Status and Plan of STCF Software

Xingtao Huang (SDU)
On behalf of the STCF Software Group

EicC-STCF Joint Meeting

University of Science and Technology of China, Hefei

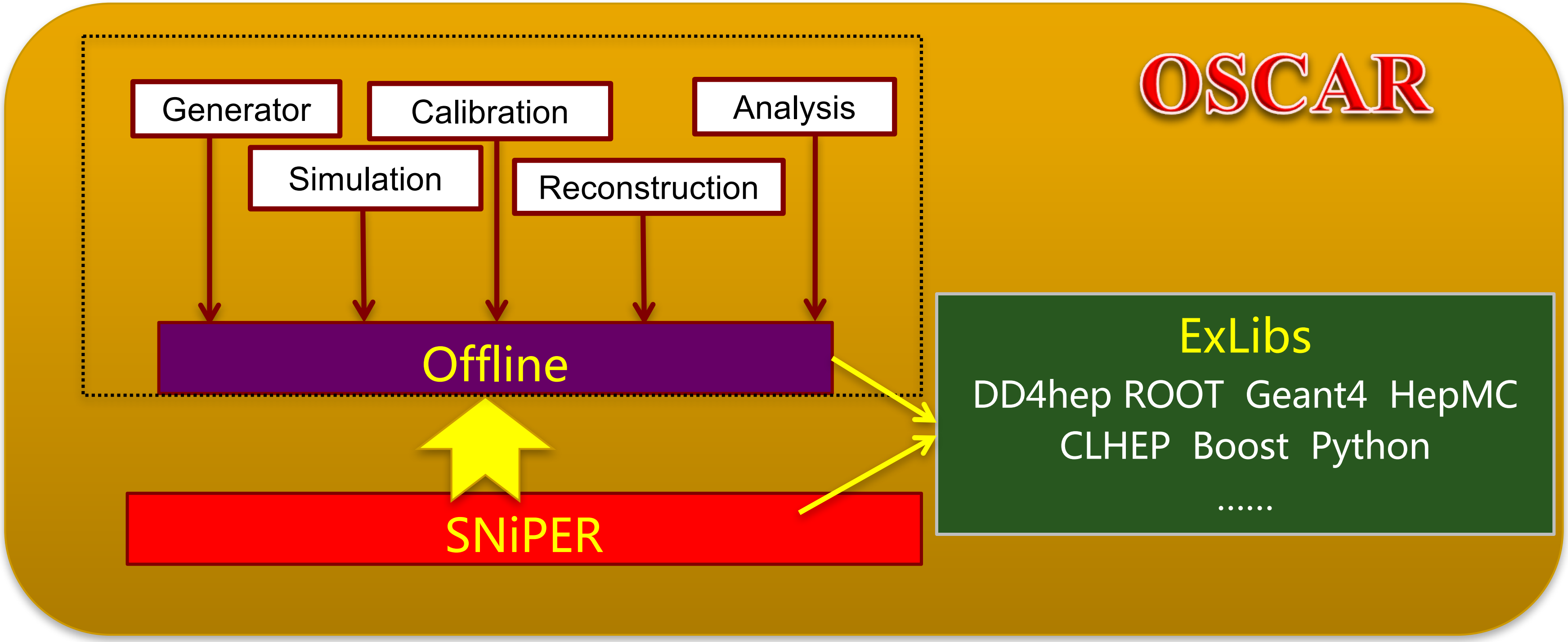
Feb 21 - 22, 2020

Outline

- STCF Offline Software System
- Framework
- Event Data Model
- Generator
- Detector Simulation
- Visualization
- Reconstruction
- Summary and Outlook

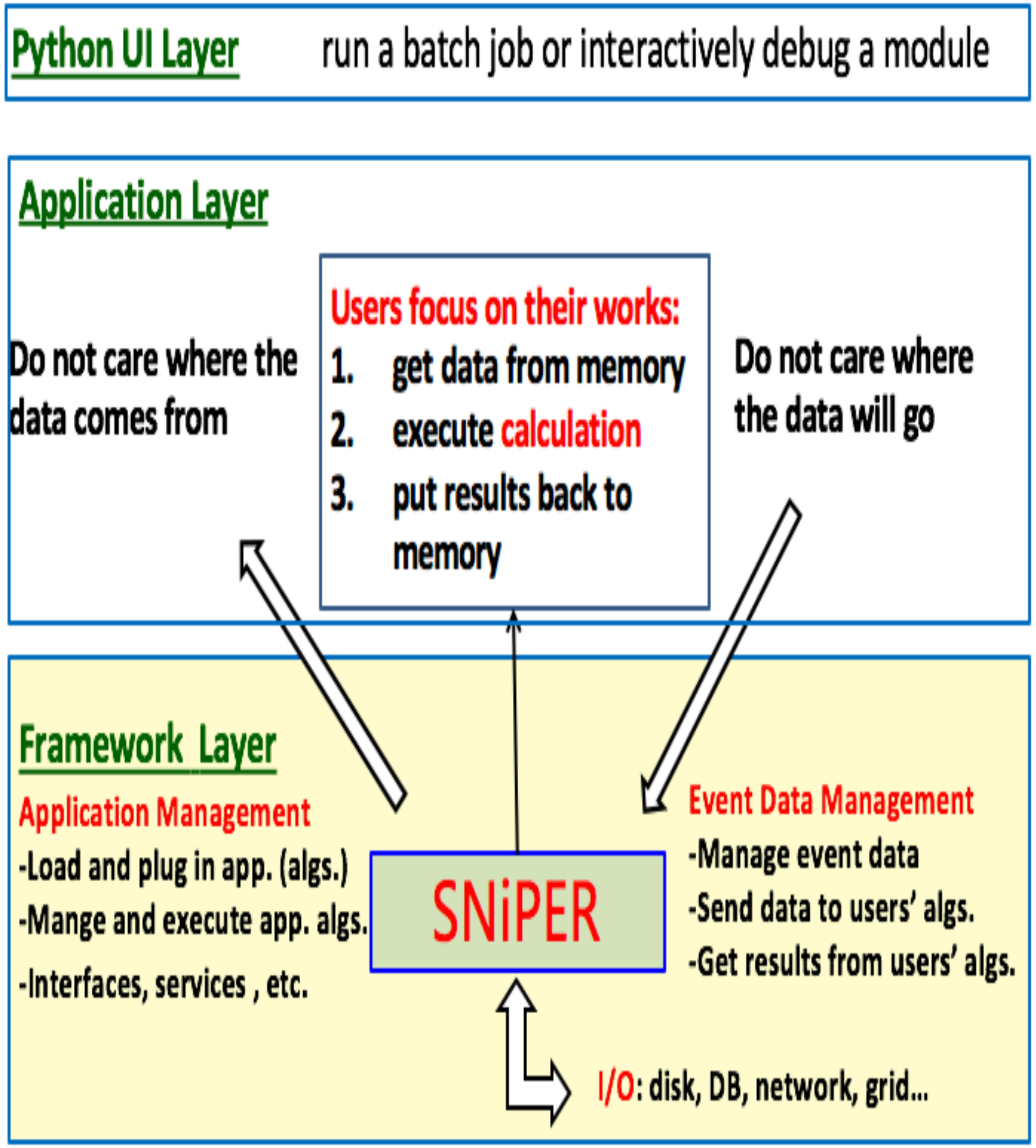
Overview of STCF Software System

OSCAR: Offline Software of Super Tau-Charm Facility



- **External Libs**: Frequently used third-party software and tools.
- **SNiPER Framework**: Providing Data Processing Management, Event data Management, Common Services, User Interface …
- **Offline** : Specific to the STCF Experiment, including extensions to SNiPER, Generator, Simulation, Calibration , Reconstruction and Analysis

Minimum Requirements of Users



Physicists

Modify/Resue algorithms and run jobs
(C++ and Python)

Application Developers

Write new Algorithms and configuration files
(C++ and Python)

Framework Developers

Provide main functions for HEP data processing
(C++, Python, SQL, multi-thread,…)

SNiPER Framework

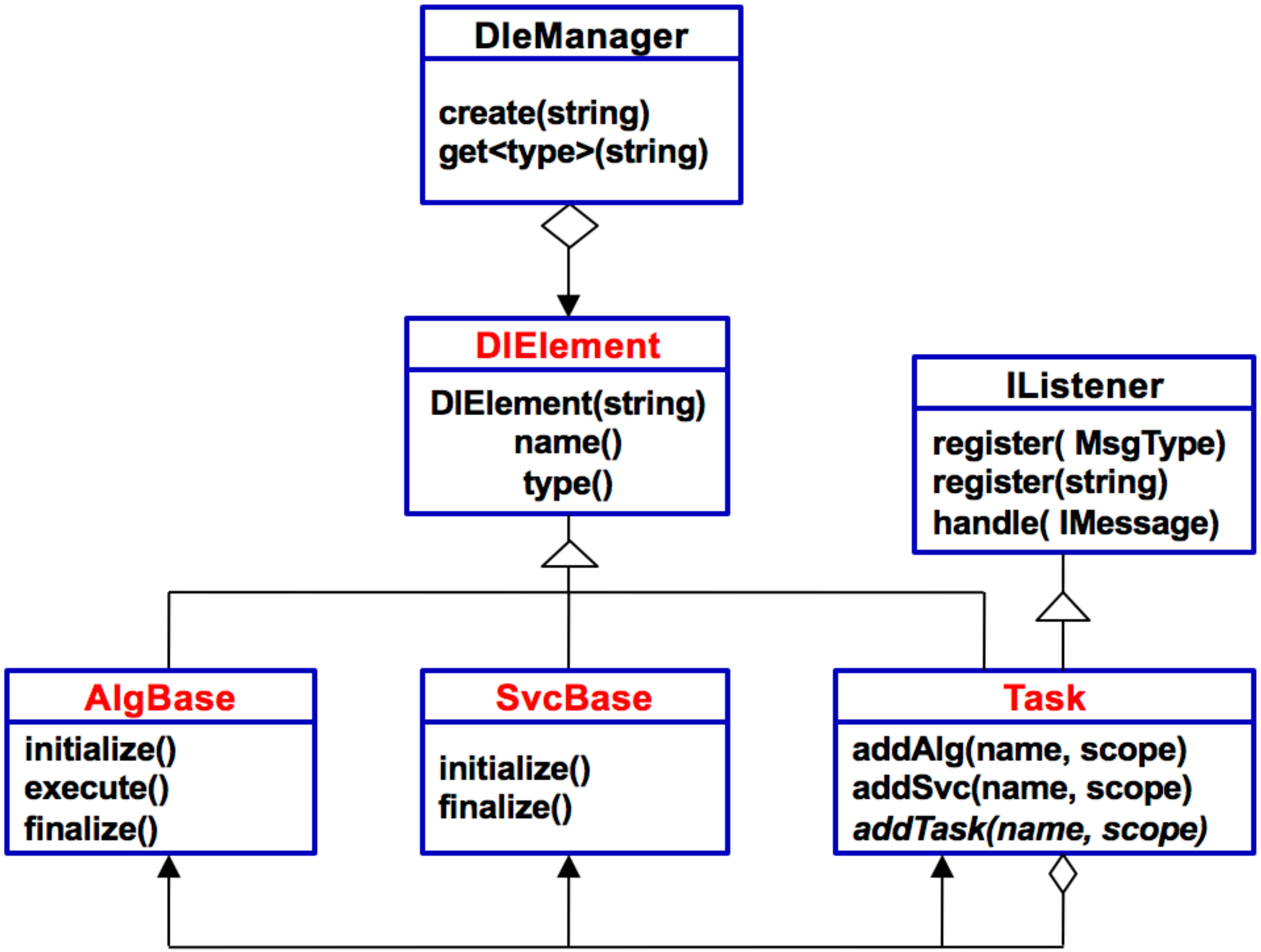
- SNiPER: the "Software for Non-collider Physics ExpeRiment"
  - Developed for JUNO experiment ,also considered for other physics experiments
  - Used by JUNO,LHAASO ,STCF, nEXO
  - Being Investigated by HERD
- The Design Goals
  - Lightweight, less dependences on third-party software/libs
  - Fast and flexible execution
  - Easy to learn and convenient to use
- A Good Team to maintain and optimize
  - SDU and IHEP

