Imperial College London

Introduction to CMS Computing

Gordon Ball
Imperial College London

gordon.ball@cern.ch
GPG 0x324543E5

Overview

Imperial College London

- This will try and provide a broad outline of how the various computing frameworks of CMS work, and how you work with them.
- These slides are meant to be useful reference as well as a course
- Rough outline:
  - CMS Computing Concepts
  - The CMS Software Framework
  - Other CMSSW Topics
  - CMSSW Exercise

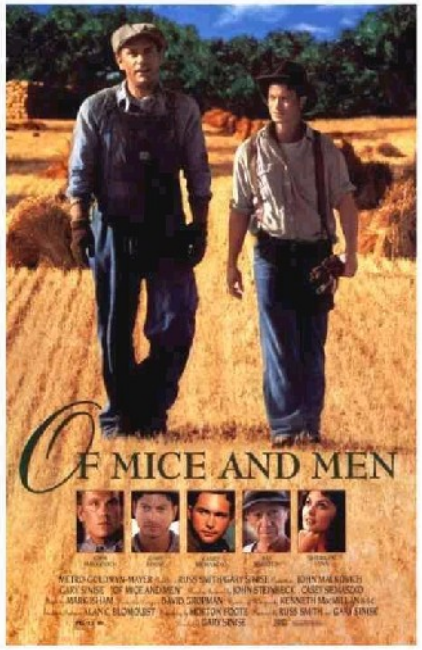
gordon.ball@cern.ch
GPG 0x324543E5

CMSSW Overview

Imperial College London

- This all comes down to learning your way around a software package called CMSSW – *The CMS Software Framework*
- This is a C++ framework designed for all physics use within CMS
  - Used for both Monte Carlo studies and real data

The best laid plans...



...often go awry

- Topics
  - How data from CMS is processed
  - How that data is stored
  - How you can access that data
  - How to get code to analyse that data
  - How to write your own analysis code
  - Advanced topics

gordon.ball@cern.ch
GPG 0x324543E5

Data Processing

Physics step

Detector step

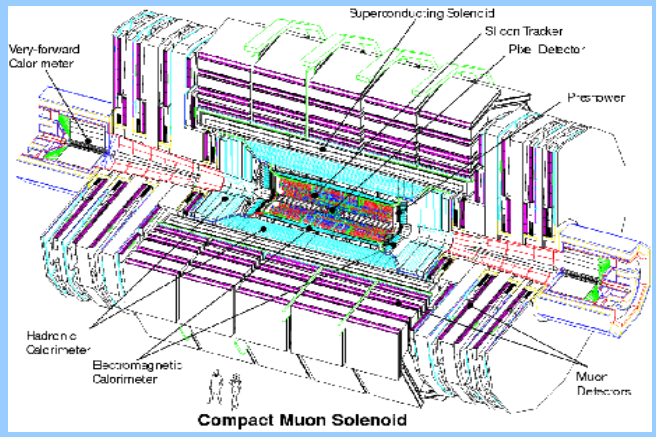
CMSSW at MC production site

Simulated physics event from HepMC event generator – eg PYTHIA, Madgraph, SHERPA etc

Geant 4

GEANT radiation simulation of event in detector, and simulated detector responses

