# ICTP-NCP SCHOOL ON LHC PHYSICS

# CMSSW Tutorial

*Taimoor Khurshid*

National Centre for Physics
Quaid-i-Azam University Campus
Islamabad

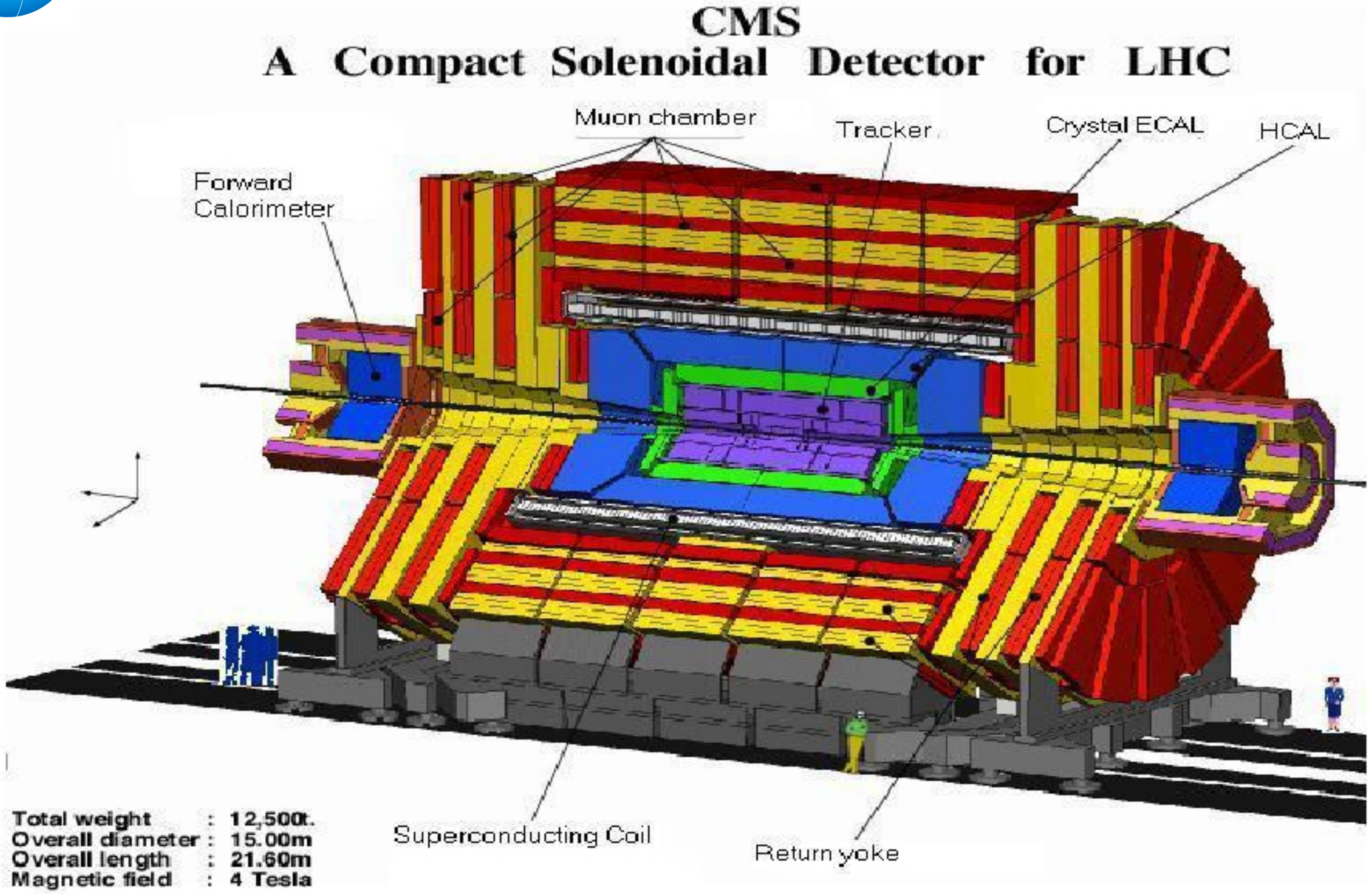
## Out Line

*Attendees should understand the basics of C++ programming and a Linux operating System.*

*This talk includes:*

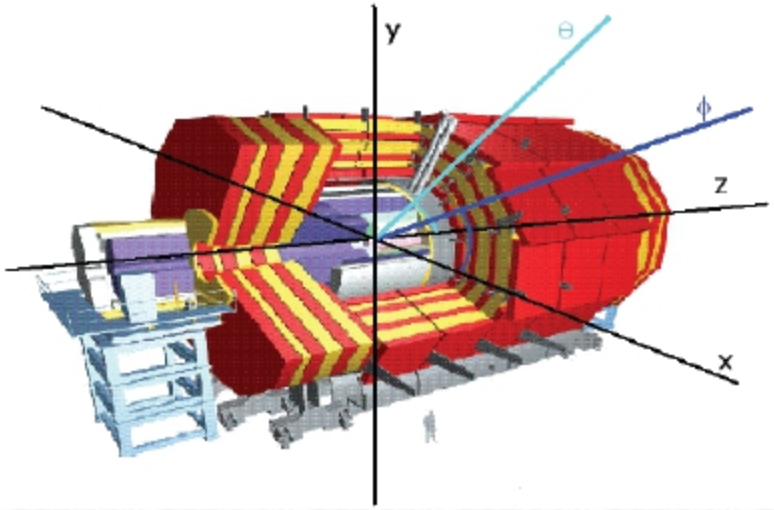
- *A bit about CMS Experiment*
- *Old CMS software Structure*
- *CMSSW*
    - *Data Formats*
- *Bare root analysis*
- *CMSSW Environment setup*
- *EDAnalyzr*
- *Example*

