A Quick Guide to the new CEPC Software Framework

Zou Jiaheng

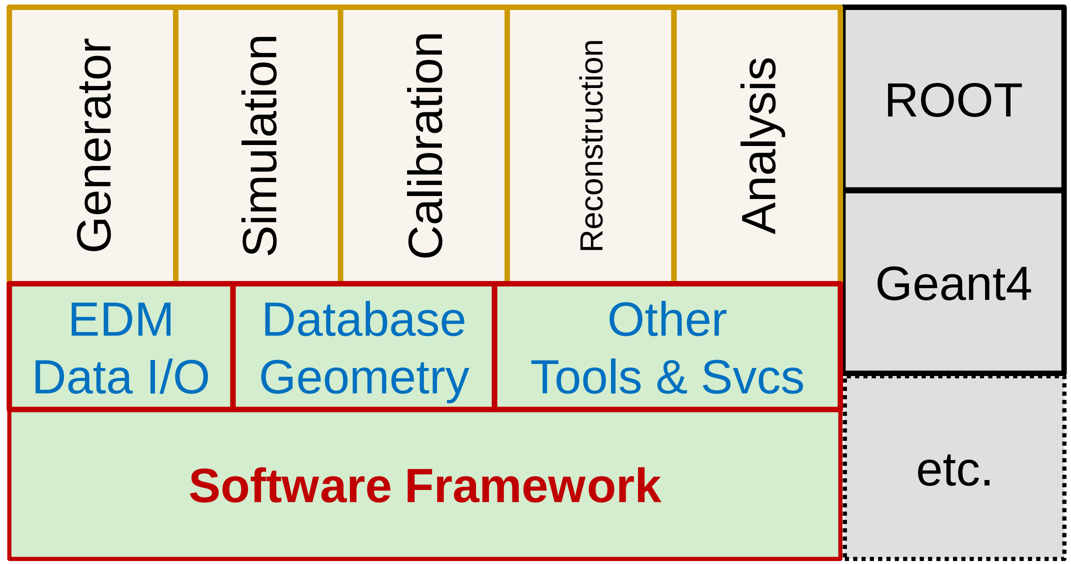
2019-04-01

Content

- **General introduction**
- **Key concepts**
- **Examples**
- **Plan for CEPC**

Offline Software System for HEP

- Application layer
- Basis layer
  - Common services
  - Software framework
- External libraries and tools



The appearance of a system is mainly determined by the framework

- Software architecture, organization, strategy
- Software development standards, user interfaces
- Framework: programming problems
- Physicists: concentrate on calculation algorithms

Considerations on CEPC

- Marlin: the framework at present
  - The developing is not very active now
  - Is not very modernized: hard to support parallel computing…
- Gaudi
  - Very powerful, but very complex and heavy
- A new framework developed by ourselves?
  - Integration with new technologies, such as parallel computing
  - Long Term and Rapid Supporting
  - Feasibility: we have the experience of SNiPER

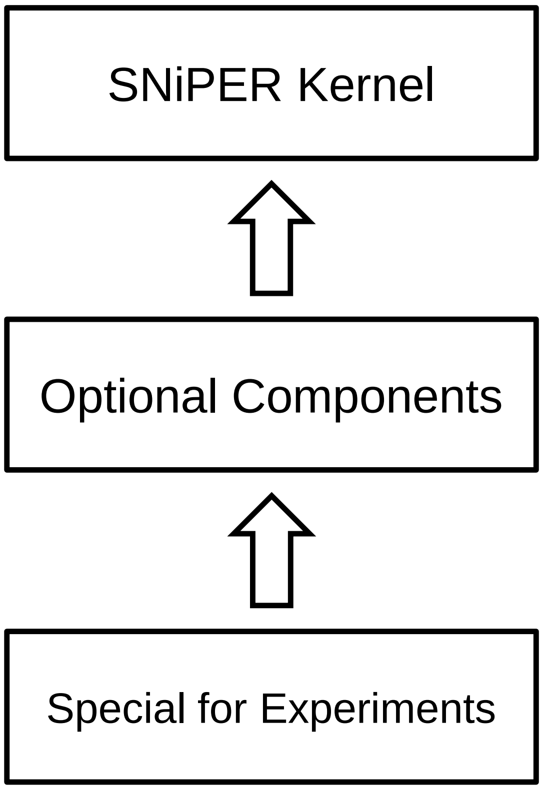
The SNiPER Framework

- Originally Developed for JUNO
  - Fulfill the requirements for neutrino experiments
  - Comprehensive and generality is considered at the beginning
    - A general purpose framework, not only for neutrino experiments
- Functions as a Framework
  - Modularized, extensible, customizable, and friendly to use
  - High performance
- Current Status
  - Has been used in JUNO, LHAASO, CSNS, nEXO
  - Is still being in developing for more application scenarios
  - *https://github.com/SNiPER-Framework/sniper*

