Geant4 — an Introduction
Lecture — 2

Experimental High Energy Particle and Astroparticle Physics
February 17, 2026

Sunanda Banerjee

What we Learnt so far

- Workflow of a Geant4 Application
- Terminologies used within Geant4
- Modelling of a detector in an application
  - Materials
  - Logical and Physical Volumes
- Extraction of Useful information
  - Sensitive detectors and Hits
  - Step points and Touchables

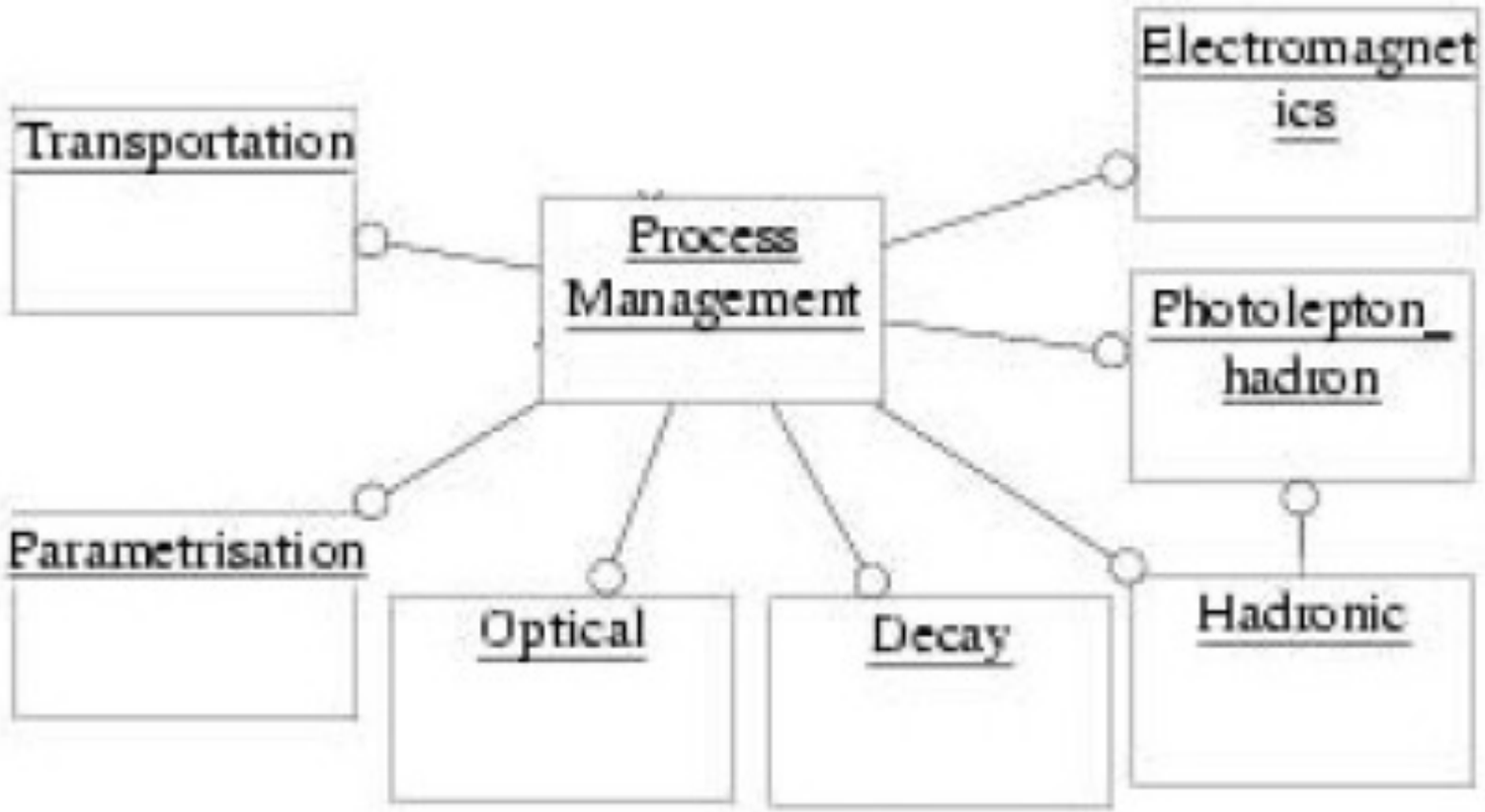
Today's Menu

- Physics Models Geant4 provides
  - Electromangentic
  - Hadronic Physics
- Physics List
- Some example codes to be used for making an application

Physics Lists

- Geant4 provides the possibility of simulating the physics processes of a variety of particles
  - each such particle can have interactions of different types: strong, electromagnetic or weak and each such interaction can be described by different models
  - unlike many other simulation tools, Geant4 leaves it to the user to decide on which particles, which interactions and which models are to be used during the simulation step
  - declaration of the list of particles and the choice of models is done using the physics list
  - the toolkit provides handles for a few well-defined physics lists which are suitable for certain specific types of application

Processes in Geant4



- Processes describe how particles interact with material or with a volume

- Three basic types:
  - At rest process (e.g. decay at rest)
  - Continuous process (e.g. ionisation)
  - Discrete process (e.g. Compton scattering)

- Transportation is also a process
  - Interacting with the volume boundary

- A process which requires the shortest interaction length limits the step

Types of Processes

- There are several types of modules which can be combined to define the physics list
  - electromagnetic physics
  - extra physics processes for photons and leptons
  - decays
  - hadronic elastic
  - hadronic inelastic
  - stopping particles and capture processes
  - ion nuclear interactions
  - step limits
  - others

- The others category includes optical photons, exotic physics processes, thermal neutron transport models, ….

