XAOD WORKSHOP SUMMARY
DAY 2

Peilong Wang
March 29, 2016

Topic

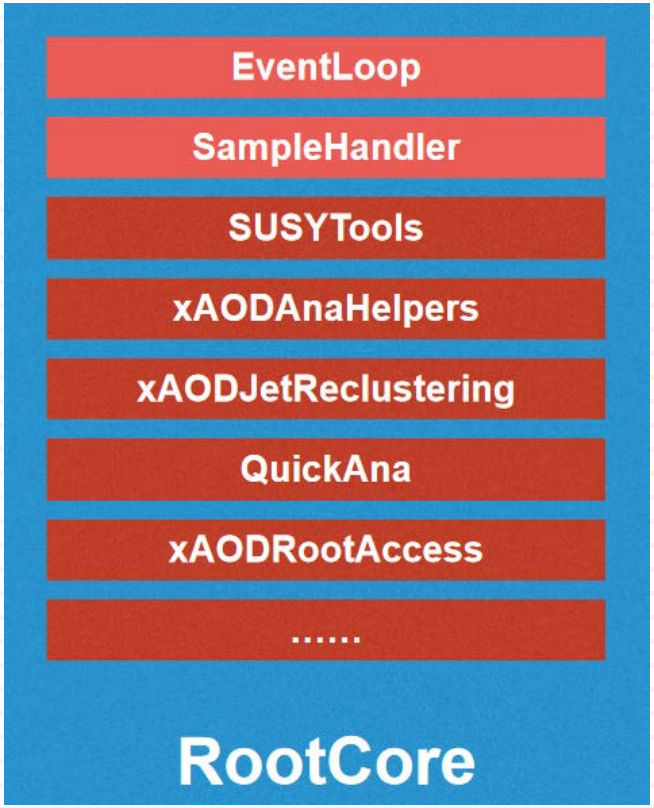
- RootCore Basics
- Derivation Framework
- Analysis Framework
- Athena
