Particle definition, Process management

CERN Geant4 tutorial

February 2010

Andreas Schälicke (DESY)

based on Slides by Marc Verderi

Introduction

Overview

- Three main classes are responsible for making particles to sensitive to physics processes

  `G4ParticleDefinition`

  - describes the intrinsic physics properties
  - Mass, charge, etc…

  `G4ProcessManager`

  - used to attach physics processes to the particle

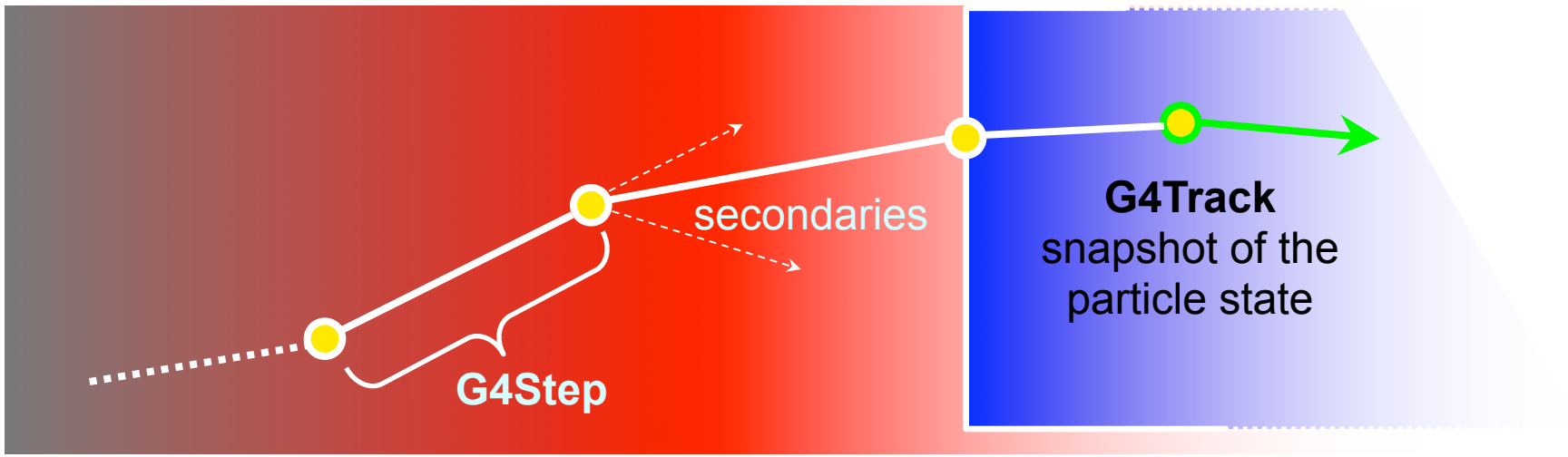
  `G4VProcess`

  - Base class for all physics processes

- It is your responsibility to add all particles and processes needed in your simulation
  - in your “physics list”

Reminder about `G4Track`

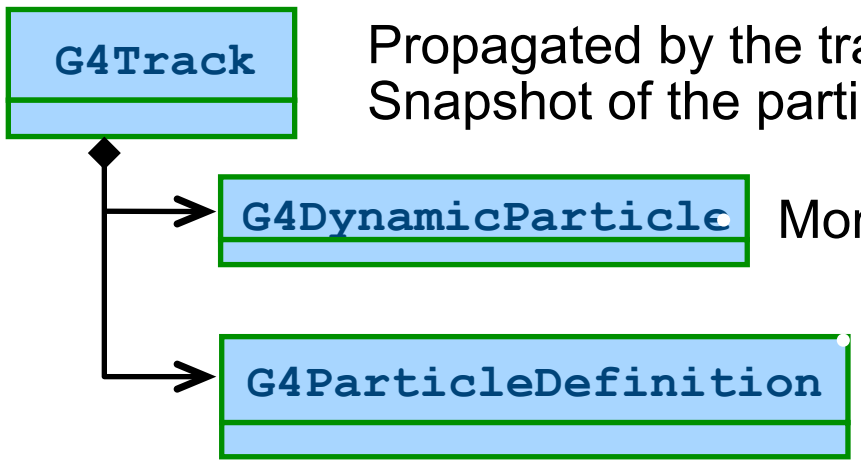
- **`G4Track`** is the object "pushed" step by step by the tracking :



- Moving by one step is the responsibility of the "stepping"
  - this is the core engine of the "tracking" machinery
- Moves/steps have to be physically meaningful
  - stepping invokes physics to realize them
- Physics is attached to the **`G4Track`**, let's see how.

