European Scientific Institute

European School of Instrumentation in Particle & Astroparticle Physics

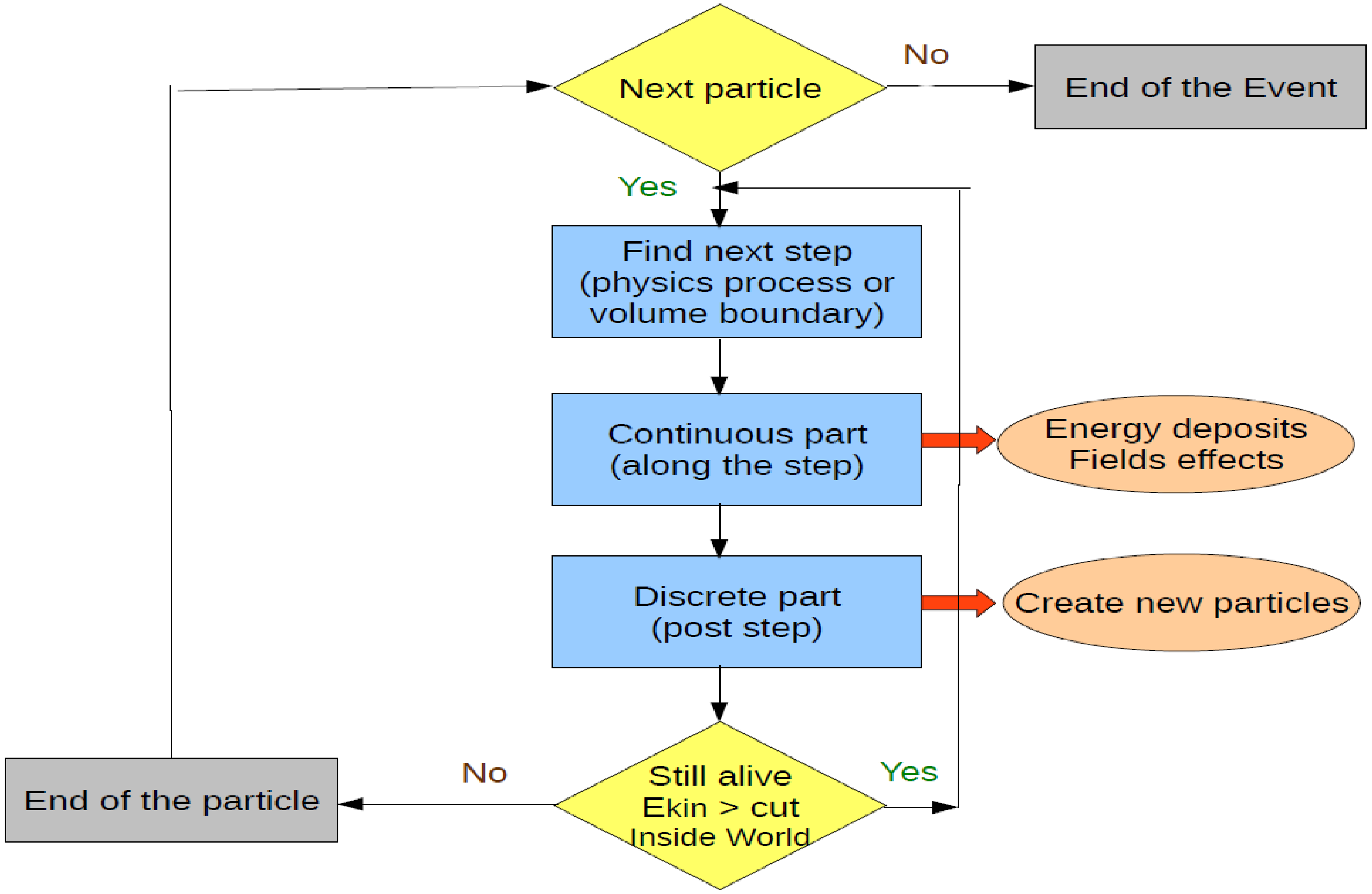
Detector Simulation

Witold Pokorski, Alberto Ribon
CERN PH/SFT

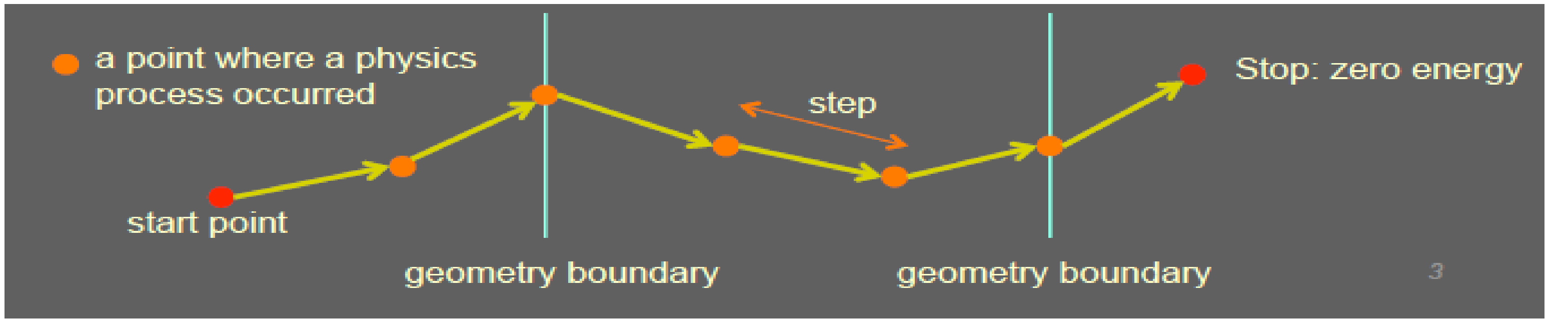
ESIPAP, Archamps, 10-11 February 2014

Where to Find Information

- Geant4 User Guides
  - **geant4.web.cern.ch/geant4/support/userdocuments.shtm**
  - User's Guide: For Application Developers
  - Physics Reference Manual
  - ...
- User Support
  - **geant4.web.cern.ch/geant4/support/index.shtml**
  - Bug reports and fixes
  - ...
- HyperNews Forum
  - **hypernews.slac.stanford.edu/HyperNews/geant4/cindex**
  - Discussion between users and developers



Next particle
No
End of the Event
Yes
Find next step (physics process or volume boundary)
Continuous part (along the step)
Energy deposits Fields effects
Discrete part (post step)
Create new particles
Still alive Ekin > cut Inside World
No
Yes
End of the particle



a point where a physics process occurred
start point
step
Stop: zero energy
geometry boundary
geometry boundary
3

Classical vs Quantum approach

- In Geant4, a particle that flies through a detector is treated as a **classical particle**, i.e. **not** a **wave function**, but a point-like object which has a well-defined momentum at each instant:
  - Space-time position **(x, y, z, t)**
  - Energy-momentum **(p_x, p_y, p_z, E)**

  This is a reasonable approximation, given that in most practical situations particles are seen as “**tracks**” in macroscopic detectors

- Geant4 is based on a **semi-classical** approach, because the particles are treated classically, but their interactions - cross sections and final states - take often into account the results of quantum-mechanical effects

Run, Event,
Particle, Track,
Step, StepPoint
Trajectory, TrajectoryPoint

Run in Geant4

- A **run** is a collection of events
  - Consists of one event loop
  - Starts with the ***/run/beamOn*** command

- Within a run, conditions do not change, i.e. the user cannot change:
  - the detector setup
  - the settings of physics processes

- A run in Geant4 is represented by the class **G4Run** or a user-defined class derived from it
- **G4RunManager** is the manager class
- **G4UserRunAction** is the optional user hook

Event in Geant4

- An **event** is the basic unit of simulation in Geant4
- At beginning of processing, primary tracks are generated; these are pushed into a stack
- A track is popped up from the stack one by one and “tracked”; resulting secondary tracks are pushed into the stack
  - this “tracking” lasts as long as the stack has a track
- When the stack becomes empty, processing the event is over
- The class **G4Event** represents an event; it has the following objects at the end of its (successful) processing:
  - List of primary vertices and particles (as input)
  - Hits and Trajectory collections (as output)
- **G4EventManager** is the manager class
- **G4UserEventAction** is the optional user hook

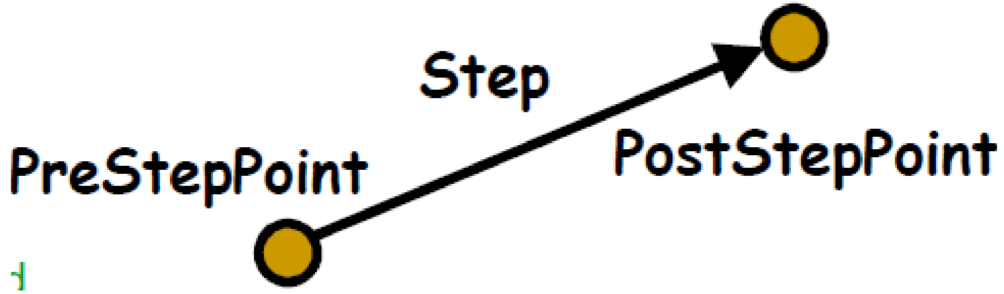
Particle in Geant4

- A **particle** in Geant4 is represented by three layers of classes
- **G4Track**
  - Position, geometrical information, etc.
  - This is a class representing a particle to be tracked
- **G4DynamicParticle**
  - "Dynamic" physical properties of a particle: momentum, energy, spin...
  - Each G4Track object has its own G4DynamicParticle object
  - This is a class representing an individual particle
- **G4ParticleDefinition**
  - "Static" properties of a particle: charge, mass, lifetime, etc.
  - **G4ProcessManager** describes the processes involving this particle
  - All G4DynamicParticle objects of the same kind of particle share the same G4ParticleDefinition

Track in Geant4

- A **track** is a snapshot of a particle
  - It has the physical quantities corresponding only to the current instance; it does not record previous quantities
  - Step is a “delta” information to a track; a track is not a collection of steps; instead, a track is updated by steps
- A track object is deleted when
  - it goes out of the world volume
  - it disappears (e.g. decay, inelastic scattering)
  - it goes down to zero kinetic energy and no “AtRest” additional process is required
  - the user decides to kill it artificially
- No track object persists at the end of event
  - For recording tracks, use trajectory class objects
- **G4Track** class represents an event
- **G4TrackingManager** is the manager class
- **G4UserTrackingAction** is the optional user hook

Step in Geant4



- A step has two points and also "delta" information of a particle (energy loss on the step, time-of-flight spent by the step, etc.)
  - A point is represented by the **G4StepPoint** class
- Each point knows the volume (and material). In case a step is limited by a volume boundary, the end point physically stands on the boundary, and it logically belongs to the next volume
  - Because one step knows the materials of two volumes, boundary processes such as transition radiation or refraction can be simulated
- **G4Step** represents a step
- **G4SteppingManager** is the manager class
- **G4UserSteppingAction** is the optional user hook

Trajectory and Trajectory Point in Geant4

- Track does not keep its trace.
  No track object persists at the end of an event
- **G4Trajectory** is the class which copies some of the **G4Track** information and persist till the end of an event
- **G4TrajectoryPoint** is the class which copies some of the **G4Step** information and persist till the end of an event
  - G4Trajectory has a vector of G4TrajectoryPoint objects
  - With the command: ***/tracking/storeTrajectory 1***
    at the end of event processing, G4Event has a collection of G4Trajectory objects
- Be careful not to store too many trajectories: memory growth
- G4Trajectory and G4TrajectoryPoint as provided by Geant4 store only the minimum information
  - users can create their own trajectory / trajectory point to store information they need

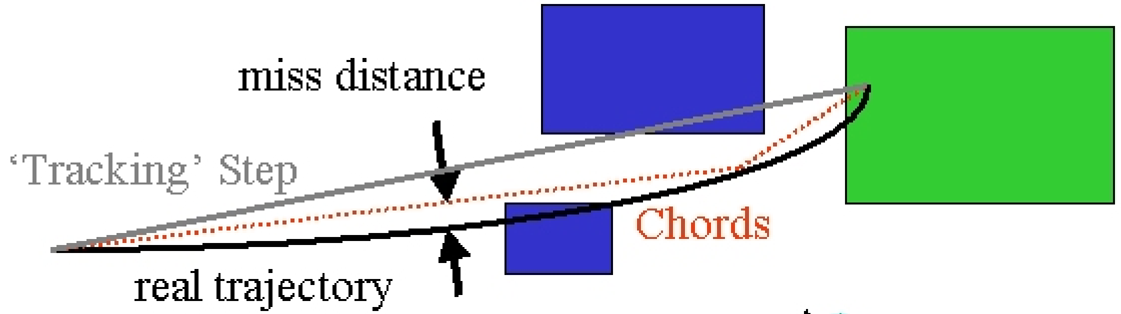
Tracking

Propagation in a Field (1)

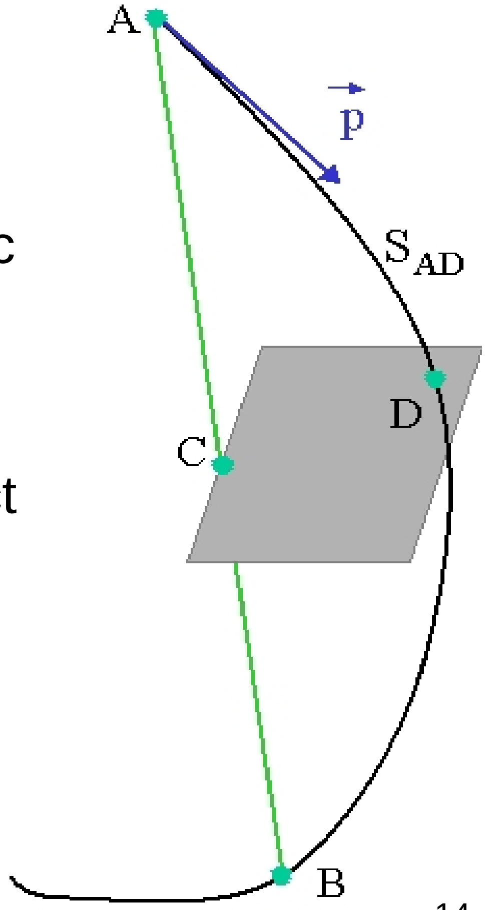
- Geant4 is capable of propagating tracks in a variety of fields:
  - **magnetic**, **electric**, **electromagnetic**, and **gravity** fields
  - uniform or non-uniform (in space and/or time)
  - with user-defined accuracy (trade-off between accuracy and performance)
- In order to propagate a track inside a field, the equation of motion of the particle in the field is integrated
  - In general, this is done using a **Runge-Kutta method** for the integration of ordinary differential equations
  - In ***examples/extended/field/*** you can see some examples of magnetic, electric and gravity fields
  - The user can also create their own type of field, inheriting from **G4VField**, and specifying its associated Equation of Motion, inheriting from the class **G4EqRhs**

Propagation in a Field (2)

The curved path, in a tracking step, is broken up into linear chord segments



miss distance: maximum estimated distance between curve and chord



delta intersection: maximum estimated distance (CD) between the curve and chord intersection on a volume boundary. This is an important parameter, related to the potential systematic errors in the momentum of reconstructed tracks.

delta one step : maximum estimated distance between the endpoint of an 'ordinary' integration step, which does not intersect a volume boundary (i.e. a physics step), and the curve endpoint

These are some of the parameters of the Field Manager, which the user can tune to optimise between accuracy and performance

Propagation in a Field (3)

- Choosing a field :
  - Uniform fields: **G4UniformMagField**, **G4UniformElectricField**, **G4UniformGravityField**
  - Non-uniform fields: concrete classes derived from: **G4MagneticField**, **G4ElectricField**, **G4ElectroMagneticField**, **G4Field** ;
    must define the method: *void GetFieldValue(...)*
- Choosing a stepper :
  - Runge-Kutta integration is used to compute the motion in a general field. There are many general steppers from which to choose, of low and high order, and specialized steppers for pure magnetic fields
  - General: **G4ClassicalRK4** (default), **G4SimpleRunge**, **G4SimpleHeum**, **G4CashKarpRKF45**, etc.
  - Specialized for pure magnetic fields: **G4NystromRK4**, **G4HelixImplicitEuler**, **G4HelixExplicitEuler**, **G4HelixSimpleRunge**, etc.

Propagation in a Field (4)

- Example of how to create a global magnetic field

```
G4UniformMagField* magField =
   new G4UniformMagField( G4ThreeVector( 0.0, 0.0, 4.0*Tesla ) );
G4FieldManager* fieldMgr =
   G4TransportationManager::GetTransportationManager()->GetFieldManager();
fieldMgr->SetDetectorField( magField );
fieldMgr->CreateChordFinder( magField );   // with default parameters
```

- Example of how to create a local magnetic field

```
logicVolume->SetFieldManager( localFieldManager, true );
```

- For more examples see: ***examples/extended/field/***
  - Tracking in magnetic field
  - Tracking in electric field
  - Tracking in overlapping fields (electric and magnetic)
  - Tracking in gravity field

Production Cuts (1)

- Production cuts for secondaries can be specified as range cuts, which are converted (at initialization) into energy thresholds (material-dependent) for secondary **gammas**, **electrons**, **positrons** and **protons**
- For **electrons** and **gammas**, production cuts are absolutely needed in, respectively, ionization and bremsstrahlung processes to avoid the infrared singularity
  - $\sigma_{brems} \sim 1/E_\gamma$; $\sigma_{ionization} \sim 1/(T_e{*}T_e)$
- For **positrons**, the production cut is almost always ignored
  - i.e. positrons are always produced in e+e- pair-production, regardless of their energy (range) (because, in matter, they annihiliate (even at rest) and always produce a pair of gammas that can fly...)
  - except for very high production cuts on gamma (greater than the electron mass, which is the minimum energy of each of the two gammas generated by a positron-electron annihilation...)

Production Cuts (2)

- For **protons**, it has the following meaning:
  if a hadron scatters elastically on a nucleus (of the detector material), this (recoiling) nucleus becomes a new G4Track (i.e. a particle to be transported by Geant4) only if its kinetic energy is above the value:

  (100 keV) * proton_production_cut_in_mm*

- If all these cases, whenever a secondary particle is not produced because it is below the production cut, then its kinetic energy contributes to the so-called **"**continuous energy deposition**"** or **"**along the step**"**
  - As for the concept of "step", also "continuous energy deposition" or "along the step" does not correspond to anything physically
  - But it is a convenient artifact to speed up the simulation

Which Processes are Using Cuts?

- Energy threshold for *gammas* is used in **bremsstrahlung**
- Energy threshold for *electrons* is used in **ionisation** and **e+e- pair-production** processes
- Energy threshold for *positrons* is used in the **e+e- pair-production** process
- Energy thresholds for *gammas* and *electrons* can be used in **all electromagnetic discrete processes** if the "ApplyCuts" option is activated
  - Photoelectric effect, Compton, etc.
- Energy threshold for *protons* are used in processes of **elastic hadron-nucleus scattering**
  - defining the kinetic energy threshold of nuclear recoil

How to Set Production Cuts?

