Hadronic Physics

Alberto Ribon
CERN PH/SFT

Geant4 Tutorial, CERN, 26-27 March 2019

Outline

- Reminder:
  - Hadronic Models, Cross Sections, Framework, Physics Lists
- NeutronHP (ParticleHP)
- Radioactive Decay
- Biasing in Hadronic Physics
- *Exercises*

Reminder:
Hadronic Models , Cross Sections , Framework , Physics Lists

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*

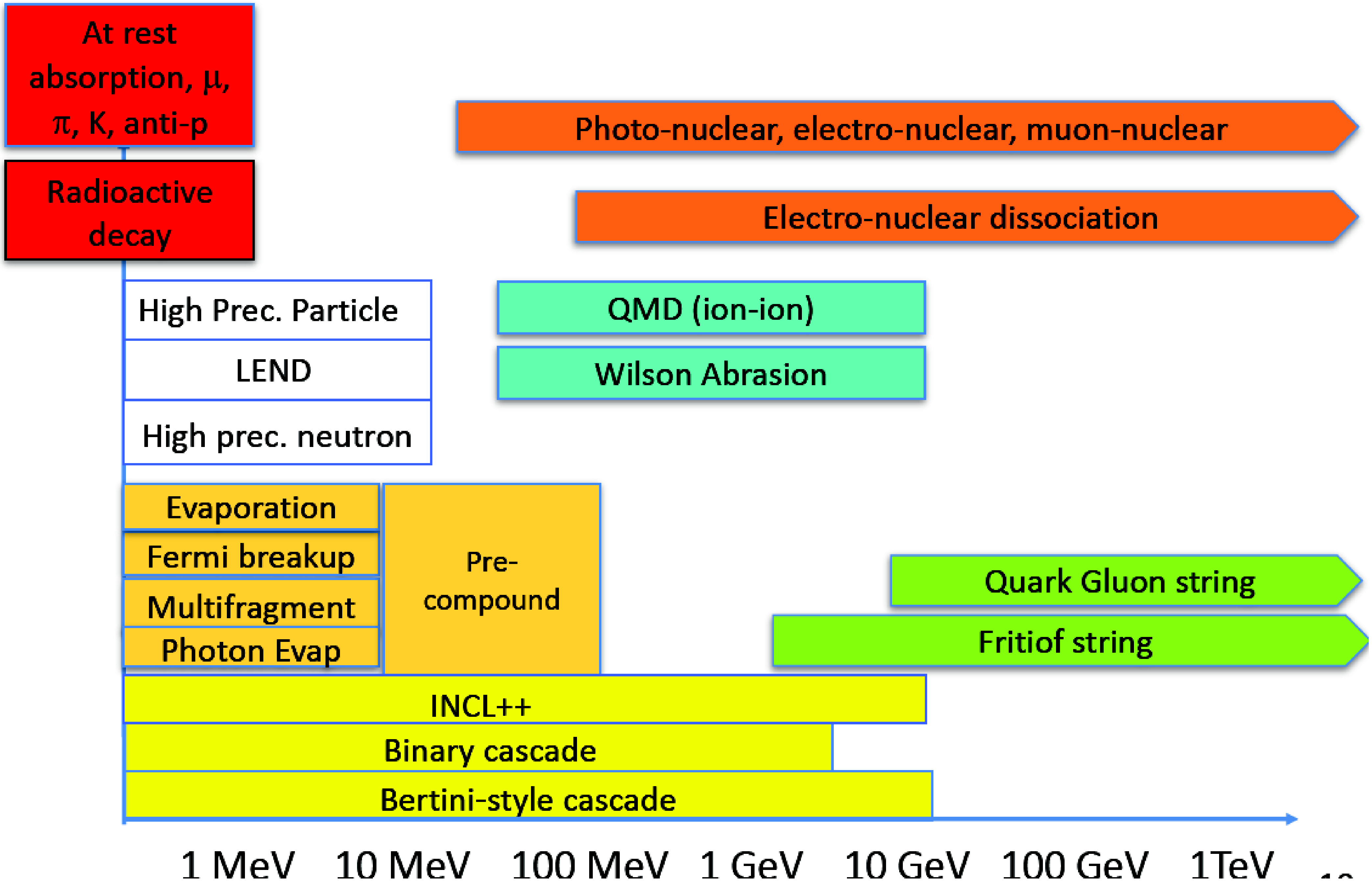
  in our detector simulations
- In principle, QCD is the theory that describes all hadronic interactions; in practice, perturbative calculations are applicable only in a tiny (but important!) phase-space region
  - the hard scattering at high transverse momentum

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

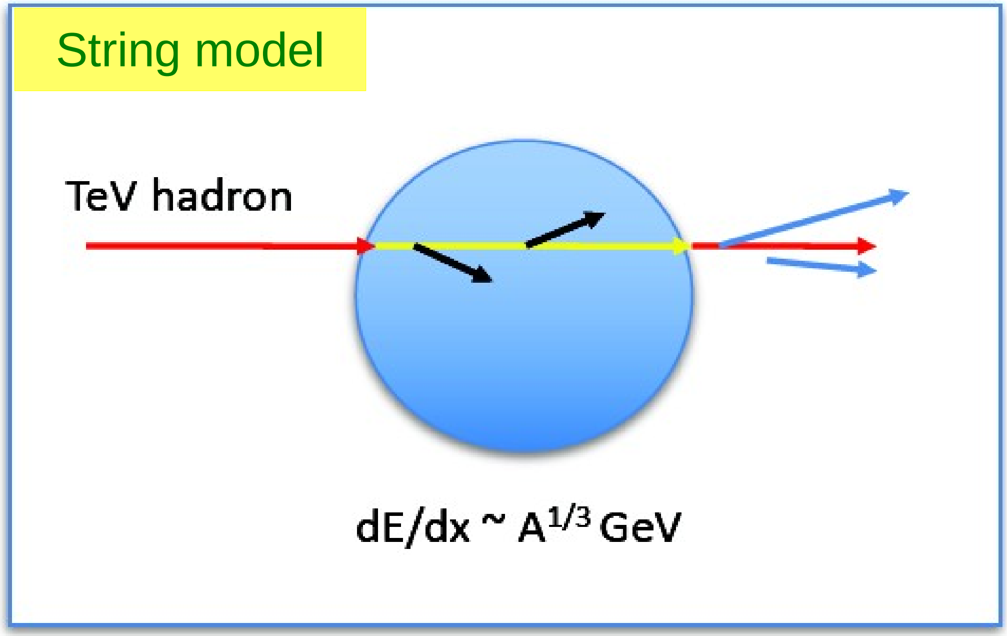
  only approximated models are available
- Hadronic models are valid for limited combinations of

