Tutorial for the SNiPER Framework

Zou Jiaheng, Lin Tao, Huang Xingtao, Li Weidong

2018-05-13

Content

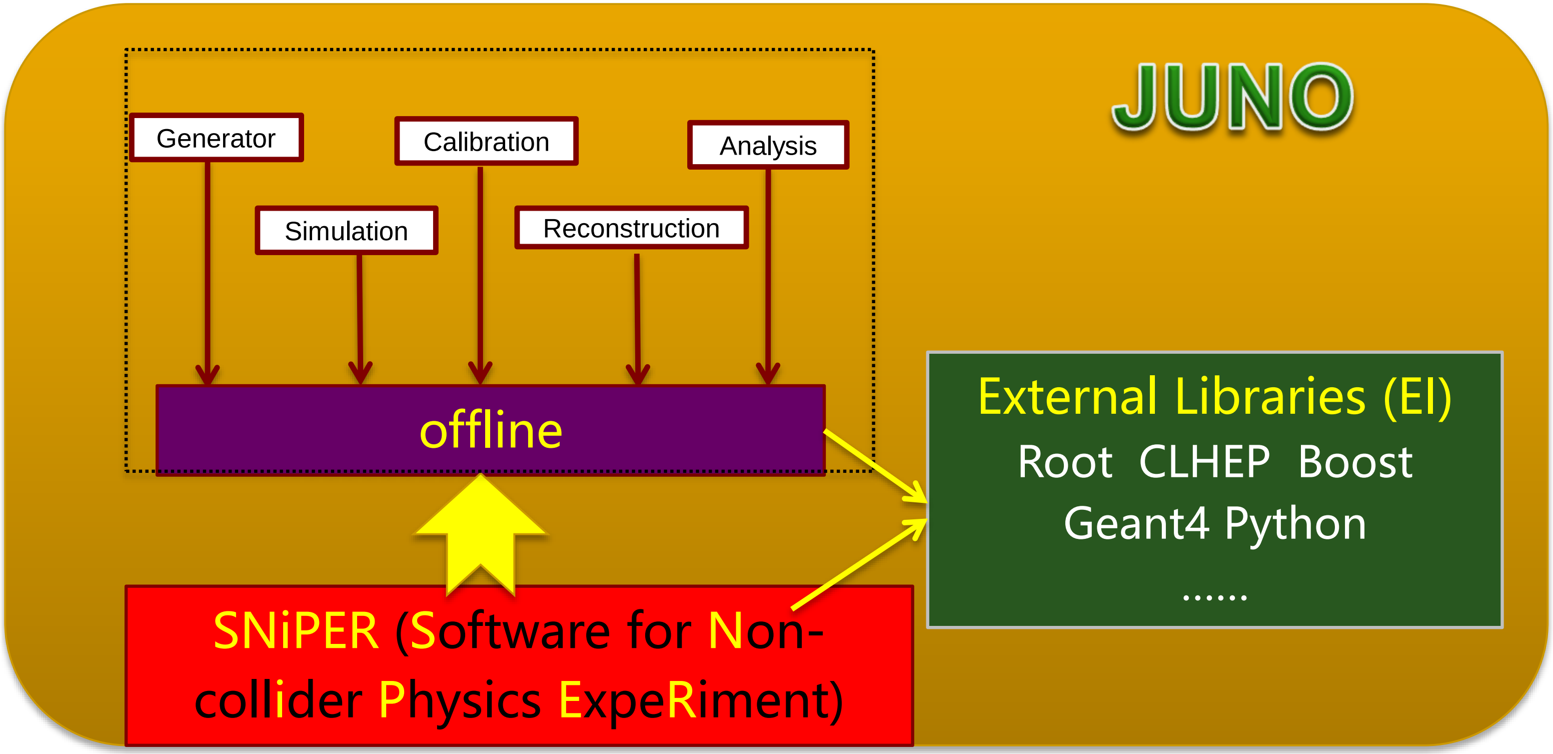
- General introduction
- Key concepts
- Running the HelloWorld

Offline Software Environments

- Programming language: hybrid programming of C++ and Python
  - Very popular in HEP field
  - Most frequently used software is implemented in C++ (ROOT, Geant4 …)
- Job configuration interface: Python
  - Very flexible
  - Easy to glue different tools together (Job scheduling, Monitoring …)
- Packages management tool: CMT(Configuration Management Tool)
  - Help developers to compile packages easily
  - Help users to setup the environment for running the application easily
- Supported Operation System: Linux
  - Official recommendation: Scientific Linux 6 / CentOS 7
  - Some colleagues compile successfully on Ubuntu, Debian …
- Codes Management: SVN
  - Keep the history of code evolution
  - Synchronization and sharing between developers
  - Tag and release

Overview of JUNO Offline Software

- SNiPER: the underlying Framework
- Offline: extension of SNiPER and applications for JUNO
- External Libraries(EI): very frequently used software and tools



Software Framework

- What's an offline software framework?
  - A framework helps users to write as less code as possible to achieve their goals
- What does a framework provide?
  - Management of Event Data
    - Interfaces to define, read, access and write event data
  - Management of data processing
    - Sequence and/or filtering of algorithms
  - Common services and tools for data processing
    - HistogramSvc, RandomSvc, DatabaseSvc …
  - Friendly user interface
    - Simple interfaces for coding: abstract base classes for algorithms and services
    - Simple interfaces for running: configure jobs via text, python …

