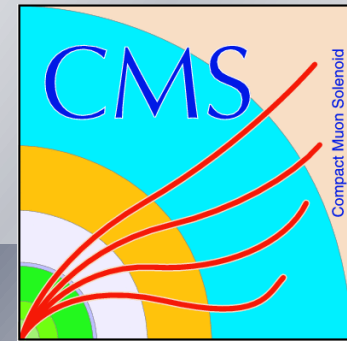


Introduction to CMS SoftWare (CMSSW):L2

Presented by

DR. MOHAMMED ATTIA MAHMOUD

- PhD, Fayoum University, Egypt and Antwerp University, Belgium.
- Researcher in ENHEP, ASRT, Fayoum Uni, and BUE.
- FSQ Gen-Contact, CMS experiment, CERN, Geneva, Switzerland.

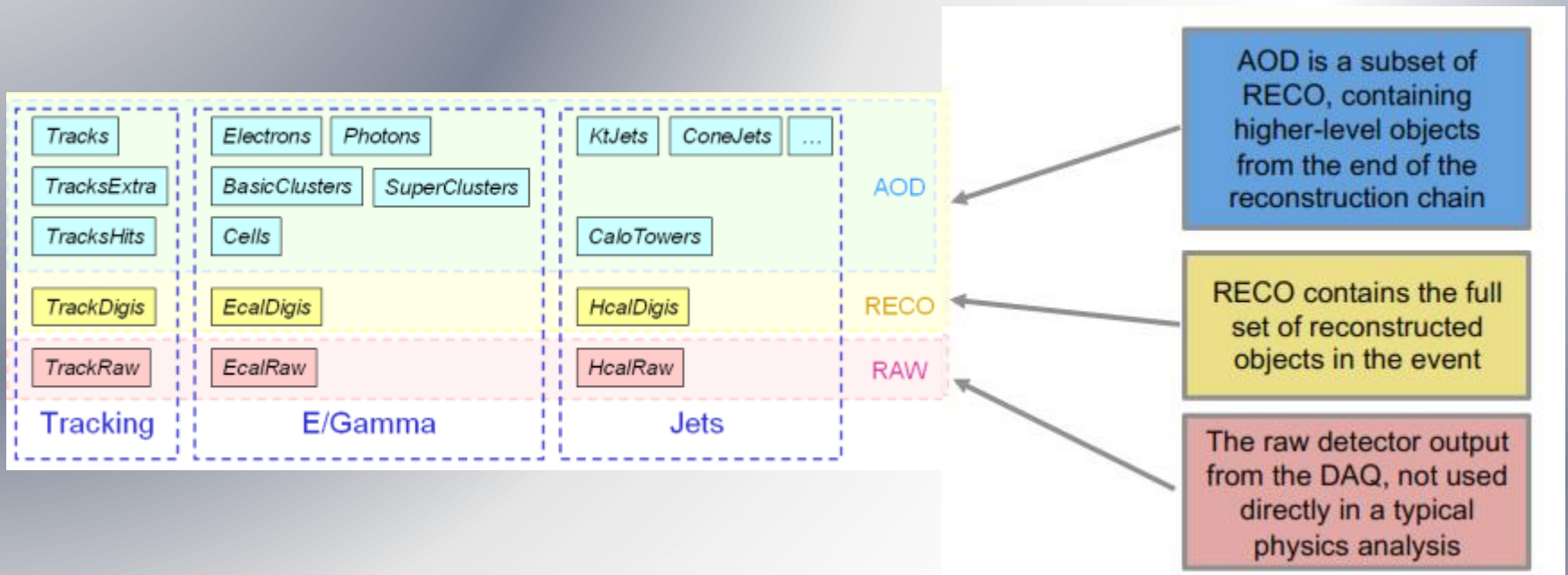


Outlines

- Data Tiers
- Event Data model
- CMSSW
- Github, LXR
- How to write your own analyzer
- Configuration file

Data Tiers

- Each step in the simulation or reconstruction chain adds an extra layer of information to the event



MC also has GEN SIM and DIGI

- **GEN**: generated event (e.g. output from Pythia, Madgraph)
- **SIM**: Generator level particle objects (inc. gen jets)
- **DIGI**: Digitised detector output

Dataset names list the tiers they contain, e.g.

