Introduction to Geant4

Matteo Negrini
INFN-Bologna

INFN

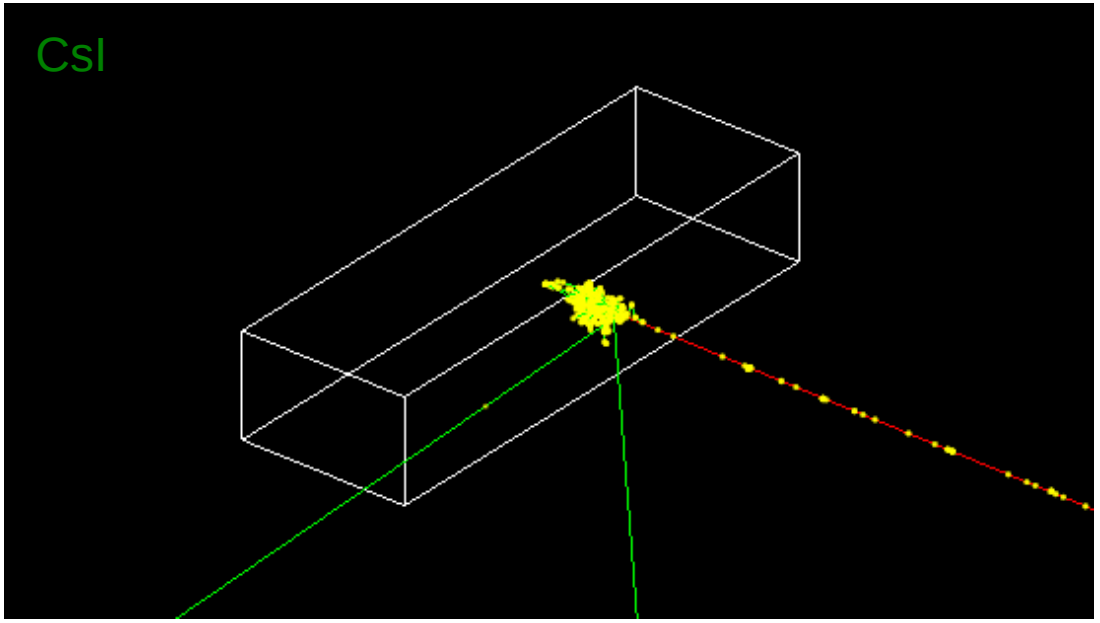
What is Geant4?

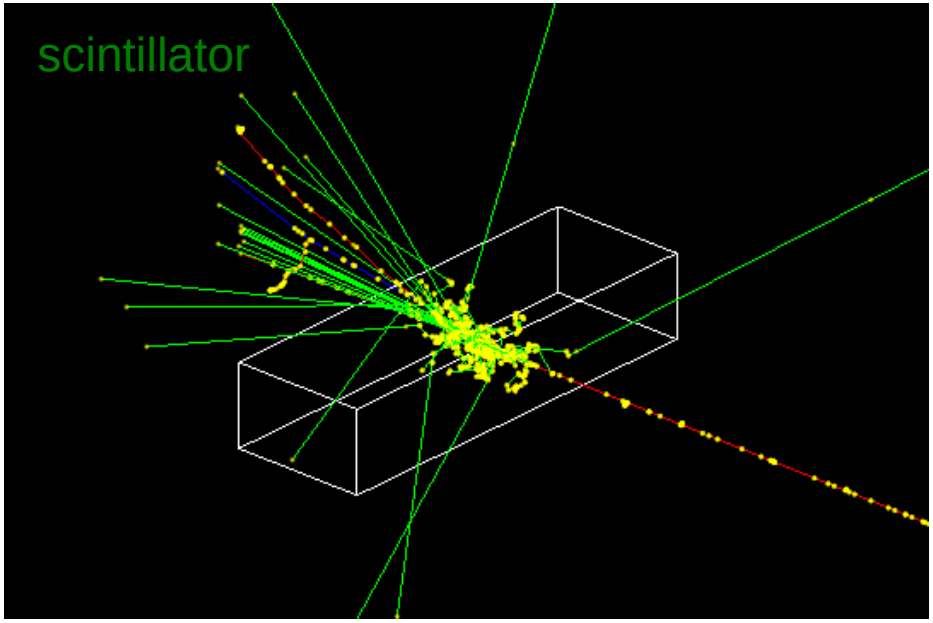
- Geant4 website: http://geant4.cern.ch/
- According to the website: “Geant4 is a toolkit for the simulation of the passage of particles through matter. Its areas of application include high energy, nuclear and accelerator physics, as well as studies in medical and space science.”
- Open source, object oriented, coded in C++
- Several examples available (provide useful guidance in developing applications)
- Applications:
  - detector design (geometry/material optimization, backgrounds, ...)
  - detector response to particles (signal amplitudes, energy spectra, resolutions, efficiencies, ...)

INFN

What is Geant4?

Simulations of an electron impacting on a block of CsI or scintillator





Images from SLAC Geant4 tutorial: http://geant4.slac.stanford.edu/SLACTutorial14/HandsOn2/

Before we start...

- Download code from the Geant4 website
- Follow installation guide: http://geant4.web.cern.ch/geant4/UserDocumentation/UsersGuides/InstallationGuide/html/index.html
- Before to proceed, you should have / install:
  - C++11 compiler
  - cmake 3.3 (or higher)
  - CLHEP library (an internal version is supplied with the Geant4 source)
  - Additional packages are needed for optional features, such as geometry visualization
- Run the geant4 setup script each time you login (linux tip: execute the script in .bashrc)
- Compile and run one example

INFN

Goal: the exercise

- The exercise is the simulation of a sampling calorimeter to test its performance on given particles
- A particle gun event generator is used

Particle gun shoots one particle

- It is derived from the standard Example B4c
  - You should: define the detector geometry, customize the primary generator, collect hits at each event, extend hit information for the analysis
  - The meaning of all these steps will be discussed in these lectures
- The example B4c can be copied in a clean dir and compiled in the same way as all Geant4 examples
  - Make a new directory where the example is run
  - From this dir (*cmake* only the 1st time, *make* every time something is changed):

```
> cmake -DGeant4_DIR=<Geant4InstallationDir> ../<dirWithB4cExample>
> make
```

Some useful links

- Getting started:

  http://geant4.cern.ch/support/gettingstarted.shtml

- In particular: user's guide for application developers:

  http://geant4.web.cern.ch/geant4/UserDocumentation/UsersGuides/ForApplicationDeveloper/html/index.html

- Online code browser:

  http://www-geant4.kek.jp/Reference/

- Slides of these lectures:

  http://www.bo.infn.it/~negrini/MonteCarloIntro.pdf

  http://www.bo.infn.it/~negrini/Geant4.pdf

INFN

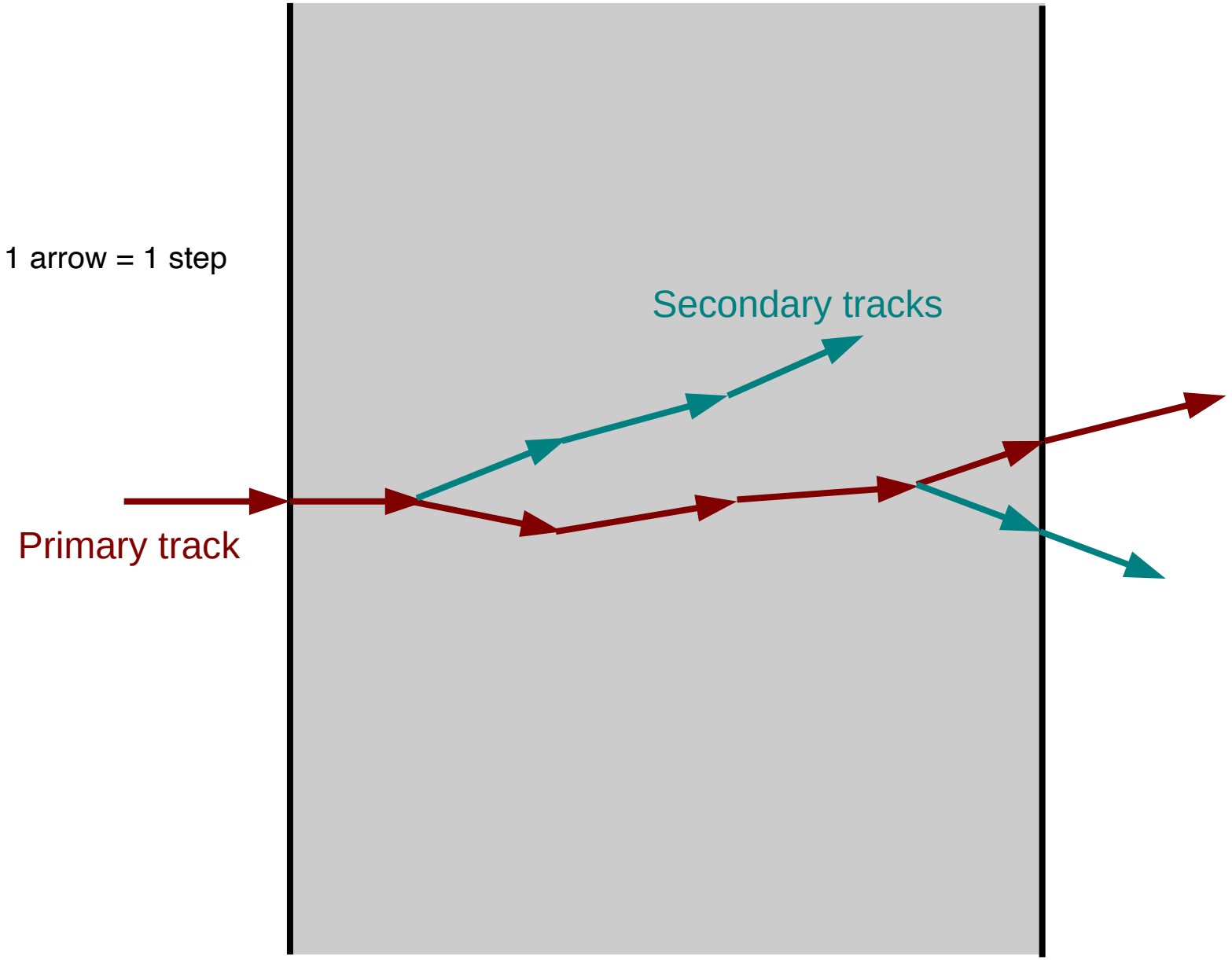
Geant4 Workflow overview

- **Your tasks:**
  - provide geometrical and material description of apparatus with a class
  - provide event generator done in the example, understand how to use it
  - choose an appropriate physics list and production cuts
- **Geant4 tasks:** more explanation later but basically I decide according to my problem to turn off some processes
  - propagate each particle in the materials
  - simulate interactions and produce secondary particles
  - compute energy release in the materials
  - iterate on the desired number of events

INFN

Terminology

- **Run:** a collection of events in the same detector conditions and settings of physics processes
- **Event:** each event starts with the generation of primary particles. Particles are added/removed from a stack when they are generated/disappear. When the stack is empty the event is finished.
- **Track:** is the snapshot of a particle. It is deleted if: it goes outside the world volume, it disappears (interaction), it comes to rest (0 kin. energy), the user decides to kill it
- **Step:** is the smallest unit of propagation of each particle. It has an initial and final point, time-of-flight, energy deposition, ...


