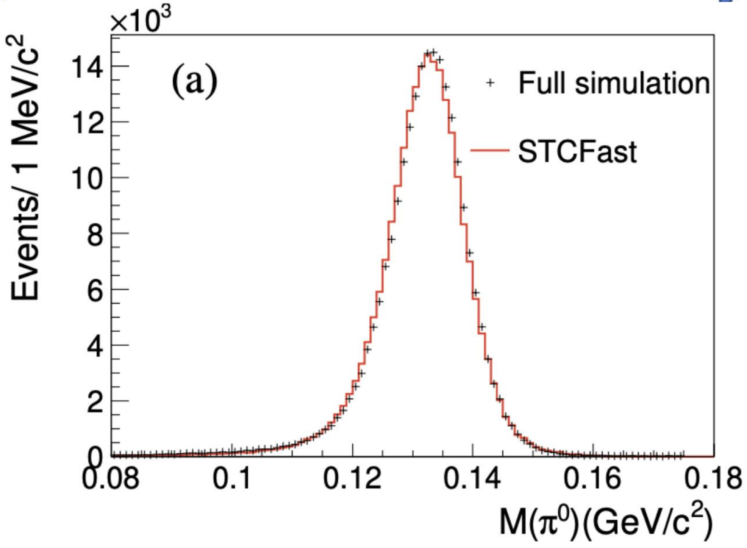


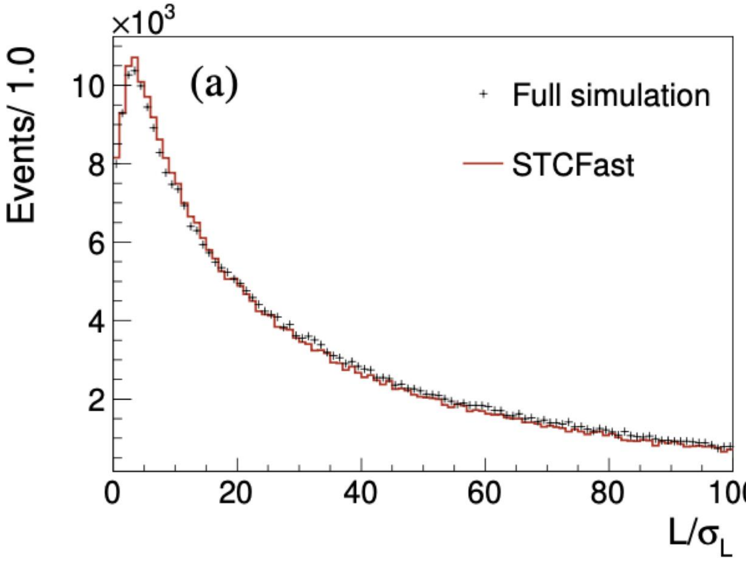
Fast Simulation Framework

- The fast simulation framework is now integrated with OSCAR (more features being developed)
- Flexible for different detecting response and friendly for physics sensitivity study
- Much faster and less disk storage consuming compared to full simulation (~2ms per event, ~2kb storage for a 4prong J/ψ decay event)









Machine Learning Support

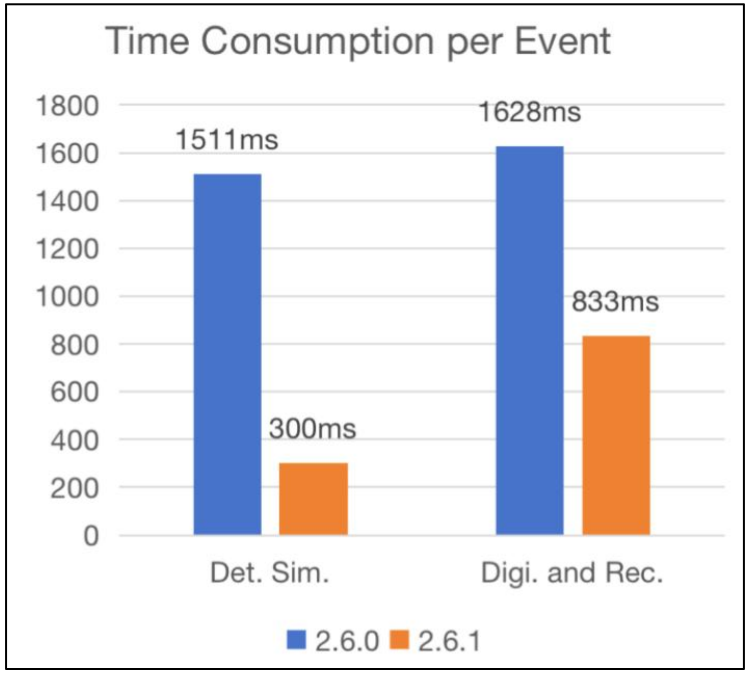
- Various applications in OSCAR (being) developed using ML techniques. Integrating trained model, with the data processing chain properly is vital
- ONNX Runtime is provided for ML model inference
  - Convert ML models to common middleware representation and embedded into OSCAR offline data processing
  - Deep-learning framework agnostic
  - Support inference on both GPU and CPU
  - Now being used in DTOF CNN-based PID algorithm
  - Being applied to GNN-based noise filtering