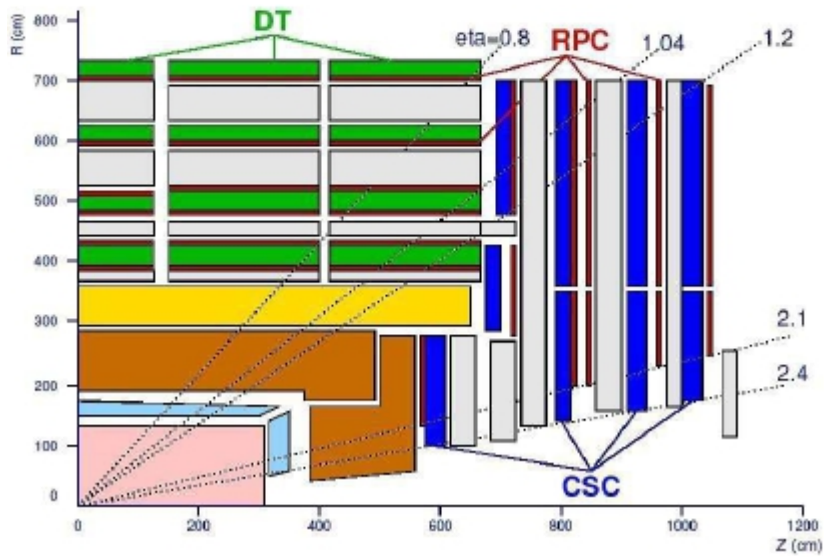
# CMS Experiment



| | | |
|---|---|---|
| Total weight | : | 12,500t. |
| Overall diameter | : | 15.00m |
| Overall length | : | 21.60m |
| Magnetic field | : | 4 Tesla |

# CMS Co-ordinates and conventions





- Azimuthal angle:

  $\phi$ = azimuthal angle
  $-\pi < \phi < \pi$

- Polar angle:

  $\theta$= polar angle
  $0 \leq \theta \leq \pi$
  Also $\eta = \ln[\tan(\theta/2)]$

  $p_T = |\mathbf{p}| \sin\theta$
  $\Delta R = \sqrt{(\Delta \eta)^2 + (\Delta\varphi)^2}$

- Energy is measured in GeV, momentum in GeV/c and mass in GeV/$c^2$

- Distance and position in cm

- Time in ns.

# Analysis Chain

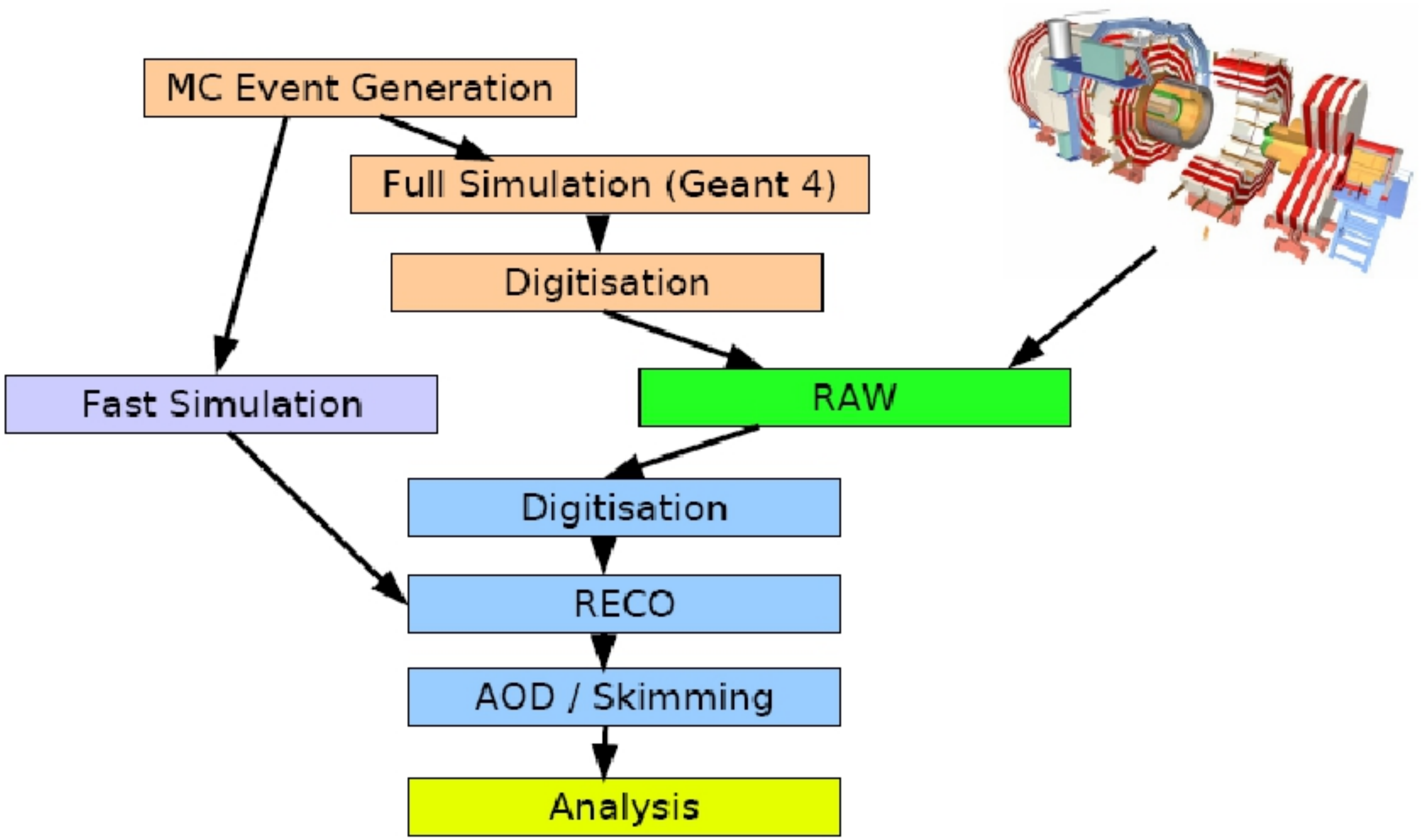
Initially one needs to choose a physics problem

| Evt Format | Content | Purpose | Evt Size(MB) |
|---|---|---|---|
| GEN | Generated Monte Carlo event | - | - |
| SIM | Energy depositions of MC particles in detector (sim hits). | - | - |
| DIGI | Sim hits converted into detector response. Basically the same as the RAW output of the detector. | - | 1.5 |
| DAQ-RAW | Detector data from front end electronics + L1 trigger result. | Primary record of physics event. Input to online HLT | 1-1.5 |
| RAW | The L1 trigger result, the result of the HLT selections (HLT trigger bits), potentially some of the higher level quantities calculated during HLT processing. | Input to Tier-0 reconstruction. Primary archive of events at CERN. | 1.5 |
| RECO | Reconstructed objects (tracks, vertices, jets, electrons, muons, etc.) and reconstructed hits/clusters | Output of Tier-0 reconstruction and subsequent re-reconstruction passes. Supports re-finding of tracks, etc. | 0.25 |
| AOD | Subset of RECO. Reconstructed objects (tracks, vertices, jets, electrons, muons, etc.). Possible small quantities of very localized hit information. | Physics analysis, limited refitting of tracks and clusters | 0.05 |

