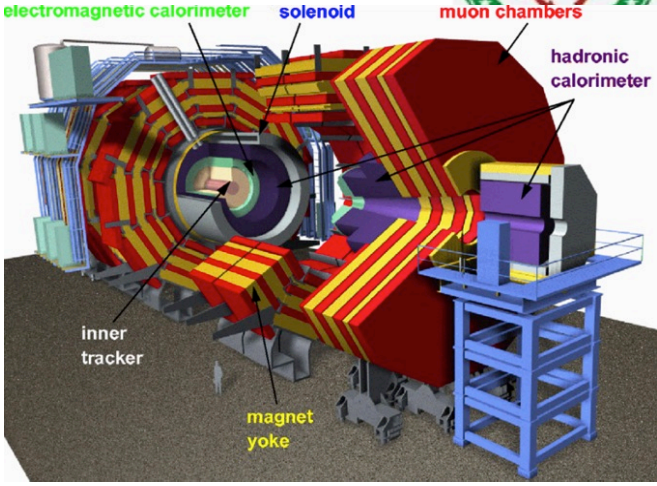
Introduction to CMS Software

Taimoor Khurshid

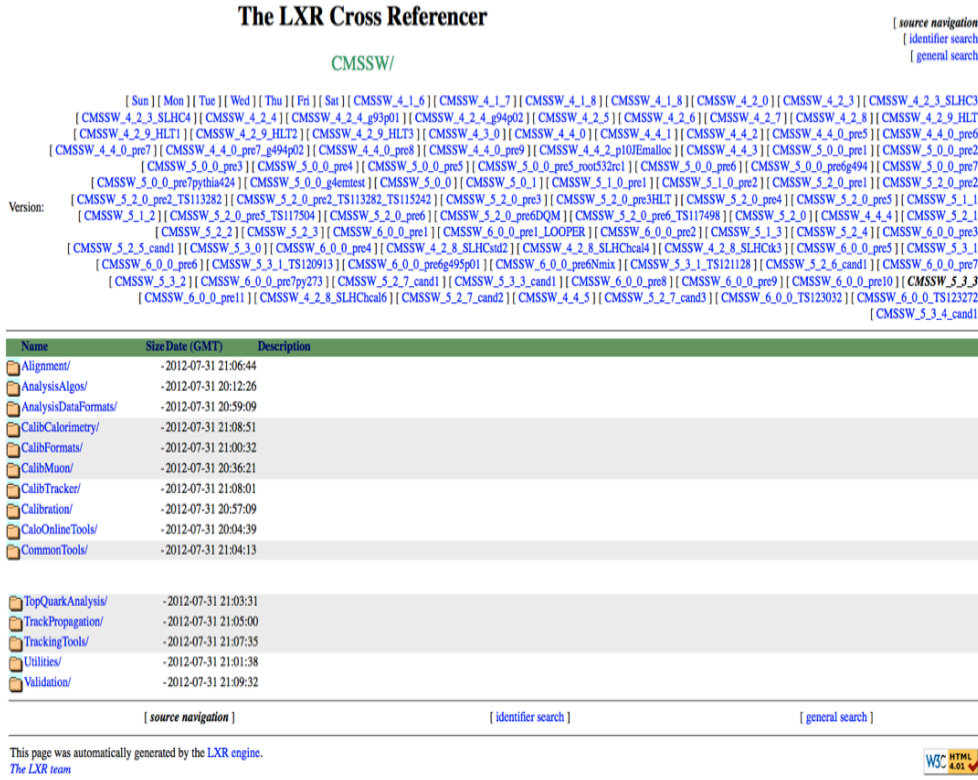
for

5th School on LHC Physics

CMSSW



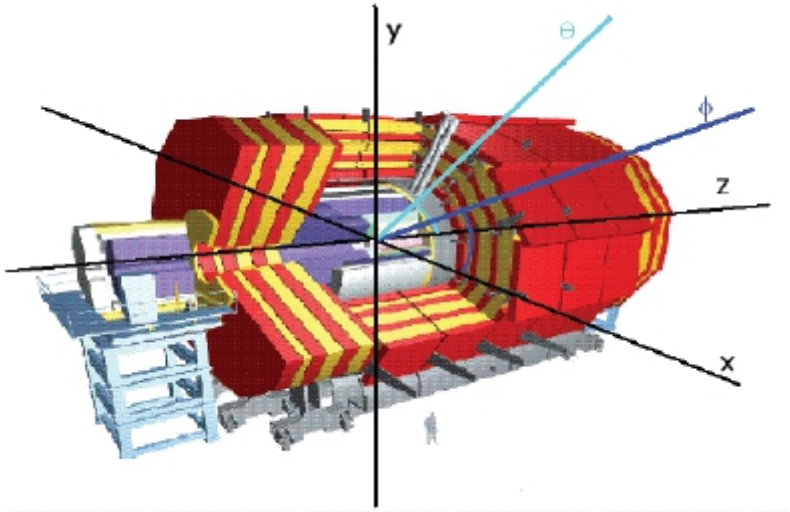
- The full suite of CMS offline software is called CMSSW
- Based on Event Data Model (EDM), used for
  - Online data taking,
  - simulation,
  - primary reconstruction,
  - physics analysis
- The two thousand (or so) packages that make up the project are organized into sub-systems with names that should be suggestive of their purpose.

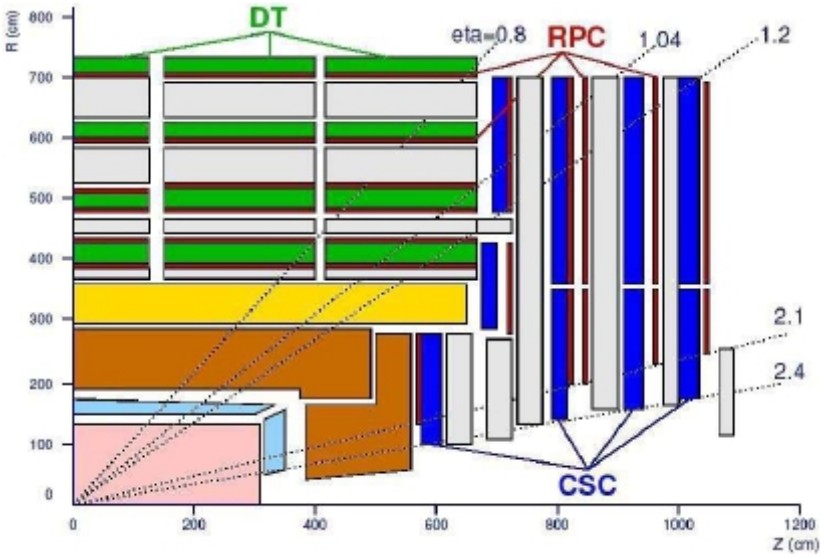


Documentation

- WorkBook - https://twiki.cern.ch/twiki/bin/view/CMS/WorkBook
  - source of one-stop shopping organization,
  - initial starting point for people new to CMS software and computing
  - intended for people to read through it, and work out the examples and tutorials it contains
  - CMS tutorials are found here
  - update frequently
- SWGuide - https://twiki.cern.ch/twiki/bin/view/CMSPublic/SWGuide
  - "industrial" strength shopping needs, expert level information
  - The software architecture, detailed descriptions of the algorithms, instructions for analysis and validation
  - Not updated as frequently
- Reference Manual - http://cmssdt.cern.ch/SDT/doxygen/

CMS Co-ordinates and conventions





- Azimuthal angle:

  ϕ = azimuthal angle

  $-\pi < \phi < \pi$

- Polar angle:

  θ= 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 you need to look for a physics problem

Event Generation

First step is to generate the events of your required physics problem, using event generators like Pythia, JETSET, herwig, cmkin, toprex etc

Simulation

Once you are done with generation, Detector effects are included.