From `G4Track` to processes

From `G4Track` to processes

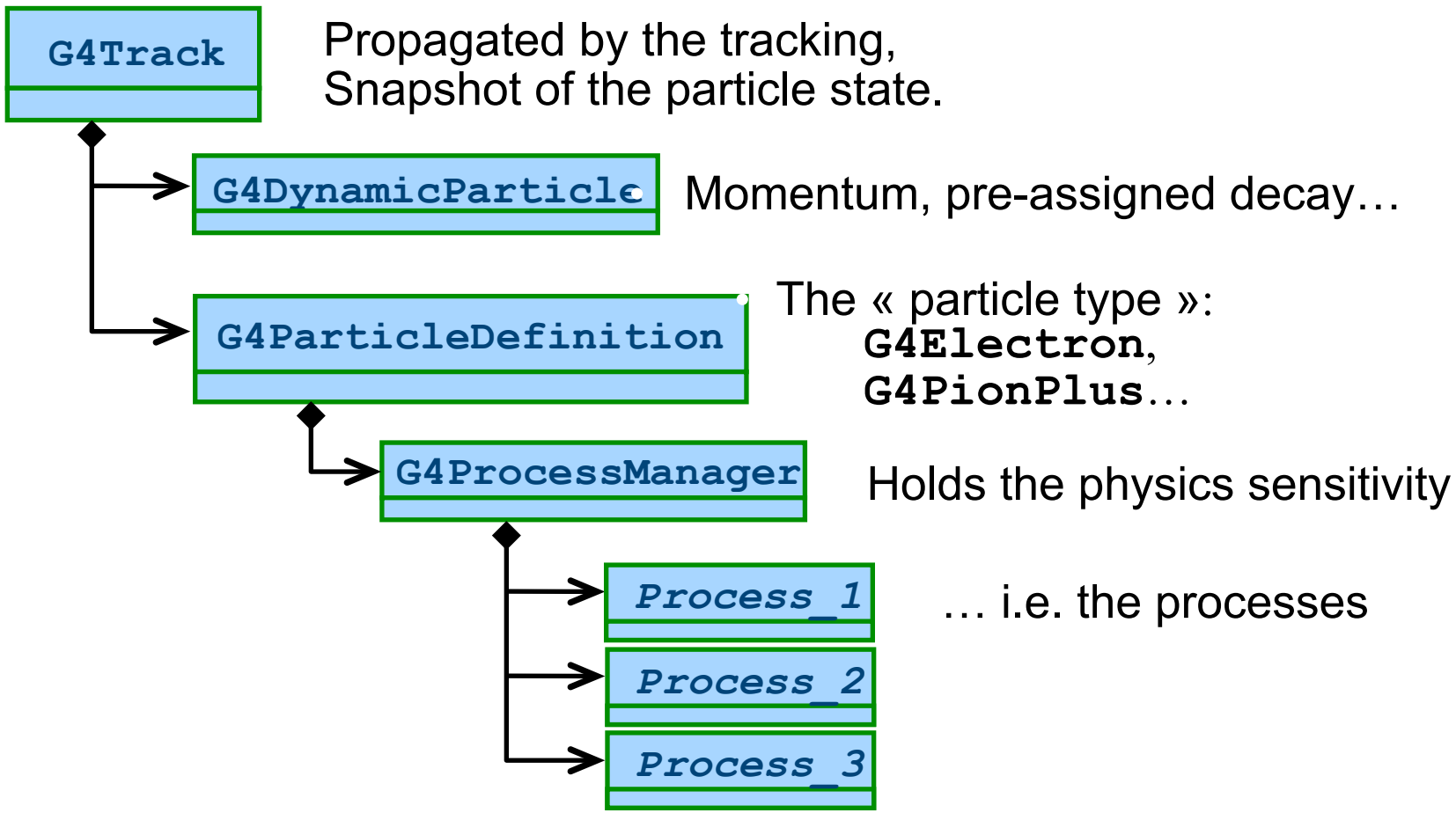


Propagated by the tracking,
Snapshot of the particle state.