# LHC Event

# From Data to Physics

## The general data flow



# Old CMS Software

CMS software structured in "projects" focussing on different tasks:

ORCA (Object Oriented Reconstruction and Analysis)
(*http://cmsdoc.cern.ch/orca/*)
Reconstruction (and simulation of electronics)

OSCAR (Object oriented Simulation for CMS Analysis and Reconstruction )
(*http://cmsdoc.cern.ch/oscar*)
Simulation with Geant-4

COBRA (Coherent Object-oriented Base for Reconstruction, Analysis and simulation )
(*http://cobra.web.cern.ch/cobra/*)
Framework: Interface to basic services

FAMOS (CMS Fast Simulation)
(*http://cmsdoc.cern.ch/famos/*)
Fast simulation and reconstruction

IGUANA (Interactive Graphics For User Analysis)
(*http://iguana.web.cern.ch/iguana/*)
Framework for visualization

IGUANACMS (Interactive Graphics and User Analysis for CMS )
(*http://iguanacms.web.cern.ch/iguanacms/*)
Visualization (e.g. event display)

Geometry (CMS Geometry Project)
(*http://cmsdoc.cern.ch/cms/software/geometry/index.html*)
XML description of the CMS detector

# CMS SoftWare

The CMS SoftWare uses *one single executable* for **everything**

→ **cmsRun** *<config file>*

•This includes

- Online data-taking
- Online high-level trigger *(HLT)*

- Monte Carlo event generation
- Detector simulation *(full, fast)*

- Reconstruction
- Analysis

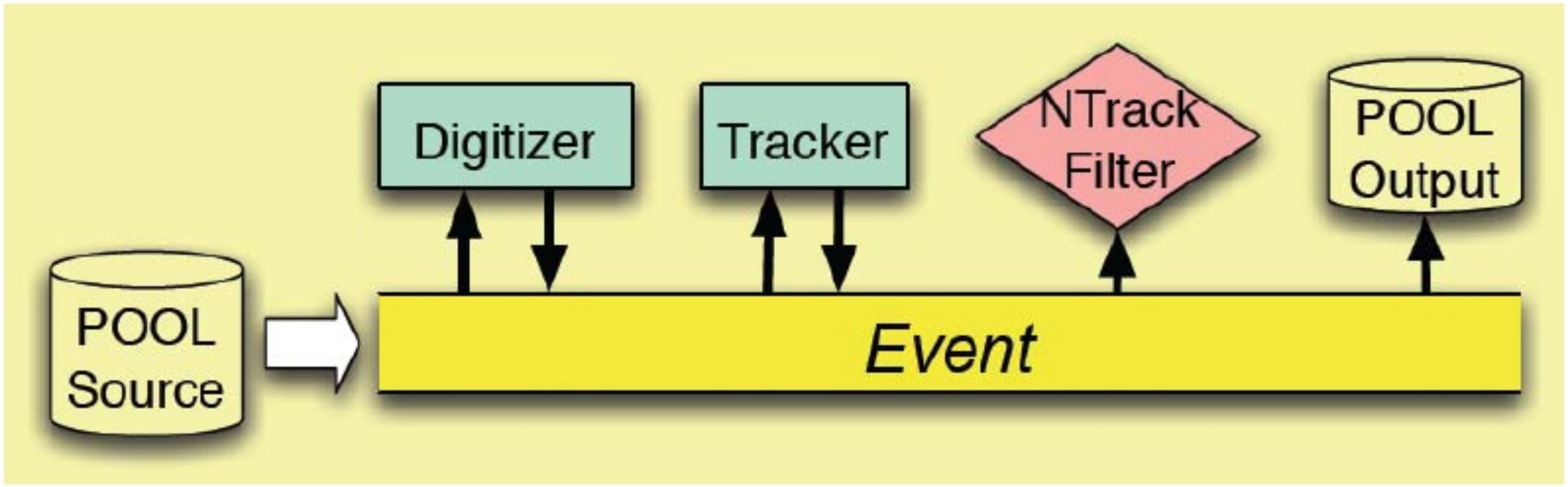
# Data flow in CMSSW

CMSSW is composed entirely on plug-ins or modules:

- *Source*
- *EDProducer*
- *EDFilter*
- *EDAnalyzer*
- *Output Module*
- ……

*Advanced: a module is an instantiation of a C++ class, i.e. an* ***object.***

**Communication between modules happens only via event!**



# The Data Format

The main CMSSW Data format is:

- It is called "Event Data Model" (EDM)
- Based of regular root tree
  - Content viewable by *TBrowser*
- Stores complex C++ objects as hierarchical branches
  - By loading so called dictionaries, ROOT becomes aware of C++ class details, called DataFormates.
- Contains two main TTrees
  - Run
  - Events
- Contains metadata, the "provenance".