Digitization

recHits, clusters etc

Reconstruction

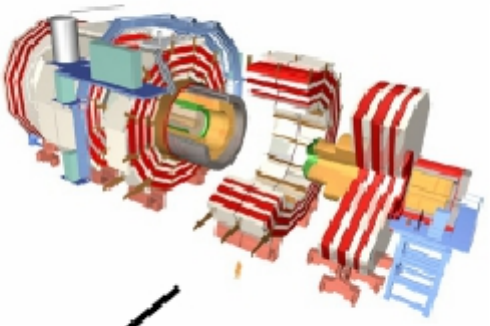
Tracks, Electrons, Muons, jets etc

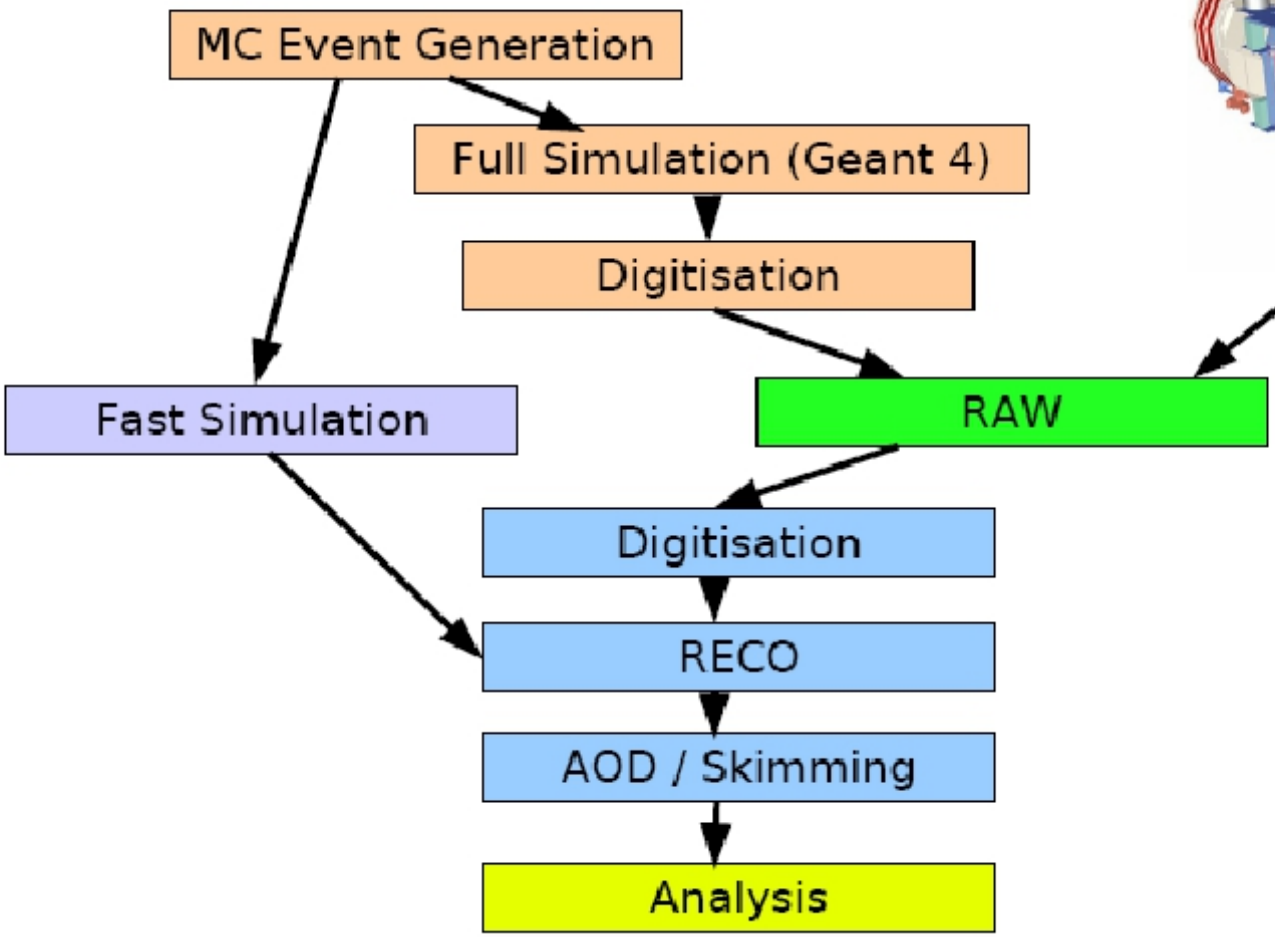
Final Analysis

Final Analysis code is written by the user.

