Status of LodeStar

Xingtao Huang

Shandong University

2017.9.22

Outline

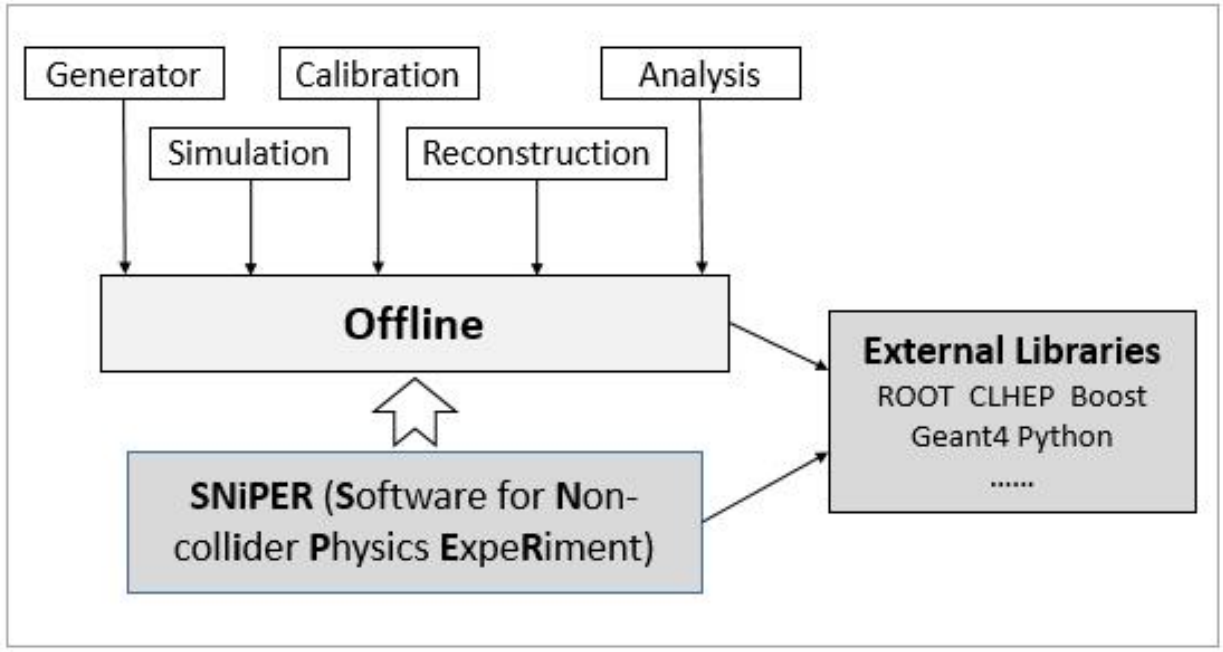
- Main Functionalities of LodeStar Framework
- Important Services
- Several applications of LodeStar
- Summary
- Planning

Main Functionalities of LodeStar

- Providing the common and unified offline data processing environment and platform
  - Software Installation, compiling, integration, release
  - Management of the data processing procedures
    - Process/Sequence management
  - Management of the event data
    - Definition of Event Data Model: Objects and Relations
    - Format of Event Data Model in Memory and in Files
    - Interfaces to put/get data from Memory
    - Read/write data from/to Files
  - Providing common services or tools during processing
    - Message Service, Histogram Service , Random Number Service,
    - Database Interface Service, Geometry Service, etc.
  - Providing friendly user interfaces
    - Easy to develop user code
    - Configure Jobs

Overview of LHAASO Offline Software

- LodeStar(北极星):
  - **L**HAASO **O**ffline **D**ata Proc**e**ssing **S**oft**t**ware Fra**m**ewo**r**k
- Constituents of LodeStar:
  - SNiPER: underlying framework
  - Offline: specific to LHAASO Experiments
  - External Libraries: frequently used third-party software or tools



Programming Language and Operator System

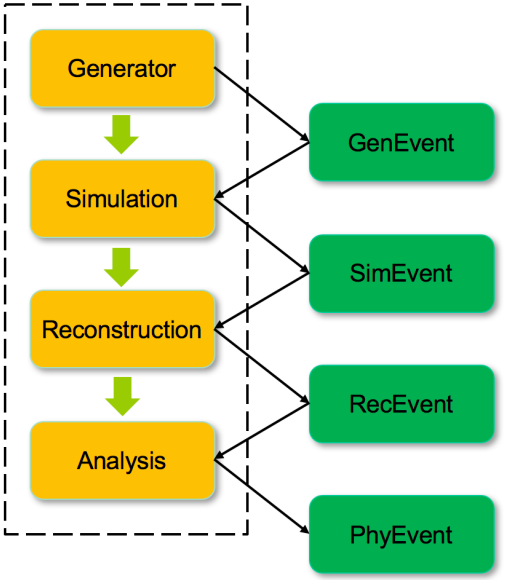
- Programming Language
  - C++,Python
  - very popular in HEP
  - most frequently used software implemented in C++
- Configuration Language：Python
  - Very flexible
  - configure jobs without re-compiling software
- Supported Operator System
  - Scientific Linux 5, 6 are currently supported
  - More OS will be tested and supported according to needs

Software Development Management

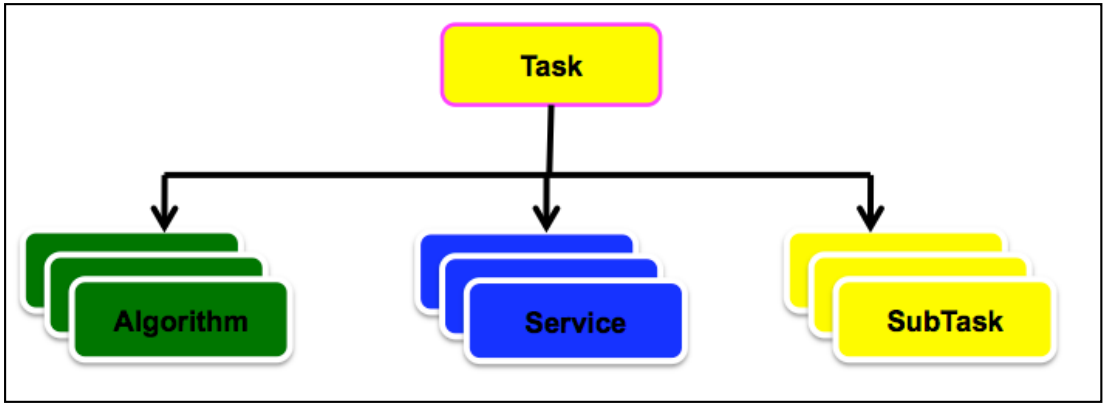
- Software Management Tool：CMT
  - Automatically compile, build packages
  - Automatically deal with relationships between package
- Version Control Tool：SVN
  - keep history of codes developing
  - synchronization and sharing between developers
  - server: https://svn.hepg.sdu.edu.cn/repos/lhaaso/

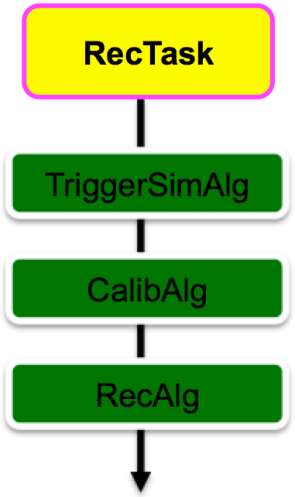
Data Processing Controlling

- Data Processing workflow:
  - Algorithm: basic unit of data processing
  - Separation between algorithms and data
  - Service: a piece of codes for common use
  - Task: manager of algorithms, services and tasks



