Introduction to the Software System Based on SNiPER

Xingtao Huang, Meng Wang
Shandong University

The 8th HERD Workshop
Dec. 16-17, 2019, Xi'an, China

Shandong University

Happy to join HERD Collaboration

History

- Founded in 1901
- The 2nd national university established in China in Qing Dynasty
- Located in Shandong

Scale

- 8 Campuses in three cities
- 32 Schools
- 8 K Employees
- 1 K Professors
- 40 K Undergraduates
- 11 K Graduates

Research Center for Particle Science and Technology

Manpower

- 7 faculty members on theoretical particle physics
- 25 faculty members (including 5 technicians) on experimental particle physics
- 13 post-docs

Experiments and Research Activities

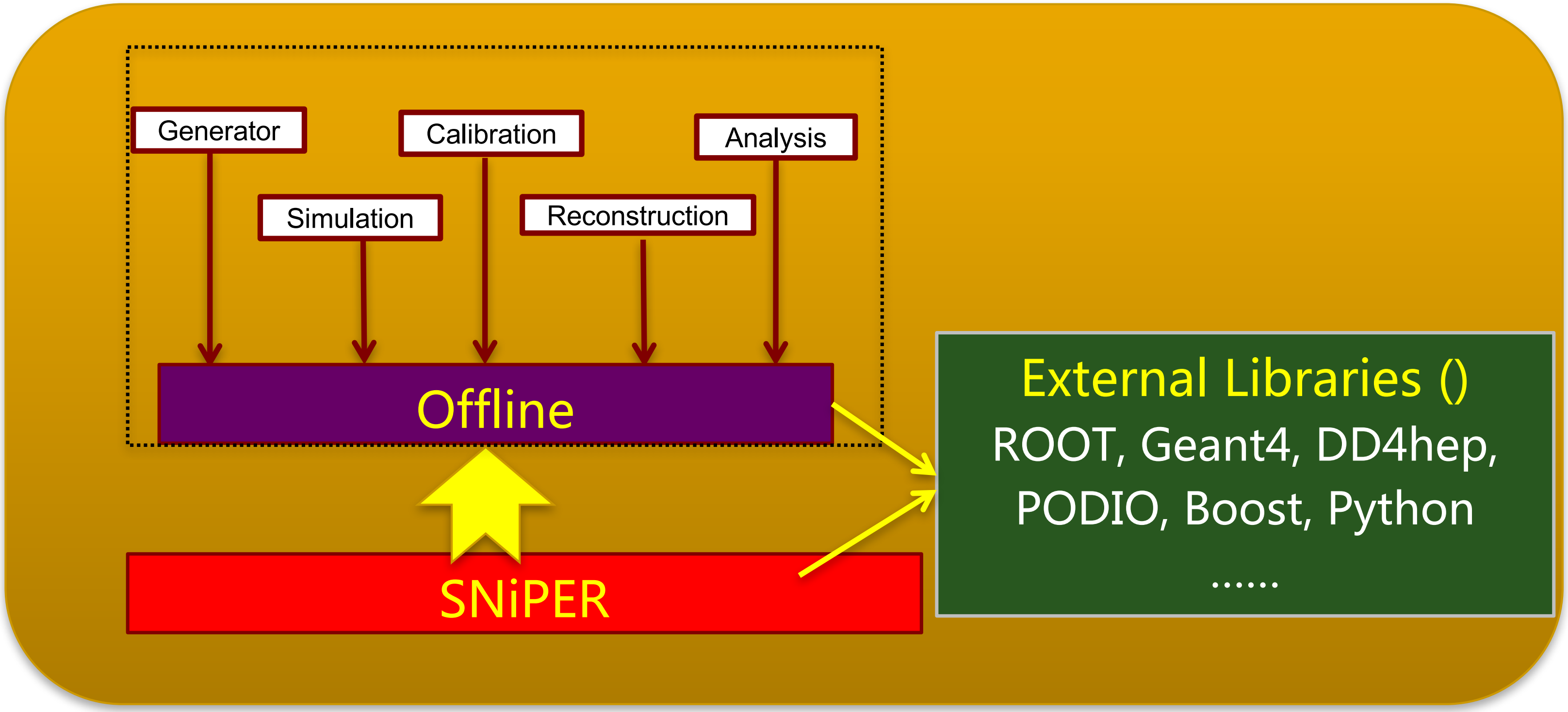
	Detector	Software	Physics
ATLAS	TGC		Top, Higgs
BESIII		Framework, tracking	Charmonium, light hadron
CEPC	Silicon	Framework,simulation	Higgs
STAR	iTPC		Spin Physics
Daya Bay/JUNO		Framework	Theta13
PandaX	DAQ		Dark matter
LHAASO	ED/PMT	Framework	Gamma sources
STCF		Framework	Charmonium, light hadron
HERD	under investigation		

Outline

- Framework
  - Introduction to the Framework, SNiPER
  - Key Features and Components of SNiPER
- Detector Simulation
  - DD4hep: Detector Geometry Description Toolkit
  - Detector Simulation within SNiPER
- Event Data Model
  - PODIO: Event Data Model Toolkit
- Software Management
  - Computing environment, Tools , Documentation
- Summary

Architecture of Offline Software System

- **Offline**
  - Specific to the Physics Experiment
  - Including Generator, Simulation, Calibration , Reconstruction and Analysis
- **SNiPER Framework**
  - Data Processing Management, Event data Management, Common Services…
- **External Libraries**
  - Frequently used third-party software and tools



About SNiPER (I)

- SNiPER: the "Software for Non-collider Physics ExpeRiment"
  - Developed for JUNO experiment
  - But also considered for other non-collider physics experiments

- Design and development
  - Learn a lot from other software frameworks, such as Gaudi
  - Based on the valuable experiences from DayaBay experiment
  - Coding from scratch since 2012

About SNiPER (II)

- Main goals
  - Lightweight, less dependences on third-party software/libs
  - Fast and flexible execution
  - Easy to learn and convenient to use
- Used by Several Experiments
  - JUNO (Jiangmen Underground Neutrino Observatory) in China
  - LHAASO (Large High Altitude Air Shower Observatory) in China
  - STCF (Super Tau-Charm Facility) in China
  - nEXO (next Enriched Xenon Observatory) in U.S.
  - ……
- A Good Team to maintain and optimize
  - SDU and IHEP

Key Features of SNiPER (I)