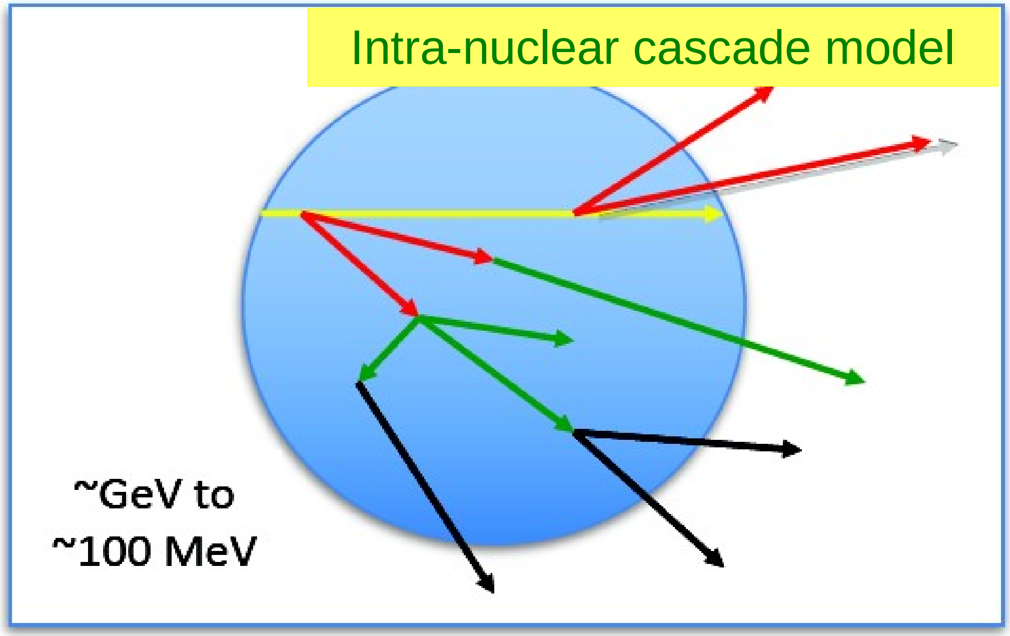
  particle type – energy – target material