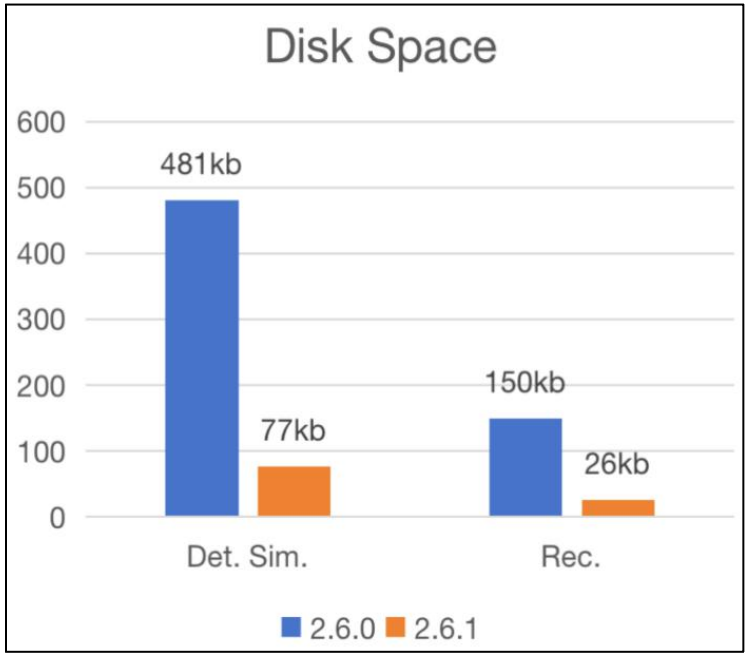
```
Ort::MemoryInfo info("Cpu", OrtDeviceAllocator, 0, OrtMemTypeDefault);

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;
    }
}
```

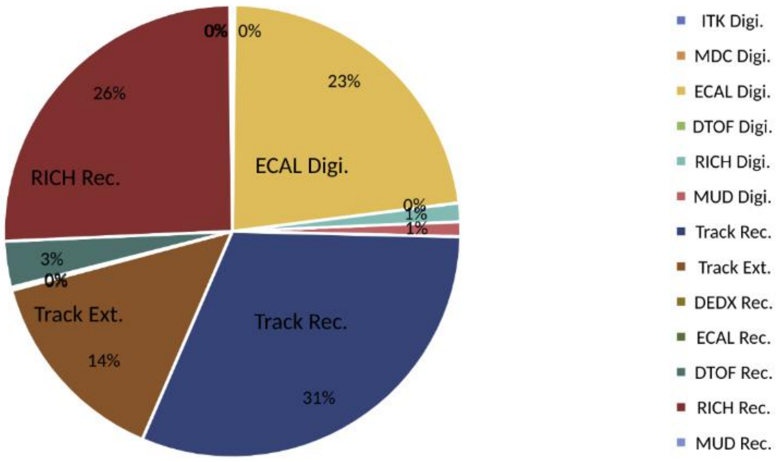
```
bool OrtInferenceAlg::initialize() {

   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);
```

Software Optimization

- Towards massive data production for the TDR, OSCAR is greatly optimized, in terms of the execution speed and output data volume
  - Lots of optimization of simulation, digitization and reconstruction algorithms (further optimization is still being performed)
  - Optimization of event data model (removal of redundant information, using more efficient data types, etc.)
  - Performance is comparable to BESIII now