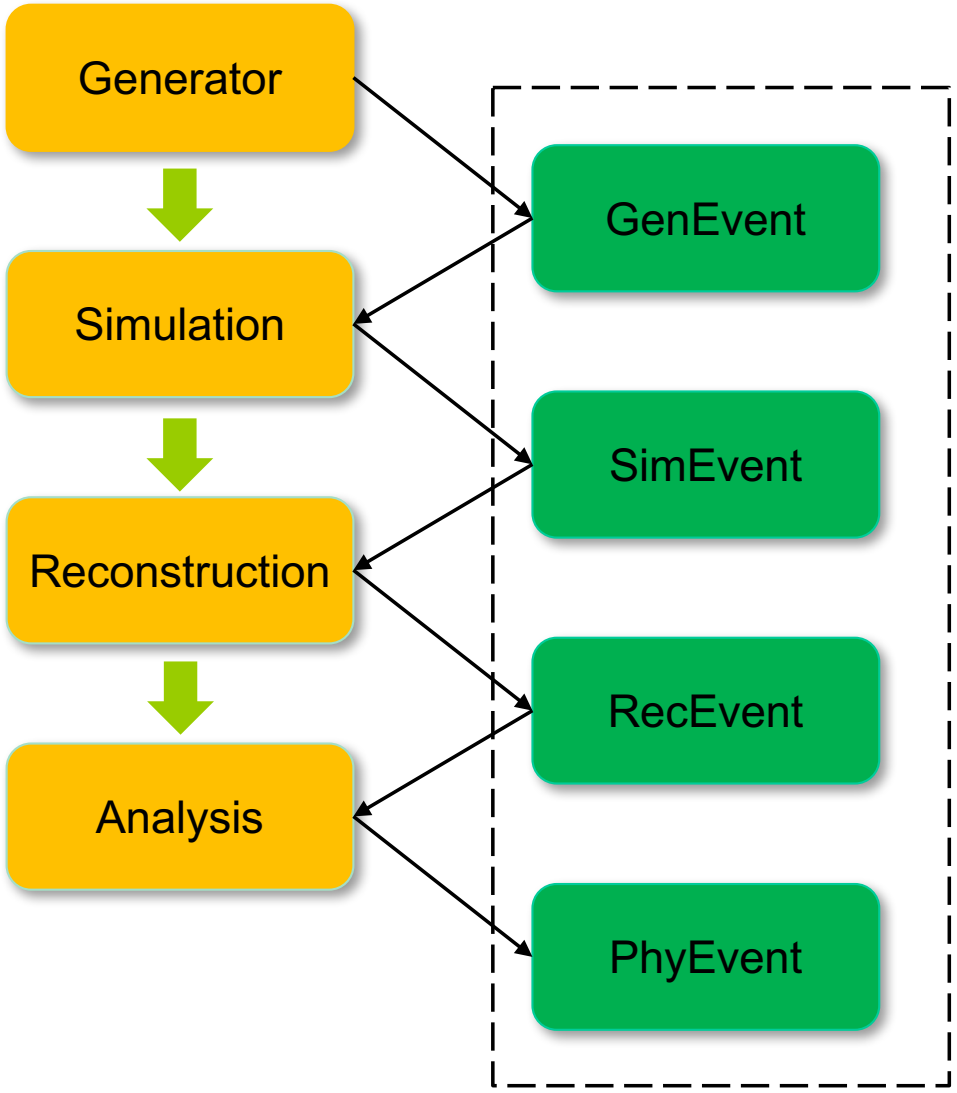
- Users /developers only need write specific algorithms.





Event Data Management

- The event information is defined in the Event Data Model objects and stored in the Data Store with its unique path