- Highly modular
  - Each module is functionally independent
  - Main functions for data processing have been implemented in kernel modules
- Standard interfaces between different modules
  - The interfaces have been very stable
  - People from each experiment only focus on event data model, algorithms, detector geometry etc.
- Dynamically loading packages/modules/elements
  - New packages can be easily loaded/used as plugins by framework

Key Features of SNiPER (II)

- Separation between data and algorithm
  - Less coupling between algorithms
  - Development of new algorithms at the same time
- Data Store for event data management
  - Algorithms retrieve/put event data from/to Data Store
- Flexible event execution
  - Sequential execution
  - Jump/nested execution
- Support multithreading
  - Underlying the intel TBB is deployed
  - Multi-tasks naturally maps with multi-threads

Key Components for Users

- Algorithm
- Service
- Task
- Data Store
- Property
- Logging
- Job Configuration with python
- Multithreading

They are dynamically Loaded Elements (DLElement) and configured in python script

Algorithm

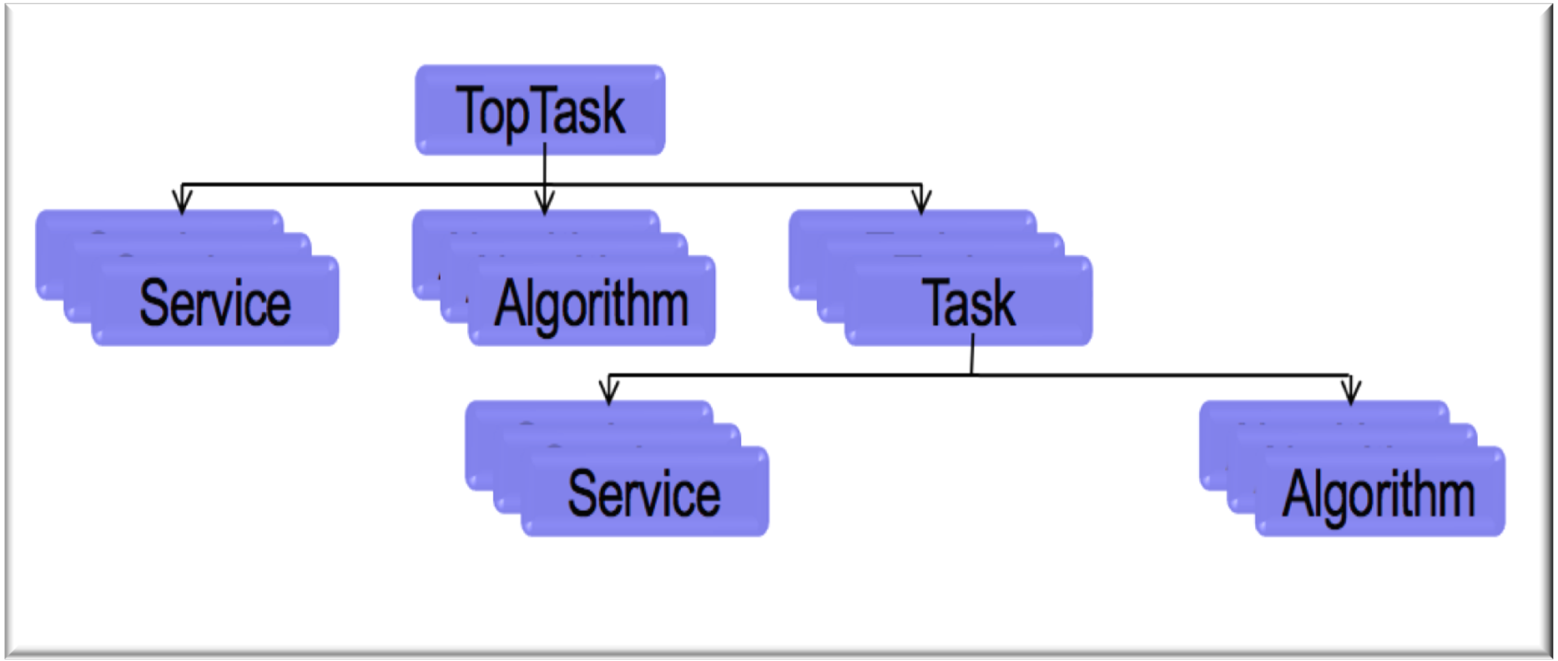
- An unit of code for event execution
  - Perform event calculation during event loop
- SNiPER provides the interface, AlgBase
- User's new algorithm inherits from AlgBase
  - Its constructor takes one std::string parameter
  - 3 member functions must be implemented
    - bool initialize() : called once 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

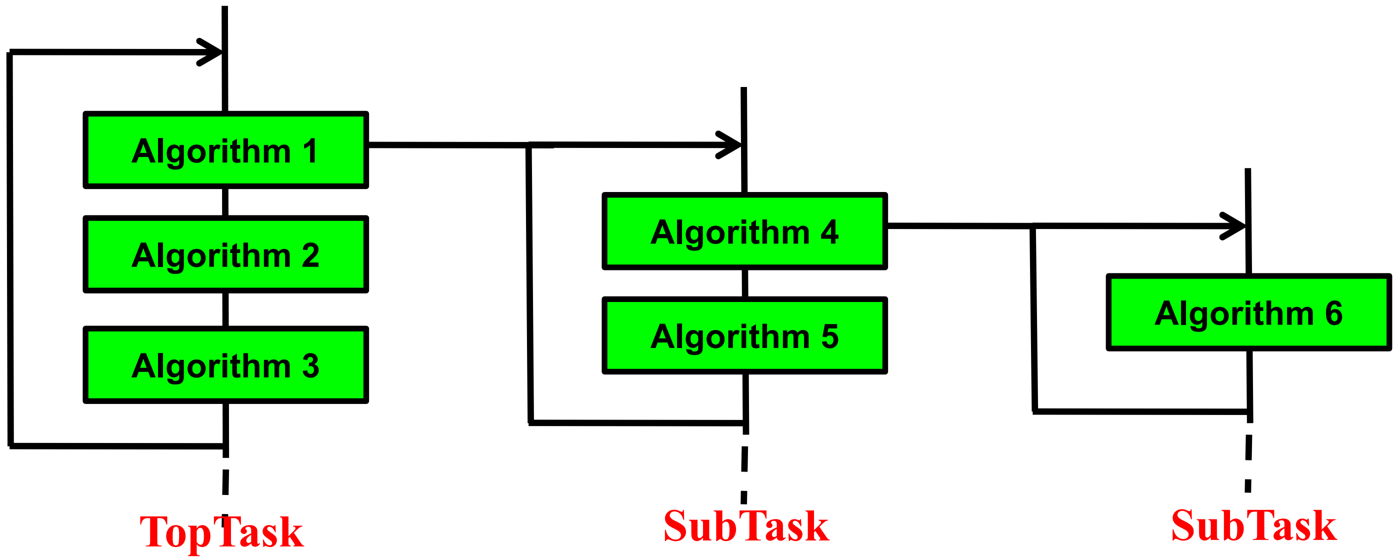
Service

- 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
- SNiPER provides the interface, SvcBase
- New services inherit 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)
- New services can be used by all algorithms