- A range cut value is set by default to **0.7 mm** in Geant4 reference physics lists.
- This can be changed via UI (User Interface) command, e.g.:

```
/run/setCut 0.05 mm
```

- You can set a different value for each particle type, e.g.:

```
/run/setCutForAGivenParticle e-       0.05 mm
/run/setCutForAGivenParticle e+       0.01 mm
/run/setCutForAGivenParticle gamma 1.0 cm
/run/setCutForAGivenParticle proton  0.2 mm
```

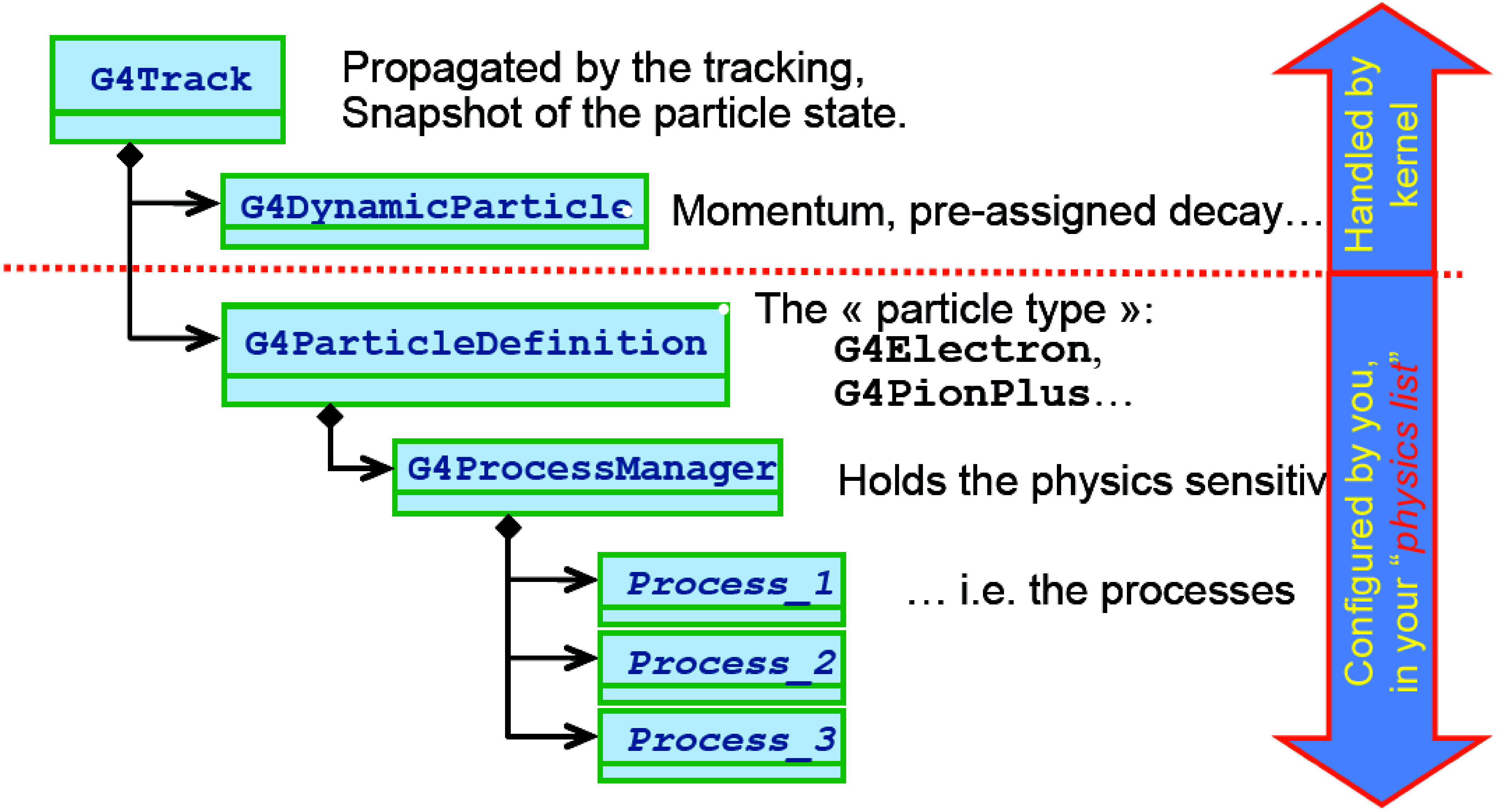
- Production cuts can be set globally or per-region
- For a complex detectors, the optimization of the range cuts per region is crucial for the CPU performance of the simulation

Special Tracking Cuts

- By default in Geant4, there are only production cuts, and **not tracking cuts**: the produced particles are tracked down to zero kinetic energy (range)
  - The treatment is reliable down to ~1 keV : below it is approximated
- For optimization reasons, a user may limit the tracking of particular particle types in specified volumes
- The approach is as follows: special user cuts are registered in the G4UserLimits class, which is associated with the logical volume class. The current default list is:
  - max allowed step size
  - max total track length
  - max total time of flight
  - min kinetic energy
  - min remaining range
- For an example, see: ***examples/basic/B2***

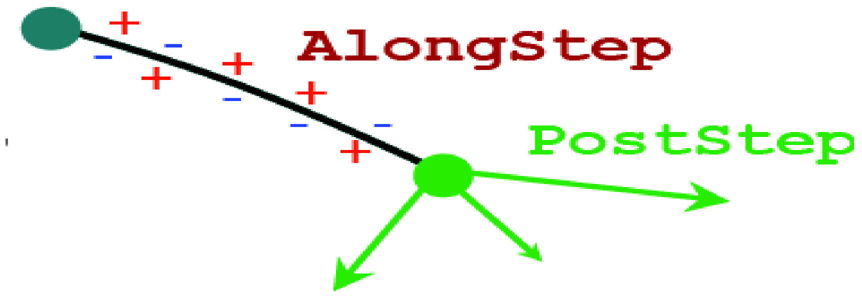
Processes

Track and Processes



Processes: 3 kinds of Actions

- Abstract class G4VProcess defines the common interface of all processes in Geant4
  - including Transportation
- Defines three kinds of actions:
  - **AtRest** actions
    - Decay at rest, e+ annihilation at rest, nuclear capture at rest, etc.
  - **AlongStep** actions
    - "Continuous" energy deposition (e.g. production below threshold); fields effect
  - **PostStep** actions
    - In-flight decays and interactions



- G4ProcessManager has 3 vectors of actions (per particle-type):
  - one for **AtRest** actions: these processes compete
  - one for **AlongStep** actions: these processes cooperate
  - one for **PostStep** actions: these processes compete

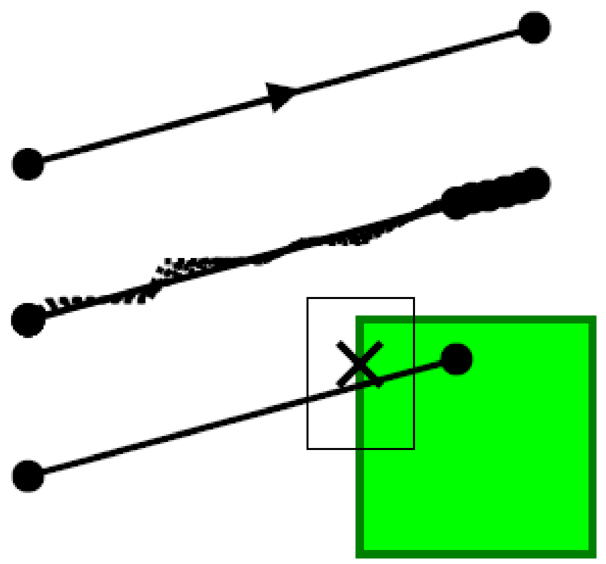
G4VProcess: action methods

- A process will implement any combination of the 3 actions:
  - **AtRest**
  - **AlongStep**
  - **PostStep**

  e.g. decay : AtRest + PostStep
- Each action defines 2 methods:
  - **GetPhysicalInteractionLength()**
    - Used to **limit the step size**
      - because the process triggers an interaction, a decay, geometry boundary, a user's limit, etc.
      - The cross section for a in-flight physics process, or the mean lifetime for an at-rest process is used
  - **DoIt()**
    - Implements the **actual action** to be applied to the track
      - typically the generation of the final state

Ordering of the Processes

- Process ordering, in general, is not critical…
- … except for multiple-scattering and transportation
- Assuming *n* processes, the ordering of the *AlongStepGetPhysicalInteractionLength* should be:
  - [n-2] …
  - [n-1] multiple scattering (before last)
  - [n] transportation (last)
- Why?
  - Processes return a "true path length"
  - The multiple scattering virtually folds up this true path length into a shorter "geometrical path length"
  - Based on this new length, the transportation can geometrically limits the step



Physics

G4 Datasets (1)

- Some physics models are data-driven, i.e. they are phenomenological models that needs some input data
- If you build G4 with the option **GEANT4_INSTALL_DATA** then the datasets are automatically downloaded and installed
- Else (you want or need to do it manually, e.g. for older versions of G4) you need to install the datasets yourself and then inform G4 where they are by defining the following environmental variables (e.g. for the latest version G4 10.0):

```
export G4LEDATA=/dir-path/G4EMLOW6.35
export G4LEVELGAMMADATA=/dir-path/PhotonEvaporation3.0
export G4SAIDXSDATA=/dir-path/G4SAIDDATA1.1
export G4NEUTRONXSDATA=/dir-path/G4NEUTRONXS1.4
export G4NEUTRONHPDATA=/dir-path/G4NDL4.4
export G4RADIOACTIVEDATA=/dir-path/RadioactiveDecay4.0
export G4REALSURFACEDATA=/dir-path/RealSurface1.0
```

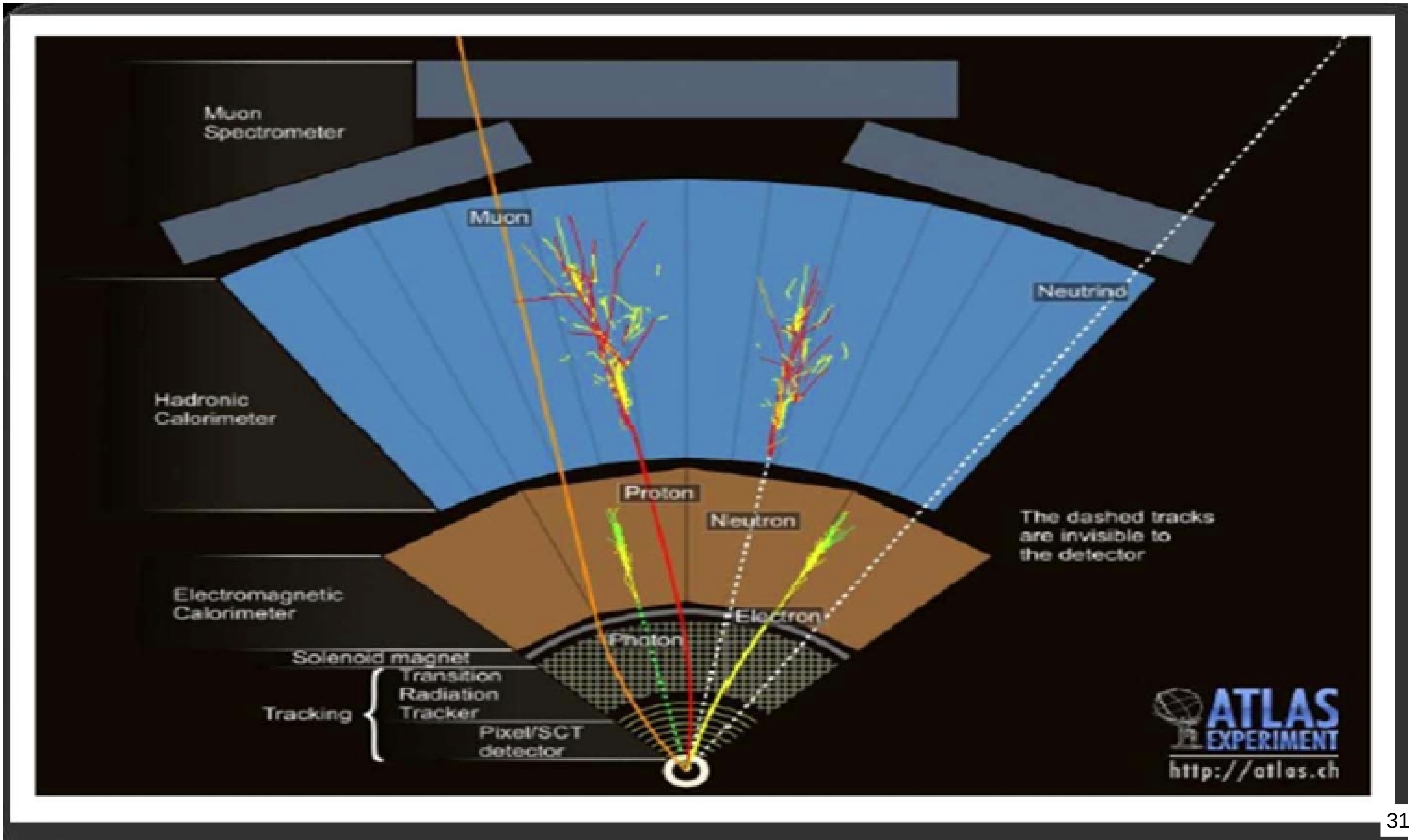
G4 Datasets (2)

- **G4LEDATA** : low-energy electromagnetic data, mostly derived from Livermore data libraries; used in all EM options
- **G4LEVELGAMMADATA** : photon evaporation data, come from the Evaluated Nuclear Structure Data File (ENSDF); used by Preco
- **G4SAIDXSDATA** : data evaluated from the SAID database for nucleon and pion cross sections below 3 GeV; used in all physics lists
- **G4NEUTRONXSDATA** : evaluated neutron cross sections derived from G4NDL by averaging in bin of energies; used in all physics lists
- **G4NEUTRONHPDATA** : evaluated neutron data of cross sections, angular distributions and final-state information; come largely from the ENDF/B-VI library; used only in _HP physics lists
- **G4RADIOACTIVEDATA** : radioactive decay data, come from the ENSDF; used only when radioactive decay is activated
- **G4REALSURFACEDATA** : data for measured optical surface reflectance look-up tables; used only when optical physics is activated

Electromagnetic physics (EM)

Particle interactions

Each particle type has its own set of physics processes.
Only electromagnetic effects are directly measurable



Main electromagnetic processes

Gamma

- Conversion :
  γ -> e+ e- , μ+ μ-
- Compton scattering :
  γ *(atomic)*e- -> γ *(free)*e-
- Photo-electric
  γ material -> *(free)*e-
- Rayleigh scattering
  γ atom -> γ atom

Muon

- Pair production
  μ- atom -> μ- e+ e-
- Bremsstrahlung
  μ- (atom) -> μ- γ
- MSC (Coulomb scattering) :
  μ- atom -> μ- atom
- Ionization :
  μ- atom -> μ- ion+ e-

Total cross section:
⇨ step length

Differential & partial cross sections :
⇨ final state
(multiplicity & spectra)

Electron, Positron

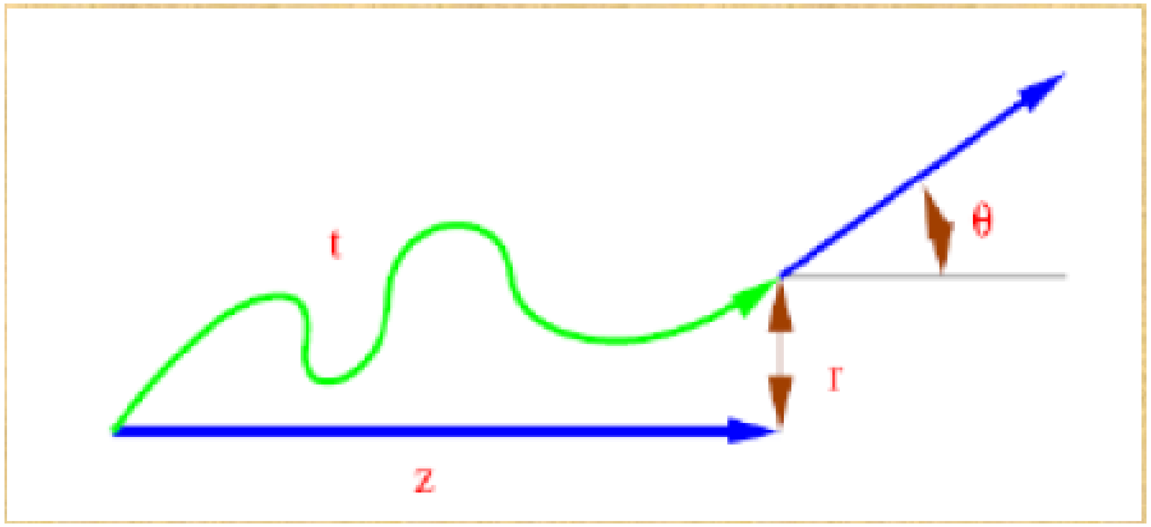
- Bremsstrahlung
  e- (atom) -> e- γ
- MSC (Coulomb scattering):
  e- atom -> e- atom
- Ionization :
  e- atom -> e- ion+ e-
- Positron annihilation
  e+ e- -> γ γ

Charged hadron, ion

- (Bremsstrahlung
  h- (atom) -> h- γ)
- MSC (Coulomb scattering):
  h- atom -> h- atom
- Ionization :
  h- atom -> h- ion+ e-

Multiple (Coulomb) scattering (MSC)

- Charged particles traversing a finite thickness of matter suffer a huge number (millions) of elastic Coulomb scatterings
- The cumulative effect of these small angle scatterings is mainly a net deflection from the original particle direction
- In most cases, to save CPU time, these multiple scatterings are not simulated individually, but in a "condensed" form
- Various algorithms exist, and new ones under development. One of the main differences between codes



Electromagnetic physics

- Typical validity of electromagnetic physics **$\geq$ 1 keV** ;
  for a few processes, extensions to lower energies
- CPU performance of electromagnetic physics is critical :
  continuous effort to improve it
- Detailed validation of electromagnetic physics is necessary before the validation of hadronic physics
- Typical precision in electromagnetic physics is **~1%**
  - QED is extremely precise for elementary processes,
    but atomic and medium effects, important for detector simulations, bring larger uncertainties...
  - Moreover, the “condensed” description of multiple scattering introduces further approximations...
  - Continuous effort to improve the models

EM options

- **Baseline** (default)
  - Used in production by ATLAS
  - Available in all reference physics lists, e.g. FTFP_BERT
- Fast (EMV)
  - Used in production by CMS: good for crystals, not for sampling calo
  - Available in _**EMV** variants of physics lists
- Accurate (EMY)
  - Used in medical and space science applications
  - Available in _**EMY** variants of physics lists
- Other options are available:
  - _**EMX** : experimental, used by LHCb
  - _**EMZ** : the most precise EM available in G4
  - _**LIV** : models based on the Livermore database
  - _**PEN** : Penelope models implemented in Geant4

Optical Photons

- A **photon** is considered to be optical when its wavelength is greater than the typical inter-atomic distance
- In Geant4, for convenience, optical photons are treated as a separated particle class, **G4OpticalPhoton**, distinct from the class of high-energy photons, **G4Gamma**
- Three processes in Geant4 can produce optical photons: Cerenkov effect, scintillation, and transition radiation
- Geant4 processes that can be associated to optical photons: refraction, reflection, absorption, scattering, wavelength shifting
- Optical properties of media (reflectivity, transmission, etc.) should be specified (in G4MaterialPropertiesTable linked to G4Material)
- For some examples, see: ***examples/extended/optical/***

Hadronic physics (HAD)

Hadronic interactions

- Hadrons ($\pi^\pm$, $K^\pm$, K°_L , p, n, α, *etc.*), produced in jets and decays, traverse the detectors (H,C,Ar,Si,Al,Fe,Cu,W,Pb...)
- Therefore we need to model hadronic interactions

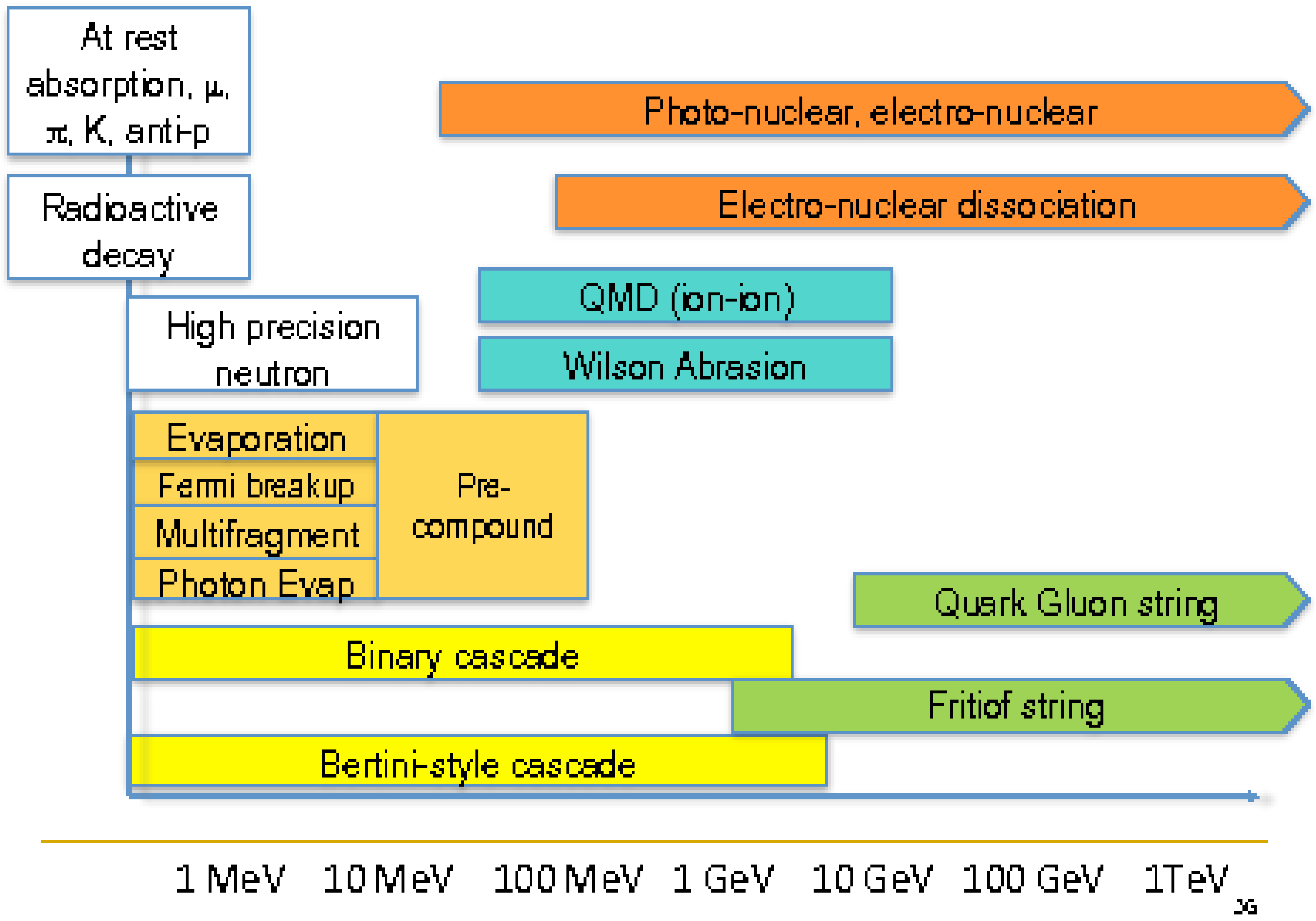
  hadron – nucleus -> *anything*

  in our detector simulations
- In principle, QCD is the theory that describes all hadronic interactions; in practice, perturbative calculations are applicable only in a tiny (but important!) phase-space region
  - the hard scattering at high transverse momentum