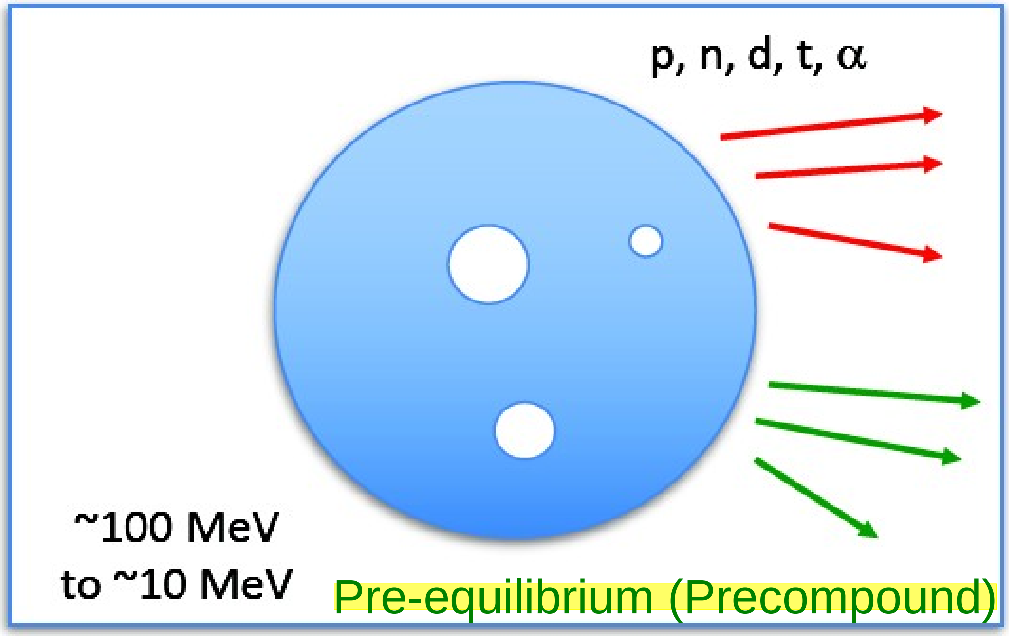
Partial Hadronic Model Inventory

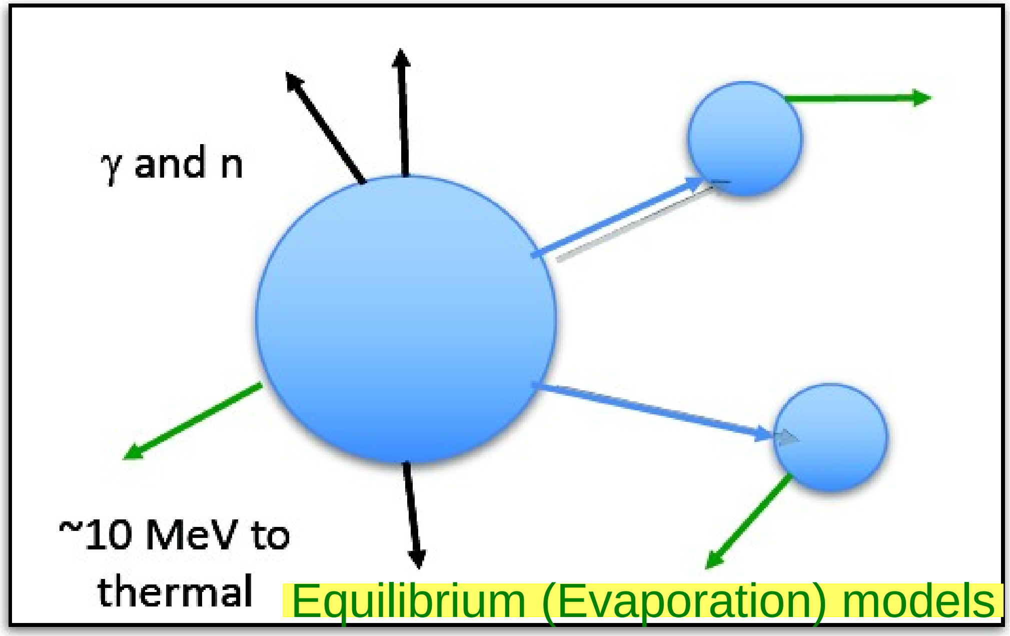


Hadronic Interactions from TeV to meV





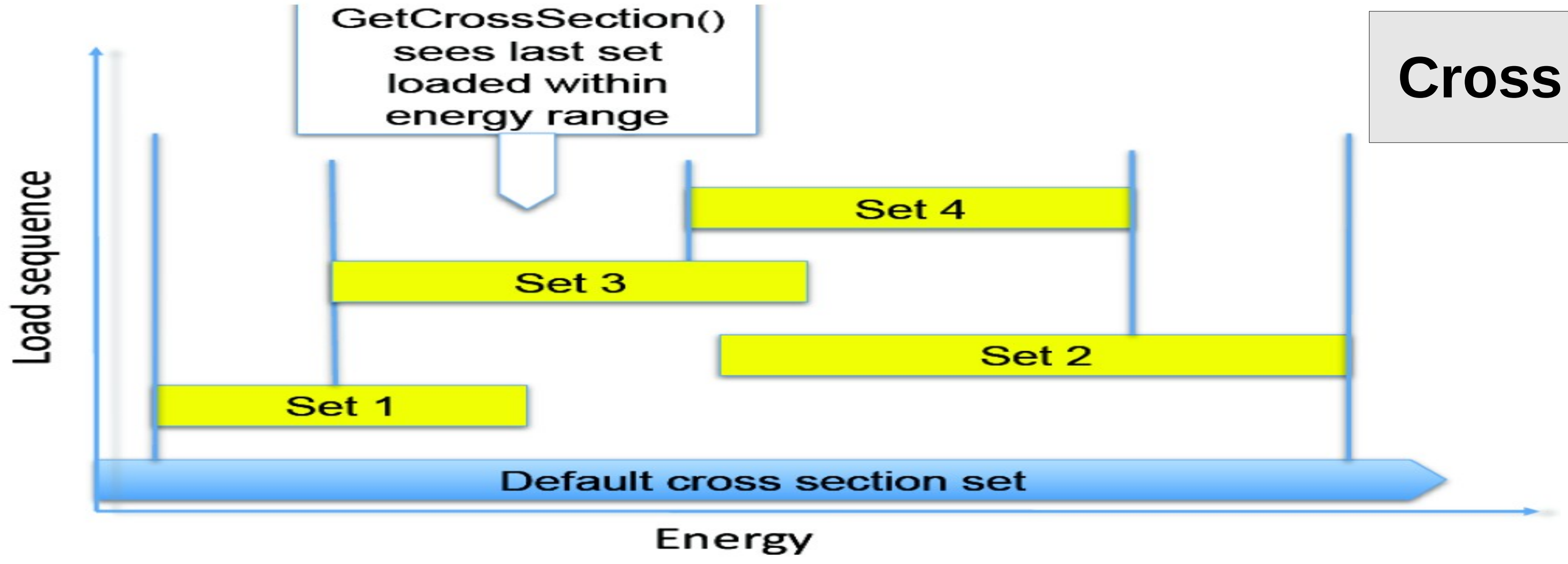




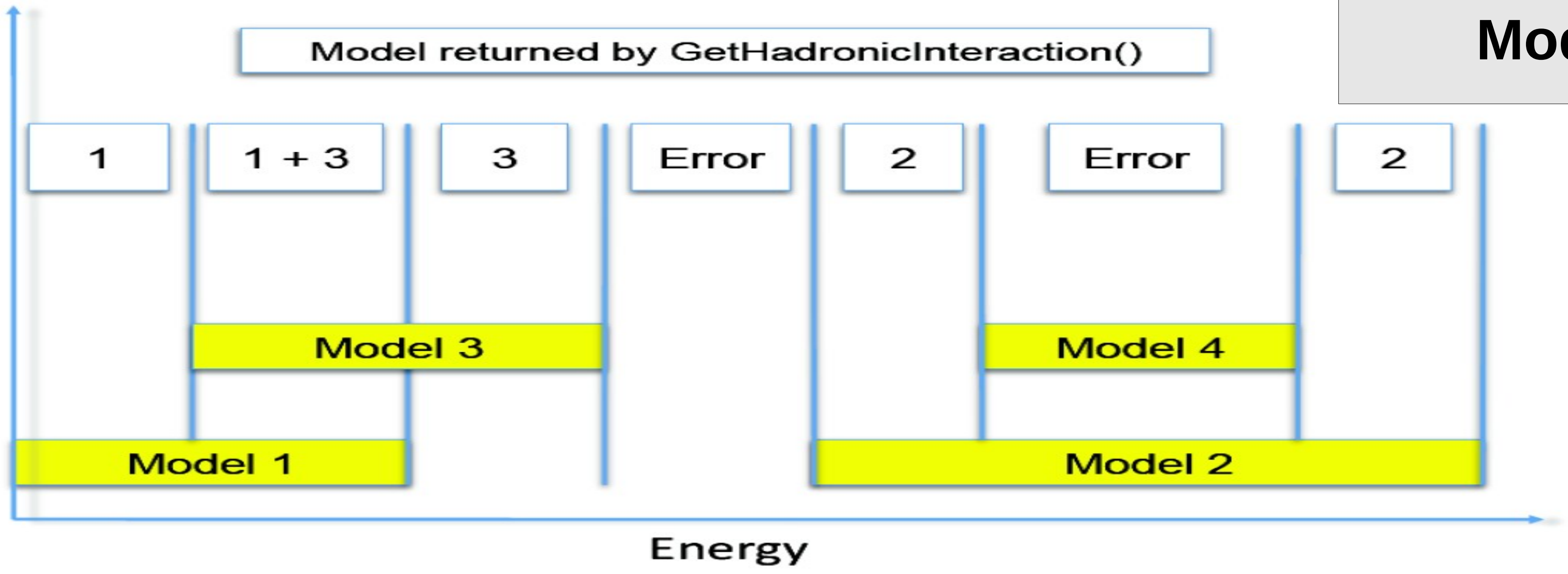
Hadronic Cross Sections and (Final-State) Models

- In Geant4, there is a clear separation between **cross sections** – related to the probability of an elastic or inelastic hadron-nucleus interaction, and therefore to the length that a hadron projectile flies in a material before interacting – and **final-state models** – related to the number, type and properties of the secondaries produced by the interaction
- For each combination of projectile – energy – target
  - ≥ 1 cross sections must be specified in a physics list :
    the first available is used
  - 1 or 2 (final-state) models must be specified in a physics list :
    if two, a random number is thrown to decide which of the two models to use
    - linear probability as a function of the energy, over an interval called **transition region**, defined arbitrarily to get smooth observables