Technical Overview of SNiPER

- Hybrid of C++ and Python
  - C++ is used in key functions for better performance
  - Python is flexible: configuration, simple algorithms and services…
  - Binding with Boost.Python
- Lightweight and Simple
  - Less dependencies: the kernel only depends on the boost library
  - Be simple to build, learn and use
  - The key ideology are similar to Gaudi (a lightweight Gaudi)
    - Similar concepts, such as algorithms, services
    - Minimize the cost of migration from Gaudi

Software Based on SNiPER



A compact kernel

The common functions for all experiments

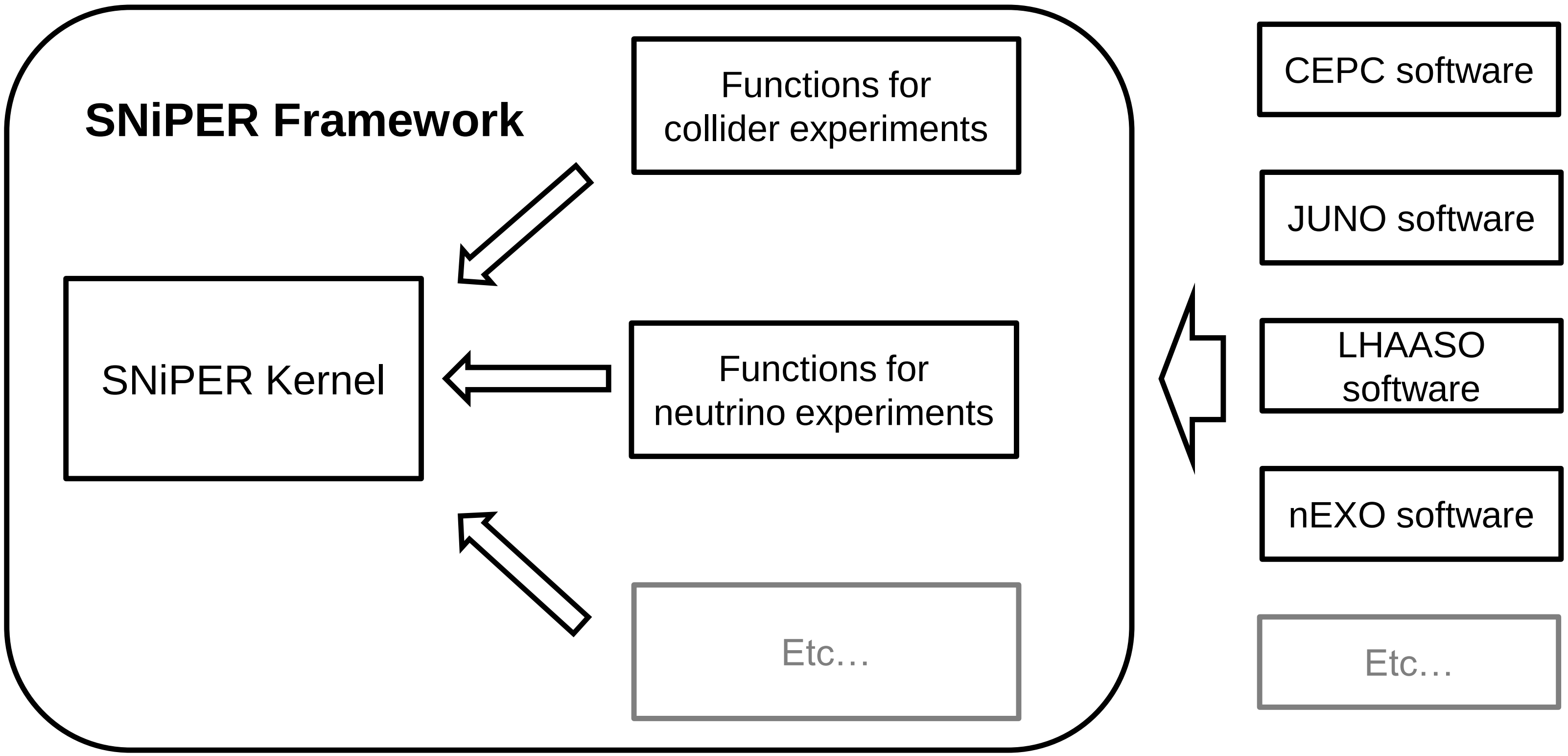
A group of optional components

- Functions for collider physics experiments
- Functions for neutrino experiments
- …