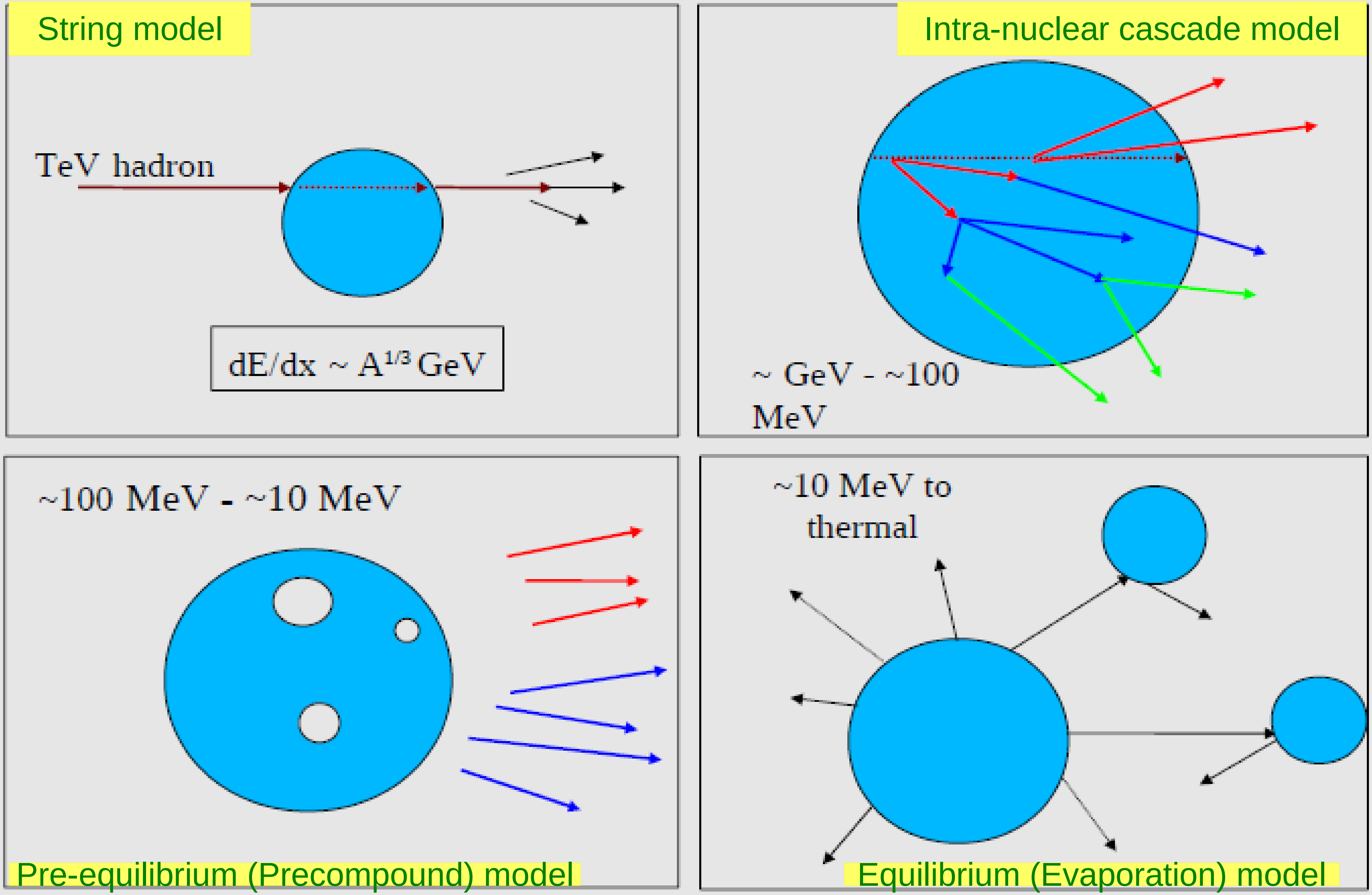
  whereas for the rest, i.e. most of the phase space
  - soft scattering, re-scattering, hadronization, nucleus de-excitation

  only approximate models are available
- Hadronic models are valid for limited combinations of
  - **particle type - energy - target material**

Partial hadronic model inventory



Hadronic Interactions from TeV to meV


String model
TeV hadron
dE/dx ~ A^1/3 GeV
Intra-nuclear cascade model
~ GeV - ~100 MeV
~100 MeV - ~10 MeV
Pre-equilibrium (Precompound) model
~10 MeV to thermal
Equilibrium (Evaporation) model

An interesting complication: Neutrons

- Neutrons are abundantly produced
  - Mostly "soft" neutrons, produced by the de-excitations of nuclei, after hadron-nucleus interactions
  - It is typically the 3rd most produced particle (after e-, γ)
- Before a neutron "disappears" via an inelastic interaction, it can have many elastic scatterings with nuclei, and eventually it can "thermalize" in the environment
- The CPU time of the detector simulation can vary by an order of magnitude according to the physical accuracy of the neutron transportation simulation
  - For typical high-energy applications, a simple treatment is enough (luckily!)
  - For activation and radiation damage studies, a more precise, data-driven and isotope-specific treatment is needed, especially for neutrons of kinetic energy below **~ MeV**

Neutron-HP

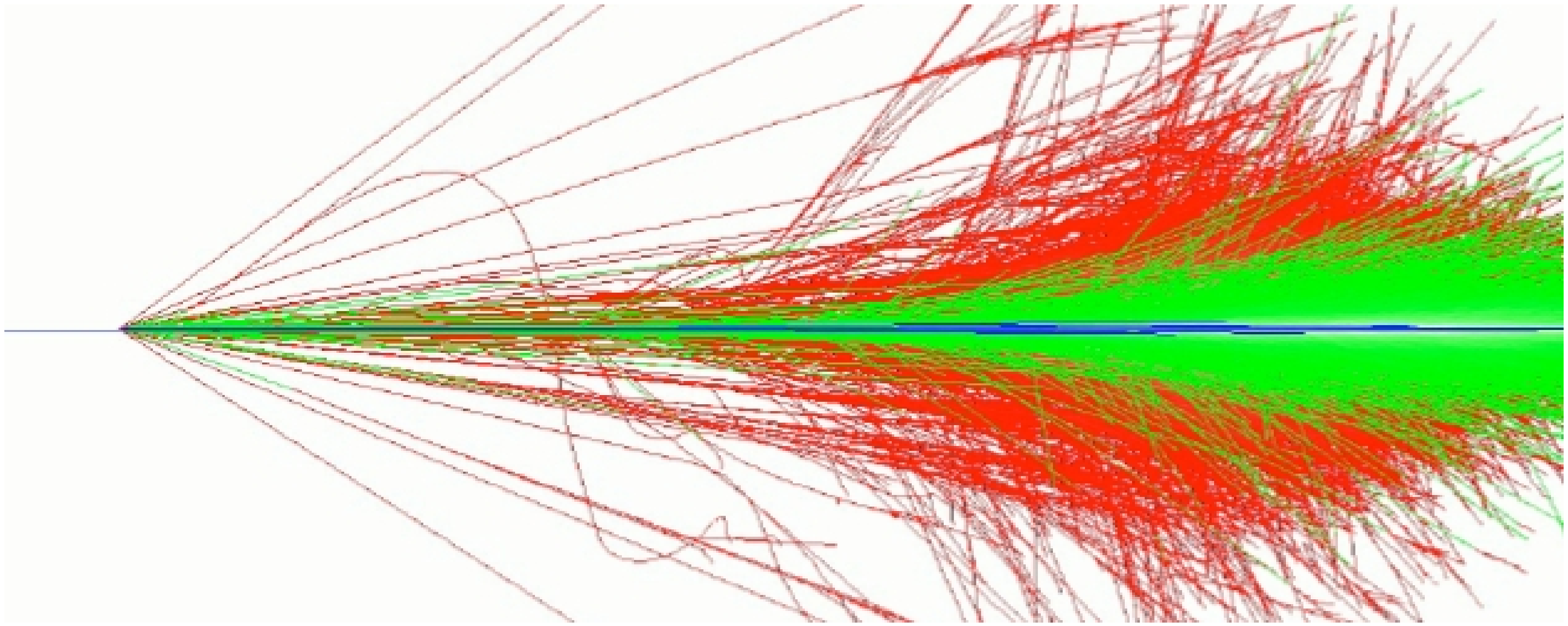
- **H**igh **P**recision treatment of low-energy neutrons
  - **E$_{\text{kin}}$ < 20 MeV** , down to thermal energies
  - Includes 4 types of interactions:
    radiative capture, elastic scattering, fission, inelastic scattering
  - Based on evaluated neutron scattering data libraries
    (pointed by the environmental variable **G4NEUTRONHPDATA**)
  - It is precise, but very slow!

- It is not needed for most high-energy applications; useful for:
  - cavern background, shielding, radiation damage, radio-protection

- Not used in most physics lists.
  If you need it, use one of the _**HP** physics lists:
  FTFP_BERT_**HP** , QGSP_BERT_**HP** , QGSP_BIC_**HP** , Shielding

Hadronic showers

- A single hadron impinging on a large block of matter (e.g. a hadron calorimeter) produces secondary hadrons of lower energies, which in turn can produce other hadrons, and so on: the set of these particles is called a **hadronic shower**
    - e-/e+/γ (electromagnetic component) are also produced copiously because of **π° -> γ γ** and ionization of charged particles
- The development of a hadronic shower involves many energy scales, from hundreds of GeV down to thermal energies



Jets

The simulation of hadronic showers is an important ingredient for the simulation of jets

- The other ingredients are:
  - the Monte Carlo event generator
  - the experiment-specific aspects: geometry, digitization, pile-up
- Jets (collimated sprays of hadrons) are produced by strong (QCD) or electroweak (hadronic decays of τ / W / Z / H) interactions
- Jets can be part of the signal and/or the background
  - multi-jets in the same event are typical in hadron colliders as LHC, but it is also frequent in high-energy e+-e- linear colliders as ILC/CLIC
- For future accelerators (e.g. ILC/CLIC), the simulation of jets is essential for the optimal design of the detector
- For ATLAS and CMS, the simulation of jets is now important for physics analysis

Physics Lists

What is a Physics List ?

- A class that collects all the particles, physics processes, and production thresholds needed by your application
- One and only one physics list should be present in each Geant4 application
- There is no default physics list: it should always be explicitly specified
- It is a very flexible way to build a physics environment:
  - Users can pick only the particles they need
  - Users can assign to each selected particle only the processes they are interested in
- But, users must have a good understanding of the physics required in their application:
  - Omission of particles or physics processes will cause errors or poor simulation

Why do we need a Physics List ?

Nature has just one "physics": so why Geant4 does not provide a complete and unique set of particles and physics processes that everyone can use?

- There are many different physics models, corresponding to a variety of approximations of the real phenomena
  - very much the case for hadronic physics,
  - but also for electromagnetic physics.

  According to the application, one can be better than another. Comparing them can give an idea of systematic errors.

- Simulation speed is important
  - a user may prefer a less detailed but faster approximation

- Often all the physics and particles are not needed:
  - e.g. most high-energy applications do not need a detailed transportation of low-energy neutrons

Reference Physics Lists

- Writing a complete and realistic physics list for EM physics and even more for hadronic physics is involved, and it depends on the application. To make things easier, pre-packaged **reference physics lists** are provided by Geant4, according to some reference use cases
- Few choices are available for EM physics (different production cuts and/or multiple scattering); instead, several possibilities are available for hadronics physics: e.g. **FTFP_BERT**, **FTFP_BERT_HP**, **Shielding** **QGSP_FTFP_BERT**, **QGSP_BIC_EMY**
- These lists are "best guess" of the physics needed in a given case; they are intended as starting point (and their builders can be re-used); users are responsible of validating the physics lists for their application

FTFP_BERT

Recommended physics list for High-Energy Physics.
Its main components are the following:

- **FTF** (Fritiof string) model, above 4 GeV
- **BERT** (Bertini cascade) model, below 5 GeV
- Nucleus de-excitation: **P**recompound + evaporation
- Neutron capture
- Nuclear capture of negatively charged hadrons at rest
- Gamma- and electro-nuclear
- Standard electromagnetics

- NO : neutron-HP, radioactive decay, optical photons

How to use a reference Physics List

Let's consider the example of FTFP_BERT :
In your main program:

```
 #include "FTFP_BERT.hh"
...
int main( int argc, char** argv ) {
   ...
   G4VModularPhysicsList* physicsList = new FTFP_BERT;
   runManager->SetUserInitialization( physicsList );
   ...
}
```

How to add extra physics to a reference P.L.

- Adding radioactive decay :
  In your main program:

```
#include "G4RadioactiveDecayPhysics.hh"
int main( int argc, char** argv ) {
  ...
  G4VModularPhysicsList* physicsList = new FTFP_BERT;
  physicsList->RegisterPhysics( new G4RadioactiveDecayPhysics );
  runManager->SetUserInitialization( physicsList );
  ...
}
```

- Adding optical photon and its processes :
  In your main program:

```
#include "G4OpticalPhysics.hh"
int main( int argc, char** argv ) {
  ...
  G4VModularPhysicsList* physicsList = new FTFP_BERT;
  physicsList->RegisterPhysics( new G4OpticalPhysics );
  runManager->SetUserInitialization( physicsList );
  ...
}
```

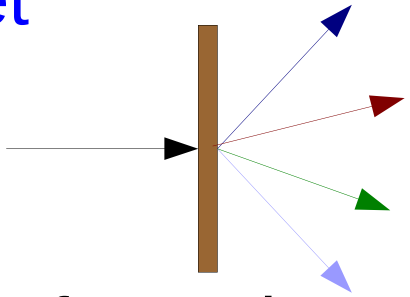
Recap: Model, Process, Physics List

- Physics model = final-state generator
  - Validated and tuned by Geant4 developers with thin-target data
- Physics process = cross section + final-state model
  - Different physics models can share the same cross section
- Physics list = a list of physics processes associated to each particle present in the simulation
  - Chosen by users: tradeoff accuracy vs. speed
  - Geant4 offers some reference physics lists ready to be used
  - Validated by the users with (test-beam and/or collision) data

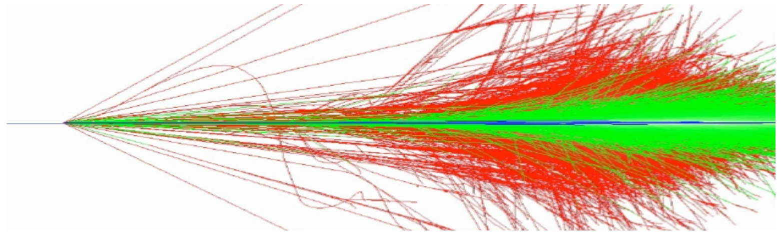
Validation

Validation & tuning of hadronic models

- The developers of the hadronic models are responsible of the tuning & validation of these models with thin-target (microscopic, single-interaction) measurements



- Validation of complete physics configurations is performed by users mostly via measurements of hadronic showers in calorimeter test-beam setups (thick targets)



- The most important application of the hadronic models for collider experiments is the simulation of jets, which involves:

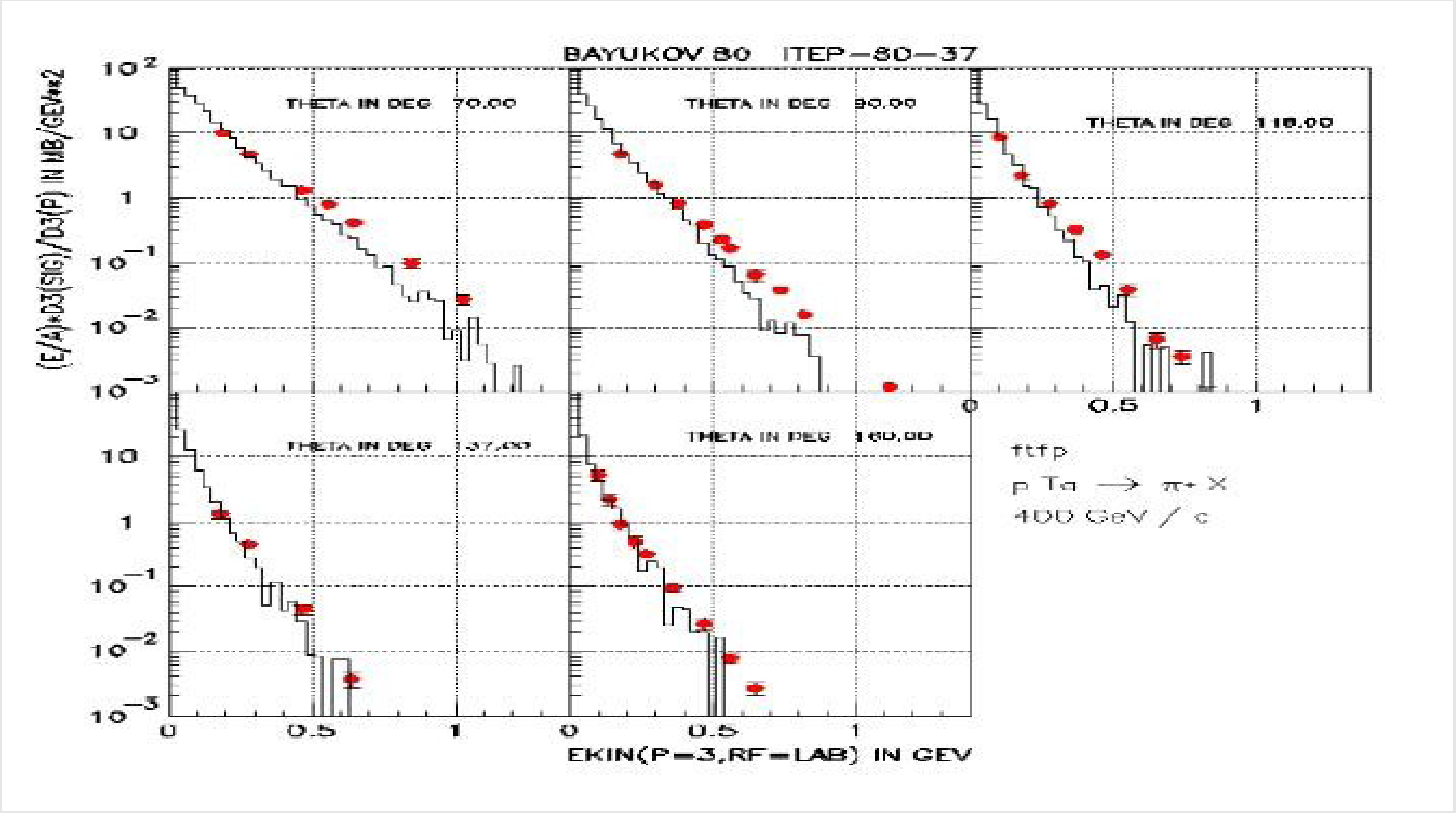
  1. the Monte Carlo event generator

  2. the convolution of the showers for each constituent hadron

  3. experiment specific: geometry & materials, digitization, etc.