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



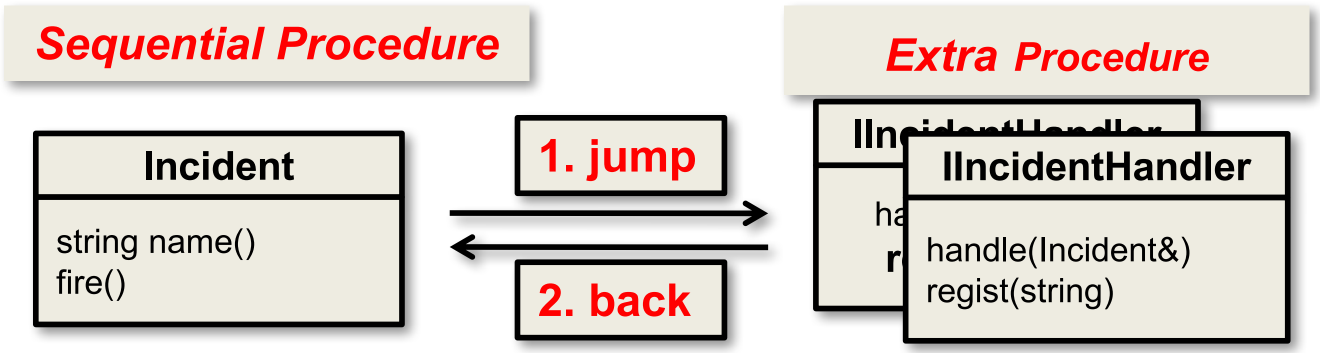


- Algorithms in one Task are sequentially executed
  - In the order of algorithm position

- SubTask provides jump execution
  - It will be invoked on demand
  - After execution, return back to the upper task

Incident

- Incident provides jump execution procedure
- IncidentMgr correlates incidents with their handlers
  - Incident: trigger the execution of corresponding handlers
  - IncidentHandler: wrapper of any specific execution procedure

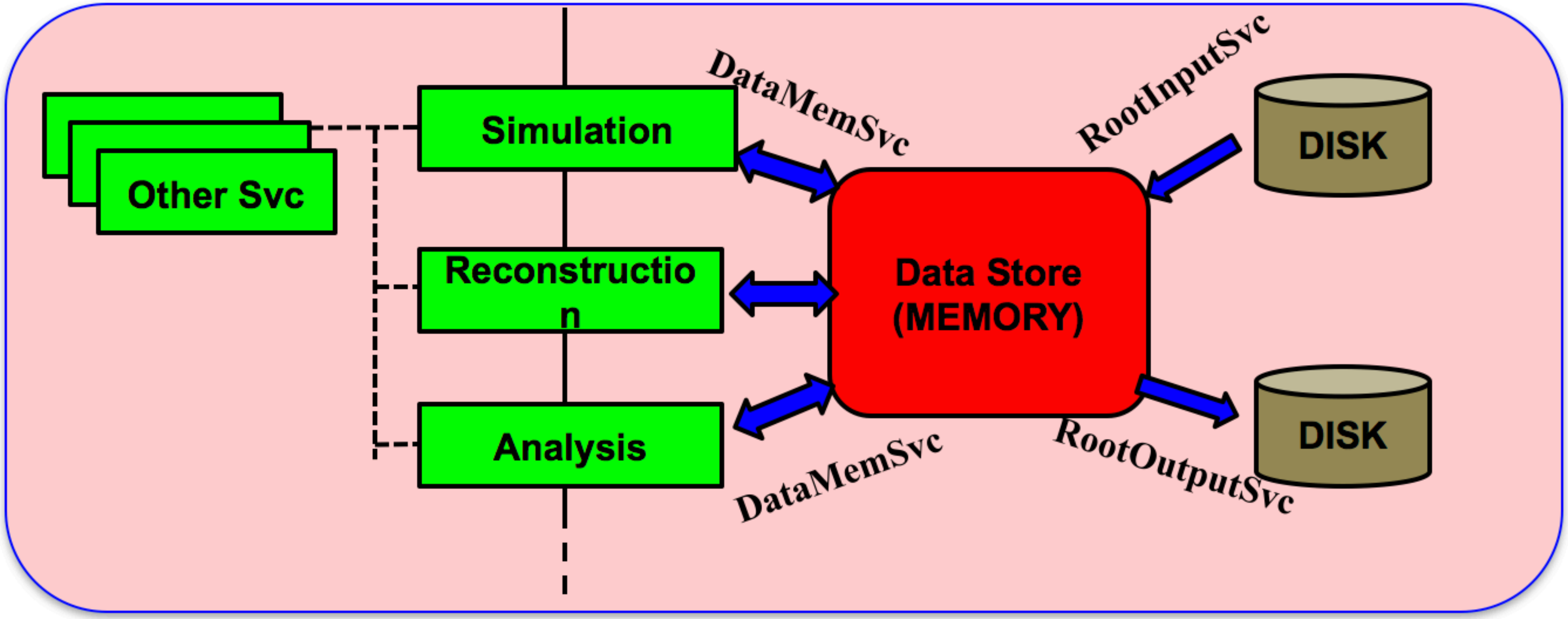


1. Regular execution procedure jumps to another extra procedure
2. Back to the original procedure after all corresponding Handlers are executed

- Both Algorithms and Services can fire incidents
  - Root I/O is based on incident mechanism

Data Store

- It is the dynamically allocated memory place to hold events data which are being processed
- Algorithms retrieve event data from the Data Store and put new event data back to Data Store