Actual collision event in the LHC at point 5

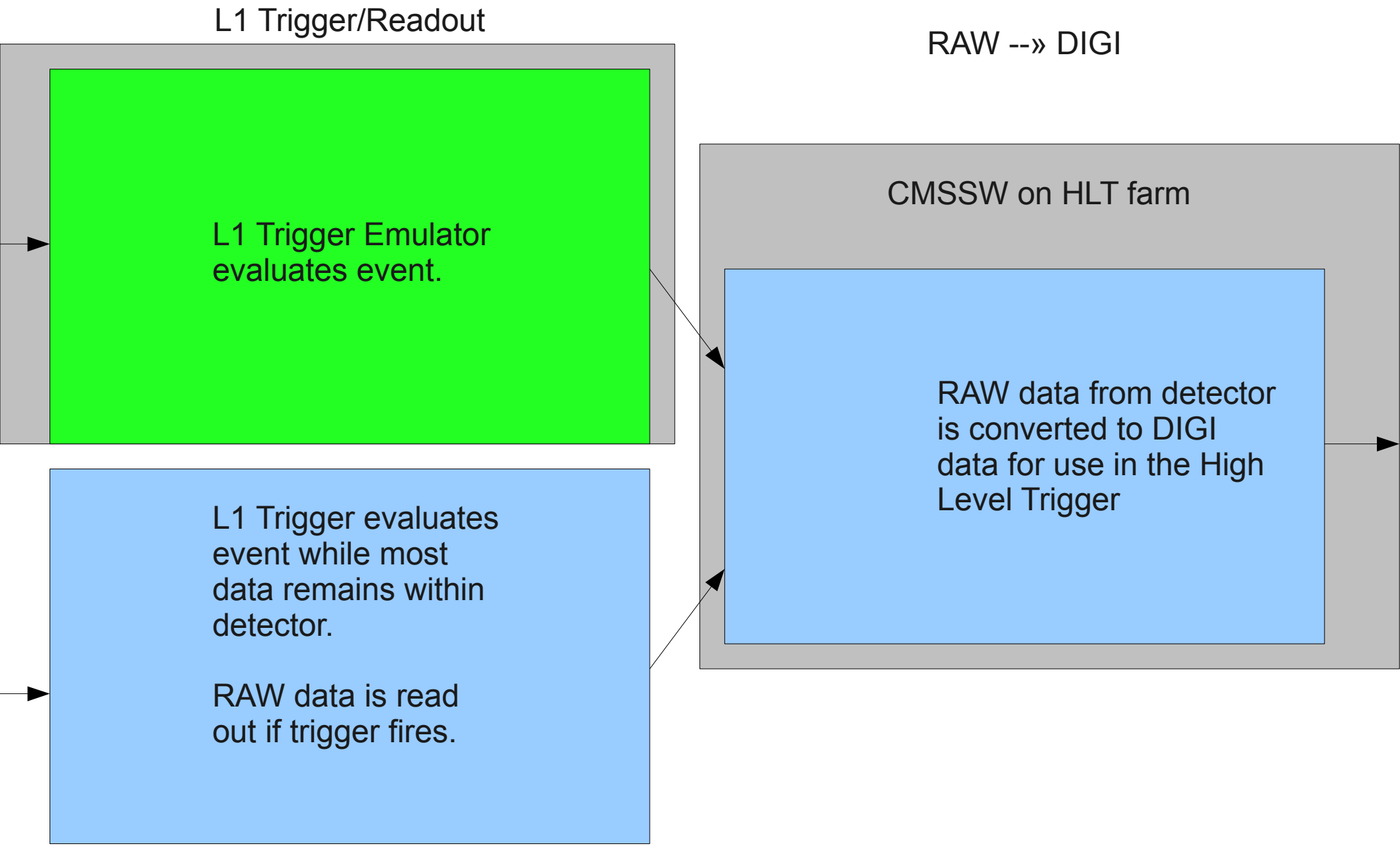


CMS detector systems

gordon.ball@cern.ch
GPG 0x324543E5

Data Processing

Imperial College London



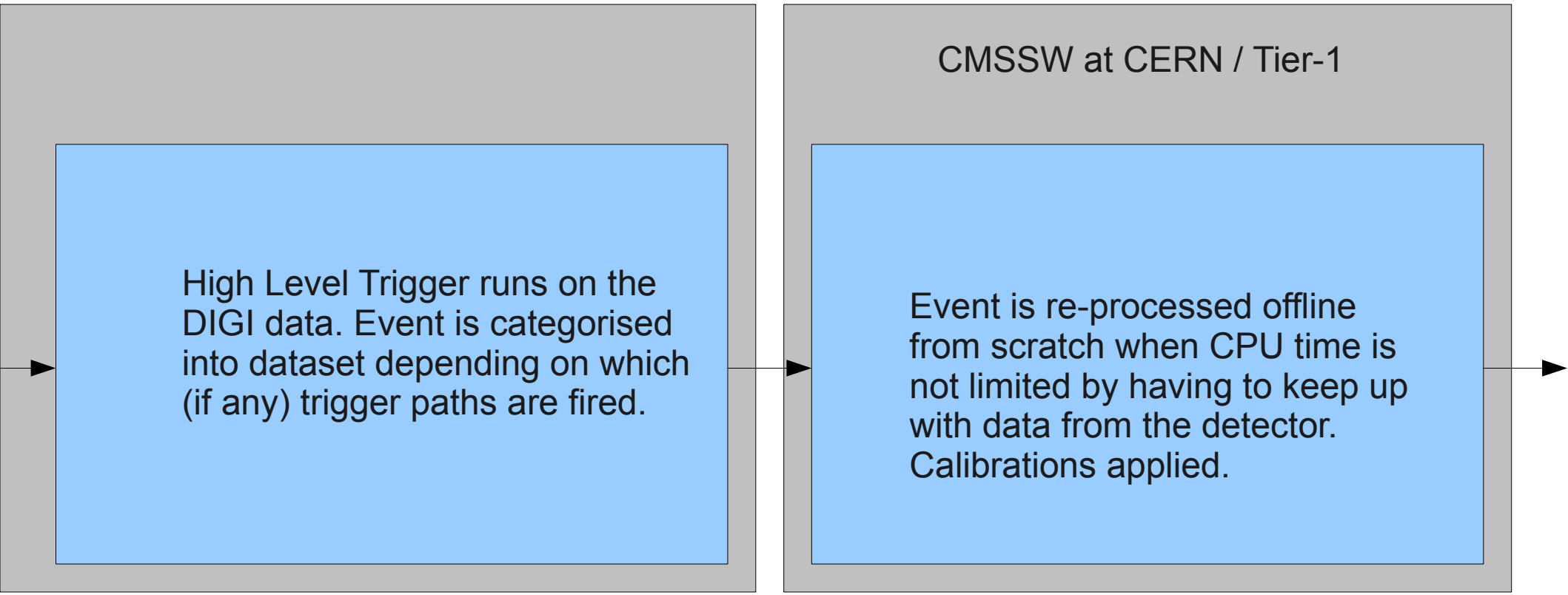
gordon.ball@cern.ch
GPG 0x324543E5

Data Processing

Imperial College London

High Level Trigger

Reconstruction



gordon.ball@cern.ch
GPG 0x324543E5

Data Processing

Imperial College London

CMSSW at own site / grid site

User retrieves reconstructed data from storage and analyses.

CMSSW at own site / grid site

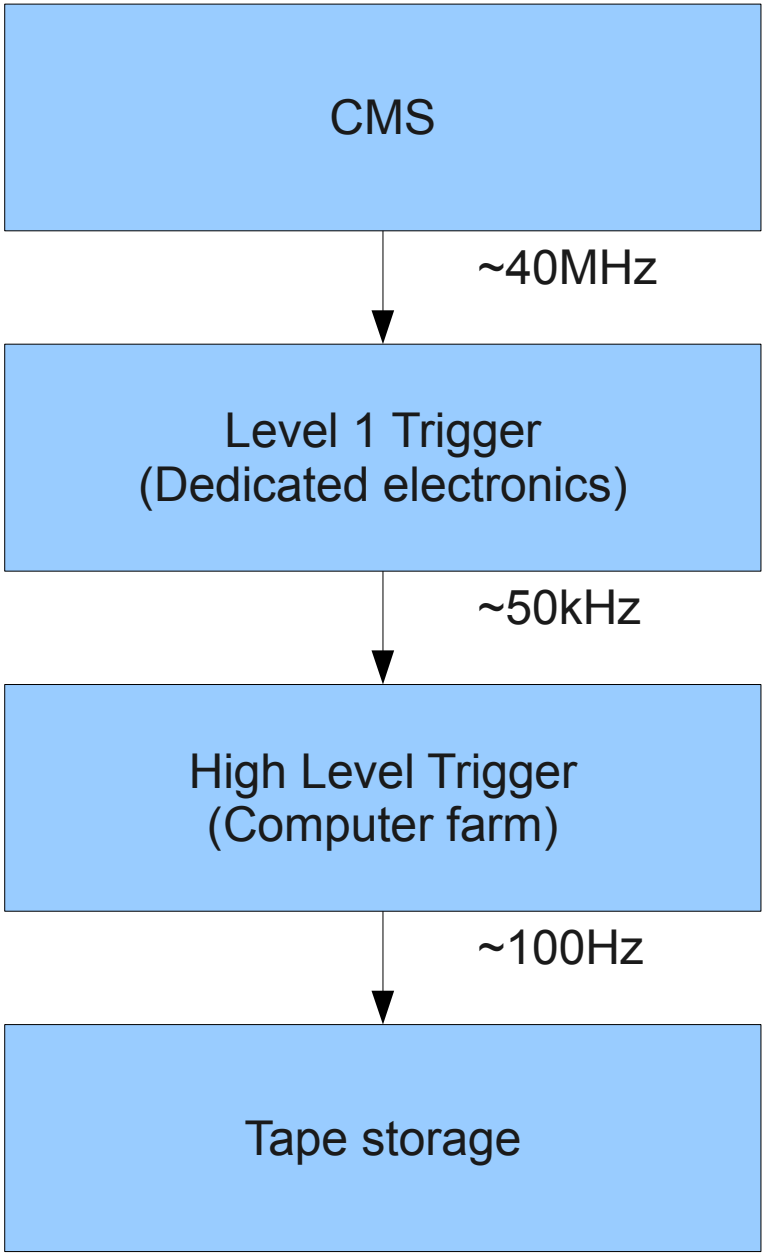
Higher level objects (eg, the Physics Analysis Toolkit (PAT)) are created.

ball@cern.ch
0x324543E5

Triggering

Imperial College London

- The decision to keep or drop an event in CMSSW is made by two systems
- Level 1 Trigger
  - Dedicated electronics, underground at Point 5
  - Sets “L1 Trigger Bits” indicating which event feature caused it to pass
  - Triggers can be “prescaled” - only accept 1 in N events
- High Level Trigger
  - CPU farm, surface at Point 5

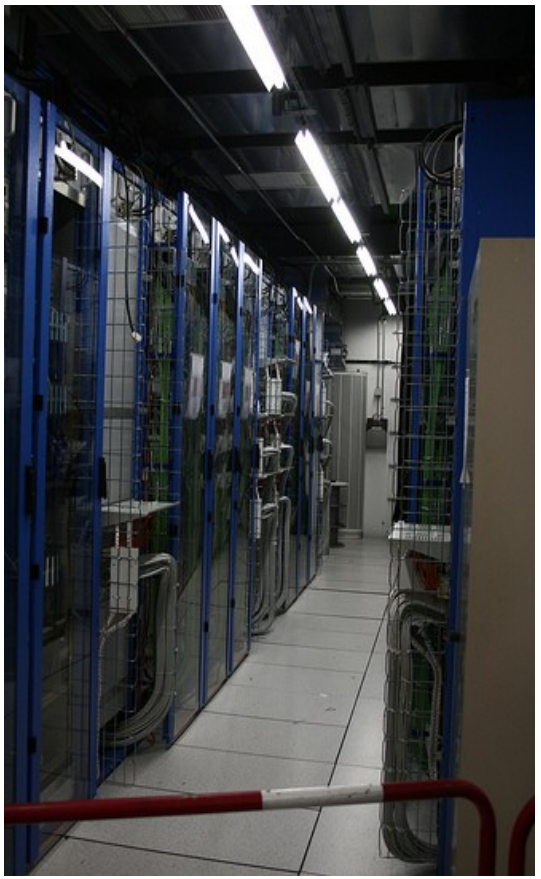


gordon.ball@cern.ch
GPG 0x324543E5

High Level Trigger

Imperial College London

- The High Level Trigger is an entirely software trigger level, running CMSSW
- It runs on a dedicated server farm of ~2000 CPUs at CERN



← This is actually the L1 Trigger. My bad.

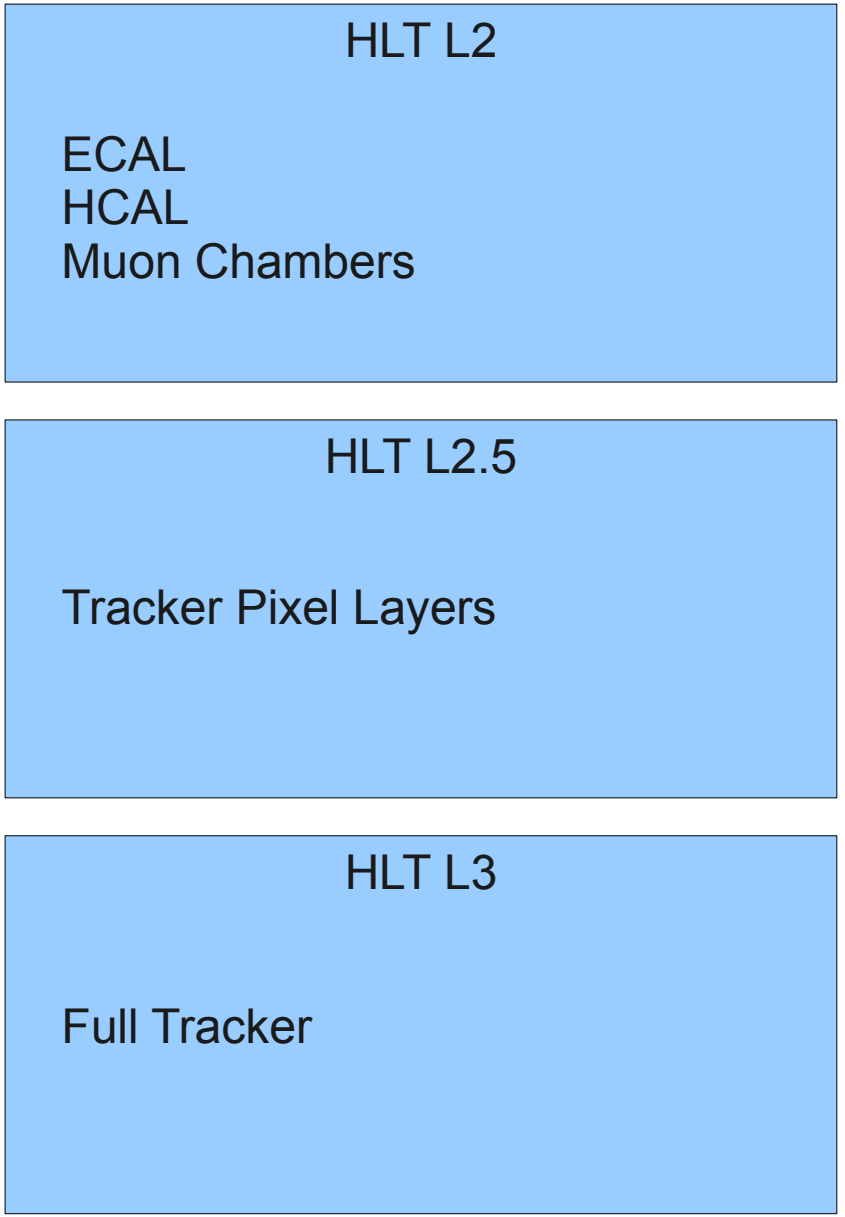
- Each physics group will have their own trigger paths of interest, and you will learn the complexities of those that are relevant to you
- A trigger path is a sequence of CMSSW modules which ultimately makes a boolean decision to keep or drop an event
- All trigger paths run on all events that pass L1, regardless of the L1 bit fired