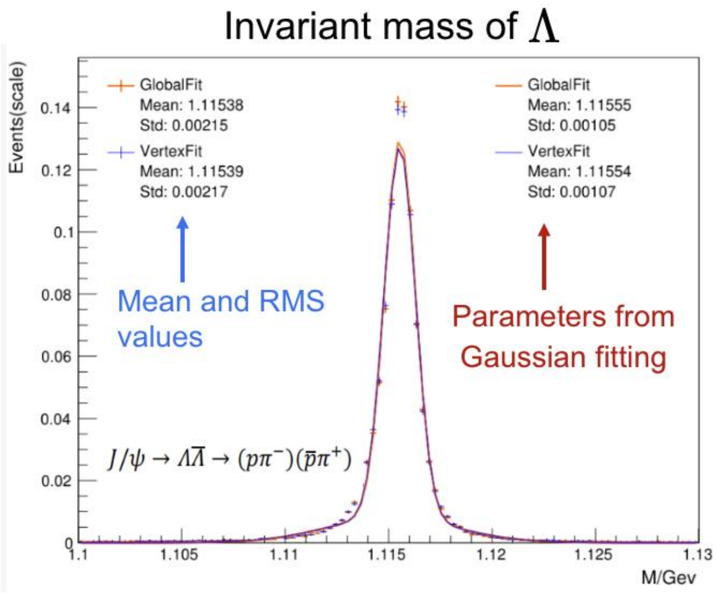


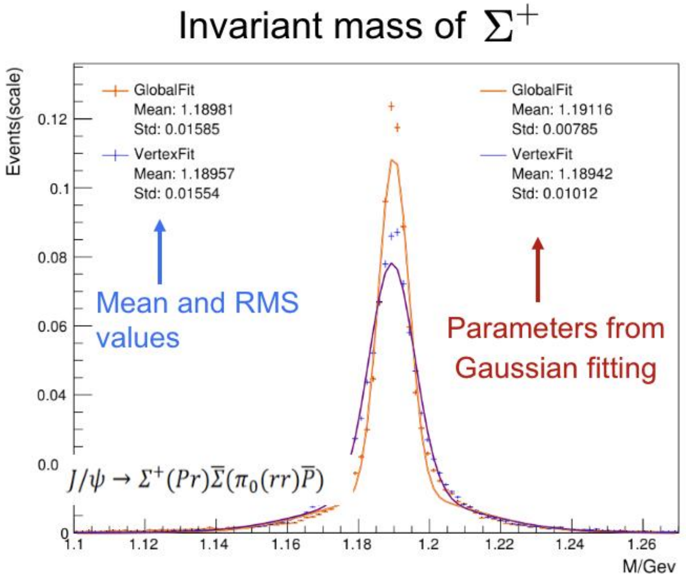


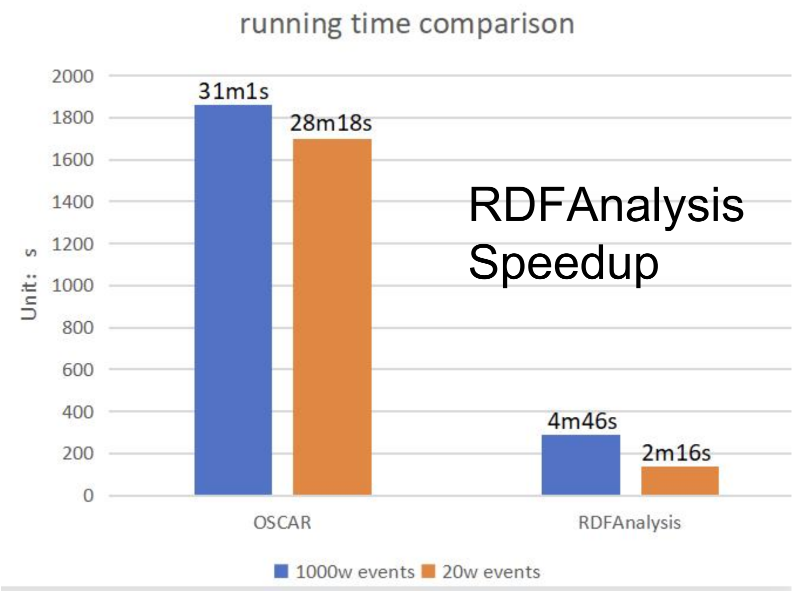
J/ψ decay event

Data Analysis

- GlobalFit package designed for STCF based on the tree fitting algorithm of Belle II, showing better performance than VertexFit imported from BESIII







- RDataFrame-based analysis framework keeps being enriched and tested
  - Physical analysis results are consistent using $J/\Psi -> \Lambda\bar{\Lambda}$
  - Running speed significantly improved using parallel computing technique

Other Development Activities

- New features and updates of OSCAR being developed:
  - Analysis Event Data Model is being developed
    - Skimmed data on full reconstruction EDM (like BESIII DST)
    - Greatly simplify analysis, and reduce disk storage burden
    - Initial design was done
  - Software deployment based on Spack
    - As a multi-platform package manager that builds and installs multiple versions and configurations of software, allowing flexible management of various external libraries
  - AlmaLinux9 support
    - Now OSCAR runs in CentOS7 simularity containers, updating to el9 is being performed
  - Fast calorimeter simulation based on GAN
    - Can greatly reduce computation resource comsuption for MC production

Summary

- We introduced the basic design and functionalities of STCF core software
- Based on the core components, STCF full simulation and reconstruction chain has been established
- A dedicated OSCAR tutorial is performed during 2025 Feburary
  - Including how to simulate and reconstruct data, how to perform data analysis, and how to develop new algorithms in OSCAR
  - A lot of physics analyzers are now involved, using OSCAR to perform physics studies
  - OSCAR has been improved greatly since then (thanks a lot to all the feedbacks)
  - TDR preparation has begun based on OSCAR
- We have been continuously improving the core software
  - To improve the software and physics performance