Layout in Data Store and Root File

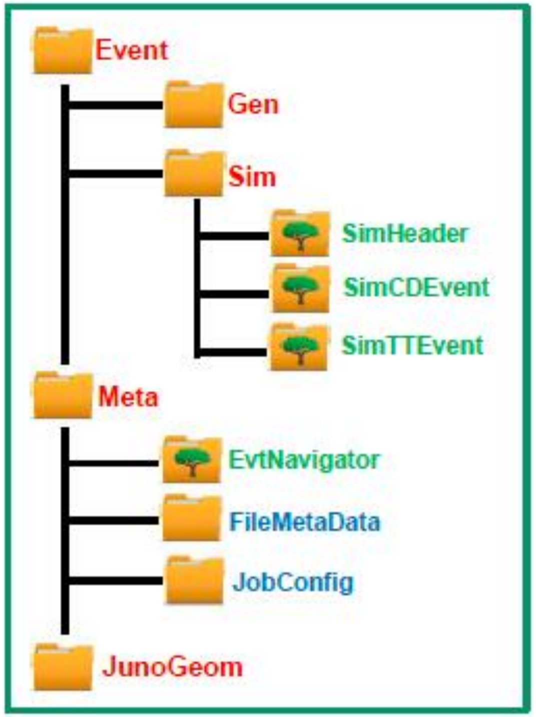
In Data Store

- Directory Structure
- Unique path

```
/Event
/Event/Gen
……
/Event/Sim
/Event/Sim/SimHeader
/Event/Sim/SimCDEvent
/Event/Sim/SimTTEvent
……
```

In Root File

- Tree Structure
- Tree/branch name
  - same with Data Store path



Standard interfaces for Access to Event Data

- User interface, SniperDataPtr, is provided to retrieve the Event Buffer and Get Current Event with the path

```
SniperDataPtr<SimCDEvent>  navBuf(getScope(), "/Event/Sim/SimCDEvent");
m_buf = navBuf.data();
SimCDEvent* nav=m_buf->curEvt();
```

- The Service, BufferMemMgr, is used to put/adopt event back to Buffer with a unique path

```
SniperPtr<IDataMemMgr> mMgr(getScope(), "BufferMemMgr");
SimCDEvent* cdevent = new SimCDEvent();
mMgr->adopt(nav, "/Event/Sim/SimCDEvent");
```

Property

- Configurable variable at run time
- Algorithm, Service and Task can declare their member variable as Property

```
//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

- SniperLog 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;
```

- Alg/Svc/Task can set their own LogLevel at run time

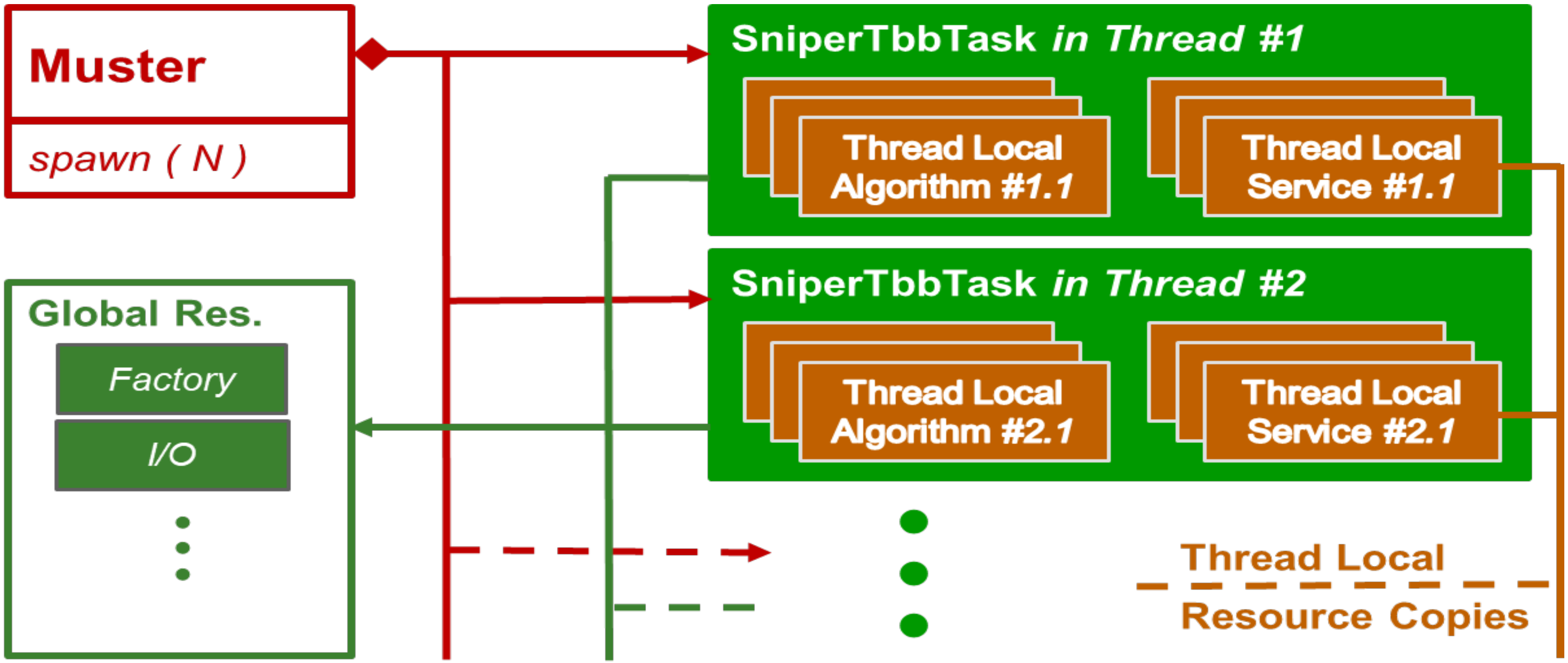
- The output message includes more information , such as
  - where it comes from
  - Level of message
  - Contents of message

very helpful for debugging codes

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

Multithreading of SNiPER: MT-SNiPER

- Developed based on Intel TBB.
  - Muster: Multiple SNiPER Task Scheduler
  - SniperTbbTask: Binding of a SNiPER Task to a TBB task
- JUNO detector simulation works well with MT-SNiPER



A Job Configuration File with python

Helloworld.py

```python
import Sniper

task = Sniper.Task("task")
task.setLogLevel(2)

import HelloWorld
alg = task.createAlg("HelloAlg/dalg")
alg.property("someString").set("some value")

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