Hadronic Framework


GetCrossSection() sees last set loaded within energy range
Load sequence
Set 4
Set 3
Set 2
Set 1
Default cross section set
Energy

Cross Sections



Models

FTFP_BERT

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

- **FTF** (Fritiof string) model, used above 3 GeV
- **BERT** (Bertini cascade) model, used below 12 GeV
- Nucleus de-excitation: **P**recompound + evaporation
- Neutron capture
- Nuclear capture of negatively charged hadrons at rest
- Gamma- and electron-nuclear interactions
- Hadron elastic
- Standard electromagnetic physics

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

A few other Physics Lists

- FTFP_BERT_**HP** : as FTFP_BERT, but with **NeutronHP** for neutrons of kinetic energy below 20 MeV
  - **Shielding** : similar to FTFP_BERT_HP, but with **Radioactive Decay** and **QMD** (Quantum Molecular Dynamics) for ions
    - QMD used in the range [100 MeV, 10 GeV] : below BIC, above FTFP
- FTFP_**INCLXX** : similar to FTFP_BERT, but using **INCL**XX instead of BERT for some particles
  - Protons, neutrons, charged pions below 20 GeV; FTFP above 15 GeV
- **QGSP**_FTFP_BERT : similar to FTFP_BERT, but using **QGS** (Quark Gluon String) model at high energies
  - [6, 8] GeV transition BERT − FTFP ; [12, 25] GeV transition FTFP − QGSP
- QGSP_**BIC** : similar to FTFP_BERT but using QGS and BIC (Binary Cascade) instead of FTF and BERT when possible
  - Protons, neutrons : BIC < 9.9 GeV , FTFP in [9.5, 25] GeV , QGSP > 12 GeV
    Pions & kaons : BERT < 5 GeV , FTFP in [4, 25] GeV , QGSP > 12 GeV

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 );
  ...
}
```

- It can be “extended”, e.g. adding radioactive decay :

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

NeutronHP (ParticleHP)

An interesting complication: Neutrons

- Neutrons are abundantly produced
  - Mostly “soft” neutrons, produced by the de-excitation 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**

NeutronHP

- **H**igh **P**recision treatment of low-energy neutrons
  - **E**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
    - **G4NDL4.5**
    - Include both cross sections and final states
    - Based on the **ENDF/B-VII** database
    - 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_(All)**HP** , Shielding(LEND)

Notes about NeutronHP (1/2)

- Because of several reasons (binned look-up data tables, inclusive - incomplete, without correlations – information, *etc.*) there will always be small energy non-conservations
  - For all types of interactions (elastic, capture, fission and inelastic)
  - To avoid that, by default Geant4 uses some “tricks” (e.g. emitting some gammas) to conserve energy-momentum. This, however, can affect the average values, so for applications (like nuclear reactors) which care about energy conservations on average, the following enviromental variable should be set to avoid any “adjustment”: **G4NEUTRONHP_DO_NOT_ADJUST_FINAL_STATE**

- Doppler broadening of the resonances, due to target thermal motion, is calculated on-the-fly (from T = 0 K values)
  - Very CPU intense: for those applications that do not need it, it can be switched off by setting the environmental variable **G4NEUTRONHP_NEGLECT_DOPPLER**

Notes about NeutronHP (2/2)

- Geant4 Neutron Data Libraries:
  - Data files for element heavier than Uranium are omitted from public release but can be provided under request
    - Only for peaceful applications
  - Alternative neutron data libraries for Geant4 are available from IAEA (***https://www-nds.iaea.org/geant4/***)
    - Based on JEFF, JENDL, CENDL and BROND (instead of ENDF) neutron data libraries

Thermal Scattering

- For handling elastic scattering at thermal energies **< 4 eV** from chemically bound atoms
  - At thermal neutron energies, atomic translational motion as well as vibration and rotation of the chemically bound atoms affect the neutron scattering cross section and the energy and angular distribution of secondary neutrons
  - Based on the **S(α , β)** model
  - Thermal neutron scattering files from ENDF/B-VII thermal data
    - There are ~ **20 materials** : al_metal, be_metal, be_beo, benzen, d_heavy_water, d_ortho_d2, d_para_d2, fe_metal, graphite, h_l_ch4, h_ortho_h2, h_para_h2, h_polyethylene, h_s_ch4, h_water, h_zrh , o_beo, o_uo2, u_uo2, zr_zrh
  - Can be activated with the elastic constructor **G4ThermalNeutrons**
    - *physicsList → RegisterPhysics(* ***new G4ThermalNeutrons(*** *0* ***)*** *);*
  - Example:

examples/extended/hadronic/Hadr04

ParticleHP

- Extension of NeutronHP for : **p , d , t , 3He , α**
  - For high-precision elastic and inelastic interactions below **200 MeV**
    - Of interest for medical and nuclear physics
  - Also data-driven, based on the **TENDL** database
    - Based on TALYS code
    - Optional database that can be downloaded from the Geant4 web site
      - **G4TENDL1.3.2**
    - Need to be pointed by the environmental variable **G4PARTICLEHPDATA**
  - The two codes, NeutronHP and ParticleHP, have been merged
  - Validation in progress, good comparisons so far with MCNP
  - Available in the **QGSP_BIC_AllHP** reference physics list

GEANT4
A SIMULATION TOOLKIT

Radioactive Decay

Nuclides in Geant4 (1/2)

- Based on data files from the Evaluated Nuclear Structure Data Files (**ENSDF**), Geant4 knows about **~ 6'500 nuclides** with half life **> 1 ns**
  - ~3'000 ground states + ~3'500 meta-stable states (isomers)
  - As of Geant4 10.5 , their properties (Z , A , E , τ , etc.) are in **G4ENSDFSTATE2.2** (pointed by G4ENSDFSTATEDATA)

- Two ways to have unstable nuclides in Geant4 :

1. Radioactive source as a primary particle

  – e.g. Na24m :

