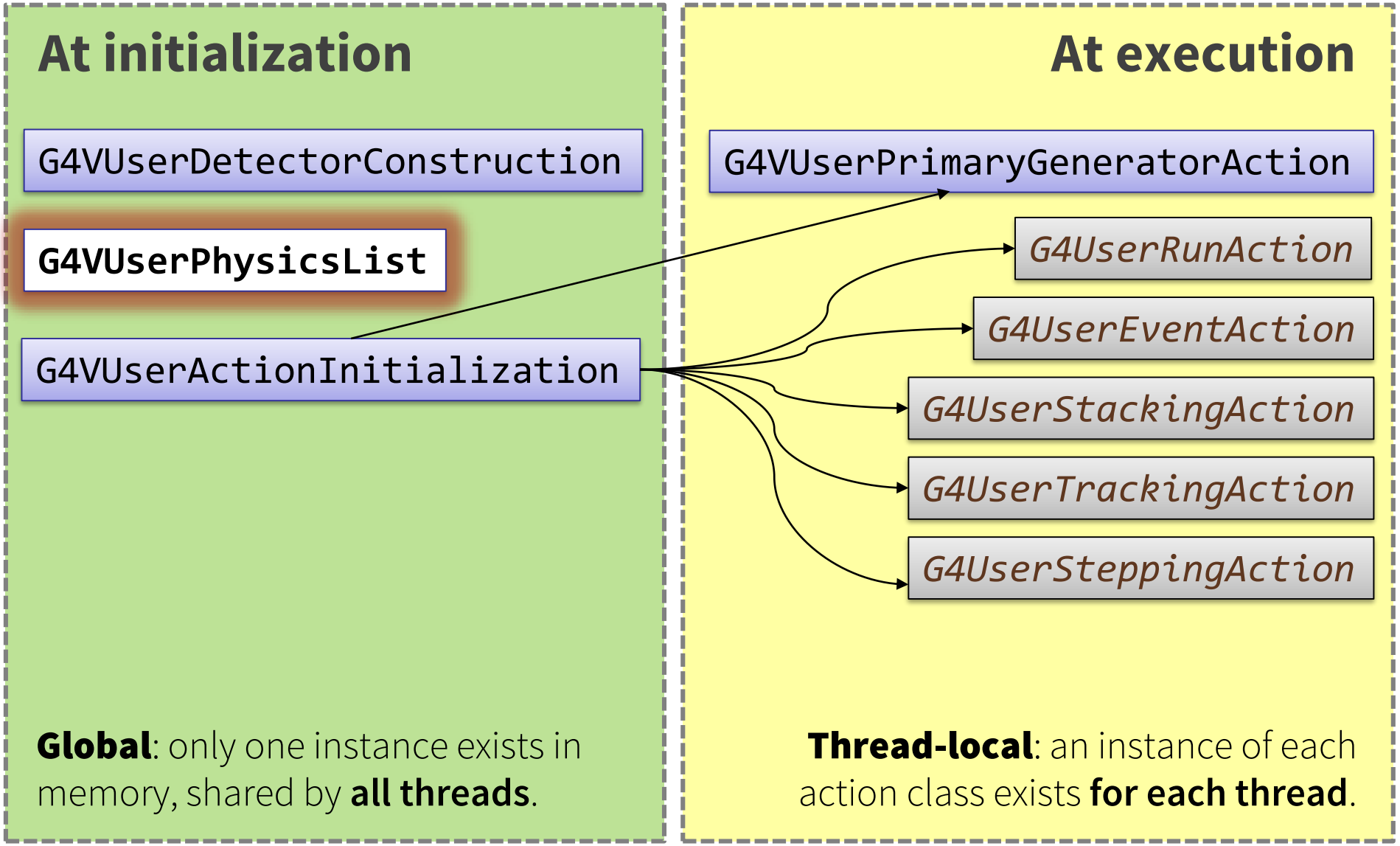
XIV Seminar on Software for Nuclear, Subnuclear and Applied Physics

Alghero, 5-9 June 2017

Physics in Geant4: Particles and processes

Geant4 tutorial

User classes (...continued)



Contents

- Physics in Geant4 – motivation
- Particles
- Processes
- Physics lists

…Part 2:

- Production cuts
- Electromagnetic / hadronic physics

“Shouldn’t there be just one universal and complete physics description?”

