Core Software of STCF

Teng LI on behalf of the STCF core software development team

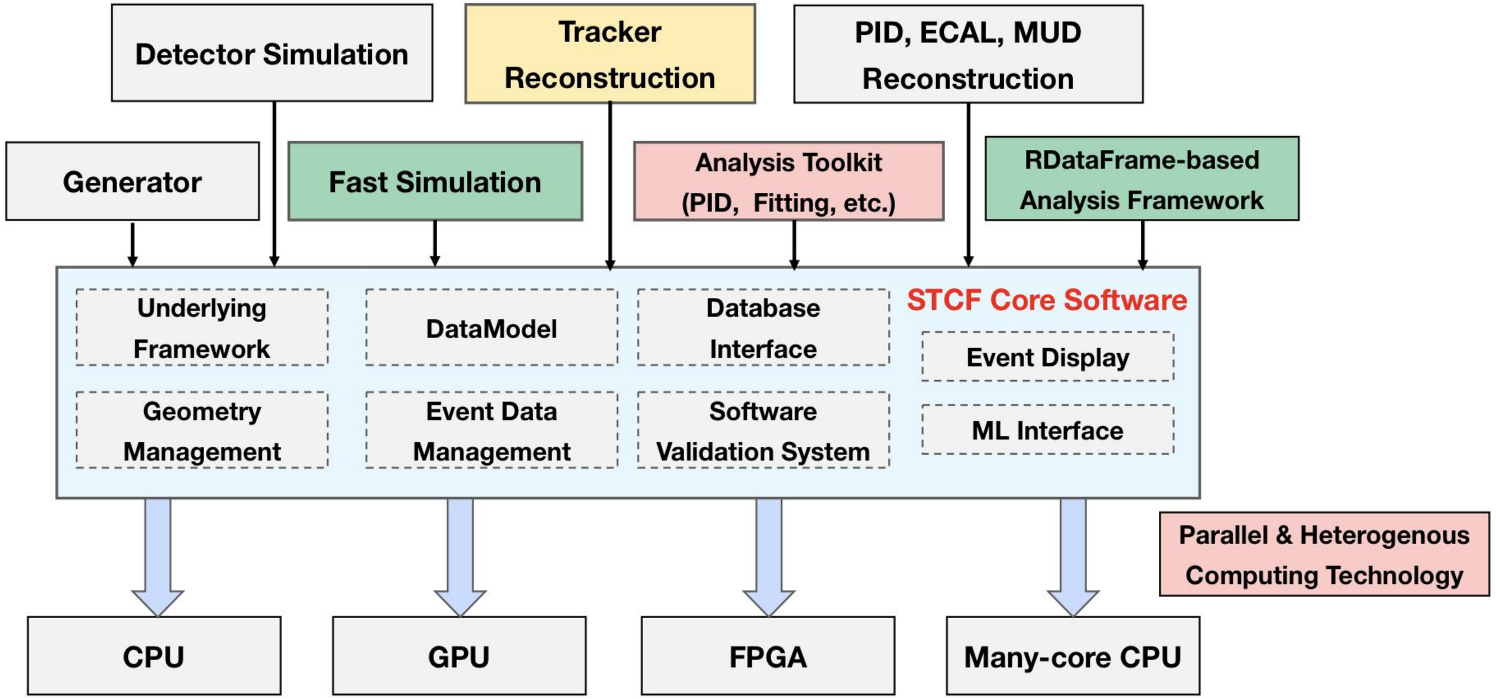
Shandong University

2024-7-9

STCF 2024 Workshop, LZU

Introduction

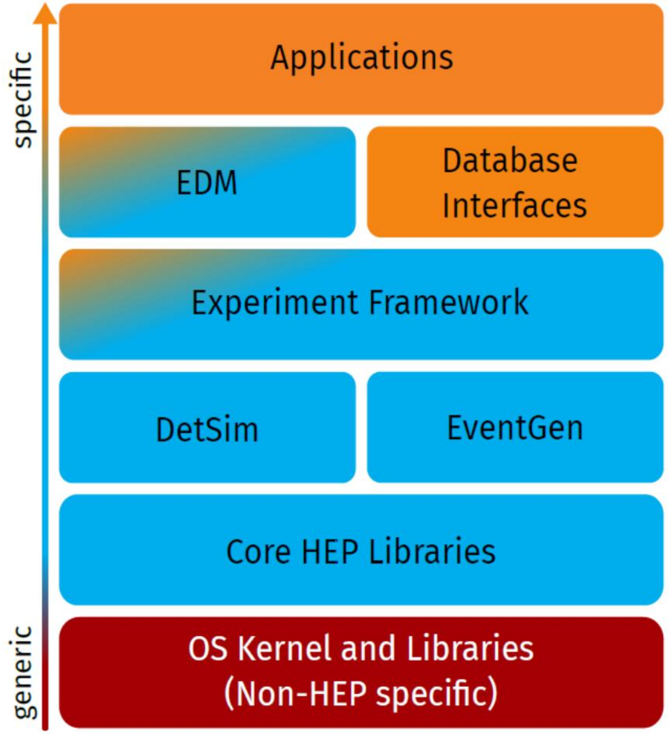
- The task of the STCF core software
  - To fulfill official offline data processing tasks, i.e. detector simulation, digitization, calibration and reconstruction
  - Provide a common platform for users to develop and embed analysis code
- The scope of the STCF core software



- 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

Introduction

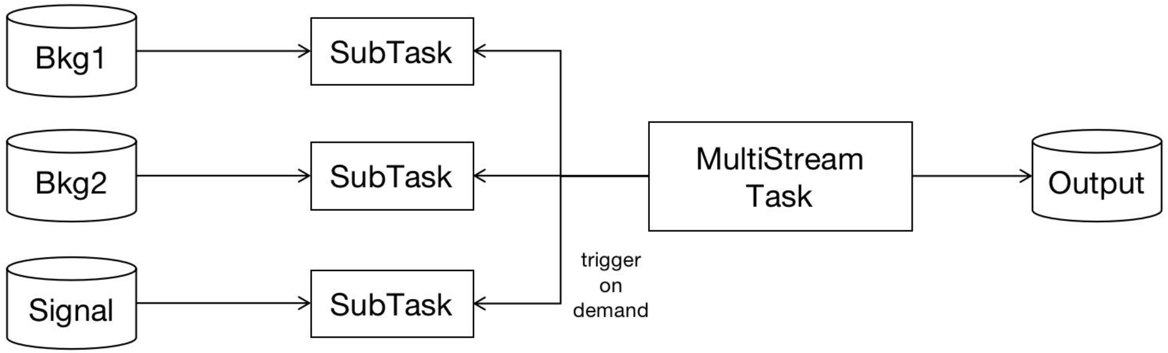
- Main R&D challenges and innovations for STCF core software
  - The huge data volume (~100 times of BESIII) requires much more advanced performance
    - Relying on pure CPU resource to process **exabytes** of data is hardly realistic under previous cost-model
    - Parallel computing, and heterogeneous resources, like GPUs, or FPGAs need to be supported to overcome the challenges.
    - The core software needs to provide ready-to-use development and run time environment for heterogeneous processers.
    - Support of flexible ML inference is nesessary
  - 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.