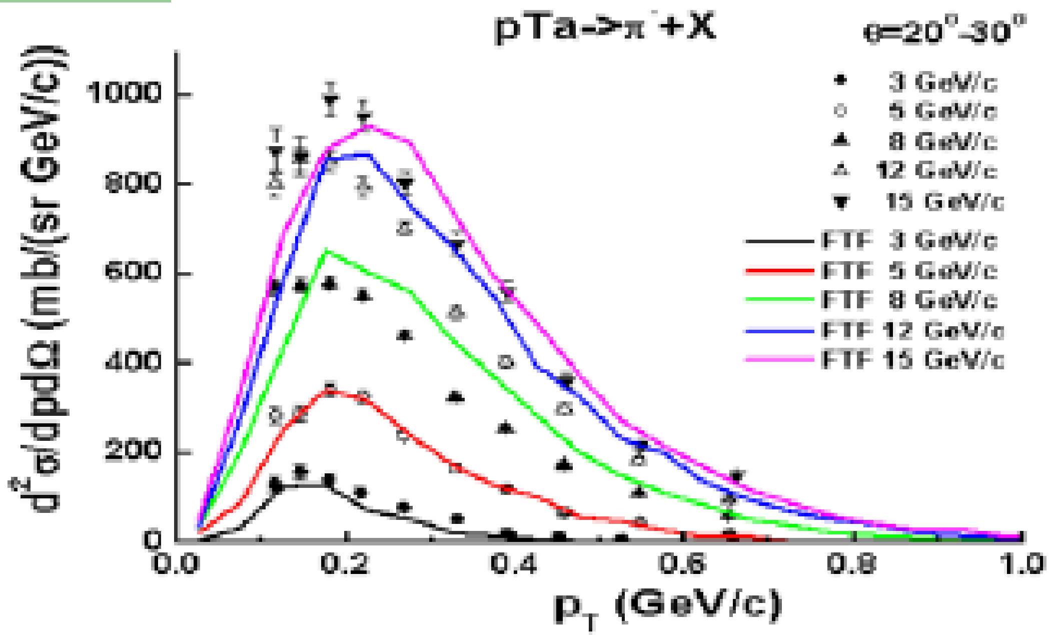
Model-level thin-target test

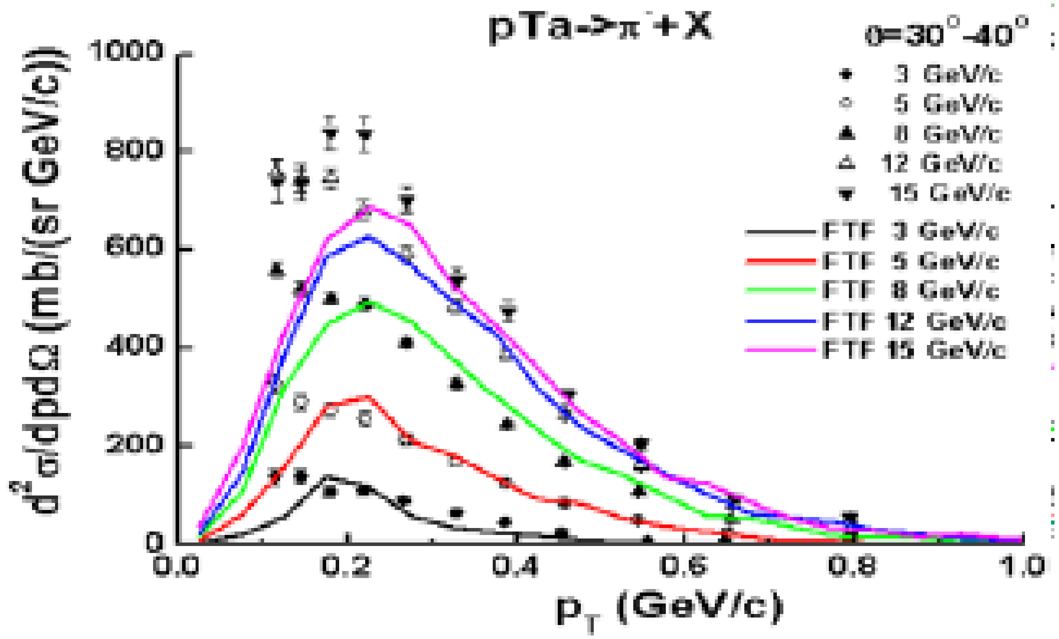
FTF Results at 400 GeV/c
p Ta -> pi+ X

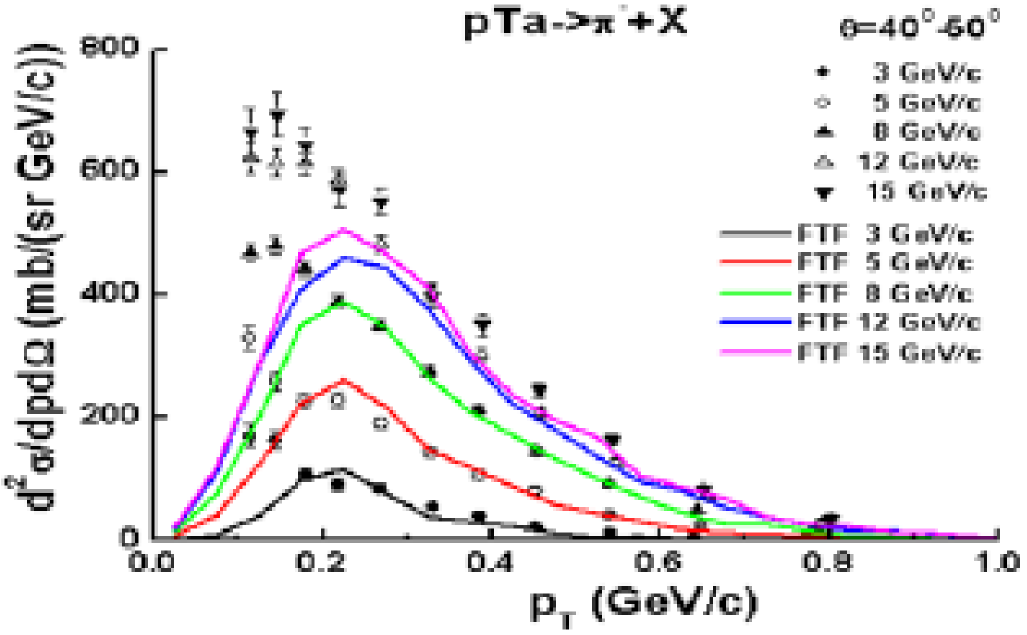


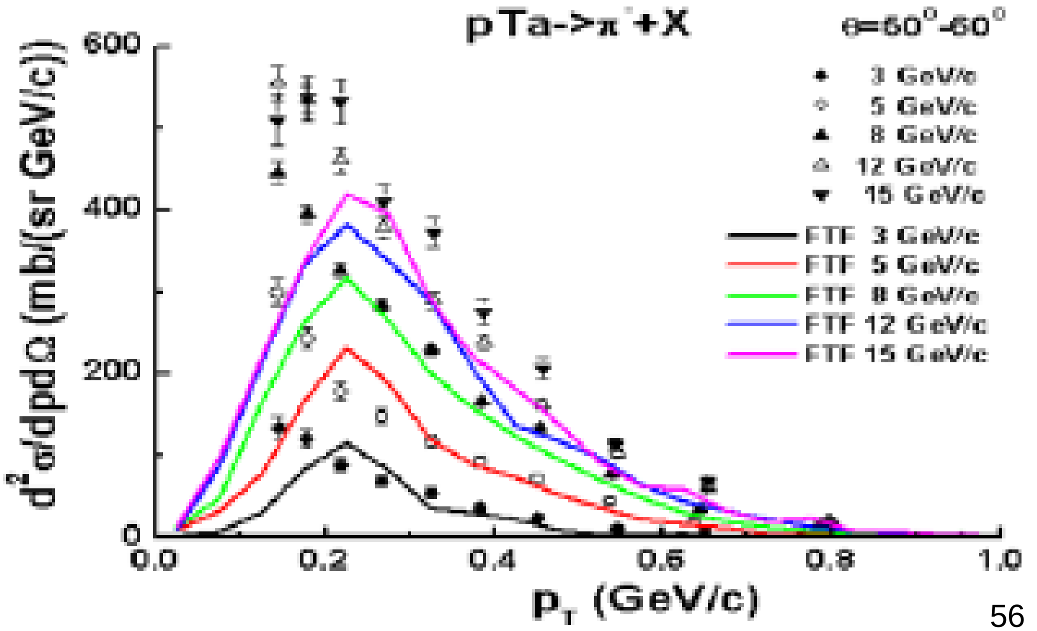
Model-level thin-target test

FTF validation, HARP-CDP data



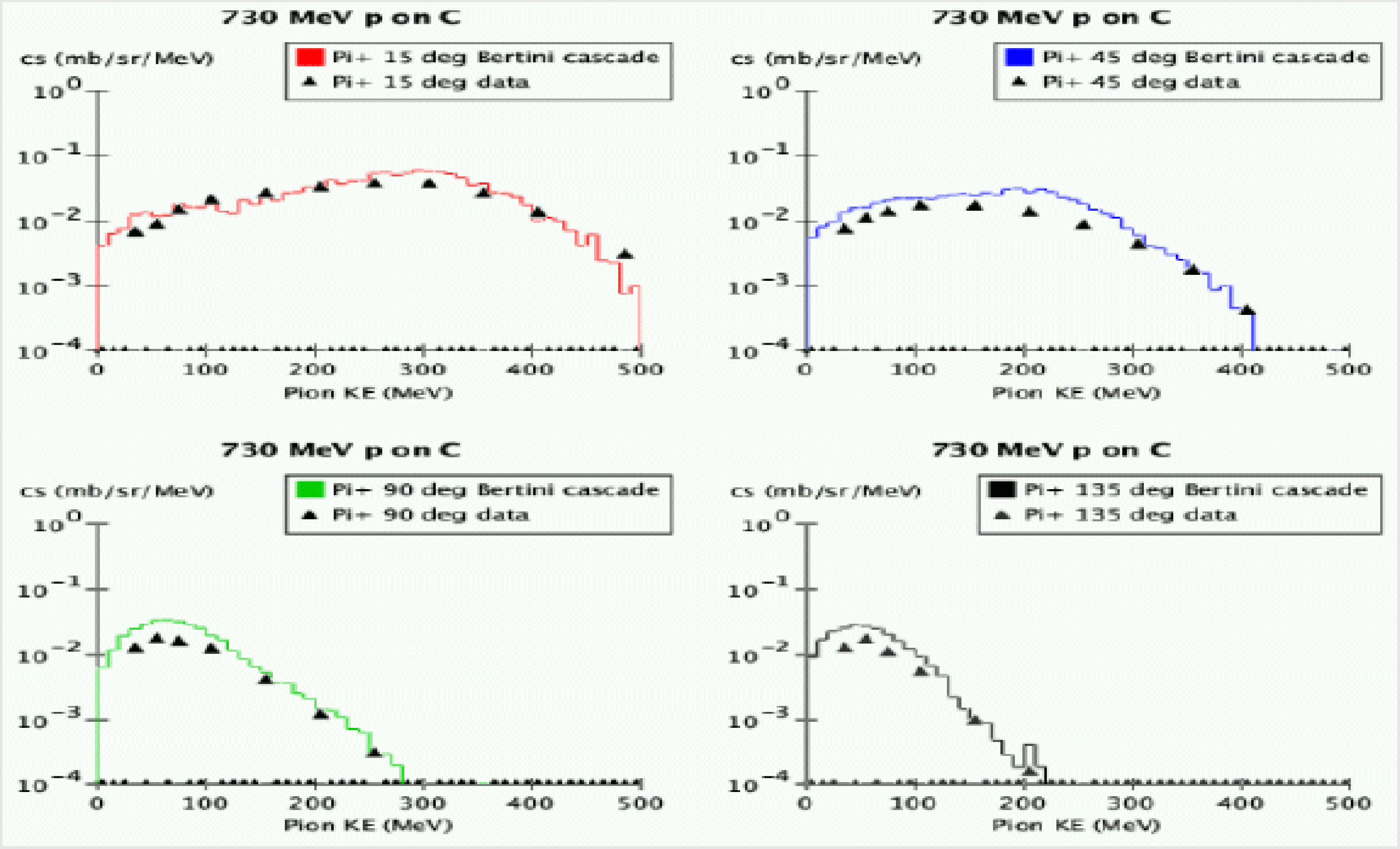






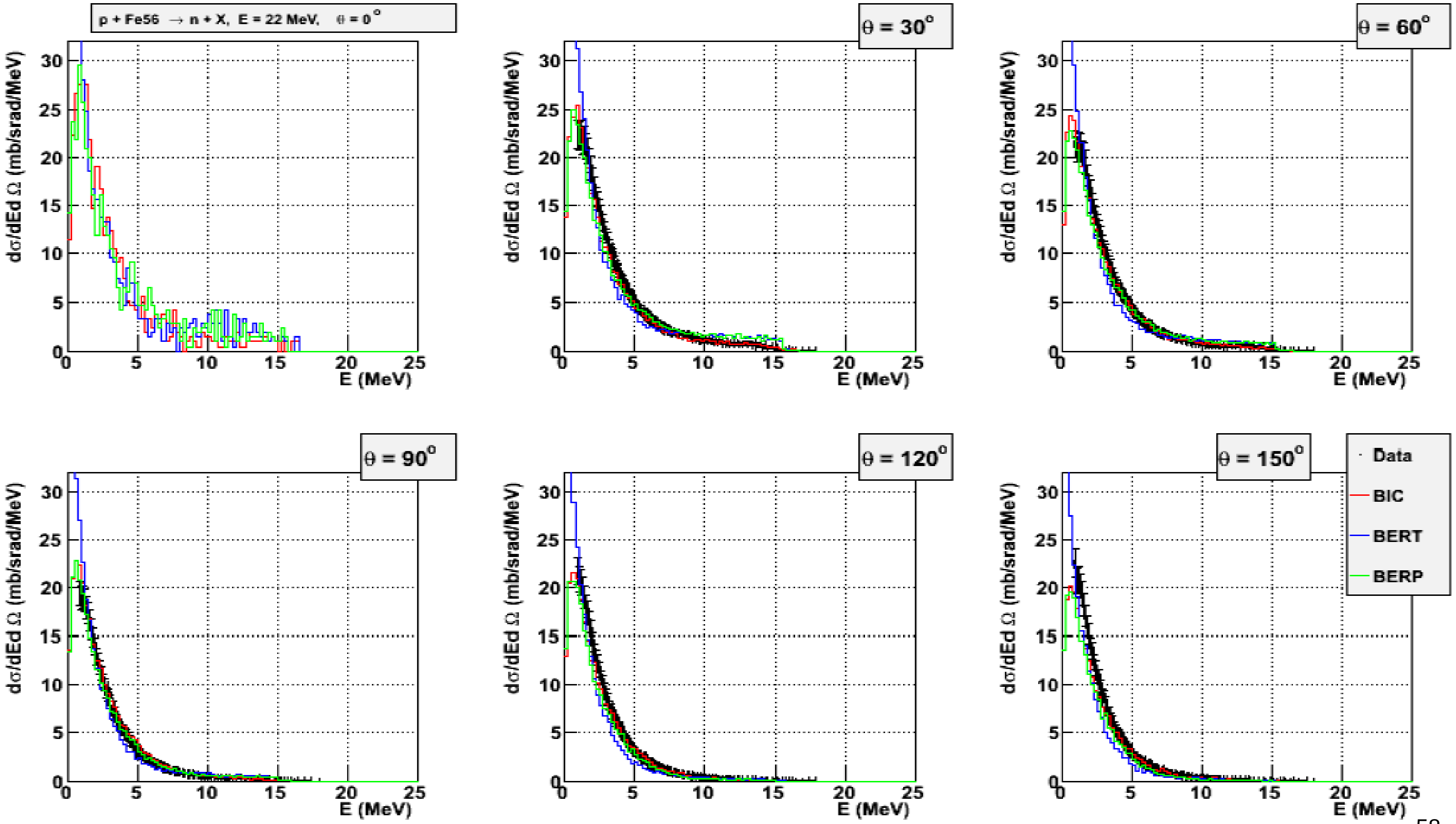
Model-level thin-target test

Validation of the Bertini Cascade

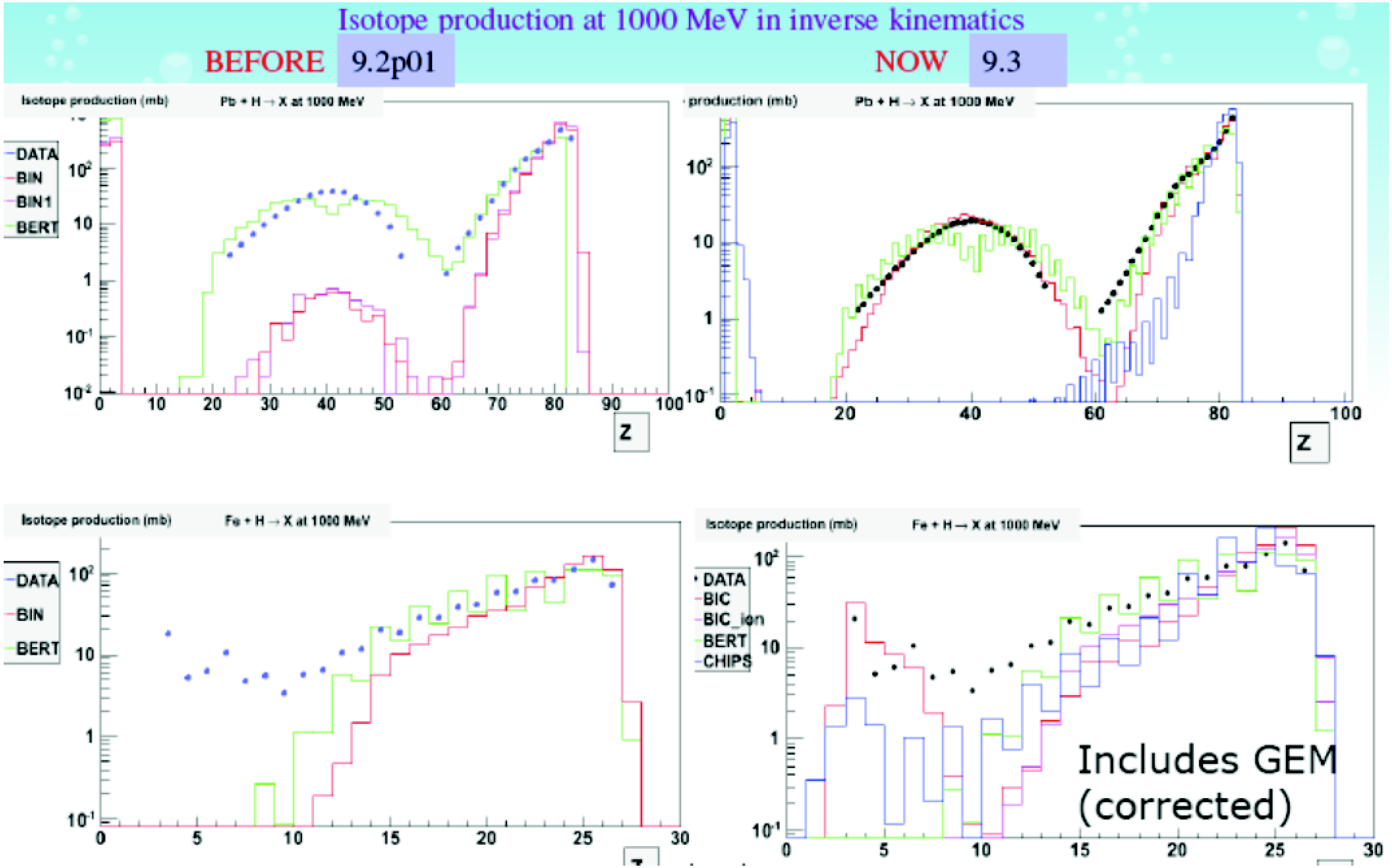


Model-level thin-target test

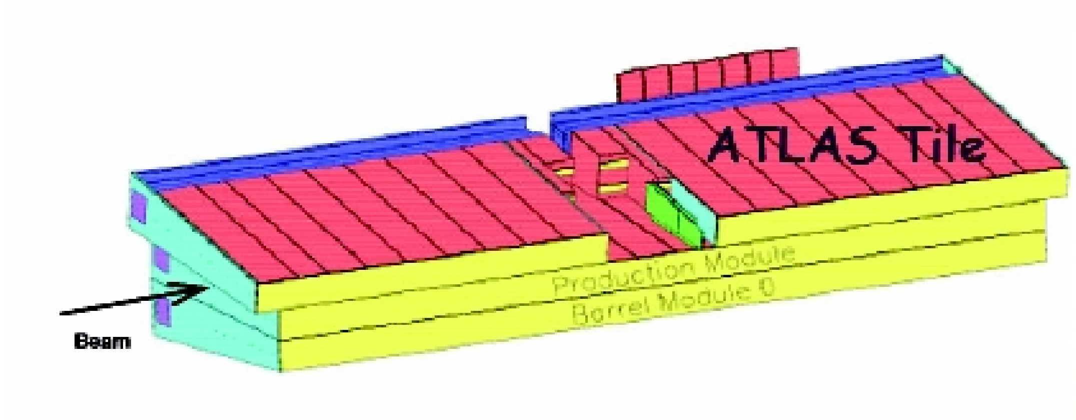
Preco validation, 22 MeV p – Fe -> n

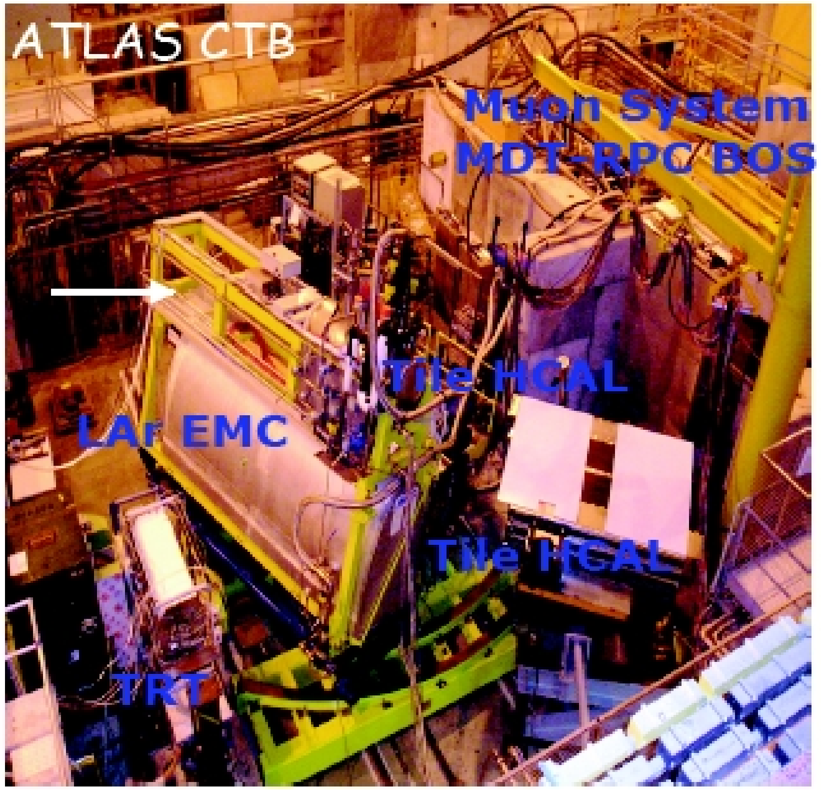


Validation of Precompound & de-excitation



LHC calorimeter test-beams





Calorimeter observables

- The simulation of hadronic showers can be validated with calorimeter test-beam set-ups, with pion and proton beams of various energies, considering the following observables:
  - Energy response: E_{rec} / E_{beam}
  - Energy resolution: Δ E_{rec} / E_{rec}
  - Shower profile:
    - Longitudinal: $E_{rec}(z)$ / E_{rec}
    - Lateral (transverse or radial): $E_{rec}(r)$ / E_{rec}

- Note that we can test directly only single-hadron showers in calorimeter test-beam set-ups, whereas for a collider experiment (e.g. ATLAS and CMS) jets are measured.
  The simulation of jets involves:
  1. the Monte Carlo Event Generator
  2. the convolution of the showers for each constituent hadron

A long journey...