1 arrow = 1 step
Secondary tracks
Primary track

User application

- Geant4 is a toolkit, the main() file to build the application must be provided by the user
- In the main file the user must:
  - construct the G4RunManager
  - notify the G4RunManager the existence of user-defined classes
  - initialize optional features, like the visualization manager

Overview of some user classes

- G4VUserDetectorConstruction: defines the detector setup (mandatory)
- G4VUserPhysicsList: defines the active physics processes (mandatory)
- G4VUserPrimaryGeneratorAction: defines the event generator (mandatory)
- G4VUserActionInitialization: to initialize user's actions
- G4VUserRunAction: to define actions to be performed at the beginning/and of the run
- G4VUserEventAction: to define actions to be performed at the beginning/and of the event
- G4VUserSteppingAction: to define actions to be performed at the beginning/and of the step

INFN

Simple example of main

Example from the Geant4 user's guide

```
#include "G4RunManager.hh"
#include "G4UImanager.hh"

#include "ExG4DetectorConstruction01.hh"
#include "ExG4PhysicsList00.hh"
#include "ExG4ActionInitialization01.hh"

int main()
{
  // construct the default run manager
  G4RunManager* runManager = new G4RunManager;

  // set mandatory initialization classes
  runManager->SetUserInitialization(new ExG4DetectorConstruction01);
  runManager->SetUserInitialization(new ExG4PhysicsList00);
  runManager->SetUserInitialization(new ExG4ActionInitialization01);

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

  // get the pointer to the UI manager and set verbosities
  G4UImanager* UI = G4UImanager::GetUIpointer();
  UI->ApplyCommand("/run/verbose 1");
  UI->ApplyCommand("/event/verbose 1");
  UI->ApplyCommand("/tracking/verbose 1");

  // start a run
  int numberOfEvent = 3;
  runManager->BeamOn(numberOfEvent);

  // job termination
  delete runManager;
  return 0;
}
```

Some examples of user action classes

- G4UserRunAction
  - Define parameters or define custom additional information
  - Book variables / histograms / analysis tools, ... to collect information for the run
- G4UserEventAction
  - Event selection and analysis
  - Hit collection
- G4UserSteppingAction
  - Control particle propagation
  - Collect specific information on interactions

INFN

1. Geometry and materials

System of units

- Units defined in: G4SystemOfUnits.hh
- Always include units when defining input parameters

```
G4double block_size = 10.0 *cm;
G4double pmma_density = 1.18*g/cm3;
```

- Retrieve output specifying the unit (otherwise internal units used)

```
G4cout << KineticEnergy/keV << " keV";
```

```
Geant4 internal system of units:

millimeter                  (mm)
nanosecond                  (ns)
Mega electron Volt          (MeV)
positron charge             (eplus)
degree Kelvin               (kelvin)
the amount of substance     (mole)
luminous intensity          (candela)
radian                      (radian)
steradian                   (steradian)
```

Materials

Materials are made of elements, which are made of isotopes. These concepts are represented in 3 classes in Geant4:

- G4Isotope: properties of atoms: atomic number, number of nucleons, mass per mole, etc
- G4Element: properties of elements: effective atomic number, effective number of nucleons, effective mass per mole, number of isotopes, shell energy, cross section per atom, ...
- G4Material: macroscopic properties of matter: density, state, temperature, pressure, radiation length, mean free path, dE/dx, ...

Geant4 contains tables of elements and common materials, derived from the NIST databases of elements and materials, that can be retrieved

```
G4NistManager* nist = G4NistManager::Instance();
G4Element* el_H  = nist->FindOrBuildElement("H");
G4Material* mat_air = nist->FindOrBuildMaterial("G4_AIR");
```

INFN

User defined materials

- Materials not included in the list can be defined by the user, specifying the composition and the density

```
// define a material from elements.   case 1: chemical molecule
density = 8.280*g/cm3;
G4Material* PbWO4= new G4Material(name="PbWO4", density, ncomponents=3);
PbWO4->AddElement(elO , natoms=4);
PbWO4->AddElement(elW , natoms=1);
PbWO4->AddElement(elPb, natoms=1);

// define a material from elements.   case 2: mixture by fractional mass
density = 1.290*mg/cm3;
G4Material* Air = new G4Material(name="Air  "  , density, ncomponents=2);
Air->AddElement(elN, fractionmass=0.7);
Air->AddElement(elO, fractionmass=0.3);
```

Detector construction

- Implement a class describing the detector geometry, inheriting from the abstract G4VUserDetectorConstruction

```
class B1DetectorConstruction : public G4VUserDetectorConstruction
{
  public:
    ...
    virtual G4VPhysicalVolume* Construct();
    ...
};
```

Contains the geometrical description

- The Construct method must be implemented
  - contains all the geometry (volumes and solids) and material information
  - returns the pointer to the “world” volume

Geometry structure

The detector geometry is defined using 3 different concepts, associated to different objects:

- Solid: information on shape and size of a detector component forma
- Logical Volume: information associated with detector elements represented by a given Solid and Material, independently from its physical position in the detector di cosa è fatto
- Physical Volume: represent the spatial positioning of the volumes describing the detector elements. Can be simple placement of a single copy or repeated positioning. dove è

All the volumes are contained in the "world" volume, that **must** be defined by the user

Basic strategy:

1. create a solid
2. associate properties to the solid through a logical volume
3. place the volume in the world

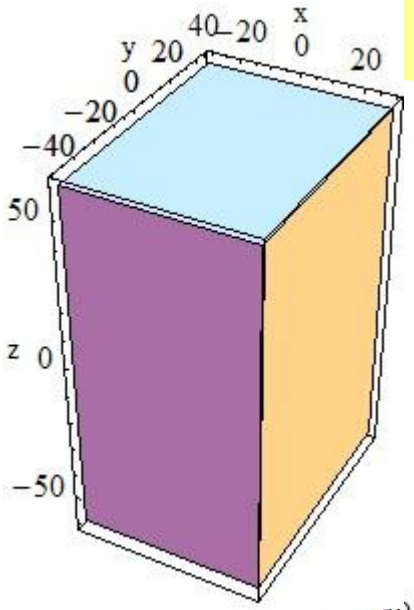
```
G4Tubs* pipe = new G4Tubs("Pipe", rint, rext, 0.5*len, 0*deg,
360*deg);

G4LogicalVolume* logicPipe = new G4LogicalVolume(pipe,
pipe_mat, "Pipe");

G4ThreeVector pospipe = G4ThreeVector(0*cm, 0*cm, 10*cm);
new G4PVPlacement(rotpipe, pospipe, logicPipe, "Pipe",
logicMotherVol, false, 0, checkOverlaps);
```

INFN

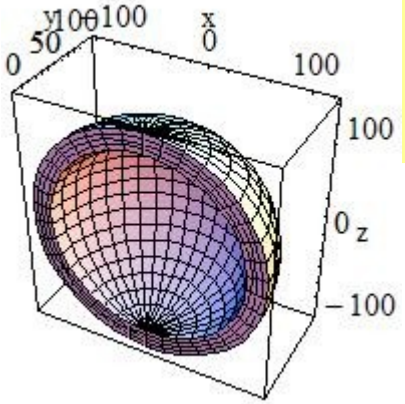
since we are building a samplig cal we will just need the box

Solids

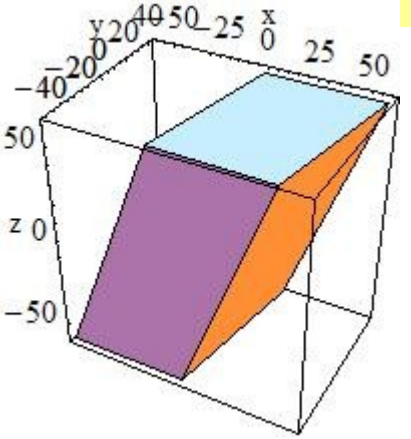
```
G4Box(const G4String& pName,
      G4double  pX,
      G4double  pY,
      G4double  pZ)
```

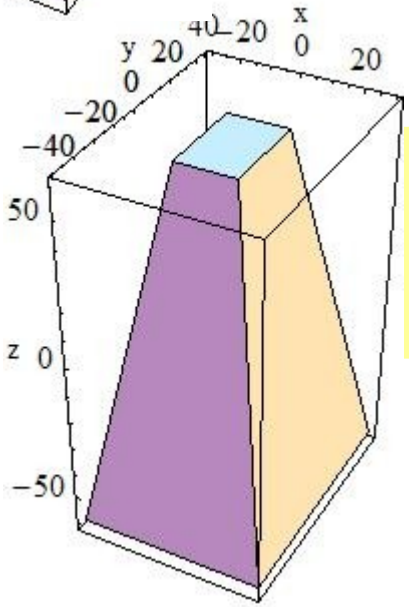


```
G4Sphere(const G4String& pName,
         G4double   pRmin,
         G4double   pRmax,
         G4double   pSPhi,
         G4double   pDPhi,
         G4double   pSTheta,
         G4double   pDTheta)
```