- Event Displays

RootCore Basics

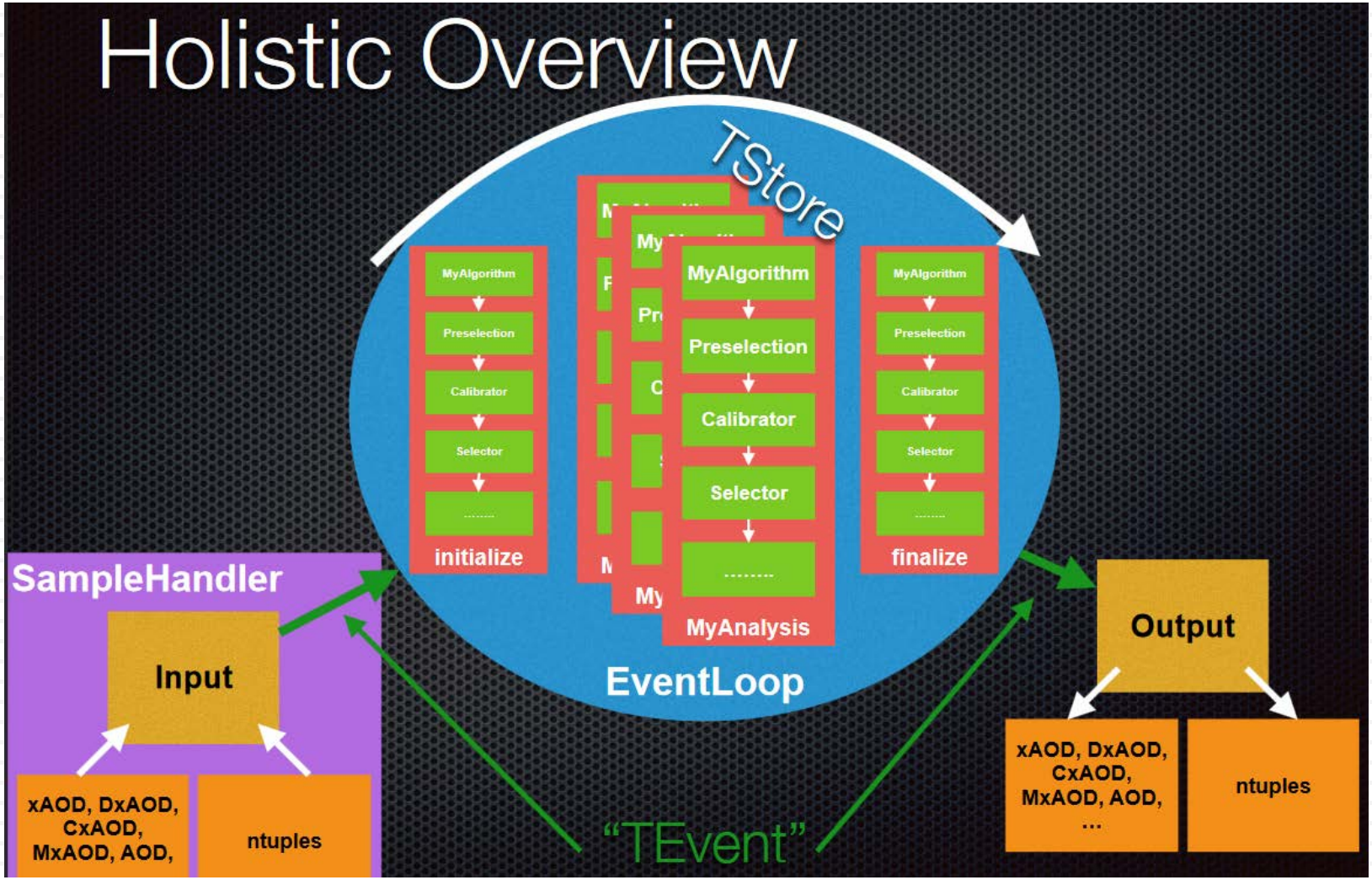
- What is RootCore
- Overview of the RootCore
- EventLoop
  - What is EventLoop
  - How to set up
  - How to run it