Special functions for each experiment

- Data model
- Algorithms
- …

Prospect of a SNiPER Ecosystem



- Be attractive to community developers
- To find more application scenarios

Content

- General introduction
- **Key concepts**
- Examples
- Plan for CEPC

Key Concepts

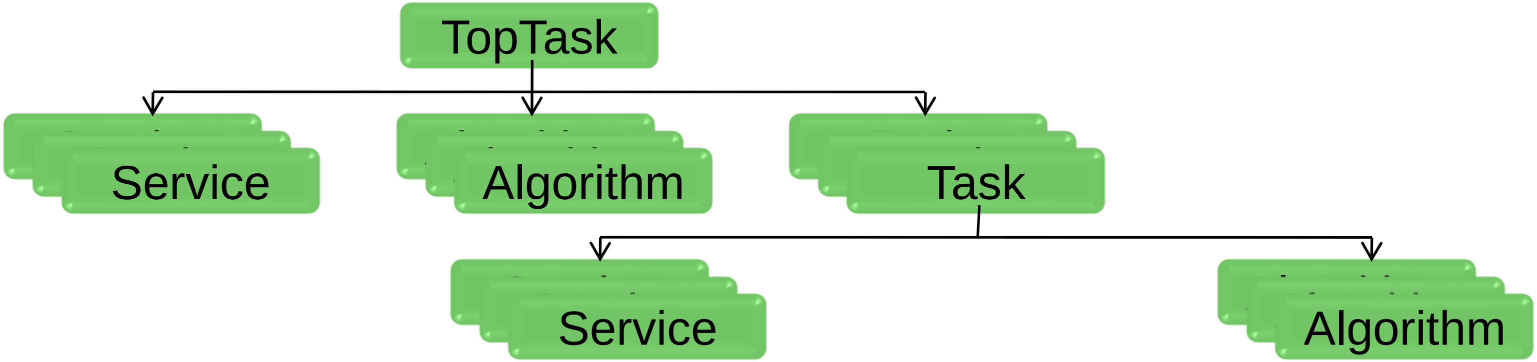
- DLElement: Dynamically Loadable Element
  - Task
  - Algorithm
  - Service
  - Tool

Each DLElement object has a unique string name(path)

- Data Memory Service
- Incident
- Property
- Log (message output)

Task

- Similar to the Gaudi application manager
  - Management of algorithms, services and sub-tasks
  - Controlling the execution of algorithms
  - Has its own data memory management service
  - Has its own I/O management service
- There can be more than one Tasks in a single job (e.g. event mixing)
- All DLEs are organized in a tree structure



Algorithm in C++

- A concept inherited from Gaudi
- An unit of codes for Data Processing, (similar to marlin::Processor)
  - the event calculation during event loop
  - Most frequently used by users
- AlgBase, the abstract base class in SNiPER
  - User's algorithm must be inherited from AlgBase
  - Its constructor takes one std::string parameter as the object name
  - 3 abstract interfaces must be implemented, they are called by SNiPER automatically
    - 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)

Service in C++

- A concept inherited from Gaudi
- A piece of code for common uses
  - Such as RootIOSvc, GeometrySvc …
  - Be invoked by users, not limited to the event loop
  - Be initialized before algorithms in each Task
- SvcBase, the abstract base class in SNiPER
  - A new service must be inherited from SvcBase
  - Its constructor takes one std::string parameter as the object name
  - 2 abstract interfaces 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)

Tool

- A concept inherited from Gaudi
- Tool is also a Dynamically Loadable Element
- It belongs to an algorithm and helps the algorithm to organize code more clearly
- One algorithm can have one or more tools
- A tool can be accessed via its name