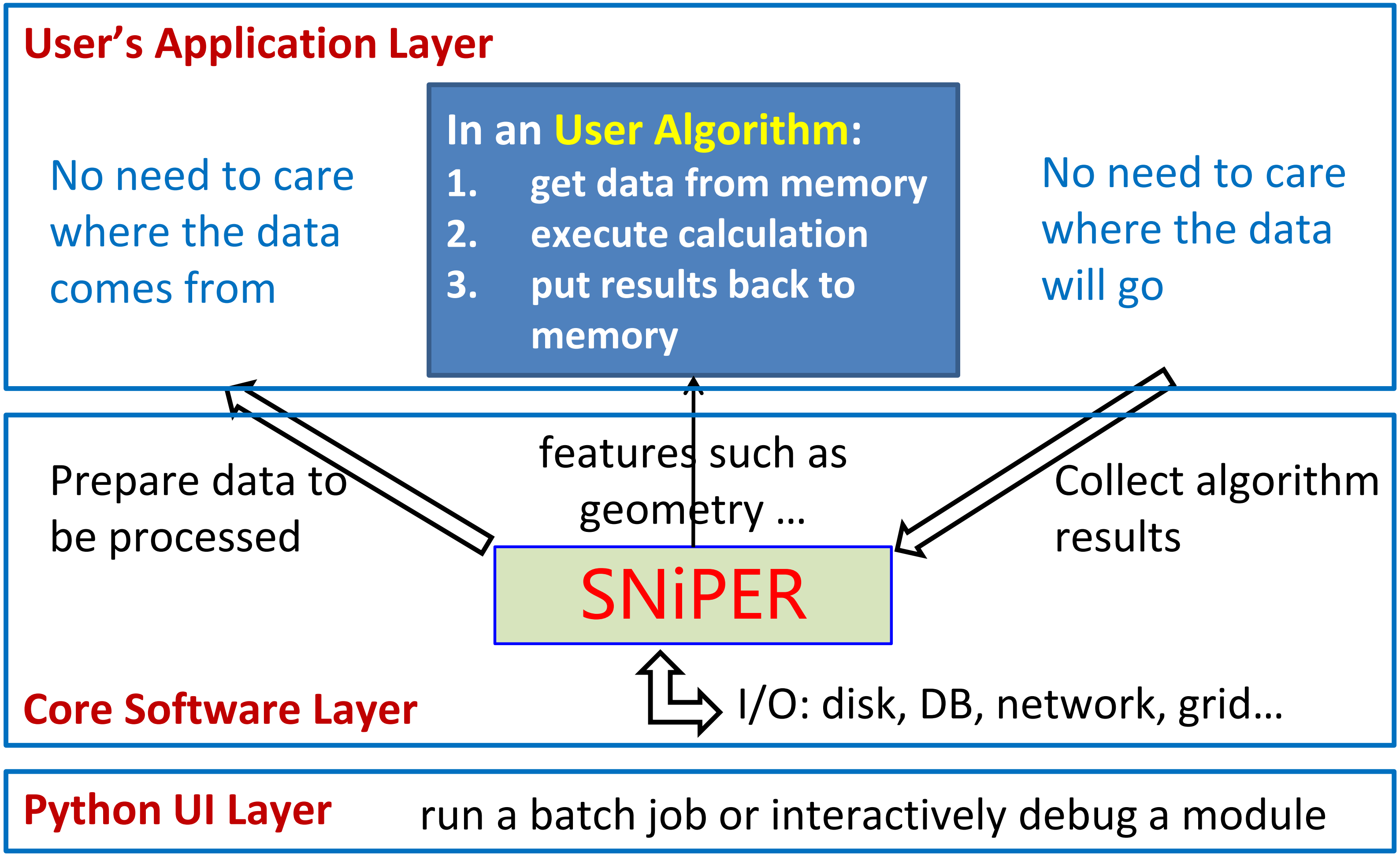
Software Framework for JUNO

- SNiPER: Software for Non-collider Physics ExpeRiment
- Main goals
  - Lightweight, less dependences on third-party software/libs
  - Fast and flexible execution
  - Easy to learn and convenient to use
- Design and development
  - Learn a lot from other software frameworks, such as Gaudi
  - Based on the valuable experiences of Daya Bay Experiment
  - Coding from scratch
- Current Status
  - Performs well for JUNO (and LHAASO, a cosmic ray exp. in China)
  - Several other projects and potential users (CSNS, nEXO …)

Key Functionalities of Framework

- Dynamically loading packages and elements
  - User’s packages can be executed as plugins
  - It is easy to customize a job
- Flexible execution
  - Task, TopTask, Incident
  - Very useful for event splitting and mixing
- Event management in memory
  - Multiple events within time windows accessible
  - Very convenient for events correlation analysis
- Parallel computing (will come soon)

Working with SNiPER



User's Application Layer

No need to care where the data comes from

In an User Algorithm:

1. **get data from memory**
2. **execute calculation**
3. **put results back to memory**

No need to care where the data will go

Prepare data to be processed

features such as geometry …

Collect algorithm results

SNiPER

I/O: disk, DB, network, grid…

Core Software Layer

Python UI Layer run a batch job or interactively debug a module

Key Concepts

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

Each DLElement object has a unique string name

- Data Buffer
- Incident
- Property
- Log (message output)

Algorithm

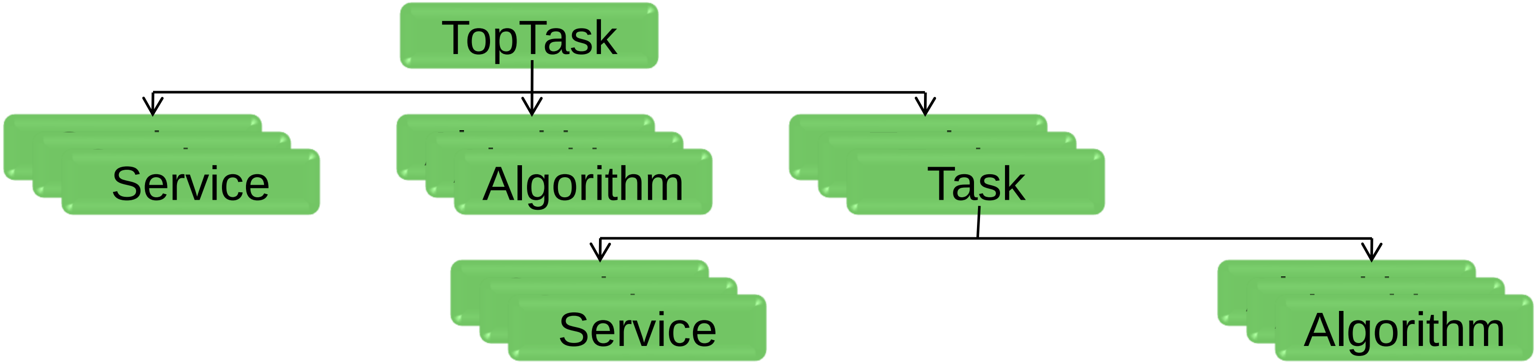
- An unit of codes for Data Processing
  - the calculation during event loop
  - Most frequently used by users
- AlgBase, the abstract base class in SNiPER
  - User’s algorithm must be inherited from AlgBase
  - Its constructor takes one std::string parameter
  - 3 abstract interfaces must be implemented, 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)
- We will show how to create an algorithm later

Service

- Similar with Algorithm
  - An Dynamically Loadable Element
  - One Task probably composes of one or more services
- But different from Algorithm
  - A piece of code for common use (RootIOSvc, GeometrySvc …)
  - They are called by user’s request, not limited to event loop
- SvcBase, the abstract base class in SNiPER
  - A new service must be inherited from SvcBase
  - Its constructor takes one std::string parameter
  - 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)
- We will show how to create a service later

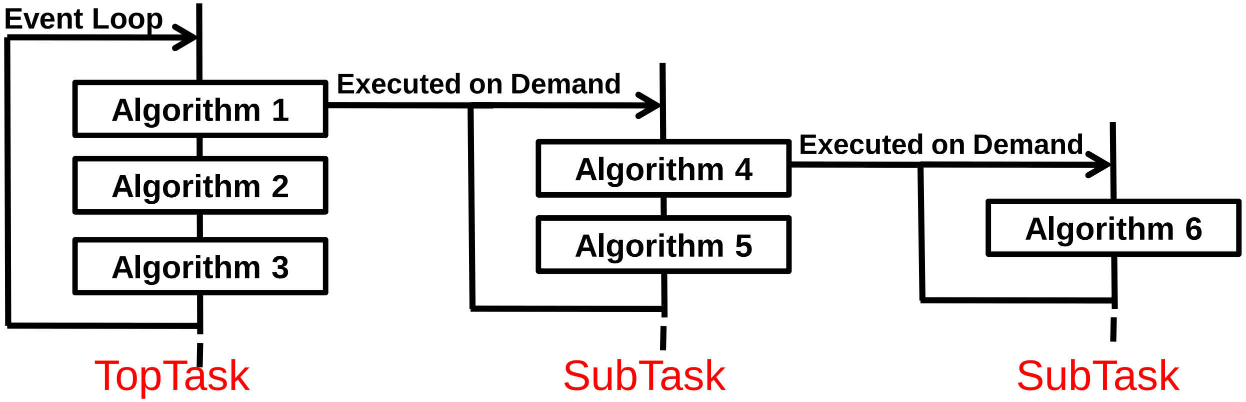
Task

- A lightweight traditional Application Manager
  - Management of algorithms, services and tasks
  - Controlling the execution of algorithms
  - Has its own data memory management
  - Has its own I/O management
- One job can has more than one Tasks(e.g. event mixing)
- All DLEs are organized in a tree structure