```
lhaaso:~ huangxt$ python Helloworld.py
```

Run it !

Integration with other promising tools

- Members of the FCC, ILC, CEPC, SCT, STCF,CLIC, LHC communities met for a Future-Collider-Software Workshop in Bologna on June 12&13 https://agenda.infn.it/event/19047/

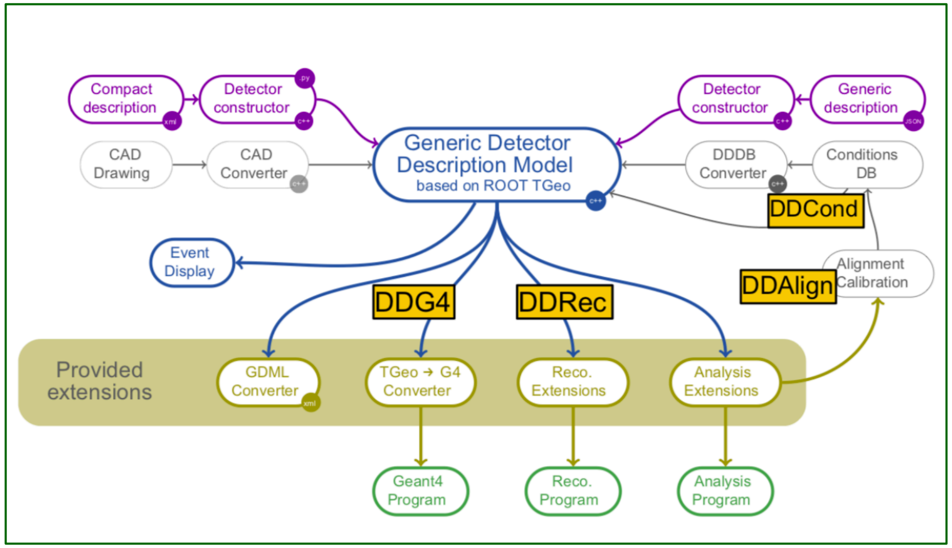
- Reached an Agreement to share the common packages or tool and create common turnkey software stack(Key4hep)
  - DD4hep for Detector Geometry Description
  - PODIO for building Event Data Model (EDM4hep)
  - ……

DD4hep: a generic Detector Description tool for HEP

AIDA 2020

[F. Gaede (CHEP2019)]

- Developed in AIDA/AIDA2020 , and used by ILD, CLICdp, FCC-ee, FCC-hh, CEPC, LHCb, CMS, SCT and STCF.
- Support the full life cycle of the experiment
  - Detector concept development
  - Detector optimization
  - Construction and operation



- Consistent description with one single data source for
  - Simulation, reconstruction and analysis
- Geometry description with compact xml-files and C++ drivers
- Use Root TGeo as geometry implementation
- Provide output formats or interfaces: Geant4 , GDML…

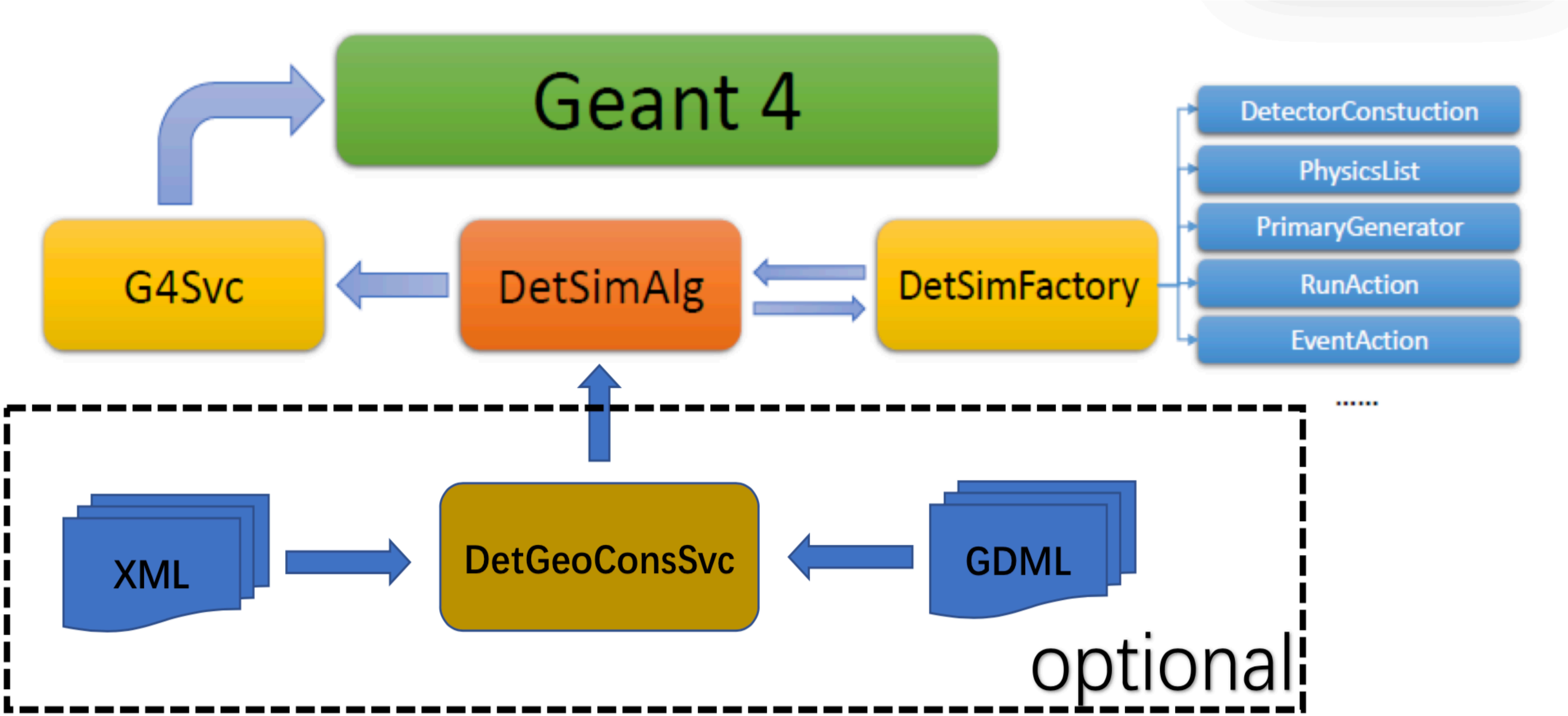
Detector Simulation within SNiPER

- SNiPER manages detector simulation with Task
  - A dedicated algorithm (DetSimAlg) for all sub-detectors simulation
  - A dedicated service (DetGeoConsSvc) to convert xml or gdml of DD4hep to Geant4
  - A dedicated service (G4Svc) to launch Geant4
  - A user-end service (DetSimFactory) to set up all the Geant4 related classes

- G4VUserDetectorConstruction
- G4VUserPhysicsList
- G4VUserPrimaryGeneratorAction

- G4UserRunAction
- G4UserEventAction
- G4UserStackingAction
- G4UserTrackingAction
- G4UserStepingAction

Overview of Detector Simulation System



DD4hep example: STCF Detector Description

- 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

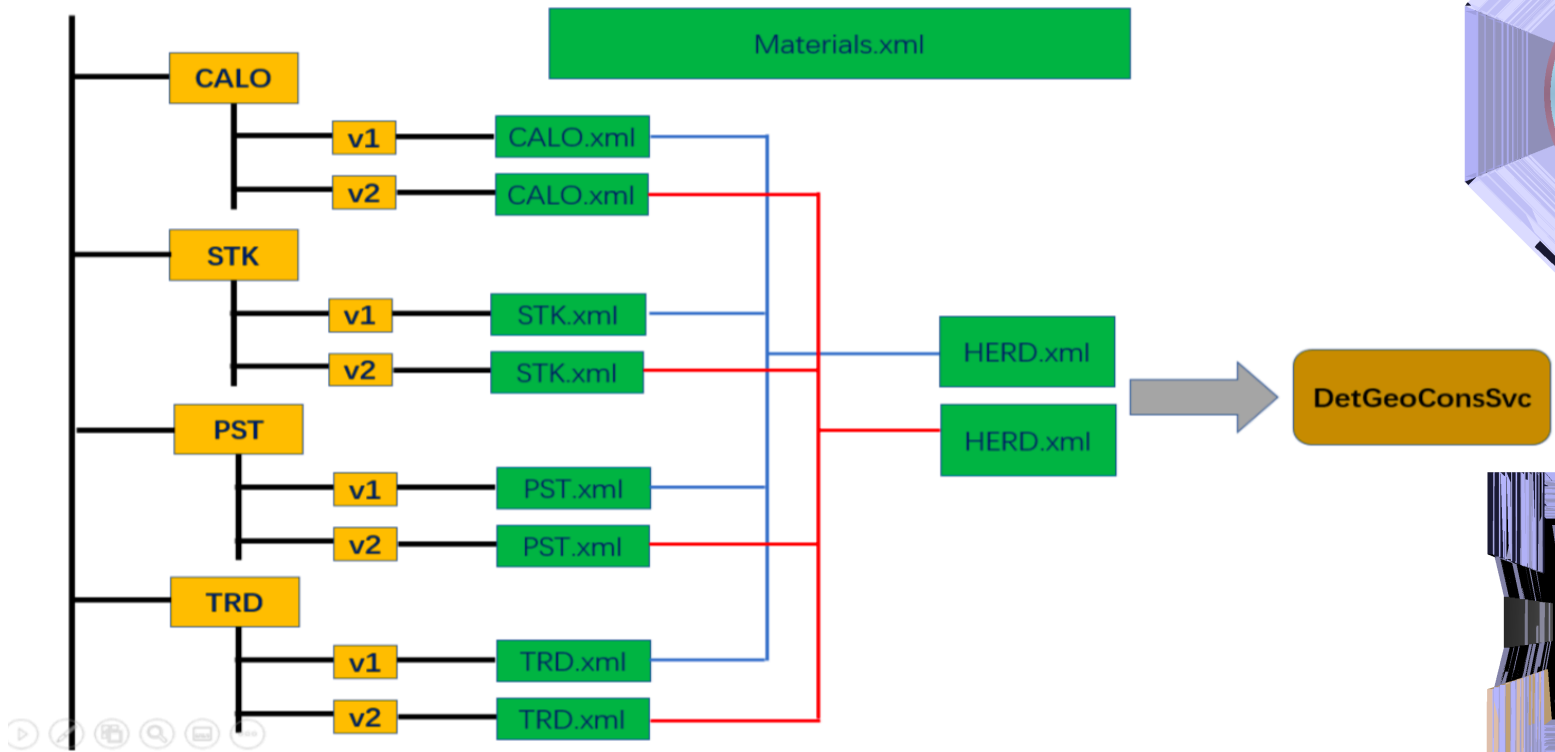
```
-bash-4.1$ ls
AirTube_geo.cpp     DIRC_geo.cpp                           SCTube_geo.cpp     Tracker_geo.cpp
BarrekDIRC_geo.cpp  InnerPlanarTracker_geo.cpp             STCF_BEMC_geo.cpp  TrackerSupport_geo.cpp
detectorMUD.cpp     PolyhedraEndcapCalorimeter2_geo.cpp    STCF_EEMC_geo.cpp  ZPlanarTracker_geo.cpp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