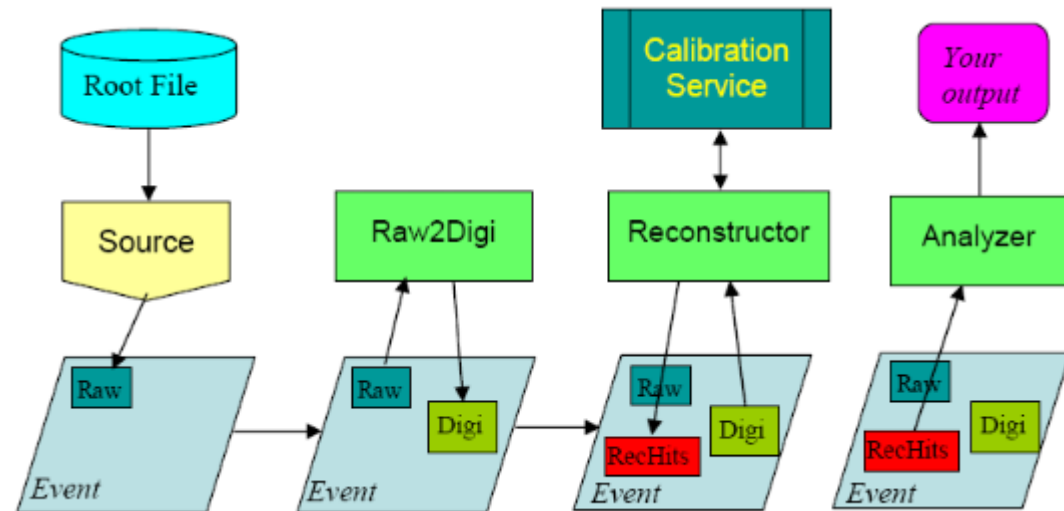
- /Neutrino_Pt2to20_gun/Summer12-UpgradeL1TDR-PU100_POSTLS161_V12-v1/GEN-SIM-DIGI-RAW

Event Data Model (EDM)

- All data is stored in ROOT files as it leaves the HLT
- Each event is stored as an edm::Event object in a TTree
- An edm::Event is a flexible container that contain type arbitrary sets of data
- Everything for RAW to RECO data is stored in the edm::Event
- Each file typically contains ~1000s of events
- Files are grouped into datasets

➤ ***For real data, a dataset is defined by a related set of triggers***

➤ ***In MC this means a particular process, e.g. Z+jets***

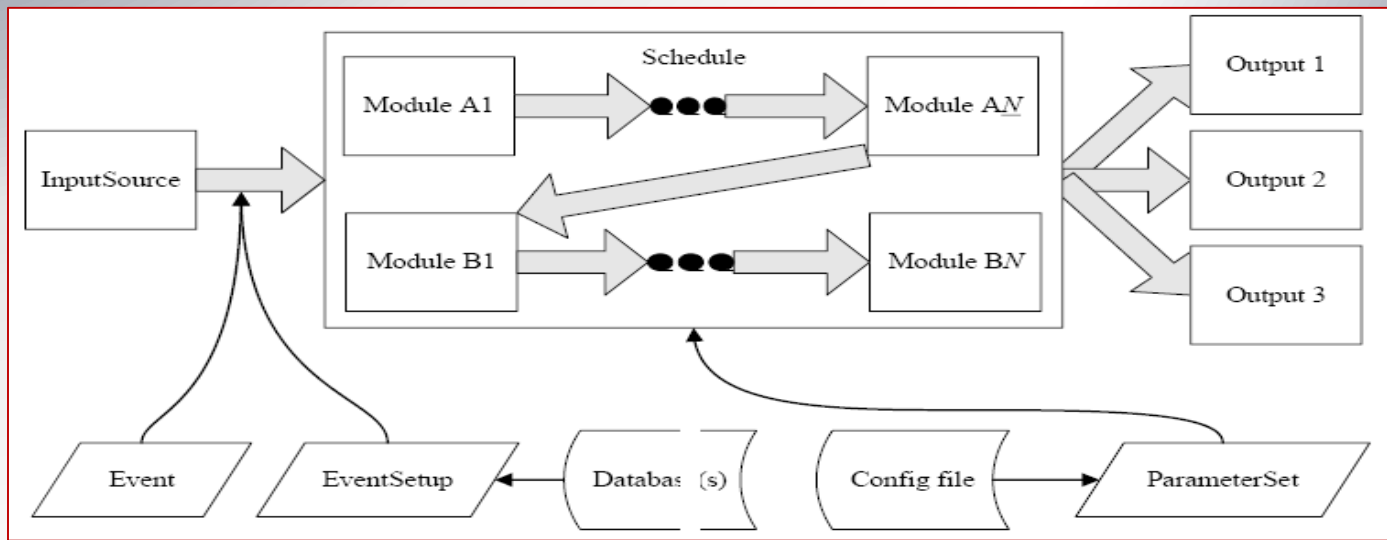


Event Data Model (EDM)