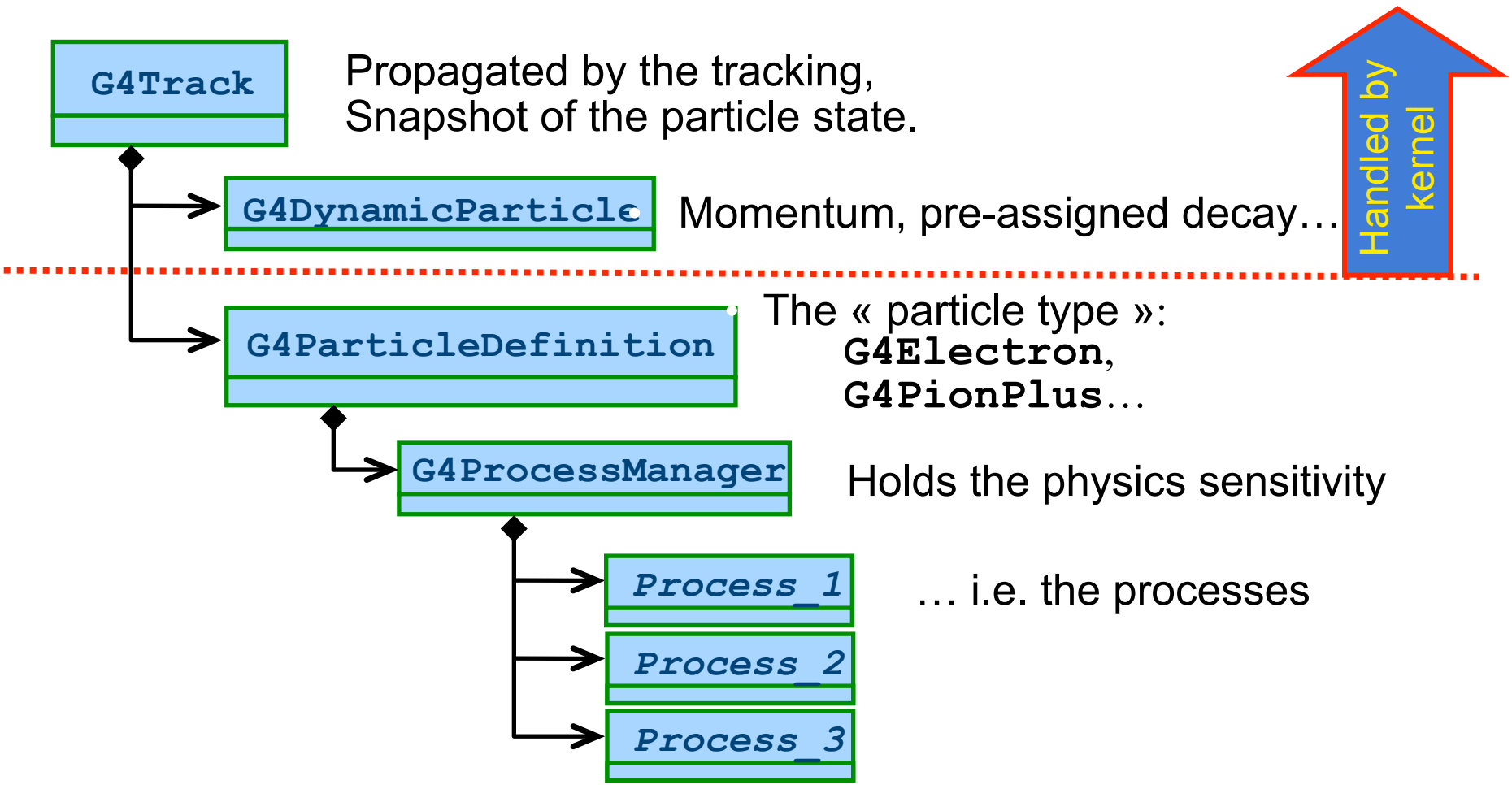
Momentum, pre-assigned decay…

The « particle type »:
`G4Electron`,
`G4PionPlus`…

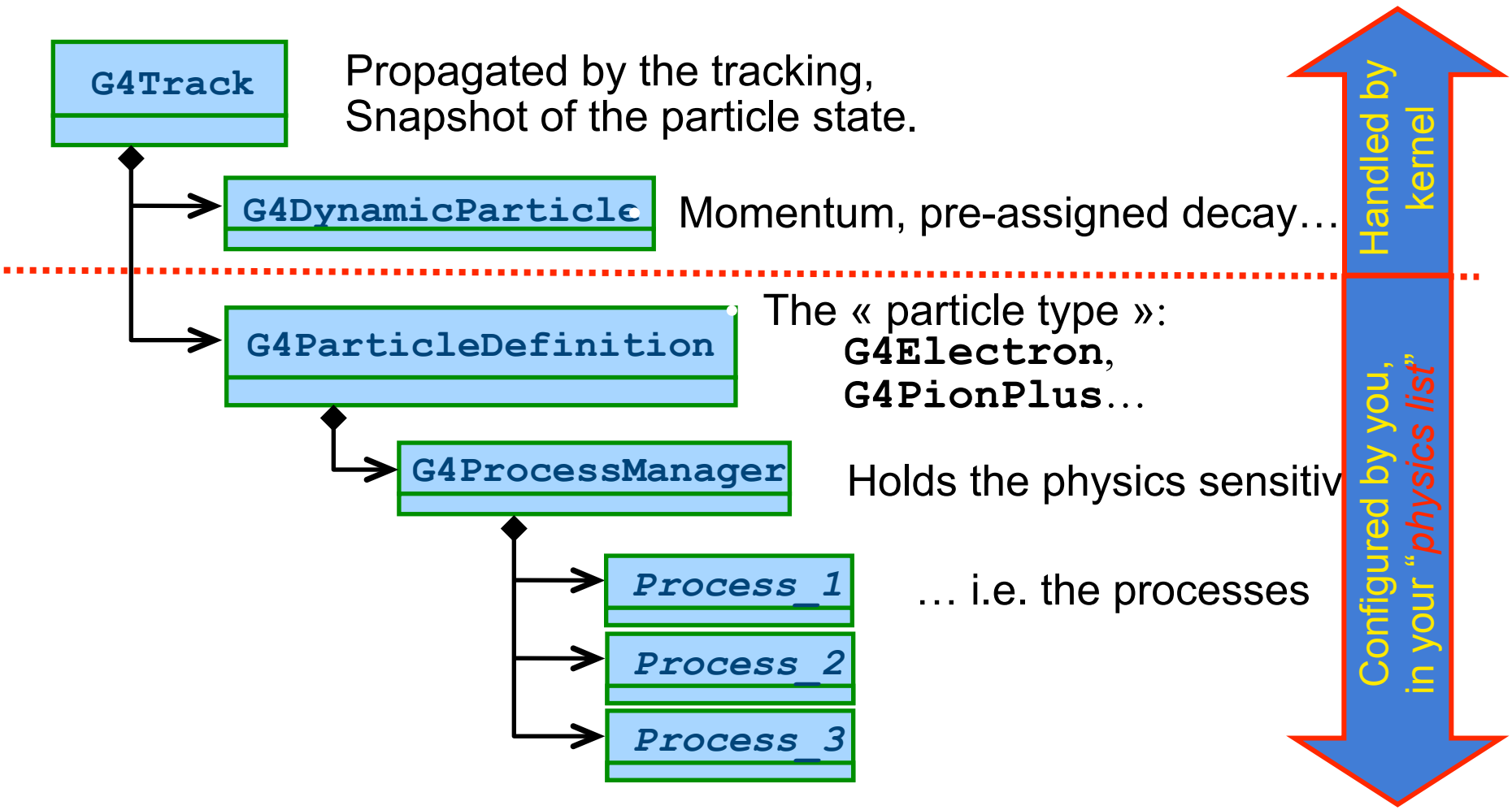
From **G4Track** to processes



From **G4Track** to processes



From **G4Track** to processes



Processes 3 kind of actions (1)

- Abstract class **`G4VProcess`** defining the common interface of all processes in Geant4:
  - also used by the transportation, etc…
- Defines three kinds of actions:

Processes 3 kind of actions (1)

- Abstract class **`G4VProcess`** defining the common interface of all processes in Geant4:
  - also used by the transportation, etc…
- Defines three kinds of actions:
  - **`AtRest`** actions:

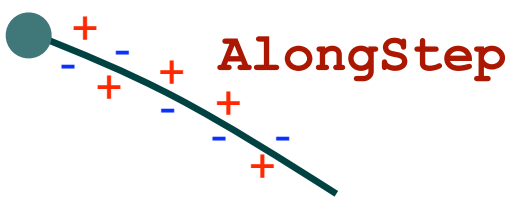
    Decay, e^+ annihilation …

Processes 3 kind of actions (1)