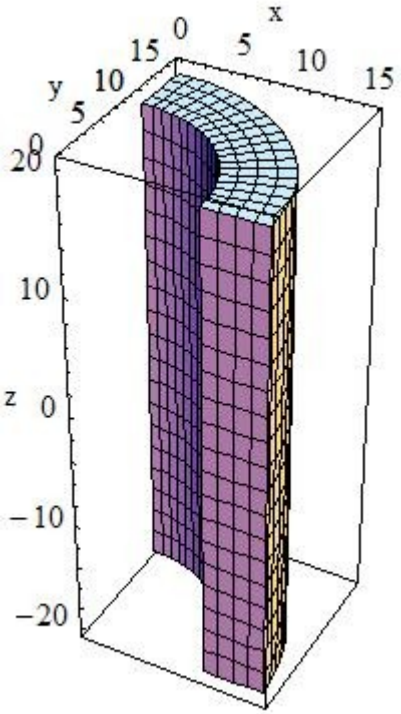


```
G4Para(const G4String& pName,
       G4double   dx,
       G4double   dy,
       G4double   dz,
       G4double   alpha,
       G4double   theta,
       G4double   phi)
```





```
G4Trd(const G4String& pName,
      G4double  dx1,
      G4double  dx2,
      G4double  dy1,
      G4double  dy2,
      G4double  dz)
```



```
G4Tubs(const G4String& pName,
       G4double  pRMin,
       G4double  pRMax,
       G4double  pDz,
       G4double  pSPhi,
       G4double  pDPhi)
```

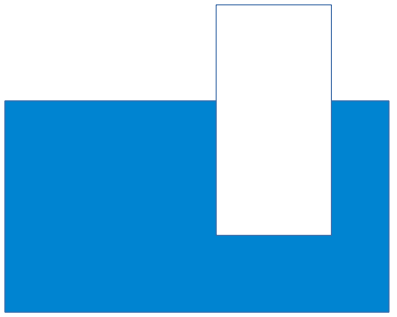
Full list at: https://geant4.web.cern.ch/geant4/UserDocumentation/UsersGuides/ForApplicationDeveloper/html/ch04.html

Boolean solids

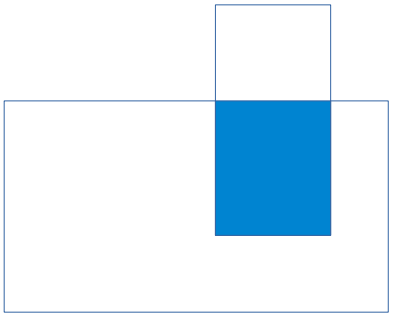
- Simple solids can be combined in using boolean operations, requiring:
  - two solids
  - an operation: union, subtraction, intersection
  - an optional transformation for the second solid
- The result of the operation is a solid
- Examples: hole on a sphere, cylinder ending with a cone

G4UnionSolid

G4SubtractionSolid



G4IntersectionSolid



Logical Volume

- The Logical Volume manages the information associated with detector elements represented by a given Solid and Material, independently from its physical position in the detector.
- A Logical Volume knows which physical volumes are contained within it. A Logical Volume thus represents a hierarchy of unpositioned volumes whose positions relative to one another are well defined.
- By creating Physical Volumes, which are placed instances of a Logical Volume, this hierarchy or tree can be repeated.
- A Logical Volume also manages the information relative to the Visualization attributes and user-defined parameters related to tracking, electro-magnetic field or cuts.

INFN

Physical Volume

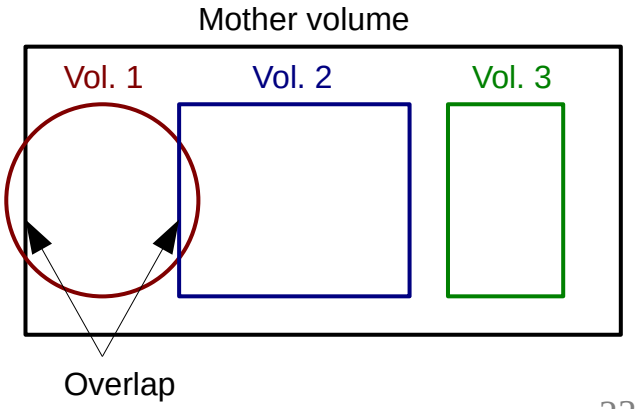
- Physical volumes represent the spatial positioning of the volumes describing the detector elements.
- G4PVPlacement: The simple placement involves the definition of a transformation matrix for the volume to be positioned.

placement basic method, but we will use the replica

- G4PVReplica: Repeated positioning is defined using the number of times a volume should be replicated at a given distance along a given direction.
  - G4PVParameterised for more complex positioning

Overlaps are not allowed:

- volumes intersecting each other
- volumes extending beyond the boundaries of their mother volumes



2. Physics model

Physics processes

- Physics processes define how particles can interact with the materials

  Turning on all processes at the same time is a waste of cpu time

- The process:
  - defines if a given interaction occurs during the step, based on the cross-section
  - changes momentum of the particle
  - generates secondary particles
- Examples: ionization, photoelectric effect, positron annihilation, hadronic interaction, ...
- All details in the Geant4 physics reference manual:

  geant4.web.cern.ch/geant4/UserDocumentation/UsersGuides/PhysicsReferenceManual/fo/PhysicsReferenceManual.pdf

INFN

Physics lists

- Geant4 provides a flexible framework that allow the implementation of any user-defined model to describe physics processes

I need to tell geant which phys list to use

- The definition of a physics list is mandatory

```
class MyPhysicsList: public G4VUserPhysicsList
{
public:
MyPhysicsList();
~MyPhysicsList();
void ConstructParticle();
void ConstructProcess();
void SetCuts();
}
```

I will not need to build it, already done in initialization

- Modular physics lists can be constructed from basic blocks
- Reference physics lists are complete lists with realistic physics models that can be chosen according to the use case

Reference physics lists

- Reference physics lists are listed at http://geant4.cern.ch/support/proc_mod_catalog/physics_lists/referencePL.shtml
- How to initialize a reference physics list:

```
#include "QGSP_BIC_HP.hh"

int main(int argc,char** argv)
{
  ...
  G4RunManager* runManager = new G4RunManager;

  // Physics list
  G4VModularPhysicsList* physicsList = new QGSP_BIC_HP;
  runManager->SetUserInitialization(physicsList);

  ...
  runManager->Initialize();
  ...
}
```

This is a phy list that can be used also for hadronic int
see slide 34 for more info on the header

Production cuts

- Particles are produced above defined thresholds
- Cuts are only at the production stage, no tracking cut is applied
  - Cuts are used to limit the generation of secondary particles
  - After production a particle is tracked until it stops or disappears
  - Cuts are expressed in range, corresponding to a different energy for each material

spacial range

- It is possible to apply cuts to specific volumes

Electromagnetic physics - basic concepts

- The same physics interaction or process can be described by different models
  - they may be different / complementary / suggested in some energy ranges
- Example: available Compton scattering models inGeant4
  - Klein-Nishina
  - Livermore (for low E, based on Livermore database)
  - Penelope (for low E, based on Penelope model)
  - LowEP (Monash Univ. model, 3D fully relativistic treatment)
  - Polarized (add polarization to Klein-Nishina or Livermore)
- EM physics list constructors: http://geant4.cern.ch/collaboration/working_groups/electromagnetic/physlist10.3.shtml

INFN

Standard EM models

- Complete set of models for: γ, e, μ, hadrons, ions
- Designed for high-energy physics applications
  - Less CPU intensive
  - Approximations in the low-energy regime
- Theoretical or phenomemological models (Bethe-Bloch, Klein-Nishina, ...)
- Specific high-energy extensions available (e.g. $\gamma \rightarrow \mu^+\mu^-$)
- Dedicated library for optical photons (Cherenkov)

INFN

Livermore EM models

- Based on publicly available data tables from the Livermore data library for e^-, γ
- Tables go down to ~10 eV
- Applications: low-energy EM, medical, underground, rare events, space, ...
- Possible to use polarized models
  - same cross-section calculation
  - describe in detail the interaction of polarized photons

Penelope EM models

- Based on the low-energy models for e^-, e^+, γ from the PENELOPE Monte Carlo code
  - NIM B 2017 (2003) 107
- Mixed approach: analytical, parameterized, database-driven
- Energy range: 100 eV - 1 GeV
- Applications: similar to Livermore but includes positrons

Low-energy EM models should be adopted for EM physics at the ~keV scale.

Not to be used for EM physics at the >MeV scale: same results as Standard with lower performance

Hadronic physics - basic concepts

- More complex than EM physics
- Involves several process, in different energy regimes
  - < 100 MeV: low-energy
  - 100 MeV - 10 GeV: resonance and cascade region
  - >20 GeV: QCD strings
  - thermal to 20 MeV neutrons
- Several models available in each energy regime, many of them are phenomenological
- Hadronic processes: elastic, inelastic, capture, fission

user will still in any case just shoot a particle

Reference physics lists for hadronic interactions

- String models:
  - QGSP: uses the quark-gluon string model for high-energy hadronic interactions between, p, n, π, K, nuclei. Low Energy Parameterised (LEP) model applied to all particles.
  - FTFP: uses the FRITIOF model of string excitation and fragmentation
- Alternative cascade models:
  - BERT: Geant4 Bertini cascade model for primary p, n, π and K below ~10 GeV, instead of the LEP model. More secondary neutrons and protons produces than the LEP model, yielding a better agreement to experimental data.
  - BIC: Geant4 Binary cascade for primary protons and neutrons with energies below ~10GeV, instead of the LEP model. Binary cascade better describes production of secondary particles produced in interactions of protons and neutrons with nuclei.