No.

Physics – the challenge

- Huge amount of different processes for various purposes (only a handful relevant)
- Competing descriptions of the same physics phenomena (necessary to choose)
  - fundamentally different **approaches**
  - balance between **speed** and **precision**
  - different **parameterizations**
- Hypothetical processes & exotic physics

Solution: Atomistic approach with modular **physics lists**

Part I:
Particles

Particles: basic concepts

There are three levels of class to describe particles in Geant4:

`G4ParticleDefinition`

Particle static properties: name, mass, spin, PDG number, etc.

`G4DynamicParticle`

Particle dynamic state: energy, momentum, polarization, etc.

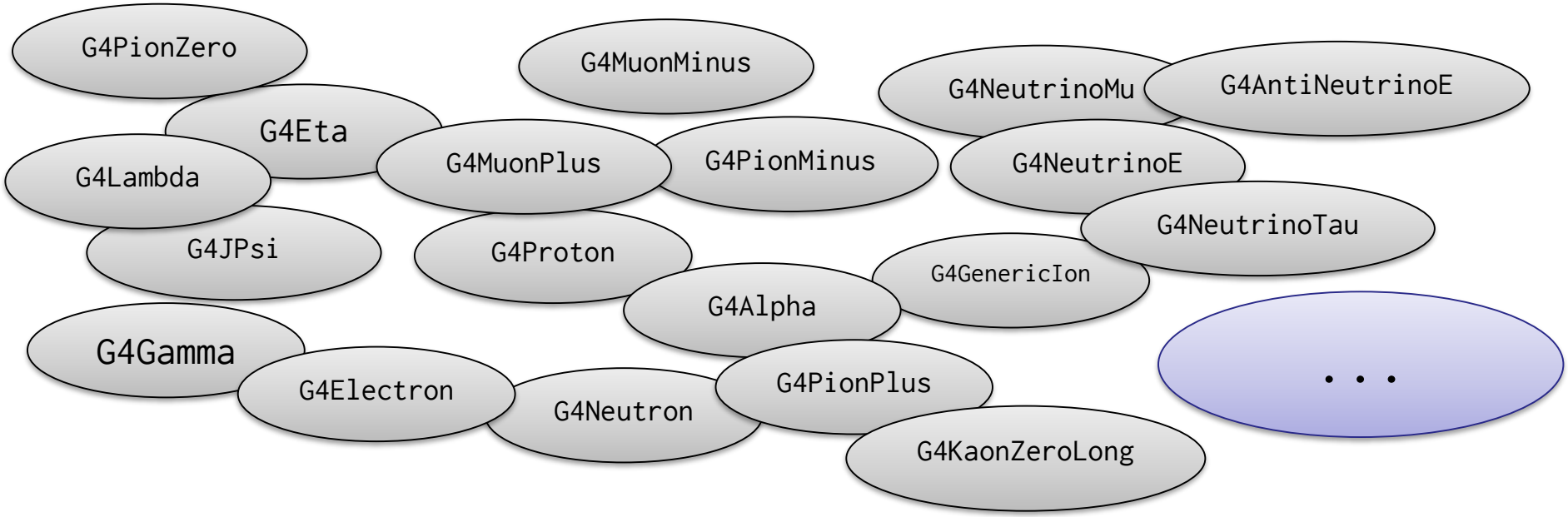
`G4Track`

Information for tracking in a detector simulation: position, step, current volume, track ID, parent ID, etc.