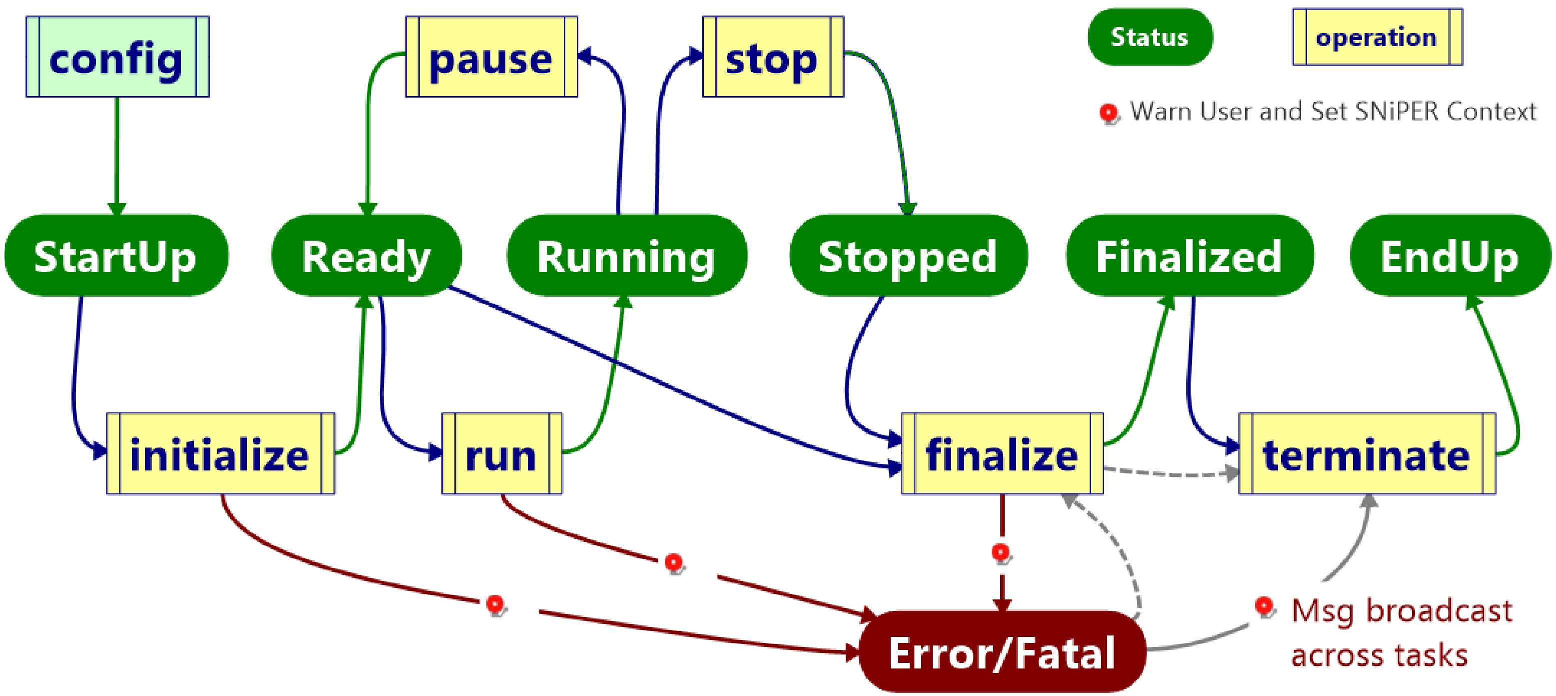
```
bool DummyAlg::execute()
{
    //Valid log level: LogDebug, LogInfo, LogWarn, LogError, LogFatal
    LogDebug << "Processing event " << m_iEvt << std::endl;

    //call a tool
    DummyTool* ptool = tool<DummyTool>("dtool");
    ptool->doSomeThing();

    return true;
}
```

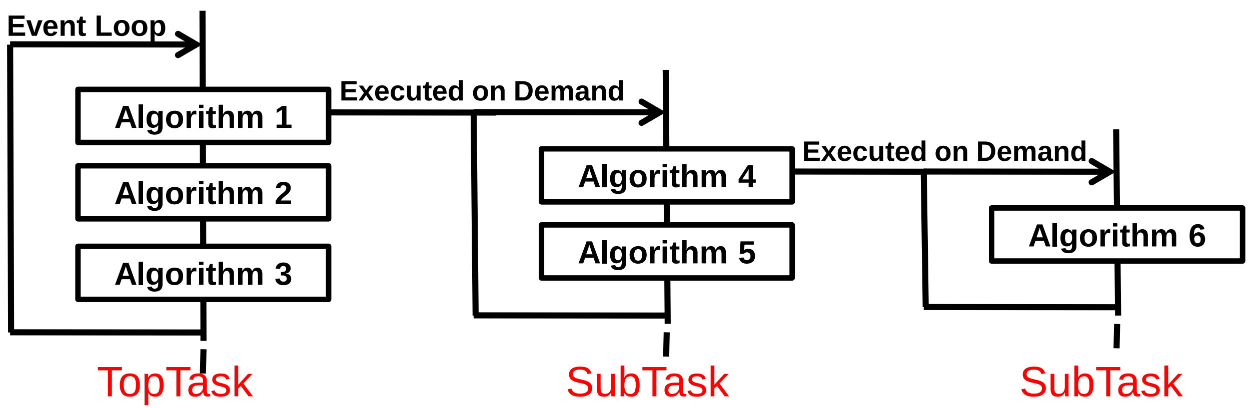
Task Execution

A state machine of the execution:



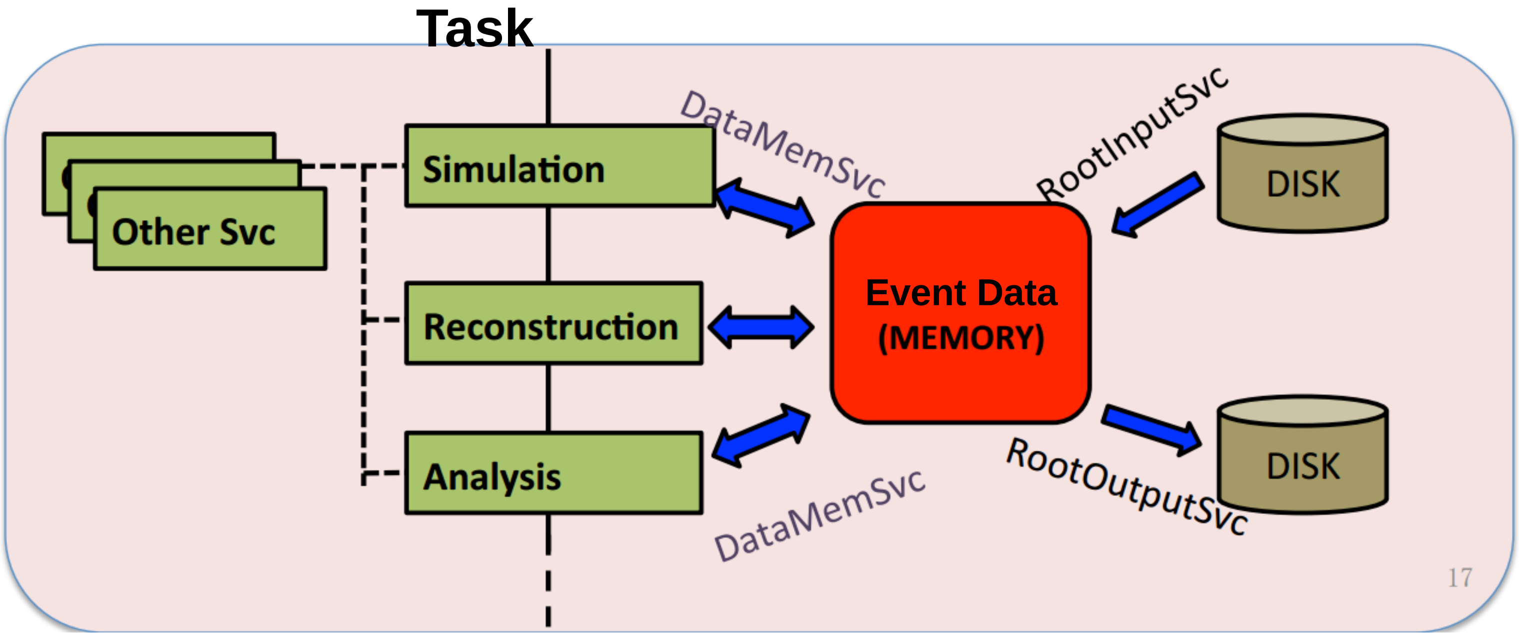
Data Processing with Task

- Task means the event processing procedure (a event loop)
- Task and SubTask provide a more flexible execution procedure
  - SubTask(s) are executed synchronously on demand
  - Can be used for different event loops
  - *Multi-Thread Computing (run each task in an individual thread)*
- Task is a FSM (finite-state machine)
  - Startup
  - Ready
  - Running
  - Finalized
  - Endup



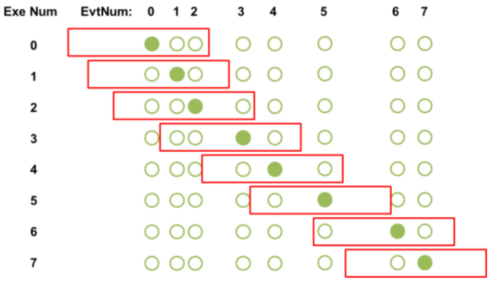
Data Memory Service

- Data memory service is in charge of the dynamically allocated memory, by which to hold events data that being processed
- Applications (in terms of algorithms) get events data via the data memory service and update them after processing