Main Features of SNiPER

- Highly modular
- Dynamically loading packages/modules/elements
- Standard interfaces between different modules
- Separation between data and algorithm
- Data Store for event data management
- Flexible event execution
  - Sequential and Jump/nested execution
- Support multithreading
  - Underlying the intel TBB is deployed

Key Components of SNiPER

- User Interfaces (Dynamically Loadable Elements)
  - Algorithm
  - Service
  - Task
- Data Store
- Property
- Logging
- Parallelism



Algorithm

- An unit of code for event execution
  - ⇨ Perform event calculation during event loop
  - ⇨ Users only focus on processing "One Event"
- Framework provides the interface, AlgBase
- User's new algorithm inherits from AlgBase
  - ⇨ Its constructor takes one std::string parameter
  - ⇨ 3 member functions must be implemented
    - bool initialize() : called once per Task (at the beginning of a Task)
    - bool execute() : called once per Event
    - bool finalize() : called once per Task (at the end of Task)
- Then, the new algorithms can be called by Framework

Example: HelloAlg

```
class HelloAlg: public AlgBase {

    public:
        HelloAlg(const std::string& name);
        ~HelloAlg();

        bool initialize();
        bool execute();
        bool finalize();

    private:
        int m_count;
        std::string m_string;

};
```

Service

- Similar with Algorithm, but
  - A piece of code for common use, i.e. GeometrySvc, DatabaseSvc…
  - They are called by algorithms or other services, wherever needed
- Framework provides the interface, SvcBase
- New services inherits from SvcBase
  - Its constructor takes one std::string parameter
  - 2 member functions must be implemented
    - bool initialize() : called once per Task (at the beginning of a Task)
    - bool finalize() : called once per Task (at the end of Task)

Example: HelloSvc

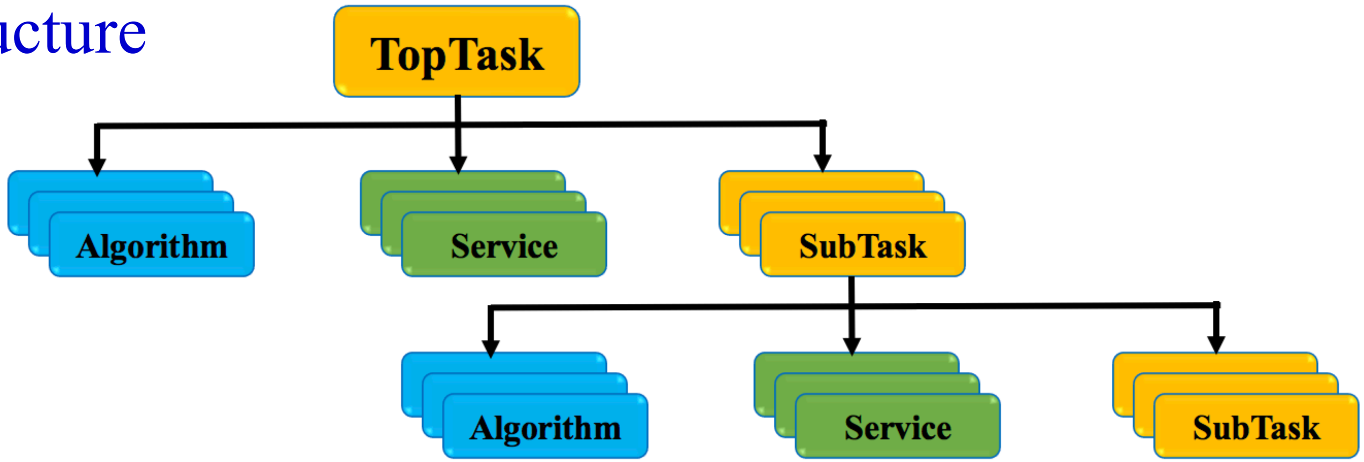
```cpp
class HelloSvc: public SvcBase {

    public:
        HelloSvc(const std::string& name);
        ~HelloSvc();

        bool initialize();
        bool finalize();
        void doSomething();

};
```

Existing Services

- Data Store Management Service
- Detector Geometry Construction Service
- Unified Geometry Provider Service
- Random Number Service
- Database Service
- Root File Input/Output Service
- Root Histogram/N-tuple Service
- ……

Task

- A lightweight application manager
  - ⇨ Consist of algorithms, services and sub-tasks
  - ⇨ Control algorithms' execution
  - ⇨ Has its own data store and I/O system (see next slide)
- One job can have more than one Tasks
- The objects of algorithms or services are organized in a tree structure

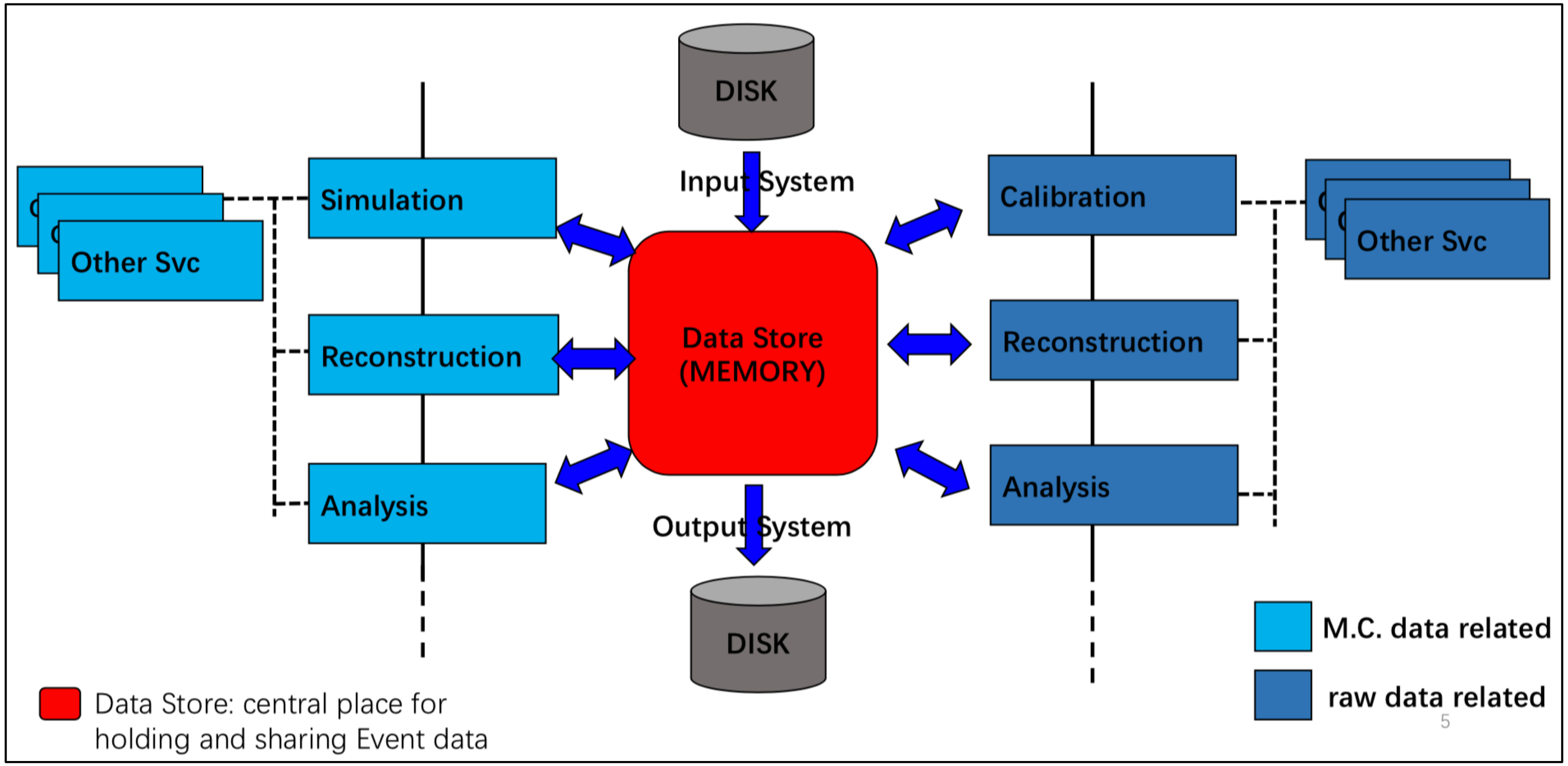


Python Script Example

```python
import Sniper

task = Sniper.Task("task")
#task.asTop()
task.setLogLevel(3)

import HelloWorld
alg = task.createAlg("HelloAlg/hAlg")
alg.property("VarString").set("some value")
alg.createTool("HelloTool/htool")

svc = task.createSvc("HelloSvc/hSvc")

task.setEvtMax(5)
task.show()
task.run()
```