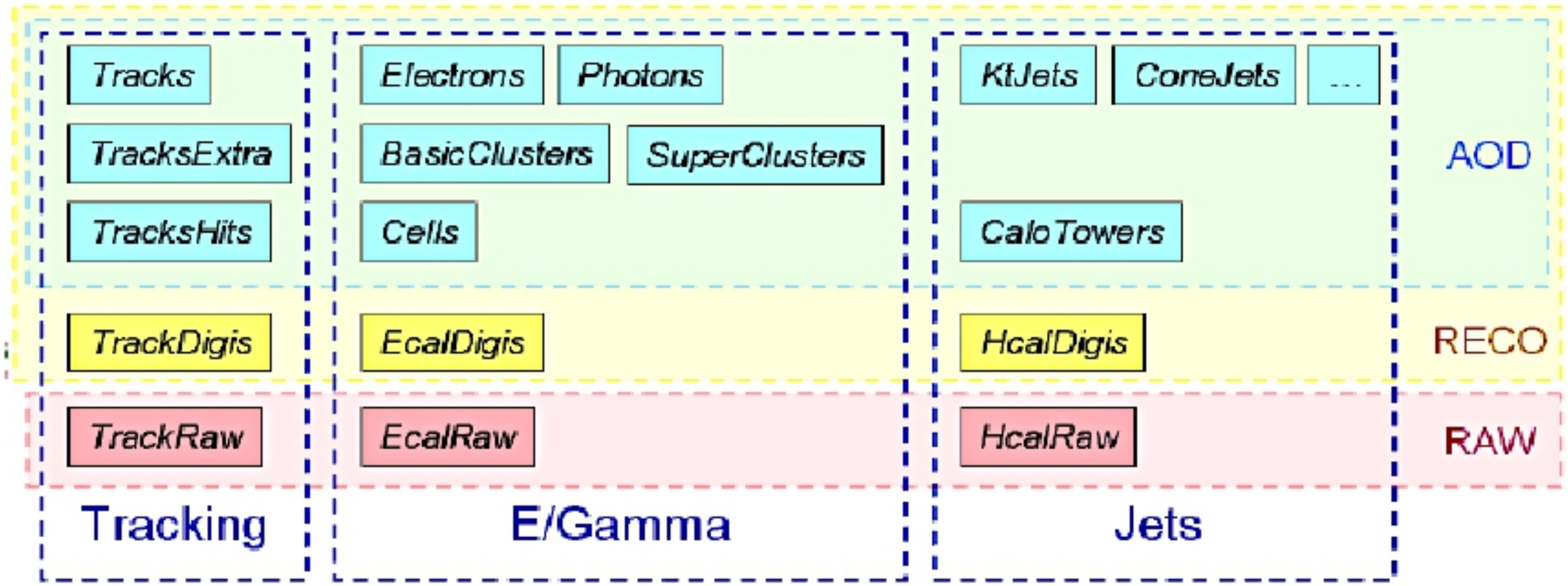
# EDM Contents

What is stored in the event files?

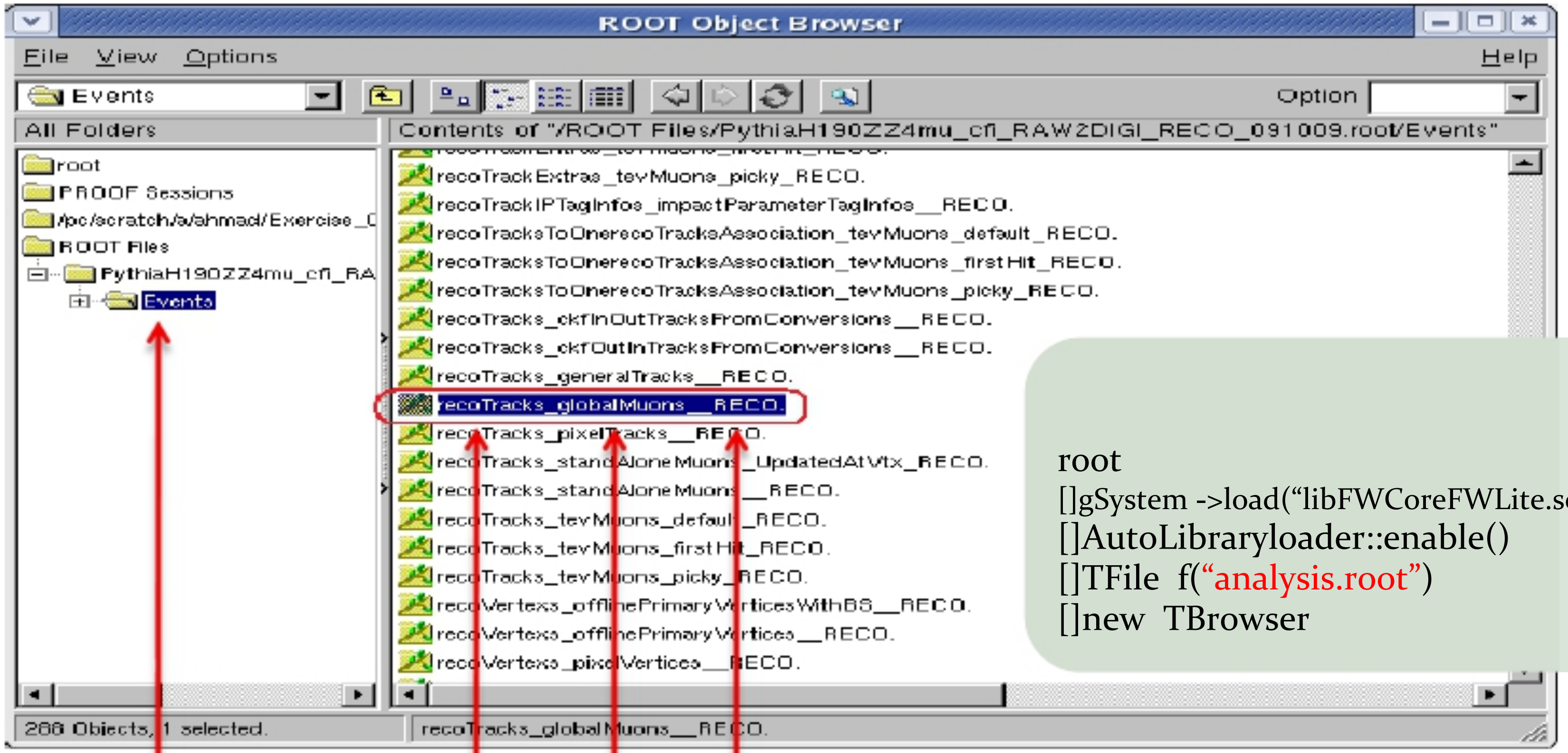
•A collection of products generated by CMSSW producers