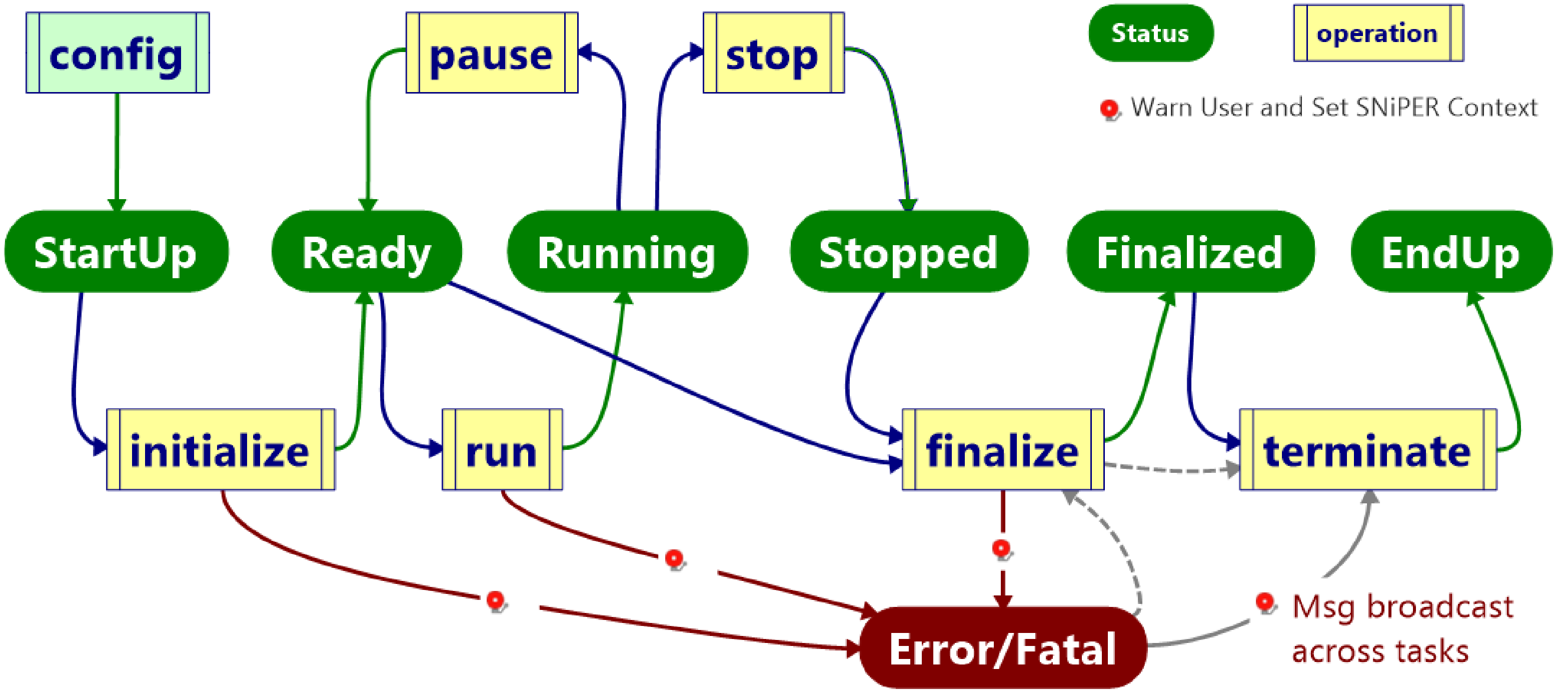


Data Processing with Task

- Task means the event processing procedure (event loop)
- SubTask provides nested event loop
  - It will be executed on demand
- Task and SubTask provide more flexible execution
  - Meet the requirements of Event Mixing and Event Splitting
  - *Multi-Thread Computing (run each task in an individual thread)*
- Task is a FSM (finite-state machine)
  - Startup
  - Ready
  - Running
  - Finalized
  - Endup



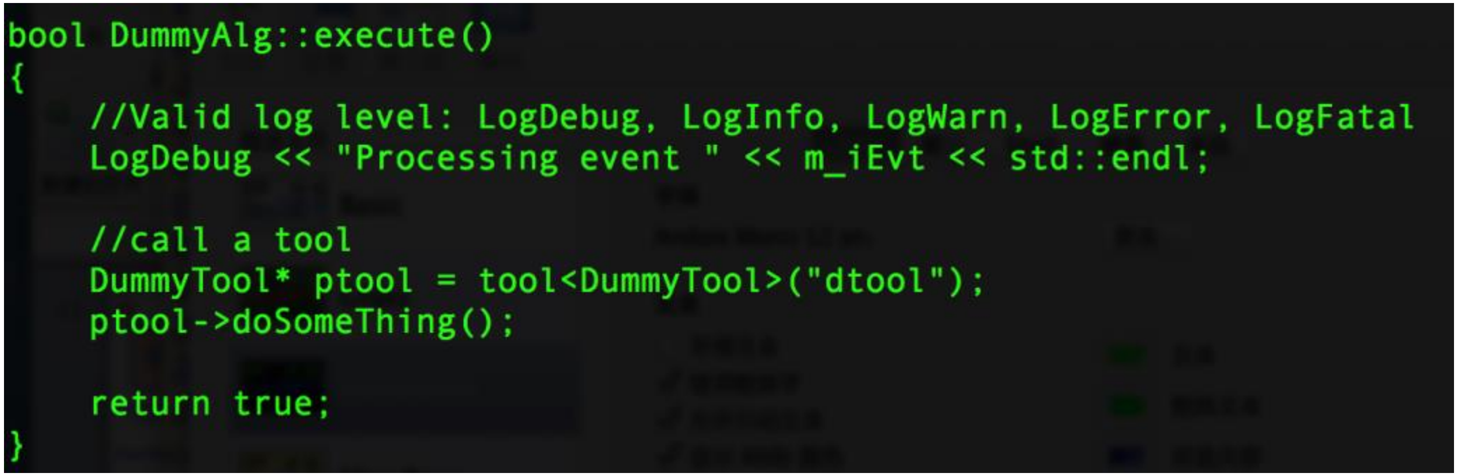
Task Status



config
pause
stop
Status
operation
Warn User and Set SNiPER Context
StartUp
Ready
Running
Stopped
Finalized
EndUp
initialize
run
finalize
terminate
Error/Fatal
Msg broadcast across tasks