Data Store

- It is the dynamically allocated memory place to hold event(s) which are being processed
- Algorithms get event data from the Data Store and update/add event data after executions



Interfaces for access to Data Store

- DataStoreMgr is to adopt Event Data in DataStore under a certain path

```
simHeader->setEvent(simEvent);
SniperPtr<IDataStoreMgr> mMgr(getParent(), "DataStoreMgr");
mMgr->adopt(simHeader, "/Event");

simEvent->setNtracks(m_iEvt);
```

- EvtDataPtr is to retreive Event Data from DataStore with a unique path

```
EvtDataPtr<OSCAR::StcfGenHeader> edp(this->getRoot(),"/Event/StcfGenEvent");
OSCAR::StcfGenHeader* header = edp.data();
OSCAR::StcfGenEvent* event = header->event();
event->pEvt()->print();
```

Property :set parameters at runtime

- Configurable variable at run time
- Declare a property in DLElement(Alg, Svc, Tool and Task)

```
//suppose m_str is a string data member
declProp("MyString", m_str);
```

- Configure a property in Python script

```
alg.property("MyString").set("string value")
```

- Types can be declared as properties:
  - scalar: C++ build in types and std::string
  - std::vector with scalar element type
  - std::map with scalar key type and scalar value type

Logging：manage output message

- SniperLog: a simple log mechanism supports different output levels

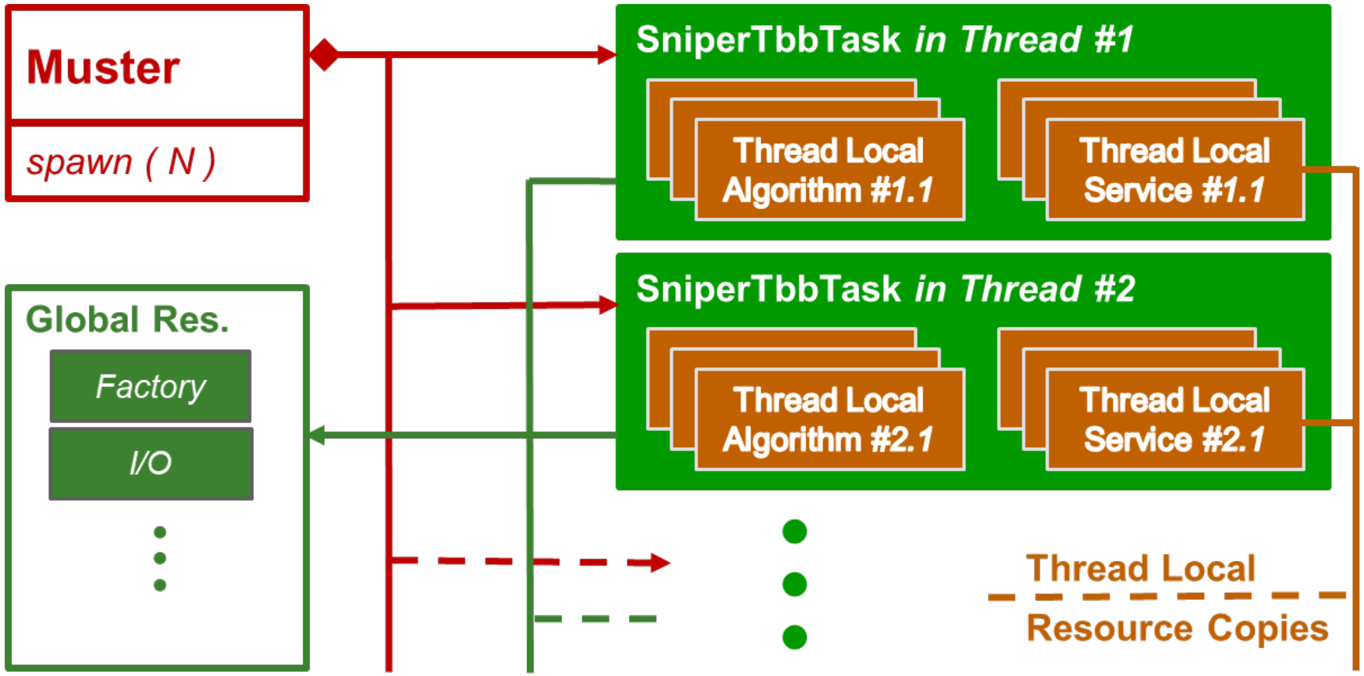
0: LogTest
2: LogDebug
3: LogInfo
4: LogWarn
5: LogError
6: LogFatal

```
LogDebug << "A debug message" << std::endl;
LogInfo  << "An info message" << std::endl;
LogError << "An error message" << std::endl;
```

```
aHelloAlg.execute           DEBUG: A debug message
aHelloAlg.execute            INFO: An info message
aHelloAlg.execute           ERROR: An error message
```

- Each DLElement(Alg,Svc,Tool, Task) has its own LogLevel and can be set at run time
  - ⇨ very helpful for debugging
- The output message includes more information , such as
  - ⇨ where it happens
  - ⇨ message level
  - ⇨ message contents

Parallelism



- Developed based on Intel TBB to Support event level parallelism
  - Muster: Multiple SNiPER Task Scheduler
  - SniperTbbTask: Binding of a SNiPER Task to a TBB task
- Global DataStore to provide events for multi-tasks (or multi-threads)
- A dedicated task(thread) is used to read/write event data from/to files

A typical Job configuration file

```python
import Sniper

task = Sniper.Task("task")
task.setLogLevel(3)

import DetSimAlg
alg = task.createAlg("DetSimAlg/DetSimAlg")
alg.property("DetFactory").set("DemoFacory")

import OSCARSim
tool = alg.createTool("GeoAnaMgr/GeoAnaMgr")
tool.property("Gdml2RootEnable").set(1)
tool.property("GdmlPath").set("/Event/DemoEvent")

import DataStoreMgr
task.createSvc("DataStoreMgr")

import RootIOSvc
oSvc = task.createSvc("RootOutputSvc/OutputSvc")
oSvc.property("OutputStream").set({"/Event/DemoEvent" : "DemoEvent.root"})

import G4Svc
g4svc = task.createSvc("G4Svc/G4Svc")

import DemoSim
factory = task.createSvc("DemoSimFactory/DemoFacory")
factory.property("AnaMgrList").set(["DemoAnaMgr", "GeoAnaMgr"])

task.setEvtMax(10)
task.show()
task.run()
~
```