•These products are grouped into useful categories"

- •FEVT:  Full Event
- •RECO: RECOnstruction
- •RECOSIM: RECOnstruction + selected simulation information
- •AOD: Analysis Object Data (a compact subject of RECO fromat)
- •AODSIM: AOD + generator information



# Opening EDM Files with bare root



Contents of "/ROOT Files/PythiaH190ZZ4mu_cfi_RAW2DIGI_RECO_091009.root/Events"

root
PROOF Sessions
/pc/scratch/a/ahmad/Exercise_C
ROOT Files
PythiaH190ZZ4mu_cfi_RA
Events

recoTrackExtras_tevMuons_picky_RECO.
recoTrackIPTagInfos_impactParameterTagInfos__RECO.
recoTracksToOnerecoTracksAssociation_tevMuons_default_RECO.
recoTracksToOnerecoTracksAssociation_tevMuons_firstHit_RECO.
recoTracksToOnerecoTracksAssociation_tevMuons_picky_RECO.
recoTracks_ckfInOutTracksFromConversions__RECO.
recoTracks_ckfOutInTracksFromConversions__RECO.
recoTracks_generalTracks__RECO.
recoTracks_globalMuons__RECO.
recoTracks_pixelTracks__RECO.
recoTracks_standAloneMuons_UpdatedAtVtx_RECO.
recoTracks_standAloneMuons__RECO.
recoTracks_tevMuons_default_RECO.
recoTracks_tevMuons_firstHit_RECO.
recoTracks_tevMuons_picky_RECO.
recoVertexs_offlinePrimaryVerticesWithBS__RECO.
recoVertexs_offlinePrimaryVertices__RECO.
recoVertexs_pixelVertices__RECO.

288 Objects, 1 selected.    recoTracks_globalMuons__RECO.

```
root
[]gSystem ->load("libFWCoreFWLite.so")
[]AutoLibraryloader::enable()
[]TFile  f("analysis.root")
[]new  TBrowser
```

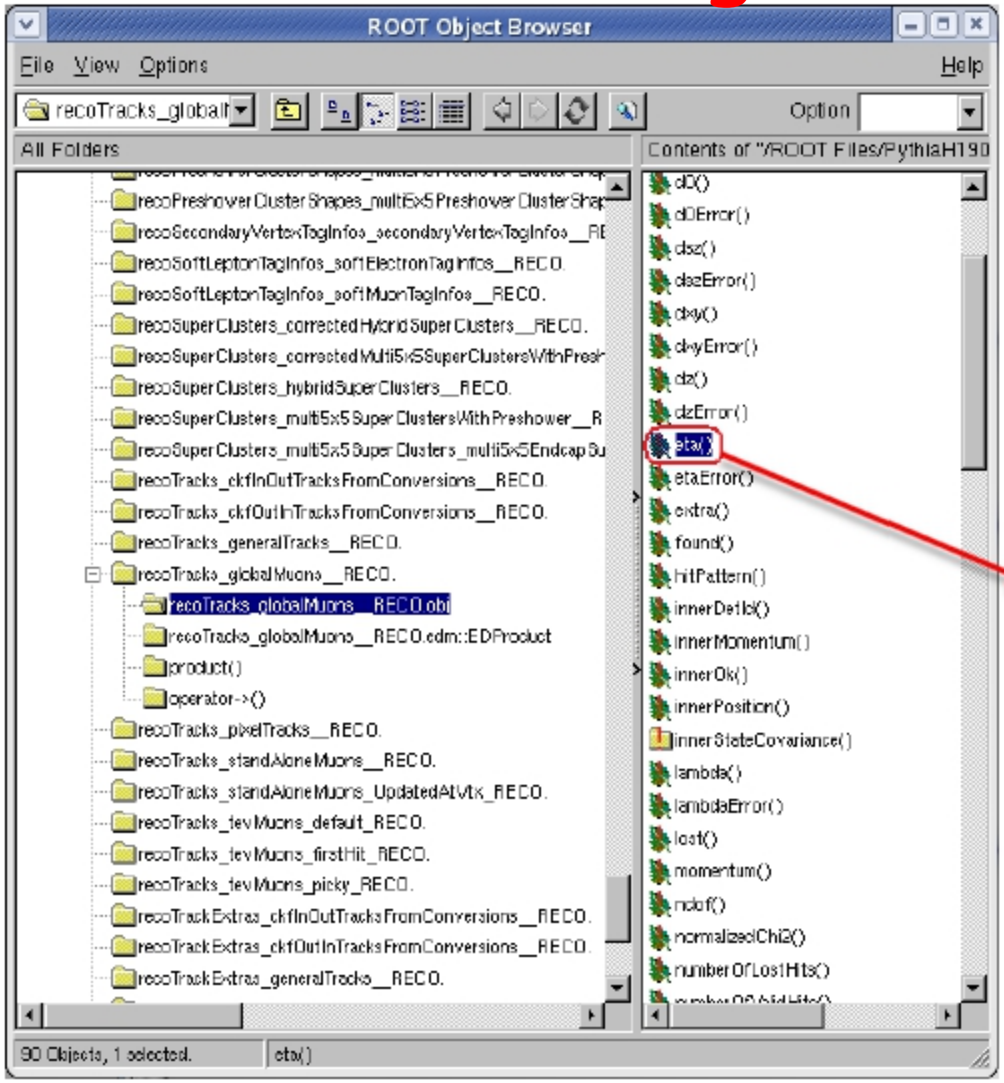
The "*Process*" name

The product "*label*"

The main EDM TTree

The C++ class name( Here: vector <reco::Track>)