In slide 27 ho un QGSP_BIC_HP
HP means High energy Pregigion on low en neutrons (prossima slide)

Neutron HP model

- Dedicated High-Precision (HP) model available for very-low energy neutrons
- Cross-sections and final state information based on tabulated data (G4LEND)
  - elastic, inelastic scattering, capture, fission, isotope production
- Applied from thermal energies up to 20 MeV
  - Very precise but also slow (thermal neutron tracking is CPU intensive)

Particle HP models

- Data driven approach extended to inelastic interactions of p, d, t, ^{3}He, α (G4TENDL)
- Recently added to Geant4 (since v 10.2)
- Same energy range of applicability of neutron HP models
- The new package include neutrons
  - `NeutronHP` → `ParticleHP`

3. Primary event generator

G4VUserPrimaryGeneratorAction

- Mandatory user class, controls the generation of primary particles
  - Constructor:
    - Instantiate primary generator (i.e. G4ParticleGun())
    - Define default values (i.e. energy)

generator that shoots one particle

```
G4ParticleGun* particleGun = new G4ParticleGun(n_particle);
particleGun -> SetParticleEnergy(1.0*GeV);
```

  - GeneratePrimaries():
    - User defined modifications of generator parameters (e.g. energy, vertex position)
    - Ends invoking GeneratePrimaryVertex() of the generator

```
particleGun->GeneratePrimaryVertex(anEvent);
```

G4ParticleGun

- Provided implementation of the primary generator designed to
  - Shoot a single particle
  - From a given point
  - In a given direction
- Use the different Set methods available

```
G4ParticleDefinition* particle = particleTable->FindParticle("gamma");
particleGun->SetParticleDefinition(particle);
particleGun->SetParticleEnergy(100*keV);
particleGun->SetParticlePosition(G4ThreeVector(x0,y0,z0));
particleGun->SetParticleMomentumDirection(G4ThreeVector(xdir,ydir,zdir));

...

particleGun->GeneratePrimaryVertex(anEvent);
```

INFN

4. Tracking

Particle production and propagation

- Primary particles are produced by the event generator, secondary particles by processes
  - G4ParticleDefinition: particle definition properties (mass, charge, PDGID, name, ...)
  - G4DynamicParticle: dynamic properties (momentum, polarization, ...)
- G4Track contains the information for tracking the particle inside the detector geometry (position, step, current volume, ...)
- The information is updated after each step
- The track is deleted when the particle disappears in an interaction, has 0 kinetic energy, goes outside the world volume, or is manually killed by the user

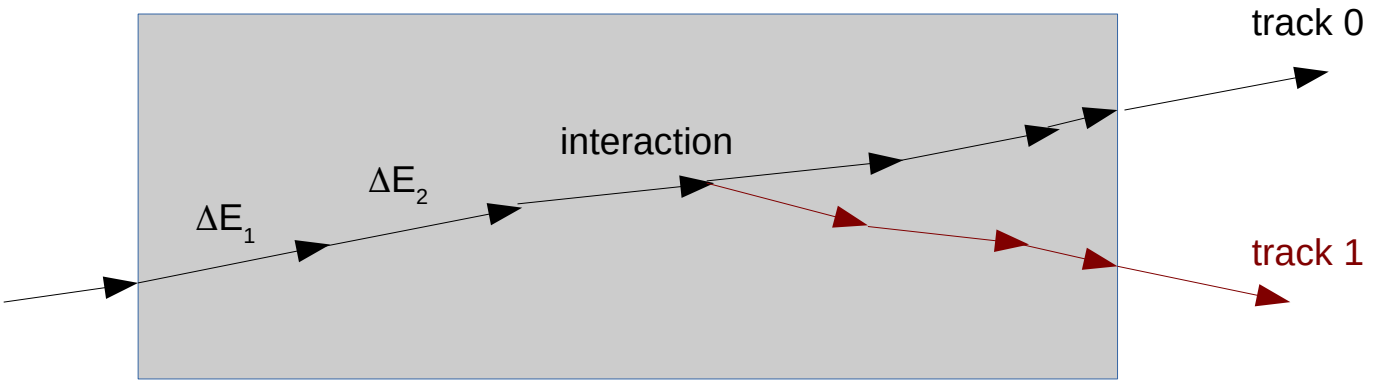
INFN

Step

- G4Step represents the smaller unit of particle propagation
- The step contains information about the initial and final points, energy depositions in the materials, ...
- If a boundary between volumes is crossed, the step length is limited to end at the boundary
- The user can define limits to the step length, that can assume different values for different volumes

Particle propagation procedure

- A particle is shot
- The geometrical step length is computed, based on the cross-sections of all processes and choosing the smallest one
  - Check if crossing boundaries
- All processes involved are executed
- Post-step procedures are executed (track update, secondaries added to the stack, ...)



Production cuts

- It is unpractical to track all secondary particles of “very low” energy
  - The simulation should balance between accuracy and execution time (performance)
- The user can define limits on particles that are produced
  - limits can be general or specific for given regions
  - Geant4 allows to define a range limit (the resulting energy cut depends on the particle and the material)
  - particles that cannot travel for at least the cut value are not produced
- No tracking cuts exist in Geant4: once a particle is added to the stack it is tracked down to 0 energy (or until it disappears)

INFN

5. Hits

G4Hit

- The definition of a Hit is provided by the user, so various type of information can be stored in a Hit
- Typically a Hit contain the information needed to describe the energy deposition in the detector
  - position, time, energy deposition
  - can also contain additional information associated to the G4Step, such as: particle type and momentum, volume, ...
- Hits are then created and filled by the SensitiveDetector
- Hits are stored in the HitCollection attached to the event and can be read back at the end of the event

Note: an example of the Hit/SD implementation can be found in the example B5

INFN

G4Hit

```
class MyCalorimeterHit : public G4VHit
{
  public:
      MyCalorimeterHit();
      ~MyCalorimeterHit();
      ...

  private:
      G4double edep;
      G4ThreeVector pos;
      ...

  public:
      inline void SetEdep(G4double de)
      { edep = de; }
      inline G4double GetEdep() const
      { return edep; }
      inline void SetPos(G4ThreeVector xyz)
      { pos = xyz; }
      inline G4ThreeVector GetPos() const
      { return pos; }
      ...
};

using MyCalorimeterHitsCollection =
      G4THitsCollection<MyCalorimeterHit>;
```

Since a simulation may contain different detectors (e.g. calorimeter and Si tracker), it is possible to create several G4Hit classes storing different information

Sensitive Detector and hit collection

- The definition of the Sensitive Detector (SD) is a powerful and flexible way of extracting information from the simulation
  - Derive concrete classes from G4VSensitiveDetector and G4VHit
- The method ProcessHit() of the SD is invoked at each step starting in the SD logical volume. It must create, fill and store the Hit objects
  - Initialize() and EndOfEvent() methods are invoked at the beginning and at the end of the event
- The Hits are stored in a HitCollection (an array of hits)
  - the G4Event has an associated G4HCofThisEvent object, storing all hit collections created during the event
  - All the hits are accessible in the EndOfEventAction() method of the EventAction
  - retrieved using G4Event::GetHCofThisEvent()

INFN

G4SensitiveDetector

```
class MyCalorimeterSD : public G4VSensitiveDetector
{
  public:
    MyCalorimeterSD(G4String name);
    virtual ~MyCalorimeterSD();

    virtual void Initialize(G4HCofThisEvent* HCE);
    virtual G4bool ProcessHits(G4Step* aStep, G4TouchableHistory*ROhist);
    virtual void EndOfEvents(G4HCofThisEvent* HCE);

  private:
    MyCalorimeterHitsCollection* fHitsCollection;
    G4int fHCID;
};
```

G4SensitiveDetector - Initialize()

```
void MyCalorimeterSD::Initialize(G4HCofThisEvent* hce)
{
  fHitsCollection
    = new MyCalorimeterHitsCollection(SensitiveDetectorName,collectionName);
  fHCID = G4SDManager::GetSDMpointer()->GetCollectionID(fHitsCollection);

  hce->AddHitsCollection(fHCID,fHitsCollection);

  fHitsCollection->insert(new MyCalorimeterHit());

}
```

- Constructs all the HitCollections
- Insert them into the G4HCofThisEvent

G4SensitiveDetector - ProcessHits()

```
G4bool MyCalorimeterSD::ProcessHits(G4Step* step, G4TouchableHistory*)
{

  MyCalorimeterHit hit;

  G4double edep = step->GetTotalEnergyDeposit();
  ... // All variables of interest

  hit.SetEnergyDeposit(edep);
  ... // set all hit member variables

  hitCollection.insert(hit);

  return true;
}
```