- Abstract class **`G4VProcess`** defining the common interface of all processes in Geant4:
  - also used by the transportation, etc…
- Defines three kinds of actions:
  - **`AtRest`** actions:

    Decay, e^+ annihilation …
  - **`AlongStep`** actions:

    To describe continuous (inter)actions, occuring along the path of the particle, like ionisation;



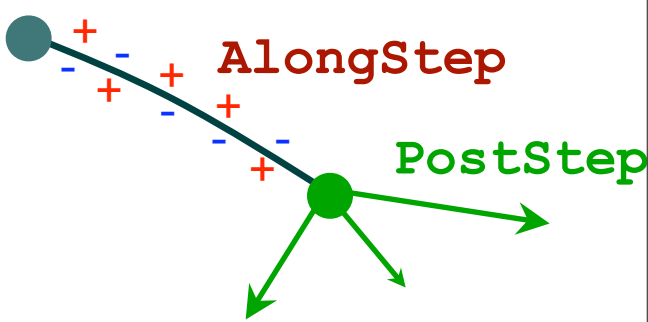
Processes 3 kind of actions (1)

- Abstract class **`G4VProcess`** defining the common interface of all processes in Geant4:
  - also used by the transportation, etc…
- Defines three kinds of actions:
  - **`AtRest`** actions:

    Decay, e^+ annihilation …
  - **`AlongStep`** actions:

    To describe continuous (inter)actions, occuring along the path of the particle, like ionisation;
  - **`PostStep`** actions:

    For describing point-like (inter)actions, like decay in flight, hard radiation…



Layout summary

I. The **`G4ParticleDefinition`** class

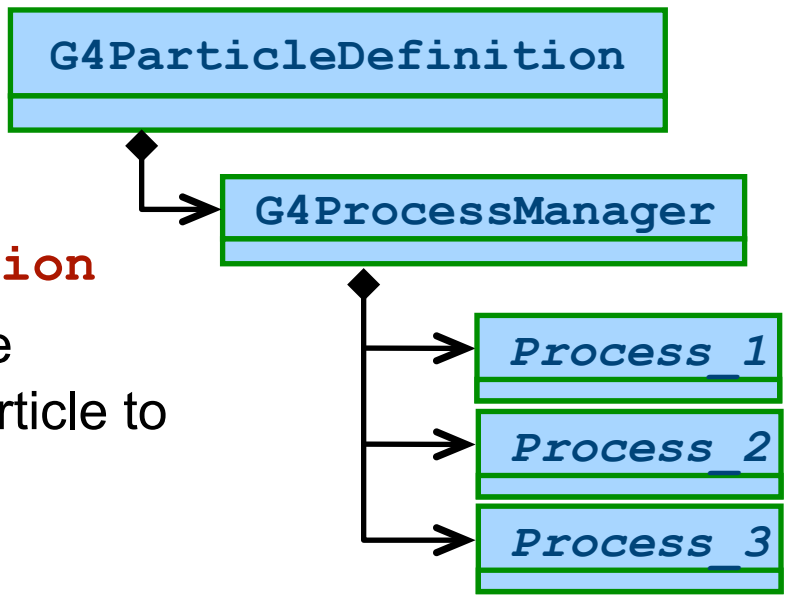
II. The **`G4ProcessManager`** class

III. The **`G4VProcess`** class process interface

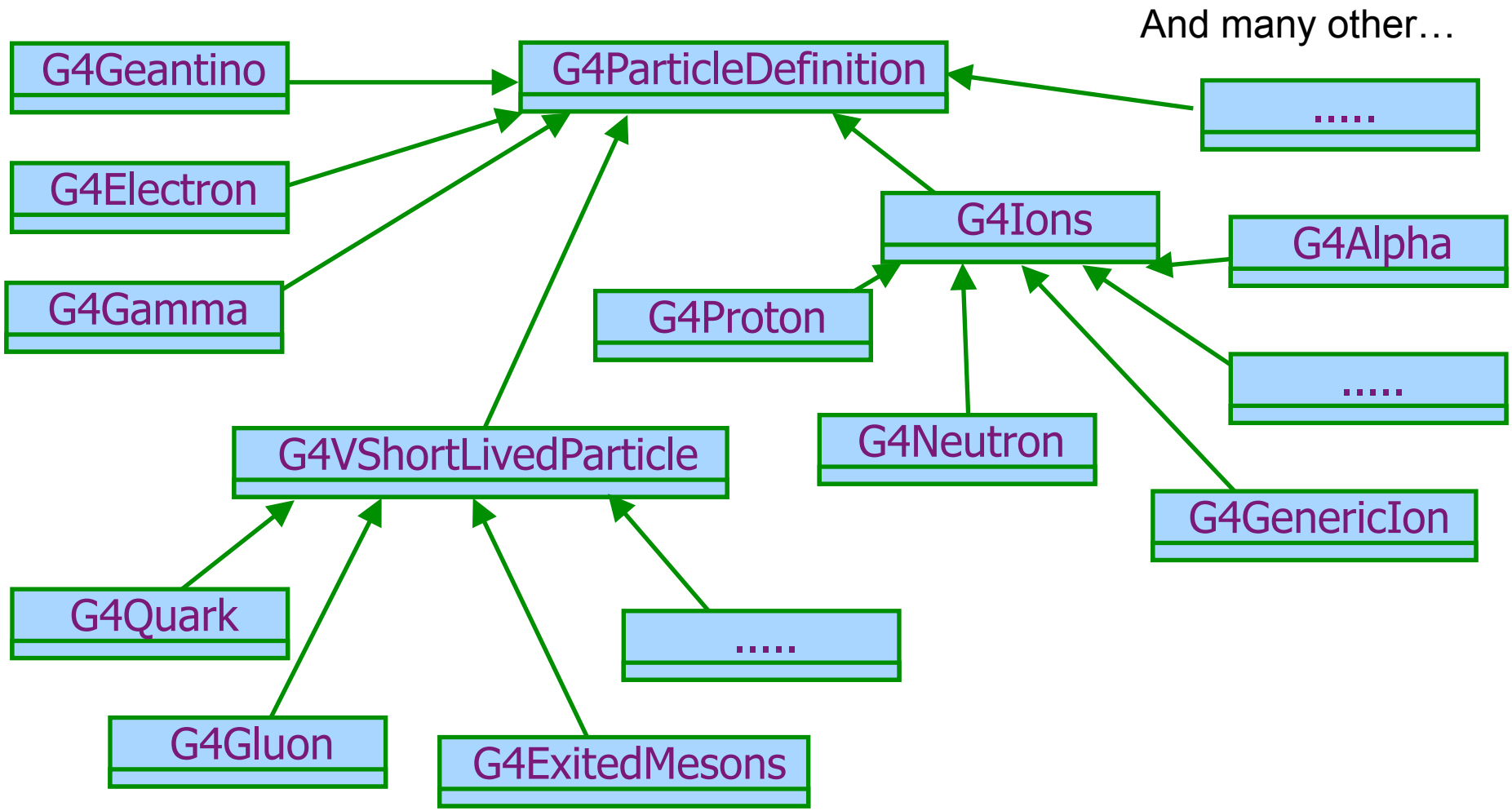
I. G4ParticleDefinition

G4ParticleDefinition

- The particle types in GEANT4 are described by the **G4ParticleDefinition** class;
- Describes the « intrisic » particle properties:
  - Mass, width, spin, lifetime…
- Describes its « sensitivity » to physics:
  - realized by a **G4ProcessManager**
  - attached to the **G4ParticleDefinition**
  - the **G4ProcessManager** manages the list of processes the user wants the particle to be sensitive to



Particle definition hierarchy



Adding Particles

- Most common particles, are implemented as static classes:
  - Like e^-, K^0_S, gamma, pions, but also α…
  - To allow –say– electrons in the simulation, the following call should be made in the « physics list »:
- Should be instantiated by the user's PhysicsList
- Example:

```
void PhysicsList::ConstructParticle()
{
  // In this method, static member functions should be called
  // for all particles which you want to use.

  G4Electron::ElectronDefinition();
  G4Positron::PositronDefinition();

  G4Gamma::Gamma();
}
```

Displaying processes of a particle

When you application has started and when the run manager has been initialized, you can:

- Check what particles exist:

  ```
  /particle/list
  ```

- Check a particle property:

  ```
  /particle/select e-
  /particle/property/dump
  ```

- Please type “help” to get the full set of commands

But also…

Check the physics processes attached and their ordering:

```
/particle/select e-
/particle/processes/dump
```

1st Hands-on

- Task3b
  - Exercise 4
    - Add muon to the list of processes
    - Use UI commands to learn about properties and processes of particles

II. G4ProcessManager

G4ProcessManager

G4ProcessManager

- `G4ProcessManager` maintains three vectors of actions :
  - One for the `AtRest` methods of the particle;
  - One for the `AlongStep` ones;
  - And one for the `PostStep` actions.