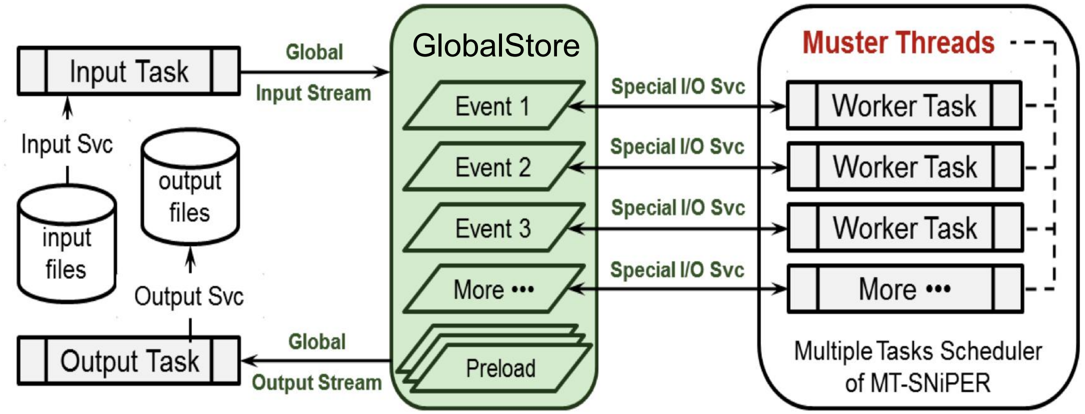


Key4hep
Thomas Madlener,
Epiphany Conference 2021

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.
- OSCAR adopts SNiPER as the underlying framework
  - Developed since 2012, maintained by **10+ developers from IHEP, SDU, SYSU etc.**
  - Adopted by JUNO (neutrino), LHAASO (cosmic ray), nEXO (neutrinoless double beta decay) and HERD (dark matter)
- Advantages of SNiPER
  - Lightweighted, efficient and highly extendable. Flexible data processing chain.
  - Efficient multithreading. C++/Python hybrid programing.





Underlying Framework: SNiPER

- Advantages of SNiPER
  - Lightweighted, efficient and highly extendable. Flexible data processing chain.
  - Efficient multithreading. C++/Python hybrid programing.

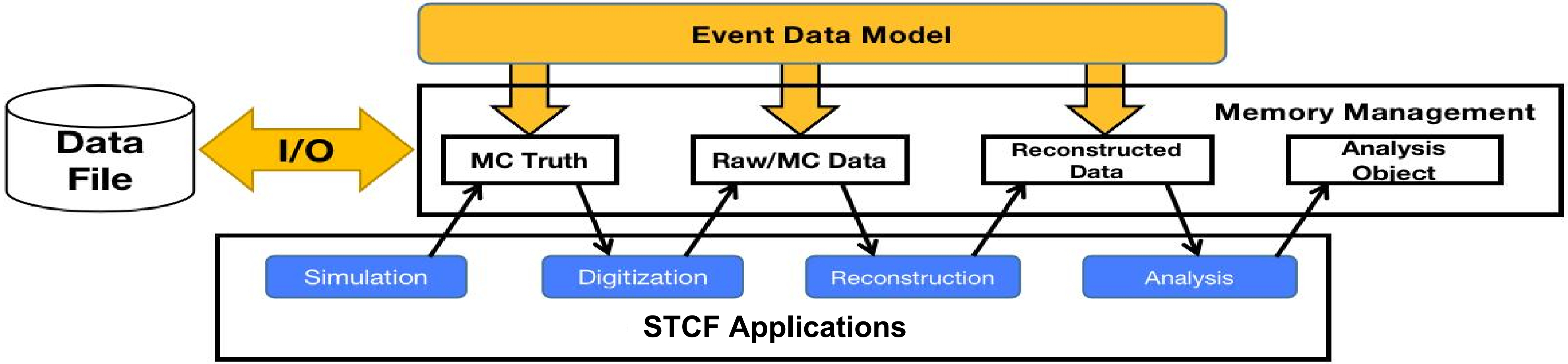
```python
#!/usr/bin/env python
#-*- coding: utf-8 -*-
# Author: Teng LI <tengli@sdu.edu.cn>

from OSCAR import *
from GeometrySvc import GeometryModule
from RandomSvc import RandomModule

app = OSCARApplication()

# Random engine
app.registerModule(RandomModule())

# Geometry
app.registerModule(GeometryModule())

# Detector simulation
app.registerModule(DetectorSimulation())

# Data management
app.registerModule(DataManagement())

app.run()
```

```
-bash-4.2$ python simulation.py -h
**************************************************
Welcome to SNiPER 2.1.0
Running @ stcf02.hep.ustc.edu.cn on Mon Mar 25 15:27:38 2024
**************************************************
usage: simulation.py [-h] [--loglevel {Test,Debug,Info,Warn,Error,Fatal}] [--dryrun] [--evtmax EVTMAX] [--use
                     [--no-profiling] [--profiling-detail] [--no-profiling-detail] [--seed SEED] [--seed-stat
                     [--seed-status-vector SEED_STATUS_VECTOR [SEED_STATUS_VECTOR ...]] [--geometry-compact-
                     [--sub-detector-list {ITK,MDC,BTOF,DTOF,RICH,ECAL,MUD,NONE} [{ITK,MDC,BTOF,DTOF,RICH,ECA
                     [--generator-random-seed GENERATOR_RANDOM_SEED] [--g4-run-mac G4_RUN_MAC] [--g4-commands
                     [--particle-translation PARTICLE_TRANSLATION [PARTICLE_TRANSLATION ...]] [--enable-qgsp-
                     [--enable-itk-pai] [--disable-itk-pai] [--enable-optical-sim] [--disable-optical-sim] [-
                     [--enable-itkm-geo-svc] [--disable-itkm-geo-svc] [--enable-mdc-qsim] [--disable-mdc-qsi
                     [--enable-mdc-save-drift-electron] [--disable-mdc-save-drift-electron] [--mdc-enable-sa
                     [--mdc-wire-pos-file MDC_WIRE_POS_FILE] [--mdc-wire-tension-file MDC_WIRE_TENSION_FILE]
                     [--mdc-neighbor-range MDC_NEIGHBOR_RANGE] [--mdc-detector-name MDC_DETECTOR_NAME] [--ena
                     [--disable-ecal-point] [--mud-avalanche-width MUD_AVALANCHE_WIDTH] [--enable-geometry-ou
                     [--gdml-file-name GDML_FILE_NAME] [--enable-tgeo-output] [--disable-tgeo-output] [--enab
                     [--input INPUT [INPUT ...]] [--output OUTPUT] [--output-colls OUTPUT_COLLS [OUTPUT_COLLS
                     [--transfer-colls-exclude TRANSFER_COLLS_EXCLUDE [TRANSFER_COLLS_EXCLUDE ...]] [--transf
                     {gun,babayaga,bbbrem,diag36,kkmc,phokhara,evtgen} ...

positional arguments:
  {gun,babayaga,bbbrem,diag36,kkmc,phokhara,evtgen}
                        Detector Simulation Mode (Generator)
    gun                 Geant4 particle gun (gps)
    babayaga            MC generator for BHABHA precess
    bbbrem              Bremsstrahlung beam background
    diag36              MC generator for photon-photon scattering events with 4 leptons in the final state
    kkmc                MC event generator for e+e- -> ff_bar+n_gamma (standalone KKMC)
    phokhara            MC event generator for e+e- -> hadrons + gamma from ISR
    evtgen              MC event generator for heavy flavour particles decays (KKMC + EvtGen)
```