- Invoked at every step
- Generates and stores hits

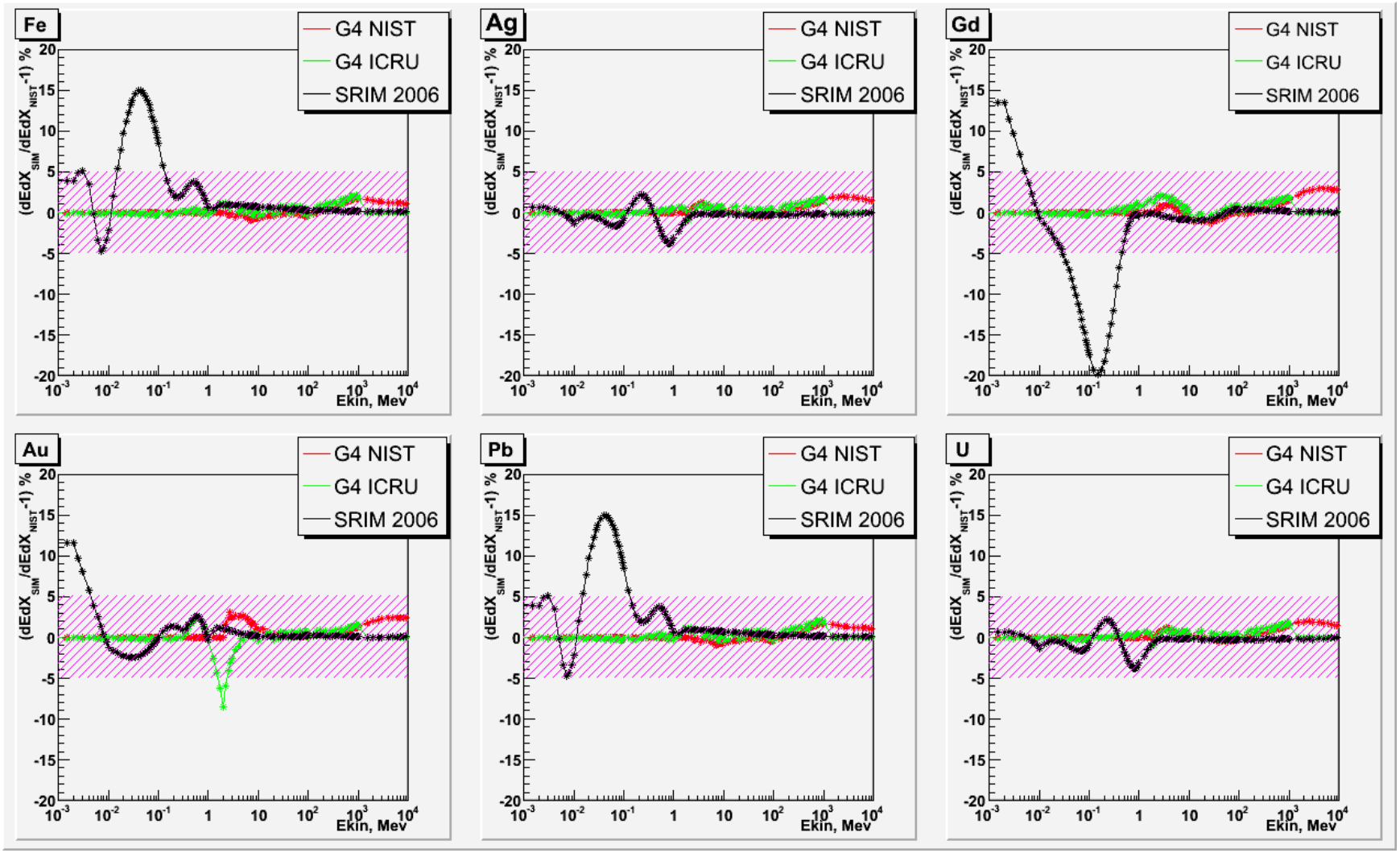
6. Performance and Validation

Geant4 performance and validation

- Performance (CPU and memory) and validation tests regularly performed by the Geant4 collaboration
- New releases tested against
  - experimental data
  - previous releases
- Collected in http://geant4.cern.ch/results/results.shtml
- Some examples in the following