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

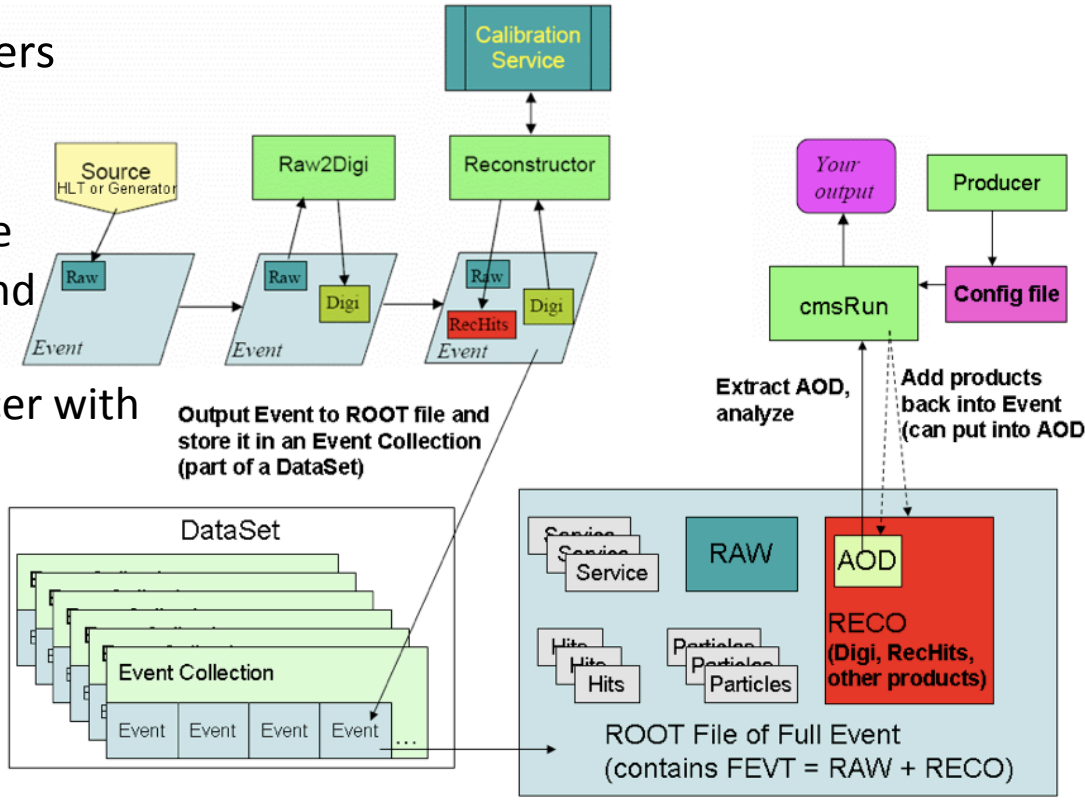
The CMS SoftWare uses *one single executable* for **everything**

→ **cmsRun** <*config file*>

- This includes
  - Online data-taking
  - Online high-level trigger *(HLT)*
  - Monte Carlo event generation
  - Detector simulation *(full, fast)*
  - Reconstruction
  - Analysis

CMS Event Data Model (EDM)

- A CMS Event* starts as a collection of the RAW data from a detector or MC event
- As the event data is processed, products are stored in the Event as reconstructed (RECO) data objects
- Event contains = triggered physics event + derived data + metadata(software config) + condition and calibration data
- Products are stored as C++ containers
- Because it's modular
  - you can inspect/debug
- Several instances of the same module can be run in the same application and you will still be able to uniquely identify their products, eg. JetProducer with different cone sizes. Identified by
  - C++ type,
  - producer label,
  - instance name,
  - process name
- The FW automatically tracks the provenance of what is produced.



* single readout of the detector electronics and the signals that will (in general) have been generated by particles, tracks, energy deposits, present in a number of bunch crossings

Concept of Provenance

Why should you care?
Provenance helps answer the question *"how was this root file made?", and "Why does my plot look different then Asif's," or simply "How is this file made?"*

All "tracked" parameters values used in the job that created the file are recorded in the file.

- Any result should be reproducible starting from
  - the input data file,
  - the plugins, and
  - the output provenance

Another source of provenance is the "Dataset Aggregation Service",DAS, which stores the top level configuration for each file.

What is stored in the event files?

In CMSSW a set of standard data formats is defined, they are collections of several products managed centrally in CMSSW

- RAW
  - Data like they come from the detector
- RECO (Reconstruction):
  - Output of the event reconstruction
- AOD (Analysis Object Data):
  - Subset of data needed for standard analysis
- RAWSIM, RECOSIM, AODSIM, MINIAOD:
  - with additional simulation information

What are the stored products?