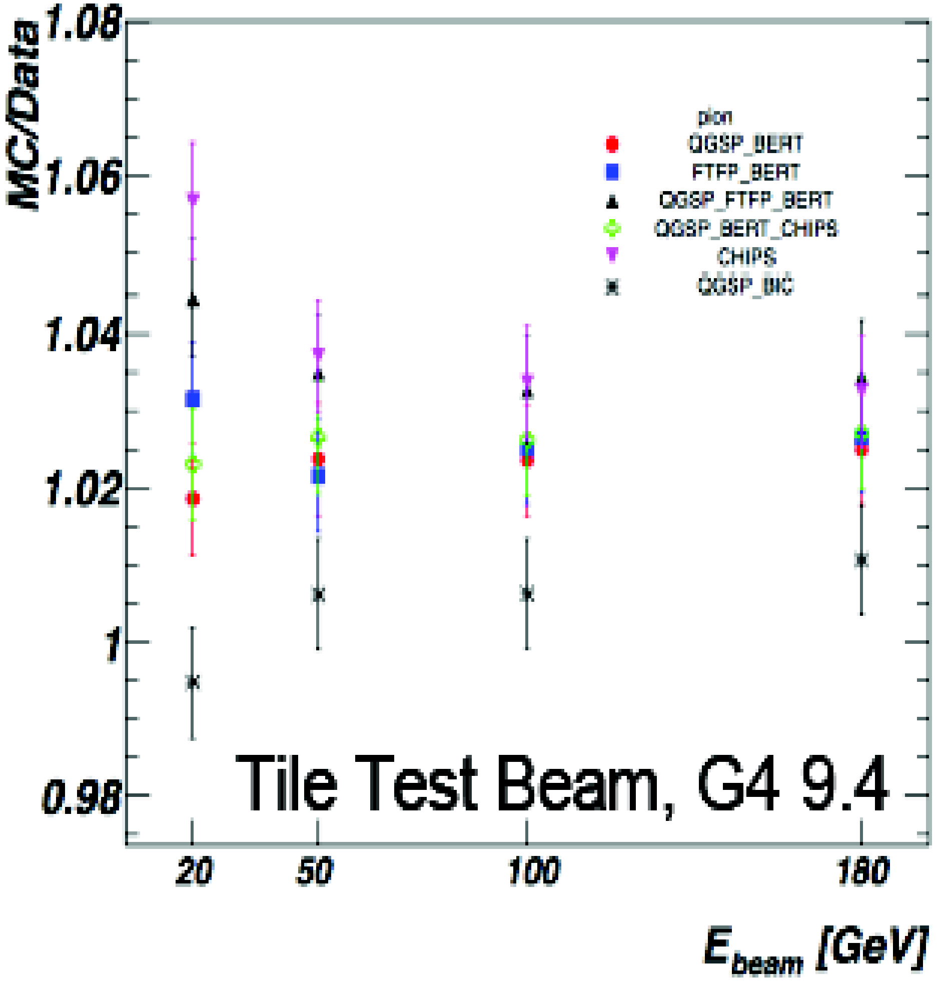
- Once you have collected data from a calorimeter test-beam set-up with hadron beams, there is a long work needed before drawing conclusions on the hadronic simulation:
  - Cleaning/selection cuts to have the purest possible sample
  - Model beam composition and spread
  - Check material composition, geometry, dead material
  - Model quenching effects (Birks’ law), photo-statistics, etc.
  - Include noise, cross-talk, DAQ time-window, and digitization

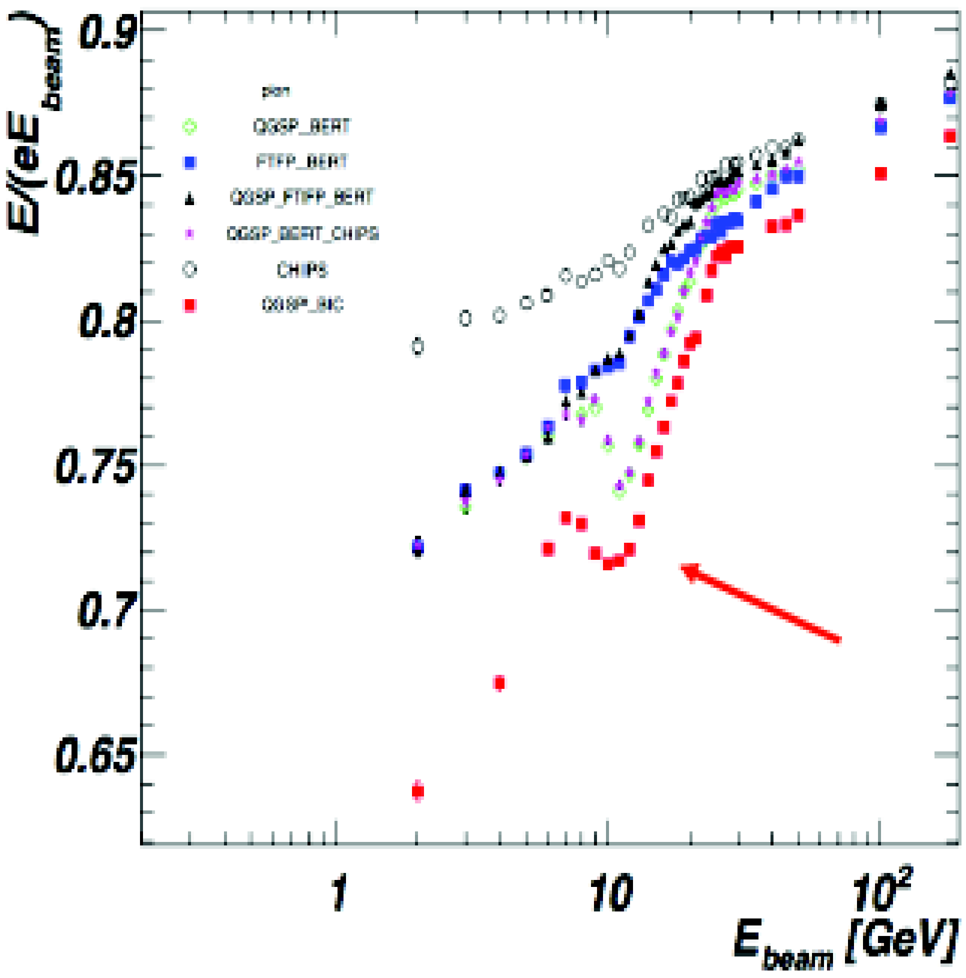
To help on these steps:

- Special triggers
- Muon beam
- Electron beam (also needed for the electromagnetic calibration)