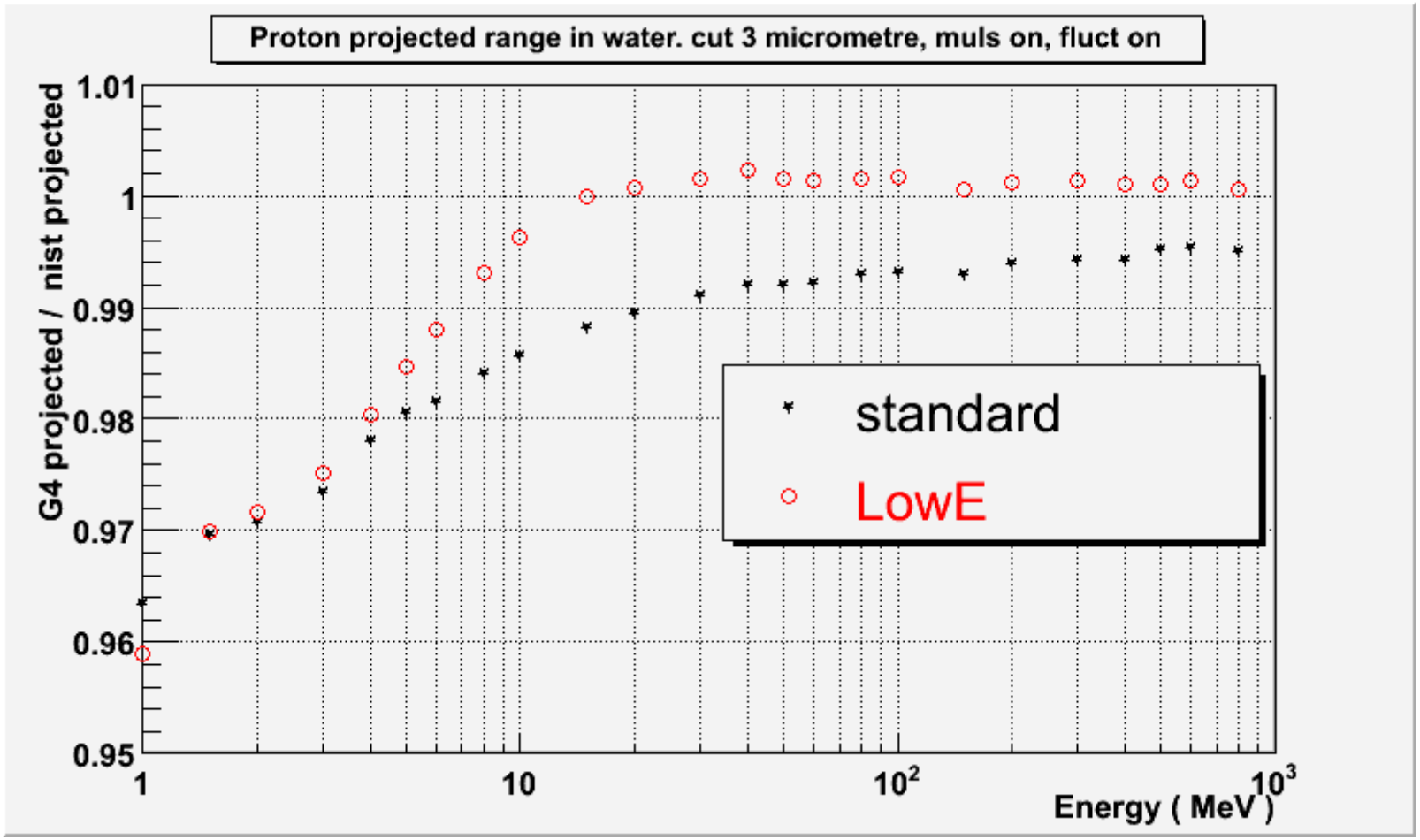
EM physics validation - 1

Proton stopping power compared with NIST data - Data accuracy ~2% for E>1MeV



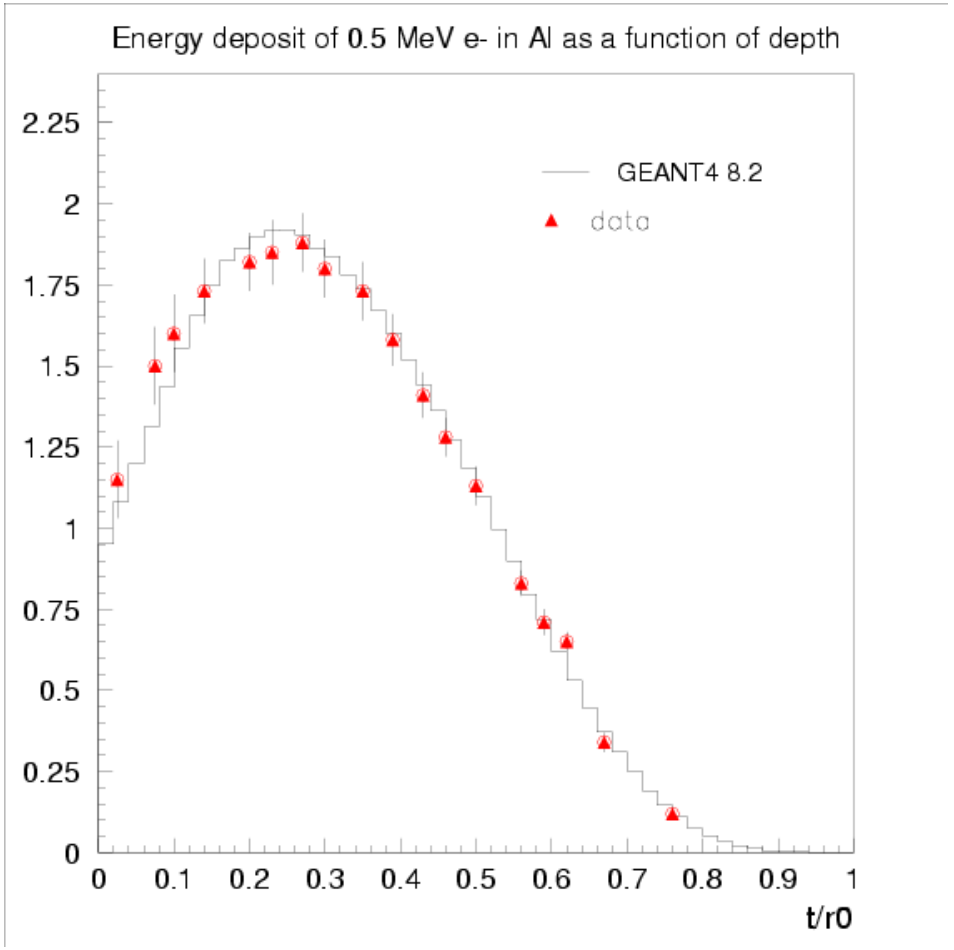
EM physics validation - 2

Proton range compared with NIST data - Data accuracy ~2% for E>1MeV



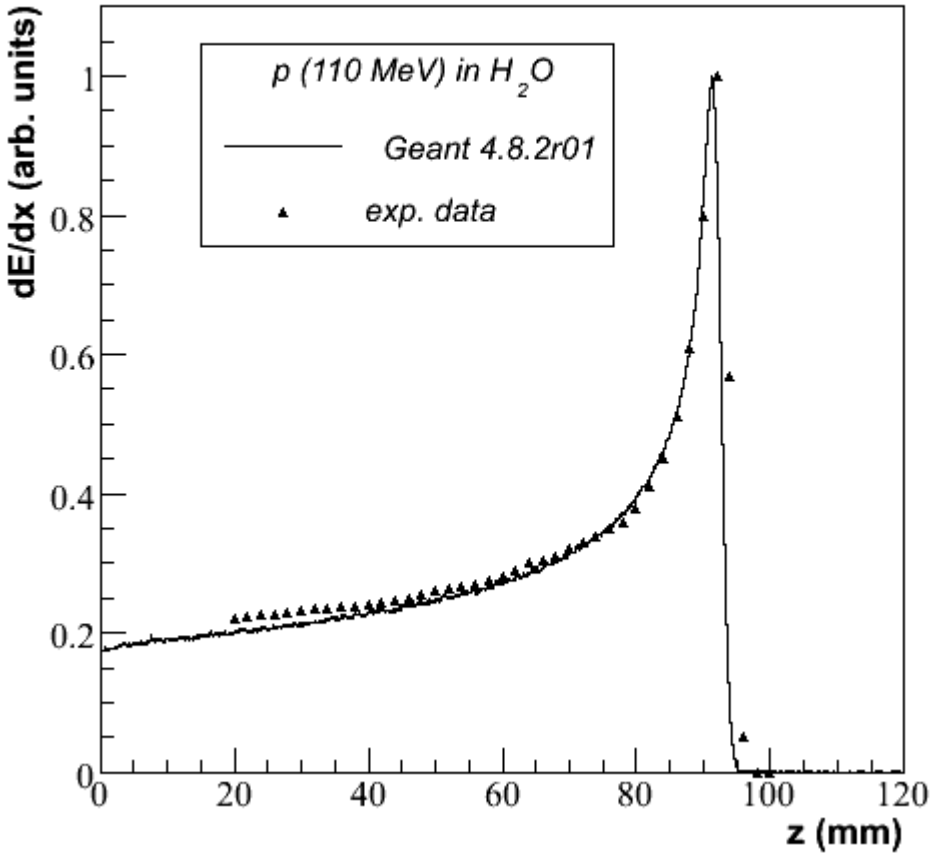
EM physics validation - 3

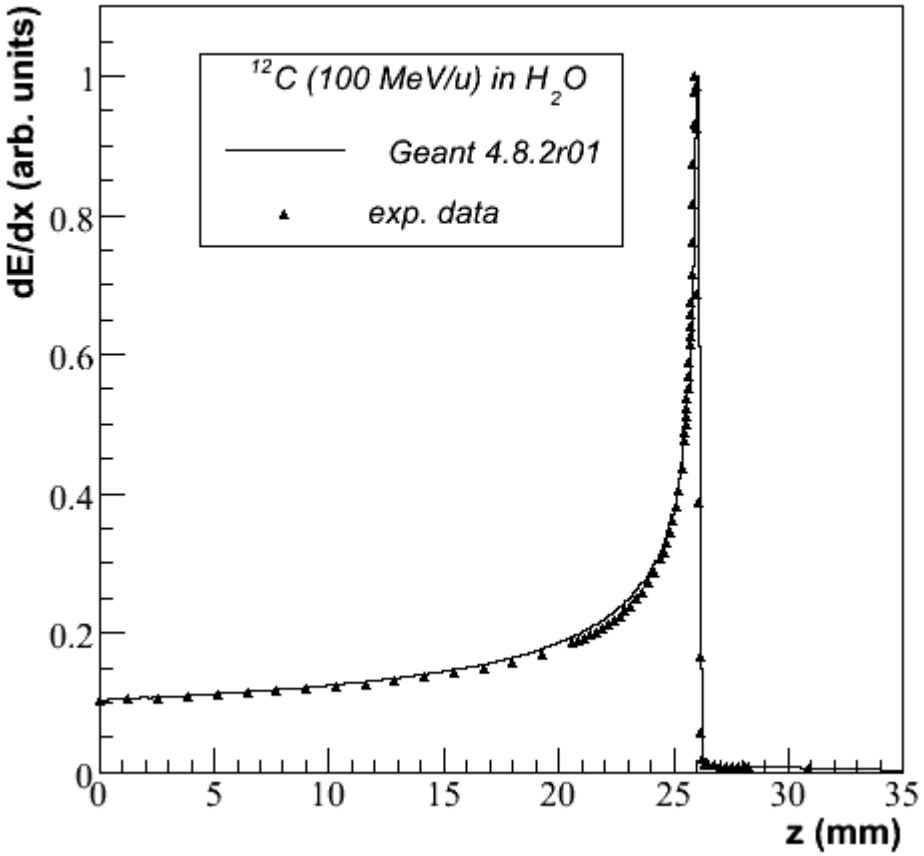
Energy deposition profile of 0.5 MeV e^- in Al



EM physics validation - 4

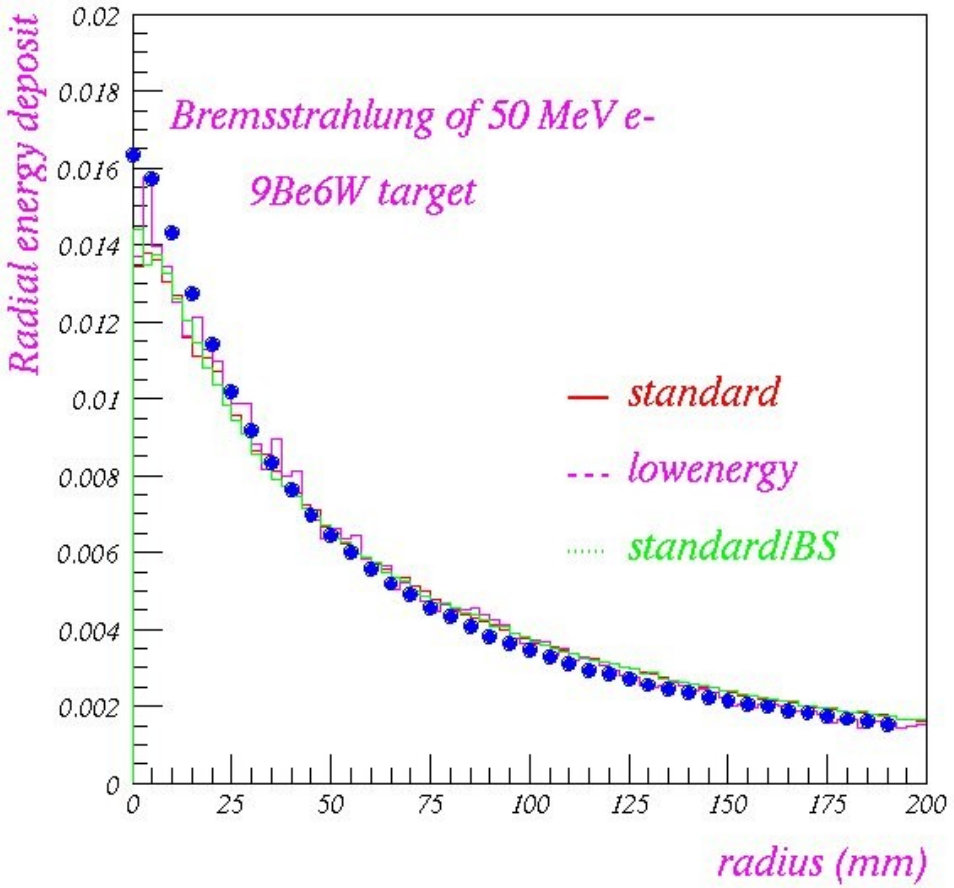
Bragg peaks of p and ^{12}C in water

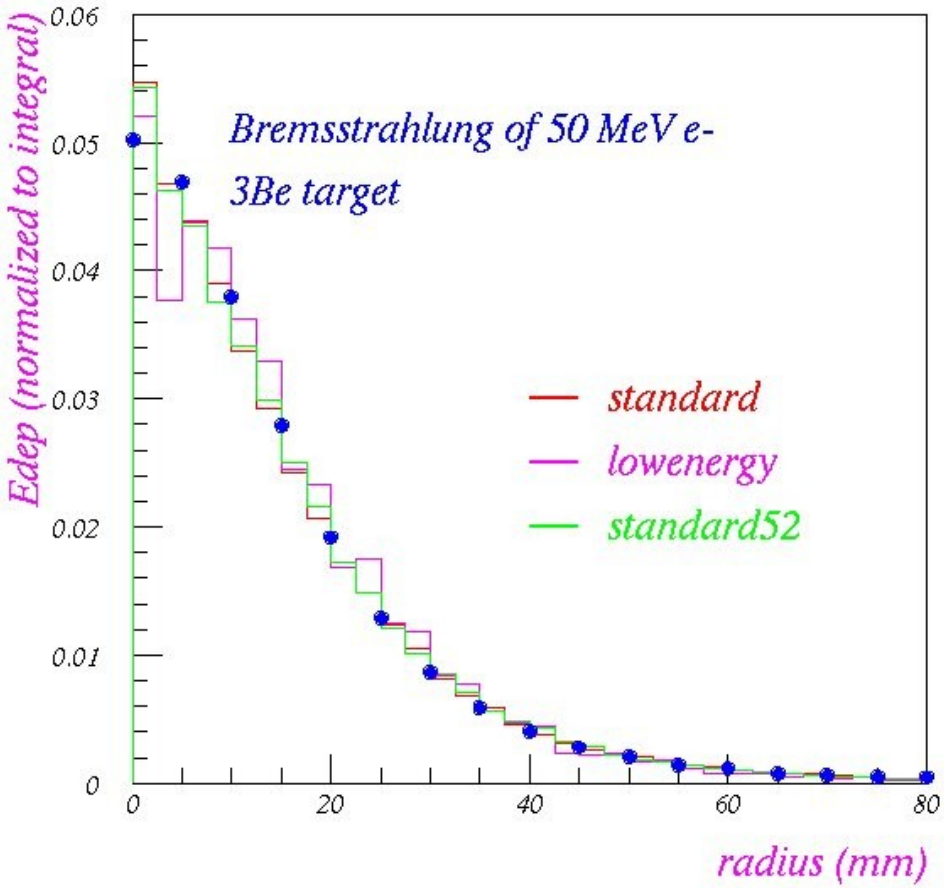




EM physics validation - 5

Radial energy deposition for bremsstrahlung photons by 50 MeV e^- in different targets