gordon.ball@cern.ch
GPG 0x324543E5

HLT Levels

Imperial College London

- The CMS High Level Trigger is divided into layers, with increasing amounts of information available at each layer.
- Each layer requires more CPU time to unpack and work with, so lower levels filter as many events as possible to reduce the average time required
- The average time required for all events at the HLT must be less than 40ms to keep up with L1
- The HLT starts at L2 (L1 is the hardware trigger)

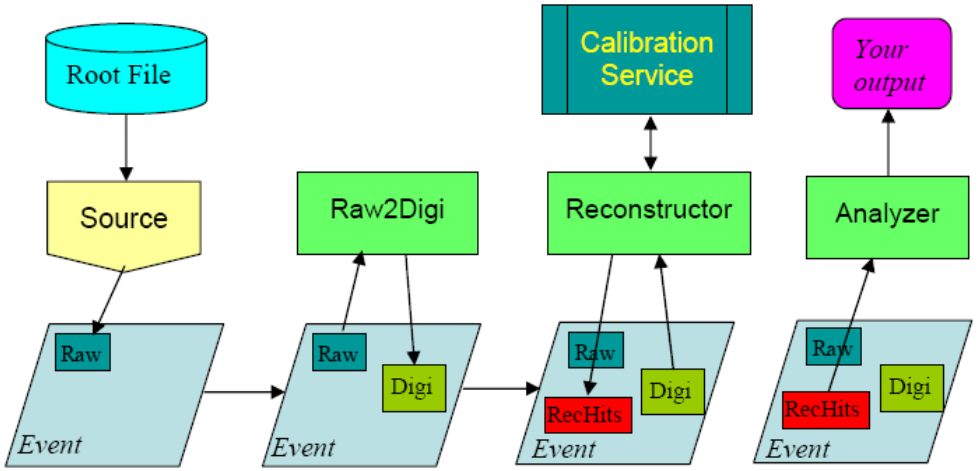


gordon.ball@cern.ch
GPG 0x324543E5

Event Data Model (EDM)

Imperial College London

- As soon as data leaves the HLT farm, it is stored in ROOT files
- These ROOT files contain an array of *edm::Event* objects, one per physical event
- *edm::Event* is a container that can contain any form of data, including arbitrary C++ objects
- All levels of data are stored in these containers, from RAW detector data to high level reconstructed objects
- Records are kept of provenance – the origin of each object in an *Event* to ensure results are reproduceable



- Events are split into files, each file typically containing thousands of events
- Files are grouped in datasets, containing a particular type of Monte Carlo data or real data passing a particular trigger path

gordon.ball@cern.ch
GPG 0x324543E5

Data Tiers

Imperial College London

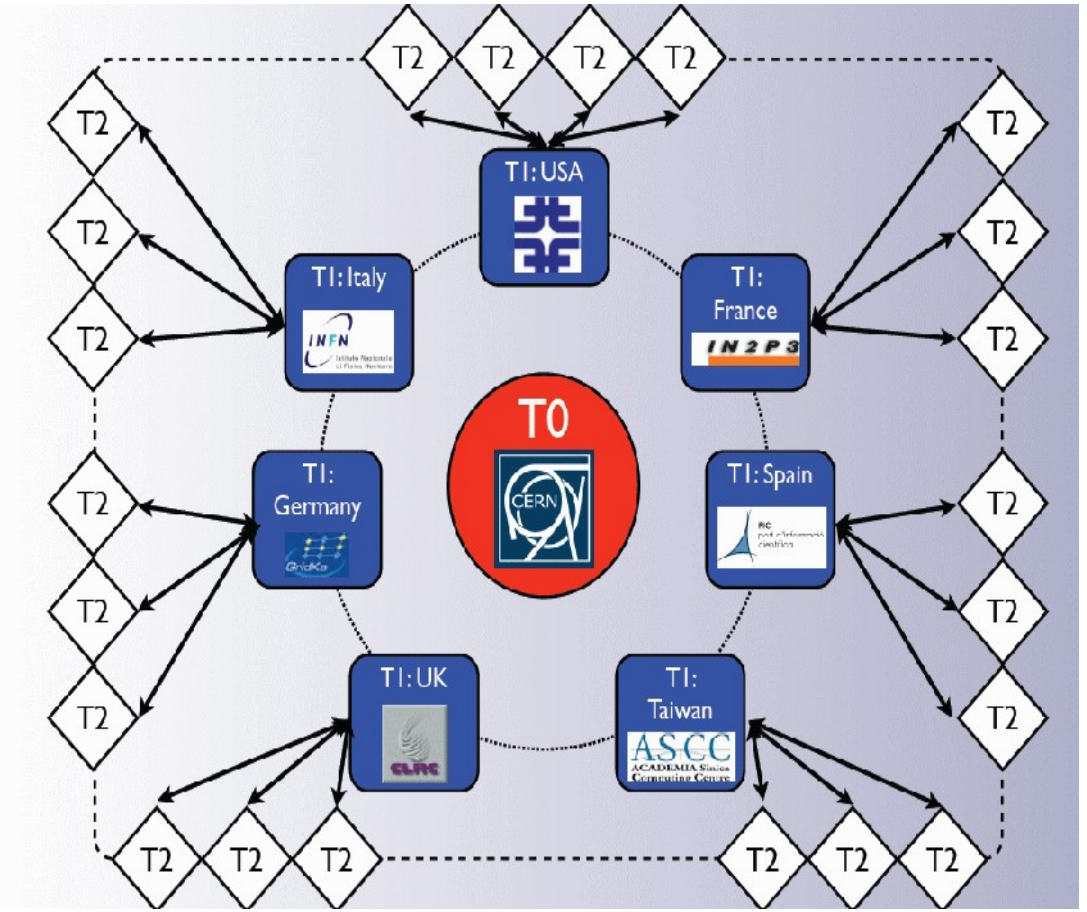
- Data is classified in Tiers depending on what types of data the events contain
- Dataset labels usually contain a list of tier labels indicating what data they contain
- GEN
  - Original generated event (MC only)
- SIM
  - Detector simulation (MC only)
- RAW
  - Raw output from the detector
- DIGI
  - Digitised detector output
- HLT
  - High Level Trigger output
- RECO
  - Offline Reconstruction output
- AOD
  - Analysis objects; a subset of RECO objects intended to summarise the physics processes while keeping the size down
- FEVT
  - Full Event - RAW+RECO

gordon.ball@cern.ch
GPG 0x324543E5

Data Storage

Imperial College London

- The volume of data CMS will produce is huge - 5PB a year or more
- All data should be retained on tape storage at CERN, but CERN lacks the CPU or network capacity to handle re-processing and analysis of all this data
- At least one copy of all data is stored at a Tier-1 site
- Users who want access to data cannot usually analyse it directly at a Tier-1 site but must request a copy is made to their local Tier-2 site for their access