What is RootCore

- RootCore is a package manager
  - It knows about analysis release
    - Base, SUSY, Top
  - I knows how to compile packages
  - It knows where to find packages from SVN



Overview of RootCore



EventLoop

- A package helps user to run his own loop algorithm
- Set up
  - Header
  - initialize()
  - execute()
- How to run it
  - C++ macro

Header

```
private:
  xAOD::TEvent *m_event; //!
```

```
EL::StatusCode MyAlgorithm :: initialize ()
{
  m_event = wk()->xaodEvent();
  // further initialization stuff
  return EL::StatusCode::SUCCESS;
};
```

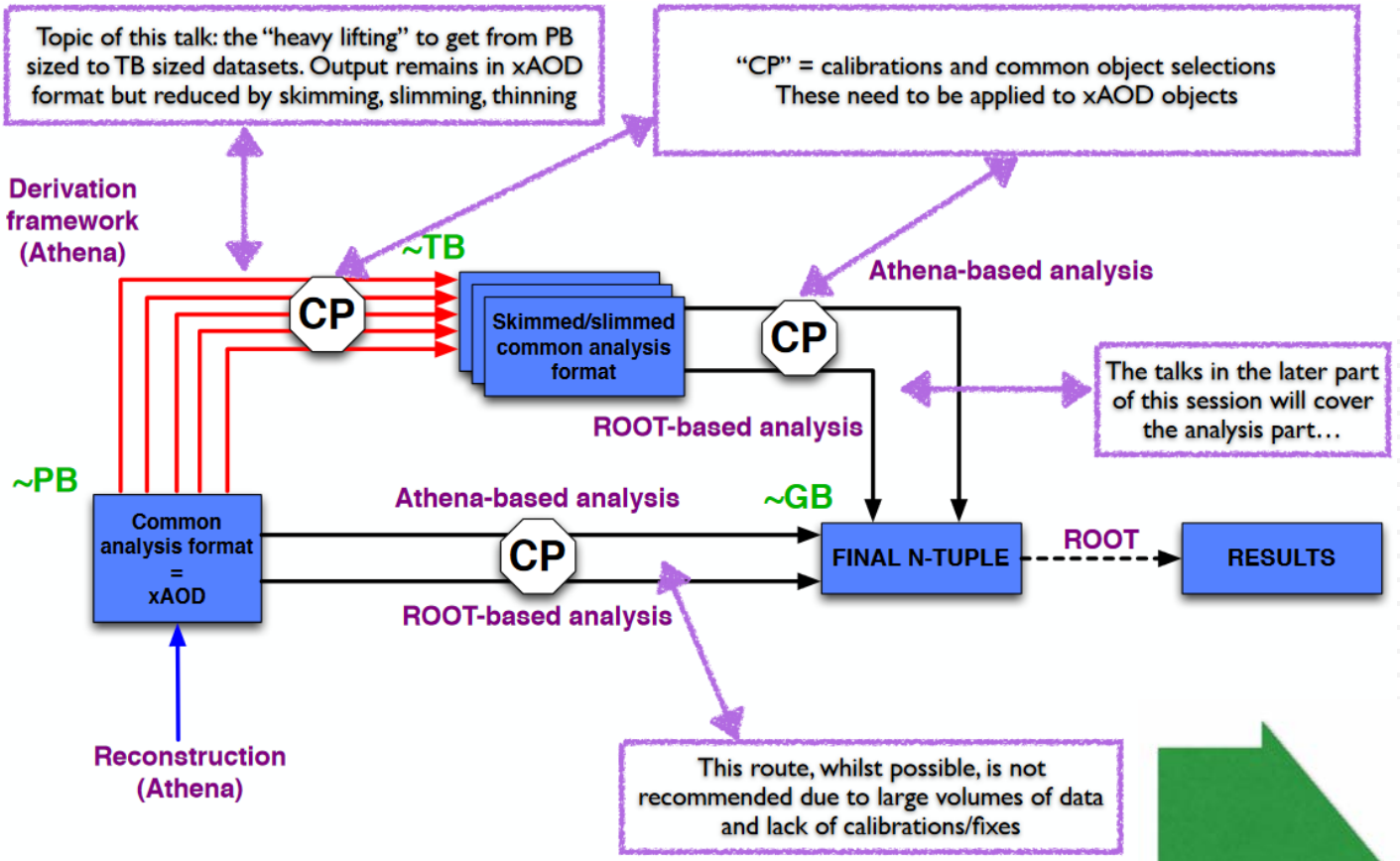
```
EL::StatusCode MyAlgorithm :: execute ()
{
  xAOD::JetContainer* jets(nullptr);
  if(!m_event->retrieve(jets, "AntiKt4EMTopoJets").isSuccess()){
    std::cout << "Could not retrieve AntiKt4EMTopoJets from TEvent" << std::endl;
    return EL::StatusCode::FAILURE;
  }

  std::cout << "Analyzing Event" << std::endl;
  for(const auto& jet: *jets){
    std::cout << "\tJet pT = " << jet->pt()/1.e3 << " GeV" << std::endl;
  }
 return EL::StatusCode::SUCCESS;
};
```