Definition of a particle

Geant4 provides **G4ParticleDefinition** daughter classes to represent a large number of elementary particles and nuclei, organized in six major categories:

leptons*, *mesons*, *baryons*, *bosons*, *short-lived *and* ***ions***



User must define ***all particle*** *types which might be used in the application: not only* ***primary particles*** *but also all other particles which may appear as* ***secondaries*** *generated by the used physics processes*

Particles in Geant4

- **Particle Data Group (PDG) particles**
- **Optical photons** (different from gammas!)
- Special particles: **geantino** and **charged geantino**
  - Only transported in the geometry (no interactions)
  - Charged geantino also feels the EM fields
- **Short-lived** particles ($\tau < 10^{-14}$ s) are **not transported** by Geant4 (decay applied)
- **Light ions** (as deuterons, tritons, alphas)
- **Heavier ions** represented by a single class: **`G4Ions`**

http://geant4.cern.ch/G4UsersDocuments/UsersGuides/ForApplicationDeveloper/html/TrackingAndPhysics/particle.html

Leptons & bosons table

Particle name	Class name	Name (in GPS...)	PDG
electron	G4Electron	e-	11
positron	G4Positron	e+	-11
muon +/-	G4MuonPlus G4MuonMinus	mu+ mu-	-13 13
tauon +/-	G4TauPlus G4TauMinus	tau+ tau-	-15 15
electron (anti)neutrino	G4NeutrinoE G4AntiNeutrinoE	nu_e anti_nu_e	12 -12
muon (anti)neutrino	G4NeutrinoMu G4AntiNeutrinoMu	nu_mu anti_nu_mu	14 -14
tau (anti)neutrino	G4NeutrinoTau G4AntiNeutrinoTau	nu_tau anti_nu_tau	16 -16
photon (γ, X)	G4Gamma	gamma	22
photon (optical)	G4OpticalPhoton	opticalphoton	(0)
geantino	G4Geantino	geantino	(0)
charged geantino	G4ChargedGeantino	chargedgeantino	(0)

Common hadrons & ions table

Particle name	Class name	Name (in GPS...)	PDG
(anti)proton	G4Proton G4AnitProton	proton anti_proton	2212 -2212
(anti)neutron	G4Neutron G4AntiNeutron	neutron anti_neutron	2112 -2112
(anti)lambda	G4Lambda G4AntiLambda	lambda anti_lambda	3122 -3122
pion	G4PionMinus G4PionPlus G4PionZero	pi- pi+ pi0	-211 211 111
kaon	G4KaonMinus G4KaonPlus G4KaonZero G4KaonZeroLong G4KaonZeroShort	kaon- kaon+ kaon0 kaon0L kaon0S	-321 321 311 130 310
(anti)alpha	G4Alpha G4AntiAlpha	alpha anti_alpha	1000020040 -1000020040
(anti)deuteron	G4Deteuron G4AntiDeuteron	deuteron anti_deuteron	1000010020 -1000010020
Heavier ions	G4Ions	ion	100ZZZAAAI*

*ZZZ=proton number, AAA=nucleon number, I=excitation level

Part II:
Processes

Processes

How do particles interact with materials?

Responsibilities:

1. *decide **when** and **where** an interaction occurs*
   - `GetPhysicalInteractionLength...()`

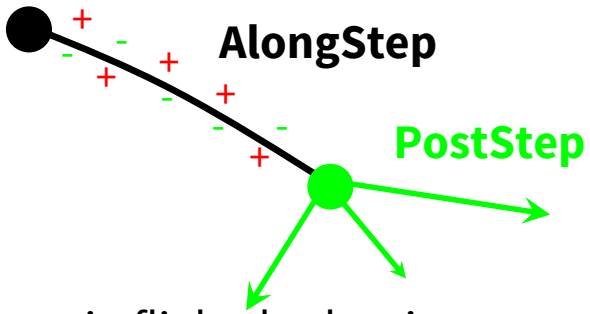
     → *limit the step*
   - this requires a cross section
   - for the **transportation** process, the distance to the nearest object

2. *generate the **final state** of the interaction*
   - changes momentum, generates secondaries, etc.)
   - method: `DoIt...()`
   - this requires a model of the physics

G4VProcess class

- Abstract class as a base for all processes in Geant4
  - Used by all physics processes (also by the transportation, etc…
  - Defined in **source/processes/management**
- Define three kinds of actions:

  - **AtRest** actions:
    - Decay, e^+ annihilation …
  - **AlongStep** actions:
    - To describe continuous (inter)actions, occurring along the path of the particle, like ionisation;
  - **PostStep** actions:
    - For describing point-like (inter)actions, like decay in flight, hadronic interactions …