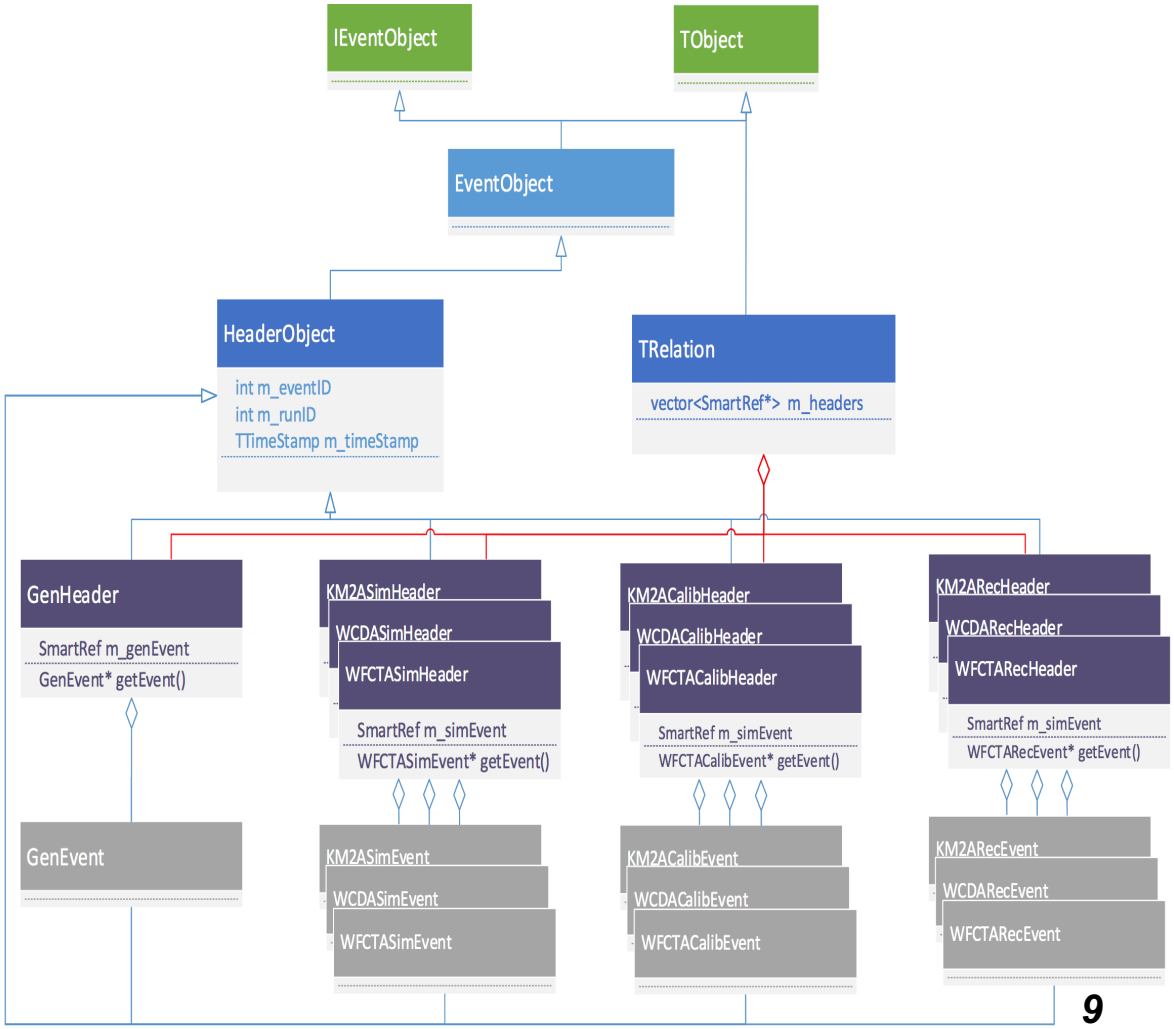


/Event/GenEvent
/Event/SimEvent
/Event/RecEvent
/Event/PhyEvent

Event Data Model

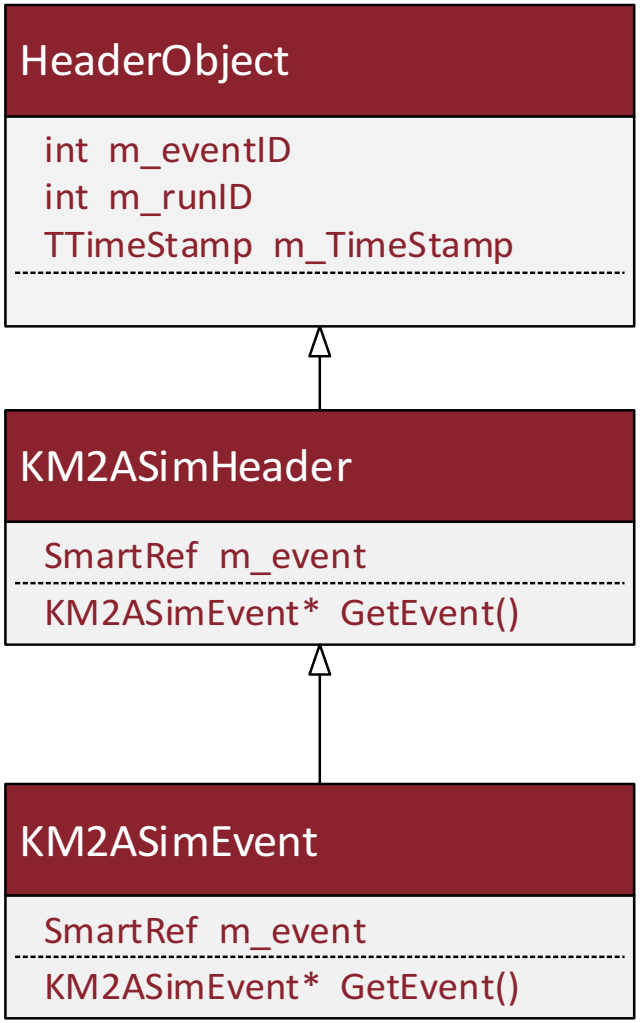
Design of LHAASO Event Data Model

- Each process defines their Event Objects
- Event data for different sub detectors are separated
- EventObject inherits from TObject.
- TRelations: matching between Headers (optional)



Event Data Model

- Each event data consists of two parts:
  - HeaderObject
  - EventObject
  - Separate meta data from event data
- Some “tag” data can be put into header
  - Speed up event selection
- SmartRef: a new type of smart pointer
  - Support correlation of data objects
  - Support lazy-loading

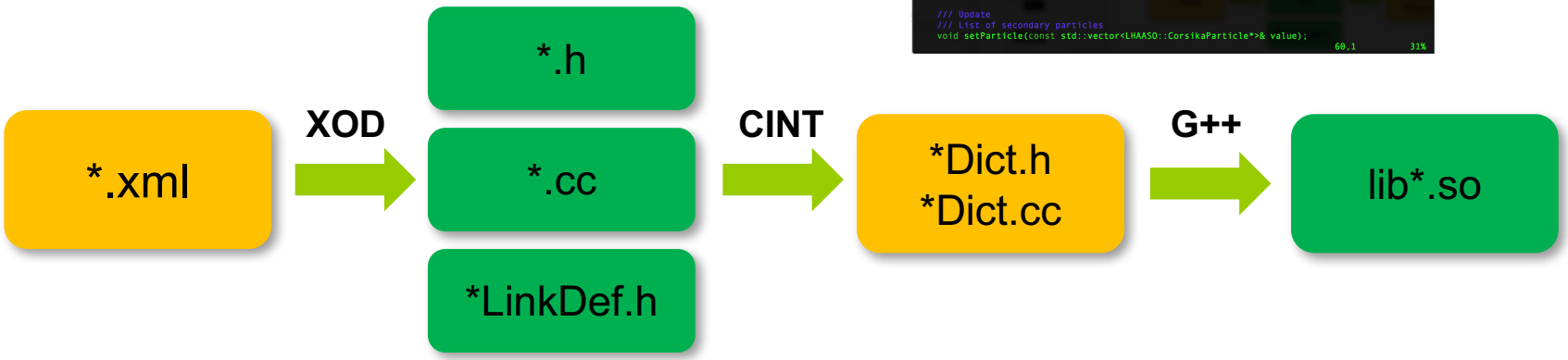


XmlObjDesc: the Tool to define DataModel with XML

- Traditionally writing C++ Code by hand
  - Many repeatable work such as Getters and Setters
  - Difficult to be maintained
- New way to define EDM with XML file
  - Strong syntax (DTD, XML Schema)
  - More readable, easier to maintain
  - Automatically generate the Get-, Set- functions, Streamers
- XmlObjDesc (XOD) is used as a tool to define EDM with XML

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

<xdd>
    <package name = "CorsikaEvent">

        <import name="Event/HeaderObject"/>
        <import name="Event/CorsikaEvent"/>

        <class name="CorsikaHeader"
               author="LI Teng"
               desc="Header Class for Corsika input">

            <base name="HeaderObject"/>
            <SmartRelation type="LHAASO::CorsikaEvent"
                           name="event"
                           desc="Smart pointer to the CorsicaEvent"
                           nonconstaccessor="TRUE"/>

        </class>
    </package>
</xdd>
```

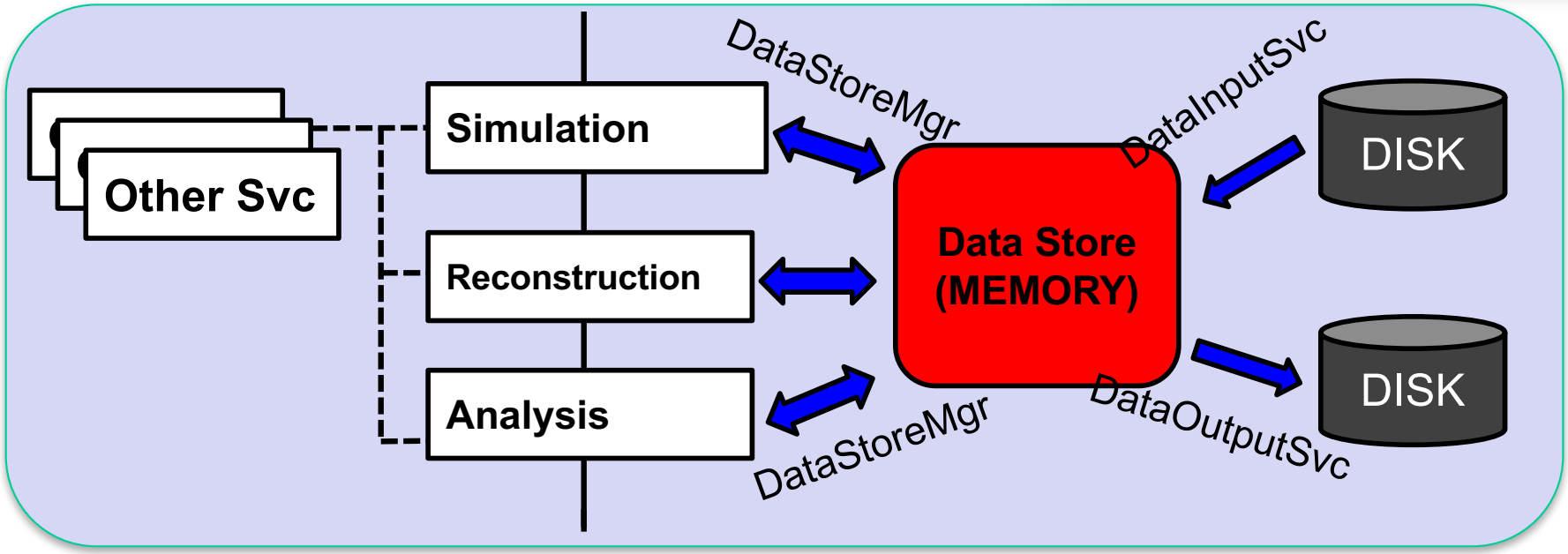
```
class CorsikaEvent: public EventObject
{
private:

  std::vector<LHAASO::CorsikaParticle*>  m_particle;  // List of secondary particles
  std::vector<LHAASO::CorsikaChePhoton*> m_chePhoton; // List of Cherenkov photons


protected:


public:

  /// Default Constructor
  CorsikaEvent() : m_particle(),
                   m_chePhoton() {}

/// destructor
~CorsikaEvent();

  /// add a CorsikaParticle
  void addParticle(LHAASO::CorsikaParticle* value);

  /// add a Cherenkov photon
  void addChePhoton(LHAASO::CorsikaChePhoton* value);

  /// Retrieve const
  /// List of secondary particles
  const std::vector<LHAASO::CorsikaParticle*>& particle() const;

  /// Retrieve
  /// List of secondary particles
  std::vector<LHAASO::CorsikaParticle*>& particle();

  /// Update
  /// List of secondary particles
  void setParticle(const std::vector<LHAASO::CorsikaParticle*>& value);
                                                                60,1          31%
```



Event Data Management

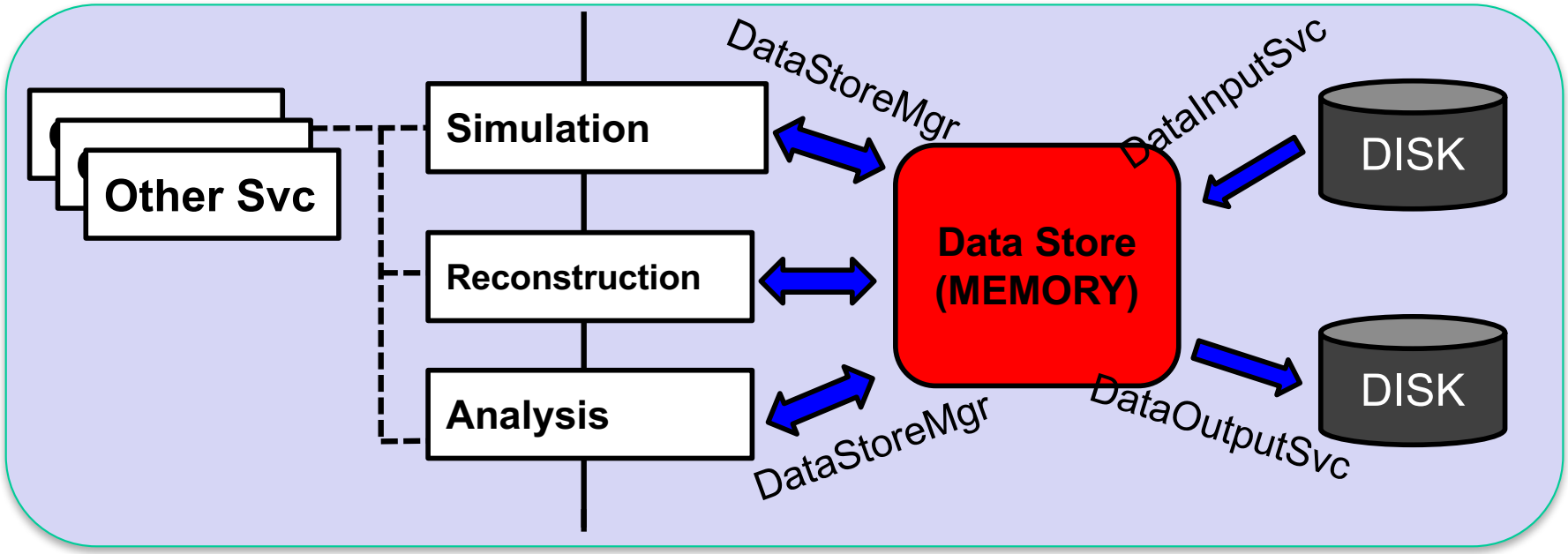
- Event Data Management in Lodestar:
  - Event data is stored in the Data Store, managed by DataStoreMgr
  - Data Store is a dynamically allocated place, shared by all the algorithms
  - All algorithms/services/tools can retrieve/put event data from/to the Data Store with the unique paths
  - Data Store is automatically configured with Data I/O Service

```cpp
#include "DataStoreMgr/EvtDataPtr.h"
#include "Event/CorsikaHeader.h"
#include "Event/KM2ASimEvent.h"

using namespace LHAASO;

bool HelloAlg::execute()
{
    ++m_count;
    LogInfo << "Hello world: count: " << m_count << std::endl;

    // Get CorsikaEvent from DataStore
    EvtDataPtr<CorsikaHeader> edp("/Event/CorsikaEvent");
    CorsikaHeader* header = edp.data();
    CorsikaEvent* event = header->event();

    // Particle list:
    std::vector<CorsikaParticle*> particles = event->particle();

    // Create SimEvent and put it into DataStore
    KM2ASimHeader* simHeader = new KM2ASimHeader;
    KM2ASimEvent* simEvent = new KM2ASimEvent;
    simHeader->setEvent(simEvent);
    SniperPtr<IDataStoreMgr> mMgr(getScope(), "DataBufferMgr");
    mMgr->adopt(simHeader, "/Event");

    return true;
}
```



Data I/O Service

- Event Data Management in LodeStar:
  - Data Input/Output is implemented with Data I/O Services:
  - Responsible for reading/ writing event data from/to files.
  - Currently support:
    - Root Input/Output streams
    - Corsika Input stream

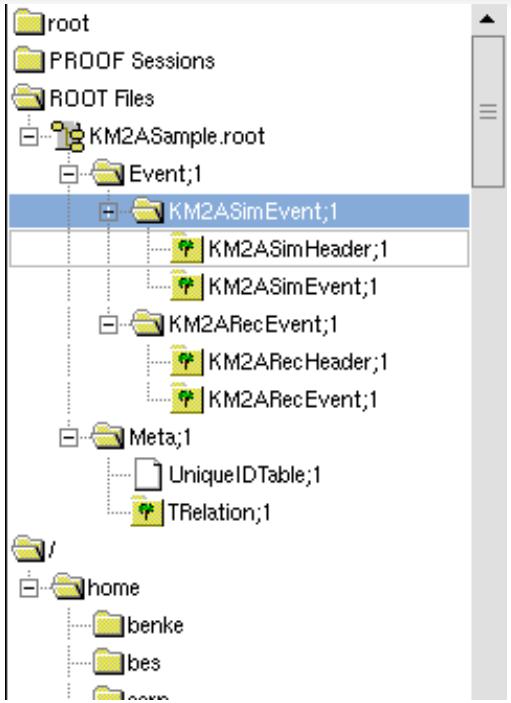


Data I/O Service

- CorsikaInputSvc
  - Convert the raw corsika file to the DataModel objects
  - Support different types of input files (particles, Cherenkov photons, Longitudinal parameters)
  - Support event-splitting