Optional Data Memory Services

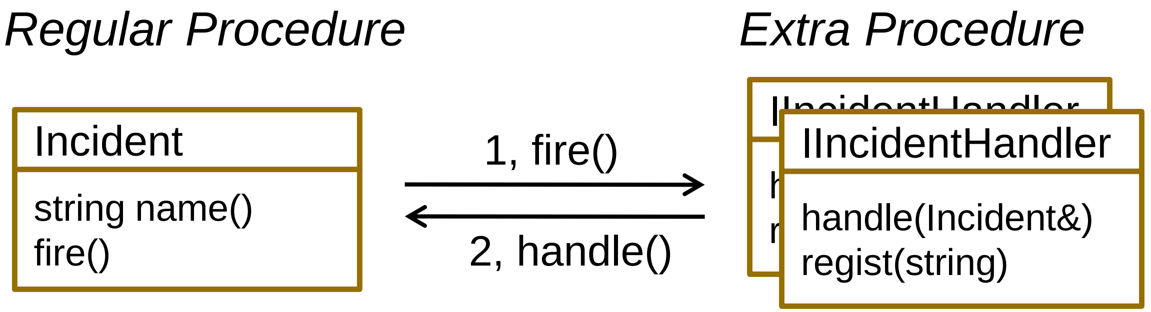
- Different type of experiments have different requirements
  - Several implementations to select
- DataBuffer for neutrino experiments
  - A sequence of events in a time window
  - Be able to handle events correlations



- PyDataStore: transfer data between C++ and Python
  - Writing algorithms in Python
  - Mixing execution of C++ algorithms and Python algorithms
  - Examples/HelloWorld
- EventStore (to be implemented): similar to the TDS in Gaudi
  - Reset event by event automatically

Incident

- Provides an additional degree of execution freedom:
  - Incident: trigger the execution of corresponding handlers
  - IncidentHandler: the wrapper of any specific procedure

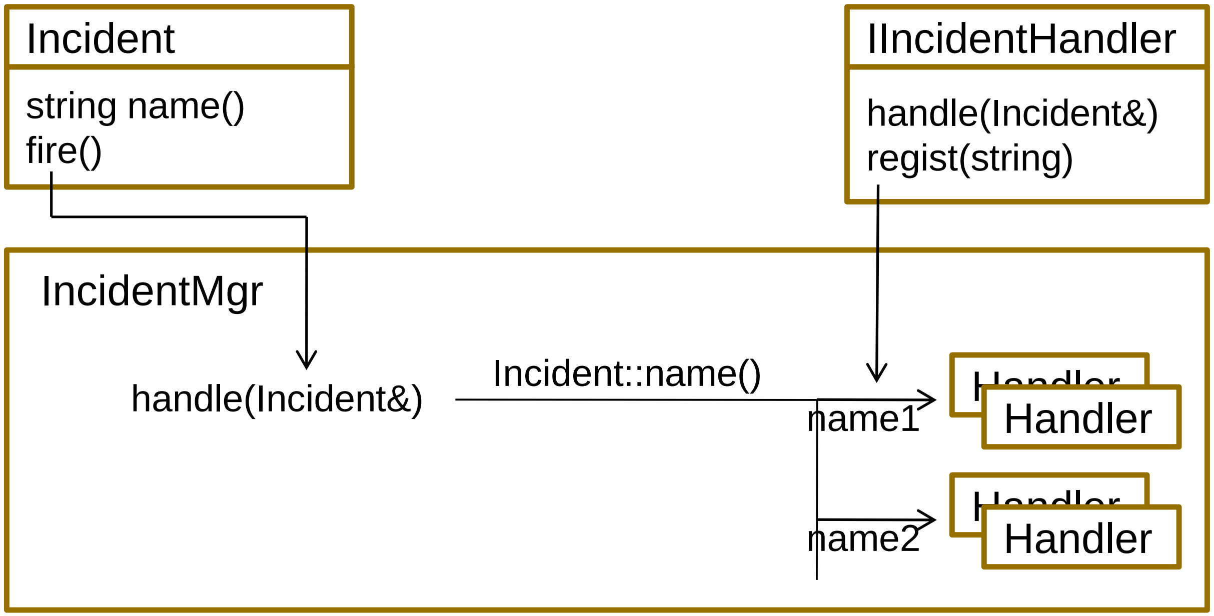


1. Regular execution procedure jumps to another extra procedure
2. Back to the original procedure after all corresponding Handlers are executed