A process can implement a combination of them (decay = AtRest + PostStep)

Example processes

- Discrete process: Compton Scattering, hadronic inelastic, ...
  - step determined by cross section, interaction at end of step
    - PostStepGPIL(), PostStepDoIt()
- Continuous process: Čerenkov effect
  - photons created along step, roughly proportional to step length
    - AlongStepGPIL(), AlongStepDoIt()
- At rest process: muon capture at rest
  - interaction at rest
    - AtRestGPIL(), AtRestDoIt()

pure

- Rest + discrete: positron annihilation, decay, ...
  - both in flight and at rest
- Continuous + discrete: ionization
  - energy loss is continuous
  - knock-on electrons (δ-ray) are discrete

combined

Handling multiple processes

1 a particle is shot and “transported”

2 all processes associated to the particle propose a geometrical step length (depends on process cross-section)

3 The process proposing the shortest step “wins” and the particle is moved to destination (if shorter than “Safety”)

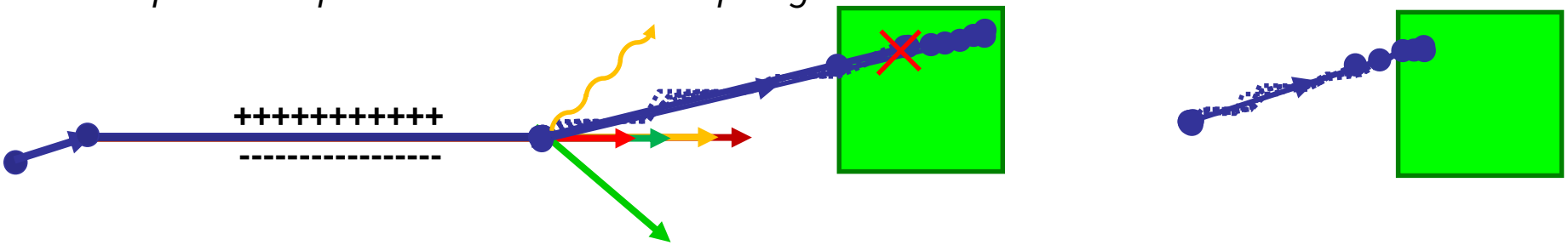
4 All processes along the step are executed (e.g. ionization)

5 post step phase of the process that limited the step is executed.
New tracks are “pushed” to the stack