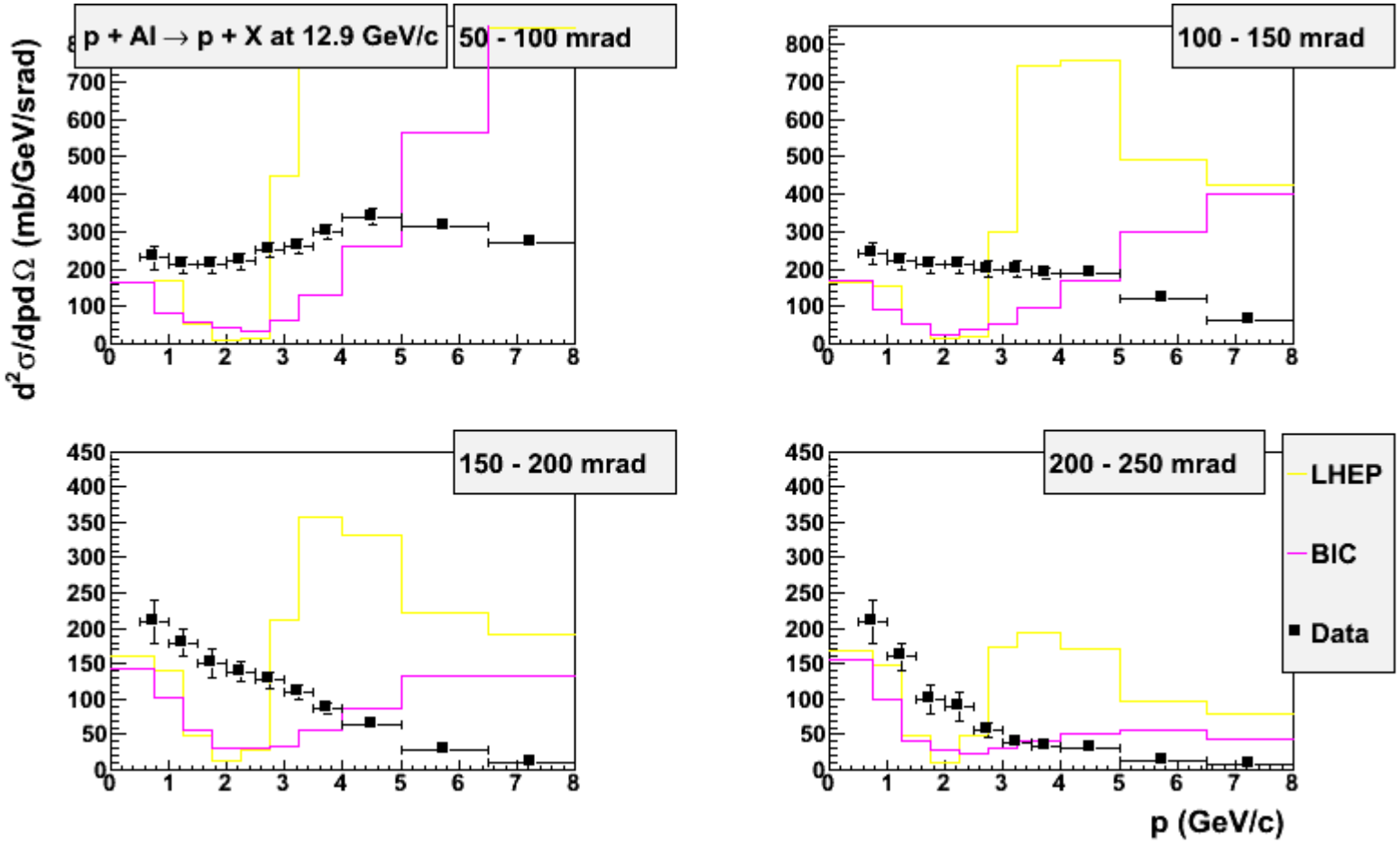




Hadronic physics validation - 1

Hadronic generation of inelastic processes compared with the HARP experiment

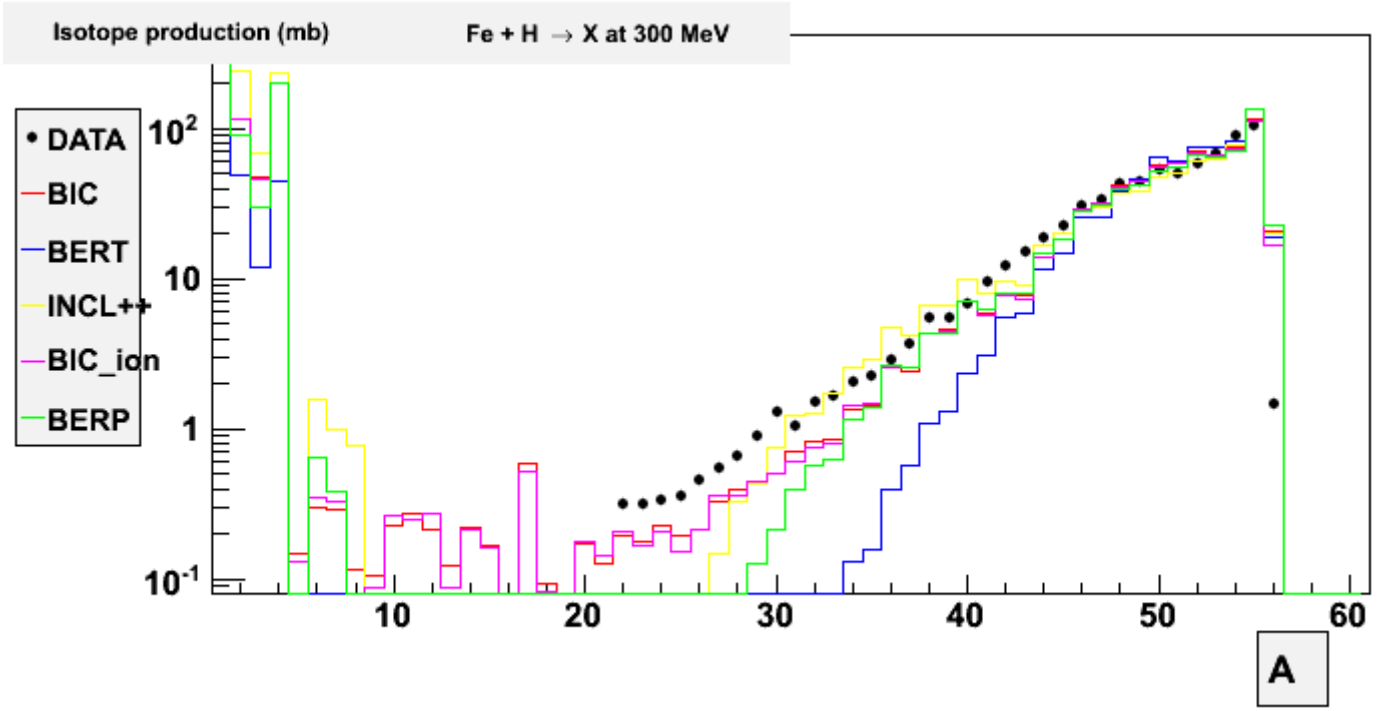
p + Al -> p + X, 12.9 GeV/c



Hadronic physics validation - 2

IAEA benchmark of nuclear spallation models

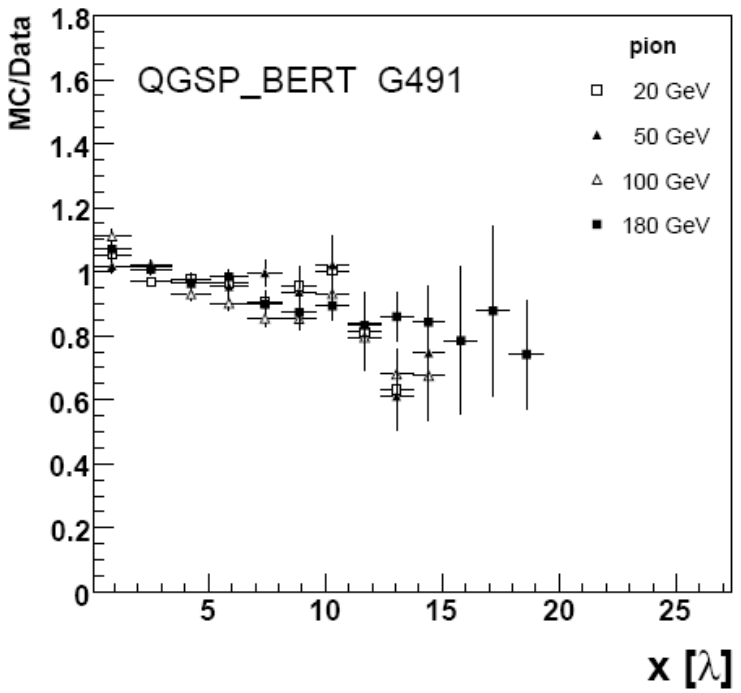
p + Fe -> n + X, 300 MeV



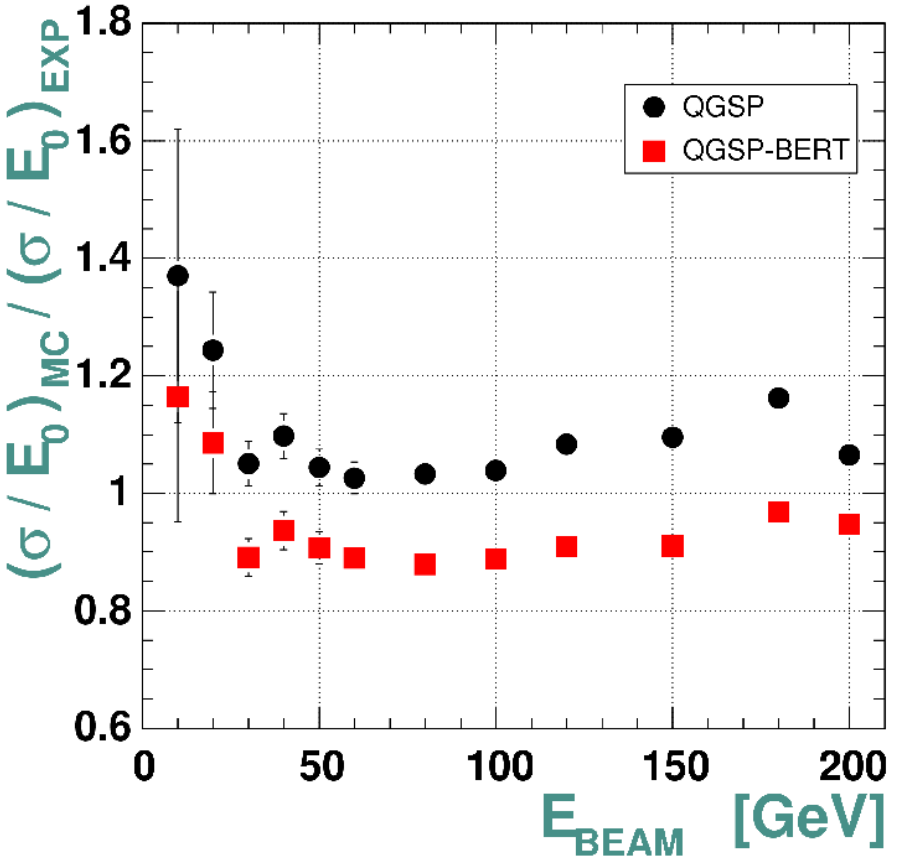
Hadronic physics validation - 3

Shower characteristics of ATLAS calorimeter

5.0 GeV π^- on ATLAS Tile cal.