Load the Library

Setup a Task

Add algorithm into the Task

Define Detector Geometry

Define Event Manager

Output Event Data to Files

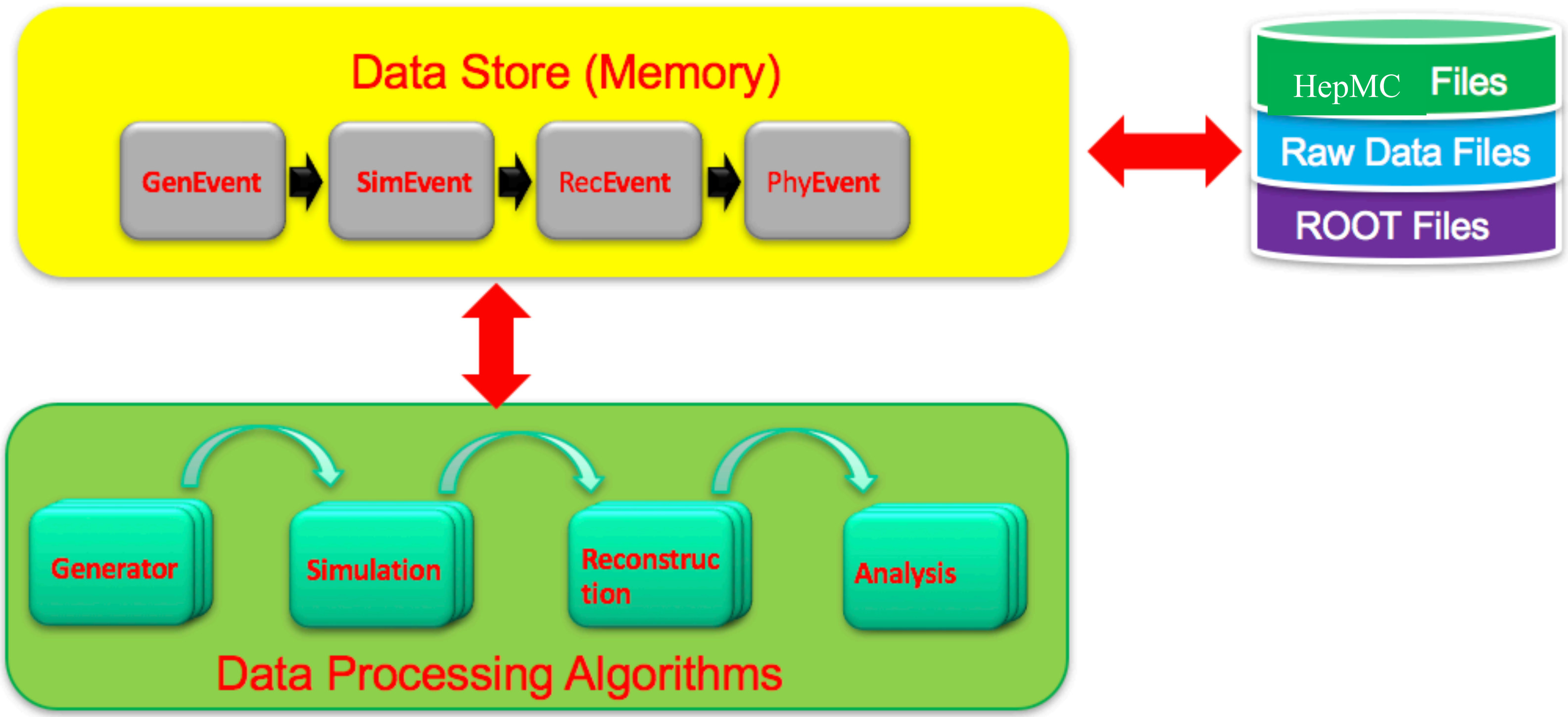
Set the Handler of Geant4

Define the number of Events to be proceed

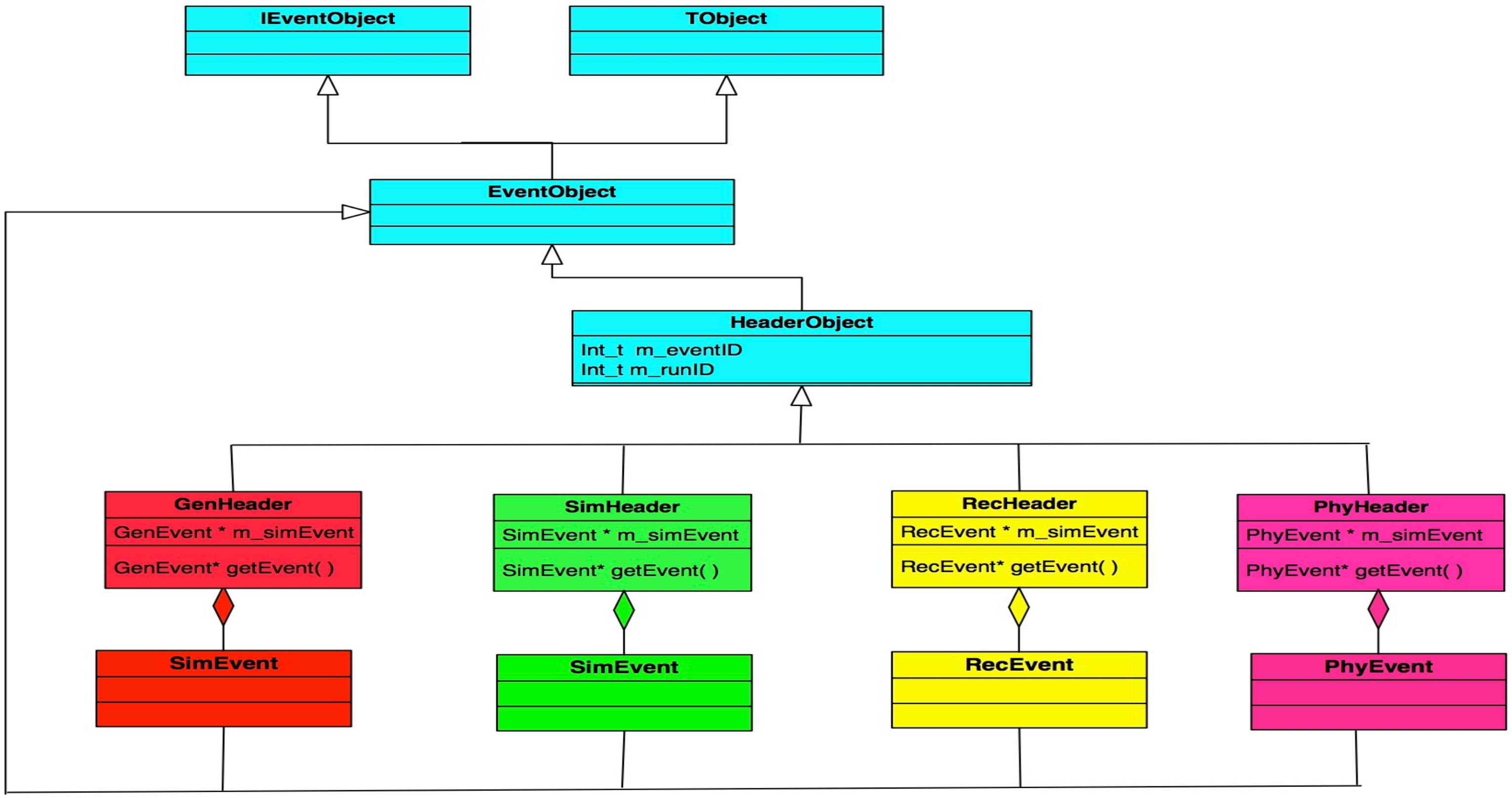
Invoke running

Event Data Model

- Definition of Event Information and correlation in different processing stages
- Key component and important for the software performance



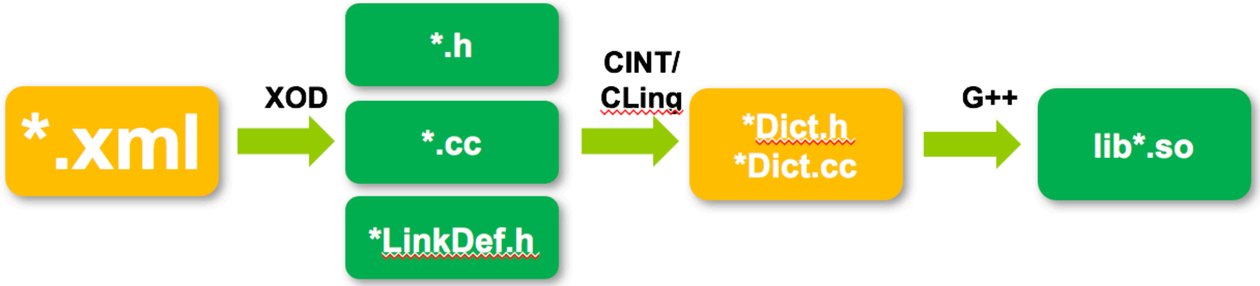
Event Data Model



- Event Objects are based on ROOT TObject
- One EDM both in memory and ROOT files to avoid conversion
- For each stage, Two-layer definition: HeaderObject and EventObject
- SmartRef for the correlation and supporting data-lazy loading

XOD: EDM Generation Toolkit

- Use XML file to define EDM
- XOD is developed to automatically generate class codes



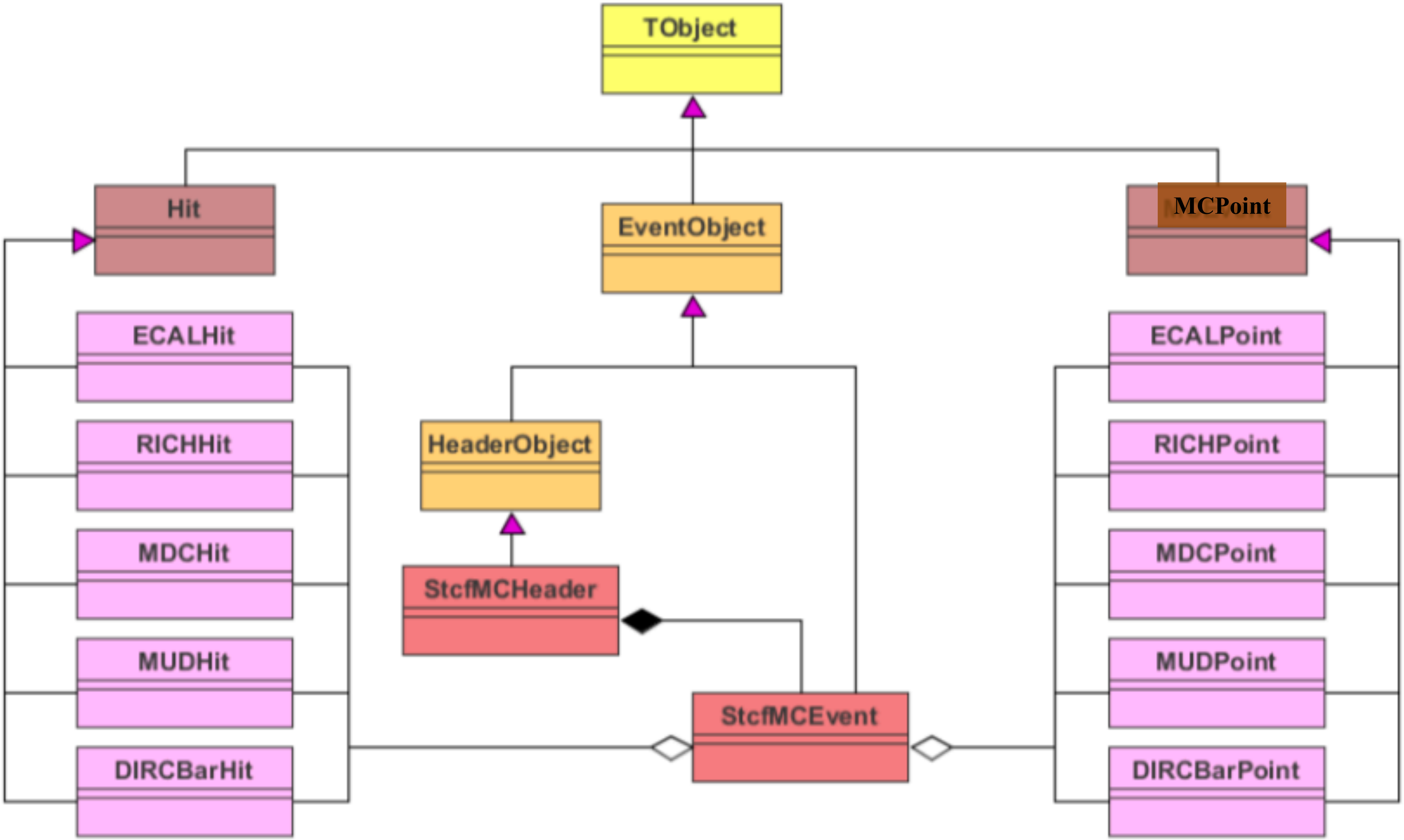
MCTrack.xml

```xml
<?xml version="1.0" encoding="UTF-8"?>
<!DOCTYPE xdd SYSTEM "xdd.dtd">
<xdd>
  <package name="StcfMCEvent">

    <import name="TObject"/>
    <import name="TVector3.h"/>
    <import name="TLorentzVector.h"/>

      <class name="MCTrack"
        author="Song Yong"
        desc="Data class for storing Monte Carlo tracks">
        <base name="TObject"/>

        <attribute name="PdgCode"
                   type="Int_t"
                   desc="PDG particle code"
                   init="0"/>

        <attribute name="FourMomentum"
                   type="TLorentzVector"
                   desc="Four-Momentum at track vertex [GeV]"
                   nonconstaccessor="TRUE"/>
```