edmDumpEventContent <filename>

C++ class type | product alias | label | process name

```
vector<reco::MET>          "tcMet"              ""          "RECO."
vector<reco::Muon>         "muons"              ""          "RECO."
vector<reco::Muon>         "muonsFromCosmics"   ""          "RECO."
vector<reco::Muon>         "muonsFromCosmics1Leg" ""        "RECO."
vector<reco::PFCandidate>  "particleFlow"       ""          "RECO."
vector<reco::PFCandidate>  "particleFlow"       "electrons" "RECO."
vector<reco::PFJet>        "ak5PFJets"          ""          "RECO."
```

```
Handle<reco::MuonCollection> muons;
Event.getByLabel("muons",muons );
```

Access the single Product in the framework module

reco::MuonCollection is a typedef for vector<reco::Muon>

Accessing Event Data

We can access the products
in the module using the Handle

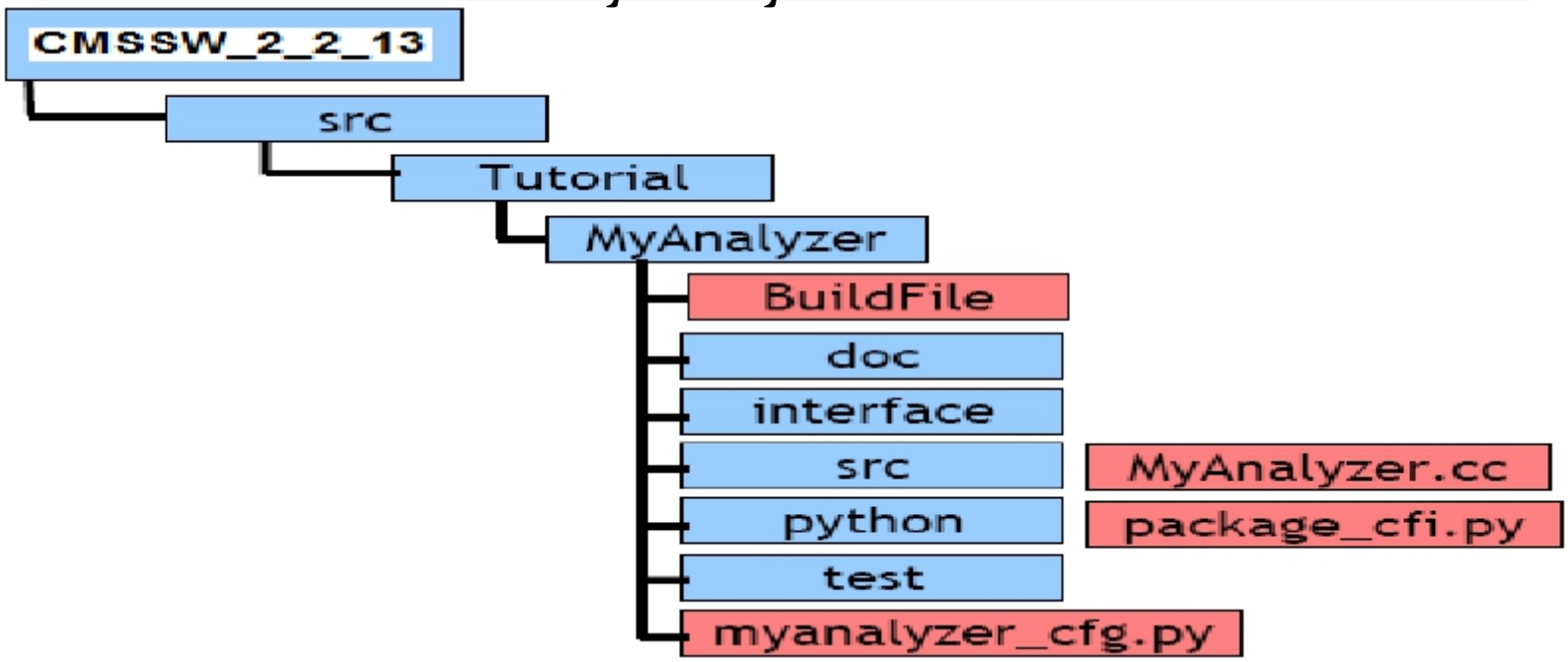
```
# by module and default product label
Handle<reco::MuonCollection> muons;
iEvent.getByLabel(”muons", muons );

# by module and product label
Handle<vector<reco::PFCandidate> > particleFlow;
iEvent.getByLabel("particleFlow", ”electrons" , particleFlow_electrons );
```

Framework modules are written in C++ , you can find a basic C++ guide at:

https://twiki.cern.ch/twiki/bin/view/CMS/WorkBookBasicCPlusPlus

Creating an EDAnalyzr

```
$ cd CMSSW_5_3_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)
```

Running the example II

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

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*

Getting Started With an Analysis

What do you want to do? What do you really(!) have to do to achieve this?