Tool

- 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



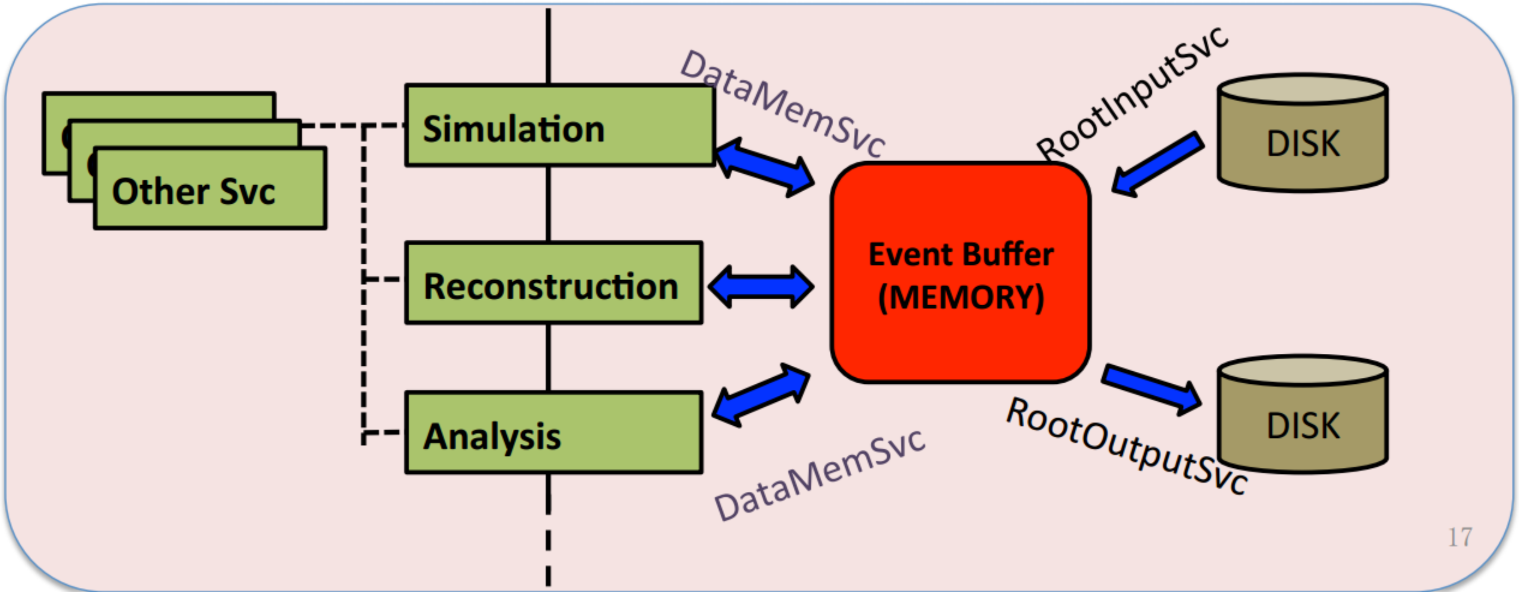
```
bool DummyAlg::execute()
{
    //Valid log level: LogDebug, LogInfo, LogWarn, LogError, LogFatal
    LogDebug << "Processing event " << m_iEvt << std::endl;

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

    return true;
}
```

Data Buffer

- Data Buffer is the dynamically allocated memory place to hold events data which are being processed
- Applications (in terms of algorithms) get events data from the buffer and update them after processing



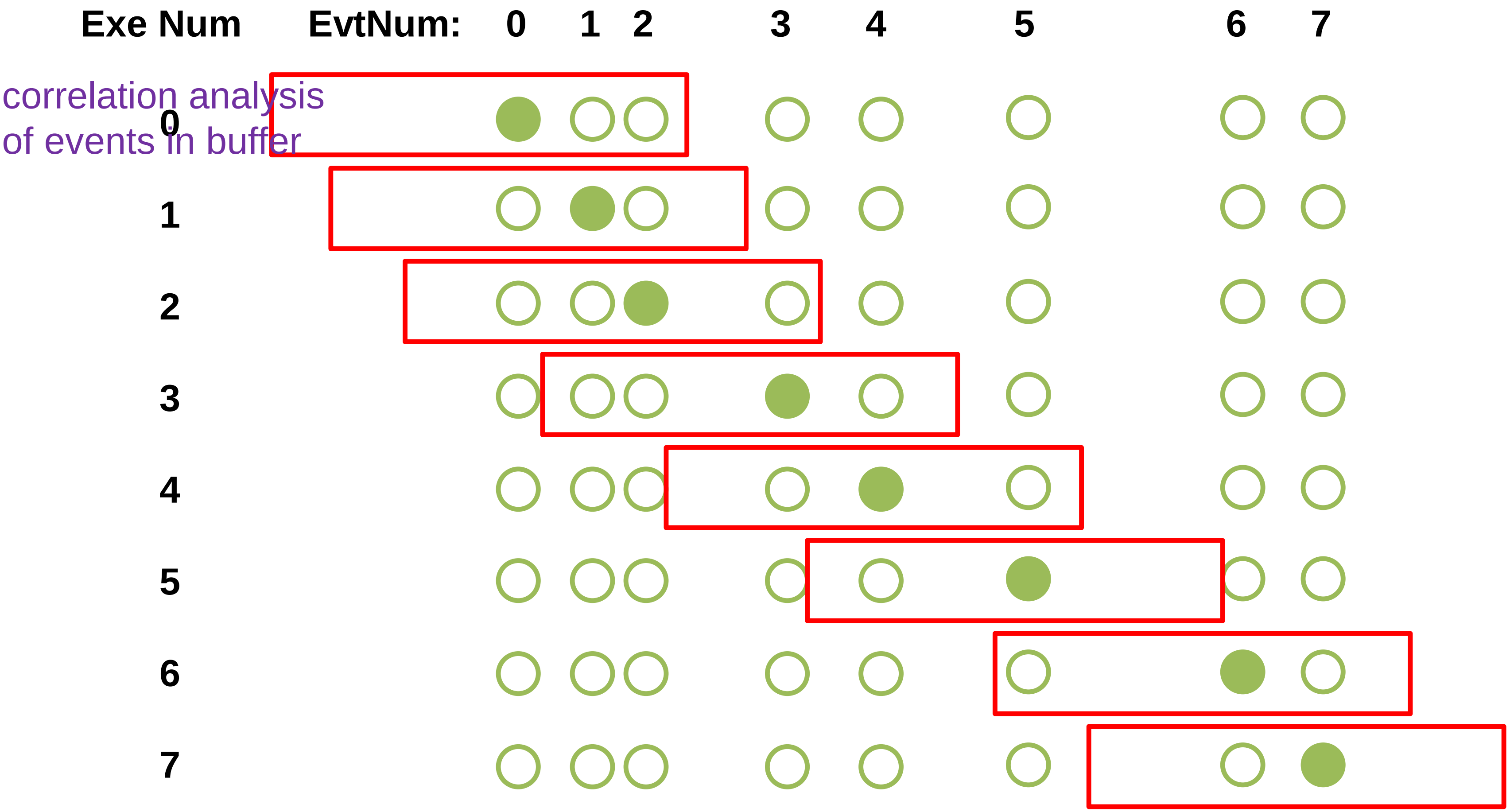
Data Buffer in Memory

● Current event

○ Other events

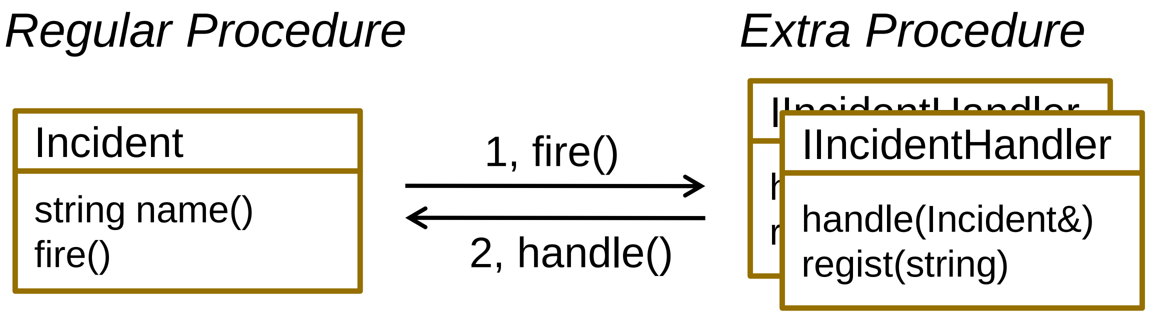
🟥 Event buffer

Buffer: a sequence of events in a time window



Incident

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



Regular Procedure

Extra Procedure

Incident

string name()
fire()