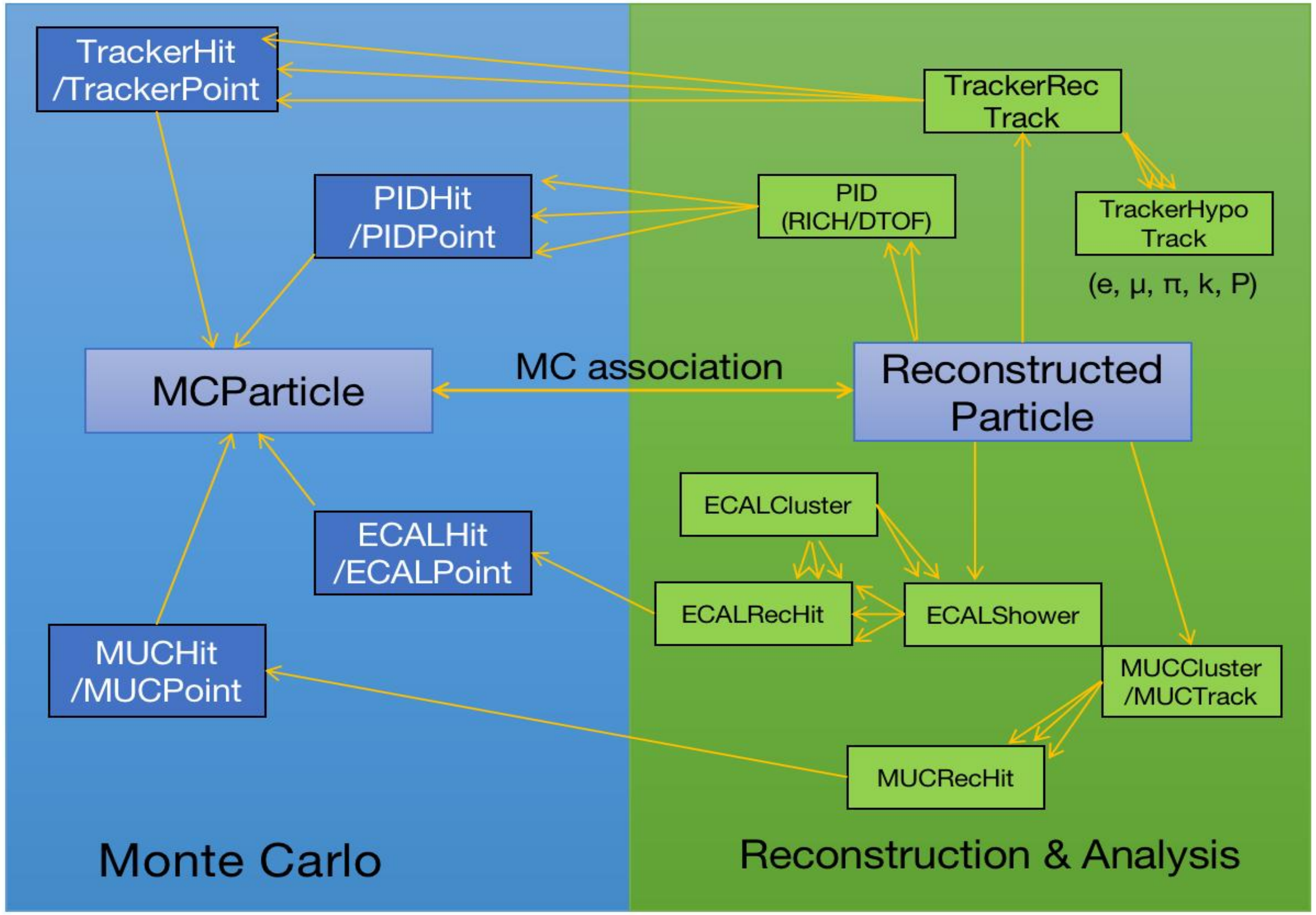
OSCAR Python UI

Event Data Management: **Requirements**

- 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 long time running of STCF.**
  - **Guarantee thread-safety, and provide high performance for MT applications**