```
8  import DataStoreMgr
9  task.createSvc("DataStoreMgr")
10
11 import DataIOSvc
12
13 iSvc = task.createSvc("DataInputSvc/InputSvc")
14 iSvc.property("InputStream").set(\
15 {"/Event/CorsikaEvent" : "DAT050001.part"})
16
17 oSvc = task.createSvc("DataOutputSvc/OutputSvc")
18 oSvc.property("OutputStream").set(\
19 {"/Event/KM2ASimEvent" : "SimEvent.root"})
```

- RootInputSvc/RootOutputSvc:
  - Writing/Reading official ROOT data files
  - Support lazy-loading mechanism



Property Setting Service

- Provides an efficient way to dynamically set parameters/variables in your algorithm
- Supported Types
  - ✓ scalar: C++ build in types and std::string
  - ✓ std::vector of scalar type
  - ✓ std::map with scalar key type and scalar value map

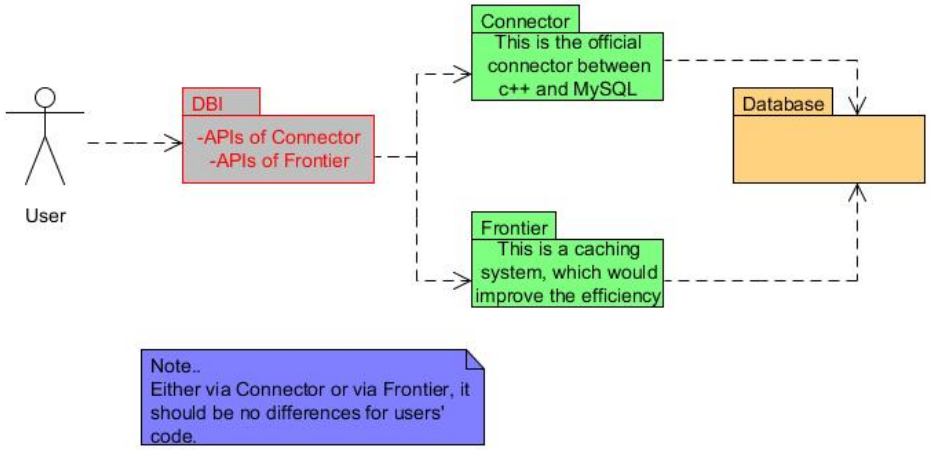
```
std::string                    m_string;
std::vector<int>               m_vector_int;
std::map<std::string, int>     m_str_int;
```

```
declProp("VarString", m_string);
declProp("VectorInt", m_vector_int);
declProp("MapStrInt", m_str_int);
```

```
10  import HelloWorld
11  alg = task.createAlg("Hello/hAlg")
12  alg.property("VarString").set("Some value")
```

Database Service

- Data types to manage
  - Calibration constants
  - Geometry /PMT position
  - Optical parameters
- Underlying databases to be used:
  - MySQL(already used)
  - SQLite(alternative)
  - NoSQL(alternative)
- Functionalities
  - Automatically write data into Database
  - Easily access to the data from Database in your algorithms
- Two ways are under developments




```
mysql> desc MaterialProperty;
+---------+--------------+------+-----+---------+-------+
| Field   | Type         | Null | Key | Default | Extra |
+---------+--------------+------+-----+---------+-------+
| sftVer  | varchar(16)  | YES  |     | NULL    |       |
| name    | varchar(256) | YES  |     | NULL    |       |
| keyId   | int(11)      | YES  |     | NULL    |       |
| notes   | varchar(256) | YES  |     | NULL    |       |
+---------+--------------+------+-----+---------+-------+
```

```
//this one is to test lookupquery function
DBIRequest request("select * from offline_db.test");

DBIResultPtr dbptr("DatabaseSvc", request);
dbptr.Session();
//dbsvc.Session(request);
int row = dbptr.GetMaxRowcount();
for(int i = row; i >= 0; i--)
{
dbptr.GetResByRowNum(i);
std::cout << "id= " << dbptr.GetInt("id") << std::endl;
//std::cout << "LocalName= " << dbptr.GetString("LocalName
//std::cout << "Capital= " << dbptr.GetString("Capital") <
}
```

Logging Service

- The logging service 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;
```

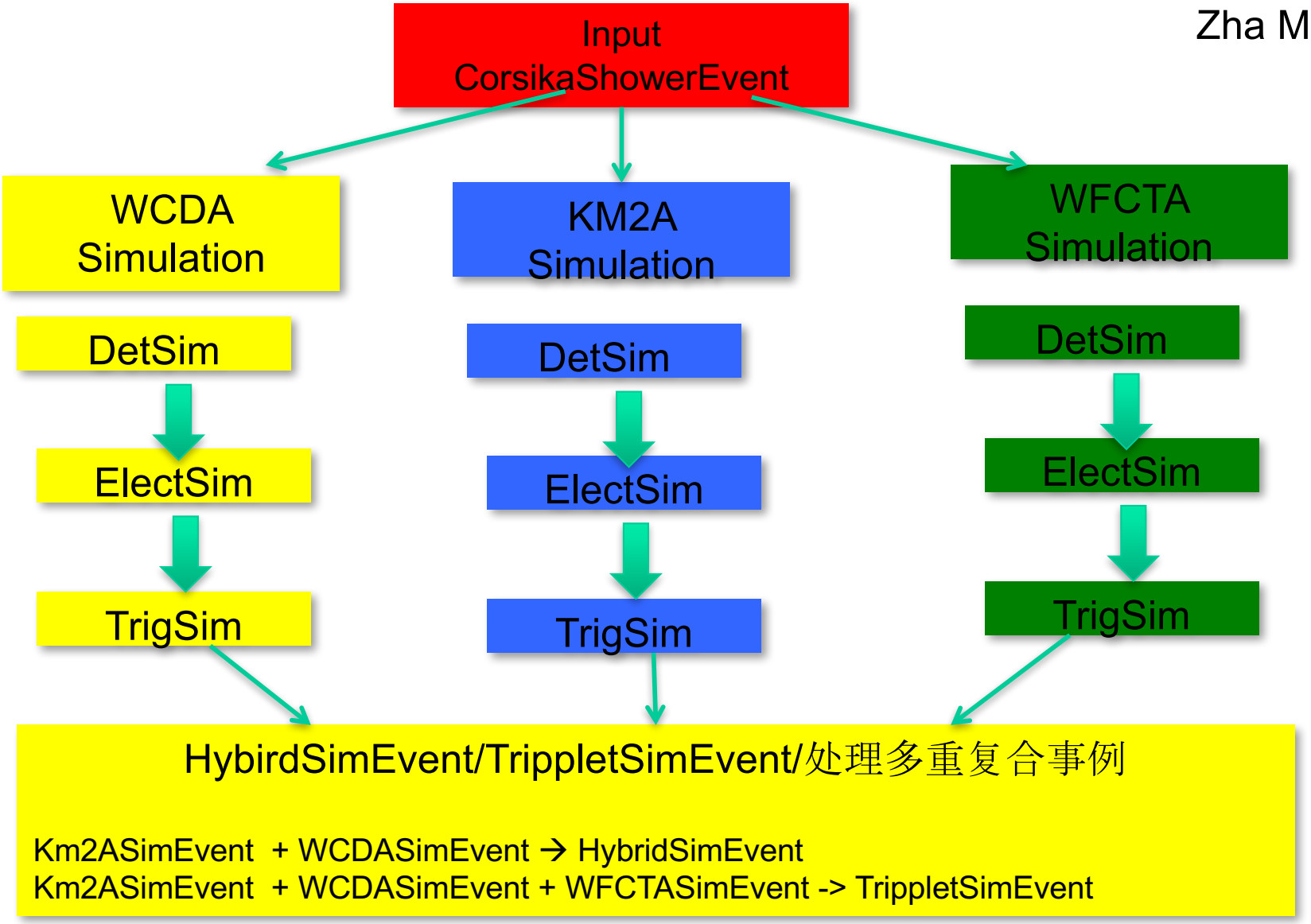
```
aHelloAlg.execute          DEBUG: A debug message
aHelloAlg.execute           INFO: An info message
aHelloAlg.execute          ERROR: An error message
```

- Each DLElement(Task, Algorithm, Service, Tool) has their own LogLevel and can be set at the run time
  - very helpful for code debugging
- The output message includes more information
  - its contents
  - where it happens
  - the message level

```
4  import Sniper
5
6  task = Sniper.Task("task")
7  task.asTop()
8  task.setLogLevel(3)
9
10 Sniper.loadDll("libHelloWorld.so")
11 alg = task.createAlg("Hello/hAlg")
12
13 task.setEvtMax(5)
14 task.show()
15 task.run()
```