1, fire()

2, handle()

IIncidentHandler

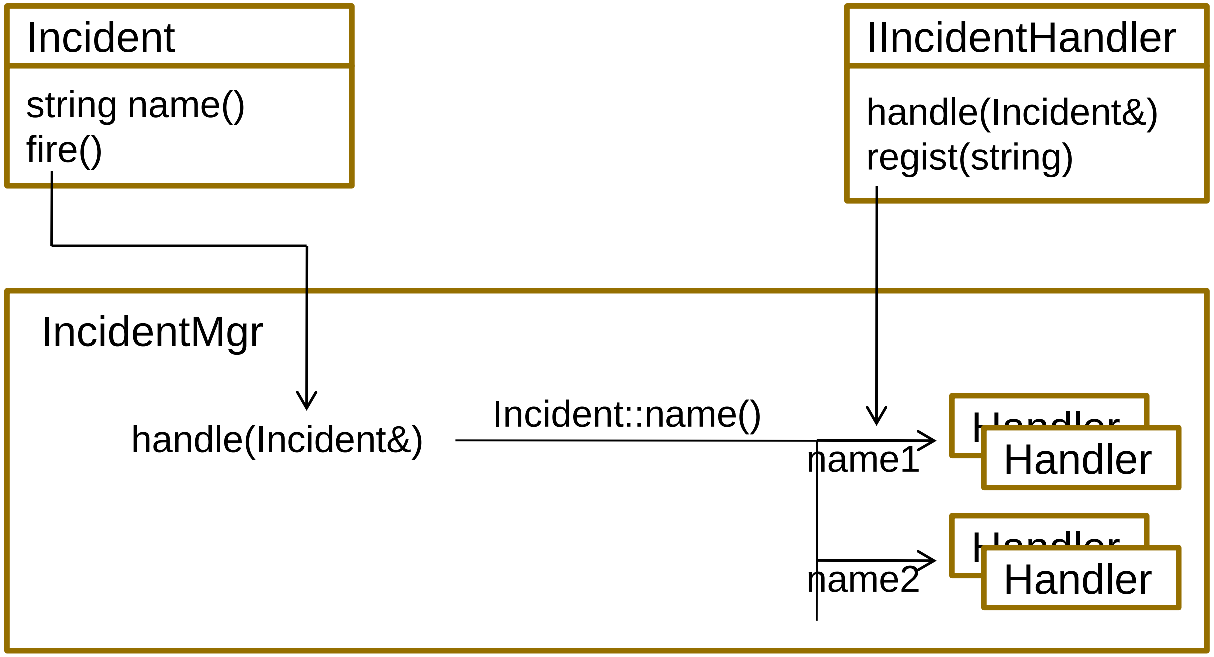
handle(Incident&)
regist(string)

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 according to their needs

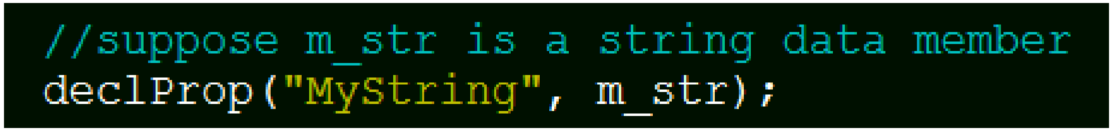
Incident Management

- 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



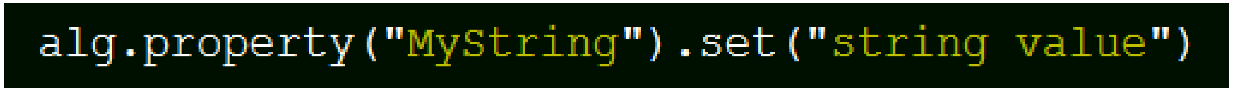
Property

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



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

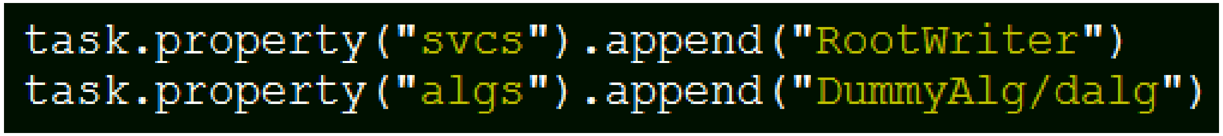
- Configure a property in Python script



```
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")
```