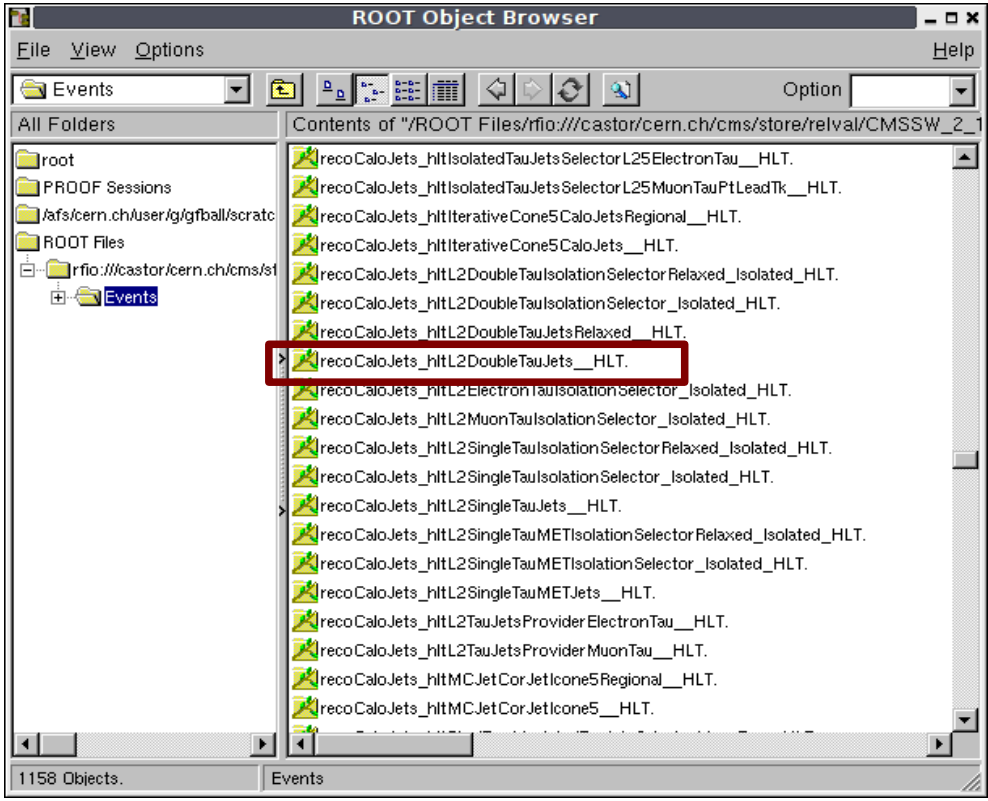
- Typical FEVT data is 1-2MB per event, requiring terabytes of storage for useful-sized datasets
- How to access data will be covered later

gordon.ball@cern.ch
GPG 0x324543E5

Inside an Event

Imperial College London

- Events are instances of edm::Event, stored in a ROOT tree
- Objects are named in the format *classname_module_label_process*
  - Classname is the name of the C++ class or type stored, according to ROOT's naming rules
    - eg '*std::vector<reco::CaloJet>*' becomes '*recoCaloJets*'
  - Module is the name of the CMSSW module that produced the data
    - eg hltL2DoubleTauJets
  - Label is an optional label added by the module (often blank)
  - Process is the name of the CMSSW process than added this data
    - eg HLT or RECO
  - This example becomes "recoCaloJets_hltL2DoubleTauJets__HLT"



An event open in ROOT

gordon.ball@cern.ch
GPG 0x324543E5

Skimming

Imperial College London

- Full CMSSW data generally runs to 1-2MB per event
- Skimming is the process of
  - Discarding information in the Event not needed for a given analysis
  - Discarding events not meeting some criteria
- This allows faster running on the data of interest
- There is no reliable naming convention for skims
  - Each physics group produces their own according to their own criteria and naming convention
- You will probably spend much of your time working on group-produced skims
- Be sure you understand the skim criteria when doing analysis with skimmed data
  - And how it could be biassing your results...

gordon.ball@cern.ch
GPG 0x324543E5

NTuples

- An NTuple is a general term for personal output formats containing a very limited subset of the information in the original data
- These are used for tasks that require a lot of rapid re-running
  - eg, optimisation of cuts
- Usually these are files containing a ROOT::TTree or ROOT::TNtuple, with branches for each value of interest
- Analysis is then done in ROOT, PyROOT or anything else that can read the output format
- This is useful for doing exotic things, but be careful of making something that is impossible for anyone else to reproduce

gordon.ball@cern.ch
GPG 0x324543E5

FWLite

Imperial College London

- It is possible to analyse CMSSW data without using CMSSW
- Since all the data is stored in ROOT-compatible formats, we can load the necessary libraries in ROOT and look at data directly
- This isn't a good way of doing anything complex, but it lets you look at things quickly and make simple plots without having to use full-scale CMSSW
- Also works with PyROOT

```
# set up CMSSW environment
root -l

#Welcome to ROOT
#Abandon all hope, all ye who enter here.

root[0] gSystem->Load("libFWCoreFWLite.so");
root[1] AutoLibraryLoader::enable();
root[2] gSystem->Load("libDataFormatsFWLite.s
```

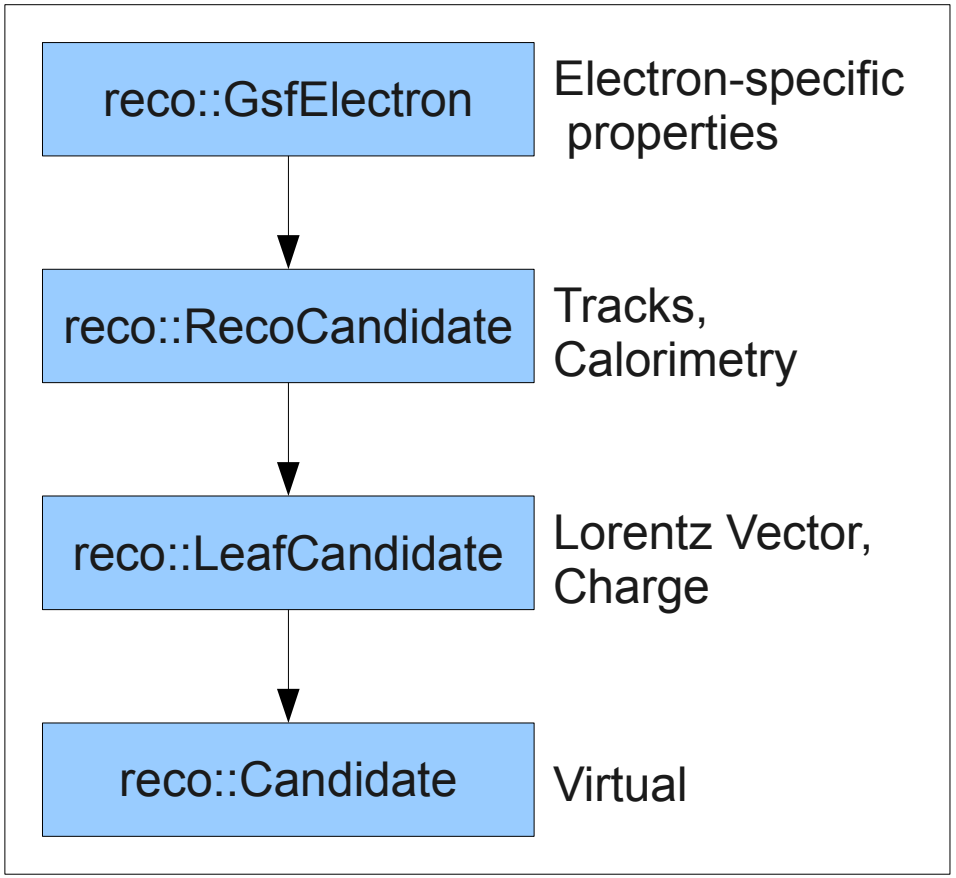
gordon.ball@cern.ch
GPG 0x324543E5

Data Formats

Imperial College London

The classes representing each type of particle use inheritance to build up their properties.

Eg, for a (Gaussian-Sum-Fitter) Electron class **reco::GsfElectron**



For each ParticleClass (eg GsfElectron), there are usually

A collection type

```
typedef reco::GsfElectronCollection =
 std::vector<reco::GsfElectron>
```

A reference type

```
reco::GsfElectronRef =
 edm::Ref<reco::GsfElectronCollection>
```

These are usually defined in the “Fwd” header file – eg `GsfElectronFwd.h` in this case

`edm::Ref` acts like a pointer to objects stored in ROOT files.

Classes used for particle storage are in CVS directory DataFormats/

gordon.ball@cern.ch
GPG 0x324543E5

Dataset Names

Process: ttbar

Data Tiers: GEN, SIM, RECO

Generator: pythia6 at 7TeV using Tune Z2 and taus decayed with tauola

/TT_TuneZ2_7TeV-pythia6-tauola/FALL10-START38_v12-v1/GEN-SIM-RECO

Production (FALL10)
Large blocks of data are produced in groups (FALL10, WINTER10...) against a chosen version of CMSSW (3.8.7 in WINTER10). Smaller blocks of data called 'RelVal' (release validation) are created with each software release to test it behaves consistently.