```
/gun/particle ion
/gun/ion 11 24 0 472. keV
```

Nuclides in Geant4 (2/2)

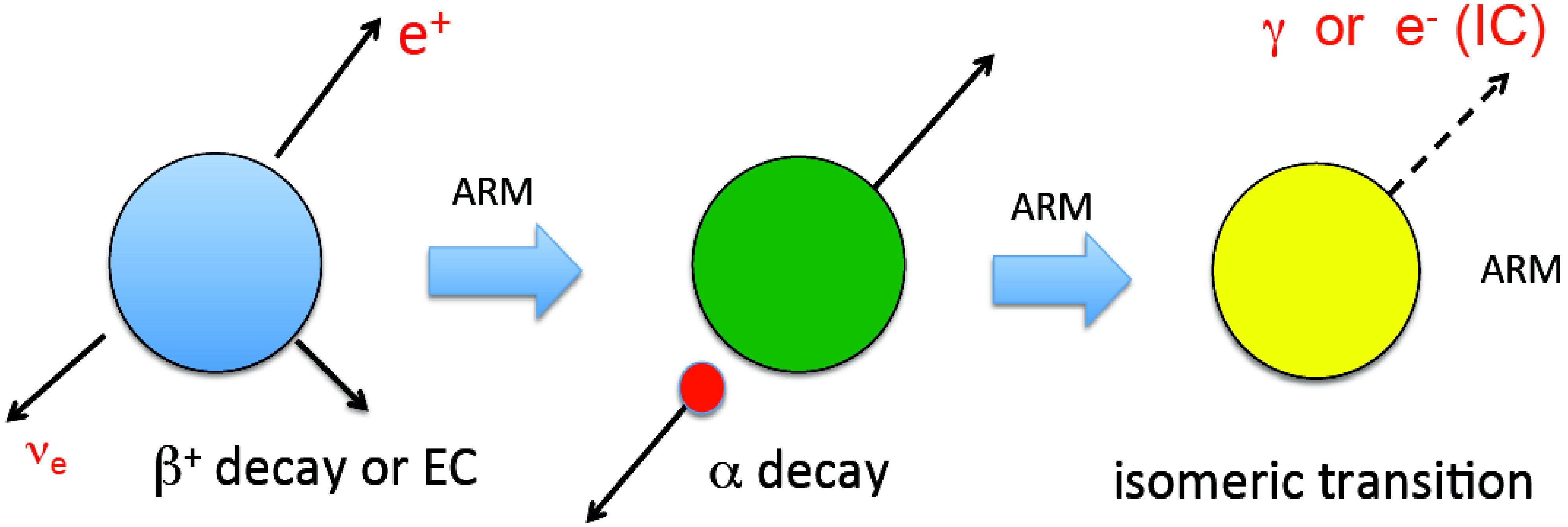
2. Induced radioactivity (activation) :
   in hadron – nucleus and nucleus – nucleus reactions, the de-excitation nuclear models can create nuclides with lifetime greater than a threshold:
   - **1000 seconds** by default, when Radioactive Decay is **not** used
     - Very few nuclides, to avoid CPU overhead for HEP applications
     - These nuclides are treated as “stable” (because RDM is not used)
   - **1 microsecond** by default, when Radioactive Decay **is** used

Radioactive Decay

- Process to simulate radioactive decay of (unstable) nuclei, both in flight and at rest
- So far implemented the following types of decay :
  α , **β-**, **β+** , **γ** (i.e. isomeric transitions, and Internal Conversions (IC))
  EC (Electron Capture) , **p** , **n**

- Empirical and data-driven
  - Data files from Evaluated Nuclear Structure Data Files (**ENSDF**)
    - As of Geant4 10.5 , these are in **RadioactiveDecay5.3** pointed by the enviromental variable G4RADIOACTIVEDATA
  - These data files contain properties such as:
    half-lives, nuclear level structure for parent and daughter nuclides, type of decay, decay branching ratios, energy of decay process, *etc.*

Radioactive Decay Chain



EC: electron capture

IC: internal conversion

ARM: atomic relaxation model

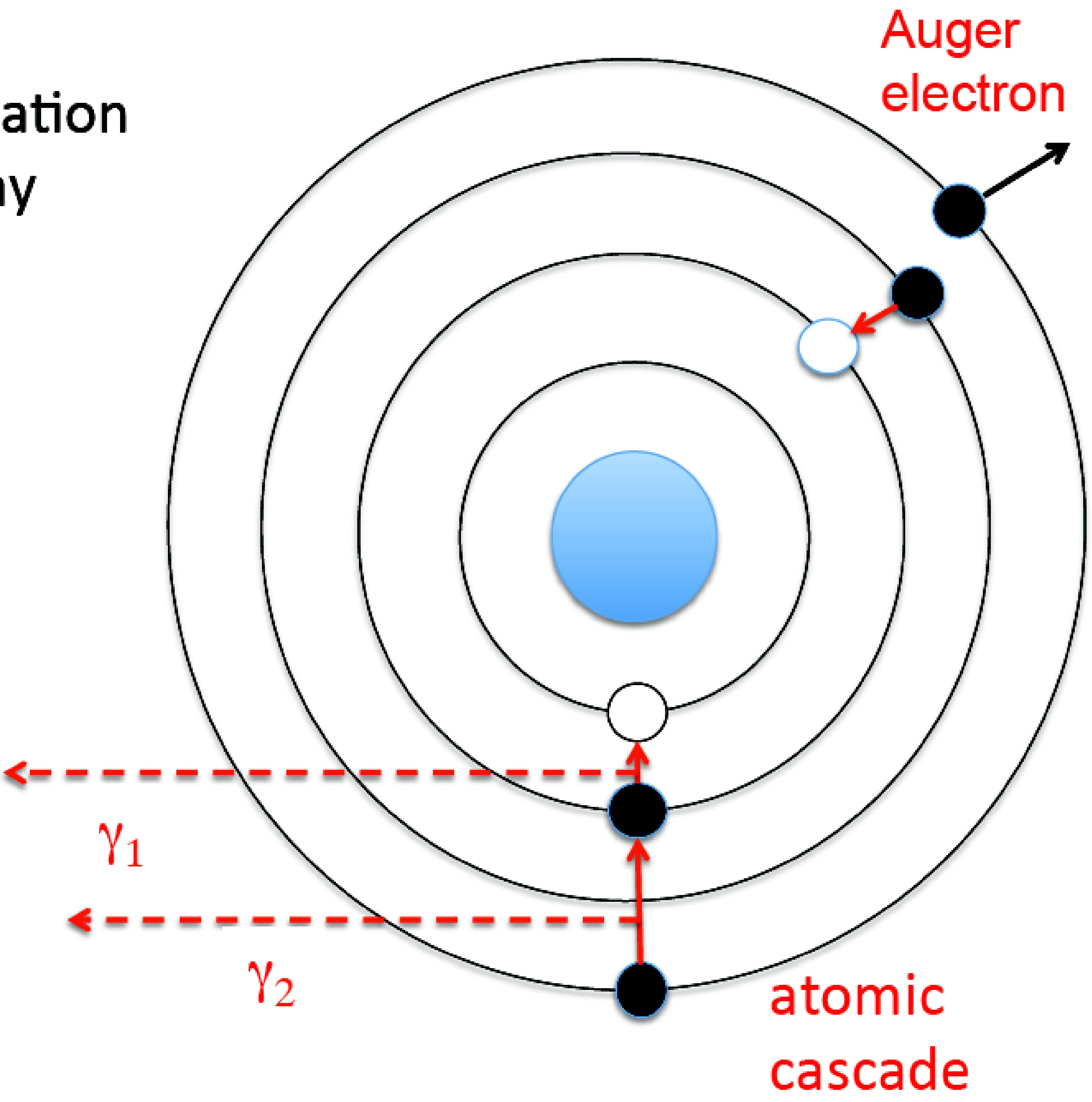
Atomic Relaxation Model

electron shell configuration may change after decay

inner holes filled by atomic cascade

either photons or Auger electrons are emitted

fluorescence option also available



Gamma (or Electron) Emission

- If a daughter of a nuclear decay is an isomer, prompt de-excitation is done by using **G4PhotonEvaporation**
  - Uses **ENSDF** files with all known gamma levels (in total ~ 25'500 levels with half life > 10^-23 s) for ~ 3'000 isotopes
    - As of Geant4 10.5 , these are in **PhotonEvaporation5.3** pointed by the enviromental variable G4LEVELGAMMADATA
  - Internal conversion (*i.e.* nuclear de-excitation via emission of atomic electrons) is enabled as a competing process to gamma de-excitation
- Nuclides with half life **< 1 ns** (< 1 μs for biasing) are forced to decay immediately (can be set via UI command "**/grdm/hlThreshold**")
- Option to enable atomic relaxation after decay
  - When Radioactive Decay is activated, Fluorescence and Auger emissions are switched on by default (overriding any EM default settings). User can use UI command to change this

Sampling Radioactive Decay : **Analogue** mode (default)

- Several options available via UI commands :
  - Enable/disable radioactive decay in various geometry volumes
    "**/grdm/selectVolume**" , "**/grdm/deselectVolume**"
  - Limits the nuclei in which radioactive decay can be applied (useful to limit the decay chain, i.e. to avoid decays of daughters)
    "**/grdm/nucleusLimits**"
  - Supply a user-defined radioactive decay datafile for a given isotope (useful, for instance, to amplify rare decay branches)
    "**/grdm/setRadioactiveDecayFile**"
  - Supply a user-defined evaporation datafile for a given isotope
    "**/grdm/setPhotoEvaporationFile**"
  - Switch on/off atomic relaxation
    "**/grdm/applyARM**" (default: true)

Sampling Radioactive Decay : **Biased** mode (alternative)

- Several options available via UI commands:
  - Set all decay branches equal
    "**/grdm/BRbias**" default: **true**
  - "Splitting" : perform nuclear decay N times for each event
    "**/grdm/splitNuclei**" default: 1
  - Activation : integrate decay chain over time windows (in seconds)
    "**/grdm/decayBiasProfile**" , "**/grdm/sourceTimeProfile**"
    using Bateman equations
    "**/grdm/analogueMC false**" (default: true)
  - Collimation of decay products
    "**/grdm/decayDirection**" , "**/grdm/decayHalfAngle**"

Using Radioactive Decay

- Set environmental variables to point to the data libraries:
  - **G4ENSDFSTATEDATA -> G4ENSDFSTATE2.2**
  - **G4LEVELGAMMADATA -> PhotonEvaporation5.3**
  - **G4RADIOACTIVEDATA -> RadioactiveDecay5.3**
- Add the physics constructor **G4RadioactiveDecayPhysics** to a reference physics list
  - Or do something like this in your own physics list :