MCTrack.h

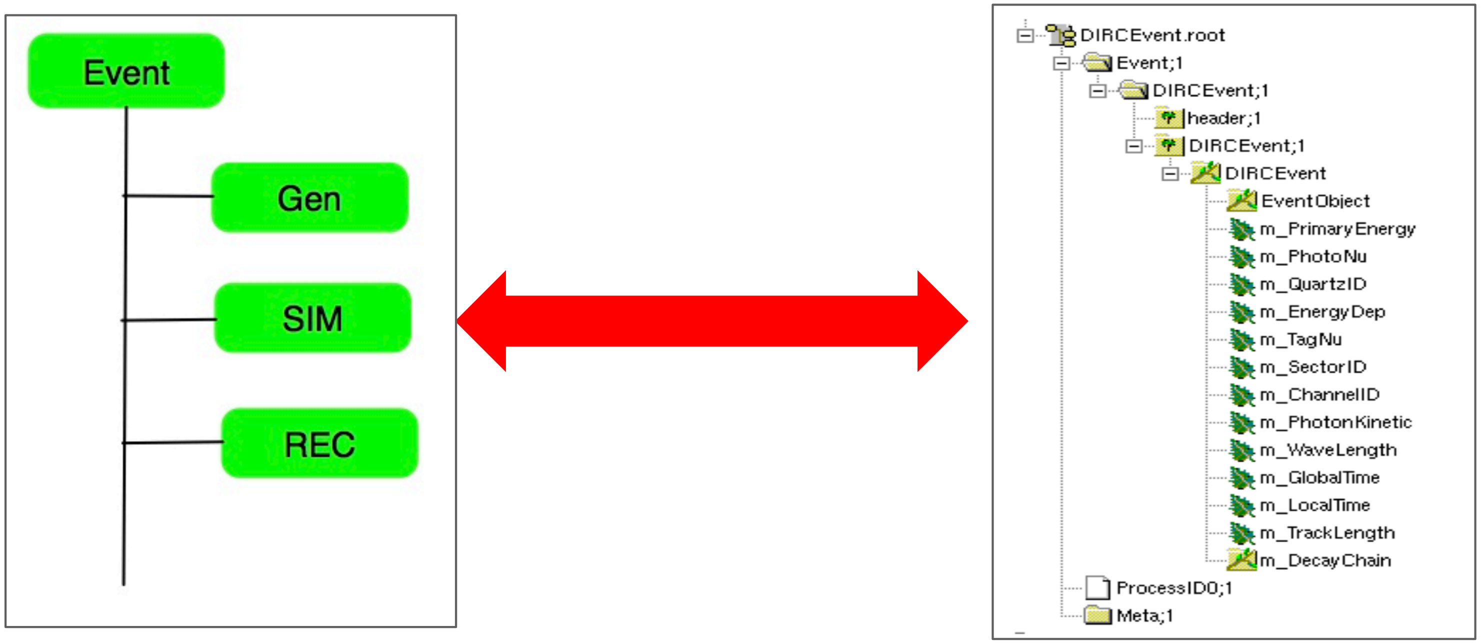
```cpp
class MCTrack: public TObject
{
private:

    Int_t          m_PdgCode;        // PDG particle cod
    TLorentzVector m_FourMomentum;   // Four-Momentum at
    Int_t          m_MotherID;       // Index of mother
    Int_t          m_GeneratorFlags; // Flag if particle
    TLorentzVector m_StartVertex;    // Start track vert
    TLorentzVector m_StopVertex;     // Stop track verte

    /// PDG particle code
    const Int_t& PdgCode() const;
    /// PDG particle code
    void setPdgCode(const Int_t& value);
    /// Four-Momentum at track vertex [GeV]
    const TLorentzVector& FourMomentum() const;

    /// Retrieve
    /// Four-Momentum at track vertex [GeV]
    TLorentzVector& FourMomentum();

```

MCEvent


TObject
Hit
EventObject
MCPoint
ECALHit
RICHHit
MDCHit
MUDHit
DIRCBarHit
HeaderObject
StcfMCHeader
StcfMCEvent
ECALPoint
RICHPoint
MDCPoint
MUDPoint
DIRCBarPoint

ROOT Input/Output System

- General RootInputSvc/RootOutputSvc
  - RootInputSvc: read Event Data from Root Files to Data Store
    - Correlation between header and event will automatically build up
  - RootOutputSvc: write Event Data from Data Store to Root Files
    - Root Files could be analyzed with root macro scripts
  - All Event data can be read/written automatically with current IO system



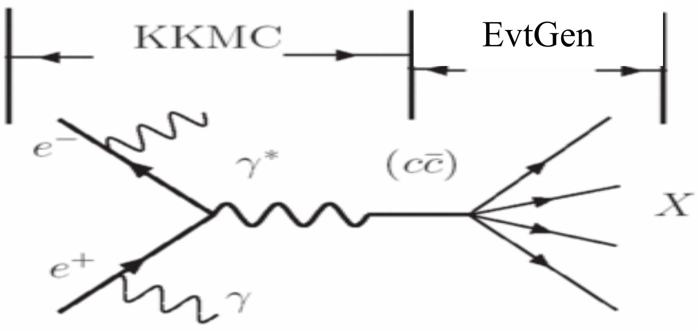
DataStore **ROOT File**

Generator

- **Babayaga**
  - e^+e^- --> e^+e^-, $\mu^+\mu^-$, $\gamma\gamma$ and $\pi^+\pi^-$ QED processes at flavor factories
- **Phokhara**
  - **e^+e^-** annihilation into hadrons plus an energetic photon from initial state radiation (ISR)
- **KKMC**
  - Charmonium production with beam spread and ISR
- **EvtGen**
  - Charmonium decays

Generator Name: Babayaga
--Options:
Channel: 1: $e^+e^- \to (n\gamma)e^+e^-$
2: $e^+e^- \to (n\gamma)\mu^+\mu^-$
3: $e^+e^- \to (n\gamma)\gamma\gamma$
4: $e^+e^- \to (n\gamma)\pi^+\pi^-$
C.M. Energy = 2* E_{beam}
Running α : 0=off, 1=on
FSR switch (for ICH=2): 0=off, 1=on
---cuts:
charged particles: E_{min} (MinimumEnergy), θ_{min} (MinThetaAngle), θ_{max} (MaxThetaAngle) MaximumAcollinearity
photons : MinEnergyCutG MinAngCutG MaxAngCutG

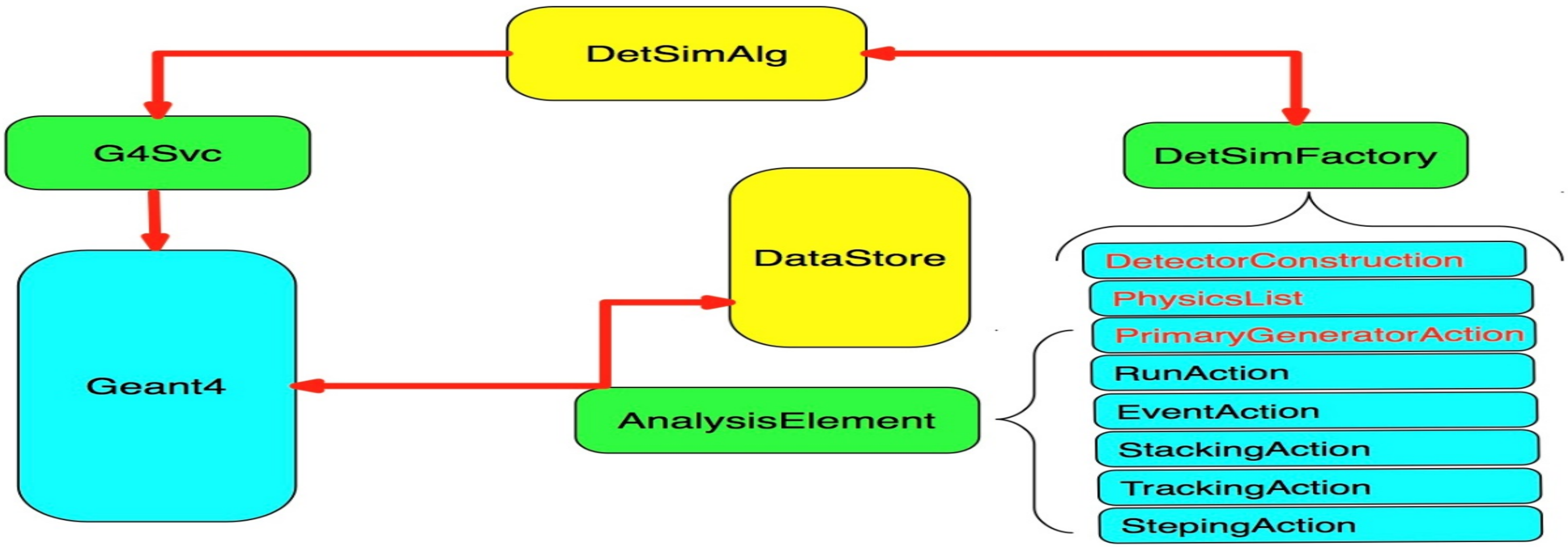
Mode
$\pi^+\pi^-$
$\pi^0\pi^0\pi^+\pi^-$
$2(\pi^+\pi^-)$
ppbar
nnbar
K^+K^-
KsK_L
$\pi^+\pi^-\pi^0$
$\Lambda\Lambda$bar