- Each event is contained in an edm::Event object stored within a ROOT TTree
- Each branch corresponds to an object, or a collection of similar objects, with a naming convention:
 - className_moduleLabel_productInstanceLabel_processName
 - e.g. recoMuons_muons__RECO
 - **moduleLabel**: is the label given to the instance of the module that created the collection
 - **productInstanceLabel**: Another label given to the collection by the producing module which is empty by default. It should be used in modules that produce multiple collections of the same type
 - **processLabel**: The name of the process in the job that created the data, e.g. HLT, RECO

CMSSW

- CMSSW is essentially a single program, cmsRun which loads modules and services on demand as specified in a configuration file
- You will mostly spend time writing modules which do all the analysis work, although you will also use services to get information from databases and to produce output
- Each event is passed through a series of modules called a path. A **path** is an ordered list of **Producer/Filter/Analyzer** modules which sets the exact execution order of all the modules.
- A module is able to both read data from the event and put new data into it.



CMSSW

- New versions of CMSSW are released regularly, with releases denoted by three numbers in the form CMSSW_X_Y_Z.
- Data and MC samples produced with a particular version must usually be analysed with the same version (though usually ensuring X_Y is the same is sufficient)
- The full package is quite large (~10GB), so when you set up a project area you only check-out the parts you want to change.
- The software is organised into packages, under the **CMSSW_X_Y_Z/src** directory of a release, with the convention: SubSystem/PackageName
 - – e.g. HiggsAnalysis/HiggsTauTau, DataFormats/MuonReco
- Each package has a common structure (see next slide)

Github

CMSSW moved from CVS to Git last year

– Not the easiest transition: Git is very powerful but conceptually very different from CVS

• CMSSW repository is here: <https://github.com/cms-sw/cmssw>

– Some extra CMSSW-specific tools are provided to work the repository:

<http://cms-sw.github.io/cmssw>

– Mostly needed to emulate the ability to checkout & compile a single package a la CVS

• Good idea to get a github account: <https://github.com>. Configure global setting on your machine:

```
git config --global user.name <First Name> <Last Name>
git config --global user.email <Your-Email-Address>
git config --global user.github <Your-Just-Created-GitHub-Account>
```

To checkout a package

```
$ cd CMSSW_5_3_4/src
$ cmsenv
$ git cms-addpkg DataFormats/MuonReco
```

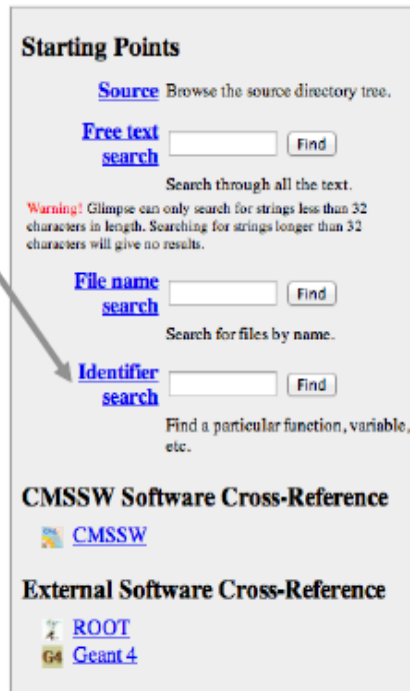

Github

- Some of the commonly-used commands are given below:
- `git clone [project URL]`: Make a local copy of a remote repository
- `git status`: Set what has changed since the last commit
- `git diff [file]`: shows the differences between your local version and the version in the git index
- `git add [file or folder]`: Add this to the staging area: the list of files that will be included in the next commit
- `git commit [-m "commit message"]`: Create a commit on the current branch from the changes that were added to the staging area
- `git pull`: Get new commits from a remote repository
- `git push`: Send local commits you created to the remote repository

CVS LXR

- A searchable version of the CMSSW repository: <http://cmslxr.fnal.gov/lxr/>
- Very useful for finding the source files where particular objects or methods are defined

For example, entering
GsfElectron finds
/DataFormats/
EgammaCandidates/
interface/GsfElectron.h



Starting Points


[Source](#) Browse the source directory tree.