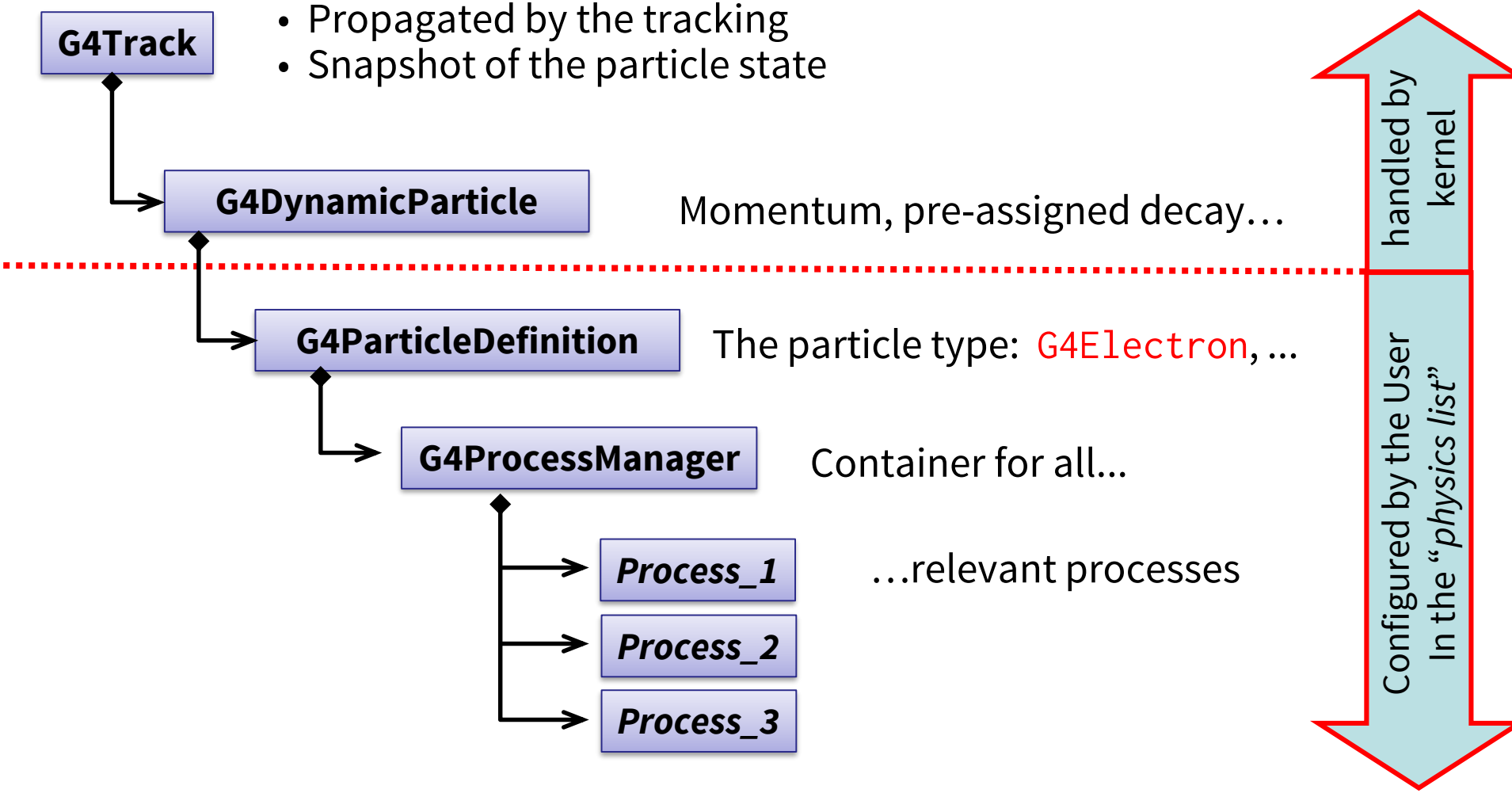
6 If E_{kin}=0 all at rest processes are executed; if particle is stable the track is killed.
Else:

7 New step starts and sequence repeats...

- *Processes return a “true path length”. The multiple scattering “virtually folds up” this true path length into a shorter ”geometrical” path length.*
- *Transportation process can limit the step to geometrical boundaries.*



From particles to processes



Part III:
Physics lists

Physics list

- One instance per application
  - registered to run manager in `main()`
  - inheriting from `G4VUserPhysicsList`
- Responsibilities
  - all **particle types** (electron, proton, gamma, ...)
  - all **processes** (photoeffect, bremsstrahlung, ...)
  - all **process parameters** (...)
  - **production cuts** (e.g. 1 mm for electrons, ...)

3 ways to get a physics list

1) **Manual:** Specify all particles & processes that may occur in the simulation. (difficult)

2) **Physics constructors:** Combine your physics from pre-defined sets of particles and processes.
Still you define your own class – modular physics list (easier)

3) **Reference physics lists:** Take one of the pre-defined physics lists. You don't create any class (easy)

Way 1

G4VUserPhysicsList class

Implement 3 methods:

```
class MyPhysicsList : public G4VUserPhysicsList {
public:
    // ...
    void ConstructParticle();  // pure virtual
    void ConstructProcess();   // pure virtual
    void SetCuts();
    // ...
}
```

Advantage: most flexible

Disadvantages:

- most verbose
- most difficult to get right

G4VUserPhysicsList: implementation

`ConstructParticle()`:

- choose the particles you need in your simulation, define all of them here

`ConstructProcess()` :

- for each particle, assign all the physics processes relevant to your simulation

`SetCuts()` :

- set the range cuts for secondary production for processes with infrared divergence

i) ConstructParticle()

Due to the large number of particles can be necessary to instantiate, this method sometimes can be not so comfortable

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

It is possible to define **all** the particles belonging to a **Geant4 category:**