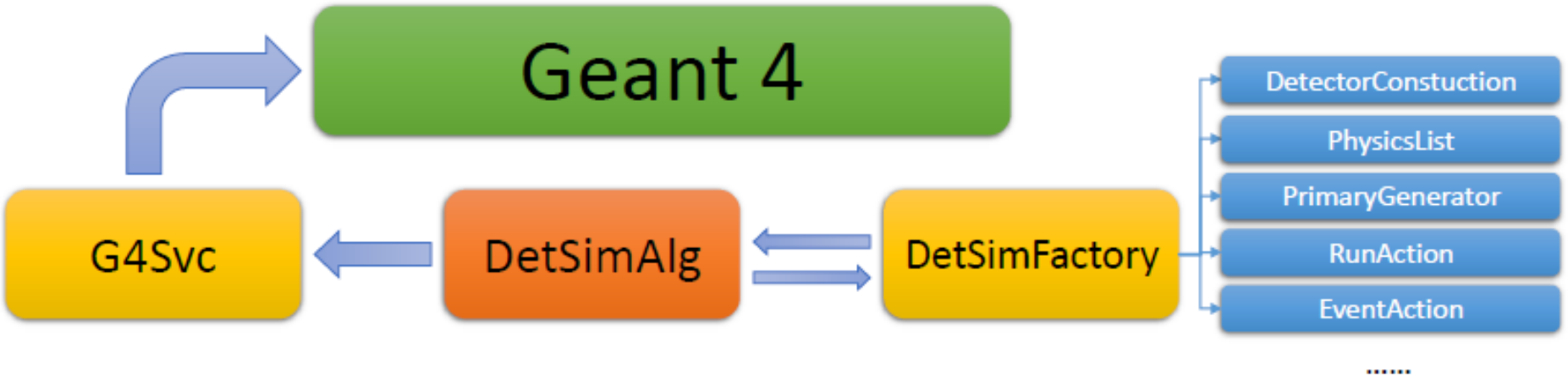
Example: Simulation Chain

Zha Min



Application: G4argo

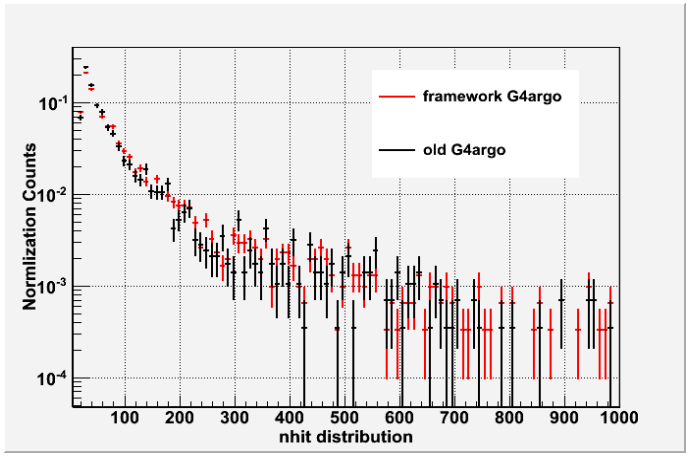
- An example of G4-based detector simulation package
  - DetSimAlg: Common algorithm of Geant4 detector simulation
  - G4Svc: Interface to the Geant4 core
  - ArgoSimFactory: Build all the detector simulation options
    - Generator
    - Geometry
    - User-actions

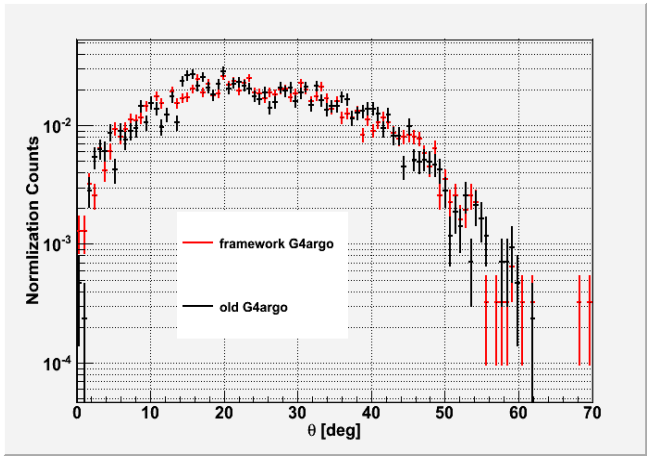


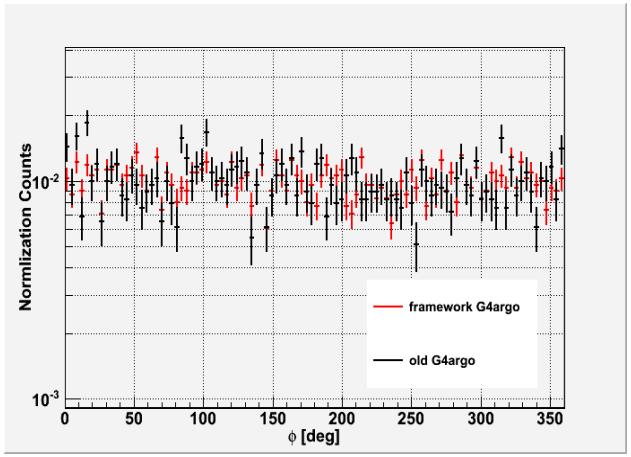
Application: G4argo

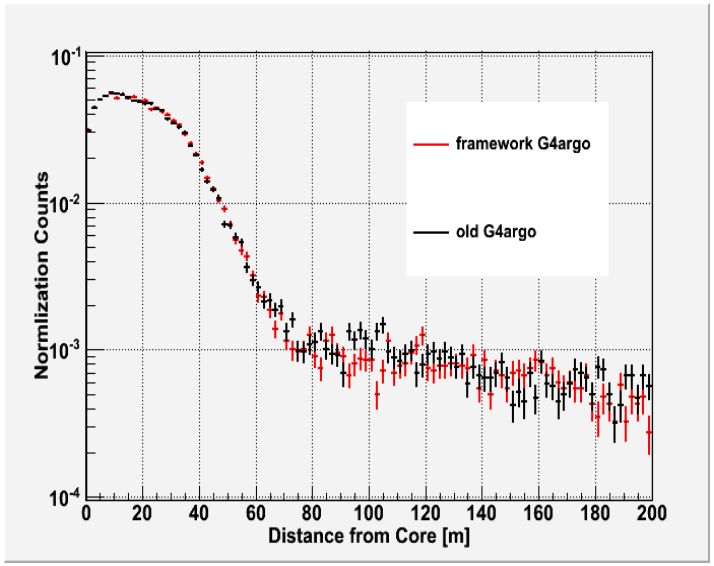
- Some comparison from the origin results of G4argo:

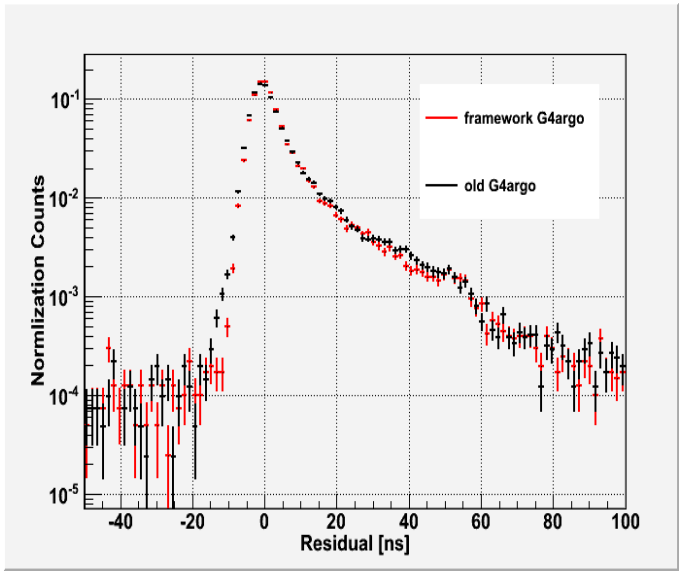
Guo Yiqing,
Li Teng









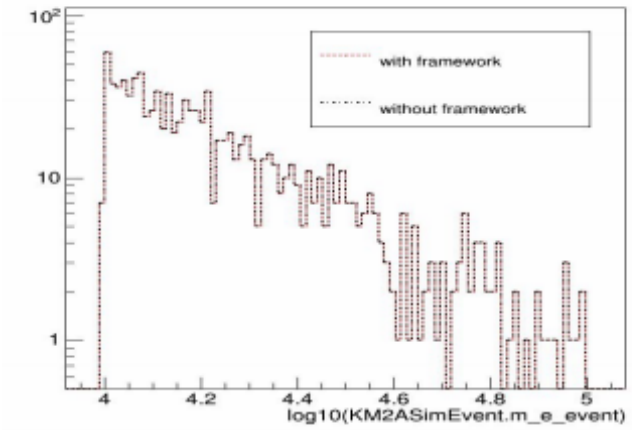


Application：KM2A FastSim

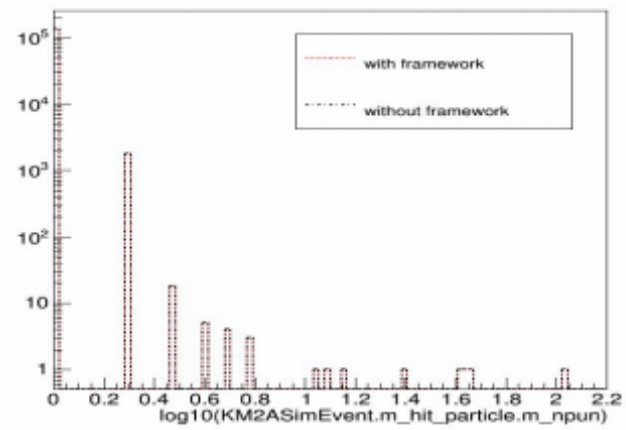
- KM2A fast simulation
  - Implemented with KM2ADetSimAlg and KM2ARecAlg
  - Some comparison of the results:

Liu Ye
Li Teng

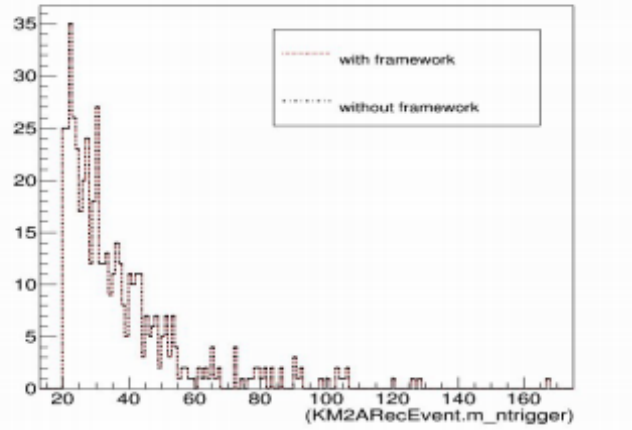
primary event energy



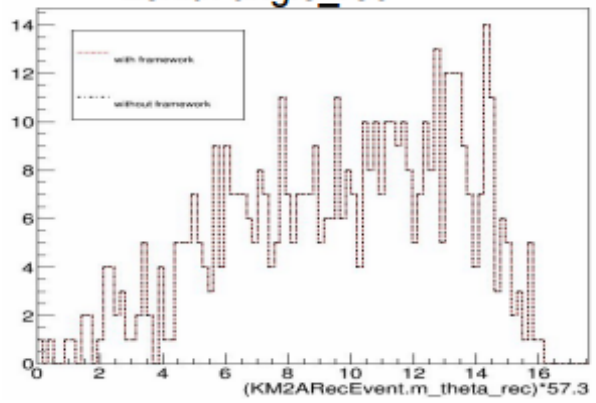
npe produced by particles



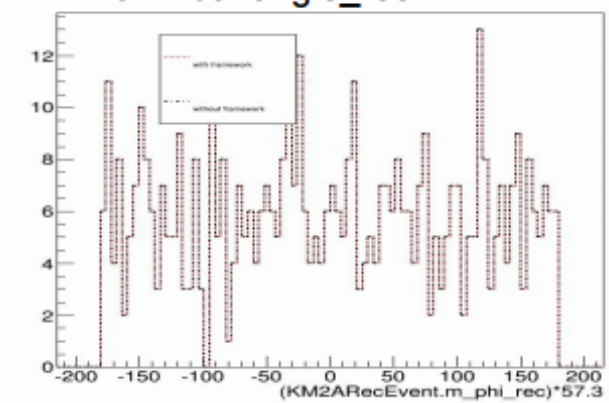
ntrig



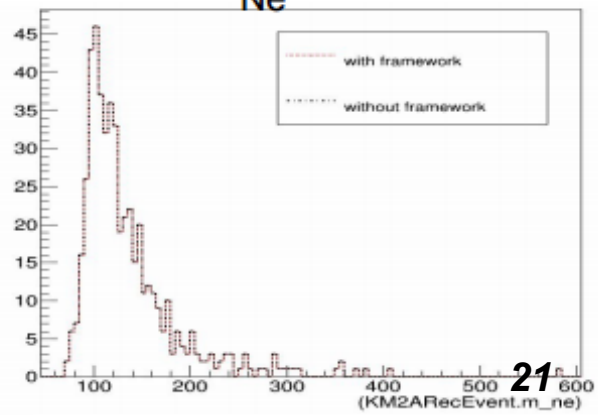
zenith angle_rec



azimuth angle_rec



Ne



Application：WFCTA Simulation

- WFCTA Simulation
  - Implemented with WFCTADetSimAlg
  - Some comparison of the results:

Ma Lingling
Li Teng

	Lodestar	Origin
Corex, corey	30857.314453, 21321.181641	30857.314453, 21321.181641
Zenith, azimuth	38.221312, 4.110079	38.221312, 4.110079
Totalphoton	720363989.469576	720363989.469576
Photons arriving telescope	2204600	2204600
Photon after reflecting	1432315	1432315
Photon after raytracing	220824	220824

Summary

- LHAASO Offline Software System has been Setup
  - Installed at IHEP: /afs/ihep.ac.cn/users/l/lhasoft/slc5/lodestar-dev/
  - A installation tool has been provided : LHAASOENV
- Main Functionalities of Framework has been implemented
- Several Applications works very well under LodeStar
- Several tutorials have been arranged

Planning

- Further improvements of LodeStar needs more considerations and discussions
- More closely work together to setup the chain for the whole offline data processing
  - Move the current applications into LodeStar (What, Who, When)
  - Produce M.C. data in large scale
  - Ready for M.C. tuning with ¼ prototype data
- Two major versions should be released each year
- Advanced functionalities are under implementation:
  - Multi-thread computing based on TBB
- Welcome more people to join us!

Thanks for your attention !