```
G4RadioactiveDecay* rDecay = new G4RadioactiveDecay;
G4PhysicsListHelper* plh =  G4PhysicsListHelper::GetPhysicsListHelper();
rDecay -> SetICM( true );    // Internal conversion : obsolete, always true!
rDecay -> SetARM( true );  //  Atomic relaxation
plh -> RegisterProcess( rDecay, G4GenericIon::GenericIon() );
```

- Many options can be set via UI commands

Examples of using RDM

- ***examples/extended/radioactivedecay***
  - ***rdecay01/***
    - Shows basic features of the radioactive decay of nuclei : energy spectrum of emitted particles, time of life, activity
    - Analogue mode only, with user-defined RadioactiveDecay and PhotonEvaporation files
  - ***rdecay02/***
    - Induced radioactivity by nuclear reactions
    - Shows advanced features – e.g. selected decay channels, time window, etc. – in both analogue and biasing mode
  - ***Activation/***
    - Induced radioactivity by nuclear reactions
    - Shows the evolution of each metastable isomer as a function of time, taking into account the time of exposure of the beam; analogue mode only

GEANT4

A SIMULATION TOOLKIT

Biasing in Hadronic Physics

Built-in Biasing in Hadronics

- Radioactive Decay
  - Via UI commands (see before)
- Cross Sections
  - Possibility to scale any hadronic cross section via the method ***G4HadronicProcess::BiasCrossSectionByFactor(G4double aScale)***
  - No UI commands: need to write some code in the physics list, e.g. ***theElectronNuclearProcess->BiasCrossSectionByFactor(1000.0);***
- Leading Particle Biasing
  - At each hadronic interaction, keep only the most energetic particle (and randomly one particle of each species: meson, baryon, π°, γ) via the method: ***G4HadFinalState* G4HadLeadBias::Bias(G4HadFinalState* result)***
  - No UI commands: need to modify the PostStepDoIt method of hadronic processes (to which we want to apply this biasing) to create an instance of the class G4HadLeadBias and call its Bias method

Built-in Biasing in Hadronics

- Radioactive Decay **~ OK**
  - Via UI commands (see before)
- Cross Sections **Need your own P.L.**
  - Possibility to scale any hadronic cross section via the method **G4HadronicProcess::BiasCrossSectionByFactor(G4double aScale)**
  - No UI commands: need to write some code in the physics list, e.g. **theElectronNuclearProcess->BiasCrossSectionByFactor(1000.0);**
- Leading Particle Biasing **Too invasive !**
  - At each hadronic interaction, keep only the most energetic particle (and randomly one particle of each species: meson, baryon, π°, γ) via the method:
    G4HadFinalState* G4HadLeadBias::Bias(G4HadFinalState* result)
  - No UI commands: need to modify the PostStepDoIt method of hadronic processes (to which we want to apply this biasing) to create an instance of the class G4HadLeadBias and call its Bias method

Generic Bias for Hadronics

- This is the new and recommended approach for biasing in Geant4 – not only for hadronics !
    - Allow to mix biasing options via “building blocks” (instead of built-in functionalities)
- Examples available in:

    examples/extended/biasing/

  we discuss here the following two (relevant for hadronics):

    1. ***GB01/*** : cross-sections biasing (i.e. changing the natural xsec)
    2. ***GB02/*** : force-collision biasing (i.e. forcing an interaction in a volume)

  Note: there is plenty of user code, but nearly all of it can be re-used: only a tiny part needs to be customized per use-case !

  Note: at whatever level (stepping action, or sensitive detector) the **statistical weight of a track** can be obtained as:

  w = track->GetWeight()

Cross-Section Generic Biasing (GB01)