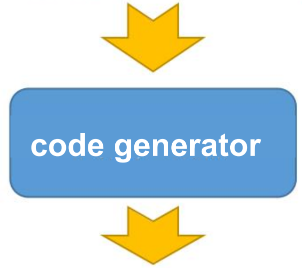
Event Data Model and of OSCAR



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.
```



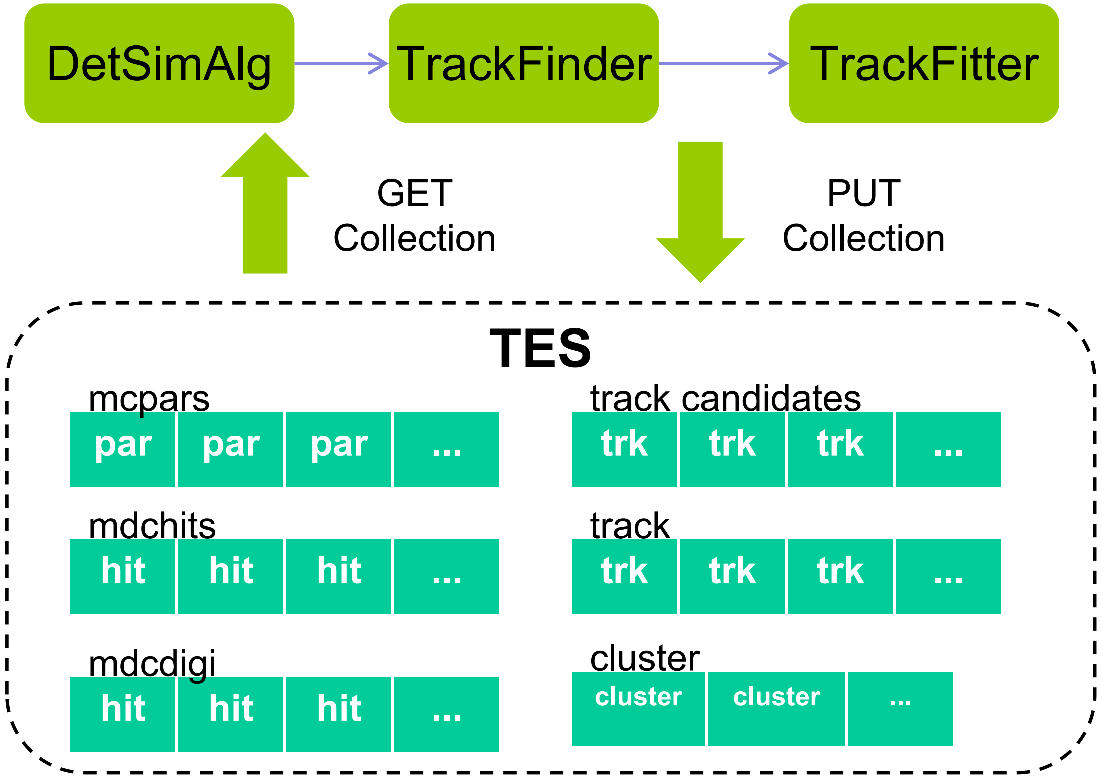
C++

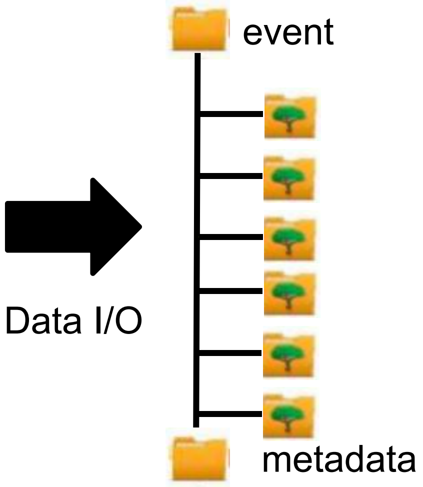
MCParticle.h	MCParticle.cpp
MutableMCParticle.h	MutableMCParticle.cpp
MCParticleObj.h	MCParticleObj.cpp
MCParticleData.h	MCParticleData.cpp
MCParticleCollection.h	MCParticleCollection.cpp
MCParticleCollectionData.h	MCParticleCollectionData.cpp

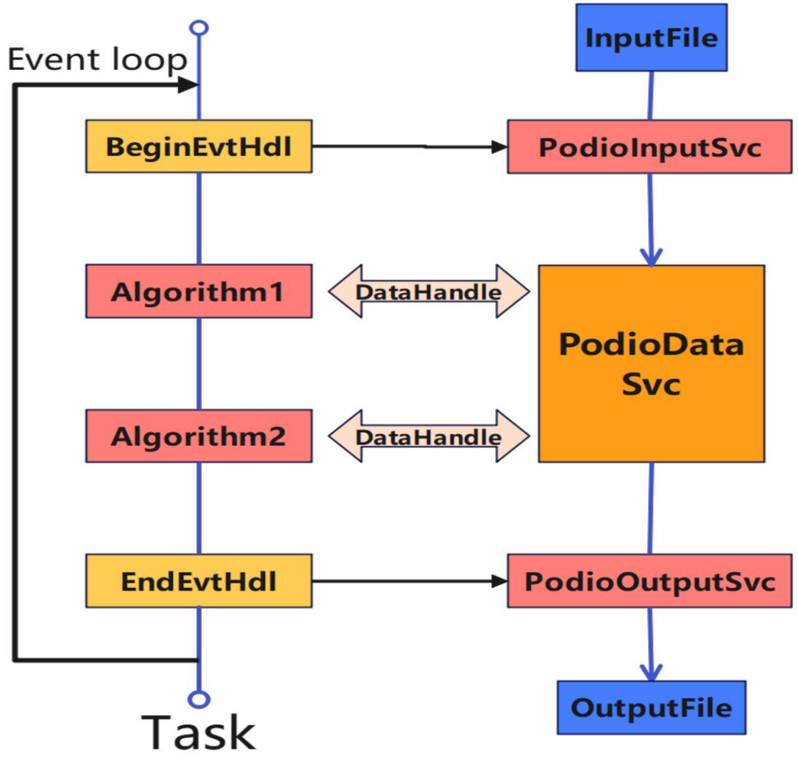
Based on YAML definition, generate EDM C++ code accordingly

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
  - User Algorithms access event data via collections





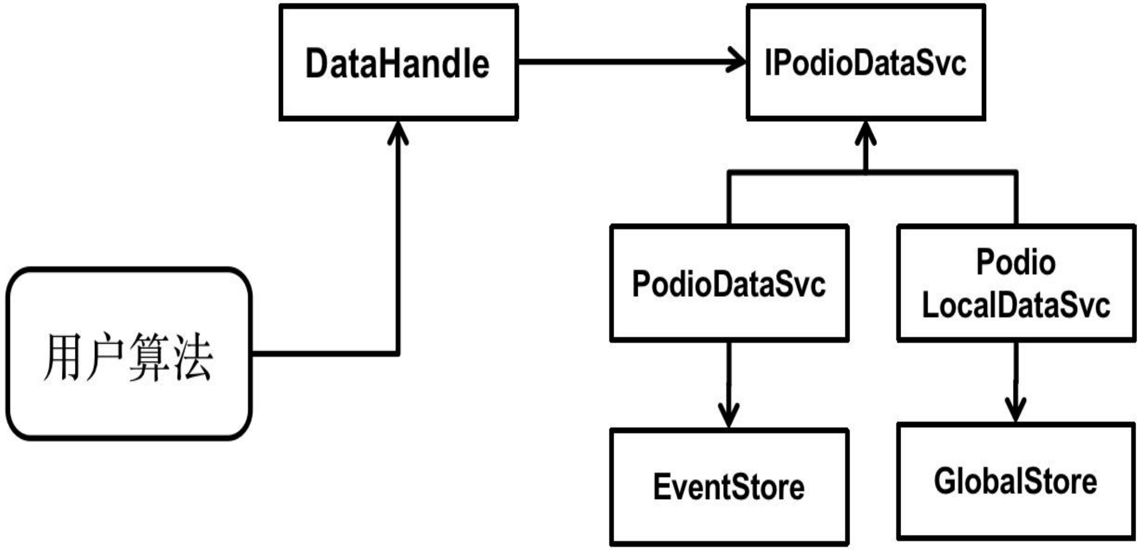


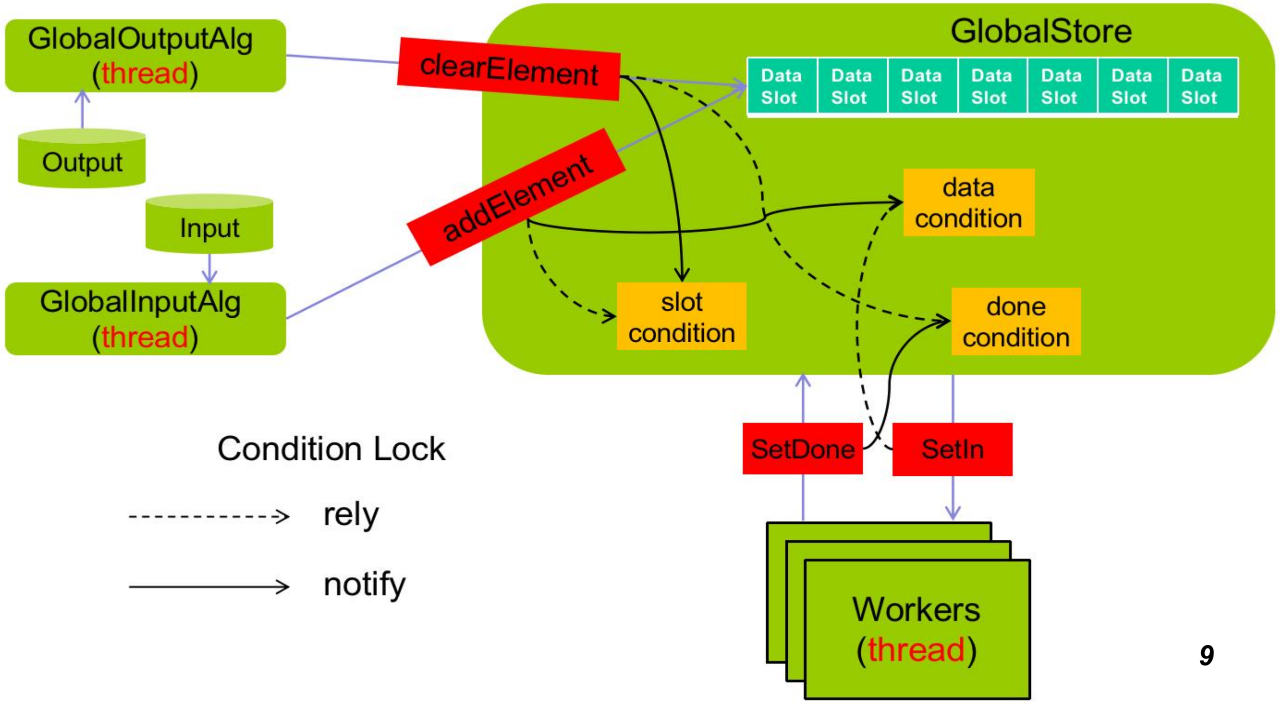
Implementation of TES and data I/O

- PodioDataSvc
- PodioInputSvc
- PodioOutputSvc

Parallelized Event Data Management

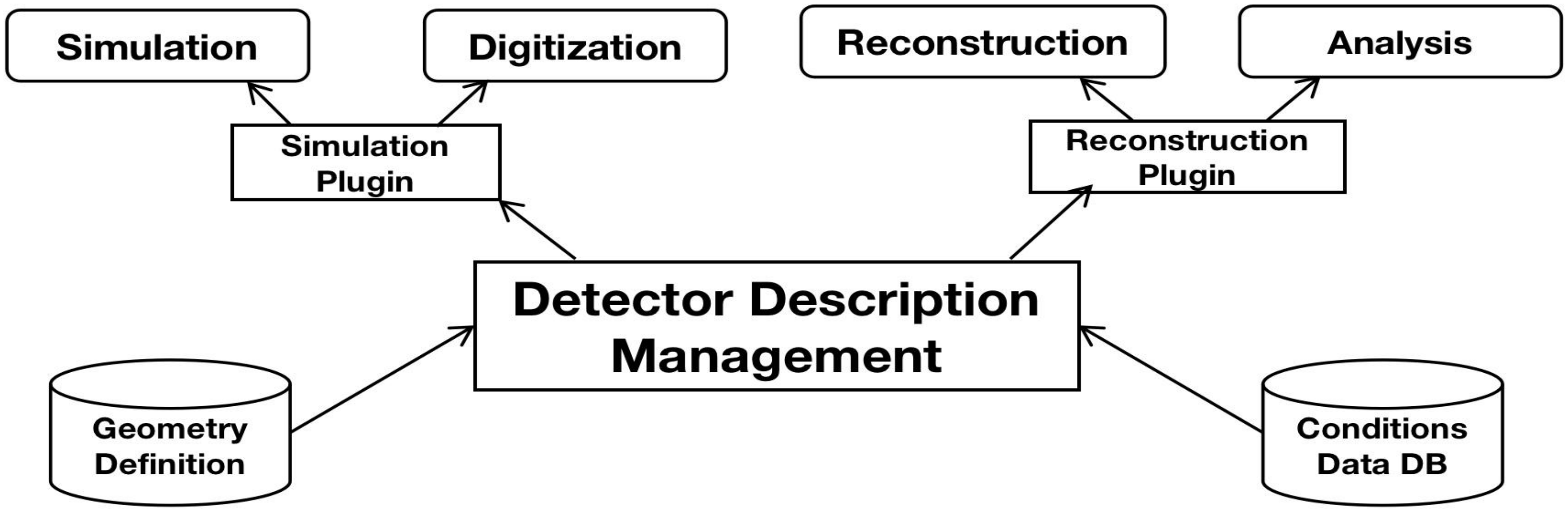
- To enable parallelized data processing, a GlobalStore is developed based on podio
  - Re-implement podio::EventStore to cache multiple events (each within one data slot)