Energy response

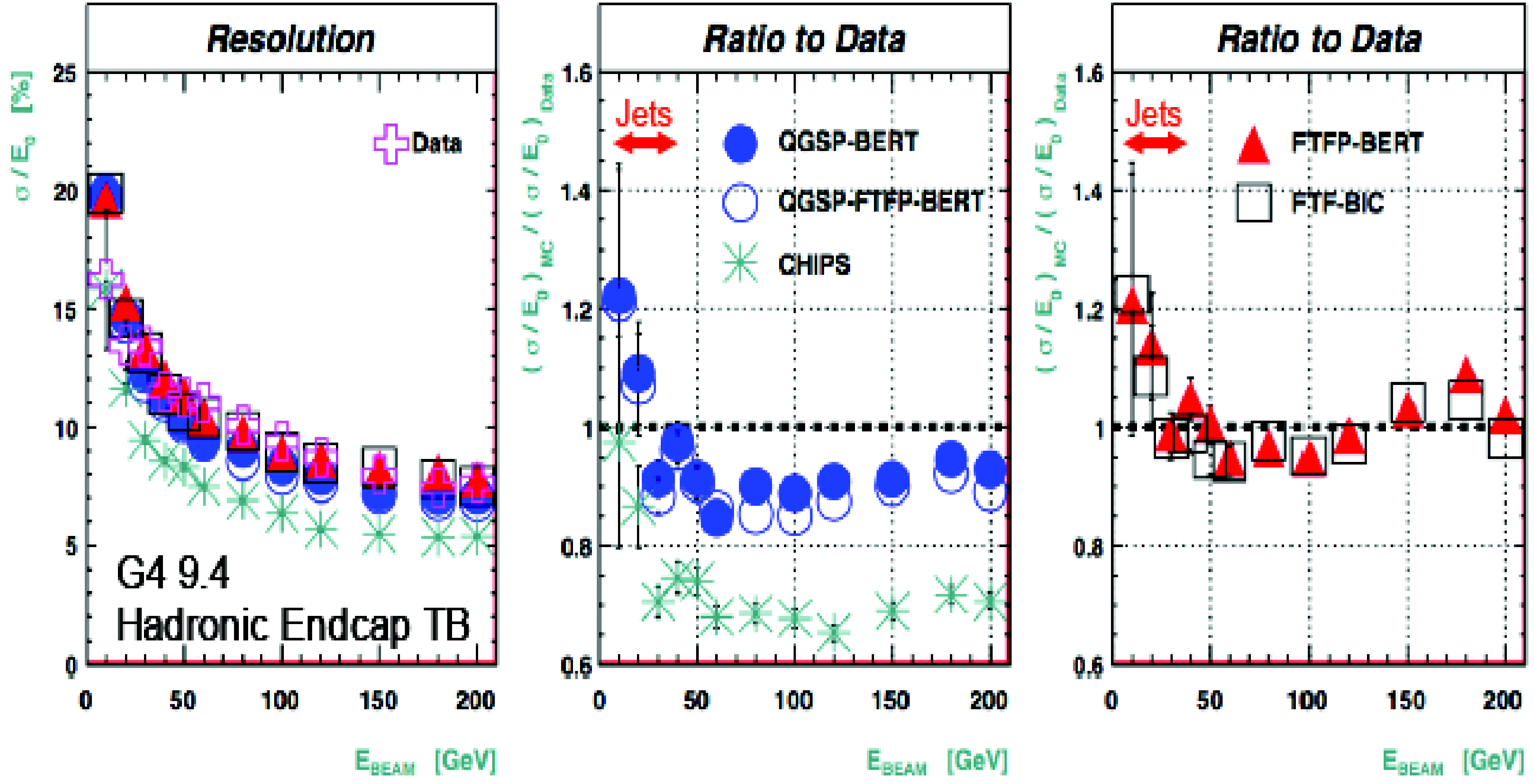
ATLAS TileCal test-beam





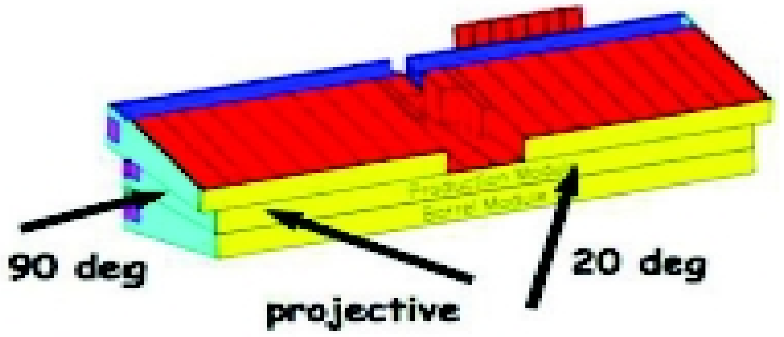
Energy resolution

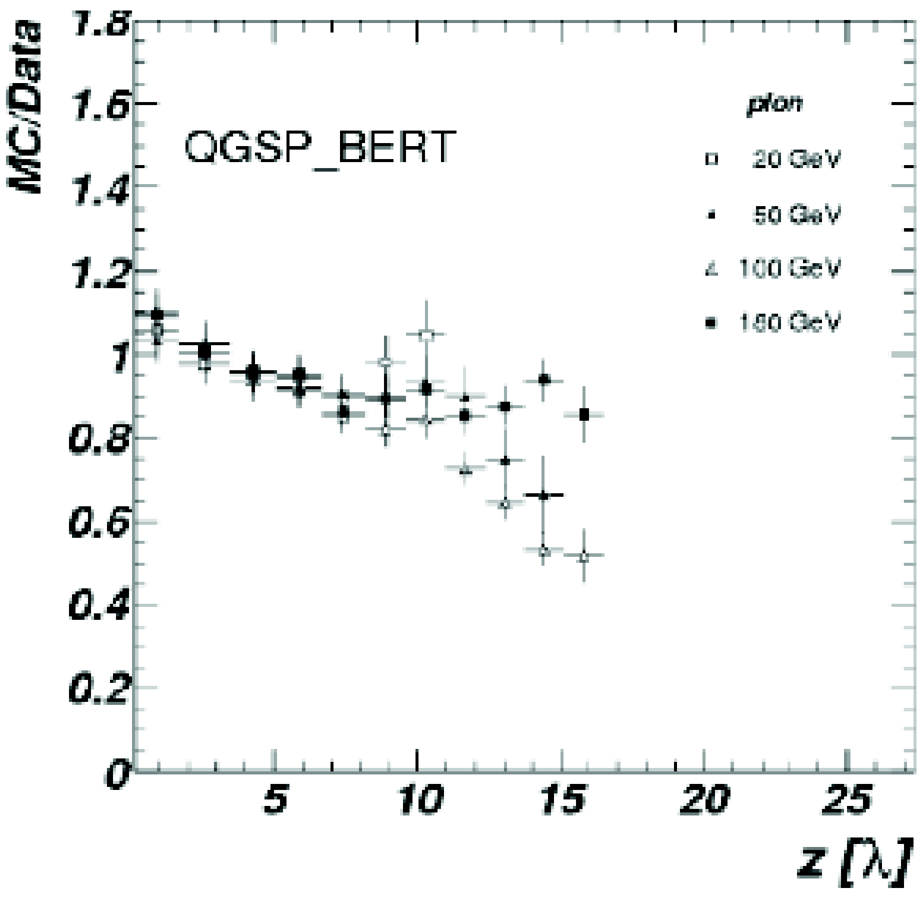
ATLAS HEC test-beam

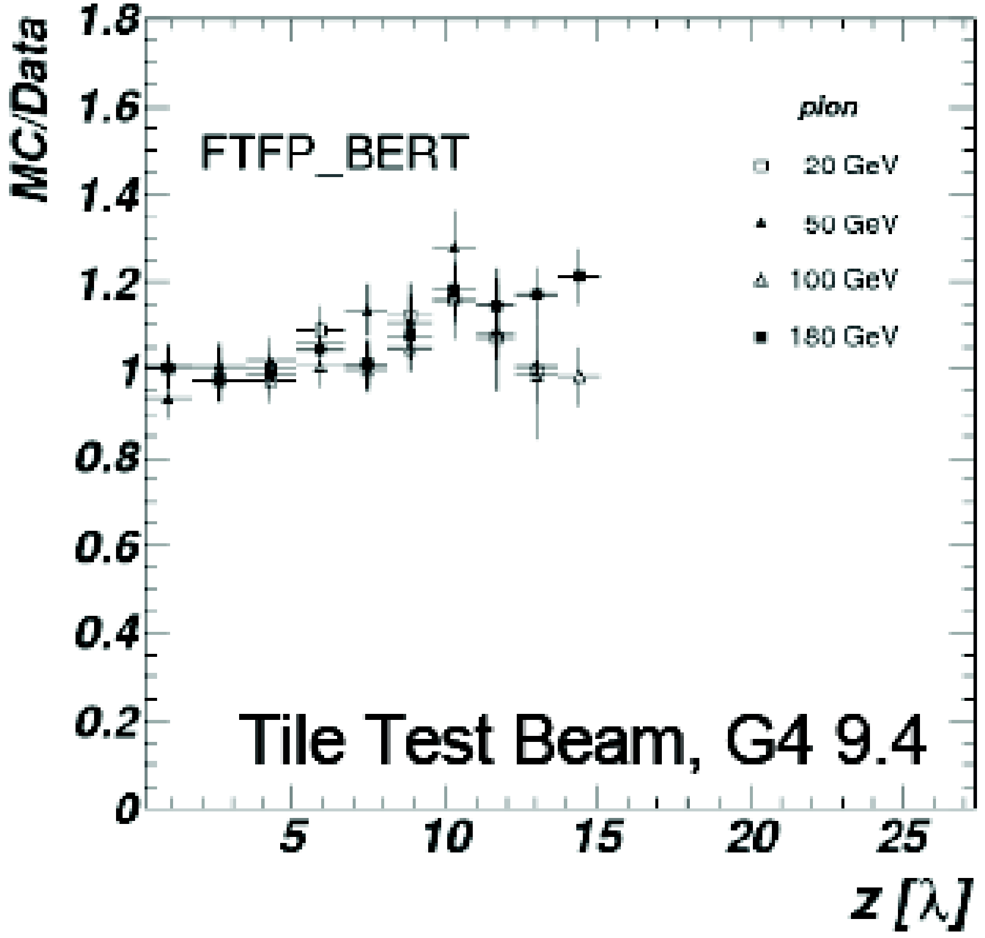


Longitudinal shower shapes

ATLAS TileCal test-beam @90°

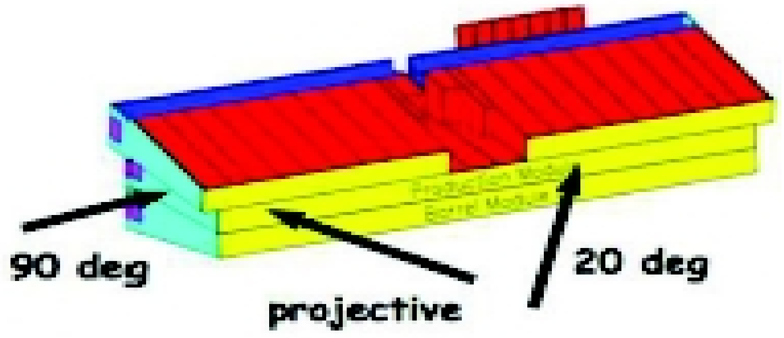


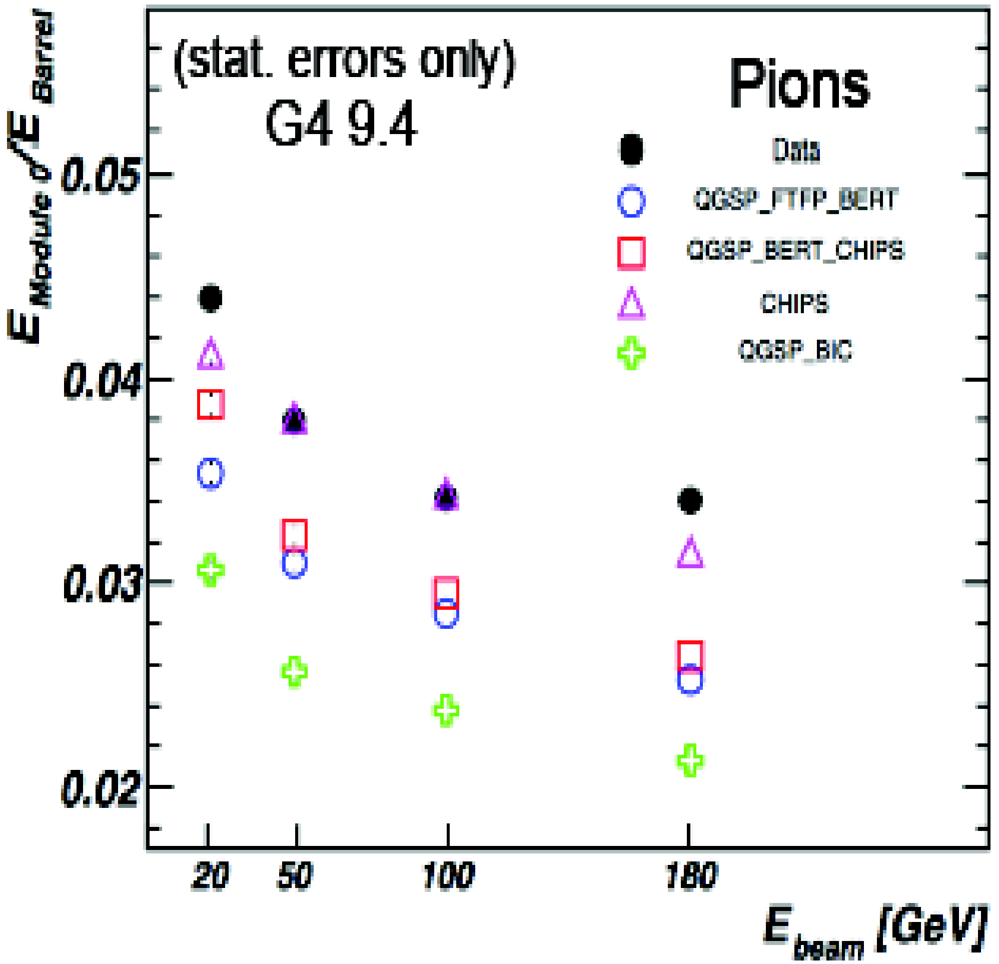


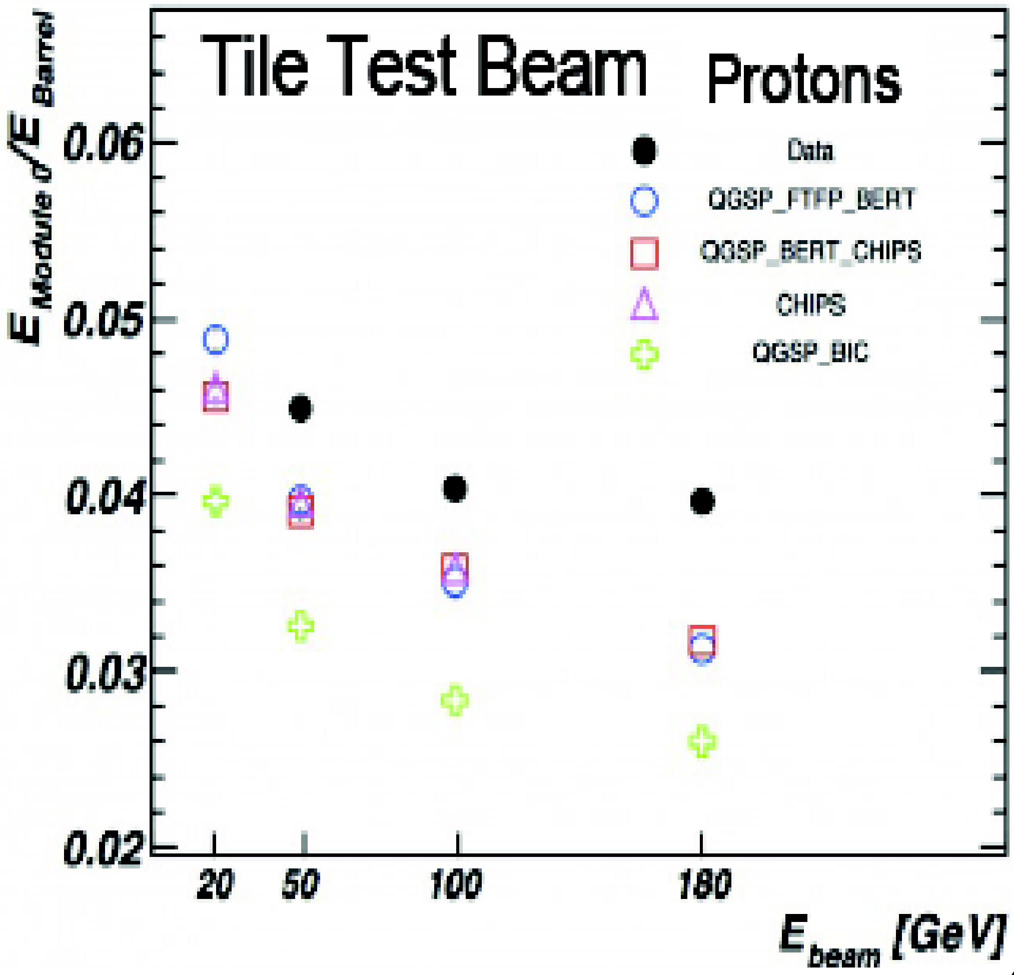


Lateral shower shapes

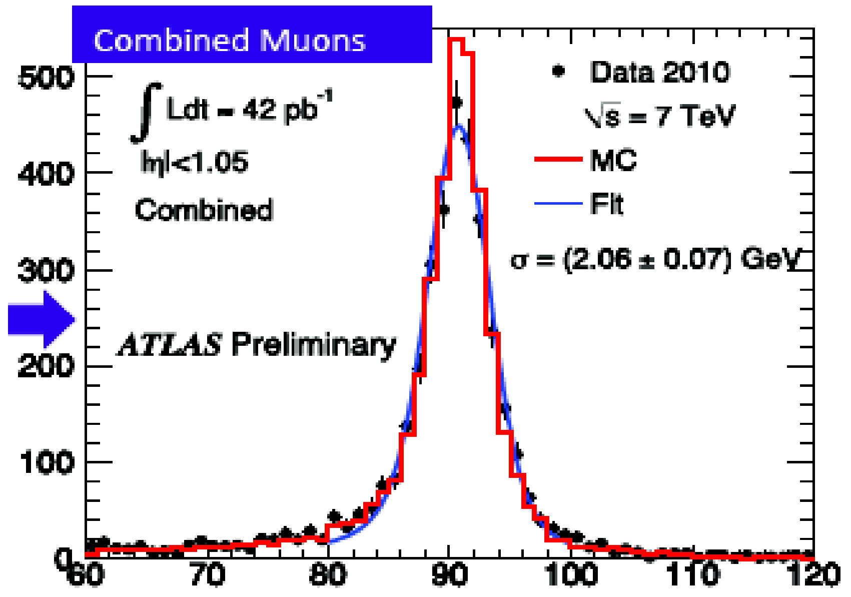
ATLAS TileCal test-beam @90°





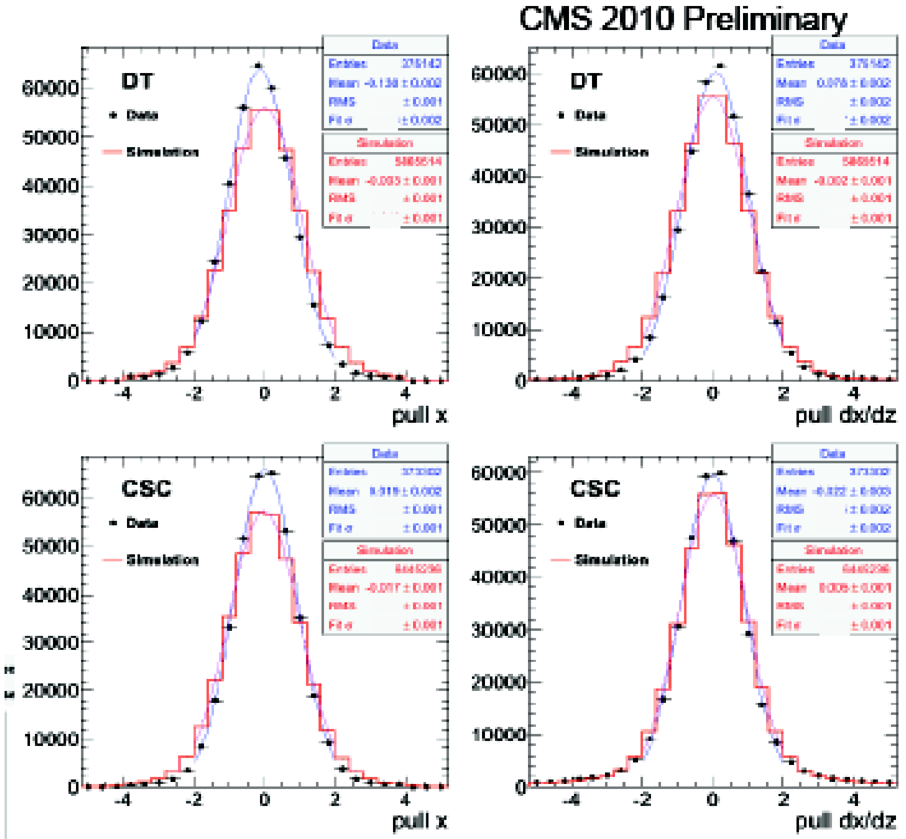


Muon simulation *vs.* p-p collision data



Z mass resolution of combined muons in the barrel, which require both an inner detector and muon spectrometer track

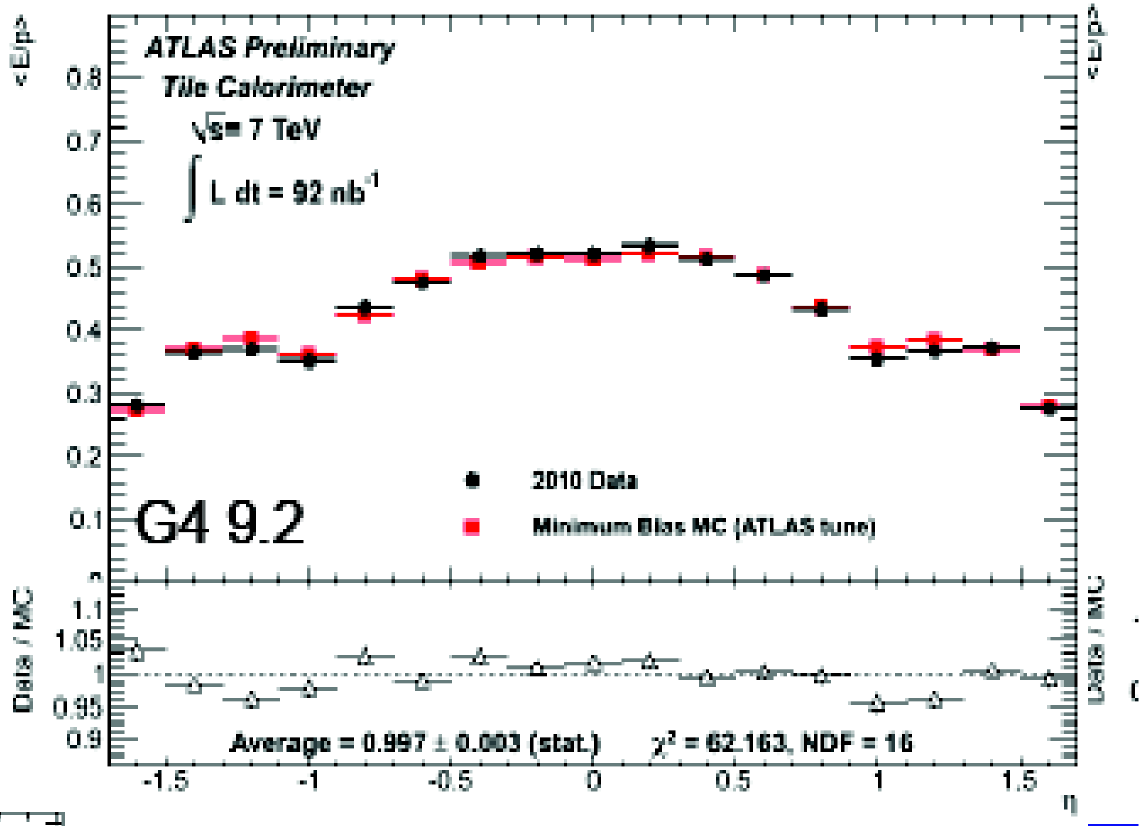
- Muon physics in G4 is extensively tested and validated in the energy range 10 GeV – 10 TeV



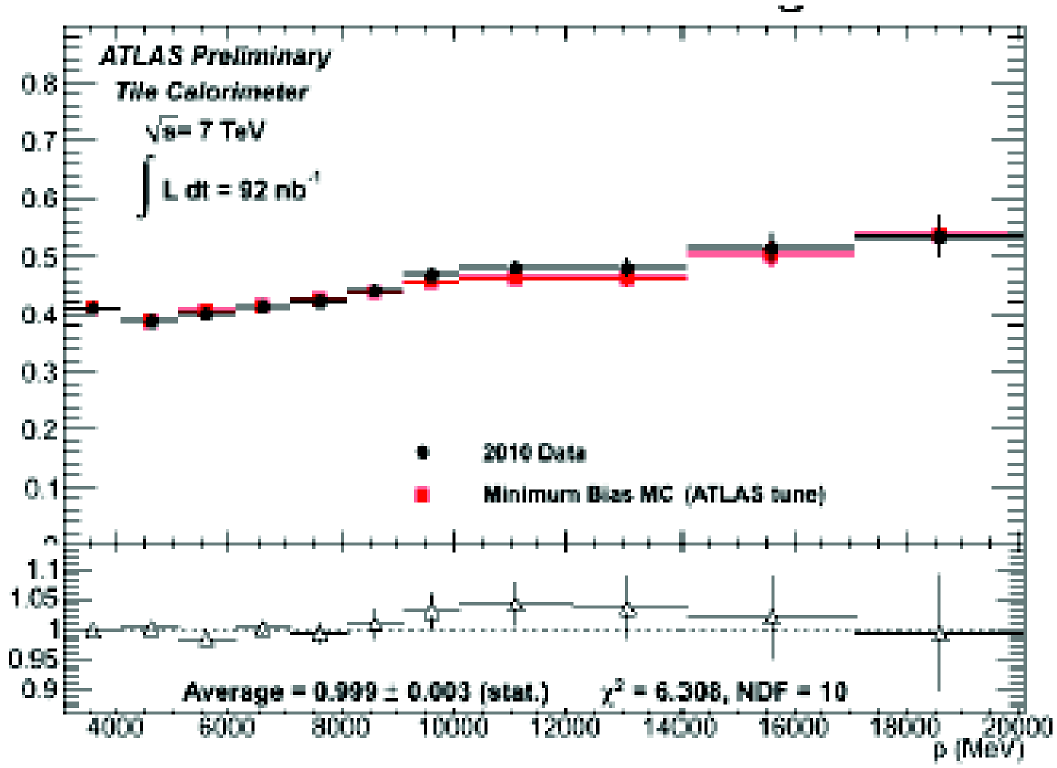
- Resolutions of extrapolations from central tracker to muon segments
  - checks proper implementation of material, multiple scattering through solenoid, absorber

Isolated single hadron response: simulation *vs.* ATLAS p-p data

E/p vs η

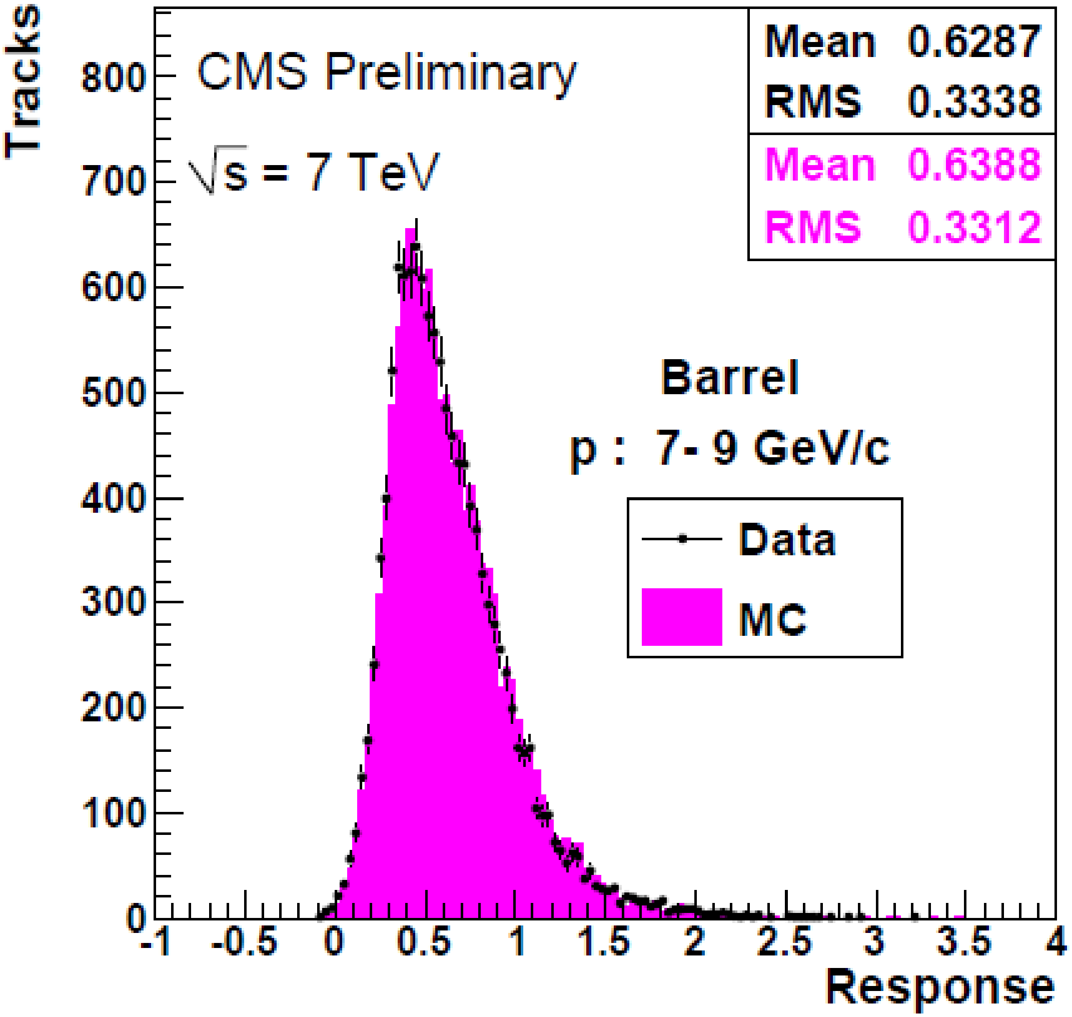


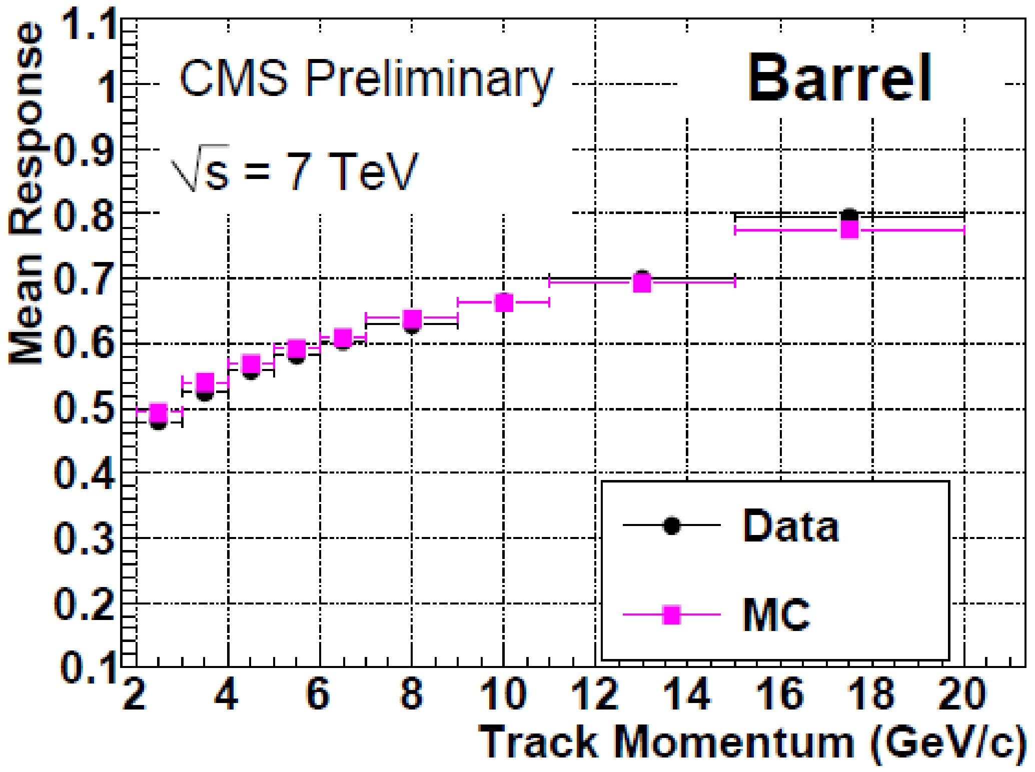
E/p vs p



Isolated single hadron response: simulation *vs.* CMS p-p data

Agreement is better than ±3% between 2-20 GeV/c





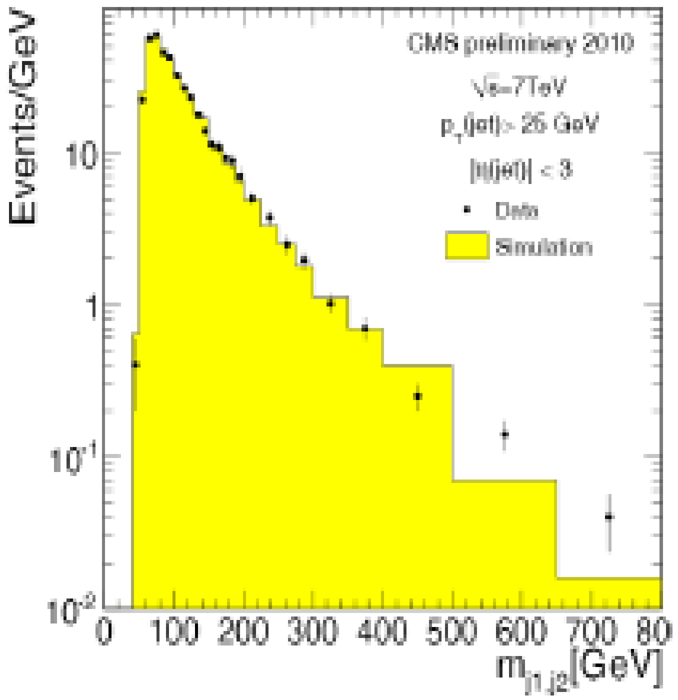
Di-jet invariant mass: simulation *vs.* CMS p-p data

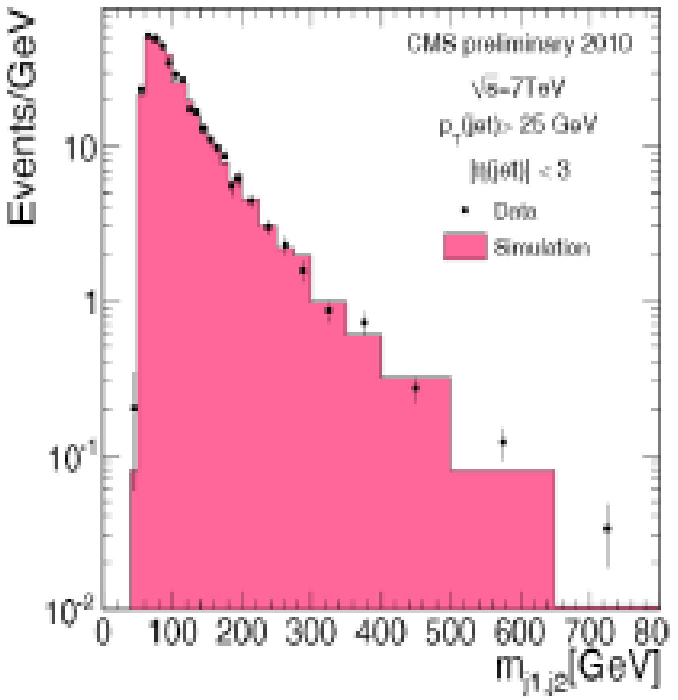
Very good agreement between simulation and collision data!