Free text search
Search through all the text.
Warning! Glimpse can only search for strings less than 32 characters in length. Searching for strings longer than 32 characters will give no results.



File name search
Search for files by name.

Identifier search
Find a particular function, variable, etc.

CMSSW Software Cross-Reference

 [CMSSW](#)

External Software Cross-Reference

 [ROOT](#)
 [Geant4](#)



CMSSW Software Cross-Reference

This is a cross referenced display of the [CMS](#) source code. This includes all the sources for all the currently available releases of all CMS projects on the CMS AFS area at CERN, as well as the latest snapshot; these pages are updated many times a day, so they should be pretty close to the latest-and-greatest.

It's possible to search through the entire CMS source text; or to search for files whose name matches a pattern; or to search for the definitions of particular functions, variables, etc.

The individual files of the source code are formatted on the fly and presented with clickable identifiers. An *identifier* is a macro, typedef, class, struct, enum, union, function, function prototype or variable. Clicking on them shows you a summary of how and where they are used.

The [free-text search](#) command is implemented using [Glimpse](#), so all the capabilities of Glimpse are available.

If you are using an OpenSearch compatible browser ([Firefox](#) 2.0 or above, Internet Explorer 7, etc.) you can add these search engines to your search bar, either from the search bar itself or clicking on the icons next each link, below.

Warning: don't use a web-crawler to try and download all of these pages: the CGIs will feed you several *gigabytes* worth of generated HTML!

The pages here are generated by the [LXR](#) tool, which was originally written to display the source code of the Linux kernel (LXR stands for "Linux Cross Reference"). Check out the [main LXR site](#) for more information.

Send complaints, suggestions and praises to [the maintainer](#).

CMSSW Modules

A CMSSW task is built out a series of modules, which will process each event in turn.

There are three kinds of module (each implemented as a C++ class), each with a specific function. Your own module should be a class which inherits from one of these

- `edm::EDAnalyzer`

```
virtual void analyze(const edm::Event&, const edm::EventSetup&);
```

- An EDAnalyzer examines the event and produces output: histograms, ntuples etc. Note the **const** `edm::Event` in the analyzer method, it isn't able to modify the event content

- `edm::EDProducer`

```
virtual void produce(edm::Event&, const edm::EventSetup&);
```

- Generates some new objects and stores them in the event

- `edm::EDFilter`

```
virtual bool filter(edm::Event&, const edm::EventSetup&);
```

- The filter method is used to examine the event, returning false instructs the framework to stop processing the current event and move to the next one

- As well as the main analyze, produce and filter methods, all modules have a `beginJob` and `endJob` method, the former being called automatically before processing events and the latter after

How to write your own analyzer

- You have to have CERN account, for login to lxplus

```
ssh -Y OR X username@lxplus.cern.ch
```

- Listing the available CMSSW:

```
scram list
```

- Choosing CMSSW By using this command:

```
cmsrel CMSSW_X_Y_Z
```

- Apply cms environment :

```
cmsenv
```

- First, create a subsystem area. The actual name used for the directory is not important, we'll use First_analysis.

```
mkdir First_analysis  
cd First_analysis
```

How to write your own analyzer

- Create the "skeleton" of an EDAnalyzer module

```
mkedanlzl DemoAnalyzer
```

- Compile the code:

```
cd DemoAnalyzer  
Scram b
```

Configuration file

Always need to import this

```
import FWCore.ParameterSet.Config as cms
```

A config must always create a cms.Process called process. The label (MAIN in this case) will be applied to any objects added to the root files

```
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 the source file you want to use
```

```
    fileNames = cms.untracked.vstring(
```

```
        'file:/afs/cern.ch/cms/Tutorials/TWIKI_DATA/TTJets_8TeV_53X.root'
```

Create a data source, a list of files in this case.

Number of events to process. -1 means process all events in the input files. NB. When submitted jobs to the grid this parameter will be overridden automatically

```
process.demo = cms.EDAnalyzer('DemoAnalyzer'
```

```
)
```

The last step is usually to define one or more paths containing the sequence of modules to actually run

Define some modules. The first argument should be the name of the module as defined in the C++ class (usually the class name). The subsequent named arguments define the parameter set for that module.

```
process.p = cms.Path(process.demo)
```

Thanks!