- The user needs to define the physics list through the 3 handles:
  - RegisterPhysics
  - ReplacePhysics
  - RemovePhysics

An Example of Making a Physics List

```
// EM Physics
RegisterPhysics(new G4EmStandardPhysics(verbosity));

// Synchroton Radiation & GN Physics
G4EmExtraPhysics* gn = new G4EmExtraPhysics(verbosity);
RegisterPhysics(gn);

// Decays
this->RegisterPhysics(new G4DecayPhysics(verbosity));

// Hadron Elastic scattering
RegisterPhysics(new G4HadronElasticPhysics(verbosity));

// Hadron Physics
RegisterPhysics(new G4HadronPhysicsFTF_BIC(verbosity));

// Stopping Physics
RegisterPhysics(new G4StoppingPhysics(verbosity));

// Ion Physics
RegisterPhysics(new G4IonPhysics(verbosity));

// Neutron tracking cut
RegisterPhysics(new G4NeutronTrackingCut(verbosity));
```

Electromagnetic Physics

- Applicable to
  - electrons and positrons
  - γ, X-ray and optical photons
  - muons
  - charged hadrons
  - ions

- Several physics models are available.
  Standard EM physics is extended to low energies using many data-driven techniques to improve the quality of simulation at low energies

- *All obey the same abstract Process interface*: transparent to tracking

Models available for
- Multiple scattering
- Bremsstrahlung
- Ionization
- Annihilation
- Photoelectric effect
- Compton scattering
- Pair production
- Rayleigh scattering
- γ conversion
- Synchrotron radiation
- Transition radiation
- Reflection, refraction
- Cherenkov radiation
- Scintillation
- .....

Geant4 Options for EM Physics

- There are 10+ options for EM physics
  - opt1 (EMV): a fast but less precise version used in HE physics lists
  - opt2 (EMX): also a fast and less precise version for HE physics lists
  - opt3 (EMY): provides a more accurate simulation of photons and charged hadrons
  - opt4 (EMZ): most precise but slow description of EM physics for HE applications
  - LIV: similar to opt3 but models for photos and electrons make use of Livermore set of models
  - PEN: similar to opt3 with models for photons and electrons making use of Penelope set of models
  - _GS: substitute Urban multiple scattering models with the Goudschmidt-Sanderson model
  - _LE: low energy WentzelVI model is used for multiple scattering
  - WVI: WentzelVI model and ATIMA ion ionisation models are used for a better description of multiple scattering
  - _SS: single scattering models used on top of standard EM configuration

- Please note that the same model describes EM physics over the entire energy region

Hadronic Processes

- Hadronic processes are often implemented in terms of a model class

- There are usually several models for a given process
  - user must choose
  - can, and sometimes must, have more than one per process

- A process must also have cross sections assigned
  - here too, there are options

- Default cross-section sets are provided for each type of hadronic process
  - fission, capture, elastic, inelastic
  - can be overridden or completely replaced

- Different types of cross-section sets exist
  - some contain a few numbers to parametric the cross-section as a function of energy
  - some represent large databases
  - some are purely theoretical (equation driven)