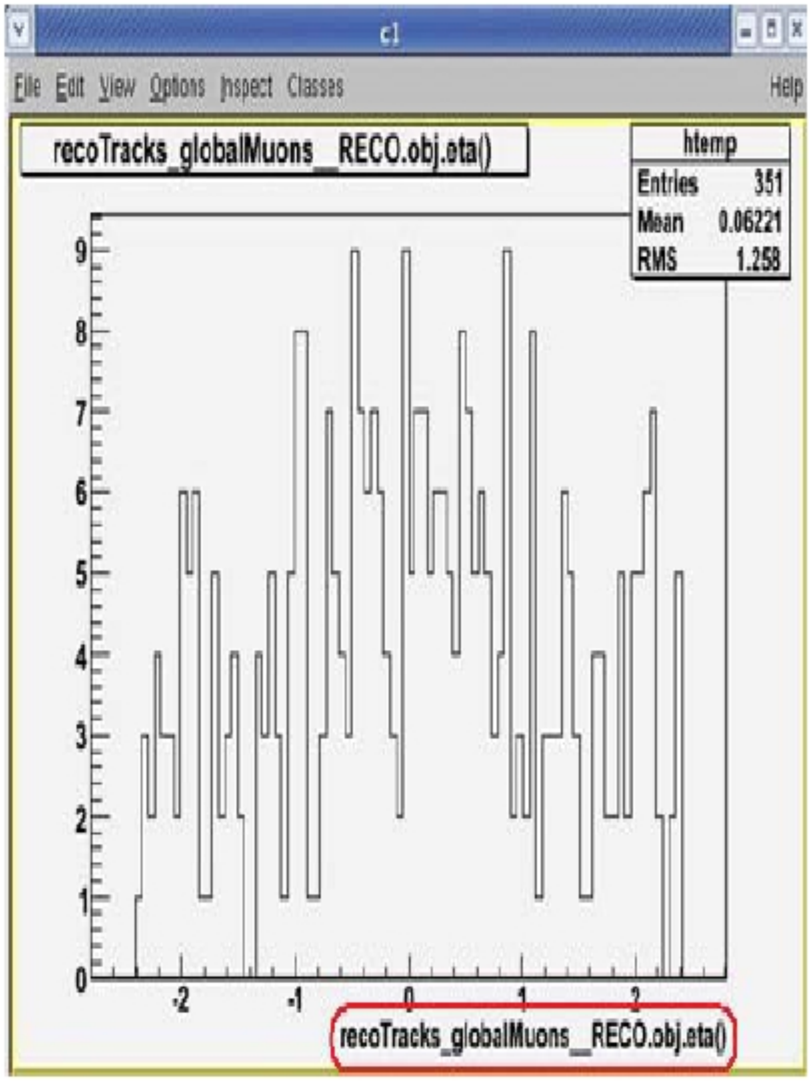
# Fwlite: Accessing C++ methods using dictionaries



```
recoPreshowerClusterShapes_multi5x5PreshowerClusterShap
recoSecondaryVertexTagInfos_secondaryVertexTagInfos__RE
recoSoftLeptonTagInfos_softElectronTagInfos__RECO.
recoSoftLeptonTagInfos_softMuonTagInfos__RECO.
recoSuperClusters_correctedHybridSuperClusters__RECO.
recoSuperClusters_correctedMulti5x5SuperClustersWithPresh
recoSuperClusters_hybridSuperClusters__RECO.
recoSuperClusters_multi5x5SuperClustersWithPreshower__R
recoSuperClusters_multi5x5SuperClusters_multi5x5EndcapSu
recoTracks_ckfInOutTracksFromConversions__RECO.
recoTracks_ckfOutInTracksFromConversions__RECO.
recoTracks_generalTracks__RECO.
recoTracks_globalMuons__RECO.
    recoTracks_globalMuons__RECO.obj
    recoTracks_globalMuons__RECO.edm::EDProduct
    product()
    operator->()
recoTracks_pixelTracks__RECO.
recoTracks_standAloneMuons__RECO.
recoTracks_standAloneMuons_UpdatedAtVtx_RECO.
recoTracks_tevMuons_default_RECO.
recoTracks_tevMuons_firstHit_RECO.
recoTracks_tevMuons_picky_RECO.
recoTrackExtras_ckfInOutTracksFromConversions__RECO.
recoTrackExtras_ckfOutInTracksFromConversions__RECO.
recoTrackExtras_generalTracks__RECO.
```

```
d0()
d0Error()
dsz()
dszError()
dxy()
dxyError()
dz()
dzError()
eta()
etaError()
extra()
found()
hitPattern()
innerDetId()
innerMomentum()
innerOk()
innerPosition()
innerStateCovariance()
lambda()
lambdaError()
lost()
momentum()
ndof()
normalizedChi2()
numberOfLostHits()
```

```
90 Objects, 1 selected.    eta()
```



## Using the root Tree

```
void analysis()
{
    gSystem ->load(“libFWCoreFWLite.so”);
    AutoLibraryloader::enable();
    Tfile  f(“analysis.root”);
    new  Tbrowser;
    new T1HF(“jetEta”, “jet eta” , 100 , -3 , 3”);
    Events->Draw(“recoCaloJets_sisCone5CaloJets__HLT.obj.eta() >> jetEta” ,
    “recoCaloJets_sisCone5CaloJets__HLT.obj.pt() > 50”);
}
```

Root analysis.C

- Very simple and very fast way to get useful plots.
  You can add histogram formatting, simple formulas, and many more things
  Read up on TTree::Draw() in the ROOT documentation.

- Can not replace a CMSSW C++ Analyzer,
  but can be used to generate the final plots for instance.

# Accessing Event Data I

As we have seen, contents of an event are organized in "products"

DataType_ModuleName_InstanceName_ProcessName

```
//  accessing event data from C++

//  by module and default product label
  Handle<CaloJetCollection> caloJets;
  iEvent.getByLabel("sisCone5CaloJets", Jets);

//  getting polymorphic collections via the base class
  Handle<View<Jet>> Jets;
  iEvent.getByLabel("sisCone5CaloJets", Jets);

//  by module and product instance label
  Handle<SimHitVector> simHits;
  iEvent.getByLabel("detsim", "pixel", simHits);

//  by type
  vector< Handle<SimHitVector> > allsimHits;
  iEvent.getByType(allsimHits);
```