Backup

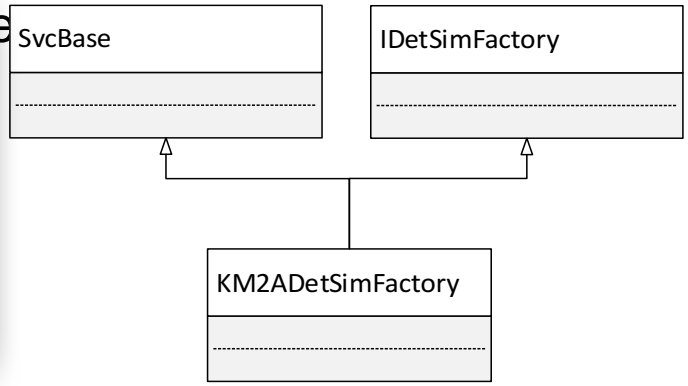
Directory Structure of DetSim

- DetSim is located in lodestar/offline/Simulation/, the sub-directories:
  - DetSimSvc and DetSimAlg: Packages for G4Svc and DetSimAlg
  - DetSimUtil: Some general utilities and base classes
  - AnalysisCode: Some general tools for selecting and saving data
  - ARRAY: User-end second directory
    - LHAASOSim: Some general simulation options
    - KM2ASim: KM2A-related DetSimFactory and options
    - ……

DetSim Factory

- DetSimFactory is A detector factory will construct the detector geometry, physics lists, also the user defined actions.
  - Derived from SvcBase and IDetSimFactory

```
1  class IDetSimFactory {
2    public:
3      virtual ~IDetSimFactory() {};
4      virtual G4VUserDetectorConstruction*   createDetectorConstruction() = 0;
5      virtual G4VUserPhysicsList*            createPhysicsList() = 0;
6      virtual G4VUserPrimaryGeneratorAction* createPrimaryGenerator() = 0;
7      virtual G4UserRunAction*               createRunAction() = 0;
8      virtual G4UserEventAction*             createEventAction() = 0;
9      virtual G4UserStackingAction*          createStackingAction() = 0;
10     virtual G4UserTrackingAction*          createTrackingAction() = 0;
11     virtual G4UserSteppingAction*          createSteppingAction() = 0;
12 };
```



User Actions and IAnalysisElement

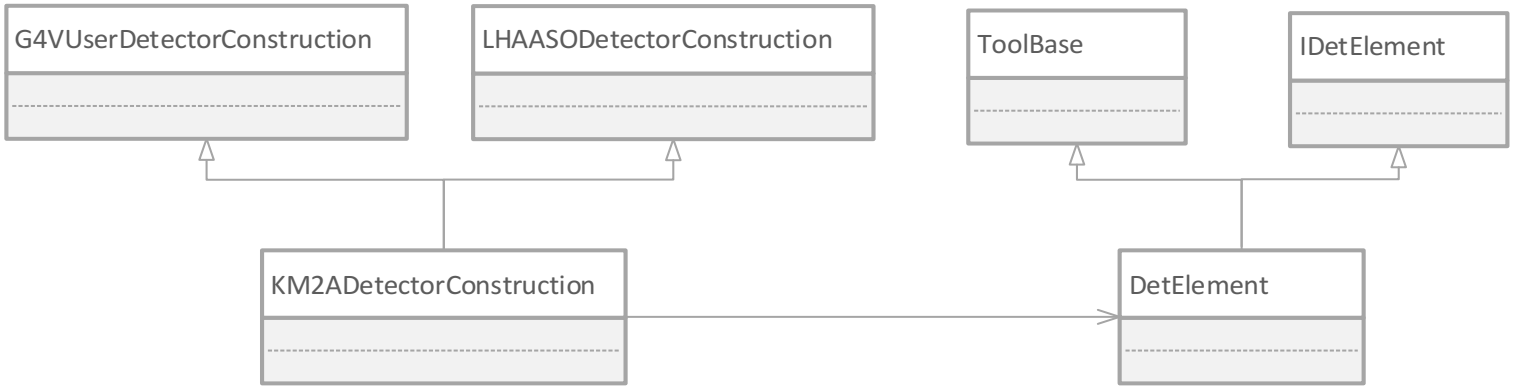
- IAnalysisElement provides interfaces of:

```
25  class IAnalysisElement {
26      public:
27          virtual ~IAnalysisElement() {}
28
29          // Run Action
30          virtual void BeginOfRunAction(const G4Run*) {}
31          virtual void EndOfRunAction(const G4Run*) {}
32          // Event Action
33          virtual void BeginOfEventAction(const G4Event*) {}
34          virtual void EndOfEventAction(const G4Event*) {}
35          // Stacking Action
36          virtual G4ClassificationOfNewTrack ClassifyNewTrack(const G4Track* aTrack) {}
37          virtual void NewStage() {}
38          virtual void PrepareNewEvent() {}
39          // Tracking Action
40          virtual void PreUserTrackingAction(const G4Track*) {}
41          virtual void PostUserTrackingAction(const G4Track*) {}
42          // Stepping Action
43          virtual void UserSteppingAction(const G4Step*) {}
44  };
```

  - Focus one AnalysisElement on just one thing, so that they can be easily re-used.
  - As a tool, AnalysisElement can be configured in Python

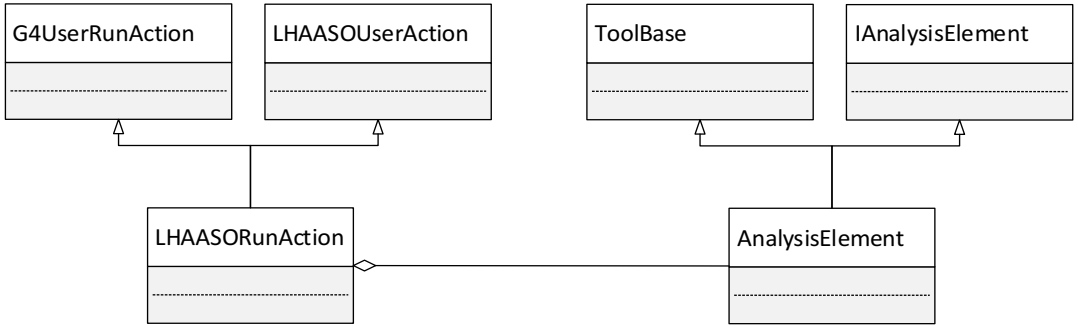
Detector Construction and DetElement

- The detector construction is derived from G4VUserDetectorConstruction and LHAASODetectorConstruction:
  - LHAASODetectorConstruction base class will provide a function which can dynamically load/create a DetElement.
  - DetElement is derived from ToolBase and IDetElement, IDetElement will provide abstract interface to construct the Logical Volume and inject other DetElement
  - The DetElement is very useful when user wants to load the detector elements dynamically.
  - The DetElement can be configured in Python as a tool.



User Actions and IAnalysisElement

- In DetSim/ARRAY/LHAASOSim, we have already provided some default actions:
  - LHAASORunAction, LHAASOEventAction, LHAASOTrackAction, LHAASOSteppingAction
  - Take LHAASORunAction as an example:



  - LHAASORunAction does nothing but loop or AnalysisElement of it:

```
14  void LHAASORunAction::BeginOfRunAction(const G4Run* aRun)
15  {
16      G4cout << "### Run : " << aRun->GetRunID() << " start." << G4endl;
17      for_each(m_ac.begin(), m_ac.end(), bind2nd( mem_fun( &IAnalysisElement::BeginOfRunAction ), aRun));
18  }
19
20  void LHAASORunAction::EndOfRunAction(const G4Run* aRun)
21  {
22      for_each(m_ac.begin(), m_ac.end(), bind2nd( mem_fun( &IAnalysisElement::EndOfRunAction ), aRun));
23  }
```

From Z.G. Yao

Data Volume

Sub-Detectors	Data Volume (MB/s)	Each Year (PB)
KM2A	10	0.32
WCDA	380	12
WFCTA	30	0.10

Joint physics analysis

Correlation	KM2A	WCDA	WFCTA
KM2A	-	weak	no
WCDA	weak	-	no
WFCTA	Strong	weak	-