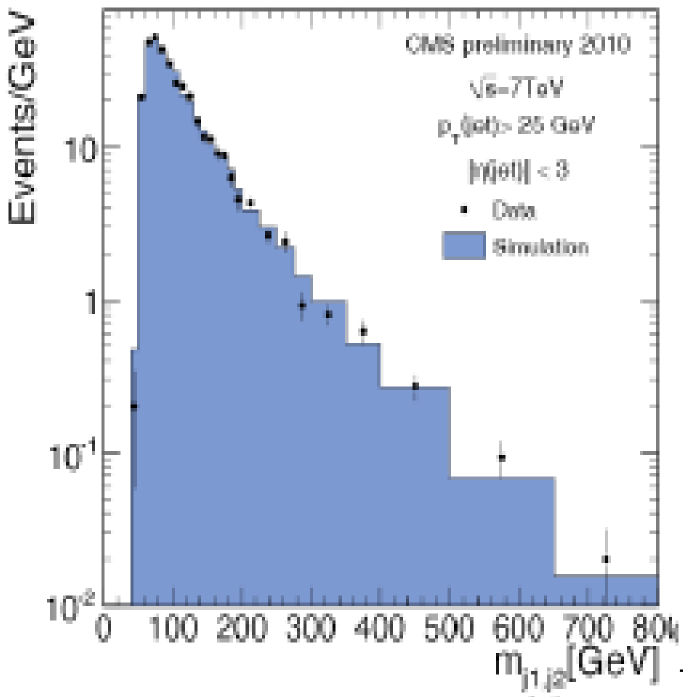
Three ingredients are convoluted in the simulation:

- Monte Carlo event generator: Pythia
- Detector simulation engine: Geant4
- Experiment-specific aspects: geometry/materials, digitization, calibration, rec.

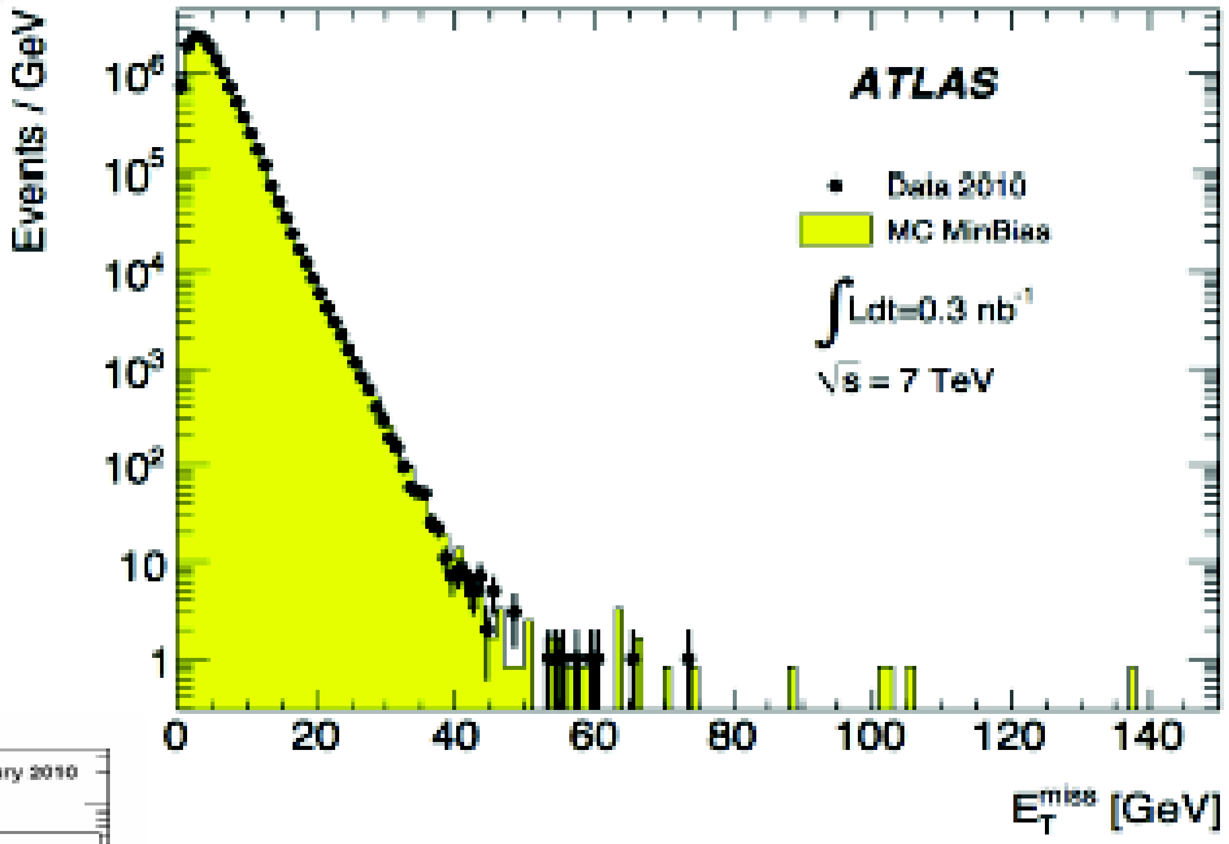
Calo jets **JPT jets** **PF jets**

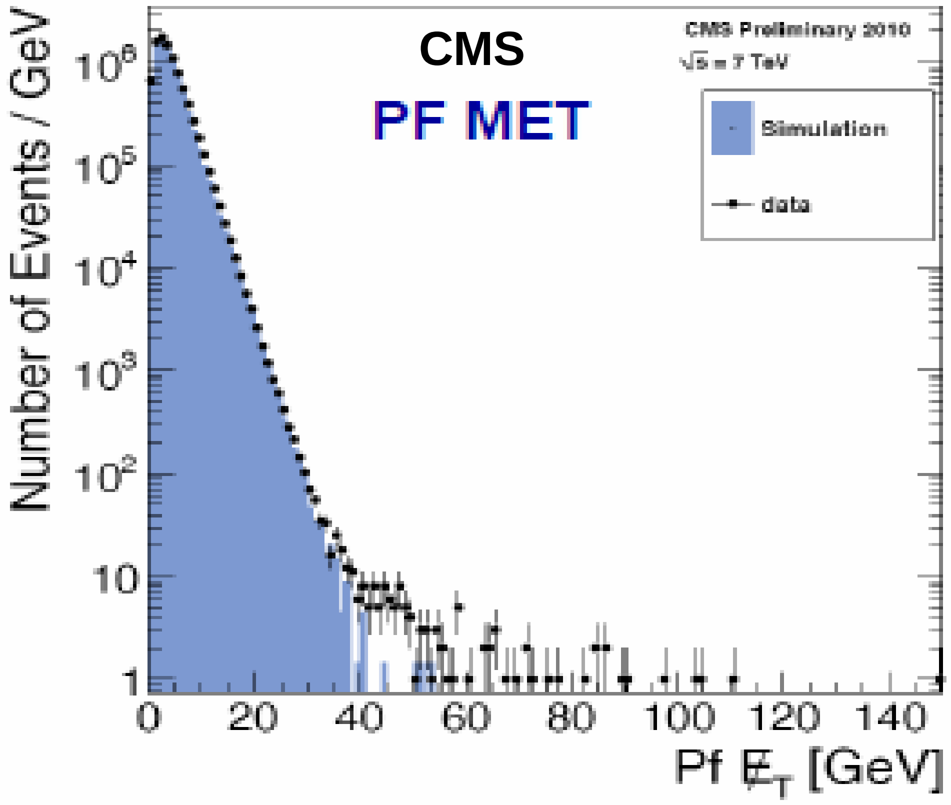






Missing ET : simulation *vs.* collision data





Missing ET is a very complex (global) variable

Good agreement over 6 orders of magnitudes!

Examples

Learning and Working with Geant4 Examples

- Most of the people, **learn** how to use Geant4 by “playing” with an existing example
  - Choose an example as close as possible to your use-case
  - Look at the code, to see how things are done
  - Read the relevant parts of the “*User's Guide: For Application Developers*” to understand better how things work
  - Modify the example to do what you need

- Beside learning: many **real** detector-simulation **applications** originated from a Geant4 example
  - by adapting the detector description, sensitive detectors, hits, primary source, user actions, and analysis

- Bottom line: examples are very useful!

Geant4 Examples

- ***geant4/examples/***
  - ***basic/*** : oriented to novice users and covering the most typical use-cases of Geant4 applications
  - ***extended/*** : covers many specific use-cases; may require some additional libraries besides of G4
  - ***advanced/*** : real and complete applications for different simulation studies; may require additional third-party products to be built

- There are **README** files in each directory which briefly explain the content of each directory...

Geant4 **Basic** Examples

- ***geant4/examples/basic/*** :
  - ***B1/***
    Simple geometry with a few solids.
    Scoring total dose in a selected volume; user action classes.
  - ***B2/***
    Simplified tracker geometry with global constant magnetic field.
    Scoring within tracker via G4 sensitive detector and hits.
  - ***B3/***
    Schematic Positron Emitted Tomography system.
    Radioactive source.
    Scoring within Crystals via G4 scorers.
  - ***B4/***
    Simplified calorimeter with layers of two materials.
    Scoring within layers in four ways: (a) via user actions;
    (b) via user data object; (c) via hits & sensitive detectors; (d) via scorer
  - ***B5/***
    A double-arm spectrometer with wire chambers, hodoscopes and calorimeters with a local constant magnetic field.
    Scoring used in wire chambers;
    G4 sensitive detector and hits used for hodoscopes and calorimeters.

Geant4 **Extended** Examples

- ***geant4/examples/extended*** :

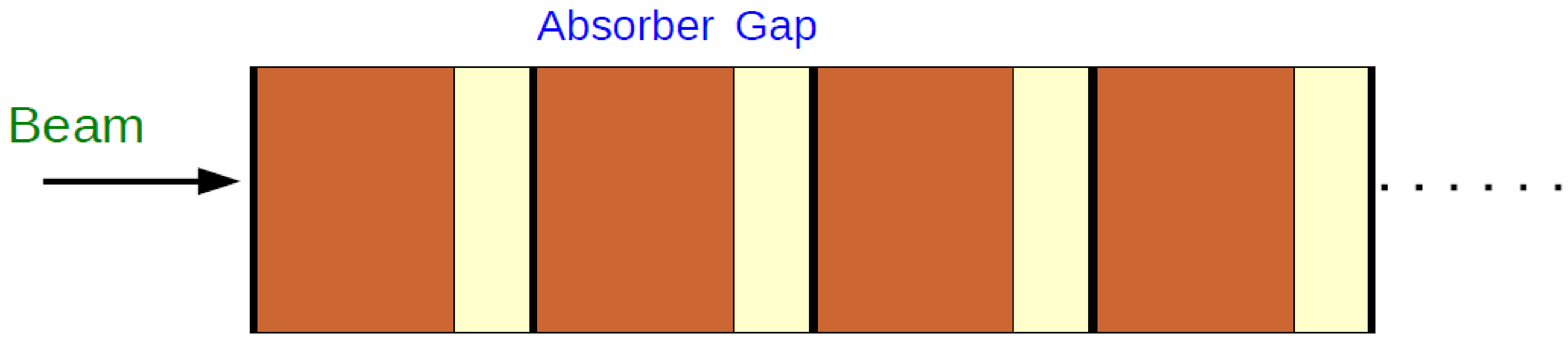
  - ***analysis/***
  - ***biasing/*** : examples of event biasing, scoring and reverse-MC
  - ***common/***
  - ***electromagnetic/*** : many, different things...
  - ***errorpropagation/***
  - ***eventgenerator/*** : examples of G4ParticleGun, G4GeneralParticleSource, HepMC, and Pythia
  - ***exoticphysics/*** : examples of monopoles and phonons
  - ***field/***
  - ***g3tog4/***
  - ***geometry/***
  - ***hadronic/*** : G4PhysListFactory, cross sections, ions, neutron-HP, etc.
  - ***medical/***
  - ***optical/***
  - ***parallel/*** : examples of event-level parallelism
  - ***parameterisations/*** : examples of fast shower parameterisations
  - ***persistency/*** : of geometry (GDML) and simulation output (ROOT I/O)
  - ***polarisation/***
  - ***radioactivedecay/***
  - ***runAndEvent/*** : MC-true, scorers, parallel world, readout geometry
  - ***visualization/***

Geant4 **Advanced** Examples

- ***geant4/examples/advanced*** :
  - *air_shower/*
  - *amsEcal/*
  - *brachytherapy/*
  - *ChargeExchangeMC/*
  - *composite_calorimeter/*
  - *dnageometry/*
  - *dnaphysics/*
  - *eRosita/*
  - *gammaknife/*
  - *gammaray_telescope/*
  - *hadrontherapy/*
  - *human_phantom/*
  - *iort_therapy/*
  - *lAr_calorimeter/*
  - *medical_linac/*
  - *microbeam/*
  - *microdosimetry/*
  - *microelectronics/*
  - *nanobeam/*
  - *purging_magnet/*
  - *radioprotection/*
  - *underground_physics/*
  - *xray_fluorescence/*
  - *xray_telescope/*

A closer look to the basic example **B4**

- ***geant4/examples/basic/B4***
  - a simple sampling calorimeter setup
  - 4 variants of scoring:
    - ***B4a/*** : user actions
    - ***B4b/*** : user data object
    - ***B4c/*** : hits and sensitive detectors
    - ***B4d/*** : scorer



- The **calorimeter** is a box made of a number of layers
- A **layer** consists of an **absorber plate** and of a **detection gap**
- The layer is **replicated**

Content of ***geant4/examples/basic/B4/B4a/***

- **CMakeLists.txt** : to build the example using Cmake (recommended)
- **GNUmakefile** : to build the example with the old GNUmake system (deprecated)
- **exampleB4a.cc** : the main program
- **exampleB4.in** : macro file (there are also others: run1.mac, run2.mac, ... *.mac)
- ***include/*** : header files (.hh) of the example:

  B4DetectorConstruction.hh
  B4aActionInitialization.hh
  B4PrimaryGeneratorAction.hh
  B4RunAction.hh
  B4aEventAction.hh
  B4aSteppingAction.hh

- ***src/*** : source files (.cc) of the example:

  B4DetectorConstruction.cc
  B4aActionInitialization.cc
  B4PrimaryGeneratorAction.cc
  B4RunAction.cc
  B4aEventAction.cc
  B4aSteppingAction.cc

A look into a G4 macro file: **exampleB4.in**

```
# e+ 300MeV
/gun/particle e+
/gun/energy 300 MeV
/run/beamOn 1
#
# list the existing physics processes
/process/list
#
# switch off MultipleScattering
/process/inactivate msc
/run/beamOn 1
#
# switch on MultipleScattering
/process/activate msc
#
# change detector parameter
/gun/particle gamma
/gun/energy 500 MeV
/run/beamOn 1
```

3 runs, each with a different configuration;

1 event for each run

A look into a G4 main program: **exampleB4a.cc**

```
...
int main( int argc, char** argv ) {
 ...
 // Build the detector
 B4DetectorConstruction* detConstruction = new B4DetectorConstruction();
 runManager->SetUserInitialization( detConstruction );

 // Choose the physics list
 G4VModularPhysicsList* physicsList = new FTFP_BERT;
 runManager->SetUserInitialization( physicsList );

 // Instantiate the user actions
 B4aActionInitialization* actionInitialization
                        = new B4aActionInitialization( detConstruction );
 runManager->SetUserInitialization( actionInitialization );

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

}
```

B4DetectorConstruction (1)

```
int main( int argc, char** argv ) {
  ...
  // Build the detector
  B4DetectorConstruction* detConstruction = new B4DetectorConstruction();
  runManager->SetUserInitialization( detConstruction );

  // Choose the physics list
  G4VModularPhysicsList* physicsList = new FTFP_BERT;
  runManager->SetUserInitialization( physicsList );
  // Instantiate the primary generator and the user actions
  B4aActionInitialization* actionInitialization
                = new B4aActionInitialization( detConstruction );
  runManager->SetUserInitialization( actionInitialization );
  // Initialize G4 kernel
  runManager->Initialize()
  ...
}
```

```
G4VPhysicalVolume*
B4DetectorConstruction::Construct() {
  DefineMaterials();
  return DefineVolumes();
}

void B4DetectorConstruction::DefineMaterials() {
  // Lead material defined using NIST Manager
  G4NistManager* nistManager = G4NistManager::Instance();
  nistManager->FindOrBuildMaterial( "G4_Pb" );

  // Liquid argon material
  G4double a;  // mass of a mole
  G4double z;  // number of protons
  G4double density;
  new G4Material( "liquidArgon", z=18., a= 39.95*g/mole, density= 1.390*g/cm3 );

  // Vacuum
  new G4Material( "Galactic", z=1., a=1.01*g/mole,density= universe_mean_density,
                           kStateGas, 2.73*kelvin, 3.e-18*pascal );

  // Print materials
  G4cout << *( G4Material::GetMaterialTable() ) << G4endl;
}
```

B4DetectorConstruction (2)

```
G4VPhysicalVolume* B4DetectorConstruction::DefineVolumes() {
  ...
  // --- World ---
  G4VSolid* worldS  = new G4Box( "World",          // its name
                             worldSizeXY/2, worldSizeXY/2, worldSizeZ/2 ); // its size
  G4LogicalVolume* worldLV = new G4LogicalVolume( worldS,              // its solid
                                                  defaultMaterial,   // its material
                                                  "World" );           // its name
  G4VPhysicalVolume* worldPV = new G4PVPlacement( 0,                        // no rotation
                                                   G4ThreeVector(),  // at (0,0,0)
                                                   worldLV,      // its logical volume
                                                   "World",      // its name
                                                   0,              // its mother  volume
                                                   false,         // no boolean operation
                                                   0,             // copy number
                                                   fCheckOverlaps );

 //  --- Calorimeter ---
 G4VSolid* calorimeterS = new G4Box( "Calorimeter", calorSizeXY/2, calorSizeXY/2,
                                                         calorThickness/2 );
 G4LogicalVolume* calorLV = new G4LogicalVolume(  calorimeterS, defaultMaterial,
                                                         "Calorimeter" );
 new G4PVPlacement( 0, G4ThreeVector(), calorLV, "Calorimeter", worldLV, false, 0,
                    fcheckOverlaps );
```

CONTINUE...

B4DetectorConstruction (3)

```
  // --- Layer ---
  G4VSolid* layerS  = new G4Box( "Layer", calorSizeXY/2, calorSizeXY/2,
                                          layerThickness/2 );
  G4LogicalVolume* layerLV = new G4LogicalVolume( layerS, defaultMaterial, "Layer" );
  new G4PVReplica( "Layer",                // its name
                   layerLV,                // its logical volume
                   calorLV,                // its mother
                   kZAxis,                 // axis of replication
                   nofLayers,              // number of replica
                   layerThickness );  // witdth of replica
   // --- Absorber ---
  G4VSolid* absorberS = new G4Box( "Abso", calorSizeXY/2, calorSizeXY/2,
                                           absoThickness/2 );
  G4LogicalVolume* absorberLV = new G4LogicalVolume( absorberS, absorberMaterial,
                                                     "Abso" );
  fAbsorberPV = new G4PVPlacement( 0, G4ThreeVector( 0., 0., -gapThickness/2 ),
                                   absorberLV, "Abso", layerLV, false, 0,
                                   fCheckOverlaps );
   // --- Gap ---
  G4VSolid* gapS = new G4Box( "Gap", calorSizeXY/2, calorSizeXY/2, gapThickness/2 );
  G4LogicalVolume* gapLV = new G4LogicalVolume( gapS, gapMaterial, "Gap" );
  fGapPV = new G4PVPlacement( 0, G4ThreeVector( 0., 0., absoThickness/2 ), gapLV, "Gap",
                              layerLV, false, 0, fCheckOverlaps );
  ...
  return worldPV;
}
```

B4aActionInitialization

```
int main( int argc, char** argv ) {
  ...
  // Build the detector
  B4DetectorConstruction* detConstruction = new
B4DetectorConstruction();
  runManager->SetUserInitialization( detConstruction );
  // Choose the physics list
  G4VModularPhysicsList* physicsList = new FTFP_BERT;
  runManager->SetUserInitialization( physicsList );

  // Instantiate the primary generator and the user actions
  B4aActionInitialization* actionInitialization
              = new B4aActionInitialization(detConstruction);
  runManager->SetUserInitialization( actionInitialization );

  // Initialize G4 kernel
  runManager->Initialize()
  ...
}
```

```
void B4aActionInitialization::Build() const {

   SetUserAction( new B4PrimaryGeneratorAction );

   SetUserAction( new B4RunAction );

   B4aEventAction* eventAction = new B4aEventAction;
   SetUserAction( eventAction );

   SetUserAction( new B4aSteppingAction( fDetConstruction, eventAction ) );

}
```

In this example, 2 user actions are not used:
- Tracking action
- Stacking action

B4PrimaryGeneratorAction

```
void B4aActionInitialization::Build() const {
  SetUserAction( new B4PrimaryGeneratorAction );

  SetUserAction( new B4RunAction );

  B4aEventAction* eventAction = new B4aEventAction;
  SetUserAction( eventAction );

  SetUserAction( new B4aSteppingAction( fDetConstruction,
eventAction ) );
}
```

```
B4PrimaryGeneratorAction::B4PrimaryGeneratorAction() ... {
  G4int nofParticles = 1;
  fParticleGun = new G4ParticleGun( nofParticles );
  // default particle kinematic
  G4ParticleDefinition* particleDefinition
           = G4ParticleTable::GetParticleTable()->FindParticle( "e-" );
  fParticleGun->SetParticleDefinition( particleDefinition );
  fParticleGun->SetParticleMomentumDirection( G4ThreeVector(0., 0., 1. ) );
  fParticleGun->SetParticleEnergy( 50.*MeV );
}


void B4PrimaryGeneratorAction::GeneratePrimaries( G4Event* anEvent ) {
  // This function is called at the begining of event
  ...
 // Set gun position
 fParticleGun->SetParticlePosition( G4ThreeVector( 0., 0., -worldZHalfLength ) );
 fParticleGun->GeneratePrimaryVertex( anEvent );

}
```