Alternative Cross Sections

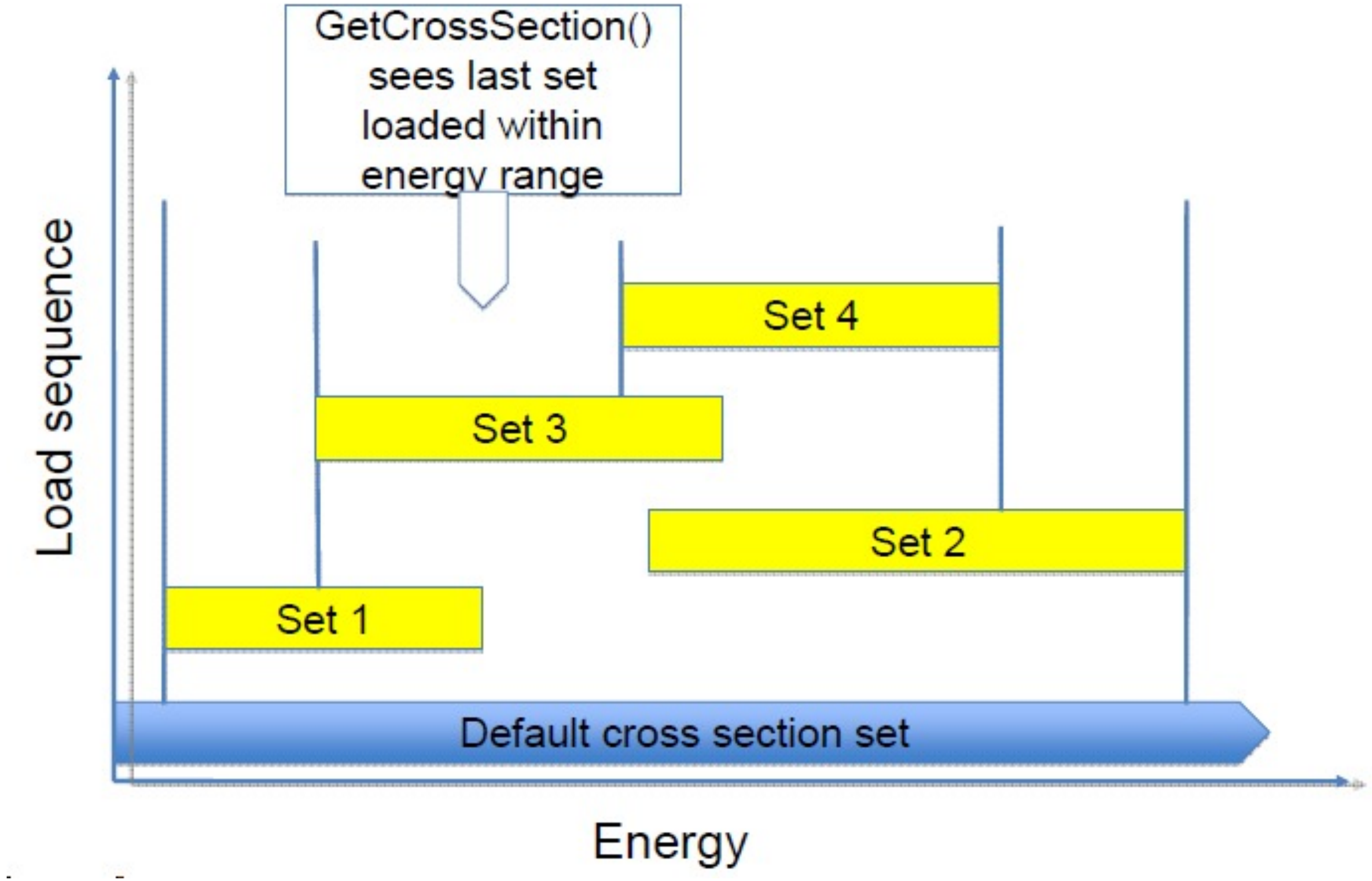
- Cross-section databases are available for low-energy neutrons
  - G4NDL available among the Geant4 distribution files
  - Livermore database (LEND) is also available
  - these are available with or without thermal cross-sections

- Cross section table is available for medium energy neutrons and protons
  - 14 MeV < E < 20 MeV

- Several alternatives exist for ion-nucleus cross-section
  - these are empirical and parametrised cross-section formulae with some theoretical insight
  - these are good for E/A < 10 GeV

- Alternative cross-sections also exist for pion cross-section

Cross Section Management



Data Driven Hadronic Models

- These are characterised by lots of data on
  - cross sections
  - angular distributions
  - multiplicities, etc.

- To get interaction length and final state, these models depend on interpolation of data
  - cross sections, Legendre coefficients, ..

- Examples:
  - neutrons with E < 20 MeV
  - coherent elastic scattering (pp, np, nn)
  - radioactive decays

Models in Hadronic Interactions

- Data-driven models: When sufficient data are available with sufficient coverage over the phase space, a data-driven approach is the optimal way
  - neutron transport, photon evaporation, absorption at rest, isotope production, inclusive cross section, ……

- Parametrised models: Extrapolation of cross sections and parametrisation of multiplicity distributions and final state kinematics
  - adaptation of GHEISHA in some earlier versions of Geant4

- Theory-based models: These include a set of theoretical models describing hadronic interactions depending on the addressed energy range
  - diffractive string excitation, dual parton model or quark-gluon string model at medium to high energies
  - intra-nuclear cascade models air medium to low energies
  - nuclear evaporation, fission models, …… at very low energies

Theory Driven Hadronic Models

- These are dominated by theoretical arguments (QCD, Glauber theory, exciton model, …)
- Final states (number and type of particles in the final sate with their energy and angular distributions) are determined by sampling theoretically calculated distributions
- This type of models is preferred as they are the most predictive
- Examples:
  - quark-gluon string models (projectiles with E > 20 GeV)
  - intra-nuclear cascade models (intermediate energies)
  - nuclear de-excitation and break-up