- **G4LeptonConstructor**
- **G4MesonContructor**
- **G4BaryonConstructor**
- **G4BosonConstructor**
- **G4ShortlivedConstructor**
- **G4IonConstructor**

```
void MyPhysicsList::ConstructParticle() {
     // Construct all baryons
     G4BaryonConstructor bConstructor;
     bConstructor.ConstructParticle();
     // Construct all leptons
     G4LeptonConstructor lConstructor;
     lConstructor.ConstructParticle();
     // ...
}
```

ii) ConstructProcess()

1. For each particle, get its **process manager**.

```
G4ProcessManager *elManager = G4Electron::ElectronDefinition()->GetProcessManager();
```

2. Construct all **processes** and **register** them.

```
elManager->AddProcess(new G4eMultipleScattering, -1, 1, 1);
elManager->AddProcess(new G4eIonisation, -1, 2, 2);
elManager->AddProcess(new G4eBremsstrahlung, -1, -1, 3);
elManager->AddDiscreteProcess(new G4StepLimiter);
```

3. Don’t forget **transportation**.

```
AddTransportation();
```

iii) SetCuts()

- Define all **production** c… gamma, electrons and positr…
  - Recently also for …s
- Notice: this is … **duction cut**, not a tracking cu…

MORE ON THIS LATER

Example: **Put it together**

```
void StandardPhysics::ConstructParticle()
{
        // We are interested in gamma, electrons and possibly positrons
        G4Electron::ElectronDefinition();
        G4Positron::PositronDefinition();
        G4Gamma::GammaDefinition();
}

void StandardPhysics::ConstructProcess()
{
        // Transportation is necessary
        AddTransportation();

        // Electrons
        G4ProcessManager *elManager = G4Electron::ElectronDefinition()->GetProcessManager();
        elManager->AddProcess(new G4eMultipleScattering, -1, 1, 1);
        elManager->AddProcess(new G4eIonisation, -1, 2, 2);
        elManager->AddProcess(new G4eBremsstrahlung, -1, -1, 3);
        elManager->AddDiscreteProcess(new G4StepLimiter);

        // Positrons
        G4ProcessManager *posManager = G4Positron::PositronDefinition()->GetProcessManager();
        posManager->AddProcess(new G4eMultipleScattering, -1, 1, 1);
        posManager->AddProcess(new G4eIonisation, -1, 2, 2);
        posManager->AddProcess(new G4eBremsstrahlung, -1, -1, 3);
        posManager->AddProcess(new G4eplusAnnihilation, 0, -1, 4);
        posManager->AddDiscreteProcess(new G4StepLimiter);

        // Gamma
        G4ProcessManager *phManager = G4Gamma::GammaDefinition()->GetProcessManager();
        phManager->AddDiscreteProcess(new G4ComptonScattering);
        phManager->AddDiscreteProcess(new G4PhotoElectricEffect);
        phManager->AddDiscreteProcess(new G4GammaConversion);

        // TODO: Introduce Rayleigh scattering. It has large cross-section than Pair production
}

void StandardPhysics::SetCuts()
{
        // TODO: Create a messenger for this
        defaultCutValue = 0.03 * mm;
        SetCutsWithDefault();
}
```

Way 2

G4VModularPhysicsList

- Similar structure as `G4VUserPhysicsList` (same methods to override – though not necessary):

```
class MyPhysicsList : public G4VModularPhysicsList {
public:
    MyPhysicsList();            // define physics constructors
    void ConstructParticle();   // optional
    void ConstructProcess();    // optional
    void SetCuts();             // optional
}
```

Differences to "manual" way:

- Particles and processes typically handled by **physics constructors** (still customizable)
- **Transportation** automatically included

Physics constructor

= "**module**" of the **modular physics list**

- Inherits from **`G4VPhysicsConstructor`**
- Defines **ConstructParticle**() and **ConstructProcess**()
  - to be fully imported in modular list (behaving in the same way)
- **GetPhysicsType**()
  - enables switching physics of the same type, if possible (see next slide)