  - Use several condition lock to enable safety exchanging data between threads
  - I/O services are binded to dedicated I/O threads, to ensure performance and flexible post- or pre-processing
- Users could switch serial/parallel by just changing job configuration





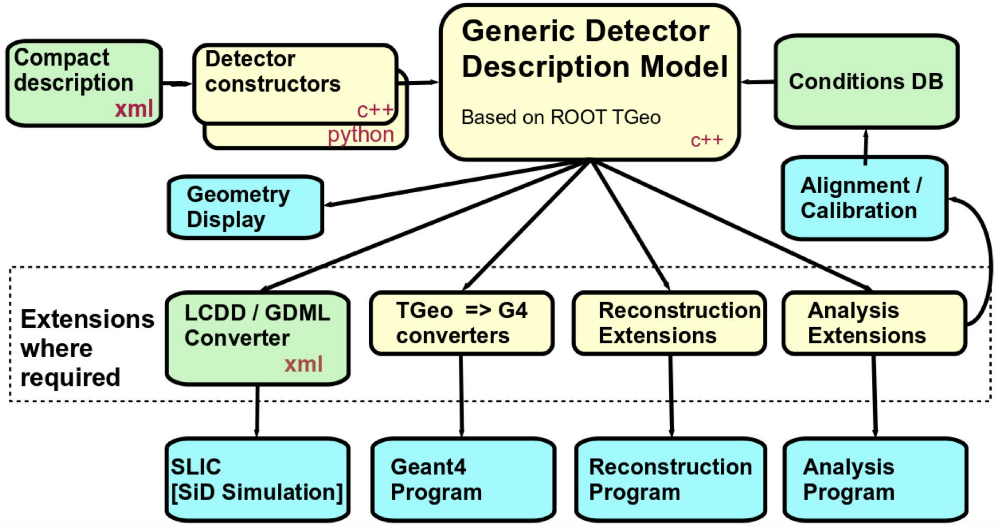
Detector Description Management: **Requirements**

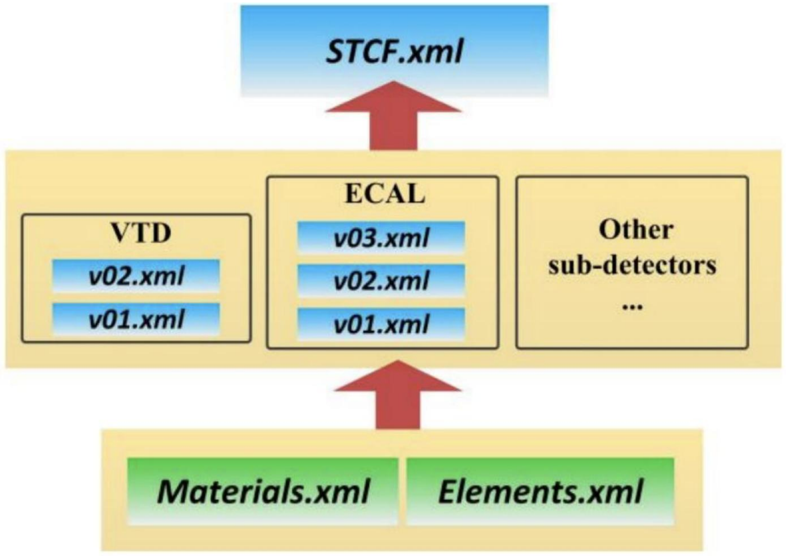
- A powerful detector description management system is necessary across the full offline data processing workflow
  - Provide simple method for **geometry description definition**
  - Provide **consistent detector description** for all applications
  - Provide **geometry conversion** for different applications,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

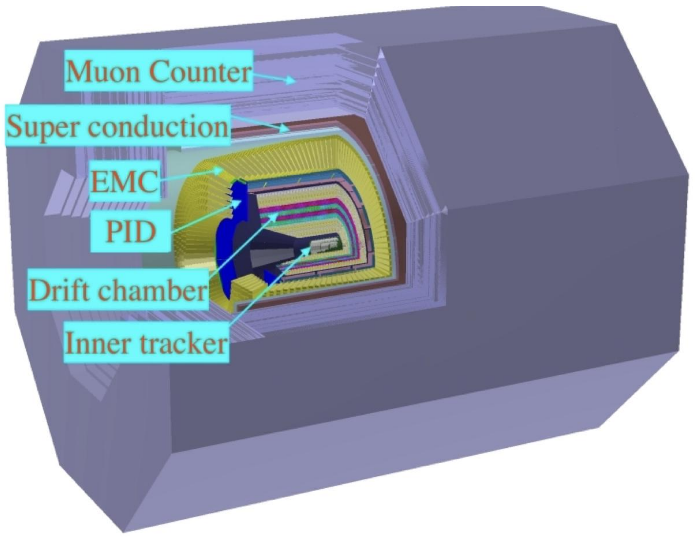


Geometry Management System

- 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

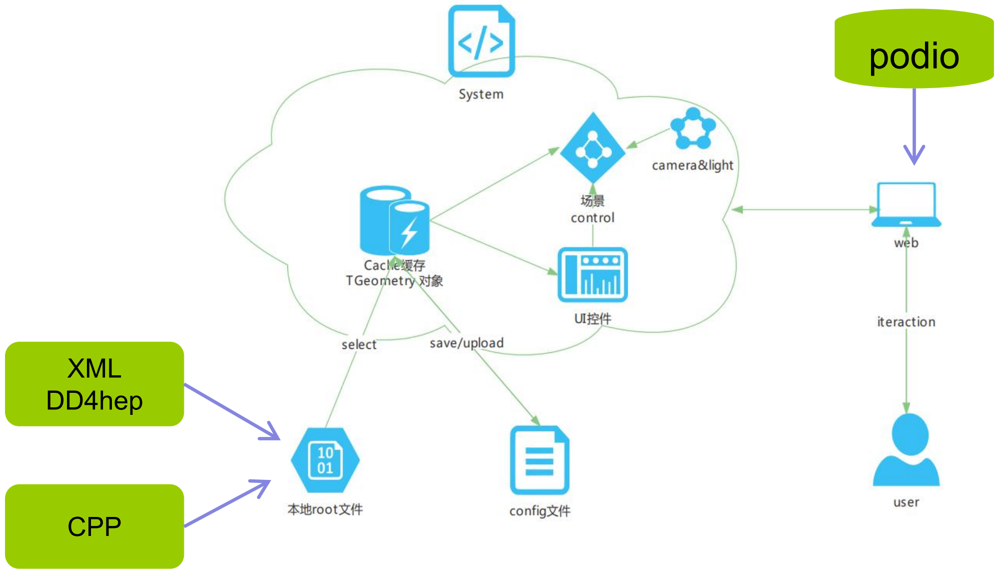


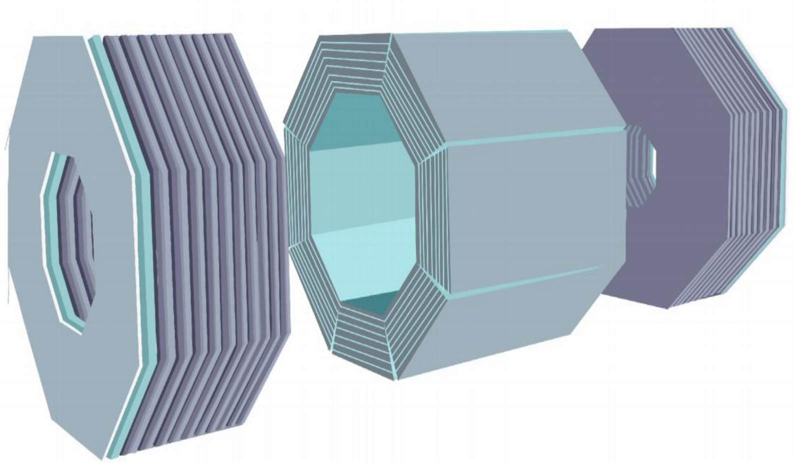


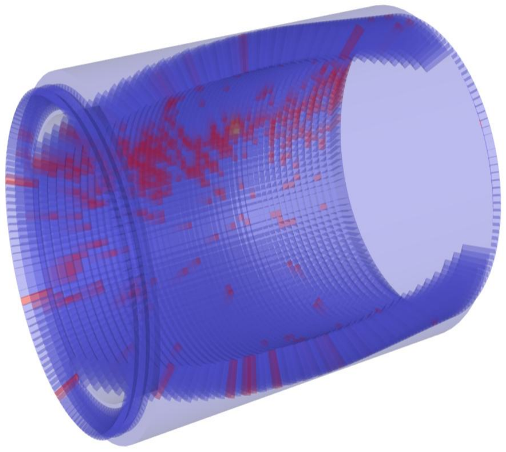


Detector and Event Display

- 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
  - Using the Vue.js HTML5 development framework to implement the Web interface