Parametrised Hadronic Models

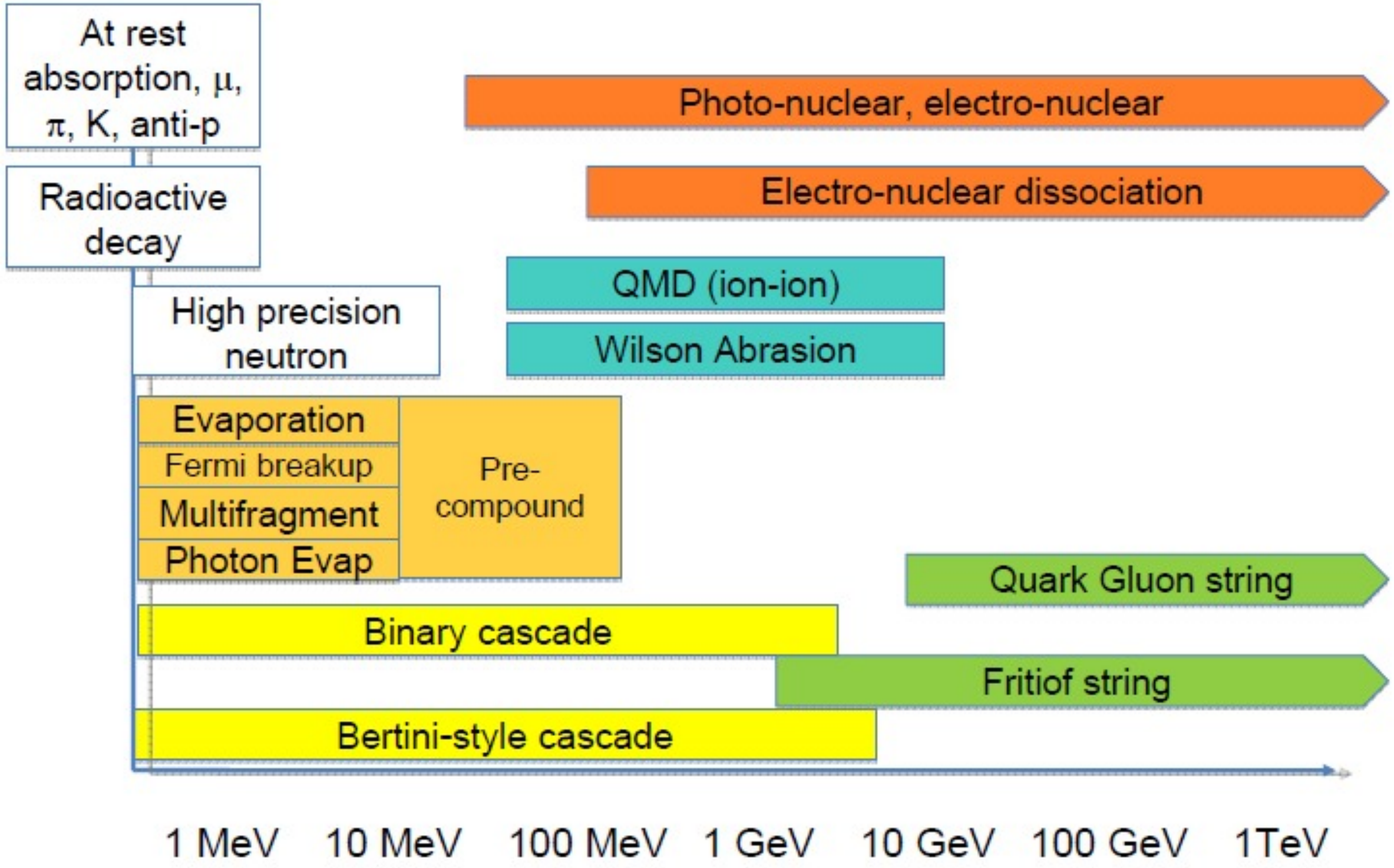
- Current versions do not contain any parametrised version. In versions preceding Geant4 10.0, two models existed. They were re-engineered versions of the Fortran Gheisha code used in Geant3

- These models depended mostly on fits to data with some theoretical guidance

- Two such models existed:
  - Low Energy Parametrised (LEP) for E < 20 GeV
  - High Energy Parametrised (HEP) for E > 20 GeV
  - each type referred to a collection of models (one for each type of hadron)

- These codes were fast and existed for all types of particles. But they were not detailed enough and there was no-one to maintain these codes

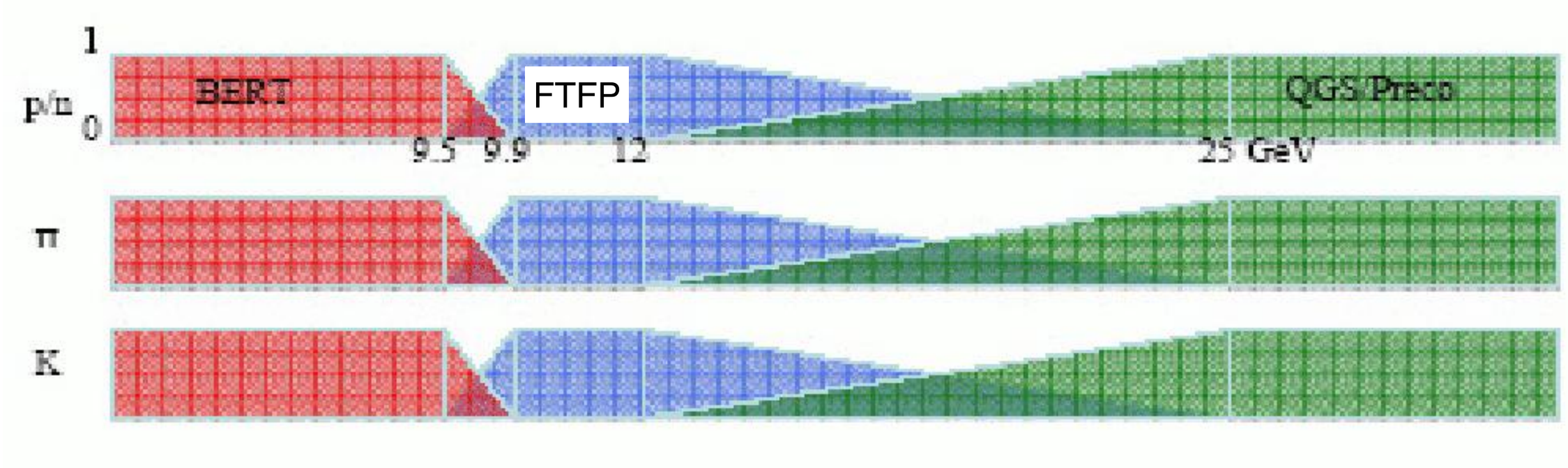
Partial Hadronic Model Inventory



Inelastic Hadronic Interactions

- No single model for hadron inelastic process can cover the entire energy region required in a high-energy physics experiment
  - Quark-gluon string models are good at high energies
  - Nuclear cascade models are good at medium and low energies
  - At very low energies, models for fission and pre-combination are required