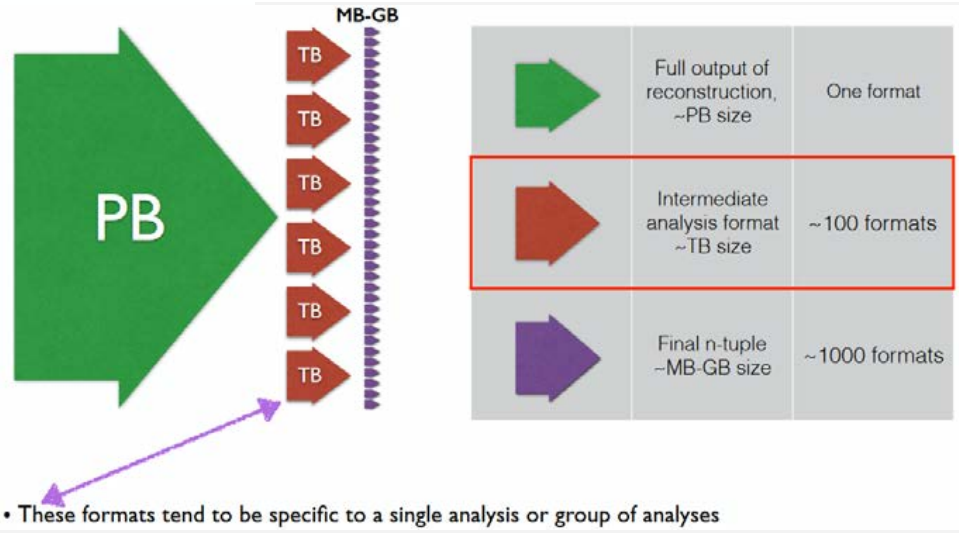
It can also be set up using python

Derivation Framework

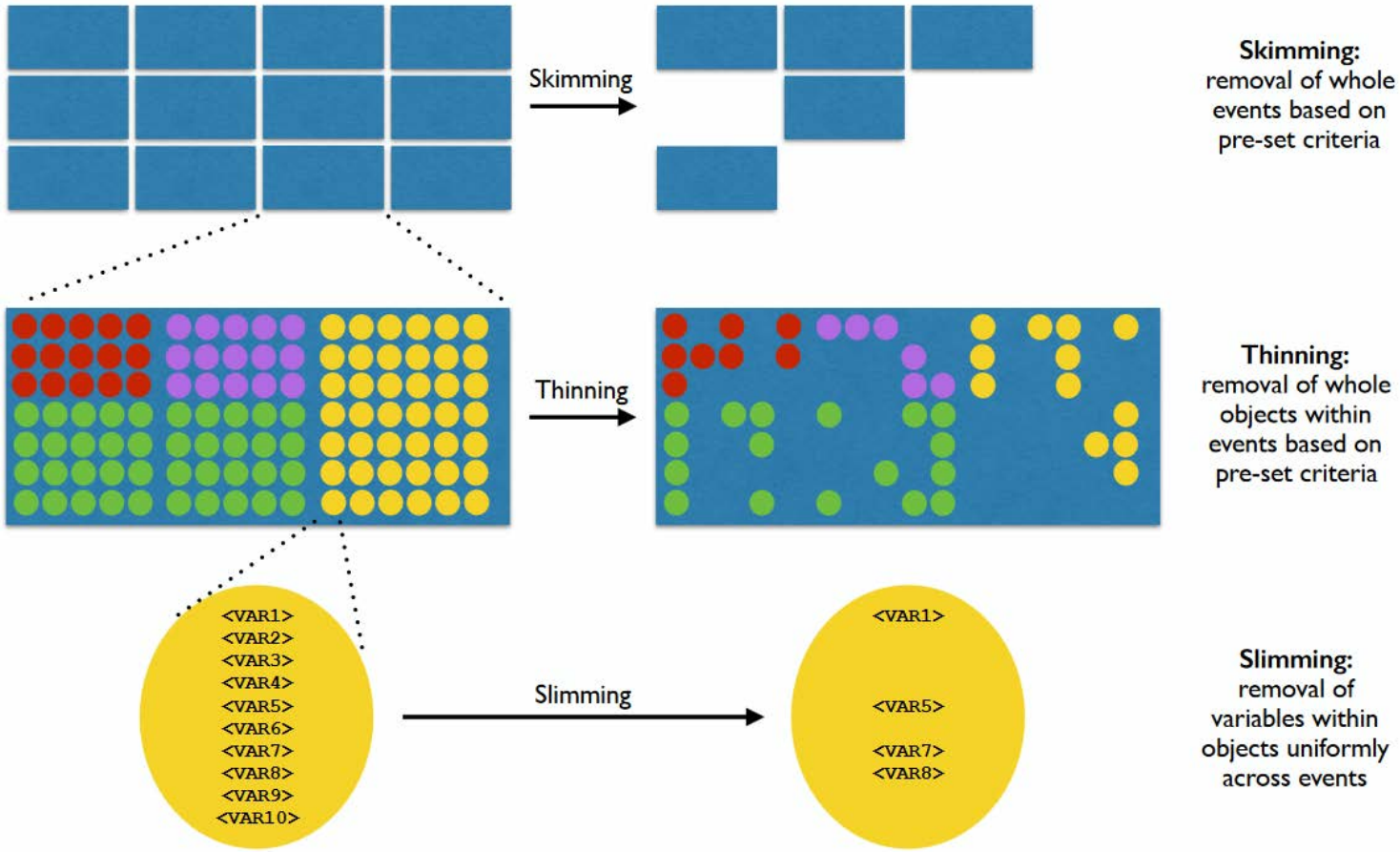
- Helps user get specific data locally (tier3)
- Data reduction operations
  - Skimming
  - Thinning
  - Slimming