G4ProcessManager

- `G4ProcessManager` maintains three vectors of actions :
  - One for the `AtRest` methods of the particle;
  - One for the `AlongStep` ones;
  - And one for the `PostStep` actions.
- These vectors you have to set up in your "physics list"
  - they will be used by the tracking.

G4ProcessManager

- `G4ProcessManager` maintains three vectors of actions :
  - One for the `AtRest` methods of the particle;
  - One for the `AlongStep` ones;
  - And one for the `PostStep` actions.
- These vectors you have to set up in your "physics list"
  - they will be used by the tracking.
- The ordering of processes provided by/to the **`G4ProcessManager`** vectors is relevant and used by the stepping
  - There is **one** critical point you should be aware of…

About process ordering

- Process ordering is not critical…
- **… except for multiple-scattering and transportation.**
- In your physics list, you should **always** have, for the ordering of the `AlongGetPhysicalInteractionLength(…)` methods:
  - Transportation last
    - For all particles
  - Multiple scattering second last
    - For charged particles only, of course
    - of n processes, the ordering must be:

      [n-2] …

      [n-1] multiple scattering

      [n] transportation

About process ordering

- Process ordering is not critical…
- **… except for multiple-scattering and transportation.**
- In your physics list, you should **always** have, for the ordering of the `AlongGetPhysicalInteractionLength(…)` methods:
  - Transportation last
    - For all particles
  - Multiple scattering second last
    - For charged particles only, of course
    - of n processes, the ordering must be:

      [n-2] …

      [n-1] multiple scattering

      [n] transportation

Why ?

About process ordering

- Process ordering is not critical…
- **… except for multiple-scattering and transportation.**
- In your physics list, you should **always** have, for the ordering of the `AlongGetPhysicalInteractionLength(…)` methods:
  - Transportation last
    - For all particles
  - Multiple scattering second last
    - For charged particles only, of course
    - of n processes, the ordering must be:

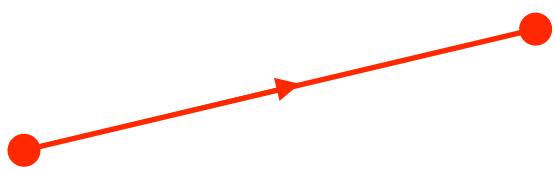
      [n-2] …

      [n-1] multiple scattering

      [n] transportation

Why ?

- Processes return a « true path length »;



About process ordering

- Process ordering is not critical…
- **… except for multiple-scattering and transportation.**
- In your physics list, you should **always** have, for the ordering of the `AlongGetPhysicalInteractionLength(…)` methods:
  - Transportation last
    - For all particles
  - Multiple scattering second last
    - For charged particles only, of course
    - of n processes, the ordering must be:

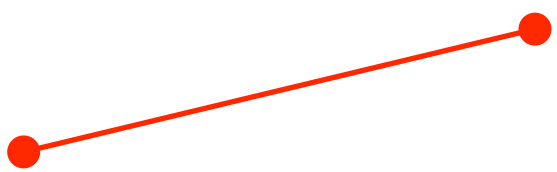
      [n-2] …

      [n-1] multiple scattering

      [n] transportation

Why ?

- Processes return a « true path length »;



About process ordering

- Process ordering is not critical…
- **… except for multiple-scattering and transportation.**
- In your physics list, you should **always** have, for the ordering of the `AlongGetPhysicalInteractionLength(…)` methods:
  - Transportation last
    - For all particles
  - Multiple scattering second last
    - For charged particles only, of course
    - of n processes, the ordering must be:

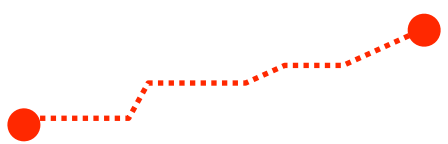
      [n-2] …

      [n-1] multiple scattering

      [n] transportation

Why ?

- Processes return a « true path length »;
- The multiple scattering « virtually folds up » this true path length into a ***shorter*** « geometrical » path length;



About process ordering

- Process ordering is not critical…
- **… except for multiple-scattering and transportation.**
- In your physics list, you should **always** have, for the ordering of the `AlongGetPhysicalInteractionLength(…)` methods:
  - Transportation last
    - For all particles
  - Multiple scattering second last
    - For charged particles only, of course
    - of n processes, the ordering must be:

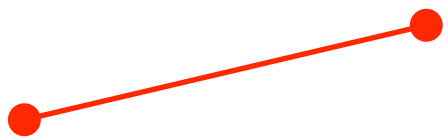
      [n-2] …

      [n-1] multiple scattering

      [n] transportation

Why ?