Conditions: START38_v12

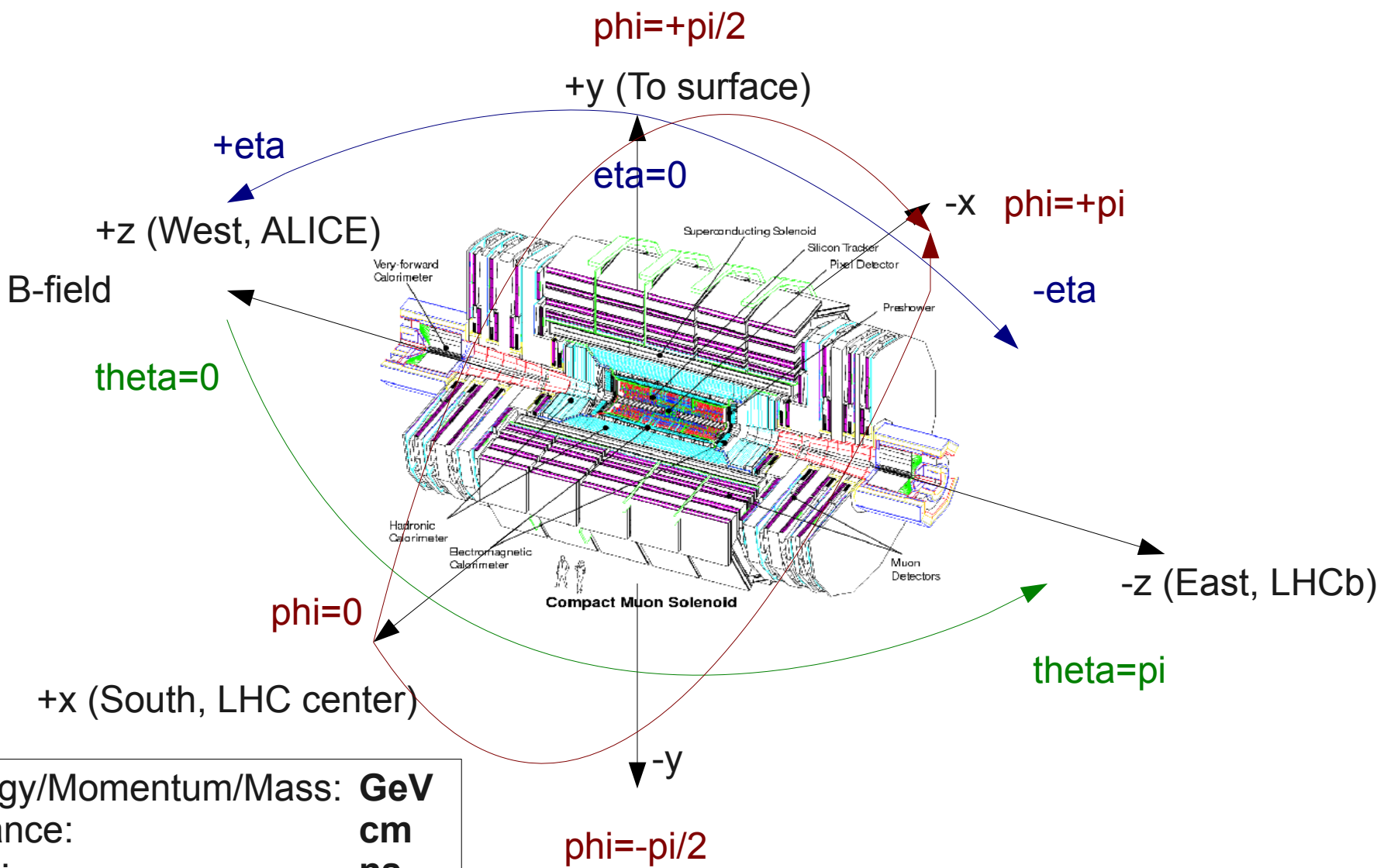
This indicates the detector simulation is set up for startup conditions (beam condition, detector condition, pileup, etc), and CMSSW 3.8.x. IDEAL conditions data also exists.

More on how to locate datasets later.

gordon.ball@cern.ch
GPG 0x324543E5

Coordinate System

Imperial College London



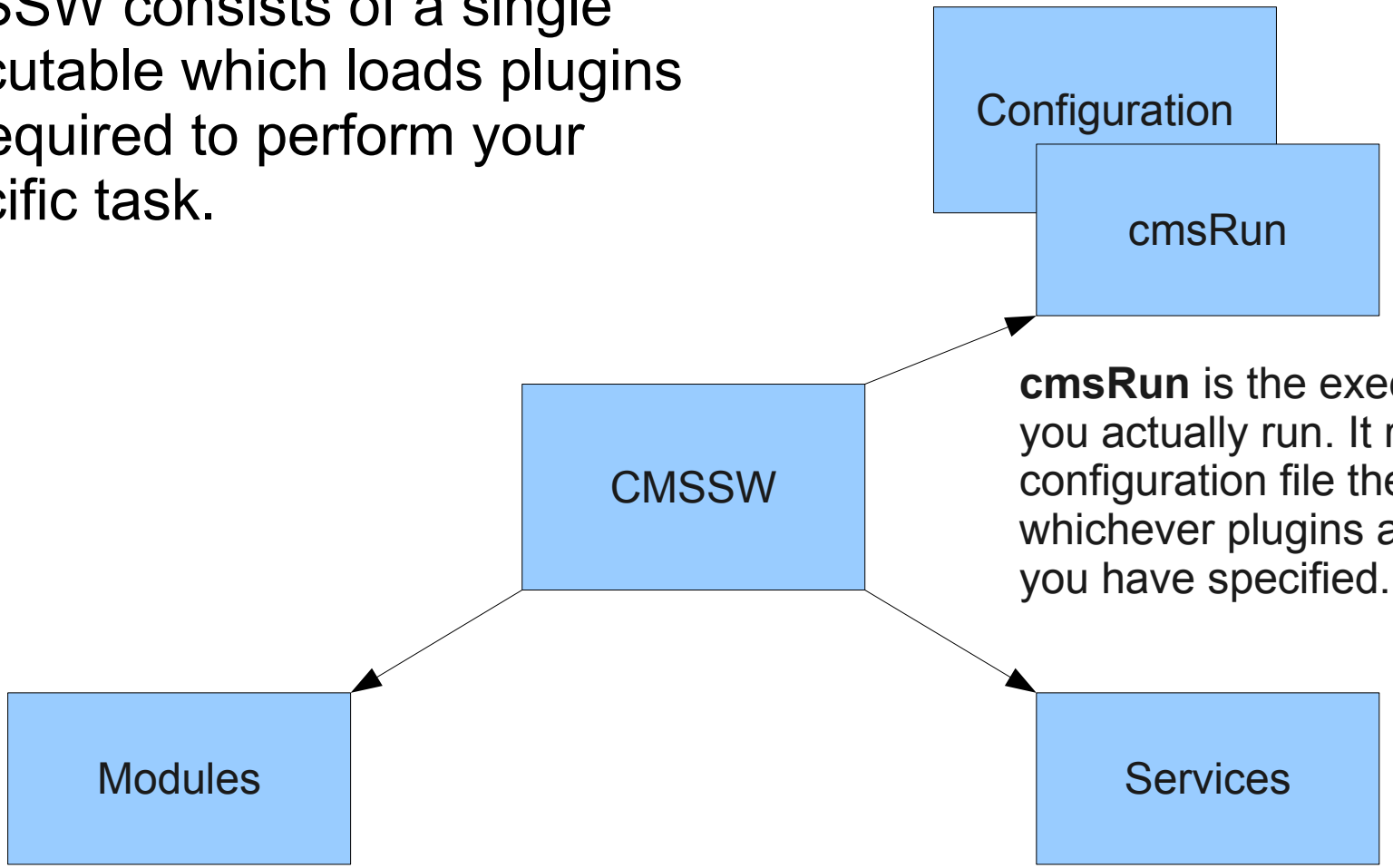
Energy/Momentum/Mass:	**GeV**
Distance:	**cm**
Time:	**ns**
Angles:	**rad**
B-Field:	**T**

gordon.ball@cern.ch
GPG 0x324543E5

CMSSW Framework

Imperial College London

- CMSSW consists of a single executable which loads plugins as required to perform your specific task.



cmsRun is the executable you actually run. It reads your configuration file then loads whichever plugins and data you have specified.

Modules are the sharp end of CMSSW that you will have to use and create to perform the analysis tasks.

Services are background processes like message loggers and file handlers. You probably won't need to write these, but most jobs will require that you run one or more.

gordon.ball@cern.ch
GPG 0x324543E5

Setting up CMSSW

Imperial College London

- CMSSW runs on Scientific Linux ("SLC")
  - You'll probably run it either on lxXX here or lxplus at CERN
- CMSSW has new releases often
  - The most recent Monte-Carlo production is for 3_8_x, data is being taken with 3_10_x or 3_11_x
  - Which version you use will largely depend on what version was used to generate your MC data and what version other members of your group are using
  - You can have areas for many different versions at once
- CMSSW is a very large piece of software
  - However, when you create a project area you do not get a copy of all code.
  - You only check out those bits you want to change, and everything else is loaded from a server containing a complete copy, with your modifications overriding the original copy
- All the code for CMSSW is kept in a CVS repository (Concurrent Version System)
  - http://cmssw.cvs.cern.ch/
  - More on CVS later

gordon.ball@cern.ch
GPG 0x324543E5

Setting up CMSSW

Imperial College London

- At Imperial only, you need to

```
source /vols/cms/grid/setup.sh
```

- Then, go to the directory you want to create a CMSSW area in and do

```
scramv1 project CMSSW CMSSW_X_Y_Z
```

- This will create a new project area that looks like

```
ls CMSSW_X_Y_Z
bin/   config/   doc/   external/
include/  lib/   logs/  module/
python/   share/ src/   test/  tmp/
```

*(Most of these directories are never used)

- You then need to set up your environment for this area

```
cd CMSSW_X_Y_Z/src
cmsenv
```

This sets up a list of needed environment variables for CMSSW and needs to be done each time you log in or open a new shell.