Shower longitudinal profile

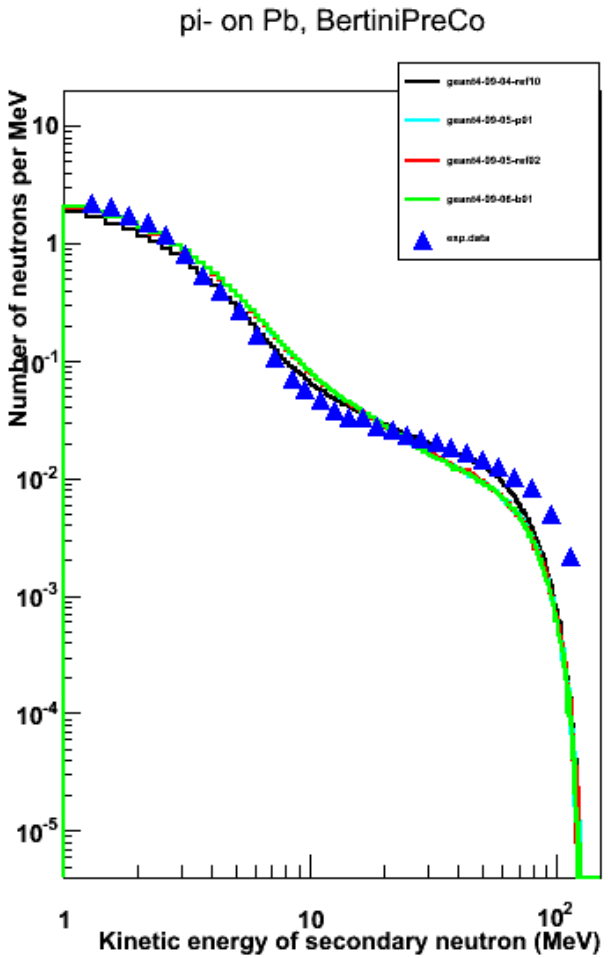


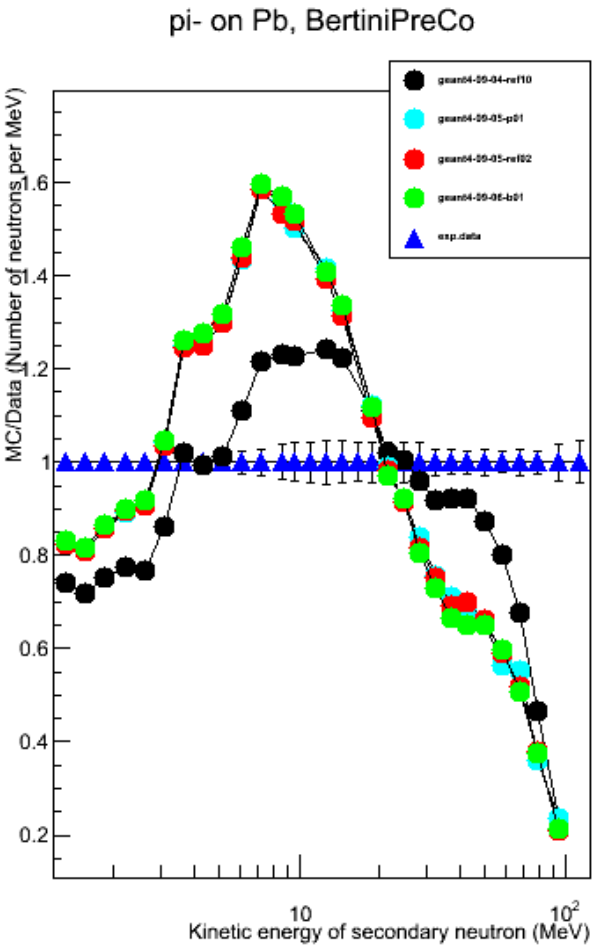
5.0 GeV π^- on ATLAS Forw. cal.
Energy resolution



Hadronic physics validation - 4

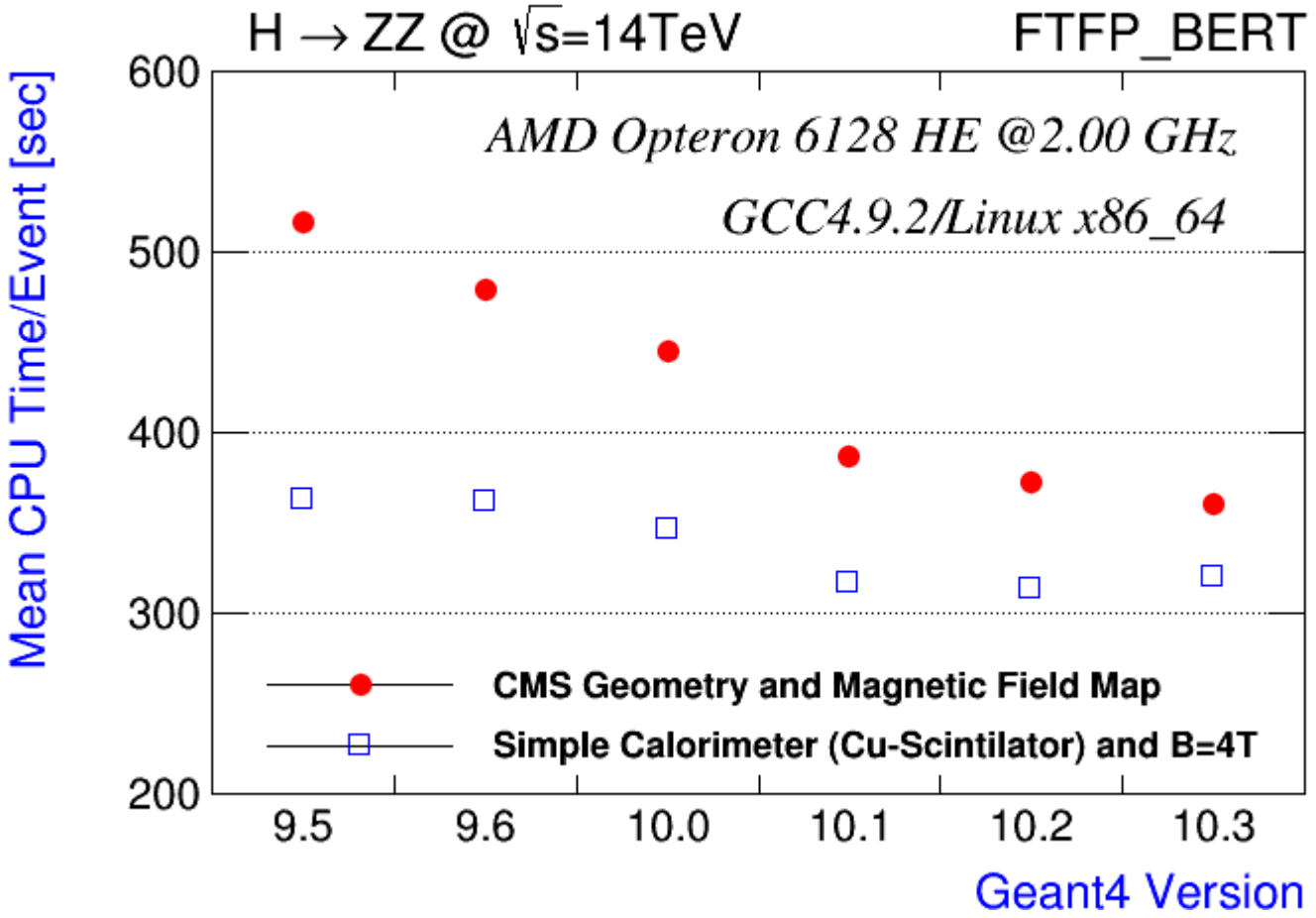
Kinetic energy of secondary neutrons for 5.0 GeV π^- on Pb





Geant4 release performance - 1

Time to simulate a H→ZZ event using a simplified or the CMS calorimeter



Geant4 release performance - 2

Memory to simulate 51 H→ZZ events using a simplified calorimeter
(1 mem. count = 1 byte)