B4RunAction

```
void B4aActionInitialization::Build() const {
  SetUserAction( new B4PrimaryGeneratorAction );

  SetUserAction( new B4RunAction );

  B4aEventAction* eventAction = new B4aEventAction;
  SetUserAction( eventAction );

  SetUserAction( new B4aSteppingAction( fDetConstruction, eventAction ) );
}
```

```
B4RunAction::B4RunAction() ... {
   ...
   G4AnalysisManager* analysisManager = G4AnalysisManager::Instance();
   // Book histograms, ntuple
   analysisManager->CreateH1(...);
   analysisManager->CreateNtuple(...);
   ...
}

void B4RunAction::BeginOfRunAction(...) {
  ...
  G4AnalysisManager* analysisManager = G4AnalysisManager::Instance();
  // Open an output file
  analysisManager->OpenFile( fileName );
}

void B4RunAction::EndOfRunAction(...) {
  // Print something
  ...
 // Save histograms & ntuple
 analysisManager->Write();
 analysisManager->CloseFile();
}
```

B4aEventAction

```
B4aEventAction::B4aEventAction( ) : ... fEnergyAbs(0.) , fEnergyGap( 0. ) ... {}

void B4aEventAction::BeginOfEventAction(...) {
   // initialisation per event
   fEnergyAbs = 0. ;  fEnergyGap = 0.;
   ...
}

void B4aEventAction::AddAbs( G4double de, G4double dl ) {
   fEnergyAbs += de;
   ...
}

void B4aEventAction::AddGap( G4double de, G4double dl ) {
  fEnergyGap += de;
  ...
}

void B4aEventAction::EndOfEventAction( const G4Event* event ) {
   // Accumulate statistics: fill histograms and ntuples
   G4AnalysisManager* analysisManager = G4AnalysisManager::Instance();
   analysisManager->FillH1( 2, fEnergyAbs );
   ...
   analysisManager->FillNtupleDColumn( 1, fEnergyAbs );
   ...
   // Print per-event information
   ...
}
```

```
void B4aActionInitialization::Build() const {
  SetUserAction( new B4PrimaryGeneratorAction );

  SetUserAction( new B4RunAction );

  B4aEventAction* eventAction = new B4aEventAction;
  SetUserAction( eventAction );

  SetUserAction( new B4aSteppingAction( fDetConstruction, eventAction ) );
}
```

B4aSteppingAction

```
void B4aActionInitialization::Build() const {
  SetUserAction( new B4PrimaryGeneratorAction );

  SetUserAction( new B4RunAction );

  B4aEventAction* eventAction = new B4aEventAction;
  SetUserAction( eventAction );

  SetUserAction( new B4aSteppingAction( fDetConstruction, eventAction ) );
}
```

```
void B4aSteppingAction::
UserSteppingAction( const G4Step* step ) {
  // Collect energy and track length step by step

  // Get volume of the current step
  G4VPhysicalVolume* volume
    = step->GetPreStepPoint()->GetTouchableHandle()->GetVolume();

  // Get energy deposit
  G4double edep = step->GetTotalEnergyDeposit();

  // Get step length
  G4double stepLength = 0.;
  if ( step->GetTrack()->GetDefinition()->GetPDGCharge() != 0. ) {
    stepLength = step->GetStepLength();
  }

  if ( volume == fDetConstruction->GetAbsorberPV() ) {
    fEventAction->AddAbs( edep, stepLength );
  }
  if ( volume == fDetConstruction->GetGapPV() ) {
    fEventAction->AddGap( edep, stepLength );
  }
}
```

Backup

Optical Photons (1)

- A **photon** is considered to be optical when its wavelength is greater than the typical inter-atomic distance
- In Geant4, optical photons are treated as a separated particle class, **G4OpticalPhoton**, distinct from the class of high-energy photons, **G4Gamma**
  - This allows to incorporate some of the wave-like properties of electromagnetic radiation into the optical photon processes
  - But there is no smooth transition as a function of energy between optical photons and gammas
- Optical photons in G4 must always have linear polarization
  - This is guaranteed by the processes that can create them
  - For primary particle, the linear polarization should be set by the user: in the case of an unpolarized source, the linear polarization should be sampled randomly for each new primary photon

Optical Photons (2)

- Three processes in Geant4 can produce optical photons: Cerenkov effect, scintillation, and transition radiation
  - optical photons are generated without energy conservation
- Geant4 processes that can be associated to optical photons:
  - refraction and reflection at medium boundaries
  - bulk absorption
  - Mie and Rayleigh scattering
  - wavelength shifting (WLS)
- Optical properties of media:
  - the processes associated to optical photons require some optical properties of media: reflectivity, transmission efficiency, dielectric constants, surface properties
  - these properties are stored as entries in G4MaterialPropertiesTable which is linked to G4Material

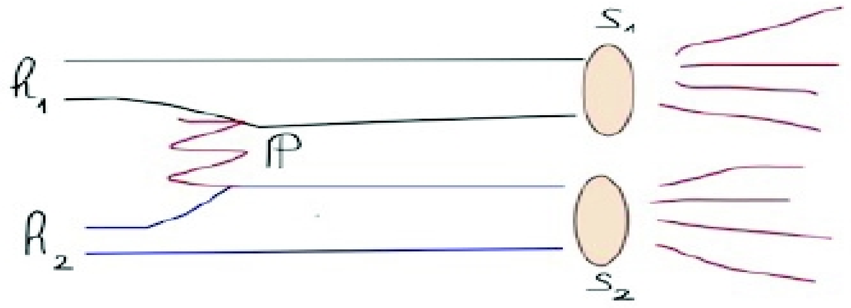
Optical Photons (3)

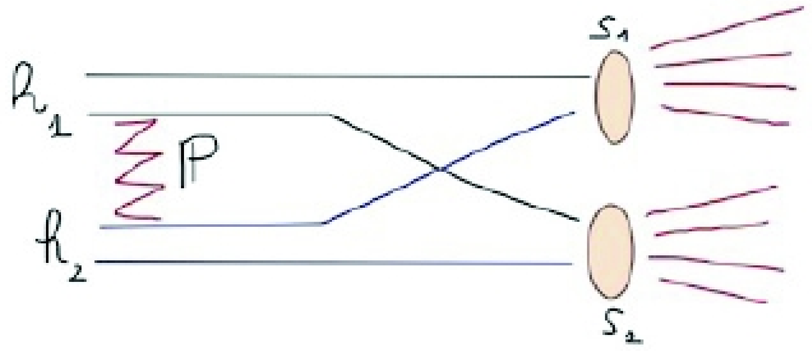
- For some examples, see: ***examples/extended/optical/***
  - ***OpNovice/*** : simulation of optical photons generation and transport
  - ***LXe/*** : scintillation inside a bulk scintillator with PMT
  - ***wls/*** : propagation of photons inside a Wave Length Shifting (WLS) fiber

- In practice, optical photons are not used frequently
  - Substantial slow down of the simulation
  - Very difficult to model accurately all properties (and imperfections!) of real, experimental set-ups
  - Easier to treat the effects of optical photons at the level of “digitization”, i.e. applying a factor (determined from data) to the deposited energy obtained from Geant4
  - Most of the physics lists do not include optical photons, and their related processes, by default

Fritiof (FTF) model (1)

- High-energy string model valid for any hadron projectile with kinetic energy between ~ 3 GeV and ~ 1 TeV
- Selection of collision partners: projectile, nucleon
- Splitting of nucleons into quarks and diquarks
- Formation and excitation of strings (between constituents)
- String hadronization/fragmentation
  - Insert q-q pair , u : d : s : qq = 1 : 1 : 0.27 : 0.1
  - At break: new string plus hadron
    - gets $P_{||}$ from sampling fragmentation functions
    - gets P_T from sampling a Gaussian





FTF (2)

- Build up 3D model of nucleus
- Calculate impact parameter with all nucleons
  - Calculate hadron-nucleon collision probabilities
  - Multiple collisions are allowed
  - Use gaussian density distributions for hadrons and nucleons
- Sample number of strings exchanged in each collision
- String formation and then fragmentation into hadrons
- Remnant nucleus
  - After the HE interaction is performed an excited nucleus remains
  - De-excitation via precompound model
- Under developing/testing: re-scattering
  - Use Binary Cascade or Bertini to propagate string fragments in nuclear media

Bertini-like intra-nuclear cascade model (BERT)

- The Geant4 implementation started as the original Bertini (1960), but it was then extended and modified significantly
- Valid for p, n , π, K, hyperons with E_{kin} ~< 10 GeV
- The incident hadron penetrates the nucleus, and propagates in a density-dependent nuclear potential
- All hadron-nucleon interactions are based on free-space cross sections, angular distributions, etc., but the Pauli exclusion principle is taken into account
- Each secondary is propagated in the nuclear potential until re-interacts or leaves the nucleus
- Particle-hole excitons are created during the cascade

Preco: pre-equilibrium

Native pre-equilibrium de-excitation model in Geant4 is a version of the standard exciton model. Key ingredients:

- Internal transition rates:
  - CEM (Cascade Exciton Model): default
  - Blann-Machner's parameterization
- Emission rates:
  – Nucleon emission in standard exciton formulation
  – Complex particle emission (d , t , 3He , α) from CEM

The pre-equilibrium phase continues until:

number of excitons ≤ number of excitons in equilibrium

then transition to equilibrium

Preco: equilibrium

Five processes are considered:

Alternates:

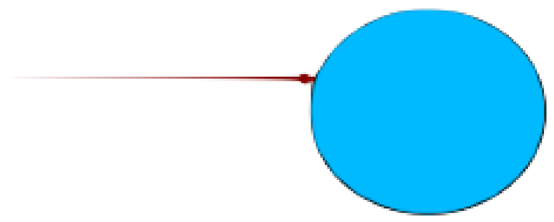
- Fermi Breakup, for $Z < 9$ and $A < 17$ (Botvina et al)
- Statistical Multifragmentation, for $E^*/A > 3$ MeV (Botvina et al)

Competitors:

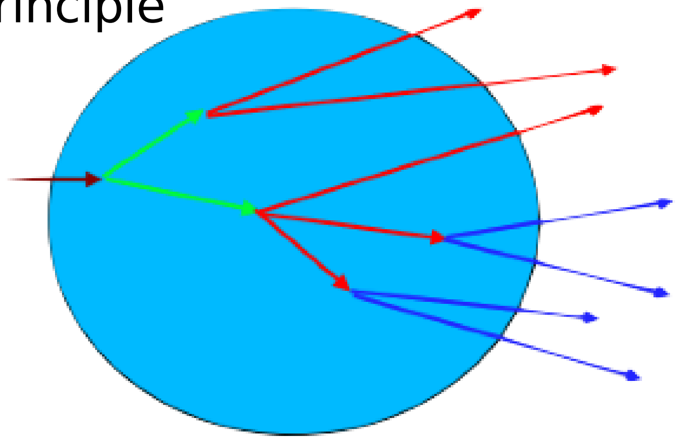
- Fission (Bohr-Wheeler model + Amelin prescript.)
- Particle Evaporation :
  - Evaporation Model WE (Weisskopf-Ewing) (evaporation of: n , p , d , t , 3He , α)
  - Generalized Evaporation Model GEM (Furihata) (heavier ejected fragments: $Z < 13$ and $A < 29$)
- Photon Evaporation :
  - Discrete (tabulated E1, M1, E2)
  - Continuum (GDR strength)

BERT + Preco (1)

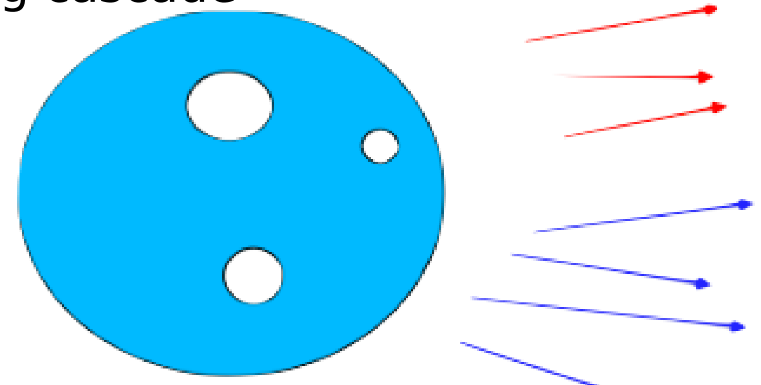
Hadron penetrates nucleus
Nucleus: density-dependent potential



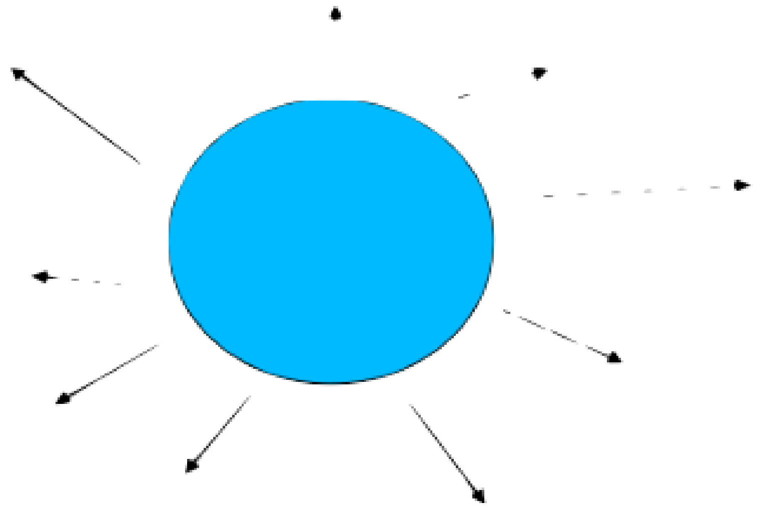
Free-space σ and angular distributions
Pauli principle



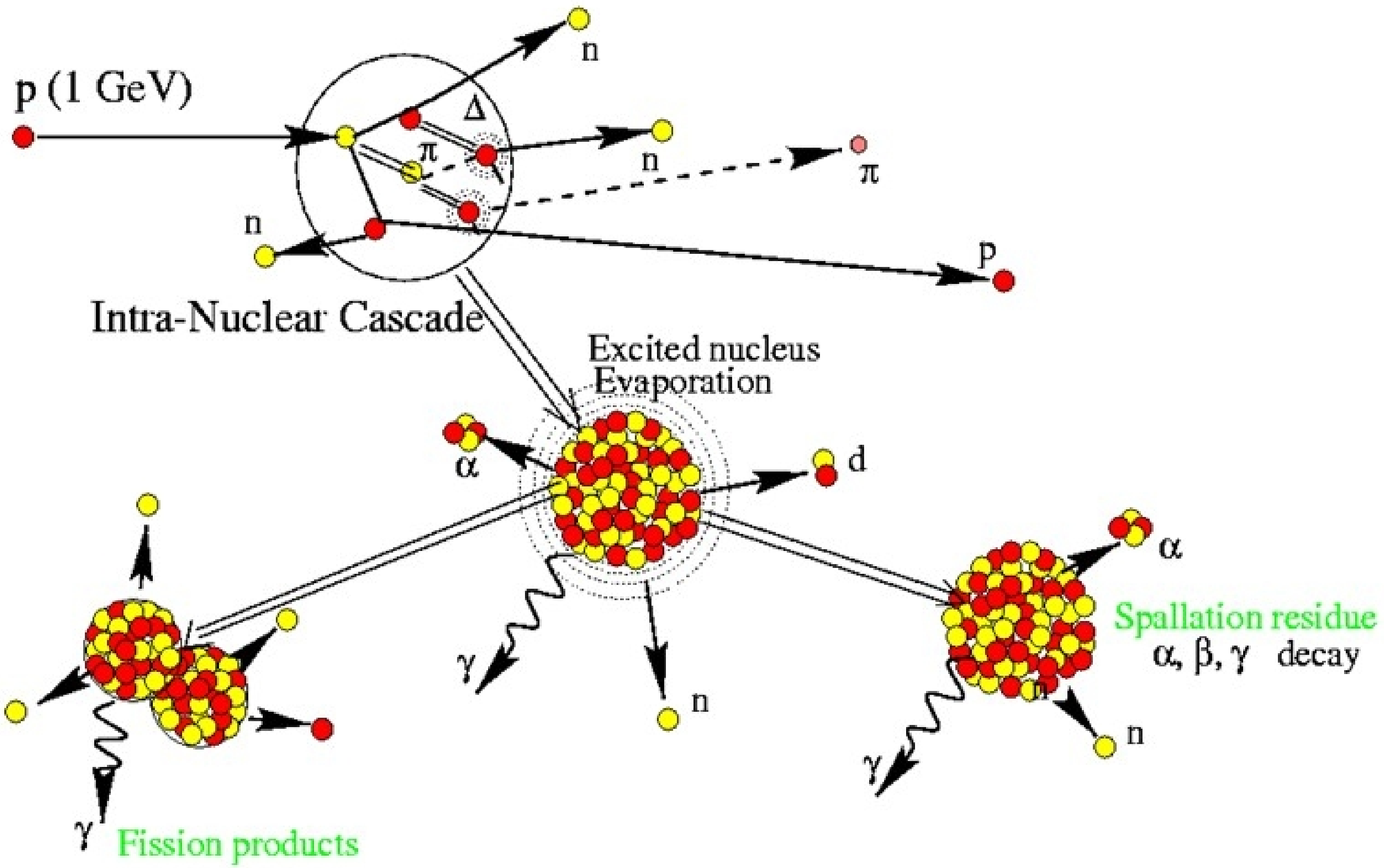
Particle-hole excitations are created
during cascade



Nucleus is de-excited



BERT + Preco (2)


p (1 GeV)
n
Δ
π
n
π
n
p
Intra-Nuclear Cascade
Excited nucleus
Evaporation
α
d
α
Spallation residue
α, β, γ decay
γ
n
γ
n
γ
Fission products

Component-based approach to physics

Geant4 takes a component-based approach to physics:

- Provide many physics components (processes) which are decoupled from one another
- User selects these components in custom-designed physics lists in much the same way as a detector geometry is built
- Exceptions:
  - A few electromagnetic processes must be used together
  - Future processes involving interference of electromagnetic and strong interactions may require coupling as well

G4VUserPhysicsList

- All physics lists must derive from this class
  - and then be registered with the Run Manager
- Example:

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

- User must implement the methods: **ConstructParticle** , **ConstructProcess** , and **SetCuts**

ConstructParticle()

- Choose the particles you need in your simulation and define all of them here. Example:

```
void MyPhysicsList::ConstructParticle() {
  G4Electron::ElectronDefinition();
  G4Positron::PositronDefinition();
  G4Gamma::GammaDefinition();
  G4Proton::ProtonDefinition();
  G4Neutron::NeutronDefinition();
  …
}
```

- It is possible to use Geant4 classes that create group of particles (instead of individual ones):
G4BosonConstructor , G4LeptonConstructor , G4MesonConstructor , G4BaryonConstructor , G4IonConstructor

ConstructProcess()

For each particle defined in ConstructParticle() assign all the physics processes that you want to consider in your simulation

```
void MyPhysicsList::ConstructProcess() {

    AddTransportation();
    // method provided by G4VUserPhysicsList , assign transportation process
    // to all particles defined in ConstructParticle()

    ConstructEM();
    // convenience method that user may define; put electromagnetic physics here

    ConstructGeneral();
    // convenience method that user may define;
}
```

ConstructEM()

```
void MyPhysicsList::ConstructEM() {
  theParticleIterator->reset();
  while ( (*theParticleIterator)() ) {
    G4ParticleDefinition* particle = theParticleIterator->value();
    G4ProcessManager* pmanager =
                                     particle->GetProcessManager();
    G4String particleName = particle->GetParticleName();

    if ( particleName == “gamma” ) {
      pmanager->AddDiscreteProcess(
                         new G4GammaConversion()  );
      …
    }
    …
  }
}
```

ConstructGeneral()

```
void MyPhysicsList::ConstructGeneral() {
   G4Decay* theDecayProcess = new G4Decay();
   theParticleIterator->reset();
   while ( (*theParticleIterator)() ) {
      G4ParticleDefinition* particle = theParticleIterator->value();
      G4ProcessManager* pmanager =
                                            particle->GetProcessManager();
      if ( theDecayProcess->IsApplicable( *particle ) ) {
         pmanager->AddProcess( theDecayProcess );
         pmanager->SetProcessOrdering( theDecayProcess,
                                                        idxPostStep );
         pmanager->SetProcessOrdering( theDecayProcess,
                                                        idxAtRest );
      }
   }
}
```

SetCuts()

```
void MyPhysicsList::SetCuts() {
   defaultCutValue = 1.0*mm;

   SetCutValue( defaultCutValue, “gamma” );
   SetCutValue( defaultCutValue, “e-” );
   SetCutValue( defaultCutValue, “e+” );
}
```

A simple G4VModularPhysicsList

- Constructor

```
MyModPhysList::MyModPhysList() : G4VModularPhysicsList() {
   defaultCutValue = 1.0*mm;
   RegisterPhysics( new ProtonPhysics() );
   // all physics processes having to do with protons

   RegisterPhysics( new ElectronPhysics() );
   // all physics processes having to do with electrons

   RegisterPhysics( new DecayPhysics() );
   // decay of unstable particles
}
```

- SetCuts

```
void MyModPhysList::SetCuts() {
   SetCutsWithDefault();
}
```

G4VModularPhysicsList

Use G4ModularPhysicsList to build a realistic physics list which would be too long, complicated and hard to maintain with the previous approach.
Its features are:

- Derived from **G4VUserPhysicsList**
- AddTransportation() automatically called for all registered particles
- Allows to define “physics modules”:
  - electromagnetic physics
  - hadronic physics
  - decay physics
  - ion physics
  - radioactive physics
  - optical physics
  - etc.

Physics Constructors

Allow you to group particle and process construction according to physics domains

```
class ProtonPhysics : public G4VPhysicsConstructor {
  public:
      ProtonPhysics( const G4String& name = “proton” );
      virtual ~ProtonPhysics();

      virtual void ConstructParticle();
      // easy – only one particle to build in this case

      virtual void ConstructProcess();
      // put here all the processes a proton can have
}
```