Physics constructors

- Huge set of pre-defined ones
  - **EM**: Standard, Livermore, Penelope
  - **Hadronic inelastic:** QGSP_BIC, FTFP_Bert, ...
  - **Hadronic elastic:** G4HadronElasticPhysics, ...
  - ... (decay, optical physics, EM extras, ...)
- You can implement your own (of course) by inheriting from the G4VPhysicsConstructor class

Code: `$G4INSTALL/source/physics_lists/constructors`

Using physics constructors

Add **physics** **constructor** in the **constructor**:

```
MyModularList::MyModularList() {
    // Hadronic physics
    RegisterPhysics(new G4HadronElasticPhysics());
    RegisterPhysics(new G4HadronPhysicsFTFP_BERT_TRV());
    // EM physics
    RegisterPhysics(new G4EmStandardPhysics());
}
```

This already works and no further method overriding is necessary ☺.

To be continued (if you want to customize)...

Customize G4VModularPhysicsList

You can override the `CreateParticle()`, `CreateProcess()`, and `SetCuts()` methods:

```
void MyModularList::ConstructProcess() {
     // Call the default implementation, otherwise you break the behaviour
     G4VModularPhysicsList::ConstructProcess();

     // Add your customization
     G4ProcessManager *elManager = G4Electron::ElectronDefinition()->GetProcessManager();
     elManager->AddDiscreteProcess(new MyElectronProcess);
}
```

Replace physics constructors

You can **add** or **remove** the physics constructors after the list instance is created:

- e.g. in response to **UI command**
- only **before initialization**
- physics of the same type can be **replaced**

```
void MyModularList::SelectAlternativePhysics() {
    AddPhysics(new G4OpticalPhysics);
    RemovePhysics(fDecayPhysics);
    ReplacePhysics(new G4EmLivermorePhysics);
}
```

Way 3

Reference physics lists

- Pre-defined physics lists
  - already containing a complete set of particles & processes (that work together)
  - targeted at specific area of interest (HEP, medical physics, ...)
  - constructed as **modular physics lists**, built on top of **physics constructors**
  - customizable (by calling appropriate methods before initialization)

Using a reference physics list

In the `main()` function, just register an instance of the physics list to the **`G4(MT)RunManager:`**

```
#include "QGSP_BERT.hh"
int main() {
    // Run manager
    G4RunManager * runManager = new G4RunManager();
    // ...
    G4VUserPhysicsList* physics = new QGSP_BERT();
    // Here, you can customize the “physics” object
    runManager->SetUserInitialization(physics);
    // ...
}
```

Alternative: **Reference list by name**

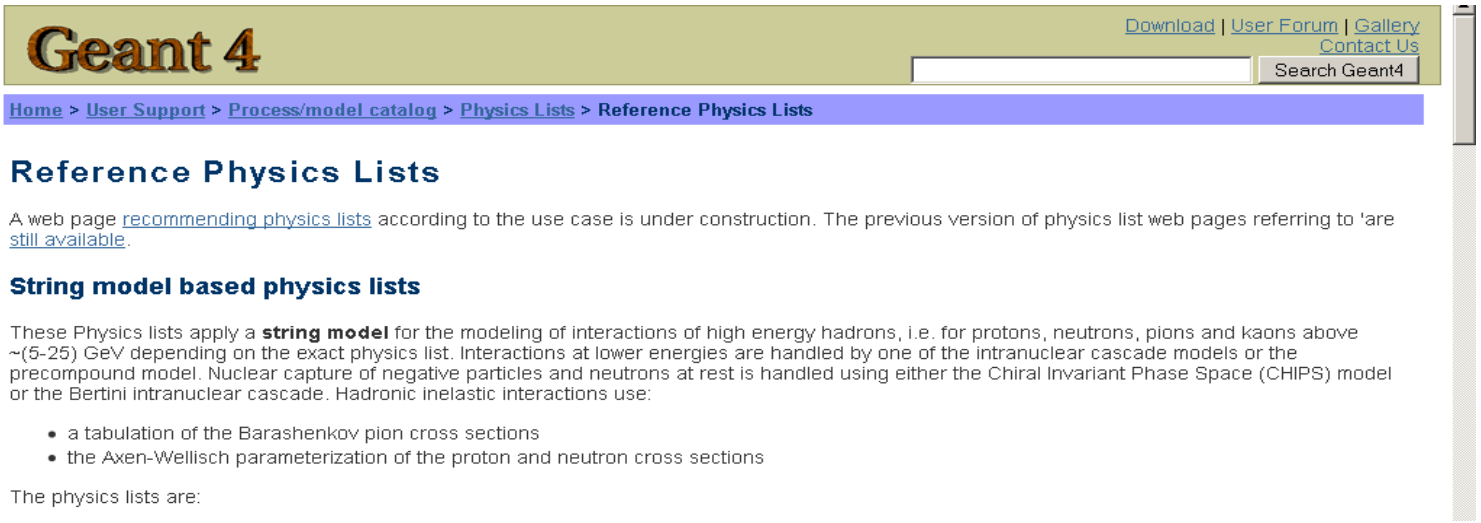
- If you want to get reference physics lists by **name** (e.g. from environment variable), you can use the `G4PhysListFactory` class:

```
#include "G4PhysListFactory.hh"
int main() {
    // Run manager
    G4RunManager* runManager = new G4RunManager();
    // E.g. get the list name from environment varible
    G4String listName{ getenv("PHYSICS_LIST") };
    auto factory = new G4PhysListFactory();
    auto physics = factory->GetReferencePhysList(listName);
    runManager->SetUserInitialization(physics);
    // ...
}
```