- Processes return a « true path length »;
- The multiple scattering « virtually folds up » this true path length into a ***shorter*** « geometrical » path length;



About process ordering

- Process ordering is not critical…
- **… except for multiple-scattering and transportation.**
- In your physics list, you should **always** have, for the ordering of the `AlongGetPhysicalInteractionLength(…)` methods:
  - Transportation last
    - For all particles
  - Multiple scattering second last
    - For charged particles only, of course
    - of n processes, the ordering must be:

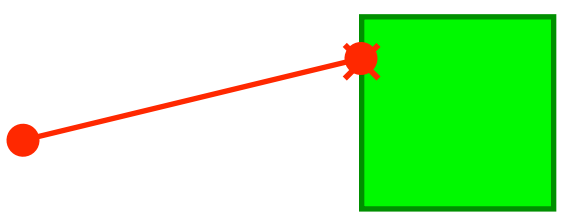
      [n-2] …

      [n-1] multiple scattering

      [n] transportation

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.



Adding a process in your physics list

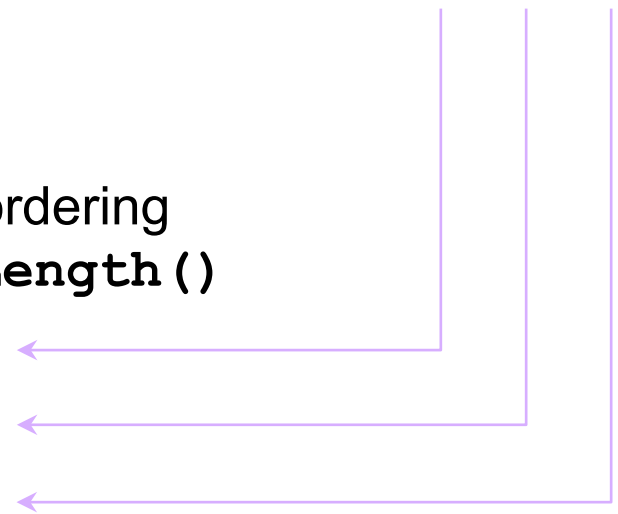
- Get the process manager of the particle:

```
G4ProcessManager* pmanager = particle->GetProcessManager();
```

- Add the process:

```
pmanager->AddProcess(new G4eIonisation,       -1, 2, 2);
```

  - The indices define the ordering of the `DoIt()` vectors
  - This is by default the **reverse** of the ordering of the `GetPhysicalInteractionLength()`
    - Index in `AtRestDoIt()` vector
    - Index in `AlongStepDoIt()` vector
    - Index in `PostStepDoIt()` vector



Adding a process in your physics list (2)

- Get the process manager of the particle:

```
G4ProcessManager* pmanager = particle->GetProcessManager();
```

- Add the process:

```
G4VProcess* process = new G4Decay();
pmanager->AddProcess(process);
pmanager->SetProcessOrdering(process, idxPostStep);
pmanager->SetProcessOrdering(process, idxAtRest);
```

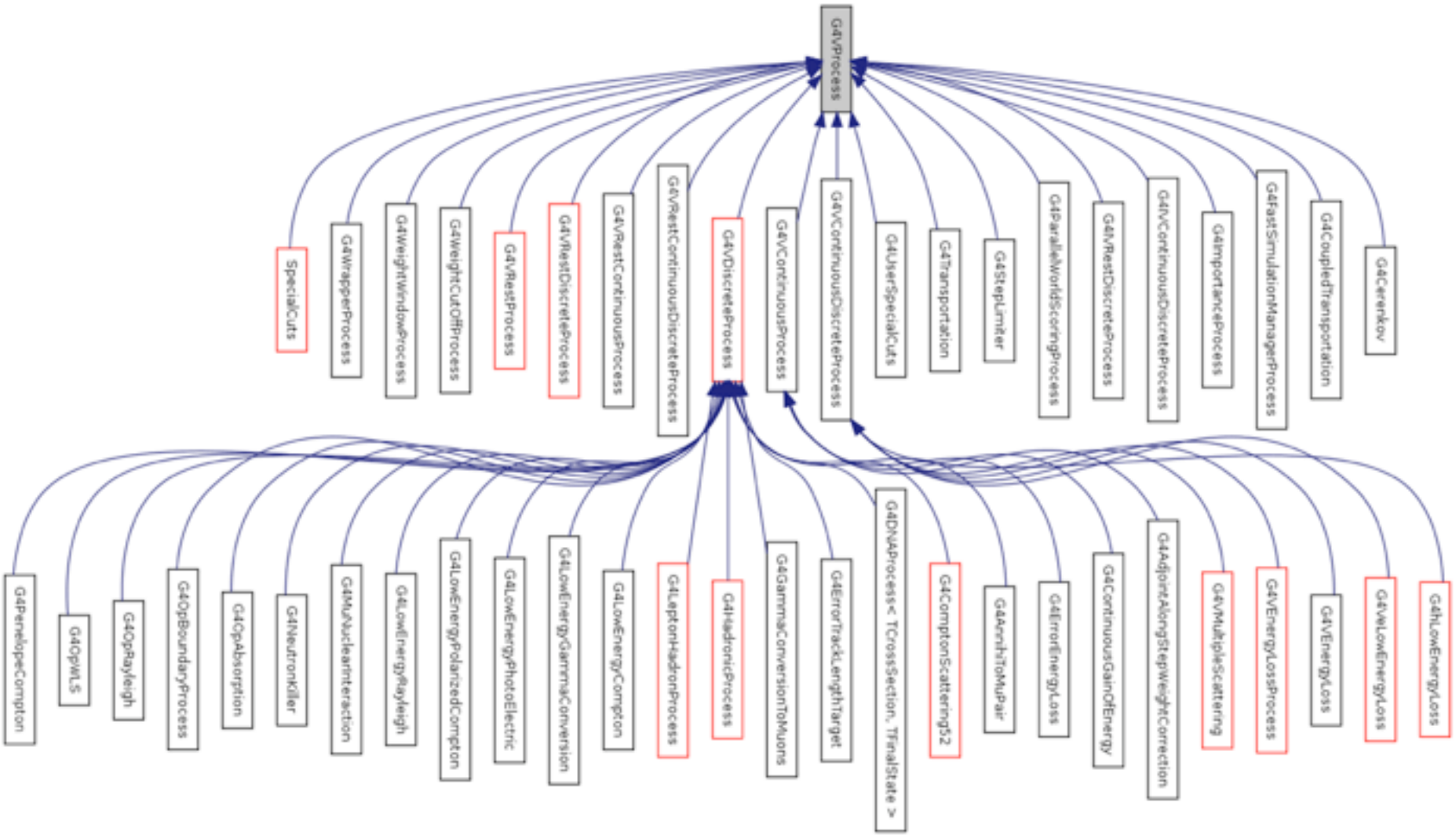
  - Convenient alternative if no explicit ordering is needed.
    - idxAtRest for **`AtRestDoIt()`** vector
    - idxAlongStep for **`AlongStepDoIt()`** vector
    - idxPostStep for **`PostStepDoIt()`** vecto