  - Reducing 3D motion lag by the multi-threading capabilities of Web Worker framework
  - Geometry information from detector description from DD4hep (XML)， and event data read from podio

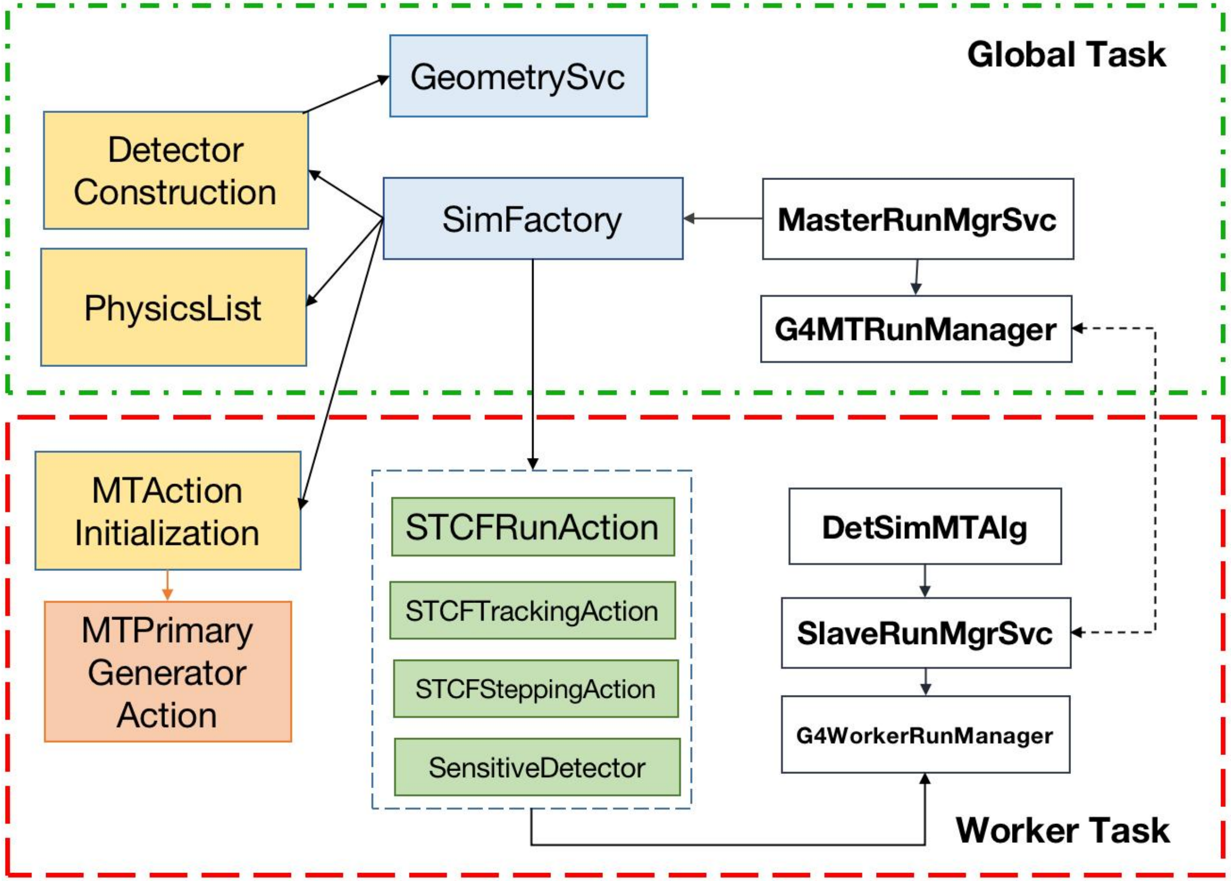


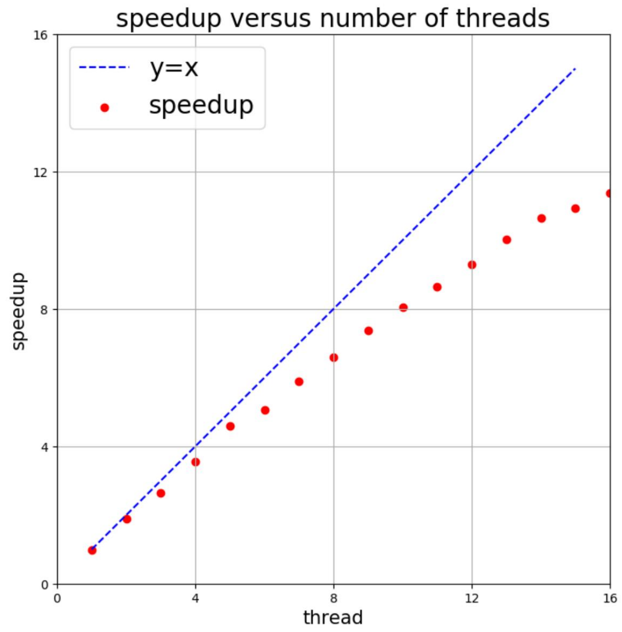


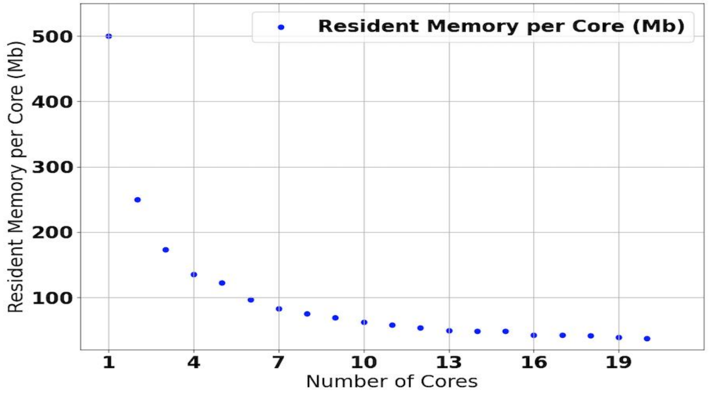


Parallelized Detector Simulation

- Based on the MT-SNiPER and parallelized DM system, parallelized detector simulation applications are developed
  - Basic performance tests show promising scalability

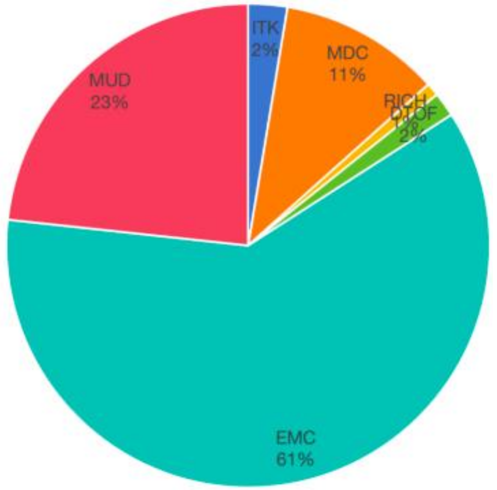






Fast ECAL Simulation based on GAN

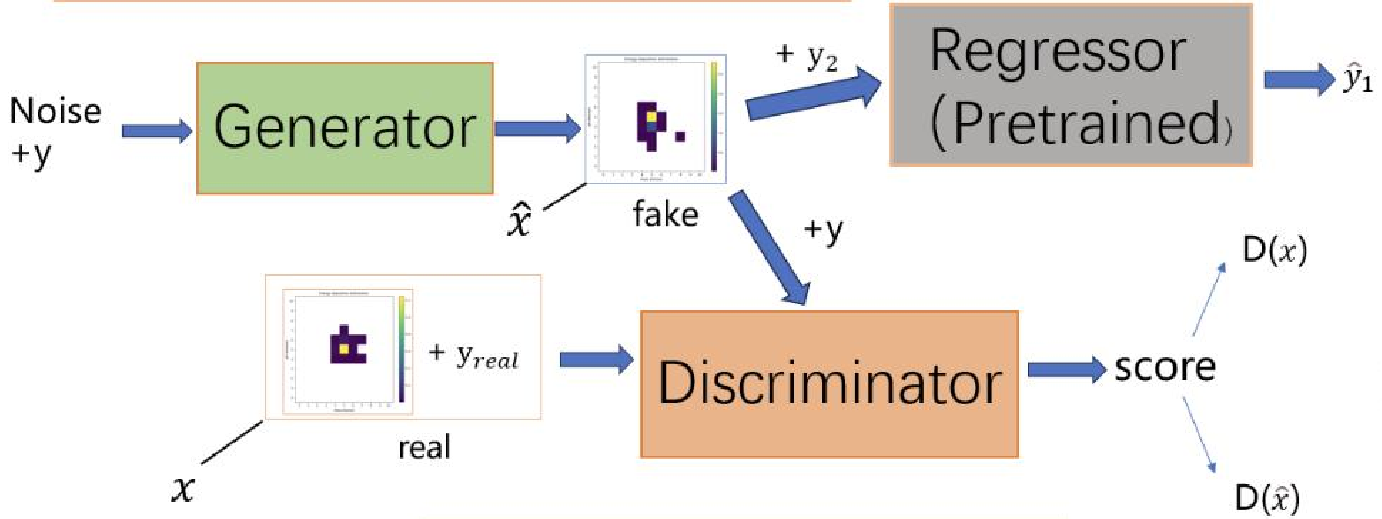
- A ECAL fast simulation software based on DCGAN is being developed and optimized
  - Integrate fast simulation and Geant4-based full simulation
  - Expect to speed up the ECAL simulation by 1-2 orders of magnitude



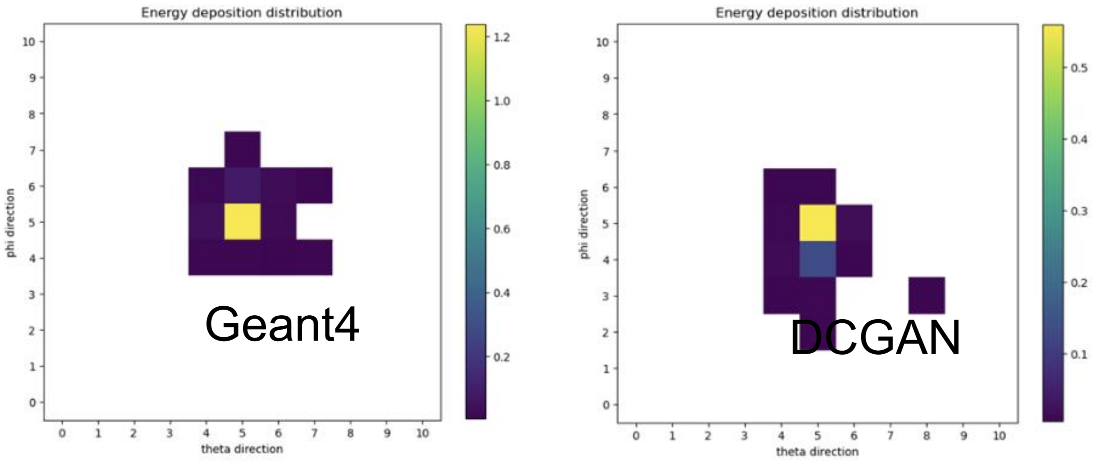
Geant4模拟耗时对比

full simulation:
~1s/per rhopi

$$\max_{D} \mathbb{E}_{x\sim p(data)}\log(D(x)) + \mathbb{E}_{\hat{x}\sim p(fake)}\log(1\text{-}D(\hat{x}))$$