The data we need





Data reduction operations



Events (slimming):

- `count(Muons.pt > 25.0*GeV && Muons.eta < 2.5) >= 4`

Objects (thinning):

- `InDetTrackParticles.pt > 5.0*GeV`

Analysis Framework

- Avaible Frameworks
  - CxAOD, HWW PxAOD, AnalysisSUSY, QuickAna…
- QuickAna
  - Design Goals
  - ANA_CHECK macro
- xAODAnaHelpers
  - Design Goals

QuickAna – Tool Scheduler

- Runs all combined performance tools (CP tools)
- Highlight the code that makes the physics decisions
  - Written in simple and straightforward C++
  - Following closely the actual physics logic
  - Understandable without looking at infrastructure code
- Use xAOD objects, stores all information in StoreGate/TStore
- Works in both RootCore and Athena
- Used by SUSY, Top and several groups in Higgs

ANA_CHECK macro

- new in this iteration of the tutorial: ANA_CHECK() macro
  - it works like CHECK(), ATH_CHECK(), etc.
  - however: ANA_CHECK works with any status code (and some other types as well)
- you can call

```
ANA_CHECK (someFunction(x,y,z));
```

  and it will abort the current function if the call fails
- and you can set the return type as well, by putting this at the beginning of your function:

```
ANA_CHECK_SET_TYPE (EL::StatusCode);
```

- you also put it in non-member functions, if you add

```
using namespace asg::msgUserCode;
```

  (you can also define your own message categories)

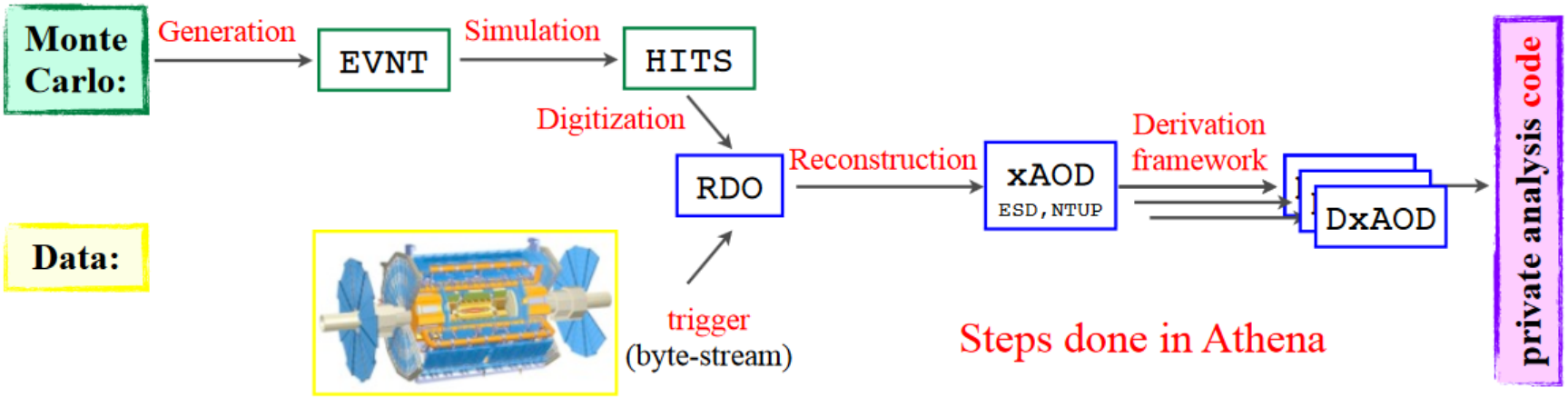
xAODAnaHelpers

- Designed to be the minimal needed to use the CP tools properly to calibrate, select, and correct the physics objects used for most physics analyses
- Python based

Athena

- Where you can use Athena
- Package structure
- EventLoop -> Athena
- CP Tools development for Athena
  - ASG Tools
  - Abstract & Concrete Classes