- So all physics lists for inelastic hadronic interaction combine a number of models



Physics List Library

- There are a number of Physics Lists available in the Geant4 library which can be directly incorporated into the user code
  - FTFP_BERT — High Energy Experiments
  - FTFP_BERT_ATL
  - FTFP_BERT_HP
  - FTFP_BERT_TRV
  - FTFP_INCLXX
  - FTFQGSP_BERT
  - FTF_BIC
  - QBBC — Medical and Space Physics Applications
  - QGSP_BERT — Former default for High Energy Experiments
  - QGSP_BERT_HP
  - QGSP_BIC — Cosmic Ray applications
  - QGSP_BIC_AllHP
  - QGSP_BIC_HIP
  - QGSP_FTFP_BERT
  - QGSP_INCLXX
  - QGS_BIC
  - Shielding — Recommended for shielding studies
  - ShieldingLEND
  - LBE
  - NuBeam

Main Program

- Simple example of the main program:

```
#include "G4RunManager.hh"
#include "G4UImanager.hh"
#include "Randomize.hh"
#include "time.h"
#include "MyDetectorConstruction.hh"
#include "MyEventAction.hh"
#include "MyPrimaryGeneratorAction.hh"
#include "G4PhysListFactory.hh"
#include "QGSP_BERT.hh"

int main(int argc,char** argv) {
// Choose the Random engine
CLHEP::HepRandom::setTheEngine(new CLHEP::RanecuEngine);
// Set random seed with system time
G4long seed = time(NULL);
CLHEP::HepRandom::setTheSeed(seed);

// Construct the default run manager, which manages start and stop simulation
G4RunManager * runManager = new G4RunManager;

// Set mandatory initialization classes:
// =====================================
// Initilization detector construction class
MyDetectorConstruction* detector = new MyDetectorConstruction;
runManager->SetUserInitialization(detector);

G4PhysListFactory factory;
runManager->SetUserInitialization(factory.GetReferencePhysList("QGSP_BERT"));
```

Main Program (cont)

```
// Set user generator action class
MyPrimaryGeneratorAction* genAction = new MyPrimaryGeneratorAction();
runManager->SetUserAction(genAction);

// Set user event-action class
MyEventAction* eventAction = new MyEventAction(detector);
runManager->SetUserAction(eventAction);

// Initialize G4 kernel
runManager->Initialize();

// Get the pointer to the User Interface manager
G4UImanager* UI = G4UImanager::GetUIpointer();

G4String command = "/control/execute ";
G4String fileName = argv[1];
UI->ApplyCommand(command+fileName);

// Job termination
// Free the store: user actions, physics_list and detector_description are
//                 owned and deleted by the run manager, so they should not
//                 be deleted in the main() program !
delete runManager;

return 0;
```

Hit Class

```
#include "G4VHit.hh"
#include "G4THitsCollection.hh"
#include "G4Allocator.hh"

class G4AttDef;

class MyHit : public G4VHit {

public:
  MyHit();
  MyHit(G4int id, G4double e, G4double t);
  ~MyHit() = default;
  MyHit(const MyHit &right);
  const MyHit& operator=(const MyHit &right);
  G4int operator=(const MyHit &right) const;

  inline void *operator new(size_t);
  inline void operator delete(void *aHit);
  void Draw() {}
  void Print() {}

private:
  G4int           cellID_;
  G4double        edep_, time_;

  ...
```

```
public:
  inline void setCellID(G4int id) { cellID_ = id; }
  inline G4int cellID() const { return cellID_; }
  inline void setEdep(G4double de) { edep_ = de; }
  inline void addEdep(G4double de) { edep_ += de; }
  inline G4double getEdep() const { return edep_; }
  inline void setTime(G4double t) { time_ = t; }
  inline G4double time() const { return time_; }

};

typedef G4THitsCollection<MyHit> MyHitsCollection;
extern G4Allocator<MyHit> MyHitAllocator;

inline void* MyHit::operator new(size_t) {
  void *aHit;
  aHit = (void *) MyHitAllocator.MallocSingle();
  return aHit;
}

inline void MyHit::operator delete(void *aHit) {
  MyHitAllocator.FreeSingle((MyHit*) aHit);
}
```

Detector Construction

```
#include "G4VUserDetectorConstruction.hh"

#include "G4RunManager.hh"
#include "G4LogicalVolume.hh"
#include "G4PVPlacement.hh"
#include "G4SystemOfUnits.hh"
#include "G4VPhysicalVolume.hh"
#include "G4LogicalVolume.hh"

class MyDetectorConstruction : public G4VUserDetectorConstruction {
public:
  MyDetectorConstruction() {}
  ~MyDetectorConstruction() override = default;
  G4VPhysicalVolume* Construct();

  void setSensitive();

private:
  G4LogicalVolume   *logSens;
};
```