# Accessing Event Data II

```
PythiaH190ZZ4mu_cfi_RAW2DIGI_RECO_091009.root
FEDRawDataCollection   "rawDataCollector"   ""   "HLT."
L1GlobalTriggerObjectMapRecord   "hltL1GtObjectMap"   ""   "HLT."
L1GlobalTriggerReadoutRecord   "hltGtDigis"   ""   "HLT.“
vector<reco::GenJet>   "iterativeCone5GenJets"   ""   "HLT."
vector<reco::GenJet>   "kt4GenJets"   ""   "HLT."
vector<reco::GenJet>   "kt6GenJets"   ""   "HLT."
vector<reco::GenJet>   "sisCone5GenJets"   ""   "HLT."
vector<reco::GenJet>   "sisCone7GenJets"   ""   "HLT."
vector<reco::GenMET>   "genMet"   ""   "HLT."
vector<reco::GenMET>   "genMetNoNuBSM"   ""   "HLT."
vector<reco::GenParticle>   "genParticles"   ""   "HLT."
trigger::TriggerEvent   "hltTriggerSummaryAOD"   ""   "HLT."
L1GlobalTriggerReadoutRecord   "gtDigis"   ""   "RECO."
L1MuGMTReadoutCollection   "gtDigis"   ""   "RECO."
```

```
iEvent.getByLabel(“sisCone5GenJets”, genJets);
```

Another option: running the “EventContentAnalyzer” CMSSW module

## scram

SCRAM = Software release and management tool

Used for managing the software,
compiling your own software,...

*Listing installation software*

```
$ scram list CMSSW
[...]
```

*Compiling packages in your source area*

```
$ cd CMSSW_5_3_13 /src
$ cmsenv
$ scram b
```

*Tip: add –j 6  to go faster*

## Setting up a CMSSW environment

*creating your local area*

```
$ cmsrel CMSSW_5_3_13
[...]
$ cd CMSSW_5_3_13 /src
```

*Setting runtime variables*

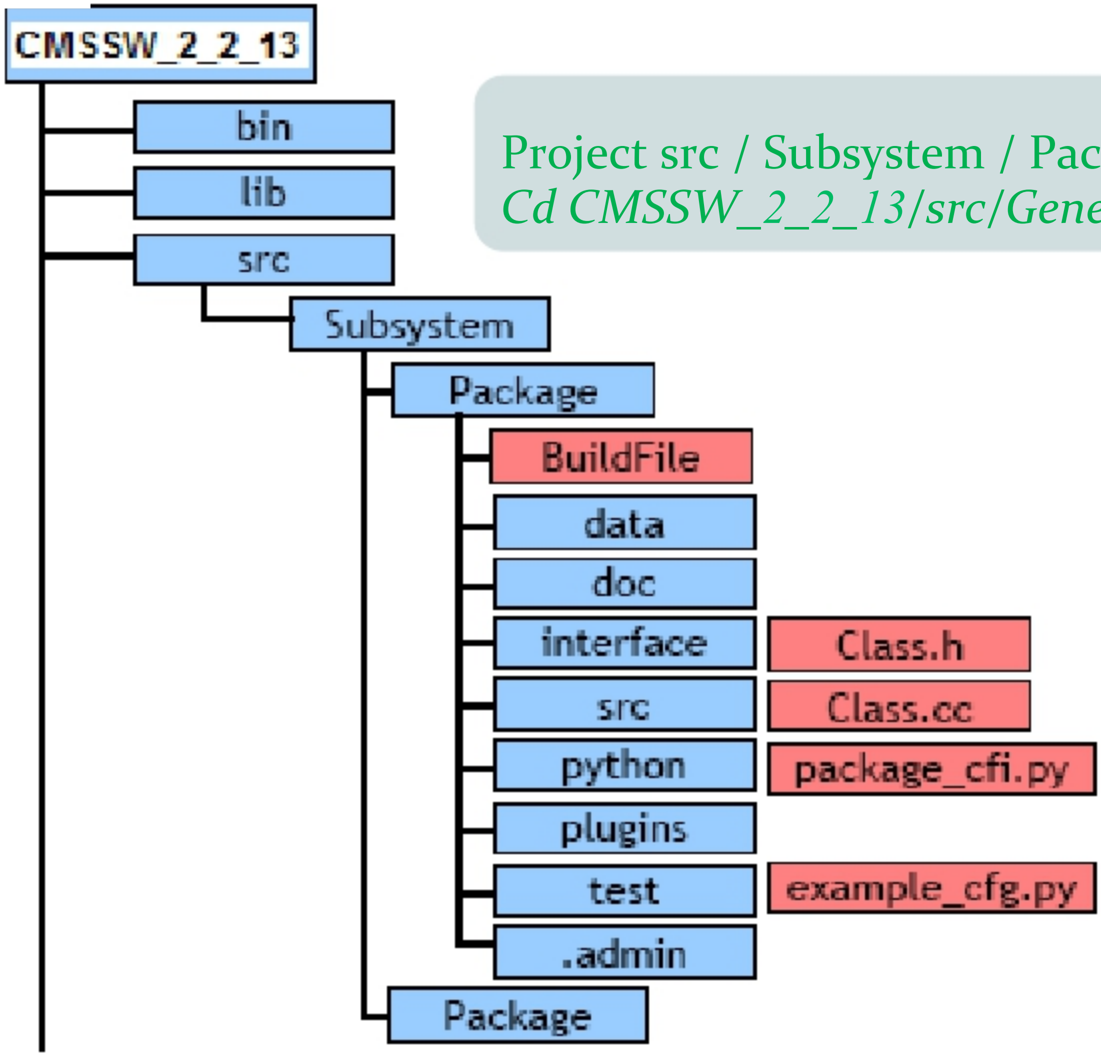
```
$ cmsenv
```

*adding a CMSSW package from the CVS server*

```
$ addpkg Subproject/package
```

# The Release Area

Project src / Subsystem / Package
*Cd CMSSW_2_2_13/src/GeneratorInterface/Pythis6Interface*



Only a subset

## Writing CMSSW modules

What kind of module to use?