2nd Hands-on

- Task3b
  - Exercise 5
    - Add a Decay process to the muon
  - Exercise 6
    - Add electromagnetic interactions for the muon

III. The process interface

G4VProcess: derivatives



`G4VProcess`: action methods

Each action defines two methods:

- `GetPhysicalInteractionLength()`:

  Used to ***limit the step***:

  either because the process « triggers » an interaction, a decay

  or any other reasons, like fraction of energy loss, geometry boundary, user's limit …

- `DoIt()`:

  Implements the ***actual action*** to be applied on the track;

  And the related production of secondaries.

`G4VProcess` : actions summary

`G4VProcess` : actions summary

- The « action » methods are thus:
  - `AtRestGetPhysicalInteractionLength()`, `AtRestDoIt()`;
  - `AlongStepGetPhysicalInteractionLength()`, `AlongStepDoIt()`;
  - `PostStepGetPhysicalInteractionLength()`, `PostStepDoIt()`;
- `G4VProcess` defines also other methods:
  - `G4bool IsApplicable(const G4ParticleDefinition &)`;
    - Used to check if a process can handle the given particle type
      It is called by the kernel when you set up your physics list
  - And methods called at the beginning and end of tracking of a particle, etc…

Process implementation

- A process can implement any combination of the three **AtRest**, **AlongStep** and **PostStep** actions:
  - eg: decay = **AtRest** + **PostStep**

```
class G4Decay : public G4VRestDiscreteProcess
{
public:
  G4Decay(const G4String& processName ="Decay");

  G4VParticleChange *PostStepDoIt(const G4Track&, const G4Step&);
  G4VParticleChange* AtRestDoIt(const G4Track&, const G4Step&);
  G4bool IsApplicable(const G4ParticleDefinition&);
  G4double AtRestGetPhysicalInteractionLength(const G4Track&,
                                              G4ForceCondition*);

  G4double PostStepGetPhysicalInteractionLength(const G4Track&,
                                          G4double, G4ForceCondition*);

  //...
```

Conclusion/summary

Conclusion/summary

- The classes **`G4ParticleDefinition`** and **`G4ProcessManager`** make the particles acquiring physics sensitivity

Conclusion/summary

- The classes **`G4ParticleDefinition`** and **`G4ProcessManager`** make the particles acquiring physics sensitivity
- To make a particle type existing in the simulation, its "definition" has to be called in the physics list:

```
G4XXXX::XXXXDefinition();
```

Conclusion/summary

- The classes **`G4ParticleDefinition`** and **`G4ProcessManager`** make the particles acquiring physics sensitivity
- To make a particle type existing in the simulation, its "definition" has to be called in the physics list:

```
G4XXXX::XXXXDefinition();
```

- To attach the physics processes to this particle, you have to message the "**`AddProcess(…)`**" method of the **`G4ProcessManager`**

Conclusion/summary

- The classes **`G4ParticleDefinition`** and **`G4ProcessManager`** make the particles acquiring physics sensitivity
- To make a particle type existing in the simulation, its "definition" has to be called in the physics list:

```
G4XXXX::XXXXDefinition();
```

- To attach the physics processes to this particle, you have to message the "**`AddProcess(…)`**" method of the **`G4ProcessManager`**
- This physics configuration is your responsibility
  - Even if (see later this week) helper classes exist

3rd Hands-on

- Task3b
  - Exercise 7
    - make the muon decay spin dependent
    - change the properties of the muon (muon decay table)
  - Exercise 8
    - Add a magnetic field to verify that new properties are active