Babayaga, Phokhara and KKMC are working in OSCAR
EvtGen will be ready soon

Detector Simulation Framework

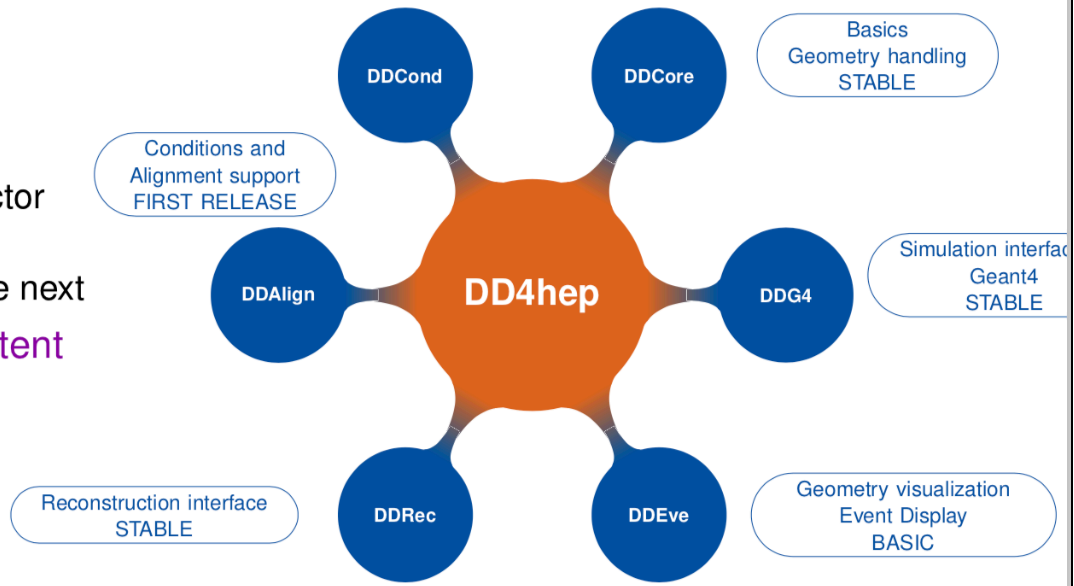
- OSCAR manages detector simulation with **Task**,
  - ⇨ The algorithm (**DetSimAlg**) for all sub-detectors simulation
  - ⇨ The service (**G4Svc**) to launch Geant4 within OSCAR
  - ⇨ The user-end service(**DetSimFactory**) to set up the Geant4 related classes
  - ⇨ The user-end service(**AnalysisElement**) to retrieve G4Event and create Event Data in Data Store



Detector Geometry Description :DD4hep

DD4hep – Overview

- Complete Detector Description
  - Providing geometry, materials, visualization, readout, alignment, calibration. . .
- Supports full experiment life cycle
  - Detector concept development, detector optimization, construction, operation
  - Facile transition from one stage to the next
- Single source of information $\rightarrow$ consistent description
  - Use in simulation, reconstruction, analysis, etc.
- Ease of Use
- Few places for entering information
- Minimal dependencies



- Used by ILC and CLIC, FCC, CEPC, STCF and SCT …

Use XML file and C++ driver to build Detectors

Detector XML

AIDA 2020 CERN

- XML structure to set parameters for detectors
- C++ driver to interpret XML parameters and create DetElements and Volumes
  - Define sensitive parts (attached with SensitiveDetector) and radiator, which has to be known fo
- Attach sensit in XML

```
<readout name
  <segmentati

  <id>...x:32
</readout>
```

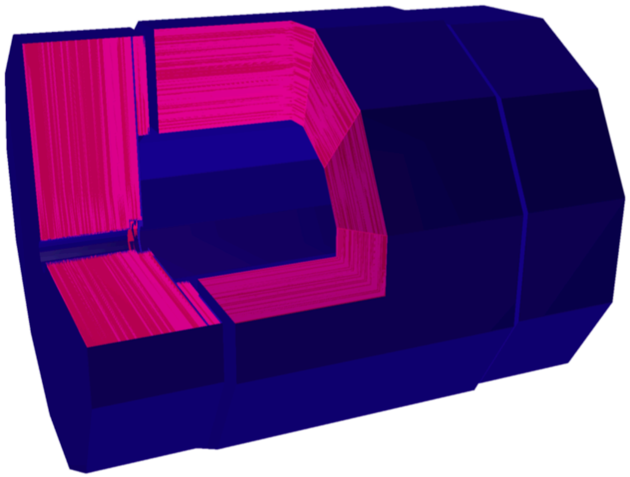
```
<detector
    name="ECalBarrel"
    type="GenericCalBarrel_o1_v01"
    id="42" readout="ECB">
  <dimensions
      numsides="ECalBarrel_symmetry"
      rmin="ECalBarrel_inner_radius"
```

A. Sailer

Detector Driver

AIDA 2020 CERN

- C++ model of separation of ‘data’ and ‘behaviour’
- Drivers return single ‘reference’ to the DetElement object



```
static dd4hep::Ref_t create_element(
    dd4hep::Detector& description,
    xml_h element,
    dd4hep::SensitiveDetector sens) {
  xml_det_t e = element;
  DetElement aDetector(e.nameStr(), e.id());
  //...
  sens.setType("calorimeter");
  //...
  return aDetector;
}
DECLARE_DETELEMENT(AName, create_element)
```

Detector Description with DD4hep

- Define geometry and materials in xml files

```
-bash-4.1$ ls
detectorDIRC.xml  detectorMUD.xml           detectorVTD.xml  materials01.xml     STCFECAL.xml
detectorECal.xml  detectorPID.xml           elements01.xml   materials02.xml     STCF_test.xml
detectorMDC.xml   detectorRICHBarrel.xml    elements02.xml   materials.xml       STCF.xml
detectorMUC.xml   detectorSC.xml            elements.xml     muondetector2.xml
```

- Construct detector in c++ driver files

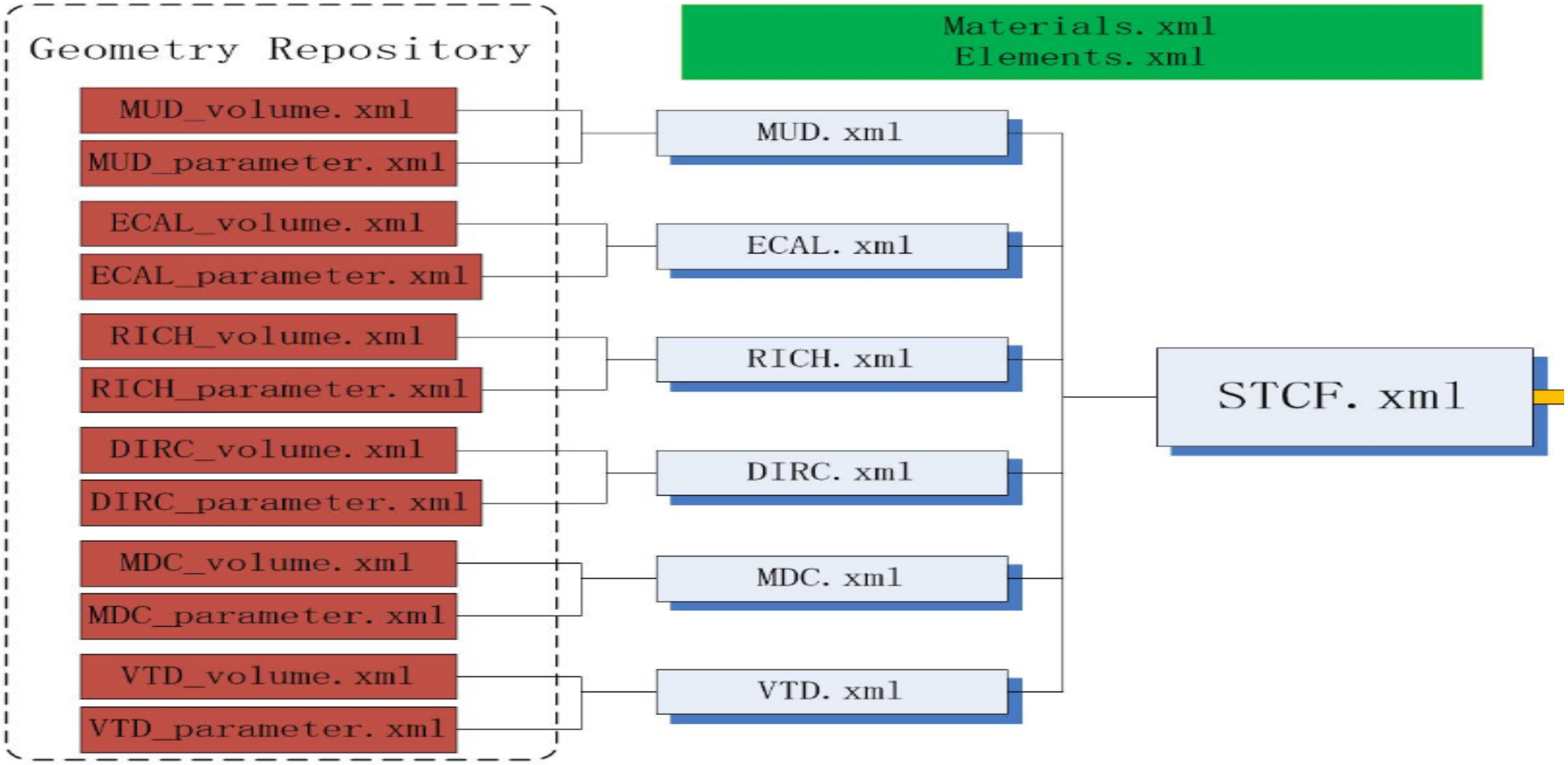
```
-bash-4.1$ ls
AirTube_geo.cpp    DIRC_geo.cpp                           SCTube_geo.cpp     Tracker_geo.cpp
BarrekDIRC_geo.cpp InnerPlanarTracker_geo.cpp             STCF_BEMC_geo.cpp  TrackerSupport_geo.cpp
detectorMUD.cpp    PolyhedraEndcapCalorimeter2_geo.cpp    STCF_EEMC_geo.cpp  ZPlanarTracker_geo.cpp
```

- Deliver detector geometry to Geant4

```
import DetGeoConsSvc
myxmlsvc = task.createSvc("DetGeoConsSvc")
myxmlsvc.property("DetGeoConsSvcEnable").set(1)
myxmlsvc.property("GeoCompactFileName").set("/afs/ihep.ac.cn/soft
install/examples/ClientTests/compact/detectorSC.xml")
```