Sensitive Detector

```
#include "G4VSensitiveDetector.hh"
#include "MyHit.hh"

class MyDetectorConstruction;
class G4Step;
class G4HCofThisEvent;
class G4TouchableHistory;

class MySensitiveDetector : public G4VSensitiveDetector {
public:
  MySensitiveDetector(MyDetectorConstruction*, G4String);
  ~MySensitiveDetector();

  void     Initialize(G4HCofThisEvent*HCE);
  G4bool   ProcessHits(G4Step*aStep, G4TouchableHistory*ROhist);
  void     EndOfEvent(G4HCofThisEvent*HCE);
  void     clear();
  void     DrawAll();
  void     PrintAll();

private:
  const MyDetectorConstruction *detector_;
  MyHitsCollection *calCollection_;
  G4int hcID_;

};
```

Primary Generator Action

```
#include "G4VUserPrimaryGeneratorAction.hh"
#include "globals.hh"
#include "G4ParticleGun.hh"
#include "G4ThreeVector.hh"

class G4Event;

class PrimaryGeneratorAction : public G4VUserPrimaryGeneratorAction {

public:
  PrimaryGeneratorAction();
  virtual ~PrimaryGeneratorAction();

  void GeneratePrimaries(G4Event*);
  G4ParticleGun* GetParticleGun() {return particleGun;};
private:
  G4ParticleGun*                  particleGun;      //pointer a to G4  class
  G4double                        xVertex, yVertex, zVertex;
};
```