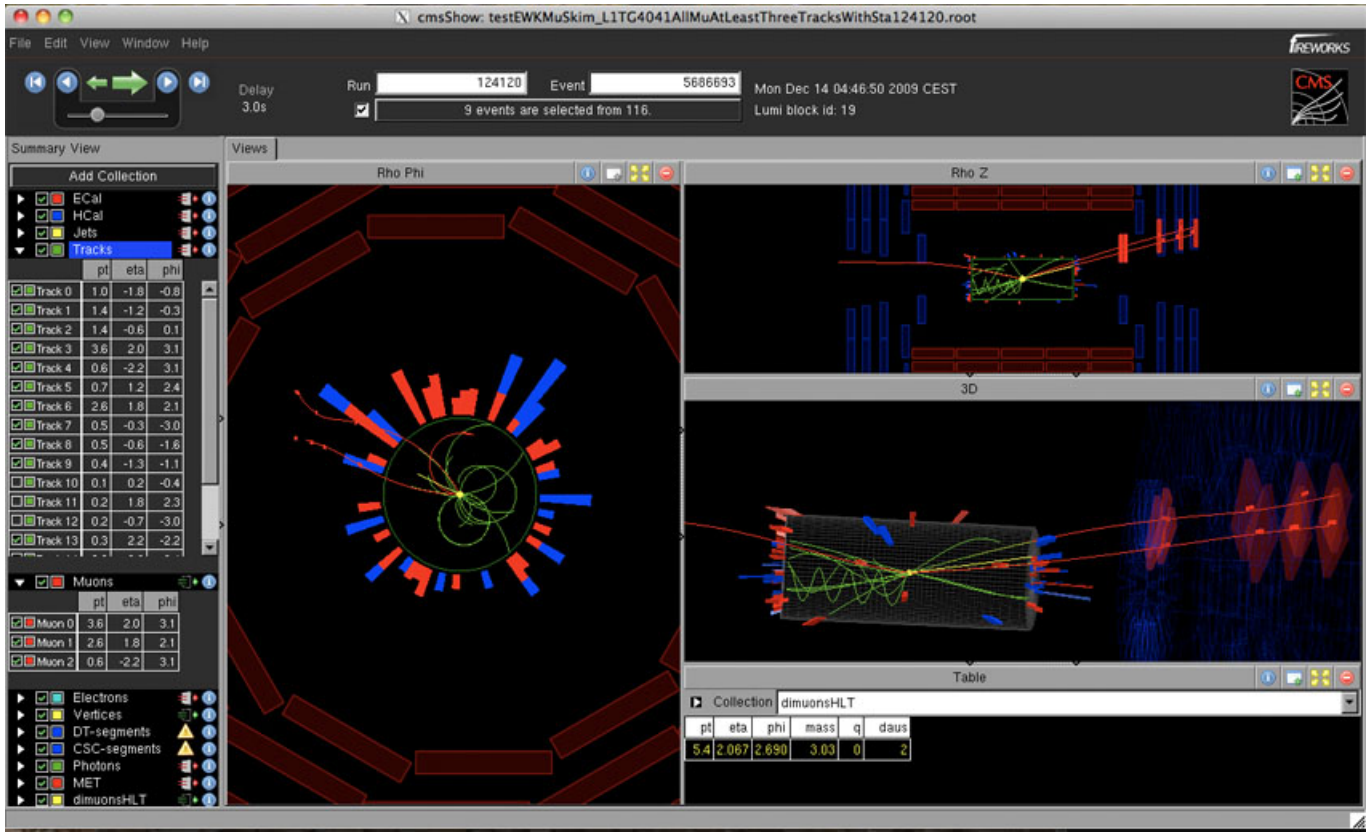
Physics

is	is not
• Selecting events. • Understanding data. • Understanding corrections. • Convincing others of the results • Writing notes and papers!	• Writing your own histogram plotting tool • Writing/maintaining your own ntupelizer • Convincing others about your variable definitions

Events can be visualized

In CMS there are visualization tools, one of them is:

Fireworks is the light weight event display for analysis. It can be installed on your laptop. You can find it at: https://twiki.cern.ch/twiki/bin/view/CMS/WorkBookFireworks



- Next
  - Exercises from Dr. Muhammad Ahmad