Log Mechanism

- SniperLog: a simple log mechanism supports different output levels

0: LogTest

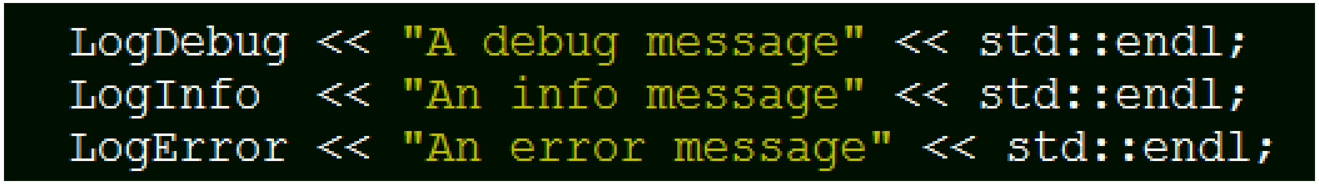
2: LogDebug

3: LogInfo

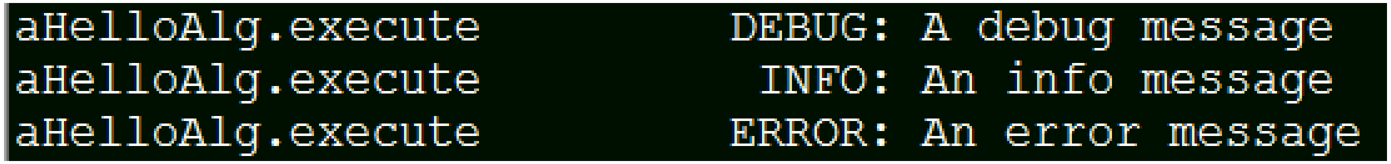
4: LogWarn

5: LogError

6: LogFatal



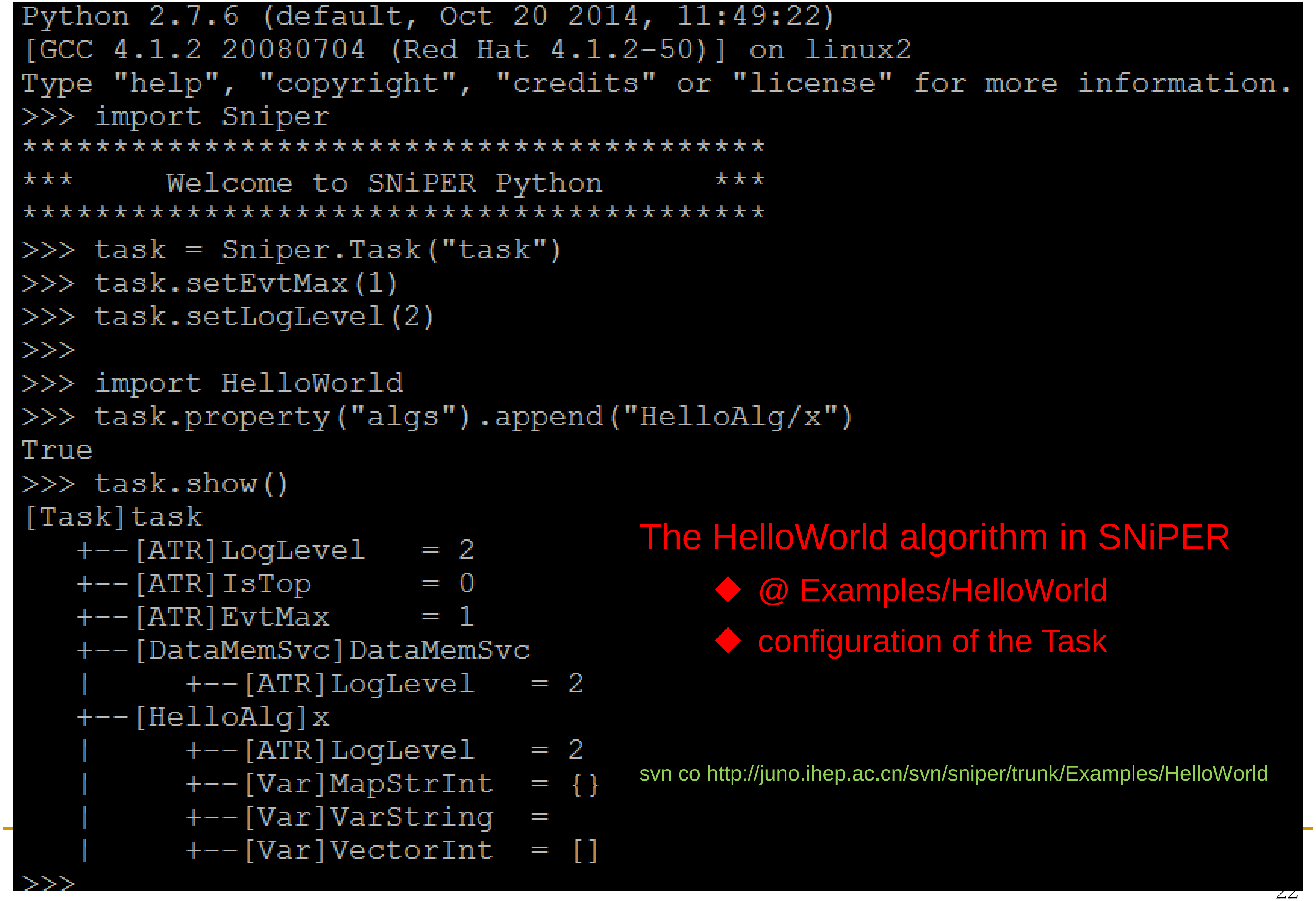
```
  LogDebug << "A debug message" << std::endl;
  LogInfo  << "An info message" << std::endl;
  LogError << "An error message" << std::endl;
```



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

- Each DLElement 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

HelloWorld (I)



```
Python 2.7.6 (default, Oct 20 2014, 11:49:22)
[GCC 4.1.2 20080704 (Red Hat 4.1.2-50)] on linux2
Type "help", "copyright", "credits" or "license" for more information.
>>> import Sniper
*****************************************
***     Welcome to SNiPER Python      ***
*****************************************
>>> task = Sniper.Task("task")
>>> task.setEvtMax(1)
>>> task.setLogLevel(2)
>>>
>>> import HelloWorld
>>> task.property("algs").append("HelloAlg/x")
True
>>> task.show()
[Task]task
   +--[ATR]LogLevel   = 2
   +--[ATR]IsTop      = 0
   +--[ATR]EvtMax     = 1
   +--[DataMemSvc]DataMemSvc
   |     +--[ATR]LogLevel   = 2
   +--[HelloAlg]x
   |     +--[ATR]LogLevel   = 2
   |     +--[Var]MapStrInt  = {}
   |     +--[Var]VarString  =
   |     +--[Var]VectorInt  = []
>>>
```