```
#include "FTFP_BERT.hh"
#include "G4GenericBiasingPhysics.hh"
…
int main( … ) {
  …
  FTFP_BERT* physicsList = new FTFP_BERT;
  G4GenericBiasingPhysics* biasingPhysics = new G4GenericBiasingPhysics;
  biasingPhysics->Bias( "gamma" );
  biasingPhysics->Bias( "neutron" );
  physicsList->RegisterPhysics( biasingPhysics );
  …
}
```

Enable biasing only for a subset of particle types

```
void GB01DetectorConstruction::ConstructSDandField() {
  …
  GB01BOptrMultiParticleChangeCrossSection* biasingOperator =
       new GB01BOptrMultiParticleChangeCrossSection;
  biasingOperator->AddParticle( "gamma" );
  biasingOperator->AddParticle( "neutron" );
  biasingOperator->AttachTo( logicVolumeToBias );
}
```

Possible to define xsec biasing for sets of particle types in specified logical volumes

```
class GB01BOptrMultiParticleChangeCrossSection : public G4VBiasingOperator {
  public:
    …
    void AddParticle( G4String particleName );
    void StartTracking( const G4Track* track );
  private:
    virtual G4VBiasingOperation* ProposeOccurenceBiasingOperation(...);
    virtual void OperationApplied(...);
    …
    std::map<const G4ParticleDefinition*, GB01BOptrChangeCrossSection*> fBOptrForParticle;
    std::vector< const G4ParticleDefinition* > fParticlesToBias;
    GB01BOptrChangeCrossSection* fCurrentOperator;
    G4int fnInteractions;
};

void GB01BOptrMultiParticleChangeCrossSection::AddParticle( G4String particleName ) {
  const G4ParticleDefinition* particle =
      G4ParticleTable::GetParticleTable()->FindParticle( particleName );
  ...
  GB01BOptrChangeCrossSection* optr =
          new GB01BOptrChangeCrossSection( particleName );
  fParticlesToBias.push_back( particle );
  fBOptrForParticle[ particle ] = optr;
}

void GB01BOptrMultiParticleChangeCrossSection::StartTracking( const G4Track* track )
   // Fetch the underneath biasing operator, if any, for the current particle type
   // and store it in fCurrentOperator
}
```

```
G4VBiasingOperation* GB01BOptrMultiParticleChangeCrossSection::
ProposeOccurenceBiasingOperation( const G4Track* track,
                                  const G4BiasingProcessInterface* callingProcess ) {
  // Examples of limitations imposed to apply the biasing:
  if ( track->GetParentID() != 0 )  return 0;  // Limit application of biasing to primary particles only
  if ( fnInteractions > 4 )         return 0;  // Limit to at most 5 biased interactions
  if ( track->GetWeight() < 0.05 ) return 0;  // Limit to a weight of at least 0.05
  if ( fCurrentOperator ) return fCurrentOperator->
                    GetProposedOccurenceBiasingOperation( track, callingProcess );
  else return 0;
}

void GB01BOptrMultiParticleChangeCrossSection::OperationApplied(...) {
   fnInteractions++;  // Count number of biased interactions
   // Inform the underneath biasing operator that a biased interaction occured:
   if ( fCurrentOperator ) fCurrentOperator->ReportOperationApplied(...);
}
```

```
class GB01BOptrChangeCrossSection : public G4VBiasingOperator {
 public:
   GB01BOptrChangeCrossSection( G4String particleToBias, G4String name = "ChangeXS" );
   …
   virtual void StartRun();
 private:
   virtual G4VBiasingOperation* ProposeOccurenceBiasingOperation(...);
   ...
   using G4VBiasingOperator::OperationApplied;
   virtual void OperationApplied(...);
   std::map< const G4BiasingProcessInterface*, G4BOptnChangeCrossSection* >
                fChangeCrossSectionOperations;
   const G4ParticleDefinition* fParticleToBias;
}

void GB01BOptrChangeCrossSection::StartRun() {
  const G4ProcessManager* processManager = fParticleToBias->GetProcessManager();
  const G4BiasingProcessSharedData* sharedData =
              G4BiasingProcessInterface::GetSharedData( processManager );
  for ( size_t i = 0 ; i < (sharedData->GetPhysicsBiasingProcessInterfaces()).size(); i++ ) {
      const G4BiasingProcessInterface* wrapperProcess =
                (sharedData->GetPhysicsBiasingProcessInterfaces())[i];
      G4String operationName = "XSchange-" +
           wrapperProcess->GetWrappedProcess()->GetProcessName();
      fChangeCrossSectionOperations[ wrapperProcess ] =
              new G4BOptnChangeCrossSection( operationName );
  }
}
```

```
G4VBiasingOperation* GB01BOptrChangeCrossSection::
ProposeOccurenceBiasingOperation(...) {
  if ( track->GetDefinition() != fParticleToBias ) return 0;
  G4double analogInteractionLength =
     callingProcess->GetWrappedProcess()->GetCurrentInteractionLength();
  if ( analogInteractionLength > DBL_MAX/10.0 ) return 0;
  G4double analogXS = 1.0/analogInteractionLength;
  G4BOptnChangeCrossSection* operation =
          fChangeCrossSectionOperations[ callingProcess ];
  ...
  operation->SetBiasedCrossSection( 2.0 * analogXS );  //<--- Scaling factor for the xsec !
  operation->Sample();
  …
  return operation;
}
```

Alternatively, one could use different xsec scaling factors according to the particle type and/or process type

```
void GB01BOptrChangeCrossSection::OperationApplied(...) {
   G4BOptnChangeCrossSection* operation =
                 fChangeCrossSectionOperations[ callingProcess ];
   if ( operation == occurenceOperationApplied ) operation->SetInteractionOccured();
}
```

Force-Collision Generic Biasing (GB02)