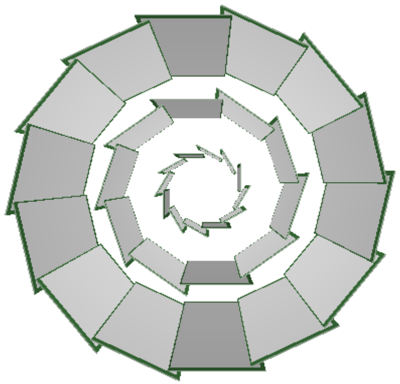
Detector Geometry Management

- Sub-detectors are described with DD4hep
- Each sub-detector is independent with others, different version in different path
- Flexible to build a full detector with different combinations of sub-detectors
- Common files for materials and elements

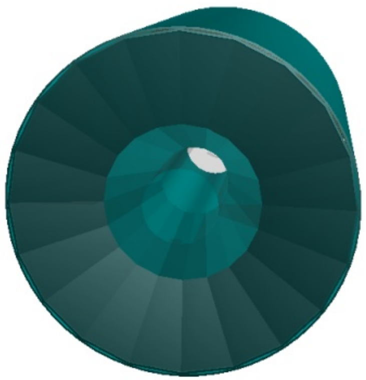


Detector Visualization

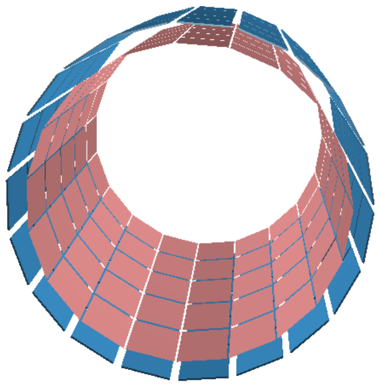
- Sub-detectors can be displayed individually with geoDisplay Plugin