Lists of reference physics lists

Source code: `$G4INSTALL/source/physics_lists/lists`

```
FTF_BIC.hh
FTFP_BERT.hh
FTFP_BERT_HP.hh
FTFP_BERT_TRV.hh
FTFP_INCLXX.hh
FTFP_INCLXX_HP.hh
G4GenericPhysicsList.hh
G4PhysListFactoryAlt.hh
G4PhysListFactory.hh
G4PhysListRegistry.hh
G4PhysListStamper.hh
INCLXXPhysicsListHelper.hh
LBE.hh
NuBeam.hh
QBBC.hh
QGS_BIC.hh
QGSP_BERT.hh
QGSP_BERT_HP.hh
QGSP_BIC_AllHP.hh
QGSP_BIC.hh
QGSP_BIC_HP.hh
QGSP_FTFP_BERT.hh
QGSP_INCLXX.hh
QGSP_INCLXX_HP.hh
Shielding.hh
```



Geant 4

Download | User Forum | Gallery
Contact Us
Search Geant4

Home > User Support > Process/model catalog > Physics Lists > Reference Physics Lists

Reference Physics Lists

A web page recommending physics lists according to the use case is under construction. The previous version of physics list web pages referring to 'are still available.

String model based physics lists

These Physics lists apply a **string model** for the modeling of interactions of high energy hadrons, i.e. for protons, neutrons, pions and kaons above ~(5-25) GeV depending on the exact physics list. Interactions at lower energies are handled by one of the intranuclear cascade models or the precompound model. Nuclear capture of negative particles and neutrons at rest is handled using either the Chiral Invariant Phase Space (CHIPS) model or the Bertini intranuclear cascade. Hadronic inelastic interactions use:

- a tabulation of the Barashenkov pion cross sections
- the Axen-Wellisch parameterization of the proton and neutron cross sections

The physics lists are:

Where to find information?

User Support

1. Getting started
2. Training courses and materials
3. Source code
    a. Download page
    b. LXR code browser -or- draft doxygen documentation
4. Frequently Asked Questions (FAQ)
5. Bug reports and fixes
6. User requirements tracker
7. User Forum
8. Documentation
    a. Introduction to Geant4
    b. Installation Guide
    c. Application Developers Guide
    d. Toolkit Developers Guide
    e. Physics Reference Manual
    f. Software Reference Manual
9. Physics lists
    a. Electromagnetic
    b. Hadronic

http://geant4.web.cern.ch/geant4/support/index.shtml

3 ways to get a physics list (summary)

1) Manual:

`G4VUserPhysicsList`

2) Physics constructors:

`G4VModularPhysicsList`

3) Reference physics lists:

no class ☺

Conclusion

- Geant4 description of physics is very flexible
  - many particles
  - many processes
  - many models
  - many physics lists

…End of process