```
#include "FTFP_BERT.hh"
#include "G4GenericBiasingPhysics.hh"
…
int main( … ) {
  …
  FTFP_BERT* physicsList = new FTFP_BERT;
  G4GenericBiasingPhysics* biasingPhysics = new G4GenericBiasingPhysics;
  biasingPhysics->Bias( "gamma" );
  biasingPhysics->Bias( "neutron" );
  physicsList->RegisterPhysics( biasingPhysics );
  …
}
```

Enable biasing only for a subset of particle types

```
void GB02DetectorConstruction::ConstructSDandField() {
  …
  GB02BOptrMultiParticleForceCollision* biasingOperator =
        new GB02BOptrMultiParticleForceCollision;
  biasingOperator->AddParticle( "gamma" );
  biasingOperator->AddParticle( "neutron" );
  biasingOperator->AttachTo( logicVolumeToBias );
}
```

Possible to define force collision for sets of particle types in specified logical volumes

```
class GB02BOptrMultiParticleForceCollision : public G4VBiasingOperator {
 public:
   ...
   void AddParticle( G4String particleName );  // Declare particles to be biased
   virtual void StartTracking( const G4Track* track );
 private:
   virtual G4VBiasingOperation* ProposeNonPhysicsBiasingOperation(...);
   virtual G4VBiasingOperation* ProposeOccurenceBiasingOperation(...);
   virtual G4VBiasingOperation* ProposeFinalStateBiasingOperation(...);
   void OperationApplied(...);
   void ExitBiasing(...);
   std::map< const G4ParticleDefinition*, G4BOptrForceCollision* > fBOptrForParticle;
   std::vector< const G4ParticleDefinition* > fParticlesToBias;
   G4BOptrForceCollision* fCurrentOperator;
};


void GB02BOptrMultiParticleForceCollision::AddParticle( G4String particleName ) {
 const G4ParticleDefinition* particle =
    G4ParticleTable::GetParticleTable()->FindParticle( particleName );
 if ( particle == 0 ) { … }  // just warning exception and return
 G4BOptrForceCollision* optr =
     new G4BOptrForceCollision( particleName, "ForceCollisionFor" + particleName );
 fParticlesToBias.push_back( particle );
 fBOptrForParticle[ particle ] = optr;
}
```

```
G4VBiasingOperation* GB02BOptrMultiParticleForceCollision::
ProposeOccurenceBiasingOperation(...) {
  if ( fCurrentOperator )
    return fCurrentOperator->GetProposedOccurenceBiasingOperation( track, callingProcess );
  else return 0;
}
```

Possible to impose here some limitations on when to apply biasing, e.g. only for a certain process type

```
G4VBiasingOperation* GB02BOptrMultiParticleForceCollision::
ProposeFinalStateBiasingOperation(...) {
  if ( fCurrentOperator )
    return fCurrentOperator->GetProposedFinalStateBiasingOperation( track, callingProcess );
  else return 0;
}

void GB02BOptrMultiParticleForceCollision::StartTracking( const G4Track* track ) {
  const G4ParticleDefinition* definition = track->GetParticleDefinition();
  std::map< const G4ParticleDefinition*, G4BOptrForceCollision* >:: iterator it =
                                                                    fBOptrForParticle.find( definition );
  fCurrentOperator = 0;
  if ( it != fBOptrForParticle.end() ) fCurrentOperator = (*it).second;
}

void GB02BOptrMultiParticleForceCollision::OperationApplied(...) {
   if ( fCurrentOperator ) fCurrentOperator->ReportOperationApplied(...);
}

void GB02BOptrMultiParticleForceCollision::OperationApplied(...) {
  if ( fCurrentOperator ) fCurrentOperator->ReportOperationApplied(...);
}

void GB02BOptrMultiParticleForceCollision::ExitBiasing(...) {
  if ( fCurrentOperator ) fCurrentOperator->ExitingBiasing( track, callingProcess );
}
```

GEANT4
A SIMULATION TOOLKIT

Exercises

Exercise 1 : NeutronHP

- Consider the example:
  - ***examples/basic/B4/B4a***
- Enlarge the calorimeter (from a typical EM to a typical HAD)
  - In the method ***B4DetectorConstruction::DefineVolumes*** increase both ***nofLayers*** and ***calorSizeXY*** by a factor of ***10***
- Observe how the execution time changes
  - Consider a ***pi−*** beam of ***10 GeV*** energy
  - With **FTFP_BERT** physics list
  - With **FTFP_BERT_HP** physics list
  - With **FTFP_BERT_HP + thermal scattering** physics list
  - With **FTFP_BERT_HP + thermal scattering** physics list and replacing the active material ***liquidArgon*** with e.g. ***G4_WATER***

Exercise 2 : Radioactive Decay

- Consider the example:
  - ***examples/extended/radioactivedecay/rdecay01/***
- Study the decays of **As74**
  - which has a rather complicated decay scheme in β-, β+ and EC

in both analogue and bias modes

  - You can limit the decay chain, i.e. to avoid to decay its daughters as follows:

/grdm/nucleusLimits 74 74 33 33

Exercise 3 : Generic Biasing (1/2)

- Consider the example:
    - ***examples/extended/biasing/GB01***
- Run the example (using for instance as input *exampleGB01.in*) and compare the following 4 modes:

1. As it is, with biased cross sections for all gamma and neutron processes
2. Bias x 100 the cross sections for all gamma processes, and x 200 the cross sections for all neutron processes
    – Look at the method *ProposeOccurenceBiasingOperation* ...
3. Keeping biased cross sections only for gamma, but not for neutron
    – Simple: it is enough to comment out one line...
4. Bias the cross section only for the "*photonNuclear*" gamma process
    – Check the name of the wrapped process:
    callingProcess->GetWrappedProcess()->GetProcessName()

Exercise 3 : Generic Biasing (2/2)

- Consider the example:
  - ***examples/extended/biasing/GB02***
- Run the example (using for instance as input *exampleGB02.in*) and compare the following 4 modes:

1. As it is, with forced interaction for all gamma and neutron processes
2. Apply biasing only for the "*photonNuclear*" gamma process
   - Check the name of the wrapped process: *callingProcess->GetWrappedProcess()->GetProcessName()*

   After disabling the forced interaction (for any particle)
   - Simple: it is enough to comment out one line...
3. With natural cross sections
4. With x 100 bias of the cross section of the "*photonNuclear*" gamma process, for gamma of energy > 50 MeV
   - Difficult: copy 2 classes from GB01... and then use one of these in: *GB020DetectorConstruction::ConstructSDandField*