$$\min_{G} \mathbb{E}_{\hat{x}\sim p(fake)}\log(1\text{-}D(\hat{x})) + \|y_1 - \widehat{y_1}\|_1$$



Machine Learning Model Integration

- ONNX Runtime has been integrated with OSCAR to support runtime inference
  - Lot's of applications in OSCAR are based on ML models, such as fast simulation, reconstruction, PID etc.
  - As an easy and unified way to integrate different models in OSCAR and run inference easily
  - Convert from other models to ONNX, such as Tensorflow, PyTorch etc.
  - Potentially to accelerate inference of larger model on different hardware platform (CPU/GPU)

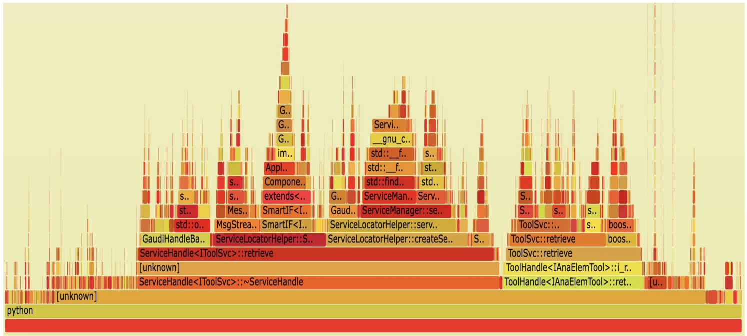
```
CNNModel::CNNModel(const std::string& modelPath): env(ORT_LOGGING_LEVEL_WARNING, "CNNONNX"),
  sessionOptions.SetIntraOpNumThreads(1);
  sessionOptions.SetGraphOptimizationLevel(GraphOptimizationLevel::ORT_ENABLE_EXTENDED);
  session = Ort::Session(env, modelPath.c_str(), sessionOptions);
}
```

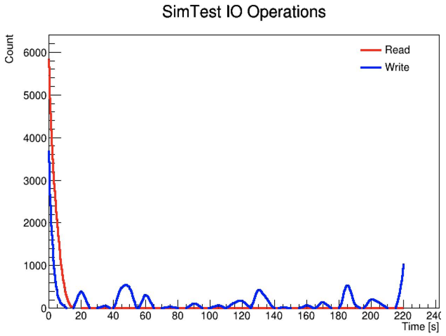
```
bool CNNModel::predict(std::vector<float>& image, std::vector<float>& feature, s
    size_t num_input_nodes = session.GetInputCount();
    size_t num_output_nodes = session.GetOutputCount();