On lxplus, you should request (if you don't have already) access to *scratch space*. This will give you a directory *scratch0* in your home dir to supplement your limited AFS space. Note this space isn't backed up.

- You should now have a working CMSSW area ready to check out code into and run analysis with.

gordon.ball@cern.ch
GPG 0x324543E5

See also the CMSSW workbook https://twiki.cern.ch/twiki/bin/view/CMS/WorkBook

Setting up CMSSW

Imperial College London

- You should now have a blank CMSSW area, which isn't much use to you
- Before you can do anything, you need to check out appropriate packages or create your own
  - Checkout will be covered later, under CVS
- The format of a CMSSW package name is “SubSystem/PackageName”
  - eg “HiggsAnalysis/HiggsTo2photons”
  - Must use this structure or code won't compile
  - TwoWords/TwoWords is the naming convention

```
#to create a new package
cd CMSSW_X_Y_Z/src
cmsenv

#create a subsystem 'Demo'
mkdir Demo
cd Demo

#use the mkedanlzr skeleton creator
mkedanlzr DemoAnalyzer

cd DemoAnalyzer
```

#can also use mkedfltr, mkedprod to make other types of skeleton code. These will be discussed shortly.

gordon.ball@cern.ch
GPG 0x324543E5

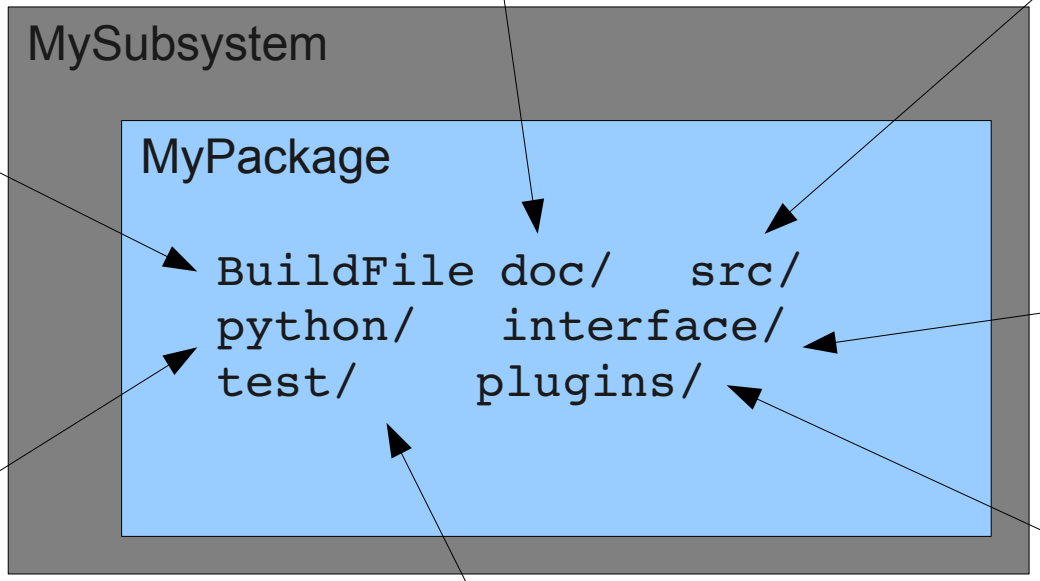
Structure of a Package

Imperial College London

BuildFile is a (slightly arcane) XML file that indicates which other packages you depend on so they are linked at compile time. Often best to find someone else's that works and copy it.

Documentation for your package. Surprise people and write some.

C++ source code for any framework modules in this package. Also needs to include plugin definitions (more later).



C++ header files for your classes.

Python configuration files for your modules, containing sensible defaults for yours and other packages to load.

Test directory contains anything you want. Typically this contains your specialist test scripts and whatever root files/plots/core dumps they spit out.

The plugin and src directories are used somewhat interchangeably. In the plugin directory, put both source and header files here instead of using the interface directory. This is in theory for code that isn't directly usable as a module, but will be dynamically loaded by a runnable module.

gordon.ball@cern.ch
GPG 0x324543E5

CMSSW Modules

Imperial College London

- A module is a CMSSW plugin that represents a single algorithm. Made by extending one of three main classes.
- Any analysis process should be logically split into steps that use each of these.
- Each contains the same basic methods:-

```
constructor(ParameterSet)
~destructor()
void beginJob(EventSetup)
void endJob()
```

edm::EDAnalyzer

void analyze(Event,EventSetup)
Analyzes the event and produces output (eg histograms, ntuples) but doesn't modify the data.

edm::EDProducer

void produce(Event,EventSetup)
Take the event and save some new information into it.

edm::EDFilter

bool filter(Event,EventSetup)
Makes a filter decision on the event, returning false to stop processing this event.

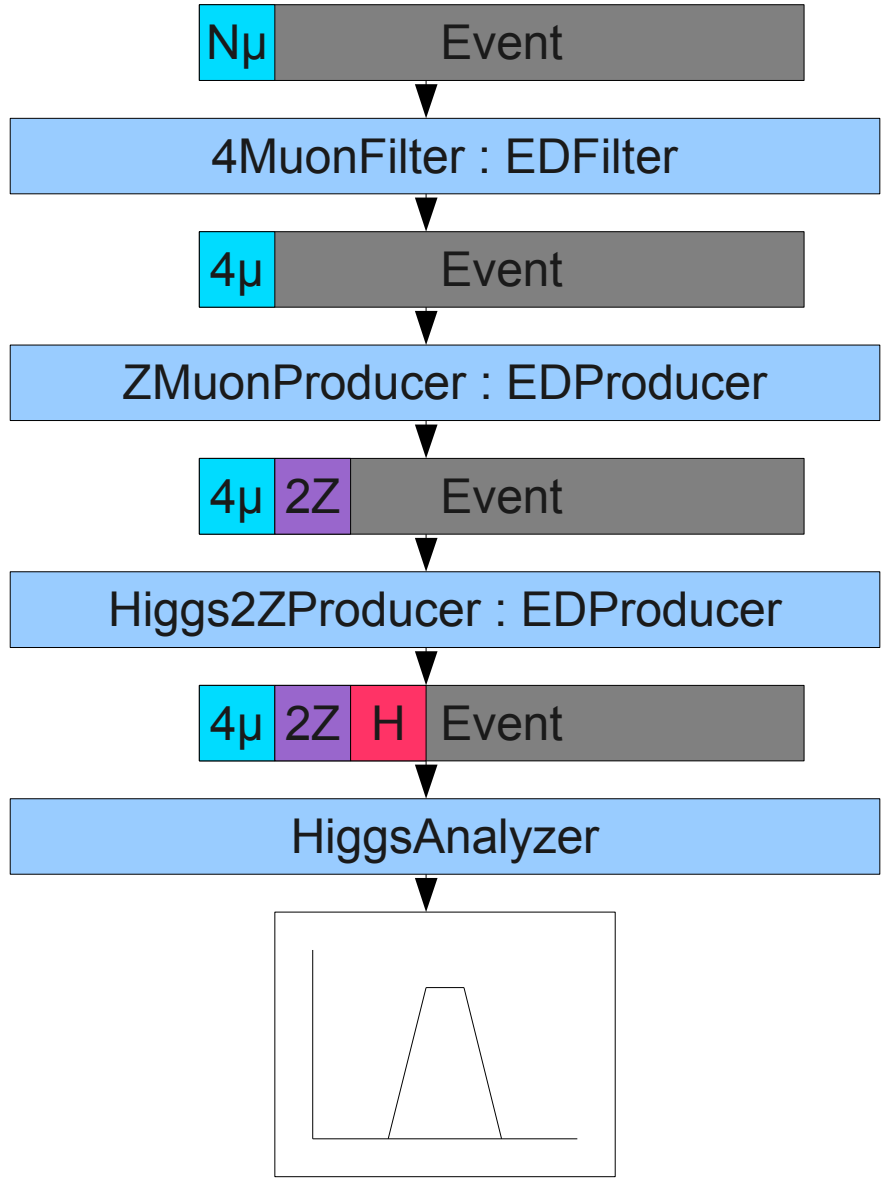
gordon.ball@cern.ch
GPG 0x324543E5

Structure Example

Imperial College London

- To analyze the H-»2Z-»4μ channel, for instance:
  - **Filter** events containing 4 identified muons
  - **Produce** Z candidates from pairs of muons with appropriate invariant mass
  - **Produce** Higgs candidate from Z candidates
  - **Analyze** distribution of Higgs candidates
  - **Win** free trip to Stockholm

(oversimplified, of course)



gordon.ball@cern.ch
GPG 0x324543E5

Writing a module

Imperial College London

- Getting parameters

Constructors have a const edm::ParameterSet& argument. This is a set of parameters supplied by the config file.