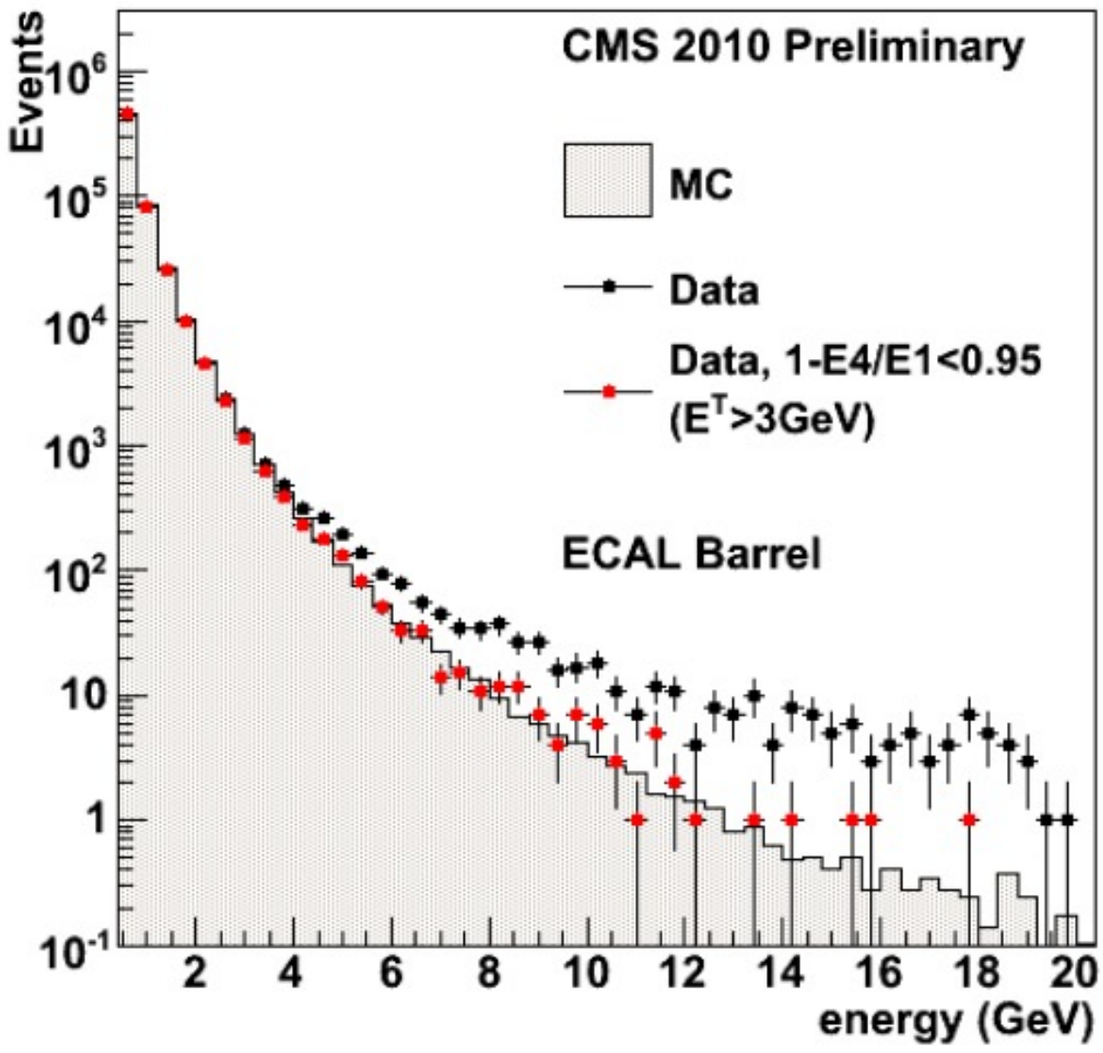
EventAction

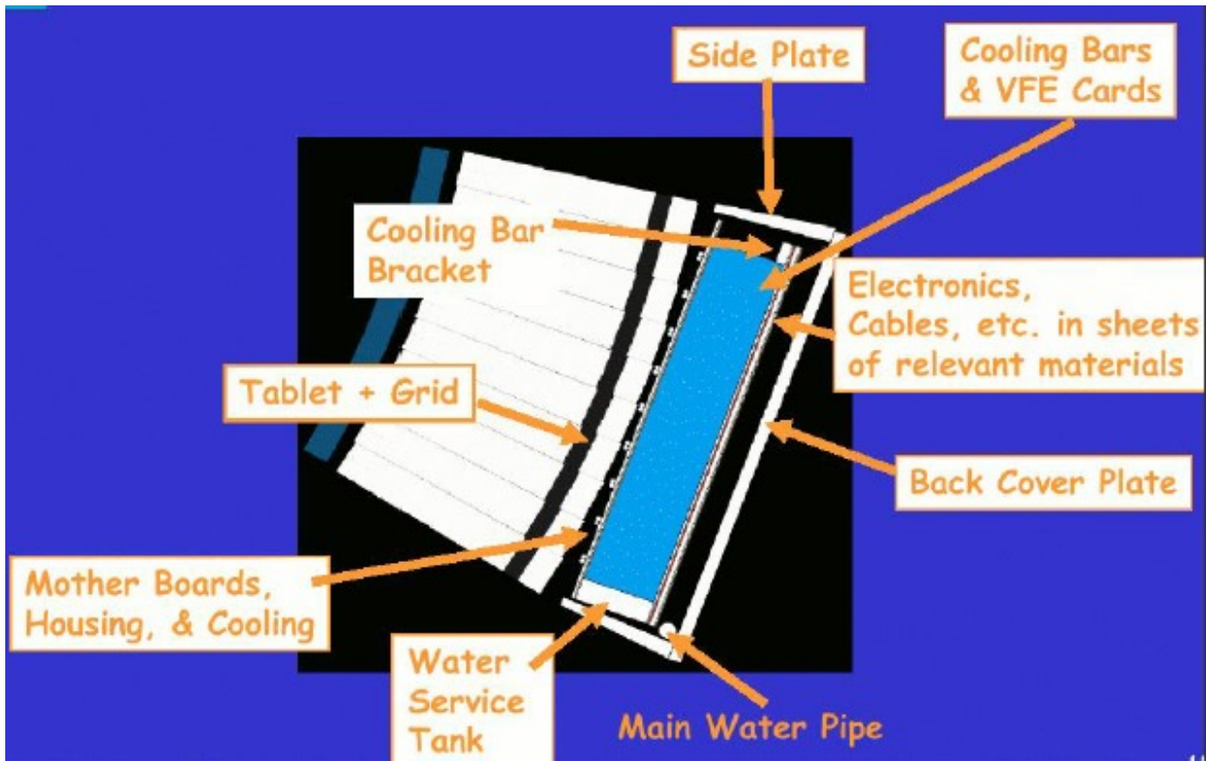
```
#include <iostream>
#include <vector>
#include "TFile.h"
#include "TTree.h"
#include "G4UserEventAction.hh"
#include "globals.hh"

#include "MyDetectorConstruction.hh"

class MyEventAction : public G4UserEventAction {

public:

  MyEventAction(MyDetectorConstruction *det);
  virtual ~MyEventAction();

  void  BeginOfEventAction(const G4Event*);
  void    EndOfEventAction(const G4Event*);

private:
  const MyDetectorConstruction *detCon;

  ///tree variables
  std::vector<int> *cellX, *cellY, *layer;

  G4double gunPx, gunPy, gunPz, gunP, gunPt, gunE;
  G4double distX, distY, gunX, gunY, gunZ;

  TFile *file;
  TTree *tree;
};
```

Additional Slides

Understanding Issues

- During early studies of collisions at the large hadron collider, CMS experiment observed some unusual number of events in their detector system