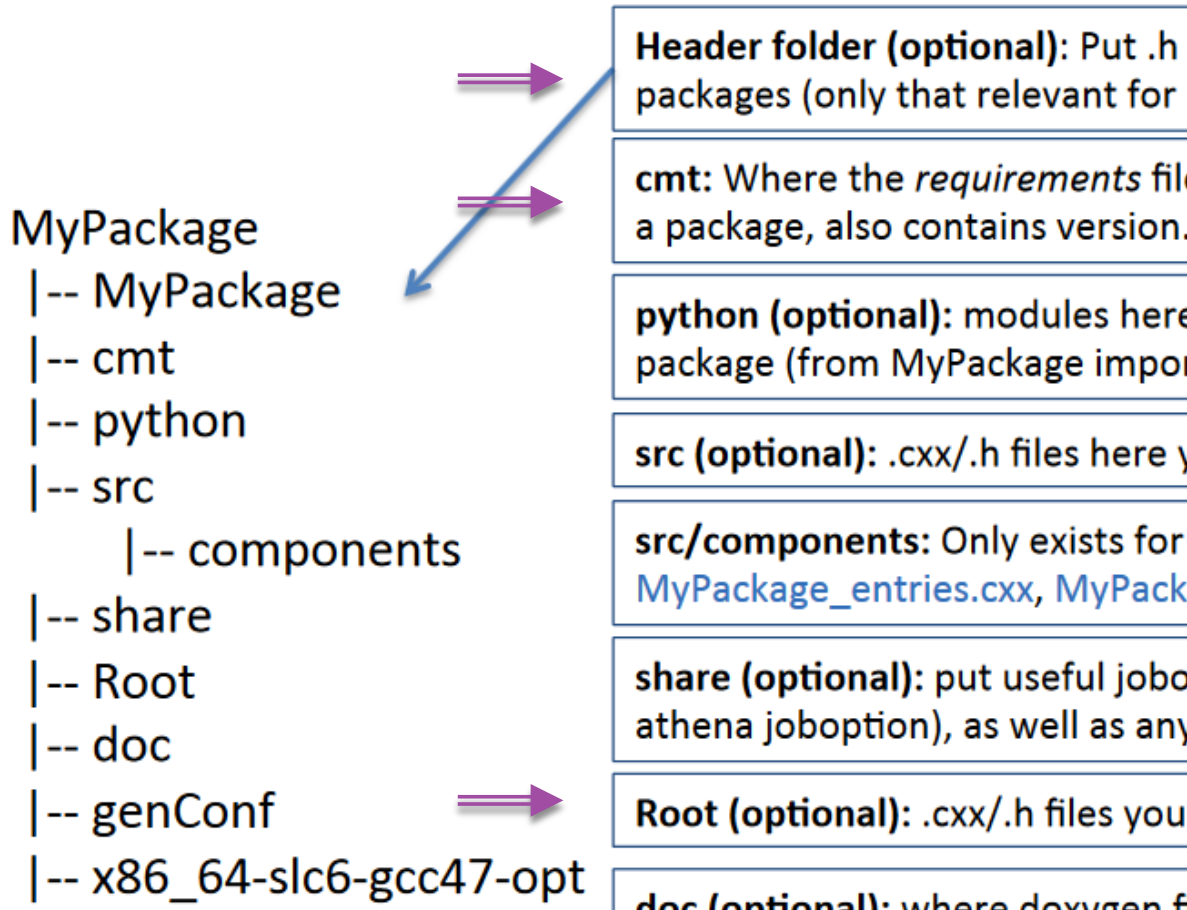
Where you can use Athena



- many ATLAS activities done exclusively in Athena
  - detector and object reconstruction (detector activities, CP groups, fancy physics search requiring detector information)
  - MC simulation, digitization, and reconstruction
  - derivation framework
- other steps are optional:
  - final physics analysis code

Good to know your way around Athena

Package Structure



Header folder (optional): Put .h files here that you would want to include from other packages (only that relevant for installed or dual_use). Folder added to INCLUDEPATH

cmt: Where the *requirements* file lives (also Makefile.RootCore). When you "cmt co" a package, also contains version.cmt, which has version in it

python (optional): modules here that you want to belong to MyPackage python package (from MyPackage import MyModule)

src (optional): .cxx/.h files here you don't want RootCore or other packages to see

src/components: Only exists for libraries with Configurables. Contains two files: MyPackage_entries.cxx, MyPackage_load.cxx

share (optional): put useful joboptions here (can be used with 'include' function in athena joboption), as well as any configuration files