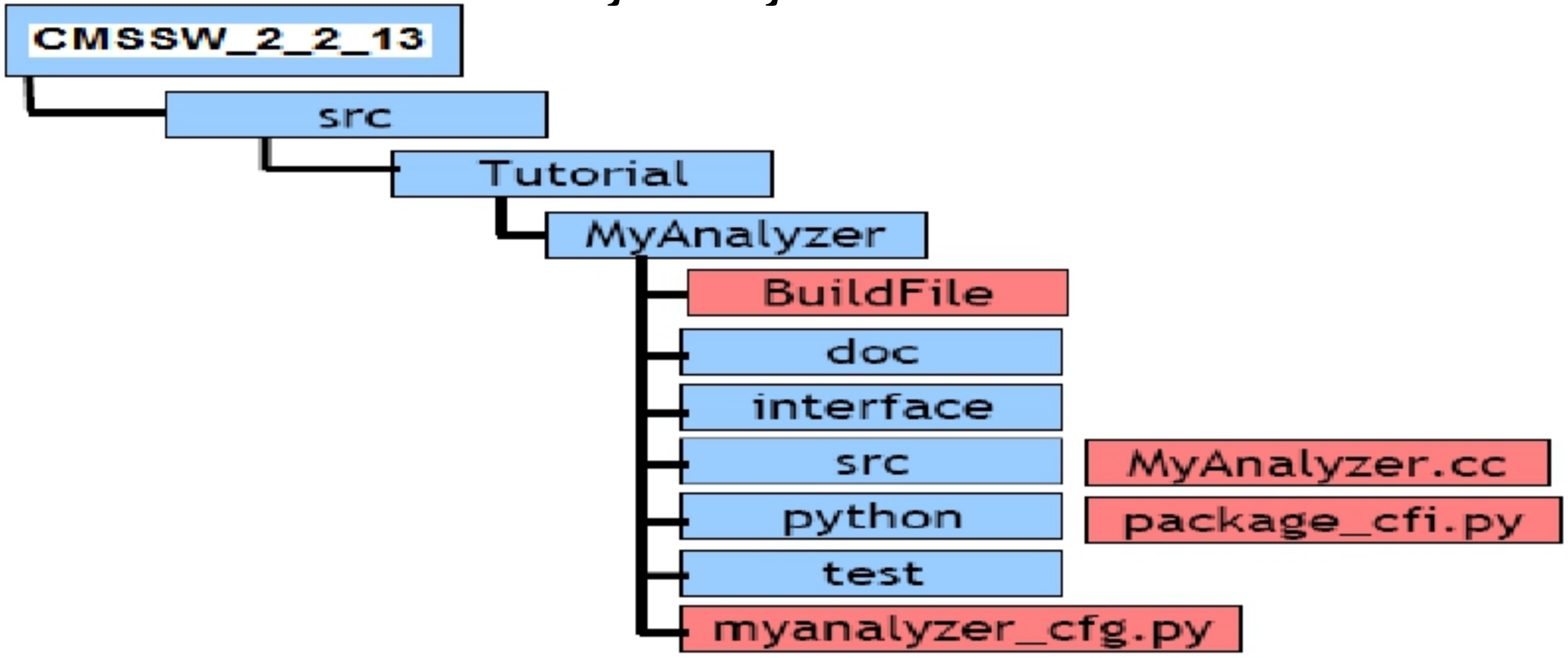
EDAnalyzer
- Reading data only
- Creating histograms
- the standard use case

EDProducer
- You want to create new products
- You want to share your reconstruction code with others
- You want to make different algorithms plugable

EDFilter
- You want to know if an object could be produced
- you want to control the analysis flow or skimming

## Creating an EDAnalyzr

```
$ cd CMSSW_2_2_13 /src
$ cmsenv
$ mkdir Tutorial
$ cd Tutorial
$ mkedanlzr –list
$ mkedanlzr –histo MyAnalyzr
```



# src/MyAnalyzer.cc

Class MyAnalyzer : public edm::EDAnalyzer

```
private:
void beginJob(const edm::EventSetup &);
void analyze(const edm::Event &, const edm::EventSetup &);
void endJob();
```

```
//---------------------- method called for each event ----------------
void
MyAnalyzer::analyze(const edm::EventSetup &iEvent, const edm::EventSetup &iSetup)
{
using namespace edm;
using reco::TrackCollectiion;
Handle<TrackCollection> tracks;
iEvent.getByLabel("moduleLabel", tracks);
for(TrackCollection::const_iterator itTrack = tracks->begin(); itTrack != tracks->end();
++itTracks)
    {
       int charge = itTrack->charge();
    }
}
```

```
DEFINE_FWK_MODULE(MyAnalyzer);   // define as CMSSW plugin
```

## Build the example

The build file

```
<use name=FWCore/Framework>
<use name=FWCore/PluginManager>
<use name=FWCore/ParameterSet>
<use name=DataFormats/TrackReco>
<flags EDM_PLUGIN=1>
<export>
    <lib name=TutorialMyAnalyzer>
    <use name=FWCore/Framework>
    <use name=FWCore/PluginManager>
    <use name=FWCore/ParameterSet>
    <use name=DataFormats/TrackReco>
</export>
```

Then to compile file use:

```
$ cd MyAnalyzer
$ scram b –j 3
```

## Running the example

Myanalyzer_cfg.py

```
Import FWCore .ParameterSet.Config as cms
Process = cms.Process(“”Demo)
Process.load(“FWCore.MessageService.MessageLogger.”_cfi)
Process.maxEvents = cms.untracked.PSet(input=cms.untracked.int32(-1))

Process.source = cms.Source(“PoolSource”,
#replace ‘myfile.root’ with source file you want to use
fileNames = cms.untracked.vstring(‘file:myfile.root’)
)
process.demo = cms.EDAnalyzer(‘MyAnalyzer’,
         tracks = cms.untracked.InputTag(‘generalTracks’)
)
process.p = cms.path(process.demo)
```

ctfWithMaterialTracks has been replaced by generalTracks
You need to put a real EDM file in, you can copy it from my home directory.

## Running the example II

```
$ cmsenv   (if not already executed)
$ cmsRun MyAnalyzer_cfg.py
```

***Next will be the interactive exercise session…***

## ***Backup Slides***

# More small but useful tools

```
# Inspect a configuration
$ edmConfigEditor <configfile>

# dump the provenance information
$ edmProvDump<rootfile>

# Create code skeleton
$ mkedanlzr <name> / mkedanlzr <-template> <name>
$ mkedprod <name>
$ mkedfltr <name>

# Translating symbols into human readablestring(error)
$ C++filt <symbol>

# To check the tags of installed packeges
$ showtags -r

# other useful tools
$ edm*
```