- We can fire an incident anywhere according to the requirements
- It’s easy to define and use a customized incident

Incident Management

- IncidentMgr correlates incidents with their handlers
  - Incidents are distinguished by its name, such as “BeginEvent”, “EndEvent”
  - One IncidentHandler can be registered to several Incidents
  - One Incident can be handled by several IncidentHandlers
- Currently Event I/O and SubTask execution are based on incident mechanism



Property

- Configurable variable at run time
- Declare a property in DLElement (C++ code)

```
//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

This mechanism is also used to create and load algorithms and services:

```
task.property("svcs").append("RootWriter")
task.property("algs").append("DummyAlg/dalg")
```

Logs

- SniperLog: simple and thread-safe, supports different output levels

0: LogTest

2: LogDebug

3: LogInfo

4: LogWarn

5: LogError

6: LogFatal

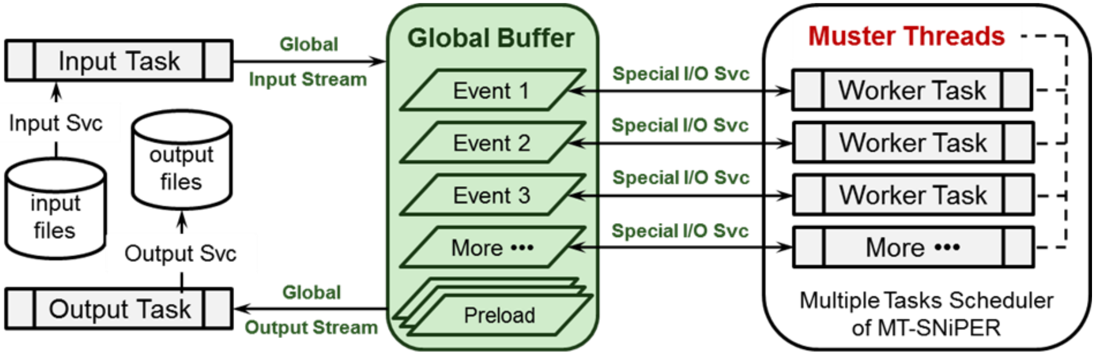
```
LogDebug << "A debug message" << std::endl;
LogInfo  << "An info message" << std::endl;
LogError << "An error message" << std::endl;
```

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

- Each DLElement has its own LogLevel and can be set at run time
  - very helpful for debugging
- The output message includes more information
  - where it happens
  - the message level
  - The message contents

Parallel Computing

- Current Status of MT-SNiPER
  - Run each SNiPER Task in a separated thread
  - Based on Intel TBB, implemented the prototype SniperMuster
  - Non-invasive, no change to the SNiPER kernel module
  - Almost be transparent to users, easy to migrate from serial apps
  - The testing of JUNO simulation shows a reasonable result
- Next
  - More general
  - Parallel algorithms
  - MPI



Content

- General introduction
- Key concepts
- **Examples**
- Plan for CEPC

Create an Algorithm and a Service

- Package management
- C++ and Python coding
- CMT configuration
- Compile and run

Advanced topic: a job with multiple-tasks

svn co http://juno.ihep.ac.cn/svn/juno/people/zoujh/example/FirstToy

Coding and Running

- FirstToy C++
  - FirstAlg, our first algorithm
    - Show different level of logs
  - FirstSvc, our first service
    - A string message as property (can be modified in python)
    - An interface to print the string message (*answer()*)
  - SecondAlg
    - Call the service in an algorithm
- FirstToy Python

```python
import Sniper
Sniper.loadDll("libFirstAlg.so")
```

VS.