To get parameters:

```
MyAnalyzer::MyAnalyzer(const edm::ParameterSet& cfg) {
  d = cfg.getParameter<double>("myDouble");
  s = cfg.getParameter<std::string>("myString");
  it = cfg.getParameter<edm::InputTag>("myTag");
  vi = cfg.getParameter<std::vector<int> >("myArray");
}
```

You should be reading these into variables already declared in the class definition. This doesn't cover all the types that can be put in a parameter set but probably these are the ones you'll use most.

- Reading from the event

To read from an event you need an `edm::InputTag` or `const char*` string containing the object's label (probably from your parameter set), and an `edm::Handle` which will receive the object.

You need to include the necessary headers

```
#include
  DataFormats/<Package>/include/<Object>.h

void MyFilter::filter(const edm::Event& iEvent, const edm::EventSetup& iSetup) {

  edm::Handle<ObjectClass> myHandle;
  iEvent.getByLabel(myInputTag,myHandle);

}
```

edm::Handle behaves a lot like a pointer to the object. A reference to the object can be got with *myHandle;

or the object methods called directly with myHandle->method();

gordon.ball@cern.ch
GPG 0x324543E5

Working with objects

Imperial College London

The typical activity for a module is to load some collection of objects, examine them one by one and either store extra info or filter them.

After you've loaded an object (previous slide):

To loop over a collection

```
for (ObjectCollection::const_iterator
iter=handle->begin(); iter!=handle->end();
iter++) {
	std::cout << iter->method() << std::endl;
}
```

or

```
for (int i=0; i<handle->size(); i++) {
	ObjectRef ref(handle,i);
	std::cout << ref->method() << std::endl;
}
```

Common methods you might be interested in include:

p4() - Return the LorentzVector
charge()
pdgId() - Particle ID, if monte-carlo data
mass()
eta()
energy()
phi()
status() - decayed, final state, etc
numberOfDaughters()
daughter(i) – daughter particle i, candidate*
clone()

gordon.ball@cern.ch
GPG 0x324543E5

Writing a module

Imperial College London

- Writing data to an event

Data to be written to an event must first be declared in the constructor.

```
produces<ObjectClass>(“name”);
```

(name is optional and only required if you're creating multiple objects in one producer)

An auto pointer to the object is then created and inserted into the event.

```
void MyProducer::produce(edm::Event& iEvent, const edm::EventSetup& iSetup) {

  std::auto_ptr<MyObject> myProduct(new MyObject());

  iEvent.put(myProduct,”MyProduct”);

}
```

(Note that it is not necessary to supply a name for either the declaration or the put statement – this will be the _label field in the ROOT file that is usually left blank (if your producer only produces a single object).

- Declaring a plugin

For your new class to be seen as a plugin by CMSSW (and not just a class to be used by something else) you need to make a plugin declaration at the end of the file.

```
#include “FWCore/Framework/interface/MakerMacros.h”

DEFINE_FWK_MODULE(MyFilter);
```

gordon.ball@cern.ch
GPG 0x324543E5

Using ROOT / TFileService

Imperial College London

- You can use ROOT classes from within CMSSW
- You need to include the relevant header
  - #include “TFile.h”
  - #include “TH1F.h”
- Then they can be used normally
- Remember to **free** objects you create with **new** to avoid memory leaks
- You need to add “root” and “rootmath” to the BuildFile in this case

TFileService is a useful way of storing ROOT information from multiple modules in a single ROOT file.

In your python config

```
process.TFileService = cms.Service(“TFileService,
fileName = cms.string(“a.root”))
```

In your C++ code

```
#include "FWCore/ServiceRegistry/interface/Service.h"
#include
“CommonTools/UtilAlgos/interface/TFileService.h”

edm::Service<TFileService> fs;
TH1F* myHistogram = fs->make<TH1F>(“name”,
“label”, 1, 0, 1);
```

With this you can make any kind of ROOT object in as many modules as you want, which will all be stored in the same file.

gordon.ball@cern.ch
GPG 0x324543E5

Useful classes

Imperial College London

reco::Candidate
DataFormats/Candidate/interface/Candidate.h

Base class of other particles. Sometimes useful to cast them back to this.

deltaR
DataFormats/Math/interface/deltaR.h

Function to find the dR between two objects of any vaguely particle, vector or lorentzvector like types.

edm::RefToBase<T>
DataFormats/Common/interface/RefToBase.h

Use to get an `edm::Ref` to a base class from an `edm::Ref` of a superclass.

edm::View<T>
DataFormats/Common/inteface/View.h

Lets you create Handles to superclasses. Useful so you can load multiple collections, eg Electrons and Muons, into handles of the same type, or if you don't know what type of particle a generic function will handle

```
edm::Handle<edm::View<reco::Candidate> > anyCandidateHandle;
```

could be used to load any type of particle (although only to access non-specific information)

gordon.ball@cern.ch
GPG 0x324543E5

scramv1

Imperial College London

- To manage the merging of small amounts of your custom code, we have the **scram** tool
- Typically once you have used it to create a project area, the only command you will need is

```
scramv1 b
```

- This recursively builds all source under the current directory
  - Expect to be using this repeatedly
- Get online help

```
scramv1 command help
```

- Options for **scramv1 b**
  - -j n
    - Use multiple threads – useful if you're doing a big rebuild
  - clean
    - Delete all compiled files first and rebuild from scratch
  - (subsystem name)
    - Build only this subsystem
  - ProjectRename
    - If you have moved the project area, this rebuilds links

See https://twiki.cern.ch/twiki/bin/view/CMS/SWGuideScram for more info

gordon.ball@cern.ch
GPG 0x324543E5

CVS

Imperial College London

- CVS stands for Concurrent Version System, and is a system for managing contributions to code from many people
- To get software packages from other users you need to check them out of CVS into your project area
- You will probably be given access to commit changes to code in your group's packages
- You also have access to a UserCode area where you can store your own code

To check out some code into your project area

cd CMSSW_X_Y_Z/src
cmsenv

cvs co -r CMSSW_X_Y_Z Subsystem/Package

This checks out the code for that package for the corresponding CMSSW version.

In many cases you will instead be given the name of a “tag” to check out. A tag is a label applied to a particular set of file versions.

cvs co -r TAGNAME Subsystem/Package

In some cases you want the very latest version

cvs co Subsystem/Package

To see what versions you are currently using

showtags

CVS has long been obsolete compared to newer systems like SVN, Git, Mercurial and Bazaar, but we're stuck with it.

gordon.ball@cern.ch
GPG 0x324543E5

See http://ximbiot.com/cvs/manual/ for a CVS manual (as the URL might suggest).

CVS

- To update your local copy (after other people have made changes)
  - cvs update
- To add a file to the repository
  - cvs add filename
    - In a directory already in CVS
- To see the status of the files
  - cvs status
    - Detailed status of a file
      cvs status -v filename
- To see the differences between your copy and CVS
  - cvs diff filename
- None of the previous commands make any changes until you do
  - cvs commit
- You will be asked to write a log message and then the changes will be committed to CVS
- Good etiquette in this case is to always ask people before committing modifications to their code, and properly test and document your code before submitting it where other people will see it
  - This ought to be obvious but apparently isn't...

gordon.ball@cern.ch
GPG 0x324543E5

Config files

- Creating modules for CMSSW is only half the story – you also have to create config files that control how your job runs.
- A config file defines:
  - The source of the data (files, monte carlo generators, etc)
  - Which modules are in the path events are run through, and the parameters for each
  - Which background services should run
  - What happens to the data afterwards (store in a file, dump etc)
- Config files are written in python (from CMSSW 2), replacing an earlier custom text-based language.
- This means that you can do anything possible in python at startup time, which is useful
  - eg rename output files according to run parameters
  - you can also make your configs really baroque and confusing – be careful

gordon.ball@cern.ch
GPG 0x324543E5

Config files

Imperial College London

All configs must create a process object called 'process'. The label is applied to objects this process adds to the root files.

Create some modules. *MyFilter* or *MyAnalyzer* are names of C++ classes. The keyword arguments that follow make a ParameterSet. Note you have to use cms.* types instead of python primitives.

Two different methods of loading files from elsewhere. *process.load* executes the named file separately, while import loads the object into the current scope.

```
import FWCore.ParameterSet.Config as cms

process = cms.Process("TEST")

myfiles = cms.untracked.vstring(
    'file1.root',
    'file2.root'...
)
process.source = cms.Source(
    "PoolSource",
    fileNames = myfiles
)
process.maxEvents = cms.untracked.int32(-1)

process.myfilter = cms.EDFilter("MyFilter",
                        pi = cms.double(3.14)
                                )
process.myanalyzer = cms.EDAnalyzer("MyAnalyzer",
                        outfile = cms.string("test.root")
                                )
process.load("FWCore/MessageService/MessageLogger_cfi")
from MySubsystem.MyPackage.MyProducer_cfi import MyProducer

process.path1 = cms.Path(process.myfilter+process.myanalyzer)
process.path2 = cms.Path(MyProducer)
```

All configs need to import this

Create a list of files for the source to open.