Root (optional): .cxx/.h files you want RootCore+cmt to see (but not other packages)

doc (optional): where doxygen files go

Build folder (auto-generated): where .so files are created. Folder name determined by $CMTCONFIG

genConf (auto-generated): where python wrappers (classes) for Configurables are created by cmt

EventLoop versus Athena

Steering Macro

- define and configure a **job** (inputs/outputs/evts)
- configure **algs** for job

EL::Algorithm

- create and configure tools
- *initialize()*

execute()

finalize()

EventLoop

Athena

Job options

- create and configure **algs** for job
- configure tools
- configure services

AthAlgorithm

- create and configure tools
- *initialize()*

execute()

finalize()

CP tools development -- ASG

- A class with a collection of functions
- Configurable through properties
- Dual-use, can be run in Athena and ROOT

- *cmt/* - where the **compilation configuration** is for both Athena and RootCore
- *MyPackage/* - contains the **public header files**
- *Root/* - contains **C++ code**
- *src/* - contains code that is exclusively for Athena only

```
setupATLAS
lsetup asetup
asetup 2.3.45,AthAnalysisBase,here
cmt new_pkg MyPackage
cd MyPackage
acmd.py cmt new-asgtool MyTool
cmt find_packages
cmt compile
```

Abstract and Concrete Classes

- *IMyTool.h* – this is what's known as a **abstract class**
  - Also known as an **interface**
  - It **defines what functions are publically provided** for a user to use, so internally-used private functions should not be listed here
  - This **cannot be changed without potentially breaking downstream code** (code that uses your tool), as it changes how the tool 'looks'
- *MyTool.h* – this is the header for the **concrete class**
  - This is a **normal header file** that contains all the functions you create (both private and public) in your tool as well as the data members
  - MyTool inherits from IMyTool
  - It also **must contain all functions that are in the abstract class**

- Abstract class, defines an interface for subsequent classes to override.
- Cant create objects of an abstract class
- Classes inherit from an abstract class can

Abstract class – code example

Typically interfaces begin with I and will inherit from IAsgTool

```
class IMyTool : public virtual asg::IAsgTool {
   public:
     ASG_TOOL_INTERFACE( IMyTool )
     virtual bool isSelected(const xAOD::IParticle&
particle, bool someAdditionalArgument = false) const = 0;
};
```

MyPackage/IMyTool.h

Necessary macro

Functions in the interface must be 'virtual' and be set to = 0

Inherits from interface and AsgTool

```
class MyTool: public asg::AsgTool, public virtual IMyTool {
public:
   ASG_TOOL_CLASS( MyTool , IMyTool )
   MyTool( const std::string& name );
   virtual StatusCode initialize() override;
   virtual bool isSelected(const xAOD::IParticle& particle,
bool someAdditionalArgument) const override;
   double m_nProperty;
};
```

MyPackage/MyTool.h

Necessary macro

Declares constructor and initialize()

Overrides the function that was declared in the interface

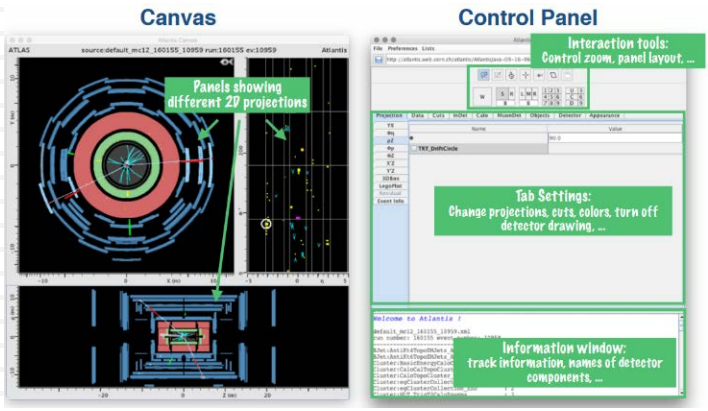
double to store property

Event Displays

- Atlantis (2D)
  - Generate an XML
  - Event Viewing GUI
- Virtual Point (3D)
  - Get event numbers and run numbers
  - Lookup lumiblock information
  - Find RAW files
  - Produce ESD files
  - Running VP1