    // 获取输入节点名称和形状
    std::vector<std::string> input_node_names(num_input_nodes);
    for (int i = 0; i < num_input_nodes; i++) {
    auto input_name = session.GetInputNameAllocated(i, allocator);
    input_node_names[i] = input_name.get();
    //auto input_shape = session.GetInputTypeInfo(i).GetTensorTypeAndShapeInfo()
    }
    const char* input_node_names_cstr[num_input_nodes];
    for (int i = 0; i < num_input_nodes; i++) {
    input_node_names_cstr[i] = input_node_names[i].c_str();
    }

    // 获取输出节点名称和形状
    std::vector<const char*> output_node_names(num_output_nodes);
    auto output_name = session.GetOutputNameAllocated(0, allocator);
    output_node_names[0] = output_name.get();
    //auto output_shape = session.GetOutputTypeInfo(0).GetTensorTypeAndShapeInfo

    // 创建输入张量
    std::vector<int64_t> image_shape = {1, 200, 217, 3};
    auto memory_info = Ort::MemoryInfo::CreateCpu(OrtArenaAllocator, OrtMemTypeD
    Ort::Value image_tensor = Ort::Value::CreateTensor<float>(memory_info, image
```

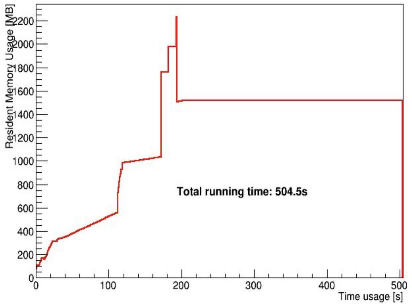
DTOF PID CNN model

Automated Software Validation

- Software validation system is developed, to support building software validation on different levels
  - Unit test, integrated test, software performance profiling and physics result validation
- Integrated with Gitlab Action system for automated validation
  - Used CTest to integrate validation cases integrated with CI system
  - Trigger validation jobs on different levels on schedule/commits








```
-bash-4.2$ ctest
Test project /home/tli/2.5.0_test/offline/build
      Start  1: test_random_svc.py
 1/16 Test  #1: test_random_svc.py ..............   Passed    1.04 sec
      Start  2: test_demoSim.py
 2/16 Test  #2: test_demoSim.py .................   Passed    2.89 sec
      Start  3: test_read_update.sh
 3/16 Test  #3: test_read_update.sh .............   Passed    8.52 sec
      Start  4: test_simple_rw.sh
```

Summary

- We introduced the basic design and functionalities of STCF core software
  - Developed partially based on Key4hep
  - Many components are extended specificlly for STCF, but are also re-usable by other experiments
- Based on the core components, many STCF applications are (being) developed
  - Detector simulation, reconstruction algorithms, event display, analysis toolkit including particle ID, Vertex/KineticFit, RDataframe based analysis framework etc.
  - On-going physics analysis studies with MC data are in progress
- We have been continuously improving the core software
  - Software and physics performance has been continuously improved
  - Many applications are being developed based on concurrent/heterogeneous computing and machine learning