Create a data source. This reads from the filenames, supplied, alternatively could be a MC generator.

Process unlimited events. Set to a positive number to only handle the first n events.

Create paths. Each path runs separately. Notice modules we've defined are 'added'. Other operations are possible (to determine how the results of filters are AND'd or OR'd together)

Config files

Imperial College London

Names ending **_cfi** or **_cff** are a hangover from the old system (configuration file fragment). They are not essential but now seem to be a convention.

Typically you should create a config file fragment for each class in **python/ClassName_cfi.py** containing default values which other people can then easily import.

Symlinks for python files are created by doing **scramv1 b** in your package directory. Files in **python/** can then be imported as **SubSystem.Package.Example_cfi** (Subsystem/Package/python/Example_cfi.py)

Valid classes in python ParameterSets include:

Python	C++
cms.double	**double**
cms.int32	**int**
cms.uint32	**unsigned int**
cms.string	**std::string**
cms.bool	**bool**
cms.v*	**std::vector<*>**
cms.InputTag	**edm::InputTag**
cms.PSet	**edm::ParameterSet**
cms.V*	**std::vector<*>**

- Finally, you should have a package containing some C++ analysis/filter/production classes and one or more python config files.
- Running it is then just a matter of doing
  - cmsRun myconfig.py

gordon.ball@cern.ch
GPG 0x324543E5

Working with data

Imperial College London

- Working with real data involves complications not involved in MC
  - You need to look up the luminosity of a dataset after your jobs have run (later)
  - You need to be aware that the available branches and names of triggers may change during a dataset, and handle this appropriately
- Data is usually available in “PromptReco” and “ReReco” flavours
  - PromptReco is reconstructed immediately after it is recorded, using the current CMSSW data taking version
  - ReReco passes are done about monthly, and the whole dataset is reconstructed using a single, usually newer, CMSSW version.

gordon.ball@cern.ch
GPG 0x324543E5

Useful tools

- edmConfigBrowser
  - Interactive display of the contents of a configuration file
- edmConfigEditor
  - Interactive editor for configuration files
- edmConfigToHTML
  - Converts a config file to an HTML page, making it easier to view and search
- edmConfigToGraph
  - Converts a config file into an image showing the data relationships between modules
- edmFileUtil
  - Allows you to examine a CMSSW root file, eg for the runs/lumi it contains and all the branches it contains. Very useful.
- edmMakePhDThesis
  - Yes, this actually exists.

gordon.ball@cern.ch
GPG 0x324543E5

Useful Links

Imperial College London

- CMSSW CVS Repository
  - http://cmssw.cvs.cern.ch
- CMSSW Lexer (search for classes or names in CMSSW)
  - http://cmslxr.fnal.gov/lxr
- CMS Twiki (CERN login required)
  - http://twiki.cern.ch/twiki/bin/view/CMS/WorkBook[TopicName]
- CMS Hypernews (registration required)
  - http://hn.cern.ch/cms
- CMS Software Guide
  - http://twiki.cern.ch/twiki/bin/view/CMS/SWGuide[TopicName]
- CMS Doxygen (Automatic software documentation)
  - http://cmssdt.cern.ch/SDT/

gordon.ball@cern.ch
GPG 0x324543E5

Where to get help

Imperial College London

- TWiki
  - There is lots of information here, but it isn't easy to search. Knowing the names of things will help a bit – and of course, if you need to find something and it isn't there then add it...
- Hypernews
  - Hypernews forums are usually the best place to contact an expert in the relevant field, and get an answer quickly. Beware of being overwhelmed with hundreds of daily emails if you subscribe to many though...
    - (I have a script for reading hypernews with RSS if you find it useful – the ideal mechanism would be NNTP but this doesn't appear to be possible either).
- The web
  - If you have C++ or Python problems, just seaching for the exact error will usually bring up solutions to it...
- People
  - Biting is fairly rare.
  - Any of us for CMSSW questions
- Imperial Mailing List
  - hep-cms-computing

gordon.ball@cern.ch
GPG 0x324543E5

Useful shell commands

Imperial College London

- Since almost all work is done in Linux, it's helpful to know shell commands which can make your life easier or get you out of problems
- If you need help try
  - `command --help` (quick help)
  - `man command` (manual page)
  - `whatis command` (one line summary)
  - `apropos query` (search manual pages)
  - Search the web
- List of useful commands at http://www.hep.ph.ic.ac.uk/~gfball/tmp/cheatsheet.pdf

- Useful shell tricks

`cmd` Run command

`cmd &` Run command, detached from terminal (eg, open an editor while still allowing you to work in the terminal)

`cmd1 | cmd2` "Pipe" the output of cmd1 as input to cmd2

`cmd > filename` Store the output of cmd in file

`cmd < filename` Run cmd with input data in file

gordon.ball@cern.ch
GPG 0x324543E5

Useful shell commands

Imperial College London

- There are different shells, and the commands used are not always the same
- You will probably encounter
- "sh-like"
  - `bash`
  - `dash`
  - `sh`
- "csh-like"
  - `csh`
  - `tcsh`
- Pick one and stick with it. I recommend `bash`, but you'll find idealogues for each...

- Some important concepts

·Source-ing
  - The command "`source filename`" runs the named file and updates your environment variables with any changes the file makes
  - Many operations (eg, Grid copying), require you "source" the appropriate environment first

·Environment Variables
  - These are values set in your shell that programs running in your shell can see. Eg, the variable PATH tells the shell which directories to search to find a command you issue.
  - To see a list of environment variables, use "`env`"
  - To set an environment variable, use "`export NAME=value`" in bash and "`setenv NAME=value`" in csh
  - To see the value of an individual variable, use "`echo $NAME`"

gordon.ball@cern.ch
GPG 0x324543E5

Shell concepts

Imperial College London

- STDOUT
  - this is the data that the command is printing back to the terminal - “standard output”
  - it can be redirected to a file with `cmd > file`
- STDIN
  - many commands can accept a stream of data
    - `cmd < file`
    - `othercmd | cmd`
- STDERR
  - errors are also printed to the terminal, but in a seperate stream
    - `cmd 2> errors`
    - `cmd &> alloutput`
- Keyboard shortcuts
  - Ctrl-C - (usually) terminate a running program
  - Ctrl-D - exit the current shell
  - Ctrl-Z – suspend running program
    - this is useful for getting back to the terminal from an unresponsive program so you can kill it
  - TAB – try and autocomplete file or command names

gordon.ball@cern.ch
GPG 0x324543E5

Scripting

Imperial College London

- There are many repetitive commands you can automate
- The file ".bashrc" in your home directory contains commands that will be run on login
  - Use it to set up commonly used environments and your aliases
- Aliases allow you to replace a long string of commands with a shorter one

```
alias shortname='very_long_command'
```

- To write a script file

Create a new file, usually with a .sh extension

The first line should be the magic
`#!/bin/bash`
This tells the shell to execute this text file with the named program. You can use this for other types of files, eg `#!/usr/bin/python`

Add your commands one per line and save the file.

Make the file executable with
`chmod +x script.sh`

Run the script with
`./script.sh`

```
#!/bin/bash

echo "Hello World"
```

gordon.ball@cern.ch
GPG 0x324543E5

Possibly useful software

Imperial College London

- screen
  - A detachable terminal (so you can log out of a remote linux machine and your jobs keep running)

```
screen
(some long command)
(logout)
(go to pub)
(login)
screen -r
(command still running)
```

- If you only need to run a single command "`nohup command`" detaches it from your current terminal (but doesn't let you rejoin the session like screen does)

- sshfs
  - SSHFS allows you to mount a directory on a remote linux machine using SSH, and then access the contents as if they were local files
  - So you can use eg, editor programs locally without the annoyances of using them over X forwarding
  - You can compile AFS kernel modules for direct access to your CERN AFS space if you want more of a challenge

```
sshfs user@remote:/ /mnt/remote

cd /mnt/remote
(filesystem of remote machine)
```

gordon.ball@cern.ch
GPG 0x324543E5

Possibly useful software

Imperial College London

- unison
  - linux synchronization software, useful if you want to keep work files synchronised eg between laptop, imperial and CERN
- synergy
  - keyboard/mouse sharing software, crossplatform, useful if you have a laptop and desktop on your desk
- zotero
  - firefox plugin that works as a reference manager to keep track of all your papers/ citable websites
- zim
  - a linux desktop wiki, which might be useful for keeping a labbook type thing
- ROOT
  - if you must, repositories for this can be found for most major linux distros so you can have a copy on your laptop
- eclipse
  - fairly heavyweight development environment, which can be used for python/java/c++ if you prefer not just using a text editor
- git
  - a much better revision tool than CVS, but with a bit of a learning curve

gordon.ball@cern.ch
GPG 0x324543E5