Atlantis

- Step 1: Generate an XML file with interesting events
  - "JiveXML" in Athena, typically run on the grid
  - RAW, ESD, AOD, or xAOD
- Step 2: Open the XML in Atlantis event viewer
  - Java application, runs locally
  - Format event displays as desired
  - Save output images



Step 1: Generating XML

* Example grid submission generating XML from data:

```
ssh -Y lxplus.cern.ch
setupATLAS
asetup 17.2.4.10,slc5,here
lsetup panda

get_files JiveXML_jobOptions_PhysicsRAW.py

pathena \
  JiveXML_jobOptions_PhysicsRAW.py \
  --eventPickEventList rrr.txt \
  --eventPickDataType RAW \
  --outDS user.NICKNAME.JiveXMLTest \
  --extOutFile "JiveXML*.xml" \
  --eventPickStreamName physics_JetTauEtMiss
```

Use "RAW", "ESD", or "AOD"

File with the run numbers and event numbers that you want to save, e.g.

213754 232636371
205112 37740915
...

Change to your stream, e.g.
"physics_egamma"
"physics_Bphysics"

* Download grid job output with "dq2-get" or "rucio get"

* NOTE: If pathena returns a warning that the data needs to be copied from tape to disk, then wait for an automated email and rerun with the extra command `--eventPickStagedDS=LinkFromEmail` where `LinkFromEmail` looks something like `user.NICKNAME.67efrrr-28sik-9d-f11/`

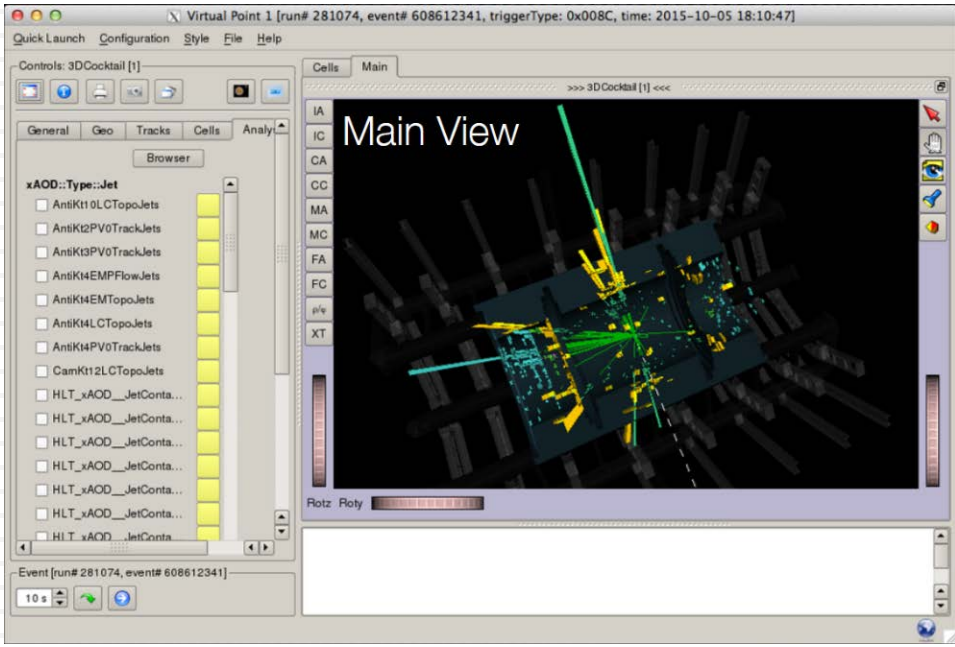
Step 2: Event Viewing GUI

- Several ways to launch the GUI…
  - Start it locally by downloading Java WebStart, right click → open
    - You may need to modify your security settings
  - Start it locally using zip/tgz archives
    - You'll need to install Java JDK if you don't have it already
      - Check by typing "`java -version`", want version ≥ 1.6.0
    - Unpack archive and `cd` into the top folder
    - run "`java -jar atlantis.jar`"
  - Start it from athena environment, e.g. from lxplus (with X11 forwarding)
    - "`lsetup atlantis`" followed by "`runAtlantis`"
- GUI will open with tutorial MC events, default color config

Virtual Point (3D)

A little bit complex

- Virtual Point (3D)
- Get event numbers and run numbers
- Lookup lumiblock information
- Find RAW files
- Produce ESD files
- Running VP1



The end