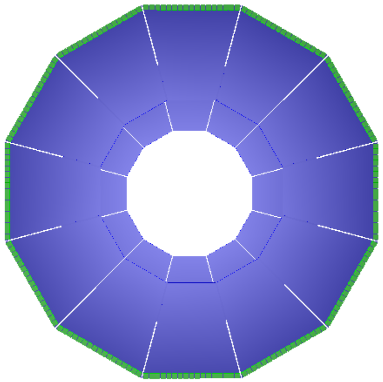
Vertex Detector



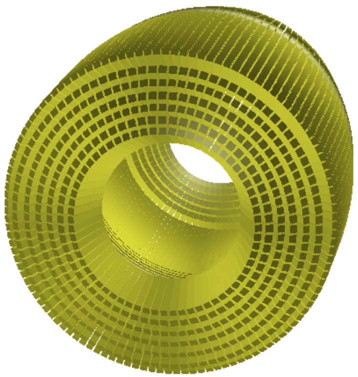
Main Drift Chamber



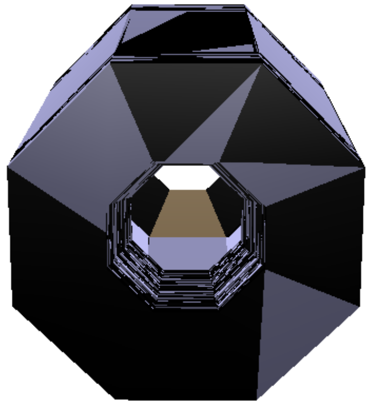
RICH



DIRC

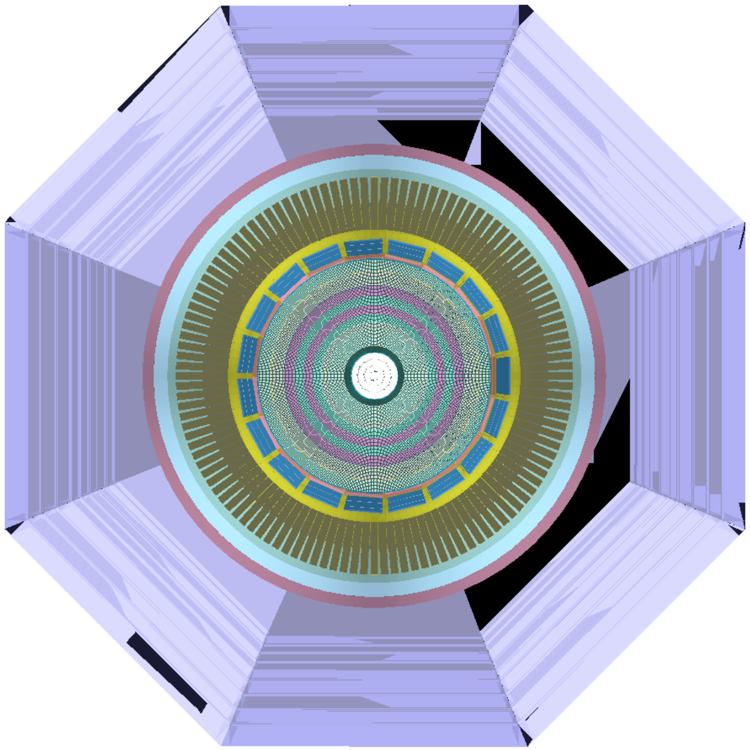


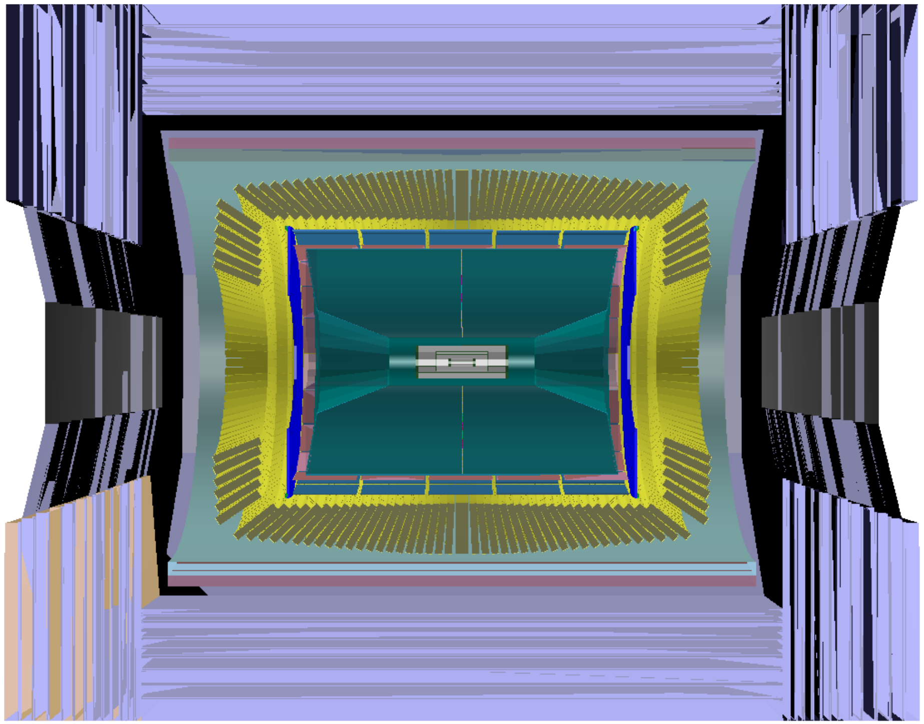
Calorimeter



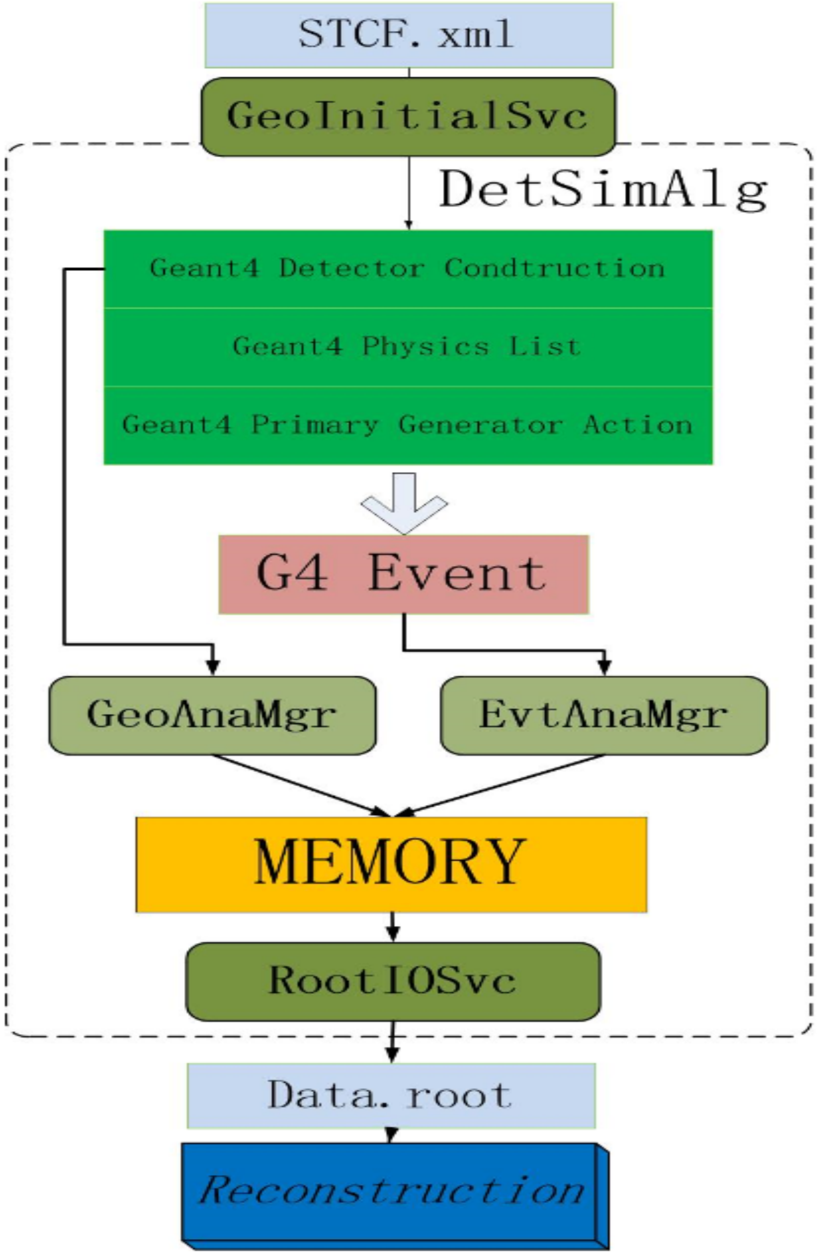
Muon Detector

Detector Visualization

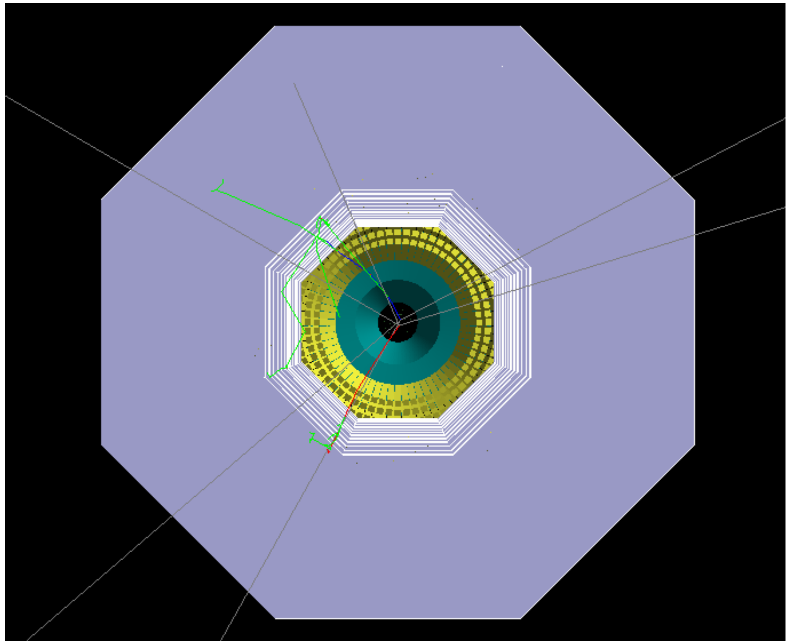


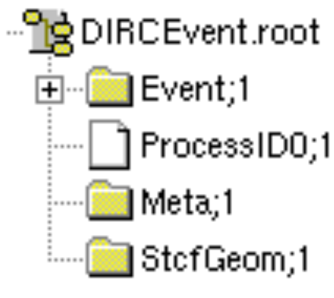


Set up Full Detector Simulation Chain



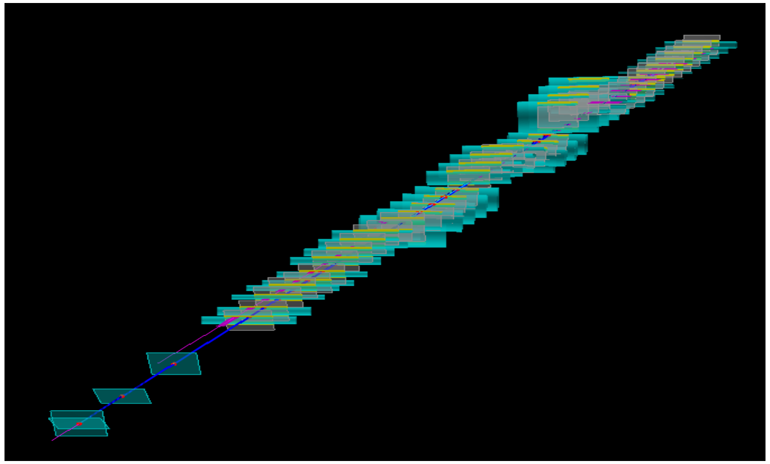
Display of a Event: e+ e- @Ecm =7GeV
Geometry was initialized with DDG4 from xml file



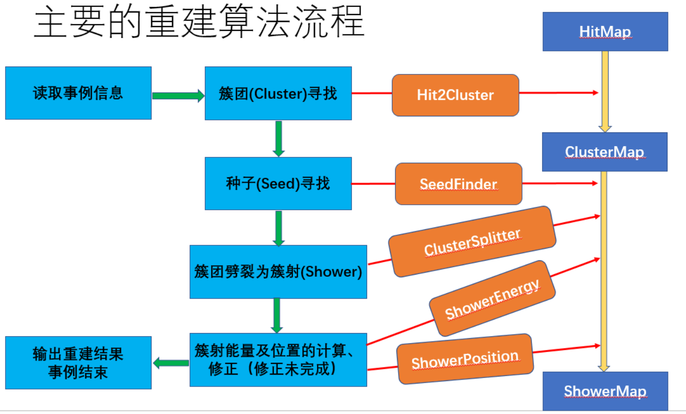


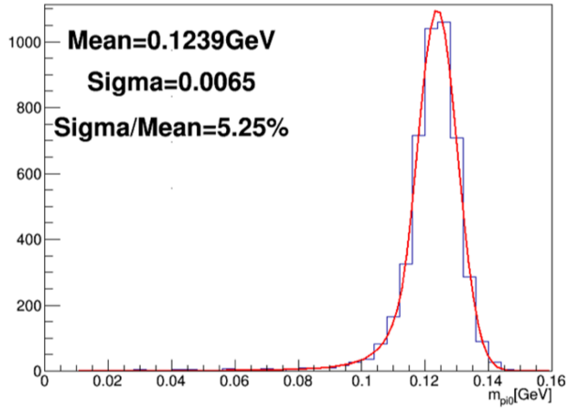
Reconstruction

- MDC
  - Single tracking study
- EMC
  - Seed finding
  - Clustering
- PID
  - DIRC: ML method
  - Muon: BDT method

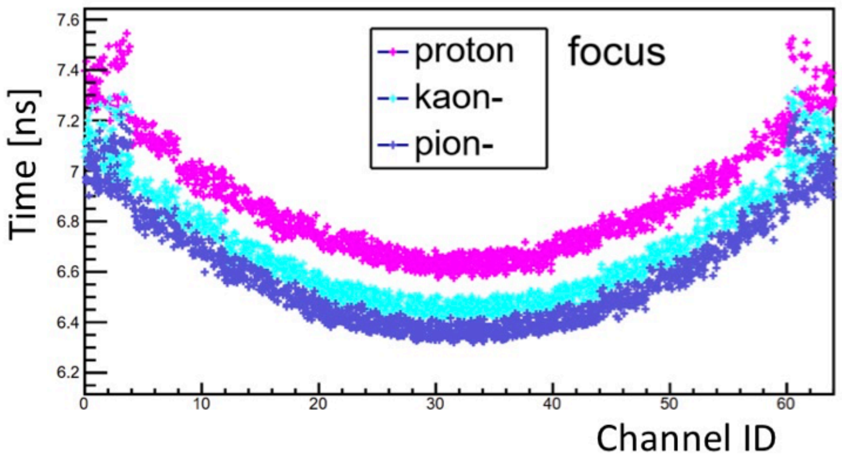


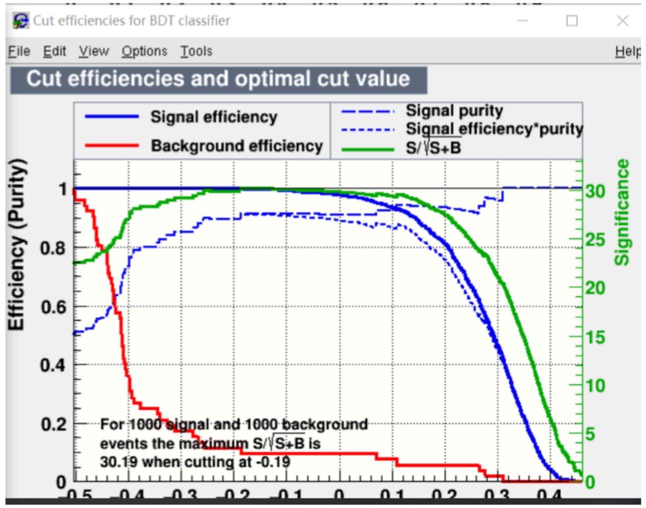
主要的重建算法流程





1.5GeV π^0静质量的重建结果





Installation, documentation and SVN

- The latest version of OSCAR is installed in USTC nodes
  - stcf01.ustc.edu.cn
  - stcf01.ustc.edu.cn

- Installation
  - Automatic installation of the whole offline software with a shell script
  - svn export http://202.141.163.202/svn/oscar/installation/trunk/setup-trunkj.sh

- Documentation
  - OSCAR User Guide
    - http://cicpi.ustc.edu.cn/indico/getFile.py/access?contribId=1&resId=0&materialId=slides&confId=1610

- SVN repository
  - http://202.141.163.202/svn/oscar/

Summary and Outlook

- **OSCAR is developed for STCF**
  - Based on SNiPER and DD4hep
  - Event Data Management
  - Data proceeding management
  - Common Services and User interface
  - Serve as the unifed platform for application development
- **Lots of progress have be made**
  - Generators: Babayaga, Phokhara and KKMC
  - Detector geometry description with DD4hep: modular and flexible
  - Detector geometry management: Xml->Geant4-> ROOT->Recon.
  - Event data model: currently based on ROOT
  - Root Input/Output System
  - The detector simulation chain has been setup
  - Development of reconstruction algorithms is in progress

Summary and Outlook

- **Lots of works ahead, more people are welcome**
  - Event Data Model for Simulation and Reconstruction
  - Generator framework: More generators and Unified interface
  - Optimize detector description:
    - Missing parts, precision description, digitization and realization
  - Study of Calibration and Reconstruction methods
  - Compare sub-detector performances between simulation and beam testing.
- **Setup a full chain from generator to reconstruction for optimization of Detector design and performance study.**
  - Tracking efficiency
  - Energy, Momentum, position resolutions
  - Discrimination of electron/pion, muon/pion, kaon/pion
- **Keep eyes on the new development of the community**
  - DD4hep: common Detector Description (already used by STCF)
  - EDM4hep: common Event Data Model (STCF prototype for testing)
  - Key4hep: common Software Stack

Thanks for your attention!