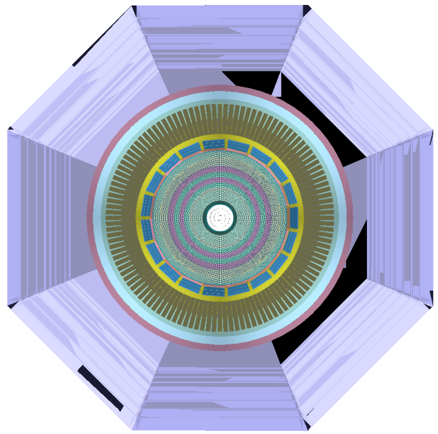
- Deliver detector geometry to Geant4

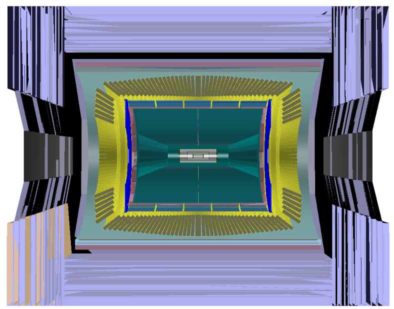
```
import DetGeoConsSvc
myxmlsvc = task.createSvc("DetGeoConsSvc")
myxmlsvc.property("DetGeoConsSvcEnable").set(1)
myxmlsvc.property("GeoCompactFileName").set("/afs/ihep.ac.cn/soft
install/examples/ClientTests/compact/detectorSC.xml")
```