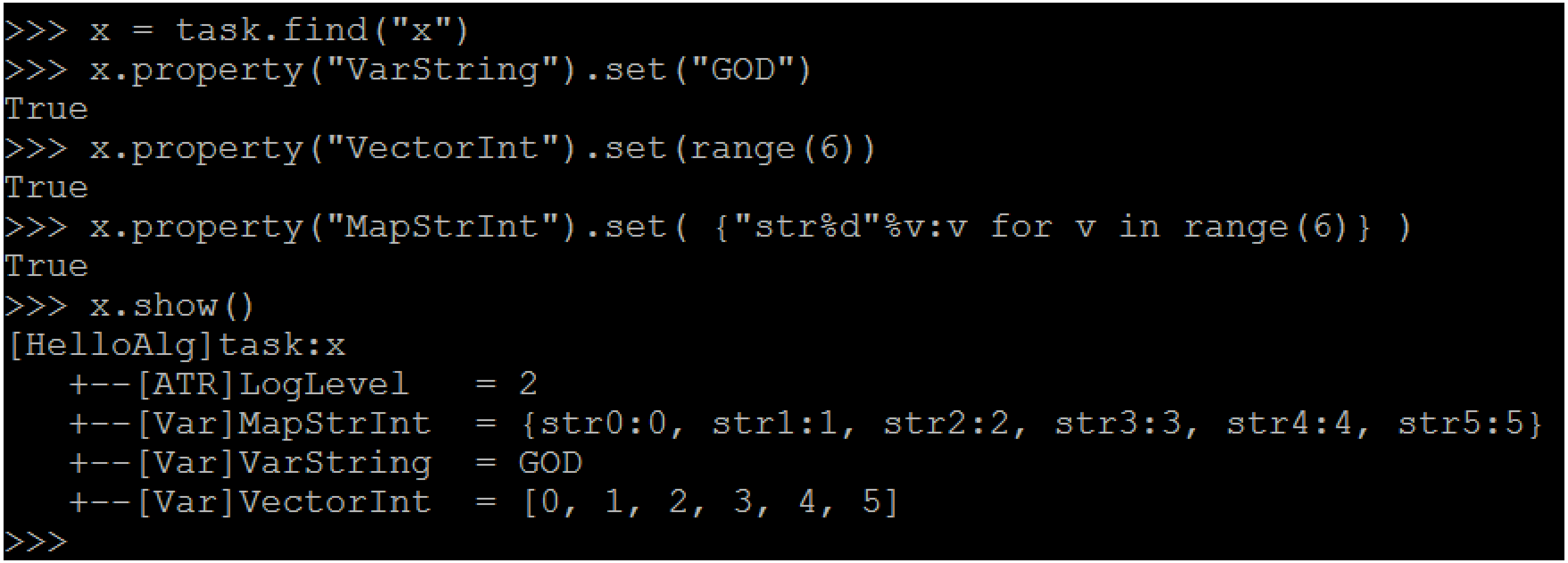
The HelloWorld algorithm in SNiPER

- @ Examples/HelloWorld
- configuration of the Task

svn co http://juno.ihep.ac.cn/svn/sniper/trunk/Examples/HelloWorld

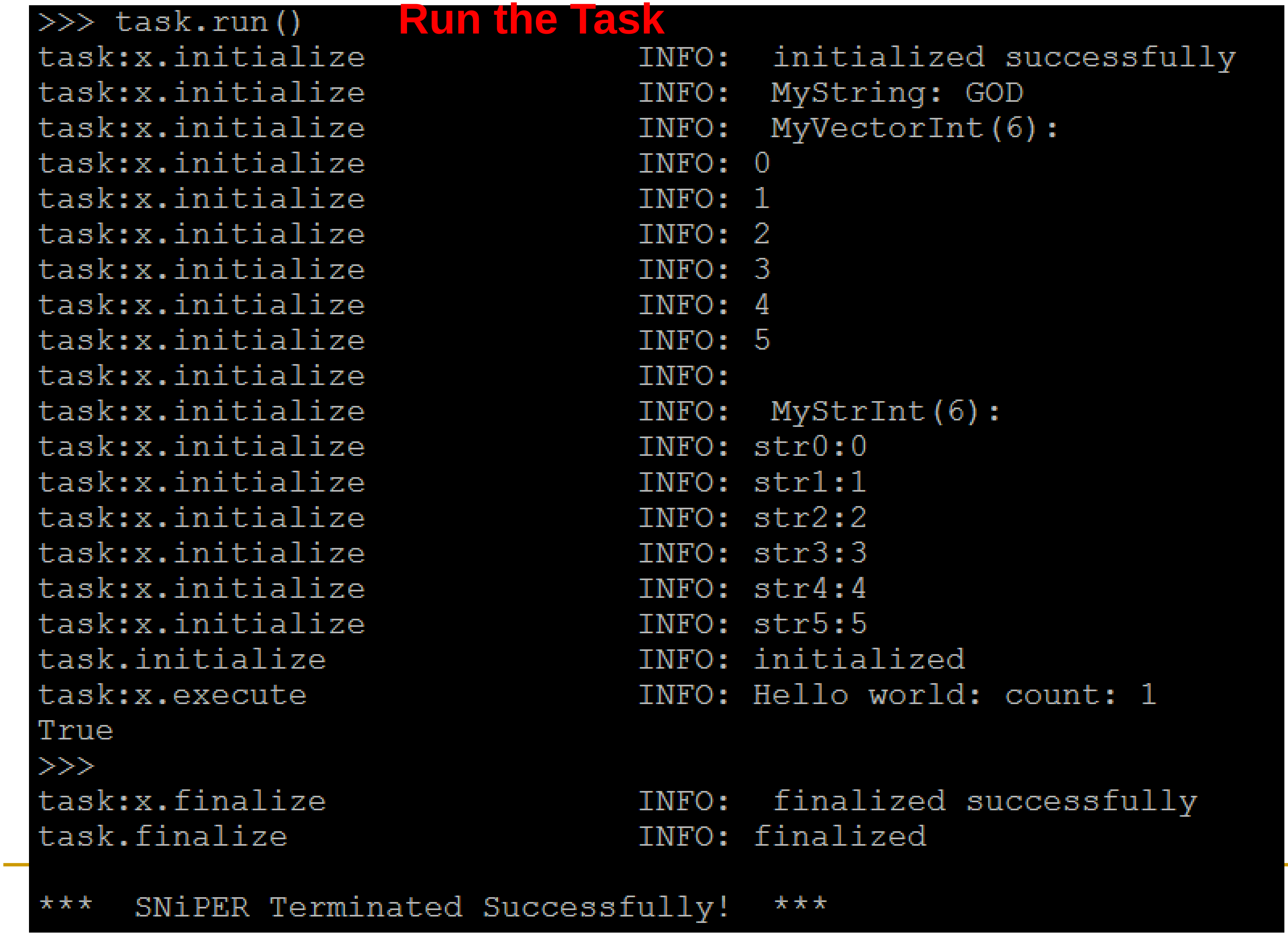
HelloWorld (II)

Configuration of the algorithm properties



```
>>> x = task.find("x")
>>> x.property("VarString").set("GOD")
True
>>> x.property("VectorInt").set(range(6))
True
>>> x.property("MapStrInt").set( {"str%d"%v:v for v in range(6)} )
True
>>> x.show()
[HelloAlg]task:x
   +--[ATR]LogLevel   = 2
   +--[Var]MapStrInt  = {str0:0, str1:1, str2:2, str3:3, str4:4, str5:5}
   +--[Var]VarString  = GOD
   +--[Var]VectorInt  = [0, 1, 2, 3, 4, 5]
>>>
```

HelloWorld (III)



Run the Task

```
>>> task.run()
task:x.initialize              INFO:  initialized successfully
task:x.initialize              INFO:  MyString: GOD
task:x.initialize              INFO:  MyVectorInt(6):
task:x.initialize              INFO: 0
task:x.initialize              INFO: 1
task:x.initialize              INFO: 2
task:x.initialize              INFO: 3
task:x.initialize              INFO: 4
task:x.initialize              INFO: 5
task:x.initialize              INFO:
task:x.initialize              INFO:  MyStrInt(6):
task:x.initialize              INFO: str0:0
task:x.initialize              INFO: str1:1
task:x.initialize              INFO: str2:2
task:x.initialize              INFO: str3:3
task:x.initialize              INFO: str4:4
task:x.initialize              INFO: str5:5
task.initialize                INFO: initialized
task:x.execute                 INFO: Hello world: count: 1
True
>>>
task:x.finalize                INFO:  finalized successfully
task.finalize                  INFO: finalized

***  SNiPER Terminated Successfully!  ***
```