```python
import FirstAlg
```

- Compile with CMT (or CMake), and run in Python

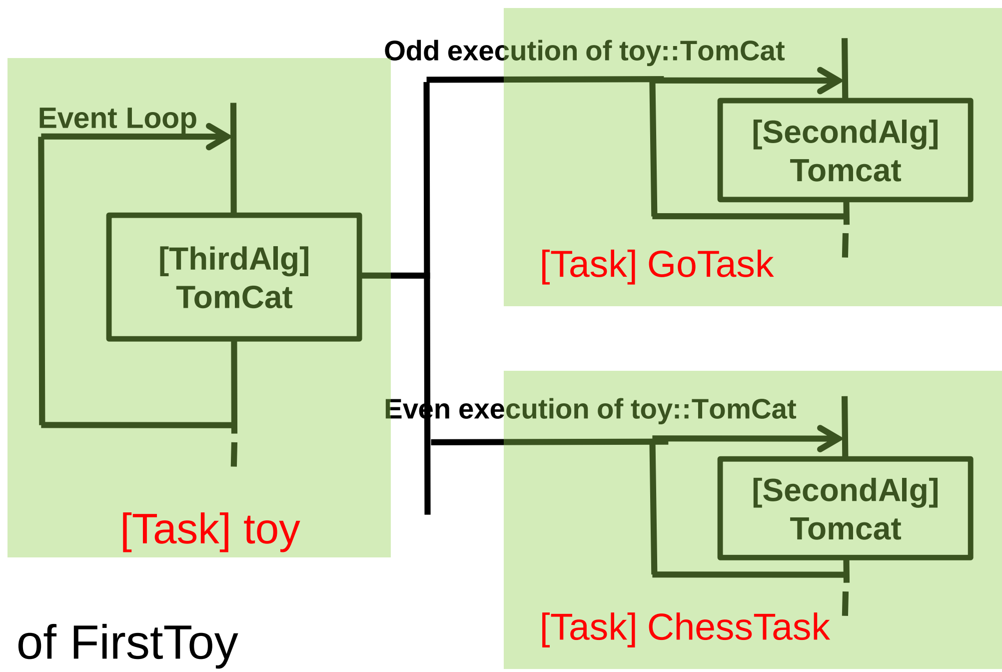
Advanced Topic: multiple-tasks job

The DLElement Map of

ThirdAlg + SecondAlg + FirstSvc + Task

```
[Task]toy
    |
    +--[ThirdAlg]TomCat
    |
    +--[Task]toy:GoTask
    |      +--[FirstSvc]FirstSvc
    |      +--[SecondAlg]SecondAlg
    |
    +--[Task]toy:ChessTask
    |      +--[FirstSvc]FirstSvc
    |      +--[SecondAlg]SecondAlg
```

SubTask(s) are executed on demand



Details can be found in ThirdAlg of FirstToy

Configuration with Python

Execute a dummy algorithm in SNiPER, create 2 root output files:

- Examples/DummyAlg/share/run.py

```python
import Sniper

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

import RootWriter
task.property("svcs").append("RootWriter")
rw = task.find("RootWriter")
rw.property("Output").set({"FILE1": "output1.root", "FILE2": "output2.root"})

import DummyAlg  #infact DummyTool is imported at the same time
alg = task.createAlg("DummyAlg/dalg")
alg.createTool("DummyTool/dtool")

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

Each job must has at least 1 Task instance

Create and set the RootWriter service

Create a DummyAlg instance with a DummyTool

Set event number and begin the execution

Execution with Python

```
zoujh@office share $ python run.py
**************************************************
***          Welcome to SNiPER Python          ***
**************************************************
Running @ debian on Mon Apr  1 12:24:30 2019
task:dalg.initialize            INFO:  initialized successfully
task.initialize                 INFO: initialized
task:dalg.execute              DEBUG: Processing event 1
task:dtool.doSomeThing          INFO: DummyTool is running :)
task:dalg.execute              DEBUG: Processing event 2
task:dtool.doSomeThing          INFO: DummyTool is running :)
task:dalg.execute              DEBUG: Processing event 3
task:dtool.doSomeThing          INFO: DummyTool is running :)
task:dalg.execute              DEBUG: Processing event 4
task:dtool.doSomeThing          INFO: DummyTool is running :)
task:dalg.execute              DEBUG: Processing event 5
task:dtool.doSomeThing          INFO: DummyTool is running :)
task:dalg.finalize              INFO:  finalized successfully
task.finalize                   INFO: events processed 5

**************************************************
Terminating @ debian on Mon Apr  1 12:24:30 2019
SNiPER::Context Running Mode = { BASIC }
SNiPER::Context Terminated Successfully
```

Startup

Initialization

Event loop

Messages for each event

Finalization

Endup

Plans for CEPC

- Common Functions
  - EventStore for Collider Physics Experiments
  - Data Model: be similar to the LCIOEvent, but ROOT based
  - Data (ROOT format) I/O Services
  - Before the end of April 2019 ?
  - Convert the existed LCIO data to ROOT data for analysis
- Other services and algorithms
  - Geometry service based on DD4hep
  - marlin::Processor -> Sniper Algorithm migration should be easy
    - Keep similar interfaces, such as data model and geometry

Thanks for your attention

Any questions?