DD4hep example :Geometry management

- 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



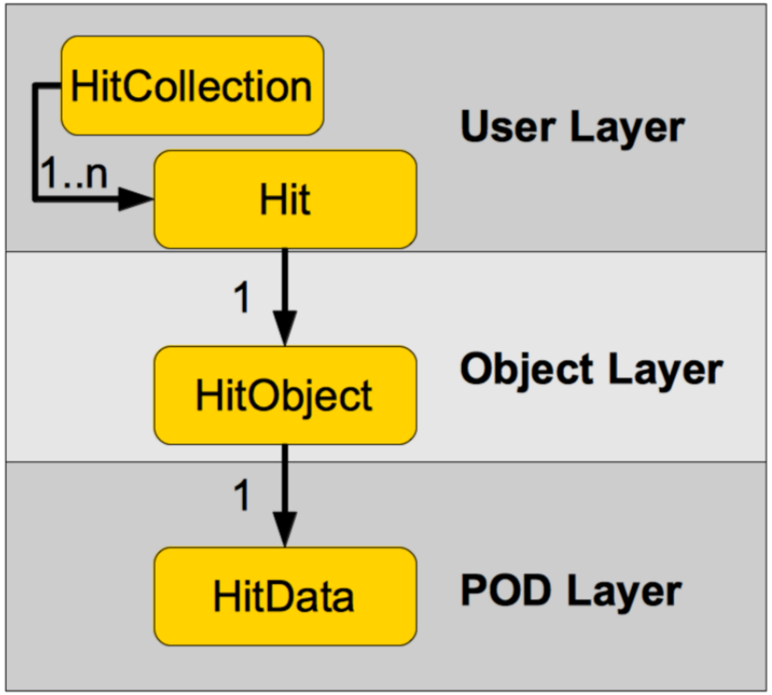




PODIO: an Event Data Model toolkit for HEP

- Based on the use of POD (Plain-Old-Data) for the event data
- Developed in AIDA2020 and originally for FCC study, but potentially to be re-used by other HEP

- user layer (API):
  - handles to EDM objects (e.g. **Hit**)
  - collections of EDM object handles (e.g. **HitCollection**).
- object layer
  - transient objects (e.g. **HitObject**) handling *references* to other objects and *vector members*
- POD layer
  - the actual POD data structures holding the persistent information (e.g. **HitData**)



direct access to POD also possible - if needed for performance reason

F. Gaede (CHEP2019)

Core Features of PODIO I

- clear design of **ownership** (hard to make mistakes) in two stages:
  - objects added to event store are *owned by event store*
  - objects created stand-alone are *reference counted* and automatically garbage collected:
- allow to have *1-1*, *1-N* or *N-M* **relationships**
  - referenced objects can be accessed via iterator or directly
  - also stand-alone relations between arbitrary EDM objects

```
# LCIO MCParticle
  MCParticle:
    Description: "LCIO MC Particle"
    Author : "F.Gaede, B. Hegner"
    Members:
      - int pDG                 // PDG code of the particle
      - int generatorStatus     // status as defined by the gene
      - int simulatorStatus     // status from the simulation
        #...
    OneToManyRelations:
      - MCParticle parents // The parents of this particle.
      - MCParticle daughters // The daughters this particle.
    ExtraCode:
      const_declaration:
      "bool isCreatedInSimulation() const {
          return simulatorStatus() != 0 ;
       } \n"
```

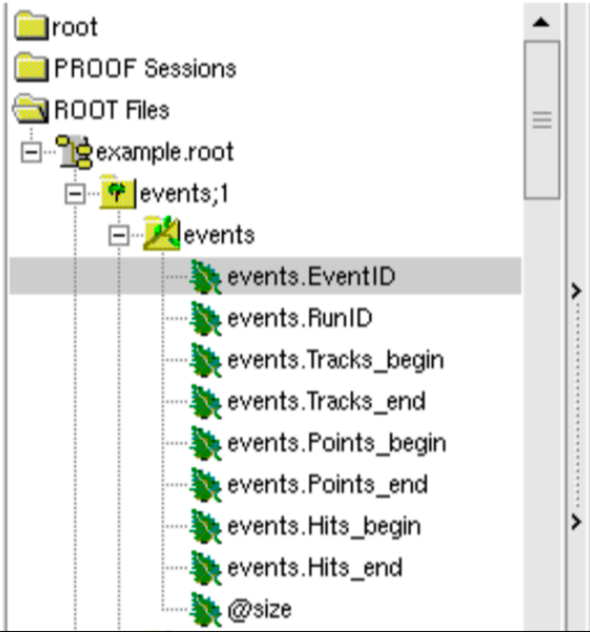
- **code generation** (C++/Python) for EDM classes from *yaml* files
  - EDM objects (data structures) are built from basic types and components
  - additional user code (member functions) can be defined in the yaml files

Integration of PODIO into SNiPER

- Integrated PODIO into SNiPER
  - Define Event Data in the yaml file
  - Python script for c++ code generation
  - Algorithm uses the Event Data Object
  - PODIO writer writes the Event Data into the ROOT file

Yaml file → C++ codes → Root File

```yaml
datatypes :
# Datatypes are components that can be stored in a Collection
  STCF::MDCTrack:
    Description : "MDC Track"
    Author : "Q. Y. LI"
    Members :
     - int                  PdgCode                  // pdgcode
     - int                  MotherID                 // motherid
     - int                  GeneratorFlags           // gflags
     - STCF::LorentzVector FourMomentum             // fourmomentum
    OneToOneRelations:
     - STCF::Vertex        StartVertex              // Start track vertex [cm, ns]
     - STCF::Vertex        StopVertex               // Stop track vertex [cm, ns]
    OneToManyRelations:
     - STCF::MDCHit        Hits                     // contains many hits
    ExtraCode :
      declaration: "
      /// operator to allow pointer like calling of members    \n
      {name}* operator->() { return ({name}*) this ; }  \n
      "
```