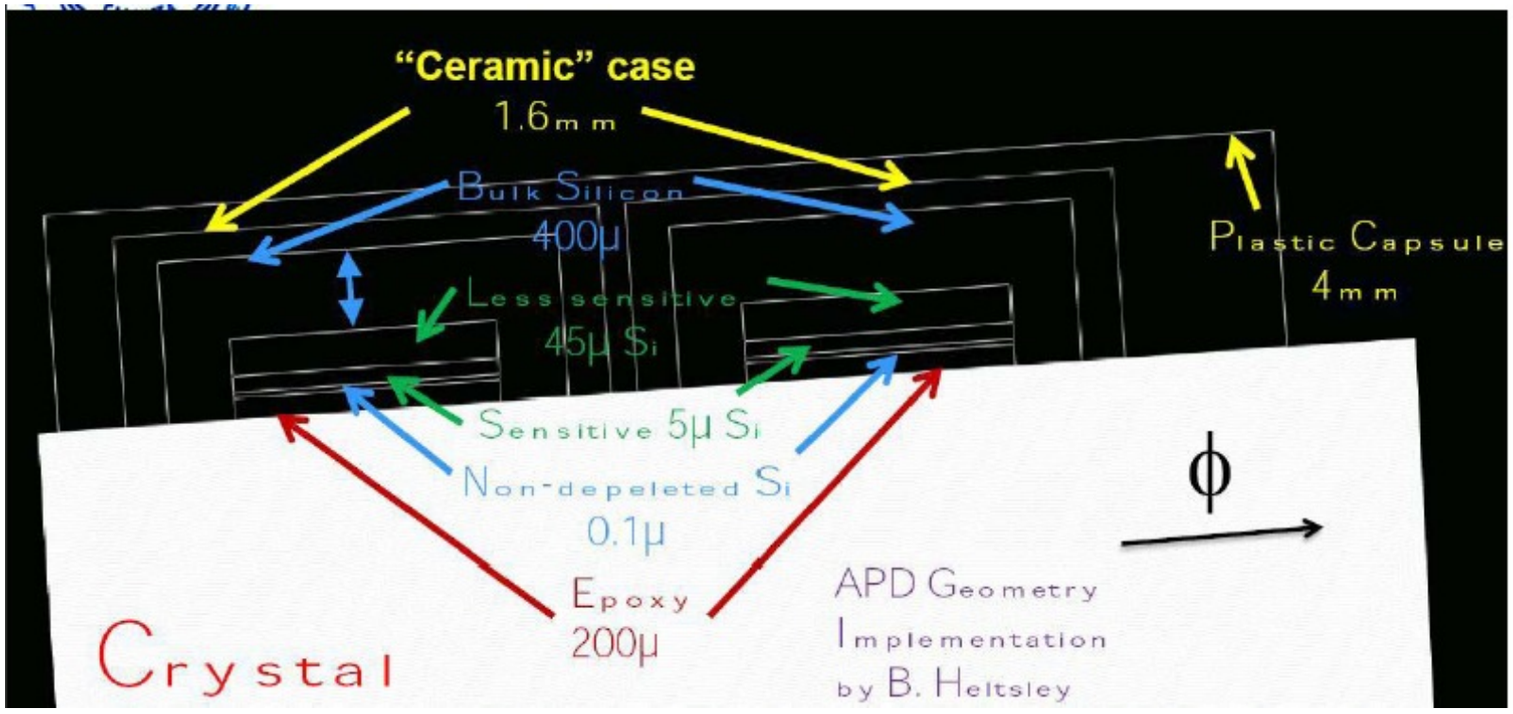


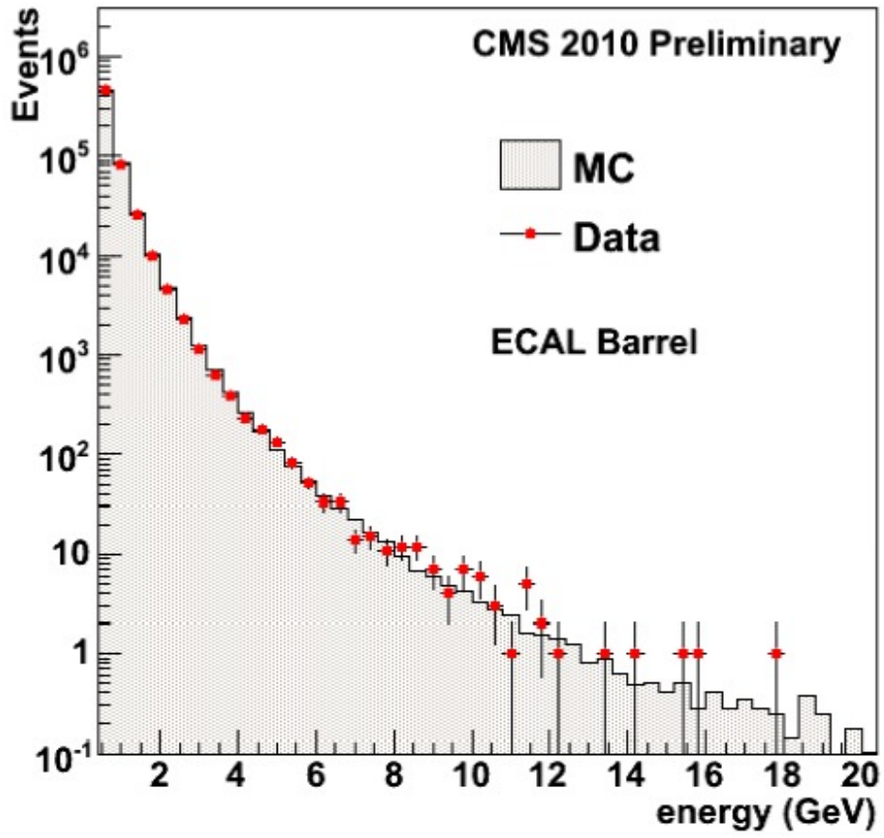


- CMS measured energy deposited in a crystal calorimeter. The excess was observed at higher energies

Understanding Issues

- CMS tried to understand the readout unit of the calorimeter and realised there could be excess energy if some heavily ionising particles go through the readout unit. This excess is not light produced in the crystal but charge deposited in the readout unit





- Simulating the readout unit removed the observed excess

- A qualitative as well as quantitative understanding of everything observed is crucial in providing results with the highest precision and confidence

- Simulation and Geant4 provide this confidence