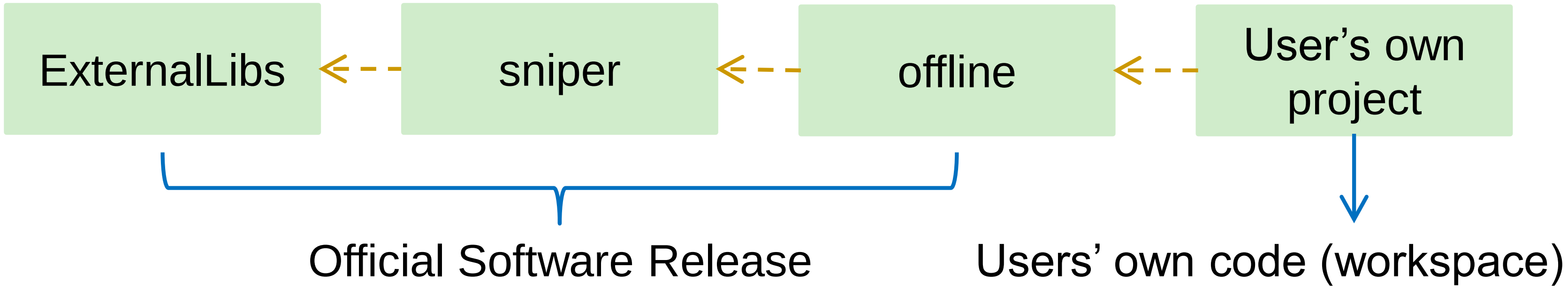
Create an Algorithm and a Service

- 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
```

Preparation



ExternalLibs

sniper

offline

User’s own project

Official Software Release

Users’ own code (workspace)

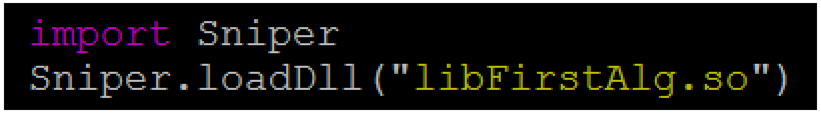
1. Setup the official release environment
   1. $ source ~/juno-dev/setup.sh
2. Create your own project
   1. $ cmt create_project Tutorial
   2. $ cd Tutorial
   3. $ vi cmt/project.cmt (→ use offline)
   4. $ vi cmt/version.cmt (→ v0)
3. Create your own package
   1. $ cmt create MyPackage v0
   2. $ source MyPackage/cmt/setup.sh

Package Management

1. Create a new package with CMT
   - $ cmt create TestAlg v0
2. Orgnization of subdirectory and files
   1. Subdirectory cmt/
      - File requirements: tell CMT how to setup and compile this package
   2. Subdirectory src/: the directory for source code (C++)
   3. Subdirectory FirstAlg: an optional directory for header files to share
   4. Subdirectory python/: an optional directory for python code
   5. Subdirectory share/: an optional directory for scripts of tutorial
   6. Subdirectory Linux-x86_64 or anything like this: the compiling results that automatically generated by CMT

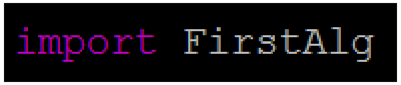
Coding and Running

- 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



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

vs.



```
import FirstAlg
```

- Comple and run the example
  - $ cmt make ## in any subdirectory of the package (cmt/ recommended)
  - $ python run.py ## details in run.py

CMT Configuration

- The package name and author

  package SecondAlg

  Author Zou Jiaheng <zoujh@ihep.ac.cn>   ## optional

- Dependencies while compiling

```
use SniperKernel        v*
use FirstSvc            v*     FirstToy
```

- How to generate the .so library file

  library SecondAlg *.cc

  apply_pattern linker_library library=SecondAlg   ##Whether load all dependencies automatically while loading this library. Some times it is not necessary

- Copy C++ headers and Python into CMT InstallArea

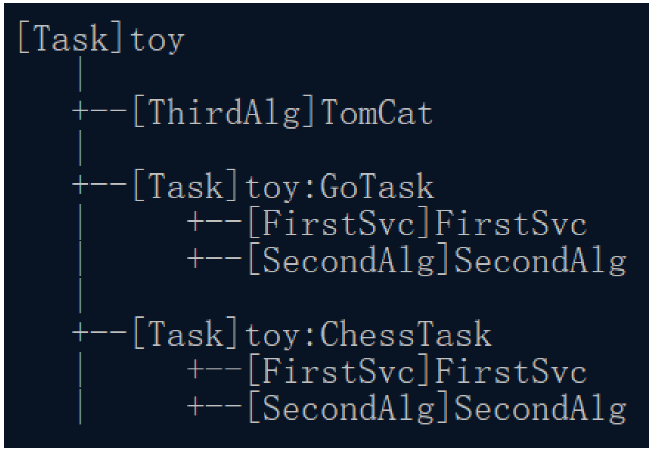
  apply_pattern install_more_includes more=FirstSvc   ## unnecessary if no shared headers

  apply_pattern install_python_modules

Advanced Topic: multiple-tasks job

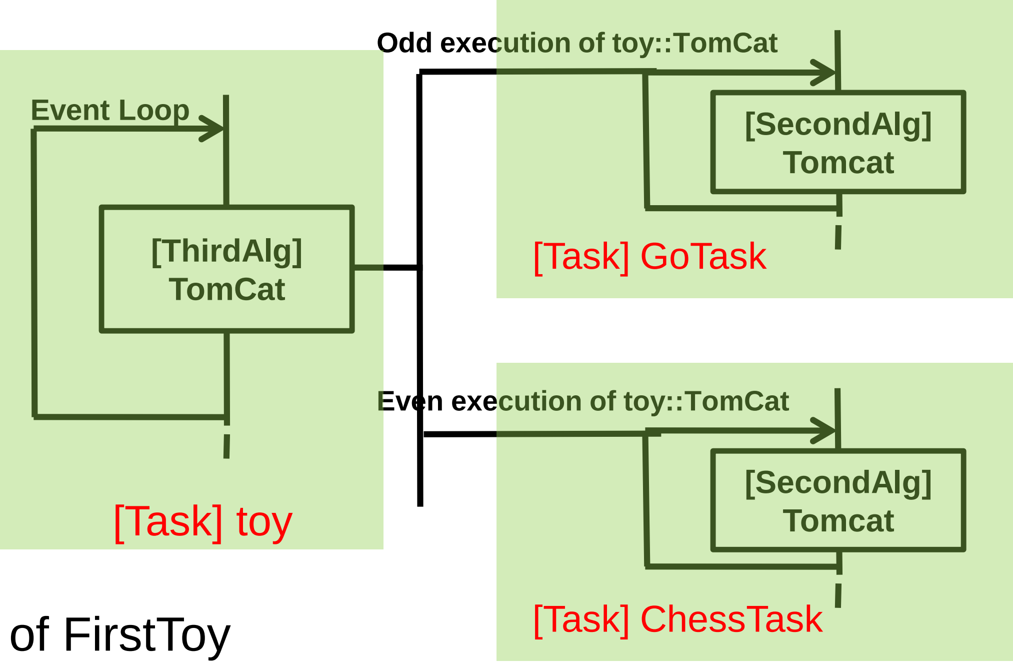
The DLElement Map of

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

Thanks !

Any questions?

Accounts

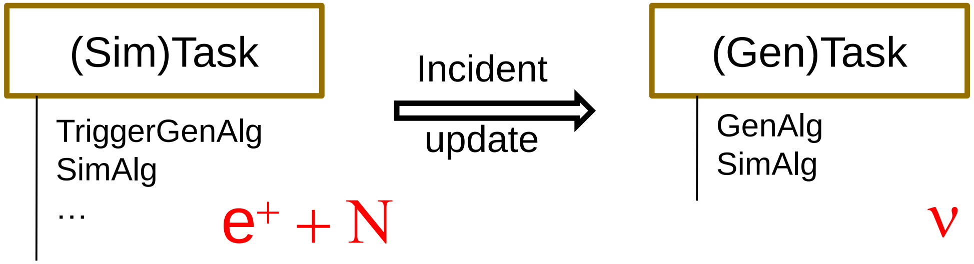
- AFS (IHEP computer cluster) account
  - http://afsapply.ihep.ac.cn:86/ccapply/userapplyaction.action
- JUNO SVN account
  - We use subversion as the version control system
  - A public read only account: juno/jiangmen
  - A personal account is necessary for updating purpose
    - Register an account in juno trac first: http://juno.ihep.ac.cn/trac/
    - Send email to lintao@ihep.ac.cn or maqm@ihep.ac.cn
      - Your user name in trac
      - Your affiliation (institute or university)

Possible Use Cases of Multi-Task Job

1, Multi I/O streams, such as background mixing

- create a Task for each I/O stream
- each Task holds its own data memory
- each Task handles only one Input (and Output) stream
- I/O service can be much simplified

2, Event amount changed, such as IBD simulation



(Sim)Task

TriggerGenAlg
SimAlg
…

$e^+ + N$

Incident

update

(Gen)Task

GenAlg
SimAlg

ν

3, Multi-Thread Computing (run each task in an individual thread)