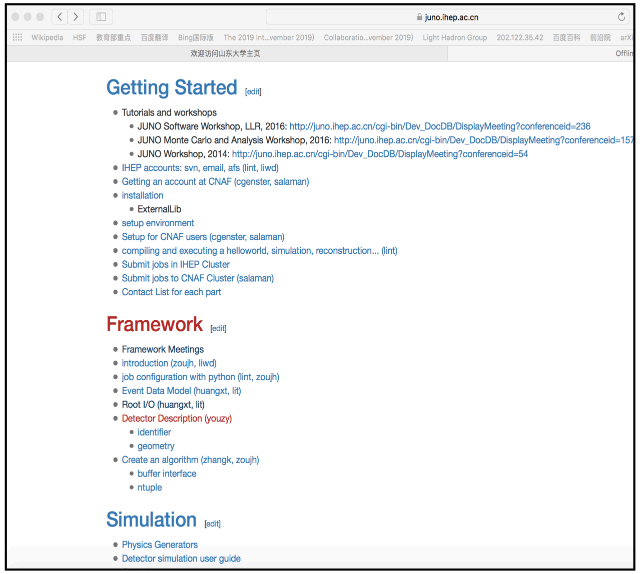

```cpp
class MDCTrack {

  friend MDCTrackCollection;
  friend MDCTrackCollectionIterator;
  friend ConstMDCTrack;

public:

  /// default constructor
  MDCTrack();
  MDCTrack(int PdgCode,int MotherID,int GeneratorFlags,STCF::LorentzVector FourMomentum)
;

  /// constructor from existing MDCTrackObj
  MDCTrack(MDCTrackObj* obj);
```

Software Environments and Management

- Programming language: C++ and Python
  - C++ : main part implementation
  - Python : job configuration interface
- Packages management tool: CMake
  - Help developers to compile packages easily
  - Help users to setup the environment for running the application easily
- Operation System: Scientific Linux
  - Scientific Linux 6/CentOS 7 or higher
  - G++ > 6.5.0 (C++14)
- Version Control System: GitLab
  - Keep the history of code evolution
  - Synchronization and sharing between developers
  - Tag and release

Installation and Documentation

- Installation
  - A shell script is provided to Automatically install the whole offline software
- Documentation
  - JUNO User Guider Wiki page
  - LodeStar User Guider for LHAASO Experiment
  - OSCAR User Guider for STCF Experiment



juno.ihep.ac.cn

Getting Started [edit]

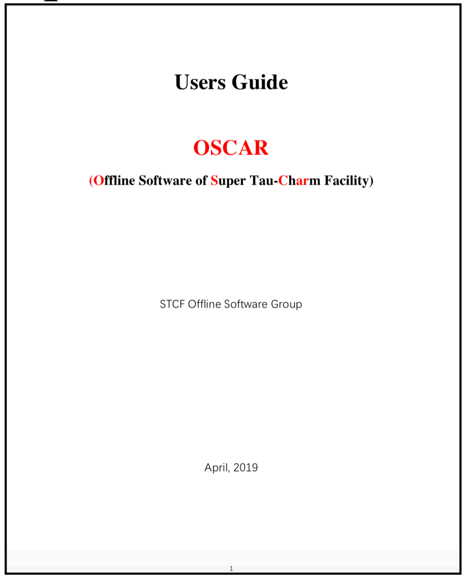
- Tutorials and workshops
  - JUNO Software Workshop, LLR, 2016: http://juno.ihep.ac.cn/cgi-bin/Dev_DocDB/DisplayMeeting?conferenceid=236
  - JUNO Monte Carlo and Analysis Workshop, 2016: http://juno.ihep.ac.cn/cgi-bin/Dev_DocDB/DisplayMeeting?conferenceid=157
  - JUNO Workshop, 2014: http://juno.ihep.ac.cn/cgi-bin/Dev_DocDB/DisplayMeeting?conferenceid=54
- IHEP accounts: svn, email, afs (lint, liwd)
- Getting an account at CNAF (cgenster, salaman)
- installation
  - ExternalLib
- setup environment
- Setup for CNAF users (cgenster, salaman)
- compiling and executing a helloworld, simulation, reconstruction... (lint)
- Submit jobs in IHEP Cluster
- Submit jobs to CNAF Cluster (salaman)
- Contact List for each part

Framework [edit]

- Framework Meetings
- introduction (zoujh, liwd)
- job configuration with python (lint, zoujh)
- Event Data Model (huangxt, lit)
- Root I/O (huangxt, lit)
- Detector Description (youzy)
  - identifier
  - geometry
- Create an algorithm (zhangk, zoujh)
  - buffer interface
  - ntuple

Simulation [edit]

- Physics Generators
- Detector simulation user guide



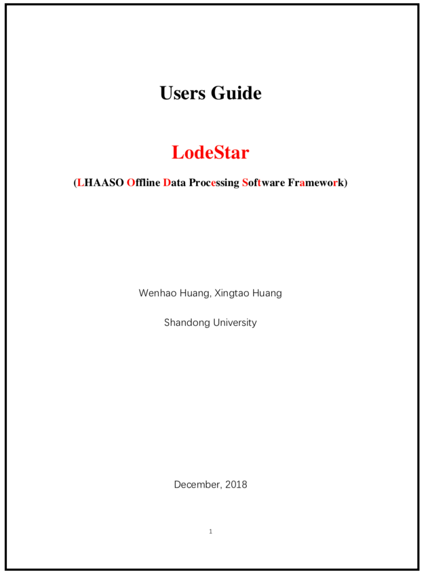
Users Guide

OSCAR

(Offline Software of Super Tau-Charm Facility)

STCF Offline Software Group

April, 2019



Users Guide

LodeStar

(LHAASO Offline Data Processing Software Framework)

Wenhao Huang, Xingtao Huang

Shandong University

December, 2018

Summary

- SNiPER is originally developed for JUNO, also used by LHAASO,STCF,CSNS, nEXO…
  - Main functions for data processing have been implemented
- MT-SNiPER is developed to support Multithreading
  - JUNO Detector simulation works well
- Some promising toolkits such as DD4hep, PODIO have been integrated with SNiPER
  - Describe detector geometry with DD4hep
  - Define Event Data Model with PODIO
- Most popular tools/compiler have been used
  - Cmake, Gitlab, C++14
- Installation toolkits and documentations also provided

Thanks for your attention !

Thanks to members of the Working Group:

Wenhao Huang[1], Xingtao Huang[1], Qiyun Li[1], Weidong Li[2], Tao Lin[2], Xueyao Zhang[1], Jiaheng Zou[2]

[1]SDU , [2]IHEP